UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO DE ELETRÔNICA
CURSO DE ENGENHARIA ELETRÔNICA

GUSTAVO FERNANDES ALVES DA SILVA

Projeto e Simulação de Funções Embarcadas Automotivas: Estudo de Caso para Carrocerias

TRABALHO DE CONCLUSÃO DE CURSO

PONTA GROSSA
2017

GUSTAVO FERNANDES ALVES DA SILVA

Projeto e Simulação de Funções Embarcadas Automotivas: Estudo de Caso para Carroceria

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Bacharel em Engenharia Eletrônica, do Departamento de Eletrônica (DAELE), da Universidade Tecnológica Federal do Paraná.

Orientador: Prof. Dr. Max Mauro Santos.

PONTA GROSSA

2017

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Campus Ponta Grossa

Diretoria de Graduação e Educação Profissional
Engenharia Eletrônica

TERMO DE APROVAÇÃO

Projeto e Simulação de Funções Embarcadas Automotivas: Estudo de Caso para Carroceria

por

GUSTAVO FERNANDES ALVES DA SILVA

Este Trabalho de Conclusão de Curso (TCC) foi apresentado em 27 de Junho de 2017 como requisito parcial para a obtenção do título de Bacharel em Engenharia Eletrônica. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____

MAX MAURO SANTOS

Prof. Orientador

_____

ABRAHAM ELIAS ORTEGA PAREDES
Membro titular

_____

JEFERSON JOSÉ GOMES

Membro titular

- A Folha de Aprovação assinada encontra-se arquivada na Secretaria Acadêmica -

AGRADECIMENTOS

Aos meus pais Hernani e Fátima, a minha irmã Ângela, minha sogra Janete, meu sogro “Tato”, minha namorada Talita e amigos mais próximos por todo apoio que me dão desde sempre.

Ao Professor Max Mauro pela colaboração que tem dado durante toda a graduação.

A UTFPR, funcionários e professores que de alguma forma colaboraram positivamente.

RESUMO

FERNANDES, Gustavo. **Projeto e Simulação de Funções Embarcadas Automotivas: Estudo de Caso para Carroceria**. 2017. 56 páginas. Trabalho de Conclusão de Curso (Bacharelado em Engenharia Eletrônica) - Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2017.

Os sistemas embarcados automotivos são sistemas mecatrônicos complexos e que exigem aplicação de métodos e ferramentas na fases de criação e testes. Desta forma, há um fluxo de processos para desenvolvimento de funções de um sistema automotivo. O modelo de controle, o código gerado à partir dele e seu funcionamento no controlador são submetidos a testes para validação. Este trabalho realiza um estudo de caso sobre as fases de desenvolvimento de *software* automotivo, propõe a criação de modelos de controle para funções de carroceria de um veículo utilizando as ferramentas da *Mathwork*s® e realiza as duas primeiras etapas de testes.

Palavras-chave: Simulação, Sistemas Embarcados Automotivos, Unidade de Controle Eletrônica, Módulo de Controle de Carroceria.

ABSTRACT

FERNANDES, Gustavo. **Project and Simulation of Automotive Embedded Systems: A Body's Case Study.** . 2017. 56 páginas. Trabalho de Conclusão de Curso (Bacharelado em Engenharia Eletrônica) - Federal Technology University - Parana. Ponta Grossa, 2017.

The automotive embedded systems are complex mecatronic systems that demands the use of tools and methods during its development. Therefore, there's a work flow of an automotive embedded function development. O modelo de controle, o código gerado à partir dele e seu funcionamento no controlador são submetidos a testes para validação. The control model, the generated code and its working on a controller must be tested in order to be validated. This paper is a case study about the automotive software development process, proposes the generation of control model for automotive body functions using *Mathworks*® tools and makes the two first tests for it.

Keywords: Simulation; Automotive Embedded System; Electronic Unit Control; Body Control Module

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

BCM	*Body Control Module*
ECU	*Electronic Control Unit*
ECM	*Engine Control Module*
E/E	*Elétrica e Eletrônica*
HMI	*Human Machine Interface*
ICM	*Instrument Control Module*
IPC	*Instrument Panel Cluster*
USB	*Universal Serial Bus*

SUMÁRIO

1. INTRODUÇÃO

Para o domínio de aplicação automotivo, sistemas de controle embarcados são atualmente o foco principal de inovações e melhoria na qualidade destes produtos. Isto acontece em função da crescente demanda dinâmica dos atores deste mercado. O intensivo uso de componentes elétricos e eletrônicos na indústria automotiva tem impulsionado inovações no desenvolvimento e produção que visam diminuir custos, tempo de produção e melhorando a qualidade do produto final, proporcionando maior conforto e segurança. Consequentemente, engenheiros têm trabalhado para aumentar a funcionalidade destes elementos de eletrônica em substituição a sistemas mecânicos

No domínio automotivo os subsistemas automotivos são gerenciados por unidades de controle eletrônicas denominadas simplesmente por ECU (*Electronic Control Unit*). Um veículo, que é um conjunto de subsistemas, possui diversos tipos de ECUs capazes de gerenciar funções como controle do motor, transmissão, carroceria, entre outros. Para o gerenciamento de cada um destes subsistemas as ECUs são nomeadas de acordo com o propósito de controle, sendo 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, entre outro.

A engenharia de *software* automotiva exige a adoção de métodos, processos, ferramentas e padronizações adequados que assegurem aos produtos automotivos um nível de confiabilidade, segurança e conforto perceptíveis aos clientes e todos envolvidos . Este processo possui três etapas: desenvolvimento; produção e serviços.

Este trabalho apresenta ferramentas e conceitos sobre eletrônica embarcada automotiva, e exercita os conceitos elaborando diagramas de estados de funções controladas por uma *BCM* que podem ser reutilizados futuramente para as fases de testes e que podem gerar um código C para implementação em controlador.

1.1. PROBLEMA

Atualmente existe uma ampla utilização de sistemas embarcados automotivos, mas desenvolvidos utilizando diferentes técnicas e conceitos. Portanto para conseguir efetivamente integrar os diferentes módulos, as companhias precisam desenvolver uma tecnologia nova. Para evitar esse problema, aplica-se o conceito de desenvolvimento de *software* independente de arquitetura de *hardware*, o que possibilita a reutilização de funções já desenvolvidas, diminuindo o tempo da etapa de desenvolvimento e permitido o maior foco nas fases de testes.

