



**Universidade de Brasília**  
**Faculdade do Gama**

**Implementação de Modelo  
para Detecção de Falhas em Máquinas  
Rotativas com FPGA**

Natália Schulz Teixeira

**Universidade de Brasília**  
**Faculdade do Gama**

**Implementação de Modelo  
para Detecção de Falhas em Máquinas  
Rotativas com FPGA**

Natália Schulz Teixeira

Projeto Final de Curso submetido como requisi-  
to parcial para obtenção do grau de Enge-  
nheiro Eletrônico.

Orientador: Prof. Dr. Daniel Mauricio Muñoz Arboleda

Brasília  
2024

## FICHA CATALOGRÁFICA

Schulz Teixeira, Natália.

Implementação de Modelo para Detecção de Falhas em Máquinas Rotativas com FPGA / Natália Schulz Teixeira; orientador Daniel Mauricio Muñoz Arboleda. -- Brasília, 2024.

125 p.

Projeto Final de Curso (Engenharia Eletrônica) -- Universidade de Brasília, 2024.

1. Aprendizagem de máquina. 2. Falha. 3. FPGA. 4. Vibração. I. Arboleda, Daniel Mauricio Muñoz , orient. II. Título.

**Universidade de Brasília**  
**Faculdade do Gama**

**Implementação de Modelo para Detecção de Falhas em  
Máquinas Rotativas com FPGA**

Natália Schulz Teixeira

Projeto Final de Curso submetido como requi-  
sito parcial para obtenção do grau de Enge-  
nheiro Eletrônico.

Trabalho aprovado. Brasília, 16 de Julho de 2024:

---

**Prof. Dr. Daniel Mauricio Muñoz Arboleda,**  
**UnB/FGA**  
Orientador

---

**Prof. Dr. ,**  
**UnB/FGA**  
Henrique Gomes de Moura

---

**Prof. Dr.,**  
**UnB/FGA**  
Gilmar Silva Beserra

*Este trabalho é dedicado à minha família, meu noivo e meus amigos, que sempre forneceram todo o suporte necessário durante a minha trajetória acadêmica.*

# Agradecimentos

Gostaria de expressar minha profunda gratidão a todos que, de alguma forma, contribuíram para a realização deste trabalho. Em primeiro lugar, agradeço ao meu **Deus**, que é vivo e fiel, por ter me sustentado especialmente na última semana da faculdade, a mais desafiadora de todas. Ele esteve ao meu lado em cada momento, abençoando-me em tantas matérias e provendo forças quando eu mais precisei. Reconheço que foi por Sua infinita bondade e graça que pude chegar até aqui, não apenas para alcançar a bênção da minha formação, mas também a promessa da vida eterna. Sem Ele, nada disso seria possível.

Agradeço ao meu **pai**, que não apenas me deu um apoio moral incondicional, mas também me ajudou com questões técnicas sempre que precisei. Sua experiência e paciência foram fundamentais para resolver desafios que surgiram ao longo do caminho.

À minha **mãe**, agradeço pelo incentivo constante, pelas palavras de motivação e por sempre acreditar em mim, mesmo nos momentos mais desafiadores. Seu carinho e apoio foram pilares essenciais para que eu chegasse até aqui.

Ao meu **irmão**, Gustavo, agradeço pela ajuda essencial na elaboração das imagens, tradução de conteúdos e montagem de figuras. Sua criatividade, atenção aos detalhes e disponibilidade para colaborar foram indispensáveis para a qualidade visual e clareza deste trabalho.

Um agradecimento especial também ao meu **noivo**, Lucas, que dedicou tardes, noites e manhãs ao meu lado, ajudando-me incansavelmente com bugs nos programas, revisões de códigos e tantos outros desafios técnicos que surgiram ao longo do caminho. Seu apoio emocional, paciência e dedicação foram fundamentais para que eu pudesse superar os momentos mais difíceis e seguir em frente.

Por fim, gostaria de expressar minha profunda gratidão ao meu **orientador**, Prof. Dr. Daniel Muñoz, pelas inúmeras reuniões, muitas vezes extensas, em que dedicou horas ao meu lado para discutir ideias, corrigir rumos e aprimorar este trabalho. Sua disponibilidade para atender dúvidas, mesmo fora do horário de atendimento, e seu comprometimento com minha formação foram exemplares e fizeram toda a diferença.

*“Davi, porém, disse ao filisteu: “Você vem contra mim com espada, com lança e com dardos, mas eu vou contra você em nome do Senhor dos Exércitos, o Deus dos exércitos de Israel, a quem você desafiou.”*

*(1 Samuel 17:45-47)*

# Resumo

O presente trabalho propôs uma abordagem para a detecção e análise preditiva de falhas em máquinas elétricas rotativas. Por meio da utilização de técnicas de monitoramento de vibração, foram captados sinais indicativos de anomalias, os quais foram submetidos a um processamento de sinais. Posteriormente, implementou-se um modelo matemático, visando a construção de um sistema de aprendizado capaz de identificar padrões associados a tipos de falhas. Destaca-se que, para otimização do processamento, o referido modelo matemático foi embarcado em um *FPGA* (*Field Programmable Gate Array*), proporcionando uma execução mais ágil e eficiente. Este estudo representou um avanço significativo no campo da manutenção preditiva, oferecendo uma solução eficaz para a identificação precoce de falhas em máquinas elétricas rotativas, com impacto comprovado na redução de custos operacionais e nas tarefas de manutenção.

**Palavras-chave:** Aprendizagem de máquina. Falha. FPGA. Vibração.

# Abstract

This study proposed an approach for the detection and predictive analysis of failures in rotating electrical machines. Through the use of vibration monitoring techniques, signals indicative of anomalies were captured and subjected to signal processing. Subsequently, a mathematical model was implemented, aiming to build a learning system capable of identifying patterns associated with types of failures. It is noteworthy that, to optimize processing, the aforementioned mathematical model was embedded in an *FPGA (Field Programmable Gate Array)*, enabling faster and more efficient execution. This study represented a significant advancement in the field of predictive maintenance, offering an effective solution for the early identification of failures in rotating electrical machines, with proven impact on reducing operational costs and maintenance tasks.

**Keywords:** Machine learning. Failure. FPGA. Vibration.

# Lista de figuras

Figura 1.1	Robôs analisando a vibração de máquinas em uma indústria. . . . .	17
Figura 2.1	Grafico de uma senoide e o RMS . . . . .	21
Figura 2.2	Caixa de Engrenagens . . . . .	24
Figura 2.3	Desalinhamento de um rolamento no pinhão da caixa de engrenagens . . . . .	26
Figura 2.4	Característica do espectro de vibração do desalinhamento paralelo	27
Figura 2.5	Desalinhamento Angular . . . . .	27
Figura 2.6	Desalinhamento Paralelo . . . . .	27
Figura 2.7	FPGA Arquitetura . . . . .	33
Figura 2.8	Arquitetura de um <i>slice</i> em <i>FPGA</i> de serie 7. . . . .	34
Figura 2.9	Arquitetura simplificada de um <i>FPGA</i> com Matriz de Roteamento.	35
Figura 2.10	Esquema do <i>DSP SLICE</i> disponível nos <i>FPGAs</i> da série 7 da <i>Xilinx</i> .	35
Figura 2.11	Esquema da Zynq7000 . . . . .	36
Figura 2.12	Comunicação AXI4-Stream . . . . .	38
Figura 3.1	Caminho de Dados do Sinal com Falha . . . . .	46
Figura 3.2	Caminho de Dados do Sinal Sem Falhas . . . . .	46
Figura 3.3	Placa de Desenvolvimento PYNQ-Z2 . . . . .	47
Figura 3.4	Bancada didática para análise de vibração. . . . .	49
Figura 4.1	FFT do Sinal sem falha após o filtro Butterworth. . . . .	50
Figura 4.2	FFT do Sinal com falha após o filtro Butterworth. . . . .	51
Figura 4.3	Sinal após a Transformada de Hilbert, sendo o azul com falha e o sinal laranja sem falha. . . . .	51
Figura 4.4	Sinal sem falha após a Transformada de Hilbert e a FFT . . . . .	52
Figura 4.5	Sinal com falha após a Transformada de Hilbert e a FFT . . . . .	52
Figura 4.6	Arvore de Decisão gerada pelo Weka . . . . .	54
Figura 4.7	Matriz de Confusão . . . . .	55
Figura 4.8	Resultado da classificação de 2 valores de Dados Falho e Saudável	56
Figura 4.9	Acelerômetro posicionado no mancal interno. . . . .	57
Figura 4.10	Sinal de Aceleração dos dados reais em domínio do tempo com filtro de butterworth. . . . .	57
Figura 4.11	Sinal de Aceleração dos dados reais em domínio do tempo após uma Transformada de Hilbert. . . . .	58
Figura 4.12	Sinal no Domínio da Frequênciia do Modelo F após a Filtragem das Frequências Relevantes. Sendo o Sinal Vermelho o Sinal com Falha e o Azul o Sinal Sem Falha . . . . .	58

Figura 4.13 Sinal no Domínio da Frequência do Modelo FHF após a Filtragem das Frequências Relevantes. Sendo o Sinal Vermelho o Sinal com Falha e o Azul o Sinal Sem Falha . . . . .	59
Figura 4.14 Tabela de parâmetros de Classificação . . . . .	61
Figura 4.15 Ruido Gaussiano. . . . .	62
Figura 4.16 Base de Dados Gerada para o Modelo F. Sendo o Falho o Sinal Verde e o Sem falha o Sinal Azul . . . . .	62
Figura 4.17 Base de Dados Gerada para o Modelo FHF. Sendo o Falho o Sinal Verde e o Sem falha o Sinal Azul . . . . .	62
Figura 4.18 Configuração do IP - FFT . . . . .	65
Figura 4.19 Configuração da implementação do IP - FFT . . . . .	66
Figura 4.20 Canal de Configuração . . . . .	66
Figura 4.21 Maquina de Estados para envio de dados. . . . .	68
Figura 4.22 Top Module do sistema. . . . .	69
Figura 4.23 Teste do IP - FFT com a PYNQ Z2. . . . .	71
Figura 4.24 Teste do IP - FFT com a PYNQ Z2 com zoom. . . . .	71
Figura 4.25 Block Design do sistema. . . . .	72
Figura 4.26 Espectro esperado na saída do IP - FFT . . . . .	73
Figura 4.27 Netlist e Consumo de Recursos . . . . .	73
Figura 4.28 Utilização de Recursos Pós-Implementação . . . . .	74
Figura 4.29 Consumo de energia . . . . .	74

# **Lista de tabelas**

Tabela 2.1 Resumo dos Métodos para Diagnóstico de Falhas e Multi-Falhas em Máquinas Rotativas. . . . .	40
Tabela 4.1 Configurações para o Modelo J48 no Weka . . . . .	53
Tabela 4.2 Accuracy using the obtained decision tree for the <i>FHF</i> preproces- sing model. . . . .	53

# Sumário

<b>1</b>	<b>Introdução . . . . .</b>	<b>15</b>
1.1	Introdução . . . . .	15
1.2	Perguntas de Pesquisa . . . . .	16
1.3	Justificativa . . . . .	18
1.4	Objetivos . . . . .	18
1.4.1	Objetivo Geral . . . . .	18
1.4.2	Objetivos Específicos . . . . .	18
1.5	Contribuições do Trabalho . . . . .	19
1.6	Organização do Trabalho . . . . .	19
<b>2</b>	<b>Fundamentação Teórica . . . . .</b>	<b>20</b>
2.1	Analise de Vibração . . . . .	20
2.1.1	Vibração . . . . .	20
2.1.2	Medição de Vibração . . . . .	21
2.2	Falhas em Máquinas Rotativas . . . . .	23
2.2.1	Caixa de Engrenagens . . . . .	23
2.2.2	Falha de Desbalanceamento . . . . .	25
2.2.3	Falha de Desalinhamento . . . . .	25
2.3	Aprendizagem de Máquina . . . . .	28
2.3.1	Arvores de Decisão . . . . .	28
2.3.2	Weka: Ferramenta para Aprendizado de Máquina . . . . .	28
2.4	Processamento de sinais . . . . .	29
2.4.1	Filtro Butterworth . . . . .	29
2.4.2	FFT . . . . .	29
2.4.3	Transformada de Hilbert . . . . .	30
2.4.4	Parâmetros de Avaliação . . . . .	31
2.5	Hardware Reconfigurável . . . . .	32
2.5.1	SoC . . . . .	32
2.5.2	SoC- FPGAs . . . . .	32
2.5.3	IP - FFT . . . . .	36
2.5.4	AXI4-Stream . . . . .	38
2.6	Máquinas de Estados . . . . .	39
2.6.1	Máquina de Moore . . . . .	39
2.6.2	Máquina de Mealy . . . . .	39
2.7	Estado da Arte . . . . .	39
<b>3</b>	<b>Metodologia . . . . .</b>	<b>43</b>

3.1	Simulação de Falhas em Máquinas Rotativas . . . . .	43
3.2	Criação de uma Base de Dados . . . . .	44
3.3	Proposta de Processamento de Sinais de Vibração de Máquinas Rotativas	44
3.3.1	Modelo F . . . . .	44
3.3.2	Modelo FHF . . . . .	45
3.4	Modelo de Classificação de Falhas . . . . .	45
3.5	Implementação em <i>SoC FPGA</i> do Modelo de Predição . . . . .	46
3.6	Materiais e Ferramentas . . . . .	47
3.7	Planejamento Experimental . . . . .	48
<b>4</b>	<b>Resultados</b> . . . . .	<b>50</b>
4.1	Resultados da cadeia de processamento de sinais . . . . .	50
4.1.1	Modelo F . . . . .	50
4.1.2	Modelo FHF . . . . .	51
4.2	Resultados do modelo de classificação . . . . .	52
4.2.1	Implementação do Modelo J48 na Esp32 . . . . .	55
4.3	Aquisição de dados reais . . . . .	56
4.3.1	Manipulação de dados . . . . .	56
4.3.2	Testes com Arvore de decisão . . . . .	59
4.3.3	Classificação pelo RMS . . . . .	60
4.3.4	Criação de novas base de dados . . . . .	61
4.3.5	Configuração do IP-FFT . . . . .	63
4.3.6	Implementação na Pynq . . . . .	67
4.3.7	top_module . . . . .	68
<b>5</b>	<b>Conclusões</b> . . . . .	<b>75</b>
5.1	Cronograma de Atividades . . . . .	76
<b>Referências</b> . . . . .	<b>77</b>	
<b>Apêndices</b>	<b>83</b>	
<b>Apêndice A Códigos de programação</b> . . . . .	<b>84</b>	
A.1	Gerador de Sinal de Caixa de Engrenagens . . . . .	84
A.2	Exemplo de Teste Simulação Caixa de Engrenagens . . . . .	87
A.3	Teste Embarcando Modelo na Esp32 . . . . .	90
A.4	Código VHDL do <i>top module</i> . . . . .	94
A.5	Código Verilog da memória . . . . .	99
A.6	Código VHDL para detecção de falhas . . . . .	102
<b>Apêndice B Códigos de programação</b> . . . . .	<b>105</b>	
B.1	Gerador de Sinal de Caixa de Engrenagens . . . . .	105
B.2	Exemplo de Teste Simulação Caixa de Engrenagens . . . . .	108

B.3	Teste Embarcando Modelo na Esp32 . . . . .	111
B.4	Código VHDL do <i>top module</i> . . . . .	115
B.5	Código Verilog da memória . . . . .	120
B.6	Código VHDL para detecção de falhas . . . . .	123

# 1 Introdução

## 1.1 Introdução

A análise de vibrações desempenhou um papel fundamental na manutenção eficaz de equipamentos industriais, servindo como uma ferramenta essencial para a redução de custos operacionais e a prevenção de falhas. Em ambientes industriais, máquinas rotativas estiveram sujeitas a diversos tipos de falhas, como desbalanceamento, desalinhamento e folgas mecânicas, que puderam comprometer sua operação e levar a paradas não planejadas. Dentre essas, o desbalanceamento e o desalinhamento foram duas das falhas mais comuns e críticas (Jesus; Cavalcante, 2011). Ambas as condições geraram vibrações anormais que, se não detectadas precocemente, puderam causar danos severos aos componentes, como desgaste acelerado de rolamentos, engrenagens e acoplamentos, além de aumentar o consumo de energia e o risco de falhas catastróficas. A detecção e correção oportunas dessas falhas foram, portanto, essenciais para garantir a confiabilidade e a eficiência operacional dos equipamentos (Marçal, 2000) (Teixeira *et al.*, 2024).

Para enfrentar esse desafio, técnicas de processamento de sinais, como a Transformada Rápida de Fourier (FFT - do inglês *Fast Fourier Transform*) (Adamsab; Shivakumar, 2018), a Transformada de Hilbert (McInerny SA e Dai, 2003) e a Transformada Wavelet (Tao *et al.*, 2020), foram amplamente utilizadas. Esses métodos permitiram a identificação de padrões de vibração característicos associados a diferentes estados operacionais, facilitando a detecção precoce de falhas. Por exemplo, a FFT foi capaz de decompor sinais de vibração em suas componentes de frequência, revelando anomalias que indicavam desbalanceamento, desalinhamento ou outras falhas. A Transformada de Hilbert, por sua vez, foi útil para analisar o envelope do sinal, enquanto a Wavelet permitiu uma análise multirresolução, ideal para detectar falhas em sinais não estacionários.

A integração de algoritmos de *Machine Learning* (ML) em sistemas embarcados (Verma *et al.*, 2021) representou um avanço significativo na detecção automatizada de falhas. Esses algoritmos puderam ser treinados para identificar padrões complexos em sinais de vibração, incluindo aqueles associados a desbalanceamento e desalinhamento, tornando a manutenção preditiva mais eficiente e precisa. No entanto, a implementação desses algoritmos em hardware embarcado, como FPGAs (*Field-Programmable Gate Arrays*), apresentou desafios técnicos, como a necessidade de otimização de recursos computacionais e o processamento em tempo real de grandes volumes de dados.

Além disso, a aplicação da análise de vibração em robôs móveis ganhou relevância, especialmente em sistemas de robótica multiagente. Robôs equipados com sensores de vibração e técnicas de *beamforming* acústico puderam navegar autonomamente por ambientes industriais, localizando fontes de ruído e vibração com precisão (Argentieri; Danes; Soueres, 2006). Essa abordagem combinou a análise de vibração com a localização acústica, permitindo a detecção precoce de falhas, como desbalanceamento e desalinhamento, e a monitorização contínua de equipamentos. A sinergia entre essas tecnologias possibilitou a criação de sistemas de manutenção preditiva mais robustos e autônomos.

Nesse contexto, a implementação de métodos de processamento de sinais e algoritmos de ML em Sistemas em Chip (SoC, do inglês *System-on-Chip*) baseados em FPGAs destacou-se pela sua eficiência computacional e capacidade de processamento em tempo real. FPGAs permitiram a execução paralela de operações complexas, como a FFT e a Transformada de Hilbert, com baixa latência e consumo de energia reduzido. Essa capacidade foi crucial para aplicações industriais, onde a detecção rápida e precisa de falhas pode evitar paradas dispendiosas e prolongar a vida útil dos equipamentos (Espindola, 2011).

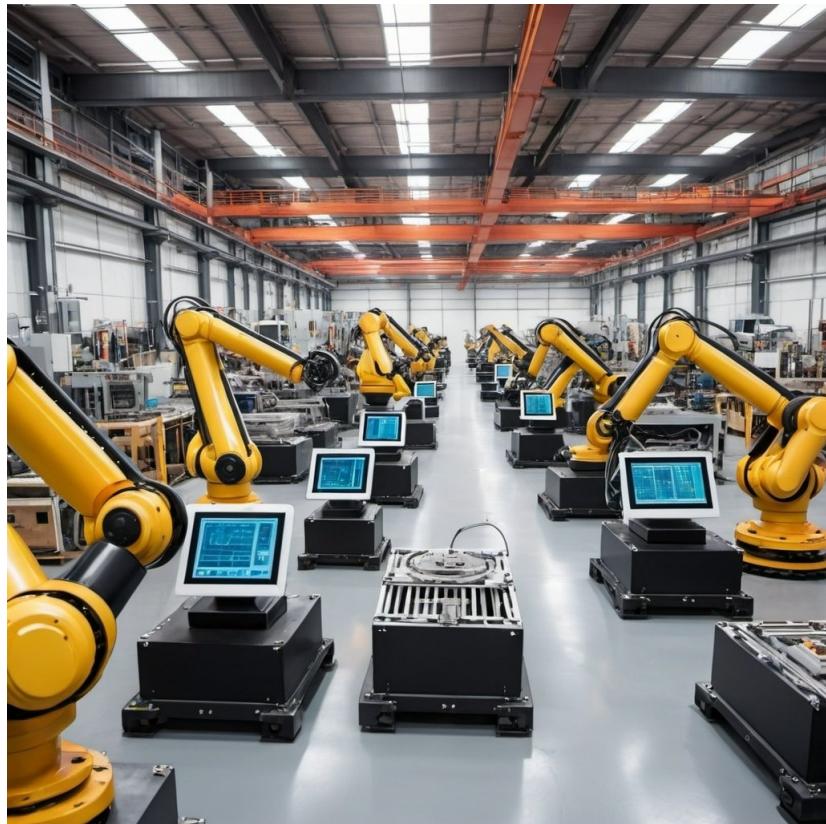
Este trabalho propôs uma abordagem integrada para a detecção de falhas em máquinas rotativas, combinando técnicas de processamento de sinais, como a FFT e a Transformada de Hilbert, com algoritmos de ML embarcados em FPGAs e outras formas de classificação. Além disso, explorou a aplicação dessas técnicas em robôs móveis para inspeção autônoma em chão de fábrica. A implementação eficiente desses métodos em hardware embarcado visou não apenas melhorar a precisão da detecção de falhas, como desbalanceamento e desalinhamento, mas também reduzir custos operacionais e otimizar a manutenção preditiva, contribuindo para a confiabilidade e a eficiência de sistemas industriais.

## 1.2 Perguntas de Pesquisa

As perguntas de pesquisa que norteiam o presente trabalho são fundamentais para compreender e otimizar a aplicação de tecnologias avançadas na inspeção de máquinas elétricas rotativas utilizando robôs móveis. Neste contexto, três perguntas-chave foram formuladas:

- Qual é o método de pré-processamento de dados mais eficaz para evidenciar falhas de vibração em máquinas elétricas rotativas, dentre os estudados neste trabalho?
- Métodos de ML baseados em árvores de decisão são adequados para uso em robôs móveis de inspeção de máquinas rotativas?

Figura 1.1 – Robôs analisando a vibração de máquinas em uma indústria.



Fonte: Criada por <https://openart.ai/>

- O uso de *SoC FPGA* é a melhor abordagem para esta aplicação?

Estas perguntas são essenciais para direcionar a pesquisa e o desenvolvimento de soluções inovadoras na área de manutenção preditiva. A primeira pergunta busca identificar as técnicas de pré-processamento de dados que melhor destacam os sinais de falhas, garantindo uma análise mais precisa e confiável. Além disso, explora como diferentes métodos de filtragem, normalização e extração de características podem ser combinados para otimizar a detecção de anomalias em diversos cenários operacionais. A segunda pergunta investiga a adequação dos métodos de ML baseados em árvores de decisão para serem implementados em robôs móveis, avaliando sua eficiência e eficácia na detecção de falhas. A terceira pergunta explora os benefícios e limitações do uso de *SoC FPGA* para a aplicação proposta, considerando aspectos como desempenho, custo e flexibilidade. Adicionalmente, analisa como a escolha de diferentes arquiteturas de hardware embarcado pode impactar a escalabilidade e a adaptabilidade do sistema em diferentes contextos industriais.

## 1.3 Justificativa

A implementação de um algoritmo de detecção de falhas em máquinas rotativas utilizando uma solução embarcada em *SoC FPGA* apresenta uma relevância significativa no contexto atual da indústria. Há uma crescente demanda por sistemas de monitoramento não intrusivo e diagnóstico de máquinas industriais, visando aumentar a eficiência operacional, reduzir custos de manutenção e prevenir falhas que podem resultar em paradas não programadas e perda de produção ([MOREIRA, 2020](#)).

A escolha de uma solução embarcada em *SoC FPGA* ocorre devido à alta flexibilidade e capacidade de reconfiguração desses dispositivos, que são aptos a implementar algoritmos complexos de processamento de sinais ([Duarte, 2024](#)). Além disso, a implementação embarcada em *FPGA* proporciona baixa latência e alta velocidade de processamento.

De mesmo modo há também os benefícios técnicos, o trabalho de TCC focado na implementação de um algoritmo de detecção de falhas em máquinas de rolamento utilizando uma solução embarcada em *FPGA* contribuirá para o avanço do conhecimento científico e tecnológico na área de manutenção preditiva e monitoramento de máquinas industriais. Os resultados obtidos podem ser aplicados em diversos setores industriais, proporcionando impactos positivos tanto em termos de eficiência operacional quanto de segurança dos trabalhadores e do ambiente de trabalho.

## 1.4 Objetivos

### 1.4.1 Objetivo Geral

Desenvolver e implementar um algoritmo de detecção de falhas em máquinas rotativas utilizando uma solução embarcada em *SoC FPGA* eficiente para monitoramento e diagnóstico.

### 1.4.2 Objetivos Específicos

- Desenvolver um modelo para pré-processamento de sinais de vibração, visando filtrar ruídos e extrair características relevantes dos dados coletados, como amplitude e frequência a fim de preparar os sinais para análise subsequente.
- Construir um modelo baseado em inteligência artificial para classificação de falhas em caixas de engrenagens, utilizando técnicas de aprendizado de

---

máquina para identificar e categorizar diferentes tipos de falhas com base nos padrões de vibração detectados.

- Realizar a implementação em um *SoC FPGA* do modelo de inteligência artificial desenvolvido, integrando-o com o *hardware* embarcado para possibilitar a classificação de falhas diretamente no dispositivo de monitoramento, garantindo eficiência e baixa latência na detecção de problemas nas máquinas industriais.

## 1.5 Contribuições do Trabalho

A contribuição deste trabalho reside em um estudo para o desenvolvimento de um sistema integrado para identificar falhas maquinás rotativas em um *SoC*. O que resulta em melhorias na eficiência operacional, redução de custos de manutenção e prevenção de paradas não programadas na indústria.

## 1.6 Organização do Trabalho

No [Capítulo 2](#), é apresentada a fundamentação teórica, abordando os conceitos essenciais relacionados ao monitoramento de caixas de engrenagens, técnicas de análise de vibração, algoritmos de inteligência artificial aplicados à detecção de falhas.

O [Capítulo 3](#) descreve a metodologia proposta para o desenvolvimento do sistema de detecção de falhas em caixas de engrenagens, incluindo o processo de pré-processamento de sinais de vibração, a construção do modelo de inteligência artificial para classificação de falhas e a implementação em *SoC FPGA*.

No [Capítulo 4](#), são apresentados os resultados obtidos a partir da aplicação da metodologia proposta, incluindo análises quantitativas e qualitativas da eficácia do sistema de detecção de falhas em diferentes cenários de teste.

Por fim, o [Capítulo 5](#) apresenta as conclusões do trabalho, destacando as contribuições alcançadas, as limitações identificadas, sugestões para trabalhos futuros e a importância do sistema desenvolvido para a área de manutenção preditiva e monitoramento de máquinas industriais.

## 2 Fundamentação Teórica

Para iniciar o desenvolvimento do projeto, é necessário realizar uma análise da fundamentação teórica relacionada à vibração, análise de vibração, caixas de engrenagens e os diversos tipos de erros comumente encontrados nesses componentes. Será essencial compreender os princípios físicos por trás das vibrações em máquinas industriais, assim como os métodos e técnicas utilizados para sua análise e interpretação. Além disso, serão abordados modelos matemáticos, tais como árvores de decisão que pode ser empregado na classificação de falhas em sistemas de rolamentos com base nos padrões de vibração detectados. Por fim, será discutido o papel dos *hardwares* reconfiguráveis, com ênfase em sistemas embarcados em *FPGA*, explorando suas capacidades e vantagens para a implementação eficiente de algoritmos de processamento de sinais. Essa análise teórica será o alicerce necessário para o desenvolvimento de um sistema robusto e eficaz de detecção de falhas em máquinas de rolamento.

### 2.1 Analise de Vibração

#### 2.1.1 Vibração

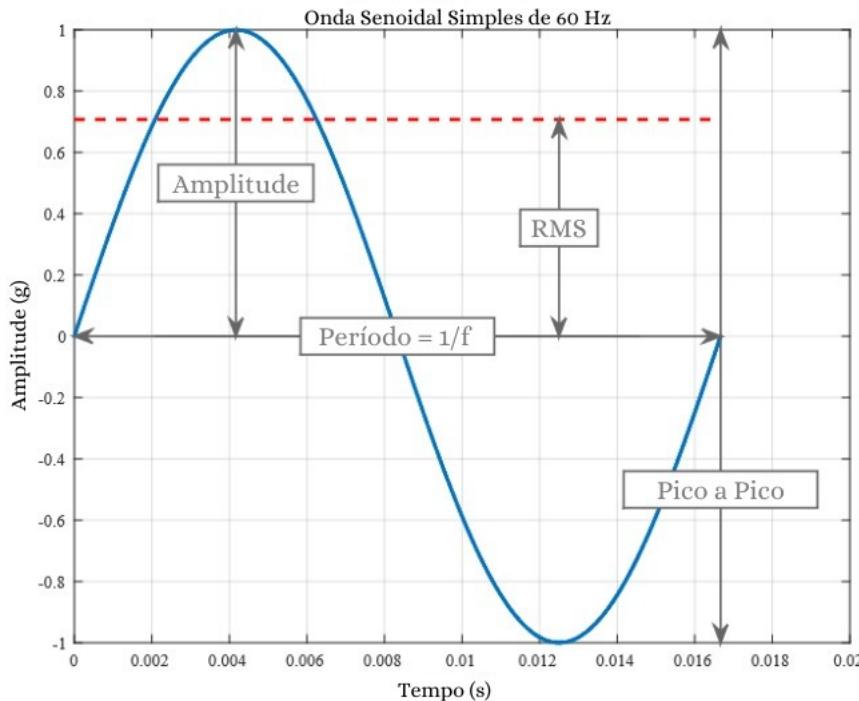
A vibração é um fenômeno físico que envolve o movimento repetitivo de um objeto ao redor de uma posição de equilíbrio (Menezes, 2015). Ela pode ser encontrada em diversos contextos na natureza, desde o movimento das moléculas em um sólido até os padrões de ondas sonoras e a oscilação de sistemas mecânicos. Na geologia, vibrações sísmicas resultam de terremotos e outras atividades tectônicas (Bacci, 2016). Além disso, a vibração é amplamente utilizada na tecnologia, desde aplicações em dispositivos de comunicação e sensores até na indústria, onde é crucial para o funcionamento de máquinas e equipamentos (Admiral, 2021).

A vibração de máquinas e equipamentos é um importante parâmetro utilizado para avaliar o seu estado de funcionamento. Uma das medidas amplamente empregadas para quantificar a intensidade da vibração é o valor **RMS** (Root Mean Square, ou Raiz Média Quadrática), onde podemos observar na Fig 2.1 (Igba *et al.*, 2016). O valor RMS oferece uma indicação da energia presente no sinal vibratório, levando em consideração tanto a amplitude quanto a frequência dos componentes que o compõem. Este valor é calculado pela fórmula:

$$RMS = \sqrt{\frac{1}{T} \int_0^T x(t)^2 dt}$$

onde  $x(t)$  representa o sinal vibratório em função do tempo, e  $T$  é o período de análise.

Figura 2.1 – Grafico de uma senoide e o RMS



Fonte: <https://blog.endaq.com/vibration-measurements-vibration-analysis-basics>

### 2.1.2 Medição de Vibração

Na medição de vibração com acelerômetros *MEMS* (Sistemas Microeletromecânicos), o princípio básico é semelhante ao dos acelerômetros convencionais, mas com uma escala muito menor e utilizando a tecnologia de microfabricação. Os acelerômetros *MEMS* são dispositivos miniaturizados que empregam estruturas microscópicas, como massas suspensas ou elementos piezoelétricos, para detectar variações na aceleração. Esses acelerômetros podem medir a aceleração em três direções, permitindo a coleta de informações tridimensionais sobre o movimento vibratório (Souza; Machado, 2016).

Nesses dispositivos, a aceleração faz com que pequenas estruturas mecânicas se deformem, gerando um sinal elétrico proporcional à magnitude da aceleração. Esse sinal elétrico é processado por circuitos integrados embutidos no próprio chip do acelerômetro (Dadafshar, 2015), (Son *et al.*, 2016). A principal vantagem dos

acelerômetros MEMS é a miniaturização, que permite o uso em diversas aplicações, incluindo monitoramento de vibrações em ambientes restritos ou em equipamentos de baixo custo.

Na medição de vibração, os acelerômetros MEMS são fixados na superfície do equipamento ou estrutura a ser monitorada, assim como os acelerômetros convencionais. Quando ocorre uma vibração, a deformação das estruturas microscópicas é detectada pelo acelerômetro MEMS, gerando um sinal elétrico que pode ser analisado para fornecer informações detalhadas sobre as características da vibração, como a amplitude, frequência e forma de onda (Souza; Machado, 2016).

A análise de vibrações pode ser realizada em diferentes domínios:

- **Domínio Temporal:** No domínio temporal, o sinal de vibração é analisado em função do tempo. Esse tipo de análise fornece informações sobre a evolução da amplitude da vibração ao longo do tempo. Em um gráfico temporal, é possível observar variações instantâneas nas vibrações e identificar padrões, como picos repentinos ou oscilações contínuas. Este método é útil para identificar anomalias no comportamento dinâmico do sistema, como impactos ou variações abruptas (Wagner, 2017).
- **Domínio Frequencial:** No domínio frequencial, o sinal vibratório é decomposto em suas componentes de frequência por meio da transformada de Fourier. Isso permite identificar as frequências dominantes no sinal de vibração e correlacioná-las com possíveis falhas mecânicas, como desbalanceamento, desalinhamento ou falhas nos rolamentos. A análise espectral é extremamente útil para diagnosticar problemas que ocorrem em frequências específicas (Passos, 2016).

Além disso, de acordo com Araújo (2023a), a análise de parâmetros específicos é essencial para avaliar o estado das vibrações:

- **Amplitude:** A amplitude da vibração é um dos parâmetros mais importantes na análise de sinais vibratórios. Ela representa a magnitude do movimento vibratório e é geralmente expressa em unidades de aceleração ( $m/s^2$ ) ou deslocamento (mm). A amplitude é diretamente proporcional à intensidade da vibração e pode indicar o nível de desgaste ou defeito em um componente. Vibrações de alta amplitude podem ser indicativas de falhas graves ou condições operacionais adversas.
- **Frequência:** A frequência da vibração refere-se à taxa de oscilação do sistema, geralmente medida em Hertz (Hz). Cada componente mecânico possui uma frequência natural de vibração, e a análise de frequências pode revelar ressonâncias ou problemas específicos, como desbalanceamento, desalinhamento

---

ou falhas em rolamentos. Identificar essas frequências permite um diagnóstico preciso e direcionado.

- **Fase:** A fase da vibração refere-se ao deslocamento relativo das vibrações entre diferentes pontos do sistema. A fase é importante para entender como diferentes partes do sistema vibram em relação umas às outras. Em muitos casos, diferenças de fase podem indicar problemas no alinhamento ou no comportamento dinâmico de diferentes componentes, como motores e acoplamentos.

A análise de vibrações é amplamente utilizada para determinar a condição operacional e mecânica de equipamentos. A principal vantagem dessa técnica é a capacidade de identificar problemas em desenvolvimento antes que se tornem graves e causem paradas não programadas. Para isso, o monitoramento regular das vibrações das máquinas pode ser realizado de forma contínua ou em intervalos programados. A análise de tendências é uma técnica útil para acompanhar a evolução das condições de vibração ao longo do tempo e prever falhas antes que elas ocorram (Scheffer; Girdhar, 2004).

