UNIVERSIDADE DO ESTADO DO AMAZONAS - UEA
ESCOLA SUPERIOR DE TECNOLOGIA
ENGENHARIA ELÉTRICA

KEVEN SOARES DA COSTA

DISPOSITIVO DE SIMULAÇÃO DE TRANSFORMADOR TRIFÁSICO PARA VALIDAÇÃO DE CIRCUITOS DE MEDIÇÃO DE ENERGIA NA ETAPA DE PÓS CONDICIONAMENTO

Manaus
2023

KEVEN SOARES DA COSTA

DISPOSITIVO DE SIMULAÇÃO DE TRANSFORMADOR TRIFÁSICO PARA VALIDAÇÃO DE CIRCUITOS DE MEDIÇÃO DE ENERGIA NA ETAPA DE PÓS CONDICIONAMENTO

Pesquisa desenvolvida durante a disciplina de Trabalho de Conclusão de Curso II e apresentada à banca avaliadora do Curso de Engenharia Elétrica da Escola Superior de Tecnologia da Universidade do Estado do Amazonas, como pré-requisito para a obtenção do título de Engenheiro Eletricista.

Orientador: Jozias Parente de Oliveira, Dr

Manaus
2023

Universidade do Estado do Amazonas – UEA
Escola Superior de Tecnologia - EST

Reitor:
André Luiz Nunes Zogahib
Vice-Reitor:
Kátia do Nascimento Couceiro
Diretor da Escola Superior de Tecnologia:
Ingrid Sammyne Gadelha Figueiredo
Coordenador do Curso de Engenharia Elétrica:
Israel Gondres Torné

Banca Avaliadora composta por: *Data da defesa: 17/03/2023.*
Prof. Jozias Parente de Oliveira, Dr *(Orientador)*
Prof. Fábio de Sousa Cardoso, Dr
Prof. Rubens de Andrade Fernandes, Ms

CIP – Catalogação na Publicação

da Costa, Keven Soares

DISPOSITIVO DE SIMNULAÇÃO DE TRANSFORMADOR TRIFÁSICO PARA VALIDAÇÃO DE CIRCUITOS DE MEDIÇÃO DE ENERGIA NA ETAPA DE PÓS CONDICIONAMENTO/ KEVEN SOARES DA COSTA; [orientado por] Jozias Parente de Oliveira. – Manaus: 2023.
121 p.: il.

Trabalho de Conclusão de Curso (Graduação em Engenharia Elétrica). Universidade do Estado do Amazonas, 2023.

1. Distorção Harmônica. 2. Transformada rápida de Fourier. 3. Microcontrolador. I. Oliveira, Jozias Parente de.

KEVEN SOARES DA COSTA

DISPOSITIVO DE SIMULAÇÃO DE TRANSFORMADOR TRIFÁSICO PARA VELIDAÇÃO DE CIRCUITOS DE MEDIÇÃO DE ENERGIA NA ETAPA DE PÓS CONDICIONAMENTO

Pesquisa desenvolvida durante a disciplina de Trabalho de Conclusão de Curso II e apresentada à banca avaliadora do Curso de Engenharia Elétrica da Escola Superior de Tecnologia da Universidade do Estado do Amazonas, como pré-requisito para a obtenção do título de Engenheiro Eletricista.

Nota obtida: 10,0 (Dez vírgula zero)

Aprovada em 17 / 03 / 2023.

Área de concentração: Sistemas Embarcados

BANCA EXAMINADORA

Orientador: Jezias Parente de Oliveira, Dr.

Avaliador: Fábio de Sousa Cardoso, Dr.

Avaliador: Rubens de Andrade Fernandes, Ms.

Manaus 2023

Dedicatória

A memória de Claudionor Ramos, incrível professor com quem tive a honra de aprender e o privilégio de chamar de amigo. Dedico-lhe essa conquista, como forma de gratidão.

Agradecimentos

Agradeço a minha companheira Bianca, que ao longo de vários anos sempre me manteve no caminho certo e sempre estimulou a crescer como pessoa e profissional.
Agradeço a minha mãe, em meio a todas as intercorrências da vida sempre me apoiou.
Agradeço ao meu orientador Jozias, por ter me ajudado com tanta dedicação e paciência.
Agradeço aos meus amigos Keven, Gabriel e Mateus, pela ajuda paciência e conhecimentos compartilhados para esse trabalho.

RESUMO

O Sistema de Gestão da Operação e Combate às Perdas em Rede de Distribuição de Energia Elétrica de Baixa Tensão (SGRede-BT) é responsável por monitorar remota e em tempo real os parâmetros elétricos e físicos em cada circuito da rede elétrica de baixa tensão por meio de unidades remotas de medição e aquisição de dados (URs), software de telesupervisão e banco de dados. Durante o processo de desenvolvimento do firmware dos medidores de energia da empresa, por vezes os desenvolvedores ficam expostos ao risco de choque elétrico, haja vista que precisam realizar ligações em pontos de comunicação que ficam próximos a partes vivas do medidor. Este projeto teve como objetivo desenvolver um protótipo capaz gerar sinais que reproduzam o comportamento de um transformador trifásico em funcionamento normal, e em eventos, tais como falta de fase, sobretensão, subtensão, e harmônicas de até 40ª ordem, mas gerando esses sinais com amplitude e potências irrisórias visando dessa forma eliminar o risco de choque elétrico. Para isso, foi utilizado um sistema composto por uma placa de circuito impresso com microcontrolador, memória e conversores digital/analógico, e um software de controle com capacidade de realizar os cálculos necessários para gerar as formas de onda que reproduzam os eventos supracitados, criando um ambiente confiável e seguro para realização dos testes de firmware do medidor de energia. Como principais resultados é possível mencionar o sucesso na implementação de uma versão funcional do *hardware*, *firmware* e *software* do dispositivo, esse capaz de gerar sinais que não apresentam risco de choque elétrico para o usuário e compatíveis com os dispositivos de medição de energia.

Palavras-Chave: Choque Elétrico, Conversor Digital/Analógico, Microcontrolador.

ABSTRACT

The Management System for Operations and Combating Losses in the Low Voltage Electricity Distribution Network (SGRede-BT) is responsible for remotely monitoring, in real time, the electrical and physical parameters in each circuit of the low voltage electricity network through of remote measurement and data acquisition units (RUs), telesupervision software and database. During the firmware development process for the company's energy meters, developers are sometimes exposed to the risk of electric shock, given that they need to make connections at communication points that are close to live parts of the meter. The purpose of this project is to develop a prototype capable of generating signals that reproduce the behavior of a three-phase transformer in normal operation, and in events such as phase loss, overvoltage, undervoltage, and harmonics of up to 40th order, but generating these signals with negligible amplitude and powers, thus eliminating the risk of electric shock. For this, a system composed of a printed circuit board with a microcontroller, memory and digital/analog converters, and a control software capable of performing the necessary calculations to generate the waveforms that reproduce the aforementioned events will be used. Creating a reliable and secure environment for performing power meter firmware tests. As main results, it is possible to mention the successful implementation of a functional version of the hardware, firmware and software of the device, capable of generating signals that do not pose a risk of electric shock to the user and compatible with energy measurement devices.

Keywords: Electric Shock, Digital/Analog Converter, Microcontroller.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

INTRODUÇÃO

O tema deste trabalho é dispositivo de simulação de transformador trifásico para validação de circuitos de medição de energia na etapa de pós condicionamento.

O Sistema de Gestão da Operação e Combate às Perdas em Rede de Distribuição de Energia Elétrica de Baixa Tensão, indicado pela sigla SGRede-BT, tem como finalidade realizar o monitoramento remoto e em tempo real dos parâmetros elétricos e físicos em cada circuito da rede elétrica de baixa tensão (VÓRTICE TECNOLOGIA E INOVAÇÃO).

Composto por Unidades Remotas de Medição e Aquisição de Dados (UR's), Software de Telesupervisão e Banco de Dados, o sistema monitora as condições de operação da rede e os saldos de energia fornecida e faturada, direcionando medidas efetivas para combater perdas e melhorar a rede (VÓRTICE TECNOLOGIA E INOVAÇÃO).

As unidades remotas do SGRede são dispositivos inteligentes que, instalados próximos a transformadores, funcionam de forma independente utilizando o conceito de processamento distribuído e realizam coleta e análise de dados da rede de distribuição (VÓRTICE TECNOLOGIA E INOVAÇÃO).

Verificação, teste e depuração compõe uma etapa do ciclo de vida de um firmware. Verificação significa uma tentativa de provar que o código está livre de erros. O teste ajuda você a entender suas tentativas de encontrar erros em seu código. Depurar significa encontrar e corrigir erros encontrados em seu código (DAVIDSON e SHRIVER, 1978).

O interesse em validar software e firmware, entre outros coisas, levou ao estudo de métodos de verificação de software. Estes incluem a execução simbólica do programa, que pode demonstrar acurácia para uma série de entradas, e provas de acurácia, geralmente feitas realizando afirmações sobre o comportamento de entrada/saída do programa e, em seguida, provando algebricamente que essas afirmações refletem com precisão o comportamento real do programa. A maior dificuldade com esses métodos até agora é a complexidade de programas não triviais, que muitas vezes tem um grande número de caminhos de execução, que por sua vez, tornam as asserções muito complexas para os métodos atuais manipularem (DAVIDSON e SHRIVER, 1978).

Como ficou claro, o interesse em realizar testes de software e firmware é algo antigo e que, dependendo do número de variáveis, pode ser algo bastante complexo, mas além desse ponto, algo intrínseco ao desenvolvimento das UR'S é o perigo de exposição ao choque elétrico.

O choque elétrico é uma perturbação de natureza e efeitos diversos que se manifesta no organismo humano ou animal quando percorrido por uma corrente elétrica, corrente de choque, provocando efeitos de importância e gravidades variáveis, bem como fatais. Os efeitos da corrente elétrica, percorrendo a resistência ôhmica do corpo humano, podem causar diversas perturbações ou lesões no organismo, cuja atividade dependerá do tempo de duração, da intensidade e da natureza da corrente, do percurso da corrente no corpo humano e das condições orgânicas do indivíduo acidentado (FUNDACENTRO, 2011).

Segundo Bortoluzzi (2009) ao passar pelo corpo humano, a corrente elétrica pode causar uma série de perturbações, que variam desde pequenas contrações musculares até inibição do centro nervoso com parada respiratória, podendo produzir fibrilação ventricular e até parada cardíaca, queimaduras profundas, bem como alteração de características do sangue.

As partes do corpo humano que normalmente são afetadas pela passagem de correntes elétricas pelo corpo humano são, logicamente, devido à natureza do trabalho: mãos, pés, pernas, tronco e peito. Há um risco maior se a corrente passar entre os braços, pois pode atingir diretamente o coração (BORTOLUZZI, 2009).

De acordo com Thomé e Beline (2018) podemos traçar um grau de lesão em função da corrente que percorre o corpo, começando em 0,1mA causando apenas um leve formigamento, 0,5mA começamos a apresentar pequenas contrações musculares, 10mA não apresenta efeito perigoso desde que o tempo de exposição seja inferior a 5 segundos, 30mA apresenta risco de fibrilação cardíaca e a partir de 500mA podemos sofrer parada cardíaca (THOMÉ e BELINE, 2018).

Tendo em vista o problema supracitado, o presente trabalho teve como objetivo o desenvolvimento de um dispositivo de geração de sinais que emula um transformador cuja saída passou pela etapa de condicionamento da UR. Dessa forma o sinal de saída do dispositivo pode ser inserido diretamente na etapa de pós condicionamento, de forma que o firmware das URs possa ser depurado com um número de variáveis reduzido e com o risco de choque elétrico minimizado. Para que

as URs e suas variantes sejam consideradas operacionais, elas devem ser capazes de realizar a medição de energia por meio da aquisição de um sinal de tensão trifásica e de corrente trifásica, ambos provenientes da rede elétrica de baixa tensão. Nas versões do SGRede destinadas a instalação em indústrias, outra característica que define seu pleno funcionamento é a capacidade de ler harmônicas da rede elétrica. Assim, o dispositivo além de gerar os sinais de tensão e corrente, deve ser capaz de gerar distorções harmônicas nos mesmos, com a finalidade de validar o caso específico de URs destinadas a indústria.

Para ordenação dos assuntos a serem abordados de forma clara e objetiva, este trabalho está dividido em 4 capítulos, além das referências.

Capítulo 1 – Referencial Teórico: apresenta conceito fundamentais, conversão Digital/Analógico, memórias, linguagens de programação utilizadas, análise de Fourier, sistemas microcontrolados, conceitos no contexto industrial e apresenta também os circuitos integrados que são o alvo do trabalho.

Capítulo 2 – Metodologia: neste capítulo é descrito os dispositivos e softwares utilizados no projeto e o funcionamento do projeto.

Capítulo 3 – Implementação do projeto: descreve os procedimentos realizados durante a implementação do projeto, mostrando cada passo seguido.

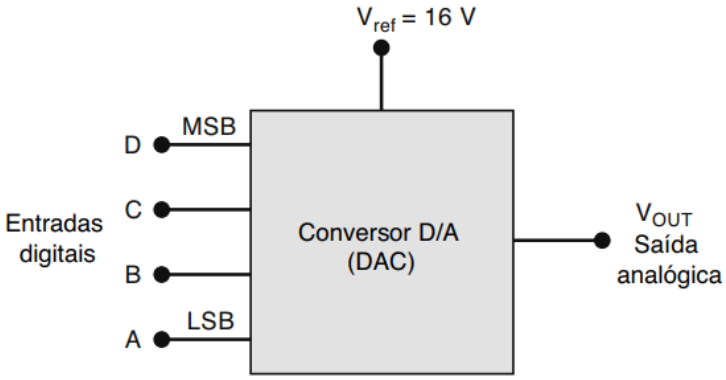
Capítulo 4 – Resultados: descreve os resultados obtidos decorrente da análise, gerando informações necessárias para a conclusão que é apresentada no fim do trabalho.

1. REFERENCIAL TEÓRICO

1.1 Conversão Digital/Analógico

A conversão D/A (digital para analógico) é basicamente o processo de conversão de um valor representado por um código digital (como binário ou BCD) em uma tensão ou corrente proporcional ao valor digital. A Figura 1 mostra o símbolo para um DAC típico de bits.

Figura 1: Representação Simbólica do conversor D/A



Fonte: (TOCCI, WIDMER e MOSS, 2017)

Note que existe uma entrada para uma tensão de referência, V_{ref}. Essa é utilizada para definir a tensão de fundo de escala, isto é, a tensão máxima que o conversor pode atingir. As entradas D, C, B e A são, em geral, acionadas pela saída do registrador de um sistema digital. Os $2^4 = 16$ diferentes números binários representados por esses quatro bits podem ser observados na Tabela 1.

Tabela 1: Tabela de Saída do Conversor D/A

A	B	C	D	V_{out}(V)
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4

0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

Fonte: (TOCCI, WIDMER e MOSS, 2017)

Para cada número de entrada, a tensão de saída do conversor D/A é única. Na realidade, para esse caso, a tensão analógica V_{out} é igual em *volts* ao número binário. Também poderia ser duas vezes o número binário, ou qualquer outro fator de proporcionalidade. A mesma ideia seria se a saída do conversor D/A fosse uma corrente I_{out}. Em geral, o valor da tensão analógico pode ser obtido de acordo com a equação 01:

$$saída\ analógica = K * entrada\ digital \quad (1)$$

em que K é o fator de proporcionalidade e é constante para determinado DAC (*Digital/Analogic Converter*) conectado em uma tensão de referência fixa. A saída analógica pode ser tensão ou corrente. Quando for tensão, K terá unidade de tensão, e quando for corrente K terá unidade de corrente. Para o DAC mostrado na figura 01, $K = 1V$, de forma que:

$$V_{out} = (1\text{V}) * Entrada\ Digital \quad (2)$$

É possível usar essa equação para calcular V_{out} para qualquer valor de entrada digital. Por exemplo, com uma entrada digital $1100_2 = 12_{10}$, temos (TOCCI, WIDMER e MOSS, 2017).

$$V_{out} = 1\text{V} * 12 = 12\text{V} \quad (3)$$

1.1.1 Pesos de Entrada

Observe que no DAC mostrado na Figura 1, cada entrada contribui com uma quantidade diferente para a saída analógica. Isso pode ser visto facilmente considerando que apenas uma entrada na tabela é HIGH. A contribuição de cada entrada digital é ponderada de acordo com sua posição em binário, sendo D o LSB (*Less Significative Bit*) e A o MSB (*Most Significative Bit*) (TOCCI, WIDMER e MOSS, 2017).

1.1.2 Resolução

A resolução de um conversor D/A é definida como a menor variação na saída analógica como resultado de uma mudança na entrada digital. Tendo como referência a Tabela 1 a resolução é de 1V, uma vez que V_{out} não pode ter uma saída menor que 1V (TOCCI, WIDMER e MOSS, 2017).

1.2 Memórias

As memórias são dispositivos que armazenam informações. Elas são muito úteis na computação pois são usadas principalmente em computadores e periféricos. Elas também são usadas em outros sistemas baseados em microprocessadores, como kits e projetos específicos. Armazenam informações para processamento, programação e criação de programas internos para funcionalidade do sistema (IDOETA e CAPUANO, 2011).

1.2.1 Classificação das Memórias

Segundo Capuano (2011) as memórias são classificadas em torno de 4 parâmetros, são eles:

- Acesso – As memórias acessam informações em lugares denominados localidades de memória. Cada uma das localidades recebem o nome de endereço, e o acesso a dado endereço pode ser dado de forma

sequencial ou aleatório, no acesso sequencial é necessário fazer a varredura de todas as posições anteriores a desejada, e na aleatória é possível acessar diretamente a informação de determinado endereço de forma direta;

- Volatilidade – As memórias podem ser voláteis (perdem a informação quando desenergizadas) e não voláteis (mantém a informação quando desenergizadas);
- Troca de dados – As memórias podem ser de escrita/leitura ou apenas escrita;
- Tipo de armazenamento – Podem ser estáticas (uma vez gravadas as informações lá permanecem) e dinâmicas (as informações precisam ser reescritas com certa recorrência, pois devido as características do dispositivo após um tempo as informações serão perdidas).

1.2.2 Memória EEPROM

As memórias de nome EEPROM ou E²PROM (Electrically Erasable Programmable Read-Only Memory) são dispositivos que permitem o apagamento dos dados armazenados eletricamente, e ainda, isoladamente por palavra de dados, sem necessidade de reprogramação total. Dessa forma as alterações de programação podem ser efetuadas pelo próprio sistema no qual a memória está inserida (IDOETA e CAPUANO, 2011).

1.3 Linguagem C

Na década de 1970, a linguagem de programação C foi desenvolvida por Dennis Ritchie na empresa Bell Labs. Ela surgiu como uma linguagem de programação de sistema para a implementação do sistema operacional UNIX, também desenvolvido em Bell Labs (RITCHIE).

Devido à sua eficiência, portabilidade e facilidade de uso, a linguagem C rapidamente se tornou popular, sendo fundamental para o desenvolvimento de muitos sistemas e tecnologias, como a internet, jogos eletrônicos, sistemas embarcados,

entre outros. Além disso, a linguagem C influenciou outras linguagens de programação como C++, Java e Python.

Atualmente, a linguagem C ainda é amplamente utilizada em todo o mundo devido à sua versatilidade e capacidade de oferecer controle próximo ao hardware (KERNIGHAN e RITCHIE, 1988).

1.4 Sistemas Microcontrolados

Sistemas microcontrolados são sistemas eletrônicos compostos por um microcontrolador e demais periféricos necessários para sua implementação. Um microcontrolador é um tipo de circuito integrado que incorpora um processador, memória e dispositivos de entrada e saída em um único chip. Esses componentes são projetados para trabalhar juntos, permitindo que o microcontrolador execute tarefas específicas com precisão e eficiência.

Os microcontroladores são usados em uma variedade de aplicações, desde eletrônicos de consumo até sistemas de controle industrial. Eles podem ser programados para realizar uma ampla variedade de funções, como ler sensores, controlar motores e atuadores, gerenciar sistemas de comunicação, executar algoritmos complexos e muito mais.

Os microcontroladores são projetados para serem eficientes em termos de energia e tamanho, o que os torna ideais para sistemas embarcados, que são sistemas eletrônicos incorporados em outros dispositivos, como automóveis, eletrodomésticos, dispositivos médicos e muito mais.

Alguns exemplos de microcontroladores comuns incluem o PIC da Microchip Technology, o AVR da Atmel e o STM32 da STMicroelectronics. O uso de microcontroladores é bastante comum em eletrônica, robótica e automação, entre outros campos.

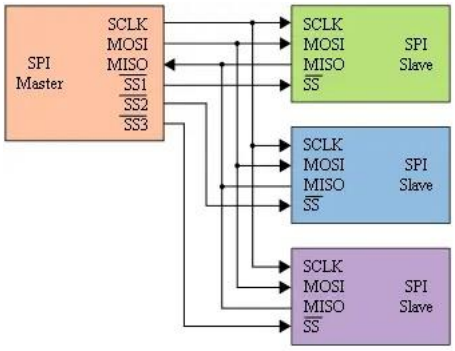
Esses sistemas são comumente utilizados em aplicações em que é necessário controle preciso e automatizado, como em equipamentos eletrônicos, automação industrial, robótica, sistemas de controle de acesso, entre outros.

1.4.1 Protocolo SPI

O protocolo SPI (Serial Peripheral Interface) é um padrão de comunicação serial síncrona utilizado para interconectar dispositivos eletrônicos. Ele é especialmente útil em aplicações que exigem uma comunicação rápida e confiável entre dispositivos, como em sistemas embarcados e de automação industrial (EMBARCADOS).

O SPI é baseado em uma arquitetura mestre-escravo, em que um dispositivo mestre controla um ou mais dispositivos escravos por meio de uma interface serial. O protocolo utiliza quatro linhas de comunicação: MOSI (Master Output Slave Input), MISO (Master Input Slave Output), SCLK (Serial Clock) e SS (Slave Select) (EMBARCADOS).

Figura 2: Diagrama em Blocos - Protocolo SPI



Fonte: (EMBARCADOS)

Durante a transmissão de dados, o mestre envia um sinal de clock (SCLK) para sincronizar a transferência de bits entre os dispositivos. O mestre também define qual dispositivo escravo está sendo acessado por meio do sinal SS (Slave Select). Em seguida, o mestre envia os dados por meio da linha MOSI e recebe a resposta do dispositivo escravo pela linha MISO. A velocidade de transmissão é definida pela frequência do sinal de clock e pode variar de acordo com a aplicação (EMBARCADOS).

1.1.1.1 Relógio em Tempo Real

Real-time clock (RTC), ou relógio em tempo real, é um dispositivo eletrônico utilizado para manter o controle preciso do tempo em aplicações que exigem alta precisão, como em sistemas de automação industrial, controle de processos, sistemas de segurança, entre outros. Ele é um componente comumente utilizado em sistemas embarcados, que precisam manter a hora e a data atualizadas mesmo após uma interrupção de energia elétrica. Existem diversas opções de RTC disponíveis no mercado, desde dispositivos independentes até circuitos integrados que incluem funcionalidades adicionais, como memória não-volátil (SIEWERT, 2016).

1.4.2 Interrupções

As interrupções em um microcontrolador são mecanismos essenciais para permitir que o processador execute tarefas específicas em resposta a eventos externos ou internos, sem afetar a execução do programa principal. Quando uma interrupção é acionada, o processador interrompe temporariamente a execução do programa atual, salva seu estado atual e inicia a execução da rotina de interrupção correspondente. A rotina de interrupção é geralmente de curta duração para minimizar o impacto na execução do programa principal (MIZIDI, NAIMI e NAIMI, 2011).

Depois que a rotina de interrupção é concluída, o processador retorna ao estado anterior e continua a execução do programa principal a partir do ponto em que foi interrompido. As interrupções são amplamente utilizadas em aplicações que exigem uma resposta rápida a eventos externos, como no controle de sensores ou atuadores em tempo real (ALVANO, 2012).

1.4.3 Timers (Temporizadores)

Os timers são componentes eletrônicos presentes em microcontroladores que possibilitam a medição de intervalos de tempo e a geração de sinais de temporização precisos. Eles são amplamente utilizados em aplicações que exigem controle de tempo, como em sistemas de automação industrial, eletrônica de consumo e robótica. Dependendo das necessidades da aplicação, os timers podem contar pulsos de

entrada externos ou gerar sinais de saída em intervalos predefinidos (PEREIRA, 2009).

Alguns microcontroladores possuem múltiplos timers que permitem a execução simultânea de várias tarefas de temporização. Além disso, os timers possuem diversos modos de operação, tais como modo de contagem ascendente, modo de contagem descendente e modo PWM (Pulse Width Modulation), dentre outros. A programação dos timers em um microcontrolador é realizada por meio de registros específicos que devem ser configurados de acordo com as necessidades da aplicação (MICROCHIP).

1.5 Análise de Fourier

1.5.1 Transformada de Fourier de Tempo Discreto

A análise de Fourier consiste em analisar sinais no domínio da frequência. A análise de sinais nesta faixa requer um mecanismo de transformação e utiliza a Transformada Discreta de Fourier (DFT). Basicamente, a transformada de Fourier de tempo discreto é usada para sinais periódicos e aperiódicos. No entanto, para sinais periódicos existe uma generalização chamada série de Fourier.

Para uma série de Fourier, apenas parte do sinal é necessária para representar o espectro de sinais periódicos, ou seja, parte do período igual a N. Com os valores amostrais desse intervalo, é possível montar os coeficientes espectrais do sinal. Série de Fourier, pois para este tipo de sinal temos um espectro que também é discreto (OPPENHEIM e WILLSKY, 2010).

$$a_k = \frac{1}{N}\sum_{n=0}^{N-1} x[n]e^{-jkw_0 n} \qquad (4)$$

Sendo:

a, o valor no domínio da frequência;

k, o coeficiente de frequência;

N, o período do sinal;

n, os valores de tempo discreto;

x, o sinal analisado;

w_0, a frequência fundamental;

A equação 4 pode representar todos os valores de a_k, onde k é o que informa a frequência fundamental do sinal. Algo interessante sobre esta representação é que como temos um sinal periódico $x[n]$ com período N, então a_k também é periódico com o mesmo período N.

Como temos uma representação no domínio da frequência, é importante que haja alguma forma de recuperar o sinal no domínio do tempo, fazendo o inverso, onde o sinal no domínio do tempo pode ser expresso como uma combinação linear de exponenciais complexas, e os valores que os multiplicam são exatamente nossos coeficientes espectrais.

$$x[n] = \sum_{k=0}^{N-1} a_k e^{jkw_0 n} \tag{5}$$

Sendo:

x, o sinal recuperado;

n, os valores de tempo discreto;

N, o período do sinal;

a, o valor no domínio da frequência;

k, o coeficiente de frequência;

w_0, a frequência fundamental;

1.5.2 Transformada Rápida de Fourier

O algoritmo da transformada rápida de Fourier foi desenvolvida com o intuito de calcular a transformada discreta de Fourier de forma a utilizar menos recursos computacionais. A transformada rápida de Fourier (FFT, do inglês *Fast Fourier Transform*) reaproveita resultados computados anteriormente, reduzindo o número total de operações aritméticas, melhorando significativamente a sua eficiência (GOMES, 2019).

1.6 Linguagem de Programação Python e suas Aplicações em Sistemas Microcontrolados

Python é uma linguagem de programação de alto nível e de código aberto, criada em 1991 por Guido van Rossum. Ela é amplamente utilizada em diversas áreas,

incluindo ciência de dados, inteligência artificial, desenvolvimento web, entre outras. Uma das principais vantagens da linguagem Python é a sua facilidade de uso e sintaxe simples, o que a torna uma das linguagens mais populares no mundo da programação. Além disso, a grande quantidade de bibliotecas disponíveis para Python permite que os desenvolvedores possam criar soluções para diversas aplicações com facilidade (PYTHON).

Uma das possibilidades de uso do Python em associação com microcontroladores é a criação de interfaces gráficas para esses dispositivos. Através da biblioteca PyQT5, é possível desenvolver interfaces gráficas para microcontroladores de forma simples e eficiente. Essa biblioteca permite a criação de janelas, botões, gráficos, entre outros elementos gráficos que podem ser controlados pelo microcontrolador através de um protocolo de comunicação, como o SPI ou I2C. Dessa forma, é possível criar interfaces amigáveis para o usuário final e facilitar a interação com o dispositivo (THE QT COMPANY).

