FELIPE MATHEUS LAURINDO DE OLIVEIRA

Desenvolvimento de Interface de Hardware com Data Logger para OBD-II

Campo Grande, MS, Brasil

Junho de 2023

FELIPE MATHEUS LAURINDO DE OLIVEIRA

Desenvolvimento de Interface de Hardware com Data Logger para OBD-II

Trabalho de Conclusão de Curso (Relatório Técnico) apresentado ao Curso de Engenharia de Computação da Universidade Federal de Mato Grosso do Sul - UFMS, em cumprimento às exigências legais para a obtenção do título de bacharel em Engenharia de Computação, sob a orientação do Prof. Fábio Iaione.

Universidade Federal de Mato Grosso do Sul – UFMS

Faculdade de Computação

Campo Grande, MS, Brasil

Junho de 2023

Resumo

O uso de sistemas eletrônicos está cada vez mais presente nos automóveis, portanto a análise do funcionamento e interação desses sistemas é interessante para a observação do modo de condução do usuário, a verificação de anomalias, ou estimar pontos de melhora em cada sistema. Os dados coletados são fornecidos pela central eletrônica do veículo por meio do conector OBD-II, utilizando-se o protocolo CAN para a comunicação. Portanto, o objetivo deste trabalho é o desenvolvimento de um sistema de baixo custo, capaz de obter dados do veículo periodicamente e armazená-los em um cartão micro SD, possibilitando uma análise da condução e funcionamento do automóvel.

Palavras-chaves: Arduino. data logger. OBD-II.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

ABS	Anti-lock braking system
ACK	Acknowledge
CRC	Cyclic redundancy check
DTC	Diagnostic trouble code
ECU	Electronic control unit
EOF	End of frame
ISO	International Organization for Standardization
PID	Parameter ID
OBD	On-board diagnostics
SAE	Society of Automotive Engineers
SOF	Start of frame

Sumário

Introdução

A modernização dos veículos automotores é um processo que ocorre constantemente, seja pela concorrência do mercado, busca de maior economia no consumo de combustível ou mesmo por maior segurança e confiabilidade. Há algumas décadas teve início uma nova fase que trouxe mudanças significativas na indústria automotiva, motivada pela criação da central de injeção eletrônica (ECU, abreviação em inglês para *electronic control unit*).

Sistemas que eram controlados somente por meios mecânicos e independentes entre si começaram, então, a serem controlados eletronicamente pela ECU. Exemplo clássico é o sistema de admissão, no qual o carburador perdeu totalmente sua função e peças como o corpo de borboleta e os bicos injetores passaram a ser controladas pela ECU.

Para um controle cada vez mais refinado dos sistemas nos automóveis, houve a necessidade natural de inserir mais sensores e atuadores. Logo, sistemas auxiliares surgiram para trabalhar em conjunto com a ECU, como os módulos de freios ABS (*anti-lock braking system*), módulo de *airbag*, painel de instrumentos com computador de bordo, entre outros.

Em 1996 começou a ser utilizada nos Estados Unidos a padronização OBD-II, com o objetivo de fiscalizar a emissão de poluentes. Todos os veículos nos Estados Unidos deveriam possuir um conector padronizado, e serem compatíveis com um conjunto de códigos de diagnóstico que fazem a requisição de dados de sensores à ECU. Esse padrão se expandiu e foi aderido por diversos países, inclusive pelo Brasil em 2010.

Essa padronização facilitou a criação de ferramentas de diagnóstico capazes de analisar os dados dos diversos sensores presentes e detectar possíveis anomalias, ou mesmo monitorar o funcionamento de um sistema específico. Tais ferramentas podem ter um custo elevado (R$500,00 a R$6000,00) e apresentarem limitações, como não permitir uma coleta de dados durante períodos longos, como dias.

Assim, esse trabalho visa desenvolver uma interface de hardware configurável, capaz de realizar a coleta de dados do sensor desejado por meio do conector OBD-II, com intervalo de amostragem ajustável. Com isso, torna-se possível uma análise mais detalhada do sensor, ou mesmo o registro histórico, com custo relativamente mais baixo.

1 Fundamentação teórica

O trabalho desenvolvido envolve alguns conceitos sobre a padronização OBD-II e o protocolo CAN, que serão apresentados a seguir.

1.1 Padrão OBD-II

Uma evolução do OBD-I, o padrão OBD-II especifica detalhes do conector utilizado, como os pinos/sinais e os protocolos de comunicação. Originalmente desenvolvido com o intuito de monitorar a emissão de poluentes, o OBD-II foi adotado pela maioria das montadoras também para monitorar e configurar outros sistemas.

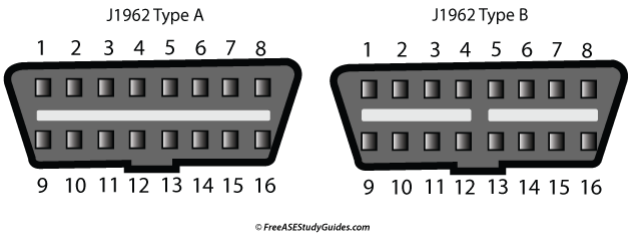
O conector, normalmente posicionado dentro da cabine ao alcance do motorista (Figura 1), é formado por 16 pinos, dispostos em duas fileiras, e possui duas variações: tipo A e tipo B. Sendo o tipo A utilizado em sistemas de 12 V, e o tipo B em sistemas de 24 V (SOCIETY OF AUTOMOTIVE ENGINEERS, 2002a).

Figura 1 – Posição do conector OBD-II tipo A (esquerda) e vista aproximada (direita).

Fonte: Elaborada pelo Autor

A Figura 2 e a Tabela 1 mostram, respectivamente, a posição dos pinos/sinais no conector e a descrição de cada um deles.

Figura 2 – Posição dos pinos nos conectores OBDII padrão A e B.



