UNIVERSIDADE FEDERAL DO TOCANTINS
CAMPUS PALMAS
PROGRAMA DE PÓS-GRADUAÇÃO PROFISSIONAL EM MODELAGEM COMPUTACIONAL DE SISTEMAS

Gabriel Wermuth Stroligo

CONTROLE DA FROTA AUTOMOTIVA DO PODER JUDICIÁRIO ATRAVÉS DA INTERNET DAS COISAS

PALMAS
2020

Gabriel Wermuth Stroligo

CONTROLE DA FROTA AUTOMOTIVA DO PODER JUDICIÁRIO ATRAVÉS DA INTERNET DAS COISAS

Projeto de qualificação apresentado ao Programa de Pós-Graduação Profissional em Modelagem Computacional de Sistemas da Universidade Federal do Tocantins como requisito parcial para obtenção do título de Mestre em Modelagem Computacional de Sistemas (PPGMCS/UFT).
Orientador: Prof. Dr. Humberto Xavier de Araujo

PALMAS
2020

Agradeço a oportunidade que me é dada de estudar em uma instituição pública de qualidade. Que mesmo vivendo tempos sombrios de sucateamento do ensino superior, se mantém forte, produzindo ciência e agregando valor à sociedade.

Agradeço ao meu orientador pela obstinação em busca dos melhores resultados sendo fundamental para que esse trabalho fosse desenvolvido.

RESUMO

O conceito de otimização de recursos é amplamente utilizado em setores de tecnologia, principalmente no que diz respeito ao desenvolvimento de software. No entanto, a aplicação de tais otimizações tem avançado para diversas outras áreas. O principal exemplo são as empresas, tanto públicas como privadas que otimizam desde recursos humanos, passando por controle dos custos operacionais até a potencialização do uso de ferramentas necessárias para a operação dos negócios. Dessa forma, a utilização de Internet das Coisas tem auxiliado amplamente na diminuição de desperdícios, com uso de sensores capazes de transformar em números precisos algo que na maioria dos casos era estimado erroneamente. Os veículos continuam sendo uma das principais ferramentas utilizadas por diversos ramos de negócios e contribui para ser uma das principais fontes de custos. Dessa forma, diante das profundas transformações que estão em curso no cenário econômico, social e ambiental no Brasil, e que estão refletindo de maneira tão significativa nas condutas de gestão pública atualmente, surge cada vez mais a necessidade de uma contribuição para que tais normativas possuam um embasamento técnico assertivo, acerca da utilização da frota automotiva do Poder Judiciário do Estado do Tocantins, para conduzir seus processos e procedimentos de forma eficiente e idônea. Portanto, é desenvolvido e testado um sistema embarcado capaz de obter dados de veículos através do protocolo CAN e enviá-los a um banco de dados NoSQL por meio de GPRS servindo de base para um posterior desenvolvimento de um sistema de gerenciamento de frotas. Para fins de validação o sistema foi testado em tempo real em um veículo em movimento.

PALAVRAS-CHAVE: Internet das Coisas. Monitoramento de Frota. Tracker veicular.

ABSTRACT

The concept of resource optimization is widely used in technology sectors, mainly with regard to software development. However, the application of such optimizations has advanced to several other areas. The main example are companies, both public and private, that optimize from human resources, going through the control of a company's costs until the potential use of tools necessary for the business operation. In this way, the use of Internet of Things has largely helped to reduce waste, with the use of sensors capable of transforming into precise numbers something that in most cases was estimated erroneously. Vehicles continue to be one of the main tools used by various branches of business and contribute to being a major source of costs. Thus, facing the profound transformations that are taking place in the economic, social and environmental scenario in Brazil, and which are reflecting in a significant way in the current public management practices, there is an increasing need for a contribution so that such regulations have an assertive technical background on the use of the automotive fleet of the Judicial Branch of the State of Tocantins to conduct its processes and procedures in an efficient and appropriate manner. Thus, an embedded system capable of obtaining vehicle data through the CAN protocol and sent to a NoSQL database through GPRS is developed and tested serving as a basis for further development of a fleet management system. For validation purposes the system was tested in real time on a moving vehicle.

KEYWORDS: Internet of Things. Fleet Monitoring. Vehicle Tracker.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIAÇÕES

AP	Access Point
CAN	Controller Area Network
DRC	Design Rule Checking
DSI	Divisão de Sistemas de Informática
DTC	Diagnostic Trouble Code
FGV	Fundação Getúlio Vargas
GPRS	General Packet Radio Service
GPS	Global Positioning System
GSM	Sistema Global para Comunicações Móveis
I/O	Input/Output
I²C	Inter-IC
IDE	Integrated Development Environment
IoT	Internet of Things
LED	Light Emitter Diode
MOSFET	Metal Oxide Semiconductor Field Effect Transistor
MVP	Minimum Viable Product
OBD	On-Board Diagnostic
PCB	Printed Circuit Board
PES	Projeto Esplanada Sustentável
PID	Parameter ID
PROM	Programmable Read-Only Memory
RAM	Random Access Memory
RF	Radiofrequência
RPM	Revoluções por Minuto
SAE	Society of Automotive Engineers
SD	Secure Digital
SMD	Surface Mount Device
SOC	System on Chip

SPI	Serial Peripheral Interface
TQFP	Thin Quad Flat Pack
UFT	Universidade Federal do Tocantins
USB	Universal Serial Bus
WiFi	Wireless Fidelity

SUMÁRIO

1 INTRODUÇÃO

1.1 CONSIDERAÇÕES INICIAIS

O mundo está passando por imensas transformações no que diz respeito a produtividade e eficiência com o intuito de melhorar não só os processos de empresas ou utilizar melhor os recursos disponíveis, mas também, ser responsável por aumentar a sustentabilidade. Em se tratando de políticas públicas, o cenário converge para o mesmo resultado. Dessa forma, essa dissertação oferece uma solução eficiente e idônea para a utilização da frota automotiva do Poder Judiciário do Estado do Tocantins.

Além do uso eficiente dos recursos e boa gestão do gasto público, é necessário manter a segurança dos servidores que fazem uso da frota de veículos através da supervisão das rotas utilizadas, bem como o monitoramento do comportamento dos condutores. Todos esses dados são transmitidos com garantia máxima de que não serão interceptados, com a devida implantação de serviço de encriptação em uma base segura (FIREBASE, 2019).

Já existe uma busca natural e consciente aos processos de gestão do gasto público de maneira eficiente. Como, por exemplo, o Projeto Esplanada Sustentável (PES) (IPEIA, 2012), “um projeto de iniciativa do Governo Federal que visa integrar ações para a melhoria da eficiência no uso racional dos recursos públicos pela melhoria da gestão de processos, considerando a inserção da variável socioambiental no ambiente de trabalho”, que desenvolveu uma Coletânea de Melhores Práticas de Gestão do Gasto Público, onde já se recomenda a utilização de um Sistema Eletrônico de Controle de Combustível. Contudo, a solução proposta pelo presente artigo busca uma abordagem mais simples e barata de ser implantada, além de recolher e disponibilizar um maior número de informações.

Além disso, no cenário atual, há uma onda de tecnologia revolucionária conhecida como Internet das Coisas (do inglês Internet of Things, por isso conhecida pela sigla IoT) que a cada dia está conectando mais e mais dispositivos à internet, tem-se uma previsão de 25 bilhões de aparelhos conectados até 2020 (PARKES, 2016). Esses dispositivos não têm apenas a função de retirar informações do ambiente e levá-las a um servidor ou banco de dados, mas também podem ter a função de atuadores em algum tipo de sistema, permitindo desde o controle de um motor de indução trifásico em uma indústria até o controle automático de temperatura em um prédio de acordo com as variações da temperatura externa.

Dessa forma, essa dissertação se delimita na criação de um protótipo para possível implantação em toda a frota automotiva do Poder Judiciário do Estado do Tocantins, integrado com as informações obtidas pela Diretoria de Transporte do mesmo órgão, junto com a equipe técnica da DSI (Divisão de Sistemas de Informática).

1.2 JUSTIFICATIVA

Diante das profundas transformações que estão em curso no cenário econômico, social e ambiental no Brasil, e que estão refletindo de maneira tão significativa nas condutas de gestão pública atualmente, a presente dissertação surge com a intenção de contribuir para que tais normativas possuam um embasamento técnico assertivo, acerca da utilização da frota automotiva do Poder Judiciário do Estado do Tocantins, para conduzir seus processos e procedimentos de forma eficiente e idônea.

Para tanto, é possível enumerar diversas melhorias relevantes que a promoção do controle da frota propiciará em diversas frentes. No que concerne especificamente a economia financeira, pode-se afirmar que a boa gestão do gasto público permite a redução de desperdícios, possibilitando o aumento dos recursos financeiros disponíveis para o Estado atender melhor a população de outras diversas formas e em outros nichos, conhecidamente tão carentes de recursos.

Além disso, vale ressaltar que esta redução, para lá de gerar contenções fazendárias, também proporciona outro tipo de economia: a de recursos naturais. Fato este que hoje, dadas as circunstâncias ambientais em que vivemos, se torna extremamente imprescindível. Por isto mesmo, nota-se um crescimento do número de propostas cujo objetivo são a adoção de modelos de gestão organizacionais e de processos estruturados na implantação de ações voltadas ao uso racional destes recursos.

Um outro ponto importante a ser abordado, é o de promover a manutenção da integridade física dos servidores e, sobretudo, dos magistrados já que o monitoramento de suas rotas e do comportamento de seus condutores gera um resguardo a mais acerca de sua seguridade. Evidencia-se aqui que tais recolhimentos de informações se darão com a garantia máxima de que os dados não serão interceptados, com a devida implantação do serviço em base segura.

Por fim, é imprescindível citar a abordagem relativa à transparência que permeia todo o embasamento do projeto e seus testes.

Desde que de forma cautelosa, a divulgação e difusão dos dados colhidos caminha em conjunto com as mais atuais diretrizes de lisura nos processos de gestão do gasto público, que inclusive já possuem algumas iniciativas que vêm se consolidando há algum tempo.

Além disso, existem outras organizações que se pautam nesse preceito, como o Centro de Justiça e Sociedade (FGV Direito Rio) e o Grupo de Estudos Anticorrupção (FGV Direito SP), que em parceria com a Transparência Internacional - Programa Brasil, desenvolveram um conjunto de medidas anticorrupção a serem apresentadas para a sociedade brasileira. Elas buscam construir, em conjunto com outros setores da sociedade, uma plataforma de propostas de reforma legislativa, administrativa e institucional, para colaborar com o debate público e buscar meios de contribuir com soluções para combater a corrupção a longo prazo. Um de seus anteprojetos de leis, sobre a Política Nacional de Dados Abertos, corrobora que o acesso à informação deve ser promovido pelo poder público nos termos da Lei nº 12.527, de novembro de 2011, na Lei de Responsabilidade Fiscal e demais normas vigentes.

Em suma, é possível concluir que se pode validar de forma positiva todos os impactos no que concerne a aplicação deste projeto, que se mostra viável, relevante e exequível, além de se evidenciar por um cunho particularmente inovador e estar fundamentado em iniciativas atuais já validadas.

1.3 OBJETIVO GERAL

Controlar a frota automotiva do Poder Judiciário através de um dispositivo acoplado ao veículo que envia dados em tempo real a um servidor. Com esses dados será possível monitorar e rastrear todos os carros da instituição.

Como base de estudo, esse projeto irá permear o mundo da *IoT (Internet of Things)* ou *Internet das Coisas*, incluindo microeletrônica básica, sistemas embarcados e finalizando a pesquisa com uso de banco de dados NoSQL, visando a usabilidade da ferramenta.

A finalidade é propor um modelo, com baixo custo de implementação e código aberto a ser desenvolvido exclusivamente para o Tribunal de Justiça do Estado do Tocantins, mas que futuramente possa ser replicado em outros órgãos públicos.

1.4 OBJETIVOS ESPECÍFICOS

Este trabalho é fundamentado, tendo como principais objetivos específicos:

- Manutenção preventiva da frota;
- Controle do consumo de combustível;
- Localização em tempo real dos veículos;
- Criação e manutenção de um banco de dados NoSQL.

A pesquisa se inicia através da criação de um protótipo com conexão à internet por meio do módulo *GSM GPRS Quad-band Sim808* com *GPS*, além disso, coletará dados do veículo pelo controlador *CAN MCP2515* conectado na porta ODB II, tendo como objetivo entregar na conclusão deste processo, um modelo eletrônico para miniaturização, com a colocação de todos estes componentes em um sistema embarcado, único e com documentação pública.

Ao final, a pesquisa se encerra com a entrega de uma plataforma web e banco de dados Firebase, onde serão exibidos os dados e a localização de todos os veículos da frota em tempo real, com um registro de histórico de manutenção dos mesmos, servindo como uma ferramenta para gerenciamento e controle.

1.5 METODOLOGIA

1.5.1 ESTRUTURA DA DISSERTAÇÃO

De forma objetiva esse projeto foi pensado em 3 etapas interligadas. Na primeira etapa, é entregue um protótipo (o hardware) da solução conforme descrito no próximo tópico, em seguida será realizado o desenvolvimento da lógica do sistema e conexões com o banco de dados e por fim, na última etapa, será realizada a aplicação e coleta de resultados.

Será apresentado os materiais e métodos utilizados para o desenvolvimento do trabalho. Além disso, características da pesquisa, o *hardware* e *software* utilizados na construção do protótipo, vão ser detalhados, bem como a demonstração do seu funcionamento.

1.5.2 CARACTERÍSTICAS DA PESQUISA

Nesta pesquisa, como metodologia, propõe-se a abordagem qualitativa, na qual a principal ferramenta é plataforma a ser desenvolvida. Ele que faz a análise dos dados coletados, buscando conceitos, princípios, soluções e propondo novos significados. Neste método de pesquisa o resultado depende inteiramente do intelecto do pesquisador, pois é ele quem conclui o melhor caminho a ser seguido.

A pesquisa tem caráter subjetivo, com objetivo valorativo, para melhorar as ferramentas de uso comum do Poder Judiciário do Estado do Tocantins, através do controle e do levantamento de dados da sua frota veicular. Isto se dará de forma exploratória, pois o objeto de pesquisa é pouco conhecido, portanto necessita-se de mais informações a seu respeito.

Quanto a natureza da pesquisa, o foco será aplicado partindo de conhecimentos de outras pesquisas básicas. O trabalho tem como viés criar uma solução com dados próprios e ferramentas moldadas conforme as necessidades.

Os procedimentos utilizados serão aplicação de estudo de caso, com estudo documental e fundamentação teórica, através de análise textual discursiva, com a apresentação de um protótipo final.

Essa dissertação foi dividida em capítulos, sendo um total de sete. No capítulo 1 foi justificado o objetivo do projeto bem como a metodologia empregada na elaboração do protótipo.

Os capítulos 2 e 3 são o embasamento teórico do projeto. Foram descritos os blocos que compõem o sistema proposto de modo a ser possível o desenvolvimento do protótipo em si, descrevendo os pormenores das tecnologias adotadas como o protocolo CAN utilizado tanto na indústria como no meio automotivo, a porta OBD II, a comunicação entre chips, GPS, GPRS e conceitos relacionados a fabricação de protótipos eletrônicos.

Por outro lado, o capítulo 4 é o desenvolvimento do projeto em si. Nele é concentrado todos os detalhes e preparação referente à conexão dos blocos explanados no capítulo 3, bem como a explicação da comunicação do barramento CAN, detalhamento dos chips utilizados (microcontrolador, chip de RF e controlador de CAN) e esquema 3D da placa eletrônica. Também é mostrado o software de desenvolvimento e suas principais características.

O capítulo 5 detalha a criação da plataforma e uso do banco de dados. Nele é descrito o Firebase, bem como seu funcionamento, segurança, ferramentas que podem ser utilizadas em conjunto e uma exemplificação de como o banco de dados foi estruturado no projeto.

No capítulo 6 foram mostrados os resultados alcançados, exibindo os passos que foram seguidos e as dificuldades enfrentadas como a dificuldade de encontrar um programador específico para a placa desenvolvida. Também foi apresentado a placa de circuito impresso montada com todos os componentes. Além disso, foi mostrado os testes realizados, o emulador de OBD desenvolvido e os resultados alcançados.

Por fim, o capítulo 7 conclui a dissertação, comentando os resultados obtidos e elucidando a proposta de trabalhos futuros.

2 VISÃO GERAL DA SOLUÇÃO PROPOSTA

A pesquisa tem como objetivo a criação de um protótipo capaz de recolher dados disponibilizados pela porta OBD II dos veículos, bem como sua localização em tempo real através de GPS e, além disso, ser capaz de enviar esses dados para um servidor (armazenando um buffer local para o caso de falha de conexão). O resultado final será um sistema embarcado, único e com documentação pública.

Inicialmente será necessário o estudo teórico das formas de comunicação entre o veículo, o protótipo e o servidor. Começando pelo barramento CAN (CAN Bus como é referido no inglês) e padrão OBD II que está presente na maioria dos carros modernos. Em seguida, será estudado os conceitos relativos à comunicação de sistemas embarcados com servidores utilizando o Sistema Global para Comunicações Móveis (GSM). Existe, atualmente, muito material acerca desse tema, por estar intimamente relacionado ao conceito de Internet das Coisas (*Internet of Things - IoT*). Por último será realizada uma pesquisa sobre o uso de GPS para aquisição da localização.

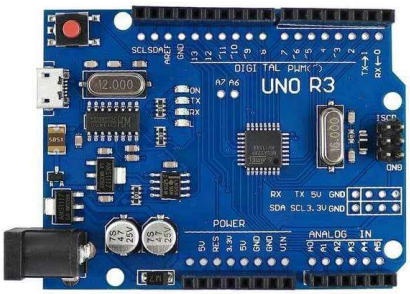
Felizmente, a comunicação entre o dispositivo, satélites e servidores, já foi abstraída, cabendo a nós apenas estudarmos os comandos necessários para que o dispositivo seja capaz de enviar/receber as informações solicitadas. Dessa forma, conceitos relacionados a ondas de RF (radiofrequência), modulação e encriptação não serão abordados em nossa pesquisa.

Depois do estudo teórico, será necessário pesquisar quais serão os dispositivos existentes no mercado no qual utilizaremos como base para o desenvolvimento do protótipo, sendo que alguns já foram previamente estudados como mostrado a seguir.

Como unidade de processamento, foi escolhida a placa de desenvolvimento Arduino UNO da plataforma de mesmo nome mostrada na Figura 1. Esses dispositivos de prototipagem são amplamente utilizados pela cultura *maker* e possuem diversas vantagens e desvantagens como será explanado no tópico 5.1.

É sabido que o desenvolvimento do Arduino foi extremamente responsável pela facilidade em se utilizar um microcontrolador. Sendo assim, até leigos em desenvolvimento eletrônico passaram a ser capazes de utilizar um microcontrolador, como é o caso de artistas e músicos. Além disso, a rapidez em se desenvolver um projeto utilizado um chip desses atraiu muitos entusiastas para a área, se tornando hoje a porta de entrada para esses projetos.

