UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

DEPARTAMENTO ACADÊMICO DE ENGENHARIA ELETRÔNICA

BACHARELADO EM ENGENHARIA ELETRÔNICA

FELIPE ADALBERTO FARINELLI

DESENVOLVIMENTO DE UM SISTEMA EMBARCADO DE BORDO PARA MONITORAMENTO DE PARÂMETROS VIA DIAGNÓSTICO DE REDE CAN

TRABALHO DE CONCLUSÃO DE CURSO

PONTA GROSSA

2016

FELIPE ADALBERTO FARINELLI

DESENVOLVIMENTO DE UM SISTEMA EMBARCADO DE BORDO PARA MONITORAMENTO DE PARÂMETROS VIA DIAGNÓSTICO DE REDE CAN

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Bacharel, do Departamento Acadêmico de Engenharia Eletrônica da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Sergio Luiz Stevan Jr.

PONTA GROSSA

2016

TERMO DE APROVAÇÃO

DESENVOLVIMENTO DE UM SISTEMA EMBARCADO DE BORDO PARA MONITORAMENTO DE PARÂMETROS VIA DIAGNÓSTICO DE REDE CAN

por

FELIPE ADALBERTO FARINELLI

Este Trabalho de Conclusão de Curso (TCC) foi apresentado em 18 de novembro de 2016 como requisito parcial para a obtenção do título de Bacharel em Engenharia Eletrônica. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____

SERGIO LUIZ STEVAN JUNIOR
Prof. Orientador

_____

HUGO VALADARES SIQUEIRA
Membro titular

_____

FERNANDA CRISTINA CORREA
Membro titular

- A Folha de Aprovação assinada encontra-se arquivada na Secretaria Acadêmica -

Dedico este trabalho à minha família,
como forma de compensação aos meus
momentos ausente.

.

AGRADECIMENTOS

É evidente que esses parágrafos são insuficientes para agradecer a todas as pessoas que de alguma forma fizeram parte da evolução do meu conhecimento, portanto desde já me desculpo pelos que aqui não citei, mas mesmo assim podem ter certeza que foram de grande valia para que eu conseguisse chegar até aqui.

Agradeço a Deus por todo o esforço e dedicação de me concedeu durante o desenvolvimento desse trabalho, bem como de toda minha vida e conquistas acadêmicas.

Agradeço aos meus familiares pela compreensão e incentivo para que eu chegasse onde estou e por todas as oportunidades a mim concedidas.

Agradeço à minha namorada por me dirigir as melhores palavras nos momentos mais adequados.

Agradeço ao meu Professor Orientador Dr. Sergio Luiz Stevan Junior por ser sempre franco com relação aos meus trabalhos e fazer com que eu melhorasse a cada dia.

Agradeço aos meus colegas do Grupo de Sistemas Automotivos por todas as ideias e opiniões, bem como pelo enorme conhecimento compartilhado comigo.

Enfim, a todos que de alguma forma foram importantes no decorrer do desenvolvimento desse trabalho.

RESUMO

FARINELLI, Felipe Adalberto. **Desenvolvimento De Um Sistema Embarcado De Bordo Para Monitoramento De Parâmetros Via Diagnóstico De Rede CAN.** 2016. 104 f. Trabalho de Conclusão de Curso (Bacharelado em Engenharia Eletrônica) - Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2016.

Os itens optativos presentes nos veículos fabricados no Brasil são limitados nas versões mais básicas, logo não são todas as pessoas que possuem acesso às tecnologias presentes nos veículos mais modernos. Uma vez que muitos veículos saem das fábricas sem computadores de bordo, item presente em carros de versões com custo mais elevado, foi proposto nesse trabalho a construção de um aparelho que disponibiliza algumas informações do veículo tais como a temperatura do liquido de arrefecimento, o nível de combustível, a velocidade, a temperatura ambiente, entre outros. Foram escolhidos dispositivos de prototipagem rápida para a realização de testes e para o desenvolvimento do equipamento final, bem como todos os módulos necessários para que se tornasse possível uma comunicação entre o dispositivo e a rede automotiva. Foi necessário que se compreende-se alguns protocolos de comunicação entre microcontroladores, como o SPI (*Serial Peripheral Interface*) e entre o dispositivo desenvolvido e o veículo (CAN, ou *Controller Area Network*), que por fim foi acondicionado em uma caixa de madeira e instalado provisoriamente no automóvel teste a fim de possibilitar a avaliação e a validação do equipamento desenvolvido.

Palavras-chave: Computador de Bordo. Redes CAN. Comunicação SPI. Diagnóstico automotivo.

ABSTRACT

FARINELLI, Felipe Adalberto. **Development of an Onboard Embedded System for parameter monitoring by way of CAN Network Diagnosis.** 2016. 104 p. Work of Conclusion Course (Graduation in Electronics Engineering) - Federal Technology University - Paraná. Ponta Grossa, 2016.

The optative items in Brazilian vehicles are limited on basic versions; therefore, it is not all people that can access technologies present in newest cars. Once many vehicles does not come out from factory with an Onboard Computer, which is present in its expensive models, it has been proposed the construction of a dispositive that shows some information about the car, like engine coolant temperature, fuel tank level, vehicle instant speed, external temperature and some other important information. It were chosen rapid prototyping platforms for testing and for the development of the final equipment, as well as all-necessary modules for being possible the communication between the developed device and the car's network. It was necessary to understand some communication protocols between microcontrollers, like SPI (*Serial Peripheral Interface*) and between the developed device and the test vehicle (CAN, or *Controller Area Network*). Finally, the device were conditioned in a wooden box and provisionally installed in the test vehicle, in order to evaluate and validate the developed equipment.

Keywords: Onboard Computer. CAN Networks. SPI Communication. Automotive Diagnosis.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

O computador de bordo é um dispositivo que está em desenvolvimento no Brasil desde os anos 1990, ano em que foi proposta uma das primeiras arquiteturas nacionais para o monitoramento de informações importantes de parâmetros internos de locomotivas (PINTO; AMARAL; FILHO, 1990), que se dava através de um *display* alfanumérico acoplado à cabine. Junto ao computador dentro da locomotiva, era possível realizar o levantamento de informações de viagem, pois além de disponibilizar dados ao maquinista em tempo real, o dispositivo as armazenava em uma memória não volátil para que essas fossem transferidas a um computador externo posteriormente e então fosse possível realizar a análise da viagem.

Com o passar dos anos, o desenvolvimento nacional dos computadores de bordo se estenderam também à área automotiva, visto que nos anos 2000 foi proposto um dos primeiros voltados a visualização de parâmetros veiculares em tempo real patenteados no Brasil (MARTINS, 2000), cujo acesso às informações de medição do veículo era realizado por sensores montados separadamente. Uma vez que se obtinha as informações, era possível medir, armazenar e controlar os elementos do veículo. As informações eram mostradas em um *display* simples de cristal líquido dotado de duas linhas, cada uma com 16 caracteres. Logo, nota-se a limitação na exibição instantânea de parâmetros devido ao pequeno espaço provido pela tela.

Mais tarde, em 2007, foi patenteada uma das primeiras tecnologias brasileiras capazes de monitorar informações em uma motocicleta (JUNIOR; RIBEIRO, 2007), porém ainda de forma indireta visto que era necessária a instalação de sensores externos.

Mesmo com a necessidade de instalação de outros elementos, o computador microcontrolado para motocicletas era capaz de disponibilizar ao seu proprietário informações referentes ao trajeto percorrido total ou apenas de viagem, tais como a velocidade, a rotação do motor e outras funções interessantes como alarme de velocidade programado, velocidade média do percurso e o tempo total pelo qual o motor da motocicleta permaneceu em funcionamento.

Aparelhos voltados a essa área passaram a ser aliados do motorista no auxílio à condução, uma vez que foi proposto em 2008 outro dispositivo com o propósito de auxiliar na manutenção do veículo (LIMA, 2008), mantendo o motorista atualizado

quanto a possíveis danos no sistema do veículo e também retornando as informações já citadas anteriormente.

Sistemas de monitoramento passaram a ser amplamente utilizados, devido a facilidade com a qual uma central pode auxiliar na observação de sistemas complexos. Houveram propostas de computadores para até a observação de condomínios (LIMA; SILVA, 2008), possibilitando o gerenciamento de áreas úteis, o monitoramento referente a segurança e a todos os sensores e atuadores disponibilizados dentro do lugar.

Já em 2013, uma proposta importante quanto a integração de vários dispositivos através de um computador de bordo foi apresentada (QUENEHEN, 2013), integrando funções que vão desde leitura biométrica até a identificação do consumo de bebidas alcoólicas por parte do motorista.

A título de aplicação, já existem dispositivos de bordo que possuem a capacidade de auxiliar o motorista a evitar uma colisão quando ocorre qualquer tipo de movimento brusco na frente do veículo. Esta aplicação sugere que um computador de bordo pode ir além de apenas mostrar informações importantes ao motorista.

Nota-se que até então foram citadas algumas alternativas quanto a visualização de parâmetros do veículo. Uma vez que a evolução da tecnologia veicular se mostrou acentuada na década de 80 e até hoje ainda há notáveis sinais desta no mercado, foi introduzido, ainda na década de 80, um sistema conhecido como OBD (*On-board Diagnostics*). Este foi desenvolvido inicialmente pela *General Motors* e é empregado nos veículos da marca a partir de então (POSADA, 2015).

Nos dias de hoje, o sistema OBD está em sua segunda versão, o qual foi definido como padrão e começou a ser adotado mundialmente a partir de 1996, quando se tornou obrigatório no Estados Unidos (POSADA, 2015). O sistema tornou possível a obtenção de várias informações quanto ao estado do veículo, a velocidade na qual ele se encontra, a velocidade de rotação do motor e a temperatura do líquido de arrefecimento, responsável por manter o motor resfriado e adequado ao uso.

O acesso à interface de comunicação entre sensores do veículo através de ferramentas externas é dificultado pelos fabricantes por razões óbvias de concorrência comercial (ENRIQUEZ et al., 2012), o que resulta em uma certa dificuldade de se obter informações mais específicas de um veículo para exibir ao motorista através de uma

conexão direta à interface do carro, sem que exista uma certa normalização das informações entre os fabricantes.

A padronização OBD surgiu com o intuito de disponibilizar informações importantes do veículo de uma maneira mais acessível e normatizada, a fim de tornar mais simples o diagnóstico de informações veiculares. Isto possibilitou o desenvolvimento de diversas tecnologias para a realização de diagnóstico automotivo exemplificado através do desenvolvimento de microcontroladores próprios para trabalhos dedicados a esse fim, como o ELM327 fabricado pela *ELM® Electronics.* Este tem sido utilizado na realização da interface entre o veículo e dispositivos que utilizam informações de diagnóstico, como o exemplo no trabalho de Ambata et al. (2015), no qual a velocidade do veículo é pré-requisito para o cálculo de distâncias objetivado à prevenção de acidentes.

Uma vez que a comunicação com o veículo foi facilitada com o surgimento da interface ELM327, surgiram outras alternativas para monitoramento automotivo, como a interconexão entre veículos e celulares através de aplicativos desenvolvidos em linguagem Java (TAHAT et al., 2012) conectados a um dispositivo OBD, via *Bluetooth*, dotado do microcontrolador.

Nesta linha de coleta de informações e troca de dados, surgiram propostas de dispositivos que conectam o carro à internet através de um computador servidor e uma rede de dados móvel, responsável por tomar decisões quanto a ocorrência de alguma falha no veículo (JHOU et al., 2013), visando propor a melhor solução para o usuário.

Apesar dos sistemas mais complexos já citados, existem outras alternativas que consistem apenas de monitoramento de bordo automotivo, sem a necessidade de um processamento mais expressivo, mas com relevância em termos de apresentação de informações ao motorista. É o caso dos protótipos desenvolvidos por Park e Lee (2012) e Niazi et al. (2013) que apenas mostram ao usuário do veículo algumas das informações que o OBD pode fornecer de uma maneira simplificada através de um smartphone conectado à interface de diagnóstico, o que difere deste trabalho no qual foi proposto um dispositivo dedicado à mesma função.

Além de detalhar os dados do veículo ao usuário através de um celular com um aplicativo instalado, o equipamento proposto por Amarasinghe et al. (2015) consegue detectar anomalias na condução do motorista e alertá-lo, a fim de evitar

qualquer tipo de acidente que possa ocorrer devido a aquecimento excessivo do motor ou quedas bruscas no nível de combustível.

KHANAPURI et al. (2015) propôs um sistema de interação com o motorista que pudesse orientá-lo quanto a forma mais econômica e/ou de melhor desempenho, baseado em informações oriundas dos resultados da queima do combustível através da avaliação do sinal da sonda Lambda.

Mesmo com várias informações disponibilizadas no OBD do veículo, há alguns trabalhos que propõem a validação dessas através de sensores externos, como é o caso de um experimento desenvolvido por Ceuca et al. (2013) que validou a velocidade do veículo comparando-a com a apresentada por um sistema de posicionamento global (ou *Global Positioning System,* GPS).

Ainda quanto a dispositivos autônomos, Tessaro (2013) sugere uma série de sensores de bordo, acoplados a uma central, com o objetivo de reunir informações úteis em uma tela para auxiliar o motorista através de dados quanto a consumo de combustível, velocidade, rotação do motor e outras, utilizando apenas um microcontrolador para o processamento de toda a interface homem-máquina.

Uma vez que existem variados protocolos para diagnóstico automotivo, é importante a escolha de um deles para que se reduzam custos eliminando microcontroladores dedicados a essa função, como o ELM327 supracitado. Isso garante mais autonomia no desenvolvimento pois não há o uso de ferramentas fechadas, apesar de reduzir o campo de aplicação do dispositivo.

Um protocolo de comunicação importante, também utilizado pelo ELM327, surgido em meados dos anos 80 por Robert Bosch e empregado até hoje, é o CAN (*Controller Area Network*) que pode alcançar velocidades teóricas de até 1Mbit/s e possui baixo custo de implementação, visto que depende de apenas um par de fios trançado para transportar informações (LUGLI; SANTOS, 2009).

Dessa forma, Burje, Karande e Jagadale (2014) sugeriram um aparelho baseado em um microcontrolador já com o protocolo CAN embutido, que foi programado e conectado ao OBD e que se tornou capaz de exibir ao usuário, em uma tela de cristal líquido, algumas informações quanto ao estado de variáveis veiculares como temperaturas, velocidade e outras sem nenhum tipo de sensoriamento externo.

De forma geral, observou-se que todas as técnicas citadas até então obtêm dados do veículo, mas não as disponibilizam de uma maneira amigável ao usuário

final, uma vez que ou mostram suas informações em telas simples e pequenas ou dependem de um computador proprietário ou de smartphones para que sejam de fato úteis, sendo estes últimos, elementos de uso proibidos durante a direção pela legislação vigente no Brasil, segundo a lei 13.281/16.

Neste cenário, o aparelho proposto por este trabalho tem como objetivo apresentar de forma independente de sensores externos vários parâmetros de um veículo provenientes do barramento de diagnóstico CAN, os quais serão destacados ao longo do desenvolvimento.

Será apresentado um produto final personalizável e de interface amigável, exibida por uma tela LCD de 2,4” com 65000 cores atrelada à três botões de interação, para a realização de toda a configuração do dispositivo que funcionará utilizando uma tensão de alimentação proveniente da bateria do próprio veículo no qual o aparelho será instalado.

1.1 TEMA

Desenvolvimento de um protótipo para leitura e interpretação de informações úteis disponibilizadas na rede CAN de veículos através da utilização de placas de desenvolvimento microprocessadas, possibilitando ao usuário usar o dispositivo como um pequeno computador de bordo de fácil uso com aplicação à maioria dos veículos.

1.2 DELIMITAÇÃO DO TEMA

Neste trabalho será desenvolvido um protótipo de um sistema de bordo automotivo, desde a comunicação com a rede CAN de um veículo compatível até a exibição gráfica dos parâmetros em uma tela TFT LCD (*Thin-film-transistor Liquid Crystal Display*) colorida.

Devido à questões de padronização, o sistema será desenvolvido para ser utilizado em veículos fabricados a partir de 2010, quando o padrão OBD-II foi regularizado no Brasil junto a norma SAE J1979. Qualquer veículo fabricado antes desse período e dotado de rede de diagnóstico CAN, pode também estar adequado ao sistema, porém não há garantia de funcionamento em função da padronização mais atual utilizada.

Esse trabalho será limitado às normas SAE J1979 (2006), ISO 15765 (2006) e ISO 11898 (2003) por questões de tempo hábil ao desenvolvimento. Dessa forma, serão disponibilizadas as informações que o carro contiver e que forem legíveis dentro da rede CAN do veículo, o que possibilitará testes e geração de resultados que serão validados com base no painel de instrumentos do próprio veículo teste.

O sistema final poderá ser utilizado no veículo dentro das condições citadas para a visualização de velocidade, rotação do motor e outras condições hábeis à leitura, de acordo com as normas escolhidas para o desenvolvimento da ferramenta, que visa atingir diretamente o público de veículos básicos que não possuem computador de bordo.

1.3 PROBLEMA

Todos os veículos brasileiros dotados de barramento CAN possuem uma forma de acesso ao barramento de diagnóstico realizado através da norma SAE J1979, mas, em sua forma mais básica, não são vendidos juntos de um computador de bordo. Logo, a informação existe na rede de comunicação do veículo, mas não é apresentada em lugar nenhum por padrão, o que torna o dispositivo desenvolvido neste trabalho uma alternativa ao computador de bordo veicular vendido como item opcional na compra de um veículo.

1.4 PREMISSAS

Foi considerado que o equipamento monitorará veículos cuja interface de diagnóstico é compatível com o protocolo CAN. Dessa forma é possível ter a certeza de que o instrumento desenvolvido será útil e assertivo na leitura de dados do barramento.

O protótipo foi desenvolvido utilizando inicialmente a plataforma de código aberto Arduino®, logo a quantidade de bibliotecas de programação disponíveis é grande e foram amplamente utilizadas no decorrer da programação do protótipo, excetuando-se pela interface CAN que foi programada a baixo nível, entretanto, vislumbra-se a adequação de microcontroladores com requisitos automotivos mais adequados à aplicação, sem prejuízo do projeto aqui apresentado.

Uma vez que a plataforma Arduino® foi utilizada, além de bibliotecas, há inúmeras extensões de *hardware* disponíveis no mercado. Para este trabalho, foram utilizadas duas extensões, uma que possibilita a leitura do barramento CAN e uma tela, na qual foram mostradas as informações obtidas via diagnóstico.

Como base de desenvolvimento, as normas SAE J1979 (2006) e ISO 15765 (2006) foram amplamente utilizadas, visto que contém toda a normatização necessária para a leitura dos dados no barramento CAN dos veículos. Partiu-se do princípio que todas as mensagens e sinais CAN utilizados estavam disponíveis nessa norma.

1.5 OBJETIVOS GERAIS

O presente trabalho tem por objetivo geral o desenvolvimento de um protótipo de computador de bordo automotivo parametrizável.

1.6 OBJETIVOS ESPECÍFICOS

Estão listados a seguir todos os objetivos específicos que se desejam alcançar no decorrer do desenvolvimento do dispositivo.

a) Definir como funcionam as redes CAN de modo geral, de forma que fique possível entender como ocorre o tráfego de dados e como é realizada a leitura destes;
b) Realizar um estudo avançado de tudo que pode ser lido do barramento CAN através dos padrões estabelecidos pelas normas SAE J1979 e ISO 15765;
c) Encontrar a melhor alternativa para ler o barramento CAN através do conector OBD-II;
d) Implementar na plataforma Arduino® um programa capaz de realizar a leitura dos dados na rede CAN através de um módulo CAN compatível programado em baixo nível;
e) Estudo das bibliotecas de desenvolvimento para o Arduino® existentes para se trabalhar com telas TFT LCD;

f) Desenvolvimento do programa de decodificação dos dados lidos da CAN e de toda a interface homem-máquina (IHM), visando o fácil uso pelo usuário com opções de personalização;
g) Fabricação de uma embalagem própria para o protótipo, em MDF.

1.7 JUSTIFICATIVA

Uma vez que a inovação tecnológica surge, ocorre o aumento de custo no processo de fabricação dos veículos, o que acaba impactando na quantidade de itens opcionais presentes em modelos de veículos mais básicos. Esse tema foi escolhido por possibilitar que qualquer usuário possa instalar em seu carro, desde que compatível, um computador de bordo que disponibilizará informações úteis no momento da utilização do veículo.

Visa-se o desenvolvimento de uma alternativa interessante frente às disponíveis no mercado, com o objetivo de alcançar usuários comuns que possuem veículos relativamente novos, mas sem computador de bordo.

1.8 PROCEDIMENTOS METODOLÓGICOS

O protótipo desenvolvido trata-se de um aparelho para a leitura de dados da rede CAN de um veículo, realizada através de módulos específicos para a plataforma de desenvolvimento utilizada.

Antes do início do desenvolvimento do protótipo, foram levantadas informações sobre como ocorre o tráfego de dados dentro de redes CAN e como foi possível enviar e receber mensagens utilizando um módulo compatível com o protocolo. Logo, o conhecimento para a leitura e tratamento dos dados auxiliou no desenvolvimento do aparelho.

O procedimento seguinte consistiu na definição dos requisitos de funcionamento do sistema, que foram determinados com base nas necessidades do usuário. Foi realizado um projeto base da interface homem-máquina (IHM) em acordo com a proposta do trabalho, no qual foram definidas as principais telas do sistema junto às possibilidades iniciais de personalização deste.

Como veículo teste, foi utilizado um Volkswagen Gol Comfortline 2014, dotado de barramento CAN e compatível às normas anteriormente citadas e, após a definição das informações contidas no barramento explorado, foi desenvolvida a programação da plataforma de prototipagem para o desempenho das funções propostas.

Ao concluir a parte eletrônica do protótipo, foi construída uma carcaça em MDF com base na quantidade de módulos eletrônicos que foram utilizados e no espaço necessário para abrigar o protótipo. Da mesma forma, foi construído o cabo para a interface do carro com o equipamento, tendo por fim um aparelho completo e independente alimentado pela própria bateria do veículo.

1.9 ORGANIZAÇÃO DO TRABALHO

Este trabalho está subdividido em três partes principais: o referencial teórico apresentado no segundo capítulo, toda a seção de desenvolvimento mostrada no terceiro capítulo e os resultados detalhados pelo quarto capítulo, onde é possível ver o dispositivo instalado e funcionando no veículo de testes.

No Capítulo 2, intitulado por Referencial Teórico e Aplicações Iniciais, são dadas todas as informações necessárias ao desenvolvimento, tais como todos os meios de comunicação envolvidos, normas utilizadas e procedimentos necessários para a obtenção dos dados.

Após todas as definições apresentadas no Capítulo 2, o Capítulo 3, intitulado como Desenvolvimento, apresenta todos os procedimentos adotados pelo idealizador do trabalho, bem como todas as ligações entre os componentes utilizados.

Uma vez desenvolvido, o trabalho foi testado no veículo e algumas comparações foram feitas, todas apresentadas no Capítulo 4, intitulado como Resultados e Discussões.

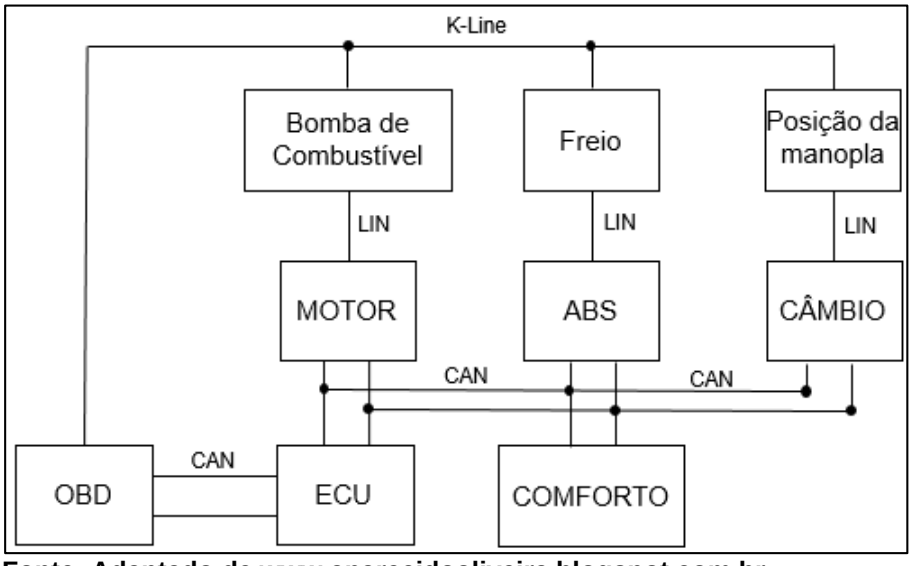
2 REFERENCIAL TEÓRICO E APLICAÇÕES INICIAIS

O elevado desenvolvimento de tecnologias automotivas levou à necessidade da criação de novos sistemas para a comunicação entre os módulos veiculares (ou ECUs, *Engine Control Unit*), devido ao aumento do tráfego de dados e ao custo elevado dos sistemas existentes para esse fim até então (BARBOSA, 2003). Dessa forma, foi necessário então definir um sistema compatível ao do veículo teste para a comunicação entre o dispositivo desenvolvido e o veículo.

Existem outros protocolos além do CAN para o diagnóstico automotivo, mesmo que este seja o principal em veículos mais novos, realizado através da porta OBD, como exemplo o LIN (*Local Interconnect Network*) e o K-line. Estas duas são redes de apenas um fio e de baixa velocidade, porém diferentes da CAN que pode operar a velocidades altas (SAE J1939, 2006) e trabalha com dois fios, detalhada no decorrer deste capítulo.

A presença de vários módulos de comunicação na estrutura veicular exige uma rede interna de tráfego de dados, conforme visto na Figura 1 onde nota-se a CAN como rede principal, K-line para a comunicação entre os sensores e o OBD e LIN para a comunicação dos sensores com as ECUs dedicadas. Dessa forma, qualquer dispositivo utilizado para o diagnóstico deve ser compatível com o protocolo utilizado para este fim.

Figura 1: Estrutura de uma rede de comunicação automotiva.



Fonte: Adaptado de www.aparecidooliveira.blogspot.com.br.

No caso do dispositivo desenvolvido neste trabalho, além do protocolo CAN necessário para a conexão deste com o veículo, foi usado outro protocolo, o SPI (*Serial Peripheral Interface*), que interligou todos os microcontroladores utilizados no desenvolvimento interno do equipamento.

Nota-se que, para o desenvolvimento do equipamento, foi importante definir todos os elementos necessários para a montagem do dispositivo. Para isso, foram escolhidos quatro elementos importantes e que necessitaram de um aprofundamento detalhado para a programação: ATMEGA328p e ATMEGA2560, microcontroladores presentes nas placas de desenvolvimento Arduino UNO e MEGA respectivamente, e TJA1050 e MCP2515, transceptor e microcontrolador, presentes no módulo CAN e responsáveis pela interface entre o dispositivo desenvolvido e o veículo teste.

Para a obtenção dos dados, são definidos parâmetros para diagnóstico normalizados pela SAE J1979 (2006) e ISO 15765 (2006), amplamente utilizadas no desenvolvimento do equipamento, cujo aprofundamento foi necessário para possibilitar a aquisição da maior quantidade de dados possível do veículo.

A fim de construir uma boa base para o desenvolvimento do trabalho, cada um dos elementos supracitados foram descritos em todos os seus detalhes e são apresentados no decorrer deste capítulo.

2.1 O PROTOCOLO DE REDE AUTOMOTIVA CAN

O protocolo de rede CAN (em inglês, *Controller Area Network*) surgiu em 1986, desenvolvido incialmente por Robert Bosch (LUGLI; SANTOS, 2009). Uma vez que é de implementação fácil e barata, foi adotado mundialmente na década de 80 para a comunicação de dispositivos veiculares e na automação industrial (FARSI; RATCLIFF; BARBOSA, 1999).

Por se tratar de uma rede de comunicação de dados, as redes CAN possuem classificação conforme a Pirâmide da Automação, muito conhecida por dividir as inúmeras redes industriais que existem no mercado em cinco níveis bem definidos, conforme mostra a Figura 2.

Figura 2: Pirâmide da Automação.

Fonte: adaptado de Harjunkoski, Nyström e Horch (2009).

Para exemplificar cada nível da pirâmide, é interessante citar alguns exemplos de redes industriais disponíveis no mercado e relacioná-los com a definição de cada nível. Dessa forma, pode-se definir cada um dos níveis conforme mostra a lista a seguir (HARJUNKOSKI; NYSTRÖM; HORCH, 2009).

- Nível 1, 2 e 3: Esses níveis compreendem os dispositivos de controle e as redes de comunicação voltadas apenas à transferência de dados. Nesses níveis estão compreendidos os dispositivos de processo, de campo e de controle (MERSCH; SCHLUTTER; EPPLE, 2010) e são exemplos de redes deste nível a CAN, ProfiBUS DP e PA, FieldBUS H1, HART e ASI (MORAES; CASTRUCCI, 2007);
- Nível 4: Neste nível, estão os dispositivos que auxiliam no planejamento da produção através do controle e da logística de suplementos. Dessa forma, aqui encontram-se computadores conectados aos dispositivos dos níveis inferiores funcionando através de protocolos Ethernet, MAC e TCP/IP (MORAES; CASTRUCCI, 2007);
- Nível 5: Este nível compreende os dispositivos responsáveis por gerir todo o sistema de automação que trabalha com programas conectados a dispositivos pelos mesmos protocolos do nível 4.

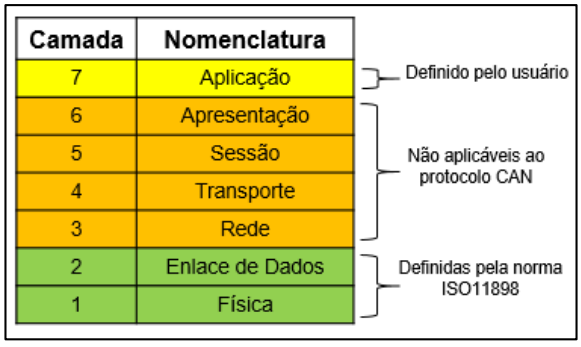
Sendo assim, nota-se que o barramento CAN se trata de uma rede compreendida na base da pirâmide, ou seja, é um protocolo de comunicação entre dispositivos de chão de fábrica ou, no caso da rede CAN veicular, entre ECUs (*Engine Control Unit*)

O protocolo CAN foi regularizado através da norma ISO11898 de 93/1994 pela Organização Internacional de Padronização (ISO, em inglês, *International Standardization Organization*) e, desde então, foi adotado como padrão mundial (LUGLI; SANTOS, 2009). Dessa forma, possui características bem definidas quanto a meio físico de transferência de dados, taxas de transmissão, detecção e correção de erros de comunicação e outras que serão definidas no decorrer deste trabalho.

