CENTRO PAULA SOUZA

FACULDADE DE TECNOLOGIA

FATEC SANTO ANDRÉ

Tecnologia em Eletrônica Automotiva

Gustavo Espelho Machado

Weslei Alves Silva

ENSAIO EM PROTOCOLO CAN

Santo André

2018

CENTRO PAULA SOUZA
FACULDADE DE TECNOLOGIA
FATEC SANTO ANDRÉ
Tecnologia em Eletrônica Automotiva

Gustavo Espelho Machado
Weslei Alves Silva

ENSAIO EM PROTOCOLO CAN

Monografia apresentada ao Curso de Tecnologia em Eletrônica Automotiva da FATEC Santo André, como requisito parcial para conclusão do curso em Tecnologia em Eletrônica Automotiva.

Orientador: Prof. Paulo Alexandre Pizará Hayashida.
Coorientador:Prof Dr. Armando Antonio Maria Lagana

Santo André
2018

FICHA CATALOGRÁFICA

M149e

Machado, Gustavo Espelho

Ensaio em protocolo CAN / Gustavo Espelho Machado, Weslei Alves Silva. - Santo André, 2018. – 69f: il.

Trabalho de Conclusão de Curso – FATEC Santo André.

Curso de Tecnologia em Eletrônica Automotiva, 2018.

Orientador: Prof. Paulo Alexandre Pizará Hayashida

1. Eletrônica. 2. Veículos. 3. Linguagem. 4. Protocolo CAN. 5. Rede. 6. Software. 7. Comunicação. 8. Informática. 9. Tecnologia. I. Silva, Alexandre Carlos da. II. Estudo de suspensão veicular.

629.2

LISTA DE PRESENÇA

Santo André, 10 de Julho de 2018

LISTA DE PRESENÇA REFERENTE À APRESENTAÇÃO DO TRABALHO DE CONCLUSÃO DE CURSO COM O TEMA: **"ENSAIOS EM PROTOCOLO CAN"** DOS ALUNOS DO 6º SEMESTRE DESTA U.E.

BANCA

PRESIDENTE:
PROF. ORLANDO DE SALVO JUNIOR_____

MEMBROS:
PROF. KLEBER NOGUEIRA HODEL_____

SR. PAULO ALEXANDRE PIZARA HAYASHIDA_____

ALUNOS :

GUSTAVO ESPELHO MACHADO_____

WESLEI ALVES SILVA_____

AGRADECIMENTOS

Agradeço imensamente a todos os nossos professores que nos deram incentivo, confiança e nos passaram o conhecimento necessário para a realização deste trabalho, além é claro, de nos auxiliarem durante todo tempo. Agradeço também a minha namorada Leticia, meus familiares, e meu parceiro Weslei que durante todo o curso me incentivaram e me deram todo o tipo de apoio, para conseguirmos concluir mais esta fase em minha vida.

Agradeço a Deus por todos os seus feitos, os quais percebo e aqueles que ignoro. Agradeço a minha esposa Késia pelo apoio extraordinário e paciência, aos mestres da Fatec Santo André, Prof. Paulo Alexandre Pizará Hayashida, Dr. Armando Antonio Maria Laganá, Dr. Kleber Hodel, Mestre Fernando Garup que nos apoiaram e incentivaram, o meu parceiro neste trabalho Gustavo, e meus amigos pelo incentivo, principalmente a Marcelo Kai.

“O que sabemos é uma gota, o que ignoramos é um oceano.”
Newton, Isaac

RESUMO

Após realizar alguns ensaios utilizando o barramento CAN (*Controller Area Network*) dos veículos *Fiat* Strada e *Volkswagen* Gol, foi possível entender melhor e observar a teoria do protocolo de comunicação CAN sendo aplicados e compreender parte da complexidade do *software* de comunicação e do tráfego de dados do barramento, bem como o conteúdo das mensagens do tráfego e os seus significados, tornando assim possível a construção de um *software* que é capaz de conectar-se ao barramento e colher as informações que estão em linguagem de máquina, converter e torna-las uteis para linguagem homem/máquina.

Palavras Chaves: CAN, Diagnose automotiva.

ABSTRACT

After performing some tests using the Controller Area Network (CAN) bus of the Fiat Strada and Volkswagen Gol vehicles, it was possible to better understand and observe the theory of the CAN communication protocol being applied and to understand part of the complexity of communication software and data traffic of the bus, as well as the content of the traffic messages and their meanings, thus making possible the construction of software that is able to connect to the bus and collect the information that are in machine language, convert and make them useful for man/machine language.

Keyword: CAN, *automotive diagntics*.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELA

LISTA DE SIGLAS, ACRÔNIMOS E ABREVIATURAS

ABS - Antlock Braking System

ACK - Acknowledge

AMP - Arbitration on Message Priority

BOB - Break Out Box

BRP - Baud Rate Prescaler

CAN - Controller Area Network

CD - Collision Detection

CONAMA - Conselho Nacional do Meio Ambiente

CRC - Checksum

CPU - Central Process Unit

CSMA - Carrier Sense Multiple Access

DLC - Data Length Code

GND - Graduated Neutral Density

ID - Identificador

IDE - Message Identifier

ISO - International Organization for Standardization

LCD - Liquid Crystal Display

LLC - Logic Link Control

MAC - Media Access Control

OBD - On-Board Diagnostic

OSI - Open System Interconnection

RPM - Rotação por Minuto

RTR - Requisição de Transmissão Remota

SAE - Society of Automotive Engineering

SRR - Substituto de Requisição Remota

SOF - Start of Frame

TPS - Throttle Position Sensor

TQ - Time Quantum

USB - Universal Serial Bus

SUMÁRIO

1 Introdução

A evolução dos veículos automotores trouxe novos desafios para a indústria, pois está se viu a lidar com várias pressões e novas mudanças no seu produto. Estas pressões se deram tanto pelos consumidores, por normas ambientais e governamentais, como também pela concorrência do mercado entre fabricantes de automóveis (Bosch, 1991). A evolução nos automóveis e os desafios à engenharia da indústria automotiva praticamente se deu de forma mais perceptível e na direção que em suma busca tratar este trabalho, com a necessidade de implementação de dispositivos elétricos e eletrônicos nos veículos sejam para atender a fatores legais ou econômicos.

Crescendo a eletroeletrônica no automóvel geração após geração de inovações e novas possibilidades, multiplicaram-se os dispositivos eletroeletrônicos demais itens de conforto e segurança nos automóveis tais como, vidros com acionamento elétrico, limpadores de para-brisa, sistemas de injeção eletrônica, freios inteligentes, assistência elétrica de direção, dentre outros (Hodel K. N., 2009). Estas novas possibilidades trouxeram com elas grandes desafios à indústria automotiva, pois se viu crescer os custos de produção, e, principalmente, o peso dos veículos, implicando no aumento de insumos de produção e exigindo motores mais potentes e maiores.

Além disso, normas ambientais, como a Resolução do (Conama, 1989), passaram a exigir menores emissões de poluentes, pressionando assim a indústria a reduzir custos e buscar soluções forçando a engenharia a desenvolver métodos e novas tecnologias e aplicações nos automóveis. Esse fato levou ao surgimento, por exemplo, de módulos controladores para gerenciamentos dos diversos dispositivos eletroeletrônicos dos automóveis tais como motor, câmbio, sistemas de freios inteligentes, ar condicionado e demais itens do veículo.

Estas tecnologias trouxeram enormes avanços na indústria, mas a engenharia se deparou com novos problemas, como o aumento de cabos e conexões, gerando dificuldade de manutenção destes sistemas. Uma vez que cada sistema do veículo, como controle de freios, motor, transmissão, possuía sua própria diagnose e particularidade, pois não era integrada, a manutenção tornou-se extremamente complexa (Hodel K. N., 2009).

A indústria identifica, conforme (Brennan, Buckland, & Christen, 2007) e (Mahmud & Alles, 2005), a necessidade de encontrar novas soluções para integrar, e de alguma maneira, compartilhar as informações dos sensores e dispositivos distribuídos pelo veículo, aperfeiçoar os sistemas através da redução dos custos de produção, simplificando a manutenção e tornando os automóveis mais baratos, simples e confiáveis.

Para atender a esta nova demanda emergente foi proposto da década de 1980 pela Bosch o protocolo de comunicação serial CAN, que foi lançado oficialmente em 1986, sendo o primeiro veículo a utilizar essa nova solução de engenharia o *Mercedes-Benz* W140, em 1991. Este projeto contou com a colaboração do Dr. Wolfhard Lowrenz da universidade de Brauschweing Wolfnbuttel, e do Dr. Horst Westtstem da universidade de Karlsrushe, além da Mercedes-Benz, fabricante de automóveis. A abreviação CAN veio do nome que o professor Dr. Wolfhard deu ao protocolo de comunicação serial "*Controller Area Network",* proposto por ele (Hodel K. N., 2009).

1.1 Motivação

A evolução tecnológica dos veículos automotores forçou a revolução na área de manutenção da frota de automóveis e veículos comerciais, o que vem exigindo cada vez mais a especialização do profissional de manutenção automotiva, gerando maiores oportunidades e desafios para esta área de atuação. Essa situação faz com que o profissional de manutenção automotiva busque a expansão de suas fronteiras de conhecimento para além do que já estava habituado como, por exemplo, realizar manutenção em veículos carburados, onde não havia nenhuma exigência de conhecimento de dispositivo multiplexados ou tampouco sobre eletrônica, o que vem sendo cada vez mais necessário nos automóveis e veículos comerciais atuais.

1.2 Objetivo

O objetivo deste trabalho é desenvolver método para definição dos conteúdos das mensagens do fluxo de dados em uma rede comunicação CAN e compreender o funcionamento tráfego de dados, conteúdo dos dados trafegados e estrutura do protocolo serial CAN utilizados em veículos automotores.

1.3 Conteúdo e organização

No capítulo a seguir será apresentado com detalhes o protocolo serial CAN e sua estrutura, bem como o resumo de como eram as estruturas de comunicação primitivas anteriores a aplicação deste protocolo nos automóveis. Serão apresentados a título de informações as principais vantagens do protocolo CAN, suas principais desvantagens e problemas, além, do seu funcionamento teórico básico e tipos de aplicações onde são recomendados e onde não são recomendados.

2 Referencial teórico

Este capítulo apresenta a fundamentação teórica básica para o entendimento deste projeto, passando pelos sistemas de comunicação, o protocolo, estratégia, funcionamento, a segurança e o funcionamento de sistema autodiagnostico.

2.1 Sistemas individualizados

Os sistemas de gerenciamento eletroeletrônicos individuais nos automóveis que utilizam dispositivos eletrônicos (sensores e atuadores elétricos) foram os primeiros a serem usados pela indústria, (Hodel K. N., 2009).

Este tipo de arquitetura destaca se por não ter nenhum tipo de interação com outros sistemas do veículo, por exemplo, o sistema de gerenciamento de motor não tem qualquer compartilhamento ou interação com dados provenientes do sistema eletrônico dos freios, ou com o sistema de gerenciamento do *AirBag* (Guimarães, 2001). Embora essa característica traga algumas vantagens, como simplicidade do hardware, trazem também desvantagens consideráveis, pois torna difícil, complexa e cara a diagnose e manutenção do veículo, uma vez que serão necessários diferentes tipos de ferramentas para realizar a diagnose de um mesmo veículo.

Outro ponto negativo neste sistema é a duplicidade de sensores, uma vez que cada sistema de gerenciamento precisa de seus próprios sensores para obtenção de uma mesma informação. O sensor de temperatura da injeção eletrônica não é utilizado para informar a temperatura do motor para o ar condicionado ou painel de instrumentos, necessitando assim de um sensor para cada sistema. Essa necessidade deixa a produção do veículo com custos ainda mais elevados, pois além do aumento do número de sensores também cresce o número de cabos e conexões dos sistemas (Guimarães, 2001).

2.2 Sistemas centralizados

Nos sistemas de arquitetura eletrônica centralizada, um único controlador é responsável pelo gerenciamento de todos os sistemas do automóvel. Esse modelo de arquitetura elimina a necessidade de várias ferramentas de diagnose, facilitando e simplificando a manutenção. O *hardware* é simples, pois um único é responsável por receber todos os dados de todos os sensores do veículo e acionar os atuadores (Hodel, Specht, & Onisic, 2002).

Estas são ótimas vantagens, pois toda a informação está disponível em um único ponto de diagnose. Entretanto, há desvantagens significativas, pois não possibilita a expansão do sistema após conclusão e aplicação do projeto, por exemplo, se for concebido sem sistema de ABS (*Antlock Braking System*) o veículo não poderá contar com esse opcional após fabricação.