Fonte: FreeASEStudyGuides.com

Tabela 1 – Pinos/sinais do conector OBD-II.

Número do pino	**Descrição**
1, 3, 8, 9, 11, 12, 13	A critério do fabricante
2	Positivo do PWM e VPW (SAE J1850)
4, 5	Ground/Chassi
6	CAN high (ISO 15765-4 e SAE J2284)
7	K-line (ISO 9141-2 e ISO 14230-4)
10	Negativo do PWM (SAE J1850)
14	CAN low (ISO 15765-4 e SAE J2284)
15	L-line (ISO 9141-2 e ISO 14230-4)
16	Tensão da bateria (+12V ou +24V)

Fonte: SAE J1962

1.2 Barramento CAN

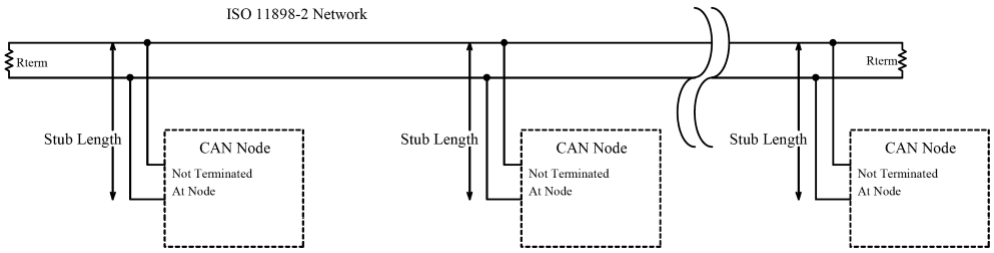
O barramento CAN (*Controller Area Network*), desenvolvido pela empresa alemã Bosch na década de 80, é um dos mais utilizados atualmente pela indústria automotiva devido a sua velocidade de transmissão e alta confiabilidade.

A comunicação no barramento é feita por meio de dois fios: CAN High e CAN Low (pinos 6 e 14, respectivamente). A ECU e outros periféricos são conectados ao barramento ligando seus pinos CAN High e CAN Low diretamente ao fio correspondente, com resistores de 120Ω nas extremidades do barramento (Figura 3). Esses resistores tem a finalidade de terminar a linha com uma impedância igual a impedância característica da linha de transmissão, evitando reflexões que deterioram os sinais transmitidos (WATTERSON, 2017).

O barramento, projetado para operar de 20 kbps a 1 Mbps, é padronizado pela norma ISO 11898-2[1] para altas velocidades (taxas de transmissão até 1 Mbps) e pela norma ISO 11898-3[2] para baixas velocidades (taxas de transmissão até 125 kbps).

[1] ISO 11898-2: Road vehicles - Controller area network (CAN) - Part 2: High-speed medium access unit
[2] ISO 11898-3: Road vehicles - Controller area network (CAN) - Part 3: Low-speed medium interface

Figura 3 – Rede CAN (ISO 11898-2).

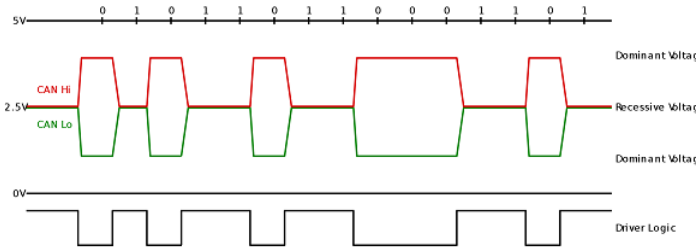


Fonte: TekEye.uk

Essas normas definem o valor das tensões nos fios CAN High e CAN Low para que representem o bit 0 (chamado de dominante) e o bit 1 (chamado de recessivo).

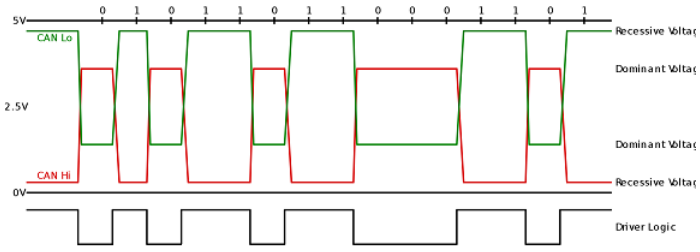
Na ISO 11898-2, o bit 1 é representado por CAN HIGH e CAN LOW com tensão de 2,5V (Figura 4). Já na ISO 11898-3, o bit 1 é representado por CAN HIGH com tensão próxima de 0V, e CAN LOW com tensão próxima de 5V (Figura 5). Em ambas o bit 0 é representado por CAN HIGH com tensão de 3,5V e CAN LOW com tensão de 1,5V (ISO INTERNATIONAL STANDARD, 2006).

Figura 4 – Sinal para altas velocidades (ISO 11898-2).



Fonte: gcanbox.com

Figura 5 – Sinal para baixas velocidades (ISO 11898-3).



Fonte: gcanbox.com

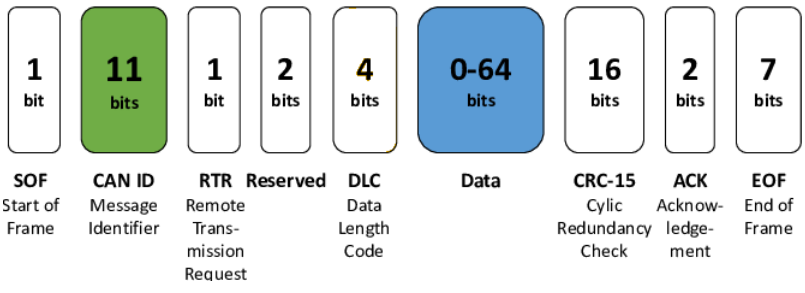
O barramento CAN não inclui um *clock* na transmissão de dados. Todos os nós da rede devem operar na mesma taxa de bits e o erro entre os *clocks* internos de cada nó deve estar dentro da tolerância, para que os nós da rede se comuniquem (BOSCH, 2018).

