CENTRO PAULO SOUZA
FATEC SANTO ANDRÉ
TECNOLOGIA EM ELETRÔNICA AUTOMOTIVA
TECNOLOGIA EM MECÂNICA AUTOMOBILÍSTICA

FELIPE BATISTA GONÇALVES
TIAGO BARBOSA MENDONÇA DA SILVA

PLATAFORMA DE DIAGNÓSTICO PARA BATERIAS AUTOMOTIVAS 12 VOLTS: CHUMBO-ÁCIDO

SANTO ANDRÉ
2023

FELIPE BATISTA GONÇALVES
TIAGO BARBOSA MENDONÇA DA SILVA

PLATAFORMA DE DIAGNÓSTICO PARA BATERIAS AUTOMOTIVAS 12 VOLTS: CHUMBO-ÁCIDO

Trabalho de Conclusão de Curso apresentado aos cursos de Tecnologia em Mecânica Automobilística e Eletrônica Automotiva da Fatec Santo André, como requisito parcial a obtenção dos títulos de Tecnólogo em Mecânica Automobilística e Tecnólogo em Eletrônica Automotiva.

Prof. Orientador: Dr. Edson C. Kitani

SANTO ANDRÉ
2023

Fatec CENTRO PAULA SOUZA GOVERNO DO ESTADO DE SÃO PAULO

Faculdade de Tecnologia de Santo André

LISTA DE PRESENÇA

Santo André, 13 DE JUNHO DE 2023

LISTA DE PRESENÇA REFERENTE À APRESENTAÇÃO DO TRABALHO DE CONCLUSÃO DE CURSO COM O TEMA: **"PLATAFORMA DE DIAGNÓSTICO PARA BATERIAS AUTOMOTIVAS 12 VOLTS: CHUMBO-ÁCIDO"** DOS ALUNOS DO 6º SEMESTRE DESTA U.E.

BANCA

PRESIDENTE:
PROF. EDSON CAORU KITANI_____

MEMBROS:
PROF. WESLLEY MEDEIROS TORRES_____

PROF. FERNANDO GARUP DALBO_____

ALUNOS:

FELIPE BATISTA GONÇALVES Felipe Batista Gonçalves

TIAGO BARBOSA MENDONÇA DA SILVA Tiago Barbosa Mendonça da Silva

www.fatecsantoandre.com.br fatecstdo@gmail.com
Rua Prefeito Justino Paixão, 150 – Centro – Santo André – SP - CEP: 09020-130
Fone (0xx11) 4437-2215

FICHA CATALOGRÁFICA

G635p

Gonçalves, Felipe Batista

Plataforma de diagnóstico para baterias automotivas 12 volts: chumbo-ácido / Felipe Batista Gonçalves, Tiago Barbosa Mendonça da Silva. - Santo André, 2023. – 90f: il.

Trabalho de Conclusão de Curso – FATEC Santo André.
Curso de Tecnologia em Eletrônica Automotiva, 2023.

Orientador: Prof. Dr. Edson C. Kitani

1. Eletrônica. 2. Plataforma de diagnóstico. 3. Baterias chumbo-ácido. 4. Recarga. 5. Tecnologia. 6. Baterias automotivas. 7. Teste. 8. Eficiência energética. 9. Desenvolvimento. I. Silva, Tiago Barbosa Mendonça da. II. Plataforma de diagnóstico para baterias automotivas 12 volts: chumbo-ácido.

629.2

A minha mãe Lucimara e minha futura esposa Talita, que sempre estiveram ao meu lado.

Felipe Batista.

Aos meus pais, Gilvan e Gilvania, minha irmã, Ingrid e ao meu padrasto Jeferson, família e amigos.

Tiago Barbosa.

AGRADECIMENTOS

Gostaríamos de expressar nossa gratidão a todos que nos apoiaram durante este projeto. Em primeiro lugar, nossos sinceros agradecimentos aos nossos familiares e amigos que estiveram ao nosso lado durante todo o processo e nos incentivaram a seguir em frente.

Agradecemos também aos nossos colegas de trabalho, que nos ajudaram em todos os momentos e cuja colaboração foi fundamental para o sucesso deste projeto.

Gostaríamos de estender um agradecimento especial ao professor Fernando Garup, cuja total assistência e suporte foram cruciais para o desenvolvimento deste projeto. Além disso, gostaríamos de agradecer ao professor Edson Kitani pela orientação do trabalho e pela ajuda que nos foi fornecida durante toda a jornada de desenvolvimento.

Por fim, gostaríamos de expressar nossa gratidão a todo o corpo docente e funcionários da Fatec de Santo André, que nos proporcionaram um ambiente de aprendizado inspirador e nos apoiaram ao longo deste projeto.

Muito obrigado a todos pela ajuda e pelo apoio ao longo deste projeto. Estamos imensamente gratos por tudo que fizeram por nós.

‘A persistência é o caminho do êxito.’

Charles Chaplin

RESUMO

Com o crescente interesse e necessidade por veículos elétricos, em virtude do aquecimento global, observa-se um intenso uso de baterias de Lithio. Contudo, as baterias de chumbo ácido continuam muito utilizadas dado o baixo custo e fácil fabricação em qualquer lugar do planeta, mas ela ainda apresenta baixo desempenho quando comparado com as baterias de Lithio. Compreender melhor o estado de carga das baterias é uma maneira de aumentar o tempo de vida delas. Este trabalho consiste no desenvolvimento de uma plataforma de diagnóstico para baterias automotivas leves de chumbo-ácido, que visa realizar recargas de alta eficiência energética e testes padronizados sobre as condições das baterias, visto que no mercado há um déficit na realização da recarga de bateria e métodos pouco assertivos no teste. O equipamento elaborado baseia-se em um mecanismo combinatório entre um carregador de baterias, que possui como estrutura uma fonte ATX modificada, e com controle de tensão e corrente, e um testador com resistências fixas e chaveamento por reles. A plataforma de diagnostico proposta possibilita obter um resultado melhor e mais preciso no que diz respeito a condição de vida útil da bateria testada, escolha e o aproveitamento conforme o uso e a redução do descarte inadequado de baterias que gera um alto custo para o setor automobilístico e causa severos impactos ambientais.

Palavras-chave: Baterias automotivas. Recarga de bateria. Teste de bateria. Eficiência energética.

ABSTRACT

With the growing interest and need for electric vehicles due to global warming, there is an intense use of Lithium batteries. However, lead-acid batteries are still widely used due to their low cost and easy manufacturing anywhere on the planet, although they still exhibit lower performance compared to Lithium batteries. Better understanding the state of charge of batteries is a way to increase their lifespan. This work consists of developing a diagnostic platform for lightweight automotive lead-acid batteries, aiming to perform high-energy efficiency recharges and standardized tests on battery conditions, considering the existing deficit in battery recharging in the market and the lack of accurate testing methods. The developed equipment is based on a combined mechanism between a battery charger, which has a modified ATX power supply with voltage and current control, and a tester with fixed resistors and relay switching. The proposed diagnostic platform allows for better and more accurate results regarding the tested battery's lifespan condition, selection, utilization according to usage, and reduction of improper battery disposal, which incurs high costs for the automotive sector and causes severe environmental impacts.

Keywords: Automotive battery. Recharge battery. Battery test. Efficiency.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ADC - Analog-to-Digital Converter

AES - Advanced Encryption Standard

Ah - Ampere-hora

A - Ampere

ACS712 - Allegro MicroSystems' Current Sensor 712

Arduino - Plataforma de prototipagem eletrônica de código aberto

ATX - Advanced Technology Extended

BLE - Bluetooth Low Energy

CCA - Cold Cranking Amps (Amperes de Arranque à Frio)

DAC - Digital-to-Analog Converter

ECC - Elliptic Curve Cryptography

ESP-IDF - Espressif IoT Development Framework

ESP32 - Espressif Systems' Platform 32

FLA - Flooded Lead Acid (Bateria de Chumbo-Ácido)

FreeRTOS - Free Real-Time Operating System

GPIO - General Purpose Input/Output

I2C - Inter-Integrated Circuit

Inmetro - Instituto Nacional de Metrologia, Qualidade e Tecnologia

KB - Kilobytes

LabVIEW - Laboratory Virtual Instrument Engineering Workbench

Li-ion - Lithium-ion (Bateria de Íon-Lítio)

Li-Po - Lithium Polymer (Bateria de Polímero de Lítio)

MB - Megabytes

MHz - Megahertz

mA - Miliamperes

NTC - Negative Temperature Coefficient

PWM - Pulse Width Modulation

Queues - Filas (Comunicação entre tarefas dentro do RTOS)

RC - Reserva de Capacidade

RSA - Rivest-Shamir-Adleman

RTOS - Real-Time Operating System

SHA-2 - Secure Hash Algorithm 2

SPI - Serial Peripheral Interface

SRAM - Static Random Access Memory

UART - Universal Asynchronous Receiver-Transmitter

ULP - Ultra-Low-Power

V - Volts

WiFi - Wireless Fidelity

SUMÁRIO

INTRODUÇÃO

As baterias têm sido usadas por séculos como fonte para armazenar e fornecer energia elétrica através das reações eletroquímicas que ocorrem entre diversos metais quando submetidos a um meio ácido.

Embora muitas tecnologias em baterias tenham surgido desde então, a bateria de chumbo-ácido continua sendo amplamente utilizada, principalmente no setor automotivo desde a década de 1920. Ela oferece confiabilidade, baixo custo e capacidade de fornecer altas correntes de descarga, sendo essencial para fornecer energia à partida dos automóveis.

No entanto, o grande crescimento da eletrificação no ramo automobilístico tem impulsionado o surgimento de novas tecnologias em baterias para atender às novas demandas. Entre essas tecnologias estão as baterias de íon-lítio, que têm maior densidade de energia e são mais leves e compactas, e as baterias de estado sólido, que têm maior durabilidade e segurança.

Apesar do surgimento dessas novas tecnologias, a bateria de chumbo-ácido continua dominando o mercado. No futuro é possível que outras tecnologias em baterias sejam adotadas à medida que a tecnologia evolui e novas demandas surjam no mercado.

1.1 MOTIVAÇÃO

A motivação principal deste trabalho é aplicar os conceitos aprendidos durante o curso de forma prática e didática. Através da integração de conhecimentos em carga e partida, eletrônica analógica e digital. Além do *software* embarcado e visualização de dados em plataforma via LabVIEW, o objetivo é desenvolver uma plataforma de diagnóstico acessível e eficiente para baterias automotivas de chumbo-ácido. Com essa iniciativa, busca-se contribuir para o aprimoramento da manutenção preventiva e corretiva desses sistemas, visando o aumento da vida útil das baterias e redução de gastos com trocas desnecessárias.

1.2 OBJETIVO

O objetivo deste trabalho é desenvolver um testador de baterias mais eficiente, com uma aplicação visual mais detalhada do que os testadores semelhantes já existentes no mercado, a fim de facilitar a *interface* entre a máquina e o operador. Como parte do projeto serão comparados os resultados obtidos pelo *hardware* deste projeto com os testadores disponíveis no mercado.

1.3 ORGANIZAÇÃO

O Capítulo 2 discorrerá sobre algumas referências bibliográficas e um resumo teórico sobre as baterias de chumbo ácido, no Capítulo 3 será apresentado o desenvolvimento do projeto, no Capítulo 4 será apresentado os testes e a análise dos resultados e, finalmente, no Capítulo 5 a conclusão e discussões sobre o projeto apresentado.

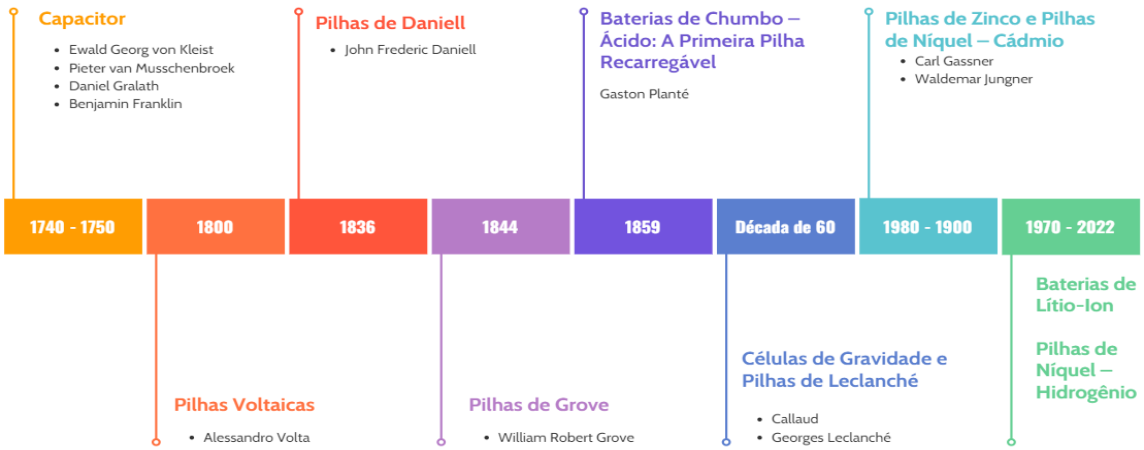
2 REVISÃO BIBLIOGRÁFICA

Este capítulo inicia com uma breve descrição histórica e teórica sobre o funcionamento das baterias de chumbo ácido e a descrição de alguns trabalhos correlatos que inspiraram este projeto.

2.1 Evolução das baterias

A história das baterias é de extrema importância para a sociedade atual, pois esses dispositivos são amplamente utilizados em diversas aplicações do nosso dia a dia. A Figura 1 ilustra uma linha do tempo representando a evolução dos sistemas de armazenamento de cargas elétricas.

Figura 1 - Linha do tempo da história sobre armazenamento de energia.



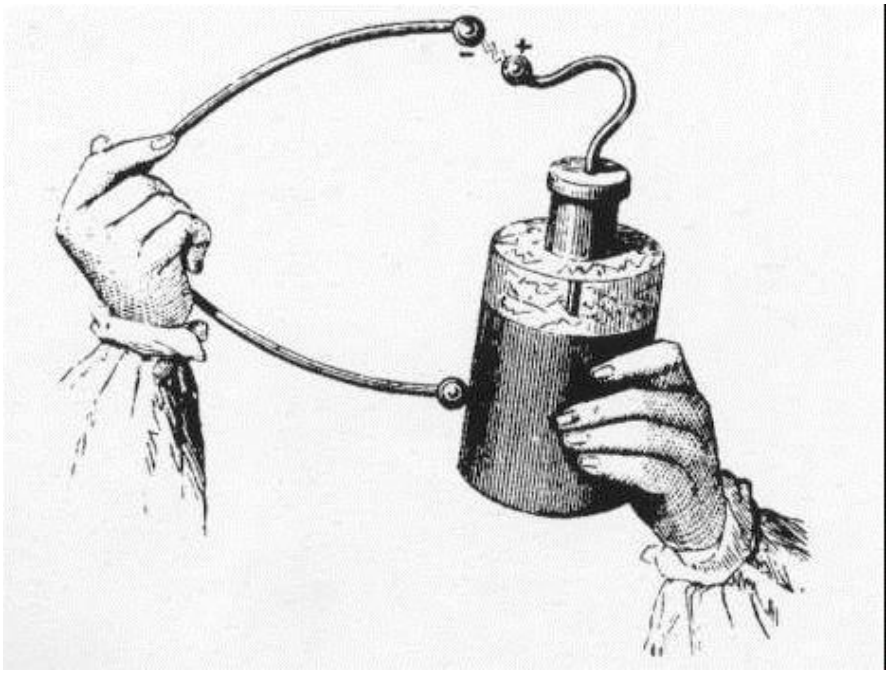
Fonte: Autoria própria, 2023.

Acredita-se que a primeira pilha foi construída há mais de 2.000 anos, onde hoje é a cidade de Bagdá. Essa suposta pilha era constituída por um jarro de barro, preenchido com uma solução de vinagre ou vinho, onde se colocava uma haste de ferro dentro de um cilindro de cobre. Não há registros escritos sobre quem as fabricaram e suas aplicações, mas cientistas acreditam que essas baterias eram usadas no processo que hoje conhecemos por galvanização. (ENERGIZER, 2023)

Os primeiros registros sobre o armazenamento de energia elétrica surgiram no século XVIII, quando o físico alemão Ewald Georg von Kleist conduziu experimentos com uma jarra de vidro cheia de água e um dispositivo eletrostático, descobrindo que as cargas elétricas poderiam ser armazenadas em sua superfície. Seus estudos foram continuados pelo físico holandês Pieter van Musschenbroek que aprimorou a jarra de

vidro, dando origem à Garrafa de Leyden. Benjamin Franklin descobriu que a Garrafa de Leyden (Figura 2) poderia armazenar cargas elétricas na sua superfície, o que foi um importante avanço para o desenvolvimento dos capacitores. (UFRGS, 2022)

Figura 2 - Garrafa de Leyden.



Fonte: Newton Braga, 2022.

De acordo com o site minf.ufpa.br, ainda no século XVIII, o físico italiano Luigi Galvani realizou experimentos que envolveram rãs. Ao inserir diferentes metais no corpo do animal, como um gancho de bronze e um bisturi de ferro, Galvani notou que os músculos do anfíbio se contraiam.

A primeira explicação para o experimento era de que a contração muscular era causada pela eletricidade produzida pelo animal. Posteriormente, o físico italiano Alessandro Volta, concluiu que não era a rã que produzia eletricidade, mas sim a interação dos metais diferentes que gerava eletricidade. (MINF.IFPA.BR, 2023)

Anos depois Alessandro Volta foi o responsável pela criação de pilhas de discos de metais intercalados por um disco de pano embebido em solução de salmoura, que antecedeu as atuais pilhas elétricas. Essa pilha produzia uma corrente elétrica contínua e aumentado o número de discos, observa-se o aumento da diferença de potencial entre os terminais dos eletrodos. A Figura 3 ilustra a imagem da pilha voltaica desenvolvida por Volta em 1800. Nota-se um empilhamento de discos alternados de

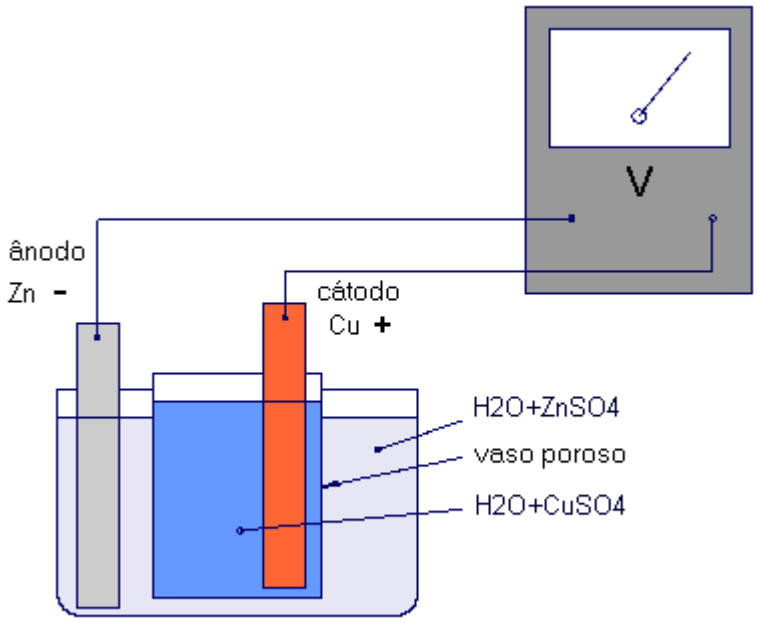
cobre e zinco e nas extremidades da pilha os contatos positivo e negativo da pilha. (BRINGIT.COM.BR, 2022)

Figura 3 – Foto da pilha voltaica de Alessandro Volta.

Fonte: Imagem extraída de https://commons.wikimedia.org/wiki/File:VoltaBattery.JPG.

Em 1836, o químico inglês John Frederic Daniell criou um dispositivo diferente da pilha de Volta. Sua "pilha" consistia em dois eletrodos interligados, formados por um metal imerso em uma solução aquosa de um sal. Essa pilha tinha uma tensão mais estável do que a pilha de Volta e, posteriormente, se tornou amplamente utilizada.

Figura 4 - Exemplo de Experimento do Frederic Daniell.



Fonte: Engenharia Elétrica - UNIFACS, 2023.

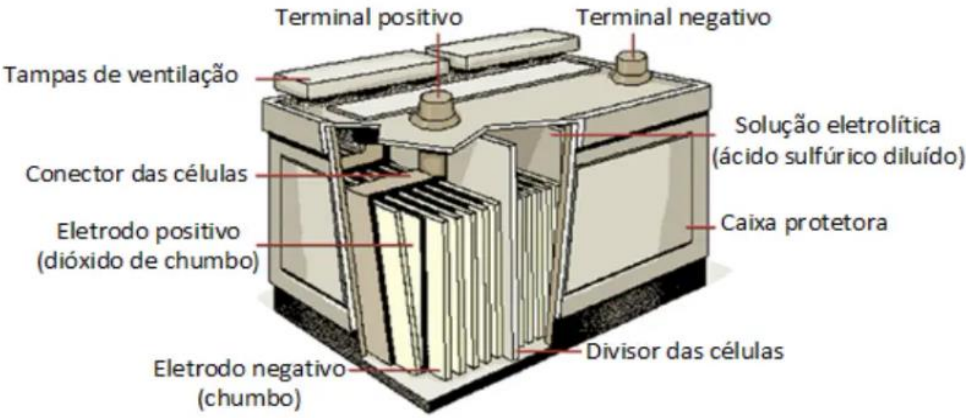
Com todas essas contribuições e descobertas, as pilhas evoluíram e se tornaram mais eficientes. As associações de pilhas, série e paralelo, deram início ao conceito de bateria que hoje é usado e consequentemente, permitiu a criação de dispositivos que utilizam energia elétrica em larga escala. Atualmente, existem diversos tipos de baterias, cada uma com suas características específicas que são utilizadas em equipamentos eletrônicos, veículos, sistemas de energia, entre outros. (BRINGIT.COM.BR, 2021).

2.2 Baterias de Chumbo-Ácido

As baterias de chumbo-ácido recarregáveis foram desenvolvidas na década de 1860 pelo cientista francês Gaston Planté. Elas são compostas por placas de chumbo e alguns aditivos, separadas por papelão ou plástico e imersas em uma solução aquosa de ácido sulfúrico. Essa estrutura é denominada pilha, célula, vaso ou elemento, e sua associação em série resulta na denominação: baterias de chumbo-ácido. Essas pilhas são conectadas em série para gerar diferentes tensões de trabalho, e uma bateria automotiva típica é composta por seis células de 2,13 V cada, conectadas em série, totalizando 12,78 V. Na Figura 5 observa-se uma representação de uma bateria de chumbo ácido automotivo e os seus diferentes elementos que

compõe a construção desse dispositivo. Nota-se que as células estão montadas internamente na caixa protetora e estão conectadas em série, mas somente os terminais positivo e negativo das extremidades da ligação série são expostas para fora da caixa de protetora.

Figura 5 - Bateria de chumbo-ácido.



Fonte: Site Embarcados, 2020.