Outra desvantagem, conforme (Hodel, Specht, & Onisic, 2002) é o elevado número de cabos necessários para conectar o controlador aos sensores e atuadores dispostos pelo veículo, pois o preço dos cabos eleva custos de produção e adicionam pesos indesejados no automóvel.

2.3 Sistemas de arquitetura distribuída

Por fim, os sistemas elétricos com arquitetura distribuída utilizam controladores individuais para cada sistema do veículo (Hodel K. N., 2009). Por exemplo, temos um controlador para gerenciamento do motor, outro para gerenciamento do câmbio automático e outro para gerenciar os freios eletrônicos, mas com vantagem de estarem interconectados através de uma rede de comunicação que permite o compartilhamento de informações entre os vários sistemas do veículo. Esta estratégia trouxe melhorias significativas para os automóveis e para a indústria, conforme (Fredriksson, 1994), pois possibilitou a redução de cabos e conexões, uma vez que os controladores podem ser instalados próximos aos sensores e atuadores, facilitando a aplicação. Por fim, trouxe também um único sistema de diagnose que atende a todos os sistemas do veículo, gerando economia com a manutenção e produção dos veículos, pois acabou com a duplicidade de sensores e reduziu o tamanho do cabeamento e número de conexões.

No entanto, este sistema demanda um *software* complexo para um gerenciamento da rede, o que dificulta o desenvolvimento por depender de escolha do protocolo de comunicação entre os controladores. Outro ponto é a dificuldade na determinação da taxa de transmissão, e dos componentes para os sistemas dos controladores a serem utilizados no projeto do veículo (Hofstee & Goense, 1999).

2.4 O protocolo de comunicação

Atualmente a indústria automotiva emprega o sistema de arquitetura distribuída em seus produtos (automóveis), e o protocolo CAN, conforme (Hodel K.

N., 2009), é o mais utilizado para estabelecer o controle do tráfego e compartilhamento de informações entre os diversos controladores do veículo.

O protocolo CAN, possibilita enormes benefícios, tais como, redução de peso pelo menor número de conexões e cabos, facilidade de diagnose, uma vez que esta é unificada, e flexibilidade para a expansão do sistema, permitindo que sejam adicionados outros sistemas posteriormente ao projeto, a exemplo a instalação de opcionais como sistema multimídia (Fredriksson, 1994).

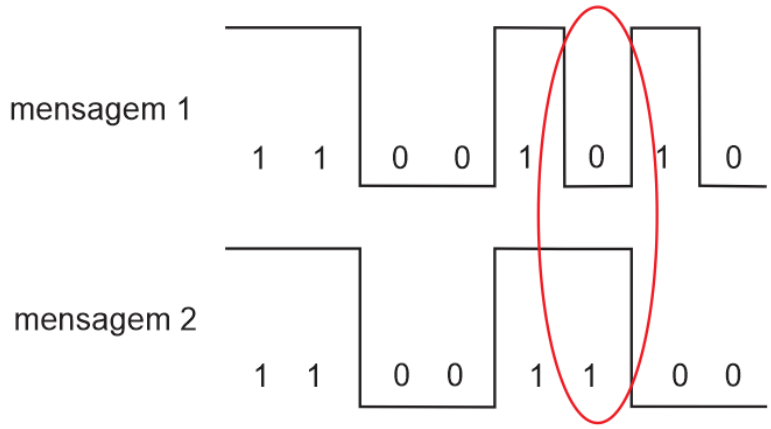
2.5 O CAN

Embora haja outras aplicações do protocolo CAN, estas não serão abordadas, pois esse trabalho limita aquelas utilizadas pela indústria automotiva, que são os protocolos CAN 2.0A e 2.0B. O que diferencia o CAN 2.0A e CAN 2.0B é o número de bits utilizados pelo identificador, que são 11 bits e 29 bits respectivamente. As normas que definem os padrões para estes protocolos são, (ISO 11898-2, 2003), (ISO 11898-3, 2006), (ISO 11992-1, 1998) e (SAE J2411, 2000).

O protocolo CAN é de comunicação serial síncrona realizada através de um par de cabos trançados (sinal diferencial), o sincronismo entre os dispositivos se dá no início da mensagem que trafegam na rede e pode operar a velocidade de até 1 Mbps (*Mega bit per second*). No entanto, possui restrições de velocidade em virtude do comprimento do barramento que interconecta os controladores, que possuem capacidade de reconhecer simultaneamente outros controladores e as mensagens que trafegam na rede, embora a escrita ou transmissão de uma mensagem só é permitida por um único controlador de cada vez (Hodel K. N., 2009).

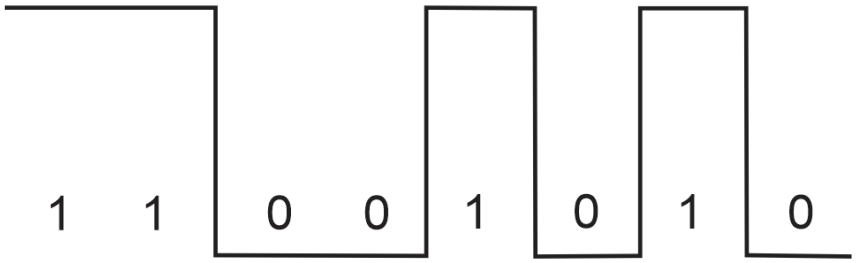
O protocolo CAN utiliza o conceito de multimestre, CSMA/CD+AMP (*Corrier Sense Multiplie Access/Collission Detection More Arbitration Or Message Priority*) para arbitrar a transmissão de dados no barramento. Conforme (Hodel, Specht, & Onisic, 2002) essa arbitragem se dá na ocorrência de dois ou mais controladores entrarem em conflito de transmissão. Utiliza-se então o arbítrio de comparação binária onde a mensagem com maior prioridade é transmitida no barramento. A prioridade é definida através do nível lógico de cada bit do identificador da mensagem, por exemplo, “0” é tido por bit dominante e “1” bit recessivo, sendo assim o nível lógico “0” tem prioridade em relação ao nível lógico “1”. Este conceito é ilustrado através das Figuras (1) e (2).

Figura 1 - Bit Dominante e Bit Recessivo



Fonte: Autor.

Figura 2 - Mensagem Transmitida



Fonte: Autor.

Outra característica do protocolo CAN é que as mensagens que trafegam no barramento não contêm endereços específicos de transmissão ou recepção, apenas o identificador de origem onde foi gerada. Assim os demais controladores receptores podem identificar o conteúdo e utilizar ou desprezar a informação (Hodel K. N., 2009).

2.6 Especificações ISO/OSI para o CAN

As especificações ISO/OSI (*International Organization for Standardization/Open System Interconnection*) para o protocolo CAN determinam se as camadas dessa rede se dividem em camada de enlace, física e aplicação.

- A camada física define como realmente os sinais são transmitidos dentro desta especificação da camada física não é definida de modo a permitir que o meio de transmissão e implementação de nível de sinal a serem otimizadas para suas aplicações.

- A camada de transferência representa o núcleo do protocolo CAN. Ele apresenta mensagens recebida na camada de objeto e aceita mensagens a serem transmitidas da camada de objeto. A camada de transferência é responsável pelo tempo de bit e sincronização e confinamento de falhas.
- A camada de objetos está relacionada à filtragem de mensagens, bem como ao estado e manipulação de mensagens.

O escopo desta especificação é definir a camada de transferência e as consequências de protocolo CAN nas camadas circundantes (Bosch, 1991). A camada de enlace é responsável pelo gerenciamento de falhas e erros de transmissão ou recepção e utiliza-se das subcamadas para isso o LLC (*Logic Link Control*) e Mac (*Media Access Control*).

- LLC é responsável pela filtragem de mensagens, notificação de sobrecarga e controle de recuperação;
- MAC encapsula/desencapsula dados, realiza codificação dos quadros (**"**bit *Stuffing***",** caso cinco bits consecutivos apresentam o mesmo nível lógico, insere-se um bit com valor inverso), controle de acesso ao meio, detecção e sinalização de erros.

Estas duas subcamadas são responsáveis ainda pelo confinamento de falhas, ou seja, um nó que estiver com muitos erros de transmissão ou recepção será automaticamente desligado da rede. O controlador CAN é responsável por lidar automaticamente com estes serviços de forma que o *software* não precisa se preocupar com estes serviços (Hodel K. N., 2009).

2.7 O tráfego de dados no CAN

O protocolo CAN utiliza para sincronização e tráfego de dados, quadros de transmissão. Estes quadros são apresentados na tabela (1) além da quantidade de bits de cada para o CAN 2.0A e 2.0B.

Tabela 1 - Quadros do protocolo e número de bits de cada quadro para CAN 2.0A e CAN 2.0B.

Nome	Quantidade de bits (CAN 2.0A)	Quantidade de bits (CAN 2.0A)
Start of Frame (SOF)	1	1
Identificador	11	29
RTR	1	1
SRR	0	1
IDE	1	1
DLC	4	4
Dados	8 a 64	8 a 64
CRC	15	15
ACK	1	1
End of Frame (EOF)	10	10

Fonte: Autor.

O primeiro bit do início de fluxo de dados é denominado SOF (*Start of Frame*), e é um bit dominante que tem como função iniciar a mensagem. O próximo quadro é o RTR (Requisição de Transmissão Remota) que identifica se o fluxo de dados é de requisição ou de dados transmitidos. O quadro seguinte é o IDE (*Message Identifier*) que indica se o CAN utilizado é o 2.0A ou 2.0B.

O quadro SRR (Substituto de Requisição Remota), indica a manutenção da compatibilidade entre o quadro estendido e padrão sendo um bit dominante para o CAN 2.0A e recessivo para o CAN 2.0B. Os bits R0 e R1 são reservados.

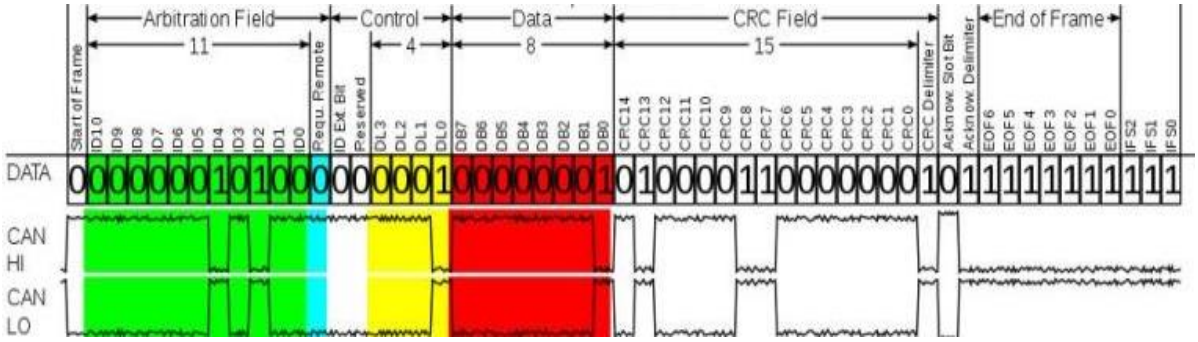
O corpo do Identificador indica o nó no barramento que está transmitindo a mensagem, e é responsável por determinar a prioridade de escrita no barramento, nesse campo se encontra os bits RTR e os bits que são utilizados para identificação da “assinatura” dos nós na rede para o CAN 2.0A padrão.

Os demais corpos de controle abrigam o DLC (*Data Length Code*), R0 e R1 (CAN 2.0B) que são bits de informação que indica o tamanho do corpo de dados que estão sendo transmitidos, o DLC utiliza 4 bits para tal.

O campo de dados utiliza 8 bytes de 8 bits para transmitir as informações de dados no CAN, exemplo, temperatura, velocidade ou qualquer outro tipo de informação.

O campo CRC, este utiliza 15 bits e é utilizado para gerenciamento de erros ao barramento e ao final do campo o bit é recessivo denominado delimitador. Após o CRC (*Checksum*), é o campo ACK (*Acknowledge*) este campo é preenchido por dois bits recessivos, caso a mensagem seja gerada de transmissão e dominante pelo nó receptor ao receber a mensagem transmitida este campo arbitra um tempo para transmissão e resposta entre nós do barramento. Ao final de um quadro de transmissão completo há sete bits recessivos. A Figura (3) ilustra a composição de cada campo.

Figura 3 - Esquema do Protocolo CAN



Fonte: (Nuñez, 2017)

2.8 Estratégias e funcionamento da comunicação

A comunicação entre os módulos controladores da rede, ao ser estabelecido, pode ser de requisição de dados, que ocorre quando algum módulo necessita de alguma informação e a solicita a algum outro controlador, sendo que este não possui o campo de dados e o bit RTR será recessivo (Bosch, 1991).

