Universidade Federal do ABC
Centro de Engenharia, Modelagem e Ciências Sociais Aplicadas

Sistema IoT para medição de energia elétrica

Luiz Ricardo Bitencourt

Santo André/SP
Setembro de 2024

Luiz Ricardo Bitencourt

Sistema IoT para medição de energia elétrica

Trabalho de Graduação apresentado ao curso de Engenharia de Informação da Universidade Federal do ABC, como parte dos requisitos necessários para a obtenção do grau de Bacharel em Engenharia de Informação.

Universidade Federal do ABC – UFABC

Centro de Engenharia, Modelagem e Ciências Sociais Aplicadas

Orientador: Prof. Dr. João Henrique Ranhel Ribeiro

Santo André/SP

Setembro de 2024

Resumo

Diante do crescente problema de perda de energia elétrica nos sistemas de distribuição, nota-se que uma das possibilidades de contenção está relacionada ao conhecimento do consumo de energia elétrica das unidades individuais, de forma digitalizada e centralizada, possibilitando aos órgãos de regulação o conhecimento das demandas e uso de forma setorizada. A informação de consumo atualizada em tempo real e de fácil acesso pode beneficiar também o consumidor final, possibilitando sua administração mais efetiva. Na chamada *Smart Grid*, considera-se de grande relevância a medição de energia elétrica, não apenas para tarifação, mas também para tornar as redes mais inteligentes por meio de monitoramento, controle, gerenciamento do fluxo de energia, acionamento de fontes suplementares, previsão de consumo e desagregação de cargas. Este trabalho apresenta o desenvolvimento de um sistema de Internet das Coisas (IoT) para medição de energia elétrica, desde o dimensionamento dos circuitos de condicionamento de sinais, programação dos algoritmos de cálculo de parâmetros da energia elétrica, até a disponibilização dos dados na nuvem. O foco é apresentar uma aplicação de um sistema de monitoramento de IoT, mas com a motivação de ser um possível produto aplicado no mercado de energia elétrica.

Palavras-chaves: medição de energia, internet das coisas, qualidade de energia.

Abstract

Given the growing issue of energy loss in distribution systems, one of the possible containment measures is related to the knowledge of electricity consumption in individual units, in a digitized and centralized manner. This enables regulatory bodies to understand the demands and usage in a sectorized way. Real-time, easily accessible consumption information can also benefit the end consumer, allowing for more effective management. In the so-called Smart Grid, electricity metering is considered highly relevant, not only for billing but also to make the grids smarter through monitoring, control, energy flow management, activation of supplementary sources, consumption forecasting, and load disaggregation. This work presents the development of an Internet of Things (IoT) system for electricity metering, from the design of signal conditioning circuits and programming of algorithms for calculating electrical parameters to the availability of the data in the cloud. The focus is to present an application of an IoT monitoring system, with the motivation of being a potential product for the electricity market.

Keywords: energy measurement, internet of things, energy quality.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

ABRADEE	Associação Brasileira de Distribuidores de Energia Elétrica
ADC	*Analog to Digital Converter*
ANEEL	Agência Nacional de Energia Elétrica
API	*Application Programming Interface*
CA	Corrente alternada
CC	Corrente contínua
CNC	Comando Numérico Computadorizado
DMA	*Direct Memory Access*
DPDT	*Double Pole, Double Throw*
EDA	*Electronic Design Automation*
EPE	Empresa de Pesquisa Energética
FP	Fator de Potência
GND	*Ground* (potencial de referência em circuitos eletrônicos)
HTML	*Hyper Text Markup Language*
I2C	*Inter-Integrated Circuit*
IDE	*Integrated Development Environment*
IO	*Input Output*
IoT	*Internet of Things*
JSON	*JavaScript Object Notation*
LCK	Lei das correntes de Kirchhoff
MCU	*Microcontroller Unit*
PCB	*Printed Circuit Board*
PLL	*Phase-Locked Loop*

RMS	*Root Mean Square*
RTC	*Real Time Clock*
SPI	*Serial Peripheral Interface*
TC	Transformador de Corrente
USART	*Universal Synchronous Asynchronous Receiver Transmitter*

Lista de símbolos

E	Energia elétrica
f	Frequência
$i(i)$	Vetor de amostras de corrente elétrica
$I_{offset}(i)$	Vetor de corrente elétrica com defasagem
i_{rms}	Valor eficaz de corrente elétrica
N	Número de amostras
P	Potência ativa
phi	Ângulo de defasagem entre tensão e corrente elétrica
Q	Potência reativa
S	Potência aparente
t	Tempo
v_{rms}	Valor eficaz de tensão elétrica
$v(i)$	Vetor de amostras de tensão elétrica
$x(i)$	Vetor de amostras de um sinal genérico
Y_{RMS}	Valor eficaz de um sinal genérico

Sumário

1 Introdução

Apresenta-se neste trabalho o desenvolvimento de um sistema de medição de energia elétrica, baseado na coleta da corrente e da tensão de um sistema de alimentação residencial monofásico, seguida do envio de dados de consumo de energia elétrica para um sistema de monitoramento na nuvem. Um maior detalhamento é apresentado nos tópicos que seguem.

1.1 Objetivos

O objetivo deste trabalho é apresentar o desenvolvimento completo de um protótipo capaz de:

- captar os sinais de tensão e corrente elétrica de uma instalação residencial monofásica, através de um circuito eletrônico dedicado e de baixo custo;
- condicionar tais sinais, de forma a eliminar ruídos sem importância nas análises propostas, além de prepará-los para a conversão analógico-digital;
- processar digitalmente os sinais em um microcontrolador de forma a obter os parâmetros de interesse na medição do consumo de energia elétrica em uma instalação residencial;
- enviar os dados oriundos do processamento digital para armazenamento em nuvem e visualização remota.

Diante do exposto, apresenta-se aqui o dimensionamento completo de um circuito eletrônico capaz de captar e condicionar os sinais de tensão e corrente elétrica de uma instalação monofásica, realizando a digitalização destes sinais e tratando-os de forma a se obter os seguintes parâmetros:

- tensões e correntes eficazes, de pico e pico a pico;
- período e frequência;
- defasagem entre tensão e corrente e fator de potência;
- potências ativa, reativa e aparente.

Além disso, a partir do processamento dos dados obtidos, o sistema deve gerar as informações do consumo de energia elétrica, podendo representar o montante diário ou o acumulado dentro de um período de tempo estipulado.

1.2 Justificativa

Segundo o Anuário Estatístico de Energia Elétrica de 2024 (EPE, 2024), considerando o estado de São Paulo, as categorias de consumidores mais representativas são as residenciais e as industriais, compondo mais de 67% do total de consumo de energia elétrica. Este cenário pode ser observado nos dados[1] das tabelas exibidas nas Figuras 1 e 2, na qual são apresentados o consumo (em GWh) por categoria de consumidor e o número de unidades consumidoras no estado de São Paulo para o intervalo de 2014 a 2023.

Classe	2014	2015	2016	2017	2018	2019	2020	2021	2022	2023	% (2023/2022)
Comercial	29.595	29.350	27.957	28.186	28.408	29.575	26.131	27.744	28.873	30.764	6,5%
Consumo Próprio	199	212	201	191	182	205	178	222	215	213	-0,9%
Iluminação Pública	3.167	3.165	3.196	3.260	3.238	3.251	3.078	3.178	2.741	2.739	-0,1%
Industrial	52.254	48.893	46.508	47.485	48.805	47.636	45.960	51.289	51.456	50.027	-2,8%
Poder Público	3.222	3.168	3.031	3.028	2.997	3.053	2.432	2.530	2.802	2.991	6,7%
Residencial	39.437	38.006	38.091	38.969	39.924	41.150	42.085	42.627	42.360	45.625	7,7%
Rural	3.348	3.047	3.152	3.351	3.487	2.372	3.830	3.895	3.974	3.535	-11,0%
Serviço Público	5.260	4.975	5.034	5.138	5.394	5.608	5.531	5.875	5.601	5.443	-2,8%

Figura 1 – Consumo por classe no estado de São Paulo entre 2014 e 2023.
Fonte: EPE (2024).

Classe	2014	2015	2016	2017	2018	2019	2020	2021	2022	2023	% (2023/2022)
Comercial	1.077.671	1.126.185	1.116.948	1.117.386	1.116.499	1.130.843	1.125.725	1.133.622	1.155.375	1.186.639	2,7%
Consumo Próprio	4.881	1.575	1.573	1.596	1.625	1.528	1.576	1.680	1.692	3.170	87,4%
Iluminação Pública	19.736	21.283	22.223	23.063	22.544	24.495	24.987	21.169	22.147	22.434	1,3%
Industrial	124.041	106.380	105.912	104.681	102.118	100.849	97.897	95.168	90.113	91.839	1,9%
Poder Público	86.553	88.096	88.224	88.265	88.281	89.081	90.705	91.847	93.873	97.714	4,1%
Residencial	15.909.887	16.265.435	16.594.746	16.953.937	17.384.160	17.832.649	18.019.714	18.370.167	18.763.521	19.470.378	3,8%
Rural	266.754	270.190	272.662	274.627	276.756	268.347	271.678	267.690	252.626	249.363	-1,3%
Serviço Público	12.964	13.301	13.586	13.845	14.154	14.624	14.976	15.181	15.372	15.650	1,8%

Figura 2 – Número de consumidores por classe no estado de São Paulo entre 2014 e 2023.
Fonte: EPE (2024).

Para melhor visualização, a Figura 3 apresenta em forma gráfica os dados de consumo de energia elétrica para o estado de São Paulo em 2023, tendo sua maior representatividade com as categorias residencial, com 32% e industrial, com 35%.

Tomando-se os dados relativos à unidades consumidoras, a categoria residencial corresponde a 92% do total de unidades no estado de São Paulo em 2023, com 19.470.378 unidades. A representatividade deste fato pode ser observada na Figura 4.

Ainda de acordo com EPE (2024), observou-se que, somente em 2023, houve 20% de perdas[2] no estado de São Paulo. O conceito de perda de energia é apresentado pela Agência Nacional de Energia Elétrica (ANEEL) como sendo a energia gerada mas não comercializada, que ocupa as linhas de transmissão e as redes de distribuição (ANEEL, 2022). Pode-se distinguir dois principais tipos de perdas: as técnicas e as não técnicas.

[1] Dados da Tabelas 4.1 e 4.1 do referido anuário.
[2] Dados da Tabela 2.8 do referido anuário.

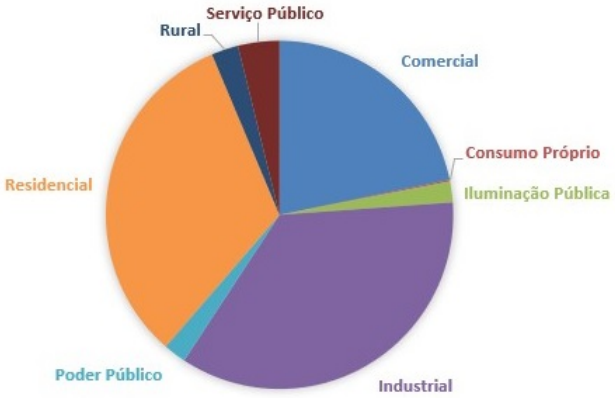


Figura 3 – Consumo de energia elétrica em São Paulo por categoria em 2023.
Fonte: Gerado a partir dos dados da EPE (2024).

Figura 4 – Quantidade de unidades consumidoras em São Paulo por categoria em 2023.
Fonte: Gerado a partir dos dados da EPE (2024).

As primeiras podem ser: perdas por efeito joule; perdas em núcleos de transformadores; perdas dielétricas etc. Já as perdas não técnicas incluem: furto ou desvio direto da rede; fraude por adulterações no medidor; além dos erros de medição e de faturamento.

Segundo o Relatório de Perdas de Energia Elétrica da ANEEL (ANEEL, 2022), as perdas totais sobre a energia injetada representam cerca de $14,1\%$ da energia injetada no Brasil em 2023, sendo aproximadamente $7,4\%$ de perdas técnicas e $6,7\%$ de perdas não técnicas, conforme Figura 5.

Considerando o montante gerado de 708.119 GWh em 2023 (EPE, 2024), as perdas não técnicas regulatórias correspondem a um custo estimado em R\$$6,9$ bilhões, sendo repassadas ao consumidor (ANEEL, 2022), e as perdas não técnicas reais, na ordem de R\$$9,96$ bilhões.

Conforme Figura 6, no estado de São Paulo, as perdas não técnicas estimadas em 2023 pela principal distribuidora, a Enel, chegam a R\$$142,1$ milhões, podendo ultrapassar R\$200 milhões se considerar as demais distribuidoras do estado.

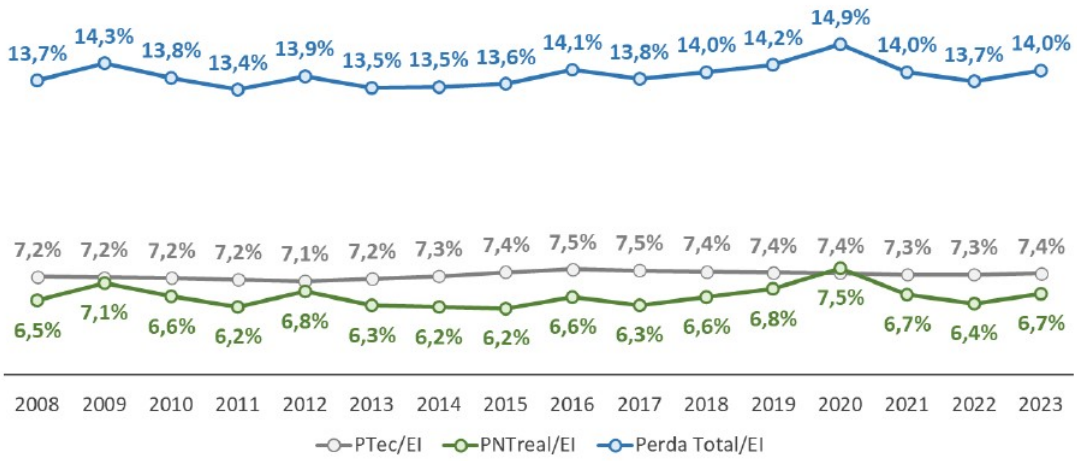


Figura 5 – Perdas sobre a energia injetada entre 2008 e 2023.
Fonte: ANEEL (2022).

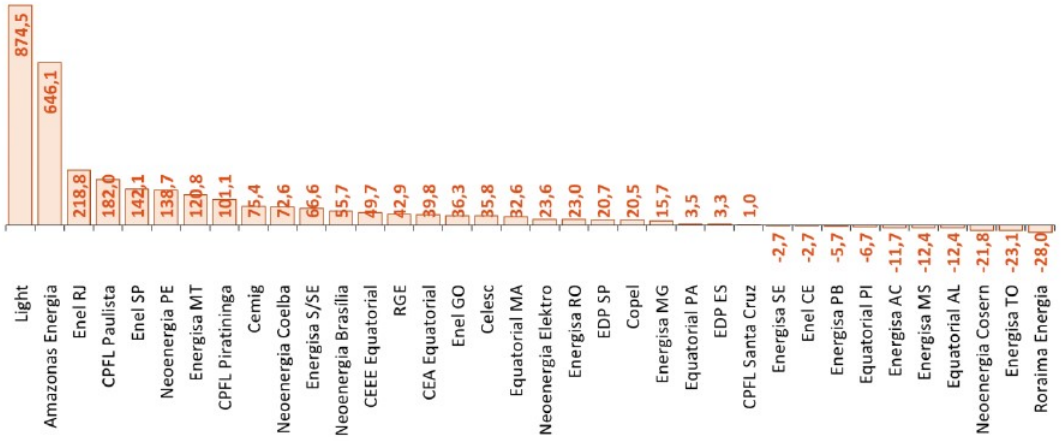


Figura 6 – Valores de perdas não técnicas por distribuidora em 2023.
Fonte: ANEEL (2022).

Diante dos dados apresentados, observa-se a necessidade de controlar as perdas, em especial as não técnicas, dado o prejuízo econômico gerado. Nota-se que uma possível abordagem no controle de tais perdas está ligado a um controle mais preciso do consumo. Dado o elevado número de unidades consumidoras, uma possibilidade deste controle pode estar relacionado à adoção de sistemas de medição inteligentes, capazes de fornecer dados que podem ser usados para uma análise criteriosa do sistema de distribuição visando a localização dos pontos de perda, independente de sua natureza, possibilitando uma análise posterior para tomada de ações preventivas. Chega-se portanto ao propósito deste trabalho, no desenvolvimento de um protótipo capaz de realizar tal função. A ideia é que o dispositivo possa ser usado em sistemas mais complexos em trabalhos futuros, como o apresentado por Souza (2016) na detecção de perdas de energia em sistemas de distribuição.

Nota-se, adicionalmente, uma potencial funcionalidade do protótipo sugerido na conscientização do consumo por parte do cliente final. Os dados gerados pelo sistema de medição devem ser transparentes ao consumidor que, tendo consciência de seus valores,

pode se utilizar dos mesmos como base disciplinar para consumo de energia elétrica, evitando desperdícios.

1.3 Metodologia

Este trabalho possui caráter prático, baseado no desenvolvimento de um protótipo com o qual se obtém dados de energia elétrica que, após processados, resultam nas informações dos parâmetros e consumo de energia elétrica.

1.4 Estrutura do trabalho

O Capítulo 1 contempla a introdução e conceituação do tema, esclarecendo os objetivos que norteiam este trabalho bem como a motivação e justificativa seguida da exposição da metodologia adotada.

O Capítulo 2 descreve os materiais e métodos adotados para desenvolvimento do protótipo, incluindo breve introdução teórica, bem como da programação e recursos de processamento digital de sinais para as análises relacionadas à qualidade de energia elétrica.

O Capítulo 3 descreve os resultados dos ensaios realizados bem como os dados obtidos seguidos de uma breve discussão.

O Capítulo 4 apresenta uma discussão final sobre os objetivos prometidos e os resultados obtidos, bem como uma breve discussão sobre as possibilidades futuras para esta área de pesquisa.

2 Metodologia e desenvolvimento

Neste capítulo é apresentado todo o detalhamento do dimensionamento dos circuitos eletrônicos, desenvolvimento dos algoritmos e programação do sistema web para apresentação dos dados coletados pelo sistema de medição.

Para melhor compreensão do desenvolvimento, o sistema foi dividido em partes bem definidas para que seja possível detalhar todas as etapas de projeto. Inicialmente apresenta-se o projeto dos circuitos eletrônicos, detalhando a medição de tensão e corrente, gerador de *zero crossing*, circuito do microcontrolador com as etapas de comunicação e armazenamento de dados e concluindo com o detalhamento da fonte de alimentação. Na sequência, é feito um detalhamento do projeto de *software*, tanto do lado do microcontrolador com a programação dos algoritmos, como da lado da nuvem, com o projeto da aplicação web para visualização dos dados. As seções a seguir apresentam todo este detalhamento.

2.1 Projeto dos circuitos eletrônicos

Nesta seção, são apresentados os recursos de *hardware* utilizados no trabalho, divididos nos subtópicos a seguir, definindo os circuitos de condicionamento para medição dos sinais de tensão e corrente elétrica. Ao final, é apresentado o circuito completo de medição e conversão dos sinais.

2.1.1 Sistema de medição de tensão elétrica

Para realização da medição de tensão elétrica, o circuito proposto deve possuir as etapas descritas no diagrama da Figura 7.

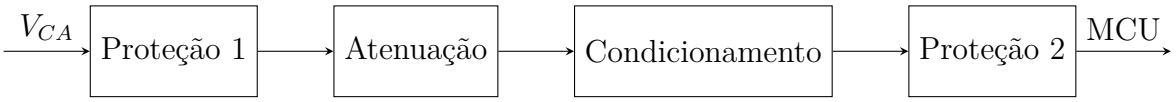


Figura 7 – Etapas de tratamento do sinal de tensão elétrica

O termo "V_{CA}" faz referência ao sinal de tensão alternado de entrada do circuito. A etapa "Proteção 1", deve proteger o circuito de surtos de grande intensidade incomuns, que podem ocasionar falhas nas etapas seguintes do sistema de medição. A etapa "Atenuação" é responsável por diminuir a amplitude do sinal, de forma a compatibilizar-se com os níveis de tensão do circuito de medição. Na etapa "Condicionamento" é realizada a atenuação do ruído de modo comum, bem como a adição de um *offset* para que o sinal, de origem CA, possa ser lido como CC, além da possibilidade de ser adicionado um ganho ao sinal. A última etapa, "Proteção 2", fornece proteção ao microcontrolador (cuja referência se

faz ao termo *Microcontroller Unit* – MCU), de forma a evitar que possíveis surtos, não protegidos na primeira etapa, possam ocasionar falhas no circuito de conversão embarcado.

A descrição das etapas do circuito de medição de tensão, bem como seu dimensionamento, é realizada nos subtópicos a seguir.

2.1.1.1 Etapa de atenuação

Para reduzir a amplitude do sinal de tensão, emprega-se a técnica do divisor resistivo, dado seu baixo custo e relativa linearidade quando comparada, por exemplo, ao uso de transformadores de potencial. A Figura 8 mostra a topologia do circuito utilizado.

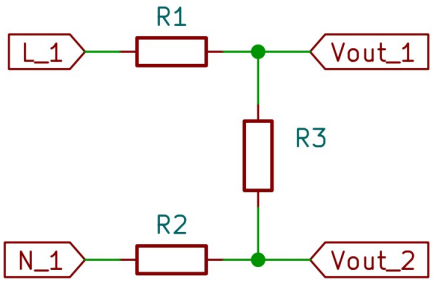


Figura 8 – Circuito divisor resistivo para atenuação da tensão de entrada.

No circuito divisor resistivo empregado, foram utilizados três resistores em série de forma a evitar que um eventual rompimento de alguma das conexões pudesse resultar na existência de alta tensão na saída do circuito. Para o dimensionamento dos resistores $R1$, $R2$ e $R3$, faz-se o cálculo do divisor de tensão, conforme Equação 2.1, levando-se em conta a amplitude do sinal de tensão de entrada V_p e o valor desejado da amplitude atenuada na saída V_{out}.

$$V_{out} = V_p \cdot \frac{R3}{R1 + R2 + R3} \tag{2.1}$$

O foco deste projeto é aplicar o sistema de medição em uma instalação elétrica monofásica, realizando medições entre Fase e Neutro, cuja tensão eficaz é de $127V/60Hz$. Desta forma, $V_p = 127 \cdot \sqrt{2} = 179,60V$. Já o sinal de saída, deve ser lido por um microcontrolador cuja entrada analógica aceita tensões de 0 a $3,3V$, o que motiva a adoção de $V_{out} = 3V$, mantendo uma margem de segurança para leitura de possíveis picos e sobressinais. Além disso, para simplificação, adota-se $R1 = R2 = R$, obtendo-se:

$$3 = 179,60 \cdot \frac{R3}{2R + R3}$$
$$6R + 3R3 = 179,6R3$$

$$R3 = \frac{6R}{176,6} \tag{2.2}$$

de onde se pode obter o valor de R, ou seja, $R1$ e $R2$, além do valor de $R3$.

Uma das estratégias de definição do valor dos resistores $R1$, $R2$ e $R3$ é atribuir, arbitrariamente, um valor para R na Equação 2.2.

Para que o efeito de consumo de potência pelo circuito seja mínimo, propõe-se a realização de um estudo numérico para definição do melhor valor de R. Soma-se a isso a ideia de que o circuito de medição sempre influencia no objeto medido, então a proposta é que esta influência seja mínima. Sendo assim, fechando-se a malha do circuito da Figura 8, pode-se calcular a corrente usando a primeira Lei de Ohm, como segue:

$$
\begin{aligned}
I &= \frac{V_p}{R_{eq}} \\
I &= \frac{179,6}{R1 + R2 + R3} \\
I &= \frac{179,6}{R + R + \frac{6R}{176,6}} \\
I &= \frac{179,6}{2R + \frac{6R}{176,6}}
\end{aligned} \tag{2.3}
$$

Finalmente, variando-se R de 100Ω a $10M\Omega$ da Equação 2.3, obtém-se o gráfico da Figura 9.