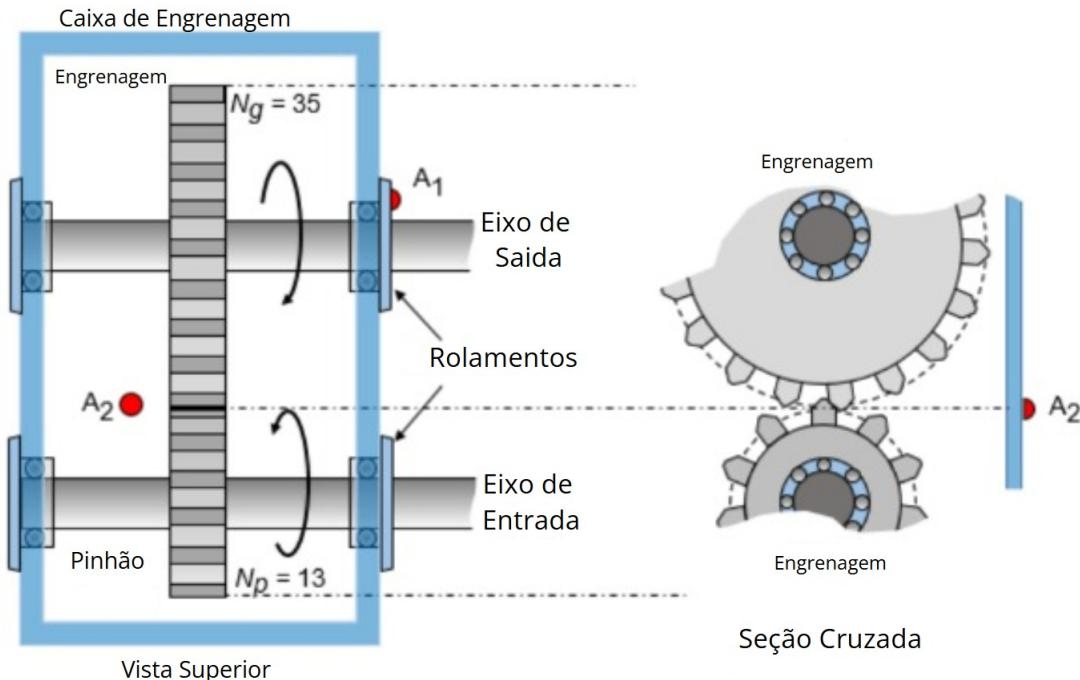
## 2.2 Falhas em Máquinas Rotativas

### 2.2.1 Caixa de Engrenagens

As caixas de engrenagens são dispositivos mecânicos projetados para transmitir energia e controlar a velocidade de rotação em uma variedade de aplicações industriais (Túma, 2009), como pode ver na imagem 2.2. Elas são encontradas em uma ampla gama de equipamentos, desde maquinário pesado, como guindastes e escavadeiras, até sistemas de transporte, como trens e elevadores. Uma caixa de engrenagens típica é composta por vários componentes rotativos, incluindo engrenagens, eixos e rolamentos. As engrenagens são responsáveis por transmitir o torque do motor para o equipamento ou máquina, enquanto os eixos proporcionam suporte estrutural e guiam o movimento das engrenagens. Os rolamentos ajudam a reduzir o atrito e suportam cargas radiais e axiais (Scheffer; Girdhar, 2004).

A caixa de engrenagem é um equipamento rotativo que pode gerar harmônicos de baixa frequência no espectro de vibração devido ao seu movimento contínuo e à interação das engrenagens. Esses harmônicos de baixa frequência, como o  $1\times \text{rpm}$  (revoluções por minuto), são esperados, pois estão diretamente associados à rotação do eixo principal da caixa de engrenagens. No entanto, além desses harmônicos básicos, também é possível observar componentes de alta frequência no espectro, que são gerados devido a outros fenômenos, como os impactos dos dentes das engrenagens e os rolamentos. Esses impactos ocorrem durante a interação entre os dentes

Figura 2.2 – Caixa de Engrenagens



Fonte: <https://www.mathworks.com/help/signal/ug/vibration-analysis-of-rotating-machinery.html>

das engrenagens, especialmente quando as engrenagens não estão perfeitamente alinhadas ou há desgaste nos dentes (Henkes *et al.*, 2016).

O espectro de vibração de uma caixa de engrenagens, portanto, revela informações importantes sobre a condição de operação do sistema. Entre os sinais mais comuns que aparecem no espectro de vibração de uma caixa de engrenagens, destacam-se os seguintes:

- **1× rpm:** Este pico no espectro corresponde diretamente à rotação do eixo principal da caixa de engrenagens. Ele representa a rotação fundamental do sistema, associada diretamente à velocidade de rotação do eixo.
- **2× rpm:** Este pico é um harmônico da rotação do eixo e ocorre devido à interação dos dentes das engrenagens e ao modo como o movimento rotacional é transmitido entre elas.
- **Frequência de Malha da Engrenagem (GMF):** A GMF (Gear Mesh Frequency) é uma frequência crítica no espectro de vibração de uma caixa de engrenagens. Ela é gerada pela interação dos dentes das engrenagens à medida que se engatam durante a rotação. A GMF pode ser calculada como o produto do número de dentes de uma engrenagem e sua velocidade de operação. Para uma engrenagem com  $N$  dentes e velocidade de rotação  $RPM$  (rotações por minuto), a GMF é dada pela seguinte fórmula:

$$\text{GMF} = \frac{N \times \text{RPM}}{60} \quad (2.1)$$

Onde:

- $N$  é o número de dentes da engrenagem,
- $\text{RPM}$  é a rotação por minuto,
- A divisão por 60 converte a unidade de tempo para segundos, já que a GMF é medida em Hertz (Hz), a unidade de frequência.

Esse pico de GMF no espectro de vibração é extremamente importante para a análise de falhas em caixas de engrenagens. Anomalias ou aumentos anormais dessa frequência podem indicar problemas como desgaste nos dentes das engrenagens, desbalanceamento ou desalinhamento das engrenagens, que são sinais precoces de falhas mecânicas. Portanto, a GMF fornece informações cruciais para o diagnóstico e a manutenção preditiva de sistemas de transmissão (Rodrigues, 2018).

### 2.2.2 Falha de Desbalanceamento

A Organização Internacional de Normalização (ISO) define desbalanceamento como:

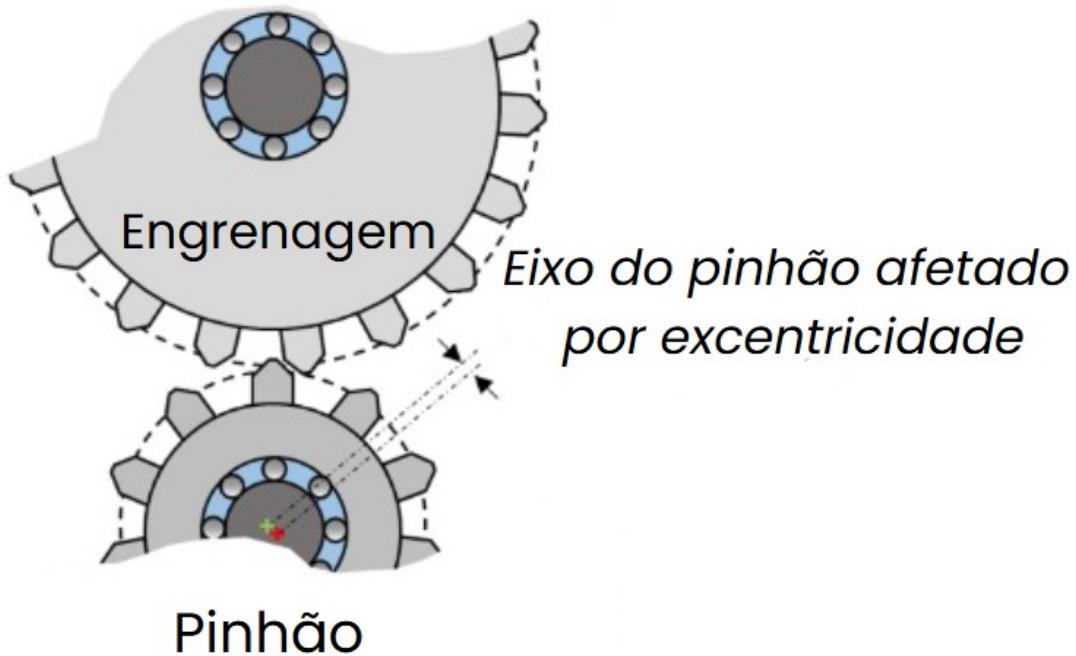
**Definição 2.1.** "A distribuição desigual de massa ao redor da linha central de rotação de um rotor. (Scheffer; Girdhar, 2004)"

Como regra geral, falhas distribuídas, como excentricidade e desalinhamento de engrenagens, produzirão bandas laterais e harmônicos que têm alta amplitude próximo à frequência de engrenamento dos dentes (Scheffer; Girdhar, 2004), como é possível observar na imagem 2.3. Essas irregularidades na transmissão de potência podem resultar em impactos significativos no desempenho e na integridade do sistema de engrenagens. Quando há desalinhamento ou excentricidade, as forças desiguais exercidas sobre as engrenagens durante o engrenamento geram vibrações adicionais, criando componentes de frequência lateral e harmônicos. Essas frequências adicionais são indicadores cruciais de falhas potenciais na caixa de engrenagens e podem ser detectadas por meio de análises de espectro de vibração.

### 2.2.3 Falha de Desalinhamento

O desalinhamento ocorre quando um eixo rotativo e os eixos acoplados a ele não giram em torno do mesmo eixo central. Na dinâmica de rotores, o desalinhamento pode ser paralelo, angular, ou uma combinação de ambos. O desalinhamento paralelo

Figura 2.3 – Desalinhamento de um rolamento no pinhão da caixa de engrenagens



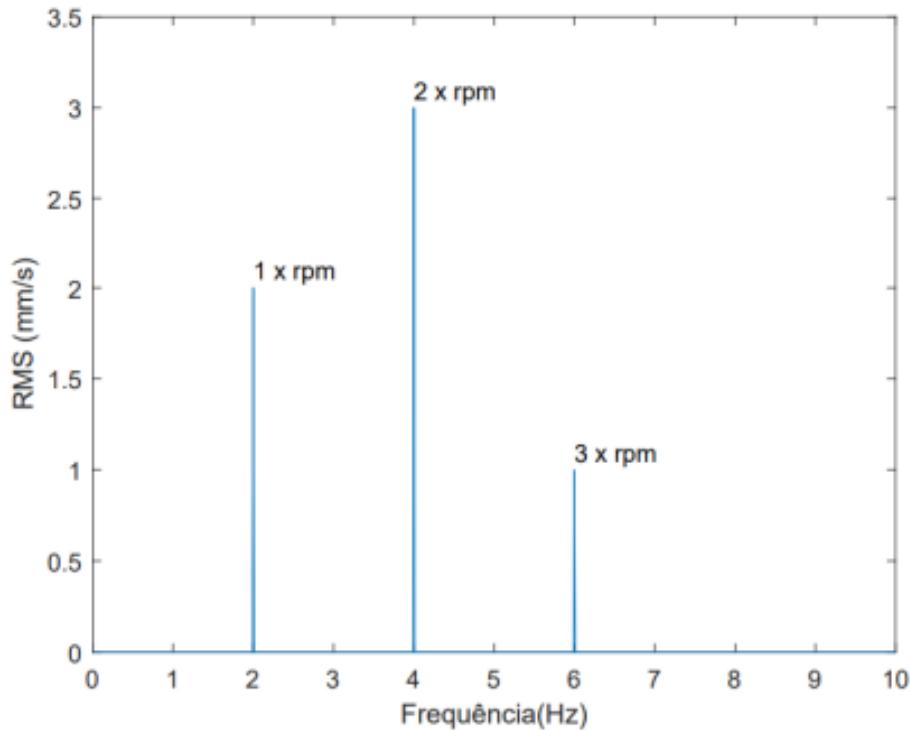
Fonte: <https://www.mathworks.com/help/signal/ug/vibration-analysis-of-rotating-machinery.html>

acontece quando os eixos de rotação não são colineares e não se cruzam, enquanto o desalinhamento angular ocorre quando os eixos de rotação não são colineares, mas suas linhas centrais se intersectam (Desouki et al., 2020).

Ambos os tipos de desalinhamento, paralelo e angular, podem ocorrer tanto na direção vertical quanto na horizontal, como apresentado nas figuras 2.6 2.5. A combinação desses desalinhamentos, conhecida como desalinhamento combinado, ocorre quando há desalinhamento paralelo e angular simultaneamente. Entre as principais causas desse problema estão falhas na instalação, desgaste ou folgas no maquinário, além de variações bruscas de carga. O desalinhamento pode gerar sobrecargas adicionais em várias partes do equipamento, o que aumenta vibrações e temperatura, resultando em desgaste acelerado do sistema (Desouki et al., 2020)(Araújo, 2023a).

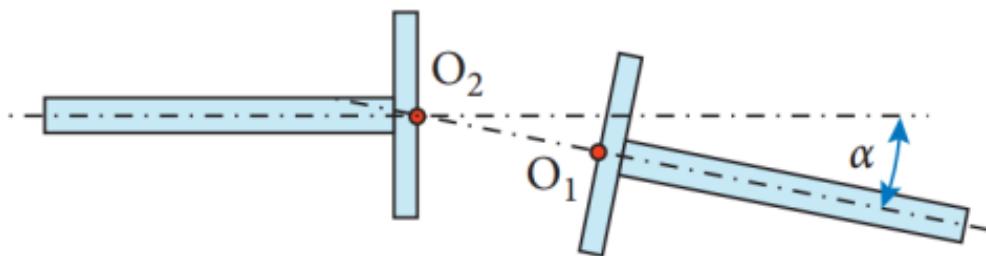
A detecção desse problema é feita por meio de análises no domínio da frequência, onde se observam picos nas frequências de 1x, 2x e 3x a frequência de rotação do motor, assim como um aumento no valor RMS da velocidade ou da aceleração, como apresentado nas figuras 2.4 e 2.5.

Figura 2.4 – Característica do espectro de vibração do desalinhamento paralelo



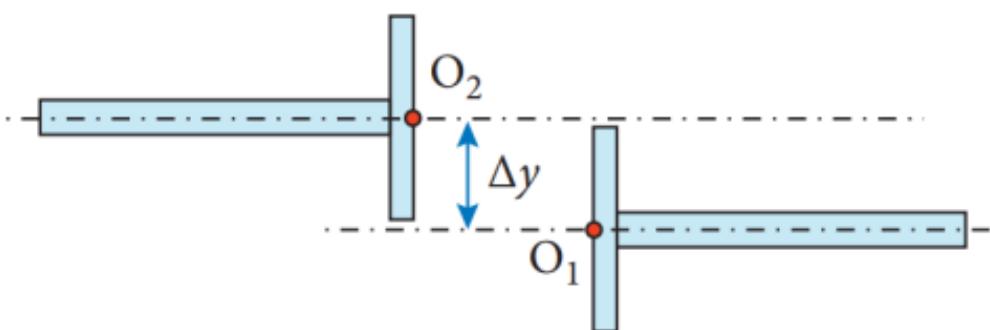
Fonte: (Desouki *et al.*, 2020)(Araújo, 2023a)

Figura 2.5 – Desalinhamento Angular



Fonte: (Desouki *et al.*, 2020)(Araújo, 2023a)

Figura 2.6 – Desalinhamento Paralelo



Fonte: (Desouki *et al.*, 2020)(Araújo, 2023a)

## 2.3 Aprendizagem de Máquina

A aprendizagem de máquina é um subcampo da inteligência artificial que se concentra na criação de algoritmos e modelos capazes de aprender a partir de dados e fazer previsões ou tomar decisões sem serem explicitamente programados para realizar essas tarefas. Esses algoritmos identificam padrões e relações nos dados, permitindo que o sistema melhore seu desempenho com o tempo à medida que é exposto a mais informações (Izbicki; Santos, 2020).

### 2.3.1 Árvores de Decisão

O modelo de árvore de decisão é uma técnica utilizada na aprendizagem de máquina para tarefas de classificação e regressão, caracterizada pela sua estrutura hierárquica que se assemelha a uma árvore.

Neste modelo, cada nó interno da árvore representa uma decisão baseada em um atributo dos dados, enquanto cada ramo indica o resultado dessa decisão, e os nós folhas correspondem às previsões finais do modelo (Meena; Choudhary, 2017) (Solanki, 2014). A principal vantagem das árvores de decisão reside na sua interpretabilidade e facilidade de visualização, permitindo que a lógica do processo de decisão seja facilmente compreendida. Além de ser facilmente implementada em Linguagem e Descrição de Hardware. Apenas com a utilização de sentenças condicionais.

### 2.3.2 Weka: Ferramenta para Aprendizado de Máquina

O Weka é um software de código aberto amplamente utilizado para mineração de dados e aprendizado de máquina. Desenvolvido pela Universidade de Waikato, na Nova Zelândia, ele oferece uma coleção de algoritmos de aprendizado de máquina para tarefas de classificação, regressão, agrupamento e seleção de atributos, além de ferramentas para pré-processamento de dados e visualização (Solanki, 2014).

O funcionamento do Weka é baseado em quatro componentes principais:

- **Explorador:** Interface gráfica que permite carregar conjuntos de dados, aplicar algoritmos de aprendizado de máquina e visualizar os resultados.
- **Experimenter:** Ferramenta para realizar experimentos controlados e comparar o desempenho de diferentes algoritmos.
- **KnowledgeFlow:** Ambiente visual para criar fluxos de trabalho de processamento de dados e aprendizado de máquina.
- **Comandos:** Interface de linha de comando para usuários avançados que preferem automatizar tarefas.

No contexto deste projeto, o Weka pode ser aplicado para treinar e avaliar modelos de aprendizado de máquina, como árvores de decisão, visando a detecção de falhas em máquinas rotativas. Por exemplo, é possível utilizar o Weka para pré-processar os dados de vibração, selecionar atributos relevantes e treinar modelos de classificação que identifiquem padrões associados a desbalanceamento, desalinhamento ou outras falhas. Além disso, a ferramenta permite a avaliação do desempenho dos modelos por meio de métricas como acurácia, precisão e recall, garantindo a escolha do algoritmo mais adequado para a aplicação proposta (Solanki, 2014).

## 2.4 Processamento de sinais

### 2.4.1 Filtro Butterworth

Um filtro passa-baixa é um tipo de filtro que permite a passagem de sinais com frequências abaixo de uma determinada frequência de corte enquanto atenua ou bloqueia sinais com frequências acima desse limite. A frequência de corte é o ponto em que a resposta em frequência do filtro começa a diminuir, marcando o limite entre as frequências que são atenuadas e as que são permitidas a passar. O filtro *Butterworth* é um tipo comum de filtro passa-baixa que apresenta uma resposta de magnitude de frequência maximamente plana na banda de passagem e uma rápida queda na banda de rejeição(Oppenheim; Willsky, 2010).

### 2.4.2 FFT

A Transformada de *Fourier* é uma ferramenta matemática fundamental utilizada na análise de sinais e sistemas. Ela permite decompor um sinal complexo no domínio do tempo em suas componentes individuais de frequência, revelando as diferentes contribuições de frequência que compõem o sinal original (Oppenheim; Willsky, 2010). Isso é crucial porque muitas vezes os sinais do mundo real são compostos por múltiplas frequências, e a Transformada de *Fourier* nos permite entender melhor a composição desses sinais. A Transformada de *Fourier* é definida pela seguinte equação:

$$X(f) = \int_{-\infty}^{\infty} x(t)e^{-i2\pi ft} dt \quad (2.2)$$

onde  $x(t)$  é o sinal no domínio do tempo,  $X(f)$  é a Transformada de *Fourier* do sinal em frequência, e  $f$  é a frequência. Esta equação descreve como o sinal  $x(t)$  é decomposto em suas componentes de frequência  $X(f)$  ao longo de todo o espectro de frequência (Oppenheim; Willsky, 2010).

Ao analisar um sinal no domínio da frequência, podemos identificar facilmente padrões, picos e tendências que podem não ser tão evidentes no domínio do tempo. Isso é útil na análise de sinais de áudio, sinais de vibração mecânica, sinais de comunicação e muitos outros tipos de sinais.

A FFT é um algoritmo eficiente para calcular a Transformada de *Fourier* discreta de um sinal. Ela é amplamente utilizada devido à sua rapidez computacional, permitindo analisar sinais e lidar com grandes conjuntos de dados de forma eficiente. A FFT desempenha um papel crucial em uma variedade de aplicações, desde processamento de sinais digitais até processamento de imagens médicas. Ao aplicar a FFT, podemos converter rapidamente um sinal do domínio do tempo para o domínio da frequência, facilitando a análise e extração de informações importantes do sinal (Oppenheim; Willsky, 2010).

#### 2.4.3 Transformada de Hilbert

A Transformada de *Hilbert* é uma ferramenta matemática que permite calcular o sinal analítico de um sinal no domínio do tempo (Sette; Filho, 2017). O sinal analítico consiste em duas partes: a parte real, que representa o sinal original, e a parte imaginária. A equação matemática da Transformada de *Hilbert* para um sinal  $x(t)$  é dada por:

$$H(x(t)) = \frac{1}{\pi} \int_{-\infty}^{\infty} \frac{x(\tau)}{t - \tau} d\tau \quad (2.3)$$

Essa equação descreve a convolução do sinal  $x(t)$  com  $\frac{1}{\pi t}$ , resultando no sinal analítico  $H(x(t))$  (Sette; Filho, 2017).

A Transformada de *Hilbert* O sinal envelope representa a magnitude do sinal original ao longo do tempo, destacando as características de amplitude do sinal. Isso é especialmente útil na análise de sinais de vibração em máquinas, como caixas de engrenagens, onde as mudanças na amplitude do sinal podem indicar a presença de falhas ou anomalias.

Para a aplicação de modelos de falhas de caixa de engrenagens através da vibração, a obtenção do sinal envelope usando a Transformada de *Hilbert* é crucial. O sinal envelope fornece informações essenciais sobre as características de vibração da caixa de engrenagens, permitindo a detecção de padrões de falha, como desequilíbrio, desalinhamento e desgaste excessivo. Além disso, o sinal envelope simplifica a análise do sinal de vibração, tornando mais fácil a identificação de padrões de falha e a implementação de algoritmos de classificação e diagnóstico de falhas (Feldman, 2011).

## 2.4.4 Parâmetros de Avaliação

### 2.4.4.1 EAM

O EAM (Erro Absoluto Médio) é uma métrica de avaliação que calcula a média das diferenças absolutas entre os valores observados  $y_i$  e os valores previstos  $\hat{y}_i$ . A equação do EAM é dada por:

$$MAE = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

onde:

- $n$  é o número total de amostras,
- $y_i$  são os valores reais,
- $\hat{y}_i$  são os valores previstos.

Essa métrica oferece uma visão clara da precisão de um modelo, uma vez que penaliza as discrepâncias entre os valores reais e estimados, sem levar em consideração o sinal da diferença. Diferentemente de outras métricas, como EQM, o EAM não amplifica o impacto de grandes erros, tornando-o uma medida mais robusta para dados com possíveis ruídos ou variações extremas. No contexto de sistemas de monitoramento de vibrações, o EAM pode ser útil para avaliar o desempenho dos modelos de classificação ao identificar a discrepancia entre os sinais reais e os previstos ([Schneider; Xhafa, 2022](#)).

Além disso, o EAM pode ser aplicado diretamente na detecção de defeitos em máquinas rotativas. Ao comparar o sinal de vibração de uma máquina em operação normal (saudável) com o sinal de uma máquina potencialmente falha, o erro absoluto médio entre esses sinais pode ser um indicativo da presença de uma falha. Um aumento significativo no valor do EAM pode sugerir que há uma discrepancia entre os padrões de vibração esperados e os observados, o que pode ser um sinal de desbalanceamento, desalinhamento ou outras falhas mecânicas. Dessa forma, o EAM não apenas avalia a precisão de modelos preditivos, mas também pode ser utilizado como uma ferramenta direta para a identificação de anomalias em sistemas de monitoramento de vibração.

### 2.4.4.2 EQM

O EQM (*Erro Quadrático Médio*), ou *Mean Squared Error* (MSE), é uma métrica amplamente utilizada para avaliar a precisão de um modelo. Ele calcula a média dos quadrados das diferenças entre os valores observados  $y_i$  e os valores esperados  $\hat{y}_i$ . A equação do EQM é dada por:

$$EQM = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

onde:

- $n$  é o número total de amostras,
- $y_i$  são os valores reais,
- $\hat{y}_i$  são os valores previstos.

O EQM é sensível a grandes variações no sinal, ou seja, ele penaliza de forma mais intensa os erros maiores, o que pode ser útil quando é necessário considerar a gravidade de discrepâncias significativas. Em sistemas desbalanceados, como os analisados neste trabalho, o EQM pode ser valioso para detectar grandes flutuações no sinal, mas, ao mesmo tempo, pode ser mais suscetível a ruídos ou flutuações pequenas, o que pode afetar a precisão na classificação. Apesar disso, o EQM ainda é um parâmetro importante quando se busca uma análise mais sensível a erros de grande magnitude ([Binieli, 2025](#)).

## 2.5 Hardware Reconfigurável

### 2.5.1 SoC

Um *SoC* é um chip de silício que integra um ou mais núcleos de processadores – microprocessadores (*MPUs*), microcontroladores (*MCUs*) e/ou processadores digitais de sinais (*DSPs*) – juntamente com memória, aceleradores de hardware, funções periféricas e possivelmente diversas outras funcionalidades ([Abdelkrim; Othman; Saoud, 2017](#)).

### 2.5.2 SoC- FPGAs

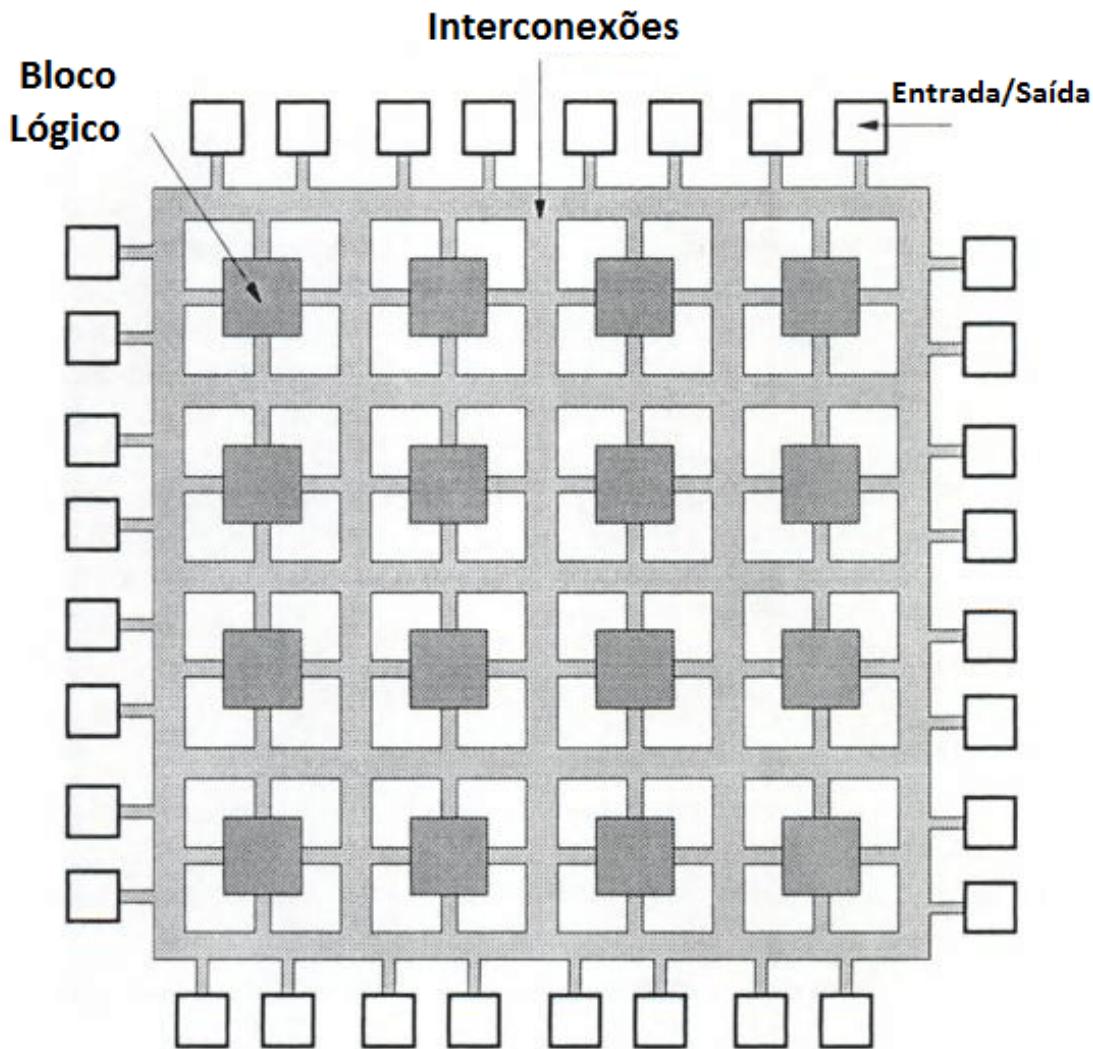
Um dispositivo *FPGA* é uma matriz de portas lógicas programáveis possui uma estrutura constituída de uma matriz de blocos programáveis interconectados por uma rede de conexões reconfiguráveis, como apresentado em [2.7](#). A principal vantagem de um *FPGA* é a capacidade de configurar seu hardware para executar qualquer combinação de funções digitais desejadas ([Xilinx, 2016](#)).

No entanto, atualmente o termo "matriz de portas lógicas programáveis" não reflete mais a capacidade e a funcionalidade desses dispositivos programáveis modernos. Além disso, os *FPGAs* atuais podem incorporar um ou mais núcleos de processadores, sejam eles *soft* ou *hard cores*, por isso surge assim os *SoCs FPGAs*.

A arquitetura reconfigurável dos *FPGAs* significa que a estrutura e a topologia do hardware podem ser modificadas conforme necessário para se adequarem a

problemas específicos. Como parte dos dispositivos lógicos programáveis (*PLDs*), os *FPGAs* oferecem uma grande flexibilidade, tornando-os adequados para uma ampla variedade de aplicações (Xilinx, 2016).

Figura 2.7 – FPGA Arquitetura



Fonte: [https://www.researchgate.net/figure/Figura-1-Arquitetura-basica-de-um-FPGA-FPGAs-utilizam-o-conceito-de-Blocos-fig1\\_303773247](https://www.researchgate.net/figure/Figura-1-Arquitetura-basica-de-um-FPGA-FPGAs-utilizam-o-conceito-de-Blocos-fig1_303773247) [acessado 3 Jul, 2024]

A arquitetura dos *FPGAs* inclui quatro componentes principais:

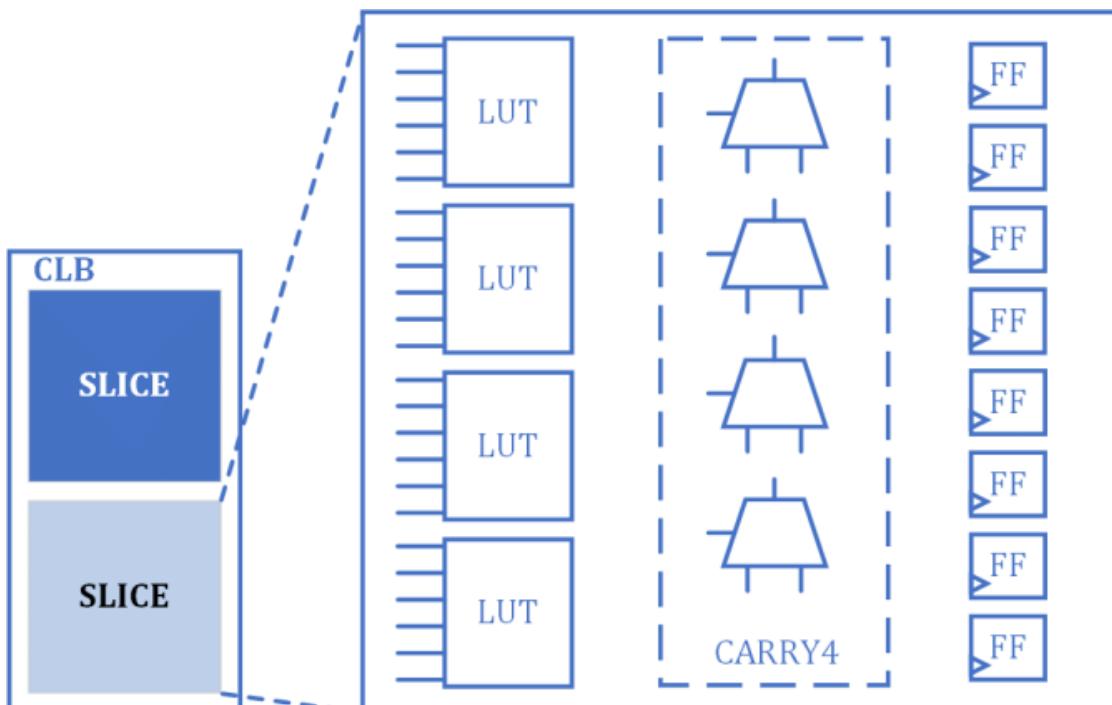
- Módulo de Lógica ou Bloco Lógico Configurável (CLB): Estes blocos são responsáveis pela execução das funções lógicas. Cada CLB contém "Slices", e cada *Slice* inclui *Look-up Tables* (LUTs), *flip-flops* (FFs) e multiplexadores. Por exemplo, na família 7 da Xilinx, cada CLB possui 2 *Slices*, e cada *Slice* tem 4 LUTs de 6 entradas, 8 *flip-flops* e alguns multiplexadores, como demonstrado em 2.8.
- Elemento de Roteamento: Este componente conecta os diferentes blocos

lógicos configuráveis 2.9, permitindo a criação de circuitos mais complexos.

Os circuitos combinacionais são implementados nas *LUTs*, que podem ser interconectadas para formar circuitos mais sofisticados.

- Memória: Armazena o *bitstream*, que é o arquivo de configuração que define como o *FPGA* deve ser programado.
- Circuitos Especiais: Incluem *DSPs* (processadores de sinal digital), como é possível ver em 2.10. Além de *RAM* (*Random Access Memory*), *PLLs* (*Phase-Locked Loops*) e outros componentes especializados que aumentam a capacidade de processamento do *FPGA*.

Figura 2.8 – Arquitetura de um *slice* em *FPGA* de serie 7.

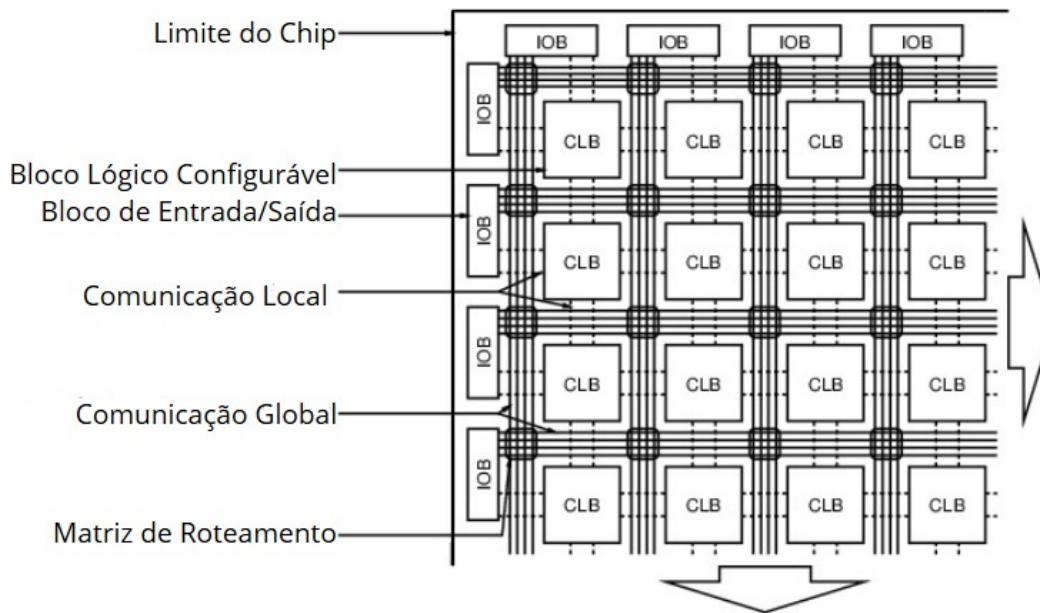


Fonte: [https://www.researchgate.net/figure/The-underlying-architecture-of-Xilinx-7-series-FPGA\\_fig2\\_333156195](https://www.researchgate.net/figure/The-underlying-architecture-of-Xilinx-7-series-FPGA_fig2_333156195) [acessado 3 Jul, 2024]

Em sua estrutura interna os circuitos combinacionais nos *FPGAs* são implementados nas *LUTs*, que atuam como tabelas de consulta para funções lógicas. Os circuitos sequenciais utilizam *FFs* para armazenar o estado atual do circuito. A arquitetura permite que os *CLBs* sejam acessados em paralelo, permitindo que algoritmos mapeados em hardware usem o paralelismo para acelerar a execução (Xilinx, 2016).