1.7 I.H.M (Interface Homem Máquina)

Uma interface homem-máquina composta por painéis que permitem ao usuário interagir com uma máquina, programa de computador ou sistema. Tecnicamente, o acrônimo IHM pode ser usado para qualquer display usado para interagir com um dispositivo, mas é mais frequentemente usado para descrever displays usados em ambientes industriais. As interfaces do usuário exibem dados em tempo real e permitem que os usuários controlem os dispositivos por meio de uma interface gráfica do usuário (COPADATA, 2022).

1.8 Medição de Distorção Harmônica no Contexto da Indústria

A medição de harmônicas da rede elétrica é uma prática importante em ambientes fabris, pois permite avaliar a qualidade da energia elétrica fornecida para as instalações industriais. A presença de harmônicas pode resultar em problemas como queda de produtividade, falhas em equipamentos e aumento dos custos operacionais. Além disso, a norma NBR 5410 estabelece limites para a distorção

harmônica na rede elétrica, a fim de garantir a segurança das instalações e dos usuários.

A medição de harmônicas pode ser realizada por meio de equipamentos específicos, como analisadores de qualidade de energia. Esses equipamentos permitem identificar a presença de harmônicas, bem como a magnitude e a frequência dessas distorções.

Um dispositivo capaz de realizar as medições de distorção harmônica em ambiente industrial é o SGRede-QE, desenvolvido pela empresa VTInova, a descrição do dispositivo diz:

> O SGRede-QE é um analisador e medidor de energia compacto e de fácil instalação, destinado as concessionárias distribuidoras de energia, que precisam realizar campanhas de medição da ANEEL para atender os requisitos do Prodist 8. O SGRede-QE realiza medições de tensão e corrente em sistemas monofásicos, bifásicos e trifásicos (VÓRTICE TECNOLOGIA E INOVAÇÃO).

Um estudo realizado por Mohd Azhar bin Abdul Razak et al. (2019) avaliou a presença de harmônicas em uma fábrica de cimento na Malásia. Os resultados mostraram que as harmônicas estavam presentes na rede elétrica da fábrica, o que pode ter afetado o desempenho dos equipamentos e a eficiência energética. Os autores concluíram que a medição de harmônicas é uma prática importante para garantir a qualidade da energia elétrica fornecida para as instalações industriais.

Outro estudo realizado por Patrycja Tomczyk et al. (2020) avaliou a presença de harmônicas em uma instalação industrial na Polônia. Os resultados mostraram que as harmônicas estavam presentes na rede elétrica da instalação, o que pode ter afetado a eficiência energética e a vida útil dos equipamentos. Os autores destacaram a importância da medição de harmônicas para identificar e corrigir problemas na rede elétrica.

Portanto, a medição de harmônicas é uma prática importante em ambientes fabris, pois permite avaliar a qualidade da energia elétrica fornecida e identificar possíveis problemas que podem afetar a produtividade e a eficiência energética.

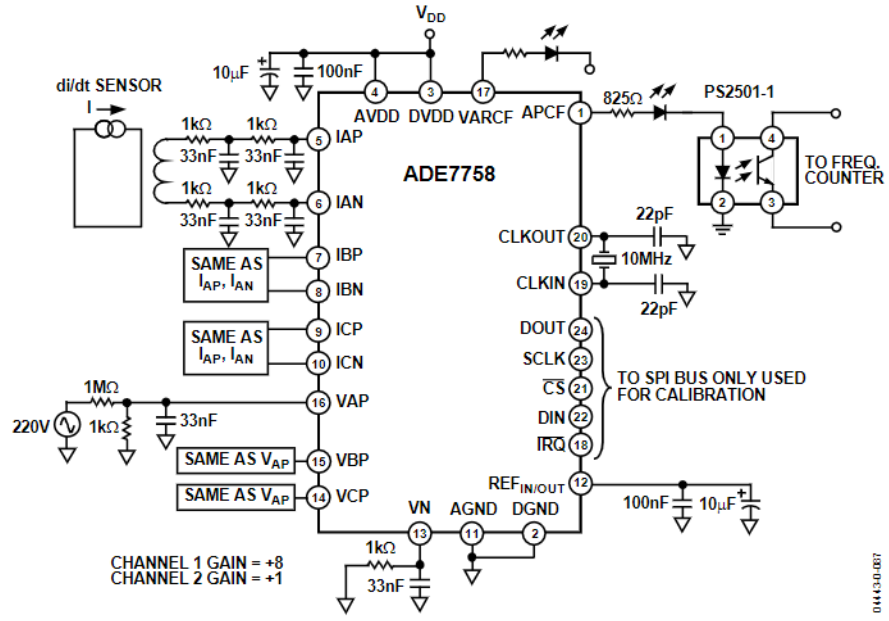
1.9 ADE 7758 e ADE9000

Segundo o datasheet do ADE7758:

> O ADE7758 é um medidor de energia elétrica trifásico de alta precisão com interface serial e duas saídas de pulso. O ADE7758 incorpora ADCs Sigma Delta de segunda ordem, um integrador, circuitos de referência, um sensor de temperatura e todos os processamentos de sinal necessário para realizar a medição de energia ativas, reativas e aparente e cálculos rms (ANALOG-DEVICES, 2011).

Na figura 3 temos o diagrama esquemático de um circuito sugerido pelo fabricante do ADE7758 para realização de testes e validação do mesmo, o circuito apresentado se assemelha em grande parte ao circuito utilizado pela empresa no projeto de hardware a UR.

Figura 3: Circuito mínimo sugerido pelo fabricante para o ADE7758



Fonte: (ANALOG-DEVICES, 2011)

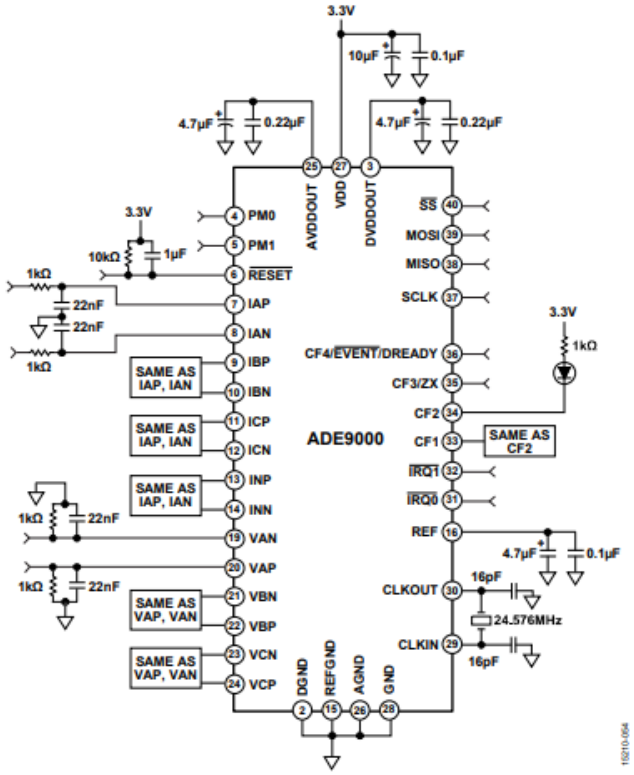
Segundo o datasheet do ADE9000:

> O ADE9000 é um CI medidor trifásico de energia elétrica de alta precisão que possui uma interface de comunicação serial e duas saídas de pulsos. O ADE9000 possui ADCs do tipo Sigma Delta, integradores digitais, circuito de referência, sensor de temperatura e todo o processamento de sinal requerido para executar a medição de energias ativa, reativa e aparente efetivos, além de oferecer um buffer de forma de onda flexível e integrado

que amostra a uma taxa de 128 amostras por ciclo, armazenando até 4 ciclos por canal (ANALOG-DEVICES, 2017).

Na figura 4, assim como o caso do ADE7758, temos o diagrama esquemático de um circuito sugerido pelo fabricante do ADE9000 para realização de testes e validação do mesmo, o circuito apresentado se assemelha em grande parte ao circuito utilizado pela empresa no projeto de hardware a UR.

Figura 4: Circuito mínimo sugerido pelo fabricante para o ADE9000



Fonte: (ANALOG-DEVICES, 2017)

2 METODOLOGIA

O Trabalho apresentado é uma Pesquisa Aplicada, e teve como objetivo a realização de Pesquisa exploratória sobre o material bibliográfico e de laboratório. Foram utilizados procedimentos técnicos de pesquisa e bibliografia experimental. O método de abordagem hipotético-dedutivo e o método de procedimento monográfico foram utilizados em sua elaboração. A coleta de dados foi feita através da observação direta intensiva e documentação indireta, sendo estes dados qualitativos e interpretados de forma global.

A seguir são abordados os aspectos metodológicos da pesquisa realizada, escrevendo-se os procedimentos necessários para o desenvolvimento de um dispositivo de geração de sinais que emula um transformador cuja saída passou pela etapa de condicionamento da UR.

2.1 Materiais

Nesta seção apresenta-se o ambiente em que ocorreu o desenvolvimento deste trabalho, bem como as ferramentas de desenvolvimento, os materiais e os cenários de testes utilizados na validação do dispositivo gerador de sinais.

2.1.1 Ambiente de Desenvolvimento

O hardware utilizado nesta pesquisa tem a seguinte configuração:

- 16GB Memória RAM DD4;
- Processador Intel Core i7 12ª Geração;
- Placa de vídeo RTX3050 – 8 GB;

2.1.2 Especificação de Dispositivos e Softwares Utilizados no Projeto

Este tópico detalha os dispositivos utilizados para a geração de sinais, medição das formas de onda equivalentes e o microcontrolador utilizado. Também serão

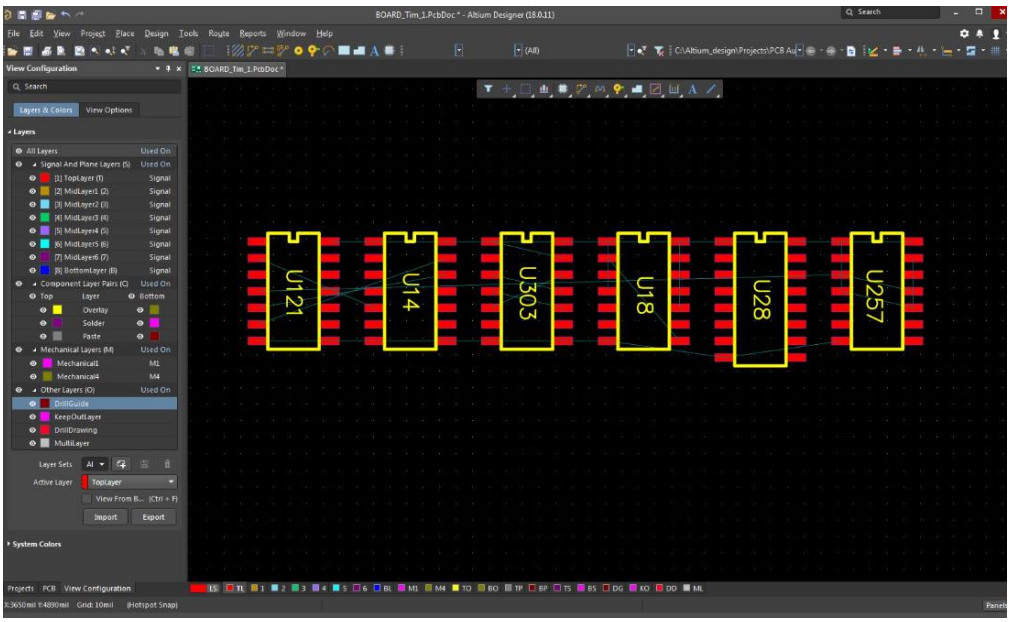
citados os softwares utilizados para realização dos cálculos apresentação dos gráficos de forma visual.

2.1.2.1 Altium Designer

O Altium Designer é um software de design de PCI (Placa de Circuito Impresso) ou PCB (*Printed Circuit Board*), desenvolvido pela empresa australiana Altium Limited, que fornece recursos avançados para projetar, simular e gerenciar circuitos eletrônicos. Ele é utilizado em diversos setores, incluindo eletrônica de consumo, automotivo, aeroespacial e militar.

O software oferece várias funcionalidades, como esquemáticos, bibliotecas de componentes, roteamento de trilhas, gerenciamento de projetos e simulação de circuitos, além de suportar vários sistemas operacionais. É uma escolha popular entre engenheiros e designers de PCB em todo o mundo devido à sua eficiência e funcionalidades avançadas (ALTIUM LIMITED). Na figura 5 é ilustrado um exemplo de projeto em desenvolvimento utilizando a ferramenta Altium Designer.

Figura 5: Interface para desenho de layout – Altium Designer



Fonte: (ALTIUM LIMITED)

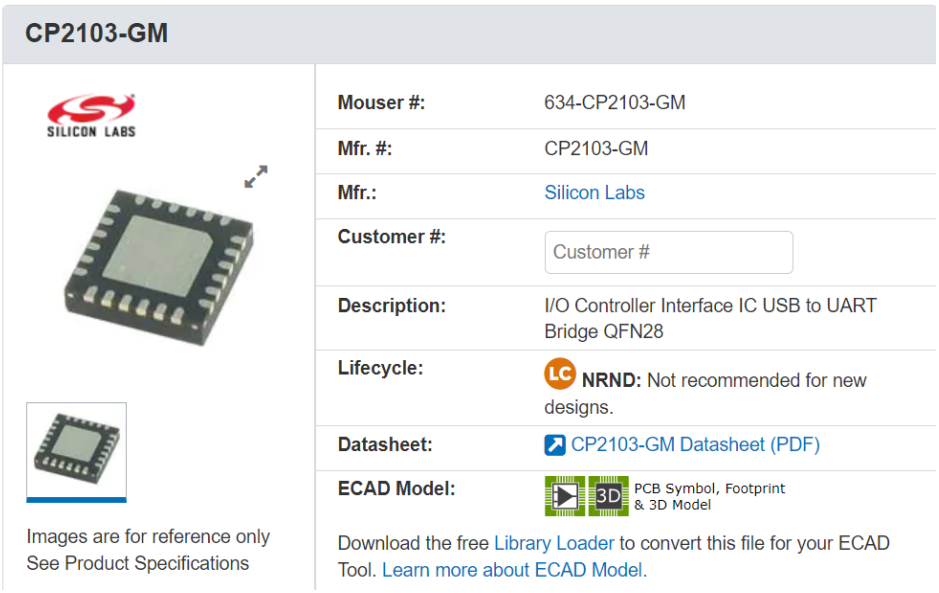
2.1.2.2 Placa de Circuito Impresso

Para o projeto foi desenvolvida uma PCI contendo todos os componentes eletrônicos necessários para a implementação das funcionalidades descritas no documento de requisitos estabelecidos no início do projeto.

2.1.2.2.1 **Conversor USB-Serial CP2103**

O CP2103 é um controlador de interface USB-UART produzido pela Silicon Labs, que permite a conexão entre dispositivos com interfaces UART e computadores com interfaces USB. Além disso, o CP2103 oferece recursos como proteção ESD, controle de energia e alta velocidade de transferência de dados, tornando-o uma escolha popular para muitos projetos de eletrônica. A variedade de plataformas de desenvolvimento e sistemas operacionais suportados pelo CP2103 permite sua utilização em diferentes aplicações (SILICONLABS).

Figura 6: Circuito Integrado CP2103 – Descrição Resumida



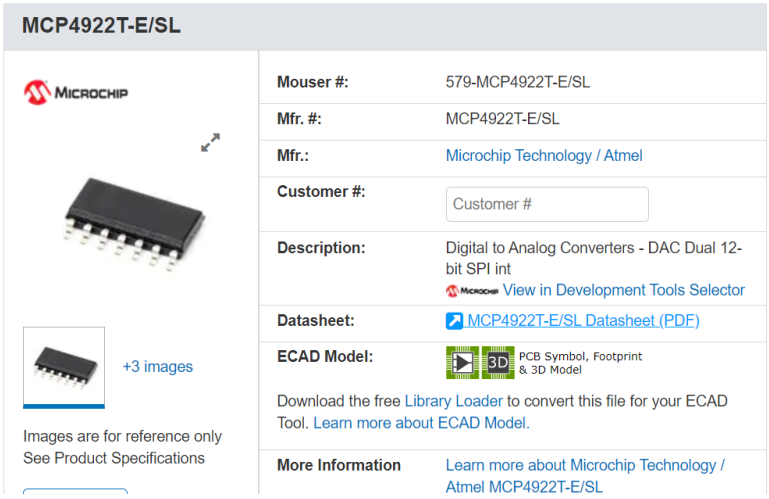
Fonte: (MOUSER)

2.1.2.2.2 **Conversor Digital-Analógico MCP4922T**

O MCP4922T é um circuito integrado (CI) de conversor digital-analógico (DAC) de alta precisão com resolução de 12 bits e taxa de atualização de até 1 MHz. Ele possui um buffer de saída, controle de ganho, shutdown de alimentação e seleção de referência de tensão, tornando-o uma escolha popular em projetos que exigem a geração de sinais analógicos precisos.

O CI MCP4922T é produzido pela Microchip Technology e pode ser controlado por microcontroladores e outros dispositivos digitais por meio da interface de comunicação SPI. Além disso, é adequado para aplicações em áreas como equipamentos de teste e medição, controle de motor e fontes de alimentação programáveis (MICROCHIP).

Figura 7: MCP4922T - Descrição Resumida



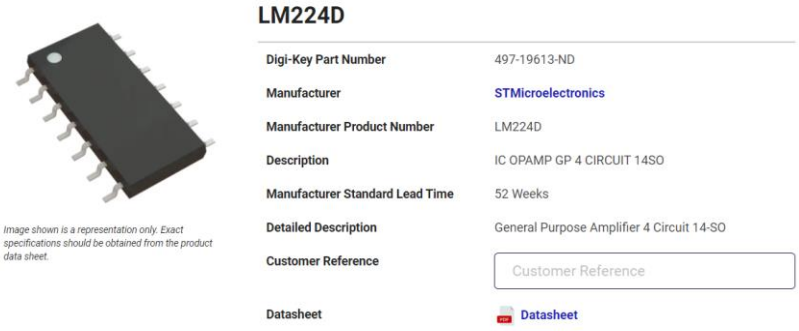
Fonte: (MOUSER)

2.1.2.2.3 **Amplificador Operacional LM224**

O CI LM224 é um amplificador operacional (op-amp) de quatro canais produzido pela Texas Instruments. Ele é utilizado para amplificar sinais em circuitos eletrônicos e tem uma ampla faixa de tensão de alimentação. O LM224 possui uma alta impedância de entrada, baixa corrente de polarização e uma largura de banda alta, o que o torna adequado para várias aplicações em eletrônica, como amplificação de sinais de baixa potência, filtros ativos, osciladores, entre outros. O CI é compatível com uma ampla variedade de circuitos eletrônicos e é amplamente utilizado em projetos em todo o mundo. Além disso, a Texas Instruments disponibiliza exemplos

de aplicação e notas de aplicação, tornando-o uma escolha popular para engenheiros e designers que trabalham com amplificação de sinal e outros projetos em eletrônica (TEXAS INSTRUMENTS).

Figura 8: LM224 - Descrição Resumida



Fonte: (DIGIKEY)

2.1.2.2.3.1 Configuração Seguidor de Tensão ou *Buffer* de Tensão

Segundo (JUNIOR, 2017) além de apresentar ganho unitário, nessa configuração o amplificador operacional opera da forma mais próxima possível ao ideal, apresentando alta impedância de entrada e baixa impedância de saída. Na figura 9 é ilustrado o amplificador operacional na configuração denominada *buffer* de tensão.

A equação de saída que define o ganho do circuito desse circuito é:

$$A_{vf} = \frac{V_o}{V_i} = 1 \qquad (6)$$

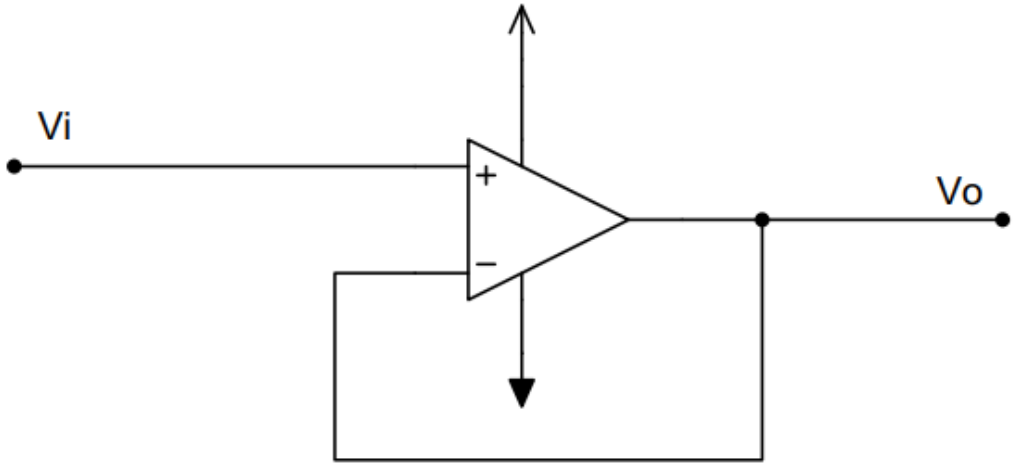
Onde:

A_{vf} = Ganho do Estágio;

V_o = Tensão de Saída;

V_i = Tensão de Entrada;

Figura 9: Amplificador Operacional - Configuração Buffer



Fonte: (Elaborado pelo autor)

Essa topologia de circuito é utilizada normalmente para:

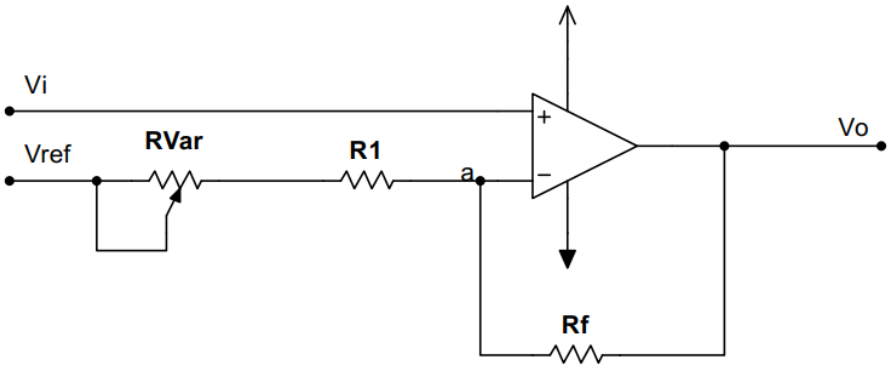
- Isolação entre estágios do circuito;
- Reforço de corrente;
- Casamento de Impedâncias, etc.

No caso em específico do projeto, essa configuração foi utilizada para gerar a tensão de referência para o conversor D/A.

2.1.2.2.3.2 Configuração Amplificador Subtrator

Esse circuito é utilizado para obter na sua saída a diferença entre dois sinais de entrada, e os cálculos do funcionamento do circuito variam de acordo como ele é montado, não caso específico do projeto, ele foi utilizado para remover a componente de tensão direta que é agregada ao sinal quando o mesmo é gerado pelo conversor D/A (JUNIOR, 2017). Na figura 10 é ilustrado o amplificador operacional na configuração denominada amplificador subtrator.

Figura 10: Configuração Subtrator – Utilizada no Projeto



Fonte: (Elaborado pelo autor)

2.1.2.2.4 **Regulador de Tensão NCP1117**

O NCP1117 é um regulador de tensão linear de baixa queda de tensão (LDO) produzido pela ON Semiconductor. Ele é projetado para oferecer uma saída de tensão estável e regulada em uma variedade de aplicações eletrônicas, incluindo fontes de alimentação, sistemas de controle de motor e circuitos de iluminação LED. O NCP1117 é capaz de fornecer uma corrente de até 1A e é compatível com uma ampla variedade de tensões de entrada, de 1,8 V a 20 V. Além disso, ele possui recursos de proteção, como proteção contra sobrecorrente e proteção contra sobreaquecimento, para garantir a segurança e a confiabilidade das aplicações.

O NCP1117 é amplamente utilizado em projetos de eletrônica em todo o mundo, devido à sua alta eficiência e desempenho confiável. Ele também é apoiado por recursos como exemplos de aplicação e notas de aplicação, tornando-o uma escolha popular para engenheiros e designers que trabalham com fontes de alimentação e outros projetos em eletrônica (ON SEMI).

No caso do projeto, a versão do CI escolhida foi o NCP1117LPST33T3G (Figura 11), essa versão já apresenta uma tensão de saída fixa em 3,3V, valor esse necessário para alimentar todos os circuitos integrados do circuito.

Figura 11: Regulador de Tensão Linear NCP1117ST33T3G – Descrição Resumida

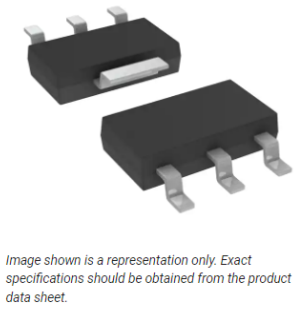


Image shown is a representation only. Exact specifications should be obtained from the product data sheet.

NCP1117LPST33T3G

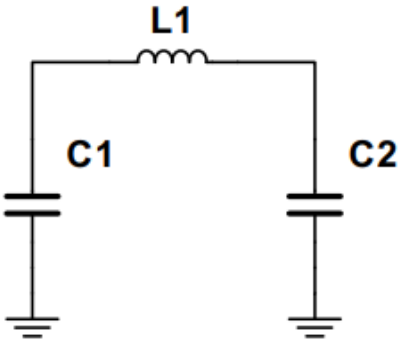
Digi-Key Part Number	NCP1117LPST33T3GOSTR-ND - Tape & Reel (TR) NCP1117LPST33T3GOSCT-ND - Cut Tape (CT) NCP1117LPST33T3GOSDKR-ND - Digi-Reel®
Manufacturer	onsemi
Manufacturer Product Number	NCP1117LPST33T3G
Description	IC REG LINEAR 3.3V 1A SOT223
Manufacturer Standard Lead Time	45 Weeks
Detailed Description	Linear Voltage Regulator IC Positive Fixed 1 Output 1A SOT-223
Customer Reference	Customer Reference
Datasheet	Datasheet

Fonte: (DIGIKEY)

2.1.2.2.5 **Filtro PI**

O filtro Pi é um tipo de filtro passa-baixas que tem como função filtrar ou atenuar frequências acima de uma determinada frequência de corte. O filtro Pi é composto por um indutor e dois capacitores, conforme apresentado na Figura 12, configurados em uma estrutura em forma de Pi. A frequência de corte do filtro é determinada pelos valores do indutor e dos capacitores utilizados. O filtro Pi é bastante comum em diversas aplicações eletrônicas, como em fontes de alimentação, filtros de áudio e de comunicação, entre outras (ELECTRONICS TUTORIALS).

Figura 12: Filtro PI



Fonte: (Elaborado pelo autor)

2.1.2.2.6 **Diodo Schottky MBRS260T3**

O diodo Schottky MBRS260T3 (Figura 13) é um dispositivo semicondutor que atua como um retificador de alta velocidade e baixa queda de tensão. Produzido pela ON Semiconductor, ele possui uma tensão de ruptura direta máxima de 60V, corrente direta máxima de 2A e uma queda de tensão direta típica de 0,34V a uma corrente de 2A.

O diodo Schottky MBRS260T3 é amplamente utilizado em aplicações de eletrônica, como retificação de fontes de alimentação, proteção contra reversão de polaridade e retificação de sinais de alta frequência. Ele possui uma construção de junção metal-semicondutor que permite que ele opere com alta eficiência e velocidade em comparação com diodos retificadores convencionais.

Além disso, o diodo MBRS260T3 é produzido com materiais de alta qualidade e é submetido a rigorosos testes de qualidade para garantir a confiabilidade e o desempenho consistente em uma ampla variedade de aplicações (ON SEMI).