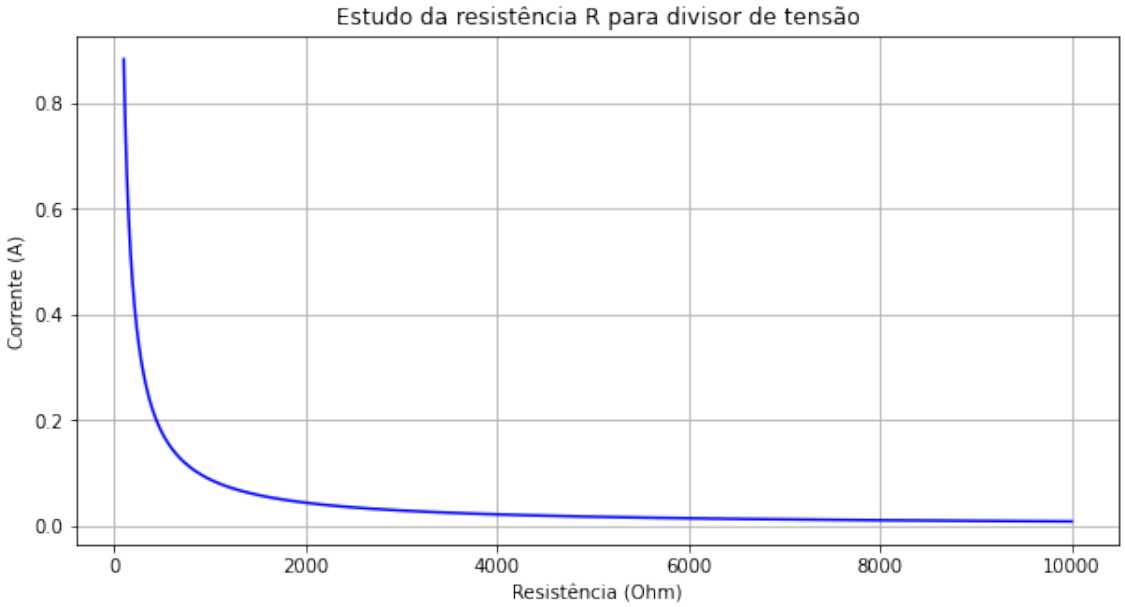


Figura 9 – Gráfico da variação de corrente com a alteração do valor de R.

É esperado que a corrente seja tão menor quanto maior for o valor dos resistores, mas com este estudo pode-se encontrar rapidamente os melhores valores de acordo com a corrente desejada (visando o menor consumo de potência). Foram adotados os seguintes valores para $R1$ e $R2$:

$$R1 = 10M\Omega$$
$$R2 = 10M\Omega$$

de forma a corrente do circuito ser $8,83\mu A$, conforme simulação.

Desta forma, usando a Equação 2.2, obtém-se:

$$R3 = 339750.85\Omega$$

adotando-se o valor de $R3 = 330k\Omega$ por ser o valor comercial mais próximo.

Recalculando-se a tensão do circuito divisor, substituindo os valores teóricos pelos valores comerciais, obtém-se:

$$V_{out} = 2.92V$$
$$I = 8,83\mu A$$

cujos valores estão dentro do esperado para esta aplicação.

O circuito final do divisor de tensão é representado na Figura 10.

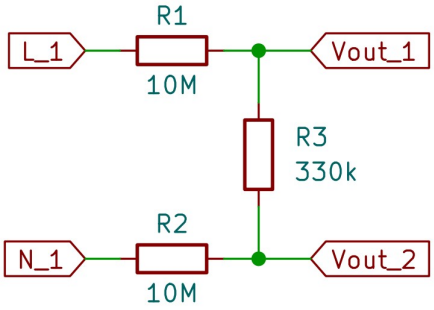


Figura 10 – Circuito divisor resistivo final.

Para validação da ideia, foram realizados ensaios práticos com a topologia proposta bem como os valores de dimensionamento dos componentes. Destes ensaios, cujo detalhamento poderá ser visto no Capítulo 3, foi constatado que a entrada do circuito ficou flutuante em relação ao circuito de medição. Isto é, não há referência comum entre as duas etapas do circuito, o que causou baixa qualidade no sinal obtido. Isso motivou a adaptação do circuito, adicionando-se capacitores de acoplamento entre os pontos `Vout_1` e `Vout_2`, além da remoção do resistor R3, cujo efeito pode ser desprezado dada a alta impedância de entrada do amplificador operacional da etapa seguinte (condicionamento). Essas modificações podem ser vistas no circuito final, na seção 2.1.1.4.

2.1.1.2 Etapa de condicionamento

Na etapa de condicionamento do sinal de tensão, deve-se adicionar um nível CC para deslocar o sinal verticalmente de forma que seja sempre positivo, ou seja, deve-se

realizar um *offset* positivo no sinal. Além disso é importante eliminar o ruído de modo comum, para que o sinal captado tenha a menor contaminação possível. Para realizar estes objetivos, partiu-se de um circuito amplificador diferencial, conforme Figura 11.

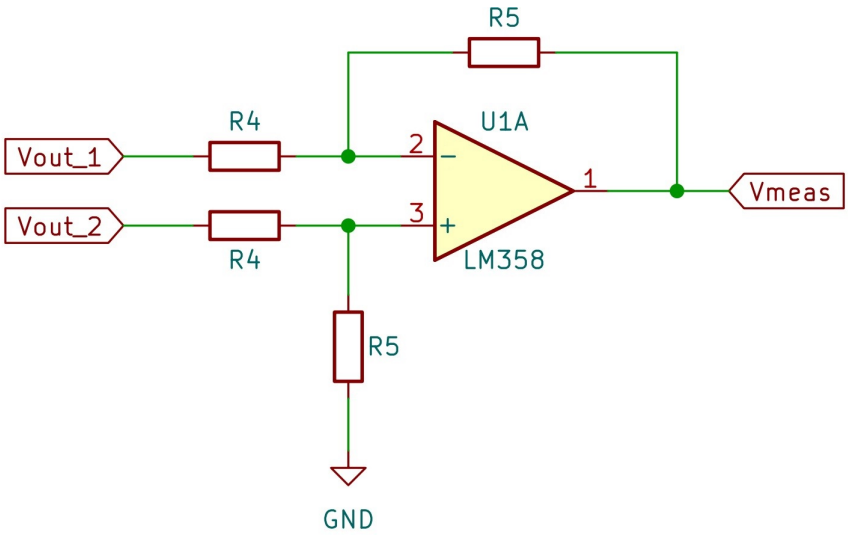


Figura 11 – Circuito amplificador diferencial.

A adoção do LM358 foi motivada pela sua fácil aquisição no mercado nacional, além do seu baixo custo. É importante destacar que existem no mercado outros circuitos integrados com amplificadores operacionais que podem ser mais apropriados para aplicações de instrumentação e medições de sinais elétricos. O estudo e ensaios com tais componentes pode ser uma motivação para trabalhos futuros, inclusive realizando a comparação dos dados obtidos com o projeto atual.

Considerando a adição de um nível CC, o circuito da Figura 11 deve ser modificado de forma a alterar sua referência (GND), inserindo em seu lugar uma tensão CC para gerar um *offset* de tensão no sinal. Para tanto, considera-se o circuito da Figura 12.

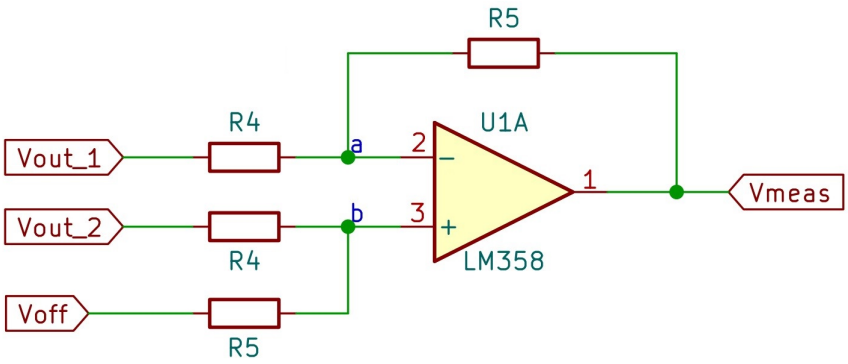


Figura 12 – Circuito amplificador diferencial com nova referência.

Realizando a análise nodal no circuito da Figura 12, conforme descrito por Pertence (1990), considerando os nós a e b, obtém-se as seguintes equações pela Lei das Correntes de Kirchhoff (LCK):

$$\frac{V_{out1} - V_a}{R4} + \frac{V_{meas} - V_a}{R5} = 0$$
$$\frac{V_{out2} - V_b}{R4} + \frac{V_{off} - V_b}{R5} = 0$$

cuja solução, tomando-se $V_a = V_b$, resulta:

$$V_{meas} = (V_{out1} - V_{out2}) \frac{R5}{R4} + V_{off} \tag{2.4}$$

onde os sinais `Vout_1` e `Vout_2` são os pontos de medição do sinal de tensão elétrica, conforme indicados na Figura 12.

A Equação 2.4 mostra que basta adicionar uma tensão contínua no ponto V_{off} do circuito da Figura 12 para que o efeito de *offset* de tensão seja realizado. Além disso, como realiza-se uma subtração dos sinais de saída do divisor resistivo (circuito da Figura 10), elimina-se o ruído de modo comum. Os resistores $R4$ e $R5$ podem atuar de forma a alterar o ganho do sinal, mas para o propósito do trabalho e por conveniência, adotam-se valores iguais para $R4$ e $R5$ como 10kΩ.

O circuito final desta etapa pode ser visto na Figura 13, com a referência dos resistores ajustada bem como seus valores finais. Para a tensão de *offset* foi calculado um divisor resistivo considerando $R8$ e $R9$ de $3,3k\Omega$ e $22k\Omega$, respectivamente, adotados de forma empírica, onde:

$$V_{off} = 12 \cdot \frac{3300}{22000 + 3300} = 1,56V$$

cuja tensão de saída satisfaz o propósito do *offset* de tensão a ser aplicado no sinal. Pode-se ainda usar um trimpot no lugar de $R9$, de forma a facilitar o ajuste fino de *offset*. Vale ressaltar que foi adotada uma tensão de 12V, não simétrica, na alimentação do circuito amplificador operacional para que houvesse sobra de tensão para funcionamento dos circuitos internos do circuito integrado, além da facilidade de disponibilização de fontes de alimentação modulares neste nível de tensão.

Ainda com relação à Figura 13, os pinos 8 e 4 do LM358 são, respectivamente, o V+ (alimentação positiva) e o V- (alimentação negativa). Tais pinos estão ocultos nesta representação, mas podem ser visualizados no circuito completo disponível no Apêndice A.

2.1.1.3 Etapas de proteção

A proteção do circuito de medição de tensão é composta de duas etapas: proteção da entrada do divisor resistivo; e, proteção da entrada do MCU. Para proteção de entrada

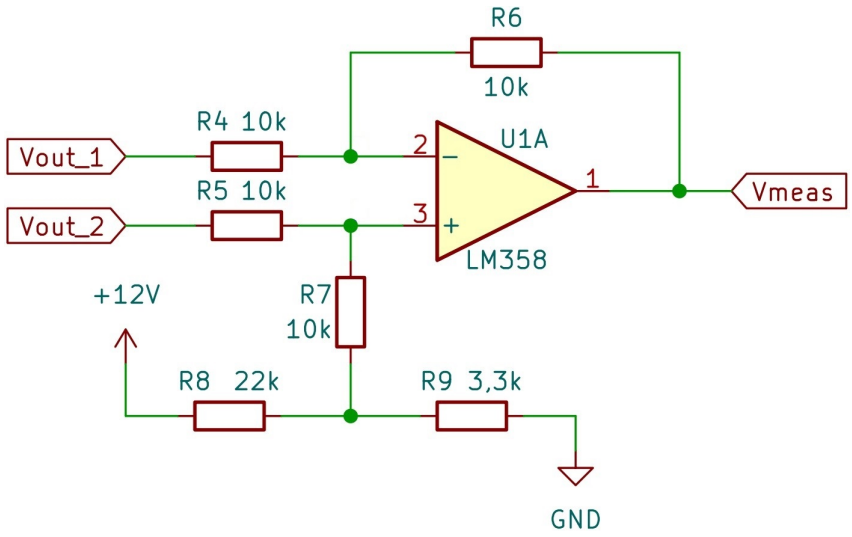


Figura 13 – Circuito de condicionamento final para medição de tensão.

do divisor, utiliza-se um tradicional circuito baseado em varistor, conforme Figura 14, sendo dimensionado para uma tensão nominal de 127V. O fusível adotado deve ser de ação rápida, com corrente de fusão de $1mA$, o que corresponde a 100 vezes a corrente nominal do circuito divisor resistivo.

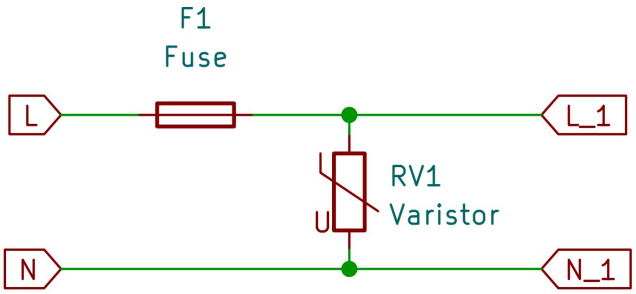


Figura 14 – Circuito de proteção da entrada de tensão CA.

Para a proteção da entrada do ADC do microcontrolador, utiliza-se apenas um diodo Zener com $V_z = 3,3V$, conforme Figura 15. Desta forma, caso ocorra algum sinal maior do que a tensão Zener, ele é drenado pelo diodo.

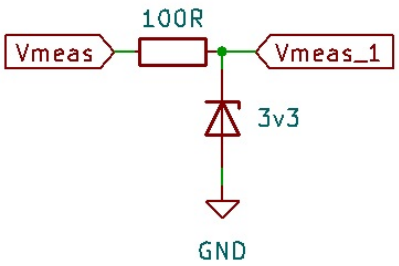


Figura 15 – Circuito de proteção da entrada do MCU.

2.1.1.4 Circuito final para medição de tensão

O circuito final para medição de tensão é apresentado na Figura 16.

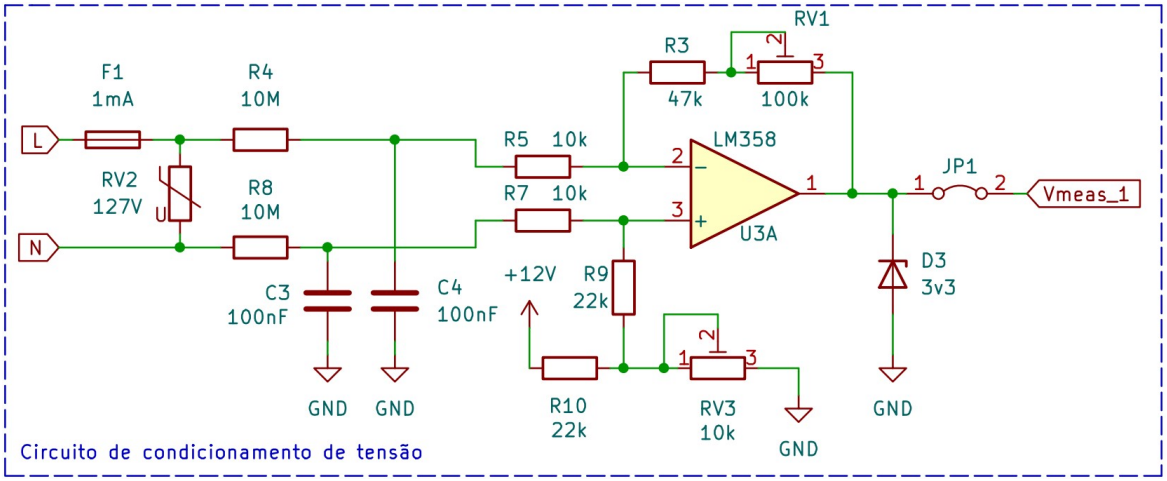


Figura 16 – Circuito final para medição de tensão.

Nota-se que alguns valores de componentes foram ajustados de forma a criar uma margem de variação quando aplicados os trimpots RV1 e RV3. Isso aconteceu com o resistor de realimentação R3 e o conjunto de resistores de *offset*: R9 e R10. A inclusão dos trimpots ocorreu para que fosse possível a realização de ajustes finos do sinal obtido, conforme pode ser verificado na descrição dos testes no Capítulo 3.

Os pinos 8 e 4 do LM358 são, respectivamente, o V+ (alimentação positiva) e o V- (alimentação negativa). Tais pinos estão ocultos nesta representação, mas podem ser visualizados no circuito completo disponível no Apêndice A.

2.1.2 Sistema de medição de corrente elétrica

Para realização da medição de corrente elétrica, o circuito proposto deve possuir as etapas descritas no diagrama da Figura 17.

Figura 17 – Etapas de tratamento do sinal de corrente elétrica

O termo "I_{CA}" faz referência ao sinal de corrente cujo circuito está observando, sendo proveniente da instalação elétrica e identificado pelo transdutor de corrente CA, baseado em um transformador de corrente (TC), conforme será apresentado na seção 2.1.2.1. Na etapa "Conversão I-V", o transdutor de corrente converte o sinal de corrente em um sinal de tensão com comportamento proporcional. De acordo com as características do transdutor usado, a etapa "Ganho" realiza o ajuste do nível de tensão para se adequar

a faixa de valores aceitáveis pelo MCU. Finalmente, a etapa “Proteção” evita que sinais sobrepostos causem danos ao ADC embarcado no microcontrolador.

A descrição das etapas do circuito de medição de corrente, bem como seu dimensionamento, é realizada nos sub tópicos a seguir.

2.1.2.1 Conversão V-I

Para converter o sinal de corrente em um sinal de tensão proporcional, a fim de ser lido pela entrada analógica do microcontrolador, foi utilizado um transdutor de corrente[1], conforme Figura 18.

Figura 18 – Transdutor de corrente.

Trata-se de um transdutor indutivo fabricado pela YHDC, baseado em um TC, cuja entrada máxima de $30A$ gera uma saída máxima de $1V$. A escolha deste transdutor foi motivada pelo fato de não ser invasivo, ou seja, não há necessidade de interrupção do condutor, o qual se deseja medir corrente, para sua instalação.

Como o sinal de saída do transdutor é de natureza CA, deve-se adicionar um nível CC para correta utilização com o circuito conversor analógico-digital. Dessa forma, utiliza-se o circuito da Figura 19 para inserir um *offset* de tensão no circuito.

Nota-se que a adoção de um trimpot $RV1$ possibilita o ajuste do nível de tensão adequado conforme a aplicação prática.

[1] As informações detalhadas do transdutor podem ser vistas no *datasheet*, disponível no Anexo A.

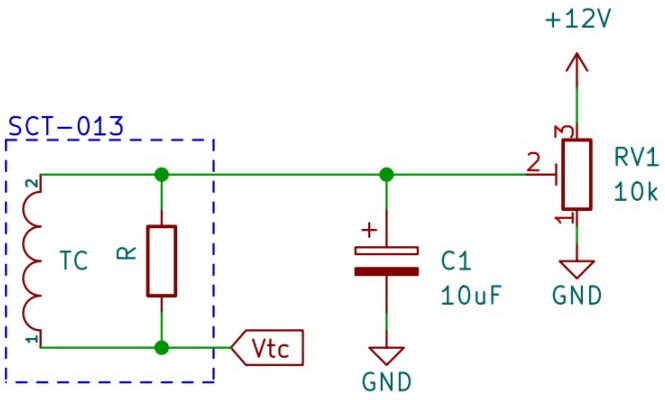


Figura 19 – Circuito para *offset* de tensão no transdutor de corrente.

2.1.2.2 Etapa de ganho

Para elevar a tensão de saída do transdutor de corrente, utiliza-se o circuito amplificador não inversor da Figura 20.

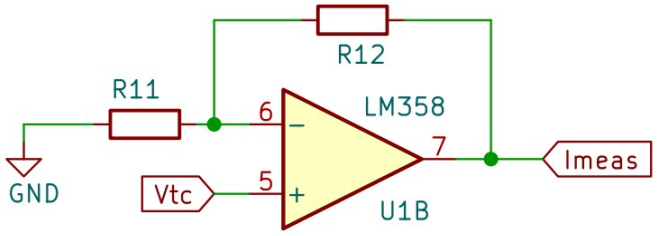


Figura 20 – Circuito de ganho para medição de corrente.

A tensão de saída do circuito amplificador operacional não inversor, conforme Pertence (1990), é dada pela Equação 2.5.

$$V_{Imeas} = Vtc \cdot \left(\frac{R12}{R11} + 1\right) \tag{2.5}$$

Considerando o transdutor usado, cuja saída máxima é de $1V$, nota-se que a amplificação deve ser de, pelo menos, 3 vezes o sinal de sua saída para que o microcontrolador receba um sinal com boa faixa de aproveitamento. Desta forma pode-se utilizar um trimpot no lugar de $R12$, mantendo $R11$ fixo, para que seja possível calibrar o ganho conforme a necessidade. Adotou-se, portanto, os valores de $22k\Omega$ e $100k\Omega$ para $R11$ e $R12$, respectivamente, onde $R12$ será um trimpot, possibilitando um ganho máximo próximo de 5 vezes.

A Figura 21 mostra o circuito final para esta etapa. De maneira semelhante ao descrito anteriormente, os pinos 8 e 4 do LM358 são, respectivamente, o V+ (alimentação positiva) e o V- (alimentação negativa). Tais pinos estão ocultos nesta representação, mas podem ser visualizados no circuito completo disponível no Apêndice A.

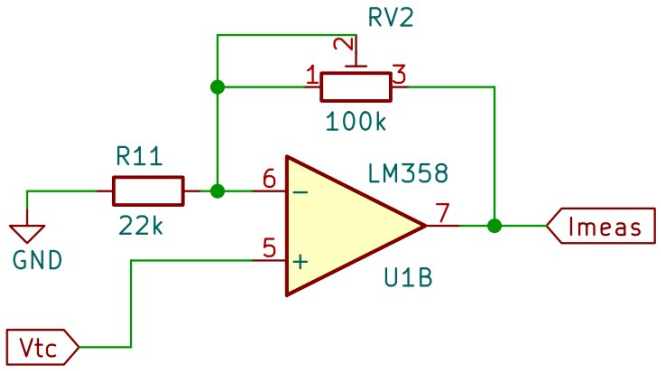


Figura 21 – Circuito de ganho e *offset* de tensão para medição de corrente.

2.1.2.3 Etapa de proteção

Para a proteção da entrada do ADC do microcontrolador, utiliza-se a mesma topologia da proteção usada no circuito de medição de tensão, com um diodo Zener de $Vz = 3,3V$, conforme Figura 22.

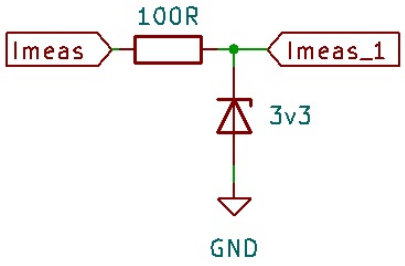


Figura 22 – Circuito de proteção da entrada do MCU.

2.1.2.4 Circuito final para medição de corrente

O circuito final para medição de tensão é apresentado na Figura 23.

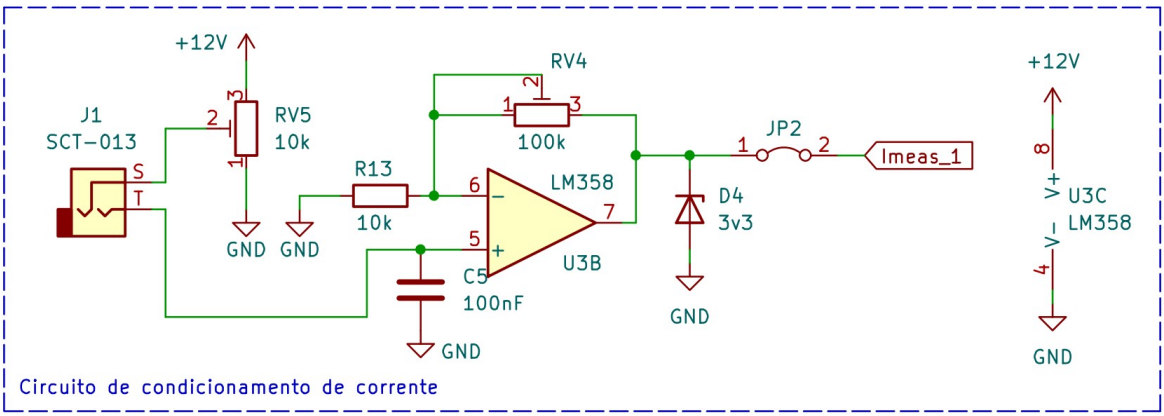


Figura 23 – Circuito final para medição de corrente.

