INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE MINAS GERAIS – CAMPUS FORMIGA
BACHARELADO EM ENGENHARIA ELÉTRICA

FÁBIO AUGUSTO SANTOS

MODELAGEM E REPRESENTAÇÃO DE *VEHICLE FUNCTIONS*

FORMIGA-MG

2020

FÁBIO AUGUSTO SANTOS

MODELAGEM E REPRESENTAÇÃO DE *VEHICLE FUNCTIONS*

Trabalho de Conclusão de Curso apresentado ao curso de Engenharia Elétrica do Instituto de Educação, Ciência e Tecnologia de Minas Gerais, *Campus* Formiga, como requisito para obtenção do título de Bacharel em Engenharia Elétrica.

Orientador: Prof. Dr. Gustavo Lobato Campos
Coorientador: Prof. Dr. Patrick Santos de Oliveira

FORMIGA-MG

2020

Santos, Fábio Augusto.
S237m Modelagem e representação de Vehicle Functions / Fábio Augusto Santos -- Formiga : IFMG, 2020.
91p. : il.

Orientador: Prof. Dr. Gustavo Lobato Campos
Co-orientador: Prof. Dr. Patrick Santos de Oliveira
Trabalho de Conclusão de Curso – Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais – *Campus* Formiga.

1. Vehicular Functions. 2. Model Based Design. 3. Automação Veicular. 4. Sistemas Embarcados. 5. Sistema Veicular. I. Campos, Gustavo Lobato. II. Oliveira, Patrick Santos de. III. Título.

CDD 621.3

Ficha catalográfica elaborada pela Bibliotecária Msc. Simoni Júlia da Silveira

FÁBIO AUGUSTO SANTOS

MODELAGEM E REPRESENTAÇÃO DE *VEHICULAR FUNCTIONS*

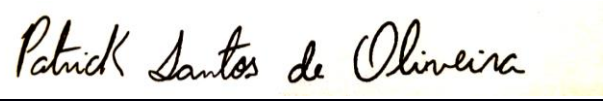
Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia Elétrica do Instituto Federal de Minas Gerais como requisito para obtenção do Título de Bacharel em Engenharia Elétrica.

Avaliado em: 20 de outubro de 2020.

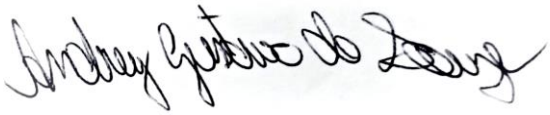
Nota: 100

BANCA EXAMINADORA

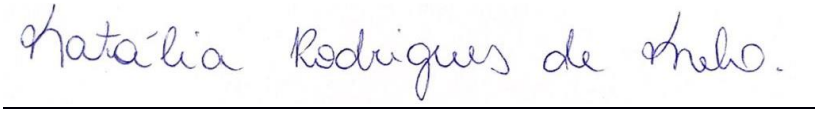
Prof. Dr. Gustavo Lobato Campos



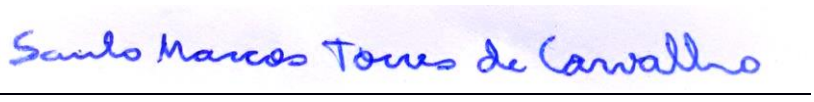
Prof. Dr. Patrick Santos de Oliveira



Andrey Gustavo de Souza



Natália Rodrigues de Melo



Saulo Marcos Torres de Carvalho

Dedico este trabalho à Deus, a meu pai que desde o começo não mediu esforços para me ajudar, à toda minha família e à minha namorada. Dedico de forma mais do que especial à minha mãe, que estará para sempre em meu coração e era o seu sonho a chegada deste momento. Sei que de onde ela estiver estará orgulhosa e feliz. Dedico também aos meus amigos e colegas de turma, que me deram forças para continuar no momento mais difícil da minha vida até hoje**.**

RESUMO

A indústria automobilística está a todo momento em crescimento, de tal forma que sua influência no mercado mundial não passe despercebida. Isso se dá ao grande investimento por parte das montadoras em pesquisa e desenvolvimento de novas tecnologias. Consequentemente os níveis de automação veicular estão cada vez mais altos, de forma a proporcionar ao condutor mais conforto e comodidade. Com isso os sistemas desenvolvidos devem atender crucialmente níveis de segurança elevados. O projeto de funções veiculares, do inglês, *Vehicular Functions* (VFs), objeto de desenvolvimento deste trabalho, complementa parte do desenvolvimento do sistema veicular, sendo elas responsáveis por descrever o comportamento e integração do sistema embarcado dentro da rede automotiva. Para conseguir tal objetivo, ocorre primeiramente a definição das VFs a serem trabalhadas e, a partir de simulações e construções de modelos baseados no *Model Based Desgin* (MBD), ocorre a validação dos sistemas construídos. Por fim, tem-se a montagem de um protótipo em escala reduzida, com base em circuitos desenvolvidos em simulador *online* e a construção da documentação de cada VF escolhida. De acordo com os resultados obtidos, é possível afirmar que os sistemas atenderam às expectativas e a documentação realizada pode ser compreendida de forma simples, clara e objetiva por pessoas interessadas pela área.

Palavras chave: *Vehicular Funcions*, *Model Based Desgin*, Automação Veicular, Sistemas Embarcados.

ABSTRACT

The automotive industry is constantly growing, so that its influence on the world market does not go unnoticed. This is due to the large investment by automakers in research and development of new technologies. Consequently, vehicle automation levels are increasingly higher, in order to provide the driver with more comfort and convenience. Thus, the developed systems must meet crucially security levels. The design of vehicle functions, from English, Vehicle functions (VFs), object of development of this work, complements part of the development of the vehicle system, being responsible for characterizing the behavior and integration of the embedded system within the automotive network. In order to achieve this objective, the definition of the VFs to be worked on occurs primarily and, based on simulations and model construction based on the Model Based Desgin (MBD), the built systems are validated. Finally, there is the assembly of a small scale prototype, based on circuits developed in an online simulator and the construction of documents for each chosen VF. According to the results obtained, it is possible to state that the systems met the expectations and the documentation made can be understood in a simple, clear and objective manner by people required by the area.

Keywords: Vehicular Functions, Model Based Desgin, Vehicle Automation, Embedded Systems.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A mobilidade individual se tornou um elemento de extrema importância para as pessoas atualmente. Com isso, a busca por veículos seguros e eficientes tem sido um critério do consumidor na hora de adquirir seu automóvel.

Enquanto os primeiros veículos eram baseados em sistemas puramente mecânicos, os modelos atuais são baseados em sistemas elétricos e eletrônicos. Isso vem acompanhado de uma complexidade que tende a evoluir de acordo com as configurações desejadas (HILLENBRAND, 2012).

A evolução do sistema mecânico para os sistemas elétricos e eletrônicos foi possível a partir da eletrônica embarcada que revolucionou a aplicabilidade e cobertura de muitos dos opcionais e funções oferecidas aos consumidores. Essa revolução na eletrônica automotiva tem melhorado significativamente o desempenho, a confiabilidade e o conforto nos automóveis (POGGETO, 2009).

Um sistema embarcado é um sistema projetado para atender funções específicas através de sensores, controladores e atuadores. Para uma comunicação eficiente de um sistema veicular, os sistemas embarcados fazem parte de uma grande rede composta por centrais eletrônicas de processamento, do inglês, *Electronic Control Units* (ECUs). Essas centrais são responsáveis por gerenciar diversas funções como o controle do motor, do sistema de transmissão, da carroceria, entre outros. Elas são nomeadas levando em consideração o controle a ser gerenciado, e como exemplo pode-se citar ECM (*Engine Control Module*) para gerenciamento do motor, TCM (*Transmission Control Module*) para a transmissão, BCM (*Body Control Module*) para a carroceria, ICM (*Instrument Control Module*) para o painel de instrumentos IPC *(Instrument Panel Control*) e PAM (*Parking Assistence Module*) para assistência de estacionamento (SILVA, 2017). Destaca-se que tais nomenclaturas podem variar de fabricante para fabricante.

Para realizar o controle e definir a lógica do algoritmo a ser implementado nas centrais eletrônicas, houve a necessidade do desenvolvimento de *Vehicular Functions* (VFs), uma vez que estes sistemas possuem equipamentos que são produzidos por diferentes fabricantes. Assim, uma documentação simples, clara e objetiva é necessária, pois ela é o guia para que a comunicação dos componentes seja possível, em uma linguagem única e de forma efetiva. A Figura 1 representa o desenvolvimento de uma VF.

Figura 1 - Desenvolvimento de uma VF

Fonte: Autor.

As diferentes linguagens representam os componentes, que são geralmente adquiridos de fornecedores diferentes. A pasta representa a documentação desenvolvida após os testes de verificação do correto funcionamento dos componentes e consequentemente da VF. De posse desta documentação, é possível a implementação no veículo, representadas pelas diversas funções que um veículo pode possuir.

Por parte das montadoras, a documentação das VFs de um determinado veículo é de extremo sigilo, pois descreve totalmente o comportamento que o *software* implementado para atender certas funcionalidades deve ter. Com isso, as VFs possuem impacto no projeto de um veículo e podem ser diferencial de uma determinada montadora frente aos concorrentes. Na Figura 2 são mostradas diversas montadoras que utilizam esse tipo de documentação, possuindo cada uma sua especificidade.

Figura 2 - Grande parte das montadoras automotivas.

Fonte: AutoEsporte (2019).

Ciente da crescente exigência de um mercado cada vez mais disputado e da necessidade das companhias em atendê-lo, a proposta do presente trabalho é apresentar uma metodologia de desenvolvimento de algumas VFs, desde suas especificações e requisitos, tais como insumos eletrônicos entre sensores e atuadores, análise de relevância e por fim, a simulação computacional dos sistemas requeridos, bem como a prototipagem em escala reduzida das mesmas. Para tal, a modelagem é construída a partir do *software* Matlab/*Simulink* e, juntamente com a plataforma Arduino, ocorre a validação dos sistemas por meio da montagem dos protótipos.

1.1 Justificativa

A indústria automotiva desempenha um papel muito importante no cenário econômico, tanto nacional quanto mundial. No Brasil a geração de empregos foi de 130,4 mil com o fechamento em dezembro de 2018, 1,7 % a mais do que no mesmo período de 2017. O mercado interno foi o grande destaque de 2018, com quase 2,5 milhões de veículos leves licenciados, 13,8% a mais que em 2017. Foi o segundo ano seguido de alta, mostrando que a crise ficou para trás (ANFAVEA, 2019).

Este setor é responsável também pelo crescimento da inovação, pois é precursora na implantação de novos processos produtivos, investindo grandes volumes de capital em Pesquisa e Desenvolvimento (P&D) atraindo diversas outras indústrias para o seu entorno (FRAINER, 2010).

Como muitos outros setores, a indústria automobilística está expandindo rapidamente a utilização de sistemas e componentes eletrônicos. Praticamente todas as funções dos veículos modernos, como aceleração, frenagem, controles de tração, de estabilidade e de injeção de combustível (incluindo injeção eletrônica), sistemas de combustão *lean-burn*, dirigibilidade, segurança, ajuste da posição da direção e dos bancos, navegação, proteção anti-choque, sistemas de controle de voz e entretenimento, já são controladas e/ou viabilizadas pela eletrônica embarcada (CARVALHO, 2008).

Os sistemas embarcados automotivos são constituídos por vários componentes, os quais são desenvolvidos por diferentes fabricantes. Assim a interação destes se tornam um pouco mais trabalhosa devido à utilização de padrões e normas distintas de construção. Desta forma, a indústria vem buscando desenvolver padrões para a construção destes componentes de forma a facilitar e agilizar a gestão dos projetos (SILVA, 2017). Consequentemente a busca pelo

desenvolvimento de novas VFs se torna cada vez maiore, uma vez que diversos requisitos de conforto e segurança estão sendo desenvolvidos. O dinamismo no desenvolvimento das VFs impacta diretamente no tempo e custo do desenvolvimento do componente. A Tabela 1 trata-se de uma revisão bibliométrica realizada visando a busca por trabalhos de quaisquer categorias sobre o tema abordado.

Tabela 1 - Revisão bibliométrica

Palavras procuradas	**Quantidade de trabalhos**
'Vehicular' 'functions'	659.000
'Vehicular' 'functions' 'simulink'	12.100
'Vehicular' 'functions' 'matlab'	38.900
***'Vehicular' 'functions'* 'arduino'**	2.480
***'Vehicular' 'functions'* 'arduino' 'matlab'**	3.900
***'Vehicular' 'functions'* 'arduino' 'matlab' '*simulink'**	8.390
'Desenvolvimento' '*Vehicular' 'functions'* 'arduino' 'matlab' '*simulink'*	187
'simulação' 'funções' 'veicular'	8.770
'simulação' 'funções' 'veicular' 'arduino'	130
'simulação' 'funções' 'veicular' 'arduino' 'matlab' '*simulink'*	96
'funções' 'veicular' 'arduino' 'matlab' '*simulink'*	68

Fonte: Autor

Pela análise da Tabela 1 vale ressaltar que a quantidade de trabalhos na área é baixa, assim como que nenhum trabalho foi encontrado especificamente do tema abordado por este trabalho de conclusão de curso, acredita-se que devido ao sigilo requerido para tal tipo de documentação.

A falta de literatura disponível para o desenvolvimento de uma VF fez com que surgisse o interesse pelo tema, por que a descrição de uma VF é um ponto de extremo cuidado das montadoras, e trata-se de uma competência importante para um engenheiro que atue neste setor.

1.2 Objetivos

Nesta seção são apresentados os objetivos gerais e específicos deste trabalho de conclusão de curso (TCC).

1.2.1 Objetivos gerais

Este trabalho tem como objetivo geral apresentar uma metodologia de desenvolvimento de VFs, assim como validação da funcionalidade através de ensaios práticos e construção de sua documentação de forma clara e objetiva. Para isso, construiu-se um modelo computacional a partir da ferramenta *Simulink* e, para comprovação, sua aplicação por meio da plataforma de prototipagem Arduino. A documentação é embasada em experiências de pesquisadores do Grupo de Pesquisa CNPq GSE (Grupo de Soluções em Engenharia).

1.2.2 Objetivos específicos

Para estabelecer os objetivos gerais, de forma sequencial alguns objetivos específicos devem ser alcançados, sendo estes apresentados a seguir:

- Levantamento bibliográfico sobre funções veiculares, simulações no MATLAB/*Simulink*;
- Definir o modelo que represente corretamente o funcionamento das VFs;
- Estudo e análise de implementação das VFs via *Simulink*;
- Estudo e análise de implementação de VFs via plataforma Arduino;
- Montagem do modelo em *protoboard*;
- Análise e testes de validação das VFs modeladas a partir de testes práticos em escala reduzida a partir da plataforma Arduino;
- Documentação clara e objetiva das VFs.

1.3 Estrutura do trabalho

O presente trabalho será composto por 5 capítulos, sendo o primeiro responsável pela introdução ao tema, justificativa para o desenvolvimento do trabalho, descrição dos objetivos gerais e específicos e sua estrutura.

O segundo capítulo trata-se do referencial teórico, o qual descreve em quais trabalhos e artigos o trabalho foi embasado, tratando de temas como desenvolvimento de VFs, identificação dos componentes que compõe um sistema embarcado automotivo, simulação em *Simulink* e prototipagem com uso da plataforma Arduino.

O terceiro capítulo trata-se da metodologia para o desenvolvimento do trabalho. Descreve-se os passos que foram seguidos para a implementação, desde a estrutura de uma VF,

passando pelas simulações até a aplicação prática na plataforma Arduino. Por fim, realizar sua documentação.

No quarto capítulo ocorre a apresentação dos resultados das simulações e do funcionamento dos sistemas embarcados, de forma bem explicativa para que o leitor possa compreender todo o funcionamento por trás das VF's que foram desenvolvidas. Vale ressaltar que por se tratar de uma documentação sigilosa, VFs, é extremamente difícil encontrar na literatura trabalhos sobre esse tema para efeito comparativo e validação da proposta.

A conclusão e considerações finais são apresentadas no quinto capítulo, assim como uma proposta de continuidade do tema para trabalhos futuros.

2 REFERENCIAL TEÓRICO

O referencial teórico busca apresentar o embasamento necessário para o desenvolvimento do trabalho. Com isso, este capítulo apresenta conceitos importantes sobre sistemas embarcados automotivos, descrição e estrutura de uma VF.

2.1 Automatização do setor veicular

Nos dias atuais, os Sistemas Embarcados (SEs) estão presentes em praticamente todos os equipamentos eletrônicos disponibilizados no mercado. Com a crescente evolução da tecnologia por trás desses sistemas, SEs fazem parte de todos os tipos de equipamentos, sejam simples ou complexos e buscam sempre maior comodidade e confiança para quem usufruir de suas funcionalidades (CARRO; WAGNER, 2003).

No setor automotivo, área de grande impacto na economia mundial, não poderia ser diferente. Este está recebendo cada vez mais atenção por parte das montadoras tradicionais e detentores de tecnologias inovadoras na área (DOS ANJOS, 2011). Com isso, espera-se que este mercado seja ainda mais revolucionário na próxima década, de tal forma que aumente, consequentemente, os níveis de automação implementados nos veículos.

A *Society of Automotive Engineers* (SAE) estabelece níveis de inteligência e as capacidades de automação dos veículos, classificando-os de 0 a 5. De acordo com Napol (2017) estes níveis caracterizam os veículos que possuem recursos totalmente manuais a totalmente autônomos, como segue:

Nível 0: Veículo totalmente manual

O SAE nível 0 é responsável pela maioria dos veículos em circulação, com todos os aspectos da direção sendo totalmente humanos e controlados manualmente.

Nível 1: Um único aspecto automatizado

O SAE nível 1 é o nível mais baixo de automação. O veículo possui um único aspecto de automação que ajuda o motorista. Exemplos disso incluem controle de direção, velocidade ou frenagem, mas nunca mais que um deles.

Nível 2: Recursos automatizados de direção e aceleração

O SAE nível 2 é onde o veículo é capaz de controlar os recursos de direção e aceleração/desaceleração. Embora isso permita ao veículo automatizar certas partes da experiência de dirigir, o motorista permanece sob controle completo do veículo o tempo todo.

Destacam-se aplicações que auxiliam os veículos a permanecerem nas faixas e os recursos de estacionamento.

Nível 3: Detecção de ambiente

Capazes de detectar o ambiente ao seu redor. Os veículos de nível 3 contêm o sistema de nível mais baixo que é classificado como um sistema de direção automatizado, em oposição a um sistema manual. Estes veículos podem tomar decisões informadas por si mesmos, como ultrapassar veículos em movimento mais lento. No entanto, diferentemente dos veículos autônomos com classificação mais alta, a substituição humana é necessária quando a máquina não consegue executar a tarefa em mãos ou em falhas do sistema.

Nível 4: Nenhuma interação humana é necessária

A principal diferença entre a automação de nível 3 e de nível 4 é que os veículos de nível 4 são capazes de intervir se algo der errado ou ocorrer uma falha no sistema. Nesse sentido, esses carros são deixados completamente por conta própria, sem qualquer intervenção humana na grande maioria das situações, embora a opção de substituir manualmente permaneça em circunstâncias difíceis ou preferíveis.

Nível 5: A condução humana é completamente eliminada

Da mesma forma, veículos de nível 5 não requerem atenção humana. No entanto, a principal diferença é a qualidade, com os veículos de nível 5 oferecendo um serviço muito mais ágil e refinado, comparável ao da condução manual adaptativa e situacional. Exemplos de onde os veículos de nível 5 se destacam incluem a condução fora de estrada e outros terrenos que os veículos de nível 4 podem não ser capazes de detectar ou compreender de forma inteligente.

Em outras palavras, os veículos de nível 5 têm um sistema de detecção de ambiente muito mais avançado. Essa é a única classe de veículos automatizados que não apresenta controles de direção típicos, como volantes, pedais de acelerador e freio, ou outros, com o motorista completamente eliminado.

Quanto maior o nível de automação presente em um veículo, mais SEs o mesmo carrega, juntamente com um grande pacote de dados para a comunicação destes sistemas. Assim, pode-se destacar a grande importância e responsabilidade que este mecanismo muito simples e funcional representa no meio automotivo. Estes sistemas serão tratados na próxima seção.

2.2 Sistemas Embarcados

Um Sistema Embarcado (SE) é um sistema microprocessado completamente dedicado ao dispositivo ou sistema que ele controla. Diferentemente de um computador pessoal, um SE

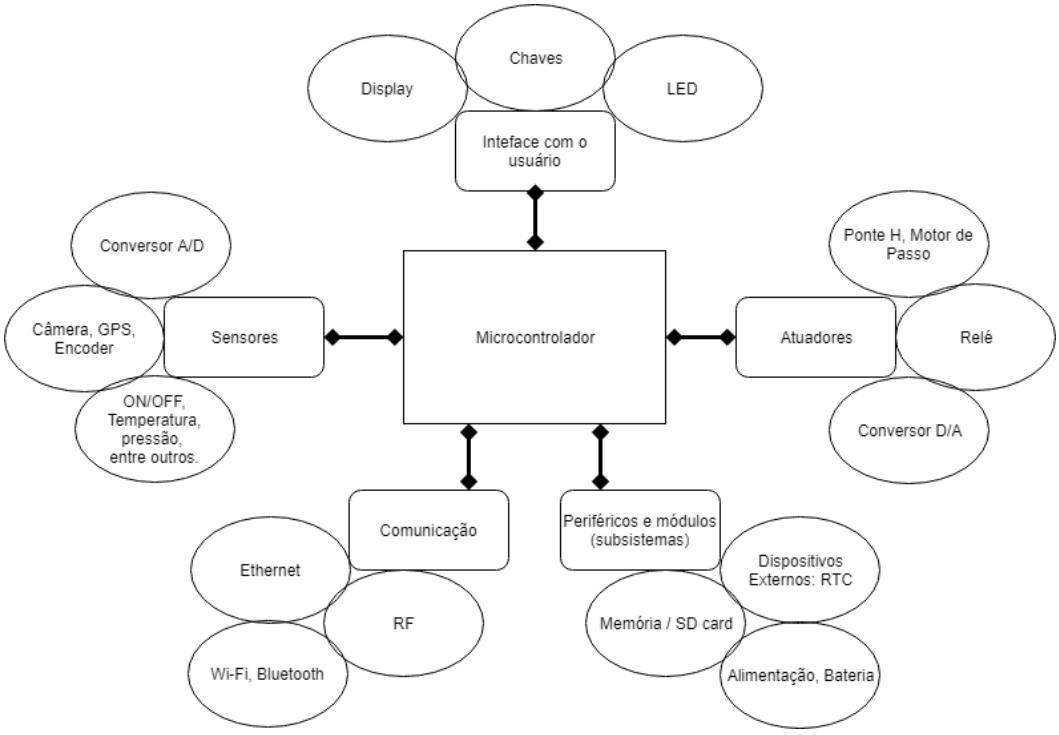
realiza um conjunto de tarefas predefinidas, geralmente com requisitos específicos. Já que o sistema é dedicado a tarefas específicas, é possível através de engenharia otimizar o projeto reduzindo tamanho, recursos computacionais e custo.

Com base em uma definição mais específica sobre SEs, temos que:

> Colocar capacidade computacional dentro de um circuito integrado, equipamento ou sistema. Esta é uma definição para o que é um sistema embarcado. Note que um sistema como este deve ser mais do que um simples computador. É um sistema completo e independente, mas preparado para realizar apenas uma determinada tarefa (CUNHA, 2011, p.1).

Os SEs respeitam uma estrutura básica, composta por sensores, microcontrolador e atuadores. A Figura 3 destaca essa estrutura com alguns componentes que podem exercer tais funções.

Figura 3 - Estrutura e composição de um SE.



Fonte: Garcia (2019).

Desta forma, os principais integrantes deste tipo de estrutura serão tratados nas próximas subseções.

