UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DIRETORIA DE PESQUISA E PÓS-GRADUAÇÃO
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
CURSO DE ESPECIALIZAÇÃO EM SISTEMAS EMBARCADOS PARA INDÚSTRIA AUTOMOTIVA

LUIS ALBERTO GAVLOVSKI

PROJETO DE UMA ARQUITETURA DISTRIBUÍDA PARA SISTEMAS AUTOMOTIVOS: ARDUINO E SHIELD CAN

MONOGRAFIA DE ESPECIALIZAÇÃO

CURITIBA
2020

LUIS ALBERTO GAVLOVSKI

PROJETO DE UMA ARQUITETURA DISTRIBUÍDA PARA SISTEMAS AUTOMOTIVOS: ARDUINO E SHIELD CAN

Monografia de Especialização, apresentada ao Curso de Especialização em Sistemas Embarcados para Indústria Automotiva, do Departamento Acadêmico de Eletrônica – DAELN, da Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para obtenção do título de Especialista.

Orientador: Prof. Dr. Max Mauro Dias Santos

CURITIBA
2020

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Curitiba

Diretoria de Pesquisa e Pós-Graduação
Departamento Acadêmico de Eletrônica
Curso de Especialização em Sistemas Embarcados para Indústria Automotiva

TERMO DE APROVAÇÃO

PROJETO DE UMA ARQUITETURA DISTRIBUÍDA PARA SISTEMAS AUTOMOTIVOS: ARDUINO E SHIELD CAN

por

LUIS ALBERTO GAVLOVSKI

Esta monografia foi apresentada em 04 de Maio de 2020 como requisito parcial para a obtenção do título de Especialista em Sistemas Embarcados para Indústria Automotiva. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____
Prof. Dr. Max Mauro Dias Santos
Orientador

_____
Prof. Dr. Kleber Kendy Horikawa Nabas
Membro titular

_____
Prof. M. Sc. Omero Francisco Bertol
Membro titular

- O Termo de Aprovação assinado encontra-se na Coordenação do Curso -

DEDICATÓRIA

Dedico este trabalho a minha família que sempre me incentivaram na dedicação aos estudos, principalmente a minha esposa, meu suporte nas horas que estive ausente me dedicando aos estudos.

RESUMO

GAVLOVSKI**,** Luis Alberto**. Projeto de uma arquitetura distribuída para sistemas automotivos: Arduino e Shield CAN**. 2020. 56 p. Monografia de Especialização em Sistemas Embarcados para Indústria Automotiva, Departamento Acadêmico de Eletrônica, Universidade Tecnológica Federal do Paraná. Curitiba, 2020.

O uso extensivo de novas tecnologias e sistemas de comunicações nos automóveis se tornou um desafio para os fabricantes manterem-se competitivos e com diferenciais que conquistem os consumidores, agregando novos recursos aos seus produtos, mas sem abrir mão da segurança, confiabilidade e qualidade. Ao longo dos anos, o uso da eletrônica, sensores e atuadores vem se tornando um elemento cada vez mais protagonista nos veículos automotivos, desempenhado um papel fundamental no controle dos diversos sistemas automotivos. Isto fez surgir o uso de redes de comunicação, bem como as unidades eletrônicas de controle (ECUs), que inicialmente operavam isoladamente, restritas a partes específicas do automóvel, mas gradualmente integraram-se aos demais subsistemas. Esta integração fez surgir a necessidade de uma comunicação eficiente entre estas unidades, o que muitas vezes não podia ocorrer de forma direta, na mesma velocidade, sendo necessário o uso de um dispositivo adaptador para viabilizar esta comunicação, foram então desenvolvidos dispositivos conhecidos como *gateways*. O objetivo deste trabalho é apresentar além da teoria relacionada a esses dispositivos, a construção de um ambiente prático de uma arquitetura distribuída utilizando Arduinos para simular ECUs e controladores *shield* CAN, representando duas sub-redes, funcionando em velocidades distintas, uma com velocidade de 250 Kbits/s e outra com velocidade de 500 Kbits/s, que poderiam representar como exemplos as redes relacionadas a carroceria e *dashboard* e outra representando a rede de gerenciamento do motor. Essas duas sub-redes foram conectadas a um terceiro Arduino com duas CAN Shields fazendo a função de *gateway* realizando o roteamento dos dados entre essas duas sub-redes.

Palavras-chave: Sistemas embarcados. Redes automotivas. Arduino. Shield CAN. Gateway.

ABSTRACT

GAVLOVSKI**,** Luis Alberto. **A distributed architecture project for embedded systems: Arduino and Shield CAN**. 2020. 56 p. Monografia de Especialização em Sistemas Embarcados para Indústria Automotiva, Departamento Acadêmico de Eletrônica, Universidade Tecnológica Federal do Paraná. Curitiba, 2020.

The extensive use of new technologies and efficient communication systems in vehicles has become a challenge for vehicle manufactures, mainly to keep themselves competitive, while differentiating them from the competitors, develop new resources is essential to get the consumers attention, adding new valuable resources to their products, but at the same time, preserving the security, reliability and quality. Over the years, electronic devices are getting improvements, and performing a main role over the controls of vehicles sensors and actuators, this led to the introduction of data communication networks and electronic control units (ECUs), that initially, operated in a standalone mode, in specific vehicle systems, but gradually started the integration with another onboard subsystems. This integration demanded an efficient data exchange, which sometimes could not interoperate directly due to non-compatible speed exchange rates. To make this happen it was introduced devices know as gateways. This paper presents, in addition to the theory related to these devices, the construction of a practical environment of a distributed architecture using Arduinos to simulate ECUs and shield CAN controllers, representing two subnets, operating at different timing speeds, one operating of 250 Kbits / s and another in 500 Kbits / s, they are representing, as instance, networks related to the body and dashboard, and another representing the engine network. These two subnets were connected to a third Arduino using two CAN Shields, and this Arduino is acting as a gateway routing the data between these two subnets.

Keywords: Embedded systems. Automotive networks. Arduino. CAN shield. Gateway.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

LISTA DE SIGLAS E ACRÔNIMOS

ABS	*Anti-lock Breaking System*
ADAS	*Advanced Driver-Assistance Systems*
CAN	*Controller Area Network*
CI	Circuito Integrado
CISC	*Complex Instruction Set Computer*
CPU	*Central Processing Unit*
ECU	*Electronic Control Unit*
EEPROM	*Electrically Erasable Programmable Read-Only Memory*
IIC	*Inter-Integrated Circuit*
ISO	*International Standardization Organization*
LIN	*Local Interconnect Network*
MISO	*Master Input Slave Output*
MOSI	*Master Output Slave Input*
MOST	*Multimedia-Oriented System Transport*
OSI	*Open Systems Interconnection*
PWM	*Pulse Width Modulation*
RAM	*Random Access Memory*
RISC	*Reduced Instruction Set Computer*
SPI	*Serial Peripheral Interface*
USB	*Universal Serial Bus*

1 INTRODUÇÃO

Devido ao crescente número de unidades de controle, sensores e atuadores embarcados nos veículos automotores, criou-se a necessidade de desenvolvimento de arquitetura de barramentos para aperfeiçoar a comunicação e o controle destes dispositivos, não sendo incomuns veículos fazendo o uso de inúmeros barramentos que devido aos conceitos de controle moderno exigem que estes dispositivos e barramentos possam ser interligados entre si, esta conexão é viabilizada pelo uso de dispositivo conhecidos como *gateways*, que no caso automotivo, permite conectar diversas redes e barramentos operando de forma distinta.

A CAN (em inglês *Controller Area Network*) é uma das tecnologias de comunicação mais empregadas para a troca de dados críticos, sua aplicação abrange desde redes de baixo custo até redes de altas velocidades empregadas em áreas industriais, médico hospitalares e automotiva. As características do protocolo CAN permitem a integração de diversos dispositivos utilizando e trocando dezenas de milhares de mensagens em um mesmo barramento, mas isto acarreta em um problema de largura banda e a limitação de alguns dispositivos operarem na mesma faixa de operação, sendo necessário segmentar os barramentos para operarem em domínios distintos.

A maioria das atuais implementações de *gateways* são baseadas em software embarcado em uma unidade dotada de diversas interfaces para a conexão dos barramentos CAN, essas interfaces compartilham uma unidade de memória e controlador comum. Unidades *gateway* normalmente possuem recursos de monitoramento, além de manipularem uma grande quantidade de dados trocados entre várias unidades CAN, o *gateway* faz o uso do monitoramento de tráfego dos barramentos para permitir que as mensagens presentes no barramento possam ser enviadas pela rota mais rápida para maior confiabilidade no gerenciamento de mensagens críticas. A transmissão é feita com base nas prioridades obtidas dos identificadores das mensagens. A interconexão entre barramentos CAN pode ser feita por meio de dispositivos *gateways* responsáveis pela transição dos dados entre esses barramentos.

1.1 TEMA

Projeto de uma arquitetura distribuída automotiva para fins didáticos a fim de apresentar os principais componentes ECU e rede CAN. O sistema consiste em uma rede de alta velocidade que suporta funções para gerenciamento de motor e uma rede de baixa velocidade que suporta funções de carroceria e *dashboard*. Um gateway que conecta as duas sub-redes é desenvolvido para mostrar suas funcionalidades.

1.2 DELIMITAÇÃO DO TEMA

Um veiculo terrestre atual é composto de subsistemas controlados eletronicamente por ECU que são interligados por redes de comunicação como LIN, CAN, FlexRay, MOST e Ethernet para que as funções nas ECU possam comunicar entre elas através por sinais pela troca de mensagens na memória ou rede de comunicação. O próprio *gateway* atualmente embarcado nos automóveis exerce outra dezena de funções, contudo não existem muitas literaturas e fontes de pesquisa disponíveis abortando como desenvolver uma arquitetura distribuída com duas sub-redes sendo de alta e baixa velocidade. Este trabalho será focado ao uso de tecnologia de rede CAN em que é amplamente usada no setor automotivo em sistemas embarcados, uma breve introdução aos principais conceitos de *gateways* em uso pela indústria automotiva, e a demonstração através de fotos e captura de telas de um protótipo, para introduzir o conceito de roteamento usado nos *gateways* para interligar os diferentes domínios de um veículo que operam em velocidades de comunicação distintas.

1.3 PROBLEMA

Um *gateway* é um dispositivo similar a um roteador, que gerencia o fluxo de dados entre duas ou mais redes. Para viabilizar esta comunicação entre as redes, este dispositivo deve possuir interfaces em cada uma das redes e muitas vezes ter a capacidade de determinar quais mensagens serão encaminhadas e se

reconfigurações ou conversões dos pacotes de dados serão necessários. Existem diversos cenários nos quais um *gateway* pode ser utilizado, e os requisitos serão diferentes em cada um desses cenários. Nesse trabalho pretende-se simular uma arquitetura distribuída de redes, simular ao que pode ser encontrado em veículos terrestres, com duas sub-redes CAN operando em velocidades distintas, porém permitindo que as informações possam trafegar entre elas através de um dispositivo emulando a função de um *gateway.*

1.4 OBJETIVOS GERAIS

Análise de alguns resultados experimentais, através de imagens de mensagens sendo transmitidas entre as redes CAN, demonstrando o recebimento, o tratamento e a retransmissão da mensagem pelo dispositivo *gateway*; a simulação de um atuador, representado no protótipo por um potenciômetro deslizante, e o resultado desta ação na forma de ativação progressiva de dispositivos leds conectados a um receptor, emulando, por exemplo, um acelerador de um automóvel e seu respectivo conta-giros.

1.5 ORGANIZAÇÃO DO TRABALHO

Esta monografia encontra-se subdividida em cinco seções principais:

- A primeira seção faz a uma breve introdução do tema e também a motivação e a estrutura geral do trabalho.
- Na segunda seção são apresentados os referenciais teóricos, os conceitos das comunicações seriais, a evolução nas interligações das ECUs, a camada OSI envolvida na comunicação do barramento CAN, uma introdução ao funcionamento da rede CAN, seus *frames*, características e normas, também em síntese, apresenta-se as principais características e conceitos dos *gateways* utilizados na indústria automotiva, e finalmente uma sucinta descrição dos elementos utilizados no protótipo desenvolvido para demonstrar de forma didática o funcionamento do recurso de roteamento do *gateway*.

- Na terceira seção são descritos os principais módulos utilizados no protótipo, bem como as principais características.
- Na quarta seção o protótipo é apresentado com os esquemáticos de montagem e as evidências coletadas através de fotos e capturas de tela durante o seu funcionamento para ilustrar os resultados do seu funcionamento.
- Finalizando com as conclusões gerais e desafios futuros apresentados na quinta seção, bem como anexos relacionados ao desenvolvimento dos códigos de programação utilizados no protótipo.

2 DESENVOLVIMENTO

Em uma transmissão de sinais digitais seriais, a informação é transmitida na forma de sequência de *bits*, sendo a variação do sinal a representação do número de *bits*, conforme demonstrado na Figura 1.

Figura 1 – Representação de uma comunicação serial

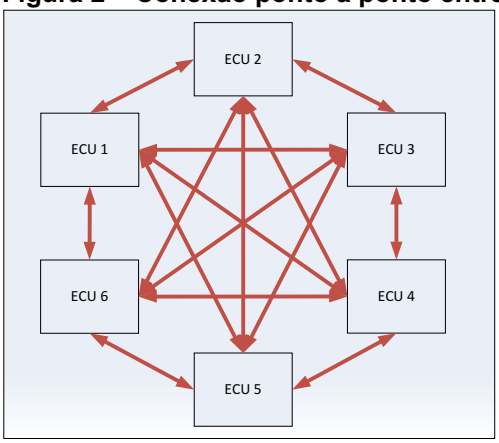
Fonte: Autoria própria.

Em um meio físico, a transmissão lógica de bits ocorre na forma de variações de tensões e dependendo da aplicação, um ou mais fios podem ser utilizados, Nas comunicações que utilizam um único fio, os níveis de tensões são diferenciados como níveis alto e baixo para representar as variações de tensões, enquanto que em comunicações de utilizam dois fios, a diferença simultânea entre as tensões entre estes dois fios são utilizados para representar a variação de tensão.

A comunicação serial envolve o empacotamento dos dados, conhecido como *frames*, que resumidamente consiste de um endereço ou identificador, a informação propriamente dita e um conjunto de bits gerados através de um mecanismo de cálculo para assegurar a que a informação transmitida corresponde à recebida, mecanismo conhecido como *checksum.*

Em um passado recente, a forma de transferir dados entre dispositivos eletrônicos era através da conexão ponto-a-ponto entre cada dispositivo, sensor e atuador utilizando uma conexão dedicada, conforme exemplificado na Figura 2, levando ao uso de uma quantidade enorme de cabos, ocasionado excesso de peso, problemas de conexões, custos elevados e dificultando a expansão do sistema. Atualmente, alguns automóveis de luxo fazem o uso de mais de uma centena de ECUs e consomem mais de 4 km de cabos, o que seria inviável de ser instalado sem o recurso de um barramento de comunicação entre as ECUs.

Figura 2 – Conexão ponto a ponto entre ECUs



Fonte: Autoria própria.

Para resolver estes problemas, uma abordagem completamente nova foi adotada, a introdução de um novo meio de comunicação por compartilhamento de barramento baseado em comunicação serial, permitindo que todas as ECUs possam ter a capacidade de comunicar-se entre elas através do uso deste barramento.