2.1.3 Circuito gerador de *zero crossing*

Para auxiliar no sincronismo dos sinais medidos, foi adicionado um circuito indicador de cruzamento por zero para o sinal de tensão. Este tipo de circuito é conhecido na literatura como *zero crossing*, e é baseado em uma ponte retificadora e um acoplador óptico configurados de tal forma que um pulso de curta duração seja gerado no cruzamento da senoide de tensão por $0V$. A Figura 24 mostra o circuito utilizado.

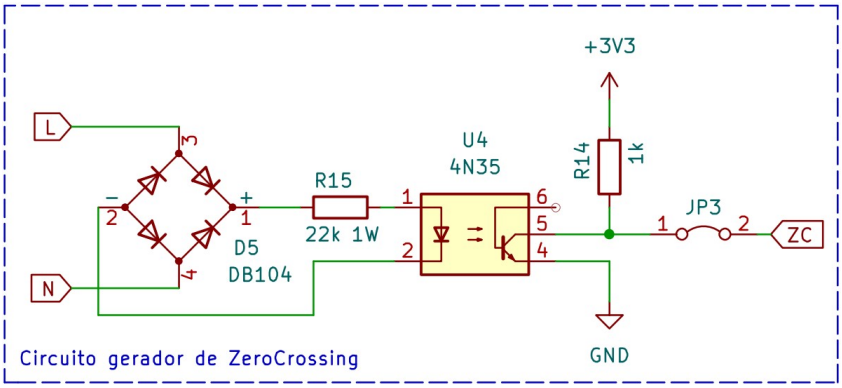


Figura 24 – Circuito gerador de *zero crossing*.

2.1.4 Circuito do microcontrolador

Uma vez que os sinais de tensão e corrente estão pré condicionados, eles são inseridos nas entradas dos ADCs do microcontrolador responsável pelo processamento digital de tais sinais. Para este trabalho, foi adotado o STM32F103, um microcontrolador de 32 bits com núcleo de arquitetura ARM, fabricado pela ST Microelectronics. Pela simplicidade de montagem do circuito básico do microcontrolador, foi adotada a placa de testes não oficial da ST, chamada BluePill, conforme Figura 25, cujo esquema eletrônico pode ser observado no Anexo B.

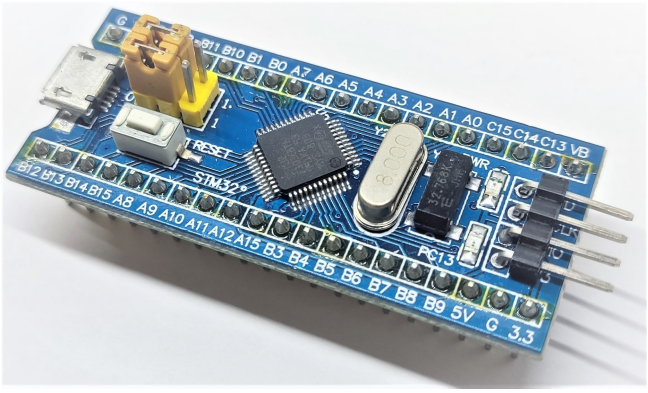


Figura 25 – Sistema microcontrolado baseado no STM32F103.

A escolha deste microcontrolador foi motivada pelo domínio do autor na arquitetura, além da disponibilidade deste recurso. É importante ressaltar que, em projetos futuros,

pode-se realizar uma escolha mais criteriosa levando-se em consideração a capacidade de processamento, custo e disponibilidade de recursos necessários ao produto final.

A Figura 26 mostra o circuito adotado para o microcontrolador. Vale ressaltar que U1 faz referência à placa da BluePill, e não diretamente ao microcontrolador.

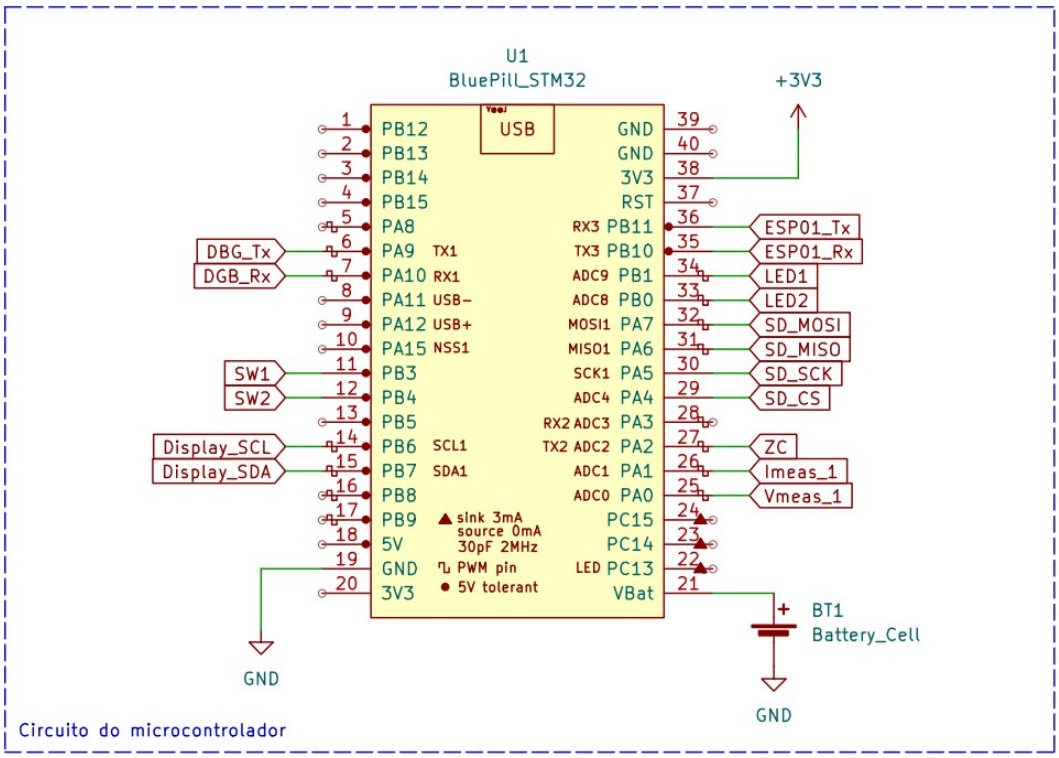


Figura 26 – Circuito do microcontrolador.

No circuito apresentado, foi adicionada uma baterial de 3V, padrão CR2032, ao pino VBat, de forma a possibilitar a utilização do RTC embutido no microcontrolador sem perda das informações de horário mesmo com o sistema desenergizado. Foram incluídos também dois botões do tipo *push button*, para interação com o sistema, além de dois LEDs para sinalização do comportamento do sistema. Isso pode ser visto na Figura 27.

Ainda na Figura 27, é possível observar as conexões com um display que usa o padrão I2C. Trata-se de um display gráfico OLed, baseado no controlador SSD1306 (SOLOMON-SYSTECH, 2008), de 0,91 polegadas com 128x32 pontos, conforme Figura 28.

2.1.5 Comunicação e armazenamento de dados

A Figura 29 apresenta o circuito de comunicação e armazenamento, onde ficam destacadas as conexões com o módulo de WiFi, identificado por U5, com o módulo da interface com o SDCard em J2, além da disponibilização de conexão UART em J3 para eventual *debug*, usado durante o desenvolvimento do *firmware*.

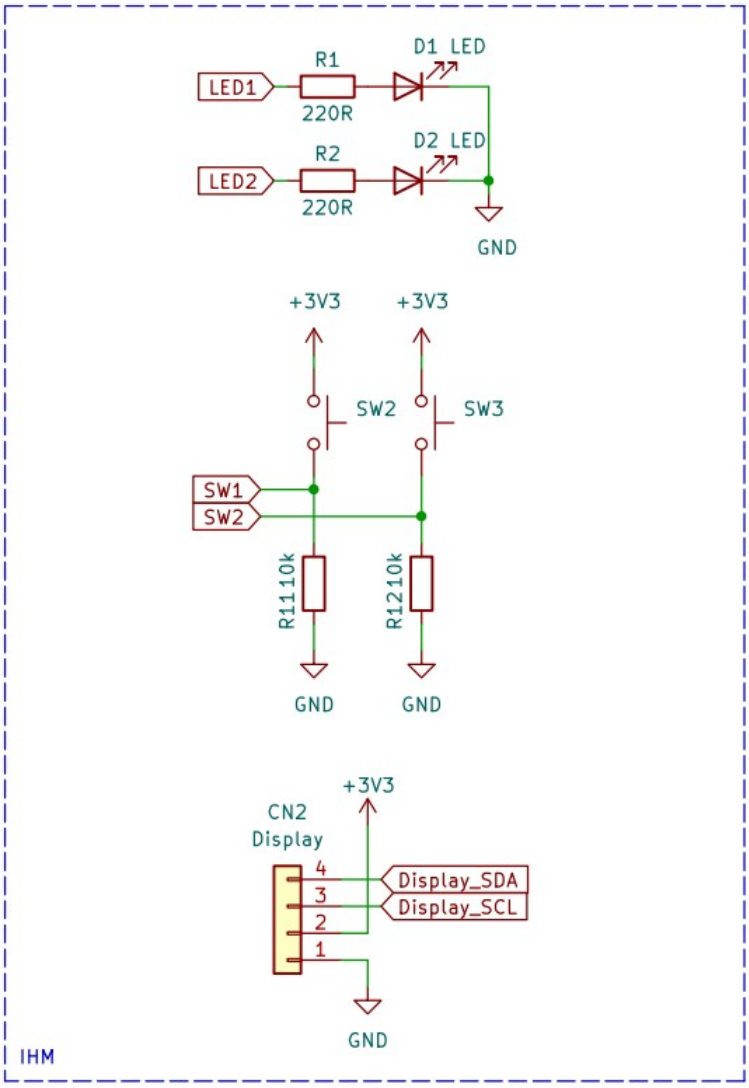


Figura 27 – Circuito da interface com o usuário.

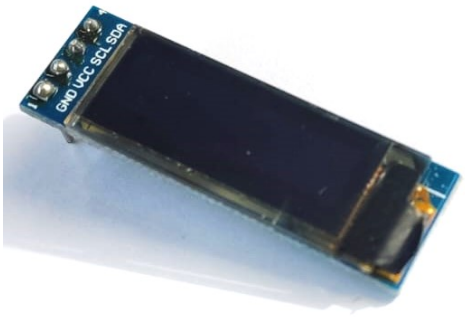


Figura 28 – Módulo do display gráfico OLed de 0,91 polegadas.

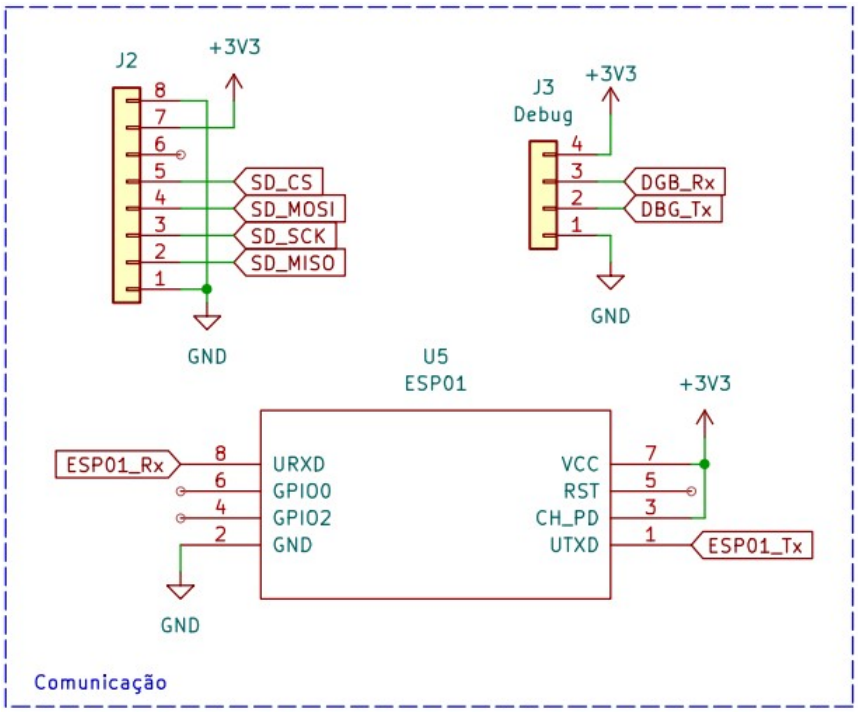


Figura 29 – Circuito de comunicação e armazenamento.

O módulo WiFi escolhido é baseado no ESP8266, fabricado pela Espressif Systems (ESPRESSIF, 2020). Trata-se do módulo ESP01, conforme Figura 30, escolhido devido ao seu baixo custo e simplicidade de utilização. Sua conexão é feita através de uma USART e a comunicação é baseada em comandos AT. Apesar do módulo possuir um microcontrolador e recursos de processamento que poderiam suprir as necessidades do projeto, o uso deste módulo neste projeto ficou limitado a atuação como interface com a internet, de forma a possibilitar o envio dos dados de medição para a nuvem.

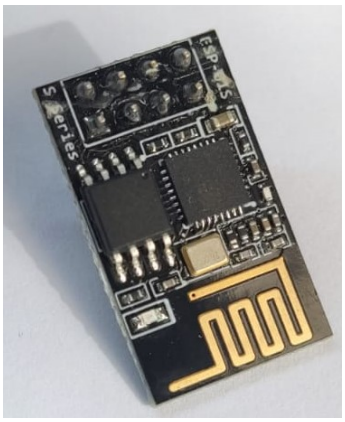


Figura 30 – Módulo WiFi ESP 01.

Por fim, foi incluído um módulo para interface com um SDCard. A ideia é armazenar localmente os dados de medição a fim de aumentar a segurança do sistema em possíveis situações de perda de conexão ou indisponibilidade do servidor. O módulo usado segue o padrão de comunicação SPI, sendo uma versão genérica sem marca especificada, podendo

ser observado na Figura 31.

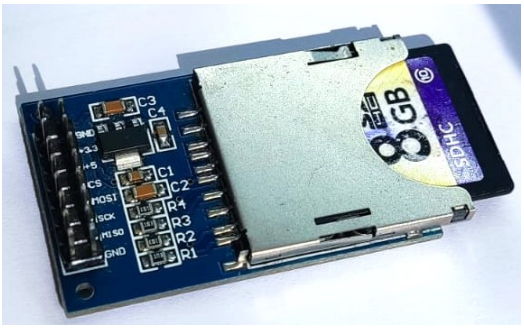


Figura 31 – Módulo SDCard.

2.1.6 Fonte de alimentação da parte lógica

Para a alimentação do microcontrolador e dos circuitos de condicionamento de sinais, foi adotada uma fonte de alimentação chaveada com saída de 12V e 0,42A, correspondendo a 5W de potência. Trata-se da fonte modular HLK-5M12, fabricada pela Hi-Link (HI-LINK, 2019), conforme Figura 32.

Figura 32 – Fonte de 12V Hi-Link HLK-5M12.

Não foi prevista uma proteção contra sobretensão na entrada da fonte de alimentação já que a fonte possui uma faixa aceitável de 100 a 240V em tensão alternada a 60Hz. O propósito do circuito é atuar em um sistema monofásico de 127V/60Hz, o que está muito aquém do limite de entrada da fonte.

A escolha desta fonte de alimentação foi motivada pelo seu encapsulamento compacto, facilitando a sua inclusão na placa, além da tensão de 12V ser necessária ao correto funcionamento dos circuitos de condicionamento, baseados no amplificador operacional LM358. Para alimentação dos demais circuitos, faz-se necessária uma tensão de 3,3V, o que foi obtido através do regulador integrado LM1117-3.3, cujos capacitores C1 e C2 foram adicionados conforme orientações de aplicação do fabricante. O circuito da fonte pode ser visto na Figura 33.

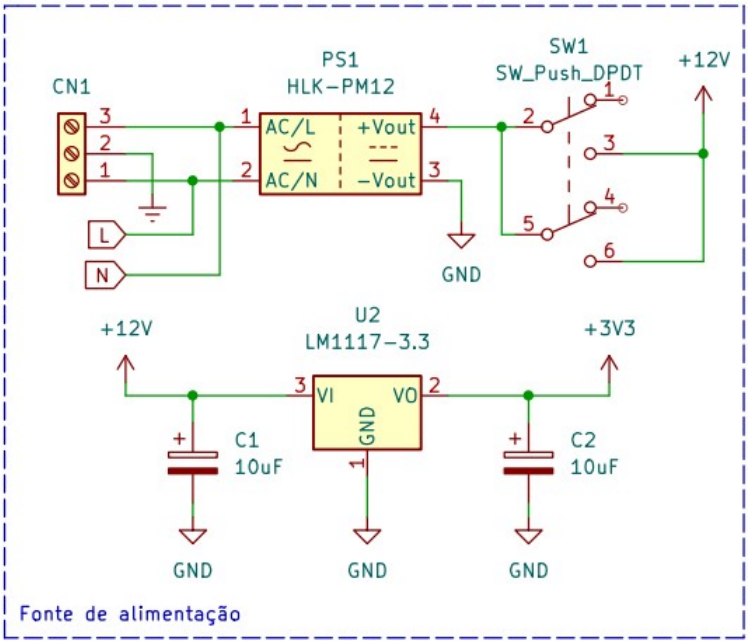


Figura 33 – Circuito da fonte de alimentação.

Vale destacar que a corrente demandada pelo circuito foi estimada através de ensaios práticos com montagem dos circuitos em protoboard, o que motivou a escolha deste modelo cuja corrente máxima de saída atende ao conjunto.

Ainda pela Figura 33, observa-se que foi adicionada uma chave DPDT na saída de 12V para servir de chave geral do sistema de baixa potência.

2.1.7 Hardware do protótipo final

Após testes dos circuitos individuais, foi realizada a montagem do circuito completo em uma matriz de contatos de forma a iniciar a validação e observação do comportamento do circuito completo. A primeira montagem pode ser visualizada na Figura 34. Vale ressaltar que este protótipo foi usado para definição do circuito final e ajuste de alguns valores de componentes, conforme citado em seções anteriores.

Para melhor visualização, o circuito completo do medidor pode ser encontrado no Apêndice A.

Após ajustes e validação da montagem em matriz de contatos, foi desenvolvido o layout da placa de circuito impresso (do inglês PCB - *printed circuit board*) para confecção do protótipo final. Para este trabalho, foi usado o software KiCad EDA versão 7.0. Trata-se de um software Open Source de captura de esquema elétrico e desenvolvimento de projetos de layout de PCBs. A Figura 35 apresenta o layout final à esquerda e o detalhamento da posição dos componentes com suas referências à direita. Na Figura 36 pode-se observar uma projeção em 3D gerada pelo KiCad para previsão mecânica do projeto.

Finalmente, na Figura 37 pode-se observar a placa final já com os componentes

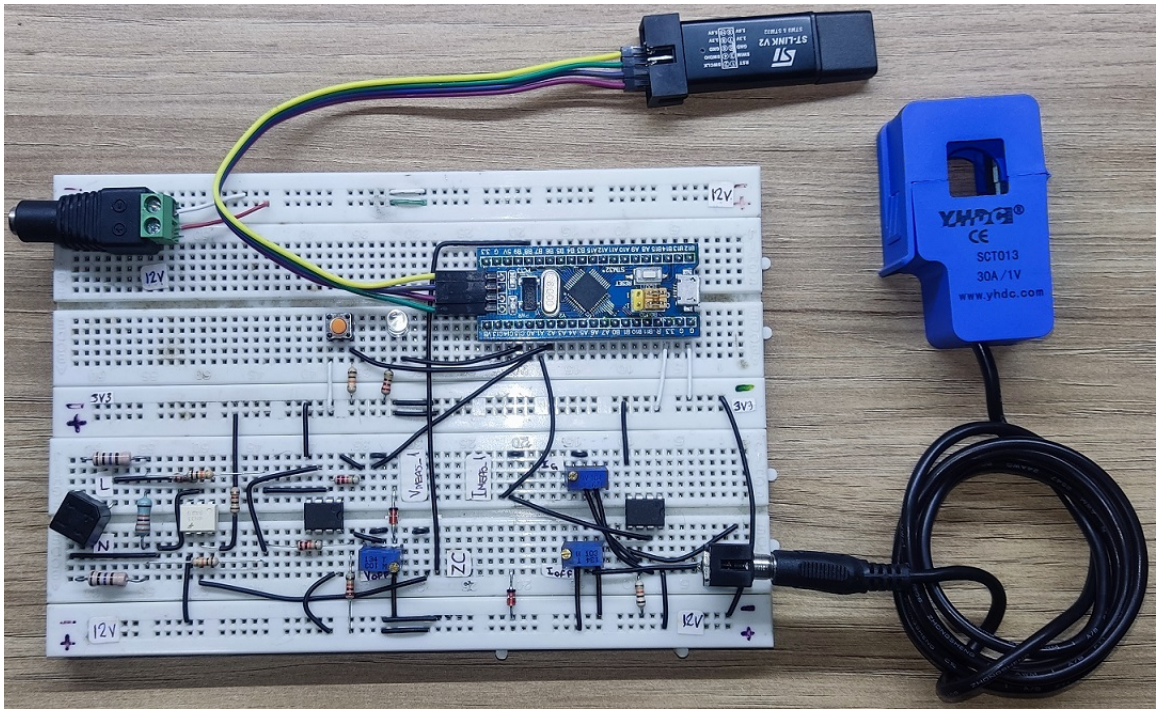


Figura 34 – Montagem em matriz de contatos para validação.

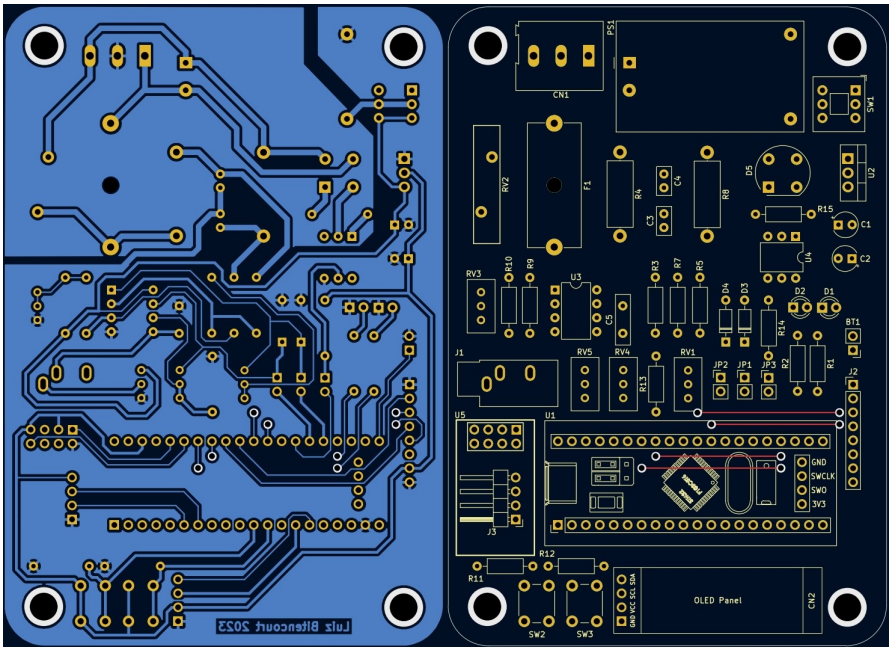


Figura 35 – Layout da PCB do protótipo final.

