TRABALHO DE CONCLUSÃO DE CURSO

Caixa Preta para Automóveis

Cristiano Carvalho Montenegro

Brasília, Setembro de 2022

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia

TRABALHO DE CONCLUSÃO DE CURSO

Caixa Preta para Automóveis

Cristiano Carvalho Montenegro

Trabalho de Conclusão de Curso submetido ao Departamento de Engenharia Elétrica como requisito parcial para obtenção do grau de Graduando em Engenharia de Redes de Comunicação

Banca Examinadora

Prof. Dr. Ricardo Zelenovsky, ENE/UnB _____
Orientador

Prof. Dr. Alexandre Ricardo Soares Romariz, ENE/UnB _____
Examinador Interno

Porf. Dr. Eduardo Peixoto Fernandes da Silva, ENE/UnB _____
Examinador Interno

FICHA CATALOGRÁFICA

MONTENEGRO, CRISTIANO CARVALHO
Caixa Preta para Automóveis [Distrito Federal] 2022.
xvi, 68 p., 210 x 297 mm (ENE/FT/UnB, Graduando em Engenharia de Redes de Comunicação, Engenharia de Redes de Comunicação, 2022).
Trabalho de Conclusão de Curso - Universidade de Brasília, Faculdade de Tecnologia.
Departamento de Engenharia Elétrica
1. Caixa-preta
2. Automóvel
3. Acidente de trânsito
4. Perícia
I. ENE/FT/UnB

REFERÊNCIA BIBLIOGRÁFICA

MONTENEGRO, C. C (2022). *Caixa Preta para Automóveis*. Trabalho de Conclusão de Curso, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 68 p.

CESSÃO DE DIREITOS

AUTOR 1: Cristiano Carvalho Montenegro
TÍTULO: Caixa Preta para Automóveis.
GRAU: Graduando em Engenharia de Redes de Comunicação ANO: 2022

Cristiano Carvalho Montenegro
Depto. de Engenharia Elétrica (ENE) - FT
Universidade de Brasilia
Campus Darcy Ribeiro
CEP 70919-970 - Brasília - DF - Brasil

Dedicatória

Dedico este trabalho aos meus pais.

Cristiano Carvalho Montenegro

Agradecimentos

Meu agradecimento vai principalmente para a minha mãe, Margarida Silva Carvalho, que esteve sempre me motivando, me encorajando e jamais deixou de acreditar em mim. Esse apoio foi fundamental para que eu percorresse todas as etapas da graduação e também deste trabalho de conclusão de curso.

Sou grato também ao suporte, atenção e conhecimentos adquiridos com o professor Ricardo Zelenovsky, que por sua competência e disponibilidade a ajudar desempenhou um ótimo papel como orientador, tornando a experiência de desenvolver este trabalho algo bastante enriquecedor.

Cristiano Carvalho Montenegro

RESUMO

O intuito deste projeto é programar e testar um dispositivo projetado para carros que é capaz de armazenar em tempo real leituras de GPS, magnetômetro, acelerômetro e giroscópio, de forma que, em caso de acidente, possam ser posteriormente analisadas para melhorar a compreensão do acontecido. Estas leituras permitirão determinar a orientação do veículo, sua localização, bem como toda a movimentação realizada pelo mesmo, garantindo à perícia de trânsito uma boa ferramenta para realizar uma reconstrução mais precisa do que pode ter levado ao acidente. Para se obter esses dados, se fará uso de diversos sensores. Estes, operando em conjunto, irão garantir uma maior precisão na estimativa das informações que se deseja obter, tais como velocidade do carro no instante do acidente e as acelerações e giros a que ele foi submetido. Obtidos os dados, e com a devida análise, pode-se tentar determinar uma provável causa para o acidente e, deste modo, impedir que outros com características semelhantes possam vir a ocorrer.

Para integrar e implementar estas funcionalidades, serão utilizados o microcontrolador Arduino ATMEGA-2560AU (programado na linguagem C++), a unidade de medição inercial MPU-9250, o módulo GPS NEO-6Q-GPS e também uma interface gráfica programada na linguagem *Python*, que irá permitir um controle mais simplificado da Caixa Preta.

O dispositivo desenvolvido encontra-se na sua terceira versão, que foi desenvolvida recentemente. Portanto, um dos objetivos principais deste trabalho é verificar o funcionamento do hardware, de modo a garantir que esteja operando adequadamente e que os resultados obtidos sejam representados conforme se espera. Além disso, também se busca desenvolver e verificar o funcionamento de ferramentas que facilitem a interação do usuário com o dispositivo, permitindo controlá-lo e realizar testes da forma que se deseja.

Palavras-chave: Caixa-preta, automóvel, acidente de trânsito, perícia, sensores

ABSTRACT

This project aims to test and program a device designed for cars that is capable of storing GPS, magnetometer, accelerometer and gyroscope readings in real time so that, in the event of an accident, they can be later analyzed and provide a better understanding of what happened. With these readings, it will be possible to determine the vehicle's orientation, its location and its movement, allowing the forensic analysis to get an accurate reconstruction of what may have led to the accident. In order to obtain this data, sensors like an accelerometer, gyroscope, magnetometer and GPS will be used. This will guarantee greater precision in the estimation of the desired information, such as the speed of the car at the moment of the event and the accelerations and turns to which it was subjected. Once the data is obtained and with due analysis, it will be possible to search for (or get close to) a probable cause of the accident, allowing the prevention of another accident with similar characteristics from occurring.

The device is currently on its third version, which was recently developed. Therefore, one of the main goals of this project is to check its operation, in order to ensure that it is operating properly and that the results are represented as expected. Furthermore, we would like to develop tools to facilitate the user-device interaction and verify if it is working properly. This will allow the user to control and test the device functions in various ways.

Keywords: Black box, vehicles, traffic accident, forensic analysis, sensor

SUMÁRIO

1 INTRODUÇÃO 1
1.1 MOTIVAÇÃO 1
1.2 OBJETIVOS 2
1.3 ORGANIZAÇÃO DO TRABALHO 2

2 FUNDAMENTAÇÃO TEÓRICA 4
2.1 DESCRIÇÃO DO DISPOSITIVO 4
2.2 SENSORES (MPU-9250) 6
2.2.1 ACELERÔMETRO 7
2.2.2 GIROSCÓPIO 8
2.2.3 MAGNETÔMETRO 9
2.2.4 GPS (SISTEMA DE POSICIONAMENTO GLOBAL) 10
2.3 PROTOCOLOS DE COMUNICAÇÃO 11
2.3.1 UART 12
2.3.2 I2C 13
2.3.3 SPI 14
2.3.4 CAN BUS 15
2.4 CONCEITOS TEÓRICOS 17
2.4.1 ÂNGULOS DE EULER E GIMBAL LOCK 17
2.4.2 QUATÉRNIOS E ROTAÇÃO 19

3 DESCRIÇÃO DO CIRCUITO E MODOS DE OPERAÇÃO DO DISPOSITIVO 21
3.1 ALIMENTAÇÃO E CIRCUITO DA CAIXA PRETA 21
3.1.1 CIRCUITO DE PROTEÇÃO 22
3.1.2 ALIMENTAÇÃO APÓS DESLIGAMENTO DO AUTOMÓVEL 23
3.1.3 AUTO DESLIGAMENTO DO ARDUINO 25
3.1.4 MEDIÇÕES E ANÁLISE DO CIRCUITO 27
3.1.5 INTERRUPÇÕES E MONITORAMENTO DAS TENSÕES 29
3.2 MODOS DE OPERAÇÃO 32
3.2.1 MODO TESTE 32
3.2.2 MODO DE OPERAÇÃO 35
3.3 INTERFACE GRÁFICA PARA CONTROLE DA CAIXA PRETA 36

4 ENSAIOS E RESULTADOS DOS TESTES 41
4.1 TESTE DO MODO DE OPERAÇÃO 41
4.2 ENSAIO COM O SUPERCAPACITOR 44

5 Conclusão 47
5.1 Trabalhos Futuros 47

REFERÊNCIAS BIBLIOGRÁFICAS 49

A Coleção de Funções utilizadas pela Caixa Preta 1
A.1 #TWI - Biblioteca para o TWI (I2C) 1
A.2 #LCD - Biblioteca para o LCD (I2C ou TWI) 4
A.3 #SW - Biblioteca para as teclas 10
A.4 #LEDS - Biblioteca para os LEDs 17
A.5 #Timers - Biblioteca para os Timers 20
A.6 #GPS - Biblioteca GPS 23
A.7 #Serial - Biblioteca Serial (Porta 0) para o Arduino 29
A.8 #MPU - Biblioteca MPU 37
A.9 #EEPROM - Biblioteca de acesso à memória EEPROM 41
A.10 #Flash - Biblioteca de acesso à memória Flash 44
A.11 #SRAM-SPI - Biblioteca para a SRAM e SPI 47
A.12 #Strings - Biblioteca para Strings 53

B Códigos-fonte 58
B.1 Código da Interface Gráfica 58
B.1.1 *main.py* 58
B.1.2 *customSerial.py* 62
B.1.3 *GUI.py* 64

LISTA DE FIGURAS

2.1 Primeira versão da Caixa Preta (imagem da esquerda) e segunda versão (imagem da direita), com dimensão 13 x 20 cm 4
2.2 Representação 3D da Caixa Preta 5
2.3 Caixa Preta e seus principais componentes 6
2.4 Esquemático do circuito com dois MPU-9250 para as leituras inerciais 7
2.5 Acelerômetro MEMS 8
2.6 Deslocamento devido à aceleração de Coriolis 9
2.7 Representação do Efeito Hall em um magnetômetro 10
2.8 Módulo GPS da série NEO-6 11
2.9 Diagrama de Blocos UART 12
2.10 Formato do pacote do protocolo UART 13
2.11 Representação de um barramento I2C 13
2.12 Mensagem de uma comunicação I2C 14
2.13 Representação da comunicação SPI em um barramento com 3 escravos 14
2.14 Representação de um barramento CAN 16
2.15 Estados Dominante e Recessivo do barramento CAN 16
2.16 Ângulos de Euler 17
2.17 Problema do *Gimbal Lock* 18
2.18 Ângulos de Euler para a convenção ZYX (Adaptação) 18

3.1 Ensaio com supercapacitor para verificar a duração do comportamento errático do Arduino 22
3.2 Circuito de proteção 23
3.3 Circuito de *backup* da alimentação com o emprego de super capacitores 24
3.4 Circuito que possibilita o auto desligamento do Arduino 25
3.5 Ensaio de energização da Caixa Preta. Em azul, tensão sobre G2 e em vermelho o instante em que a CPU assumiu o controle. 27
3.6 Ensaio do tempo em que C4 consegue sustentar a alimentação da Caixa Preta. Em azul, tensão sobre G2 28
3.7 Ensaio de liga/desliga da Caixa Preta. O ensaio inicia quando a alimentação é ligada. Depois, o programa inicia e faz PG.0 = nível alto (sinal em vermelho), garantindo a alimentação. Decorrido 1 segundo o programa faz PG.0 = nível baixo para desligar a Caixa Preta. 29
3.8 Mesma situação da figura anterior, mas agora a linha vermelha indica a tensão de 5V, que alimenta todos os circuitos da Caixa Preta. 29
3.9 Ilustração dos sinais analógicos monitorados pela Caixa Preta. 30
3.10 Pequeno trecho da máquina de estados que roda dentro da rotina de interrupção e que determina o estado das chaves. 31

3.11 Síntese dos estados para monitoramento das chaves e tensões 31
3.12 Opções do Modo Teste 33
3.13 Visualização de Mapa (*Map View*) 34
3.14 Visualização do Posicionamento dos Satélites (*Satellite Position View*) 34
3.15 Visualização do Sinal dos Satélites (*Satellite Signal View*) 35
3.16 Opções do Modo de Operação 36
3.17 Interface Gráfica do Usuário 37
3.18 Interface no modo de operação 38

4.1 Início do Modo de Operação 41
4.2 Momento em que o botão “SEL” é pressionado 42
4.3 Início da representação dos dados obtidos pelo MPU (delimitador #[m) 42
4.4 Final do modo de Aquisição 43
4.5 LCD antes de pressionar o botão “SEL” 44
4.6 LCD após pressionar o botão “SEL” 44
4.7 Gráfico da carga do super capacitor 45
4.8 Gráfico da descarga do super capacitor enquanto sustenta a Caixa Preta 45

A.1 Esquemático do PCF8574 e LCD 8
A.2 Lógica para detecção das teclas acionadas. 13
A.3 Ilustração do circuito (direita) e ilustração numérica (esquerda) 15
A.4 Diagrama de Estados 26
A.5 Registrador de Modo da memória 23LC1024. 52

LISTA DE TABELAS

3.1 Ciclo de interrupções para monitoramento das chaves e tensões 30
3.2 Resumo das operações realizadas nos diversos estados 32
3.3 Formato dos dados no modo de Aquisição de Dados 36

A.1 Funções - TWI 1
A.2 Funções - LCD 4
A.3 Funções - SW 10
A.4 Disposição da Teclas 11
A.5 Monitoramento das Chaves e Tensões 13
A.6 Gabaritos para configurar ADC (8 bits alinhado pela esquerda, ler apenas ADCH) . 14
A.7 Configuração para os diversos canais 14
A.8 LEDS: Sequência VD – AM – AZ – VM 17
A.9 Funções - LEDS 18
A.10 Funções - TIMERS 20
A.11 Gabarito para configurar os registradores do TC1 21
A.12 Gabaritos para configurar ADC (8 bits alinhados pela esquerda, ler apenas ADCH) 21
A.13 Timer 2 22
A.14 Gabarito para configurar os registradores do TC2 23
A.15 Funções - GPS 23
A.16 Formato das mensagens do GPS 25
A.17 Faixas e níveis de precisão 29
A.18 Funções para a Serial 0 30
A.19 Gabarito dos registradores de configuração da porta serial USART0 36
A.20 Exemplos de *Baudrate* para o Arduino Mega (16 MHz) 36
A.21 Funções - MPU 37
A.22 Funções EEPROM 41
A.23 Faixas de endereços das memórias Flash (2 memórias de 128 KB cada) 44
A.24 Funções - FLASH 45
A.25 Funções - SRAM/SPI 48
A.26 Conjunto de instruções para a memória 23LC1024 52
A.27 Gabaritos para configurar ADC (8 bits alinhado pela esquerda, ler apenas ADCH) . 53
A.28 Funções - STRINGS 54

LISTA DE ALGORITMOS

B.1 Arquivo *main.py*, adaptado de "Simple-Serial-PyQt5"[1] 58
B.2 Arquivo *customSerial.py*, adaptado de "Simple-Serial-PyQt5"[1] 62
B.3 Arquivo *GUI.py*, adaptado de "Simple-Serial-PyQt5"[1] 64

LIST OF ACRONYMS

CXP	*Caixa Preta*
IMU	*Inertial Measurement Unit*
TWI	*Two-Wire Interface*
GPS	*Global Positioning System*
LCD	*Liquid Crystal Display*
SRAM	*Static Random Access Memory*
VCC	*Voltage Common Collector*
EEPROM	*Eletronically-Erasable Programmable Read-Only Memory*
USB	*Universal Serial Bus*
UART	*Universal Asynchronous Transmitter Receiver*
SPI	*Serial peripheral interface*
MEMS	*Microelectricalmechanical Systems*
ADC	*Analog to Digital Converter*

1 INTRODUÇÃO

Dados da Polícia Rodoviária Federal (PRF) apontam que em 2021 foram registradas mais mortes por acidentes de trânsito que em 2020 nas rodovias federais, em que notou-se um aumento de 90 casos. Estes dados do Anuário 2021 da PRF mostram que no ano de 2021, 5.381 pessoas foram vítimas de acidentes apenas em rodovias federais. Um aumento também foi observado nos acidentes de trânsito, pois foram registrados 893 acidentes a mais do que no ano de 2020. Os dados de 2021 indicam cerca de 54 mil vítimas com ferimentos leves e cerca de 17 mil vítimas graves [2].

Embora os acidentes ocorram por variados fatores e causas, a grande maioria poderia ser evitada, ou seja, na maior parte dos casos há um fator causador [3]. Portanto, é importante que se busque mais meios de prevenir tais acidentes ou maneiras de detectar os fatores causadores dos mesmos. Desta forma, ficarão mais claras as ações a serem tomadas para evitar que mais casos venham a ocorrer.

Após ocorrer um acidente, a equipe responsável por fazer a análise pericial possui um papel muito importante, que é justamente o de identificar as causas do mesmo. Para isso, são feitos levantamentos técnicos e científicos no local do acidente e no automóvel envolvido [3], o que exige bastante tempo e precisão na coleta de informações. Tendo isso em vista, qualquer tecnologia que possa auxiliar no processo de coleta de dados é bem vinda, principalmente no que diz respeito a determinar a velocidade e movimentação feita pelo automóvel, o que é essencial para a reconstrução do evento por parte da perícia de trânsito.

1.1 MOTIVAÇÃO

Assim como em aviões, a presença de uma caixa-preta em automóveis seria importante, pois possibilitaria uma maior precisão ao determinar as prováveis causas de um acidente de trânsito. Um dispositivo capaz de realizar leituras em tempo real (que podem ser armazenadas para posterior análise) permitiria uma maior agilidade na análise pericial do veículo envolvido. Deste modo, seriam gastos tempo e recursos menores nas investigações unindo dados levantados no local com informações pré/pós-acidente armazenadas na caixa preta. Além do mais, não haveria uma dependência muito grande do cenário onde o acidente ocorreu, visto que este, por estar sujeito a alterações, deve ser preservado e que as informações mais essenciais se encontrariam na coleta feita pelo dispositivo proposto.

Sabendo-se a velocidade e movimentação do veículo, bem como o trajeto feito pelo mesmo, ficam mais evidentes as causas do acidente, sendo possível determinar se foi causado principalmente pelo motorista ou se seriam necessárias alterações ou ajustes no ambiente onde este ocorreu. Isso possibilitaria verificar alternativas para se evitar futuros acidentes que poderiam vir a ocorrer de forma semelhante. Para tanto, é necessário que haja uma reconstrução bem precisa do evento, permitindo que as alterações a serem feitas (caso necessárias) sejam as mais eficazes possíveis e garantam a segurança dos motoristas e de todos que trafegarem pela rodovia em questão.

1.2 OBJETIVOS

O intuito deste projeto é o teste e programação de um dispositivo embarcado em um carro capaz de realizar e armazenar leituras em tempo real que permitirão que se faça uma análise posterior para determinar as possíveis causas de um acidente de trânsito. Neste dispositivo ficarão registradas leituras como a localização e trajeto feito pelo veículo, sua orientação e toda a movimentação feita pelo mesmo, como movimentos de giro, por exemplo. Para se obter esses dados serão utilizados sensores como GPS, acelerômetro, giroscópio e magnetômetro. A operação em conjunto destes sensores irá garantir uma análise mais precisa na tentativa de reconstruir a cena do acidente. Portanto, com o desenvolvimento deste dispositivo busca-se obter uma ferramenta importante para os profissionais envolvidos na análise pericial e, principalmente, para evitar que acidentes semelhantes venham a ocorrer.

Este trabalho é uma continuação dos projetos finais de graduação de Paulo B. Teixeira Neto: "Caixa Preta para carros: proposta para calibração, coleta e fusão de dados" [4], José Luiz G. Nogueira: "Caixa Preta para Carros: Comparação de métodos de estimativa de inclinação usando acelerômetro, giroscópio e magnetômetro" [5], Gabriela da Silva Lopes: "Caixa Preta para Veículos Automotivos" [6] e Matheus Rotta Ribeiro: Projeto e Fabricação de Hardware para Caixa Preta Automotiva e Densímetro [7].

Mais especificamente, pretende-se testar o hardware desenvolvido por Matheus Rotta Ribeiro em seu trabalho para garantir que funcione conforme esperado e para identificar possíveis pontos a serem melhorados. Trata-se da terceira versão da Caixa Preta, então serão analisados os resultados dos testes para garantir o bom funcionamento das principais funcionalidades do dispositivo.

1.3 ORGANIZAÇÃO DO TRABALHO

O capítulo 1, que é o capítulo atual, descreve a motivação e os objetivos do trabalho proposto.

O capítulo 2 apresenta a fundamentação teórica, abordando os elementos que compõem o dispositivo, os protocolos de comunicação, além de uma breve base teórica, que é considerada importante para se entender as formas de descrever a movimentação do principal objeto de estudo do trabalho, que é o automóvel.

O capítulo 3 aborda a forma como o circuito do dispositivo está configurado e também os modos de operação que podem ser acessados de diferentes formas. No capítulo 4 é onde se encontram a descrição dos testes realizados e resultados obtidos com as funcionalidades programadas no dispositivo.

O Apêndice A traz um resumo das funções utilizadas no programa, envolvendo não só testes como também funções que são implementadas na aquisição de dados no modo de operação da caixa preta. Este apêndice ficou um pouco extenso, mas se considerou importante detalhar todas as funções para servir como forma de documentação e também como referência para outros alunos que darão prosseguimento ao projeto.

No Apêndice B se encontram os códigos-fonte utilizados para programar a interface gráfica que controla a Caixa-Preta e executa suas funcionalidades, permitindo que o usuário verifique se o funcionamento do dispositivo está conforme esperado.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo será feita a abordagem dos principais recursos utilizados para realizar as medições, além de estruturar uma base teórica de modo a permitir uma melhor interpretação e análise dos dados obtidos.

2.1 DESCRIÇÃO DO DISPOSITIVO

Como já se afirmou, esta é a terceira versão da Caixa Preta. As versões anteriores estão apresentadas na Figura 2.1 abaixo:

Figura 2.1: Primeira versão da Caixa Preta (imagem da esquerda) e segunda versão (imagem da direita), com dimensão 13 x 20 cm

A versão mais recente da Caixa preta foi projetada utilizando uma placa de circuito impresso em FR-4 [7]. Ela tem dimensão 10 x 10 cm e está apresentada na Figura 2.2 abaixo. Comparando-se com a versão do dispositivo utilizado no trabalho desenvolvido por Paulo B. Teixeira Neto [4], por exemplo, se nota uma redução considerável em suas dimensões, sendo, portanto, uma versão mais compacta. Isso facilita bastante a inserção da Caixa Preta nos automóveis.

Uma evolução que também merece destaque é a capacidade de permitir a coleta de dados mesmo após um evento que interrompa a alimentação do equipamento, isto é, uma alimentação de *back-up*. Portanto, optou-se por utilizar supercapacitores para desempenharem esta função. Isso se deve ao fato de uma bateria acabar ocupando muito espaço, indo contra a proposta de ser um dispositivo mais compacto. Além disso, também haveria o risco da bateria em questão se soltar da placa, comprometendo a posterior análise de informações a respeito do acidente [7].

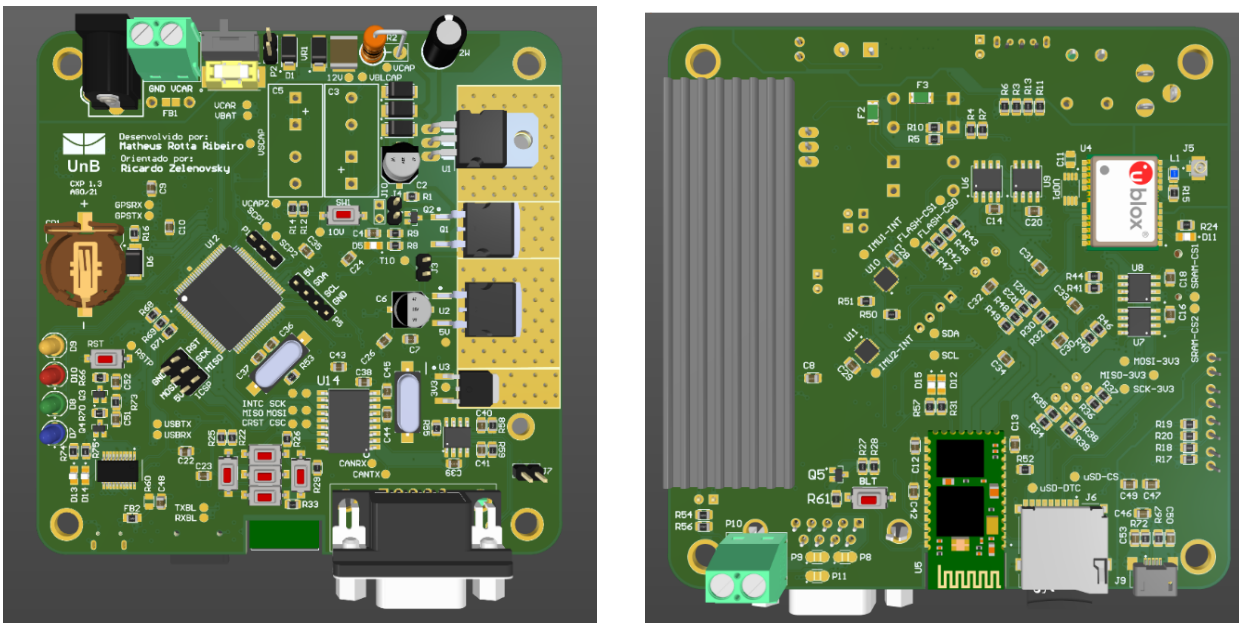

Figura 2.2: Representação 3D da Caixa Preta [7]

Para seu correto funcionamento e para que se possa obter os dados desejados, o dispositivo conta com vários componentes. Entre eles vale destacar:

• Módulo Bluetooth: permite conexão com outros dispositivos para transmitir os dados capturados em tempo real ou em um momento posterior.

• Módulo GPS: permite realizar uma estimativa da localização do veículo bem como identificar o trajeto e velocidade do mesmo.

• Supercapacitores: funcionam como alimentação de *back-up*, caso a alimentação principal seja interrompida por um acidente.

• Botões: podem ser utilizados para selecionar o modo de operação ou outras funcionalidades. Eles são rotulados como CIMA, BAIXO, DIREITA, ESQUERDA e o botão central para selecionar. Existe ainda o botão RST para realizar o *reset*.

• Display LCD: conectado via barramento I2C para facilitar a interação com o usuário.

• Conector microSD: uma alternativa para se coletar os dados armazenados no dispositivo.

• Módulo MPU-9250: responsável por gerar os dados da movimentação do automóvel. É composto por uma unidade de medição inercial com acelerômetro, giroscópio e magnetômetro.

• Microcontrolador Arduíno Mega 2560: responsável por coordenar e integrar todos os componentes e suas funcionalidades. O modelo de microcontrolador é o ATMEGA-2560AU, que é produzido pela Microchip e conta com 256K de memória Flash, 4 kB de EEPROM (*eletronically-erasable programmable read-only memory*) e 8 kB de SRAM (*static random acess memory*), além de 86 pinos GPIO (*general purpose input-output*), e 16 canais A/D de 10 bits [8].

• Interface CAN (*Controller Area Network*): utilizada no circuito para comunicação CAN Bus, através dos chips MCP2515 e TJA1050 [7]

Estes componentes principais e outros secundários podem ser observados na Figura 2.3 abaixo. Lembrando que o *display* LCD não aparece na figura por não ser preso à placa, mas sim ao invólucro do produto.

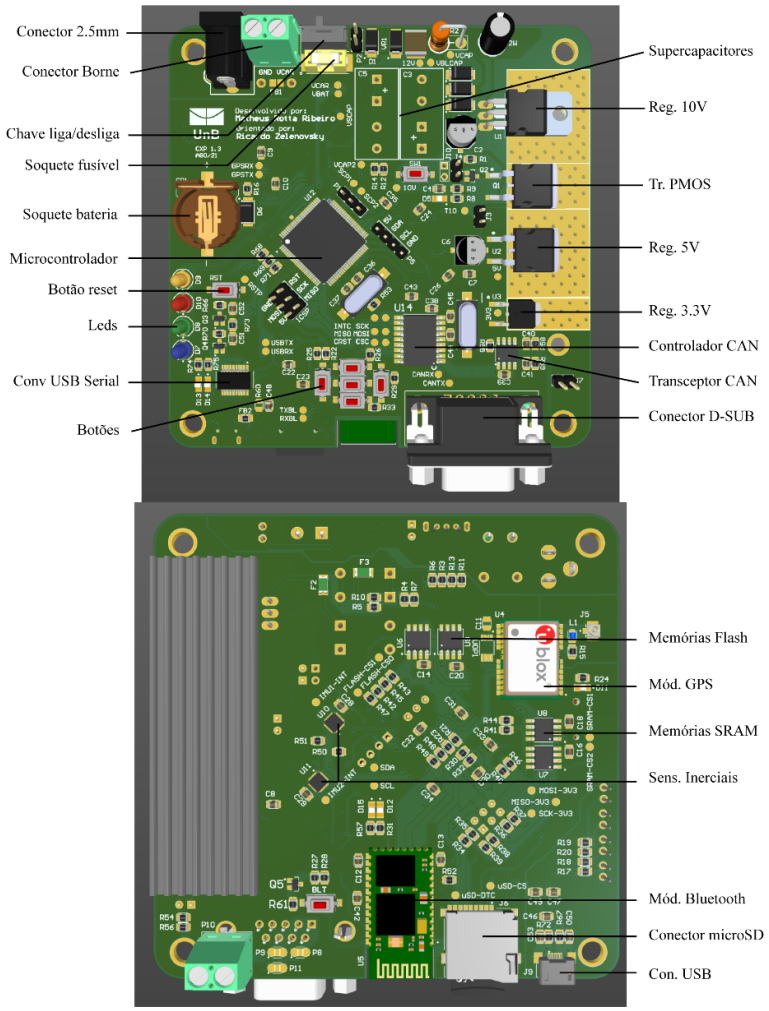

Figura 2.3: Caixa Preta e seus principais componentes [7]

2.2 SENSORES (MPU-9250)

O MPU-9250 é uma unidade de medição inercial com encapsulamento QFN (*Quad Flat No leads*) e dimensões de 3x3x1mm. Ele é caracterizado pela combinação de dois chips, o MPU-6500, composto por acelerômetro e giroscópio (ambos com 3 eixos) e o magnetômetro AK8963, também com 3 eixos [9]. Estes 3 sensores do tipo MEMS (Sistemas Microeletromecânicos), combinados, permitirão obter os dados desejados a respeito de toda a movimentação feita pelo veículo, permitindo identificar as causas e como ocorreu o acidente.

Sua tensão de operação fica na faixa de 2,4 a 3,6V e permite realizar comunicações através dos protocolos I2C e SPI. Pode medir velocidades angulares de ±250 a ±2000 °/s, acelerações de ±2 a ±16g (onde 1g equivale a 9.80665 m/s²) e campos magnéticos de até ±4800 μT (em que T representa a unidade de medida Tesla) [9].

É importante lembrar que, apesar do MPU-9250 possibilitar medições de ±16 g para aceleração e ±2000 graus/s para velocidade angular, ao se utilizar escalas maiores, tem-se como consequência uma sensibilidade menor, o que pode comprometer a precisão na obtenção dos dados. Para contornar isso, foram adicionados dois módulos MPU-9250 na placa, possibilitando leituras com a utilização de escalas maiores com a garantia de que se tenha também uma maior sensibilidade, ou seja, resultados mais satisfatórios e confiáveis para analisar e estimar a dinâmica do evento do acidente [7].

O esquemático do circuito pode ser observado na Figura 2.4 abaixo:

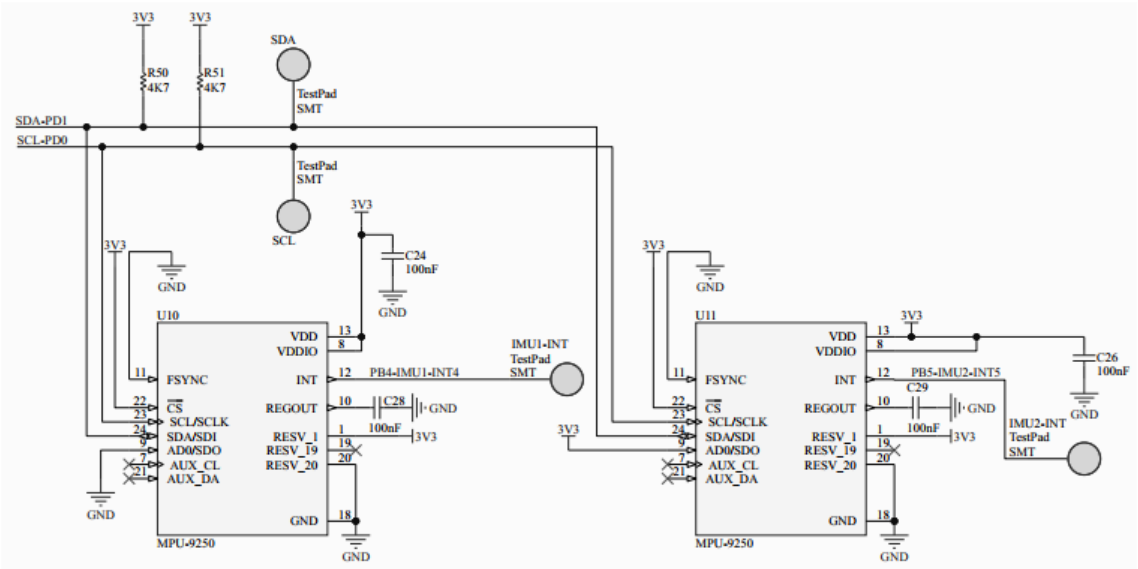

Figura 2.4: Esquemático do circuito com dois MPU-9250 para as leituras inerciais [7]

2.2.1 Acelerômetro

Um acelerômetro é um dispositivo que pode faz medições de aceleração em até três dimensões ortogonais, podendo ser aplicado nos mais variados campos da engenharia e da indústria. Para isso, se utiliza sensores em cada uma dessas dimensões ou eixos. Estes sensores ficam responsáveis por garantir que se converta a aplicação de uma força externa, isto é, um vetor de aceleração em um sinal elétrico. Esta informação é digitalizada em cada eixo por conversores analógico-digitais de 16 bits e é representada com relação ao campo gravitacional da Terra, expressa em g. Sua aplicação permite que se obtenha dados tendo como base atividades sísmicas, vibrações, inclinações, velocidade, aceleração e até mesmo efeitos da gravidade [10].

Com a utilização de componentes eletrônicos e não-eletrônicos, o acelerômetro obtém os dados desejados. Isso é feito com base em uma estrutura fixa e outra móvel, onde, quando ocorre mudança na aceleração, ocorre também mudança na capacitância entre essas duas estruturas. Deste modo, pode-se estimar a aceleração a que a peça está submetida. A vantagem desse tipo de acelerômetro é o baixo consumo de energia e uma boa precisão nas medições [11]. Uma exemplificação do acelerômetro descrito encontra-se na Figura 2.5 abaixo.

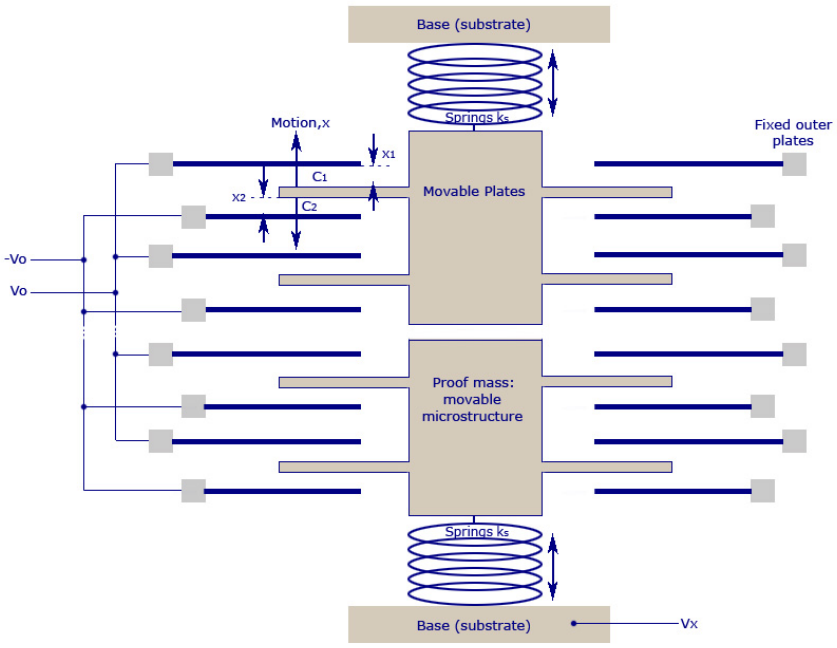

Figura 2.5: Acelerômetro MEMS [12]

2.2.2 Giroscópio

Um giroscópio é um dispositivo com a capacidade de realizar medições de velocidade angular de um determinado objeto. No caso deste projeto, trata-se de um giroscópio em um dispositivo integrado com sensores inerciais, ou seja, apresenta características como tamanho bem pequeno e alta precisão [13]. Esse tipo de característica possibilita sua aplicação em diversas áreas, como a automobilística, por exemplo, que é justamente a área de interesse deste projeto.