Informações de sobrecarga ocorrem por dificuldade de processamento de memória na recepção de algum quadro de recepção, sendo necessário um maior tempo de processamento, até que passa a receber novo quadro de dados. Este quadro possui 14 bits sendo os 6 primeiros dominantes e os 8 últimos recessivos, sendo estes últimos os delimitadores do quadro, e os 6 primeiros para campo de *Flag* (Hodel K. N., 2009).

Quando ocorre erro ativo todos os bits serão dominantes, ou todos os bits recessivos para *Flag* de erro passivo. Este tipo de informação é denominado, quadro de erro. Os quadros de sobrecarga possuem uma regra que consiste em uma limitação de 2 quadros em sequência.

2.9 O gerenciamento dos erros no can

Os erros detectáveis no CAN são cinco, sendo, erros de codificação, erro de bit, erro de CRC, erro de formação e erro de ACK. Em cada controlador da rede CAN há um registrador de erro de transmissão e recepção que são utilizados para conferir falhas e controlar os erros do barramento, isso é de extrema importância para confiabilidade do funcionamento do sistema e prevenir o colapso da rede (Bosch, 1991).

Assim, (Hodel K. N., 2009) é possível definir o estado de cada módulo controlador e tomar decisões de ativar ou desativar um controlador da rede. O estado dos controladores pode ser ativo, quando não há qualquer erro e o nó participar sem restrição do barramento, ou passivo, quando os registradores de erro têm valor igual a 128. Neste estado o nó terá restrições de transmitir pacotes de dados, só podendo voltar ao estado ativo caso seja decrementado a um valor inferior a 128 e os registradores de erro inativos. Caso os registradores tenham valor igual ou superior a 256, neste estado o nó não participa do barramento sendo impedido de realizar qualquer modificação.

Um controlador após estar no estado inativo só poderá retornar ao estado ativo após ocorrer 128 pacotes de 11 bits recessivos, e seus contadores de erros serem zerados.

2.10 Ordem de transmissão

O tempo que separa os quadros de aquisição e transmissão de dados, também é utilizado para controle da transmissão dos controladores passivos. Este tempo se divide em duas áreas para os controladores ativos, área de interdição e área livre para transmissão (Hodel K. N., 2009).

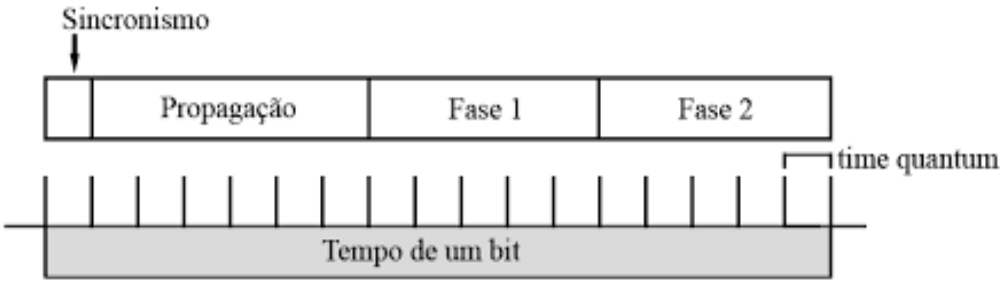
A área de interdição possui três bits recessivos. No caso dos nós em estado passivo há uma área chamada intermediária que é de 8 bits recessivos, neste estado o nó não transmite quadros nesta área. No entanto este controlador poderá receber transmissões de mensagens de outros controladores. As mensagens transmitidas no barramento são filtradas no nó por meio do identificador, onde é comparado com o filtro do identificador previsto. Ao receber uma mensagem onde não há erros o controlador processará os dados recebidos e transmitirá sinalização de recepção concluída sinalizando com bit ACK dominante (Hodel K. N., 2009).

2.11 Segurança e configuração do protocolo em função do meio de transmissão

Um ponto extremamente importante para o funcionamento do protocolo CAN em sua aplicação, são as regras e características de suas conFigurações, como exemplo, o comportamento e reações do meio físico em que é construído que deve ser levado em conta, pois influenciará diretamente no desempenho do sistema, podendo afetá-lo drasticamente causando falha.

Conforme (Hodel K. N., 2009) sendo o CAN um sistema de comunicação de tempo real, qualquer perturbação afetará a comunicação, sendo necessária uma série de conFigurações para que possa minimizar os problemas. Por exemplo, para que se atinja a velocidades determinadas no projeto, algumas regras de conFiguração devem ser respeitadas. Então parte se do entendimento de quanto tempo 1 bit necessita para ser transmitido na rede, para isso secciona o tempo de um bit em quatro partes, mostrado na Figura (4).

Figura 4 - Fracionamento do Tempo de Transmissão de 1 Bit



Fonte: Extraído de (Maurici, 2005)

Sendo estas divisões nomeadas de sincronismo, propagação, fase 1 e fase 2. Dentro destas há subdivisões de uma unidade temporal denominada TQ (*Time Quantum*) (Bosch, 1991), está por sua vez é definida pela taxa de transmissão selecionada do barramento CAN e a velocidade da rede.

Estes segmentos são seccionados da seguinte forma, Segmento de sincronismo tem o tamanho temporal de 1 TQ o segmento de propagação possui 8 TQs, sendo este segmento responsável pela compensação dos atrasos causados pelo meio físico de tráfego, tais como cabos de cobre, características dos componentes eletrônicos utilizados na construção dos nós, entre outros. Este segmento também compensa aos atrasos temporais de transmissão, propagação e recepção das mensagens do sistema nos nós.

O segmento de fase 1 também poderá ter de 1 a 8 TQs e pode variar durante o trafego de informações, pois está ligado diretamente a sincronização dos *Clocks* do sistema de nós. Neste segmento é determinado o ponto onde será executada a medição do valor do sinal que está sendo monitorado no momento.

O segmento de fase 2 também poderá chegar a ter no máximo 8 TQs, e as transmissões só ocorrerão no barramento após a conclusão da verificação deste segmento. Outro ponto importante é que há um fator de início de sincronização que poderá ter até 4 TQs, que segue uma regra de não ser maior que qualquer dos segmentos de fase 1 ou fase 2, servindo para indicar o quanto o ponto de verificação do sinal poderá ser movido para resincronização do sinal, que novamente reitero ser dependente das características físicas dos materiais de construção do sistema, que interferem nos relógios *clocks* do sistema.

2.12 X-by-wire

Os sistemas *X-By-Wire* são sistemas eletronicamente controlados que originalmente eram totalmente mecânicos, hidráulicos ou pneumáticos, mas foram substituídos a fim de aperfeiçoar alguma determinada operação, por exemplo, o sistema de freios antitravamento de um veículo (ABS) (Quigley, Mcmurrn, Jones, & Faithfull, 2007). Esse sistema monitora as rodas de um veículo e a solicitação de frenagem do condutor, evitando que as rodas do veículo travem, preservando assim a eficiência da frenagem e reduzindo a chance de acidentes. Este tipo de sistema de evento “frenagem solicitada” e resposta “efetivamente” com atuação dos freios diminuindo a velocidade das rodas e consequentemente do veículo, não pode ser, por exemplo, via barramento CAN, pois um dos principais problemas aqui é que a flexibilidade do CAN tolera atrasos nas mensagens, também ocorre que se há alguma mensagem sendo transmitida no barramento, esta não poderá ser interrompida até sua conclusão.

Nota-se então um grave problema, pois se uma mensagem estivesse sendo transmitida na rede, e o sinal da solicitação de frenagem do veículo surgisse, ela só iria ganhar o barramento após a completa recepção do nó destinado, e em uma emergência este tempo crucial interferiria na eficiência dos freios. O protocolo CAN não é indicado para este tipo de operação, sendo para tal outras ferramentas mais apropriadas, pois o tempo de resposta é extremamente crítico.

Quanto mais nós no sistema, maior taxa de ocupação do barramento poderá ocorrer, assim gerando também maior número de atrasos, tanto de transmissão como de recepção. A taxa de ocupação do barramento é denominada *Bus-Load* (Bosch, 1991).

Dado essa introdução sobre o protocolo CAN e seu funcionamento, será apresentado os trabalhos práticos de aquisição de dados de tráfego da rede CAN do veículo *Fiat* Strada e *Volkswagen* Gol, selecionado os sinais de freios, velocidade, rotação do motor e identificadas dentre todas as mensagens do tráfego.

Será desenvolvido um circuito interface e *software* capaz de identificar e tornar os dados úteis para o usuário, facilitando a compreensão de leigos e profissionais da manutenção automotiva das informações que trafegam no barramento CAN.

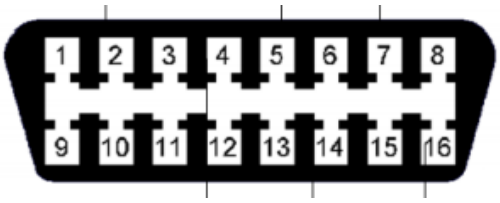
2.13 Diagnose veicular

Atualmente as funções básicas do veículo dependem da eletrônica e estes sistemas devem atender altas exigências de confiabilidade, ao mesmo tempo em que o sistema de gerenciamento eletrônico do motor deve manter os níveis de emissões de acordo com os limites estabelecidos na legislação (Bosch, 1991). Com isso, a solução é acrescentar no *software* de funcionamento do veículo a estratégia de autodiagnostico. Baseia-se na eletrônica já instalada no veículo para monitorar as principais partes do sistema continuamente além dos sinais de entradas e saídas e as comunicações com os módulos do veículo. O conjunto das funções dedicadas ao diagnóstico de falhas forma o sistema diagnóstico embarcado do veículo e representa cerca de 50% de todo o *software* presente nos sistemas de controle eletrônico dos veículos atuais. A importância dos sistemas de diagnóstico embarcados aumentou significativamente devido as leis ambientais que regulamentaram a quantidade de poluentes emitidos (Belo, 2003).

Dentre as regulamentações ambientais, a qual definem limites rígidos para a quantidade de poluentes emitida pelos veículos automotores. Logo após foram estabelecidos novos processos de monitoramento e diagnóstico de falhas que levaram ao surgimento da segunda geração de sistemas de diagnósticos embarcados. A conexão ao sistema consiste em um conector padronizado que foi sancionado como obrigatório na Europa e nos Estados Unidos para todos os

veículos produzidos desde 2000 e 1996, respectivamente dado o nome de OBD-2. Já no Brasil surgiu o OBDBr-2 a partir de 2010, a medida principal tem a característica de controle ativo das emissões veiculares, porém também tem a finalidade de popularizar o serviço de reparo eletrônico, reduzindo drasticamente o custo das oficinas, possibilitando os consumidores pagarem mais barato por esse gênero de serviço. Além disso, a padronização e abertura dos protocolos de comunicação trouxeram ao mercado equipamentos extremamente baratos e de fácil acesso. O protocolo OBD define um conector padrão para todos os veículos a fim de se padronizar a aplicação que antes era definida pelo próprio fabricante do veículo. Tal conector é ilustrado pela Figura (5) e a função de seus terminais é agrupada na tabela (2).

Figura 5 - Conector OBD



Fonte: Adaptado de (Napro, 2015)

Tabela 2 – Função dos terminais do conector OBD-II.

1	Fabricante
2	SAE J1850 positivo
3	Fabricante
4	Terra do chassi
5	Sinal de Terra
6	CAN High
7	Linha K
8	Fabricante
9	Fabricante
10	SAE J1850 negativo
11	Fabricante
12	Fabricante
13	Fabricante

14	CAN Low
15	Linha L
16	Bateria

Fonte: Adaptado de (Napro, 2015)

3 Metodologia

Tomando o barramento CAN dos veículos de estudo *Fiat* Strada e *Volkswagen* Gol como objetos de estudo. Foi utilizado o equipamento "*CANALYSTII*" para comunicação entre veículo e o computador para a aquisição dos dados de tráfego do barramento. Também foi confeccionado um cabo adaptador para obter acesso ao barramento CAN pelo conector OBD do veículo Strada.

Efetuamos a varredura do tráfego de dados do barramento foi selecionado e identificado os seguintes dados e sinais, de TPS (*Throttle Position Sensor*) pelo ID 0x361 e no *byte* 04 do frame de dados a informação da posição e estado da borboleta do acelerador onde os bits 3E posição de repouso da borboleta e 0xFF plena carga, ou seja, abertura máxima da borboleta.

Logo após foi observado no mesmo identificador o dado referente à rotação do motor sendo está localizada no quinto e sexto byte, sendo 0x00, para rotação nula, ou seja, motor em repouso e 0xFFFF para máxima rotação do motor. Neste mesmo ID temos ainda a informação do pedal de acelerador no primeiro byte variando de 0x00 a 0xFF respectivamente mínimo e máximo do pedal de acelerador.