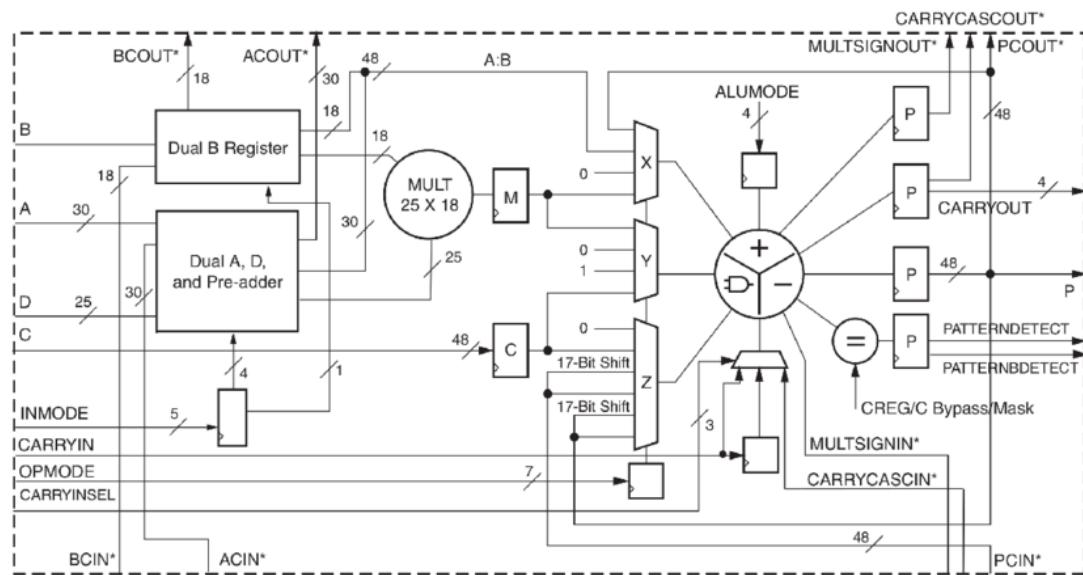
Neste trabalho foi utilizado o *SoC Zynq-7000* (dispositivo XC7Z020-1CLG400C) (AMD (Advanced Micro Devices), 2024), que integra a programabilidade de software de um processador *dual core Arm® Cortex A9* com a programabilidade de *hardware* de um *FPGA* da familia *Artix 7*. Isso permite a realização de análises avançadas e aceleração de *hardware*, além de integrar CPU (*Central Processing Unit*, *DSPs*,

Figura 2.9 – Arquitetura simplificada de um *FPGA* com Matriz de Roteamento.



Fonte: [https://www.researchgate.net/figure/Simplified-schematic-of-an-FPGA-top-left-corner\\_fig1\\_4008529](https://www.researchgate.net/figure/Simplified-schematic-of-an-FPGA-top-left-corner_fig1_4008529) [acessado 3 Jul, 2024]

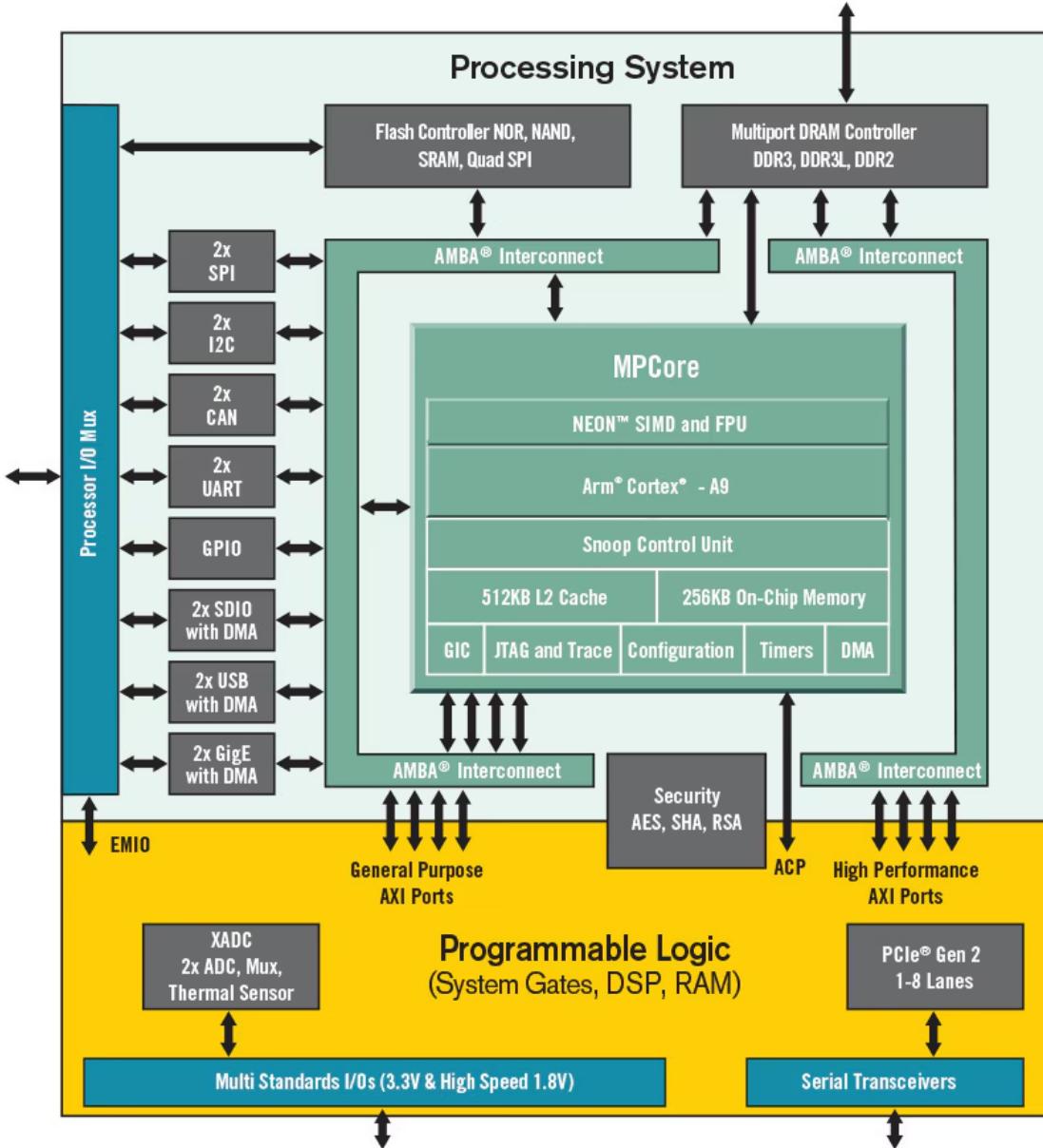
Figura 2.10 – Esquema do *DSP SLICE* disponível nos *FPGAs* da série 7 da *Xilinx*.



Fonte: *Near-Data Query Processing on Heterogeneous FPGA-based Systems - Scientific Figure on ResearchGate*. Disponível em: [https://www.researchgate.net/figure/Schematic-of-the-DSP-SLICE-available-in-the-Xilinx-7-series-FPGAs-Xil18a\\_fig4\\_360188046](https://www.researchgate.net/figure/Schematic-of-the-DSP-SLICE-available-in-the-Xilinx-7-series-FPGAs-Xil18a_fig4_360188046) [acessado 3 Jul, 2024]

ASSPs(*(Application-specific standard parts)*) e funcionalidades de sinais mistos em um único dispositivo, é possível visualizar sua arquitetura na Fig. 2.11.

Figura 2.11 – Esquema da Zynq7000



Fonte:<https://www.amd.com/pt/products/adaptive-socs-and-fpgas/soc/zynq-7000.html#tabs-90e9af6e7c-item-a343e90353-tab>[acessado 3 Jul, 2024]

### 2.5.3 IP - FFT

Um IP (Intellectual Property) é um bloco de propriedade intelectual que pode ser reutilizado em projetos de hardware ou software. No contexto de FPGAs (Field-Programmable Gate Arrays), um IP é um circuito digital pré-projetado e otimizado para realizar uma função específica, como processamento de sinais, controle de memória, ou, no caso deste trabalho, a Transformada Rápida de Fourier. O uso de IPs

permite que os desenvolvedores acelerem o processo de design, reduzindo o tempo de desenvolvimento e aumentando a confiabilidade do sistema.

O IP da FFT da Xilinx é um núcleo de processamento de sinais que implementa a Transformada Rápida de Fourier, um algoritmo eficiente para calcular a Transformada Discreta de Fourier (DFT). Esse IP é altamente configurável e suporta diferentes tamanhos de transformada, precisões de dados e arquiteturas, tornando-o adequado para uma variedade de aplicações, como processamento de sinais digitais, comunicações sem fio e análise espectral ([Xilinx, 2022](#)).

#### 2.5.3.1 Funcionamento do IP da FFT

O IP da FFT da Xilinx implementa o algoritmo de Cooley-Tukey, que é uma abordagem eficiente para calcular a DFT. Ele suporta transformadas de tamanho  $N = 2^m$ , onde  $m$  varia de 3 a 16, permitindo transformadas de 8 a 65536 pontos. O IP pode ser configurado para realizar tanto a FFT direta quanto a inversa (IFFT), e suporta diferentes tipos de aritmética, incluindo ponto fixo, ponto flutuante e aritmética de bloco flutuante ([Xilinx, 2022](#)).

#### 2.5.3.2 Recepção e Processamento de Dados

O IP da FFT recebe os dados de entrada através de uma interface AXI4-Stream, que é um protocolo padrão para transferência de dados em sistemas baseados em FPGAs. Os dados de entrada são compostos por amostras complexas, onde cada amostra contém uma parte real e uma parte imaginária. Esses dados são enviados ao IP em ordem natural, e o IP pode ser configurado para produzir a saída em ordem natural ou em ordem bit-reversa.

O processamento dos dados é realizado em estágios, onde cada estágio executa operações para calcular a FFT. Dependendo da arquitetura escolhida (Pipelined Streaming I/O, Radix-4 Burst I/O, Radix-2 Burst I/O ou Radix-2 Lite Burst I/O), o IP pode processar os dados de forma contínua ou em blocos, com diferentes trade-offs entre velocidade de processamento e utilização de recursos ([Xilinx, 2022](#)).

#### 2.5.3.3 Configuração do IP

A configuração do IP da FFT é feita através de uma interface de configuração AXI4-Stream, onde parâmetros como o tamanho da transformada (NFFT), a direção da transformada (FFT ou IFFT), o escalonamento e o comprimento do prefixo cílico (para aplicações de comunicação) podem ser ajustados em tempo de execução.

O IP também oferece opções de arredondamento e escalonamento, que podem ser configuradas para garantir que os resultados da FFT estejam dentro da faixa dinâmica desejada, evitando overflow ou perda de precisão([Xilinx, 2022](#)).

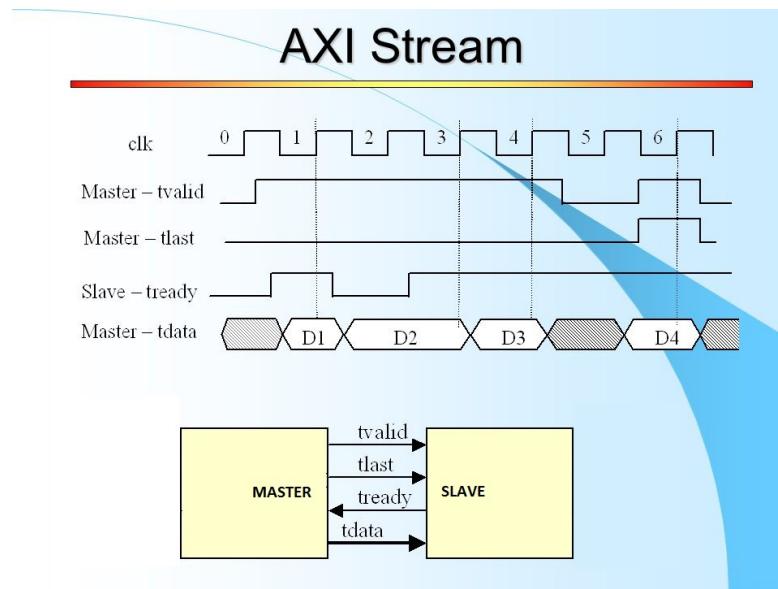
### 2.5.4 AXI4-Stream

No contexto deste trabalho, o **AXI4-Stream** foi utilizado como uma interface de comunicação fundamental para o processamento eficiente dos dados de vibração. O **AXI4-Stream** é um barramento de alta performance que faz parte da arquitetura **AMBA** (Advanced Microcontroller Bus Architecture) da ARM, sendo projetado para a transmissão de dados de forma contínua entre diferentes módulos de um sistema, sem a necessidade de endereçamento complexo. Ele é amplamente utilizado em sistemas baseados em FPGA, como o que foi empregado neste trabalho, para garantir a transferência eficiente de grandes volumes de dados (Angadi; Shenvi; Lokesh, 2024).

O funcionamento do **AXI4-Stream** é relativamente simples, como apresentado na Fig. 2.12. Ele permite a comunicação entre um produtor de dados e um consumidor, utilizando três sinais principais para garantir a transferência eficiente:

- **TVALID**: Indica que o dado no barramento é válido. Ele está pronto para ser processado.
- **TREADY**: Indica que o consumidor está pronto para receber o dado.
- **TDATA**: Contém os dados a serem transferidos.

Figura 2.12 – Comunicação AXI4-Stream



Fonte:[https://adaptivesupport.amd.com/s/question/0D52E00006hpNEvSAM/axi-stream-t-valid?language=en\\_US](https://adaptivesupport.amd.com/s/question/0D52E00006hpNEvSAM/axi-stream-t-valid?language=en_US)[acessado 02 Fev, 2025]

Essa interface será importante para o projeto visto que, o **IP da FFT da Xilinx**, que foi utilizado para o processamento dos sinais de vibração, requer o uso do **AXI4-Stream** para garantir que os dados sejam transmitidos de forma eficiente e sem atrasos entre os módulos (Setiawan; Adiono, 2018).

## 2.6 Máquinas de Estados

As máquinas de estados finitos (MEFs) são um modelo matemático utilizado para representar sistemas com comportamento discreto, que podem estar em um de vários estados em determinado momento. Elas são compostas por um conjunto de estados, um conjunto de entradas possíveis e um conjunto de transições que determinam como o sistema muda de estado com base nas entradas (Alavi; Aliaga; Murga, 2016).

Existem duas variações importantes de máquinas de estados finitos: as *Máquinas de Moore* e as *Máquinas de Mealy*, que se diferenciam principalmente pela maneira como geram as saídas durante o processamento de uma entrada.

### 2.6.1 Máquina de Moore

A *Máquina de Moore* é uma máquina de estados finitos onde a saída é associada exclusivamente ao estado em que a máquina se encontra. Ou seja, para cada estado, existe uma saída específica que será gerada, independentemente da entrada recebida. Isso significa que a máquina produz uma saída baseada apenas no estado atual, e não nas transições ou entradas (Klimovich; Solov'ev, 2010).

### 2.6.2 Máquina de Mealy

Já a *Máquina de Mealy* também é uma máquina de estados finitos, mas, ao contrário da máquina de Moore, a saída não depende apenas do estado atual. A saída de uma máquina de Mealy é gerada durante as transições entre estados e depende tanto do estado atual quanto da entrada recebida. Ou seja, a saída varia conforme a combinação entre o estado e o símbolo de entrada (Brito; Martendal; Oliveira, 2003).

Esse tipo de máquina é útil em sistemas onde a saída precisa refletir tanto o estado quanto a entrada no momento da transição.

## 2.7 Estado da Arte

Analizar vibrações dentro de sistemas de rolamentos é crucial para a manutenção eficaz de máquinas industriais, minimizando despesas, otimizando a eficiência operacional e prevenindo falhas. Técnicas como Transformadas Rápidas de Fourier (Adamsab; Shivakumar, 2018), Transformadas de Hilbert (McInerny SA e Dai, 2003) e Transformadas de Wavelet (Tao *et al.*, 2020) são utilizadas por sua capacidade de discernir padrões de vibração associados a diversos estados operacionais. As FFTs, em particular, destacam-se na decomposição de sinais no domínio da frequência,

auxiliando na identificação de assinaturas de vibração distintas correspondentes a diferentes condições operacionais ([Qiu et al., 2006](#)).

Entender disparidades na visualização de formas de onda é essencial para abordar falhas de rolamento em caixas de engrenagens, especialmente problemas de desequilíbrio que representam uma parte significativa dos problemas de equipamentos rotativos ([Scheffer; Girdhar, 2004](#)) ([MathWorks, 2024](#)).

No âmbito da análise de vibração e diagnóstico de falhas em sistemas de rolamentos, a integração de aprendizado de máquina em abordagens de sistemas embarcados apresenta uma via promissora. Algoritmos de ML permitem a automação da detecção de falhas e extração de padrões de vibração intrincados, aprimorando a eficácia e precisão dos esforços de manutenção preditiva. Implementar ML em sistemas embarcados oferece uma abordagem alternativa para análise de vibração e diagnóstico de falhas usando métodos de computação de borda ([Verma et al., 2021](#)).

Métodos de aprendizado profundo estão sendo cada vez mais estudados para monitoramento de saúde em máquinas rotativas, aproveitando recursos de vibração e térmicos. Por exemplo, Redes Neurais Convolucionais (CNNs) junto com transformadas de *Wavelet* para monitorar a saúde de máquinas rotativas foram propostos em ([Ong et al., 2024](#)). O método híbrido proposto alcançou uma evolução promissora na análise de vibração e diagnóstico de falhas, levando a uma compreensão mais abrangente do estado operacional das máquinas. Em relação às soluções de sistemas embarcados para análise de vibração, o trabalho apresentado em ([Bengherbia et al., 2020](#)) propôs um sensor sem fio com base em uma implementação de *hardware* de um coprocessador de rede neural ADALINE, destacando capacidades de processamento eficientes em comparação com métodos tradicionais baseados em FFT. Essas soluções destacam as diversas abordagens disponíveis para análise de vibração em sistemas de rolamentos industriais. A tabela 2.1 apresenta um resumo dos trabalhos relacionados que implementam técnicas de inteligência computacional para diagnóstico de falhas em sistemas de rolamentos.

Tabela 2.1 – Resumo dos Métodos para Diagnóstico de Falhas e Multi-Falhas em Máquinas Rotativas.

<b>Artigo/Técnica</b>	<b>Descrição</b>
Son, J. et al., 2016: Acelerômetros baseados em MEMS e sensores de corrente ( <a href="#">Son et al., 2016</a> ).	Sensores com comunicação sem fio e métodos de classificação como <i>random forest</i> e <i>Fuzzy k-nearest</i> . Plataforma: computador <i>desktop</i> . Detalhes do computador pessoal não foram fornecidos.

Continua

Gawde, S. et al., 2024: Estratégia Explicável de Manutenção Preditiva para Diagnóstico de Multi-Falhas usando Fusão de Dados de Múltiplos Sensores ( <a href="#">Gawde et al., 2024</a> ).	Diagnosticar múltiplas falhas com base na fusão de sensores proporcionou considerável precisão e validade convincente usando dados brutos de FFT. Métodos de classificação empregados incluíram <i>Random Forest</i> , KNN e SVM. Plataforma: computador <i>desktop</i> . Detalhes do computador pessoal não foram fornecidos.
Ong, P. et al., 2024: Abordagem de aprendizado profundo usando vibrações e características térmicas ( <a href="#">Ong et al., 2024</a> ).	Imagens térmicas fornecem informações mais claras com diferenciação mais distinta para classificação. Métodos de classificação empregados incluem SNN, CNN e transformada <i>Wavelet</i> . Plataforma: computador <i>desktop</i> . Detalhes do computador pessoal não foram fornecidos.
Bengherbia, B. et al., 2020: Implementação <i>FPGA</i> de um co-processador de rede neural ADALINE ( <a href="#">Bengherbia et al., 2020</a> ).	Aplicação de rede de sensores sem fio. Método de classificação: rede neural Adaline e comparação com análise FFT tradicional. Plataforma: <i>FPGA Artix 7 Cmod A7</i> .
Qiu, H. et al., 2006: Detecção de assinatura fraca baseada em filtro <i>Wavelet</i> ( <a href="#">Qiu et al., 2006</a> ).	Detecção de sinais fracos de defeitos mecânicos. Método de classificação/processamento: decomposição <i>Wavelet</i> e <i>denoising</i> baseado em filtro <i>Wavelet</i> . Entropia mínima de <i>Shannon</i> e decomposição em valores singulares para detecção de periodicidade. Conclui que o filtro <i>Wavelet</i> é mais eficaz para sinais de impulso, enquanto a decomposição é melhor para sinais suaves. Plataforma: computador <i>desktop</i> . Detalhes do computador pessoal não foram fornecidos.
Merendino, et al, 2011 ( <a href="#">Merendino et al., 2011</a> ): Um sistema embarcado para análise de vibração em tempo real	Análise de vibração no diagnóstico de falhas em máquinas rotativas utilizando métodos espetrais avançados para detecção precoce. Plataforma: ARM9 STR912F44W RISC de 32 bits.

Continua

---

Contreras, et al, 2008 ([Contreras-Medina et al., 2008](#)): Sistema embarcado de analisador de vibração multi-canal baseado em *FPGA* para aplicações industriais em detecção automática de falhas

---

Analisadores de espectro de canal único tradicionais para análise de vibração não possuem a capacidade de análise multi-canal simultânea. Este trabalho apresenta um analisador de vibração de 3 eixos baseado em *FPGA* de baixo custo com pós-processamento personalizável para detecção automática de falhas em máquinas, demonstrado por meio de dois estudos de caso. Plataforma: *FPGA Xilinx Spartan-3* com frequência de operação de 25MHz.

---

## 3 Metodologia

### 3.1 Simulação de Falhas em Máquinas Rotativas

Para esse trabalho foi utilizado um exemplo de simulação do *MathWorks* ([MathWorks, 2024](#)), o qual analisa vibrações para uma caixa de engrenagens cujo os eixos giram a uma velocidade fixa.

Foi considerada uma caixa de engrenagens idealizada composta por um pinhão com 13 dentes engrenado com uma engrenagem de 35 dentes. O pinhão é acoplado a um eixo de entrada que está conectado a um motor principal, enquanto a engrenagem está ligada a um eixo de saída. Os eixos são suportados por rolamentos na carcaça da caixa de engrenagens. Dois acelerômetros,  $A_1$  e  $A_2$ , são posicionados nas carcaças do rolamento e da caixa de engrenagens, respectivamente, e operam a uma taxa de amostragem de 20 kHz, como mostra a figura 2.2. O pinhão gira a uma taxa  $f_{Pinhão} = 22,5$  Hz ou 1350 rpm. A velocidade de rotação da engrenagem e do eixo de saída é dada por

$$f_{Engrenagem} = f_{Pinhão} \times \frac{\text{Número de dentes do pinhão } (N_p)}{\text{Número de dentes da engrenagem } (N_g)}. \quad (3.1)$$

A frequência da malha do dente, também chamada de frequência da malha da engrenagem, é a taxa na qual os dentes da engrenagem e do pinhão engatam periodicamente,

$$f_{Malha} = f_{Pinhão} \times N_p = f_{Engrenagem} \times N_g. \quad (3.2)$$

Para simular uma falha distribuída em um rolamento, como excentricidade ou desalinhamento, foram introduzidas componentes de bandas laterais ao redor da frequência de malha da engrenagem. Essas bandas laterais são características típicas de falhas distribuídas e se manifestam como componentes de frequência adicionais, estreitamente agrupadas em torno de múltiplos inteiros da frequência de malha ([MathWorks, 2024](#)) ([Scheffer; Girdhar, 2004](#)). No modelo de simulação, foram adicionadas três componentes de bandas laterais de amplitudes decrescentes de cada lado da frequência de malha da engrenagem. Esses sinais de bandas laterais foram então somados ao sinal de vibração, resultando em uma modulação de amplitude que reflete o comportamento típico de um rolamento com falha distribuída. Essa abordagem permite a criação de um sinal de vibração mais realista, facilitando a análise e identificação de falhas distribuídas em rolamentos.

## 3.2 Criação de uma Base de Dados

Para criar o conjunto de dados, foram utilizadas ferramentas do *Matlab* ([MathWorks, 2024](#)) e *Python* para emular sinais de vibração de uma caixa de engrenagens com componentes rotativos. O conjunto de dados compreende dois conjuntos de sinais: um representando um sistema saudável e outro introduzindo desalinhamento no pinhão.

Para cada conjunto de sinais, a análise iniciou-se com a geração de dados de vibração de uma caixa de engrenagens com eixos rotativos em velocidade fixa. Em seguida, foi realizada uma amostragem do sinal saudável e do sinal com desalinhamento a uma taxa de 23kHz. Dessa forma, ambos os sinais foram combinados para constituir a base de dados.

No entanto, ambos os conjuntos de sinais passaram por duas técnicas de pré-processamento antes de serem agrupados na base de dados. O primeiro, denominado *FHF-model*, inclui um filtro *Butterworth* com frequência de corte de 1KHz ([Scheffer; Girdhar, 2004](#)) transformada de *Hilbert* seguida de uma *FFT*, antes de serem exportados como arquivos de texto para constituir a base de dados (ver Figura 3.1). O segundo, denominado *F-model*, inclui o filtro *Butterworth* com frequência de corte de 1KHz e apenas uma etapa de análise *FFT*, antes de exportar os valores para outro arquivo de texto (ver Figura 3.2) ([Menezes, 2015](#)). Ambos os modelos são detalhados na seguinte seção.

## 3.3 Proposta de Processamento de Sinais de Vibração de Máquinas Rotativas

### 3.3.1 Modelo F

O modelo F de pré-processamento é uma técnica utilizada para preparar sinais para análise de falha da vibração de rolamentos. Este modelo é projetado para lidar tanto com sinais que possuem o ruído de desbalanceamento quanto com sinais que representam um rolamento saudável.

O processo começa com a aplicação de um filtro *Butterworth* aos sinais, que possuem uma frequência de corte de 1kHz. Este filtro é escolhido por sua característica de resposta em frequência suave e ausência de ondulações, o que o torna ideal para remover ruídos indesejados sem distorcer significativamente o sinal original ([Scheffer; Girdhar, 2004](#)) ([Menezes, 2015](#)).

Após a filtragem, os sinais processados passam por uma *FFT*. Isso permite a análise das componentes de frequência do sinal, facilitando a identificação de padrões

e anomalias que poderiam indicar a presença de falhas.

Os resultados da *FFT* fornecem valores de frequência e suas respectivas intensidades. Esses dados são cruciais para a análise, pois permitem a identificação de características específicas que podem estar associadas a diferentes tipos de falhas mecânicas. Para facilitar a análise e a construção de uma base de dados robusta, os valores de frequência e intensidade resultantes do processamento são exportados para um arquivo CSV. Para que assim, a base de dados seja construída.

### 3.3.2 Modelo FHF

O modelo de pré-processamento denominado FHF é uma técnica aplicada para preparar sinais de vibração, tanto ruidosos quanto não ruidosos, visando à construção de uma base de dados para análise subsequente. Este modelo envolve várias etapas de processamento para garantir que as características importantes dos sinais sejam preservadas e destacadas, facilitando a detecção de falhas em sistemas mecânicos.

A primeira etapa do modelo FHF é a aplicação de um filtro *Butterworth* aos sinais. O filtro *Butterworth* é configurado com uma frequência de corte de 1kHz.

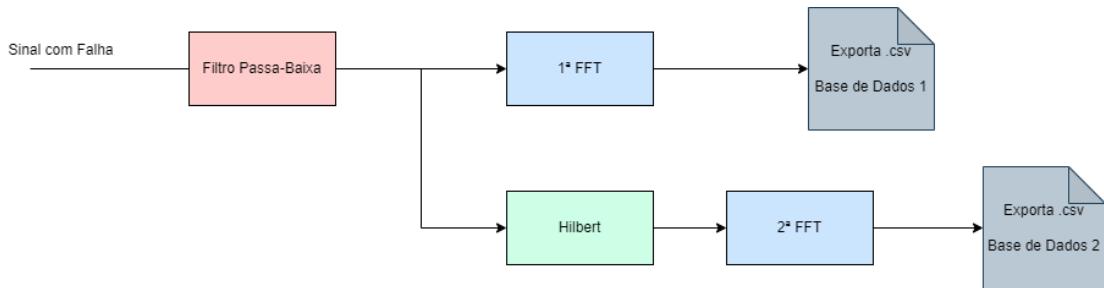
Após a filtragem, os sinais processados passam pela transformada de *Hilbert*. A transformada de *Hilbert* é uma técnica matemática que permite obter o envelope do sinal, ou seja, uma representação que destaca a amplitude das oscilações no tempo. Esta etapa é crucial, pois o envelope do sinal pode revelar padrões e características que não são facilmente observáveis no sinal original (Menezes, 2015). Em seguida, é realizada uma Transformada Rápida de *Fourier* sobre o sinal obtido da transformada de *Hilbert*.

Finalmente, os valores de frequência e intensidade resultantes da *FFT* são exportados para um arquivo CSV. Este formato de arquivo é amplamente utilizado devido à sua simplicidade e compatibilidade com diversas ferramentas de análise de dados. Dessa forma é possível construir uma base de dados organizada, que pode ser utilizada para treinamento e validação de algoritmos de detecção de falhas, conforme descrito na seguinte seção.

## 3.4 Modelo de Classificação de Falhas

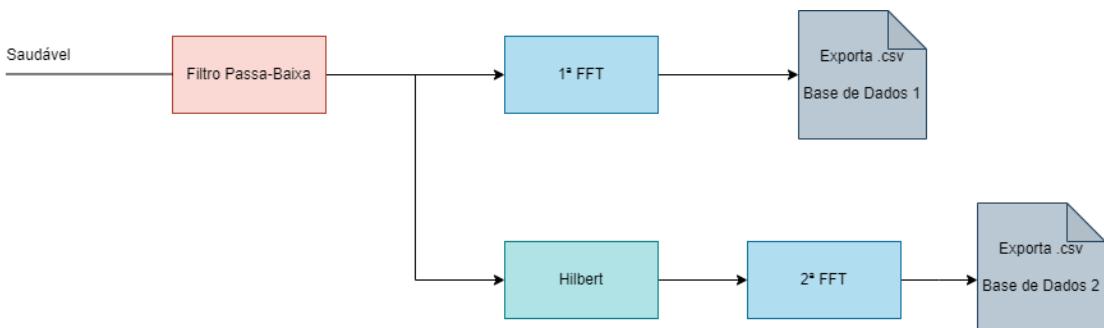
O modelo J48, também conhecido como C4.5 no software *WEKA*, foi escolhido para classificar a falha de desbalanceamento na caixa de engrenagens. Este algoritmo de árvore de decisão é amplamente utilizado para a classificação de dados. Ele constrói a árvore de decisão analisando um conjunto de dados de treinamento e emprega o conceito de ganho de informação para determinar o atributo que melhor

Figura 3.1 – Caminho de Dados do Sinal com Falha



Fonte: Autora

Figura 3.2 – Caminho de Dados do Sinal Sem Falhas



Fonte: Autora

divide os dados. Este processo envolve a partição recursiva do conjunto de dados em subconjuntos menores com base nos atributos selecionados. A divisão continua até que cada subconjunto contenha instâncias pertencentes à mesma classe, momento em que um nó folha representando essa classe é criado na árvore de decisão. Através do componente *KnowledgeFlow* no *WEKA*, os usuários podem realizar vários testes para melhorar o desempenho da classificação. Este componente avalia todos os testes potenciais para dividir o conjunto de dados, selecionando aquele que proporciona o maior ganho de informação ([Solanki, 2014](#)).

A escolha do modelo J48 foi motivada pela sua robustez e eficácia na classificação de padrões, como os encontrados em dados de vibração de caixas de engrenagens. Ao aplicar o modelo J48 para detectar desbalanceamento, foi possível identificar de forma razoável os sinais de falha, contribuindo para a manutenção preditiva.

### 3.5 Implementação em SoC FPGA do Modelo de Previsão

A implementação do algoritmo de detecção de falhas em máquinas de rolagem será realizada utilizando um *SoC FPGA*, onde será desenvolvida uma máquina de estados finitos (*FSM - Finite State Machine*) para gerenciar o fluxo de dados e a lógica de controle. Detalhes técnicos específicos e a metodologia completa de imple-

mentação serão apresentados no TCC2.

### 3.6 Materiais e Ferramentas

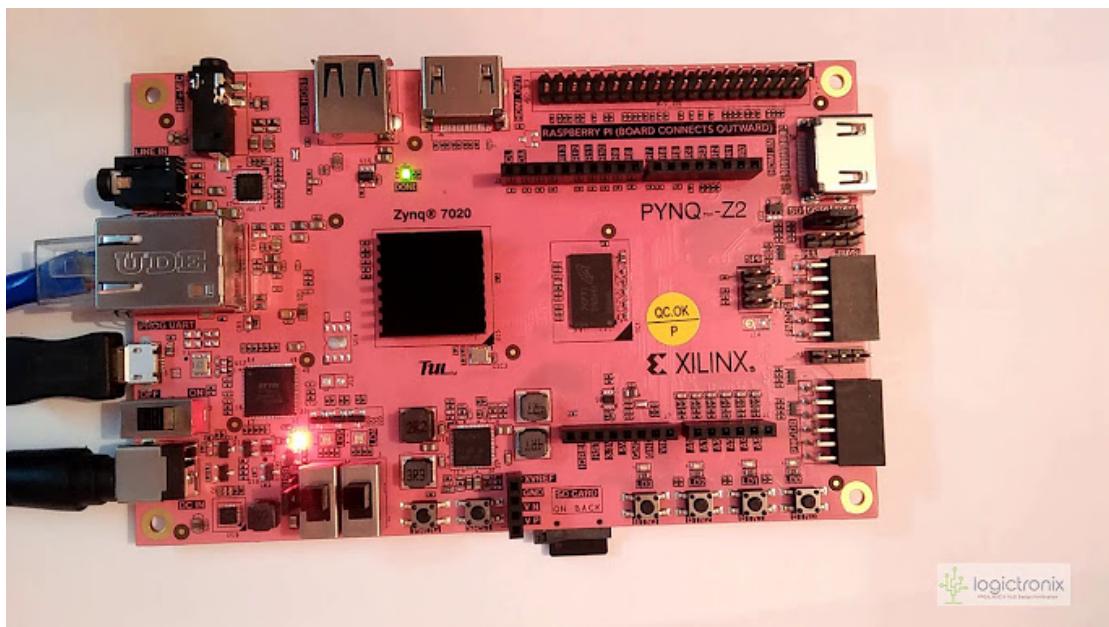
O *Octave* e o *Python* foram selecionados para a simulação de sinais com falha e sem falha, assim como para a implementação dos modelos de pré-processamento FHF e F, devido à sua facilidade de uso, extensas bibliotecas científicas, e capacidade de processamento de dados eficiente.

O *Octave* é uma alternativa de código aberto ao *Matlab*, permitindo realizar análises de sinal e processamento de dados com alto desempenho. *Python*, por sua vez, oferece bibliotecas robustas como *NumPy* e *SciPy*, que facilitam a simulação e análise de sinais.

Em termos de *hardware*, foi escolhido a utilização da plataforma *SoC FPGA Pynq Z2*, ver em 3.3.

A placa *PYNQ* é uma plataforma de desenvolvimento baseada no *SoC Zynq-7000 XC7Z020-1CLG400C*, que oferece uma variedade de interfaces de I/O, memória, *switches* e *LEDs*, portas de expansão e monitoramento de energia. As interfaces de I/O incluem circuito de programação *USB-JTAG*, *USB OTG* 2.0, ponte *USB-UART*, *Ethernet* de 10/100/1G, entrada *HDMI*, saída *HDMI*, interface *I2S* com *DAC* de 24 bits e conector *TRRS* de 3,5 mm, e entrada de linha com conector de 3,5 mm.