1.2. OBJETIVOS

1.2.1 Objetivo Geral

Definir a lógica de funcionamento e realizar a geração do código computacional para controle de 5 funções controlados pelo Módulo de Controle de Carroceria.

1.2.2 Objetivos Específicos

- Desenvolver a estratégia de controle de cada função baseada nos eventos aos quais estão sujeitas;
- Desenvolver diagrama *Stateflow* das funções;
- Confrontar os sinais de teste e as respostas esperadas;
- Realizar a geração do código em linguagem C;

1.3. JUSTIFICATIVA

Os sistemas utilizados nos veículos são desenvolvidos por diferentes fabricantes, sendo que por vezes existe uma grande dificuldade em fazer com que eles operem facilmente em conjunto, uma vez que são desenvolvidos utilizando diferentes técnicas e padrões. Contudo, há algumas iniciativas no setor objetivando a

unificação dos métodos, conceitos e ferramentos para o desenvolvimento de *software* automotivo. Estes conceitos e conhecimentos vêm para nosso país com determinada limitação e atraso, visto que os veículos com elevado nível de funções ainda são desenvolvidos, fabricados e consumidos fora de nosso território nacional. Desta forma, temos assim um motivador para estudos na área de desenvolvimento de *software* automotivo como forma de se aprofundar no tema, além da contribuição de projetar e gerar códigos de uma função automotiva.

1.4. METODOLOGIA

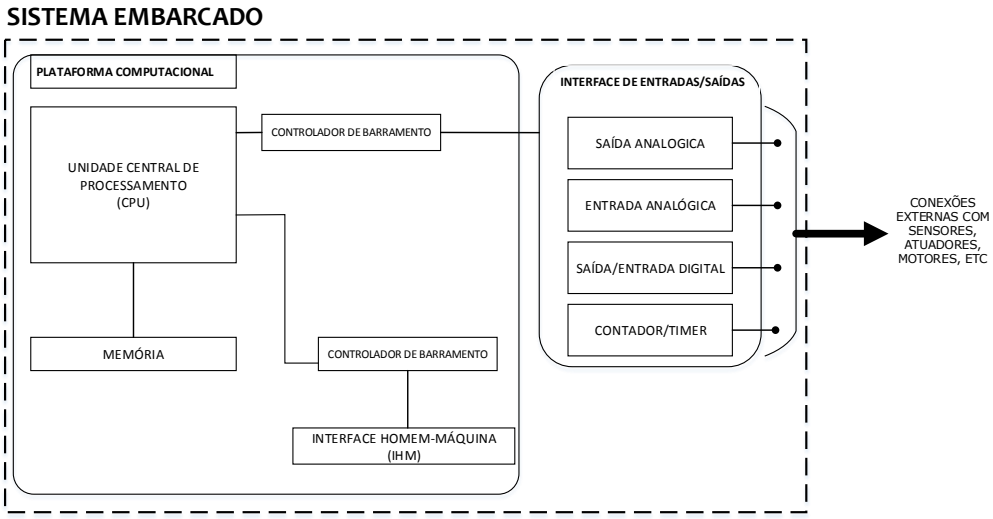
O desenvolvimento deste trabalho é baseado em pesquisas bibliográficas, e informações disponibilizadas por fabricantes. A partir do embasamento teórico e entendimento de todas as variáveis e funções do processo, serão desenvolvidos os modelos de controle para as funções propostas. Em seguida, será realizada a simulação e ajuste dos modelos, antes de chegar à etapa final onde serão gerados os códigos computacionais de cada função.

2. REFERENCIAL TEÓRICO

2.1 SISTEMAS EMBARCADOS

Sistemas Embarcados são sistemas computacionais fisicamente limitados, com restrições de memória, tamanho, energia e que possuem um número limitado e específico de funções. São sistemas criados para desempenhar tarefas específicas, e implementados em um circuito integrado dedicado e que trabalham independente de outras operações e oferecendo respostas em tempo real. Geralmente estão embutidos em outro produto, como um eletrodoméstico ou um veículo. (TANENBAUM, 2003). Seu funcionamento pode ser baseado na ação de um usuário ou em respostas automáticas a eventos do meio externo (Figura 1) .

Figura 1 - Arquitetura Básica de um Sistema Embarcado



Fonte: Autoria Própria

Quando estes sistemas não dependem apenas de seu comportamento funcional, mas também de seu comportamento temporal, eles são classificados como sistemas embarcados de tempo real (MARWEDEL; GOOSSENS, 2013), (MARQUES; SIEGERT; BRISOLARA, 2014). Sistemas de tempo real respondem a eventos externos em tempo hábil e o tempo de resposta é garantido (LI; YAO, 2003).

Diferentemente do computador pessoal de propósito geral, que executa diversos programas, os sistemas embarcados não possuem flexibilidade de *software* e/ou de *hardware* que os permita fazer outra tarefa que não seja aquela para o qual foi desenvolvido. Ao surgir a necessidade de alteração em seu funcionamento, é necessário que seja feito o *update* de seu *software* com novas versões com correções desejadas ou ainda mesmo a criação de novas funções que o tornem mas robusto. Geralmente, este tipo de atualização é feita pelos fabricantes.

2.2 SISTEMAS EMBARCADOS AUTOMOTIVOS

Aproveitando-se do rápido desenvolvimento da eletrônica, a indústria automobilística vem trabalhando para uma melhora contínua dos sistemas automotivos, não só no que diz respeito a segurança do condutor e dos passageiros, mas para o carro como um todo. A eletrônica está presente desde sistemas de antifurto, até sistemas mais complexos que controlam a estabilidade do veículo.

Inicialmente, o uso de eletrônica em veículos se restringia aos módulos de injeção eletrônica, fazendo o controle do motor e seus sensores. Logo após, foi implementado o conceito de *Instrument Panel Cluster (IPC),* que substituíram pouco a pouco os antigos medidores mecânicos por sistemas eletrônicos digitais, com *displays* multicoloridos de lâmpadas e tubos de raios catódicos, até evoluir para os LEDs (diodos emissores de luz) e LCD (*Liquid Crystal Display*), proporcionando uma para o usuário uma maior facildade na monitoração de parâmetros essenciais para o bom funcionamento do veículo, como quantidade de combustível ou velocidade, e aprimorando a experiência de diagnóstico de falhas (Figura 2).