2.1.1 O MODELO OSI DO PROTOCOLO CAN

O modelo OSI (em inglês, *Open System Interconnection*) é o modelo padrão para a definição de protocolos de comunicação constituído de 7 camadas nomeadas respectivamente como Física, Enlace de Dados, Rede, Transporte, Sessão, Apresentação e Aplicação (TANENBAUM, 2010). A Figura 3 mostra a organização do protocolo CAN dentro deste modelo.

Figura 3: O protocolo CAN dentro do modelo OSI



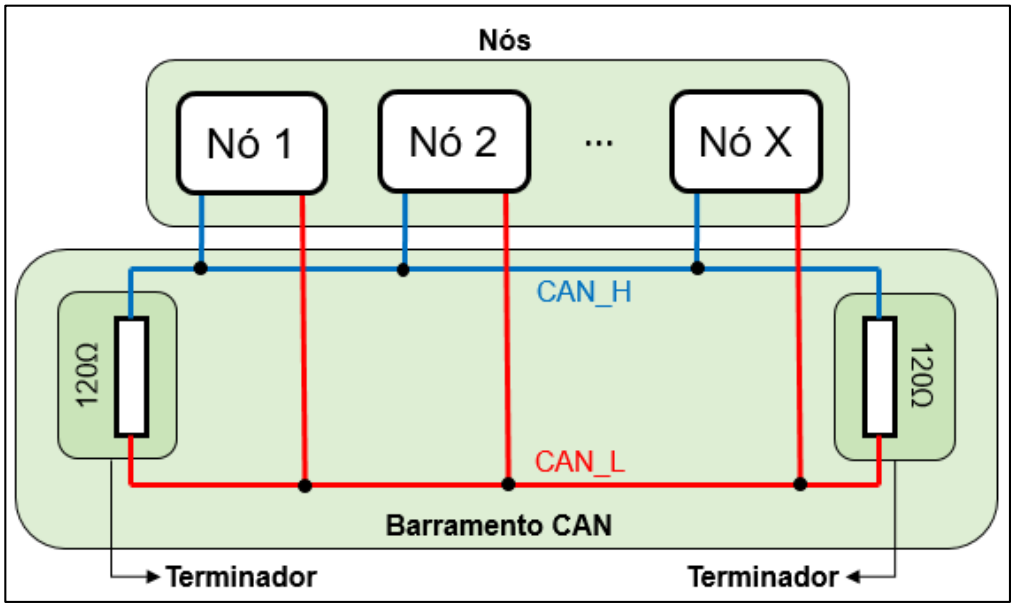
Fonte: Adaptado de Santos e Luigi (2009).

Sendo assim, é possível notar que a rede CAN está adepta aos níveis mais baixos da pirâmide da automação, cuja aplicação abrange uma enorme área a qual pode ser definida pelo usuário. Uma vez que tanto a camada de enlace de dados (também conhecida por *Data Link*, em inglês) quanto a física estão definidas pela norma ISO11898, então pode-se estudar tanto sua topologia, quanto as técnicas de tratamento de dados e de acesso ao barramento.

2.1.2 CAMADA FÍSICA E TOPOLOGIA

O meio de transmissão pelo qual os dados se propagam é um par de fios trançado, tornando a rede CAN mais barata que outras nas quais as vezes são necessários cabos blindados para a transmissão de dados (FARSI; RATCLIFF; BARBOSA, 1999). A Figura 4 mostra a topologia, exclusivamente em barramento, das redes CAN.

Figura 4: Topologia e componentes das redes CAN.



Fonte: Autoria Própria.

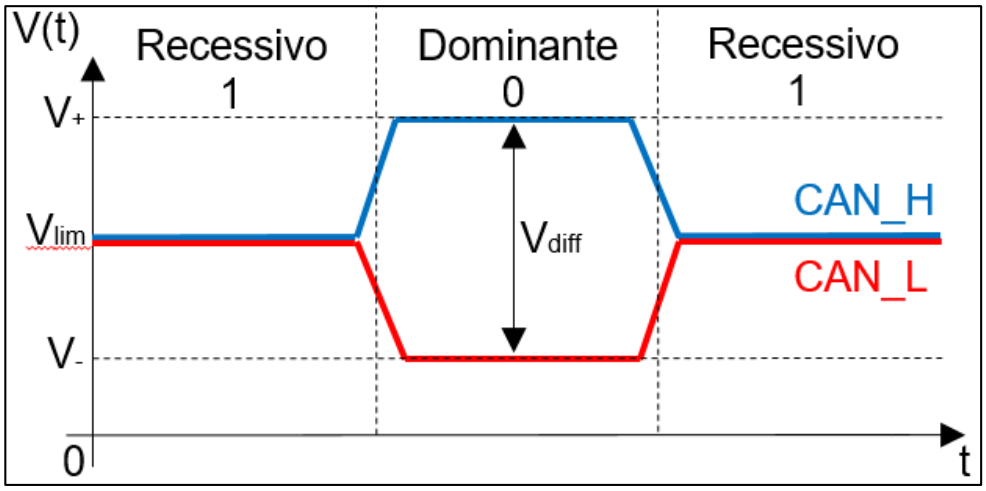
Devido as características da topologia, é possível conectar qualquer dispositivo à rede simplesmente criando uma ramificação, pois não é necessária a obstrução dos fios que compõem o meio de transmissão que possui capacidade para até 32 dispositivos (ou nós) compatíveis com o protocolo CAN (DI NATALE, 2009).

Uma vez criada a rede, é necessária a instalação de terminadores para evitar a reflexão de informações no barramento (RICHARDS, 2002), que pode causar a leitura incorreta ou até ilegibilidade dos dados pelos nós se a rede possuir baixa impedância. Para evitar esse efeito, de acordo com a norma ISO11898 (2003), devem ser utilizados resistores de 120Ω nas terminações do barramento CAN.

As duas linhas, CAN_H (em inglês *CAN High*, ou linha CAN de alto nível de tensão) e CAN_L (em inglês *CAN Low*, ou linha CAN de baixo nível de tensão),

representam o par de fios trançado pelo qual os dados trafegam, numa forma semelhante à da Figura 5.

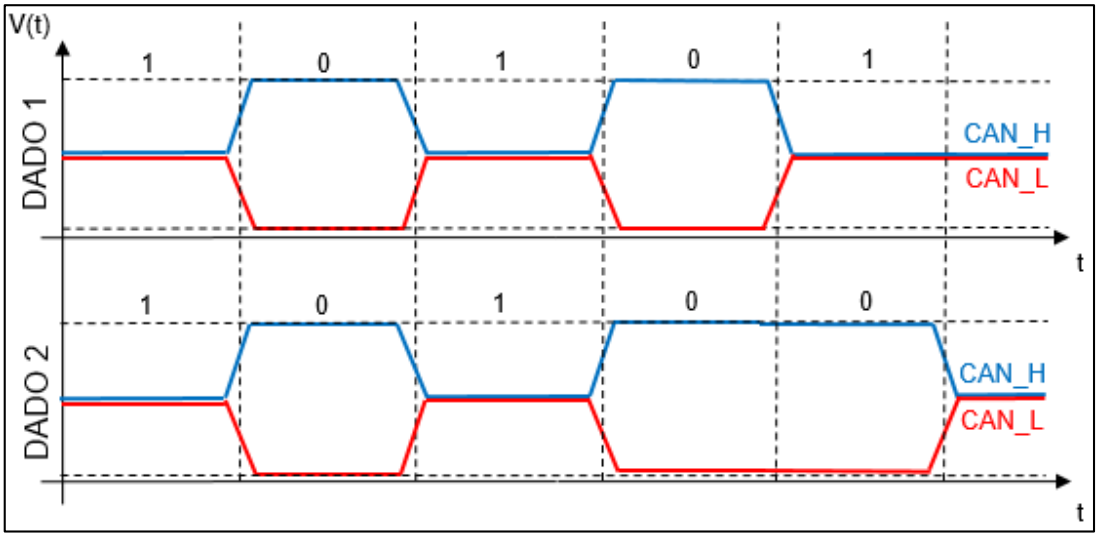
Figura 5: Formato de um bit no barramento CAN



Fonte: Autoria Própria.

De acordo com a norma ISO11898 (2003), a rede CAN é uma rede com prioridade de passagem de dados. Sendo assim, é considerado 1 como bit recessivo e 0 como bit dominante, em que a tensão diferencial V_{diff} é maior no bit dominante que no bit recessivo. Um exemplo de prioridade é apresentado pela Figura 6, pois o bit dominante prevalece no mesmo intervalo de tempo, o que faz com que o DADO 2 seja transmitido com prioridade sobre o DADO 1.

Figura 6: Exemplo de prioridade de transmissão de dados nas redes CAN.



Fonte: Autoria Própria.

A tensão de limiar, ou de modo comum, segundo a norma ISO11898, deve estar entre -1 e 6V, mas é recomendado 2,5V como valor nominal. Devido ao CAN se tratar de um barramento com tensão diferencial, então a mesma norma prevê valores mínimos e máximos para considerar um bit recessivo ou dominante através dos parâmetros apresentados na Tabela 1.

Tabela 1 - Valores mínimos e máximos para bits recessivos e dominantes no barramento.

Bit	Notação	Unidade	Valor		
			Mínimo	Nominal	Máximo
Recessivo	V_{CAN_L}	V	$V_{CC} - 0{,}3$	-	-
	V_{CAN_H}	V	-	-	$0{,}3$
	V_{diff}	V	$-V_{CC}$	-	$-V_{CC} + 0{,}6$
Dominante	V_{CAN_L}	V	-	-	$1{,}4$
	V_{CAN_H}	V	$V_{CC} - 1{,}4$	-	-
	V_{diff}	V	$V_{CC} - 2{,}8$	-	V_{CC}

Fonte: Adaptado de ISO11898-3 (2006).

A tensão contínua considerada para os cálculos deve ser de 5V, pois este deve ser o valor nominal segundo a norma. Sendo assim, para um bit ser recessivo, um nó de rede deve enviar uma tensão para CAN_L de no mínimo 4,7V e para CAN_H de no máximo 0,3V, cuja média de valores garante uma tensão de modo comum de 2,5V citada anteriormente. Já para um bit dominante, os valores máximo de CAN_L e mínimo de CAN_H passam a respectivamente 1,4V e 3,6V, com uma tensão diferencial de no mínimo 0V e no máximo 2,2V.

Uma vez que estão presentes no barramento duas tensões e terminadores resistivos, então a velocidade de transmissão pode ser alterada conforme impedância, a distância entre os nós e/ou com o comprimento total do barramento que pode ser definido como de baixa (até 125Kbits/s) ou de alta velocidade (maior que 125Kbits/s) (LAWRENZ, 2013). A Tabela 2 mostra quais as velocidades de transmissão e o comprimento máximo do barramento para cada uma delas.

Tabela 2: Taxa de transmissão conforme o comprimento do barramento CAN. (continua)

Taxa de Transferência (Kbits/s)	Distância Máxima (m)
5	10000
10	6700
20	3300
50	1300
100	620

Tabela 2: Taxa de transmissão conforme o comprimento do barramento CAN. (conclusão)

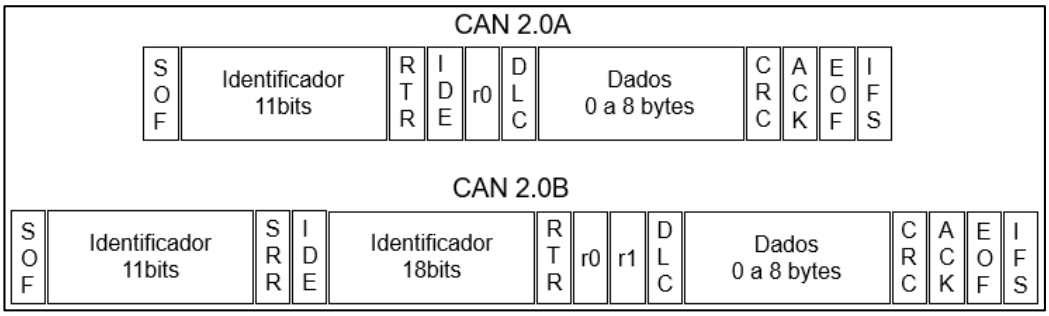
Taxa de Transferência (Kbits/s)	Distância Máxima (m)
125	530
250	270
500	130
1000	40

Fonte: Adaptado de Santos e Lugli (2009).

2.1.3 CAMADA *DATA LINK* OU DE ENLACE DE DADOS

É importante estabelecer a diferença quanto a dois tipos de protocolo CAN previstos em norma: 2.0A e 2.0B, pois cada um deles tratará os dados de uma forma diferente quanto a formação das mensagens (ou *frames,* em inglês) transmitidas no barramento. As mensagens possuem estrutura semelhante à mostrada na Figura 7 para o protocolo CAN 2.0A e 2.0B respectivamente, nas quais a primeira pode ser chamada de mensagem CAN padrão e a segunda de mensagem CAN estendida.

Figura 7 - Tipos de mensagens para cada um dos protocolos CAN: 2.0A e 2.0B.



Fonte: Adaptado de Santos e Lugli (2009).

Cada mensagem é composta de vários bits, que vão desde dados a identificação de início e fim, padrões do protocolo CAN e previstos na norma ISO11898. Dessa forma, a lista a seguir (LUGLI; SANTOS, 2009) mostra qual a função e as variantes de cada uma das partes da mensagem tanto para a mensagem estendida quanto para a mensagem padrão.

- SOF (*Start-of-Frame,* 1 bit): É o bit que identifica o início de uma mensagem do barramento e é o utilizado para a sincronização de todos os nós dentro do barramento;

- Identificador (11 bits para a mensagem padrão e 29 bits para a mensagem estendida): Esse campo contém o identificador (ID) da mensagem. Nele é definida a prioridade de cada mensagem, sabendo que quanto menor o identificador, maior é a prioridade;
- RTR (*Remote Transmission Request,* 1 bit): através desse bit é feita uma requisição de transmissão remota;
- SRR (*Substitute Remote Request*, 1 bit*)*: Presente apenas na mensagem estendida, esse bit sobrescreve o RTR de uma mensagem CAN padrão;
- IDE (*Identifier Extension,* 1 bit): quando esse bit é dominante (0), então significa que a mensagem é padrão, com um identificador de 11 bits apenas. Caso contrário, a mensagem possui um identificador extendido, de 29 bits.
- r0 (*Reserved bit,* 1 bit): É um bit reservado, utilizado para outros protocolos variantes do CAN.
- DLC (*Data Length Code*, 4 bit): O DLC é utilizado para a definição da quantidade de *bytes* de dados que serão transmitidos junto da mensagem.
- Dados (0 a 8 *bytes*, até 64 bits): É o conteúdo útil da mensagem, que carrega toda a informação transmitida de um nó a outro.
- CRC (*Cyclic Redundancy Check*, 16 bits): Nesse campo são armazenados todos os bits necessários para a técnica de detecção de erros CRC, capaz de detectar até seis bits errados.
- ACK (*Acknowledge*, 2 bits): Esses bits indicam se a mensagem foi recebida corretamente pelo nó de destino, dos quais o primeiro é o ACK propriamente dito e o segundo um delimitador.
- EOF (*End Of Frame*, 7 bits): Esses bits indicam o fim de uma mensagem dentro do barramento CAN.
- IFS (*Inter-Frame Space*, 7 bits): São sete bits sequenciais utilizados para demarcar um espaço entre duas mensagens. Esses bits são alocados conforme o tempo que o receptor necessita para armazenar a mensagem recebida em sua memória interna.
- r1 (*Reserved bit*, 1 bit): assim como o r0, esse é um bit reservado a outras aplicações, mas presente apenas na mensagem CAN do tipo 2.0B.

Uma vez diferenciados cada um dos campos de uma mensagem CAN, é necessário então aderir a um dos tipos, seja o 2.0A com identificador de 11 bits ou o

2.0B com identificador de 29 bits. Nesse caso, a escolha é baseada conforme a aplicação e, como durante esse trabalho serão utilizadas apenas mensagens de 11 bits (CAN 2.0A), é interessante detalhar alguns dos bits citados anteriormente.

2.1.3.1 IDENTIFICADOR (ID)

O identificador é um dos campos mais importantes da mensagem, pois é nele que será definida a prioridade com a qual esta será transmitida. Sendo assim, quanto menor o identificador maior a prioridade da mensagem, pois como já citado, é o bit dominante que a define. Na Tabela 3, como exemplo, são vistos quatro identificadores diferentes de quatro dados definidos arbitrariamente pelo autor.

Tabela 3: Exemplos de identificadores de dados escolhidos arbitrariamente.

Dado	Tipo	Identificador
Dado 1	Hexadecimal	1D1
	Decimal	465
	Binário (11bits)	00111010001
Dado 2	Hexadecimal	1C1
	Decimal	449
	Binário (11bits)	00111000001
Dado 3	Hexadecimal	7FF
	Decimal	2047
	Binário (11bits)	11111111111
Dado 4	Hexadecimal	0FF
	Decimal	255
	Binário (11bits)	00011111111

Fonte: Autoria Própria.

Uma vez definidos os identificadores, então pode-se estabelecer qual a prioridade de cada dado. A título de comparação, suponha que o Dado 1 e o Dado 2 representem duas mensagens: a primeira contém apenas a velocidade de rotação do motor e a segunda dados críticos sobre o mesmo componente, tais como temperatura, falha de algum sensor ou dano físico. Uma vez que o Dado 2 é mais importante que o Dado 1, então a prioridade deste deve ser maior dentro do barramento que pode conter até 2048 mensagens (11 bits) sendo transmitidas sequencialmente

(CORRIGAN, 2008).

Ainda utilizando o exemplo proposto na Tabela 3, nota-se definir que, conforme o tamanho decimal dos identificadores, a prioridade no barramento é do Dado 4, seguido pelos dados 2, 1 e 3 respectivamente.

2.1.3.2 REMOTE TRANSMISSION REQUEST (RTR)

O bit RTR define o tipo de mensagem que será transmitida por um nó, ou seja, se é uma mensagem de dados (quando dominante) ou uma mensagem remota (quando recessivo) (CORRIGAN, 2008).

Quando a mensagem transmitida é remota, o campo de dados da mensagem não existe pois esta serve apenas como requisição de algum dado presente dentro do barramento CAN, tanto que normalmente uma mensagem remota é procedida por uma mensagem de dados.

Uma vez que o bit RTR é recessivo quando indica uma mensagem remota, então isso significa que a prioridade da mensagem de dados referente à requisição é sempre maior (CAN IN AUTOMATION, 2016), visto que se uma mensagem de dados for solicitada, a mensagem remota não atrapalhará o fluxo no barramento.

2.1.3.3 DATA LENGTH CODE (DLC)

Os quatro bits de DLC definem o tamanho do campo de dados da mensagem, que pode variar entre 0 e 8 bytes. Nesse caso, a escolha do DLC deve ser adequada à quantidade de informação que será transmitida e, de acordo com a norma ISO11898, deve seguir o padrão mostrado pela Tabela 4.

Tabela 4: Preenchimento do campo DLC de uma mensagem de dados. (continua)

Bytes	**DLC**			
	DLC3	DLC2	DLC1	DLC0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0

Tabela 4: Preenchimento do campo DLC de uma mensagem de dados. (conclusão)

Bytes	DLC			
	DLC3	DLC2	DLC1	DLC0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	X	X	X

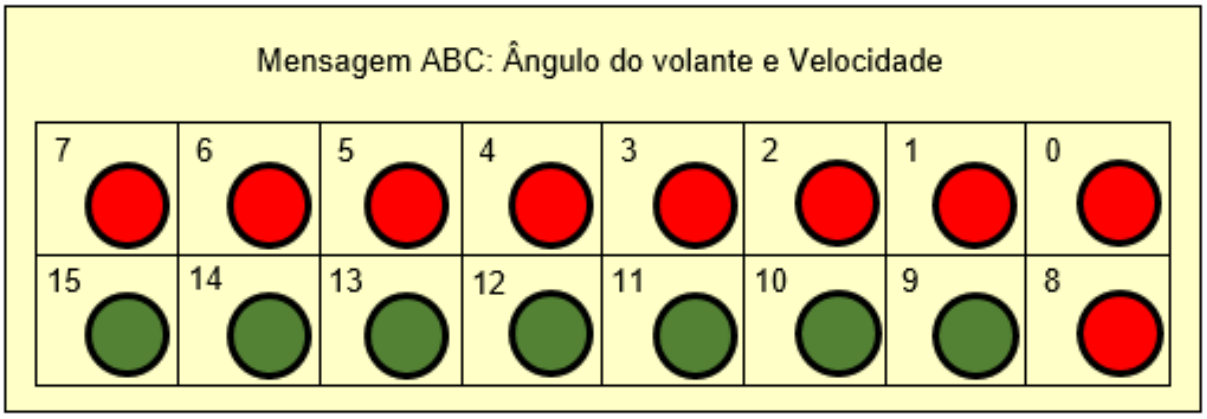
Fonte: Adaptado de ISO11898-1 (2003).

2.1.3.4 DADOS

O campo de dados de uma mensagem é responsável por armazenar todo o seu conteúdo útil e, dessa forma, possibilitar a comunicação entre dois nós de um mesmo barramento CAN. Sendo assim, cada mensagem possui um campo de dados delimitado pelos bits DLC que varia entre 0 bytes, no caso das mensagens remotas e 8 bytes no caso das mensagens de dados (CAN IN AUTOMATION, 2016).

Quanto ao campo de dados da mensagem é importante definir alguns parâmetros de organização, pois dentro de uma mensagem podem existir muitas informações relacionadas a vários componentes da rede, sendo que cada uma dessas pode ser definida como um sinal (VECTOR INFORMATIK, 2016). A Figura 8 ilustra o campo de dados de uma mensagem aleatória com DLC de 2 bytes (16 bits) com o sinal de velocidade de um veículo e do ângulo da barra de direção, representados respectivamente pelos círculos verdes e vermelhos.

Figura 8: Campo de dados de uma mensagem aleatória com DLC de 2 bytes.



Fonte: Autoria Própria.

Nota-se que o sinal de velocidade possui 7 bits, enquanto que o sinal de ângulo da barra de direção possui 9 bits. Nesse caso, a velocidade do veículo varia entre 0 e 127 unidades, enquanto que o ângulo do volante varia entre 0 e 511 unidades.

O tratamento dado aos sinais pode ser feito de diversas formas. No caso da velocidade, o valor real pode ser de 0 a 127 multiplicado por 2, por exemplo, se caso o sistema não exigir muita precisão para esse dado, o que resulta em uma velocidade máxima de 254 quilômetros por hora; já o sinal de ângulo pode exigir mais precisão, porém a ele deve ser aplicado um deslocamento de 255 unidades, fazendo com que o ângulo nulo real seja um número binário diferente de 0 e com precisão de 1°. A Tabela 5 demonstra algumas operações que podem ser aplicadas aos dados da mensagem ABC.

Tabela 5: Exemplos de operações que podem ser aplicadas a sinais.

Operação	**Dados (Decimais)**				**Cálculo**	**Unidade Inicial**	**Valor Mín.**	**Valor Máx.**	**Unidade Final**	**Preci-são (+-)**
	Velocidade		Ângulo do volante							
	Inicial	Final	Inicial	Final						
Offset	-	-	0	511	Valor - 255	°	-255	256	°	1°
Multiplicação	0	127	-	-	Valor * 2	km/h	0	254	km/h	2
Divisão	-	-	0	511	Valor / 10	°	0	51,1	°	0,1
Conversão	0	127	-	-	Valor / 3,6	km/h	0	35,28	m/s	0,28

Fonte: Autoria Própria.

Uma vez que são enormes as possibilidades de se codificar os dados de uma mensagem CAN dentro do barramento, então as montadoras acabam aplicando inúmeras operações a eles para evitar que qualquer pessoa com uma interface de leitura CAN tenha acesso aos dados que circulam na rede. Isso faz com que existam ferramentas de organização e manipulação de dados de redes CAN, tais como os programas CANape®, CANoe® e CANalyzer®, da empresa Vector Informatik® (VECTOR INFORMATIK, 2016), MATLAB® e Simulink® da empresa MathWorks® (MATHWORKS, 2016) e BUSmaster (BUSMASTER, 2016), um programa de código aberto e gratuito de acesso livre disponível em <https://rbei-etas.github.io/busmaster/>.

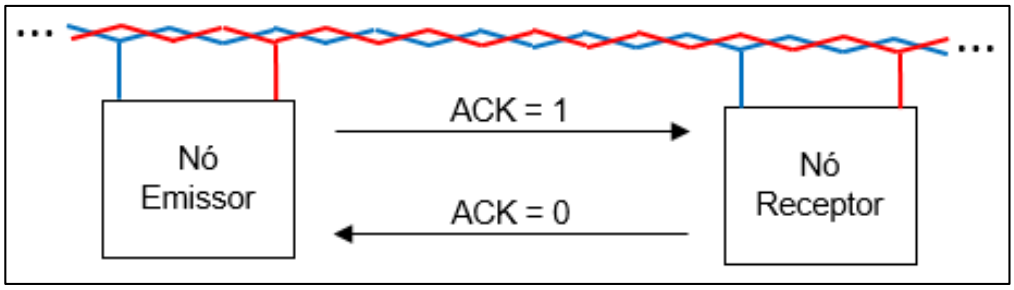
2.1.3.5 CYCLIC REDUNDANCY CHECK E ACKNOWLEDGE

O sistema de correção de erros que o protocolo CAN emprega nas mensagens do tipo 2.0A é um CRC de 15 bits que auxilia na segurança dos dados que trafegam na rede. Devido a técnica empregada e a extensão do código CRC, é possível corrigir até 6 bits errados enviados por um nó.

Apesar do CRC utilizar um espaço reservado dentro da mensagem CAN de 15 bits, há ainda um bit delimitador deste código que indica o fim dele para início da leitura do próximo campo, o que torna o espaço necessário para o código CRC um campo de 16 e não 15 bits.

Uma vez enviada a mensagem, então o receptor faz a leitura dos dois bits ACK, assim como acontece no campo onde o CRC está presente. Quando a mensagem foi enviada, mas ainda não foi lida, o bit ACK é recessivo, enquanto que se a mensagem já tiver sido recebida pelo nó de destino, esse retorna a mesma mensagem ao barramento com o ACK dominante. A Figura 9 exemplifica essa situação.

Figura 9: Exemplo de funcionamento do bit ACK na rede CAN.



Fonte: Autoria Própria

2.1.4 CARACTERÍSTICAS GERAIS

Todas as principais características citadas até então demonstram que o protocolo CAN foi desenvolvido com base na técnica CSMA/CD + AMP (*Carrier Sense Multiple Access with Colision Detection and Arbitration on Message Priority, em inglês*), ou seja, a rede CAN é uma rede confiável, por contar com os sistemas já

citados de detecção de erros, prevê prioridades e é capaz de perceber quando o barramento está ocupado (LUGLI; SANTOS, 2009).

Por possuir um campo de dados de no máximo 8 bytes, considerado pequeno se comparado ao campo de dados de um pacote Ethernet (até 1500 bytes), a rede CAN consegue atingir até 1 Mbit de velocidade a curtas distâncias, o que a torna ideal para ser utilizada em veículos de pequeno, médio e grande porte (SAE J1979, 2006).

O barramento se trata de um par de fios trançado, o que torna a rede bastante resistente a interferências através do processo de cancelamento eletromagnético (TANENBAUM, 2010), que consiste na anulação dos campos eletromagnéticos gerados pela corrente que passa pelos fios do barramento.

Uma vez que o barramento é constituído de apenas dois fios, a quantidade de cabeamento utilizado também é menor, o que torna a rede CAN uma rede de baixo custo além de simples, confiável e muito segura.

2.2 *SERIAL PERIPHERAL INTERFACE* - SPI

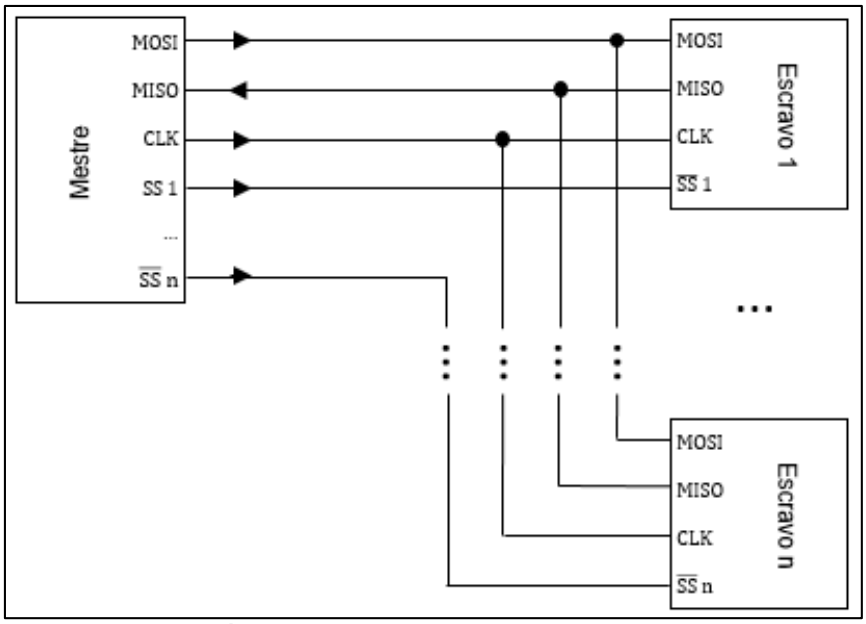
A integração de vários microcontroladores para apenas uma aplicação se tornou comum devido ao surgimento de variados chips destinados a trabalhos dedicados, tais como memórias EEPROM e conversores analógicos digitais. Dessa forma, uma vez que a necessidade de poder de processamento aumentou junto a demanda de novas aplicações, a Interface de Comunicação Serial entre Periféricos (em inglês, *Serial Peripheral Interface* ou SPI) passou a ser utilizada para fins de comunicação entre dispositivos e periféricos (ANAND et al, 2014) por mostrar-se eficiente e rápida, sincronizada por osciladores em altas frequências.