1.3 *Frame* de Dados

A transmissão dos dados no barramento é feita bit a bit, e as mensagens são transmitidas em formato de frame (Figura 6). Por definição, um frame CAN tem os seguintes componentes:

- SOF (*Start of Frame*) - Marca o início do frame;
- Campo de arbitragem – Inclui o ID da mensagem e o bit RTR (*Remote Transmission Request*), que distingue o *frame* de dados do *frame* remoto;
- Campo de controle – Usado para determinar o tamanho dos dados e o comprimento do ID da mensagem;
- Campo de dados – Contém os dados da mensagem;
- Campo CRC (*Cyclic Redundancy Check*) - Soma de Verificação;
- ACK (*Acknowledge*) - Cada nó que recebe uma mensagem substitui o bit mais alto, recessivo na mensagem original, por um bit dominante, indicando que uma mensagem sem erros foi enviada. O bit mais baixo deve sempre ser recessivo;
- EOF (*End of Frame*) – Marca o fim do *frame*.

Figura 6 – Estrutura de um *frame* no barramento CAN

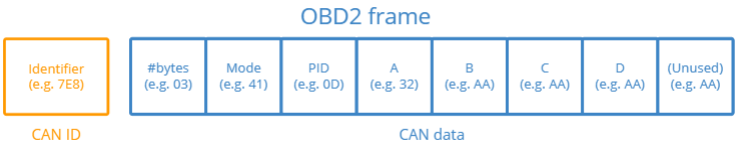


Fonte: Pesé et al. (2019)

O frame de dados é o tipo de mensagem mais comum, e sua função é transmitir os dados no barramento, enquanto o frame remoto tem como objetivo solicitar a transmissão de dados de outro nó (BOSCH, 1991).

Na Figura 6 dois campos estão destacados dos demais. Esses campos são os que contêm dados importantes para a obtenção das informações relevantes no contexto do OBD-II (Figura 7).

Figura 7 – Interpretação do *frame* CAN no contexto do padrão OBD-II.



Fonte: csselectronics.com

O campo CAN ID é a identificação de qual dispositivo enviou o *frame* de dados, ou de qual dispositivo se está requisitando dados. Possui duas variações, uma padrão, com 11 bits, e uma estendida com 29 bits. O campo Data pode conter até 8 bytes, sendo o primeiro byte para indicar quantos bytes após ele contêm dados, o segundo byte para indicar o modo, o terceiro byte para indicar o PID (Parameter ID) e os próximos bytes são dados em si (CSS ELECTRONICS, 2022b).

1.4 Modos e PIDs

Existem 10 serviços (ou modos) de diagnóstico no OBD-II, conforme descrito na SAE J1979. O modo 1 (modo empregado na elaboração desse trabalho) é usado para observar parâmetros em tempo real, como velocidade do veículo, rotação do motor, posição do acelerador, etc. Outros modos são usados, por exemplo, para mostrar/limpar códigos de diagnóstico de problemas (DTCs)(Tabela 2).

Tabela 2 – Serviços de diagnóstico do padrão OBD-II.

Número do serviço	**Descrição**
01	Mostrar dados atuais
02	Mostrar frame congelado
03	Mostrar DTC armazenado
04	Limpar DTCs e valores armazenados
05	Resultados de teste para sensor de oxigênio
06	Resultados de teste para sistema de monitoramento
07	Mostrar DTCs pendentes
08	Operação de controle de sistemas on-board
09	Solicitar informações do veículo
10	DTCs permanentes

Fonte: SAE J1979

Os fabricantes não precisam oferecer suporte a todos os serviços de diagnóstico, e podem oferecer suporte a modos diferentes desses 10 serviços, ou seja, serviços OBD-II específicos do fabricante (SOCIETY OF AUTOMOTIVE ENGINEERS, 2002b).

Outra parte importante da SAE J1979 é a especificação de uma faixa de códigos padronizados, e público, que remete a inúmeros sensores do veículo. Vários parâmetros são acessados por meio desse código chamado PID. A Tabela 3 apresenta alguns exemplos de códigos PID e seus respectivos parâmetros, bem como a fórmula para cálculo dos parâmetros a partir dos dados "brutos". A fórmula utiliza a nomenclatura dos bytes seguindo a estrutura apresentada na Figura 7 (KIM et al., 2011).

Tabela 3 – Exemplos de PIDs e respectivas fórmulas.

Número do PID	**Parâmetro**	**Fórmula**
05	Temperatura do líquido de arrefecimento (°C)	A-40
12	Rotação do motor (RPM)	$((256 * A) + B)/4$
13	Velocidade do veículo (Km/h)	A
15	Temperatura do ar de admissão (°C)	A-40
17	Abertura do corpo de borboleta (%)	A/2,55
20	Sensor de oxigênio 1 (V)	A/200
21	Sensor de oxigênio 2 (V)	A/200
47	Nível do tanque de combustível (%)	A/2,55
66	Tensão do módulo de controle (V)	$((256 * A) + B)/1000$
73	Posição do pedal do acelerador (%)	A/2,55

Fonte: SAE J1979

Novamente, nem todos os veículos oferecem suporte a todos os PIDs, e os fabricantes podem definir PIDs customizados não definidos na especificação.

2 Metodologia

A seguir é descrita a metodologia utilizada para o desenvolvimento do trabalho, apresentando os materiais utilizados, a conexão entre os módulos e protocolos de comunicação utilizados entre eles, e o *firmware* desenvolvido.

2.1 Materiais Utilizados

Os seguintes materiais foram utilizados na montagem do protótipo:

- Arduino Uno R3;
- Módulo CAN Bus MCP2515;
- SD Shield W5100;
- 3 LEDs;
- 3 resistores de 220Ω/5%;
- 4 diodos 1N4007;
- Cartão MicroSD;
- Conectores macho e fêmea molex de 4 pinos;
- Cabos e jumpers.