Figura 3.3 – Placa de Desenvolvimento PYNQ-Z2



Fonte: <https://www.youtube.com/playlist?app=desktop&list=PLL7OaZvVVz4UY0FF-SZZ3RwXXT6q9XJu5> [acessado 3 Jul, 2024]

A memória consiste em 512 MB de *DDR3* com barramento de 16 bits @ 1050

Mbps, 128 Mbit de *Flash Quad-SPI* e conector para cartão *Micro SD*. A placa possui 2 *switches*, 2 *LEDs RGB*, 4 *LEDs* e 4 botões de pressão. Os relógios incluem um de 125 MHz para *PL (Programmable Logic)* e um de 50 MHz para *PS (Processing System)*.

As portas de expansão incluem 2 portas *Pmod*, 16 I/O totais do *FPGA* (8 pinos compartilhados com o conector *Raspberry Pi*), 1 conector de *shield Arduino*, 24 I/O totais do *FPGA*, 6 entradas analógicas *single-ended* 0-3,3V para *XADC*, e conector *Raspberry Pi* com 28 I/O totais do *FPGA* (8 pinos compartilhados com a porta *Pmod A*). O monitoramento de energia abrange o monitoramento ativo das correntes e voltagens da fonte de alimentação.

Dessa forma, o *software Vivado* foi escolhido como a ferramenta de desenvolvimento de *hardware* devido ao seu suporte completo para design, simulação e implementação de sistemas baseados em *FPGA*, permitindo uma síntese eficiente e uma depuração detalhada do circuito projetado.

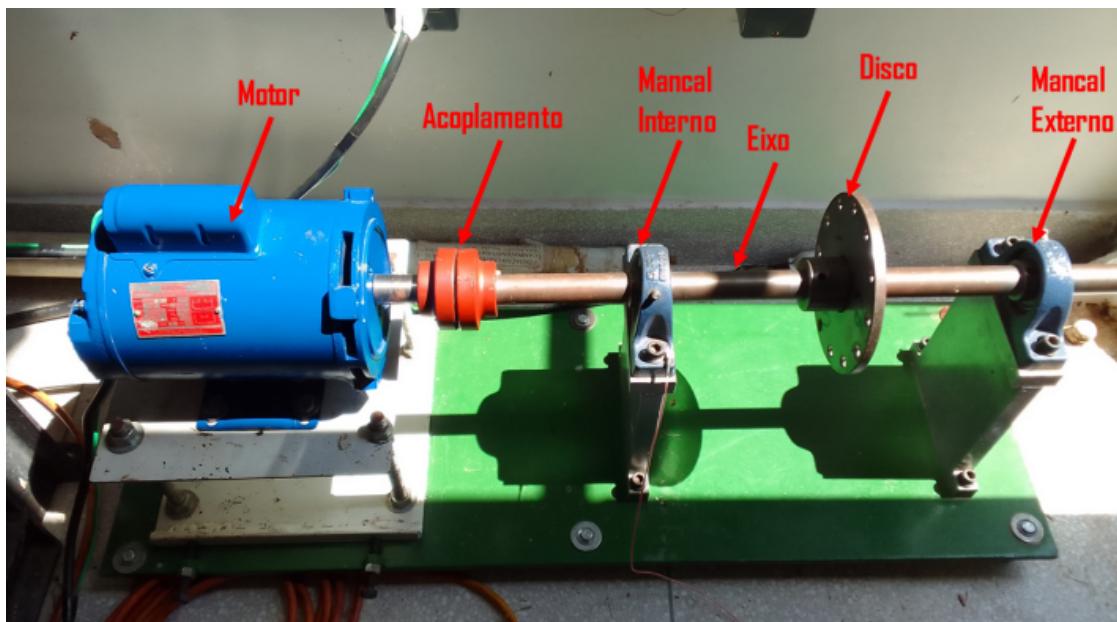
### 3.7 Planejamento Experimental

No desenvolvimento deste projeto, era de interesse trabalhar com dados reais, uma vez que isso possibilita uma análise mais precisa e representativa das condições de operação de máquinas em situações reais. No entanto, devido à dificuldade em encontrar dados reais de uma caixa de engrenagens, foi utilizado uma bancada experimental como alternativa. A bancada está localizada no LabNVH, no prédio LDTEA da Universidade de Brasília, Campus Gama, e possui um conjunto de equipamentos adequado para a simulação de falhas em sistemas mecânicos, apresentado na Fig. 3.4.

A bancada consiste em um motor de 3520 RPM montado sobre uma plataforma, utilizado para simular falhas de desalinhamento e desbalanceamento. O eixo do motor é acoplado a outro eixo sustentado por mancais, com um disco entre eles para simular o desbalanceamento, o que pode ser observado na Fig. 3.4. Antes de realizar as aquisições dos dados, a máquina passa por um processo de alinhamento para garantir que esteja em condições ideais de operação, evitando que falhas indesejadas interfiram nos sinais adquiridos (Araújo, 2023b).

Uma vez o alinhamento feito, são coletados sinais de vibração em duas condições: uma balanceada (sem a adição de massa) e outra com a simulação de falhas, como o desbalanceamento, introduzindo uma massa desbalanceadora no volante.

Figura 3.4 – Bancada didática para análise de vibração.



Fonte: ([Araújo, 2023b](#)).

## 4 Resultados

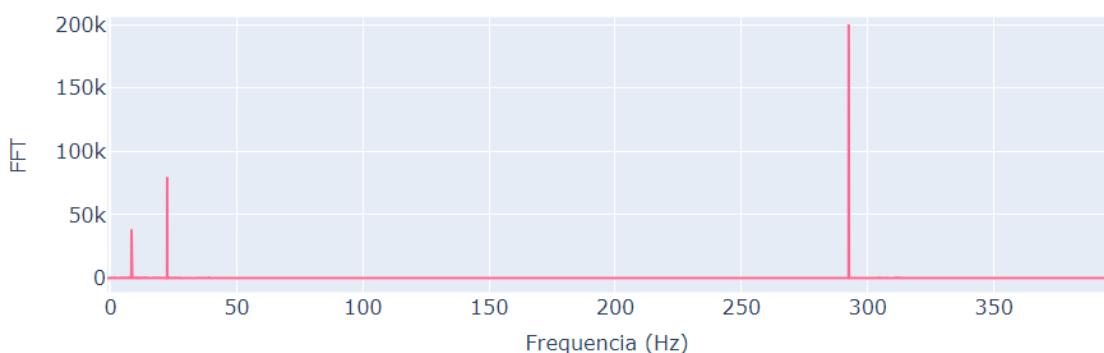
### 4.1 Resultados da cadeia de processamento de sinais

Para uma melhor visualização dos efeitos do pré-processamento de sinais do *modelo F*, um código em *Python* foi desenvolvido utilizando a biblioteca *plotly.graph\_objects* para permitir uma análise visual mais detalhada, conforme mostrado nas Figuras 4.2, 4.1, 4.3, 4.4 e 4.5. Nesse processo, é possível observar as diferenças nas componentes de frequência entre sinais com falha e sem falha. Essa distinção é essencial para que o classificador compreenda claramente como diferenciá-los. (Solanki, 2014)

#### 4.1.1 Modelo F

Ao aplicar o modelo F, que consiste na aplicação de uma Transformada Rápida de *Fourier* no sinal, foi observado um aparecimento de bandas laterais nas proximidades da frequência da caixa de engrenagens para o sinal em que foi introduzida a falha de desbalanceamento, onde podemos visualizar em 4.2 em comparação com a figura que possui o sinal da caixa de engrenagens saudável 4.1, nessa figura os dados não foram normalizados e o eixo y é indicado como as intensidades de cada componente de frequência.

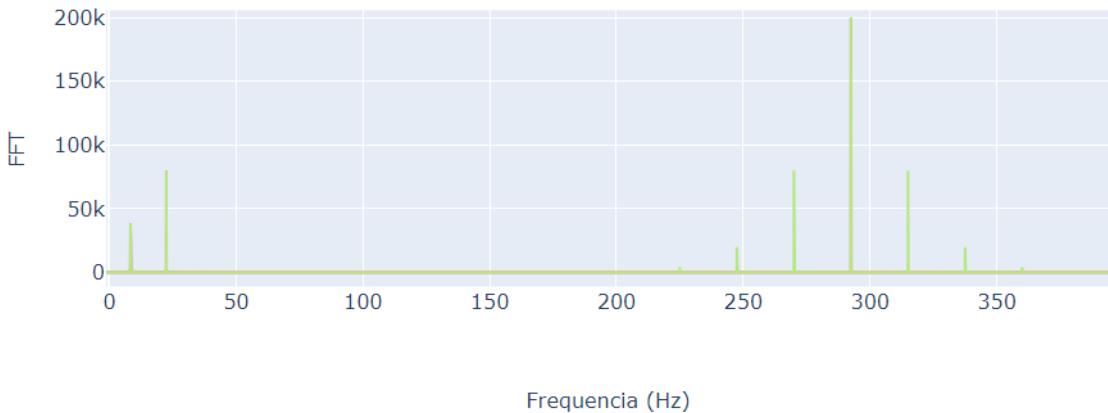
Figura 4.1 – FFT do Sinal sem falha após o filtro Butterworth.



Fonte: Autora

Esse aparecimento de bandas laterais indica a presença de componentes de frequência adicionais que não estão presentes em um sinal sem falhas. A identificação dessas alterações nas componentes de frequência pode ser extremamente útil para a construção de um modelo de predição de falhas, pois permite ao classificador distinguir entre sinais normais e aqueles que indicam problemas de desbalanceamento na

Figura 4.2 – FFT do Sinal com falha após o filtro Butterworth.



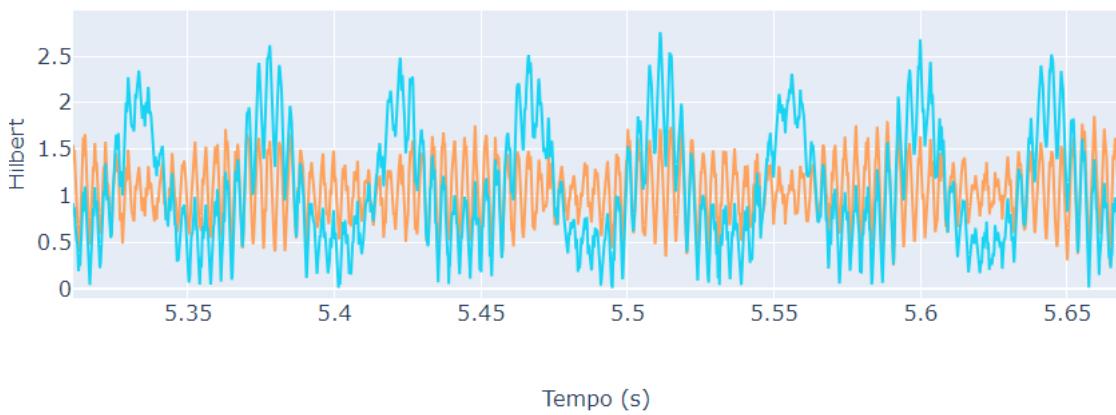
Fonte: Autora

caixa de engrenagens.

#### 4.1.2 Modelo FHF

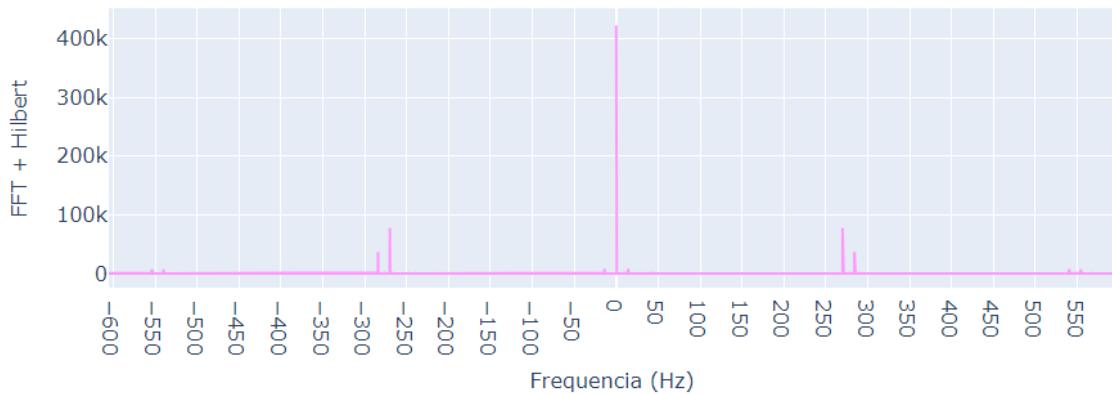
O modelo FHF, que consiste em aplicar uma Transformada de *Hilbert* 4.3 para obter o sinal envelope e, em seguida, realizar uma Transformada Rápida de *Fourier* nesse sinal. Nesse modelo, além do surgimento das bandas laterais características, foi possível observar um espalhamento na base da frequência máxima e nas suas bandas laterais, como podemos observar em 4.5 e 4.4. Esse comportamento não apenas destaca as componentes de frequência fundamentais, mas também revela alterações na estrutura do sinal que podem ser importantes para a detecção e análise de falhas.

Figura 4.3 – Sinal apó a Transformada de Hilbert, sendo o azul com falha e o sinal laranja sem falha.



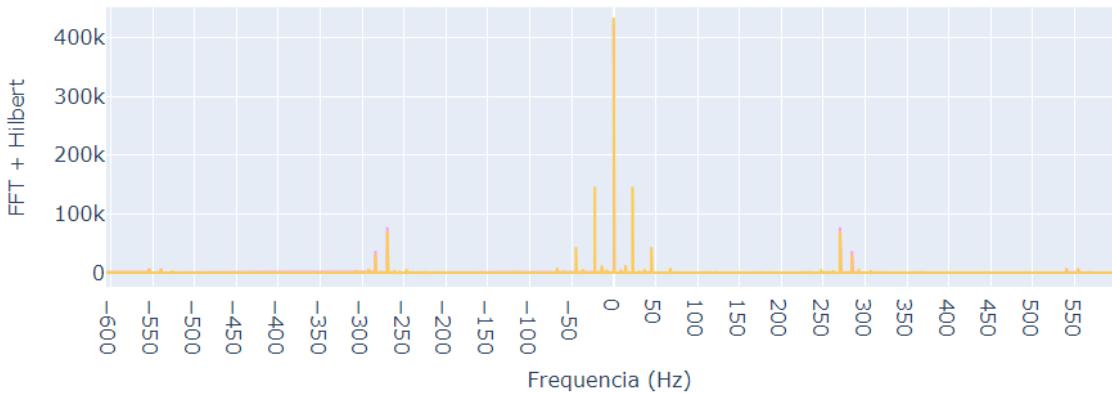
Fonte: Autora

Figura 4.4 – Sinal sem falha após a Transformada de Hilbert e a FFT



Fonte: Autora

Figura 4.5 – Sinal com falha após a Transformada de Hilbert e a FFT



Fonte: Autora

## 4.2 Resultados do modelo de classificação

Os valores de frequência e intensidade de ambos os espectros, tanto do modelo FHF quanto do modelo F, foram exportados para análise posterior. Após a exportação desses dados, foi criada uma base de dados consolidada contendo essas informações. Essa base de dados foi então importada para o *software WEKA*, onde foi utilizado para treinar um modelo de árvore de decisão J48 ([Solanki, 2014](#)) ([Meena; Choudhary, 2017](#)). O objetivo desse treinamento foi desenvolver um classificador capaz de identificar falhas com base nas características de frequência e intensidade dos sinais analisados. O uso do modelo J48 permitiu a criação de um classificador, aproveitando os dados detalhados fornecidos pelos modelos FHF e F. Os dados utilizados para esse treinamentos foram os dados simulados de uma caixa de engrenagens.

Inicialmente, foi realizado um teste com ambas as bases de dados utilizando os parâmetros padrão do modelo J48 no *WEKA*. No entanto, com esses parâmetros padrão, o modelo não apresentou uma boa acurácia. Após essa constatação, foram realizados testes alternando os parâmetros do modelo J48 até encontrarmos os mais

adequados para a base de dados em questão. Os parâmetros otimizados que proporcionaram uma melhor performance do modelo estão dispostos na 4.1. Foi utilizado também uma porcentagem de teste de 10% e uma porcentagem de treinamento de 90%.

Tabela 4.1 – Configurações para o Modelo J48 no Weka

Configurations	Setting	Description
binarySplits	False	Se deve usar divisões binárias em atributos nominais ao construir as árvores ( <a href="#">Solanki, 2014</a> ).
confidenceFactor	0.25	O fator de confiança usado para poda (valores menores implicam em mais poda) ( <a href="#">Solanki, 2014</a> ).
minNumObj	7700	O número mínimo de instâncias por folha ( <a href="#">Solanki, 2014</a> ).

#### 4.2.0.1 Pelo Modelo F

Após realizar o teste com a base de dados gerada pelo modelo F, constatou-se que a acurácia obtida foi muito baixa perto de 50%. Isso sugere uma dificuldade significativa em distinguir entre os diferentes padrões de falha. Nesse teste foram classificados corretamente 59.98% e foram classificados incorretos 50.02%. Por essa razão o modelo F foi provado como insuficiente para essa aplicação.

#### 4.2.0.2 Pelo Modelo FHF

Após realizar o mesmo teste com a base de dados gerada pelo modelo FHF, observou-se uma melhora significativa nos resultados. O modelo apresentou uma acurácia de 85.16%, indicando uma capacidade maior de distinguir entre os diferentes padrões de falha em comparação com o modelo F. Essa melhoria na acurácia sugere que a abordagem de combinar a transformada de *Hilbert* com a *FFT* proporcionou uma representação mais informativa dos dados, permitindo uma classificação mais precisa das falhas. Isso demonstra a importância de considerar não apenas a transformação do sinal bruto, mas também seu envelope, para uma análise mais completa e eficaz.

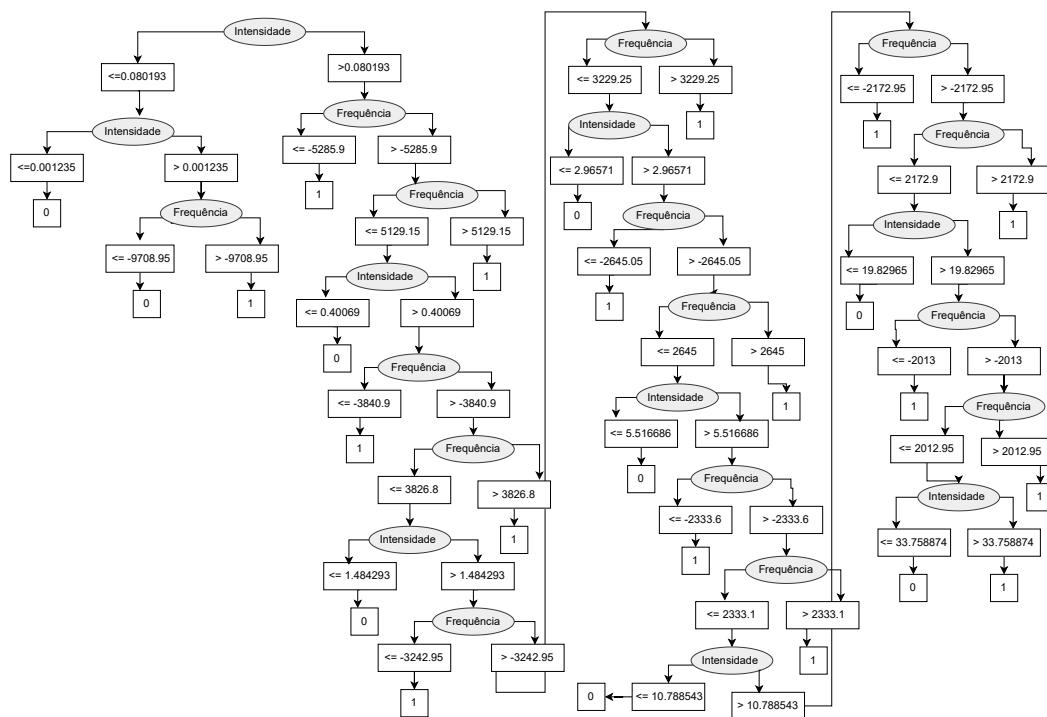
Tabela 4.2 – Accuracy using the obtained decision tree for the *FHF* preprocessing model.

Correctly Classified	85.1612%
Incorrectly Classified	14.8387%

Abaixo está a imagem 4.6 que descreve a árvore de decisão gerada pelo *Weka* e também a matriz de confusão 4.7. Nela, é possível observar como ocorre a classificação

dos dados de entrada e os critérios utilizados para essa classificação. Os critérios de avaliação foram a frequência da componente e sua intensidade, que determinam os caminhos seguidos na árvore para atribuir uma classe específica a cada instância. O resultado 0 corresponde à classe do sinal sem falha, enquanto o resultado 1 representa o sinal com falha. A árvore de decisão é uma representação visual do processo de tomada de decisão do modelo J48, permitindo uma compreensão intuitiva das regras que governam a classificação dos dados. Cada nó interno da árvore representa uma decisão baseada em um atributo específico, enquanto os nós folha representam as classes de destino ou as conclusões finais. Essa visualização facilita a interpretação do modelo e a identificação dos principais fatores que influenciam na classificação das instâncias.

Figura 4.6 – Arvore de Decisão gerada pelo Weka



Fonte: Autora

Através dos testes realizados, foi identificado que a acurácia do modelo FHF possui uma dependência significativa com a frequência de corte do filtro *Butterworth*. Especificamente, ao utilizar uma frequência de corte (FC) de 1000 Hz, houve um aumento substancial na acurácia do modelo, alcançando 90%. Isso indica que a escolha adequada da frequência de corte pode otimizar o desempenho do modelo, melhorando a capacidade de detecção de falhas.

Além disso, também foi observado um incremento na acurácia quando a base de dados incluiu tanto os valores positivos quanto os negativos do espectro de

Figura 4.7 – Matriz de Confusão

<b>==== Matriz de Confusão ====</b>			
		<b>a</b>	<b>b</b>
		<-- Classificado como	
31792	8224		<b>a = 0</b>
3647	36337		<b>b = 1</b>

Fonte: Software Weka

frequência. Esses resultados destacam a importância de ajustar os parâmetros de filtragem e a composição da base de dados para maximizar a eficácia na predição de falhas em sistemas de engrenagens e rolamentos.

#### 4.2.1 Implementação do Modelo J48 na Esp32

Além disso, utilizamos o modelo de árvore de decisão J48, gerado pelo *software Weka*, e o transformamos em sentenças condicionais ‘if’ e ‘else’ na linguagem C, adequando-o para a implementação na *ESP32* via *IDE do Arduino*, como apresentado em 4.8. A tradução direta das regras da árvore de decisão preservou a lógica de classificação original, mapeando a estrutura da árvore para uma sequência de condições que conduzem à decisão final. No código do *Arduino*, foram adicionados dois conjuntos de dados, um representando um estado falho e outro um estado saudável, baseados em valores de intensidade e frequência.

Os testes de desempenho revelaram um tempo de execução de 50 microsegundos, demonstrando a rapidez da *ESP32* em processar as condições. O consumo de memória foi de 205594 *bytes* (15%) do espaço de armazenamento disponível para programas e 13416 *bytes* (4%) da memória dinâmica para variáveis globais, deixando 314264 *bytes* disponíveis para variáveis locais. Esses resultados confirmam a viabilidade e eficiência da implementação do modelo J48 na *ESP32*.

Esses testes foram realizados com o intuito de identificar o tempo de execução e o consumo de recursos da *ESP32*. Ao medir o tempo necessário para processar as condições do modelo J48 e avaliar o uso de memória.

Figura 4.8 – Resultado da classificação de 2 valores de Dados Falho e Saudável



```
A máquina está falha.  
A máquina está saudável.  
Tempo de execução: 50 microssegundos  
  
A máquina está falha.  
A máquina está saudável.  
Tempo de execução: 50 microssegundos
```

Fonte: Autora

### 4.3 Aquisição de dados reais

Neste trabalho, optou-se por utilizar bases de dados previamente coletadas, uma vez que a calibração da bancada experimental apresentou dificuldades devido à carência de instrumentos adequados no laboratório.

Essa limitação motivou a utilização das bases de dados obtidas por Lucas Araújo em seu TCC ([Araújo, 2023b](#)), no qual foram coletadas quatro bases de dados relacionadas a desbalanceamentos rotativos. Essas bases consistem em sinais de vibração obtidos com massas desbalanceadoras de 3g, 2g, 8g e 20g, totalizando cinco conjuntos de dados, com a base de dados balanceada.

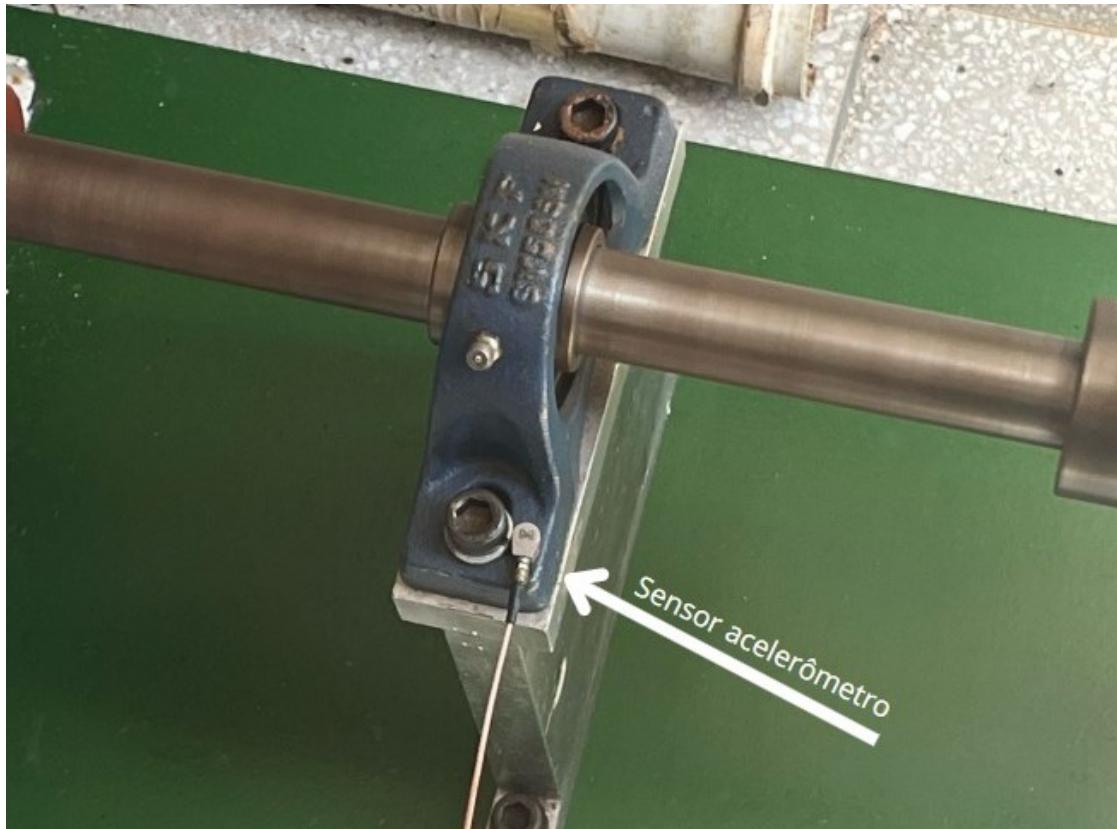
Para a aquisição dos sinais, foi utilizado um acelerômetro da marca Endevco Isotron modelo 10901, com sensibilidade de 104.9 volts/g, instalado no mancal interno, apresentado na figura [4.9](#). A placa de aquisição empregada foi a National Instruments modelo 9234, configurada com uma taxa de amostragem de 2048 Hz e um total de 20480 amostras por aquisição. A coleta dos sinais foi realizada por meio do software LabVIEW, uma ferramenta amplamente utilizada para monitoração, simulação, controle de sistemas, aquisição e processamento de sinais ([Azevedo, 2017](#)).

O diagrama utilizado para a aquisição e armazenamento dos dados gerou arquivos no formato .txt, nos quais a primeira coluna representa a aceleração em m/s<sup>2</sup>, e a segunda coluna corresponde ao tempo em segundos. Dessa forma, os dados utilizados neste trabalho apresentam essa estrutura, facilitando a análise e o processamento subsequente. É possível observar o sinal de aceleração da bancada após um filtro Butterworth de FC = 1000 Hz, apresentado na Fig. [4.10](#). E também o sinal após uma transformada de Hilbert, apresentado na Fig. [4.11](#).

#### 4.3.1 Manipulação de dados

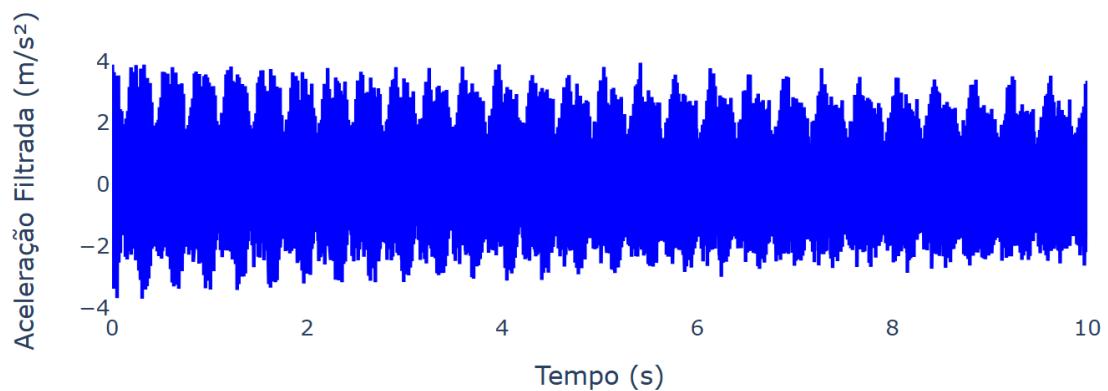
Nesta etapa, foram realizados procedimentos para o tratamento e análise dos dados coletados. Inicialmente, destaca-se que apenas os três primeiros picos do

Figura 4.9 – Acelerômetro posicionado no mancal interno.



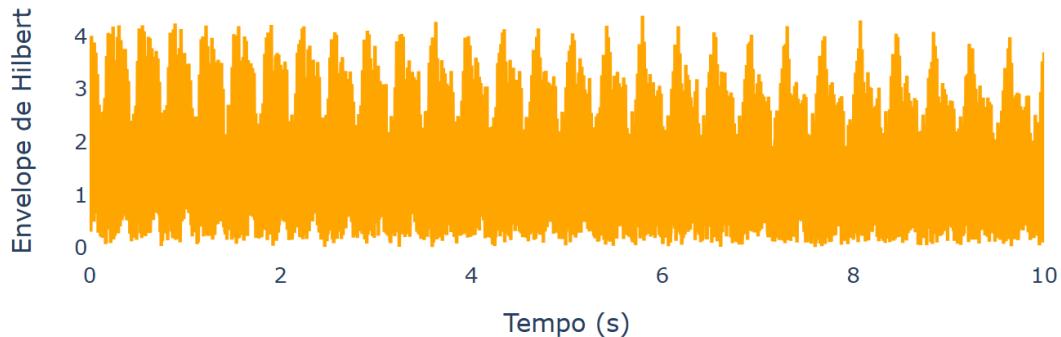
Fonte: Autora

Figura 4.10 – Sinal de Aceleração dos dados reais em domínio do tempo com filtro de butterworth.



Fonte: Autora

Figura 4.11 – Sinal de Aceleração dos dados reais em domínio do tempo após uma Transformada de Hilbert.



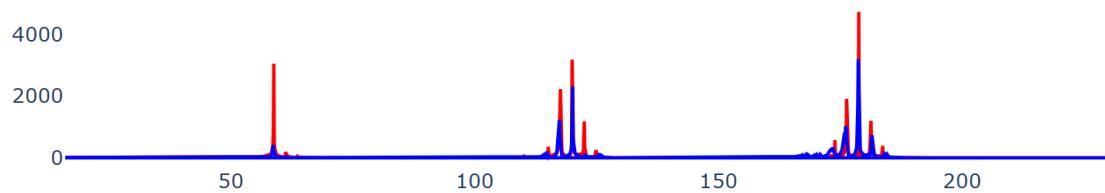
Fonte: Autora

espectro de frequência são suficientes para identificar a falha de desalinhamento, uma vez que esses harmônicos carregam as informações mais relevantes sobre o comportamento dinâmico do sistema.

Portanto, o sinal passou por um filtro passa-banda entre 58,6 Hz e 176,1 Hz, abrangendo do primeiro ao terceiro harmônico. Foram aplicados dois filtros passa-banda específicos: um entre 58,8 Hz e 117,3 Hz, e outro entre 117,5 Hz e 176 Hz. Essa filtragem foi realizada com o intuito de isolar as entradas do sistema, garantindo que frequências não essenciais não influenciem no cálculo do valor RMS, como está representado nas figuras 4.12 e 4.13. Além disso, essa abordagem contribui para eliminar outros tipos de falhas, como, por exemplo, defeitos na pista de rolamento (Araújo, 2023b).

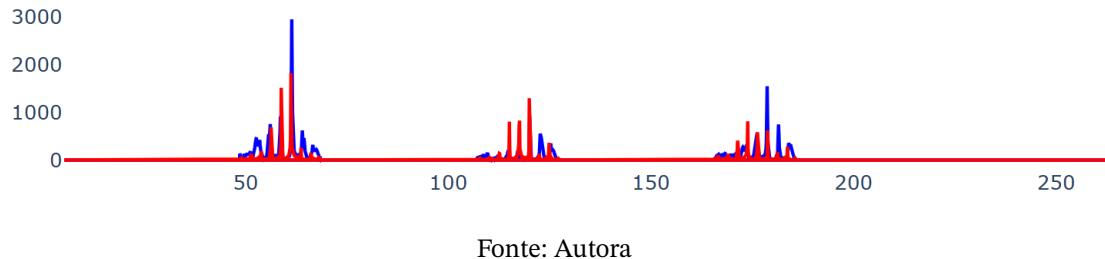
Todos esses processamentos foram implementados em Python, utilizando bibliotecas como NumPy, Plotly, SciPy e outras ferramentas específicas para análise de sinais e visualização de dados.

Figura 4.12 – Sinal no Domínio da Frequência do Modelo F após a Filtragem das Frequências Relevantes. Sendo o Sinal Vermelho o Sinal com Falha e o Azul o Sinal Sem Falha



Fonte: Autora

Figura 4.13 – Sinal no Domínio da Frequência do Modelo FHF após a Filtragem das Frequências Relevantes. Sendo o Sinal Vermelho o Sinal com Falha e o Azul o Sinal Sem Falha



Fonte: Autora

#### 4.3.2 Testes com Arvore de decisão

Inicialmente, foram realizados testes com o modelo de Árvore de Decisão utilizando os mesmos parâmetros que haviam sido aplicados aos dados simulados. No entanto, esses parâmetros não se mostraram eficazes para os dados reais, resultando em uma acurácia insatisfatória.

Diante desse resultado, optou-se por ajustar os parâmetros do modelo e testar outras variações de algoritmos baseados em árvores, como *Random Forest*, *Random Tree* e *REP Tree*, na tentativa de melhorar o desempenho. Além disso, foram explorados outros modelos de inteligência artificial que pudessem ser implementados em hardware, como *SMO* (Sequential Minimal Optimization) e *Simple Logistic*. Contudo, nenhum desses modelos alcançou uma acurácia superior a 60%, o que indicou a necessidade de uma abordagem alternativa para o problema.

Uma possível explicação para a diferença de desempenho entre os dados simulados e reais pode estar relacionada ao posicionamento do sensor. Nos dados simulados, o acelerômetro foi posicionado na carcaça da caixa de engrenagens, enquanto, nos dados reais, o sensor foi instalado no mancal interno. Essa diferença de posicionamento pode ter afetado significativamente os sinais adquiridos, uma vez que o mancal interno está sujeito a vibrações e ruídos diferentes em comparação com a carcaça da caixa de engrenagens. Como resultado, os sinais reais podem apresentar características mais complexas e menos previsíveis, o que dificultou a obtenção de resultados consistentes com os modelos testados.

Algumas possíveis soluções foram tentadas para melhorar a qualidade dos sinais e, consequentemente, o desempenho dos modelos. Entre elas, destacam-se o aumento da resolução do sinal por meio do processo de *padding* e a redução da resolução para simplificar os dados. No entanto, nenhuma dessas abordagens resultou em uma melhoria significativa na acurácia dos modelos.

