Universidade de Brasília - UnB
Faculdade UnB Gama - FGA
Curso de Engenharia Eletrônica

PLATAFORMA DIDÁTICA PARA ESTUDO DE REDES AUTOMOTIVAS

Autor: Paulo Augusto Miguel Fonseca de Souza
Orientador: Prof.Dr.Evandro Leonardo Silva Teixeira

Brasília, DF
2019

PAULO AUGUSTO MIGUEL FONSECA DE SOUZA

PLATAFORMA DIDÁTICA PARA ESTUDO DE REDES AUTOMOTIVAS

Monografia submetida ao curso de graduação em Engenharia Eletrônica da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharelem Engenharia Eletrônica.

Orientador: Prof. Dr. Evandro Leonardo Silva Teixeira

Brasília, DF
2019

CIP – Catalogação Internacional da Publicação*

Souza, Paulo Augusto Miguel Fonseca de.
Plataforma Didática para Estudo de Redes Automotivas/ Paulo Augusto Miguel Fonseca de Souza Brasília: UnB, 2019. 92 p. : il. ; 29,5 cm.

Monografia (Graduação) – Universidade de Brasília Faculdade do Gama, Brasília, 2019. Orientação: Evandro Leonardo Silva Teixeira.

1. CAN. 2. Redes automotivas. 3. Arquiteturas Eletroeletrônicas. 4. Model-Based Design I. Teixeira, Evandro Leonardo Silva. II. Bancada Didática para Simulação De redes Automotivas.

CDU Classificação

PLATAFORMA DIDÁTICA PARA ESTUDO DE REDES AUTOMOTIVAS

Paulo Augusto Miguel Fonseca de Souza

Monografia submetida como requisito parcial para obtenção do Título de Bacharel em Engenharia Eletrônica da Faculdade UnB Gama - FGA, da Universidade de Brasília, em 12/07/2019 apresentada e aprovada pela banca examinadora abaixo assinada:

Prof. Dr.: Evandro Leonardo Silva Teixeira, UnB/ FGA
Orientador

Prof. Dr.: Diogo Caetano Garcia, UnB/ FGA
Membro Convidado

Prof. Dr.: André Murilo de Almeida Pinto, UnB/ FGA
Membro Convidado

Brasília, DF
2019

Esse trabalho é dedicado a todos amantes da ciência que estão sempre em busca do progresso e de inovações.

AGRADECIMENTOS

Agradeço toda ajuda que recebi durante todo esse tempo na Universidade. Primeiramente a Deus, minha família e a minha esposa que sempre me incentivaram. E ao professor Evandro por sua orientação.

Uma mente que se abre a uma nova ideia, jamais retorna ao seu tamanho original.
Einstein.

RESUMO

Atualmente na indústria automotiva, vem sendo feito cada vez mais investimentos em eletroeletrônica veicular. Isso foi impulsionado principalmente pelos desejos de mais economia, conforto, comodidade e segurança. Com isso, há vários módulos eletrônicos em um automóvel, e por serem muitas vezes desenvolvidos de maneira independentes, há a necessidade de comunicação entre os sistemas, pois muitas são necessárias a trocas de dados. Com essa dependência e com a necessidade de se comunicarem, fez com que a indústria automotiva desenvolvesse meios de comunicação. Entre esses meios, se pode destacar o protocolo de comunicação CAN. Contudo esses sistemas se tornam cada vez mais complexos de serem desenvolvidos, e por isso tem se difundido o uso do MBD. A partir disso, esse trabalho apresenta uma plataforma, uma solução, que auxilia no desenvolvimento de projetos de arquiteturas eletroeletrônicas envolvendo as redes de comunicação CAN junto com o MBD. A partir dessa solução foram feitos cinco estudos de casos, com diferentes graus de dificuldade, para validarem a funcionalidade e a viabilidade da plataforma. Por fim, são apresentados os resultados dos estudos de caso desenvolvidos, e com isso, foi verificado que como a plataforma possui limitações, o ultimo estudo teve seu desempenho afetado, o que gerou maiores tempos de transmissão de dados.

Palavras-chave: CAN. Redes Automotivas. Arquiteturas eletroeletrônicas. Model-Based Design

ABSTRACT

Currently, in the automotive industry, more and more investments in vehicle electronics have been made. This was driven mainly by the desires of more economy, comfort, comfort and safety. With this, there are several electronic modules in an automobile, and because they are often developed in an independent manner, there is a need for communication between systems, since many are necessary for data exchange. With this dependence and the need to communicate, the automotive industry developed the means of communication. Among these means, the CAN communication protocol can be highlighted. However, these systems become increasingly complex to develop, and therefore the use of MBD has become widespread. From this, this work presents a platform, a solution, that assists in the development of projects of electro-electronic architectures involving CAN communication networks along with MBD. From this solution, five case studies with different degrees of difficulty were made to validate the functionality and viability of the platform. Finally, the results of the developed case studies are presented, and with this, it was verified that since the platform has limitations, the last study had its performance affected, which generated larger times of data transmission.

Keywords: CAN. Automotive Network. Electrical and electronic architectures. Model-Based Design

LISTA DE ILUSTRAÇÕES

Figura 1: Etapas de Estudo 16
Figura 2: Proporção de eletrônica em um veículo (Bosch GmbH, 2007) 18
Figura 3: Crescimento ao longo do tempo de sistemas eletroeletrônicos nos veículos (Hallestrand, 2005) 19
Figura 4: Módulos eletrônicos presentes em um veículo (Bosch GmbH, 2007) 19
Figura 5: Exemplo de uma ECU 20
Figura 6: Diagrama de blocos de uma ECU 21
Figura 7: Arquitetura Centralizada 21
Figura 8: Exemplo Arquitetura Centralizada 22
Figura 9: Arquitetura Distribuída 22
Figura 10: Exemplo Arquitetura Distribuída 23
Figura 11: Crescimento da quantidade de microcontroladores em um veículo 24
Figura 12: Arbitragem dos bits (Bosch GmbH, 2007) 26
Figura 13: Sistemas de transmissão de dados (Bosch GmbH, 2007) 26
Figura 14: Relação entre comprimento da rede e taxa de transmissão (Guimarães & Saraiva, 2002) 27
Figura 15: Formato CAN 2.0A (Paret, 2005) 28
Figura 16: Formato CAN 2.0B (Paret, 2005) 28
Figura 17: Fluxo de trabalho do MBD 32
Figura 18: Diagrama de modelo V 33
Figura 19: Modelo geral da plataforma 35
Figura 20: Modelo Geral Hardware da Plataforma 36
Figura 21: Arduino Nano 36
Figura 22: Módulo MCP2515 37
Figura 23: Potenciômetro e "*Push Button*" 37
Figura 24: Sinal PWM 38
Figura 25: Filtro Passa Baixa 38
Figura 26: Diagrama de Bode Filtro Passa Baixa 39
Figura 27: Mini-Protoboard 39
Figura 28: Barra de pinos 40
Figura 29: Plataforma em modelo Digital (Proteus) 40
Figura 30: Plataforma Construída Fisicamente 41
Figura 31: Dispositivo CAN Ligado ao Matlab/Simulink 42
Figura 32: Blocos de programação "Vehicle Network Toolbox" 42
Figura 33: Blocos de programação do Arduino 43
Figura 34: Bloco S-Function Builder 44
Figura 35: Caixa de Diálogo S-Function Builder 44
Figura 36: Geração dos blocos Customizados 45
Figura 37: Fluxograma de Transmissão (E), Modelo Simulink de Transmissão (D) 45
Figura 38: Fluxograma de recepção (E), Modelo Simulink de Recepção (D) 46
Figura 39:Fluxograma para gerar o modelo de transmissão e recepção 46
Figura 40: Modelo de transmissão e recepção no Simulink 46
Figura 41: Modos Execução Arduino 47
Figura 42: Exemplo do funcionamento de um sensor de rotação 48
Figura 43: Sensor de Rotação em um Veículo 48
Figura 44: Modelo Caso 01 49
Figura 45: Onda enviada (E) e onda recebida (D) 50
Figura 46: Monitor Serial Mensagens Recebidas Caso 01 50

Figura 47: Exemplo de sensores presentes em um veículo 51
Figura 48: Modelo Caso 02 51
Figura 49: Execução Externa: ECU1 ID356 e ECU2 ID 356 52
Figura 50: Comparação Entrada (E) e Saída ID 550 (D) 53
Figura 51: Comparação Entrada (E) e Saída ID 356 (D) 53
Figura 52: Monitor Serial do Segundo Estudo 54
Figura 53: Tela de diagnóstico de um Scanner Automotivo 54
Figura 54: Modelo Caso 04 55
Figura 55: Estudo Verificação de Erro na Placa 55
Figura 56: Aplicativo Estudo 03 57
Figura 57: Diagrama verificação e monitoramento sensores 58
Figura 58: Diagrama de verificar ou limpar memória 58
Figura 59: Diagrama Base para a ECU4 59
Figura 60: Scanner conectado a placa 59
Figura 61: Memória de avarias Limpa 60
Figura 62: Visualização dos erros gerados de forma progressiva 60
Figura 63: Visualização de todos os erros 60
Figura 64: Monitor Serial: Memória Limpa (E) e Presença de Erros (D) 61
Figura 65: Fluxograma de ligação entre as duas ECUs 61
Figura 66: Especificação Física da temperatura do motor para implementação 62
Figura 67: Aplicativo Estudo 04 63
Figura 68: Fluxograma ECU01 64
Figura 69: Fluxograma 1 para a ECU02 64
Figura 70: Fluxograma 2 para a ECU02 65
Figura 71: Telas Aplicativo com diferentes valores de Quantização e Offset 65
Figura 72: Monitor Serial para os diferentes Valores de Quantização e Offset 66
Figura 73: Visualização dos Valores de Calibração 66
Figura 74: Monitor Serial Mensagens 66
Figura 75: Forças que agem sobre um veículo 67
Figura 76: Modelo Último Estudo 67
Figura 77: Aplicação Último Estudo 68
Figura 78: Diagrama Funcionamento SEM 70
Figura 79: Diagrama modelo da GMS 72
Figura 80: Diagrama modelo da BMS 73
Figura 81: Modelo ICL 73
Figura 82: Painel Virtual 74
Figura 83: Início aquisição de dados 74
Figura 84: Aquisição após ativação 75
Figura 85: Aquisição após inicio movimento (E) e após certo tempo de execução (D) 75
Figura 86: Monitor Serial Mensagens do Ultimo Estudo 77

LISTA DE TABELAS

Tabela 1: Relação de alguns protocolos existentes e suas características (Bosch GmbH, 2007) 24
Tabela 2: Dicionário de dados Estudo de Caso 01 49
Tabela 3: Estatísticas dos resultados do Estudo de Caso 01 50
Tabela 4: Dicionário de dados Estudo de Caso 02 52
Tabela 5: Dados Estatísticos do Segundo Estudo 54
Tabela 6: Relação DTC, Sensores, Tipo de Erro e ECU 56
Tabela 7: Dicionário Dados Quadros Remotos 56
Tabela 8: Dicionário de dados das mensagens com os erros 57
Tabela 9: Tabela para Calibração e Conversão de Unidades 62
Tabela 10: Dicionário de dados Estudo de Caso 04 63
Tabela 11: Dicionário de dados do Último Estudo de Caso 69
Tabela 12: Dados Estatísticos do Último Estudo 76

LISTA DE ABREVIATURAS E SIGLAS

ACK – *ACKnowledgement*
AD – Analógico – Digital
BMS – *Brake Management System*
CAN – *Controller Area Network*
CI – Circuito Integrado
CRC – Verificação de redundância Cíclica (*Cyclic Redundancy Checksum*)
DLC – Código de tamanho de dados (*Data length Code*)
DTC - *Diagnostic Trouble Code*
ECU – *Electronic Control Unit* (Unidade de Controle Eletrônico)
EMS – *Engine Management System*
GMS – *Gear Management System*
HID – Endereço de hardware (Hardware ID)
Hz – Hertz
ICL – *Istrument Cluster*
Kbps – kilo bits por segundo
MBD – *Model-Based Design*
ms - milissegundos
PWM – *Pulse Width Modulation*
RTR – Requisição de Transmissão Remota (*Remote Transmission Request*)
Rx - Receptor
S - segundo
SAE – *Society of Automotive Engineers*
SPI – *Serial Peripheral Interface*
TI – Tensão Inferior
TS – Tensão Superior
Tx - Transmissor
V - Volts
VNT – *Vehicle Network Toolbox*

SUMÁRIO

1 INTRODUÇÃO 15
1.1 OBJETIVOS 15
1.2 ESTRUTURA DO TRABALHO 16
1.3 METODOLOGIA 16
2 REVISÃO BIBLIOGRÁFICA 18
2.1 A IMPORTÂNCIA DA ELETROELETRÔNICA NA INDÚSTRIA AUTOMOTIVA 18
2.2 COMPONENTES ELETROELETRÔNICOS 19
2.3 ARQUITETURA ELETROELETRÔNICA 21
2.3.1 Arquitetura Centralizada 21
2.3.2 Arquitetura Distribuída 22
2.4 REDES AUTOMOTIVAS 23
2.4.1 *Controller Area Network* (CAN) 24
2.5 IMPLEMENTAÇÃO DE UMA REDE DE COMUNICAÇÃO CAN BUS 30
2.6 *PROJETO BASEADO EM MODELOS (MODEL-BASED DESIGN)* 31
3 PROJETO DA PLATAFORMA 35
3.1 SOLUÇÃO DE HARDWARE 35
3.2 SOLUÇÃO DE SOFTWARE 41
3.2.1 Módulos customizados 44
4 ESTUDOS DE CASO 48
4.1 ESTUDO DE CASO 01 48
4.1.1 Projeto 49
4.1.2 Análise dos Resultados 49
4.2 ESTUDO DE CASO 02 51
4.2.1 Projeto 51
4.2.2 Análise dos Resultados 52
4.3 ESTUDO DE CASO 03 54
4.3.1 Projeto 55
4.3.2 Análise dos Resultados 59
4.4 ESTUDO DE CASO 04 61
4.4.1 Projeto 62
4.4.2 Análise dos Resultados 65
4.5 ESTUDO DE CASO 05 67
4.4.1 Projeto 68
4.4.2 Análise dos Resultados 73
5 CONCLUSÃO 78
Bibliografia 80

1 INTRODUÇÃO

O uso de eletrônica em um automóvel tem crescido de maneira constante ao longo do tempo e continuará em crescimento no futuro. O sucesso dos veículos passa pela contínua inovação produzida pela indústria automotiva. Com isso, as novas funções aparecem em conjunto com sistemas de auxilio ao motorista, que proporciona mais segurança, conforto, comodidade e economia (Bosch GmbH, 2007).

Com esse constante crescimento de sistemas eletroeletrônicos em um veículo houve a troca de sistemas hidráulicos e mecânicos por eletrônicos. Com isso, inúmeros componentes foram adicionados ao projeto de um automóvel, fazendo com que aumentasse a complexidade dos sistemas em um veículo. (Bosch GmbH, 2007)

Grandes partes desses sistemas são desenvolvidos de forma individual, com isso, naturalmente surgiu à necessidade da troca de informações entre eles. Isso fez com fosse necessário a utilização de barramentos e protocolos de comunicação, dentre os quais se tem o *Controller Area Network* (CAN). CAN é um protocolo de comunicação serial que suporta com eficiência o controle de aplicações em tempo real com um alto nível de segurança, e foi desenvolvido na década de 1980 pela Bosch. Esse sistema de comunicação serial foi aplicado com sucesso em muitos campos devido a sua alta confiabilidade e eficiência, como por exemplo, na indústria automotiva e agrícola (Wu & Chung, 2014).

Contudo, os projetos dessas arquiteturas eletroeletrônicas envolvidos com redes de comunicação, se tornam cada vez mais complexos, o que pode gerar maiores custos e tempo de execução. Por isso, tem se difundido cada vez mais o uso de projetos baseados em modelos, MBD, em que consiste na criação de modelos computacionais. Essa criação de modelos permite que a indústria obtenha melhores resultados relacionados principalmente a custos e tempo de desenvolvimento de um projeto.

Por isso, o desenvolvimento de uma plataforma didática, para o desenvolvimento de projetos que envolvam arquiteturas eletroeletrônicas automotivas com rede CAN, permite melhor entendimento e compreensão acerca dos sistemas eletroeletrônicos automotivos. Por fim, o uso de uma plataforma didática visa menores custos e o ensino de maneira prática dos projetos que envolvam redes de comunicação automotivas.

1.1 OBJETIVOS

O objetivo principal deste trabalho é desenvolver uma plataforma de desenvolvimento que permite o estudo e melhor compreensão sobre arquiteturas eletroeletrônicas automotivas com o uso do protocolo de comunicação CAN. A partir desse objetivo inicial, serão desenvolvidos projetos em que permitam diferentes estudos e formas de uma arquitetura eletroeletrônica presente em um automóvel real. Também tem como objetivo aplicar os conceitos de MBD com o uso do

Matlab/Simulink para auxiliar o desenvolvimento de um projeto de redes de comunicação automotiva.

1.2 ESTRUTURA DO TRABALHO

A estrutura desse trabalho está dividida em cinco capítulos. O segundo capítulo apresenta a revisão bibliográfica com os principais conceitos abordados no trabalho. O terceiro apresenta o projeto da plataforma didática, tanto a sua solução em hardware quanto em software. No quarto são apresentados os estudos de caso desenvolvidos e as análises dos resultados de cada estudo. Por fim, no quinto é apresentado às conclusões a respeito do trabalho.

1.3 METODOLOGIA

Após a definição da estrutura do trabalho, nessa parte será apresentado a Metodologia utilizada para o desenvolvimento. Essa metodologia pode ser evidenciada na Figura 1.

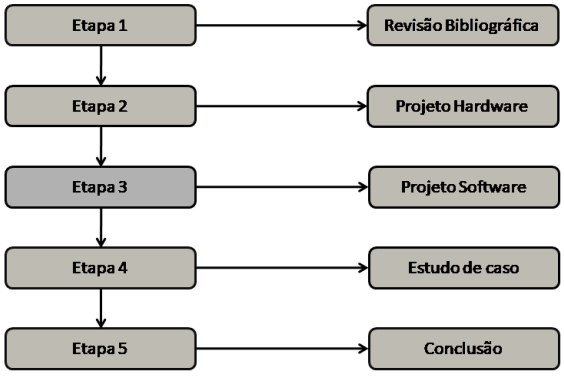

Figura 1: Etapas de Estudo

A primeira etapa, Revisão Bibliográfica, é feito um estudo sobre a eletroeletrônica automotiva e a sua importância dentro da indústria. Com o objetivo de explicar os conceitos sobre a eletrônica embarcada automotiva, trazendo principalmente o conceito de redes de comunicação, com ênfase no protocolo de comunicações CAN. Além desses conceitos, apresenta MBD e como aplicá-lo.

A segunda etapa consiste em apresentar a estrutura de hardware desenvolvida para a plataforma. Nesta parte são definidos o tamanho, forma, quais e quantos componentes são necessários para a sua produção. A partir da definição foi possível construir fisicamente plataforma para o estudo de arquiteturas eletroeletrônicas.

A terceira etapa é apresentada a solução de software desenvolvida para a programação da placa de desenvolvimento. Esse processo foi realizado utilizando o

Matlab/Simulink. O Simulink possui pacotes para programação do Arduino e para comunicação CAN. Com o uso desses recursos, foi possível simular em cada microcontrolador uma ECU.

A quarta etapa consiste em apresentar os estudos de caso. Eles servem para testar e validar a funcionalidade da plataforma, como desenvolver projetos de arquiteturas eletroeletrônicas com o protocolo de comunicação CAN. Também é estudado aqui como é feita a comunicação entre as ECU's e o computador, quais as mensagens serão mostradas para que seja feito a análise se a rede de comunicação está com comportamento adequado. Após os desenvolvimentos de cada estudo, são apresentados os resultados de cada um, para saber se foram corretamente interpretados.

A quinta etapa apresenta a conclusão sobre o trabalho. Alem de finalizar o trabalho, indicando se a plataforma alcançou seu objetivo, traz ideias para trabalhos futuros utilizando a plataforma e como ela poderá ser aplicada. E por fim analisando a sua utilidade para o curso de engenharia da universidade.

2 REVISÃO BIBLIOGRÁFICA

Neste capítulo serão apresentados os principais conceitos abordados no decorrer do trabalho. Começando como a eletroeletrônica influencia de maneira ativa os sistemas da indústria automotiva, avançando para os conceitos de redes automotivas e o protocolo de comunicação CAN, e os conceitos de MBD.

2.1 A IMPORTÂNCIA DA ELETROELETRÔNICA NA INDÚSTRIA AUTOMOTIVA

Inicialmente a indústria automotiva se caracterizou pela utilização de componentes mecânicos e hidráulicos. Ao longo do tempo, o progresso na indústria automotiva se caracterizou pela adição de sistemas eletroeletrônicos em seus projetos. Muitos desses progressos foram feitos principalmente pela busca por mais conforto, segurança e a temas relacionados à redução de emissão de poluentes. (Bosch; 2007).

Com a adição dos sistemas eletroeletrônicos a maioria das inovações será feita a partir do desenvolvimento de sistemas embarcados automotivos, em que 90% dessas inovações serão na área de eletrônica, e desses, 80% serão em software. Com esse desenvolvimento estima-se que 35% dos custos totais de um veículo virão da aplicação de sistemas eletroeletrônicos. Isso faz com que os custos relacionados à implementação desses sistemas sejam triplicados, enquanto áreas de potência e estruturas devem ter pequenos incrementos durante a produção de um automóvel (Schätz & Krcmar, 2011).

A Figura 2 mostra um comparativo entre os anos de 2000 e 2010, como ao passar houve um aumento dos sistemas eletroeletrônicos em um automóvel. Esse crescimento pode ser percebido principalmente pelo incremento de tecnologia microeletrônica e de sensores. O uso de sistemas eletrônicos teve incremento em todas as partes, e principalmente na área de navegação e comunicação (Bosch GmbH, 2007). Esse crescimento pode ser visto na Figura 3 que trás como a eletrônica automotiva foi crescendo de maneira percentual em um veículo e a Figura 4 mostra os inúmeros módulos eletroeletrônicos em um automóvel.