Para esta medição, consideram-se os resultados da aceleração de Coriolis, que pode ser compreendida como o aumento da velocidade relativa decorrente da força que surge quando uma massa em movimento de rotação se desloca. No exemplo da Figura 2.5 ocorre um deslocamento de uma massa do centro, isto é, o eixo, para a parte mais externa. A taxa com que ocorre esse aumento de velocidade tangencial pode ser chamado de aceleração de Coriolis [14]. A Figura 2.5 representa bem as forças que surgem a depender da direção do deslocamento da massa (em direção ao centro ou à parte mais externa). Tendo como base essas forças, se pode estimar a taxa de rotação através de vibrações que são detectadas pelo MPU-9250 (composto por um giroscópio MEMS com 3 eixos) levando em consideração um determinado referencial inercial e a movimentação de massas de prova, o que irá ocasionar variação de capacitância. Isso permite a produção de uma tensão proporcional à velocidade angular do sensor, que então será digitalizada pelo conversor ADC de 16 bits do MPU-9250 e possibilitará a leitura desejada dos dados [15].

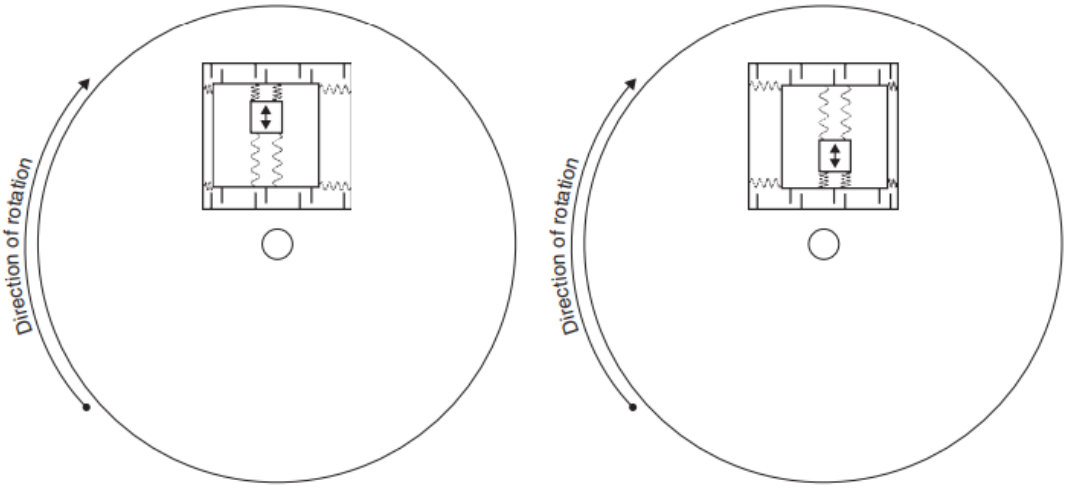

Figura 2.6: Deslocamento devido à aceleração de Coriolis [14]

Com estes dados sobre as rotações, onde quer que ocorram, torna-se possível identificar com maior riqueza de detalhes a movimentação realizada pelo objeto de estudo, que é o automóvel em questão. Nisto podem-se incluir derrapagens, colisões, inclinações e até mesmo capotamento. Isso é possível devido à presença de 3 eixos no giroscópio, o que permite que sejam feitas leituras em três dimensões diferentes.

2.2.3 Magnetômetro

Neste projeto, a função do magnetômetro MEMS é a de fazer a medição e detecção de campos magnéticos que estiverem por perto do dispositivo. O MPU é composto por um magnetômetro de 3 eixos, que coleta dados fundamentando-se no efeito Hall [9].

O efeito Hall tem origem quando uma corrente é introduzida em um condutor ou semicondutor em um campo magnético perpendicular. No fluxo de cargas, quando uma determinada carga passa por um campo magnético, surge uma força chamada força de Lorentz, que gera uma mudança na direção e trajeto percorrido pela carga. Sendo assim, a presença de um campo magnético gera uma interação da carga com uma combinação de forças elétrica e magnética. Da combinação destes efeitos é gerada uma tensão que pode ser medida. É justamente este efeito de se obter uma tensão mensurável através de um campo magnético que é denominado efeito Hall [16]. A Figura 2.7 abaixo representa bem este efeito.

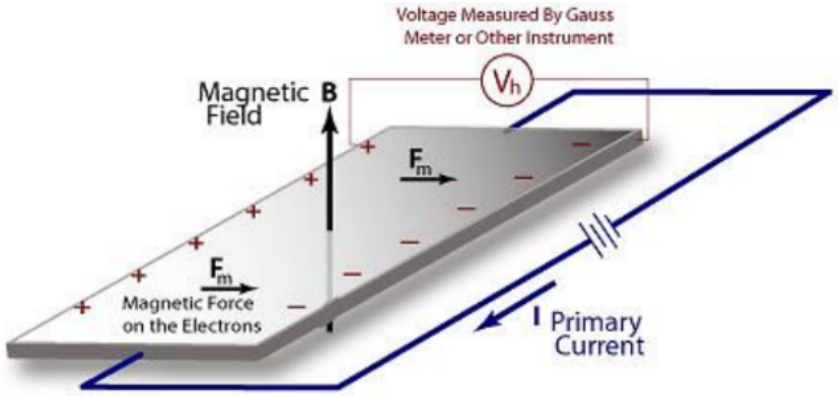

Figura 2.7: Representação do Efeito Hall em um magnetômetro [16]

A medição do campo magnético poder sofrer influência de fatores magnéticos externos, gerando distorções. Esses efeitos podem ser classificados como *hard-iron* e *soft-iron*. Tratando-se da distorção *hard-iron*, esta é produzida por materiais que geram um campo constante e aditivo ao campo magnético da Terra. Deste modo, acaba também gerando valores aditivos nos eixos do magnetômetro. A distorção *soft-iron*, ao contrário da *hard-iron*, não gera um campo magnético aditivo, porém surge de materiais que geram distorções e têm influência sobre um determinado campo magnético. Tendo isso em vista, a distorção produzida por materiais *soft-iron* acaba dependendo da orientação dos mesmos com relação ao sensor e a um campo magnético [17].

2.2.4 GPS (Sistema de Posicionamento Global)

O Módulo GPS é o componente que terá a função de não só auxiliar na localização do veículo pós-acidente, como também de determinar o trajeto que foi feito pelo mesmo antes do evento. No caso desta versão da caixa preta, foi utilizado o modelo NEO-6Q-GPS, da empresa u-blox.

Figura 2.8: Módulo GPS da série NEO-6 [18]

Este modelo conta com um sistema dedicado de aquisição que possibilita buscas em grande escala para encontrar satélites de forma instantânea. Além disso, possui 50 canais que permitem um TTFF (*Time-To-First-Fix*) de 1 segundo [19]. TTFF pode ser compreendido como o tempo necessário para que um dispositivo GPS faça a aquisição dos dados de navegação e sinais de satélite que serão utilizados no cálculo do posicionamento.

Entre as principais características deste dispositivo vale destacar:

• Sistema com 50 canais que opera na banda L1 (1575,42 MHz), C/A Code (um código modulado na banda L1 que contém informações de posicionamento) e SBAS (Sistema de Aumento Baseado em Satélite, utilizado para corrigir os sinais transmitidos): WAAS, EGNOS, MSAS;

• TTFF: 26 segundos em *cold start*, isto é, na primeira vez sendo utilizado ao ligar, juntamente com os processos de inicialização e 1 segundo em *hot start*;

• Interface UART para comunicação serial, SPI e USB;

• Tensão de entrada: 2,7 a 3,6V. [19]

2.3 PROTOCOLOS DE COMUNICAÇÃO

De modo a se obterem as informações capturadas pela Caixa Preta e garantir a transmissão das mesmas em tempo real, faz-se necessária a utilização de protocolos de comunicação. É através deles que se torna possível a comunicação entre os mais variados componentes implementados no dispositivo. Além do mais, estes componentes se comunicam necessariamente através de um dos protocolos que serão descritos a seguir, isto é, UART, I2C, SPI ou CAN Bus.

2.3.1 UART

O UART (*Universal Asynchronous Transmitter Receiver*) é um protocolo bastante utilizado para comunicações seriais *full-duplex*. Trata-se de um protocolo LSI (*Large Scale-Integration*) projetado para realizar comunicações assíncronas entre um dispositivo e outro, ou seja, uma comunicação ponto-a-ponto [20]. Ele se encontra presente em microcontroladores e tem a função de converter dados transmitidos ou recebidos em uma sequência de bits. Isso ocorre através dos seus dois componentes principais: transmissor e receptor, conforme exibido no diagrama de blocos da Figura 2.8. O gerador de *baudrate* é responsável por controlar a velocidade a que o transmissor e receptor transmitem ou recebem os dados. Esta velocidade deve ser a mesma para os dois e vai de 110 bps (bits por segundo) até 230400 bps [20].

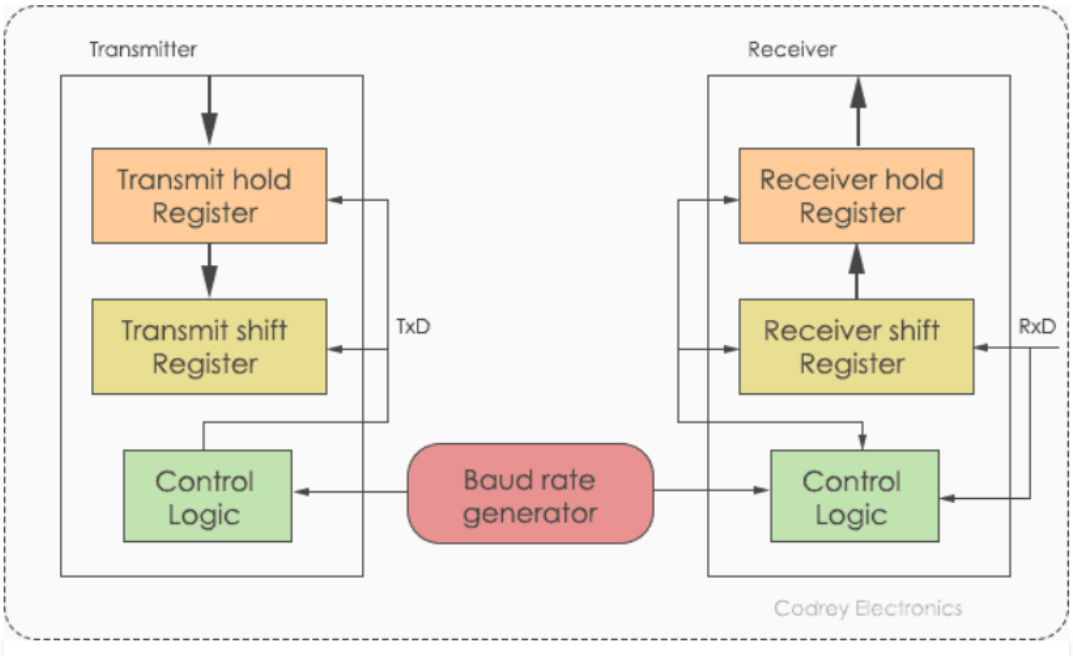

Figura 2.9: Diagrama de Blocos UART [20]

O funcionamento do UART faz uso de bits específicos para garantir uma comunicação serial adequada. De modo geral, o transmissor e receptor utilizam o mesmo *start bit* e *stop bit*, além de parâmetros de tempo para que fiquem em sincronia um com o outro. O *start bit* é responsável por iniciar a comunicação e o *stop bit* encerra a transmissão de dados. Além disso, também há um bit de paridade que é verificado no receptor para identificar o tamanho correto da informação recebida [20].

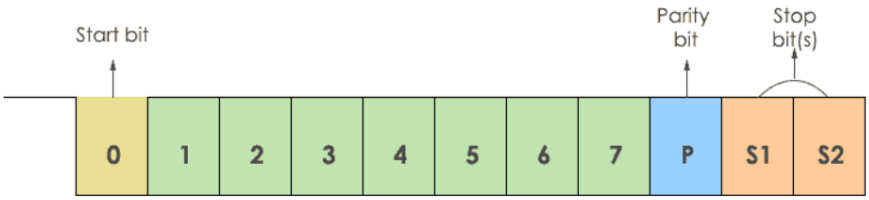

Figura 2.10: Formato do pacote do protocolo UART [20]

Comparado com outros protocolos, o UART acaba não sendo o mais recomendado caso se busque uma comunicação veloz, pois se trata de um protocolo mais simples e que exige menos recursos para ser implementado. No caso deste projeto, o protocolo UART foi utilizado para comunicação no módulo GPS NEO-6Q-0, o módulo bluetooth HC-05, no conversor USB-Serial FT231XS e no transceiver CAN TJA1050 [7].

2.3.2 I2C

O I2C (Inter-integrated circuit) é um protocolo utilizado com o objetivo de permitir múltiplas comunicações entre um "mestre" e vários "escravos", além de permitir múltiplos mestres controlando um ou vários escravos. Essa comunicação serial é feita de forma síncrona e half-duplex.

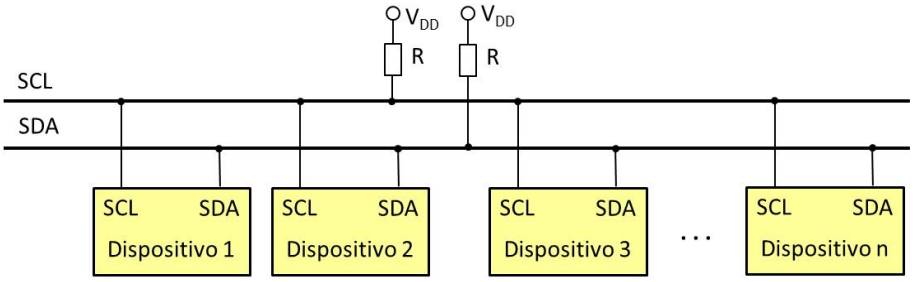

Figura 2.11: Representação de um barramento I2C [21]

Tendo como base a Figura 2.10, vale destacar as linhas utilizadas na transmissão de dados. A linha SDA (*Serial Data*) tem a função de transportar informações do mestre para o escravo ou vice-versa. Já a linha SCL (*Serial Clock*) é a que transporta o sinal de *clock* já que, por se tratar de uma comunicação síncrona, faz-se necessária a utilização da amostragem de bits de um sinal de *clock* entre mestre e escravo para garantir a sincronia no *output* de bits. Este sinal de *clock* é sempre controlado pelo mestre [22].

A implementação deste protocolo requer apenas dois fios e permite alcançar velocidades de transmissão de 100 kbps no *Standard-mode* e até 3.4 Mbps no *High Speed mode* [22]. Sendo assim, sua aplicação é mais direcionada para circuitos integrados que não demandam velocidades muito altas, sendo uma boa opção para projetos mais simples e com baixo custo. No caso deste projeto, o I2C é necessário para realizar a comunicação com o MPU-9250 e a EEPROM 24AA32A, além do *display* LCD [7].

Uma mensagem utilizada na comunicação I2C é composta por um frame de endereço do escravo, um ou dois campos para dados, condições de início e parada, bits que determinam leitura ou escrita, além de bits de ACK/NACK que garantem a recepção da mensagem com integridade após fazer verificações [22].

Figura 2.12: Mensagem de uma comunicação I2C [21]

2.3.3 SPI

O protocolo de comunicação SPI (*Serial Peripheral Interface*) é um tipo de interface bastante utilizado entre microcontroladores e circuitos integrados. Seu funcionamento é síncrono, *full-duplex* e, assim como o I2C, baseia-se em um sistema mestre-escravo. Interfaces SPI podem ter apenas um mestre e um ou múltiplos escravos [23].

A comunicação entre mestre e escravo é comumente feita através de 4 linhas (*4-wire*), conforme representado na Figura 2.13 abaixo.

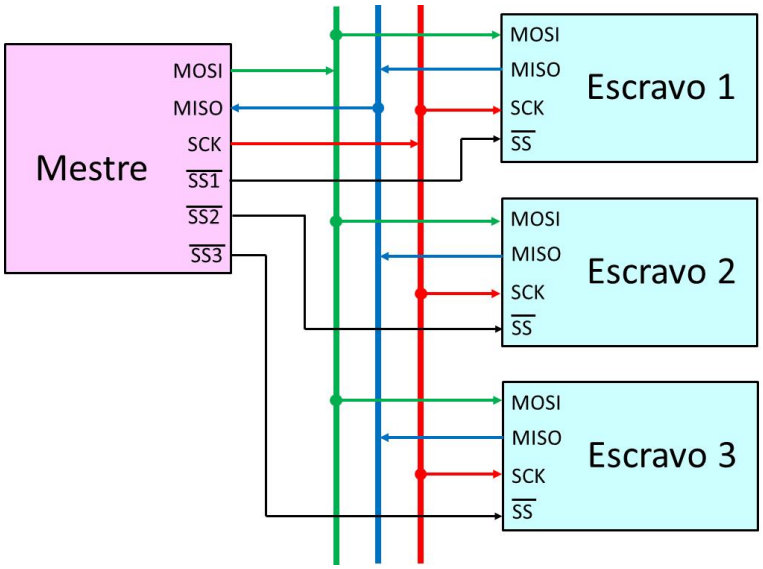

Figura 2.13: Representação da comunicação SPI em um barramento com 3 escravos [21]

A respeito destas 4 linhas que foram mencionadas, MISO (*Master In, Slave Out*) e MOSI (*Master Out, Slave In*) são as linhas responsáveis por realizarem a transferência de dados, sendo que MOSI é para envio de dados do mestre para o escravo e MISO o caminho inverso. Vale lembrar que estas só ocorrem em uma via, isto é, são *simplex* [23]. Adicionalmente, a linha SS (*Slave Select*) ou CS (*Chip Select*) é utilizada pelo meste para selecionar o escravo para o qual os dados serão enviados. No caso da linha SCLK, esta é utilizada pelo mestre para enviar o sinal de *clock*, o que possibilita garantir a sincronia na comunicação.

Comparado com os outros protocolos abordados anteriormente, o SPI apresenta como vantagem uma maior velocidade na transmissão de dados e não tem a necessidade de utilizar bits para início ou parada da transmissão (já que os dados podem ser enviados sem interrupção). Entretanto, não possui uma forma de verificar a integridade das informações recebidas [24].

Para este projeto, o SPI foi aplicado na comunicação com as memórias SRAM 23LC1024, FLASH W25Q64, conector micro SD, com o microcontrolador ATMEGA2560-TH e o controlador CAN MCP 2515.

2.3.4 CAN Bus

O *Controller Area Network* (CAN) é um protocolo de comunicação originalmente desenvolvido para ser aplicado na indústria automotiva. Ele foi desenvolvido em 1986 pela BOSCH, uma empresa alemã [25].

Entre suas principais características merece destaque o fato de ser um protocolo multi-mestre, do tipo *broadcast*, bidirecional e além do mais permite uma taxa de transferência de até 1 megabit por segundo (Mbps). Diferentemente das redes tradicionais, com o envio de grandes blocos de dados ponto-a-ponto, a implementação do CAN Bus faz com que pequenas blocos sejam enviados para todos os nós do sistema (*broadcast*), garantindo uma maior consistência das informações [25].

O padrão CAN é um protocolo definido pela *International Stadardization Organization* (ISO) e faz uso de um barramento diferencial de pares trançados (*two-wire bus*). Suas especificações exigem um meio de comunicação com alta imunidade a interferências elétricas e a capacidade de realizar uma autodiagnose e reparo de erros nos dados, isso possibilitou expandir suas áreas de aplicação para além da área automotiva [25].

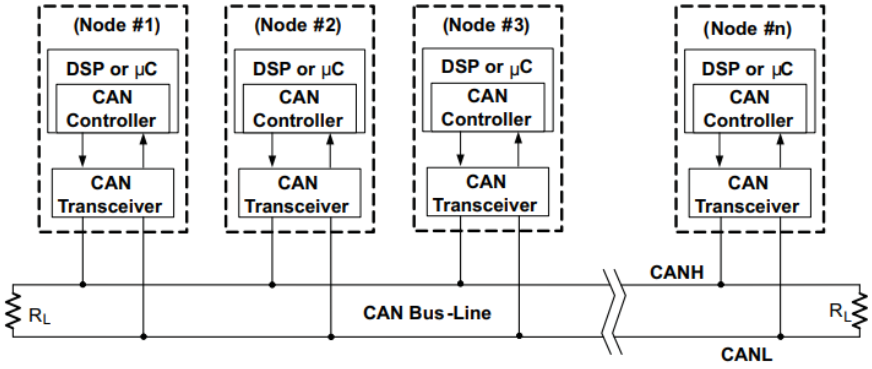

Figura 2.14: Representação de um barramento CAN [25]

Conforme representado na Figura 2.14 acima, o barramento é composto pelas linhas CANH (CAN *High*) e CANL (CAN *Low*), além de resistores *pull-up* para manter as linhas a 2,5 V caso fiquem ociosas. Seu funcionamento se dá em dois estados: um estado "dominante", onde CANH possui tensão maior que CANL (gerando um diferencial de 2V) e um estado "recessivo", onde CANH é menor que CANL.

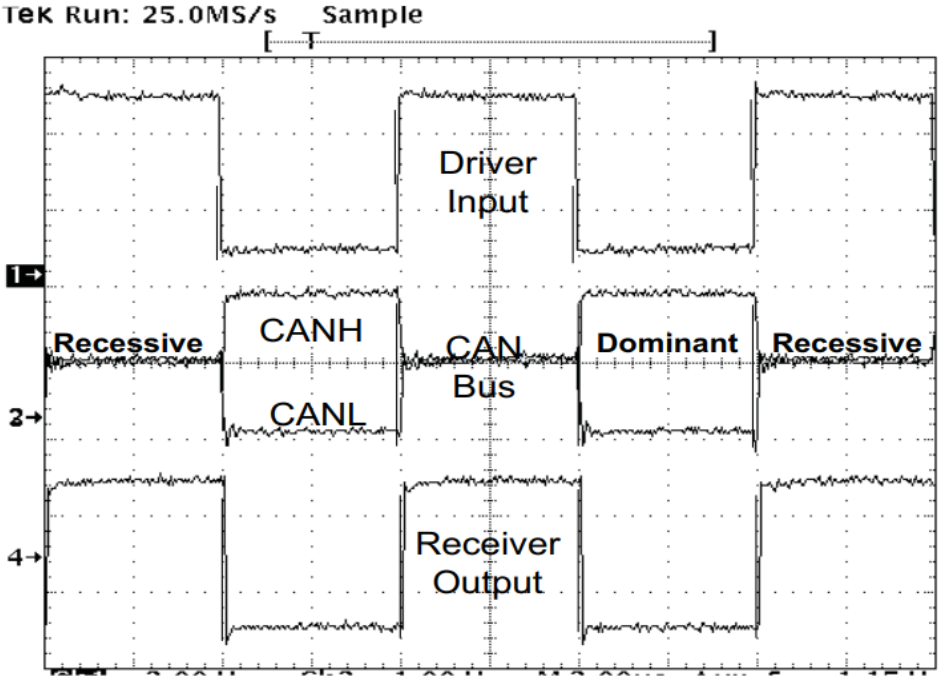

Figura 2.15: Estados Dominante e Recessivo do barramento CAN [25]

Neste projeto o CAN BUS foi utilizado na Caixa Preta nos chips MCP-2515 e TJA1050, responsáveis pela comunicação CAN no projeto. Isso ocorre com a filtragem para reduzir sobrecargas no microcontrolador e com a tradução entre o controlador MCP-2515 e o barramento CAN [7].

2.4 CONCEITOS TEÓRICOS

2.4.1 Ângulos de Euler e Gimbal Lock

Com o intuito de descreverem-se rotações, pode-se ter como base os Ângulos de Euler. Estes foram introduzidos por Leonhard Euler como uma forma de definir rotações em três eixos com três ângulos diferentes. Como pode ser observado na Figura 2.13 abaixo, a rotação em cada eixo (x, y e z) pode ser definida pelos ângulos ϕ, θ e ψ, também conhecidos como *roll*, *pitch* e *yaw* [26].

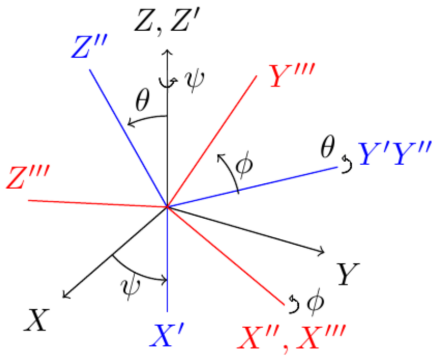

Figura 2.16: Ângulos de Euler [4]

Para tornar melhor a representação destes ângulos, pode-se transferi-los para uma matriz de rotação. Esta matriz é representada da seguinte forma:

$$\boldsymbol{R} = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix} \tag{2.1}$$

Para representar a rotação em cada eixo, essa matriz assume a seguinte forma:

$$\begin{aligned} R_x(\theta) &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{pmatrix} \\ R_y(\theta) &= \begin{pmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ sin(\theta) & 0 & \cos(\theta) \end{pmatrix} \\ R_z(\theta) &= \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \end{aligned} \tag{2.2}$$

Onde Rx, Ry e Rz representam as rotações nos eixos x, y e z, respectivamente. Apesar da possibilidade de se poder descrever rotações, não se pode ignorar um problema conhecido como *Gimbal Lock*. Este problema ocorre quando dois dos três eixos se alinham de forma paralela durante a rotação, o que gera a perda de um dos graus de liberdade, ou seja, em uma rotação conforme representado na Figura 2.17 abaixo:

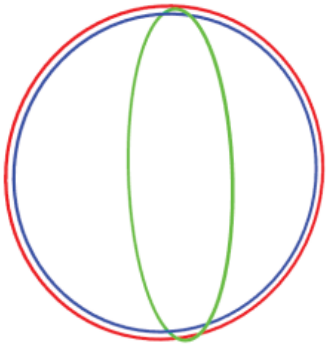

Figura 2.17: Problema do *Gimbal Lock* [13]

Uma rotação pode ter diferentes combinações de rotação de eixos, ou seja, ZYX, XYZ, entre outras. A ordem destas rotações e, principalmente, a primeira rotação, influenciam diretamente o resultado e a leitura final dos dados. Portanto, para evitar esse tipo de problema, deve-se saber inicialmente o tipo de convenção a ser adotada, garantindo que o problema do Gimbal Lock seja evitado e que se possa obter os resultados mais adequados [27]. Uma convenção que é bastante adotada é a convenção ZYX, ou *yaw*, *pitch*, *roll*, que é a convenção adotada neste projeto. Como consequência desta escolha, tem-se a presença do *Gimbal Lock* em $\theta = \pi/2$, o que pode gerar imprecisões nas medidas das rotações.

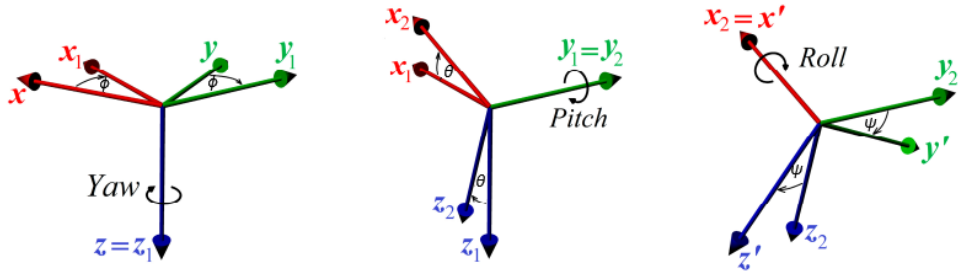

Figura 2.18: Ângulos de Euler para a convenção ZYX (Adaptação) [26]

2.4.2 Quatérnios e Rotação

Tendo em vista as dificuldades que podem surgir ao representar rotações através dos ângulos de Euler, pode-se buscar uma outra alternativa para descrever esse tipo de movimentação. Essa outra alternativa seria uma representação por quatérnios, uma álgebra desenvolvida por Hamilton em 1843 que descreve rotações através de números complexos. Para isso, são necessários quatro valores, onde um componente, isto é, q_0, é real e os outros três são complexos (q_1, q_2 e q_3), conforme a equação 2.3 abaixo [27]:

$$\mathbf{q} = q_0 + q_1 i + q_2 j + q_3 k\,, \tag{2.3}$$

onde os eixos complexos i,j e k atendem à seguinte condição:

$$i^2 = j^2 = k^2 = ijk = -1\,. \tag{2.4}$$

Um quatérnio também pode ser representado por um vetor normalizado, o que pode ser encontrado através da equação 2.5 abaixo:

$$\mathbf{q}_{norm} = \frac{q}{||q||}\,, \tag{2.5}$$

$$||q|| = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2}\,. \tag{2.6}$$

Um quatérnio pode ser representado como vetor da seguinte forma:

$$\mathbf{q} = [q_0\ q_1\ q_2\ q_3]^t. \tag{2.7}$$

Sendo assim, também se pode realizar operações com quatérnios, onde, no caso da adição de dois deles, basta somar cada um dos elementos de cada vetor. Já no caso da multiplicação de dois vetores q_a e q_b, onde q_a é (a_1+b_1i+c_1j+d_1k) e q_b é (a_2+b_2i+c_2j+d_2k), esta pode ser representada pela expressão a seguir:

$$\begin{aligned}(a_1 + b_1 i + c_1 j + d_1 k)(a_2 + b_2 i + c_2 j + d_2 k) = \\ a_1a_2 - b_1b_2 - c_1c_2 - d_1d_2 \\ +(a_1b_2 + b_1a_2 + c_1d_2 - d_1c_2)i \\ +(a_1c_2 - b_1d_2 + c_1a_2 + d_1b_2)j \\ +(a_1d_2 + b_1c_2 - c_1b_2 + d_1a_2)k\end{aligned} \tag{2.8}$$

Vale destacar que o resultado obtido com a multiplicação q_aq_b é diferente do resultado obtido com q_bq_a. No caso da multiplicação de um vetor por um escalar, basta multiplicar cada elemento por este escalar. Para se representar uma rotação de θ graus em torno de um eixo definido por um vetor n = [n_x n_y n_z] se pode utilizar a seguinte equação:

$$\mathbf{q} = [n_x i \; n_y j \; n_z k] \sin(\theta/2) + \cos(\theta/2) \; . \tag{2.9}$$

Comparada com a representação através dos Ângulos de Euler, uma representação com quatérnios acaba sendo mais eficiente pelo fato de não necessitar de três matrizes de rotação, além de evitar o risco do problema do *Gimbal Lock*. Portanto, permite uma coleta e análise de dados mais simples e precisa [27].

3 DESCRIÇÃO DO CIRCUITO E MODOS DE OPERAÇÃO DO DISPOSITIVO

3.1 ALIMENTAÇÃO E CIRCUITO DA CAIXA PRETA

Se tratando do circuito para esta versão da Caixa Preta, projetado pelo aluno Matheus Rotta Ribeiro no trabalho "Projeto e Fabricação de Hardware para Caixa Preta Automotiva e Densímetro"[7], na forma como o circuito foi projetado a alimentação da Caixa Preta é feita através da saída do acendedor de cigarros do automóvel, que fornece uma tensão de 12 V. Para a implementação deste projeto foi utilizado o Arduino Mega, que é alimentado pela entrada VIN. Esta entrada supre o regulador de 5 V que gera a tensão para todo o Arduino. Segundo o site do dispositivo [28], VIN deve estar na faixa de 7 a 12V (com limites máximos de 6 V e 20 V).

Tendo em vista as informações da alimentação do Arduino, pode-se supor que não haveria problemas em utilizar os 12 V com o carro desligado. Porém, quando o carro está em funcionamento, temos o alternador operando e motores elétricos sendo acionados, além do centelhamento para a queima do combustível. Todos estes fatores em conjunto acabam gerando uma grande quantidade de perturbações na alimentação de 12 V, sendo necessária a implementação de um circuito de proteção [29].

Um outro fator, que é também um dos objetivos do projeto, é garantir que o Arduino funcione por pelo menos 5 segundos (tempo para gravar os dados da memória RAM na memória Flash) após o desligamento do carro, ou seja, no caso de um acidente e desligamento do automóvel, por exemplo, a Caixa Preta continuaria obtendo dados por um certo período de tempo. Para isso, pensou-se na possibilidade de utilizar baterias recarregáveis, porém isso demandaria que periodicamente essas baterias fossem trocadas. Portanto, uma solução mais simples é a implementação com um super capacitor [30].

Com o automóvel desligado e as operações finalizadas, não há mais motivo para que o Arduino continue ligado. Sendo assim, há a demanda de um recurso que permita que o Arduino se autodesligue. Entre as vantagens disso se pode destacar: diminuição da amplitude dos ciclos de carga/descarga do super capacitor, já que não será completamente exaurido; impede que o processador opere com alimentação próxima ao seu limite inferior de tensão [29], que não é uma boa condição, conforme será explicado a seguir.

Os processadores usualmente possuem um circuito de "*Brownout Reset*", que força um reset do processador quando a alimentação está abaixo de um determinado nível. Isso evita que ele funcione de forma errática por estar alimentado com uma tensão abaixo do mínimo necessário.

Foi observado que o emprego de um super capacitor pode colocar o Arduino em uma condição errática quando a tensão fornecida pelo super capacitor se aproxima do limite inferior especificado pelo fabricante, o que foi algo inesperado. Quando a tensão fica abaixo do limite inferior, o circuito de *Brownout* coloca o processador em *reset*, fazendo com que o consumo de corrente caia. Com a diminuição no dreno de corrente, a tensão fornecida pelo super capacitor aumenta um pouco, o suficiente para desativar o *Brownout* e repetir o ciclo [29]. A Figura 3.1 abaixo ilustra um ensaio realizado para medir quanto tempo um super capacitor conseguiria sustentar a CPU. Pode-se notar que a CPU ficou num comportamento errático durante 52 segundos.

