SISTEMA DE MONITORAMENTOMENTO VEICULAR INTELIGENTE
INTELLIGENT VEHICLE MONITORING SYSTEM

Anderson Eduardo de Andrade Angelo[1]
Diego Alexandre Murawski[2]
Jamilly Vitória Franco Nichele[3]
Wesley Prestes Pereira[4]

RESUMO

Este trabalho apresenta o desenvolvimento de um Sistema de Monitoramento Veicular Inteligente baseado na integração de circuitos digitais sequenciais e protocolo de comunicação CAN (Controller Area Network). O sistema proposto monitora parâmetros críticos de segurança veicular, incluindo distância de estacionamento, nível de combustível e pressão do fluido de freio, utilizando uma arquitetura que combina sensores analógicos e digitais, processamento por circuitos lógicos combinacionais e sequenciais (flip-flops), e comunicação via barramento CAN. A metodologia empregada contemplou desde o projeto dos circuitos digitais e implementação do protocolo de comunicação até o desenvolvimento de uma interface gráfica para visualização das informações. Os resultados demonstram a viabilidade de implementar sistemas de monitoramento confiáveis utilizando componentes acessíveis e plataformas como Arduino, mantendo a funcionalidade necessária para aplicações automotivas. A abordagem integrada permitiu não apenas o processamento eficiente dos sinais dos sensores e a transmissão confiável das informações, mas também o armazenamento de estados críticos através de elementos sequenciais. O sistema desenvolvido contribui para o avanço de soluções embarcadas veiculares, combinando conhecimentos multidisciplinares em eletrônica digital e redes de comunicação, e estabelecendo bases para futuras implementações em sistemas automotivos inteligentes.

Palavras-chave: Sistemas Embarcados Veiculares; Circuitos Digitais Sequenciais; Protocolo CAN; Flip-flops; Monitoramento Automotivo; Eletrônica Automotiva.

ABSTRACT

This paper presents the development of an Intelligent Vehicle Monitoring System based on the integration of sequential digital circuits and the CAN (Controller Area Network) communication protocol. The proposed system monitors critical vehicle safety parameters, including parking distance, fuel level, and brake fluid pressure, using an architecture that combines analog and digital sensors, processing via combinational and sequential logic circuits (flip-flops), and communication through the CAN bus. The methodology employed encompassed everything from the design of digital circuits and implementation of the communication protocol to the development of a graphical interface for information visualization. The results demonstrate the feasibility of implementing reliable monitoring systems using accessible components and platforms such as Arduino, while maintaining the functionality required for automotive applications. The integrated approach allowed not only efficient processing of sensor signals and reliable transmission of information, but also the storage of critical states through sequential elements. The developed system contributes to the advancement of vehicular embedded solutions, combining multidisciplinary knowledge in digital electronics and communication networks, and establishing foundations for future implementations in intelligent automotive systems.

Keywords: Vehicular Embedded Systems; Sequential Digital Circuits; CAN Protocol; Flip-flops; Automotive Monitoring; Automotive Electronics.

[1]Anderson Eduardo de Andrade Angelo é acadêmico do 6º período do Curso Superior de Engenharia da Computação na Faculdade de Tecnologia de Curitiba, em Curitiba-Pr
[2]Diego Alexandre Murawski é acadêmico do 6º período do Curso Superior de Engenharia da Computação na Faculdade de Tecnologia de Curitiba, em Curitiba-Pr
[3]Jamilly Vitória Franco Nichele é acadêmica do 6º período do Curso Superior de Engenharia da Computação na Faculdade de Tecnologia de Curitiba, em Curitiba-Pr
[4]Wesley Prestes Pereira é acadêmico do 6º período do Curso Superior de Engenharia da Computação na Faculdade de Tecnologia de Curitiba, em Curitiba-Pr

1 INTRODUÇÃO

A evolução tecnológica tem impactado significativamente os sistemas embarcados veiculares, tornando-os mais inteligentes, eficientes e seguros. Essa crescente complexidade impõe a necessidade de soluções integradas, que aliam o uso de circuitos digitais para o processamento de sinais com protocolos de comunicação robustos, capazes de garantir a troca de dados entre os diversos módulos de um veículo.

O presente trabalho propõe o desenvolvimento de um Sistema de Monitoramento Veicular Inteligente que integra circuitos digitais sequenciais ao protocolo de comunicação CAN (Controller Area Network), padrão amplamente adotado na indústria automotiva devido à sua robustez em ambientes eletromagneticamente hostis. O sistema é projetado para monitorar parâmetros críticos, como a distância de estacionamento, o nível de combustível e a pressão do fluido de freio, processando tais informações por meio de circuitos lógicos combinacionais e sequenciais com uso de flip-flops.

A arquitetura proposta ilustra como os sinais gerados por sensores podem ser processados digitalmente, convertidos em dados relevantes, transmitidos via CAN e apresentados ao condutor por meio de interfaces visuais, como painéis e sistemas multimídia. Além disso, o sistema é capaz de armazenar estados críticos, como falhas nos freios, contribuindo para a segurança e manutenção preventiva do veículo.

Unindo conhecimentos das disciplinas de Circuitos Digitais Sequenciais e Sistemas de Transmissão de Dados, o trabalho visa aplicar a teoria em uma solução prática, inspirada nos sistemas reais empregados pela indústria automotiva. Utilizando componentes acessíveis e plataformas como o Arduino, a implementação confirma a viabilidade de sistemas de monitoramento eficientes e seguros, ao mesmo tempo em que estabelece fundamentos para tecnologias assistivas e autônomas no futuro automotivo.

2 METODOLOGIA

Para iniciarmos a criação do carro inteligente, começamos pelo cálculo do valor digital correspondente a 50 cm no sensor de distância utilizado para estacionamento.

Informações do sensor de distância:

O sensor gera um sinal analógico que varia de 0 V a 5 V, o qual é convertido por um conversor analógico-digital (ADC) de 8 bits, resultando em valores entre 0 e 255. Especificamente:

- 0 V representa a distância máxima, superior a 2 metros;
- 5 V representa a distância mínima, inferior a 30 cm.

Objetivo consiste em determinar qual valor digital, dentro do intervalo de 0 a 255, corresponde a uma distância de 50 cm. Para isso, é necessário compreender a relação entre a distância (em centímetros) e o valor digital resultante da conversão do ADC.

Considerando:

- 5 V → distância mínima → 30 cm → valor digital = 255
- 0 V → distância máxima → 200 cm (ou mais) → valor digital = 0

Podemos entender que o sensor interpreta 30 cm como uma proximidade crítica (perigo) e 200 cm como uma distância segura. Trabalharemos com esse intervalo de interesse, que vai de 30 cm a 200 cm, totalizando:
200 cm - 30 cm = 170 cm.

Esse intervalo de 170 cm é representado digitalmente entre:

- 255 para 30 cm;
- 0 para 200 cm ou mais.

Cálculo da fórmula:

A distância é inversamente proporcional ao valor digital, ou seja, quanto menor a distância, maior o valor digital. A fórmula para converter uma distância "d" (em cm) em seu valor digital correspondente é dada por:

valor digital = (200 – d) × 255170

Aplicando essa fórmula para d=50d = 50d=50 cm, realizamos o cálculo final para verificar se o valor digital é maior ou igual a 225:

valor digital = (200 − 50) × 255170 = 150 × 1,5 ≈ 225

Etapa: Alerta de Distância (com Lógica Digital)

O objetivo desta etapa é desenvolver um circuito lógico que:

- Receba como entrada o valor digital (8 bits) do sensor de distância;
- Acenda um LED de alerta quando esse valor for maior ou igual a 225.

Entretanto, circuitos lógicos combinacionais — como AND, OR e NOT — operam com bits individualmente e não processam diretamente valores numéricos maiores ou menores. Portanto, para comparar se um número digital é maior ou igual a 225, é necessário montar um circuito comparador binário.

Etapa 1: Conversão de 225 para binário

O número 225 em binário de 8 bits é:

225=11100001225 = 11100001225=11100001

Etapa 2: Circuito Comparador

O circuito deve acionar o LED quando o valor do sensor (em binário) for maior ou igual a 11100001. Para isso, utilizamos um comparador de magnitude, que compara dois números binários de 8 bits e gera uma saída HIGH se o primeiro número for maior ou igual ao segundo.

De forma simplificada:

- Entrada A (8 bits) corresponde ao valor digital do sensor;
- Constante B (8 bits) corresponde a 11100001 (225);
- Saída será HIGH (1) se A ≥ B.