2.1 O MODELO OSI

O modelo OSI (em inglês, *Open System Interconnection*) é o modelo padrão para a definição de protocolos de comunicação constituído de sete camadas nomeadas respectivamente como Física, Enlace de Dados, Rede, Transporte, Sessão, Apresentação e Aplicação (TANENBAUM, 2010).

A Figura 3 destaca em verde as partes contempladas pelo padrão CAN e o conjunto de 7 camadas de acordo com o modelo OSI (*Open System Interconnection*) da ISO (ISO/IEC 7498-1).

Figura 3 – O protocolo CAN dentro do modelo OSI

Camada	Nomenclatura	
7	Aplicação	Definido pelo usuário
6	Apresentação	Não aplicáveis ao protocolo CAN
5	Sessão	
4	Transporte	
3	Rede	
2	Enlace de Dados	Definidas pela norma ISO11898
1	Física	

Fonte: Adaptado de Farinelli e Stevan Jr (2016).

2.2 HISTÓRICO DA REDE CAN

Em meados dos anos 80, a Bosch iniciou o desenvolvimento um sistema de comunicação serial a qual foi dado o nome de CAN (em inglês, *Controller Area Network*) (LUGLI; SANTOS, 2009), a qual continua plenamente em uso devido as suas características de alta confiabilidade na transmissão de dados que satisfazem os requisitos de comunicação em tempo real.

A tecnologia CAN foi padronizada em 1.994, e é descrita por documentos da ISO. A ISO 11898-1 descreve o protocolo CAN em relação ao modelo de comunicação. A ISO 11898-2 e ISO 11898-3 cobrem as duas camadas físicas: a) rede CAN de alta velocidade (*high speed CAN*); e b) rede CAN de baixa velocidade *(low speed CAN)*. Elas diferem principalmente pelas voltagens e taxas de transmissão que elas utilizam.

Inicialmente a diversidade de fabricantes de módulos eletrônicos e a dificuldade de comunicação entre eles geraram dificuldades nos seus projetos, que acabaram por criar novas normas, como por exemplo, para o setor agrícola foi criada a ISO 11783-1 que faz uso do protocolo CAN.

Normas que fazem uso do protocolo CAN:

- NMEA 2000: aplicações navais e aéreas;
- SAE J1939: aplicações veículos comerciais;
- ISO 11783-1: aplicações agrícolas.

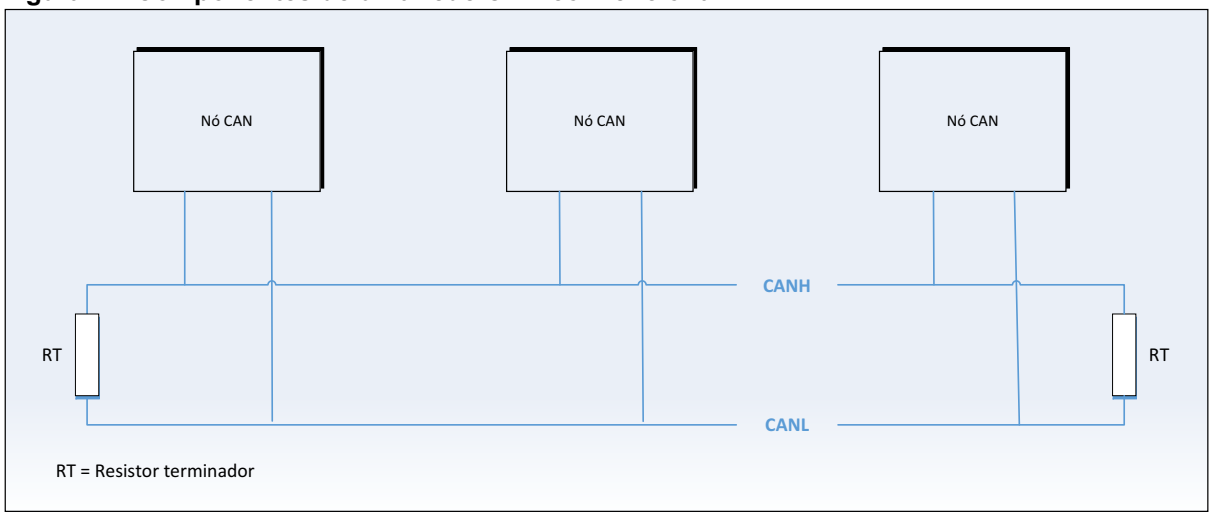
A CAN de baixa velocidade (ISO 11898-3) permite taxas de transferência de até 125 Kbit/s, com uso comum em áreas de convivência do automóvel, enquanto que a CAN de alta velocidade opera em velocidades de até 1 Mbit/s, e é aplicada principalmente no chassis e no conjunto de trem-de-força do automóvel (*powertrain* - sistema de propulsão que pode ter um motor ICE, máquina elétrica ou híbrida).

O meio de comunicação de dados pelo qual os dados se propagam é um par de fios trançado, tornando a rede CAN mais barata que outras nas quais às vezes são necessários cabos blindados para a transmissão de dados (FARSI; RATCLIFF; BARBOSA, 1999).

Uma ECU conectada a um barramento CAN é referenciada como um nó CAN, representado na Figura 4. Devido às características da topologia, é possível conectar qualquer dispositivo à rede simplesmente criando uma ramificação, pois

não é necessária a obstrução dos fios que compõem o meio de transmissão que possui capacidade para até 32 dispositivos (ou nós) compatíveis com o protocolo CAN (DI NATALE *et al.*, 2012).

Figura 4 – Componentes de uma rede CAN convencional



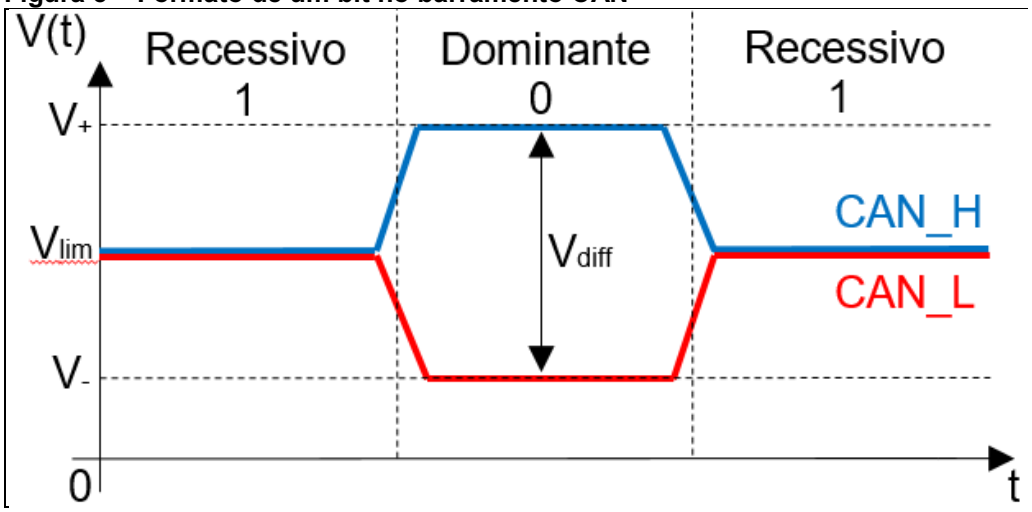
Fonte: Autoria própria.

Em redes de barramento aberto, se faz necessário a instalação de terminadores para evitar a reflexão de informações no barramento (RICHARDS, 2002), que pode causar a leitura incorreta ou até ilegibilidade dos dados pelos nós se a rede possuir baixa impedância. Para evitar esse efeito, de acordo com a norma ISO 11898-3, devem ser utilizados resistores de 120 Ohm nas terminações do barramento CAN.

As duas linhas, CAN_H (*em inglês CAN High*, ou linha CAN de alto nível de tensão) e CAN_L (em inglês *CAN Low*, ou linha CAN de baixo nível de tensão), representam o par de fios trançado pelo qual os dados trafegam, numa forma semelhante à da Figura 5. A tensão nominal das linhas CAN H e CAN L é de 2,5 V.

O que define um bit no barramento CAN é o módulo da tensão diferencial que circula nas duas linhas.

Figura 5 – Formato de um bit no barramento CAN



Fonte: Farinelli e Stevan Jr (2016).

De acordo com a norma ISSO 11898-3, a rede CAN é uma rede com prioridade de passagem de dados. Pela norma, um bit 1 é considerado como bit recessivo, e o bit 0 como bit dominante, em que a tensão diferencial $Vdiff$ é maior no bit dominante que no bit recessivo.

A CAN utiliza o esquema de codificação NRZ (*Non-Return to Zero*) em que o nível do sinal mantém-se constante ao longo do tempo, sendo que para o bit 0 é representado pelo nível baixo e para o bit 1 pelo nível alto.

A tensão de limiar, ou de modo comum, segundo a norma ISO 11898-3, deve estar entre -1 e 6V, mas é recomendado 2,5V como valor nominal. Devido ao CAN se tratar de um barramento com tensão diferencial, então a mesma norma prevê valores mínimos e máximos para considerar um bit recessivo ou dominante através dos parâmetros apresentados na Tabela 1.

Tabela 1 – Valores de tensões para bits recessivos e dominantes no barramento

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

Fonte: Autoria própria[1].

[1] Fonte: **ISO 11898-3:2006**. Disponível em: <https://www.iso.org/standard/36055.html>. Acesso em: 10 abr. 2020.

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

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

A Tabela 2 mostra quais as velocidades de transmissão e o comprimento máximo do barramento para cada uma delas.

Tabela 2 – Taxa de transmissão conforme o comprimento do barramento CAN

Taxa de Transferência Kbits/s	Distância Máxima (m)
5	10000
10	6700
20	3300
50	1300
100	620
125	530
250	270
500	130
1000	40

Fonte: Adaptado de Lugli e Santos (2009).

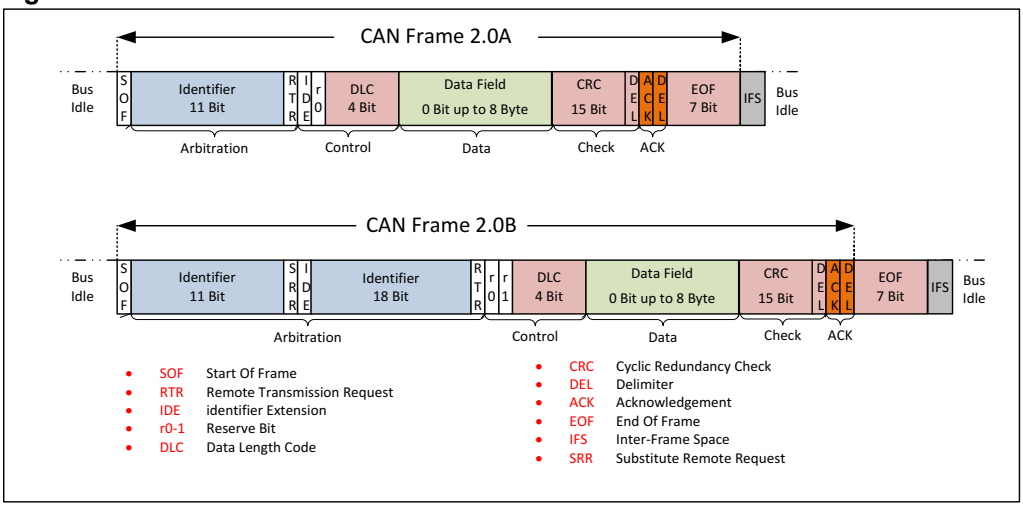
Em uma rede CAN, os nós diferem no volume de mensagens CAN que eles conseguem receber e enviar, bem como a frequência dessas mensagens, um nó da rede CAN pode receber, por exemplo, 5 mensagens em um intervalo de 10 milissegundos, enquanto outro nó recebe uma única mensagem a cada 100 milissegundos, o que necessita o desenvolvimento de diferentes arquiteturas que levam em consideração ou não o armazenamento de uma mensagem para posterior processamento.

2.3 ESTRUTURA DE UM QUADRO CAN

O barramento CAN é caracterizado por todos os componentes conectados ao barramento terem a capacidade de poderem “ouvir” todas as mensagens disponíveis no barramento (em inglês - *broadcast*), a decisão de fazer o processamento ou não da mensagem será determinada pela configuração do dispositivo. Não existe a possibilidade na rede CAN de se fazer uma transmissão seletiva, na qual uma mensagem é direcionada para um dispositivo específico (em inglês - *node addressing*).

A Figura 6 representa uma estrutura típica de um quadro CAN (em inglês, *frame*) para CAN 2.0A e CAN 2.0B, quando o barramento está ocioso, a diferença de tensão entre as linhas CAN H e CAN L é 0 V. O primeiro bit conhecido como início do quadro (SOF) representa a mudança do estado físico, alterando a diferença de tensão entre as linhas CAN H e CAN L para 5 V o que representa o estado o nível lógico 1. Esta mudança de estado é detectada por todos os transceptores das ECUs conectadas ao barramento.

Figura 6 – CAN *Frame*



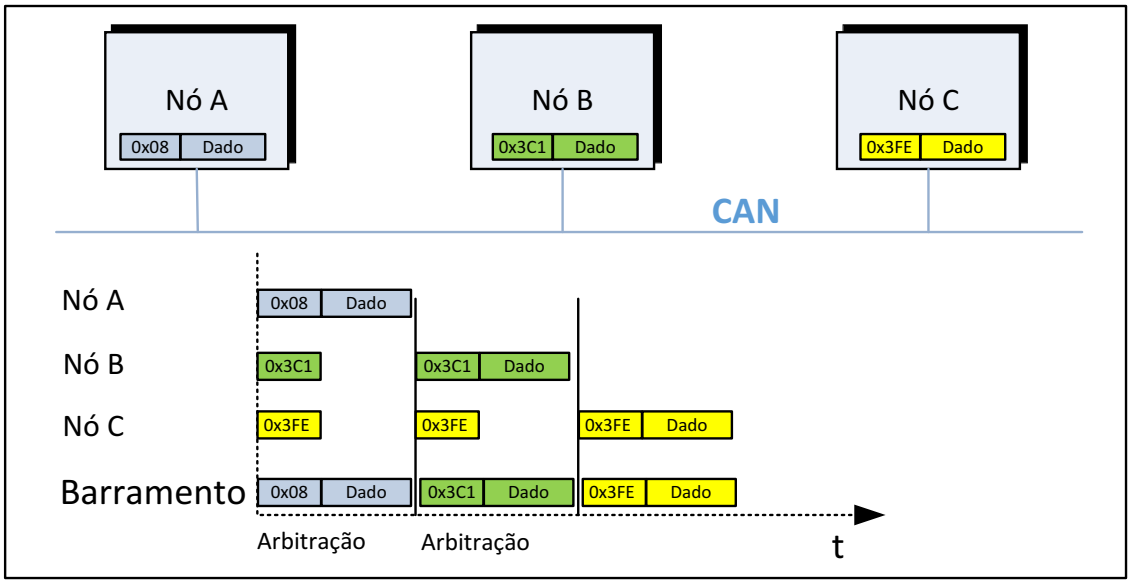
Fonte: Autoria própria.