Atualmente, a variedade dos sistemas embarcados está presente tanto em sistemas cuja finalidade é aumentar o conforto e melhorar a experiência dos usuários, por exemplo, informações de tráfego, sistemas de GPS, climatizadores e janelas elétricas, em sistemas de segurança como travas de portas, alarmes e avisos de ausência de cinto de segurança, e, finalmente, em sistemas críticos e que podem afetar a dinâmica e padrões de segurança do veículo, como freios inteligentes, controle de estabilidade e controle de tração

Figura 2 - Modelo antigo de painel (esquerda) e modelo moderno (direita)

Fonte: blog.thedetroithub.com

Um veículo moderno pode conter em sua arquitetura eletrônica entre 70 a 100 ECUs e um veículo comum contém entre 30 a 50 ECUs (CHARETTE, 2010). Para realizar a comunicação entre esses módulos, é comum um veículo comum tem dois ou três barramentos (BUS) de *Controller Area Network* (CAN), com taxas de 25 a 500 Kbps, dois ou três barramentos de rede de baixa velocidade que fazem a interconexão local e, opcionalmente, alguns links de alta velocidade para troca de informações. As aplicações de segurança são planejadas para serem executadas, normalmente a taxas entre 20 a 100 ms. Estas aplicações apresentam desafios devido a alta distribuição, complexidade e interoperabilidade (SANGIOVANNI-VINCENTELLI; NATALE, 2007).

Os sistemas embarcados automotivos são classificados comos sistemas de tempo real, e são divididos entre domínios funcionais “centrados nos veículos”, como controle de trem de força (motor e sistema de transmissão), controle de chassi e sistemas de segurança ativos ou passivos, e domínios “centrados nos passageiros” que podem ser identificados nos sistemas de multimídia/telemática, corpo/conforto e interface homem-máquina (ZURAWSKI, 2009). O domínio funcional “centrado no passageiro” identificado no corpo/conforto contém funções incorporadas em um veículo que estão relacionadas aos sistemas de limpadores de vidro, de faróis, de portas, de janelas, de poltronas, de airbag e de retrovisores. Geralmente, estes sistemas não estão sujeitos a restrições rigorosas de desempenho. Do ponto de vista da segurança, eles não representam uma parte crítica do sistema. No entanto, existem

certas funções que têm de respeitar restrições de tempo real, por exemplo, o sistema de airbag, que tem na segurança dos passageiros (ZURAWSKI, 2009).

2.3 – DOMÍNIOS DE SISTEMAS EMBARCADOS

Os fabricantes costumam distinguir esses domínios onde há presença de eletrônica embarcada. Historicamente, são citados 5 (NAVET, 2009):

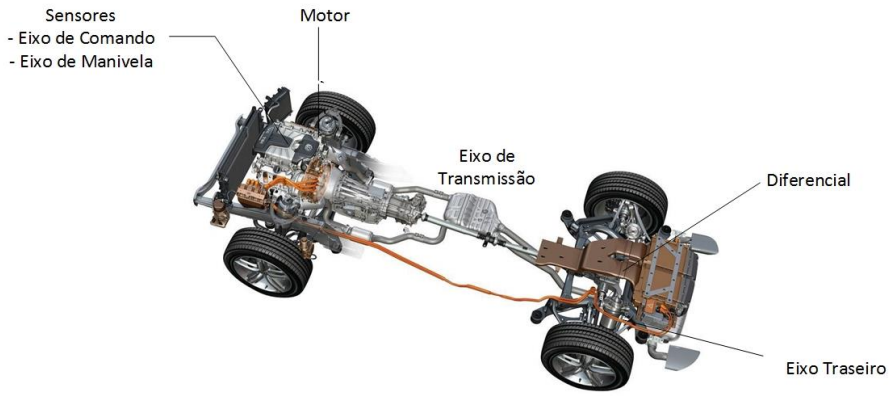
- Conjunto de Trem de Força
- Chassis
- Carroceria
- HMI (*Human-Machine Interface*)
- Telemática

2.3.1 Trem de Força

Este domínio é relativo a sistemas que controlam o motor de acordo com ações do motorista como acelerar ou desacelerar o veículo, de acordo com a posição de um pedal de aceleração ou de frenagem, ou mesmo responder a requisições de outros módulos do carro, como o de controle de temperatura ou de controle de estabilidade. É designado para que possa facilitar a experiência do motorista e otimizar consumo de recursos como o combustível, entre outros.

As aplicações deste domínio requer sensores cujas especificações precisam ter como critério a relação custo/resolução e que possuam certa robustez, pois estão suscetíveis a interferências e a calor, uma vez que trabalham muito próximos ao motor. Também é importante ressaltar que seus sensores e atuadores precisam obedecer a requisitos de tempo de resposta, pois se tratam de funções que afetam diretamente a dinâmica do veículo e a segurança dos usuários. (NAVET, 2009). A Figura 3 traz alguns exemploes de funções deste domínio.

Figura 3 - Sistemas do domínio de Trem de Força



Fonte: Autoria Própria

2.3.2 Chassis

É composto por sistemas que têm como objetivo controlar e otimizar a interação do veículo com a estrada, visando partes como suspensão e rodas, por exemplo. Este domínio atua respondendo a requisições do motorista, como mudança de direção ou aceleração e analisando o perfil do ambiente onde o veículo está, analisando, por exemplo, paràmetros de umidade na estrada ou vento. Em seu domínio estão funcionalidades bem conhecidas, como Freio ABS, controle de estabilidade automático e tração das rodas.

Assim como o domínio de *Trem de Força,* afeta diretamente a dinâmica do veículo, visa otimizar a experiência de direção e aumentar a segurança de seus usuários. Desta forma, o controlador também deve prover tempos de respostas precisos e ter a capacidade de trabalho num ambiente mais hostil.

2.3.3 Telemática

Incluem sistemas com suporte à troca de informações entre o veículo e a infra-estrutura da estrada ou mesmo com outros veículos. O rápido avanço da capacidade de processamento de controladores e de implantação de redes de comunicações móveis tem viabilizado o desenvolvimento de sistemas de navegação, de segurança e de serviços de emergência para os casos de acidentes e de problemas mecânicos.