O ID 0x560 abriga as informações referentes a porta, faróis e freio de estacionamento. O terceiro byte possui a informação da porta, sendo 0x40 porta aberta e 0x48 porta fechada, o oitavo byte estato dos faróis, sendo 0x80 apagados e 0xA0 acessos e por fim o estado do freio de estacionamento no segundo byte sendo 0x00 para não acionado e 0x20 acionado. Essa informação é útil, por exemplo, para que a partida do motor seja liberada ou não. Sendo também observado no ID 0x3A1 no sexto byte o estado do pedal freio que também é observado para liberação ou não do sinal de partida do motor o valor 0x00 pedal em repouso e 0x10 pedal freio pressionado.

A informação de temperatura do motor que será utilizada por este trabalho se encontra no ID 0x561 no quarto byte onde o valor de 0x7D representa 85°C.

Observou-se que a informação de temperatura e atualizada do barramento de 1 em 1 grau observando sua variação após aquecer o motor até 85° C e utilizando um multímetro *Mastech* 8229 CAN um termopar para medir a temperatura após desligar o motor e observar a queda da temperatura até que atingiu a temperatura ambiente onde o teste foi encerrado.

O ID 0x56B é responsável pelo tráfego de informações referentes ao estado dos comandos da transmissão *Dualogic* do veículo o campo de dados do byte 06 identifica as marchas do veículo engatadas, o quadro abaixo ilustra esta informação efetuando a mudança das marchas no veículo, no byte 07 há a informação do botão de modo *Sport*, seleção de modo automático e manual das trocas de marchas.

Por fim o ID 0x5A0 traz a informação da velocidade do veículo conforme a Figura (6) a informação trafegada é a velocidade referida a informação está por sua vez será utilizada nesse trabalho. Esta informação foi levantada com o veículo em funcionamento e simulando no dinamômetro as condições de tráfego levando o veículo de 0 a 80 km/h utilizando a leitura do ôdometro para leitura da velocidade indicada e o seu dado atribuído no barramento CAN. As Figuras (6) e (7) se encontram os dados obtidos nos primeiros testes realizados no veículo *Fiat* Strada.

Figura 6 - Dados coletados via CANALYSTII na CAN da Fiat Strada em 500Kbps

<table>
<tr><th>ID</th><th>DLC</th><th>Byte 0</th><th>Byte 1</th><th>Byte 2</th><th>Byte 3</th><th>Byte 4</th><th>Byte 5</th><th>Byte 6</th><th>Byte 7</th></tr>
<tr><td>0x5A0</td><td>8</td><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="2">Velocidade =
(B6 * 30) + (B7/3)</td></tr>
<tr><td rowspan="4">0x56B</td><td rowspan="4">8</td><td rowspan="4"></td><td rowspan="2">Câmbio
0b00000001, 0
= Sem Sport
1 = Com Sport</td><td rowspan="4"></td><td rowspan="4"></td><td rowspan="4"></td><td>Neutro 0x00</td><td rowspan="4"></td><td rowspan="4"></td></tr>
<tr><td>1º Marcha 0x02
2º Marcha 0x04</td></tr>
<tr><td rowspan="2">Câmbio 0b00010000
0 = Manual
1 = Automático</td><td>3º Marcha 0x06
4º Marcha 0x08</td></tr>
<tr><td>5º March 0x0A
6º Marcha 0x0C</td></tr>
<tr><td>0x560</td><td>8</td><td>Farol Desligado
0b00100000
0 = desligado
1 = ligado</td><td></td><td></td><td></td><td>Temp. do Motor =
B4 - 40</td><td>Portas
0b00001000
0 = fechada
1 = aberta</td><td></td><td></td></tr>
<tr><td>0x3A1</td><td>8</td><td></td><td></td><td>Freio Estacionamento
0b00010000
1 = Acionado
0 = desacionado</td><td></td><td></td><td></td><td>Freio de Serviço Acionado
0b00100000
1 = Acionado
0 = desacionado</td><td></td></tr>
<tr><td>0x361</td><td>8</td><td></td><td colspan="3">RPM = (B1 + B2)/3
Obs.: RPM do motor</td><td></td><td></td><td></td><td></td></tr>
</table>

Fonte: Autor.

Com os dados adquiridos seguiu-se o desenvolvimento do s*oftware* de comunicação para a placa com o *transceiver* CAN e apresentação das mensagens selecionadas ao conecta-la ao barramento CAN do veículo *Fiat* Strada ou *Volkswagen* Gol. Estes testes foram realizados na rede CAN entre módulos de motor e de câmbio *Dualogic* e após notar a imprecisão de algumas informações foi decidido novamente levantar os dados de informações sobre as trocas de marchas que não foi possível à constatação se o veículo possuía uma 5ª marcha, pois o mesmo não foi testado no dinamômetro então os testes realizados não foram executados com carga no motor e o sistema não permitia a troca de todas as marchas.

Realizado mais um teste com o *Fiat* Strada, desta vez no dinamômetro, e devidamente instrumentada, tomou-se por acesso ao barramento CAN a porta de comunicação OBD foi observado uma grande diferença, pois a rede CAN no conector é diferente do que observou no teste anterior, está também contém

mensagem de IDs diferente e para as mesmas informações os dados são diferentes além de que esta rede presente no conector possui taxa de transmissão de apenas 50Kbps e não 500Kbps do barramento entre o controlador do motor e da transmissão *Dualogic*.

Optou-se por utilizar no projeto prático desde o início os dados levantados através deste barramento do conector OBD. A Figura (7) detalha os dados levantados nestes testes suas informações e significados.

Figura 7 - Dados coletados via CANALYSTII na CAN da Fiat Strada em 50Kbps

<table>
<tr><th>ID</th><th>DLC</th><th>Byte 0</th><th>Byte 1</th><th>Byte 2</th><th>Byte 3</th><th>Byte 4</th><th>Byte 5</th><th>Byte 6</th><th>Byte 7</th></tr>
<tr><td rowspan="4">0x380</td><td rowspan="4">8</td><td rowspan="4"></td><td rowspan="4"></td><td rowspan="4"></td><td rowspan="4"></td><td>Portas
0b01001101
0 = fechada
1 = aberta</td><td rowspan="4"></td><td rowspan="4"></td><td>Freio de Estacionamento
0b00100001
0 = ligado
1 = desligado</td></tr>
<tr><td>Lanterna
0b01001011
0 = ligado
1 = desligado</td><td rowspan="3"></td></tr>
<tr><td>Farol
0b01001000
0 = Farol Alto
1 = Farol Baixo</td></tr>
<tr><td>Freio de Serviço
0b01001101
0 = ligado
1 = desligado</td></tr>
<tr><td rowspan="2">0x28B</td><td rowspan="2">8</td><td rowspan="2"></td><td rowspan="2"></td><td rowspan="2"></td><td rowspan="2"></td><td rowspan="2"></td><td>Neutro 0x1F
Marcha Ré 0x1C
1º Marcha 0x02
2º Marcha 0x04</td><td>Câmbio 0b00100000,
0 = Sem Sport
1 = Com Sport</td><td rowspan="2"></td></tr>
<tr><td>3º Marcha 0x06
4º Marcha 0x08
5º Marcha 0x0A
6º Marcha 0x0C</td><td>Câmbio 0b10000000
0 = Manual
1 = Automático</td></tr>
<tr><td>0x281</td><td>8</td><td></td><td colspan="2">RPM = (B1 + B2)/3</td><td></td><td>Temp. do Motor =
B4 - 40</td><td></td><td></td><td></td></tr>
<tr><td>0x180</td><td>8</td><td></td><td></td><td></td><td></td><td></td><td>Setas Desligadas 0x00
Seta p/ Direita 0x20
Seta p/ Esquerda 0x40
Alerta Ligado 0x60</td><td></td><td></td></tr>
</table>

Fonte: Autor.

Os mesmos testes realizados no veículo *Fiat* Strada foram realizados no *Volkswagen* Gol. A tabela (3) identifica os dados obtidos através da rede CAN.

Tabela 3 - Dados coletados via CANALYSTII na CAN da Volkswagen Gol em 500Kbps

ECU	ID	DLC	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
Motor1	0x280	8	Pedal_Acelerador @bit 0 0b00000001 1 Solto, 0 Press Embreagem_Status @bit 3 0b00001000 1 solto, 0 press		Rotação Fator 1/4, RPM			Pedal 0 - 100% 0 - 250		
Motor2	0x288			T_Agua Fator 1,0 Offset 75°C	Pedal_Freio1 @bit 0 0b00000001 Pedal_Freio2 @bit 1 0b00000010 Cruise_Control @bit 4 0b00010000					
Motor4	0x480			EPC @bit 2 0b00000100 1 liga, 0 desliga AC_ON_OFF @bit 4 0b00010000 1 desliga, 0 liga	Consumo de combustível acumulado ul - microlitros OBS: Bit 15 fica grampeado em 1 após o primeiro estouro do valor acumulado.					
Motor8 (exclusivo ECU v2)	0x590		Ref. Torque 0 - 100% Fator 0,01		Velocidade Medida Fator 0,01 Km/h		Marcha Engatada 0 (neutro) - 5	Pedal_Embreagem @bit 0 0b00000001 1 Press, 0 Solto Pedal_Freio @bit 1 0b00000010 1 Press, 0 Solto Modo_Cruzeiro @bit 2 0b00000100 1 ligado, 0 desligado Modo_Pedal_Sim @bit 3 0b00001000 1 ligado, 0 desligado	Ref. Cruzeiro Fator 1,0 Km/h	Erro de Hardware 0 - Sem erro > 0 - Erro detectado! OBS: Ver"protocolo.txt" para detalhes do erro.
Painel1	0x320	8		Freio_Estac @bit 1 0b00000010 1 ligado, 0 desligado Luz_Bateria @bit 2 0b00000100 1 ligado, 0 desligado	Tanque_Comb bit0 - bit6, 0-127 L Reserva_Comb @bit 7 0b10000000 1 Sim, 0 Não				Velocidade Fator 4/3, Km/h	
Painel2	0x420			T_Ambiente_delay Fator 0,5 Offset -50, °C	T_Ambiente Fator 0,5 Offset -50, °C	T_Oleo °C - não tem	T_Agua_Painel Fator °C			
Painel3	0x520							Hodômetro (km)		
Rede	0x470	5	Seta_Esquerda @bit 0 0b00000001 Seta_Direita @bit 1 0b00000010 Pisca_Alerta @bit 3 0b00001000 Luz_Ré @bit 5 0b00100000 1 ligado/aberta, 0 desligado/fechada	Porta_Motorista @bit 0 0b00000001 Porta_Passageiro @bit 1 0b00000010 Porta_TE @bit 2 0b00000100 Porta_TD @bit 3 0b00001000 Capô @bit 4 0b00010000 Porta_Malas @bit 5 0b00100000	Lanterna / Backlight Painel 0 - desligado, min 37 max 76?					
ACC1 (exclusivo ECU v2)	0x200	4	"E" Token de segurança	"C" Token de segurança	"U" Token de segurança	Pedal Simulado 0 - 100% 0 - 255				
ACC2 (exclusivo ECU v2)	0x201	8	"E" Token de segurança	"C" Token de segurança	"U" Token de segurança	Modo_Pedal_Sim 1 Liga, 0 Desliga	Modo_Operação 1 Econom, 0 Normal	Modo_Ref_Mlenta 0 - 11 0 = "default" 800RPM, M. Lenta (RPM) = 600 + (dado-1)*50	Reservado	Reservado

Fonte: Autor.

3.1 Ferramentas

Nesta seção será descrito as ferramentas utilizadas no presente trabalho, assim como uma breve descrição destas.

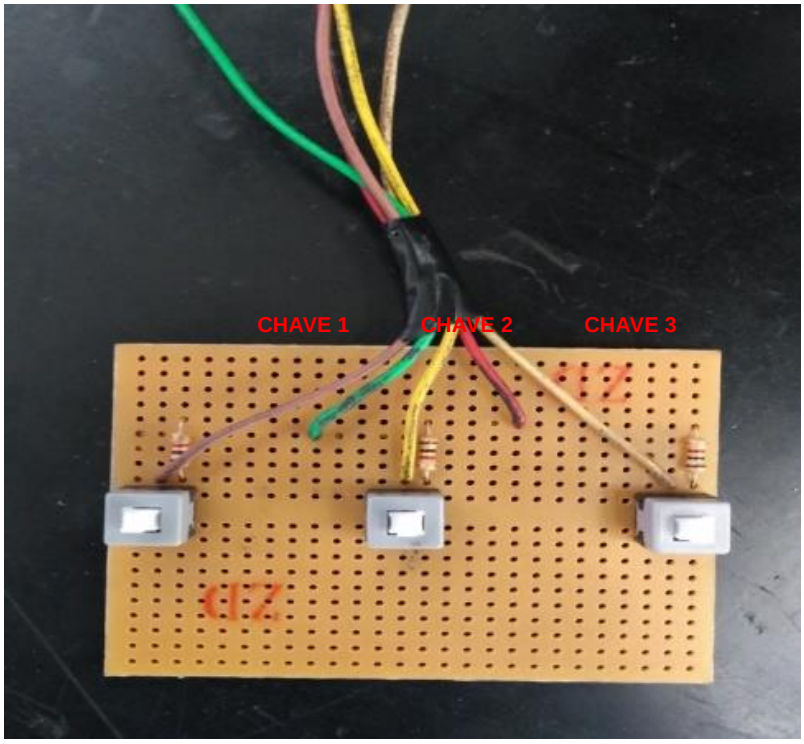
3.1.1 Canalystii

O “*CANALYSTII*” é uma ferramenta de análise profissional para coletar, desenvolver, testar, manter e gerenciar a rede CAN de um veículo e tem operação em todos sistemas operacionais. Ele é integrado com 2 canais CAN independentes que atendem à norma (ISO 11898-2, 2003) e podem processar mensagens CAN CAN 2.0A e 2.0B e está equipado com interfaces USB (*Universal Serial Bus*).