Seguido do início do quadro, vem o identificador da mensagem (11 bits na CAN 2.0A e 29 bits na CAN 2.0B) juntamente com o RTR (*Remote Transmission Request* – utilizado em mensagens de requisição de pacotes), ambos fazem parte da fase de arbitração, que determina a prioridade de uma mensagem quando dois ou mais dispositivos conectados ao barramento, estão disputando o seu uso, como a

rede CAN trabalha com prioridade de acesso, as mensagens com um identificador menor, sempre terá prioridade de acesso ao barramento.

Como demonstrado na Figura 7, os três nós desejam ter o direito de transmissão no barramento ao mesmo tempo, uma vez que o protocolo CAN possui um mecanismo para impedir a colisão de pacotes, entra em ação a fase de arbitração, permitindo que os nós apresentem os identificadores das mensagens e ao mesmo tempo realizem a sua comparação bit a bit. Por padrão, o pacote com um identificador de menor valor numérico terá prioridade. Os nós B e C detectam que seus identificadores possuem uma prioridade menor e automaticamente irão cancelar a solicitação de transmissão e irão mudar o seu estado para o modo recebedor de mensagens no barramento, pois a mensagem sendo transmitida pelo nó A pode ser de interesse de um, ou de ambos. Os nós B e C não irão cancelar as suas transmissões, mas ao invés, eles tentaram novamente ao detectar que o barramento está disponível novamente, e caso ocorra uma nova disputa entre os nós B e C, uma nova fase de arbitração será iniciada, e desta vez, a transmissão do nó B terá prioridade, e mais uma vez, o nó C terá que aguardar pela disponibilidade do barramento para uma nova tentativa. Por definição um barramento é considerado como livre após uma sequência de 11 bits recessivos – sequência de onze 1s, presentes no final do *frame* (EOF) junto com o IFS (em inglês, *Inter-frame Space*), em um barramento de 1 Mbps, esta sequência dura um tempo de 11µs.

Figura 7 – Arbitração de acesso ao barramento para a transmissão de dados



Fonte: Autoria própria.

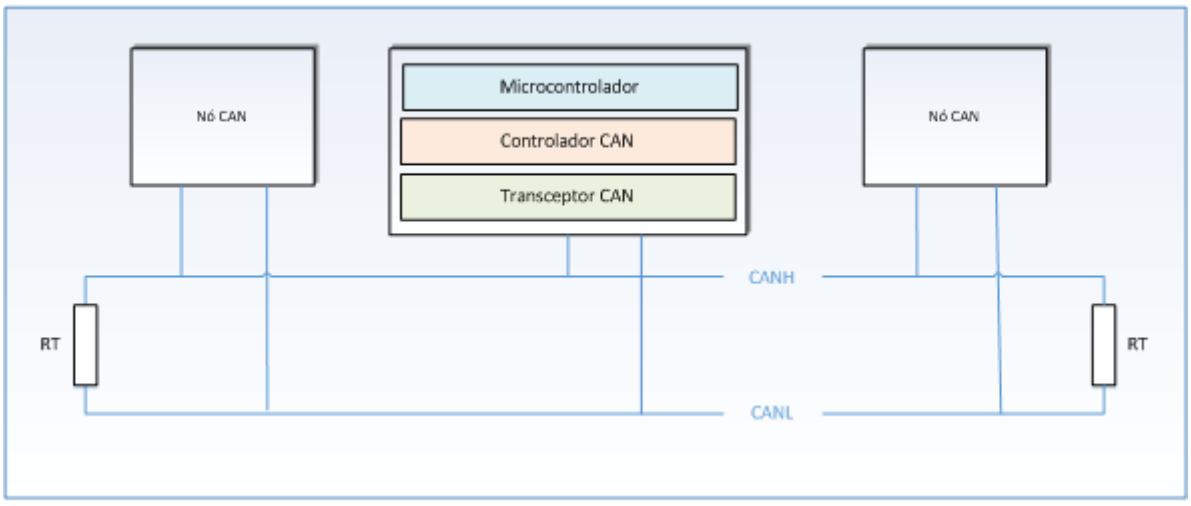
Após os 12 bits da fase de arbitração, vem 6 bits referente ao campo de controle. O primeiro bit referente ao IDE (em inglês, *Identifier Extension*), que define se o quadro CAN será 2.0 A ou 2.0 B, que é dominante (bit 0) para CAN 2.0 A, o bit seguinte r0 (em inglês, *Reserved bit*) é reservado ao protocolo e defino como dominante. Os 4 bits restantes (DLC – em inglês, *Data Length Code*) especificam o número de bytes de dados a serem transmitidos na mensagem, o campo de dados pode variar de 0 a 8 bytes em tamanho e possui o conteúdo relevante da mensagem, como por exemplo um valor presente em um sensor.

O campo do CRC (em inglês, *Cyclic Redundancy Check*) vem em seguida ao de dados e é utilizado para detectar erros na transmissão, consiste de 15 bits, seguido de um bit delimitador recessivo, o seu conteúdo é determinado pelo uso de um cálculo polinomial. O campo final, é o campo de confirmação de recebimento, são dois bits um ACK (em inglês, *Acknowledgement*) e DEL (em inglês, *Delimiter*), nessa etapa, o nó transmissor envia um bit recessivo, e como resposta, todos os outros nós que estão no barramento, irão enviar um bit dominante confirmando que a recepção da mensagem ocorreu de forma correta.

2.4 ESTRUTURA DE UMA ECU CONECTADA AO BARRAMENTO CAN

Uma ECU conectada ao barramento CAN contêm pelo menos 3 elementos funcionais que são essências, como mostrada na Figura 8.

Figura 8 – Principais elementos de uma ECU



Fonte: Autoria própria.

O micro controlador executa as operações definidas por software, sempre que uma mensagem precisa ser transmitida o micro controlador aciona o controlador e repassa a mensagem a ser transmitida. O controlador CAN encapsula a mensagem no formato do *frame* CAN, aciona o barramento CAN para obter o controle e passa a mensagem para o transceptor CAN na forma de uma sequência serial de bits. A sequencia de bits é convertido para níveis de voltagens pelo transceptor CAN e transmitidas no cabeamento da rede CAN. O barramento CAN exige o uso de resistores terminadores nas suas extremidades, pois os níveis de voltagem podem variar milhares de vezes por segundo o que ocasionaria reflexões em um barramento aberto, ocasionando a sobreposições do sinal, e a sua destruição.

2.5 O QUE É UM GATEWAY

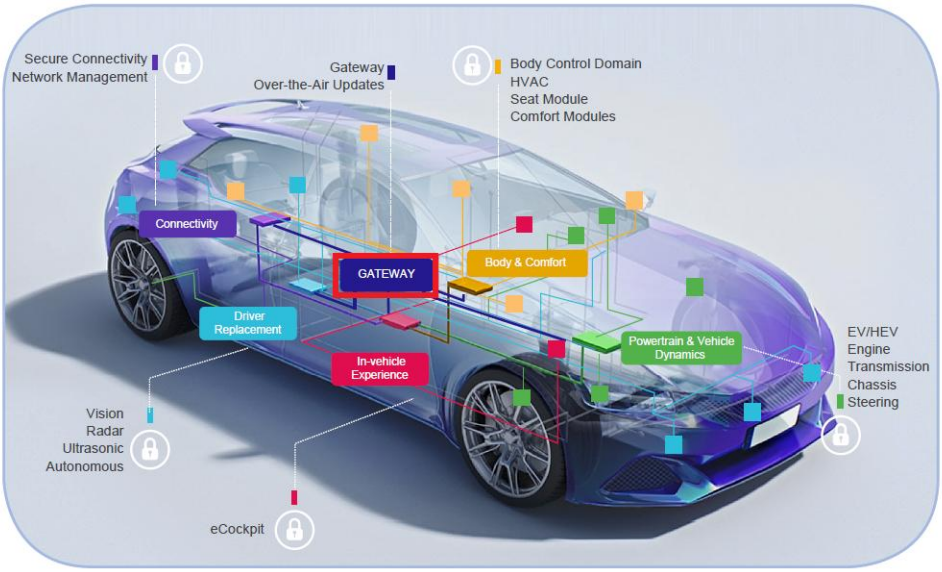
Em telecomunicações, o termo *gateway*, refere-se a um dispositivo ou nó que entre outras funções, seja capaz de realizar a comunicação entre partes distintas que não podem se conectar diretamente, por exemplo, duas ou mais redes operando com diferentes protocolos e/ou velocidades de transmissão.

Cabe ao *gateway* traduzir e adaptar os pacotes originários de uma rede, para que estes possam atingir o destinatário, e também, traduzir as respostas e devolvê-las ao local de origem da comunicação. *Gateways* habilitam a comunicação entre diferentes arquiteturas e ambientes, ele realiza a conversão dos dados de um ambiente para o outro, de tal modo que cada ambiente seja capaz de entender os dados.

Na indústria automotiva, tem-se g*ateways* sendo utilizados para conectar redes CAN abertas, que consiste na conexão de um trator com um implemento agrícola, ou um veículo com um trailer, por exemplo. Mas na indústria automotiva, os *gateways* não estão restritos ao uso de redes CAN, a crescente demanda por mais funcionalidades e recursos nos veículos modernos, estimularam a introdução de ambientes mais complexos e por consequência, o aumento de ECUs, que podem passar de uma centena em um único veículo, esses dispositivos operam com diferentes interfaces de rede e protocolos, pode-se citar aqui redes LIN, FlexRay, MOST, redes Ethernet e a própria CAN. Essas redes utilizam protocolos únicos e proprietários operando em um vasto espectro de taxas de transferência de dados

com aplicações distintas em um veículo automotor. Como demonstrado na Figura 9, o *gateway* destacado exerce a função de centralizar toda a comunicação processando os dados de uma forma segura e confiável através das redes heterogêneas presentes no veículo, ao mesmo tempo em que fornece o isolamento físico dessas redes, atuando como um elemento de segurança, conversor de protocolos e roteador de dados entre os diferentes domínios do veículo como o conjunto do trem-de-força, chassi, controle da carroceria, infotenimento, ADAS, etc. Uma nova função vem sendo incorporada aos *gateways* atuais, que é a capacidade de realizar a comunicação com o veículo através de redes sem fio (em inglês OTA – *Over-The-Air*), habilitando opções no uso de aplicações de monitoramento remoto do veículo e atualizações remotas do veículo, para citar alguns.

Figura 9 – Gateway atuando como elemento central de interligação



Fonte: NPX (2018).

As redes LINs operam em baixas velocidades (20 Kbps), principalmente no controle de sensores e atuadores, a rede CAN nos sistemas de média velocidade, especialmente na comunicação entre ECUs (1-5 Mbps), a Flexray opera nos sistemas que requerem comunicações em tempo real (10 Mbps), como nos sistemas de segurança crítica, aqui representados pelo sistema de freios ABS, *airbags*, controle de estabilidade, entre outros. Recentemente, redes MOST e *Ethernets* começaram a ser introduzidas no universo automotivo, para o controle de sistema de infotenimento, controles avançados de direção assistida (em inglês, ADAS - *Advanced Driver-Assistance Systems*), e interfaces de conexões com redes sem fio

3G, 4G, 5G e *wi-fi* (em inglês, *wireless*), podendo operar em velocidades superiores a 100 Mbps.

Um dispositivo *gateway* é o elemento central responsável por interligar todas essas redes automotivas heterogêneas, fornecendo o isolamento entre elas ao mesmo tempo em que realiza a translação e o roteamento dos dados entre os subsistemas funcionais do veículo, demonstrando ser um componente primordial no aperfeiçoamento dos veículos para a introdução de novos recursos.

2.6 RECURSOS DOS GATEWAYS MODERNOS

As principais funções de um *gateway* estão em fornecer segurança, comunicação estável entre as redes e ECUs e atuar como um elemento ponte entre as comunicações internas e externas do veículo. Cabe a este dispositivo também, garantir uma operação regular na transferência dos dados, assegurando que as ECUs possuem as informações necessárias no tempo correto, sem atrasos ou latências no tempo de propagação dos dados.

Existem muitos requisitos que um *gateway* deve atender para garantir um nível satisfatório. A Tabela 3 fornece um resumo dos principais recursos disponíveis nos *gateways* atuais.

Tabela 3 – Principais recursos de um *gateway*

Recurso	Descrição
Tradução de protocolos	Tradução de dados e controle de informações entre redes utilizando protocolos distintos para habilitar a comunicação entre elas.
Roteamento	Roteamento de dados para chegar ao destino final, podendo ocorrer entre redes distintas com conversão de protocolos.
Roteamento de diagnóstico	Roteamento de mensagens de diagnóstico entre dispositivos de diagnóstico externo e as ECUs.
Firewall	Filtragem de pacotes de rede baseados em regras, bloqueando transferência de dados para destinos não aprovados.
Espelhamento de mensagens	Captura dados de uma rede e transmite para outra rede para efeitos de diagnóstico ou armazenamento (*logging)*.
Detecção de intrusão	Monitora a rede e detecta anomalias que podem representar uma invasão.
Gerenciamento da rede	Gerencia o estado e configuração da rede e das ECUs conectadas, fornecendo suporte ao diagnóstico de falhas.
Gerenciamento de chaves e certificados	Processamento da segurança e armazenamento de chaves de segurança e certificados digitais para veículos com conexão à redes externas.

Fonte: NPX (2018).

Teoricamente o termo *Gateway* não é empregado corretamente nas aplicações automotivas nas quais ele é utilizado exclusivamente para conectar redes operando em diferentes velocidades, ou na conexão de domínios de redes distintos, operando em um mesmo sistema. Neste contexto, o termo ponte (em inglês, *bridge*), deveria ser utilizado para descrever um dispositivo de sistemas de comunicações que interliga e realiza o roteamento de sinais entre barramentos de rede. Este dispositivo não modifica os pacotes ou as mensagens, somente realiza a sua leitura, e os encaminha para diferentes segmentos do barramento.

Em aplicações automotivas e industriais, porém, o termo *gateway* normalmente é empregado mesmo quando utilizados para a transferência de dados entre barramentos, pois muitas vezes eles realizam funções adicionais, como por exemplo, a filtragem de mensagens para prevenir a sobrecarga de barramentos operando em velocidades reduzidas, integração de mensagens em um único pacote e a sincronização de mensagens baseadas em disparo por tempo (em inglês, *time-triggered*), que necessitam que a informação seja atualizada no tempo correto.

Em geral, *gateways* podem ser desenvolvidos através de software, mas como desvantagem do uso desta técnica, um grande volume de mensagens pode ocasionar uma sobrecarga na CPU, comprometendo a capacidade de manipular operações de tempo real, cruciais nos sistemas de segurança veiculares. Assim, o desenvolvimento de *gateways* por software deve priorizar a redução de demandas no uso da CPU.