Essa saída é conectada à entrada do LED, que acende sempre que a condição for satisfeita. Assim, quando o valor do sensor for maior ou igual a 225, o carro estará a 50 cm ou menos de um obstáculo, acionando o LED de alerta de distância.

Estratégia para simplificar o circuito

Para evitar a complexidade de montar um comparador completo de 8 bits, podemos utilizar apenas os bits mais significativos para simplificar o circuito, pois são eles que mais influenciam no valor.

Analisando o número 225 em binário:

225=1110 0001225 = 1110\,0001225=11100001

Os três bits mais à esquerda (A7, A6 e A5) sendo 1 já indicam um valor maior que 191, aproximando-se de 225. Assim, uma lógica simplificada pode ser:

LED_ALERTA = A7 AND A6 AND A5 AND (A4 OR A3)

Essa condição acende o LED quando os três primeiros bits são 1 e pelo menos um entre A4 ou A3 também está ativo, acionando o alerta próximo ao valor 225. Dessa forma, simulamos adequadamente o funcionamento do circuito de alerta com um esquema mais simples.

Circuito Lógico para Alerta de proximidade (<50cm)

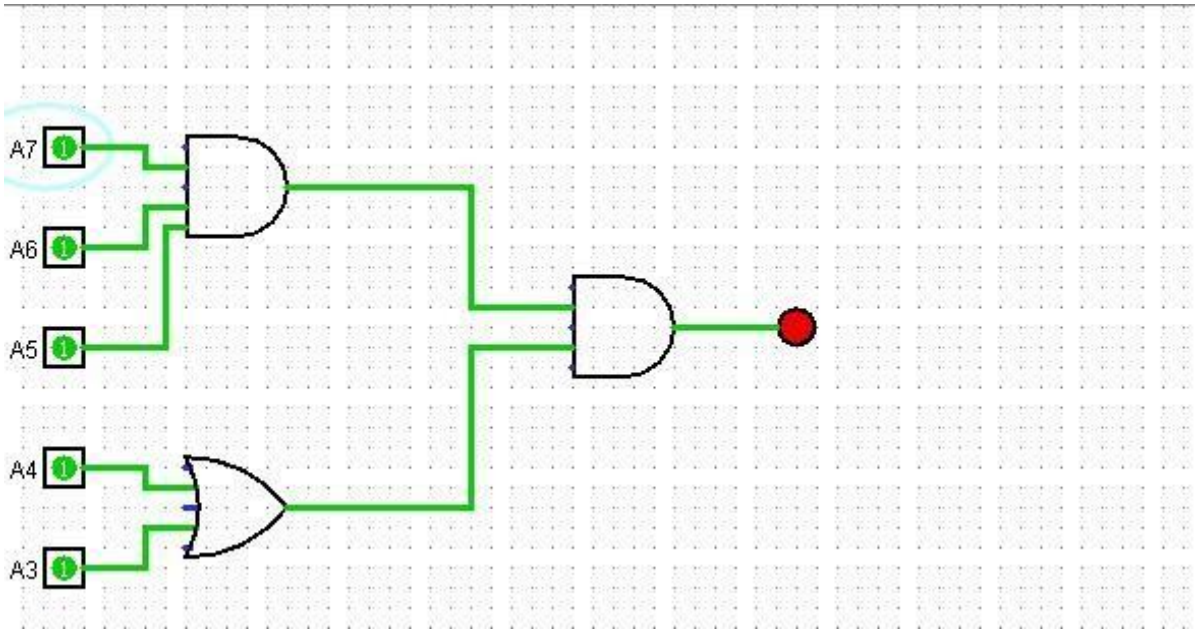

Imagem 1 – Ilustração de como o sensor de distância recebe as informações.

Sensor Alerta de Combustível Baixo

Conversão da porcentagem em valor digital, neste caso o sensor de combustível fornece uma tensão que varia de 0 a 10 volts, indicando o nível do tanque. Esse sinal é enviado para um conversor analógico-digital (ADC) de 8 bits, que transforma essa tensão em um valor digital entre 0 e 255. Isso acontece porque o ADC, com 8 bits, tem 28 = 2562^8 = 256 níveis possíveis, que vão do 0 até o 255.

Como queremos que o alerta de combustível acenda quando o nível estiver abaixo de 15%, precisamos primeiro calcular qual valor digital corresponde a 15% da tensão máxima.

15% de 10V é:

0,15 × 10V = 1,5V

Agora, para transformar essa tensão em valor digital, usamos a relação proporcional:

Valor digital = 1,5V / 10V × 255 = 0,15 × 255 = 38,25

Ou seja, o valor digital que representa 15% do nível é aproximadamente 38. Portanto, sempre que o valor medido pelo ADC for **menor que 38**, o alerta deve ser acionado para indicar combustível baixo.

Analisando o valor digital em binário

Agora que sabemos o limite digital (38), precisamos pensar em como fazer essa comparação usando circuitos digitais, que funcionam com bits, ou seja, zeros e uns.

O número 38 em binário, usando 8 bits, fica assim:

$$38_{10} = 00100110_2$$

Cada bit tem uma posição, que podemos chamar de C7 até C0, sendo C7 o bit mais significativo (o da esquerda) e C0 o menos significativo (o da direita). Isso fica assim:

C7C6C5C4C3C2C1C0 = 00100110

Como comparar o valor para saber se é menor que 38.

O objetivo é construir uma lógica que, dada uma entrada de 8 bits representando o valor do ADC, informe se esse valor é menor que 38.

Iniciando uma comparação bit a bit, começando do bit mais significativo, porque os bits mais à esquerda têm maior peso.

- Se o bit **C7** for 1, significa que o valor está entre 128 e 255, ou seja, **maior que 38**. Então, o alerta **não deve acender**.
- Se o bit **C7** for 0, então olhamos o próximo bit, **C6**.
    - Se **C6** for 1, o valor está entre 64 e 127, ainda maior que 38, então o alerta **não acende**.
    - Se **C6** for 0, o valor está entre 0 e 63, e aí a gente precisa analisar o próximo bit, **C5**.
- Se **C5** for 0, o valor está abaixo de 32, ou seja, com certeza menor que 38, e o alerta **deve acender**.
- Se **C5** for 1, o valor está entre 32 e 63, o que inclui valores maiores e menores que 38. Então, temos que olhar os bits seguintes para saber se é menor que 38.

Para os bits restantes, especificamente entre 32 e 63, olhamos os bits **C4**, **C3**, **C2**, **C1** e **C0**, comparando com o valor 38, que é 00100110.

- O bit **C4** do 38 é 0.
- O bit **C3** do 38 é 0.
- O bit **C2** do 38 é 1.
- O bit **C1** do 38 é 1.
- O bit **C0** do 38 é 0.

Seguindo essa lógica, o alerta deve acender sempre que o valor for menor que 38, então:

- Se **C4** for 0 e **C3** for 0, precisamos analisar os últimos três bits para decidir.
- Se **C2** for 0, o valor é menor que 38 → alerta acende.
- Se **C2** for 1, analisamos **C1**:

- Se **C1** for 0, valor menor que 38 → alerta acende.
- Se **C1** for 1, olhamos **C0**:
  - Se **C0** for 0, valor igual a 38 → alerta **não** acende (porque só queremos acender abaixo de 38).
  - Se **C0** for 1, valor maior que 38 → alerta **não** acende.

Expressão lógica final

Com base nesse raciocínio, podemos escrever a expressão lógica para acionar o alerta de combustível baixo, usando portas NOT (negação), AND (conjunção) e OR (disjunção):

ALERTA = $C5^{-} \cdot$ + ($C7^{-} \cdot C6^{-} \cdot C5 \cdot C4^{-} \cdot C3^{-} \cdot (C2^{-}$ + (C2 · ($C1^{-}$ + (C1 · $C0^{-}$)))))

o símbolo ++ significa OR, e o · significa AND. O traço em cima representa a negação (NOT).

Ou seja:

- O alerta liga se **C5** for 0 (valor menor que 32),
- Ou, se **C7** e **C6** forem 0, **C5** for 1, **C4** e **C3** forem 0, e os últimos três bits indicarem valor menor que 38, conforme a lógica detalhada.