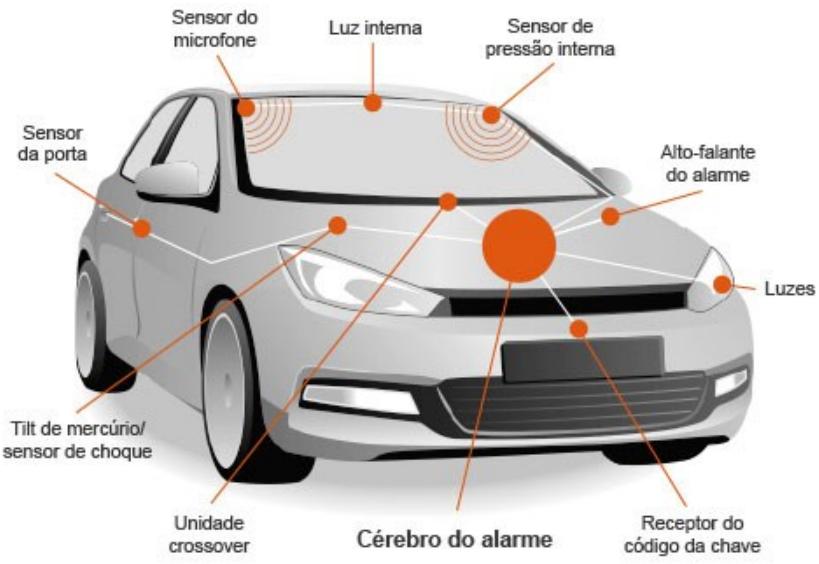
2.2.1 Sensores

Os sensores são dispositivos sensíveis a algum tipo de variação de energia, seja ela luminosa, térmica, cinética e são capazes de relacionar essa energia à algum tipo de grandeza que pode ser medida, como por exemplo, velocidade, temperatura, pressão, corrente, tensão, etc (WENDLING, 2010).

A interação ou comunicação entre o controlador em um sistema de controle, por exemplo, muita das vezes não é possível ser realizada de forma direta. O sinal que o sensor emite geralmente é manipulado. Essa função normalmente é desempenhada por um circuito de interface que adequa o sinal de tal forma que este possa ser lido pelo controlador (WENDLING, 2010).

Em sistemas automotivos, os sensores possuem a capacidade de identificar determinados sinais e transformá-los em informação útil ao motorista, de tal forma que a condução fique mais segura e facilite a manutenção dos veículos por meio de diagnósticos precisos. Cada sensor trabalha como um coletor de dados para o sistema eletrônico do carro. Esses dados são processados e comparados com diversos padrões (RODRIGUES, 2017). Dependendo do resultado, são gerados avisos, que podem ser sonoros ou luminosos, juntamente com uma informação no painel do carro ou uma gravação na central eletrônica para uma futura consulta. A Figura 4 apresenta alguns dos sensores presentes em sistemas automotivos, mais especificamente para a parte de proteção por alarme.

Figura 4 - Sensores presentes em um sistema de alarme.



Fonte: *Connect parts* (2019).

2.2.2 Atuadores

Um atuador é um componente de um SE responsável por controlar um mecanismo ou um sistema, sendo operado por corrente elétrica, fluído hidráulico ou pressão pneumática. Estas fontes de energia são convertidas em sinais elétricos digitais ou analógicos ou em movimentos mecânicos (DOS ANJOS, 2011). Estão presentes em praticamente todas as funções implementadas em um veículo. A Figura 5 apresenta alguns atuadores utilizados em sistemas automotivos, mais especificamente em um sistema de injeção eletrônica.

Figura 5 - Atuadores no sistema de injeção eletrônica veicular.

Fonte: Adaptado de Rabelo (2017).

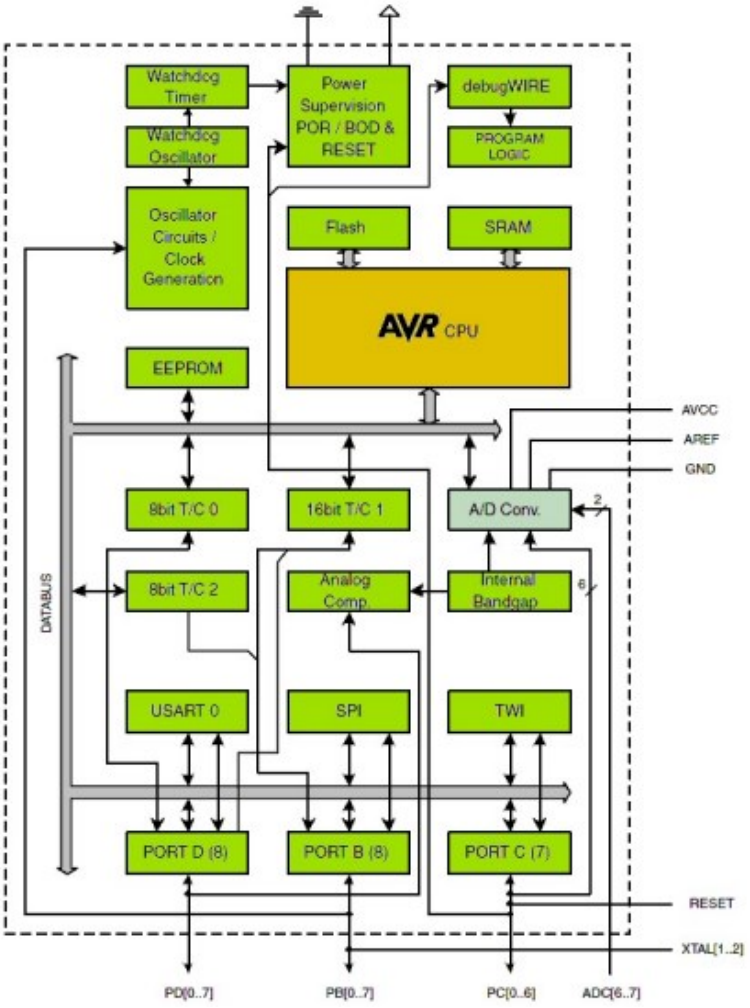
2.2.3 Microcontroladores

Um microcontrolador é uma unidade de processamento que traz todos os módulos necessários ao seu funcionamento (como memória RAM, memória ROM, blocos de entrada e saída, conversores A/D e D/A, barramentos para troca de dados) no interior de um único *chip*, diferentemente de um microprocessador, podendo assim ser programado para executar uma função específica em determinado componente (BENTES, 2013).

Em comparação aos microprocessadores, pode-se constatar que os microcontroladores operam em uma frequência muito baixa, porém adequada a maioria das aplicações as quais são designados. Devido a esta característica, o consumo de energia é pequeno, normalmente em torno de 50 mW. Além disso os microcontroladores podem entrar em modo de espera, aguardando por um evento externo, tal como pressionar de uma tecla ou acionamento de um sensor (BENTES, 2013).

As aplicações deste componente compreendem principalmente automação e controle de produtos periféricos, como sistemas de controle de motores automotivos, máquinas industriais, de escritório e residenciais, entre outros. Por possuírem tamanho, custo e consumo de energia reduzidos se comparados aos microprocessadores, é uma alternativa eficiente para controlar processos e aplicações. A Figura 6 mostra um esquemático de um microcontrolador da fabricante Atmel. Ele possui um único encapsulamento e os componentes básicos ao seu funcionamento estão todos embutidos.

Figura 6 - Microcontrolador Atmel.



Fonte: Bentes (2013).

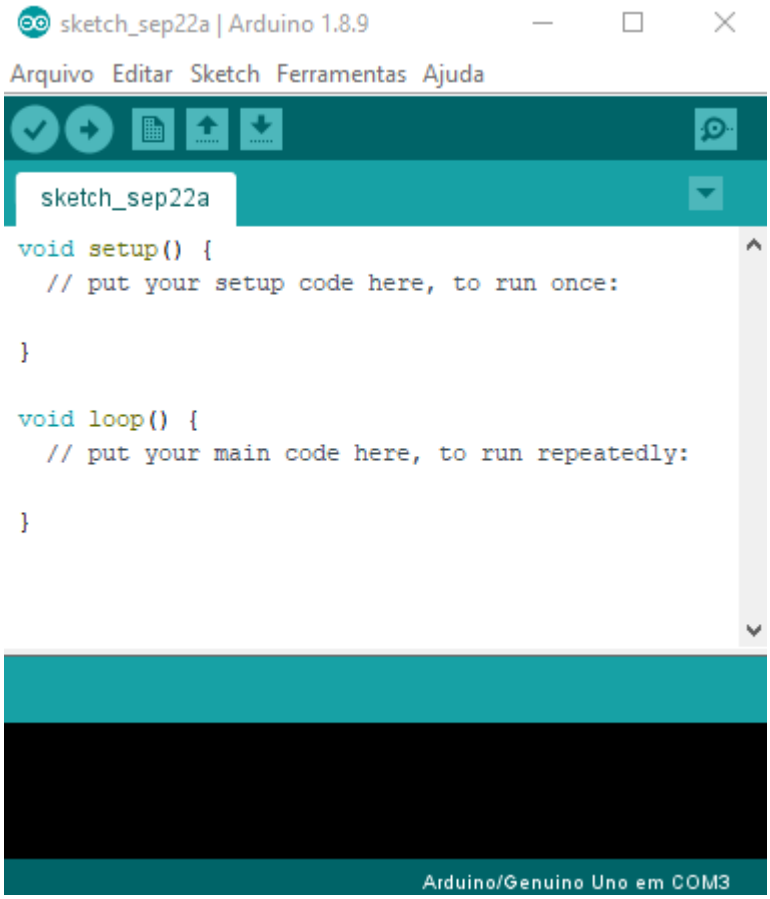
Dentro deste contexto, na próxima subseção será tratado o microcontrolador foco deste trabalho que é o Arduino. Serão apresentadas suas principais características.

2.2.3.1 Plataforma de prototipagem Arduino

A plataforma Arduino surgiu em 2005, desenvolvida por Massimo Benzi, David Cuartielles, Tom Igoe, Gianluca Martino e David Mellis, com o objetivo de criar um dispositivo mais acessível e simples do que os disponíveis no mercado, para controlar projetos que lidem com sistemas microcontrolados (SILVA et al., 2014). Com isso, ela trabalha com o conceito de *software* livre, onde as informações são disponibilizadas de forma irrestrita sobre o projeto do *hardware* e diagramas eletrônicos.

A placa do Arduino é composta de um microcontrolador Atmel AVR, um cristal ou oscilador (responsável pelo envio dos pulsos de *clock*) e um regulador linear de 5 V. A placa expõe os pinos de entrada/saída do microcontrolador, para que se possa conectá-los a outros circuitos ou sensores (MCROBERTS, 2011) e uma linguagem de programação padrão baseada em C/C++. A Figura 7 apresenta a IDE de programação da plataforma Arduino.

Figura 7 - IDE de programação da plataforma Arduino.



Fonte: Autor.

Desde o seu surgimento até os dias atuais, várias versões da placa foram desenvolvidas, cada uma com um nome e *hardware* específicos, seguindo algumas características em sua construção, tais como:

- Microcontrolador;
- Placa base com reguladores de tensão adequados ao microcontrolador;
- Linhas de entrada e saída digitais e analógicas;
- Linhas de comunicação serial;
- Interface USB para programação e interação com um computador hospedeiro.

A Tabela 2 apresenta algumas características de algumas placas desenvolvidas ao longo dos anos.

Tabela 2 - Características de algumas placas Arduino construídas

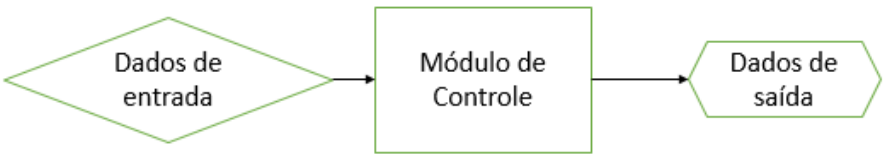
Modelo	**Microcontrolador**	**E/S Digitais**	**E/S Analógicas**	**Alimentação**	***Clock***	**Memória**
Arduino Uno	ATmega328	14	6	7-12 V	16M Hz	32 K
Arduino Mega	ATmega2560	54	16	7-12 V	16 MHz	256 K
Arduino Leonardo	ATmega32u4	20	12	7-12 V	16 MHz	32 K
Arduino Due	AT91SAM3X8E	54	12	7-12 V	84 MHz	512 K
Arduino ADK	ATmega2560	54	12	7-12 V	16 MHz	256 K
Arduino Nano	ATmega168	14	16	7-12 V	16 MHz	16 K
Arduino Pro Mini	ATmega168	14	8	3.35-12 V	8 MHz	16 K
Arduino Esplora	ATmega32u4	-	8	7-12 V	16 MHz	32 K

Fonte: Adaptado de: Lima (2016).

2.3 Estrutura de uma VF

A partir de reuniões e discussões com pesquisadores do GSE, pôde-se observar e concluir que uma VF se estrutura por meio de um circuito eletrônico, compondo-se por dados de entrada, um dispositivo controlador que avalia e interpreta os dados de entrada para assim coordenar os dados de saída. A Figura 8 exemplifica essa estrutura.

Figura 8 - Estrutura básica de uma VF.



Fonte: Autor.

Pode-se destacar pela Figura 8 que a estrutura é similar à de um SE genérico, mas a VF descreve o comportamento e a integração dos SEs com os demais componentes do sistema integrado. Os dados de entrada são obtidos geralmente por sensores e os dados de saída são comandos para os mais diversos atuadores. O módulo de controle é onde especifica-se o microcontrolador, sendo este programado para atender uma única função.

2.4 *Model Based Design* (MBD)

O MDB fornece uma abordagem matemática e visual para desenvolver sistemas complexos de controle e processamento de sinais. Ele se concentra no uso de modelos de sistema em todo o processo de desenvolvimento para *design*, análise, simulação, geração automática de código e verificação (MATHWORKS, 2019).

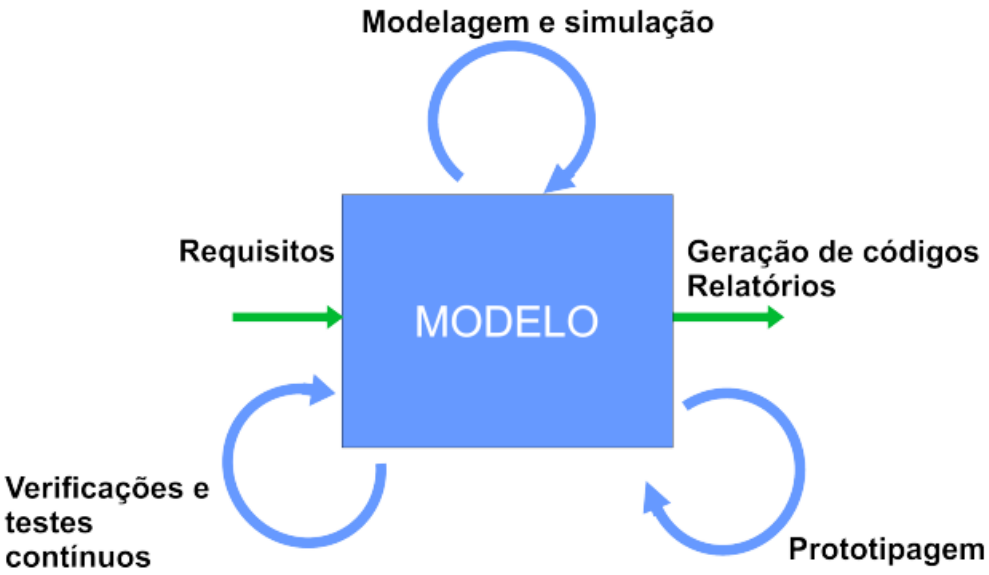
Um modelo é criado para especificar o comportamento de um sistema incorporado. O modelo, que consiste em diagramas de blocos, programas textuais e outros elementos gráficos, permite a execução de simulações para testar ideias e verificar projetos ao longo do processo de desenvolvimento (MATHWORKS, 2019). Apresenta diversos benefícios para o desenvolvedor, tais como:

- O *design* pode ser testado, refinado e testado novamente durante todo o processo de desenvolvimento;

- É menos trabalhoso tentar novas ideias, porque não há necessidade de criar protótipos;
- O teste e a validação são feitos continuamente, e não no final do processo, para que muitos erros sejam encontrados e corrigidos antes dos testes de *hardware*;
- O código incorporado pode ser gerado automaticamente a partir do modelo do sistema, o que reduz o esforço e elimina erros de codificação manual. Este código pode ser usado para testar simulações em tempo real;
- Os modelos podem ser adaptados e reutilizados em projetos subsequentes.

Na metodologia do MBD, muitas dessas técnicas são descartadas no meio industrial, uma vez que o modelo do sistema é mantido no centro do processo durante todas as etapas de desenvolvimento. A Figura 9 ilustra este conceito, onde a partir dos requisitos do projeto até a geração automática de códigos, ocorre etapas como a implementação do código, testes e verificações contínuas, mas o modelo é sempre mantido no centro do processo (SILVA, 2017).

Figura 9 - Fases de desenvolvimento do MBD com o modelo no centro.



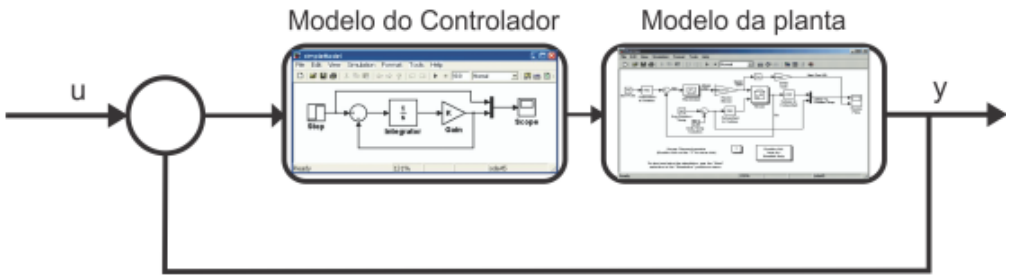
Fonte: Silva (2017).

No MBD ocorre a aplicação dos processos *Model-in-Loop* (MIL), *Software-in-Loop* (SIL), *Processor-in-Loop* (PIL) e *Hardware-in-Loop* (HIL) como métodos de verificação e validação. Este tipo de prática é comum nos simuladores de aplicações baseadas em modelos e permite a descoberta de erros básicos. Nas subseções a seguir serão descritas as características de cada um desses processos (SILVA, 2017).

2.4.1 *Model-in-Loop*

Esse processo consiste em desenvolver um modelo da planta real em qualquer ambiente de simulação, por exemplo, o Simulink, que captura a maioria dos recursos importantes do sistema de *hardware*. Após a criação do modelo da planta, ocorre a escolha do modelo do controlador de forma que ele possa controlar a planta conforme a exigência. Essa é a etapa em que ocorre o teste da lógica do controlador no modelo simulado da planta. Se o controlador funcionar como desejado, ocorre o registo das entradas e saídas do controlador para que estas possam ser usadas no estágio posterior da verificação (MATHWORKS, 2019), que será descrito na próxima subseção. A Figura 10 descreve este processo.

Figura 10 - *Model-in-loop.*

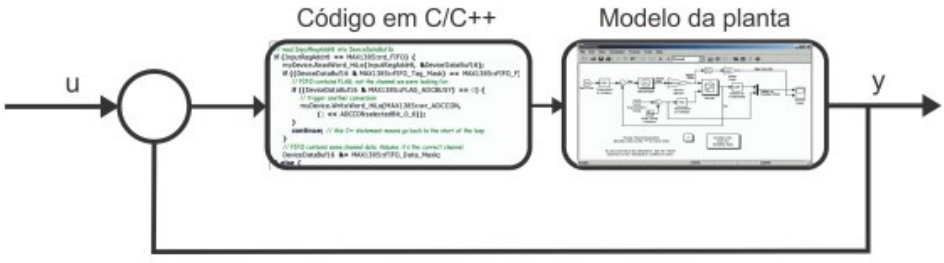


Fonte: Silva (2017).

2.4.2 *Software-in-Loop*

Esse é o próximo estágio que consiste na geração de um código (em C ou outra linguagem) para representar um modelo da aplicação, conectado a um modelo de planta física. Também não é utilizado *hardware* neste método. É utilizado para testar o funcionamento da aplicação com funções de tempo real, para verificar a validade do modelo contra as especificações (NUNES,2017). A Figura 11 descreve o *Software-in-Loop*.

Figura 11 - *Software-in-loop*.

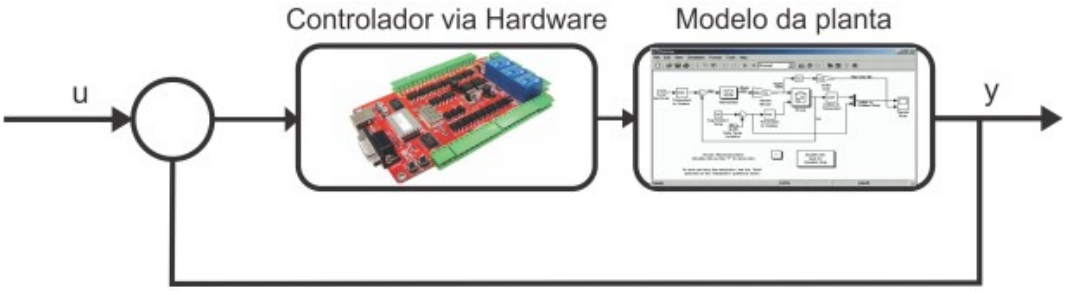


Fonte: Silva (2017).

2.4.3 *Processor-in-Loop*

Esta etapa consiste em colocar o modelo do controlador em uma placa de processador incorporada e executar uma simulação de *loop* próximo com a planta simulada. Esta análise ajudará a identificar se a placa do processador é capaz de executar a lógica de controle desenvolvida. Se houver falhas, é necessário retornar ao SIL ou MIL e retificá-las (MATHWORKS, 2019). A Figura 12 representa o *Processor-in-Loop*.

Figura 12 - *Processor-in-loop.*

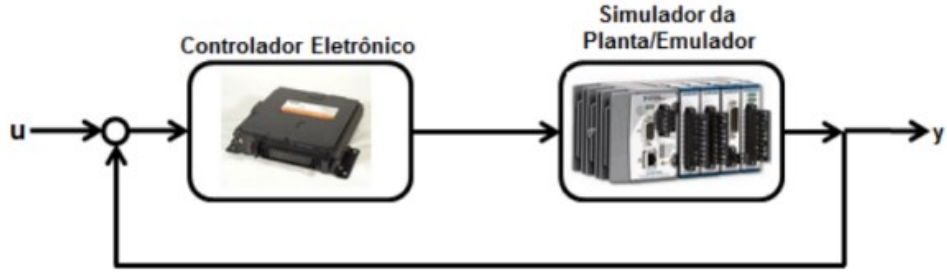


Fonte: Silva (2017).

2.4.4 *Hardware-in-Loop*

Depois de verificar o modelo da sua planta usando o PIL, ocorre a substituição do modelo da planta pelo hardware original e em seguida o sistema é testado. É um método que permite buscar erros de integração, que podem envolver falhas de segurança ou custos elevados de reparo durante os testes reais (NUNES, 2017). A Figura 13 descreve o *Hardware-in-Loop*.

Figura 13 - *Hardware-in-loop.*



Fonte: Silva (2017).

2.5 Trabalhos relacionados

Nesta seção, serão apresentados alguns trabalhos semelhantes e que nortearam o desenvolvimento deste trabalho durante o levantamento bibliográfico.

Silva (2017) destaca que os sistemas embarcados automotivos são sistemas mecatrônicos complexos e que exigem aplicação de métodos e ferramentas nas fases de criação e testes. Assim, há um fluxo de processos para desenvolvimento de funções de um sistema automotivo. Em seu trabalho, ele realiza um estudo de caso sobre as fases de desenvolvimento de *software* automotivo e propõe a criação de modelos de controle para funções relacionadas a carroceria de um veículo. Para isso ele utiliza as ferramentas da Mathworks® e realiza as duas primeiras etapas de testes, as quais consistem no desenvolvimento do *software* de controle e sua implementação.