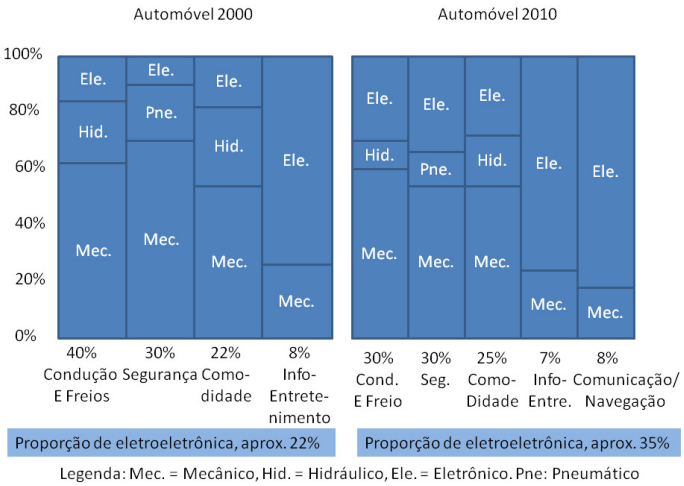

Figura 2: Proporção de eletrônica em um veículo (Bosch GmbH, 2007)

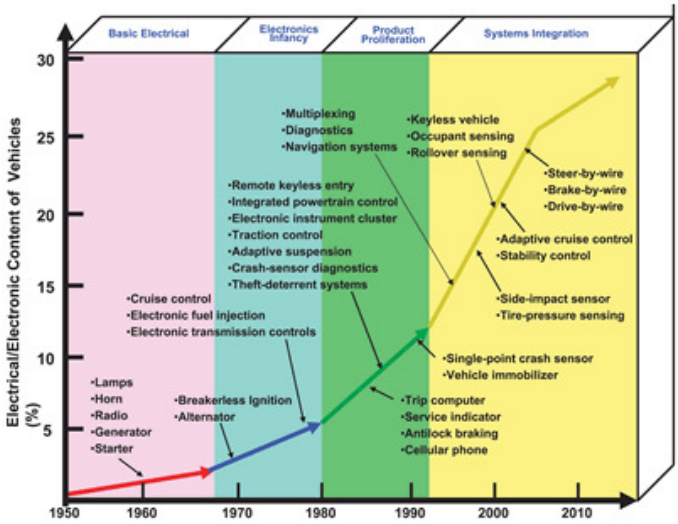

Figura 3: Crescimento ao longo do tempo de sistemas eletroeletrônicos nos veículos (Hallestrand, 2005)

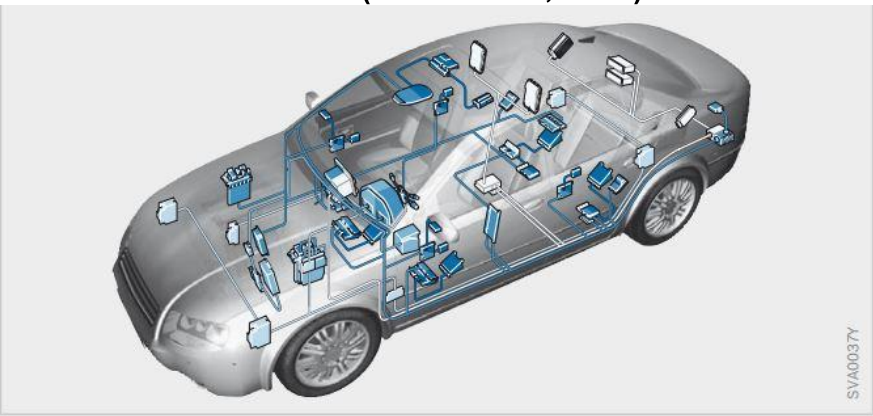

Figura 4: Módulos eletrônicos presentes em um veículo (Bosch GmbH, 2007)

Esses sistemas eletrônicos podem ser divididos em duas categorias: controle em tempo real das partes mecânicas e a de informações-entretenimento. A primeira parte possui sistemas para controle relacionados à estrutura de um veiculo, motor, transmissão que juntas formam o *powertrain*, e os sistemas de seguranças ativas, que são responsáveis por agir de modo a evitar um acidente. Enquanto a segunda é caracterizada por sistemas de gerenciamento de informações, navegação, computação, entretenimento e comunicação externa (Vincentelli & Natale, 2007).

Essas tecnologias eletroeletrônicas aprimoram os benefícios que são gerados aos usuários quanto à segurança, manutenção e conforto, economia de combustível e ao meio ambiente com a redução de emissões; a indústria em relação ao tempo de projeto e atualizações. Com esses benefícios é possível verificar que a aplicação e o desenvolvimento da eletroeletrônica na indústria automotiva tende a agregar e melhorar vários domínios dessa cadeia produtiva (Santos, 2010).

2.2 COMPONENTES ELETROELETRÔNICOS

Com a difusão da eletroeletrônica em um veiculo houve um aumento no uso de componentes eletrônicos na construção de um automóvel. Dentre esses pode se

citar os sensores, que possuem a característica de converter uma grandeza física em um sinal de tensão, os atuadores que convertem um sinal elétrico em energia mecânica. Para fazer o controle e o gerenciamento desses componentes houve a necessidade do uso das unidades de controle eletrônico, as ECUs.

2.2.1 Unidade de Controle Eletrônico (ECU)

As Unidades de Controle Eletrônico, *Electronic Control Unit* (ECU), são módulos eletrônicos cuja responsabilidade é a integração entre as grandezas físicas do meio externo através de sensores e atuadores, e com isso gerenciar e monitorar o funcionamento dos sistemas eletroeletrônicos através de softwares instalados em sua memória, e por fazer o funcionamento dos protocolos de comunicação. Esse módulo é composto por um microprocessador, ou um microcontrolador que é o responsável por executar as tarefas gravadas em memória e pelo processamento dos dados tanto de entradas, sensores, como de saídas, atuadores (Guimarães A. d., 2007). A Figura 5 mostra um exemplo de uma ECU.

Figura 5: Exemplo de uma ECU[1]

A estrutura de uma ECU pode ser dividida em três partes: a entrada de sinais, o processamento dos sinais, e os sinais de saída. As entradas e saídas podem ser definidas analógicas ou digitais. Sinais de entrada digitais são sinais com dois estados definidos, e sinais analógicos que não possuem um estado definido, mas sim uma variação contínua de tensão (Santos, 2010). A Figura 6 apresenta um diagrama de blocos em que mostra a estrutura de uma ECU.

Abaixo são apresentadas algumas aplicações das ECUs em um veículo, e como os seus nomes específicos representam diferentes aplicações (Guimarães A. d., 2007):

- *Body Control Module* (BCM): Módulo de controle da carroceria. Realiza funções básicas, como controle de farol e seta, ou adicionais, como controle de trava elétrica.
- *Engine Control Module* (ECM): Módulo de controle do motor: responsável por funções referentes ao funcionamento do motor.

[1] https://auto.howstuffworks.com/under-the-hood/trends-innovations/car-computer1.htm

- *Transmission Control Module* (TCM): Módulo de controle da transmissão: controla transmissão, sistema de câmbio, marchas.
- *Powertrain Control Module* (PCM): Módulo de controle do motor e transmissão: controle conjunto do motor e transmissão.
- *Telematics Control Unit* (TCU): Unidade de controle de telemáticas: navegação do veículo, GPS por exemplo.

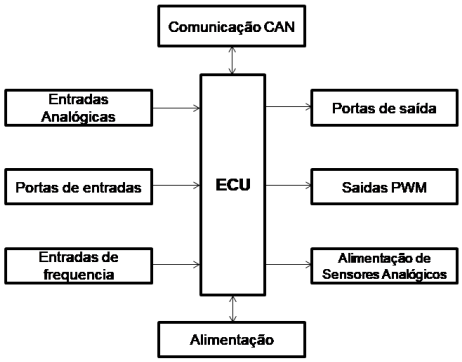

Figura 6: Diagrama de blocos de uma ECU[2]

2.3 ARQUITETURA ELETROELETRÔNICA

As tarefas de sistemas eletroeletrônicos com a utilização de ECUs podem ser realizadas em ambientes que são classificados em dois tipos: arquitetura centralizada e arquitetura distribuída.

2.3.1 Arquitetura Centralizada

Na arquitetura centralizada, uma única ECU é responsável por coletar sinais de sensores, processá-los e aplicá-los em uma saída. Dentro da ECU principal, há hardware e software que podem ter inúmeras entradas e várias saídas (Santos, 2010). A Figura 7 mostra os esquemáticos dessa arquitetura.

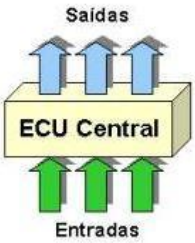

Figura 7: Arquitetura Centralizada
(Fonte: http://www.alexag.com.br/CAN_Bus_Parte_1.html)

Como vantagem a arquitetura centralizada possui um hardware mais simples com o uso de uma única ECU e todos os dados ficam disponíveis a unidade de controle central enquanto o sistema está em operação. Como desvantagem ela necessita de uma grande quantidade de cabos e os seus sistemas são mais limitados à expansão, pois modificar algo na ECU implica em uma modificação no

[2] https://www.ni.com/pt-br/innovations/white-papers/06/building-flexible--cost-effective-ecu-test-systems.html

hardware (Guimarães & Saraiva, 2002). A Figura 8 mostra uma aplicação de uma arquitetura centralizada:

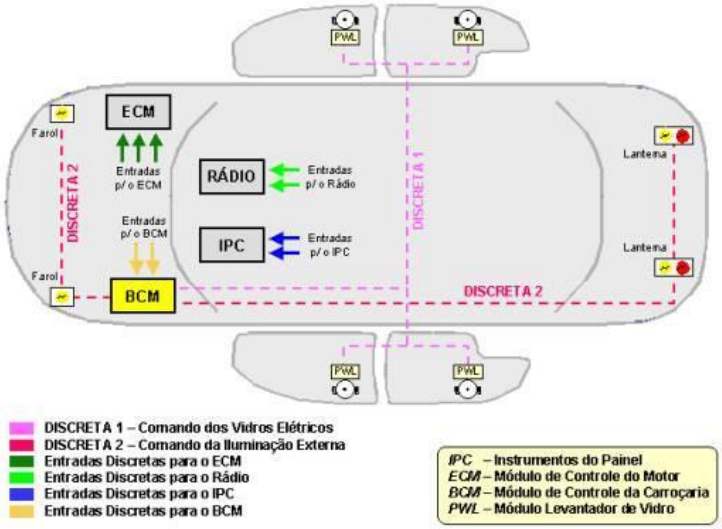

Figura 8: Exemplo Arquitetura Centralizada
(Fonte: http://www.pcs.usp.br/~laa/Grupos/EEM/CAN_Bus_Parte_1.html)

2.3.2 Arquitetura Distribuída

Na Arquitetura Distribuída várias ECUs com propriedades de entradas e saídas estão interligadas através de protocolos de comunicação, o que permite que haja uma distribuição de funções, o que seria restrito a uma única ECU pode está distribuído a inúmeras outras (Santos, 2010). A Figura 9 traz o esquemático desse tipo de ligação.

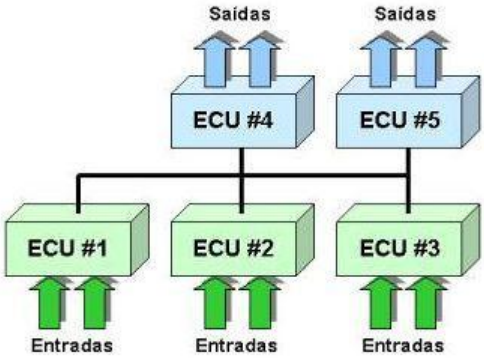

Figura 9: Arquitetura Distribuída
(Fonte: http://www.alexag.com.br/CAN_Bus_Parte_1.html)

Essa arquitetura tem como vantagem uma quantidade reduzida do uso de cabos, tornando o cabeamento mais barato e mais fácil de ser instalado. Além disso, possui conexões mais curtas entre os sensores e os módulos eletrônicos, um sistema mais flexível que permite que sejam adicionadas ou removidas Unidades de Controle Eletrônico (ECU's). Uma manutenção mais fácil, cada módulo pode ser produzido e testado individualmente de acordo com a necessidade para uma aplicação especifica (Fredriksson, 1994). Além disso, a arquitetura distribuída permite o uso de protocolos de comunicação aumentando a confiabilidade dos sistemas eletroeletrônicos, contudo como há vários módulos eletrônicos o uso de

protocolo se torna obrigatório para essa arquitetura (Guimarães & Saraiva, 2002). A Figura 10 traz um exemplo de uma arquitetura distribuída e as várias ECU's, se comunicando entre si.

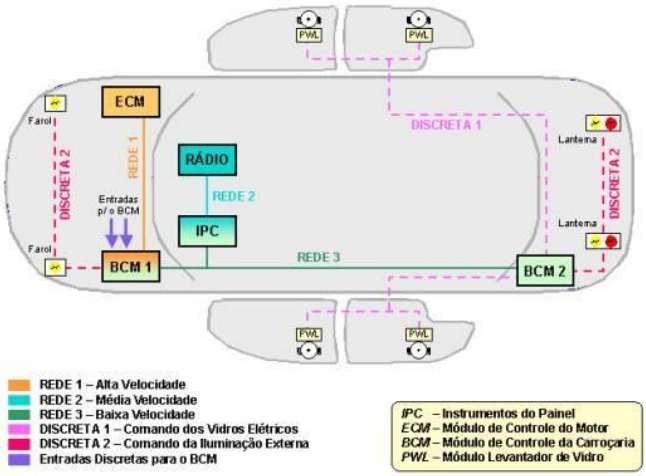

Figura 10: Exemplo Arquitetura Distribuída
(Fonte: http://www.pcs.usp.br/~laa/Grupos/EEM/CAN_Bus_Parte_1.html)

Por fim, muitos fatores devem ser levados em conta para a escolha da arquitetura a se utilizar. Dentre eles tem-se o quanto são complexos os sistemas que deverão ser controlados, componentes eletrônicos disponíveis para a montagem, o tempo que se tem para a produção e custo desejado do sistema final. Levando em conta a relação entre os fatores e aplicação desejada, então é feita a escolha da arquitetura que melhor se adéqua a necessidade (Guimarães A. d., 2007).

2.4 REDES AUTOMOTIVAS

Com a difusão da arquitetura distribuída e o aumento do número de ECUs, houve naturalmente a necessidade da troca de informações entre os sistemas eletrônicos. Essa necessidade surgiu, pois, muitos sinais que são processados em sistemas individuais precisam ser transmitidos as outras ECUs. Uma solução, para a troca de informações, veio com o desenvolvimento de sistemas de barramentos de comunicação que podem ser usados para transmitir grandes volumes de dados de diferentes módulos eletroeletrônicos, o que forma as redes de comunicação. A Figura 11 mostra a como essa quantidade de controladores foram crescendo ao longo dos anos em um automóvel (Bosch GmbH, 2007).

Essa troca de dados, entre as unidades de controle, é realizada por meio de um protocolo de comunicação, esses protocolos permitem a integração de boa parte dos sistemas eletrônicos. Os protocolos de comunicação são definidos como "meios de transmissão e recepção de dados utilizados para comunicar módulos eletrônicos e/ou sensores e atuadores inteligentes equipado com microcontroladores". Nesse caso ele é um meio para transmissão e recepção de dados para comunicação entre as ECU's (Guimarães A. d., 2007).

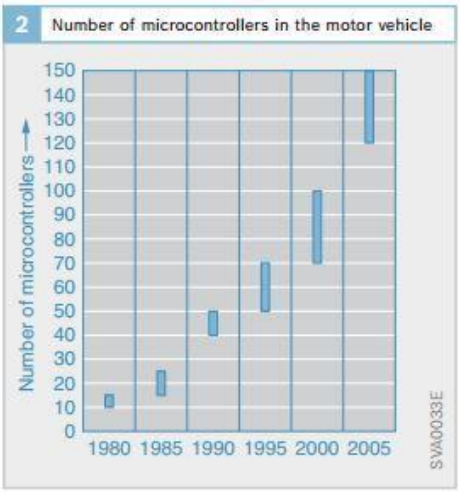

Figura 11: Crescimento da quantidade de microcontroladores em um veículo (Fonte: Bosch; 2007)

Há vários tipos de protocolos de comunicação, no qual cada um possui uma característica, e com isso aplicações mais apropriadas a uma determinada tarefa. Dentre eles existem, por exemplo, o protocolo LIN, o *FlexRay*, *Controller Area Network* (CAN). Esses tipos de protocolos são classificados quanto a sua velocidade, implicando assim na escolha de qual será usado conforme a aplicação necessária. Essa classificação é feita em classes, que são classe A, B, C, C+ que podem ser vista na Tabela 1.

Tabela 1: Relação de alguns protocolos existentes e suas características (Bosch GmbH, 2007)

	Classe A	Classe B	Classe C	Classe C+
Taxa de transmissão	Até 10kbps	10kbps – 125kbps	125kbps – 1Mbps	1Mbps – 10Mbps
Aplicação	Geralmente relacionada as funções de confort0	Geralmente relacionadas a entretenimento	Geralmente relacionadas aos sistemas de segurança	Geralmente relacionadas a controle te tempo real
Tipo de rede	LIN	CAN de baixa velocidade	CAN de alta velocidade	FlexRay

2.4.1 *Controller Area Network* (CAN)

Dentre os protocolos de comunicação existentes, o protocolo CAN possui uma grande variedade de configurações possíveis, sendo capaz de atender grande parte das necessidades em relação à troca de dados dos veículos automotivos existentes e em desenvolvimento. Essa capacidade de atender grande parte das necessidades faz com que o protocolo CAN, seja o protocolo mais utilizado em veículos (Guimarães A. d., 2007).

O *Controller Area Network* (CAN) é um protocolo de comunicação serial síncrono que consegue fazer com eficiência o controle em tempo real com um bom nível de segurança. Com isso, ele consegue atender sistemas com alto grau de complexidade e tendo um custo baixo (Bosch GmbH, 1991). Este sincronismo é

feito através dos tempos conhecidos e regulares em que as mensagens são enviadas através do barramento (Guimarães & Saraiva, 2002).

Paret (2005) mostra algumas propriedades do protocolo CAN:

- Hierarquia das mensagens;
- Garantir certo tempo de latência;
- Configuração flexível;
- Recepção de múltiplas fontes com sincronização de tempo (*multicast reception*);
- Operação multimestre;
- Detecção e sinalização de erro;
- Retransmissão automática de mensagens corrompidas em que o barramento retorna para o estado ocioso;
- Distinção entre erros temporários e falhas permanentes dos nós e o desligamento automático do nó em falha.

A hierarquia de mensagens é definida conforme a sua prioridade, que é determinado pelo seu identificador. A garantia do tempo de latência indica manter constante o tempo entre a requisição da transmissão e o seu tempo real de início (Paret, 2005). A configuração flexível permite que outros módulos eletrônicos sejam adicionados sem que haja a necessidade de mudanças significativas em hardware e software por outros sistemas (Bosch GmbH, 1991).

O conceito de *multicast* significa que as ECUs presentes na rede, recebem todas as mensagens, ele vem do conceito de filtrar as mensagens, ou seja, após a recepção a ECU verifica se a mensagem recebida foi direcionada para ela. O conceito de multimestre significa que qualquer ECU pode enviar uma mensagem para o barramento quando ele estiver livre (Bosch GmbH, 1991). Isso quer dizer que não tem como enviar uma mensagem exclusivamente para apenas um módulo, todos eles recebem a mensagem e fazem a filtragem para a identificação (Ataide e Pereira; 2012).

A detecção e sinalização de erro são identificadas, pois o barramento está em constante monitoramento, com isso é possível identificar o tipo de erro presente. Quando há uma identificação de erro em alguma mensagem, ela é rejeitada e inaceitável, com isso, deve haver uma retransmissão automática da mensagem. E quando há uma falha de alguma ECU ela é desligada da rede (Paret, 2005).

Como o protocolo CAN possui a característica de multimestre, ele é baseado no conceito *Carrier Sense Multiple Access/Colision Detection with Non-Destructive Arbitration* (Acesso Múltiplo a transmissão/ Detecção de Colisão com Arbitração não Destrutiva, CSMA/CD *with* NDA). Isso quer dizer que todas as unidades de controle verificam se há ou não mensagem trafegando na rede com maior prioridade, então o módulo de menor prioridade encerra a sua transmissão e aguarda até o barramento está livre novamente para iniciar o envio de dados. Quando o barramento está livre o módulo começa a transmissão, enviando o identificador da mensagem bit a bit (Fredriksson, 1994).

Com o envio de mensagens bit a bit o protocolo garante certa robustez na transmissão (Guimarães A. d., 2007). Durante a transmissão, todo transmissor

compara o nível do bit que é transmitido com o nível do bit que está sendo monitorado no barramento. Quando um bit recessivo, bit “1”, é enviado e a unidade percebe um bit dominante, bit “0”, então ela para a transmissão imediatamente. E o módulo com maior prioridade tem preferência sobre o barramento e com isso ele envia a sua mensagem (Bosch GmbH, 1991). A Figura 12 mostra como é feita a arbitração dos bits, em que na linha de transmissão prevalecerá à mensagem com maior prioridade (Bosch GmbH, 2007).

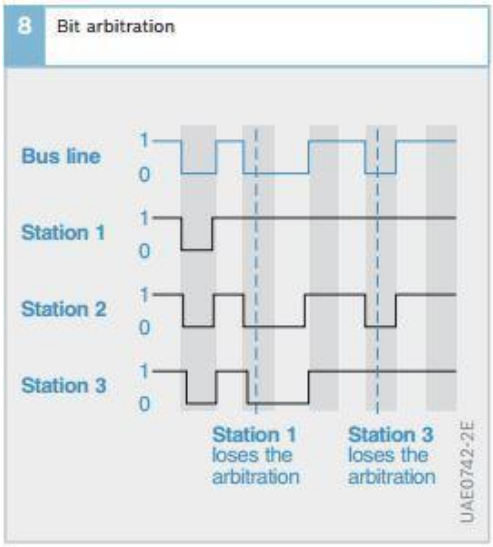

Figura 12: Arbitragem dos bits (Bosch GmbH, 2007)

Além disso, o barramento CAN faz a utilização de um par trançado diferencial. Esse conceito de par trançado faz com que interferências eletromagnéticas tenham menos efeito sobre o barramento, uma vez que qualquer ação sobre um dos fios afetará o outro. Essa interferência causa a mesma variação em ambos os fios, com isso a diferença de tensão entre ele permanece a mesma (Ataide & Pereira, 2012). O par trançado é formado pelos fios denominados CAN High (CAN_H) e CAN Low (CAN_L). Essa ligação pode ser vista na Figura 13, que mostra como é ligado os módulos, nós de redes, dos dois fios.

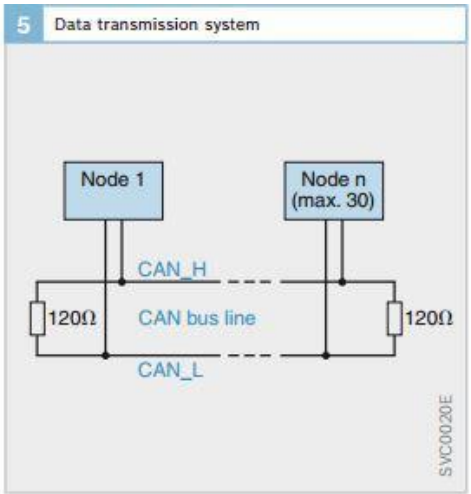

Figura 13: Sistemas de transmissão de dados (Bosch GmbH, 2007)

Com a utilização do par diferencial, quanto maior o comprimento do barramento, menor a velocidade de transmissão, com isso a velocidade é inversamente proporcional ao comprimento do barramento. Para um barramento de 40 metros de comprimento o protocolo CAN consegue uma taxa de transmissão de até 1Mbps. A Figura 14 mostra a relação entre o comprimento do chicote, que são os fios utilizados, e a taxa de transmissão dos dados (Guimarães & Saraiva, 2002).

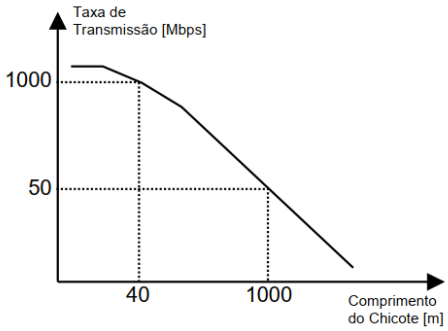

Figura 14: Relação entre comprimento da rede e taxa de transmissão (Guimarães & Saraiva, 2002)

2.4.1.1 Formato das Mensagens

A partir dessas definições, existem dois formatos de mensagens no protocolo CAN, o CAN 2.0A e o CAN 2.0B. O primeiro contém 11 bits de identificação, permitindo um total de 2048 mensagens trafegando na rede. A Figura 15 apresenta o formato do CAN 2.0A. E o segundo, CAN 2.0B, também conhecido como formato estendido, possui além 11 bits de identificação mais 18 bits, ou seja, um identificador de 29 bits, com esse formato é possível ter aproximadamente, 537 milhões de mensagens possíveis para trafegar na rede, o que diminui as limitações quanto à quantidade de módulos eletrônicos, mas em contrapartida há um aumento do tempo de envio de uma mensagem, o que pode ocasionar problemas para situações em tempo real (Santos, 2010). A Figura 16 apresenta o quadro de mensagem no formato estendido.