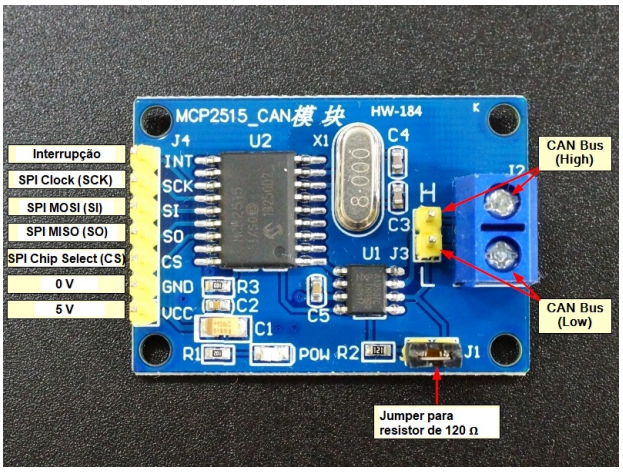
2.1.1 Módulo CAN Bus

O módulo utilizado para realizar a comunicação com o barramento CAN do OBD-II (Figura 8), é composto por dois CIs (circuitos integrados), o MCP2515 e o TJA1050.

O TJA1050 é um CI transceptor da Philips Semiconductors para interface entre o protocolo CAN e o barramento físico. O dispositivo apresenta capacidade de transmissão diferencial para o barramento e capacidade de recepção diferencial para o controlador (PHILIPS SEMICONDUCTORS, 2003). Ele executa a função de converter os sinais CAN High e CAN Low em bits 0s e 1s e vice-versa, realizando a leitura/transmissão no barramento CAN. A comunicação é feita diretamente com o CI MCP2515.

O MCP2515 é um controlador CAN da Microchip, desenvolvido para simplificar aplicações que requerem interface com um barramento CAN. Ele executa as funções de receber e transmitir *frames* no protocolo CAN, fazendo a interface com o microcontrolador por meio do padrão SPI (Interface Serial Periférica). As mensagens são transmitidas carregando o *buffer* de mensagens e registradores de controle, e a transmissão é iniciada usando bits de controle via interface SPI. Qualquer mensagem recebida é verificada

Figura 8 – Módulo CAN Bus MCP2515.



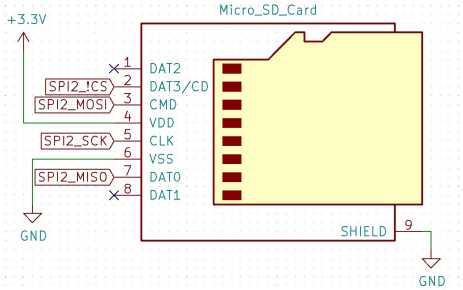
Fonte: Elaborada pelo autor

quanto a erros e, em seguida, movida para um dos dois buffers de recepção (MICROCHIP TECHNOLOGY INC., 2007).

2.1.2 Módulo SD

Apesar de sua aplicação principal ser a conexão Ethernet, o *shield* W5100 possui um adaptador para cartão MicroSD que facilita a sua utilização, além de permitir a conexão de outros dispositivos às portas de E/S do Arduino. O cartão de memória MicroSD utiliza como protocolo padrão de comunicação o SPI. Os pinos 2, 3, 5 e 7 (chip select, MOSI, clock e MISO, respectivamente) são os responsáveis pela conexão SPI do cartão de memória com o Arduino (ELM-CHAN, 2019)(Figura 9).

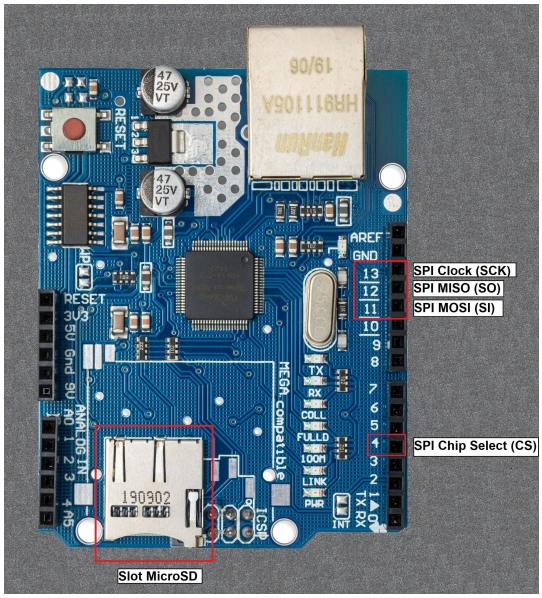
Figura 9 – Pinos do cartão MicroSD.



Fonte: github.com

No *shield*, o cartão de memória recebe a tensão necessária para funcionamento e os pinos de comunicação SPI são conectados diretamente às portas 4, 11, 12 e 13 do Arduino, conforme a Figura 10.

Figura 10 – SD Shield W5100.

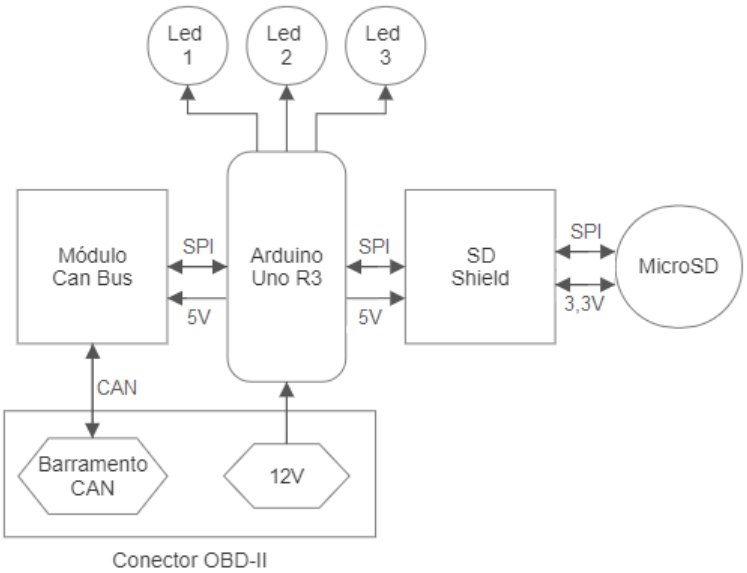


Fonte: Elaborada pelo autor

2.2 Arquitetura da Interface de Hardware para OBD-II

A Figura 11 mostra o diagrama de blocos da interface de hardware com data logger para OBDII.