Figura 1 – Placa de prototipagem Arduino Uno

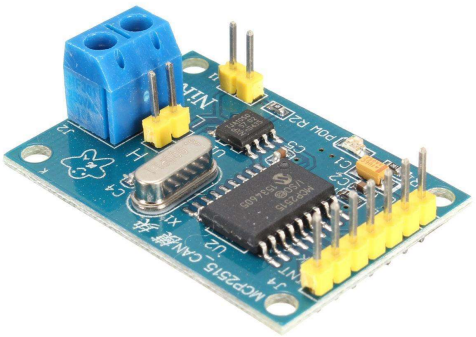


Fonte: Site de vendas DX[1]

Contudo, mesmo sendo um objeto simples de se utilizar, não significa que ela não possua seu valor. Quando bem utilizado, um ATMEGA328P (o microcontrolador da placa Arduino UNO) se mostra bastante eficiente e poderoso, sendo o chip mais escolhido quando se trata de prototipagem rápida.

Por outro lado, se tratando da comunicação entre a unidade de processamento (microcontrolador) e o veículo (através da porta OBD II) será estudado a viabilidade de um módulo vendido no mercado brasileiro, conhecido como Módulo Can Bus MCP2515 TJA1050 como mostrado na Figura 2.

Figura 2 - Módulo Can Bus MCP2515 TJA1050



Fonte: Site de vendas FilipeFlop[2]

[1] Disponível em: <https://www.dx.com/p/micro-usb-socket-atmega328p-development-board-for-arduino-uno-r3-blue-black-2049752.html#.XzNNrehKjIU>. Acesso em 20 Jul. 2020.

[2] Disponível em: <https://www.filipeflop.com/produto/modulo-can-bus-mcp2515-tja1050/>. Acesso em 20 Jul. 2020.

Esse módulo possui dois chips importantes para o perfeito funcionamento da comunicação com o barramento CAN.

Seria necessário apenas o chip TJA1050 para essa aplicação, pois ele é um *transceiver* que "transforma" a linguagem do barramento CAN para um formato que microcontroladores conseguem decifrar. Contudo, utilizando apenas o TJA1050 seria necessário portar todo o controle do protocolo CAN para o microcontrolador ATMEGA328P. Isso faria com que a já limitada memória RAM fosse mais comprometida. Sendo assim, foi utilizado mais um chip em conjunto com o TJA1050, o MCP2515 que assume o controle do protocolo CAN, por isso é chamado de controlador CAN.

Por último, para comunicação entre a unidade de processamento e os servidores, e geolocalização, será estudada a viabilidade de utilizar um módulo conhecido como SIM808 GSM GPS EVB V3.2, mostrado na figura 3.

Figura 3 - Módulo SIM808 GSM GPS EVB V3.2

Fonte: Site de vendas Aliexpress[3]

Esse módulo possui como principal e único chip o SIM808 que como mencionado no capítulo 4 é um conjunto de chips capazes de se comunicar com satélites de GPS e antenas de celular para comunicação Wireless. É um chip poderoso, que consome bastante energia e possui diversas funções que pode ser controlado por comandos AT.

Após o estudo de viabilidade do uso dos módulos supracitados e a unidade de processamento, o protótipo inicial será montado utilizando *protoboard* apenas para

[3] Disponível em: <https://pt.aliexpress.com/item/32865543253.html>. Acesso em 20 Jul. 2020.

comprovar o funcionamento em conjunto dos módulos selecionados, além de servir como um MVP (*minimum viable product* - produto mínimo viável).

Por fim, baseando-se nos módulos selecionados e na unidade de processamento, será projetada uma placa de circuito impresso utilizando os chips presentes nesses módulos de modo a miniaturizar o circuito eletrônico. Testes serão realizados em laboratório, após a fabricação do protótipo, utilizando emuladores criados justamente para esse fim. Além disso, testes serão realizados em campo utilizando carros reais.

3 SISTEMAS DE CONTROLE AUTOMOTIVO EMBARCADO

O trabalho apresentado é um estudo do desenvolvimento de um protótipo eletrônico capaz de reunir informações para compor um sistema de controle de frota veicular. Portanto, é importante ressaltar que se trata de um projeto eletrônico de aplicação prática.

Dessa forma, os temas a serem abordados neste trabalho, serão, como citado na introdução, relacionados ao protótipo, como: Barramento CAN, Framework Arduino, comunicação serial e placa de circuito impresso.

O barramento CAN engloba tanto partes físicas (utilização de dois fios como pares diferenciais) quanto o protocolo em si, que foi criado em 1986 (BOSCH, 1995), para aplicações automotivas, mas que também tem seu uso em automação industrial.

Brudna (2000) afirma que o barramento CAN é muito eficiente e de baixo custo. Além disso, possui muitos circuitos integrados controladores no mercado devido a sua popularidade. Sendo assim, seu uso é bastante difundido em várias aplicações, não somente em automóveis.

De maneira análoga, esse protocolo é bastante robusto, tendo facilidade para operação em ambientes nocivos com alta temperatura e ruídos. Isso garante uma comunicação confiável, segura e eficiente entre os mais diversos sensores e atuadores presentes em um veículo (DE ALMEIDA GUIMARÃES; SARAIVA, 2002).

A grande maioria dos carros atuais fazem uso desse barramento para transmissão e recepção dos dados entre os chips. Sendo assim, a escolha de se utilizar o CAN foi bastante natural, sendo basicamente a opção mais viável e também mais econômica, visto que os módulos que suportam protocolo CAN não passam de R$23,00.

O Arduino, muitas vezes conhecido como apenas uma placa de prototipagem é, na verdade, um framework de desenvolvimento de projetos eletrônicos. Um framework engloba toda a estrutura necessária para a programação de uma família de microcontroladores, desde a linguagem de programação, até o ambiente de desenvolvimento integrado (IDE) que possui diversas ferramentas como compiladores, gravadores e até editores de texto.

A família de microcontroladores que faz parte do framework Arduino é a ATMEGA da ATMEL, antiga desenvolvedora de semicondutores que foi incorporada pela Microchip Technology que criou os microcontroladores PIC.

McRoberts (2015) afirma que a principal vantagem do Arduino é a sua facilidade de uso em comparação com outros microcontroladores, fazendo com que o tempo gasto em um

projeto seja bem menor. Atualmente, em conjunto com o baixo consumo, o conceito de *Fast Time to Market* influencia fortemente a escolha do microcontrolador por desenvolvedores.

E por mais que seja de fácil uso, o Arduino não deixa a desejar em termos de performance e sofisticação que os engenheiros de sistemas embarcados necessitam em seus produtos de mercado (MARGOLIS, 2011), sendo assim o candidato perfeito para utilização no presente trabalho.

Segundo Axelson (2000) no universo dos computadores, são necessárias três coisas: Computadores (hardware), o programa (software) e a comunicação. Essa comunicação pode ser feita por diversos meios, desde fios comuns até por meio de ondas eletromagnéticas.

Os mais diversos requerimentos de comunicação exigem suas interfaces correspondentes em termos de velocidade de transmissão, tamanho dos dados, formato e número de computadores.

Margolis (2011), mostra que o Arduino pode se comunicar utilizando tanto I²C quanto SPI, o que facilita toda comunicação entre os mais variados chips que existem no mercado, tanto os controladores CAN quanto com o SIM808.

O resultado final desse trabalho funciona inteiramente em uma placa de circuito impresso. Essas placas, inicialmente, foram utilizadas para facilitar a montagem de dispositivos eletrônicos substituindo fios e interligando os mais diversos componentes como resistores, transistores e diodos.

Posteriormente as placas foram evoluindo e funcionando como suportes mecânicos e interface com partes móveis, chegando a ser bem complexas e com várias camadas para a miniaturização de dispositivos.

Têm-se como restrição o tamanho mecânico reduzido do protótipo eletrônico. Dessa forma serão utilizados componentes SMD e se for necessário, placa com quatro camadas de cobre.

Para o desenvolvimento dessas placas, é necessário um *checklist* de ações a serem tomadas (ROBERTSON, 2003):

- Esquemático do circuito e *netlist*;
- Desenho da placa de circuito impresso;
- Acabamento e inspeção da placa;
- Desenho de montagem.

Cada um desses itens é abordado com mais detalhes no capítulo 6 deste trabalho.

4 DESENVOLVIMENTO DA SOLUÇÃO

Sistemas embarcados fazem uso de uma unidade de processamento. Essa unidade é conhecida como microcontrolador, é um pequeno computador SOC (*System on Chip*) que em um único circuito integrado possui um núcleo de processador, periféricos I/O (*input* e *output*) programáveis e memória que pode ser FLASH, RAM ou PROM, sendo esta última uma memória programável apenas de leitura. Eles são utilizados nas mais distintas aplicações de dispositivos automatizados, desde controle de veículos e eletrodomésticos, até ferramentas elétricas e outros sistemas embarcados.

Em comparação a um dispositivo microprocessado, a utilização do microcontrolador também propicia a redução de custos, devido ao seu baixo consumo, situado na casa dos miliwatts, sendo que, em modo de espera, o mesmo pode chegar até a nanowatts.

No mercado brasileiro, o número de microcontroladores é reduzido, contudo ainda é possível encontrar uma quantidade razoável desses dispositivos. Ultimamente, existem diversos microcontroladores que possuem além das funcionalidades descritas acima, conexão sem fio, seja por WiFi ou Bluetooth devido a necessidade desse tipo de conexão para desenvolvimento de Internet das Coisas.

Para a escolha do microcontrolador, foi levado em consideração as seguintes características:

- Facilidade de implementação;
- Rapidez de prototipagem;
- Fácil reprodução de hardware;

Logo, utiliza-se um microcontrolador em que a linguagem seja de simples entendimento, além de possuir um bom ambiente de desenvolvimento com grande número de bibliotecas, documentação atualizada e completa e suporte a hardware e comunicação (sensores, protocolos de comunicação, *shields* e atuadores) (MARGOLIS, 2011). Além disso, esse microcontrolador precisa ser rápido para configurar os registradores no que diz respeito a velocidade de *clock*, modo de funcionamento, *interrupts* e contadores.

Para se obter protótipo eletrônico fácil de ser replicado, foi pesquisado um microcontrolador que possui hardware aberto, para que se possa criar facilmente sua versão *standalone* na placa de circuito impresso do protótipo (ARDUINO UNO SMD, 2016). Além disso, buscou-se chips que possuem a versão SMD entre os seus invólucros, devido a necessidade de miniaturização do protótipo.

Sendo assim, reduziram-se os candidatos a um grupo pequeno. Sendo eles a família do Arduino, fazendo uso do microcontrolador ATMEGA328 que equipa as placas Arduino UNO e Arduino Nano. Os ESP, têm-se o ESP8266 e seu irmão mais velho ESP32 e o MSP430 da Texas Instruments.

Todas essas opções supracitadas podem ser programadas utilizando o framework do Arduino, logo a linguagem deles é comum e de fácil entendimento e além disso, a configuração do microcontrolador é abstraída pelo framework, de modo a agilizar a prototipagem. Ainda mais, essas opções são de hardware aberto ou de fácil implementação *standalone*.

Os microcontroladores PIC não foram incluídos nessa lista pois não possuem uma rapidez na prototipagem desejada. Contudo, são muito baratos, robustos e algumas versões são *Ultra Low Power*.

Dentre as opções acima, foi escolhido ATMEGA328P por ser o que mais atende os requisitos, além de possuir uma comunidade muito ativa e sólida, o que torna muito fácil a busca por soluções de problemas que possam ocorrer ao longo do desenvolvimento do projeto. O ESP32 e o ESP8266, apesar de serem recentes, também possuem uma comunidade ativa e muito suporte na internet, contudo, foi considerado um desperdício a utilização desses dispositivos, uma vez que não será realizado comunicações via WiFi ou Bluetooth.

Dessa forma, para montagem do protótipo inicial foi utilizado a placa de prototipagem Arduino UNO, que será explanada no tópico seguinte. Mais precisamente, optou-se pela placa que possui o chip SMD referenciado como ATMEGA328-AU que possui um encapsulamento TQFP 32 ocupando bem menos área que a versão *Through-Hole*.

4.1 COMPONENTES E PROTOCOLO

Inserindo o ATMEGA328-AU em uma placa embarcada com suporte de entrada e saída embutidos, e linguagem de programação simples, essencialmente C/C++ com algumas modificações, foi concebido o Arduino. Projetado em 2005 na Itália, tem código aberto e algumas funcionalidades que o tornam um ótimo início para os estudos e elaboração do protótipo previsto.

O ponto de destaque deste sistema embarcado é a capacidade de implementar acessórios inteligentes, conhecidos como *shields* ou módulos, que diminuem o tempo entre a concepção e a execução do projeto, podendo interagir com o ambiente, por meio de sensores e atuadores, sendo uma ótima ferramenta para prototipar objetos nesse universo *IoT*.

Essa placa de prototipagem possui além do microcontrolador e os componentes essenciais para seu funcionamento, uma série de outros componentes que permitem a programação do chip através de uma conexão USB, sua proteção e conectores que facilitam o acesso rápido aos seus pinos I/O.

No protótipo final, vários desses componentes serão removidos, restando apenas o essencial para o funcionamento do microcontrolador em conjunto com os outros chips do sistema. Ou seja, para programar o chip, será necessário o uso de uma placa programadora externa desenvolvida e mostrada no capítulo 6 (tópico 6.1). Isso reduz bastante o espaço na placa de circuito impresso do protótipo.

Na comunicação com os veículos, será utilizado um conector comumente chamado de porta OBD II. Contudo, no que diz respeito ao protocolo de comunicação adotado no projeto, é necessário entender a distinção entre OBD e CAN.

Em eletrônica, sinais são enviados de um chip para outro utilizando fios. Uma maneira simples de se implementar isso é utilizando um fio para cada bit de informação que se queira transmitir. Um bit de informação só pode conter dois valores, que em computação são referidos como 0 ou 1. Na maioria dos chips atuais (que utilizam MOSFETs), 0 e 1 são equivalentes a níveis de tensão, por exemplo, 0 equivale a 0 Volts e 1 equivale a +3,3 Volts.

Portanto, um bit de informação é como responder uma pergunta com apenas sim ou não, por exemplo: “O farol baixo está ligado? ” Se estiver ligado, terá +3,3V naquele fio. Se estiver desligado, terá 0 V no fio.

Bom, essa abordagem é chamada de comunicação paralela e é perfeita para utilizar quando o sistema possui um ou poucos bits de informação. Porém, mais dados requerem mais fios. Infelizmente, uma maior quantidade de fios equivale a uma maior complexidade. Um carro moderno, possui centenas de sensores e atuadores, logo, adotando o sistema de um fio para cada bit de informação significaria centenas de fios, o que acrescentaria peso e custo de fabricação. É objetivo das montadoras evitar esse tipo de custo, sendo necessária uma outra maneira de transmitir os dados do carro entre seus chips.

Um meio de resolver esse problema é utilizando comunicação serial (AXELSON, 2000) em contradição a comunicação paralela abordada anteriormente. Esse tipo de transmissão adota um sistema onde vários bits são enfileirados em um mesmo fio, portanto se quisermos enviar 100 informações, basta enviar 100 bits um após o outro, utilizando apenas um fio. Solução perfeita para comunicação entre dois chips, o ponto negativo é só o aumento da complexidade, pois agora é necessário interpretar um bloco de 100 bits para extrair a informação necessária.

Na verdade, interpretar 100 bits individuais seria uma tarefa bastante dispendiosa para um chip, o que acontece na realidade é a utilização de combinações de bits para cada informação que se queira enviar. Por exemplo, com uma combinação de 2 bits, conseguimos enviar 4 informações. Combinando 3 bits, conseguimos enviar 8 informações e, com 8 bits, 256 informações. A tabela 01 mostra um exemplo de como poderíamos utilizar um sistema com 2 bits para comunicação entre os chips que controlam os faróis do carro.

Tabela 1 - Exemplo de sistema serial com dois bits

Bit A	Bit B	Informação
0	0	Farol alto Desligado
0	1	Farol alto Ligado
1	0	Farol baixo Desligado
1	1	Farol baixo Ligado

Essa tabela representa um sistema rústico serial assíncrono de 2 bits. Na verdade, ainda seriam necessários mais bits para indicar o início da transmissão, final da transmissão e bits de verificação. Mas o importante é que, se economiza um fio em relação ao sistema paralelo, pois são necessárias 2 combinações para indicar o estado de cada farol.

Claro que, utilizar 2 bits não justifica o uso de transmissão serial, mas isso muda quando se utiliza 8 bits, onde seriam necessários 128 fios em um sistema paralelo para transmitir a mesma quantia de informações que 1 fio de um sistema serial.

Em um carro, além de transmitir uma massiva quantidade de dados, há diversos chips. O exemplo acima, mostra a comunicação assíncrona entre apenas dois chips. Porém, para que um chip transmissor comunique com dois chips receptores, por exemplo, para requisitar o valor do sensor de temperatura do chip do motor e o valor do sensor de nível de combustível do chip do tanque, pode-se adotar mais um fio, de endereçamento.

Nesse caso, com o fio de endereçamento em valor baixo (0V) há uma comunicação com o chip do motor e com valor alto (+3,3V) a comunicação se dá com o chip do tanque. Contudo, há uma maneira mais eficiente de comunicar com vários chips utilizando sistema serial. Basta adicionar mais bits de endereçamento no próprio fio de transmissão de dados. Seguindo a mesma lógica, com 3 bits poderíamos comunicar com 8 chips e 8 bits com 256 chips.

Essa maneira estruturada dos bits, facilita muito a programação e criação de bibliotecas para envio e recebimento de dados entre vários chips. Cada padrão de comunicação faz uso de uma estrutura diferente.

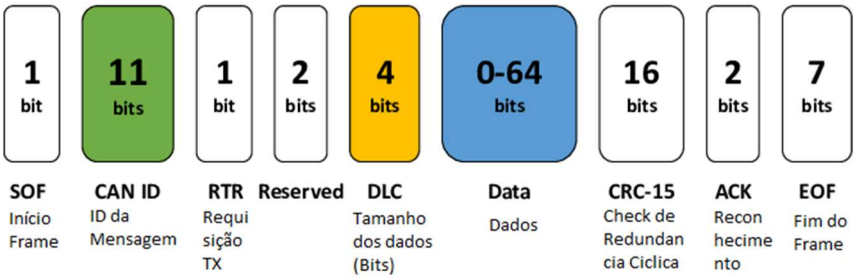
Portanto, utilizando bits enfileirados, divididos em blocos, onde alguns significam o início da transmissão, outros a informação, outros o endereçamento, pareamento (verificação), fim de transmissão e diversas outras técnicas, pode-se enviar muitas informações, entre vários chips, utilizando apenas um fio, que chamamos de barramento.

Existem outros conceitos como transmissão síncrona, *half duplex* e *full duplex* em sistemas seriais, que não serão abordados aqui pois estão fora do escopo do projeto. Nesses conceitos utilizam-se mais de um fio para comunicação serial.

O padrão CAN (*Controller Area Network*) é um tipo de barramento muito utilizado na comunicação entre chips de diversos veículos e montadoras atuais (SALUNKHE, KAMBLE E JADHAV, 2016). É robusto e simples de implementar. Ele utiliza dois fios para comunicação, pois emprega uma técnica para diminuir ruídos causados pelo ambiente externo, que é comum em veículos, principalmente próximo ao sistema de ignição das velas. Essa técnica é chamada de par diferencial, muito utilizada na eletrônica atual.