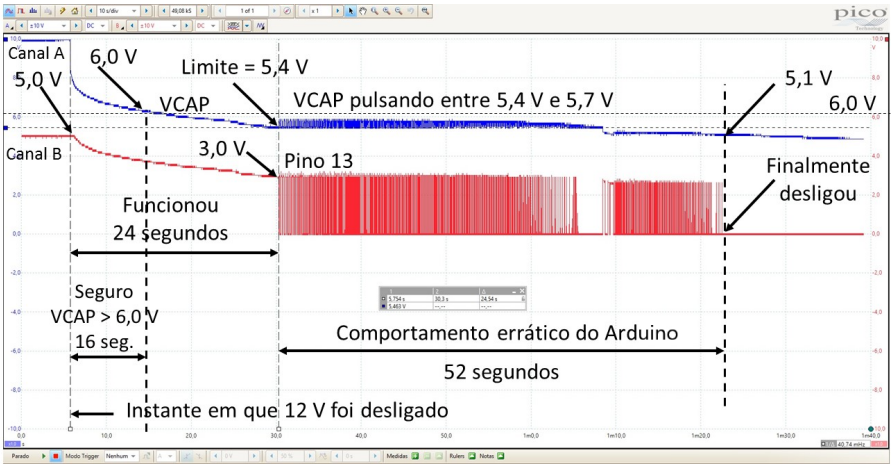

Figura 3.1: Ensaio com supercapacitor para verificar a duração do comportamento errático do Arduino [29]

3.1.1 Circuito de proteção

Para garantir a proteção do circuito dos picos de energia que possam vir a surgir com o carro em funcionamento, foi utilizado o circuito de proteção da Figura 3.2 abaixo:

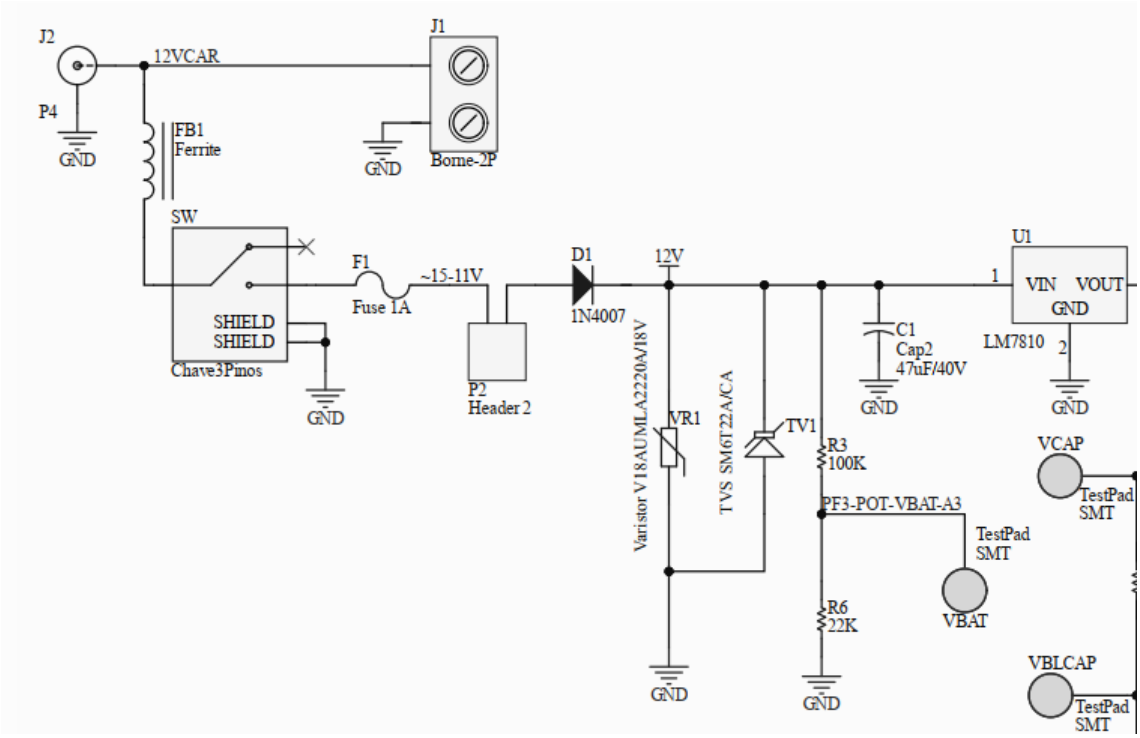

Figura 3.2: Circuito de proteção [7]

Destacam-se duas preocupações: a primeira tem relação com os picos de tensão que podem danificar todo o circuito. Estes são eliminados com um dispositivo TVS (*Transient Voltage Supressor*) [31], que é um supressor de transientes e um varistor. Como os picos de tensão não são tão altos e nem velozes, o tipo mais adequado é o diodo TVS. Esses dois dispositivos receberam os nomes TV1 e VR1.

A segunda preocupação é com o possível surgimento de um pico de tensão negativa. Sendo assim, foi utilizado o diodo retificador D1 (1N4007) em série com a alimentação. Este entra em polarização reversa, isto é, em corte, na presença de picos negativos. Quando conduzindo, esse diodo apresenta uma queda de tensão de 0,7 V, o que permite reduzir os 12 V fornecidos pelo caro e se afastar de forma razoável do limite superior de 12 V do Arduino. O regulador U1 reduz a tensão para 10 V e funciona como uma proteção secundária. Já para o caso de curto-circuito, foi adicionado um fusível em série [29].

3.1.2 Alimentação após desligamento do automóvel

De modo a atender à demanda do funcionamento por um curto período da Caixa Preta após o desligamento do carro, foi utilizado um super capacitor. No mercado nacional não foi possível encontrar um dispositivo com tensão de 12 V, e sim 5,5 V. Então, a solução foi combinar dois destes em série para obter a tensão de 11 V [29]. É importante notar que os super capacitores estão após o regulador de 10 V.

Para garantir a carga do super capacitor enquanto o carro está funcionando e sua atuação imediata quando a energia for cortada, foi montado um circuito com os diodos D2, D3 e D4, conforme representado na Figura 3.3 abaixo.

Ao ligar o carro (com o super capacitor totalmente descarregado), o super capacitor se comporta como um curto-circuito momentâneo. Para evitar esse curto para a terra e a queima do fusível de 2 A, foi colocado um pequeno resistor (R2) em série com o circuito de carga do super capacitor. Tendo como base uma alimentação de 12 V e o limite de corrente de 1A do fusível, se chega a 12 Ω, que é o valor mínimo para o resistor. Como esse valor influencia diretamente no tempo necessário para carregar o super capacitor, se optou por um resistor de 100 Ω [29].

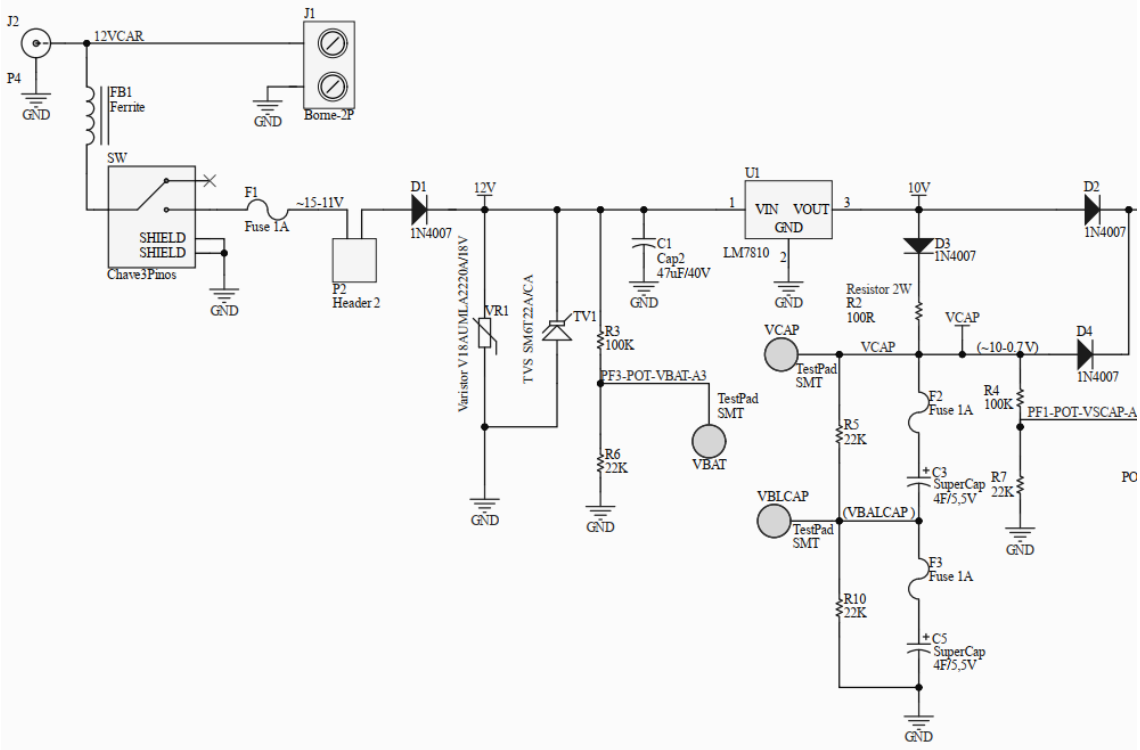

Figura 3.3: Circuito de *backup* da alimentação com o emprego de super capacitores [7]

Como pode ser observado na Figura 3.3 acima, os diodos D3 e D4 garantem o funcionamento adequado do super capacitor, isto é, sua carga e descarga. Quando a fonte de 12 V está ligada, o diodo D3 é polarizado diretamente e garante a carga do super capacitor. No mesmo cenário, o diodo D4 está cortado, pois tanto o anodo quanto o catodo de D4 têm a mesma tensão. Isto impede que o super capacitor forneça energia ao Arduino.

No caso do diodo D3, quando a fonte é desligada, ele passa a ficar reversamente polarizado (entra em corte), ao passo que, no mesmo instante, D4 fica diretamente polarizado e passa a alimentar o Arduino.

3.1.3 Auto Desligamento do Arduino

Em princípio, a Caixa Preta acompanharia a energização do carro. Isto significa que ela liga quando o carro é ligado e desliga quando o motorista desligar a chave. Porém, em caso de acidente, mesmo com a energia do automóvel comprometida, a Caixa Preta precisa continuar a registrar as informações, para depois gravá-las na memória Flash e só então se desligar.

Conclui-se, então, que a Caixa Preta precisa comandar seu próprio desligamento. Os resistores R3 e R6, da Figura 3.4 abaixo, indicam se a energia do carro está ligada e assim permitem que a Caixa Preta decida sobre seu funcionamento. Abaixo está uma descrição completa das condições de liga/desliga:

• Sempre que o carro é ligado, a Caixa Preta é ligada. Isto é possível com os componentes R9 e C4. O capacitor C4 apresenta, momentaneamente, uma tensão alta na porta de Q2, forçando-o para a condução e ligando a Caixa Preta.

• Se a energia do carro é desligada e não houve acidente, esta ausência de energia externa é indicada pelo divisor resistivo R3 e R4 e a Caixa Preta se autodesliga.

• Se aconteceu um acidente, tendo ou não energia externa, a Caixa Preta completa a aquisição dos dados, grava-os na memória Flash e depois se autodesliga.

É importante mencionar que o regulador U2 recebe a tensão VIN e entrega 5 V para alimentar os principais circuitos da CXP. Existe também o regulador U3, que gera 3,3 V para alguns circuitos especiais.

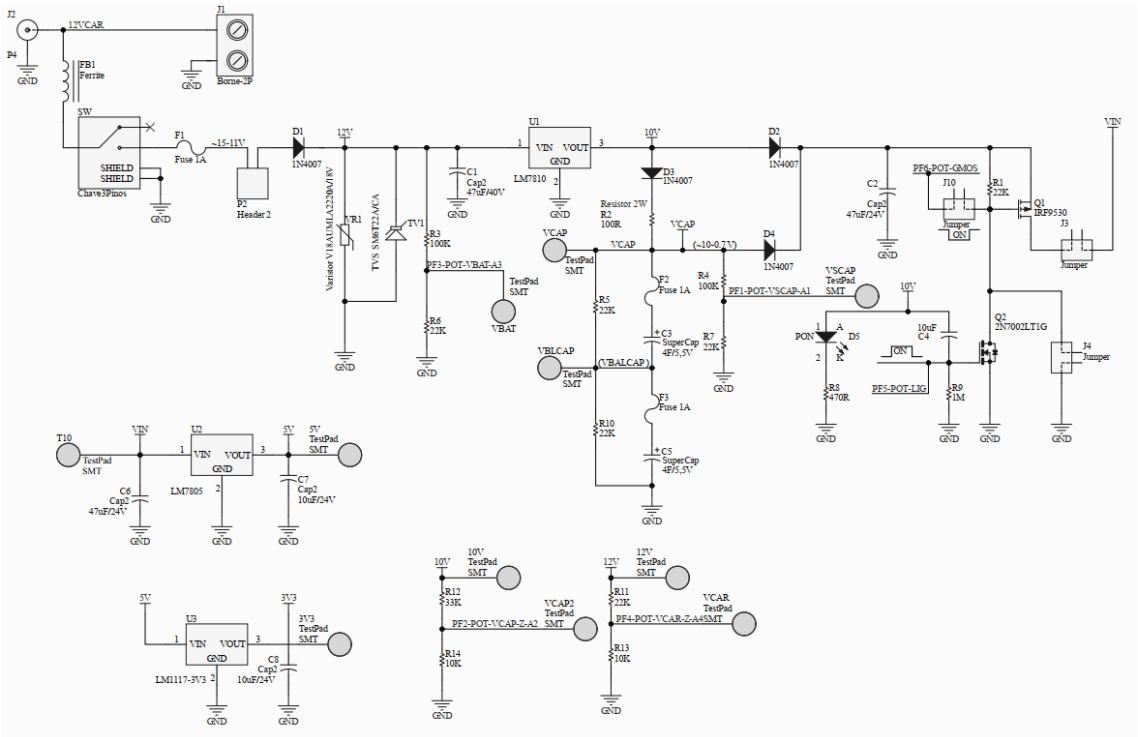

Figura 3.4: Circuito que possibilita o auto desligamento do Arduino [7]

O componente principal para que seja possível a função de auto desligamento é Q1, o MOSFET IRF9530, que tem canal P e atua no corte e saturação. Logicamente, para que a alimentação seja entregue ao circuito, o jumper J3 deve estar fechado. No momento que o MOSFET entra em saturação, é como se formasse um curto-circuito entre o Dreno (D1) e a Fonte (S1). O controle entre o corte e a saturação é feito pela Porta (G1).

• G1 = nível alto (*HIGH*) -> Q1 corta e remove a alimentação da Caixa Preta

• G1 = nível baixo (*LOW*) -> Q1 conduz e alimenta a Caixa Preta.

Vale destacar que R1 funciona como um resistor de *pullup*, que coloca G1 em nível alto quando a porta não é acionada. Esta porta G1 é controlada pelo transistor Q2, o 2N7002LT1G, um MOSFET de canal N, e pelo jumper J4. Sendo assim, Q2 controla o nível de tensão de G1, ou seja, é o responsável por ligar ou desligar Q1. Então temos que:

• Se J4 = fechado, então G1 = nível baixo e a alimentação da CXP está sempre ligada

• Se J4 = aberto, então a alimentação da CXP é controlada pelo transistor Q2.

Se J4 = aberto, o transistor Q2 controla a porta (G1) de Q1 e, por consequência, a alimentação da Caixa Preta. Logo:

• Se G2 = nível alto, Q2 conduz, o que faz G1 = nível baixo e a Caixa Preta é alimentada

• Se G2 = nível baixo, Q2 corta e G1 = 10V graças ao *pullup* R1, então a Caixa Preta fica sem alimentação.

Para controlar o transistor Q2 utiliza-se um pino digital do Arduino. Neste caso, o pino PG.0 foi conectado à porta G2 de Q2, conforme representado na Figura 3.4. Deste modo, ao se controlar o nível lógico do pino é possível manter o Arduino ligado ou desligá-lo.

Para garantir que a Caixa Preta ligue toda vez em que o carro for ligado, foi adicionado o capacitor C4. Ao ligar o circuito, ele está sem carga e por isso a tensão de 10V surge na porta de Q2, que conduz e liga o circuito. O capacitor se carrega lentamente pelo resistor R9 (1 MΩ), o que dá tempo para a Caixa Preta fazer as inicializações necessárias e colocar o pino PG.0 em nível alto, garantindo a alimentação. O tópico a seguir apresenta medições e faz uma análise detalhada desta partida.

3.1.4 Medições e Análise do Circuito

Com o jumper J4 aberto, que é o funcionamento normal, tem-se o seguinte problema: Como levar o gate (G2) de Q2 para nível alto e ligar a alimentação da Caixa Preta? A solução é dada por C4 (10 μF) e R9 (1 MΩ). No momento em que está desligado, C4 se descarrega. No instante em que a alimentação é fornecida (VIN = 10V), como C4 está descarregado, ele funciona como um curto. À medida que C4 se carrega por R9, a tensão em G2 começa a cair. O importante é que C4 consiga manter G2 = nível alto o tempo suficiente para a CPU fazer as inicializações, rodar o programa e fazer PG.0 = nível alto, garantindo então a alimentação para o funcionamento da Caixa Preta.

Foram feitos alguns ensaios acerca das condições de alimentação da Caixa Preta, conforme mostrados a seguir.

Na Figura 3.5 abaixo temos o funcionamento normal, com o jumper J4 aberto. Ao ligar, o capacitor C4 está descarregado e com isso surge em G2 uma tensão de, aproximadamente, 5,8 V. Esta tensão é suficiente para manter Q2 conduzindo. Após 908 ms, é terminada a inicialização do Arduino e é entregue a execução para o programa do usuário, que, como primeira tarefa faz o pino PG.0 = nível alto (G2 = nível alto). Assim, Q2 conduz de forma permanente e resulta em G1 = nível baixo, o que garante a alimentação da Caixa Preta.

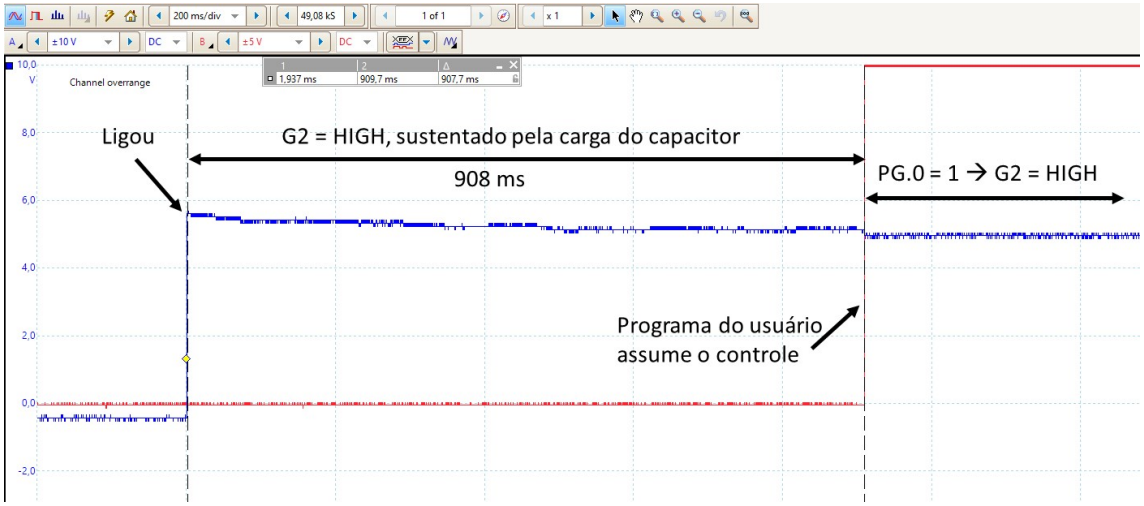

Figura 3.5: Ensaio de energização da Caixa Preta. Em azul, tensão sobre G2 e em vermelho o instante em que a CPU assumiu o controle.

Na Figura 3.6 é apresentado o caso em que a CPU não faz PG.0 = nível alto. Por esse motivo, a alimentação é temporariamente garantida pela carga no capacitor C4 (G2 = nível alto). A intenção foi a de verificar o intervalo máximo de sustentação da alimentação, sem que o programa fizesse qualquer intervenção. Como pode ser visto, com a carga do capacitor, a tensão em G2 foi caindo e quando chegou em 4V, a CPU se desligou por falta de alimentação (*brownout*). A conclusão é a de que, ao ligar, existe o limite de tempo de 1,905 segundos para se fazer PG.0 = nível alto (G2 = nível alto). Na Figura 3.5 se pode ver que o programa do usuário consegue ativar esta condição PG.0 = nível alto em menos de 1 segundo, o que é suficiente. A folga é de aproximadamente 100%.

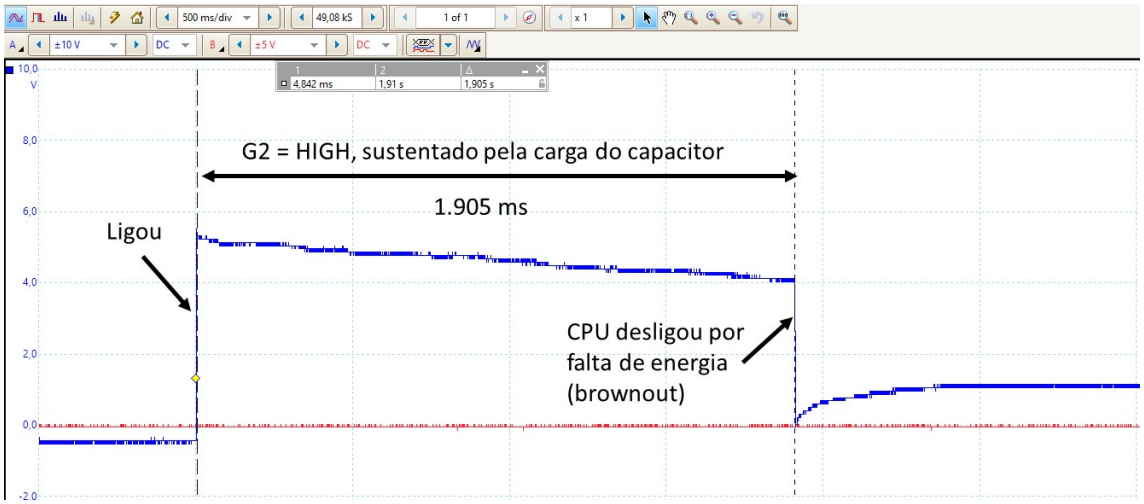

Figura 3.6: Ensaio do tempo em que C4 consegue sustentar a alimentação da Caixa Preta. Em azul, tensão sobre G2

A Figura 3.7 apresenta uma visão mais ampla. A alimentação é aplicada, o Arduino é energizado e depois de 0,9 segundos, o programa do usuário assume o controle e faz PG.0 = nível alto, garantindo a alimentação. Depois de 1 segundo rodando, o programa desliga a Caixa Preta, fazendo PG.0 = nível baixo. Desta forma, se consegue ver os acontecimentos ao ligar e desligar a CXP.

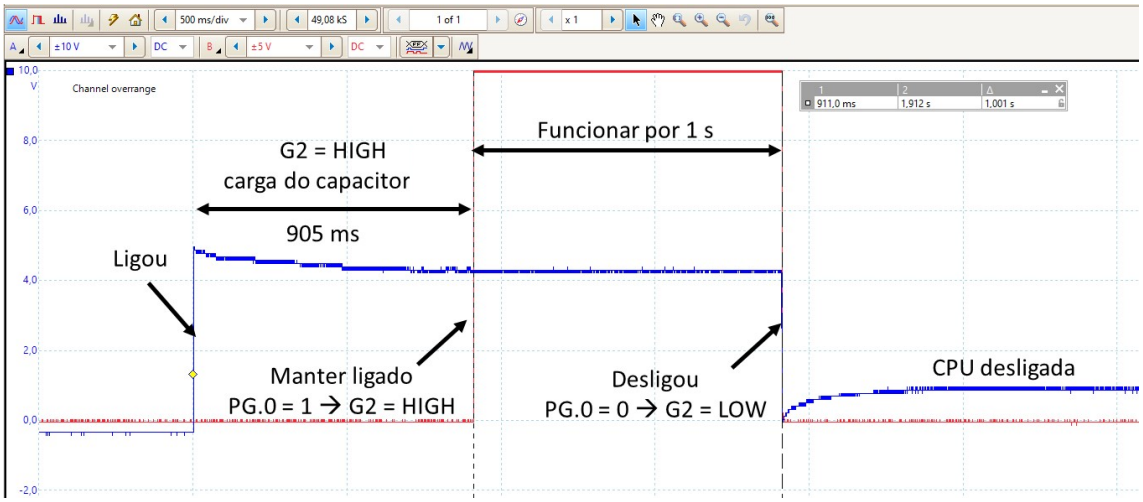

Figura 3.7: Ensaio de liga/desliga da Caixa Preta. O ensaio inicia quando a alimentação é ligada. Depois, o programa inicia e faz PG.0 = nível alto (sinal em vermelho), garantindo a alimentação. Decorrido 1 segundo o programa faz PG.0 = nível baixo para desligar a Caixa Preta.

A Figura 3.8 apresenta a mesma situação, porém a linha vermelha agora indica a alimentação de 5 V, que é ativada logo após a Caixa Preta ser ligada. Esta tensão permanece estável até o instante em que se desliga a Caixa Preta. Vale notar que com a Caixa Preta desligada (G2 = nível baixo) ainda permanece uma tensão residual de 0,7 V, aproximadamente.

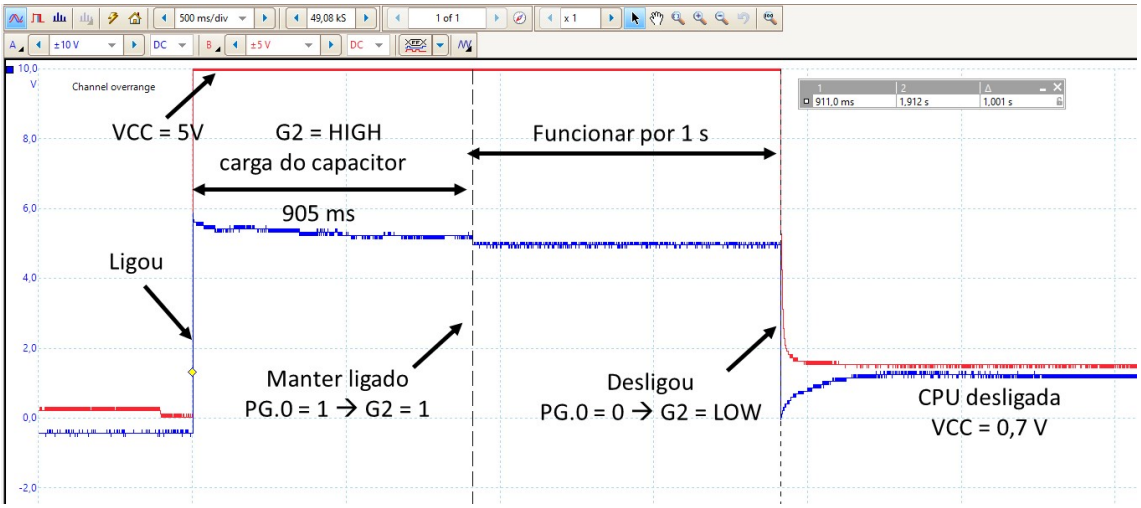

Figura 3.8: Mesma situação da figura anterior, mas agora a linha vermelha indica a tensão de 5V, que alimenta todos os circuitos da Caixa Preta.

3.1.5 Interrupções e Monitoramento das Tensões

Para criar uma base de tempo para a Caixa Preta, um dos timers da CPU é programado para gerar uma interrupção a cada 10 ms (100 Hz). Isto facilitou a execução de tarefas periódicas e a geração de pequenos atrasos (*delays*). São três as principais tarefas desta interrupção:

• Se for preciso, disparar a atualização do LCD;

• Verificar e atualizar a situação das filas de dados e entrada e saída

• Disparar o ADC para ler sinais analógicos importantes.

O comando do ADC é um item muito importante e merece uma explicação detalhada. Na Caixa Preta, temos 3 sinais analógicos monitorados na rotina de interrupção, como mostrado na Figura 3.9 abaixo. É importante lembrar que o esquema com as 5 chaves foi construído com divisores resistivos, daí o monitoramento pelo ADC.

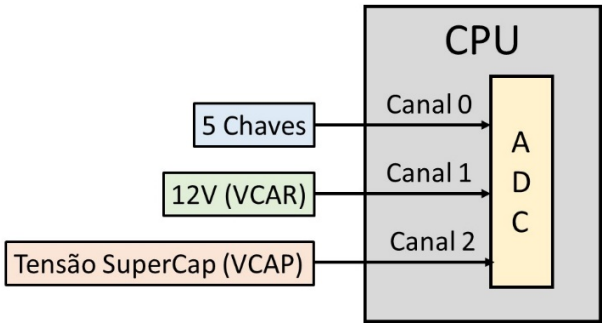

Figura 3.9: Ilustração dos sinais analógicos monitorados pela Caixa Preta.

É sabido que a conversão realizada pelo ADC consome tempo. Uma outra restrição é a necessidade de se aguardar um pequeno período a cada troca de canal analógico. Para que tudo isto acontecesse de forma automática e que o programa tivesse, a qualquer momento, disponíveis as informações das chaves, da tensão de alimentação (VCAR) e da tensão sobre o supercapacitor (VCAP), foi criada uma máquina com 32 estados, apresentada na tabela 3.1 abaixo.

Tabela 3.1: Ciclo de interrupções para monitoramento das chaves e tensões

0) ADC Start	8) +Ler, ADC start*	16) ADC Start	24) +Ler, ADC start*
1) Ler, ADC start	9) Ler, ADC start	17) Ler, ADC start	25) Ler, ADC start
2) +Ler, ADC start*	10) +Ler, ADC start*	18) +Ler, ADC start*	26) +Ler, ADC start*
3) Ler, ADC start	11) Ler, ADC start	19) Ler, ADC start	27) Ler, ADC start
4) +Ler, ADC start*	12) +Ler, Canal1(VCAR)*	20) +Ler, ADC start*	28) +Ler, Canal 2(VCAP)*
5) Ler, ADC start	13) ADC Start	21) Ler, ADC start	29) ADC Start
6) +Ler, ADC start*	14) Ler, ADC start	22) +Ler, ADC start*	30) Ler, ADC start
7) Ler, ADC start	15) +Ler, Canal 0	23) Ler, ADC start	31) +Ler, Canal 0

* indica a fase para calcular a média da leitura do teclado e identificar o estado das chaves

Esta tabela está muito densa e merece algumas explicações. Iniciamos indicando como as chaves são tratadas. Para determinar os estados das chaves, é usada a média de duas conversões consecutivas. Por exemplo, os estados 3 e 4, como mostrado na Figura 3.10 abaixo.

No estado 3, é feita uma leitura do ADC e é disparada uma nova conversão. No estado 4, é feita a leitura do ADC, disparada uma nova conversão, calculada a média (com a leitura anterior) e o estado das chaves é determinado. Assim, o processo se repete.

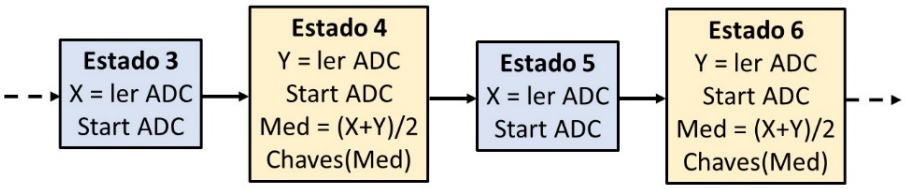

Figura 3.10: Pequeno trecho da máquina de estados que roda dentro da rotina de interrupção e que determina o estado das chaves.

A troca de canal do ADC precisa de um certo tempo para estabilização, por isso há um período de 10 ms após cada troca de canal. Essa troca acontece nos estados 12, 15, 28 e 31, e a primeira conversão é disparada nos estados seguintes, ou seja, estados 13, 16, 29 e 0. A Figura 3.11 abaixo faz uma síntese dos estados.

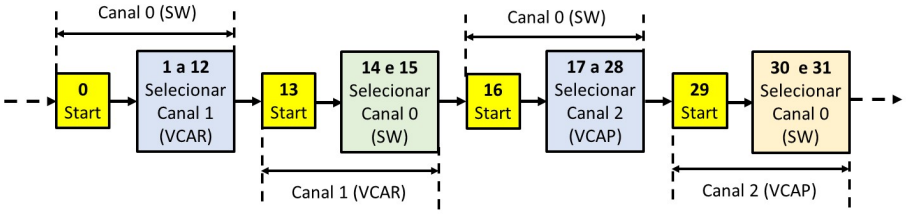

Figura 3.11: Síntese dos estados para monitoramento das chaves e tensões

A tabela 3.2 abaixo apresenta um resumo das operações mais importantes realizadas em cada estado. Podemos então afirmar que:

• A tensão de entrada VCAR (12 V) é atualizada a cada 320 ms;

• A tensão sobre o supercapacitor VCAP é atualizada a cada 320 ms;

• O estado das chaves é atualizado 12 vezes a cada 320 ms, ou seja, o estado das chaves é atualizado, aproximadamente, a cada 27 ms (320/12 ms).

A função que monitora as chaves mantém atualizadas as variáveis que indicam o estado instantâneo de cada chave e qual delas passou do estado de “aberta” para “fechada”. Testes experimentais indicaram que a taxa de 37 Hz (período = 27 ms) é suficiente para garantir o bom funcionamento das chaves.

Tabela 3.2: Resumo das operações realizadas nos diversos estados

Canal	Estado	Descrição
0	0	Start do ADC
0	1 a 12	A cada 2 conversões calcular a média e analisar o estado das chaves. A última operação do estado 12 é selecionar o canal 1.
1	13	Start do ADC
1	14 e 15	Usar a média de 2 para determinar a tensão VCAR (12 V). A última operação do estado 15 é selecionar o canal 0.
0	16	Start do ADC
0	17 a 28	A cada 2 conversões calcular a média e analisar o estado das chaves. A última operação do estado 12 é selecionar o canal 1.
2	29	Start do ADC
2	30 e 31	Usar a média de 2 para determinar a tensão VCAP (Super capacitor). A última operação do estado 31 é selecionar o canal 0.

3.2 MODOS DE OPERAÇÃO

Para facilitar o teste dos diversos componentes do sistema e ajudar no desenvolvimento das primeiras versões operacionais foi criado um laço principal que opera por menus. Existem dois modos de operação: o modo teste e o modo operação.

Ao ligar a Caixa Preta, é feita a inicialização dos diversos componentes e é estabelecida a comunicação serial com o PC via cabo USB ou via Bluetooth. Ao final da inicialização, são enviadas mensagens seriais e o LCD é atualizado. A partir deste instante, fica disponível um menu com opções que podem ser selecionadas pelos botões presentes na placa ou pela porta serial.

3.2.1 Modo Teste

Neste modo, é possível realizar diferentes testes individualmente ou em conjunto para garantir que os elementos que compõem a Caixa Preta estão funcionando adequadamente. Isso garante que se possa partir para o modo de operação com uma maior confiança no sistema, pois através dos testes se pode verificar se os dados que estão sendo obtidos possuem coerência e estão sendo gerados conforme o esperado. As opções de testes são exibidas conforme a Figura 3.12 abaixo:

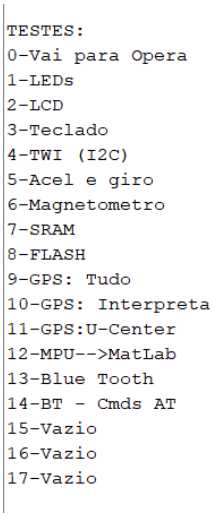

Figura 3.12: Opções do Modo Teste

Vale destacar que algumas opções estão caracterizadas como “Vazio”. Isso significa que são espaços vazios para que o usuário possa ter a liberdade de implementar o tipo de teste que desejar. No código do Arduino há espaços específicos separados para este propósito, bastando que o usuário escreva a função de acordo com os objetivos pretendidos. Também é importante lembrar que foi definido que o comando “x” interrompe qualquer teste que esteja em execução e volta para o menu onde as funcionalidades são exibidas. Pode-se inserir a letra “x” no terminal serial ou pressionar o botão central na placa, o que também é possível através da interface gráfica que será explicada mais adiante.