Banik (2017) destaca que versatilidade, rapidez e robustez são pautas que estão inteiramente ligadas ao desenvolvimento de novas funções automotivas. Com o consumidor exigindo cada vez mais conforto e inovação, os desenvolvedores precisam estar atentos a cada possibilidade de mudança, a fim de alcançar novos patamares na programação. Em seu trabalho ela visa aplicar o MDB na construção de uma plataforma móvel, idealizada inicialmente pela Marcopolo S.A, que é uma empresa transnacional brasileira fabricante de carrocerias de ônibus. Ela realiza diversos testes de verificação a fim de aumentar a robustez do sistema e torna-lo compatível com a ISO26262, uma norma que trata da segurança funcional de *software* embarcado. Após as verificações, ocorre a migração do sistema para a arquitetura automotiva AUTOSAR com o intuito de simular seu funcionamento.

Neme, Santos e Teixeira (2015) destacam a aplicação MBD para o *design* do sistema de iluminação exterior automotivo. É um sistema de controle orientado a eventos normalmente necessário para iluminação e sinalização de automóveis, principalmente à noite. Além disso, esse sistema é obrigatório para qualquer veículo rodoviário de acordo com as leis e leis brasileiras vigentes. Tem como semelhança para este trabalho as ferramentas utilizadas e de forma indireta, tudo o que foi desenvolvido por eles passam primeiramente pela criação de uma VF.

Aqui pode-se destacar mais uma vez que a literatura por trás da proposta deste trabalho é escassa, assim comprova-se novamente o sigilo que as montadoras possuem em relação a esse tema e suas tecnologias desenvolvidas.

3 DESENVOLVIMENTO

Este capítulo tem como objetivo estabelecer a sequência de procedimentos para o desenvolvimento de aplicações para SEs, com base nas práticas e ferramentas adotadas pelas grandes empresas do segmento automotivo, tendo como principal referência experiência de pesquisadores do GSE, atuantes como engenheiros de empresas do ramo automotivo. Na próxima subseção serão tratadas as principais ferramentas utilizadas para o desenvolvimento deste TCC.

3.1 Ferramentas para desenvolvimento

As ferramentas para desenvolvimento foram escolhidas ao decorrer do andamento do trabalho. Esta subseção tem por finalidade apresentar características gerais de cada uma, e ao longo do texto serão destacadas quando e como foram utilizadas.

3.1.1 Arduino Uno

Para testes em *protoboard* das VFs implementadas optou-se pela placa Arduino Uno por atender suficientemente as necessidades impostas para o trabalho, como a quantidade E/S digitais e analógicas, portas PWM e comunicação com *software* Matlab/Simulink.

A placa é baseada no microcontrolador ATmega328 e possui 14 pinos de E/S digitais, onde 6 dessas portas podem ser utilizadas como saídas PWM, um *clock* de 16MHz que define e frequência de operação do microcontrolador, conexão USB, pino de alimentação externa e botão de *reset*, como apresentado na Figura 14. Ela possui todos os circuitos integrados na placa, assim faz com que o usuário necessite apenas inserir seus componentes e embarcar o código de forma correta para atender a funcionalidade do projeto desejado. Como ela possui um gerenciador de inicialização do microcontrolador, *bootloader*, o processo de gravação é facilitado (ARDUINO, 2019).

Figura 14 - Composição da placa Arduino Uno.

Fonte: *Hackerearth* (2016).

3.1.2 Tinkercad

O Tinkercad é um aplicativo gratuito para projetos 3D, componentes eletrônicos e codificação. Trata-se de *software* desenvolvido e disponibilizado pela Autodesk que permite realizar simulação da plataforma Arduino. É um dos poucos *softwares* que gratuitamente possui essa funcionalidade (AUTODESK, 2014). O logo da plataforma é apresentado na Figura 15.

Figura 15 - Logo Tinkercad.

Fonte: Autodesk (2019).

3.1.3 Simulink

O Simulink é um ambiente de diagrama de blocos para simulação de vários domínios e *design* baseado em modelo. Ele suporta *design* no nível do sistema, simulação, geração automática de código e teste e verificação contínuos de sistemas embarcados. O Simulink fornece um editor gráfico, bibliotecas de blocos personalizáveis e solucionadores para modelar e simular sistemas dinâmicos. Ele é integrado ao MATLAB, permitindo incorporar algoritmos MATLAB em modelos e exportar resultados de simulação para o MATLAB para análises adicionais (MATHWORKS, 2019).

Possui diversas características das quais podemos destacar:

- Editor gráfico para criar e gerenciar diagramas de blocos hierárquicos;
- Bibliotecas de blocos predefinidos para modelagem de sistemas de tempo contínuo e de tempo discreto;
- Escopos e exibições de dados para visualização dos resultados da simulação;
- Bloco de funções MATLAB para importar algoritmos MATLAB para modelos;
- Ferramenta de código herdado para importar código C e C ++ para modelos.

3.1.4 *Stateflow*

O *Stateflow* é um *toolbox* do MATLAB que é utilizada para a criação de diagramas de estado, diagramas de fluxo e tabelas verdade dentro do Simulink. Esse recurso auxilia a programação e o desenvolvimento de lógicas de controle de forma visual (Tutorial FIAT, 2017).

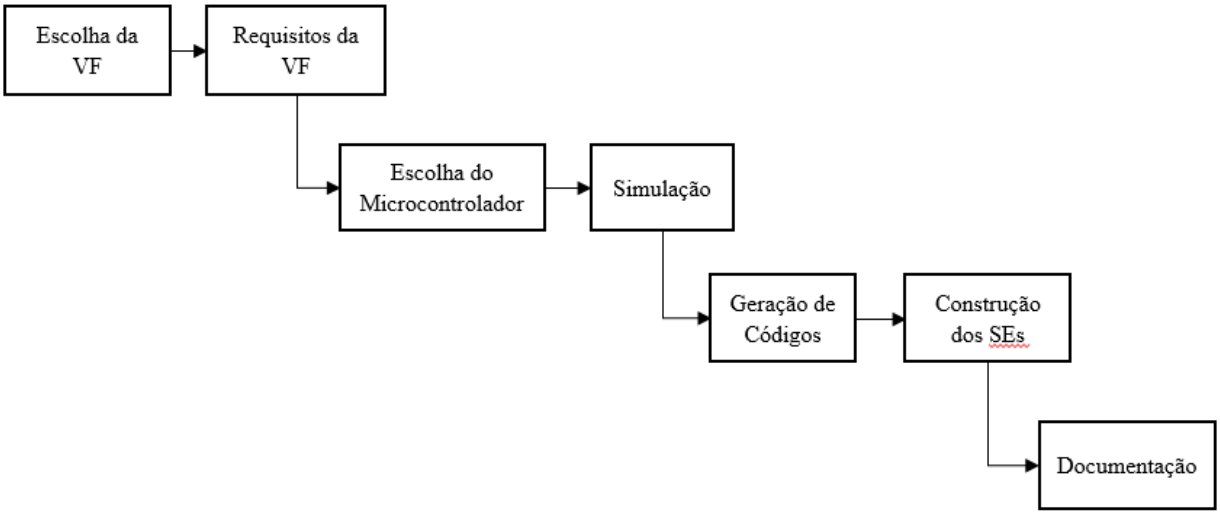
Pode ser usado para descrever algoritmos MATLAB e os modelos Simulink reagem aos sinais de entrada, eventos e condições baseadas no tempo. Permite projetar e desenvolver controle de supervisão, agendamento de tarefas, gerenciamento de falhas, protocolos de comunicação, interfaces de usuário e sistemas híbridos (MATHWORKS, 2019).

Com o *Stateflow*, é possível modelar a lógica a qual pode ser simulada como um bloco em um modelo do Simulink ou executada como um objeto no MATLAB. A animação gráfica permite analisar e depurar sua lógica durante a execução. As verificações de tempo de edição e tempo de execução garantem a consistência e a integridade do design antes da implementação (MATHWORKS, 2019).

3.2 Metodologia

A metodologia para o desenvolvimento deste TCC segue as etapas apresentadas na Figura 16 que representa o fluxo de desenvolvimento das atividades.

Figura 16 - Etapas do desenvolvimento.



Fonte: Autor.

3.2.1 Informações e pontos importantes

Primeiramente, após reuniões e discussões com pesquisadores do GSE foram definidas duas VFs para o desenvolvimento. Todas as VFs possuem informações necessárias para sua escrita, tais como:

- Nome: se baseia nos equipamentos a serem atendidos pela função, contendo o número da VF, sua versão e as revisões que ela sofreu. Exemplo: Gerenciamento das luzes de freio [VF020_V1_R1].
- Área de atuação: os veículos possuem funções específicas para cada parte do veículo, como descrito na seção de introdução deste trabalho. Então a área de atuação está ligada diretamente quais partes do veículo a VF irá atender, como por exemplo, equipamentos ligados a carroceria, equipamentos ligados ao motor, equipamentos ligados ao sistema de transmissão, etc.
- Descrição: essa característica é responsável por descrever quais instrumentos a VF irá atender. Está diretamente relacionada ao nome da VF.
- Componentes: de forma quantitativa, os componentes da VF são especificados após ter uma certeza de onde ela irá atuar. Esta é uma das partes mais importantes devido os componentes serem de fabricantes diferentes como já citado na seção de introdução de trabalho.
- Entradas / Saídas: como se trata de um sistema embarcado, é necessário saber quais tipos de E/S este possui para um correto tratamento das variáveis pelo

microcontrolador. Elas podem ser analógicas ou digitais, respeitando o número de portas disponível pelo microcontrolador.

- Atuação: descreve como as entradas e saídas irão se comportar no sistema e a quais instrumentos as variáveis estão relacionadas dentro do sistema veicular. Como por exemplo, uma chave que pode representar um pedal de freio, com seu estado normalmente aberto quando o pedal não está pressionado e seu estado normalmente fechado quando o pedal está pressionado.

Estas informações estão ligadas diretamente com a escolha do microcontrolador devido a necessidade de levantar alguns pontos importantes relacionados com a funcionalidade da VF. Estas características ou pontos principais são: capacidade de memória, quantidade de entradas e saídas, tanto digitais quanto analógicas, e sua resolução.

A memória do microcontrolador tem que ser grande o suficiente para que o código gerado possa ser embarcado no mesmo. As entradas digitais e analógicas são responsáveis pela leitura e conversão dos sinais recebidos pelo microcontrolador a partir dos demais componentes do sistema. É necessário saber a sua resolução para uma devida tomada de decisão na construção do *software*, para que atenda todas as características necessárias.

3.2.2 Modelagem e geração de códigos no Simulink

As simulações para as funções escolhidas para serem apresentadas neste TCC seguiram o mesmo padrão de desenvolvimento.

Após o levantamento das principais informações de cada VF, o início da simulação se dá a partir da construção do diagrama de estados utilizando o *toolbox* Stateflow que é integrado ao *software* Simulink. Nele é construída toda a lógica por trás do funcionamento da VF.

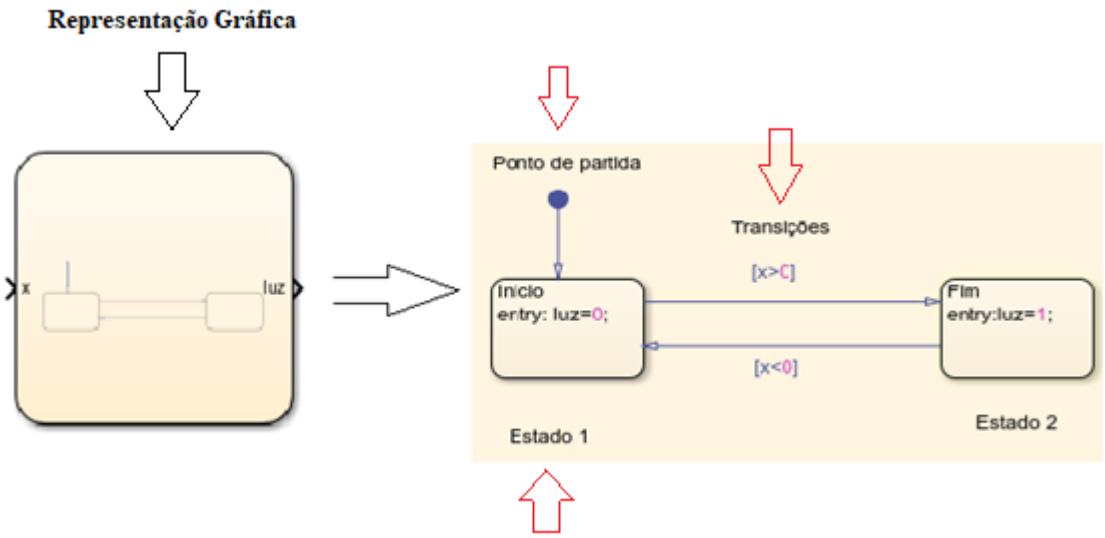
O estado é responsável por manter a execução de tal objeto, enquanto as condições que o mantém no estado forem verdadeiras. As condições são inseridas nas transições, onde estas são responsáveis pelo fluxo de decisões do diagrama. Elas ocorrem de um estado para o outro.

Dentro dos estados há a utilização de alguns comandos de execução. Os principais são:

- *Entry*: executado na entrada do estado;
- *During*: executado constantemente, enquanto estiver dentro do estado;
- *Exit*: executado quando há uma saída do estado.

A Figura 17 apresenta um diagrama de estados genérico para visualização das características tratadas anteriormente.

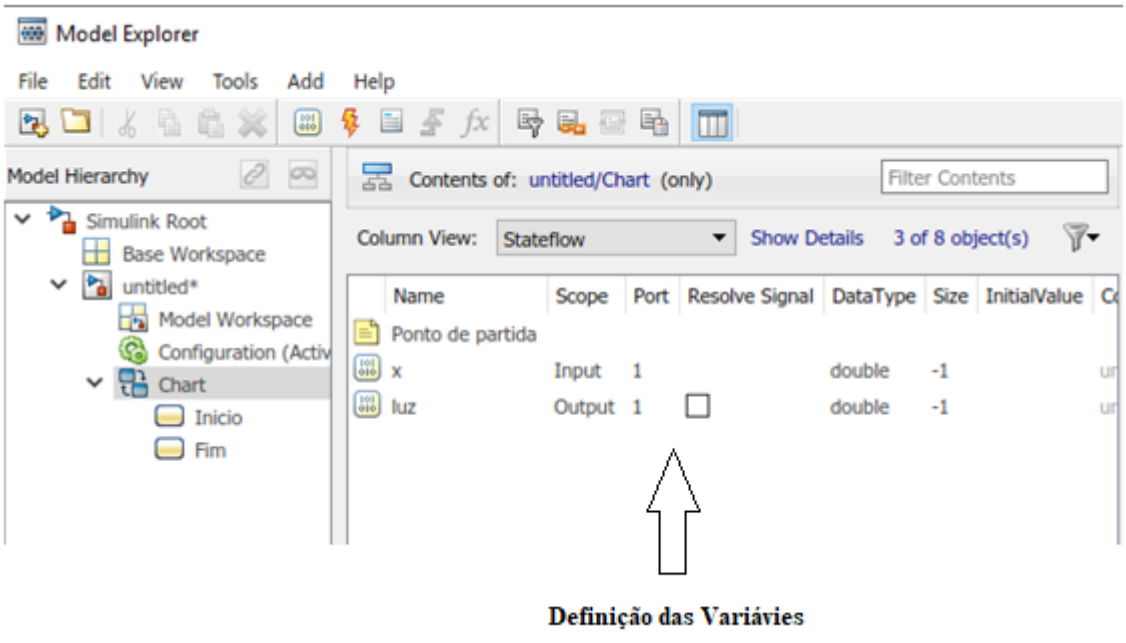
Figura 17 - Diagrama de estados genérico.



Fonte: Autor.

A definição das variáveis também é feita nesta etapa, informando seus nomes e quais são de entrada, de saída ou internamente de uma função do Simulink. O tipo de variável, que é um ponto muito importante para a construção do diagrama, também é definido nesta etapa. A Figura 18 apresenta o ambiente denominado *Model Explorer*, onde estas definições são realizadas.

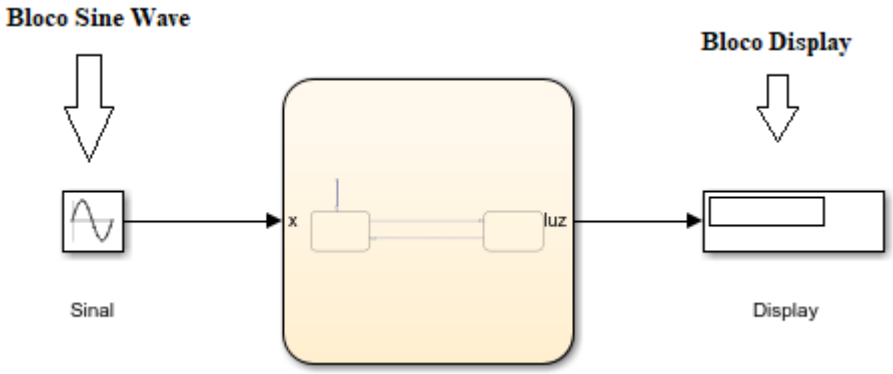
Figura 18 - Ambiente da opção Model Explorer.



Fonte: Autor.

Com a construção da lógica, se tem uma base para o restante da construção do modelo. Assim, com o acesso as demais bibliotecas presentes no *software* Simulink, blocos restantes podem ser inseridos no sistema, conforme a Figura 19, para assim finalizar o modelo. Vale ressaltar que cada bloco possui uma configuração a ser realizada dependendo da função que vai desempenhar no modelo.

Figura 19 - Modelo genérico para simulação.

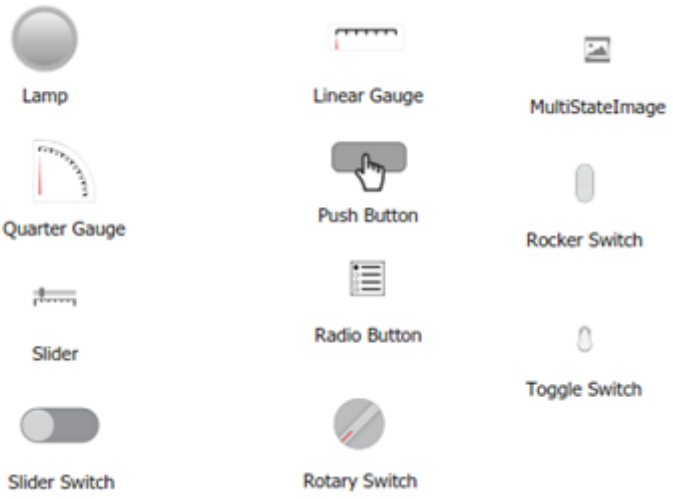


Fonte: Autor.

Até este ponto é possível executar o modelo e verificar o seu funcionamento para possíveis ajustes.

A representação física de alguns equipamentos, como chaves, luzes, tensão da bateria do veículo, é feita através de *Dashboards* presentes na biblioteca do Simulink. Alguns destes são apresentados na Figura 20.

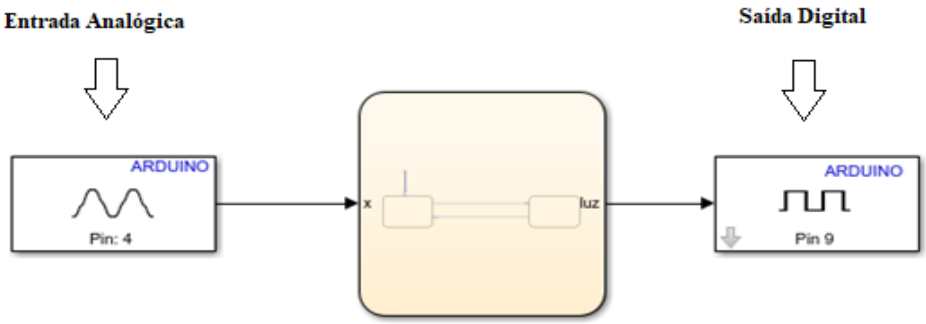
Figura 20 - Alguns Dashboards presentes na biblioteca.



Fonte: Autor

A geração de código a partir do Simulink depende inicialmente da escolha do microcontrolador, pois é necessária a instalação de pacotes de suporte para que seja possível realizar o embarque do código no dispositivo. Neste trabalho optou-se pela escolha da plataforma Arduino por haver o suporte disponível para instalação. Para a geração do código, o modelo apresentado na Figura 19 se torna o modelo da Figura 21.

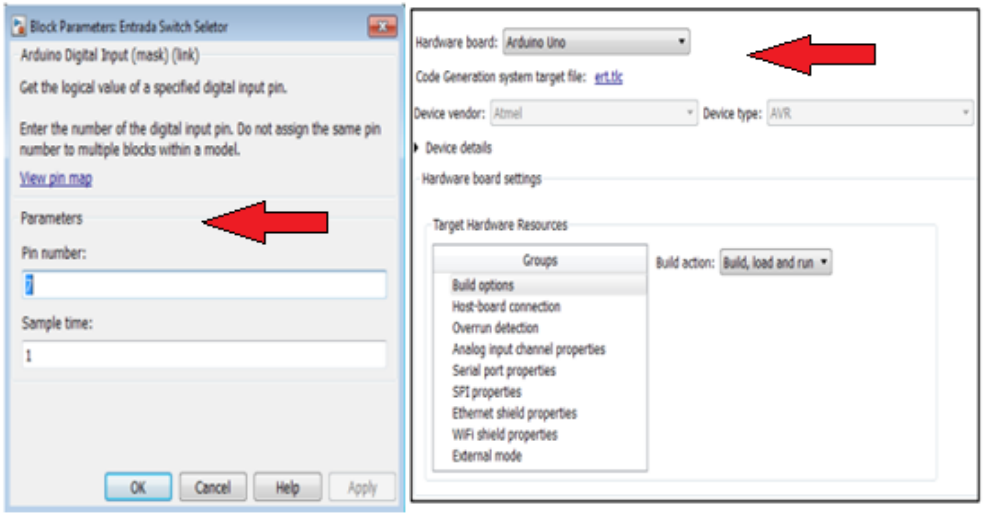
Figura 21 - Modelo genérico para simulação com blocos para geração do código.



Fonte: Autor.

A configuração dos blocos de entradas e saídas é semelhante, sendo necessário o ajuste de apenas dois parâmetros: tempo de amostragem e o pino do microcontrolador a ser utilizado, Figura 22 (a). Parâmetros de configuração, como identificação de porta de comunicação e o compilador referente à arquitetura do microprocessador são adicionados automaticamente ao Simulink pelo pacote de suporte ao Arduino, Figura 22 (b).

Figura 22 - (a) Definição dos pinos do microcontrolador. (b) Configuração para comunicação do Simulink com o microcontrolador.

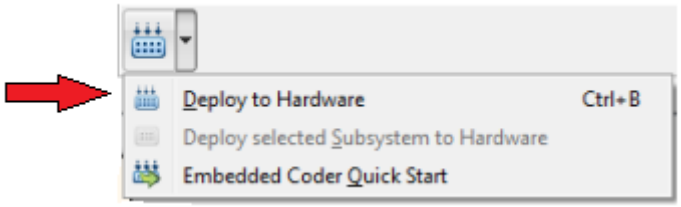


(a) (b)

Fonte: Autor.

Com o dispositivo configurado, a opção de implementação em *hardware* (*Deploy to Hardware*) realiza o *upload* do código para a plataforma Arduino, conforme apresentado na Figura 23.

Figura 23 - Opção para embarque do software.



