CENTRO PAULA SOUZA
FACULDADE DE TECNOLOGIA DE SANTO ANDRÉ
TECNOLOGIA EM ELETRÔNICA AUTOMOTIVA

ANTONIO APARECIDO ARIZA CASTILHO
BRUNO RAFAEL BERDUSCO

IMPLEMENTAÇÃO DE INTERFACE DE COMUNICAÇÃO PC – REDE CAN

Santo André
2021

ANTONIO APARECIDO ARIZA CASTILHO

BRUNO RAFAEL BERDUSCO

INTERFACE DE COMUNICAÇÃO PC – REDE CAN

Trabalho de Conclusão de Curso apresentado à Fatec Santo André, como requisito parcial para obtenção do título de Tecnólogo em Eletrônica Automotiva, orientado pelo Prof. Me. Weslley Medeiros Torres, e coorientado pelo Eng. Elton Inacio Alves Junior.

Santo André

2021

FICHA CATALOGRÁFICA

C352i

Castilho, Antonio Aparecido Ariza

Implementação de interface de comunicação PC-Rede Can / Antonio Aparecido Ariza Castilho, Bruno Rafael Berdusco. - Santo André, 2021. – 50f: il.

Trabalho de Conclusão de Curso – FATEC Santo André. Curso de Tecnologia em Eletrônica Automotiva, 2021.

Orientador: Prof. Weslley Medeiros Torres

1. Eletrônica embarcada. 2. Sensores. 3. Atuadores. 4. Rede CAN. 5. Tecnologia. 6. Sistemas embarcados. 7. Veículos. 8. Eletrônica automotiva. 9. Transceptor. 10. Protocolos de comunicação. 11. Software. I. Berdusco, Bruno Rafael. II. Implementação de interface de comunicação PC-Rede Can.

629.2

ANTONIO APARECIDO ARIZA CASTILHO

BRUNO RAFAEL BERDUSCO

INTERFACE DE COMUNICAÇÃO PC – REDE CAN

Trabalho de Conclusão de Curso apresentado a FATEC SANTO ANDRÉ como requisito parcial à obtenção de título de Tecnólogo em Eletrônica Automotiva.

Professor Orientador
Prof. Dr. Weslley Medeiros Torres

Professor Coorientador
Eng. Elton Inacio Alves Junior

MEMBROS DA BANCA EXAMINADORA

Presidente da Banca
Prof. Me. Weslley Medeiros Torres
Fatec Santo André

Primeiro membro da Banca
Prof. Dr. Fernando Dalbo Garup
Fatec Santo André

Segundo Membro da Banca
Eng. Elton Inacio Alves Junior.
Fatec Santo André

Local: Fatec Santo André
Horário: 10:00
Data: 15/12/2021

Santo André

2021

Dedico este trabalho a minha esposa Adriana de Simone, a minha mãe Alairce Berdusco e a minha Família que tiveram compreensão quanto a minha ausência em determinados eventos.

Bruno Rafael Berdusco

Dedico este trabalho a minha querida e amada esposa, Eliana Genoveza.

Antonio Aparecido Ariza Castilho

AGRADECIMENTOS

Agradecemos a Deus, por nos manter com saúde e firmes em nosso propósito.

À nossa família, pelo apoio e compreensão, que permitiu, um sentimento de tranquilidade para realizar o trabalho.

Ao professor Weslley Medeiros Torres, que nos orientou com muito empenho, colocando a nossa disposição uma imensidão de conhecimentos e ao Eng. Elton Inacio Alves Junior, pela ajuda no entendimento das técnicas programação e as boas dicas para o êxito do trabalho.

Na pessoa do professor Fernando Dalbo Garup, agradecemos a todos os professores da Fatec, que sempre nos incentivaram, mostrando que com trabalho e dedicação todos os sonhos se realizam.

Ao diretor da FATEC Santo André, professor Alexsander Tressinode Carvalho e a todos os funcionários, que trabalham eficientemente para que tudo funcione na faculdade, proporcionando uma vida acadêmica plena e intensa.

Ao professor Carlos Alberto Morioka, coordenador do curso, que sempre atento, ajudou nessa caminhada, resolvendo os problemas, sempre da melhor forma.

Nossos sinceros agradecimentos.

“Uma vez fatecano, sempre fatecano.”
Antonio Aparecido Ariza Castilho

“Quando a necessidade substitui a verdade, o mal se torna banal”
Hannah Arendt

RESUMO

A Rede CAN é uma realidade nos veículos e sistemas com Eletrônica Embarcada, permitindo o controle com eficiência e economia de recursos e materiais. As informações que circulam no barramento CAN, são provenientes dos eventos gerados e processados em unidades de controle dedicadas, através da leitura de sensores e comando de atuadores de controle de componentes chaves, para obter os melhores resultados, durante o funcionamento dos sistemas. Essas informações estão disponíveis na rede CAN, podendo ser lidas por qualquer unidade que a requisite. Podendo também, serem extraídas por ferramentas apropriadas a partir do conhecimento dos protocolos de comunicação utilizados.

Este trabalho implementou um barramento CAN, a partir de dois PIC18F4550 e dois *transceivers* (MCP2515, TJA1050). Também, foi desenvolvido um *software* para PC e os *firmwares* necessários aos PIC's, para verificar e gerar tráfego no barramento.

Palavras-chave> CAN. Ferramentas CAN. PIC18F4550. MCP2515. Transceptor. Eletrônica Embarcada. Sistemas Embarcados. Eletrônica Automotiva.

ABSTRACT

The CAN Network is a reality in vehicles and systems with Embedded Electronics, allowing an efficient control and saving of resources and materials. The information that circulates on the CAN bus comes from events generated and processed in dedicated control units, by means of sensor reading and command of key component control actuators, in order to obtain the best results during system operation. This information is available on the CAN network and can be read by any unit that requests it. They can also be extracted by appropriate tools from knowledge of the communication protocols used.

This work implemented a CAN bus, from two PIC18F4550 and two transceivers (MCP2515, TJA1050). In addition, PC software and the necessary firmware for the PIC's were developed to verify and generate traffic on the bus.

Keywords> CAN. CAN Tools. PIC18F4550. MCP2515. Transceiver. Embedded Electronics. Embedded systems. Automotive Electronics.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

°C	Graus Celsius
µA	Microampere, equivale a 10^{-6} A (Amper)
ACC	*Adaptative Cruise Controle*
ASCII	*American Standard Code for Information Interchange*
ACK	*Acknowledgement*
CAN	*Controller Area Network*
CD	*Compact disc*
CPU	Unidade de Processamento Central
CRC	Cyclic Redundancy Check
CSMA	*Carrier Sense Multiple Access*
EBDCIC	*Extended Binary Coded Decimal Interchange Code*
EOF	*End of Frame*
ESP	*Electronic Stability Program*
DIP	*Dual In-line Package* (DIP *switch*)
DLC	*Data Length Code*
ECU	do inglês Electronic Control Unit, Unidade de Controle Eletrônica
FATEC	Faculdade de Tecnologia
FTP	*File Transfer Protocol*
FW	*Firmware*
GIF	*Graphics Interchange Format*
GND	*Terra*
GUI	*Graphical User Interface* (Interface Gráfica do Usuário)
HD	*Hardware*
IDE	Identificador de Extensão
IEC	Comissão Eletrotécnica Internacional
IP	*Internet Protocol*
ISO	Organização Internacional de Normalização
JPEG	*Joint Photographic Experts Group*

LAN	Local Área *Network*
LLC	*Logical Link Control*
MAC	Controle de Acesso Médio
MPEG	*Motion Picture Experts Group*
MTU	Unidade Máxima de Transmissão
mV	mili Volts
NBT	Tempo Nominal de *Bits*
OSI	*Open Sytem Interconection*
PC	*Personal Computer* (Computador Pessoal)
PCI	Placa de Circuito Impresso
PCS	*Physical Coding Sub-layer*
PMA	*Physical* Média *Attachment*
PMD	*Physical* Média Dependente
RTR	*Remote Transmission Request*
SAE	*Society of Automotive Engineers*
SCP	Protocolo de Controle de Sessão
SMTP	*Simple Mail Transfer Protocol*
SPI	*Serial Peripheral Interface*
SRR	Requisição Remota Substituída
SW	*Software*
TCP	*Transmission Control Protocol*
TIFF	*Tagged Image File Format*
TQ	*Time Quantum*
UDP	*User Data gram Protocol*
USB	*Universal Serial Bus*
USB-B	Conector USB tipo B
USB-IF	*Universal Serial Bus Implementers* Fórum
V	Volts
VAN	*Vehicle Area Network*
Vout	*Tensão de Saída*
VS	Alimentação Positiva

WAN	*Wide Área Network*
ZIP	*Zone Information Protocol*

SUMÁRIO

1 INTRODUÇÃO

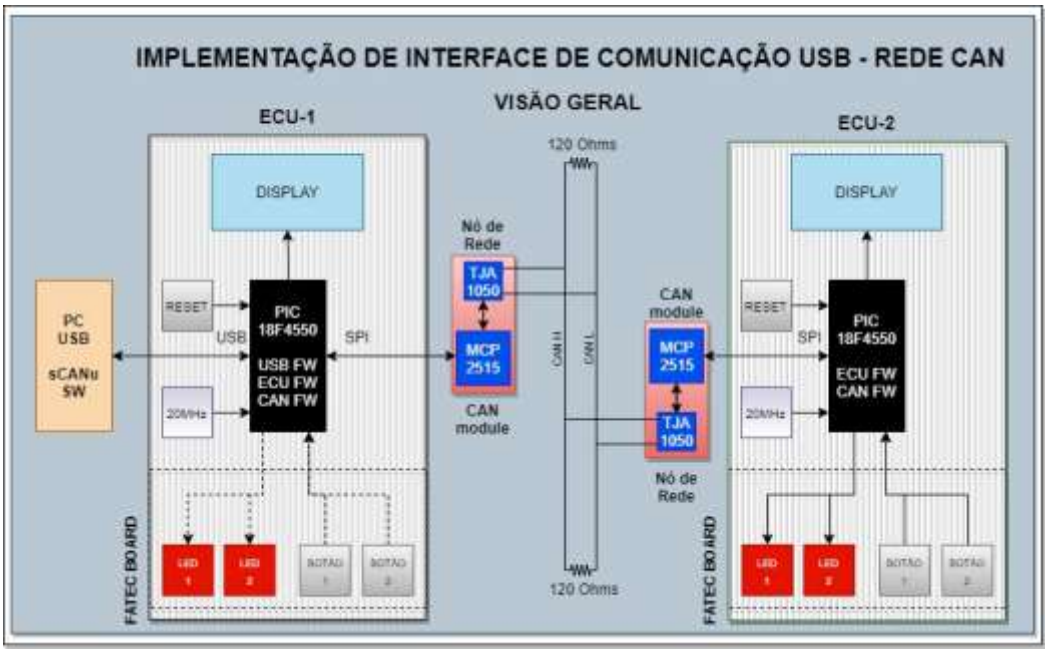
Os veículos atuais possuem muitas unidades de controle eletrônico (ECU, do inglês *Electronic Control Unit*). Essa demanda foi fortemente impulsionada pela necessidade de aumentar a segurança, a comodidade, economia e principalmente pelas leis de controle de emissões e preservação do meio ambiente, cada vez mais rigorosas. De acordo com Robert Bosch Gmbh (2007, pg. 82), atender essa demanda só foi possível com a eletrônica embarcada e a tendência é que a aplicação da eletrônica nos veículos, continue aumentando, com a aplicação de mais ECUs.

A implementação dessas ECUs, segue um padrão *cross-system functions*, onde, conforme Robert Bosch Gmbh (2007), um sinal produzido e processado em uma das ECU, fica disponível, para outras ECUs que precisem dele para executar sua função. Um exemplo, é *o Adaptative Cruise Controle* (ACC), onde as ECUs, do motor, do *Electronic Stability Program* (ESP) e do controle da transmissão, se comunicam e agem, reagindo ao sensor do radar, para manter a distância do veículo a frente, conforme o fluxo de tráfego, fazendo intervenções no freio e transmissão, além de ajustar a carga imposta ao motor. Com esse exemplo, verifica-se, como o método convencional de comunicação, que usa linhas de dados dedicadas, se torna obsoleta rapidamente com o aumento de ECUs implementadas, atingindo limites de sua viabilidade conforme BOSCH (2005), seja no número de pinos nos conectores das ECUs, limitando o seu desenvolvimento, bem como o aumento da complexidade do chicote elétrico.

A solução, segundo BOSCH (2005), foi o emprego de sistemas de barramentos seriais, próprios para aplicações automotivas, entre os quais o *Controller Area Network* (CAN), que se tornou padrão e um componente importante para as novas implementações de sistemas eletrônicos.