Um sistema de comunicação SPI é composto de dois dispositivos conhecidos por mestre, responsável pela transmissão e requisição de dados de outro dispositivo, e escravo, que recebe informações do mestre, realiza a manipulação destas e as transmite novamente ao mestre (STEVAN et al., 2011).

Para uma comunicação eficiente entre mestre e escravo, é necessário a existência de quatro canais conhecidos por MOSI (em inglês *Master Output Slave Input,* ou Saída do Mestre para Entrada do Escravo), MISO (em inglês *Master Input Slave Output* ou Entrada do Mestre para Saída do Escravo), SS (em inglês *Slave*

Select, ou Seletor do Escravo) e CLK (em inglês *Clock* ou Relógio de Sincronismo) conectados do mestre para o escravo conforme sugere a Figura 10.

Figura 10: Conexão entre o microcontrolador mestre e os escravos.



Fonte: Autoria Própria

O fluxo de informações segue conforme as setas mostram na Figura 10. A comunicação via SPI ocorre de maneira síncrona, logo depende de um relógio de sincronismo cujo sinal é emitido sempre pelo mestre através de CLK. Essa frequência pode variar conforme a frequência máxima suportada em cada um dos escravos, que depende do poder de processamento e de leitura de dados de cada um.

Como exemplo, pode-se citar um dos elementos utilizados no desenvolvimento desse trabalho: o microprocessador dedicado MCP2515, o qual foi escolhido para a interpretação e envio de mensagens CAN, capaz de funcionar com frequências de até 10MHz (MICROCHIP TECHNOLOGIES, 2005).

A título de comparação com o microcontrolador MCP2515, o micro cartão de memória SD (em inglês, *Secure Digital*), chega a operar em frequências de até 50MHz (SD ASSOCIATION, 2011) dependendo da capacidade de armazenamento e da velocidade de leitura e escrita do periférico.

Para que ocorra a correta transmissão de informações, é necessário que o escravo correspondente seja ativado através de um sinal lógico baixo nos pinos SS. O SPI permite que vários escravos sejam conectados a um mesmo mestre, desde que durante a transmissão o escravo endereçado seja ativado, o que ocorre normalmente

e através do envio de um sinal lógico de nível baixo através do pino de seleção (STEVAN et al., 2011).

Quando ativado, o escravo recebe informações enviadas pelo mestre através do pino MOSI byte a byte através de pulsos no pino CLK. A cada pulso, um bit é enviado pelo mestre através de MOSI e outro bit é recebido pelo mestre através de MISO, devido ao SPI ser um protocolo de comunicação *full-duplex*, indicando que informações são recebidas e enviadas ao mesmo tempo (FOROUZAN, 2006), semelhante a uma via de mão dupla.

Como já citado, a comunicação pelo protocolo SPI ocorre de forma síncrona entre todos os dispositivos e em *full-duplex*. Dessa forma, deve-se levar em consideração algumas características específicas do protocolo, tais como os modos de operação e a forma detalhada com a qual ocorre essa comunicação, analisada diretamente no barramento através de uma ferramenta conhecida como Analisador Lógico.

2.2.1 MODOS DE OPERAÇÃO

O SPI pode operar de quatro modos diferentes conhecidos respectivamente por Modo 0, Modo 1, Modo 2 e Modo 3. Todos os modos estão relacionados a forma como são lidas e enviadas as informações no barramento em relação ao pulso de *clock* enviado pelo mestre (CYPRESS PERFORM, 2010).

São conhecidas duas características importantes quanto ao *clock*: a polarização (CPOL) e a fase (CPHA), responsáveis por caracterizar cada um dos modos de operação conforme mostra a Tabela 6.

Tabela 6: Modos de operação do SPI

Modo	CPOL	CPHA
Modo 0	0	0
Modo 1	0	1
Modo 2	1	0
Modo 3	1	1

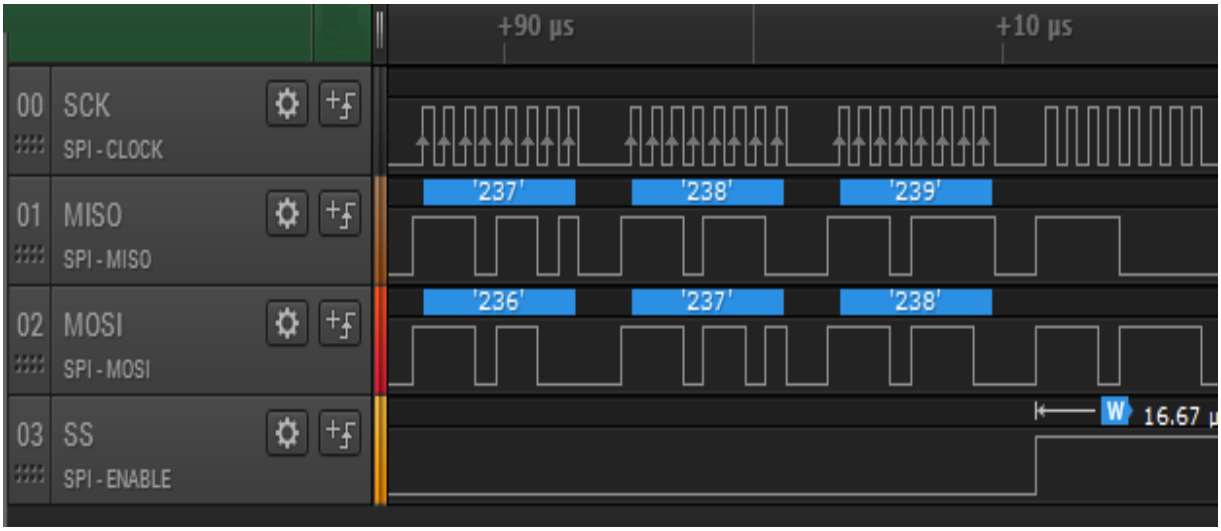
Fonte: Autoria Própria.

Quando CPOL = 0, então indica-se que quando o sinal de *clock* é normalmente baixo (*clock* em nível baixo quando não há transmissão) nenhum tipo de leitura está ativa, ao passo que quando CPOL = 1, então não há leitura quando o pulso de *clock* é normalmente alto (*clock* em nível alto quando não há transmissão). Da mesma forma, CPHA controla quando o dado é amostrado, se na borda de subida do pulso de *clock*, quando CPHA = 0, ou na borda de descida, quando CPHA = 1.

2.2.1.1 MODO 0

Este é o modo mais comum de trabalho do SPI, no qual o *clock* é normalmente baixo e o dado é amostrado na borda de subida. A Figura 11 ilustra a aquisição de um dado utilizando esta configuração.

Figura 11: Configuração do SPI em Modo 0.



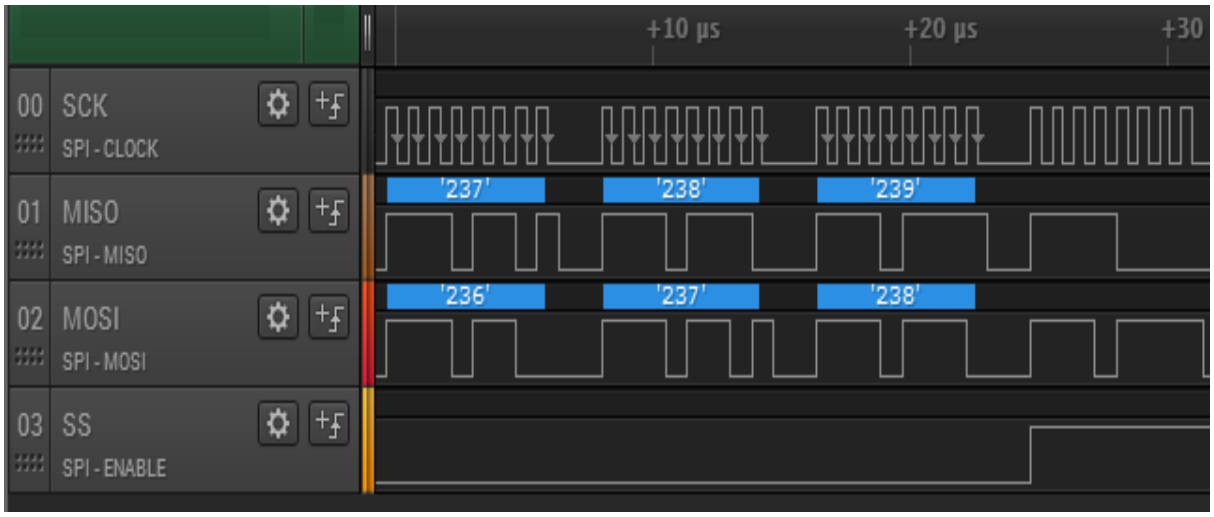
Fonte: Autoria Própria

Nota-se que o microcontrolador mestre está recebendo do escravo o byte 11101101, que em base decimal significa 237. Dessa forma, o mesmo mestre envia nos mesmos pulsos de *clock* um byte 11101100, que significa 236. Uma vez que o escravo está habilitado nos três primeiros bytes através de SS em nível lógico baixo, então 236, 237 e 238 são enviados pelo mestre e recebidos pelo escravo, enquanto que o último byte também é transmitido pelo mestre, porém o escravo não o recebe.

2.2.1.2 MODO 1

Neste modo, o SPI continua operando com o pulso de *clock* normalmente baixo, porém o dado passa a ser amostrado nas bordas de descida, conforme mostra a Figura 12. Nesse caso, os mesmos bytes são enviados e recebidos apenas para fins de comparação.

Figura 12: Configuração do SPI em Modo 1.

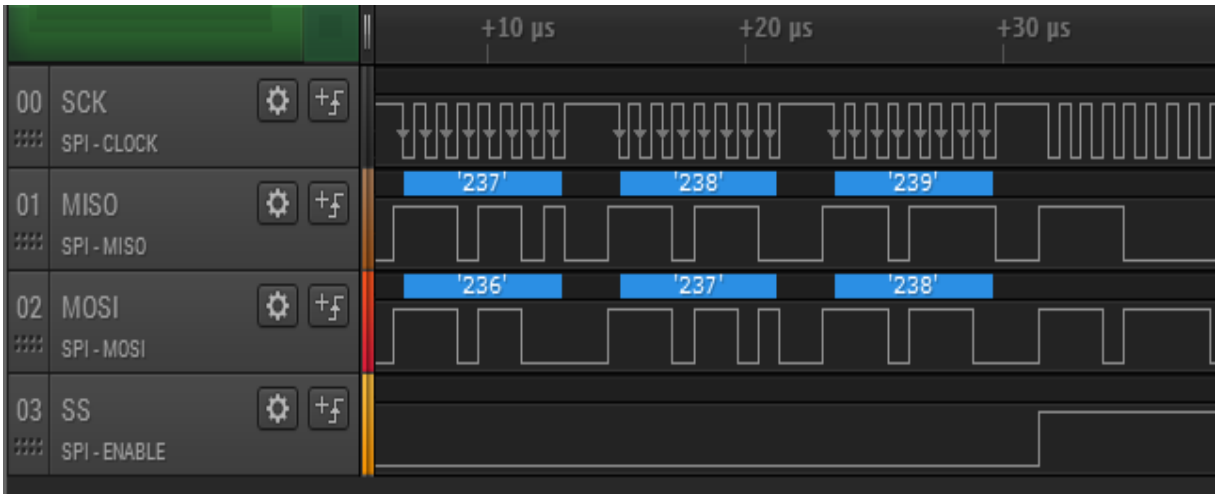


Fonte: Autoria Própria

2.2.1.3 MODO 2

A partir dessa configuração, o *clock* passa a ser normalmente alto e a amostragem é realizada na borda de descida. A Figura 13 ilustra como atua um dispositivo mestre comunicando-se com um escravo nesse modo, utilizando os mesmos bytes já detalhados nos modos anteriores.

Figura 13: Configuração do SPI em Modo 2.

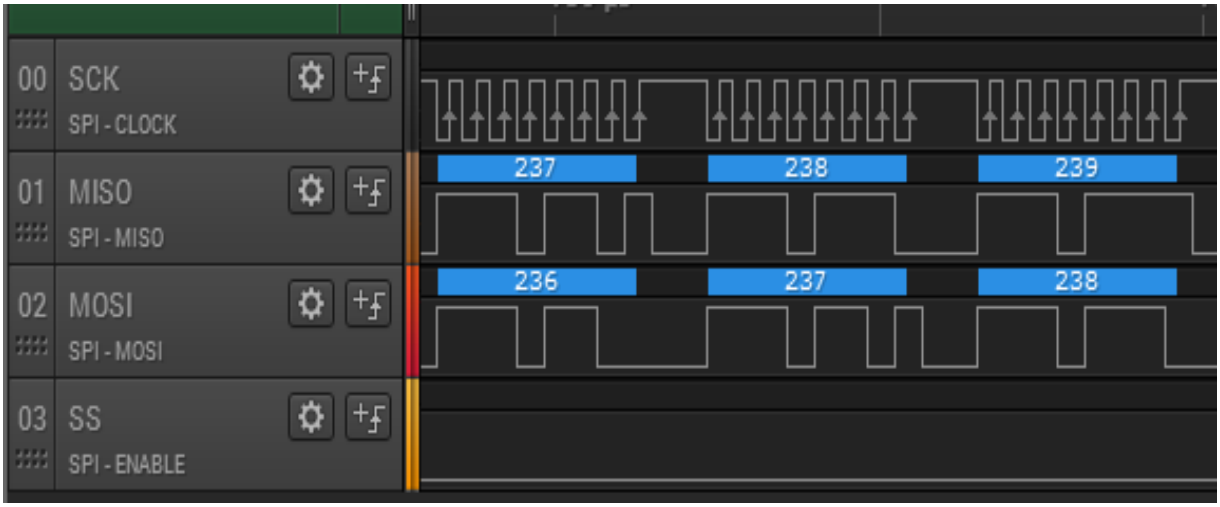


Fonte: Autoria Própria.

2.2.1.4 MODO 3

Assim como no Modo 2, o *clock* é normalmente alto, enquanto que a amostragem passa a ser realizada na borda de subida dos pulsos. Dessa forma, os mesmos bytes enviados e recebidos são lidos conforme mostra a Figura 14, porém da forma característica no Modo 3.

Figura 14: Configuração do SPI em Modo 3.



Fonte: Autoria Própria

Uma vez que todos os dispositivos usados para esse trabalho são compatíveis com o Modo 0, este foi utilizado no desenvolvimento para evitar qualquer tipo de

conflito e, dessa forma, estabelecer a comunicação de maneira adequada entre o mestre e todos os periféricos.

2.3 PLATAFORMA ARDUINO

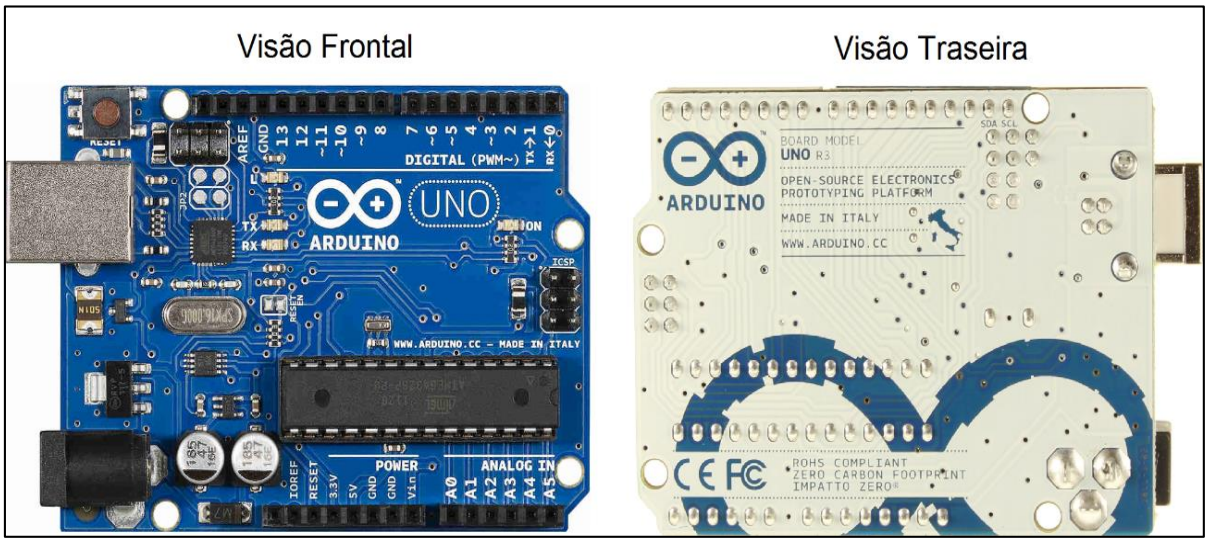
Para o desenvolvimento do dispositivo de monitoramento CAN proposto nesse trabalho foram utilizados dois elementos importantes: um Arduino UNO® e um Arduino MEGA. Uma vez que cada um desses tem papel fundamental para o desenvolvimento do dispositivo final, então é importante tratar de alguns detalhes para cada um deles.

2.3.1 ARDUINO UNO

A plataforma de prototipagem rápida Arduino UNO® conta com um microcontrolador ATMEGA328P fabricado pela empresa Atmel® (ATMEL CORPORATION, 2016) que opera como o cérebro da placa. É uma placa pequena com suporte a comunicação USB e que possui um ambiente de desenvolvimento (IDE, em inglês, *Integrated Development Environment*) aberto programado em linguagem Java®, disponibilizado pela própria fabricante (ARDUINO, 2016)

Na Figura 15 é possível ver a placa e alguns de seus detalhes. Nota-se que, apesar de pequena, é uma placa bem completa constituída não apenas de um microcontrolador, mas sim de todos os elementos necessários ao seu funcionamento já instalados.

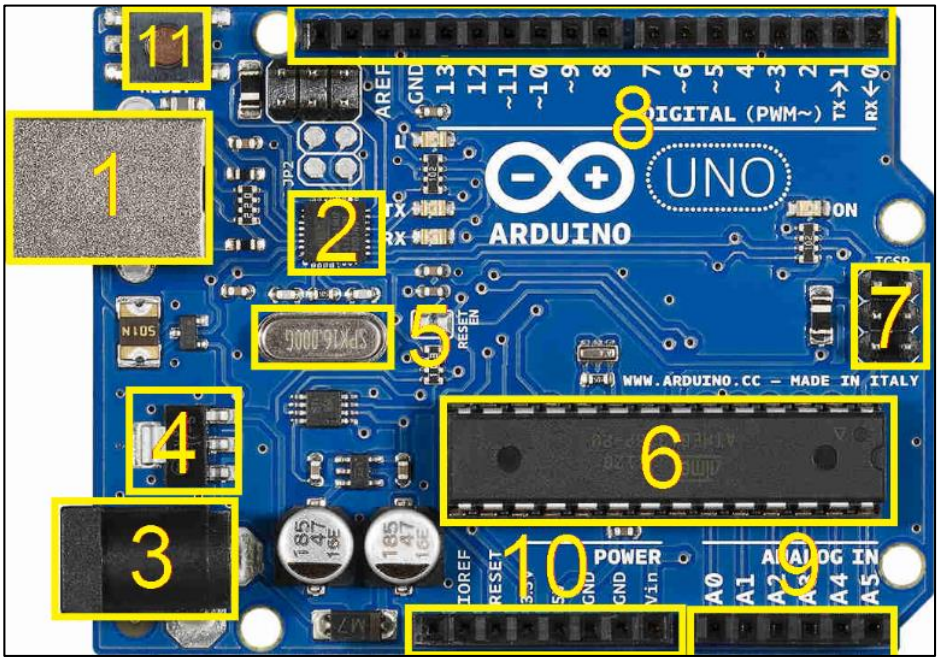
Figura 15: Visão frontal e traseira da placa de desenvolvimento Arduino UNO®.



Fonte: Adaptado de www.robotechshop.com.

É importante avaliar alguns elementos presentes na placa que são de fundamental importância ao funcionamento do circuito completo. Dessa forma, na Figura 16 é possível observar alguns dos elementos, tais como onde é realizada a conexão USB da placa com o computador, o oscilador, todas as entradas e saídas, o conector de alimentação externa e um botão de reinicialização da placa, todos detalhados pela Tabela 7.

Figura 16: Elementos principais da placa Arduino UNO.



Fonte: Adaptado de www.robotechshop.com

Tabela 7: Descrição dos elementos contidos na placa Arduino UNO.

Elemento	Descrição
1	Interface de conexão USB entre a placa e o computador
2	Microcontrolador responsável pela interface serial entre o microcontrolador ATMEGA328p e a USB.
3	Conector para fonte de alimentação externa independente
4	Regulador da tensão da fonte de alimentação externa
5	Oscilador de 16MHz compatível ao ATMEGA328p
6	Microcontrolador principal ATMEGA328p
7	Interface de comunicação SPI
8	Pinos de entrada e saída digitais
9	Pinos de entrada analógicos
10	Pinos de alimentação da placa
11	Botão de reinicialização da placa

Fonte: Autoria Própria.

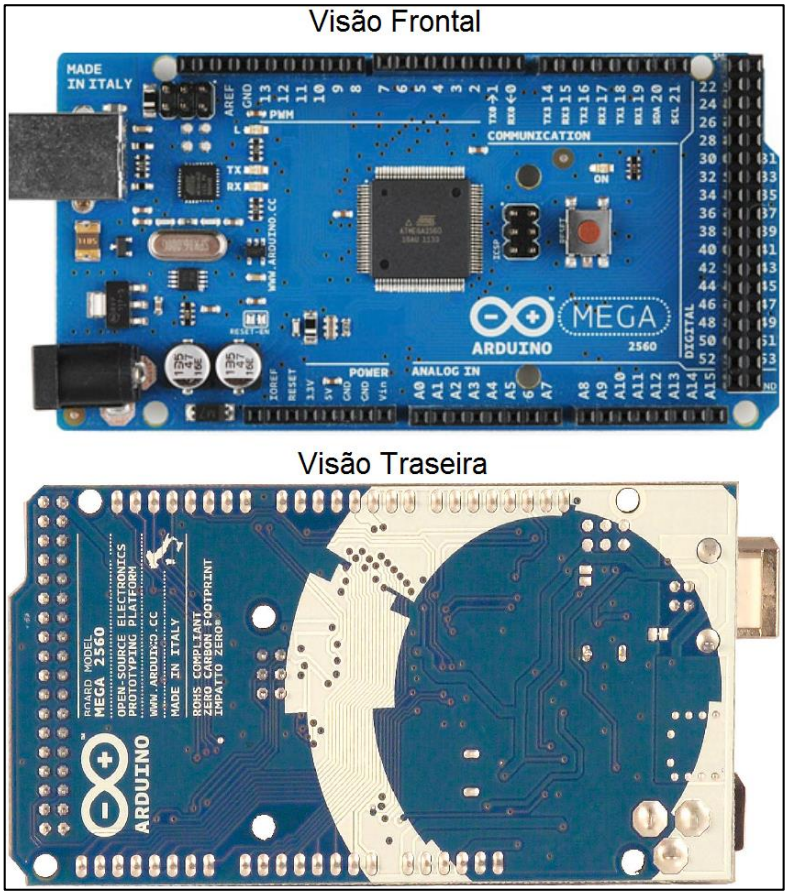
Nota-se que há uma diferença na quantidade de portas digitais e analógicas. Isso ocorre devido ao fato de as portas digitais, nas quais a tensão eficaz é nula ou 5V, excetuando-se pelas portas compatíveis com PWM (*Pulse Width Modulation*) cuja tensão varia nesse intervalo, serem compatíveis tanto com entrada quanto com saída de tensão, diferente das analógicas, em que são aceitos, se configurados, somente sinais de entrada de natureza analógica (ARDUINO, 2016).

2.3.2 ARDUINO MEGA®

Apesar de possuir estrutura muito parecida à do Arduino UNO® para manter a compatibilidade entre os módulos de extensão desenvolvidos para este, o Arduino MEGA é ainda mais poderoso e conta com várias entradas e saídas a mais que o seu antecessor.

Conta com um microprocessador Atmel® ATMEGA2560 mais potente que o ATMEGA328p (ATMEL CORPORATION, 2016) em vários quesitos destacados adiante e pode-se, através da Figura 17, ter ideia das visões frontal e traseira da placa de desenvolvimento.

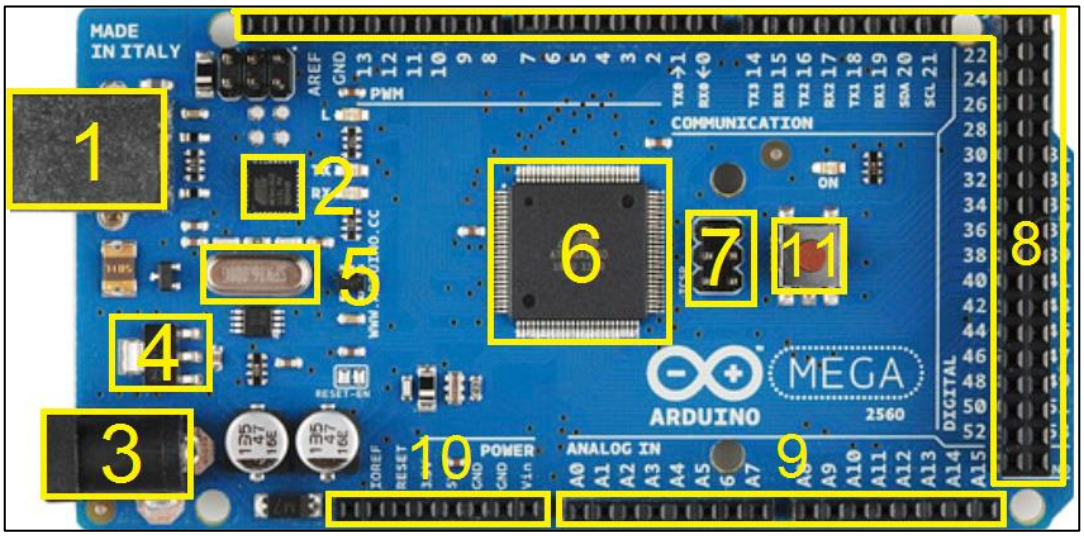
Figura 17: Visões Frontal e Traseira da placa de Desenvolvimento Arduino MEGA.



Fonte: Adaptado de www.robotech.com.

Bem como o Arduino UNO, o Arduino MEGA possui elementos importantes e que devem ser explorados. Através da Figura 18 e da Tabela 8 pode-se notar que a placa difere em alguns aspectos da citada anteriormente como a quantidade de entradas e saídas digitais, de entradas analógicas, a disposição do microcontrolador principal bem como o modelo deste e a posição do botão de reinicialização, porém para os demais pinos de conexão a localização se mantém, fator que garante a compatibilidade entre ambas as placas.

Figura 18: Elementos principais da placa Arduino MEGA.



Fonte: Adaptado de www.robotech.com.

Tabela 8: Descrição dos elementos contidos na placa Arduino MEGA.

Elemento	Descrição
1	Interface de conexão USB entre a placa e o computador
2	Microcontrolador responsável pela interface serial entre o microcontrolador ATMEGA2560 e a USB.
3	Conector para fonte de alimentação externa independente
4	Regulador da tensão da fonte de alimentação externa
5	Oscilador de 16MHz compatível ao ATMEGA2560
6	Microcontrolador principal ATMEGA2560
7	Interface de comunicação SPI
8	Pinos de entrada e saída digitais
9	Pinos de entrada analógicos
10	Pinos de alimentação da placa
11	Botão de reinicialização da placa

Fonte: Autoria Própria.

2.3.3 ESPECIFICAÇÕES DAS PLACAS ARDUINO UNO E ARDUINO MEGA

Devido as placas Arduino possuírem um microcontrolador como elemento principal, então é necessário que se faça uma comparação entre as elas, pois desta forma é possível justificar tanto a escolha das funções que cada uma vai desempenhar quanto a importância de cada uma das placas.

Na Tabela 9 é possível comparar ambas as placas em relação à quantidade de memória para programas (ou memória *Flash*) ou para variáveis globais (ou memória de acesso randômico, RAM, em inglês, *Random Access Memory*) bem como a velocidade da CPU (*Central Processing Unit*, em inglês ou Unidade de Processamento Central), quantidade de portas de entrada e saída digitais e analógicas e outras informações importantes.

Tabela 9: Especificações técnicas das placas Arduino Uno e Arduino Mega.

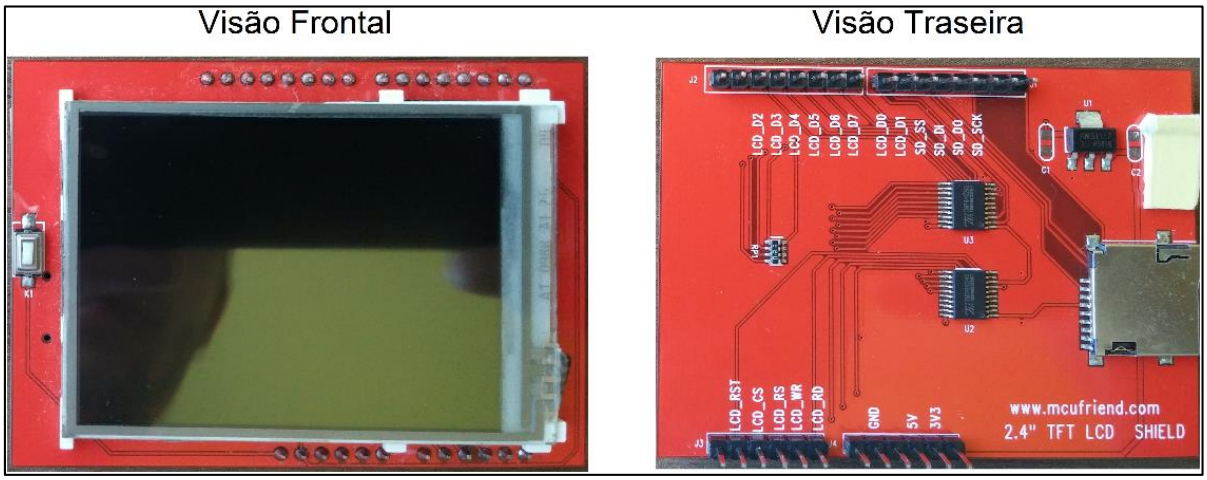
Característica	Arduino UNO	Arduino MEGA
Processador	ATmega328p	ATmega2560
Tensão de Operação	5V	5V
Velocidade da CPU	16MHz	16Mhz
Entradas Analógicas	6	16
Entradas e Saídas Digitais	14	54
Memória EEPROM	1 kB	4 kB
Memória SRAM	2 kB	8 kB
Memória FLASH	32 kB	256 kB
Portas de comunicação Serial	1	4

Fonte: Adaptado de Arduino (2016).