Sendo assim CAN é um padrão de barramento que define o hardware: Quantidade de fios, a quantidade de bits e como os blocos de bits estão organizados (chamados de *frames*). Por exemplo, no *frame* do padrão CAN o primeiro bit significa o início da transmissão, os 11 bits subsequentes significam o identificador do dispositivo (endereçamento) e assim sucessivamente (DE SOUZA; CAMPOS, 2017), conforme figura 4.

Figura 4 - Frame do Padrão de barramento CAN



Fonte: PESÉ, M. et al., 2019

Por outro lado, OBD (*On-Board Diagnostic*) é um termo utilizado no meio automobilístico no que diz respeito à capacidade que os sistemas do carro têm de supervisionar os subsistemas e caso algum deles falhe, conseguir se diagnosticar automaticamente. Aos poucos, esse termo acabou virando um padrão, regulado pela SAE, Sociedade de Engenheiros Automotivos dos Estados Unidos.

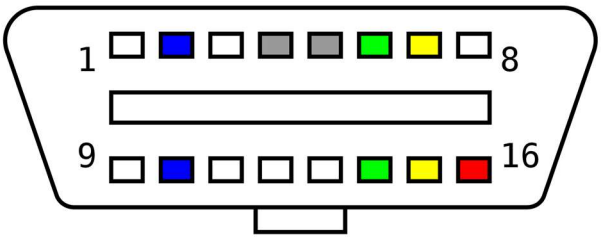
No início o OBD apenas acendia luzes no painel, mostrando que havia defeito, sem informar a natureza dele. Porém, atualmente o sistema é muito complexo, mostrando dados

em tempo real e usando um padrão de códigos que informa com precisão a natureza do defeito (DTC - *Diagnostic Trouble Code*).

O padrão OBD, dentre outras funções, define a interface com a qual outros dispositivos podem trocar informações com o sistema do carro e quais os protocolos que são aceitos.

Por exemplo, a interface padrão que é utilizada pelas montadoras a partir de 2008 é o conector OBD II, introduzido pela norma SAE J1962 (STANDARD, 2003), mostrado na figura 5.

Figura 5 - Conector OBD II



Fonte: ON-BOARD Diagnostics

Essa interface possui 16 pinos e suporta diversos padrões de barramentos, um deles, é o padrão CAN supramencionado, que faz uso de apenas dois pinos, representados pela cor verde na figura 5.

Além da interface, o OBD II também engloba os protocolos utilizados, que seria como a língua de comunicação do sistema. A estrutura é definida pelo padrão CAN (quantidade de bits, ordem que os blocos de bits são transmitidos, etc), mas a linguagem é definida pelo protocolo OBD II.

Em outras palavras, a estrutura mostrada na figura 4 é fixa e não pode ser alterada. Contudo, os bits de dados, mostrados na figura 4, podem ter até 64 bits, pode ser alterada conforme o protocolo de comunicação utilizado. No caso, a norma em vigor é o protocolo OBD II (STANDARD, 2002). Esse protocolo, define a maneira como os dados estão organizados, algo parecido com a tabela 01.

O SAE J1979 está organizado em Serviços e PID (STANDARD, 2002). Por exemplo, para requisitar a velocidade instantânea do veículo, precisamos enviar como dados (excluindo os bits de pareamento, início e fim de transmissão, etc): 0111 1101 1111 0010 0001 1101. Onde PID é o último conjunto de bits e Serviço é o penúltimo.

Contudo, não é necessário se preocupar com nível binário, uma vez que o protocolo já se encarregou de organizar isso de maneira estruturada. Sendo assim, o programador fará uso de um sistema legível para humanos, o hexadecimal. Logo, a informação acima fica:

0x7DF 0x02 0x01 0x0D, onde PID é 0x0D e Serviço é 0x01. Esses valores hexadecimais estão tabelados conforme o protocolo indica (STANDARD, 2002).

Por possuir total compatibilidade com plataforma Arduino e ainda comunicação por interface SPI, escolheu-se o módulo mostrado na figura 2. Esse módulo possui dois chips, o MCP2515 e o TJA1050. Ambos os chips são fáceis de encontrar no mercado e possuem documentação bem escrita e difundida.

O MCP2515 é um controlador CAN que possui interface SPI. Logo, é capaz de comunicar com o microcontrolador (usando SPI) e "traduzir" para o *frame* do padrão CAN. Contudo, sua saída não é capaz de se conectar diretamente a um barramento CAN, para isso é necessário um chip *transceiver*, o TJA1050. Assim como o chip ATMEGA328-AU, escolheu-se os componentes SMD dos chips MCP2515 e TJA1050, para economizar espaço na placa de circuito impresso.

Por fim, para comunicação com servidores e requisição de dados do GPS, utilizamos o módulo que contém o chip SIM808 (Figura 6). Esse chip é de fácil implementação e muito difundido na comunidade *maker*. Existem várias placas de prototipagem, dentre elas utilizamos a EVB Versão 3.2 por possuir uma excelente documentação.

Dentre os chips do projeto, esse é o que possui maior tamanho. Encontramos outros como GE866 QUAD, ou UE866 que possui dimensões bem menores em comparação com o SIM808, contudo, necessitam de um módulo externo para GPS.

O maior desafio para utilização desse chip é a implementação do circuito de regulagem de tensão, pois durante o funcionamento, o chip pode consumir, momentaneamente até 2A de corrente.

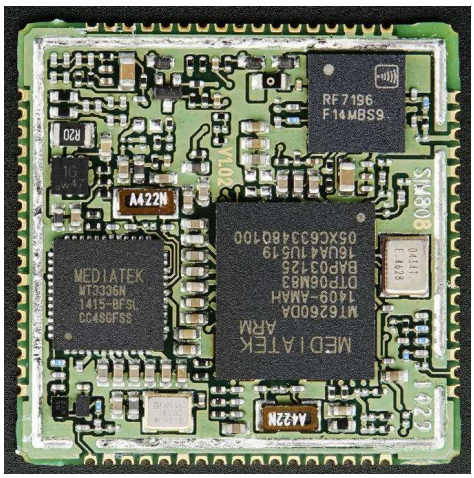
Figura 6- Placa SIM808 com conjunto de chips

Fonte: FRIED, 2015

Na verdade, o SIM808 é uma placa com conjunto de chips, como mostrado na figura 7 (FRIED, 2015). O microcontrolador MT6206, o chip de GPS MT3336 e o chip de GSM RF7169. Esses chips, com exceção do MT6206, são de Radiofrequência, logo, é necessário tomar os devidos cuidados que sinais de RF necessitam, como: Espessura e largura correta

da trilha (para um bom casamento de impedância), escolha correta do material dielétrico da placa, posicionamento (*floorplanning*) correto para evitar interferências e cuidadoso roteamento das trilhas.

Figura 7 - Placa SIM808 com conjunto de chips visão "interna"



Fonte: FRIED, 2015

Portanto, é usual que empresas já tomem esses cuidados e criem uma placa-chip que pode ser soldada em uma placa de circuito impresso convencional. Os microcontroladores ESP8266 e ESP32 seguem essa técnica, pois também possuem sinais RF de WiFi e Bluetooth.

Por outro lado, o SIM808 comunica com o microcontrolador utilizando comunicação serial, simplificando a 2 fios a transmissão dos dados para o servidor. Nesse caso, pode-se utilizar um serial por software, para evitar de usar o serial hardware (pinos exclusivamente utilizados para a comunicação UART).

Durante o desenvolvimento do sistema, achou-se necessário adicionar um adaptador para cartão de memória com intuito de armazenar localmente em um buffer, os dados obtidos do GPS e do carro que não puderem ser enviados ao servidor devido a falta de conexão, algo que acontece em alguns pontos remotos da cidade.

Por fim, a placa de circuito impresso terá o SIM808 para comunicações com GPS e servidores (através do GSM). MCP2515 e TJA1050 para comunicação com barramento CAN. Um adaptador para cartão de memória e um adaptador para cartão SIM (para ser utilizado em conjunto com o SIM808).

4.2 DIMENSIONAMENTO DA SOLUÇÃO

Para desenvolver os passos citados no capítulo 3, foi utilizado um software de projeto eletrônico que, dentre outras funções, é capaz de desenhar esquemático, *layout* de placa (trilhas e componentes) e desenho 3D.

Uma vez que o protótipo foi dividido em duas placas de circuito impresso, foi feito o mesmo com relação ao esquemático, sendo assim, o anexo B contém o esquemático da placa principal e o anexo C o esquemático da placa de potência.

A solução proposta é bastante simples, apenas a comunicação entre os diversos chips através de SPI e comunicação serial, sendo necessário alguns periféricos para o funcionamento dos CIs como capacitores, resistores e cristais de quartz. O microcontrolador ATMEGA328P é referenciado como U11 no esquemático. Esse componente controla a comunicação entre o SIM808 (U41), cartão de memória (J11) e o MCP2515 (U21).

Tanto o cartão de memória, quanto o MCP2515 compartilham do *bus* de interface de comunicação SPI (Serial Peripheral Interface). Logo, pode ser observado que os pinos 17, 16, 15 do microcontrolador, que são SCK, MISO e MOSI respectivamente, estão conectados tanto no MCP2515 quanto no cartão de memória. Os pinos 14 e 13, chamados de CS (chip select) definem com qual periférico o *bus* irá trafegar dados, sendo necessário manter o nível lógico alto quando não há comunicação e o nível lógico baixo para se comunicar com o periférico. Ou seja, antes de iniciar a comunicação com o carro, é necessário colocar o pino 14 em nível lógico baixo.

Por outro lado, para se comunicar com o SIM808, é necessário apenas o uso dos pinos 9 e 10 do ATMEGA328, RX e TX respectivamente, se configurando uma comunicação serial. Nesse caso, como os dois CIs possuem níveis lógicos diferentes (5V para o ATMEGA328 e 3,3V para o SIM808) foi necessário o uso de um conversor de nível lógico bidirecional, referenciado como U42 (TXB0104). O mesmo conversor foi utilizado com o cartão de memória, o U12 no esquemático.

No esquemático, todos os componentes referenciados com a letra P, são conectores, com exceção dos conectores SMA de antena do GPS e GSM. De maneira análoga, o dispositivo J41 é um suporte para cartão SIM, que é o chip de dados. Como o chip pode ser colocado e retirado com o circuito ligado, optou-se por usar um CI de proteção contra descargas eletrostáticas (U43), que pode ser causada pela mão humana durante o manuseio do cartão SIM.

Por fim, o transistor Q11 é uma maneira de reiniciar o SIM808 através do ATMEGA328P, isso é muito útil para o caso de o SIM808 deixar de responder os comandos do microcontrolador. Além disso, foi colocado LEDs para indicação de funcionamento e um botão para reiniciar o sistema manualmente em caso de falha.

Foi utilizado capacitores para estabilizar a tensão nos circuitos integrados, tomando o cuidado para colocar esses capacitores os mais próximos possíveis do pino de VDD dos CIs. Além disso, tanto o microcontrolador quanto o MCP2515 necessitam de um cristal para referência de frequência.

A placa de potência é bem simples e contém apenas reguladores de tensão, referenciados como VR31, VR32 e VR33 no anexo C. São três reguladores, um para cada nível de tensão. O VR31 possui o nível de 5V na sua saída, para alimentar o microcontrolador e os chips MCP2515 e o MCP2551 (equivalente ao TJA1050). O VR32 possui o nível 3,3V e o VR33 possui saída de 3,7V.

A entrada do sistema é alimentada com 12V provenientes do conector OBD2. O maior desafio foi o projeto do VR33 (LM2596). Foi utilizado a configuração de saída ajustável, sendo necessário o uso de um indutor (L31) cujo o valor é indicado no *datasheet* (TEXAS INSTRUMENTS, 2020), sendo esse valor dependente do valor da corrente máxima da carga. Além disso, o valor da tensão de saída é proporcional ao valor de R_{32}, calculado através da equação abaixo, retirada do *datasheet*.

$$R_{32} = R_{31}\left(\frac{V_{OUT}}{V_{REF}} - 1\right)$$

É recomendado que R_{31} tenha um valor fixo de um $1k\Omega$ com precisão de 1%. V_{OUT} é a tensão de saída, que é 3,7V e V_{REF} é definido no datasheet com o valor de 1,23V, logo, o valor de R_{32} é de aproximadamente $2{,}2k\Omega$.

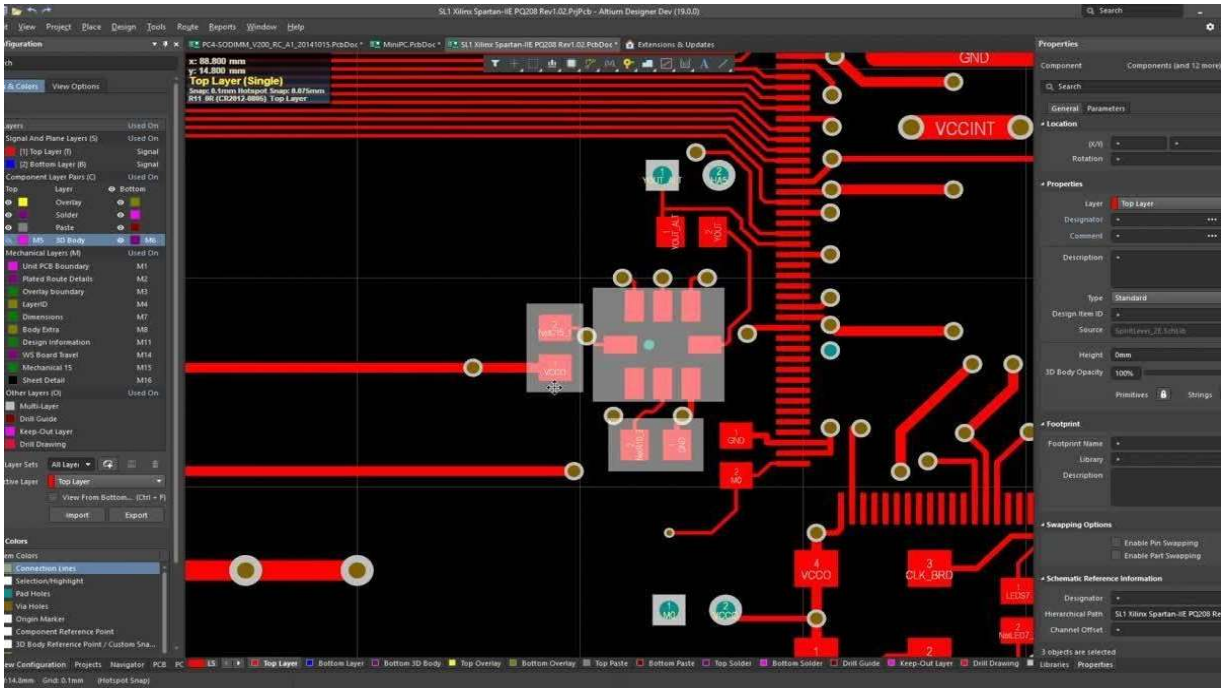
Após a finalização do esquemático, é necessário o desenho do layout. Por se tratar de circuitos digitais, não foi necessário realizar simulações. A Figura 8 mostra o *layout* sendo desenhando no *software* Altium Designer. Os anexos D e E mostram o resultado do posicionamento dos componentes e tamanho da placa de circuito impresso.

O posicionamento foi realizado de forma a manter o mais próximo possível os componentes, mantendo os sinais analógicos (antenas de GPS e GSM) distantes dos sinais digitais que possuem muito ruído. Além disso, tomou-se o cuidado de manter o casamento de impedância das trilhas das antenas, fazendo com que essas possuíssem uma resistência próxima a 50Ω. Para tal, utilizou-se uma ferramenta do próprio Altium Designer para fazer

esse casamento, bastando apenas inserir o valor da espessura da trilha de cobre que no caso foi de 1 onça (1 oz.).

Depois de posicionar os componentes e fazer o roteamento das trilhas, foi exportado um desenho em 3D do protótipo como mostrado nas figuras 9 e 10. Para lá de mostrar esteticamente o resultado, esse modelo em 3D é utilizado para a fabricação de invólucros de plástico para proteger o circuito.

Figura 8 - Desenho de *layout* no software Altium Designer 2018

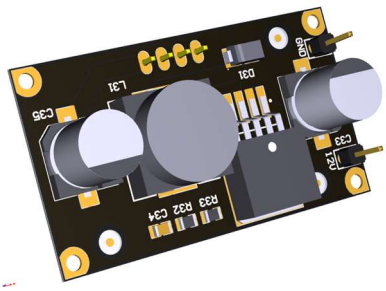


Fonte: Dados do autor

Figura 9 - Esquema 3D da placa de processamento

Fonte: Dados do autor

Figura 10 - Esquema 3D da placa de Potência



Fonte: Dados do autor

O item 6.1 detalha mais profundamente o uso e a capacidade desse software no projeto do controle de frota.

O desenvolvimento do firmware foi feito em linguagem C/C++ utilizando a plataforma Arduino. O resultado é apresentado no anexo F.

São utilizadas as bibliotecas Arduino.h, mcp_can.h, SPI.h e SoftwareSerial.h. A biblioteca Arduino.h contém todas as definições de pinos, frequência de operação, configuração do ATMEGA328P de modo a acelerar o processo de programação na plataforma.

Dessa forma, ao invés de todo o código estar em uma função principal chamada de *main* como é o costume da programação em C, a plataforma Arduino faz uso de duas funções, a *setup* e a *loop*. Na função *setup*, que é executada apenas uma vez, contém toda a configuração do microcontrolador e de seus periféricos. Por outro lado, a função *loop*, é executada em um ciclo infinito, até que a energia seja retirada do microcontolador.

A biblioteca mcp_can.h contém tudo relacionado ao protocolo de comunicação com o carro. Dessa forma, basta utilizar comandos simples e a própria biblioteca realiza a conversão dos dados para o barramento CAN na padronização correta.

SPI.h e SoftwareSerial.h são bibliotecas utilizadas para a comunicação serial. A primeira utiliza os pinos MISO, MOSI, SCK e CS, explanados anteriormente e a segunda transforma qualquer pino do microcontrolador em TX e RX da comunicação serial, livrando, dessa forma, o serial de hardware para a comunicação UART com um computador para mostrar *logs* de funcionamento.

Além das bibliotecas, foram criadas algumas funções, para tornar o código mais organizado e de mais fácil leitura. Essas funções são definidas antes da função *setup*, porém seu escopo é apresentado no final, depois da função *loop*.

Os pontos a serem destacados se encontram nas funções *setup* e *loop*. A função *setup* é executada apenas na inicialização do sistema e possui toda configuração do sistema, tanto do MCP2515 quanto do SIM808. Além disso, é nessa função que são configurados os pinos do microcontrolador. Observe que essa função possui a configuração de comunicação serial que durante os testes é utilizada para mostrar na tela de um computador os *logs* do sistema.

Além disso, os filtros do MCP2515 e inicialização dos módulos de GPS e GPRS do SIM808 são executados nessa função. Quando tudo se inicializa corretamente, é apresentado uma mensagem: "Programa de inicialização completo: MCP2515+GPS+GRRS". Caso contrário é apresentado uma mensagem de erro.