Figura 11 – Diagrama de blocos da interface de hardware construída.



Fonte: Elaborada pelo autor

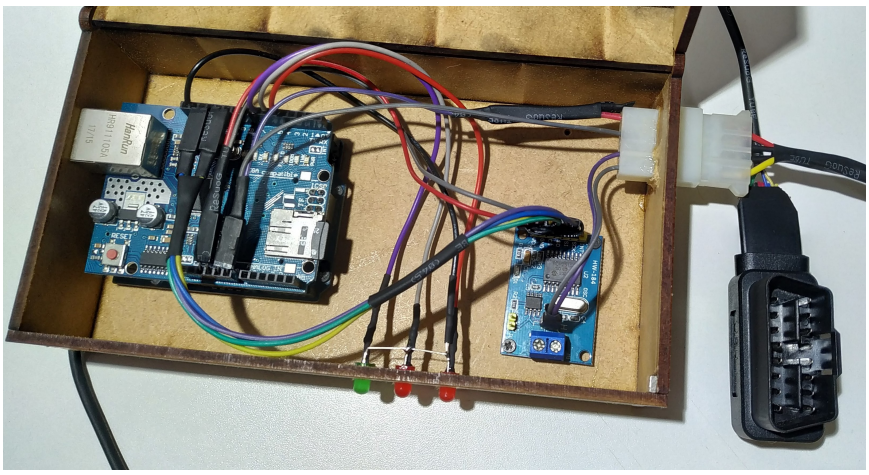
Com o *shield* SD conectado ao Arduino Uno, os pinos CS (*chip select*), MOSI, MISO e SCK (*clock*) do módulo CAN foram conectados às portas 10, 11, 12 e 13, respectivamente. Formando, assim, uma rede SPI na qual o Arduino é o mestre e os módulos CAN e SD são os escravos. Ambos os módulos são alimentados com a tensão de 5V fornecida pelo Arduino (ARDUINO DOCS, 2022).

O módulo SD e o módulo CAN são acessados via protocolo SPI em momentos diferentes, logo, a escrita/leitura no cartão de memória e a escrita/leitura no barramento CAN não apresentam problemas de conflito.

O Arduino é alimentado com a tensão de 12V fornecida por meio do cabo ligado ao conector OBD-II do veículo. Como essa tensão não é constante e pode alcançar 14V em alguns momentos, quatro diodos 1N4007 foram conectados em série para se obter uma queda de tensão quando polarizados diretamente. Essa queda, medida com auxílio de um multímetro, é de 3,2V, suficiente para que a tensão na entrada do Arduino fique dentro de sua faixa de funcionamento (7V a 12V).

Pelo mesmo cabo também é feito o acesso os fios CAN High e CAN Low do barramento CAN. Esses dois fios são conectados aos respectivos conectores no módulo CAN. Um conector molex de quatro pinos foi adicionado na outra extremidade do cabo para facilitar a conexão e desconexão (Figura 12).

Figura 12 – Interface de hardware para OBD-II montada em caixa de madeira (L=175mm, A=60mm, P=95mm).



Fonte: Elaborada pelo autor

Por fim, três LEDs foram conectados às portas 5, 6 e 7 do Arduino, com resistores de 220Ω soldados a seus anodos para limitar a corrente, tendo como finalidade indicar se houve erro ou sucesso na conexão com o cartão de memória, acesso aos arquivos e gravação dos dados.

2.3 *Firmware* do Arduino

O *firmware* foi criado utilizando-se a IDE Arduino e a Figura 14 mostra o fluxograma do mesmo.

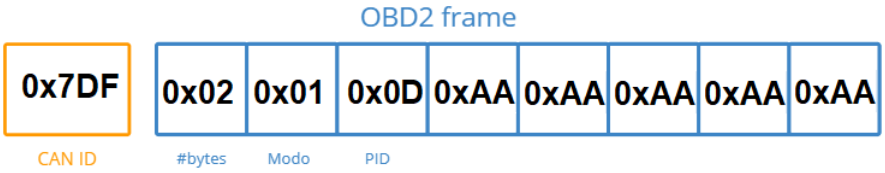
Após as definições dos pinos e declaração das variáveis globais, são inicializados os módulos CAN e SD, utilizando bibliotecas específicas.

Então, é realizado o acesso ao cartão de memória e a busca por um arquivo chamado "config.txt". Este arquivo deve conter os PIDs a serem lidos e o tempo de amostragem desejado, em milissegundos, seguindo a seguinte formatação:

$$PID_1, PID_2, PID_3, ..., PID_n, tempo(ms)$$

O mesmo tempo é aplicado para todos os parâmetros, não havendo um limite máximo. A requisição à ECU é feita enviando um *frame* remoto com campo ID contendo o endereço 0x7DF (chamado de endereço de *broadcast*), o qual corresponde a uma mensagem de solicitação no OBD-II. O modo de operação é o modo 1, o PID depende do arquivo config e os demais bytes são desconsiderados (Figura 13)(CSS ELECTRONICS, 2022a).

Figura 13 – Exemplo de *frame* solicitando a velocidade do veículo (PID 13).