Sua aplicação no dispositivo é atuar como um seletor de fontes, onde cada fonte fica protegida contra tensões de retorno, podendo assim ser ligadas em paralelo garantindo várias fontes de alimentação possíveis para o dispositivo.

Figura 13: Diodo Schottky MBRS260T3 – Descrição Resumida

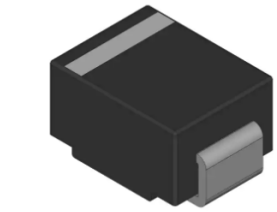


Image shown is a representation only. Exact specifications should be obtained from the product data sheet.

MBRS260T3

Digi-Key Part Number	MBRS260T3OS-ND - Tape & Reel (TR)
Manufacturer	onsemi
Manufacturer Product Number	MBRS260T3
Description	DIODE SCHOTTKY 60V 2A SMB
Detailed Description	Diode 60 V 2A Surface Mount SMB
Customer Reference	Customer Reference
Datasheet	Datasheet

Fonte: (DIGIKEY)

2.1.2.2.7 **Microcontrolador STM32F100**

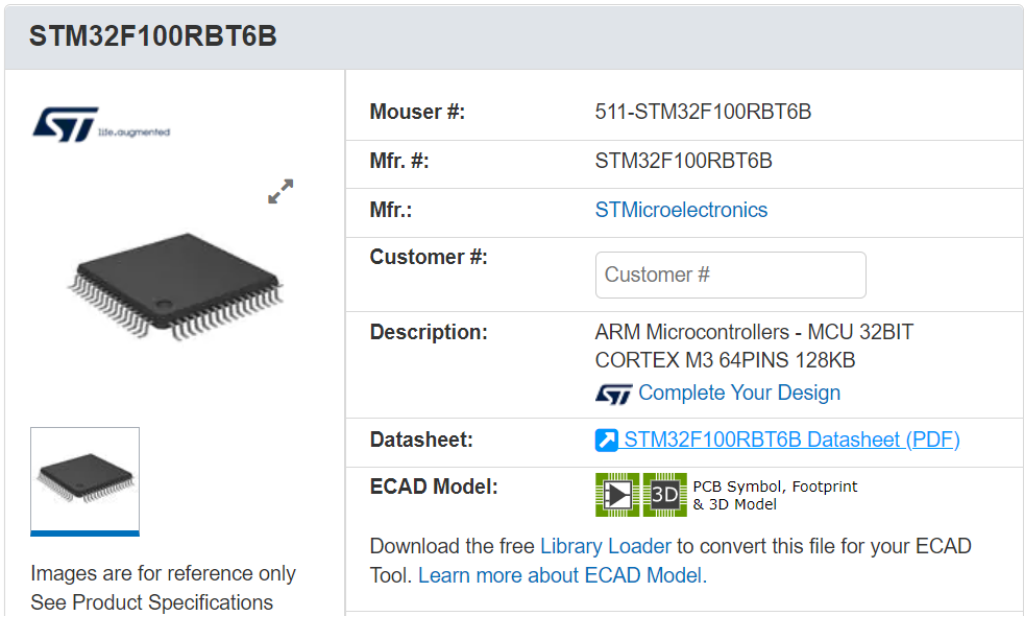
O STM32F100 é um microcontrolador de 32 bits que utiliza a arquitetura ARM Cortex-M3, fabricado pela STMicroelectronics. Sua versatilidade o torna adequado para diversas aplicações, como sistemas de controle industrial, equipamentos médicos, dispositivos de comunicação e automação, entre outros.

Ele é equipado com diversos recursos, como memória flash, memória RAM, periféricos de comunicação, temporização e conversores analógico-digital (ADC) e digital-analógico (DAC), além de ser programável por meio das linguagens C ou C++.

O microcontrolador também é suportado por várias ferramentas de desenvolvimento, incluindo o software de desenvolvimento integrado (IDE) STM32CubeIDE da própria STMicroelectronics (STMICROELECTRONICS).

Uma breve descrição do circuito integrado e seus periféricos pode ser observada na Figura 14.

Figura 14: Microcontrolador STM32F100xx e sua descrição breve



Fonte: (MOUSER ELECTRONICS)

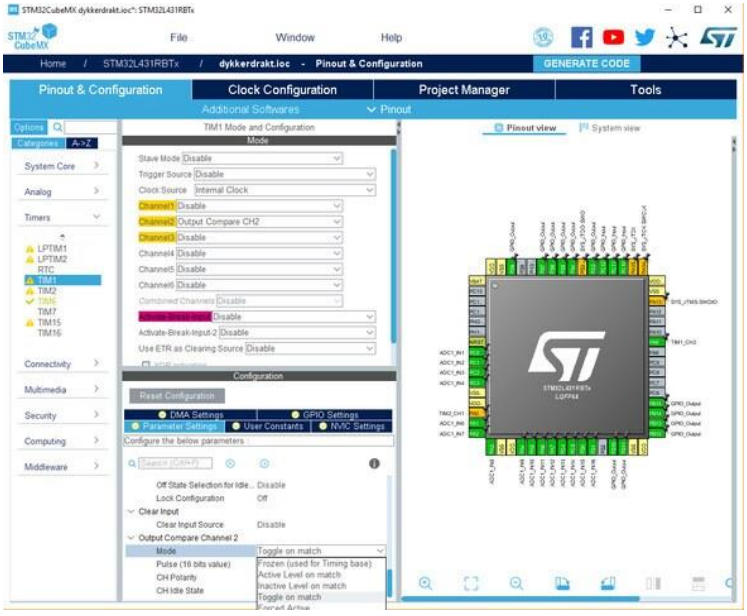
2.1.2.3 STMCubeMX

O STM32CubeMX é uma ferramenta de configuração de software desenvolvida pela STMicroelectronics para microcontroladores STM32. A ferramenta facilita o processo de desenvolvimento, permitindo que os desenvolvedores configurem e gerem código inicial para seus projetos.

Com uma interface gráfica do usuário (GUI), que pode ser visualizada na Figura 15, o STM32CubeMX permite selecionar e configurar periféricos, gerar código e configurar pinos de E/S. Além disso, a ferramenta suporta várias plataformas de desenvolvimento e sistemas operacionais, garantindo a portabilidade do código gerado entre diferentes plataformas. O STM32CubeMX é gratuito e de código aberto, disponível para download no site da STMicroelectronics.

Também possui uma ampla variedade de recursos de suporte, como exemplos de código, documentação e fóruns, o que o torna uma escolha popular para desenvolvedores que utilizam microcontroladores STM32 em seus projetos (STMICROELECTRONICS).

Figura 15: Interface Gráfica de Configuração STM32Cubemx



Fonte: (STMICROELECTRONICS)

No projeto, a parametrização do microcontrolador foi feita completamente através da ferramenta STM32CubeMX.

2.1.2.4 Keil µVision

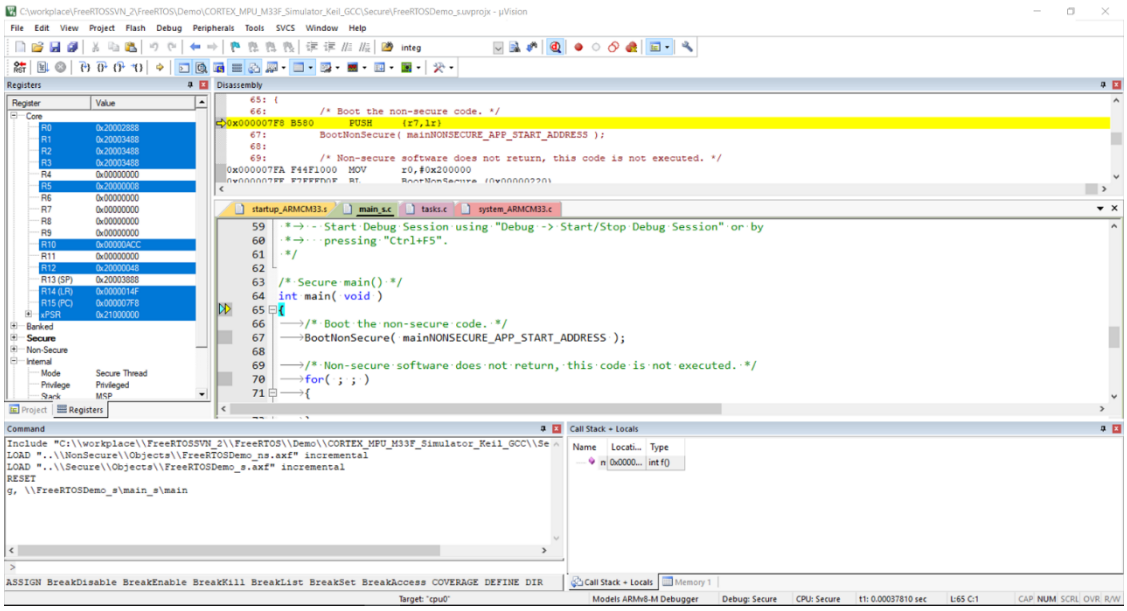
Keil µVision é uma plataforma de desenvolvimento integrado (IDE) para programação de microcontroladores baseados em ARM, oferecendo uma variedade de ferramentas, como editor de código, compilador, linker, debugger e simulador.

Ele é projetado para fornecer um ambiente completo de desenvolvimento para dispositivos ARM Cortex-M, Cortex-R e Cortex-A, permitindo aos desenvolvedores criar e depurar aplicativos para esses dispositivos. O Keil µVision é amplamente utilizado em todo o mundo por desenvolvedores de microcontroladores ARM devido à sua facilidade de uso, interface intuitiva e recursos avançados de depuração, incluindo simulação de código em tempo real e rastreamento de código em tempo real.

Além disso, ele é compatível com uma ampla variedade de dispositivos de depuração, permitindo aos desenvolvedores escolher a solução de depuração mais adequada para seus projetos (ARM).

A interface de programação do Keil pode ser observada na Figura 16.

Figura 16: Interface de Programação - Keil



Fonte: (FREERTOS)

No projeto, o algoritmo do dispositivo foi desenvolvido utilizando a versão gratuita da IDE Keil µVision.

2.1.2.5 Biblioteca math.h em C

A biblioteca math em C é uma coleção de funções matemáticas que permitem que os programadores realizem cálculos matemáticos complexos em seus programas em C. Essa biblioteca contém funções para operações comuns, como raiz quadrada, exponenciação, trigonometria, logaritmos e outras funções matemáticas mais avançadas. As funções matemáticas da biblioteca estão prontas para uso, permitindo que os desenvolvedores chamem essas funções facilmente em seus programas, sem precisar implementar o algoritmo do zero.

A biblioteca math em C é uma biblioteca padrão incluída na maioria das implementações do compilador C, o que significa que os programadores não precisam instalar nenhuma biblioteca adicional para utilizá-la. A biblioteca é amplamente utilizada em programas científicos, engenharia e outros projetos que envolvem cálculos matemáticos. A biblioteca math é uma ferramenta valiosa para programadores que precisam de precisão matemática em seus programas e que desejam economizar tempo ao utilizar funções matemáticas prontas para uso em vez de implementar algoritmos complexos por conta própria (CPPREFERENCE).

No projeto a biblioteca é utilizada para fazer o cálculo das formas de onda para funcionamento independente, ou seja, quando o dispositivo não estiver ligado à sua interface de programação de eventos, ele ainda assim é capaz de gerar um único pacote de formas onda fixas e pré-determinadas.

2.1.2.6 PyCharm

PyCharm é um software desenvolvido pela JetBrains que consiste em um ambiente de desenvolvimento integrado (IDE) para programação em Python. Este ambiente oferece uma ampla gama de recursos que auxiliam na eficiência e produtividade do desenvolvimento de software em Python, incluindo depuração integrada, sugestões de correção de erros, integração com ferramentas de controle de versão, gerenciamento de pacotes, suporte para testes automatizados, completude de código, entre outros.

O PyCharm está disponível em duas versões, sendo a primeira a Community Edition, que é gratuita e de código aberto, e a segunda, a Professional Edition, que é

paga e oferece recursos adicionais, como suporte a diferentes linguagens de programação, análise de código avançada e integração com bancos de dados (JETBRAINS).

No projeto a IDE PyCharm foi utilizada para o desenvolvimento dos códigos da interface gráfica de controle do dispositivo, nesse software foram integradas bibliotecas de elementos gráficos, cálculos matemáticos avançados e comunicação com dispositivos periféricos, como o caso da porta serial.

2.2 Métodos

As etapas metodológicas utilizadas no desenvolvido deste trabalho foram as seguintes: definição do protocolo entre o dispositivo e a interface de usuário, dimensionamento dos componentes eletrônicos necessários para implementação do protótipo, elaboração do esquema elétrico, elaboração do layout, confecção da placa de circuito impresso, montagem e testes do protótipo, desenvolvimento do firmware, desenvolvimento da interface gráfica, testes e validações do dispositivo gerador de sinais.

2.2.1 Elaboração do Diagrama Esquemático

Nessa etapa os componentes supracitados bem como os complementares necessários foram inseridos na página de esquemático e as ligações necessárias entre eles foram feitas.

2.2.2 Elaboração do layout

Em complemento a etapa 2.2.1, nessa etapa os componentes são exportados do esquemático para o ambiente de trabalho de layout, nesse ambiente os componentes são dispostos de forma a facilitar o processo de ligação por meio de trilhas condutivas, as ligações são executadas e por fim são gerados os arquivos de fabricação da placa.

2.2.3 Confecção da Placa de Circuito Impresso

Uma vez que os arquivos de fabricação foram gerados, os mesmos foram inseridos em uma pasta compactada e enviados a um fabricante estrangeiro para sua fabricação.

2.2.4 Montagem e Testes do Protótipo

Com os componentes e a placa em mãos a montagem dos componentes foi efetuada utilizando uma estação de ferro de solda. Com a placa montada, testes básicos como verificação de existência de curtos-circuitos e tensões de alimentação foram realizados.

Uma vez aprovado nesses testes, o dispositivo passou pelo processo de gravação de um firmware simples de testes, no qual o mesmo recebeu um código para receber uma mensagem na porta serial e responder via serial com a mensagem enviada, para validar que o microcontrolador estava operando.

Em posse da placa completamente operacional foi possível dar início a etapa de desenvolvimento do firmware.

2.2.5 Desenvolvimento do Firmware

Nessa etapa, utilizando a linguagem C, a ferramenta STMCubeMX e o ambiente de programação Keil µVision, o firmware foi implementado seguindo as regras estabelecidas de comunicação entre a interface gráfica de usuário e o dispositivo. Regras essas que são aprofundadas posteriormente na seção 3 deste trabalho.

2.2.6 Desenvolvimento da Interface Gráfica

Nessa etapa, utilizando Python, a ferramenta Pycharm e as ferramentas gráficas disponíveis no repositório do Python, o software foi implementado seguindo as regras estabelecidas de comunicação entre a interface gráfica de usuário e o

dispositivo. Regras essas que são aprofundadas posteriormente na seção 3 deste trabalho.

2.2.7 Testes e Validações do Dispositivo Gerador de Sinais

Para validação foram definidos dois cenários de testes, quais sejam: dispositivo gerador de sinais sem carga, e dispositivo gerador de sinais com carga.

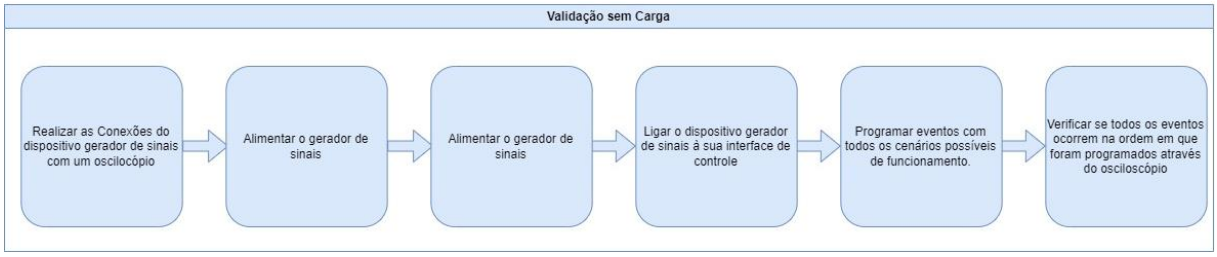
O fluxograma da Figura 17 mostra o funcionamento do projeto quando ligado para testes sem carga, ou seja, sem a presença do medidor de energia ao qual seria ligado, sendo assim ligado somente a um osciloscópio.

Nesse primeiro momento o dispositivo tem suas saídas de tensão e corrente ligados diretamente aos canais do osciloscópio, em seguida é alimentado, e nesse momento espera-se visualizar as formas de onda que o dispositivo foi programado para gerar caso não tenha nenhuma interação entre o usuário e o gerador de sinais.

Após isso, o dispositivo deve ser ligado por meio da porta USB ao computador que tenha sua interface de utilização, e eventos que abranjam subtensão, sobretensão, funcionamento normal, falta de fase e geração de N harmônicas, devem ser programados e carregados no dispositivo.

Com isso, basta validar se todos os eventos ocorrem na hora programada, com isso já é possível validar que o dispositivo está operando normalmente.

Figura 17: Fluxograma do funcionamento do projeto sem carga



Fonte: (Elaborado pelo autor, 2023)

Em posse desses resultados é possível utilizar o dispositivo ligado diretamente a carga, que nesse caso é o medidor de energia, na etapa pós condicionamento conforme ilustrado na figura 18. Para isso devemos ligar as saídas do dispositivo às entradas do medidor de energia na etapa de pós condicionamento, alimentar ambos os dispositivos, ligar o gerador de sinais à sua interface de programação, realizar o

processo de calibração do medidor para que ele trabalhe normalmente com o gerador de sinais, e por fim realizar coletas de medição de energia.

Figura 18: Fluxograma do projeto funcionando com carga

Fonte: (Desenvolvido pelo autor, 2023)

A partir desse ponto, fica a critério do usuário qual a melhor forma de gerar os eventos, pois como o dispositivo foi desenvolvido para facilitar e tornar seguro o processo de desenvolvimento de firmware, somente o desenvolvedor de firmware que estiver utilizando o dispositivo saberá qual a melhor sequência e quanto tempo cada evento deverá ter para que seu firmware seja validado.

3. IMPLEMENTAÇÃO DO PROJETO

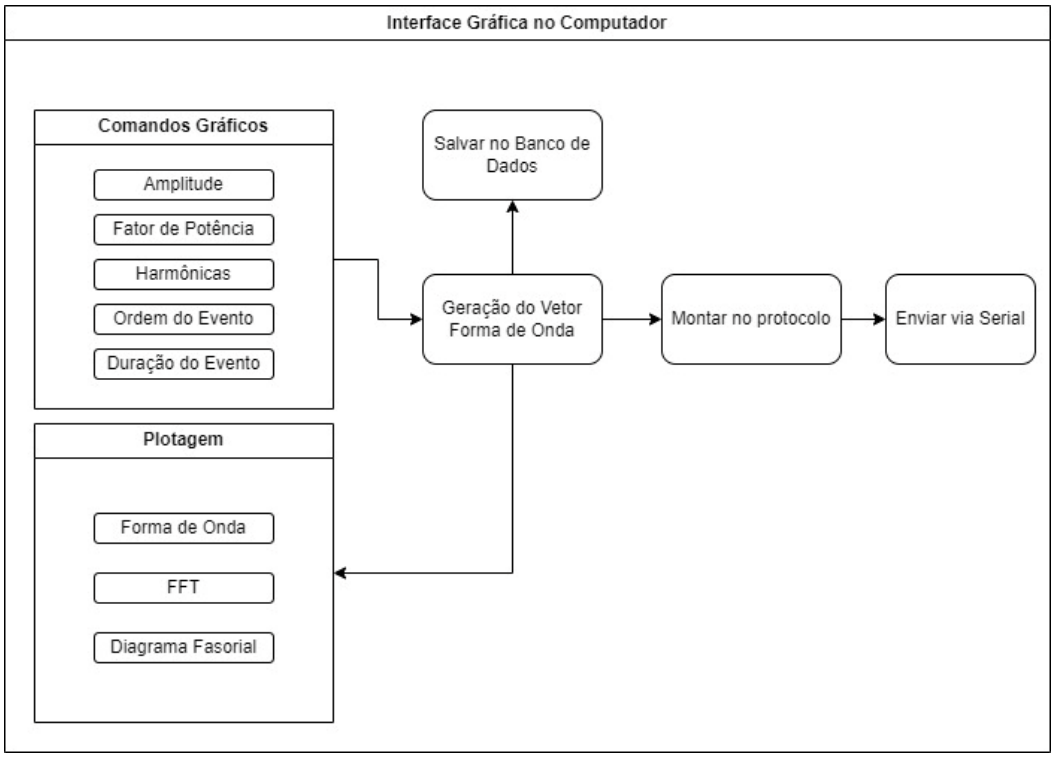
O capítulo de implementação do projeto está dividido basicamente em sete etapas:

a) Diagrama em bloco e fluxograma da interface gráfica;
b) Fluxograma do Firmware;
c) Diagrama em blocos e dimensionamento do hardware;
d) Cenários de testes;
e) Comunicação entre o software e o dispositivo;
f) Inicialização da Comunicação SPI;

3.1. Diagrama em Blocos da Interface de Gráfica de Usuário

Na Figura 19, é apresentado o diagrama em bloco das funcionalidades do software, nele é possível ver que o dispositivo é programado de forma gráfica, e o cálculo das formas de onda ocorre todo no computador, então são gerados os vetores foram de onda que são plotados para o usuário na tela, junto de diagrama fasorial e espectro de frequências, contendo a FFT de cada um dos sinais, o vetor é então montado junto do protocolo e enviado via porta serial para o dispositivo.

Figura 19: Diagrama em Blocos do Funcionamento do Software



Fonte: (Elaborado pelo autor, 2023)

No APÊNDICE F, é possível visualizar como as etapas de funcionamento ocorrem no software, no momento em que os parâmetros da fonte são inseridos pelo usuário, e a conexão entre a interface e o dispositivo é garantida. O funcionamento padrão do software esperado é descrito a seguir.

Utilizando o método calcWaveForm, ilustrado abaixo, é feito o cálculo da forma de onda, já considerando as harmônicas que podem ter sido previamente inseridas.

```
def calcWaveForm(self):
        n_rad = self.eixoN * self.fo * pi / (self.samples * 30.0)
        self.vetorWaveForm = np.round(self.amp * np.sin(n_rad +
self.angulo), 3)
        if self.flagHarmonica:
            for i in range(len(self.harmonica)):
                n_rad = self.eixoN * self.fo * pi * (2 + i) /
(self.samples * 30.0)
                self.vetorWaveForm += np.round(self.amp *
self.harmonica[i] * np.sin(n_rad + self.angulo) / 100, 2)
```

Feito isso, é calculada então a FFT de cada uma das formas de onda, conforme trecho do código ilustrado abaixo.

```
    def calcFFT(self):
        self.flagHarmonica = True
        self.calcWaveForm()
        self.vetorFFT = abs(np.fft.fft(self.vetorWaveForm,
self.samples)) / (self.samples / 2)
        self.vetorFFT = self.vetorFFT[0:self.samples / 2]
```

Em posse das formas de onda calculadas, os métodos de plotagem são chamados, quais sejam:

- plotWaveForm – Responsável por plotar na interface do usuário o gráfico da forma de onda:

```
    def plotWaveForm(self):
            a = []
            for valor in self.vetorWaveForm:
                a.append(int(valor * 0.7 * 2047 / 127) +
    2047)
            print 'vetor[i] = {' + str(a)[1:-2] + '}'
```

- plotFFT – Responsável por plotar o espectro de frequências:

```
        def plotFFT(self, nome):
            if self.amp != 0:
                self.calcFFT()
                flagHarmonica = True
                fig, ax = plt.subplots()
                ax.plot(self.eixoFreq, self.vetorFFT,
    '.')
                ax.grid()
                ax.set(xlabel='Frequencia', ylabel=nome,
                       title=nome)
                major_ticks = np.arange(0, 1000, 60)
                major_ticks = np.append((major_ticks),
    (np.arange(1020, 2470, 120)))
                ax.set_xticks(major_ticks)
                plt.xlim(0, 2470)
                canvas = FigureCanvas(fig)
                plt.close(fig)
                plt.cla()
                plt.clf()
    return canvas
    else:
                return False
```

- plotWaveformFases – Responsável por plotar o diagrama fasorial:

```
        def plotWaveFormFases(self, waveform_2,
    waveform_3, nome):

            if nome == 'tensao':
                labels = ['VA', 'VB', 'VC', 'Tensao(V)',
    'TENSAO']
            else:
                labels = ['IA', 'IB', 'IC', 'Corrente(A)',
    'CORRENTE']

            fig, ax = plt.subplots()
            ax.plot(self.eixoN, self.vetorWaveForm, '-',
    label=labels[0])
            ax.plot(self.eixoN, waveform_2.vetorWaveForm,
    '-', label=labels[1])
            ax.plot(self.eixoN, waveform_3.vetorWaveForm,
    '-', label=labels[2])
            leg = plt.legend(loc='upper right', ncol=3,
    mode="normal", shadow=True, fancybox=True)
            leg.get_frame().set_alpha(0.5)
            ax.set(xlabel='Amostras', ylabel=labels[3],
                   title=labels[4])
            ax.grid()
            major_ticks = np.arange(0, 130, 16)
            ax.set_xticks(major_ticks)
            canvas = FigureCanvas(fig)
            plt.close(fig)
            plt.cla()
            plt.clf()
            return canvas
```

Uma vez que o botão de “Enviar” é pressionado na interface, o método create_jason cria um arquivo que concatena todos os vetores junto de seus respectivos cabeçalhos chamando durante seu funcionamento o método build_protocol. Após essa etapa, o método denominado get_vetor_waveform captura

esses vetores e utilizando o método denominado send_protocol. Então os dados são enviados para o dispositivo.

Com todas as etapas supracitadas atendidas basta o usuário clicar no botão “Iniciar” e uma janela com a contagem de tempo de cada um dos eventos abre e através dela é possível monitorar o andamento dos eventos.

3.2. Fluxograma do Firmware

O funcionamento do código embarcado pode ser entendido através do fluxograma presente no APÊNDICE E, no mesmo é possível observar que no firmware existe a função principal onde é executado um *loop* infinito de forma síncrona, e duas interrupções principais, uma delas da Porta Serial huart1, esta responsável por receber pacotes provenientes da interface de usuário, e uma interrupção que ocorre a cada *overflow* do temporizador.

A função principal não executa praticamente nenhuma de suas funções sem receber estímulos provenientes da interrupção da porta serial. Sem estímulos a única função do *loop* é reproduzir de forma infinita a função “ondasPadrao”, sendo essa responsável por gerar as formas de onda calculadas para gerar formas de onda defasadas 120º entre si com fator de potência 0 na saída do dispositivo, garantindo assim um funcionamento mínimo do dispositivo quando não estiver ligado a interface de usuário.

A interrupção da porta de recepção da serial tem como função receber os pacotes provenientes da serial e trata-los segundo o protocolo que será aprofundado no tópico 3.5 dessa mesma seção.