Esse trabalho relaciona as informações necessárias para implementação de uma rede CAN genérica, demostrando o *hardware* (HW) e *softwares* necessários, a partir da implementação de um barramento CAN, de arquitetura simples, entre duas ECUs, com interface com um Computador (PC), conforme o diagrama da figura 1.

Figura 1 – Diagrama da arquitetura da rede CAN proposta



Fonte: Autores (2020)

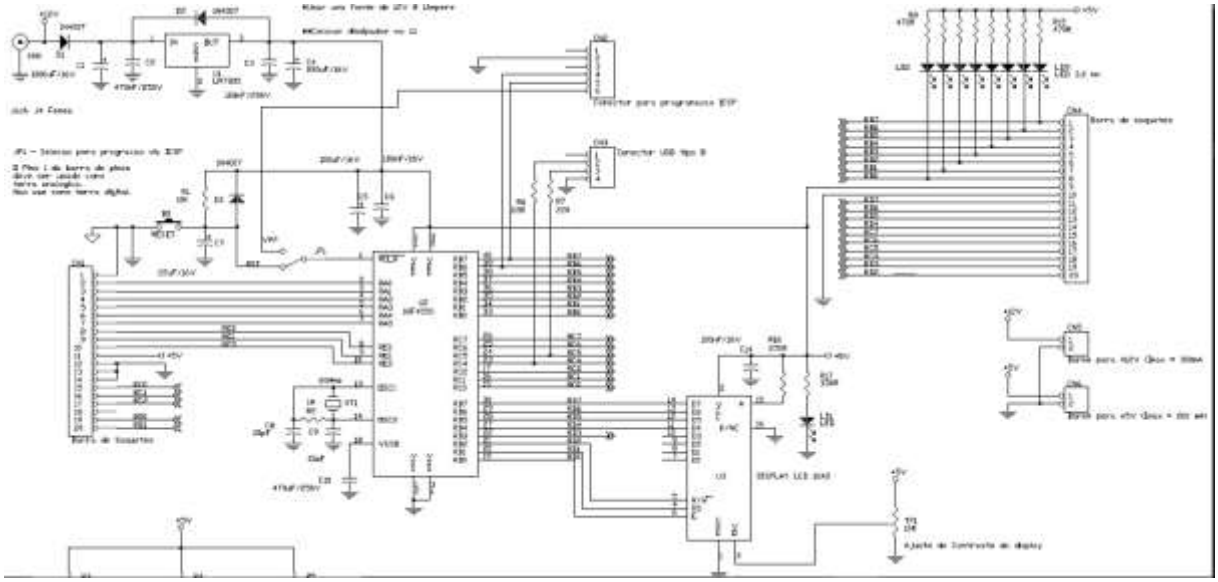
As ECUs serão implementadas a partir da Placa de Circuito Impresso (PCI) existente, projetada e confeccionada para apoiar o curso de Tecnologia em Eletrônica Automotiva, da FATEC Santo André. Essa PCI, chamada de FATEC *Board*, é uma plataforma de desenvolvimento para estudo dos Microcontroladores (MCU, do inglês *Microcontroller Unit*), a partir do Microchip PIC18F4550.

A FATEC *Board* não dispõe de HW para comunicação CAN, sendo necessário a complementação com PCI específico. Nesse trabalho utiliza-se um módulo CAN, que incorpora o controlador MCP2515, da Microchip e o *transceiver* TJA1050 da Philips.

A FATEC *Board* tem como objetivo dar suporte a aprendizagem dos alunos do curso de tecnologia em Eletrônica Automotiva, a placa foi projetada para suportar o PIC 18F4550, nela contém 1 display de segmentos, 8 *leds*, 4 botões, conexão via USB , algumas entradas e saídas digitais e analógicas, alguns componentes eletrônicos como resistores, capacitores, diodos e transistores, esta é a parte de *hardware* da placa de desenvolvimento FATEC, já o *software* do PIC foi desenvolvido através da ide Mplab X, a FATEC *Board* tem uma capacidade de aplicação muito diversa desde um simples acender de led a fazer o gerenciamento de um veículo, o diagrama da placa esta descriminado na figura 2 nele é possível ver todo o esquema elétrico da placa.

Nesse tipo de arquitetura, a ECU fica responsável pelo controle e tratamento das entradas e saída, trocando informações pelas portas de comunicação especificas com um PC e com o controlador CAN, que conforme GUIMARÃES (2003), é o responsável pelo empacotamento dos dados no formato do Protocolo, além da transmissão e recepção de dados da rede CAN.

Figura 2 – Diagrama Elétrico FATEC *BOARD*



Fonte: Santos (2020).

1.1 Motivação

Este trabalho é compatível com o curso de Tecnologia em Eletrônica Automotiva da FATEC Santo André, do Estado de São Paulo, pois abrange diversas áreas estudadas, o que torna possível avançar na pesquisa sobre redes automotivas. Além disso, as redes são e serão cada vez mais utilizadas na implantação da eletrônica embarcada (ROBERT BOSCH GMBH, 2007), seja nos veículos ou em outros produtos e sistemas, o que torna o conhecimento sobre redes CAN e sua implementação importante, para o desenvolvimento de novos sistemas. E ainda autores como GUIMARÃES (2007), reforçam que o CAN Bus é o protocolo que revolucionou a indústria automotiva e tem influenciado fortemente as aplicações na indústria, agricultura, construção naval e aviação, com grande possibilidade de crescimento.

1.2 Objetivo

Criar uma experiência objetiva sobre rede CAN e interface com PC, implementando um barramento de comunicação, a partir de um setup, que permita a efetiva aplicação do conceito e sua verificação.

1.3 Metodologia e estrutura do trabalho

O trabalho está dividido em duas partes, a primeira, trata-se de uma pesquisa sobre o tema, reunindo informações sobre os conceitos e componentes do projeto, e a segunda, descreve a experiência da implementação da rede, a confecção dos SW e testes executados.

2 CONCEITOS

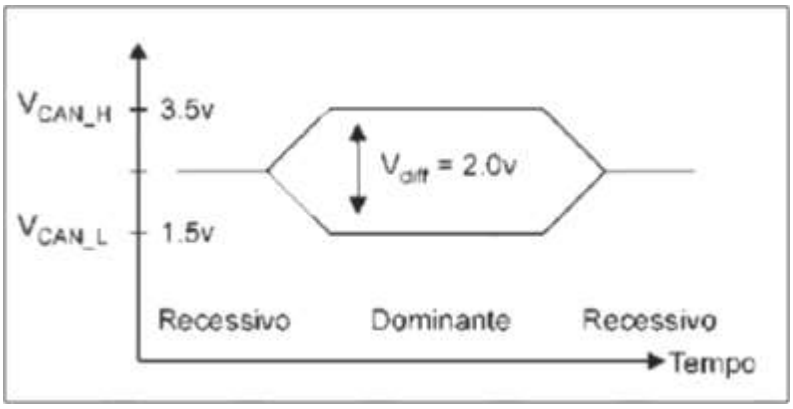
Nesse capítulo são abordados os principais temas relativos ao objetivo desse trabalho, para a compreensão dos conceitos, sistemas e componentes.

Segundo CIA (2021), por volta de fevereiro de 1986, a Bosch apresentou um sistema de barramento serial, denominado *Controller Area Network*, ou simplesmente Rede CAN, no congresso da *Society of Automotive Engineers* (SAE), que revolucionária a comunicação para sistemas embarcados. Foi tão bem-sucedido que atualmente, pelo menos m barramento CAN está implementado nos veículos e sua utilização se espalhou para muitas outras aplicações.

Segundo Bosch (2005) a rede CAN é um protocolo de comunicação serial síncrono entre módulos (ECU). O sincronismo é feito no início de cada mensagem enviada a rede, através do estabelecimento de um *clock*, em tempos conhecidos e periódicos. É baseada no conceito de multimestre, em que todos os módulos ora enviam dados (Mestre), ora recebem (Escravos). As mensagens são enviadas para o barramento da rede CAN e ficam disponíveis a todos os módulos, que recebem conforme suas configurações definidas nos *software* de controle.

Os envios são escalonados através de um processo denominado de arbitragem, onde se estabelece prioridades de acordo com as ID dos dados, quem tiver a maior prioridade continua enviando, quem não tiver cessa sua transmissão. No CAN os dados são representados por *bits* recessivos e dominantes representados pela diferença de tensão que os formam, como é mostrado na figura 3.

Figura 3 – Bits dominantes e recessivos no barramento CAN



Fonte: GUIMARÃES (2007)

2.1 Modelo *Open Sytem Interconection* (OSI)

De acordo com Bosch (2005) o modelo OSI define em termos de uma pilha vertical de sete camadas. As camadas superiores do modelo OSI representam *software* que implementa serviços de rede como criptografia e gerenciamento de conexão. As camadas inferiores do modelo OSI implementam funções orientadas ao *hardware* mais primitivas, como roteamento, endereçamento e controle de fluxo.

O modelo OSI foi introduzido em 1984. Embora tenha sido projetado para ser um modelo abstrato, o modelo OSI continua sendo uma estrutura prática para as principais tecnologias de rede de hoje, como *Ethernet* e protocolos como *Internet Protocol* (IP).

O modelo OSI deve ser usado como um guia de como os dados são transmitidos pela rede. É uma representação abstrata do caminho dos dados e deve ser tratada como tal.

O modelo OSI foi feito especificamente para conectar sistemas abertos. Esses sistemas são projetados para serem abertos para comunicação com quase qualquer outro sistema. O modelo foi feito para quebrar cada camada funcional para que a complexidade geral do design pudesse ser reduzida. Como mostrado na figura 4, o modelo foi construído com sete camadas para o fluxo de informações.

VII Camada de aplicação

VI Camada de apresentação

V Camada de sessão

IV Camada de transporte

III Camada de rede

II Camada de enlace de dados

I Camada física

Figura 4 -- As 7 Camadas do modelo OSI

Fonte: https://sites.google.com/site/wagnerbertoldi/redes-e-sistemas-distribuidos/aula-4---modelo-osi

2.1.1 Camada de Aplicação (7)

Segundo Bosch (2005) a camada de aplicação fornece um meio para o usuário acessar informações na rede por meio de um aplicativo. Essa camada é a principal interface para o usuário interagir com o aplicativo e, portanto, com a rede.

A camada de aplicativo é a camada OSI mais próxima do usuário final, o que significa que tanto a camada de aplicativo OSI quanto o usuário interagem diretamente com o aplicativo de *software*. Essa camada interage com aplicativos de *software* que implementam um componente de comunicação. Esses programas de aplicativos estão fora do escopo do modelo OSI. As funções da camada de aplicativo geralmente incluem a identificação de parceiros de comunicação, determinação da disponibilidade de recursos e sincronização da comunicação. Ao identificar os parceiros de comunicação, a camada de aplicativo determina a identidade e a disponibilidade dos parceiros de comunicação para um aplicativo com dados a serem transmitidos. Ao determinar a disponibilidade de recursos, a camada de aplicação deve decidir se existem recursos de rede suficientes para a comunicação solicitada. Na sincronização da comunicação, toda comunicação entre os aplicativos requer cooperação, que é gerenciada pela camada do aplicativo.

Alguns exemplos de implementações da camada de aplicativo incluem *Telnet, File Transfer Protocol* (FTP) e *Simple Mail Transfer Protocol* (SMTP).

2.1.2 Camada de Apresentação (6)

Segundo Guimarães (2007) a camada de apresentação gerencia a apresentação das informações de maneira ordenada e significativa. A função primária desta camada é a sintaxe e a semântica da transmissão de dados. Ele converte as representações de dados do computador *host* local em um formato de rede padrão para transmissão na rede. No lado receptor, ele muda o formato da rede para o formato do computador *host* apropriado, de forma que os dados possam ser utilizados independentemente do computador *host*. As conversões *American Standard Code for Information Interchange* (ASCII) e *Extended Binary Coded Decimal Interchange Code* (EBCDIC), criptografia e semelhantes são tratados aqui.

Segundo Guimarães (2007) a camada de apresentação fornece uma variedade de funções de codificação e conversão que são aplicadas aos dados da camada de aplicativo. Essas funções garantem que as informações enviadas da camada de aplicativo de um sistema possam ser lidas pela camada de aplicativo de outro sistema. Alguns exemplos de esquemas de codificação e conversão de camada de apresentação incluem formatos de representação de dados comuns, conversão de formatos de representação de caracteres, esquemas de compressão de dados comuns e esquemas de criptografia de dados comuns.