Esses resultados evidenciaram que os dados reais possuem características mais complexas e desafiadoras em comparação com os dados simulados, exigindo,

portanto, técnicas mais robustas ou um pré-processamento mais refinado para obter um desempenho satisfatório.

#### 4.3.3 Classificação pelo RMS

Diante das dificuldades encontradas com os modelos de inteligência artificial, buscou-se outras formas de classificar os dados de falha. Uma abordagem alternativa foi a classificação com base no valor RMS do sinal de vibração.

Quanto maior o valor RMS do sinal, maior é o nível de desalinhamento ou desbalanceamento da máquina. Isso ocorre porque o RMS captura a energia total do sinal, e, em sistemas desbalanceados, as amplitudes das vibrações aumentam, resultando em um valor RMS mais elevado, como apresentado na figura 4.12. No entanto, no modelo FHF o valor de RMS se mostrou muito parecido entre o falho e o saudável. E por esse motivo outros parâmetros foram analisados para distinguir esses dados.

Além do RMS, outros parâmetros foram analisados para distinguir os dados saudáveis dos falhos, como o EQM (*Erro Quadrático Médio*) e o MAE (*Mean Absolute Error*). O EQM mede a média dos quadrados das diferenças entre os valores observados e os valores esperados, sendo sensível a grandes variações no sinal. Já o MAE calcula a média das diferenças absolutas, fornecendo uma medida mais robusta em relação a outliers.

E notou-se que, embora visualmente não seja possível distinguir claramente os sinais saudáveis e defeituosos do modelo FHF (Figura 4.13), por meio dos parâmetros de RMS, é possível observar que as bases de dados do modelo FHF apresentam um MAE mais elevado, indicando que os dados são mais distintos entre si, como é possível observar na tabela da figura 4.14.

A Figura 4.14 apresenta as bases de dados dispostas em colunas, sendo elas: 2g, 3g, 8g e 05\_20g. Esses valores correspondem aos diferentes pesos adicionados à bancada experimental para induzir um desbalanceamento no sistema. Na segunda coluna, são mostradas as respectivas bases de dados associadas a cada peso, com as seguintes descrições:

- **Original F:** refere-se à base de dados original de cada peso, a qual passou apenas pelo modelo de processamento F;
- **Original FHF:** é a base de dados original de cada peso, mas que passou pelo modelo de pré-processamento FHF.
- **Deslocado F:** engloba as bases de dados que foram geradas a partir de pequenos deslocamentos aleatórios aplicados às originais, com o objetivo de simular o comportamento de cada base e gerar dados adicionais para o cálculo dos

índices de desempenho, como o EQM e o MAE, a fim de avaliar as respostas do sistema. E esses dados são referentes ao processamento pelo modelo F.

- **Deslocado FHF:** Base de dados com pequenos deslocamentos que passou pelo modelo FHF.

Figura 4.14 – Tabela de parâmetros de Classificação

Parâmetros de Classificação	Base de Dados	2g	3g	8g	05_20g
EQM	Original F	3137,1519	1892,09	3266,139	5195,25
	Original FHF	2340,5293	2814,32	2515,406	3298,016
	Deslocado F	3130,439	1858,823	3224,754	5217,206
	Deslocado FHF	2310,888	2870,0551	2478,11	3257,6506
MAE	Original F	4,091	3,701	4,4305	5,4541
	Original FHF	5,649	6,409	5,92182	7,0464
	Deslocado F	4,082	3,6899	4,4555	5,472
	Deslocado FHF	5,6381	6,4706	5,963	7,0072

Fonte: Autora

#### 4.3.4 Criação de novas base de dados

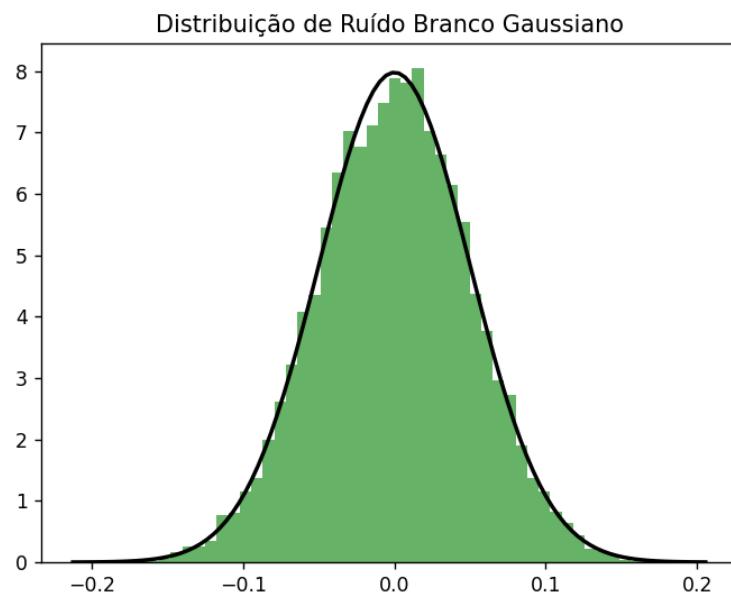
Devido ao número limitado de bases de dados disponíveis, foi necessário gerar novas bases a partir das existentes para aumentar a diversidade e a robustez dos dados utilizados nos testes. Essas novas bases foram criadas por meio da adição de ruído gaussiano aos sinais originais, uma técnica comum em processamento de sinais para simular variações e imperfeições que podem ocorrer em condições reais.

O ruído gaussiano, também conhecido como ruído aleatório, é um tipo de ruído que segue uma distribuição normal, apresentado na Fig. 4.15. Ele é caracterizado por apresentar valores aleatórios que se distribuem simetricamente em torno de uma média, com uma variância específica. A adição desse ruído aos sinais originais permite simular condições mais realistas, como interferências externas, variações no ambiente de medição ou imperfeições no equipamento de aquisição.

Foram geradas bases de dados para duas condições distintas: uma máquina saudável e uma máquina com falha de desbalanceamento. Para a última condição, os dados foram processados seguindo dois modelos distintos: o modelo F e o modelo FHF.

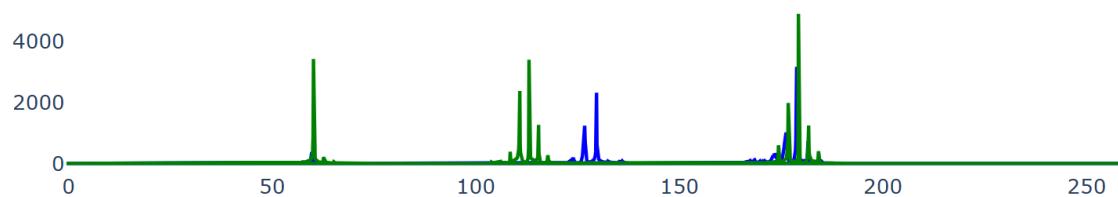
O processo de geração das novas bases de dados foi implementado em Python. A função `random` da biblioteca `numpy` foi utilizada para gerar números aleatórios com distribuição normal, que foram aplicados como fatores de deslocamento nas

Figura 4.15 – Ruido Gaussiano.



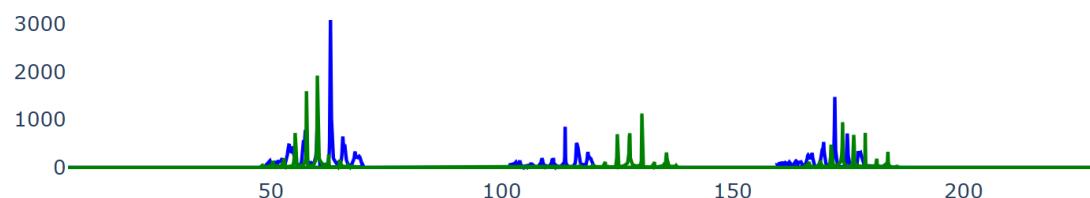
Fonte: Autora

Figura 4.16 – Base de Dados Gerada para o Modelo F. Sendo o Falho o Sinal Verde e o Sem falha o Sinal Azul



Fonte: Autora

Figura 4.17 – Base de Dados Gerada para o Modelo FHF. Sendo o Falho o Sinal Verde e o Sem falha o Sinal Azul



Fonte: Autora

frequências e amplitudes dos sinais. O percentual máximo utilizado para deslocar a curva em frequência foi de 5%, enquanto o percentual máximo de deslocamento da intensidade da frequência foi de 10%. Essa escolha foi baseada no fato de que a falha de desbalanceamento se manifesta principalmente na amplitude da intensidade de frequência, e não no deslocamento em frequência. Deslocamentos significativos em frequência estão mais associados a outras falhas em máquinas rotativas, como folgas mecânicas.

As novas bases foram salvas em arquivos de texto, contendo as frequências e amplitudes deslocadas, para posterior análise e utilização, como pode ser visto nas figuras 4.16. 4.17. No entanto, não foram testadas em software ou na implementação de hardware.

#### 4.3.5 Configuração do IP-FFT

A configuração do IP da FFT da Xilinx envolve duas etapas principais: a configuração inicial no Vivado Block Design, onde os parâmetros básicos são definidos, e a configuração dinâmica durante a operação, realizada através do sinal `s_axis_config_tdata`. Ambas as etapas são essenciais para o funcionamento correto do IP.

##### 4.3.5.1 Configuração Inicial no Vivado Block Design

A configuração inicial do IP da FFT é realizada no Vivado Block Design, onde os parâmetros básicos são ajustados através de uma interface gráfica. As duas imagens fornecidas mostram diferentes aspectos dessa configuração, que são detalhados a seguir.

###### 4.3.5.1.1 Configuração Básica

Na primeira imagem 4.18, são definidos os parâmetros básicos do IP da FFT:

- **Número de Canais:** O IP foi configurado para operar com um único canal (Number of Channels = 1).
- **Tamanho da Transformada:** O tamanho da transformada foi definido como 32768 pontos (Transform Length = 32768), o que significa que o IP realizará uma FFT de 32768 pontos.
- **Frequência do Clock:** A frequência do clock foi configurada para 100 MHz (Target Clock Frequency = 100 MHz).
- **Taxa de Dados:** A taxa de dados foi definida como 50 MSPS (Mega Samples Por Segundo) (Target Data Throughput = 50 MSPS).

- **Arquitetura:** A arquitetura escolhida foi a Pipelined, Streaming I/O, que permite processamento contínuo de dados com alta taxa de transferência.
- **Transformada Configurável em Tempo de Execução:** A opção Run Time Configurable Transform Length foi habilitada, permitindo que o tamanho da transformada seja alterado dinamicamente durante a operação do sistema.

#### 4.3.5.1.2 Configuração Implementação

Na segunda imagem 4.19, são mostradas configurações mais detalhadas:

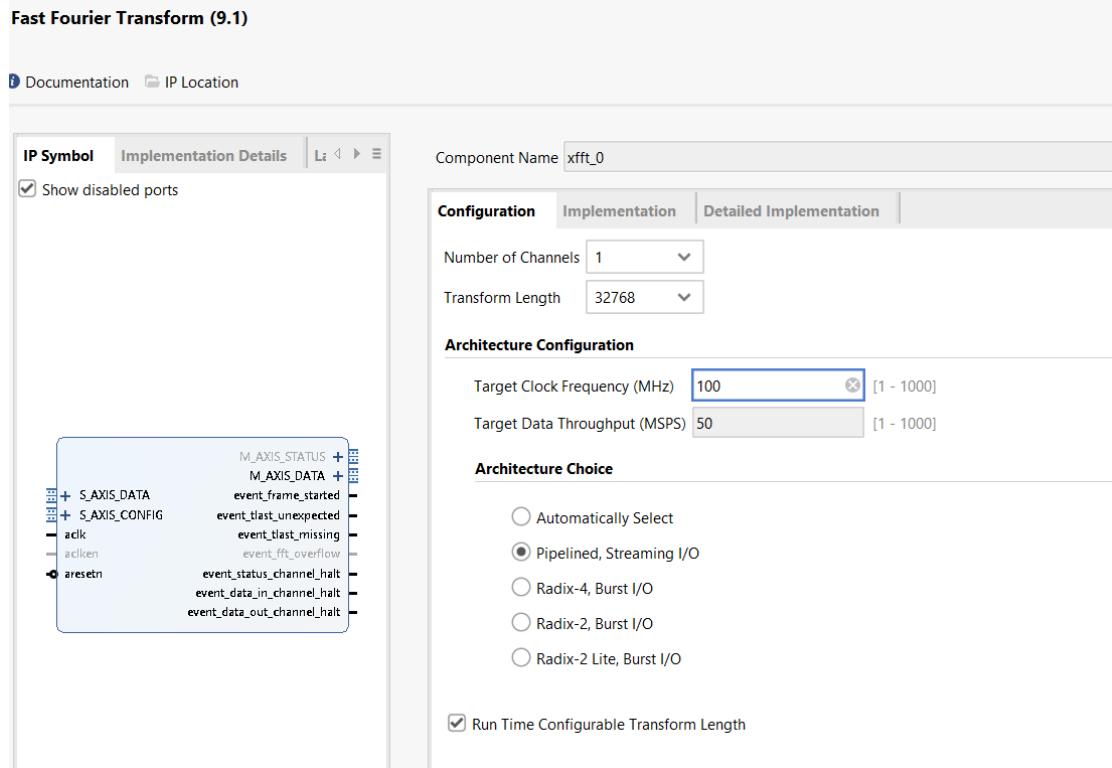
- **Formato dos Dados:** O IP foi configurado para operar com dados em formato de ponto fixo (Data Format = Fixed Point).
- **Opções de Escalonamento:** Foi selecionada a opção Scaled, o que significa que o IP aplicará um escalonamento aos dados durante o processamento para evitar overflow.
- **Modo de Arredondamento:** O modo de arredondamento foi definido como Truncation, onde os bits menos significativos são simplesmente truncados após cada estágio da FFT.
- **Precisão dos Dados:** A largura dos dados de entrada e dos fatores de fase foi definida como 16 bits (Input Data Width = 16 e Phase Factor Width = 16).
- **Sinais de Controle:** O sinal de reset assíncrono (ARESETn) foi habilitado, e é necessário que ele seja assertado por pelo menos 2 ciclos de clock para garantir um reset adequado.
- **Ordem de Saída:** A ordem de saída dos dados foi definida como Bit/Digit Reversed Order, o que significa que os dados de saída serão fornecidos em ordem bit-reversa.
- **Campos Opcionais de Saída:** O campo OVFL0 foi habilitado, o que permite que o IP indique a ocorrência de overflow durante o processamento. O campo XK\_INDEX não foi habilitado.
- **Esquema de Throttle:** Foi selecionado o modo Non Real Time, que permite maior flexibilidade no tempo de processamento, em contraste com o modo Real Time, que exige que os dados sejam processados em tempo real.

#### 4.3.5.2 Configuração Dinâmica via s\_axis\_config\_tdata

Além da configuração inicial no Vivado Block Design, o IP da FFT permite ajustes dinâmicos durante a operação através do sinal s\_axis\_config\_tdata. Esse sinal é enviado via interface AXI4-Stream e permite configurar parâmetros como o

tamanho da transformada, a direção da FFT (direta ou inversa), o escalonamento e o comprimento do prefixo cíclico.

Figura 4.18 – Configuração do IP - FFT



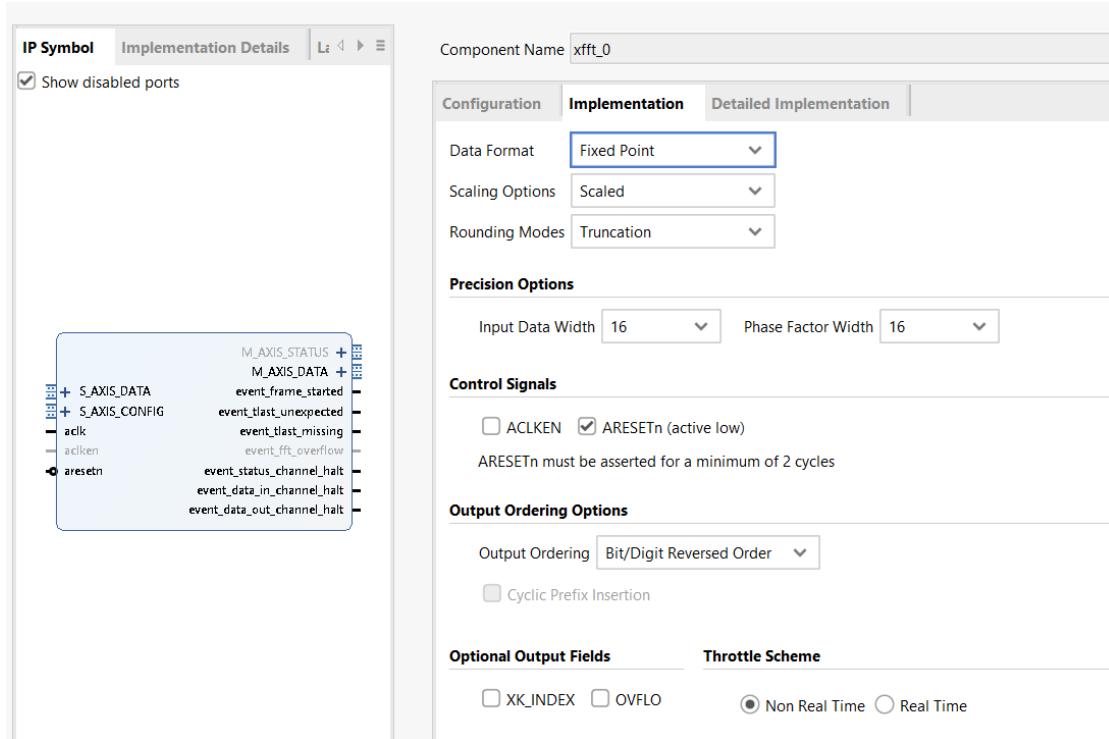
Fonte: Autora

#### 4.3.5.2.1 Estrutura do s\_axis\_config\_tdata

O vetor `s_axis_config_tdata` é composto por vários campos, cada um responsável por configurar um aspecto específico do IP da FFT, como apresentado na Fig. 4.20 . Nos testes deste projeto foi adicionado o valor de 10101010101010100001110, ao sinal de configuração onde :

- **NFFT (5 bits)**: Define o tamanho da transformada. O valor é dado como  $\log_2(N)$ , onde  $N$  é o número de pontos da FFT. No exemplo fornecido, os bits 00011 correspondem a uma FFT de 8 pontos.
- **CP\_LEN (10 bits)**: Especifica o comprimento do prefixo cíclico, que é utilizado em sistemas de comunicação para evitar interferência entre símbolos. No exemplo, os bits 1010101010 correspondem a um prefixo cíclico de 682 amostras.
- **FWD\_INV (1 bit)**: Define a direção da transformada. Um valor 1 indica uma FFT direta (transformada de Fourier), enquanto um valor 0 indica uma FFT inversa (IFFT). No exemplo, o bit 1 indica que a FFT será direta.

Figura 4.19 – Configuração da implementação do IP - FFT



Fonte: Autora

- **SCALE\_SCH (10 bits):** Especifica o escalonamento aplicado em cada estágio da FFT. Esse campo é relevante apenas quando o IP está configurado para usar aritmética escalonada. Cada par de bits define o fator de escalonamento para um estágio da FFT, onde 00 significa nenhum escalonamento, 01 significa divisão por 2, 10 significa divisão por 4 e 11 significa divisão por 8. No exemplo, os bits 101010 indicam um escalonamento de 10 (divisão por 4) para os três primeiros estágios.

Figura 4.20 – Canal de Configuração

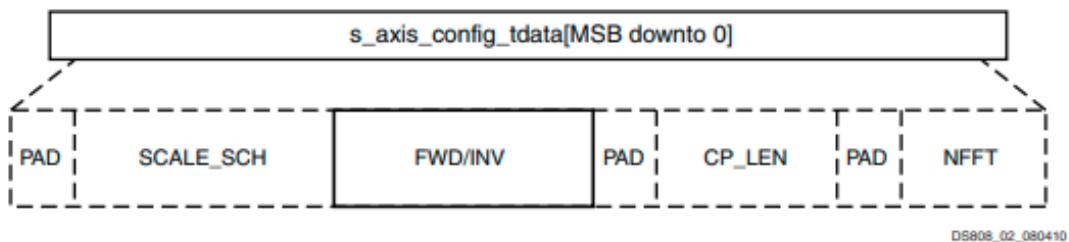


Figure 3-2: Configuration Channel TDATA (s\_axis\_config\_tdata) Format

Fonte: Autora

### 4.3.6 Implementação na Pynq

A implementação do sistema na placa Pynq envolveu a integração de vários componentes de hardware, visando processar os sinais de vibração. O IP da FFT utilizado recebe dados no protocolo AXI4-Stream, enquanto o IP de memória disponibilizado pela Xilinx opera com o protocolo AXI-Lite. Essa diferença de protocolos exigiu a criação de uma solução para a comunicação entre os módulos.

Para resolver essa incompatibilidade, foi desenvolvida uma memória dentro do Vivado, que lê os dados de um arquivo .mem e os envia para o IP da FFT por meio de uma máquina de estados. Essa máquina de estados controla o fluxo de dados, garantindo que os valores sejam enviados de forma correta e sincronizada ao IP da FFT, os códigos podem ser encontrados no GITHUB em ([Schulz, 2023](#)).

#### 4.3.6.1 Memória

Como falado anteriormente, criou-se uma memoria que enviasse os dados através de uma maquina de estados. A máquina de estados foi implementada no módulo datasrc, que opera com cinco estados principais:

1. **s0**: Estado inicial, onde o sistema aguarda o sinal de start para iniciar a leitura e envio dos dados.
2. **s1**: Estado de leitura e preparação dos dados, onde o próximo dado é carregado da memória e o endereço é incrementado.
3. **s2**: Estado de envio dos dados, onde o dado atual é enviado ao IP da FFT se o sinal tready estiver ativo. Caso contrário, o sistema transita para o estado **s3**.
4. **s3**: Estado de espera, onde o sistema aguarda que o sinal tready seja ativado para retomar o envio dos dados.
5. **s4**: Estado que identifica o fim da memória e retorna para o estado **s0**.

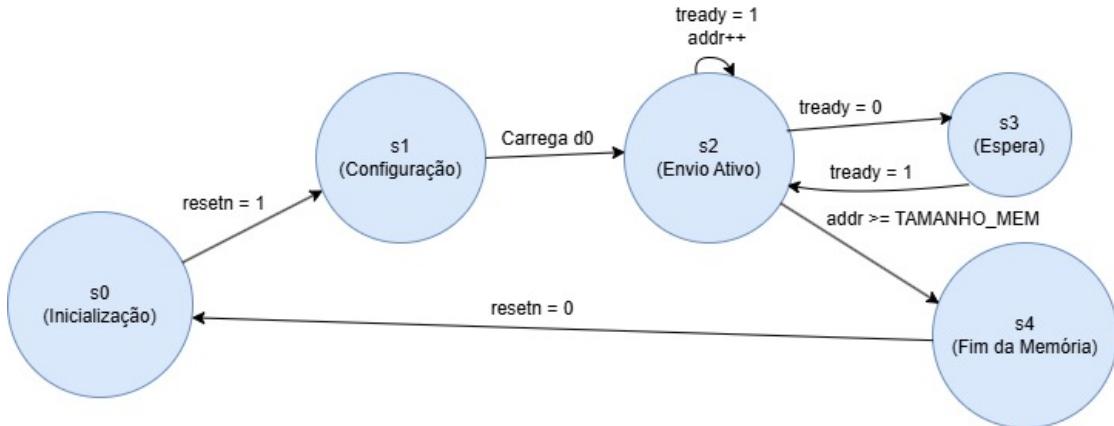
Além disso, o sinal tlast é ativado quando o último dado da memória é enviado, indicando o fim do processo de transmissão, como apresentado na Fig.4.21.

#### 4.3.6.2 Detector de falha

O detector de falha é a principal funcionalidade do sistema, responsável por identificar anomalias nas máquinas rotativas com base nos dados da Transformada Rápida de Fourier para o modelo F. A operação do detector de falha é realizada dentro de um processo sensível ao sinal de clk e ao sinal de resetn, garantindo que o sistema seja reinicializado adequadamente quando necessário.

A detecção de falhas é baseada na análise dos primeiros 3 picos de frequência presentes no sinal de vibração. Por meio da simulação no Vivado, é possível identificar

Figura 4.21 – Maquina de Estados para envio de dados.



Fonte: Autora

o período de tempo em que esses picos ocorrem. Assim, a ideia principal é implementar um contador que, ao atingir o tempo correspondente aos 3 picos, comece a capturar os valores de pico da FFT dentro de uma banda de 20 dB. Esses valores são então comparados com um limiar pré-determinado. Se os valores estiverem acima do limiar, isso indica a presença de uma falha. Caso contrário, o sistema considera que não há falha.

#### 4.3.7 top\_module

O código do módulo `top_module` descreve a arquitetura de um sistema. O módulo se comunica com componentes de entrada e saída de dados, incluindo a interface com o módulo `datasrc` para captura de dados, o `xfft_0` para realizar a FFT, e o `faulty_detection` para identificar falhas, como apresentado na Fig. 4.22.

##### 4.3.7.1 Componentes Principais

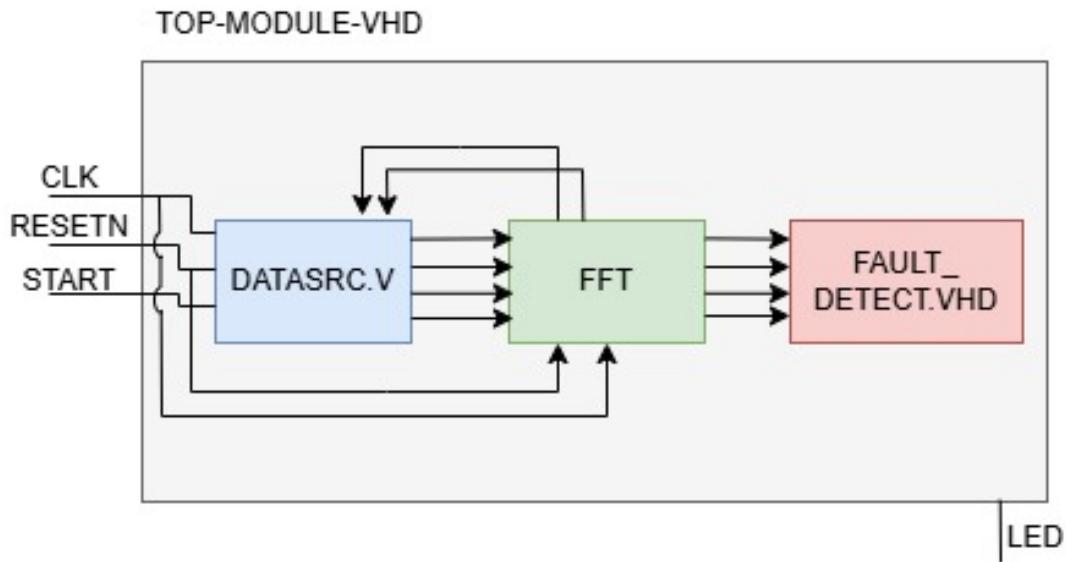
- **datasrc:** Memoria de dados.
- **xfft\_0:** Ip da FFT.
- **faulty\_detection:** Algoritimo de detecção de falha.
- **ila\_0:** Componente para depuração.

O processo interno (`process(clk, resetn)`) lida com a configuração da FFT. Quando o sinal `start` é ativado e as condições de configuração estão atendidas, a configuração da FFT é enviada para o componente `xfft_0`.

Durante o processamento, o `s_axis_data_tready_int` controla a disponibilidade de dados de entrada para o componente FFT, enquanto o `m_axis_data_tdata` contém os dados de saída da FFT. O componente de detecção de falhas então monitora

esses dados e ativa o led\_fault caso um pico nos dados seja detectado, indicando uma possível falha.

Figura 4.22 – Top Module do sistema.



Fonte: Autora

#### 4.3.7.2 Block Design

Além dos componentes de memória, da FFT e do identificador de falha, foi adicionado um ILA (Integrated Logic Analyzer) ao projeto. O ILA é uma ferramenta de depuração embarcada que permite monitorar sinais internos do FPGA em tempo real, facilitando a identificação de problemas durante a execução do sistema.

Foram incluídos também blocos de constantes no projeto, com valores pré-definidos para configurar parâmetros específicos do IP da FFT, como tamanhos de dados e limites de operação.

Outro componente adicionado foi o clk\_wiz, um bloco de gerenciamento de clock que permite gerar diferentes frequências de operação para os módulos do sistema. Isso foi necessário para garantir que todos os componentes operassem em sincronia, especialmente considerando as diferentes taxas de processamento exigidas pelo IP da FFT e pela memória personalizada.

Também foi incluído um bloco de reset verification, que garante que o sistema seja reinicializado corretamente sempre que necessário, evitando estados inconsistentes durante a operação.

Após a implementação dos componentes no block design, foram realizadas a síntese e a implementação do sistema, o que possibilitou a análise do consumo de energia. O sistema apresentou um consumo de energia de 75% para a parte dinâmica

e 25% para a parte estática, onde é possível ver na figura 4.29. Além disso, ao analisar a utilização dos recursos, verificou-se, conforme mostrado na Figura 4.28, que as BRAMs foram utilizadas em 95,71%. Na Figura 4.27, é possível observar a implementação no do sistema, onde as diferentes áreas são indicadas por cores: a parte amarela representa a FFT, a laranja corresponde ao bloco de reset, a rosa indica o ILA core, a verde representa a memória implementada, e o azul escuro refere-se ao bloco Clock Wise.

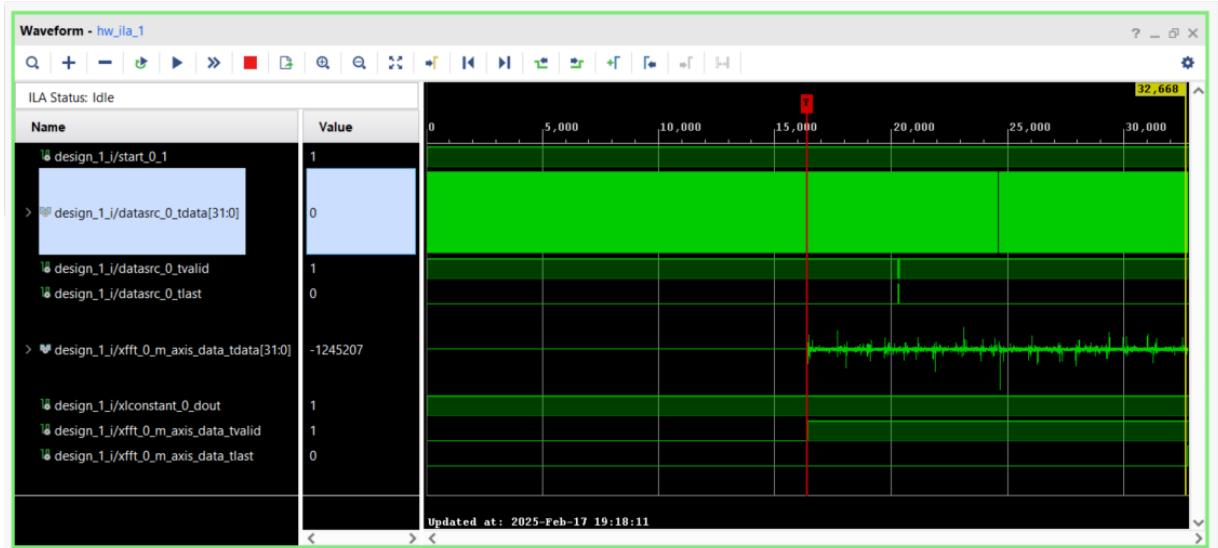
As entradas do sistema foram definidas como start, reset e clk. O sinal start inicia o processamento dos dados, o reset reinicializa o sistema, e o clk fornece o clock necessário para a sincronização de todos os módulos.

Foram realizadas simulações no Vivado para validar o funcionamento do sistema. Essas simulações incluíram testes de envio de dados da memória personalizada para o IP da FFT, verificação da máquina de estados e análise do comportamento do sistema em diferentes cenários de operação. Os resultados das simulações confirmaram que o sistema era capaz de processar os dados corretamente, mas também revelaram desafios na interpretação dos dados pelo IP da FFT, principalmente devido à complexidade dos sinais de vibração e à necessidade de ajustes finos nos parâmetros de configuração.

Apesar de o sistema estar enviando os dados corretamente para o IP da FFT, houve desafios na obtenção da curva FFT correta na saída. A FFT não está processando os dados adequadamente, o que resulta em curvas que não correspondem à realidade. As figuras 4.23 e 4.24 mostram as curvas da FFT, porém essas não refletem a resposta esperada. A curva real foi gerada utilizando Python e é representada na Figura 4.26.

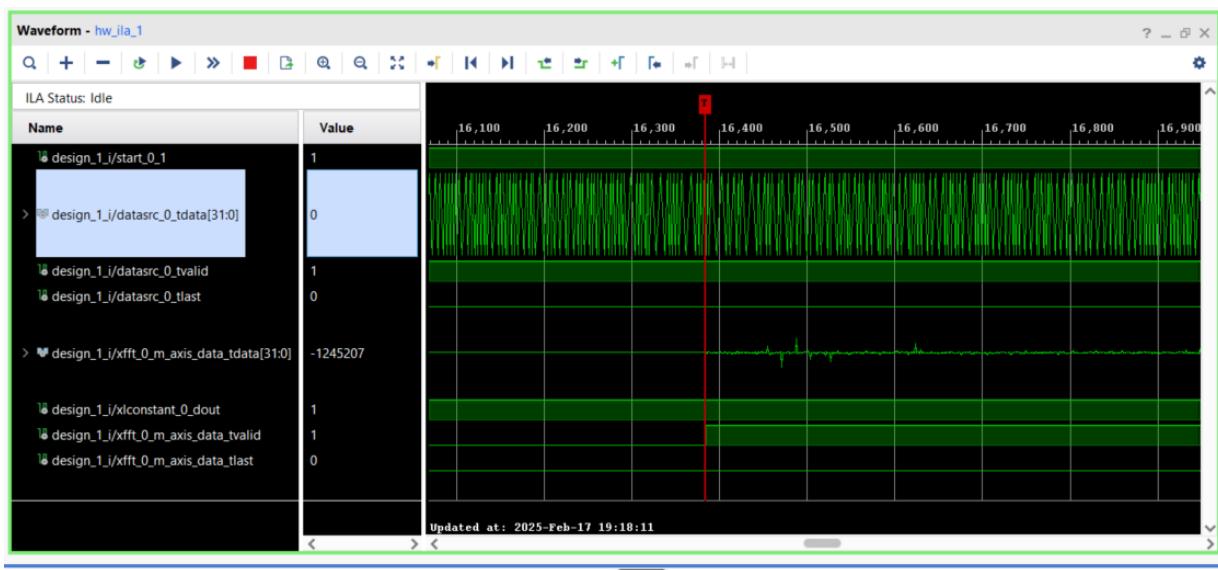
Esse problema pode estar relacionado ao fato de que o IP da FFT espera tanto dados reais quanto imaginários, o que pode estar causando uma interpretação incorreta dos dados processados. Para tentar contornar esse problema, foi testado definir os valores imaginários como zero, mas isso não resolveu a situação. Outra abordagem foi configurar os valores imaginários para serem iguais aos valores reais, mas essa tentativa também não teve sucesso. Embora a análise tenha mostrado essa possível causa, o tempo disponível não foi suficiente para corrigir esse problema. A investigação e ajuste desse comportamento do IP da FFT para garantir o processamento correto dos dados será uma área de foco para futuras melhorias no sistema.

Figura 4.23 – Teste do IP - FFT com a PYNQ Z2.



Fonte: Autora

Figura 4.24 – Teste do IP - FFT com a PYNQ Z2 com zoom.



Fonte: Autora

Figura 4.25 – Block Design do sistema.