2.4 TELA TFT LCD

Para a programação de toda a interface entre o usuário e o dispositivo construído através deste trabalho, optou-se por utilizar uma tela TFT LCD de 2,4 polegadas modelo 2.4 TFT LCD Shield fabricada por MCUfriend® mostrada na Figura 19. Essa tela foi escolhida devido a ter compatibilidade completa com ambas as placas de desenvolvimento utilizadas e, além disso, possuir uma biblioteca pronta especialmente para Arduino que possibilitou toda a programação da interface homem máquina (IHM).

Figura 19: Visão frontal e traseira do módulo Tela TFT LCD utilizado.



Fonte: Autoria Própria

2.5 INTERFACE CAN: *TRANSCEIVER* TJA1050 E MICROCONTROLADOR MCP2515

Para a construção da interface entre o Arduino e a rede CAN, foram utilizados dois elementos fundamentais: um transceptor (em inglês, *transceiver*) CAN e um microcontrolador dedicado a traduzir o sinal disponibilizado serialmente pelo primeiro componente. Os dois componentes fazem parte de um módulo CAN destinado para uso em Arduino, porém optou-se por realizar toda a programação do microcontrolador CAN em baixo nível para que fosse possível exportar facilmente todo o código para outra plataforma no futuro. O módulo completo utilizado pode ser visto na Figura 20.

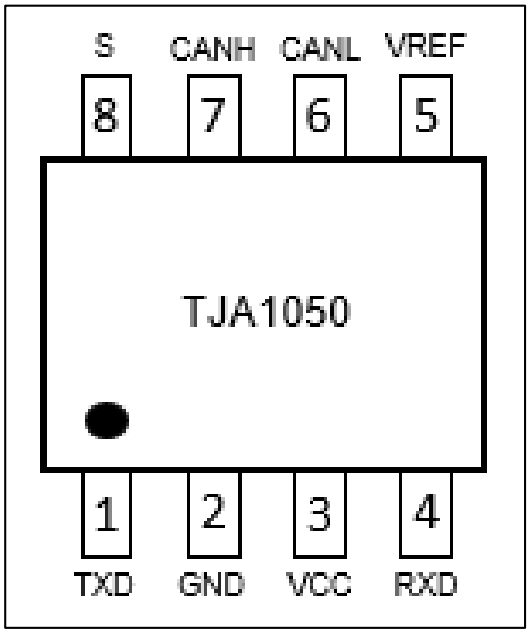
Figura 20: Módulo CAN Completo.

Fonte: Autoria Própria

2.5.1 TRANSCEPTOR TJA1050

O transceptor utilizado nesse trabalho foi o TJA1050, fabricado pela empresa NXP® Philips® *Semiconductors,* por ser completamente compatível com a norma ISO11898, suportar CAN de alta velocidade e realizar toda a interface entre a rede e o barramento físico (NXP PHILIPS SEMICONDUCTORS, 2003). A Figura 21 mostra o componente utilizado, junto à descrição simplificada de cada um dos pinos.

Figura 21: Transceiver TJA1050.



Fonte: Adaptado de NXP Philips Semiconductors (2003)

Uma vez que as entradas para o *transceiver* podem ser diretamente as linhas CANL e CANH do barramento CAN real, nos pinos 6 e 7 respectivamente, então basta conecta-lo ao barramento e alimentar o circuito integrado com uma tensão de 5V no pino VCC e com o terra comum no pino GND, representados por 2 e 3 respectivamente.

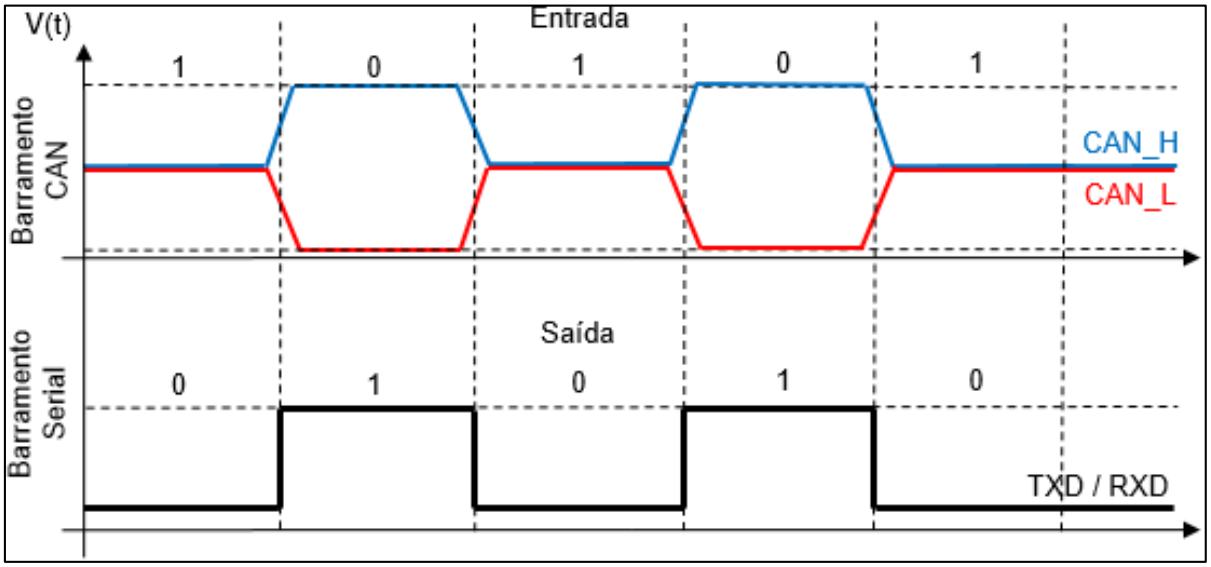
O pino S habilita uma função do *transceiver* chamada de “*Silent Mode*”, onde, se aplicada uma tensão de 5V, o modo é ativado, impedindo o componente de enviar sinais CAN para o barramento. Caso uma tensão 0V seja conectada ao pino, então o *transceiver* passa a operar de forma normal.

Depois realizar todas as conexões, VREF fornece a tensão média do barramento CAN, a qual segundo a folha de dados do componente deve estar entre 45 e 55% da tensão de alimentação deste, sendo 50% a tensão nominal para que se evite qualquer tipo de erro de leitura do barramento.

Detalhados todos os pinos de configuração e de leitura do *transceiver*, é importante avaliar os pinos TXD e RXD, respectivamente 1 e 4, responsáveis por receber ou fornecer o sinal serial que será utilizado para a decodificação das mensagens CAN pelo microcontrolador.

A cada bit dominante decodificado pelo *transceiver,* é enviado um bit 1 via serial pelo pino TXD, enquanto que a cada bit recessivo, é enviado um bit 0 como mostrado na Figura 22, o que faz com que todas as mensagens sejam transferidas bit a bit do transceptor ao microcontrolador MCP2515 que decodificará a sequência de acordo com o protocolo CAN.

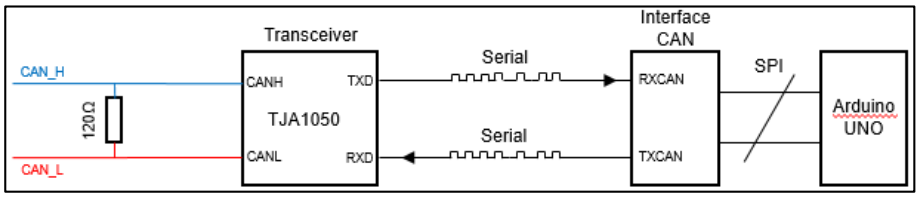
Figura 22: Diagrama de entrada e saída do transceptor TJA1050.



Fonte: Autoria Própria.

O pino RXD é o responsável por receber a mensagem CAN originada pelo microcontrolador, enquanto que após receber uma mensagem por completo, envia ao barramento CAN real através dos pinos CANH e CANL. A Figura 23 representa a relação entre o transceptor TJA1050, o microcontrolador CAN e o barramento real, que representa o *transceiver* realizando a interface entre o real e o dado digital.

Figura 23: Conexão entre o transceptor, o microcontrolador CAN e o Arduino.



Fonte: Autoria Própria

A velocidade de recepção ou transmissão serial depende da velocidade da rede CAN à qual os elementos estão conectados. Sendo assim, esta será explicada com mais detalhes no tópico a seguir, o qual apresentará toda a configuração do microcontrolador CAN MCP2515.

2.5.2 MICROCONTROLADOR CAN MCP2515

O microcontrolador CAN MCP2515 é fabricado pela Microchip® desde 2005 e foi escolhido devido a seus atributos serem suficientes à aplicação destacada por este trabalho, que inclui a necessidade da compatibilidade com redes de alta velocidade, comunicação entre microcontroladores via SPI e recepção e envio de mensagens CAN de formato padrão e estendido (MICROCHIP TECHNOLOGIES, 2005).

Uma vez que esse componente tem plena capacidade de se comunicar com outro microcontrolador via barramento SPI, então são disponibilizadas pela fabricante algumas instruções necessárias à realização de todas as escritas e leituras dos registradores que compõem o MCP2515, mostradas na Tabela 10.

Tabela 10: Principais instruções utilizadas no tratamento direto com registradores.

Instrução	Formato da Instrução	Descrição
Escrita	0000 0010	Grava dados em registradores compatíveis.
Leitura	0000 0011	Lê os dados de um registrador qualquer.
Modificar Bit	0000 0101	Modifica apenas alguns bits de um registrador
Reiniciar	1100 0000	Reinicia todos os registradores às configurações de fábrica.

Fonte: Adaptado de Microchip Technologies (2005)

Nem todas as instruções constantes na Tabela 10 são compatíveis com todos os registradores. A instrução “Modificar Bit” por exemplo, consegue acessar apenas

registradores específicos sem que o conteúdo desses seja destruído acidentalmente. Para evitar qualquer tipo de acidente com dados, na Tabela 11 estão todos os registradores acessíveis ao usuário via SPI, quais instruções podem ser utilizadas com cada um deles e seus endereços.

Tabela 11: Mapeamento dos registradores acessíveis ao usuário do microcontrolador MCP2515

***Nibble* menos significante do endereço**	***Nibble* mais significante do endereço**			
	0000	0001	0010	0011
0000	RXFOSIDH	RXF3SIDH	RXMOSIDH	TXBOCTRL
0001	RXFOSIDL	RXF3SIDL	RXMOSIDL	TXBOSIDH
0010	RXFOEIDB	RXF3EID8	RXMOEID8	TXBOSIDL
0011	RXFOEIDO	RXF3EIDO	RXMOEIDO	TXBOEID8
0100	RXF1SIDH	RXF4SIDH	RXM1SIDH	TXBOEIDO
0101	RXF1SIDL	RXF4SIDL	RXM1SIDL	TXBODLC
0110	RXF1EID8	RXF4EID8	RXM1EID8	TXBODO
0111	RXF1EIDO	RXF4EIDO	RXM1EIDO	TXBOD1
1000	RXF2SIDH	RXF5SIDH	CNF3	TXBOD2
1001	RXF2SIDL	RXF5SIDL	CNF2	TXBOD3
1010	RXF2EID8	RXF5EID8	CNF1	TXBOD4
1011	RXF2EIDO	RXF5EIDO	CANINTE	TXBOD5
1100	BFPCTRL	TEC	CANINTF	TXBOD6
1101	TXRTSCTRL	REC	EFLG	TXBOD7
1110	CANSTAT	CANSTAT	CANSTAT	CANSTAT
1111	CANCTRL	CANCTRL	CANCTRL	CANCTRL
***Nibble* menos significante do endereço**	***Nibble* mais significante do endereço**			
	0100	0101	0110	0111
0000	TXB1CTRL	TXB2CTRL	RXBOCTRL	RXB1CTRL
0001	TXB1SIDH	TXB2SIDH	RXBOSIDH	RXB1SIDH
0010	TXB1SIDL	TXB2SIDL	RXBOSIDL	RXB1SIDL
0011	TXB1EID8	TXB2EID8	RXBOEID8	RXB1EID8
0100	TXB1EIDO	TXB2EIDO	RXBOEIDO	RXB1EIDO
0101	TXB1DLC	TXB2DLC	RXBODLC	RXB1DLC
0110	TXB1DO	TXB2DO	RXBODO	RXB1DO
0111	TXB1D1	TXB2D1	RXBOD1	RXB1D1
1000	TXB1D2	TXB2D2	RXBOD2	RXB1D2
1001	TXB1D3	TXB2D3	RXBOD3	RXB1D3
1010	TXB1D4	TXB2D4	RXBOD4	RXB1D4
1011	TXB1D5	TXB2D5	RXBOD5	RXB1D5
1100	TXB1D6	TXB2D6	RXBOD6	RXB1D6
1101	TXB1D7	TXB2D7	RXBOD7	RXB1D7
1110	CANSTAT	CANSTAT	CANSTAT	CANSTAT
1111	CANCTRL	CANCTRL	CANCTRL	CANCTRL

Fonte: Adaptado de Microchip Technologies (2005)

Os registradores sombreados na Tabela 11 são compatíveis com a instrução “Modificar Bit”, ou seja, é possível alterar apenas um bit de todo o byte enquanto os demais permanecem em seu estado anterior.

Cada registrador possui um byte de endereçamento que será utilizado por algumas instruções para que sejam feitas as configurações necessárias. Vale destacar que o endereçamento de cada um dos registradores é diferente de seu conteúdo, ou seja, citando como exemplo o registrador CANCTRL localizado no endereço XXXX1111, não quer dizer que a configuração inicial de seus bits tem o mesmo valor numérico do endereçamento.

Os detalhes referentes a cada um dos registradores serão citados a medida que a configuração do microcontrolador for descrita, pois é adequado que se atribua a devida atenção a cada um dos bits de leitura ou escrita de cada registrador, visto que para algumas configurações um bit errado pode ocasionar um erro capaz de impedir a interação do microcontrolador com o *transceiver*.

2.5.2.1 INSTRUÇÕES COMPATÍVEIS COM O SPI

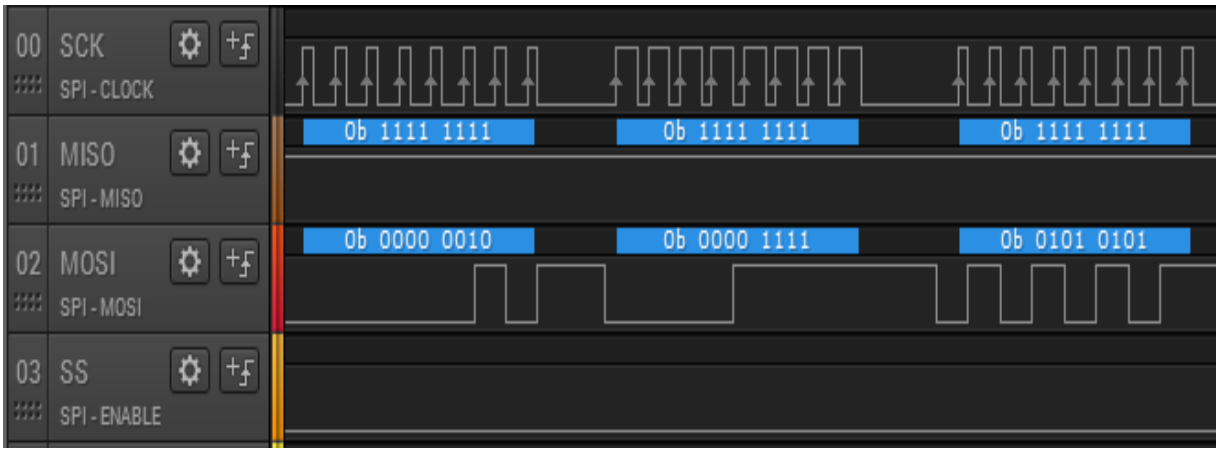
Existem diversas instruções utilizadas para várias tarefas, conforme consta na folha de dados do microcontrolador. Uma vez que nem todas foram utilizadas neste trabalho, este tópico visa detalhar as mesmas já citadas anteriormente tanto em estrutura quanto em retorno de dados.

Serão mostradas nos itens a seguir todas as informações necessárias à compreensão do funcionamento de cada uma das instruções e como estas atuam diretamente nos dados dos registradores utilizando como referência seus endereços específicos.

2.5.2.1.1 ESCRITA

A instrução de escrita pode ser dividida em três partes: *byte* da instrução, o endereço do registrador a ser sobrescrito e o dado que será transferido ao registrador. Entretanto a sequência dos comandos deve ser a da apresentada na Figura 24.

Figura 24: Sequência lógica para a escrita de um registrador.



Fonte: Autoria Própria

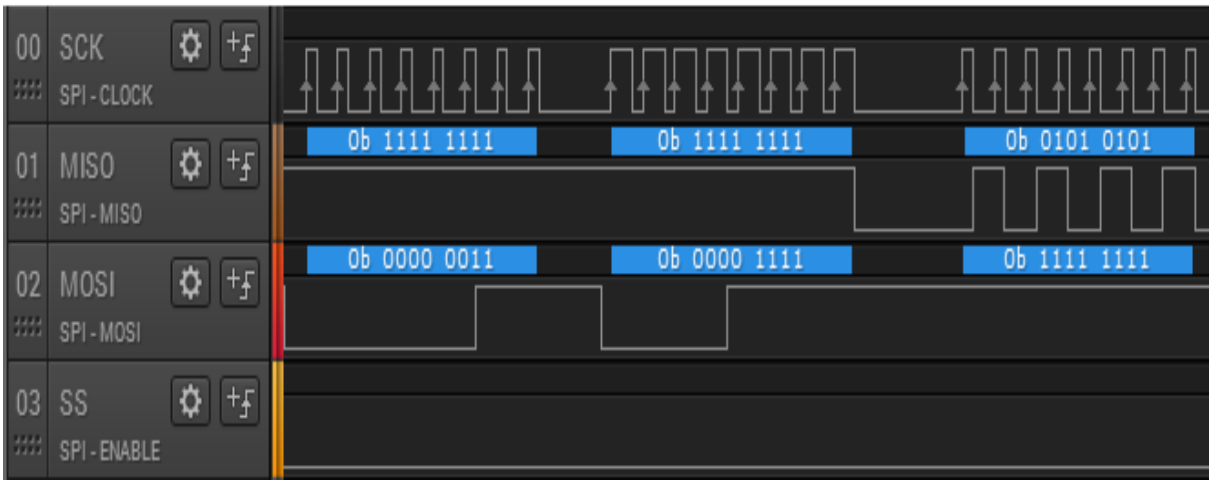
Nota-se que inicialmente o mestre deve mandar a sequência binária 0000 0010, que compõe a instrução Escrita. Dessa foma, a próxima sequência corresponde ao endereço do registrador, que em relação ao CANCTRL é XXXX 1111, seguido do valor para o qual o conteúdo do registrador deve ser alterado. Nesse caso foi escolhido alterar o valor de CANCTRL para 0101 0101 apenas para fins de demonstração, já que este é um registrador que suporta todas as instruções SPI destacadas na folha de dados do MCP2515.

A instrução Escrita não obriga o escravo a retornar um valor, logo o canal MISO permanece o tempo todo em nível lógico alto. É importante salientar que toda a instrução de Escrita foi trabalhada enquanto o escravo estava ativo com a tensão no pino SS em nível lógico baixo, logo para executar uma nova instrução deve-se desabilitar e habilitar o escravo novamente segundo instruções da folha de dados do microcontrolador MCP2515.

2.5.2.1.2 LEITURA

Propositalmente, foi executada uma gravação no registrador CANCTRL. Para que fosse possível comprovar que a gravação foi bem sucedida, a instrução de leitura foi realizada com o mesmo registrador para fins de validação. A Figura 25 mostra a estrutura do comando de leitura, seguido do retorno do escravo.

Figura 25: Sequência lógica para a leitura de um registrador.



Fonte: Autoria Própria

Assim como ocorreu na instrução Escrita, o primeiro passo é o envio da sequência binária 0000 0011 que corresponde a instrução Leitura. Sendo assim, é enviado então o endereço do registrador que deseja-se obter a leitura, para então o escravo mandar através do canal MISO o valor do registrador.

Nota-se que o valor retornado pela instrução de leitura em MISO é o mesmo enviado na instrução de Escrita, o que comprova que tanto a leitura quanto a escrita foram executados corretamente.

2.5.2.1.3 MODIFICAR BIT

Essa instrução é de extrema importância em alguns casos, pois nem sempre é necessário alterar todos os bits de um registrador. Exemplos mais concisos dessa instrução serão destacados à medida que ela for utilizada no decorrer do desenvolvimento, enquanto que nesse caso foi utilizado o mesmo registrador já citados nas duas instruções anteriores.

Inicialmente deve-se enviar via SPI a sequência binária atrelada ao comando Modificar Bit, ou seja, 0000 0101. Diferente das instruções anteriores, nesse caso é preciso de quatro ao invés de três etapas, pois deve-se enviar uma máscara a qual será utilizada pelo comando para definir o que será modificado ou não no conteúdo do registrador.

Uma vez enviada a sequência lógica do comando, é necessário transmitir ao microcontrolador o endereço do registrador a ser alterado e a máscara binária que dará as instruções ao comando sobre quais bits modificar. Foi utilizada a máscara 0000 0011, cuja permissão de alteração de bit é concedida apenas onde o valor binário da máscara é 1.

Sabe-se então que o comando está permitido a alterar apenas os dois últimos bits da informação que já estava contida no microcontrolador. Devido ao formato da máscara, qualquer alteração solicitada nos primeiros seis bits será ignorada, pois é incompatível com a máscara predefinida.

Como exemplo, a Figura 26 mostra a sequência lógica utilizada no comando Modificar Bit. Para exemplo, foi enviada inicialmente a instrução Modificar Bit, seguida pelo endereço do registrador a ser modificado, a máscara binária 0000 0011 e um exemplo fictício de alteração, definido como 1111 1110.

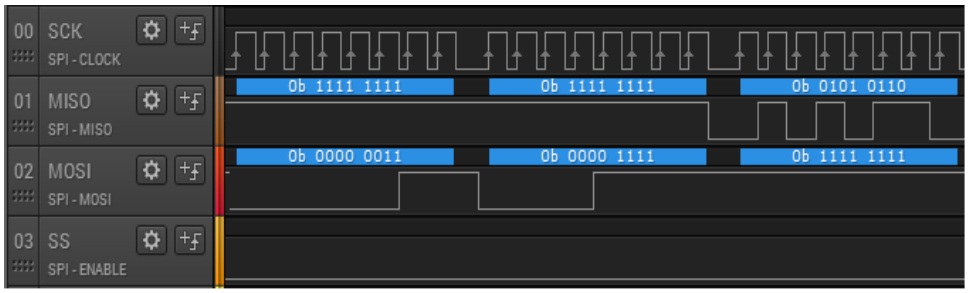
Figura 26: Sequência lógica para a mudança apenas de bits específicos de um registrador compatível.

Fonte: Autoria Própria

Uma vez que a máscara permite a alteração apenas dos dois últimos bits do registrador CANCTRL, então o valor gravado anteriormente, 0101 0101, deve ser alterado para 0101 0110. Para comprovar essa alteração, a Figura 27 mostra a leitura

do registrador, onde de fato foram modificados apenas os últimos dois bits conforme foi solicitado.

Figura 27: Leitura do novo valor do registrador após a instrução Modificar Bit.



Fonte: Autoria Própria.

2.5.2.1.4 REINICIAR

A função Reiniciar é a mais simples de todas, apesar de ser necessário que o microcontrolador esteja no modo de configuração. O registrador que controla o modo de operação do MCP2515 é o próprio CANCTRL, cuja estrutura é apresentada pela Figura 28.

Figura 28: Estrutura do registrador CANCTRL.

Registrador: CANCTRL							
L/E	L/E	L/E	L/E	L/E	L/E	L/E	L/E
REQOP2	REQOP1	REQOP0	ABAT	OSM	CLKEN	CLKPRE1	CLKPRE0
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

Fonte: Adaptado de Microchip Technologies (2005).

Pode-se reiniciar o microcontrolador via *software*, através do comando Reiniciar, ou removendo a alimentação do componente. Para que o microcontrolador seja reiniciado via *software*, é necessário, através do comando Modificar bit, escolher uma máscara capaz de alterar apenas os bits 7, 6 e 5 para 1, 0, 0 respectivamente, pois essa sequência é a que disponibiliza o MCP2515 para o modo de configuração, de acordo com a folha de dados do microcontrolador.

A máscara definida para realizar essa alteração foi 1110 0000. Sendo assim, deve-se mandar a instrução Modificar Bit, seguido pelo endereço do registrador CANCTRL, a máscara e então a sequência 1000 0000. Dessa forma, o

microcontrolador já está apto a ser reiniciado às suas configurações originais e agora a instrução Reiniciar, 1100 0000, já pode ser enviada.

Apesar de detalhada nesse tópico, a instrução Reiniciar não foi utilizada no decorrer do trabalho, visto que cada uma das vezes que o microcontrolador é desligado, os registradores já retornam a seu estado original.

Para evitar qualquer tipo de problema e configurações incorretas, foi estabelecida uma rotina de inicialização do microcontrolador que será apresentada no decorrer do desenvolvimento do trabalho, na qual é realizada toda a configuração inicial que torna o microcontrolador apto a operar conforme for solicitado.

2.5.2.2 ENVIO DE MENSAGENS CAN PARA O BARRAMENTO

O microcontrolador Microchip® MCP2515 possui três *buffers* de envio de mensagens. Sendo assim, é possível através desse mesmo sistema simular um barramento CAN automotivo, por exemplo, que normalmente é constituído de inúmeras informações sobre os sensores de monitoramento do veículo.

Há vários registradores que devem ser escritos para que uma mensagem seja enviada por completo. Dessa forma, é através do preenchimento dos registradores via SPI que são especificados o identificador da mensagem (ID), o tipo, o tamanho do campo de dados e os dados úteis a outros dispositivos da rede.

Uma vez que existem três *buffers* de envio de mensagens, segundo a folha de dados do MCP2515, a cada um deles é atribuída uma nomenclatura simples: *Buffer 0, Buffer 1 e Buffer 2*. Todos eles são configurados através de registradores individuais, portanto a configuração do *Buffer 0,* do *Buffer 1* e do *Buffer 2* pode ser equivalente que nenhum tipo de erro referente a configuração ocorrerá.

O primeiro passo para a configuração dos *buffers* de envio de mensagem é realizada pelo registrador TXBnCTRL, em que *n* significa o número atrelado ao *buffer* de envio em questão. A estrutura do registrador pode ser vista na Figura 29.

Figura 29: Estrutura dos registradores TXBnCTRL.

Registrador: TXBnCTRL							
L	L	L	L	L/E	L	L/E	L/E
-	ABTF	MLOAD	TXERR	TXREQ	-	TXP1	TXP0
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

Fonte: Adaptado de Microchip Technologies (2005).

Esse registrador suporta a instrução Modificar Bit, logo, esta deve ser utilizada nesse caso pois nem todos os bits suportam leitura e escrita segundo a folha de dados do microcontrolador. Dessa forma, os bits 6, 5 e 4 são sinalizadores (ou, em inglês, *flags*) que indicam respectivamente se a transmissão de uma mensagem foi interrompida, se a mensagem perdeu a arbitração no barramento ou se ocorreu um erro de transmissão.

Como inicialmente não será requerido o envio da mensagem, o bit 3 deve ser configurado como 0. A partir do momento que o usuário define esse bit como 1, a mensagem é enviada ao barramento e assim que a transmissão é finalizada, ele passa a ser 0 novamente. Portanto, para evitar que a mensagem seja enviada incompleta, deve-se no primeiro passo definir esse bit com 0.

Há ainda dois bits que devem ser configurados, relacionados diretamente com a prioridade do *buffer* que está sendo utilizado para o envio de mensagens. Como são 3 *buffers* de envio e essa operação não é instantânea, o buffer com maior prioridade deve assumir os bits 1 e 0 com os valores 1 e 1, enquanto que o de menor prioridade deve assumir os mesmos bits como 0 e 0.

Já configurado o principal registrador para iniciar o processo de envio de mensagem, antes de realizar o envio é necessário adicionar as informações que se deseja ao *buffer* em questão. Não há uma ordem obrigatória para a realização desse processo, logo será utilizada a ordem padrão dos bits de uma mensagem CAN, começando pela definição do identificador e terminando no conteúdo do campo de dados.

É possível configurar apenas alguns elementos da mensagem, tais como o identificador (ID), o tipo de mensagem (RTR), o tamanho do campo de dados (DLC) e os 8 bytes de dados úteis (D0 a D7). Outros bits como SOF, r0, CRC, ACK, EOF e IFS são gerados automaticamente pelo microcontrolador baseados na norma ISO 11898 (2003), pois são padrões do protocolo CAN.

Há ainda a opção de enviar mensagens com ID estendido, ou seja, não de 11, mas de 29 bits. Como não serão utilizados identificadores deste tipo nesse trabalho, então serão dadas as diretivas para a configuração apenas de identificadores de 11 bits, as quais remetem diretamente a dois registradores principais: TXBnSIDH e TXBnSIDL, cujas estruturas são apresentadas na Figura 30.

Figura 30: Estrutura dos registradores TXBnSIDH e TXBnSIDL.

Registrador: TXBnSIDH							
L/E	L/E	L/E	L/E	L/E	L/E	L/E	L/E
SID10	SID9	SID8	SID7	SID6	SID5	SID4	SID3
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

Registrador: TXBnSIDL							
L/E	L/E	L/E	L/E	L/E	L/E	L/E	L/E
SID2	SID1	SID0	-	EXIDE	-	EID17	EID16
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

Fonte: Adaptado de Microchip Technologies (2005).