3.1.2 Placa auxiliar

A placa auxiliar que montamos insiste em apenas 3 botões seletores que foram soldados junto à resistores para limitar a corrente que entra no microcontrolador, está sendo utilizada para controlar os dados selecionados no LCD.

Figura 8 - Placa Auxiliar

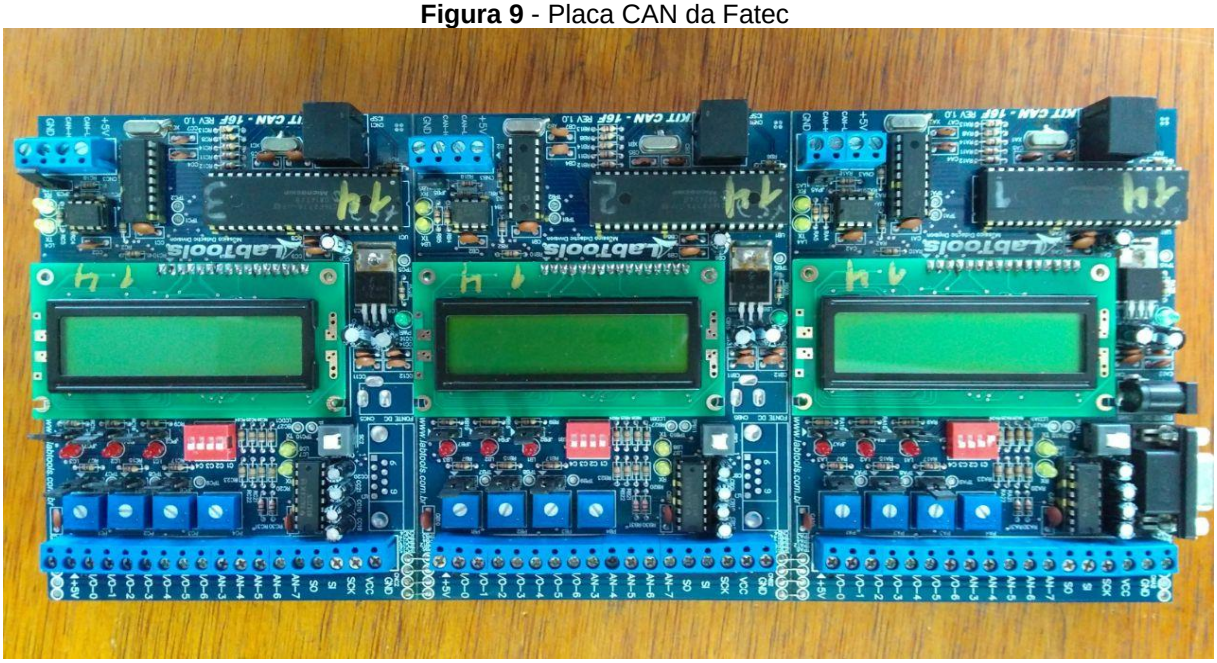


Fonte: Autor.

3.1.3 Placa CAN da Fatec Santo André

Esta ferramenta está pronta para uso para nós aluno da Fatec Santo André e serve para fazermos testes de comunicação com outros *hardwares*, basta que o aluno construa a aplicação para tal tarefa conforme o *hardware* desejado, tendo 15 entradas ou saídas e um display alfanumérico.

Figura 9 - Placa CAN da Fatec



Fonte: Autor.

3.2 Definições dos parâmetros para configuração do software

De posse das taxas de transmissão do *Volkswagen* Gol e da *Fiat* Strada calculou-se o tempo de bit de cada um dos barramentos sendo o tempo de bit o inverso da taxa de transmissão, essa informação é necessária para o cálculo do valor de TQ, que no *Volkswagen* Gol foi de 200 nS. A fórmula utilizada é ilustrada pela equação (1).

$$Tq \quad \frac{2*(BRP+1)}{20Mhz} \qquad \textbf{(1)}$$

onde BRP (*Baud Rate Prescaler*) foi estimado em 1 e 20MHz é a frequência do *clock* da placa utilizada neste trabalho, assim com o valor de TQ e o tempo de bit foi possível determinar os segmentos de sincronismo, propagação, fase 1 e fase 2, pois tornou-se possível determinar a quantidade de TQs no tempo de bit do barramento do *Volkswagen* Gol e da *Fiat* Strada que é expresso por “N° de TQ=Tempo de bit/Tempo de TQ”, no veículo *Volkswagen* Gol foi encontrado de 10 TQs, determinou-se então, 1 TQ na fase de sincronismo, 1 TQ na fase de propagação, 4 TQs na fase 1 e 4 TQs na fase 2. Para o veículo *Fiat* Strada utilizou-se os mesmos métodos, obtendo 20uS de tempo de bit, o valor de BRP estimado em 19, tempo de TQ de 2uS e por fim, foi encontrado 1 TQs. Logo, determina-se 1 TQ na fase de sincronismo, 1 TQ na fase de propagação, 4 TQs na fase 1 e 4 TQs na fase 2.

Estas informações são necessárias para a conFiguração do *software* do micro controlador e a comunicação CAN da placa que será conectada aos veículos através do barramento, são identificadas para conFiguração do CNF1, CNF2 e CNF3.

A seguir definiram-se os filtros e máscaras para ignorar o trafego das demais mensagens do barramento e possibilitar a recepção apenas das mensagens desejadas que serão apresentadas no *display*.

3.3 O *Software*

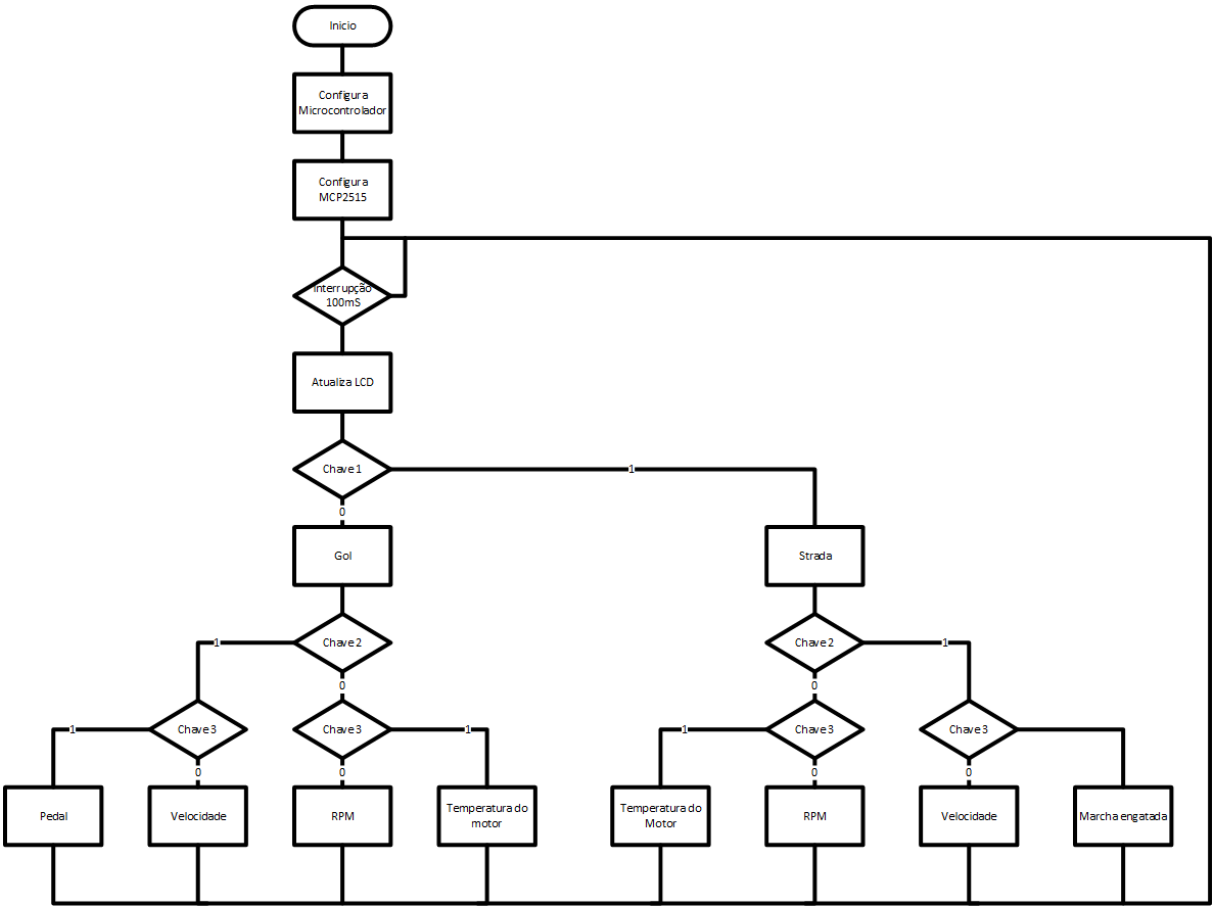
O *software* construído para aplicação deste trabalho se divide em 4 secções, sendo elas a configuração do LCD, onde serão apresentadas as mensagens desejadas, onde o código é encontrado no apêndice(A).

O código de configuração da comunicação do SPI localizado no apêndice(C), onde tem a função de fazer a comunicação entre o *transceiver* (MCP2515) e o microcontrolador (PIC16F877A).

A configuração dos valores obtidos do veículo tem como objetivo de definir as informações e mensagens que serão apresentadas no LCD e a forma como serão escritas, código apresentado no apêndice(B), tal como a definição da leitura das chaves seletoras de função da placa auxiliar, onde o programa fará a leitura dos estados das chaves e assim será designada a função desejada e a informação que será exibida no LCD.

A última secção descrita é a função principal do *software* denominada de "*main*", ela agrega e unifica todas as demais funções para a execução de todo o programa, nesta secção é configurado o controle dos filtros e mascaras das mensagens selecionadas que se deseja coletar e despreza as demais mensagens do trafego do barramento CAN, nesta secção também é configurada a taxa de transmissão, tipo de barramento, podendo ser padrão ou estendido, tempo de bit e suas divisões de fases (sincronismo, propagação, fase 1 e fase 2), o código está localizado no apêndice (D).

Figura 10 - Arquitetura do Software



Fonte: Autor.

3.4 Os Testes

Desenvolvemos uma placa ilhada auxiliar com botões seletores, para mostrar os dados desejados ao usuário. A primeira chave seleciona qual veículo será utilizado, *Fiat* Strada ou *Volkswagen* Gol. A segunda e terceira chave são utilizadas para a seleção dos modos de operação a ser realizados em cada veículo definidos no programa como mostrado na Figura (10).

Figura 11 - Tabela da Verdade das Chaves Seletoras

CHAVE 1	Resposta
0	Strada
1	Gol

STRADA		
CHAVE 2	CHAVE 3	Resposta
0	0	RPM
0	1	TEMPERATURA
1	0	VELOCIDADE
1	1	MARCHAS

GOL		
CHAVE 2	CHAVE 3	Resposta
0	0	RPM
0	1	TEMPERATURA
1	0	VELOCIDADE
1	1	ACELERADOR

Fonte: Autor.

Os botões de retenção são alimentados com a tensão de 5V e GND. Foi necessário utilizar um resistor para cada botão de 10KΩ na alimentação para limitar a corrente que entra no microcontrolador. Os sinais de saída dos botões estão ligados nos terminais 0, 1 e 2 do microcontrolador.

Utilizamos o BOB (*Break Out Box*) no veículo *Volkswagen* Gol para o acesso ao barramento CAN, a fim de se realizar a leitura os dados e testar o programa, pois o conector OBD do veículo estava danificado. Com o BOB instalado conectamos o barramento CAN do veículo à placa CAN da Fatec, onde programamos o microcontrolador com o programa desenvolvido para recepção dos dados para ser apresentados no LCD da placa CAN da Fatec.