De acordo com a operação a ser realizada um cabeçalho diferente é gerado identificando a função que deve ser executada, podendo ser:

- Sincronização de Relógio – O horário do computador é capturado e enviado ao dispositivo que utiliza para ajustar o horário do RTC;
- Atualização de Forma de Onda – Nesse modo são recebidos diversos pacotes contendo as formas de onda e horário de início de cada evento;

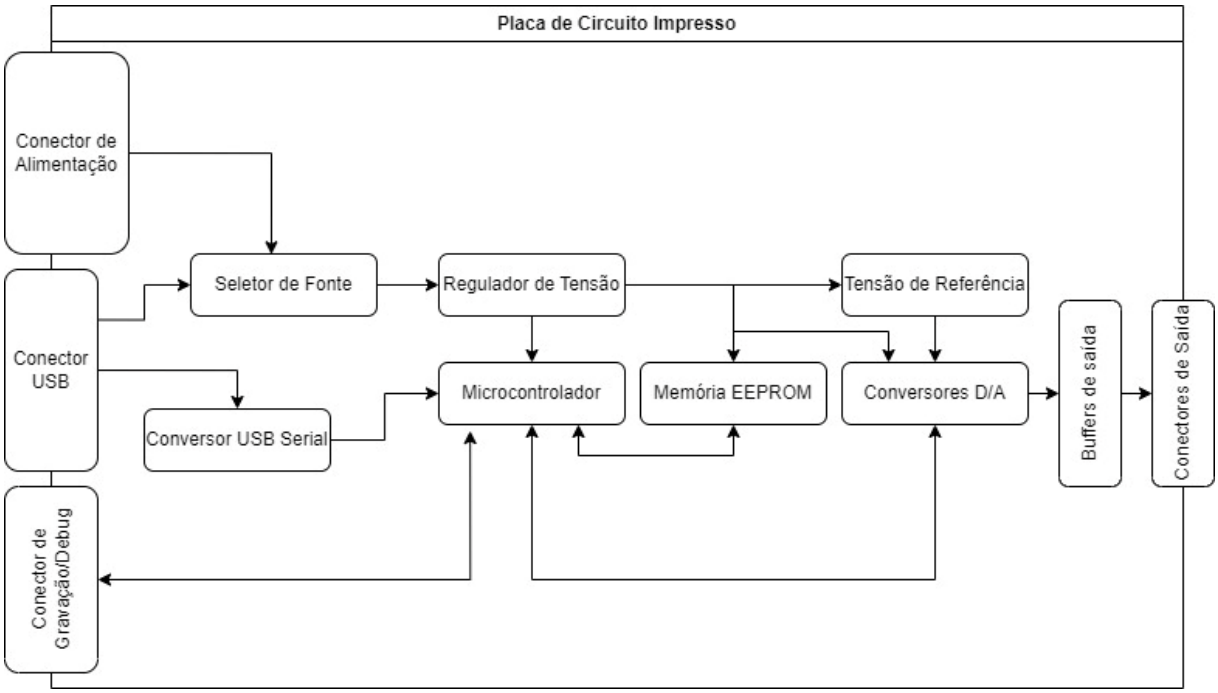
- Execução Imediata – Nesse modo são recebidos o horário atual do computador e a forma de onda que estiver presente na tela do usuário no momento que o comando for enviado.

E por fim, a interrupção que ocorre todas as vezes que o temporizador atinge o *overflow* estabelecido. Ele tem por objetivo externar as amostras armazenadas no *buffer* de saída que compõem a forma de onda dentro o período necessário parar gerar sempre a forma de onda com a frequência fundamental de 60Hz.

3.3. Diagrama em Blocos e Dimensionamento do Hardware

O diagrama em blocos da figura 20, mostra o escopo do circuito presente no esquemático completo do APÊNDICE A, que de forma resumida apresenta os conectores de alimentação, comunicação (USB) e gravação/depuração do gerador de sinais. Os conectores de alimentação e comunicação são utilizados majoritariamente pelo usuário, e o de gravação/depuração, foram utilizados no processo de desenvolvimento do dispositivo.

Figura 20: Diagrama em blocos do hardware



Fonte: (Elaborado pelo autor, 2023)

O circuito conta com um seletor simples de fonte que foi implementado pelo uso de do diodo schottky supracitado, além de contar com o microcontrolador, memória, conversores D/A e circuitos de bufferização e eliminação de nível DC também supracitados.

O modelo do microcontrolador foi escolhido com base na quantidade de memória RAM disponível para realização das atividades. O principal cuidado foi em relação a capacidade que o dispositivo deveria ter em manipular os vetores que continham as informações dos pontos de cada forma de onda. Dessa maneira, como requisito, foi estabelecido que o número mínimo de pontos que cada forma de onda teria seria de 256 pontos, o que representa no contexto do microcontrolador 256 bytes por forma de onda. Como estamos trabalhando com 6 canais, sendo três para tensão e três para corrente chegamos com facilidade no cálculo:

$$RAM\ min = 256\ \text{bytes/amostra} * 6\ \text{canais}\ = 1536\ \text{bytes} \quad (7)$$

Desse modo, é considerado apenas o pacote com as amostras que compõem o sinal, para um cálculo mais exato devemos considerar outros fatores, como o tamanho do cabeçalho que irá constituir a comunicação entre o dispositivo e a I.H.M., além de outras atividades que podem ser executadas em paralelo no microcontrolador, como a utilização da comunicação SPI entre o microcontrolador e a memória, e o microcontrolador e os conversores D/A.

Assim, considerando boas margens de segurança, ficou determinada uma memória de 8Kbytes.

A memória EEPROM por sua vez foi determinada considerando o tamanho de cada pacote de dado com seu respectivo cabeçalho. Conforme descrito na Tabela 2, descrita no tópico 3.5 deste trabalho, o pacote a ser armazenado na memória contém um total de 260 Bytes, como requisito de projeto foi estabelecido que o dispositivo deveria ser capaz de armazenar até cinco eventos, contendo cada três canais de tensão e três canais de corrente, o tamanho mínimo em *bits* que a memória deveria ter seria de:

$$TamMinMemo = 260\ \text{bytes} * 6\ \text{canais} * 5\ \text{Eventos} * 8 = 61440\ \text{bits} \quad (8)$$

Então para isso uma memória de 64Kbits ou 64000 bits, consegue atender com certa margem de segurança.

Por fim, era necessário garantir que tanto os conversores D/A, quanto os amplificadores operacionais tivessem a capacidade de reproduzir as formas de onda amostradas. Para isso, foi tomado o pior caso, que seria o caso da frequência fundamental de 60Hz multiplicada por sua 40ª harmônica, o que resulta em uma frequência de 2.400Hz. Assim sendo, utilizando o critério de Nyquist, chegamos a uma amostragem mínima de 4.800Hz por canal, então para o dimensionamento do conversor e do amplificador operacional, foi escolhido um conversor de 1 MHZ ou 1.000.000 Hz, um valor muito maior do que o necessário para garantir o funcionamento de todos os 6 canais com folga.

3.4. Cenários de Testes

Dois cenários de testes foram estabelecidos para a validação do dispositivo, um deles inclui um medidor de energia de testes, esse é considerado o teste com carga, e o outro é o cenário do dispositivo ligado diretamente ao osciloscópio.

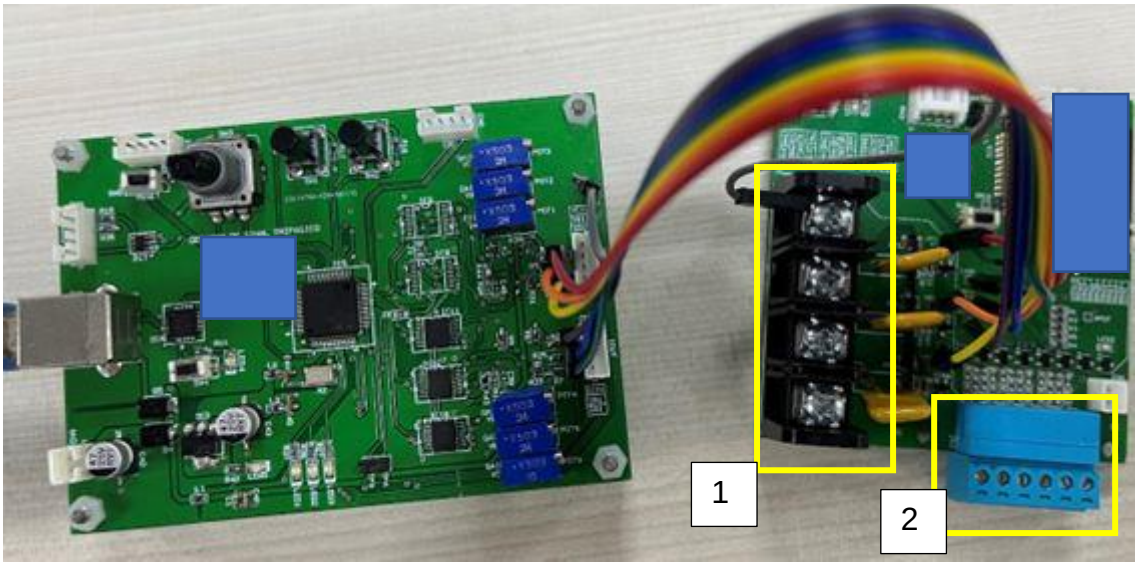
3.4.1. Ligação do Dispositivo ao Medidor de Testes

Devido à natureza sigilosa dos medidores de energia utilizados, as imagens estão parcialmente ofuscadas, de forma a preservar a imagem da empresa onde as placas foram fabricadas.

Na Figura 21 é possível observar o dispositivo gerador de sinais ligado ao medidor de energia na etapa de pós condicionamento, e isso pode ser observado pois as ligações feitas entre o gerador e o medidor são todas feitas por meio de sondas que ignoram os conectores principais de tensão (destacado no bloco 1) e corrente (destacado no bloco 2).

O medidor de testes disponibilizado possui o CI ADE7758, e uma interface gráfica para coleta de amostras de energia e calibração. Os resultados das leituras estarão descritos no tópico denominado RESULTADOS.

Figura 21: Dispositivo Gerador de Sinais Ligado a um Medidor de Energia de Testes



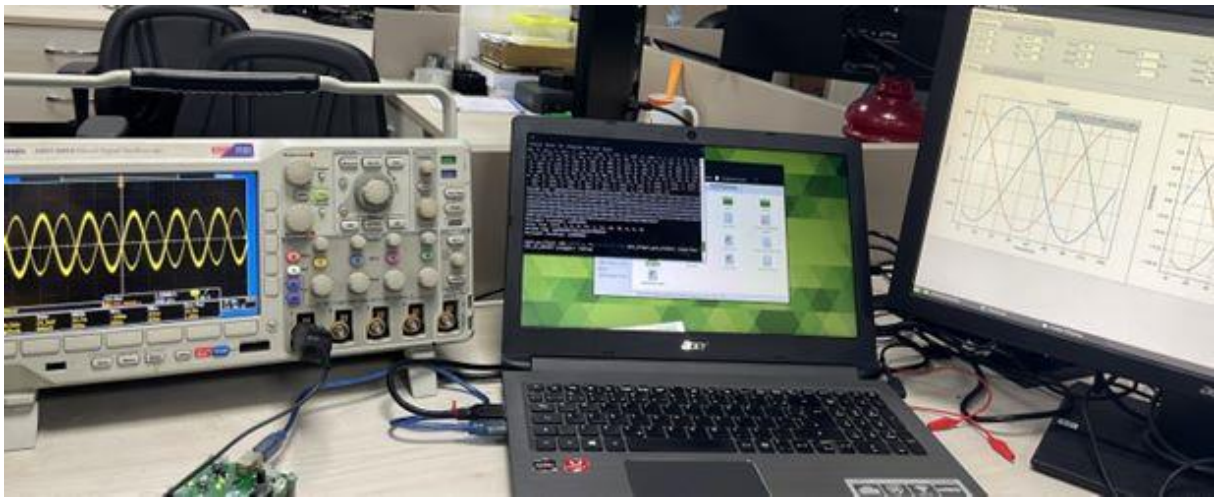
Fonte: (Elaborado pelo autor, 2023)

3.4.2. Ligação do Dispositivo ao Osciloscópio

Na figura 22 é possível observar o cenário de testes do dispositivo sem a presença da carga. Nesse caso seria apenas o dispositivo com seus canais diretamente ligados ao osciloscópio e à sua interface de controle gráfica.

Os resultados dos testes também serão apresentados no tópico denominado RESULTADOS.

Figura 22: Cenário de Testes sem Carga



Fonte: (Elaborado pelo autor, 2023)

3.5. Comunicação Entre o Software e o Dispositivo

A comunicação entre o software e o dispositivo é realizada por meio de um protocolo proprietário. Na Tabela 2 é descrita a estrutura geral do pacote entre interface e o dispositivo.

Tabela 2: Estrutura Geral do Pacote – Interface → Dispositivo

PAYLOAD			
Item	**Nome**	**Tam (Byte)**	**Descrição**
1	Header	1	Identificador de dado válido
2	ID função	1	Identificador da função
3	Tamanho_Pacote	2	Tamanho do Pacote
4	Pacote	-	Pacote de Dados
Tam_Cabeçalho		4	

Fonte: (Elaborado pelo autor, 2023)

Conforme descrito na Tabela 2, estabelecemos que o vetor a ser enviado sempre terá em sua estrutura um byte de identificador válido, o identificador da função, o tamanho do pacote e o pacote em si.

Depois disso precisamos mapear os identificadores de função que serão utilizados conforme descrito na Tabela 3.

Tabela 3: Identificadores de Função

COMPUTADOR → STM	
Descrição	**Valor**
AtualizarFormaOnda()	**0x00**
Sincronizar_RTC()	**0x02**
Executar_Imediatamente()	**0x04**

Fonte: (Elaborado pelo autor, 2023)

Cada um dos identificadores é tratado posteriormente na interrupção da porta serial presente no firmware da fonte, conforme ressaltado no APÊNDICE E.

3.5.1. Função “maquina()” – Firmware

Essa é a função que é executada a cada vez que ocorre uma interrupção no pino de recepção da porta serial. Nesta função, a estrutura mais importante é a denominada *switch-case,* visto que é por meio dessa estrutura o identificador de função é tratado.

Nela, a segunda posição do vetor bufferRx, que é o vetor de entrada é selecionada, e de acordo com o valor presente, ela escolhe entra as 3 funções possíveis supracitadas.

No primeiro caso, o código solicita que o restante do *buffer* recebido seja alocado em uma variável auxiliar chamada *output.* Esta variável é responsável por coletar as amostras e dentro do loop executar a escrita da amostra no *buffer* que será armazenado na memória.

No segundo caso, o buffer é particionado de forma que o RTC seja atualizado com o horário enviado pela interface.

E no último caso, os valores do RTC são atualizados e o último pacote de forma de onda enviado é executado. A seguir é ilustrado o trecho do código que realiza a função denominada “máquina”.

```
      switch(bufferRx[1])
      {
          case 0x00://Atualizar Forma de Onda
           tempo = bufferRx[6] + bufferRx[7]*256;

           for(int i = 0; i <= 127; i++)
           {
             outPut = bufferRx[8 + 2*i] + bufferRx[9 +
2*i]*256 ;
             if (i == 127)
               wflag = 1;
           }
           HAL_UART_Transmit(&huart1,bufferTx,4,1);
           break;

          case 0x02://Atualizar Relógio
           HAL_UART_Transmit(&huart1,bufferTx,4,1);
           DateToUpdate.Month = bufferRx[5] ;
           DateToUpdate.Date = bufferRx[6];
           DateToUpdate.Year = bufferRx[4];
           sTime.Hours = bufferRx[7];
           sTime.Minutes = bufferRx[8];
           sTime.Seconds = bufferRx[9];
           HAL_RTC_SetTime(&hrtc, &sTime, RTC_FORMAT_BIN);
           HAL_RTC_SetDate(&hrtc, &DateToUpdate,
RTC_FORMAT_BIN);

           break;

           case 0x04://Execução Imediata
```

```
            HAL_UART_Transmit(&huart1,bufferTx,4,1);
           Ano = bufferRx[4];
           Mes = bufferRx[5];
           Dia = bufferRx[6];
           Hora = bufferRx[7];
           Minuto = bufferRx[8];
           Segundo = bufferRx[9];
           executar = bufferRx[10];

          break;
      }
```

3.6. Inicialização da Comunicação SPI

A inicialização da SPI se faz necessária, uma vez que tanto os conversores D/A quanto a memória funcionam por meio da comunicação SPI.

Como mencionado anteriormente, a inicialização de todos os periféricos do STM, foi feita com o auxílio da ferramenta STM32CubeMx, nela as comunicações desejadas são configuradas de forma gráfica, incluindo a opção de habilitar a interrupção daquele periférico.

Uma vez que a comunicação tenha sido habilitada na ferramenta, ela automaticamente gera no código a inicialização do periférico apresentado a seguir.

```
static void MX_SPI1_Init(void)
{
  /* Inicialização da Comunicação SPI*/
  hspi1.Instance = SPI1;
  hspi1.Init.Mode = SPI_MODE_MASTER;
  hspi1.Init.Direction = SPI_DIRECTION_2LINES;
  hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
  hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
  hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
  hspi1.Init.NSS = SPI_NSS_SOFT;
  hspi1.Init.BaudRatePrescaler =
SPI_BAUDRATEPRESCALER_2;
  hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
  hspi1.Init.CRCCalculation =
SPI_CRCCALCULATION_DISABLE;
  hspi1.Init.CRCPolynomial = 10;
  if (HAL_SPI_Init(&hspi1) != HAL_OK)
  {
    Error_Handler();
  }
}
```

3.6.1. Escrita e Leitura das formas de onda na memória

Para escrever as formas de onda junto de seus respectivos horários de execução na memória é utilizada a função "*WriteMemo()".* Para ela são passados como parâmetro o dado que será escrito, o endereço inicial de escrita da memória, e um índice para manipulação interna dos ciclos de escrita conforme apresentado no código a seguir.

```
void writeMemo(uint8_t date, uint16_t adress, int i)
{
        uint8_t bufferOut[4];
        uint8_t dado;

        dado = 0x00 & dado;
        dado = 0x06 | dado; //EnableWrite

        bufferOut[0] = dado;
        bufferOut[1] = adress >> 8;
         bufferOut[2] = adress;

        GPIOB->BSRR = (uint32_t)CS_EEPROM_Pin <<
16U;//desliga o cs1
           HAL_SPI_Transmit(&hspi2,bufferOut, 3, 1);
           GPIOB->BSRR = CS_EEPROM_Pin;//liga o cs1

        dado = 0x00 & dado;
           dado = 0x02 | dado;  //comando para escrita
        bufferOut[0] = dado;

        bufferOut[3] = date;
        GPIOB->BSRR = (uint32_t)CS_EEPROM_Pin <<
16U;//desliga o cs1
           HAL_SPI_Transmit(&hspi2,bufferOut, 4, 1);
           GPIOB->BSRR = CS_EEPROM_Pin;//liga o cs1
           dadoOut[i] = bufferOut[3];

}
```

Já para realizar a leitura das formas de onda já salvas na memória para usar é utilizada a função "*readMemo()",* para essa função é passado apenas o endereço inicial e um índice, que será utilizado para varredura da memória conforme apresentado no código a seguir.

```
uint8_t readMemo(uint16_t adress,int i)
{
        uint8_t bufferOut[4];
        uint8_t dado;
     uint8_t resposta[2];
        dado = 0x00 & dado;
           dado = 0x03 | dado;  //comando para leitura
        bufferOut[0] = dado;
           bufferOut[1] = adress >> 8;
           bufferOut[2] = adress;
        GPIOB->BSRR = (uint32_t)CS_EEPROM_Pin <<
16U;//desliga o cs1
           HAL_SPI_Transmit(&hspi2,bufferOut, 3, 1);
        HAL_SPI_Receive(&hspi2,&bufferOut[3],1,1);
           GPIOB->BSRR = CS_EEPROM_Pin;//liga o cs1
        resposta[0] = bufferOut[3];
        dadoOut[i] = resposta[0];
        return dadoOut[i];
}
```

3.6.2. Escrita da Forma de Onda no Respectivo Canal de Saída

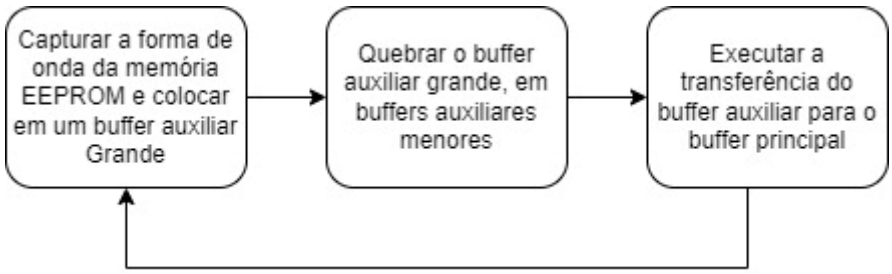
O processo de escrita do sinal na saída do conversor D/A foi dividido em etapas, conforme ilustrado no fluxograma da figura 23. Na primeira etapa a forma de onda é carregada da memória para o buffer de saída. Na segunda etapa, a próxima forma de onda a ser executada é carregada para um buffer auxiliar de saída. Isso é feito para que o próximo sinal que será executado esteja na memória RAM do dispositivo, evitando descontinuidade entre os sinais que ocorrem em um evento devido a latência gerada no processo de leitura.

Esse processo é feito por meio de várias funções, que podem ser observadas no APÊNDICE B, sequenciadas dessa forma:

- A função *getFromMemo()* é utilizada para capturar a forma de onda salva na memória EEPROM para um *buffer* auxiliar utilizando a função *readMemo();*
- Ainda dentro da função *getFromMemo(),* o buffer auxiliar é agora passado como parâmetro para a função *genWave(),* dessa forma os valores das amostras armazenados no buffer auxiliar são partidos em buffers auxiliares menores, um para cada canal de tensão e corrente;

- Quando chega o momento da forma de onda ser executada, a função *transferWave(),* realiza o papel de trazer os sinais armazenados nos buffers menores auxiliares para os buffers de saída dos conversores D/A.
- Com isso o ciclo se repete;

Figura 23: Fluxograma de Escrita no Conversor D/A



Fonte: (Elaborado pelo autor, 2023)

3.6.3. Início da Execução dos Eventos

Para que seja dado início a execução dos eventos programados basta que o horário verificado do RTC seja igual ao especificado pelo usuário, essa verificação é feita dentro do *loop* principal conforme apresentado no trecho de código a seguir.

```
else if((gTime.Hours*3600 + gTime.Minutes*60 + gTime.Seconds)
== (Hora * 3600 + Minuto*60 + Segundo))
    {
                transferWave();
                getFromMemo();
                Hora = Hora + userh;
                Minuto = Minuto + userm;
                Segundo = Segundo + users;
                if(Segundo >= 60)
                {
                     Minuto++;
                  Segundo = Segundo - 60;
                }
                if(Minuto >= 60)
                {
                     Hora++;
                  Minuto = Minuto - 60;
                }
                tempo = tempoAux;
```

```
                arrumatempo(tempo);
                HAL_TIM_Base_Start_IT(&htim4);
        }
```

4. RESULTADOS

Os resultados podem ser separados para os dois cenários de testes supracitados, sem carga utilizando apenas o osciloscópio para validar os sinais de saída do gerador, e com carga utilizando o medidor para fazer a aquisição de amostras de leitura de energia.

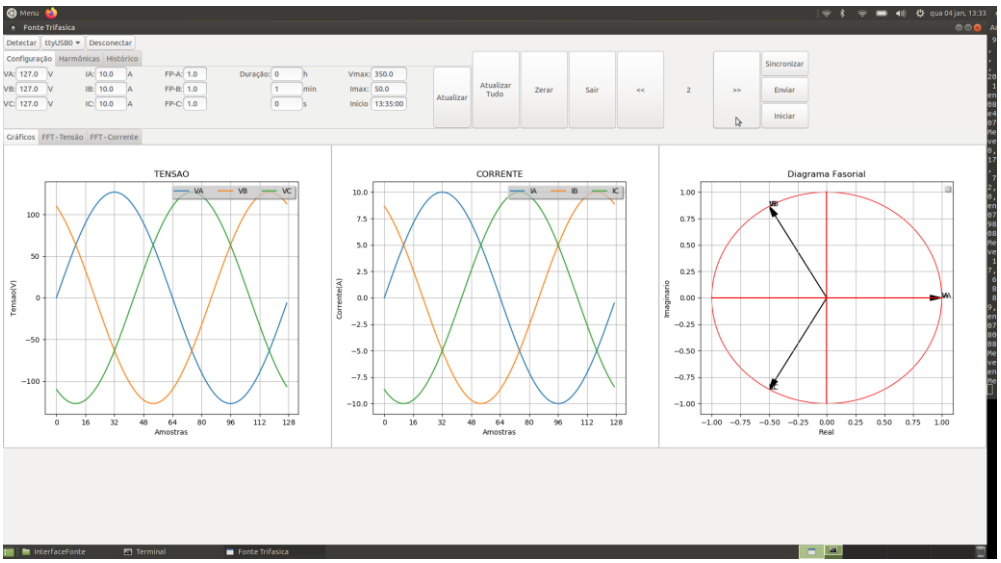
Para o caso sem carga, os resultados estão ilustrados nas figuras 24 a 31. Na figura 24 está ilustrado o evento de normalidade gerado na interface gráfica e a forma de onda captura para este evento está ilustrada na figura 25. Conforme ilustrado na figura 24, os parâmetros gerados foram:

- 127 Vrms por fase;
- Todas as fases defasadas 120º entre si;
- Todas as frequências em 60Hz.

Os resultados obtidos vistos na figura 25 foram:

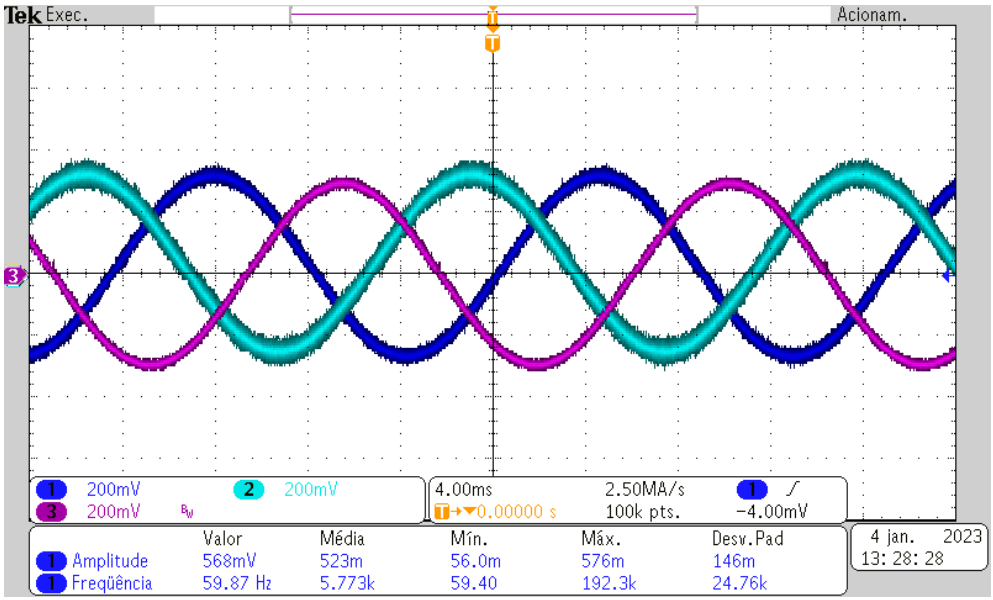
- Em torno de 184 mVRMS nas fases;
- 120º de defasagem;
- Frequências em 60 Hz;

Figura 24 – Evento Normalidade Gerado na Interface Gráfica



Fonte: (Elaborado pelo autor, 2023)

Figura 25: Evento Normalidade Capturado no Osciloscópio



Fonte: (Elaborado pelo autor, 2023)

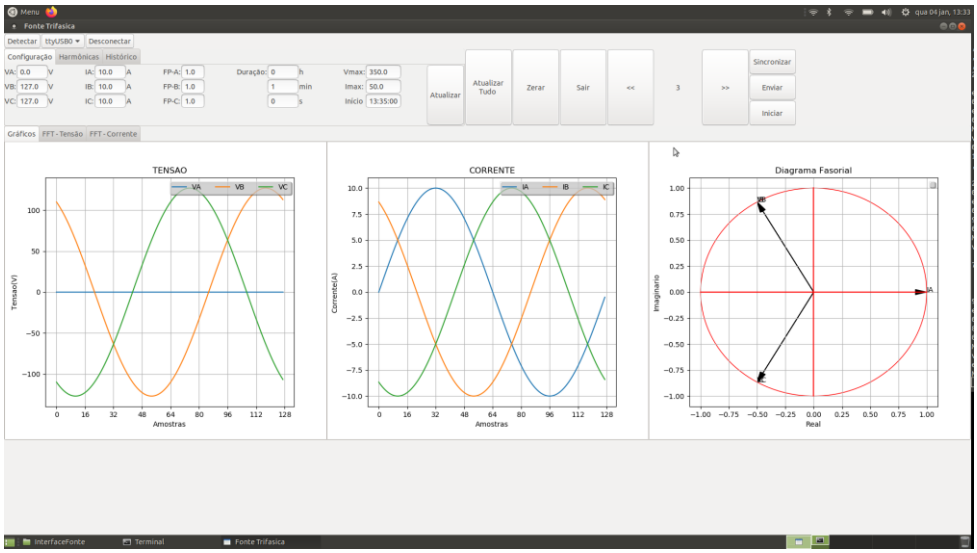
Na figura 26 está ilustrado o evento de falta de fase gerado na interface gráfica e a forma de onda captura para este evento está ilustrada na figura 27. Conforme ilustrado na figura 26, os parâmetros gerados foram:

- 127 Vrms nas fases B e C;
- Sem tensão na fase A;
- Todas as fases defasadas 120º entre si;
- Todas as frequências em 60Hz.