Por outro lado, a função *loop* executa infinitamente até a energia ser removida do sistema. Essa função executa em ordem os comandos para:

- Requisitar os dados do veículo;
- Armazenar em uma *string*;
- Requisitar a localização;
- Concatenar na *string* criada anteriormente;
- Enviar essa *string* para o servidor.

As outras funções foram criadas apenas para auxiliar nos comandos mostrados acima.

4.3 EQUIPAMENTO SIMILAR ENCONTRADO NO MERCADO

Existe atualmente um dispositivo capaz de gerenciar frotas de veículos através do recolhimento de dados. A empresa Cobli emprega um equipamento que faz uso do protocolo CAN para recolher dados como velocidade, odômetro, RPM, consumo de combustível e outros. De maneira análoga, esse aparelho possui um localizador GPS e comunicação na nuvem para envio dos dados recolhidos, porém é necessário pagar taxa de adesão de R$30,00 e mais mensalidade de cerca de R$100,00 por veículo.

A vantagem da solução proposta é não ter um valor mensal, sendo necessário apenas comprar o produto que tem custo de cerca de R$170,00 por unidade conforme mostrado na Tabela A.1 do anexo A. Além disso, o software é desenvolvido de maneira a ser totalmente personalizado para o Tribunal de Justiça, sendo utilizado um servidor local, mantendo a segurança das informações contidas, sem terceiros tendo acesso a esses dados.

5 DESENVOLVIMENTO DA PLATAFORMA

A plataforma compreende o banco de dados e a aplicação Web, comumente dividida em o que é chamado de *back-end* e *front-end*. Basicamente o *back-end* contém tanto o sistema embarcado (placa de prototipagem) quanto o banco de dados. Já o *front-end* contém a aplicação web ou app de smartphones, que essencialmente é a ferramenta com o qual o usuário irá interagir.

5.1 BANCO DE DADOS FIREBASE

O Firebase é uma plataforma de *back-end* criada para desenvolvimento mobile e web e que foi adquirida pela Google em 2014. Essa plataforma possui diversos produtos relacionados à infraestrutura de uma aplicação web/*mobile* como:

- Hospedagem;
- Armazenamento nas nuvens;
- Banco de dados em tempo real;
- Sistema de autenticação;
- Monitoramento de desempenho;
- Integração com sistema de anúncios do Google e muito mais.

Dessa forma o desenvolvedor fica mais livre para focar no melhoramento da sua aplicação e "terceiriza" a preocupação com segurança, estabilidade, escalabilidade e infraestrutura em geral.

Outra vantagem do Firebase é a sua gratuidade para testes a nível de protótipo. Dessa forma, para o estudo de viabilidade da aplicação não é necessário nenhum investimento de tempo (para a criação dessa infraestrutura) nem em dinheiro (caso queira terceirizar esse processo). Além disso, uma vez que a aplicação tem sua viabilidade comprovada, basta contratar um plano mensal para continuar utilizando os serviços de maneira mais escalável, sem a necessidade de utilizar um sistema para teste e depois desenvolver um outro sistema para a aplicação real (tornando muito rápido a transição de protótipo para produto final).

Obviamente, por ser um sistema robusto e seguro, ele se torna confiável para conter informações delicadas como os dados dos veículos de uma frota (localização, quilômetros rodados, etc).

Dentre os produtos do Firebase, por hora será utilizado apenas o sistema de autenticação, hospedagem, armazenamento nas nuvens e o banco de dados em tempo real.

Basicamente, o sistema de autenticação é uma maneira segura de certificar quem ou o que tem acesso aos dados do banco de dados ou acesso à aplicação web. Por exemplo, pode-se utilizar sistema de acesso de verificação em dois passos, utilizar contas do Google, Facebook ou Github, além de permitir uma configuração de mesclagem de contas de maneira bem rápida (Sistemas complexos de autenticação podem levar meses para ficarem prontos, além de ser necessário a contratação de mão de obra especializada).

O sistema de hospedagem torna fácil a implementação da aplicação web, podendo ser desenvolvida localmente utilizando alguma plataforma como Angular (ou similar), e fazendo a implantação direto na plataforma. Caso necessite de um domínio específico do seu projeto, basta adicionar seu domínio às configurações e confirmar a propriedade do domínio.

O armazenamento diz respeito ao tamanho disponível nos servidores do Firebase para o armazenamento do banco de dados, imagens, arquivos da aplicação web e mais. Se for necessário mais espaço, é sempre possível fazer um upgrade no plano contratado. Na versão grátis, têm-se 1GB de espaço com um limite de 10GB de download mensal.

Por fim, o banco de dados em tempo real do Firebase é um banco NoSQL que significa "*not only SQL*" fazendo alusão ao fato de que não faz uso apenas da linguagem SQL (*Structured Query Language*) de banco de dados.

O *Firebase Realtime Database* (Banco de dados em tempo real do Firebase) é um banco de dados hospedado na nuvem. Os dados são armazenados no formato JSON e sincronizados em tempo real com todos os clientes conectados. Os arquivos JSON (*JavaScript Object Notation*) são amplamente utilizados em aplicações web e mobile, sendo assim, esses bancos são facilmente acessados por diversos tipos de API (*Application Programming Interface*) como:

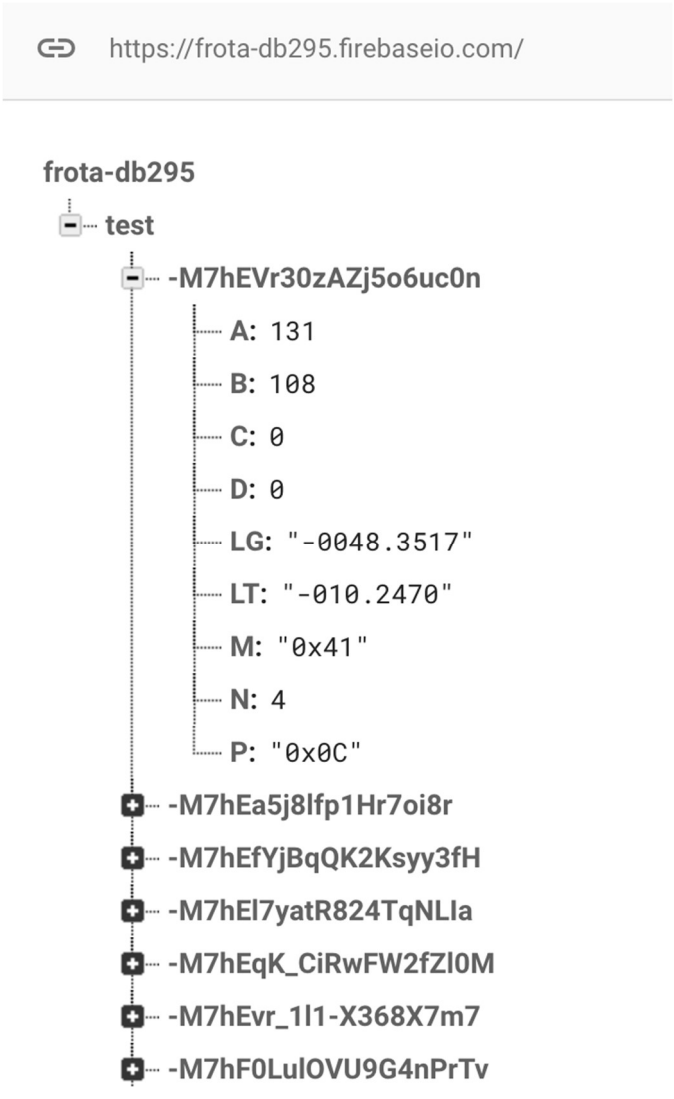
- Apps iOS;
- Apps Android;
- API REST (Representational State Transfer);
- Aplicações Web (que fazem uso de JavaScript);

O protótipo desenvolvido tem capacidade de comunicar com o banco de dados através da API REST fazendo uso de requests GET, PUT e POST através do protocolo HTTPS. A maneira com que o *front-end* se comunica com o banco de dados é explicada no tópico a seguir.

Em relação à estrutura do banco de dados do firebase, ele é organizado em uma árvore JSON, ao contrário de um banco de dados SQL, não há tabelas nem registros. Portanto, para acessar um dado é necessário passar o caminho do "galho" correspondente dessa árvore.

Inicialmente se estrutura o banco de dados para obter um histórico de posição GPS e o valor RPM do veículo, como mostrado na Figura 11. Posteriormente, pode-se adicionar mais campos, como dados de odômetro, nível do tanque e outros na medida em que for necessário.

Figura 11 - Estrutura do Banco de Dados



Fonte: Dados do autor

Observe que em cima temos o endereço do banco de dados, embaixo temos o nó “test” e dentro dele vários nós cada um com um código. Esse código é gerado automaticamente quando fazemos um *POST request*. Dessa forma ele adiciona um nó ao invés de substituir os valores existentes naquele nó, ou seja, armazena vários dados enfileirados um após o outro, ao invés de apenas trocar o valor anterior por um novo, criando um histórico de dados que pode ser acessado pelo front-end.

Dentro dos nós com os códigos temos as informações A, B, C, D, LG, LT, M, N e P, que são respectivamente: Primeiro byte de dados, segundo byte de dados, terceiro byte de dados, quarto byte de dados, longitude, latitude, modo de operação, número de bytes com informações e parâmetro ID.

O motivo de escolher siglas ao invés do uso de palavras legíveis por humanos é a escalabilidade, uma vez que os dados são transmitidos no formado JSON e cada letra, número, caracter faz uso de um byte. Uma frota com mais de 100 veículos poderia causar lentidão inesperada no servidor.

De acordo com a figura 12 temos que o primeiro byte contém a informação 131, o segundo 108, que a longitude é -48.3517 e a latitude é -10.2470, o modo é 0x41 (modo de resposta), tem 4 bytes de informação e o PID é 0x0C (RPM).

Analisando a tabela de PID, temos que para RPM a fórmula é (256*A+B)/4, ou seja, no instante em que essa informação foi enviada ao banco de dados, o carro estava com 8411 RPM.

Observe que o pré-processamento dessa informação (RPM) poderia ser feito localmente pelo protótipo, sendo necessário transmitir apenas o valor final. Contudo, se mostrou mais viável processar essa informação no *front-end* e deixar os recursos do sistema embarcado mais ocupados com a requisição e transmissão dos dados para o banco de dados. Além disso, essa estrutura tornou possível que a configuração de quais informações requisitar do veículo seja feita pelo *front-end*, eliminando a necessidade de requisitar centenas de informações desnecessárias e focar apenas nas mais úteis para o cliente.

Contudo, a eficiência reduz, sendo necessário um POST para cada informação desejada, ou seja, se quiser saber qual a velocidade e o RPM do carro, são necessárias dois POSTs. Se quiser saber a velocidade, RPM e temperatura do ar de admissão, são necessários três POSTs e assim sucessivamente.

5.2 APLICAÇÃO WEB

Como mencionado anteriormente, a interface em que o usuário irá interagir é comumente chamada de *front-end*. Pode ser tanto uma aplicação web, como o Gmail, em que você faz login e lê seus e-mails através de um navegador. Pode ser também um app para ser acessado pelos smartphones.

Escolheu-se aplicação web devido a rapidez e facilidade em ser desenvolvida. Apesar de que, hoje em dia, o desenvolvimento de apps está bastante ágil também, utilizando tecnologias como Flutter e React, é possível desenvolver apenas um app para as duas plataformas (iOS e Android). Em um trabalho futuro, o app será desenvolvido para uma interação mais fluida com o usuário.

Aplicações web fazem uso de HTML e CSS que são uma linguagem de marcação e uma linguagem de estética, respectivamente. A maioria dos sites hoje em dia fazem uso do HTML e CSS.

Basicamente com o HTML você pode criar estruturas da página, como Título, parágrafo, botões, tabelas, campos de inserção de dados e vários outros. E com CSS é possível estilizar essas estruturas, mudando a cor da fonte, tamanho, tipo, posicionamento na tela, cor que o botão exibe entre outros.

Contudo, apenas HTML e CSS não é capaz de executar funções mais complexas a não ser exibir na tela um padrão pré-definido, o que tornaria os sites estáticos e nada dinâmicos. Para o caso de ter que mostrar dados como um mapa mostrando a localização dos carros em tempo, *dashboards* mostrando informações como a velocidade e RPM ao clicar nos veículos, seria necessárias funções complexas capazes de pegar esses dados do servidor para mostrar ao usuário.

Para isso faz-se o uso de JavaScript que é uma linguagem de programação interpretada pelo navegador do usuário. Com essa linguagem é possível criar funções capazes de acessar o banco de dados ao clicar em um botão ou após uma autenticação de login.

Toda a aplicação, que consiste dos arquivos HTML, CSS e JS (sigla de JavaScript) pode ser carregada no firebase e ser acessada de qualquer lugar do globo. Devido ao prazo, apenas a tela de login foi desenvolvida e o esboço de uma página do usuário. Sendo assim, os testes realizados no capítulo 6 foram feitos apenas com o *back-end.*

6 RESULTADOS E DISCUSSÕES

Depois de reunir todas as informações necessárias, listadas abaixo, foi projetada uma placa de circuito impresso contendo todos os componentes citados, em conjunto com os dispositivos inerentes a proteção e alimentação do protótipo:

- Tensão de alimentação;
- Tensão de operação dos circuitos integrados;
- Restrições de roteamento;
- Regras de fabricação da placa de circuito impresso;
- Sistema de comunicação entre os chips;
- Área final da placa de circuito impresso;

Essa etapa do projeto é, depois do estudo prévio, a mais demorada e elaborada. Geralmente é executada por uma equipe de engenheiros e a negociação é ponto crucial de um projeto final que, além de agradar o cliente, não seja muito caro do ponto de vista de desenvolvimento. Dentro dos limites da física, na engenharia tudo pode ser feito, depende do valor que o cliente pode investir e qual o prazo final do desenvolvimento.

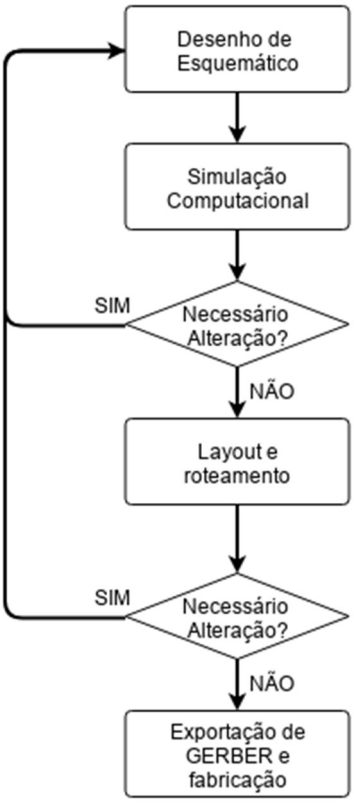
Portanto, é muito comum nessa etapa a mudança de microcontroladores e outros chips. Podendo até ocorrer a alteração de toda a arquitetura do protótipo a fim de alcançar os requisitos de área, preço e consumo de energia (principalmente em produtos que utilizam baterias).

Além disso, é muito comum que várias versões sejam desenvolvidas antes do produto final. A primeira versão é sempre uma prova de conceito e geralmente não possui todas funcionalidades, ou tem um tamanho exagerado para facilitar nos testes, ou utiliza componentes genéricos que vão ser trocados em futuras versões ou ainda vêm com erros inesperados que precisam ser corrigidos na versão final.

Em alguns casos, pode até ocorrer de uma nova tecnologia ser muito melhor e mais barata que a utilizada nas versões anteriores. Sendo assim, uma nova versão é desenvolvida para um melhoramento no produto. Esse é o caso desse protótipo. Foi utilizado o SIM808 para o desenvolvimento da versão inicial, contudo, foi encontrado no mercado um chip mais novo, com menor consumo, menor área e ainda exclui a necessidade do uso de um microcontrolador. Dessa forma, a próxima versão virá com esse chip, o A8 da AI-Thinker.

Os passos a serem seguidos estão no diagrama da figura 12.

Figura 12 - Diagrama projeto de PCB



Fonte: Dados do autor

Primeiramente é desenhado o esquemático do circuito eletrônico, que indica os componentes e como esses estão ligados entre si. Depois disso, faz-se uma simulação computacional para verificar o funcionamento do sistema em conjunto. Confere se as simulações condizem com o esperado. Caso contrário, faz-se as modificações necessárias no esquemático e executa a simulação novamente.

Em conjunto com o desenvolvimento de esquemático, se dá a criação de bibliotecas que contém a representação dos componentes na folha de projeto, bem como o desenho desses componentes na placa final, o que chamamos de *footprint*.

Depois de finalizada essa etapa, começa-se o desenho da placa de circuito impresso em si, seguindo as restrições de área. Primeiramente posiciona-se todos os componentes de uma maneira eficiente em termos de conexões, levando em conta a separação mínima necessária entre os sinais digitais e analógicos (para evitar ruídos) e a proximidade da fonte de energia (para evitar excessivas quedas de tensão). Essa etapa, chamada de *floorplanning*, deve ser muito bem elaborada, caso contrário, pode-se tornar impossível a conexão entre todos os componentes.

Depois de realizado o *floorplanning*, é necessário fazer o roteamento das trilhas. Nessa etapa, o tamanho físico final já foi definido, logo, existe a restrição de área e posicionamento dos furos para os parafusos de fixação. Portanto, é objetivo do projetista utilizar a mínima ocupação de área, ou seja, menor largura de trilha permitida, porém, levando em consideração a menor queda de tensão possível, pois uma menor largura significa maior resistência. Outro ponto importante a ser levado em consideração, são as trilhas de potência. Caso a corrente seja muito elevada, precisamos de escolher um material que possua uma espessura de trilha considerável, caso contrário, extrapolará a restrição de área.

De acordo com nossos cálculos, mesmo tendo um pico de 2A, pode-se utilizar uma placa de 1 oz de cobre e as trilhas de potência não terão áreas tão largas a ponto de causar problema no roteamento.

Depois do roteamento, alguns softwares conseguem extrair o comportamento parasita de trilhas paralelas e próximas (capacitâncias e indutâncias). Porém, no nosso caso, foi utilizado apenas as ferramentas que conferem se não há erros de projeto, como curto circuito ou larguras de trilhas inferiores ao mínimo permitido pela fabricante da placa (DRC - *Design Rule Checking*). Depois de corrigir as eventuais falhas, exportar os arquivos GERBER (formato de arquivos aceitos pela indústria de confecção de placas de circuito impresso) e os enviamos para fabricação.

Depois de fabricado, os componentes foram soldados na placa e os testes de laboratório foram realizados. Esses testes são realizados para o caso de ocorrer alguma falha, que será corrigida na segunda versão do protótipo.

6.1 PROJETO DE PLACA DE CIRCUITO IMPRESSO

O *design* do protótipo foi realizado utilizando o Altium Designer 18, um poderoso software que permite aos engenheiros realizar a maioria dos passos necessários para o desenvolvimento completo de placas PCB (*Printed Circuit Board*), desde a criação de bibliotecas (contendo as entidades de esquemático, *footprints* de placa e modelos em 3D dos componentes), passando pela facilidade de separar, de maneira organizada, o esquemático em vários arquivos (onde é possível modularizar o desenvolvimento do protótipo, utilizando metodologia ágil), renderização em 3D da placa final (para facilitar a criação de invólucros de plástico) até a facilidade de trabalhar com placas flexíveis e placas com mais de 4 camadas de cobre (ideal para projetos de alta densidade).