Formatos de representação de dados comuns, ou o uso de formatos de imagem, som e vídeo padrão, permitem o intercâmbio de dados de aplicativos entre diferentes tipos de sistemas de computador. Usando diferentes representações de texto e dados, como EBCDIC e ASCII, usa esquemas de conversão para trocar informações com sistemas. Os esquemas de compactação de dados padrão permitem que os dados sejam compactados. ou criptografado no dispositivo de origem para ser descompactado adequadamente ou decifrado no destino.

As implementações da camada de apresentação geralmente não estão associadas a uma pilha de protocolo específica. Alguns padrões conhecidos para vídeo incluem *QuickTime* e *Motion Picture Experts Group* (MPEG). *QuickTime* é uma especificação da *Apple Computer* para vídeo e áudio, e MPEG é um padrão para compressão e codificação de vídeo.

Entre os formatos de imagem gráfica mais conhecidos estão *Graphics Interchange Format* (GIF), *Joint Photographic Experts Group* (JPEG) e *Tagged Image File Format* (TIFF). GIF é um padrão para compactar e codificar imagens

gráficas. JPEG é outro padrão de compactação e codificação para imagens gráficas e TIFF é um formato de codificação padrão para imagens gráficas.

2.1.3 Camada de Sessão (5)

Segundo Guimarães (2007) a camada de sessão coordena o diálogo, sessão, conexão, entre dispositivos na rede. Esta camada gerencia as comunicações entre as sessões conectadas. Exemplos dessa camada são o gerenciamento de *tokens* (a camada de sessão gerencia quem tem o *token*) e a sincronização de horário da rede.

A camada de sessão estabelece, gerencia e encerra sessões de comunicação. As sessões de comunicação consistem em solicitações de serviço e respostas de serviço que ocorrem entre aplicativos localizados em diferentes dispositivos de rede. Essas solicitações e respostas são coordenadas por protocolos implementados na camada de sessão. Alguns exemplos de implementações de camada de sessão incluem *Zone Information Protocol* (ZIP), o protocolo *AppleTalk* que coordena o processo de vinculação de nomes; e o Protocolo de Controle de Sessão (SCP), o protocolo de camada de sessão *Decent Phase* IV.

2.1.4 Camada de Transporte (4)

Segundo Bosch (2005) a camada de transporte é responsável pela transmissão confiável de dados e especificações de serviço entre *hosts*. A principal responsabilidade dessa camada é a integridade dos dados - os dados transmitidos entre os *hosts* são confiáveis e oportunos. Os gramas de dados da camada superior são divididos em gramas de dados do tamanho da rede, se necessário, e então implementados usando o controle de transmissão apropriado. A camada de transporte cria uma ou mais conexões de rede, dependendo das condições. Essa camada também controla o tipo de conexão que será criada. Dois principais protocolos de transporte são o *Transmission Control Protocol* (TCP) e o *User Data gram Protocol* (UDP).

Características importantes da camada de transporte:

1 A camada de transporte garante um serviço confiável.

2 Divide a mensagem (da camada de sessões) em pacotes menores, atribui um número de sequência e os envia.

3 Conexões de transporte confiáveis são construídas sobre IP.

4 No caso de IP, os pacotes perdidos que chegam fora de serviço devem ser reordenados.

Recursos importantes do TCP / UDP:

1 TCP / IP Amplamente usado para camada de rede / transporte no sistema operacional UNIX.

2 TCP :Este é um protocolo orientado a conexão.

3 UDP: Este é um protocolo de camada de transporte sem conexão.

4 Os programas aplicativos que não precisam de protocolo orientado a conexão geralmente usam UDP.

2.1.5 Camada de Rede (3)

Segundo Bosch (2005) a camada de rede é responsável pelo roteamento de dados (pacotes) pela rede; lida com o endereçamento e entrega de dados. Essa camada fornece controle de congestionamento, informações de contabilidade para a rede, roteamento, endereçamento e várias outras funções. *Internet Protocol* (IP) é um bom exemplo de protocolo de camada de rede. A camada de rede não lida com mensagens perdidas.

Recursos importantes dos protocolos da camada de rede:

Preocupado com a transmissão de pacotes.

Escolha o melhor caminho para enviar um pacote (roteamento).

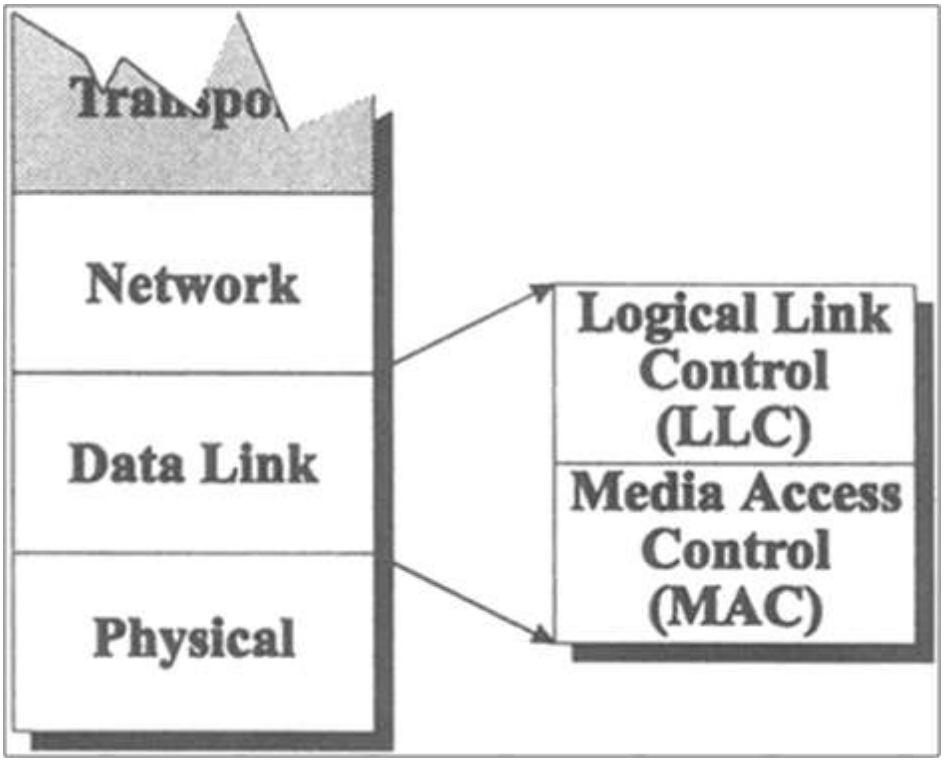
O roteamento pode ser complexo em uma grande rede (por exemplo, *Internet*).

O roteamento de pacotes através de uma rede pode ser realizado usando rotas estáticas simples ou algoritmos de roteamento dinâmico complexos.

2.1.6 Camada de Enlace de Dados (2)

Segundo Bosch (2005) a camada de enlace de dados fornece a entrega confiável de dados em uma rede física. Esta camada lida com questões como regulação de fluxo, detecção e controle de erros e *frames*. Essa camada tem a importante tarefa de criar e gerenciar quais quadros são enviados na rede exemplificado na figura 5. O quadro de dados da rede, ou pacote, é feito de *checksum*, endereço de origem, endereço de destino e os próprios dados. O maior tamanho de pacote que pode ser enviado define a Unidade Máxima de Transmissão (MTU).

Figura 5 -- Exemplo da camada de enlace



Fonte: https://networkencyclopedia.com/logical-link-control-llc-layer/

Recursos importantes da camada de enlace de dados:

1 Lida com erros na camada física.

2 Agrupa *bits* em quadros e garante sua entrega correta.

3 Adiciona alguns *bits* no início e no final de cada quadro mais a soma de verificação.

3 O receptor verifica a soma de verificação.

4 Se a soma de verificação não estiver correta, ele solicitará a retransmissão. (Envie uma mensagem de controle).

5 Consiste em duas subcamadas:

6 *Logical Link Control* (LLC) define como os dados são transferidos pelo cabo e fornece serviço de *link* de dados para as camadas superiores.

O Controle de Acesso Médio (MAC) define quem pode usar a rede quando vários computadores estão tentando acessá-la simultaneamente (ou seja, passagem de *token*, *Ethernet* [*Carrier Sense Multiple Access* (CSMA) / *Compact disc* (CD)]).

A camada de enlace de dados fornece trânsito confiável de dados em um *link* de rede física. Diferentes especificações da camada de enlace de dados definem diferentes características de rede e protocolo, incluindo endereçamento físico,

topologia de rede, notificação de erro, sequenciamento de quadros e controle de fluxo. O endereçamento físico (em oposição ao endereçamento de rede) define como os dispositivos são endereçados na camada de enlace de dados. A topologia de rede consiste nas especificações da camada de enlace que geralmente definem como os dispositivos devem ser conectados fisicamente, como em um barramento ou em uma topologia em anel. A notificação de erro alerta os protocolos da camada superior de que ocorreu um erro de transmissão e o sequenciamento dos quadros de dados reordena os quadros que são transmitidos fora da sequência. Finalmente,
os protocolos usados na camada de enlace são SLIP, PPP, MTU e CSLP.

2.1.7 Camada Física (1)

Segundo Marques (2004) Lida com a comunicação elétrica de nível de *bit* através do canal da rede. A camada física define as especificações elétricas, mecânicas, procedimentais e funcionais para ativar, manter e desativar o *link* físico entre os sistemas de rede em comunicação. As especificações da camada física definem características como mídia, níveis de tensão, temporização das mudanças de tensão, taxas de dados físicos, distâncias máximas de transmissão e conectores físicos.

Basicamente, essa camada garante que um *bit* enviado de um lado da rede seja recebido corretamente do outro lado.

Os dados viajam da camada de aplicação do emissor, descendo pelos níveis, pelos nós do serviço de rede e subindo pelos níveis do receptor.

Para controlar a transmissão, cada camada "envolve" os dados e o cabeçalho da camada anterior com seu próprio cabeçalho. Um pequeno pedaço de dados será transmitido com vários cabeçalhos de camada anexados a ele. Na extremidade receptora, cada camada retira o cabeçalho que corresponde ao seu respectivo nível.

A camada física se preocupa com o seguinte:

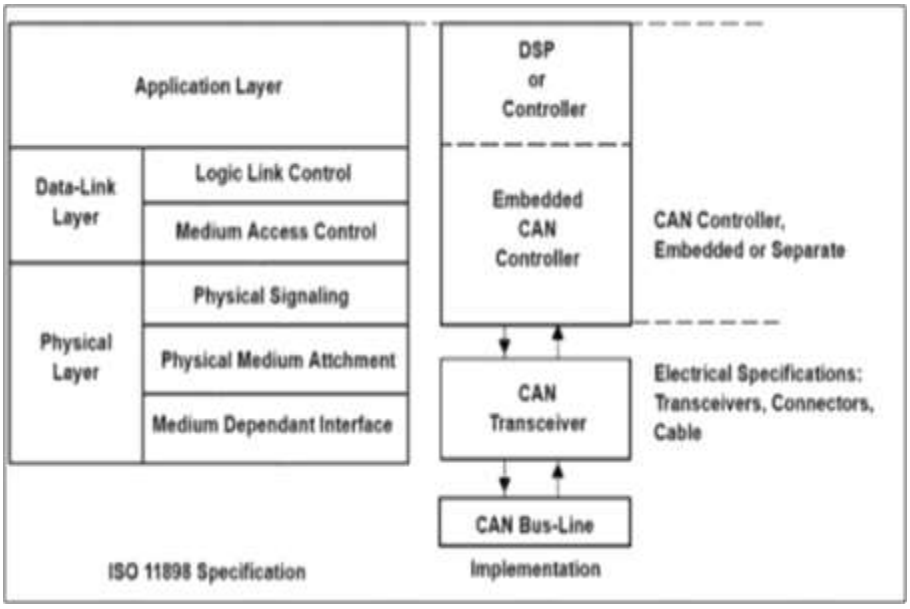
1 Características da interface física, como especificações elétricas e mecânicas,

2 Número de *bits* de segundo a serem transmitidos,

3 Tipo de transmissão como *duplex* ou *half-duplex* etc.

Protocolos da camada física usados com frequência na aplicação automotiva estão demonstrado na figura 6.

Figura 6 -- Protocolos da Camada Física



Fonte: Santos (2020).