Também têm auxiliado em casos de roubo, possibilitando o rastreamento, diagnóstico mecânico remoto e banco de dados com informações médicas do motorista (Figura 4)

Figura 4 - Exemplo de estrutura de Telemática Automotiva

Fonte: www.keystoneict.com

2.3.4 HMI

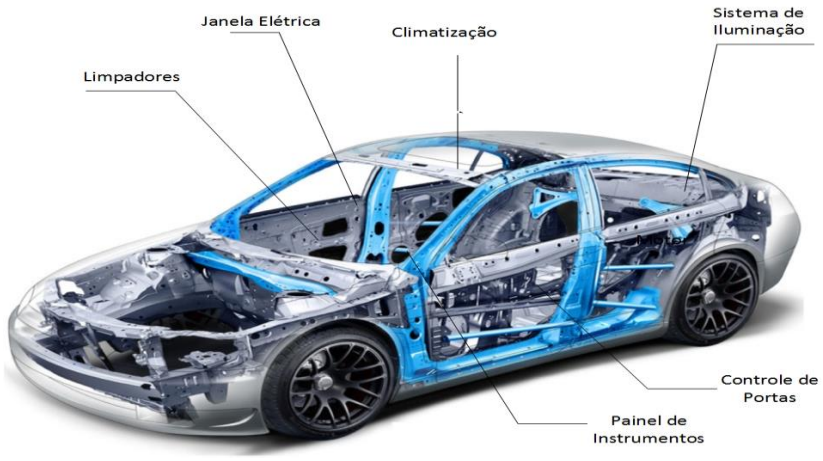
Incluem sistemas que, de uma forma geral, possibilitam a interação entre o motorista e os demais passageiros, apresentando informações de *status* do carro como velocidade, nível de óleo, estados das porta, estados das luzes, etc.

2.3.5 Carroceria

Contém funcionalidades que não possuem relação direta com a dinâmica do veículo, mas que podem maximizar a experiência de seus passageiros. Limpadores, luzes, travas elétricas, portas elétricas, controle de espelhos e climatizadores são exemplos dessas funções. Do ponto de vista de segurança, não representam uma parte crítica do sistema. Entretanto, há algumas funções de segurança, por exemplo, as que implicam em acesso ao veículo, que necessitam respeitar certas limitações de tempo (NAVET 2009).

A Figura 5 traz alguns exemplos de funções deste domínio:

Figura 5 - Funções do Domínio de Carroceria

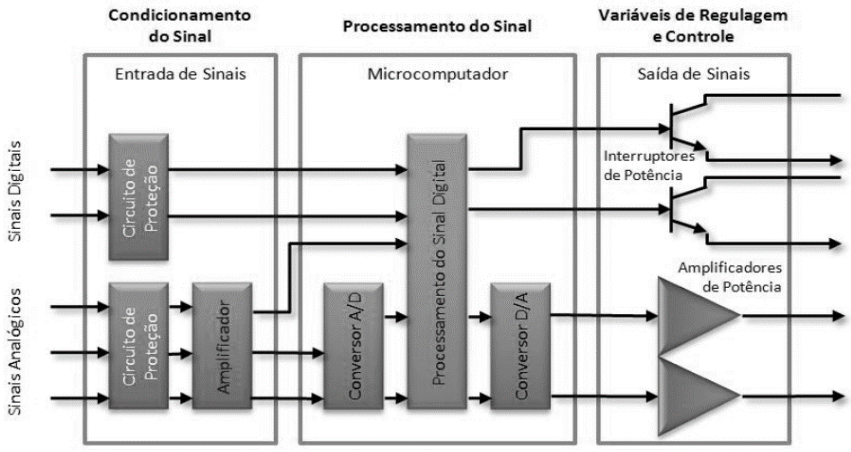


Fonte: Autoria Própria

2.4 UNIDADE DE CONTROLE ELETRÔNICA (*ECU – Electronic Control Unit)*

Os subsistemas automotivos são gerenciados por um módulo conhecido como Unidades de Controle Eletrônica (*ECU - Electronic Control Unit*). A ECU é responsável por realizar leituras de entradas, interpretação dos dados lidos, acionar saída e gerenciar protocolos de comunicação (Figura 6). Possui internamente uma placa de circuito impresso com um microprocessador ou microcontrolador e um programa gravado em uma memória.

Figura 6 - Arquitetura Básica de uma ECU



Fonte: Guimarães, 2007

Uma ECU automotiva é genericamente formada pelos seguintes elementos:

• **Interfaces de Entrada** (Sensores e Entradas de comunicação): Os sensores capturam continuamente parâmetros significantes na forma de variáveis físicas ou estados, tais como velocidade do motor, velocidade do veículo ou temperatura e os converte para um valor elétrico correspondente. As entradas de comunicação são interfaces padronizadas que recebem *frames* de dados e controle de protocolos que contém sinais de grandezas físicas e estados;

• **Software de Controle**: Baseado nas informações de entrada, um algoritmo ou estratégia de controle implementada em *software* determina a ação que a ser realizada, controlando ou regulando as saídas do veículo;

• **Interfaces de Saída** : As interfaces de saída controlam os atuadores para realizar uma ação de controle determinada pela malha de controle implementada no *software* executado sob o microcontrolador dentro da ECU; as saídas de comunicação são interfaces padronizadas que transmitem *frames* de dados e controle de protocolos que contém sinais de grandezas físicas e estados relacionados ao sistema a controlar.

. Um veículo é um conjunto de subsistemas que possui diversos tipos de ECUs capazes de gerenciar desde funções críticas como controle do motor e transmissão, até outras menos prioritárias nas diferentes funções de carroceria, como vidro elétrico, trava elétrica, retrovisores elétricos e teto solar elétrico. Para o gerenciamento de cada um destes subsistemas, as ECUs são nomeadas de acordo com o propósito de controle, como por exemplo: *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, dentre outros (SANTOS, 2010).