No registrador TXBnSIDH se encontram os 8 bits mais significativos do identificador. Dessa forma, os outros 3 bits restantes devem ser configurados no registrador TXBnSIDL, escrevendo apenas nos bits 7, 6 e 5 deste registrador. O bit 3 define o tipo de identificador que será enviado, logo deve ser definido como 0 para mensagens de identificador padrão.

Há ainda dois bits irrelevantes no registrador TXBnSIDL para as configurações definidas. Os bits 1 e 0 desse registrador compõem a continuação do identificador para casos onde são utilizados identificadores estendidos, que se tornam irrelevantes quando o bit 3 é nulo.

Definido o identificador da mensagem a ser transmitida, é importante detalhar qual o tipo da mensagem que está prestes a ser enviada, que pode ser tanto de dados quanto apenas uma requisição, ou seja, uma mensagem remota. Essa configuração é definida pelo registrador TXBnDLC, apresentado detalhadamente pela Figura 31.

Figura 31: Estrutura do registrador TXBnDLC.

Registrador: TXBnDLC							
L/E	L/E	L/E	L/E	L/E	L/E	L/E	L/E
-	RTR	-	-	DLC3	DLC2	DLC1	DLC0
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

Fonte: Adaptado de Microchip Technologies (2005).

Além da definição do tipo de mensagem, feita através do bit 6, aqui também é decidido o tamanho do campo de dados. Se a mensagem é remota, ou seja, o bit 6 é 0, então não importa a configuração do DLC, pois mensagens remotas não possuem campo de dados. Já se a mensagem é de dados, então os bits 3, 2, 1 e 0 devem ser configurados conforme o tamanho da mensagem, vai desde 0 (0000) até 8 (1000) bytes.

Resta agora apenas a configuração do conteúdo da mensagem a ser enviada. Para isso, existem 8 registradores, um para cada byte, que são apresentados pela Figura 32, onde n significa o *buffer* preparado para a transmissão e m o byte que deve ser escrito.

Figura 32: Estrutura dos registradores de dados TXBnDm.

Registrador: TXBnDm							
L/E	L/E	L/E	L/E	L/E	L/E	L/E	L/E
BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

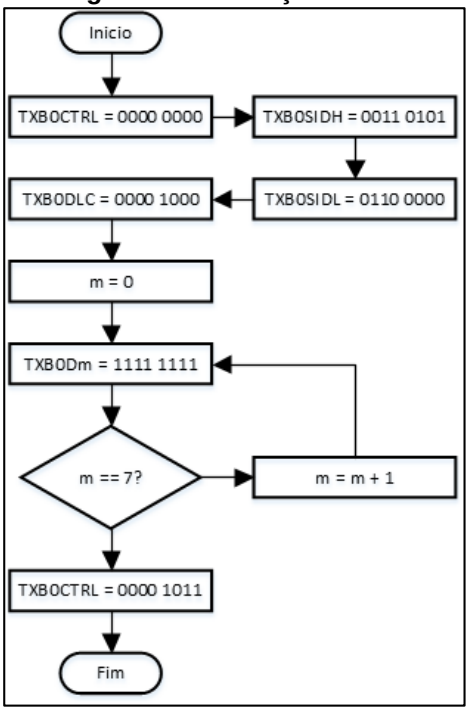
Fonte: Adaptado de Microchip Technologies (2005).

Vale lembrar que serão enviados apenas os bytes definidos previamente pelo DLC, ou seja, se foi utilizado um DLC de 5 bytes, por exemplo, serão enviados apenas os bytes 0, 1, 2, 3 e 4, enquanto que os bytes 5, 6 e 7 serão descartados automaticamente pelo microcontrolador.

Já definido todo o conteúdo da mensagem e suas configurações, pode-se então dar início à transmissão definindo o bit 3 do registrador TXBnCTRL para 1. Isso fará com que o microcontrolador envie todos bits configurados nos passos citados anteriormente além dos gerados pelo próprio protocolo CAN. O bit 3 também é o próprio sinalizador de mensagem enviada, uma vez que quando o processo de transmissão é concluído este passa a valer 0 novamente, independente da interação do usuário.

O processo de envio de mensagens é resumido pelo fluxograma apresentado pela Figura 33, que demonstra todo o ciclo com um exemplo de configuração do *buffer* 0 com prioridade máxima, ID 0x1AB e DLC de 8 bytes, que pode ser aplicado também aos outros *buffers* seguindo os mesmos cuidados.

Figura 33: Exemplo de transmissão de mensagem sem detecção de erros.



Fonte: Autoria Própria.

2.5.2.3 RECEPÇÃO DE MENSAGENS CAN DO BARRAMENTO

Assim como há elementos para armazenamento de mensagens para envio, há também os registradores e os *buffers* para recebimento de mensagens, que funcionam basicamente da mesma forma porém em sentido oposto.

Para o controle de mensagens recebidas há 3 *buffers* implementados no microcontrolador, sendo que dois destes são manipuláveis pelo usuário e um é utilizado pelo próprio componente para o armazenamento temporário de mensagens provenientes do barramento CAN segundo a folha de dados do MCP2515. Esse último é importante visto que existem possibilidades de filtrar quais mensagens deseja-se receber nos *buffers* manipuláveis, logo é possível evitar sobrecarga ou perda de mensagens.

Aos *buffers* manipuláveis dá-se o nome de RXB0 e RXB1. Cada um deles possui características importantes, principalmente relacionadas aos filtros e máscaras aplicadas na recepção das mensagens vindas do *buffer* temporário, logo RXB0 é o *buffer* preferencial para recebimento de mensagens e conta com apenas uma

máscara e dois filtros de aceitação de mensagens ao passo que RXB1 tem a menor prioridade e conta com uma máscara e quatro filtros de aceitação de mensagem.

Trabalhando inicialmente sem a utilização dos filtros, é importante começar pela seleção de qual *buffer* será utilizado inicialmente para a recepção de mensagens, visto que RXB0 e RXB1 possuem algumas configurações diferentes. Dessa forma, há dois registradores que devem ser utilizados na configuração do recebimento de mensagens: RXB0CTRL para RXB0 e RXB1CTRL para RXB1, apresentados estruturalmente na Figura 34.

Figura 34: Registradores de configuração para o recebimento de mensagens.

Registrador: RXB0CTRL							
L/E	L/E	L/E	L/E	L	L/E	L	L
-	RXM1	RXM0	-	RXRTR	BUKT	BUKT1	FILHIT0
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

Registrador: RXB1CTRL							
L/E	L/E	L/E	L/E	L	L	L	L
-	RXM1	RXM0	-	RXRTR	FILHIT2	FILHIT1	FILHIT0
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

Fonte: Adaptado de Microchip Technologies (2005).

O que difere na configuração dos dois registradores é basicamente os bits sinalizadores dos filtros, que para RXB0 é apenas o bit 0 e para RXB1 os bits 0, 1 e 2, e a opção de escrever uma mensagem recebida pelo *buffer* temporário no RXB1 caso RXB0 já esteja ocupado com outro conteúdo, controlado pelo bit 2 de RXB0.

Visto que não serão utilizados os filtros e nem a transferência de mensagem para RXB1, então basta escolher um dos *buffers* e programa-lo com os bits 5 e 6 como 0 e 0 utilizando a instrução Modificar Bit via SPI. Esse procedimento configura os *buffers* da maneira mais simples possível para receber todas as mensagens contidas dentro do barramento.

Existe ainda um registrador que controla quando uma mensagem é recebida, através de um sinalizador. Os bits do registrador CANINTF, apresentado na Figura 35, apontam interrupções que ocorrem no MCP2515 a cada mensagem recebida, enviada ou erros ocorridos, visto que o dispositivo mestre deve ter tempo suficiente para ler o conteúdo da mensagem e então depois reabilitar o microcontrolador a receber uma nova mensagem desativando a interrupção gerada através desse mesmo registrador.

Figura 35: Estrutura do registrador CANINTF.

Registrador: CANINTF							
L/E	L/E	L/E	L/E	L/E	L/E	L/E	L/E
MERRF	WAKIF	ERRIF	TX2IF	TX1IF	TX0IF	RX1IF	RX0IF
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

Fonte: Adaptado de Microchip Technologies (2005).

De todos os bits apresentados no registrador, para a recepção e o envio de mensagens os mais importantes são os bits 4, 3, 2, 1 e 0, pois eles apresentam o estado dos três *buffers* 0, 1 e 2 de envio (bits 4, 3 e 2, respectivamente) e dos dois *buffers* 0 e 1 manipuláveis de recebimento (bits 1 e 0, respectivamente).

A inicialização de qualquer um dos *buffers* de recebimento ocorre a partir do registrador CANINTF. Se antes foi configurado RXB0 para o recebimento de mensagens, então deve-se escrever 0 no bit 0 desse registrador e aguardar que ele valha 1 novamente, pois isso indica que todo o *buffer* de recebimento já foi preenchido com uma nova mensagem válida.

Feito isso, pode-se ler os registradores que armazenam o conteúdo da mensagem recebida, tal como seu identificador, padrão ou estendido, o tipo de mensagem, o DLC e todos os campos de dados.

Para ler o identificador da mensagem recebida deve-se atentar aos registradores RXBnSIDH e RXBnSIDL, nos quais *n* indica o número do *buffer,* 0 ou 1, utilizado para o recebimento da mensagem. Nesse caso, o processo de leitura ocorre de maneira simplificada através do barramento SPI, onde basta solicitar a leitura dos registradores que possuem estrutura semelhante à Figura 36 e então interpretar os dados.

Figura 36: Estrutura dos registradores de armazenamento dos identificadores de mensagens recebidas.

Registrador: RXBnSIDH							
L	L	L	L	L	L	L	L
SID10	SID9	SID8	SID7	SID6	SID5	SID4	SID3
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

Registrador: RXBnSIDL							
L	L	L	L	L	L	L	L
SID2	SID1	SID0	SRR	IDE	-	EID17	EID16
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

Fonte: Adaptado de Microchip Technologies (2005)

Através destes mesmos registradores é possível identificar se a mensagem é uma requisição remota, caso o bit 4 do registrador RXBnSIDL seja 1, ou se o identificador recebido for do tipo estendido, no caso de o bit 3 ser 1. Uma vez que o barramento ao qual o dispositivo será conectado não dispõe mensagens com identificadores estendidos, então os bits 1 e 0 do mesmo registrador são irrelevantes nessa configuração.

A interpretação e a disposição dos bits referentes ao registrador rege a mesma forma de avaliação mostrada anteriormente para o envio de mensagens, ou seja, o bit mais significativo do identificador é o bit 7 de RXBnSIDH e o menos significativo é o bit 5 de RXBnSIDL.

Já avaliado o identificador, é importante que se faça a leitura do DLC para determinar o tamanho do campo de dados da mensagem e evitar processamento desnecessário lendo bytes irrelevantes, ou seja, que não foram escritos nos registradores referentes ao campo de dados devido ao tamanho do DLC, processo que pode ser realizado através da leitura do registrador RXBnDLC apresentado pela Figura 37.

Figura 37: Registrador de armazenamento do DLC da mensagem recebida.

Registrador: RXBnDLC							
L	L	L	L	L	L	L	L
-	RTR	RB1	RB0	DLC3	DLC2	DLC1	DLC0
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

Fonte: Adaptado de Microchip Technologies (2005)

A partir do mesmo registrador é possível a leitura dos bits reservados r0 e r1 (bits 4 e 5, respectivamente), padrões do protocolo CAN, e se é do tipo remota ou de dados através do bit 6 caso a mensagem recebida seja de identificador estendido.

A leitura dos dados é feita através dos registradores RXBnDm, nos quais *n* e *m* são os números referentes ao *buffer* de recebimento escolhido e ao campo de dados que será lido, respectivamente, conforme o tamanho do DLC identificado no passo anterior. Esse registrador tem estrutura semelhante à mostrada na Figura 38, padrão tanto para RXB0 quanto para RXB1.

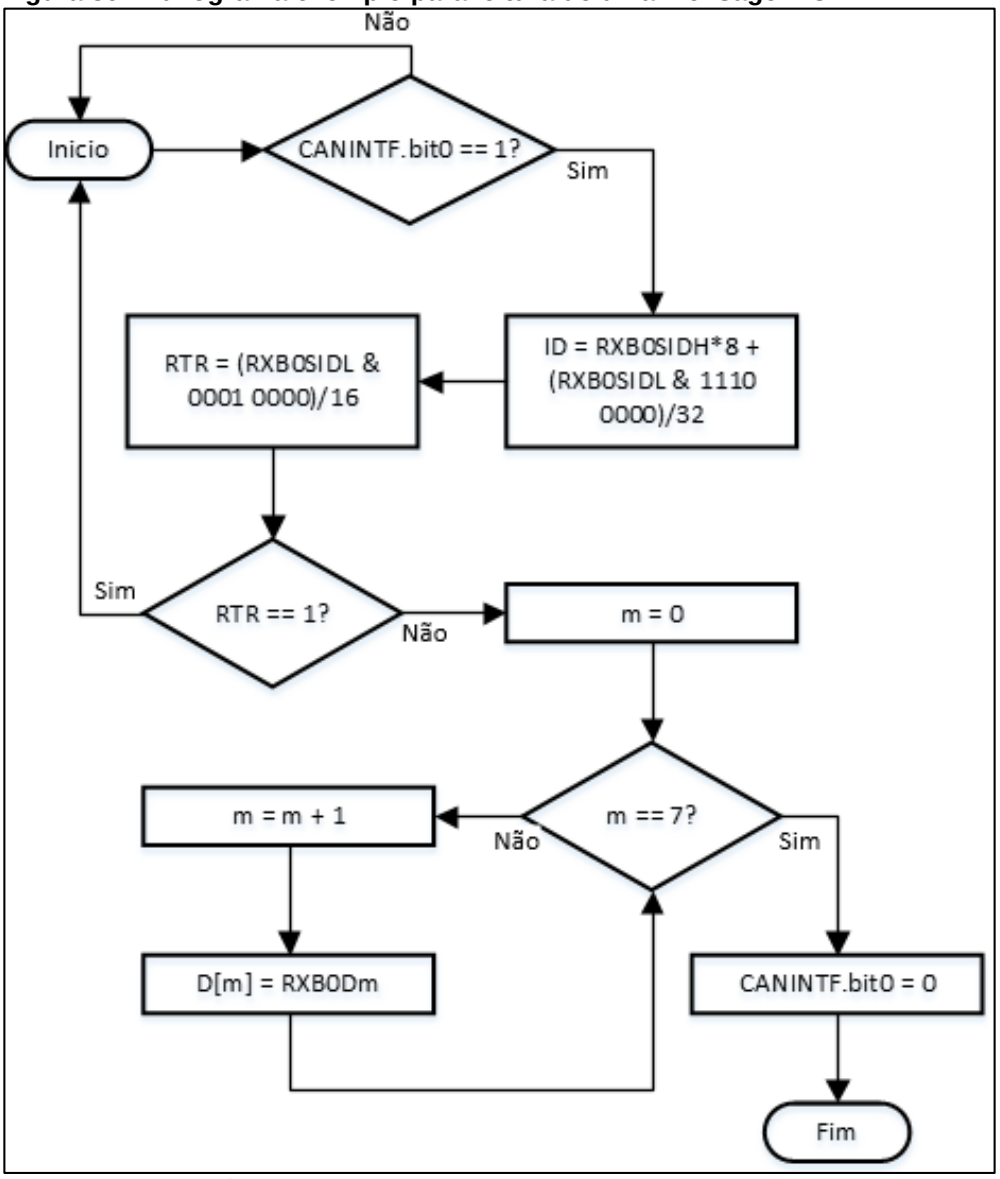
Figura 38: Estrutura dos registradores do campo de dados de mensagens recebidas.

Registrador: RXBnDm							
L	L	L	L	L	L	L	L
BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

Fonte: Adaptado de Microchip Technologies (2005).

Já de posse de todas as configurações e registradores necessários para a leitura de mensagens, a Figura 39 mostra um fluxograma representando um exemplo de como pode ser feita a leitura de uma mensagem recebida através do barramento CAN, com base no *buffer* RXB0.

Figura 39: Fluxograma exemplo para leitura de uma mensagem CAN.



Fonte: Autoria Própria.

2.6 DIAGNÓSTICO AUTOMOTIVO

As normas SAE J1979 (2006) junto a ISO 15031 (2006) estabelecem todos os padrões para que uma ferramenta de diagnóstico automotivo consiga se comunicar com o carro e trazer ao usuário as informações solicitadas, bem como a ISO 15765 (2005) regulamenta a estrutura que um barramento CAN de diagnóstico deve possuir. Dessa forma, existem instruções para a ferramenta descobrir o que se pode extrair de informações de um veículo bem como todas as diretrizes para processar os dados recebidos.

A norma SAE J1979 (2006) é composta de vários serviços, dentre eles o serviço de leitura de parâmetros que foi o utilizado neste trabalho para o desenvolvimento de toda a comunicação entre o veículo e o protótipo. Uma vez que a quantidade de serviços disponíveis é ampla, limitou-se apenas a obtenção de parâmetros importantes que serão destacados no decorrer do desenvolvimento do trabalho.

2.6.1 REQUISITOS PARA A REALIZAÇÃO DE DIAGNÓSTICOS

Segundo a norma SAE J1979 (2006), é considerado como diagnóstico automotivo todo tipo de trabalho realizado através de um computador com o objetivo de obter os códigos de falhas especificados na norma, realizar leitura de parâmetros e interagir com o veículo. Uma vez que nesse trabalho serão utilizados serviços de leitura através da interface de diagnóstico do veículo, então considera-se que será realizado um diagnóstico a fim de obter algumas informações disponíveis através de procedimentos detalhados pelo documento normativo.

Os veículos fabricados no Brasil e que possuem um barramento CAN para serviços de diagnóstico devem seguir a norma ISO 15765 (2006), que prevê todas as aplicações de diagnóstico possíveis aplicadas em barramentos desse tipo. Dessa forma, é previsto pela resolução CONAMA nº 324 (2004) que os fabricantes de veículos devem apresentar via interface de diagnóstico informações sobre vários sensores veiculares relacionados a emissão de poluentes e alguns outros, tais como informações do sensor de velocidade e de rotação do motor a partir do ano de 2010.

Uma vez que o veículo teste utilizado no desenvolvimento do protótipo foi fabricado no ano de 2014 e possui um barramento CAN para diagnóstico automotivo, então são aplicadas as normas ISO 15765 e SAE J1979

2.6.2 NORMAS DE PADRONIZAÇÃO

Até então foram citadas três normas importantes à realização de diagnósticos CAN: ISO 15031 (2006), ISO 15765 (2006) e SAE J1979 (2006), em que cada uma delas exerce um papel fundamental na execução de um correto diagnóstico automotivo. Há ainda uma regulamentação proposta pela SAE quanto ao conector padrão para executar as funções de diagnóstico (MI-JINKIM; JANG; YU, 2010), também presente no veículo utilizado como base para o desenvolvimento do protótipo.

2.6.2.1 NORMA ISO 15765

Toda a regulamentação e os pré-requisitos do barramento onde será realizado o diagnóstico veicular é proposto por essa norma. Baseada na ISO 11898 (2003), que regulamenta o protocolo CAN, a ISO 15765 (2006) propõe como este deve ser adaptado e instalado no veículo para tornar possível a execução de diagnósticos por ferramentas devidamente propostas para esse fim.

2.6.2.2 NORMAS SAE J1979 E ISO 15031

As normas SAE J1979 (2006) e ISO 15031 (2006) são as mais importantes para a construção de uma ferramenta básica de diagnóstico, pois contém todas as instruções que devem ser seguidas para a obtenção de dados provenientes do barramento CAN de diagnóstico. Uma vez que são estruturalmente semelhantes segundo consta na própria norma SAE J1979, então é importante destacar algumas diferenças entre elas.

Enquanto a norma ISO 15031 fornece uma base para a execução de diagnósticos, a SAE J1979 exemplifica e faz alguns adendos em alguns dos itens

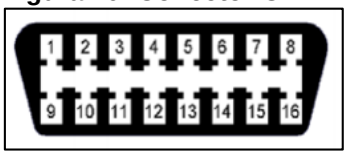
contidos na primeira, tais como exemplos de serviços de diagnóstico, adição de parâmetros e mudanças estruturais quanto a exibição de dados decimais.

Como base para o desenvolvimento do protótipo, foi considerado que o carro está dentro da regulamentação do barramento de diagnóstico proposta pela ISO 15765 e que segue toda a padronização de serviços e parâmetros propostos pela SAE J1979.

2.6.3 INTERFACE DE CONEXÃO PARA DIAGNÓSTICO

A realização de diagnóstico é realizada através do conector OBD (em inglês, *On-board Diagnostics* ou Conector de Diagnóstico de Bordo) padronizado pela SAE em 1988 e adotado mundialmente (MI-JINKIM; JANG; YU, 2010). Normalmente está localizado em áreas de fácil acesso no veículo tais como embaixo do painel, dentro da caixa de fusíveis, ou logo abaixo do capô (SNAP-ON, 2013) e, na Figura 40, é possível visualizar a estrutura e a disposição dos pinos no conector, detalhados na Tabela 12.

Figura 40: Conector OBD-II.



Fonte: Autoria Própria

Tabela 12: Disposição dos pinos no conector OBD-II

Pino	Descrição
1	Reservado ao fabricante
2	Linha positiva do barramento SAE J1850
3	Reservado ao fabricante
4	Aterramento em relação ao Chassis para alimentação
5	Aterramento em para a leitura de sinais
6	CAN_H compatível com a ISO 15765
7	K-LINE compatível com a ISO 9141
8	Reservado ao fabricante
9	Reservado ao fabricante
10	Linha negativa do barramento SAE J1850
11	Reservado ao fabricante
12	Reservado ao fabricante
13	Reservado ao fabricante
14	CAN_H compatível com a ISO 15765
15	L-LINE compatível com a ISO 9141
16	Alimentação proveniente da bateria

Fonte: Adaptado de Snap-On (2013).

Nota-se pela Tabela 12 que existem outros protocolos além do CAN para diagnóstico automotivo. Uma vez que não é possível através de uma interface CAN acessar a outros meios de diagnóstico, então o trabalho ficou limitado apenas a veículos que possuem o protocolo CAN como principal meio de obtenção de informações.

O conector além de possibilitar a conexão com o barramento de diagnóstico ainda disponibiliza ao usuário uma linha de alimentação que fornece a tensão da bateria do carro. Logo, é possível realizar a leitura dos parâmetros e alimentar o protótipo através do mesmo conector, realizando apenas um rebaixamento da tensão proveniente do OBD para uma compatível à porta de alimentação externa do Arduino (6 a 12V) através de um regulador de tensão, uma vez que, através de testes práticos, foi possível constatar que quando o motor encontra-se ligado, a tensão da bateria sobe de 12 para 14,4V, que extrapola a tensão máxima aceita pelo Arduino.

Como já citado, a camada física do protocolo CAN consiste em duas linhas de tensão, CAN_H e CAN_L, que variam suas tensões conforme o bit que trafega pela rede. Sendo assim, devido a diferenciação entre bits na camada física do barramento se tratar de uma tensão diferencial, é desnecessário que se utilize um aterramento, pois a tensão de referência é a diferencial entre CAN_H e CAL_L.

Definidas as conexões necessárias ao funcionamento do protótipo, então é possível através de quatro fios realizar a interface física entre o equipamento e o veículo, cuja alimentação, o aterramento desta, as linhas CAN_H e CAN_L devem ser conectados respectivamente aos pinos 16, 4, 6 e 14.

2.6.4 SERVIÇOS DE DIAGNÓSTICO

A norma SAE J1979 prevê vários serviços, tanto de leitura quanto de escrita em Unidades Centrais de Controle (ou, em inglês, *Engine Control Unit*, ECU) conectadas ao barramento CAN no qual o protótipo é conectado. A Tabela 13 mostra os principais serviços que podem ser executados através de uma ferramenta de diagnóstico.

Tabela 13: Serviços disponibilizados pela norma SAE J1979

Serviço	Descrição
$01	Faz a requisição do valor atual de dados da ECU de *Powertrain*.
$02	Faz a requisição do valor congelado pela ECU de *Powertrain*
$03	Faz a requisição de DTC's
$04	Reinicia as informações de falhas presentes na rede
$05	Faz a requisição dos valores de monitoramento do sensor de oxigênio
$06	Faz a requisição de monitoramento de bordo para testes
$07	Faz a requisição de de DTC's ocorridos durante a viagem atual ou passada
$08	Faz a requisição de controle de um sistema de bordo, teste ou componente
$09	Faz a requisição de informações do veículo
$0A	Faz a requisição de DTC's permanentes depois de uma limpeza geral

Fonte: Adaptado de SAE J1979.

Há inúmeras possibilidades de leitura de dados de diagnóstico em barramentos dedicados para estes fins. Nota-se que, através dos serviços destacados pela Tabela 13, é possível fazer a requisição de várias informações sobre DTC's (*Data Trouble Codes* em inglês, ou Códigos de Defeito) e de informações disponibilizadas por uma ECU responsável pelo diagnóstico do chassis do carro (ou *Powertrain*, em inglês).

Para cada tipo de requisição há um serviço diferente. No caso desse trabalho, foi utilizado apenas o serviço $01, pois se mostrou suficiente para a obtenção de grande parte das informações disponibilizadas ao usuário, conforme segue detalhado no próximo tópico.

2.6.4.1 SERVIÇO $01

Este serviço foi o único utilizado para a obtenção das informações processadas pelo protótipo. Consiste basicamente em quais parâmetros podem ser requisitados da ECU do veículo, bem como a forma com a qual estes devem ser interpretados pelo equipamento de diagnóstico.

A norma SAE J1979 trata cada parâmetro com um código PID (*Parameter IDentifier* em inglês, ou Identificador do Parâmetro), que deve ser requisitado junto do serviço $01. Da mesma forma, a interpretação da resposta também é sugerida pela norma e varia conforme o parâmetro solicitado, como mostra alguns exemplos apresentados pela Tabela 14.

Tabela 14: PIDs constantes na norma SAE J1979

PID	Descrição	Unidade	Interpretação
$00	Retorna os PIDs disponíveis de $01 a $20	-	Detalhado pela Figura 41
$20	Retorna os PIDs disponíveis de $21 a $40	-	Detalhado pela Figura 41
$40	Retorna os PIDs disponíveis de $41 a $60	-	Detalhado pela Figura 41
$04	Carga do motor calculada	%	$\frac{100}{255} * A$
$05	Temperatura do Líquido de Arrefecimento	ºC	$A - 40$
$0C	Rotação do Motor	RPM	$\frac{256 * A + B}{4}$
$0D	Velocidade do Veículo	Km/h	A
$1F	Tempo desde o acionamento do motor	s	$256 * A + B$
$2F	Nível de Combustível	%	$\frac{100}{255} * A$
$42	Tensão da Bateria	V	$\frac{(256 * A + B)}{1000}$
$46	Temperatura Ambiente	ºC	$A - 40$

Fonte: Adaptado de SAE J1979

Quando é feita a requisição de um PID utilizando o serviço $01, a ECU do veículo retorna uma mensagem CAN de dados com DLC de 8 *bytes* nomeados de A a E, onde cada byte carrega toda a informação requisitada ou apenas parte dela conforme apresentado pela coluna Interpretação da Tabela 14.

É importante, antes de fazer qualquer tipo de requisição, descobrir quais são os PIDs compatíveis com o veículo no qual a ferramenta de diagnóstico está conectada. Os PIDs $00, $20 e $40 foram os utilizados no veículo de teste, mas são apenas três de todos os PIDs utilizados para entender quais informações podem ser obtidas do veículo. Após a requisição desses PIDs, a ECU retorna uma resposta estruturada e interpretada conforme a Figura 41.

Figura 41: Exemplo de resposta à requisição do PID $00.

	Byte A								*Byte* B								*Byte* C								*Byte* D							
Hexadecimal	FF								1F								FA								01							
Binário	1	1	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0
Suportado?	S	S	S	S	S	S	S	S	N	N	N	S	S	S	S	S	S	S	S	S	S	N	S	N	N	N	N	N	N	N	N	S
PID	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20

Fonte: Adaptado de SAE J1979

Nota-se que no exemplo sugerido pela Figura 41 há compatibilidade com vários PIDs, porém há outros não disponibilizados pelo veículo. Dessa forma, a requisição pode até ser realizada, porém não haverá o devido retorno da ECU.

2.6.5 REQUISIÇÃO DE UM PID ATRAVÉS DE UM SERVIÇO

A requisição de um PID através de um serviço deve ser feita por meio de uma mensagem de dados CAN com DLC de 8 *bytes* (ISO 15765) estruturada conforme sugere a Figura 42. Vale lembrar que todo o campo de dados da mensagem deve ser preenchido, mesmo que com zeros, para que se respeite a condição do DLC.

Figura 42: Estrutura da mensagem de requisição de Diagnóstico.

Identificador	DLC	Dados							
		-	-	-	A	B	C	D	E
$7DF	8	Numero de Bytes de dados	Serviço	PID	$00	$00	$00	$00	$00

Fonte: Adaptado de ISO 15765.

O campo “Número de Bytes de Dados” deve ser preenchido conforme a quantidade de bytes que procedem na requisição. Sendo assim, para uma mensagem de requisição de um PID qualquer utilizando o serviço $01, este campo deve ser preenchido com 2, o que indica que a ECU deverá ler os próximos dois *bytes,* que indicam o serviço e o PID requisitado, e descartar o resto.

2.6.6 RESPOSTA A UMA REQUISIÇÃO

Uma vez enviada a requisição ao barramento, é necessário aguardar a resposta antes de enviar uma nova. Dessa forma, a ECU retornará uma mensagem de dados também com um DLC de 8 *bytes* porém de identificador diferente, que no caso da ECU do *Powertrain* conforme a norma ISO 15765, deve ser $7E8, estruturada de maneira semelhante à requisição, conforme mostra a Figura 43.

Figura 43: Estrutura da mensagem de resposta à requisição de Diagnóstico.

Identificador	DLC	Dados							
		-	-	-	A	B	C	D	E
$7E8	8	Numero de Bytes de dados	Serviço + $40	PID	?	?	?	?	?

Fonte: Adaptado de ISO 15765.

Nesse caso, o veículo retorna a resposta ao PID solicitado através do barramento e preenche automaticamente todos os campos. Um detalhe importante é que o serviço solicitado retorna acrescido de 40 unidades hexadecimais e, como dependendo do parâmetro podem ser enviados mais ou menos dados, o número de bytes de dados deve ser diferente do enviado, pois além do serviço e do PID, a resposta vem seguida pelos bytes de A a E.