2.7 CONCEITO DE *GATEWAYS*

O desenvolvimento de *gateways* no setor automotivo baseia-se em três conceitos:

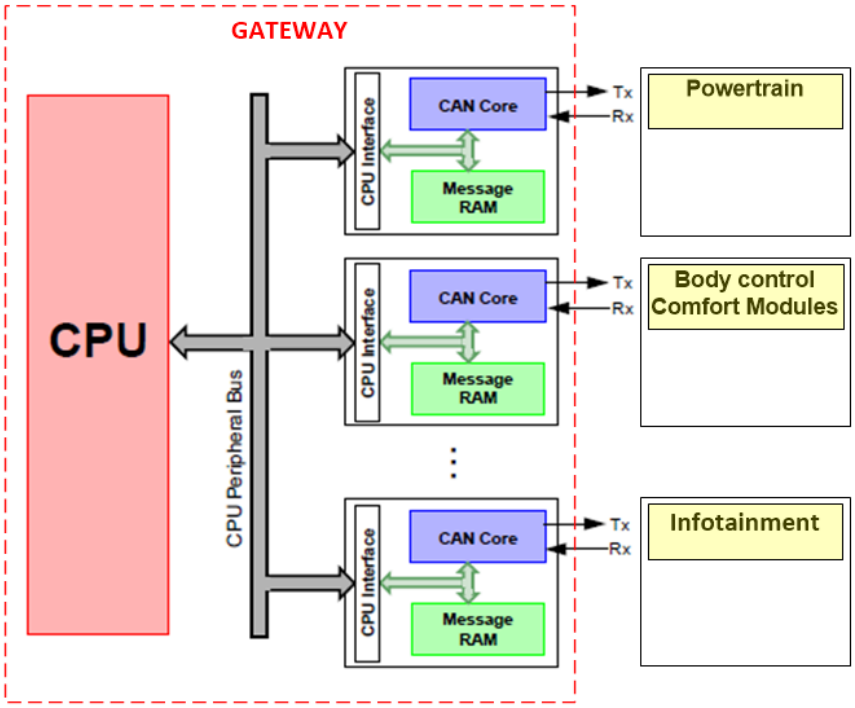
- Desenvolvimento de *gateways* de canal discreto, baseado em um controlador CAN multicanal, o qual armazena as mensagens em uma memória compartilhada gerenciada por uma CPU, este conceito demonstra-se flexível em relação ao número de barramentos CAN que podem ser gerenciados, viabilizando a adição de mais canais CAN de maneira simplificada, mas como requisito, demanda o uso de CPU de alto desempenho, para assegurar operações de tempo real, mesmo em caso de sobrecarga dos barramentos.

- O segundo conceito baseia-se em um controlador *gateway* de barramento CAN complexo, limitando o número de barramentos CAN gerenciados, fazendo o uso de um elaborado mecanismo de controle que processa funções de forma independente, reduzindo o uso da CPU.
- O terceiro conceito envolve a junção das principais vantagens dos dois primeiros conceitos, conhecidos como *gateways* modulares, que serão detalhados mais adiante.

2.7.1 Gateways de Canal Discreto

O conceito mais difundido é o *gateway* de canal discreto, ele consiste dos componentes CPU, barramento periférico da CPU e diversos módulos simples de barramentos CAN como mostrado na Figura 10.

Figura 10 – Estrutura de um módulo gateway de canal discreto



Fonte: Taube, Hartwich e Beikirch (2005).

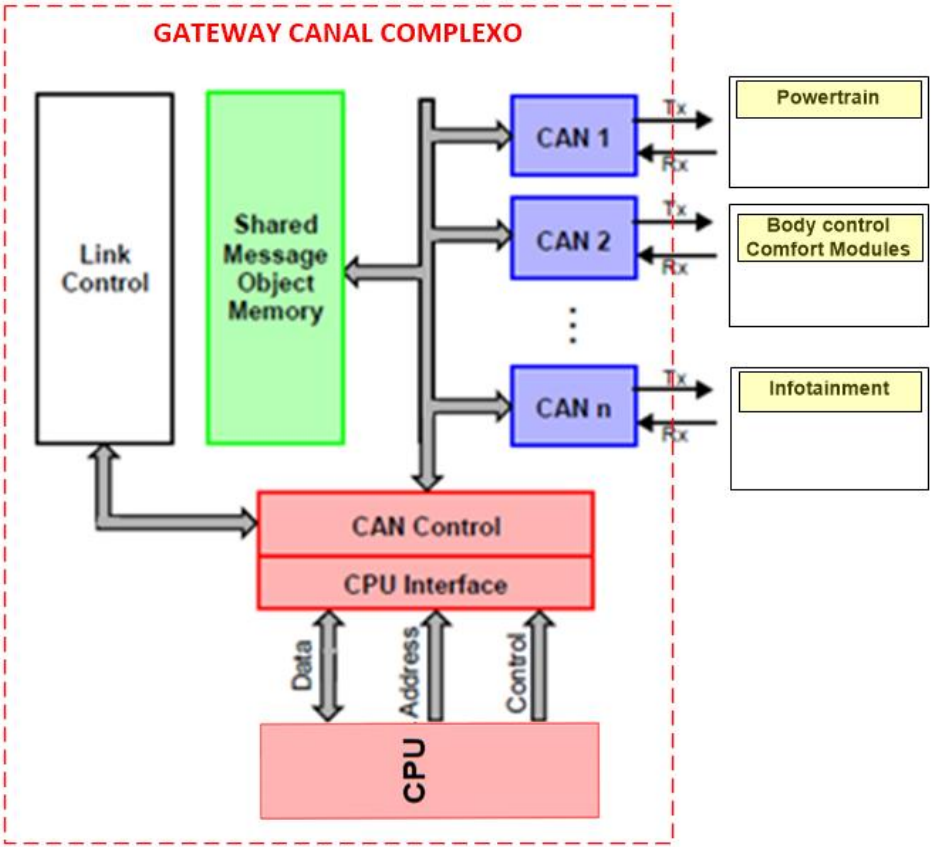
Na maioria das aplicações, o software controla não somente a função *gateway*, mas também algumas outras funções. Cada módulo CAN está conectado diretamente à CPU através do barramento periférico da CPU. Durante a operação do *gateway*, a CPU lê todas as informações de controle enviadas por um dos

módulos CAN no barramento periférico, e retransmite esses mesmos dados para outro módulo CAN presente no barramento. Mensagens concorrentes serão tratadas de forma sequencial, e o número de módulos CAN conectado ao barramento periférico pode ser expandido conforme a necessidade, entretanto isto irá resultar em um aumento no volume de operações da CPU, se mantido a mesma velocidade de operação.

2.7.2 Gateways de Canal Complexo

Este conceito tem como objetivo, a redução do uso da CPU, através da adição de outros elementos na operação: uma memória de uso compartilhado (*Shared Message Object Memory*), módulos CAN (CAN 1, CAN 2,CAN n), uma unidade interna de controle dos núcleos CAN (*CAN Control*), e em alguns casos um controlador de conexão (*Link Control)* atuando como um *gateway* simplificado, como mostrado na Figura 11.

Figura 11 – Estrutura de um módulo gateway de canal complexo



Fonte: Taube, Hartwich e Beikirch (2005).

A unidade de controle CAN gerencia o fluxo de dados entre os módulos CAN, a memória compartilhada e a interface com a CPU, sendo ainda responsável também pelo controle de acesso as diferentes instâncias da memória compartilhada, prevenindo a corrupção ou sobreposição dos dados em memória. As mensagens oriundas dos canais CAN são combinadas em um único bloco da memória compartilhada, reduzindo assim a necessidade de transferência interna de dados e, portanto, reduzindo a carga da CPU. O controlador de conexão, quando presente, atua com regras básicas de roteamento pré-definidas, como por exemplo, uma cópia integral de mensagem, a transferência dos dados da mensagem para um novo identificador, unificação de mensagens, etc. este processo é realizado em conjunto com a unidade de controle CAN, sem o envolvimento da CPU, o que reduz significantemente o seu uso. Como desvantagem deste conceito o número de canais CAN não pode ser facilmente expandido, acarretando em alterações complexas no controlador de conexões e na unidade de controle CAN, especialmente no controle de acesso ao barramento e à memória compartilhada. Normalmente módulos gateway de canal complexo ficam limitados ao número de até 5 canais CAN.

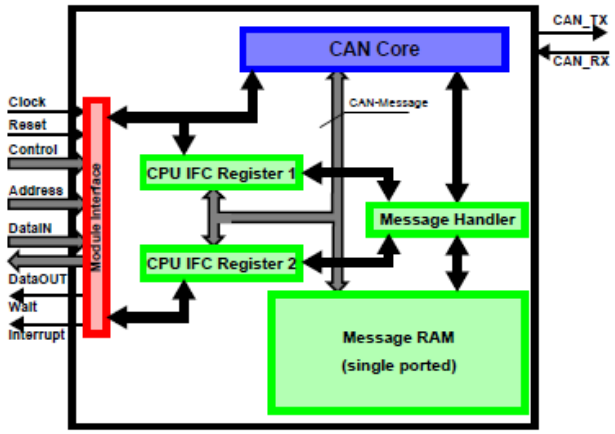
2.7.3 Gateways Modulares

O módulo CAN de canal simples possui uma estrutura básica de operação para a conexão ao barramento CAN e que se conecta a uma interface *gateway*.

Esses módulos CAN simples podem ser agrupados e gerenciados de forma centralizada por uma única unidade *gateway* dedicada.

O módulo CAN de canal simples, apresentado na Figura 12, é composto pelo núcleo CAN (*CAN core*), manipulador de mensagens (*message handler*), um módulo de memória (*message RAM*) e registradores CPU IFC. O núcleo CAN executa a comunicação serial no barramento CAN. As mensagens são armazenadas no message RAM e são gerenciadas pelo message Handler, que também realiza a transferência de mensagens entre o CAN Core e o message RAM. Dois conjuntos de interface de registro de CPU são usados para a transferência de dados entre o barramento periférico da CPU e o message RAM. Eles consistem de dados completos, incluindo o cabeçalho e as informações de controle da mensagem que são movidos como um pacote de dados único para o barramento de dados interno.

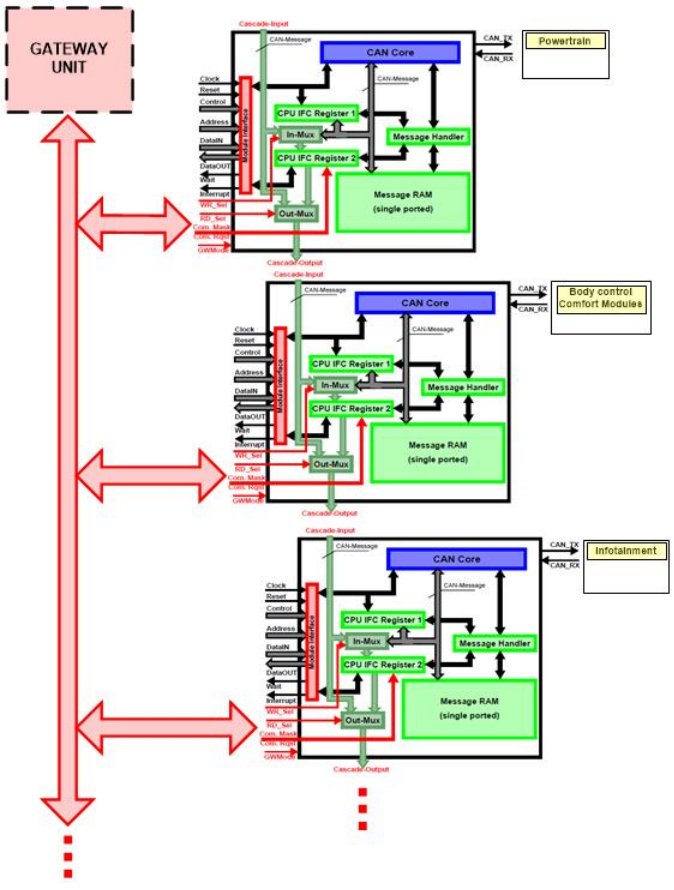
Figura 12 – Módulo CAN sem a interface *Gateway*



Fonte: Taube, Hartwich e Beikirch (2005).

O módulo CAN de canal simples pode ser expandido com diversos blocos funcionais para adaptá-lo em uma célula gateway, esses blocos funcionais são multiplexadores de entrada (*in-Mux*), e multiplexadores de saída (*out-Mux*) juntamente com sinais de controle para direcionar o fluxo de dados, como mostrado na Figura 13.

Figura 13 – Módulos CAN com a unidade *Gateway*

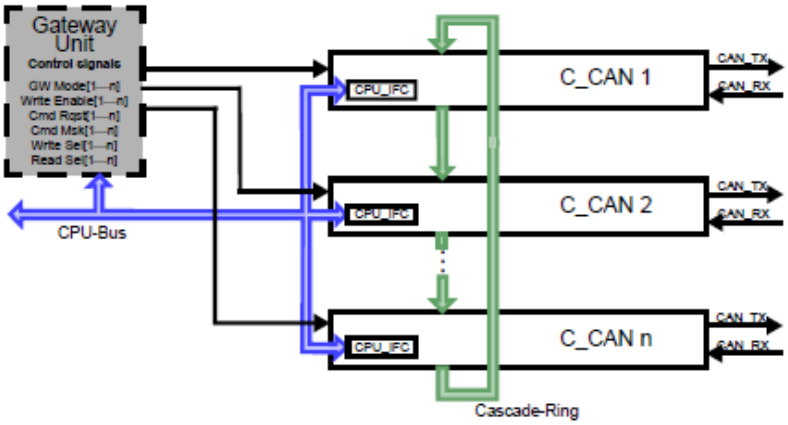


Fonte: Taube, Hartwich e Beikirch (2005).

Os dois multiplexadores permitem o acesso ao barramento interno de dados tornando possível usar os registros CPU IFC para inserir ou resgatar dados, através das portas de entrada e saída *cascade input* e *cascade output*, bem como fazer um roteamento direto entre estas duas portas. Estas portas permitem a transferência de uma mensagem CAN completa e informações de controle necessárias em um único passo, diretamente entre um módulo CAN para outro, evitando-se gargalos no barramento periférico da CPU.

Quando vários módulos são cascateados em um módulo gateway, apresentado na Figura 14, as portas de entrada e saída são conectadas ao *cascade-ring*, permitindo que mensagens e sinais de controle sejam transferidos entre todos os módulos em um único ciclo de relógio. O fluxo de dados entre os módulos é controlado por uma unidade *gateway* baseada em aplicação que controla os sinais para os multiplexadores e a as operações de carga/armazenamento de mensagens nas message RAMs.

Figura 14 – Estrutura de um *Gateway* modular



Fonte: Taube, Hartwich e Beikirch (2005).

2.8 COMPARAÇÃO ENTRE OS CONCEITOS DE GATEWAYS

Fabricantes e desenvolvedores de sistemas irão usar parâmetros diferentes quando comparam as vantagens e desvantagens dos três conceitos. Para fabricantes de semicondutores, estes parâmetros podem ser a quantidade de opções de conexões e sua adaptabilidade com diversos canais CAN, e a possibilidade de conexão com diversas outras interfaces de protocolo, como por exemplo, Flexray, LIN, MOST, etc.. Para desenvolvedores, os parâmetros como

tempo de acesso para leitura e/ou transferência de dados, bem como o desempenho da CPU são elementos mais relevantes e que são levados em consideração. Em geral, um meio termo tem que ser encontrado entre estes parâmetros, especialmente a estrutura do sistema, tamanho do módulo e a necessidade de um desempenho satisfatório da CPU.