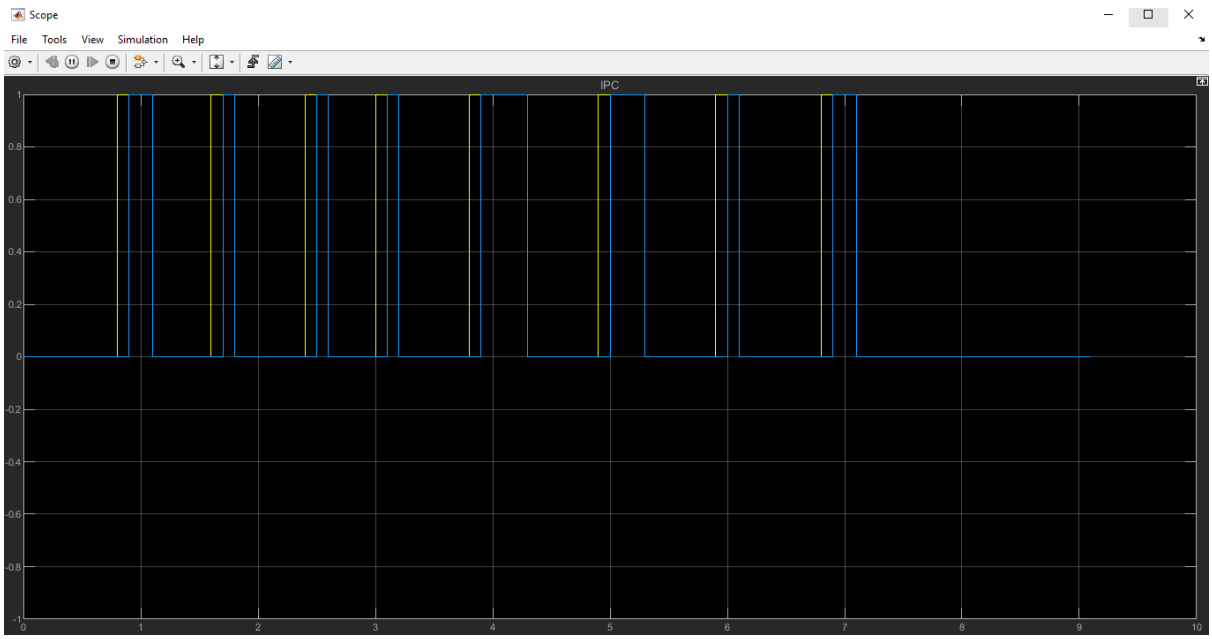
Fonte: Autor.

Assim, se nenhum erro for encontrado, o *software* é embarcado no protótipo real em escala reduzida, com os devidos componentes selecionados para atender as especificações da VF selecionada.

3.2.3 Analisador lógico

Para se ter uma base lógica do funcionamento dos modelos desenvolvidos, a partir do bloco *Scope* disponibilizado em uma das bibliotecas do Simulink, é possível analisar a relação entre os sinais de entrada e saída de forma gráfica. A Figura 24 demonstra o ambiente de visual do bloco.

Figura 24 - Ambiente visual do bloco *Scope*.

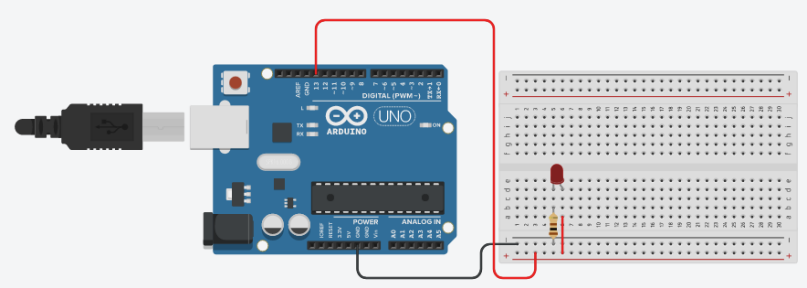


Fonte: Autor

3.2.4 Simulação do protótipo no *software* Tinkercad

Para testes antes do embarque do *software* desenvolvido na linguagem C, tanto o *software* quanto o protótipo em escala reduzida podem ser testados no Tinkercad. Ele possui diversos componentes em sua plataforma de desenvolvimento, sendo estes adequados para atender o propósito deste TCC. A Figura 25 demonstra a parte de interligação dos componentes no ambiente desta ferramenta.

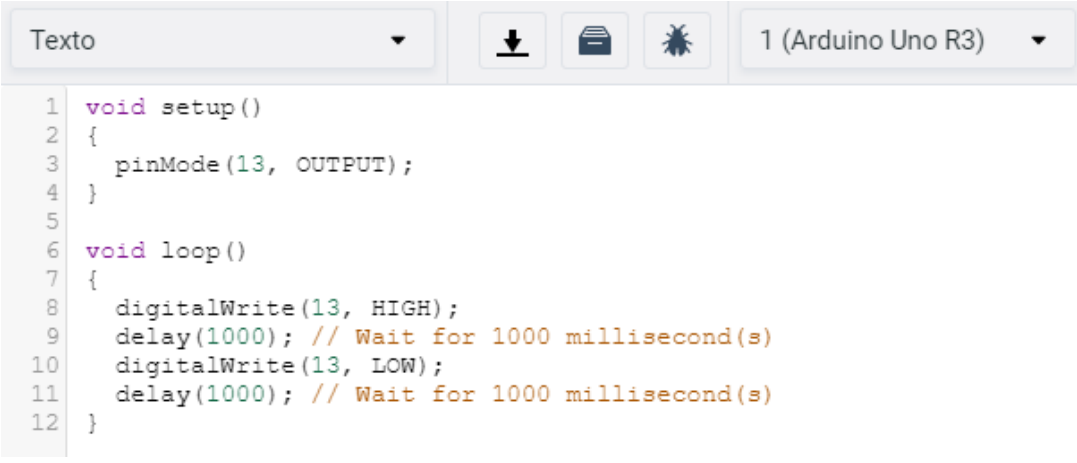
Figura 25 - Conexão dos componentes no ambiente de simulação.



Fonte: Autor.

A parte de edição do código está apresentada na Figura 26.

Figura 26 - Ambiente para edição do *software*.



Fonte: Autor.

Já a Figura 27 apresenta os componentes que podem ser utilizados.

Figura 27 - Parte que abrange os componentes que podem ser utilizados.

Fonte: Autor.

3.2.5 Montagem dos protótipos

Com os códigos verificados e corrigidos, os SEs agora podem ser desenvolvidos em escala reduzida, onde os circuitos são construídos com os componentes analisados nos requisitos de cada VF e seguindo os diagramas dos modelos. A Tabela 3 apresenta os componentes necessários para atender a montagem das duas VFs selecionadas.

Tabela 3 - Componentes utilizados e quantidade

Descrição	**Quantidade**
Potenciômetro linear	4
Push buttons	4
Resistor	2
Buzzer	1
Sensor Ultrassônico	3
LED	1
Conectores	20
Protoboard	1
Arduino Uno	1

Fonte: Autor.

3.2.6 Estilo de documentação proposta

Constatado o correto funcionamento dos SEs, a parte de documentação é realizada, para que os sistemas aqui desenvolvidos possam ser recriados por outros interessados na área, tendo como referência apenas a documentação construída. Os tópicos a seguir tratam de como é formulado este documento, da seguinte forma:

- Nome: O nome da VF será relacionado com o principal equipamento a ser controlado;
- Descrição: A descrição está relacionada ao funcionamento da VF;
- Diagrama do modelo: Apresentação do diagrama *stateflow* construído;
- Microcontrolador: Definição do microcontrolador;
- Componentes e suas funções: Descrição dos principais componentes utilizados e suas funções no sistema.

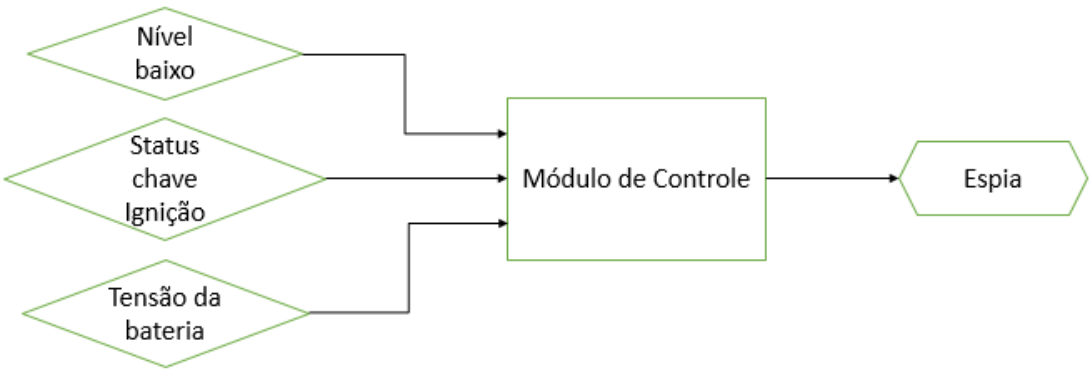
4 RESULTADOS E DISCUSSÕES

Para as VFs escolhidas como objeto deste trabalho, nas subseções a seguir serão detalhados os resultados obtidos, desde a criação do diagrama de controle de cada uma até a montagem do protótipo, contemplando assim a documentação por trás da VF.

4.1 VF Nível do Óleo de Freio (NOF)

O diagrama de Figura 28 apresenta a composição da função NOF, com suas entradas e saídas respectivamente.

Figura 28 - Diagrama VF NOF.



Fonte: Autor.

De acordo com a Figura 28, esta VF possui como entradas:

- Nível baixo: No sistema quem monitora o nível do óleo de freio é um sensor de nível, que no caso pode ser representado por uma chave. Sendo assim, o estado normalmente aberto indica que o nível está ideal e o estado normalmente fechado indica que o nível está abaixo do nível adequado.
- Status chave de ignição: A chave de ignição representa a pré energização do sistema, ou seja, o sistema está pronto para dar partida no motor e demais funções. Também sendo representada por uma chave, onde o estado normalmente aberto indica que a ignição não está acionada e o estado normalmente fechado indica que a ignição está acionada.
- Tensão da bateria: O sistema necessita de uma tensão mínima para entrar em funcionamento. Com isso é necessário que sempre haja uma verificação da tensão da bateria do veículo.

Possui como saída:

- Espia: O sinal de alerta é mostrado no painel de instrumentos do veículo. É acionada uma espia, que no caso será representada por um *led*.

A partir desta análise, pode-se definir as entradas e saídas digitais da VF, da seguinte forma:

- Entradas digitais:
  - Nível baixo, onde 0 indica que o nível está dentro do normal e 1 que o nível está em estado crítico.
  - Status da chave de ignição, onde 0 indica chave desligada e 1 indica chave ligada.
- Entradas analógicas:
  - Tensão da bateria, que é um valor variável e tem como restrição para funcionamento que seu valor seja maior ou igual a 10,8 volts.
- Saída digital:
  - Espia, onde 0 o Led se mantém desligado e 1 o Led é acionado.

Dessa forma, toda a lógica de funcionamento do sistema respeita o pseudocódigo da Figura 29, que nada mais é uma forma simplificada do algoritmo.

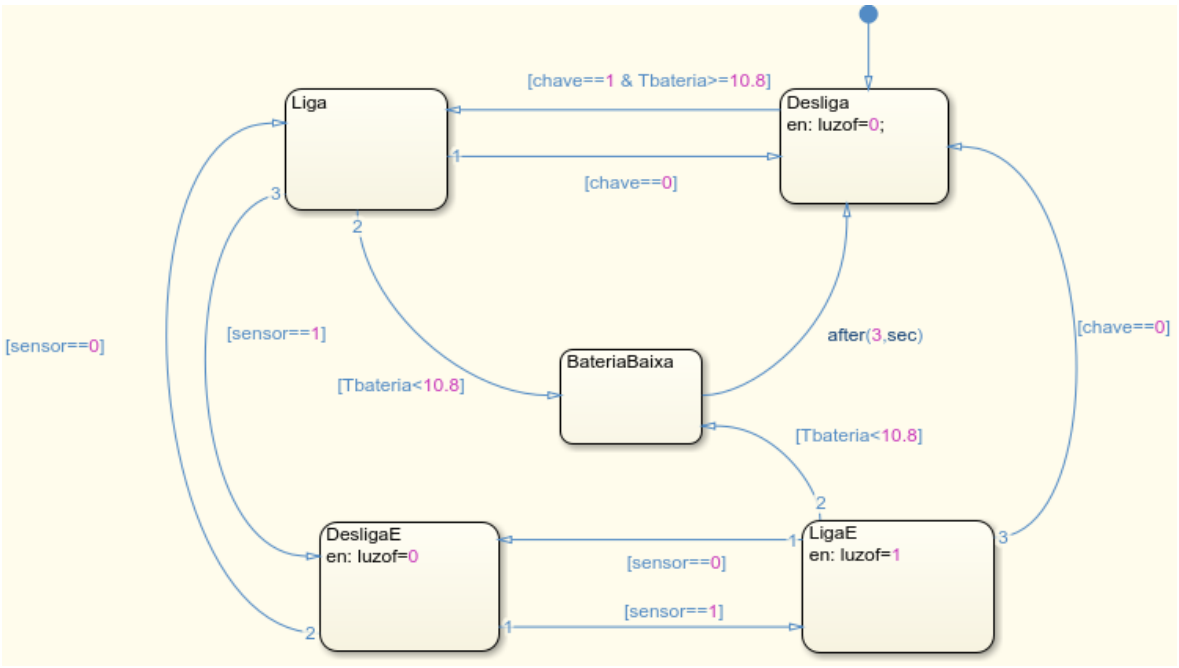
Figura 29 - Pseudocódigo para a VF NOF.

```
Início
    Parâmetros:
        chave de ignição
        tensão da bateria
        nível do óleo de freio
        espia no painel
    se a chave de ignição estiver ativa e a tensão bateria >=10.8 faça:
        se o nível do óleo de freio estiver baixo faça:
            espia é ativada
        fim se
        se o nível do óleo estiver ideal faça:
            espia é desativada
        fim se
    fim se
Fim
```

Fonte: Autor

A partir dessas informações, o diagrama de estados apresentado na Figura 30 pode ser construído.

Figura 30 - Diagrama *Stateflow* da VF NOF.



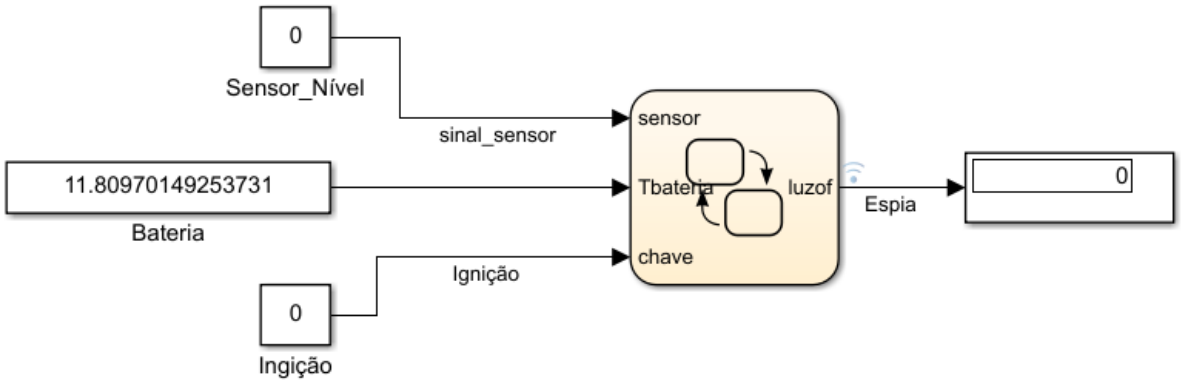
Fonte: Autor.

As seguintes variáveis adotadas e suas características, conforme a Figura 30, são:

- chave: Entrada digital que recebe o *status* da chave de ignição;
- sensor: Entrada digital que recebe sinal quando o nível baixo de óleo é detectado;
- Tbateria: Entrada analógica que recebe o sinal da bateria;
- luzof: Saída digital que envia o sinal para acionamento da espia

Na Figura 31 é apresentado o modelo adotado para simulação e verificação da lógica final apresentada anteriormente, após a identificação de possíveis falhas e erros.

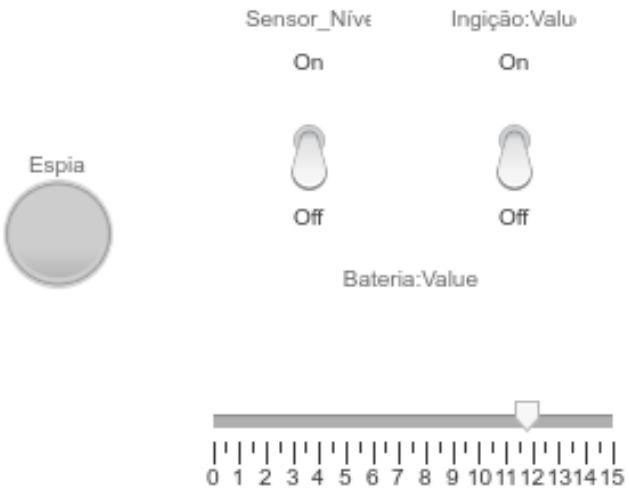
Figura 31 - Modelo adotado para simulação, VF NOF.



Fonte: Autor

A representação dos equipamentos que fornecem os dados de entrada e recebem os dados de saída, na execução do modelo anterior, é apresentada na Figura 32.

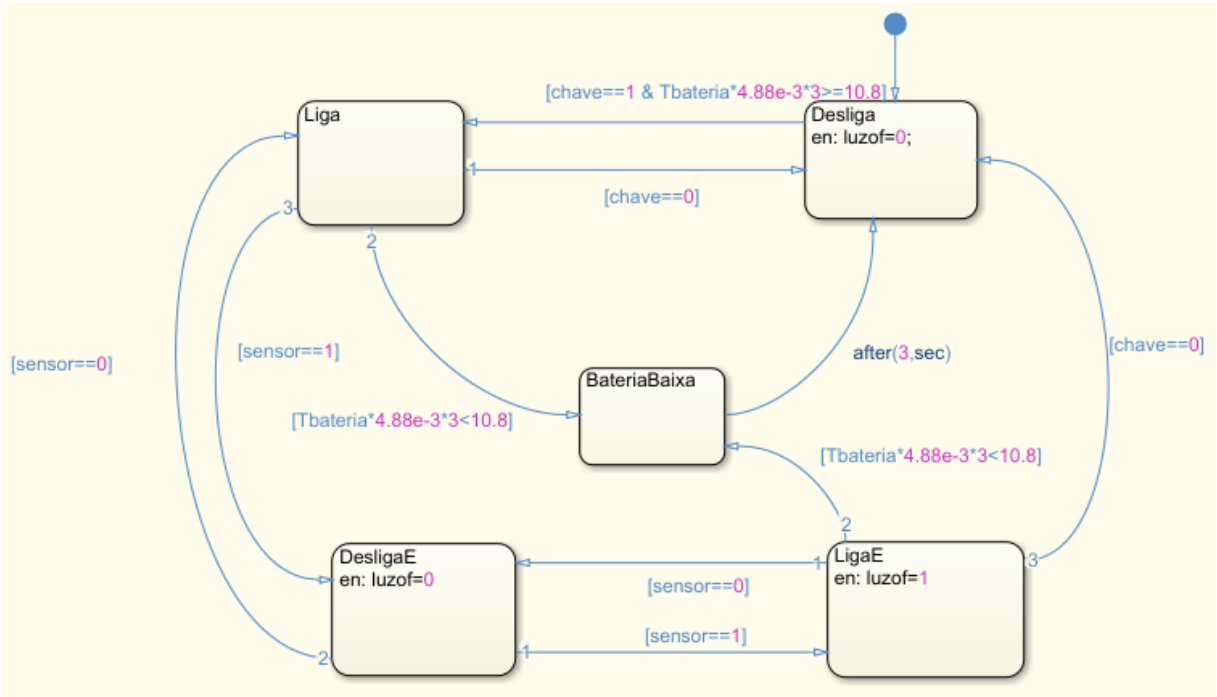
Figura 32 - Representação dos equipamentos.



Fonte: Autor.

Para embarque do *software* na plataforma Arduino Uno o modelo apresentado na Figura 30 necessitou sofrer alterações nas transições que dependiam do valor monitorado da bateria, se tornando o modelo da Figura 33.

Figura 33 - Modelo para embarque no Arduino.



Fonte: Autor.

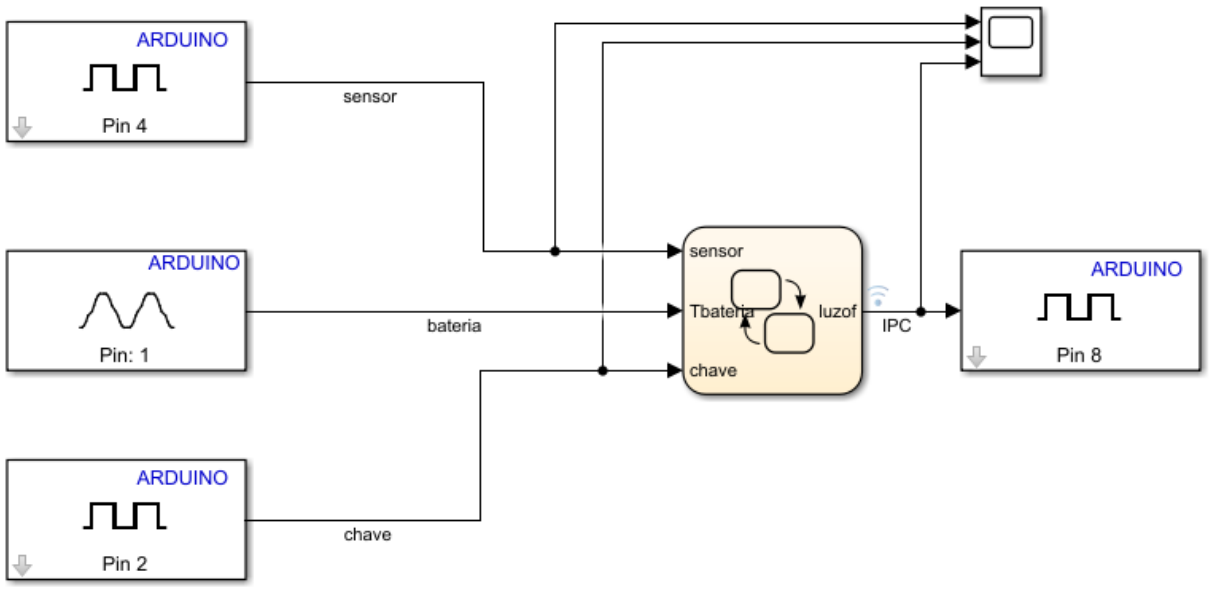
Essa alteração foi necessária devido os dados obtidos pela porta analógica serem em bits, necessitando utilizar o fator de conversão que no caso é $4{,}88mV$, obtido pela Equação 1.

$$Resolução = \frac{5\ Volts}{2^8} = 4{,}88\ mV \quad (1)$$

A multiplicação por três é necessária para se obter valores mais próximos aos reais como requisito da VF para a tensão da bateria, que é 10,8 volts, uma vez que a fonte utilizada no protótipo é a própria da plataforma, no valor de 5 volts.

Após os testes e validações anteriores, o modelo para embarque do s*oftware* no microcontrolador Arduino é apresentado na Figura 34.

Figura 34 - Modelo para embarque do código no Arduino.