Alguns dos padrões importantes que lidam com as especificações da camada física são:

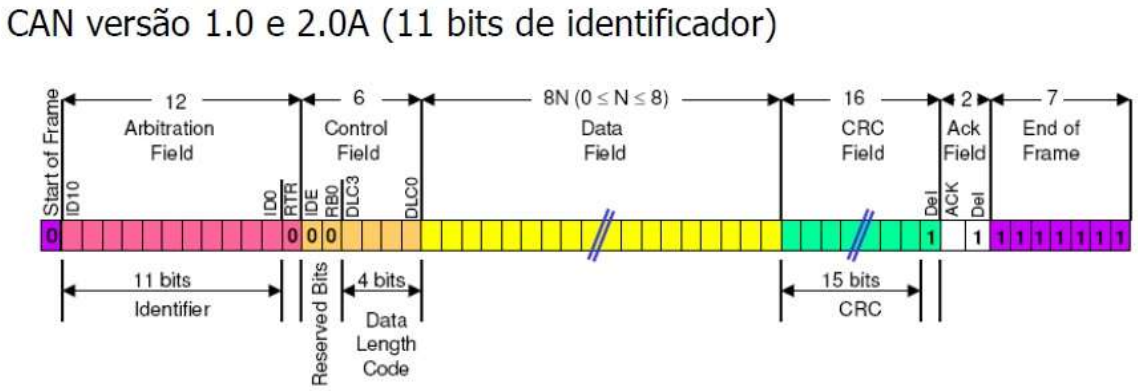
RS-232 (para linhas de comunicação serial), X.21, EIA 232 e G730.

As implementações da camada física e da camada de enlace de dados podem ser categorizadas como especificações de Local Área *Network* (LAN) ou *Wide Área Network* (WAN).

2.2 Estrutura de um frame CAN

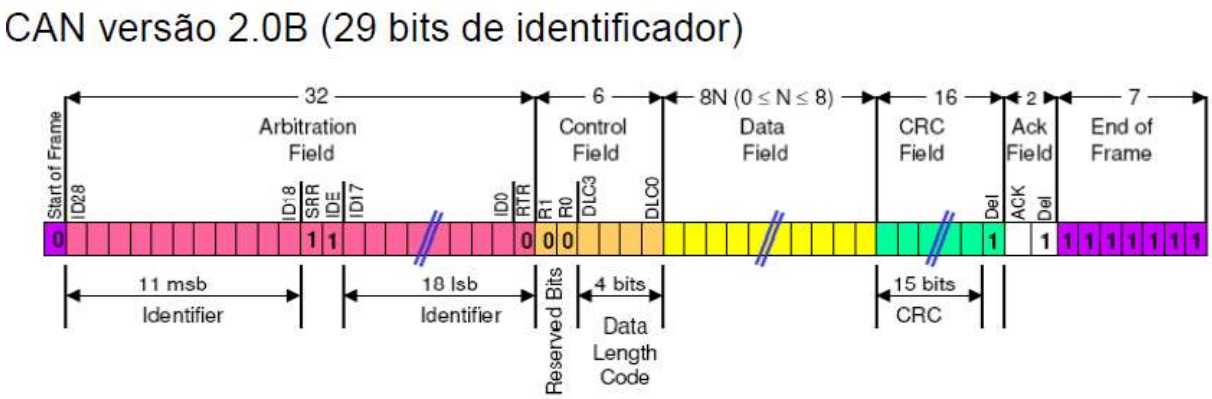
Segundo Marques (2004) o protocolo CAN tem duas versões mais utilizadas o CAN versão 1.0 e o 2.0, sendo que o 2.0 tem duas versões o CAN 2.0A (Padrão) e o 2.0B (Estendido), a grande diferença entre as versões basicamente é o número de *bits* do identificador, versão 1.0 e 2.0A tem 11 *bits* como demonstrado na figura 7 e o 2.0B pode ter de 11 *bits* e 29 *bits* como demonstrado na figura 8. Na versão padrão a quantidade de mensagens transmitidas pode chegar a 2048 identificadores mandando mensagens distintas, já a versão estendida esses identificadores podem chegar a 537 milhões distintos.

Figura 7 -- Estrutura do CAN 1.0 e 2.0A



Fonte: Santos (2020).

Figura 8 -- Estrutura do CAN 2.0B

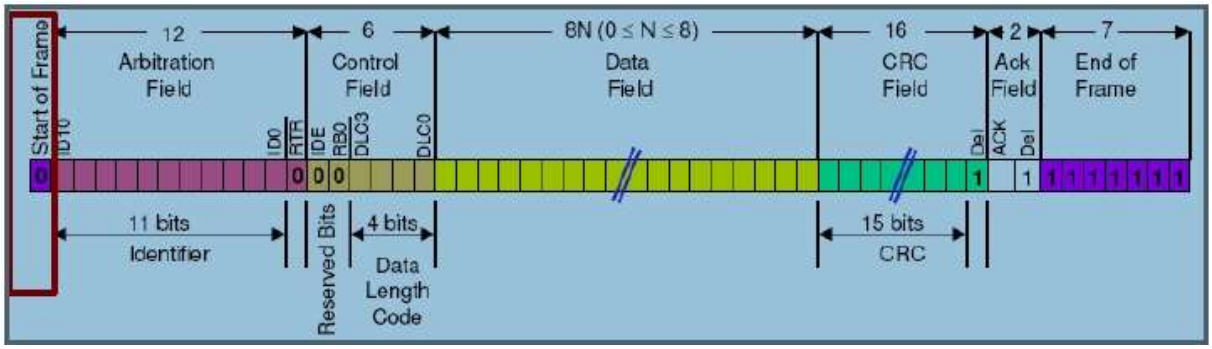


Fonte: Santos (2020).

2.2.1 Campo SOF (*Start of Frame*)

Segundo Santos (2020), é o espaço de um *bit* (dominante “0”) conforme evidenciado na figura 9. Este *bit* indica o início de uma mensagem, em contrapartida o barramento permanece inativo sob o estado lógico representado pelo *bit* (recessivo “1”).

Figura 9 -- *Start of Frame* em Evidência



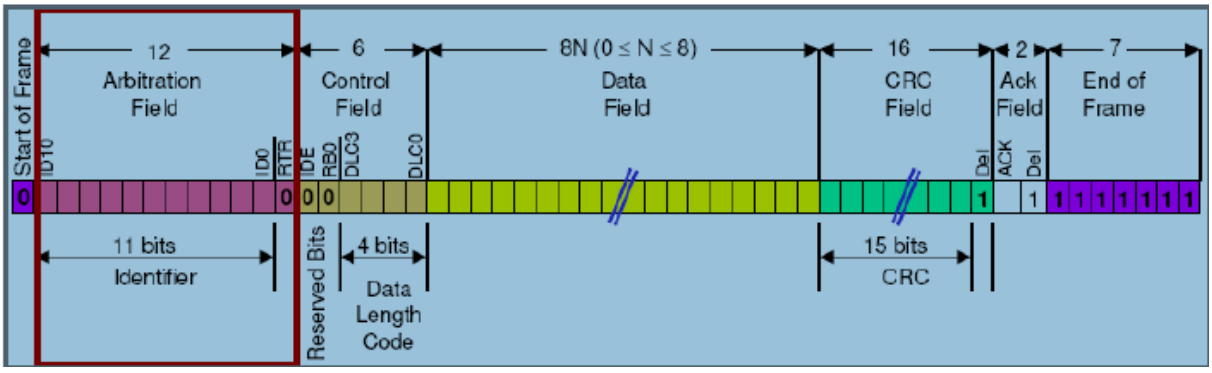
Fonte: Santos (2020).

2.2.2 Campo de Arbitragem e do Identificador e RTR

Segundo Santos (2020), este campo pode ter um identificador de 11 bits, evidenciado na figura 10, ou 29 *bits* evidenciado na figura 11, e um *bit* de *Remote Transmission Request* (RTR). O processo de arbitrariedade ocorre sempre que dois ou mais ECU's no caso dos veículos, por exemplo, tentam acessar o barramento ao mesmo tempo. Esta disputa ocorre *bit* a *bit* e ganha quem tem a maior prioridade, determinada a partir da ECU que enviou a maior sequência de bits dominantes. As ECU's que perderam mudam seu estado de transmissoras para receptoras.

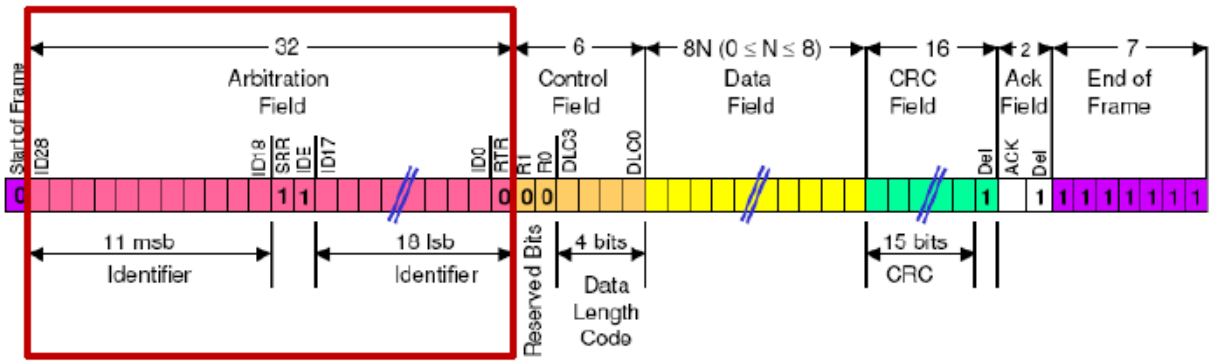
Deste modo, o Identificador (ID) que possuir o menor valor binário tem prioridade sobre o que possuir maior valor binário. O campo *Remote Transmission Request* (RTR) indica que se a mensagem é um *frame* de requisição ou um *frame* de dados, se dominante e recessivo respectivamente, isto também é utilizado na Requisição Remota Substituída (SRR), com *bit* dominante pois não envia *frame* remoto na versão *extended*.

Figura 10 -- Campos de Arbitragem, Identificador 11 *bits* e RTR em Evidência



Fonte: Santos (2020).

Figura 11 -- Campo de Arbitragem, Identificador 29 *bits* e RTR em Evidência

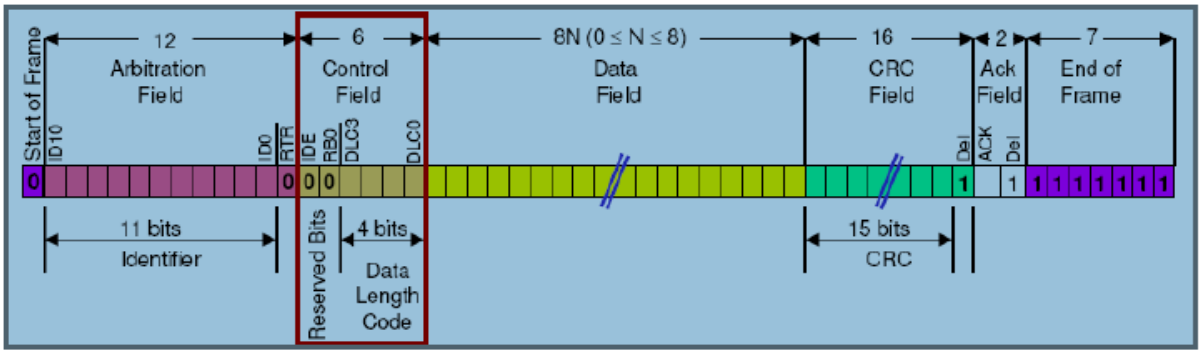


Fonte: Santos (2020).

2.2.3 Campo de Controle

Segundo Santos (2020), este campo mostrado na figura 12, composto pelo Identificador de Extensão (IDE), que indica se a mensagem é padrão, se for dominante, ou *extended* se for recessivo. Também é composto por *bits* reservados no caso da versão padrão é apenas um o r0, já no caso da versão *extended* são dois o r0 e r1, esses *bits* são reservados para possíveis mudanças futuras e estes *bits* são enviados como dominante, este campo contém também o *Data Length Code* (DLC) com 4 *bits* e mostra quantos *bits* terá mensagem.

Figura 12 -- Campo de Controle em Evidência



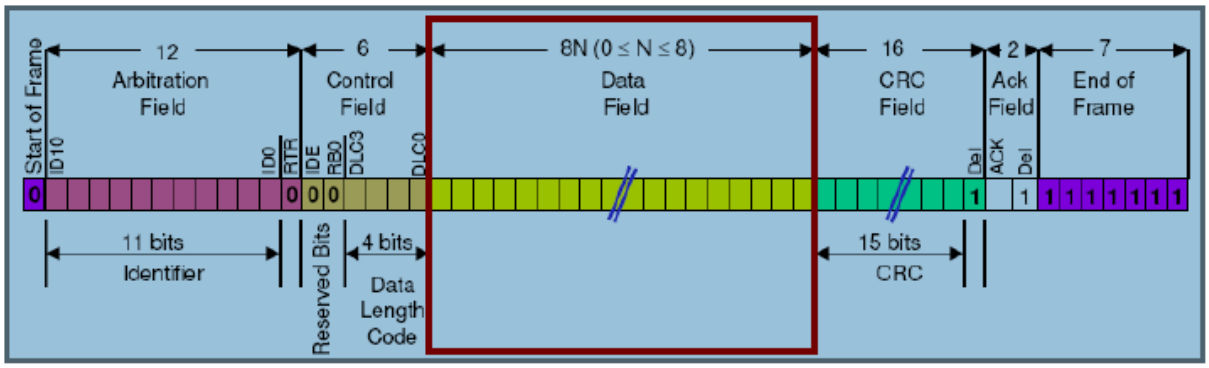
Fonte: Santos (2020).