Fonte: Elaborada pelo autor

O firmware faz a requisição de um PID por vez e espera a resposta da ECU. A cada PID um laço acessa mil vezes o *buffer* do módulo CAN em busca da resposta desejada. Os bytes recebidos passam por uma função de conversão, utilizando as fórmulas conforme a Tabela 3.

A informação de tempo é obtida pela função millis() do Arduino. Após passar por todos os PIDs e seus dados terem sido convertidos, uma nova linha é escrita no arquivo "dados.csv", no seguinte formato:

$$tempo(s), valor_{PID_1}, valor_{PID_2}, valor_{PID_3}, ..., valor_{PID_n}$$

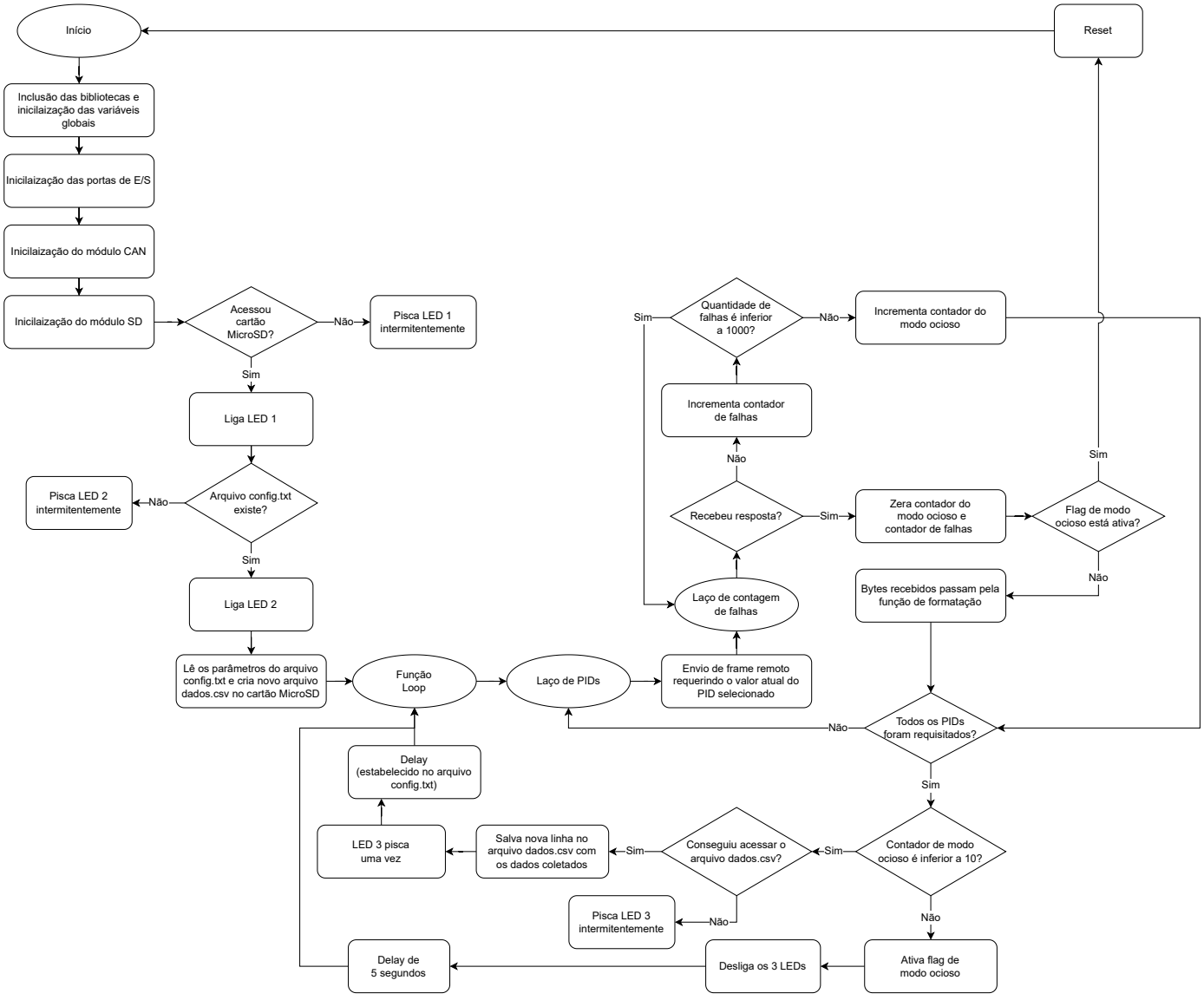
Um período de tempo é então aguardado, de acordo com o valor informado no "config.txt", e uma nova rodada de requisições dos PIDs é realizada.

Caso o Arduino não obtenha resposta, considera-se uma falha e o valor zero é gravado, seguindo para o próximo PID. Ao completar 10 falhas consecutivas, é ativada

a variável de ociosidade, e o tempo de espera entre cada leitura muda para 5 segundos[1], assumido que o veículo foi desligado, e a ECU parou de responder.

Ao ligar a ignição do veículo novamente, o Arduino é reiniciado e o firmware é executado desde o início, criando um novo arquivo "dados.csv".

Figura 14 – Fluxograma do firmware do Arduino.



Fonte: Elaborada pelo autor

A leitura do barramento CAN foi realizada a 500 kbps, por ser a velocidade utilizada pela montadora do veículo utilizado (obtida por tentativa e erro em testes realizados). As taxas de transmissão padronizadas pela norma ISO 15765-4 são de 250 kbps e 500

[1] Tempo razoável para perceber que a ECU ligou, caso o tempo informado no "config.txt"seja muito grande, na grandeza de minutos, por exemplo

kbps, podendo variar de acordo com a legislação de cada país (ISO INTERNATIONAL STANDARD, 2005).

3 Resultados

Nos primeiros testes realizados o *firmware* somente lia as informações no barramento. Verificou-se que *frames* eram exibidos sucessivamente no terminal e não seguiam padronização. Posteriormente descobriu-se que eram *frames* enviados pela ECU ao painel de instrumentos do veículo, com codificação própria da montadora.