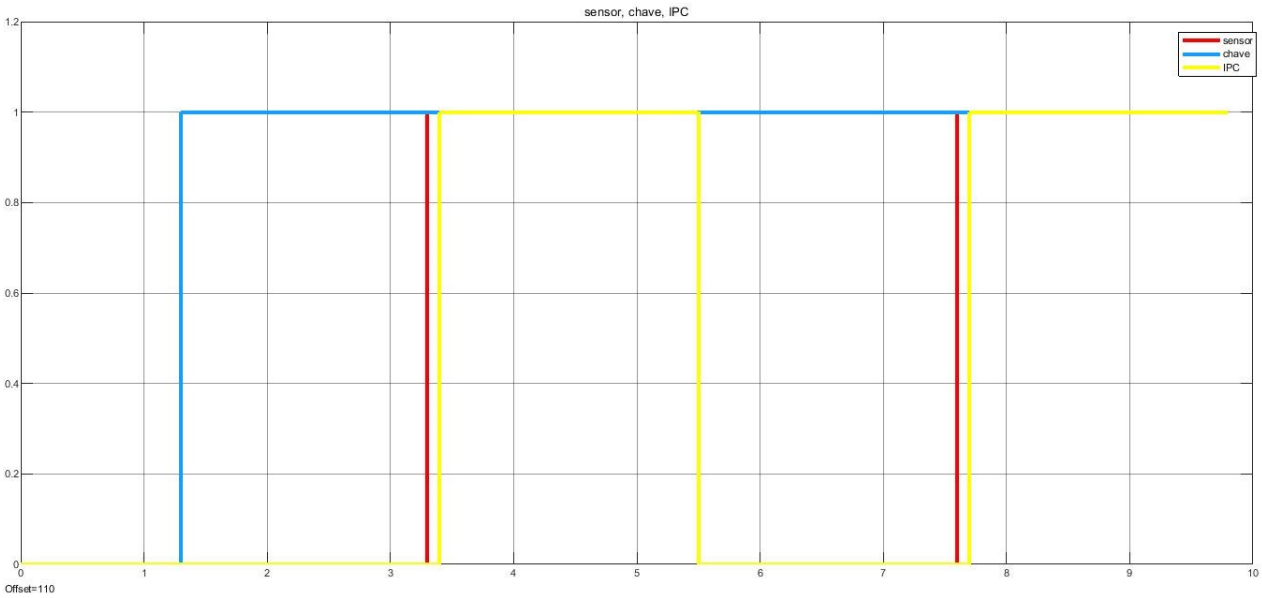


Fonte: Autor.

Como pode ser observado, foram utilizados os pinos 2 e 4 para obtenção dos dados digitais de entrada e para obtenção do dado analógico o pino 1, que no Arduino é representado como A01. Para a saída digital foi utilizado o pino 8. O código gerado é apresentado no Anexo B deste TCC.

Com o auxílio do analisador lógico, pode-se verificar os sinais e quais estavam entrelaçados no funcionamento do sistema. Pela Figura 35, tem-se que o sinal digital da chave é o sinal em azul, o sinal obtido pelo sensor é o sinal em vermelho e o sinal para acionamento do *led* é o sinal em amarelo. É possível observar que quando o sinal chave e sensor estão em 1, tem-se que o valor do sinal IPC está em 1 também. Se um dos dois sinais anteriores estiverem em zero, o sinal IPC se mantém em zero, confirmando assim o correto funcionamento do sistema.

Figura 35 - Sinais digitais do sistema em funcionamento.

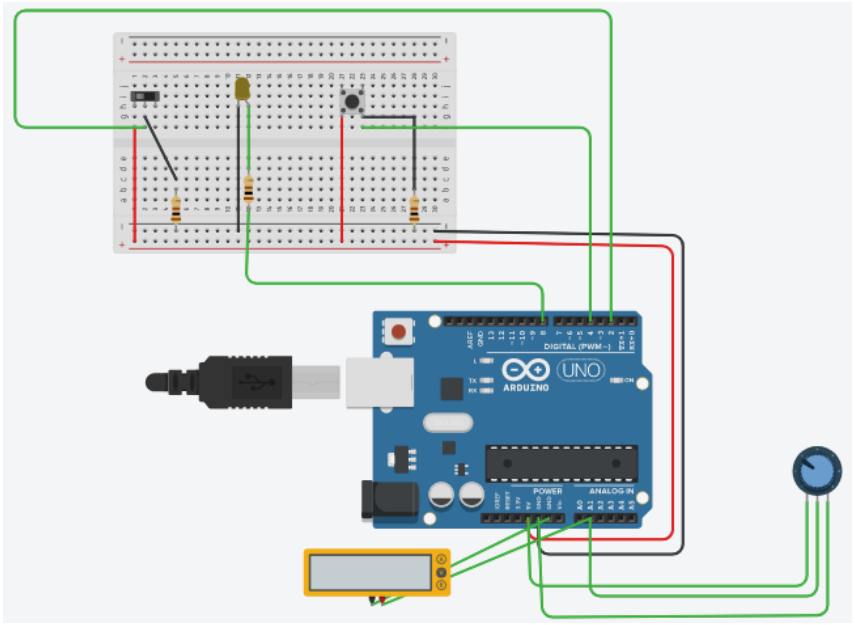


Fonte: Autor.

4.1.1 Simulação no Tinkercad e protótipo

A Figura 36 apresenta o modelo desenvolvido no Tinkercad para a simulação a partir do *software* desenvolvido no IDE do próprio simulador que obedece às mesmas características do IDE principal do Arduino.

Figura 36 - Modelo desenvolvido no Tinkercad, VF NOF.

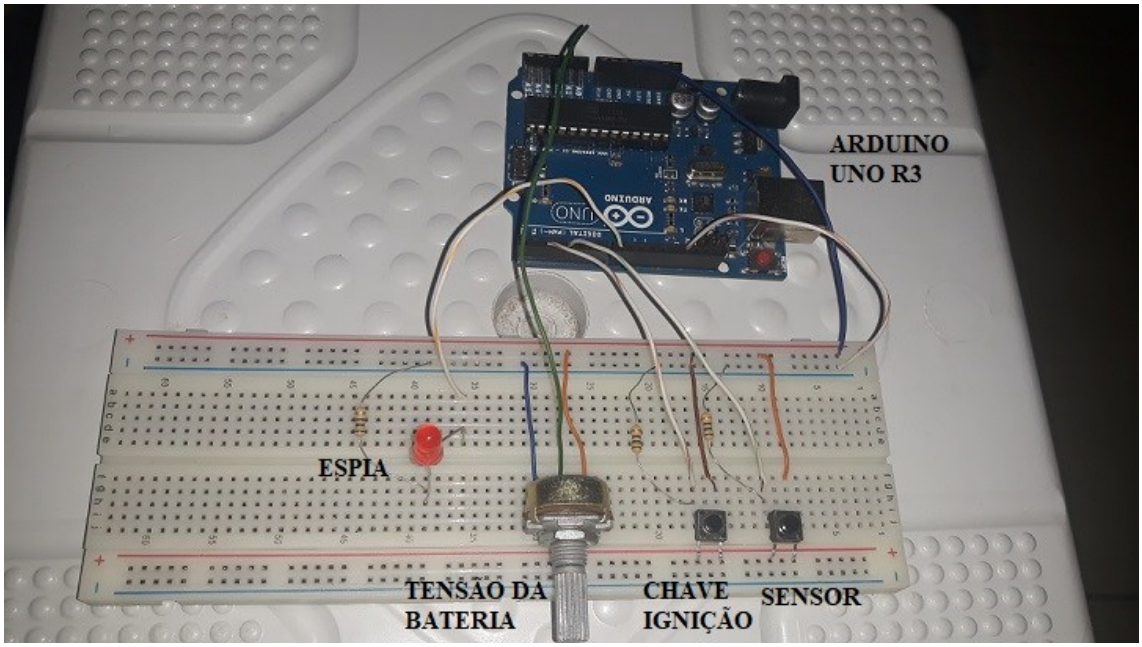


Fonte: Autor.

O *software* se encontra no Anexo A e respeita o pseudocódigo da Figura 29. É um código bem mais enxuto do que o código gerado pelo Simulink, mas apresenta maiores dificuldades no desenvolvimento e testes do funcionamento do sistema, devido necessitar trabalhar diretamente com a construção do circuito. Possui a mesma condição de conversão da leitura analógica apresentada na subseção anterior.

A montagem do protótipo foi realizada de forma a atender o diagrama que é apresentado na Figura 34, e o circuito da Figura 36, podendo verificar na prática o correto funcionamento dos *softwares* construídos. Esse protótipo é apresentado na Figura 37.

Figura 37 - Protótipo para a VF NOF.



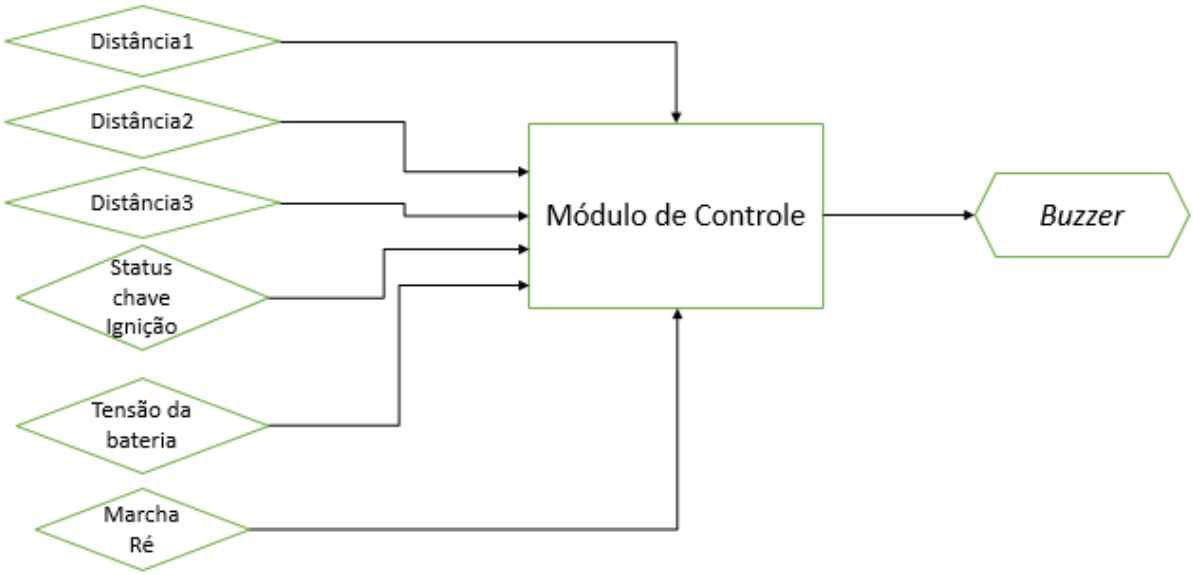
Fonte: Autor.

Pela Figura 37 tem-se que o *led* representa a espia do painel de instrumentos e serve de alerta para o condutor. Tanto a chave de ignição quanto o sensor de nível foram representados por *push buttons*, que ao serem pressionados indicam a atuação dos componentes. A tensão da bateria é regulada através do potenciômetro, para verificar o funcionamento quando a tensão está ou não no nível ideal.

4.2 VF Alerta de Estacionamento (AE)

O diagrama de Figura 38 apresenta a composição da função AE, com suas entradas e saídas respectivamente.

Figura 38 - Diagrama para a VF AE.



Fonte: Autor.

De acordo com a Figura 38, esta VF possui como entradas:

- Distância 1: Distância esta monitorada por um sensor ultrassônicos localizado na parte central na traseira do veículo.
- Distância 2: Distância esta monitorada por um sensor ultrassônicos localizado mais à esquerda da parte traseira do veículo.
- Distância 3: Distância esta monitorada por um sensor ultrassônicos localizado mais à direita da parte traseira do veículo.
- *Status* chave de ignição: A chave de ignição representa a pré-energização do sistema, ou seja, o sistema está pronto para dar partida no motor e demais funções. Também sendo representada por uma chave, onde o estado normalmente aberto indica que a ignição não está acionada e o estado normalmente fechado indica que a ignição está acionada.
- Tensão da bateria: O sistema necessita de uma tensão mínima para entrar em funcionamento. Com isso é necessário que sempre haja uma verificação da tensão da bateria do veículo.
- Marcha Ré: O sistema só entra em funcionamento quando a marcha ré é acionada. Esta pode ser representada por uma chave, onde o estado normalmente aberto indica que a marcha não foi acionada e o estado normalmente fechado indica que a marcha foi engatada.

Possui como saída:

- *Buzzer*: O sinal de alerta é um sinal sonoro e é emitido através de um *buzzer* que, ao passo que a distância vai diminuindo, o sinal se torna mais intenso para uma melhor atenção do condutor.

A partir desta análise, pode-se definir as entradas e saídas da VF, da seguinte forma:

- Entradas digitais:
  - Status da marcha ré, onde 0 indica que a marcha não está acionada e 1 que a marcha está acionada.
  - Status da chave de ignição, onde 0 indica chave desligada e 1 indica chave ligada.
- Entradas analógicas:
  - Tensão da bateria, que é um valor variável e tem como restrição para funcionamento que seu valor seja maior ou igual a 10,8 volts
  - Distância 1, que é um valor variável e recebe informações da distância da parte central em relação à algum objeto.
  - Distância 2, que é um valor variável e recebe informações da distância mais à esquerda em relação à algum objeto.
  - Distância 3, que é um valor variável e recebe informações da distância mais à direita em relação à algum objeto.
- Saída digital:
  - *Buzzer*, onde de acordo com a distância detectada, emite um sinal sonoro com certa frequência de acordo com o sinal PWM recebido.

Dessa forma, toda a lógica de funcionamento do sistema respeita o pseudocódigo da Figura 39.

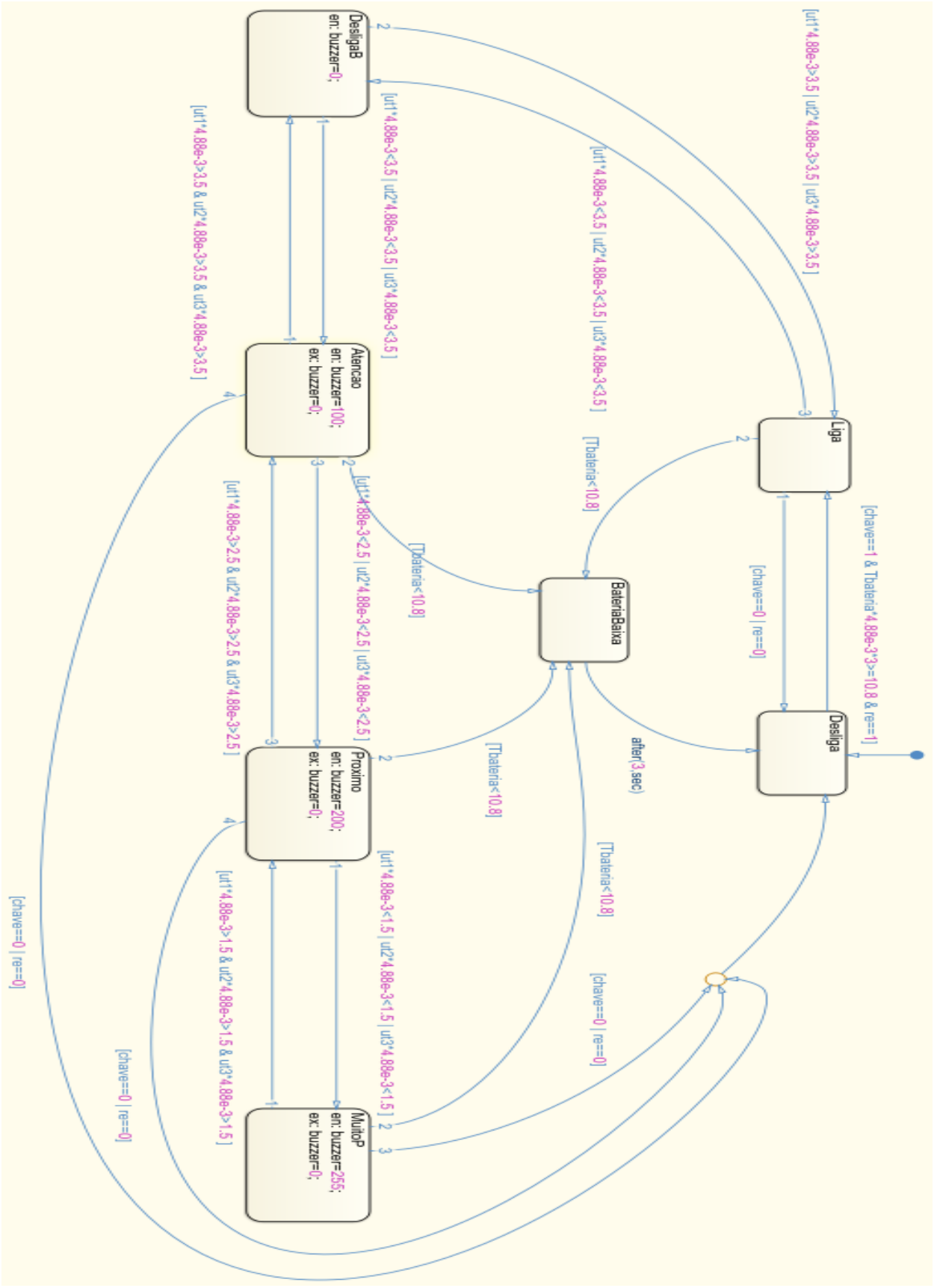
Figura 39 - Pseudocódigo para a VF AE.

```
Início
    Parâmetros:
        chave de ignição
        tensão da bateria
        marcha à ré
        distâncias:
            d1
            d2
            d3
        frequências:
            f1
            f2
            f3
        buzzer
    se a chave de ignição estiver ativa e a tensão bateria >=10.8 faça:
        se a marcha à ré for acionada faça:
            se  d1 <  30 cm ou d2 < 30 cm ou d3 < 30 cm faça:
                buzzer= f1
            fim se
            se  d1 <  20 cm ou d2 < 20 cm ou d3 < 20 cm faça:
                buzzer= f2
            fim se
            se  d1 <  10 cm ou d2 < 10 cm ou d3 < 10 cm faça:
                buzzer= f3
            fim se
        fim se
    fim se
Fim
```

Fonte: Autor.

A partir dessas informações, o diagrama de estados apresentado na Figura 40 pode ser construído.

Figura 40 - Diagrama de estados VF AE.



Fonte: Autor.

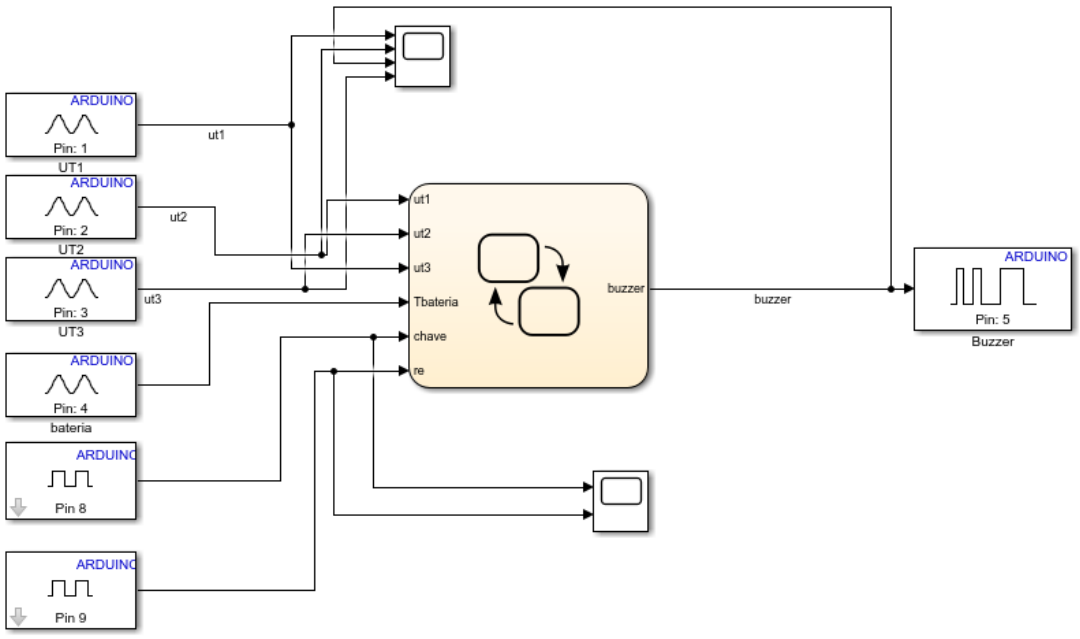
As seguintes variáveis adotadas e suas características, conforme a Figura 40, são:

- chave: Entrada digital que recebe o *status* da chave de ignição.
- re: Entrada digital que recebe sinal quando a marcha à ré é acionada.
- ut1: Entrada analógica que recebe a distância do sensor ultrassônico 1.
- ut2: Entrada analógica que recebe a distância do sensor ultrassônico 2.
- ut3: Entrada analógica que recebe a distância do sensor ultrassônico 3.
- Tbateria: Entrada analógica que recebe o sinal da bateria.
- buzzer: Saída digital que envia o sinal para acionamento do *buzzer*.

Como a plataforma Arduino possui bibliotecas próprias para trabalhar com sensores ultrassônicos, não foi possível realizar a comunicação efetiva do Simulink com os sensores ultrassônicos por não ser possível a integração dessas bibliotecas. Os mesmos foram representados através de sinais de tensão, necessitando assim da conversão conforme a Equação 1. Dessa forma, como foi utilizada a própria fonte do Arduino que possui um valor máximo de 5 V, os valores presentes no diagrama da Figura 40 para comparação como, 1.5, 2.5 e 3.5 correspondem à 10 cm, 20 cm e 30 cm correspondentemente.

Diferentemente da VF NOF o modelo para simulação já foi desenvolvido com a comunicação do Simulink e o Arduino já estabelecida. Isso ocorreu devido primeiramente o sucesso da VF NOF na comunicação, mas como ponto principal, a dificuldade de representar o correto acionamento do *buzzer*. O modelo proposto é apresentado na Figura 41.

Figura 41 - Modelo adotado para simulação, VF AE.



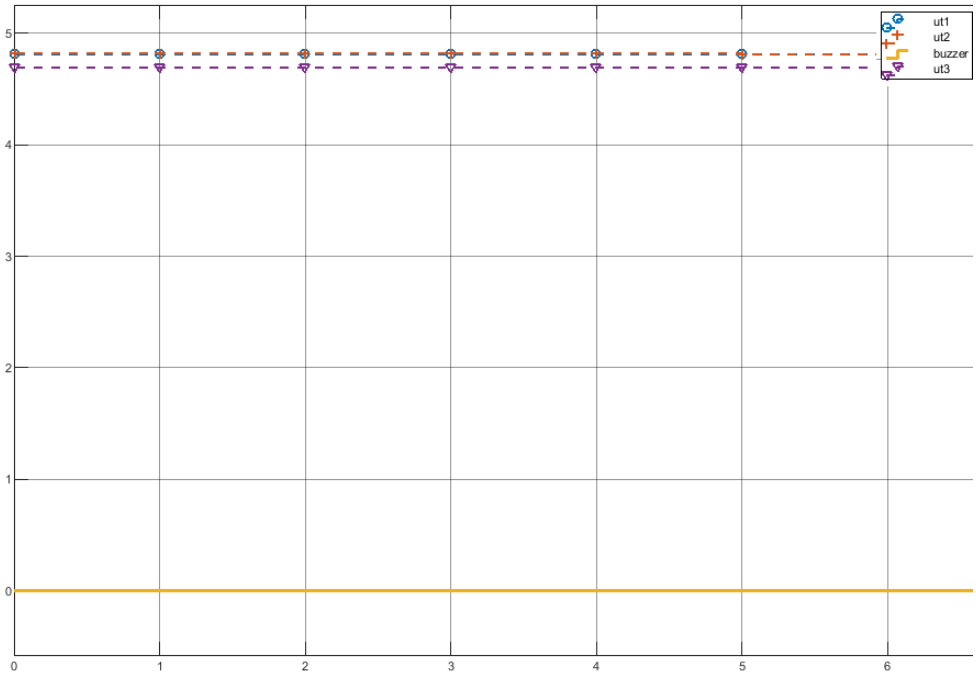
Fonte: Autor.