Durante a descarga, a reação química ocorre com o dióxido de chumbo no cátodo reagindo com o ácido sulfúrico para produzir sulfato de chumbo e água. No ânodo, o chumbo reage com íons de sulfato para também formar sulfato de chumbo. A reação global produz apenas sulfato de chumbo e água. A equação química da descarga pode ser representada por, $Pb + PbO2 + HSO4 \rightarrow PbSO4 + H2O$; onde PbO2 representa o eletrodo de dióxido de chumbo (cátodo), Pb representa o eletrodo de chumbo (ânodo) e H2SO4 é o ácido sulfúrico utilizado como eletrólito.

No processo de recarga, a reação química ocorre de forma inversa, com o sulfato de chumbo sendo convertido novamente em chumbo e com a água sendo transformada em ácido sulfúrico. A equação química da recarga pode ser representada por, $PbSO4 + H2O \rightarrow Pb + PbO2 + HSO4$; onde os produtos da reação química são os mesmos da descarga, mas os reagentes estão invertidos. Essas reações podem ser revertidas aplicando-se tensão e uma corrente elétrica contínua nos polos da bateria, permitindo a sua recarga.

No ramo automotivo, a bateria de chumbo-ácido é utilizada principalmente para fornecer alta corrente por um curto período, o que se faz necessário para que o motor de arranque funcione e o motor de combustão comece a trabalhar. Além disso, a

bateria armazena a energia fornecida pelo alternador durante o funcionamento do veículo e fornecer energia para os dispositivos do carro quando necessário.

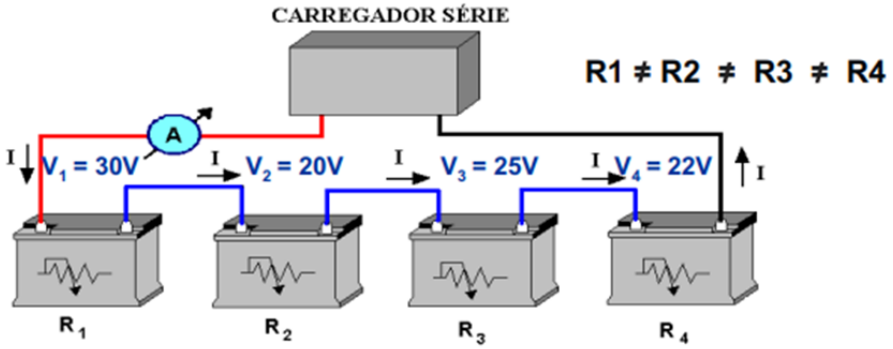
2.3 Métodos de Carregamento

É fundamental carregar corretamente as baterias de chumbo-ácido para preservar sua vida útil, o que requer controle e monitoramento adequados. A redução do tempo de carga pode ser obtida por meio do uso de correntes de carga maiores e diferentes estágios de tensão. Em geral, existem dois métodos principais para carregar completamente uma bateria:

- Corrente Constante
- Tensão Constante

A corrente constante consiste em carregar a bateria com 10% a 20% de sua capacidade nominal, a fim de assegurar que ela não sobrecarregue. Esse processo vale para recargas de baterias em série conforme ilustra a figura 6 e devem ser usadas apenas na fabricação das baterias. Depois de usadas, as baterias sofrem alterações em suas resistências internas e consequentemente as tensões de recarga também serão diferentes quando submetidas a uma mesma corrente.

Figura 6 - Carregador de baterias série.



Fonte: Manual de treinamento Tudor, 2017

A Tabela 1 mostra a relação entre o tempo de carga de uma bateria (em horas) em função do seu estado interna de carga, baseado apenas na tensão da bateria medida sem carga.

Tabela 1 - Tempo de Recarga

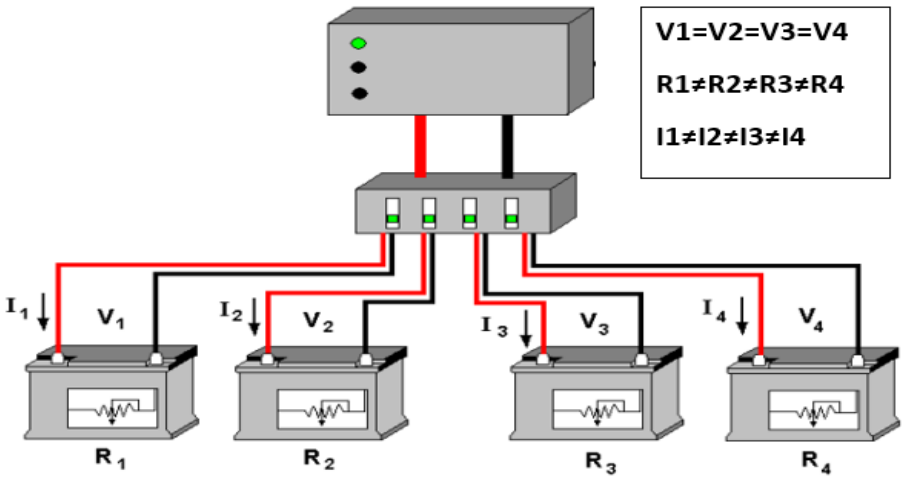
Tensão da bateria em vazio (volts)	Tempo de recarga (horas)
12,00 a 12,20	4,5

11,80 a 11,99	7,0
11,50 a 11,79	9,0
11,00 a 11,49	11,0
Baterias profundamente descarregadas	15,0

Fonte: Manual de Baterias Bosch.

Um outro método de carregamento é o da tensão constante, ilustrado na figura 7, o qual consiste em carregar a bateria utilizando uma tensão fixa conforme indicado pelo fabricante, por um determinado período e isso pode ser aplicado com mais de uma bateria de diferentes capacidades desde que sejam ligadas em paralelo. Nesse caso, a corrente irá subir até o momento que a tensão da bateria alcançar o valor estipulado e manter-se constante; análogo ao funcionamento do alternador dos veículos.

Figura 7 - Carregador de baterias paralelo



Fonte: Manual de treinamento Tudor, 2017

Tabela 2 - Tempo de recarga por tensão constante

Tensão da Bateria em Vazio (volts)	Tempo de Recarga (horas)
12,00 a 12,20	6 a 12
11,80 a 11,99	10 a 16
11,50 a 11,79	16 a 20
11,00 a 11,49	20 a 24
Baterias profundamente descarregadas	24 a 30

Fonte: Manual das baterias Bosch

De acordo com o manual da Bosch sobre baterias de chumbo-ácido, a temperatura durante o processo de recarga não deverá ultrapassar 50° C, uma vez que em temperaturas superiores a perda de água torna-se altamente prejudicial.

2.4 Método de Teste

O método de teste de baterias é de extrema importância para garantir a qualidade e segurança dos produtos comercializados no mercado brasileiro. Desde 2011, todas as baterias vendidas no país são certificadas com o selo do Inmetro, que garante que diferentes marcas foram submetidas a testes e estão dentro de um processo de normatização rigoroso.

Durante o processo de certificação, diversos testes e medições são realizados para conferir se a bateria cumpre as especificações indicadas no seu rótulo. Alguns dos testes mais importantes incluem a verificação do número de Amperes/Hora, reserva de capacidade (RC) e CCA. Esse último parâmetro aplica-se em baterias automotivas, foco do trabalho apresentado.

Um dos principais parâmetros medidos é o CCA, sigla em inglês para Cold Cranking Amps, que significa Amperes de Arranque à Frio. Esse valor é essencial, pois quanto mais frio estiver, mais energia é necessária para ligar o motor. Além disso, a reação química na bateria de chumbo-ácido é menos eficiente no frio, tornando esse parâmetro ainda mais crucial. Quanto maior for o valor do CCA, melhor será a condição da bateria para dar a partida, garantindo o funcionamento do veículo mesmo em condições de baixa temperatura.

2.5 Trabalhos Semelhantes

Durante o desenvolvimento deste projeto foram consultados nos repositórios das Universidades alguns trabalhos cuja palavra-chave referia-se às baterias de chumbo ácido. Foram encontrados vários, mas foram selecionados apenas três, cujo conteúdo era relevante para este projeto.

Em (CUNHA & SENDA, 2015), os autores desenvolveram um projeto de monitoração do estado das baterias a partir de um dispositivo móvel, que neste caso era um telefone celular (Smartphone). O projeto envolveu o desenvolvimento do *hardware* de monitoração baseado no Arduíno Uno e uma *interface* de comunicação

via Bluetooth. Fundamentalmente, o projeto se limitava a monitorar o estado da carga da bateria via medição direta da tensão e apresentava o resultado num aplicativo desenvolvido para Android via uma plataforma conhecida como APP Inventor.

Os autores também desenvolveram uma tabela de erros que enviava mensagens para alertar o usuário sobre as ações que deveriam ser tomadas em virtude desses erros. Entretanto, o grande mérito do trabalho foi integrar um sistema de comunicação em Bluetooth para monitorar grandezas físicas e apresentá-las num aplicativo desenvolvido por eles num sistema Android.

Figura 8 – Tabela de mensagens de erros gerados no projeto de (CUNHA & SENDA, 2015)

Erro	Mensagem para dispositivo móvel
ERRO 1 e 6	Queda de tensão anormal na partida detectada. É possível que a bateria esteja atingindo sua vida útil. MANUTENÇÃO RECOMENDADA
ERRO 2 e 10	Tensão crítica atingida. Bateria Arriada. RECOMENDA-SE A TROCA DA BATERIA
ERRO 3	Bateria com menos de 50% da carga. Problemas na partida serão mais frequentes. MANUTENÇÃO RECOMENDADA
ERRO 4 e X	Alto consumo de corrente detectado. Possível consumidor acionado mesmo com motor desligado. Verifique!!!“
ERRO 5 e 7	Tensão Zero detectada. Possível falha do sensor de tensão da bateria. VERIFIQUE
ERRO 8	Fuga de corrente detectada. MANUTENÇÃO RECOMENDADA
ERRO 9	Alto consumo de corrente detectado. Possível consumidor acionado mesmo chave fora do contato. VERIFIQUE

Fonte: (CUNHA & SENDA, 2015)

Figura 9 – Tabela de testes do estado da bateria conforme (CUNHA & SENDA, 2015).

Variável	Código	Descrição	Valor
Estado Bateria Tensão	E_B_T_ZER	Tensão zero	T=0 V
	E_B_T_PDF	Tensão inferior a 8,5V durante partida	T=8,50 V
	E_B_T_DES	Bateria descarregada	T=11,90 V
	E_B_T_025	25% de carga	T=12,00 V
	E_B_T_050	50% de carga	T=12,20 V
	E_B_T_070	70% de carga	T=12,36 V
	E_B_T_075	75% de carga	T=12,40 V
	E_B_T_100	100% de carga	T=12,60 V
	E_B_T_CAR	Bateria carregando	T=13,80 V
Estado Bateria Corrente	E_B_C_NOR	Corrente normal	I=0 mA
	E_B_C_FDC	Fuga de corrente	I=97,8 mA
	E_B_C_CCL	Consumidor Ligado	I=1,7 A
Estado Linha15	E_L_15_OFF	Linha 15 Desligada	0
	E_L_15_ON	Linha 15 Ligada	1

Fonte: (CUNHA & SENDA, 2015)

Em (LUNA FILHO, 2017), o autor desenvolveu o estudo da análise do estado e autonomia de uma bateria chumbo-ácido baseado na modelagem matemática das baterias. Segundo o autor, existem diversos tipos de modelos matemáticos que descrevem o comportamento elétrico e químico das baterias de chumbo-acido. Existem modelos químicos, que analisam as baterias apenas pelos aspectos físico-químicos e existem modelos que analisam as baterias sobre a modelagem elétrica. Há também os modelos híbridos que tentam combinar as duas técnicas anteriores.

O modelo que o autor utilizou para desenvolver o trabalho é conhecido como KiBaM (Kinetic Baterry Model), que foi desenvolvido especificamente para modelar os processos químicos no interior da bateria através do processo cinético. O modelo considera a existência de duas fontes de cargas denominadas: carga disponível e carga presa ou limitada. O modelo é baseado na resolução de equações diferenciais que modelam o comportamento da carga sobre diferentes regimes de carga e consumo, bem como parâmetros associados aos elementos químicos presentes na bateria. O trabalho apresentado indicou que os modelos empíricos podem trazer boas respostas sobre o estado baterias quando comparados aos modelos teóricos.

Outro trabalho que discute a análise do comportamento das baterias de chumbo ácido em regime de descarga foi apresentado por (PERGHER, 2018). Nesse trabalho o autor estudou o comportamento das baterias de chumbo ácido utilizados em sistemas de emergência de elevadores prediais. O trabalhou fez um conjunto de testes para avaliar o estado de saúde da bateria (SoH – *State of Helth*) em diferentes condições de consumo de carga e uma comparação por um modelo estimado do estado da carga. Os resultados apresentados pela medição empírica, quando comparado com os métodos analíticos, não trouxeram uma boa correlação, principalmente pelo fato de os testes terem sido realizados em baterias usadas, sobre as quais não há informações precisas que possam alimentar os modelos analíticos adequadamente. Contudo, esse trabalho trouxe também indicativos de que os métodos de avaliação empíricos, por medição, ainda são úteis para realizar as estimativas do tempo de carga.

3 METODOLOGIA DO PROJETO

No capítulo de Projeto e Desenvolvimento, serão apresentadas as etapas detalhadas de construção do *hardware* e *software* do projeto, bem como os principais componentes eletrônicos utilizados durante o desenvolvimento. O projeto foi dividido em três partes distintas: o *hardware* principal, o *hardware* secundário e o *software* de visualização de dados, desenvolvido em LabVIEW. Essa divisão permitiu uma melhor organização do projeto, facilitando o desenvolvimento e a integração dos diferentes componentes.

No *hardware* principal, foram utilizados componentes como o microcontrolador, fonte de alimentação e o circuito de carga, responsável por realizar a carga da bateria. Já no *hardware* secundário, foram utilizados sensores de temperatura e corrente, assim como um *display* para a visualização dos dados.

O *software* de visualização de dados, desenvolvido em LabVIEW, permitiu a criação de uma interface amigável e intuitiva para a visualização dos dados coletados pelos sensores, permitindo uma análise mais precisa e rápida dos resultados obtidos.

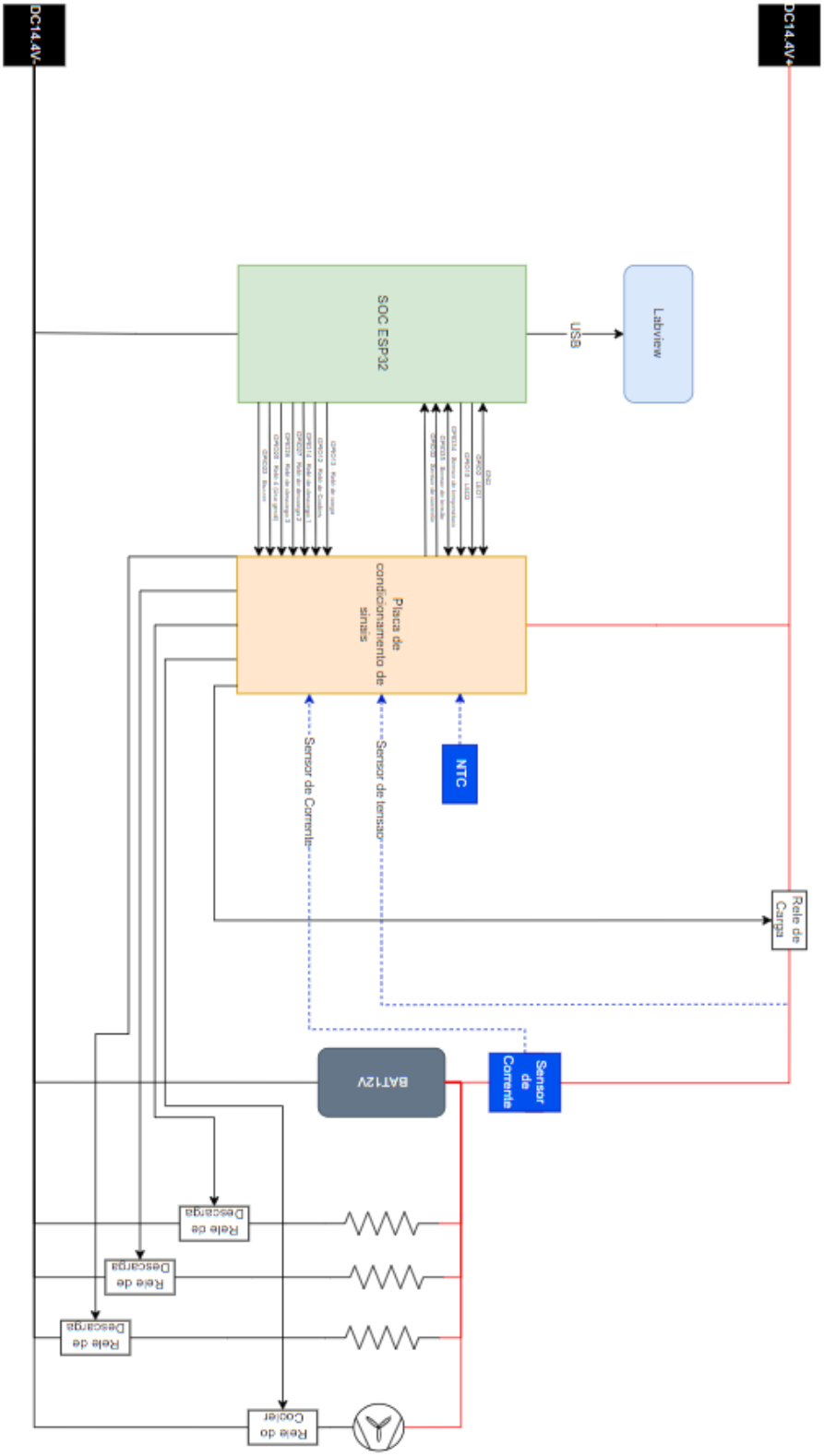
3.1 Hardware

O *hardware* principal é uma parte fundamental do projeto, responsável pelo controle geral das funções cruciais. Seu objetivo é monitorar e armazenar as informações de corrente e tensão durante o processo de carregamento e descarga de teste, garantindo que os dados sejam precisos e confiáveis. Além disso, o *hardware* principal monitora o sensor de temperatura para controlar emergências e prevenir danos à bateria. Outra função importante do *hardware* principal é controlar a corrente e a tensão do *hardware* secundário na fase de carga, garantindo que a bateria seja carregada de maneira segura e eficiente.

Por fim, o *hardware* principal também serve como interface entre o *hardware* e o *software* de visualização de dados LabVIEW, permitindo que o usuário visualize e analise os dados de forma clara e organizada. A Figura 10 ilustra o diagrama de blocos simplificado da proposta deste projeto. Um microcontrolador ESP-32 foi escolhido para controlar a “placa de condicionamento de sinais”, que contém toda a eletrônica para condicionar e ler os diferentes sensores utilizados para medir as principais grandezas da bateria, sob o ponto de vista elétrico. Observa-se também que o ESP-32 será integrado ao LabVIEW para enviar os dados dos sensores via um canal de

comunicação RS-232C emulado via USB. Os sensores utilizados são de tensão, corrente e temperatura.

Figura 10 - Diagrama de blocos do projeto.



Fonte: Autoria própria, 2023.

SoC ESP32

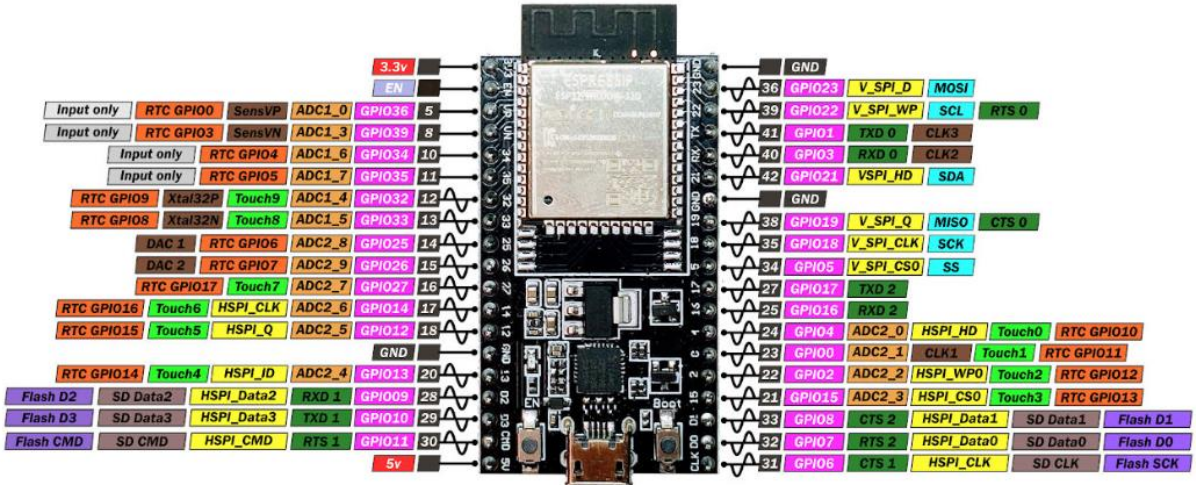
O ESP32 é um microcontrolador com um núcleo dual-core Tensilica LX6 de 32 bits, desenvolvido pela empresa chinesa Espressif Systems. É um dos microcontroladores mais populares atualmente. É utilizado em uma variedade de projetos que exigem conectividade WiFi e Bluetooth de baixo consumo de energia.

Algumas das especificações da ESP32 são descritas abaixo:

- Núcleo dual-core Tensilica LX6 de 32 bits, com frequência de até 240 MHz
- Coprocessador *ultra-low-power* (ULP) para processamento de sinais em tempo real
- Conectividade WiFi 802.11 b/g/n
- Conectividade Bluetooth 4.2 e BLE (Bluetooth Low Energy)
- 520 KB de memória SRAM
- 4 MB de memória flash
- Periféricos de entrada e saída, incluindo GPIO, PWM, I2C, SPI, UART, ADC e DAC
- Segurança avançada com suporte para criptografia AES, SHA-2, RSA e ECC
- Suporte para sistemas operacionais em tempo real (RTOS) como o FreeRTOS
- Baixo consumo de energia com modos de suspensão profunda e de espera

O ESP32 também é compatível com uma variedade de ambientes de desenvolvimento, incluindo a plataforma Arduino e a ferramenta de desenvolvimento da própria Espressif, o ESP-IDF. Isso permite que desenvolvedores de diferentes níveis de habilidade possam utilizar o ESP32 para projetos de diferentes complexidades, a figura 11 ilustra o kit de desenvolvimento utilizado na construção desse projeto.

Figura 11 - Kit de Desenvolvimento ESP32 e a respectiva pinagem.



Fonte: Espressif, 2020.

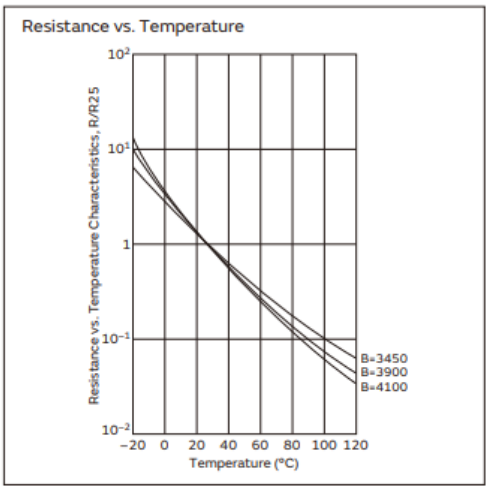
3.1.1 Sensor de Temperatura

O sensor de temperatura é um componente de segurança no processo de carga de baterias, pois ele permite que o microcontrolador monitore e controle a temperatura da bateria. Isso é importante porque o aumento excessivo de temperatura durante o carregamento pode levar a danos irreparáveis na bateria, como a perda de capacidade e vida útil reduzida.