Como pode ser visto na Figura 3.12 acima, a partir do Modo Teste é possível ir para o Modo Opera, além de possibilitar testes dos LEDs presentes na placa e verificar se o LCD, as teclas e a comunicação I2C (também conhecida como TWI) estão funcionando corretamente. As opções 5 e 6 imprimem dados obtidos com acelerômetro, giroscópio e magnetômetro e as opções 7 e 8 disponibilizam informações a respeito das memórias SRAM e FLASH. Na opção 9, são impressas no terminal serial todas as informações obtidas com o GPS, lembrando que através da opção 10 se pode extrair e interpretar as mensagens do GPS.

A opção 11 “GPS: U-Center” fornece dados que são tratados de uma forma diferenciada, isto é, de forma conjunta com uma plataforma que os interpreta. O U-Center é um software da U-blox, uma empresa suíça com foco em semicondutores e módulos sem fio para aplicação em diferentes áreas, como automotiva e industrial. O software tem como principal funcionalidade a capacidade de obter dados relacionados ao Sistema Global de Navegação por Satélite (GNSS) e permitir que o usuário obtenha representações de localização de uma forma intuitiva e personalizada. Após selecionar o dispositivo na porta COM para comunicação serial e realizar as configurações desejadas, o usuário pode ter acesso a diferentes formas de visualização, como a visualização de mapa, visualização do posicionamento dos satélites e visualização do sinal dos satélites [32].

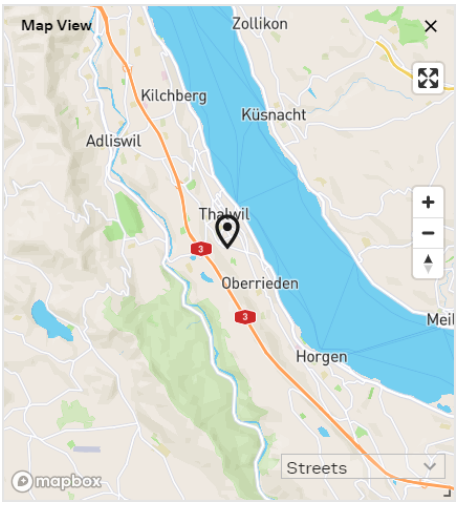

Figura 3.13: Visualização de Mapa (*Map View*) [33]

No modo representado na Figura 3.13 acima é exibida a localização obtida com os dados do dispositivo selecionado ou com um arquivo de log. Na visualização é possível controlar o mapa normalmente, com a possibilidade de aplicar zoom ou mudar o tipo de mapa apresentado [33].

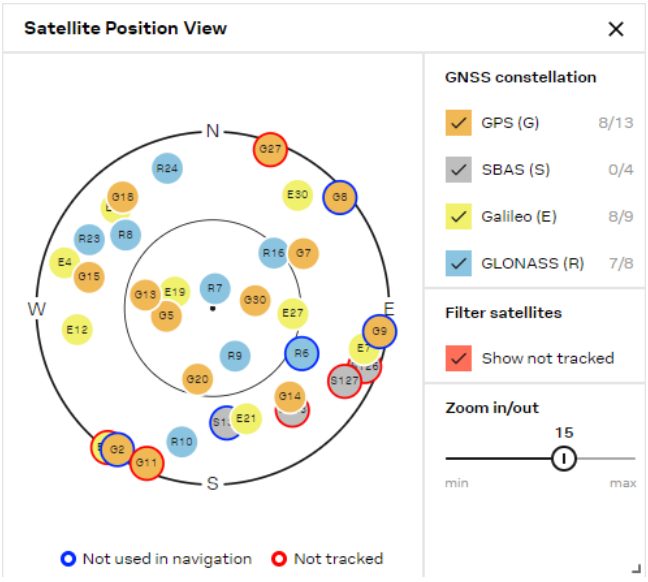

Figura 3.14: Visualização do Posicionamento dos Satélites (*Satellite Position View*) [33]

No modo de visualização da Figura 3.14 acima é possível observar a posição dos satélites e a forma como estão distribuídos, isto é, a constelação. No menu que fica disponível no lado direito da tela é possível utilizar um filtro para que sejam exibidos apenas os tipos de satélite desejados [33].

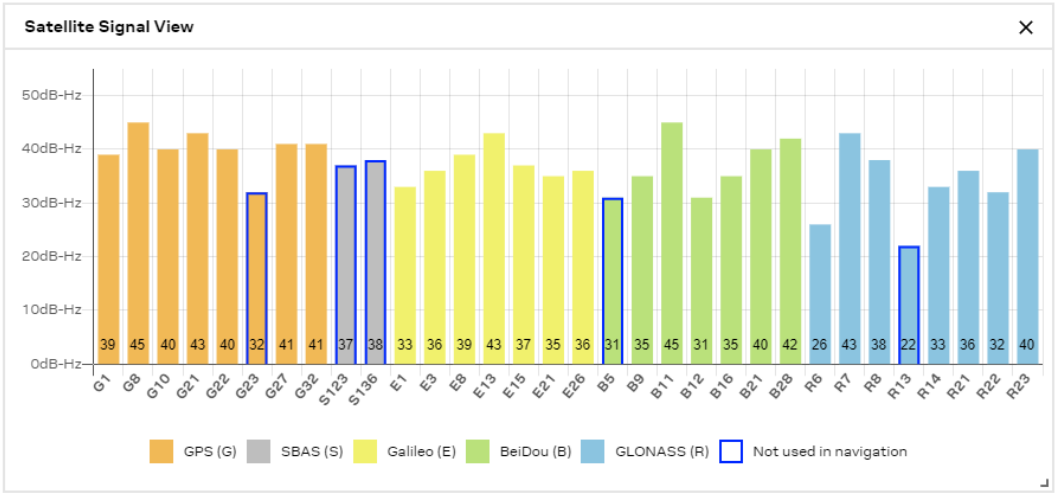

Figura 3.15: Visualização do Sinal dos Satélites (*Satellite Signal View*) [33]

No modo de exibição mostrado na Figura 3.15 acima se pode verificar a força do sinal dos satélites detectados tendo como base o dispositivo conectado ou um arquivo com logs. Os sinais são representados e agrupados de acordo com sua constelação GNSS em diferentes cores.

Além dos testes descritos anteriormente, também é possível verificar o funcionamento da conexão bluetooth da Caixa Preta. Adicionalmente, se pode realizar o teste de comandos AT, uma linguagem de comandos com pequenos textos que desempenham uma determinada operação. Isso pode ser utilizado para configurar o módulo bluetooth HC05, bastando segurar o botão localizado logo ao lado do módulo, o que vai conduzir o dispositivo a um modo que recebe comandos da porta COM de número 0. Entretanto, se preferiu não disponibilizar e aplicar este último teste tendo em vista que isso iria realizar alterações indesejadas nas filas, as quais foram priorizadas na implementação de outras funcionalidades.

3.2.2 Modo de Operação

O modo de operação é para se ensaiar o funcionamento, de fato, da Caixa Preta, o que permitirá obter e armazenar todas as informações necessárias para avaliar toda a movimentação feita pelo automóvel e se analisar a dinâmica do acidente. É neste modo que o dispositivo fica constantemente coletando dados com o MPU e GPS, gravando-os na SRAM até que ocorra um evento que interrompa esse ciclo, como uma batida ou um acidente. Neste ponto, conforme explicado anteriormente, mesmo com o corte da energia, os supercapacitores (se necessário) sustentam a alimentação de forma a permitir que se complete o ciclo de coleta de dados e a consequente gravação na memória Flash. Este modelo de operação possibilita que se tenha as informações antes, durante e após o evento de um acidente, gerando uma maior riqueza de detalhes e viabilizando uma análise mais completa.

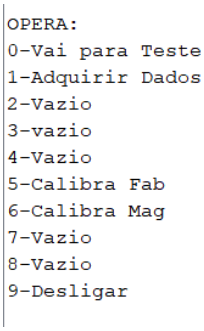

Figura 3.16: Opções do Modo de Operação

Como pode ser visto na Figura 3.16 acima, neste modo é possível não só iniciar a aquisição de dados, mas também realizar calibrações. Uma das opções é a calibração do magnetômetro e a outra é a calibração de fábrica, que traz os dados da preparação para o funcionamento da CXP. São dados de bias dos sensores, aceleração da gravidade no local de calibração, parâmetros do magnetômetro, temperatura, resultados de *self-test*, entre outros. Tudo isso fica gravado na EEPROM da CPU.

Ao selecionar a opção 1, "Adquirir dados", tem início o processo de obtenção de informações através do acelerômetro, giroscópio, magnetômetro e também do GPS, ou seja, neste modo é gerado um bloco de dados que possibilita a análise pós-acidente. Todas essas informações ficam separadas de acordo com o formato mostrado na tabela 3.3 abaixo:

Tabela 3.3: Formato dos dados no modo de Aquisição de Dados [4]

Delimitador	**Descrição**
#[m ... m]#	Dados de aquisição do MPU-9250
#[g ... g]#	Dados do GPS
#[l ... l]#	Dados com resultados da Calibração ao Ligar
#[f ... f]#	Dados da Calibração de Fábrica

3.3 INTERFACE GRÁFICA PARA CONTROLE DA CAIXA PRETA

Foi mencionado anteriormente que o terminal serial e o *display* LCD permitem ao usuário selecionar o modo de operação e funcionalidades da CXP. Visando tornar isso ainda mais intuitivo, foi desenvolvida uma GUI (Interface Gráfica do Usuário) que permite o controle da Caixa Preta pelo computador de uma forma mais simplificada e com um visual mais claro, mostrando ao usuário o modo em que a Caixa Preta está e as opções que podem ser acessadas apenas com um *click*.

A interface foi desenvolvida com a linguagem de programação Python, na versão 3.9, e com a biblioteca PyQt5, que, juntamente o *Qt Designer*, uma ferramenta utilizada para desenvolver GUIs, torna possível desenvolver e customizar interfaces do modo que o usuário preferir. Outra biblioteca fundamental é a *pySerial*, necessária para realizar a comunicação serial através das portas COM, viabilizando o envio e recebimento de informações. Foi usado como base o código do tutorial do canal DoTic do *Youtube* [1], realizando adaptações para os objetivos deste projeto.

A Figura 3.17 abaixo mostra a interface que é exibida para o usuário:

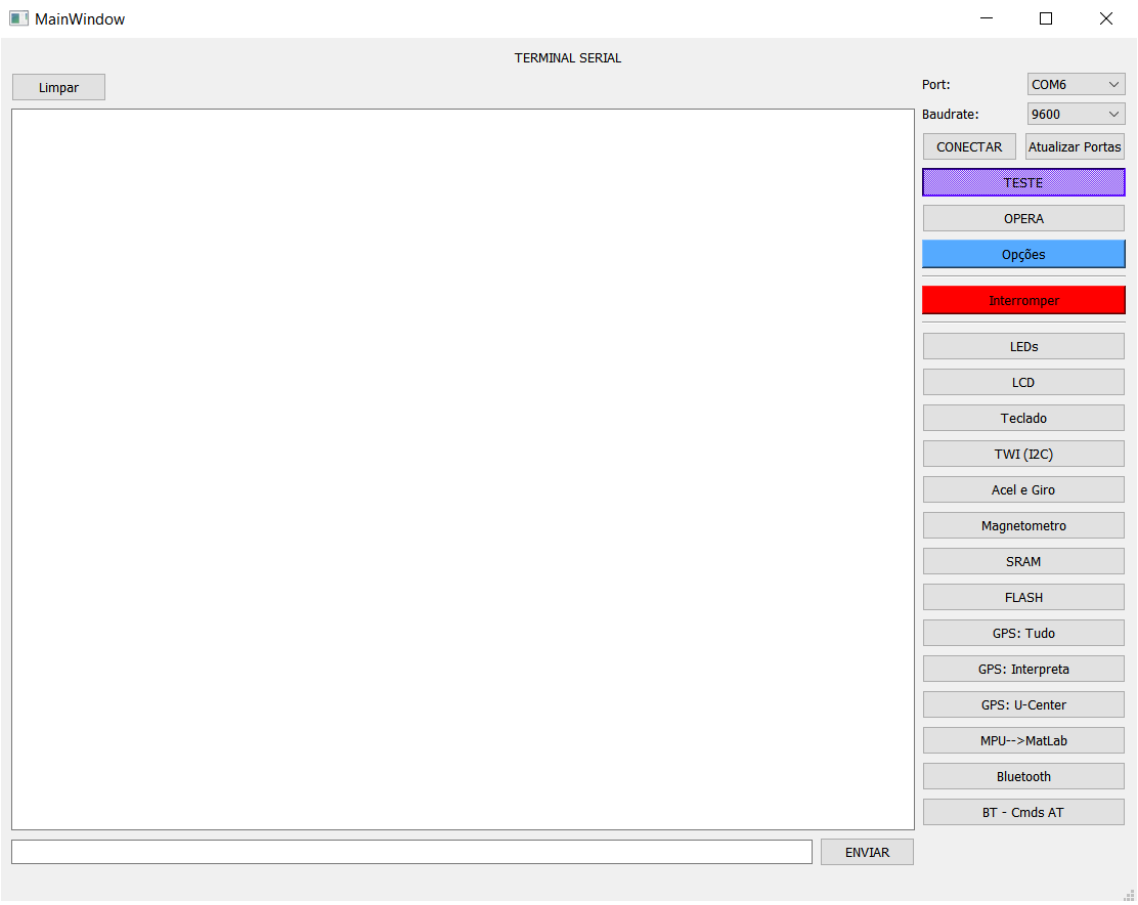

Figura 3.17: Interface Gráfica do Usuário

A interface, além de possuir botões para controle da Caixa Preta, conta com um terminal serial que atualiza em tempo real. Este terminal conta com um espaço para digitar e enviar comandos e com um botão para limpar o texto que está sendo exibido.

Assim que a interface inicia-se, o usuário pode atualizar as portas COM disponíveis, escolher a porta COM desejada e o *Baudrate*, que é a taxa de transferência de dados na comunicação serial. Em seguida deve pressionar o botão "CONECTAR" para iniciar a comunicação serial. Neste momento o botão é atualizado para a opção "DESCONECTAR", que pode ser pressionado a qualquer momento para desconectar o dispositivo. Caso seja pressionado, o botão "CONECTAR" fica disponível novamente.

Feita a conexão, o usuário pode então selecionar o modo de operação, que fica iluminado, indicando o modo em que a Caixa Preta está operando. Ao pressionar o botão “Opções”, são exibidas todas as funcionalidades disponíveis no terminal serial. Estas são as mesmas opções em forma de botão que estão disponíveis logo abaixo do botão “Interromper”. É importante destacar que estes botões são sensíveis ao modo de operação, ou seja, eles são atualizados de acordo com o modo em que a Caixa Preta se encontra. Isso pode ser observado na Figura 3.18 abaixo, onde o dispositivo se encontra no modo “OPERA”.

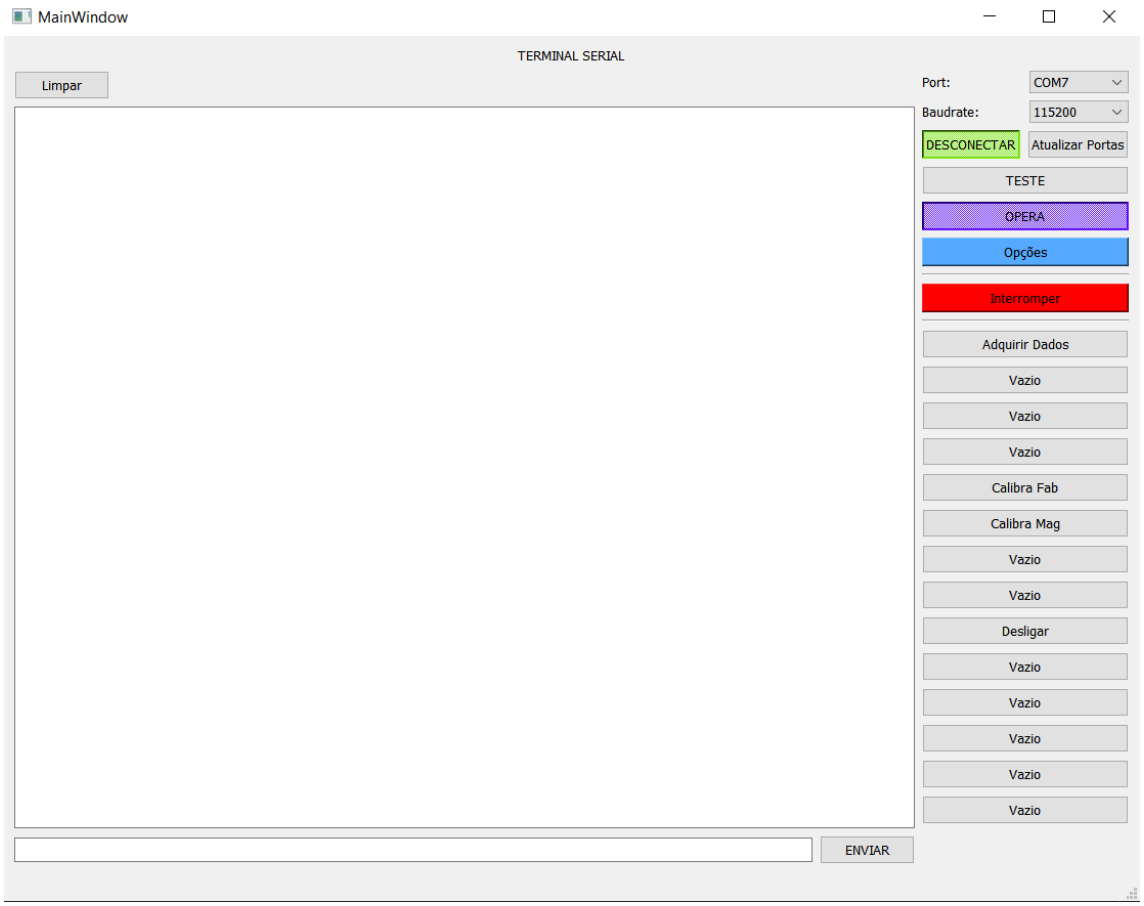

Figura 3.18: Interface no modo de operação

Outro fator importante é o botão “Interromper”. Ele se aplica nos casos em que alguma funcionalidade foi acionada dentro de um dos modos de operação. No caso do teste dos LEDs, por exemplo, quando o botão “Interromper” é pressionado, a função será interrompida e então se retorna à etapa de seleção de funcionalidades.

O programa também possibilita que, em segundo plano, seja criado um arquivo texto com todas as informações exibidas no terminal serial. Isso se dá assim que o usuário clica no botão "CONECTAR", o que inicia a comunicação serial. Neste exato momento se cria um arquivo texto no formato "dia-mês-ano—hora-minutos-segundos.txt". Então tudo que for impresso no terminal será enviado para este arquivo. No momento que o usuário clica em "DESCONECTAR" ou fecha a interface, o arquivo é salvo. Isso permite que o usuário tenha uma espécie de relatório do que foi feito, podendo realizar diversas funções para posteriormente analisar os dados obtidos e verificar o funcionamento da Caixa Preta.

- **Códigos**

Conforme mencionado anteriormente, o código para a implementação desta interface gráfica é uma adaptação do código do tutorial disponível no *Youtube* do canal DoTic [1]. Os códigos estão divididos em três arquivos Python: *main.py*, *customSerial.py* e *GUI.py* (todos disponíveis no apêndice). Este último é o que contém todas as informações do que foi desenvolvido na ferramenta *Qt Designer*. No *Qt Designer*, ao salvar o projeto, é gerado um arquivo *.ui* (neste caso, *GUI.ui*). É nele que está descrito (em um formato semelhante a xml) tudo que foi desenvolvido de forma gráfica. Após obter este arquivo, no terminal (ou *cmd* no sistema operacional *Windows*), o usuário deve executar o seguinte comando dentro da pasta onde este está localizado para fazer a conversão e obter o arquivo *.py*:

#pyuic5 [nome do arquivo].ui -o [nome do arquivo].py

Este comando deve ser executado sempre que alguma alteração for feita no projeto da interface gráfica desenvolvida no *Qt Designer*. É isso que permite gerar todos os elementos presentes na interface para que possam ser utilizados na lógica do código, como o apertar de um botão, por exemplo.

No arquivo *main.py* é onde toda a lógica é implementada para que a interface seja exibida e para que os comandos gerem os resultados esperados. É importante lembrar que deve-se importar tudo que está presente no arquivo *GUI.py*, o que é feito com o comando abaixo:

```
1 from GUI import *
```

O mesmo vale para o arquivo *customSerial.py*, que serve como suporte para o arquivo principal no que diz respeito à comunicação serial. Algo fundamental implementado neste arquivo é a funcionalidade de *threading*, que permite que certas ações sejam feitas simultaneamente. Sua importância se dá principalmente ao fato de ser necessário fazer leituras ao mesmo tempo que se recebe comandos, pois a leitura é feita dentro de um loop infinito. Para que seja possível utilizar este recurso, é necessário importar a biblioteca *threading*, conforme pode ser observado no código disponibilizado no apêndice.

Com relação ao arquivo *main.py*, há comandos que devem ser levados em consideração ao se trabalhar com os botões presentes na interface, como por exemplo:

```
self.ui.botaoTeste.clicked.connect(self.teste)
```

No exemplo acima é verificado se o botão “TESTE” foi pressionado. Em caso afirmativo, é chamada a função “teste”. A partir daí será executado tudo que estiver dentro da função.

Outro comando importante é o do exemplo abaixo:

```
self.ui.connectBtn.setText('DESCONECTAR')
```

Este comando é utilizado para alterar o texto que é exibido no botão. Neste caso se trata do botão “CONECTAR”, que ao ser pressionado tem seu texto modificado para “DESCONECTAR”. Há também um comando que permite definir o status de um botão como pressionado ou não. Isso se aplica a botões que, ao serem pressionados, se mantêm desta forma, como os botões “CONECTAR”, “TESTE” e “OPERA”. Para que o botão se mantenha pressionado basta utilizar o parâmetro “True” e para que volte ao normal o parâmetro “False”. Este comando é o do exemplo abaixo:

```
self.ui.botaoTeste.setChecked(False)
```

4 ENSAIOS E RESULTADOS DOS TESTES

4.1 TESTE DO MODO DE OPERAÇÃO

Ao testar o modo de operação, que é onde, de fato, acontece a aquisição de dados com os sensores e demais elementos da caixa preta, se notou que ela se comportou conforme o esperado.

Inicialmente, assim que o "Modo Opera" é selecionado tem início o processo de preenchimento da SRAM com dados. Nesta etapa as informações exibidas no terminal serial são as informações obtidas pelo GPS, conforme mostrado na Figura 4.1 abaixo:

```
[1] Adquirir Dados
Zerando SRAM
SEL = Inic Aquisicao  SW_INF = Interrompe
S=sel, I=Interrompe, X=sai
             1.85 1.25 1.37 414
A 141628 070922 1552.86209 S 04803.95330 W 0.082 K 0.044  1239.4 M 08 1.85 1.25 1.37 2196
A 141629 070922 1552.86232 S 04803.95334 W 0.024 K 0.013  1240.0 M 08 1.85 1.25 1.37 3978
A 141630 070922 1552.86249 S 04803.95337 W 0.036 K 0.019  1240.3 M 08 1.85 1.25 1.37 5796
A 141631 070922 1552.86256 S 04803.95335 W 0.070 K 0.038  1240.4 M 08 1.85 1.25 1.37 7596
A 141632 070922 1552.86267 S 04803.95337 W 0.044 K 0.024  1240.5 M 08 1.85 1.25 1.37 9396
A 141633 070922 1552.86266 S 04803.95334 W 0.035 K 0.019  1240.3 M 09 1.65 0.97 1.33 11214
A 141634 070922 1552.86255 S 04803.95330 W 0.045 K 0.024  1240.0 M 09 1.65 0.97 1.33 13014
A 141635 070922 1552.86248 S 04803.95327 W 0.011 K 0.006  1239.7 M 09 1.65 0.97 1.33 14814
A 141636 070922 1552.86240 S 04803.95322 W 0.061 K 0.033  1239.3 M 09 1.65 0.97 1.33 16614
A 141637 070922 1552.86236 S 04803.95320 W 0.003 K 0.002  1239.2 M 09 1.65 0.97 1.33 18414
A 141638 070922 1552.86225 S 04803.95315 W 0.041 K 0.022  1238.9 M 09 1.65 0.97 1.33 20214
A 141639 070922 1552.86218 S 04803.95313 W 0.066 K 0.036  1238.8 M 09 1.65 0.97 1.33 22014
A 141640 070922 1552.86216 S 04803.95313 W 0.005 K 0.003  1238.7 M 09 1.65 0.97 1.33 23814
A 141641 070922 1552.86210 S 04803.95311 W 0.092 K 0.050  1238.6 M 09 1.65 0.97 1.33 25614
A 141642 070922 1552.86205 S 04803.95309 W 0.063 K 0.034  1238.6 M 09 1.65 0.97 1.33 27414
A 141643 070922 1552.86198 S 04803.95307 W 0.068 K 0.037  1238.6 M 09 1.65 0.97 1.33 29196
A 141644 070922 1552.86189 S 04803.95305 W 0.078 K 0.042  1238.6 M 09 1.65 0.97 1.33 31014
A 141645 070922 1552.86180 S 04803.95304 W 0.018 K 0.010  1238.7 M 09 1.65 0.97 1.33 32814
A 141646 070922 1552.86173 S 04803.95302 W 0.064 K 0.034  1238.8 M 09 1.65 0.97 1.33 34614
A 141647 070922 1552.86166 S 04803.95302 W 0.044 K 0.024  1239.0 M 09 1.65 0.97 1.33 36396
```

Figura 4.1: Início do Modo de Operação

A forma como o dispositivo está configurado nos dá a liberdade de realizar os mais variados testes. Tendo isso em vista, buscou-se implementar uma forma de escolher o momento em que se inicia a leitura e registro dos dados, que mais adiante serão transferidos para a memória EEPROM. Isso foi feito utilizando o botão central da placa, que fica entre os direcionais e é rotulado como "SEL". Ao ser pressionado, a caixa preta inicia o modo de aquisição, lendo e registrando todas as informações obtidas com os sensores, o que ocorre por um determinado tempo (aproximadamente um minuto e trinta segundos). Estes dados ficam organizados da forma que foi mencionada na seção 3.2, com os respectivos delimitadores para cada tipo de informação.

```
==>> ADQUIRINDO <<==
A 141850 070922 1552.86112 S 04803.95212 W 0.238 K 0.129  1239.2 M 09 1.84 1.24 1.36 23454
A 141851 070922 1552.86107 S 04803.95210 W 0.055 K 0.030  1239.1 M 09 1.84 1.24 1.36 25236
A 141852 070922 1552.86106 S 04803.95209 W 0.280 K 0.151  1239.2 M 09 1.84 1.24 1.36 27054
A 141853 070922 1552.86104 S 04803.95206 W 0.302 K 0.163  1239.1 M 09 1.84 1.24 1.36 28836
A 141854 070922 1552.86099 S 04803.95202 W 0.144 K 0.078  1239.2 M 09 1.84 1.24 1.36 30636
A 141855 070922 1552.86096 S 04803.95197 W 0.347 K 0.187  1239.2 M 09 1.84 1.24 1.36 32436
A 141856 070922 1552.86093 S 04803.95193 W 0.505 K 0.273  1239.4 M 09 1.84 1.24 1.36 34236
A 141857 070922 1552.86083 S 04803.95190 W 0.103 K 0.056  1239.4 M 09 1.84 1.24 1.36 36036
A 141858 070922 1552.86074 S 04803.95188 W 0.140 K 0.076  1239.4 M 09 1.84 1.24 1.36 37836
A 141859 070922 1552.86064 S 04803.95185 W 0.043 K 0.023  1239.4 M 09 1.84 1.24 1.36 39636
A 141900 070922 1552.86055 S 04803.95185 W 0.166 K 0.090  1239.3 M 09 1.84 1.24 1.36 41454
```

Figura 4.2: Momento em que o botão “SEL” é pressionado

Como pode ser observado na Figura 4.2 acima, após o evento, o registro da caixa preta exibe dados do GPS e em seguida os dados do MPU, delimitados por “#[m” e “m]#”, conforme a Figura 4.3 disponibilizada logo abaixo.

```
#[m


6039
43740
232576
+21331 +20046 +2 +250 +4308 -385 +354 -16530 +46 -113 +258
+8242 +12337 +14080 +13358 +13102 +13056 +22051 +8224 +8257 +17476 +21024
+20046 +20046 +2 +250 +100 +5 +1  +2  +3  +4  +5
12720
00000 00000000 -00404 +00396 -16716 +00050 -00136 +00258 -00144 +00418 -00228
00001 00000012 -00424 +00356 -16672 +00047 -00136 +00260 -00135 +00423 -00230
00002 00000024 -00400 +00384 -16732 +00048 -00136 +00258 -00131 +00419 -00230
00003 00000036 -00436 +00404 -16780 +00050 -00135 +00258 -00134 +00416 -00226
00004 00000048 -00464 +00400 -16744 +00050 -00135 +00256 -00137 +00417 -00226
00005 0000005A -00392 +00360 -16664 +00053 -00135 +00256 -00129 +00409 -00228
00006 0000006C -00464 +00336 -16716 +00052 -00135 +00255 -00137 +00419 -00224
00007 0000007E -00468 +00388 -16572 +00053 -00135 +00256 -00137 +00419 -00224
00008 00000090 -00432 +00416 -16668 +00052 -00133 +00255 -00137 +00419 -00224
00009 000000A2 -00412 +00416 -16616 +00052 -00136 +00255 -00137 +00425 -00220
00010 000000B4 -00456 +00396 -16540 +00051 -00138 +00256 -00133 +00423 -00228
00011 000000C6 -00388 +00352 -16600 +00052 -00137 +00258 -00135 +00409 -00218
00012 000000D8 -00472 +00408 -16560 +00050 -00133 +00256 -00139 +00417 -00226
00013 000000EA -00376 +00380 -16704 +00050 -00134 +00256 -00133 +00419 -00226
00014 000000FC -00444 +00396 -16708 +00051 -00135 +00255 -00129 +00419 -00219
00015 0000010E -00416 +00388 -16640 +00053 -00136 +00254 -00131 +00419 -00231
00016 00000120 -00396 +00352 -16748 +00053 -00134 +00253 -00133 +00425 -00229
00017 00000132 -00400 +00372 -16620 +00054 -00133 +00251 -00141 +00415 -00231
00018 00000144 -00460 +00380 -16680 +00053 -00136 +00251 -00137 +00415 -00223
00019 00000156 -00408 +00396 -16736 +00055 -00135 +00252 -00143 +00419 -00221
00020 00000168 -00432 +00432 -16744 +00055 -00133 +00250 -00135 +00417 -00219
00021 0000017A -00396 +00424 -16652 +00056 -00133 +00252 -00141 +00419 -00229
00022 0000018C -00432 +00400 -16636 +00056 -00133 +00254 -00139 +00421 -00223
00023 0000019E -00436 +00360 -16736 +00056 -00132 +00254 -00139 +00413 -00231
00024 000001B0 -00356 +00400 -16664 +00055 -00133 +00257 -00140 +00424 -00223
00025 000001C2 -00432 +00308 -16648 +00052 -00132 +00256 -00136 +00424 -00223
00026 000001D4 -00460 +00400 -16712 +00050 -00133 +00255 -00143 +00421 -00225
```

Figura 4.3: Início da representação dos dados obtidos pelo MPU (delimitador #[m)

Os dados exibidos em seguida, conforme mostrado na Figura 4.4, são do GPS e dados de calibração, assim como mencionado anteriormente na seção 3.2, dando fim ao registro completo feito pelo dispositivo.

```
g]#
#[l
21331
21331
20046
20046
83
58400
100
5
2
250
8
-08553 +00318 -22759 +04781 +04102 +03344 +03437
100
5
2
250
+00004 +00004 +00004 +01000 +01000 +01000
43740
232576
6039
+21331 +20046 +20046 +20046 +20046 +20046
20046
43722

l]#
#[f
+21331
24/07/20
Brasilia
9.8066501
9.7808437
01023
01020
113
100
5
2
250
256
-00385 +00354 -16530 +04308 +00046 -00113 +00258
4294868724 90868 4290735576 1103062 11994 4294938346 66172
-00372 +00352 -16508 +04286 +00047 -00111 +00254
-00384 +00404 -16424 +04333 +00048 -00115 +00258
+20046
-00103 +00071 -04117 +00047 -00113 +00260
+01702 +01828 -01952 +16178 +18701 +24195
+00015 +00014 +00015 +00006 +00005 +00024
-00018 -00018 -00002 +00293 -00579 +00159
+08242
f]#
```

Figura 4.4: Final do modo de Aquisição

Vale lembrar que durante todo este processo também se tem informações sendo exibidas no *display* LCD conectado à caixa preta, conforme exibido na Figura 4.5. O LCD vai exibindo e atualizando em tempo real a quantidade de informações obtidas com o MPU e GPS. Além disso, quando ocorre a transição para o modo de aquisição após pressionar o botão “SEL”, o *display* mostra o tempo restante para obtenção de dados. Isso pode ser observado na Figura 4.6.

Figura 4.5: LCD antes de pressionar o botão “SEL”

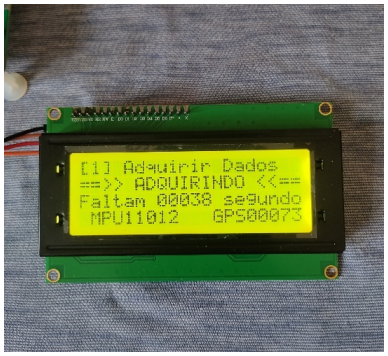

Figura 4.6: LCD após pressionar o botão “SEL”

4.2 ENSAIO COM O SUPERCAPACITOR

A seguir são apresentados os resultados de ensaios de carga do supercapacitor e sustentação do circuito da Caixa Preta. A Figura 4.7 apresenta a carga do super capacitor. Quando a alimentação foi ligada, a tensão no super capacitor era de 4V. Depois de 8 minutos ela chegou a 8,9 V. Pode parecer um período longo, entretanto, se for lembrado que o dispositivo será usado em um carro, que se espera que fique longos períodos ligado, não chega a ser um grande inconveniente. Certamente, a substituição de R2 por um resistor menor vai ajudar nisso. Por outro lado, uma carga de 8V já é suficiente para sustentar razoavelmente o circuito, o que demandaria muito menos tempo.

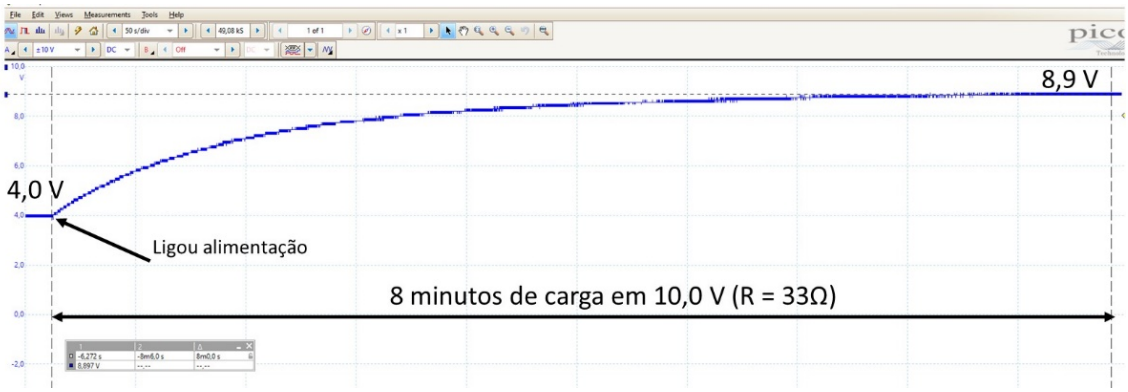

Figura 4.7: Gráfico da carga do super capacitor

Na etapa seguinte, se buscou verificar quanto tempo o super capacitor consegue sustentar a alimentação da Caixa Preta. Vale destacar que na parte do circuito composto pelos diodos D2, D3 e D4 foi utilizado um resistor de 33 Ω para o resistor R2, que está em série (conforme exibido na Figura 3.3). A Figura 4.8 abaixo mostra a descarga quando a alimentação foi desligada estando o super capacitor carregado com 9,0 V. Para este experimento, os 4 leds ficaram piscando, o que gera um grande consumo e, certamente, não aconteceria numa operação normal. A intenção foi a de se exagerar um pouco no consumo.