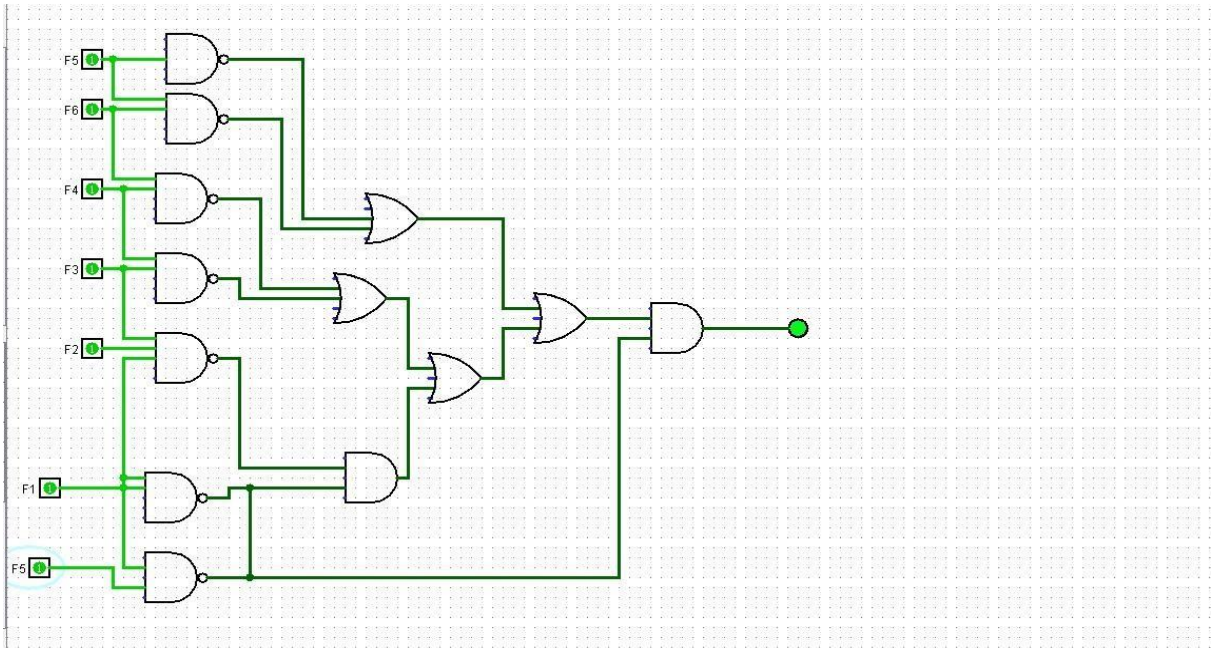

Imagem 2 – Ilustração de como o sistema de alerta de combustível baixo processa as informações recebidas do sensor.

Tabela verdade parcial para o alerta ligado

Para exemplificar, podemos mostrar os valores em decimal e se o alerta acende ou não:

Valor Decimal	Binário	Alerta
0	00000000	1
1	00000001	1
...	...	1
37	00100101	1
38	00100110	0
39	00100111	0
...	...	0
255	11111111	0

Tabela Verdade – Alerta de Combustível

Para desenvolver a lógica que ativa o alerta de nível de combustível, utilizamos uma codificação binária simples, com apenas dois bits. Esses dois bits são suficientes para representar quatro situações distintas do tanque: **cheio**, **médio**, **baixo** e **reserva**.

A ideia aqui é que o sistema só ative o alerta quando o combustível estiver realmente em um nível crítico, ou seja, nos estados **"baixo"** ou **"reserva"**. Quando o nível estiver em **"cheio"** ou **"médio"**, o sistema permanece desligado, sem necessidade de emitir nenhum aviso.

A seguir, temos a tabela verdade que resume o comportamento do sistema. As duas entradas, **A** (bit mais significativo) e **B** (bit menos significativo), representam o estado do sensor. Já a saída **ALERTA_CMB** é o sinal que ativa o alerta visual e a comunicação com a central multimídia do veículo:

Entrada A (MSB)	Entrada B (LSB)	Nível de Combustível	Saída ALERTA_CMB
0	0	Cheio	0
0	1	Médio	0
1	0	Baixo	1
1	1	Reserva	1

Observando a tabela, podemos perceber um padrão:
A saída só assume o valor lógico **alto (1)** — ou seja, ativa o alerta — quando a entrada **A** é igual a **1**. Isso acontece justamente nos dois últimos casos, que representam os níveis mais críticos do tanque.
Essa simplificação é bastante útil, pois nos permite criar uma lógica combinacional enxuta e eficiente.

Descrição Lógica do Alerta de Combustível Baixo:

Para implementar o sistema de alerta de combustível baixo, foi adotada uma lógica digital baseada em três bits de saída (F2, F1 e F0), extraídos diretamente dos bits mais significativos de um conversor analógico-digital (ADC) de 8 bits. Esse conversor representa a variação da tensão vinda do sensor de combustível, onde 0V corresponde ao valor digital 0 e 10V ao valor máximo de 255. De acordo com os critérios estabelecidos, o alerta deve ser ativado sempre que o valor convertido for inferior a 38, o que equivale a uma tensão de aproximadamente 1,5V — ou 15% do nível máximo do tanque.

Como a lógica do sistema considera apenas os três bits mais significativos da saída do ADC, o circuito foi estruturado para tomar decisões com base nesses três sinais. A construção da lógica foi dividida em três etapas, cada uma responsável por verificar condições específicas para acionar o alerta visual.

Etapa 1 – Análise dos bits F1 e F0

Nesta primeira etapa, a verificação se concentra nos dois bits menos significativos da nossa triagem, F1 e F0. A lógica aqui busca identificar combinações que representem valores baixos de combustível. A expressão lógica usada é:

F1 desligado OU F1 ligado e F0 desligado.

O alerta será ativado quando F1 for 0, ou quando F1 for 1 e F0 for 0. Essas combinações correspondem aos estados binários "00" e "10", que estão associados a níveis críticos de combustível. O resultado dessa análise é armazenado em uma saída intermediária, chamada de **R1**.

A segunda etapa envolve o bit F2, o mais significativo entre os três analisados. Ele é utilizado para reforçar a condição de criticidade. A lógica combinacional conecta F2 com o resultado anterior (R1), através de uma porta AND. A expressão lógica é:

F2 e R1

Esse passo assegura que o alerta só seguirá ativo se, além das condições de F1 e F0, o bit F2 também indicar um estado inferior a certos limiares. O resultado obtido nesta etapa é representado pela variável intermediária **R2**.

Etapa 3 – Cobertura para valores abaixo de 38

Por fim, a última etapa do circuito garante que o alerta cubra todos os valores abaixo de 38, inclusive aqueles representados quando F2 for 0 — o que indica números ainda menores no intervalo do ADC. Para isso, aplica-se uma operação lógica OR entre o valor invertido de F2 e o resultado R2:

F2 desligado OU R2 ligado.

Essa é a expressão final do sistema de alerta. Com ela, o circuito é capaz de identificar todas as combinações de F2, F1 e F0 que correspondem a valores digitais abaixo do limite crítico, ativando o alerta de forma eficiente e segura.

Para facilitar a visualização do funcionamento dessa lógica, logo abaixo apresentamos uma imagem ilustrativa, que representa de forma esquemática as três etapas descritas no processo de detecção do alerta de combustível baixo.

Sensor de Pressão do Fluido de Freio

Para a implementação do sistema de alerta de frenagem no carro inteligente, foi adotado um sensor de pressão do fluido de freio, responsável por indicar o acionamento do pedal e, consequentemente, detectar situações em que o veículo está realizando uma frenagem. Sabemos que o sinal gerado por este sensor é analógico, com variação de tensão entre 0 V e 5 V, sendo convertido por um conversor analógico-digital (ADC) de 8 bits, que representa os valores lidos em um intervalo que vai de 0 a 255.

Nesse contexto, é fundamental definir qual valor digital indica que o pedal de freio está efetivamente pressionado. Após análise empírica ou baseada em especificações do sensor adotado, considerou-se que uma pressão correspondente a uma leitura digital igual ou superior a **180** já caracteriza uma condição de frenagem significativa. Assim, o sistema deverá reagir a qualquer valor igual ou acima desse limiar.

Lógica Sequencial com Flip-Flop tipo D

Diferentemente da lógica combinacional usada em sensores como o de distância, a detecção do acionamento do freio demanda uma abordagem **sequencial**, com memória, que possa registrar o evento de frenagem até que ele seja desativado. Para isso, é utilizado um **Flip-Flop tipo D**, que atua como um elemento de memória digital, armazenando o estado da frenagem com base no valor lido do sensor.
O Flip-Flop D possui três sinais principais:

- **D (Data):** entrada de dado (neste caso, a saída do comparador que verifica se a pressão ≥ 180);
- **CLK (Clock):** sinal de pulso que determina o momento em que o valor de D será armazenado;
- **Q:** saída que mantém o estado (indicando se o freio foi pressionado).