Para garantir que a temperatura da bateria permaneça dentro dos limites seguros, o sensor de temperatura utilizado no projeto é do tipo NTC. Esse tipo de sensor é amplamente utilizado na indústria devido à sua alta sensibilidade e baixo custo, tornando-o uma escolha popular para aplicações de monitoramento de temperatura. Além disso, os sensores NTC têm uma característica única de terem uma resistência inversamente proporcional à temperatura, o que os torna ideais para medir a temperatura da bateria.

É importante ressaltar que o uso do sensor de temperatura permite que o processo de carga seja mais preciso e controlado, evitando que a bateria seja submetida a temperaturas excessivas que possam comprometer sua integridade. Portanto, a escolha do sensor de temperatura é um fator crítico para garantir a qualidade e a vida útil da bateria. A figura 12 mostra a curva característica da relação entre resistência e temperatura de um sensor NTC típico.

Figura 12 - Características do Sensor NTC.



Fonte: Datasheet MuRata, 2022.

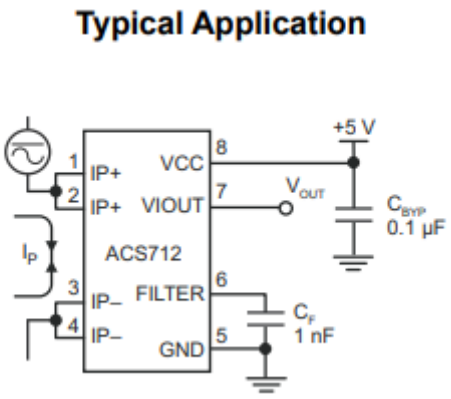
3.1.2 Sensor de Corrente

O sensor de corrente ACS712 30A é um componente fundamental no processo de carga de baterias, pois ele permite que o microcontrolador monitore e controle a corrente que está sendo fornecida para a bateria. Isso é importante porque uma corrente excessiva pode levar a danos irreparáveis na bateria, como superaquecimento e possibilidade de explosão.

O sensor de corrente ACS712 30A é um sensor Hall que utiliza o efeito Hall para medir a corrente. Ele possui uma faixa de medição de até 30A com uma sensibilidade de 66mV/A, o que permite medir com precisão a corrente fornecida à bateria. Além disso, o ACS712 30A possui uma ampla faixa de temperatura de operação, o que o torna adequado para aplicações em ambientes de alta temperatura.

A utilização do sensor de corrente ACS712 30A no projeto permite que o processo de carga seja mais preciso e controlado, evitando que a bateria seja submetida a correntes excessivas que possam comprometer sua integridade. Além disso, o sensor de corrente é fundamental para garantir a segurança do usuário, pois evita que a bateria seja sobrecarregada ou submetida a correntes excessivas que possam levar a acidentes. Portanto, a escolha do sensor de corrente ACS712 30A é um fator crítico para garantir a qualidade e a vida útil da bateria, bem como a segurança do usuário durante o processo de carga. A Figura 13 ilustra a pinagem do sensor de corrente ACS712 e o respectivo circuito mínimo para o funcionamento dele.

Figura 13 – Pinagem do Sensor de Corrente ACS712.



Fonte: Datasheet ACS712.

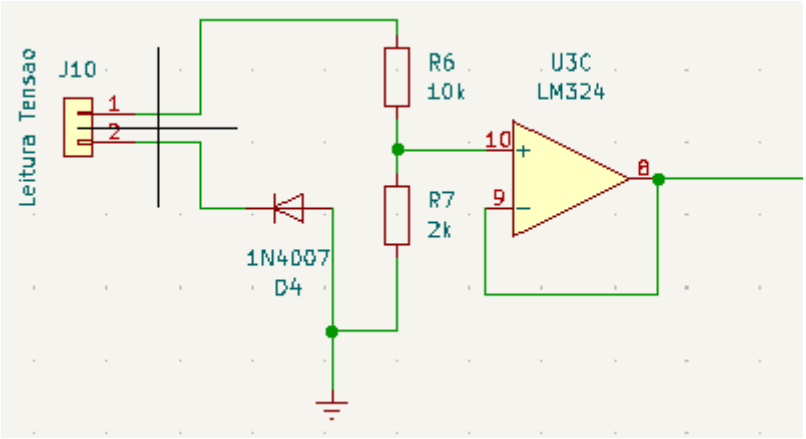
3.1.3 Leitura de Tensão

A estratégia de leitura de tensão, ilustrada na figura 14, é baseada em um divisor resistivo composto por dois resistores, R1 e R2, onde a tensão de entrada é aplicada em R1 e a tensão de saída é medida em R2. O valor dos resistores é escolhido de forma a obter uma queda de tensão significativa, mas sem prejudicar a precisão da leitura. No caso específico do projeto, os valores escolhidos para R1 e R2 são de 10k e 2k, respectivamente. Isso resulta em uma queda de tensão de aproximadamente 1/6 do valor de entrada. A saída desse divisor é conectada a uma das portas do amplificador operacional LM324, que funciona como um seguidor de tensão.

O seguidor de tensão tem como função garantir que a impedância de entrada do circuito de medição seja alta, de modo que a leitura da tensão não seja afetada pela carga que está sendo medida. A tensão medida pelo LM324 é, em seguida, convertida em um valor digital pelo microcontrolador, utilizando o conversor analógico-digital (ADC) integrado. O valor digital é então processado pelo *software* embarcado, que realiza os cálculos necessários para determinar a tensão real da bateria.

Essa estratégia de leitura de tensão é simples e eficaz, permitindo que a tensão da bateria seja medida de forma precisa e confiável. Além disso, o uso de um seguidor de tensão garante que a leitura da tensão não seja afetada pela carga da bateria, o que pode melhorar ainda mais a precisão da medição.

Figura 14 - Esquema elétrico da leitura de tensão.



Fonte: Autoria própria, 2023.

3.1.4 Relé Automotivo

O relé é composto por duas partes principais: uma bobina e um conjunto de contatos. Quando uma corrente elétrica é aplicada à bobina, ela cria um campo magnético que atrai o conjunto de contatos, fechando um circuito elétrico e permitindo que a corrente flua através do dispositivo de carga. As vantagens principais do uso de relés é o fato deles terem, o que se conhece como “contato seco”. Os contatos dos relés são livres de potenciais. Logo, não há nenhuma interferência do circuito de medição, além de suportar altas corrente, isolando o circuito elétrico da medição em relação à eletrônica de controle. Nesta aplicação optou-se pelo uso de relés automotivos, dado ao baixo custo, fácil disponibilidade no mercado local e capacidade de comutação de cargas com alto consumo de corrente. O relé utilizado no projeto é ilustrado na figura 15.

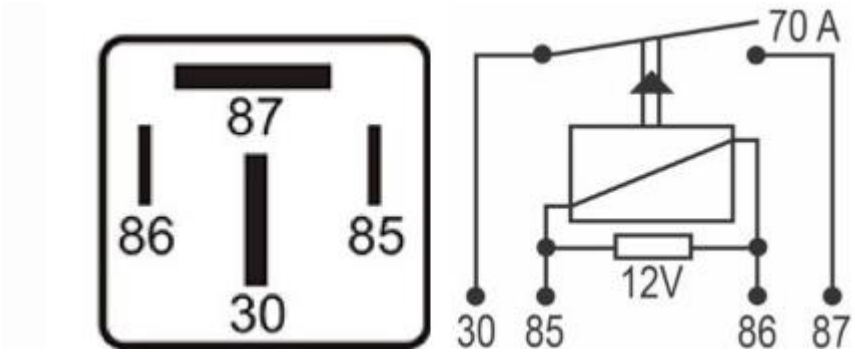
Figura 15 - Relé automotivo de 70A.

Fonte: DNI, 2023.

O relé automotivo escolhido para o sistema de controle de carga e descarga da bateria é o modelo de 12V com capacidade para suportar correntes de até 70A. Este relé foi selecionado devido à sua capacidade de suportar correntes elevadas por um período suficiente para realização dos testes, que são de 15 segundos.

Além disso, este relé possui uma alta confiabilidade e durabilidade, o que o torna adequado para uso em sistemas de controle de carga e descarga de baterias que exigem alto desempenho e segurança. Ele é capaz de atuar em condições extremas de temperatura, vibração e umidade, com uma longa vida útil. Na Figura 16 observa o diagrama do circuito interno do relé e a pinagem.

Figura 16 - Diagrama do Relé 70A.



Fonte: DNI, 2023.

3.1.5 Resistores de descarga elétrica

Certos tipos de resistores são ideais para a descarga de corrente elétrica em diversas aplicações, incluindo testes em baterias de chumbo ácido. Nesse caso, um tipo de resistor escolhido é o Nikrothal® 80, que é uma liga austenítica de níquel-cromo capaz de suportar temperaturas de até 1.200°C (2.190°F). Além disso, possui alta resistividade, boa resistência à oxidação e estabilidade estrutural, ilustrada na figura 17.

Figura 17 - Resistores de Descarga e Relé 70A

Fonte: Autoria própria, 2023.

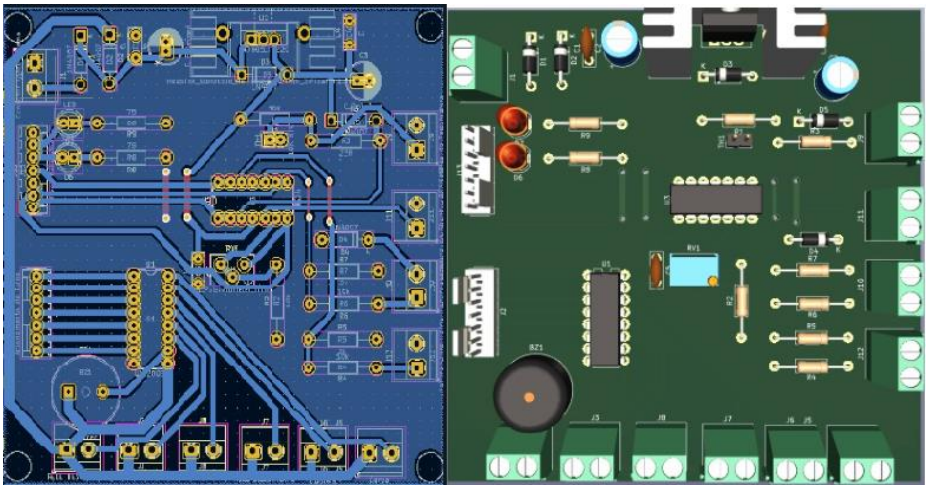
Um benefício adicional do Nikrothal® 80 é sua boa ductilidade após o uso, o que significa que pode suportar a descarga de corrente elétrica por longos períodos sem sofrer danos significativos. A soldabilidade excelente também permite que ele seja facilmente incorporado em diversos projetos elétricos.

3.1.6 Hardware de controle

O *hardware* de controle utilizado no sistema embarcado de carga e descarga de baterias de chumbo ácido é composto por um bloco de condicionamento de sinal analógico e um bloco de atuadores, além de contar com uma fonte de alimentação de 5V, leds para comunicação e um *buzzer*. Esse *hardware* é essencial para o funcionamento correto do sistema, permitindo a leitura de sinais analógicos dos sensores de temperatura e tensão da bateria, bem como a atuação nos reles para controlar a carga e descarga da bateria. Com a ajuda dos leds e *buzzer*, é possível visualizar e ouvir as informações de comunicação entre o sistema e o *hardware* de controle. Sem esse componente, o sistema não seria capaz de realizar a carga e descarga da bateria de forma eficiente e segura.

O protótipo da PCB foi feito no *software* Open-Source Kicad, um *software* de design eletrônico de código aberto que tem se tornado cada vez mais popular entre os projetistas e engenheiros da área. O resultado do circuito e a simulação pode ser observado na Figura 18.

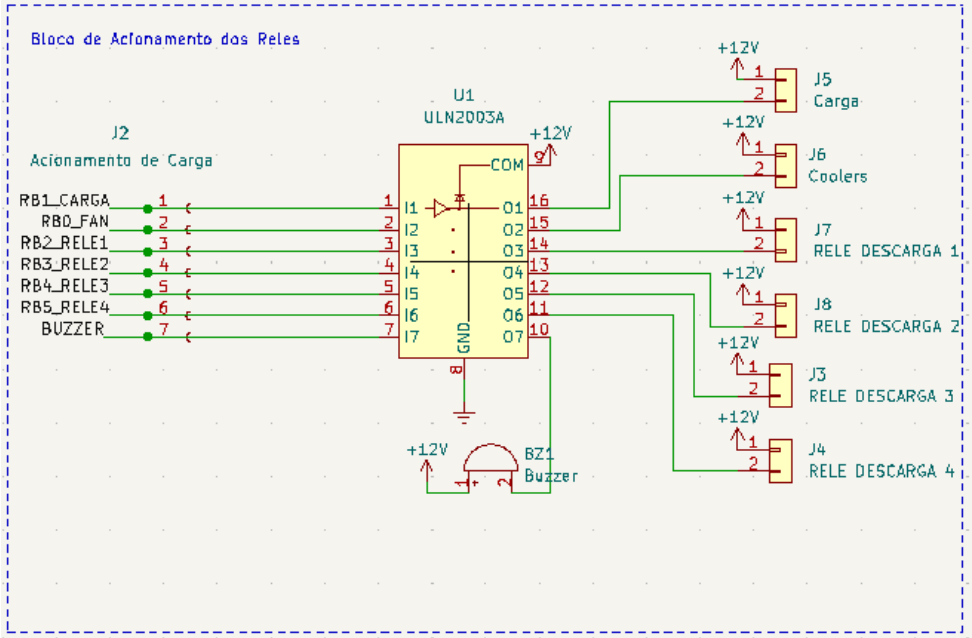
Figura 18 - Layout da placa de circuito impresso finalizado



Fonte: Autoria própria, 2023.

O *hardware* de controle possui como objetivo principal receber e processar os valores de tensão provenientes dos sensores, utilizando o microcontrolador para tomar as decisões necessárias e acionar os atuadores de acordo com a programação definida. Na configuração dos atuadores, é utilizado um circuito integrado ULN2003 que, quando acionado pelo microcontrolador, fornece a corrente necessária para acionar os relés. Essa configuração é ilustrada na Figura 19. O ULN2003 é um circuito integrado de alta tensão e alta corrente, projetado para ser utilizado como driver para cargas indutivas, como relés, motores, entre outros. Sua utilização garante maior eficiência e segurança ao sistema de controle.

Figura 19 - Esquema elétrico do bloco de acionamento dos relés



Fonte: Autoria própria, 2023.

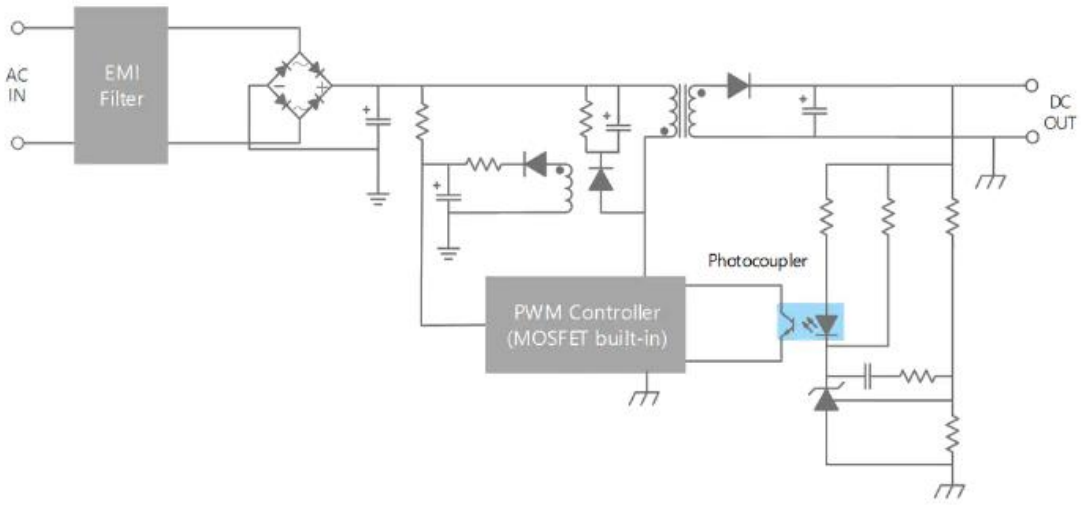
3.1.7 Hardware de carga

O *hardware* secundário do projeto consiste em uma fonte de alimentação do tipo chaveada, mais precisamente, um modelo ATX amplamente utilizado em computadores e projetos similares. As fontes de alimentação podem ser divididas em dois tipos principais: fontes lineares e fontes chaveadas. As fontes chaveadas regulam a tensão de saída por meio de circuitos integrados, atendendo às necessidades específicas do sistema. Por outro lado, as fontes lineares utilizam outros componentes, como transformadores e filtros, para manter a tensão de saída.

Embora as fontes chaveadas apresentem melhor rendimento, sejam mais compactas e capazes de dissipar uma potência maior, elas são mais complexas e geralmente têm um custo de reparo e um valor agregado superior às fontes lineares.

No caso do projeto em questão, a escolha pela fonte chaveada do tipo ATX se deve à sua ampla disponibilidade no mercado e comprovada eficiência para alimentar sistemas eletrônicos. A figura 20 ilustra um exemplo de uma fonte chaveada do tipo *flayback.*

Figura 20 - Esquema Elétrico FlyBack AC-DC.



Fonte: Toshiba, 2022.

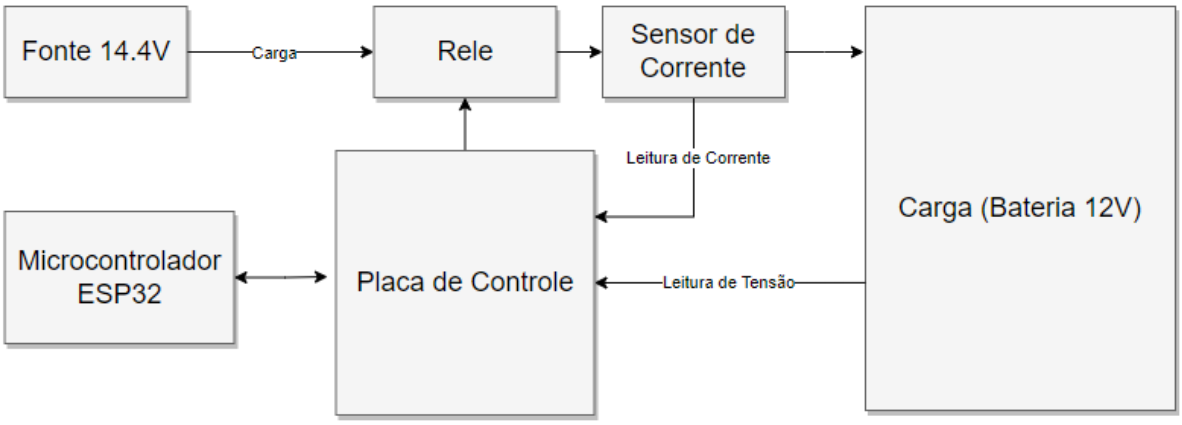
3.1.8 Circuito de Carga

O circuito de carga é um conjunto de componentes responsáveis por realizar os testes da bateria chumbo-ácido. Para atingir esse objetivo, utilizaremos aspectos baseados na análise de CCA, que é o mais indicado no ramo automobilístico. No entanto, é importante ressaltar que as análises de CCA possuem condições de teste rigorosas e controles específicos em laboratório, o que pode resultar em diferenças significativas em relação aos resultados obtidos em condições fora dos padrões exigidos.

Para minimizar essa discrepância, realizaremos uma compensação de valores baseada em literaturas de pesquisa e valores reais medidos em campo. Dessa forma, podemos garantir uma análise mais precisa e confiável da condição da bateria.

O circuito de carga deste projeto contará com uma carga resistiva fixa e um circuito de leitura de tensão, que indicará a condição da bateria, como ilustra a figura 21. Esses dados serão tratados e apresentados de forma simples e direta

Figura 21 - Diagrama de Blocos do Circuito de Carga.



Fonte: Autoria própria, 2023.

3.2 Firmware

O *firmware* é um dos componentes mais importantes do projeto, uma vez que é responsável por gerenciar e controlar todo o sistema de recarga e descarga da bateria. Ele é um *software* embarcado, ou seja, é executado diretamente em *hardware* específico, neste caso o ESP32.

O *firmware* é composto por diferentes algoritmos e protocolos que permitem o controle preciso da carga e descarga de baterias. Isso inclui o algoritmo de controle de carga, que é responsável por gerenciar a quantidade de energia fornecida à bateria durante o processo de carga, garantindo que ela seja carregada com segurança e eficiência. Além disso, o *firmware* também inclui o protocolo de comunicação entre o *firmware* e o *software* LabVIEW, que permite que o usuário monitore e controle o sistema de carregamento e descarregamento de baterias através de uma *interface* gráfica intuitiva.

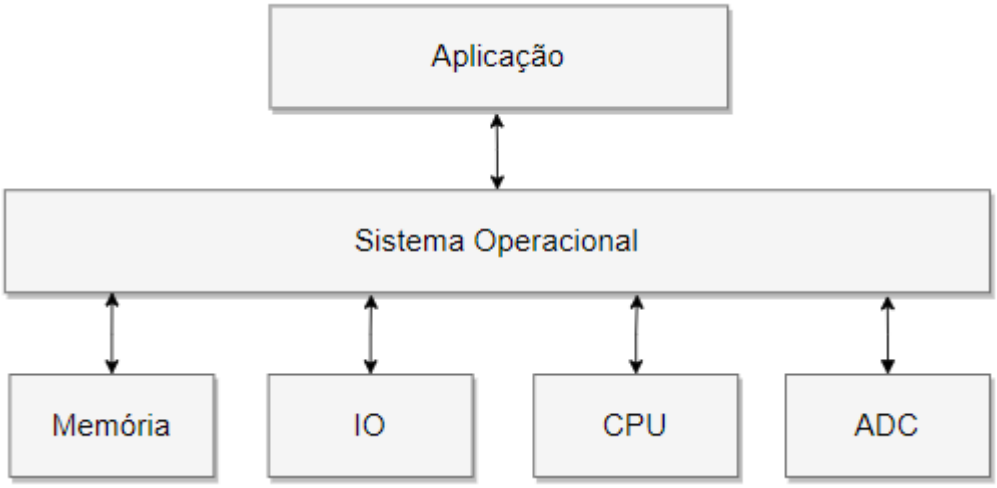
Uma das tecnologias utilizadas no *firmware* é o FreeRTOS, um sistema operacional em tempo real de código aberto, que permite o gerenciamento de múltiplas tarefas simultaneamente. Isso é particularmente importante no caso do projeto, uma vez que o *firmware* precisa ser capaz de executar várias tarefas em paralelo, como a leitura de sensores, o controle de carga e a comunicação com o *software* LabVIEW.

3.2.1 Descrição geral do firmware

O ESP32 é um microcontrolador que suporta um sistema operacional de tempo real (RTOS), que atua como uma camada de abstração entre a aplicação e os recursos do sistema. O RTOS gerencia a alocação de memória, o processamento, a comunicação com as portas de entrada e saída do microcontrolador e outros recursos importantes, permitindo que a aplicação seja desenvolvida de forma mais eficiente e escalável.