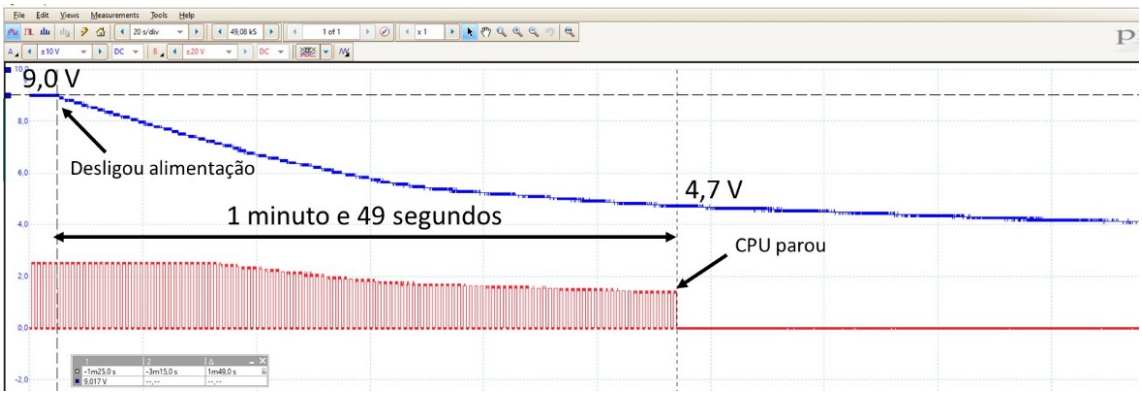

Figura 4.8: Gráfico da descarga do super capacitor enquanto sustenta a Caixa Preta

Pode-se ver que o circuito funcionou durante 109 segundos. Um período com certa folga para o que se pretende. É possível observar que o sinal em vermelho, gerado pela CPU, vai caindo de amplitude à medida em que a tensão do supercapacitor cai. Podemos ver que nos primeiros 40 segundos, a amplitude esteve bastante razoável, por isso, podemos tomar esse valor como uma faixa muito segura. Um outro dado é a tensão de 4,7 V, que parece ser o limite para a operação da Caixa Preta.

A Caixa Preta conta com uma memória SRAM de 256 KB para gravar os dados do carro. O GPS fornece um registro por segundo, enquanto o MPU envia 18 bytes numa taxa de 100 Hz. Em outras palavras, o MPU gera 100 vezes mais dados que o GPS. Vamos então considerar, somente para cálculo, que vamos preencher toda a memória SRAM apenas com dados do MPU. Seriam necessários 145 segundos, pois cada gravação consome 18 bytes (ax – ay – az – gx – gy – gz – hx – hy – hz), dois bytes por eixo. Porém, após um acidente, apenas metade da memória é preenchida, já que a outra metade é preenchida antes do evento. Então seriam necessários apenas 73 segundos. É preciso levar em conta a gravação na memória Flash. A memória empregada grava blocos de 128 bytes em 3 ms. Então, os 256 KB da SRAM são gravados em um pouco mais de 6 segundos. Concluindo, se o super capacitor sustentar o circuito por mais de 80 segundos, consegue-se atender à especificação mais severa.

5 CONCLUSÃO

Com este trabalho foi possível implementar a terceira versão da Caixa Preta e testar sua capacidade de obter dados inerciais, o que permitiu verificar seu potencial para que possa ser aplicada por profissionais que realizam a perícia de trânsito. Isso foi possível com a utilização de sensores e protocolos de comunicação específicos, além do Arduino, que proporcionou a programação de diversas funcionalidades. Foi verificado que o dispositivo apresentou o comportamento esperado ao gerar e armazenar todas as informações necessárias para a análise da movimentação do veículo, isto é, dados nos três eixos do acelerômetro, giroscópio e magnetômetro, bem como informações obtidas pelo módulo GPS.

De modo a se ter um maior controle sobre o hardware e suas funcionalidades, a utilização de uma interface gráfica se provou bem eficiente, pois facilita ainda mais a interação do usuário com a Caixa Preta, deixando mais claros os comandos que são passados bem como as informações que são retornadas. Ainda no quesito interação usuário-dispositivo, as teclas presentes na placa e o display LCD se mostraram bastante eficientes, pois se apresentam como uma boa alternativa para controle do dispositivo. Isso evidencia o bom funcionamento do algoritmo implementado, capaz de monitorar as teclas e tensões.

Do ponto de vista do circuito, também foi submetida a teste a capacidade da Caixa Preta de se manter funcionando mesmo após um evento que interrompa sua alimentação. Com os resultados, se constatou que a aplicação dos supercapacitores é suficiente para gerar dados em um tempo adequado, o que pode aumentar a riqueza de detalhes na análise pós-acidente. De modo geral, se pode afirmar que a forma como o circuito foi projetado permite que o dispositivo tenha uma boa performance e garanta que a CPU do Arduino funcione de forma satisfatória.

Um aspecto importante não só para este trabalho, como também para os trabalhos futuros, foi a documentação das funções utilizadas no algoritmo, disponibilizada no apêndice. Isso possibilitou uma maior compreensão de todas as ações desempenhadas pela Caixa Preta e também serve como referência, caso se deseje fazer uma consulta por funções específicas para avaliar os parâmetros de entrada e os resultados produzidos.

5.1 TRABALHOS FUTUROS

Como contribuições futuras, propõe-se uma forma de representar visualmente o trajeto e movimentação do carro, como uma animação, por exemplo. Desta forma, após a obtenção dos dados e aplicação dos mesmos em uma forma de representação visual, se reduziria bastante o tempo de análise, já que ficaria mais intuitivo e de fácil interpretação.

Também seria importante, assim que o programa estiver na sua versão definitiva, a implementação de uma forma de se medir o consumo. Além disso, é necessário submeter a interface CAN a mais testes, para verificar como o dispositivo se comporta ao ser alimentado pelo carro.

Nos testes verificou-se que ocorreram falhas no *self-test* do MPU, portanto seria fundamental realizar um debug nas funções responsáveis por esta ação para que esta funcionalidade desempenhe seu papel e favoreça a disponibilização dos resultados não só das leituras, como também dos *self-tests*.

REFERÊNCIAS BIBLIOGRÁFICAS

1 MORA, J. F. M. *Terminal Serial Python, PyQt5. Disponível em: <https://www.youtube.com/watch?v=4LfNUH039ls>. Códigos disponíveis em: <https://github.com/jsonfm/Simple-Serial-PyQt5>.*

2 CZERWONKA, M. *Acidentes de trânsito mataram 5.381 pessoas nas rodovias federais em 2021*. 2022. Disponível em: <https://www.portaldotransito.com.br/noticias/acidentes-de-transito-mataram-5-381-pessoas-nas-rodovias-federais-em-2021/>. Acesso em: Maio de 2022.

3 GONÇALVES, M. Ângelo. *A Perícia de Acidentes de Trânsito*. 2022. Disponível em: <http://www.peritodetransito.eng.br/blog/ver/1/a-pericia-de-acidentes-de-transito>. Acesso em: Agosto de 2022.

4 NETO, P. B. T. Relatório de trabalho de conclusão de curso. *Caixa Preta para carros: proposta para calibração, coleta e fusão de dados*. Universidade de Brasília, 2021.

5 NOGUEIRA, J. L. G. Relatório de trabalho de conclusão de curso. *Caixa Preta para Carros: Comparação de métodos de estimativa de inclinação usando acelerômetro, giroscópio e magnetômetro*. Universidade de Brasília, 2021.

6 LOPES, G. da S. Relatório de trabalho de conclusão de curso. *Caixa preta para veículos automotivos*. Universidade de Brasília, 2018.

7 RIBEIRO, M. R. Relatório de trabalho de conclusão de curso. *Projeto e Fabricação de Hardware para Caixa Preta Automotiva e Densímetro*. Universidade de Brasília, Abril de 2022.

8 MICROCHIP. Atmega640/v-1280/v-1281/v-2560/v-2561/v datasheet. 2014.

9 MPU-9250 Nine-Axis (Gyro + Accelerometer + Compass) MEMS MotionTracking™ Device. Disponível em: <https://invensense.tdk.com/products/motion-tracking/9-axis/mpu-9250/>. Acesso em: Julho de 2022.

10 DU, Y.; LIU, C.; WU, D.; JIANG, S. Measurement of international roughness index by using z-axis accelerometers and gps. 2014.

11 NIU, W.; FANG, L.; XU, L.; LI, X.; HUO, R.; GUO, D.; QI, Z. Summary of research status and application of mems accelerometers. Dezembro de 2018.

12 MEMS Accelerometer, Instrumentation Today. Disponível em: <http://www.instrumentationtoday.com/mems-accelerometer/2011/08/>. Acesso em: Julho de 2022.

13 XIA, D.; YU, C.; KONG, L. The development of micromachined gyroscope structure and circuitry technology. Janeiro de 2014.

14 ZUMBAHLEN, H. *Linear Circuit Design Handbook*. Newnes, Janeiro de 2008.

15 INVENSENSE. Mpu-9250 product specification revision 1.1.

16 PATEL, G.; CHUDASAMA, D. *Investigate the Origin of the Hall Effect*. Julho de 2021.

17 KONVALIN, C. *Compensating for Tilt, Hard-Iron, and Soft-Iron Effects*. Disponível em: <https://www.fierceelectronics.com/components/compensating-for-tilt-hard-iron-and-soft-iron-effects>. Acesso em: Agosto de 2022.

18 NEO-6 series Versatile u-blox 6 GPS modules. Disponível em: <https://www.u-blox.com/en/product/neo-6-series>. Acesso em: Agosto de 2022.

19 UBLOX. u-blox 6 gps modules data sheet. document number gps.g6-hw-09005-e.

20 CODREY Electronics, UART Communication Protocol – How it works? Disponível em: <https://www.codrey.com/embedded-systems/uart-serial-communication-rs232/>. Acesso em: Agosto de 2022.

21 ZELENOVSKY, R. *Livro do Microcontrolador MSP430 da Texas Instruments (Ainda em produção)*.

22 CIRCUIT Basics, BASICS OF THE I2C COMMUNICATION PROTOCOL. Disponível em: <https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/>. Acesso em: Agosto de 2022.

23 DHAKER, P. Analog dialogue. introduction to spi interface. Setembro de 2018.

24 CIRCUIT Basics, BASICS OF THE SPI COMMUNICATION PROTOCOL. Disponível em: <https://www.circuitbasics.com/basics-of-the-spi-communication-protocol>. Acesso em: Agosto de 2022.

25 INSTRUMENTS, T. *Introduction to the Controller Area Network (CAN), 2016. SLOA101B–August 2002–Revised May 2016*.

26 JANOTA, A.; ŠIMáK, V.; NEMEC, D.; HRBčEK, J. Improving the precision and speed of euler angles computation from low-cost rotation sensor data. Março de 2015.

27 REVUELTA Álvaro L. Orientation estimation and movement recognition using low cost sensors. Juho de 2017.

28 ARDUINO Mega 2560 Rev3, Arduino Store. Disponível em: <https://store-usa.arduino.cc/products/arduino-mega-2560-rev3>. Acesso em: Agosto de 2022.

29 ZELENOVSKY, R. *Arduino Embarcado em um Carro*. Disponível em: <https://labdegaragem.com/profiles/blogs/arduino-embarcado-em-um-carro>. Acesso em: Agosto de 2022.

30 BATTERY University, How does a Supercapacitor Work? Disponível em: <https://batteryuniversity.com/article/bu-209-how-does-a-supercapacitor-work>. Acesso em: Agosto de 2022.

31 CONHEçA o TVS (Transient Voltage Suppressor), Instituto Newton C Braga. Disponível em: <https://www.newtoncbraga.com.br/index.php/novos-componentes/52-artigos-tecnicos/artigos-diversos/10745-conheca-o-tvs-transient-voltage-suppressor-art253>. Acesso em: Agosto de 2022.

32 U-CENTER GNSS evaluation software for Windows. Disponível em: <https://www.u-blox.com/en/product/u-center>. Acesso em: Setembro de 2022.

33 U-CENTER 2 user guide. Disponível em: <https://www.u-blox.com/en/info/u-center-2-user-guide>. Acesso em: Setembro de 2022.

Apêndice A
Coleção de Funções utilizadas pela Caixa Preta

A. COLEÇÃO DE FUNÇÕES UTILIZADAS PELA CAIXA PRETA

A.1 #TWI - BIBLIOTECA PARA O TWI (I2C)

A variável twi_busy indica se o TWI está ocupado.

Usam o TWI: LCD, 24LC1025 (Flash), MPU9250

Tabela A.1: Funções - TWI

byte	twi_er_ok	(byte adr)
byte	twi_er_check	(byte ee, byte ix)
void	twi_scan	(void)
void	twi_config_400k	(void)
void	twi_config_100k	(void)
void	void twi_start	(byte ix)
void	void twi_start_rep	(int ix)
void	twi_stop	(void)
void	twi_er	(byte eer, byte ix)
void	twi_et	(byte eet, byte ix)
void	twi_dado_er	(byte dado, byte ix)
byte	twi_dado_et_ack	(byte ix)
byte	byte twi_dado_et_nack	(byte ix)
byte	byte espera_twint	(byte ix)
void	twi_erro	(int cod, int ix)

• byte twi_er_ok (byte adr)
Verificar se tem resposta do escravo receptor no endereço “adr”.
É semelhante à próxima.
TRUE se escravo respondeu com ACK.
FALSE se escravo respondeu com NACK.

• byte twi_er_check (byte ee, byte ix)
Enviar endereço de Escrita do Escravo Receptor (ER) e esperar ACK (ee = endereço do escravo).
É semelhante à anterior.
Retorna TRUE se escravo gerou ACK.
Retorna FALSE se gerou NACK (não gera msg de erro).

• void twi_scan (void)

Buscar por todos os dispositivos TWI para escrita. Imprime na serial o que encontrar.
Como busca por escrita, faz adr«1 = bbbb bbb0.

• **void twi_config_400k (void)**
Configurar TWI para 400k (frequência da linha SCL).

• **void twi_config_100k (void)**
Configurar TWI para 100k (frequência da linha SCL).

• **void twi_start (byte ix)**
Gerar um START no TWI.

• **void twi_start_rep (int ix)**
Gerar um START Repetido no TWI.

• **void twi_stop (void)**
Enviar STOP.

• **void twi_er (byte eer, byte ix)**
Enviar endereço de Escrita do Escravo (ER) e esperar ACK.
eer = endereço do escravo receptor.

• **void twi_et (byte eet, byte ix)**
Enviar endereço de Leitura do Escravo (ET) e esperar ACK.
eet = endereço do escravo transmissor.

• **void twi_dado_er (byte dado, byte ix)**
Enviar dado para escravo previamente endereçado.

• **byte twi_dado_et_ack (byte ix)**
Receber dado e gerar ACK.

• **byte twi_dado_et_nack (byte ix)**
Receber dado e gerar NACK.

• **byte espera_twint (byte ix)**
Esperar TWINT.

• **void twi_erro (int cod, int ix)**
ERROS no TWI (I2C).
1 = Erro ao gerar START.

2 = Erro ao gerar START Repetido.
3 = Erro Escravo Receptor endereçado (ER) não enviou ACK.
4 = Erro Escravo Transmissor endereçado (ET) não enviou ACK.
5 = Erro Escravo Receptor (ER) não enviou ACK após envio do dado.
6 = Erro ao receber um dado do Escravo Transmissor (ET) e gerar um ACK.
7 = Erro ao receber um dado do Escravo Transmissor (ET) e gerar um NACK.
8 = Erro ao esperar TWINT - Timeout esperando TWINT ir para 1.

A.2 #LCD - BIBLIOTECA PARA O LCD (I2C OU TWI)

Tabela A.2: Funções - LCD

void	lcd_procura	(void)
void	lcd_refresh	(byte lin)
void	lcd_inic	(void)
void	lcd_aux	(byte dado, byte ix)
void	lcd_pcf_wr	(byte dado)
void	lcd_gps_data	(byte lin, byte col)
void	lcd_gps_hora	(byte lin, byte col)
void	lcd_gps_lat	(byte lin, byte col)
void	lcd_gps_long	(byte lin, byte col)
void	lcd_gps_vel_kph	(byte lin, byte col)
void	lcd_gps_dado	(byte lin, byte col, byte qual)
void	lcd_float	(byte lin, byte col, float fx, byte prec)
void	lcd_dec32	(byte lin, byte col, long dt)
void	lcd_dec32u	(byte lin, byte col, long dt)
void	lcd_dec32nz	(byte lin, byte col, long dt)
void	lcd_dec32unz	(byte lin, byte col, long dt)
void	lcdv_dec32unz	(byte lin, byte col, long dt)
void	lcd_hex32	(byte lin, byte col, long dt)
void	lcd_dec16	(byte lin, byte col, int dt)
void	lcd_dec16u	(byte lin, byte col, word dt)
void	lcd_dec16nz	(byte lin, byte col, int dt)
void	lcd_dec16unz	(byte lin, byte col, word dt)
void	lcd_hex16	(byte lin, byte col, byte dt)
void	lcd_dec8	(byte lin, byte col, byte dt)
void	lcd_dec8u	(byte lin, byte col, byte dt)
void	lcd_dec8nz	(byte lin, byte col, byte dt)
void	lcd_dec8unz	(byte lin, byte col, byte dt)
void	lcd_hex8	(byte lin, byte col, byte dt)
void	lcd_spc	(byte lin, byte col, byte qtd)
void	lcd_str	(byte lin, byte col, char *pt)
void	lcd_char	(byte lin, byte col, char dt)
void	lcd_apaga_lin	(byte lin)
void	lcd_apaga	(void)
void	lcdt_str	(byte *pt)
void	lcdt_char	(byte dt)
void	lcdt_cursor	(byte lin, byte col)
void	lcdt_cmdo	(byte cmdo)

• void lcd_procura (void)
Procura pelo LCD.

• void lcd_refresh (byte lin)
Faz o refresh de uma linha do LCD, ou seja, envia uma linha do lcd_buf para o LCD. Em geral é chamada de dentro da interrupção.

Posiciona o cursor na linha correta e transfere todos os dados.

• void lcd_inic (void)
Faz a inicialização do LCD e apaga o lcd_buf. Usa a função lcd_aux().

• void lcd_aux (byte dado, byte ix)
Usado pela função lcd_inic(). Envia um nibble para PCF8574.

• void lcd_pcf_wr (byte dado, byte ix)
Escrever um dado no PCF8574. Gera START, envia o dado e depois gera o STOP.

• void lcd_gps_data (byte lin, byte col)
Imprimir data (DD/MM/AA) do GPS no LCD a partir de (lin,col).

• void lcd_gps_hora (byte lin, byte col)
Imprimir HH:MM:SS no LCD a partir de (lin,col).

• void lcd_gps_lat (byte lin, byte col)
Imprimir Latitude DD MM.MMMMM N/S no LCD a partir de (lin,col).

• void lcd_gps_long (byte lin, byte col)
Imprimir Longitude DDD MM.MMMMM E/W no LCD a partir de (lin,col).

• void lcd_gps_vel_kph (byte lin, byte col)
Imprimir velocidade kph a partir de (lin,col).

• void lcd_gps_dado (byte lin, byte col, byte qual)
Imprimir um determinado dado do GPS. Argumento "qual"indica a posição no vetor gps_dados.

• void lcd_float (byte lin, byte col, float f, byte prec)
No Arduino, double e float têm a mesma precisão
Escreve no LCD (posição lin,col) o float "f"com "prec"casas após a vírgula e apresenta o sinal.
Formato = + xxx xxx xxx , ddd ddd ddd ddd (usar char msg[24])
Limite da parte inteira = 9 dígitos. Limite da parte fracionária = 12 dígitos.
Caso ultrapasse esses limites imprime ###, ###
O máximo é 999.999.999,999999. Se ultrapassar o máximo, escreve ###,###.
Na verdade, o máximo é 999.999.999,999967. Exemplos:
999.999.999,0 -> imprime 999.999.936,000000 (por causa da precisão da representação)
876.543.210,123456789 -> imprime 876543232,000000 (por causa da precisão da representação)

• **void lcd_dec32 (byte lin, byte col, long dt)**
Escrever no LCD (posição lin,col) 32 bits em decimal, com sinal e com zeros à esquerda.

• **void lcd_dec32u (byte lin, byte col, long dt)**
Escrever no LCD (posição lin,col) 32 bits em decimal, sem sinal e com zeros à esquerda.

• **void lcd_dec32nz (byte lin, byte col, long dt)**
Escrever no LCD (posição lin,col) 32 bits em decimal, com sinal e sem zeros à esquerda.

• **void lcd_dec32unz (byte lin, byte col, long dt)**
Escrever no LCD 32 (posição lin,col) bits em decimal, sem sinal e sem zeros à esquerda.

• **void lcd_hex32 (byte lin, byte col, long dt)**
Escrever no LCD (posição lin,col) 32 bits em hexadecimal.

• **void lcd_dec16 (byte lin, byte col, word dt)**
Escrever no LCD (posição lin,col) 16 bits em decimal, com sinal e com zeros à esquerda.

• **void lcd_dec16u (byte lin, byte col, word dt)**
Escrever no LCD (posição lin,col) 16 bits em decimal, sem sinal e com zeros à esquerda.

• **void lcd_dec16nz (byte lin, byte col, word dt)**
Escrever no LCD (posição lin,col) 16 bits em decimal, com sinal e sem zeros à esquerda.

• **void lcd_dec16unz (byte lin, byte col, word dt)**
Escrever no LCD (posição lin,col) 16 bits em decimal, sem sinal e sem zeros à esquerda.

• **void lcd_hex16 (byte lin, byte col, word dt)**
Escrever no LCD (posição lin,col) 16 bits em hexadecimal.

• **void lcd_dec8 (byte lin, byte col, byte dt)**
Escrever no LCD (posição lin,col) 8 bits em decimal, com sinal e com zeros à esquerda.

• **void lcd_dec8u (byte lin, byte col, byte dt)**
Escrever no LCD (posição lin,col) 8 bits em decimal, sem sinal e com zeros à esquerda.

• **void lcd_dec8nz (byte lin, byte col, byte dt)**
Escrever no LCD (posição lin,col) 8 bits em decimal, com sinal e sem zeros à esquerda.

• **void lcd_dec8unz (byte lin, byte col, byte dt)**
Escrever no LCD (posição lin,col) 8 bits em decimal, sem sinal e sem zeros à esquerda.

• **void lcd_hex8 (byte lin, byte col, byte dt)**
Escrever no LCD (posição lin,col) 8 bits em hexadecimal.

• **void lcd_spc (byte lin, byte col, byte qtd)**
Escrever uma qtd de espaços a partir da posição (lin,col)

• **void lcd_str (byte lin, byte col, char *msg)**
Escrever uma string a partir da posição (lin,col)

• **void lcd_char (byte lin, byte col, byte dt)**
Escrever um char, na posição (lin,col)

• **void lcd_apaga_lin (byte lin)**
Escrever brancos na linha indicada (no lcd_buf).

• **void lcd_apaga (void)**
Apagar todo o LCD - Escreve brancos (no lcd_buf).

• **void lcdt_str (byte *msg)**
Modo Terminal. Não usa o buffer do LCD. Escreve direto no LCD. Escreve uma string no LCD. Antes, deve-se posicionar o cursor.

• **void lcdt_char (byte x)**
Modo Terminal. Não usa o buffer do LCD. Escreve direto no LCD.
Escreve um caractere no LCD. Antes, deve-se posicionar o cursor.

• **void lcdt_cursor (byte lin, byte col)**
Modo Terminal. Não usa o buffer do LCD. Escreve direto no LCD.
Posiciona o cursor.

• **void lcdt_cmdo (byte cmdo)**
Modo Terminal. Não usa o buffer do LCD. Escreve direto no LCD. Envia um comando para o LCD.

Sobre o LCD:

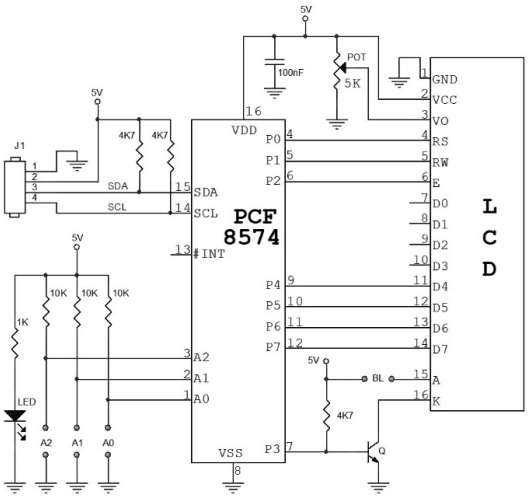

Figura A.1: Esquemático do PCF8574 e LCD

P7	P6	P5	P4	P3	P2	PC1	PC0
D7	D6	D5	D4	BL	E	R/#W	RS

```
#define NRL 4    //Quantidade de linhas do LCD, numeradas de 0, 1, 2, 3
#define NRC 20   //Quantidade de colunas do LCD, numeradas de 0, 1, 2, ..., 20
```

Acesso ao LCD SEMPRE é feito via um buffer.
A escrita no LCD é lenta e há a incerteza de quanto tempo que se gasta esperando o BUSY = 0. Para ficar mais rápido e facilitar o acesso ao LCD, foi criada a matriz **lcd_buf [NRL] [NRC]**. As escritas acontecem nesse buffer, sendo que cada escrita marca, na variável **lcd_mudou**, a linha do LCD que foi alterada. O Timer 1 (100 Hz) monitora essa variável e chama o refresh de uma linha do LCD quando necessário. A cada 10ms somente uma linha pode passar pelo refresh.

Os bits da variável **lcd_mudou** indicam quais linhas necessitam ser atualizadas. Assim, se **lcd_mudou** = FALSE, significa que nenhuma linha precisa de alteração. Se **lcd_mudou** != FALSE, é preciso ver qual linha deve ser atualizada.

Lcd_mudou = 0000 0000 -> nenhuma linha para atualizar;
Lcd_mudou = 0000 0001 -> atualizar a linha 0
Lcd_mudou = 0000 0010 -> atualizar a linha 1
Lcd_mudou = 0000 0110 -> atualizar as linhas 1 e 2.

O Timer 1 sempre consulta **lcd_mudou** para saber se deve chamar a função que faz o refresh de uma linha. No Timer 1 busca usando a variável lcd_rr = 0, 1, 2 ou 3, que indica qual bit de **lcd_mudou** deve ser consultado, ou seja, qual linha deve ser verificada. Quando encontra o bit igual a 1, chama o refresh da linha correspondente. Note que a variável lcd_rr nunca é reinicializada, isto garante uma alternância entre as linhas, ou seja, evita se a linha o sempre seja a primeira a ser consultada.

Existe ainda variável lcd_busy que indica que uma atualização do LCD está acontecendo e que uma nova não deve ser iniciada.

A interrupção do Timer 1, (se lcd_busy = FALSE) chama a função lcd_linha_alterou(lcd_rr) indicando qual linha consultar. Caso retorne TRUE, habilita a interrupção do Timer 2.

Numa versão, a consulta era sempre feita da linha 0 até a última. Acontecia que se a linha 0 era alterada com uma frequência muito grande, as demais linhas nunca eram atualizadas. Para corrigir esse problema, se optou por uma ordem de verificação circular.

Abaixo está o trecho da rotina de interrupção do Timer 1 que busca por uma linha do LCD a ser atualizada. Ela sempre altera a sequência de busca. Notar o for() com NRL repetições, sendo lcd_rr incrementado a cada repetição. Ao sair do for, lcd_rr é novamente incrementada. Isso não foi feito na versão anterior. Essa versão sempre busca na ordem 0,1,2,3. A linha 0 tinha a maior prioridade. Assim, se a linha 0 mudava com muita frequência, as demais linhas não eram mostradas.

```
1  if (lcd_mudou){
2      if (lcd_busy==FALSE && twi_busy==FALSE){      //LCD disponivel e TWI dispon vel
3        for (i=0; i<NRL; i++){
4          if ((lcd_mudou&(1<<lcd_rr)) != 0) break;____//por aqui tem i<NRL
5          lcd_rr++;
6          lcd_rr &= 3;
7        }
8        if (i<NRL){ //i<NRL indica que saiu pelo if
9          lcd_mudou &= ~(1<<lcd_rr);  //apagar o bit que marcava a linha
10         lcd_refresh(lcd_rr);
11         lcd_rr++;
12         lcd_rr &= 3;
13       }
14     }
15   }
```

Variáveis usadas:
Byte **lcd_buf [NRL] [NRC+1];** -> matriz que funciona como buffer do LCD.

	(col=0)	(col=1)	(col=2)	...	...	(col=19)
Linha=0	A	B	C	...	...	D
Linha=1	E	F	G	...	...	H
Linha=2	I	J	K	...	...	L
Linha=3	M	N	O	...	...	P

Byte **lcd_mudou**; -> indicar qual linha precisa de atualização, faixa de 0, ..., 16.
• BIT0 = 1, (0000 0001) -> linha 0 necessita de atualização
• BIT1 = 1, (0000 0010) -> linha 1 necessita de atualização
• BIT2 = 1, (0000 0100) -> linha 2 necessita de atualização
• BIT3 = 1, (0000 1000) -> linha 3 necessita de atualização

A.3 #SW - BIBLIOTECA PARA AS TECLAS

Tabela A.3: Funções - SW

void	sw_qq_tecla	(void)
void	sw_fila_limpa	(void)
byte	sw_tira	(byte *cha)
void	sw_config	(void)
void	sw_ler	(byte val)
void	sw_busca_seq1	(byte val)
void	sw_busca_seq2	(byte val)
byte	sw_poe	(byte cha)

• void sw_qq_tecla (void)
Espera pelo acionamento de uma tecla qualquer. Retorna quando conseguir retirar algo da fila. Ignora o que for lido. Não imprime qualquer tipo de mensagem.

• void sw_fila_limpa (void)
Limpar fila do teclado. Joga tudo fora.

• void sw_ler (byte val)
Analisa como foi o passado e, se for o caso, coloca o código da tecla na fila do teclado.

• void sw_busca_seq1 (byte val)
Todo código de tecla aceita que é colocado na fila circular é passado para essa função.
Ela tenta identificar a Sequência 1 (SEQ1 = ESQ, SUP, DIR). O código de SEQ1 é colocado na fila logo após o código da última tecla (tecla DIR) que o caracterizou.

• void sw_busca_seq2 (byte val)
Todo código de tecla aceita que é colocado na fila circular é passado para essa função.

Ela tenta identificar a Sequência 2 (SEQ2 = ESQ, INF, DIR). O código de SEQ2 é colocado na fila logo após o código da última tecla (tecla DIR) que o caracterizou.

• **void sw_config (void)**
Configura o ADC e a fila circular para as chaves.

• **byte sw_poe (byte cha)**
Colocar na fila circular o código de uma chave. Retorna
TRUE: se conseguiu colocar o código na fila circular das chaves.
FALSE: se a fila estava cheia e então não conseguiu colocar o código na fila circular.

• **byte sw_tira (byte *cha)**
Retira da fila o código da próxima chave. Retorna:
TRUE: se conseguir retirar um código da fila e esse código é copiado em *cha.
FALSE: se a fila estava vazia e então não conseguir retirar um código da fila circular.

Tabela A.4: Disposição da Teclas

	SW_CIMA	
SW_ESQ	SW_SEL	SW_DIR
	SW_BAIXO	

Temos duas sequências especiais:
SW_SEQ1: SW_ESQ, SW_CIMA, SW_DIR
SW_SEQ2: SW_ESQ, SW_BAIXO, SW_DIR

Cada chave (tecla) tem um código que a identifica:

```
1 #define  SW_SEL     0
2 #define  SW_ESQ     1
3 #define  SW_DIR     2
4 #define  SW_SUP     3
5 #define  SW_INF     4
6 #define  SW_SEQ1    5
7 #define  SW_SEQ2    6
8 #define  SW_NADA    7
9 #define  SW_NAOSEI  8
```

Está disponível um vetor de ponteiros que permite acesso aos nomes de 3 letras para as teclas:

```
1 // Teclas - Nome das teclas com apenas 3 letras
```

```
//                    0      1      2      3      4      5      6      7      8
char *sw_nome[]={"SEL","ESQ", "DIR","SUP","INF", "SQ1", "SQ2", "NAD", "???"};//
```

Variáveis globais usadas pelo teclado:

```
volatile char sw_fila[SER_TX_FILA_TAM]; //Espaco para a fila teclado
volatile byte sw_pin, sw_pout;          //Ponteiros para usar a fila
volatile byte sw_1,sw_2,sw_n,sw_v;      //Vari veis para detectar teclas
    acionadas
volatile byte sw_st_seq1,sw_st_seq2;    //Maq Estados para buscar sequ ncias SEQ1
    e SEQ2
```

Era para ser usado um conjunto particular de resistores para a lógica das teclas. Não foi usado. Então, o teclado está com a seguinte configuração:

```
// Limiares para as chaves (NADA = chaves soltas)
// Teste se valor acima de limiar tal, chave tal
// 0x00     0x27     0x6F     0xB4     0xE1     0xF3     0xFF
//    | SEL  | ESQ  | SUP  | DIR  | INF  | NADA  |
#define LIM_NADA 0xF3
#define LIM_INF  0xE1
#define LIM_DIR  0xB4
#define LIM_SUP  0x6F
#define LIM_ESQ  0x27
#define LIM_SEL  0x00
```

O Timer 1 é o responsável por ler o ADC. A cada 2 leituras, calcula a média, guarda em **sw_val** e chama a função **sw_ler(sw_val)**. Ver fluxograma logo adiante. Esta função conhece o passado do teclado e, se for o caso, valida a nova tecla colocando o seu código na fila circular. Uma nova tecla só é aceita se o teclado passou anteriormente pelo estado de nenhuma tecla acionada.

O Timer 1 está programado para interromper a cada 10 ms. Porém, ele não faz a leitura do ADC toda vez. Ele tem outras funções, de acordo com a tabela abaixo. A consulta do ADC para o teclado é feita, aproximadamente, em 37,5 Hz (100*(12/32)), ou seja, uma leitura a cada 26,6 ms.

Tabela A.5: Monitoramento das Chaves e Tensões

0) ADC Start	8) +Ler, ADC start*	16) ADC Start	24) +Ler, ADC start*
1) Ler, ADC start	9) Ler, ADC start	17) Ler, ADC start	25) Ler, ADC start
2) +Ler, ADC start*	10) +Ler, ADC start*	18) +Ler, ADC start*	26) +Ler, ADC start*
3) Ler, ADC start	11) Ler, ADC start	19) Ler, ADC start	27) Ler, ADC start
4) +Ler, ADC start*	12) +Ler, Canal1(VCAR)*	20) +Ler, ADC start*	28) +Ler, Canal 2(VCAP)*
5) Ler, ADC start	13) ADC Start	21) Ler, ADC start	29) ADC Start
6) +Ler, ADC start*	14) Ler, ADC start	22) +Ler, ADC start*	30) Ler, ADC start
7) Ler, ADC start	15) +Ler, Canal 0	23) Ler, ADC start	31) +Ler, Canal 0

* indica a fase para tirar a média na leitura do teclado.

Lógica para detectar teclas acionadas

sw_1 e sw_2 -> garantir duas leituras iguais, evita transitórios;
sw_n -> nova tecla
sw_v -> tecla velha
Tecla é considerada válida quando duas leituras seguidas são iguais: sw_1=sw_2
Neste caso sw_n recebe o código da tecla.
Se sw_n<>sw_v e sw_v = NADA, nova tecla é colocada no buffer e depois sw_v=sw_n.