Os resultados obtidos vistos na figura 27 foram:

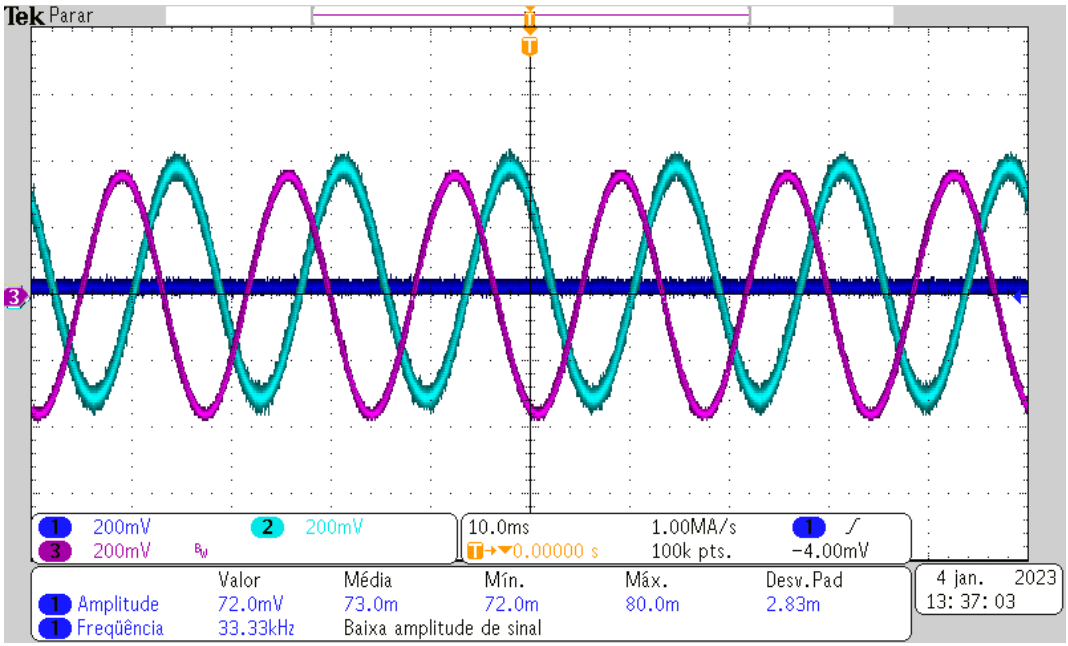
- Em torno de 184 mVRMS nas fases B e C;
- Sem tensão na fase A;
- 120º de defasagem;
- Frequências em 60º;

Figura 26: Evento de Falta de Fase Gerado na Interface Gráfica



Fonte: (Elaborado pelo autor, 2023)

Figura 27: Evento Falta de Fase Capturado no Osciloscópio



Fonte: (Elaborado pelo autor, 2023)

Na figura 28 está ilustrado o evento de sobretensão gerado na interface gráfica e a forma de onda captura para este evento está ilustrada na figura 29. Conforme ilustrado na figura 28, os parâmetros gerados foram:

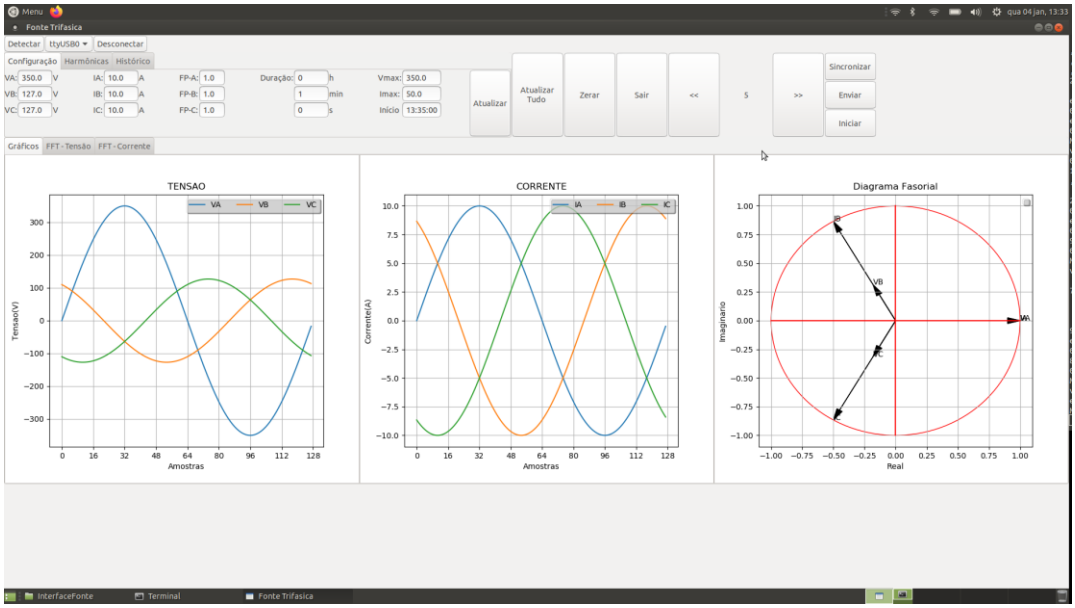
- 127 Vrms nas fases B e C;
- 350 Vrms na fase A;

- Todas as fases defasadas 120º entre si;
- Todas as frequências em 60Hz.

Os resultados obtidos vistos na figura 29 foram:

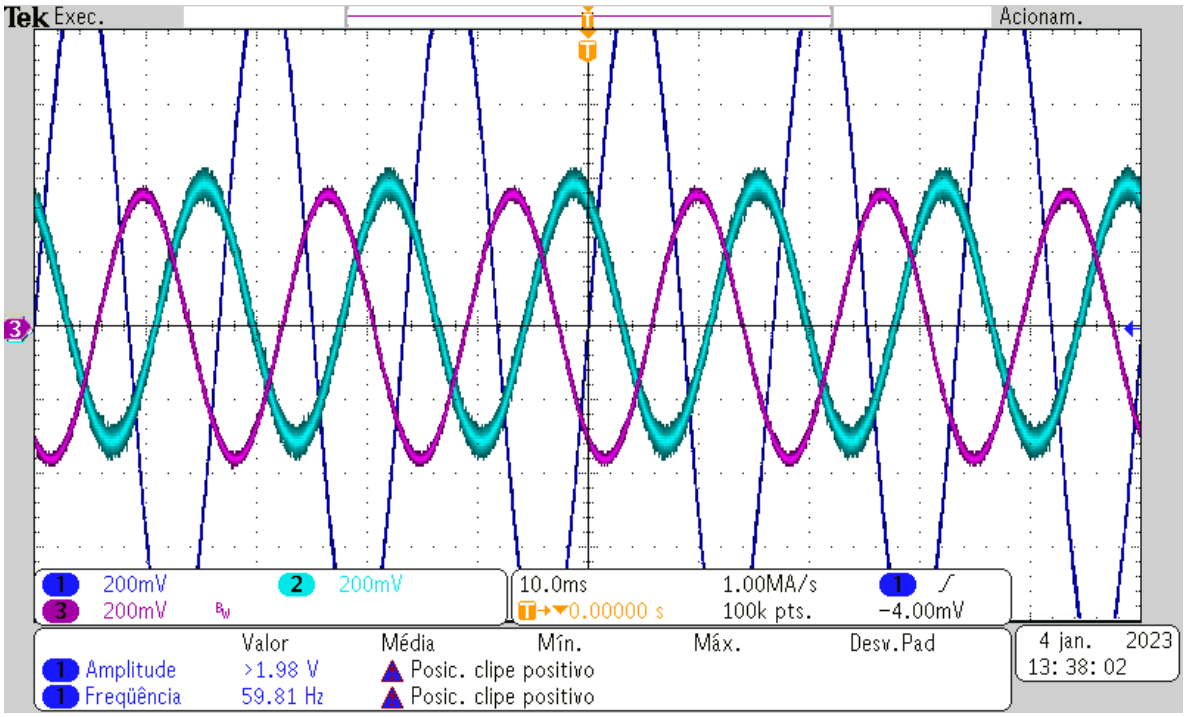
- Em torno de 184 mVRMS;
- Em torno de 565 Mvrms na fase A;
- 120º de defasagem;
- Frequências em 60º;

Figura 28: Evento de Sobretensão Gerado na Interface Gráfica



Fonte: (Elaborado pelo autor, 2023)

Figura 29: Evento Sobretensão capturado no Osciloscópio



Fonte: (Elaborado pelo autor, 2023)

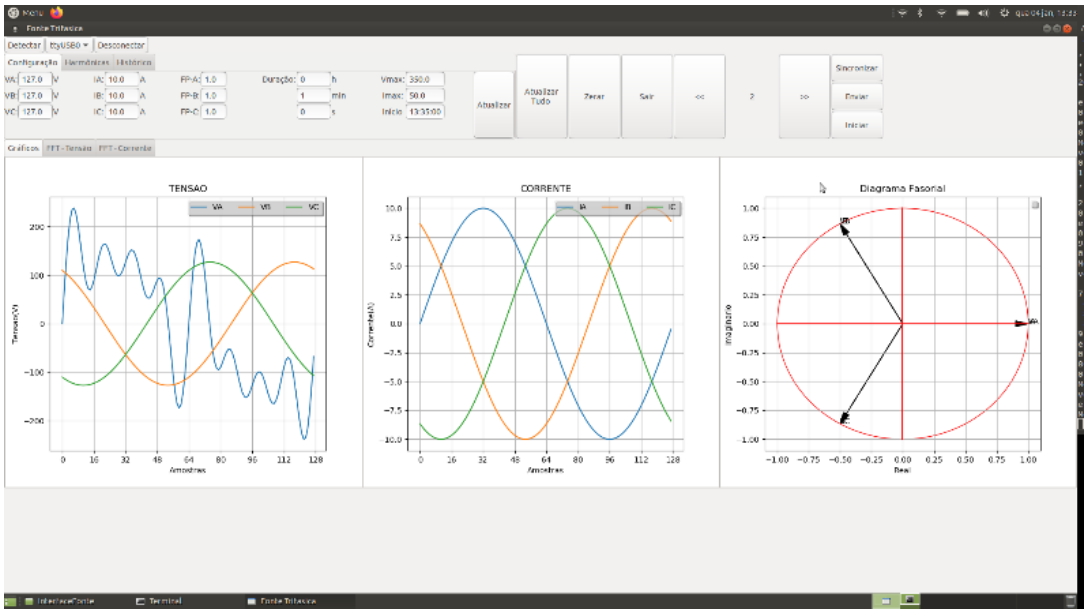
Na figura 30 está ilustrado o evento de distorção harmônica gerado na interface gráfica e a forma de onda captura para este evento está ilustrada na figura 31. Conforme ilustrado na figura 30, os parâmetros gerados foram:

- 127 Vrms nas fases B e C;
- Introdução de 20% das harmônicas 3, 5 e 7 na fase A;
- Todas as fases defasadas 120º entre si;
- Todas as frequências em 60Hz.

Os resultados obtidos vistos na figura 31 foram:

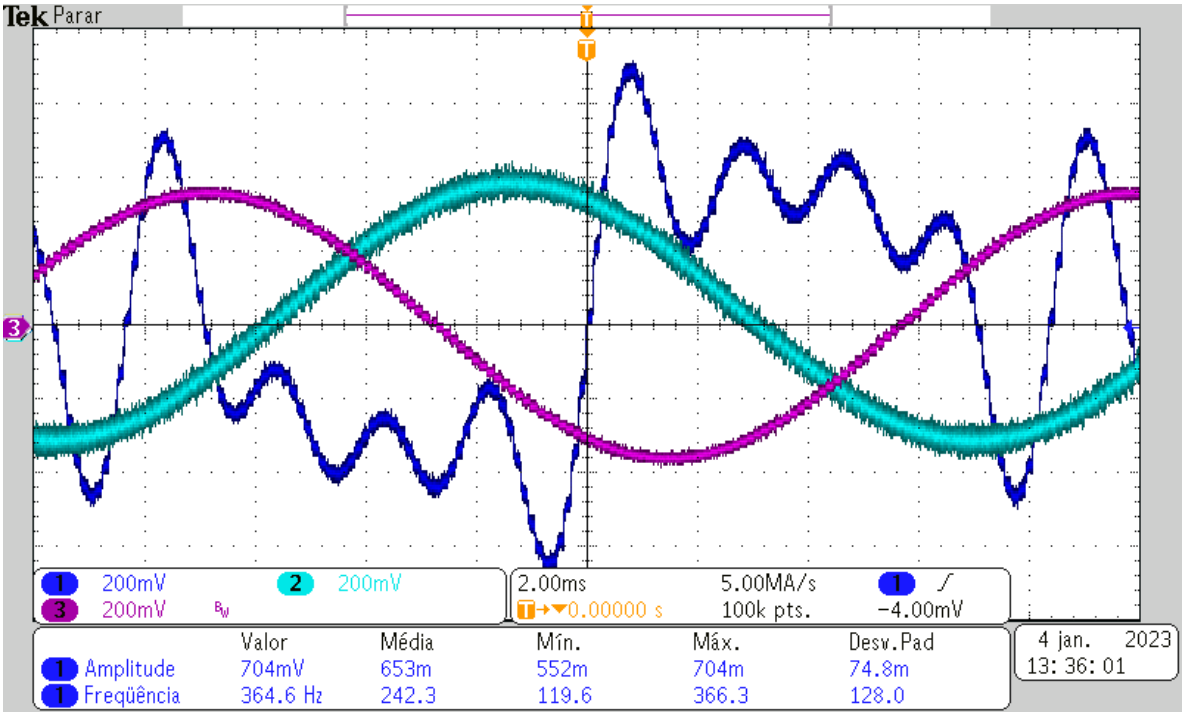
- Em torno de 184 mVRMS em cada fase;
- Harmônicas geradas na interface capturadas no osciloscópio;
- 120º de defasagem;
- Frequências em 60º;

Figura 30: Sinal com Harmônicas Gerado na Interface Gráfica



Fonte: (Elaborado pelo autor, 2023)

Figura 31: Sinal com Harmônicas Capturado no Osciloscópio



Fonte: (Elaborado pelo autor, 2023)

É possível observar que para todos os casos sem cargas, o dispositivo se comporta exatamente como o esperado, isto é, recebendo eventos sequenciados e executando-os de forma sequencial.

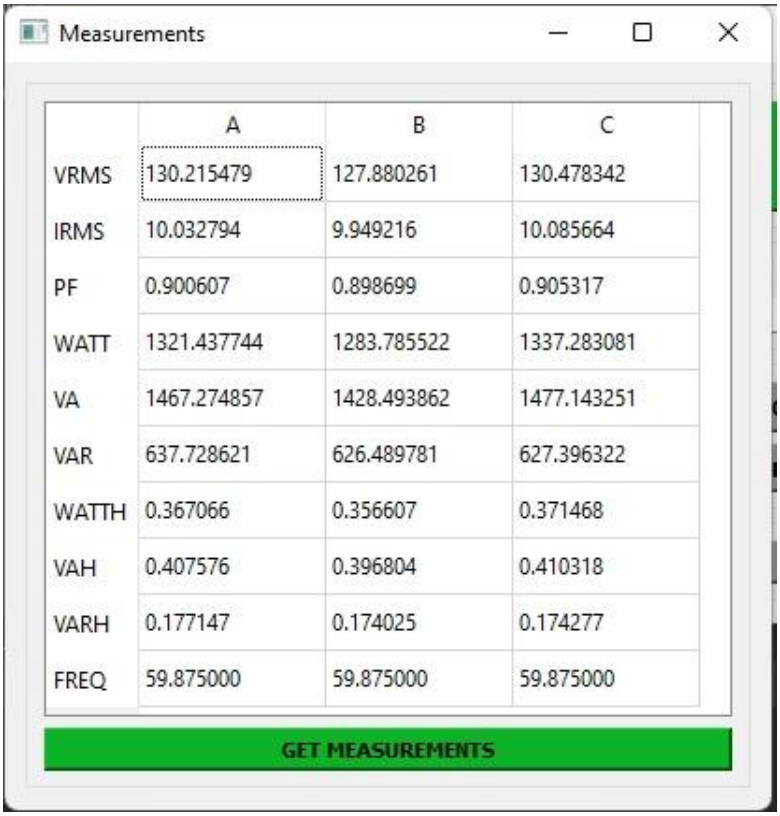
E para o caso de testes com carga, os seguintes resultados foram obtidos através do medidor de testes.

No primeiro caso de teste o medidor de energia não estava calibrado para trabalhar com o dispositivo gerador de sinais, os valores inseridos eram de:

- VA = VB = VC = 127V;
- IA = IB = IC = 10A;
- FP: 1

Os valores obtidos na medição podem ser observados na Figura 32.

Figura 32: Caso de teste 1: Medidor Não Calibrado



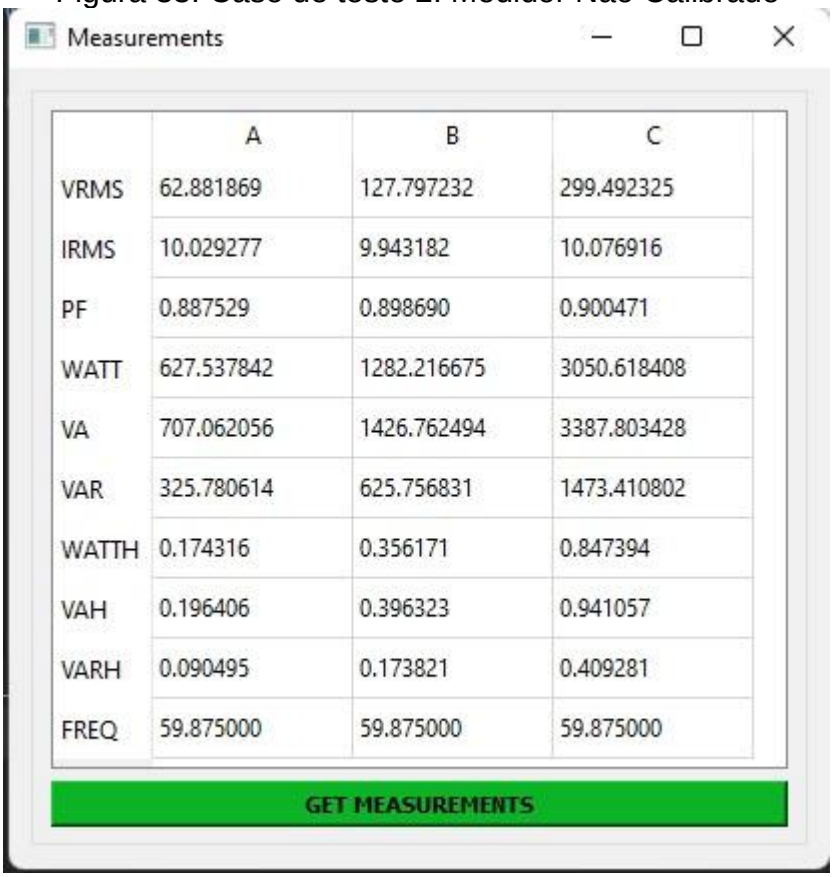
Fonte: (Elaborado pelo autor, 2023)

No segundo caso de teste o medidor de energia não estava calibrado para trabalhar com o dispositivo gerador de sinais, os valores inseridos forma iguais a:

- VA = 60V (subtensão);
- VB = 127V (normalidade);
- VC = 300V (sobretensão);
- IA = IB = IC = 10A;
- FP: 1;

Os valores obtidos na medição podem ser observados na Figura 33.

Figura 33: Caso de teste 2: Medidor Não Calibrado



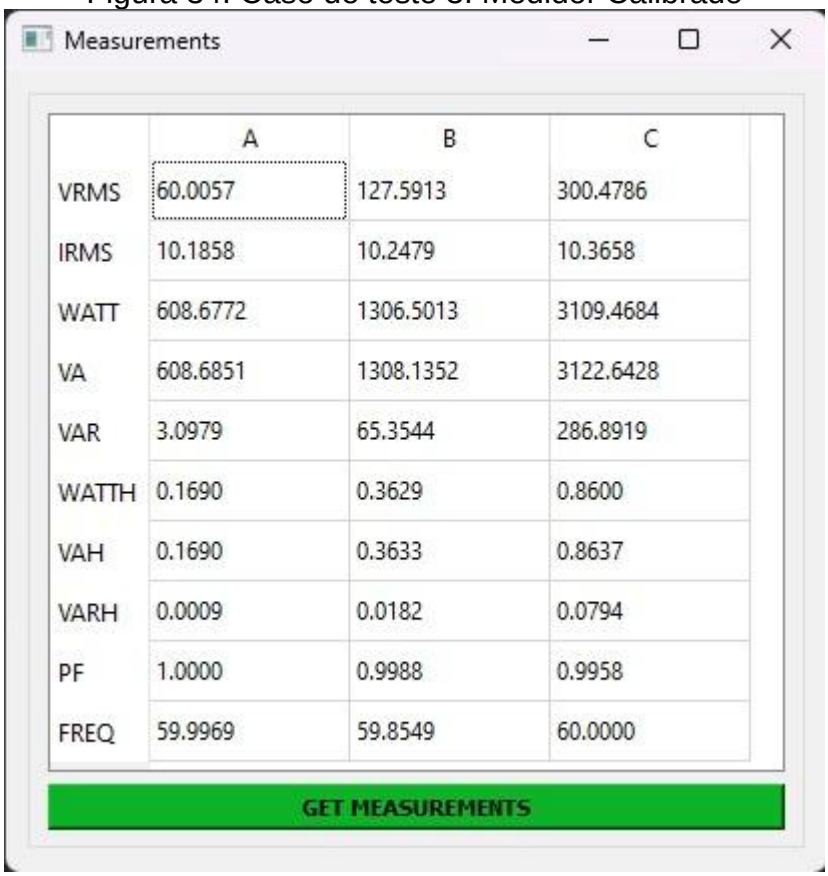
Fonte: (Elaborado pelo autor, 2023)

No terceiro caso de teste o medidor de energia estava calibrado para trabalhar com o dispositivo gerador de sinais, os valores inseridos foram iguais a:

- VA = 60V (subtensão);
- VB = 127V (normalidade);
- VC = 300V (sobretensão);
- IA = IB = IC = 10A;
- FP: 1;

Os valores obtidos na medição podem ser observados na Figura 34.

Figura 34: Caso de teste 3: Medidor Calibrado



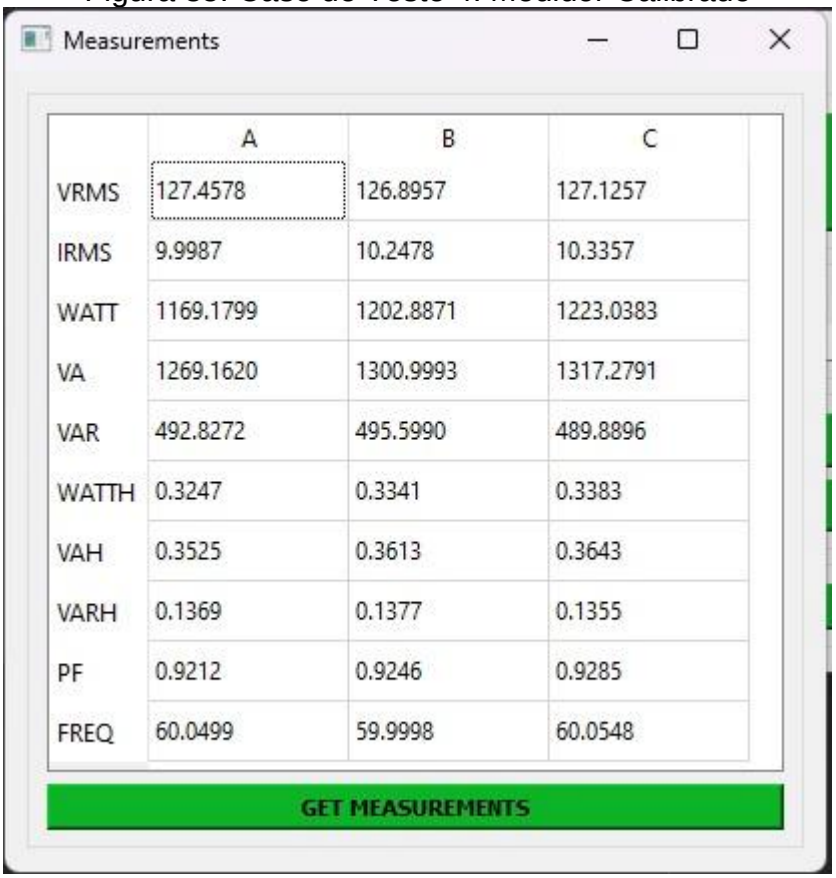
Fonte: (Elaborado pelo autor, 2023)

No terceiro caso de teste o medidor de energia estava calibrado para trabalhar com o dispositivo gerador de sinais, os valores inseridos foram iguais a:

- VA = VB = VC = 127V;
- IA = IB = IC = 10A;
- FP = 0,92

Os valores obtidos na medição podem ser observados na Figura 34.

Figura 35: Caso de Teste 4: Medidor Calibrado



Fonte: (Elaborado pelo autor, 2023)

Em todos os casos de teste, as leituras realizadas pelo medidor de energia de testes mantiveram um erro absoluto abaixo de 1%, dessa forma é possível afirmar que o dispositivo desenvolvido pode ser utilizado com confiabilidade.

4.1 Dispositivos Similares de Mercado

Inicialmente uma comparação com a fonte geradora de sinais PPS400.3 ilustrada na figura 36. Este equipamento possui 3 canais de tensão e 3 canais de corrente, bem como um bloco de medição para validar que o sinal de saída está conforme o solicitado, e também possui uma I.H.M. integrada cuja interação é feita por meio de um display de cristal líquido e um *enconder* para navegação e confirmação, esse modelo pode ser visto na Figura 36. Apesar de ser capaz de gerar distorção harmônica em seus canais, esse gerador é utilizado na etapa pré-condicionamento, ou seja, o dispositivo trabalha com tensões e correntes em

amplitude suficiente para apresentar risco de choque elétrico para o usuário. Sendo assim sua comparação com o dispositivo desenvolvido nesse trabalho é praticamente descartável, pois cada dispositivo apresenta um cerne de funcionamento diferente.

Figura 36 – Gerador de Sinais PPS400.3

Fonte: (MTE, 2022)

Assim, é mais adequado comparar o dispositivo desenvolvido nesse trabalho com o dispositivo NIMyDAQ (Figura 37), desenvolvido pela fabricante National Instruments, segundo o site da fabricante:

> Combina um conjunto abrangente de instrumentos de laboratório baseados em computador plug-and-play com portabilidade para o aprendizado prático do aluno dentro ou fora do laboratório.
>
> Os dispositivos de aquisição de dados do aluno myDAQ apresentam oito instrumentos de laboratório baseados em computador plug-and-play comumente usados com base no LabVIEW, incluindo um multímetro digital (DMM), osciloscópio e gerador de funções. Os alunos podem acessar todos os instrumentos de software prontos para executar para realizar experimentos e exercícios com o analisador Bode, gerador de forma de onda arbitrária, analisador de sinal dinâmico (transformada rápida de Fourier), entrada digital e saída digital. Esses dispositivos acessíveis permitem a engenharia real e, quando combinados com os softwares LabVIEW e NI Multisim, dão aos alunos o poder de prototipar sistemas e analisar circuitos fora de aulas e laboratórios tradicionais (NATIONAL INSTRUMENTS).

No caso o dispositivo NIMyDAQ, podemos comparar o dispositivo desenvolvido nesse trabalho, com o seu gerador de funções, ainda segundo o fabricante, as especificações que podemos comparar estão especificadas na Tabela 4.