Gateways de canal discreto são soluções ideais para modularidade e quando se faz necessárias soluções de tamanho reduzido. Diversas células CAN podem ser conectadas no barramento CAN, somente sendo necessário configurar um identificador a célula, não sendo necessária a inclusão de sinais de controle para o controle de requisições concorrentes no barramento, por que somente a CPU pode acessar as células, isto permite a interconexão de um número ilimitado de canais CAN.

A estrutura simples do sistema, sem adicionais lógicas de interconexão reduz o tamanho do módulo ao máximo, mas em contrapartida a ausência de dispositivos suportando as atividades de *gateway* aumenta o uso da CPU, visto que todas as transferências entre os módulos, a manipulação dos dados e as inserções precisam ser executadas pela CPU. Especialmente os ciclos de leitura e escrita sequenciais para transferências de mensagens entre células.

Mesmo com a desvantagem do alto uso da CPU, este modelo de *gateway* fornece mais flexibilidade in termos de programação uma vez que a função de *gateway* é desempenhada por software, o que atende a maioria das demandas necessárias.

Nas aplicações automotivas, onde o número de células CAN interconectadas e mensagens roteadas causam um alto uso da CPU, o conceito de *gateway* que oferecem recursos adicionais baseados em hardware, como por exemplo, os *gateways* de canal complexo são privilegiados, uma vez que fazem o uso de memória RAM compartilhada, permitindo que as mensagens de todos os canais CAN conectados sejam configuradas e armazenadas no mesmo bloco de memória RAM. Uma mensagem que será recebida por um canal e retransmitida para outro irá ocupar somente um segmento de memória RAM, reduzindo a carga da CPU no gerenciamento desta atividade. Retransmissões automáticas de mensagens recebidas podem ser viabilizadas se *Link Control* estiver presente, a unidade de controle CAN detecta a recepção de uma mensagem e verifica as tabelas de

roteamento no *Link Control*, se alguma regra de roteamento existir, ela será executada diretamente pela unidade de controle CAN.

A redução de carga da CPU acarreta no desenvolvimento de um sistema menos flexível, devido à necessidade do uso de um sistema de controle complexo para a manipulação dos controles e manipulação das mensagens entre CPU, células CAN, RAM e requisições concorrentes arbitrárias tentando fazer o uso da memória RAM compartilhada, requerendo o uso de sinalizadores e semáforos de controle para assegurar a consistências dos dados. Uma reengenharia de toda a estrutura da solução é necessária sempre que uma aplicação específica do *gateway* requer novas funções de transferências, ou ocorre um aumento no número de canais CAN, principalmente nas unidades de controle de link e da CAN.

O *gateway* modular é uma junção das soluções de canal discreto e de canais complexos, combinando as vantagens de ambos os conceitos. A estrutura otimizada permite a transferência rápida entre as várias células sem incorrer em uma carga da CPU. Embora a transferência de mensagens entre uma *message* RAM para outra *message* RAM, através do *cascate ring* consuma um tempo maior do que um *gateway* baseado em canais complexos equipados com uma unidade de *Link control*, a transferência utilizando o *cascate ring* ocorre em menos de dois ciclos de bit na CAN, e permite a transferência simultânea da mensagem para uma ou todas as células conectadas.

A estrutura modular permite também a flexibilidade da programação da função de *gateway*, mesmo quando esta função é controlada por uma máquina de estado finito, a CPU mantêm o acesso total de todas as funções de cada célula CAN, podendo ler, escrever e iniciar a transferência de mensagens. Requisições concorrentes enviadas para a CPU são resolvidas de uma forma determinística, sendo desnecessário o uso de semáforos e adição de sinalizadores de controle, aumentando a flexibilidade do módulo. Pode-se citar como outra vantagem a capacidade de inserir módulos especiais de controle no *cascate ring* sempre que novas funções forem necessárias, pode-se citar aqui a interpolação de mensagens como um exemplo.

Aplicações diferentes demandam diferentes soluções. Uma estrutura modular permite projetar um novo *gateway* combinando componentes de uma biblioteca, acelerando o desenvolvimento significativamente. O tamanho desse módulo é

ligeiramente maior do que a estrutura de um *gateway* de canal, pois são adicionadas uma unidade *gateway* e funções opcionais de manipulação de mensagens.

Uma breve comparação de algum dos principais parâmetros dos modelos de *gateways* está listada na Tabela 4.

Tabela 4 – Visão dos parâmetros fundamentais no desenvolvimento de *gateways*

Parâmetros	canal complexo	canal discreto	modular
Flexibilidade no desenvolvimento	baixo	alto	alto
Tamanho físico	alto	baixo	médio
Expansibilidade	difícil	fácil	fácil
Funcionalidade do hardware	alto	baixo	alto
Necessidade de CPU de alto desempenho	baixo	alto	baixo
Tempo de transferência interna de mensagens	baixo	alto	baixo

Fonte: Taube, Hartwich e Beikirch (2005).

Gateways de canal discreto não são mais empregados em controles de aplicações complexas no setor automotivo, pois informações de tempo real trocadas entre ECUs não podem mais ser garantidas por estes modelos, para esta necessidade os *gateways* de canais complexos possuem um melhor desempenho por usarem menos recursos da CPU na transferência de dados, uma vez que utilizam módulos de manipulação de dados internos, reduzindo dramaticamente a necessidade de desempenho da CPU, porém a sua restrição na expansão dos canais CAN e a impossibilidade de implantar novas funções são uma desvantagem.

Os *gateways* de estrutura modular conseguem atender as necessidades de desempenho da CPU, fornecem flexibilidade de desenvolvimento e expansão, além de possuírem a capacidade de operar requisições de tempo real.

3 DISPOSITIVOS UTILIZADOS NO PROTÓTIPO DO GATEWAY

Para a montagem do protótipo foram utilizados módulos comerciais de prototipagem baseados nas plataformas ARDUINO, todos os componentes foram fixados em uma base de vidro para permitir uma melhor visualização, integração dos componentes, manuseio para manipular as conexões entre os módulos e facilitar as conexões USB com o computador utilizado para carregar os *sketches* dos módulos.

3.1 SKETCH

Um projeto do Arduino é chamado de sketch, e consiste tipicamente em duas partes: a rotina de setup, que inicializa o sketch, e a rotina de *loop*, que normalmente contém o código principal do programa (é como a função “main” da linguagem C).

3.2 CARACTERÍSTICAS DOS PRINCIPAIS MÓDULOS DO PROTÓTIPO

Arduino é uma plataforma de código aberto (em inglês, *open-source*) de software e hardware para prototipagem eletrônica, onde são colocados todos os componentes necessários para que este funcione e se comunique com o computador. Existem diversas versões de placas que são mantidas sob licença *open-source*, e são destinadas a criação de projetos e protótipos.

A plataforma pode receber entradas de uma variedade de sensores e controlar diversos dispositivos atuadores em resposta aos sinais de entrada e de controle.

A linguagem de programação é modelada a partir da linguagem de programação padrão, a qual tem origem em *Wiring* – plataforma de prototipagem eletrônica de hardware livre composta de por uma linguagem de programação, um ambiente de desenvolvimento integrado (IDE, em inglês, *Integrated Development Environment*) e um micro controlador de placa única. Quando pressionado o botão *upload* da IDE, o código escrito é traduzido para a linguagem C e é transmitido para o compilador, que realiza a tradução dos comandos para uma linguagem que pode ser compreendida pelo micro controlador.

O Ciclo de programação do Arduino pode ser dividido da seguinte maneira:

- Conexão da placa a uma porta USB do computador;
- Desenvolvimento de um *sketch* com os comandos para a placa;
- Carregamento do *sketch* para a placa, utilizando a comunicação USB.
- Aguardar a reinicialização, após ocorrerá à execução do *sketch* criado.

A partir do momento que o *sketch* está carregado no Arduino, ele atua como um computador independente, dispensando o uso do computador, conseguindo executar o *sketch* criado de forma autônoma, desde que este esteja ligado a uma fonte de energia.

As placas Arduino fazem uso de placas de expansão (em inglês, s*hields*), que são placas de circuito impresso normalmente fixadas no topo do aparelho através de uma conexão alimentada por pinos-conectores. São expansões que disponibilizam várias funções específicas, desde a manipulação de motores até sistemas de rede sem fio.

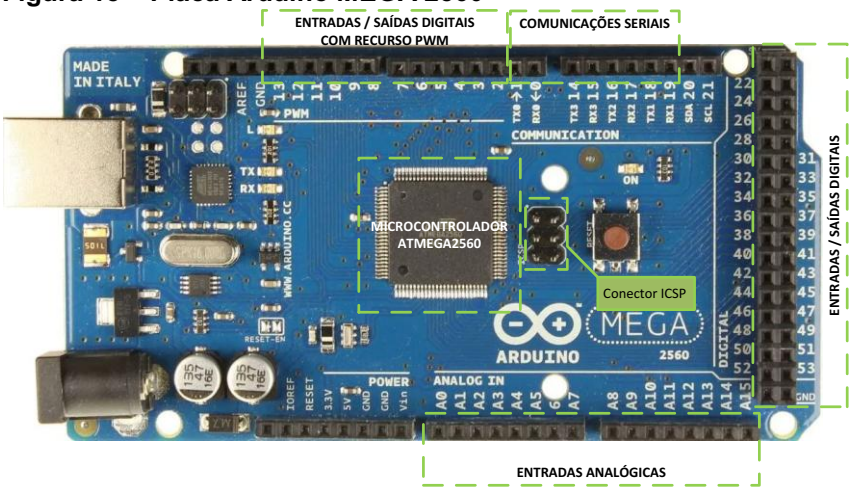
3.3 ARDUINO MEGA

A placa Arduino MEGA 2560 é uma placa da plataforma Arduino utilizada para prototipagem e na elaboração de projetos. Baseado no microcontrolador ATmel ATMEGA 2560, um microcontrolador de 8 bits de arquitetura RISC avançada, que conta com 256 KB de Flash (mais 8 KB são utilizados para o processo de inicialização da placa), 8 KB de RAM e 4 KB de EEPROM. Capacidade 16 MIPS (milhões de instruções por segundo), operando em 16 MHz. A placa ainda possui 54 pinos de entrada e saídas digitais, sendo que 15 destas podem ser configuradas como saídas PWM, conta ainda com suporte a comunicação USB e a programação da placa utiliza um ambiente de desenvolvimento aberto programado em linguagem Java, disponibilizado pela própria fabricante. Também possui 16 entradas analógicas e 4 portas de comunicação serial compatível com SPI e outros protocolos de comunicação como o IIC (*Inter-Integrated Circuit*). A alimentação externa é feita através de conector Jack, onde o valor de tensão da fonte externa pode variar de 6V a 20V. Por ser uma plataforma *open-source*, há várias bibliotecas de programação disponíveis na Internet que facilitam o desenvolvimento utilizando este hardware.

Descrição de alguns pinos que possuem funções especiais e que podem ser visualizados na Figura 15:

- Comunicação Serial - Serial 0 (RX) e 1 (TX); Serial 1: 19 (RX) e 18 (TX); Serial 2: 17 (RX) e 16 (TX); Serial 3: 15 (RX) e14 (TX). Os pinos 0 e 1 estão conectados aos pinos do ATmega16U2 responsável pela comunicação USB.
- Interrupções externas - 2 (interrupção 0), 3 (interrupção 1), 18 (interrupção 5), 19 (interrupção 4), 20 (interrupção 3), e 21 (interrupção 2). Estes pinos podem ser configurados para disparo da interrupção tanto na borda de subida ou descida, ou em níveis lógico alto ou baixo, conforme a necessidade do projeto.
- PWM: os pinos 2 a 13 e 44 a 46 podem ser utilizados como saídas PWM. O sinal PWM possui 8 bits de resolução.
- Comunicação SPI: Pinos: 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS). Estes pinos estão ligados ao conector ICSP.
- Comunicação I2C: (TWI): pinos 20 (SDA) e 21 (SCL).

Figura 15 – Placa Arduino MEGA 2560



Fonte: Autoria própria.

3.4 ARDUINO UNO

A placa Arduino UNO é uma placa da plataforma Arduino utilizada para prototipagem e na elaboração de projetos. O componente principal da placa é um micro controlador ATMEGA328, de 8 bits de arquitetura RISC avançada, que conta com 32 KB de Flash (mais 512 Bytes são utilizados para o processo de inicialização

da placa), 2 KB de RAM e 1 KB de EEPROM. Capacidade 16 MIPS (milhões de instruções por segundo), operando em 16 MHz. Possui suporte a comunicação USB e a programação da placa utiliza um ambiente de desenvolvimento aberto programado em linguagem Java, disponibilizado pela própria fabricante.

Conforme por ser visto na Figura 16 a placa Arduino UNO possui diversos conectores que servem para a integração com elementos externos. A seguir tem-se uma explicação de como cada pino da placa pode ser utilizado.

- 14 pinos de entra e saída digital (pinos 0-13): Esses pinos podem ser utilizados como entradas ou saídas digitais de acordo com a necessidade do projeto e conforme foi definido no *sketch* criado na IDE.
- 6 pinos de entradas analógicas (pinos 0 - 5): Esses pinos são dedicados a receber valores analógicos, por exemplo, a tensão de um sensor. O valor a ser lido deve estar na faixa de 0 a 5 V onde serão convertidos para valores entre 0 e 1023.
- 6 pinos de saídas analógicas (pinos 3, 5, 6, 9, 10 e 11): São pinos digitais que podem ser programados para ser utilizados como saídas analógicas, utilizando modulação PWM.

Figura 16 – Placa Arduino UNO

Fonte: Autoria própria.

3.5 SERIAL PERIPHERAL INTERFACE - SPI

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

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

Para uma comunicação eficiente entre mestre e escravo, é necessário a existência de quatro canais conhecidos por MOSI (em inglês, *Master Output Slave Input*), MISO (em inglês, *Master Input Slave Output*), SS (em inglês, *Slave Select*) e CLK (em inglês, *Clock*) conectados do mestre para o escravo.

A comunicação via SPI ocorre de maneira síncrona, logo depende de um relógio de sincronismo cujo sinal é emitido sempre pelo mestre através de CLK.

Essa frequência pode variar conforme a frequência máxima suportada em cada um dos escravos, que depende do poder de processamento e de leitura de dados de cada um. Como exemplo, pode-se citar um dos elementos utilizados no desenvolvimento desse trabalho, o microprocessador dedicado MCP2515, o qual foi escolhido para a interpretação e envio de mensagens CAN, capaz de funcionar com frequências de até 10MHz (MICROCHIP, 2005).

Para que ocorra a correta transmissão de informações, é necessário que o escravo correspondente seja ativado através de um sinal lógico baixo nos pinos SS.

O SPI permite que vários escravos sejam conectados a um mesmo mestre, desde que durante a transmissão o escravo endereçado seja ativado, o que ocorre normalmente e através do envio de um sinal lógico de nível baixo através do pino de seleção (MARINKOVIC; POPOVICI, 2011).