A figura 22 ilustra como o RTOS funciona como um intermediário entre a aplicação e os recursos do sistema. A aplicação se comunica com o RTOS, que por sua vez gerencia as tarefas e os recursos, garantindo a execução de forma precisa e confiável. Além disso, o RTOS oferece recursos avançados, como filas, semáforos e *mutex*, que podem ser usados para garantir a sincronização e comunicação entre as tarefas, bem como prevenir problemas como condições de corrida e *deadlocks*.

Figura 22 - Arquitetura de Software do RTOS.



Fonte: Autoria própria, 2023.

As tarefas, são a unidade fundamental de execução em sistemas operacionais de tempo real, como o FreeRTOS que é suportado pelo ESP32. Cada tarefas é uma função que pode ser executada concorrentemente com outras tarefas e tem seu próprio contexto de execução, incluindo seu próprio *stack* e registradores.

As tarefas são projetadas para serem executadas de forma cooperativa, ou seja, cada tarefa deve explicitamente liberar o processador para que outras tarefas possam ser executadas.

Com o objetivo de organizar o processamento das diferentes tarefas de forma eficiente, foram implementadas as tarefas conforme a tabela 3. Cada uma das tarefas é executada de forma cooperativo pelo processador, o que significa que o processador é responsável por interromper a execução de uma tarefa quando outra tarefa com prioridade mais alta precisa ser executada. Isso permite que o *firmware* seja desenvolvido de forma mais fácil e modular, já que as tarefas são independentes e podem ser atualizadas individualmente sem afetar o restante do sistema. Além disso, a utilização de tarefas aumenta a estabilidade e confiabilidade do sistema, pois reduz as chances de bloqueios ou travamentos.

Tabela 3 - Lista de tarefas dentro da estrutura do RTOS

Tarefa	Nome	Descrição	Taxa de Execução (ms)
Gerenciamento de sensores	xGerenciaSensores	Responsável pela leitura dos sensores conectados ao ESP32	100 ms
Gerenciamento de UART	xGerenciaUart	Gerencia a comunicação via UART com outros dispositivos externos	50 ms
Diretoria	xDiretoria	Responsável pelo controle de fluxo de outras tarefas, definindo prioridades e interrupções	portMAX_DELAY
Gerenciamento de atuadores	xGerenciaAtuadores	Gerencia os atuadores conectados ao ESP32	portMAX_DELAY (se possível encontrar um nome para esse tipo de delay)
Gerenciamento de carregador	xGerenciaCarregador	Gerencia o carregamento da bateria do dispositivo	100000 ms
Gerenciamento de testador	xGerenciaTestador	Gerencia os testes realizados na placa	portMAX_DELAY
HearthBit	xHearthBit	Responsável por enviar um sinal de confirmação de que o sistema está em funcionamento	1000 ms
Alarme	xAlarme	Responsável pelo acionamento de um alarme em caso de falhas no sistema	50 ms

Fonte: Autoria própria, 2023.

A tarefa principal neste sistema é a "Diretoria", responsável por tomar decisões com base na leitura dos *event groups* e notificações de outras tarefas, como a "UART". A tarefa "UART" envia comandos para a "Diretoria", que verifica se algum *bit* do *event group* está configurado como '1' (verdadeiro) e, em seguida, executa a ação correspondente de acordo com essa verificação. A "Diretoria" desempenha um papel central no processamento e na coordenação das tarefas, garantindo que as ações sejam tomadas com base nas condições corretas provenientes dos *event groups* e notificações das outras tarefas.

Os *event groups* são uma funcionalidade do sistema operacional de tempo real do ESP32 que permitem que as tarefas possam se comunicar entre si de maneira efetiva. Eles consistem em um conjunto de *bits* que podem ser definidos ou limpos por várias tarefas. Cada *bit* representa um evento específico.

No sistema em questão, a tarefa principal é a diretoria, que é responsável por controlar toda a operação do sistema operacional. Ela utiliza os *event groups* para

monitorar vários eventos relevantes para o sistema. Por exemplo, o *bit* 0 representa o modo de carga da bateria, enquanto o *bit* 1 e o *bit* 2 representam diferentes testes de carga. Quando uma tarefa, como a UART, envia um comando para a diretoria, ela verifica se algum bit do *event group* está configurado como '1' e toma a ação apropriada.

Para implementar o *event group* no sistema, foi criado um *handle* chamado "Evt_Diretoria", que representa o conjunto de bits do *event group*. Cada *bit* do *event group* é identificado por um número, que pode ser usado para definir, limpar ou verificar o estado de um determinado *bit.* Por exemplo, a diretoria pode definir o *bit* 0 usando a função "xEventGroupSetBits", indicando que a bateria está em modo de carga. Em seguida, outras tarefas podem verificar o estado do *bit* 0 usando a função "xEventGroupGetBits".

A tabela abaixo apresenta os bits do event group e sua descrição:

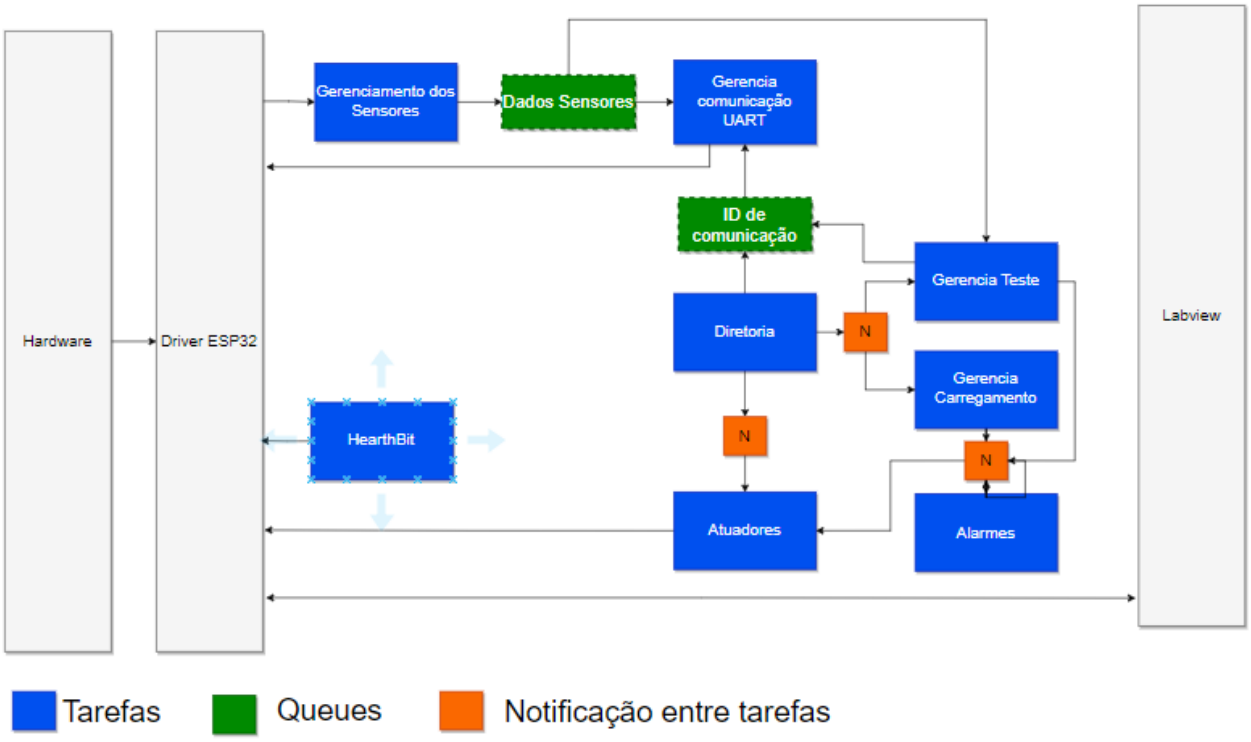
Tabela 4 - Todos os *bits* de eventos utilizados no *firmware*.

Bit	Descrição
0	Bateria está em modo de carga
1	Teste de Carga 70A
2	Teste de Carga 140A
3	Temperatura acima de 50°C
4	Tempo de espera após o teste
5	Corrente de carga próxima a 0A
6	Tensão entre 12.2V e 13.2V
7	Tensão abaixo de 10V
8	Bateria em modo de teste
9	Tensão acima ou igual a 14.3V

Fonte: Autoria própria, 2023.

A figura 23 apresenta um diagrama que ilustra a comunicação entre as tarefas para o correto encaminhamento do frame de dados até o *LabView.*

Figura 23 - Diagrama de Blocos do funcionamento do *Firmware.*



Fonte: Autoria própria, 2023.

3.2.2 Algoritmo de controle de carga

A função de gerenciamento do carregador é responsável por controlar o processo de carregamento da bateria. Essa função é executada como uma tarefa e seu objetivo principal é monitorar constantemente a tensão, corrente e temperatura durante o carregamento. Com base nas condições verificadas, a função toma ações apropriadas para garantir um carregamento seguro e eficiente da bateria.

A apêndice A, mostra como a tarefa é responsável por controlar a carga da bateria. Durante a execução da tarefa, a função verifica se a tensão se encontra 14.4V e a corrente está em de 100 mA, o que indica que a carga foi concluída com sucesso. Nesse caso, a função notifica as tarefas de gerenciamento dos atuadores e alarme para desligar o sistema e informar que o teste foi concluído. Além disso, o *bit* correspondente ao modo de carga é limpo no *event group* "Evt_Diretoria". Essa ação garante que as tarefas relevantes sejam informadas sobre o término do teste.

Outra verificação importante feita pela função é a temperatura. Se a temperatura da bateria estiver acima do limite estabelecido, a função também notifica

as tarefas para desligar o sistema e informa sobre a temperatura elevada. Essa medida de segurança é essencial para evitar danos à bateria e garantir um ambiente de operação seguro. Durante o processo de carregamento, a função exibe mensagens de *status* para indicar que o modo de carga está ativo. Isso pode ser útil para fins de depuração e acompanhamento do processo.

3.2.3 Algoritmo de Teste

O algoritmo de teste é responsável por gerenciar a execução dos testes da bateria. levando em consideração características específicas, como a corrente por hora da bateria a ser testada. Dependendo do tipo de bateria, o algoritmo aciona diferentes elementos de descarga para garantir a precisão do teste.

No caso de baterias voltadas para motos, o algoritmo aciona um único relé, que por sua vez ativa um resistor de descarga capaz de consumir 70A por 15 segundos. Esse procedimento permite uma descarga controlada da bateria.

Por outro lado, quando a bateria a ser testada é de um carro, o algoritmo aciona dois resistores através de dois relés, resultando em uma descarga que consome 140A.

É importante mencionar que o resistor utilizado possui uma resistência de 0,165 ohms, o que garante o controle adequado da descarga.

Ele recebe comandos para realizar diferentes tipos de testes e segue um processo de dois passos: descarregar a bateria e deixá-la descansar por um tempo, e em seguida verificar a tensão.

A função xGerenciaTestador é responsável por executar esse algoritmo. Ela recebe um comando de teste e inicia o procedimento correspondente. Durante a execução do teste, ela faz uso de dados lidos dos sensores para monitorar o estado da bateria. No início do teste, a função verifica o comando recebido e realiza as devidas verificações para garantir que a tensão da bateria esteja dentro da faixa aceitável. Caso a tensão esteja adequada, o teste é iniciado e o primeiro passo é executado: a descarga da bateria. Durante esse passo, são coletados dados da tensão da bateria e comparados com um valor de referência. Após o tempo determinado para a descarga, o próximo passo é iniciado.

No segundo passo, a bateria é deixada em repouso por um período definido. Novamente, dados da tensão são coletados e comparados com um valor de

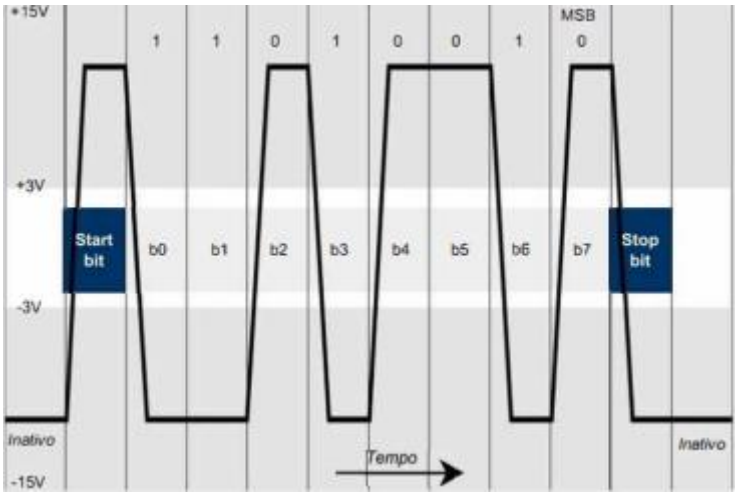
referência. Após esse período, a função verifica se a tensão está acima de um limite determinado. Se a tensão estiver acima desse limite, a bateria é considerada em boas condições e uma mensagem é registrada. Caso contrário, a bateria é considerada em condições ruins ou defeituosa e uma mensagem correspondente é registrada.

Ao finalizar o teste, a função limpa os bits do grupo de eventos correspondentes ao teste em execução e, se necessário, aciona um alarme para indicar a conclusão do teste.

3.2.4 Protocolo de comunicação entre firmware e *software* LabVIEW

O protocolo UART é utilizado para enviar e receber dados entre dispositivos eletrônicos de forma serial assíncrona, sem a necessidade de um *clock* externo para sincronização. A comunicação UART é realizada por meio de bits sequenciais, com a adição de bits de controle para garantir a integridade dos dados. A figura 24 ilustra o frame de dados de uma comunicação serial.

Figura 24 - Sinal serializado para transmissão em RS232.

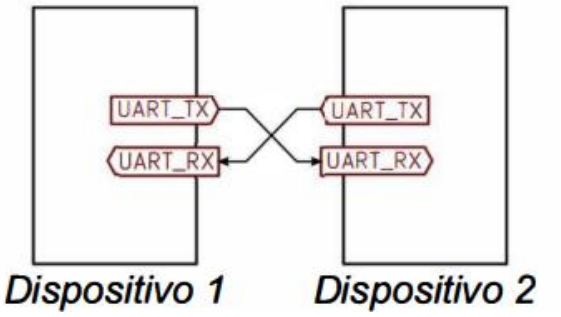


Fonte: ALMEIDA, Rodrigo Maximiano A.; MORAES, Carlos Henrique V.; SERAPHIM, Thatyana F. Piola. Programação de Sistemas Embarcados. 1. ed. São Paulo: Grupo Gen., 2021. p. 292.

No contexto da comunicação entre o ESP32 e o LabVIEW, o protocolo UART é utilizado para transmitir os dados de tensão, corrente e temperatura do ESP32 para o LabVIEW. O ESP32 configura a UART com parâmetros como taxa de *bits*, número de *bits* de dados, número de *bits* de parada e paridade. Em seguida, os dados são

transmitidos serialmente pelo fio de transmissão (TX) e recebidos pelo fio de recepção (RX) no LabVIEW.

Figura 25 - Comunicação UART.



Fonte: ALMEIDA, Rodrigo Maximiano A.; MORAES, Carlos Henrique V.; SERAPHIM, Thatyana F. Piola. Programação de Sistemas Embarcados. 1. ed. São Paulo: Grupo Gen, 2021. p. 291.

A escolha do protocolo UART se dá pela sua simplicidade de implementação e ampla disponibilidade de dispositivos que o suportam. Além disso, ele é adequado para a comunicação em distâncias curtas, o que é suficiente para a comunicação entre o ESP32 e o LabVIEW no ambiente de teste.

No código fornecido é realizada a configuração da UART no ESP32. Os parâmetros de comunicação, como a taxa de bits e o número de bits de dados, são definidos no *struct* uart_config_t. Em seguida, a função *uart_param_config* é chamada para configurar a UART com os parâmetros definidos. A função *uart_set_pin* é utilizada para configurar os pinos GPIO que serão utilizados para a comunicação UART. Por fim, a função *uart_driver_install* é chamada para instalar e inicializar o *drive* da UART. Essas configurações permitem que o ESP32 estabeleça a comunicação UART com o LabVIEW, permitindo a transmissão dos dados necessários para o teste.

Após o processamento dos dados recebidos, a tarefa monta uma *string* contendo os dados de corrente, tensão, temperatura e uma mensagem definida no LabVIEW. Em seguida, a *string* é enviada pela UART usando a função uart_write_bytes.

Essa abordagem permite que o microcontrolador receba os comandos do LabVIEW e envie os dados de volta para exibição no LabVIEW. Dessa forma, há uma comunicação bidirecional entre o microcontrolador e o LabVIEW por meio do protocolo

UART. O *frame* de dados enviado do ESP32 para o LabVIEW pode ser visualizado na tabela 5.

Tabela 5 - *Frame* de dados.

Posição	Dado
0	ID
1	Corrente
2	Tensão
3	Temperatura
4	Mensagem de status

Fonte: Autoria própria, 2023.

Cada posição no frame de dados contém um determinado tipo de informação:

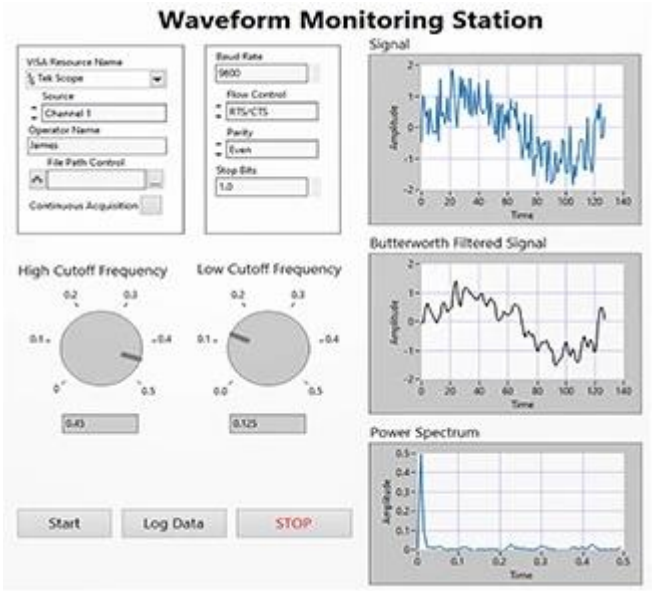
- ID: Identificador único da amostra.
- Corrente: Valor da corrente medida.
- Tensão: Valor da tensão medida.
- Temperatura: Valor da temperatura medida.
- Mensagem de status: Mensagem de status definida no LabVIEW.

3.3 SOFTWARE LABVIEW

O *software* LabVIEW, desenvolvido pela National Instruments, será a *interface* utilizada no projeto para aquisição e processamento de dados. O LabVIEW é uma plataforma de programação gráfica que permite criar aplicativos de teste, medição e controle, além de fornecer rápido acesso ao *hardware* e a informações obtidas a partir dos dados.

Uma das principais vantagens do LabVIEW é sua *interface* gráfica de usuário (IHM) intuitiva e fácil de usar, que permite criar *Dashboards* e visualizar dados em tempo real através de gráficos. Essa *interface* gráfica torna o processo de programação mais rápido e eficiente, além de facilitar a análise dos dados coletados, conforme ilustra a figura 26.

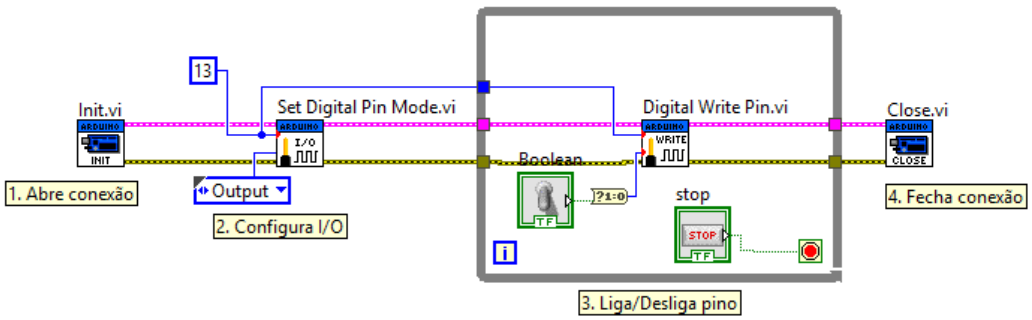
Figura 26 - Exemplo de um Dashboard gerado em LabVIEW.



Fonte: LabVIEW, 2022.

O LabVIEW também possui uma linguagem de programação gráfica chamada "G". Essa linguagem gráfica, ilustrada na figura 27, usa diagramas de blocos para representar a lógica do programa, o que simplifica o processo de programação para usuários sem conhecimento prévio em programação. O uso do LabVIEW no projeto permitirá que os dados coletados pelo circuito de carga sejam facilmente tratados e apresentados ao operador de uma maneira clara e direta. Através da IHM, será possível visualizar informações relevantes sobre a bateria em tempo real, o que possibilita uma tomada de decisão mais rápida e assertiva.

Figura 27 - Exemplo Programação LabVIEW.



Fonte: FilipeFlop, 2022.

3.4 Dashboard em LabVIEW

O *dashboard* desenvolvido em LabVIEW permite que você tenha uma visão completa do estado da bateria em tempo real. Com ele, você pode realizar duas ações principais: carregar a bateria e testá-la de acordo com a sua faixa nominal. Além disso, é possível acompanhar informações importantes que são enviadas pelo ESP32 por meio de uma caixa de mensagem de texto. Isso ajuda a identificar qualquer problema que possa estar acontecendo com a bateria, permitindo que se tome as medidas necessárias para solucioná-los. Outra funcionalidade presente no *dashboard* é o gráfico de carga e descarga da bateria, que exibe a variação da corrente ao longo do tempo. Com isso, é possível identificar possíveis problemas de desempenho ou falhas no sistema. O d*ashboard* pode ser visto visualizado na Figura 28.

Figura 28 - Plataforma de Diagnóstico feito em LabVIEW.

Fonte: Autoria própria, 2023.

3.5 Estratégia de aquisição de dados

A estratégia de aquisição de dados adotada no sistema consiste em estabelecer uma comunicação bidirecional entre o dispositivo embarcado e o *software* LabVIEW. Ao solicitar informações específicas, o usuário aciona a tarefa responsável pela comunicação, que recebe os parâmetros necessários para compor um *frame* de dados a ser enviado via UART.

O *frame* de dados é composto por diferentes informações relevantes, incluindo um identificador (ID) que representa o status atual do equipamento. Além disso, são incluídos os valores da corrente, tensão e temperatura coletados pelo sistema. Essas

informações são formatadas adequadamente utilizando a função *sprintf* e armazenadas em um *buffer*.

Após a formatação do *frame* de dados, ele é enviado para o LabVIEW por meio da porta UART. O *software* LabVIEW, por sua vez, recebe os dados enviados e realiza o processamento adequado. Utilizando a função *'Scan From String'*, as informações contidas no *frame* de dados são extraídas e interpretadas.