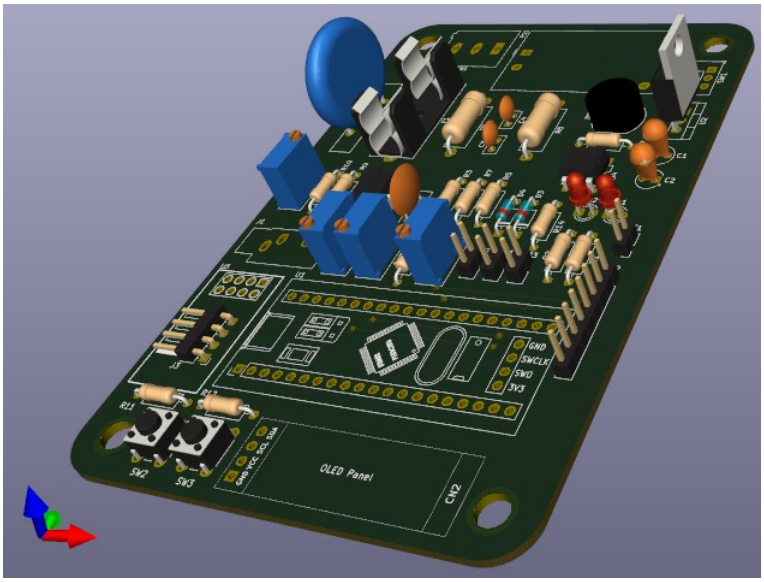


Figura 36 – Previsão em 3D do protótipo final.

instalados. A placa foi confeccionada em uma router CNC de forma a possibilitar a rápida finalização da etapa de montagem do protótipo.

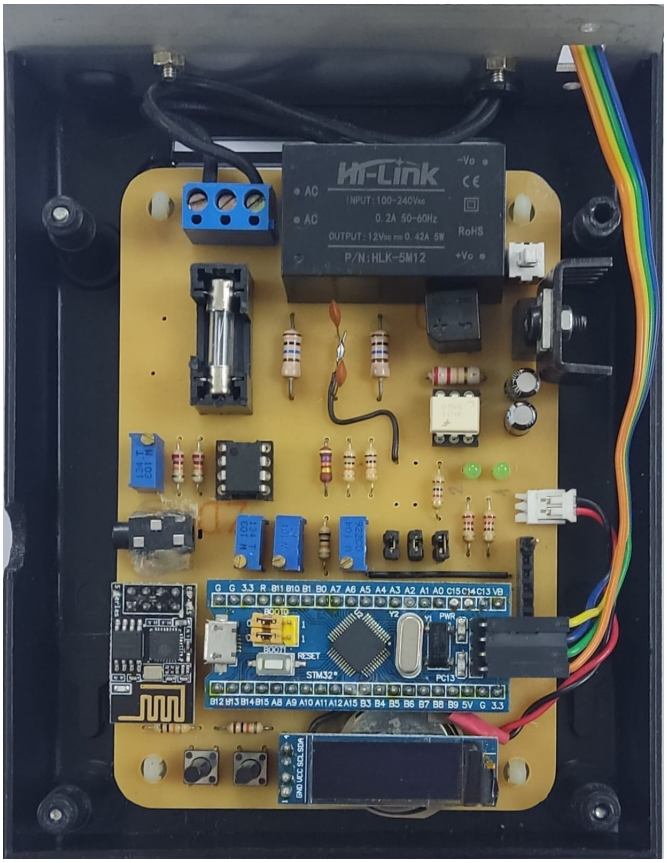


Figura 37 – Protótipo final.

2.2 Desenvolvimento do *firmware*

Com os circuitos de condicionamento de sinais dimensionados e devidamente conectados às entradas analógicas do microcontrolador, além dos demais circuitos conectados, o sistema está pronto para o desenvolvimento do *firmware*.

Todo o desenvolvimento de *firmware* foi realizado no ambiente do STM32 Cube IDE versão 1.12.1. A adoção deste software foi motivada pela sua estrutura de programação facilitada, os recursos de configuração de hardware automatizados, além do fato de ser o ambiente oficial do fabricante do microcontrolador utilizado, a ST Microelectronics. Esta seção apresenta as principais configurações de hardware relacionadas ao projeto de *firmware*, além do detalhamento do desenvolvimento dos algoritmos usados nos cálculos dos parâmetros de interesse.

2.2.1 Configurações de IOs

Antes de iniciar a programação, é realizada a configuração do *hardware* do STM32f103 para que seja gerado um código fonte inicial de forma automatizada. A Figura 38 mostra a configuração dos pinos do microcontrolador de acordo com os recursos utilizados.

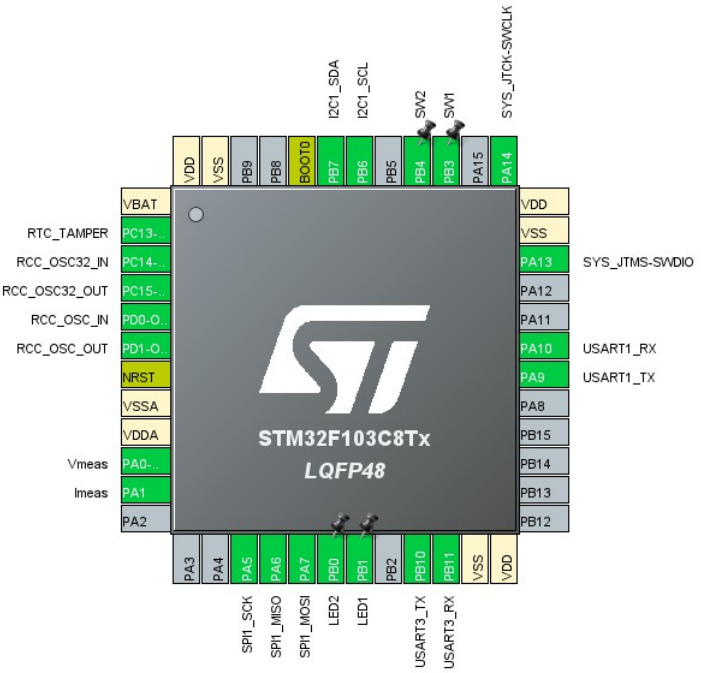


Figura 38 – Configuração do *hardware* do STM32f103.

A Tabela 1 mostra a definição dos pinos usados com seus rótulos e uma rápida descrição.

Em seguida, apresenta-se a configuração de clock na Figura 39. É importante destacar que foram usados os cristais osciladores originais, presentes na placa da BluePill.

Pino	Rótulo	Descrição
PA0	Vmeas	Entrada analógica para medição de tensão
PA1	Imeas	Entrada analógica para medição de corrente
PA5	SPI1_SCK	Pino de clock do SPI (SDCard)
PA6	SPI1_MISO	Pino de entrada de dados do SPI (SDCard)
PA7	SPI1_MOSI	Pino de saída de dados do SPI (SDCard)
PA9	USART1_TX	Pino de transmissão da USART1 (Debug)
PA10	USART1_RX	Pino de recepção da USART1 (Debug)
PA13	SYS_JTMS-SWDIO	Debug Serial Wire
PA14	SYS_JTCK-SWCLK	Debig Serial Wire
PB0	LED2	Saída digital (LED)
PB1	LED1	Saída digital (LED)
PB3	SW1	Entrada digital (botão)
PB4	SW2	Entrada digital (botão)
PB6	I2C1_SCL	Pino de clock do I2C1 (display OLED)
PB7	I2C1_SDA	Pino de dados do I2C1 (display OLED)
PB10	USART3_TX	Pino de transmissão da USART1 (Módulo WiFi)
PB11	USART3_RX	Pino de recepção da USART1 (Módulo WiFi)
PC13	RTC_TAMPER	Pino de Tamper do RTC
PC14	RCC_OSC32_IN	Oscilador externo de 32,768 kHz
PC15	RCC_OSC32_OUT	Oscilador externo de 32,768 kHz
PD0	RCC_OSC_IN	Oscilador externo de 8 MHz
PD1	RCC_OSC_OUT	Oscilador externo de 8 MHz

Tabela 1 – Definições dos pinos utilizados

As frequências dos cristais osciladores são de 32,768 kHz e 8 Mhz, sendo que o primeiro é dedicado ao circuito do relógio em tempo real (RTC).

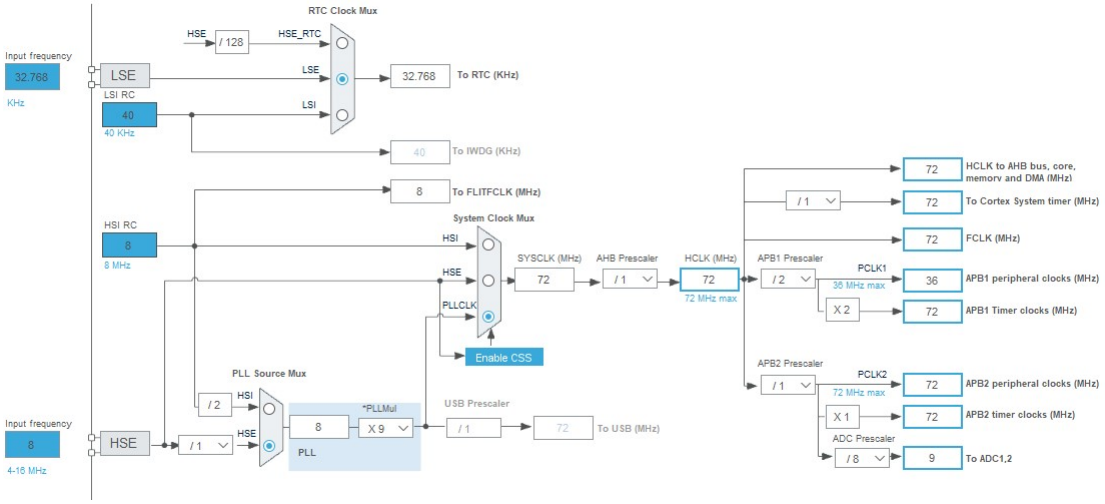


Figura 39 – Configuração do *clock* do STM32f103.

Ainda pela Figura 39, pode-se observar que foi habilitado o recurso de PLL para gerar um clock interno de 72 MHz. Isso se fez necessário para que fosse possível alcançar um clock de 9 MHz no circuito dos ADCs, possibilitando a maior taxa de amostragem possível, conforme será mais detalhado na seção 2.2.3.

2.2.2 Interface com o usuário

Para facilitar a visualização das informações, bem como comandar o acionamento de recursos, foi implementada uma interface com o usuário baseada em uma máquina de estados para exibição de telas diferentes no display OLED, de acordo com os comandos dos botões SW1 e SW2 (como definidos na seção anterior). Isso permitiu exibir os dados instantâneos calculados, dados do RTC, ativar ou desligar o SDCard ou a internet. O mapa das telas pode ser visualizado na Figura 40.

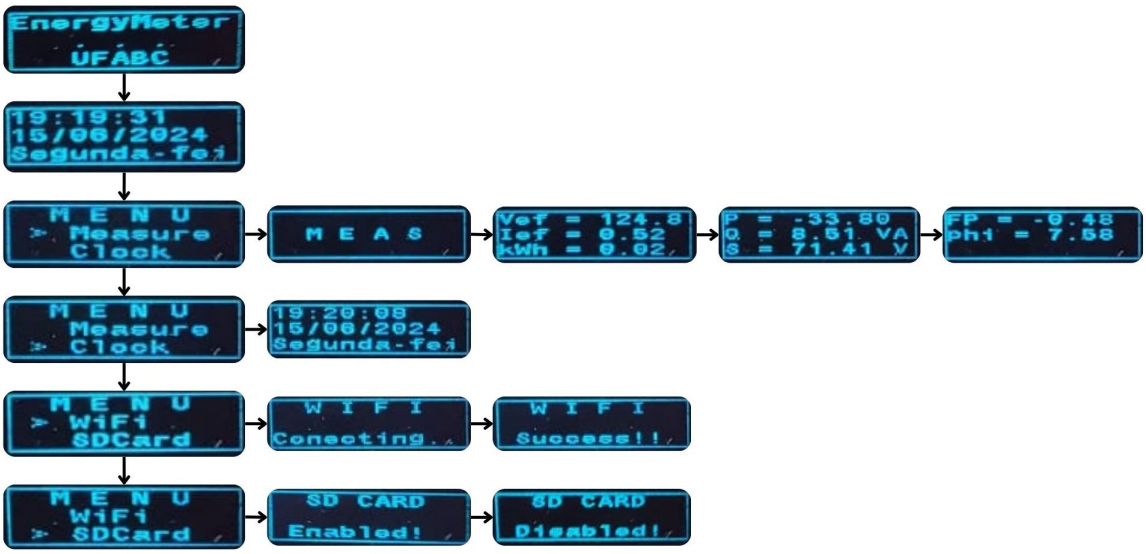


Figura 40 – Mapa de telas do sistema.

O menu “Measure” permite exibir: os valores instantâneos da tensão e corrente eficazes; o valor acumulado de energia elétrica consumida; os valores das potências ativa, reativa e aparente; além do valor do ângulo de defasagem entre tensão e a corrente e seu cosseno (fator de potência). O menu “Clock” exibe data, hora e dia da semana. O menu “WiFi”, no primeiro acesso, inicia o processo de conexão com uma rede WiFi previamente configurada, e, no próximo acesso, realiza a desconexão. O menu “SDCard”, no primeiro acesso, habilita o módulo, e, no acesso seguinte, o desabilita.

2.2.3 Configurações dos módulos ADC

As configurações dos módulos ADC seguiram as recomendações do documento AN3116 (STMICROELECTRONICS, 2024a), que é um conjunto de notas de aplicação fornecido pelo fabricante do microcontrolador.

Os conversores foram configurados para atuação em *dual regular simultaneous mode*, com o recurso DMA ativado. Dessa forma, as conversões acontecem de maneira simultânea e os dados originados da conversão são diretamente armazenados na memória em um único vetor de 32 bits. As conversões são disparadas por software, sendo que o

sistema gera interrupções quando os dados são transferidos para a memória. Isso facilita a programação e não prende o processador nesta tarefa.

Para obtenção do tempo de conversão (TCONV) do microcontrolador, as notas de aplicação mostram a Equação 2.6.

$$TCONV = \frac{12,5 + SAMPLINGTIME}{Fadc} \tag{2.6}$$

Conforme mostrado na Figura 39, o clock interno foi configurado de forma a possibilitar o maior valor de clock para os módulos ADC, que foi de 9 MHz. A ideia é obter o menor tempo possível de conversão, possibilitando o máximo de conversões, isto é, uma maior taxa de amostragem. O parâmetro *sampling time* do microcontrolador foi configurado para 239,5 ciclos para que as leituras de cada amostra tivessem um tempo adequado de estabilização, melhorando sua precisão. Usando estes parâmetros na Equação 2.6, tem-se:

$$TCONV = \frac{12,5 + 239,5}{9 \cdot 10^6} = 28\mu s$$

Considerando-se que o sinal da rede elétrica tem uma frequência de $60Hz$, o que corresponde a um período de $16,67ms$, é possível coletar cerca de 595 amostras do sinal de 1 ciclo considerando o tempo de conversão de $28\mu s$.

2.2.4 Configurações dos canais de comunicação serial

Para este projeto foram configurados dois canais de comunicação USART (USART1 e USART3); um canal de comunicação SPI e um I2C. Os canais USART foram usados para comunicação com o módulo WiFi e debug. O canal I2C possibilitou a utilização do display OLED. Finalmente, o canal SPI foi usado com o módulo SDCard.

Os canais USART foram configurados no modo assíncrono, com baud rate de 115200 bps, 8 bits de largura de palavra (sem paridade) e 1 bit de parada. Além disso, foram habilitadas as interrupções para este recurso de comunicação. Um dos canais, USART3, é usado com o módulo WiFi ESP01, possibilitando o envio de dados à nuvem. O segundo canal, USART1, foi disponibilizado para possibilitar a realização de debug, durante as etapas de desenvolvimento do firmware para testes das bibliotecas de comunicação.

2.2.5 Processamento dos sinais de tensão e corrente elétrica

Com os circuitos dimensionados e devidamente conectados às entradas analógicas do microcontrolador, deve-se realizar o tratamento dos dados a fim de vincular seu significado com a grandeza que está sendo medida. Descreve-se aqui, portanto, todos os passos seguidos desde a coleta dos dados até sua apresentação.

2.2.5.1 Obtenção dos dados dos ADCs

O microcontrolador foi configurado de forma a realizar leituras dos dois módulos ADC com transferência direta dos dados de medição à memória interna. Trata-se de um recurso presente no STM32 chamado DMA (do inglês, *Direct Memory Access*), cujas notas de aplicação estão disponíveis no documento AN2548 (STMICROELECTRONICS, 2010), fornecido pela ST Microelectronics. Neste contexto, tem-se um melhor aproveitamento do processamento, já que não há necessidade de manipulação direta dos dados dos registradores do ADC, sem ocupação desnecessária do núcleo. Dessa forma, optou-se por realizar medições uma vez por segundo, sendo que no intervalo entre essas medições fosse possível realizar todos os cálculos de interesse deste trabalho.

O recurso do DMA armazena um vetor com os dados dos dois módulos ADC de maneira concatenada. Faz-se necessária a separação dos dados, que pode ser verificada no trecho de código listado a seguir:

```
1 for (int i = 0; i < SAMPLES; i++) {
2         adcVoltage[i] = adcRaw[i] & 0x0000ffff;
3         adcCurrent[i] = adcRaw[i] >> 16;
4 }
```

Basicamente, os dados oriundos dos módulos ADC são copiados do vetor `adcRaw[]` para os vetores `adcVoltage[]`, usando uma máscara para que sejam copiados apenas os primeiros 16 bits, e `adcCurrent[]`, cujos dados estão nos 16 bits mais significativos de `adcRaw` exigindo um deslocamento de bits.

2.2.5.2 Ajuste dos dados de tensão e corrente

Os valores armazenados até agora são resultado das conversões, o que faz com que seus valores possam assumir a faixa de 0 a 4095. Faz-se necessário um ajuste nos valores para correta correspondência com os valores de tensão e corrente medidos. Para tanto, usou-se o trecho de código a seguir que ajusta a escala dos valores baseado na máxima tensão de entrada de $3.3V$, máximo valor convertido de 4095, offset (`voff` e `ioff`) e ganho (`tp` e `tc`). Estes dois últimos parâmetros foram ajustados em ensaios práticos usando-se entradas de sinais controlados e com valores conhecidos.

```
1 for (int i = 0; i < SAMPLES; i++) {
2         sampleVoltage[i] = (((adcVoltage[i]) * (3.3 / 4095)) - voff) * tp;
3         sampleCurrent[i] = (((adcCurrent[i]) * (3.3 / 4095)) - ioff) * tc;
4 
5         if (sampleVoltage[i] > vpp) vpp = sampleVoltage[i];
6         if (sampleVoltage[i] < vpn) vpn = sampleVoltage[i];
```

```
7         if (sampleCurrent[i] > ipp) ipp = sampleCurrent[i];
8         if (sampleCurrent[i] < ipn) ipn = sampleCurrent[i];
9 }
```

Nota-se, pelo código apresentado, que as linhas 5 a 8 atualizam as variáveis `vpp`, `vpn`, `ipp` e `ipn`. Tais variáveis correspondem aos valores de pico positivos e negativos dos sinais de tensão e corrente. Estes dados foram armazenados para auxiliar no processo de calibração dos parâmetros `voff`, `ioff`, `tp` e `tc`.

Vale destacar também que o sinal de corrente deve sofrer um ajuste de fase, a fim de compensar a defasagem causada pela natureza do transdutor, que é indutivo. Isso é feito pelo trecho de código a seguir, executado logo após os ajustes mostrados anteriormente.

```
1     for (int i = 0; i < SAMPLES; i++) {
2                 if (i < SAMPLES - iphase) sampleCurrent[i] =
  ↪   sampleCurrentDephased[i + iphase];
3                 else if (i >= SAMPLES - iphase) sampleCurrent[i] =
  ↪   sampleCurrentDephased[i - (SAMPLES - iphase)];
4         }
```

Nota-se, pelo código apresentado, que a correção de fase é baseada em um *offset* de fase, gerado pela variável `iphase`, podendo assumir um valor entre 0 e o número máximo de amostras, gerando um deslocamento do vetor `sampleCurrentDephased[i]`. Para tal ajuste, deve-se usar uma carga puramente resistiva, coletar dados de tensão e corrente, analisar a defasagem entre os sinais e, finalmente, estimar o valor de `iphase`.

Neste trabalho, todos os ajustes de parâmetros foram realizados no modo de *Debug* no STM32 Cube IDE, atuando em tempo real sobre as variáveis de calibração, conforme se observam os valores das medições realizadas. No trecho abaixo pode-se verificar a atribuição de valores aos parâmetros de calibração, os quais foram obtidos no processo citado.

```
1 float tp = 168;
2 float voff = 1.789;
3 float tc = 2.69;
4 float ioff = 1.498;
5 int iphase = 275;
```

Para ajuste dos dados de tensão, atuou-se em `tp` e `voff`, que são as variáveis de calibração para ajustar o ganho e o *offset* de fase do sinal obtido. Estes ajustes foram guiados pelos valores de pico positivo e negativo da senoide da tensão, além do seu valor eficaz, cujos valores aplicados são conhecidos, conforme Figura 41. Sendo assim, as variáveis

de calibração da tensão foram ajustados de forma que o valor eficaz calculado ficasse o mais próximo possível de 127V, bem como os picos positivos de 179,6V e negativos de -179,6V.

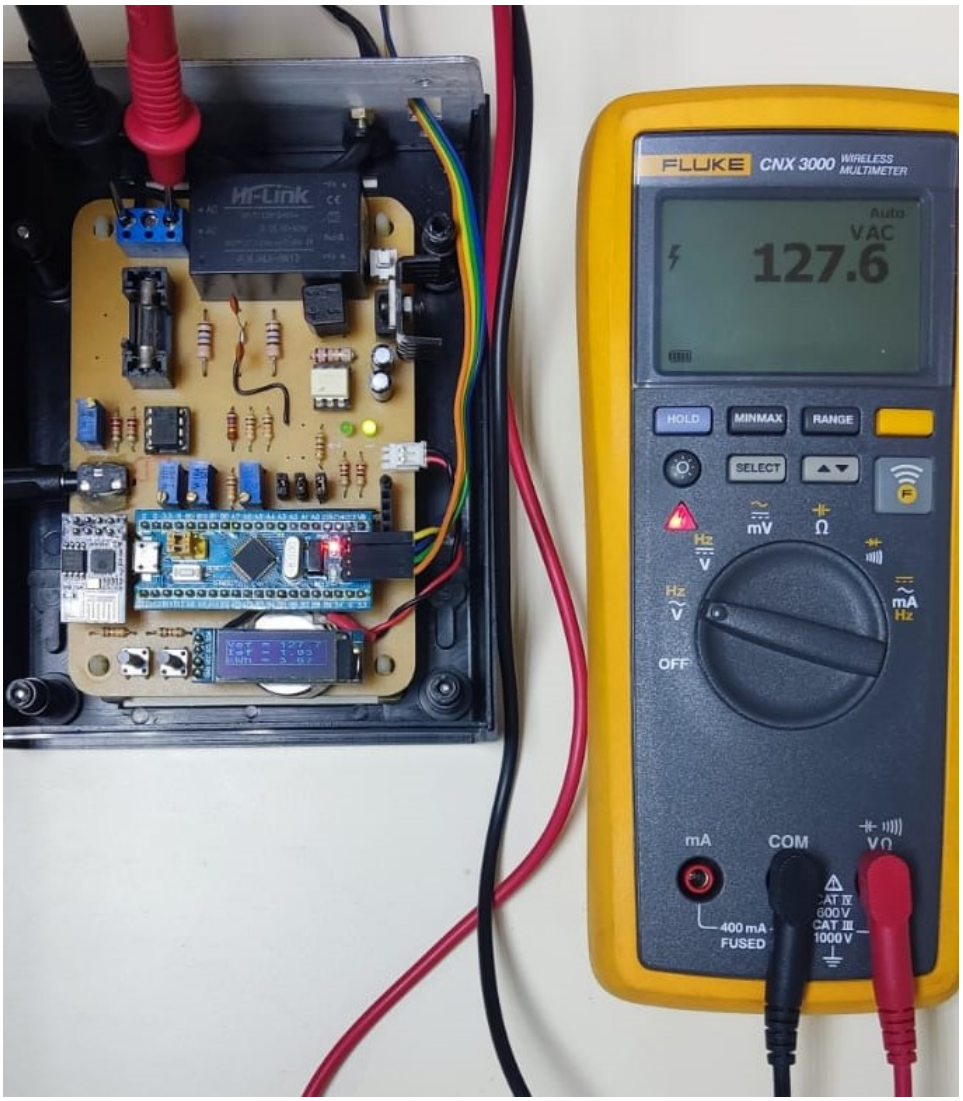


Figura 41 – Validação do ajuste de tensão com o Fluke CNX3000.