Quando ativado, o escravo recebe informações enviadas pelo mestre através do pino MOSI byte a byte através de pulsos no pino CLK. A cada pulso, um bit é

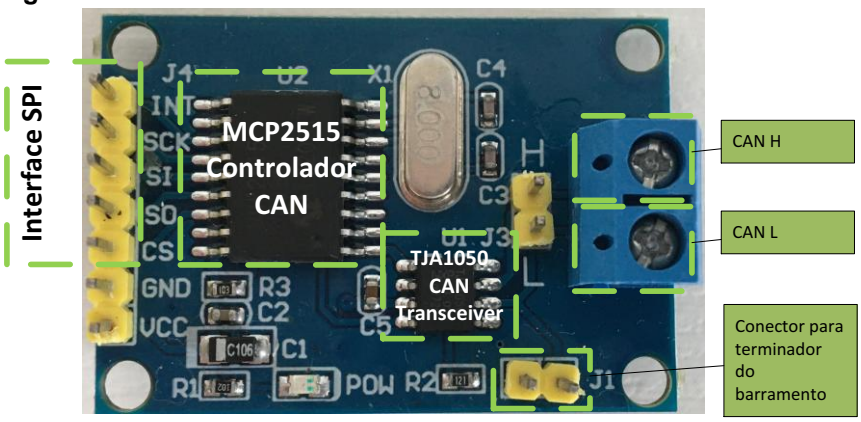
enviado pelo mestre através de MOSI e outro bit é recebido pelo mestre através de MISO, devido ao SPI ser um protocolo de comunicação *full-duplex*, indicando que informações são recebidas e enviadas ao mesmo tempo (FOROUZAN, 2006).

3.6 MÓDULO CAN DE CONTROLE DE BARRAMENTO MCP2515

O módulo MCP2515 é um controlador CAN compatível com as especificações CAN versão 2.0B, capaz de transmitir e receber pacotes em ambos os padrões, o estendido de 29 bits e o padrão de 11 bits, suporta velocidades de comunicação de até 1 Mbps, baseado no circuito integrado MCP2515 controlador CAN e o circuito integrado TJA1050 transceptor CAN (em inglês *transceiver*), que atua como uma interface entre o MCP2515 e o barramento CAN físico.

A Figura 17 mostra os componentes e respectivos pinos de conexão de um módulo MCP2515.

Figura 17 – Módulo CAN MCP2515

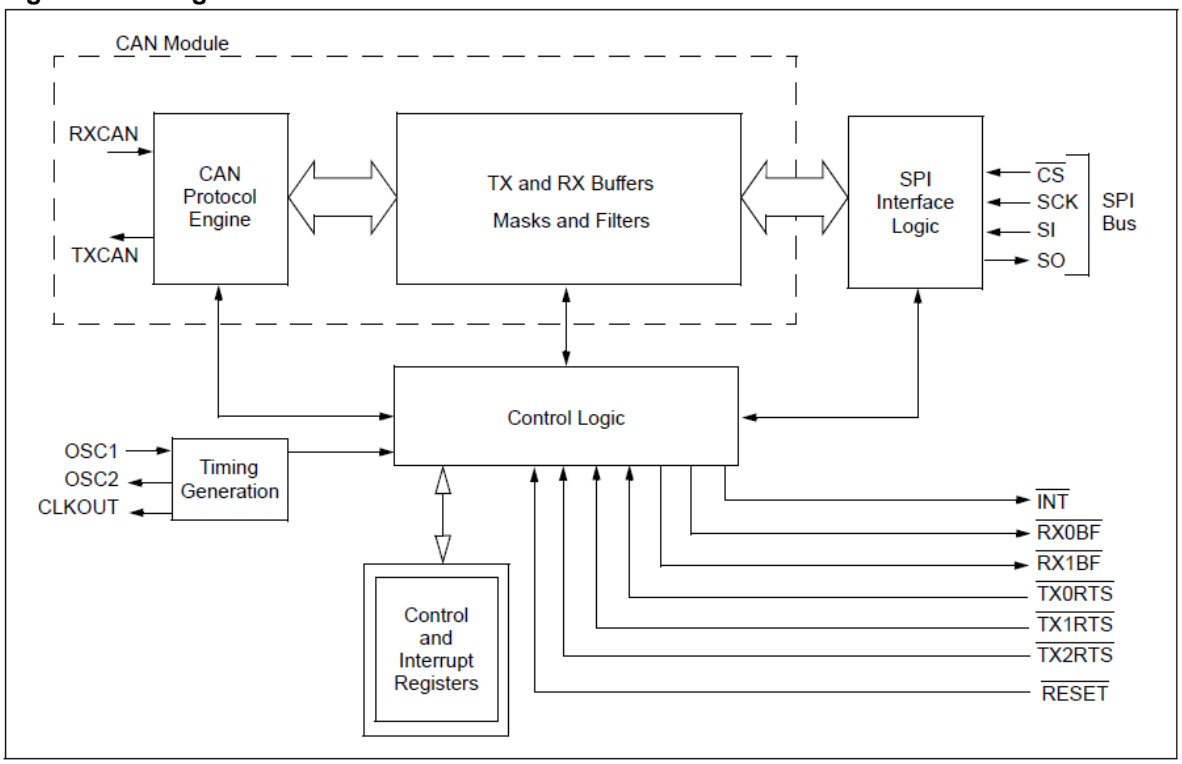


Fonte: Autoria própria.

Desenvolvido para aplicações que requerem a conexão com um barramento CAN, ele consiste de três blocos básicos e representado pelo diagrama de blocos da Figura 18:

1. O modulo CAN, consiste do mecanismo CAN, máscaras, filtros, e os *buffers* de recepção e transmissão.
2. Registradores e o controle lógico responsável pela operação e configuração do dispositivo.
3. Bloco do protocolo SPI.

Figura 18 – Diagrama de blocos do controlador CAN MCP2515



Fonte: Microchip (2005).

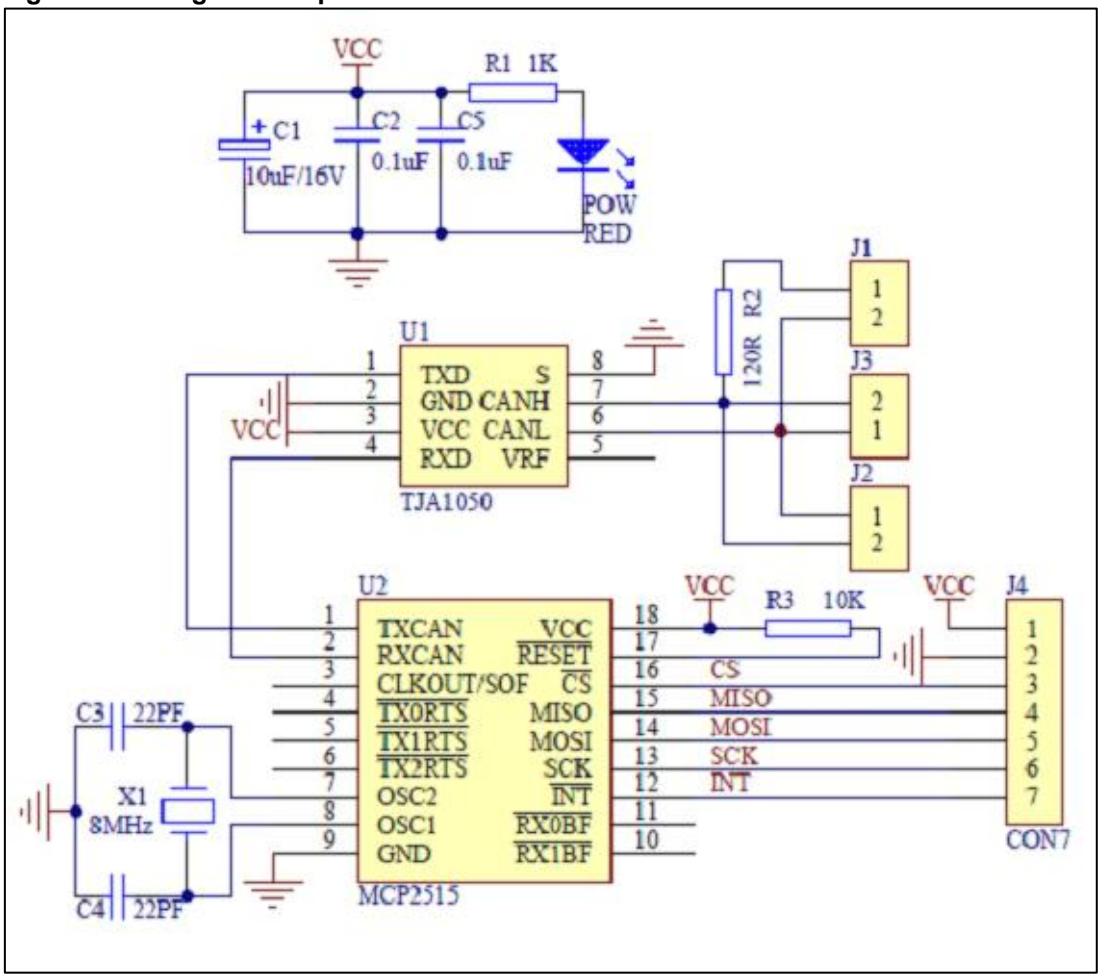
3.7 DIAGRAMA ESQUEMÁTICO DO MÓDULO CAN MCP2515

O Módulo CAN é responsável pela transmissão e recepção de mensagens no barramento CAN. O Controlador Lógico manipula a configuração e a operação do MCP2515 conectando todos os blocos. O bloco SPI é responsável pela interface de comunicação SPI.

Enquanto que o CI TJA1050, uma vez que atua como uma interface entre o controlador CAN MCP2515 e o barramento físico CAN, é responsável por pegar os dados do controlador e retransmiti-los para o barramento.

A Figura 19 mostra o diagrama esquemático do módulo CAN MCP2515 como os CIs são conectados.

Figura 19 – Diagrama esquemático do controlador CAN MCP2515

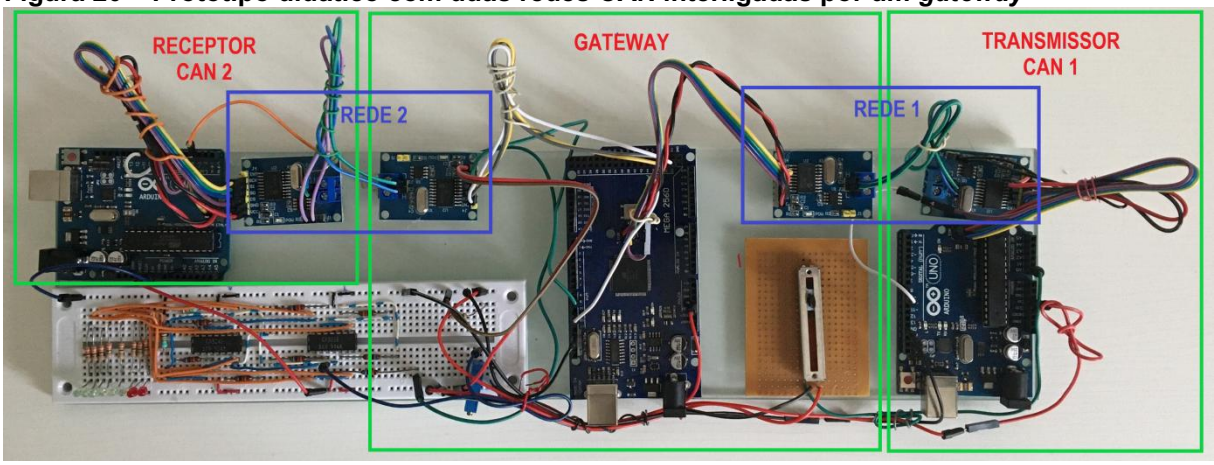


Fonte: Silva (2019).

4 DESCRIÇÃO DO PROTÓTIPO

Para a demonstração prática e fins didáticos no uso do *gateway* CAN, foram utilizados oito elementos importantes: dois dispositivos Arduino UNO, um Arduino MEGA, quatro módulos CAN MCP2515, um potenciômetro deslizante de 20K e um *protoboard* com um circuito detector de variação de voltagem, que utiliza leds para demonstrar visualmente do sinal de saída, conforme demonstrado na Figura 20.

Figura 20 – Protótipo didático com duas redes CAN interligadas por um *gateway*

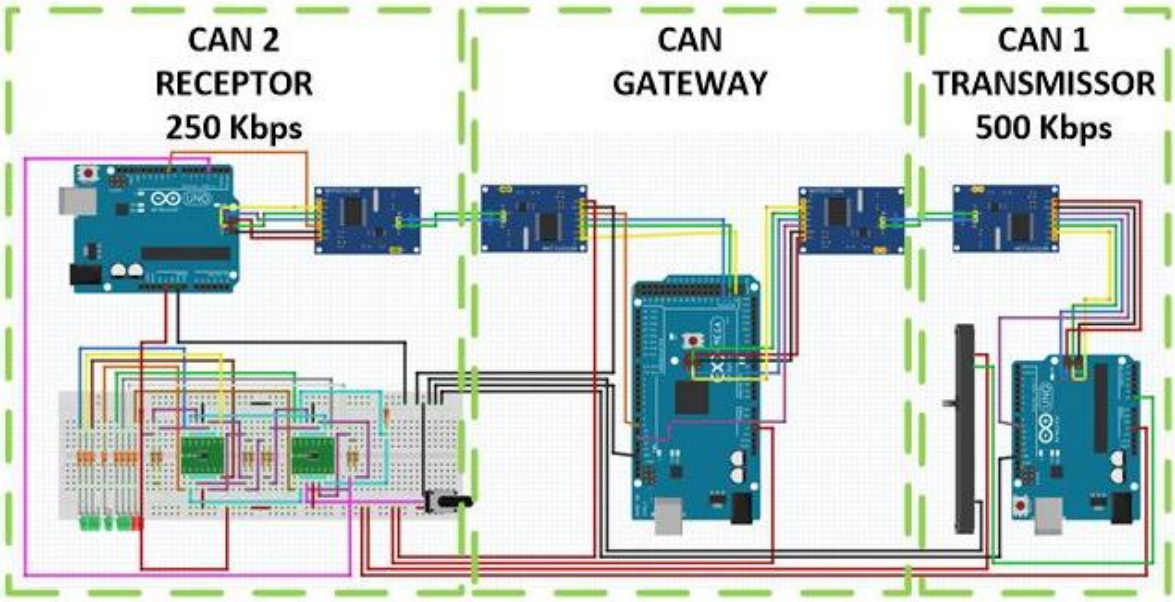


Fonte: Autoria própria.

4.1 DIAGRAMA DE CONEXÕES DOS MÓDULOS

O protótipo foi montado com o objetivo de emular na prática a operação de duas redes CAN conectadas através de um *gateway*. O protótipo pode ser dividido em 3 blocos, como mostra a Figura 21: a) receptor, b) *gateway*, e a) transmissor. As implementações das configurações dos blocos estão apresentadas no “Apêndice A – Sketchs de Configuração do Protótipo”.

Figura 21 – Desenho esquemático do protótipo



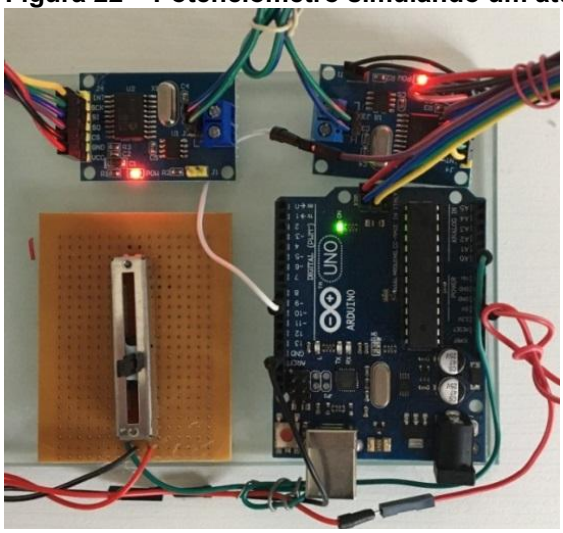
Fonte: Autoria própria.