Pela Figura 41 pode ser observado que foram utilizados os pinos 1, 2, 3 e 4 para obtenção dos sinais analógicos, sendo estes dos três sensores e da tensão da bateria. Como entradas digitais optou-se pelos pinos 8 e 9, para a chave de ignição e *status* da marcha ré. Como saída digital, utilizou-se do bloco PWM no pino 5, para acionamento do *buzzer*.

Pelo bloco *Scope* pode-se ter uma análise do comportamento dos sinais dos sensores com o funcionamento do sistema, principalmente com o acionamento do *buzzer*. A seguir é apresentada uma sequência de acionamento de apenas um sensor, pois, para os outros sensores o comportamento do sistema é o mesmo.

A Figura 42 apresenta o os sinais quando o sistema está ativado, mas apenas na fase de monitoramento das distâncias. O sistema em estado de monitoração significa que ele está em pleno funcionamento após atender todos os requisitos definidos na lógica, mas nenhuma das distâncias pré-definidas foram detectadas pelos sensores. Os sinais azul, roxo e vermelho são os sinais captados pelos sensores e o sinal em amarelo o sinal enviado para o *buzzer*. O gráfico foi montado na escala de Tensão x Tempo, sendo a tensão medida em volts e o tempo em segundos.

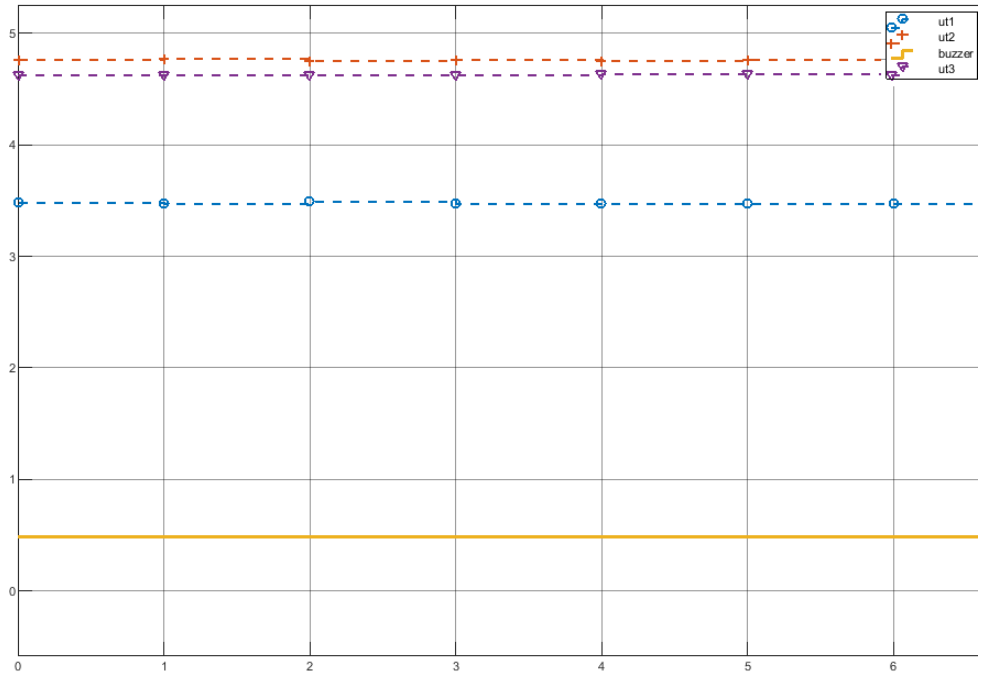
Figura 42 - Sistema em estado de monitoração.



Fonte: Autor.

A Figura 43 apresenta os mesmos sinais, mas agora apenas o sensor ultrassônico 1 identificou um objeto a uma distância menor que 30 cm.

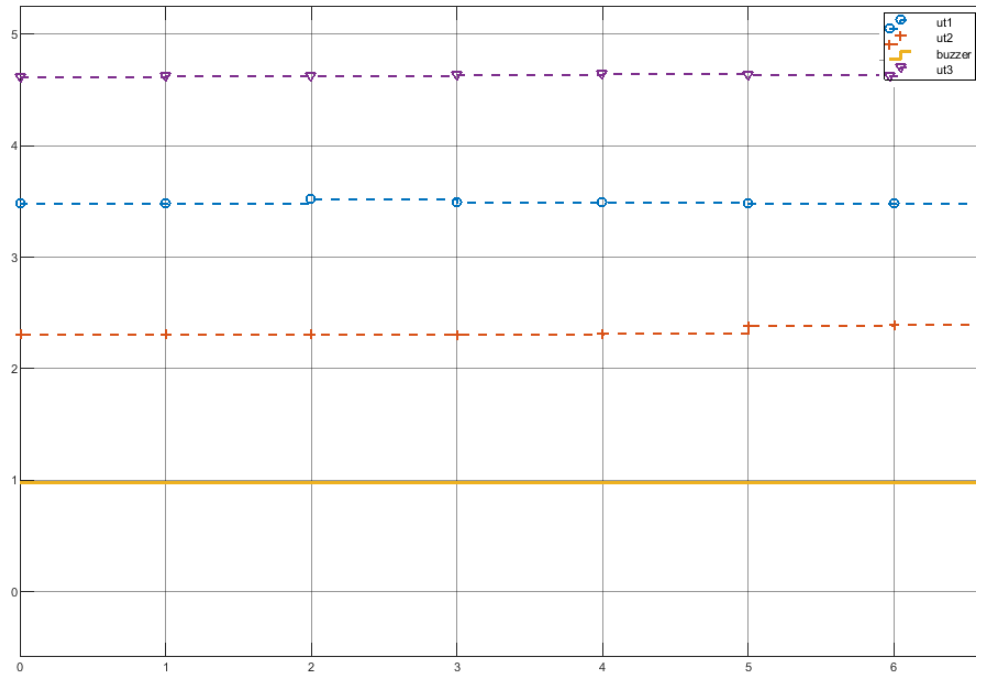
Figura 43 - Detecção de um objeto pelo sensor ultrassônico 1 a uma distância menor do que 30 cm.



Fonte: Autor.

Já Figura 44 apresenta um objeto identificado a uma distância menor que 20 cm pelo sensor ultrassônico 2 e o sensor ultrassônico 1 ainda identifica um objeto à uma distância menor que 30 cm mas maior que 20 cm.

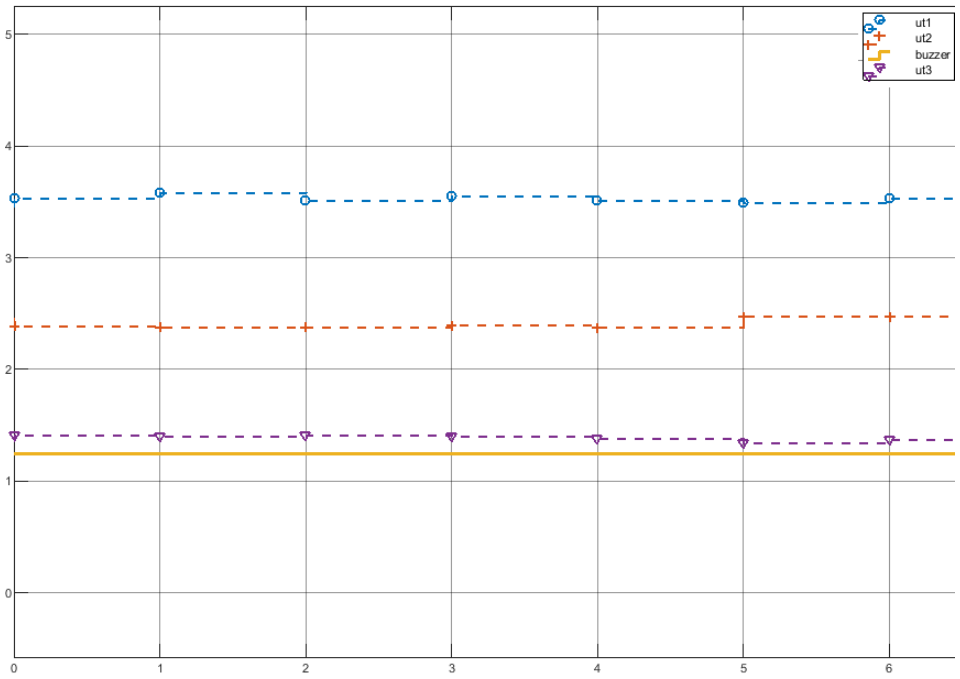
Figura 44 - Detecção de um objeto pelo sensor ultrassônico 2 a uma distância menor do que 20 cm.



Fonte: Autor.

A Figura 45 apresenta um objeto a uma distância menor que 10 cm identificado pelo sensor ultrassônico 3 e os mesmos objetos identificados pelos sensores ultrassônicos 1 e 2.

Figura 45 - Detecção de um objeto pelo sensor ultrassônico 3 a uma distância menor do que 30 cm.



Fonte: Autor.

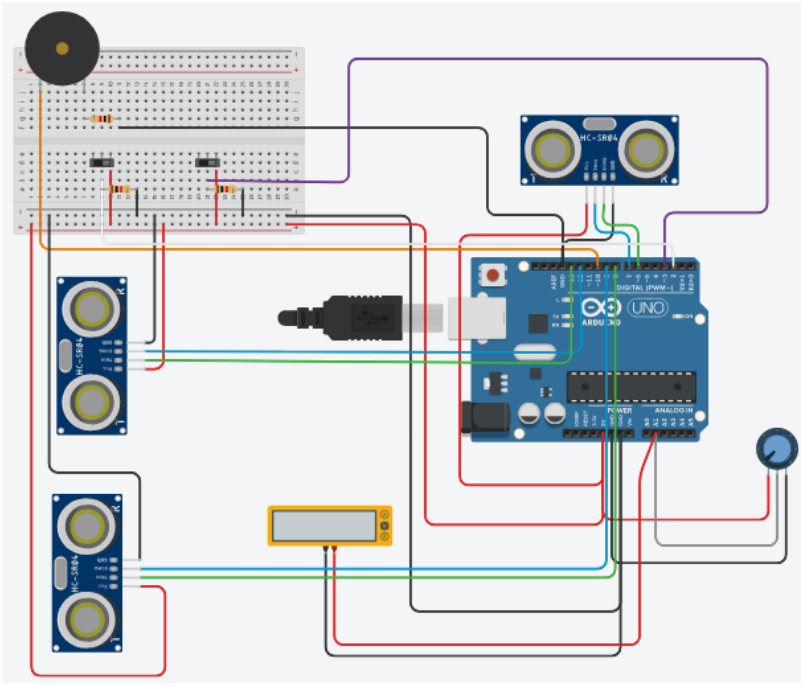
Ao analisar os sinais desde a Figura 42 até a Figura 45, pode-se observar que quanto menor a distância, maior o valor do sinal PWM enviado a porta para acionamento do *buzzer*, aumentando assim sua frequência de acionamento. Pode-se avaliar também o correto funcionamento do sistema dando prioridade para um alerta mais chamativo para o objeto que se encontra a menor distância.

Inicialmente apenas os dados precisaram ser captados do Arduino, sendo que as ações a serem tomadas eram de responsabilidade do Simulink. Posteriormente o embarque do código foi realizado, este presente no Anexo B.

4.2.1 Simulação no Tinkercad

A Figura 46 apresenta o circuito proposto para simulação. O código desenvolvido se encontra no Anexo A, respeitando o pseudocódigo da Figura 39.

Figura 46 - Modelo desenvolvido no Tinkercad, VF AE.

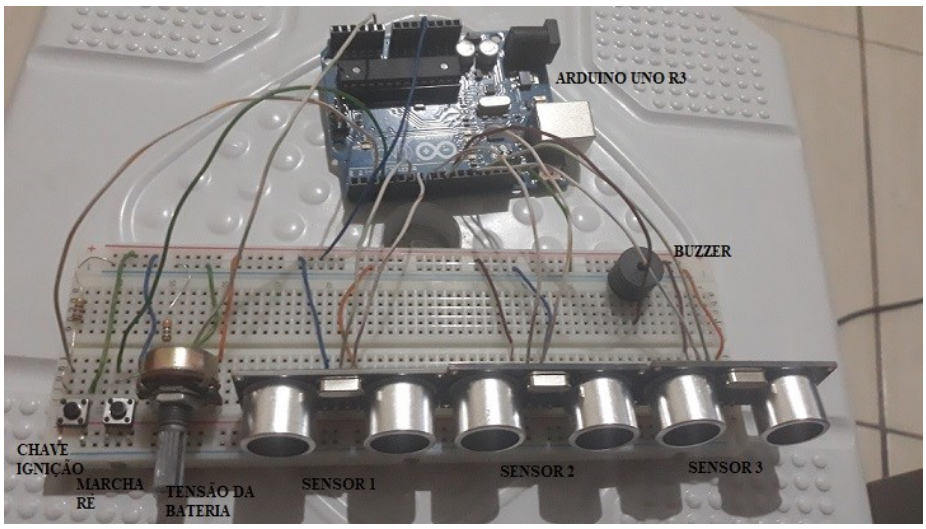


Fonte: Autor.

Pela Figura 46 pode-se observar que o modelo possui duas chaves como representação da chave de ignição e marcha ré. Possui também o *buzzer* para alerta sonoro, três sensores ultrassônicos para identificação das distâncias e um potenciômetro para controle da carga da bateria.

Este sistema juntamente com o *software* desenvolvido atendeu as condições de funcionamento esperadas, podendo ser construído em escala real, como é apresentado pela Figura 47.

Figura 47 - Protótipo VF AE.



Fonte: Autor.

5 CONCLUSÕES

O presente trabalho teve como objetivo o desenvolvimento de funções veiculares, assim como sua documentação, nos moldes de desenvolvimento das grandes empresas do setor, com uma visão mais acadêmica. Assim qualquer discente que esteja interessado pela área é capaz de reproduzir os resultados aqui encontrados.

Toda a codificação obtida a partir do ambiente gráfico do Simulink foi implementada com sucesso no microcontrolador Arduino Uno. Embora esse microcontrolador não possua a robustez necessária para a implantação real em veículo, ele possibilita a verificação prática dos códigos e possíveis ajustes durante o desenvolvimento, sendo uma importante ferramenta para o desenvolvimento dos conceitos dos sistemas embarcados.

A adoção do MBD para criação e reutilização de sistemas construídos a partir da integração Simulink e Arduino se mostra eficiente, pois elimina repetições excessivas nas fases iniciais do desenvolvimento e permite maior foco nas fases de testes e validação, assim possibilita maior confiabilidade aos *softwares* construídos, o que resulta em um sistema mais seguro.

Neste trabalho obteve-se resultados para VFs de forma isolada, no caso, nível de óleo do freio e alerta de estacionamento. Contudo, pode ser desenvolvido como trabalho futuro, a unificação de VFs de tal forma que o sistema possa identificar qual está atuando em determinado tempo, através de um único microcontrolador, ou seja, realizar a junção das VFs em apenas um sistema. De forma resumida se teria uma única central eletrônica responsável por tais funções.

Ainda tratando-se de trabalhos futuros, poderia desenvolver as mesmas VFs com microcontroladores diferentes para avaliar o impacto na comunicação, desenvolver a comunicação como uma rede automotiva entre as centrais eletrônicas.

Por fim, este trabalho possibilitou a construção de um artigo, com submissão na Revista de Engenharia e Pesquisa Aplicada (REPA) no dia 02/03/2020 e o mesmo foi aceito para publicação em uma edição da revista no dia 20/07/2020.

REFERÊNCIAS BIBLIOGRÁFICAS

AUTOESPORTE. **MEU CARRO SAIU DE LINHA: BOA HORA PARA COMPRAR OU VENDER?** 2019. Disponível em: <https://revistaautoesporte.globo.com/Noticias/noticia/2019/03/meu-carro-saiu-de-linha-boa-hora-para-comprar-ou-vender.html>. Acesso em: 04 nov. 2019.

ANUÁRIO DA INDÚSTRIA AUTOMOBILÍSTICA BRASILEIRA: Brazilian Automotive Industry Yearbook. São Paulo: ANFAEVA, jan. 2019.

ANJOS, Eduardo Giovannetti Pereira dos. **A Evolução da Eletrônica Embarcada na Indústria Automobilística Brasileira**. 2011. 125 f. Monografia (Especialização) - Curso de Engenharia Automotiva, Centro Universitário do Instituto Mauá de Tecnologia, São Caetano do Sul, 2011.

AUTODESK. **Da mente ao projeto em minutos**. 2014. Disponível em: <https://www.tinkercad.com/>. Acesso em: 04 nov. 2019.

BENTES, Leandro Maurício Araújo. **SISTEMA DE SEGURANÇA VEICULAR COM USO DE GPS BASEADO EM ARDUINO**. 2013. 114 f. TCC (Graduação) - Curso de Engenharia da Computação, Universidade do Estado do Amazonas, Manaus, 2013.

BANIK, Taysa Millena. **PROCESSO DE DESENVOLVIMENTO BASEADO EM MODELO PARA SOFTWARE AUTOMOTIVO: MIGRAÇÃO PARA O PADRÃO AUTOSAR**. 2017. 68 f. TCC (Graduação) - Curso de Engenharia Eletrônica, Universidade TecnolÓgica Federal do ParanÁ, Ponta Grossa, 2017.

CARRO, L.; WAGNER, F. R. Sistemas Computacionais Embarcados. **XXII Jornadas de Atualização em Informática - JAI**, p. Capítulo 2, 2003

CARVALHO, Enéas Gonçalves de. Inovação tecnológica na indústria automobilística: características e evolução recente. **Economia e Sociedade**, Campinas, v. 17, n. 3, p.429-461, dez. 2008.SAE.

CUNHA, A. F.. **O que são sistemas embarcados?.** Saber Eletrônica, v. 414, p. 39-43, 2007.

CONNECTPARTS. **Como escolher o melhor Alarme Automotivo para seu carro**. 2014. Disponível em: <https://www.connectparts.com.br/guia-de-compras/alarme-automotivo>. Acesso em: 04 nov. 2019.

CASIMIRO, A.; RUFINO, J.; PINTO, R. C.; et al. A kernel-based architecture for safe cooperative vehicular functions. Proceedings of the 9th IEEE International Symposium on Industrial Embedded Systems (SIES 2014). **Anais...** p.228–237, 2014. IEEE. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=687 1208>. Acesso em: 30 out. 2019.

FRAINER, Daniel Massen. **A estrutura e a dinâmica da indústria automobilística no Brasil**. Porto Alegre: 2010.

GARCIA, Fernando Deluno. **Introdução aos sistemas embarcados e microcontroladores.** 2018. Disponível em: <https://www.embarcados.com.br/sistemas-embarcados-e-microcontroladores/>. Acesso em: 04 nov. 2019.

HILLENBRAND, M. et al. Development of electric/electronic architectures for safety-related vehicle functions. Software: Practice and Experience, [s.l.], v. 42, n. 7, p.817-851, 31 jan. 2012. Wiley. http://dx.doi.org/10.1002/spe.1154.

JOHANSSON, K. H.; TÖRNGREN, M.; NIELSEN, L. Vehicle applications of controller area network. **Handbook of Networked and Embedded Control Systems**. v. VI, p.741–765, 2005. Birkhäuser Boston.

LIMA, Izabelle. **Qual Arduino utilizar em seu projeto?** 2016. Disponível em: <https://autocorerobotica.blog.br/qual-arduino-utilizar-em-seu-projeto/>. Acesso em: 04 nov. 2019.

MATHWORKS. **Stateflow**. 2019. Disponível em: <https://la.mathworks.com/products/stateflow.html>. Acesso em: 04 nov. 2019.

MATHWORKS. **What is MIL, SIL, PIL, HIL and how do they integrate in Model Based Design approach?** 2019. Disponível em: <https://la.mathworks.com/matlabcentral/answers/440277-what-is-mil-sil-pil-hil-and-how-do-they-integrate-in-model-based-design-approach>. Acesso em: 04 nov. 2019.

NAPOL, Igor. **O que significam os 5 níveis da direção autônoma dos carros?** 2017. Disponível em: <https://www.tecmundo.com.br/carro/116608-significam-5-niveis-direcao-autonoma-carros.htm>. Acesso em: 04 nov. 2019.

NEME, João Henrique; SANTOS, Max Mauro Dias; TEIXEIRA, Evandro Leonardo Silva. Model Based Design for Automobile External Lighting Systems. **Sae Technical Paper Series**, [s.l.], 22 set. 2015. SAE International. http://dx.doi.org/10.4271/2015-36-0374.

POGGETO, Gustavo dal. **Critérios para seleção da arquitetura elétrica veicular em mercados emergentes.** 2009. 195 f. Dissertação (Mestrado) - Curso de Engenharia Automotiva, Escola Politécnica da Universidade de São Paulo, São Paulo, 2009.

RABELO, Laerte. **Diagnóstico do Sistema Eletrônico utilizando grandezas elétricas e instrumentos de medição**. 2017. Disponível em: <https://www.oficinabrasil.com.br/noticia/consultor-ob/diagnostico-do-sistema-eletronico-utilizando-grandezas-eletricas-e-instrumentos-de-medicao>. Acesso em: 04 nov. 2019.

SILVA, Gustavo Fernandes Alves da. **Projeto e Simulação de Funções Embarcadas Automotivas: Estudo de Caso para Carroceria**. 2017. 102 f. TCC (Graduação) - Curso de Bacharel em Engenharia Eletrônica, Departamento de Eletrônica, Universidade Tecnológica Federal do Paraná, Ponta Grossa, 2017.

SILVA, Rafael Rodrigues da. **Projeto de controladores para um sistema de direção elétrica utilizando a metodologia de projeto baseado em modelos**. 2017. 99 f. Dissertação (Mestrado) - Curso de Sistemas Mecatrônicos, Universidade de Brasília, Brasília, 2017.

SAUERWALD, B. M. CAN bus, Ethernet, or FPD-Link: Which is best for automotive communications?, 2014.

TUOHY, S.; GLAVIN, M.; HUGHES, C.; et al. Intra-Vehicle Networks: A Review. **IEEE Transactions on Intelligent Transportation Systems**, v. 16, n. 2, p. 534–545, 2015. Disponível em: <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=68 19448>.

TUOHY, Shane et al. Intra-Vehicle Networks: A Review. **Ieee Transactions On Intelligent Transportation Systems**, [s.l.], v. 16, n. 2, p.534-545, abr. 2015. Institute of Electrical and Electronics Engineers (IEEE).

WENDLING, M. Sensores. **Unesp**, v. 2, p. 19, 2010.