Começou-se o desenvolvimento através da alimentação. Os níveis de tensão dos circuitos integrados utilizados no protótipo estão listados abaixo:

- 3,3V
- 3,7V
- 5V
- 12V

O protótipo será alimentado pela tensão de 12V fornecida pela porta OBD II do carro. Não se utilizou essa tensão em nenhum circuito integrado, pois, eles trabalham com os outros níveis de tensão já mencionados. Logo, se faz o uso de reguladores para alimentar os CI's da PCB.

O nível de 3,3V foi utilizado pelo cartão de memória. Para esse nível de tensão, fez-se o uso do regulador de tensão LM1117, na configuração de tensão de saída fixa.

O nível de 3,7V é exclusivo para o SIM808. Esse circuito integrado pode consumir até 2A de corrente de pico. Sendo assim, foi necessário utilizar o regulador de tensão LM2596. Isso se tornou um desafio, pois foi necessário utilizar a configuração de tensão de saída ajustável, para gerar o valor de 3,7V.

O nível de 5V foi utilizado pelo microcontrolador ATMEGA328-AU, pelo *transceiver* MCP2551 e pelo controlador MCP2515. Para esse nível de tensão, foi empregado o regulador de tensão LM340, na configuração de tensão de saída fixa.

Observe que o *transceiver* TJA1050 foi substituído pelo MCP2551. São chips semelhantes em função e níveis de tensão de operação, portanto, em termos de funcionamento, não possui nenhuma diferença. A escolha desse segundo CI se deu pelo fato dele estar mais disponível no mercado.

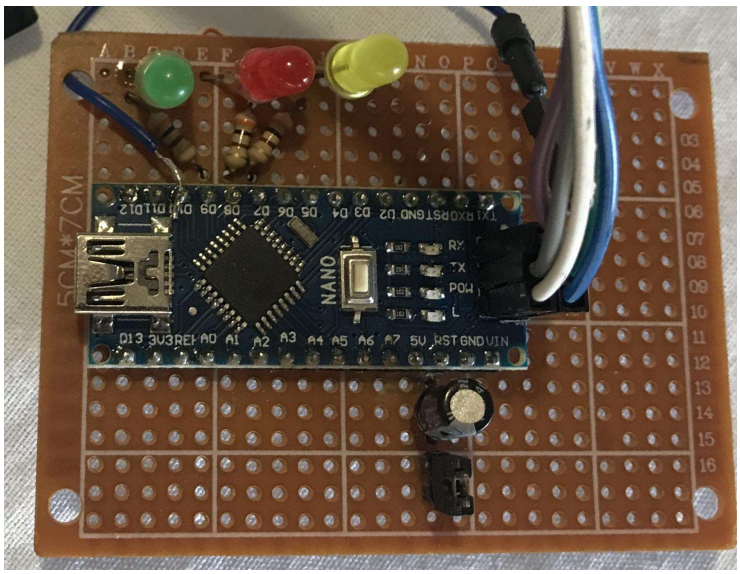
Durante o *flooplanning* e roteamento das trilhas tomou-se o cuidado para afastar o máximo possível os sinais digitais (microcontroladores, controladores, cartão de memória, chip de dados e *transceiver*) dos sinais analógicos de radiofrequência (antena de GPS e antena de GSM), para evitar ruídos e perda da qualidade do sinal.

Além disso, foi respeitado as larguras mínimas, bem como as mínimas distâncias entre trilhas exigidas pela fabricante. Para isso, foi utilizada uma ferramenta que durante o roteamento nos informa, através de erros, que as regras de fabricação estão sendo infringidas, bastando uma pré-configuração do software. Essa ferramenta é chamada de DRC, que além das regras da fabricante, checa erros como curto circuito ou trilha em aberto.

Outro ponto muito importante no desenho de um sistema é a comunicação entre os circuitos integrados, para evitar erros de transmissão de dados. É crucial para saber a disponibilidade de pinos do microcontrolador. No protótipo desenvolvido a comunicação entre o microcontrolador e o SIM808 é feita através da serial por software, utilizando dois pinos. A mesma técnica é utilizada pelo cartão de memória. Por outro lado, o microcontrolador se comunica com o controlador CAN através de interface SPI (CORPORATION, 2011) que faz uso de pinos específicos para isso.

Para gravar o código no microcontrolador também é utilizado a interface SPI. Sendo assim, foi utilizado um Arduino Nano configurado como programador (Figura 14). Não há conflito com o controlador CAN, desde que se configure o pino SS corretamente. De maneira análoga, não há conflito com o uso do serial em hardware (conhecidos como pinos 0 e 1) na comunicação com o SIM808, ou seja, pode-se utilizá-lo para a comunicação com esse chip, contudo, escolheu-se o serial em software, pois os pinos 0 e 1 podem ficar ocupados com o uso de algumas funções de *debug* dos códigos.

Figura 13 - Gravador de códigos do protótipo



Fonte: Dados do Autor

Por fim, para atender as exigências de área e posicionamento dos furos, foi necessária a realização de algumas modificações. Era esperado que o produto coubesse em uma caixinha de diagnóstico de OBD II para USB (bbfly-BF32302), com dimensões internas de 63x41x20 mm (excluindo o conector OBD II).

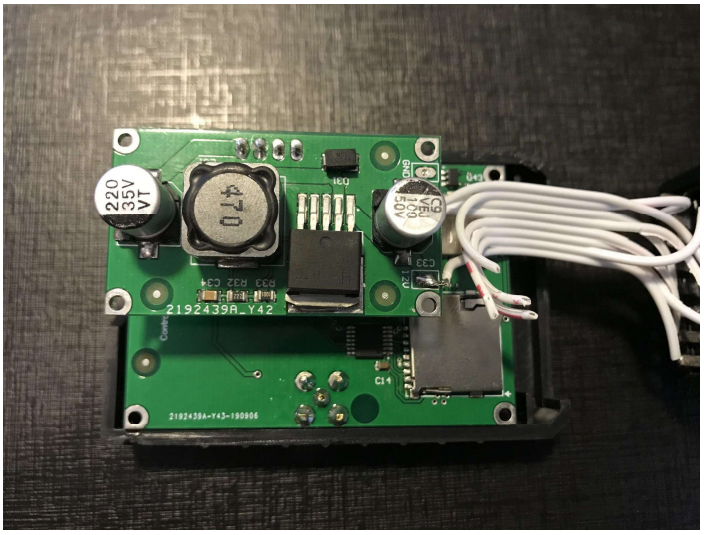
Sendo assim, o projeto foi dividido em duas placas, uma de potência e outra de processamento conectadas através de um conector *header* de 4 pinos. Na placa de

processamento, foi necessário o uso de cristais menores do que o planejado inicialmente e o posicionamento dos chips foi alterado diversas vezes.

Por fim, as dimensões de largura e comprimento foram respeitadas, porém, a dimensão de altura extrapolou os 20mm para 32mm, devido a altura dos capacitores do regulador LM2596.

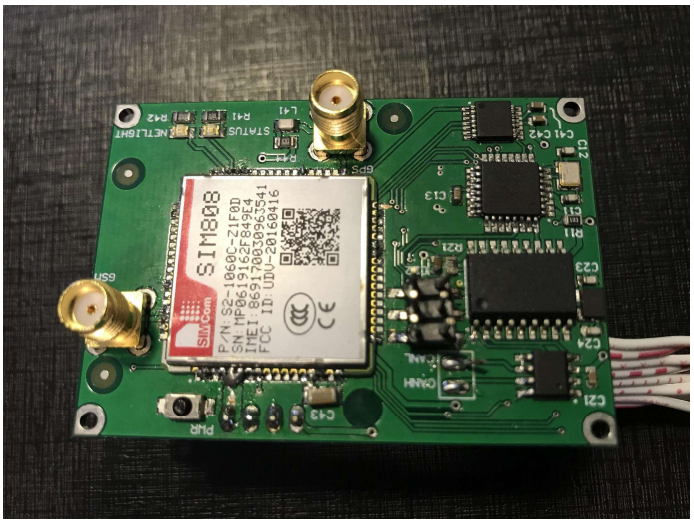
O resultado final com os componentes soldados é mostrado nas Figuras 15 a 18.

Figura 14 - Vista superior do protótipo



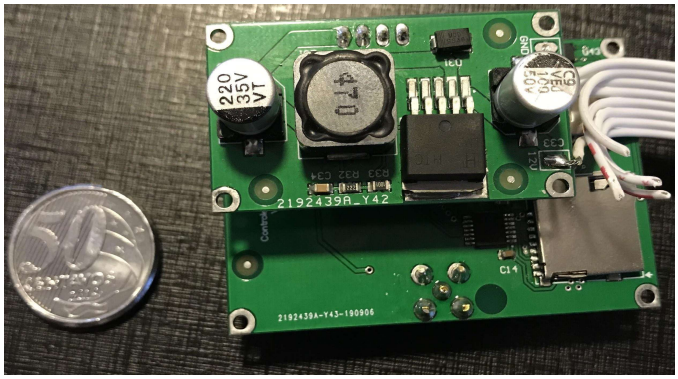
Fonte: Dados do Autor

Figura 15 - Vista inferior do protótipo



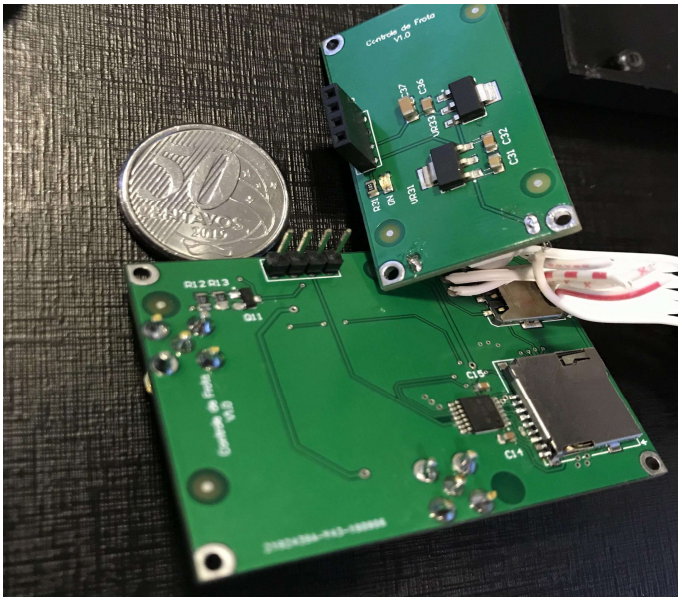
Fonte: Dados do Autor

Figura 16 - Vista superior do protótipo em perspectiva com moeda



Fonte: Dados do Autor

Figura 17 - Protótipo desmontado



Fonte: Dados do Autor

6.2 TESTES DE HARDWARE

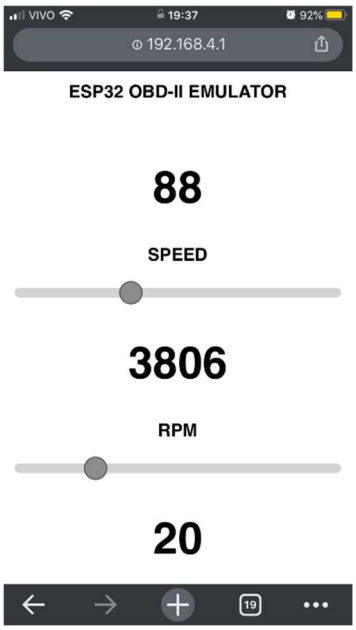
Depois de soldar os componentes foram realizados os testes de funcionamento utilizando vários códigos escritos exclusivamente para essa função. Nesse ponto, utilizamos um emulador de OBD II para poder fazer os testes em laboratório e um veículo popular para realizar os testes em campo.

6.2.1 EMULADOR DE OBD II

O emulador foi feito utilizando um microcontrolador ESP32 que possui conexão Wi-Fi e Bluetooth, além de fazer uso de um transceiver CAN MCP2551. Dessa forma, o emulador foi configurado para gerar uma conexão Wi-Fi como *Access Point* (modo AP),

logo, é possível conectar com um celular no emulador e através de uma página HTML controlar valores de RPM, Throttle (posição do pedal) e velocidade, simulando um veículo em pleno funcionamento, como mostrado na Figura 19.

Figura 18 - Tela do emulador de OBD II



Fonte: Dados do Autor

Os componentes do emulador foram soldados em uma placa de prototipagem e posicionados dentro de uma caixa de plástico com as instruções de uso, como mostrado na Figura 20. Do lado de fora deixou-se o conector OBD-II fêmea (o mesmo utilizado nos carros) e um conector de fonte de corrente contínua de 12V.

Basta alimentar o emulador com a fonte correta, conectar o protótipo no emulador através do cabo OBDII e seguir as instruções da caixa. Dessa forma o controle do emulador é feito através de uma página da internet conforme mostrado na figura 19.

Com o emulador em funcionamento, foi iniciado os testes do protótipo. Primeiramente foi conferida a operação do microcontrolador através da gravação de um código que pisca um LED acoplado a um dos pinos da placa. Em seguida se testou a comunicação com o barramento CAN com um código que faz leitura do RPM do veículo.

Posteriormente foi realizado o teste de operação do SIM808 através de um código que adquire a posição GPS e envia os dados para o banco de dados. Para comprovar o funcionamento dos testes é utilizado um sistema de *debugging* em que o código envia mensagens críticas para o UART acerca do funcionamento dos componentes.

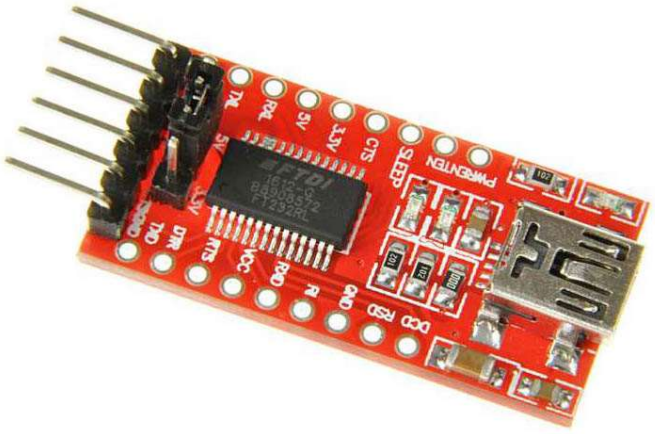
Figura 19 - Emulador de OBD II dentro de uma caixinha de plástico

Fonte: Dados do Autor

Por exemplo, quando a configuração do MCP2515 é realizada com sucesso, o código envia a seguinte mensagem para a UART: “MCP2515 Iniciado com Sucesso!”, caso a configuração falhe, o sistema envia: “Erro ao inicializar o MCP2515... Falha permanente! Por favor checar o código e conexões”.

Dessa forma, é possível monitorar o funcionamento do protótipo através da conexão de um adaptador de USB para UART como o módulo conversor mostrado na Figura 21.

Figura 20 - Módulo Conversor de USB para UART/RS232



Fonte: Site de vendas DX[4]

[4] Disponível em: <https://www.amazon.com/BBOXIM-FT232RL-Download-Serial-Module/dp/B07VSC1DQN>. Acesso em 23 Jul. 2020.

Conecta-se o módulo conversor na configuração mostrada na lista abaixo e utiliza-se o programa "*screen*" através de um terminal para mostrar as mensagens do protótipo na tela de um computador.

- TX do protótipo no RX do módulo.
- RST do protótipo no DTR to módulo, através de um capacitor de 0.1uF.
- 5V do protótipo no VCC do módulo.
- GND do protótipo no GND do módulo.

Após todos esses testes, o protótipo está preparado para receber o software final de funcionamento do sistema, onde demonstra que é possível estabelecer comunicação com o veículo possibilitando a verificação de diversas informações em tempo real e ainda enviá-las para um servidor. O resultado do *debugging* desse teste final é mostrado no quadro abaixo onde o valor do RPM do veículo é lido, juntamente com a posição do GPS e a informação é enviada para o banco de dados do firebase (Figura 11).

```
MCP2515 Iniciado com Sucesso!
AT
AT
OK
AT+CGPSPWR=0
AT+CGPSPWR=0
OK
AT+CGPSPWR=1
AT+CGPSPWR=1
OK
AT+CGPSRST=0
AT+CGPSRST=0
OK
AT+CGPSSTATUS?
AT+CGPSSTATUS?
+CGPSSTATUS: Location Unknown
```

```
OK

AT+CGPSSTATUS?
AT+CGPSSTATUS?
+CGPSSTATUS: Location Not Fix

OK

AT+CGPSSTATUS?
AT+CGPSSTATUS?
+CGPSSTATUS: Location 3D Fix

OK

AT+CGPSSTATUS?
AT+CGPSSTATUS?
+CGPSSTATUS: Location 3D Fix
AT+SAPBR=0,1
AT+SAPBR=0,1
OK
AT+CPIN?
AT+CPIN?
+CPIN: READY
AT+SAPBR=3,1,"Contype","GPRS"
AT+SAPBR=3,1,"Contype","GPRS"
OK
AT+SAPBR=3,1,"APN","gprs.oi.com.br"
AT+SAPBR=3,1,"APN","gprs.oi.com.br"
OK
AT+SAPBR=1,1
AT+SAPBR=1,1
OK
AT+SAPBR=2,1
```

```
AT+SAPBR=2,1
+SAPBR: 1,1,
==================================================
Programa de comunicacao completo: MCP2515+GPS+GPRS
==================================================
Mensagem enviada com sucesso!
"N": 4, "M": "0x41", "P": "0x0C", "A":67, "B":220, "C":0, "D":0
AT+CGPSINF=2
AT+CGPSINF=2
+CGPSINF: 2
AT+HTTPINIT
AT+HTTPINIT
ERROR
AT+HTTPTERM
AT+HTTPTERM
OK
AT+HTTPINIT
AT+HTTPINIT
OK
AT+HTTPSSL=1
AT+HTTPSSL=1
OK
AT+HTTPPARA="CID",1
AT+HTTPPARA="CID",1
OK
AT+HTTPPARA="URL","frota-db295.firebaseio.com/test.json"
AT+HTTPPARA="URL","frota-db295.firebaseio.com/test.json"
OK
AT+HTTPDATA="126",10000
AT+HTTPDATA="126",10000
DOWNLOAD
{"LT":"-010.2470","LG":"-0048.3517","N": 4, "M": "0x41", "P": "0x0C", "A":67,
"B":220, "C":0, "D":0}
```

```
OK
AT+HTTPACTION=1
AT+HTTPACTION=1
OK
```

Observe que o motivo dos comandos duplicados é que o *echo* do terminal estava ligado, então, todo comando enviado é repetido pelo SIM808 quando ele confirma o recebimento. Inicialmente é mostrado que o MCP foi iniciado normalmente com as configurações que foram passadas para o chip, logo na primeira linha aparece: “MCP iniciado com sucesso!”

Depois disso, o comando AT é enviado para o SIM808, e é esperado a resposta OK, logo em seguida é enviado o comando AT+CGPSPWR=0 que desliga o circuito do GPS. Depois o comando AT+CGPSPWR=1 liga o circuito novamente. Esse procedimento é uma redundância para o caso do aparecimento de erros inesperados.