O protótipo foi configurado para inicializar o bloco transmissor da CAN 1 com uma velocidade de 500 Kbps, enquanto o receptor da rede CAN 2, irá operar com uma velocidade de 250 Kbps.

4.2 CAPTURA DE EVIDÊNCIAS DO PROTÓTIPO

O transmissor, configurado no Apêndice A: “A.1 Sketch de Configuração do Transmissor”, possui um potenciômetro conectado ao Arduino através de uma porta analógica que foi configurada como entrada, fazendo a leitura dos diferentes níveis de tensão gerados na porta a partir da variação da resistência do potenciômetro, conforme Figura 22, o potenciômetro simula um atuador, como por exemplo, um pedal de acelerador. O Arduino realiza esta leitura e transfere a informação para o primeiro módulo MCP2515, representado como CAN 1.

Figura 22 – Potenciômetro simulando um atuador

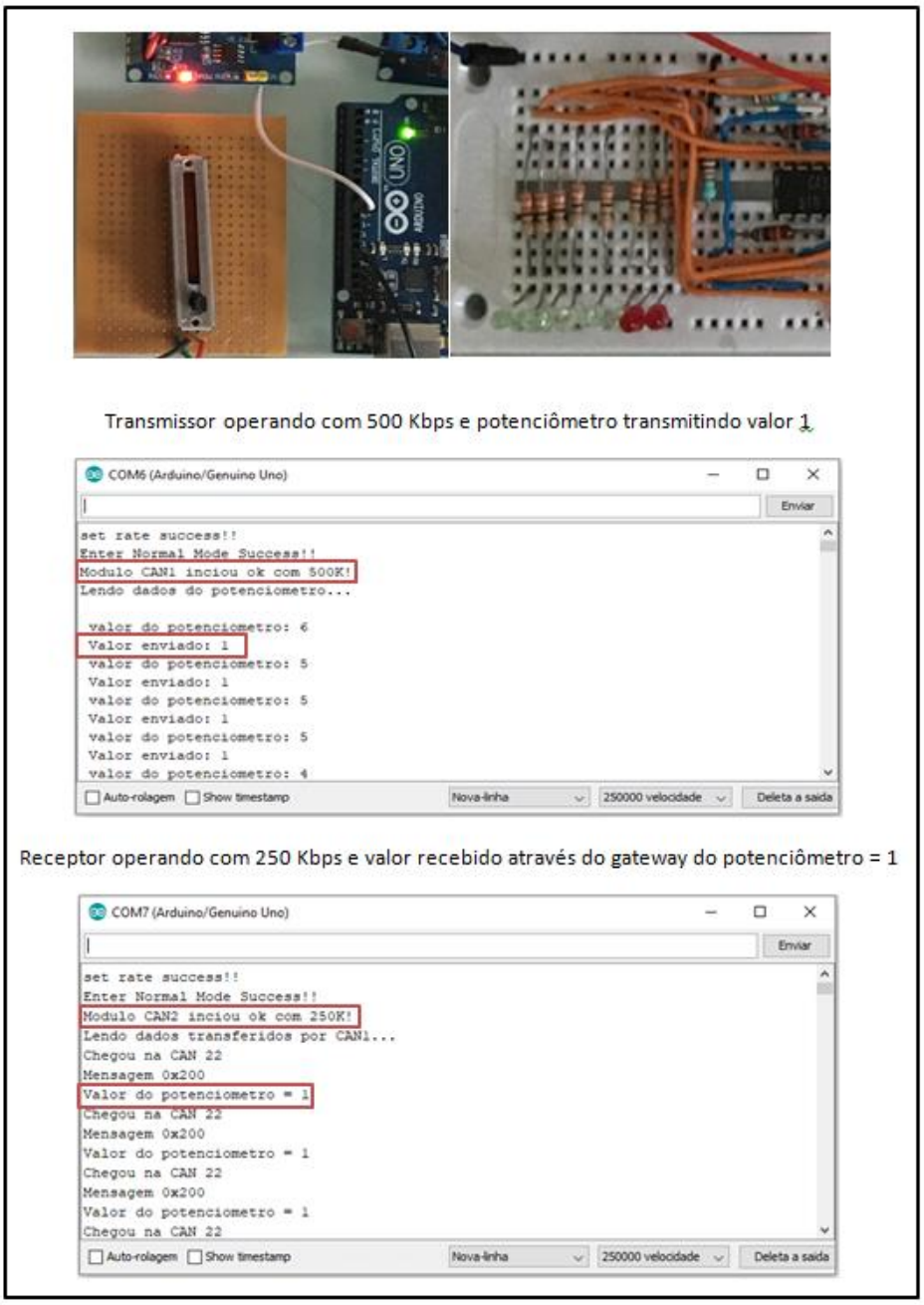


Fonte: Autoria própria.

A CAN1 recebe os dados através da porta SPI e empacota esta informação em *frames* CAN que são transmitidos ao módulo CAN de entrada do *gateway*, o *gateway* recebe estes dados e armazena em registradores internos para a transferência para a o módulo MCP2515 conectado a CAN2, que está operando a uma velocidade de 250 Kbps. O módulo receptor foi configurado no Apêndice A: “A.3 Sketch de Configuração do Receptor”, para receber os dados pela porta SPI e transferir estes dados para uma porta digital configurada como saída PWM que irá emular as variações do sinal de entrada em variações de tensões transmitidas ao *protoboard*. A porta digital foi conectada a um circuito comparador de tensões, que interpreta as variações dessas tensões e realiza o acionamento do conjunto de leds de forma sequencial representando o incremento da tensão de entrada. Conforme descrito a seguir:

Com o potenciômetro em seu valor mínimo, o transmissor envia um valor 1, este valor corresponde ao valor decimal que tem limites entre 0 e 255, correspondentes a capacidade de um byte (8 bits -> 2^8 = 256). Como não existe uma tensão de saída, os leds não sofrem nenhum acionamento, conforme pode ser verificado na Figura 23, que inclui fotos do protótipo e as capturas de tela do monitoramento da serial do transmissor e do receptor no IDE.

Figura 23 – Captura de dados referentes ao estado inicial do potenciômetro

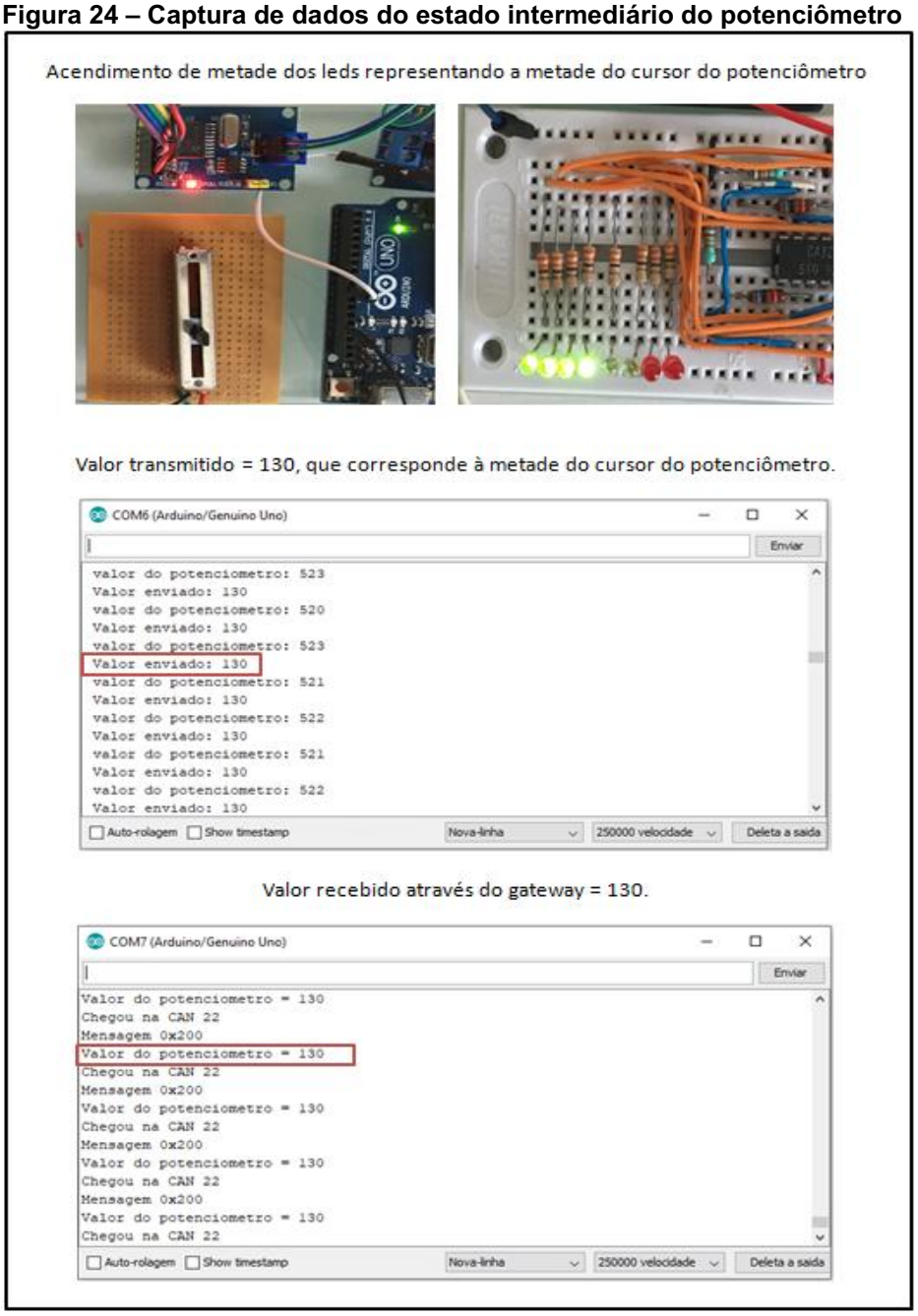


Fonte: Autoria própria.

Movimentando o cursor do potenciômetro até aproximadamente a metade do seu curso, irá causar uma tensão de aproximadamente 2,5 V na entrada analógica do Arduino receptor (porta A0) a qual será convertida em um valor decimal próximo a metade da escala de um byte, neste caso 2^7 = 128. No caso do protótipo pode-se verificar o valor de 130, mas que se encontra dentro da margem, uma vez que a leitura dos valores do potenciômetro tiveram que ser ajustados para serem armazenados em uma variável de números inteiros (8 bits), enquanto que os valores lidos pela porta A0 variam de 0 a 1024, que ocupam 10 bits, e para a transmissão

pela rede CAN eles devem ocupar apenas 8 bits (1 byte), e neste caso lido pela porta analógica teve que ser divido por 4 para a transmissão. Na Figura 24, pode-se verificar as evidências.

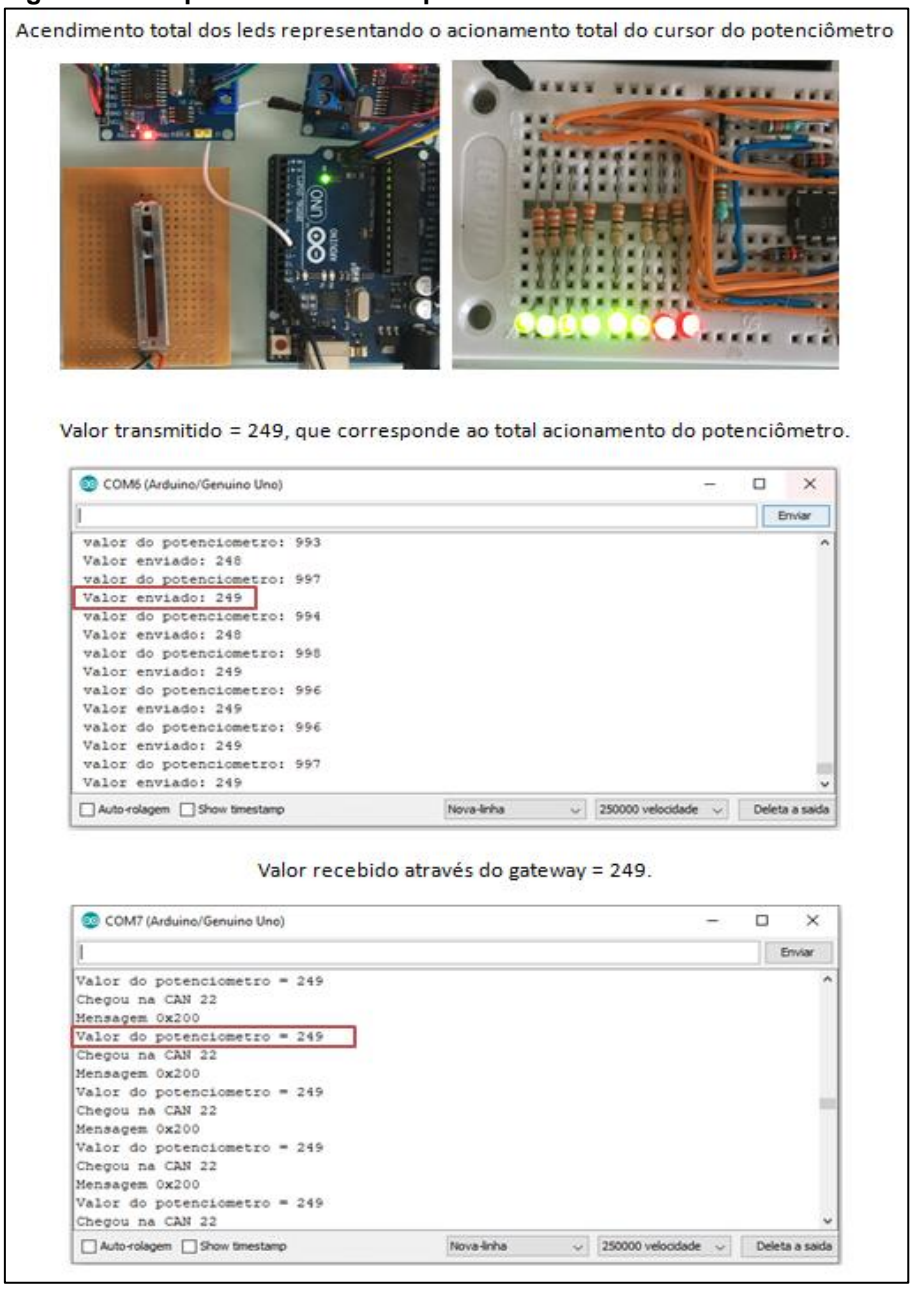
Figura 24 – Captura de dados do estado intermediário do potenciômetro



Fonte: Autoria própria.

Com o cursor do potenciômetro no valor máximo, verifica-se um valor próximo ao limite do byte = 255, neste caso o valor máximo não foi atingido devido a pequenas perdas no potenciômetro, mas está próximo da valor máximo 249, como mostrado na Figura 25.