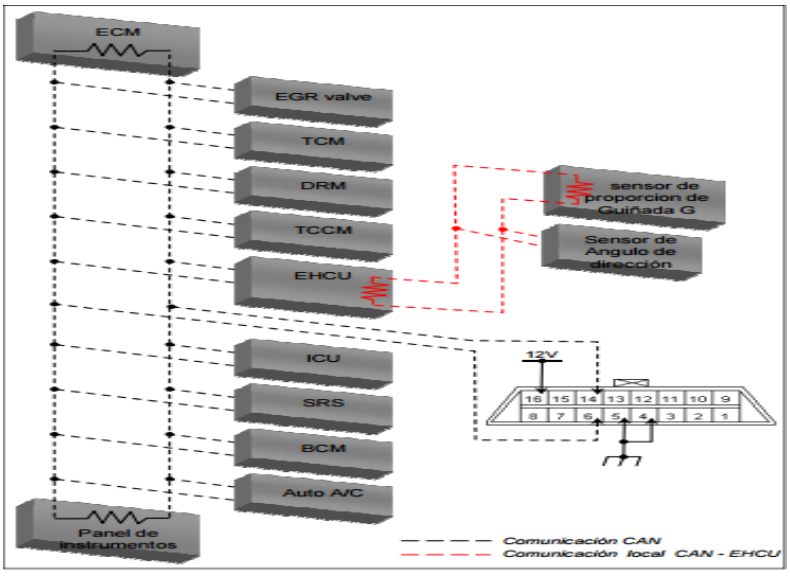
Os vários tipos de ECUs existentes se diferenciam pela natureza das funções realizadas e pelas características técnicas relativas ao *hardware*. Essas diferenças também influeciam no local de instalação da unidade. (GUIMARÃES, 2007)

2.4.1 Módulo de Controle de Carroceria (*BCM – Body Control Module)*

Nos primeiros veículos presença de eletrônica, como sistemas de iluminação e controle de portas, esses sistemas eram controlados por diferentes componentes. O módulo conhecido como BCM reuniu a função de todos esses dispositivos independentes para facilitar o controle e melhorar o desempenho do sistema elétrico do automóvel, centralizando o controle de funções de carroceira, de conforto e apoio ao usuário.

O propósito da *BCM é* enviar e repassar comandos de controle, leitura de sensores, ativação de relés e gerenciar a troca de dados entre diferentes unidades de controles. Para isso, realiza muitas funções que um computador normal executa, incluindo armazenar dados em uma memória volátil, assim como também envia e recebe sinais por ondas de rádio. A *BCM* , na essência, é uma *ECU* que controla diferentes *ECU*s distribuídas pelo veículo, mantendo constante troca informações com outras centrais do carro (Figura 7).

Figura 7: ECUs que trocam informação com a BCM



Fonte: automotrizenvideo.com

Além de receber informações de *status* do veículo e de outros módulos para diagnóstico, a *BCM* ainda controla funções de personalização visando maior conforto do usuário. Funções como climatização, janelas elétricas, controle de assentos,

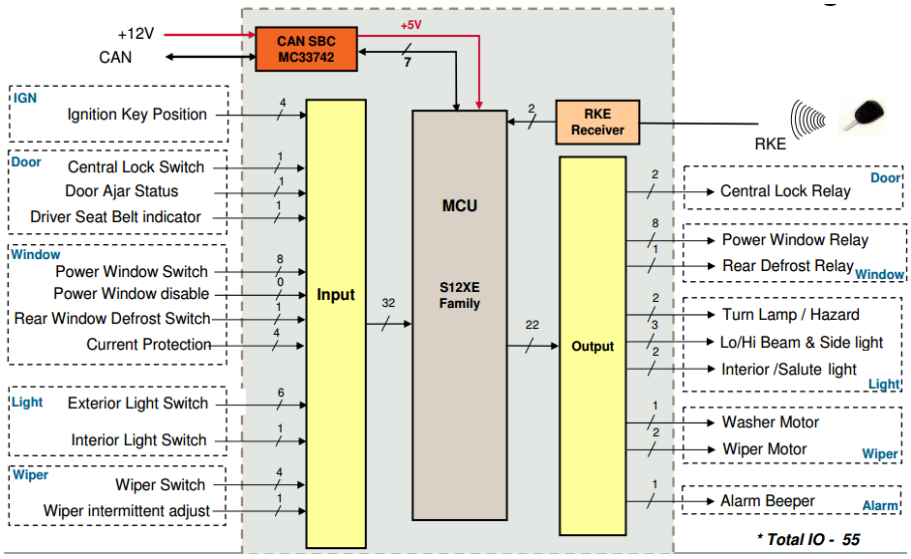
controle de retrovisores e espelhos e limpador automatizado não interferem na dinâmica ou na condução do veículo, mas tem efeito direto tanto na experiência de utilização do veículo, como nos requisitos mínimos *de hardware* da *BCM.*

.2.4.2 Classificação de BCMs

Uma consequência natural da relação *Custos* vs *Requisitos do Usuário* vs *Requisitos da BCM* é uma diferenciação dos módulos quanto a poder de processamento e número de entradas e saídas.

É classificada como *Low End* uma *BCM* que contém apenas funções básicas de controle, diagnóstico e conforto. É o modelo comum em veículos de menor custo ou em seus modelos básicos. Do ponto de vista do desenvolvimento, o efeito prático dessa diferenciação é uma *BCM* com um número reduzido de entradas e um menor poder computacional, o que impacta em uma menor complexidade do controlador (Figura 8).

Figura 8: Diagrama de Blocos de uma *Low End BCM* com exemplos de funções básicas.



Fonte: Freescale

De modo análogo, há as chamadas *High End BCM*, voltadas a produtos de maior custo e modelos superiores, para veículos que tem como foco usuários de maior poder aquisitivo ou que busquem um veículo com as tecnologias de ponta. O

acréscimo de funções de conforto personalizadas ocasiona um rápido crescimento do número de dispositivos de entrada e saída da *BCM*, principalmente em funções que podem ter interface com todos os passageiros do carro, como por exemplo, uma janela elétrica.

A Tabela 1 traz um rápido comparativo entre especificações técnicas de controladores da fabricante *Infineon*, expondo as diferenças de capacidade computacional e de interfaces de entradas e saídas.

Tabela 1 – Comparativo de um controlador *BCM High End* e um *Low End*.