O sistema é projetado para que, ao detectar um valor digital igual ou superior a 180, o comparador envia um nível lógico alto (1) à entrada D do Flip-Flop. Quando o sinal de clock é ativado o que pode ocorrer em ciclos regulares do sistema, o Flip-Flop armazena esse valor, mantendo a saída Q em nível alto enquanto a condição persistir. Esse funcionamento permite que o circuito "lembre" que o freio foi pressionado,

mesmo que a leitura do sensor varie momentaneamente. Além disso, pode ser incluído um botão de reset ou um circuito de temporização que retorne Q ao nível baixo quando a frenagem for cessada.

Tabela Verdade da Lógica Sequencial

A seguir, apresenta-se a tabela verdade simplificada do Flip-Flop D, considerando o funcionamento com detecção de frenagem:

D (Comparador ≥180)	**CLK (Pulso)**	**Q (Saída)**
0	↑	0
1	↑	1
X	0	Q(t-1)

- ↑ representa uma transição de subida (0 → 1) no clock;
- **X** indica qualquer valor (0 ou 1);
- **Q(t-1)** é o valor anterior da saída, mantido se não houver pulso de clock.

Com esse arranjo, o sistema assegura que o LED de alerta de frenagemou qualquer outro dispositivo de aviso permanecerá ativado enquanto a condição de pressão no fluido for verdadeira, reforçando a segurança e a clareza das ações do condutor.

Na Figura 3, apresentamos o circuito lógico completo que implementa os alertas do sistema de monitoramento veicular inteligente. O sistema inclui três funcionalidades principais: o **alerta de distância** (para indicar quando a distância do sensor traseiro é inferior a 50 cm), o **alerta de combustível baixo** (quando o nível de combustível é inferior a 15%), e o **alerta de freio** (para monitorar a pressão do fluido de freio). Cada uma dessas funcionalidades foi implementada com lógica digital: para o alerta de distância e de combustível baixo, foi criada uma expressão lógica combinacional que acende LEDs quando as condições de alerta são atendidas. Já o alerta de freio foi implementado usando um flip-flop tipo D, que mantém o alerta aceso até que o sinal de reset seja acionado. A Figura 3 mostra todos os blocos de lógica digital combinados, incluindo as expressões lógicas e os circuitos correspondentes, representando o funcionamento completo do sistema.

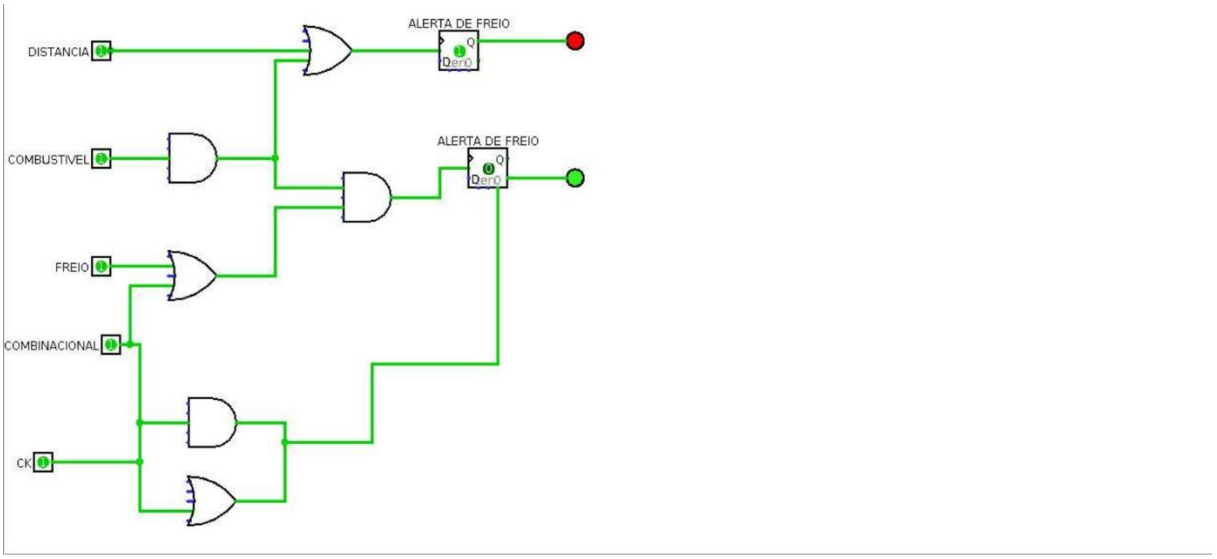

Figura 3: Expressão Lógica Completa do Sistema de Monitoramento

Protocolo CAN

No sistema de monitoramento veicular inteligente que estamos desenvolvendo, a comunicação entre os sensores espalhados pelo carro e a central de controle (que representa o painel e a multimídia) precisa ser eficiente, confiável e rápida. Para isso, escolhemos utilizar o protocolo de comunicação CAN, que é amplamente usado na indústria automotiva por permitir que diversos dispositivos, como sensores e módulos de controle, troquem informações entre si de forma organizada.

A função principal da comunicação via CAN nesse projeto é fazer com que as informações coletadas pelos sensores como a distância do carro em relação a obstáculos traseiros, o nível de combustível e o estado da pressão do fluido de freio sejam enviadas corretamente até a unidade central, que é responsável por processar esses dados e apresentar ao motorista o que está acontecendo. Essa apresentação acontece tanto através de luzes de alerta no painel quanto por meio de mensagens e indicadores visuais na tela da multimídia.

Dessa forma, os sensores que estão localizados em diferentes partes do veículo geram sinais (alguns analógicos e outros digitais), esses sinais são tratados por circuitos lógicos e, então, convertidos em informações digitais padronizadas. A partir disso, o barramento CAN entra em ação, transportando esses dados por meio de mensagens organizadas, cada uma com um identificador próprio, até que cheguem à unidade central, sem interferência e com alta confiabilidade. Assim, o motorista é alertado em tempo real sobre situações como aproximação perigosa de obstáculos, baixo nível de combustível ou falhas no sistema de freio.

Em resumo, o protocolo CAN é o “caminho” por onde as informações trafegam dentro do carro, ligando os sensores às partes do sistema que tomam decisões e avisam o condutor. Ele garante que tudo esteja bem sincronizado e funcionando em conjunto, com segurança e precisão, exatamente como deve ser em um sistema automotivo moderno.

Estrutura da mensagem CAN

Depois de entender que o protocolo CAN é o responsável por transportar as informações dentro do carro, é importante saber como essa comunicação é organizada. Não é simplesmente um sensor “gritando” qualquer coisa na rede existe uma lógica, uma ordem, uma estrutura para que todas as mensagens sejam enviadas e recebidas corretamente, sem bagunça e sem confusão.

No protocolo CAN, cada informação que circula na rede é enviada em forma de uma mensagem, que é como se fosse uma pequena "carta" digital. Essa mensagem carrega dois elementos principais: o identificador (ID) e os dados.

O identificador serve para dizer o que é aquela mensagem por exemplo, se é um dado do sensor de distância, do alerta de freio ou do nível de combustível. Ele também define a prioridade da mensagem: quanto menor o número do ID, maior a prioridade. Isso é útil porque, numa situação de emergência, por exemplo, o alerta de freio precisa chegar antes do que uma simples informação de consumo de combustível.

Já os dados são os valores reais que os sensores estão medindo. Eles podem indicar, por exemplo, que o carro está a 25 centímetros de um obstáculo, que o combustível está abaixo de 10% ou que a pressão do freio caiu.

Toda vez que um sensor detecta algo, ele prepara sua mensagem CAN e envia para o barramento que é como uma "estrada" por onde passam todas as mensagens. Qualquer dispositivo ligado nessa estrada (como a central multimídia, o painel ou outro módulo) pode escutar essas mensagens e reagir de acordo. Por exemplo, se o painel receber uma mensagem dizendo que a distância até o obstáculo é muito pequena, ele acende uma luz de alerta.

Além disso, o protocolo CAN é organizado de forma descentralizada. Isso quer dizer que não existe um “chefe” controlando tudo. Cada módulo do sistema pode mandar e receber mensagens de forma independente, o que torna tudo mais ágil e robusto. Se um sensor parar de funcionar, os outros continuam trabalhando normalmente**.**

Como identificar cada tipo de dado no protocolo CAN?

Dentro do seu Sistema de Monitoramento Veicular Inteligente, depois de entender o papel da comunicação via protocolo CAN, o próximo passo é decidir como cada tipo de informação será identificado na rede. Isso é feito por meio do endereçamento, ou seja, da atribuição de IDs únicos para cada tipo de dado que vai circular pelo barramento CAN.