Figura 25 – Captura de dados do potenciômetro com cursor no máximo



Fonte: Autoria própria.

4.3 VISUALIZANDO A MENSAGEM ATRAVÉS DO OSCILOSCÓPIO

Através do uso do osciloscópio foi possível realizar a captura da mensagem CAN sendo enviada pelo transmissor, Figura 26, e da mensagem CAN sendo recebida pelo receptor conforme Figura 27. Tanto no transmissor como no receptor, o canal CH1 do osciloscópio foi conectado para capturar o sinal presente na CAN H, enquanto que o CH2 captura o sinal do CAN L, e ambos os canais do osciloscópio estão usando a mesma base de tempo por divisão, neste caso 50 µs.

Figura 26 – Osciloscópio: Captura de tela da mensagem CAN no transmissor

Fonte: Autoria própria.

Figura 27 – Osciloscópio: Captura de tela da mensagem CAN no receptor

Fonte: Autoria própria.

Através da análise das telas pode-se comprovar alguns conceitos vistos na seção 2:

- As tensões de operação do protocolo CAN, o osciloscópio foi configurado para 2 V por divisão, e pode-se visualizar que a tensão média no barramento é de 2,5 V que corresponde ao bit recessivo.
- A operação simétrica e oposta dos canais CAN H (em verde) e CAN L (em amarelo).
- O *frame* CAN composto pelos campos de arbitragem, DLC, dados, CRC, ACK e EOF.
- A codificação NRZ (*Non-Return to Zero*) em que o nível do sinal mantém-se constante ao longo do tempo.

Comparando-se as informações presentes em ambas as telas, pode-se verificar que para o *frame* no transmissor, tem-se uma frequência próxima a 5 KHz, que correspondente à 1/100 (um centésimo) da taxa de transmissão configurada no transmissor que foi de 500 Kbps, o que faz sentido uma vez que no *sketch* do transmissor, foi configurado para realizar um *looping* de envio a cada 10 ms.

O mesmo ocorre no receptor, no qual tem-se um *frame* com frequência igual a 2,5 KHz que corresponde a 1/100 de 250Kbps, que foi a configuração de operação nos *sketches* do *gateway* (Apêndice A: “A.2 Sketch de Configuração do Gateway”) e do receptor (Apêndice A: “A.3 Sketch de Configuração do Receptor”).

Nota-se também a integralidade da estrutura do *frame* no receptor, com toda sua estrutura dos dados preservada. Através de uma análise mais aprofundada, poder-se-ia também obter o tempo de cada bit transmitido e determinar o início e fim de cada campo do *frame,* sendo possível determinar o ID da mensagem e o valor transmitido no campo dados, o que seria útil na análise de um problema.

5 CONCLUSÕES

Através do conhecimento da teoria associada ao uso de ferramentas, tais como protótipos, instrumentos de medição e softwares de captura de dados podem-se comprovar a eficiência no uso de redes CAN e do dispositivo *gateway* atuando na função de roteamento e transição de dados entre barramentos operando em velocidades distintas.

Os veículos automotivos estão adotando cada vez mais novas tecnologias embarcadas e se tornando conectados, aumentando o nível de complexidade, desempenho e requisitos de segurança. As ECUs dos futuros veículos autônomos devem trabalhar juntos para detectar, processar e agir na condução do veículo. Isso requer mover e processar uma enorme quantidade de dados com segurança e de forma rápida entre as ECUs. Há uma tendência do uso de redes *Ethernet* gigabit nas redes internas dos veículos e, eventualmente, como espinha dorsal das comunicações entre todos os segmentos do veículo. Esta transição para a Ethernet pode atribuir uma nova funcionalidade ao *gateway* que seria a de controlador de segmentos, podendo ocorrer o uso de mais de um *gateway* operando em diferentes domínios, mas ao mesmo tempo tendo que se integrarem com outros *gateways* de outros domínios.

Não tem como não pensar a respeito da questão de segurança relacionado aos veículos conectados às redes externas, cada vez mais presentes nos automóveis equipados com sistemas de pareamento de dispositivos móveis, a evolução dos *gateways* deve levar em consideração o uso de sistemas de criptografia, certificados digitais e mecanismos de controle de acesso cada vez mais robustos.

REFERÊNCIAS

ANAND, Nitin; *et al*. **Design and implementation of a high speed serial peripheral interface**. Advances in Electrical Engineering (ICAEE), 2014 International Conference on, Vellore, 2014, pp. 1-3. Disponível em: <https://ieeexplore.ieee.org/document/6838431>. Acesso em: 15 mar. 2020.

DI NATALE, Marco; *et al*. **Understanding and using the Controller Area Network Communication Protocol:** Theory and practice. Springer Science & Business Media, LLC 2012.

FARINELLI, Felipe Adalberto; STEVAN JR, Sergio Luiz. **Decodificação e análise dos dados de um Sensor Comercial de Esterçamento de Volante (SAS) com comunicação CAN**. Journal of Applied Instrumentation and Control (JAIC), v. 4, n. 2, p.17-25, dez. 2016. Disponível em: <http://dx.doi.org/10.3895/bjic.v4n2.5229>. Acesso em: 11 mar. 2020.

FARSI, M.; RATCLIFF, K.; BARBOSA, M. **An overview of Controller Area Network**. Computing & Control Engineering Journal**.** Stevenage, v. 10, p. 113-120. jun. 1999.

FOROUZAN, Behrouz A. **Comunicação de dados e redes de computadores.** 3. ed. Porto Alegre: Bookman, 2006.

LAWRENZ, Wolfhard. **CAN System Engineering**: From theory to practical applications. 2. ed. Wolfenbüttel: Springer, 2013. 375 p.

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

MARINKOVIC, Stevan J.; POPOVICI, Emanuel M. **Nano-Power Wireless Wake-Up Receiver with Serial Peripheral Interface**. IEEE Journal on Selected Areas in Communications. v. 29, n. 8, p.1641-1647, set. 2011. Disponível em: <http://dx.doi.org/10.1109/jsac.2011.110913>. Acesso em: 15 abr. 2020.

MICROCHIP. **MCP2515:** Stand-Alone CAN Controller with SPI Interface. Copyright© 2003-2019 Microchip Technology Inc. 2005. Disponível em: <http://ww1.microchip.com/downloads/en/DeviceDoc/MCP2515-Stand-Alone-CAN-Controller-with-SPI-20001801J.pdf>. Acesso em: 06 jun. 2020.

NPX. **Automotive gateway**: A key component to securing the connected car. Copyright© 2018 NXP B.V. Disponível em: <https://www.nxp.com/docs/en/white-paper/AUTOGWDEVWPUS.pdf>. Acesso em: 04 abr. 2020.

RICHARDS, Pat. **A CAN Physical Layer discussion.** Copyright © Microchip Technology Inc 2002. Disponível em: <http://ww1.microchip.com/downloads/en/AppNotes/00228a.pdf>. Acesso em: 01 abr. 2020.

SILVA, Ismael Lopes da. **Barramento CAN entre Arduinos UNO**. Copyright© Embarcados, post publicado em: out. 2019. Disponível em: <https://www.embarcados.com.br/barramento-can-entre-arduinos-uno/>. Acesso em: 15 abr. 2020.

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

TAUBE, Jan; HARTWICH, Florian; BEIKIRCH, Helmut. **Comparison of CAN Gateway Modules for Automotive and Industrial Control Applications**. CAN in Automation, 2005. Disponível em: <https://www.can-cia.org/fileadmin/resources/documents/proceedings/2005_taube.pdf>. Acesso em: 06 abr. 2020.

APÊNDICE A – SKETCHS DE CONFIGURAÇÃO DO PROTÓTIPO

A.1 SKETCH DE CONFIGURAÇÃO DO TRANSMISSOR

```
*/ Carrega as bibliotecas necessárias para a operação do módulo MCP2515*/

#include <mcp_can.h>
#include <mcp_can_dfs.h>
#include <SPI.h>

const int SPI_CS_PIN = 9; // Configura o pino 9 do Arduino para controlar o módulo da rede CAN1
long int potenciometro = 0;
unsigned char valor = 0;

MCP_CAN CAN(SPI_CS_PIN); //Inicializa o modulo no pino definido acima

#define pino_poten A0 // Define a entrada analógica A0 como entrada do potenciômetro

void setup()
        {
        Serial.begin(250000); //inicializa a comunicação serial do IDE como 250 Kbps
        while (CAN_OK != CAN.begin(CAN_500KBPS)) // inicializa a CAN1 com 500K bps
                {
                Serial.println("Modulo CAN1 falhou"); //emite mensagem na serial do IDE caso a inicilização do módulo falhe
                Serial.println("Tente inicializar novamente");
                delay(10);
                 }
        Serial.println("Modulo CAN1 iniciou ok com 500K!"); //confirma que o módulo iniciou com sucesso
        Serial.println("Lendo dados do potenciometro..."); //mostra na serial mensagem inicial de leitura do potenciômetro
        pinMode(pino_poten,INPUT); //define a entrada A0 ligada ao potenciômetro como entrada
        }

void loop()
        {
        potenciometro = analogRead(pino_poten); //realiza a leitura do potenciômetro
        Serial.print("\n valor do potenciometro: "); //imprime na serial o valor lido
        Serial.print(potenciometro);
        valor=potenciometro>>2; //como o potenciômetro lê valores de 0 a 1024, o valor teve que ser ajustado para ir de 0 a 255, desprezando-se os 2 bits menos significantes
        Serial.print("\n Valor: ");
        Serial.print(valor);
        CAN.sendMsgBuf(0x200, CAN_STDID, 1, &valor); // envia os dados: id = 0x200, standard frame, data len = 1, stmp: data  buf

        delay(10); // leitura dos dados 100 vezes por segundo
        }
```

A.2 SKETCH DE CONFIGURAÇÃO DO GATEWAY

```
*/ Carrega as bibliotecas necessárias para a operação do módulo MCP2515*/

#include <mcp_can.h>
#include <mcp_can_dfs.h>
#include <SPI.h>

unsigned char len = 0;//cria a variavel para definir o tamanho do quadro CAN
unsigned char buf[8]; //cria variável para armazenar os dados transmitidos
unsigned long ID = 0; //cria variável para definir o ID do dispositivo que está transmitindo
const int SPI_CS_PIN_CAN1 = 11; // Define o pino 11 que será utilizado para controlar o módulo receptor da CAN 1
const int SPI_CS_PIN_CAN2 = 9; // Define o pino 9 que será utilizado para controlar o módulo transmissor para a CAN 2

MCP_CAN CAN1(SPI_CS_PIN_CAN1); //configura os módulos CAN para utilizar os pinos definidos acima
MCP_CAN CAN2(SPI_CS_PIN_CAN2);

void setup()
{
  Serial.begin(250000); //inicializa a comunicação serial do IDE como 250 Kbps
  while (CAN_OK != CAN1.begin(CAN_500KBPS)) // inicializa a CAN1 para operar com com 500 Kbps
  {
    Serial.println("Modulo CAN 1 falhou"); //emite mensagem na serial do IDE caso a inicilização do módulo falhe
    Serial.println("Tente inicializar o módulo CAN 1 novamente");
    delay(10);
  }
  Serial.println("Modulo CAN 1 iniciou ok com 500K!"); //confirma que o módulo iniciou com sucesso

  while (CAN_OK != CAN2.begin(CAN_250KBPS)) // inicializa a CAN1 para operar com com 250 Kbps
  {
    Serial.println("Modulo CAN 2 falhou"); //inicializa a CAN2 para operar com com 250 Kbps
    Serial.println("Tente inicializar o módulo CAN 2 novamente");
    delay(10);
  }
  Serial.println("Modulo CAN 2 iniciou ok com 250 Kbps!"); //confirma que o módulo CAN 2 iniciou com sucesso
}

void loop()
{
 if(CAN_MSGAVAIL == CAN1.checkReceive()) // Verifica se o dado já está disponível no registro de entrada

    CAN1.readMsgBufID(&ID, &len, buf); //recebe o frame enviado pela CAN1
    /*Este bloco envia mensagens para a serial para efeitos de visualização do conteúdo das varáveis recebidas da CAN1*/
    Serial.print("\n***Valor Recebido da CAN1***");
    Serial.print("\nValor ID da mensagem:");
    Serial.print("\t");
    Serial.print(ID, HEX);
    Serial.print("\nDado recebido do potenciômetro : ");
    Serial.print(buf[0]);
    Serial.print("\n");
```

```
   /*Este bloco envia mensagens para a serial para efeitos de visualização do conteúdo das variáveis enviado para a CAN2*/
    CAN2.sendMsgBuf(ID, 0, 1, &buf[0]); //transmite o frame enviado para a CAN2
    Serial.print("\n***Valor Enviado para a CAN2***");
    Serial.print("\nValor ID da mensagem:");
    Serial.print("\t");
    Serial.print(ID, HEX);
    Serial.print("\nDado enviado do potenciômetro : ");
    Serial.print(buf[0]);
    Serial.print("\n_____");
    Serial.print("\n");
    delay(10);
  }
```

A.3 SKETCH DE CONFIGURAÇÃO DO RECEPTOR

```
*/ Carrega as bibliotecas necessárias para a operação do módulo MCP2515*/

#include <mcp_can.h>
#include <mcp_can_dfs.h>
#include <SPI.h>

const int SPI_CS_PIN = 9; //Define o pino 9 como controlador do MCP_2515
unsigned char len = 0; //Variável para armazenar o tamanho do frame CAN
unsigned char buf[8]; //Variável para armazenar os dados transmitidos
unsigned long ID = 0;  //Variável para armazenar o ID da mensagem de origem

#define pino_pwm 3 //Define o pino 3 como saída PWM

MCP_CAN CAN(SPI_CS_PIN); //configura o módulos CAN para utilizar o pino definido acima

void setup()
{
  Serial.begin(250000); //inicializa a comunicação serial do IDE como 250 Kbps
  /* inicializa um loop para iniciar o módulo da CAN2, para operar na mesma velocidade e testa se inicializou de acordo*/
  while (CAN_OK != CAN.begin(CAN_250KBPS))
  {
    Serial.println("Modulo CAN2 falhou");
    Serial.println("Tente inicializar novamente");
    delay(100);
  }
  Serial.println("Modulo CAN2 iniciou ok com 250K!");
  Serial.println("Lendo dados transferidos por CAN1...");
  pinMode(pino_pwm,OUTPUT);
}
void loop()
{
   if(CAN_MSGAVAIL == CAN.checkReceive()) // Verifica se o dado já está disponível no registro de entrada
   {
     /*Este bloco envia mensagens para a serial para efeitos de visualização do conteúdo das variáveis recebidas pela a CAN2*/
      ID = CAN.getCanId();
      Serial.print("***Chegou na CAN 2***");
      CAN.readMsgBufID(&ID, &len, buf);
        Serial.print("\r\n");
```

```
      Serial.print("Mensagem ID:0X");
      Serial.print(ID, HEX);
      Serial.print('\n');
      Serial.print("Valor do potenciometro = ");
      Serial.print(buf[0]);
      Serial.print('\n');
      Serial.println();
   }
   analogWrite (pino_pwm,buf[0]); //Envia frequência de PWM equivalente ao valor do potenciômetro para gerar tensão de saída proporcional
   delay(10); // aguarda 10ms antes de enviar um novo valor
}
```