De maneira análoga, para ajuste dos dados de corrente, atuou-se em `tc` e `ioff`. Estes ajustes foram guiados pelos valores de pico positivo e negativo da senoide da corrente, além do seu valor eficaz, cujos valores aplicados são conhecidos, conforme Figura 42. Utilizou-se uma carga puramente resistiva com demanda de corrente de 1A eficaz. Sendo assim, as variáveis de calibração da corrente foram ajustados de forma que o valor eficaz calculado ficasse o mais próximo possível de 1A, bem como os picos positivos de 1,41A e negativos de -1,41V. Considerando-se a natureza indutiva do transdutor de corrente, o ajuste de fase foi realizado atuando-se sobre a variável `iphase` e observando o valor calculado de `phi` de forma que ele ficasse o mais próximo possível de zero, quando se encontra o valor de `iphase`. É importante destacar que a carga resistiva demandou uma corrente de aproximadamente 200mA, o que motivou a adaptação de 5 voltas do condutor no transdutor de corrente, a

fim de se obter um valor maior de corrente medida (cerca de 1A), melhorando a qualidade dos dados obtidos.

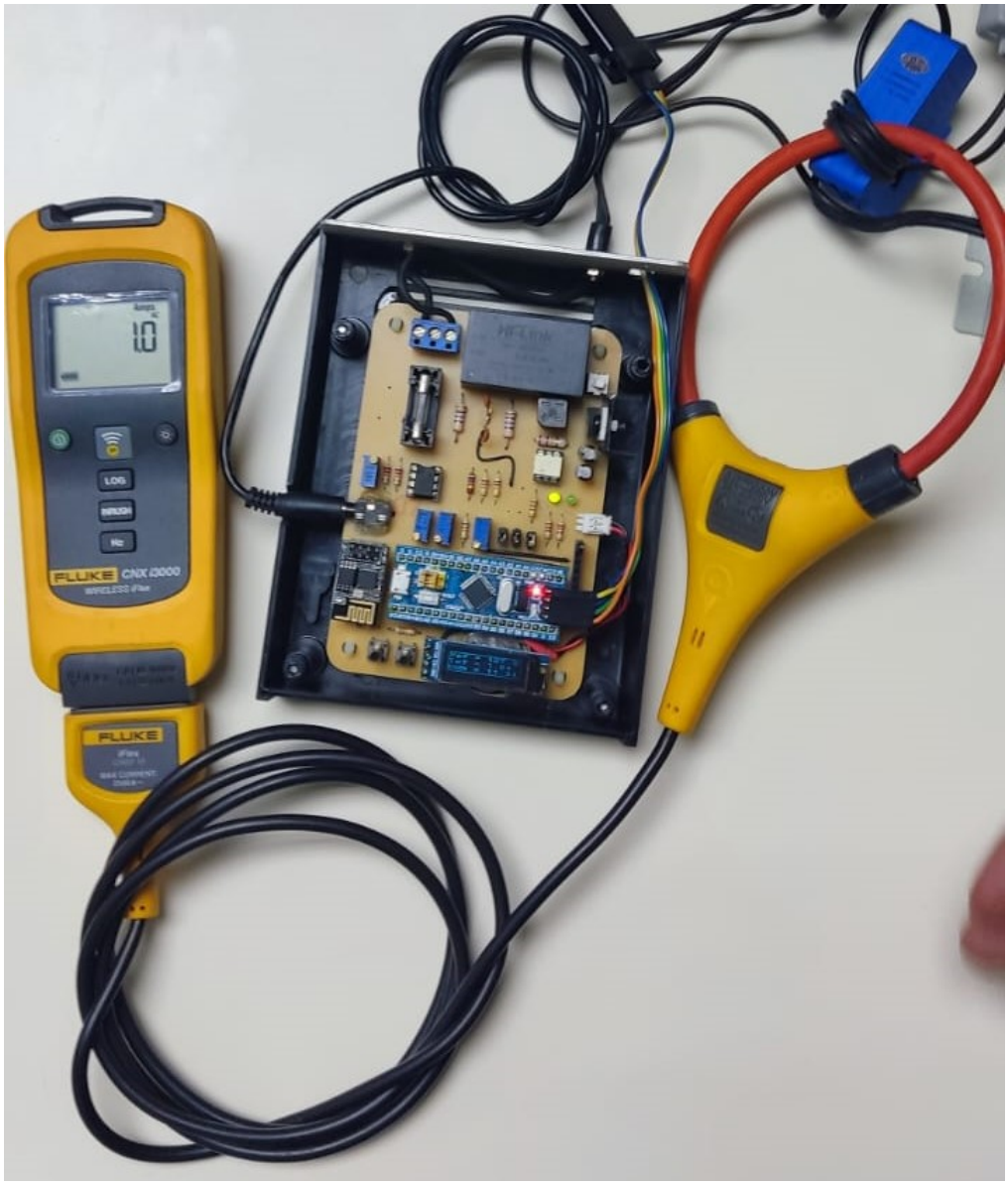


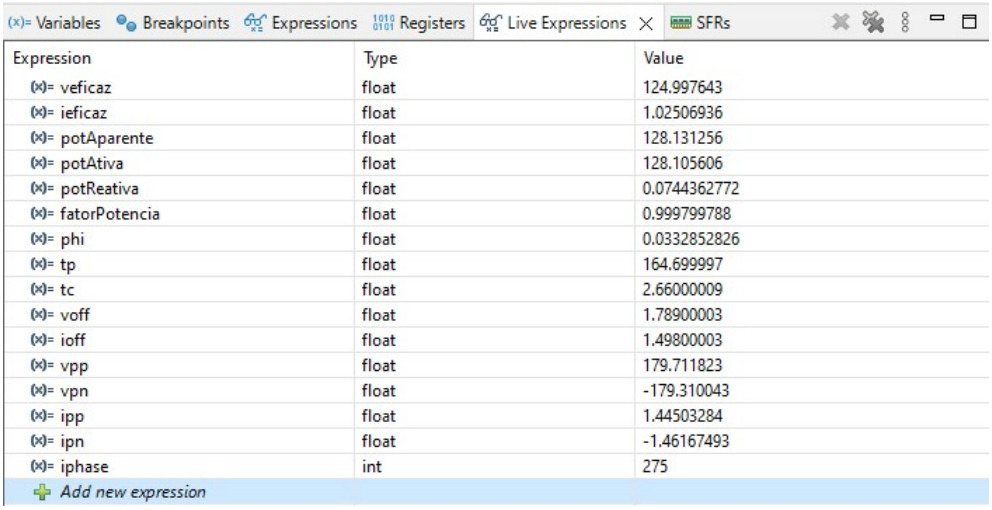
Figura 42 – Validação do ajuste de corrente com o Fluke CNXi3000.

A Figura 43 mostra a tabela de *Live Expressions*, vista na perspectiva de *debug* do STM32 Cube IDE durante o processo de ajustes das variáveis de calibração dos sinais de tensão e corrente. O uso deste recurso do ambiente de programação é descrito no manual de usuário UM2609, fornecido pelo fabricante (STMICROELECTRONICS, 2024b).

Vale a observação de que a descrição dos algoritmos usados na obtenção dos parâmetros mostrados na Figura 43 é apresentada nas seções seguintes deste tópico.

2.2.5.3 Cálculo do valor eficaz

Para o cálculo do valor eficaz, conforme Turgel (1974), utiliza-se a Equação 2.7:



(x)= Variables | Breakpoints | Expressions | Registers | Live Expressions | SFRs

Expression	Type	Value
(x)= veficaz	float	124.997643
(x)= ieficaz	float	1.02506936
(x)= potAparente	float	128.131256
(x)= potAtiva	float	128.105606
(x)= potReativa	float	0.0744362772
(x)= fatorPotencia	float	0.999799788
(x)= phi	float	0.0332852826
(x)= tp	float	164.699997
(x)= tc	float	2.66000009
(x)= voff	float	1.78900003
(x)= ioff	float	1.49800003
(x)= vpp	float	179.711823
(x)= vpn	float	-179.310043
(x)= ipp	float	1.44503284
(x)= ipn	float	-1.46167493
(x)= iphase	int	275
Add new expression		

Figura 43 – Tabela de *Live Expressions* do STM32 Cube IDE durante o *debug*.

$$Y_{rms} = \sqrt{\frac{1}{N}\sum_{i=0}^{N-1} x[i]^2} \tag{2.7}$$

onde o valor RMS calculado Y_{rms} depende no número de amostras N e cujos valores são armazenados em um vetor x. A função `getRMS()`, listada abaixo, é responsável pela realização deste cálculo.

```
1 float getRMS(float signal[], int samples) {
2         float integral = 0;
3 
4         for (int i = 0; i < samples; i++) {
5                 integral += (signal[i] * signal[i]);
6         }
7 
8         return sqrt(integral/samples);
9 }
```

Tal função foi criada para facilitar o cálculo do valor eficaz, podendo ser aplicada nos dados de tensão e de corrente elétrica, obtidos do conversor analógico digital do microcontrolador.

2.2.5.4 Cálculo das potências

Em um sistema de alimentação alternada onde as cargas não são puramente resistivas, tem-se três potências: a ativa, dada pela parcela resistiva da carga; a reativa, dada pela parcela indutiva e capacitiva; e a aparente, correspondendo à soma vetorial das anteriores. Desta forma, faz-se necessário o cálculo separado de tais potências. As equações listadas a seguir foram introduzidas por Paula (2000).

Para o cálculo da potência ativa, utiliza-se a Equação 2.8:

$$P = \frac{1}{N} \sum_{i=0}^{N-1} v[i] \cdot i[i] \tag{2.8}$$

onde a potência ativa P é baseada no somatório do produto de cada amostra de tensão $v[i]$ com cada amostra de corrente $i[i]$. Ao final, divide-se o valor calculado pelo total de amostras N. A função `getActivePower()`, listada abaixo, é responsável pela realização deste cálculo.

```
1 float getActivePower(float voltage[], float current[], int samples) {
2         float integral = 0;
3 
4         for (int i = 0; i < samples; i++) {
5                 integral += voltage[i] * current[i];
6         }
7 
8         return (integral/samples);
9 }
```

Para o cálculo da potência reativa, utiliza-se a estratégia de defasar a corrente em 90 graus em relação aos dados originais, conforme a Equação 2.9:

$$I_{\text{offset}}[i] = \begin{cases} I[i + \frac{N}{4}] & \text{se } i < \frac{3N}{4} \\ I[i - \frac{3N}{4}] & \text{se } i \geq \frac{3N}{4} \end{cases} \tag{2.9}$$

para, posteriormente, aplicar na Equação 2.10:

$$Q = \frac{1}{N} \sum_{i=1}^{N} V_i \cdot I_{\text{offset}}[i] \tag{2.10}$$

cuja função `getReactivePower()`, listada abaixo, é responsável pela realização do cálculo.

```
1 float getReactivePower(float voltage[], float current[], int samples) {
2         float offset[samples];
3         float integral = 0;
4 
5         for (int i = 0; i < samples; i++) {
6                 if (i < ((samples/4)*3)) offset[i] = current[i +
  ↪  (samples/4)];
7                 else if (i >= ((samples/4)*3)) offset[i] = current[i -
  ↪  ((samples/4)*3)];
```

```
        }

        for (int i = 0; i < samples; i++) {
                integral += voltage[i] * offset[i];
        }

        return integral/samples;;
}
```

Finalmente, a potência aparente pode ser obtida pela simples multiplicação dos valores eficazes de tensão e corrente, conforme Equação 2.11.

$$S = v_{rms} \cdot i_{rms} \tag{2.11}$$

cuja função `getApparentPower()`, listada abaixo, é responsável pela realização do cálculo.

```
float getApparentPower(float vrms, float irms) {
        return vrms*irms;
}
```

Além das potências, pode-se também obter o valor do ângulo de defasagem entre a tensão e a corrente, bem como o seu cosseno, o chamado fator de potência. A Equação 2.12 realiza o cálculo do ângulo φ de defasagem entre a tensão e a corrente:

$$\varphi = \cos^{-1}\left(\frac{Q}{S}\right) \tag{2.12}$$

cuja função `getPhi()`, listada abaixo, é responsável pela realização do cálculo.

```
float getPhi(float reactivePower, float apparentPower) {
        return (asinf(reactivePower / apparentPower)) * (180 / M_PI);
}
```

A Equação 2.13 realiza o cálculo do fator de potência:

$$FP = \frac{Q}{S} \tag{2.13}$$

cuja função `getPowerFactor()`, listada abaixo, é responsável pela realização do cálculo.

```
float getPowerFactor(float activePower, float apparentPower){
        return activePower/apparentPower;
}
```

2.2.5.5 Cálculo da energia elétrica consumida

Para o cálculo da energia elétrica, pode-se partir da definição do conceito como na Equação 2.14:

$$E = S \cdot t \tag{2.14}$$

cujo tempo é dado em segundos. Sabendo-se que o sistema foi programado para calcular os parâmetros em intervalos regulares de 1 segundo, a energia instantânea é definida por $E(t) = S(t)$. Disso, pode-se considerar o acumulado de potência consumida como a somatória dos valores obtidos em intervalos de tempo regulares de 1 segundo. Além disso, para que o valor seja apresentado em VAh (Volt Ampère hora), deve-se dividir o tempo por 3600. Sendo assim, o cálculo da energia acumulada pode ser dado pela Equação 2.15:

$$E = \sum_{t=0}^{\infty} S[t] \cdot \frac{1}{3600} \tag{2.15}$$

cuja implementação em código é apresentada no trecho abaixo:

```
1 energia = energia + (potAparente) / 3600;
```

Ainda na Equação 2.15, a notação "t = 0" indica que a somatória inicia quando o sistema de medição é ativado, e permanece em constante atualização enquanto o sistema permanecer em medição.

A listagem do código principal do firmware pode ser verificada no Apêndice B.

2.3 Apresentação dos dados na nuvem

O sistema de medição conta com um módulo de comunicação WiFi para envio de dados à nuvem, conforme descrito anteriormente. Para armazenamento e disponibilização destes dados, foi utilizada a plataforma de IoT ThingSpeak[2], que disponibiliza sua API de forma gratuita, com algumas limitações. Para envio de dados à plataforma, basta criar um vetor de dados, com o número de campos que se deseja publicar: cada campo é um parâmetro do sistema de medição. O comando abaixo é realizado pelo sistema de medição, uma vez estabelecida a conexão com a internet:

```
GET https://api.thingspeak.com/update?api_key=[CHAVE_DA_API]&field1=[VALOR]
```

A plataforma ThingSpeak permite acrescentar até 8 campos, com envio de dados em um intervalo mínimo de 15s. Estas são as limitações do plano gratuito disponibilizado

[2] Disponível em https://thingspeak.com (acesso em agosto de 2024).

pela plataforma e adotado para realização dos testes deste trabalho. Para enviar vários campos simultaneamente, basta configurar a plataforma e adicionar `&fieldX=[VALOR]` ao endereço do GET apresentado anteriormente. No termo fieldX, o "X" deve ser substituído pelo número do campo, variando de 1 a 8.

Vale observar que as limitações de intervalo de tempo entre envios de dados e número de campos não representaram nenhuma barreira para este trabalho, visto que os dados visualizados apresentam o acumulado de consumo de energia elétrica, não havendo perdas.

Para visualização dos dados armazenados na plataforma, o ThingSpeak apresenta uma aplicação web que possibilita uma visualização gráfica. Existe também a possibilidade de se criar uma aplicação em outro servidor, consumindo os dados disponibilizados pela API do ThinkSpeak. Esta última foi a opção escolhida com o objetivo de possibilitar a personalização da forma como os dados são apresentados. Desta forma, foi desenvolvida uma aplicação web em NodeJS, usando a plataforma Glitch[3] para consumir os dados do ThingSpeak e apresentá-los em forma de três gráficos: variação da tensão eficaz; variação da corrente eficaz; e, o acumulado do consumo de energia elétrica.

Para que a aplicação web consuma os dados da API do ThingSpeak, a plataforma disponibiliza o seguinte comando:

```
GET https://api.thingspeak.com/channels/[PUBLIC_CHANNEL_ID]/feeds.json?results=X
```

onde o campo `[PUBLIC_CHANNEL_ID]` configura a identificação do canal público da conta utilizada e, o campo `X`, a quantidade de resultados a serem exibidos. O resultado do comando é a apresentação de um JSON com todos os campos gerados e seus respectivos dados na quantidade solicitada. No trecho abaixo, pode-se visualizar um exemplo de uma dessas solicitações para 1 resultado de medição dos 7 campos enviados à plataforma:

```
{
"channel": {
  "id": #######,
  "name": "Medidor de energia",
  "latitude": "0.0",
  "longitude": "0.0",
  "field1": "Tensão eficaz",
  "field2": "Corrente eficaz",
  "field3": "Energia",
  "field4": "Potência Ativa",
  "field5": "Potência Reativa",
```

[3] Disponível em https://glitch.com (acesso em agosto de 2024).

```
    "field6": "Potência Aparente",
    "field7": "Fator de Potência",
    "created_at": "2024-05-30T18:20:28Z",
    "updated_at": "2024-06-15T00:39:39Z",
    "last_entry_id": 6028
  },
  "feeds": [
    {
      "created_at": "2024-08-03T00:45:51Z",
      "entry_id": 6027,
      "field1": "127.546928",
      "field2": "0.317575",
      "field3": "115.604477",
      "field4": "12.327680",
      "field5": "-1.840486",
      "field6": "40.505684",
      "field7": "0.304344"
    }
  ]
}
```

O código a seguir apresenta a aplicação, do lado do servidor, cuja tarefa é buscar os dados coletados pela API do ThingSpeak. Vale destacar que os dados de data e hora foram ajustados para melhor formato de exibição.

```
const express = require('express');
const fetch = require('node-fetch');
const path = require('path');
const moment = require('moment-timezone');

const app = express();
const PORT = process.env.PORT || 3000;
const THINGSPEAK_API_URL =
↪  "https://api.thingspeak.com/channels/#######/feeds.json?results=50";

app.get('/', (req, res) => {
  res.sendFile(path.join(__dirname, '/views/index.html'));
});

app.get('/data', async (req, res) => {
  try {
```

```
16      const response = await fetch(THINGSPEAK_API_URL);
17      const data = await response.json();
18
19      data.feeds = data.feeds.map(feed => {
20        feed.created_at =
          ↪  moment.utc(feed.created_at).tz('America/Sao_Paulo').format('MM/DD/YYYY
          ↪  HH:mm:ss');
21        return feed;
22      });
23
24      res.json(data);
25    } catch (error) {
26      res.status(500).send('Erro ao buscar dados do ThingSpeak');
27    }
28  });
29
30  app.listen(PORT, () => {
31    console.log(`Servidor rodando na porta ${PORT}`);
32  });
```

O código abaixo roda no lado do cliente, possibilitando a disbonibilização dos dados para posterior exibição em forma de gráficos, bem como sua atualização automática a cada 15 segundos.

```
1   async function fetchData() {
2     const response = await fetch("/data");
3     const data = await response.json();
4     updateTable(data);
5   }
6
7   function updateTable(data) {
8     const tableBody = document.getElementById("data-table-body");
9     tableBody.innerHTML = ""; // Limpar a tabela
10
11    data.feeds.forEach((feed) => {
12      const row = document.createElement("tr");
13
14      const dateCell = document.createElement("td");
15      dateCell.textContent = feed.created_at;
16      row.appendChild(dateCell);
17
```

```
    const field1Cell = document.createElement("td");
    field1Cell.textContent = feed.field1;
    row.appendChild(field1Cell);

    const field2Cell = document.createElement("td");
    field2Cell.textContent = feed.field2;
    row.appendChild(field2Cell);

    const field3Cell = document.createElement("td");
    field3Cell.textContent = feed.field3;
    row.appendChild(field3Cell);

    const field4Cell = document.createElement("td");
    field4Cell.textContent = feed.field4;
    row.appendChild(field4Cell);

    tableBody.appendChild(row);
  });
}

async function pollForUpdates() {
  try {
    const response = await fetch("/poll");
    const data = await response.json();
    updateTable(data);
  } catch (error) {
    console.error("Erro ao buscar atualizações:", error);
  } finally {
    setTimeout(pollForUpdates, 15000); // Tentar novamente em 15 segundos
  }
}

window.onload = () => {
  fetchData();
  pollForUpdates();
};
```

O código HTML da página principal que apresenta os dados em forma de gráficos é apresentado no Apêndice C. Vale destacar que neste ponto foi usada a API do Google Charts, facilitando o processo de desenho e estilização dos gráficos.

3 Resultados e Discussão

Neste capítulo é apresentada toda informação gerada pelos ensaios realizados com o medidor de energia elétrica. Para melhor organização, o capítulo foi dividido nas seguintes seções para abordar os ensaios individuais realizados em cada circuito, bem como os ensaios de coleta de dados de tensão e corrente, seus parâmetros, além do consumo de energia.

3.1 Condições e equipamentos

Os ensaios iniciais foram realizados em um *protoboard* pela praticidade em redimensionar circuitos e ajustar parâmetros. A montagem pode ser visualizada na Figura 34, apresentada na Seção 2.1.7 deste trabalho. Após os ensaios de validação, foi projetado e confeccionado um circuito eletrônico dedicado que originou as medições apresentadas neste capítulo.

Vale ressaltar que, para as medições realizadas, foram usados os seguintes equipamentos/instrumentos:

- Osciloscópio digital de 60MHz e 400MSa/s, modelo 2534 da BK Precision;
- Multímetro digital CNX3000 da Fluke;
- Amperímetro digital CNXi3000 da Fluke;

3.2 Ensaios com o circuito *zero crossing*

Com o circuito completo montado, foi realizada a medição do sinal de referência de cruzamento por zero, o *zero crossing*. De acordo com a topologia do circuito, um pulso de curta duração é gerado a cada cruzamento por zero da senoide de tensão da rede elétrica. As Figuras 44 e 45 exibem o sinal bem como os dados de medição realizados pelo o osciloscópio.

Nota-se que o sinal possui um período de 119,9 Hz, correspondente aos dois cruzamentos que a senoide realiza em um período de oscilação. Além disso, o valor de pico do sinal é de 2,40 V, o que atende aos requisitos de leitura das entradas digitais do microcontrolador utilizado.

Este circuito foi incluído no projeto com a intenção de possibilitar o cálculo da frequência do sinal da rede elétrica, bem como facilitar o sincronismo dos sinais. Conforme o desenvolvimento metodológico apresentado, este circuito não foi utilizado para os cálculos realizados neste projeto, sendo mantido para possíveis aplicações futuras.

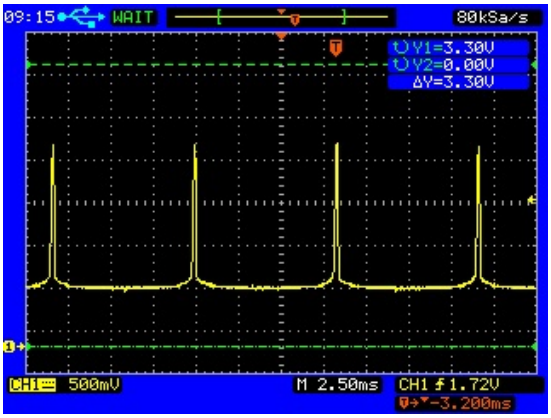


Figura 44 – Visualização do *zero crossing*.

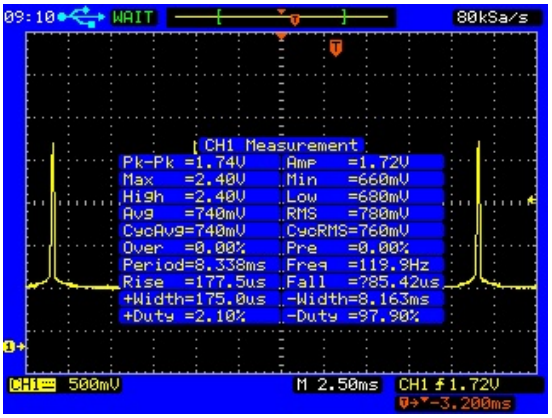


Figura 45 – Medição do *zero crossing*.

3.3 Ensaios com o circuito de tensão

O circuito de condicionamento de tensão elétrica foi testado individualmente dos demais circuitos, de forma a verificar sua resposta quando aplicado à rede elétrica. As Figuras 46 e 47 mostram o sinal medido na saída do circuito de condicionamento da tensão com o circuito conectado a uma rede de 127V/60Hz.