Tabela 4 – Comparação entre o dispositivo de mercado e o desenvolvido no trabalho.

Especificações	Dispositivo Desenvolvido no Trabalho	NI-myDAQ
Resolução do DAC	12 bits	16bits
Capacidade de Gerar Harmônicas	Até 40ª	Não possui
Capacidade de Gerar Eventos	Até 5 eventos	Não possui
Tensão Máxima de Saída	±1,5V	±10V
Número de Canais	6	2
I.H.M utilizando computador	Sim	Sim
Custo	US$ 18,2	US$ 483

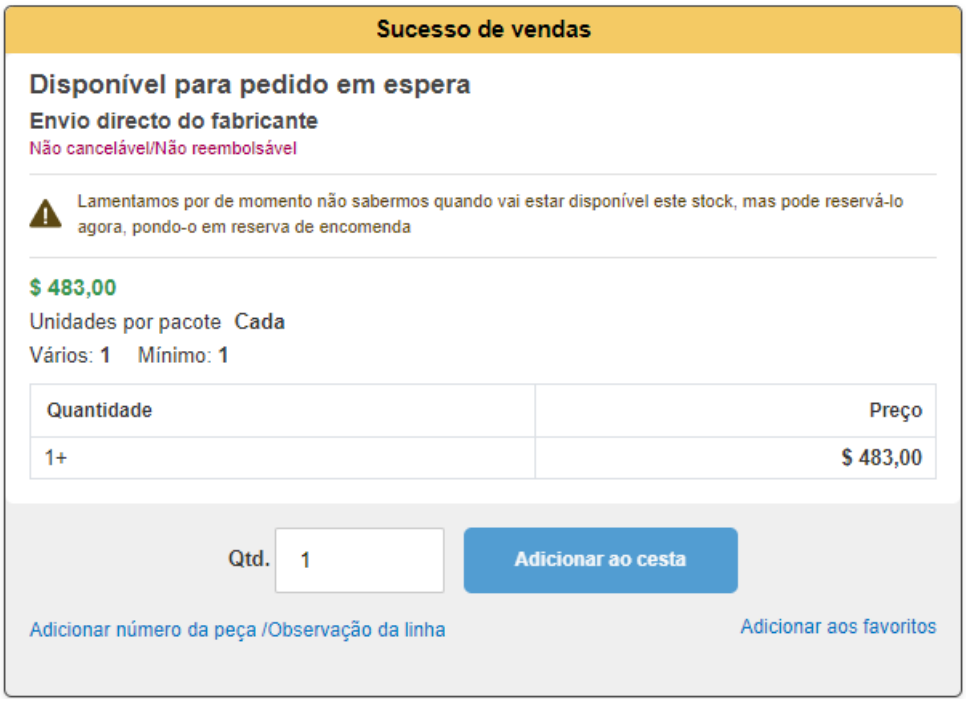
Fonte: (Elaborado pelo autor, 2023)

Os parâmetros de funcionamento do dispositivo NIMyDAQ foram retirados diretamente do site da fabricante, já seu valor foi retirado de um revendedor oficial, que pode ser observado na Figura 38, para gerar um valor aproximado do custo do dispositivo desenvolvido nesse trabalho foram utilizados o site de um revendedor internacional de componentes e do fabricante internacional de placas de circuito impresso, esses podem ser observados na Figura 39, em nenhum dos casos foram aplicados custos de transporte e taxas de importação (NATIONAL INSTRUMENTS).

Figura 37 – NImyDAQ e sua descrição breve

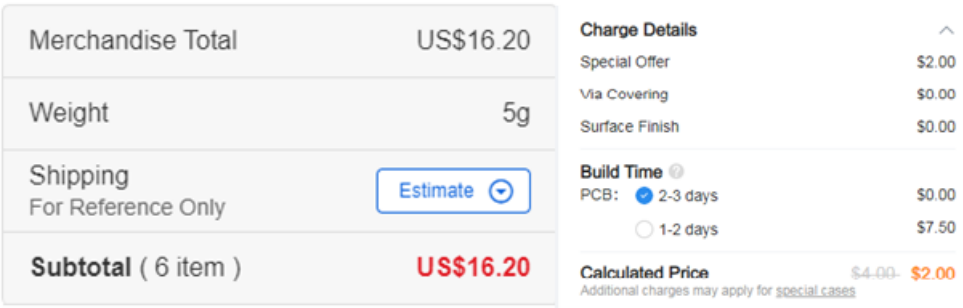
Fonte: (NEWARK)

Figura 38 - Custo do Dispositivo NImyDAQ



Fonte: (NEWARK)

Figura 39 - Custo de Fabricação do Dispositivo Gerador de Sinais



Fonte: (LCSC) (JLCPCB)

CONCLUSÃO

Este trabalho teve o intuito de desenvolver um sistema eletrônico capaz de gerar sinais de validação para um dispositivo de medição de energia na etapa de pós condicionamento.

O dispositivo deveria ser composto por um software de controle em uma I.H.M. em um computador, e uma placa de circuito impresso microcontrolada, além de ter compatibilidade com os fundos de escala dos circuitos integrados ADE7758 e ADE9000.

As funcionalidades que o dispositivo deveria apresentar seriam as seguintes: gerar sinais mesmo sem conexão com o software de controle, gerar até 5 eventos sequenciados, e gerar sinais com uma frequência fundamental de 60Hz e até a 40ª frequência harmônica.

Para validação das funcionalidades foram tomados 3 cenários, um utilizando apenas um osciloscópio visando validar que a geração dos sinais ocorria conforme o esperado, bem como o sequenciamento de eventos. Outro cenário foi a utilização do medidor de energia sem a devida calibração. E por fim, a utilização do mesmo dispositivo medidor de energia agora com a devida calibração.

A partir dos resultados é possível concluir que o dispositivo implementado apresenta características suficientes para ser considerado funcional. Além disso por se tratar de um sistema eletrônico de baixa potência, ele consegue reduzir de forma satisfatória o risco apresentado no início do projeto, isto é, a exposição do desenvolvedor de firmware a choques elétricos.

Do ponto de vista de desenvolvimento, o projeto aglutina múltiplas áreas de conhecimento estudas dentro do curso de engenharia. O dispositivo, por se tratar de um sistema microcontrolado, apresenta um funcionamento muito estável e de fácil reprodução. Como a carga de cálculos pesados fica por responsabilidade do computador externo, o microcontrolador fica responsável basicamente por processar pacotes e fazer a escrita deles no momento adequado.

O projeto em seu estágio atual pode servir como ponto de partida para um novo estágio. No contexto de testes de medidores de energia, existe um procedimento chamado *burn-in*, em que os medidores de energia ficam expostos a altas temperaturas e passam por diversas sequências de eventos que hoje são inseridos

de forma manual pelo responsável pela manutenção e calibração dos medidores. Para a implementação de um processo de *burn-in* automatizado pelo dispositivo desenvolvido nesse trabalho, seria necessário a implementação de uma etapa de potência, na qual seriam colocados circuitos capazes de amplificar o sinal de saída gerado no gerador de sinais para amplitudes correspondentes as da rede elétrica. Além disso, para que o dispositivo seja capaz de realizar a calibração dos medidores de energia seria necessário a introdução de uma malha de realimentação, onde o dispositivo captura o sinal de saída e corrige o sinal que está sendo externado para que sempre esteja estável.

Um possível ponto de melhoria do projeto seria a geração de um arquivo executável compatível com Windows, hoje a interface funciona única e exclusivamente no sistema operacional Linux, e isso torna seu uso limitado a usuários de Linux.

REFERÊNCIAS BIBLIOGRÁFICAS

ALTIUM LIMITED. Altium Designer. Disponivel em: <https://www.altium.com/altium-designer/>. Acesso em: 22 Fevereiro 2023.

ALTIUM LIMITED. Part Placement Shortcuts in Altium Designer. Disponivel em: <https://resources.altium.com/p/part-placement-shortcuts-altium-designer>. Acesso em: 22 Fevereiro 2023.

ALVANO, J. W. **Embedded Systems:** Introduction to Arm(r) Cortex -M Microcontrollers: Introduction to Arm(r) Cortex(tm)-M Microcontrollers. [S.l.]: Createspace, 2012.

ANALOG-DEVICES. ADE7758. **Poly Phase Multifunction Energy Metering**, 2011. Disponivel em: <https://www.analog.com/media/cn/technical-documentation/data-sheets/ade7758.pdf>. Acesso em: 20 Setembro 2022.

ANALOG-DEVICES. ADE9000. **High Performande, Multiphase Energy and Power Quality Monitoring IC Data Sheet**, 2017. Disponivel em: <https://www.analog.com/media/en/technical-documentation/data-sheets/ade9000.pdf>.

ARM. Keil MDK. **Arm Developer Program**. Disponivel em: <https://developer.arm.com/Tools%20and%20Software/Keil%20MDK>. Acesso em: 18 Fevereiro 2023.

BORTOLUZZI, H. CHOQUE ELÉTRICO - BARRASHOPPINGSUL, Porto Alegre, 2009.

COPADATA. O que é HMI? **COPODATA**, 2022. Disponivel em: <https://www.copadata.com/pt/produtos/zenon-software-platform/visualizacao-controle/o-que-e-hmi-a-interface-homem-maquina-copa-data/>. Acesso em: 21 Setembro 2022.

CPPREFERENCE. Documentação da biblioteca math em C. **cppreference.com**. Disponivel em: <https://en.cppreference.com/w/c/numeric/math>. Acesso em: 18 Fevereiro 2023.

DAVIDSON, S.; SHRIVER, B. D. An Overview of Firmware Engineering. **Computer**, May 1978.

DIGIKEY. onsemi MBRS260T3. Disponivel em: <https://www.digikey.com/en/products/detail/onsemi/MBRS260T3/660718>. Acesso em: 18 Fevereiro 2023.

DIGIKEY. onsemi NCP1117LPST33T3G. **Digikey**. Disponivel em: <https://www.digikey.com/en/products/detail/onsemi/NCP1117LPST33T3G/2194024>. Acesso em: 18 Fevereiro 2023.

DIGIKEY. STMicroelectronics LM224D. Disponivel em: <https://www.digikey.com/en/products/detail/stmicroelectronics/LM224D/1038643>. Acesso em: 15 Fevereiro 2023.

ELECTRONICS TUTORIALS. Pi Filter Design. Disponivel em: <https://www.electronics-tutorials.ws/filter/filter_7.html>. Acesso em: 18 Fevereiro 2023.

EMBARCADOS. Embarcados - Comunicação SPI. **Embarcados**. Disponivel em: <https://embarcados.com.br/spi-parte-1/>. Acesso em: 20 Fevereiro 2023.

FREERTOS. ARM Cortex-M33 (ARMv8-M) Keil Simulator Demo. Disponivel em: <https://www.freertos.org/RTOS-Cortex-M33-Keil-Simulator.html>. Acesso em: 24 Fevereiro 2023.

FUNDACENTRO. **Engenharia de Segurança do Trabalho na Indústria da Construção**. São Paulo: FUNDACENTRO, 2011.

GOMES, K. D. C. DESENVOLVIMENTO DE UM DISPOSITIVO ELETRÔNICO DE BAIXO CUSTO PARA A MEDIÇÃO DE DISTORÇÃO HARMÔNICA CONFORME A NORMA IEC61000-4-7, 2019., 2019.

IDOETA, I. V.; CAPUANO, F. G. 8.6 - Memórias. In: CAPUANO, F. G. **Elementos de Eletrônica Digital - 40ª Edição**. [S.l.]: Érica, 2011. p. 401-416.

JETBRAINS. PyCharm. Disponivel em: <https://www.jetbrains.com/pycharm/>. Acesso em: 22 Fevereiro 2023.

JLCPCB. JLC - SHOPPING CART. Disponivel em: <https://cart.jlcpcb.com/shopcart/cart?_ga=2.9372336.460256463.1678212918-1795942695.1666706974>. Acesso em: 7 Março 2023.

JUNIOR, A. P. **Amplificadores Operacionais e Filtros Ativos**. 8ª. ed. [S.l.]: Bookman, 2017.

KERNIGHAN, B. ; RITCHIE,. **C Programming Language**. [S.l.]: Prentice Hall, 1988.

LCSC. LCSC - CART. Disponivel em: <https://www.lcsc.com/cart>. Acesso em: 07 Março 2023.

MICROCHIP. **Timer Peripherals**. Disponivel em: <https://www.microchip.com/en-us/products/microcontrollers-and-microprocessors/8-bit-mcus/core-independent-and-analog-peripherals/timer-peripheral>. Acesso em: 20 Fevereiro 2023.

MICROCHIP. Datasheet - MCP4922T. Disponivel em: <https://www.mouser.com/datasheet/2/268/21897a-70809.pdf>. Acesso em: 15 Fevereiro 2023.

MIZIDI, M. A.; NAIMI, S.; NAIMI, S. **The AVR microcontroller and embedded systems using Assembly and C**. [S.l.]: Pearson, 2011.

MOUSER. Disponivel em: <https://www.mouser.com/ProductDetail/Microchip-Technology-Atmel/MCP4922T-E-SL?qs=8U9qMIGhlkqc26Xt7UKk2w%3D%3D>. Acesso em: 15 Fevereiro 2023.

MOUSER ELECTRONICS. STMicroelectronics STM32F100RBT6B. **mouser**. Disponivel em: <https://www.mouser.com/ProductDetail/STMicroelectronics/STM32F100RBT6B?qs=eF7XX6ARXFHMlfWFz2g6gg%3D%3D&gclid=Cj0KCQiA3eGfBhCeARIsACpJNU_VOjmuEbazDeOdMurbaw3D10aMlQqth0y2HKpOP_fnDhsYL_oWxtcaAtAkEALw_wcB>. Acesso em: 15 Janeiro 2023.

MOUSER. Silicon Labs CP2103-GM. Disponivel em: <https://www.mouser.com/ProductDetail/Silicon-Labs/CP2103-GM?qs=yHCtFi5ROqIZGxXx3OFcgw%3D%3D>. Acesso em: 15 Fevereiro 2022.

MTE. MTE - PPS400.3. **MTE**, 2022. Disponivel em: <https://www.mte.ch/products/portable-test-equipment/power-sources-37/pps-4003-65>. Acesso em: 30 Setembro 2022.

NATIONAL INSTRUMENTS. myDAQ Specifications. Disponivel em: <https://www.ni.com/docs/en-US/bundle/mydaq-specs/page/specs.html>. Acesso em: 07 Março 2023.

NATIONAL INSTRUMENTS. NI-myDAQ. Disponivel em: <https://www.ni.com/pt-br/shop/hardware/products/mydaq-student-data-acquisition-device.html>. Acesso em: 07 Março 2023.

NEWARK. 781326 - University Kit, myDAQ, Data Acquisition Device, Hardware Only. Disponivel em: <https://www.newark.com/pt-BR/ni/781326-01/mydaq-university-kit/dp/14AJ5286>. Acesso em: 07 Março 2023.

ON SEMI. **On Semi Products**. Disponivel em: <https://www.onsemi.com/products/power-management/voltage-regulators/linear-voltage-regulators/ncp1117 >. Acesso em: 15 Fevereiro 2023.

ON SEMI. Datasheet do diodo MBRS260T3. Disponivel em: <https://www.onsemi.com/pdf/datasheet/mbrs260t3-d.pdf>. Acesso em: 18 Fevereiro 2023.

OPPENHEIM, A. V.; WILLSKY, A. S. **Sinais e Sistemas**. [S.l.]: Cap. 3: Representação de sinais periódicos em série de Fourier. Ed 2: p 104 – 107, 2010.

PEREIRA, F. **Microcontroladores PIC:** Programação em C. [S.l.]: Editora Érica, 2009.

PYTHON. About Python. **Python**. Disponivel em: <https://www.python.org/about/>. Acesso em: 22 Fevereiro 2023.

RITCHIE, D. M. The Development of the C Language. **bell-labs.com**. Disponivel em: <https://www.bell-labs.com/usr/dmr/www/chist.html>. Acesso em: 22 Fevereiro 2023.

SIEWERT. **Real-Time Embedded Components and Systems with Linux and RTOS**. [S.l.]: [s.n.], 2016.

SILICONLABS. Datasheet CP2103. Disponivel em: <https://www.silabs.com/documents/public/data-sheets/CP2103.pdf>. Acesso em: 15 Fevereiro 2023.

STMICROELECTRONICS. ST Community. ISSN https://community.st.com/s/question/0D50X0000ALuGf6SQF/stm32cubemx-versions-and-code-generation. Acesso em: 22 Fevereiro 2023.

STMICROELECTRONICS. STM32Cube initialization code generator. **ST**. Disponivel em: <https://www.st.com/en/development-tools/stm32cubemx.html>. Acesso em: 22 Fevereiro 2023.

STMICROELECTRONICS. STM32F100x4 STM32F100x6 STM32F100x8 STM32F100xB Datasheet. **Mouser**. Disponivel em: <https://www.mouser.com/datasheet/2/389/stm32f100cb-1851080.pdf>. Acesso em: 15 Janeiro 2023.

TEXAS INSTRUMENTS. **Datasheet LM224**. Disponivel em: <https://www.ti.com/lit/ds/symlink/lm124-n.pdf>. Acesso em: 15 Fevereiro 2023.

THE QT COMPANY. Introduction - PYQT5. Disponivel em: <https://doc.qt.io/qtforpython/contents.html>. Acesso em: 22 Fevereiro 2023.

THOMÉ, P.; BELINE, E. L. CHOQUE ELÉTRICO: CAUSAS, CONSEQUÊNCIAS E SEUS EFEITOS PARA O CORPO HUMANO. **XII EEPA**, 2018.

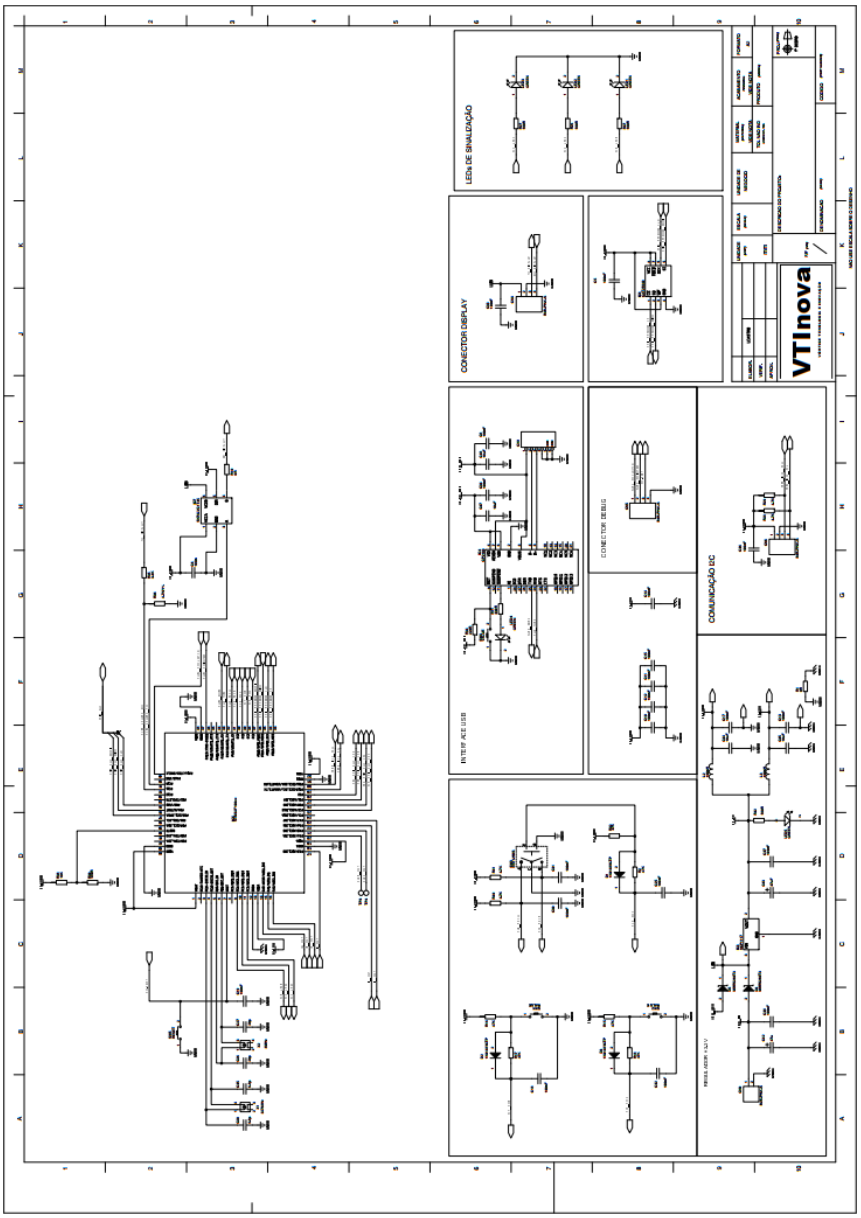
TOCCI, R. J.; WIDMER, N. S.; MOSS, G. L. Capítulo 11 - Interface Com o Mundo Analógico. In: TOCCI, R. J.; WIDMER, N. S.; MOSS, G. L. **Sistemas Digitais - Princípios e Aplicações**. [S.l.]: Pearson, 2017. p. 814 - 835.

VÓRTICE TECNOLOGIA E INOVAÇÃO. SGRede-QE. **vtinova**. Disponivel em: <http://www.vtinova.com.br/index.php/sgrede-qe>. Acesso em: 07 Março 2023.

VÓRTICE TECNOLOGIA E INOVAÇÃO. VTInova - Produtos - SGRede-BT. **vtinova.com.br**. Disponivel em: <http://www.vtinova.com.br/index.php/sgrede-qe?id=63>. Acesso em: 15 Setembro 2022.

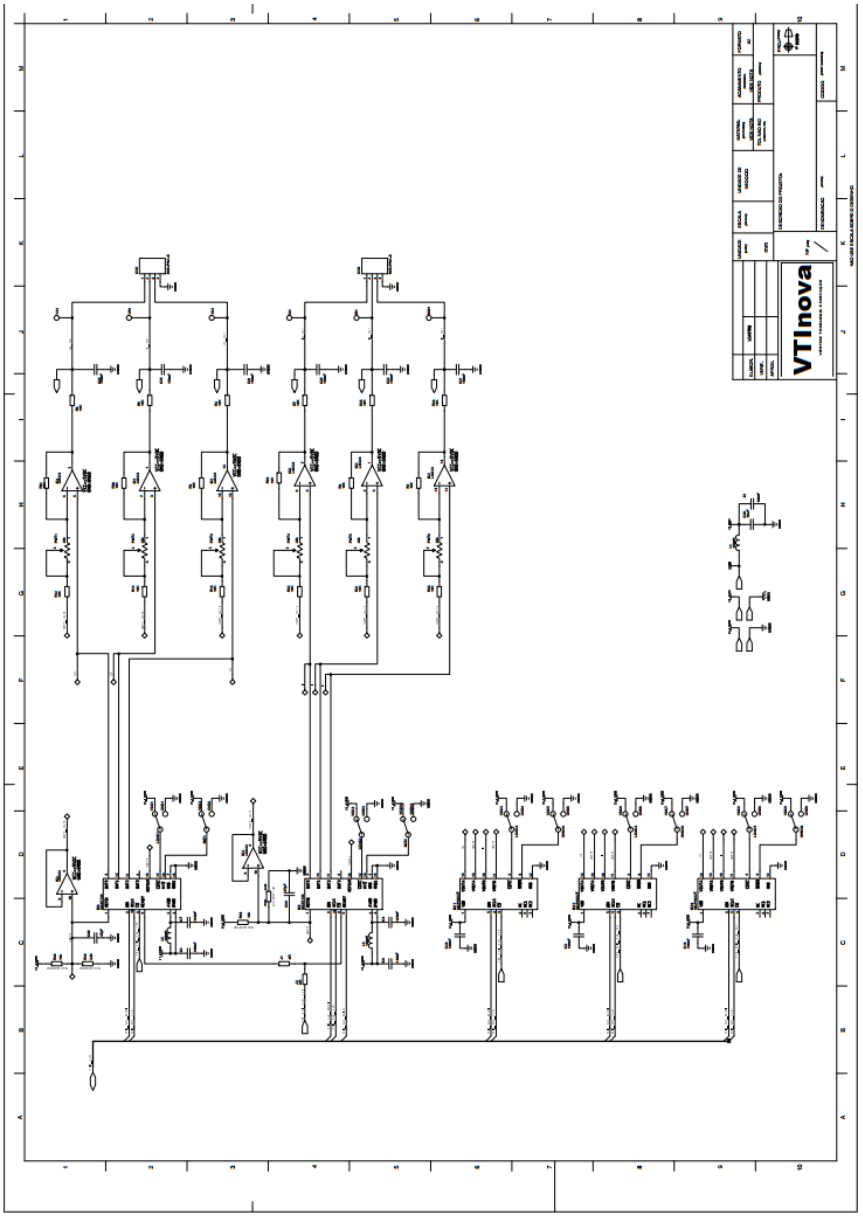
APÊNDICE A – DIAGRAMA ESQUEMÁTICO DO GERADOR DE SINAIS

Figura 40: Microcontrolador, alimentação e memória do dispositivo.



Fonte: (Elaborado pelo autor, 2023)

Figura 41: Conversores D/A e circuitos de ajuste do sinal de saída.