A partir da definição do formato, o protocolo possui diferentes tipos de quadros (*frames*). Esses quadros possuem a seguintes classificações (Bosch GmbH, 1991):

- Quadro de Dados;
- Quadro Remoto;
- Quadro de Erro;
- Quadro de sobrecarga;
- Quadro inter-quadros (*interframe*)

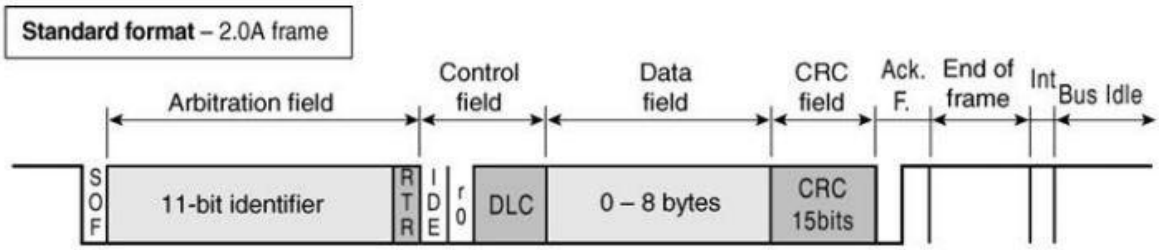

Figura 15: Formato CAN 2.0A (Paret, 2005)

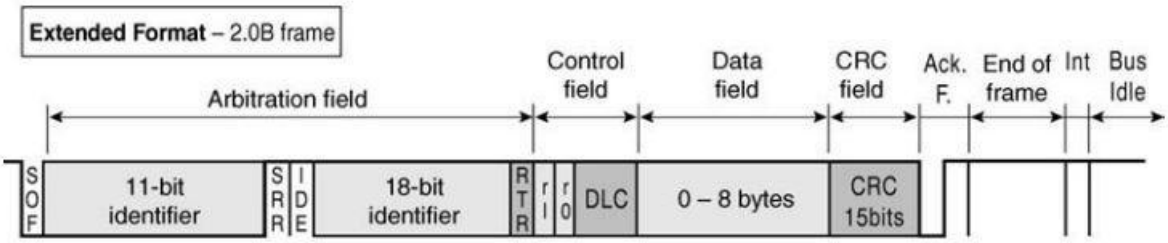

Figura 16: Formato CAN 2.0B (Paret, 2005)

2.4.1.1.1 Quadro de Dados

O quadro de dados para o CAN 2.0A, Figura 15, e o CAN 2.0B, Figura 16, podem ser divididos em sete partes principais, chamadas ‘campos’. (Paret, 2005):

- Início da mensagem (SOF);
- Campo de arbitração;
- Campo de controle;
- Campo de dados;
- Sequencia CRC;
- Campo *ACKnowledgement*;
- Fim do frame

O início da mensagem, *start of frame*, (SOF) consiste em um único bit dominante, sinalizando para todos os receptores o início de uma troca de mensagem. Essa troca só pode ocorrer se o barramento estiver sem tráfego de mensagem (Paret, 2005).

Já o campo de arbitração consiste nos *bits* identificadores, 11 bits para o formato padrão e 29 bits para o formato estendido, e do bit que vem imediatamente em seguida, chamado de requisição de transmissão remota, *remote transmission request* (RTR), em um quadro de dados, o bit RTR deve ser dominante. E o CAN 2.0B possui algumas diferenças em relação ao CAN 2.0A. Essa diferença consiste além dos 18 bits adicionais de identificação, no bit de substituição de requisição remota, *substitute remote request* (SRR), e no identificador do formato estendido, *identifier extension* (IDE) (Paret, 2005).

O campo de controle consiste em seis bits. Os dois primeiro bits são reservados para usos posteriores e para compatibilidade de crescimentos (Paret;

2005). Os outros quatro bits, código do tamanho de dados, *data length code* (DLC) indicam o número de bytes que contém o campo de dados. Isso permite que o receptor identifique quando todos os bytes da mensagem tenham sido recebidos (Bosch GmbH, 2007).

O campo de dados, é o que contém a realmente a mensagem que está entre 0 e 8 bytes, 0 a 64 bits. Isso permite que mais de um dado possa ser transmitido em uma única mensagem, como por exemplo, a temperatura e a rotação do motor (Paret, 2005).

Campo verificação redundante cíclica, *cyclic redundancy checksum* (CRC), contém 16 bits de checagem, 15 bits desde o bit inicial ao bit final do campo de dados. O 16º bit é um recessivo que encerra esse campo. O CRC é um meio de detectar uma possível interferência na transmissão (Bosch GmbH, 2007).

Campo *ACKnowledgement* (ACK) consiste em dois bits, ACK *slot* e ACK delimitador. ACK *slot* é transmitido como recessivo pelo transmissor e sobre escrito pelo receptor como dominante para dizer que a mensagem foi corretamente recebida. E o ACK delimitador deve ser sempre um bit recessivo (Bosch; 2007).

O fim da mensagem é identificado por uma sequência de sete bits recessivos que marcam o fim da transmissão (Paret; 2005). O espaço inter-quadros é composto por três bits recessivo que servem para separar mensagens sucessivas. Depois de um total de 10 bits recessivos, os módulos estão permitidos a começar uma nova transmissão (Bosch GmbH, 2007).

2.4.1.1.2 Quadro Remoto

Neste formato de mensagem, o bit RTR é recessivo, o que indica que não há dados a serem transmitidos. Esse quadro indica que a ECU que o enviou para o barramento está esperando algum dado de outra ECU com um quadro de dados, ou seja, ele é utilizado para requisição de dados (Paret, 2005).

2.4.1.1.3 Quadro de Erro

Esse quadro é enviado ao barramento quando a ECU detecta um erro na mensagem. Ele é composto por dois campos diferentes: a flag de erro e o delimitador de erro. A flag de erro é composta por seis bits que são classificados em erros ativo e passivo. O erro ativo é constituído por uma sequência de seis bits dominantes, enquanto o passivo por seis recessivos. O delimitador de erro é constituído por uma sequência de oito bits recessivos (Bosch GmbH, 1991).

2.4.1.1.4 Quadro de Sobrecarga

Quando uma ECU, ou um nó, está sobrecarregado de mensagens, ela envia um quadro de sobrecarga, indicando para os outros módulos interromperem momentaneamente o envio de mensagens. Esse quadro é formado por dois campos,

uma flag de sobrecarga constituída por seis bits dominantes e por um delimitador de sobrecarga formado por oito bits recessivos.

2.4.1.1.5 Quadro inter-quadros (*interframes*)

Quando há o envio de um quadro de dados ou quadro remoto, há o chamado espaço entre os quadros, ou quadro inter-quadros, o que indica que uma mensagem foi enviada e o barramento está se preparando para outro envio. Ele é formado por dois ou três campos, dependendo do que estava sendo transmitido (Paret, 2005). Esses campos são descritos abaixo.

Campo de intermissão: formado por três bits recessivos, indica que nenhuma ECU tem permissão para iniciar uma transmissão, seja de dados ou remota (Bosch GmbH, 1991).

Campo de bits ociosos: indica que a partir desse ponto algum módulo pode iniciar a transmissão de algum tipo de quadro de mensagens (Bosch GmbH, 1991).

Campo de suspensão de transmissão: após a transmissão de alguma mensagem de erro passivo, são enviados oito bits recessivos seguidos do campo de intermissão, e após o campo de bits ociosos o barramento estará livre novamente para envio de mensagens (Bosch GmbH, 1991).

2.5 IMPLEMENTAÇÃO DE UMA REDE DE COMUNICAÇÃO CAN BUS

Com a definição dos conceitos sobre redes de comunicação, é possível expressar os pontos mais importantes na hora de projetar e implementar uma rede de comunicação veicular. Para a implementação de uma rede CAN é preciso, a determinação da arquitetura eletroeletrônica, a escolha de um padrão CAN referente à aplicação, a especificação do dicionário de dados, e também o desenvolvimento de hardware e do software de aplicação (Guimarães A. d., 2007)

Ao se considerar uma tarefa que a rede deve realizar como primeiro passo pode ser feita a escolha da arquitetura eletroeletrônica que vai ser utilizada. Nesse ponto, podem-se enfrentar duas situações (Guimarães A. d., 2007)

1) Ao usar ECUs prontas que ainda não trabalham em rede, estabelecer uma comunicação entre elas;
2) Ou projetar as ECUS, considerando as leituras da entrada e os respectivos processamentos, as ações a serem tomadas nas saídas e a troca de dados entre as ECUs através do barramento de comunicação.

Com isso, para o desenvolvimento de um projeto de uma rede CAN pode-se destacar as seguintes tarefas (Guimarães A. d., 2007):

- Ler entradas digitais ou analógicas;
- Encapsular os dados no formato determinado pelo CAN;
- Transmitir os dados pelo barramento CAN à outra ECU;

- Receber dados de outra ECU, transmitidos pela rede CAN;
- Processar esses dados, e comandar as saídas necessárias.

Após a escolha da arquitetura, também deve ser feita uma análise da norma relacionada à aplicação. Essas normas são informações que especificam as classes de aplicação, como os cabos e conectores que são necessários e os tipos de mensagens que irão trafegar na rede (Guimarães A. d., 2007).

Com isso, outro conceito importante a ser estabelecido, é o dicionário de dados. Ele pode ser caracterizado como uma tabela em que as mensagens existentes na aplicação, são relacionadas as ECUs responsáveis por sua transmissão e recepção. É a parte mais dedicada da aplicação quando se trabalha com um protocolo como o CAN (Guimarães A. d., 2007).

Após a definição do dicionário de dados a melhor forma de organizar um dicionário é criar uma matriz que contem os módulos existentes na rede. A matriz deve mostrar cada mensagem e relacionar os módulos que transmitem e recebem. Nela pode conter o tempo de atualização dos valores de uma mensagem, o tempo entre uma transmissão e outra e o valor do seu identificador. Além desses identificadores, também deve contar uma descrição dos dados existentes na rede (Guimarães A. d., 2007).

Após a definição do dicionário de dados relativo à aplicação, pode-se ter o desenvolvimento do projeto de hardware das ECU's. Para o projeto de hardware se tem duas alternativas para se implementar uma ECU (Guimarães; 2007):

1) Utilizar um microcontrolador (sem CAN) conectado a um controlador CAN, transmissor-receptor, transceptor, *transceiver* (utilizando dois CIs diferentes interligados). O microcontrolador, é responsável pela análise das entradas e pela tomada de decisão das saídas, trocando informações por portas de comunicação especificas como o controlador CAN, responsável pelo empacotamento dos dados no formato do protocolo, sua transmissão e recepção;
2) Utilizar um microcontrolador com CAN incorporado. O CI passa a ser responsável não só pela recepção e tratamento das entradas e o comando para as saídas, como também pelo empacotamento dos dados no formato CAN, sua transmissão e recepção.

Por fim, é necessário o desenvolvimento dos softwares que deverão ser gravados nas ECUs. Eles são responsáveis pelo desenvolvimento de rotinas de gerenciamento e funcionamento da rede e do seu sistema eletroeletrônico. Com isso, é possível realizar o monitoramento e gerenciamento de entradas e saídas e da comunicação entre as ECUs(Guimarães; 2007).

2.6 *PROJETO BASEADO EM MODELOS (MODEL-BASED DESIGN)*

Seguindo a premissa da lei de Moore em que o número de transistores dobra em um circuito integrado a cada dois anos, a quantidade de sistemas eletroeletrônicos cresceu de forma significativa ao longo dos tempos. Visto isso, um

veículo poderá conter em torno de 60 microprocessadores em sua arquitetura eletroeletrônica e até 10 milhões de linhas de código poderão estar sendo executadas em automóveis com eletrônica embarcada mais sofisticada (Aaresntrup, 2010).

Esse aumento no número de sistemas eletroeletrônicos torna cada vez mais complexo o desenvolvimento e a inovação na indústria. Além disso, esses sistemas possuem tempo menor de projeto e restrições de custos, mas os desafios vão além da parte técnica como em questões organizacionais e culturais. A partir dessa necessidade maior por evolução os desenvolvedores e empresas passaram a aplicar o "*Model-Based Design*" (MBD) (Nicolescu & Mosterman, 2010).

MBD, Projeto Baseado em Modelos, é uma abordagem para desenvolvimento de projetos centralizada em modelos, o que muitas vezes não depende da criação de protótipos físicos. Ele permite a simulação em nível de sistemas e de componentes separadamente; geração de código para sistemas embarcados, por exemplo; verificação e testes contínuos ao longo de um projeto (Aaresntrup, 2010). A Figura 17 mostra como é o funcionamento do MBD.

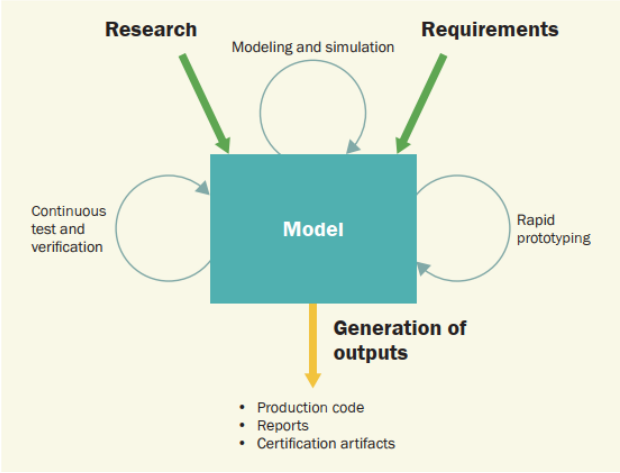

Figura 17: Fluxo de trabalho do MBD

Além da sua definição o MBD é centrado nos conceitos listados a baixo (Aaresntrup, 2010):

- Especificação executável;
- Simulação em nível de sistemas;
- O que será analisado;
- Elaboração do Modelo;
- Protótipo virtual;
- Testes e verificação contínua;
- Automação;
- Gerenciar e capturar conhecimento.

A especificação executável é o modelo que engloba todas as informações do projeto, assim diminui as margens para ambiguidade. A simulação em nível de sistemas significa que o modelo do sistema completo é simulado para verificar as interações entre os componentes existentes no projeto. O que será analisado é um

método de simulação para testar novos conceitos e aprender sobre o sistema em análise.

A elaboração do modelo é um processo que permite criar uma simulação com o maior grau de fidelidade a sistemas reais e que todo o sistema seja sempre testado. Protótipo virtual é uma técnica que possibilita validar um projeto antes de ser construído em hardware o que economiza tempo visto que criar um modelo é mais rápido do que fazer a sua construção física. Testes e verificação contínua estão presentes em todas as etapas da elaboração de um projeto, essa parte pode ser exemplificada na Figura 18 em que é mostrado o modelo V de desenvolvimento do projeto, onde os sistemas são testados conforme os requisitos.

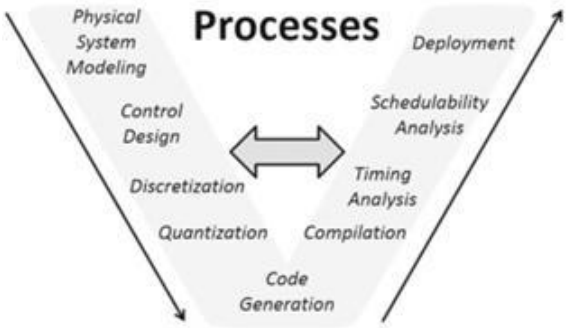

Figura 18: Diagrama de modelo V

A Automação é utilizada com métodos e ferramentas em que envolvam tarefas repetitivas que estão mais propensas a erros. Nela está incluída a geração de código, desenvolvimento de códigos para hardwares específicos, conduzir as verificações dos modelos para garantir que estão sempre em conformidade com o esperado, promover documentos para as especificações dos sistemas e promover medidas corretivas quanto à geração de código. E por fim gerenciar e capturar conhecimento significa que, como muitas partes de um projeto são feitas separadamente, há a necessidade das suas informações estarem documentadas (Aaresntrup, 2010).

Vistos os principais conceitos, o objetivo do MBD é produzir modelos de alto nível que permitam aos projetistas analisar e especificar aplicações por meio de simulações, em vez de iniciarem análises através de protótipos físicos. Assim, ao serem utilizados modelos os desenvolvedores conseguem gerenciar de forma mais fácil e prática a complexidade das aplicações durante o seu desenvolvimento e quando há a necessidade de modificações (Parteno, 2000).

Ao ter a aplicação do MBD os projetistas conseguem os primeiros resultados mais rapidamente, ter um produto final com qualidade superior e expansão das capacidades de se criar sistemas mais complexos (Mathworks, 2014). Alguns benefícios dessa aplicação podem ser listados abaixo:

- Reduz o número de protótipos físico utilizados, assim, um único protótipo pode ser testado para vários tipos de aplicações reduzindo custos e utilizando componentes mais baratos que atendam as especificações necessárias;

- Reutilização e adaptação de modelos criados: reduzindo tempo para testes e verificações, adaptando vários modelos a um único protótipo criado
- Uso de equipes de desenvolvimento menor: torna possível maior velocidade de execução com quantidade reduzida de pessoas trabalhando, ajudando ainda mais a reduzir custos e tempo.
- Identificar e corrigir erros continuamente ao longo da fase de desenvolvimento e execução;
- Melhorar os algoritmos para ser possível usar em múltiplos domínios, ou seja, um único algoritmo base servir para execução em várias outras aplicações.

Adotar o MBD implica diretamente no desenvolvimento de sistemas embarcados, e em todos os cenários do desenvolvimento são obtidos contínuos benefícios durante a execução. Tanto para grandes ou pequenos projetos, a criação de apenas um protótipo implica diretamente no tempo e no custo final de uma aplicação (Mathworks, 2014). Por fim, a criação e simulação de modelos exigem o uso de ferramentas computacionais que são essenciais ao desenvolvimento do projeto.

3 PROJETO DA PLATAFORMA

Neste capítulo será apresentado o projeto da plataforma didática. Ela consiste em um dispositivo que permite o desenvolvimento de projetos com a utilização de arquiteturas eletroeletrônicas automotiva. Pode-se dividir a aplicação da plataforma em três partes: hardware, no qual uma placa de desenvolvimento foi construída fisicamente; software, em que são desenvolvidos programas para as ECUs; e integração, em que é possível integrar tanto software quanto hardware e criar um sistema de comunicação com o usuário que desenvolveu alguma aplicação. A Figura 19 mostra o modelo geral da plataforma com as partes separas e o projeto como um todo.

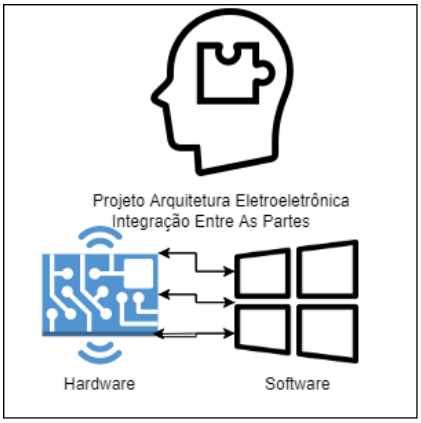

Figura 19: Modelo geral da plataforma

A partir dessa concepção geral foram definidos os requisitos para a construção da plataforma:

- Placa de desenvolvimento compacta e de baixo custo;
- Entradas digitais e analógicas;
- Saídas Digitais e analógicas;
- Área de prototipagem;
- Módulo para comunicação CAN;
- Permitir o desenvolvimento de projetos utilizando arquiteturas eletroeletrônicas automotivas;
- Flexibilidade para permitir diferentes desenvolvimentos de aplicações;
- Aplicação dos conceitos de MBD.

3.1 SOLUÇÃO DE HARDWARE

Definidos os requisitos da plataforma, o objetivo dessa solução é agregar em uma única placa, componentes que podem ser utilizados para projetar e testar funcionalidades de uma arquitetura eletroeletrônica automotiva. De forma geral o hardware da placa foi definido conforme mostra a Figura 20. A plataforma foi pensada de forma a possuir módulos em seu projeto, que podem ser definidos como: Entradas, Saídas, Controladores, Comunicação, Prototipagem.

A partir da definição geral do modelo pode se definir cada módulo separadamente. O módulo de Controladores foi constituído por quatro microcontroladores. De forma a atender os requisitos da plataforma foram escolhidos

quatro Arduinos Nano. Os mesmos possuem fácil acesso, com preço baixo, e são excelentes para o aprendizado, pois não possuem alta complexidade para criar aplicações. Além disso, possui várias entradas analógicas e digitais, e saídas digitais, com a possibilidade de diferentes projetos de arquiteturas. A Figura 21 traz uma foto desse controlador.

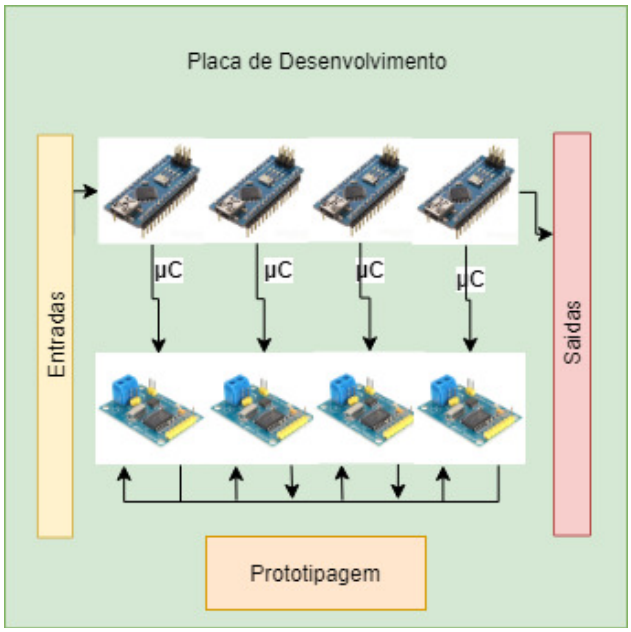

Figura 20: Modelo Geral Hardware da Plataforma

Figura 21: Arduino Nano

Ligado ao módulo dos Controladores, tem-se a parte de Comunicação, que juntos formam as ECUs. Esse módulo foi criado para que fosse possível a comunicação através do protocolo CAN, pois o microcontrolador utilizado no Arduino Nano, o Atmega 328p não possui o transceptor CAN. Com isso foram escolhidos quatro dispositivos que utilizam o CI MCP2515, pois é um dispositivo compacto e de baixo custo. Ele é um controlador CAN, um *transceiver*, que permite tanto a transmissão quanto recepção de um quadro de mensagens no barramento de comunicação. A sua comunicação com o Arduino é possível com a utilização do padrão de comunicação SPI (“*Serial Peripheral Interface”*). A Figura 22 mostra o módulo MCP2515.

Como o Arduino possui entradas analógicas e digitais, o módulo de entrada é responsável por fornecer sinais de sensores, sejam eles analógicos ou digitais. Para isso, foi definido um total de oito entradas, quatro analógicas e quatro digitais. As entradas analógicas foram definidas através de potenciômetros resistivos, pois eles

fornecem uma variação contínua de tensão. Para as entradas digitais foram colocados quatro *"push buttons"* que fazem alterações de estados definidos de tensão, 0 ou 5V, *On Off*. A Figura 23 mostra um modelo de potenciômetro e de um botão.

Figura 22: Módulo MCP2515

Figura 23: Potenciômetro e "*Push Button*"

Através dos valores da entrada e com o processamento dos dados, os controladores podem atribuir certos comandos através dos seus pinos, e para isso foi criado um módulo para saídas digitais, constituído por oito leds. Com isso, a escolha dos leds se deve ao fato de serem componentes pequenos, de baixo custo e baixo consumo de energia, e mostram de maneira didática e simples, ações tomadas pelas ECUs, pois possuem apenas dois estados, aceso e apagado, *On Off*.

Além do módulo de saída digital, foi desenvolvido outro módulo para saídas analógicas, cuja finalidade é mostras variações contínuas de sinais através do processamento de dados de um Arduino. Contudo, os controladores utilizados não possuem saídas definidas para sinais analógicos, com isso foram utilizadas as saídas controladas através do sinal PWM ("*Pulse Width Modulation*"). PWM é um sinal digital, com certo valor de frequência, formando uma onda quadrada, em que é controlada a largura do pulso, "*Duty Cycle*" (D) dado em porcentagem, entre 0 e 100%, ou seja, o tempo em que está em 0V ou 5 V, como mostra a Figura 24. A largura do pulso pode ser controlada, por exemplo, através do nível de tensão dos potenciômetros, quanto menor o valor de tensão da entrada, menor a largura do pulso e quanto maior o valor de tensão da entrada maior a largura do pulso. Com essa aplicação do PWM é possível controlar a intensidade luminosa de algum led da saída por exemplo.