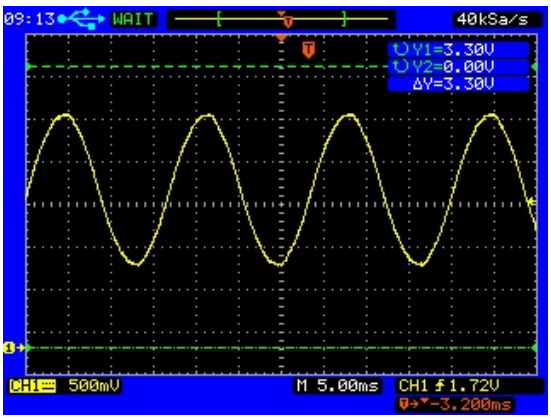


Figura 46 – Sinal tensão.

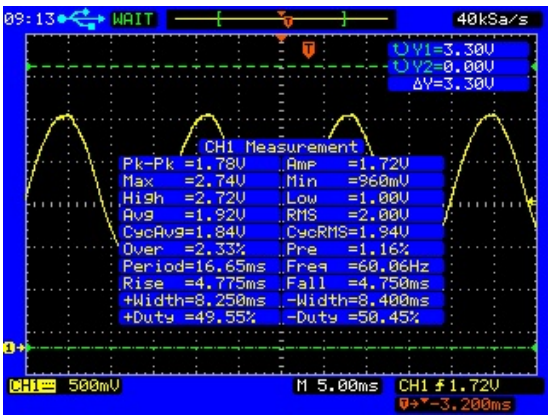


Figura 47 – Medições do sinal tensão.

Nota-se que o sinal é CC, graças ao *offset* gerado no circuito amplificador diferencial. Além disso, o sinal foi ajustado para que sua amplitude não ultrapassasse a tensão de 3,3V, que é a máxima suportada pela entrada analógica do microcontrolador usado. O sinal possui uma frequência de 60,06Hz, com valor mínimo de 0,96V e máximo de 2,74V, estando dentro da faixa aceitável e seguro contra pequenas oscilações.

3.4 Ensaios com o circuito corrente

O circuito de condicionamento de corrente elétrica foi testado individualmente dos demais circuitos, de forma a verificar sua resposta quando aplicado à rede elétrica com uma carga puramente resistiva. A carga de teste utilizada foi um ferro de solda Hikari Plus, modelo SC-30, de 25W. Nota-se que a corrente eficaz teórica demandada pela carga

é de cerca de 197 mA. Para que o sinal apresentado tivesse uma amplitude maior, sem aplicar uma carga com corrente maior, foram realizadas 5 voltas do condutor fase ao redor do transdutor de corrente, o que multiplica a corrente medida em 5. A Figura 48 mostra a configuração citada.

Figura 48 – Configuração com 5 voltas no transdutor de corrente.

As Figuras 49 e 50 mostram o sinal medido na saída do circuito de condicionamento da corrente com o circuito conectado à carga de teste.

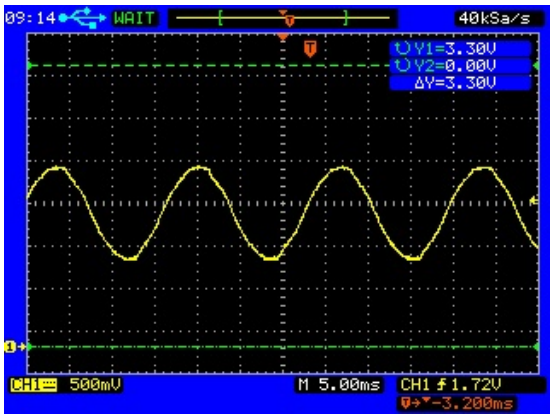


Figura 49 – Sinal corrente.

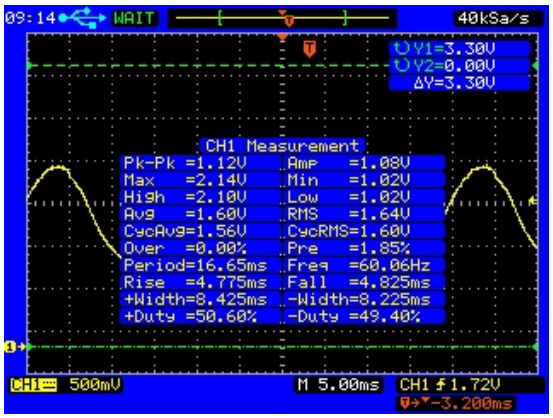


Figura 50 – Medições do sinal corrente.

Nota-se que o sinal é CC, graças ao *offset* gerado no circuito de condicionamento. Além disso, o sinal foi ajustado para que sua amplitude não ultrapassasse a tensão de 3,3V,

de maneira semelhante ao circuito de medição de tensão. O sinal possui uma frequência de 60,06Hz, com valor mínimo de 1,02V e máximo de 2,14V, estando dentro da faixa aceitável e seguro contra pequenas oscilações.

3.5 Medições realizadas com o microcontrolador

Conforme apresentado no Capítulo 2, o firmware do sistema de medição foi programado para realizar 595 amostras de um intervalo de um ciclo do sinal, ou seja, de um intervalo de 16,67ms. Os algoritmos coletam estas amostras da memória a cada 1s e realizam os cálculos dos parâmetros do sinal. No ambiente de programação STM32 Cube IDE, em modo debug com a placa conectada ao computador e à rede elétrica, medindo tanto tensão como corrente elétrica, foi iniciado o processo de medição e coletada uma amostra dos vetores de tensão e corrente. Esta amostra foi realizada em dois pontos do processo: antes da correção de fase da corrente; e, após a correção. Com os dados, foram plotados os gráficos das Figuras 51 e 52, respectivamente antes e depois da correção de fase do sinal de corrente.

Figura 51 – Gráfico dos sinais de tensão e corrente (sem ajuste).

A correção da fase da corrente foi realizada com base na aplicação de uma carga puramente resistiva, uma vez que este tipo de carga não altera a fase do sinal. A defasagem originalmente observada ocorre devido à natureza do transdutor de corrente, que é indutivo. É importante observar que o sinal de corrente teve sua amplitude multiplicada por 50 nos gráficos apresentados para que fosse melhor visualizado em comparação com o sinal de tensão.

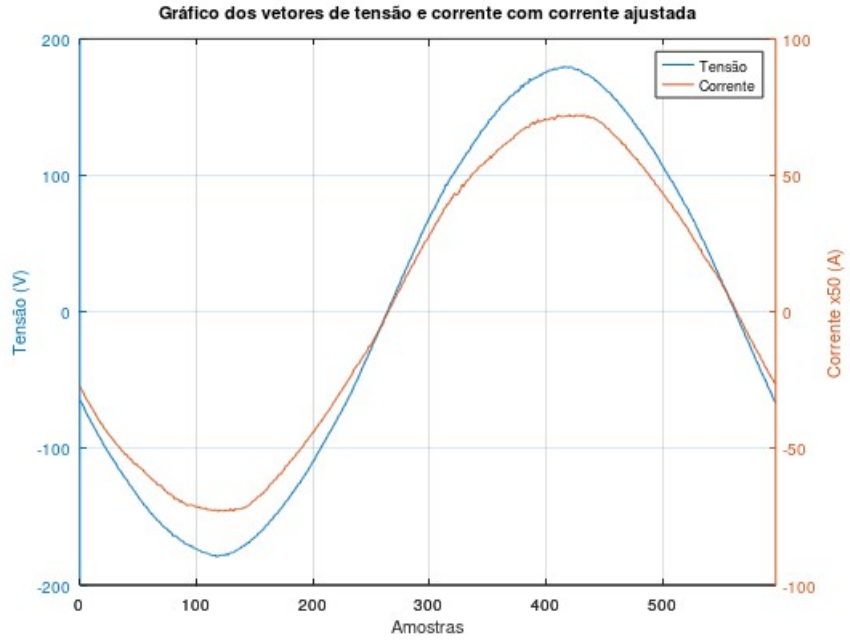


Figura 52 – Gráfico dos sinais de tensão e corrente (com ajuste).

É possível notar a eficácia do algoritmo de correção de fase, uma vez que os sinais estão próximos de amplitude 0 nos mesmos instantes. De maneira análoga, os sinais apresenta-se próximos dos picos também nos mesmos instantes.

Finalmente, os valores calculados para este ensaio podem ser vistos na tela de visualização das variáveis, no modo de debug do STM32 Cube IDE, conforme apresentado na Figura 43 da Seção 2.2.5.2. Para melhor visualização, os valores são apresentados na Tabela 2.

Parâmetro	Valor medido
Tensão eficaz (V_{ef})	124,99 V
Corrente eficaz (I_{ef})	1,02 A
Defasagem (φ)	0,03 ^{o}
Fator de Potência (FP)	0,99
Potência Ativa (P)	128,11 W
Potência Reativa (Q)	0,07 VAr
Potência Aparetente (S)	128,13 VA
Tensão de pico (V_P)	179,71 V
Corrente de pico (I_P)	1,44 A

Tabela 2 – Valores medidos no ensaio com carga puramente resistiva

Para efeito de comparação, foram realizados outros ensaios com outras cargas, de forma a verificar os valores apresentados. A Tabela 3, mostra os valores obtidos de um ensaio com um carregador de celular comum de 127V/60Hz com 27W de potência máxima nominal.

Parâmetro	Valor medido
Tensão eficaz (V_{ef})	127,39 V
Corrente eficaz (I_{ef})	1,64 A
Defasagem (φ)	3,72 ^{o}
Fator de Potência (FP)	-0,66
Potência Ativa (P)	94,65 W
Potência Reativa (Q)	-9,23 VAr
Potência Aparetente (S)	142,32 VA
Tensão de pico (V_P)	181,41 V
Corrente de pico (I_P)	3,22 A

Tabela 3 – Valores medidos no ensaio com carregador de celular.

A Tabela 4, mostra os valores obtidos de um ensaio com um eletrodoméstico em operação: um liquidificador de 127V/60Hz com 550W de potência nominal atuando em baixa rotação.

Parâmetro	Valor medido
Tensão eficaz (V_{ef})	127,14 V
Corrente eficaz (I_{ef})	3,19 A
Defasagem (φ)	-6,38 ^{o}
Fator de Potência (FP)	-0,89
Potência Ativa (P)	360,19 W
Potência Reativa (Q)	-45,15 VAr
Potência Aparetente (S)	406,29 VA
Tensão de pico (V_P)	182,05 V
Corrente de pico (I_P)	3,75 A

Tabela 4 – Valores medidos no ensaio com um liquidificador.

Vale ressaltar que não se dispunha de um equipamento de medição de energia para fins de comparação e validação dos dados apresentados. Até este ponto, a validação se dá de forma teórica.

3.6 Visualização dos dados na nuvem

Conforme discutido no Capítulo 2, uma das possibilidades deste sistema é o envio dos dados coletados para a nuvem. Sendo assim, foi utilizada a plataforma ThingSpeak.com, sendo montada a página de exibição de dados da Figura 53.

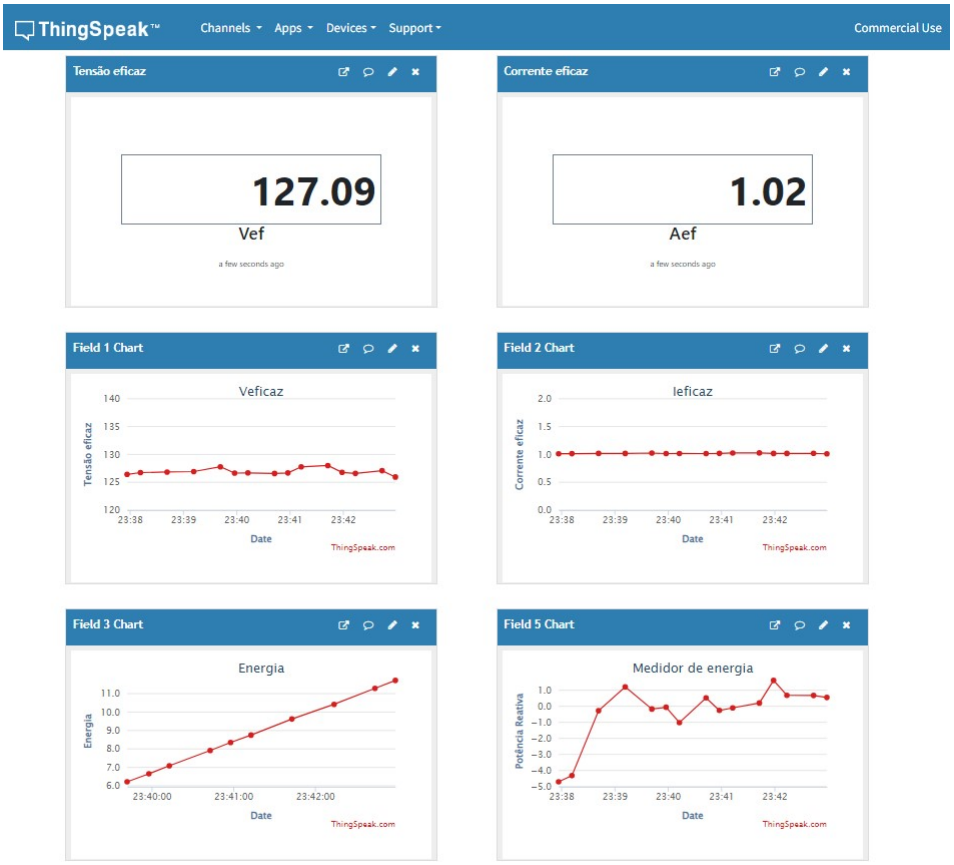


Figura 53 – Página do ThingSpeak exibindo as informações do medidor.

Além dos dados exibidos na plataforma do ThingSpeak.com, uma página foi desenvolvida para exibição das informações em gráficos, de forma mais personalizada. Isso foi possível graças a plataforma gratuita Glitch.com, que permite a hospedagem de sites e uso de algumas tecnologias de desenvolvimento, como NodeJS. O site basicamente exibe gráficos, como os vistos nas Figuras 54, 55 e 56.

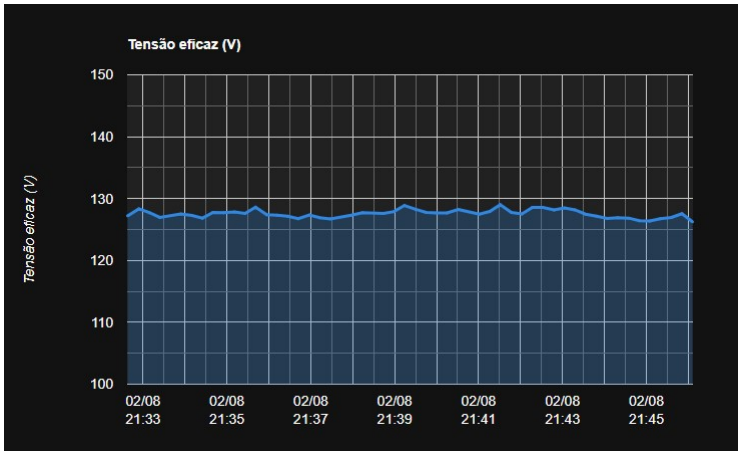


Figura 54 – Gráfico de tensão da aplicação *web*.

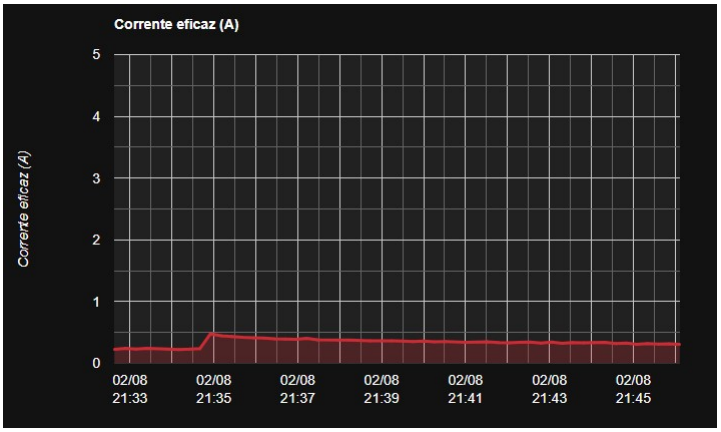


Figura 55 – Gráfico de corrente da aplicação *web*.

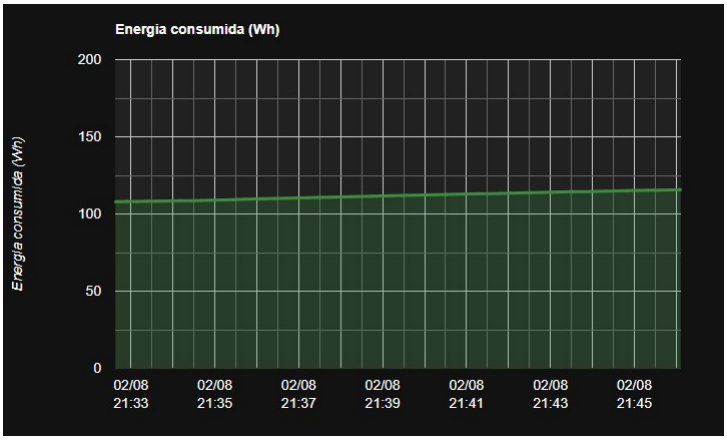


Figura 56 – Gráfico de energia da aplicação *web*.

4 Considerações finais

Neste capítulo final, apresentam-se as considerações finais do trabalho, considerando-se a proposta inicial, o processo de desenvolvimento e o resultado final. Para melhor organização, as considerações foram divididas nos tópicos a seguir, finalizando com algumas sugestões para trabalhos futuros baseados neste sistema.

4.1 Considerações sobre *hardware*

Foram dimensionados circuitos específicos para realização do condicionamento dos sinais de tensão e corrente, para que fosse possível a realização da medição pelo microcontrolador. A proposta do circuito foi aplicar uma solução de baixo custo, incluindo algumas proteções contra surtos já que ele trabalha diretamente conectado à rede elétrica. Pelos ensaios e medições realizadas, observou-se uma boa coerência entre medições realizadas por instrumentos de mercado e os valores observados no sistema.

Vale uma observação importante quanto à segurança: o circuito destinado à medição de tensão elétrica é baseado em um divisor de tensão resistivo, justamente pelo baixo custo. A topologia do circuito não isola a rede elétrica do circuito de medição, o que pode ocasionar riscos de choque elétrico, curto-circuito direto da rede elétrica, além de expor o circuito a potenciais elevados em determinadas circunstâncias (como rompimento de alguma conexão do circuito divisor ou queima de algum componente). É importante que esta característica seja cuidadosamente analisada em aplicações futuras.

Durante os testes, em raros momentos foi observada uma certa variação na amplitude do sinal de tensão quando o microcontrolador era conectado ao computador pela porta USB. A suspeita é mau contato, já que este comportamento não se apresentou como um padrão bem definido. O fato é que a topologia do circuito de medição de tensão não proporciona um referencial bem definido, o que pode levar a flutuação do sinal de saída. Para usos futuros, deve-se investigar estes comportamentos, mas para o propósito deste trabalho, consideram-se bem sucedidos os circuitos usados.

Não foi possível tecer considerações quando a qualidade dos sinais captados. Isso se deve à não disponibilidade de equipamentos de medição de qualidade de energia que pudessem fornecer dados comparáveis aos gerados pelo sistema de medição. Sugere-se, em aplicações futuras, analisar os sinais e os dados gerados de forma a compará-los com os oriundos de equipamentos específicos para avaliar os circuitos quanto a sua topologia e construção.

4.2 Considerações sobre *firmware*

A proposta do *firmware* é captar os sinais em intervalos de 1 segundo, calculando todos os parâmetros de interesse entre cada aquisição. Nota-se que o microcontrolador usado foi capaz de realizar tais cálculos de forma efetiva e dentro dos limites esperados. Foram usadas variáveis em ponto flutuante, mesmo que o microcontrolador não tivesse hardware dedicado a sua manipulação eficiente, o que poderia ser otimizado em testes futuros com outros microcontroladores ou adotando apenas variáveis de tipo inteiro, devidamente manipuladas para os propósitos deste trabalho. Nota-se que o sistema operou quase no limite, ocupando mais de 90% da memória RAM e mais de 80% da Flash, conforme Figura 57.

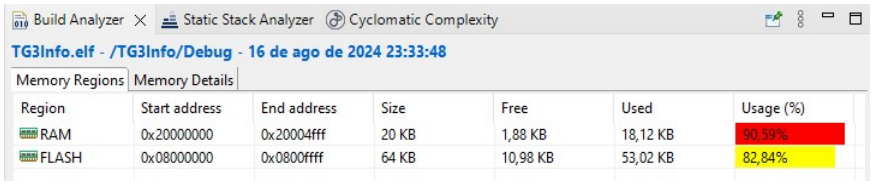


Figura 57 – Detalhe do consumo de recursos do microcontrolador.

As funções principais de: captar os sinais de tensão e corrente elétrica; calcular os parâmetros seguindo algoritmos específicos; e, finalmente, disponibilizar os dados para visualização local e na nuvem; foram concluídas com sucesso.

Vale ressaltar que, apesar de implementado o hardware de armazenamento local em um SDCard, tal recurso não foi de fato utilizado.

4.3 Sugestões para trabalhos futuros

Este trabalho aborda uma grande quantidade de conceitos e áreas de pesquisa. Em termos de *hardware*, foram realizados os projetos de circuitos dedicados a captação de sinais de tensão e corrente elétrica. A topologia dos circuitos foi escolhida pelos critérios de simplicidade e baixo custo. Existem, porém, algumas outras topologias que podem apresentar um balanço favorável entre custo e benefício. Sendo assim, uma sugestão de pesquisa é justamente o desenvolvimento de um estudo comparativo de algumas técnicas de captação e condicionamento de sinais, de forma a possibilitar futuros projetos no processo de escolha e aplicação.

Ainda sobre *hardware*, a utilização do microcontrolador STM32F103 deu-se pela sua facilidade de obtenção no mercado nacional, bem como nos conhecimentos prévios do desenvolvedor. Seria interessante uma pesquisa mais abrangente em termos de tecnologias e fabricantes distintos, a fim de selecionar um microcontrolador que seja o mais próximo possível das necessidades de *hardware* e capacidades de processamento e armazenamento.

Como forma de validação das medições e para possibilitar uma calibração mais efetiva, sugere-se a utilização de equipamentos de medição mais avançados, como analisadores de energia. Isso pode melhorar a qualidade das informações fornecidas pelo medidor, além de validar os circuitos utilizados. Além disso, o processo de validação poderia usar cargas puramente capacitivas e puramente indutivas, como forma de constatar a interação do transdutor com esses tipos de carga, melhorando ainda mais a qualidade dos dados.

Em situações de falha de comunicação, queda de energia ou falhas no sistema, o medidor deve ser capaz de criar um *buffer* de dados localmente, realizando o sincronismo quando reestabelecidas as condições normais de funcionamento. Isso configura mais uma proposta de melhoria no sistema, que ainda pode se beneficiar de teoria de filas.

Outra área que pode se beneficiar deste projeto e trazer melhorias é a parte da web. Existem várias possibilidades de melhoria, incluindo a criação de uma aplicação web com banco de dados e demais recursos para possibilitar um histórico dos dados coletados. Além disso, pode-se aplicar conceitos de segurança da informação, criando uma metodologia para transmissão segura das informações, uma vez que se o medidor for aplicado em residências, o consumo energético pode ser uma informação sensível.

Referências

ANEEL. *Relatório de Perdas de Energia Elétrica na Distribuição*. [S.l.]: ANEEL - Agência Nacional de Energia Elétrica - Ministério de Minas e Energia - Governo Federal do Brasil, 2022. Acesso em: 16 ago 2024. Citado 3 vezes nas páginas 2, 3 e 4.

EPE. *Anuário Estatístico de Energia Elétrica 2024 (Ano base 2023)*. Rio de Janeiro: EPE - Empresa de Pesquisa Energética - Ministério de Minas e Energia - Governo Federal do Brasil, 2024. Acesso em: 16 ago 2024. Citado 2 vezes nas páginas 2 e 3.

ESPRESSIF. *ESP8266 Technical Reference*. Shanghai, China: Espressif, 2020. Citado na página 21.

HI-LINK. *10W ultra-compact power module: Technical reference*. Shenzhen, China: Hi-Link, 2019. Citado na página 22.

PAULA, A. de. *Sistema de Medição de Demanda e Qualidade de Energia Elétrica*. Dissertação (Mestrado) — Escola de Engenharia de São Carlos - Universidade de São Paulo, São Carlos, São Paulo, 2000. Citado na página 34.