2.2.4 Campo de Dados

Segundo Santos (2020) este campo contém os dados da mensagem enviados podendo conter de 0 a 64 *bits*, evidenciado na figura 13, ela contém todas as

informações que serão transmitidas no barramento, informações tais como exemplo leitura de um sensor, acionamento de um atuador dentre outras informações que necessitem ser transmitidas.

Figura 13 -- Campo de Dados em Evidência

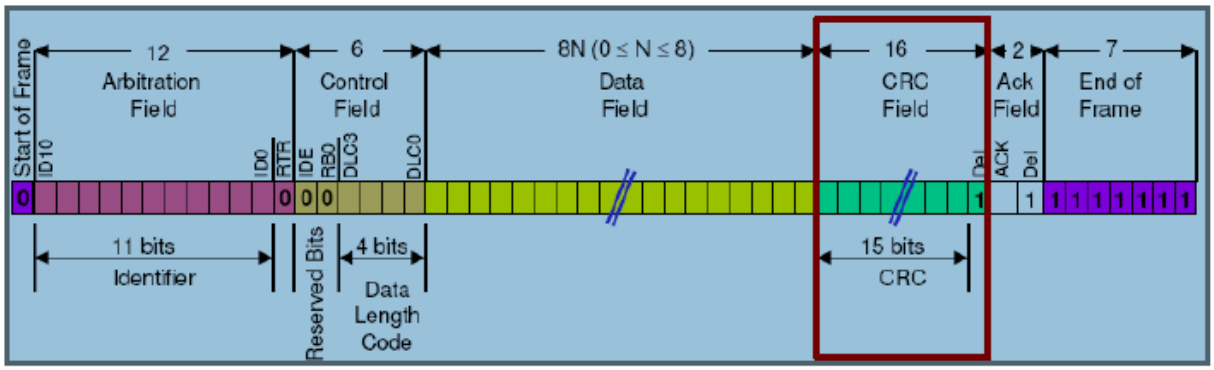


Fonte: Santos (2020).

2.2.5 *Cyclic Redundancy Check* (CRC)

Segundo Santos (2020) este campo está evidenciado na figura 14, faz o controle de erros das mensagens, fazendo um teste de redundância cíclica de 15 *bits* através de um polinômio identificador seguido de mais um *bit* recessivo limitador no final da sequência, se o cálculo do receptor não for o mesmo que o emissor é gerado um erro de CRC e a mensagem é descartada e é enviado no barramento um *frame* de erro.

Figura 14 -- Campo de CRC em Evidência

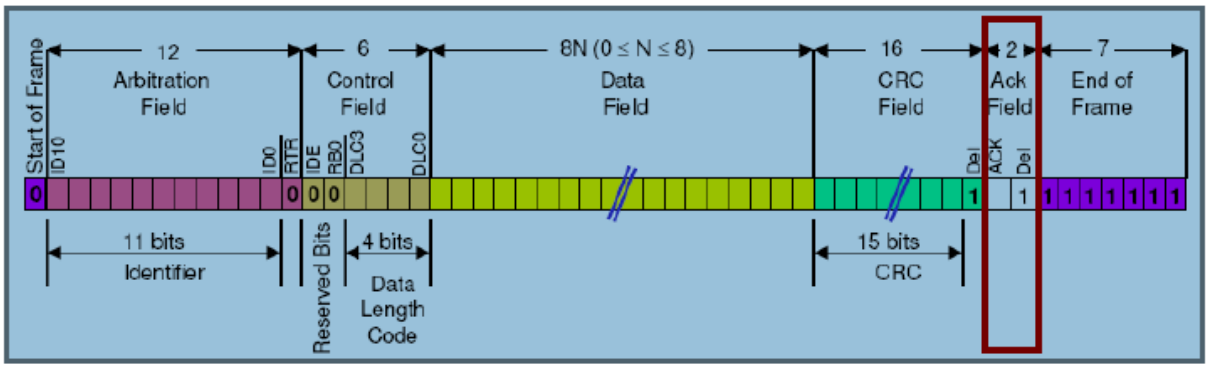


Fonte: Santos (2020).

2.2.6 Campo *Acknowledgement* (ACK)

Segundo Santos (2020) este campo contém dois *bits* um é do ACK *slot e o outro é o ACK delimiter,* evidenciado na figura 15, estes *bits* ambos são enviados com *bits* recessivos, porém se ocorrer algum erro durante a mensagem transmitida, a ECU que identificar primeiro este erro notifica ao barramento enviando um estado dominante no *bit*.

Figura 15 -- Campo *Acknowledgement* (ACK) em Evidência

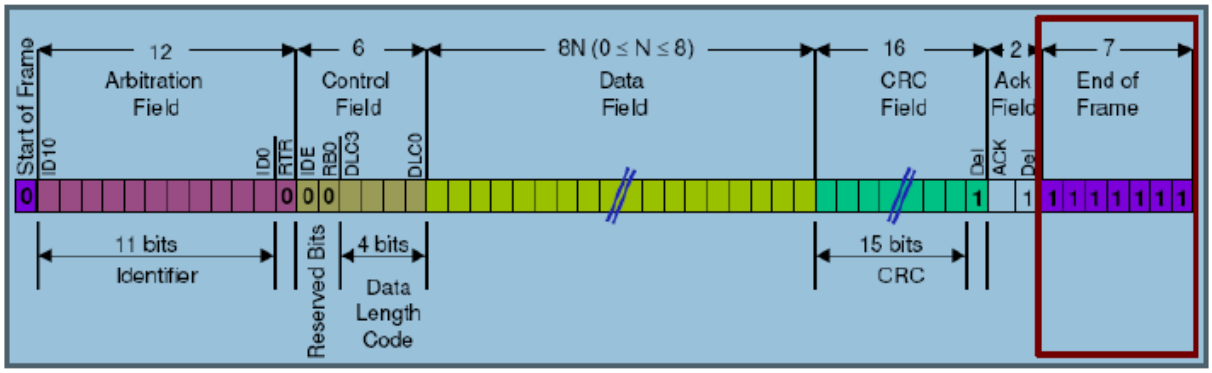


Fonte: Santos (2020).

2.2.7 Campo *End of Frame* (EOF)

Neste campo compõem 7 *bits* enviados ao barramento recessivos que sinalizam o fim da mensagem, que estará evidenciado na figura 16.

Figura 16 -- Campo *End of Frame* (EOF) em Evidência



Fonte: Santos (2020).

2.3 *Frame* de Erro Remoto e *Bit Stuff*

Segundo Santos (2020) o *frame* de erro remoto é semelhante ao *frame* de dados porem não possui campo específico como o *frame* de dados sua função é

solicitar mensagens enviadas de outras ECUs, podendo assim as ECUs transmissoras trabalhar sob demanda, não necessitando enviar os seus dados, somente quando o receptor mandar um identificador igual ao da mensagem solicitada, o *frame* de erro é utilizado quando há um erro na mensagem ou para identificar uma central com defeito, abortando a transmissão sinalizando com 6 *bits* dominantes e 8 *bits* recessivos.

Bit stuff consiste em inserir 1 *bit* com valor inverso a cada 5 *bits* iguais e consecutivos, tem a finalidade de limitar a distância máxima entre pontos de sincronização, quando o receptor recebe uma mensagem ela terá que retirar o *bit stuff* para ter os dados corretos, este processo está demonstrado na figura 17 onde a primeira linha mostra um exemplo de um campo mensagem normal, e na segunda linha está composta como ficaria com a inserção do *bit stuff* nas duas condições dominante e recessivo.

Figura 17 -- Exemplo de formação de *Bit Stuff*

Destuffed bit stream	01011111010	10100000101	01011111000010	10100000111101
Stuffed bit stream	01011111o010	10100000i101	01011111o0000i10	10100000i1111o01
"0", "o" = dominant (stuff) bit; "1", "i" = recessive (stuff) bit				

Fonte: Santos (2020).

2.4 Modelo Funcional da Camada Física

A implementação da camada física liga um node ao barramento CAN, essa quantidade e limitação de node depende da quantidade de cargas elétricas ligadas no barramento e do protocolo da camada de enlace, a camada física é modelada pela ISO e pela Comissão Eletrotécnica Internacional (IEC) 8802-3 e possui três subcamadas, *Physical Coding Sub-layer* (PCS), *Physical* Média *Attachment* (PMA), *Physical* Média Dependente (PMD).

O PMS abrange funções referente a codificação e decodificação de *bits*, sincronização e temporização assim como detecção de falhas de barramento, e fornece o serviço de PCS *data request* que é da subcamada MAC para a camada física solicitar transmissão de um *bit* recessivo ou dominante, e o serviço PCS *data indicate* que é da camada física para a subcamada MAC para indicar a recepção de um *bit* recessivo ou dominante. A subcamada PMA abrange os circuitos funcionais do

barramento de transmissão e recepção. E a subcamada PMD abrange as interfaces elétricas e mecânicas entre o meio físico e a subcamada PMA.

Há seguir será abordado a especificação PCS, estrutura de do *bit* na CAN.

O tempo referente a um *bit* é uniforme ao longo do barramento CAN e é dado pela fórmula descrita abaixo:

fNBT = 1/tNBT

fNBT = Taxa de Transmissão do Barramento

tNBT = Tempo nominal de um *Bit* no Barramento

O tempo utilizado por um *bit* é dividido em 4 segmentos e não se sobrepõe, como demonstrado na figura 16, o ponto de amostragem descrito na figura 16 é a posição do ponto de amostragem real se tiver uma única amostra por *bit* e esta for selecionada, já se três amostras por *bit* forem selecionadas, o ponto de amostra indicado na figura 18 indicara a posição final da amostra. O período do Tempo Nominal de *Bits* (NBT), é a soma das durações dos 4 segmentos como demonstrado na fórmula.

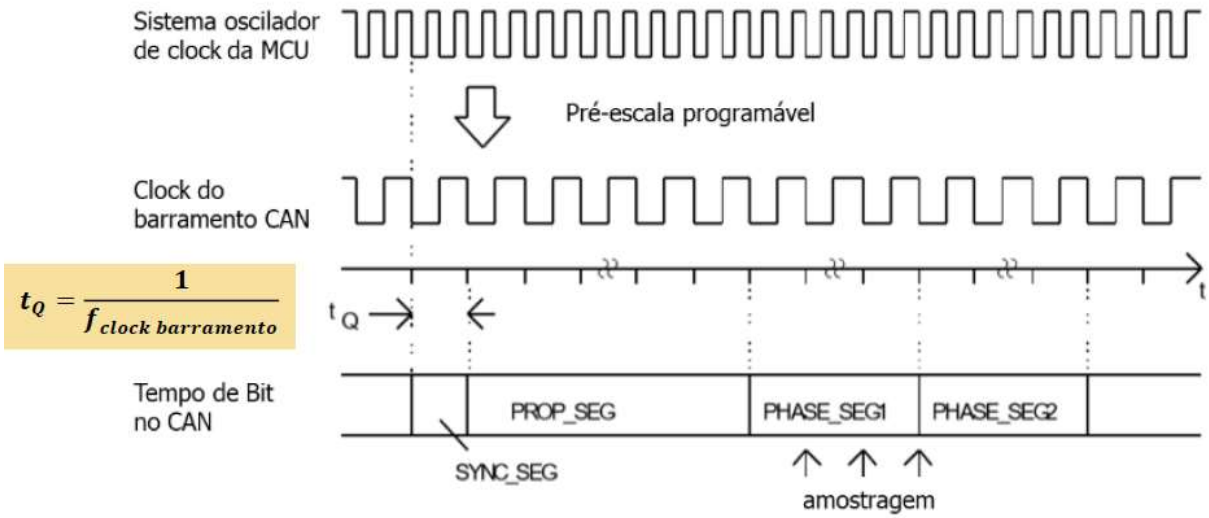
tNBT = *tSYNC_SEG*+ *tPROP_SEG*+*tPHASE_SEG*1+*tPHASE_SEG*2

Figura 18 – NBT

Fonte: Santos (2020).