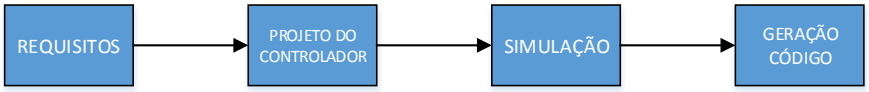
Especificação	*High End*	*Low End*
Modelo	XC2200H	XC2200L
Clock	100 MHz	66MHz
Pinos Entrada/Saída	150	49
Memória *Flash*	1.6 Mbytes	64-160 Kbytes
Memória *SRAM*	138 Kbytes	12 Kbytes

Fonte: Autoria Própria

3. DESENVOLVIMENTO

A primeira etapa do trabalho consiste na definição dos requisitos para funcionamento do controlador, estabelecendo a lógica de funcionamento, quais as entradas e saídas necessárias e como elas devem se comportar. Na etapa de projeto do controlador são elaborados os diagramas de estados de cada função, definindo a base para as fases de simulação e geração de código (Figura 9).

Figura 9 – Etapas do trabalho



Fonte: Autoria Própria

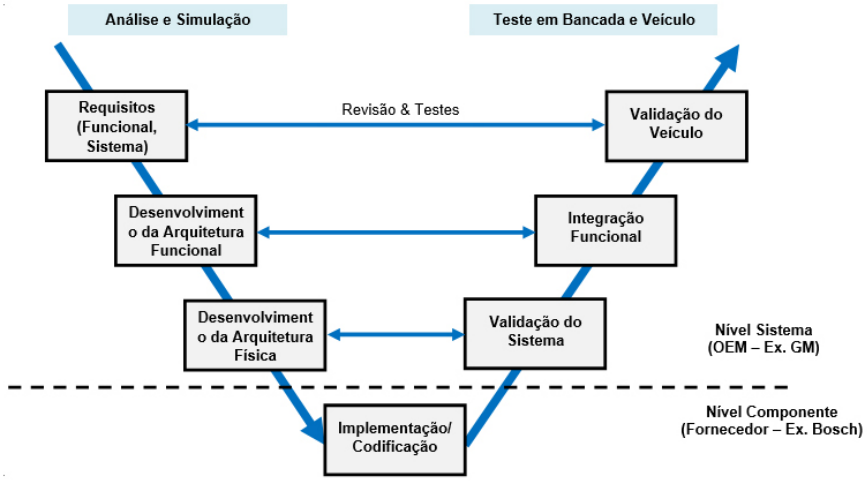
3.1 Modelo V

O desenvolvimento de *software* embarcado automotivo sob uma Arquitetura Elétrica e Eletrônica (E\E) é baseado na metodologia de desenvolvimento *V-Model*, que já possui considerável grau de maturidade. A especificação é definida em conjunto e coordenadamente entre as matrizes dos fabricantes originais de equipamentos (*Original Equipment Manufacturer* ou OEMs), fornecedores e fornecedores de ferramentas. O processo tem como obetivo enfatizar a qualidade e o controle do *software* de forma a detectar erros antecipadamente.

A primeira etapa é a de definição dos requisitos do sistema desejado, seguido da modelagem do funcionamento da planta, chamada de arquitetura funcional. Em seguida, baseada na arquitetura funcional é criada a arquitetura física ou planta física, que consiste em uma montagem física de componentes que simulam o sistema planejado. Com as arquiteturas já desenvolvidas é possível gerar o código em linguagem C do controlador. A estrutura funcional é utilizada para validação via *software*, onde a planta ainda está sendo executada exclusivamente em *software* e o controlador executado tanto em hardware quanto em *software*. Utilizando a arquitetura

física, que pode ser considerada a planta física do modelo, é possível se realizar a verificação com o controlador sendo executado tanto em *software* quanto em *hardware*. A verificação utilizando as plantas funcional e física, até esta altura, é desenvolvida e testada em uma estrutura de diagramas de blocos. Com o código gerado automaticamente na quarta etapa é possível fazer a validação do sistema em ambas as arquiteturas, porém utilizando o código em linguagem C para controlar o processo e não mais modelos de diagramas de blocos. Por último a integração funcional e a validação no veículo servem para fazer a verificação e validação final em ambientes reais, seja em protótipos ou no alvo final do desenvolvimento. (NEME, 2014). A Figura 10 traz uma representação visual destas etapas.

Figura 10 - Exemplo de diagrama do modelo em V



Fonte: Neme, 2014

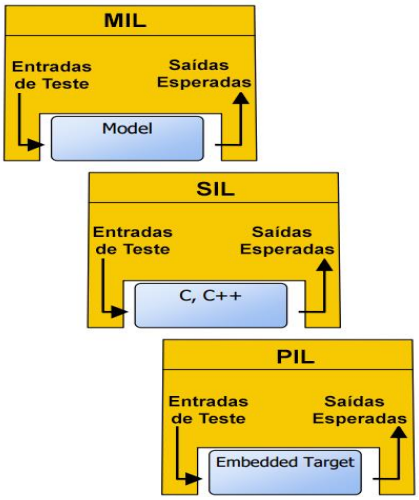
. Esta metodologia é amplamente aplicada no desenvolvimento de software automotivo em sintonia com fornecedores. Os desenvolvedores de ferramentas disponibilizam ferramentas adequadas para que cada fase e ciclo do desenvolvimento possam ser posteriormente integrados ao processo de desenvolvimento de produto da OEM.

Este trabalho focará nas 4 primeiras fases, ou seja, no desenvolvimento do *software* de controle e sua implementação.

3.2 TESTES EM MODELOS

Para as fases de testes, são utilizadas tradicionais técnicas de validação chamadas *Model-in-the-Loop*, *Software-in-the-Loop*, *Processor in-the-Loop* e *Hardware-in-the-Loop*. A utilização destes métodos segue uma sequência definida, além disso, as três primeiras etapas utilizam as mesmas entradas de teste (Figura 11)

Figura 11 - Sequência de métodos para verificação de sistema embarcado

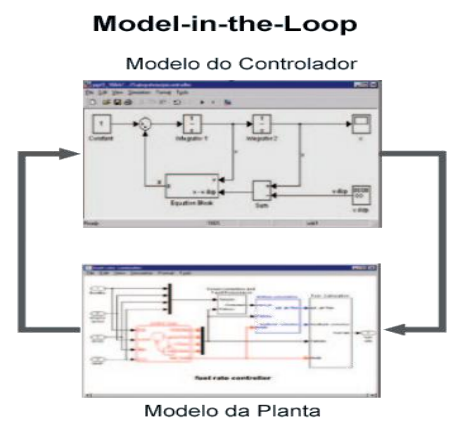


Fonte: Mathworks, 2017.