O comando AT+CGPSRST=0 reseta o GPS para as configurações padrões, depois o comando AT+CGPSSTATUS? solicita qual o status do GPS. Nesse caso a resposta que nos interessa é +CGPSSTATUS: Location 3D Fix seguida de OK, o que significa que ao menos três satélites estão no campo de visão para uma triangulação e aquisição das coordenadas de localização.

Em seguida os comandos AT resetam o GPRS, verificam se o chip está conectado e configura o APN para o GPRS da operadora Oi. Depois aparece a mensagem “Programa de comunicação completo: MCP2515+GPS+GPRS” que dá início a um loop onde são: enviada uma requisição de um parâmetro para o carro, recebimento do valor do parâmetro, requisição da localização, abertura de protocolo HTTP e envio dos dados formatados para o endereço do banco de dados. Depois disso o *loop* se repete e o resultado pode ser conferido na figura 13.

A localização do GPS é obtida através do comando AT+CGPSINF=2, que retorna a a posição no formato de latitude e longitude (SIMCOM, 2014). E os dados são enviados para o servidor através do comando AT+HTTPDATA.

6.3 RESULTADOS ALCANÇADOS

6.3.1 MANUTENÇÃO PREVENTIVA DA FROTA DE VEÍCULOS

Por meio da porta OBD II presente em todos os carros da frota atual do Poder Judiciário do Estado do Tocantins, o protótipo irá consultar diversas informações, dentre elas as que competem a manutenção preventiva. Pode-se destacar: os códigos de erros do veículo, o consumo de combustível, odômetro, mistura de combustível e até a pressão dos pneus (em alguns casos).

6.3.2 CONTROLE DO CONSUMO DE COMBUSTÍVEL

Com esse protótipo será possível controlar o nível dos combustíveis, se existe mistura no mesmo, e fazer uma análise do desempenho do condutor conforme a velocidade que o mesmo dirige o veículo. Também será possível manter um histórico de últimos abastecimentos e assim gerar relatórios de consumo.

6.3.3 LOCALIZAÇÃO EM TEMPO REAL DE VEÍCULOS

Por meio do sistema de posicionamento global (GPS), é possível rastrear com precisão de até 5 metros a rota do veículo, fornecendo um total controle da movimentação dos carros e da localização dos mesmos.

6.3.4 TRANSPARÊNCIA DE GASTOS PÚBLICOS

Com esses dados será possível tornar pública essas informações e gerar relatórios de gastos, corroborando com o princípio da transparência na gestão pública.

7 CONCLUSÕES

O protótipo teve seu funcionamento comprovado e mostrou a possibilidade de se desenvolver um sistema de rastreamento e recolhimento de informações do barramento de dados do carro, que visa trazer benefícios para a gestão consciente dos recursos da administração pública, em especial, o Tribunal de Justiça do Tocantins onde será empregado como teste.

Esse sistema é relevante para segurança, manutenção e eficiência de frotas de veículos devido às diversas informações que podem ser coletadas em tempo real. Sendo elas o

consumo de combustível, rotas utilizadas, velocidade média, odômetro e acesso a código de defeitos do veículo.

De maneira análoga, devido ao monitoramento do tráfego dos veículos, é possível acionar o sistema de segurança caso o veículo trafegue por rotas não previstas ou pré informadas. De modo a agir automaticamente no caso de furto ou sequestro de veículo.

Durante o desenvolvimento do projeto, foram encontrados alguns problemas como mal contatos no emulador e na placa gravadora, isso atrasou bastante os testes. Portanto, sugere-se como trabalho futuro o desenvolvimento de uma placa de circuito impresso para o emulador e outra para a placa gravadora, substituindo o uso de fios, que são passíveis de mal contato, por trilhas que são muito mais robustas.

Ademais, sugere-se que para o emulador, seja acrescentado mais opções de PID além dos três existentes, tornando seu uso mais próximo de um carro real e flexibilizando a possibilidade de testes mais complexos. Uma outra melhoria no emulador, seria o uso de um display LCD com botões para o controle dos valores dos PID simulados, eliminando a necessidade de uma conexão com a internet para o controle do mesmo.

Para a próxima versão será necessário fazer algumas correções na placa do protótipo, sendo elas: Acrescentar um resistor de 120Ohms de fim de linha entre os pinos CANH e CANL. Esse resistor absorve o sinal do barramento e evita que o mesmo seja refletido para a fonte anulando a leitura do sinal. Também é necessário acrescentar um resistor Pull Up no pino CS do MCP2515, para desabilitar esse chip durante a gravação do código no ATMEGA328P. E por fim, é necessário corrigir os pinos do TXB0104, o mesmo não foi utilizado devido a incompatibilidade dos pinos.

Também é necessária a finalização da aplicação web que possuirá grande parte do processamento dos dados brutos enviados ao servidor, será a porta de entrada dos usuários contendo o controle dos veículos, mapa com localização em tempo real, *dashboards* com as informações dos veículos e além disso terá a inteligência necessária para a gestão propriamente dita da frota.

A segunda versão do hardware já está sendo desenvolvida para uma miniaturização mais efetiva e eliminação da complexidade de se atuar com dois chips, além disso, se utilizará um microcontrolador com capacidades muito superiores ao utilizado na versão 1, o que possibilitará o pré-processamento local dos PIDs antes de enviá-los ao banco de dados, se tornando um sistema mais eficiente.

Dessa forma, a aplicabilidade desse projeto impacta positivamente em todos os pontos abordados anteriormente, além de ser viável, relevante e inovador.

8 REFERÊNCIAS

ABREU, ADS; MICROCONTROLADA, **Arduino–Plataforma Eletrônica**. 124p. 2012. Tese de Doutorado. Dissertação (Bacharel)–Centro de Ciências Exatas e Tecnologia–Departamento de Engenharia de Eletricidade, Universidade Federal do Maranhão, São Luís.

ARDUINO UNO SMD. **Arduino**, 2016. Disponível em: https://www.arduino.cc/en/Main/ArduinoBoardUnoSMD. Acesso em: 20 de Nov. de 2019.

ATZORI, L.; IERA, A.; MORABITO, G. **The Internet of Things:** A survey. Computer Networks. 2010.

AXELSON, Jan. **Serial port complete**. Lakeview Research, 2000.

BOSCH, R. **CAN Specification: Version 2.0.** Stuttgart. 1995.

BRUDNA, Cristiano. **Desenvolvimento de sistemas de automação industrial baseados em objetos distribuídos e no barramento CAN**. 2000.

Centro de Justiça e Sociedade. Disponível em: <https://direitorio.fgv.br/cjus>. Acesso em: 26 jul. 2018.

COOKSEY, Diana. **Understanding the global positioning system (gps)**. Online Report, Montana State University-Bozeman, 2016.

COORPORATION, Atmel. **Atmel ATMEGA328P datasheet**. Accessed: Apr, v. 26, p. 2018, 2011.

DE ALMEIDA GUIMARÃES, Alexandre; SARAIVA, Antônio Mauro. **O Protocolo CAN: Entendendo e Implementando uma Rede de Comunicação Serial de Dados baseada no Barramento "Controller Area Network"**. 2002.

DE SOUZA, Andrey Gustavo; CAMPOS, Gustavo Lobato. **Rede CAN Veicular:** levantamento bibliográfico e apresentação de conceitos iniciais. ForScience, v. 5, n. 1, 2017.

FERNANDES, Jose Natanael Oliveira. **A real-time embedded system for monitoring of cargo vehicles, using controller area network (CAN)**. IEEE Latin America Transactions, v. 14, n. 3, p. 1086-1092, 2016.

FIELDING, Roy T.; TAYLOR, Richard N. **Architectural styles and the design of network-based software architectures.** Irvine: University of California, Irvine, 2000.

FRIED, Limor. Downloads, 2015. Disponível em: https://learn.adafruit.com/adafruit-fona-808-cellular-plus-gps-breakout/downloads. Acesso em: 20 de Nov. de 2019.

GUOHUAN, L.; HAO, Z.; WEI, Z. **Research on designing method of CAN bus and Modbus protocol conversion interface**, 2009 International Conference on Future BioMedical Information Engineering (FBIE), Sanya, 2009, pp. 180-182.

HSU, W.; LIU, S. **Design and Implementation of CAN-USB Converter Based on ARM7 Serial Protocol API**, 2012 International Symposium on Computer, Consumer and Control, Taichung, 2012, pp. 333-336.

IPEIA. **Projeto Esplanada Sustentável (PES)**, 2012. Disponível em: http://www.ipea.gov.br/portal/index.php?option=com_content&view=article&id=16914 Acesso em: 1 de ago. de 2018.

KHAMAMKAR, R.; JADHAV, A.; THOKE, S.; CHAPLE, G**. Smart Vehicle Safety Monitoring System Using CAN Protocol**, 2018 IEEE Punecon, Pune, India, 2018, pp. 1-5.

MARGOLIS, Michael. **Arduino cookbook:** recipes to begin, expand, and enhance your projects. " O'Reilly Media, Inc.", 2011.

MCCORD, Keith. **Automotive Diagnostic Systems:** Understanding OBD I and OBD II. CarTech Inc, 2011.

MCROBERTS, Michael. **Arduino Básico-2ª edição:** Tudo sobre o popular microcontrolador Arduino. Novatec Editora, 2015.

ON-BOARD Diagnostics. In**: Wikipédia:** a enciclopédia livre. Disponível em: <https://en.wikipedia.org/wiki/On-board_diagnostics> Acesso em: 8 jun. 2019.

PARKES, S. **Internet of Things could be de low cost 'connectivity key'.** ITU. 2016. Disponível em: < http://www.itu.int/net/pressoffice/press_releases/2016/02.aspx#.XydWsShKjIW>. Acesso em: 23 de jun. de 2019.

PESÉ, M. et al. **LibreCAN:** Automated CAN Message Translator. Nov, 2019.

Privacidade e Segurança no Firebase. **Firebase**. 2019. Disponível em: <https://firebase.google.com/support/privacy>. Acesso em: 15 de jun. de 2019.

ROBERTSON, Christopher T. **Printed circuit board designer's reference:** basics [en línea]. Estados Unidos de América. The fabrication process and fabrication notes, 22h, 2003.

SALCIANU, M.; FOSALAU, C. **A new CAN diagnostic fault simulator based on UDS protocol**, 2012 International Conference and Exposition on Electrical and Power Engineering, Iasi, 2012, pp. 820-824.

SALUNKHE, A. A.; KAMBLE, P. P.; JADHAV, R.. **Design and implementation of CAN bus protocol for monitoring vehicle parameters**, 2016 IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT), Bangalore, 2016, pp. 301-304.

SIMCOM. **SIM808 GPS Application.** Jan, 2014.

STANDARD, S. A. E. **Diagnostic Connector Equivalent to ISO/DIS 15031-3, SAE J1962**, 2003.

STANDARD, S. A. E. **E/E diagnostic test modes. SAE J1979**, Abr, 2002.

SUN, Y.; LIU, X.; YU, M. **Real-time communication protocol for CAN based on node transfer**, 2010 8th World Congress on Intelligent Control and Automation, Jinan, 2010, pp. 1140-1144.

TEXAS INSTRUMENTS. **LM2596 SIMPLE SWITCHER® Power Converter 150-kHz 3-A Step-Down Voltage Regulator.** Fev, 2020.

ZHU, M.; JIANG, G.; GAO, J. **Design of CAN higher layer protocol for airborne applications**, 2011 IEEE 3rd International Conference on Communication Software and Networks, Xi'an, 2011, pp. 288-293.

ANEXO A

Tabela A.1 - Valor médio da aquisição dos componentes do protótipo.

Componentes	**Quant.**	**Valor Unitário**	**Valor Total**
Diodo Schottky SMD SS36	1	R$0,59	R$0,59
Regulador de tensão LM2596	1	R$4,69	R$4,69
Capacitor Eletrolítico 100uF	1	R$1,36	R$1,36
Capacitor Eletrolítico 220uF	2	R$1,27	R$2,54
Capacitor Cerâmico 10uF	1	R$0,64	R$0,64
Capacitor Cerâmico 100nF 0603	7	R$0,10	R$0,73
Capacitor Cerâmico 100nF 0805	4	R$0,11	R$0,45
Capacitor Cerâmico 22pF	4	R$0,10	R$0,42
Resistor 2K2	1	R$0,10	R$0,10
Resistor 4K7	2	R$0,10	R$0,20
Resistor 100R	2	R$0,11	R$0,21
Resistor 10R	1	R$0,10	R$0,10
Resistor 10K	3	R$0,10	R$0,30
Resistor 1K	1	R$0,10	R$0,10
Indutor 33nH	1	R$0,24	R$0,24
Indutor 47uH	1	R$2,30	R$2,30
Suporte chip SIM	1	R$3,74	R$3,74
Regulador de Tensão LM340	1	R$11,13	R$11,13
Capacitor Cerâmico 220nF	1	R$0,23	R$0,23
LED Vermelho	2	R$0,26	R$0,52
Suporte Cartão de Memória SD	1	R$2,33	R$2,33
LED Verde	1	R$0,56	R$0,56
Regulador de Tensão LM1117	1	R$1,42	R$1,42
Conector de RF Coaxial	2	R$9,33	R$18,66

Botão RESET	1	R$2,30	R$2,30
Cristal Oscilador 16MHz	1	R$1,35	R$1,35
Cristal Oscilador 8MHz	1	R$2,93	R$2,93
Barra de Pinos Macho 1x40 2,54mm	1	R$0,95	R$0,95
Barra de Pinos Macho 2x6 2,54mm SMD	1	R$1,12	R$1,12
Transistor NPN S8050	1	R$0,25	R$0,25
Microcontrolador ATMEGA328-AU	1	R$31,24	R$31,24
Conversor de Nível	2	R$4,62	R$9,25
Controlador CAN MCP2515	1	R$7,04	R$7,04
Transceiver CAN MCP2551	1	R$5,07	R$5,07
Módulo SIM808 GSM/GPRS com GPS	1	R$41,86	R$41,86
Chip de proteção ESD	1	R$0,92	R$0,92
Barra de Pinos Fêmea 1x4 2,54mm	1	R$0,78	R$0,78
Placa de Circuito Impresso de potência	1	R$4,21	R$4,21
Placa de Circuito Impresso de dados	1	R$4,21	R$4,21
TOTAL			**R$167,05**

A tabela acima contempla os valores de fretes e impostos de importação. Os valores possuem mais de 4 casas decimais de precisão, contudo, utilizou-se arredondamento para facilitar a leitura.

Tabela A.2 - Valor médio da aquisição dos componentes do emulador.

Materiais	**Quantidade**	**Valores**
Caixa Plástica	1	R$20,24
Módulo CAN Bus	1	R$36,90
Placa de Fenolite Ilhada	5	R$14,95
Conversor Nivel Lógico	1	R$21,35
Módulo MCP2551	1	R$25,41
Conector OBD Fêmea	1	R$39,00
ESP32	1	R$45,00
TOTAL		**R$202,85**

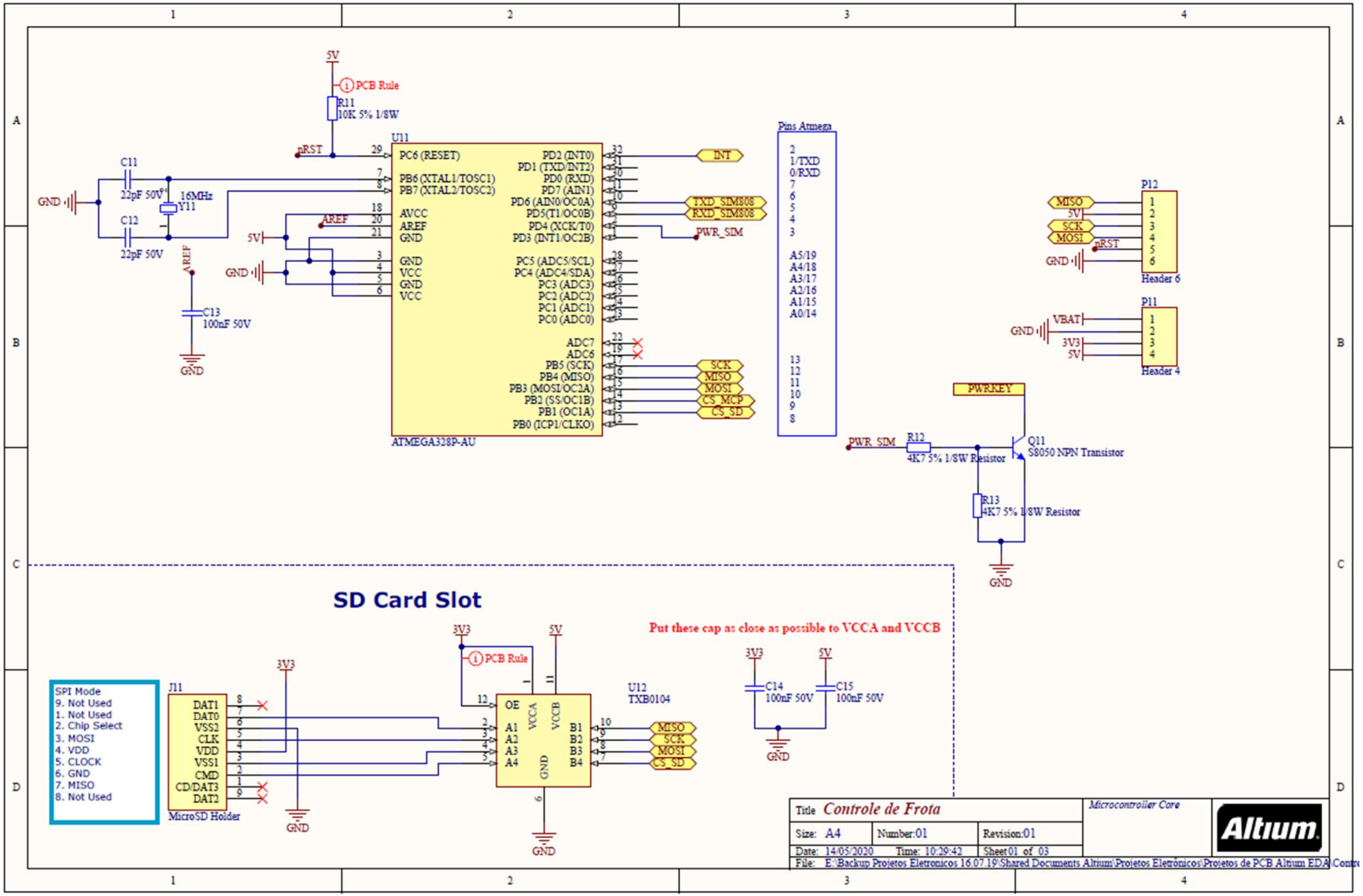
A tabela acima contempla os valores de fretes.