Cada um destes segmentos é múltiplo inteiro de uma unidade de tempo *Time Quantum* (TQ), o quanto tem de duração um TQ é igual ao período do *clock* do sistema CAN, que deriva do *clock* do sistema do microcontrolador ou do oscilador por meio da pré-escala programável e suas subdivisões, chamada de pré-escala de taxa de transmissão como demonstrado na figura 19.

Figura 19 – Pré-escala programável



Fonte: Santos (2020).

2.5 *Universal Serial Bus* (USB)

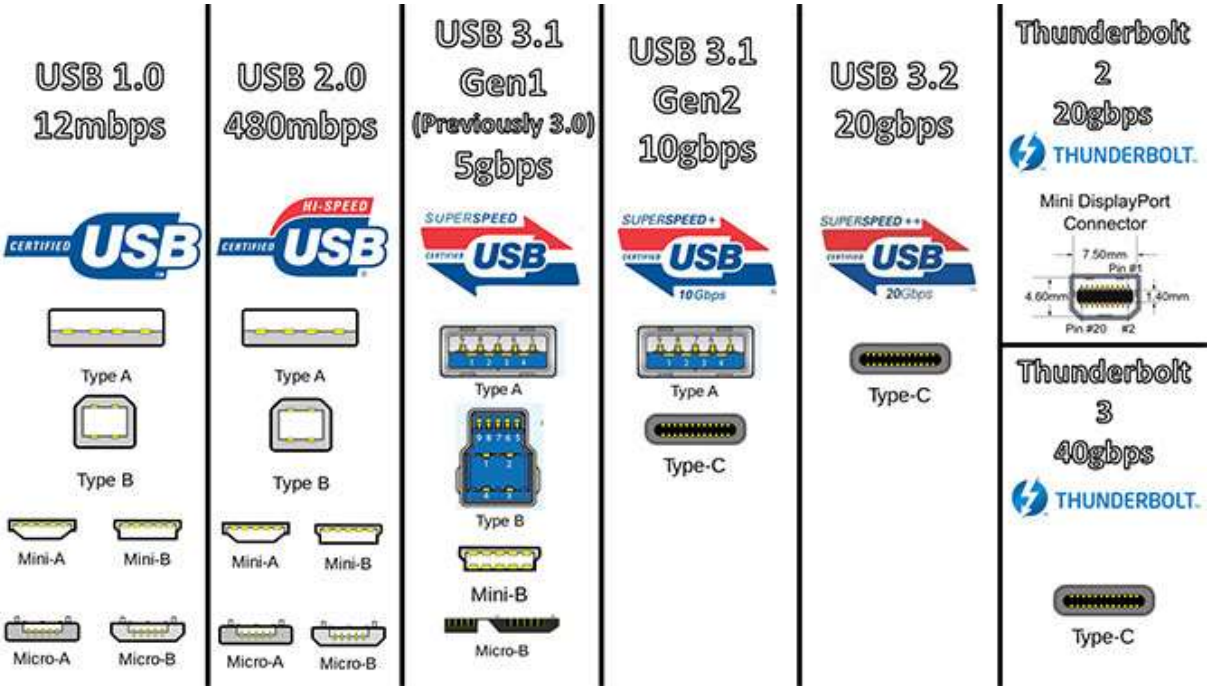
Segundo Anderson (1997) *Universal Serial Bus* (USB) é uma interface padronizado para conectar dispositivos periféricos a um computador. O sistema USB foi originalmente desenvolvido por um grupo de empresas que são: Compaq, Digital Equipment, IBM, Intel, Microsoft e Northern Telecom por volta de 1995, para substituir o sistema de conector misto existente por uma arquitetura mais simples.

O USB foi projetado para substituir a grande quantidade de cabos e conectores necessários para conectar dispositivos periféricos diferentes a um computador. O principal objetivo do USB era tornar a conexão de dispositivos periféricos mais rápida e fácil. Todos os dispositivos USB compartilham algumas características importantes para tornar isso possível, todos os dispositivos USB são auto identificados no barramento, todos os dispositivos são *hot-pluggable* para permitir a verdadeira capacidade *Plug'n'Play*. Além disso, alguns dispositivos podem obter energia do USB, o que elimina a necessidade de adaptadores de energia extras.

Para garantir a máxima interoperabilidade, o padrão USB define todos os aspectos do sistema USB, desde a camada física (mecânica e elétrica) até a camada de *software*. O padrão USB é mantido e aplicado pelo USB *Implementers* Fórum

(USB-IF). Os dispositivos USB devem passar por um teste de conformidade USB-IF para serem considerados em conformidade e poderem usar o logotipo USB demonstrado na figura 20.

Figura 20 -- Logotipos Padrão USB *Implementers* Fórum (USB-IF)



Fonte:https://geektrooper.wordpress.com/2019/02/27/usb-if-troca-nome-de-todos-os-conectores-usb-tipo-c-do-2-0-ate-o-3-2-gen-2/

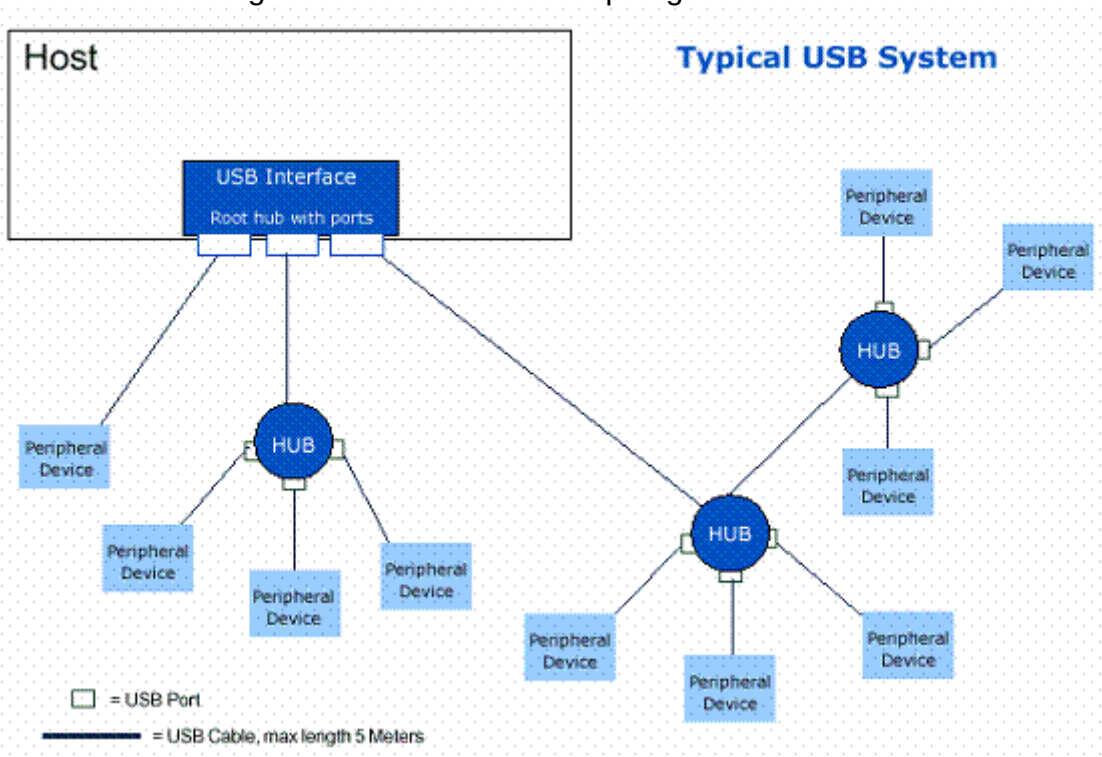
Segundo Anderson (1997) o USB 1.0 foi introduzido pela primeira vez em 1996, mas não foi amplamente adotado até 1998 com o USB 1.1. Já em 2000, o USB 2.0 foi lançado e desde então se tornou o padrão de fato para conectar dispositivos a computadores. Em 2008, a especificação USB foi expandida com USB 3.0, também conhecido como Super*Speed* USB. O USB 3.0 representa uma mudança significativa na operação básica do USB. Para simplificar a experiência do usuário, o USB 3.0 foi projetado para ser *plug-and-play* compatível com o USB 2.0.

O USB é um protocolo de barramento serial baseado em *token* e agendado por *host*. O USB permite a conexão de até 127 dispositivos em um único controlador *host* USB, porém há um limite máximo de 7 camadas de dispositivos, o que significa que no máximo 5 *hubs* podem ser conectados em linha.

O USB tem uma topologia em estrela e em camadas. Na camada raiz está o *host* USB. Todos os dispositivos se conectam ao *host* diretamente ou por meio de um

hub. De acordo com as especificações do USB, um *host* USB só pode suportar no máximo sete camadas. Um Computador (PC) *host* pode ter vários controladores de *host*, o que aumenta o número máximo de dispositivos USB que podem ser conectados a um único computador mostrado na figura 21.

Figura 21 -- Amostra da Topologia USB



Fonte: https://www.gta.ufrj.br/grad/06_1/wusb/descricao.htm

Segundo Anderson (1997) os dispositivos podem ser conectados e desconectados à vontade. O PC *host* é responsável por instalar e desinstalar os *drivers* dos dispositivos USB conforme sua necessidade.

Um único sistema USB é composto por um *host* USB de um ou mais dispositivos USB. Também pode haver zero ou mais *hubs* USB no sistema. Um *hub* USB é uma classe especial de dispositivo. O *hub* permite a conexão de vários dispositivos *downstream* a um *host* ou *hub upstream*. Desta forma, o número de dispositivos que podem ser fisicamente conectados a um computador pode ser ampliado.

Um dispositivo USB é um dispositivo periférico que se conecta ao PC *host*. A gama de funcionalidades dos dispositivos USB está sempre aumentando. O dispositivo pode suportar uma ou várias funções. Por exemplo, uma única impressora multifuncional pode apresentar vários dispositivos ao *host* quando está conectada via

USB. Ele pode apresentar um dispositivo de impressora, um dispositivo de *scanner*, um dispositivo de *fax* etc.

Todos os dispositivos em um único USB devem compartilhar a largura de banda disponível no barramento. É possível que um PC *host* tenha vários barramentos, todos com sua própria largura de banda separada. Na maioria das vezes, as portas da maioria das placas-mãe são emparelhadas, de forma que cada barramento tenha duas portas *downstream*.

3 DESENVOLVIMENTO

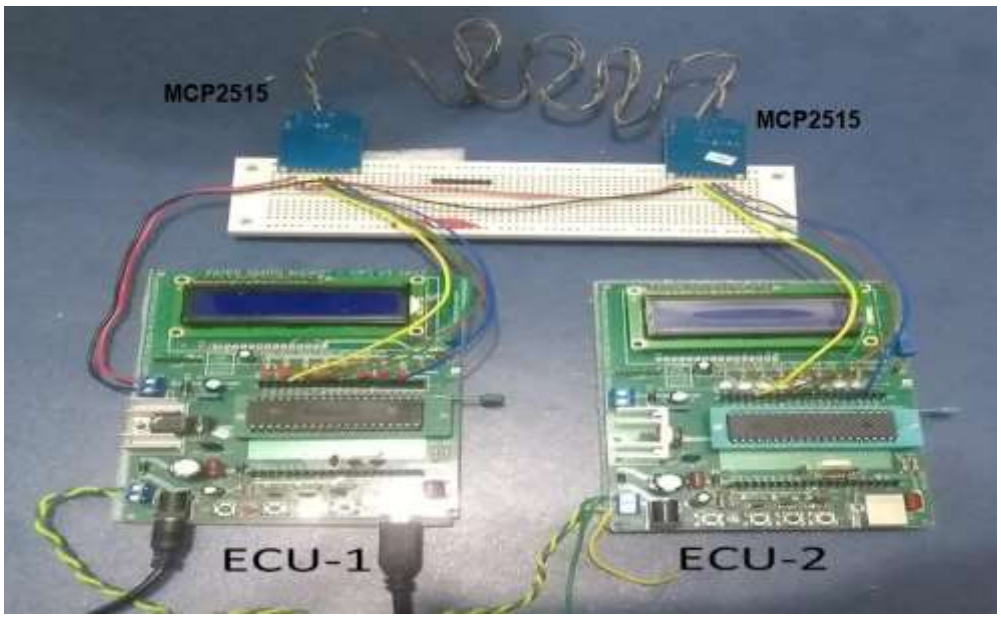
O projeto está dividido em 4 etapas:

- Desenvolvimento de *software* de comunicação USB, denominado de sCANu e implementação no PC;
- Desenvolvimento e teste de *firmware* para comunicação USB do PC com a ECU-1;
- Desenvolvimento e teste de *firmware* para comunicação CAN.
- Validação do projeto a partir de testes de comunicação, lendo dados extraídos da CAN.