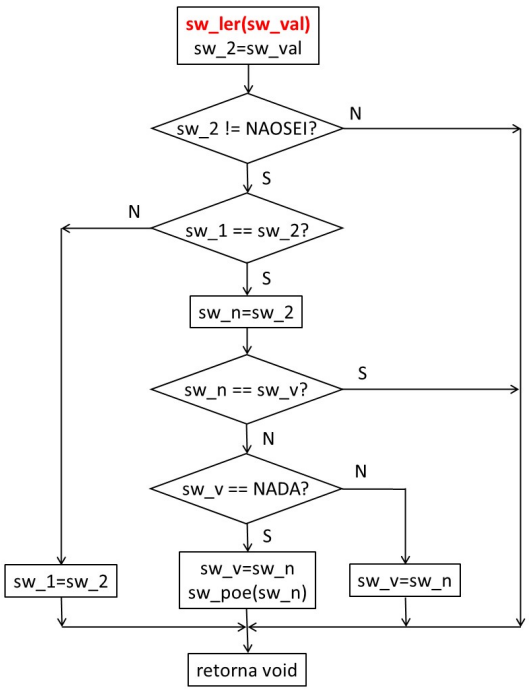

Figura A.2: Lógica para detecção das teclas acionadas.

Tabela A.6: Gabaritos para configurar ADC (8 bits alinhado pela esquerda, ler apenas ADCH)

	7	6	5	4	3	2	1	0
ADMUX	REFS1 0,1,1	REFS0 1,1,1	ADLAR 1	MUX4 0	MUX3 0	MUX2 0	MUX1 0,0,1	MUX0 0,1,0
ADCSRA	ADEN 1	ADSC 0	ADATE 1	ADIF 0	ADIE 0	ADPS2 0	ADPS1 1	ADPS0 1
ADCSRB	- -	ACME 0	- -	- -	MUX5 0	ADST2 1	ADST1 0	ADST0 1

Tabela A.7: Configuração para os diversos canais

Analógico	Canal ADC	MUX5:0	REFS1:0	REF volts
Teclado	0	0 0 0 0 0	0 1	AVCC
Tensão VCAR	1	0 0 0 0 1	1 1	2,56 V
Tensão VCAP	2	0 0 0 1 0	1 1	2,56 V

Teclado: ADMUX = (1«REFS0) | (1«ADLAR); //Ref = AVCC
VCAR: ADMUX = (1«REFS1) | (1«REFS0) | (1«ADLAR) | (1«MUX0); //Canal 1, Ref = 2,56
VCAP: ADMUX = (1«REFS1) | (1«REFS0) | (1«ADLAR) | (1«MUX1); //Canal 2, Ref = 2,56

Cálculo dos resistores do divisor resistivo do teclado
Observação: Somente como ilustração. Não foi feito exatamente desta forma

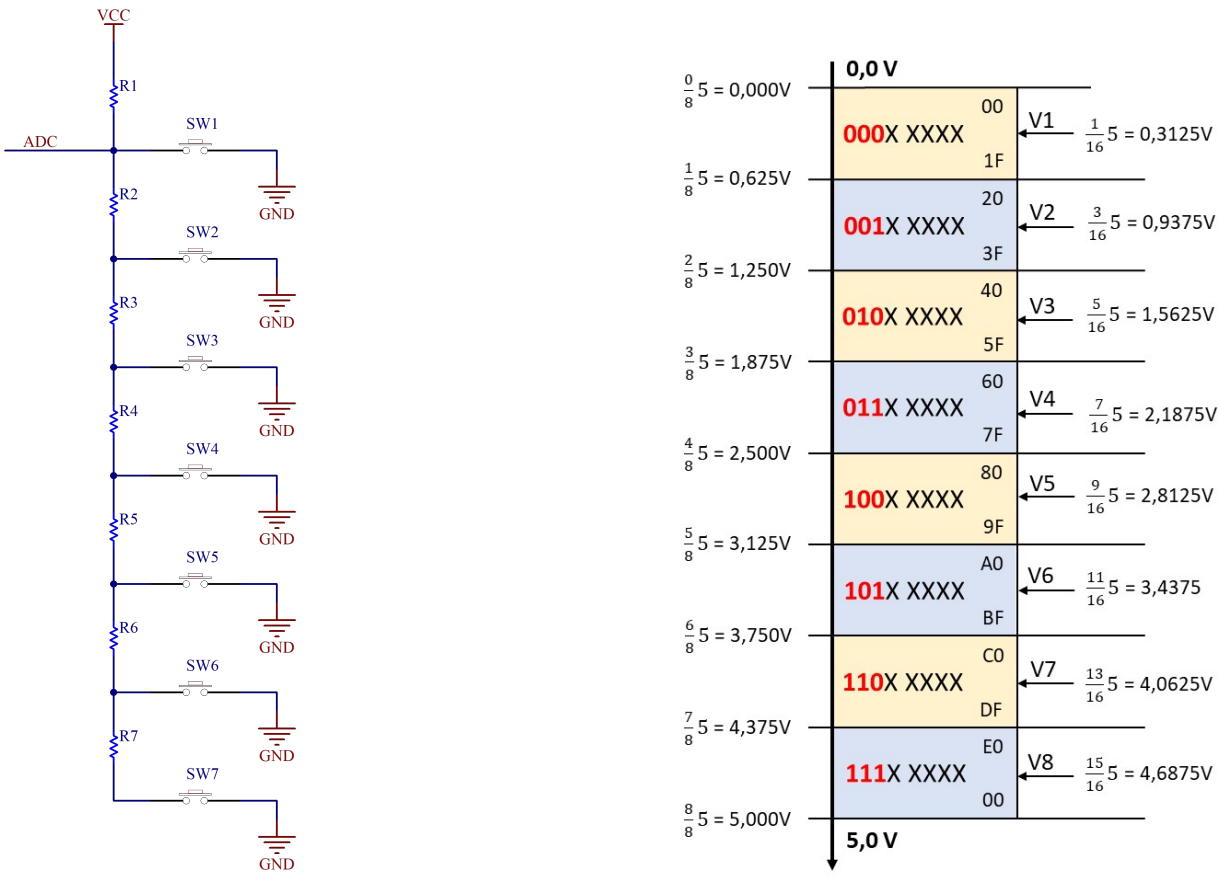

Figura A.3: Ilustração do circuito (direita) e ilustração numérica (esquerda)

$$V1 = 0$$

$$V2 = VCC\frac{R2}{R1+R2} \rightarrow (VCC - V2)R2 - V2 \cdot R1 = 0$$

$$V3 = VCC\frac{R2+R3}{R1+R2+R3} \rightarrow (VCC - V3)R3 + (VCC - V3)R2 - V3 \cdot R1 = 0$$

$$V4 = VCC\frac{R2+R3+R4}{R1+R2+R3+R4} \rightarrow (VCC - V4)R4 + (VCC - V4)R3 + (VCC - V4)R2 - V4 \cdot R1 = 0$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & VCC - V2 & -V2 \\ 0 & 0 & 0 & 0 & VCC - V3 & VCC - V3 & -V3 \\ 0 & 0 & 0 & VCC - V4 & VCC - V4 & VCC - V4 & -V4 \\ 0 & 0 & Vcc - V5 & Vcc - V5 & Vcc - V5 & Vcc - V5 & -V5 \\ 0 & Vcc - V6 & Vcc - V6 & Vcc - V6 & Vcc - V6 & Vcc - V6 & -V6 \\ Vcc - V7 & Vcc - V7 & Vcc - V7 & Vcc - V7 & Vcc - V7 & Vcc - V7 & -V7 \end{bmatrix} \cdot \begin{bmatrix} R7 \\ R6 \\ R5 \\ R4 \\ R3 \\ R2 \\ R1 \end{bmatrix} = \begin{bmatrix} 10K \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Resposta:

	Calculados	Comerciais
R1	10.000,00	10.000
R2	2.307,69	2.400
R3	2.237,76	2.200
R4	3.232,32	3.300
R5	5.079,37	5.100
R6	9.142,86	9.100
R7	21.333,33	22.000

Resistores Comerciais

1.0 Ω	1.1 Ω	1.2 Ω	1.3 Ω
1.5 Ω	1.6 Ω	1.8 Ω	2.0 Ω
2.2 Ω	2.4 Ω	2.7 Ω	3.0 Ω
3.3 Ω	3.6 Ω	3.9 Ω	4.3 Ω
4.7 Ω	5.1 Ω	5.6 Ω	6.2 Ω
6.8 Ω	7.5 Ω	8.2 Ω	9.1 Ω

$$V0 = 0$$

$$V1 = VCC\frac{R1}{R0 + R1}$$

$$V2 = VCC\frac{R1 + R2}{R0 + R1 + R2}$$

$$V3 = VCC\frac{R1 + R2 + R3}{R0 + R1 + R2 + R3}$$

$$V4 = VCC\frac{R1 + R2 + R3 + R4}{R0 + R1 + R2 + R3 + R4}$$

$$V5 = VCC\frac{R1 + R2 + R3 + R4 + R5}{R0 + R1 + R2 + R3 + R4 + R5}$$

$$V6 = VCC\frac{R1 + R2 + R3 + R4 + R5 + R6}{R0 + R1 + R2 + R3 + R4 + R5 + R6}$$

$$V7 = VCC\frac{R1 + R2 + R3 + R4 + R5 + R6 + R7}{R0 + R1 + R2 + R3 + R4 + R5 + R6 + R7}$$

	R	V	Hexa		R	V	Hexa		3 Bits
1	10.000,00	0,0	0		10.000	0,0	00		000
2	2.307,69	0,9375	30		2.400	0,9677	32		001
3	2.237,76	1,5625	50		2.200	1,5753	51		010
4	3.232,32	2,1875	70		3.300	2,2067	71		011
5	5.079,37	2,8125	90		5.100	2,8261	91		100
6	9.142,86	3,4375	B0		9.100	3,4424	B0		101
7	21.333,33	4,0625	D0		22.000	4,0758	D1		110

Aproximação para a Caixa Preta que só tem 5 chaves:

	R1	R1	R1	R1	R1	R1	R1
Comerciais	10.000	2.400	2.200	3.300	5.100	9.100	22.100
Aprox. 5 chaves	10.000	4.700	4.700	3.300	5.600	33.000	33.000
Nr no PCB	R3	R4	R4	R5	R6	R7	R7

Com os valores de resistores da tabela, são geradas as tensões abaixo.

SEL -> SW3 -> adc = 0V -> 0d -> 0x00 -> 0000 0000
ESQ -> SW1 -> adc =1,598V -> 81d -> 0x51 -> 0101 0001
SUP -> SW2 -> adc =2,22V -> 113d -> 0x71 -> 0111 0001
DIR -> SW5 -> adc =2,88V -> 147d -> 0x93 -> 1001 0011
INF -> SW4 -> adc =4,11V -> 210d -> 0xD2 -> 1100 0010

A.4 #LEDS - BIBLIOTECA PARA OS LEDS

Tabela A.8: LEDS: Sequência VD – AM – AZ – VM

	Led
(22) PA.0	Amarelo
(23) PA.1	Vermelho
(24) PA.2	Azul
(13) PJ.7	Verde
(7) PH4	Scope 1
(8) PH5	Scope 2

Tabela A.9: Funções - LEDS

void	leds_cont	(byte ct)
void	leds_config	(void)
void	desligar	(void)
void	led_vd	(void)
void	led_VD	(void)
void	led_Vd	(void)
void	led_az	(void)
void	led_AZ	(void)
void	led_Az	(void)
void	led_am	(void)
void	led_AM	(void)
void	led_Am	(void)
void	led_vm	(void)
void	led_VM	(void)
void	led_Vm	(void)
void	scp1	(void)
void	SCP1	(void)
void	Scp1	(void)
void	scp2	(void)
void	SCP2	(void)
void	Scp2	(void)
void	scope_config	(void)

• **void leds_cont (byte ct)**
Apresenta o argumento em binário nos 4 leds: Laranja - Verde - Amarelo - Vermelho

• **void leds_config (void)**
Configurar pinos dos leds. Configura também PORTB.6 para fazer o liga/desliga.

• **void desligar (void)**
Desligar a Caixa preta, desliga Q2.

• **void led_vd (void)** -> VD = Apagado

• **void led_VD (void)** -> VD = Aceso

• **void led_Vd (void)** -> VD = Invertido
VD (PJ7) = Verde: Apagar / Acender / Inverter

• **void led_az (void)** -> VD = Apagado

• **void led_AZ (void)** -> VD = Aceso

• **void led_Az (void)** -> VD = Invertido
AZ (PA2) = Azul: Apagar / Acender / Inverter

• **void led_am (void)** -> AM = Apagado

• **void led_AM (void)** -> AM = Aceso

• **void led_Am (void)** -> AM = Invertido
AM (PA0) = Amarelo: Apagar / Acender / Inverter

• **void led_vm (void)** -> VM = Apagado

• **void led_VM (void)** -> VM = Aceso

• **void led_Vm (void)** -> VM = Invertido
VM (PA1) = Vermelho: Apagar / Acender / Inverter

• **void scp1 (void)** -> SCP1 = Apagado

• **void SCP1 (void)** -> SCP1 = Aceso

• **void Scp1 (void)** -> SCP1 = Invertido
Pino 7 = SCP1 (PH4) = Scope 1: Apagar / Acender / Inverter

• **void scp2 (void)** -> SCP2 = Apagado

• **void SCP2(void)** -> SCP2 = Aceso

• **void Scp2 (void)** -> SCP2 = Invertido
Pino 8 = SCP2 (PH5) = Scope 2: Apagar / Acender / Inverter

• **void scope_config (void)**
Configurar pinos para Osciloscópio

A.5 #TIMERS - BIBLIOTECA PARA OS TIMERS

Tabela A.10: Funções - TIMERS

void	timer1_config	void
void	timer2_config	void
byte	lcd_linha_alterou	(byte qual)
	ISR(TIMER1_COMPA_vect)	
	ISR(TIMER2_COMPA_vect)	

• void timer1_config (void)
Configurar Timer 1 para FREQ_T1 (ver define). Coloca no Modo CTC com limite indicado por OCR1A.

• void timer2_config (void)
Configurar Timer 2 para FREQ_T2 (ver define).
Coloca no Modo CTC com limite indicado por OCR2A.

• byte lcd_linha_alterou (byte qual)
A variável lcd_mudou indica nos seus bits quais linhas foram alteradas. Esta função consulta o bit indicado pela variável "qual"e, se for 1, ela atualiza lcd_lin com o endereço do buffer da linha a ser atualizada, apaga o bit na variável lcd_mudou e retorna TRUE.

• ISR (TIMER1_COMPA_vect)
Atender à interrupção do Timer 1: FREQ_T1 (100 Hz)

• ISR (TIMER2_COMPA_vect)
Atender à interrupção do Timer 2: FREQ_T2 (1000 Hz)

O Arduino Mega tem 5 timers, usados da forma listada abaixo:
Timer1 -> gerar interrupção a cada 100 Hz (10 mseg);
Timer2 -> Atualizar linhas do LCD (1 kHz);
Timer3 -> nada
Timer4 -> nada
Timer5 -> Cronômetro;

TIMER 1

Timer 1 interrompe com taxa de 100 Hz (10 mseg).

Timer 1: Provocar interrupção numa dada freq (f_{OC1A})
CS12:0 = 011B -> Configurar prescaler = 64 -> 16 MHz / 64 = 250 kHz.

WGM13:0 = 0100B -> Modo 4 CTC na coincidência com OCR1A;

FREQ_T1 -> OCR1A = $\frac{f_{clkIO}}{N \cdot freq}$ - 1 = $\frac{16 \cdot 10^6}{64 \cdot freq}$ = $\frac{10^6}{freq}$ = $\frac{25.000}{freq/10}$ (conta para facilitar ao programa calcular OCR1A)

```
#define FREQ_T1 100   //Freq de interrup   o do timer 1
//...
OCR1A = (25000/(FREQ_T1/10))-1;
```

Se for 50 Hz (20 mseg) -> OCR1A = $\frac{f_{clkIO}}{N \cdot freq}$ - 1 = $\frac{16 \cdot 10^6}{64 \cdot 50}$ - 1 = $\frac{10.000}{2}$ - 1 = 4.999

Se for 100 Hz (10 mseg) -> OCR1A = $\frac{f_{clkIO}}{N \cdot freq}$ - 1 = $\frac{16 \cdot 10^6}{64 \cdot 100}$ - 1 = $\frac{10.000}{4}$ - 1 = 2.499

Tabela A.11: Gabarito para configurar os registradores do TC1

	7	6	5	4	3	2	1	0
TCCR1A	COM1A1 0	COM1A0 0	COM1B1 0	COM1B0 0	COM1C1 0	COM1C0 0	WGM11 0	WGM10 0
TCCR1B	ICNC1 0	ICES1 0	- 0	WGM13 0	WGM12 1	CS12 0	CS11 1	CS10 1
TCCR1C	FOC1A 0	FOC1B 0	FOC1C 0	- 0	- 0	- 0	- 0	- 0
TIMSK1	- 0	- 0	ICIE1 0	- 0	OCIE1C 0	OCIE1B 0	OCIE1A 1	TOIE1 0
TIRF1	- 0	- 0	ICF1 0	- 0	OCF1C 0	OCF1B 0	OCF1A 0	TOV1 0

Sobre o ADC

Tabela A.12: Gabaritos para configurar ADC (8 bits alinhados pela esquerda, ler apenas ADCH)

	7	6	5	4	3	2	1	0
ADMUX	REFS1 0,1,1	REFS0 1,1,1	ADLAR 1	MUX4 0	MUX3 0	MUX2 0	MUX1 0,0,1	MUX0 0,1,0
ADCSRA	ADEN 0	ADSC 0	ADATE 0	ADIF 0	ADIE 1	ADPS2 0	ADPS1 1	ADPS0 1
ADCSRB	- -	ACME 0	- -	- -	MUX5 0	ADST2 1	ADST1 0	ADST0 1

Seleção do relógio do ADC -> ADPS2:0 = 3 -> 16 MHz/8 = 2 MHz

Condição	Tempo de conversão (em ciclos)
Primeira conversão	25
Conversão única, canal simples	13

Ao mudar de canal, conversão consome 25 ciclos -> 12,5 μs (80 kHz) -> 200 instruções do AVR.
Conversão com o mesmo canal 13 ciclos -> 6,5 μs (154 kHz) -> 104 instruções do AVR.

Não precisamos de taxas de conversão tão altas e não queremos gastar tempo esperando o ADC converter. A solução é usar o Timer 1 para disparar o ADC numa interrupção e ler o resultado na interrupção seguinte. Porém, precisamos ler 3 entradas analógicas diferentes:
• Canal 0: ler o teclado,
• Canal 1: ler a tensão do carro (12 V) e
• Canal 2: ler a tensão do supercapacitor.

A solução foi usando um contador timer1_cont, contando de 0 até 31. Ele é incrementado a cada interrupção, assim, o Timer 1 sabe o que deve fazer a cada instante. Toda leitura do ADC é fruto da média de duas conversões espaçadas de 10 ms.

• Atualiza tensão do carro (**vcar_val**), na taxa de 100/32 Hz (3Hz).
• Atualiza tensão do super cap (**vcap_val**), na taxa de 100/32 Hz (3Hz).
• Leitura das teclas (**sw_val**) na taxa de 100*(12/32) = 37,5 Hz, uma leitura a cada 27 ms.

Funcionou bem: **timer1_cont** = 0, 1, ..., 31, 0, 1, ..., interrupção em 100 Hz

0) ADC Start	8) +Ler, ADC start*	16) ADC Start	24) +Ler, ADC start*
1) Ler, ADC start	9) Ler, ADC start	17) Ler, ADC start	25) Ler, ADC start
2) +Ler, ADC start*	10) +Ler, ADC start*	18) +Ler, ADC start*	26) +Ler, ADC start*
3) Ler, ADC start	11) Ler, ADC start	19) Ler, ADC start	27) Ler, ADC start
4) +Ler, ADC start*	12) +Ler, Canal1(VCAR)*	20) +Ler, ADC start*	28) +Ler, Canal 2(VCAP)*
5) Ler, ADC start	13) ADC Start	21) Ler, ADC start	29) ADC Start
6) +Ler, ADC start*	14) Ler, ADC start	22) +Ler, ADC start*	30) Ler, ADC start
7) Ler, ADC start	15) +Ler, Canal 0	23) Ler, ADC start	31) +Ler, Canal 0

sw_val = última leitura do teclado
vcar_val = última leitura da tensão (12 V) gerada pelo carro e
vcap_val = última leitura da tensão sobre o supercapacitor.

TIMER 2

Tabela A.13: Timer 2

Freq Timer 2	Atualizar uma linha (42 int)	Instr do AVR
1.000 Hz	42 ms	16.000
5.000 Hz	8,4 ms	3.200
10.000 Hz	4,2 ms	1.600

Por enquanto, está sendo usado 1.000 Hz.
Podemos tentar com a freq. = 5.000 Hz. Isto significa que se consegue atualizar uma linha do LCD entre duas interrupções do Timer 1, que é de 100 Hz (10 ms). Entretanto, gera uma grande quantidade de interrupções próximas. Pesar esse ponto em consideração.

Timer 2: Provocar interrupção numa dada freq (f_{OC2A})

CS22:0 = 100B -> Configurar prescaler = 64 -> 16 MHz / 64 = 250 kHz.
WGM12:0 = 010B -> Modo 2 CTC na coincidência com OCR1A;

Se for 1000 Hz (10 mseg) -> OCR1A = $\frac{f_{clkIO}}{N \cdot freq}$ - 1 = $\frac{16 \cdot 10^6}{64 \cdot 1.000}$ - 1 = $\frac{1.000}{4}$ - 1 = 249

Se for 10.000 Hz (10 mseg) -> OCR1A = $\frac{f_{clkIO}}{N \cdot freq}$ - 1 = $\frac{16 \cdot 10^6}{64 \cdot 10.000}$ - 1 = $\frac{100}{4}$ - 1 = 24

Se for 20.000 Hz (10 mseg) -> OCR1A = $\frac{f_{clkIO}}{N \cdot freq}$ - 1 = $\frac{16 \cdot 10^6}{64 \cdot 20.000}$ - 1 = $\frac{50}{4}$ - 1 = 11

```
1 #define FREQ_T2 1000   //Freq de interrup   o do timer 2
2 //...
3 OCR2A = (25000/(FREQ_T2/10))-1;
```

Tabela A.14: Gabarito para configurar os registradores do TC2

	7	6	5	4	3	2	1	0
TCCR2A	COM2A1 0	COM2A0 0	COM2B1 0	COM2B0 0	-	-	WGM21 0	WGM20 0
TCCR2B	FOC2A 0	FOC2B 0	-	-	WGM22 0	CS22 1	CS21 0	CS20 0
TIMSK2	- -	- -	- -	- -	- -	OCIE2B 0	OCIE2A 1/0	TOIE2 0
TIFR2	- -	- -	- -	- -	- -	OCF2B 0	OCF2A 0	TOV2 0
ASSR	- -	EXCLK 0	AS2 0	TCN2UB 0	OCR2AUB 0	OCR2BUB 0	TCR2AUB 0	TCR2BUB 0
GTCCR	TMS 0	-	-	-	-	-	PSRASY 0	PSRSYNC 0

A.6 #GPS - BIBLIOTECA GPS

Funções

Tabela A.15: Funções - GPS

void	gps_int	(void)
void	gps_des_int	(void)
void	gps_extrai	(byte *vt)
byte	gps_come_vg	(byte ini, byte *vt, byte qtd)
void	gps_cpy_vg	(byte *fonte, byte *dest)
byte	gps_idtf	(byte *vt)
void	gps_str	(byte *msg)
void	gps_char	(byte dt)
void	gps_config	(long br)
	ISR(USART3_TX_vect)	
	ISR(USART3_RX_vect)	

• void gps_int (void)
GPS: Habilitar recepção e interrupção (RX3). UCSR3B: RXIE=1, RXEN=1 .

• **void gps_des_int (void)**
GPS: Desabilitar recepção e interrupção (RX3). UCSR3B: RXIE=0, RXEN=0.

• **void gps_extrai (byte *vt)**
Extrair os dados de uma mensagem do GPS.
Identifica o tipo de mensagem e extrai os dados de interesse.
Atualiza os dados no vetor gps_dados.

• **byte gps_come_vg (byte ini, byte *vt, byte qtd)**
Avançar uma certa quantidade (qtd) de vírgulas a partir da posição ini no vetor vt.
Retorna indexador para a primeira posição após a vírgula.

• **void gps_cpy_vg (byte *fonte, byte *dest)**
Copiar trecho da string fonte para a string dest.
Para (interrompe) quando encontra uma vírgula ou "*". Coloca um zero no final de dest.

• **byte gps_idtf (byte *vt)**
Identifica o tipo de mensagens que está no vetor vt.
Retorna número correspondente:
0=GPS_NADA 1=GPS_RMC 2=GPS_VTG 3=GPS_GGA 4=GPS_GSA 5=GPS_GSV
6=GPS_GLL

• **void gps_str (byte *msg)**
Enviar msg para GPS.

• **void gps_char (byte dt)**
Enviar um char para o GPS. Não usa interrupção.

• **void gps_config (long br)**
Configurar porta serial 3. Não habilita TX e nem RX. Não habilita interrupções.

• **ISR(USART3_TX_vect)**
TX3: Interrupção por dado enviado

• **ISR(USART3_RX_vect)**
RX3: Interrupção por dado recebido.
Se gps_msg_fase = 0 -> armazena em gps_msg_0 [gps_msg_ix++].
Se gps_msg_fase = 1 -> armazena em gps_msg_1 [gps_msg_ix++].
Quando recebe uma mensagem completa (termina com 0xD), marca o final com '\0' (ver posição exata) e faz gps_msg_ok=TRUE.

Sobre o GPS

A recepção do GPS usa 2 vetores alternadamente, enquanto um é preenchido, o outro é analisado, depois troca.
byte **gps_msg_0**[GPS_MSG_TAM], gps_msg_1[GPS_MSG_TAM];
byte **gps_msg_fase** -> indicar qual vetor recebe dados que chegam pela porta serial RX3
byte **gps_msg_ok** -> indicar que completou a recepção de uma mensagem GPS
byte **gps_msg_ix** -> indexador para escrever nos vetores

gps_msg_flase	**gps_msg_0**	**gps_msg_1**
FALSE	Recebendo	Analisando
TRUE	Analisando	Recebendo

Mensagens vindas do GPS têm o formato abaixo:

Tabela A.16: Formato das mensagens do GPS

$	G	P	...	...			xx	CR	LF
24	47	50	...	...			xx	0D	0A

Iniciam com $ e terminam com 0xD e 0xA.

Após o armazenamento no vetor (gps_msg_0 ou gps_msg_1), é colocado um '\0' no lugar do 0xD final.

$	G	P	...	...			xx	'\0'	LF
24	47	50	...	...			xx	'\0'	0A

Iniciam com $ e terminam com 0x00.

As 6 letras iniciais servem para identificar o tipo de mensagem (GPRMC, GPGSA, ...). O tipo de mensagem é descoberto usando o diagrama de estados abaixo.

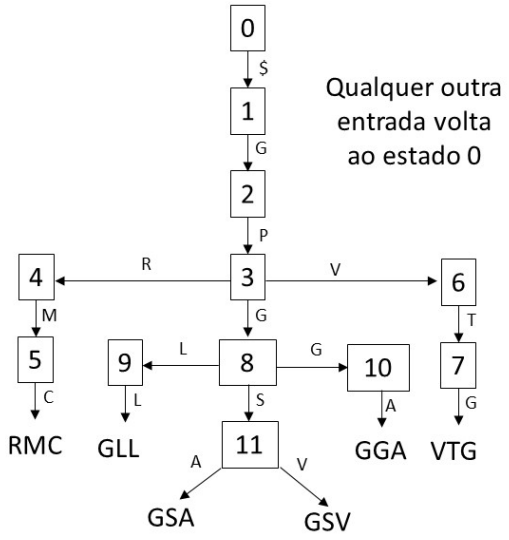

Figura A.4: Diagrama de Estados

Resumo das mensagens de onde se retiraram informações

GPRMC

$GPRMC,hhmmss,status,latitude,N,longitude,E,spd,cog,ddmmyy,mv,mvE,mode*cs<CR><LF>
$GPRMC,083559.00,A,4717.11437,N,00833.91522,E,0.004,77.52,091202,,,A*57

0	1	2	3	4	5	6
$GPRMC,	hhmmss.sss,	Stat,	Lat:ddmm.mmmmm,	N/S,	Long:dddmm.mmmmm,	E/W,
$GPRMC,	083559.00,	A,	4717.11437,	N,	00833.91522,	E,
7	12	2	11	2	12	2

7	8	9	10	11	12	13	14
Speed:ddd.ddd,	Curso:ddd.ddd,	Data:ddmmyy	Mv,	mvE,	Modo	*Check	CR LF
0.004,	77.52,	091202,	,	,	A	*57	0xD 0xA
?8	?8	7	?8	?2	1	3	2

Tamanho = 80 bytes (foi utilizado tamanho 100)
Lido com Arduino:
$GPRMC,131732.00,A,1548.62581,S,04748.65809,W,0.299,,260120,,,A*72

GPGSA

$GPGSA,Smode,FS,sv,PDOP,HDOP,VDOP*cs<CR><LF>
$GPGSA,A,3,23,29,07,08,09,18,26,28,,,,,1.94,1.18,1.54*0D

Mostra identificador de até 12 satélites

0	1	2	3	4	5	6	7	8	9	10
$GPGSA,	Smode,	Fix,	Sat1,	Sat2,	Sat3,	Sat4,	Sat5,	Sat6,	Sat7,	Sat8
$GPGSA,	A,	3,	23,	29,	07,	08,	09,	18,	26,	28,
7	2	2	3	3	3	3	3	3	3	3

11	12	13	14	15	16	17	18	19
Sat9,	Sat10	Sat11,	Sat12,	PDOP,	HDOP,	VDOP	*Check	CR LF
18,	18,	18,	18,	1.94,	1.18,	1.54	*0D	0xD 0xA
3	3	3	3	5	5	5	3	2

Tamanho = 67 bytes (foi utilizado tamanho 100)
Lido com Arduino: $GPGSA,A,3,07,09,16,23,04,01,,,,,,,8.16,1.24,8.06*09

GPVTG

$GPVTG,cogt,T,cogm,M,sog,N,kph,K,mode*cs<CR><LF>
$GPVTG,77.52,T,,M,0.004,N,0.008,K,A*06

Mostra curso e velocidade

0	1	2	3	4	5	6
$GPVTG,	Curso T Cogt:ddd.dd,	Fix,	Curso M Cogm:ddd.dd,	Fix	Veloc Nós Sog:ddd.ddd,	Unidade
$GPVTG,	77.52,	T,	ddd.dd,	M,	0.004,	N,
7	7	2	7	3	8	2

7	8	9	10	11
Veloc kph Sog:ddd.ddd,	Unidade	Modo	*Check	CR LF
0.008	K,	A,	*06	0xD 0xA
8	2	2	3	2

Tamanho = foi utilizado tamanho de 100 bytes
Lido com Arduino: $GPVTG,,T,,M,0.079,N,0.146,K,A*2E

GPGGA

$GPGGA,hhmmss.ss,Latitude,N,Longitude,E,FS,NoSV,HDOP,msl,m,Altref,
m,DiffAge,DiffStation*cs<CR><LF>

$GPGGA,092725.00,4717.11399,N,00833.91590,E,1,8,1.01,499.6,M,48.0,M,,0*5B

Mostra ...

0	1	2	3	4	5	6	7
$GPGGA,	hhmmss.sss,	Lat:ddmm.mmmmm,	N/S,	Long:dddmm.mmmmm,	E/W,	FS,	Nr sat:dd,
$GPGGA,	083559.00,	4717.11437,	N,	00833.91522,	E,	1,	8,
7	12	11	2	12	2	2	

8	9	10	11	12
HDOP,	Alt Msl:dddd.ddd,	uMsl	Altref:dddd.ddd,	Usep
1.18,	1.01,	M,	48.0,	M
5	9	2	9	2

13	14	15	16
Diffage,	DiffStation	*Check	CR LF
S,	0	*57	0xD 0xA
2	2	3	2

Tamanho = foi utilizado tamanho de 100 bytes
Lido com Arduino:
$GPGGA,185326.00,1548.63054,S,04748.65655,W,1,06,2.06,1052.0,M,-11.8,M,,*4F

Vetor usado para separar o que é importante

Vetor **gps_dados[GPS_DADOS_TAM]** armazena as informações extraídas do GPS

RMC	RMC	RMC	RMC	RMC	RMC	RMC
0	2	13	20	31	33	45
Status	Hora	Data	Latitude	N/S	Longitude	E/W
A0	hhmmss.sss 0	ddmmyy 0	ddmm.mmmmm 0	N 0	dddmm.mmmmm 0	E 0
A	083559.00	091202	4717.11437	N	00833.91522	E
2	11	7	11	2	12	2

RMC	RMC	GSA	GSA	GSA	VTG	VTG
47	55	63	69	75	81	88
Velocidade Nós	Curso	PDOP	HDOP	VDOP	Speed km/h	Unidade
ddd.ddd0	ddd.ddd0	dd.dd0	dd.dd0	dd.dd0	xxx.xx0	K0
0.004	77.52	99.99	99.99	99.99	?125.12	K
8	8	6	6	6	7	2

VTG	GGA	GGA	GGA	-	
90	92	95	102	105	110
?Fix?	Qtd Sat	Altitude	Unidade	Adr SRAM	
d0	dd0	dddd.d0	m0	HHHHH0	
1	4	627.4	m	3F4CA	
2	3	7	2	5	

Adr SRAM indica onde estava o ponteiro de gravação da SRAM quando essa mensagem do GPS foi gravada. Vai permitir a sincronização do GPS com o MPU.

```
1 // Marcar posicao de cada um dos parametros guardados em gps_dados[GPS_DADOS_TAM]
2 #define GPS_STATUS      0                   //2 bytes
3 #define GPS_HORA        (GPS_STATUS+2)      //11 bytes
```

```
4  #define GPS_DATA        (GPS_HORA+11)      //7 bytes
5  #define GPS_LAT         (GPS_DATA+7)       //11 bytes
6  #define GPS_NS          (GPS_LAT+11)       //2 bytes
7  #define GPS_LONG        (GPS_NS+2)         //12 bytes
8  #define GPS_EW          (GPS_LONG+12)      //2 bytes
9  #define GPS_VEL_NOS     (GPS_EW+2)         //8 bytes
10 #define GPS_CURSO       (GPS_VEL_NOS+8)    //8 bytes
11 #define GPS_PDOP        (GPS_CURSO+8)      //6 bytes
12 #define GPS_HDOP        (GPS_PDOP+6)       //6 bytes
13 #define GPS_VDOP        (GPS_HDOP+6)       //6 bytes
14 #define GPS_VEL_KPH     (GPS_VDOP+6)       //7 bytes
15 #define GPS_VEL_UN      (GPS_VEL_KPH+7)    //2 bytes
16 #define GPS_FIX         (GPS_VEL_UN+2)     //2 bytes
17 #define GPS_QTD_SAT     (GPS_FIX+2)        //3 bytes sem uso
18 #define GPS_ALT         (GPS_QTD_SAT+3)    //7 bytes
19 #define GPS_ALT_UN      (GPS_ALT+7)        //2 bytes
20 #define GPS_ADR_SRAM    (GPS_ALT_UN+2)     //5 bytes
```

U-Center inicia com as mensagens
B5 62 0A 00 34 B5 62 0A 04 34
B5 62 0A 04 34 B5 62 0A 00 34
B5 62 0A 04 34 B5 62 0A 04 34

O nível dos DOPs (na tabela a seguir) mostra as faixas e níveis de precisão.