PERTENCE, A. *Amplificadores operacionais e filtros ativos: teoria, projetos, aplicações e laboratorio*. [S.l.]: Makron Books, 1990. ISBN 9780074502525. Citado 2 vezes nas páginas 11 e 16.

SOLOMON-SYSTECH. *SSD1306 Advance Information*. Hong Kong: Solomon-Systech, 2008. Citado na página 19.

SOUZA, M. A. de. *Detecção e Identificação de Perdas Comerciais de Energia Elétrica*: Uma abordagem para smartgrids. Dissertação (Mestrado) — Universidade Federal de Juiz de Fora, Juiz de fora, 2016. Citado na página 4.

STMICROELECTRONICS. *AN2548 Application note: Introduction to DMA controller for STM32 MCUs*. Geneva, Switzerland: STMicroelectronics, 2010. Citado na página 30.

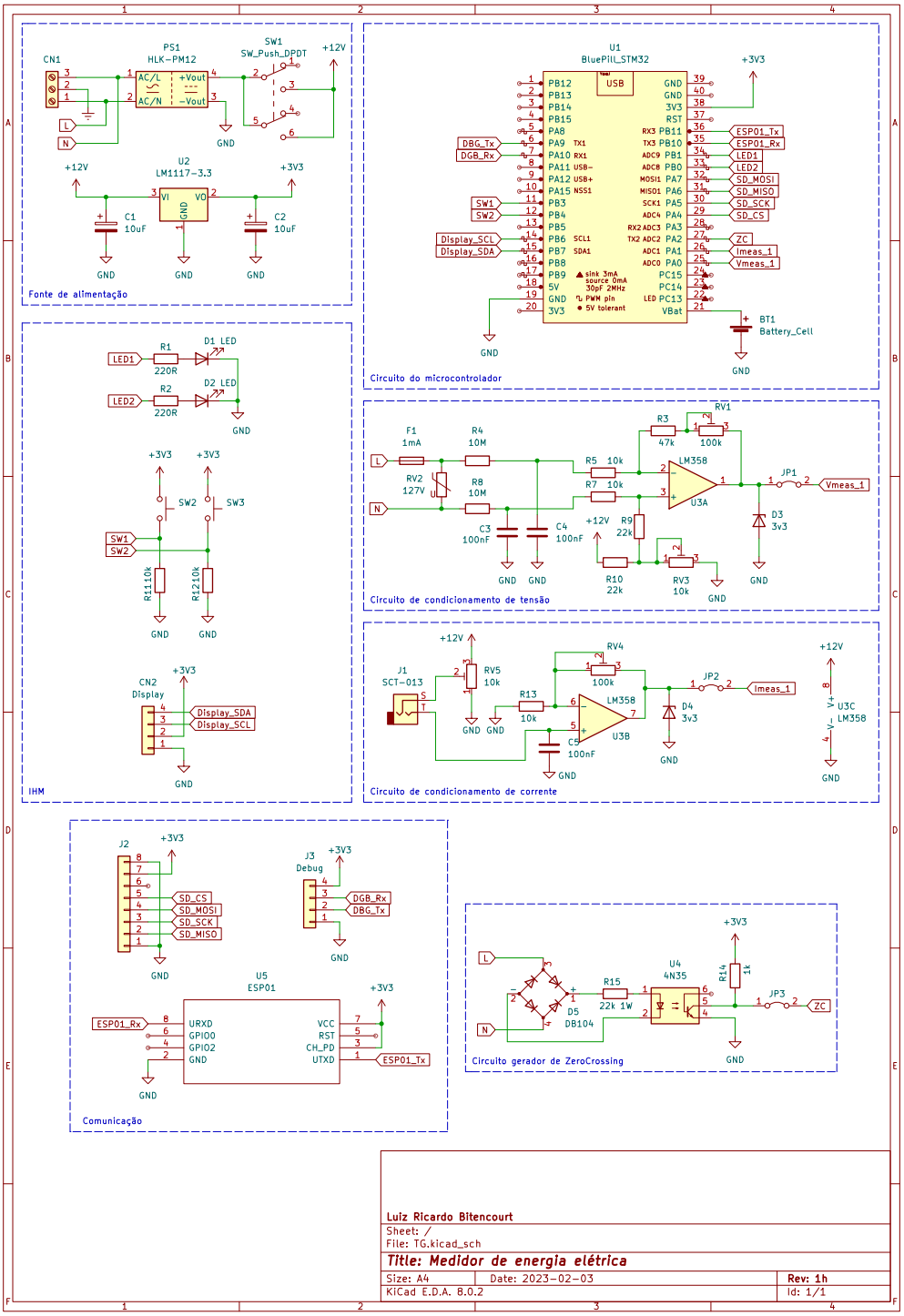
STMICROELECTRONICS. *AN3116 Application note: STM32's ADC modes and their applications*. Geneva, Switzerland: STMicroelectronics, 2024. Citado na página 28.

STMICROELECTRONICS. *UM2609 STM32Cube IDE user guide*. Geneva, Switzerland: STMicroelectronics, 2024. Citado na página 33.

TURGEL, R. S. Digital wattmeter using a sampling method. *IEEE Transactions on Instrumentation and Measurement*, v. 23, n. 4, p. 337–341, 1974. Citado na página 33.

Apêndices

APÊNDICE A – Esquema eletrônico do medidor de energia



APÊNDICE B – Código fonte do *firmware*

O código abaixo foi desenvolvido para realização das medições propostas e envio dos dados à nuvem.

```
/**
 ************************************************************
 * Nome do projeto:        Medidor de Energia
 * Desenvolvimento:        Luiz Ricardo Bitencourt
 ************************************************************
 */
/* Includes ------------------------------------------------------*/
#include "main.h"
#include "fatfs.h"

/* Private includes ----------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "screens.h"
#include "clock.h"
#include "energy.h"
#include "sdcard.h"
#include "math.h"
#include "stdbool.h"
#include "ssd1306.h"
#include "esp.h"
#include <string.h>
#include <stdio.h>
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------*/
/* USER CODE BEGIN PTD */
typedef enum {
        ST_IDLE,
        ST_MENU_PRE,
        ST_MENU,
        ST_MEASURE,
        ST_MEAS_1,
        ST_MEAS_2,
        ST_MEAS_3,
```

```
35          ST_CLOCK,
36          ST_WIFI,
37          ST_SDCARD
38  } State;
39
40  typedef enum {
41          MENU_MEASURE, MENU_CLOCK, MENU_WIFI, MENU_SDCARD
42  } Menu;
43  /* USER CODE END PTD */
44
45  /* Private define ------------------------------------------------*/
46  /* USER CODE BEGIN PD */
47  #define SAMPLES                         598
48
49  #define MAX_MENU                4
50  #define MAX_MENU_PAGE        MAX_MENU/2
51  #define START_MENU                MENU_MEASURE
52
53  #define TICKTIME1                800
54  #define TICKTIME2                100
55  #define TICKTIME3                800
56  #define TICKTIME4                900
57  #define TICKTIME5                1000
58  #define TICKTIME6                15000
59  /* USER CODE END PD */
60
61  /* Private variables ---------------------------------------------*/
62  ADC_HandleTypeDef hadc1;
63  ADC_HandleTypeDef hadc2;
64  DMA_HandleTypeDef hdma_adc1;
65
66  I2C_HandleTypeDef hi2c1;
67
68  RTC_HandleTypeDef hrtc;
69
70  SPI_HandleTypeDef hspi1;
71
72  TIM_HandleTypeDef htim3;
73
74  UART_HandleTypeDef huart1;
75  UART_HandleTypeDef huart3;
```

```
76
77 /* USER CODE BEGIN PV */
78 State currentState = ST_IDLE;
79 Menu currentMenu = MENU_MEASURE;
80 uint8_t currentMenuPage = 0;
81 char menuNames[MAX_MENU][11] = { "   Measure ", "   Clock   ", "   WiFi
   ↪  ","   SDCard  " };
82
83 bool sdCardSwitch = false;
84 bool wifiSwitch = false;
85
86 unsigned int tick1Now;
87 unsigned int tick2Now;
88 unsigned int tick3Now;
89 unsigned int tick4Now;
90 unsigned int tick5Now;
91 unsigned int tick6Now;
92
93 uint32_t adcRaw[SAMPLES];
94 uint32_t adcVoltage[SAMPLES];
95 uint32_t adcCurrent[SAMPLES];
96
97 float sampleVoltage[SAMPLES];
98 float sampleCurrent[SAMPLES];
99 float sampleCurrentDephased[SAMPLES];
100
101 char bufferVars[15];
102
103 float veficaz;
104 float ieficaz;
105 float potAparente;
106 float potAtiva;
107 float potReativa;
108 float fatorPotencia;
109 float phi;
110 float energia = 0;
111 float vpp = 0;
112 float vpn = 0;
113 float ipp = 0;
114 float ipn = 0;
115
```

```
116 float tp = 205.2;
117 float voff = 1.782;
118 float tc = 2.66;
119 float ioff = 1.498;
120 int iphase = 272;
121 /* USER CODE END PV */
122
123 /* Private function prototypes -----------------------------------*/
124 void SystemClock_Config(void);
125 static void MX_GPIO_Init(void);
126 static void MX_DMA_Init(void);
127 static void MX_ADC1_Init(void);
128 static void MX_ADC2_Init(void);
129 static void MX_I2C1_Init(void);
130 static void MX_RTC_Init(void);
131 static void MX_SPI1_Init(void);
132 static void MX_USART1_UART_Init(void);
133 static void MX_USART3_UART_Init(void);
134 static void MX_TIM3_Init(void);
135 /* USER CODE BEGIN PFP */
136 void measAll();
137 void sendAll();
138 /* USER CODE END PFP */
139
140 /* Private user code ---------------------------------------------*/
141 /* USER CODE BEGIN 0 */
142 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
143 }
144
145 void measAll() {
146         // ADC stop, start and copy raw values:
147         HAL_ADC_Stop(&hadc2);
148         HAL_ADCEx_MultiModeStop_DMA(&hadc1);
149         HAL_ADC_Start(&hadc2);
150         HAL_ADCEx_MultiModeStart_DMA(&hadc1, adcRaw, SAMPLES);
151
152         // To get the peak values:
153         vpp = 0;
154         vpn = 0;
155         ipp = 0;
156         ipn = 0;
```

```
157
158         // Separate the measures from the DMA array:
159         for (int i = 0; i < SAMPLES; i++) {
160                 adcVoltage[i] = adcRaw[i] & 0x0000ffff;
161                 adcCurrent[i] = adcRaw[i] >> 16;
162         }
163
164         // Adjust the values:
165         for (int i = 0; i < SAMPLES; i++) {
166                 sampleVoltage[i] = (((adcVoltage[i]) * (3.3 / 4095)) -
    ↪  voff) * tp;
167                 sampleCurrentDephased[i] = (((adcCurrent[i]) * (3.3 /
    ↪  4095)) - ioff) * tc;
168
169                 // Update the peak values:
170                 if (sampleVoltage[i] > vpp)
171             vpp = sampleVoltage[i];
172                 if (sampleVoltage[i] < vpn)
173             vpn = sampleVoltage[i];
174                 if (sampleCurrentDephased[i] > ipp)
175             ipp = sampleCurrentDephased[i];
176                 if (sampleCurrentDephased[i] < ipn)
177             ipn = sampleCurrentDephased[i];
178         }
179
180         // Adjust current phase
181         for (int i = 0; i < SAMPLES; i++) {
182                 if (i < SAMPLES - iphase) sampleCurrent[i] =
    ↪  sampleCurrentDephased[i + iphase];
183                 else if (i >= SAMPLES - iphase) sampleCurrent[i] =
    ↪  sampleCurrentDephased[i - (SAMPLES - iphase)];
184         }
185
186         veficaz = getRMS(sampleVoltage, SAMPLES);
187         ieficaz = getRMS(sampleCurrent, SAMPLES);
188         potAparente = getApparentPower(veficaz, ieficaz);
189         potAtiva = getActivePower(sampleVoltage, sampleCurrent, SAMPLES);
190         potReativa = getReactivePower(sampleVoltage, sampleCurrent,
    ↪  SAMPLES);
191         fatorPotencia = getPowerFactor(potAtiva, potAparente);
192         phi = getPhi(potReativa, potAparente);
```

```
193         energia = energia + (potAparente) / 3600;
194 }
195
196 void sendAll() {
197         // TODO: Select witch data will be sent
198         float Value_Buf[7];
199
200         if (wifiSwitch) {
201                 Value_Buf[0] = veficaz;
202                 Value_Buf[1] = ieficaz;
203                 Value_Buf[2] = energia;
204                 Value_Buf[3] = potAtiva;
205                 Value_Buf[4] = potReativa;
206                 Value_Buf[5] = potAparente;
207                 Value_Buf[6] = fatorPotencia;
208
209                 esp_SendM("5MMX7IAC2OS8DEQO", 7, Value_Buf);
210         }
211 }
212
213 /* USER CODE END 0 */
214
215 /**
216  * @brief  The application entry point.
217  * @retval int
218  */
219 int main(void) {
220         /* MCU Configuration--------------------------------------------------------*/
221
222         /* Reset of all peripherals, Initializes the Flash interface and
    ↪  the Systick. */
223         HAL_Init();
224
225         /* Configure the system clock */
226         SystemClock_Config();
227
228         /* Initialize all configured peripherals */
229         MX_GPIO_Init();
230         MX_DMA_Init();
231         MX_ADC1_Init();
232         MX_ADC2_Init();
```

```
233         MX_I2C1_Init();
234         MX_RTC_Init();
235         MX_SPI1_Init();
236         MX_USART1_UART_Init();
237         MX_USART3_UART_Init();
238         MX_TIM3_Init();
239         MX_FATFS_Init();
240         /* USER CODE BEGIN 2 */
241         /*=====================================================================*/
242         ssd1306_Init();                 // Initialize the display
243         //clock_setTime(14, 15, 00);
244         clock_setDate(15, RTC_MONTH_JUNE, (2024-1970),
    ↪  RTC_WEEKDAY_SATURDAY);
245 
246         // Special menu:
247         for (int i = 0; i < 10; i++) {
248                 HAL_Delay(5);
249                 while (HAL_GPIO_ReadPin(SW1_GPIO_Port, SW1_Pin)         &&
    ↪  HAL_GPIO_ReadPin(SW2_GPIO_Port, SW2_Pin)) {
250                         displayText(1, 1, Font_11x18, "S E C R E T");
251                         displayText(2, 1, Font_11x18, "           ");
252                         displayText(3, 1, Font_11x18, "  M E N U  ");
253 
254                         // TODO: Special menu to change some constants and
    ↪  calibrations
255                         clock_adjust();
256                 }
257         }
258         /*=====================================================================*/
259 
260         HAL_ADC_Start(&hadc2);
    ↪  Inicializa o ADC2
261         HAL_ADCEx_MultiModeStart_DMA(&hadc1, adcRaw, SAMPLES);//
    ↪  Inicializa o ADC1 multimode com DMA usando a variável adcg para
    ↪  armazenar os valores
262         HAL_DMA_Init(&hdma_adc1);
263         /* USER CODE END 2 */
264 
265         /* Infinite loop */
266         /* USER CODE BEGIN WHILE */
267         /*=====================================================================*/
```

```
268         tick1Now = HAL_GetTick();
269         tick3Now = HAL_GetTick();
270         tick5Now = HAL_GetTick();
271         tick6Now = HAL_GetTick();
272 
273         while (1) {
274                 /*=============================================*/
275                 // ALWAYS DO:
276                 /*=============================================*/
277                 if (HAL_GetTick() >= tick1Now + TICKTIME1) {
278                         tick1Now = HAL_GetTick();
279                         HAL_GPIO_TogglePin(LED1_GPIO_Port, LED1_Pin);
280                 }
281 
282                 if (currentState == ST_MEAS_1 || currentState == ST_MEAS_2
    ↪  || currentState == ST_MEAS_3) {
283                         // Do the measures:
284                         if (HAL_GetTick() >= tick5Now + TICKTIME5) {
285                                 tick5Now = HAL_GetTick();
286                                 measAll();
287 
288                                 // Toggle measuring indicator (LED2):
289                                 HAL_GPIO_TogglePin(LED2_GPIO_Port,
    ↪  LED2_Pin);
290                         }
291 
292                         // Send to the cloud (if it is connected):
293                         if (HAL_GetTick() >= tick6Now + TICKTIME6) {
294                                 tick6Now = HAL_GetTick();
295                                 sendAll();
296                         }
297                 }
298 
299                 /*=============================================*/
300                 /* MAIN STATE MACHINE: */
301                 /*=============================================*/
302                 switch (currentState) {
303 
304                 /* ---------------------------- */
305                 // State: IDLE Opening screen
306                 /* ---------------------------- */
```

```
307                 case ST_IDLE:
308                         /* --- DISPLAY UPDATE --- */
309                         ssd1306_DrawRectangle(1, 1, SSD1306_WIDTH - 1,
    ↪  SSD1306_HEIGHT - 1, White);
310                         displayText(1, 1, Font_11x18, "EnergyMeter");
311                         displayText(2, 1, Font_11x18, "   . . .   ");
312                         displayText(3, 1, Font_11x18, "   UFABC   ");
313                         /* ---------------------- */
314
315                         // Take a time to go to the next state:
316                         if (HAL_GetTick() >= tick3Now + TICKTIME3) {
317                                 tick3Now = HAL_GetTick();
318
319                                 currentState = ST_CLOCK;
320                                 setScreen(Clean);
321                         }
322                         break;
323
324                         /* ---------------------------- */
325                         // State: MENU_PRE
326                         /* ---------------------------- */
327                 case ST_MENU_PRE:
328                         /* --- DISPLAY UPDATE --- */
329                         ssd1306_DrawRectangle(1, 1, SSD1306_WIDTH - 1,
    ↪  SSD1306_HEIGHT - 1, White);
330                         displayText(1, 1, Font_11x18, "  M E N U  ");
331                         displayText(2, 1, Font_11x18,
    ↪  menuNames[currentMenuPage]);
332                         displayText(3, 1, Font_11x18,
    ↪  menuNames[currentMenuPage + 1]);
333                         /* ---------------------- */
334
335                         // Turn off the measuring indicator (LED2):
336                         HAL_GPIO_WritePin(LED2_GPIO_Port, LED2_Pin,
    ↪  GPIO_PIN_RESET);
337
338                         // GOTO MENU
339                         currentState = ST_MENU;
340                         break;
341
342                         /* ---------------------------- */
```

```
343                         // State: MENU to show items and selection
344                         /* ---------------------------- */
345                 case ST_MENU:
346                         // Put the cursor:
347                         if (currentMenu % 2 == 0) {
348                                 displayText(2, 1, Font_11x18, " > ");
349                         } else {
350                                 displayText(3, 1, Font_11x18, " > ");
351                         }
352
353                         // Change the position of the cursor and the page
    ↪  of the menu:
354                         if (HAL_GPIO_ReadPin(SW1_GPIO_Port, SW1_Pin) ==
    ↪  GPIO_PIN_SET) {
355                                 HAL_Delay(10);
356                                 while (HAL_GPIO_ReadPin(SW1_GPIO_Port,
    ↪  SW1_Pin) == GPIO_PIN_SET);
357                                 HAL_Delay(10);
358
359                                 currentMenu++;
360                                 currentState = ST_MENU_PRE;
361
362                                 if (currentMenu == MAX_MENU) {
363                                         currentMenu = START_MENU;
364                                 }
365                                 if (currentMenu % 2 == 0) {
366                                         currentMenuPage += 2;
367                                         if (currentMenuPage >=
    ↪  MAX_MENU_PAGE + 2) {
368                                                 currentMenuPage = 0;
369                                         }
370                                 }
371                         }
372
373                         // GOTO the selected menu:
374                         if (HAL_GPIO_ReadPin(SW2_GPIO_Port, SW2_Pin) ==
    ↪  GPIO_PIN_SET) {
375                                 HAL_Delay(10);
376                                 while (HAL_GPIO_ReadPin(SW2_GPIO_Port,
    ↪  SW2_Pin) == GPIO_PIN_SET);
377                                 HAL_Delay(10);
```

```
378
379                                 if (currentMenu == MENU_MEASURE) {
380                                         currentState = ST_MEASURE;
381                                         tick3Now = HAL_GetTick();
382                                 } else if (currentMenu == MENU_CLOCK)
383                                         currentState = ST_CLOCK;
384                                 else if (currentMenu == MENU_WIFI)
385                                         currentState = ST_WIFI;
386                                 else if (currentMenu == MENU_SDCARD)
387                                         currentState = ST_SDCARD;
388
389                                 currentMenuPage = 0;
390                                 currentMenu = MENU_MEASURE;
391                                 setScreen(Clean);
392                         }
393                         break;
394
395                         /* ----------------------------- */
396                         // State: CLOCK
397                         /* ----------------------------- */
398                 case ST_CLOCK:
399                         /* --- DISPLAY UPDATE --- */
400                         ssd1306_DrawRectangle(1, 1, SSD1306_WIDTH - 1,
    ↪ SSD1306_HEIGHT - 1, White);
401                         displayText(1, 1, Font_11x18,
    ↪ clock_getRTC(RTCClock));
402                         displayText(2, 1, Font_11x18,
    ↪ clock_getRTC(RTCDate));
403                         displayText(3, 1, Font_11x18,
    ↪ clock_getRTC(RTCWeekday));
404                         /* ---------------------- */
405
406                         // Any key pressed exit the CLOCK:
407                         if (HAL_GPIO_ReadPin(SW1_GPIO_Port, SW1_Pin) ==
    ↪ GPIO_PIN_SET || HAL_GPIO_ReadPin(SW2_GPIO_Port, SW2_Pin) ==
    ↪ GPIO_PIN_SET) {
408                                 HAL_Delay(10);
409                                 while (HAL_GPIO_ReadPin(SW1_GPIO_Port,
    ↪ SW1_Pin) == GPIO_PIN_SET);
410                                 while (HAL_GPIO_ReadPin(SW2_GPIO_Port,
    ↪ SW2_Pin) == GPIO_PIN_SET);
```

```
411                                 HAL_Delay(10);
412
413                                 currentState = ST_MENU_PRE;
414                                 setScreen(Clean);
415                         }
416                         break;
417
418                         /* ---------------------------- */
419                         // State: MEASURE
420                         /* ---------------------------- */
421                 case ST_MEASURE:
422                         /* --- DISPLAY UPDATE --- */
423                         ssd1306_DrawRectangle(1, 1, SSD1306_WIDTH - 1,
    ↪  SSD1306_HEIGHT - 1, White);
424                         displayText(1, 1, Font_11x18, "           ");
425                         displayText(2, 1, Font_11x18, "  M E A S  ");
426                         displayText(3, 1, Font_11x18, "           ");
427                         /* ---------------------- */
428
429                         // Take a time to go to the next state:
430                         if (HAL_GetTick() >= tick3Now + TICKTIME3) {
431                                 tick3Now = HAL_GetTick();
432
433                                 // Get the measure start time:
434                                 tick5Now = HAL_GetTick();
435                                 tick6Now = HAL_GetTick();
436
437                                 // GOTO MEAS page 1:
438                                 currentState = ST_MEAS_1;
439                                 setScreen(Clean);
440                         }
441                         break;
442
443                         /* ---------------------------- */
444                         // State: MEASURE Page 1
445                         /* ---------------------------- */
446                 case ST_MEAS_1:
447                         /* --- DISPLAY UPDATE --- */
448                         ssd1306_DrawRectangle(1, 1, SSD1306_WIDTH - 1,
    ↪  SSD1306_HEIGHT - 1, White);
449                         sprintf(bufferVars, "Vef = %3.2f", veficaz);
```

```
450                         displayText(1, 1, Font_11x18, bufferVars);
451                         sprintf(bufferVars, "Ief = %3.2f", ieficaz);
452                         displayText(2, 1, Font_11x18, bufferVars);
453                         sprintf(bufferVars, "kWh = %3.2f", energia);
454                         displayText(3, 1, Font_11x18, bufferVars);
455                         /* ---------------------- */
456 
457                         if (HAL_GPIO_ReadPin(SW1_GPIO_Port, SW1_Pin) ==
    ↪  GPIO_PIN_SET) {
458                                 HAL_Delay(10);
459                                 while (HAL_GPIO_ReadPin(SW1_GPIO_Port,
    ↪  SW1_Pin) == GPIO_PIN_SET);
460                                 HAL_Delay(10);
461 
462                                 currentState = ST_MEAS_2;
463                                 setScreen(Clean);
464                         } else if (HAL_GPIO_ReadPin(SW2_GPIO_Port,
    ↪  SW2_Pin)         == GPIO_PIN_SET) {
465                                 HAL_Delay(10);
466                                 while (HAL_GPIO_ReadPin(SW2_GPIO_Port,
    ↪  SW2_Pin) == GPIO_PIN_SET);
467                                 HAL_Delay(10);
468 
469                                 currentMenuPage = 0;
470                                 currentState = ST_MENU_PRE;
471                         }
472                         break;
473 
474                         /* ---------------------------- */
475                         // State: MEASURE Page 2
476                         /* ---------------------------- */
477                 case ST_MEAS_2:
478                         /* --- DISPLAY UPDATE --- */
479                         ssd1306_DrawRectangle(1, 1, SSD1306_WIDTH - 1,
    ↪  SSD1306_HEIGHT - 1, White);
480                         sprintf(bufferVars, "P = %3.2f W", potAtiva);
481                         displayText(1, 1, Font_11x18, bufferVars);
482                         sprintf(bufferVars, "Q = %3.2f VAr", potReativa);
483                         displayText(2, 1, Font_11x18, bufferVars);
484                         sprintf(bufferVars, "S = %3.2f VA", potAparente);
485                         displayText(3, 1, Font_11x18, bufferVars);
```

```
486                         /* ---------------------- */
487
488                         if (HAL_GPIO_ReadPin(SW1_GPIO_Port, SW1_Pin) ==
    ↪  GPIO_PIN_SET) {
489                                 HAL_Delay(10);
490                                 while (HAL_GPIO_ReadPin(SW1_GPIO_Port,
    ↪  SW1_Pin) == GPIO_PIN_SET);
491                                 HAL_Delay(10);
492
493                                 currentState = ST_MEAS_3;
494                                 setScreen(Clean);
495                         } else if (HAL_GPIO_ReadPin(SW2_GPIO_Port,
    ↪  SW2_Pin)         == GPIO_PIN_SET) {
496                                 HAL_Delay(10);
497                                 while (HAL_GPIO_ReadPin(SW2_GPIO_Port,
    ↪  SW2_Pin) == GPIO_PIN_SET);
498                                 HAL_Delay(10);
499
500                                 currentMenuPage = 0;
501                                 currentState = ST_MENU_PRE;
502                         }
503                         break;
504
505                         /* ---------------------------- */
506                         // State: MEASURE Page 3
507                         /* ---------------------------- */
508                 case ST_MEAS_3:
509                         /* --- DISPLAY UPDATE --- */
510                         ssd1306_DrawRectangle(1, 1, SSD1306_WIDTH - 1,
    ↪  SSD1306_HEIGHT - 1, White);
511                         sprintf(bufferVars, "FP = %3.2f", fatorPotencia);
512                         displayText(1, 1, Font_11x18, bufferVars);
513                         sprintf(bufferVars, "phi = %3.2f", phi);
514                         displayText(2, 1, Font_11x18, bufferVars);
515                         /* ---------------------- */
516
517                         if (HAL_GPIO_ReadPin(SW1_GPIO_Port, SW1_Pin) ==
    ↪  GPIO_PIN_SET) {
518                                 HAL_Delay(10);
519                                 while (HAL_GPIO_ReadPin(SW1_GPIO_Port,
    ↪  SW1_Pin) == GPIO_PIN_SET);
```

```
520                                 HAL_Delay(10);
521
522                                 currentState = ST_MEAS_1;
523                                 setScreen(Clean);
524                             } else if (HAL_GPIO_ReadPin(SW2_GPIO_Port,
    ↪  SW2_Pin)         == GPIO_PIN_SET) {
525                                 HAL_Delay(10);
526                                 while (HAL_GPIO_ReadPin(SW2_GPIO_Port,
    ↪  SW2_Pin) == GPIO_PIN_SET);
527                                 HAL_Delay(10);
528
529                                 currentMenuPage = 0;
530                                 currentState = ST_MENU_PRE;
531                                 setScreen(Clean);
532                             }
533                             break;
534
535                             /* ---------------------------- */
536                             // State: WIFI
537                             /* ---------------------------- */
538                         case ST_WIFI:
539                             /* --- DISPLAY UPDATE --- */
540                             ssd1306_DrawRectangle(1, 1, SSD1306_WIDTH - 1,
    ↪  SSD1306_HEIGHT - 1, White);
541                             displayText(1, 1, Font_11x18, "  W I F I  ");
542                             displayText(2, 1, Font_11x18, "           ");
543                             displayText(3, 1, Font_11x18, "Conecting..");
544                             /* ---------------------- */
545
546                             wifiSwitch = false;
547                             // TODO: User input SSID and PWD
548                             ESP_Init("NomeDaRede", "Senha");
549
550                             // TODO: WIFI ON/OFF
551                             displayText(3, 1, Font_11x18, " Success!! ");
552                             wifiSwitch = true;
553
554                             // TODO: Transition without HAL_Delay
555                             HAL_Delay(500);
556
557                             currentMenu = MENU_MEASURE;
```

```
558                         currentMenuPage = 0;
559                         currentState = ST_MENU_PRE;
560                         setScreen(Clean);
561                         break;
562 
563                         /* ---------------------------- */
564                         // State: SDCARD
565                         /* ---------------------------- */
566                 case ST_SDCARD:
567                         /* --- DISPLAY UPDATE --- */
568                         ssd1306_DrawRectangle(1, 1, SSD1306_WIDTH - 1,
    ↪  SSD1306_HEIGHT - 1, White);
569                         displayText(1, 1, Font_11x18, "  SD CARD  ");
570                         displayText(2, 1, Font_11x18, "           ");
571                         /* ---------------------- */
572 
573                         // Switch SD CARD on/off:
574                         if (sdCardSwitch == false) {
575                                 sdCardSwitch = true;
576                                 sdcard_Init();
577                                 displayText(3, 1, Font_11x18, " Enabled!
    ↪  ");
578                         } else {
579                                 sdCardSwitch = false;
580                                 displayText(3, 1, Font_11x18, " Disabled!
    ↪  ");
581                         }
582 
583                         HAL_Delay(500);
584 
585                         currentMenu = MENU_MEASURE;
586                         currentMenuPage = 0;
587                         currentState = ST_MENU_PRE;
588                         setScreen(Clean);
589                         break;
590                 }
591                 /*=============================================*/
592                 /* END OF STATE MACHINE */
593                 /*=============================================*/
594 
595                 /* USER CODE END WHILE */
```

```
596         }
597 }
598 
599 /**
600  * @brief System Clock Configuration
601  * @retval None
602  */
603 void SystemClock_Config(void) {
604         RCC_OscInitTypeDef RCC_OscInitStruct = { 0 };
605         RCC_ClkInitTypeDef RCC_ClkInitStruct = { 0 };
606         RCC_PeriphCLKInitTypeDef PeriphClkInit = { 0 };
607 
608         /** Initializes the RCC Oscillators according to the specified
    ↪  parameters
609          * in the RCC_OscInitTypeDef structure.
610          */
611         RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE
612                         | RCC_OSCILLATORTYPE_LSE;
613         RCC_OscInitStruct.HSEState = RCC_HSE_ON;
614         RCC_OscInitStruct.HSEPredivValue = RCC_HSE_PREDIV_DIV1;
615         RCC_OscInitStruct.LSEState = RCC_LSE_ON;
616         RCC_OscInitStruct.HSIState = RCC_HSI_ON;
617         RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
618         RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
619         RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL9;
620         if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK) {
621                 Error_Handler();
622         }
623 
624         /** Initializes the CPU, AHB and APB buses clocks
625          */
626         RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK |
    ↪  RCC_CLOCKTYPE_SYSCLK
627                         | RCC_CLOCKTYPE_PCLK1 | RCC_CLOCKTYPE_PCLK2;
628         RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
629         RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
630         RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
631         RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
632 
633         if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) !=
    ↪  HAL_OK) {
```

```
634                 Error_Handler();
635         }
636         PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_RTC |
    ↪  RCC_PERIPHCLK_ADC;
637         PeriphClkInit.RTCClockSelection = RCC_RTCCLKSOURCE_LSE;
638         PeriphClkInit.AdcClockSelection = RCC_ADCPCLK2_DIV8;
639         if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK) {
640                 Error_Handler();
641         }
642 }
643
644 /**
645  * @brief ADC1 Initialization Function
646  * @param None
647  * @retval None
648  */
649 static void MX_ADC1_Init(void) {
650         ADC_MultiModeTypeDef multimode = { 0 };
651         ADC_ChannelConfTypeDef sConfig = { 0 };
652
653         /** Common config          */
654         hadc1.Instance = ADC1;
655         hadc1.Init.ScanConvMode = ADC_SCAN_DISABLE;
656         hadc1.Init.ContinuousConvMode = ENABLE;
657         hadc1.Init.DiscontinuousConvMode = DISABLE;
658         hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
659         hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
660         hadc1.Init.NbrOfConversion = 1;
661         if (HAL_ADC_Init(&hadc1) != HAL_OK) {
662                 Error_Handler();
663         }
664
665         /** Configure the ADC multi-mode          */
666         multimode.Mode = ADC_DUALMODE_REGSIMULT;
667         if (HAL_ADCEx_MultiModeConfigChannel(&hadc1, &multimode) !=
    ↪  HAL_OK) {
668                 Error_Handler();
669         }
670
671         /** Configure Regular Channel          */
672         sConfig.Channel = ADC_CHANNEL_0;
```

```
673         sConfig.Rank = ADC_REGULAR_RANK_1;
674         sConfig.SamplingTime = ADC_SAMPLETIME_239CYCLES_5;
675         if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK) {
676                 Error_Handler();
677         }
678 }
679
680 /**
681  * @brief ADC2 Initialization Function
682  * @param None
683  * @retval None
684  */
685 static void MX_ADC2_Init(void) {
686         ADC_ChannelConfTypeDef sConfig = { 0 };
687
688         /** Common config          */
689         hadc2.Instance = ADC2;
690         hadc2.Init.ScanConvMode = ADC_SCAN_DISABLE;
691         hadc2.Init.ContinuousConvMode = ENABLE;
692         hadc2.Init.DiscontinuousConvMode = DISABLE;
693         hadc2.Init.ExternalTrigConv = ADC_SOFTWARE_START;
694         hadc2.Init.DataAlign = ADC_DATAALIGN_RIGHT;
695         hadc2.Init.NbrOfConversion = 1;
696         if (HAL_ADC_Init(&hadc2) != HAL_OK) {
697                 Error_Handler();
698         }
699
700         /** Configure Regular Channel          */
701         sConfig.Channel = ADC_CHANNEL_1;
702         sConfig.Rank = ADC_REGULAR_RANK_1;
703         sConfig.SamplingTime = ADC_SAMPLETIME_239CYCLES_5;
704         if (HAL_ADC_ConfigChannel(&hadc2, &sConfig) != HAL_OK) {
705                 Error_Handler();
706         }
707 }
708
709 /**
710  * @brief I2C1 Initialization Function
711  * @param None
712  * @retval None
713  */
```

```
714 static void MX_I2C1_Init(void) {
715         hi2c1.Instance = I2C1;
716         hi2c1.Init.ClockSpeed = 100000;
717         hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
718         hi2c1.Init.OwnAddress1 = 0;
719         hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
720         hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
721         hi2c1.Init.OwnAddress2 = 0;
722         hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
723         hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
724         if (HAL_I2C_Init(&hi2c1) != HAL_OK) {
725                 Error_Handler();
726         }
727 }
728
729 /**
730  * @brief RTC Initialization Function
731  * @param None
732  * @retval None
733  */
734 static void MX_RTC_Init(void) {
735         RTC_TamperTypeDef sTamper = { 0 };
736
737         /** Initialize RTC Only         */
738         hrtc.Instance = RTC;
739         hrtc.Init.AsynchPrediv = RTC_AUTO_1_SECOND;
740         hrtc.Init.OutPut = RTC_OUTPUTSOURCE_NONE;
741         if (HAL_RTC_Init(&hrtc) != HAL_OK) {
742                 Error_Handler();
743         }
744
745     /** Enable the RTC Tamper         */
746         sTamper.Tamper = RTC_TAMPER_1;
747         sTamper.Trigger = RTC_TAMPERTRIGGER_LOWLEVEL;
748         if (HAL_RTCEx_SetTamper(&hrtc, &sTamper) != HAL_OK) {
749                 Error_Handler();
750         }
751 }
752
753 /**
754  * @brief SPI1 Initialization Function
```

```
755  * @param None
756  * @retval None
757  */
758 static void MX_SPI1_Init(void) {
759         hspi1.Instance = SPI1;
760         hspi1.Init.Mode = SPI_MODE_MASTER;
761         hspi1.Init.Direction = SPI_DIRECTION_2LINES;
762         hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
763         hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
764         hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
765         hspi1.Init.NSS = SPI_NSS_SOFT;
766         hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_8;
767         hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
768         hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
769         hspi1.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
770         hspi1.Init.CRCPolynomial = 10;
771         if (HAL_SPI_Init(&hspi1) != HAL_OK) {
772                 Error_Handler();
773         }
774 }
775
776 /**
777  * @brief TIM3 Initialization Function
778  * @param None
779  * @retval None
780  */
781 static void MX_TIM3_Init(void) {
782         TIM_ClockConfigTypeDef sClockSourceConfig = { 0 };
783         TIM_MasterConfigTypeDef sMasterConfig = { 0 };
784
785         htim3.Instance = TIM3;
786         htim3.Init.Prescaler = 1151;
787         htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
788         htim3.Init.Period = 62499;
789         htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
790         htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
791         if (HAL_TIM_Base_Init(&htim3) != HAL_OK) {
792                 Error_Handler();
793         }
794         sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
```

```
795         if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) !=
    ↪  HAL_OK) {
796                 Error_Handler();
797         }
798         sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
799         sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
800         if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig)
801                         != HAL_OK) {
802                 Error_Handler();
803         }
804 }
805
806 /**
807  * @brief USART1 Initialization Function
808  * @param None
809  * @retval None
810  */
811 static void MX_USART1_UART_Init(void) {
812         huart1.Instance = USART1;
813         huart1.Init.BaudRate = 115200;
814         huart1.Init.WordLength = UART_WORDLENGTH_8B;
815         huart1.Init.StopBits = UART_STOPBITS_1;
816         huart1.Init.Parity = UART_PARITY_NONE;
817         huart1.Init.Mode = UART_MODE_TX_RX;
818         huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
819         huart1.Init.OverSampling = UART_OVERSAMPLING_16;
820         if (HAL_UART_Init(&huart1) != HAL_OK) {
821                 Error_Handler();
822         }
823 }
824
825 /**
826  * @brief USART3 Initialization Function
827  * @param None
828  * @retval None
829  */
830 static void MX_USART3_UART_Init(void) {
831         huart3.Instance = USART3;
832         huart3.Init.BaudRate = 115200;
833         huart3.Init.WordLength = UART_WORDLENGTH_8B;
834         huart3.Init.StopBits = UART_STOPBITS_1;
```

```
835         huart3.Init.Parity = UART_PARITY_NONE;
836         huart3.Init.Mode = UART_MODE_TX_RX;
837         huart3.Init.HwFlowCtl = UART_HWCONTROL_NONE;
838         huart3.Init.OverSampling = UART_OVERSAMPLING_16;
839         if (HAL_UART_Init(&huart3) != HAL_OK) {
840                 Error_Handler();
841         }
842 }
843
844 /**
845  * Enable DMA controller clock
846  */
847 static void MX_DMA_Init(void) {
848
849         /* DMA controller clock enable */
850         __HAL_RCC_DMA1_CLK_ENABLE();
851
852         /* DMA interrupt init */
853         /* DMA1_Channel1_IRQn interrupt configuration */
854         HAL_NVIC_SetPriority(DMA1_Channel1_IRQn, 0, 0);
855         HAL_NVIC_EnableIRQ(DMA1_Channel1_IRQn);
856
857 }
858
859 /**
860  * @brief GPIO Initialization Function
861  * @param None
862  * @retval None
863  */
864 static void MX_GPIO_Init(void) {
865         GPIO_InitTypeDef GPIO_InitStruct = { 0 };
866
867         /* GPIO Ports Clock Enable */
868         __HAL_RCC_GPIOC_CLK_ENABLE();
869         __HAL_RCC_GPIOD_CLK_ENABLE();
870         __HAL_RCC_GPIOA_CLK_ENABLE();
871         __HAL_RCC_GPIOB_CLK_ENABLE();
872
873         /*Configure GPIO pin Output Level */
874         HAL_GPIO_WritePin(GPIOB, LED2_Pin | LED1_Pin, GPIO_PIN_RESET);
875
```

```
876         /*Configure GPIO pins : LED2_Pin LED1_Pin */
877         GPIO_InitStruct.Pin = LED2_Pin | LED1_Pin;
878         GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
879         GPIO_InitStruct.Pull = GPIO_NOPULL;
880         GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
881         HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
882
883         /*Configure GPIO pins : SW1_Pin SW2_Pin */
884         GPIO_InitStruct.Pin = SW1_Pin | SW2_Pin;
885         GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
886         GPIO_InitStruct.Pull = GPIO_NOPULL;
887         HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
888 }
889
```