Também utilizamos o equipamento "*CANALYSTII*" para monitoramento e confirmação dos dados que estavam trafegando no barramento, a análise da viabilidade do sistema para verificarmos se o sistema está funcionando corretamente.

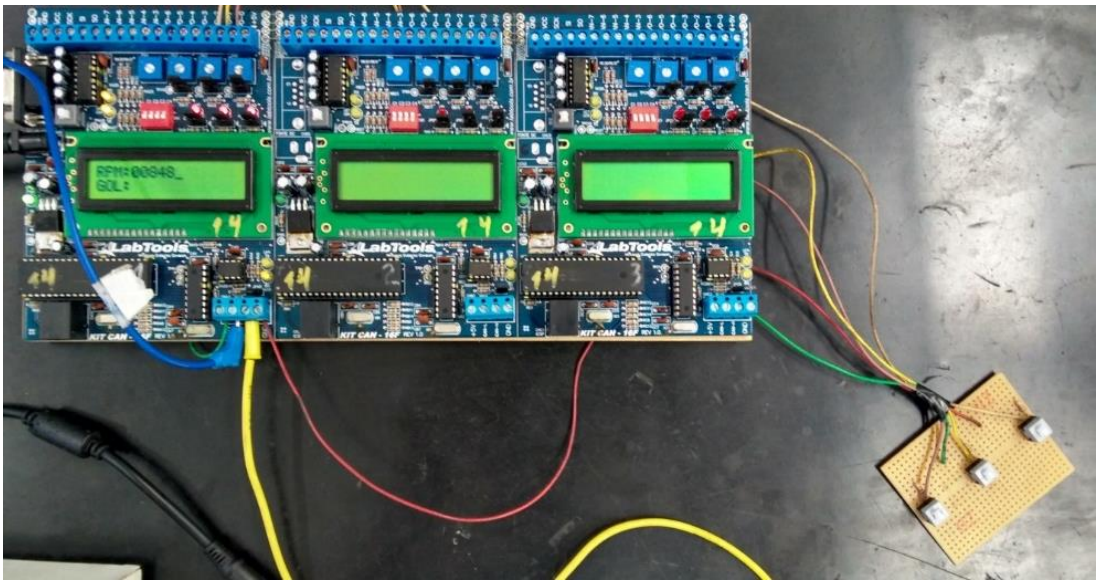
Figura 12 - Teste de Verificação do Sistema

Fonte: Autor.

No veículo *Fiat* Strada não foi necessário a utilização do BOB, pois o conector OBD não apresentava qualquer problema, então utilizamos um cabo adaptador da CAN para conseguirmos fazer a comunicação entre o veículo e o a placa CAN da Fatec. Coletado os dados e efetuados os testes, acompanhamos os valores obtidos para observar se o sistema desenvolvido é capaz de atender ao que foi proposto neste trabalho. No veículo *Volkswagen* Gol os mesmos parâmetros foram

averiguados com exceção da informação das marchas, pois este veículo não apresenta está informação devido ao seu câmbio ser mecânico sem qualquer assistência eletrônica. Substituímos esta informação pela posição do pedal de acelerador, que será apresentado no display em porcentagem.

Figura 13 - Teste de RPM em Marcha Lenta no Volkswagen Gol



Fonte: Autor.

4 Resultados obtidos

Como programado executou-se os testes após a conclusão do *software*, nos dois veículos na data de sábado do dia 07 de julho de 2018. No período das 16 horas, executamos os testes no *Volkswagen* Gol, onde conectamos o BOB no sistema eletrônico de controle do motor e então tivemos acesso ao barramento CAN do veículo, sendo os pinos 31 e 32 que são respectivamente o CAN *High* e CAN *Low*. O programa em questão que é o resultado das investigações a que se propõe este trabalho de entender o conteúdo de informações do trafego de dados e filtras aquelas de interesse do desenvolvedor e apresentar no LCD da placa CAN da Fatec. Já a placa auxiliar tende a interagir com o microcontrolador fazendo a seleção do que irá ser mostrado no LCD.

Com todos os equipamentos necessários instalados no veículo, e o mesmo no dinamômetro, iniciamos os testes em linha 15 e depois demos partida no veículo, pois alguns dados conseguiríamos somente com o veículo ligado, sempre observando as condições no LCD e selecionando-as através dos botões. Podemos notar que todas as informações apresentadas no LCD estavam condizentes ao painel do veículo, e checando todas as leituras, concluímos e validamos os testes de *software* no veículo *Volkswagen* Gol.

Os testes no veículo *Fiat* Strada seguiram-se a mesma rotina executada no *Volkswagen* Gol, porém desta vez nenhuma função funcionou e não conseguimos conectar-se ao barramento CAN deste veículo. Fizemos diversas tentativas para descobrir os motivos para tal, depois de revisarmos o *software*, não encontramos as causas, exceto a linha 128 do código, que continha a função de carregar o *buffer* de saída com uma mensagem de teste que carregamos para testar a comunicação, que não foi um problema no *Volkswagen* Gol. Suspeitamos que os *softwares* de segurança da *Fiat* Strada eram mais sensíveis e inibia a participação de outro nó transmissor com uma mensagem desconhecida, então fomos forçados a desabilitar a mensagem (colocando-a como um comentário). A partir de então, regravamos o microcontrolador e testamos novamente. Não conseguimos comunicação no barramento, levantamos a hipótese de o problema estar no casamento de impedância entre a Placa CAN da Fatec e o veículo *Fiat* Strada. Mudamos alguns *jumpers* na placa CAN da Fatec para conseguirmos casar a impedância.

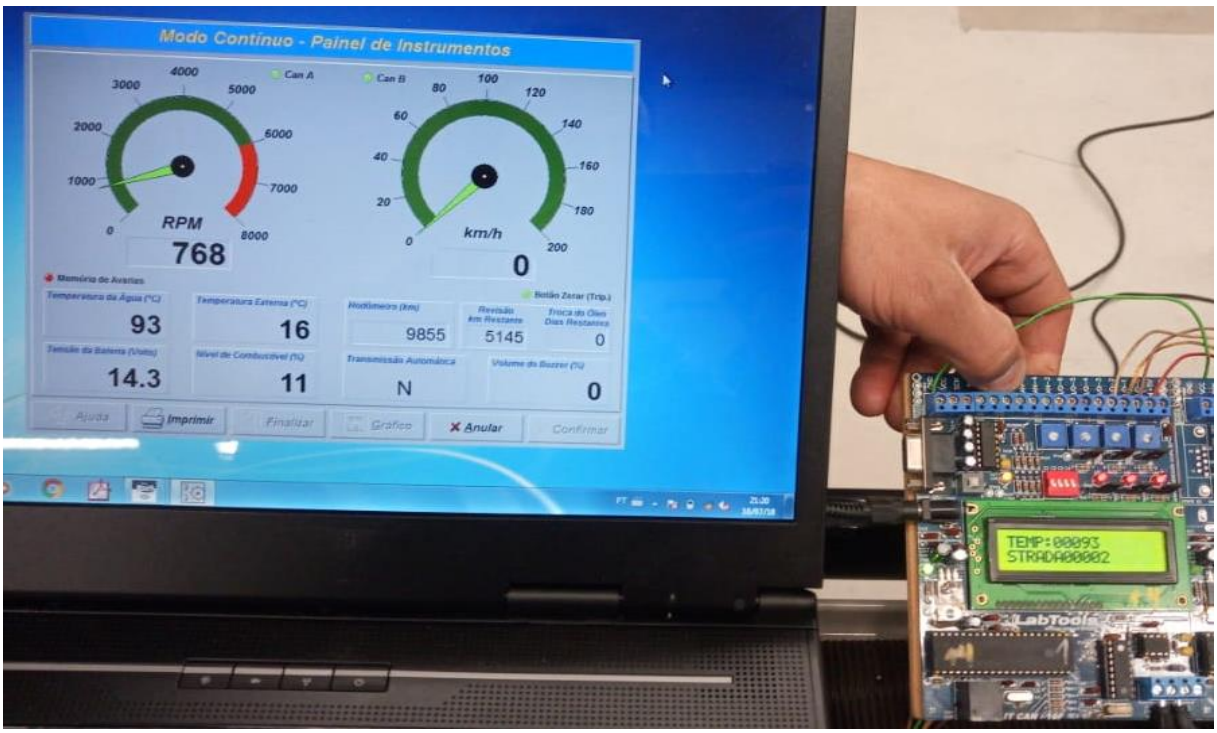
Refizemos o teste, e finalmente conseguimos estabelecer comunicação com o barramento CAN, prosseguimos os testes semelhantes aos do veículo *Volkswagen* Gol, porém no veículo *Fiat* Strada executamos a função de leitura do estado das marchas. Notamos que todas as informações apresentadas no LCD estavam condizentes ao painel do veículo, e checamos todas as leituras, concluindo e validando os testes de *software* no veículo *Fiat* Strada. Obtendo-se assim os resultados desejados validando o projeto.

4.1 Validação dos testes de temperatura do motor

Primeiramente foi necessário utilizar um termômetro digital para a medição da temperatura do motor, a fim de mostrar no LCD a mesma temperatura apresentada no painel do veículo, existe uma diferença entre o painel e o LCD, pois no veículo é dado o valor analógico, e o no LCD é apresentado o valor digital, dado em TEMP:00093, logo, a temperatura do veículo é de 93°C. A fórmula utilizada para calcular o valor é ilustrada pela equação (2).

$$Temperatura = (Byte\ 4) - 40 \quad \textbf{(2)}$$

Figura 14 - Validação dos Testes de Temperatura



Fonte: Autor.

4.2 Validação dos testes de velocidade

Neste teste, precisamos colocar o veículo no dinamômetro para conseguirmos aumentar a velocidade a fim de conferir e verificar se os valores eram semelhantes com o do veículo, a fórmula utilizada para calcular o valor é ilustrada pela equação (3).

$$Velocidade = (Byte\ 8 * 30) + \left[\frac{(Byte\ 7)}{3}\right] \quad \textbf{(3)}$$

A Figura (15) mostra VEL:00055, sendo a velocidade do veículo em 16Km/h, sendo mostrado no painel do veículo em analógico, porém consegue-se estimar que realmente a velocidade é a mesma ou estão bem próximas ao apresentado no LCD.

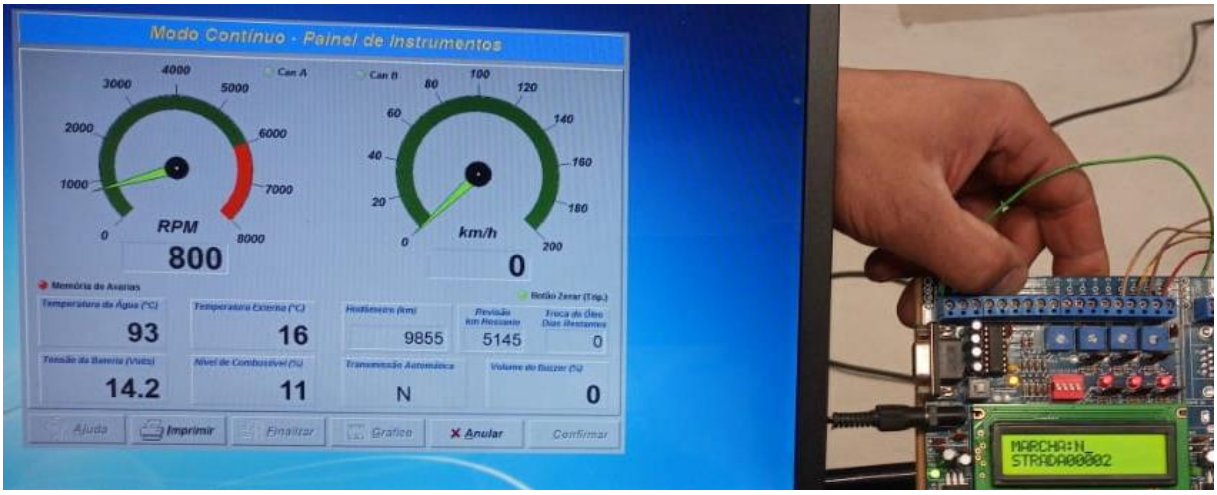
Figura 15 - Validação dos Testes de Velocidade

Fonte: Autor.

4.3 Validação dos testes do estado das marchas

A Figura (16) mostra o estado da marcha em que o veículo se encontra, sendo que neste teste nós precisamos simular as trocas de marchas normalmente para conseguirmos chegando, marcha Neutra. Onde a posição de cada marcha é dada por: 0x1C = Ré, 0x01E = Neutro, 0x02 = 1ª marcha, 0x04 = 2ª marcha, 0x06 = 3ª marcha, 0x08 = 4ª marcha e 0x0A = 5ª marcha.