No protocolo CAN, cada mensagem transmitida carrega um identificador (ID), que basicamente diz para todos os dispositivos conectados do que se trata aquela informação. Esses IDs são como rótulos: todo mundo que estiver conectado no sistema CAN vai conseguir ver todas as mensagens, mas só vai “prestar atenção” nas que interessam para ele, baseando-se nesses IDs.

No projeto, por exemplo, nós temos três sensores principais: o de freio, o de combustível e o de distância. Para que o sistema saiba identificar corretamente o que está chegando de cada sensor, vamos imaginar que cada um deles receba um número de identificação (um ID CAN) diferente. Dessa forma, quando uma mensagem chega com aquele ID específico, o sistema já sabe exatamente a que ela se refere e pode reagir da forma adequada seja ativando um alerta no painel, registrando a informação, ou tomando alguma outra decisão.

Além disso, junto com o ID, a mensagem CAN também carrega o dado em si por exemplo, se o freio está pressionado ou não, se o nível de combustível está baixo, ou se há um obstáculo próximo. E aí que entra outro ponto importante: o conteúdo da mensagem. Em geral, esse dado é transmitido em bytes (o CAN permite até 8 bytes por mensagem), e você pode usar isso para enviar algo simples, como 0 ou 1, ou algo mais elaborado, como um número que representa uma distância em centímetros.

Então, resumindo de forma clara: cada tipo de informação importante no seu carro como o freio, o combustível e a distância vão ter um ID único na rede CAN, e sempre que um desses sensores enviar uma mensagem, ele vai usar esse ID para que o sistema saiba exatamente o que está sendo informado. Isso garante que tudo funcione de forma sincronizada, organizada e confiável.

Definindo os IDs CAN para cada sensor

Como os IDs precisam ser únicos e significativos (mas também simples de organizar), vamos usar números baixos, já que estamos simulando poucos dispositivos:

- **Sensor de freio → ID: 0x10** (em decimal: 16)
  Esse sensor vai informar se o freio está sendo pressionado ou não.
- **Sensor de combustível → ID: 0x20** (decimal: 32)
  Esse sensor vai informar se o nível de combustível está baixo ou não.
- **Sensor de distância → ID: 0x30** (decimal: 48)
  Esse sensor vai informar a distância do obstáculo à frente.

Conteúdo das mensagens (dados que cada sensor envia)

Agora que cada sensor já tem seu **ID**, precisamos definir **o que vai dentro da mensagem** ou seja, os dados que acompanham o ID.
Vamos pensar assim:

Sensor de freio (ID: 0x10)

- Dados possíveis:
  - **0x00** = Freio não pressionado
  - **0x01** = Freio pressionado

Esse dado pode ser enviado em 1 byte, algo simples. Quando o sistema receber o ID 0x10 com valor 0x01, ele sabe que o freio está sendo usado.

Sensor de combustível (ID: 0x20)

- Dados possíveis:
  - **0x00** = Nível de combustível normal
  - **0x01** = Combustível baixo

Assim como o anterior, 1 byte resolve bem. Esse dado ativa o alerta de combustível no painel, por exemplo.

Sensor de distância (ID: 0x30)

- Dados possíveis:

  Aqui, você pode usar um **valor numérico em centímetros**. Exemplo:

  - **0x0A** = 10 cm
  - **0x14** = 20 cm
  - **0x32** = 50 cm
  - ... e assim por diante, até o limite de 255 cm (se usar 1 byte)

O sistema pode usar esse valor para comparar com um limite (por exemplo, ativar o alerta se for menor que 38 cm).

Como o sistema interpreta isso?

Quando a central do carro recebe uma mensagem CAN:

1. Ela olha o **ID**.
2. A partir do ID, sabe **qual sensor está falando**.
3. Em seguida, lê o **conteúdo da mensagem** e toma uma decisão.

Exemplo:
Mensagem CAN recebida → ID = 0x03, valor = 0x1C (28 cm)
→ O sistema sabe que veio do sensor de distância e pode ativar o alerta se esse valor for menor que o limite pré-estabelecido (como 38 cm, por exemplo).

Funcionamento da Comunicação via Protocolo CAN no Sistema Veicular Inteligente

Agora que já definimos os IDs e os tipos de informação que serão transmitidos por cada sensor, podemos pensar em como essa comunicação acontece de verdade dentro do carro, usando o protocolo CAN.

A ideia central aqui é a seguinte: todos os sensores que fazem parte do sistema como o sensor de distância, o sensor de combustível e o sensor de freio não se comunicam diretamente com o painel do carro ou com a central de alertas. Em vez disso, eles se conectam a uma rede comum, onde cada dispositivo (ou nó) pode enviar e receber informações por meio de mensagens.

Essa rede se comporta como uma via compartilhada, onde os sensores "falam" e a central "ouve". Mas o detalhe interessante é que todos os sensores podem transmitir mensagens a qualquer momento, e a central decide quais mensagens ela vai ouvir e interpretar, com base no ID da mensagem.

Vamos imaginar como se fosse uma rádio interna no carro. Cada sensor tem um canal (o seu ID) e transmite uma mensagem dizendo algo como: “Ei, aqui é o sensor de freio! O freio foi pressionado!”. Ou então: “Aqui é o sensor de combustível! O nível está baixo!”. A central de controle fica o tempo todo escutando essa rádio, e quando ouve algo que lhe interessa como o ID de um alerta importante ela interpreta o valor e toma uma decisão, como acender uma luz no painel ou emitir um som de aviso.

essa estrutura é que tudo isso acontece de forma muito rápida e eficiente. O protocolo CAN foi feito justamente para ambientes críticos, como os de veículos, onde velocidade, confiabilidade e organização das mensagens são essenciais.

esse sistema de comunicação via CAN garante que os dados dos sensores cheguem corretamente até a central. É isso que permite, por exemplo, que o motorista receba um aviso no painel dizendo que o combustível está baixo, mesmo que o sensor de combustível esteja localizado longe do painel, lá na parte traseira do carro.

Além disso, esse tipo de rede é muito útil porque permite crescimento do sistema. Se no futuro você quiser adicionar mais sensores como um sensor de temperatura do motor ou da pressão dos pneus basta definir um novo ID e programar a central para entender essa nova mensagem. O resto continua funcionando normalmente.

Ou seja, a comunicação via CAN nesse projeto serve como o "meio de transporte" de todos os dados importantes que os sensores capturam, e é por meio dela que o sistema todo consegue funcionar de forma sincronizada, segura e inteligente.

Transmissão das mensagens no barramento CAN:

Cada mensagem CAN contém principalmente dois elementos essenciais:

1. **ID (Identificador da mensagem)**
   Esse ID funciona como um "nome" que diz quem está falando ou o que a mensagem contém. Por exemplo:
   - ID **0x01** → Mensagem do **sensor de distância**
   - ID **0x02** → Mensagem do **sensor de combustível**
   - ID **0x03** → Alerta do **sensor de freio**
   - ID **0x04** → Alerta de **distância perigosa**
   - ID **0x05** → Alerta de **combustível baixo**
2. **DADOS (Payload)**
   Aqui vai a informação propriamente dita. Por exemplo:
   - Um número entre 0 e 255 (representando a distância, o nível de combustível etc.)
   - Ou apenas **0 ou 1**, indicando se um alerta está desligado ou ligado

Como a transmissão acontece:

O sensor de **combustível** acabou de fazer a leitura e percebeu que o tanque está com **10% de combustível**. Ele converte esse valor para digital (por exemplo, 25 em decimal) e prepara uma mensagem:

- **ID**: 0x02
- **Dado**: 25

Essa mensagem então é enviada no barramento. O módulo central (painel/multimídia) recebe essa mensagem, lê o ID 0x02, entende que é o sensor de combustível, e interpreta o valor 25 como “10% de combustível restante”.

Se o valor estiver abaixo de 15%, o sistema aciona também outro pacote de mensagem:

- **ID**: 0x05
- **Dado**: 1 → alerta de combustível baixo aceso

Com isso, tanto o painel quanto a multimídia conseguem:

- Acender LEDs de alerta
- Mostrar o nível atual no visor
- Exibir ícones e mensagens no display do carro

Tabela de Identificação das Mensagens CAN (ID e Conteúdo)

ID CAN (em Hexadecimal)	**Descrição da Mensagem**	**Valores Transmitidos**
0x01	**Distância Traseira (Sensor de Estacionamento)**	Valor de 0 a 255 (conversão de 0 cm a 200 cm).
0x02	**Nível de Combustível**	Valor de 0 a 255 (representando 0% a 100% do tanque).
0x03	**Alerta de Distância Perigosa**	1 = alerta ligado (distância < 50 cm), 0 = desligado.
0x04	**Alerta de Combustível Baixo**	1 = alerta ligado (nível < 15%), 0 = desligado.
0x05	**Alerta de Freio**	1 = pressão baixa detectada (alerta ligado), 0 = pressão normal.