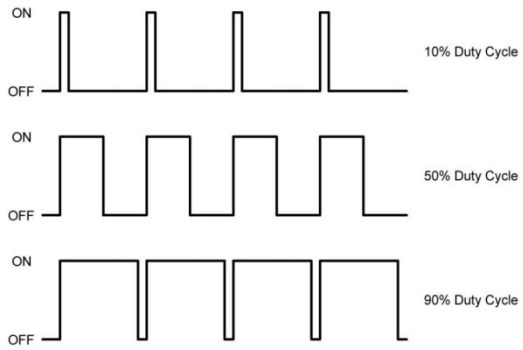

Figura 24: Sinal PWM

Com isso, o módulo desenvolvido para a representação de saídas digitais foi constituído por dois filtros passa baixa, como mostrado na Figura 25. Esse tipo de filtro foi utilizado, pois possui a característica de um conversor DA. Ele realiza a operação de filtrar sinais de entrada, Vinput, realizar uma filtragem que significa atenuar sinais de tensão com valores de frequência superiores a frequência de corte, e fornecer um sinal de saída filtrado, Voutput. Esse filtro teve algumas especificações para o seu desenvolvimento, ele obedece ao comportamento da sua função de transferência geral, que é dada pela Equação (1).

$$H(s) = \frac{V_{Output}}{V_{Input}} == \frac{\omega_0^2}{s^2 + 2s\zeta\omega_0 + \omega_0^2} \tag{1}$$

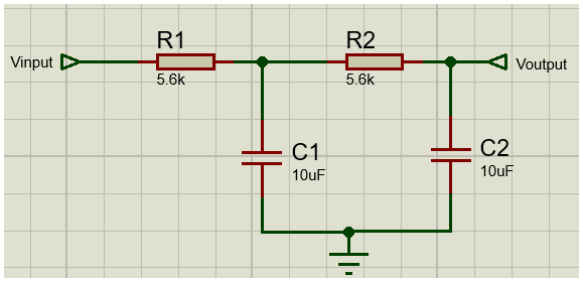

Figura 25: Filtro Passa Baixa

Conforme a especificação do filtro, com os valores de capacitores (C) e resistores (R), foi possível chegar a sua própria função de transferência conforme mostra a Equação (2). Com isso pode-se verificar o valor da sua frequência de corte, f_c, em que para valores maiores de frequência há uma atenuação significativa do valor de tensão na saída, o valor f_c é de 1Hz para o filtro projetado. Além disso, o seu comportamento pode ser visualizado na Figura 26 em que mostra o diagrama de Bode do filtro utilizado. Esse é um diagrama que mostra como para valores maiores de freqüência há a atenuação da magnitude dos valores de tensão na saída.

$$H(s) = \frac{V_{Output}}{V_{Input}} = \frac{\frac{1}{R_1R_2C_1C_2}}{s^2 + s\left(\frac{1}{R_1C_1} + \frac{1}{R_2C_1} + \frac{1}{R_1C_2}\right) + \frac{1}{R_1R_2C_1C_2}} = \frac{318.877}{s^2 + s53.57 + 318.877} \tag{2}$$

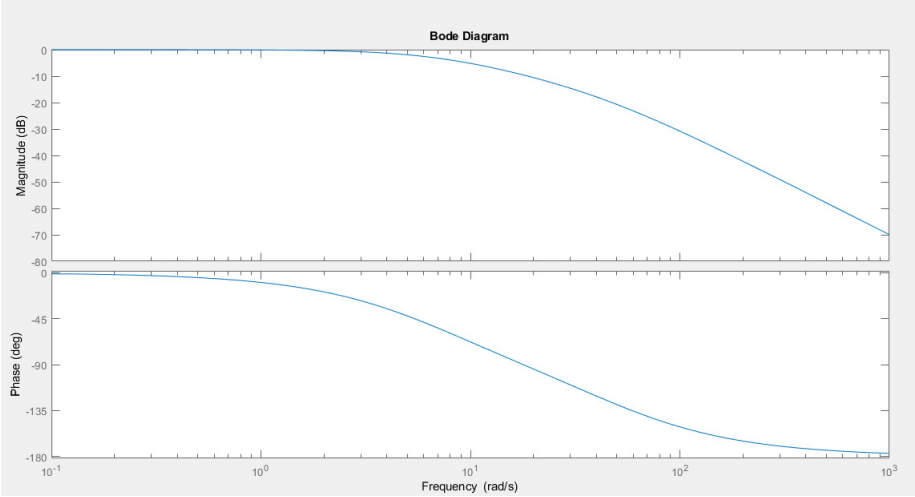

Figura 26: Diagrama de Bode Filtro Passa Baixa

Com a aplicação do filtro passa baixa, a conversão DA é feita através da largura de pulso gerado pelo sinal PWM fornecido pelo controlador. Além disso, o valor da tensão de saída no filtro passa baixa, Voutput, depende da tensão de entrada, que para o Arduino Nano é de 5V, e da largura de pulso D, como mostrado na Equação (3). E por fim a frequência do sinal PWM possui valor de 1kHz, ou seja, um valor maior que a frequência de corte do filtro passa baixa.

$$V_{Output} = V_{Input} * D \tag{3}$$

Conforme especificados os módulos de saídas e entradas, a plataforma não se limita a possuir aplicações com esses sensores e essas saídas. Em virtude disso, duas miniprotoboard foram adicionadas para representar uma área de Prototipagem. Servem para que seja possível criar diferentes aplicações de arquiteturas eletroeletrônicas, criando novos circuitos para estudos. Essa miniprotoboard pode ser vista na Figura 27

Figura 27: Mini-Protoboard

Para ligar tanto as entradas quanto as saídas e a área de prototipagem ao módulo de Controladores, foram utilizados barras de pinos, "*pinheaders*". Isso foi feito, pois permite que haja um grau de liberdade durante a escolha de quais componentes serão utilizados e a aplicação da área de prototipagem. Com isso, é

permitido que haja várias formas de projetos de arquiteturas eletroeletrônicas automotivas. A Figura 28 mostra como são essas barras de pinos.

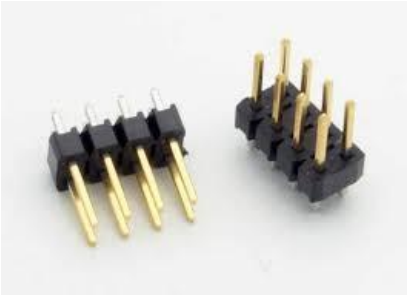

Figura 28: Barra de pinos

Por fim, a plataforma pode ser alimenta por uma fonte externa através de um conector P4, localizada na parte superior da placa de desenvolvimento. Além disso, Também é possível alimentar cada microcontrolador separadamente por seu conector USB. Esses dois tipos de alimentações são possíveis, pois a depender da aplicação, podem se necessário alimentar todos os Arduinos ou não. A Figura 29, apresenta a forma em 3D, criado no Proteus (Labcenter, 2019), da forma final da plataforma. A Figura 30 mostra a placa física em sua forma final, com dimensões finais de 20cmX20cm, garantindo assim um dispositivo compacto e de fácil transporte e com o uso de componentes de baixo custo.

Figura 29: Plataforma em modelo Digital (Proteus)

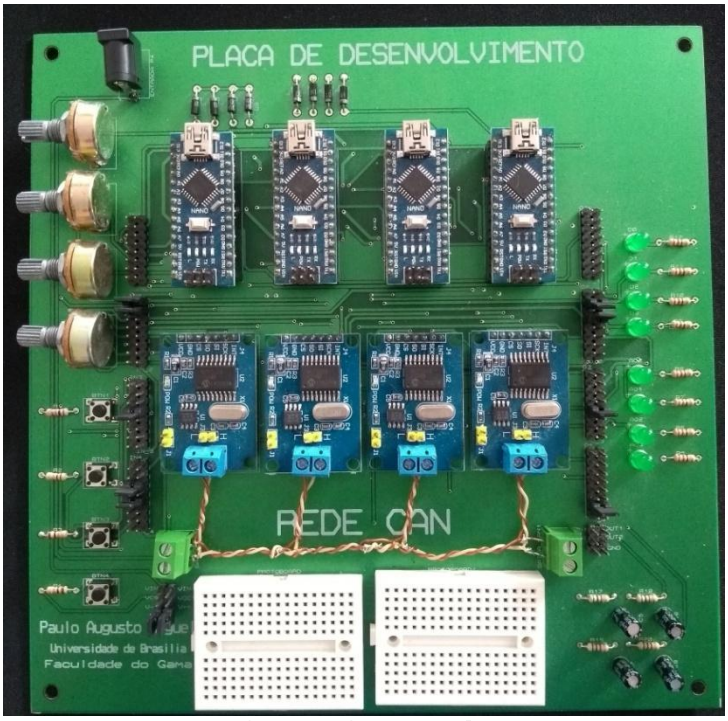

Figura 30: Plataforma Construída Fisicamente

3.2 SOLUÇÃO DE SOFTWARE

Conforme a definição de hardware foi necessário definir uma solução em software para a integração entre as duas partes. Essa solução consiste em apresentar os modelos criados através do Matlab/Simulink que foram utilizados para programar os Arduinos e criar os projetos de arquiteturas eletroeletrônicas com a utilização do protocolo de comunicação CAN. Isso é possível, através dos blocos existentes no Simulink, em que com a utilização da geração automática de códigos os modelos podem ser gravados nos Arduinos. Isso é feito com suporte para o Arduino, através do "*Simulink Support Packge for Arduino Hardware*", com isso é possível aplicar os conceitos de MBD durante a criação de projetos, Além disso, possui suporte para o uso do protocolo CAN, e a sua interatividade com dispositivos físicos através do "*Vehicle Network Toolbox"* (VNT).

"*Vehicle Network Toolbox"* é uma ferramenta que torna possível a interação de dispositivos que utilizam o protocolo CAN com o Matlab e o Simulink. Essa ferramenta possui blocos no Simulink, presentes na Figura 31, que permitem enviar, receber, encapsular e desencapsular dados em um quadro de mensagens. Com o Matlab/Simulink é possível analisar, filtrar e monitorar as mensagens transmitidas pela rede CAN. Com todas essas características, ainda é possível desenvolver tanto redes virtuais para simulação, como conectar modelos produzidos no Simulink a uma rede física (Mathwork, 2015). A Figura 32 mostra um exemplo de uma arquitetura eletroeletrônica utilizando o barramento CAN conectado ao Matlab/Simulink. As principais características dessa ferramenta podem ser listadas abaixo:

- Conectar a hardware que utilizam o protocolo CAN;
- Comunicação entre o Matlab/Simulink e o barramento CAN;
- Simular a comunicação CAN;

- Visualizar as mensagens presentes no barramento de comunicação.

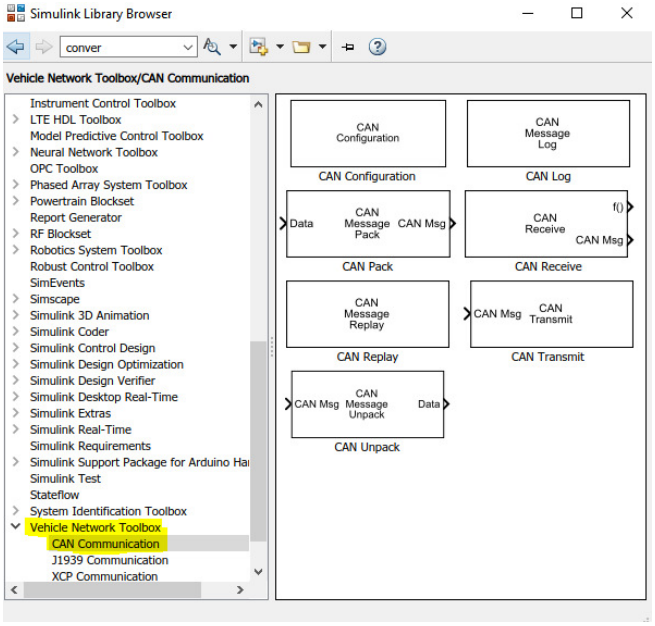

Figura 31: Blocos de programação "Vehicle Network Toolbox"

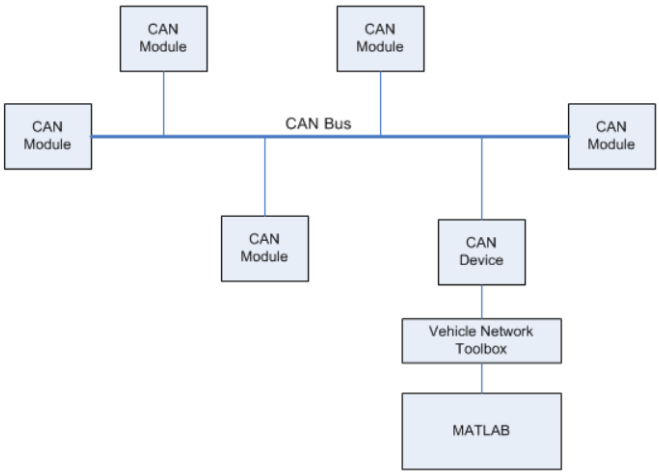

Figura 32: Dispositivo CAN Ligado ao Matlab/Simulink

A comunicação do VNT com hardware é utilizado para o desenvolvimento de projetos de redes físicas. Isto acontece para que além de serem analisados os parâmetros enviados e recebidos por um módulo de comunicação eletrônico, também seja possível configurar os parâmetros existente na rede, como por exemplo, o modo da mensagem, se será um quadro remoto ou um quatro de dados. Para que haja essa interação, é necessária a instalação de pacotes de suporte a hardwares, ou seja, pacotes de suporte do Simulink com módulos eletrônicos. Para esse trabalho, foi necessária a instalação do pacote que faz o suporte do Arduino ao Simulink, o "*Simulink Support Packge for Arduino Hardware*".

A instalação desse pacote traz as funcionalidades que são básicas para a programação do microcontrolador. São blocos que permitem controlar os pinos de

entrada e saída do Arduino, assim como alguns protocolos de comunicação. A Figura 33, mostra os blocos comuns dentro do pacote de suporte para o Arduino.

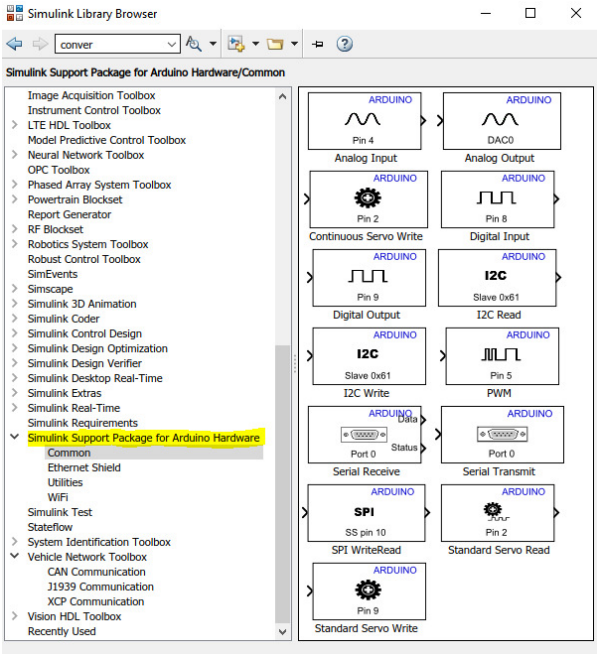

Figura 33: Blocos de programação do Arduino

Contudo, mesmo após essas etapas de configuração das ferramentas necessárias, não é possível a integração do "*Vehicle Network Toolbox*" com o Arduino. Isso se deve, pois os blocos do VNT possuem suporte apenas para alguns hardwares próprios para comunicação CAN. Entretanto, a documentação permite que sejam criados blocos customizados ("*Custom Blocks*") tanto para a transmissão, quanto para a recepção de dados (Mathwork, 2015).

Esses blocos para transmissão e recepção são o **CAN Transmit** e o **CAN Receive**. São eles que fazem com que sejam enviados e recebidos dados de uma rede tanto virtual quanto física para análise no Simulink. Com isso foi necessário o desenvolvimento de três modelos de blocos customizados para que fosse possível a integração com o Arduino. Essa criação foi feita através do bloco chamado **S-Function Builder**. Figura 34 mostra o conjunto de S-Function presente no Simulink.

Um S-function Builder é responsável por integrar um código C, C++ e criar uma *C MEX S-Function* com as diretrizes fornecidas pelo programador. Um arquivo MEX (MEX file) é o responsável por fazer os códigos fontes escrito em C serem interpretados pelo Matlab e após essa interpretação sejam, carregados e executados dentro de um dispositivo[3], por exemplo, o Arduino. Somente após isso, foi possível integrar todos os componentes para a construção da plataforma de desenvolvimento. A Figura 35 apresenta a caixa de diálogos do bloco e os campos em que são utilizados para configurar para uma aplicação desejada, mais informações podem ser acessadas no próprio site da MathWorks[4].

[3] https://www.mathworks.com/help/simulink/slref/sfunctionbuilder.html
[4] https://www.mathworks.com/help/simulink/sfg/s-function-builder-dialog-box.html

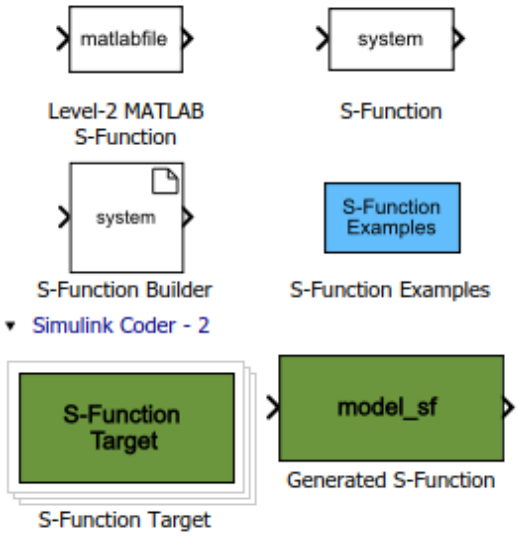

Figura 34: Bloco S-Function Builder

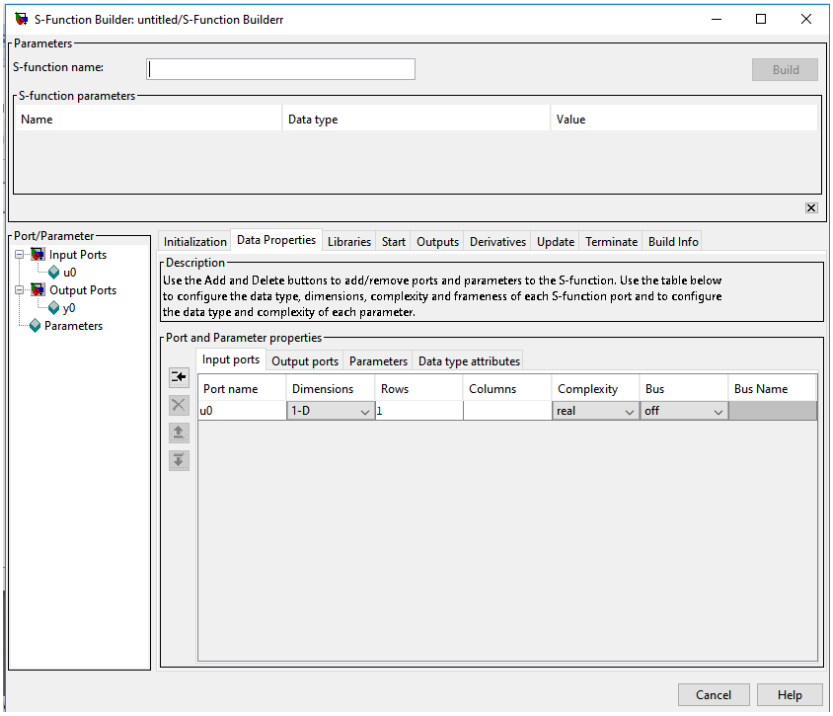

Figura 35: Caixa de Diálogo S-Function Builder

3.2.1 Módulos customizados

Após todas as instalações e configurações serem finalizadas foi possível a criação dos módulos de transmissão e de recepção customizados e adaptados ao hardware usado no trabalho, o Arduino Nano, utilizando o microcontrolador Atmega328p. Esses modelos padrões foram desenvolvidos de forma que fosse possível aplicar os conceitos do MBD. Com essa forma padrão, ou forma base, permite a integração entre o Arduino e o "*Vehicle Network Toolbox*". Esses blocos foram denominados por: **MCP2515CAN_Transmit**, para a transmissão, **MCP2515CAN_Receive**, para a recepção de dados e o **MCP2515CAN_TransmitReceive** para programar os módulos que necessitem tanto

de enviar quanto receber uma mensagem. Com isso, a etapa de desenvolvimento desses modelos segue o fluxograma apresentado na Figura 36.

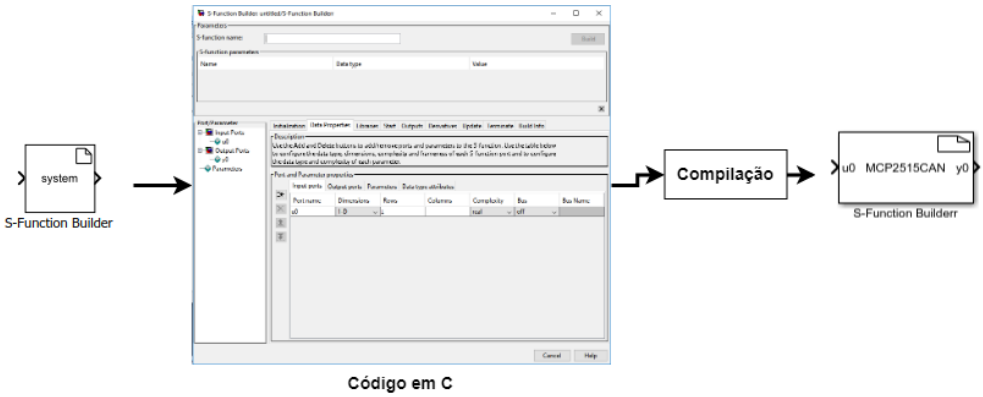

Figura 36: Geração dos blocos Customizados

O primeiro bloco a ser apresentado é o **MCP2515CAN_Transmit**, responsável por receber uma mensagem CAN através do **Can Pack** e enviá-la na rede de comunicação. A Figura 37 mostra à esquerda (E) modelo utilizado para a criação dos blocos para a transmissão e a direita (D) como são ligados os blocos para a programação dos Arduinos.

O segundo modelo apresentado é o **MCP2515CAN_Receive** que é responsável por receber os dados do barramento CAN. Após a recepção ele envia a mensagem para o bloco **CAN Unpack.** Ele é responsável pelo desencapsulamento, ou seja, separar as informações contidas no conteúdo da mensagem. A Figura 38 apresenta o modelo utilizado para a criação dos blocos e como são ligados os blocos de recepção e o Unpack.