Fonte: (Elaborado pelo autor, 2023)

APÊNDICE B – ALGORITIMO PRINCIPAL DO MICROCONTROLADOR

```
/*
  Autor: Keven Soares
  Título: Fonte Trifásica de 6 canais
*/
/* Includes ------------------------------------------
-----------------------*/
#include "main.h"
#include "math.h"
#include <string.h>
/* Private includes ----------------------------------
-----------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private typedef -----------------------------------
-----------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------
-----------------------*/
/* USER CODE BEGIN PD */

/* USER CODE END PD */

/* Private macro -------------------------------------
-----------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------
-----------------------*/
ADC_HandleTypeDef hadc1;

DAC_HandleTypeDef hdac;

I2C_HandleTypeDef hi2c2;

RTC_HandleTypeDef hrtc;
RTC_TimeTypeDef sTime = {0};
RTC_TimeTypeDef gTime = {0};
RTC_DateTypeDef DateToUpdate = {0};
```

```
SPI_HandleTypeDef hspi1;
SPI_HandleTypeDef hspi2;

TIM_HandleTypeDef htim4;

UART_HandleTypeDef huart1;
UART_HandleTypeDef huart3;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------
-----------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART3_UART_Init(void);
static void MX_I2C2_Init(void);
static void MX_ADC1_Init(void);
static void MX_DAC_Init(void);
static void MX_SPI1_Init(void);
static void MX_USART1_UART_Init(void);
static void MX_SPI2_Init(void);
static void MX_TIM4_Init(void);
static void MX_RTC_Init(void);
/* USER CODE BEGIN PFP */
  unsigned short  outPut;
     char data;
  uint8_t bufferRx[264];
     uint8_t bufferTx[4] = {0xAA,0x00,0x00,0x00};

     //uint8_t bufferserial[256];
     char data2;
     char data3[256];
     char data4[256];
     char data5[256];
     char data6[256];
     char data7[256];
     char data8[256];

  char data9[256];
     char data10[256];
     char data11[256];
     char data12[256];
     char data13[256];
     char data14[256];

     int k =0;
     _Bool wflag = 0;
     uint8_t dadoOut[259];
```

```
     uint8_t Ano = 19;
     uint8_t Mes = 02;
     uint8_t Dia = 25;
     uint8_t Hora = 10;
     uint8_t Minuto = 31;
     uint8_t Segundo = 45;
     uint16_t endereco = 0x0003;
     char executar = 0;
     uint16_t tempo;
     uint16_t tempoAux;
     void maquina();
     void OndasPadrao();
     void transferWave();
     int userh = 0;
     int userm = 0;
     int users = 0;
     void genWave(uint8_t flag, int i);
     void writeMemo(uint8_t date, uint16_t adress, int
i);
     uint8_t readMemo(uint16_t adress,int i);
     void getFromMemo();
     void arrumatempo(int usertime);

/* USER CODE END PFP */

/* Private user code ---------------------------------
-------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration---------------------------------
-------------------------*/

  /* Reset of all peripherals, Initializes the Flash
interface and the Systick. */
  HAL_Init();
```

```
  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART3_UART_Init();
  MX_I2C2_Init();
  MX_ADC1_Init();
  MX_DAC_Init();
  MX_SPI1_Init();
  MX_USART1_UART_Init();
  MX_SPI2_Init();
  MX_TIM4_Init();
  MX_RTC_Init();
  /* USER CODE BEGIN 2 */
       HAL_UART_Receive_IT(&huart1,bufferRx,1);
      // HAL_UART_Transmit_IT(&huart1,bufferRx,260);
  /* USER CODE END 2 */
    OndasPadrao(0);
       HAL_TIM_Base_Start_IT(&htim4);
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */
          arrumatempo(tempo);

     HAL_RTC_GetTime(&hrtc,&gTime,RTC_FORMAT_BIN);

       if(wflag == 1)
       {
              //HAL_UART_AbortReceive_IT(&huart1);
        for(int j=0;j<=259;j++)
        {
                writeMemo(bufferRx[5+j],endereco+j,j);
                if(j == 259)
                {
                     endereco = endereco + 260;
                     wflag = 0;
                }
        }
            HAL_UART_Receive_IT(&huart1,bufferRx,1);
```

```
        }
           else if(executar == 1)
           {
                HAL_TIM_Base_Start_IT(&htim4);
                executar = 0;
           }
           else if((gTime.Hours*3600 + gTime.Minutes*60
+ gTime.Seconds) == (Hora * 3600 + Minuto*60 +
Segundo))
    {
                transferWave();
                getFromMemo();
                Hora = Hora + userh;
                Minuto = Minuto + userm;
                Segundo = Segundo + users;
                if(Segundo >= 60)
                {
                     Minuto++;
                  Segundo = Segundo - 60;
                }
                if(Minuto >= 60)
                {
                     Hora++;
                  Minuto = Minuto - 60;
                }
                tempo = tempoAux;
                arrumatempo(tempo);
                HAL_TIM_Base_Start_IT(&htim4);
           }
           else if((gTime.Hours >= userh + Hora) &&
(gTime.Minutes >= userm + Minuto) && (gTime.Seconds >=
users + Segundo))
    {
                OndasPadrao();
           }

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}


/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
```

```
  RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};

  /**Initializes the CPU, AHB and APB busses clocks
  */
  RCC_OscInitStruct.OscillatorType =
RCC_OSCILLATORTYPE_HSI|RCC_OSCILLATORTYPE_LSE;
  RCC_OscInitStruct.LSEState = RCC_LSE_ON;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue =
RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource =
RCC_PLLSOURCE_HSI_DIV2;
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL6;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  /**Initializes the CPU, AHB and APB busses clocks
  */
  RCC_ClkInitStruct.ClockType =
RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK

|RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource =
RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct,
FLASH_LATENCY_0) != HAL_OK)
  {
    Error_Handler();
  }
  PeriphClkInit.PeriphClockSelection =
RCC_PERIPHCLK_RTC|RCC_PERIPHCLK_ADC;
  PeriphClkInit.RTCClockSelection =
RCC_RTCCLKSOURCE_LSE;
  PeriphClkInit.AdcClockSelection = RCC_ADCPCLK2_DIV2;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) !=
HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief ADC1 Initialization Function
  * @param None
```

```
  * @retval None
  */
static void MX_ADC1_Init(void)
{

  /* USER CODE BEGIN ADC1_Init 0 */

  /* USER CODE END ADC1_Init 0 */

  ADC_ChannelConfTypeDef sConfig = {0};

  /* USER CODE BEGIN ADC1_Init 1 */

  /* USER CODE END ADC1_Init 1 */
  /**Common config
  */
  hadc1.Instance = ADC1;
  hadc1.Init.ScanConvMode = ADC_SCAN_DISABLE;
  hadc1.Init.ContinuousConvMode = DISABLE;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.NbrOfConversion = 1;
  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    Error_Handler();
  }
  /**Configure Regular Channel
  */
  sConfig.Channel = ADC_CHANNEL_0;
  sConfig.Rank = ADC_REGULAR_RANK_1;
  sConfig.SamplingTime = ADC_SAMPLETIME_1CYCLE_5;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) !=
HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC1_Init 2 */

  /* USER CODE END ADC1_Init 2 */

}

/**
  * @brief DAC Initialization Function
  * @param None
  * @retval None
  */
static void MX_DAC_Init(void)
{
```

```
  /* USER CODE BEGIN DAC_Init 0 */

  /* USER CODE END DAC_Init 0 */

  DAC_ChannelConfTypeDef sConfig = {0};

  /* USER CODE BEGIN DAC_Init 1 */

  /* USER CODE END DAC_Init 1 */
  /**DAC Initialization
  */
  hdac.Instance = DAC;
  if (HAL_DAC_Init(&hdac) != HAL_OK)
  {
    Error_Handler();
  }
  /**DAC channel OUT1 config
  */
  sConfig.DAC_Trigger = DAC_TRIGGER_NONE;
  sConfig.DAC_OutputBuffer = DAC_OUTPUTBUFFER_ENABLE;
  if (HAL_DAC_ConfigChannel(&hdac, &sConfig,
DAC_CHANNEL_1) != HAL_OK)
  {
    Error_Handler();
  }
  /**DAC channel OUT2 config
  */
  if (HAL_DAC_ConfigChannel(&hdac, &sConfig,
DAC_CHANNEL_2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN DAC_Init 2 */

  /* USER CODE END DAC_Init 2 */

}

/**
  * @brief I2C2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_I2C2_Init(void)
{

  /* USER CODE BEGIN I2C2_Init 0 */

  /* USER CODE END I2C2_Init 0 */
```

```
  /* USER CODE BEGIN I2C2_Init 1 */

  /* USER CODE END I2C2_Init 1 */
  hi2c2.Instance = I2C2;
  hi2c2.Init.ClockSpeed = 100000;
  hi2c2.Init.DutyCycle = I2C_DUTYCYCLE_2;
  hi2c2.Init.OwnAddress1 = 0;
  hi2c2.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c2.Init.DualAddressMode =
I2C_DUALADDRESS_DISABLE;
  hi2c2.Init.OwnAddress2 = 0;
  hi2c2.Init.GeneralCallMode =
I2C_GENERALCALL_DISABLE;
  hi2c2.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN I2C2_Init 2 */

  /* USER CODE END I2C2_Init 2 */

}

/**
  * @brief RTC Initialization Function
  * @param None
  * @retval None
  */
static void MX_RTC_Init(void)
{

  /* USER CODE BEGIN RTC_Init 0 */

  /* USER CODE END RTC_Init 0 */

  RTC_TimeTypeDef sTime = {0};
  RTC_DateTypeDef DateToUpdate = {0};

  /* USER CODE BEGIN RTC_Init 1 */

  /* USER CODE END RTC_Init 1 */
  /**Initialize RTC Only
  */
  hrtc.Instance = RTC;
  hrtc.Init.AsynchPrediv = RTC_AUTO_1_SECOND;
  hrtc.Init.OutPut = RTC_OUTPUTSOURCE_ALARM;
  if (HAL_RTC_Init(&hrtc) != HAL_OK)
  {
```

```
    Error_Handler();
  }

  /* USER CODE BEGIN Check_RTC_BKUP */

  /* USER CODE END Check_RTC_BKUP */

  /**Initialize RTC and set the Time and Date
  */
  sTime.Hours = 0x00;
  sTime.Minutes = 0x00;
  sTime.Seconds = 0x00;

  if (HAL_RTC_SetTime(&hrtc, &sTime, RTC_FORMAT_BCD)
!= HAL_OK)
  {
    Error_Handler();
  }
  DateToUpdate.WeekDay = RTC_WEEKDAY_MONDAY;
  DateToUpdate.Month = RTC_MONTH_JANUARY;
  DateToUpdate.Date = 0x1;
  DateToUpdate.Year = 0x0;

  if (HAL_RTC_SetDate(&hrtc, &DateToUpdate,
RTC_FORMAT_BCD) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN RTC_Init 2 */

  /* USER CODE END RTC_Init 2 */

}

/**
  * @brief SPI1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_SPI1_Init(void)
{
  /* Inicialização da Comunicação SPI*/
  hspi1.Instance = SPI1;
  hspi1.Init.Mode = SPI_MODE_MASTER;
  hspi1.Init.Direction = SPI_DIRECTION_2LINES;
  hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
  hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
  hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
  hspi1.Init.NSS = SPI_NSS_SOFT;
```

```
  hspi1.Init.BaudRatePrescaler =
SPI_BAUDRATEPRESCALER_2;
  hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
  hspi1.Init.CRCCalculation =
SPI_CRCCALCULATION_DISABLE;
  hspi1.Init.CRCPolynomial = 10;
  if (HAL_SPI_Init(&hspi1) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief SPI2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_SPI2_Init(void)
{

  /* USER CODE BEGIN SPI2_Init 0 */

  /* USER CODE END SPI2_Init 0 */

  /* USER CODE BEGIN SPI2_Init 1 */

  /* USER CODE END SPI2_Init 1 */
  /* SPI2 parameter configuration*/
  hspi2.Instance = SPI2;
  hspi2.Init.Mode = SPI_MODE_MASTER;
  hspi2.Init.Direction = SPI_DIRECTION_2LINES;
  hspi2.Init.DataSize = SPI_DATASIZE_8BIT;
  hspi2.Init.CLKPolarity = SPI_POLARITY_LOW;
  hspi2.Init.CLKPhase = SPI_PHASE_1EDGE;
  hspi2.Init.NSS = SPI_NSS_SOFT;
  hspi2.Init.BaudRatePrescaler =
SPI_BAUDRATEPRESCALER_2;
  hspi2.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi2.Init.TIMode = SPI_TIMODE_DISABLE;
  hspi2.Init.CRCCalculation =
SPI_CRCCALCULATION_DISABLE;
  hspi2.Init.CRCPolynomial = 10;
  if (HAL_SPI_Init(&hspi2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN SPI2_Init 2 */

  /* USER CODE END SPI2_Init 2 */
```

```
}

/**
  * @brief TIM4 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM4_Init(void)
{

  /* USER CODE BEGIN TIM4_Init 0 */

  /* USER CODE END TIM4_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};

  /* USER CODE BEGIN TIM4_Init 1 */

  /* USER CODE END TIM4_Init 1 */
  htim4.Instance = TIM4;
  htim4.Init.Prescaler = 0;
  htim4.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim4.Init.Period = 3125;
  htim4.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim4.Init.AutoReloadPreload =
TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim4) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource =
TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim4,
&sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode =
TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim4,
&sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM4_Init 2 */

  /* USER CODE END TIM4_Init 2 */
```

```
}

/**
  * @brief USART1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART1_UART_Init(void)
{

  /* USER CODE BEGIN USART1_Init 0 */

  /* USER CODE END USART1_Init 0 */

  /* USER CODE BEGIN USART1_Init 1 */

  /* USER CODE END USART1_Init 1 */
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 115200;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;
  huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart1.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART1_Init 2 */

  /* USER CODE END USART1_Init 2 */

}

/**
  * @brief USART3 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART3_UART_Init(void)
{

  /* USER CODE BEGIN USART3_Init 0 */

  /* USER CODE END USART3_Init 0 */

  /* USER CODE BEGIN USART3_Init 1 */
```

```
  /* USER CODE END USART3_Init 1 */
  huart3.Instance = USART3;
  huart3.Init.BaudRate = 115200;
  huart3.Init.WordLength = UART_WORDLENGTH_8B;
  huart3.Init.StopBits = UART_STOPBITS_1;
  huart3.Init.Parity = UART_PARITY_NONE;
  huart3.Init.Mode = UART_MODE_TX_RX;
  huart3.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart3.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart3) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART3_Init 2 */

  /* USER CODE END USART3_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOC,
LED1_Pin|LED2_Pin|LED3_Pin|CS3_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB,
CS2_Pin|CS1_Pin|CS0_Pin|CS_EEPROM_Pin,
GPIO_PIN_RESET);

  /*Configure GPIO pins : PC13 PC0 PC4 PC12 */
  GPIO_InitStruct.Pin =
GPIO_PIN_13|GPIO_PIN_0|GPIO_PIN_4|GPIO_PIN_12;
  GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
```

```
  /*Configure GPIO pins : LED1_Pin LED2_Pin LED3_Pin
CS3_Pin */
  GPIO_InitStruct.Pin =
LED1_Pin|LED2_Pin|LED3_Pin|CS3_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  /*Configure GPIO pins : CS2_Pin CS1_Pin CS0_Pin
CS_EEPROM_Pin */
  GPIO_InitStruct.Pin =
CS2_Pin|CS1_Pin|CS0_Pin|CS_EEPROM_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

  /*Configure GPIO pins : SW2_Pin SW1_Pin SW3_Pin
ENCB_Pin */
  GPIO_InitStruct.Pin =
SW2_Pin|SW1_Pin|SW3_Pin|ENCB_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  /*Configure GPIO pin : ENCA_Pin */
  GPIO_InitStruct.Pin = ENCA_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(ENCA_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : PA11 PA12 PA15 */
  GPIO_InitStruct.Pin =
GPIO_PIN_11|GPIO_PIN_12|GPIO_PIN_15;
  GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pin : PD2 */
  GPIO_InitStruct.Pin = GPIO_PIN_2;
  GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
  HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);

  /*Configure GPIO pins : PB6 PB7 PB8 PB9 */
  GPIO_InitStruct.Pin =
GPIO_PIN_6|GPIO_PIN_7|GPIO_PIN_8|GPIO_PIN_9;
  GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

}
```

```
/* USER CODE BEGIN 4 */
void transferWave()
{
     for(int i = 0; i <= 255; i++)
     {
          data3[i] = data9[i];
          data4[i] = data10[i];
          data5[i] = data11[i];
          data6[i] = data12[i];
          data7[i] = data13[i];
          data8[i] = data14[i];
     }
}
void arrumatempo(int usertime)
{
      userh = usertime/3600;
      userm = (usertime%3600)/60;
      users = (usertime%3600)%60;
}
void OndasPadrao()
{
     for(int i = 0; i <= 127; i++)
     {
       outPut = (int) (2048 +
cos(i*(6.28/128))*2048*0.1);
          data = (outPut >> 8);
          data = 0x0F & data;
          data = 0x30 | data;
          data2 = (outPut);
       data3[2*i] = data;
       data3[2*i+1] = data2;
     }
     for(int i = 0; i <= 127; i++)
     {
       outPut = (int) (2048 +
cos((3.14/2)+(i*(6.28/128)))*2048*0.1);
          data = (outPut >> 8);
          data = 0x0F & data;
          data = 0xB0 | data;
          data2 = (outPut);
       data4[2*i] = data;
       data4[2*i+1] = data2;
     }
          for(int i = 0; i <= 127; i++)
     {
       outPut = (int) (2048 +
cos((6.28/3)+i*(6.28/128))*2048*0.1);
          data = (outPut >> 8);
          data = 0x0F & data;
```

```
          data = 0x30 | data;
          data2 = (outPut);
       data5[2*i] = data;
       data5[2*i+1] = data2;

     }
          for(int i = 0; i <= 127; i++)
     {
       outPut = (int) (2048 +
cos((3.14/2)+(6.28/3)+i*(6.28/128))*2048*0.1);
          data = (outPut >> 8);
          data = 0x0F & data;
          data = 0xB0 | data;
          data2 = (outPut);
       data6[2*i] = data;
       data6[2*i+1] = data2;
     }
          for(int i = 0; i <= 127; i++)
     {
       outPut = (int) (2048 + cos((-
6.28/3)+i*(6.28/128))*2048*0.1);
          data = (outPut >> 8);
          data = 0x0F & data;
          data = 0x30 | data;
          data2 = (outPut);
       data7[2*i] = data;
       data7[2*i+1] = data2;
     }
          for(int i = 0; i <= 127; i++)
     {
       outPut = (int) (2048 + cos((-
6.28/3)+(3.14/2)+i*(6.28/128))*2048*0.1);
          data = (outPut >> 8);
          data = 0x0F & data;
          data = 0xB0 | data;
          data2 = (outPut);
       data8[2*i] = data;
       data8[2*i+1] = data2;
     }
}
void genWave(uint8_t flag, int i)
{
     switch(flag)
     {    case 0x01 ://VA

               data = (outPut >> 8);
               data = 0x0F & data;
               data = 0x30 | data;
               data2 = (outPut);
               data9[2*i] = data;
```

```
            data9[2*i+1] = data2;
        break;

        case 0x04 ://IA

            data = (outPut >> 8);
            data = 0x0F & data;
            data = 0xB0 | data;
            data2 = (outPut);
            data10[2*i] = data;
            data10[2*i+1] = data2;

        break;

        case 0x02 ://VB

            data = (outPut >> 8);
            data = 0x0F & data;
            data = 0x30 | data;
            data2 = (outPut);
            data11[2*i] = data;
            data11[2*i+1] = data2;

        break;

        case 0x05 ://IB

            data = (outPut >> 8);
            data = 0x0F & data;
            data = 0xB0 | data;
            data2 = (outPut);
            data12[2*i] = data;
            data12[2*i+1] = data2;

        break;

        case 0x03 ://VC

            data = (outPut >> 8);
            data = 0x0F & data;
            data = 0x30 | data;
            data2 = (outPut);
            data13[2*i] = data;
            data13[2*i+1] = data2;

        break;

        case 0x06 ://IC

            data = (outPut >> 8);
```

```
                data = 0x0F & data;
                data = 0xB0 | data;
                data2 = (outPut);
                data14[2*i] = data;
                data14[2*i+1] = data2;

          break;

          default:

                data = (outPut >> 8);
                data = 0x0F & data;
                data = 0x30 | data;
                data2 = (outPut);
                data9[2*i] = data;
                data9[2*i+1] = data2;

          break;
     }
}
void HAL_UART_RxCpltCallback(UART_HandleTypeDef
*huart)
{

     //HAL_Delay(50);
     maquina();



//HAL_UART_Transmit_IT(&huart1,bufferconfirm,1);
}
void HAL_UART_TxCpltCallback(UART_HandleTypeDef
*huart)
{
   __NOP();

}
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef
*htim)
{
     if(htim -> Instance == TIM4)
     {

                GPIOB->BSRR = (uint32_t)CS1_Pin <<
16U;//desliga o cs1
                HAL_SPI_Transmit(&hspi1, &data3[k], 2,
1);
             GPIOB->BSRR = CS1_Pin;//liga o cs1
```

```
                GPIOB->BSRR = (uint32_t)CS1_Pin <<
16U;//desliga o cs1
                HAL_SPI_Transmit(&hspi1, &data4[k], 2,
1);
            GPIOB->BSRR = CS1_Pin;//liga o cs1

                GPIOB->BSRR = (uint32_t)CS2_Pin <<
16U;//desliga o cs2
                HAL_SPI_Transmit(&hspi1, &data5[k], 2,
1);
            GPIOB->BSRR = CS2_Pin;//liga o cs1

                GPIOB->BSRR = (uint32_t)CS2_Pin <<
16U;//desliga o cs2
                HAL_SPI_Transmit(&hspi1, &data6[k], 2,
1);
            GPIOB->BSRR = CS2_Pin;//liga o cs1

                GPIOC->BSRR = (uint32_t)CS3_Pin <<
16U;//desliga o cs1
                HAL_SPI_Transmit(&hspi1, &data7[k], 2,
1);
            GPIOC->BSRR = CS3_Pin;//liga o cs1

                GPIOC->BSRR = (uint32_t)CS3_Pin <<
16U;//desliga o cs1
                HAL_SPI_Transmit(&hspi1, &data8[k], 2,
1);
            GPIOC->BSRR = CS3_Pin;//liga o cs1

            k+=2;
            if(k>=256)
               k=0;
     }
}
void maquina()
{
     HAL_TIM_Base_Stop_IT(&htim4);
     if(bufferRx[0] == 0xAA)
     {

      HAL_UART_Receive(&huart1,&bufferRx[1],3,50);
      uint16_t len = bufferRx[2] + bufferRx[3]*256;
      HAL_UART_Receive(&huart1,&bufferRx[4],len,50);
      bufferTx[1] = bufferRx[1] + 1;

      switch(bufferRx[1])
      {
          case 0x00://Atualizar Forma de Onda
           tempo = bufferRx[6] + bufferRx[7]*256;
```

```
            for(int i = 0; i <= 127; i++)
            {
              outPut = bufferRx[8 + 2*i] + bufferRx[9 +
2*i]*256 ;
              if (i == 127)
                wflag = 1;
            }
            HAL_UART_Transmit(&huart1,bufferTx,4,1);
            break;

           case 0x02://Atualizar Relógio
            HAL_UART_Transmit(&huart1,bufferTx,4,1);
            DateToUpdate.Month = bufferRx[5] ;
            DateToUpdate.Date = bufferRx[6];
            DateToUpdate.Year = bufferRx[4];
            sTime.Hours = bufferRx[7];
            sTime.Minutes = bufferRx[8];
            sTime.Seconds = bufferRx[9];
            HAL_RTC_SetTime(&hrtc, &sTime,
RTC_FORMAT_BIN);
            HAL_RTC_SetDate(&hrtc, &DateToUpdate,
RTC_FORMAT_BIN);

            break;

            case 0x04://Execução Imediata

              HAL_UART_Transmit(&huart1,bufferTx,4,1);
             Ano = bufferRx[4];
             Mes = bufferRx[5];
             Dia = bufferRx[6];
             Hora = bufferRx[7];
             Minuto = bufferRx[8];
             Segundo = bufferRx[9];
             executar = bufferRx[10];

            break;
        }
      }
      else
      {

      }
      bufferRx[0] = 0;
      HAL_UART_Receive_IT(&huart1,bufferRx,1);
}
void writeMemo(uint8_t date, uint16_t adress, int i)
{
        uint8_t bufferOut[4];
```

```
        uint8_t dado;

           dado = 0x00 & dado;
           dado = 0x06 | dado; //EnableWrite

        bufferOut[0] = dado;
        bufferOut[1] = adress >> 8;
           bufferOut[2] = adress;

        GPIOB->BSRR = (uint32_t)CS_EEPROM_Pin <<
16U;//desliga o cs1
           HAL_SPI_Transmit(&hspi2,bufferOut, 3, 1);
           GPIOB->BSRR = CS_EEPROM_Pin;//liga o cs1

        dado = 0x00 & dado;
           dado = 0x02 | dado;  //comando para escrita
        bufferOut[0] = dado;

        bufferOut[3] = date;
        GPIOB->BSRR = (uint32_t)CS_EEPROM_Pin <<
16U;//desliga o cs1
           HAL_SPI_Transmit(&hspi2,bufferOut, 4, 1);
           GPIOB->BSRR = CS_EEPROM_Pin;//liga o cs1
           dadoOut[i] = bufferOut[3];

}
void getFromMemo()
{
     uint16_t end = 0x0003;
     uint8_t bufferaux[259];
     for(int j = 0; j <= 5; j++)
  {
          for(int i = 0; i<= 259; i++)
          {
               bufferaux[i] = readMemo(end+i,i);
               if(i == 259)
                    end = end + 260;
          }
          for(int i = 0; i <= 127; i++)
          {
               outPut = bufferaux[3 + 2*i] +
bufferaux[4 + 2*i]*256;
               genWave((int)bufferaux[0],i);
               tempoAux = bufferaux[1] +
bufferaux[2]*256;
          }
     }
}
uint8_t readMemo(uint16_t adress,int i)
{
```

```
        uint8_t bufferOut[4];
        uint8_t dado;
     uint8_t resposta[2];
        dado = 0x00 & dado;
           dado = 0x03 | dado;  //comando para leitura
        bufferOut[0] = dado;
           bufferOut[1] = adress >> 8;
           bufferOut[2] = adress;
        GPIOB->BSRR = (uint32_t)CS_EEPROM_Pin <<
16U;//desliga o cs1
           HAL_SPI_Transmit(&hspi2,bufferOut, 3, 1);
        HAL_SPI_Receive(&hspi2,&bufferOut[3],1,1);
           GPIOB->BSRR = CS_EEPROM_Pin;//liga o cs1
        resposta[0] = bufferOut[3];
        dadoOut[i] = resposta[0];
        return dadoOut[i];
}
/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error
occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the
HAL error return state */
  HAL_GPIO_TogglePin(GPIOC,LED1_Pin);
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and
the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source
number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the
file name and line number,
     tex: printf("Wrong parameters value: file %s on
line %d\r\n", file, line) */
```

```
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */

/************************ (C) COPYRIGHT Keven Soares
*****END OF FILE****/
```

APÊNDICE C – ALGORITIMO PRINCIPAL DO SOFTWARE

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
__author__ = 'keven'
'''
import commands
import notify2
import signal
import gi
'''
from fase import MicroServiceBase
from waveform import WaveForm
import math
from matplotlib.figure import Figure
from matplotlib.backends.backend_gtk3agg import FigureCanvas
from matplotlib.backends.backend_gtk3 import (
    NavigationToolbar2GTK3 as NavigationToolbar)
import json
from stm import STM
import time
import threading

pi = 3.14159265359


class FonteTrifasica(MicroServiceBase):
    def __init__(self):
        super(FonteTrifasica, self).__init__(self,
sender_endpoint='ipc:///tmp/sender',

receiver_endpoint='ipc:///tmp/receiver')
        self.VA = [WaveForm()]
        self.VB = [WaveForm()]
        self.VC = [WaveForm()]
        self.IA = [WaveForm()]
        self.IB = [WaveForm()]
        self.IC = [WaveForm()]
        self.inicio_hora = 0
        self.inicio_minuto = 0
        self.inicio_segundo = 0
        self.TIME = [0]
        self.VMAX = 200
        self.IMAX = 50
        self.HARM = ['HARM_1', 'HARM_2', 'HARM_3', 'HARM_4', 'HARM_5',
'HARM_6', 'HARM_7', 'HARM_8', 'HARM_9',
                     'HARM_10',
                     'HARM_11', 'HARM_12', 'HARM_13', 'HARM_14',
'HARM_15', 'HARM_16', 'HARM_17', 'HARM_18', 'HARM_19',
                     'HARM_20',
                     'HARM_21', 'HARM_22', 'HARM_23', 'HARM_24',
'HARM_25', 'HARM_26', 'HARM_27', 'HARM_28', 'HARM_29',
                     'HARM_30',
                     'HARM_31', 'HARM_32', 'HARM_33', 'HARM_34',
'HARM_35', 'HARM_36', 'HARM_37', 'HARM_38', 'HARM_39',
                     'HARM_40']
        self.stm = STM()
```

```
    def on_broadcast(self, service, data):
        pass

    def on_connect(self):
        pass

    def on_new_service(self, service, actions):
        pass

    def create_json(self):
        self.data_to_send = []
        for i in range(len(self.VA)):
            new_samples = {'A' + str(i): {'VA': [0] * 128,
                                          'VB': [0] * 128,
                                          'VC': [0] * 128,
                                          'IA': [0] * 128,
                                          'IB': [0] * 128,
                                          'IC': [0] * 128,
                                          'TIME': self.TIME[i]
                                          }
                           }
            for j in range(128):
                new_samples['A' + str(i)]['VA'][j] =
int(self.VA[i].vetorWaveForm[j] * 2047 / self.VMAX) + 2047
                new_samples['A' + str(i)]['VB'][j] =
int(self.VB[i].vetorWaveForm[j] * 2047 / self.VMAX) + 2047
                new_samples['A' + str(i)]['VC'][j] =
int(self.VC[i].vetorWaveForm[j] * 2047 / self.VMAX) + 2047
                new_samples['A' + str(i)]['IA'][j] =
int(self.IA[i].vetorWaveForm[j] * 2047 / self.IMAX) + 2047
                new_samples['A' + str(i)]['IB'][j] =
int(self.IB[i].vetorWaveForm[j] * 2047 / self.IMAX) + 2047
                new_samples['A' + str(i)]['IC'][j] =
int(self.IC[i].vetorWaveForm[j] * 2047 / self.IMAX) + 2047
            self.data_to_send.append(json.dumps(new_samples))

    def get_vetor_waveform(self, wf, i, j):
        vetor = [0xAA, 0x00]
        vetor.append(i + 1)
        vetor.append(j + 1)
        vetor.append(self.TIME[i])
        if j <= 2:
            max = self.VMAX
        else:
            max = self.IMAX
        for sample in wf[i].vetorWaveForm:
            vetor.append(int(sample * 2047 / (max*3) + 2047))
        # print vetor
        return vetor

    def send_protocol(self):
        data = []
        for i in range(len(self.VA)):
            data.append(self.get_vetor_waveform(self.VA, i, 0))
            data.append(self.get_vetor_waveform(self.VB, i, 1))
            data.append(self.get_vetor_waveform(self.VC, i, 2))
            data.append(self.get_vetor_waveform(self.IA, i, 3))
            data.append(self.get_vetor_waveform(self.IB, i, 4))
            data.append(self.get_vetor_waveform(self.IC, i, 5))
        if self.stm.conectado:
            self.stm.send_protocol(data)
```