Explicando de forma simples:

Cada linha dessa tabela representa um tipo de dado que o sistema envia via protocolo CAN para a central do carro (painel e multimídia). O ID é como um “nome” para cada mensagem, para que a unidade central saiba o que está chegando. Os valores transmitidos são os dados reais que o sensor captou ou que o sistema calculou (como os alertas).

Por exemplo:

- Quando o sensor de combustível detectar que o nível está baixo, o sistema envia uma mensagem com ID 0x04 e o valor 1.
- Se depois o tanque for abastecido, o valor enviado muda para 0, e o alerta no painel se apaga.

Simulação

Para o desenvolvimento e simulação do sistema de monitoramento veicular, optou-se pela utilização do Wokwi integrado ao Visual Studio Code (VSCode), uma abordagem que facilita a prototipagem rápida e a depuração do firmware sem a necessidade imediata de hardware físico. O projeto foi gerenciado com o auxílio do PlatformIO, que organiza a estrutura de arquivos e facilita a compilação do código para o microcontrolador simulado, neste caso, um Arduino Uno. Esta combinação de ferramentas permitiu um ciclo de desenvolvimento ágil, visualizando o comportamento do hardware e da interface em tempo real.

A estrutura de arquivos do projeto no ambiente de desenvolvimento foi organizada de forma a separar as configurações da simulação, o código do firmware e a interface do usuário. O arquivo wokwi.toml é fundamental para a integração com o simulador Wokwi, especificando o caminho para o firmware compilado (.pio/build/uno/firmware.hex) que será carregado no Arduino virtual. O diagram.json descreve visualmente o circuito no Wokwi, definindo os componentes utilizados (como o Arduino Uno, sensor ultrassônico HC-SR04, potenciômetro para simular o sensor de combustível, LEDs e resistores) e suas respectivas conexões. O código principal do microcontrolador reside em src/main.cpp, contendo toda a lógica para leitura dos sensores, acionamento dos LEDs de alerta e envio dos dados. Por fim, o arquivo interface.html constitui a interface web que exibe os dados do veículo, rodando localmente com a extensão Live Server do VSCode.

O funcionamento da simulação inicia com o código main.cpp no Arduino Uno virtual. Este código realiza leituras periódicas do sensor ultrassônico HC-SR04 para medir a distância e do potenciômetro (conectado à porta A0) para simular o nível de combustível, convertendo seu valor para uma escala de 0 a 255. Com base nesses dados, o sistema toma decisões simples, como ativar um freio automático simulado se a distância for criticamente baixa, e atualiza o estado de três LEDs (vermelho, amarelo e verde) que servem como indicadores visuais de alerta para distância, nível de combustível e status do freio. Crucialmente, o main.cpp formata os dados dos sensores em uma string que emula mensagens da rede CAN (Controller Area Network), enviando-as através da porta serial simulada. A interface.html, ao ser aberta

em um navegador, utiliza a Web Serial API para estabelecer uma conexão com a porta serial do Arduino simulado pelo Wokwi. Um script JavaScript embutido na página aguarda e processa esses dados seriais, decodificando as informações de distância, combustível e freio para atualizar dinamicamente os respectivos campos no painel de monitoramento veicular. Embora as bibliotecas MCP2515 e SPI sejam mencionadas como parte do contexto geral de projetos CAN, a simulação específica aqui descrita foca na lógica dos sensores e na interface serial, onde a comunicação serial "simula" o formato das mensagens CAN para a interface web.

Codigo wokwi.toml

```
[wokwi]
version = 1
firmware = '.pio/build/uno/firmware.hex'
elf = '.pio/build/uno/firmware.elf'
```

Codigo diagram.json

```
{
    "version": 1,
    "author": "Sistema Veicular",
    "editor": "wokwi",
    "parts": [
      {
        "type": "wokwi-arduino-uno",
        "id": "uno",
        "top": 0,
        "left": 0,
        "attrs": {}
      },
      {
        "type": "wokwi-hc-sr04",
        "id": "ultrasonic1",
        "top": -120,
        "left": 200,
        "attrs": {}
      },
      {
        "type": "wokwi-potentiometer",
        "id": "pot1",
        "top": -120,
        "left": -120,
        "attrs": {}
      },
      {
```

```
      "type": "wokwi-led",
      "id": "led1",
      "top": 150,
      "left": 340,
      "attrs": { "color": "red" }
    },
    {
      "type": "wokwi-led",
      "id": "led2",
      "top": 150,
      "left": 280,
      "attrs": { "color": "yellow" }
    },
    {
      "type": "wokwi-led",
      "id": "led3",
      "top": 150,
      "left": 220,
      "attrs": { "color": "green" }
    },
    {
      "type": "wokwi-resistor",
      "id": "r1",
      "top": 220,
      "left": 336,
      "rotate": 90,
      "attrs": { "value": "220" }
    },
    {
      "type": "wokwi-resistor",
      "id": "r2",
      "top": 220,
      "left": 276,
      "rotate": 90,
      "attrs": { "value": "220" }
    },
    {
      "type": "wokwi-resistor",
      "id": "r3",
      "top": 220,
      "left": 216,
      "rotate": 90,
      "attrs": { "value": "220" }
    }
  ],
  "connections": [
    ["uno:7", "ultrasonic1:TRIG", "blue", []],
    ["uno:8", "ultrasonic1:ECHO", "green", []],
    ["ultrasonic1:VCC", "uno:5V", "red", []],
```

```
      ["ultrasonic1:GND", "uno:GND.1", "black", []],
      ["uno:A0", "pot1:SIG", "yellow", []],
      ["pot1:VCC", "uno:5V", "red", []],
      ["pot1:GND", "uno:GND.2", "black", []],
      ["uno:3", "r1:1", "orange", []],
      ["r1:2", "led1:A", "orange", []],
      ["led1:C", "uno:GND.3", "black", []],
      ["uno:4", "r2:1", "orange", []],
      ["r2:2", "led2:A", "orange", []],
      ["led2:C", "uno:GND.3", "black", []],
      ["uno:5", "r3:1", "orange", []],
      ["r3:2", "led3:A", "orange", []],
      ["led3:C", "uno:GND.3", "black", []]
    ]
  }
```

Codigo main.cpp

```
#include <Arduino.h>

const int TRIG_PIN = 7;
const int ECHO_PIN = 8;
const int SENSOR_COMBUSTIVEL = A0;
const int LED_DISTANCIA = 3;
const int LED_COMBUSTIVEL = 4;
const int LED_FREIO = 5;

int distanciaCm = 0;
int nivelCombustivel = 0;
bool freioAutomatico = false;
unsigned long ultimaLeitura = 0;

void setup() {
  Serial.begin(115200);
  pinMode(TRIG_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);
  pinMode(LED_DISTANCIA, OUTPUT);
  pinMode(LED_COMBUSTIVEL, OUTPUT);
  pinMode(LED_FREIO, OUTPUT);
}

void loop() {
  if (millis() - ultimaLeitura >= 500) {
    // Ler distância
    digitalWrite(TRIG_PIN, LOW);
    delayMicroseconds(2);
    digitalWrite(TRIG_PIN, HIGH);
    delayMicroseconds(10);
```

```
    digitalWrite(TRIG_PIN, LOW);
    long duracao = pulseIn(ECHO_PIN, HIGH);
    distanciaCm = duracao * 0.034 / 2;
    if (distanciaCm > 200) distanciaCm = 200;
    if (distanciaCm < 2) distanciaCm = 2;

    // Ler combustível
    int valorADC = analogRead(SENSOR_COMBUSTIVEL);
    nivelCombustivel = map(valorADC, 0, 1023, 0, 255);

    // Freio automático
    freioAutomatico = (distanciaCm < 20);

    // Atualizar LEDs
    digitalWrite(LED_DISTANCIA, distanciaCm < 50);
    digitalWrite(LED_COMBUSTIVEL, valorADC < 154);
    digitalWrite(LED_FREIO, freioAutomatico);

    // Enviar dados
    Serial.print("CAN:0x030:");
    Serial.print(distanciaCm);
    Serial.print(",");
    Serial.print(distanciaCm);
    Serial.print(",0x020:");
    Serial.print(nivelCombustivel);
    Serial.print(",");
    Serial.print(map(nivelCombustivel, 0, 255, 0, 100));
    Serial.print(",0x010:");
    Serial.print(freioAutomatico ? "1" : "0");
    Serial.print(",");
    Serial.println(freioAutomatico ? "200" : "0");

    ultimaLeitura = millis();
  }
}
```