Diversas tentativas de requisição de dados foram feitas, usando a codificação apropriada, mas sem resultado. Após várias pesquisas, encontrou-se em um tópico do Arduino Forum (2022) a indicação para ser utilizado o endereço de *broadcast* 0x98DB33F1. Utilizando esse novo endereço na requisição, um *frame* com ID igualmente alto apareceu no terminal enquanto o *firmware* lia o barramento. O *frame* seguia a padronização e indicava uma resposta da ECU para a requisição. Resolvido esse problema, as novas requisições passaram a obter respostas com êxito.

A Figura 15 mostra um trecho do arquivo criado no cartão de memória, cujos PIDs escolhidos foram o 12 (rotação do motor) e o 13 (velocidade do veículo) e o tempo de amostra foi de 200 ms.

Figura 15 – Dados escritos no cartão de memória em um teste com o veículo parado.

```
time(seconds),PID [12] value,PID [13] value
0.87,817.00,0.00
1.02,807.00,0.00
1.22,813.00,0.00
1.41,817.00,0.00
1.61,811.00,0.00
1.80,812.00,0.00
2.00,814.00,0.00
2.20,809.00,0.00
2.40,811.00,0.00
2.60,816.00,0.00
2.80,812.00,0.00
3.00,806.00,0.00
3.20,812.00,0.00
3.40,816.00,0.00
3.60,812.00,0.00
3.79,811.00,0.00
3.99,811.00,0.00
4.19,808.00,0.00
4.39,800.00,0.00
4.58,806.00,0.00
4.78,809.00,0.00
```

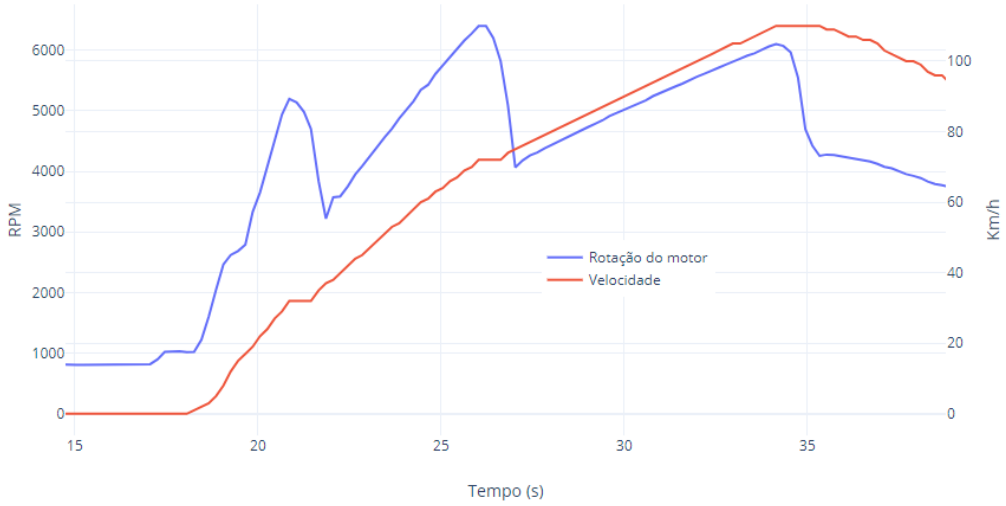
Fonte: Elaborada pelo autor

Com o arquivo é possível analisar os parâmetros, ou gerar um gráfico para observar as variações do parâmetro ao longo do tempo.

Na Figura 16 pode-se observar o aumento progressivo da velocidade do veículo e três picos no gráfico da rotação do motor, que correspondem as mudanças de marcha, primeira para segunda, segunda para terceira e terceira para quarta.

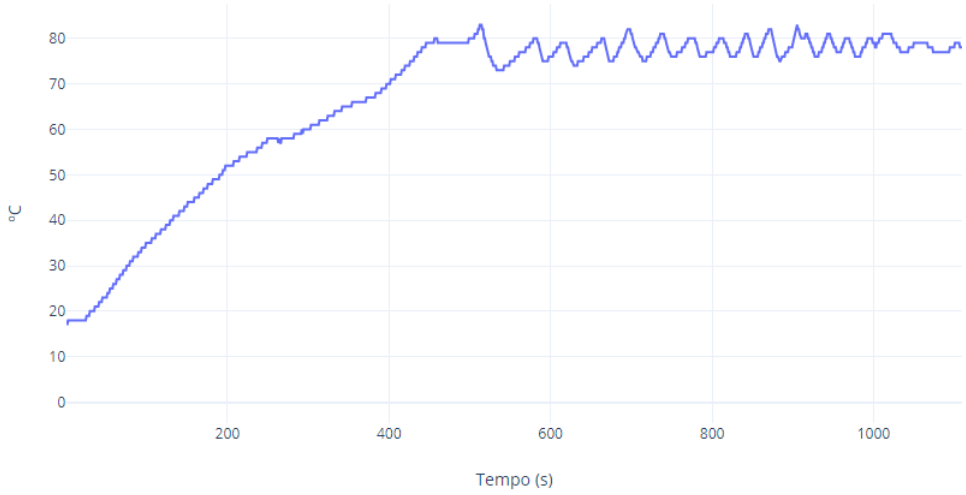
Na Figura 17 observa-se o aumento progressivo da temperatura do líquido de arrefecimento, até aproximadamente 80 °C, quando então a ventoinha do radiador começa a ligar e desligar periodicamente, gerando a oscilação de temperatura em torno de 80 °C.

Figura 16 – Gráfico mostrando os parâmetros rotação do motor e velocidade do veículo ao longo do tempo.



Fonte: Elaborada pelo autor

Figura 17 – Gráfico mostrando a temperatura do líquido de arrefecimento após uma partida "a frio", com o veículo parado.