Figura 16 - Validação dos Testes do Estado das Marcha



Fonte: Autor.

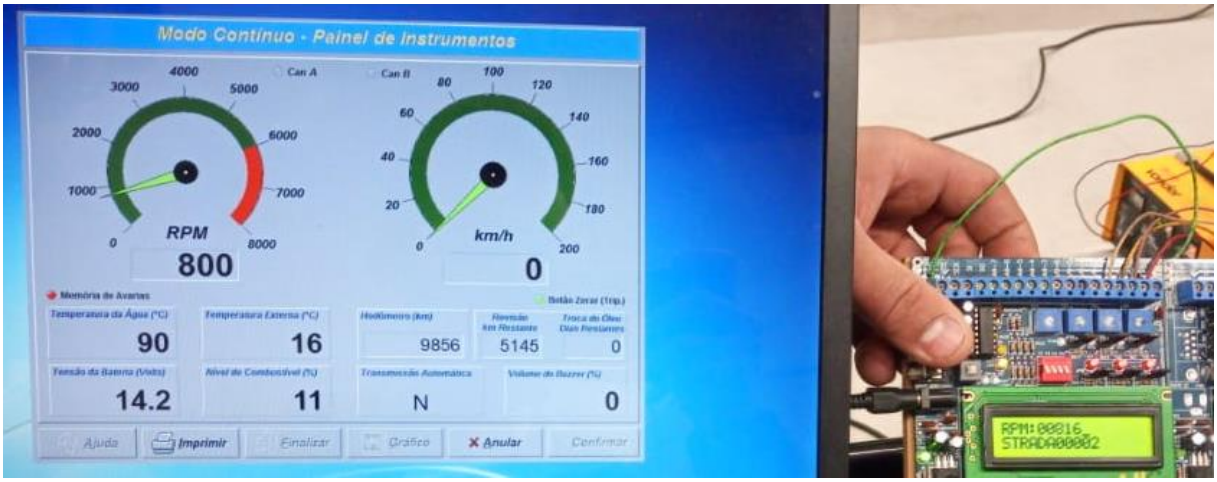
4.4 Validação dos testes de rotação do motor

E por último, mas não menos importante, o teste de RPM do veículo que acreditamos ser o mais difícil, pois para conseguirmos a atualização de dados do nosso projeto com a mesma velocidade em que pisávamos no acelerador, precisávamos manter um certo sincronismo entre nós e manter a rotação estabilizada para conseguirmos verificar se realmente os valores eram plausíveis. Para conseguirmos chegar no valor mostrado no LCD, a fórmula utilizada é ilustrada pela equação (4).

$$RPM = \frac{(Byte\ 1 + Byte\ 2)}{3} \quad \textbf{(4)}$$

Na Figura (16) é mostrado o valor em RPM em marcha lenta, sendo RPM:00816, que nos dá um valor de 800 RPM, este valor obtido foi com o veículo já aquecido. Observa-se em todas as imagens também, que abaixo dos valores mencionados está mostrando o veículo utilizado, que nas imagens é o *Fiat* Strada, caracterizado no LCD como, “STRADA”.

Figura 17 - Validação dos Testes de RPM



Fonte: Autor.

5 Conclusão

O objetivo deste trabalho como proposto era utilizar ferramentas apropriadas para acessar o barramento CAN, suas mensagens, compreende-las e identificar seus significados, sem que tivéssemos qualquer informação do fabricante do veículo ou das características especificas do projeto do barramento. Através de ferramentas de varredura do protocolo, o "*CANALYSTII*", foi possível identificarmos o trafego de dados, os IDs das mensagens e através de observação e dedução implícita, fomos capazes de filtrar no trafego as mensagens relacionadas a variadas funções no veículo de estudo, e compreender a função de cada bit do trafego relacionados aos componentes selecionados, tais como, temperatura, odómetro e velocidade. Assim conseguimos compreender o funcionamento do tráfego de dados a estrutura de prioridades.

Fomos capazes de aplicar o conhecimento adquirido no curso e aprender mais com a utilização de ferramentas para desenvolver o *software* que implementamos para atender a proposta do trabalho realizado.

Todos objetivos foram alcançados com o auxílio dessas ferramentas, que possibilitaram o recolhimento de informações e o ambiente de programação para construção do *software* e a Placa CAN da Fatec sendo o material didático utilizado oferecendo suporte essencial em nosso trabalho. Alcançamos principalmente o objetivo conseguindo o funcionamento e apresentação das mensagens no LCD de forma útil ao profissional de manutenção.

O que nos era totalmente abstrato, depois desta experiência nos parece mais palpável e simples, apesar da complexibilidade do protocolo CAN e suas particularidades. Embora o trabalho pareça singelo, este foi um grande feito pessoal, pois jamais pensávamos em aprender tanto e poder ver que ainda há um horizonte inteiro de aprendizado.

5.1 Propostas futuras

Como propostas futuras é sugerido mudar o hardware utilizado para a placa desenvolvida pelo Prof. Dr. Edson Caoru Kitani, que já inibiria a placa auxiliar com os botões. E transferir as informações da CPU (*Central Process Unit*) para um computador a fim de melhorar interface gráfica das informações. Além de aumentar o número de veículos contemplados.

6 Bibliografia

Belo, V. P. (2003). Sistema para Diagnóstico Automático de Falhas em Véiculos Automotores OBD-2. *Dissertação de Mestrado Universidade Federal de Minas Gerais*.

Bosch, R. (1991). Manueal de Tecnologia Automotiva. *25ª Edição*.

Brennan, S., Buckland, J., & Christen, U. (2007). Especial Issue on Control Applications in Automotive Engineering. *IEEE Transactions on Control Systems Technology, 15*(3), pp. 403-405.

Conama. (15 de Junho de 1989). Resolução 003.

Fredriksson, L. B. (1994). Controller Area Networks and the protocol CAN for machine control systems. *Mechatronics, 4*, pp. 159-192.

Guimarães, A. A. (2001). O Protocolo CAN Bus nas Aplicações Off-Road: Uma Análise Comparativa entre os Padrões Existentes. *SAE World Congress*(2001-01-3853).

Hodel, K. N. (2009). Limites do protocolo CAN (Controller Area Network). *Dissertação de Mestrado Escola Politécnica da Universidade de São Paulo* .

Hodel, K., Specht, S., & Onisic, H. (2002). The On-board Eletronics Innovations and Future Trends based on Customers Experiences. *SAE Word Congress*(2002-01-3376).

Hofstee, J. W., & Goense, D. (1999). Simulation of a Controller Area Network. *Journal of Agricultural Engineering Reserarch, 73*, pp. 383-394.

ISO. (Março de 1998). *ISO 11992-1*. Fonte: Road Vehicles: https://www.iso.org/standard/20661.html

ISO. (Dezembro de 2003). *ISO 11898-2*. Fonte: Road Vehicles: https://www.iso.org/standard/33423.html

ISO. (Junho de 2006). *ISO 11898-3*. Fonte: Road Vehicles: https://www.iso.org/standard/36055.html

Mahmud, S. M., & Alles, S. (2005). In Vehicle Network Architectture for the Next Generation Vehicles. *SAE Word Congress*(2005-01-1531).

Maurici, A. B. (2005). *Lisha*. Acesso em 2018, disponível em UFSC: https://www.lisha.ufsc.br/pub/Maurici_BSC_2005.pdf

Napro. (20 de Maio de 2015). *Suporte Napro*. Fonte: Fera em Tecnologia: www.suporte.napro.com.br/knowledgebase.php?article=51

Nuñez, A. (04 de Junho de 2017). *news.voyage.auto*. Acesso em 2018, disponível em An introduction to the can bus how to programmatically control a car: https://news.voyage.auto/an-introduction-to-the-can-bus-how-to-programmatically-control-a-car-f1b18be4f377

Quigley, C. P., Mcmurrn, R., Jones, R. P., & Faithfull, P. T. (2007). An Investigation into Cost Modeling for Design of Distributed Automotive Electrical Architectures. *3º Institution of Engineering and Technology Conference*, pp. 1-9.

SAE. (14 de Fevereiro de 2000). *SAE J2411*. Fonte: Single Wire Can Network for Vehicle Applications: https://www.sae.org/standards/content/j2411_200002/

APÊNDICE A – Código de Configuração do LCD

```
/*
 * File:   LCD.c
 * Author: adm-lagana
 *
 * Created on 8 de Outubro de 2016, 22:23
 */

#include "LCD.h"
#include <xc.h>
#include "config.h"

/*Envia um comando para o LCD*/
void comando(unsigned char dado){

    RS_LCD=0;
    RW_LCD =0;
    E_LCD=0;
   __delay_us(10);

   D4_LCD = (dado & 0x10)>>4;
   D5_LCD = (dado & 0x20)>>5;
   D6_LCD = (dado & 0x40)>>6;
   D7_LCD = (dado & 0x80)>>7;

   E_LCD=1;
  __delay_us(10);
   E_LCD=0;
  __delay_us(10);

   D4_LCD = (dado & 0x01);
```

```
   D5_LCD = (dado & 0x02)>>1;
   D6_LCD = (dado & 0x04)>>2;
   D7_LCD = (dado & 0x08)>>3;

   E_LCD=1;
  __delay_us(10);
   E_LCD=0;
}

   /*Envia um caracter para o LCD*/
void escrita(unsigned char dado)
{
    RS_LCD=1;
    RW_LCD=0;
    E_LCD=0;
   __delay_us(20);

   D4_LCD = (dado & 0x10)>>4;
   D5_LCD = (dado & 0x20)>>5;
   D6_LCD = (dado & 0x40)>>6;
   D7_LCD = (dado & 0x80)>>7;

   E_LCD=1;
   __delay_us(20);
   E_LCD=0;
   __delay_us(20);

   D4_LCD = (dado & 0x01);
   D5_LCD = (dado & 0x02)>>1;
   D6_LCD = (dado & 0x04)>>2;
   D7_LCD = (dado & 0x08)>>3;
```

```
    E_LCD=1;
   __delay_us(20);
    E_LCD=0;
}
    /*Função de Inicialização do LCD*/
void init_lcd()
{
    __delay_ms(45);

    RS_LCD=0;
    RW_LCD = 0;
    E_LCD=0;
    __delay_us(10);

    D4_LCD = 1;
    D5_LCD = 1;
    D6_LCD = 0;
    D7_LCD = 0;

    E_LCD=1;
    __delay_us(10);
    E_LCD=0;
    __delay_ms(5);
    RS_LCD=0;
    E_LCD=0;
    __delay_us(10);

    D4_LCD = 1;
    D5_LCD = 1;
    D6_LCD = 0;
    D7_LCD = 0;
```

```
 E_LCD=1;
__delay_us(10);
 E_LCD=0;
__delay_us(100);
 RS_LCD=0;
 E_LCD=0;
__delay_us(10);

  D4_LCD = 1;
  D5_LCD = 1;
  D6_LCD = 0;
  D7_LCD = 0;

 E_LCD=1;
__delay_us(10);
 E_LCD=0;
RS_LCD=0;
E_LCD=0;
__delay_us(10);

  D4_LCD = 0;
  D5_LCD = 1;
  D6_LCD = 0;
  D7_LCD = 0;

 E_LCD=1;
__delay_us(10);
 E_LCD=0;

 comando(0x28);
 __delay_us(40);
 comando(0x06);
```

```
   __delay_us(40);
   comando(0x0E);
   __delay_us(40);
   __delay_us(40);
   comando(0x01);
}
  /*Função para Escrita de uma string no LCD */
void escreve_frase(unsigned char *data)
{
    unsigned int i;
    for(i-0; *data!=0; i++)
    {
        escrita(*data);
        data++;
        __delay_us(50);
    }
}
  /*Converte um inteiro com sinal ou sem sinal em uma string que sera escrita no LCD*/

void escreve_inteiro(unsigned int x)
{

    unsigned char vetor_aux[] = "00000";

        unsigned int x_aux = x;
        unsigned char j;



        for(j = 5;j > 0;j--)
        {
            vetor_aux[j-1] = (x_aux % 10) + 0x30;
```

```
            x_aux = x_aux/10;
        }

        escreve_frase(vetor_aux);
        __delay_us(50);
}
```