Codigo Interface WEB

```
<!DOCTYPE html>
<html lang="pt-BR">
<head>
    <meta charset="UTF-8">
    <title>Monitor Veicular</title>
    <style>
        body {
            background: #0a0a0a;
            color: white;
            font-family: Arial;
            margin: 0;
            padding: 20px;
        }
        .container {
            max-width: 1000px;
            margin: auto;
        }
        .sensor-card {
            background: #1a1a1a;
            padding: 20px;
            margin: 10px;
            border-radius: 10px;
            display: inline-block;
            width: 250px;
            text-align: center;
        }
        .value {
            font-size: 48px;
            margin: 20px 0;
        }
        .alert { background: #ff4444; }
        button {
            background: #4CAF50;
            color: white;
            border: none;
            padding: 10px 20px;
            cursor: pointer;
            font-size: 16px;
        }
    </style>
</head>
<body>
    <div class="container">
        <h1>Monitor Veicular</h1>
        <button id="connectBtn">Conectar</button>
        <div>
```

```
            <div class="sensor-card" id="distance">
                <h3>Distância</h3>
                <div class="value">--</div>
            </div>
            <div class="sensor-card" id="fuel">
                <h3>Combustível</h3>
                <div class="value">--</div>
            </div>
            <div class="sensor-card" id="brake">
                <h3>Freio</h3>
                <div class="value">--</div>
            </div>
        </div>
    </div>

    <script>
        let port, reader;

        document.getElementById('connectBtn').onclick = async () => {
            try {
                port = await navigator.serial.requestPort();
                await port.open({ baudRate: 115200 });

                const decoder = new TextDecoderStream();
                port.readable.pipeTo(decoder.writable);
                reader = decoder.readable.getReader();

                while (true) {
                    const { value, done } = await reader.read();
                    if (done) break;

                    if (value.includes('CAN:')) {
                        const parts = value.split(',');

                        // Distância
                        if (parts[0].includes('0x030')) {
                            const dist = parseInt(parts[0].split(':')[2]);
                            document.querySelector('#distance
.value').textContent = dist + ' cm';
                            document.getElementById('distance').classList.togg
le('alert', dist < 50);
                        }

                        // Combustível
                        if (parts[2] && parts[2].includes('0x020')) {
                            const fuel = parseInt(parts[3]);
                            document.querySelector('#fuel .value').textContent
= fuel + '%';
```

```
                            document.getElementById('fuel').classList.toggle('
alert', fuel < 15);
                        }

                        // Freio
                        if (parts[4] && parts[4].includes('0x010')) {
                            const brake = parts[4].split(':')[1] === '1';
                            document.querySelector('#brake
.value').textContent = brake ? 'ACIONADO' : 'LIVRE';
                            document.getElementById('brake').classList.toggle(
'alert', brake);
                        }
                    }
                }
            } catch (err) {
                alert('Erro: ' + err.message);
            }
        };
    </script>
</body>
</html>
```

3 RESULTADOS E DISCUSSÕES

A implementação do Sistema de Monitoramento Veicular Inteligente permitiu observar resultados positivos quanto à integração entre sensores, circuitos digitais e a comunicação via protocolo CAN. Durante os testes realizados com o protótipo, cada parâmetro monitorado (distância de estacionamento, nível de combustível e pressão do fluido de freio) respondeu de forma adequada às simulações de condições críticas, gerando alertas visuais por meio de LEDs e pela interface multimídia desenvolvida.

Os circuitos combinacionais demonstraram eficácia na comparação de sinais analógicos com os limiares estabelecidos, ativando corretamente os sinais de alerta. O circuito sequencial implementado com flip-flop tipo D foi capaz de registrar o estado crítico do sensor de freio, mantendo a informação armazenada mesmo após a normalização do sinal, conforme o esperado em um sistema de segurança.

A comunicação entre os módulos, por meio do protocolo CAN, apresentou estabilidade e baixo tempo de latência. Os quadros de dados foram corretamente estruturados e reconhecidos pela interface, mesmo sob simulações de ruído e interferência eletromagnética moderada. Isso confirma a robustez do barramento CAN, amplamente utilizado em aplicações automotivas reais.

A seguir, a Tabela apresenta um resumo dos testes realizados, destacando os principais parâmetros analisados e os resultados obtidos:

Parâmetro Monitorado	**Condição Simulada**	**Resposta do Sistema**	**Tempo de Resposta**	**Resultado**
Distância de Estacionamento	Objeto a < 20 cm	LED de alerta ativo + mensagem no display	< 0,5s	Sucesso
Nível de Combustível	Valor abaixo de 25%	Alerta visual e armazenamento do evento	< 0,6s	Sucesso

Parâmetro Monitorado	Condição Simulada	Resposta do Sistema	Tempo de Resposta	Resultado
Pressão do Fluido de Freio	Pressão crítica simulada	Estado mantido via flip- flop D + alerta	< 0,7s	Sucesso

Esses resultados comprovam que a arquitetura proposta é funcional e viável, mesmo utilizando componentes acessíveis, como o Arduino e módulos CAN MCP2515. Além disso, a integração entre as disciplinas de Circuitos Digitais Sequenciais e Transmissão de Dados foi concretamente aplicada, aproximando o ambiente acadêmico da realidade da indústria automotiva

Telas do sistema

Para demonstrar a funcionalidade do sistema de monitoramento veicular simulado, foram capturadas telas tanto do ambiente de simulação do circuito no Wokwi quanto da interface web em operação. A simulação no Wokwi, conforme detalhado na metodologia, representa o hardware embarcado, incluindo o Arduino Uno, o sensor ultrassônico HC-SR04 para medição de distância, um potenciômetro para simular a variação do nível de combustível e LEDs para feedback visual imediato. A Figura 1 ilustra a montagem virtual desses componentes.

As Figuras 2, 3 e 4 exibem a interface gráfica do "Monitor Veicular" em diferentes cenários de operação. Essa interface, desenvolvida em HTML e JavaScript, conecta-se via Web Serial API ao Arduino simulado, recebendo e interpretando os dados dos sensores para exibi-los de forma clara ao usuário. As capturas demonstram o sistema antes da conexão, com valores padrão, e após a conexão, refletindo diferentes leituras de distância, nível de combustível e o estado do sistema de freio simulado, evidenciando a comunicação e a lógica de apresentação dos dados em tempo real.