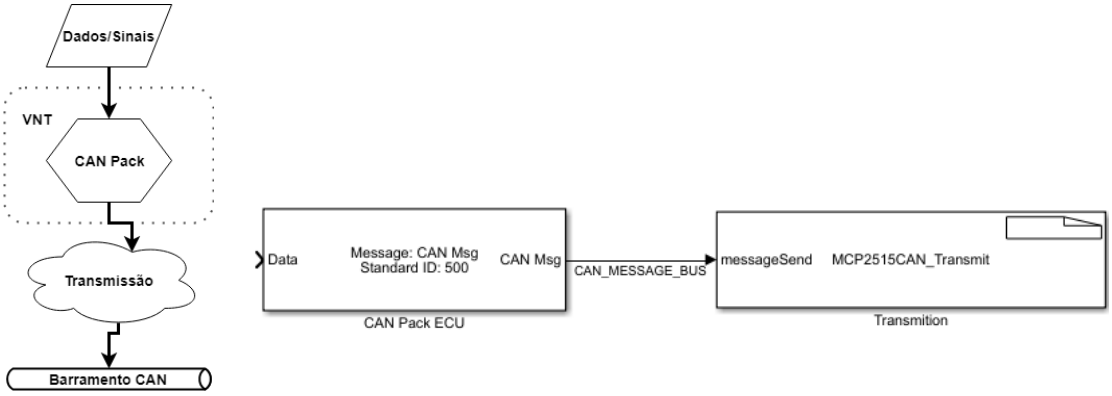

Figura 37: Fluxograma de Transmissão (E), Modelo Simulink de Transmissão (D)

O terceiro modelo apresentado é o **MCP2515CAN_TransmitReceive**. Ele é responsável tanto por enviar quanto por receber os dados do barramento CAN. Esse modelo foi criado em virtude de não ser possível utilizar o **MCP2515CAN_Transmit** e o **MCP2515CAN_Receive** juntos em uma aplicação, pois há conflitos e erros durante a geração do código para o Arduino. A Figura 39 mostra como esse modelo foi gerado e a Figura 40 mostra como esse bloco de transmissão e recepção é conectado ao Pack e ao Unpack.

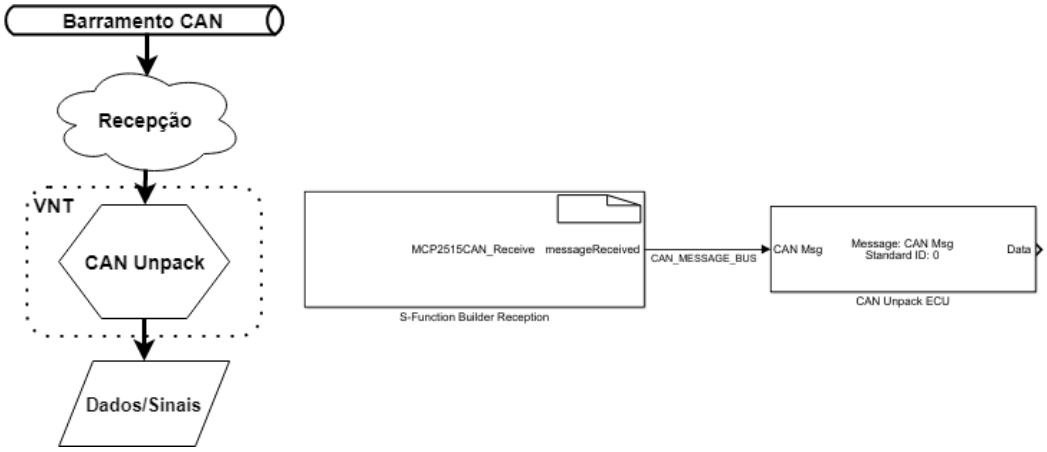

Figura 38: Fluxograma de recepção (E), Modelo Simulink de Recepção (D)

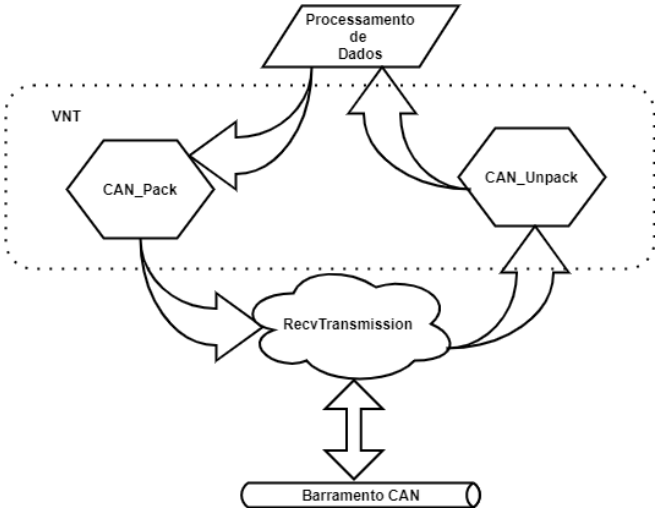

Figura 39:Fluxograma para gerar o modelo de transmissão e recepção

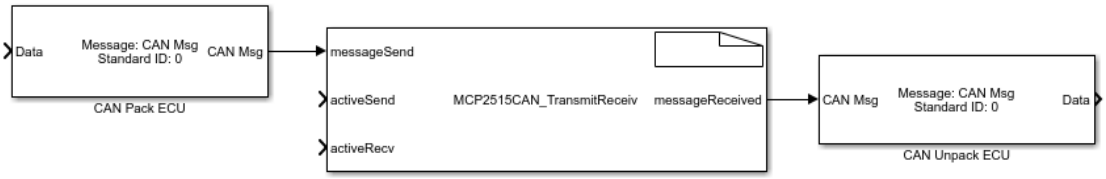

Figura 40: Modelo de transmissão e recepção no Simulink

Esse bloco foi criado, pois em diversas aplicações há a necessidade de uma ECU tanto enviar quanto receber dados para ao barramento de comunicação. Os blocos de transmissão e de recepção são utilizados para aplicações em que a ECU apenas envia ou recebe dados.

Após as criações desses blocos, há duas formas dos programas serem carregados nos Arduinos para a execução, a Figura 41 mostra a diferença entre os ícones no Simulink. A primeira consiste em colocar para simular com o botão de Play, e o modo de simulação externa, isso significa que o Simulink permanecerá em comunicação com o Aduino e será possível visualizar os dados presentes durante a execução por meio dos blocos existentes, como o Display. A segunda forma consiste em apenas carregar o código no Arduino e assim o Simulink não ficará em constante comunicação com o hardware.

Figura 41: Modos Execução Arduino

4 ESTUDOS DE CASO

Neste capítulo serão apresentados os estudos de caso realizados para testes e validações da plataforma. Eles servem para mostrar o desenvolvimento de projetos utilizando arquiteturas eletroeletrônicas automotivas com a aplicação do protocolo de comunicação CAN. Além disso, tem como objetivo simular situações reais que podem ocorrer durante o projeto de desenvolvimento e uso de um veículo. E mostra também como o MBD pode ser usado, pois há o reaproveitamento dos módulos criados para transmissão e recepção de dados, diminuindo custos e tempo de preparação e execução e com a utilização de apenas um protótipo da plataforma desenvolvido de forma física. Por fim, para todos os estudos de caso foi utilizada a taxa de transmissão de 250kbps para o barramento CAN.

4.1 ESTUDO DE CASO 01

Dentre os inúmeros sensores contidos em um veículo, podem-se destacar os digitais, como por exemplo, o sensor de rotação do motor. Ele se caracteriza por fornecer dados de forma digital e de forma periódica, como mostra a Figura 42. Através do dado que ECU ler é possível calcular e informar ao motorista qual a rotação que o motor se encontra no momento atual durante o seu funcionamento, a Figura 43 mostra um exemplo de um sensor de rotação.

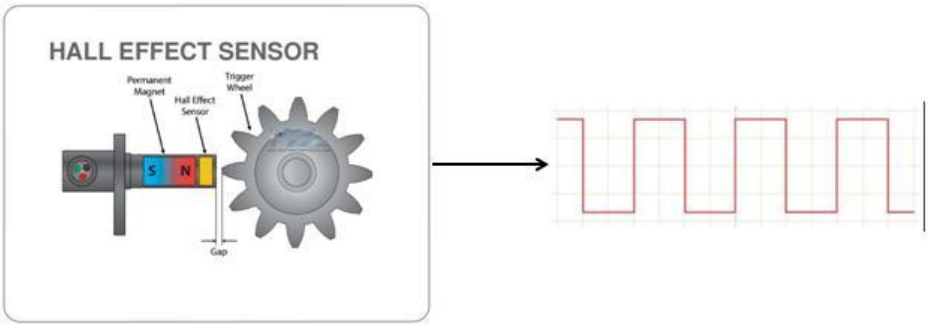

Figura 42: Exemplo do funcionamento de um sensor de rotação

Figura 43: Sensor de Rotação em um Veículo

Visto isso, esse estudo de caso mostra o projeto de uma arquitetura eletroeletrônica que contém duas ECUs, uma para envio de dados de forma digital e outra para recepção de dados. A Figura 44 apresenta o modelo para o estudo de caso. A ECU1 é a responsável por enviar um quadro de dados através do protocolo

CAN, o seu código foi carregado no controlador. A ECU2 recebe os dados através do barramento e faz a comunicação com o Simulink, isso foi realizado através do modo de simulação externa.

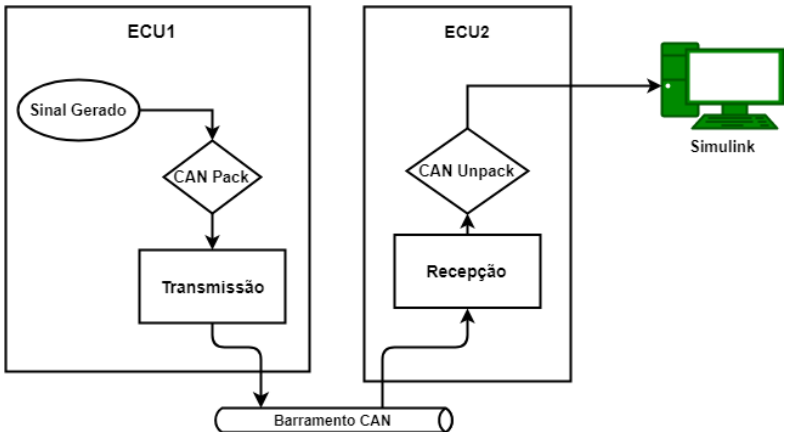

Figura 44: Modelo Caso 01

4.1.1 Projeto

A primeira ECU foi definida como a que irá enviar os dados através do barramento CAN. O dado que será encapsulado para a transmissão através do barramento, foi gerado por um bloco gerador de pulso, neste caso uma onda quadrada, que possui frequência de 1Hz e largura de pulso de 50%, o que significa que em metade do período estará em nível alto e a outra meta em nível baixo. Esse tipo de dado foi utilizado, pois simula um sensor de rotação.

Além disso, foi gerado o dicionário de dados para esse estudo como mostra a Tabela 2. Nele é especificado o endereço, o tipo do *frame*, o intervalo de tempo da transmissão, o tamanho da mensagem (DLC), a ECU transmissora, Tx, e a ECU receptora, Rx.

Tabela 2: Dicionário de dados Estudo de Caso 01

	Mensagem	ID	Tipo Frame	ECU1	ECU2	Intervalo de Tempo	DLC	Start Bit	Length Bits
1	Onda Quadrada	500	Standard	Tx	Rx	100 ms	1 Byte	0	8

O projeto da ECU2 foi desenvolvido para que fosse realizada a recepção das mensagens através do barramento. Além disso, ela é responsável pela integração dos módulos de comunicação da plataforma com o computador, o que torna possível visualizar a forma que ocorre a comunicação entre os dispositivos e apresenta a mensagem recebida.

4.1.2 Análise dos Resultados

Após o carregamento dos programas nos controladores, foi possível fazer uma análise do comportamento da rede e verificar se a comunicação ocorreu de forma como especificado. O programa para envio das mensagens foi carregado no Arduino. O modelo de recepção pode ser executado em modo externo, e com isso foi possível comparar a mensagem enviada com a mensagem recebida de forma gráfica como mostra a Figura 45.

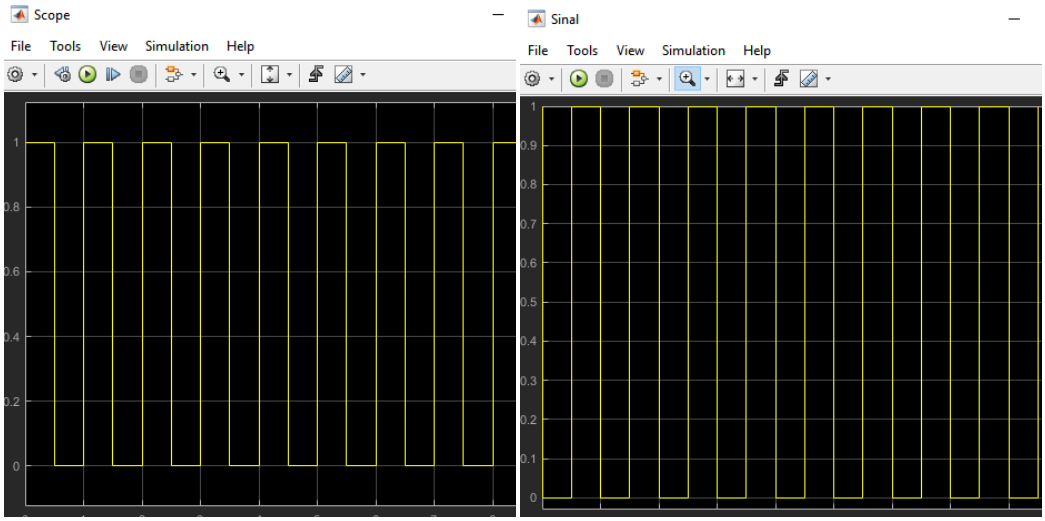

Figura 45: Onda enviada (E) e onda recebida (D)

Com a análise gráfica é possível ver que a onde recebida possui o mesmo padrão da onda enviada. Com amplitude 1 e a mesma frequência e largura de pulso. Após essa verificação foi feita uma análise estatística para verificar o tempo médio de recepção das mensagens, conforme mostra a Tabela 3, em que foi constatado que o tempo de envio foi alcançado na média.

Tabela 3: Estatísticas dos resultados do Estudo de Caso 01

ID	Intervalo Tempo	Número Amostras	Média	Desvio Padrão
500	100ms	395	100,7 ms	0.67 ms

Por fim, podem ser visualizados os dados através do monitor Serial da IDE do Arduino. Com isso, foi mostrado que apenas o primeiro byte possui dado, e a mudança entre o valor 0 e 1 da mensagem, que pode ser visto na Figura 46. Após essas análises pode ser verificado que a rede de comunicação obteve comportamento esperado e satisfatório.

```
15:32:32.292 -> Byte1   Byte2   Byte3   Byte4   Byte5   Byte6   Byte7   Byte8
15:32:32.292 -> 0
15:32:32.394 -> Message ID: 500
15:32:32.394 -> Data
15:32:32.394 -> Byte1   Byte2   Byte3   Byte4   Byte5   Byte6   Byte7   Byte8
15:32:32.394 -> 0
15:32:32.498 -> Message ID: 500
15:32:32.498 -> Data
15:32:32.498 -> Byte1   Byte2   Byte3   Byte4   Byte5   Byte6   Byte7   Byte8
15:32:32.498 -> 1
15:32:32.600 -> Message ID: 500
15:32:32.600 -> Data
15:32:32.600 -> Byte1   Byte2   Byte3   Byte4   Byte5   Byte6   Byte7   Byte8
15:32:32.600 -> 1
15:32:32.702 -> Message ID: 500
15:32:32.702 -> Data
15:32:32.702 -> Byte1   Byte2   Byte3   Byte4   Byte5   Byte6   Byte7   Byte8
15:32:32.702 -> 1
15:32:32.805 -> Message ID: 500
15:32:32.805 -> Data
15:32:32.805 -> Byte1   Byte2   Byte3   Byte4   Byte5   Byte6   Byte7   Byte8
15:32:32.805 -> 1
```

Figura 46: Monitor Serial Mensagens Recebidas Caso 01

4.2 ESTUDO DE CASO 02

O segundo estudo foi desenvolvido de modo a demonstrar o funcionamento de uma arquitetura eletroeletrônica contendo três ECUs. Além da aplicação de três controladores ele foi feito para testar e validar tanto componentes de entrada, como no caso dois potenciômetros e um botão, como componentes de saída, os leds e o circuito conversor de tensão. Com a utilização dos componentes de entrada é possível simular sensores reais presentes em um automóvel, por exemplo, um botão que funciona como um sensor de porta, indicando se está aberta ou fechada. Além disso, os potenciômetros sevem para simular sensores de temperatura ou um sensor de posição do pedal do acelerador, exemplos desses sensores podem ser visto na Figura 47.

Figura 47: Exemplo de sensores presentes em um veículo

Além disso, tem como objetivo mostrar como uma única ECU de recepção realiza a separação das mensagens recebidas através do barramento com a utilização do VNT. A Figura 48 mostra o modelo para esse tipo aplicação, em que há duas ECUs responsáveis pela leitura de dados dos sensores e envio das mensagem e o módulo de recepção e comunicação com o computador.

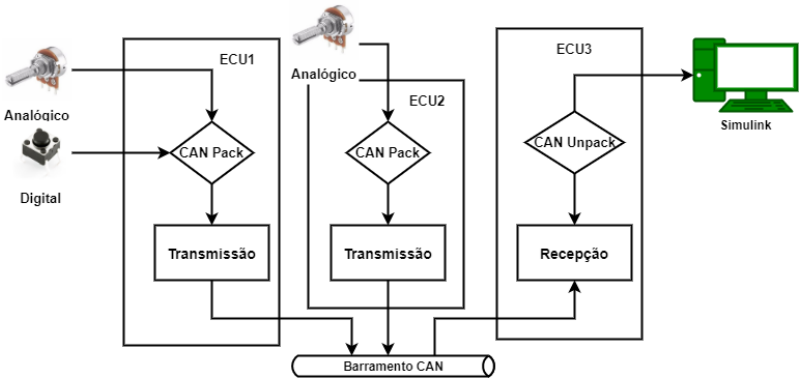

Figura 48: Modelo Caso 02

4.2.1 Projeto

O projeto consiste em três ECUs em comunicação através do protocolo CAN. Duas responsáveis pela leitura das informações dos sensores e transmissão através da rede e outra para a recepção das mensagens e comunicação com o computador. Com isso, foi definido o dicionário de dados para o desenvolvimento da arquitetura eletroeletrônica que pode ser visto na Tabela 4. Serão ao todo três sinais que serão encapsulados em dois quadros de dados.

Tabela 4: Dicionário de dados Estudo de Caso 02

Mensagem	ID	Tipo Frame	Intervalo Tempo	DLC (Bytes)	Start Bit	Length Bits/ Bytes	ECU Tx	ECU Rx
BTN 1	356	Standard	100ms	3	0	8/1	1	3
Pot. 1	356	Standard	100ms	3	8	16/2	1	3
Pot. 2	550	Standard	300ms	2	0	16/2	2	3

Com a definição do dicionário de dados, a primeira ECU faz a leitura dos sinais de dois sensores, um analógico e outro digital. Com isso ela encapsula os dados dentro de um quadro de dados para poder transmitir através do barramento a cada 100ms como definido no dicionário de dados. A segunda ECU faz a leitura de um sensor analógico encapsula em um quadro de dados e o transmite a cada 300ms através do barramento. E a terceira, recebe os dados através da comunicação CAN, faz o desencapsulamento das mensagens e se comunica com o computador para mostrar os dados recebidos.

4.2.2 Análise dos Resultados

Após o desenvolvimento do projeto foi possível realizar a comunicação entre os módulos eletrônicos e verificar o funcionamento da rede. Com isso, o código de envio de dados foi carregado nas duas ECUs de leitura dos sensores e envio de dados, mas o código de recepção foi executado em modo de simulação com a fonte externa, o que permitiu a análise tanto no Simulink da rede quando na placa de desenvolvimento através dos comandos de saídas. Após a execução em modo externo, foi possível visualizar através dos subsistemas os dados que são provenientes da rede de comunicação, isso pode ser visto na Figura 49, que mostra em um dado momento da execução alguns valores através dos displays.

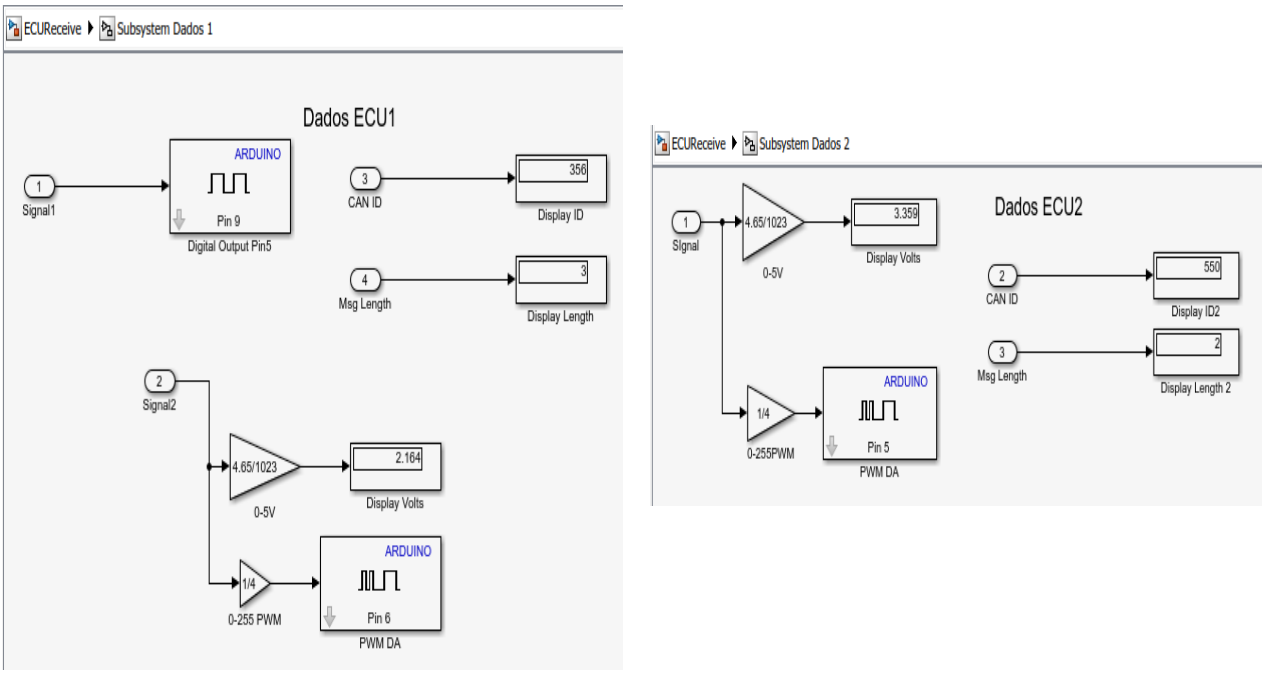

Figura 49: Execução Externa: ECU1 ID356 (E) e ECU2 ID 550 (D)

A partir dessa simulação no Simulink foi possível testar como se deu o funcionamento na placa. Primeiro foi visto o funcionamento do dado digital, em que consistia quando era pressionado o botão um led de saída era aceso, e apagado quando ele era solto. Também foi possível comparar os valores de tensão de entrada dos dois sensores analógicos com os valores mostrados pelos Displays Volts com a saída dos conversores DA. Essa comparação foi feita através da medição da tensão com um multímetro e pode ser vista na Figura 50 que mostra para a mensagem de ID 550, entrada a esquerda (E) e a saída a direita (D) e na Figura 51 para a mensagem de ID 356.