ANEXOS

Anexo A - Códigos C

```c
/* VF Nível do Óleo de Freio
    Autor: Fábio Santos
*/

int sensor = 4; //Definção das portas de cada variável
int chave = 2;
int espia = 8;
int bateria= A1;

float val=0; //Inicialização

void setup() {
pinMode(chave,INPUT); //Definição das entradas e saídas
pinMode(espia,OUTPUT);
pinMode(sensor,INPUT);

Serial.begin(9600);
}

void loop(){
  val= analogRead(bateria)*4.88e-3*3; //Leitura da porta analógica
  if(digitalRead(chave)==HIGH && val>=10.8){ //Condições de incialização
    if(digitalRead(sensor)==HIGH){ //Condição de nível baixo
        digitalWrite(espia,HIGH); // Alteração no estado do Led
    }else{
         digitalWrite(espia,LOW);
    }
   }else{
        digitalWrite(espia,LOW);
    }
  Serial.println(val);
}
```

```c
/* VF Assistente de Estacionamento
    Autor: Fábio Santos
*/
const int trigger1=7;
const int echo1=6;
float dist1;
const int trigger2=13;
const int echo2=12;
float dist2;
const int trigger3=8;
const int echo3=9;
float dist3;

int buzzer=10;
int chave = 3;
int re = 2;
int bateria= A1;
float val=0; //Inicialização

void setup(){
    Serial.begin(9600);
    pinMode(trigger1,OUTPUT);
    pinMode(echo1,INPUT);
    pinMode(trigger2,OUTPUT);
    pinMode(echo2,INPUT);
    pinMode(trigger3,OUTPUT);
```

```
    pinMode(echo3,INPUT);
    pinMode(buzzer, OUTPUT);
    pinMode(chave,INPUT);
    pinMode(re,INPUT);
 }

void loop(){
    val= analogRead(bateria)*4.88e-3; //Leitura da porta analógica

    if(digitalRead(chave)==HIGH && val>=3){ //Condições de incialização
        if(digitalRead(re)==HIGH){

            digitalWrite(trigger1,LOW);
            delayMicroseconds(5);

            digitalWrite(trigger1,HIGH);
            delayMicroseconds(5);
            digitalWrite(trigger1,LOW);

            dist1=pulseIn(echo1,HIGH);
            dist1=dist1/58;

            digitalWrite(trigger2,LOW);
            delayMicroseconds(5);

            digitalWrite(trigger2,HIGH);
            delayMicroseconds(5);
            digitalWrite(trigger2,LOW);

            dist2=pulseIn(echo2,HIGH);
            dist2=dist2/58;

            digitalWrite(trigger3,LOW);
            delayMicroseconds(5);

            digitalWrite(trigger3,HIGH);
            delayMicroseconds(5);
            digitalWrite(trigger3,LOW);

            dist3=pulseIn(echo3,HIGH);
            dist3=dist3/58;


    Serial.print("distancia1 =");
    Serial.print(dist1);
    Serial.print("distancia2 =");
    Serial.print(dist2);
    Serial.print("distancia3 =");
    Serial.print(dist3);
    Serial.write(10);
    delay(200);

        if( dist1 < 30 || dist2 < 30 || dist3 < 30 ){
            digitalWrite(10,100);
```

```
81
82          }
83          if( dist1 < 20 || dist2 < 20 || dist3 < 20 ){
84
85              digitalWrite(10,200);
86
87          }
88          if( dist1 < 10 || dist2 < 10 || dist3 < 10 ){
89              digitalWrite(10,255);
90
91          }else{
92              digitalWrite(10,0);
93          }
94        }
95      }
96  }
```

Anexo B – Códigos Simulink

File: NOF.c

```
1  /*
2  * File: NOF.c
3  *
4  * Code generated for Simulink model 'NOF'.
5  *
6  * Model version : 1.11
7  * Simulink Coder version : 9.0 (R2018b) 24-May-2018
8  * C/C++ source code generated on : Sun Oct 20 21:44:42 2019
9  *
10 * Target selection: ert.tlc
11 * Embedded hardware selection: Atmel->AVR
12 * Code generation objectives: Unspecified
13 * Validation result: Not run
14 */
15
16 #include "NOF.h"
17 #include "NOF_private.h"
18
19 /* Named constants for Chart: '<Root>/Chart' */
20 #define NOF_IN_BateriaBaixa ((uint8_T)1U)
21 #define NOF_IN_Desliga ((uint8_T)2U)
22 #define NOF_IN_DesligaE ((uint8_T)3U)
23 #define NOF_IN_Liga ((uint8_T)4U)
24 #define NOF_IN_LigaE ((uint8_T)5U)
25 #define NOF_IN_NO_ACTIVE_CHILD ((uint8_T)0U)
26
27 /* Block signals (default storage) */
28 B_NOF_T NOF_B;
29
30 /* Block states (default storage) */
31 DW_NOF_T NOF_DW;
32
33 /* Real-time model */
34 RT_MODEL_NOF_T NOF_M_;
35 RT_MODEL_NOF_T *const NOF_M = &NOF_M_;
36
37 /* Forward declaration for local functions */
```

```
38 static void matlabCodegenHandle_matlabCod_a(codertarget_arduinobase_block_T
   *obj);
39 static void matlabCodegenHandle_matlabCodeg(codertarget_arduinobase_int_a_T
   *obj);
40 static void matlabCodegenHandle_matlabC_ahu(codertarget_arduinobase_blo_a_T
   *obj);
41 real_T rt_roundd_snf(real_T u)
42 {
43 real_T y;
44 if (fabs(u) < 4.503599627370496E+15) {
45 if (u >= 0.5) {
46 y = floor(u + 0.5);
47 } else if (u > -0.5) {
48 y = u * 0.0;
49 } else {
50 y = ceil(u - 0.5);
51 }
52 } else {
53 y = u;
54 }
55
56 return y;
57 }
58
59 static void matlabCodegenHandle_matlabCod_a(codertarget_arduinobase_block_T
   *obj)
60 {
61 if (!obj->matlabCodegenIsDeleted) {
62 obj->matlabCodegenIsDeleted = true;
63 }
64 }
65
66 static void matlabCodegenHandle_matlabCodeg(codertarget_arduinobase_int_a_T
   *obj)
67 {
68 if (!obj->matlabCodegenIsDeleted) {
69 obj->matlabCodegenIsDeleted = true;
70 }
71 }
```

```
72
73  static void matlabCodegenHandle_matlabC_ahu(codertarget_arduinobase_blo_a_T
    *obj)
74  {
75  if (!obj->matlabCodegenIsDeleted) {
76  obj->matlabCodegenIsDeleted = true;
77  }
78  }
79
80  /* Model step function */
81  void NOF_step(void)
82  {
83  boolean_T rtb_DigitalInput_1_0;
84  uint16_T rtb_bateria_0;
85  boolean_T rtb_DigitalInput_0;
86  real_T sampleTime;
87  uint8_T tmp;
88
89  /* MATLABSystem: '<S2>/Digital Input' */
90  if (NOF_DW.obj_h.SampleTime != NOF_P.DigitalInput_SampleTime) {
91  NOF_DW.obj_h.SampleTime = NOF_P.DigitalInput_SampleTime;
92  }
93
94  rtb_DigitalInput_1_0 = readDigitalPin(4);
95
96  /* MATLABSystem: '<Root>/Analog Input' */
97  if (NOF_DW.obj.SampleTime != NOF_P.AnalogInput_SampleTime) {
98  NOF_DW.obj.SampleTime = NOF_P.AnalogInput_SampleTime;
99  }
100
101 MW_AnalogIn_Start(NOF_DW.obj.MW_ANALOGIN_HANDLE);
102 MW_AnalogInSingle_ReadResult(NOF_DW.obj.MW_ANALOGIN_HANDLE,
    &rtb_bateria_0, 3);
103
104 /* MATLABSystem: '<S3>/Digital Input' */
105 if (NOF_DW.obj_g.SampleTime != NOF_P.DigitalInput_SampleTime_j) {
106 if (((!rtIsInf(NOF_P.DigitalInput_SampleTime_j)) && (!rtIsNaN
107 (NOF_P.DigitalInput_SampleTime_j))) || rtIsInf
108 (NOF_P.DigitalInput_SampleTime_j)) {
```

```
109  sampleTime = NOF_P.DigitalInput_SampleTime_j;
110  }
111
112  NOF_DW.obj_g.SampleTime = sampleTime;
113  }
114
115  rtb_DigitalInput_0 = readDigitalPin(2);
116
117  /* Chart: '<Root>/Chart' incorporates:
118  * MATLABSystem: '<Root>/Analog Input'
119  * MATLABSystem: '<S2>/Digital Input'
120  * MATLABSystem: '<S3>/Digital Input'
121  */
122  if (NOF_DW.temporalCounter_i1 < 31U) {
123  NOF_DW.temporalCounter_i1++;
124  }
125
126  if (NOF_DW.is_active_c3_NOF == 0U) {
127  NOF_DW.is_active_c3_NOF = 1U;
128  NOF_DW.is_c3_NOF = NOF_IN_Desliga;
129  NOF_B.luzof = 0.0;
130  } else {
131  switch (NOF_DW.is_c3_NOF) {
132  case NOF_IN_BateriaBaixa:
133  if (NOF_DW.temporalCounter_i1 >= 30U) {
134  NOF_DW.tp_BateriaBaixa = 0U;
135  NOF_DW.is_c3_NOF = NOF_IN_Desliga;
136  NOF_B.luzof = 0.0;
137  }
138  break;
139
140  case NOF_IN_Desliga:
141  NOF_B.luzof = 0.0;
142  if (rtb_DigitalInput_0 && (rt_roundd_snf((real_T)rtb_bateria_0 * 0.00488) *
143  3.0 >= 10.8)) {
144  NOF_DW.is_c3_NOF = NOF_IN_Liga;
145  NOF_DW.tp_Liga = 1U;
146  }
147  break;
```

```
148
149  case NOF_IN_DesligaE:
150  NOF_B.luzof = 0.0;
151  if (rtb_DigitalInput_1_0) {
152  NOF_DW.is_c3_NOF = NOF_IN_LigaE;
153  NOF_B.luzof = 1.0;
154  } else {
155  NOF_DW.is_c3_NOF = NOF_IN_Liga;
156  NOF_DW.tp_Liga = 1U;
157  }
158  break;
159
160  case NOF_IN_Liga:
161  if (!rtb_DigitalInput_0) {
162  NOF_DW.tp_Liga = 0U;
163  NOF_DW.is_c3_NOF = NOF_IN_Desliga;
164  NOF_B.luzof = 0.0;
165  } else if (rt_roundd_snf((real_T)rtb_bateria_0 * 0.00488) * 3.0 < 10.8) {
166  NOF_DW.tp_Liga = 0U;
167  NOF_DW.is_c3_NOF = NOF_IN_BateriaBaixa;
168  NOF_DW.temporalCounter_i1 = 0U;
169  NOF_DW.tp_BateriaBaixa = 1U;
170  } else {
171  if (rtb_DigitalInput_1_0) {
172  NOF_DW.tp_Liga = 0U;
173  NOF_DW.is_c3_NOF = NOF_IN_DesligaE;
174  NOF_B.luzof = 0.0;
175  }
176  }
177  break;
178
179  default:
180  NOF_B.luzof = 1.0;
181  if (!rtb_DigitalInput_1_0) {
182  NOF_DW.is_c3_NOF = NOF_IN_DesligaE;
183  NOF_B.luzof = 0.0;
184  } else if (rt_roundd_snf((real_T)rtb_bateria_0 * 0.00488) * 3.0 < 10.8) {
185  NOF_DW.is_c3_NOF = NOF_IN_BateriaBaixa;
186  NOF_DW.temporalCounter_i1 = 0U;
```

```
187  NOF_DW.tp_BateriaBaixa = 1U;
188  } else {
189  if (!rtb_DigitalInput_0) {
190  NOF_DW.is_c3_NOF = NOF_IN_Desliga;
191  NOF_B.luzof = 0.0;
192  }
193  }
194  break;
195  }
196  }
197
198  /* End of Chart: '<Root>/Chart' */
199
200  /* DataTypeConversion: '<S4>/Data Type Conversion' */
201  if (NOF_B.luzof < 256.0) {
202  if (NOF_B.luzof >= 0.0) {
203  tmp = (uint8_T)NOF_B.luzof;
204  } else {
205  tmp = 0U;
206  }
207  } else {
208  tmp = MAX_uint8_T;
209  }
210
211  /* End of DataTypeConversion: '<S4>/Data Type Conversion' */
212
213  /* MATLABSystem: '<S4>/Digital Output' */
214  writeDigitalPin(8, tmp);
215  }
216
217  /* Model initialize function */
218  void NOF_initialize(void)
219  {
220  /* Registration code */
221
222  /* initialize non-finites */
223  rt_InitInfAndNaN(sizeof(real_T));
224
225  /* initialize error status */
```

```
226  rtmSetErrorStatus(NOF_M, (NULL));
227
228  /* block I/O */
229  (void) memset(((void *) &NOF_B), 0,
230  sizeof(B_NOF_T));
231
232  /* states (dwork) */
233  (void) memset((void *)&NOF_DW, 0,
234  sizeof(DW_NOF_T));
235
236  {
237  codertarget_arduinobase_int_a_T *obj;
238  MW_AnalogIn_TriggerSource_Type trigger_val;
239  real_T sampleTime;
240
241  /* Start for MATLABSystem: '<S2>/Digital Input' */
242  NOF_DW.obj_h.matlabCodegenIsDeleted = true;
243  NOF_DW.obj_h.isInitialized = 0L;
244  NOF_DW.obj_h.matlabCodegenIsDeleted = false;
245  NOF_DW.obj_h.SampleTime = NOF_P.DigitalInput_SampleTime;
246  NOF_DW.obj_h.isSetupComplete = false;
247  NOF_DW.obj_h.isInitialized = 1L;
248  digitalIOSetup(4, false);
249  NOF_DW.obj_h.isSetupComplete = true;
250
251  /* Start for MATLABSystem: '<Root>/Analog Input' */
252  NOF_DW.obj.matlabCodegenIsDeleted = true;
253  obj = &NOF_DW.obj;
254  NOF_DW.obj.isInitialized = 0L;
255  obj->Hw.AvailablePwmPinNames.f1 = '2';
256  obj->Hw.AvailablePwmPinNames.f2 = '3';
257  obj->Hw.AvailablePwmPinNames.f3 = '4';
258  obj->Hw.AvailablePwmPinNames.f4 = '5';
259  obj->Hw.AvailablePwmPinNames.f5 = '6';
260  obj->Hw.AvailablePwmPinNames.f6 = '7';
261  obj->Hw.AvailablePwmPinNames.f7 = '8';
262  obj->Hw.AvailablePwmPinNames.f8 = '9';
263  obj->Hw.AvailablePwmPinNames.f9[0] = '1';
264  obj->Hw.AvailablePwmPinNames.f9[1] = '0';
```

```
265  obj->Hw.AvailablePwmPinNames.f10[0] = '1';
266  obj->Hw.AvailablePwmPinNames.f10[1] = '1';
267  obj->Hw.AvailablePwmPinNames.f11[0] = '1';
268  obj->Hw.AvailablePwmPinNames.f11[1] = '2';
269  obj->Hw.AvailablePwmPinNames.f12[0] = '1';
270  obj->Hw.AvailablePwmPinNames.f12[1] = '3';
271  NOF_DW.obj.matlabCodegenIsDeleted = false;
272  NOF_DW.obj.SampleTime = NOF_P.AnalogInput_SampleTime;
273  obj = &NOF_DW.obj;
274  NOF_DW.obj.isSetupComplete = false;
275  NOF_DW.obj.isInitialized = 1L;
276  obj->MW_ANALOGIN_HANDLE = MW_AnalogInSingle_Open(1UL);
277  trigger_val = MW_ANALOGIN_SOFTWARE_TRIGGER;
278  MW_AnalogIn_SetTriggerSource(NOF_DW.obj.MW_ANALOGIN_HANDLE,
     trigger_val, 0UL);
279  NOF_DW.obj.isSetupComplete = true;
280
281  /* Start for MATLABSystem: '<S3>/Digital Input' */
282  NOF_DW.obj_g.matlabCodegenIsDeleted = true;
283  NOF_DW.obj_g.isInitialized = 0L;
284  NOF_DW.obj_g.matlabCodegenIsDeleted = false;
285  if (((!rtIsInf(NOF_P.DigitalInput_SampleTime_j)) && (!rtIsNaN
286  (NOF_P.DigitalInput_SampleTime_j))) || rtIsInf
287  (NOF_P.DigitalInput_SampleTime_j)) {
288  sampleTime = NOF_P.DigitalInput_SampleTime_j;
289  }
290
291  NOF_DW.obj_g.SampleTime = sampleTime;
292  NOF_DW.obj_g.isSetupComplete = false;
293  NOF_DW.obj_g.isInitialized = 1L;
294  digitalIOSetup(2, false);
295  NOF_DW.obj_g.isSetupComplete = true;
296
297  /* End of Start for MATLABSystem: '<S3>/Digital Input' */
298
299  /* Start for MATLABSystem: '<S4>/Digital Output' */
300  NOF_DW.obj_m.matlabCodegenIsDeleted = true;
301  NOF_DW.obj_m.isInitialized = 0L;
302  NOF_DW.obj_m.matlabCodegenIsDeleted = false;
```

```
303  NOF_DW.obj_m.isSetupComplete = false;
304  NOF_DW.obj_m.isInitialized = 1L;
305  digitalIOSetup(8, true);
306  NOF_DW.obj_m.isSetupComplete = true;
307
308  /* SystemInitialize for Chart: '<Root>/Chart' */
309  NOF_DW.tp_BateriaBaixa = 0U;
310  NOF_DW.temporalCounter_i1 = 0U;
311  NOF_DW.tp_Liga = 0U;
312  NOF_DW.is_active_c3_NOF = 0U;
313  NOF_DW.is_c3_NOF = NOF_IN_NO_ACTIVE_CHILD;
314  }
315  }
316
317  /* Model terminate function */
318  void NOF_terminate(void)
319  {
320  /* Terminate for MATLABSystem: '<S2>/Digital Input' */
321  matlabCodegenHandle_matlabCod_a(&NOF_DW.obj_h);
322
323  /* Terminate for MATLABSystem: '<Root>/Analog Input' */
324  matlabCodegenHandle_matlabCodeg(&NOF_DW.obj);
325
326  /* Terminate for MATLABSystem: '<S3>/Digital Input' */
327  matlabCodegenHandle_matlabCod_a(&NOF_DW.obj_g);
328
329  /* Terminate for MATLABSystem: '<S4>/Digital Output' */
330  matlabCodegenHandle_matlabC_ahu(&NOF_DW.obj_m);
331  }
332
333  /*
334  * File trailer for generated code.
335  *
336  * [EOF]
337  */
338
```