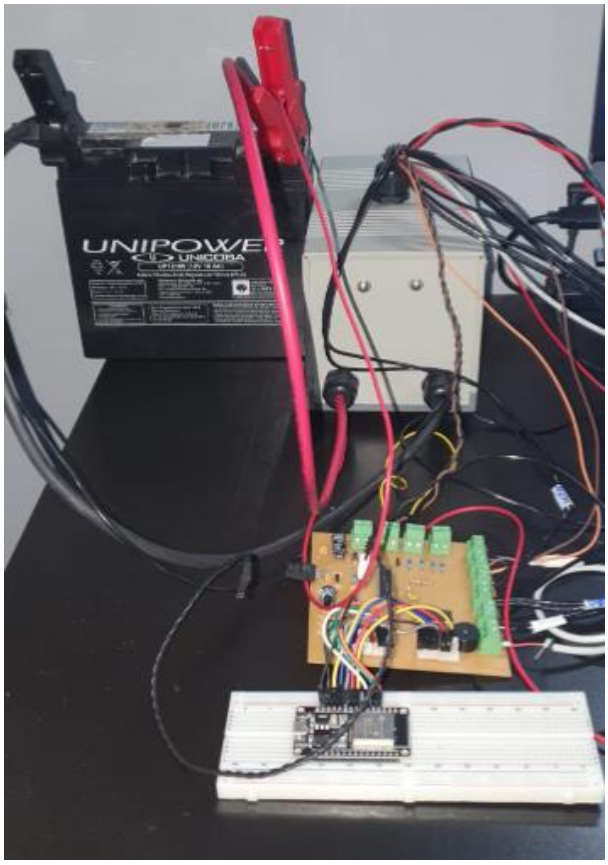
Uma vez processadas, as informações são utilizadas para fornecer um *feedback* visual para o usuário. O LabVIEW exibe a mensagem correspondente ao *status* atual do equipamento, informando sobre seu funcionamento e possíveis eventos. Além disso, os valores da corrente, tensão e temperatura são transportados para gráficos e outros elementos de visualização, permitindo ao usuário acompanhar em tempo real o comportamento do sistema. O código completo do *dashboard* desenvolvido em LabView pode ser visualizado no apêndice B.

4 TESTES E ANÁLISES DOS RESULTADOS

Nesta seção serão apresentados os resultados dos testes realizados no sistema de avaliação da saúde da bateria de chumbo ácido, abordando tanto a parte de descarga quanto a parte de carga do processo.

A montagem do sistema foi de acordo com o diagrama de blocos descrito no tópico 3.1, que aborda o *hardware*. A Figura 29 ilustra o protótipo do projeto, que inclui os seguintes componentes: uma bateria de 12V - 18Ah, um gabinete contendo resistores de descarga e um *cooler* para resfriamento, uma placa de condicionamento de sinais responsável pela leitura da tensão, temperatura e corrente, e, por fim, o microcontrolador ESP32.

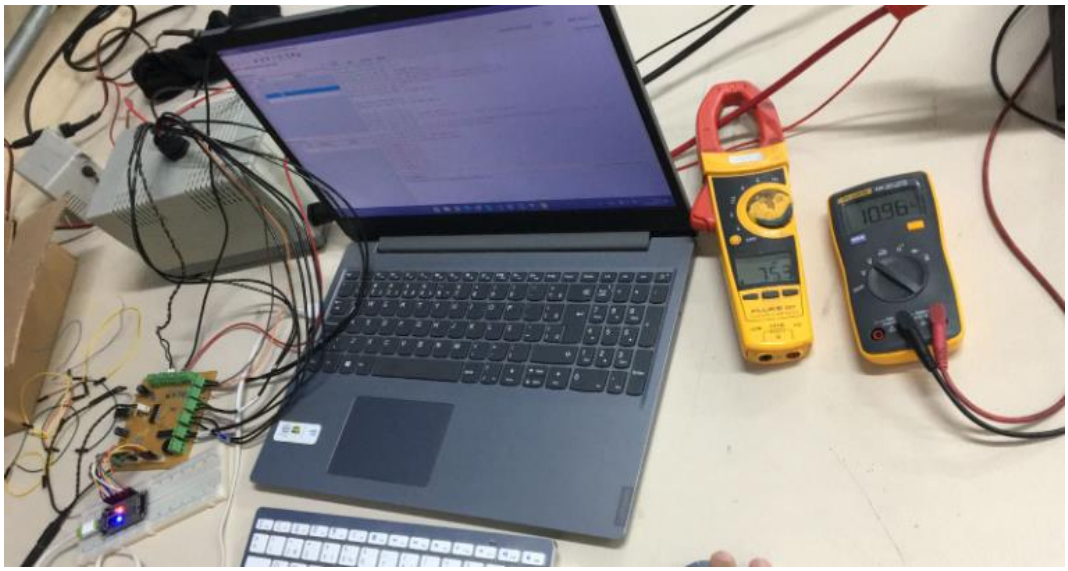
Figura 29 - Protótipo do projeto



Fonte: Autoria própria, 2023.

O usuário tem a opção de selecionar o teste com base na capacidade de corrente por hora que a bateria é capaz de fornecer. O algoritmo foi programado para acionar um único relé, o que resulta na ativação de apenas um resistor de descarga durante o teste. Esse resistor é dimensionado para gerar uma corrente de 70A durante 15 segundos, conforme ilustrado na Figura 30.

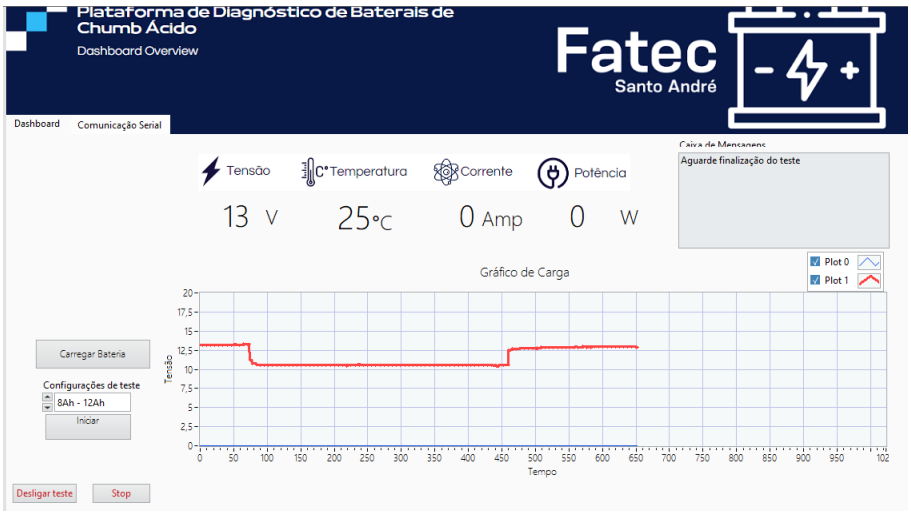
Figura 30 - Medições com multímetro e amperímetro do teste realizado.



Fonte: Autoria própria, 2023.

Durante a realização do teste é possível acompanhar a curva de descarga da bateria em tempo real no *software* LabVIEW. A Figura 31 ilustra essa curva, que representa o comportamento da bateria ao longo dos 15 segundos de teste.

Figura 31 - Plataforma durante o teste realizado.



Fonte: Autoria própria, 2023.

Após a conclusão da descarga, o microcontrolador aguarda um período adicional de 45 segundos antes de apresentar o relatório do teste. Esse intervalo é necessário para permitir um resfriamento adequado do gabinete, uma vez que os resistores utilizados na descarga podem atingir altas temperaturas. Durante esse tempo, o cooler é acionado para promover o resfriamento do sistema, garantindo a

segurança e integridade dos componentes. A Figura 32 ilustra o laudo do teste, que é exibido na caixa de mensagens do software LabVIEW.

Figura 32 - Laudo da bateria após o teste

Fonte: Autoria própria, 2023.

4.1 Descarga

Durante os testes de descarga, o sistema demonstrou um funcionamento satisfatório, cumprindo os objetivos propostos. Os sensores integrados foram capazes de coletar as informações relevantes sobre a bateria, que foram corretamente processadas pelo microcontrolador. A comunicação via UART permitiu o envio eficiente dos dados para o LabVIEW, onde foram apresentados em um formato compreensível.

Utilizando o *firmware* desenvolvido, o LabVIEW foi capaz de interpretar os dados recebidos e fornecer um bom diagnóstico do estado da bateria. Os resultados foram exibidos no *dashboard* do LabVIEW fornecendo informações sobre o estado da bateria, tensão e temperatura. Esses dados permitiram uma avaliação do desempenho da bateria durante o processo de descarga. Porém, ainda é possível verificar problemas de sincronização de mensagens entre o microcontrolador e o LabVIEW.

4.1 Carga

Durante os testes de carga, encontramos alguns desafios relacionados à leitura da corrente no sistema. Identificamos que o problema estava na leitura do conversor analógico-digital (ADC) do ESP32. A amplificação inadequada do sinal, resultou em uma leitura imprecisa da corrente, afetando a qualidade dos dados coletados.

Essa limitação no *hardware* impactou diretamente a capacidade do sistema de fornecer informações confiáveis durante o processo de carga da bateria. A leitura

imprecisa da corrente comprometeu a capacidade de determinar com precisão o estado de carregamento da bateria e fornecer um diagnóstico adequado.

Diante desses resultados, fica evidente a necessidade de revisar e melhorar o circuito de amplificação de tensão para garantir uma leitura precisa da corrente durante a fase de carga. Isso exigirá ajustes no *hardware* do sistema, buscando um desempenho otimizado do ADC e uma leitura mais confiável dos sensores envolvidos.

Em projetos futuros, é recomendável dedicar mais atenção à etapa de amplificação de tensão e ao projeto do circuito de carga, visando garantir uma leitura precisa da corrente e, consequentemente, um diagnóstico mais confiável do estado da bateria durante o processo de carga.

5 CONCLUSÃO

De posse das informações obtidas através das bibliografias estudadas e após avaliarmos os resultados dos testes efetuados, conclui-se que em baterias automotivas o valor do CCA deve ser o principal parâmetro considerado e que a vida útil da bateria depende diretamente da quantidade e qualidade dos ciclos de carga.

O trabalho apresenta de forma simplificada que testes com carga efetiva são os mais indicados e aceitos para testar a qualidade da bateria, pois eles simulam a partida do automóvel. Os testes indiretos por medição de resistência interna geram erros de resultados, podendo induzir a um falso diagnóstico e consequentemente uma troca prematura de bateria.

Observamos que quando o valor de tensão máxima de recarga é respeitado, a corrente fornecida pelo carregador tem pouca influência sobre os aspectos de sobrecarga, uma vez que a bateria drena do carregador apenas o quanto ela precisa para alcançar o valor de tensão estipulado.

Dada à importância do assunto, torna-se necessário o desenvolvimento e aplicação de métodos eficientes de recarga e testes efetivos sobre as baterias automotivas. Dessa maneira pode-se economizar recursos naturais que são finitos e diminuir o risco de acidentes ambientais causados pelo excessivo descarte de baterias de chumbo-ácido.

5.1 PROJETOS FUTUROS

Considerando o sistema atual de avaliação da saúde da bateria de chumbo ácido por meio da descarga, identificamos algumas áreas em que melhorias podem ser implementadas, visando aprimorar o desempenho e funcionalidades do sistema. Os seguintes projetos futuros são recomendados:

- **Fonte de Carga Aprimorada:** Uma fonte de carga própria pode ser desenvolvida, levando em consideração as características específicas das baterias de chumbo ácido. Uma fonte de carga dedicada oferecerá maior controle sobre o processo de carregamento, permitindo uma carga mais precisa e eficiente, contribuindo assim para a precisão das medições e prolongamento da vida útil das baterias.
- **Melhoria na Leitura dos Sensores:** Uma análise detalhada das limitações do ADC (Conversor Analógico-Digital) embutido no ESP32

pode ser realizada, buscando soluções para melhorar a precisão e a faixa de leitura dos sensores. Isso pode envolver o uso de técnicas de amplificação, filtragem e calibração para obter medições mais confiáveis e precisas.

- **Implementação de *Datalogger* no LabVIEW:** Um módulo de *datalogger* pode ser integrado ao software LabVIEW, permitindo o armazenamento e análise dos dados coletados durante os testes de avaliação da saúde da bateria. Com essa funcionalidade, será possível gerar relatórios mais detalhados e realizar análises estatísticas dos resultados obtidos, facilitando a interpretação dos dados e o monitoramento da saúde das baterias ao longo do tempo.

- **Melhoria no Circuito de Carga:** O circuito de carga utilizado no sistema pode ser aprimorado para garantir uma carga mais estável e precisa. Isso pode envolver o uso de técnicas de controle de carga, como algoritmos de carga inteligente e monitoramento constante dos parâmetros de carga, garantindo que a bateria seja carregada de maneira adequada e segura.

6 REFERÊNCIAS

ABNT. NBR 6023: **Informação e documentação: Referências** - Elaboração. Rio de Janeiro, 2002.

Fonte: ALMEIDA, Rodrigo Maximiano A.; MORAES, Carlos Henrique V.; SERAPHIM, Thatyana F. Piola. Programação de Sistemas Embarcados. 1. ed. São Paulo: Grupo Gen., 2021. p. 292.

BOSCH, **Manual de Baterias Bosch**, Material de distribuição da Bosch Brasil, 2007.

TUDOR, **Treinamento técnico em Baterias Automotivas**, 2017.

BOCCHI, Luiz Carlos Ferracin; BOCCHI, Silvia Regina Bolfarini B. **Pilhas e Baterias: Funcionamento e Impacto Ambiental**. Disponível em: http://qnint.sbq.org.br/qni/popup_visualizarConceito.php?idConceito=45&semFrame=1. Acessado em: 05 de junho de 2022.

BRINGIT. **Baterias: história e evolução da tecnologia com o passar do tempo**. Disponível em: https://www.bringit.com.br/blog/curiosidades/a-evolucao-das-baterias. Acessado em: 05 de junho de 2022.

CUNHA, Daniel Willian da; SENDA, Otávio Nizo, **Aplicação de dispositivos móveis em sistemas embarcados – "Monitoramento de Bateria**s", Trabalho de Conclusão de Curso apresentado à FATEC Santo André, 2015.

INMETRO. **Portaria n.º 299, de 14 de junho de 2012.** Disponível em: http://www.inmetro.gov.br/legislacao/rtac/pdf/rtac001843.pdf. Acessado em 08 de junho 2023

EMBARCADOS – **BATERIAS**, disponível em: https://embarcados.com.br/baterias-de-chumbo-acido/ . Acessado em: 07 de junho de 2023.

ESP32-DEV-KIT-DevKitC-v4, **pinout-mischianti.jpg**. Disponível em: https://www.mischianti.org/wp-content/uploads/2021/07/ESP32-DEV-KIT-DevKitC-v4-pinout-mischianti.jpg. Acessado em: 05 de junho de 2022.

LUNA FILHO, Gustavo José, **Previsão da autonomia de baterias chumbo-acido aplicadas a sistemas híbridos de geração de energia utilizando o método KiBaM**, Dissertação de Mestrado apresentado ao Programa de Pós Graduação em Engenharia Elétrica na Universidade Federal de Pernambuco, 2017.

NIKROTHAL 80. Disponível em: https://www.kanthal.com/pt-br/produtos-e-servi%C3%A7os/folhas-de-dados-do-material/fio/fio-de-resist%C3%AAncia-de-aquecimento-e-fio-da-resist%C3%AAncia/nikrothal-80. Acessado em: 05 de junho de 2022.

RONTEK. **Carregamento das Baterias Chumbo-Ácido**. Disponível em: <https://www.sta-eletronica.com.br/artigos/baterias-recarregaveis/baterias-de-chumbo/como-carregar-uma-bateria-selada-de-chumbo-acido>. Acessado em: 05 de junho de 2022.

THATYANA, F. Piola. **Programação de Sistemas Embarcados**. 1. ed. São Paulo: Grupo Gen, 2021. p. 291-292.

TECHTUDO. **Entenda como funciona uma fonte chaveada e suas vantagens e desvantagens**. Disponível em: https://www.techtudo.com.br/noticias/2015/02/entenda-como-funciona-uma-fonte-chaveada-e-suas-vantagens-e-desvantagens.ghtml. Acessado em: 05 de junho de 2022.

VALE, C. das Baterias do. **O que é o CCA de uma bateria?** Disponível em: https://casadasbateriasdovale.com.br/o-que-e-o-cca-de-uma-bateria. Acessado em: 05 de junho de 2022.

ENERGIZER. **História da primeira pilha**. Disponível em: https://energizer.lat/Brasil/historia-da-primeira-pilha/#:~:text=1700,umedecidos%20em%20uma%20solu%C3%A7%C3%A3o%20%C3%A1cida. Acessado em 08 de junho de 2023.

UFRGS. **Capacitor**. disponível em https://www.ufrgs.br/amlef/glossario/capacitor-2/. Acessado em 08 de junho de 2023.

MUSEU INTERATIVO DA FÍSICA. **GARRAFA DE LEYDEN**. disponível em https://minf.ufpa.br/garrafa-de-leyden acessado em 08 de junho 2023.

ENGENHARIA ELÉTRICA - UNIFACS. **Pilha de Daniell**. disponível em https://www.eecis.udel.edu/~portnoi/academic/academic-files/daniellcell.html Acessado em: 07 de junho de 2023.

WIKIPEDIA-VOLTA, https://pt.wikipedia.org/wiki/Alessandro_Volta. Acessado em 07 de junho de 2023.

7 Apêndice A: Código completo do firmware

Main.c

```
//=====================================================================
//                          Fatec Santo André                         //
//=====================================================================
// Data: <05/07/2022>                                                 //
//                                                                    //
// Historico:                   Iniciais              Motivo da Mudança  //
//                           do Projetista                            //
//                                                                    //
// 25/02/2023                    TBMS                 Versão inicial. V0 //
// IDE: VSCode                                                        //
// Compilador: ESP-IDF 4.4.4                                          //
//=====================================================================
/**
 * \file            main.c
 * \brief           Título do seu projeto
 * \author          Tiago Barbosa <tiago.silva195@fatec.sp.gov.br>
 * \copyright       Copyright (c) ano Tiago Barbosa
 * \license         Este projeto é de código aberto. Você pode copiar,
 *                  modificar e distribuir o software livremente, desde que
 *                  atribua os devidos créditos a Tiago Barbosa.
 */
//=====================================================================//
// Proposito: Plataforma de Diagnóstico de Baterias de Chumbo Ácido   //
//=====================================================================//

//=====================================================================//
//          Inclusao das Bibliotecas necessarias para a Aplicacao      //
//=====================================================================//
#include <stdio.h>
#include "globais.h"
#include "system.h"

#include "esp_timer.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/queue.h"
#include "freertos/semphr.h"
#include "freertos/timers.h"
#include "tasks.h"
#include "system.h"
#include "structs.h"
#include "definicoes.h"
#include "driver/adc.h"
#include "queue.h"
#include "event_group.h"
#include "timersRTOS.h"

/*************************************************************************
 * @brief Funcao principal da aplicacao
 *
 *************************************************************************/
void app_main(void){
    init_system();
    init_ADCs();

    //-------------------------------------------------------------------------
    //      Criação dos Timers
    //-------------------------------------------------------------------------
    xTimerUm = xTimerCreate("Timer", pdMS_TO_TICKS(TEMPO_TESTE_DESCARGA), pdFALSE, 0,
vTimerCallbackTesteUm);
```

```
    xTimerDois = xTimerCreate("Timer", pdMS_TO_TICKS(TEMPO_TESTE_DESCARGA), pdFALSE, 0,
vTimerCallbackTesteDois);
    xTimerTres = xTimerCreate("Timer", pdMS_TO_TICKS(TEMPO_TESTE_DESCARGA), pdFALSE, 0,
vTimerCallbackTesteTres);
    xTimerQuatro = xTimerCreate("Timer", pdMS_TO_TICKS(TEMPO_TESTE_DESCARGA), pdFALSE, 0,
vTimerCallbackTesteQuatro);
    xTimerCinco = xTimerCreate("Timer", pdMS_TO_TICKS(TEMPO_TESTE_DESCARGA), pdFALSE, 0,
vTimerCallbackTesteCinco);
    xTimerSeis = xTimerCreate("Timer", pdMS_TO_TICKS(TEMPO_TESTE_DESCARGA), pdFALSE, 0,
vTimerCallbackTesteSeis);
    xTimerSete = xTimerCreate("Timer", pdMS_TO_TICKS(TEMPO_TESTE_DESCARGA), pdFALSE, 0,
vTimerCallbackTesteSete);
    xTimerOito = xTimerCreate("Timer", pdMS_TO_TICKS(TEMPO_TESTE_DESCARGA), pdFALSE, 0,
vTimerCallbackTesteOito);

    Evt_Diretoria = xEventGroupCreate();
    //---------------------------------------------------------------------------
    //      Criação dos objetos QUEUE do RTOS
    //---------------------------------------------------------------------------
    Q_DadosSensores = xQueueCreate(1, (sizeof(dadosComm)));
    Q_TipoTeste     = xQueueCreate(1, sizeof(uint8_t));
    Q_Txt           = xQueueCreate(1, 100);
    //---------------------------------------------------------------------------
    //      Cria as tarefas do RTOS
    //---------------------------------------------------------------------------
    xTaskCreatePinnedToCore(xGerenciaSensores, "xGerenciaSensores", 2048, NULL, 0,
&xGerenciaSensoresHandle, 0);
    xTaskCreatePinnedToCore(xGerenciaUart, "xGerenciaUart", 4000, NULL, 0,
&xGerenciaUartHandle, 0);
    xTaskCreatePinnedToCore(xDiretoria, "xDiretoria", 2048, NULL, 0, &xDiretoriaHandle, 0);
    xTaskCreatePinnedToCore(xGerenciaAtuadores, "xGerenciaAtuadores", 2048, NULL, 0,
&xGerenciaAtuadoresHandle, 0);
    xTaskCreatePinnedToCore(xGerenciaCarregador, "xGerenciaCarregador", 2048, NULL, 0,
&xGerenciaCarregadorHandle, 0);
    xTaskCreatePinnedToCore(xGerenciaTestador, "xGerenciaTestador", 2048, NULL, 0,
&xGerenciaTestadorHandle, 0);
    xTaskCreatePinnedToCore(xHearthBit, "xHearthBit", 1024, NULL, 0, NULL, 0);
    xTaskCreatePinnedToCore(xAlarme, "xAlarme", 2048, NULL, 0, &xAlarmeHandle, 0);

    vTaskSuspend(xGerenciaCarregadorHandle);
}
```