O desenvolvimento dos programas, utilizou a linguagem C, para os FW e Python para o programa do computador, sCANu. Os códigos [(1)] estão disponíveis no Github. (GITHUB, 2020). O HW, como já informado, foi montado com as FATEC boards.

Os nós da rede, foram implementados com módulos CAN, que integram em uma placa, o controlador Microchip MCP2515 e transceptor TJA1050, conectados as CPUs através do barramento de comunicação *Serial Peripheral Interface* (SPI) e então ao barramento do CAN, criando assim uma rede. O barramento CAN tem em cada uma de suas terminações resistores de 120 Ω, para o casamento da impedância. A foto na figura 22, mostra o sistema montado.

Figura 22– Setup do projeto



Fonte: Autores (2021)

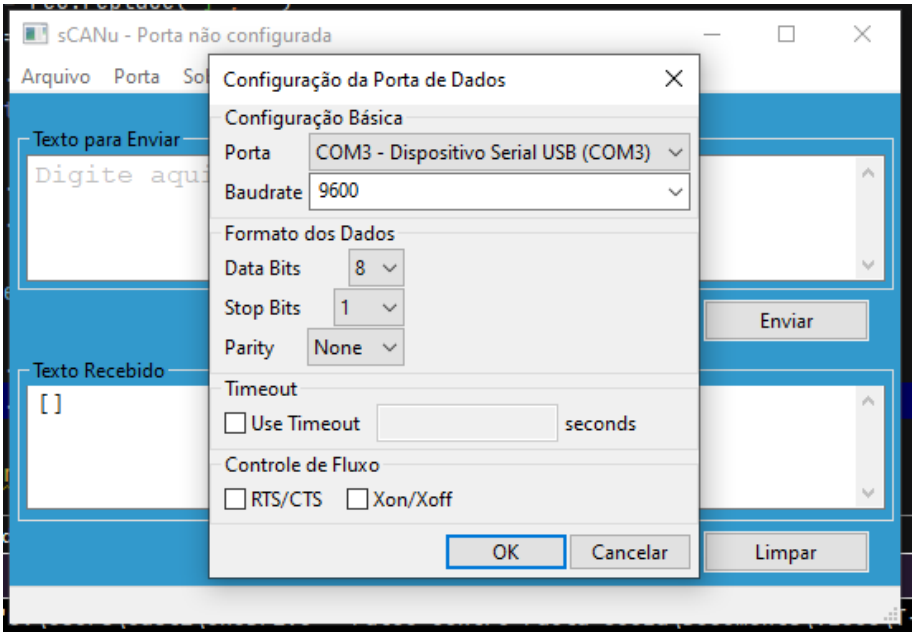
3.1 sCANu

sCANu é uma *Interface* Gráfica do Usuário (*Graphical User Interface*, GUI), para a comunicação com a ECU-1, através da porta USB do PC. Ela foi desenvolvida para demonstrar a possibilidade da comunicação entre o PC e o barramento CAN, através da FATEC *board*.

A linguagem utilizada foi o Python e suas bibliotecas (módulos de extensão), com o kit de ferramentas para a elaboração de GUI, wxPython e a pySerial, um módulo para acesso e controle da porta USB, utilizando como referência os exemplos encontrados em PYSERIAL (2020).

O SW sCANu tem como funções básicas o envio e recebimentos de dados e a configuração da porta USB, como se vê na figura 23, os itens possíveis de configuração.

Figura 23 -- *Framework* do SW sCANu



Fonte: Autores (2021)

3.2 Dicionário de Dados

O dicionário de dados é o conjunto de mensagens que podem ser transmitidas na rede CAN (GUIMARÃES, 2007). É implementado via *software* e deve ser o mesmo e ter a mesma versão em todos as ECU conectadas a rede, para garantir total compatibilidade.

Na tabela 1 é mostrado o dicionário de dados implementado para os testes de validação dos SW, com mensagens genéricas e tamanho de 1 *byte*.

Tabela 1 – Dicionário de Dados

ID Decimal	ID HEX	DLC	Dado HEX (DEC)	Enviado por:	Recebido por:	Taxa Transmissão - Recepção (milissegundos)
16	0x10	1	0x1F (31)	ECU-1 (task_0)	ECU-2 (task_2)	100 (ECU-1) - 500
32	0x20	1	0xFB (251)	ECU-1 (task_1)	ECU-2 (task_3)	200 (ECU-1) - 600
48	0x30	1	0xFA (250)	ECU-2 (task_0)	ECU-1 (task_2)	200 (ECU-2) - 400
53	0x35	1	0xFA (250)	ECU-1 (task_2)	ECU-2 (task_4)	400 (ECU-1) - 850
64	0x40	1	0xBB (187)	ECU-2 (task_1)	ECU-1 (task_3)	400 (ECU-2) - 850
69	0x45	1	0xBB (187)	ECU-1 (task_3)	ECU-2 (task_4)	850 (ECU-1) - 850
146	0x92	1	0xAA (170)	ECU-1 (comando_usb)	ECU-2 (task_5)	On demand (ECU-1) -1000

Fonte: Autores (2021) baseado em GUMARÃES (2007)

Os Ids 0x35 e 0x45, são usados para confirmar o recebimento de uma mensagem. A confirmação é feita, ao reenviar para o barramento CAN, o dado recebido, sob novos Ids, a ECU2, pode assim confirmar que os dados enviados foram recebidos.

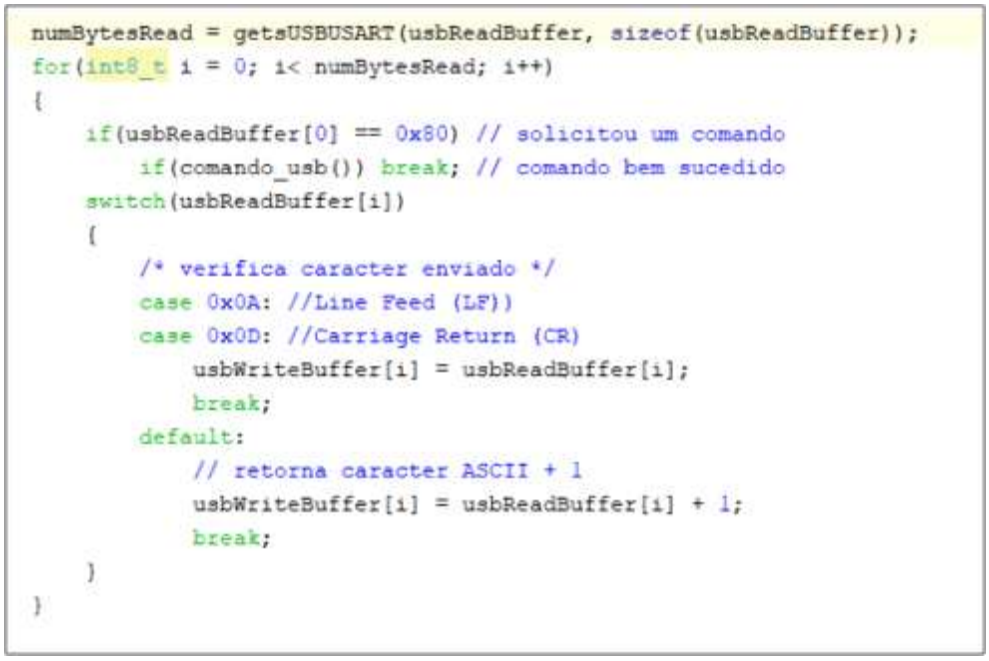
3.3 Teste de comunicação entre PC e ECU-1

Para validar a comunicação entre o PC e a ECU-1, foram realizados testes de envio de mensagens, através do sCANu, e verificada a resposta esperada conforme determinada no código.

Cenário 1

O trecho do FW da figura 24 trata as mensagens recebidas via USB, quando recebe um caractere que não tem significado pré-determinado para o sistema, entra no *default* e devolve o caractere recebido somando 1, ou seja, ASCII + 1. Na figura 25, se vê o sCANu enviando os caracteres “Castilho” e recebendo “Dbtujmip”.

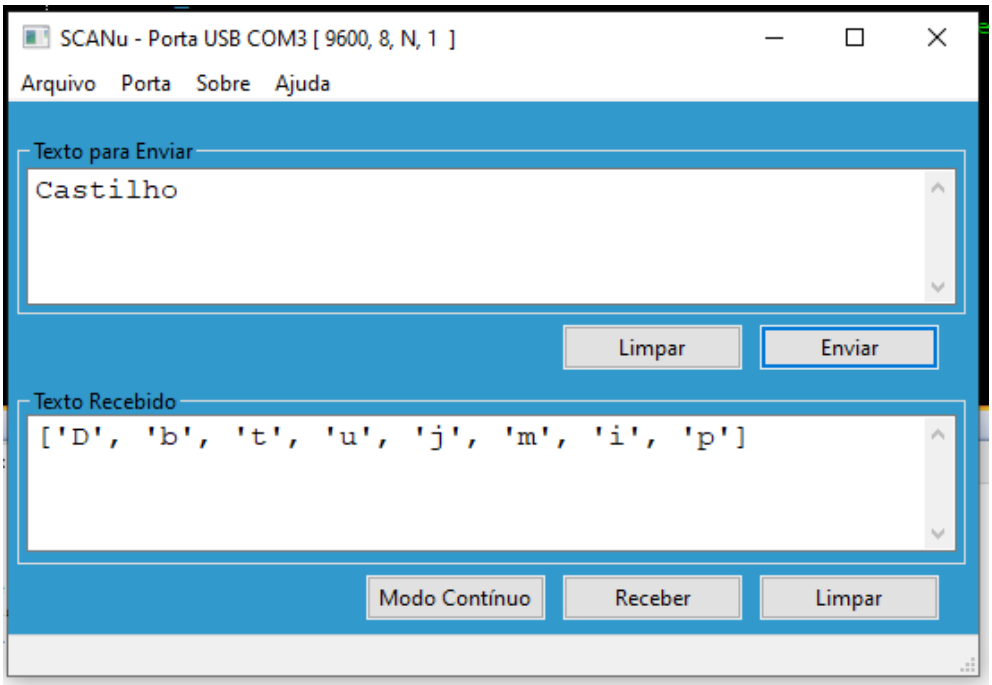
Figura 24 -- Trecho do firmware - USB-PIC



```
numBytesRead = getsUSBUSART(usbReadBuffer, sizeof(usbReadBuffer));
for(int8_t i = 0; i< numBytesRead; i++)
{
    if(usbReadBuffer[0] == 0x80) // solicitou um comando
        if(comando_usb()) break; // comando bem sucedido
    switch(usbReadBuffer[i])
    {
        /* verifica caracter enviado */
        case 0x0A: //Line Feed (LF))
        case 0x0D: //Carriage Return (CR)
            usbWriteBuffer[i] = usbReadBuffer[i];
            break;
        default:
            // retorna caracter ASCII + 1
            usbWriteBuffer[i] = usbReadBuffer[i] + 1;
            break;
    }
}
```

Fonte: Autores (2021)

Figura 25 -- Comunicação entre PC e ECU-1



Fonte: Autores (2021)

Cenário 2

Ao enviar um código, pela USB, pode ser modificado o estado dos periféricos conectados ao PIC. Para a validação, foi enviado um código, pelo sCANu e o FW da ECU-1, liga ou desliga o LED 7, conectado à porta RB7, conforme o comando enviado, conforme demonstrado na figura 26, onde se vê o trecho do código que trata os comandos enviados.