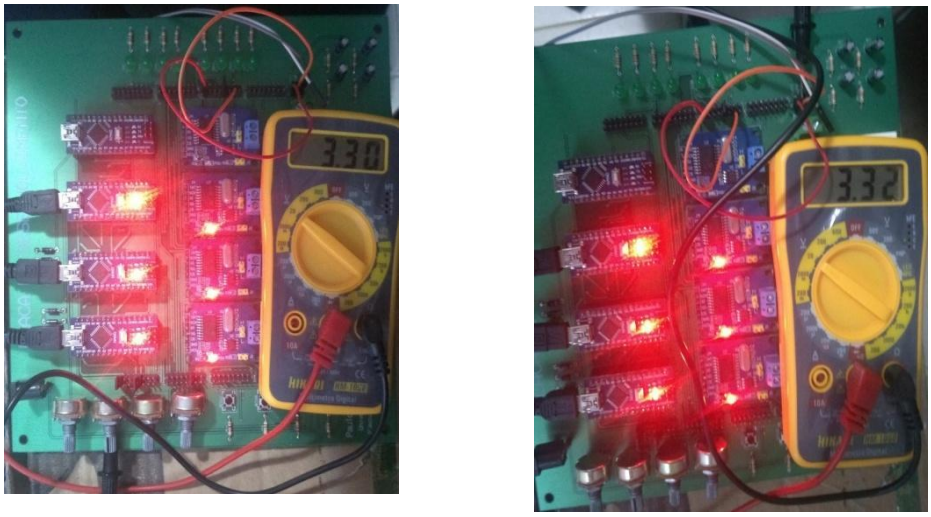

Figura 50: Comparação Entrada (E) e Saída ID 550 (D)

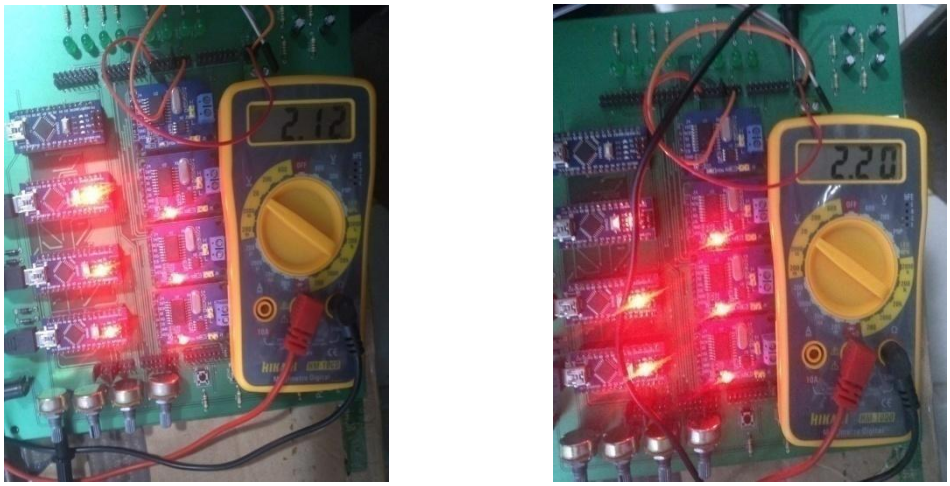

Figura 51: Comparação Entrada (E) e Saída ID 356 (D)

Também foi feita uma análise estatística dos valores da rede, em que é possível ver o tempo médio de recepção das mensagens e compará-los com os valores esperados, como mostra a Tabela 5. Com isso, é possível ver que o envio das mensagens foi realizado com um tempo médio muito próximo ao esperado, e o desvio padrão com valor baixo, ou seja, mesmo havendo o envio por dois módulos diferentes os módulos conseguiram garantir o tempo de envio especificado para cada mensagem. E por fim, é possível ver no monitor serial, na Figura 52 as mensagens recebidas através do barramento, e como o ID 550 possui um tempo de

envio três vezes maior, foi visto que a cada três mensagens do ID 356 era recebida uma do ID 550.

Tabela 5: Dados Estatísticos do Segundo Estudo

ID	Intervalo Tempo	Média	Desvio Padrão	Amostras
356	100ms	100,07ms	0,66	382
550	300ms	299,85ms	0,43	229

```
Message ID: 356
Data
Byte1   Byte2   Byte3   Byte4   Byte5   Byte6   Byte7   Byte8
0       227     1
Message ID: 356
Data
Byte1   Byte2   Byte3   Byte4   Byte5   Byte6   Byte7   Byte8
0       227     1
Message ID: 356
Data
Byte1   Byte2   Byte3   Byte4   Byte5   Byte6   Byte7   Byte8
0       227     1
Message ID: 550
Data
Byte1   Byte2   Byte3   Byte4   Byte5   Byte6   Byte7   Byte8
41      2
```

Figura 52: Monitor Serial do Segundo Estudo

4.3 ESTUDO DE CASO 03

O terceiro estudo de caso foi desenvolvido visando simular o monitoramento e o diagnóstico de uma rede automotiva. Isso se deve, pois, as ECUs devem estar sempre monitorando os sensores, atuadores e a própria rede de comunicação, e quando há algum erro ele deve ser armazenado em memória para que possa ser acessado, esses erros gravados são as memórias de avarias. A Figura 53 mostra um exemplo de um *scanner* automotivo de uso comercial, e como ele indica os erros presentes em um veículo.

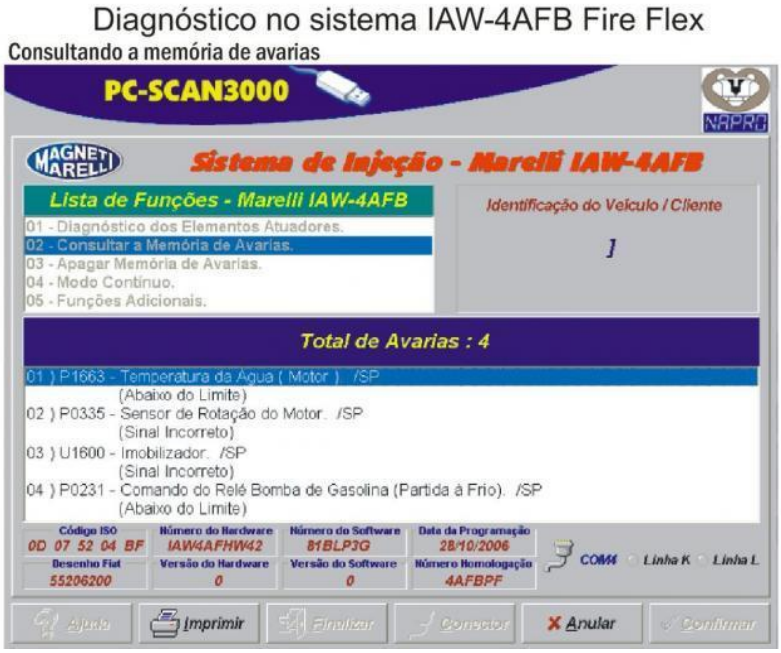

Figura 53: Tela de diagnóstico de um Scanner Automotivo

Foi desenvolvido com o uso dos quatro microcontroladores da placa, três que são responsáveis por verificarem dados de sensores, quatro analógicos e um digital, e um módulo para a comunicação com o computador, através de um aplicativo visando simular um Scanner automotivo. A Figura 54 mostra um diagrama geral utilizado para o desenvolvimento desse estudo de caso, em que as duas primeiras ECUs fazem o monitoramento de dois sensores analógicos e a terceira monitora um sensor digital. E a Figura 55 mostra como ficou na placa esse estudo de caso.

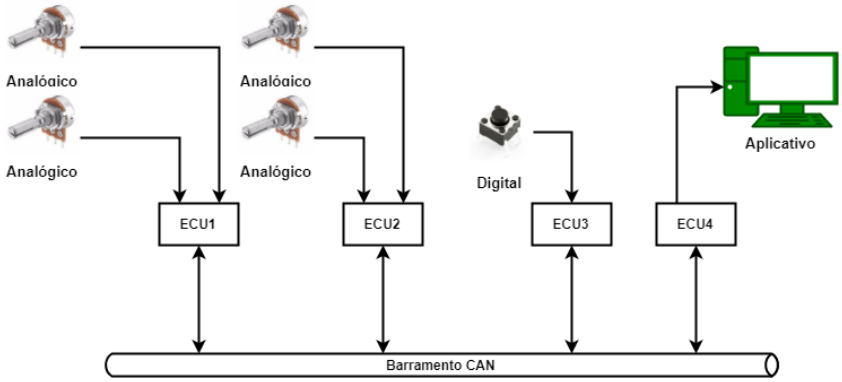

Figura 54: Modelo Caso 04

Figura 55: Estudo Verificação de Erro na Placa

4.3.1 Projeto

Os programas para serem gravados nos controladores foram divididos da seguinte forma: Três são responsáveis pelo monitoramento e verificação dos erros, pois quando há a detecção de algum erro é gravado na memória dos módulos um código, ele é chamado de código de problema de diagnóstico, DTC (*Diagnostic Trouble Code*). Além disso, esse código pode ser chamado de memória de avaria, a Tabela 6 mostra a relação desses códigos com os seus respectivos sensores.

Cada DTC possui uma especificação conforme o tipo de erro que foi detectado. No caso dos sensores analógicos, os potenciômetros, Pot., possuem o erro referente a dois limites de operação: tensão superior, TS, ou tensão inferior, TI. Como os valores de tensão podem ser compreendidos entre 0 e 5V, foi definido

como limites de funcionamento, 0.7V a TI e 3,8 a TS. Esses erros indicam que os valores de tensão medidos na entrada dos controladores extrapolaram os seus limites de operação. Para o sensor digital foi utilizado um dos botões da placa, BTN, nesse caso o erro é gerado após o botão ser pressionado por 3s ou mais. Com isso esses erros são armazenados na memória permanente do Arduino, através do seu código DTC equivalente.

Tabela 6: Relação DTC, Sensores, Tipo de Erro e ECU

DTC	Sensor	Tipo de Erro	ECU
801	Pot. 1/ S1	TS	1
802	Pot. 1/S1	TI	1
701	Pot. 2/S2	TS	2
702	Pot. 2/S2	TI	2
601	Pot. 3/S3	TS	1
602	Pot. 3/S3	TI	1
501	Pot. 4/S4	TS	2
502	Pot. 4/S4	TI	2
400	BTN. 1/S5	3s	3

A quarta ECU é responsável por se comunicar com o computador, e enviar requisições para os outros módulos eletrônicos da plataforma. Essas requisições são feitas através dos quadros de mensagens remotas, em que não há o campo de dados. Essas requisições podem ser de verificação de avarias, se há ou não erros gravados nas outras ECUs ou para apagar a memória, ou seja, limpar as informações dos erros existentes.

A partir do envio das mensagens remotas os módulos que a recebem fazem a interpretação do seu endereço. Com o endereço interpretado então é possível saber qual o tipo de requisição que está sendo feita. Os quadros remotos de verificação de avarias foram enviados de forma separada para cada ECU e o de limpar memória para todas de uma vez. Essa relação entre o valor de endereço de requisição e a ECU correspondente pode ser visto na Tabela 7.

Tabela 7: Dicionário Dados Quadros Remotos

Requisição	ID DEC, HEX	ECU Rx	ECU Tx	DLC	RTR Bit	Tipo Frame
Apagar	1034, 0x40A	Todas	4	0	1	Padrão
Verificar	1114, 0x45A	ECU1	4	0	1	Padrão
Verificar	1290, 0x50A	ECU2	4	0	1	Padrão
Verificar	1370, 0x55A	ECU3	4	0	1	Padrão

Após as especificações dos DTCs e das mensagens de requisições foi possível definir o a matriz de comunicação contendo os dados de erros que serão transmitidos pelo barramento e recebidos pela quarta ECU. Com isso, Tabela 8 apresenta o dicionário de dados das mensagens que serão enviadas com as informações da memória de avarias.

Tabela 8: Dicionário de dados das mensagens com os erros

ECU Tx	ECU Rx	DTC	ID	DLC (Bytes)	Start Bit	Lenght Bits/Bytes
1	4	801	800	8	0	16/2
		802			16	16/2
		601			32	16/2
		602			48	16/2
2	4	701	900	8	0	16/2
		702			16	16/2
		501			32	16/2
		502			48	16/2
3	4	400	700	2	0	16/2

E por ultimo foi criado um aplicativo através do AppDesigner, que é uma extensão do Matlab. Esse aplicativo funciona como uma interface entre usuário e a plataforma de desenvolvimento. Ele foi feito de forma a simular um Scanner automotivo, com as opções de conexão, desconecção, verificar e apagar avarias. O aplicativo é responsável por se comunicar com a ECU4 e a partir de comandos fornecidos pelo usuário fazer as devidas requisições para a rede. Esse aplicativo pode ser visto na Figura 56.

Figura 56: Aplicativo Estudo 03

4.3.1.1 ECU1, ECU2 e ECU3

Com os requisitos do projeto definido foi possível criar os modelos para as ECUs 1, 2 e 3. Esse modelo base para os três módulos de monitoramento foi dividido em duas partes: a primeira consiste no monitoramento dos sensores e verificação dos seus valores, e quando há erro o seu código é gravado na memória do Arduino, esse modelo pode ser expresso através do diagrama na Figura 57.

A segunda parte do modelo consiste na recepção das requisições através do barramento CAN, e nas devidas ações tomadas conforme os comandos recebidos, o diagrama utilizado para desenvolver esse modelo pode ser visto na Figura 58. Com isso, se for para verificar a memória, a ECU faz uma leitura da sua memória permanente, e envia os valores lidos através do barramento. Contudo se o quadro

de requisição for para limpar a memória, a ECU executa um procedimento de apagar os dados que estão gravados.

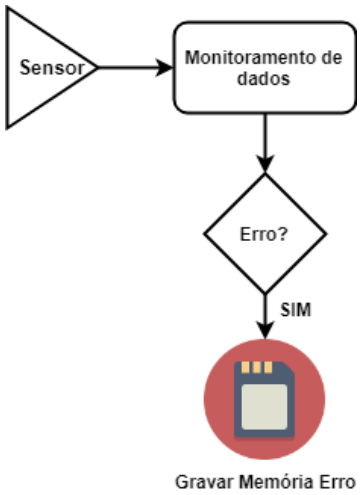

Figura 57: Diagrama verificação e monitoramento sensores

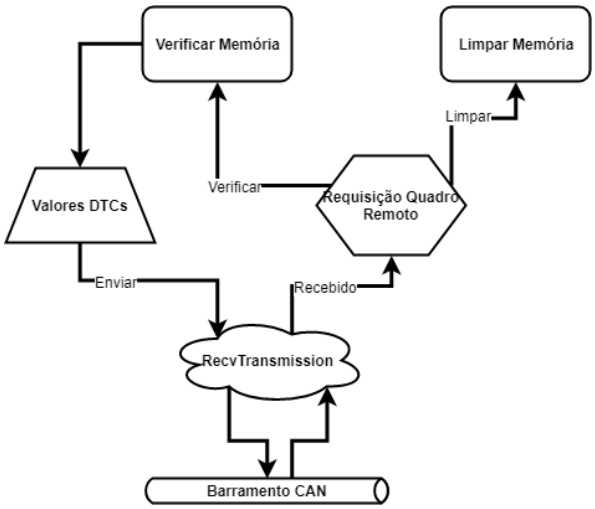

Figura 58: Diagrama de verificar ou limpar memória

4.3.1.2 ECU4

Além das três ECUs de monitoramento e análise dos sensores, tem-se a quarta, que é responsável pela interação entre o computador e a rede. O diagrama utilizado para o desenvolvimento do modelo dessa ECU foi feito como apresentado na Figura 59. Com isso, a ECU recebe um comando através do aplicativo pela sua conexão serial. A partir disso, ela interpreta esse comando e verifica se é para apagar ou verificar a memória de avarias. Se for para apagar ela envia para a rede um quadro de mensagem remota com o comando para as outras ECUs limparem a memória de avarias. Contudo se for para verificação, ela envia para as ECUs uma requisição dos valores de erros gravados em memória de cada uma. Após essa requisição ela recebe os valores de DTCs e os envia ao aplicativo para serem mostrados e analisados pelo usuário.

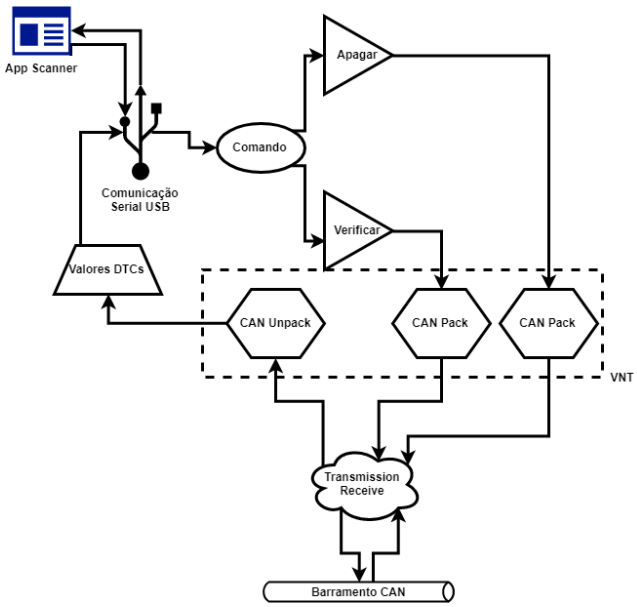

Figura 59: Diagrama Base para a ECU4

4.3.2 Análise dos Resultados

Após a finalização do projeto foi possível analisar o comportamento da rede e se houve a geração de erros conforme o especificado. Para isso todos os programas foram carregados nos respectivos Arduinos, não houve para esse caso a simulação em modo externo. Com isso, a parte da integração entre a placa e usuário foi feita através do aplicativo criado. O primeiro passo foi conectar esse aplicativo com o Arduino, como mostra a Figura 60, a luz verde indica que a conexão está feita.

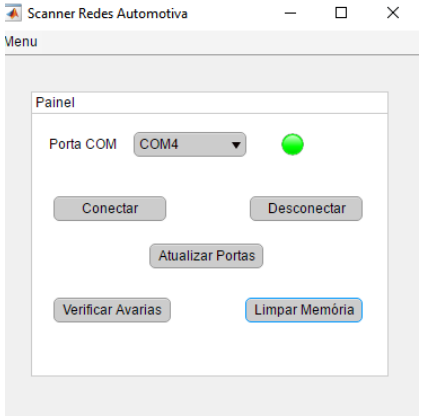

Figura 60: Scanner conectado a placa

Após essa conexão, então foi possível enviar algum dos comandos para a placa. O comando de verificar avarias foi o primeiro enviado, após esse envio o aplicativo abre uma nova janela para informar ao usuário se há ou não erros gravados em memória na rede. Essa janela com o primeiro teste, em que ainda não haviam avarias pode ser vista na Figura 61.

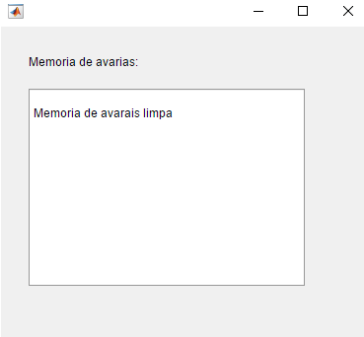

Figura 61: Memória de avarias Limpa

Após esse primeiro teste, foram gerados erros nas ECUs de forma progressiva. E em seguida a cada verificação, eles foram apagados para que fosse possível visualizar que diferentes erros fossem gerados por vez. Entretanto após a limpeza da memória, sempre que se fosse verificar novamente, era informada a mensagem de avarias limpa. E por último foi gerado todos os erros possível da rede para serem mostrados por meio do aplicativo. Essas aplicações podem ser vistas na Figura 62 e Figura 63.

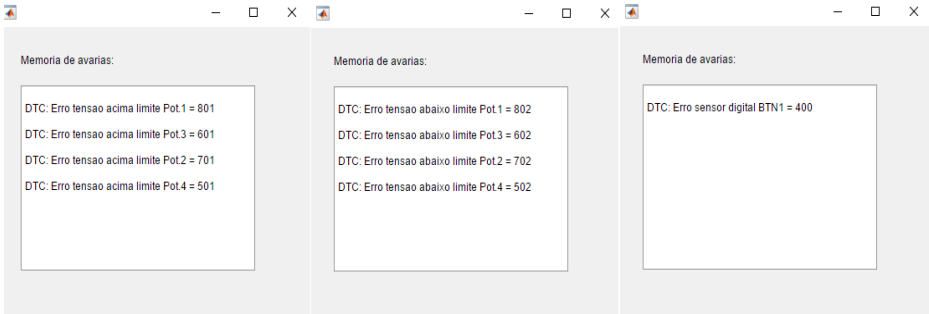

Figura 62: Visualização dos erros gerados de forma progressiva

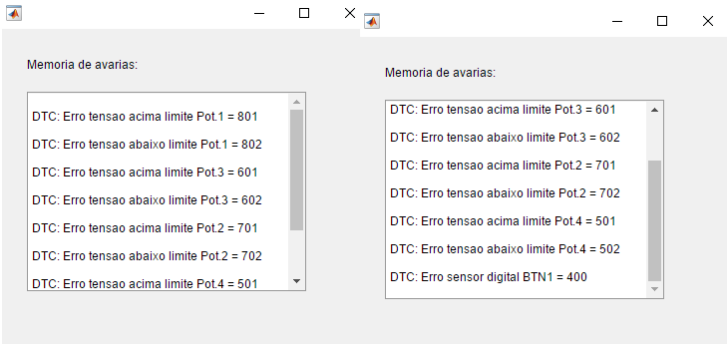

Figura 63: Visualização de todos os erros

Após essas análises, pode ser visualizado por meio do monitor serial o formato das mensagens de erros, seja com a memória de avarias limpa ou com erros presentes, isso pode ser visto na Figura 64. Por fim por não se tratar de um estudo de caso com mensagens periódicas, não houve uma análise estatística da rede. Contudo esse estudo serviu para validar o teste de monitoramento e análise de erros presentes nas ECUs.

```
Message ID: 800
Data
Byte1   Byte2   Byte3   Byte4   Byte5   Byte6   Byte7   Byte8
0       0       0       0       0       0       0       0
Message ID: 900
Data
Byte1   Byte2   Byte3   Byte4   Byte5   Byte6   Byte7   Byte8
0       0       0       0       0       0       0       0
Message ID: 700
Data
Byte1   Byte2   Byte3   Byte4   Byte5   Byte6   Byte7   Byte8
0       0
```

```
Message ID: 800
Data
Byte1   Byte2   Byte3   Byte4   Byte5   Byte6   Byte7   Byte8
33      3       34      3       89      2       90      2
Message ID: 900
Data
Byte1   Byte2   Byte3   Byte4   Byte5   Byte6   Byte7   Byte8
189     2       190     2       245     1       246     1
Message ID: 700
Data
Byte1   Byte2   Byte3   Byte4   Byte5   Byte6   Byte7   Byte8
144     1
```

Figura 64: Monitor Serial: Memória Limpa (E) e Presença de Erros (D)

4.4 ESTUDO DE CASO 04

O quarto estudo de caso foi desenvolvido visando o processo de calibração de uma ECU. Esse processo é realizado, pois durante o desenvolvimento do software de uma ECU existem basicamente duas etapas: o desenvolvimento do código, e a fase de calibração dos parâmetros da ECU. Com isso, a fase de desenvolvimento de código, consiste em programar uma aplicação para ser executada na ECU através de uma linguagem de programação (Vector, 2016).

Contudo, durante a fase de programação, existem alguns parâmetros no código da ECU, que são desenvolvidos para que possam ser alterados para valores mais específicos a depender da aplicação desejada. Com isso, essa alteração dos valores dos parâmetros consiste no processo de calibração da ECU. Além disso, com a calibração não são alterados os procedimentos executados pela ECU, apenas os valores específicos dos parâmetros (Etas, 2018).

A partir disso, esse estudo foi desenvolvido com a utilização de duas ECUs, uma responsável por enviar dados de calibração, ECU1, e outra a ser calibrada, ECU02. A primeira é responsável por comunicar com um aplicativo que permite a interação com o usuário para que se tenha o processo de calibração da ECU02. E a ECU02 é responsável por monitorar os dados de um sensor analógico, do qual simula um sensor de temperatura, e se comunica com o monitor serial para visualização do valor de temperatura. Com isso, a Figura 65 mostra o fluxograma desse estudo de caso.