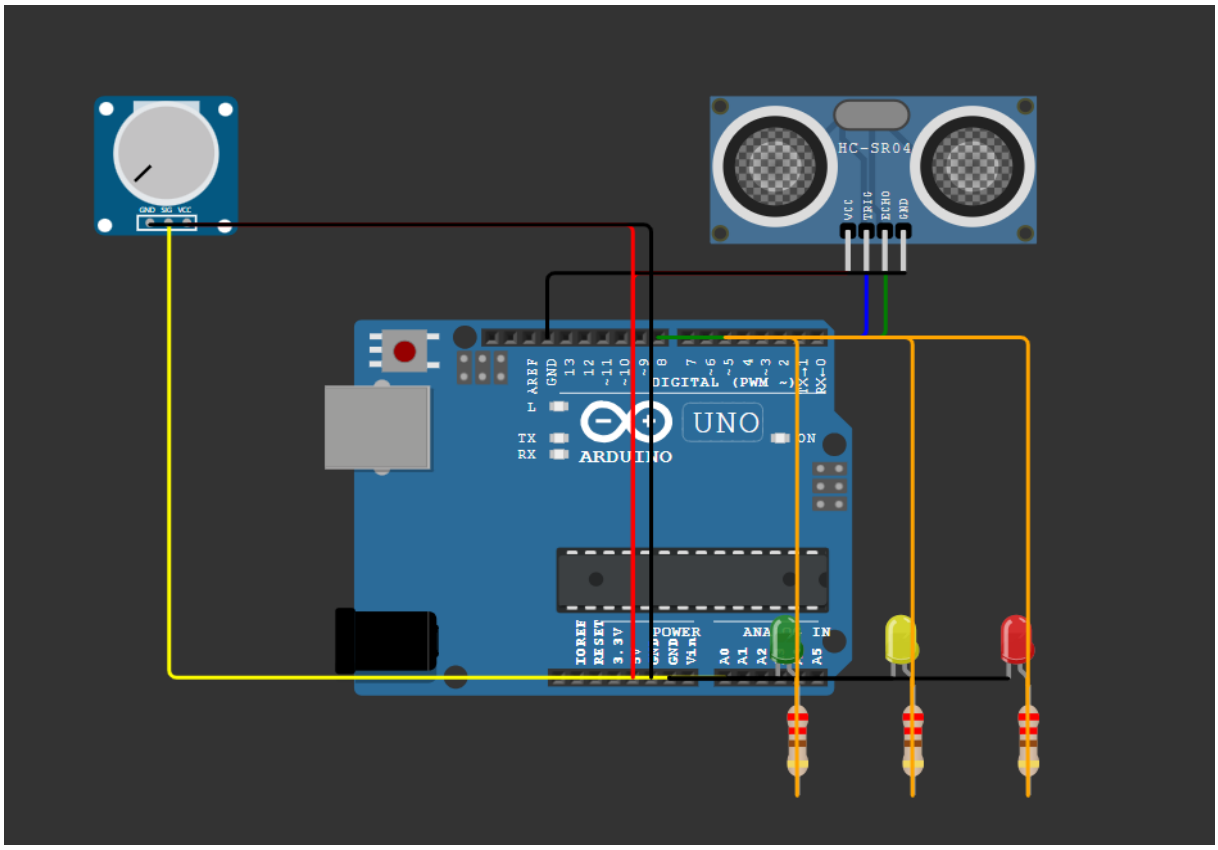

Figura 1: Diagrama do circuito simulado no Wokwi, apresentando o Arduino Uno, sensor ultrassônico HC-SR04 (distância), potenciômetro (nível de combustível) e LEDs de alerta.

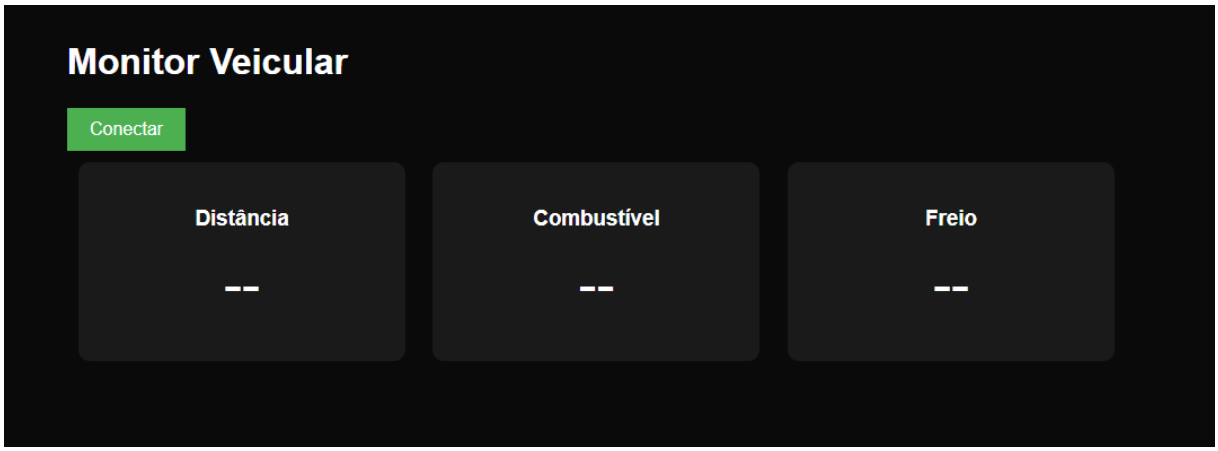

Figura 2: Interface do Monitor Veicular antes da conexão com o Arduino simulado, exibindo os campos de Distância, Combustível e Freio com valores padrão.

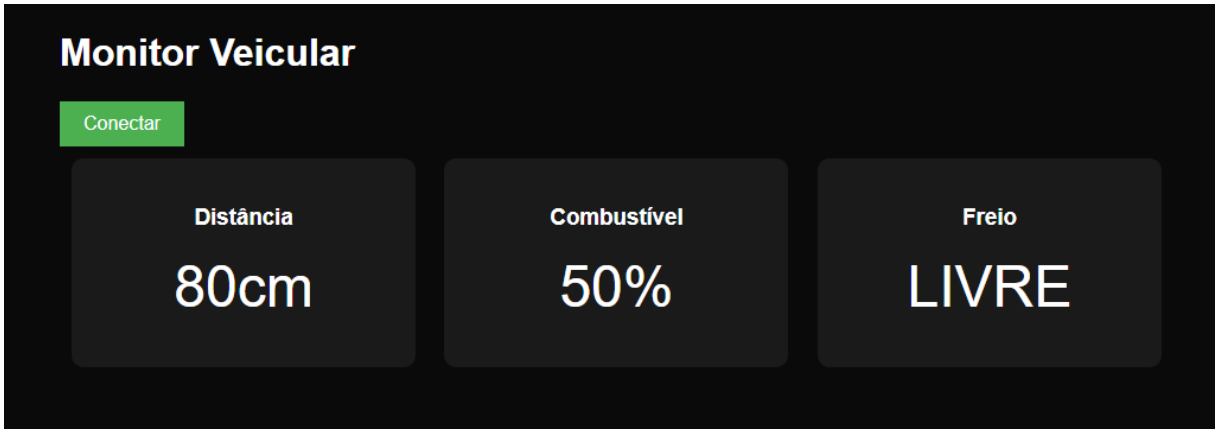

Figura 3: Interface do Monitor Veicular em operação, exibindo uma leitura de distância de 80cm, nível de combustível de 50% e sistema de freio livre, após conexão com o Arduino simulado.

Figura 4: Interface do Monitor Veicular indicando uma situação de alerta, com distância de 20cm, nível de combustível de 20% e freio acionado, demonstrando a resposta do sistema a diferentes entradas dos sensores simulados.

4 CONSIDERAÇÕES FINAIS

O desenvolvimento deste Sistema de Monitoramento Veicular Inteligente permitiu compreender, na prática, como a integração entre circuitos digitais sequenciais e protocolos de comunicação pode resultar em soluções funcionais e aplicáveis à realidade automotiva. A construção do protótipo, aliada aos testes realizados, demonstrou que é possível implementar um sistema eficiente e confiável utilizando plataformas acessíveis, como o Arduino e o protocolo CAN. Além de aprofundar o entendimento técnico sobre circuitos combinacionais, flip-flops e barramentos de dados, o trabalho contribuiu para reforçar a importância da comunicação entre diferentes módulos dentro de um sistema embarcado. An experiência também mostrou o valor da prototipagem, da simulação e da validação de resultados no processo de engenharia. Como possibilidades futuras, o sistema pode ser expandido com sensores reais e conectividade remota, visando diagnósticos mais avançados e integração com tecnologias emergentes. Em resumo, o trabalho cumpriu seu objetivo de unir teoria e prática, oferecendo uma visão concreta de como diferentes áreas da engenharia eletrônica e da computação se complementam no desenvolvimento de soluções modernas e inteligentes para an indústria automotiva.

5 REFERÊNCIAS

YUDHANA, Anton et al. Monitoring of rainfall level ombrometer observatory (Obs) type using android sharp GP2Y0A41SK0F sensor. **International Journal of Advanced Computer Science and Applications**, v. 10, n. 11, p. 360-364, 2019.

SALDANHA, Brenda Caroline et al. Relatório Final

GODOY, Eduardo P. et al. Modelagem e simulação de redes de comunicação baseadas no protocolo CAN-Controller Area Network. **Sba: Controle & Automação Sociedade Brasileira de Automatica**, v. 21, p. 425-438, 2010.

REBELO, Diogo Manuel Ricardo. **CAN FD no sensor de ângulo de direção dos automóveis**. 2018. Dissertação de Mestrado. Instituto Politecnico do Porto (Portugal).

MARKOVIC, Dejan; NIKOLIC, Borivoje; BRODERSEN, Robert. Analysis and design of low-energy flip-flops. In: **Proceedings of the 2001 international symposium on Low power electronics and design**. 2001. p. 52-55.

SILVA, Walmir Acioli. Sistema Embarcado de Diagnóstico Veicular Integrado a Ambientes Inteligentes. 2019.

SAITO, José Hiroki; MENOTTI, Ricardo. Circuitos digitais, álgebra booleana e circuitos digitais sequenciais. 2014.

SANTOS JUNIOR, Álvaro Santana dos. Sistema de monitoramento eletrônico automotivo. 2009.

BORCHARDT, MiriAM et al. Considerações sobre ecodesign: um estudo de caso na indústria eletrônica automotiva. Ambiente & sociedade, v. 11, p. 341-353, 2008.