Já definido todo o conteúdo necessário à programação do protótipo, foi dado início ao desenvolvimento do equipamento por completo, desde a programação dos módulo até a exibição dos parâmetros na tela TFT LCD.

3 DESENVOLVIMENTO

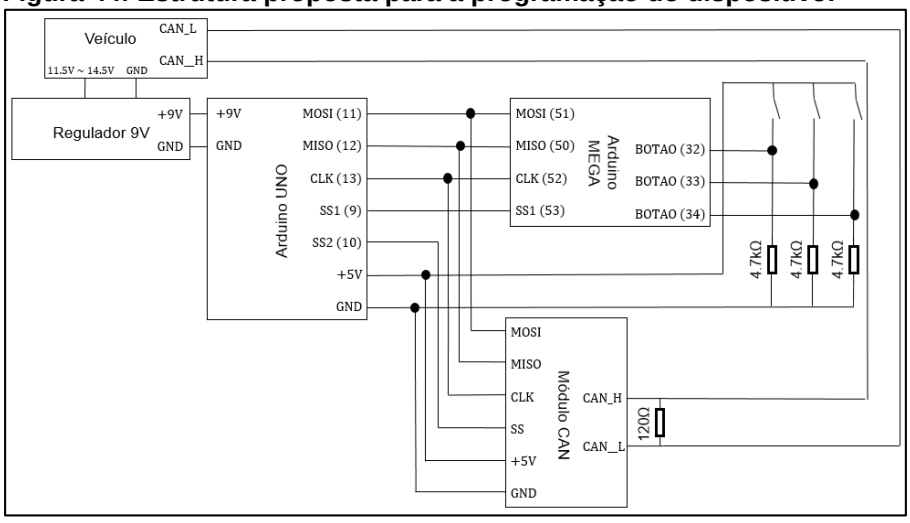
Neste capítulo é apresentado todo o desenvolvimento do trabalho, desde as definições estabelecidas pelo autor quanto a estrutura do protótipo até a implementação de todos os conceitos citados no capítulo anterior, necessários para atingir o objetivo proposto pelo trabalho.

O primeiro passo para que o desenvolvimento pudesse ser iniciado foi a definição de uma arquitetura de interconexão entre todos os elementos necessários ao funcionamento do dispositivo.

Optou-se por utilizar o Arduino MEGA® para controlar a tela visto que a carga de trabalho para o microcontrolador é elevada e a quantidade de variáveis para processamento segue a mesma linha. Logo, o Arduino UNO® foi utilizado como dispositivo mestre responsável pela captura dos dados provenientes da rede CAN e o Arduino MEGA®, bem como o módulo CAN foram configurados para operar como escravos dependentes dos dados processados pelo Arduino UNO®. Todos os elementos citados neste tópico são detalhados pelo Capítulo 2 deste trabalho.

A Figura 44 mostra como ficou a estrutura final do dispositivo junto a todas as ligações e os botões que foram adicionados para permitir a personalização do computador de bordo. Nota-se que foram adicionados alguns resistores de *pull-up* para evitar que os botões fossem acionados involuntariamente durante o funcionamento do protótipo.

Figura 44: Estrutura proposta para a programação do dispositivo.



Fonte: Autoria Própria.

Pode ser visto, ainda na Figura 44, que foram pré-definidos todos os pinos que serão utilizados dos Arduinos e dos módulos. A tela não foi inserida no diagrama pelo fato dela encaixar-se perfeitamente no Arduino Mega, que ocupa todos os pinos de alimentação, os digitais de 1 a 13 e os analógicos de 1 a 6, tanto que optou-se por alimentar os botões a partir da fonte de 5V disponibilizada no Arduino UNO.

Uma vez definida a arquitetura, foi possível dar início a programação de cada um dos elementos, a começar pela definição do dispositivo mestre e dos dispositivos escravos.

3.1 CONFIGURAÇÃO DOS DISPOSITIVOS COMO MESTRE E ESCRAVO

Uma vez que apenas um Arduino UNO foi insuficiente para o processamento tanto da tela TFT LCD quanto das informações provenientes do microcontrolador CAN devido a quantidade de pinos de entrada e saída disponíveis, bem como a quantidade de memória de programas deste ser limitada, optou-se por trabalhar em uma arquitetura de processamento com um Mestre, o Arduino UNO, e dois escravos, o Arduino MEGA e o microcontrolador CAN, cuja configuração será mostrada no decorrer do trabalho.

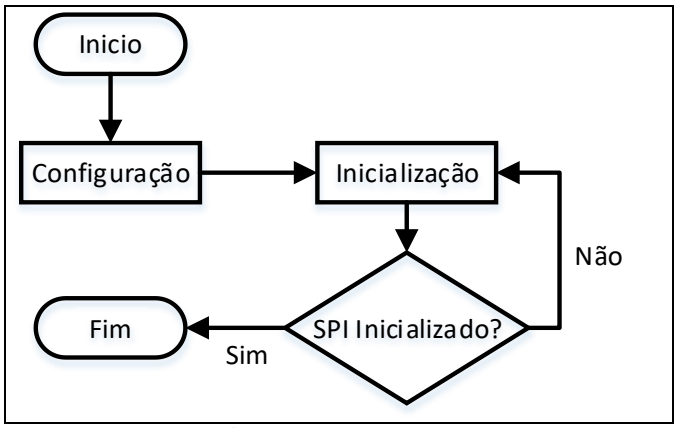
3.1.1 ARDUINO UNO MESTRE

A configuração do Arduino UNO para funcionar como dispositivo SPI mestre é simples, visto que já existem bibliotecas prontas para esse fim. A utilizada é a mesma disponibilizada pelo próprio ambiente de desenvolvimento do *hardware*, a conhecida Arduino IDE (em inglês *Integrated Development Environment*, ou Ambiente de Desenvolvimento Integrado), que possui comandos bem definidos para a comunicação SPI.

O envio e o recebimento de dados e configuração dos modos de operação, bem como a frequência do *clock* e a ordem de envio dos bits são feitos a partir de funções predefinidas pela biblioteca de programação “SPI.h”, contida dentro da pasta “libraries” no diretório onde a Arduino IDE encontra-se instalada. O ambiente de desenvolvimento pode ser encontrado facilmente no próprio site do *hardware*, em *www.arduino.cc*.

Uma vez definida a biblioteca utilizada para a configuração do SPI, é necessário então inicializar a sua configuração para que o mestre consiga se comunicar adequadamente com todos os escravos. Dessa forma, o procedimento de inicialização do módulo SPI do Arduino foi configurado de acordo com o fluxograma apresentado na Figura 45.

Figura 45: Fluxograma de inicialização do SPI.



Fonte: Autoria Própria

A configuração do SPI consiste de duas etapas, a declaração dos pinos do Arduino utilizados na comunicação e a definição dos parâmetros de configuração. Optou-se por comunicar o Arduino com os periféricos a uma frequência de 4Mhz, por ser uma velocidade suficiente e suportada por todos os elementos utilizados no desenvolvimento do protótipo.

Definida a frequência, foi necessário estabelecer a ordem com a qual os bits são enviados, que nesse caso foi do bit menos significativo do byte ao bit mais significativo, bem como o modo de operação do SPI, escolhido como 0 para garantir a compatibilidade com todos os elementos do circuito. Para evitar que os escravos fossem ativados em momentos indevidos, estes foram inicializados em nível lógico alto, o que evita que este receba ou mande qualquer tipo de dado.

Os pinos MOSI, MISO e CLK são padrões do Arduino para estas funções, sendo assim, buscou-se quanto ao SPI trabalhar de acordo com a compatibilidade do microcontrolador do Arduino Uno, o ATMEGA328P (ARDUINO, 2016). Logo, os pinos 11 e 13 foram configurados como saída e correspondem aos pinos MOSI e CLK respectivamente, enquanto que o pino MISO foi configurado no pino 12 como entrada, pois esse é o responsável por fornecer o caminho para a informação vinda do escravo.

A fim de garantir uma sincronização de dados com o Arduino MEGA, utilizado nesse caso como escravo, foram pré-determinados bytes de início de transmissão e de fim de transmissão. Uma vez que o Arduino MEGA escravo recebe os dados, então após uma sequência fixa ele os interpreta como dados úteis e após outra sequência detecta o fim da transmissão, conforme sugere a Figura 46.

Figura 46: Estrutura dos dados enviados para o Arduino Mega Escravo.

Bytes Transmitidos								
71	72	73	74	DADOS	200	210	220	230

Fonte: Autoria Própria.

Para cada um dos escravos utilizados no trabalho, foi definido um pino SS responsável por sua ativação no barramento. Dessa forma, foi utilizado apenas um barramento para todos os escravos, o que possibilitou ao mestre o total controle sobre cada um dos periféricos, ativando-os ou inativando-os a medida da necessidade do uso de cada um.

O pino SS é de responsabilidade do programador e foi definido como o pino 10 para o microcontrolador CAN e como o pino 9 para o Arduino MEGA Escravo, permanecendo nessas condições por durante todo o desenvolvimento.

3.1.2 ARDUINO MEGA ESCRAVO

Uma vez que não há bibliotecas padrão para operar um Arduino como escravo em um barramento SPI, a solução encontrada para resolver essa questão foi encontrada na própria folha de dados que descreve o microcontrolador Atmel® ATMEGA2560 (MICROCHIPE TECHNOLOGIES, 2005), microcontrolador principal do Arduino.

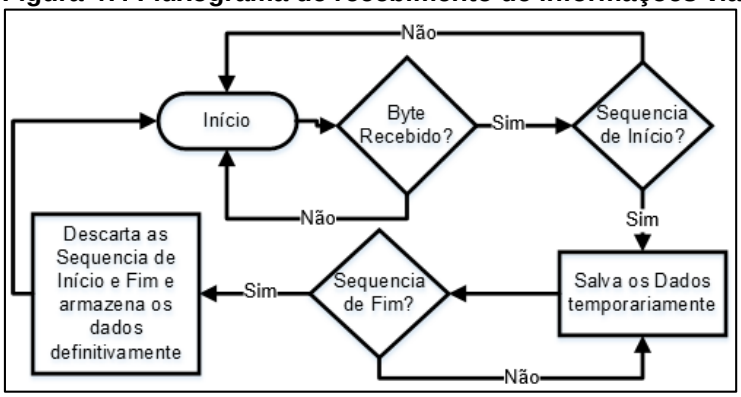
Para tornar possível o recebimento de informações via SPI pelo Arduino Mega, foi preciso programar diretamente dois registradores específicos do microcontrolador, SPCR e SPSR, bem como realizar a leitura dos dados recebidos através do registrador SPDR.

Uma vez que o escravo não sabe quando receberá dados pelo barramento, deve ser gerada uma interrupção a cada tentativa de recebimento, ou seja, quando o escravo é ativado pelo mestre através do pino SS. Sendo assim, quando a transmissão é finalizada, então o microcontrolador volta a operar normalmente executando suas outras funções.

Conforme sugestão da própria folha de dados do microcontrolador ATMEGA2560, os registradores SPCR e SPSR foram configurados com os valores binários 1100 0000 e 0000 0001, o que foi suficiente para habilitar o Arduino MEGA como dispositivo SPI escravo e então operar com o recebimento de dados.

Para que a recepção de informações ocorresse de forma correta, a sincronização das informações recebidas via SPI foi implementada através de uma rotina de reconhecimento da sequência de bits de início e de fim de transmissão, mostrada pelo fluxograma da Figura 47.

Figura 47: Fluxograma de recebimento de informações via SPI.



Fonte: Autoria Própria

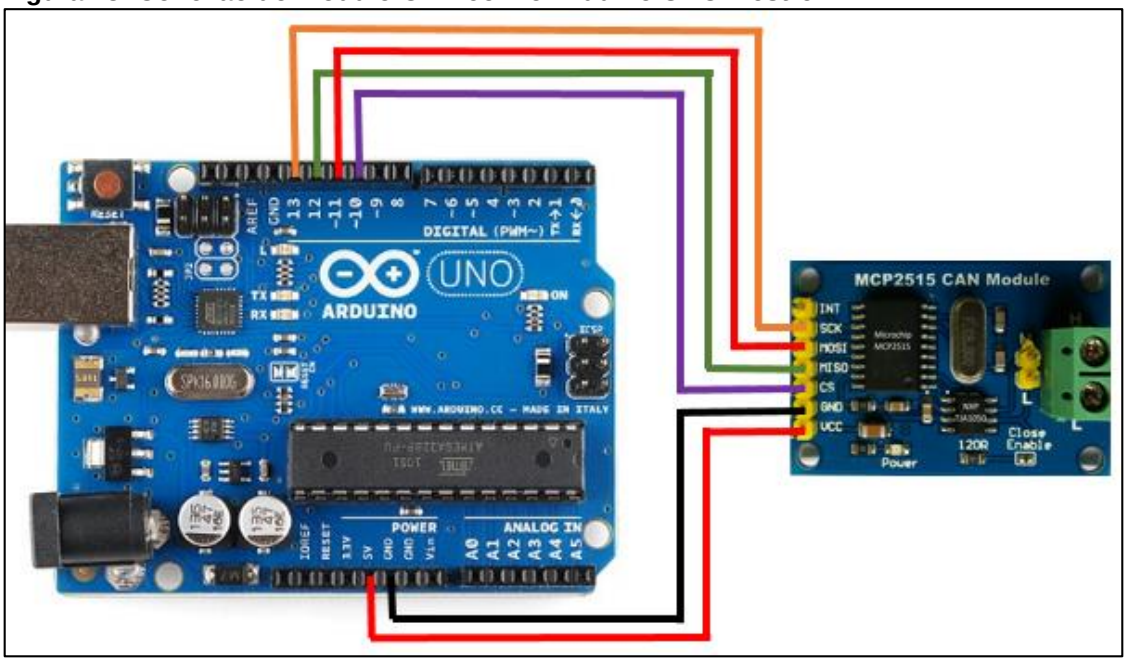
Nota-se que são descartados os bytes de início e fim de transmissão, o que torna possível salvar apenas os dados úteis e utilizá-los na decodificação das informações provenientes do barramento CAN do veículo recebidas por meio do Arduino UNO Mestre pelo Arduino MEGA Escravo.

3.2 INTERFACE DE CONEXÃO ENTRE O ARDUINO MESTRE E O MÓDULO CAN

Uma vez que o módulo CAN será um escravo, então é importante definir um pino de ativação e atribuir a ele. Nesse caso, o pino escolhido foi o 10 por estar

disponível e próximo aos canais principais de ligação do Arduino, o que resultou em uma arquitetura inicial semelhante à apresentada na Figura 48.

Figura 48: Conexão do módulo CAN com o Arduino UNO Mestre.



Fonte: Autoria Própria

3.3 PROGRAMAÇÃO DO MICROCONTROLADOR CAN MCP2515

Por questões de organização, foi de extrema importância inicialmente mapear todos os registradores e as instruções SPI fundamentais para o desenvolvimento de toda a lógica de programação do sistema. Uma vez definidos todos os elementos disponibilizados pela folha de dados, o desenvolvimento das funções acaba se tornando visualmente melhor e mais compreensível.

Nesta tarefa, foi criado um arquivo nomeado como “MCP2515_config.h” dentro do mesmo diretório onde foi salvo o código principal. Após a criação, foi definido um cabeçalho explicativo sobre o arquivo e, em cada linha, foi definido um nome para cada um dos endereços.

Como exemplo, pode-se supor os registradores CANINTE e CANINTF. Segundo a folha de dados do MCP2515, o endereço desses registradores são os hexadecimais 2B e 2C, logo foram criadas duas linhas no arquivo MCP2515_config.h seguindo a estrutura apresentada pela Figura 49.

Figura 49: Formato geral de mapeamento dos registradores.

```
//---------------------Interrupt Registers Adresses-----------------//

#define     CANINTE      0x2B    /**/
#define     CANINTF      0x2C    /**/
```

Fonte: Autoria Própria

Terminado o arquivo, esse foi referenciado no código principal para que o compilador tivesse as informações suficientes sobre cada uma das nomenclaturas utilizadas para os registradores. Optou-se por empregar como nomenclatura o próprio nome dos registradores para tornar mais fácil referenciá-los dentro das funções que os utilizarão.

3.3.1 INICIALIZAÇÃO DO MICROCONTROLADOR

Uma vez realizada a conexão dos componentes e o mapeamento dos registradores, já é possível começar a configurar o módulo CAN que fará a interface entre o barramento CAN do veículo teste e o sistema de processamento de dados, que nesse caso será o próprio Arduino UNO®.

Segundo a folha de dados do MCP2515, há quatro registradores fundamentais para a inicialização do microcontrolador e são estes CNF1, CNF2, CNF3 e CANCTRL. Cada um possui uma função específica para a configuração da velocidade da rede CAN ao qual a ferramenta desenvolvida nesse trabalho será conectada.

Para a configuração da velocidade da rede CAN é fundamental estimar o período de cada bit recebido, pois é a partir desse tempo que será configurada a velocidade na qual o dispositivo vai operar. Sabe-se que as velocidades padrões exigidas pela ISO 15765 são 500 Kbits/s ou 250 Kbits/s e, além destas, foi definida outra velocidade de 125 Kbits/s apenas por precaução no caso de incompatibilidade com alguma rede. Dessa forma, a velocidade nominal da rede (em inglês *Normal Bit Rate*, NBR) pode ser calculada utilizando a Equação (1),

$$NBR = f_{bit} = \frac{1}{T_{bit}} \quad (1)$$

onde NBR representa a velocidade da rede (bits por segundo), f_{bit} a frequência de bits e T_{bit} o período de cada bit, em segundos.

Rearranjando a Equação (1), obtém-se a Equação (2) padrão para a definição de T_{bit}, que pode ser aplicada para as velocidades anteriormente citadas.

$$T_{bit} = \frac{1}{NBR} \tag{2}$$

Uma vez definido cálculo para a obtenção do período nominal de cada bit, foi possível então, utilizando a Equação (2), estabelecer os períodos nominais adequados de cada bit referentes às velocidades citadas, ou seja, 8000ns para 125 Kbit/s, 4000ns para 250 Kbits/s e 2000ns para 500 Kbits/s.

De acordo com a folha de dados do MCP2515 e a norma ISO 15765, o período real de cada bit deve levar em consideração a soma de quatro segmentos: Segmento de sincronização, de propagação, de fase 1 e de fase 2, medidos em uma unidade chamada de *Time Quantum* (TQ), que é baseada na frequência do oscilador instalado no microcontrolador e em um *prescaler*, que nada mais é que uma subdivisão da frequência do mesmo oscilador com a aplicação voltada à obtenção de maiores tempos.

A Equação (3) apresenta como é possível calcular o período de 1 TQ baseado na frequência do oscilador (f_{osc}) e no *prescaler* (BRP),

$$TQ = \frac{2(BRP + 1)}{f_{osc}} \tag{3}$$

na qual foi utilizado o cristal oscilador de 8MHz já instalado no módulo CAN devido a este ser compatível às velocidades utilizadas e suficiente para as aplicações descritas por este trabalho.

Foi escolhido BRP = 0 para 500 Kbits/s, BRP = 1 para 250 Kbits/s e BRP = 3 para 125 Kbits/s que geram TQ iguais a 250, 500 e 1000ns para as três velocidades citadas anteriormente respectivamente.

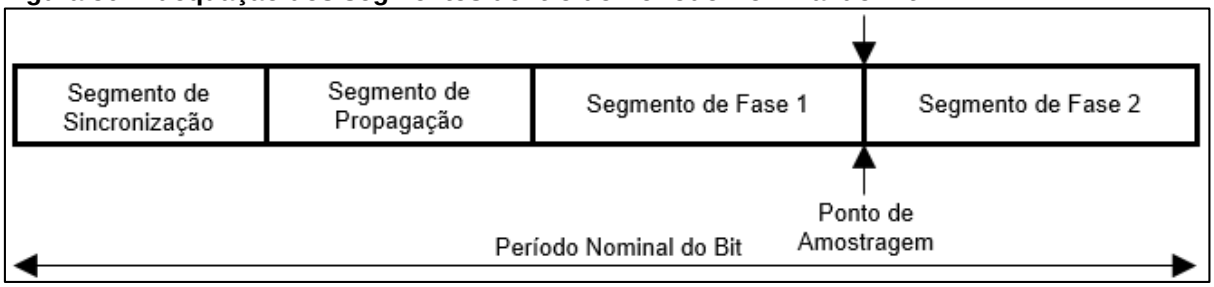
Uma vez definido o período TQ e o BRP utilizado nas configurações, então é possível identificar a quantidade de TQs disponíveis para os bits em cada uma das velocidades nominais, conhecida por *Bit Timing* (BT, ou Tempo de bit). Segundo a Equação (4), apresentada na folha de dados do MCP2515,

$$BT = \frac{T_{bit}}{TQ} \tag{4}$$

o BT para 125 Kbits/s, 250 Kbits/s e 500 Kbits/s é igual e resulta em 8TQs.

Definida a quantidade máxima de TQs para cada uma das velocidades, agora é possível definir como o BT será distribuído para cada segmento, sendo que alguns requisitos devem ser seguidos conforme rege a folha de dados do MCP2515. A Figura 50 mostra como são divididos todos os segmentos dentro do período nominal do bit.

Figura 50: Adequação dos segmentos dentro do Período Nominal do Bit



Fonte: Adaptado de Microchip Technologies (2005).

O segmento de sincronização é utilizado para a sincronização de todos os nós dentro de um barramento CAN, responsável por compensar o atraso das transições entre bits recessivos e dominantes. Este segmento tem período já definido com base nas características internas do microcontrolador e independente do período do bit sempre assume 1 TQ.

Uma vez que existem vários nós dentro de um barramento CAN, há um atraso físico provocado pelo próprio *hardware* (MICROCHIP TECHNOLOGIES, 2005). O segmento de propagação é o responsável por realizar a compensação desse atraso visto que, conforme o *hardware* ao qual o microcontrolador está atrelado, este pode ser maior ou menor e programado entre 1 e 8TQs, mas é recomendado pela folha de dados o valor de 2TQs.

Os segmentos de fase são os responsáveis por compensar problemas na leitura das bordas dos bits. Sendo assim, existem dois segmentos de fase para que haja um ponto certo de amostragem do bit, que ocorre depois do segmento de fase 1.

Uma vez definidos os segmentos do período nominal do bit, é importante seguir algumas exigências do MCP2515, recomendadas pela folha de dados e exibidas na Tabela 15, para que não haja nenhum tipo de conflito ou configuração incompatível.

Tabela 15: Recomendações de configuração dos períodos dos segmentos.

Segmento	Valor Mínimo (TQ)	Valor Máximo (TQ)	Recomendação
Sincronização	1	1	O período do Segmento de Sincronização é fixo em 1 TQ.
Propagação	1	8	Programável, mas recomenda-se 2 TQs.
Fase 1	1	8	Valor definido arbitrariamente.
Fase 2	2	8	Deve ser maior que o período do Segmento de Fase 1 somado ao período do Segmento de Propagação.

Fonte: Adaptado de Microchip Technologies (2005).

Há ainda uma configuração referente à sincronização do microcontrolador com a rede CAN chamada Largura do Pulso de Sincronização (em inglês *Synchronization Jump Width*, ou SJW), cujo valor recomendado é de 1 TQ e atende todos os casos em condições normais segundo a folha de dados do MCP2515.

Já com todas as configurações definidas para três velocidades, é possível dar início a rotina de inicialização do microcontrolador. Dessa forma, serão utilizados três registradores principais, CNF1, CNF2 e CNF3 e o registrador CANCTRL, responsável por colocar o MCP2515 em modo de configuração. A Figura apresenta a estrutura dos registradores principais.

Figura 51a: Principais registradores para a configuração da velocidade da rede CAN (continua)

Registrador: CNF1							
L/E	L/E	L/E	L/E	L/E	L/E	L/E	L/E
SJW1	SJW0	BRP5	BRP4	BRP3	BRP2	BRP1	BRP0
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

Figura 51b: Principais registradores para a configuração da velocidade da rede CAN (conclusão)

Registrador: CNF2							
L/E	L/E	L/E	L/E	L/E	L/E	L/E	L/E
BTLMODE	SAM	PHSEG12	PHSEG11	PHSEG10	PRSEG2	PRSEG1	PRSEG0
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

Registrador: CNF3							
L/E	L/E	L/E	L/E	L/E	L/E	L/E	L/E
SOF	WAKFIL	-	-	-	PHSEG22	PHSEG21	PHSEG20
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

Fonte: Autoria Própria

3.3.1.1 REGISTRADOR CNF1

O registrador CNF1 é o que contém as configurações sobre o *prescaler* e o SJW. Uma vez que foram definidas três velocidades diferentes, foram utilizados também três *prescalers* distintos. Dessa forma, deve-se definir em qual das velocidades de rede CAN o microcontrolador irá operar para então dar continuidade ao processo de inicialização.

Como SJW é igual a 1, então os bits 7 e 6 foram configurados para 0 e 0, conforme sugere a folha de dados do MCP2515. Essa configuração é comum a todas as velocidades, porém os bits BRP de 5 a 0 serão configurados para 000000 em 500 Kbits/s, para 000001 em 250 Kbits/s e 000011 para 125 Kbits/s.

3.3.1.2 REGISTRADOR CNF2

Todas as configurações referentes ao segmento de propagação e de fase 1 serão definidas nesse registrador. Como para todas as velocidades CAN sugeridas o BT foi de 8TQs, então essa configuração se manterá independente da taxa de bits que será utilizada.

Tanto o segmento de propagação quanto os de fase 1 e 2 devem ser configurados no registrador decrescidos de uma unidade. Sendo assim, o

recomendado para a propagação é 2TQs, logo os bits 2, 1 e 0 serão configurados para 0, 0 e 1 respectivamente, que, em unidades decimais, significa 1.

O mesmo acontece com o segmento de fase 1, que pode ser definido arbitrariamente. Dessa forma, foi considerado 3TQs para que no total não fosse ultrapassado o valor máximo de 8TQs calculados anteriormente para o período nominal de bit. Considerando 3TQs, então a configuração dos bits 5, 4 e 3 deve ser 0, 1 e 0 respectivamente.

Para configurar o ponto de amostragem após o término do segmento de fase 1 então o bit 6 deve ser 0, o que garante que o bit CAN será amostrado apenas uma vez e não três. Da mesma forma, o bit 7 deve ser configurado como 1 para habilitar a personalização do segmento de fase 2, feita no registrador CNF3.

3.3.1.3 REGISTRADOR CNF3

No registrador CNF3 são configuradas algumas funções internas do microcontrolador, como a função de acordar o MCP2515 ao receber uma mensagem ou de indicar em um dos pinos do componente que uma mensagem foi recebida. Como nenhuma dessas funções foram usadas no desenvolvimento do trabalho, então tanto o bit 7 quanto o bit 6 foram configurados como 0.

A configuração mais importante desse registrador é o período do segmento de fase 2. Uma vez que esse foi definido como 2TQs, então a configuração dos bits 2, 1 e 0 deve ser 0, 0 e 1 respectivamente.

3.3.1.4 MODO DE CONFIGURAÇÃO

Alguns registradores requerem que o microcontrolador esteja em modo de configuração para que estes possam ser alterados. Para entrar nesse modo, deve-se configurar os bits 7, 6 e 5 do registrador CANCTRL para 1, 0 e 0 respectivamente, porém é interessante que qualquer tipo de transmissão que por ventura esteja sendo executada seja cancelada, logo o bit 4 deve ser configurado como 1 antes do início da configuração.

Há ainda outros bits dentro desse registrador, porém no momento estes são irrelevantes. Vale lembrar que após os registradores CNF1, CNF2 e CNF3 serem completamente configurados, o microcontrolador deve voltar a trabalhar no modo de operação normal, ou seja, todos os bits de CANCTRL devem ser definidos como 0.

Já de posse de todas as definições dos registradores de configuração, foi elaborada a Tabela 16, onde é possível visualizar com clareza qual a configuração exata de cada registrador em relação às velocidades predefinidas. Nota-se pelos cálculos que o BT de 8TQs não foi ultrapassado, pois a soma de todos os segmentos deu exatamente o período nominal dos bits para cada uma das velocidades.

Tabela 16: Tabela resumo dos registradores de configuração de velocidade.

Velocidades	Registrador	Configuração
125 Kbits/s	CNF1	0000 0011
	CNF2	1001 0001
	CNF3	0000 0001
250 Kbits/s	CNF1	0000 0001
	CNF2	1001 0001
	CNF3	0000 0001
500 Kbits/s	CNF1	0000 0000
	CNF2	1001 0001
	CNF3	0000 0001

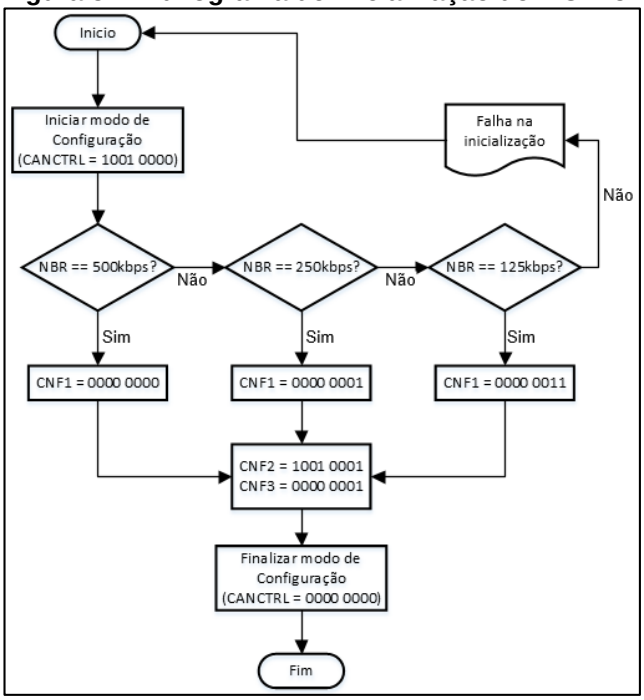
Fonte: Autoria Própria.

3.3.1.5 ROTINA DE INICIALIZAÇÃO

Para que o microcontrolador pudesse operar corretamente, então é importante que inicialmente sejam transferidos todos os dados de configuração já definidos anteriormente. Logo, é na rotina de inicialização que é definida a velocidade de operação da rede CAN, parâmetro fundamental para que ocorra o sincronismo do protótipo com o veículo.