3.2.1 Model In Loop

É o primeiro estágio de simulação que serve como referência para os estágios seguintes e fornece os valores mínimos e máximos das variáveis. A estratégia de controle e o modelo físico são de desenvolvimento rápido, possibilitando mudanças e testes rápidos no sistema (Neme, 2014). A Figura 12 ilustra esta etapa.

Figura 12 – Model In Loop

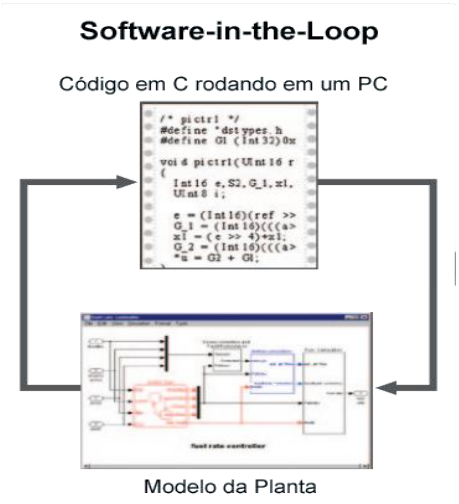


Fonte: Neme, 2014.

3.2.2 Software in Loop

É o estágio onde a ferramenta de geração automática de código fornece a estratégia de controle estabelecida na fase anterior automaticamente em código em linguagem C ou C++. É essencialmente um teste para o sistema de geração de código (seja feito de maneira automática ou manual). A interação do modelo diminui um pouco com relação ao, mas já é possível enxergar erros de codificação (Figura 13) (Neme, 2014).

Figura 13 – Software in the Loop

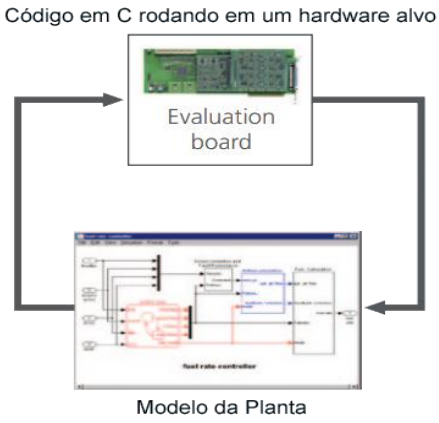


Fonte: Neme, 2014

3.2.3 Processor in the Loop

Nesta fase não se roda mais o código do sistema em simulação, pois ele é implementado em um microcontrolador. Enquanto o código do controle está rodando no microcontrolador, a planta ainda está sendo representada por um diagrama de Simulink®. Este teste é projetado para expor problemas com a execução em um sistema embarcado. Neste estágio as principais tarefas envolvem medição de memória, perfil de tempo de execução e verificação do código alvo. A Figura 14 ilustra esta etapa.

Figura 14 – Processor in the Loop

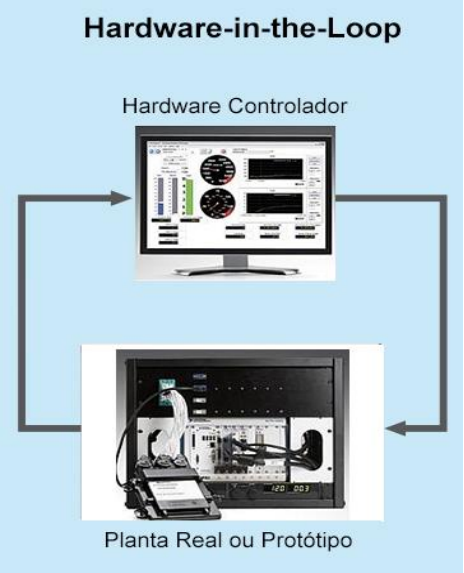


Fonte: Neme, 2014.

3.2.4 Hardware In the Loop

Nesta etapa o sistema de controle está instalado no sistema final de controle e pode apenas interagir com a planta através das entradas apropriadas do controlador. A planta é simulada em um computador em tempo real com entradas e saídas simuladas para fazer com que o controlador se comporte como se estivesse instalado na planta real. Este teste é muito próximo da realidade da aplicação final e por isso deixa claro a maior parte dos problemas que poderão ser encontrados. A fase de *Hardware in the Loop* é ilustrada pela Figura 15.

Figura 15 – Hardware in the Loop



Fonte: Neme (2014)

3.3 FERRAMENTAS PARA DESENVOLVIMENTO

3.3.1 Simulink

Simulink é um ambiente para simulação, geração de código testes contínuos e verificação de sistemas embarcados baseado em modelagem por Diagrama de Blocos. Provê ao usuário bibliotecas com blocos funcionais que podem ser customizáveis e diferentes solucionadores matemáticos para modelagem e simulação de sistemas dinâmicos.

3.3.2 Stateflow

Stateflow® é um ambiente para modelagem e simulação de combinações e sequências de decisões lógicas baseado no conceito de máquinas de estados e diagramas de fluxo. Permite a combinação de representações gráficas e Tabelas, incluindo diagramas de transição de estados, diagramas de fluxos e Tabelas-verdade para realização a modelagem de sistemas e analisar como este sistema reage a eventos externos, eventos temporais e entrada de sinais externos.

3.3.3 Plataforma *Arduino*

Arduino é uma plataforma de prototipagem eletrônica de código aberto baseada em *hardware* e *software* flexíveis e fáceis de usar, é composta basicamente por um controlador Atmel AVR de 8 bits, uma interface *serial* ou *USB* e alguns pinos digitais e analógicos, possibilitando leitura de sensores, controle de atuadores e processamento de informações. A plataforma utiliza-se de uma camada simples de *software* implementada na placa, um *bootloader*, que seria um trecho de código que informa ao componente como realizar a inicialização para execução e programação do código pelo usuário, além de uma interface no computador que utiliza a linguagem "*Processing*", baseada na linguagem C/C++, a qual também é *open source* (SOUZA, 2011). A Tabela 2 traz as principais informações sobre o Microcontrolador.