Fonte: Elaborada pelo autor

O fato de se ter utilizado *jumpers* (sem soldagem) na montagem do protótipo ocasionou mau contato em alguns momentos, interferindo no funcionamento do sistema. Quando ocorria o mau contato, parte dos *frames* enviados pelo Arduino ao barramento CAN era perdida e causava interferência no painel de instrumentos do veículo, dado que ele está ligado ao mesmo barramento.

A requisição de PIDs não suportados pela ECU do veículo também ocasionou, com menor frequência, os mesmos problemas no painel de instrumentos. Luzes e avisos de erro surgiam, como falha no AirBag, pressão baixa nos pneus, porta aberta, etc. Em outro veículo ocorreu o bloqueio momentâneo da alavanca do câmbio automático, que voltou a operar normalmente após desconectar e conectar novamente a bateria.

4 Conclusão

O objetivo desse trabalho foi desenvolver uma interface de *hardware* que possibilitasse a sua utilização como *data logger* em veículos com padrão OBD-II. Inicialmente realizou-se um estudo teórico sobre o assunto, e na sequência a montagem do *hardware* utilizando os módulos CAN e SD, a elaboração do *firmware* e os testes.

A utilização de bibliotecas específicas para os módulos facilitou o desenvolvimento do *firmware* e acelerou o desenvolvimento do sistema. Foi possível realizar o registro de vários parâmetros do veículo conjuntamente de forma satisfatória, inclusive com detecção de desligamento da ECU, reduzindo o consumo da memória de armazenamento.

Como sugestões de melhorias, pode-se citar a utilização de interrupção interna gerada por temporizador no Arduino, para melhorar a exatidão do intervalo de tempo entre as leituras, além da possibilidade de definir intervalos diferentes para cada parâmetro medido. Também, a utilização de um Arduino Nano em conjunto com um conector para memória micro SD, possibilitará reduzir as dimensões do sistema. Por fim, uma montagem utilizando solda eliminará os problemas de mau contato.

O desenvolvimento deste sistema abre possibilidades de uso do *hardware* em outras aplicações além de *data logger*, como monitoramento de velocidade em veículos rastreados, desenvolvimento de painel de instrumentos auxiliar, cronômetro para testes de aceleração de 0 a 100 km/h em pistas de competição e outros.

Por fim, dois pontos destacam o sistema, a capacidade de coletar e armazenar dados do veículo por um longo período de tempo[1,2], e o baixo custo (aproximadamente R$ 200,00). Portanto, acredita-se que esse trabalho fez uma contribuição importante na área de eletrônica automotiva.

[1] Período máximo de 49 dias, devido à limitação da função millis() do Arduino.
[2] Escolhendo somente um parâmetro, estima-se que 1MB é capaz de armazenar cerca de 80.000 leituras.

Referências

ARDUINO DOCS. *Arduino UNO R3 Product Reference Manual*. [S.l.], 2022. Disponível em: <https://docs.arduino.cc/static/dfba855f59b654b11ee185475640c9b5/A000066-datasheet.pdf>. Acesso em: 12.7.2022. Citado na página 22.

ARDUINO FORUM. *OBD2 Extended IDs: no data received and error on dashboard*. [S.l.], 2022. Disponível em: <https://forum.arduino.cc/t/obd2-extended-ids-no-data-received-and-error-on-dashboard/941865/4>. Acesso em: 24.8.2022. Citado na página 27.

BOSCH. *CAN Specification Version 2.0*. Alemanha, 1991. Citado na página 16.

BOSCH. *CAN User's Manual Revision 3.3.0*. Alemanha, 2018. Citado na página 16.

CSS ELECTRONICS. *How To Transmit CAN Bus Messages: OBD2 PID Example*. [S.l.], 2022. Disponível em: <https://www.csselectronics.com/pages/obd-ii-pid-examples>. Acesso em: 7.7.2022. Citado na página 23.

CSS ELECTRONICS. *OBD2 Explained - A Simple Intro [2022]*. [S.l.], 2022. Disponível em: <https://www.csselectronics.com/pages/obd2-explained-simple-intro>. Acesso em: 7.7.2022. Citado na página 17.

ELM-CHAN. *How to Use MMC/SDC*. [S.l.], 2019. Disponível em: <http://elm-chan.org/docs/mmc/mmc_e.html>. Acesso em: 10.4.2023. Citado na página 20.

ISO INTERNATIONAL STANDARD. *ISO 15765-4: Road vehicles — Diagnostics on Controller Area Networks (CAN) — Part 4: Requirements for emissions-related systems*. Suíça, 2005. Citado na página 25.

ISO INTERNATIONAL STANDARD. *ISO 11898-3: Road vehicles — Controller area network (CAN) — Part 3: Low-speed, fault-tolerant, medium-dependent interface*. Suíça, 2006. Citado na página 15.

KIM, T. hoon et al. *Communications in computer and information science*. Coréia do Sul, 2011. Citado na página 18.

MICROCHIP TECHNOLOGY INC. *Data Sheet - MCP2515 Stand-Alone CAN Controller With SPI Interface*. EUA, 2007. Citado na página 20.

PESé, M. D. et al. *LibreCAN: Automated CAN Message Translator*. EUA, 2019. Citado na página 16.

PHILIPS SEMICONDUCTORS. *Data Sheet - TJA1050 High speed CAN transceiver*. Holanda, 2003. Citado na página 19.

SOCIETY OF AUTOMOTIVE ENGINEERS. *SAE J1962: Diagnostic Connector Equivalent to ISO/DIS 15031*. EUA, 2002. Citado na página 13.

SOCIETY OF AUTOMOTIVE ENGINEERS. *SAE J1979: E/E Diagnostic Test Modes*. EUA, 2002. Citado na página 17.

WATTERSON, C. *AN-1123 APPLICATION NOTE: Controller Area Network (CAN) Implementation Guide*. EUA, 2017. Citado na página 14.