TimersRTOS.h

```
/**
 * @file            timersRTOS.c
 * @brief           Arquivo de cabeçalho para configuração de timers com o FreeRTOS
 * @author          Tiago Barbosa <tiago.silva195@fatec.sp.gov.br>
 * @copyright       Copyright (c) ano Tiago Barbosa
 * @license         Este projeto é de código aberto. Você pode copiar,
 *                  modificar e distribuir o software livremente, desde que
 *                  atribua os devidos créditos a Tiago Barbosa.
 */

#ifndef TIMERSRTOS_H_
#define TIMERSRTOS_H_

/**
 * @brief Callback do timer para execução do teste um
 * @param xTimer handle do timer que gerou a interrupção
 */
void vTimerCallbackTesteUm(TimerHandle_t xTimer);

/**
 * @brief Callback do timer para execução do teste dois
 * @param xTimer handle do timer que gerou a interrupção
 */
void vTimerCallbackTesteDois(TimerHandle_t xTimer);

/**
 * @brief Callback do timer para execução do teste três
 * @param xTimer handle do timer que gerou a interrupção
 */
void vTimerCallbackTesteTres(TimerHandle_t xTimer);

/**
 * @brief Callback do timer para execução do teste quatro
 * @param xTimer handle do timer que gerou a interrupção
 */
void vTimerCallbackTesteQuatro(TimerHandle_t xTimer);

/**
 * @brief Callback do timer para execução do teste cinco
 * @param xTimer handle do timer que gerou a interrupção
 */
void vTimerCallbackTesteCinco(TimerHandle_t xTimer);

/**
 * @brief Callback do timer para execução do teste seis
 * @param xTimer handle do timer que gerou a interrupção
 */
void vTimerCallbackTesteSeis(TimerHandle_t xTimer);

/**
 * @brief Callback do timer para execução do teste sete
 * @param xTimer handle do timer que gerou a interrupção
 */
void vTimerCallbackTesteSete(TimerHandle_t xTimer);

/**
 * @brief Callback do timer para execução do teste oito
 * @param xTimer handle do timer que gerou a interrupção
 */
void vTimerCallbackTesteOito(TimerHandle_t xTimer);

void vTimerCallbackTesteUm(TimerHandle_t xTimer){
    xTaskNotify(xGerenciaTestadorHandle, TESTE_UM, eSetValueWithOverwrite);
}
```

```
void vTimerCallbackTesteDois(TimerHandle_t xTimer){
    xTaskNotify(xGerenciaTestadorHandle, TESTE_DOIS, eSetValueWithOverwrite);
}
void vTimerCallbackTesteTres(TimerHandle_t xTimer){
    xTaskNotify(xGerenciaTestadorHandle, TESTE_TRES, eSetValueWithOverwrite);
}
void vTimerCallbackTesteQuatro(TimerHandle_t xTimer){
    xTaskNotify(xGerenciaTestadorHandle, TESTE_QUATRO, eSetValueWithOverwrite);
}
void vTimerCallbackTesteCinco(TimerHandle_t xTimer){
    xTaskNotify(xGerenciaTestadorHandle, TESTE_CINCO, eSetValueWithOverwrite);
}
void vTimerCallbackTesteSeis(TimerHandle_t xTimer){
    xTaskNotify(xGerenciaTestadorHandle, TESTE_SEIS, eSetValueWithOverwrite);
}
void vTimerCallbackTesteSete(TimerHandle_t xTimer){
    xTaskNotify(xGerenciaTestadorHandle, TESTE_SEIS, eSetValueWithOverwrite);
}
void vTimerCallbackTesteOito(TimerHandle_t xTimer){
    xTaskNotify(xGerenciaTestadorHandle, TESTE_OITO, eSetValueWithOverwrite);
}

#endif
```

Definições.h

```
//===========================================================================
//                          Fatec Santo André                                //
//===========================================================================
// Data: <06/03/2023>                                                       //
//                                                                          //
// Historico:                    Iniciais              Motivo da Mudança    //
//                             do Projetista                                //
//                                                                          //
// 06/03/2023                     TBMS                 Versão inicial. V0   //
//===========================================================================//
//===========================================================================//
// Proposito: Inclui todos os DEFINES da aplicacao                          //
//===========================================================================//
#ifndef DEFINICOES_H_
#define DEFINICOES_H_

#include <stdio.h>   // Para a função sprintf
#include <string.h>  // Para a função strlen
#include "driver/uart.h"  // Para a função uart_write_bytes
//===========================================================================//
//                          Definições das Pinagens                          //
//===========================================================================//
//                                 Entradas                                  //
#define LOG_FATEC 1
#if LOG_FATEC == 1
#define LOG_FATEC(texto) uart_write_bytes(UART_NUM_0, texto "\n", strlen(texto) + 1)
#else
#define LOG_FATEC(texto)
#endif

#define LOG_FATEC_PRINT_ATIVADO 1
#if LOG_FATEC_PRINT_ATIVADO == 1
#define LOG_FATEC_PRINT(fmt, ...) \
 do{ \
    char buffer[100]; \
    sprintf(buffer, fmt, __VA_ARGS__); \
    uart_write_bytes(UART_NUM_0, buffer, strlen(buffer)); \
 } while(0)
 #endif

#define SENSOR_TEMPERATURA GPIO_NUM_34
#define SENSOR_TENSAO      GPIO_NUM_35
#define SENSOR_CORRENTE    GPIO_NUM_32

//                                  Saídas                                   //
#define RELE_CARGA         GPIO_NUM_13
#define RELE_COOLERS       GPIO_NUM_12
#define RELE_UM            GPIO_NUM_14
#define RELE_DOIS          GPIO_NUM_27
#define RELE_TRES          GPIO_NUM_26
#define RELE_QUATRO        GPIO_NUM_25
#define BUZZER_IO          GPIO_NUM_23

//===========================================================================
//         Definições basicas da Aplicacao
//===========================================================================
#define DESLIGAR              0              //Comando para Desligar algo na aplicacao
#define LIGAR                 1              //Comando para ligar algo na aplicacao
//===========================================================================
//         Definições de tempo da aplicação
//===========================================================================
#define DELAY_xAPP 50
//===========================================================================
//         Definições da Task xGerenciaAtuadores
```

```
//=============================================================================

#define TESTAR_DESCARGA_70A         1
#define TESTAR_DESCARGA_140A        2
#define LIGAR_VENTILADOR            3
#define DESLIGAR_VENTILADOR         4
#define DESLIGAR_TEMPERATURA_LIMITE 5
#define DESLIGAR_GERAL              6

//=============================================================================
//         Definições da Task xGerenciaAtuadores
//=============================================================================
#define INICIA_CARGA_BATERIA 1

#define ACS712_SENSITIVITY 66.0 // mV/A
#define ACS712_OFFSET 0.41 // mV

#define TESTE_UM         1
#define TESTE_DOIS       2
#define TESTE_TRES       3
#define TESTE_QUATRO     4
#define TESTE_CINCO      5
#define TESTE_SEIS       6
#define TESTE_SETE       7
#define TESTE_OITO       8
#define CARREGAR_BATERIA 9

#define TEMPO_TESTE_DESCARGA 15000
#define TEMPO_TESTE_DESCANSO 45000

#define TODOS_OS_BITS  (BIT0 | BIT1 | BIT2 | BIT3 | BIT4 | BIT5 | BIT6 | BIT7 | BIT8 | BIT9
| BIT10)


#define ALARME_CONFIRMA 1
#define ALARME_FALHA 2
#define ALARME_TESTE_CONCLUIDO 3
```

Event_group.h

```
//=====================================================================
//                          Fatec Santo André                         //
//=====================================================================
// Data: <06/03/2023>                                                 //
//                                                                    //
// Historico:                    Iniciais              Motivo da Mudança   //
//                             do Projetista                          //
//                                                                    //
// 06/03/2023                     TBMS                 Versão inicial. V0  //
//=====================================================================//
//=====================================================================//
// Proposito: Inclui todos os handles das queues do RTOS              //
//=====================================================================//
#ifndef EVENT_GROUP_H_
#define EVENT_GROUP_H_
//=========================================================================
//     Inclusao das Bibliotecas necessarias para a Aplicacao
//=========================================================================
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/queue.h"
#include "freertos/semphr.h"
/*************************************************************************
 * Nome:            Evt_Diretoria
 * Descrição:       Responsável por controlar toda operação do Sistema Operacional
 *
 * Bits:
 *   - Bit 0:  Bateria está em modo Carga
 *   - Bit 1:  Teste de Carga 70A
 *   - Bit 2:  Teste de Carga 140A
 *   - Bit 3:  Temperatura acima do 50C
 *   - Bit 4:  Tempo de Espera após o teste.
 *   - Bit 5:  Corrente de Carga proximo a 0A.
 *   - Bit 6:  Tensao entre 12.2V e 13.2V
 *   - Bit 7:  Tensão abaixo de 10V
 *   - Bit 8:  Bateria em modo Teste
 *   - Bit 9:  Tensão acima ou maior de 14.3
 *   - Bit 10: Inicia comunicação
 *
 * Exemplo de uso:
 *
 *  A task Diretoria vai ficar responsável por monitorar esses bits e tomar alguma ação.
 *
 *************************************************************************/
EventGroupHandle_t Evt_Diretoria;

#endif //
```

Globais.h

```
//===========================================================================
//                          FATEC TECNOLOGIA                                 //
//===========================================================================
// Data: <16/07/2022>                                                        //
//                                                                           //
// Historico:                    Iniciais              Motivo da Mudança     //
//                             do Projetista                                 //
//                                                                           //
// 16/11/2022                     TBMS               Versão inicial. V0      //
//===========================================================================

//===========================================================================
// Proposito: Inclui todos as variaveis GLOBAIS e estruturas de dados da
//            aplicacao
//===========================================================================
#ifndef GLOBAIS_H_
#define GLOBAIS_H_
//===========================================================================
//     Inclusao das Bibliotecas necessarias para a Aplicacao
//===========================================================================
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/queue.h"
#include "freertos/semphr.h"
#include "freertos/event_groups.h"
//===========================================================================
//     Definições das estruturas de dados do display 16x4
//===========================================================================
typedef enum {
    ERRO         = 0u, // Funcao encontrou um erro em sua execucao
    CERTO        = 1u, // Funcao executou seu proposito adequadamente
    NOK          = 2u, // Funcao nao executou seu proposito adequadamente
    CANCELA      = 3u, // Funcao cancelou a execucao de sua rotina antes do termino
    NAOPRESENTE  = 4u, // Funcao nao encontrou o canal ou o id especificado
    NAOADD       = 5u, // Funcao entendeu que o canal ou ID nao foi adicionado a moto
    LIMITE       = 6u, // Funcao atingiu o limite de tentativas
    PROXIMO      = 7u  // Funcao que identificou que existe outro canal ou id na moto
} CommSts;              // Status de Comunicacao das Funcoes da aplicacao

//===========================================================================
//     Definições dos Handles do RTOS
//===========================================================================
TaskHandle_t xGerenciaSensoresHandle;
TaskHandle_t xGerenciaUartHandle;
TaskHandle_t xGerenciaAtuadoresHandle;
TaskHandle_t xDiretoriaHandle;
TaskHandle_t xGerenciaCarregadorHandle;
TaskHandle_t xGerenciaTestadorHandle;
TaskHandle_t xAlarmeHandle;

TimerHandle_t xTimerUm;
TimerHandle_t xTimerDois;
TimerHandle_t xTimerTres;
TimerHandle_t xTimerQuatro;
TimerHandle_t xTimerCinco;
TimerHandle_t xTimerSeis;
TimerHandle_t xTimerSete;
TimerHandle_t xTimerOito;

#endif // GLOBAIS_H_
```

Queue.h

```
//===========================================================================
//                          Fatec Santo André                                //
//===========================================================================
// Data: <06/03/2023>                                                       //
//                                                                          //
// Historico:                    Iniciais               Motivo da Mudança   //
//                             do Projetista                                //
//                                                                          //
// 06/03/2023                     TBMS                  Versão inicial. V0  //
//==========================================================================//
//==========================================================================//
// Proposito: Inclui todos os handles das queues do RTOS                    //
//==========================================================================//
#ifndef QUEUE_H_
#define QUEUE_H_
//===========================================================================
//     Inclusao das Bibliotecas necessarias para a Aplicacao
//===========================================================================
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/queue.h"
#include "freertos/semphr.h"

QueueHandle_t Q_DadosSensores;
QueueHandle_t Q_TipoTeste;
QueueHandle_t Q_Txt;

#endif // STRUCTS_H
```

Structs.h

```
//===========================================================================
//                          Fatec Santo André                                //
//===========================================================================
// Data: <06/03/2023>                                                       //
//                                                                          //
// Historico:                    Iniciais               Motivo da Mudança   //
//                             do Projetista                                //
//                                                                          //
// 06/03/2023                     TBMS                  Versão inicial. V0  //
//==========================================================================//
//==========================================================================//
// Proposito: Inclui todo tipo de structs do projeto                        //
//==========================================================================//
#ifndef STRUCTS_H_
#define STRUCTS_H_
//===========================================================================
//     Inclusao das Bibliotecas necessarias para a Aplicacao
//===========================================================================
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/queue.h"
#include "freertos/semphr.h"

typedef struct{
    float id;
    float corrente;
    float tensao;
    float temperatura;
} dadosComm;

#endif // STRUCTS_H
```

System.h

```
//=====================================================================
//                        Fatec Santo André                           //
//=====================================================================
// Data: <14/10/2022>                                                 //
//                                                                    //
// Historico:                    Iniciais              Motivo da Mudança   //
//                             do Projetista                          //
//                                                                    //
// 03/03/2023                     TBMS                 Versão inicial. V0  //
//=====================================================================

//=====================================================================
// Proposito: Inclui o prototipo das funcoes elaboradas para inicializacao
//            dos perifericos da ESP32, como GPIO, PWM, TIMERS entre outros
//=====================================================================
#ifndef SYSTEM_H_
#define SYSTEM_H_
//=====================================================================
//     Inclusao das Bibliotecas necessarias para a Aplicacao
//=====================================================================
#include "globais.h"
#include "driver/gpio.h"
#include "driver/ledc.h"
#include "driver/adc.h"
#include "driver/uart.h"
/**
 * @brief Utilizado para iniciar os perifericos da ESP32
 *
 * @return CommSts Retorno de Status da Comunicação
 */
CommSts init_system(void);

CommSts init_ADCs(void);
#endif //SYSTEM_H_
```

System.c

```
//=====================================================================
//                          Fatec Santo André                         //
//=====================================================================
// Data: 26/02/2023                                                   //
//                                                                    //
// Historico:                    Iniciais              Motivo da Mudança   //
//                             do Projetista                          //
//                                                                    //
// 26/02/2023                     TBMS                 Versão inicial. V0  //
//=====================================================================
//=====================================================================
// Proposito: Inclui funcoes elaboradas para interface com o
//            hardware do ESP32
//=====================================================================
#ifndef SYSTEM_C
#define SYSTEM_C
//=====================================================================//
//          Inclusao das Bibliotecas necessarias para a Aplicacao      //
//=====================================================================//
#include "system.h"
#include "definicoes.h"
#include "driver/adc.h"
#include "driver/uart.h"
```

```
CommSts init_system(void){
    CommSts Status = CERTO;
    // Configuracao Direcao dos Pinos - SAIDA
////////////////////////////////////////////////
    gpio_config_t io_conf;                      // Ponteiro de configuração dos pinos
    io_conf.intr_type = GPIO_PIN_INTR_DISABLE; // Desabilita Interrupçoes
    io_conf.mode = GPIO_MODE_OUTPUT;           // configura no modo SAIDA
    // io_conf.pin_bit_mask = ((1ULL << BUZZER_IO) | (1ULL << POWER_ON_OFF) | (1ULL <<
GPIO_NUM_17));
    io_conf.pin_bit_mask = ((1ULL << BUZZER_IO) | (1ULL << RELE_CARGA) | (1ULL <<
RELE_COOLERS) |
                            (1ULL << RELE_UM)   | (1ULL << RELE_DOIS)  | (1ULL <<
RELE_TRES)    |
                            (1ULL << RELE_QUATRO) | (1ULL << GPIO_NUM_2));
    io_conf.pull_down_en = 0; // Desabilita modo PULL-DOWN
    io_conf.pull_up_en = 0;   // Desabilita modo PULL-UP
    gpio_config(&io_conf);    // Configura GPIO com as configurações para SAIDA

    // // Configuracao Direcao dos Pinos - ENTRADA
////////////////////////////////////////////////
    // io_conf.pin_bit_mask = DESLIGAR;           //Zera a mascara por garantia
    // io_conf.pin_bit_mask = ((1ULL << TECLA_DIREITA)   | (1ULL << TECLA_ENTER) |
    //                         (1ULL << TECLA_ESQUERDA)  | (1ULL << TECLA_VOLTAR)); //
Defini pinos de entrada
    // io_conf.mode = GPIO_MODE_INPUT;            // configura no modo ENTRADA
    // io_conf.pull_up_en = LIGAR;                // Habilita modo PULL-UP
    // gpio_config(&io_conf);                     // Configura GPIO com as configurações
para ENTRADA

    // Configuracao Direcao dos Pinos - ADC
////////////////////////////////////////////////
    io_conf.pin_bit_mask = DESLIGAR;           // Zera a mascara por garantia
    io_conf.pin_bit_mask = ((1ULL << SENSOR_CORRENTE) | (1ULL << SENSOR_TEMPERATURA) |
(1ULL << SENSOR_TENSAO));    // Defini mascara para pinos ADC
    io_conf.pull_up_en = DESLIGAR;             // Desabilita modo PULL-UP
    gpio_config(&io_conf);                     // Configura GPIO com as configurações para
ADC

    uart_config_t uart_config = {
        .baud_rate = 9600,
        .data_bits = UART_DATA_8_BITS,
        .parity = UART_PARITY_DISABLE,
        .stop_bits = UART_STOP_BITS_1,
        .flow_ctrl = UART_HW_FLOWCTRL_DISABLE};

    uart_param_config(UART_NUM_0, &uart_config);
    uart_set_pin(UART_NUM_0, GPIO_NUM_1, GPIO_NUM_3, UART_PIN_NO_CHANGE,
UART_PIN_NO_CHANGE);
    uart_driver_install(UART_NUM_0, 1024, 0, 0, NULL, 0);

    return Status;
}

CommSts init_ADCs(void){
    CommSts Status = CERTO;

    adc1_config_width(ADC_WIDTH_12Bit);
    adc1_channel_t canais[] = {ADC_CHANNEL_6, ADC_CHANNEL_7, ADC_CHANNEL_4};
    adc1_config_channel_atten(canais[0], ADC_ATTEN_DB_11);
    adc1_config_channel_atten(canais[1], ADC_ATTEN_DB_11);
    adc1_config_channel_atten(canais[2], ADC_ATTEN_DB_11);

    return Status;
}
#endif // SYSTEM_H
```