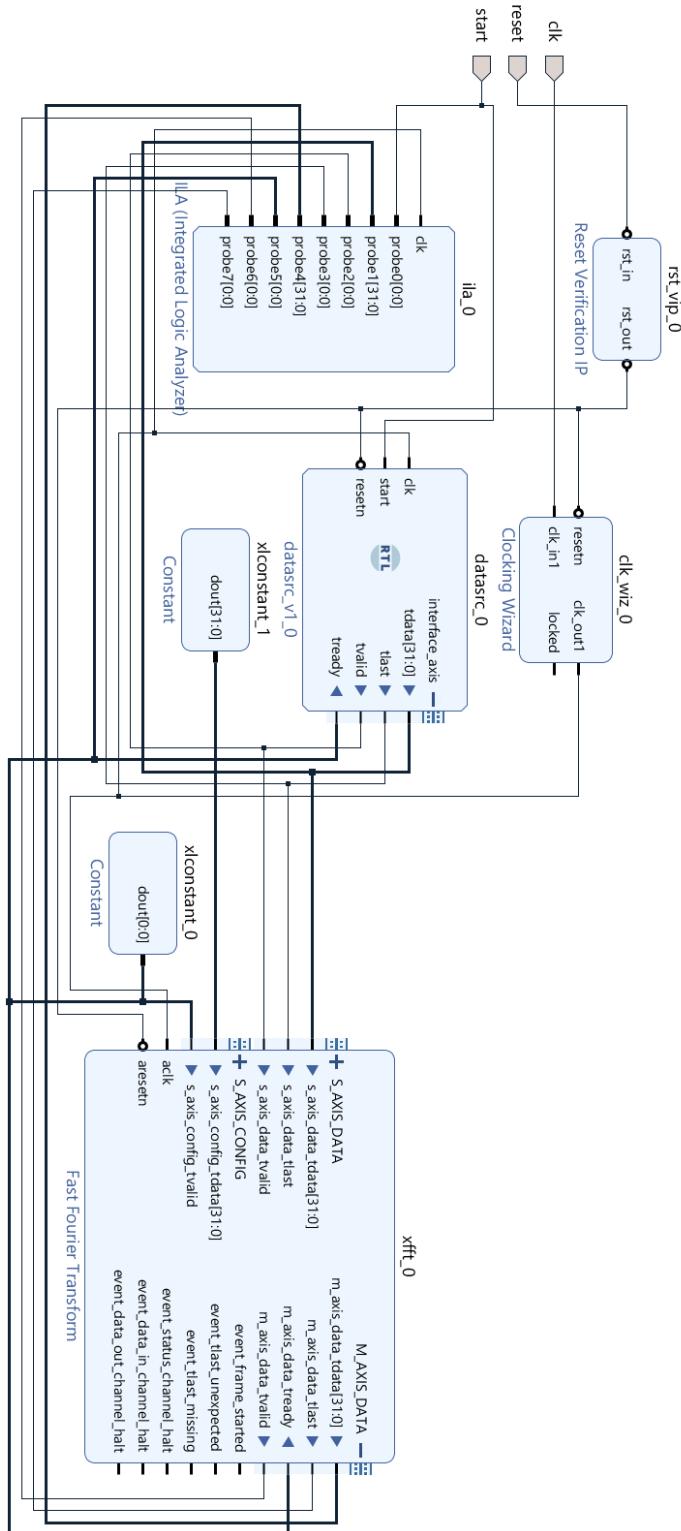
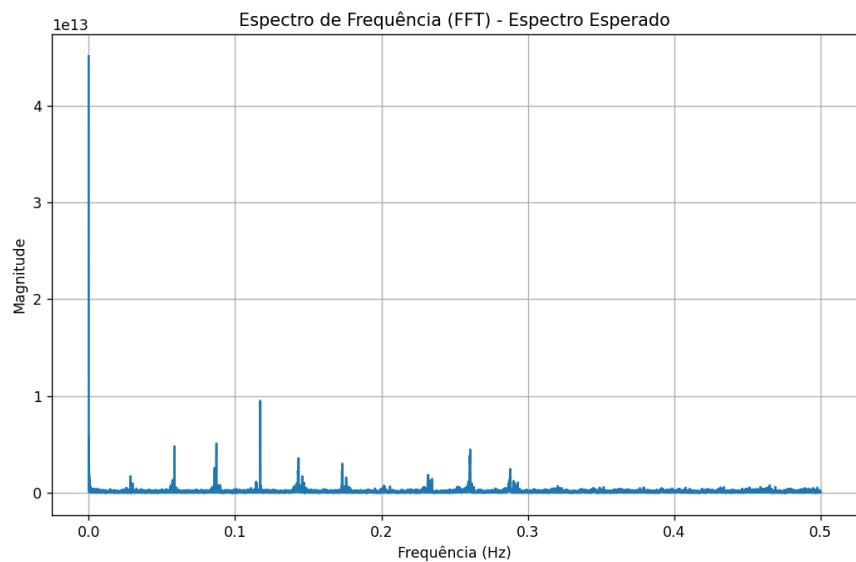
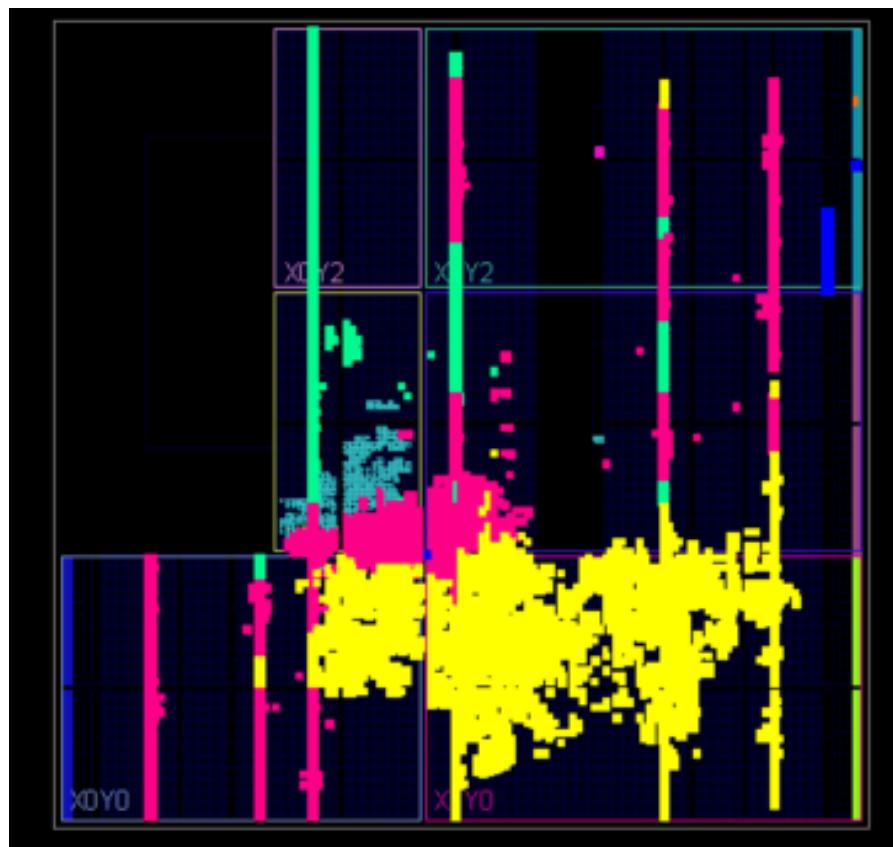


Figura 4.26 – Espectro esperado na saída do IP - FFT



Fonte: Autora

Figura 4.27 – Netlist e Consumo de Recursos



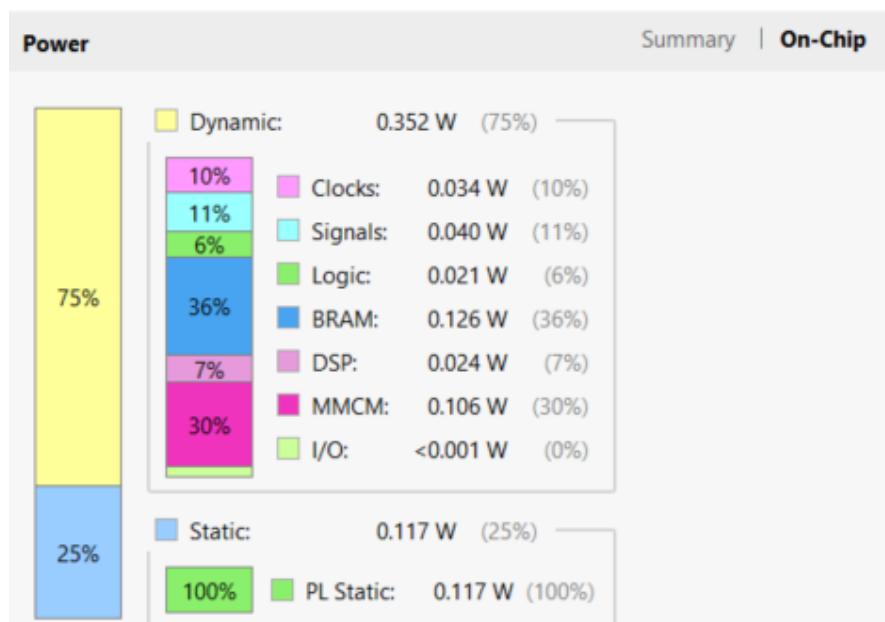
Fonte: Autora

Figura 4.28 – Utilização de Recursos Pós-Implementação

Utilization		Post-Synthesis   Post-Implementation	
		Graph   Table	
Resource	Utilization	Available	Utilization %
LUT	6182	53200	11.62
LUTRAM	1290	17400	7.41
FF	9668	106400	9.09
BRAM	134	140	95.71
DSP	21	220	9.55
IO	3	125	2.40
BUFG	3	32	9.38
MMCM	1	4	25.00

Fonte: Autora

Figura 4.29 – Consumo de energia



Fonte: Autora

## 5 Conclusões

Este trabalho teve como objetivo principal avaliar métodos de pré-processamento e classificação de sinais de vibração para a detecção de falhas em sistemas de engrenagens e rolamentos. Através da aplicação de técnicas como a Transformada Rápida de Fourier e a Transformada de Hilbert, foi possível identificar características distintas nos sinais de vibração que permitem diferenciar entre condições normais e falhas de desbalanceamento. A análise visual dos sinais simulados, realizada com o auxílio de ferramentas como a biblioteca em Python, evidenciou a presença de bandas laterais e alterações nas componentes de frequência, que são indicativos claros de falhas.

O modelo F, baseado na FFT, mostrou-se limitado em termos de acurácia, atingindo apenas 50% na classificação de falhas. Por outro lado, o modelo FHF, que combina a Transformada de Hilbert com a FFT, apresentou resultados significativamente melhores, com uma acurácia de 85,16%. Essa melhoria demonstra a importância de considerar não apenas a transformação do sinal bruto, mas também o envelope do sinal, para uma análise mais robusta e precisa.

A implementação do classificador J48 no software WEKA permitiu a criação de uma árvore de decisão capaz de distinguir entre sinais saudáveis e falhos com base nas características de frequência e intensidade. A otimização dos parâmetros do modelo J48 foi crucial para alcançar uma acurácia satisfatória, destacando a importância de ajustes finos no processo de classificação. Além disso, a implementação do modelo J48 na ESP32 demonstrou a viabilidade de utilizar hardware embarcado para a detecção de falhas em tempo real, com um tempo de execução de apenas 50 microsegundos e um consumo de memória compatível com as limitações do dispositivo.

Já análise de dados reais, obtidos a partir de uma bancada experimental, revelou desafios adicionais, como a complexidade dos sinais e a influência do posicionamento do sensor na qualidade dos dados. Ao trabalhar com os dados reais, o modelo de árvore de decisão não apresentou resultados satisfatórios, não conseguindo atingir uma acurácia maior que 60%, independentemente do processamento de sinais realizado anteriormente. Diante dessa limitação, foi encontrada uma forma mais simples e igualmente eficaz de classificar os dados de vibração de máquinas: a métrica do valor RMS. Essa abordagem mostrou-se eficiente, pois quanto maior o valor RMS, maior a probabilidade de o sinal indicar uma falha. Além disso, essa solução é mais simples de ser implementada em um FPGA, dispensando a complexidade de modelos de aprendizado de máquina.

Decidiu-se também implementar o processamento do sinal diretamente no FPGA, utilizando o IP da FFT da Xilinx. No entanto, durante a implementação,

foram encontrados problemas no processamento dos dados pela FFT. Embora o IP esteja recebendo os dados corretamente, ele não está processando os sinais de forma adequada, resultando em curvas de FFT que não correspondem ao esperado. Infelizmente, devido ao tempo limitado, não foi possível corrigir esse problema. Para trabalhos futuros, está planejado o ajuste do IP da FFT para garantir o processamento correto dos dados, bem como a adição da Transformada de Hilbert para uma análise mais completa dos sinais.

## 5.1 Cronograma de Atividades

O cronograma de atividades inicialmente planejado foi impactado por desafios imprevistos durante o desenvolvimento do projeto. Embora a execução das tarefas tenha seguido um ritmo adequado no início, o surgimento de problemas com as novas bases de dados resultou em um atraso significativo. Esses problemas envolveram questões relacionadas à grande diferença entre os dados reais e os simulados. O que exigiu ajustes e novas abordagens de processamento e análise. Como consequência, as etapas de testes e validação dos modelos sofreram alterações, resultando em um desajuste em relação ao cronograma original. Apesar disso, esforços foram feitos para mitigar os impactos desses imprevistos.

## Referências

- ABDELKRIM, H.; OTHMAN, S. B.; SAOUD, S. B. Reconfigurable soc fpga based: Overview and trends. In: **2017 International Conference on Advanced Systems and Electric Technologies (IC<sub>A</sub>SET)**. [S.l.: s.n.], 2017. p.378 – 383. Citado na p. 32.
- ADAMSAB, K.; SHIVAKUMAR, S. Vibration analysis techniques for rotating machinery and its effect on bearing faults. **Procedia Manufacturing**, v. 20, p. 247–252, 01 2018. Citado nas pp. 15 e 39.
- ADMIRAL, T. D. Using the 555 integrated circuit in bistable mode as a vibration sensor. In: . [S.l.: s.n.], 2021. Citado na p. 20.
- ALAVI, M.; ALIAGA, S.; MURGA, M. MÁquinas de Estado Finito. **Revista de InvestigaciÃEstudiantil Iluminate**, revbol, v. 8, p. 41 – 57, 11 2016. ISSN 2415-2323. Disponível em: [http://revistasbolivianas.umsa.bo/scielo.php?script=sci\\_arttext&pid=S2415-23232016000100005&nrm=iso](http://revistasbolivianas.umsa.bo/scielo.php?script=sci_arttext&pid=S2415-23232016000100005&nrm=iso). Citado na p. 39.
- AMD (ADVANCED MICRO DEVICES). **Zynq 7000 SoC and 7 Series Devices Memory Interface Solutions v4.2, LogiCORE IP Data Sheet**. [S.l.], 2024. Disponível em: <http://www.amd.com/pt/products/adaptive-socs-and-fpgas/soc/zynq-7000.html>. Disponível em: <https://docs.amd.com/viewer/book-attachment/RHoLPAXaTIdzQbRcacrnnng/2g5oxpYYwDX8lGKGSw0BPg>. Citado na p. 34.
- ANGADI, P.; SHENVI, S. V.; LOKESH, S. Design and simulation of axi4 stream interconnect using verilog. In: **2024 IEEE International Conference on Information Technology, Electronics and Intelligent Communication Systems (ICITEICS)**. [S.l.: s.n.], 2024. p. 1–4. Citado na p. 38.
- ARAÚJO, L. M. C. de. Utilização de lógica fuzzy para detecção do desbalanceamento em sistema rotativo baseado em sinais de vibração. In: . [S.l.: s.n.], 2023. p. 22–81. Citado nas pp. 22, 26 e 27.
- ARAÚJO, L. M. C. de. **Utilização de Lógica Fuzzy para detecção do desbalanceamento em sistema rotativo baseado em sinais de vibração**. Dissertação (TCC) — Universidade de Brasília - UnB, Faculdade UnB Gama - FGA, Brasília, DF, 2023. Orientadora: Profa. Dra. Maria Alzira de Araújo Nunes. Citado nas pp. 48, 49, 56 e 58.
- ARGENTIERI, S.; DANES, P.; SOUERES, P. Modal analysis based beamforming for nearfield or farfield speaker localization in robotics. In: **2006 IEEE/RSJ International Conference on Intelligent Robots and Systems**. [S.l.: s.n.], 2006. p. 866–871. Citado na p. 16.

- AZEVEDO, R. S. **Desenvolvimento de uma ferramenta virtual para detecção e análise de desbalanceamento em simulador de máquina rotativa.** Dissertação (Mestrado), 2017. Citado na p. 56.
- BACCI, D. d. L. C. Aplicação de métodos estatísticos multivariados na avaliação de vibrações sísmicas. *In: . [S.l.: s.n.], 2016.* Citado na p. 20.
- BENGHERBIA, B.; KARA, R.; TOUBAL, A.; ZMIRLI, M. O.; CHADLI, S.; WIRA, P. FPGA implementation of a wireless sensor node with a built-in adaline neural network coprocessor for vibration analysis and fault diagnosis in machine condition monitoring. **Measurement**, v. 163, p. 107960, 2020. ISSN 0263-2241. Disponível em: <https://www.sciencedirect.com/science/article/pii/S026322412030498X>. Citado nas pp. 40 e 41.
- BINIELI, M. **Aprendizagem de Máquina: Uma Introdução ao Erro Quadrático Médio e Linhas de Regressão.** 2025. Acessado em: 17 Janeiro 2025. Disponível em: <https://www.freecodecamp.org/portuguese/news/aprendizagem-da-maquina-uma-introducao-ao-erro-quadratico-medio-e-linhas-de-regressao/>. Citado na p. 32.
- BRITO, R. C. de; MARTENDAL, D. M.; OLIVEIRA, H. E. M. de. Máquinas de estados finitos de mealy e moore. 2003. Citado na p. 39.
- CONTRERAS-MEDINA, L.; ROMERO-TRONCOSO, R.; MILLAN-ALMARAZ, J.; RODRIGUEZ-DONATE, C. Fpga based multiple-channel vibration analyzer embedded system for industrial applications in automatic failure detection. *In: 2008 International Symposium on Industrial Embedded Systems.* [S.l.: s.n.], 2008. p. 229–232. Citado na p. 42.
- DADAFSHAR, S. M. o. T. S. F. A. E. M. Accelerometer and gyroscopes sensors: Operation, sensing, and applications. *In: . [S.l.: s.n.], 2015.* Citado na p. 21.
- DESOUKI, M.; SASSI, S.; RENNO, J.; GOWID, S. A. Dynamic response of a rotating assembly under the coupled effects of misalignment and imbalance. **Shock and Vibration**, v. 2020, n. 1, p. 8819676, 2020. Disponível em: <https://onlinelibrary.wiley.com/doi/abs/10.1155/2020/8819676>. Citado nas pp. 26 e 27.
- DUARTE, N. S. Flexibilidade em sistemas eletrônicos: Implementação dinâmica de funcionalidades em dispositivos de eletrônica programável. **RECIMA21-Revista Científica Multidisciplinar-ISSN 2675-6218**, v. 5, n. 1, p. e585643–e585643, 2024. Citado na p. 18.
- ESPINDOLA, B. M. Desenvolvimento e uso de modulos para processamento de sinais em fpga. **Monografia (Sistemas de Telecomunicações).** Instituto Federal de Santa Catarina. São José, 2011. Citado na p. 16.

- FELDMAN, M. Hilbert transform in vibration analysis. **Mechanical Systems and Signal Processing**, v. 25, n. 3, p. 735–802, 2011. ISSN 0888-3270. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0888327010002542>. Citado na p. 30.
- GAWDE, S.; PATIL, S.; KUMAR, S.; KAMAT, P.; KOTECHA, K. An explainable predictive maintenance strategy for multi-fault diagnosis of rotating machines using multi-sensor data fusion. **Decision Analytics Journal**, v. 10, p. 100425, 2024. ISSN 2772-6622. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2772662224000298>. Citado na p. 41.
- HENKES, F. P.; BRAGA, M.; GIROTTI, G. L.; HENTGES, R.; FRANCO, B. C. Caixa de engrenagens. In: **5ª MOEPEX**. [S.l.: s.n.], 2016. Citado na p. 24.
- IGBA, J.; ALEMZADEH, K.; DURUGBO, C.; EIRIKSSON, E. T. Analysing rms and peak values of vibration signals for condition monitoring of wind turbine gearboxes. **Renewable Energy**, v. 91, p. 90–106, 2016. ISSN 0960-1481. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0960148116300064>. Citado na p. 20.
- IZBICKI, R.; SANTOS, T. M. dos. **Aprendizado de máquina: uma abordagem estatística**. [S.l.]: Rafael Izbicki, 2020. Citado na p. 28.
- JESUS, S. S. d.; CAVALCANTE, P. F. Utilização de bancadas de ensaio para estudo do comportamento dinâmico de máquinas rotativas–vibrações mecânicas. **HOLOS**, Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte, v. 3, p. 18–40, 2011. Citado na p. 15.
- KLIMOVICH, A.; SOLOV'EV, V. Transformation of a mealy finite-state machine into a moore finite-state machine by splitting internal states. **Journal of Computer and Systems Sciences International**, Springer, v. 49, p. 900–908, 2010. Citado na p. 39.
- MARÇAL, R. F. M. **Um método para detectar falhas incipientes em máquinas rotativas baseado em análise de vibrações e lógica fuzzy**. Tese (Doutorado) — Universidade Federal do Rio Grande do Sul (UFRGS), Porto Alegre, Brasil, Dezembro 2000. Tese de Doutorado apresentada ao Programa de Pós-Graduação em Engenharia Metalúrgica, de Minas e dos Materiais (PPGEM). Disponível em: <http://hdl.handle.net/10183/123456>. Citado na p. 15.
- MathWorks. **Vibration Analysis of Rotating Machinery**. 2024. <https://www.mathworks.com/help/signal/ug/vibration-analysis-of-rotating-machinery.html>. Accessed on May 02, 2024. Citado nas pp. 40, 43 e 44.
- MCINERNY SA E DAI, Y. **Processamento básico de sinais de vibração para detecção de falhas em rolamentos**. 2003. Citado nas pp. 15 e 39.

- MEENA, G.; CHOUDHARY, R. R. A review paper on ids classification using kdd 99 and nsl kdd dataset in weka. In: **2017 International Conference on Computer, Communications and Electronics (Comptelix)**. [S.l.: s.n.], 2017. p. 553–558. Citado nas pp. 28 e 52.
- MENEZES, P. A. Análise de vibrações aplicadas à detecção de falhas em roamentos de cubo de roda. 2015. Citado nas pp. 20, 44 e 45.
- MERENDINO, G.; PIERACCI, A.; LANZONI, M.; RICCÒ, B. An embedded system for real time vibration analysis. In: **2011 4th IEEE International Workshop on Advances in Sensors and Interfaces (IWASI)**. [S.l.: s.n.], 2011. p. 6–11. Citado na p. 41.
- MOREIRA, F. C. Sistema de monitoramento de máquina industrial para manutenção preditiva utilizando internet das coisas. 2020. Citado na p. 18.
- ONG, P.; John Koshy, A.; LAI, K. H.; SIA, C. K.; ISMON, M. A deep learning approach for health monitoring in rotating machineries using vibrations and thermal features. **Decision Analytics Journal**, v. 10, p. 100399, 2024. ISSN 2772-6622. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2772662224000031>. Citado nas pp. 40 e 41.
- OPPENHEIM, A.; WILLSKY, A. **Sinais e Sistemas**. Pearson Universidades, 2010. ISBN 9788576055044. Disponível em: <https://books.google.com.br/books?id=ZOg9bwAACAAJ>. Citado nas pp. 29 e 30.
- PASSOS, J. P. d. Estimação da vida em fadiga através de análise dinâmica global-local no domínio da frequência. 2016. Citado na p. 22.
- QIU, H.; LEE, J.; LIN, J.; YU, G. Wavelet filter-based weak signature detection method and its application on rolling element bearing prognostics. **Journal of Sound and Vibration**, v. 289, n. 4, p. 1066–1090, 2006. ISSN 0022-460X. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0022460X0500221X>. Citado nas pp. 40 e 41.
- RODRIGUES, H. Estudo da influência do fator de recobrimento na detecção de falhas em engrenagens por análise de sinais vibratórios. 2018. Citado na p. 25.
- SCHEFFER, C.; GIRDHAR, P. **Practical Machinery Vibration Analysis and Predictive Maintenance**. 1st. ed. [S.l.]: Elsevier, 2004. ISBN 9780750662758. Citado nas pp. 23, 25, 40, 43 e 44.
- SCHNEIDER, P.; XHAFA, F. Chapter 3 - anomaly detection: Concepts and methods. In: SCHNEIDER, P.; XHAFA, F. (Ed.). **Anomaly Detection and Complex Event Processing over IoT Data Streams**. Academic Press, 2022. p. 49–66. ISBN 978-0-12-823818-9. Disponível em: <https://www.sciencedirect.com/science/article/pii/B9780128238189000134>. Citado na p. 31.

- SCHULZ, N. **Repositório do Trabalho de Conclusão de Curso 2 (TCC2)**. 2023. <https://github.com/Nataliaschulz/TCC2>. Acesso em: 25 out. 2023. Citado na p. 67.
- SETIAWAN, E.; ADIONO, T. Design of axi4-stream based modulator ip core for visible light communication system-on-chip. **JURNAL INFOTELInformatics - Telecommunication -Electronics**, 2018. Citado na p. 38.
- SETTE, H.; FILHO, S. N. **Transformada de Hilbert-Fundamentos**. 2017. Citado na p. 30.
- SOLANKI, A. V. Data mining techniques using weka classification for sickle cell disease. **International Journal of Computer Science and Information Technologies**, v. 5(4), p. 1–4, 2014. ISSN 09759646. Citado nas pp. 28, 29, 46, 50, 52 e 53.
- SON, J.-D.; AHN, B.-H.; HA, J.-M.; CHOI, B.-K. An availability of mems-based accelerometers and current sensors in machinery fault diagnosis. **Measurement**, v. 94, p. 680–691, 2016. ISSN 0263-2241. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0263224116304961>. Citado nas pp. 21 e 40.
- SOUZA, S. V. R. de; MACHADO, Á. M. L. Calibração de erros sistemáticos em acelerômetros mems. **Boletim de Ciências Geodésicas**, SciELO Brasil, v. 22, n. 4, p. 835–850, 2016. Citado nas pp. 21 e 22.
- TAO, X.; REN, C.; WU, Y.; LI, Q.; GUO, W.; LIU, R.; HE, Q.; ZOU, J. Bearings fault detection using wavelet transform and generalized gaussian density modeling. **Measurement**, v. 155, p. 107557, 2020. ISSN 0263-2241. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0263224120300944>. Citado nas pp. 15 e 39.
- TEIXEIRA, N. S.; PASTRANA, M. A.; MOURA, H. G.; MUÑOZ, D. M. Bringing bearing fault detection to embedded systems using low-cost machine learning techniques. In: **2024 IEEE Latin American Conference on Computational Intelligence (LA-CCI)**. [S.l.: s.n.], 2024. p. 1–6. Citado na p. 15.
- TÜMA, J. Transmission and gearbox noise and vibration prediction and control, in: 16th international congress on sound and vibration. In: . [S.l.: s.n.], 2009. Citado na p. 23.
- VERMA, A.; GOYAL, A.; KUMARA, S.; KURFESS, T. Edge-cloud computing performance benchmarking for iot based machinery vibration monitoring. **Manufacturing Letters**, v. 27, p. 39–41, 2021. ISSN 2213-8463. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2213846320301759>. Citado nas pp. 15 e 40.

WAGNER, G. B. Análise modal operacional no domínio do tempo: Um estudo crítico dos métodos de identificação. **Pontifícia Universidade Católica, Rio de Janeiro, Brazil**, 2017. Citado na p. [22](#).

XILINX. **Zynq 7000 SoC and 7 Series Devices Memory Interface Solutions v4.2, LogiCORE IP Data Sheet**. [S.l.], 2016. Disponível em: [://docs.amd.com/](http://docs.amd.com/). Disponível em: [https://docs.amd.com/v/u/en-US/ug474\\_7Series\\_CLB](https://docs.amd.com/v/u/en-US/ug474_7Series_CLB). Citado nas pp. [32, 33 e 34](#).

XILINX. **Fast Fourier Transform v9.1**. [S.l.], 2022. Product Guide, PG109. Disponível em: <https://www.xilinx.com/products/intellectual-property/fft.html>. Citado na p. [37](#).

# Apêndices

# Apêndice A – Códigos de programação

## A.1 Gerador de Sinal de Caixa de Engrenagens

Código A.1 – Código de Python

```

1 import numpy as np
2 import plotly.graph_objects as go
3 from scipy.signal import butter, filtfilt, hilbert
4 from plotly.subplots import make_subplots
5
6 # Definição dos parâmetros do sinal
7 fs = 20E3 # Taxa de amostragem (Hz)
8 Np = 13 # Número de dentes no pinhão
9 Ng = 35 # Número de dentes na engrenagem
10 fPin = 22,5 # Frequência do eixo do pinhão (Hz)
11 fGear = fPin * Np / Ng # Frequência do eixo da engrenagem (Hz)
12 fMesh = fPin * Np # Frequência da engrenagem (Hz)
13 t = np.arange(0, 20, 1 / fs)
14 vfIn = 0.4 * np.sin(2 * np.pi * fPin * t) # Forma de onda do pinhão
15 vfOut = 0.2 * np.sin(2 * np.pi * fGear * t) # Forma de onda da
16     engrenagem
17 vMesh = np.sin(2 * np.pi * fMesh * t) # Forma de onda da engrenagem
18 vNoFault = vfIn + vfOut + vMesh
19 vNoFaultNoisy = vNoFault + np.random.randn(len(t)) / 5
20
21 # Criando bandas laterais
22 SideBands = np.arange(-3, 4)
23 SideBandAmp = [0.02, 0.1, 0.4, 0, 0.4, 0.1, 0.02] # Amplitudes das
24     bandas laterais
25 SideBandFreq = fMesh + SideBands * fPin # Frequências das bandas
26     laterais
27 vSideBands = np.dot(SideBandAmp, np.sin(2 * np.pi *
28         np.outer(SideBandFreq, t)))
29
30 # Adicionando bandas laterais ao sinal com falha
31 vPinFaultNoisy = vNoFaultNoisy + vSideBands
32
33 # Aplicando filtro passa-baixa
34 fc = 2000 # Frequência de corte do filtro passa-baixa
35 b, a = butter(6, fc / (fs / 2), 'low')
36 vNoFaultFiltered = filtfilt(b, a, vNoFaultNoisy)
37 vFaultFiltered = filtfilt(b, a, vPinFaultNoisy)
38
39 # Calculando FFT dos sinais filtrados
40 fft_no_fault_result = np.abs(np.fft.fft(vNoFaultFiltered))

```

```
37 fft_fault_result = np.abs(np.fft.fft(vFaultFiltered))
38
39 # Frequências para a primeira FFT
40 freq_no_fault = np.fft.fftfreq(len(vNoFaultFiltered), 1 / fs)
41 # Frequências para a segunda FFT
42 freq_fault = np.fft.fftfreq(len(vFaultFiltered), 1 / fs)
43
44 # Salvar os valores da primeira FFT em um arquivo de texto
45 with open('valores_fft_primeira_CORRETO.txt', 'w') as file:
46     file.write("Frequencia,Intensidade,Grupo\n")
47     for freq, intensity in zip(freq_no_fault, fft_no_fault_result):
48         file.write(f"{freq},{intensity},0\n") # 0 representa sem
49             falha
50
51 # Salvar os valores da segunda FFT em um arquivo de texto
52 with open('valores_fft_segunda_CORRETO.txt', 'w') as file:
53     file.write("Frequencia,Intensidade,Grupo\n")
54     for freq, intensity in zip(freq_fault, fft_fault_result):
55         file.write(f"{freq},{intensity},1\n") # 1 representa com
56             falha
57
58 # Aplicando transformada de Hilbert
59 hilbert_no_fault_result = np.abs(hilbert(vNoFaultFiltered))
60 hilbert_fault_result = np.abs(hilbert(vFaultFiltered))
61
62 # Calculando FFT dos sinais após a transformada de Hilbert
63 fft_hilbert_no_fault_result =
64     np.abs(np.fft.fft(hilbert_no_fault_result))
65 fft_hilbert_fault_result = np.abs(np.fft.fft(hilbert_fault_result))
66
67 # Frequências para a terceira FFT
68 freq_hilbert_no_fault = np.fft.fftfreq(len(hilbert_no_fault_result),
69     1 / fs)
70 # Frequências para a quarta FFT
71 freq_hilbert_fault = np.fft.fftfreq(len(hilbert_fault_result), 1 /
72     fs)
73
74 # Salvar os valores da terceira FFT em um arquivo de texto
75 with open('valores_fft_hilbert_primeira_CORRETO.txt', 'w') as file:
76     file.write("Frequencia,Intensidade,Grupo\n")
77     for freq, intensity in zip(freq_hilbert_no_fault,
78         fft_hilbert_no_fault_result):
79         file.write(f"{freq},{intensity},0\n") # 0 representa sem
80             falha
81
82 # Salvar os valores da quarta FFT em um arquivo de texto
83 with open('valores_fft_hilbert_segunda_CORRETO.txt', 'w') as file:
84     file.write("Frequencia,Intensidade,Grupo\n")
85     for freq, intensity in zip(freq_hilbert_fault,
86         fft_hilbert_fault_result):
87         file.write(f"{freq},{intensity},1\n") # 1 representa com
88             falha
```

```

81 # Criando subplots com 5 linhas e 1 coluna
82 fig = make_subplots(rows=5, cols=1, vertical_spacing=0.1)
83
84 # Adicionando cada gráfico em um subplot diferente
85 fig.add_trace(go.Scatter(x=t, y=vNoFaultNoisy, mode='lines',
86     name='Sem Falha'), row=1, col=1)
87 fig.add_trace(go.Scatter(x=t, y=vPinFaultNoisy, mode='lines',
88     name='Com Falha'), row=1, col=1)
89 fig.add_trace(go.Scatter(x=t, y=vNoFaultFiltered, mode='lines',
90     name='Sem Falha'), row=2, col=1)
91 fig.add_trace(go.Scatter(x=t, y=vFaultFiltered, mode='lines',
92     name='Com Falha'), row=2, col=1)
93 fig.add_trace(go.Scatter(x=t, y=hilbert_no_fault_result,
94     mode='lines', name='Sem Falha'), row=3, col=1)
95 fig.add_trace(go.Scatter(x=t, y=hilbert_fault_result, mode='lines',
96     name='Com Falha'), row=3, col=1)
97 fig.add_trace(go.Scatter(x=freq_no_fault, y=fft_no_fault_result,
98     mode='lines', name='Sem Falha'), row=4, col=1)
99 fig.add_trace(go.Scatter(x=freq_fault, y=fft_fault_result,
100    mode='lines', name='Com Falha'), row=4, col=1)
101 fig.add_trace(go.Scatter(x=freq_hilbert_no_fault,
102    y=fft_hilbert_no_fault_result, mode='lines', name='Sem Falha'),
103    row=5, col=1)
104 fig.add_trace(go.Scatter(x=freq_hilbert_fault,
105    y=fft_hilbert_fault_result, mode='lines', name='Com Falha'),
106    row=5, col=1)
107
108 # Adicionando títulos e rótulos dos eixos
109 fig.update_yaxes(title_text="Amplitude", tickfont=dict(size=16),
110     titlefont=dict(size=16), row=1, col=1)
111 fig.update_yaxes(title_text="Amplitude", tickfont=dict(size=16),
112     titlefont=dict(size=16), row=2, col=1)
113 fig.update_yaxes(title_text="Hilbert", tickfont=dict(size=16),
114     titlefont=dict(size=16), row=3, col=1)
115 fig.update_yaxes(title_text="FFT", tickfont=dict(size=16),
116     titlefont=dict(size=16), row=4, col=1)
117 fig.update_yaxes(title_text="FFT + Hilbert", tickfont=dict(size=16),
118     titlefont=dict(size=16), row=5, col=1)
119
120 fig.update_xaxes(title_text="Frequencia (Hz)",
121     tickfont=dict(size=16), titlefont=dict(size=16),
122     tickvals=np.arange(np.min(freq_hilbert_no_fault),
123         np.max(freq_hilbert_no_fault) + 1, 50), row=5, col=1)
124 fig.update_xaxes(title_text="Frequencia (Hz)",
125     tickfont=dict(size=16), titlefont=dict(size=16),
126     tickvals=np.arange(np.min(freq_no_fault), np.max(freq_no_fault) +
127         1, 50), row=4, col=1)
128 fig.update_xaxes(title_text="Tempo (s)", tickfont=dict(size=16),
129     titlefont=dict(size=16), tickvals=np.arange(np.min(t), np.max(t) +
130         1, 0.05), row=3, col=1)
131 fig.update_xaxes(title_text="Tempo (s)", tickfont=dict(size=16),
132     titlefont=dict(size=16), tickvals=np.arange(np.min(t), np.max(t) +
133         1, 0.05), row=2, col=1)