Nessa etapa é realizada a transferência de todas as configurações já definidas para o microcontrolador via SPI. Para a realização dessa atividade, foi implementada no Arduino UNO® Mestre uma rotina de inicialização baseada no fluxograma apresentado na Figura 52.

Figura 52: Fluxograma de inicialização do MCP2515.



Fonte: Autoria Própria.

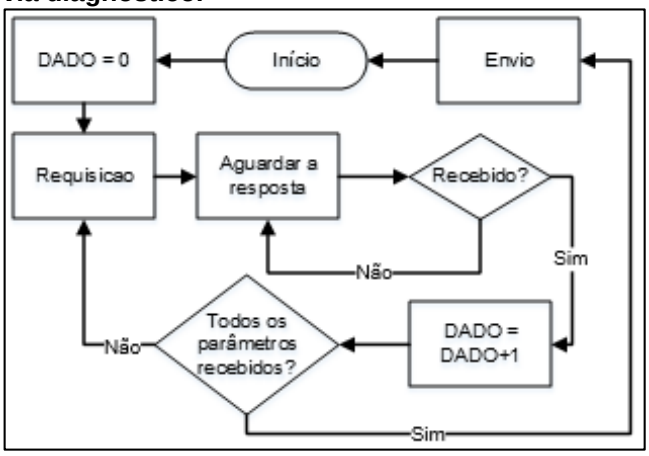
Uma vez definida a velocidade nominal da rede CAN para uma das calculadas no tópico anterior, então é possível dar continuidade e partir para o envio e o recebimento de mensagens CAN utilizando o barramento real de diagnóstico.

3.4 PROCESSO ADOTADO PARA A REQUISIÇÃO DE DADOS

O primeiro passo foi realizar a requisição dos PIDs $00, $20 e $40 do veículo teste para descobrir qual a compatibilidade e o que pode ser obtido. De todos os acusados como válidos, foram escolhidos alguns julgados interessantes, já apresentados anteriormente na Tabela 14, e então foi dado início à leitura dos parâmetros.

Nota-se que, por se tratar de leitura em tempo real, é importante que todos os dados sejam requisitados, recebidos e enviados ao Arduino MEGA o mais rápido possível e sem nenhum tipo de erro. Isso foi feito aguardando a resposta da ECU do veículo para cada um dos parâmetros e os armazenando temporariamente no Arduino UNO Mestre, para que logo em seguida esse os enviasse via SPI e realizasse novamente uma leitura da CAN, repetindo todo o processo que pode ser resumido através do fluxograma apresentado pela Figura 53.

Figura 53: Fluxograma para a obtenção dos dados via diagnóstico.



Fonte: Autoria Própria

3.5 MANIPULAÇÃO DOS DADOS RECEBIDOS DO VEÍCULO

Uma vez que é necessária a decodificação dos dados obtidos via diagnóstico, então foram estabelecidas algumas grandezas a serem medidas e mostradas na tela TFT LCD para o usuário, detalhadas nos tópicos a seguir.

A fim de evitar cálculos desnecessários e otimizar o processamento do Arduino MEGA escravo, responsável pela decodificação dos dados recebidos via SPI e pelo processamento da tela TFT LCD, foi programada uma interrupção geral de 100 ms no microcontrolador para que, a cada múltiplo desse tempo, algumas grandezas fossem calculadas. Foi definida uma prioridade para cada conforme sua dinamicidade.

A informação recebida pelo escravo consiste de uma sequência de bytes formatada conforme mostra a Figura 54. Algumas das grandezas decodificadas pelo Arduino são constituídas por mais de um *byte* de informação, logo estas são devidamente alocadas conforme segue na Figura 54.

Figura 54: Alocação dos dados dentro da informação recebida via SPI.

Início da Transmissão				Velocidade	Temperatura de arrefecimento	Rotação do Motor		Tempo de acionamento do motor		Nível de Combustível	Temperatura Ambiente	Carga do motor	Tensão da Bateria		Fim da Transmissão			
71	72	73	74	Byte A	Byte A	Byte A	Byte B	Byte A	Byte B	Byte A	Byte A	Byte A	Byte A	Byte B	200	210	220	230

Fonte: Autoria Própria.

O que difere os *bytes* A de B é basicamente a significância de cada no valor final da grandeza medida. Uma vez que um dado é composto de mais de dois *bytes*, como é o caso da rotação do motor por exemplo, o Byte A sempre será o mais significativo, enquanto o Byte B sempre será o menos significativo.

Uma vez que são dados diferentes, então é importante definir que cada grandeza tem seus bytes, logo o Byte A da velocidade não será necessariamente igual ao Byte A da tensão da bateria, por exemplo.

3.5.1 VELOCIDADE

A velocidade do veículo é obtida utilizando o PID $0D junto ao serviço $01. Essa requisição retorna um *byte* de dados que dispensa qualquer tipo de decodificação visto que o valor de retorno é o mesmo contido dentro do *byte* A, logo pode variar de 0 a 255 Km/h com precisão de 1 Km/h.

Esta variável foi considerada como de alta dinâmica, visto que a velocidade de um veículo nem sempre é constante. Logo, é atualizada no display a cada 500 milissegundos para que o usuário possa acompanhar em tempo real a velocidade na qual o veículo se encontra de uma forma precisa e digital.

3.5.2 TEMPERATURA DE ARREFECIMENTO

A temperatura do líquido de arrefecimento do motor é processada a cada 1 segundo e obtida através do PID $05 e do serviço $01, porém há uma fórmula especificada na norma SAE J1979 para decodificar a informação disponibilizada pelo barramento quanto a essa requisição, mostrada pela Equação (5),

$$Temp.Arrefecimento = A - 40 \quad (5)$$

onde A significa o *byte* referente à requisição.

Dessa forma, nota-se que a temperatura real do líquido de arrefecimento corresponde ao byte enviado pela ECU do veículo decrescido de 40 unidades que utiliza por padrão graus Celsius como unidade.

3.5.3 ROTAÇÃO DO MOTOR

A rotação do motor pode ser obtida através do PID $0C e do serviço $01, assim como a velocidade, a temperatura do liquido de arrefecimento e todas as outras grandezas obtidas do carro através do barramento CAN. Diferente das anteriores, o veículo retorna dois *bytes* de dados referentes ao valor real da rotação do motor do veículo, que pode ser decodificada utilizando a Equação (6) e é fornecida em rotações por minuto,

$$Rotação\ do\ Motor = \frac{A.256 + B}{4} \tag{6}$$

em que A representa o byte mais significativo e B o *byte* menos significativo.

Por existir um conta-giros na tela principal do dispositivo, então essa foi considerada como uma variável dinâmica e é atualizada sempre que lida do barramento.

3.5.4 TEMPO DESDE O ACIONAMENTO DO MOTOR

Essa grandeza representa o tempo total em segundos pelo qual o motor permaneceu ligado, e é reiniciado cada vez que o motor é desligado. Esse tempo é fornecido através do serviço $01 pelo PID $1F e deve ser decodificado utilizando a Equação (7),

$$Tempo\ (segundos) = A.256 + B \tag{7}$$

na qual A representa o *byte* mais significativo e B o menos significativo.

A exibição desse parâmetro para o usuário é feita em minutos, logo foi necessária a conversão desse valor, realizada utilizando a Equação (8) e exibida na tela apenas em unidades inteiras.

$$Tempo\ (minutos) = \frac{Tempo\ (segundos)}{60} \tag{8}$$

Por ser atualizada no barramento CAN apenas de segundo em segundo, então é realizada uma amostragem a cada 1 segundo dessa informação para então realizar os cálculos citados anteriormente.

3.5.5 NÍVEL DE COMBUSTÍVEL

O nível de combustível do veículo pode ser obtido através do serviço $01 pelo PID $2F, e é calculado a cada 2 segundos. O veículo retorna um *byte* A que deve ser decodificado através da Equação (9), cujo resultado fornece o percentual de líquido presente no tanque de armazenamento de combustível.

$$Nível\ de\ Combustível = \frac{100}{255} A \tag{9}$$

3.5.6 TEMPERATURA AMBIENTE

A obtenção da temperatura ambiente é realizada através da requisição do PID $46 pelo serviço $01, sempre calculada a cada 2 segundos. O retorno proveniente do barramento CAN de diagnóstico do veículo é composto de um *byte* A que pode ser decodificado, revelando a real temperatura ambiente em graus Celsius, através da Equação (10).

$$Temp.Ambiente = A - 40 \tag{10}$$

3.5.7 CARGA DO MOTOR CALCULADA

A carga do motor é fornecida pelo próprio barramento do veículo, cujo cálculo é feito pela ECU que retorna o valor já pronto, porém codificado, para a leitura. A requisição da carga do motor pode ser feita através do PID $04 junto ao serviço $01, e retorna o valor percentual calculado da carga do motor do veículo cuja interpretação é feita a cada 200 milissegundos.

Essa informação é retornada pela ECU através de um *byte* de dados A, cuja decodificação deve ser realizada através da Equação (11).

$$Carga\ do\ motor = \frac{100}{255}A \tag{11}$$

3.5.8 DISTÂNCIA PERCORRIDA

Já de posse da velocidade do veículo bem como do intervalo de tempo no qual a velocidade é amostrada, então é possível calcular a distância percorrida pelo carro através da Equação (12) (HALLIDAY; RESNICK; WALKER, 2016),

$$S = S_0 + \frac{(V + V_0)t}{2} \tag{12}$$

na qual: S significa o deslocamento final, S_0 o deslocamento inicial, V a velocidade atual, V_0 a velocidade anterior e t o tempo decorrido desde a amostragem da última velocidade até a velocidade atual.

Uma vez que a velocidade é amostrada a cada 500 milissegundos, então t foi definido como esse tempo, logo o cálculo de distância percorrida deve ser realizado também a cada 500 milissegundos.

Dessa forma, nota-se que é considerada a aceleração do veículo, então independente se a velocidade é constante ou não, o cálculo final retorna a distância

desde a última amostragem da velocidade até a amostragem atual que, se somada a distância percorrida anteriormente, retorna um valor válido de distância total percorrida que é exibida na tela em quilômetros com precisão de 100 metros.

3.5.9 VELOCIDADE MÉDIA

Uma vez que é conhecido o tempo de funcionamento do motor, então a velocidade média do veículo também pode ser calculada ainda segundo Halliday, Resnick e Walker (2016) através da Equação (13),

$$V_m = \frac{\Delta S}{t} \tag{13}$$

na qual V_m corresponde à velocidade média calculada, ΔS à distância total percorrida e t ao tempo pelo qual o motor permaneceu funcionando.

Uma vez que a distância total percorrida é uma constante já calculada anteriormente e que o tempo é fornecido pelo barramento do veículo apenas de segundo em segundo, então este cálculo foi realizado apenas a cada 1 segundo e então disponibilizado em quilômetros por hora para o usuário do dispositivo.

3.5.10 TENSÃO DA BATERIA DO VEÍCULO

A requisição para a obtenção da tensão da bateria do veículo é realizada através do PID $46 pelo serviço $01. Por se tratar de uma grandeza pouco dinâmica, a aquisição da tensão da bateria é realizada a cada 2 segundos, visto que esta não é de fundamental importância e não varia muito conforme o tempo de acordo com os experimentos práticos realizados.

São retornados dois *bytes* de dados que, se aplicados à Equação (14), torna-se possível a leitura da tensão proveniente da bateria a todos os módulos eletrônicos do veículo, em volts.

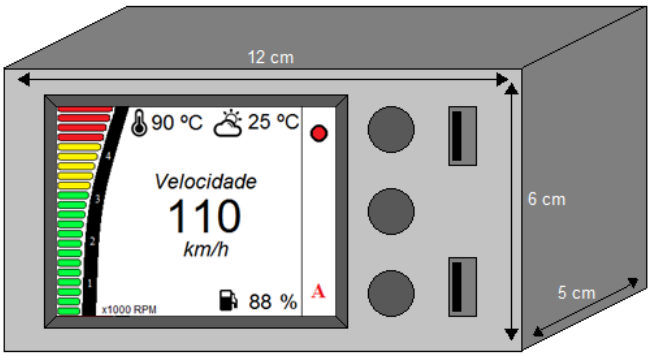
$$V = \frac{256A + B}{1000} \tag{14}$$

3.6 INTERFACE HOMEM MÁQUINA (IHM)

Foi construída uma interface de interação com o usuário baseada em três botões de configuração e uma tela TFT LCD conectada ao Arduino MEGA escravo. Através dessa interface, foi possível mostrar ao usuário todos os dados obtidos do carro e outros calculados utilizando esses mesmos dados.

O protótipo foi acondicionado em uma caixa em MDF de 12cm de largura, 6cm de altura e 5cm de profundidade, suficiente para acomodar uma tela LCD frontal e 3 botões laterais, além de 2 entradas auxiliares USB para carregar dispositivos móveis no painel, que serão implementadas em um trabalho futuro, conforme ilustrado na Figura 55.

Figura 55: Projeto de embalagem para acondicionamento do protótipo.

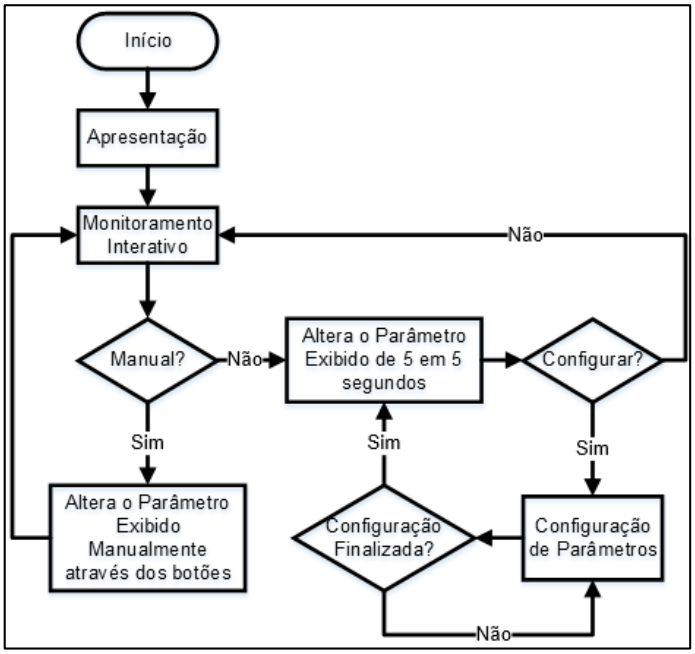


Fonte: Autoria Própria

Toda a interface foi programada utilizando a IDE do Arduino e as bibliotecas disponibilizadas na internet. Dessa forma, a biblioteca utilizada para a programação foi a *Adafruit GFX*, compatível com a tela TFT LCD utilizada e obtida no próprio endereço eletrônico do fabricante (ADAFRUIT, 2016), bem como os manuais obtidos também do mesmo local.

Foram programadas todas as telas de maneira isolada, porém buscou-se tornar a navegação por parte do usuário intuitiva e de fácil acesso. É importante que se mostre todo o funcionamento gráfico do protótipo, desde o fluxograma de funcionamento, apresentado pela Figura 56, até as funções específicas detalhadas, mostradas nos tópicos a seguir.

Figura 56: Fluxograma de funcionamento da IHM do protótipo.



Fonte: Autoria Própria

3.6.1 TELA DE APRESENTAÇÃO

Esta tela apresenta algumas informações úteis quanto a autoria do projeto e orientação, bem como o nome do equipamento e é exibida apenas uma vez, logo na inicialização do protótipo. Tem um formato semelhante ao apresentado na Figura 57 e nenhum tipo de interação é possível durante a exibição dessa tela que ocorre por 5 segundos.

Figura 57: Tela de Apresentação.

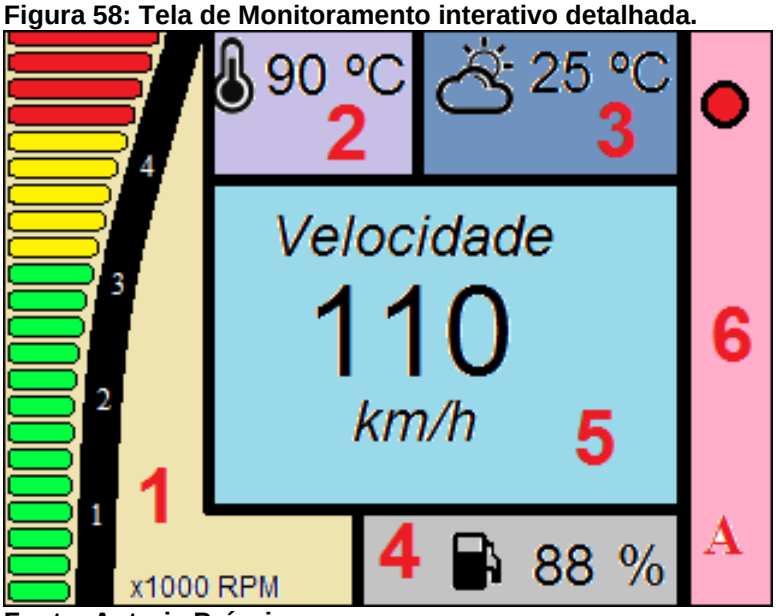
Fonte: Autoria Própria

3.6.2 MONITORAMENTO INTERATIVO

Esta é a principal tela do sistema, pois nela poderão ser visualizados todos os parâmetros lidos do veículo. Dessa forma, optou-se por uma interface com números e letras grandes para facilitar a exibição e a leitura pelo usuário.

A tela de Monitoramento Interativo é acessada automaticamente após a inicialização do protótipo e pode ser acessada também após as configurações realizadas por telas detalhadas posteriormente. Uma vez que é a tela principal, optou-se por encaixar o maior número possível de informações, conforme mostra a Figura 58, tais como a rotação do motor (1), a temperatura do líquido de arrefecimento (2), a temperatura ambiente (3), o percentual de combustível disponível no tanque do veículo (4), um parâmetro principal configurado pelo usuário (5) e os botões para a interação (6).

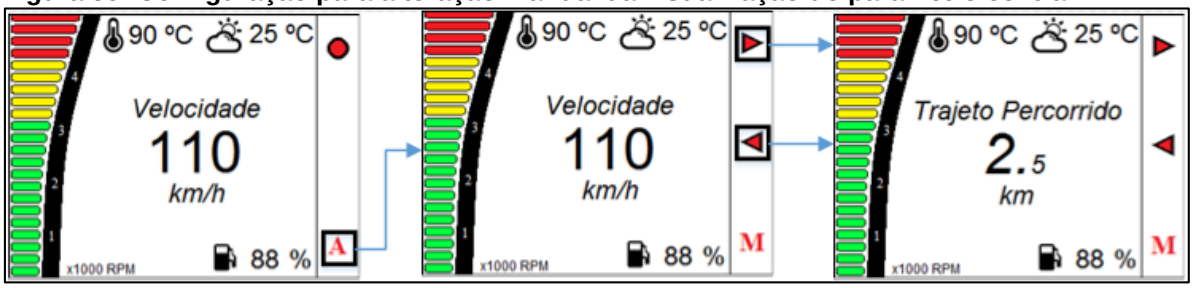
Figura 58: Tela de Monitoramento interativo detalhada.



Fonte: Autoria Própria.

A partir desta tela é possível alterar se a exibição do parâmetro central é modificada manualmente ou automaticamente a cada 5 segundos, sendo que os demais parâmetros são fixos e inalteráveis. Esta configuração é realizada pressionando o botão correspondente a “A”, o que habilita as setas para a troca manual dos parâmetros, conforme mostra a Figura 59.

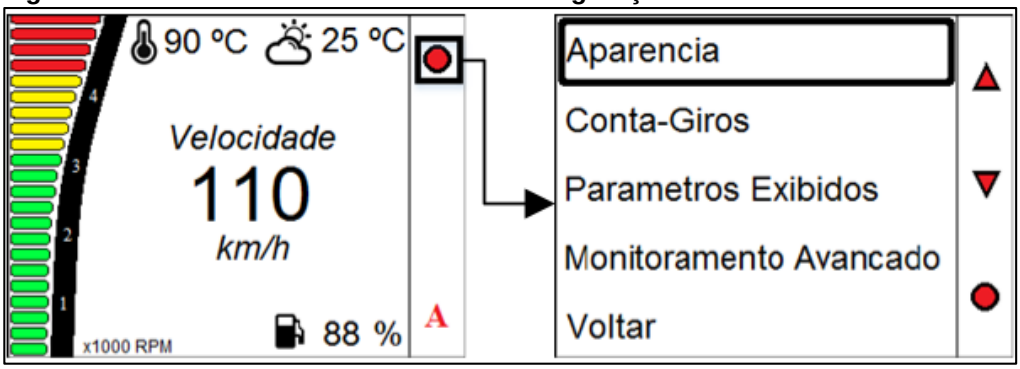
Figura 59: Configuração para alteração manual da visualização do parâmetro central.



Fonte: Autoria Própria

Quando no modo automático, é possível acessar as configurações do dispositivo utilizando o botão ●. Uma vez pressionado, é concedido o acesso as configurações de exibição do dispositivo conforme mostra a Figura 60.

Figura 60: Modo de acesso ao menu de configurações.



Fonte: Autoria Própria.

3.6.3 CONFIGURAÇÃO DE PARÂMETROS

A tela Configuração de Parâmetros é um menu com variadas opções para a personalização do dispositivo. É possível alterar as características de aparência geral do conta-giros, quais parâmetros serão exibidos manualmente ou automaticamente no campo voltado ao parâmetro principal na tela de monitoramento interativo, realizar um monitoramento avançado para desenvolvimento e então retornar a tela anterior.

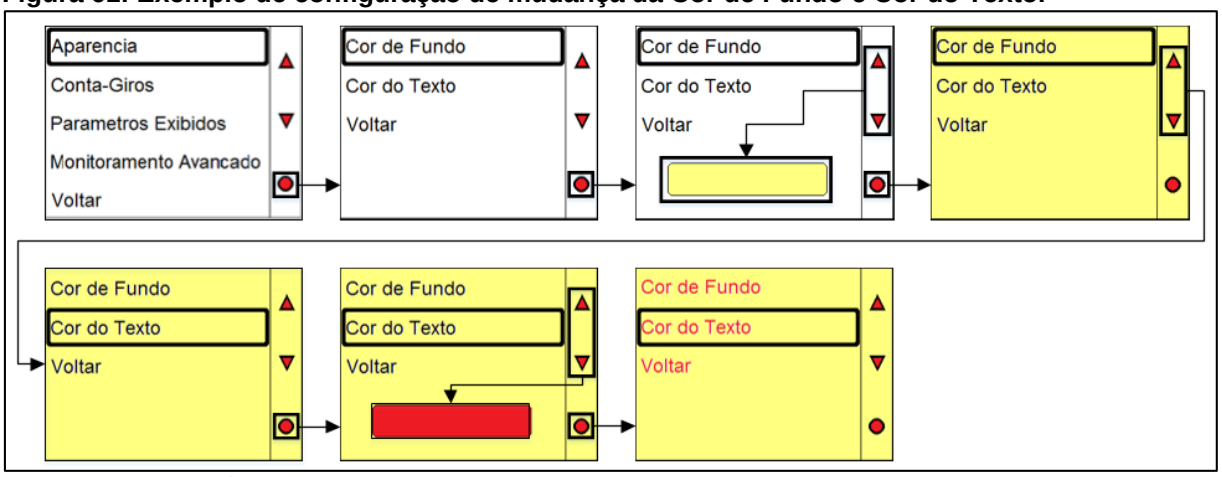
3.6.3.1 CONFIGURAÇÕES DE APARÊNCIA

Nas configurações de aparência é possível alterar tanto a cor de fundo do painel de exibição do dispositivo quanto a cor de todos os textos. Uma vez que há

várias cores para personalizar ambos os elementos, é possível que o usuário escolha as cores que mais combinam com seu veículo ou então que mais o agradam.

São disponibilizadas ao todo 9 cores voltadas a personalização do fundo de tela e mais 4 para o texto, o que gera ao todo 36 combinações diferentes. Essa configuração é acessada através da tela Configuração de Parâmetros, seguida por Aparência e então dos menus Cor do Texto e Cor de fundo, conforme exemplifica a Figura 61.

Figura 61: Exemplo de configuração de mudança da Cor de Fundo e Cor do Texto.



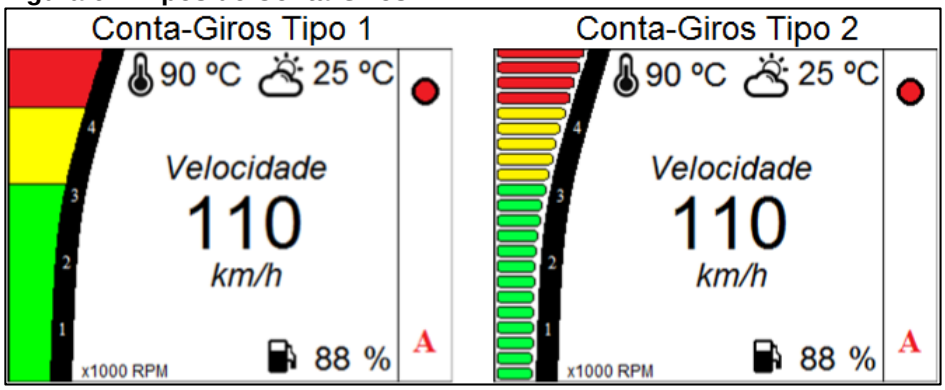
Fonte: Autoria Própria

Todas as configurações realizadas no menu Aparência são tomadas por padrão em toda a interface do dispositivo. Uma vez alteradas quaisquer uma das cores, então será mudado o fundo de todas as telas, assim como a cor do texto, incluindo as cores dos elementos da tela de Monitoramento Interativo.

3.6.3.2 CONFIGURAÇÕES DO CONTA-GIROS

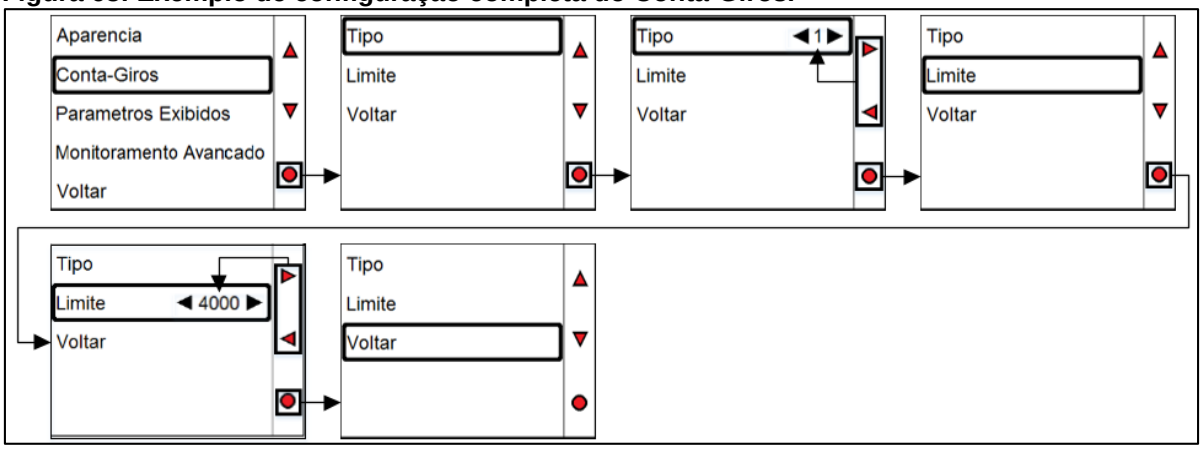
Foram programados dois tipos de conta-giros para o usuário escolher, o tipo 1 e o tipo 2 mostrados na Figura 62. Na mesma tela, é possível definir, além do tipo, a escala máxima compatível com o veículo, que pode variar entre 4000 e 9000 giros, com passos de 1000, para que esta se adapte da melhor forma à leitura da rotação do motor, conforme exemplificado na Figura 63.

Figura 62: Tipos de Conta-Giros.



Fonte: Autoria Própria.

Figura 63: Exemplo de configuração completa do Conta-Giros.



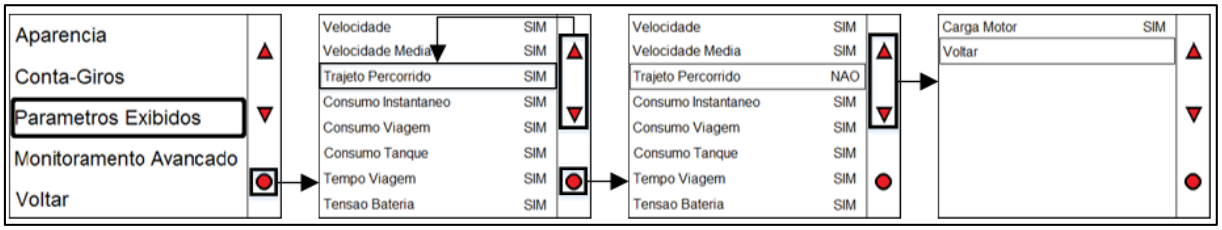
Fonte: Autoria Própria.

3.6.3.3 CONFIGURAÇÕES DOS PARÂMETROS EXIBIDOS NA TELA PRINCIPAL

Há uma série de parâmetros que foram definidos para serem exibidos na tela de Monitoramento Interativo e que podem ser configurados nesse Menu conforme as preferências do usuário. Esses parâmetros são disponibilizados ao usuário através do menu Parâmetros Exibidos, localizado também no menu de configurações.

A Figura 64 mostra como são dispostos todos os parâmetros e como estes podem ser modificados através dos botões de navegação. É importante destacar que nesta tela são configurados apenas os parâmetros principais que serão exibidos na tela de Monitoramento Interativo automática ou manualmente, conforme a escolha do usuário.

Figura 64: Configuração dos parâmetros exibidos automática ou manualmente na tela de monitoramento interativo.



Fonte: Autoria Própria

3.6.3.4 MONITORAMENTO AVANÇADO

Esta tela foi desenvolvida com o objetivo de auxiliar em trabalhos futuros utilizando o dispositivo, visto que a gama de aplicações do computador de bordo pode ir muito além de simplesmente a exibição de parâmetros. Esta tela foi usada para a leitura primária de cada um dos dados, para o teste de requisição dos parâmetros via diagnóstico, para a tradução e cálculo das grandezas citadas nos tópicos anteriores e para teste de configurações temporárias, logo esta foi necessária para testar os resultados para que então fosse possível adequá-los às outras telas.