File: AE.c

```
1  /*
```

```
2  * File: AE.c
3  *
4  * Code generated for Simulink model 'AE'.
5  *
6  * Model version : 1.4
7  * Simulink Coder version : 9.0 (R2018b) 24-May-2018
8  * C/C++ source code generated on : Sun Nov 3 19:11:32 2019
9  *
10 * Target selection: ert.tlc
11 * Embedded hardware selection: Atmel->AVR
12 * Code generation objectives: Unspecified
13 * Validation result: Not run
14 */
15
16 #include "AE.h"
17 #include "AE_private.h"
18
19 /* Named constants for Chart: '<Root>/Chart' */
20 #define AE_IN_Atencao ((uint8_T)1U)
21 #define AE_IN_BateriaBaixa ((uint8_T)2U)
22 #define AE_IN_Desliga ((uint8_T)3U)
23 #define AE_IN_DesligaB ((uint8_T)4U)
24 #define AE_IN_Liga ((uint8_T)5U)
25 #define AE_IN_MuitoP ((uint8_T)6U)
26 #define AE_IN_NO_ACTIVE_CHILD ((uint8_T)0U)
27 #define AE_IN_Proximo ((uint8_T)7U)
28
29 /* Block signals (default storage) */
30 B_AE_T AE_B;
31
32 /* Block states (default storage) */
33 DW_AE_T AE_DW;
34
35 /* Real-time model */
36 RT_MODEL_AE_T AE_M_;
37 RT_MODEL_AE_T *const AE_M = &AE_M_;
38
39 /* Forward declaration for local functions */
```

```
40  static void matlabCodegenHandle_matlabCod_k(codertarget_arduinobase_in_kk_T
    *obj);
41  static void matlabCodegenHandle_matla_kk0zw(codertarget_arduinobase_block_T
    *obj);
42  static void AE_SystemCore_release(const codertarget_arduinobase_int_k_T *obj);
43  static void AE_SystemCore_delete(const codertarget_arduinobase_int_k_T *obj);
44  static void matlabCodegenHandle_matlabCodeg(codertarget_arduinobase_int_k_T
    *obj);
45  static codertarget_arduinobase_int_k_T *arduino_PWMOutput_arduino_PWMOu
46  (codertarget_arduinobase_int_k_T *obj);
47  real_T rt_roundd_snf(real_T u)
48  {
49  real_T y;
50  if (fabs(u) < 4.503599627370496E+15) {
51  if (u >= 0.5) {
52  y = floor(u + 0.5);
53  } else if (u > -0.5) {
54  y = u * 0.0;
55  } else {
56  y = ceil(u - 0.5);
57  }
58  } else {
59  y = u;
60  }
61
62  return y;
63  }
64
65  static void matlabCodegenHandle_matlabCod_k(codertarget_arduinobase_in_kk_T
    *obj)
66  {
67  if (!obj->matlabCodegenIsDeleted) {
68  obj->matlabCodegenIsDeleted = true;
69  }
70  }
71
72  static void matlabCodegenHandle_matla_kk0zw(codertarget_arduinobase_block_T
    *obj)
73  {
```

```
74  if (!obj->matlabCodegenIsDeleted) {
75  obj->matlabCodegenIsDeleted = true;
76  }
77  }
78
79  static void AE_SystemCore_release(const codertarget_arduinobase_int_k_T *obj)
80  {
81  if ((obj->isInitialized == 1L) && obj->isSetupComplete) {
82  MW_PWM_SetDutyCycle(obj->MW_PWM_HANDLE, 0);
83  }
84  }
85
86  static void AE_SystemCore_delete(const codertarget_arduinobase_int_k_T *obj)
87  {
88  AE_SystemCore_release(obj);
89  }
90
91  static void matlabCodegenHandle_matlabCodeg(codertarget_arduinobase_int_k_T
    *obj)
92  {
93  if (!obj->matlabCodegenIsDeleted) {
94  obj->matlabCodegenIsDeleted = true;
95  AE_SystemCore_delete(obj);
96  }
97  }
98
99  static codertarget_arduinobase_int_k_T *arduino_PWMOutput_arduino_PWMOu
100 (codertarget_arduinobase_int_k_T *obj)
101 {
102 codertarget_arduinobase_int_k_T *b_obj;
103 obj->isInitialized = 0L;
104 b_obj = obj;
105 obj->Hw.AvailablePwmPinNames.f1 = '2';
106 obj->Hw.AvailablePwmPinNames.f2 = '3';
107 obj->Hw.AvailablePwmPinNames.f3 = '4';
108 obj->Hw.AvailablePwmPinNames.f4 = '5';
109 obj->Hw.AvailablePwmPinNames.f5 = '6';
110 obj->Hw.AvailablePwmPinNames.f6 = '7';
111 obj->Hw.AvailablePwmPinNames.f7 = '8';
```

```
112  obj->Hw.AvailablePwmPinNames.f8 = '9';
113  obj->Hw.AvailablePwmPinNames.f9[0] = '1';
114  obj->Hw.AvailablePwmPinNames.f9[1] = '0';
115  obj->Hw.AvailablePwmPinNames.f10[0] = '1';
116  obj->Hw.AvailablePwmPinNames.f10[1] = '1';
117  obj->Hw.AvailablePwmPinNames.f11[0] = '1';
118  obj->Hw.AvailablePwmPinNames.f11[1] = '2';
119  obj->Hw.AvailablePwmPinNames.f12[0] = '1';
120  obj->Hw.AvailablePwmPinNames.f12[1] = '3';
121  obj->matlabCodegenIsDeleted = false;
122  return b_obj;
123  }
124
125  /* Model step function */
126  void AE_step(void)
127  {
128  uint16_T rtb_ut2_0;
129  uint16_T rtb_ut3_0;
130  uint16_T rtb_ut1_0;
131  uint16_T rtb_bateria_0;
132  boolean_T rtb_DigitalInput_1_0;
133  boolean_T rtb_DigitalInput_0;
134  real_T sampleTime;
135  real_T sampleTime_0;
136  real_T tmp;
137
138  /* MATLABSystem: '<Root>/UT2' */
139  if (AE_DW.obj.SampleTime != AE_P.UT2_SampleTime) {
140  AE_DW.obj.SampleTime = AE_P.UT2_SampleTime;
141  }
142
143  MW_AnalogIn_Start(AE_DW.obj.MW_ANALOGIN_HANDLE);
144  MW_AnalogInSingle_ReadResult(AE_DW.obj.MW_ANALOGIN_HANDLE,
     &rtb_ut2_0, 3);
145
146  /* MATLABSystem: '<Root>/UT3' */
147  if (AE_DW.obj_a.SampleTime != AE_P.UT3_SampleTime) {
148  AE_DW.obj_a.SampleTime = AE_P.UT3_SampleTime;
149  }
```

```
150
151  MW_AnalogIn_Start(AE_DW.obj_a.MW_ANALOGIN_HANDLE);
152  MW_AnalogInSingle_ReadResult(AE_DW.obj_a.MW_ANALOGIN_HANDLE,
     &rtb_ut3_0, 3);
153
154  /* MATLABSystem: '<Root>/UT1' */
155  if (AE_DW.obj_d.SampleTime != AE_P.UT1_SampleTime) {
156  AE_DW.obj_d.SampleTime = AE_P.UT1_SampleTime;
157  }
158
159  MW_AnalogIn_Start(AE_DW.obj_d.MW_ANALOGIN_HANDLE);
160  MW_AnalogInSingle_ReadResult(AE_DW.obj_d.MW_ANALOGIN_HANDLE,
     &rtb_ut1_0, 3);
161
162  /* MATLABSystem: '<Root>/bateria' */
163  if (AE_DW.obj_h.SampleTime != AE_P.bateria_SampleTime) {
164  AE_DW.obj_h.SampleTime = AE_P.bateria_SampleTime;
165  }
166
167  MW_AnalogIn_Start(AE_DW.obj_h.MW_ANALOGIN_HANDLE);
168  MW_AnalogInSingle_ReadResult(AE_DW.obj_h.MW_ANALOGIN_HANDLE,
     &rtb_bateria_0, 3);
169
170  /* MATLABSystem: '<S2>/Digital Input' */
171  if (AE_DW.obj_g.SampleTime != AE_P.DigitalInput_SampleTime) {
172  if (((!rtIsInf(AE_P.DigitalInput_SampleTime)) && (!rtIsNaN
173  (AE_P.DigitalInput_SampleTime))) || rtIsInf
174  (AE_P.DigitalInput_SampleTime)) {
175  sampleTime = AE_P.DigitalInput_SampleTime;
176  }
177
178  AE_DW.obj_g.SampleTime = sampleTime;
179  }
180
181  rtb_DigitalInput_l_0 = readDigitalPin(8);
182
183  /* MATLABSystem: '<S3>/Digital Input' */
184  if (AE_DW.obj_m.SampleTime != AE_P.DigitalInput_SampleTime_p) {
185  if (((!rtIsInf(AE_P.DigitalInput_SampleTime_p)) && (!rtIsNaN
```

```
186   (AE_P.DigitalInput_SampleTime_p))) || rtIsInf
187   (AE_P.DigitalInput_SampleTime_p)) {
188   sampleTime_0 = AE_P.DigitalInput_SampleTime_p;
189   }
190
191   AE_DW.obj_m.SampleTime = sampleTime_0;
192   }
193
194   rtb_DigitalInput_0 = readDigitalPin(9);
195
196   /* Chart: '<Root>/Chart' incorporates:
197   * MATLABSystem: '<Root>/UT1'
198   * MATLABSystem: '<Root>/UT2'
199   * MATLABSystem: '<Root>/UT3'
200   * MATLABSystem: '<Root>/bateria'
201   * MATLABSystem: '<S2>/Digital Input'
202   * MATLABSystem: '<S3>/Digital Input'
203   */
204   if (AE_DW.temporalCounter_i1 < 7U) {
205   AE_DW.temporalCounter_i1++;
206   }
207
208   if (AE_DW.is_active_c3_AE == 0U) {
209   AE_DW.is_active_c3_AE = 1U;
210   AE_DW.is_c3_AE = AE_IN_Desliga;
211   } else {
212   switch (AE_DW.is_c3_AE) {
213   case AE_IN_Atencao:
214   if ((rt_roundd_snf((real_T)rtb_ut2_0 * 0.00488) > 3.5) && (rt_roundd_snf
215   ((real_T)rtb_ut3_0 * 0.00488) > 3.5) && (rt_roundd_snf((real_T)
216   rtb_ut1_0 * 0.00488) > 3.5)) {
217   AE_DW.is_c3_AE = AE_IN_DesligaB;
218   AE_B.buzzer = 0U;
219   } else if (rtb_bateria_0 < 10.8) {
220   AE_B.buzzer = 0U;
221   AE_DW.is_c3_AE = AE_IN_BateriaBaixa;
222   AE_DW.temporalCounter_i1 = 0U;
223   AE_DW.tp_BateriaBaixa = 1U;
224   } else if ((rt_roundd_snf((real_T)rtb_ut2_0 * 0.00488) < 2.5) ||
```

```
225   (rt_roundd_snf((real_T)rtb_ut3_0 * 0.00488) < 2.5) ||
226   (rt_roundd_snf((real_T)rtb_ut1_0 * 0.00488) < 2.5)) {
227   AE_DW.is_c3_AE = AE_IN_Proximo;
228   AE_B.buzzer = 200U;
229   } else {
230   if ((!rtb_DigitalInput_1_0) || (!rtb_DigitalInput_0)) {
231   AE_B.buzzer = 0U;
232   AE_DW.is_c3_AE = AE_IN_Desliga;
233   }
234   }
235   break;
236
237   case AE_IN_BateriaBaixa:
238   if (AE_DW.temporalCounter_i1 >= 3U) {
239   AE_DW.tp_BateriaBaixa = 0U;
240   AE_DW.is_c3_AE = AE_IN_Desliga;
241   }
242   break;
243
244   case AE_IN_Desliga:
245   if (rtb_DigitalInput_1_0 && (rt_roundd_snf((real_T)rtb_bateria_0 * 0.00488)
246   * 3.0 >= 10.8) && rtb_DigitalInput_0) {
247   AE_DW.is_c3_AE = AE_IN_Liga;
248   AE_DW.tp_Liga = 1U;
249   }
250   break;
251
252   case AE_IN_DesligaB:
253   if ((rt_roundd_snf((real_T)rtb_ut2_0 * 0.00488) < 3.5) || (rt_roundd_snf
254   ((real_T)rtb_ut3_0 * 0.00488) < 3.5) || (rt_roundd_snf((real_T)
255   rtb_ut1_0 * 0.00488) < 3.5)) {
256   AE_DW.is_c3_AE = AE_IN_Atencao;
257   AE_B.buzzer = 100U;
258   } else {
259   if ((rt_roundd_snf((real_T)rtb_ut2_0 * 0.00488) > 3.5) || (rt_roundd_snf
260   ((real_T)rtb_ut3_0 * 0.00488) > 3.5) || (rt_roundd_snf((real_T)
261   rtb_ut1_0 * 0.00488) > 3.5)) {
262   AE_DW.is_c3_AE = AE_IN_Liga;
263   AE_DW.tp_Liga = 1U;
```

```
264   }
265   }
266   break;
267
268   case AE_IN_Liga:
269   if ((!rtb_DigitalInput_1_0) || (!rtb_DigitalInput_0)) {
270   AE_DW.tp_Liga = 0U;
271   AE_DW.is_c3_AE = AE_IN_Desliga;
272   } else if (rtb_bateria_0 < 10.8) {
273   AE_DW.tp_Liga = 0U;
274   AE_DW.is_c3_AE = AE_IN_BateriaBaixa;
275   AE_DW.temporalCounter_i1 = 0U;
276   AE_DW.tp_BateriaBaixa = 1U;
277   } else {
278   if ((rt_roundd_snf((real_T)rtb_ut2_0 * 0.00488) < 3.5) || (rt_roundd_snf
279   ((real_T)rtb_ut3_0 * 0.00488) < 3.5) || (rt_roundd_snf((real_T)
280   rtb_ut1_0 * 0.00488) < 3.5)) {
281   AE_DW.tp_Liga = 0U;
282   AE_DW.is_c3_AE = AE_IN_DesligaB;
283   AE_B.buzzer = 0U;
284   }
285   }
286   break;
287
288   case AE_IN_MuitoP:
289   if ((rt_roundd_snf((real_T)rtb_ut2_0 * 0.00488) > 1.5) && (rt_roundd_snf
290   ((real_T)rtb_ut3_0 * 0.00488) > 1.5) && (rt_roundd_snf((real_T)
291   rtb_ut1_0 * 0.00488) > 1.5)) {
292   AE_DW.is_c3_AE = AE_IN_Proximo;
293   AE_B.buzzer = 200U;
294   } else if (rtb_bateria_0 < 10.8) {
295   AE_B.buzzer = 0U;
296   AE_DW.is_c3_AE = AE_IN_BateriaBaixa;
297   AE_DW.temporalCounter_i1 = 0U;
298   AE_DW.tp_BateriaBaixa = 1U;
299   } else {
300   if ((!rtb_DigitalInput_1_0) || (!rtb_DigitalInput_0)) {
301   AE_B.buzzer = 0U;
302   AE_DW.is_c3_AE = AE_IN_Desliga;
```

```
303   }
304   }
305   break;
306
307   default:
308   sampleTime = rt_roundd_snf((real_T)rtb_ut2_0 * 0.00488);
309   sampleTime_0 = rt_roundd_snf((real_T)rtb_ut3_0 * 0.00488);
310   tmp = rt_roundd_snf((real_T)rtb_ut1_0 * 0.00488);
311   if ((sampleTime < 1.5) || (sampleTime_0 < 1.5) || (tmp < 1.5)) {
312   AE_DW.is_c3_AE = AE_IN_MuitoP;
313   AE_B.buzzer = 255U;
314   } else if (rtb_bateria_0 < 10.8) {
315   AE_B.buzzer = 0U;
316   AE_DW.is_c3_AE = AE_IN_BateriaBaixa;
317   AE_DW.temporalCounter_i1 = 0U;
318   AE_DW.tp_BateriaBaixa = 1U;
319   } else if ((sampleTime > 2.5) && (sampleTime_0 > 2.5) && (tmp > 2.5)) {
320   AE_DW.is_c3_AE = AE_IN_Atencao;
321   AE_B.buzzer = 100U;
322   } else {
323   if ((!rtb_DigitalInput_1_0) || (!rtb_DigitalInput_0)) {
324   AE_B.buzzer = 0U;
325   AE_DW.is_c3_AE = AE_IN_Desliga;
326   }
327   }
328   break;
329   }
330   }
331
332   /* End of Chart: '<Root>/Chart' */
333
334   /* MATLABSystem: '<Root>/Buzzer' */
335   if (AE_B.buzzer < 255.0) {
336   sampleTime = AE_B.buzzer;
337   } else {
338   sampleTime = 255.0;
339   }
340
341   MW_PWM_SetDutyCycle(AE_DW.obj_k.MW_PWM_HANDLE, sampleTime);
```

```
342
343  /* End of MATLABSystem: '<Root>/Buzzer' */
344  }
345
346  /* Model initialize function */
347  void AE_initialize(void)
348  {
349  /* Registration code */
350
351  /* initialize non-finites */
352  rt_InitInfAndNaN(sizeof(real_T));
353
354  /* initialize error status */
355  rtmSetErrorStatus(AE_M, (NULL));
356
357  /* block I/O */
358  (void) memset(((void *) &AE_B), 0,
359  sizeof(B_AE_T));
360
361  /* states (dwork) */
362  (void) memset((void *)&AE_DW, 0,
363  sizeof(DW_AE_T));
364
365  {
366  codertarget_arduinobase_in_kk_T *obj;
367  MW_AnalogIn_TriggerSource_Type trigger_val;
368  real_T sampleTime;
369  real_T sampleTime_0;
370  codertarget_arduinobase_int_k_T *obj_0;
371
372  /* Start for MATLABSystem: '<Root>/UT2' */
373  AE_DW.obj.matlabCodegenIsDeleted = true;
374  obj = &AE_DW.obj;
375  AE_DW.obj.isInitialized = 0L;
376  obj->Hw.AvailablePwmPinNames.f1 = '2';
377  obj->Hw.AvailablePwmPinNames.f2 = '3';
378  obj->Hw.AvailablePwmPinNames.f3 = '4';
379  obj->Hw.AvailablePwmPinNames.f4 = '5';
380  obj->Hw.AvailablePwmPinNames.f5 = '6';
```

```
381  obj->Hw.AvailablePwmPinNames.f6 = '7';
382  obj->Hw.AvailablePwmPinNames.f7 = '8';
383  obj->Hw.AvailablePwmPinNames.f8 = '9';
384  obj->Hw.AvailablePwmPinNames.f9[0] = '1';
385  obj->Hw.AvailablePwmPinNames.f9[1] = '0';
386  obj->Hw.AvailablePwmPinNames.f10[0] = '1';
387  obj->Hw.AvailablePwmPinNames.f10[1] = '1';
388  obj->Hw.AvailablePwmPinNames.f11[0] = '1';
389  obj->Hw.AvailablePwmPinNames.f11[1] = '2';
390  obj->Hw.AvailablePwmPinNames.f12[0] = '1';
391  obj->Hw.AvailablePwmPinNames.f12[1] = '3';
392  AE_DW.obj.matlabCodegenIsDeleted = false;
393  AE_DW.obj.SampleTime = AE_P.UT2_SampleTime;
394  obj = &AE_DW.obj;
395  AE_DW.obj.isSetupComplete = false;
396  AE_DW.obj.isInitialized = 1L;
397  obj->MW_ANALOGIN_HANDLE = MW_AnalogInSingle_Open(2UL);
398  trigger_val = MW_ANALOGIN_SOFTWARE_TRIGGER;
399  MW_AnalogIn_SetTriggerSource(AE_DW.obj.MW_ANALOGIN_HANDLE,
     trigger_val, 0UL);
400  AE_DW.obj.isSetupComplete = true;
401
402  /* Start for MATLABSystem: '<Root>/UT3' */
403  AE_DW.obj_a.matlabCodegenIsDeleted = true;
404  obj = &AE_DW.obj_a;
405  AE_DW.obj_a.isInitialized = 0L;
406  obj->Hw.AvailablePwmPinNames.f1 = '2';
407  obj->Hw.AvailablePwmPinNames.f2 = '3';
408  obj->Hw.AvailablePwmPinNames.f3 = '4';
409  obj->Hw.AvailablePwmPinNames.f4 = '5';
410  obj->Hw.AvailablePwmPinNames.f5 = '6';
411  obj->Hw.AvailablePwmPinNames.f6 = '7';
412  obj->Hw.AvailablePwmPinNames.f7 = '8';
413  obj->Hw.AvailablePwmPinNames.f8 = '9';
414  obj->Hw.AvailablePwmPinNames.f9[0] = '1';
415  obj->Hw.AvailablePwmPinNames.f9[1] = '0';
416  obj->Hw.AvailablePwmPinNames.f10[0] = '1';
417  obj->Hw.AvailablePwmPinNames.f10[1] = '1';
418  obj->Hw.AvailablePwmPinNames.f11[0] = '1';
```

```
419  obj->Hw.AvailablePwmPinNames.f11[1] = '2';
420  obj->Hw.AvailablePwmPinNames.f12[0] = '1';
421  obj->Hw.AvailablePwmPinNames.f12[1] = '3';
422  AE_DW.obj_a.matlabCodegenIsDeleted = false;
423  AE_DW.obj_a.SampleTime = AE_P.UT3_SampleTime;
424  obj = &AE_DW.obj_a;
425  AE_DW.obj_a.isSetupComplete = false;
426  AE_DW.obj_a.isInitialized = 1L;
427  obj->MW_ANALOGIN_HANDLE = MW_AnalogInSingle_Open(3UL);
428  trigger_val = MW_ANALOGIN_SOFTWARE_TRIGGER;
429  MW_AnalogIn_SetTriggerSource(AE_DW.obj_a.MW_ANALOGIN_HANDLE,
     trigger_val,
430  0UL);
431  AE_DW.obj_a.isSetupComplete = true;
432
433  /* Start for MATLABSystem: '<Root>/UT1' */
434  AE_DW.obj_d.matlabCodegenIsDeleted = true;
435  obj = &AE_DW.obj_d;
436  AE_DW.obj_d.isInitialized = 0L;
437  obj->Hw.AvailablePwmPinNames.f1 = '2';
438  obj->Hw.AvailablePwmPinNames.f2 = '3';
439  obj->Hw.AvailablePwmPinNames.f3 = '4';
440  obj->Hw.AvailablePwmPinNames.f4 = '5';
441  obj->Hw.AvailablePwmPinNames.f5 = '6';
442  obj->Hw.AvailablePwmPinNames.f6 = '7';
443  obj->Hw.AvailablePwmPinNames.f7 = '8';
444  obj->Hw.AvailablePwmPinNames.f8 = '9';
445  obj->Hw.AvailablePwmPinNames.f9[0] = '1';
446  obj->Hw.AvailablePwmPinNames.f9[1] = '0';
447  obj->Hw.AvailablePwmPinNames.f10[0] = '1';
448  obj->Hw.AvailablePwmPinNames.f10[1] = '1';
449  obj->Hw.AvailablePwmPinNames.f11[0] = '1';
450  obj->Hw.AvailablePwmPinNames.f11[1] = '2';
451  obj->Hw.AvailablePwmPinNames.f12[0] = '1';
452  obj->Hw.AvailablePwmPinNames.f12[1] = '3';
453  AE_DW.obj_d.matlabCodegenIsDeleted = false;
454  AE_DW.obj_d.SampleTime = AE_P.UT1_SampleTime;
455  obj = &AE_DW.obj_d;
456  AE_DW.obj_d.isSetupComplete = false;
```

```
457   AE_DW.obj_d.isInitialized = 1L;
458   obj->MW_ANALOGIN_HANDLE = MW_AnalogInSingle_Open(1UL);
459   trigger_val = MW_ANALOGIN_SOFTWARE_TRIGGER;
460   MW_AnalogIn_SetTriggerSource(AE_DW.obj_d.MW_ANALOGIN_HANDLE,
      trigger_val,
461   0UL);
462   AE_DW.obj_d.isSetupComplete = true;
463
464   /* Start for MATLABSystem: '<Root>/bateria' */
465   AE_DW.obj_h.matlabCodegenIsDeleted = true;
466   obj = &AE_DW.obj_h;
467   AE_DW.obj_h.isInitialized = 0L;
468   obj->Hw.AvailablePwmPinNames.f1 = '2';
469   obj->Hw.AvailablePwmPinNames.f2 = '3';
470   obj->Hw.AvailablePwmPinNames.f3 = '4';
471   obj->Hw.AvailablePwmPinNames.f4 = '5';
472   obj->Hw.AvailablePwmPinNames.f5 = '6';
473   obj->Hw.AvailablePwmPinNames.f6 = '7';
474   obj->Hw.AvailablePwmPinNames.f7 = '8';
475   obj->Hw.AvailablePwmPinNames.f8 = '9';
476   obj->Hw.AvailablePwmPinNames.f9[0] = '1';
477   obj->Hw.AvailablePwmPinNames.f9[1] = '0';
478   obj->Hw.AvailablePwmPinNames.f10[0] = '1';
479   obj->Hw.AvailablePwmPinNames.f10[1] = '1';
480   obj->Hw.AvailablePwmPinNames.f11[0] = '1';
481   obj->Hw.AvailablePwmPinNames.f11[1] = '2';
482   obj->Hw.AvailablePwmPinNames.f12[0] = '1';
483   obj->Hw.AvailablePwmPinNames.f12[1] = '3';
484   AE_DW.obj_h.matlabCodegenIsDeleted = false;
485   AE_DW.obj_h.SampleTime = AE_P.bateria_SampleTime;
486   obj = &AE_DW.obj_h;
487   AE_DW.obj_h.isSetupComplete = false;
488   AE_DW.obj_h.isInitialized = 1L;
489   obj->MW_ANALOGIN_HANDLE = MW_AnalogInSingle_Open(4UL);
490   trigger_val = MW_ANALOGIN_SOFTWARE_TRIGGER;
491   MW_AnalogIn_SetTriggerSource(AE_DW.obj_h.MW_ANALOGIN_HANDLE,
      trigger_val,
492   0UL);
493   AE_DW.obj_h.isSetupComplete = true;
```

```
494
495  /* Start for MATLABSystem: '<S2>/Digital Input' */
496  AE_DW.obj_g.matlabCodegenIsDeleted = true;
497  AE_DW.obj_g.isInitialized = 0L;
498  AE_DW.obj_g.matlabCodegenIsDeleted = false;
499  if (((!rtIsInf(AE_P.DigitalInput_SampleTime)) && (!rtIsNaN
500  (AE_P.DigitalInput_SampleTime))) || rtIsInf
501  (AE_P.DigitalInput_SampleTime)) {
502  sampleTime = AE_P.DigitalInput_SampleTime;
503  }
504
505  AE_DW.obj_g.SampleTime = sampleTime;
506  AE_DW.obj_g.isSetupComplete = false;
507  AE_DW.obj_g.isInitialized = 1L;
508  digitalIOSetup(8, false);
509  AE_DW.obj_g.isSetupComplete = true;
510
511  /* End of Start for MATLABSystem: '<S2>/Digital Input' */
512
513  /* Start for MATLABSystem: '<S3>/Digital Input' */
514  AE_DW.obj_m.matlabCodegenIsDeleted = true;
515  AE_DW.obj_m.isInitialized = 0L;
516  AE_DW.obj_m.matlabCodegenIsDeleted = false;
517  if (((!rtIsInf(AE_P.DigitalInput_SampleTime_p)) && (!rtIsNaN
518  (AE_P.DigitalInput_SampleTime_p))) || rtIsInf
519  (AE_P.DigitalInput_SampleTime_p)) {
520  sampleTime_0 = AE_P.DigitalInput_SampleTime_p;
521  }
522
523  AE_DW.obj_m.SampleTime = sampleTime_0;
524  AE_DW.obj_m.isSetupComplete = false;
525  AE_DW.obj_m.isInitialized = 1L;
526  digitalIOSetup(9, false);
527  AE_DW.obj_m.isSetupComplete = true;
528
529  /* End of Start for MATLABSystem: '<S3>/Digital Input' */
530
531  /* Start for MATLABSystem: '<Root>/Buzzer' */
532  AE_DW.obj_k.matlabCodegenIsDeleted = true;
```

```
533   arduino_PWMOutput_arduino_PWMOu(&AE_DW.obj_k);
534   obj_0 = &AE_DW.obj_k;
535   AE_DW.obj_k.isSetupComplete = false;
536   AE_DW.obj_k.isInitialized = 1L;
537   obj_0->MW_PWM_HANDLE = MW_PWM_Open(5UL, 0.0, 0.0);
538   MW_PWM_Start(AE_DW.obj_k.MW_PWM_HANDLE);
539   AE_DW.obj_k.isSetupComplete = true;
540
541   /* SystemInitialize for Chart: '<Root>/Chart' */
542   AE_DW.tp_BateriaBaixa = 0U;
543   AE_DW.temporalCounter_i1 = 0U;
544   AE_DW.tp_Liga = 0U;
545   AE_DW.is_active_c3_AE = 0U;
546   AE_DW.is_c3_AE = AE_IN_NO_ACTIVE_CHILD;
547   }
548   }
549
550   /* Model terminate function */
551   void AE_terminate(void)
552   {
553   /* Terminate for MATLABSystem: '<Root>/UT2' */
554   matlabCodegenHandle_matlabCod_k(&AE_DW.obj);
555
556   /* Terminate for MATLABSystem: '<Root>/UT3' */
557   matlabCodegenHandle_matlabCod_k(&AE_DW.obj_a);
558
559   /* Terminate for MATLABSystem: '<Root>/UT1' */
560   matlabCodegenHandle_matlabCod_k(&AE_DW.obj_d);
561
562   /* Terminate for MATLABSystem: '<Root>/bateria' */
563   matlabCodegenHandle_matlabCod_k(&AE_DW.obj_h);
564
565   /* Terminate for MATLABSystem: '<S2>/Digital Input' */
566   matlabCodegenHandle_matla_kk0zw(&AE_DW.obj_g);
567
568   /* Terminate for MATLABSystem: '<S3>/Digital Input' */
569   matlabCodegenHandle_matla_kk0zw(&AE_DW.obj_m);
570
571   /* Terminate for MATLABSystem: '<Root>/Buzzer' */
```

```
572  matlabCodegenHandle_matlabCodeg(&AE_DW.obj_k);
573  }
574
575  /*
576   * File trailer for generated code.
577   *
578   * [EOF]
579   */
580
```