```

```

107 fig.update_xaxes(title_text="Tempo (s)", tickfont=dict(size=16),
108     titlefont=dict(size=16), tickvals=np.arange(np.min(t), np.max(t)
109     + 1, 0.5), row=1, col=1)
110
111 # Adicionando títulos aos subplots
112 fig.update_layout(
113     title="Análise de Sinais com e sem Falha",
114     height=2000,
115     width=1000
116 )
117
118 # Salvar o gráfico em um arquivo HTML
119 fig.write_html("graficos_com_titulos.html")
120
121 # Exibindo o endereço IP para acessar o arquivo HTML
122 import http.server
123 import socketserver
124 import threading
125
126 # Function to serve the HTML file
127 def serve_html():
128     PORT = 8000
129     Handler = http.server.SimpleHTTPRequestHandler
130     with socketserver.TCPServer(('127.0.0.1', PORT), Handler) as
131         httpd:
132             print(f"Serving at http://127.0.0.1:{PORT}")
133             httpd.serve_forever()
134
135 # Start serving the HTML file in a separate thread
136 server_thread = threading.Thread(target=serve_html)
137 server_thread.start()
138
139 # Print the URL to access the HTML file
140 print("Para visualizar os gráficos, acesse:
141     http://127.0.0.1:8000/graficos_com_titulos.html")

```

## A.2 Exemplo de Teste Simulação Caixa de Engrenagens

Código A.2 – Código de Teste no Octave

```

1 fs = 20E3;           % Sample Rate (Hz)
2
3 Np = 13;            % Number of teeth on pinion
4 Ng = 35;            % Number of teeth on gear
5
6 fPin = 22.5;        % Pinion (Input) shaft frequency (Hz)
7
8 fGear = fPin*Np/Ng; % Gear (Output) shaft frequency (Hz)
9
10 fMesh = fPin*Np;    % Gear Mesh frequency (Hz)

```

```

11 t = 0:1/fs:20-1/fs;
12
13
14 vfIn = 0.4*sin(2*pi*fPin*t); % Pinion waveform
15 vfOut = 0.2*sin(2*pi*fGear*t); % Gear waveform
16
17 vMesh = sin(2*pi*fMesh*t); % Gear-mesh waveform
18
19 plot(t, vfIn + vfOut + vMesh)
20 xlim([0 0.25])
21 xlabel('Time (s)')
22 ylabel('Acceleration')
23
24 ipf = fGear;
25 fImpact = 2000;
26
27 tImpact = 0:1/fs:2.5e-4-1/fs;
28 xImpact = sin(2*pi*fImpact*tImpact)/3;
29
30 xComb = zeros(size(t));
31
32 Ind = (0.25*fs/fMesh):(fs/ipf):length(t);
33 Ind = round(Ind);
34 xComb(Ind) = 1;
35
36 xPer = 2*conv(xComb,xImpact,'same');
37
38 vNoFault = vfIn + vfOut + vMesh;
39 vFault = vNoFault + xPer;
40
41 vNoFaultNoisy = vNoFault + randn(size(t))/5;
42 vFaultNoisy = vFault + randn(size(t))/5;
43
44 subplot(2,1,1)
45 plot(t,vNoFaultNoisy)
46 xlabel('Time (s)')
47 ylabel('Acceleration')
48 xlim([0.0 0.3])
49 ylim([-2.5 2.5])
50 title('Noisy Signal for Healthy Gear')
51
52
53 subplot(2,1,2)
54 plot(t,vFaultNoisy)
55 xlabel('Time (s)')
56 ylabel('Acceleration')
57 xlim([0.0 0.3])
58 ylim([-2.5 2.5])
59 title('Noisy Signal for Faulty Gear')
60 hold on
61 MarkX = t(Ind(1:3));
62 MarkY = 2.5;
63 plot(MarkX,MarkY,'rv','MarkerFaceColor','red')
```

```

64 hold off
65
66
67 figure
68
69 % Calculate Power Spectra
70 [Pxx1, f1] = periodogram(vFaultNoisy, hamming(length(vFaultNoisy)),
   length(vFaultNoisy), fs);
71 [Pxx2, f2] = periodogram(vNoFaultNoisy,
   hamming(length(vNoFaultNoisy)), length(vNoFaultNoisy), fs);
72
73 plot(f1, 10*log10(Pxx1), f2, 10*log10(Pxx2), ':')
74 xlabel('Frequency (Hz)')
75 ylabel('Power Spectrum (dB)')
76
77 hold on
78 plot(fGear, 0, 'rv', 'MarkerFaceColor', 'red')
79 plot(fPin, 0, 'gv', 'MarkerFaceColor', 'green')
80 plot(fMesh, 0, 'bv', 'MarkerFaceColor', 'blue')
81 hold off
82
83 legend('Faulty', 'Healthy', 'f_{Gear}', 'f_{Pinion}', 'f_{Mesh}')
84
85 figure
86 p1 = plot(f1, 10*log10(Pxx1));
87 xlabel('Frequency (Hz)')
88 ylabel('Power Spectrum (dB)')
89 xlim([250 340])
90 ylim([-70 -40])
91
92 hold on
93 p2 = plot(f2, 10*log10(Pxx2));
94
95 harmonics = -5:5;
96 SBandsGear = (fMesh+fGear.*harmonics);
97 [X1,Y1] = meshgrid(SBandsGear,ylim);
98
99 SBandsPinion = (fMesh+fPin.*harmonics);
100 [X2,Y2] = meshgrid(SBandsPinion,ylim);
101
102 p3 = plot(X1,Y1,:r');
103 p4 = plot(X2,Y2,:k');
104 hold off
105 legend([p1 p2 p3(1) p4(1)],{'Faulty Gear';'Healthy
   Gear';'f_{sideband,Gear}';'f_{sideband,Pinion}'})
106
107 tPulseIn = 0:1/fPin:max(t);
108 taPin = tsa(vFaultNoisy,fs,tPulseIn,'NumRotations',10);
109
110 tPulseOut = 0:1/fGear:max(t);
111 taGear = tsa(vFaultNoisy,fs,tPulseOut,'NumRotations',10);
112
113 figure

```

```

114 subplot(2,1,1)
115 tsa(vFaultNoisy,fs,tPulseIn,'NumRotations',10)
116 xlim([0.5 1.5])
117 ylim([-2 2])
118 title('TSA Signal for Pinion')
119
120 subplot(2,1,2)
121 tsa(vFaultNoisy,fs,tPulseOut,'NumRotations',10)
122 xlim([0.5 1.5])
123 ylim([-2 2])
124 title('TSA Signal for Gear')
125 hold on
126 plot(1.006,2,'rv','MarkerFaceColor','red')
127 hold off
128
129 figure
130 pspectrum(taGear,fs,'FrequencyResolution',2.2,'FrequencyLimits',[200
    400])
131
132 harmonics = -15:15;
133 SBandsGear=(fMesh+fGear.*harmonics);
134
135 [X1,Y1] = meshgrid(SBandsGear,ylim);
136 [XM,YM] = meshgrid(fMesh,ylim);
137
138 hold on
139 plot(XM,YM,'--k',X1,Y1,:r')
140 legend('Power Spectra','Gear-Mesh Frequency','f_{sideband,Gear}')
141 hold off
142
143 title('TSA Gear (Output Shaft)')
144
145 figure
146 pspectrum(taPin,fs,'FrequencyResolution',5.8,'FrequencyLimits',[200
    400])
147
148 SBandsPinion = (fMesh+fPin.*harmonics);
149
150 [X2,Y2] = meshgrid(SBandsPinion,ylim);
151 [XM,YM] = meshgrid(fMesh,ylim);
152
153 hold on
154 plot(XM,YM,'--b',X2,Y2,:k')
155 legend('Power Spectra','Gear-Mesh Frequency','f_{sideband,Pinion}')
156 hold off
157
158 title('TSA Pinion (Input Shaft)')
159

```

### A.3 Teste Embarcando Modelo na Esp32

### Código A.3 – Código no Arduino IDE

```

1 #include <Arduino.h>
2 #include <sys/time.h>
3
4 // Estrutura para armazenar os valores do dataset
5 typedef struct {
6     float frequency;
7     float intensity;
8 } DatasetValue;
9
10 // Função para ler os valores do dataset
11 void readDataset(DatasetValue *dataset) {
12     float values[][] = {
13         { 5129.15, 0.40069}, //falho
14         { -9708.96, 0.001236 }, //saudavel
15
16     };
17     int num_values = sizeof(values) / sizeof(values[0]);
18
19     for (int i = 0; i < num_values; i++) {
20         dataset[i].frequency = values[i][0];
21         dataset[i].intensity = values[i][1];
22     }
23 }
24
25 int predictFailure(DatasetValue *dataset, int num_values) {
26     //Serial.println("Inicio do teste: ");
27     for (int i = 1; i <= num_values; i++) {
28         float frequency = dataset[i].frequency;
29         float intensity = dataset[i].intensity;
30         //Serial.println("");
31         //Serial.print("Teste: ");
32         //Serial.println(i);
33         //Serial.print(" ");
34
35         if (intensity <= 0.080193) {
36             if (intensity <= 0.001235) {
37                 Serial.println("A máquina está saudável.");
38             } else { // intensidade > 0.001235
39                 if (frequency <= -9708.95) {
40                     Serial.println("A máquina está falha.");
41                 } else if (frequency <= 9708.95) { // frequencia >
42                     -9708.95 e <= 9708.95
43                     Serial.println("A máquina está saudável.");
44                 } else { // frequencia > 9708.95
45                     Serial.println("A máquina está falha.");
46                 }
47             }
48         } else { // intensidade > 0.080193
49             if (frequency <= -5285.9) {
50                 Serial.println("A máquina está falha.");
51             }
52         }
53     }
54 }
```

```
50 } else if (frequency <= 5129.15) { // frequencia > -5285.9 e <=
51     5129.15
52     if (intensity <= 0.40069) {
53         Serial.println("A máquina está saudável.");
54     } else if (frequency <= -3840.9) { // frequencia > -5285.9 e
55         <= 5129.15 e > -3840.9
56         Serial.println("A máquina está falha.");
57     } else if (frequency <= 3826.8) { // frequencia > -5285.9 e
58         <= 5129.15 e > -3840.9 e <= 3826.8
59         if (intensity <= 1.484293) {
60             Serial.println("A máquina está saudável.");
61         } else if (frequency <= -3242.95) { // frequencia >
62             -5285.9 e <= 5129.15 e > -3840.9 e <= 3826.8 e <=
63             -3242.95
64             Serial.println("A máquina está falha.");
65     } else if (frequency <= 3229.25) { // frequencia >
66         -5285.9 e <= 5129.15 e > -3840.9 e <= 3826.8 e >
67         -3242.95 e <= 3229.25
68         if (intensity <= 2.96571) {
69             Serial.println("A máquina está saudável.");
70         } else if (frequency <= -2645.05) { // frequencia >
71             -5285.9 e <= 5129.15 e > -3840.9 e <= 3826.8 e >
72             -3242.95 e <= 3229.25 e <= -2645.05
73             Serial.println("A máquina está falha.");
74     } else if (frequency <= 2645) { // frequencia >
75         -5285.9 e <= 5129.15 e > -3840.9 e <= 3826.8 e >
-3242.95 e <= 3229.25 e > -2645.05 e <= 2645
if (intensity <= 5.516686) {
    Serial.println("A máquina está saudável.");
} else if (frequency <= -2333.6) { // frequencia
> -5285.9 e <= 5129.15 e > -3840.9 e <=
3826.8 e > -3242.95 e <= 3229.25 e > -2645.05
e <= 2645 e <= -2333.6
Serial.println("A máquina está falha.");
} else if (frequency <= 2333.1) { // frequencia
> -5285.9 e <= 5129.15 e > -3840.9 e <=
3826.8 e > -3242.95 e <= 3229.25 e > -2645.05
e <= 2645 e > -2333.6 e <= 2333.1
if (intensity <= 10.788543) {
    Serial.println("A máquina está
saudável.");
} else if (frequency <= -2172.95) { // frequencia > -5285.9 e <= 5129.15 e >
-3840.9 e <= 3826.8 e > -3242.95 e <=
3229.25 e > -2645.05 e <= 2645 e >
-2333.6 e <= 2333.1 e <= -2172.95
Serial.println("A máquina está falha.");
} else if (frequency <= 2172.9) { // frequencia > -5285.9 e <= 5129.15 e >
-3840.9 e <= 3826.8 e > -3242.95 e <=
3229.25 e > -2645.05 e <= 2645 e >
-2333.6 e <= 2333.1 e > -2172.95 e <=
2172.9
```

```
76         if (intensity <= 19.82965) {
77             Serial.println("A máquina está
78             saudável.");
79         } else if (frequency <= -2013) { //
80             frequencia > -5285.9 e <= 5129.15 e >
81             -3840.9 e <= 3826.8 e > -3242.95 e <=
82             3229.25 e > -2645.05 e <= 2645 e >
83             -2333.6 e <= 2333.1 e > -2172.95 e <=
84             2172.9 e <= -2013
85             Serial.println("A máquina está
86             falha.");
87         } else if (frequency <= 2012.95) { //
88             frequencia > -5285.9 e <= 5129.15 e >
89             -3840.9 e <= 3826.8 e > -3242.95 e <=
90             3229.25 e > -2645.05 e <= 2645 e >
91             -2333.6 e <= 2333.1 e > -2172.95 e <=
92             2172.9 e > -2013 e <= 2012.95
93             if (intensity <= 33.758874) {
94                 Serial.println("A máquina está
95                 saudável.");
96             } else { // intensidade > 33.758874
97                 Serial.println("A máquina está
98                 falha.");
99             }
100         } else { // frequencia > 2012.95
101             Serial.println("A máquina está
102             falha.");
103         }
104     } else { // frequencia > 2172.9
105         Serial.println("A máquina está
106         falha.");
107     }
108 }
109 //}
110 //Serial.println("");
111 //Serial.println("Termino do Dataset");
112 //Serial.println("");
113 return 0;
```

```

114 }
115
116
117 void setup() {
118     Serial.begin(9600); // Inicializa a comunicação serial
119 }
120
121 void loop() {
122     struct timeval start, end;
123     gettimeofday(&start, NULL);
124
125     DatasetValue dataset[18]; // Tamanho do dataset
126
127     readDataset(dataset);
128     int result = 0;
129     result = predictFailure(dataset, 2);
130     // Imprime o resultado
131
132     gettimeofday(&end, NULL);
133     Serial.print("Tempo de execução: ");
134     Serial.print(((end.tv_sec * 1000000 + end.tv_usec) - (start.tv_sec
135         * 1000000 + start.tv_usec)));
136     Serial.println(" microsegundos");
137     Serial.println("");
138
139     delay(1000); // Aguarda 1 segundo antes da próxima iteração
}

```

## A.4 Código VHDL do *top module*

Código A.4 – Código VHDL no Vivado

```

1 -----
2 -- Company: University of Brasília
3 -- Engineer: Natlia Schulz Teixeira
4 --
5 -- Create Date: 02/14/2025 03:53:47 PM
6 -- Module Name: faulty_detection - Behavioral
7 -- Project Name: Implementação de Modelo para Detecção de Falhas em
8     M quinas Rotativas com FPGA
9 --
10 -- Revision:
11 -- Revision 0.01 - File Created
12 -- Additional Comments:
13 --
14
15 library IEEE;
16 use IEEE.STD_LOGIC_1164.ALL;
17 use IEEE.NUMERIC_STD.ALL;

```

```

18
19 entity top_module is
20   Port (
21     clk      : in STD_LOGIC;
22     resetn  : in STD_LOGIC;
23     m_axis_data_tready : in STD_LOGIC; -- interno
24     m_axis_data_tvalid : out STD_LOGIC;
25     m_axis_data_tdata  : out STD_LOGIC_VECTOR(31 downto 0);
26     m_axis_data_tlast  : out STD_LOGIC;
27     s_axis_data_tready : out STD_LOGIC; -- Alterado para 'out'
28     start    : in STD_LOGIC; -- Sinal para iniciar a configura
29               da FFT
30     auxtready : out STD_LOGIC -- Sinal auxiliar para tready
31   --      led : out STD_LOGIC -- Sinal auxiliar para tready
32 );
33 end top_module;
34
35
36 architecture Behavioral of top_module is
37
38   signal s_axis_data_tvalid_int : STD_LOGIC; -- Sinal interno
39               para tvalid
40   signal s_axis_data_tdata_int  : STD_LOGIC_VECTOR(31 downto 0);
41               -- Sinal interno para tdata
42   signal s_axis_data_tlast_int : STD_LOGIC :='0'; -- Sinal
43               interno para tlast
44   signal s_axis_data_tready_int : STD_LOGIC; -- Sinal interno
45               para tready
46   signal tlast_memory : STD_LOGIC:='0'; -- Sinal interno para
47               tready
48   signal start_int : STD_LOGIC; -- Sinal interno para start
49   -- Sinais internos para configura da FFT
50   signal s_axis_config_tdata_int : STD_LOGIC_VECTOR(23 downto 0);
51               -- Sinal interno para tdata
52   signal s_axis_config_tvalid_int : STD_LOGIC; -- Sinal interno
53               para tvalid
54   signal s_axis_config_tready_int : STD_LOGIC; -- Sinal interno
55               para tready
56   signal config_reg : STD_LOGIC_VECTOR(11 downto 0) := (others =>
57               '0');
58   signal log2nFFT    : STD_LOGIC_VECTOR(3 downto 0) := (others =>
59               '0');
60   signal FWD         : STD_LOGIC_VECTOR(4 downto 0) := "10000"; --
61               FFT
62   signal INV         : STD_LOGIC_VECTOR(4 downto 0) := "00000"; --
63               IFFT
64   -- Sinais para configura da FFT
65   signal event_frame_started : STD_LOGIC;
66   signal event_tlast_unexpected : STD_LOGIC;
67   signal event_tlast_missing   : STD_LOGIC;
68   signal event_status_channel_halt : STD_LOGIC;
69   signal event_data_in_channel_halt : STD_LOGIC;
70   signal event_data_out_channel_halt : STD_LOGIC;
71
72

```

```

58     signal m_axis_data_tdata_int : STD_LOGIC_VECTOR(31 downto 0);
59
60     -- Sinal para o limiar
61     signal threshold : STD_LOGIC_VECTOR(15 downto 0) := x"7FFF";
62
63     -- Sinais adicionais
64     signal lock : STD_LOGIC := '0'; -- Sinal de lock para controle
65           de configura
66
67     -- Componente datasrc
68     component datasrc
69         Port (
70             clk      : in STD_LOGIC;
71             resetn : in STD_LOGIC;
72             tready : in STD_LOGIC;
73             start   : in STD_LOGIC;
74             tvalid  : out STD_LOGIC;
75             tlast   : out STD_LOGIC;
76             tdata   : out STD_LOGIC_VECTOR(31 downto 0)
77         );
78     end component;
79
80     -- Componente FFT (substitua pelo nome correto do seu IP FFT)
81     component xfft_0
82         Port (
83             -- Interface de entrada de dados (S_AXIS_DATA)
84             s_axis_data_tdata  : in STD_LOGIC_VECTOR(31 downto 0);
85             s_axis_data_tlast  : in STD_LOGIC;
86             s_axis_data_tready : out STD_LOGIC;
87             s_axis_data_tvalid : in STD_LOGIC;
88             s_axis_config_tdata : in STD_LOGIC_VECTOR(23 downto 0);
89             s_axis_config_tready : out STD_LOGIC;
90             s_axis_config_tvalid : in STD_LOGIC;
91             m_axis_data_tdata  : out STD_LOGIC_VECTOR(31 downto 0);
92             m_axis_data_tlast  : out STD_LOGIC;
93             m_axis_data_tready : in STD_LOGIC;
94             m_axis_data_tvalid : out STD_LOGIC;
95             aclk      : in STD_LOGIC;
96             aresetn  : in STD_LOGIC;
97             event_frame_started : out STD_LOGIC;
98             event_tlast_unexpected : out STD_LOGIC;
99             event_tlast_missing : out STD_LOGIC;
100            event_status_channel_halt : out STD_LOGIC;
101            event_data_in_channel_halt : out STD_LOGIC;
102            event_data_out_channel_halt : out STD_LOGIC
103        );
104    end component;
105
106    -- component faulty_detection is
107    --     Port (
108    --         clk          : in STD_LOGIC;
109    --         resetn       : in STD_LOGIC;

```

```

109      fft_data      : in  STD_LOGIC_VECTOR(31 downto 0);  --
110      Dados de sada da FFT
111      fft_valid    : in  STD_LOGIC;                      --
112      Sinal de valida dos dados da FFT
113      fft_last     : in  STD_LOGIC;                      --
114      Sinal de ltimo dado da FFT
115      led_fault   : out STD_LOGIC;                      --
116      LED que indica falha
117      threshold   : in  STD_LOGIC_VECTOR(15 downto 0)  --
118      Limiar para detec de pico
119      );
120  end component;
121
122 begin
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
-- Instancia do datasrc
datasrc_inst : datasrc
  Port map (
    clk      => clk,
    start   => start_int,
    resetn  => resetn,
    tready  => s_axis_data_tready_int,  -- Conecta ao sinal
    -- interno tready
    tvalid  => s_axis_data_tvalid_int,  -- Conecta ao sinal
    -- interno tvalid
    tdata   => s_axis_data_tdata_int,   -- Conecta ao sinal
    -- interno tdata
    tlast   => tlast_memory
  );
-- Instancia do FFT IP
xfft_0_inst : xfft_0
  Port map (
    -- Interface de entrada de dados
    s_axis_data_tdata  => s_axis_data_tdata_int,  -- Conecta
    -- ao sinal interno tdata
    s_axis_data_tlast  => s_axis_data_tlast_int,  -- Conecta
    -- ao sinal interno tlast
    s_axis_data_tready => s_axis_data_tready,    -- Conecta ao
    -- sinal interno tready
    s_axis_data_tvalid => s_axis_data_tvalid_int,  -- Conecta ao sinal interno tvalid
    s_axis_config_tdata => s_axis_config_tdata_int, -- Conecta ao sinal interno tdata
    s_axis_config_tready => s_axis_config_tready_int, -- Conecta ao sinal interno tready
    s_axis_config_tvalid => s_axis_config_tvalid_int, -- Conecta ao sinal interno tvalid
    m_axis_data_tdata  => m_axis_data_tdata,
    m_axis_data_tlast  => m_axis_data_tlast,
    m_axis_data_tready => m_axis_data_tready,
    m_axis_data_tvalid => m_axis_data_tvalid,
    aclk      => clk,
  );

```

```

147         aresetn => resetn,
148         event_frame_started => event_frame_started,
149         event_tlast_unexpected => event_tlast_unexpected,
150         event_tlast_missing => event_tlast_missing,
151         event_status_channel_halt => event_status_channel_halt,
152         event_data_in_channel_halt => event_data_in_channel_halt,
153         event_data_out_channel_halt =>
154             event_data_out_channel_halt
155     );
156 
156 --      -- Instancia do indicador de falha
157 --      faulty_detection_inst : faulty_detection
158 --      Port map (
159 --          clk      => clk,
160 --          resetn   => resetn,
161 --          fft_data  => m_axis_data_tdata,
162 --          fft_valid => m_axis_data_tvalid,
163 --          fft_last   => m_axis_data_tlast,
164 --          threshold  => threshold,
165 --          led_fault  => led  -- Conecte ao pino do LED no top
166 --          level
166 --      );
167 
168 --      -- Conecta o sinal interno tready ao sinal de saida
169 --      s_axis_data_tready <= s_axis_data_tready_int;
170 
171 start_int <= start;
172 
173 process(clk, resetn)
174 begin
175     if resetn = '0' then
176         auxready <= '0';
177         lock <= '0';
178         s_axis_config_tdata_int <= (others => '0');
179         s_axis_config_tvalid_int <= '0';
180         s_axis_data_tlast_int <= '0';
181     elsif rising_edge(clk) then
182         s_axis_config_tvalid_int <= '0';
183     --         if start_int = '1' and s_axis_config_tready_int = '1'
184     and lock = '0' then
185         if start_int = '1' and lock = '0' then
186             s_axis_config_tdata_int <= "000" & "00110101010" &
187                 FWD & "01111"; -- 24 bits tirei um zero de
188                 000110101010
189             s_axis_config_tvalid_int <= '1';
190             lock <= '1';
191         end if;
192 
193         if lock = '1' then
194             s_axis_data_tready_int <= '1';
195             auxready <= '1';
196         end if;
197 
```

```

195      --          if s_axis_data_tlast_int = '1' then
196          if tlast_memory = '1' then
197              s_axis_data_tready_int <= '0';
198              lock <= '0';
199          end if;
200      end if;
201  end process;
202
203 end Behavioral;

```

## A.5 Código Verilog da memória

Código A.5 – Código Verilog no Vivado

```

1 `timescale 1ns / 1ps
2
3 module datasrc(
4     input clk,
5     input start,           // Sinal de start para iniciar a leitura e
6         envio de dados.
7     input resetn,
8     input tready,
9     output tvalid,
10    output [31:0] tdata,
11    output tlast
12 );
13
14     parameter infile = "input_sinal_tempo.mem";
15
16     reg [31:0] mem[0:32767]; // Memria com 32768 posies (0 a
17         32767)
18     localparam s0 = 'd0;      // Estado inicial (aguardando start).
19     localparam s1 = 'd1;      // Estado de leitura e prepara dos
20         dados.
21     localparam s2 = 'd2;      // Estado de envio dos dados.
22     localparam s3 = 'd3;      // Estado de espera (quando tready =
23         0).
24
25     reg [2:0] state, n_state;
26     reg [14:0] addr, n_addr; // Endereo de 15 bits para suportar
27         32768 posies
28     reg [31:0] d0, d1, d2, n_d1, n_d2;
29     reg valid, n_valid;
30     reg tlast_reg;          // Registro para armazenar o valor de
31         tlast
32
33     initial begin
34         $readmemh(infile, mem); // Inicializa a memoria com os
35             dados do arquivo
36         state = s0;           // Inicializa o estado

```



```

77      end else begin
78          n_state = s0;           // Permanece no estado s0 at
79              que start seja ativado.
80      end
81  s1: begin
82      n_d1 = d0;           // Carrega o proximo dado
83          (d0) em n_d1.
84      n_addr = addr + 1;     // Incrementa o endereco para
85          apontar para o proximo dado na memoria.
86      n_state = s2;           // Define o proximo estado
87          como s2.
88      n_valid = 1;           // Ativa o sinal de validade
89          (n_valid = 1).
90  end
91  s2: begin
92      n_d1 = d0;           // Carrega o proximo dado
93          (d0) em n_d1.
94      n_d2 = d1;           // Carrega o dado atual (d1)
95          em n_d2 (dado a ser enviado).
96      n_valid = 1;           // Mantm o sinal de validade
97          ativo (n_valid = 1).
98      if (tready) begin // Se o receptor estiver
99          pronto (tready = 1):
100         if (addr == 32767) begin // Verifica se o
101             endereco atual o ltimo (32767).
102             n_state = s0; // Retorna ao estado inicial
103                 aps o ltimo dado.
104         end else begin
105             n_addr = addr + 1; // Incrementa o endereco
106                 para o proximo dado.
107             n_state = s2; // Mantm o estado como s2
108                 para enviar o proximo dado.
109         end
110     end
111  s3: begin
112      if (tready) begin // Se o receptor estiver
113          pronto (tready = 1):
114          n_addr = addr + 1; // Incrementa o endereco para
115              o proximo dado.
116          n_state = s2; // Volta para o estado s2
117              para enviar o proximo dado.
118      end else begin // Se o receptor no estiver
119          pronto (tready = 0):
120          n_state = s3; // Muda para o estado s3 para
121              esperar.
122      end
123  end
124

```

```

110      default: begin
111          n_state = s0;           // Retorna ao estado inicial
112          s0.
113      end
114  endcase
115 end

```

## A.6 Código VHDL para detecção de falhas

Código A.6 – Código VHDL no Vivado

```

1 -----
2 -- Company: University of Brasilia
3 -- Engineer: Natlia Schulz Teixeira
4 --
5 -- Create Date: 02/14/2025 03:53:47 PM
6 -- Module Name: faulty_detection - Behavioral
7 -- Project Name: Implementa de Modelo para Detec de Falhas em
8     Mquinas Rotativas com FPGA
9 --
10 -- Revision:
11 -- Revision 0.01 - File Created
12 -- Additional Comments:
13 --
14 -----
15 library IEEE;
16 use IEEE.STD_LOGIC_1164.ALL;
17 use IEEE.NUMERIC_STD.ALL;
18
19 entity faulty_detection is
20     Port (
21         clk          : in STD_LOGIC;
22         resetn       : in STD_LOGIC;
23         fft_data     : in STD_LOGIC_VECTOR(31 downto 0);    -- Dados
24             de sada da FFT
25         fft_valid    : in STD_LOGIC;                         -- Sinal
26             de valida dos dados da FFT
27         fft_last     : in STD_LOGIC;                         -- Sinal
28             de ltimo dado da FFT
29         threshold   : in STD_LOGIC_VECTOR(15 downto 0);    -- Limiar
30             para detec de pico
31         led_fault   : out STD_LOGIC;                        -- LED
32             que indica falha
33     );
34 end faulty_detection;
35
36 architecture Behavioral of faulty_detection is

```

```

33      -- Constantes para a faixa de frequncia de 22 kHz a 40 kHz
34      constant FREQ_LOW : integer := 22000;    -- 22 kHz
35      constant FREQ_HIGH : integer := 40000;   -- 40 kHz
36
37      -- Constante para o tempo de 5 segundos
38      constant TIMER_5S : integer := 500000000;  -- 5 segundos em
39          ciclos de clock (assumindo 100 MHz)
40
41      -- Sinais internos
42      signal magnitude : unsigned(15 downto 0) := (others => '0');
43          -- Magnitude da frequncia
44      signal frequency  : integer := 0;           -- Frequncia atual
45      signal fault_flag : STD_LOGIC := '0';       -- Flag para indicar
46          falha
47      signal timer       : integer := 0;           -- Contador de tempo
48      signal led_state   : STD_LOGIC := '0';       -- Estado do LED
49      signal threshold_int : STD_LOGIC_VECTOR(15 downto 0);  --
50          Limiar para detec de pico
51
52 begin
53
54     process(clk, resetn)
55     begin
56         if resetn = '0' then
57             magnitude <= (others => '0');
58             frequency <= 0;
59             fault_flag <= '0';
60             led_state <= '0';
61             timer <= 0;
62         elsif rising_edge(clk) then
63             if fft_valid = '1' then
64                 -- Extrai a magnitude do dado da FFT (assumindo que
65                 -- os 16 bits mais significativos so a parte real)
66                 magnitude <= unsigned(fft_data(31 downto 16));
67
68                 -- Verifica se a frequncia atual est na faixa
69                 -- desejada
70                 if frequency >= FREQ_LOW and frequency <= FREQ_HIGH
71                     then
72                         -- Verifica se a magnitude est acima do limiar
73                         if magnitude > unsigned(threshold) then
74                             fault_flag <= '1';  -- Ativa a flag de falha
75                         end if;
76                     end if;
77
78                 -- Incrementa a frequncia para o proximo dado
79                 frequency <= frequency + 1;
80
81                 -- Reseta a frequncia quando o ltimo dado
82                 -- recebido
83                 if fft_last = '1' then
84                     frequency <= 0;

```

```
78      if fault_flag = '1' then
79          led_state <= '1';    -- Acende o LED
80          timer <= 0;        -- Reseta o contador de
81              tempo
82      end if;
83      fault_flag <= '0';    -- Reseta a flag de falha
84  end if;
85
86      -- Controle do tempo que o LED fica aceso
87  if led_state = '1' then
88      if timer < TIMER_5S then
89          timer <= timer + 1; -- Incrementa o contador
90      else
91          led_state <= '0';    -- Desliga o LED aps 5
92              segundos
93      end if;
94  end if;
95 end process;
96
97      -- Atribui o estado do LED ao sinal de saida
98  led_fault <= led_state;
99
100 end Behavioral;
```

# Apêndice B – Códigos de programação

## B.1 Gerador de Sinal de Caixa de Engrenagens

Código B.1 – Código de Python

```

1 import numpy as np
2 import plotly.graph_objects as go
3 from scipy.signal import butter, filtfilt, hilbert
4 from plotly.subplots import make_subplots
5
6 # Definição dos parâmetros do sinal
7 fs = 20E3 # Taxa de amostragem (Hz)
8 Np = 13 # Número de dentes no pinhão
9 Ng = 35 # Número de dentes na engrenagem
10 fPin = 22,5 # Frequência do eixo do pinhão (Hz)
11 fGear = fPin * Np / Ng # Frequência do eixo da engrenagem (Hz)
12 fMesh = fPin * Np # Frequência da engrenagem (Hz)
13 t = np.arange(0, 20, 1 / fs)
14 vfIn = 0.4 * np.sin(2 * np.pi * fPin * t) # Forma de onda do pinhão
15 vfOut = 0.2 * np.sin(2 * np.pi * fGear * t) # Forma de onda da
16     engrenagem
17 vMesh = np.sin(2 * np.pi * fMesh * t) # Forma de onda da engrenagem
18 vNoFault = vfIn + vfOut + vMesh
19 vNoFaultNoisy = vNoFault + np.random.randn(len(t)) / 5
20
21 # Criando bandas laterais
22 SideBands = np.arange(-3, 4)
23 SideBandAmp = [0.02, 0.1, 0.4, 0, 0.4, 0.1, 0.02] # Amplitudes das
24     bandas laterais
25 SideBandFreq = fMesh + SideBands * fPin # Frequências das bandas
26     laterais
27 vSideBands = np.dot(SideBandAmp, np.sin(2 * np.pi *
28         np.outer(SideBandFreq, t)))
29
30 # Adicionando bandas laterais ao sinal com falha
31 vPinFaultNoisy = vNoFaultNoisy + vSideBands
32
33 # Aplicando filtro passa-baixa
34 fc = 2000 # Frequência de corte do filtro passa-baixa
35 b, a = butter(6, fc / (fs / 2), 'low')
36 vNoFaultFiltered = filtfilt(b, a, vNoFaultNoisy)
37 vFaultFiltered = filtfilt(b, a, vPinFaultNoisy)
38
39 # Calculando FFT dos sinais filtrados
40 fft_no_fault_result = np.abs(np.fft.fft(vNoFaultFiltered))

```

```

37 fft_fault_result = np.abs(np.fft.fft(vFaultFiltered))
38
39 # Frequências para a primeira FFT
40 freq_no_fault = np.fft.fftfreq(len(vNoFaultFiltered), 1 / fs)
41 # Frequências para a segunda FFT
42 freq_fault = np.fft.fftfreq(len(vFaultFiltered), 1 / fs)
43
44 # Salvar os valores da primeira FFT em um arquivo de texto
45 with open('valores_fft_primeira_CORRETO.txt', 'w') as file:
46     file.write("Frequencia,Intensidade,Grupo\n")
47     for freq, intensity in zip(freq_no_fault, fft_no_fault_result):
48         file.write(f"{freq},{intensity},0\n") # 0 representa sem
49             falha
50
51 # Salvar os valores da segunda FFT em um arquivo de texto
52 with open('valores_fft_segunda_CORRETO.txt', 'w') as file:
53     file.write("Frequencia,Intensidade,Grupo\n")
54     for freq, intensity in zip(freq_fault, fft_fault_result):
55         file.write(f"{freq},{intensity},1\n") # 1 representa com
56             falha
57
58 # Aplicando transformada de Hilbert
59 hilbert_no_fault_result = np.abs(hilbert(vNoFaultFiltered))
60 hilbert_fault_result = np.abs(hilbert(vFaultFiltered))
61
62 # Calculando FFT dos sinais após a transformada de Hilbert
63 fft_hilbert_no_fault_result =
64     np.abs(np.fft.fft(hilbert_no_fault_result))
65 fft_hilbert_fault_result = np.abs(np.fft.fft(hilbert_fault_result))
66
67 # Frequências para a terceira FFT
68 freq_hilbert_no_fault = np.fft.fftfreq(len(hilbert_no_fault_result),
69     1 / fs)
70 # Frequências para a quarta FFT
71 freq_hilbert_fault = np.fft.fftfreq(len(hilbert_fault_result), 1 /
72     fs)
73
74 # Salvar os valores da terceira FFT em um arquivo de texto
75 with open('valores_fft_hilbert_primeira_CORRETO.txt', 'w') as file:
76     file.write("Frequencia,Intensidade,Grupo\n")
77     for freq, intensity in zip(freq_hilbert_no_fault,
78         fft_hilbert_no_fault_result):
79         file.write(f"{freq},{intensity},0\n") # 0 representa sem
80             falha
81
82 # Salvar os valores da quarta FFT em um arquivo de texto
83 with open('valores_fft_hilbert_segunda_CORRETO.txt', 'w') as file:
84     file.write("Frequencia,Intensidade,Grupo\n")
85     for freq, intensity in zip(freq_hilbert_fault,
86         fft_hilbert_fault_result):
87         file.write(f"{freq},{intensity},1\n") # 1 representa com
88             falha