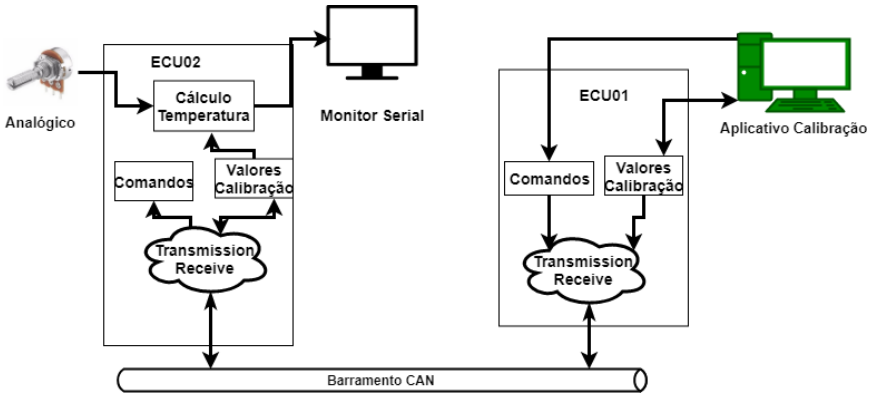

Figura 65: Fluxograma de ligação entre as duas ECUs

4.4.1 Projeto

Para o desenvolvimento desse projeto, foi utilizado o exemplo da Figura 66, extraída de Schauffele & Zurawka, 2003. Consiste na calibração dos parâmetros para o cálculo do valor de temperatura para diferentes escalas. Para esse calculo foi utilizada a Equação (4).

$$I = Q * V + O \tag{4}$$

Em que: I = Implementação, Q = Quantização, V = Valor lido do sensor e O = Offset. Com isso, o valor de quantização e offset são os parâmetros que podem ser calibrados para que haja uma mudança na escala de temperatura, como por exemplo, mostrado na Tabela 9, que mostra os valores utilizados para graus Celsius (°C), graus Fahrenheit (°F), e Kelvin (K).

Tabela 9: Tabela para Calibração e Conversão de Unidades

	Quantização	Offset
°C	1	0
°F	9/5 = 1.8	32
K	1	273

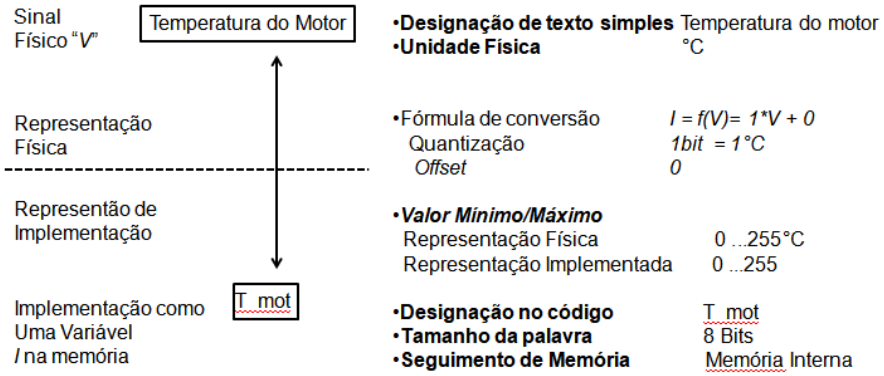

Figura 66: Especificação Física da temperatura do motor para implementação

A partir das definições, o projeto foi dividido da seguinte forma: A primeira ECU, é responsável pela comunicação com o computador e simula uma interface para o processo de calibração da segunda ECU. Com isso, essa interface faz a transmissão de comandos e valores dos parâmetros de calibração através de quatro quadros, dois remotos, um para envio do comando de verificação e outro para limpar a memória e dois quadros de dados com os valores de calibração, que são quantização (Quant) e *offset*.

E a segunda ECU é responsável por receber esses comandos e valores através do barramento CAN, e interpretá-los. Além disso, faz o monitoramento de um sensor analógico, um potenciômetro da placa, para emular um sensor de temperatura. Com isso, foi possível definir o dicionário de dados da rede, ele é mostrado na Tabela 10. E também, para duas ECUs, foi utilizado o módulo **MCP2515CAN_TransmitReceive,** pois há a necessidade tanto de transmissão quanto de recepção pelas duas ECUs.

Tabela 10: Dicionário de dados Estudo de Caso 04

	Mensagem	ID, DEC, Hex	ECU Tx	ECU Rx	RTR Bit	DLC	Start Bit	Length Bits/Bytes
1	Verificar	1370, 0x55A	1	2	1	0	*****	*****
2	Limpar	1034, 0x40A	1	2	1	0	*****	*****
3	Cal. Quant	1114,0x45A	1	2	0	2	0	16/.2
4	Cal. OffSet	1290, 0x50A	1	2	0	4	16	16/.2
5	Dado Quant.	267, 0x10B	2	1	0	2	0	16./2
6	Dado Offset	523, 0x20B	2	1	0	4	16	16/.2

Por fim, foi desenvolvido através do *AppDesigner* um aplicativo. Ele é responsável por enviar para a ECU1 os comandos e os valores dos dados de calibração. Esse aplicativo pode ser visto na Figura 67. Neles têm-se as principais funcionalidades: conectar, desconectar e atualizar dispositivos através de uma porta de comunicação serial. Também tem a opção de enviar o valor de quantização e o de offset, além das opções para verificação dos valores de calibração gravados na memória da segunda ECU e de limpar a memória, ou seja, apagar os parâmetros de calibração.

Figura 67: Aplicativo Estudo 04

4.4.1.1 ECU01

Com os requisitos definidos do projeto, a primeira ECU é responsável por integrar usuário, interface de calibração, e enviar os comandos através do barramento para a ECU a ser calibrada. O modelo de funcionamento desse módulo pode ser visto na Figura 68. Com isso, a partir do recebimento de um dos comandos, através da comunicação serial com o computador, a ECU o interpreta e faz o seu envio para a rede de comunicação.

Os comandos a serem enviados podem ser de limpar a memória, verificar, e de calibrar. Com isso, os quadros de limpar e verificar são quadros remotos, e ao se realizar o envio de verificar, a ECU espera receber os dados de quantização e *offset* que estão definidos na segunda ECU. E o comando de calibração, envia os dados de quantização e *offset*, que são necessários para que haja o processo de calibrar a segunda ECU.

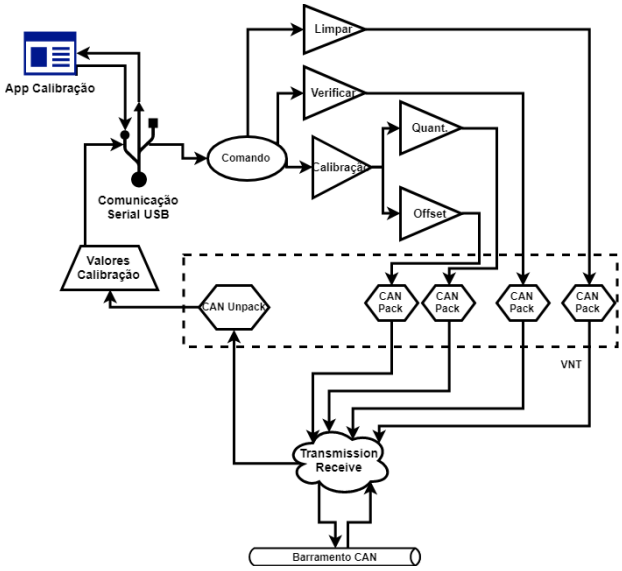

Figura 68: Fluxograma ECU01

4.4.1.2 ECU02

Após o desenvolvimento da ECU01 foi desenvolvido o modelo para a segunda, ela consiste na ECU que contém os parâmetros a serem calibrados. Com isso, foram desenvolvidos dois fluxogramas para projetar essa ECU. O primeiro pode ser visto na Figura 69, ele consiste na leitura dos valores presentes na memória e do valor do sensor para ser possível realizar o calculo do valor de temperatura, conforme a **Erro! Fonte de referência não encontrada.**, e após esse cálculo, o resultado é enviado para o monitor serial, para que possa ser visualizado pelo usuário.

Além disso, tem-se o segundo fluxograma, como mostrado na Figura 70, ele consiste no recebimento e transmissão das mensagens para a rede, além da gravação dos valores de quantização e offset na memória permanente da ECU. Com isso, após o recebimento de uma mensagem da rede, ela faz a separação conforme a sua especificação, ou seja, dependendo da mensagem recebida a ECU realiza as devidas ações.

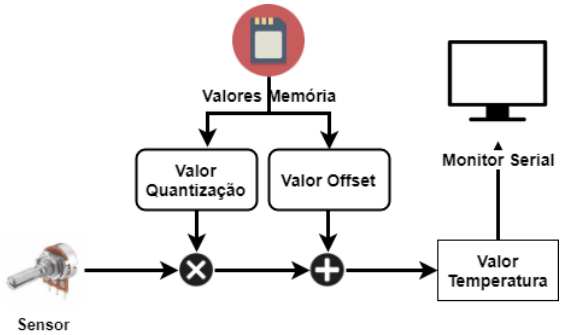

Figura 69: Fluxograma 1 para a ECU02

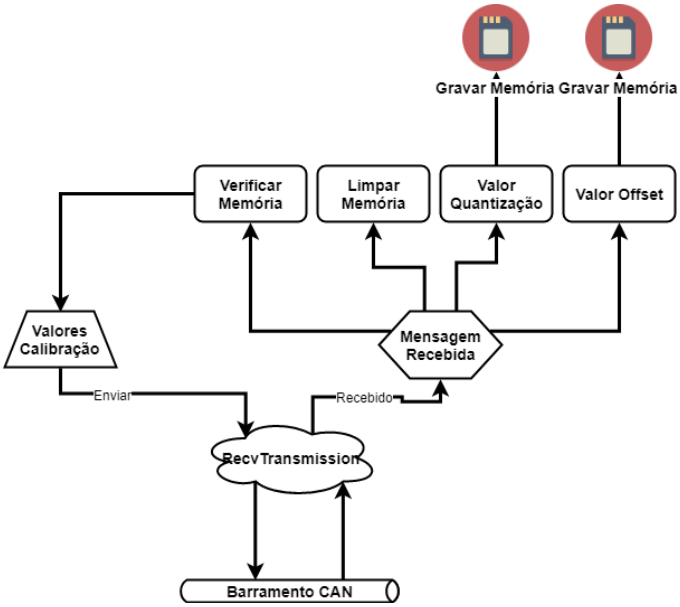

Figura 70: Fluxograma 2 para a ECU02

4.4.2 Análise dos Resultados

Após o desenvolvimento do projeto, os modelos do Simulink foram gravados nos microcontroladores, com isso, não houve simulação em modo externo. Logo após a gravação, o primeiro passo foi conectar a ECU01 ao aplicativo, como mostra a Figura 71. Após essa conexão foram transmitidos quatro valores de quantização e offset, como mostrado na mesma Figura. Após essa transmissão pode ser analisado como os valores de temperatura através do monitor serial, como mostra a Figura 72. Com isso, pode-se concluir que a ECU02 recebia esses dados e havia uma calibração na escala de temperatura, pois para cada valor recebido havia um novo valor de implementação para a ECU.

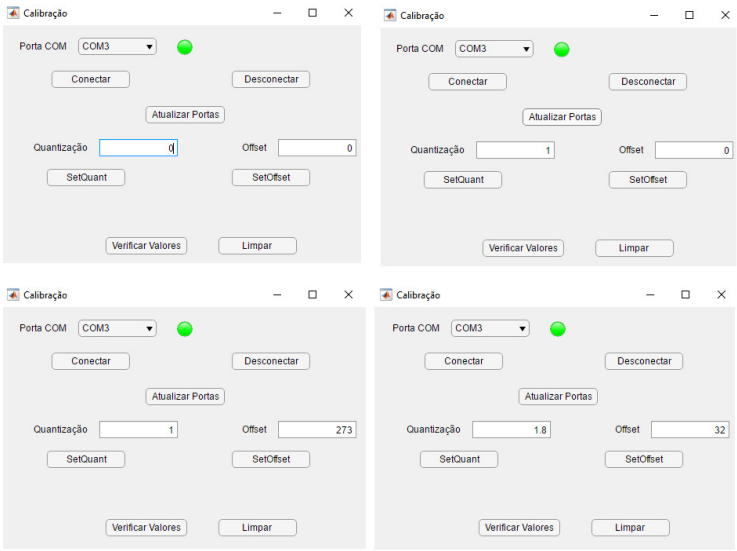

Figura 71: Telas Aplicativo com diferentes valores de Quantização e Offset

```
Temperatura = 0.00
Valor Sensor = 100
Temperatura = 0.00
Valor Sensor = 100
Temperatura = 0.00
Valor Sensor = 100
Temperatura = 0.00
Valor Sensor = 101
Temperatura = 0.00
Valor Sensor = 100
Temperatura = 0.00
Valor Sensor = 100
Temperatura = 0.00
```

Quant = 0 e Off = 0

```
Temperatura = 100.00
Valor Sensor = 100
Temperatura = 100.00
Valor Sensor = 100
Temperatura = 100.00
Valor Sensor = 101
Temperatura = 101.00
Valor Sensor = 100
Temperatura = 100.00
Valor Sensor = 100
Temperatura = 100.00
```

Quant. = 1 e Off = 0

```
Temperatura = 373.00
Valor Sensor = 100
Temperatura = 373.00
Valor Sensor = 100
Temperatura = 373.00
Valor Sensor = 100
Temperatura = 373.00
Valor Sensor = 100
Temperatura = 373.00
Valor Sensor = 100
Temperatura = 373.00
```

Quant. = 1 e Off = 273

```
Temperatura = 212.00
Valor Sensor = 100
Temperatura = 212.00
Valor Sensor = 100
Temperatura = 212.00
Valor Sensor = 100
Temperatura = 212.00
Valor Sensor = 100
Temperatura = 212.00
Valor Sensor = 100
Temperatura = 212.00
Valor Sensor = 100
Temperatura = 212.00
```

Quant. = 9/5 e Off = 32

Figura 72: Monitor Serial para os diferentes Valores de Quantização e Offset

Em seguida a essa análise de calibração, pode ser feita a verificação dos valores através do comando de verificar. Após o envio desse comando o aplicativo abre uma nova janela para visualização dos dados de quantização e offset, como mostra a Figura 73, que mostra para os valores de quantização igual a 0 e 9/5 = 1.8 e para valores de offset de 0 e 32, em que os valores de 1.8 e 32 são utilizado para conversão em Fahrenheit por exemplo. Isso mostrou que a ECU 02 gravava esses valores em memória e quando requisitada os enviava para a primeira ECU.

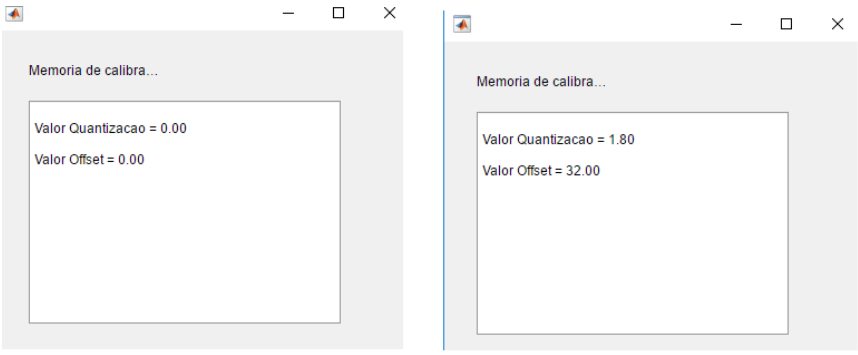

Figura 73: Visualização dos Valores de Calibração

Por fim, foi feito uma análise do formato das mensagens através do monitor serial, que serviu para validar o dicionário de dados, como mostra a Figura 74. Com isso, foi possível verificar que os quadros remotos não possuem o campo de dados, e a posição e o tamanho dos dados para as mensagens que contém os valores de calibração.

ID	Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
0x45A	0xB4	0x00						
0x50A	0x00	0x00	0x20	0x00				
0x55A								
0x40A								
0x10B	0xB4	0x00						
0x20B	0x00	0x00	0x20	0x00				

Figura 74: Monitor Serial Mensagens

4.5 ESTUDO DE CASO 05

Baseado nos princípios da dinâmica veicular, como mostra a Figura 75, ultimo estudo de caso foi desenvolvido de forma a simular o funcionamento de um veículo real. Foi projetado com a utilização de todos os quatro controladores da placa, onde cada um possui uma aplicação diferente e especifica. A Figura 76 mostra o modelo desenvolvido para esse estudo. A primeira ECU1, ICL, ficou responsável por receber os dados que estão trafegando na rede e se comunicar com um painel virtual para a sua visualização.

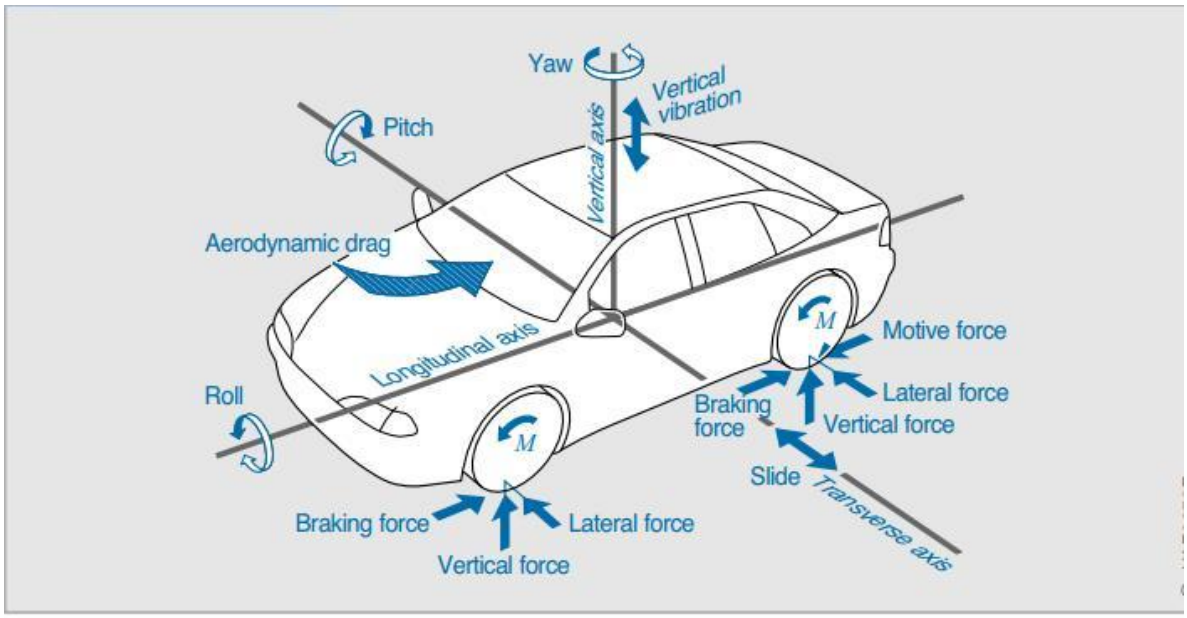

Figura 75: Forças que agem sobre um veículo

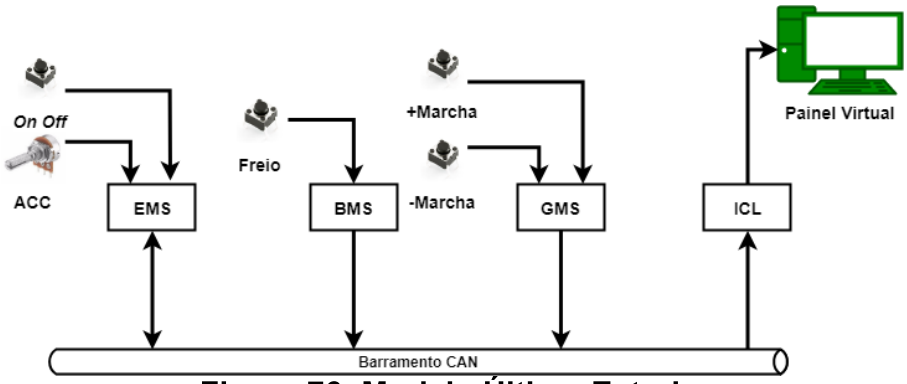

Figura 76: Modelo Último Estudo

O segundo microcontrolador, ECU2/GMS, é responsável por simular a troca de marchas presente em um veículo. O terceiro, ECU3/BMS, funciona com o monitoramento do freio e envia quando há ativação ou não. A quarta ECU, EMS, simula o sistema de controle do motor, onde estão presentes dados sobre rotação, consumo, temperatura, velocidade e recebe dados do sensor acelerador (ACC) que é representado por um potenciômetro resistivo presente na placa.

Com isso, a Tabela 11 apresenta o dicionário de dados definido para a rede. A definição desses dados foi baseada no padrão SAE J1939, como o tempo de

envio, e posição dos dados dentro do quadro de mensagens e o seu endereçamento. Os endereços foram baseados da seguinte forma em hexadecimal (HEX) e transformados em decimal (Dec): 0xAAH, onde AA é uma forma a dizer a qual parâmetro a mensagem pertence e o H é o endereço de hardware (HID) , para indicar qual ECU faz a transmissão através do barramento.

Além disso, o dicionário apresenta o DLC dado em bytes, o tamanho do dado, ou seja, a quantidade de bytes que os dados especificados ocuparão na mensagem, essa distinção entre o DLC e o tamanho do dado foi feita, pois em alguns casos o conteúdo da mensagem não se inicia obrigatoriamente no primeiro bit do campo de dados. Isso pode ser visto através das colunas *Start Bit* e *Length Bits*, que indica o bit de início do dado, e o seu comprimento em bits. E também apresenta o tempo de envio e as ECUs transmissoras e receptoras. Por fim, Figura 77 apresenta a implementação em hardware do último estudo.

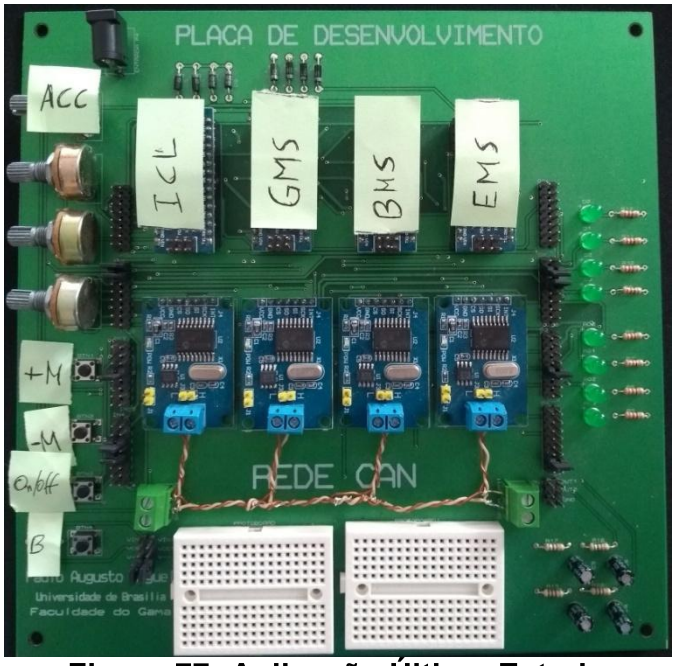

Figura 77: Aplicação Último Estudo

4.4.1 Projeto

Esse projeto consiste no desenvolvimento de uma rede para simular o processo de dinâmica veicular, e possui quatro ECUs. Cada ECU será descrita separadamente, conforme a sua aplicação e o seu desenvolvimento. Primeiro será explicado o funcionamento sobre a EMS, pois é a mais complexa e a com mais aplicações. Segundo sobre a GMS e o seu controle da marcha atual. Terceiro sobre a BMS e como ela envia os dados sobre ativação ou não do freio. E por ultimo sobre a ICL e a sua comunicação com o computador via Serial.