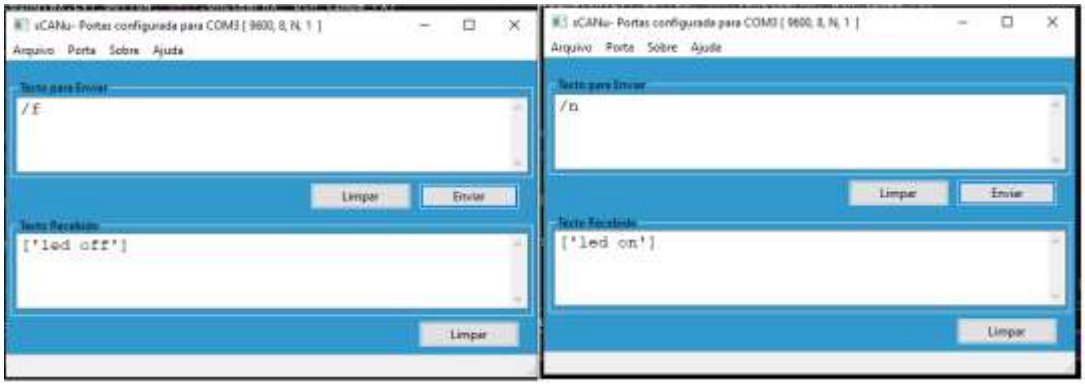
Figura 26 – Trecho do código para comandar Leds

```
        break;
    case 0x6e: // enviou n para acender o led
        LATBbits.LATB7 = OFF; // Acende o Led7
        for(uint8_t t = 0; t<6; t++)
        {
            usbWriteBuffer[t] = led_on[t];
        }
        numBytesRead = 6;
        break;
    case 0x66: // enviou f para apagar o led
        LATBbits.LATB7 = ON; // Apaga led7
        for(uint8_t t = 0; t<7; t++)
        {
            usbWriteBuffer[t] = led_off[t];
        }
        numBytesRead = 7;
        break;
    default:
```

Fonte: Autores (2021)

No sCANu, conforme a tela da figura 27, se vê como é possível a interação com o usuário. Ao digitar “/” (barra) é feito uma solicitação de tratamento de comando, assim o próximo caractere pode ser usado como comando.

Figura 27 – Comando para PIC através da USB



Fonte: Autores (2021)

3.4 Teste de comunicação do barramento CAN

No barramento CAN estão trafegando vários dados conforme, de acordo com o dicionário de dados, da tabela 1, já mostrada anteriormente. Esses dados foram verificados com o auxílio da ferramenta CANoe, obtendo resultado positivo.

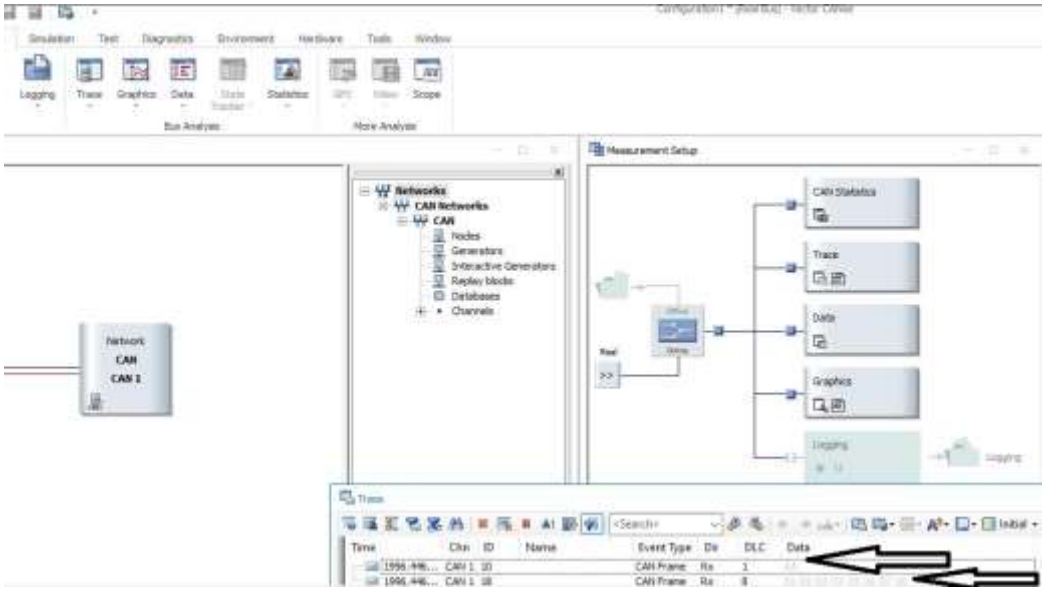
No FW da ECU-1 foram implementadas funções, chamadas de *tasks,* para controlar a comunicação com o barramento CAN. As *tasks* 0 e 1 (task enviam os dados dos Ids 0x10 e 0x20, na ECU-1 e 0x30 e 0x40 na ECU-2. Já as *tasks* 2 e 3 recebem os dados enviados pelas *tasks* 0 e 1. Na ECU-1 recebe o que foi enviado pela ECU-2 e na ECU-2, o que foi enviado pela ECU-1.

Para a ECU-2, foi acrescentada a *task* 4, que recebe alternadamente os Ids 0x35 e 0x45, que são enviados ao barramento pelas *tasks* 2 e 3 respectivamente da ECU-1. Os dados são enviados na sequência a sua leitura e quando a ECU-2, com a *task* 4 lê esses dados, confirma o tráfego dos dados no barramento entre as ECUs.

Também foi implementado na ECU-2, a task 5, que lê o Id 0x92, enviado ao barramento, após a ECU-1 receber um comando via sCANu.

Para validar o tráfego de dados do barramento foi feito um teste utilizando a ferramenta CANoe conforme a figura 28 que mostra os dados recebidos pelo simulador, evidenciado com setas, que obteve um resultado positivo para o trecho do código da figura 29.

Figura 28 -- Imagem do CANoe



Fonte: Autores (2021)

Figura 29 -- Trecho do código que envia na rede CAN

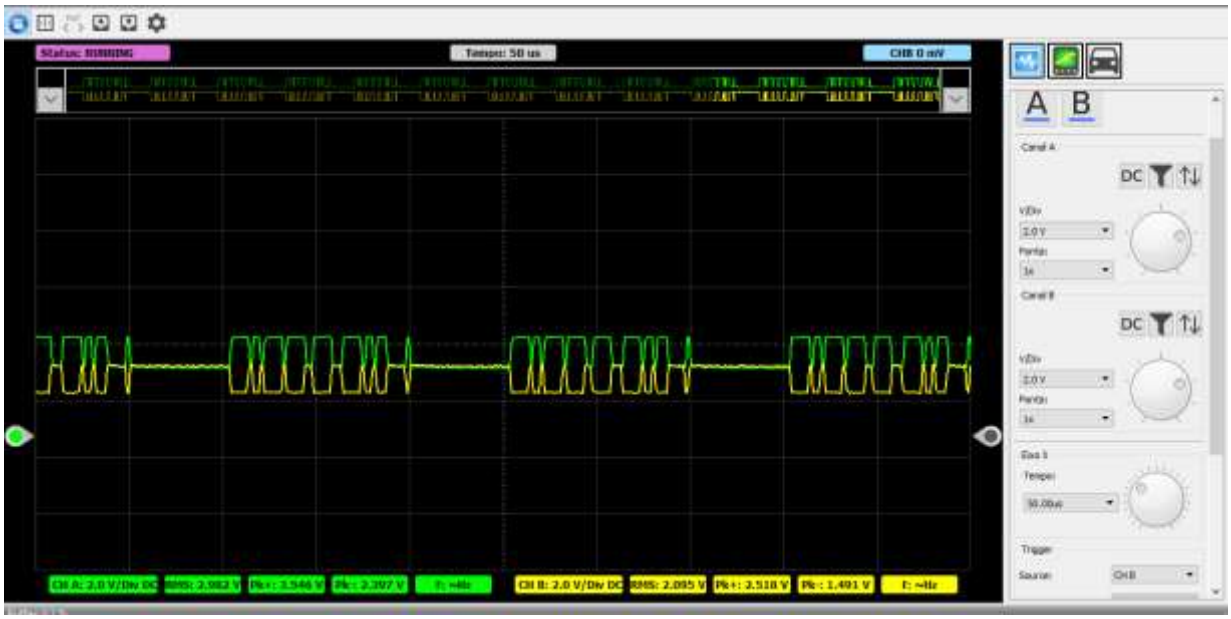
```
234
235  void task_1(void)
236  {
237      //Envia para CAN
238      uint8_t data_new = 0xFB;  // 251
239      MCP2515_ENVIA(0x20, 1, data_new);
240  }
241
242
243  void task_2(void)
244  {
245      uint8_t data_rcv[8] = {0,0,0,0,0,0,0,0};  // Dados recebido da CAN
246      uint8_t id30 = 0x30;
247      uint8_t message_one[] = "ID     :         ";
248
249      MCP2515_RECEBE(&id30, &length, data_rcv); //Recebe dados da CAN id 0x30.
250      posicao_cursor_lcd(1,0); escreve_frase_ram_lcd(message_one);
251      posicao_cursor_lcd(1,4); escreve_inteiro_lcd(id30);
252      posicao_cursor_lcd(1,9); escreve_inteiro_lcd(data_rcv);
253      do { LATBbits.LATB5 = ~LATBbits.LATB5; } while(0); //Alterna LED 7
254      MCP2515_ENVIA(0x35, 1, data_rcv[0]);
255  }
256  void task_3(void)
```

Fonte: Autores (2021)

3.5 Osciloscópio

Com o osciloscópio obtivemos uma imagem gráfica dos dados circulando pelo barramento, figura 30. Esse gráfico representa cada um dos campos do frame de dados, em relação ao tempo.

Figura 30 – Leitura do tráfego de dados via Osciloscópio



Fonte: Autores (2021)

4 CONCLUSÃO

Com esse trabalho observou-se que o barramento CAN pode ser implementado com baixo custo e é um poderoso sistema de comunicação entre sistemas eletrônicos, que compartilham informações.

Demostrou-se que com uma interface USB, pode-se ter um controle através de um PC, com a possibilidade de extração de dados, otimizando o gerenciamento de sistemas, seja para a manutenção ou para a configuração.

O trabalho mostrou os pontos que precisam de atenção redobrada numa implementação, como as ligações entre os componentes que devem garantir perfeito contato e blindagem aos ruídos do meio ambiente. Outro ponto onde se teve uma atenção redobrada devido sua complexidade de casamento, é a configuração dos tempos, que precisa ser bem definido, relacionando os *clocks* dos componentes com os tempos de *bit* da CAN.

Ainda, os FW precisam ser organizados em relação as suas funções, determinando como será o controle da comunicação e do periférico, estabelecendo hierarquias bem definidas para que cada tarefa seja cumprida.

5 PROPOSTAS FUTURAS

Melhorar sCANu para ter mais funções incluindo uma para leitura em modo contínuo, evoluindo para um escâner automotivo, por exemplo.

Implementar funções para melhorara a qualidade de recepção e transmissão de dados, como uma para *flushing* dos *buffers*.

Integrar com outros sistemas CAN, como a de um veículo real, por exemplo.

Ampliar os estudos para melhorar o entendimento e definições da rede CAN, como a temporização, esclarecendo as relações de tempo de *bit* com *clock* da rede e *clock* dos microcontroladores.

REFERÊNCIAS

USB system architecture. Reino Unido: Addison-Wesley Developers Press, 1997. (ISBN:9780201461374,0201461374)

BOSCH, Robert. **Manual de Tecnologia Automotiva**: Tradução 25ª. edição alemã. Helga Madjderey, Gunter W. Prokesch, Euryale de Jesus Zerbini, Suely Pferferman. São Paulo: Edgard Blücher, 2005. 1231 p.

CIA. **CAN in Automation**: CAN History. Disponível em <https://www.can-cia.org/en/can-knowledge/>. Acesso em: 25/10/2021, as 23:12.

GITHUB. **Códigos dos programas**. 2021. Disponível em <https://github.com/AntonioCastilho/TCC_Code_for_CAN_Network> Acesso em: 05/12/2021, as15:25.

GUIMARÃES, Alexandre de Almeida; **Eletrônica E>barcada Automotiva**. 1ª Edição. Editora Erica, 2007. (ISBN: 9788536501574)

GUIMARÃES, Alexandre de Almeida; SARAIVA, Antonio Mauro. **Um roteiro de implementação de uma rede CAN (Controller Area Network**). Anais. São Paulo: AEA, 2003.

MARQUES, M. A. **CAN Automotivo: Sistema de monitoramento**. 2004. 150 f. Dissertação programa de pós-graduação (Mestrado em Engenharia Elétrica) — Universidade Federal de Itajubá. 2004.

MICROCHIP TECHNOLOGY INC. (ed.). **PIC18F2455/2550/4455/4550 Data Sheet**. Chandler (USA): Microchip, 2009. 436 p. Disponível em: <https://www.microchip.com/en-us/product/PIC18F4550#document-table>. Acesso em: 20 nov. 2021.

PYSERIAL. **Module encapsulates the access for the serial port**. 2020. <https://github.com/pyserial/pyserial>. Acessado em: 7/11/2021, as 15:16.

ROBERT BOSCH GMBH (Alemanha) (ed.). **Bosch Automotive Electrics and Automotive Electronics**: systems and components, networking and hybrid drive. 5. ed. Plochingen: Springer Vieweg, 2007. 530 p.

SANTOS, Gláucio; **Redes de Comunicação**. Faculdade de Tecnologia – Fatec - Santo André, 2020. Notas de Aula