Tasks.h

```
//=====================================================================
//                          Fatec Santo André                         //
//=====================================================================
// Data: <14/07/2022>                                                 //
//                                                                    //
// Historico:                    Iniciais              Motivo da Mudança   //
//                             do Projetista                          //
//                                                                    //
// 03/06/2023                     TBMS                 Versão inicial. V0  //
//=====================================================================
//=====================================================================//
// Proposito: Inclui as tarefas do sistema operaciona embarcado da aplicacao //
//=====================================================================//
#ifndef TASKS_H_
#define TASKS_H_

//=====================================================================//
//==      Inclusao das Bibliotecas necessarias para o funcionamento          //
//=====================================================================//
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/queue.h"
#include "freertos/event_groups.h"
#include "freertos/semphr.h"
#include "freertos/timers.h"
#include "driver/gpio.h"
#include "driver/uart.h"
#include "definicoes.h"
#include "globais.h"
#include "system.h"
#include "string.h"
#include "structs.h"
#include "queue.h"
#include "event_group.h"

/*****************************************************************************************
*
 * @brief Responsavel por realizar a leitura do ADC dos sensores, armazenar as informações
dentro
 *        de um vetor e enviar via queue para a task diretoria.
 *        Tambem seta o bit 3 do Event Group Evt_Diretoria caso a temperatura atinja um
valor maior de 50C
 *
 * @param arg ponteiro da task
 *****************************************************************************************
/
void xGerenciaSensores(void *arg) {
    const float tensaoRef = 3.3;  // Tensão de referência do ADC
    const float resistor1 = 10000.0;  // Valor do resistor R1
    const float resistor2 = 2000.0;  // Valor do resistor R2
    const float offset = 2.5;  // Offset do sensor ACS712 em volts
    const float sensibilidade = 0.066;  // Sensibilidade do sensor ACS712 em V/A (66 mV/A)

    const float tensaoMaxima = 75.0;  // Tensão máxima (mV) correspondente a 50A
    const float correnteMaxima = 50.0;  // Corrente máxima (A) correspondente a 75mV
    const float ganhoAmplificador = 201.0;  // Ganho do amplificador

    float valorTensao;
    float valorCorrente;
    float valorTemperatura;
    float resistencia;
    float xTemperatura;
```

```
    float armazenaDados[3];

    int colunaX[34] = {10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90,
95, 100, 105, 110, 115, 120, 125};
    int colunaY[34] = {39000, 25450, 17300, 12060, 8695, 6415, 4815, 3715, 2925, 2345,
1910, 1575, 1315, 1105, 931, 788, 668, 566, 478, 401, 339, 287, 245, 210};

    dadosComm dadosSensores;

    while(true) {
        valorTensao = adc1_get_raw(ADC_CHANNEL_7);  // Leitura da tensão do ADC
        valorCorrente = adc1_get_raw(ADC_CHANNEL_4);  // Leitura de tensão do ADC
        valorTemperatura = adc1_get_raw(ADC_CHANNEL_6);  // Leitura de tensãode ADC

        valorTemperatura = (valorTemperatura * tensaoRef) / 4095;

        float correnteReal = (valorCorrente * correnteMaxima * ganhoAmplificador) /
(tensaoMaxima * 1000.0);

        if (correnteReal < 0.0) {
            correnteReal = 0.0;  // Limita a corrente mínima a 0A
        } else if (correnteReal > 15.0) {
            correnteReal = 15.0;  // Limita a corrente máxima a 15A
        }
        armazenaDados[0] = correnteReal;

        resistencia = (valorTemperatura * resistor1) / (tensaoRef - valorTemperatura);
        valorTensao = (valorTensao * tensaoRef) / 4095 / 0.157;
        armazenaDados[1] = valorTensao;

        for (int i = 0; i < 10; i++) {
            for (int i = 0; i < 35; i++) {
                if (colunaY[i] > resistencia) {
                    xTemperatura = (((resistencia - colunaY[i]) / (colunaY[i - 1] -
colunaY[i])) * (colunaX[i - 1] - colunaX[i])) + (colunaX[i]);
                }
            }
            mediaLeitura[i] = xTemperatura;
            if (i == 9) {
                for (int j = 0; j < 10; j++) {
                    xTemperatura += mediaLeitura[j];
                    if (j == 9) {
                        xTemperatura = xTemperatura / 11;
                        armazenaDados[2] = xTemperatura;
                    }
                }
            }
        }

        if (armazenaDados[0] <= 0) {
            xEventGroupSetBits(Evt_Diretoria, BIT5);
        } else {
            xEventGroupClearBits(Evt_Diretoria, BIT5);
        }

        if (armazenaDados[1] <= 13.7) {
            xEventGroupSetBits(Evt_Diretoria, BIT7);
        } else {
            xEventGroupClearBits(Evt_Diretoria, BIT7);
        }

        if (armazenaDados[1] >= 14.3) {
            xEventGroupSetBits(Evt_Diretoria, BIT9);
        } else {
            xEventGroupClearBits(Evt_Diretoria, BIT9);
        }
```

```
        if (armazenaDados[1] >= 12.2 && armazenaDados[1] <= 13.8) {
            xEventGroupSetBits(Evt_Diretoria, BIT6);
        } else {
            xEventGroupClearBits(Evt_Diretoria, BIT6);
        }

        if (armazenaDados[2] >= 50.0) {
            xEventGroupSetBits(Evt_Diretoria, BIT3);
        } else {
            xEventGroupClearBits(Evt_Diretoria, BIT3);
        }

        printf("Corrente: %f, Tensao: %f, Temperatura: %f\n", armazenaDados[0],
armazenaDados[1], armazenaDados[2]);
        dadosSensores.corrente = armazenaDados[0];
        dadosSensores.tensao = armazenaDados[1];
        dadosSensores.temperatura = armazenaDados[2];

        xQueueOverwrite(Q_DadosSensores, &dadosSensores);
        vTaskDelay(pdMS_TO_TICKS(100));
    }
}
/*****************************************************************************************
*
 * @brief Responsavel por enviar e receber dados via UART, principal comunicação entre o
microcontrolador
 *        e o Labview.
 *
 *        Labview envia os dados como string:
 *        0 - Desabilitado
 *        1 - 4Ah até 7Ah
 *        2 - 8Ah até 12Ah
 *        3 - 13Ah até 18Ah
 *        4 - 19Ah até 26Ah
 *        5 - 27Ah até 35Ah
 *        6 - 36Ah até 50Ah
 *        7 - 51Ah até 60Ah
 *        8 - 61Ah até 75Ah
 *        9 - Carregar Bateria
 *
 * @param arg ponteiro da task
 *****************************************************************************************
/
void xGerenciaUart(void *arg){

    uint8_t readUART;
    uint8_t bufferUART[256];
    static dadosComm parametros;
    char buffer[500];
    static uint8_t id;

    while (true){

        readUART = uart_read_bytes(UART_NUM_0, bufferUART, sizeof(bufferUART),
pdMS_TO_TICKS(50));
        EventBits_t Bits = xEventGroupWaitBits(Evt_Diretoria, TODOS_OS_BITS, pdFALSE,
pdTRUE, pdMS_TO_TICKS(0));
        xQueuePeek(Q_DadosSensores, ¶metros, pdMS_TO_TICKS(10));
        xQueuePeek(Q_Txt, &id, pdMS_TO_TICKS(10));

        if (readUART > 0){
            if (bufferUART[0] == '1' && bufferUART[1] == NULL){
                LOG_FATEC("4Ah ate 7Ah");
                xTaskNotify(xDiretoriaHandle, TESTE_UM, eSetValueWithOverwrite);
            } else if (bufferUART[0] == '2'){
```

```
                LOG_FATEC("8Ah até 12Ah");
                xTaskNotify(xDiretoriaHandle, TESTE_DOIS, eSetValueWithOverwrite);
            } else if (bufferUART[0] == '3'){
                LOG_FATEC("13Ah até 18Ah");
                xTaskNotify(xDiretoriaHandle, TESTE_TRES, eSetValueWithOverwrite);
            } else if (bufferUART[0] == '4'){
                LOG_FATEC("19Ah até 26Ah");
                xTaskNotify(xDiretoriaHandle, TESTE_QUATRO, eSetValueWithOverwrite);
            } else if (bufferUART[0] == '5'){
                LOG_FATEC("27Ah até 35Ah");
                xTaskNotify(xDiretoriaHandle, TESTE_CINCO, eSetValueWithOverwrite);
            } else if (bufferUART[0] == '6'){
                LOG_FATEC("36Ah até 50Ah");
                xTaskNotify(xDiretoriaHandle, TESTE_SEIS, eSetValueWithOverwrite);
            } else if (bufferUART[0] == '7'){
                LOG_FATEC("51Ah até 60Ah");
                xTaskNotify(xDiretoriaHandle, TESTE_SETE, eSetValueWithOverwrite);
            } else if (bufferUART[0] == '8'){
                LOG_FATEC("61Ah até 75Ah");
                xTaskNotify(xDiretoriaHandle, TESTE_OITO, eSetValueWithOverwrite);
            } else if (bufferUART[0] == '9'){
                LOG_FATEC("Carregar Bateria");
                vTaskResume(xGerenciaCarregadorHandle);
                xTaskNotify(xGerenciaAtuadoresHandle, CARREGAR_BATERIA,
eSetValueWithOverwrite);
                xEventGroupSetBits(Evt_Diretoria, BIT0);
                xEventGroupSetBits(Evt_Diretoria, BIT10);
                //xTaskNotify(xDiretoriaHandle, CARREGAR_BATERIA, eSetValueWithOverwrite);
            } else if (bufferUART[0] == '1' && bufferUART[1] == '0'){
                LOG_FATEC("Desliga tudo");
                xTaskNotify(xGerenciaAtuadoresHandle, DESLIGAR_GERAL,
eSetValueWithOverwrite);
                id = 10;
                xQueueOverwrite(Q_Txt, &id);
            }
            memset(bufferUART, 0, sizeof(bufferUART));
        }

        if(Bits & BIT10){
            sprintf(buffer, "%d %f %.1f %.1f \n", id, parametros.corrente,
parametros.tensao, parametros.temperatura);
            //sprintf(buffer, "%f %.1f %.1f %s", parametros.corrente, parametros.tensao,
parametros.temperatura, txt);
            uart_write_bytes(UART_NUM_0, &buffer, strlen(buffer));
        }

    }
}

/**************************************************************************************
**********
 * @brief Task responsavel por gerenciar os atuadores de acordo com a ordem da task
diretoria
 *
 * @param arg argumento da task
 **************************************************************************************
**********/
void xGerenciaAtuadores(void *arg){

    uint32_t CMD = DESLIGAR;

    while (true){
        CMD = ulTaskNotifyTake(pdTRUE, (pdMS_TO_TICKS(portMAX_DELAY))); // Aguarda o
comando

        switch (CMD){
```

```
        case CARREGAR_BATERIA:
            gpio_set_level(RELE_CARGA, true);
            LOG_FATEC("Rele de carga acionado");
            break;
        case TESTAR_DESCARGA_70A:
            gpio_set_level(RELE_COOLERS, true);
            gpio_set_level(RELE_UM, true);
            LOG_FATEC("TESTAR_DESCARGA_70A");
            break;
        case TESTAR_DESCARGA_140A:
            gpio_set_level(RELE_COOLERS, true);
            gpio_set_level(RELE_DOIS, true);
            gpio_set_level(RELE_TRES, true);
            LOG_FATEC("TESTAR_DESCARGA_140A");
            break;
        case LIGAR_VENTILADOR:
            gpio_set_level(RELE_COOLERS, true);
            LOG_FATEC("LIGAR_VENTILADOR");
            break;
        case DESLIGAR_VENTILADOR:
            gpio_set_level(RELE_COOLERS, false);
            LOG_FATEC("DESLIGAR_VENTILADOR");
            break;
        case DESLIGAR_TEMPERATURA_LIMITE:
            gpio_set_level(RELE_COOLERS, true);
            gpio_set_level(RELE_CARGA, false);
            gpio_set_level(RELE_UM, false);
            gpio_set_level(RELE_DOIS, false);
            gpio_set_level(RELE_TRES, false);
            LOG_FATEC("DESLIGAR_TEMPERATURA_LIMITE");
            break;
        case DESLIGAR_GERAL:
            gpio_set_level(RELE_COOLERS, false);
            gpio_set_level(RELE_CARGA, false);
            gpio_set_level(RELE_UM, false);
            gpio_set_level(RELE_DOIS, false);
            gpio_set_level(RELE_TRES, false);
            LOG_FATEC("DESLIGAR_GERAL");
            break;
        }
    }
}
/*****************************************************************************************
*
 * @brief Responsavel por tomar todas as decisões baseadas na leitura dos event groups
 *
 * @param arg ponteiro da task
 *****************************************************************************************
/
void xDiretoria(void *arg){

    uint32_t CMD = DESLIGAR;
    static dadosComm parametros;
    char buffer[100];
    float bufferDados[10];
    float recebeDadosSensores[3];
    EventBits_t BitsDiretoria;
    static uint8_t id;

    while(true){

        //xQueueReceive(Q_DadosSensores, &recebeDadosSensores, portMAX_DELAY);
        CMD = ulTaskNotifyTake(pdTRUE, (pdMS_TO_TICKS(portMAX_DELAY))); // Aguarda o
comando
        BitsDiretoria = xEventGroupWaitBits(Evt_Diretoria, TODOS_OS_BITS, pdFALSE, pdTRUE,
pdMS_TO_TICKS(1));
```

```
        if ((BitsDiretoria & BIT6) && (!(BitsDiretoria & BIT0))){
            switch (CMD){
            case TESTE_UM:
                if (!(BitsDiretoria & BIT8)){
                    LOG_FATEC("Tensao dentro da faixa, portanto, inicia teste");
                    xTaskNotify(xGerenciaTestadorHandle, TESTE_UM, eSetValueWithOverwrite);
                    id = 50;
                    xQueueOverwrite(Q_Txt, &id);
                    xEventGroupSetBits(Evt_Diretoria, BIT10);
                    vTaskDelay(pdMS_TO_TICKS(1000));
                } else {
                    id = 60;
                    xQueueOverwrite(Q_Txt, &id);
                }
                break;
            case TESTE_DOIS:
                if (!(BitsDiretoria & BIT8)){
                    LOG_FATEC("Tensao dentro da faixa, portanto, inicia teste");
                    id = 50;
                    xQueueOverwrite(Q_Txt, &id);
                    xEventGroupSetBits(Evt_Diretoria, BIT10);
                    vTaskDelay(pdMS_TO_TICKS(1000));
                    xTaskNotify(xGerenciaTestadorHandle, TESTE_DOIS,
eSetValueWithOverwrite);
                } else {
                    id = 60;
                    xQueueOverwrite(Q_Txt, &id);
                }
                break;
            case TESTE_TRES:
                if (!(BitsDiretoria & BIT8)){
                    LOG_FATEC("Tensao dentro da faixa, portanto, inicia teste");
                    id = 50;
                    xQueueOverwrite(Q_Txt, &id);
                    xEventGroupSetBits(Evt_Diretoria, BIT10);
                    vTaskDelay(pdMS_TO_TICKS(1000));
                    xTaskNotify(xGerenciaTestadorHandle, TESTE_TRES,
eSetValueWithOverwrite);
                } else {
                    id = 60;
                    xQueueOverwrite(Q_Txt, &id);
                }
                break;
            case TESTE_QUATRO:
                if (!(BitsDiretoria & BIT8)){
                    LOG_FATEC("Tensao dentro da faixa, portanto, inicia teste");
                    id = 50;
                    xQueueOverwrite(Q_Txt, &id);
                    xEventGroupSetBits(Evt_Diretoria, BIT10);
                    vTaskDelay(pdMS_TO_TICKS(1000));
                    xTaskNotify(xGerenciaTestadorHandle, TESTE_QUATRO,
eSetValueWithOverwrite);
                } else {
                    id = 60;
                    xQueueOverwrite(Q_Txt, &id);
                }
                break;
            case TESTE_CINCO:
                if (!(BitsDiretoria & BIT8)){
                    LOG_FATEC("Tensao dentro da faixa, portanto, inicia teste");
                    id = 50;
                    xQueueOverwrite(Q_Txt, &id);
                    xEventGroupSetBits(Evt_Diretoria, BIT10);
                    vTaskDelay(pdMS_TO_TICKS(1000));
```

```
                    xTaskNotify(xGerenciaTestadorHandle, TESTE_CINCO,
eSetValueWithOverwrite);
                } else {
                    id = 60;
                    xQueueOverwrite(Q_Txt, &id);
                }
                break;
            case TESTE_SEIS:
                if (!(BitsDiretoria & BIT8)){
                    LOG_FATEC("Tensao dentro da faixa, portanto, inicia teste");
                    id = 50;
                    xQueueOverwrite(Q_Txt, &id);
                    xEventGroupSetBits(Evt_Diretoria, BIT10);
                    vTaskDelay(pdMS_TO_TICKS(1000));
                    xTaskNotify(xGerenciaTestadorHandle, TESTE_SEIS,
eSetValueWithOverwrite);
                }else{
                    id = 60;
                    xQueueOverwrite(Q_Txt, &id);
                }
                break;
            case TESTE_SETE:
                if (!(BitsDiretoria & BIT8)){
                    LOG_FATEC("Tensao dentro da faixa, portanto, inicia teste");
                    id = 50;
                    xQueueOverwrite(Q_Txt, &id);
                    xEventGroupSetBits(Evt_Diretoria, BIT10);
                    vTaskDelay(pdMS_TO_TICKS(1000));
                    xTaskNotify(xGerenciaTestadorHandle, TESTE_SETE,
eSetValueWithOverwrite);
                }
                else{
                    id = 60;
                    xQueueOverwrite(Q_Txt, &id);
                }
                break;
            case TESTE_OITO:
                if (!(BitsDiretoria & BIT8)){
                    LOG_FATEC("Tensao dentro da faixa, portanto, inicia teste");
                    id = 50;
                    xQueueOverwrite(Q_Txt, &id);
                    xEventGroupSetBits(Evt_Diretoria, BIT10);
                    vTaskDelay(pdMS_TO_TICKS(1000));
                    xTaskNotify(xGerenciaTestadorHandle, TESTE_OITO,
eSetValueWithOverwrite);
                }
                else {
                    id = 60;
                    xQueueOverwrite(Q_Txt, &id);
                }
                break;
            }
        } else if (CMD == 9){
                //if((!(BitsDiretoria & BIT0)) && ((BitsDiretoria & BIT7))){
                    LOG_FATEC("Modo Carga de Bateria");
                    vTaskResume(xGerenciaCarregadorHandle);
                    xTaskNotify(xGerenciaAtuadoresHandle, CARREGAR_BATERIA,
eSetValueWithOverwrite);
                    xEventGroupSetBits(Evt_Diretoria, BIT0);
                    xEventGroupSetBits(Evt_Diretoria, BIT10);
                //}
        } else {
            if(!(BitsDiretoria & BIT0)){
                for(int i = 0; i < 5; i++){
                    LOG_FATEC("Erro encontrado ao acionar o teste");
                    xTaskNotify(xAlarmeHandle, ALARME_FALHA, eSetValueWithOverwrite);
```

```
                    id = 70;
                    xQueueOverwrite(Q_Txt, &id);
                    vTaskDelay(pdMS_TO_TICKS(5000));
                }
            }
        }
    }
}

// case CARREGAR_BATERIA:
//                 if(!(BitsDiretoria & BIT0)){
//                     LOG_FATEC("Modo Carga de Bateria");
//                     vTaskResume(xGerenciaCarregadorHandle);
//                     xEventGroupSetBits(Evt_Diretoria, BIT0);
//                 }
//             break;

/**
 * @brief Função para gerenciamento do carregador.
 *
 * Essa função é responsável por gerenciar o processo de carregamento da bateria. Ela é
 * executada em uma tarefa e fica constantemente verificando se a tensão e corrente estão
 * dentro da faixa ideal para o carregamento. Caso a tensão esteja acima de 14.4V e a
 * corrente esteja em 0, significa que a carga foi concluída e a função notifica as
 * tarefas de gerenciamento dos atuadores e alarme para desligar o sistema e informar que
 * o teste foi concluído. Caso a temperatura esteja acima do limite, a função também
 * notifica as tarefas para desligar o sistema e informa sobre a temperatura elevada.
 *
 * @param arg Ponteiro para um argumento que pode ser passado para a tarefa (não utilizado
nesta função).
 */
void xGerenciaCarregador(void *arg){

    static uint16_t contadorSeg;
    uint8_t id; // id da mensagem para o labview processar

    //precisa criar um timer que a cada 10 min, ele incrementa uma variavel para controle
de tempo.

    while(true){

        EventBits_t BitsDiretoria = xEventGroupWaitBits(Evt_Diretoria, TODOS_OS_BITS,
pdFALSE, pdTRUE, pdMS_TO_TICKS(1));

        if((BitsDiretoria & BIT9) && (BitsDiretoria & BIT5)){// tensão ja com 14.4v e
corrente 0 finalizar teste
            xTaskNotify(xGerenciaAtuadoresHandle, DESLIGAR_GERAL, eSetValueWithOverwrite);
            xTaskNotify(xAlarmeHandle, ALARME_TESTE_CONCLUIDO, eSetValueWithOverwrite);
            xEventGroupClearBits(Evt_Diretoria, BIT0);
            xEventGroupClearBits(Evt_Diretoria, BIT10);
            for(int i = 0; i < 20; i++){
                id = 90;
                xQueueOverwrite(Q_Txt, &id);
                vTaskDelay(pdMS_TO_TICKS(10));
            }
            LOG_FATEC("Carga concluida");
            vTaskSuspend(xGerenciaCarregadorHandle);
        } else if(BitsDiretoria & BIT3){
            xTaskNotify(xGerenciaAtuadoresHandle, DESLIGAR_GERAL, eSetValueWithOverwrite);
            xEventGroupClearBits(Evt_Diretoria, BIT0);
            LOG_FATEC("Temperatura acima do limite");
        }
        id = 80;
        xQueueOverwrite(Q_Txt, &id);
        LOG_FATEC("Modo Carga ativa");
        vTaskDelay(pdMS_TO_TICKS(100000));
```

```
    }
}
/**

@brief Função para gerenciar a execução do teste

Esta função é responsável por gerenciar a execução do teste. Ela recebe um comando

e executa o teste correspondente, que consiste em dois passos: descarregar a

bateria e deixá-la descansar por um tempo, depois verificar a voltagem.

@param arg Ponteiro vazio para os argumentos da tarefa
*/
void xGerenciaTestador(void *arg){

    float comparaValores[2];
    float recebeDadosSensores[3];
    uint32_t CMD = DESLIGAR;
    xEventGroupClearBits(Evt_Diretoria, BIT10);static uint8_t
    passoAtual = DESLIGAR;
    dadosComm recebeDados;
    uint8_t tipoTeste[2];
    static uint8_t id;
    while(true){

        CMD = ulTaskNotifyTake(pdTRUE, (pdMS_TO_TICKS(portMAX_DELAY))); // Aguarda o
comando
        xQueuePeek(Q_DadosSensores, &recebeDados, pdMS_TO_TICKS(0));
        EventBits_t BitsDiretoria = xEventGroupWaitBits(Evt_Diretoria, TODOS_OS_BITS,
pdFALSE, pdTRUE, pdMS_TO_TICKS(1));
        xTaskNotify(xAlarmeHandle, ALARME_CONFIRMA, eSetValueWithOverwrite);

        //recebeDadosSensores[0] = recebeDados.corrente;
        //recebeDadosSensores[1] = recebeDados.tensao;
        //recebeDadosSensores[2] = recebeDados.temperatura;

        switch (CMD){
        case TESTE_UM:
            xEventGroupSetBits(Evt_Diretoria, BIT8);
            if (passoAtual == 0){
                    xQueuePeek(Q_DadosSensores, &recebeDados, pdMS_TO_TICKS(DELAY_xAPP));
                    recebeDadosSensores[1] = recebeDados.tensao;
                    comparaValores[0] = recebeDadosSensores[1];
                    xTimerChangePeriod(xTimerUm, pdMS_TO_TICKS(TEMPO_TESTE_DESCARGA),
pdMS_TO_TICKS(10));
                    xTimerStart(xTimerUm, pdMS_TO_TICKS(10));
                    passoAtual++;
                    xTaskNotify(xGerenciaAtuadoresHandle, TESTAR_DESCARGA_70A,
eSetValueWithOverwrite);
            } else if (passoAtual == 1){
                    xQueuePeek(Q_DadosSensores, &recebeDados, pdMS_TO_TICKS(DELAY_xAPP));
                    xTimerChangePeriod(xTimerUm, pdMS_TO_TICKS(TEMPO_TESTE_DESCANSO),
pdMS_TO_TICKS(10));
                    xTimerReset(xTimerUm, 0);
                    recebeDadosSensores[1] = recebeDados.tensao;
                    comparaValores[1] = recebeDadosSensores[1];
                    passoAtual++;
                    xTaskNotify(xGerenciaAtuadoresHandle, DESLIGAR_TEMPERATURA_LIMITE,
eSetValueWithOverwrite);
                    id = 40;
                    xQueueOverwrite(Q_Txt, &id);
            } else if (passoAtual == 2) {
                    xTimerStop(xTimerUm, 0);
                    if (comparaValores[1] >= 9.6) {
                        for (int i = 0; i < 10; i++){
```

```
                        id = 20;
                        xQueueOverwrite(Q_Txt, &id);
                        vTaskDelay(pdMS_TO_TICKS(200));
                        }

                    }
                    else {
                        id = 30;
                        xQueueOverwrite(Q_Txt, &id);
                        vTaskDelay(pdMS_TO_TICKS(200));
                    }
                    xEventGroupClearBits(Evt_Diretoria, BIT10);
                    passoAtual = DESLIGAR;
                    xEventGroupClearBits(Evt_Diretoria, BIT8);
                    xTaskNotify(xAlarmeHandle, ALARME_TESTE_CONCLUIDO,
eSetValueWithOverwrite);
                    xTaskNotify(xGerenciaAtuadoresHandle, DESLIGAR_GERAL,
eSetValueWithOverwrite);
            }
            break;
        case TESTE_DOIS:
            xEventGroupSetBits(Evt_Diretoria, BIT8);
            if (passoAtual == 0){
                    xQueuePeek(Q_DadosSensores, &recebeDados, pdMS_TO_TICKS(DELAY_xAPP));
                    recebeDadosSensores[1] = recebeDados.tensao;
                    comparaValores[0] = recebeDadosSensores[1];
                    xTimerChangePeriod(xTimerDois, pdMS_TO_TICKS(TEMPO_TESTE_DESCARGA),
pdMS_TO_TICKS(10));
                    xTimerStart(xTimerDois, pdMS_TO_TICKS(10));
                    passoAtual++;
                    xTaskNotify(xGerenciaAtuadoresHandle, TESTAR_DESCARGA_70A,
eSetValueWithOverwrite);
            } else if (passoAtual == 1){
                    xQueuePeek(Q_DadosSensores, &recebeDados, pdMS_TO_TICKS(DELAY_xAPP));
                    xTimerChangePeriod(xTimerDois, pdMS_TO_TICKS(TEMPO_TESTE_DESCANSO),
pdMS_TO_TICKS(10));
                    xTimerReset(xTimerDois, 0);
                    recebeDadosSensores[1] = recebeDados.tensao;
                    comparaValores[1] = recebeDadosSensores[1];
                    passoAtual++;
                    xTaskNotify(xGerenciaAtuadoresHandle, DESLIGAR_TEMPERATURA_LIMITE,
eSetValueWithOverwrite);
                    id = 40;
                    xQueueOverwrite(Q_Txt, &id);
            } else if (passoAtual == 2) {
                    xTimerStop(xTimerDois, 0);
                    if (comparaValores[1] >= 10.0) {
                        for (int i = 0; i < 10; i++){
                        id = 20;
                        xQueueOverwrite(Q_Txt, &id);
                        vTaskDelay(pdMS_TO_TICKS(200));
                        }
                    }
                    else {
                        id = 30;
                        xQueueOverwrite(Q_Txt, &id);
                        vTaskDelay(pdMS_TO_TICKS(200));
                    }
                    xEventGroupClearBits(Evt_Diretoria, BIT10);
                    passoAtual = DESLIGAR;
                    xEventGroupClearBits(Evt_Diretoria, BIT8);
                    xTaskNotify(xAlarmeHandle, ALARME_TESTE_CONCLUIDO,
eSetValueWithOverwrite);
                    xTaskNotify(xGerenciaAtuadoresHandle, DESLIGAR_GERAL,
eSetValueWithOverwrite);
            }
```

```
            break;
        case TESTE_TRES:
            xEventGroupSetBits(Evt_Diretoria, BIT8);
            if (passoAtual == 0){
                    xQueuePeek(Q_DadosSensores, &recebeDados, pdMS_TO_TICKS(DELAY_xAPP));
                    recebeDadosSensores[1] = recebeDados.tensao;
                    comparaValores[0] = recebeDadosSensores[1];
                    xTimerChangePeriod(xTimerTres, pdMS_TO_TICKS(TEMPO_TESTE_DESCARGA),
pdMS_TO_TICKS(10));
                    xTimerStart(xTimerTres, pdMS_TO_TICKS(10));
                    passoAtual++;
                    xTaskNotify(xGerenciaAtuadoresHandle, TESTAR_DESCARGA_70A,
eSetValueWithOverwrite);
            } else if (passoAtual == 1){
                    xQueuePeek(Q_DadosSensores, &recebeDados, pdMS_TO_TICKS(DELAY_xAPP));
                    xTimerChangePeriod(xTimerTres, pdMS_TO_TICKS(TEMPO_TESTE_DESCANSO),
pdMS_TO_TICKS(10));
                    xTimerReset(xTimerTres, 0);
                    recebeDadosSensores[1] = recebeDados.tensao;
                    comparaValores[1] = recebeDadosSensores[1];
                    passoAtual++;
                    xTaskNotify(xGerenciaAtuadoresHandle, DESLIGAR_TEMPERATURA_LIMITE,
eSetValueWithOverwrite);
                    id = 40;
                    xQueueOverwrite(Q_Txt, &id);
            } else if (passoAtual == 2) {
                    xTimerStop(xTimerTres, 0);
                    if (comparaValores[1] >= 10.4) {
                        for (int i = 0; i < 10; i++){
                        id = 20;
                        xQueueOverwrite(Q_Txt, &id);
                        vTaskDelay(pdMS_TO_TICKS(200));
                        }
                    }
                    else {
                        id = 30;
                        xQueueOverwrite(Q_Txt, &id);
                        vTaskDelay(pdMS_TO_TICKS(200));
                    }
                    xEventGroupClearBits(Evt_Diretoria, BIT10);
                    passoAtual = DESLIGAR;
                    xEventGroupClearBits(Evt_Diretoria, BIT8);
                    xTaskNotify(xAlarmeHandle, ALARME_TESTE_CONCLUIDO,
eSetValueWithOverwrite);
                    xTaskNotify(xGerenciaAtuadoresHandle, DESLIGAR_GERAL,
eSetValueWithOverwrite);
            }
            break;
        case TESTE_QUATRO:
            xEventGroupSetBits(Evt_Diretoria, BIT8);
            if (passoAtual == 0){
                    xQueuePeek(Q_DadosSensores, &recebeDados, pdMS_TO_TICKS(DELAY_xAPP));
                    recebeDadosSensores[1] = recebeDados.tensao;
                    comparaValores[0] = recebeDadosSensores[1];
                    xTimerChangePeriod(xTimerQuatro, pdMS_TO_TICKS(TEMPO_TESTE_DESCARGA),
pdMS_TO_TICKS(10));
                    xTimerStart(xTimerQuatro, pdMS_TO_TICKS(10));
                    passoAtual++;
                    xTaskNotify(xGerenciaAtuadoresHandle, TESTAR_DESCARGA_70A,
eSetValueWithOverwrite);
            } else if (passoAtual == 1){
                    xQueuePeek(Q_DadosSensores, &recebeDados, pdMS_TO_TICKS(DELAY_xAPP));
                    xTimerChangePeriod(xTimerQuatro, pdMS_TO_TICKS(TEMPO_TESTE_DESCANSO),
pdMS_TO_TICKS(10));
                    xTimerReset(xTimerQuatro, 0);
                    recebeDadosSensores[1] = recebeDados.tensao;
```

```
                    comparaValores[1] = recebeDadosSensores[1];
                    passoAtual++;
                    xTaskNotify(xGerenciaAtuadoresHandle, DESLIGAR_TEMPERATURA_LIMITE,
eSetValueWithOverwrite);
                    id = 40;
                    xQueueOverwrite(Q_Txt, &id);
            } else if (passoAtual == 2) {
                    xTimerStop(xTimerQuatro, 0);
                    if (comparaValores[1] >= 10.8) {
                        for (int i = 0; i < 10; i++){
                        id = 20;
                        xQueueOverwrite(Q_Txt, &id);
                        vTaskDelay(pdMS_TO_TICKS(200));
                        }
                    }
                    else {
                        id = 30;
                        xQueueOverwrite(Q_Txt, &id);
                        vTaskDelay(pdMS_TO_TICKS(200));
                    }
                    xEventGroupClearBits(Evt_Diretoria, BIT10);
                    passoAtual = DESLIGAR;
                    xEventGroupClearBits(Evt_Diretoria, BIT8);
                    xTaskNotify(xAlarmeHandle, ALARME_TESTE_CONCLUIDO,
eSetValueWithOverwrite);
                    xTaskNotify(xGerenciaAtuadoresHandle, DESLIGAR_GERAL,
eSetValueWithOverwrite);
            }
            break;
        case TESTE_CINCO:
            xEventGroupSetBits(Evt_Diretoria, BIT8);
            if (passoAtual == 0){
                    xQueuePeek(Q_DadosSensores, &recebeDados, pdMS_TO_TICKS(DELAY_xAPP));
                    recebeDadosSensores[1] = recebeDados.tensao;
                    comparaValores[0] = recebeDadosSensores[1];
                    xTimerChangePeriod(xTimerCinco, pdMS_TO_TICKS(TEMPO_TESTE_DESCARGA),
pdMS_TO_TICKS(10));
                    xTimerStart(xTimerCinco, pdMS_TO_TICKS(10));
                    passoAtual++;
                    xTaskNotify(xGerenciaAtuadoresHandle, TESTAR_DESCARGA_70A,
eSetValueWithOverwrite);
            } else if (passoAtual == 1){
                    xQueuePeek(Q_DadosSensores, &recebeDados, pdMS_TO_TICKS(DELAY_xAPP));
                    xTimerChangePeriod(xTimerCinco, pdMS_TO_TICKS(TEMPO_TESTE_DESCANSO),
pdMS_TO_TICKS(10));
                    xTimerReset(xTimerCinco, 0);
                    recebeDadosSensores[1] = recebeDados.tensao;
                    comparaValores[1] = recebeDadosSensores[1];
                    passoAtual++;
                    xTaskNotify(xGerenciaAtuadoresHandle, DESLIGAR_TEMPERATURA_LIMITE,
eSetValueWithOverwrite);
                    id = 40;
                    xQueueOverwrite(Q_Txt, &id);
            } else if (passoAtual == 2) {
                    xTimerStop(xTimerCinco, 0);
                    if (comparaValores[1] >= 11.2) {
                        for (int i = 0; i < 10; i++){
                        id = 20;
                        xQueueOverwrite(Q_Txt, &id);
                        vTaskDelay(pdMS_TO_TICKS(200));
                        }
                    }
                    else {
                        id = 30;
                        xQueueOverwrite(Q_Txt, &id);
                        vTaskDelay(pdMS_TO_TICKS(200));
```