Tabela 11: Dicionário de dados do Último Estudo de Caso

Mensagem	ID(Dec, HEX)	DLC(Bytes)	Tamanho Dado (Bytes)	Start Bit	Length Bits	Tempo Envio(ms)	ECU Tx	ECU Rx
RPM	72, 0x048	5	2	24	16	20	EMS	ICL
Pos. Pedal	56, 0x038	2	1	8	8	50	EMS	ICL
Temperatura	2024, 0x7E8	4	2	16	16	1000	EMS	ICL
Velocidade	504, 0x1F8	3	2	8	16	100	EMS	ICL
Distancia	232, 0x0E8	4	4	0	32	100	EMS	ICL
Nível Combustível	1928, 0x788	5	1	32	8	1000	EMS	ICL
Freio	505, 1F9	1	1	6	2	100	BMS	EMS/ICL
Marcha Atual	1530, 0x5FA	4	1	24	8	100	GMS	EMS/ICL

4.4.1.1 EMS – Engine Management System

A *"Engine Management System"* (EMS – Sistema de Gerenciamento do Motor) faz o monitoramento da temperatura, posição do pedal do acelerador, além de calcular a velocidade, rotações por minuto (RPM), distância percorrida e o nível de combustível. Com isso, ela é uma ECU responsável por grande parte dos cálculos dinâmicos presente um veículo. Além disso, possui o HID igual a 8, ou seja, o dados presentes na Tabela 11 com esse endereço são provenientes dessa ECU. A Figura 78 apresenta o diagrama utilizado para a construção dessa ECU.

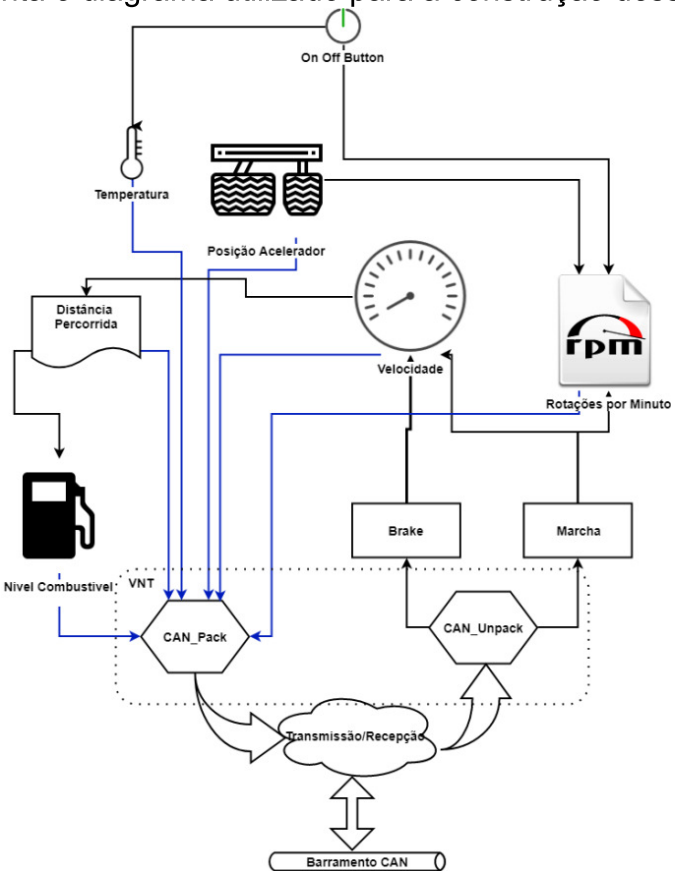

Figura 78: Diagrama Funcionamento SEM

Tanto para envio quanto para recebimento, foi utilizado em todo esta ECU o bloco **MCP2515CAN_TransmissionReceive,** adicionado de dois ativadores, um para envio e outro para recebimento. Quando é necessário o envio, a ativado o *activeSend* e para o recebimento o *activeReceive*. Isso foi necessário para que não houvesse conflito entre a parte de enviar e receber os dados. Pois para essa aplicação será necessário tanto envio quanto recepção de mensagens através do barramento CAN.

Com isso, os dados de marcha atual e ativação de freio são recebidos pela parte de leitura da rede CAN. Esses dados são importantes, pois alteram as condições dos cálculos dinâmicos de um veículo. O freio ativado indica que há uma força que dificulta o movimento, o que ocasiona a redução do valor de sua velocidade. A marcha atual ao possuir valor diferente de zero, indica que há o início de um movimento, ou seja, o seu valor depende para os cálculos do RPM e da velocidade.

A partir da definição da parte de recepção, foram feitos as outras partes do modelo. O primeiro passo foi o desenvolvimento do bloco de controle do *On Off* Button, ou botão liga desliga, que simula o ato de ligar ou desligar o motor de um carro. Isso de deve, pois muitos cálculos dependem do estado atual de funcionamento do motor. Quando esse botão é ativado, torna possíveis os cálculos de RPM, velocidade, distância, temperatura e nível de combústivel.

Após a ativação do botão liga desliga, o valor da temperatura começa a incrementar de forma progressiva, e quando há a desativação ela começa a decrementar, como no processo de aquecimento e resfriamento. A próxima parte consiste no cálculo em porcentagem da posição do pedal do acelerador. Para isso foi utilizado um potenciômetro da placa, que simula o sensor analógico responsável por informar a posição do pedal. Com isso, após a ativação do motor e do valor da posição do pedal do acelerador, é possível realizar o procedimento para o cálculo em RPM (rotações por minuto) do motor.

Contudo, além do cálculo das rotações, esse bloco também realiza as operações para obter o valor da velocidade. Com isso, outros dois dados são importantes para essa parte, o valor da marcha e o dado de ativação do freio. Isso se deve, pois ao se ter um valor diferente de zero para a marcha há o início do movimento, e para os seus valores específicos têm-se valores máximos de rotações e de velocidade. E o valor de ativação do freio, irá influenciar de forma direta a velocidade, pois quando ele é ativado ocasiona redução de velocidade e consequentemente o valor das rotações.

Com isso, após um carro sair do seu estado de repouso, ele começa a ter uma distancia percorrida em decorrência da sua velocidade e do tempo de deslocamento. Por isso, o valor atual da velocidade é necessário para o calculo da distância, que é fornecido em quilômetros, e para o cálculo do consumo de combustível

4.4.1.2 GMS – Gear Management System

Outra ECU presente no estudo de caso é a *"Gear Management System"* (GMS – Sistema de gerenciamento de Marcha) é a ECU responsável por gerenciar a troca e indicar a marcha atual, ou seja, responsável pelo controle da transmissão de um veículo, e enviar o valor atual da marcha a cada 100ms. Ela possui o HID igual a A em HEX, ou 10 em decimal. A Figura 79 mostra o diagrama utilizado para o desenvolvimento do modelo desse controlador.

A partir disso, ela utiliza dois botões da placa, um para aumentar e outro para diminuir a marcha, como uma troca sequencial. Com isso, possui o valor mínimo de zero e o valor máximo de cinco, ou seja, mesmo que algum dos botões seja pressionado mais vezes, não haverá valores maiores ou menores.Além disso, quando cada botão é pressionado, um led é aceso indicando a respectiva ação que foi tomada. Por fim, essa ECU não precisa receber, mas apenas transmitir a mensagem através do barramento CAN. Por isso, foi utilizado o bloco **MCP2515CAN_Transmit**, que é o responsável pelo envio das mensagens.

Figura 79: Diagrama modelo da GMS

4.4.1.3 BMS – Brake Management System

Outro módulo presente no projeto foi a *"Brake Management System"* (BMS – Sistema de Gerenciamento do Freio) cuja responsabilidade é enviar através do barramento CAN se o freio está acionado ou não, esse envio é realizado a cada 100ms. Essa ECU em veículos reais, também é responsável por realizar controle do sistema antitravamento dos freios e por fazer a leitura da velocidade em cada roda de um veículo. A Figura 80 mostra o diagrama utilizado para representar o funcionamento desta ECU.

Com isso, um dos botões da placa foi definido como o acionamento do freio e também foi utilizado um led na saída para indicar quando há a ativação do botão. Outra especificação é que a GMS possui o HID igual a 9 Assim como a GMS ela não necessita de receber, apenas de transmitir através do barramento, por isso foi utilizado novamente o bloco **MCP2515CAN_Transmt**.

4.4.1.4 ICL – Instrument Cluster

Por fim, tem se o desenvolvimento da *"Instrument Cluster"* (ICL – Painel de instrumento), ela é a ECU responsável por receber todos os dados da rede e se comunicar com o computador. Ao receber esses dados, eles são separados de forma a indicar os valores que são recebidos. Após a recepção, eles são enviados via porta USB Serial para o computador para serem mostrados em um painel virtual, do qual o software foi fornecido pelo professor, mas toda a comunicação foi desenvolvida no projeto da ICL. A Figura 81 mostra o diagrama utilizado para a construção do modelo dessa ECU.

Além isso, esse módulo necessita apenas receber as mensagens provenientes do barramento CAN, por isso foi necessário o uso do **MCP2515CAN_Receive**, e logo após a recepção, o quadro de mensagens CAN é enviado aos blocos CAN Unpack. Esses blocos fazem a separação das mensagens conforme as especificações. Após essa separação eles são enviados para o bloco de comunicação com o painel virtual, em que os dados provenientes das outras

ECUs poderão ser visualizados. Por fim, a Figura 82 mostra o painel virtual utilizado para esse estudo de caso.

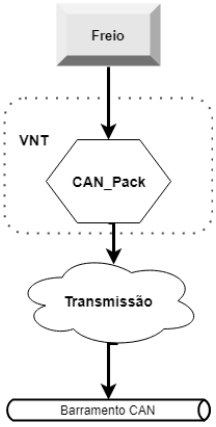

Figura 80: Diagrama modelo da BMS

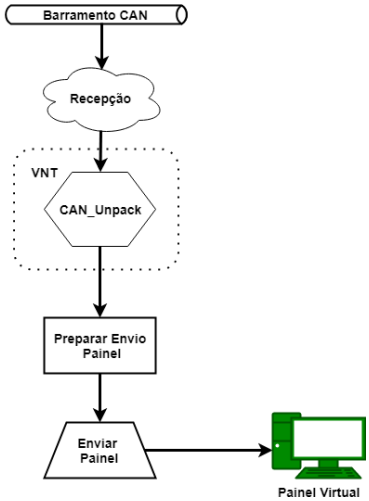

Figura 81: Modelo ICL

4.4.2 Análise dos Resultados

Após a finalização do projeto, os códigos foram carregados em seus respectivos microcontroladores, para esse estudo de caso não houve simulação com o modo de fonte externa. Com isso a comunicação com o computador se deu de forma serial através da ICL e o painel virtual. A primeira análise foi feita através da comunicação com o painel digital, e a Figura 83 mostra a sua tela inicial antes de ser pressionado o botão liga desliga, mas já se comunicando com a ICL.

Ao se pressionar o botão *On Off* foi visto a variação dos dados que simulam um motor de um veiculo ligado. A partir desse ponto o RPM assume valores conforme a posição do pedal do acelerador, possuindo um valor mínimo de 600RPM. Também é possível ver o nível de combustível simulando um tanque cheio, e a variação da temperatura conforme aumenta o tempo de execução da rede. A Figura 84 mostra o painel logo após a ativação do botão liga desliga.

Figura 82: Painel Virtual

Figura 83: Início aquisição de dados

Conforme essa aquisição iniciada simulando um motor ligado, foi feito uma alteração de incremento do valor da marcha. A partir dessa mudança é possível simular um veículo em movimento, com a velocidade crescente a media em que se altera o valor da marcha, possuindo um valor máximo de 233km/h. Como possui velocidade diferente de zero há então o deslocamento em distância, e consequentemente a redução do nível de combustível. Essa análise de movimento pode ser vista na Figura 85 em que a esquerda (E) tem se a aquisição assim que se da o inicio do movimento, e a direita (D) após duas horas de execução.

Além da análise através do painel foi feita uma análise estatística das mensagens presentes no projeto dessa arquitetura eletroeletrônica. Ela análise foi feita comparando os tempos especificados de projeto e o tempo aferido na recepção de cada mensagem, ela na Tabela 12. Primeiro é possível ver que a BMS e GMS

conseguem realizar as suas operações com um tempo médio equivalente ao tempo especificado, havendo inclusive uma leve queda em porcentagem do tempo.

Figura 84: Aquisição após ativação

Figura 85: Aquisição após inicio movimento (E) e após certo tempo de execução (D)

Diferentemente da BMS e GMS a EMS possui um código de carregamento e um número de operações mais elevado. Isso ocasionou um aumento do intervalo de tempo entre o esperado e o medido. Essa alteração é influenciada pelo número elevado de funções presentes no projeto da ECU. Isso ocorre, pois cada função necessita ser executado por certo intervalo de tempo, além disso, a EMS necessita da recepção de mensagens através do barramento.

Com esse número elevado de tarefas, foi visto que o programa carregado ocupou uma elevada porcentagem da memória do Arduino Nano, o que quase o tornou inviável para essa aplicação. Com isso, para aplicações com um grau mais elevado de complexidade utilizando esses modelos de projetos seriam necessários microcontroladores com maior processamento e memória. Entretanto, para essa aplicação foi visto que a rede obteve comportamento adequado, mesmo com essa diferença no tempo de execução da EMS. E por último a Figura 86 mostra a tela do monitor serial com algumas das mensagens presentes na rede.

Tabela 12: Dados Estatísticos do Último Estudo

Mensagem	ID	Intervalo Tempo (ms)	Média (ms)	Desvio Padrão	Número de Amostras	Porcentagem Média/Esperado -100 %	ECU
Marcha	1530	100	99,89	0.70	591	-0,11	GMS
Freio	505	100	99,94	0.69	595	-0,06	BMS
Velocidade	504	100	166,59	20.58	699	66,59	EMS
Pos. Pedal	56	50	81,71	19.61	985	63,42	EMS
Temperatura	2024	1000	1690,82	4.67	78	69,082	EMS
RPM	72	20	33,66	9.41	2436	68,3	EMS
Distancia	232	100	168,21	12.53	1889	68,21	EMS
Combustível	1928	1000	1690,36	3.91	198	69,036	EMS

```
Message ID: 56
Data
Byte1   Byte2   Byte3   Byte4   Byte5   Byte6   Byte7   Byte8
0       60
Message ID: 72
Data
Byte1   Byte2   Byte3   Byte4   Byte5   Byte6   Byte7   Byte8
0       0       0       59      15
Message ID: 504
Data
Byte1   Byte2   Byte3   Byte4   Byte5   Byte6   Byte7   Byte8
0       233     0
Message ID: 232
Data
Byte1   Byte2   Byte3   Byte4   Byte5   Byte6   Byte7   Byte8
97      1       0       0
Message ID: 72
Data
Byte1   Byte2   Byte3   Byte4   Byte5   Byte6   Byte7   Byte8
0       0       0       59      15
Message ID: 1530
Data
Byte1   Byte2   Byte3   Byte4   Byte5   Byte6   Byte7   Byte8
0       0       0       5
Message ID: 505
Data
Byte1   Byte2   Byte3   Byte4   Byte5   Byte6   Byte7   Byte8
0
Message ID: 72
Data
Byte1   Byte2   Byte3   Byte4   Byte5   Byte6   Byte7   Byte8
0       0       0       59      15
```

Figura 86: Monitor Serial Mensagens do Ultimo Estudo

5 CONCLUSÃO

Com o avanço da tecnologia a indústria automotiva está sempre buscando formas de melhorar a eficiência dos veículos. O uso da eletroeletrônica embarcada cresceu de forma considerável. Com isso, surgiu o uso das redes de comunicação automotivas. Ela fez com que a produção de inúmeros sistemas fosse descentralizada simplificando os projetos e até mesmo diminuindo tempo de produção. Contudo esses projetos ainda são uma tarefa desafiadora, por isso tem se difundido cada vez mais a aplicação do MBD.

Como esses sistemas eletroeletrônicos interligados possuem um alto grau de complexidade, o estudo de redes automotivas é uma tarefa que inclui vários desafios, como por exemplo, o alto custo dos hardwares próprios para comunicação CAN. Por esse motivo esse trabalho surgiu com o intuito de auxiliar o aprendizado no processo de desenvolvimento de projetos envolvendo arquiteturas eletroeletrônicas com a utilização do protocolo de comunicação CAN, possuindo um baixo custo.

Por isso, foram realizados os estudos de caso, que visavam verificar e validar a funcionalidade da plataforma. Além disso, foram meios de entender os conceitos e aplicações de projetos envolvendo arquiteturas eletroeletrônicas com a aplicação do protocolo CAN, além de entender e aplicar conceitos do MBD. Isso se deu, com o reaproveitamento de software através dos modelos gerados no Simulink, e com a construção de um único protótipo serviu para diferentes aplicações.

Com o propósito de testar e validar esse projeto, foram feitos quatro estudos de caso. O primeiro serviu para validar a comunicação entre os módulos, o que garantiu o correto funcionamento da plataforma e a periodicidade na transmissão da mensagem de dados. Com isso, o segundo mostrou como é possível haver mais de uma mensagem em trafego na rede, o que confere com as próprias características do barramento.

Em seguida o terceiro estudo de caso serviu para validar uma aplicação real de uma rede de comunicação CAN. Foi feito o intuito de realizar o monitoramento e verificação de erro em sensores, como acontece em um veículo. Isso se deve, pois, a própria rede necessita do seu auto diagnóstico para informar ao usuário que algo não está conforme o especificado. Isso acontece, por exemplo, quando é acesa a luz de indicação no painel de um carro, para que o usuário procure uma assistência técnica. Logo em seguida, o quarto estudo de caso serviu para validar o processo de calibração de uma ECU, o que também ocorre para inúmeras aplicações.

Por último teve o estudo da dinâmica veicular, para mostrar mais próximo de uma arquitetura eletroeletrônica de um veículo real. Embora no geral o a rede possua um bom comportamento, foi verificado que o Arduino Nano com o modelo gerado no Simulink possuiu uma alta defasagem, na transmissão das mensagens, entre o intervalo de tempo especificado e o real medido para a aplicação da EMS. Contudo para a GMS e BMS as transmissões ocorreram dentro do tempo especificado.

Com isso, após o desenvolvimento dos projetos, a plataforma obteve bom desempenho quanto à aplicação das arquiteturas eletroeletrônicas envolvendo o protocolo de comunicação CAN. Contudo, o hardware selecionado possui algumas limitações, dentre elas, uma memória com baixa capacidade para o código e dados, assim como a baixa frequência do relógio para o Atmega328p e o MCP2515 o que para sistemas maiores e mais complexos como a EMS pode ocasionar problemas na execução das tarefas. Além disso, diversas aplicações podem estar sujeitas a condições de corridas, ou seja, quando a ordem do acesso a uma determinada tarefa pode afetar o resultado das operações, ou o seu tempo de execução, como no caso da EMS novamente.

Por fim, a plataforma atendeu os seus requisitos para o qual foi confeccionada, com a ressalva para sistemas mais complexos. Visto isso, para trabalhos futuros os dispositivos eletroeletrônicos utilizados podem ser substituídos por hardwares com maior poder de processamento e com mais recursos, tanto de memória quanto a quantidade de entradas e saídas. Isso se deve para que seja possível criar arquiteturas eletroeletrônicas mais complexas e com mais aplicações.

Bibliografia

Aaresntrup, R. (2010). *Managin Model-Based Design.* The MathWorks, Inc.

Ataide, F. H., & Pereira, C. E. (2012). FTT-CAN - Estudo de Caso em Aplicação Automotiva.

Baleani, M., Ferrari, A., Mangeruca, L., Sangiovanni-Vincentelli, A., Freund, U., Schlenker, E., et al. (2005). Correct-by-Construction Transformations across Design Environments for Model-Based Embedded Software Development. *Proceeding of Design, Automation and Test I Europe Conference and Exhibition* (p. 6). IEEE.

Banks, J. (1998). *Handbook of Simulation: Principles, Methodology, Advances, Application, and Pratice.* John Wiley & Sons, Inc.

Bosch GmbH, R. (2007). *Automotive handbook.* Bentley Publishers.

Bosch GmbH, R. (2007). *Bosh Automotive Electrics and Automotive Electronics.* Springer Vieweg.

Bosch GmbH, R. (1991). *CAN Specification Version 2.0.* Stuttgart.

Dai, S., Zhang, Z., & Koutsoukos, X. (2017). A model-based design approach for simulation and virtual prototyping of automotive control systems using por-Hamiltonian systems. *Springer* , 17.

Denton, T. (2012). *Automobile electrical and electronic systems.* Routledge.

Ding, S. X. (2008). *Model-Based Fault Diagnosis Techniques: Design Schemes, Algorithms and Tools* . Springer.

Etas. (2018). *Calibrating automotive electronics*. Retrieved Maio 15, 2019, from Etas: https://www.etas.com/en/applications/applications_calibrating_automotive_electronics.php

Fredriksson, L.-B. (1994). Controller Area Networks and the Protocol CAN for Machine Systems. *Mechatronics* , 13.

Guimarães, A. d. (2007). *Eletrônica Embarcada Automotiva.* Erica LTDA.

Guimarães, A. d., & Saraiva, A. M. (2002). O protocolo CAN: Entendendo e implementando uma rede de comunicação serial de dados baseada no barramento 'Controller Area Network'. *SAE Inc* .

Hallestrand, G. (2005). *ESL development get a leg up*. Retrieved 03 15, 2019, from Electronic Systems Design Engineering: http://chipdesignmag.com/display.php?articleId=57

Hora, J., & Campos, P. (2015, 10). A review of performance criteria to validate simulation models. *Expert Systems* , p. 18.

Hoxha, B., Dokhanchi, A., & Fainekos, G. (2017). Mining parametric temporal logic properties in model-based design for cyber-physical systems. 15.

ISO. (2015). *Road vehicles — Controller area network (CAN) — Part 1: Data link layer and physical signalling*. Retrieved 03 10, 2019, from ISO 11898-1:2015(en): https://www.iso.org/obp/ui#iso:std:iso:11898:-1:ed-2:v1:en

Labcenter. (2019). *Proteus*. Retrieved 03 15, 2019, from Labcenter Proteus Design Suit: https://www.labcenter.com/

Mathwork, M. &. (2015). *Vehicle Network Toolbox, User's Guide.* The Mathworks, Inc.

Mathworks. (2019). *Introducing MEX Files*. Retrieved 03 15, 2019, from Matlab: https://www.mathworks.com/help/matlab/matlab_external/introducing-mex-files.html

Mathworks. (2019). *Mathworks*. Retrieved 02 20, 2019, from Matlab/Simulink: https://www.mathworks.com/

Mathworks. (2019). *S-Function Builder*. Retrieved 03 15, 2019, from Matlab: https://www.mathworks.com/help/simulink/slref/sfunctionbuilder.html

Mathworks. (2014). Why Adopt Model-Based Design for Embedded Control Software Development?

Nicolescu, G., & Mosterman, P. J. (2010). *Model-Based Design for Embedded System.* CRC Press.

Paret, D. (2005). *Multiplexed Netorks for Embedded Systemns.* Jonh Wiley & Sons Ltd.

Parteno, F. (2000). *Model-Based Design and Evaluation of Interactive Application.* Springer.

Reif, K. (2014). *Brakes, Brake Control and Driver Assistence Systems: Functions, Regulation and Components.* Springer.

Santos, M. M. (2010). *Redes de Comunicação Automotiva: Características, Tecnologias e Aplicações.* Érica/Saraiva.

Satamaria, D., Alarcón, F., Jiménez, A., Viguria, A., Béjar, M., & Ollero, A. (2011). Model-Based Design, Development and Validation for UAS Critical Software. *Springer Science* , 12.

Schätz, B., & Krcmar, H. (2011). What is the benefit of a model-base design of embedded software system in the car industry? 24.

Schauffele, J., & Zurawka, T. (2003). *Automotive Software Engeeering Priciples, Processes, Methods, AND Tools.* SAE Internation.

Vector. (n.d.). *Why do we need calibration?* Retrieved Maio 15, 2019, from 2016: https://assets.vector.com/cms/content/application-areas/ecu-calibration/Docs/Why_do_we_need_Calibration.pdf

Vincentelli, A. S., & Natale, M. D. (2007). Embedded systems for automotive application. *IEEE Computer* , 9.

Wu, Y.-J., & Chung, J.-G. (2014). Efficient Controller Area Netowor Data Compression for automobile applications. *Inform Technol Eletron* , 8.