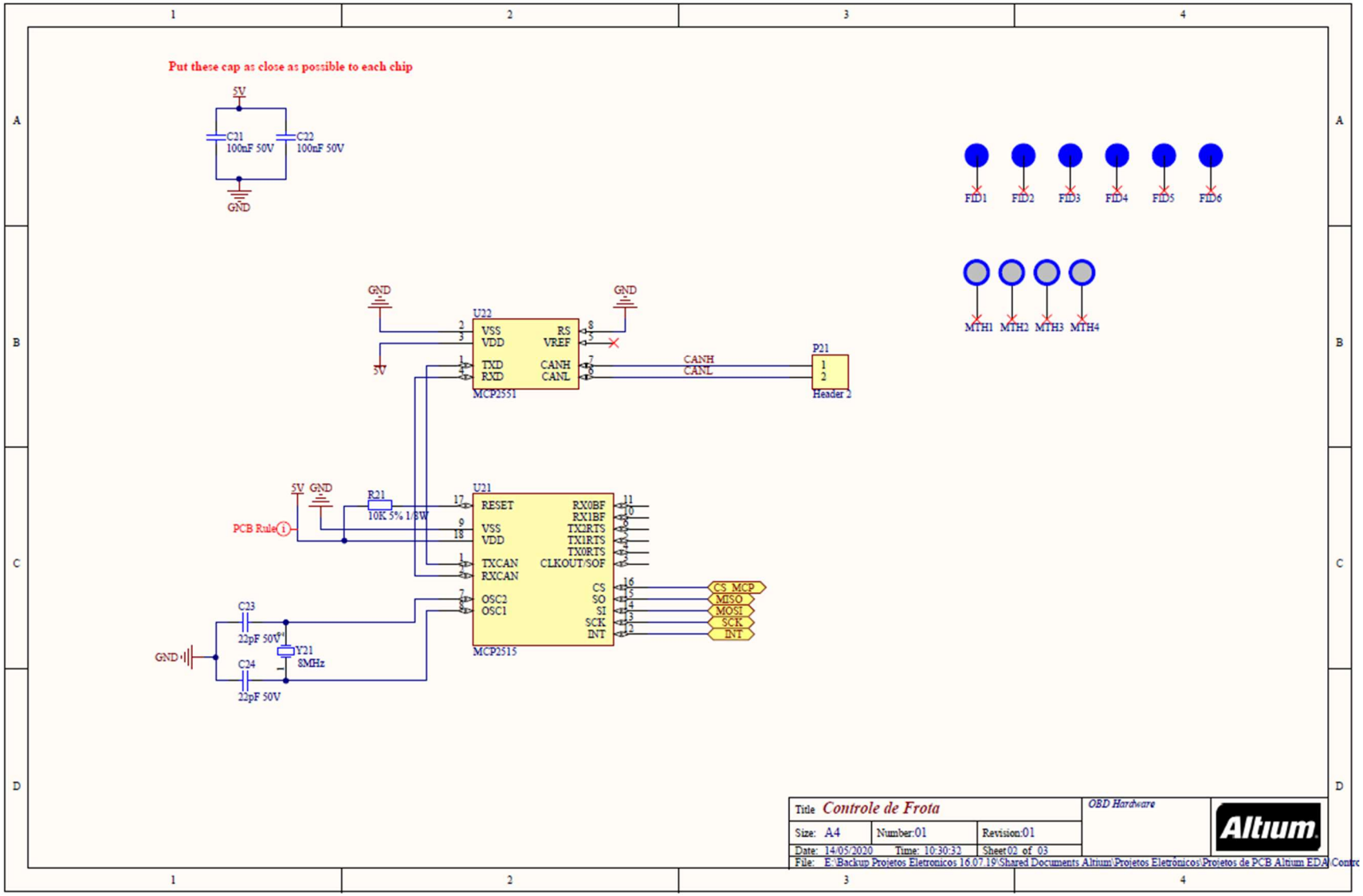
Tabela A.3 - Valor médio da aquisição dos componentes de testes.

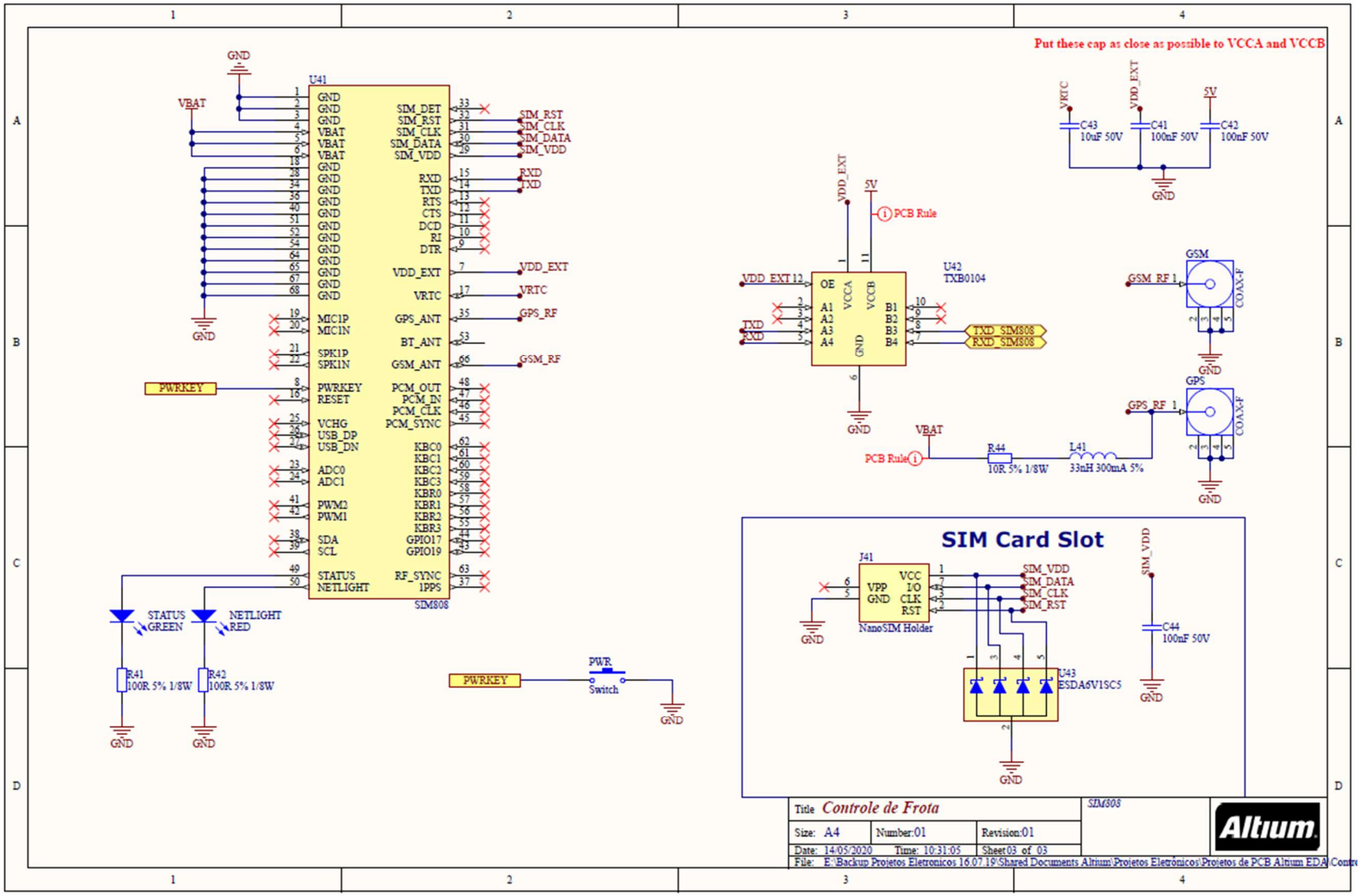
Materiais	**Quantidade**	**Valores**
Protoboard	1	R$38,00
Módulo cartão de memória SD	1	R$22,49
Cabo OBD II - USB (ELM327)	1	R$212,00
TOTAL		**R$272,49**

A tabela acima contempla os valores de fretes.

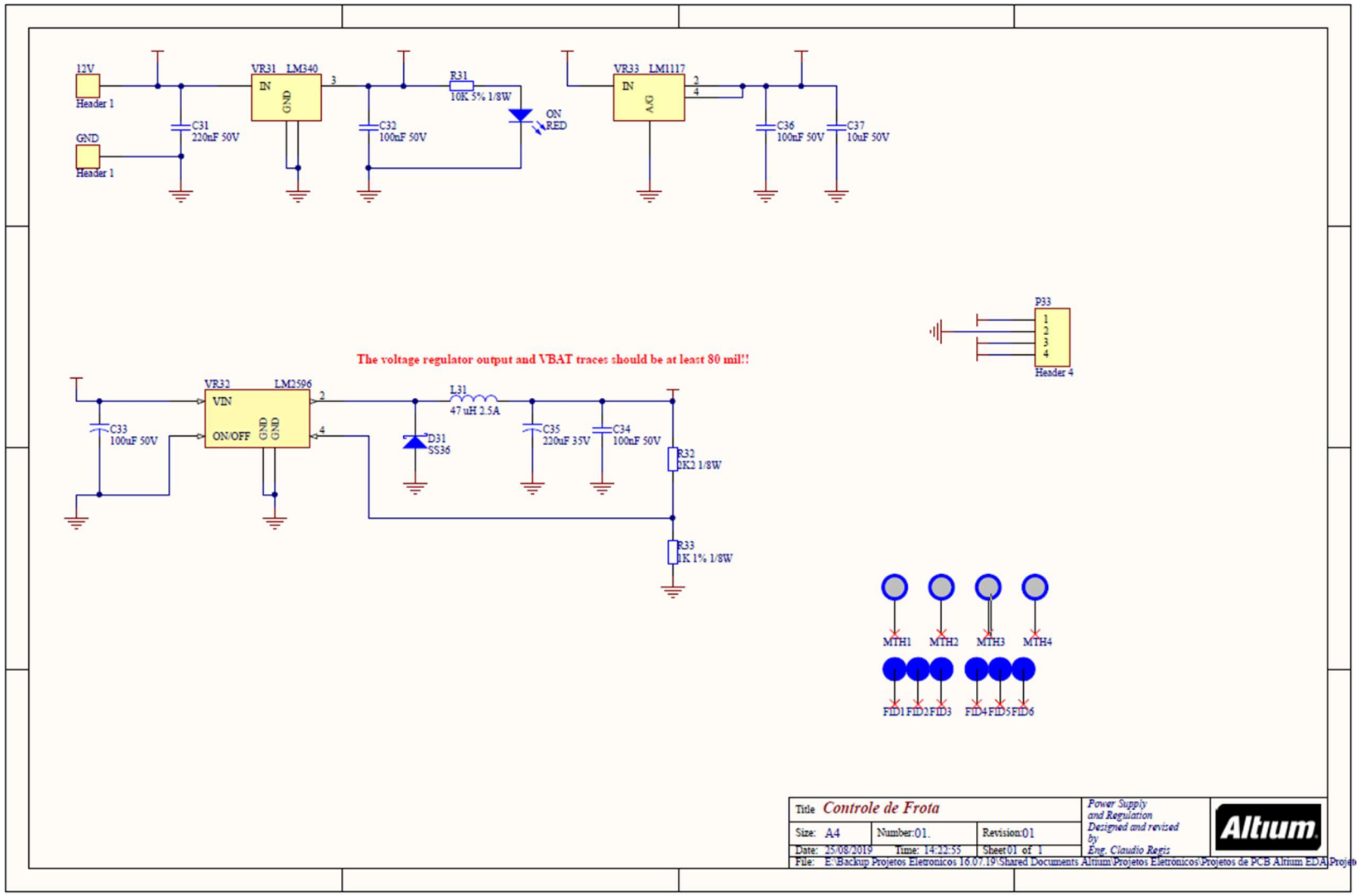
ANEXO B


SD Card Slot
Put these cap as close as possible to VCCA and VCCB
ATMEGA328P-AU
MicroSD Holder
SPI Mode
9. Not Used
1. Not Used
2. Chip Select
3. MOSI
4. VDD
5. CLOCK
6. GND
7. MISO
8. Not Used
Pins Atmega
Header 6
Header 4
S8050 NPN Transistor
4K7 5% 1/8W Resistor
TXB0104
Title Controle de Frota
Microcontroller Core
Size: A4
Number:01
Revision:01
Date: 14/05/2020
Time: 10:29:42
Sheet 01 of 03
File: E:\Backup Projetos Eletronicos 16.07.19\Shared Documents Altium\Projetos Eletronicos\Projetos de PCB Altium EDA\Contr
Altium

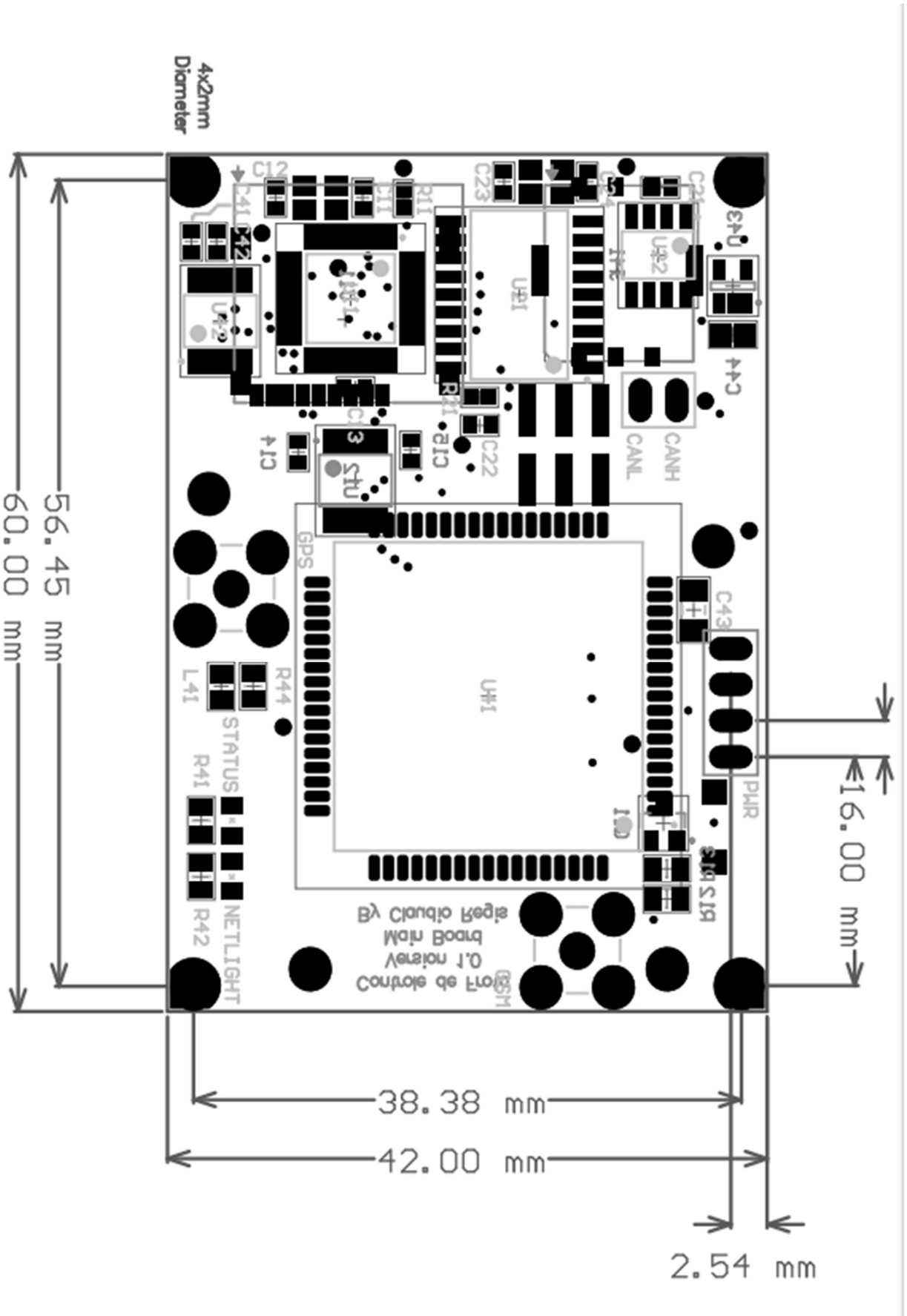

Put these cap as close as possible to each chip
5V
C21
100nF 50V
C22
100nF 50V
GND
FID1
FID2
FID3
FID4
FID5
FID6
MTH1
MTH2
MTH3
MTH4
U22
VSS
VDD
TXD
RXD
RS
VREF
CANH
CANL
MCP2551
P21
Header 2
U21
RESET
VSS
VDD
TXCAN
RXCAN
OSC2
OSC1
RX0BF
RX1BF
TX2RTS
TX1RTS
TX0RTS
CLKOUT/SOF
CS
SO
SI
SCK
INT
MCP2515
R21
10K 5% 1/8W
PCB Rule
C23
22pF 50V
C24
22pF 50V
Y21
8MHz
CS_MCP
MISO
MOSI
SCK
INT
Title Controle de Frota
OBD Hardware
Size: A4
Number:01
Revision:01
Date: 14/05/2020
Time: 10:30:32
Sheet 02 of 03
File: E:\Backup Projetos Eletronicos 16.07.19\Shared Documents Altium\Projetos Eletrônicos\Projetos de PCB Altium EDA\Contr


Put these cap as close as possible to VCCA and VCCB
U41
SIM808
U42
TXB0104
J41
NanoSIM Holder
U43
ESDA6V1SC5
SIM Card Slot
PWRKEY
PWR
Switch
STATUS
GREEN
NETLIGHT
RED
R41
100R 5% 1/8W
R42
100R 5% 1/8W
R44
10R 5% 1/8W
L41
33nH 300mA 5%
C43
10uF 50V
C41
100nF 50V
C42
100nF 50V
C44
100nF 50V
GSM
GPS
COAX-F
PCB Rule
TXD_SIM808
RXD_SIM808
Title Controle de Frota
SIM808
Size: A4
Number:01
Revision:01
Date: 14/05/2020
Time: 10:31:05
Sheet 03 of 03
File: E:\Backup Projetos Eletronicos 16.07.19\Shared Documents Altium\Projetos Eletrônicos\Projetos de PCB Altium EDA\Contr
Altium