```

```

81 # Criando subplots com 5 linhas e 1 coluna
82 fig = make_subplots(rows=5, cols=1, vertical_spacing=0.1)
83
84 # Adicionando cada gráfico em um subplot diferente
85 fig.add_trace(go.Scatter(x=t, y=vNoFaultNoisy, mode='lines',
86     name='Sem Falha'), row=1, col=1)
86 fig.add_trace(go.Scatter(x=t, y=vPinFaultNoisy, mode='lines',
87     name='Com Falha'), row=1, col=1)
87 fig.add_trace(go.Scatter(x=t, y=vNoFaultFiltered, mode='lines',
88     name='Sem Falha'), row=2, col=1)
88 fig.add_trace(go.Scatter(x=t, y=vFaultFiltered, mode='lines',
89     name='Com Falha'), row=2, col=1)
89 fig.add_trace(go.Scatter(x=t, y=hilbert_no_fault_result,
90     mode='lines', name='Sem Falha'), row=3, col=1)
90 fig.add_trace(go.Scatter(x=t, y=hilbert_fault_result, mode='lines',
91     name='Com Falha'), row=3, col=1)
91 fig.add_trace(go.Scatter(x=freq_no_fault, y=fft_no_fault_result,
92     mode='lines', name='Sem Falha'), row=4, col=1)
92 fig.add_trace(go.Scatter(x=freq_fault, y=fft_fault_result,
93     mode='lines', name='Com Falha'), row=4, col=1)
93 fig.add_trace(go.Scatter(x=freq_hilbert_no_fault,
94     y=fft_hilbert_no_fault_result, mode='lines', name='Sem Falha'),
95     row=5, col=1)
94 fig.add_trace(go.Scatter(x=freq_hilbert_fault,
95     y=fft_hilbert_fault_result, mode='lines', name='Com Falha'),
96     row=5, col=1)
96
97 # Adicionando títulos e rótulos dos eixos
98 fig.update_yaxes(title_text="Amplitude", tickfont=dict(size=16),
99     titlefont=dict(size=16), row=1, col=1)
100 fig.update_yaxes(title_text="Amplitude", tickfont=dict(size=16),
101     titlefont=dict(size=16), row=2, col=1)
102 fig.update_yaxes(title_text="Hilbert", tickfont=dict(size=16),
103     titlefont=dict(size=16), row=3, col=1)
104 fig.update_yaxes(title_text="FFT", tickfont=dict(size=16),
105     titlefont=dict(size=16), row=4, col=1)
106 fig.update_yaxes(title_text="FFT + Hilbert", tickfont=dict(size=16),
107     titlefont=dict(size=16), row=5, col=1)
107
108 fig.update_xaxes(title_text="Frequencia (Hz)",
109     tickfont=dict(size=16), titlefont=dict(size=16),
110     tickvals=np.arange(np.min(freq_hilbert_no_fault),
111         np.max(freq_hilbert_no_fault) + 1, 50), row=5, col=1)
112 fig.update_xaxes(title_text="Frequencia (Hz)",
113     tickfont=dict(size=16), titlefont=dict(size=16),
114     tickvals=np.arange(np.min(freq_no_fault), np.max(freq_no_fault) +
115         1, 50), row=4, col=1)
116 fig.update_xaxes(title_text="Tempo (s)", tickfont=dict(size=16),
117     titlefont=dict(size=16), tickvals=np.arange(np.min(t), np.max(t) +
118         1, 0.05), row=3, col=1)
119 fig.update_xaxes(title_text="Tempo (s)", tickfont=dict(size=16),
120     titlefont=dict(size=16), tickvals=np.arange(np.min(t), np.max(t) +
121         1, 0.05), row=2, col=1)

```

```

107 fig.update_xaxes(title_text="Tempo (s)", tickfont=dict(size=16),
108     titlefont=dict(size=16), tickvals=np.arange(np.min(t), np.max(t)
109     + 1, 0.5), row=1, col=1)
110
111 # Adicionando títulos aos subplots
112 fig.update_layout(
113     title="Análise de Sinais com e sem Falha",
114     height=2000,
115     width=1000
116 )
117
118 # Salvar o gráfico em um arquivo HTML
119 fig.write_html("graficos_com_titulos.html")
120
121 # Exibindo o endereço IP para acessar o arquivo HTML
122 import http.server
123 import socketserver
124 import threading
125
126 # Function to serve the HTML file
127 def serve_html():
128     PORT = 8000
129     Handler = http.server.SimpleHTTPRequestHandler
130     with socketserver.TCPServer(('127.0.0.1', PORT), Handler) as
131         httpd:
132             print(f"Serving at http://127.0.0.1:{PORT}")
133             httpd.serve_forever()
134
135 # Start serving the HTML file in a separate thread
136 server_thread = threading.Thread(target=serve_html)
137 server_thread.start()
138
139 # Print the URL to access the HTML file
140 print("Para visualizar os gráficos, acesse:
141     http://127.0.0.1:8000/graficos_com_titulos.html")

```

## B.2 Exemplo de Teste Simulação Caixa de Engrenagens

Código B.2 – Código de Teste no Octave

```

1 fs = 20E3;           % Sample Rate (Hz)
2
3 Np = 13;            % Number of teeth on pinion
4 Ng = 35;            % Number of teeth on gear
5
6 fPin = 22.5;        % Pinion (Input) shaft frequency (Hz)
7
8 fGear = fPin*Np/Ng; % Gear (Output) shaft frequency (Hz)
9
10 fMesh = fPin*Np;    % Gear Mesh frequency (Hz)

```

```

11 t = 0:1/fs:20-1/fs;
12
13
14 vfIn = 0.4*sin(2*pi*fPin*t); % Pinion waveform
15 vfOut = 0.2*sin(2*pi*fGear*t); % Gear waveform
16
17 vMesh = sin(2*pi*fMesh*t); % Gear-mesh waveform
18
19 plot(t, vfIn + vfOut + vMesh)
20 xlim([0 0.25])
21 xlabel('Time (s)')
22 ylabel('Acceleration')
23
24 ipf = fGear;
25 fImpact = 2000;
26
27 tImpact = 0:1/fs:2.5e-4-1/fs;
28 xImpact = sin(2*pi*fImpact*tImpact)/3;
29
30 xComb = zeros(size(t));
31
32 Ind = (0.25*fs/fMesh):(fs/ipf):length(t);
33 Ind = round(Ind);
34 xComb(Ind) = 1;
35
36 xPer = 2*conv(xComb, xImpact, 'same');
37
38 vNoFault = vfIn + vfOut + vMesh;
39 vFault = vNoFault + xPer;
40
41 vNoFaultNoisy = vNoFault + randn(size(t))/5;
42 vFaultNoisy = vFault + randn(size(t))/5;
43
44 subplot(2,1,1)
45 plot(t,vNoFaultNoisy)
46 xlabel('Time (s)')
47 ylabel('Acceleration')
48 xlim([0.0 0.3])
49 ylim([-2.5 2.5])
50 title('Noisy Signal for Healthy Gear')
51
52
53 subplot(2,1,2)
54 plot(t,vFaultNoisy)
55 xlabel('Time (s)')
56 ylabel('Acceleration')
57 xlim([0.0 0.3])
58 ylim([-2.5 2.5])
59 title('Noisy Signal for Faulty Gear')
60 hold on
61 MarkX = t(Ind(1:3));
62 MarkY = 2.5;
63 plot(MarkX,MarkY, 'rv', 'MarkerFaceColor', 'red')
```

```

64 hold off
65
66
67 figure
68
69 % Calculate Power Spectra
70 [Pxx1, f1] = periodogram(vFaultNoisy, hamming(length(vFaultNoisy)),
   length(vFaultNoisy), fs);
71 [Pxx2, f2] = periodogram(vNoFaultNoisy,
   hamming(length(vNoFaultNoisy)), length(vNoFaultNoisy), fs);
72
73 plot(f1, 10*log10(Pxx1), f2, 10*log10(Pxx2), ':')
74 xlabel('Frequency (Hz)')
75 ylabel('Power Spectrum (dB)')
76
77 hold on
78 plot(fGear, 0, 'rv', 'MarkerFaceColor', 'red')
79 plot(fPin, 0, 'gv', 'MarkerFaceColor', 'green')
80 plot(fMesh, 0, 'bv', 'MarkerFaceColor', 'blue')
81 hold off
82
83 legend('Faulty', 'Healthy', 'f_{Gear}', 'f_{Pinion}', 'f_{Mesh}')
84
85 figure
86 p1 = plot(f1, 10*log10(Pxx1));
87 xlabel('Frequency (Hz)')
88 ylabel('Power Spectrum (dB)')
89 xlim([250 340])
90 ylim([-70 -40])
91
92 hold on
93 p2 = plot(f2, 10*log10(Pxx2));
94
95 harmonics = -5:5;
96 SBandsGear = (fMesh+fGear.*harmonics);
97 [X1,Y1] = meshgrid(SBandsGear,ylim);
98
99 SBandsPinion = (fMesh+fPin.*harmonics);
100 [X2,Y2] = meshgrid(SBandsPinion,ylim);
101
102 p3 = plot(X1,Y1,:r');
103 p4 = plot(X2,Y2,:k');
104 hold off
105 legend([p1 p2 p3(1) p4(1)],{'Faulty Gear';'Healthy
   Gear';'f_{sideband,Gear}';'f_{sideband,Pinion}'})
106
107 tPulseIn = 0:1/fPin:max(t);
108 taPin = tsa(vFaultNoisy,fs,tPulseIn,'NumRotations',10);
109
110 tPulseOut = 0:1/fGear:max(t);
111 taGear = tsa(vFaultNoisy,fs,tPulseOut,'NumRotations',10);
112
113 figure

```

```

114 subplot(2,1,1)
115 tsa(vFaultNoisy,fs,tPulseIn,'NumRotations',10)
116 xlim([0.5 1.5])
117 ylim([-2 2])
118 title('TSA Signal for Pinion')
119
120 subplot(2,1,2)
121 tsa(vFaultNoisy,fs,tPulseOut,'NumRotations',10)
122 xlim([0.5 1.5])
123 ylim([-2 2])
124 title('TSA Signal for Gear')
125 hold on
126 plot(1.006,2,'rv','MarkerFaceColor','red')
127 hold off
128
129 figure
130 pspectrum(taGear,fs,'FrequencyResolution',2.2,'FrequencyLimits',[200
    400])
131
132 harmonics = -15:15;
133 SBandsGear=(fMesh+fGear.*harmonics);
134
135 [X1,Y1] = meshgrid(SBandsGear,ylim);
136 [XM,YM] = meshgrid(fMesh,ylim);
137
138 hold on
139 plot(XM,YM,'--k',X1,Y1,:r')
140 legend('Power Spectra','Gear-Mesh Frequency','f_{sideband,Gear}')
141 hold off
142
143 title('TSA Gear (Output Shaft)')
144
145 figure
146 pspectrum(taPin,fs,'FrequencyResolution',5.8,'FrequencyLimits',[200
    400])
147
148 SBandsPinion = (fMesh+fPin.*harmonics);
149
150 [X2,Y2] = meshgrid(SBandsPinion,ylim);
151 [XM,YM] = meshgrid(fMesh,ylim);
152
153 hold on
154 plot(XM,YM,'--b',X2,Y2,:k')
155 legend('Power Spectra','Gear-Mesh Frequency','f_{sideband,Pinion}')
156 hold off
157
158 title('TSA Pinion (Input Shaft)')
159

```

### B.3 Teste Embarcando Modelo na Esp32

### Código B.3 – Código no Arduino IDE

```

1 #include <Arduino.h>
2 #include <sys/time.h>
3
4 // Estrutura para armazenar os valores do dataset
5 typedef struct {
6     float frequency;
7     float intensity;
8 } DatasetValue;
9
10 // Função para ler os valores do dataset
11 void readDataset(DatasetValue *dataset) {
12     float values[][] = {
13         { 5129.15, 0.40069}, //falho
14         { -9708.96, 0.001236 }, //saudavel
15
16     };
17     int num_values = sizeof(values) / sizeof(values[0]);
18
19     for (int i = 0; i < num_values; i++) {
20         dataset[i].frequency = values[i][0];
21         dataset[i].intensity = values[i][1];
22     }
23 }
24
25 int predictFailure(DatasetValue *dataset, int num_values) {
26     //Serial.println("Inicio do teste: ");
27     for (int i = 1; i <= num_values; i++) {
28         float frequency = dataset[i].frequency;
29         float intensity = dataset[i].intensity;
30         //Serial.println("");
31         //Serial.print("Teste: ");
32         //Serial.println(i);
33         //Serial.print(" ");
34
35         if (intensity <= 0.080193) {
36             if (intensity <= 0.001235) {
37                 Serial.println("A máquina está saudável.");
38             } else { // intensidade > 0.001235
39                 if (frequency <= -9708.95) {
40                     Serial.println("A máquina está falha.");
41                 } else if (frequency <= 9708.95) { // frequencia >
42                     -9708.95 e <= 9708.95
43                     Serial.println("A máquina está saudável.");
44                 } else { // frequencia > 9708.95
45                     Serial.println("A máquina está falha.");
46                 }
47             }
48         } else { // intensidade > 0.080193
49             if (frequency <= -5285.9) {
50                 Serial.println("A máquina está falha.");
51             }
52         }
53     }
54 }
```

```
50 } else if (frequency <= 5129.15) { // frequencia > -5285.9 e <=
51     5129.15
52     if (intensity <= 0.40069) {
53         Serial.println("A máquina está saudável.");
54     } else if (frequency <= -3840.9) { // frequencia > -5285.9 e
55         <= 5129.15 e > -3840.9
56         Serial.println("A máquina está falha.");
57     } else if (frequency <= 3826.8) { // frequencia > -5285.9 e
58         <= 5129.15 e > -3840.9 e <= 3826.8
59         if (intensity <= 1.484293) {
60             Serial.println("A máquina está saudável.");
61         } else if (frequency <= -3242.95) { // frequencia >
62             -5285.9 e <= 5129.15 e > -3840.9 e <= 3826.8 e <=
63             -3242.95
64             Serial.println("A máquina está falha.");
65     } else if (frequency <= 3229.25) { // frequencia >
66         -5285.9 e <= 5129.15 e > -3840.9 e <= 3826.8 e >
67         -3242.95 e <= 3229.25
68         if (intensity <= 2.96571) {
69             Serial.println("A máquina está saudável.");
70         } else if (frequency <= -2645.05) { // frequencia >
71             -5285.9 e <= 5129.15 e > -3840.9 e <= 3826.8 e >
72             -3242.95 e <= 3229.25 e <= -2645.05
73             Serial.println("A máquina está falha.");
74     } else if (frequency <= 2645) { // frequencia >
75         -5285.9 e <= 5129.15 e > -3840.9 e <= 3826.8 e >
-3242.95 e <= 3229.25 e > -2645.05 e <= 2645
if (intensity <= 5.516686) {
    Serial.println("A máquina está saudável.");
} else if (frequency <= -2333.6) { // frequencia
> -5285.9 e <= 5129.15 e > -3840.9 e <=
3826.8 e > -3242.95 e <= 3229.25 e > -2645.05
e <= 2645 e <= -2333.6
Serial.println("A máquina está falha.");
} else if (frequency <= 2333.1) { // frequencia
> -5285.9 e <= 5129.15 e > -3840.9 e <=
3826.8 e > -3242.95 e <= 3229.25 e > -2645.05
e <= 2645 e > -2333.6 e <= 2333.1
if (intensity <= 10.788543) {
    Serial.println("A máquina está
saudável.");
} else if (frequency <= -2172.95) { // frequencia > -5285.9 e <= 5129.15 e >
-3840.9 e <= 3826.8 e > -3242.95 e <=
3229.25 e > -2645.05 e <= 2645 e >
-2333.6 e <= 2333.1 e <= -2172.95
Serial.println("A máquina está falha.");
} else if (frequency <= 2172.9) { // frequencia > -5285.9 e <= 5129.15 e >
-3840.9 e <= 3826.8 e > -3242.95 e <=
3229.25 e > -2645.05 e <= 2645 e >
-2333.6 e <= 2333.1 e > -2172.95 e <=
2172.9
```

```
76         if (intensity <= 19.82965) {
77             Serial.println("A máquina está
78             saudável.");
79         } else if (frequency <= -2013) { //
80             frequencia > -5285.9 e <= 5129.15 e >
81             -3840.9 e <= 3826.8 e > -3242.95 e <=
82             3229.25 e > -2645.05 e <= 2645 e >
83             -2333.6 e <= 2333.1 e > -2172.95 e <=
84             2172.9 e <= -2013
85             Serial.println("A máquina está
86             falha.");
87         } else if (frequency <= 2012.95) { //
88             frequencia > -5285.9 e <= 5129.15 e >
89             -3840.9 e <= 3826.8 e > -3242.95 e <=
90             3229.25 e > -2645.05 e <= 2645 e >
91             -2333.6 e <= 2333.1 e > -2172.95 e <=
92             2172.9 e > -2013 e <= 2012.95
93             if (intensity <= 33.758874) {
94                 Serial.println("A máquina está
95                 saudável.");
96             } else { // intensidade > 33.758874
97                 Serial.println("A máquina está
98                 falha.");
99             }
100         } else { // frequencia > 2172.9
101             Serial.println("A máquina está falha.");
102         }
103     } else { // frequencia > 2333.1
104         Serial.println("A máquina está falha.");
105     }
106 }
107 }
108 }
109 //}
110 //Serial.println("");
111 //Serial.println("Termino do Dataset");
112 //Serial.println("");
113 return 0;
```

```

114 }
115
116
117 void setup() {
118     Serial.begin(9600); // Inicializa a comunicação serial
119 }
120
121 void loop() {
122     struct timeval start, end;
123     gettimeofday(&start, NULL);
124
125     DatasetValue dataset[18]; // Tamanho do dataset
126
127     readDataset(dataset);
128     int result = 0;
129     result = predictFailure(dataset, 2);
130     // Imprime o resultado
131
132     gettimeofday(&end, NULL);
133     Serial.print("Tempo de execução: ");
134     Serial.print(((end.tv_sec * 1000000 + end.tv_usec) - (start.tv_sec
135         * 1000000 + start.tv_usec)));
136     Serial.println(" microsegundos");
137     Serial.println("");
138
139     delay(1000); // Aguarda 1 segundo antes da próxima iteração
}

```

## B.4 Código VHDL do *top module*

Código B.4 – Código VHDL no Vivado

```

1 -----
2 -- Company: University of Brasília
3 -- Engineer: Natlia Schulz Teixeira
4 --
5 -- Create Date: 02/14/2025 03:53:47 PM
6 -- Module Name: faulty_detection - Behavioral
7 -- Project Name: Implementação de Modelo para Detecção de Falhas em
8     M quinas Rotativas com FPGA
9 --
10 -- Revision:
11 -- Revision 0.01 - File Created
12 -- Additional Comments:
13 --
14
15 library IEEE;
16 use IEEE.STD_LOGIC_1164.ALL;
17 use IEEE.NUMERIC_STD.ALL;

```

```

18
19 entity top_module is
20   Port (
21     clk      : in STD_LOGIC;
22     resetn  : in STD_LOGIC;
23     m_axis_data_tready : in STD_LOGIC; -- interno
24     m_axis_data_tvalid : out STD_LOGIC;
25     m_axis_data_tdata  : out STD_LOGIC_VECTOR(31 downto 0);
26     m_axis_data_tlast  : out STD_LOGIC;
27     s_axis_data_tready : out STD_LOGIC; -- Alterado para 'out'
28     start    : in STD_LOGIC; -- Sinal para iniciar a configura
29               da FFT
30     auxtready : out STD_LOGIC -- Sinal auxiliar para tready
31   --      led : out STD_LOGIC -- Sinal auxiliar para tready
32 );
33 end top_module;
34
35
36 architecture Behavioral of top_module is
37
38   signal s_axis_data_tvalid_int : STD_LOGIC; -- Sinal interno
39               para tvalid
40   signal s_axis_data_tdata_int  : STD_LOGIC_VECTOR(31 downto 0);
41               -- Sinal interno para tdata
42   signal s_axis_data_tlast_int : STD_LOGIC :='0'; -- Sinal
43               interno para tlast
44   signal s_axis_data_tready_int : STD_LOGIC; -- Sinal interno
45               para tready
46   signal tlast_memory : STD_LOGIC:='0'; -- Sinal interno para
47               tready
48   signal start_int : STD_LOGIC; -- Sinal interno para start
49   -- Sinais internos para configura da FFT
50   signal s_axis_config_tdata_int : STD_LOGIC_VECTOR(23 downto 0);
51               -- Sinal interno para tdata
52   signal s_axis_config_tvalid_int : STD_LOGIC; -- Sinal interno
53               para tvalid
54   signal s_axis_config_tready_int : STD_LOGIC; -- Sinal interno
55               para tready
56   signal config_reg : STD_LOGIC_VECTOR(11 downto 0) := (others =>
57               '0');
58   signal log2nFFT    : STD_LOGIC_VECTOR(3 downto 0) := (others =>
59               '0');
60   signal FWD         : STD_LOGIC_VECTOR(4 downto 0) := "10000"; --
61               FFT
62   signal INV         : STD_LOGIC_VECTOR(4 downto 0) := "00000"; --
63               IFFT
64   -- Sinais para configura da FFT
65   signal event_frame_started : STD_LOGIC;
66   signal event_tlast_unexpected : STD_LOGIC;
67   signal event_tlast_missing   : STD_LOGIC;
68   signal event_status_channel_halt : STD_LOGIC;
69   signal event_data_in_channel_halt : STD_LOGIC;
70   signal event_data_out_channel_halt : STD_LOGIC;
71
72

```

```

58     signal m_axis_data_tdata_int : STD_LOGIC_VECTOR(31 downto 0);
59
60     -- Sinal para o limiar
61     signal threshold : STD_LOGIC_VECTOR(15 downto 0) := x"7FFF";
62
63     -- Sinais adicionais
64     signal lock : STD_LOGIC := '0'; -- Sinal de lock para controle
65           de configura
66
67     -- Componente datasrc
68     component datasrc
69         Port (
70             clk      : in STD_LOGIC;
71             resetn : in STD_LOGIC;
72             tready : in STD_LOGIC;
73             start   : in STD_LOGIC;
74             tvalid  : out STD_LOGIC;
75             tlast   : out STD_LOGIC;
76             tdata   : out STD_LOGIC_VECTOR(31 downto 0)
77         );
78     end component;
79
80     -- Componente FFT (substitua pelo nome correto do seu IP FFT)
81     component xfft_0
82         Port (
83             -- Interface de entrada de dados (S_AXIS_DATA)
84             s_axis_data_tdata  : in STD_LOGIC_VECTOR(31 downto 0);
85             s_axis_data_tlast  : in STD_LOGIC;
86             s_axis_data_tready : out STD_LOGIC;
87             s_axis_data_tvalid : in STD_LOGIC;
88             s_axis_config_tdata : in STD_LOGIC_VECTOR(23 downto 0);
89             s_axis_config_tready : out STD_LOGIC;
90             s_axis_config_tvalid : in STD_LOGIC;
91             m_axis_data_tdata  : out STD_LOGIC_VECTOR(31 downto 0);
92             m_axis_data_tlast  : out STD_LOGIC;
93             m_axis_data_tready : in STD_LOGIC;
94             m_axis_data_tvalid : out STD_LOGIC;
95             aclk      : in STD_LOGIC;
96             aresetn : in STD_LOGIC;
97             event_frame_started : out STD_LOGIC;
98             event_tlast_unexpected : out STD_LOGIC;
99             event_tlast_missing : out STD_LOGIC;
100            event_status_channel_halt : out STD_LOGIC;
101            event_data_in_channel_halt : out STD_LOGIC;
102            event_data_out_channel_halt : out STD_LOGIC
103        );
104    end component;
105
106    -- component faulty_detection is
107    --     Port (
108    --         clk          : in STD_LOGIC;
109    --         resetn       : in STD_LOGIC;

```

```

109      fft_data      : in  STD_LOGIC_VECTOR(31 downto 0);  --
110      Dados de sada da FFT
111      fft_valid     : in  STD_LOGIC;                      --
112      Sinal de valida dos dados da FFT
113      fft_last      : in  STD_LOGIC;                      --
114      Sinal de ltimo dado da FFT
115      led_fault     : out STD_LOGIC;                      --
116      LED que indica falha
117      threshold     : in  STD_LOGIC_VECTOR(15 downto 0)  --
118      Limiar para detec de pico
119      );
120  end component;
121
122 begin
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
-- Instancia do datasrc
datasrc_inst : datasrc
    Port map (
        clk      => clk,
        start   => start_int,
        resetn  => resetn,
        tready  => s_axis_data_tready_int,  -- Conecta ao sinal
                                                interno tready
        tvalid  => s_axis_data_tvalid_int,  -- Conecta ao sinal
                                                interno tvalid
        tdata   => s_axis_data_tdata_int,   -- Conecta ao sinal
                                                interno tdata
        tlast   => tlast_memory
    );
-- Instancia do FFT IP
xfft_0_inst : xfft_0
    Port map (
        -- Interface de entrada de dados
        s_axis_data_tdata  => s_axis_data_tdata_int,  -- Conecta
                                                ao sinal interno tdata
        s_axis_data_tlast  => s_axis_data_tlast_int,  -- Conecta
                                                ao sinal interno tlast
        s_axis_data_tready => s_axis_data_tready,    -- Conecta ao
                                                sinal interno tready
        s_axis_data_tvalid => s_axis_data_tvalid_int,  -- Conecta ao
                                                sinal interno tvalid
        s_axis_config_tdata => s_axis_config_tdata_int, -- Conecta
                                                ao sinal interno tdata
        s_axis_config_tready => s_axis_config_tready_int, -- Conecta
                                                ao sinal interno tready
        s_axis_config_tvalid => s_axis_config_tvalid_int, -- Conecta
                                                ao sinal interno tvalid
        m_axis_data_tdata  => m_axis_data_tdata,
        m_axis_data_tlast  => m_axis_data_tlast,
        m_axis_data_tready => m_axis_data_tready,
        m_axis_data_tvalid => m_axis_data_tvalid,
        aclk      => clk,

```

```

147         aresetn => resetn,
148         event_frame_started => event_frame_started,
149         event_tlast_unexpected => event_tlast_unexpected,
150         event_tlast_missing => event_tlast_missing,
151         event_status_channel_halt => event_status_channel_halt,
152         event_data_in_channel_halt => event_data_in_channel_halt,
153         event_data_out_channel_halt =>
154             event_data_out_channel_halt
155     );
156 
156 --      -- Instancia do indicador de falha
157 --      faulty_detection_inst : faulty_detection
158 --      Port map (
159 --          clk      => clk,
160 --          resetn   => resetn,
161 --          fft_data  => m_axis_data_tdata,
162 --          fft_valid => m_axis_data_tvalid,
163 --          fft_last   => m_axis_data_tlast,
164 --          threshold  => threshold,
165 --          led_fault  => led  -- Conecte ao pino do LED no top
166 --          level
166 --      );
167 
168 --      -- Conecta o sinal interno tready ao sinal de sada
169 --      s_axis_data_tready <= s_axis_data_tready_int;
170 
171 start_int <= start;
172 
173 process(clk, resetn)
174 begin
175     if resetn = '0' then
176         auxready <= '0';
177         lock <= '0';
178         s_axis_config_tdata_int <= (others => '0');
179         s_axis_config_tvalid_int <= '0';
180         s_axis_data_tlast_int <= '0';
181     elsif rising_edge(clk) then
182         s_axis_config_tvalid_int <= '0';
183 --         if start_int = '1' and s_axis_config_tready_int = '1'
184 --             and lock = '0' then
185 --                 if start_int = '1' and lock = '0' then
186 --                     s_axis_config_tdata_int <= "000" & "00110101010" &
187 --                         FWD & "01111"; -- 24 bits tirei um zero de
188 --                         000110101010
189 --                     s_axis_config_tvalid_int <= '1';
190 --                     lock <= '1';
191 --                 end if;
192 
193         if lock = '1' then
194             s_axis_data_tready_int <= '1';
195             auxready <= '1';
196         end if;
197 
```

```

195      --          if s_axis_data_tlast_int = '1' then
196          if tlast_memory = '1' then
197              s_axis_data_tready_int <= '0';
198              lock <= '0';
199          end if;
200      end if;
201  end process;
202
203 end Behavioral;

```

## B.5 Código Verilog da memória

Código B.5 – Código Verilog no Vivado

```

1 `timescale 1ns / 1ps
2
3 module datasrc(
4     input clk,
5     input start,           // Sinal de start para iniciar a leitura e
6         envio de dados.
7     input resetn,
8     input tready,
9     output tvalid,
10    output [31:0] tdata,
11    output tlast
12 );
13
14     parameter infile = "input_sinal_tempo.mem";
15
16     reg [31:0] mem[0:32767]; // Memoria com 32768 posies (0 a
17         32767)
18     localparam s0 = 'd0;      // Estado inicial (aguardando start).
19     localparam s1 = 'd1;      // Estado de leitura e prepara dos
20         dados.
21     localparam s2 = 'd2;      // Estado de envio dos dados.
22     localparam s3 = 'd3;      // Estado de espera (quando tready =
23         0).
24
25     reg [2:0] state, n_state;
26     reg [14:0] addr, n_addr; // Endereo de 15 bits para suportar
27         32768 posies
28     reg [31:0] d0, d1, d2, n_d1, n_d2;
29     reg valid, n_valid;
30     reg tlast_reg;          // Registro para armazenar o valor de
31         tlast
32
33     initial begin
34         $readmemh(infile, mem); // Inicializa a memoria com os
35             dados do arquivo
36         state = s0;           // Inicializa o estado

```



```

77      end else begin
78          n_state = s0;           // Permanece no estado s0 at
79              que start seja ativado.
80      end
81  s1: begin
82      n_d1 = d0;           // Carrega o proximo dado
83          (d0) em n_d1.
84      n_addr = addr + 1;    // Incrementa o endereco para
85          apontar para o proximo dado na memoria.
86      n_state = s2;           // Define o proximo estado
87          como s2.
88      n_valid = 1;           // Ativa o sinal de validade
89          (n_valid = 1).
90  end
91  s2: begin
92      n_d1 = d0;           // Carrega o proximo dado
93          (d0) em n_d1.
94      n_d2 = d1;           // Carrega o dado atual (d1)
95          em n_d2 (dado a ser enviado).
96      n_valid = 1;           // Mantm o sinal de validade
97          ativo (n_valid = 1).
98      if (tready) begin // Se o receptor estiver
99          pronto (tready = 1):
100          if (addr == 32767) begin // Verifica se o
101              endereco atual o ltimo (32767).
102              n_state = s0; // Retorna ao estado inicial
103                  aps o ltimo dado.
104          end else begin
105              n_addr = addr + 1; // Incrementa o endereco
106                  para o proximo dado.
107              n_state = s2; // Mantm o estado como s2
108                  para enviar o proximo dado.
109          end
110      end
111  s3: begin
112      if (tready) begin // Se o receptor estiver
113          pronto (tready = 1):
114              n_addr = addr + 1; // Incrementa o endereco para
115                  o proximo dado.
116              n_state = s2; // Volta para o estado s2
117                  para enviar o proximo dado.
118      end else begin // Se o receptor no estiver
119          pronto (tready = 0):
120              n_state = s3; // Muda para o estado s3 para
121                  esperar.
122      end
123  end

```

```

110      default: begin
111          n_state = s0;           // Retorna ao estado inicial
112          s0.
113      end
114  endcase
115 end

```

## B.6 Código VHDL para detecção de falhas

Código B.6 – Código VHDL no Vivado

```

1 -----
2 -- Company: University of Brasilia
3 -- Engineer: Natlia Schulz Teixeira
4 --
5 -- Create Date: 02/14/2025 03:53:47 PM
6 -- Module Name: faulty_detection - Behavioral
7 -- Project Name: Implementa de Modelo para Detec de Falhas em
8     Mquinas Rotativas com FPGA
9 --
10 -- Revision:
11 -- Revision 0.01 - File Created
12 -- Additional Comments:
13 --
14 -----
15 library IEEE;
16 use IEEE.STD_LOGIC_1164.ALL;
17 use IEEE.NUMERIC_STD.ALL;
18
19 entity faulty_detection is
20     Port (
21         clk          : in STD_LOGIC;
22         resetn       : in STD_LOGIC;
23         fft_data     : in STD_LOGIC_VECTOR(31 downto 0);    -- Dados
24             de sada da FFT
25         fft_valid    : in STD_LOGIC;                         -- Sinal
26             de valida dos dados da FFT
27         fft_last     : in STD_LOGIC;                         -- Sinal
28             de ltimo dado da FFT
29         threshold   : in STD_LOGIC_VECTOR(15 downto 0);    -- Limiar
30             para detec de pico
31         led_fault   : out STD_LOGIC;                        -- LED
32             que indica falha
33     );
34 end faulty_detection;
35
36 architecture Behavioral of faulty_detection is

```

```

33      -- Constantes para a faixa de frequncia de 22 kHz a 40 kHz
34      constant FREQ_LOW : integer := 22000;    -- 22 kHz
35      constant FREQ_HIGH : integer := 40000;   -- 40 kHz
36
37      -- Constante para o tempo de 5 segundos
38      constant TIMER_5S : integer := 500000000;  -- 5 segundos em
39          ciclos de clock (assumindo 100 MHz)
40
41      -- Sinais internos
42      signal magnitude : unsigned(15 downto 0) := (others => '0');
43          -- Magnitude da frequncia
44      signal frequency  : integer := 0;           -- Frequncia atual
45      signal fault_flag : STD_LOGIC := '0';       -- Flag para indicar
46          falha
47      signal timer      : integer := 0;           -- Contador de tempo
48      signal led_state   : STD_LOGIC := '0';       -- Estado do LED
49      signal threshold_int : STD_LOGIC_VECTOR(15 downto 0);  --
50          Limiar para detec de pico
51
52 begin
53
54     process(clk, resetn)
55     begin
56         if resetn = '0' then
57             magnitude <= (others => '0');
58             frequency <= 0;
59             fault_flag <= '0';
60             led_state <= '0';
61             timer <= 0;
62         elsif rising_edge(clk) then
63             if fft_valid = '1' then
64                 -- Extrai a magnitude do dado da FFT (assumindo que
65                 -- os 16 bits mais significativos so a parte real)
66                 magnitude <= unsigned(fft_data(31 downto 16));
67
68                 -- Verifica se a frequncia atual est na faixa
69                 -- desejada
70                 if frequency >= FREQ_LOW and frequency <= FREQ_HIGH
71                     then
72                         -- Verifica se a magnitude est acima do limiar
73                         if magnitude > unsigned(threshold) then
74                             fault_flag <= '1';  -- Ativa a flag de falha
75                         end if;
76                     end if;
77
78                 -- Incrementa a frequncia para o proximo dado
79                 frequency <= frequency + 1;
80
81                 -- Reseta a frequncia quando o ltimo dado
82                 -- recebido
83                 if fft_last = '1' then
84                     frequency <= 0;

```

```
78      if fault_flag = '1' then
79          led_state <= '1';    -- Acende o LED
80          timer <= 0;        -- Reseta o contador de
81              tempo
82      end if;
83      fault_flag <= '0';    -- Reseta a flag de falha
84  end if;
85
86      -- Controle do tempo que o LED fica aceso
87  if led_state = '1' then
88      if timer < TIMER_5S then
89          timer <= timer + 1; -- Incrementa o contador
90      else
91          led_state <= '0';    -- Desliga o LED aps 5
92              segundos
93      end if;
94  end if;
95 end process;
96
97      -- Atribui o estado do LED ao sinal de saida
98  led_fault <= led_state;
99
100 end Behavioral;
```