Uma vez que há um barramento CAN real acoplado ao dispositivo, então é possível a realização tanto de leitura quanto de envio de informações para qualquer dispositivo CAN compatível, seja um veículo qualquer ou um sensor. Uma vez que a aplicação do protótipo pode estender-se, esta tela foi mantida para tornar possível a execução de atividades futuras, nas quais será utilizada a mesma estrutura do trabalho e, portanto, será necessária novamente a análise de dados temporários.

4 RESULTADOS E DISCUSSÕES

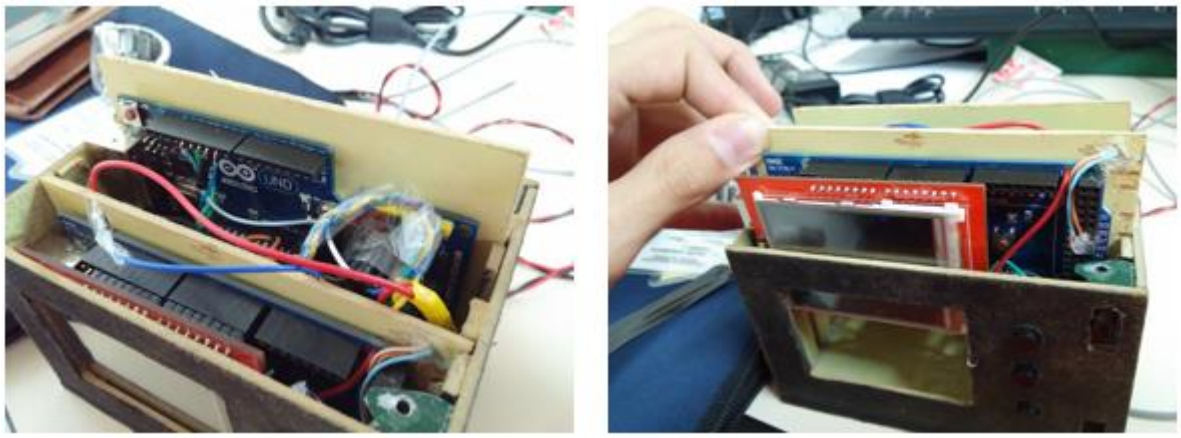
Neste tópico é demonstrado o funcionamento do protótipo já construído como um todo, assim como todas as funções do aparelho. O dispositivo resultou no aparelho mostrado pela Figura 66, onde todas as etapas apresentadas no Capítulo 3 foram implementadas.

Figura 65: Dispositivo real construído.

Fonte: Autoria Própria

Internamente, foi criada uma estrutura do tipo gavetas para armazenar todos os componentes. Uma vez que foram utilizados dois Arduinos, um modulo CAN e uma tela, foi necessário organizar tudo dentro da caixa, logo isso foi feito através de placas de madeira onde os componentes foram parafusados. Na Figura 66 é possível ver a parte interna do dispositivo.

Figura 66: Estrutura interna do dispositivo.



Fonte: Autoria Própria.

Percebe-se que o dispositivo teve um bom funcionamento e realiza todas as atividades às quais se propõe, a começar pela alimentação através da bateria do veículo como mostra a Figura 67.

Figura 67: Dispositivo conectado ao veículo.

Fonte: Autoria Própria

Para comprovar a comunicação do veículo com o dispositivo, foi realizado um teste através da medição da rotação do motor. Dessa forma, foi possível visualizar tanto o valor mostrado no carro como o mostrado no dispositivo, o que valida a leitura dos dados provenientes da rede CAN e o sucesso na obtenção dos dados de diagnóstico, conforme mostra a Figura 68.

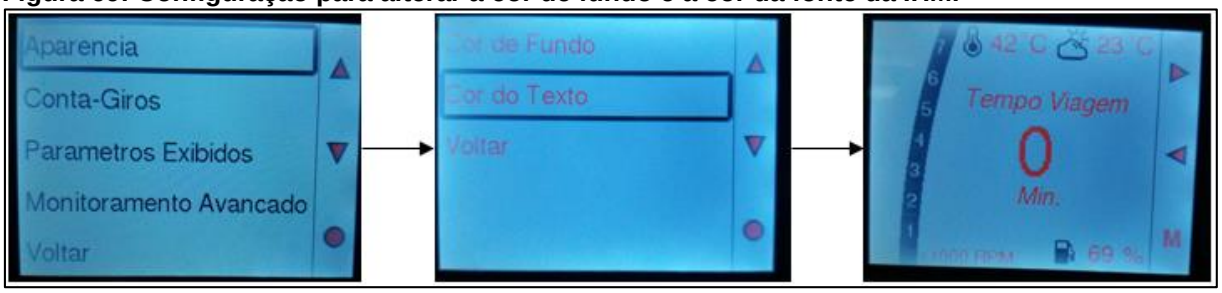
Figura 68: Validação da leitura através da comparação com o do veículo teste.

Fonte: Autoria Própria.

Ainda na Figura 68, nota-se que é possível validar também toda a comunicação através de SPI entre os microcontroladores utilizados no desenvolvimento, visto que não há nenhum problema nem na exibição e nem na leitura dos dados.

Já comprovado o sucesso na obtenção das informações do veículo, agora pode ser feita a personalização do dispositivo. Inicialmente, foi acessado o menu de configurações e então foi mudada a cor de fundo para cinza e a cor do texto para vermelho, conforme segue a Figura 69.

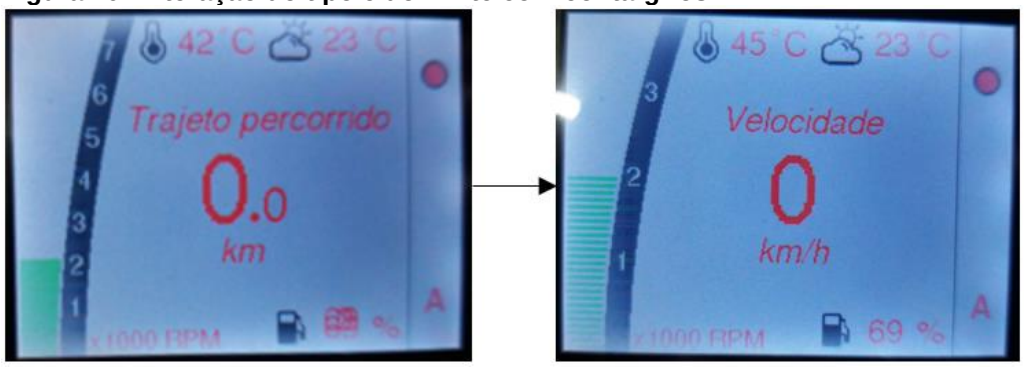
Figura 69: Configuração para alterar a cor de fundo e a cor da fonte da IHM.



Fonte: Autoria Própria.

Foram alterados também o tipo e o limite do conta-giros através do menu de personalização, como mostra a Figura 70. Visto que anteriormente o conta-giros estava com um limite de 8000 rotações por minuto, foi selecionado um limite de 4000 e também foi trocado o tipo de conta-giros. Uma vez que há leitura e a possibilidade de personalização, pode-se reafirmar que o dispositivo está funcionando adequadamente.

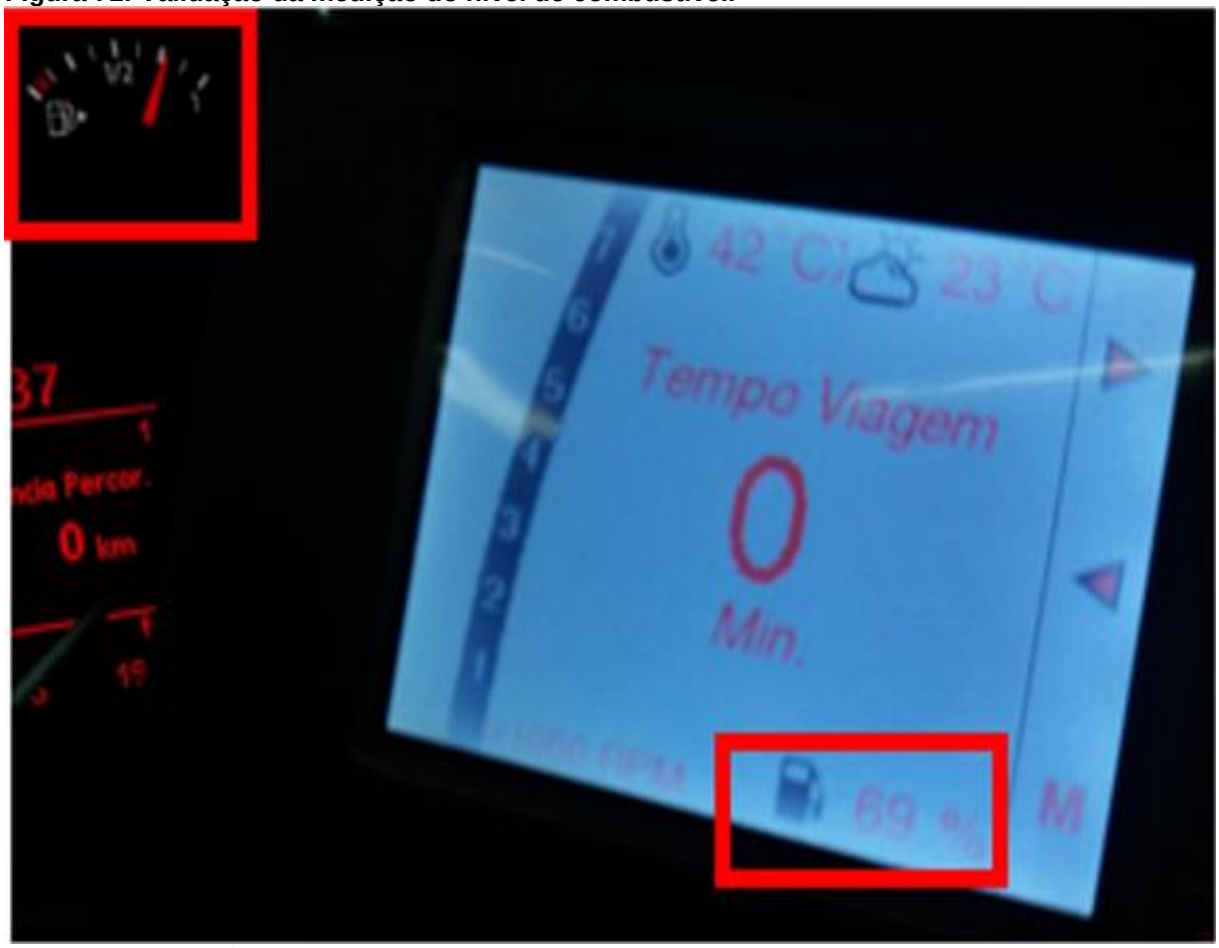
Figura 70: Alteração do tipo e do limite com conta-giros



Fonte: Autoria Própria.

A marcação do nível de combustível pode ser validada também junto à marcação mostrada no painel do veículo. Como pode ser visto na Figura 71, é apresentado um valor muito próximo do valor do veículo na tela, onde percebe-se que o pequeno erro ocorre por conta dos métodos de exibição: analógico no veículo e digital no protótipo.

Figura 71: Validação da medição do nível de combustível.



Fonte: Autoria Própria.

Já é possível confirmar que o dispositivo exibe as informações reais do barramento do veículo na tela. Dessa forma, é importante então exibir os valores calculados, tais como a velocidade média e a distância percorrida pelo veículo.

Foi feito um experimento de cinco minutos de duração onde foi rodado com o veículo teste aproximadamente 1.7 quilômetros segundo o dispositivo, conforme mostra a Figura 72. Vale destacar que o valor do tempo de acionamento do motor foi obtido diretamente da CAN de diagnóstico do veículo.

Figura 72: Distância percorrida calculada e tempo de viagem.

Fonte: Autoria Própria.

De posse tanto da distância percorrida quanto do tempo de viagem, foi possível então, por parte do dispositivo, calcular a velocidade média, apresentada na Figura 73.

Figura 73: Velocidade média do trajeto percorrido com o veículo teste.

Fonte: Autoria Própria.

Uma vez que foi possível obter a resposta às requisições feitas pelo dispositivo, logo os dados foram lidos corretamente. Como esses dados também foram exibidos na tela, então conclui-se que a comunicação entre todos os microcontroladores internos ao dispositivo também foi estabelecida e, através disso, é possível então validar tanto o dispositivo quanto a arquitetura proposta para o funcionamento deste.

5 CONCLUSÃO

O início deste trabalho consistiu de um levantamento bibliográfico que se mostrou importante para consolidar a ideia do protótipo desenvolvido, uma vez que foi mostrado que existem várias tecnologias voltadas ao diagnóstico automotivo. Através disso, notou-se que todas dependiam de um fator externo para o correto funcionamento, como o celular ou até mesmo um sensoriamento a parte, o que motivou o desenvolvimento do Computador de Bordo apresentado neste trabalho.

Uma vez definida parte de toda a gama de dispositivos parecidos encontrados em artigos e patentes, concluiu-se que a proposta apresentada é interessante, visto que o protótipo desenvolvido neste trabalho não depende nem de sensoriamento externo, nem de celular para funcionar corretamente.

Em seguida, foi dado início a todo o levantamento teórico para tornar possível o desenvolvimento do protótipo. Foram levantados conceitos sobre a rede CAN automotiva para tornar possível a comunicação de um veículo com um equipamento externo, porém, foi necessário que se fizesse um estudo sobre um importante protocolo muito utilizado na comunicação entre microcontroladores, o SPI, que tornou possível a utilização de um dispositivo dedicado a trabalho com redes CAN e a descentralização do processamento, dividindo-o entre dois Arduinos.

Determinados todos os protocolos de comunicação, foi possível dar início ao estudo das normas ISO 15765 (2006) e SAE J1979 (2006), que regulamentam toda a estrutura necessária à realização de diagnóstico automotivo. Essas normas foram importantes pois permitiram a comunicação entre o Computador de Bordo e o veículo, disponibilizando todas informações necessárias para a conexão e tradução dos dados provenientes do barramento CAN automotivo.

Já de posse de todos os requisitos para a realização da comunicação entre o veículo e o protótipo, foi definida uma estrutura base levada como referência para a construção do dispositivo físico. Todos os dispositivos internos ao protótipo foram interconectados via SPI seguindo a estrutura predefinida, que se mostrou eficiente e válida visto que foi possível validar essa comunicação.

Toda a programação do protótipo foi executada com base nos conceitos apresentados no referencial teórico deste trabalho. Foram apresentados alguns resultados reais da comunicação via SPI, como a escrita e a leitura de um dos registradores presentes no microcontrolador CAN utilizado, que foi validada através

da análise do barramento feita por um analisador lógico, ferramenta útil na leitura de dados digitais.

Já com o dispositivo montado por completo, foi programada toda a interface CAN para operar com algumas velocidades utilizadas comumente em barramentos de diagnóstico automotivo. Via SPI, foram executadas rotinas de programação responsáveis pela inicialização do microcontrolador CAN dedicado, bem como foram definidos procedimentos padrões para envio e recebimento de mensagens, necessárias à aquisição de dados do veículo.

Uma vez obtidos os dados, foi necessário processá-los e traduzi-los, trabalho destinado a um Arduino MEGA® escravo conectado via SPI a um Arduino UNO® mestre. Notou-se que a operação de um Arduino como escravo não é permitida pelas bibliotecas de desenvolvimento utilizadas, portanto foi necessário programar esta função diretamente nos registradores do microcontrolador central da placa de desenvolvimento.

Estabelecida a comunicação entre Arduinos, foi possível enviar os dados obtidos da rede CAN do Arduino UNO® para o Arduino MEGA® via SPI, no qual foram decodificados segundo as normas, tornando-os prontos para a exibição ao usuário.

Com todos os dados já prontos para a exibição, deu-se início à programação da interface homem-máquina, onde foram prezadas tanto a facilidade de uso quanto a possibilidade de personalização. Por ser programada em uma tela de cristal líquido colorida, foi possível desenvolver funções para a troca da cor de fundo e do texto exibido no protótipo, bem como foi possível inserir vários parâmetros de configuração e de monitoramento, todos detalhados e distribuídos de maneira amigável por todas as telas.

O custo final do protótipo ficou em aproximadamente R$300,00, que pode ser reduzido com o emprego de outros microcontroladores desprovidos de placas de desenvolvimento.

Após a programação completa do Computador de Bordo e sua acomodação dentro de uma carcaça de madeira, este foi conectado ao veículo e validado através de comparações com os parâmetros exibidos no painel de instrumentos do carro. Foram comparados o nível de combustível e a rotação do veículo e, uma vez confirmadas essas informações, puderam obtidas outras como tensão da bateria, velocidade e temperatura ambiente de maneira confiável, fator que levou a concluir que os objetivos propostos inicialmente pelo trabalho foram alcançados.

REFERÊNCIAS

AMARASINGHE, Malintha et al. Cloud-based driver monitoring and vehicle diagnostic with OBD2 telematics. **2015 Fifteenth International Conference On Advances In Ict For Emerging Regions (icter),** [s.l.], p.243-249, ago. 2015. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/icter.2015.7377695.

AMBATA, Leonard U. et al. Distance monitoring vision system for automobiles using image processing. **2015 International Conference On Humanoid, Nanotechnology, Information Technology,communication And Control, Environment And Management (hnicem),** [s.l.], p.1-6, dez. 2015. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/hnicem.2015.7393164.

ANAND N, G. JOSEPH, S. S. OOMMEN and R. DHANABAL. **Design and implementation of a high speed Serial Peripheral Interface**. *Advances in Electrical Engineering (ICAEE), 2014 International Conference on*, Vellore, 2014, pp. 1-3.

ARDUINO. **Compare Board Specs.** 2016. Disponível em: <https://www.arduino.cc/en/Products/Compare>. Acesso em: 05 nov. 2016.

ARDUINO. **Página Principal.** Disponível em: <https://www.arduino.cc/>. Acesso em: 05 nov. 2016.

ATMEL CORPORATION (Estados Unidos). **ATMEGA2560:** Datasheet Complete. San Jose: Atmel, 2016. 444 p. Disponível em: <http://www.atmel.com/Images/Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561_datasheet.pdf>. Acesso em: 05 nov. 2016.

ATMEL CORPORATION (Estados Unidos). **ATMEGA328p:** Datasheet Complete. San Jose: Atmel, 2016. 444 p. Disponível em: <http://www.atmel.com/Images/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_datasheet.pdf>. Acesso em: 05 nov. 2016.

ATMEL CORPORATION (San Jose). Emil Lambrache; Benjamin F. Froemming. **Serial Peripheral Interface (Spi) Apparatus With Write Buffer For Improving Data Throughput**. EUA nº 7082481, 25 jun. 2006, 26 maio 2005.

BARBOSA, Luiz Roberto Guimarães. **Rede CAN.** 2003. Disponível em: <http://www.cpdee.ufmg.br/~elt/docs/DSP/Resumo_CAN.pdf>. Acesso em: 31 ago. 2016.

BRASIL. Lei nº 13281, de 04 de maio de 2016. **Lei Nº 13.281, de 4 de Maio de 2016.**. Brasilia, DF.

BRASIL. Ministério do Meio Ambiente, Conselho Nacional de Meio Ambiente, CONAMA. **Resolução CONAMA n°324, de 14 de dezembro de 2004**. Resoluções, 2004. Disponível em <http://www.mma.gov.br/port/conama/legiabre.cfm?codlegi= 456> Acesso em 22 ago. 2016.

BURJE, Pallavi R.; KARANDE, Kailash J.; JAGADALE, Amol B.. Embedded On-Board Diagnostics system using CAN protocol. **2014 International Conference On Communication And Signal Processing,** [s.l.], p.734-737, abr. 2014. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/iccsp.2014.6949940.

BUSMASTER. 2016. Disponível em: <https://rbei-etas.github.io/busmaster/>. Acesso em: 07 set. 2016.

CAN IN AUTOMATION. **CAN data link layers in some detail.** 2016. Disponível em: <http://www.can-cia.org/can-knowledge/can/can-data-link-layers/>. Acesso em: 07 set. 2016.

CASTRUCI, P. L., MORAES C.C. **Engenharia de automação industrial**, 2ª Ed. Rio de Janeiro: LTC, 2007

CEUCA, E. et al. Embedded system for remote monitoring of OBD bus. **Proceedings Of The 36th International Spring Seminar On Electronics Technology,** [s.l.], p.305-308, maio 2013. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/isse.2013.6648262.

CORRIGAN, Steve. **Introduction to the Controller Area Network.** Dallas: Texas Instruments, 2008. 15 p. Disponível em: <http://www.ti.com/lit/an/sloa101a/sloa101a.pdf>. Acesso em: 07 set. 2016.

CYPRESS PERFORM (Org.). **Serial Peripheral Interface (SPI) Slave.** 2010. Disponível em: <http://www.cypress.com/file/132126/download>. Acesso em: 27 set. 2016.

DI NATALE, Marco. **Understanding and Using The Controller Area Network.** 2008. Disponível em: <http://inst.cs.berkeley.edu/~ee249/fa08/Lectures/handout_canbus2.pdf>. Acesso em: 01 set. 2016.

ENRIQUEZ, D.j. et al. CANOPNR: CAN-OBD programmable-expandable network-enabled reader for real-time tracking of slippery road conditions using vehicular parameters. **2012 15th International Ieee Conference On Intelligent Transportation Systems,**[s.l.], p.260-264, set. 2012. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/itsc.2012.6338905.

FARSI, M.; RATCLIFF, K.; BARBOSA, Manuel. An Overview of Controller Area Network. **Computing & Control Engineering Journal.** Stevenage, p. 113-120. jun. 1999.

FOROUZAN, Behrouz A.. **Comunicação de Dados e Redes de Computadores.** 3. ed. Porto Alegre: Bookman, 2006.

FRANCISCO POSADA ([s.i.]). **GLOBAL OVERVIEW OF ON-BOARD DIAGNOSTIC (OBD) SYSTEMS FOR HEAVY-DUTY VEHICLES.** 2015. Disponível em: <http://www.theicct.org/sites/default/files/publications/ICCT_Overview_OBD-HDVs_20150209.pdf>. Acesso em: 07 nov. 2015.

GFX, Adafruit. **Adafruit GFX Graphics Library.** Disponível em: <https://learn.adafruit.com/adafruit-gfx-graphics-library/overview>. Acesso em: 04 nov. 2016.

HALLIDAY, David; RESNICK, Robert; WALKER, Jearl. **Fundamentos de Física:** Mecânica. 10. ed. [s.i]: Ltc, 2016.

HARJUNKOSKI, Iiro; NYSTRÖM, Rasmus; HORCH, Alexander. Integration of scheduling and control—Theory or practice? **Computers & Chemical Engineering,** [s.l.], v. 33, n. 12, p.1909-1918, dez. 2009. Elsevier BV. http://dx.doi.org/10.1016/j.compchemeng.2009.06.016.

INTERNATIONAL STANDARDIZATION ORGANIZATION. **ISO 11898**: Road vehicles — Controller area network (CAN). Geneva: Iso, 2003

INTERNATIONAL STANDARDIZATION ORGANIZATION. **ISO 15031**: Road vehicles — Communication between vehicle and external equipment for emissions-related diagnostics. Geneva: Iso, 2006.

INTERNATIONAL STANDARDIZATION ORGANIZATION. **ISO 15765**: Road vehicles — Diagnostics on Controller Area Networks (CAN). Geneva: Iso, 2006.

JHOU, Jheng-syu et al. The Implementation of OBD-II Vehicle Diagnosis System Integrated with Cloud Computation Technology. **2013 Second International Conference On Robot, Vision And Signal Processing,** [s.l.], p.9-12, dez. 2013. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/rvsp.2013.55.

JUNIOR, Antônio Tadeu Lyrio de Almeida; RIBEIRO, Luiz Sérgio Pinto. **COMPUTADOR DE BORDO MICROCONTROLADO PARA MOTOCICLETAS E SIMILARES**. BR nº PI 0700973-9, 25 jan. 2007, 16 set. 2008.

KHANAPURI, Akhila V et al. On road: A car assistant application. **2015 International Conference On Technologies For Sustainable Development (ictsd),** [s.l.], p.1-7, fev. 2015. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/ictsd.2015.7095903.

LAWRENZ, Wolfhard. **CAN System Engineering:** From Theory to Practical Applications. 2. ed. Wolfenbüttel: Springer, 2013. 375 p.

LIMA, Anderson Borges de; SILVA, Anderson. **SISTEMA DE COMPUTADOR DE BORDO PARA CONDOMÍNIOS**. BR nº PI0701054-0, 15 maio 2007, 30 dez. 2008.

LIMA, Marcelo Campi. **SISTEMA DE COMPUTADOR DE BORDO PARA CONTROLE DE MANUTENÇÃO DE VEÍCULOS AUTOMOTORES**. BR nº PI0700713-2, 01 fev. 2007, 16 set. 2016. .

LUGLI, Alexandre Barantella; SANTOS, Max Mauro Dias. **Sistemas FIELDBUS para automação industrial:** DeviceNet, CANopen, SDS e Ethernet. Tatuapé: Erica, 2009.

MARINKOVIC, Stevan J. et al. Nano-Power Wireless Wake-Up Receiver With Serial Peripheral Interface. **Ieee J. Select. Areas Commun.,** [s.l.], v. 29, n. 8, p.1641-1647, set. 2011. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/jsac.2011.110913.

MARTINS, Evandro Cesar. **Computador de Bordo para Veículos Automotores**. BR nº PI 0003395-2, 26 jul. 200, 26 fev. 2002.

MATHWORKS. **Vehicle Network Toolbox:** Communicate with in-vehicle networks using CAN, J1939, and XCP protocols. 2016. Disponível em: <http://www.mathworks.com/products/vehicle-network/>. Acesso em: 07 set. 2016.

MERSCH, Henning; SCHLUTTER, Markus; EPPLE, Ulrich. Classifying services for the automation environment. **2010 Ieee 15th Conference On Emerging Technologies & Factory Automation (etfa 2010),** [s.l.], p.1-7, set. 2010. Institute of Electrical & Electronics Engineers (IEEE). http://dx.doi.org/10.1109/etfa.2010.5641170.

MICROCHIP TECHNOLOGIES (Org.). **MCP2515:** Stand-Alone CAN Controller With SPI Interface. 2005. Disponível em: <http://ww1.microchip.com/downloads/en/DeviceDoc/21801d.pdf>. Acesso em: 27 set. 2016.

MI-JINKIM; JANG, Jong-wook; YU, Yun-sik. A Study on In-Vehicle Diagnosis System using OBD-Ⅱ with Navigation. **International Journal Of Computer Science And Network Security,** Busan, v. 10, n. 9, p.136-140, set. 2010.

NIAZI, M Awais Khan et al. Development of an On-Board Diagnostic (OBD) kit for troubleshooting of compliant vehicles. **2013 Ieee 9th International Conference On Emerging Technologies (icet),** [s.l.], p.734-737, dez. 2013. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/icet.2013.6743551.

NXP PHILIPS SEMICONDUCTORS (Org.). **TJA1050:** High Speed CAN Transceiver. 2003. Disponível em: <http://www.nxp.com/documents/data_sheet/TJA1050.pdf>. Acesso em: 29 set. 2016.

PARK, Sang Hyun; LEE, Sang Yub. Development of On-board Diagnosis via CAN for a HVI (Human Vehicle Interface) Technology. **2012 Ieee 10th International Symposium On Parallel And Distributed Processing With Applications,** [s.l.], p.839-840, jul. 2012. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/ispa.2012.125.

PINTO, Benedito Geraldo Míglio; AMARAL, Paulo Faria Santos; FILHO, Antônio Bento. **Computador de Bordo para Locomotivas**. BR nº MU 7001432, 25 jul. 90, 11 fev. 1992.

QUENEHEN, Romulo. **Rastreador, navegador, computador e computador de bordo que emite alertas de segurança e monitora a estabilidade do veículo**. BR nº PI1100527-0, 15 fev. 2011, 28 maio 2013

RICHARDS, Pat (Arizona). Microchip. **A CAN Physical Layer Discussion.** Chandler: Microchip, 2002. 11 p. Disponível em: <http://ww1.microchip.com/downloads/en/AppNotes/00228a.pdf>. Acesso em: 02 set. 2016.

SD ASSOCIATION (Org.). **SD Specification:** Part E1: SDIO Simplified Specification. 2011. Disponível em: <https://www.sdcard.org/downloads/pls/>. Acesso em: 27 set. 2016.

SNAPON (Estados Unidos). **Global OBD Vehicle Communication Software Manual.** 2013. Disponível em: <https://www1.snapon.com/Files/Diagnostics/UserManuals/GlobalOBDVehicleCommunicationSoftwareManual_EAZ0025B43.pdf>. Acesso em: 03 nov. 2016.

SOCIETY OF AUTOMOTIVE AND AEROSPACIAL ENGINEERS. **SAE J1979**: E/E Diagnostic Test Modes. Detroit: Sae, 2006.

TAHAT, Ashraf et al. Android-based universal vehicle diagnostic and tracking system. **2012 Ieee 16th International Symposium On Consumer Electronics,** [s.l.], p.137-143, jun. 2012. Institute of Electrical and Electronics Engineers (IEEE). http://dx.doi.org/10.1109/isce.2012.6305105.

TANENBAUM, Andrew S.. **Redes de Computadores.** 4. ed. Amsterdam: Campus, 2010. 632 p.

TESSARO, Iron Lukas. **DESENVOLVIMENTO DE UM COMPUTADOR DE BORDO PARA AUTOMÓVEIS.** 2013. 84 f. TCC (Graduação) - Curso de Curso Superior de Tecnologia em Mecatrônica Industrial, Departamentos Acadêmicos de Eletrônica e Mecânica, Universidade Tecnológica Federal do Paraná, Curitiba, 2013.

VECTOR INFORMATIK (Baden-württemberg). **CANdb++ User's Manual.** Sttutgart: Vector Informatik Gmbh, 2010. 40 p. Disponível em: <http://vector.com/portal/medien/cmc/manuals/CANdb++_Manual_EN.pdf>. Acesso em: 07 set. 2016.