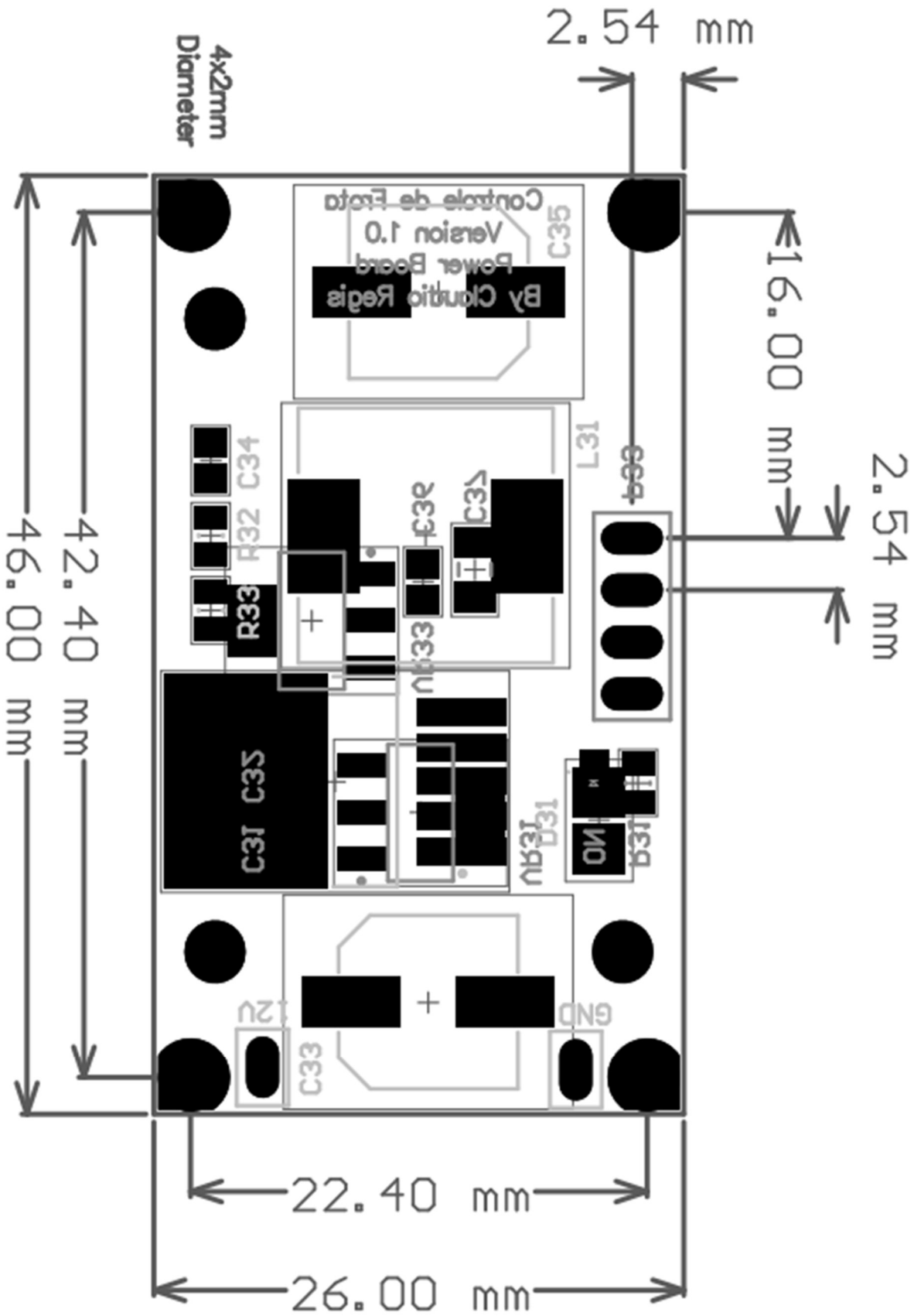
ANEXO C


12V
Header 1
GND
Header 1
C31
220nF 50V
VR31 LM340
IN
GND
3
C32
100nF 50V
R31
10K 5% 1/8W
ON
RED
VR33 LM1117
IN
A/G
2
4
C36
100nF 50V
C37
10uF 50V
P33
1
2
3
4
Header 4
The voltage regulator output and VBAT traces should be at least 80 mil!!
VR32 LM2596
VIN
ON/OFF
GND
GND
2
4
C33
100uF 50V
L31
47 uH 2.5A
D31
SS36
C35
220uF 35V
C34
100nF 50V
R32
2K2 1/8W
R33
1K 1% 1/8W
MTH1
MTH2
MTH3
MTH4
FID1
FID2
FID3
FID4
FID5
FID6
Title Controle de Frota
Size: A4
Number:01.
Revision:01
Date: 25/08/2019
Time: 14:22:55
Sheet 01 of 1
File: E:\Backup Projetos Eletronicos 16.07.19\Shared Documents Altium\Projetos Eletronicos\Projetos de PCB Altium EDA\Projeto
Power Supply and Regulation Designed and revised by Eng. Claudio Regis
Altium

ANEXO D


4x2mm
Diameter
60.00 mm
56.45 mm
16.00 mm
38.38 mm
42.00 mm
2.54 mm
Control de Trafo
Version 0.1
Main Board
By Claudio Vegia
GPS
GSM
CANH
CANL
STATUS
NETLIGHT
PWR
U#1
U#2
U#3
U#4
L41
R41
R42
R44

ANEXO E


2.54 mm
4x2mm Diameter
16.00 mm
2.54 mm
46.00 mm
42.40 mm
22.40 mm
26.00 mm
C35
L31
C34
R32
R33
C33
D31
ON
GND

ANEXO F

```
#include <Arduino.h>
#include <mcp_can.h>
#include <SPI.h>
#include <SoftwareSerial.h>


// ============================
// Definitions
// ============================
// 7E0/8 = Engine ECM
// 7E1/9 = Transmission ECM
#define LISTEN_ID 0x7EA
#define REPLY_ID 0x7E0
#define FUNCTIONAL_ID 0x7DF
#define CAN0_INT 2  // Pino 2 no INT
#define PWRSIM808 4 // Pino que liga o SIM808
// Apenas para debug
#define DEBUG 1
// ============================
// End of Definitions
// ============================

// ============================
// Variables
// ============================
byte pidRequest[] = {0x02,0x01,0x0C};
unsigned long rxID;
byte dlc;  // Data Length Code
byte rxBuf[8];
      /* Esse são os dados que serão enviados para o banco de dados
      * rxBuf[0] - N = Número de bytes de informação
      * rxBuf[1] - M = Modo de operação (0x41 - response, 0x01 - request)
      * rxBuf[2] - P = Parameter ID
      * rxBuf[3] - A = Primeiro 8 bits de data
      * rxBuf[4] - B = Segundo 8 bits de data
      * rxBuf[5] - C = Terceiro 8 bits de data
      * rxBuf[6] - D = Ultimo 8 bits de data
      * rxBuf[7] – Sem uso
      */
// "N": 128, "M": 0x4C, "P": 0x0C, "A":128, "B":128, "C":128, "D":128
char canString[90]; // Vetor para armazenar a string serial
const char* apn = "gprs.oi.com.br"; //"claro.com.br";
char latitude[15];
char longitude[15];
char frame[100];
```

```
// {"LG":"-0048.3516","LT":"-010.2470"}
char data[127];
// ============================
// End of Variables
// ============================


// ============================
// Objects
// ============================
MCP_CAN CAN0(10);   // Pin 10 no CS, lembrar de usar resistor pull up na
programação ISP
SoftwareSerial SIM808Serial(6, 5);  //TXD da placa (RX do Arduino), RXD da placa
(TX do Aruino), respectivamente
// ============================
// End of Objects
// ============================

// ============================
// Function definitions
// ============================
void powerOnCheck(void);
int8_t sendATcommand(const char* atCommand, const char* expectedAnswer, unsigned
int timeout);
int8_t startGPRS(void);
void closeGPRS(void);
int8_t getHTTP(const char *url, char *getResponse);
int8_t postHTTP(const char*url, const char *data);
int8_t startGPS(void);
uint8_t getGPS(void);
void closeGPS(void);
// ============================
// End of function definitions
// ============================

void setup() {
  #if DEBUG==1
    Serial.begin(9600);
  while(!Serial); // Wait Serial
  #endif
  SIM808Serial.begin(9600);

  // Inicializa o  MCP2515 rodando a 8MHz com baudrate de 500kb/s e com as mascaras
e filtros desativados.
  if(CAN0.begin(MCP_STDEXT, CAN_500KBPS, MCP_8MHZ) == CAN_OK){
    #if DEBUG==1
```

```
      Serial.println("MCP2515 Iniciado com Sucesso!");
    #endif
  }
  else{
    #if DEBUG==1
      Serial.println("Erro ao inicializar o MCP2515... Falha permanente!  Por favor checar o código e conexoes");
    #endif
    while(1); // Se tiver um erro na inicialização do MCP2515, trava o sistema
    // End
  }

  // Filtros de ID standard
  CAN0.init_Mask(0,0x7F00000);                // Init first mask...
  CAN0.init_Filt(0,0x7DF0000);                // Init first filter...
  CAN0.init_Filt(1,0x7E10000);                // Init second filter...

  CAN0.init_Mask(1,0x7F00000);                // Init second mask...
  CAN0.init_Filt(2,0x7DF0000);                // Init third filter...
  CAN0.init_Filt(3,0x7E10000);                // Init fouth filter...
  CAN0.init_Filt(4,0x7DF0000);                // Init fifth filter...
  CAN0.init_Filt(5,0x7E10000);                // Init sixth filter...

  CAN0.setMode(MCP_NORMAL);                   // Configurando o modo de operacao normal, logo o MCP2515 envia confirmacao para os dados recebidos.

  // Enfrentando problemas?  ============================================
  // Se voce nao esta recebendo nenhuma mensagem, descomente a linha setMode
  // abaixo para testar a conexao entre o Arduino e MCP2515.
  // Sendo assim, a mensagem enviada nesse programa será instantaneamente
  // recebida.
  // ====================================================================
  //CAN0.setMode(MCP_LOOPBACK);

  pinMode(CAN0_INT, INPUT);                   // Configurando o pino Interrupt como entrada
  pinMode(PWRSIM808, OUTPUT);
  delay(100);
  powerOnCheck();
  closeGPS();
  uint8_t GPSboot = 0;
  do{
    GPSboot = startGPS();
  } while (GPSboot==0);
  closeGPRS();
  startGPRS();
```

```
  delay(1000);

  #if DEBUG==1
    Serial.println("=================================================");
    Serial.println("Programa de comunicacao completo: MCP2515+GPS+GPRS");
    Serial.println("=================================================");
  #endif
}

void loop() {
  // Primeiramente enviamos um request para o MCP2515
  if(CAN0.sendMsgBuf(FUNCTIONAL_ID, sizeof(pidRequest), pidRequest) == CAN_OK){
    #if DEBUG==1
      Serial.println("Mensagem enviada com sucesso!");
    #endif
  }
  else {
    #if DEBUG==1
      Serial.println("Erro ao enviar mensagem...");
    #endif

  }

  // E então verificamos se ele respondeu
  if(!digitalRead(CAN0_INT)){                  // Se o pino CAN0_INT esta em nivel
baixo, fazer leitura do buffer recebido
    CAN0.readMsgBuf(&rxID, &dlc, rxBuf);       // Receber dados CAN
    // Se for um buffer de resposta 0x41
    if (rxBuf[1]==0x41) {
      sprintf(canString, "\"N\": %d, \"M\": \"0x%.2X\", \"P\": \"0x%.2X\",
\"A\":%d, \"B\":%d, \"C\":%d, \"D\":%d",
      rxBuf[0], rxBuf[1], rxBuf[2], rxBuf[3], rxBuf[4], rxBuf[5], rxBuf[6]);
      #if DEBUG==1
        Serial.println(canString);
      #endif
    }
  }
  // Depois disso, pega os dados do GPS e envia para o banco de dados
  getGPS();
  sprintf(data,"{\"LT\":\"%s\",\"LG\":\"%s\",%s}",latitude,longitude,canString);
  // POST a value
  postHTTP("frota-db295.firebaseio.com/test.json",data);
  // Espera 10s e envia de novo
  delay(10000);
}
```

```
int8_t startGPRS(){
  uint8_t answer=0;

  // Verifica se tem um cartão chip conectado
  if(sendATcommand("AT+CPIN?", "READY", 1000)==1){
    // Configura o bearer profile 1
    sendATcommand("AT+SAPBR=3,1,\"Contype\",\"GPRS\"", "OK", 1000);
    // Configura a APN
    sendATcommand("AT+SAPBR=3,1,\"APN\",\"gprs.oi.com.br\"", "OK", 1000);

    if(sendATcommand("AT+SAPBR=1,1", "OK", 5000)==1){
      // Se recebeu um IP
      if ((sendATcommand("AT+SAPBR=2,1"," 1,1,", 2000))==1){
        answer = 1;
      }
      else{
        answer = 0;
      }
    }
    else{
      answer=0;
    }
  }
  else{
    answer = 0;
  }
  return answer;
}

void closeGPRS(){
  sendATcommand("AT+SAPBR=0,1", "OK", 5000);
}

// O usuário é responsável por declarar uma getResponze com o tamanho compatível
com o tamanho do número
// de bytes da resposta do GET request
int8_t getHTTP(const char *url, char *getResponse){
  uint8_t answer=0;
  // Inicializa o serviço HTTP
  if (sendATcommand("AT+HTTPINIT", "OK", 10000)==1){
    // Configura a capacidade do HTTPS
    sendATcommand("AT+HTTPSSL=1","OK",2000);
    // Configura o parâmetro CID
    if(sendATcommand("AT+HTTPPARA=\"CID\",1", "OK", 5000)==1){
      // Configura a URL
```

```
      int sizeURL = ((int)strlen(url))+23;
      char auxUrl[sizeURL];
      memset(auxUrl, '\0', sizeURL);    // Initialize the string
      sprintf(auxUrl, "AT+HTTPPARA=\"URL\",\"%s\"", url);
      if (sendATcommand(auxUrl, "OK", 5000)==1){
        // Limpa o buffer de entrada
        while(SIM808Serial.available() > 0) SIM808Serial.read();
        // Inicia o GET request
        if(sendATcommand("AT+HTTPACTION=0", "+HTTPACTION: 0,200,", 30000)==1){
          // Limpa o buffer de entrada
          while(SIM808Serial.available() > 0) SIM808Serial.read();
          // Requisita a string GET request
          if(sendATcommand("AT+HTTPREAD", "+HTTPREAD", 2000)==1){
            int sizeResponse = (int)strlen(getResponse);
            memset(getResponse, '\0', sizeResponse);    // Initialize the string

            for(int i=0; i<sizeResponse; i++){
              if(SIM808Serial.available() != 0){
                getResponse[i] = SIM808Serial.read();
              }
            }
            sendATcommand("AT+HTTPTERM", "OK", 5000);
            answer=1;
          }
          else{
            answer=0;
          }
        }
        else{
          answer=0;
        }
      }
      else{
        answer=0;
      }
    }
    else{
      answer=0;
    }
  }
  else{
    answer=0;
  }
  return answer;
}
```

```
int8_t postHTTP(const char *url, const char *data){
  uint8_t answer=0;
  // Finaliza qualquer conexão HTTP aberta
  if(sendATcommand("AT+HTTPINIT","ERROR",3000)==1){
    sendATcommand("AT+HTTPTERM","OK",1000);
    sendATcommand("AT+HTTPINIT","OK", 10000);
  }
  // Envia a capacidade HTTO
  if (sendATcommand("AT+HTTPSSL=1","OK",2000)==1){
    // Configura o parâmetro CID
    if(sendATcommand("AT+HTTPPARA=\"CID\",1", "OK", 5000)==1){
      // Configura a URL
      int sizeURL = ((int)strlen(url))+23;
      char auxUrl[sizeURL];
      memset(auxUrl, '\0', sizeURL);    // Initialize the string
      sprintf(auxUrl, "AT+HTTPPARA=\"URL\",\"%s\"", url);
      if(sendATcommand(auxUrl, "OK", 5000)==1){
        int sizePost = ((int)strlen(data))+26;
        char auxPost[sizePost];
        memset(auxPost, '\0', sizePost);
        sprintf(auxPost, "AT+HTTPDATA=\"%d\",10000", sizePost);
        if(sendATcommand(auxPost, "DOWNLOAD", 1000)==1){
          if(sendATcommand(data,"OK",10000)==1){
            // Start the POST session
            if(sendATcommand("AT+HTTPACTION=1","+HTTPACTION: 1,200,",1500)==1){
              sendATcommand("AT+HTTPTERM", "OK", 5000);

              answer=1;
            }
            else{
              answer=0;
            }
          }
          else{
            answer=0;
          }
        }
        else{
          answer=0;
        }

      }
      else{
        answer=0;
      }
    }
```

```
    else{
      answer=0;
    }
  }
  else{
    answer=0;
  }
  return answer;
}

int8_t startGPS(){
    unsigned long previous;

    previous = millis();
    // Inicia o GPS
    sendATcommand("AT+CGPSPWR=1", "OK", 2000);
    sendATcommand("AT+CGPSRST=0", "OK", 2000);

    // Espera por uma posição fixa.
    while(( (sendATcommand("AT+CGPSSTATUS?", "2D Fix", 1000) ||
        sendATcommand("AT+CGPSSTATUS?", "3D Fix", 1000)) == 0 ) &&
        ((millis() - previous) < 90000));

    if ((millis() - previous) < 90000){
        return 1;
    }
    else{
        return 0;
    }
}

void powerOnCheck(){
  int8_t answer = 0;
  digitalWrite(PWRSIM808, HIGH);
  delay(1000);
  digitalWrite(PWRSIM808,LOW);
  // checks if the module is started
  if (sendATcommand("AT", "OK", 2000)!=1){
    // power on pulse
    // digitalWrite(onModuleLED,HIGH);
    // delay(3000);
    // digitalWrite(onModuleLED,LOW);

    // espera por uma resposta do modulo
    while(answer!=1){
      // Envia um AT a cada dois segundos
```

```
      answer = sendATcommand("AT", "OK", 2000);
    }
  }
}


uint8_t getGPS(){
    uint8_t counter, answer;
    long previous;

    // Clean the input buffer
    while(SIM808Serial.available() > 0) SIM808Serial.read();
    // request Basic string
    sendATcommand("AT+CGPSINF=2", "+CGPSINF: 2", 2000);

    counter = 0;
    answer = 0;
    memset(frame, '\0', 100);    // Initialize the strin
    previous = millis();
    // this loop waits for the NMEA string
    do{
      if(SIM808Serial.available() != 0){
          frame[counter] = SIM808Serial.read();
          counter++;
          // check if the desired answer is in the response of the module
          if (strstr(frame, "OK") != NULL)
          {
              answer = 1;
          }
      }
      // Espera por uma resposta até vencer o tempo
    }while((answer == 0) && ((millis() - previous) < 2000));

    frame[counter-3] = '\0';

    // Parses the string
    // Example: +CGPSINF: 2,182446.000,-010.2470,S,-0048.3516,W,1,8,0.91,246.4,M,-
17.6,M,,
    strtok(frame, ",");
    //strtok(NULL,","); // Jump Hour
    strcpy(latitude,strtok(NULL, ",")); // Gets latitude
    strtok(NULL,","); // Jump latitude indicator
    strcpy(longitude,strtok(NULL, ",")); // Gets longitude

    return answer;
}
```

```
// This function returns 1 if the expectedAnswer appear in response to the
// atCommand and did not reach the timeout
// Else it returns 0
int8_t sendATcommand(const char* atCommand, const char* expectedAnswer, unsigned
int timeout){
  uint8_t x=0, answer=0;
  char response[100];
  unsigned long previous;
  memset(response, '\0', 100);    // Initialize the string
  delay(100); // Delay to be sure no passed commands interfere

  while(SIM808Serial.available() > 0) SIM808Serial.read();    // Wait for clean
input buffer

  SIM808Serial.println(atCommand);    // Send the AT command
  x = 0;
  previous = millis();

  // this loop waits for the answer
  do{
    if(SIM808Serial.available() != 0){
      response[x] = SIM808Serial.read();
      x++;
      // check if the desired answer is in the response of the module
      if (strstr(response, expectedAnswer) != NULL){
        answer = 1;
      }
    }
    // Waits for the answer with time out
  }while((answer == 0) && ((millis() - previous) < timeout));
  #if DEBUG==1
    Serial.println(atCommand);
    Serial.println(response);
  #endif

  return answer;
}


void closeGPS(){
  sendATcommand("AT+CGPSPWR=0", "OK", 2000);
}
```