APÊNDICE B – Código de Configuração dos Valores Recebidas do Veículo

```
/*
 * File: painel.c
 * Author: WESLEI e Paulo hayashida
 *
 * Created on 2 de Junho de 2018, 16:26
 */

#include "config.h"

void msg_mang(void)
{
    acc_fil1 = spi_leitura_mcp (0x60);
    acc_fil1 = acc_fil1 & 0x01;
    acc_fil2 = spi_leitura_mcp (0x70);
    acc_fil2 = acc_fil2 & 0x07;

    switch(acc_fil1)
    {
        case 0:


            //Leitura
            raw_rpm_strada0 = spi_leitura_mcp (0x6B);
            raw_rpm_strada1 = spi_leitura_mcp (0x6C);
            raw_rpm_strada0 = raw_rpm_strada0;
```

```
        raw_rpm_strada1 = raw_rpm_strada1 << 8;
        raw_rpm_strada2 = raw_rpm_strada0 + raw_rpm_strada1;
        rpm_strada = raw_rpm_strada2 >> 3;

       raw_temp_mot_strada = spi_leitura_mcp (0x69);
       temp_mot_strada = raw_temp_mot_strada - 40;



        spi_escrita_mcp (0x2C, 0x00); // Limpa flag int mcp
        //data = spi_leitura_mcp (0x6A);
        break;

    case 1:

        //Leitura
        raw_vel_strada0 = spi_leitura_mcp (0x66);
        raw_vel_strada1 = spi_leitura_mcp (0x67);
        spi_escrita_mcp (0x2C, 0x00); // Limpa flag int mcp



        raw_vel_strada0 = raw_vel_strada0 * 30;
        raw_vel_strada1 = raw_vel_strada1 >> 3;

        vel_strada = raw_vel_strada0 + raw_vel_strada1;
        break;

}
```

```
switch(acc_fil2)
{
   case 2:

       //Leitura
        raw_marcha_strada = spi_leitura_mcp (0x78);
        spi_escrita_mcp (0x2C, 0x00); // Limpa flag int mcp



        //Conversão
        if(raw_marcha_strada == 0x1C)
        {

          marcha_strada = 'R';
        }
        else if(raw_marcha_strada == 0x02)
        {
            marcha_strada = '1';
        }
        else if(raw_marcha_strada == 0x04)
        {
            marcha_strada = '2';
        }
        else if(raw_marcha_strada == 0x06)
        {
            marcha_strada = '3';
        }
        else if(raw_marcha_strada == 0x08)
```

```
    {
        marcha_strada = '4';
    }
    else if(raw_marcha_strada == 0x0A)
    {
        marcha_strada = '5';
    }
    else if(raw_marcha_strada == 0x1E)
    {
        marcha_strada = 'N';
    }
    break;


case 3:

    //Leitura
    raw_rpm_gol0 = spi_leitura_mcp (0x78);
    raw_rpm_gol1 = spi_leitura_mcp (0x79);

    raw_pedal_gol = spi_leitura_mcp (0x7B);;
    pedal_gol = raw_pedal_gol/2.5;

    spi_escrita_mcp (0x2C, 0x00); // Limpa flag int mcp

    //Conversão
    raw_rpm_gol1 = raw_rpm_gol1 << 8;
    raw_rpm_gol0 = raw_rpm_gol0 + raw_rpm_gol1;
    raw_rpm_gol0 = raw_rpm_gol0 >> 2;
```

```
            rpm_gol = raw_rpm_gol0;
            break;

        case 4:

            //Leitura
            raw_temp_mot_gol = spi_leitura_mcp (0x77);
            spi_escrita_mcp (0x2C, 0x00); // Limpa flag int mcp

            //Conversão
            raw_temp_mot_gol =  raw_temp_mot_gol - 75;
            temp_mot_gol =  raw_temp_mot_gol;
            break;

        case 5:
            //Leitura
            raw_vel_gol = spi_leitura_mcp (0x7C);



            //Conversão
            raw_vel_gol = raw_vel_gol << 2;
            vel_gol = raw_vel_gol/3;
            break;
    }

}
```

```
void painel(unsigned char dado)
{

    if(CHAVE_1 == 0)
    {
         comando(0xC0);
         __delay_us(10);
         escreve_frase("STRADA");
         escreve_inteiro(acc_fil2);

          //escrita(0x30);
        if(CHAVE_2 == 0 & CHAVE_3 == 0)
        {
            comando(0x80);
            __delay_us(10);
            escreve_frase("RPM:");
            escreve_inteiro(rpm_strada);
        }
        else if(CHAVE_2 == 1 & CHAVE_3 == 0)
        {
            comando(0x80);
            __delay_us(10);
            escreve_frase("VEL:");
            escreve_inteiro(vel_strada);

        }
        else if(CHAVE_2 == 0 & CHAVE_3 == 1)
```

```
    {
        comando(0x80);
        __delay_us(10);
        escreve_frase("TEMP:");
        escreve_inteiro(temp_mot_strada);
    }
     else if(CHAVE_2 == 1 & CHAVE_3 == 1)
    {
        comando(0x80);
        __delay_us(10);
        escreve_frase("MARCHA:");
        escrita(marcha_strada);

    }
}
else
{
 comando(0xC0);
 __delay_us(10);
 escreve_frase("GOL");
 escreve_inteiro(acc_fil1);
  //escrita(0x30);
    if(CHAVE_2 == 0 & CHAVE_3 == 0)
    {
        comando(0x80);
        __delay_us(10);
        escreve_frase("RPM:");
        escreve_inteiro(rpm_gol);
    }
```

```
        else if(CHAVE_2 == 1 & CHAVE_3 == 0)
        {
            comando(0x80);
            __delay_us(10);
            escreve_frase("VEL:");
            escreve_inteiro(vel_gol);
        }
        else if(CHAVE_2 == 0 & CHAVE_3 == 1)
        {
            comando(0x80);
            __delay_us(10);
            escreve_frase("TEMP:");
            escreve_inteiro(temp_mot_gol);
        }
        else if(CHAVE_2 == 1 & CHAVE_3 == 1)
        {
            comando(0x80);
            __delay_us(10);
            escreve_frase("PEDAL:");
            escreve_inteiro(pedal_gol);
        }

    }

}
```

APÊNDICE C – Código de Configuração da Comunicação

```
#include "config.h"
void mcp_reset()
{
   PORTBbits.RB2 = 0;  //Coloca CS do MCP2515 para 0, habilita comunicação
      SSPBUF = 0xC0;
      PORTBbits.RB2 = 1;  //Coloca CS do MCP2515 para 1, finaliza comunicação
}
void spi_escrita_mcp (unsigned char end, unsigned char dado)
{
   PORTBbits.RB2 = 0;  //Coloca CS do MCP2515 para 0, habilita comunicação
   SSPBUF = 0x02; // Comando de escrita MCP2515
   while(!SSPSTATbits.BF);
   SSPBUF = end; // Endereço interno do MCP2515
   while(!SSPSTATbits.BF);
   SSPBUF = dado;  //Dado a ser escrito no MPC2515
   while(!SSPSTATbits.BF);
   PORTBbits.RB2 = 1;  //Coloca CS do MCP2515 para 1, finaliza comunicação
}
unsigned char spi_leitura_mcp (unsigned char end)
{
   unsigned char dado_recebido;
   PORTBbits.RB2 = 0;  //Coloca CS do MCP2515 para 0, habilita comunicação
   SSPBUF = 0x03; // Comando de leitura MCP2515
   while(!SSPSTATbits.BF);
   SSPBUF = end; // Endereço interno do MCP2515
   while(!SSPSTATbits.BF);
   SSPBUF = 0x00;
   while(!SSPSTATbits.BF);
```

```
    dado_recebido = SSPBUF;
    PORTBbits.RB2 = 1;  //Coloca CS do MCP2515 para 0, habilita comunicação
    return dado_recebido;
}
void conf_spi()
{
    TRISA = 0xFF;
    TRISB = 0x00;
    TRISC = 0x00;
    TRISCbits.TRISC4 = 1;
    TRISD = 0x00;
    TRISE = 0x00;
    ADCON0 = 0x00;
    ADCON1 = 0b00000110;

    SSPCONbits.SSPEN  = 0;
    SSPCONbits.SSPM0  = 1;
    SSPCONbits.SSPM1  = 0;
    SSPCONbits.SSPM2  = 0;
    SSPCONbits.SSPM3  = 0;
    SSPCONbits.CKP = 0;
    SSPSTATbits.SMP = 0;
    SSPSTATbits.CKE = 1;
    SSPIF = 0;
    SSPCONbits.SSPEN  = 1;

     mcp_reset();
}
```

APÊNDICE D – Código Principal

```
#include <xc.h>
#include  "config.h"

unsigned char x = 0;
unsigned char y = 0;
unsigned char t_lcd = 0;
unsigned char data;

void interrupt isr()
{
   if(PIE1bits.TMR1IE && PIR1bits.TMR1IF)
   {
      PIR1bits.TMR1IF = 0;
      x++;
      t_lcd++;

      if(x == 10)
      {
         y = 1;
         x = 0;
      }

      if(t_lcd == 30)
      {
         comando(0x01);
         t_lcd = 0;
      }

      TMR1L = 0;
```

```
        TMR1L = 0;
    }


}

/*funçao de inicializaçao do hardware do microcontrolador*/
void int_hw()
{
    /*conFigura as portasdo microcontrolador*/
    TRISA = 0x00;
    TRISB = 0xFF;
    TRISC = 0xFF;
    TRISD = 0x00;
    TRISE = 0x00;
    T1CON = 0b00110001;
    INTCONbits.PEIE = 1;
    INTCONbits.GIE = 1;
    PIE1bits.TMR1IE = 1;

    ADCON0bits.ADON = 0;
    /*DESLIGA CONVERSOR A/D E CONFIGURA TODAS AS PORTAS COMO DIGITAIS*/
    ADCON1=0b00000110;
}
void main()
{
    int_hw();
    conf_spi();
    init_lcd();

    mcp_reset();
```

```
spi_escrita_mcp (0x0F, 0x80); // Modo Conf


spi_escrita_mcp (0x2A, 0x09); // CNF1
spi_escrita_mcp (0x29, 0xBA); // CNF2
spi_escrita_mcp (0x28, 0x07); //CNF3

spi_escrita_mcp (0x00, 0x50); //Filtro 1 ---- RPM Strada --- Temp do Motor
spi_escrita_mcp (0x01, 0x20); //Filtro 1

spi_escrita_mcp (0x04, 0x54); //Filtro 2 ----- Velocidade Strada
spi_escrita_mcp (0x05, 0x00); //Filtro 2

spi_escrita_mcp (0x08, 0x51); // Filtro 3 ---- Marcha Strada
spi_escrita_mcp (0x09, 0x60); // Filtro 3

spi_escrita_mcp (0x10, 0x50); // Filtro 4 -- RPM Gol
spi_escrita_mcp (0x11, 0x00); // Filtro 4 --

spi_escrita_mcp (0x14, 0x51); // Filtro 5 -- Temp motor Gol
spi_escrita_mcp (0x15, 0x00); // Filtro 5 --

spi_escrita_mcp (0x18, 0x64); // Filtro 6 -- Vel Gol
spi_escrita_mcp (0x19, 0x00); // Filtro 6 --

spi_escrita_mcp (0x20, 0xFF); //CNF3
spi_escrita_mcp (0x21, 0xE0); //CNF3

spi_escrita_mcp (0x24, 0xFF); //CNF3
spi_escrita_mcp (0x25, 0xE0); //CNF3
```

```
spi_escrita_mcp (0x0F, 0x00); // Modo Normal, vide ds pag 58

spi_escrita_mcp (0x60, 0x00); // liga mascara e filtro R Buffer 0
spi_escrita_mcp (0x70, 0x00); // liga mascara e filtro R Buffer 1



spi_escrita_mcp (0x31, 0xAA); // ID do buffer de Tx0
spi_escrita_mcp (0x32, 0xA0);

spi_escrita_mcp (0x35, 0x08); // DLC do buffer Tx0

spi_escrita_mcp (0x36, 0x03); // Data 1 dp buffer Tx0
spi_escrita_mcp (0x37, 0x52);
spi_escrita_mcp (0x38, 0x25);
spi_escrita_mcp (0x39, 0x12);
spi_escrita_mcp (0x3A, 0x06);
spi_escrita_mcp (0x3B, 0x33);
spi_escrita_mcp (0x3C, 0x45);
spi_escrita_mcp (0x3D, 0x53); // Data 8 dp buffer Tx0

spi_escrita_mcp (0x30, 0x03); // Controle do Buffer Tx0

spi_escrita_mcp (0x2B, 0x00); // Desliga INt do mcp

spi_escrita_mcp (0x2C, 0x00); // Limpa flag int mcp

while(1)
{
  PORTAbits.RA5 = !PORTAbits.RA5;
```

```
      __delay_ms(50);
     // spi_escrita_mcp (0x30, 0x0F); // Controle do Buffer Tx0
      if(y == 1)
      {
          PIE1bits.TMR1IE = 0;
          painel(data);
          y = 0;
          //data = spi_leitura_mcp (0x6D);
          msg_mang();
          //spi_escrita_mcp (0x2C, 0x00); // Limpa flag int mcp
          PIE1bits.TMR1IE = 1;
      }
    }
}
```