Tabela A.17: Faixas e níveis de precisão

Nível DOP	**Qualidade**	**Descrição**
< 1	Ideal	Nível de confiança mais alto; máxima precisão possível em todos os momentos.
1-2	Excelente	Medições precisas
2-5	Bom	Medições com precisão adequadas
5-10	Moderado	Qualidade moderada. Correção recomendada
10-20	Fraco	Nível de confiança baixo. Considere descartar dados
>20	Ruim	Precisão muito baixa. Erros podem atingir 300 metros

A.7 #SERIAL - BIBLIOTECA SERIAL (PORTA 0) PARA O ARDUINO

Não usar classe “Serial.xxx”, pois é usada a interrupção de recepção da porta serial 0.
Senão surge o erro: multiple definition of ‘__vector_25’

Tabela A.18: Funções para a Serial 0

void	ser_cab	(char qual)
void	ser_lat	(byte *vet, byte ns)
void	ser_long	(byte *vet, byte ew)
void	ser_dump_memo	(long adr, byte *vet)
void	ser_gps_dados_lin	(char *gps_vt)
void	ser_gps_dados	(char *gps_vt)
void	ser_lin_ac_tp_gi	(byte *vet)
void	ser_lin_ac_gi	(byte *vet)
void	ser_lin_ac_gi_mg	(byte *vet)
void	ser_ac_gi_mg	(byte *vet)
void	ser_float	(float fx, byte prec)
void	ser_dec32	(long dt)
void	ser_dec32u	(long dt)
void	ser_dec32nz	(long dt)
void	ser_dec32unz	(long dt)
void	ser_hex32	(long dt)
void	ser_dec16	(int dt)
void	ser_dec16u	(word dt)
void	ser_dec16nz	(int dt)
void	ser_dec16unz	(word dt)
void	ser_hex16	(word dt)

void	ser_dec8	(byte dt)
void	ser_dec8u	(byte dt)
void	ser_dec8nz	(byte dt)
void	ser_dec8unz	(byte dt)
void	ser_hex8	(byte dt)
void	ser_crlf	(byte qtd)
void	ser_cr	(byte qtd)
void	ser_lf	(byte qtd)
void	ser_spc	(byte qtd)
void	ser_str	(byte *msg)
void	ser_char	(byte dt)
byte	seri_num16	(int *nr)
byte	seri_num8	(byte *nr)
byte	seri_letra	(char *cha)
char	seri_xereta	(char *cha)
void	seri_config	(void)
char	seri_poe	(char cha)
void	seri_cheia	(void)
char	seri_tira	(char *cha)
void	sero_config	(void)
char	sero_poe	(char cha)
void	sero_cheia	(void)
char	sero_tira	(char *cha)
void	sero_espera	(byte nr)
char	sero_vazia	(void)
byte	sero_uso	(void)
void	sero_timeout	(char nr)
	ISR(USART0_TX_vect)	
	ISR(USART2_TX_vect)	
	ISR(USART0_RX_vect)	
	ISR(USART2_RX_vect)	
void	ser_config	(long br)

• **void ser_cab(char qual)**
Imprimir cabeçalho para Modo de Operação em caso de batida.

• **void ser_lat(byte *vet, byte ns)**
Imprimir LATITUDE.

• **void ser_long(byte *vet, byte ew)**

Imprimir LONGITUDE.

• void ser_dump_memo (long adr, byte *vet)

Auxilia na impressão para o DUMP de memórias. Gera linhas do tipo abaixo

```
0000 0000: F8 A4 32 EB 69 6E 6F 20 65 78 70 6C 6F 72 65 72 - ino explorer
0000 0010: 20 73 74 6B 35 30 30 56 32 20 62 79 20 4D 4C 53 - stk500V2 by MLS
0000 0020: 00 42 6F 6F 74 6C 6F 61 64 65 72 3E 00 48 75 68 - .Bootloader>.Huh
```

• void ser_gps_dados_lin (char *gps_vt)

Imprimir os dados GPS do vetor (pacote gps_dados) em uma linha só. Segue a ordem abaixo. Lembrar que Status = V indica que não achou os satélites.

```
Status – data – hora – LatNS – LONGEW – vel km/h – vel nós – curso – altitude – [qtd sat – PDOP – HDOP – VDOP] – adr SRAM
```

• void ser_gps_dados (char *gps_vt)

Imprimir GPS, todos os dados no pacote gps_dados.

• void ser_lin_ac_tp_gi (byte *vet)

Montar as palavras de 16 bits e imprimir uma linha com acel, temp e giro. Recebe vetor com bytes e monta words para imprimir.

• void ser_lin_ac_gi (byte *vet)

Montar as palavras de 16 bits e imprimir uma linha com acel e giro. Recebe vetor com bytes e monta words para imprimir.

• void ser_lin_ac_gi_mg (byte *vet)

Compor e imprimir uma linha com aceleração, giroscópio e magnetômetro

• void ser_ac_gi_mg (byte *vet)

Compor aceleração, giroscópio e magnetômetro. Imprime 1 por linha

• void ser_float (float f, byte prec)

No Arduino, double e float têm a mesma precisão
Escreve no LCD o float "f"com "prec"casas após a vírgula e apresenta o sinal.
Formato = + xxx xxx xxx , ddd ddd ddd ddd (usar char msg[24])
Limite da parte inteira = 9 dígitos. Limite da parte fracionária = 12 dígitos.
Caso ultrapasse esses os limites imprime ###, ###
O máximo é 999.999.999,999999. Se ultrapassar o máximo, escreve ###,###.
Na verdade, o máximo é 999.999.999,999967. Exemplos:
999.999.999,0 -> imprime 999.999.936,000000 (por causa da precisão da representação)

876.543.210,123456789 -> imprime 876543232,000000 (por causa da precisão da representação)

• **void ser_dec32 (long dt)**
Escrever no LCD 32 bits em decimal, com sinal e com zeros à esquerda.

• **void ser_dec32u (long dt)**
Escrever na fila serial 32 bits em decimal, sem sinal e com zeros à esquerda.

• **void ser_dec32nz (long dt)**
Escrever na fila serial 32 bits em decimal, com sinal e sem zeros à esquerda.

• **void ser_dec32unz (long dt)**
Escrever na fila serial 32 bits em decimal, sem sinal e sem zeros à esquerda.

• **void ser_hex32 (long dt)**
Escrever na fila serial 32 bits em hexadecimal.

• **void ser_dec16 (word dt)**
Escrever na fila serial 16 bits em decimal, com sinal e com zeros à esquerda.

• **void ser_dec16u (word dt)**
Escrever na fila serial 16 bits em decimal, sem sinal e com zeros à esquerda.

• **void ser_dec16nz (word dt)**
Escrever na fila serial 16 bits em decimal, com sinal e sem zeros à esquerda.

• **void ser_dec16unz (word dt)**
Escrever na fila serial 16 bits em decimal, sem sinal e sem zeros à esquerda.

• **void ser_hex16 (word dt)**
Escrever na fila serial 16 bits em hexadecimal.

• **void ser_dec8 (byte dt)**
Escrever na fila serial 8 bits em decimal, com sinal e com zeros à esquerda.

• **void ser_dec8u (byte dt)**
Escrever na fila serial 8 bits em decimal, sem sinal e com zeros à esquerda.

• **void ser_dec8nz (byte dt)**
Escrever na fila serial 8 bits em decimal, com sinal e sem zeros à esquerda.

• **void ser_dec8unz (byte dt)**
Escrever na fila serial 8 bits em decimal, sem sinal e sem zeros à esquerda.

• **void ser_hex8 (byte dt)**
Escrever na fila serial 8 bits em hexadecimal.

• **void ser_crlf (byte qtd)**
Colocar na fila qtd de pares CR (0xD = '\r') e LF (0xA = '\n').

• **void ser_cr (byte qtd)**
Colocar na fila qtd Carriage Return (CR = 0xD = '\r').

• **void ser_lf (byte qtd)**
Colocar na fila qtd Line Feed (LF = 0xA = '\n').

• **void ser_spc (byte qtd)**
Colocar na fila qtd brancos (0x20).

• **void ser_str (byte *msg)**
Colocar a string apontada por msg no buffer serial.

• **void ser_char (byte dt)**
Colocar um char no buffer serial. Gera alerta no LCD se fila serial encher.

• **byte seri_num16(int *nr)**
Retirar um número de 16 bits da fila SERI.

• **byte seri_num8(byte *nr)**
Retirar um número de 8 bits absoluto da fila SERI.

• **byte seri_letra(char *cha)**
Retirar uma letra da fila SERI.

• **char seri_xereta(char *cha)**
Informa qual o próximo byte da fila SERI.

• **void seri_config(void)**
Inicializar fila SERI.

• **char seri_poe(char cha)**

Colocar um byte na fila SERI.

• **void seri_cheia(void)**
Mensagem de Fila SERI Cheia.

• **char seri_tira(char *cha)**
Tirar um byte da fila SERI.

• **void sero_config(void)**
Inicializar fila SERO.

• **char sero_poe(char cha)**
Colocar um byte na fila SERO

• **void sero_cheia(void)**
Mensagem de Fila SERO Cheia.

• **char sero_tira(char *cha)**
Tirar um byte da fila SERO.

• **void sero_espera(byte nr)**
Esperar SERO esvaziar.

• **char sero_vazia(void)**
SERO vazia?

• **byte sero_uso(void)**
SERO em uso.

• **void sero_timeout(char nr)**
TIME OUT.

• **ISR(USART0_TX_vect)**
UART0 TX: Interrupção por dado enviado.

• **ISR(USART2_TX_vect)**
UART2 TX: Interrupção por dado transmitido.

• **ISR(USART0_RX_vect)**
UART0: RX: Interrupção por dado recebido.

• ISR(USART2_RX_vect)
UART2: RX: Interrupção por dado recebido.

• void ser_config(long br)
UART0 e UART2 são configuradas aqui
Habilitar recepção e transmissão
Habilitar interrupção por transmissão e recepção.

Tabela A.19: Gabarito dos registradores de configuração da porta serial USART0

	7	6	5	4	3	2	1	0
UCSR0A	RXC0 0	TXC0 0	UDRE0 0	FE0 0	DOR0 0	UPE0 0	U2X0 1	MPCM0 0
UCSR0B	RXCIE0 0	TXCIE0 1	UDRIE0 0	RXEN0 1	TXEN0 1	UCSZ02 0	RXB80 0	TXB80 0
UCSR0C	UMSEL01 0	UMSEL00 0	UPM01 0	UPM00 0	USBS0 0	UCSZ01 1	UCSZ00 1	UCPOL0 0

Tabela A.20: Exemplos de *Baudrate* para o Arduino Mega (16 MHz)

Baud Rate (bps)	U2Xn = 1	
	UBRR	Erro
9.600	207	0,2%
19.200	103	0,2%
38.400	51	0,2%
57.600	34	-0,8%
115.200	16	2,1%
230.400	8	-3,5%
250.000	7	0%

Filas circulares para transmissão e para recepção:

Fila TX
volatile char ser_tx_fila [SER_TX_FILA_TAM]; //Espaço para a fila serial
volatile byte ser_tx_pin, ser_tx_pout; //Ponteiros para usar a fila
volatile byte ser_tx_ok; //Indica que terminou transmissão

volatile char ser_rx_fila[SER_RX_FILA_TAM]; //Espaço para a fila serial de RX
volatile byte ser_rx_pin, ser_rx_pout; //Ponteiros para usar a fila
volatile byte ser_rx_ok; //Indica que terminou recepção

A.8 #MPU - BIBLIOTECA MPU

Tabela A.21: Funções - MPU

void	mpu_rd_ac_gi_mg	(word *vetor)
byte	mpu_rd_mg_out	(int *vetor)
byte	mpu_rd_mg_reg	(byte reg)
void	mpu_rd_mg_blk	(byte reg, byte *dado, byte qtd)
void	mpu_mag_config	(void)
byte	mpu_mag_self_test	(int *vetor, byte prn)
void	mpu_mag_rd_rom	(byte *vet)
byte	mag_whoami	(void)
byte	mpu_wr_mg_reg	(byte reg, byte dado)
int	mpu_rd_bw	(void)
int	mpu_rd_freq	(void)
int	mpu_rd_esc_giro	(void)
int	mpu_rd_esc_acel	(void)
void	mpu_int	(void)
void	mpu_des_int	(void)
int	mpu_rd_tp	(void)
void	mpu_rd_ac_tp_gi	(word *vetor)
void	mpu_rd_ac_gi	(word *vetor)
void	mpu_acorda	(void)
void	mpu_dorme	(void)
byte	mpu_whoami	(void)
void	mpu_escalas	(byte gfs, byte afs)
void	void mpu_sample_rt	(byte sample_rate)
void	mpu_calibra	(int *vt, word qtd, byte esc_ac, byte esc_gi)
byte	mpu_self_test	(int *vt, byte prn)
void	mpu_wr	(byte reg, byte dado)
byte	mpu_rd	(byte reg)
void	mpu_wr_blk	(byte reg, byte *dado, byte qtd)
void	mpu_rd_blk	(byte reg, byte *dado, byte qtd)
	ISR(INT4_vect)	

• void mpu_rd_ac_gi_mg (word *vetor)
Ler Aceleração, Giro e Mag. Retorna vetor de 18 bytes. Não monta as palavras de 16 bits.
Retorna: [axh axl ayh ayl azh azl gxh gxl gyh gyl gzh gzl hxh hxl hyh hyl hzh hzl].

• byte mpu_rd_mg_out (int *vetor)
Ler Magnetômetro. Retorna vetor de 3 words [hx hy hz]. Copia DRDY para o último bit de hz.
retorna 0: dado não pronto (DRDY = 0)
retorna 1: Tudo bem
retorna 2: overflow (HOFL)

• byte mpu_rd_mg_reg (byte reg)
Ler registrador do magnetômetro

• **void mpu_rd_mg_blk (byte reg, byte *dado, byte qtd)**
Lê em bloco os registradores do magnetômetro.

• **void mpu_mag_config (void)**
Inicializar Magnetômetro – Ainda não sabemos como fazer

• **byte mpu_mag_self_test (int *vetor, byte prn)**
Realizar Self-Test no magnetômetro
Retorna: TRUE se passou no teste
FALSE se falhou no teste

• **void mpu_mag_rd_rom (byte *vet)**
Ler Fuse ROM Magnetômetro

• **byte mpu_wr_mg_reg (byte reg, byte dado)**
Escreve no magnetômetro.

• **int mpu_rd_bw (void)**
Ler no MPU a banda do filtro programada. Retorna 5, 10, 21, ..., 260
(Escala do Acelerômetro).

• **int mpu_rd_freq (void)**
Ler no MPU a freq de amostragem. Retorna 100, 200, ..., 1000.
Considera Gyro Rate = 1.000 (DLPF=0).

• **int mpu_rd_esc_acel (void)**
Ler no MPU a escala usada para o acelerômetro. Retorna 2, 4, 8 ou 16 g.

• **int mpu_rd_esc_giro (void)**
Ler no MPU a escala usada para o giroscópio. Retorna 250, 500, 1000 ou 2000 graus/seg.

• **void mpu_int (void)**
Preparar para MPU usar INT4
Pino PE4 entrada com pullup e habilitar INT4 para flanco de descida
MPU: interrupção em baixo com push-pull, pulso de 50 useg
MPU: Habilitar interrupção dado pronto

• **void mpu_des_int (void)**
Desabilitar o uso de interrupção pelo MPU.
Desabilitar INT4 e MPU: Desabilitar interrupção dado pronto.

• **int mpu_rd_tp (void)**
Retorna a leitura do registrador de temperatura.

• **void mpu_rd_ac_tp_gi (word *vetor)**
Ler Aceleração, temperatura e giro. Retorna vetor de 7 words [ax ay az tp gx gy gz].

• **void mpu_rd_ac_gi (word *vetor)**
Ler Aceleração e giro. Retorna vetor de 6 words [ax ay az gx gy gz].

• **void mpu_acorda (void)**
Acordar o MPU e programar para usar relógio Giro X.

• **void mpu_dorme (void)**
Dormir o MPU e programar para usar relógio Giro X.

• **byte mpu_whoami (void)**
Retornar a leitura do registrador WHO_AM_I.

• **void mpu_config (void)**
Colocar o MPU num estado conhecido.
Taxa = 1 kHz, Banda: Acel=5 Hz e Giro=5 Hz e delay=19 mseg
Taxa de amostragem = taxa/(1+SMPLRT_DIV) = 1k/10 = 100Hz
Escalas acel = +/2g e giro = +/-250 gr/s

• **void mpu_escalas (byte gfs, byte afs)**
Selecionar Fundo de Escalas para o MPU.
Acel: 0=+/-2g, 1=+/-4g, 2=+/-8g, 3=+/-16g.
Gyro: 0=+/-250gr/s, 1=+/-500gr/s, 2=+/-1000gr/s, 3=+/-2000gr/s.

```
#define GIRO_FS_250  0    // +/- 250 graus/seg
#define GIRO_FS_500  1    // +/- 500 graus/seg
#define GIRO_FS_1000 2    // +/- 1000 graus/seg
#define GIRO_FS_2000 3    // +/- 2000 graus/seg

#define ACEL_FS_2G   0    // +/- 2g
#define ACEL_FS_4G   1    // +/- 4g
#define ACEL_FS_8G   2    // +/- 8g
#define ACEL_FS_16G  3    // +/- 16g
```

• **void mpu_sample_rt (byte sample_rate)**
Selecionar Sample Rate, considerando Taxa = 1kHz (Registrador CONFIG)

```
// Valores para o Sample Rate, Registrador SMPLRT_DIV
// Considerando Taxa = 1kHz (Registrador CONFIG)
#define SAMPLE_RT_1kHz      0    // 1.000/(0+1) = 1000
#define SAMPLE_RT_500Hz     1    // 1.000/(1+1) = 500
#define SAMPLE_RT_333Hz     2    // 1.000/(2+1) = 333,33
#define SAMPLE_RT_250Hz     3    // 1.000/(3+1) = 250
#define SAMPLE_RT_200Hz     4    // 1.000/(4+1) = 200
#define SAMPLE_RT_166Hz     5    // 1.000/(5+1) = 166,66
#define SAMPLE_RT_142Hz     6    // 1.000/(6+1) = 142,85
#define SAMPLE_RT_125Hz     7    // 1.000/(7+1) = 125
#define SAMPLE_RT_111Hz     8    //1.000 /(8+1) = 111,11
#define SAMPLE_RT_100Hz     9    //1.000 /(9+1) = 100
```

• void mpu_calibra (int *vt, word qtd, byte esc_ac, byte esc_gi)
MPU: Calibrar
Faz uma série de leituras e retorna a média.

• byte mpu_self_test (int *vt, byte prn)
MPU: Realizar Self-Test (ST) e retornar se passou (TRUE) ou falhou (FALSE).
Opção prn = TRUE, imprime resultados, prn=FALSE, nada imprime.
Vetor int vt[24] recebe resultados de passos intermediários
vt[0, ..., 5] -> leitura dos eixos com selt test OFF
vt[6, ..., 11] -> leitura dos eixos com selt test ON
vt[12, ..., 17] -> leitura dos registradores de self test
vt[18, ..., 23] -> resultado para comparar com tolerância de 14

• void mpu_wr (byte reg, byte dado)
(10) Escrever num registrador do MPU.

• byte mpu_rd (byte reg)
(20) Ler um registrador do MPU.

• void mpu_wr_blk (byte reg, byte *dado, byte qtd)
(30) Escrever um bloco de dados no MPU a partir de um registrador.

• void mpu_rd_blk (byte reg, byte *dado, byte qtd)
(40) Ler um bloco do MPU a partir de um registrador.

• ISR(INT4_vect)
MPU: ISR para a interrupção INT4

A.9 #EEPROM - BIBLIOTECA DE ACESSO À MEMÓRIA EEPROM

Tabela A.22: Funções EEPROM

byte	eeprom_cf_dados	(void)
byte	eeprom_cf_mostra	(void)
void	eeprom_dump	(word adr, word qtd)
long	eeprom_rd_32b	(word adr)
void	eeprom_wr_32b	(word adr, long dado)
int	eeprom_rd_16b	(word adr)
void	eeprom_wr_16b	(word adr, word dado)
void	eeprom_rd_str	(word adr, byte *msg, word qtd)
void	eeprom_wr_str	(word adr, byte *msg)
void	eeprom_wr_ff	(void)
void	eeprom_rd_blk	(word adr, byte *vet, word qtd)
void	eeprom_wr_blk	(word adr, byte *vet, byte qtd)
byte	eeprom_rd	(word adr)
void	eeprom_wr	(word adr, byte dado)

(#define EEPROM_TAM 4096)

Em todo acesso à EEPROM, o campo de endereços é truncado em 12 bits. Não há verificação do limite do endereço fornecido. Durante a escrita, as interrupções são momentaneamente desabilitadas.

• byte eeprom_cf_dados (void)
Imprime apenas os dados da Calibração de Fábrica gravada na EEPROM.
Não coloca rótulo em cada dado da calibração. Ver exemplo no final deste arquivo.

• byte eeprom_cf_mostra (void)
Mostrar Calibração de Fábrica gravada na EEPROM.
Coloca um rótulo em cada dado da calibração. Ver exemplo no final deste arquivo.

• void eeprom_dump (word adr, word qtd)
Dump da EEPROM.

• long eeprom_rd_32b (word adr)
Ler um valor de 32 bits da EEPROM, Big Endian.

• Void eeprom_wr_32b (word adr, long dado)
Escrever um valor de 32 bits da EEPROM.

• int eeprom_rd_16b (word adr)

Ler um valor de 16 bits da EEPROM, Big Endian.

• **void eeprom_wr_16b (word adr, word dado)**
Escrever um valor de 16 bits da EEPROM.

• **void eeprom_rd_str (word adr, byte *msg, word qtd)**
Ler uma string à partir do endereço adr da EEPROM. Copia o zero final.
Qtd indica a quantidade máxima e, caso se chegue a esse valor, o último byte é o zero final.

• **void eeprom_wr_str (word adr, byte *msg)**
Gravar uma string na EEPROM, incluindo o zero final.

• **void eeprom_wr_ff (void)**
Gravar 0xFF em toda a EEPROM.

• **void eeprom_rd_blk (word adr, byte *vet, word qtd)**
Ler um bloco a partir de um endereço. O endereço adr é truncado em 12 bits.

• **void eeprom_wr_blk (word adr, byte *vet, byte qtd)**
Escrever um bloco a partir de um endereço. O endereço adr é truncado em 12 bits.

• **byte eeprom_rd (word adr)**
Ler um endereço da EEPROM. O endereço adr é truncado em 12 bits.

• **byte eeprom_wr (word adr, byte dado)**
Escrever o dado no endereço adr da memória. O endereço adr é truncado em 12 bits.
Nesta escrita, as interrupções são momentaneamente desabilitadas.

Impressão byte eeprom_cf_mostra (void)

```
— EEPROM: Dados da Calibração de Fábrica —
Data = 20/04/20
Local = Brasilia
g Padrao = 9,8066501 (01023)
g Local = 9,7808437 (01020)
Who am I = 0x0073
Freq (Hz) Amost = 100
ax ay az tp gx gy gz
Erro intinseco: +00632 +01053 +13162 +01429 -00333 +00133 -00055
Primeira medida: +00644 +01076 +13012 +01428 -00334 +00132 -00055
Ultima medida: +00596 +01044 +13192 +01428 -00335 +00130 -00053
Somatorio por eixo (32 bits): +0000063296 +0000105336 +0001316236
+0000142980 -0000033368 +0000013352 -0000005522
100 = Medidas para a Media
Self Test = FALHOU!
ax ay az gx gy gz
Self Test OFF: +00165 +00259 +03287 -00331 +00132 -00051
Self Test ON: +02072 +02062 +05407 +16088 +19129 +24211
Registradores: +00013 +00015 +00015 +00014 +00007 +00022
Result (<14%): -00008 -00018 -00004 +00179 -00542 +00188
— EEPROM: Fim dos Dados da Calibração de Fábrica —
```

Impressão byte eeprom_cf_dados (void)

Ao ler com o Matlab, cuidado com a vírgula na aceleração (9,80...).

```
+21331
20/04/20
Brasilia
9,8066501
9,7808437
01023
01020
0073
100
+00632 +01053 +13162 +01429 -00333 +00133 -00055
100
+0000063296 +0000105336 +0001316236 +0000142980 -0000033368
+0000013352 -0000005522
+00644 +01076 +13012 +01428 -00334 +00132 -00055
+00596 +01044 +13192 +01428 -00335 +00130 -00053
+21331
+00165 +00259 +03287 -00331 +00132 -00051
+02072 +02062 +05407 +16088 +19129 +24211
+00013 +00015 +00015 +00014 +00007 +00022
-00008 -00018 -00004 +00179 -00542 +00188
```

A.10 #FLASH - BIBLIOTECA DE ACESSO À MEMÓRIA FLASH

Tabela A.23: Faixas de endereços das memórias Flash (2 memórias de 128 KB cada)

Início	Fim	Flash	Bloco	Define	Define
0 0000	0 FFFF	1	0	FLASH1_ADR+0	0x50
1 0000	1 FFFF	1	1	FLASH1_ADR+4	0x54
2 0000	2 FFFF	2	0	FLASH2_ADR+0	0x51
3 0000	3 FFFF	2	1	FLASH2_ADR+4	0x55

Tabela A.24: Funções - FLASH

void	flash_dados_acid	(void)
void	flash_op_mostra	(void)
void	flash_dump	(long adr, long qtd)
long	flash_rd_32b	(long adr)
void	flash_wr_32b	(long adr, long dado)
int	flash_rd_16b	(long adr)
void	flash_wr_16b	(long adr, int dado)
void	flash_rd_str	(long adr, byte *msg, word qtd)
void	flash_wr_str	(long adr, byte *msg)
void	flash_apg_blk	(long adr, byte qtd)
void	flash_rd_blk	(long adr, byte *vet, word qtd)
void	flash_wr_blk	(long adr, byte *vet, byte qtd)
byte	flash_rd	(long adr)
void	flash_wr	(long adr, byte dado)
byte	flash_qual	(long adr)
void	flash_espera	(long adr)

• **void flash_dados_acid (void)**
Serve para a Flash Batida. Apresentar todos os dados do acidente. Inclui GPS e MPU.

• **void flash_op_mostra (void)**
Mostrar configuração feita ao ligar o carro.

• **void flash_dump (long adr, long qtd)**
Dump da FLASH. Mostra qtd bytes a partir do endereço adr.
Usa a função ser_dump_memo (long adr, char *vet) para mostrar uma linha.

• **long flash_rd_32b (long adr)**
Ler um valor de 32 bits da FLASH, Big Endian.

• **void flash_wr_32b (long adr, long dado)**
Escrever um valor de 32 bits na FLASH
Big Endian.

• **int flash_rd_16b (long adr)**
Ler um valor inteiro de 16 bits da FLASH. Big Endian.

• **void flash_wr_16b (long adr, int dado)**
Escrever um valor inteiro de 16 bits da FLASH. Big Endian.

• void flash_rd_str (long adr, byte *msg, word qtd)
Ler uma string da FLASH. O ponteiro msg deve ter espaço adequado.
Qtd indica a quantidade máxima, o último byte é o zero final

• void flash_wr_str (long adr, byte *msg)
Gravar uma string na FLASH, terminada em zero.

• void flash_apg_blk (long adr, byte qtd)
Escrever um bloco com FF.

• void flash_rd_blk (long adr, byte *vet, word qtd)
(80) Ler um bloco a partir de um endereço.
qtd –> não há limitação desde que fique dentro dos 16 bits (64 K).

• void flash_wr_blk (long adr, byte *vet, byte qtd)
(70) Escrever um bloco a partir de um endereço.
qtd –> precisa estar dentro de página de 128 bytes (apenas os 7
LSbits de endereço variam).
Usar a função void flash_espera(long adr) para ser avisado assim que a operação terminar.

• byte flash_rd (long adr)
(50) Ler um endereço da memória.

• byte flash_wr (long adr, byte dado)
(60) Escrever o dado no endereço adr da memória.
Usar a função void flash_espera(long adr) para ser avisado assim que a operação terminar.

• byte flash_qual (long adr)
Retornar endereço TWI (I2C) da flash a ser usada, em função do endereço a ser acessado.
#define FLASH1_ADR 0x50 //FLASH1
#define FLASH2_ADR 0x51 //FLASH2
o 0 0000 –> 0 FFFF -> retorna FLASH1_ADR+0 (0x50)
o 1 0000 –> 1 FFFF -> retorna FLASH1_ADR+4 (0x54)
o 2 0000 –> 2 FFFF -> retorna FLASH2_ADR+0 (0x51)
o 3 0000 –> 3 FFFF -> retorna FLASH2_ADR+5 (0x55)
Exemplo:
byte er,et;
er=flash_qual(endereço);
er=er«1; //Endereço escravo receptor
et=et+1; //Endereço escravo transmissor

• **void flash_espera (long adr)**

(100) Esperar Flash terminar gravação no endereço adr.

Ficar acessando com WR até ela responder.

```
1 // Enderecos da FLASH 24LC1025 (128 KB) TWI
2 #define FLASH1_ADR       0x50   //FLASH1
3 #define FLASH2_ADR       0x51   //FLASH2
4 #define FLASH1_ADR_B0    FLASH1_ADR+0   //FLASH1, Bloco 0: 64KB (0x00000 -> 0x0FFFF)
5 #define FLASH1_ADR_B1    FLASH1_ADR+4   //FLASH1, Bloco 1: 64KB (0x10000 -> 0x1FFFF)
6 #define FLASH2_ADR_B0    FLASH2_ADR+0   //FLASH2, Bloco 0: 64KB (0x00000 -> 0x0FFFF)
7 #define FLASH2_ADR_B1    FLASH2_ADR+4   //FLASH2, Bloco 1: 64KB (0x10000 -> 0x1FFFF)
8 #define FLASH_PAG    128                //Tamanho da pagina para gravacao
```

A.11 #SRAM-SPI - BIBLIOTECA PARA A SRAM E SPI

São duas memórias de 128 KB que vão trabalhar como uma única de 256 KB.

O acesso precisa ficar limitado a uma página de 128 KB.

A partir do endereço, a função decide qual memória acessar.

Tabela A.25: Funções - SRAM/SPI

void	sram_flash	(void)
void	sram_op_dados	(void)
void	sram_op_mostra	(void)
void	sram_dump	(long adr, long qtd)
void	sram_zera_mpu_gps	(void)
int	sram_rd_32b	(long adr)
void	sram_wr_32b	(long adr, long dado)
int	sram_rd_16b	(long adr)
void	sram_wr_16b	(long adr, int dado)
void	sram_wr_str	(long adr, byte *msg)
void	sram_zera	(void)
void	sram_rd_str	(long adr, byte *msg, word qtd)
void	sram_wr_blk	(long adr, byte *vet, word qtd)
void	sram_rd_blk	(long adr, byte *vet, word qtd)
void	sram_wr	(long adr, byte dado)
byte	sram_rd	(long adr)
void	sram_modo_wr	(byte qual, byte dado)
byte	sram_modo_rd	(byte qual)
void	spi_cs0	(void)
void	spi_CS0	(void)
void	spi_cs1	(void)
void	spi_CS1	(void)
void	spi_cs3	(void)
void	spi_CS3	(void)
void	spi_cs4	(void)
void	spi_CS4	(void)
byte	spi_transf	(byte dado)
void	spi_config	(byte clk)

• **void sram_flash (void)**
Copiar toda a SRAM para a FLASH. Gasta 9,4 seg.
Deveria gastar 6,144 seg (2.048 pag x 3ms = 6,144 seg) + 3 seg em SPI e TWI
// #define FLASH_PAG 128
// #define GPS_ADR_FIM 0x40000L //Fim área GPS

• **void sram_op_dados (void)**
Mostrar toda a configuração ao ligar o carro na SRAM.
Não imprime qualquer tipo de rótulo, só os dados.

• **void sram_op_mostra (void)**
Mostrar toda a configuração ao ligar o carro na SRAM
Imprime os dados rotulados.

• **void sram_dump (long adr, long qtd)**
Dump da SRAM. Mostra qtd bytes a partir do endereço adr.
Arredonda “adr” para o múltiplo de 16 inferior (evita problema com página de 128K) (adr &= 0xFFFFFF)
Usa a função ser_dump_memo (long adr, char *vet) para mostrar uma linha.

void sram_zera_mpu_gps (void)
Zerar somente área usada pelo MPU e GPS. Salva Calibração ao ligar, zera e depois a reescreve.

• **long sram_rd_32b (long adr)**
Ler um valor de 32 bits da SRAM, Big Endian.

• **void sram_wr_32b (long adr, long dado)**
Escrever um valor de 32 bits da SRAM
Big Endian.

• **int sram_rd_16b (long adr)**
Ler um valor inteiro de 16 bits da SRAM.
Big Endian.

• **void sram_wr_16b (long adr, int dado)**
Escrever um valor de 16 bits da SRAM
Big Endian.

• **void sram_wr_str (long adr, byte *msg)**
Gravar uma string na SRAM, incluindo o zero final.

• **void sram_zera (void)**
Zerar toda a SRAM. Com SCLK = 8 MHz, gasta 694 mseg.

• **void sram_rd_str (long adr, byte *msg, word qtd)**
Ler uma string da SRAM
O ponteiro "msg"deve ter espaço adequado
Qtd indica a quantidade máxima, o último byte é o zero final.

• **void sram_wr_blk (long adr, byte *vet, word qtd)**
Escrever uma sequência de "qtd"bytes a partir do endereço adr.
Faz a transição de 0x1FFFF para 0x20000.
**-> Não faz a volta de 0x3FFFF para 0x00000 (fica na pag 1: volta para 0x20000).
Se adr = 0x0 0000 -> 0x1 FFFF, SRAM0 #CS0.

Se adr = 0x2 0000 -> 0x3 FFFF, SRAM1 #CS1.

• **void sram_rd_blk (long adr, byte *vet, word qtd)**
Ler uma sequência de qtd bytes e guardar no vetor vet.
Faz a transição de 0x1FFFF para 0x20000.
**-> Não faz a volta de 0x3FFFF para 0x00000 (fica na pag 1: volta para 0x20000).
Se adr = 0x0 0000 -> 0x1 FFFF, SRAM0 #CS0.
Se adr = 0x2 0000 -> 0x3 FFFF, SRAM1 #CS1.

• **void sram_wr (long adr, byte dado)**
Escrever o dado no endereço adr da SRAM.
Se adr = 0x0 0000 -> 0x1 FFFF, SRAM0 #CS0.
Se adr = 0x2 0000 -> 0x3 FFFF, SRAM1 #CS1.

• **byte sram_rd (long adr)**
Ler o endereço adr da SRAM e retornar o dado lido.
Se adr = 0x0 0000 -> 0x1 FFFF, SRAM0 #CS0.
Se adr = 0x2 0000 -> 0x3 FFFF, SRAM1 #CS1.

• **void sram_modo_wr (byte qual, byte dado)**
Escrever no registrador de modo.
Se qual=0 –> SRAM0 #CS0 (0x0 0000 -> 0x1 FFFF).
Se qual=1 –> SRAM1 #CS1 (0x2 0000 -> 0x3 FFFF).

• **byte sram_modo_rd (byte qual)**
Ler o registrador de modo.
Se qual=0 –> SRAM0 #CS0 (0x0 0000 -> 0x1 FFFF).
Se qual=1 –> SRAM1 #CS1 (0x2 0000 -> 0x3 FFFF).

• **void spi_cs0 (void)**
CS0 (PL0) = LOW, SRAM 0 (23LC1024)

• **void spi_CS0 (void)**
CS0 (PL0) = HIGH SRAM 0 (23LC1024)

• **void spi_cs1 (void)**
CS1 (PL1) = LOW, SRAM 1 (23LC1024)

• **void spi_CS1 (void)**
CS1 (PL1) = HIGH, SRAM 1 (23LC1024)

• **void spi_cs3 (void)**
CS3 (PC3) = LOW, FLASH 0 (W25Q64)

• **void spi_CS3 (void)**
CS3 (PC3) = HIGH, FLASH 0 (W25Q64)

• **void spi_cs4 (void)**
CS4 (PC4) = LOW, FLASH 1 (W25Q64)

• **void spi_CS4 (void)**
CS4 (PC4) = HIGH, FLASH 1 (W25Q64)

• **byte SPI_transf (byte dado)**
Enviar um Byte pela porta SPI.
Rotina envia e recebe um Byte ao mesmo tempo.