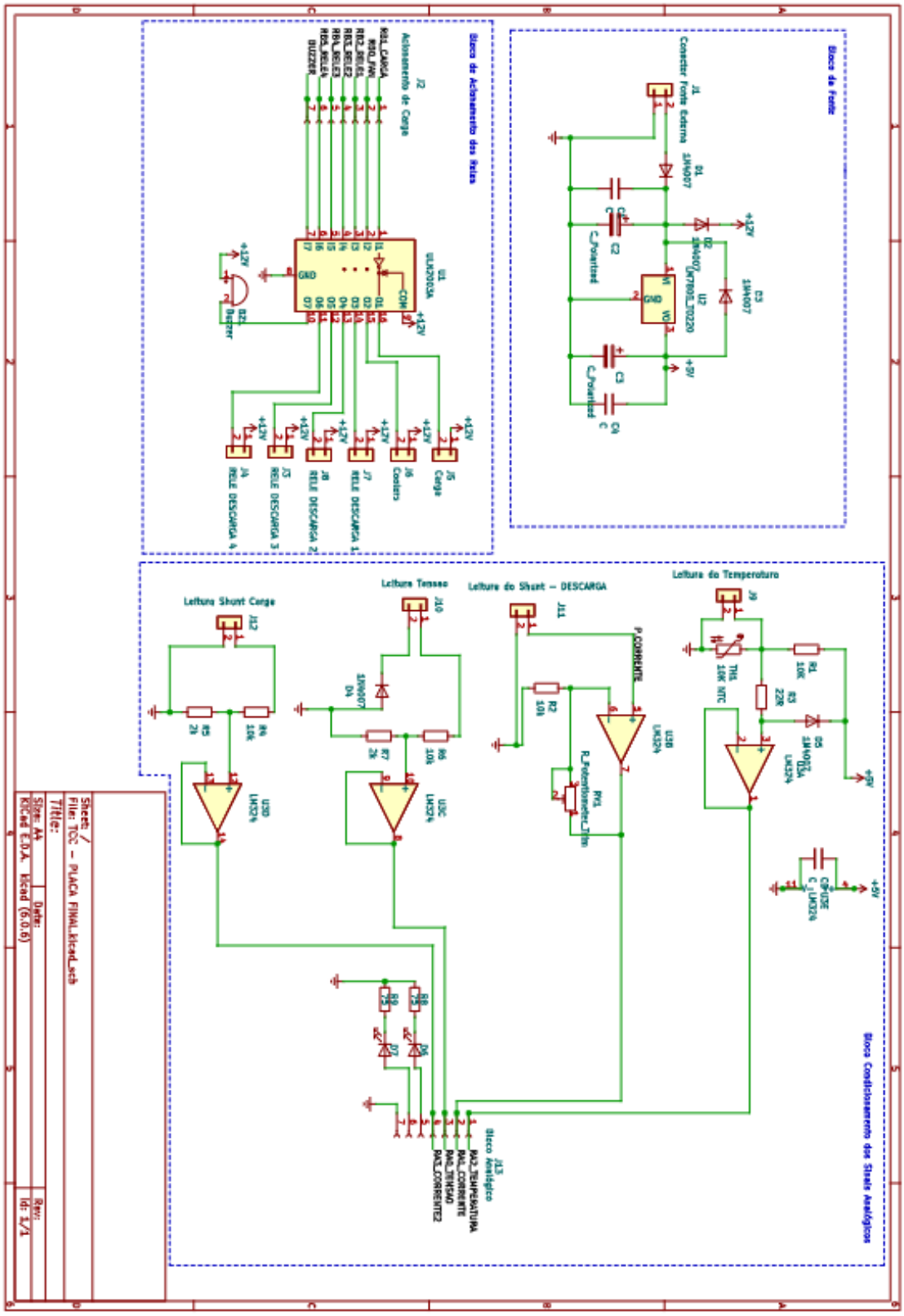
```
                    }
                    xEventGroupClearBits(Evt_Diretoria, BIT10);
                    passoAtual = DESLIGAR;
                    xEventGroupClearBits(Evt_Diretoria, BIT8);
                    xTaskNotify(xAlarmeHandle, ALARME_TESTE_CONCLUIDO,
eSetValueWithOverwrite);
                    xTaskNotify(xGerenciaAtuadoresHandle, DESLIGAR_GERAL,
eSetValueWithOverwrite);
            }
            break;
        case TESTE_SEIS:
            xEventGroupSetBits(Evt_Diretoria, BIT8);
            if (passoAtual == 0){
                    xQueuePeek(Q_DadosSensores, &recebeDados, pdMS_TO_TICKS(DELAY_xAPP));
                    recebeDadosSensores[1] = recebeDados.tensao;
                    comparaValores[0] = recebeDadosSensores[1];
                    xTimerChangePeriod(xTimerSeis, pdMS_TO_TICKS(TEMPO_TESTE_DESCARGA),
pdMS_TO_TICKS(10));
                    xTimerStart(xTimerSeis, pdMS_TO_TICKS(10));
                    passoAtual++;
                    xTaskNotify(xGerenciaAtuadoresHandle, TESTAR_DESCARGA_140A,
eSetValueWithOverwrite);
            } else if (passoAtual == 1){
                    xQueuePeek(Q_DadosSensores, &recebeDados, pdMS_TO_TICKS(DELAY_xAPP));
                    xTimerChangePeriod(xTimerSeis, pdMS_TO_TICKS(TEMPO_TESTE_DESCANSO),
pdMS_TO_TICKS(10));
                    xTimerReset(xTimerSeis, 0);
                    recebeDadosSensores[1] = recebeDados.tensao;
                    comparaValores[1] = recebeDadosSensores[1];
                    passoAtual++;
                    xTaskNotify(xGerenciaAtuadoresHandle, DESLIGAR_TEMPERATURA_LIMITE,
eSetValueWithOverwrite);
                    id = 40;
                    xQueueOverwrite(Q_Txt, &id);
            } else if (passoAtual == 2) {
                    xTimerStop(xTimerSeis, 0);
                    if (comparaValores[1] >= 10.2) {
                        for (int i = 0; i < 10; i++){
                        id = 20;
                        xQueueOverwrite(Q_Txt, &id);
                        vTaskDelay(pdMS_TO_TICKS(200));
                        }
                    }
                    else {
                        id = 30;
                        xQueueOverwrite(Q_Txt, &id);
                        vTaskDelay(pdMS_TO_TICKS(200));
                    }
                    xEventGroupClearBits(Evt_Diretoria, BIT10);
                    passoAtual = DESLIGAR;
                    xEventGroupClearBits(Evt_Diretoria, BIT8);
                    xTaskNotify(xAlarmeHandle, ALARME_TESTE_CONCLUIDO,
eSetValueWithOverwrite);
                    xTaskNotify(xGerenciaAtuadoresHandle, DESLIGAR_GERAL,
eSetValueWithOverwrite);
            }
            break;
        case TESTE_SETE:
            xEventGroupSetBits(Evt_Diretoria, BIT8);
            if (passoAtual == 0){
                    xQueuePeek(Q_DadosSensores, &recebeDados, pdMS_TO_TICKS(DELAY_xAPP));
                    recebeDadosSensores[1] = recebeDados.tensao;
                    comparaValores[0] = recebeDadosSensores[1];
                    xTimerChangePeriod(xTimerSete, pdMS_TO_TICKS(TEMPO_TESTE_DESCARGA),
pdMS_TO_TICKS(10));
                    xTimerStart(xTimerSete, pdMS_TO_TICKS(10));
```

```
                    passoAtual++;
                    xTaskNotify(xGerenciaAtuadoresHandle, TESTAR_DESCARGA_140A,
eSetValueWithOverwrite);
            } else if (passoAtual == 1){
                    xQueuePeek(Q_DadosSensores, &recebeDados, pdMS_TO_TICKS(DELAY_xAPP));
                    xTimerChangePeriod(xTimerSete, pdMS_TO_TICKS(TEMPO_TESTE_DESCANSO),
pdMS_TO_TICKS(10));
                    xTimerReset(xTimerSete, 0);
                    recebeDadosSensores[1] = recebeDados.tensao;
                    comparaValores[1] = recebeDadosSensores[1];
                    passoAtual++;
                    xTaskNotify(xGerenciaAtuadoresHandle, DESLIGAR_TEMPERATURA_LIMITE,
eSetValueWithOverwrite);
                    id = 40;
                    xQueueOverwrite(Q_Txt, &id);
            } else if (passoAtual == 2) {
                    xTimerStop(xTimerSete, 0);
                    if (comparaValores[1] >= 10.4) {
                        for (int i = 0; i < 10; i++){
                        id = 20;
                        xQueueOverwrite(Q_Txt, &id);
                        vTaskDelay(pdMS_TO_TICKS(200));
                        }
                    }
                    else {
                        id = 30;
                        xQueueOverwrite(Q_Txt, &id);
                        vTaskDelay(pdMS_TO_TICKS(200));
                    }
                    xEventGroupClearBits(Evt_Diretoria, BIT10);
                    passoAtual = DESLIGAR;
                    xEventGroupClearBits(Evt_Diretoria, BIT8);
                    xTaskNotify(xAlarmeHandle, ALARME_TESTE_CONCLUIDO,
eSetValueWithOverwrite);
                    xTaskNotify(xGerenciaAtuadores, DESLIGAR_GERAL,
eSetValueWithOverwrite);
            }
            break;
        case TESTE_OITO:
            xEventGroupSetBits(Evt_Diretoria, BIT8);
            if (passoAtual == 0){
                    xQueuePeek(Q_DadosSensores, &recebeDados, pdMS_TO_TICKS(DELAY_xAPP));
                    recebeDadosSensores[1] = recebeDados.tensao;
                    comparaValores[0] = recebeDadosSensores[1];
                    xTimerChangePeriod(xTimerOito, pdMS_TO_TICKS(TEMPO_TESTE_DESCARGA),
pdMS_TO_TICKS(10));
                    xTimerStart(xTimerOito, pdMS_TO_TICKS(10));
                    passoAtual++;
                    xTaskNotify(xGerenciaAtuadoresHandle, TESTAR_DESCARGA_140A,
eSetValueWithOverwrite);
            } else if (passoAtual == 1){
                    xQueuePeek(Q_DadosSensores, &recebeDados, pdMS_TO_TICKS(DELAY_xAPP));
                    xTimerChangePeriod(xTimerOito, pdMS_TO_TICKS(TEMPO_TESTE_DESCANSO),
pdMS_TO_TICKS(10));
                    xTimerReset(xTimerOito, 0);
                    recebeDadosSensores[1] = recebeDados.tensao;
                    comparaValores[1] = recebeDadosSensores[1];
                    passoAtual++;
                    xTaskNotify(xGerenciaAtuadoresHandle, DESLIGAR_TEMPERATURA_LIMITE,
eSetValueWithOverwrite);
                    id = 40;
                    xQueueOverwrite(Q_Txt, &id);
            } else if (passoAtual == 2) {
                    xTimerStop(xTimerOito, 0);
                    if (comparaValores[1] >= 10.6) {
                        for (int i = 0; i < 10; i++){
```

```
                        id = 20;
                        xQueueOverwrite(Q_Txt, &id);
                        vTaskDelay(pdMS_TO_TICKS(200));
                        }
                    }
                    else {
                        id = 30;
                        xQueueOverwrite(Q_Txt, &id);
                        vTaskDelay(pdMS_TO_TICKS(200));
                    }
                    xEventGroupClearBits(Evt_Diretoria, BIT10);
                    passoAtual = DESLIGAR;
                    xEventGroupClearBits(Evt_Diretoria, BIT8);
                    xTaskNotify(xAlarmeHandle, ALARME_TESTE_CONCLUIDO,
eSetValueWithOverwrite);
                    xTaskNotify(xGerenciaAtuadoresHandle, DESLIGAR_GERAL,
eSetValueWithOverwrite);
            }
            break;
        }
        //printf("Valor 1: %f, Valor 2: %f\n", comparaValores[0], comparaValores[1]);
    }
}

/**
*
*  @brief Função que realiza o batimento de coração do sistema.
*  Essa função alterna o estado lógico do pino GPIO_NUM_2 do dispositivo em intervalos
regulares
*  de 1 segundo, permitindo verificar se o sistema está funcionando corretamente.
*  É executada em loop infinito dentro da tarefa alocada.
*
*  @param arg Ponteiro void para os argumentos da tarefa
*/
void xHearthBit(void *arg){

    static bool flag;

    while(true){
        gpio_set_level(GPIO_NUM_2, flag);
        flag = !flag;
        vTaskDelay(pdMS_TO_TICKS(1000));
    }
}

void xAlarme(void *arg){

    uint32_t CMD;
    TickType_t xDelayConfirma = pdMS_TO_TICKS(200); // tempo de acionamento do alarme de
confirmação
    TickType_t xDelayFalha = pdMS_TO_TICKS(2000); // tempo de acionamento do alarme de
falha
    TickType_t xDelayTesteConcluido = pdMS_TO_TICKS(10000); // tempo de acionamento do
alarme de teste concluído

    while(true){

        CMD = ulTaskNotifyTake(pdTRUE, (pdMS_TO_TICKS(DELAY_xAPP))); // Aguarda o comando

        switch(CMD){
            case ALARME_CONFIRMA:
                gpio_set_level(RELE_QUATRO, 1); // Liga o relé de confirmação
                vTaskDelay(xDelayConfirma); // Aguarda o tempo de acionamento do alarme de
confirmação
                gpio_set_level(RELE_QUATRO, 0); // Desliga o relé de confirmação
```

```
                vTaskDelay(xDelayConfirma); // Aguarda o tempo de acionamento do alarme de
confirmação
                gpio_set_level(RELE_QUATRO, 1); // Liga o relé de confirmação
                vTaskDelay(xDelayConfirma); // Aguarda o tempo de acionamento do alarme de
confirmação
                gpio_set_level(RELE_QUATRO, 0); // Desliga o relé de confirmação
                vTaskDelay(xDelayConfirma); // Aguarda o tempo de acionamento do alarme de
confirmação
                gpio_set_level(RELE_QUATRO, 1); // Liga o relé de confirmação
                vTaskDelay(xDelayConfirma); // Aguarda o tempo de acionamento do alarme de
confirmação
                gpio_set_level(RELE_QUATRO, 0); // Desliga o relé de confirmação
                vTaskDelay(xDelayConfirma); // Aguarda o tempo de acionamento do alarme de
confirmação
                break;
            case ALARME_FALHA:
                gpio_set_level(RELE_QUATRO, 1); // Liga o relé de falha
                vTaskDelay(xDelayFalha); // Aguarda o tempo de acionamento do alarme de
falha
                gpio_set_level(RELE_QUATRO, 0); // Desliga o relé de falha
                break;
            case ALARME_TESTE_CONCLUIDO:
                for(int i = 0; i < 10; i++){
                    gpio_set_level(RELE_QUATRO, 1);   // Liga o relé de teste concluído
                    vTaskDelay(xDelayConfirma); // Aguarda o tempo de acionamento do alarme
de teste concluído
                    gpio_set_level(RELE_QUATRO, 0);   // Desliga o relé de teste concluído
                    vTaskDelay(xDelayConfirma); // Aguarda o tempo de acionamento do alarme
de teste concluído
                    gpio_set_level(RELE_QUATRO, 1);   // Liga o relé de teste concluído
                    vTaskDelay(xDelayConfirma); // Aguarda o tempo de acionamento do alarme
de teste concluído
                    gpio_set_level(RELE_QUATRO, 0);   // Desliga o relé de teste concluído
                    vTaskDelay(xDelayConfirma); // Aguarda o tempo de acionamento do alarme
de teste concluído
                    gpio_set_level(RELE_QUATRO, 1);   // Liga o relé de teste concluído
                    vTaskDelay(xDelayConfirma); // Aguarda o tempo de acionamento do alarme
de teste concluído
                    gpio_set_level(RELE_QUATRO, 0);   // Desliga o relé de teste concluído
                    vTaskDelay(xDelayConfirma); // Aguarda o tempo de acionamento do alarme
de teste concluído
                    gpio_set_level(RELE_QUATRO, 1);   // Liga o relé de teste concluído
                    vTaskDelay(xDelayConfirma); // Aguarda o tempo de acionamento do alarme
de teste concluído
                    gpio_set_level(RELE_QUATRO, 0);   // Desliga o relé de teste concluído
                    vTaskDelay(xDelayTesteConcluido); // Aguarda o tempo de acionamento do
alarme de teste concluído
                }
                break;
        }

    }
}

#endif // TASKS_C
```

8 Apêndice B: Esquema elétrico da placa de condicionamento de sinais



9 Apêndice C: Programação Labview