APÊNDICE C – Código fonte da página HTML

O código abaixo foi desenvolvido para exibição dos gráficos dos parâmetros medidos pelo sistema.

```
<!DOCTYPE html>
<html>
  <head>
    <title>EnergyMeter UFABC</title>
    <style>
      body {
        background-color: #121212;
        color: white;
        font-family: Arial, sans-serif;
      }
      .chart-container {
        width: 100%;
        max-width: 100%;
        margin: 0 auto;
        padding: 10px;
      }
      .chart {
        width: 100%;
        height: 300px;
      }
      h1 {
        text-align: center;
      }
      @media (min-width: 600px) {
        .chart {
          height: 500px;
        }
      }
    </style>
    <script type="text/javascript"
    ↪  src="https://www.gstatic.com/charts/loader.js"></script>
    <script type="text/javascript">
```

```
32      google.charts.load("current", { packages: ["corechart"] });
33
34      async function fetchData() {
35        const response = await fetch("/data");
36        const data = await response.json();
37
38        const field1Data = [["Data", "Field1"]];
39        const field2Data = [["Data", "Field2"]];
40        const field3Data = [["Data", "Field3"]];
41        const field4Data = [["Data", "Field4"]];
42
43        data.feeds.forEach((feed) => {
44          field1Data.push([new Date(feed.created_at),
            ↪ parseFloat(feed.field1)]);
45          field2Data.push([new Date(feed.created_at),
            ↪ parseFloat(feed.field2)]);
46          field3Data.push([new Date(feed.created_at),
            ↪ parseFloat(feed.field3)]);
47          field4Data.push([new Date(feed.created_at),
            ↪ parseFloat(feed.field4)]);
48        });
49
50        drawChart(
51          field1Data,
52          "Tensão eficaz (V)",
53          "field1_chart",
54          { min: 100, max: 150 },
55          "#1E88E5"
56        );
57        drawChart(
58          field2Data,
59          "Corrente eficaz (A)",
60          "field2_chart",
61          { min: 0, max: 5 },
62          "#D32F2F"
63        );
64        drawChart(
65          field3Data,
66          "Energia consumida (Wh)",
67          "field3_chart",
68          { min: 0, max: 200 },
```

```
69            "#388E3C"
70          );
71        }
72
73        async function updateChartsPeriodically(intervalMs) {
74          await fetchData();
75
76          setInterval(async () => {
77            await fetchData();
78          }, intervalMs);
79        }
80
81        function drawChart(
82          dataArray,
83          fieldName,
84          elementId,
85          vAxisRange,
86          lineColor
87        ) {
88          const data = google.visualization.arrayToDataTable(dataArray);
89          const options = {
90            title: fieldName,
91            hAxis: {
92              title: "",
93              format: "dd/MM HH:mm",
94              textStyle: { color: "#FFF" },
95              titleTextStyle: { color: "#FFF" },
96            },
97            vAxis: {
98              title: fieldName,
99              viewWindow: {
100               min: vAxisRange.min,
101               max: vAxisRange.max,
102             },
103             textStyle: { color: "#FFF" },
104             titleTextStyle: { color: "#FFF" },
105           },
106           backgroundColor: "#121212",
107           titleTextStyle: { color: "#FFF" },
108           legend: "none",
109           colors: [lineColor],
```

```
110           chartArea: { backgroundColor: "#212121" },
111           lineWidth: 3,
112           pointSize: 1,
113           areaOpacity: 0.25,
114         };
115         const chart = new google.visualization.AreaChart(
116           document.getElementById(elementId)
117         );
118         chart.draw(data, options);
119       }
120
121       google.charts.load("current", { packages: ["corechart"] });
122
123       google.charts.setOnLoadCallback(function () {
124         fetchData();
125         updateChartsPeriodically(1000);
126       });
127
128       window.addEventListener("resize", fetchData);
129     </script>
130   </head>
131   <body>
132     <h1>EnergyMeter</h1>
133     <div class="chart-container">
134       <div id="field1_chart" class="chart"></div>
135     </div>
136     <div class="chart-container">
137       <div id="field2_chart" class="chart"></div>
138     </div>
139     <div class="chart-container">
140       <div id="field3_chart" class="chart"></div>
141     </div>
142   </body>
143 </html>
```

Anexos

ANEXO A – Datasheet do transdutor SCT-013

Split core current transformer

Model: SCT-013 **Rated input current: 5A/100A**

Characteristics: Openning size: 13mm*13mm,
Non-linerity±3%（10%—120% of rated input current）
1m leading wire, standard Φ3.5 three core plug output.
Current output type and voltage output type (voltage output type built-in sampling resistor)

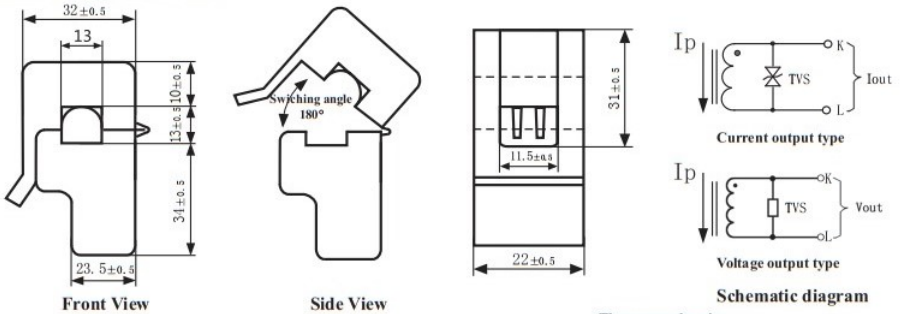
Purpose: Used for current measurement, monitor and protection for AC motor, lighting equipment, air compressor etc

Core material: ferrite

Mechanical strength: the number of switching is not less than 1000 times(test at 25℃)

Safty index: Dielectric strength(between shell and output)1000V AC/1min
Fire resistance property: In accordance with UL94-Vo
Work temperature: -25℃～+70℃

Outline size diagram: (in mm)



Front View Side View Schematic diagram

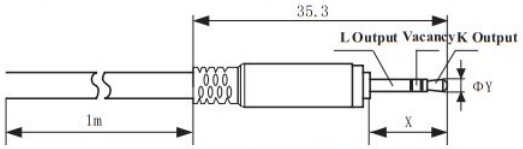


Diagram for standard three-core plug

Three-core plug size

	X	Y	
2.5mm Audio Plug	11.9	2.5	Optional
3.5mm Audio Plug	15.0	3.5	standard

Table of technical parameter:

Model	SCT-013-000	SCT-013-005	SCT-013-010	SCT-013-015	SCT-013-020
Input current	0-100A	0-5A	0-10A	0-15A	0-20A
Output type	0-50mA	0-1V	0-1V	0-1V	0-1V
Model	SCT-013-025	SCT-013-030	SCT-013-050	SCT-013-060	SCT-013-000V
Input current	0-25A	0-30A	0-50A	0-60A	0-100A
Output type	0-1V	0-1V	0-1V	0-1V	0-1V

※ Output type: voltage output type built-in sampling resistor, current output type built-in protective diode.

Tel: 86-13933609279 Fax: 86-7929499-804 skype:macymeng1
Web (China): www.yhdc.com Web (Latvia): www.yhdc.lv

ANEXO B – Esquema eletrônico da BluePill