• **void spi_config (byte clk)**
Inicializar porta SPI do Mega

```
1 #define FLASH_PAG    128                //Tamanho da pagina para gravacao
```

Exemplo de void sram_op_mostra (void)

```
— SRAM: Dados da Configuração ao Ligar o carro —
Caixa Preta Não Acidentada.
Sef Test = NOK.
Calibracao de Fabrica = 8 medidas para Calibrar ao Ligar
Calibra Escala Acel = +/- 2g
Calibra Escala Giro = +/- 250 gr/s
Calibracao (ax-ay-az-tp-gx-gy-gz): +05103 +00500 +10773 +00969
+02997 +02895 +02559
Freq de Amostragem = 100 Hz =
Opera Escala Acel = +/- 8g
Opera Escala Giro = +/- 2000 gr/s
Limiares de disparo (ax-ay-az-gx-gy-gz): 4 4 4 1000 1000 1000
Disparo no endereco: 0x00000000
Quem Disparou: AX AY AZ GX GY GZ
N N N N N N
Data do acidente: dd/mm/yy
Hora do acidente: hh:mm:ss
— SRAM: Fim dos Dados da Configuração ao Ligar o carro —
```

Exemplo de void sram_op_dados (void)

```
20046
20046
00083 8
0
0
+05155 +00556 +10846 +01068 +03807 +03589 +03199
0
0
0 0 0
9
2
3
0
+00004 +00004 +00004 +01000 +01000 +01000
0
0
+20046 +20046 +20046 +20046 +20046 +20046
ddmmyy
hhmmss.sss
```

Conjunto de instruções para a memória 23LC1024:

Tabela A.26: Conjunto de instruções para a memória 23LC1024

Instrução	Formato	Hexa	Descrição
READ	0000 0011	0x03	Ler a memória a partir do endereço selecionado
WRITE	0000 0010	0x02	Escrever na memória a partir do endereço selecionado
EDIO	0011 1011	0x3B	Habilitar o acesso Dual
EQIO	0011 1000	0x38	Habilitar o acesso Quad
RSTIO	1111 1111	0xFF	Reset do acesso Dual e Quad
RDMR	0000 0101	0x05	Ler o Registrador de Modo
WRMR	0000 0001	0x01	Escrever no Registrador de Modo

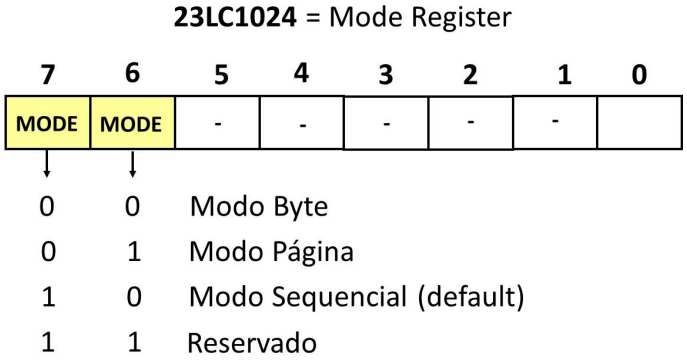

Figura A.5: Registrador de Modo da memória 23LC1024.

Tabela A.27: Gabaritos para configurar ADC (8 bits alinhado pela esquerda, ler apenas ADCH)

	7	6	5	4	3	2	1	0
SPCR	SPIE 0	SPE 1	DORD 0	MSTR 1	CPOL 0	CPHA 0	SPR1 0,1	SPR0 0,1
SPSR	SPIF 0	WCOL 0	- -	- -	- -	- -	- -	SPI2X 0,1

Para modo mestre (MSTR) é preciso SS=1

SPI2X	SPR1	SPR0	Frequência SCK	Arduino f_{osc} = 16 MHz
0	0	0	f_{osc}/4	4 MHz
0	0	1	f_{osc}/16	1 MHz
0	1	0	f_{osc}/64	250 kHz
0	1	1	f_{osc}/128	125 kHz
1	0	0	f_{osc}/2	8 MHz
1	0	1	f_{osc}/8	2 MHz
1	1	0	f_{osc}/32	500 kHz
1	1	1	f_{osc}/64	250 kHz

```
// Velocidades SPI, verificar o dobrador (SPI2X)
#define SPI_125K      0 //SCL=125KHz, SPI2X=0
#define SPI_250K      1 //SCL=250KHz, SPI2X=0
#define SPI_500K      2 //SCL=500KHz, SPI2X=1
#define SPI_1M      3 //SCL=1MHz, SPI2X=0
#define SPI_2M      4 //SCL=2MHz SPI2X=1
#define SPI_4M      5 //SCL=4MHz SPI2X=0
#define SPI_8M      6 //SCL=8MHz SPI2X=1

#define MISO      50 //Master Input
#define MOSI      51 //Master Output
#define SCK       52 //Saída do relógio
#define CS0       49 //(PL0) Controla o estado do #CS0
#define CS1       48 //(PL1) Controla o estado do #CS1
#define CS2       47 //(PL2) Controla o estado do #CS1
#define SS        53 //Controla o estado do #CS do SD Card
```

A.12 #STRINGS - BIBLIOTECA PARA STRINGS

Gerar e operar strings para facilitar a impressão no LCD, na porta serial, entre outros.

Tabela A.28: Funções - STRINGS

void	str_long	(byte *longi, byte ew, byte *msg)
void	str_lat	(byte *lati, byte ns, byte *msg)
void	str_fs_acel	(byte esc, char *vt)
void	str_fs_giro	(byte esc, char *vt)
byte	str_copia_zf	(byte *ft, byte *dest)
byte	str_copia	(byte *ft, byte *dest)
byte	str_tam	(byte *ft)
void	str_rmvz_u	(char *msg)
void	str_rmvz_s	(char *msg)
void	str_float	(float f, byte prec, char *msg)
long	str_to_dec32	(char *vt)
void	str_dec32	(long c, char *msg)
void	str_dec32u	(long c, char *msg)
void	str_hex32	(long c, char *msg)
void	str_dec16u	(word c, char *msg)
void	str_dec16	(int c, char *msg)
void	str_hex16	(word c, char *msg)
void	str_dec8u	(char c, char *msg)
void	str_dec8	(byte c, char *msg)
void	str_hex8	(byte c, char *msg)
void	str_spc	(char qtd, char *msg)
void	str_crlf	(char qtd, char *msg)
void	str_cr	(char qtd, char *msg)
void	str_lf	(char qtd, char *msg)
byte	asc_nib	(byte asc)

• **void str_long (byte *longi, byte ew, byte *msg)**
Retorna String com a LONGITUDE para Googlemaps
Recebe = dddmm.mmmmm (degraus and minutes)
ns = ’E’ ou ’W’
msg = +/-ddd.ddddddd.

• **void str_lat (byte *lati, byte ns, byte *msg)**
Retorna String com a LATITUDE para Googlemaps
Recebe = ddmm.mmmmm (degraus and minutes)
ns = ’N’ ou ’S’
msg = +/-dd.ddddddd.

• **void str_fs_acel (byte esc, char *vt)**
Copia em vt, string com a escala Aceleração. São 8 caracteres incluindo o zero final.

• void str_fs_giro (byte esc, char *vt)
Escala Giro (Fundo de Escala) - Retorna string com a escala
Precisa de espaço de 14 caracteres incluindo o zero final.

• byte str_copia_zf (byte *ft, byte *dest)
Faz cópia de uma string e retorna tamanho string copiada. Inclui o Zero Final.

• byte str_copia (byte *ft, byte *dest)
Faz cópia de uma string e retorna tamanho string copiada, não conta o zero.
Nome criado para não confundir com strcpy().

• byte str_tam (byte *ft)
Retorna tamanho da string, não conta o zero final.

• void str_rmvz_u (char *msg)
Remove os zeros à esquerda da string de número sem sinal que está em msg.

• void str_rmvz_s (char *msg)
Remove os zeros à esquerda da string de número com sinal que está em msg.

• void str_float (float f, byte prec, char *msg)
No Arduino, double e float têm a mesma precisão
Escreve em msg o float fx com prec casas após a vírgula e apresenta o sinal.
Formato = + xxx xxx xxx , ddd ddd ddd ddd (usar char msg[24])
Limite da parte inteira = 9 dígitos.
Limite da parte fracionária = 12 dígitos.
Caso ultrapasse os limites imprime ###, ###
O máximo é 999.999.999,999999. Se ultrapassar o máximo, escreve ###,###. Na verdade, o máximo é 999.999.999,999967. Exemplos:
999.999.999,0 -> imprime 999.999.936,000000 (por causa da precisão da representação)
876.543.210,123456789 -> imprime 876543232,000000 (por causa da precisão da representação)

• long str_to_dec32 (char *vt)
Transforma uma string em long.

• void str_dec32 (long c, char *msg)
Escreve em msg o *long* decimal 32 bits com sinal e com zeros à esquerda.
Usar char msg[12], pois +4 294 967 295 \0 - 12 posições.

• void str_dec32u (long c, char *msg)

Escreve em msg o *unsigned long* decimal 32 bits sem sinal e com zeros à esquerda.
Usar char msg[12], pois +4 294 967 295 \0 - 12 posições.

• void str_hex32 (long c, char *msg)
Escreve em msg o *long* hexadecimal de 32 bits. Usar char msg[9].

• void str_dec16 (int c, char *msg)
Escreve em msg o *int* decimal 16 bits com sinal e com zeros à esquerda.
Usar char msg[7], pois +67 295 \0 - 7 posições.

• void str_dec16u (word c, char *msg)
Escreve em msg o *word* decimal 16 bits sem sinal e com zeros à esquerda.
Usar char msg[7], pois +67 295 \0 - 7 posições.

• void str_hex16 (word c, char *msg)
Escreve em msg o *word* hexadecimal de 16 bits. Usar char msg[5].

• void str_dec8 (char c, char *msg)
Escreve em msg o *char* decimal 8 bits com sinal e com zeros à esquerda.
Usar char msg[5], pois +123 - 5 posições.

• void str_dec8u (byte c, char *msg)
Escreve em msg o *byte* decimal 8 bits sem sinal e com zeros à esquerda.
Usar char msg[5], pois +295 \0 - 5 posições.

• void str_hex8 (byte c, char *msg)
Escreve em msg o *byte* hexadecimal de 8 bits. Usar char msg[3].

• void str_spc(char qtd, char *msg)
Escrever em msg uma qtd de espaços = 0x20 (Espaço em Branco). Prever msg com tamanho adequado.

• void str_crlf(char qtd, char *msg)
Escrever em msg uma qtd de pares CR (‘\r’=0xD) e LF (‘\n’=0xA). Prever msg com tamanho adequado.

• void str_cr(char qtd, char *msg)
Escrever em msg uma qtd de CR (‘\r’=0xD). Prever msg com tamanho adequado.

• void str_lf(char qtd, char *msg)
Escrever em msg uma qtd de LF (‘\n’=0xA). Prever msg com tamanho adequado.

• **byte asc_nib (byte asc)**

Converter ASCII em nibble.

ASCII = 0x30, 0x31, ..., 0x49, 0x41, ..., 0x46. (0,1,2, ..., F)

B. CÓDIGOS-FONTE

B.1 CÓDIGO DA INTERFACE GRÁFICA

B.1.1 *main.py*

```
import sys
import struct
import time
from PyQt5.QtWidgets import QMainWindow, QApplication
from GUI import *
from customSerial import customSerial


class MiApp(QMainWindow):
    def __init__(self):
        super().__init__()
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)

        # Serial
        self.serial = customSerial()

        self.ui.BaudList.addItems(self.serial.baudratesDIC.keys())
        self.ui.BaudList.setCurrentText('9600')
        self.update_ports()

        # Events
        self.ui.connectBtn.clicked.connect(self.connect_serial)
        self.ui.sendBtn.clicked.connect(self.send_data)
        self.ui.updateBtn.clicked.connect(self.update_ports)
        self.ui.clearBtn.clicked.connect(self.clear_terminal)
        self.ui.botaoInterromp.clicked.connect(self.interromp)
        self.ui.botaoOpera.clicked.connect(self.opera)
        self.ui.botaoTeste.clicked.connect(self.teste)
        self.ui.botaoOpcoes.clicked.connect(self.opcoes)
        self.ui.botaoNum1.clicked.connect(self.tecla_1)
        self.ui.botaoNum2.clicked.connect(self.tecla_2)
        self.ui.botaoNum3.clicked.connect(self.tecla_3)
        self.ui.botaoNum4.clicked.connect(self.tecla_4)
        self.ui.botaoNum5.clicked.connect(self.tecla_5)
        self.ui.botaoNum6.clicked.connect(self.tecla_6)
        self.ui.botaoNum7.clicked.connect(self.tecla_7)
        self.ui.botaoNum8.clicked.connect(self.tecla_8)
        self.ui.botaoNum9.clicked.connect(self.tecla_9)
        self.ui.botaoNum10.clicked.connect(self.tecla_10)
        self.ui.botaoNum11.clicked.connect(self.tecla_11)
```

```
42          self.ui.botaoNum12.clicked.connect(self.tecla_12)
43          self.ui.botaoNum13.clicked.connect(self.tecla_13)
44          self.ui.botaoNum14.clicked.connect(self.tecla_14)
45
46          self.serial.data_available.connect(self.update_terminal)
47
48      def update_terminal(self, data):
49          self.ui.Terminal.append(data)
50
51      def connect_serial(self):
52          if (self.ui.connectBtn.isChecked()):
53              port = self.ui.portList.currentText()
54              baud = self.ui.BaudList.currentText()
55              self.serial.serialPort.port = port
56              self.serial.serialPort.baudrate = baud
57              self.serial.connect_serial()
58              # Se conseguiu conectar
59              if (self.serial.serialPort.is_open):
60                  self.ui.connectBtn.setText('DESCONECTAR')
61
62              # Se nao conseguiu se conectar
63              else:
64                  # print("Nao conectou")
65                  self.ui.connectBtn.setChecked(False)
66
67          else:
68              # print("Desconectar")
69              self.serial.disconnect_serial()
70              self.ui.connectBtn.setText('CONECTAR')
71
72      def send_data(self):
73          data = self.ui.input.text()
74          self.serial.send_data(data)
75
76      def interromp(self):
77          cmd = 'x'
78          self.serial.send_data(cmd)
79          time.sleep(0.5)
80
81      def opera(self):
82          self.ui.botaoTeste.setChecked(False)
83          self.ui.botaoNum1.setText('Adquirir Dados')
84          self.ui.botaoNum2.setText('Vazio')
85          self.ui.botaoNum3.setText('Vazio')
86          self.ui.botaoNum4.setText('Vazio')
87          self.ui.botaoNum5.setText('Calibra Fab')
88          self.ui.botaoNum6.setText('Calibra Mag')
89          self.ui.botaoNum7.setText('Vazio')
90          self.ui.botaoNum8.setText('Vazio')
91          self.ui.botaoNum9.setText('Desligar')
92          self.ui.botaoNum10.setText('Vazio')
93          self.ui.botaoNum11.setText('Vazio')
```

```
94          self.ui.botaoNum12.setText('Vazio')
95          self.ui.botaoNum13.setText('Vazio')
96          self.ui.botaoNum14.setText('Vazio')
97          cmd = 'o'
98          self.serial.send_data(cmd)
99          time.sleep(0.2)
100
101     def teste(self):
102         self.ui.botaoOpera.setChecked(False)
103         self.ui.botaoNum1.setText('LEDs')
104         self.ui.botaoNum2.setText('LCD')
105         self.ui.botaoNum3.setText('Teclado')
106         self.ui.botaoNum4.setText('TWI (I2C)')
107         self.ui.botaoNum5.setText('Acel e Giro')
108         self.ui.botaoNum6.setText('Magnetometro')
109         self.ui.botaoNum7.setText('SRAM')
110         self.ui.botaoNum8.setText('FLASH')
111         self.ui.botaoNum9.setText('GPS: Tudo')
112         self.ui.botaoNum10.setText('GPS: Interpreta')
113         self.ui.botaoNum11.setText('GPS: U-Center')
114         self.ui.botaoNum12.setText('MPU-->MatLab')
115         self.ui.botaoNum13.setText('Bluetooth')
116         self.ui.botaoNum14.setText('BT - Cmds AT')
117         cmd = 't'
118         self.serial.send_data(cmd)
119         time.sleep(0.2)
120
121     def opcoes(self):
122         cmd = '?'
123         self.serial.send_data(cmd)
124         time.sleep(0.5)
125
126     def tecla_1(self):
127         cmd = 1
128         cmd2 = b''
129         cmd2 += struct.pack('!B', int(cmd))
130         self.serial.send_data(cmd2)
131         time.sleep(0.5)
132
133     def tecla_2(self):
134         cmd = 2
135         cmd2 = b''
136         cmd2 += struct.pack('!B', int(cmd))
137         self.serial.send_data(cmd2)
138         time.sleep(0.5)
139
140     def tecla_3(self):
141         cmd = 3
142         cmd2 = b''
143         cmd2 += struct.pack('!B', int(cmd))
144         self.serial.send_data(cmd2)
145         time.sleep(0.5)
```

```
146
147     def tecla_4(self):
148         cmd = 4
149         cmd2 = b''
150         cmd2 += struct.pack('!B', int(cmd))
151         self.serial.send_data(cmd2)
152         time.sleep(0.5)
153
154     def tecla_5(self):
155         cmd = 5
156         cmd2 = b''
157         cmd2 += struct.pack('!B', int(cmd))
158         self.serial.send_data(cmd2)
159         time.sleep(0.5)
160
161     def tecla_6(self):
162         cmd = 6
163         cmd2 = b''
164         cmd2 += struct.pack('!B', int(cmd))
165         self.serial.send_data(cmd2)
166         time.sleep(0.5)
167
168     def tecla_7(self):
169         cmd = 7
170         cmd2 = b''
171         cmd2 += struct.pack('!B', int(cmd))
172         self.serial.send_data(cmd2)
173         time.sleep(0.5)
174
175     def tecla_8(self):
176         cmd = 8
177         cmd2 = b''
178         cmd2 += struct.pack('!B', int(cmd))
179         self.serial.send_data(cmd2)
180         time.sleep(0.5)
181
182     def tecla_9(self):
183         cmd = 9
184         cmd2 = b''
185         cmd2 += struct.pack('!B', int(cmd))
186         self.serial.send_data(cmd2)
187         time.sleep(0.5)
188
189     def tecla_10(self):
190         cmd = 10
191         cmd2 = b''
192         cmd2 += struct.pack('!B', int(cmd))
193         self.serial.send_data(cmd2)
194         time.sleep(0.5)
195
196     def tecla_11(self):
197         cmd = 11
```

```
198             cmd2 = b''
199             cmd2 += struct.pack('!B', int(cmd))
200             self.serial.send_data(cmd2)
201             time.sleep(0.5)
202 
203         def tecla_12(self):
204             cmd = 10
205             cmd2 = b''
206             cmd2 += struct.pack('!B', int(cmd))
207             self.serial.send_data(cmd2)
208             time.sleep(0.5)
209 
210         def tecla_13(self):
211             cmd = 13
212             cmd2 = b''
213             cmd2 += struct.pack('!B', int(cmd))
214             self.serial.send_data(cmd2)
215             time.sleep(0.5)
216 
217         def tecla_14(self):
218             cmd = 14
219             cmd2 = b''
220             cmd2 += struct.pack('!B', int(cmd))
221             self.serial.send_data(cmd2)
222             time.sleep(0.5)
223 
224         def update_ports(self):
225             self.serial.update_ports()
226             self.ui.portList.clear()
227             self.ui.portList.addItems(self.serial.portList)
228 
229         def clear_terminal(self):
230             self.ui.Terminal.clear()
231 
232         def closeEvent(self, e):
233             self.serial.disconnect_serial()
234 
235 
236 if __name__ == '__main__':
237     app = QApplication(sys.argv)
238     w = MiApp()
239     w.show()
240     sys.exit(app.exec_())
```

Código B.1: Arquivo *main.py*, adaptado de "Simple-Serial-PyQt5"[1]

B.1.2 *customSerial.py*

```
1 import serial, serial.tools.list_ports
```

```
2  from threading import Thread, Event
3  from PyQt5.QtCore import QObject, pyqtSignal, pyqtSlot
4  import time
5
6  class customSerial(QObject):
7      data_available = pyqtSignal(str)
8
9      def __init__(self):
10         super().__init__()
11         self.serialPort = serial.Serial()
12         self.serialPort.timeout = 0.5
13
14         self.baudratesDIC = {
15             '1200': 1200,
16             '2400': 2400,
17             '4800': 4800,
18             '9600': 9600,
19             '19200': 19200,
20             '38400': 38400,
21             '57600': 57600,
22             '115200': 115200
23         }
24         self.portList = []
25
26         self.thread = None
27         self.alive = Event()
28
29     def update_ports(self):
30         self.portList = [port.device for port in serial.tools.list_ports.comports
       ()]
31         print(self.portList)
32
33     def connect_serial(self):
34         try:
35             self.serialPort.open()
36         except:
37             print("ERROR SERIAL")
38
39         if (self.serialPort.is_open):
40             self.start_thread()
41
42     def disconnect_serial(self):
43         self.stop_thread()
44         self.serialPort.close()
45
46     def read_serial(self):
47         while (self.alive.isSet() and self.serialPort.is_open):
48             name = time.strftime("%d-%m-%y---%H-%M-%S")
49             name = name + ".txt"
50             textfile = open(name, 'w')
51
52             # Lendo...
```

```
53              while (self.alive.isSet() and self.serialPort.is_open):
54
55                  data = self.serialPort.readline().decode("ISO-8859-1").strip()
56
57                  textfile.write(data + '\n')
58
59                  if (len(data) > 1):
60                      self.data_available.emit(data)
61
62      def send_data(self, data):
63          if (self.serialPort.is_open):
64              message = str(data) + "\n"
65              self.serialPort.write(message.encode())
66
67      def start_thread(self):
68          self.thread = Thread(target=self.read_serial)
69          self.thread.setDaemon(1)
70          self.alive.set()
71          self.thread.start()
72
73      def stop_thread(self):
74          if (self.thread is not None):
75              self.alive.clear()
76              self.thread.join()
77              self.thread = None
```

Código B.2: Arquivo *customSerial.py*, adaptado de "Simple-Serial-PyQt5"[1]

B.1.3 *GUI.py*

```
1  # -*- coding: utf-8 -*-
2
3  # Form implementation generated from reading ui file 'GUI.ui'
4  #
5  # Created by: PyQt5 UI code generator 5.15.4
6  #
7  # WARNING: Any manual changes made to this file will be lost when pyuic5 is
8  # run again.  Do not edit this file unless you know what you are doing.
9
10
11 from PyQt5 import QtCore, QtGui, QtWidgets
12
13
14 class Ui_MainWindow(object):
15     def setupUi(self, MainWindow):
16         MainWindow.setObjectName("MainWindow")
17         MainWindow.resize(1111, 845)
18         MainWindow.setStyleSheet("#connectBtn:checked{\n"
19 "background-color: rgb(115, 210, 22);\n"
```

```
"}\n"
"\n"
"#botaoGrava{\n"
"background-color: rgb(255, 0, 0);\n"
"}\n"
"\n"
"#botaoTeste:checked{\n"
"    background-color: rgb(85, 0, 255);\n"
"}\n"
"\n"
"#botaoOpera:checked{\n"
"    background-color: rgb(85, 0, 255);\n"
"}")
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.verticalLayout_3 = QtWidgets.QVBoxLayout(self.centralwidget)
        self.verticalLayout_3.setObjectName("verticalLayout_3")
        self.label = QtWidgets.QLabel(self.centralwidget)
        self.label.setAlignment(QtCore.Qt.AlignCenter)
        self.label.setObjectName("label")
        self.verticalLayout_3.addWidget(self.label)
        self.horizontalLayout_5 = QtWidgets.QHBoxLayout()
        self.horizontalLayout_5.setObjectName("horizontalLayout_5")
        self.verticalLayout_2 = QtWidgets.QVBoxLayout()
        self.verticalLayout_2.setObjectName("verticalLayout_2")
        self.horizontalLayout_6 = QtWidgets.QHBoxLayout()
        self.horizontalLayout_6.setObjectName("horizontalLayout_6")
        self.clearBtn = QtWidgets.QPushButton(self.centralwidget)
        self.clearBtn.setObjectName("clearBtn")
        self.horizontalLayout_6.addWidget(self.clearBtn)
        spacerItem = QtWidgets.QSpacerItem(40, 20, QtWidgets.QSizePolicy.Expanding
    , QtWidgets.QSizePolicy.Minimum)
        self.horizontalLayout_6.addItem(spacerItem)
        self.verticalLayout_2.addLayout(self.horizontalLayout_6)
        self.Terminal = QtWidgets.QTextBrowser(self.centralwidget)
        self.Terminal.setObjectName("Terminal")
        self.verticalLayout_2.addWidget(self.Terminal)
        self.horizontalLayout_4 = QtWidgets.QHBoxLayout()
        self.horizontalLayout_4.setObjectName("horizontalLayout_4")
        self.input = QtWidgets.QLineEdit(self.centralwidget)
        self.input.setObjectName("input")
        self.horizontalLayout_4.addWidget(self.input)
        self.sendBtn = QtWidgets.QPushButton(self.centralwidget)
        self.sendBtn.setObjectName("sendBtn")
        self.horizontalLayout_4.addWidget(self.sendBtn)
        self.verticalLayout_2.addLayout(self.horizontalLayout_4)
        self.horizontalLayout_5.addLayout(self.verticalLayout_2)
        self.verticalLayout = QtWidgets.QVBoxLayout()
        self.verticalLayout.setObjectName("verticalLayout")
        self.horizontalLayout = QtWidgets.QHBoxLayout()
        self.horizontalLayout.setObjectName("horizontalLayout")
        self.label_2 = QtWidgets.QLabel(self.centralwidget)
```

```
71          self.label_2.setObjectName("label_2")
72          self.horizontalLayout.addWidget(self.label_2)
73          self.portList = QtWidgets.QComboBox(self.centralwidget)
74          self.portList.setObjectName("portList")
75          self.horizontalLayout.addWidget(self.portList)
76          self.verticalLayout.addLayout(self.horizontalLayout)
77          self.horizontalLayout_2 = QtWidgets.QHBoxLayout()
78          self.horizontalLayout_2.setObjectName("horizontalLayout_2")
79          self.label_3 = QtWidgets.QLabel(self.centralwidget)
80          self.label_3.setObjectName("label_3")
81          self.horizontalLayout_2.addWidget(self.label_3)
82          self.BaudList = QtWidgets.QComboBox(self.centralwidget)
83          self.BaudList.setObjectName("BaudList")
84          self.horizontalLayout_2.addWidget(self.BaudList)
85          self.verticalLayout.addLayout(self.horizontalLayout_2)
86          self.horizontalLayout_3 = QtWidgets.QHBoxLayout()
87          self.horizontalLayout_3.setObjectName("horizontalLayout_3")
88          self.connectBtn = QtWidgets.QPushButton(self.centralwidget)
89          self.connectBtn.setCheckable(True)
90          self.connectBtn.setObjectName("connectBtn")
91          self.horizontalLayout_3.addWidget(self.connectBtn)
92          self.updateBtn = QtWidgets.QPushButton(self.centralwidget)
93          self.updateBtn.setObjectName("updateBtn")
94          self.horizontalLayout_3.addWidget(self.updateBtn)
95          self.verticalLayout.addLayout(self.horizontalLayout_3)
96          self.botaoTeste = QtWidgets.QPushButton(self.centralwidget)
97          self.botaoTeste.setCheckable(True)
98          self.botaoTeste.setChecked(True)
99          self.botaoTeste.setObjectName("botaoTeste")
100         self.verticalLayout.addWidget(self.botaoTeste)
101         self.botaoOpera = QtWidgets.QPushButton(self.centralwidget)
102         self.botaoOpera.setCheckable(True)
103         self.botaoOpera.setObjectName("botaoOpera")
104         self.verticalLayout.addWidget(self.botaoOpera)
105         self.botaoOpcoes = QtWidgets.QPushButton(self.centralwidget)
106         self.botaoOpcoes.setStyleSheet("background-color: rgb(85, 170, 255);")
107         self.botaoOpcoes.setObjectName("botaoOpcoes")
108         self.verticalLayout.addWidget(self.botaoOpcoes)
109         self.line_2 = QtWidgets.QFrame(self.centralwidget)
110         self.line_2.setStyleSheet("color: rgb(0, 0, 0);")
111         self.line_2.setFrameShape(QtWidgets.QFrame.HLine)
112         self.line_2.setFrameShadow(QtWidgets.QFrame.Sunken)
113         self.line_2.setObjectName("line_2")
114         self.verticalLayout.addWidget(self.line_2)
115         self.botaoInterromp = QtWidgets.QPushButton(self.centralwidget)
116         self.botaoInterromp.setStyleSheet("background-color: rgb(255, 0, 0);")
117         self.botaoInterromp.setObjectName("botaoInterromp")
118         self.verticalLayout.addWidget(self.botaoInterromp)
119         self.line = QtWidgets.QFrame(self.centralwidget)
120         self.line.setStyleSheet("color: rgb(0, 0, 0);")
121         self.line.setFrameShape(QtWidgets.QFrame.HLine)
122         self.line.setFrameShadow(QtWidgets.QFrame.Sunken)
```

```
123         self.line.setObjectName("line")
124         self.verticalLayout.addWidget(self.line)
125         self.botaoNum1 = QtWidgets.QPushButton(self.centralwidget)
126         self.botaoNum1.setObjectName("botaoNum1")
127         self.verticalLayout.addWidget(self.botaoNum1)
128         self.botaoNum2 = QtWidgets.QPushButton(self.centralwidget)
129         self.botaoNum2.setObjectName("botaoNum2")
130         self.verticalLayout.addWidget(self.botaoNum2)
131         self.botaoNum3 = QtWidgets.QPushButton(self.centralwidget)
132         self.botaoNum3.setObjectName("botaoNum3")
133         self.verticalLayout.addWidget(self.botaoNum3)
134         self.botaoNum4 = QtWidgets.QPushButton(self.centralwidget)
135         self.botaoNum4.setObjectName("botaoNum4")
136         self.verticalLayout.addWidget(self.botaoNum4)
137         self.botaoNum5 = QtWidgets.QPushButton(self.centralwidget)
138         self.botaoNum5.setObjectName("botaoNum5")
139         self.verticalLayout.addWidget(self.botaoNum5)
140         self.botaoNum6 = QtWidgets.QPushButton(self.centralwidget)
141         self.botaoNum6.setObjectName("botaoNum6")
142         self.verticalLayout.addWidget(self.botaoNum6)
143         self.botaoNum7 = QtWidgets.QPushButton(self.centralwidget)
144         self.botaoNum7.setObjectName("botaoNum7")
145         self.verticalLayout.addWidget(self.botaoNum7)
146         self.botaoNum8 = QtWidgets.QPushButton(self.centralwidget)
147         self.botaoNum8.setObjectName("botaoNum8")
148         self.verticalLayout.addWidget(self.botaoNum8)
149         self.botaoNum9 = QtWidgets.QPushButton(self.centralwidget)
150         self.botaoNum9.setObjectName("botaoNum9")
151         self.verticalLayout.addWidget(self.botaoNum9)
152         self.botaoNum10 = QtWidgets.QPushButton(self.centralwidget)
153         self.botaoNum10.setObjectName("botaoNum10")
154         self.verticalLayout.addWidget(self.botaoNum10)
155         self.botaoNum11 = QtWidgets.QPushButton(self.centralwidget)
156         self.botaoNum11.setObjectName("botaoNum11")
157         self.verticalLayout.addWidget(self.botaoNum11)
158         self.botaoNum12 = QtWidgets.QPushButton(self.centralwidget)
159         self.botaoNum12.setObjectName("botaoNum12")
160         self.verticalLayout.addWidget(self.botaoNum12)
161         self.botaoNum13 = QtWidgets.QPushButton(self.centralwidget)
162         self.botaoNum13.setObjectName("botaoNum13")
163         self.verticalLayout.addWidget(self.botaoNum13)
164         self.botaoNum14 = QtWidgets.QPushButton(self.centralwidget)
165         self.botaoNum14.setObjectName("botaoNum14")
166         self.verticalLayout.addWidget(self.botaoNum14)
167         spacerItem1 = QtWidgets.QSpacerItem(20, 40, QtWidgets.QSizePolicy.Minimum,
    QtWidgets.QSizePolicy.Expanding)
168         self.verticalLayout.addItem(spacerItem1)
169         self.horizontalLayout_5.addLayout(self.verticalLayout)
170         self.verticalLayout_3.addLayout(self.horizontalLayout_5)
171         MainWindow.setCentralWidget(self.centralwidget)
172         self.menubar = QtWidgets.QMenuBar(MainWindow)
173         self.menubar.setGeometry(QtCore.QRect(0, 0, 1111, 26))
```

```
174         self.menubar.setObjectName("menubar")
175         MainWindow.setMenuBar(self.menubar)
176         self.statusbar = QtWidgets.QStatusBar(MainWindow)
177         self.statusbar.setObjectName("statusbar")
178         MainWindow.setStatusBar(self.statusbar)
179 
180         self.retranslateUi(MainWindow)
181         QtCore.QMetaObject.connectSlotsByName(MainWindow)
182 
183     def retranslateUi(self, MainWindow):
184         _translate = QtCore.QCoreApplication.translate
185         MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow"))
186         self.label.setText(_translate("MainWindow", "TERMINAL SERIAL"))
187         self.clearBtn.setText(_translate("MainWindow", "Limpar"))
188         self.sendBtn.setText(_translate("MainWindow", "ENVIAR"))
189         self.label_2.setText(_translate("MainWindow", "Port:"))
190         self.label_3.setText(_translate("MainWindow", "Baudrate:"))
191         self.connectBtn.setText(_translate("MainWindow", "CONECTAR"))
192         self.updateBtn.setText(_translate("MainWindow", "Atualizar Portas"))
193         self.botaoTeste.setText(_translate("MainWindow", "TESTE"))
194         self.botaoOpera.setText(_translate("MainWindow", "OPERA"))
195         self.botaoOpcoes.setText(_translate("MainWindow", "Op   es"))
196         self.botaoInterromp.setText(_translate("MainWindow", "Interromper"))
197         self.botaoNum1.setText(_translate("MainWindow", "LEDs"))
198         self.botaoNum2.setText(_translate("MainWindow", "LCD"))
199         self.botaoNum3.setText(_translate("MainWindow", "Teclado"))
200         self.botaoNum4.setText(_translate("MainWindow", "TWI (I2C)"))
201         self.botaoNum5.setText(_translate("MainWindow", "Acel e Giro"))
202         self.botaoNum6.setText(_translate("MainWindow", "Magnetometro"))
203         self.botaoNum7.setText(_translate("MainWindow", "SRAM"))
204         self.botaoNum8.setText(_translate("MainWindow", "FLASH"))
205         self.botaoNum9.setText(_translate("MainWindow", "GPS: Tudo"))
206         self.botaoNum10.setText(_translate("MainWindow", "GPS: Interpreta"))
207         self.botaoNum11.setText(_translate("MainWindow", "GPS: U-Center"))
208         self.botaoNum12.setText(_translate("MainWindow", "MPU-->MatLab"))
209         self.botaoNum13.setText(_translate("MainWindow", "Bluetooth"))
210         self.botaoNum14.setText(_translate("MainWindow", "BT - Cmds AT"))
```

Código B.3: Arquivo *GUI.py*, adaptado de "Simple-Serial-PyQt5"[1]