```
            self.stm.send_protocol([[0xAA, 0x04, 2019, 2, 25,
self.inicio_hora, self.inicio_minuto, self.inicio_segundo, 0,
len(self.VA)]])

    def send_one_protocol(self, i):
        data = []
        data.append(self.get_vetor_waveform(self.VA, i, 0))
        data.append(self.get_vetor_waveform(self.VB, i, 1))
        data.append(self.get_vetor_waveform(self.VC, i, 2))
        data.append(self.get_vetor_waveform(self.IA, i, 3))
        data.append(self.get_vetor_waveform(self.IB, i, 4))
        data.append(self.get_vetor_waveform(self.IC, i, 5))
        if self.stm.conectado:
            self.stm.send_protocol(data)

    def enviar_json(self):
        if self.stm.conectado:
            self.stm.send_json(self.data_to_send)

    def plot_graficos(self, pos, fft):
        graficos = []
        graficos.append(self.VA[pos].plotWaveFormFases(self.VB[pos],
self.VC[pos], "tensao"))
        graficos.append(self.IA[pos].plotWaveFormFases(self.IB[pos],
self.IC[pos], "corrente"))
        graficos.append(self.VA[pos].plotFasores(self.VB[pos],
self.VC[pos], self.IA[pos], self.IB[pos], self.IC[pos]))
        if fft:
            graficos.append(self.VA[pos].plotFFT('VA_H'))
            graficos.append(self.VB[pos].plotFFT('VB_H'))
            graficos.append(self.VC[pos].plotFFT('VC_H'))
            graficos.append(self.IA[pos].plotFFT('IA_H'))
            graficos.append(self.IB[pos].plotFFT('IB_H'))
            graficos.append(self.IC[pos].plotFFT('IC_H'))
        return graficos

    def salvar_graficos(self, data):
        pos = data['ID'] - 1
        self.VMAX = data['VMAX']
        self.IMAX = data['IMAX']
        graficos = []
        if len(self.VA) >= pos + 1:
            self.VA[pos].amp = data['VA']
            self.IA[pos].amp = data['IA']
            self.VA[pos].angulo = 0.0
            self.IA[pos].angulo = math.acos(data['FPA'])
            self.IA[pos].fp = data['FPA']
            self.VB[pos].amp = data['VB']
            self.IB[pos].amp = data['IB']
            self.VB[pos].angulo = 2 * pi / 3
            self.IB[pos].angulo = 2 * pi / 3 + math.acos(data['FPB'])
            self.IB[pos].fp = data['FPB']
            self.VC[pos].amp = data['VC']
            self.IC[pos].amp = data['IC']
            self.VC[pos].angulo = -2 * pi / 3
            self.IC[pos].angulo = -2 * pi / 3 + math.acos(data['FPC'])
            self.IC[pos].fp = data['FPC']
            self.VA[pos].calcWaveForm()
            self.VB[pos].calcWaveForm()
            self.VC[pos].calcWaveForm()
            self.IA[pos].calcWaveForm()
```

```
            self.IB[pos].calcWaveForm()
            self.IC[pos].calcWaveForm()
            self.TIME[pos] = data['TIME']
            graficos = self.plot_graficos(pos, data['FFT'])

        else:
            self.VA.append(WaveForm())
            self.VB.append(WaveForm())
            self.VC.append(WaveForm())
            self.IA.append(WaveForm())
            self.IB.append(WaveForm())
            self.IC.append(WaveForm())
            self.TIME.append(0)
            graficos = self.salvar_graficos(data)
        return graficos

    def get_graficos(self, pos):
        self.new_data = self.plot_graficos(pos, True)
        data = {'VA': self.VA[pos].amp, 'VB': self.VB[pos].amp, 'VC':
self.VC[pos].amp, 'IA': self.IA[pos].amp,
                'IB': self.IB[pos].amp, 'IC': self.IC[pos].amp, 'FPA':
self.IA[pos].fp, 'FPB': self.IC[pos].fp,
                'FPC': self.IC[pos].fp, 'TIME': self.TIME[pos], 'FFT':
False}
        return data

    def get_param(self, param):
        if param == 'VA_H':
            return self.VA
        elif param == 'VB_H':
            return self.VB
        elif param == 'VC_H':
            return self.VC
        elif param == 'IA_H':
            return self.IA
        elif param == 'IB_H':
            return self.IB
        elif param == 'IC_H':
            return self.IC

    def get_fft(self, pos, param):
        self.new_data = self.plot_graficos(pos, True)
        sinal = self.get_param(param)
        data = {self.HARM[i]: sinal[pos].harmonica[i] for i in
range(len(self.HARM))}
        data['FFT'] = True
        return data

    def salvar_harmonicos(self, data):
        pos = data['ID'] - 1
        if data['PARAM'] == 'VA_H':
            self.VA[pos].setHarm(data)
        elif data['PARAM'] == 'VB_H':
            self.VB[pos].setHarm(data)
        elif data['PARAM'] == 'VC_H':
            self.VC[pos].setHarm(data)
        elif data['PARAM'] == 'IA_H':
            self.IA[pos].setHarm(data)
        elif data['PARAM'] == 'IB_H':
            self.IB[pos].setHarm(data)
        elif data['PARAM'] == 'IC_H':
```

```
            self.IC[pos].setHarm(data)
        # self.get_fft(pos,data['PARAM'])
        return self.plot_graficos(pos, True)

    def zerar_harmonicos(self, pos):
        self.VA[pos].zerarHarm()
        self.VB[pos].zerarHarm()
        self.VC[pos].zerarHarm()
        self.IA[pos].zerarHarm()
        self.IB[pos].zerarHarm()
        self.IC[pos].zerarHarm()

    def get_new_data(self):
        return self.new_data

    def salvar_inicio(self, data):
        self.inicio_hora = int(data[0:2])
        self.inicio_minuto = int(data[3:5])
        self.inicio_segundo = int(data[6:8])



"""#########################################################################
########################################"""
    """################################################## ACTIONS
##################################################"""

"""#########################################################################
########################################"""

    @MicroServiceBase.action
    def gerar_graficos(self, service, data):
        self.new_data = self.salvar_graficos(data)
        self.request_action('update_grafico', {})

    @MicroServiceBase.action
    def receber_graficos(self, service, data):
        if len(self.VA) >= data:
            self.request_action('update_grafico', self.get_graficos(data
- 1))
        else:
            self.request_action('clear_grafico', {})

    @MicroServiceBase.action
    def receber_harmonicos(self, service, data):
        if len(self.VA) >= data['ID']:
            self.request_action('update_grafico', self.get_fft(data['ID']
- 1, data['PARAM']))
        else:
            self.request_action('clear_grafico', {})

    @MicroServiceBase.action
    def gerar_harmonicos(self, service, data):
        if len(self.VA) >= data['ID']:
            self.new_data = self.salvar_harmonicos(data)
            self.request_action('update_grafico', {})

    @MicroServiceBase.action
    def clear_harmonicos(self, service, data):
        self.zerar_harmonicos(data)
```

```
    @MicroServiceBase.action
    def enviar_dados(self, service, data):
        t = threading.Thread(target = self.send_protocol, args=())
        t.start()

    @MicroServiceBase.action
    def sincronizar(self, service, data):
        if self.stm.conectado:
            self.stm.send_protocol([[0xAA, 0x02]])

    @MicroServiceBase.action
    def update_inicio(self, service, data):
       self.salvar_inicio(data['INICIO'])

    @MicroServiceBase.action
    def conectar_stm(self, service, data):
        if not data['CONNECT']:
            self.stm.desconectar()
        else:
            if data['DEV']:
                self.stm.set_device(data['DEV'])
                self.stm.conectar()
        self.request_action('receive_status', self.stm.conectado)

    @MicroServiceBase.action
    def start_process(self, service, data):
        data['H_INIC_PROC'] =
self.inicio_hora*60*60+self.inicio_minuto*60+self.inicio_segundo
        data['H_FIM_PROC']  = data['H_INIC_PROC']
        for t in self.TIME:
            data['H_FIM_PROC'] += t
        self.request_action('inicial_info', data)

    @MicroServiceBase.action
    def wf_info(self, service, data):
        if data['i'] == -1:
            data['H_WF'] = 0
            data['H_INIC_WF'] = 0
            data['H_FIM_WF'] =
self.inicio_hora*60*60+self.inicio_minuto*60+self.inicio_segundo
        elif data['i'] < len(self.TIME):
            # self.send_one_protocol(data['i'])
            data['H_WF'] = self.TIME[data['i']]
            data['H_INIC_WF'] = self.inicio_hora * 60 * 60 +
self.inicio_minuto * 60 + self.inicio_segundo - self.TIME[data['i']]
            data['H_FIM_WF'] = self.inicio_hora * 60 * 60 +
self.inicio_minuto * 60 + self.inicio_segundo
            for i in range(data['i'] + 1):
                data['H_INIC_WF'] += self.TIME[i]
                data['H_FIM_WF'] += self.TIME[i]

        else:
            pass

        del data['i']
        self.request_action('wf_info_ack', data)

def main():
    app = FonteTrifasica()
    app.execute()
```

```
if __name__ == '__main__':
    main()
```

APÊNDICE D – CLASSE WAVEFORM

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
__author__ = 'keven'

import sys

reload(sys)
sys.setdefaultencoding('utf8')

import math
import commands
import matplotlib.pyplot as plt
import numpy as np
from fase import MicroServiceBase
import pylab
from matplotlib.figure import Figure
from matplotlib.backends.backend_gtk3agg import FigureCanvas
from matplotlib.backends.backend_gtk3 import (
    NavigationToolbar2GTK3 as NavigationToolbar)
import json
import time

pylab.rcParams['figure.figsize'] = (5.6, 5.6)
pi = 3.14159265359


class WaveForm(object):
    def __init__(self):
        self.samples = 128
        self.harmonica = np.arange(40) * 0.0
        self.eixoN = np.arange(self.samples)
        self.eixoFreq = self.eixoN[0:self.samples / 2] * 60
        self.fo = 60
        self.angulo = 0.0
        self.amp = 0.0
        self.fp = 1.0
        self.flagHarmonica = False
        self.calcWaveForm()
        self.calcFFT()
        # self.plotWaveForm()
        self.HARM = ['HARM_1', 'HARM_2', 'HARM_3', 'HARM_4', 'HARM_5',
'HARM_6', 'HARM_7', 'HARM_8', 'HARM_9',
                     'HARM_10',
                     'HARM_11', 'HARM_12', 'HARM_13', 'HARM_14',
'HARM_15', 'HARM_16', 'HARM_17', 'HARM_18', 'HARM_19',
                     'HARM_20',
                     'HARM_21', 'HARM_22', 'HARM_23', 'HARM_24',
'HARM_25', 'HARM_26', 'HARM_27', 'HARM_28', 'HARM_29',
                     'HARM_30',
                     'HARM_31', 'HARM_32', 'HARM_33', 'HARM_34',
'HARM_35', 'HARM_36', 'HARM_37', 'HARM_38', 'HARM_39',
                     'HARM_40']

    def plotWaveForm(self):
        a = []
        for valor in self.vetorWaveForm:
```

```
            a.append(int(valor * 0.7 * 2047 / 127) + 2047)
        print 'vetor[i] = {' + str(a)[1:-2] + '}'

    def calcWaveForm(self):
        n_rad = self.eixoN * self.fo * pi / (self.samples * 30.0)
        self.vetorWaveForm = np.round(self.amp * np.sin(n_rad +
self.angulo), 3)
        if self.flagHarmonica:
            for i in range(len(self.harmonica)):
                n_rad = self.eixoN * self.fo * pi * (2 + i) /
(self.samples * 30.0)
                self.vetorWaveForm += np.round(self.amp *
self.harmonica[i] * np.sin(n_rad + self.angulo) / 100, 2)

    def plotFFT(self, nome):
        if self.amp != 0:
            self.calcFFT()
            flagHarmonica = True
            fig, ax = plt.subplots()
            ax.plot(self.eixoFreq, self.vetorFFT, '.')
            ax.grid()
            ax.set(xlabel='Frequencia', ylabel=nome,
                   title=nome)
            major_ticks = np.arange(0, 1000, 60)
            major_ticks = np.append((major_ticks), (np.arange(1020, 2470,
120)))
            ax.set_xticks(major_ticks)
            plt.xlim(0, 2470)
            canvas = FigureCanvas(fig)
            plt.close(fig)
            plt.cla()
            plt.clf()
            return canvas
        else:
            return False

    def calcFFT(self):
        self.flagHarmonica = True
        self.calcWaveForm()
        self.vetorFFT = abs(np.fft.fft(self.vetorWaveForm, self.samples))
/ (self.samples / 2)
        self.vetorFFT = self.vetorFFT[0:self.samples / 2]

    def zerarHarm(self):
        for i in range(len(self.HARM)):
            self.harmonica[i] = 0.0

    def setHarm(self, data):
        for i in range(len(self.HARM)):
            self.harmonica[i] = data[self.HARM[i]]
        self.calcFFT()
        self.plotFFT(data['PARAM'])

    def plotWaveFormFases(self, waveform_2, waveform_3, nome):

        if nome == 'tensao':
            labels = ['VA', 'VB', 'VC', 'Tensao(V)', 'TENSAO']
        else:
            labels = ['IA', 'IB', 'IC', 'Corrente(A)', 'CORRENTE']

        fig, ax = plt.subplots()
```

```
        ax.plot(self.eixoN, self.vetorWaveForm, '-', label=labels[0])
        ax.plot(self.eixoN, waveform_2.vetorWaveForm, '-',
label=labels[1])
        ax.plot(self.eixoN, waveform_3.vetorWaveForm, '-',
label=labels[2])
        leg = plt.legend(loc='upper right', ncol=3, mode="normal",
shadow=True, fancybox=True)
        leg.get_frame().set_alpha(0.5)
        ax.set(xlabel='Amostras', ylabel=labels[3],
               title=labels[4])
        ax.grid()
        major_ticks = np.arange(0, 130, 16)
        ax.set_xticks(major_ticks)
        canvas = FigureCanvas(fig)
        plt.close(fig)
        plt.cla()
        plt.clf()
        return canvas

    def get_maior_amp(self, amplitudes):
        maior = 0
        for amplitude in amplitudes:
            if amplitude > maior:
                maior = amplitude
        return maior

    def plotFasores(self, waveform_2, waveform_3, waveform_4, waveform_5,
waveform_6):
        maior_tensao = self.get_maior_amp([self.amp, waveform_2.amp,
waveform_3.amp])
        maior_corrente = self.get_maior_amp([waveform_4.amp,
waveform_5.amp, waveform_6.amp])
        circle1 = plt.Circle((0, 0), 1, color='r', fill=False)
        va = plt.arrow(0.0, 0.0, 0.6, 0.8)
        x = np.arange(-1, 1.1, 0.1)
        y = np.arange(len(x)) * 0.0
        fig, ax = plt.subplots()
        ax.add_artist(circle1)

        if self.amp > 0:
            fator = (1 - 0.1 * maior_tensao / self.amp)
            if fator <= 0:
                fator = 0.0001
            ax.arrow(0.0, 0.0, fator * math.cos(self.angulo) * self.amp /
maior_tensao,
                     fator * math.sin(self.angulo) * self.amp /
maior_tensao, fc="k", ec="k",
                     head_width=0.05, head_length=0.1, label='VA',
color='r')
            ax.annotate("VA", xy=(
            math.cos(self.angulo) * self.amp / maior_tensao,
math.sin(self.angulo) * self.amp / maior_tensao))
        if waveform_2.amp > 0:
            fator = (1 - 0.1 * maior_tensao / waveform_2.amp)
            if fator <= 0:
                fator = 0.0001
            ax.arrow(0.0, 0.0, fator * math.cos(waveform_2.angulo) *
waveform_2.amp / maior_tensao,
                     fator * math.sin(waveform_2.angulo) * waveform_2.amp
/ maior_tensao, fc="k", ec="k",
```

```
                    head_width=0.05, head_length=0.1, label='VA',
color='r')
            ax.annotate("VB", xy=(math.cos(waveform_2.angulo) *
waveform_2.amp / maior_tensao,
                                  math.sin(waveform_2.angulo) *
waveform_2.amp / maior_tensao))

        if waveform_3.amp > 0:
            fator = (1 - 0.1 * maior_tensao / waveform_3.amp)
            if fator <= 0:
                fator = 0.0001
            ax.arrow(0.0, 0.0, fator * math.cos(waveform_3.angulo) *
waveform_3.amp / maior_tensao,
                     fator * math.sin(waveform_3.angulo) * waveform_3.amp
/ maior_tensao, fc="k", ec="k",
                     head_width=0.05, head_length=0.1, label='VA',
color='r')
            ax.annotate("VC", xy=(math.cos(waveform_3.angulo) *
waveform_3.amp / maior_tensao,
                                  math.sin(waveform_3.angulo) *
waveform_3.amp / maior_tensao))

        if waveform_4.amp > 0:
            fator = (1 - 0.1 * maior_corrente / waveform_4.amp)
            if fator <= 0:
                fator = 0.0001
            ax.arrow(0.0, 0.0, fator * math.cos(waveform_4.angulo) *
waveform_4.amp / maior_corrente,
                     fator * math.sin(waveform_4.angulo) * waveform_4.amp
/ maior_corrente, fc="k", ec="k",
                     head_width=0.05, head_length=0.1, label='VA',
color='r')
            ax.annotate("IA", xy=(math.cos(waveform_4.angulo) *
waveform_4.amp / maior_corrente,
                                  math.sin(waveform_4.angulo) *
waveform_4.amp / maior_corrente))

        if waveform_5.amp > 0:
            fator = (1 - 0.1 * maior_corrente / waveform_5.amp)
            if fator <= 0:
                fator = 0.0001
            ax.arrow(0.0, 0.0, fator * math.cos(waveform_5.angulo) *
waveform_5.amp / maior_corrente,
                     fator * math.sin(waveform_5.angulo) * waveform_5.amp
/ maior_corrente, fc="k", ec="k",
                     head_width=0.05, head_length=0.1, label='VA',
color='r')
            ax.annotate("IB", xy=(math.cos(waveform_5.angulo) *
waveform_5.amp / maior_corrente,
                                  math.sin(waveform_5.angulo) *
waveform_5.amp / maior_corrente))

        if waveform_6.amp > 0:
            fator = (1 - 0.1 * maior_corrente / waveform_6.amp)
            if fator <= 0:
                fator = 0.0001
            ax.arrow(0.0, 0.0, fator * math.cos(waveform_6.angulo) *
waveform_6.amp / maior_corrente,
                     fator * math.sin(waveform_6.angulo) * waveform_6.amp
/ maior_corrente, fc="k", ec="k",
```

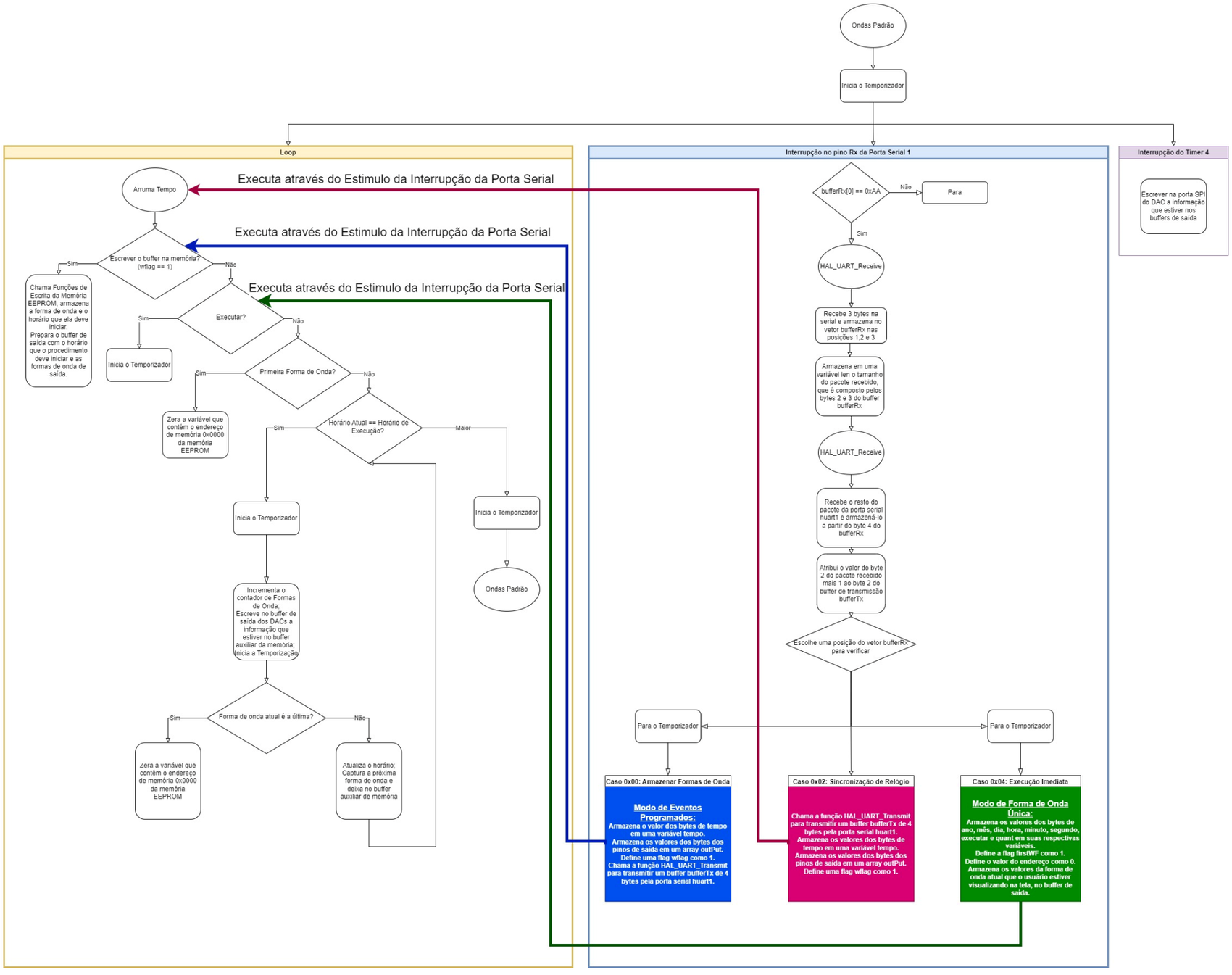
```
                    head_width=0.05, head_length=0.1, label='VA',
color='r')
            ax.annotate("IC", xy=(math.cos(waveform_6.angulo) *
waveform_6.amp / maior_corrente,
                                  math.sin(waveform_6.angulo) *
waveform_6.amp / maior_corrente))

        ax.plot(x, y, color='r')
        ax.plot(y, x, color='r')
        plt.xlim(-1.1, 1.1)
        plt.ylim(-1.1, 1.1)
        leg = plt.legend(loc='upper right', ncol=3, mode="normal",
shadow=True, fancybox=True)
        leg.get_frame().set_alpha(0.5)
        ax.set(xlabel='Real', ylabel='Imaginario',
               title='Diagrama Fasorial')
        ax.grid()
        canvas = FigureCanvas(fig)
        plt.close(fig)
        plt.cla()
        plt.clf()
        return canvas

    # fig = Figure(figsize=(5,4), dpi=100)
    # ax = fig.add_subplot(111)
    # ax.plot([1,2,3])
    # canvas = FigureCanvas(fig)
```

APÊNDICE E – FLUXOGRAMA COMPLETO DO FIRMWARE

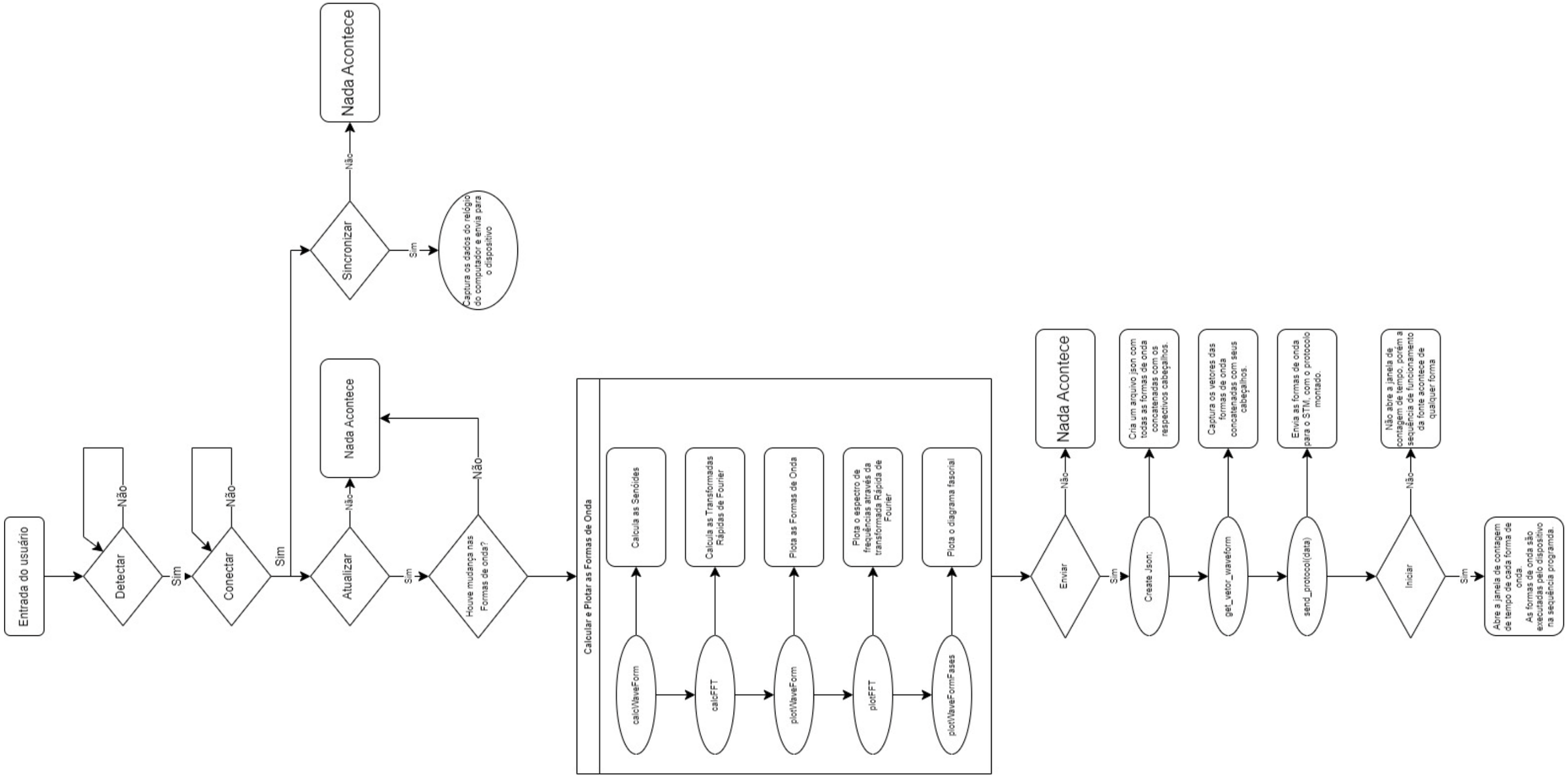
Figura 42: Fluxograma Completo do Firmware



Fonte: (Elaborado pelo autor, 2023)

APÊNDICE F – FLUXOGRAMA COMPLETO DO SOFTWARE DA INTERFACE DE USUÁRIO

Figura 43: Fluxograma completo do software da interface de usuário



Fonte: (Elaborado pelo autor, 2023)