Tabela 2 - Especificações Arduino UNO

Especificações Arduino UNO	
Processador	Atmega328P
Clock	16 Mhz
Entradas/Saídas Digitais	14
Entradas Analógicas	5
Memória Flash	32 kB
Memória EEPROM	2 kB
Memória	1 kB

Fonte: Autoria Própria

3.4 FUNÇÕES DESENVOLVIDAS

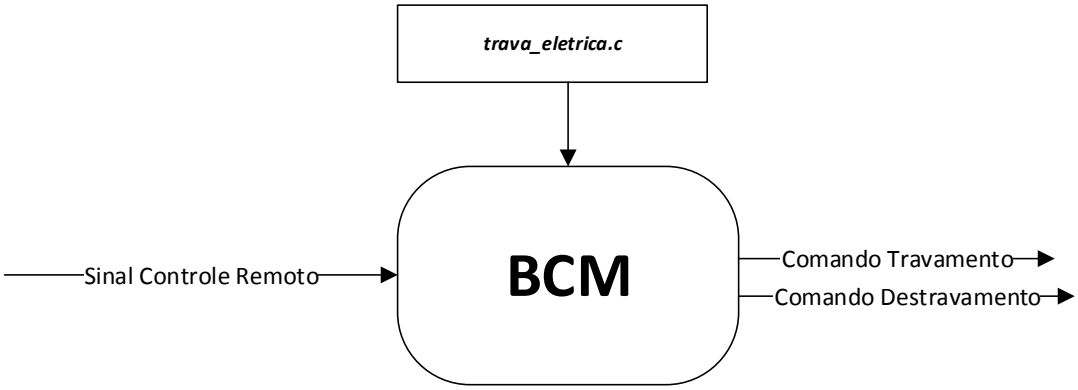
A proposta deste trabalho é a elaboração dos diagramas de estados de funções controladas por uma *BCM (Body Control Module)* no ambiente *Simulink* ® com a biblioteca *Stateflow*® e, à partir deles, obter um código para implementação em um microcontrolador. O código gerado será utilizado na plataforma *Arduino,* por questões de custos, facilidade e objetividade, mas deve ser facilmente embarcável em qualquer outra arquitetura disponível, uma vez que com a ajuda do *Embedded Coder* o código C da função pode ser gerado.

3.4.1 Trava Elétrica

O sistema de trava elétrica foi escolhido por ser tratar de uma função presente quase que na totalidade dos modelos atuais, dos mais simples aos mais sofisticados. Embora não afete a dinâmica do carro, está intimamente ligada com a segurança pois pode permitir acesso ao veículo, requerendo um rápido tempo de resposta.

O diagrama de blocos exemplificando a função é mostrado na Figura 16:

Figura 16 – Diagrama de blocos do sistema de trava elétrica;



Fonte: Autoria Própria

Para acionamento da trava elétrica de uma porta foram usadas 3 entradas e 2 saídas (Tabela 3):

Tabela 3 – Entradas e saídas do sistema de trava elétrica

Nome	Tipo
lock_ c	Entrada Digital
unlock_c	Entrada Digital
lock_on	Saida Digital
lock_off	Saída Digital

Fonte: Autoria Própria

A entrada *lock_c* pode assumir 2 valores: 1 é a requisição para travamento, 0 é o estado ocioso.

A entrada un*lock_c* pode assumir 2 valores: 1 é a requisição para destravamento, 0 é o estado ocioso.

A saída *lock_on* pode assume valor 1 quando for enviado o comando para travamento;

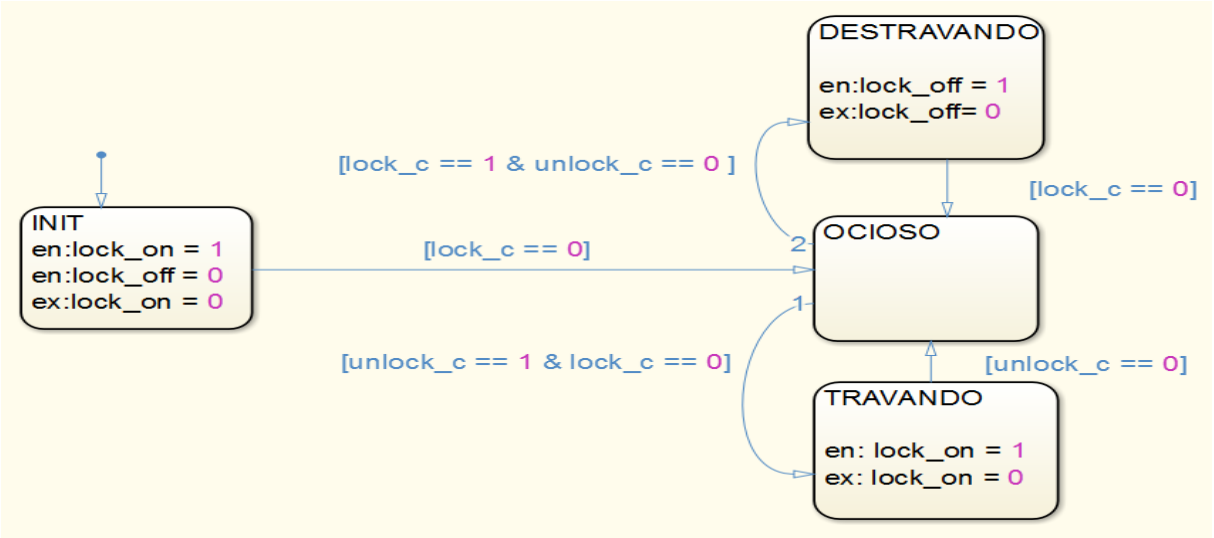
A saída *lock_of* pode assume valor 1 quando for enviado o comando para destravamento;

A execução da função deve obedecer a seguinte lógica:

1) Para ganhar acesso ao veículo, deve-se realizar o destravamento através do sinal do controle remoto (entrada *unlock_c*);
2) A variável *unlock_on* é a saída para destravamento
3) Para realizar o travamento, aciona-se a entrada *lock_on;*
4) A função pode ser executada sem chave na ignição;

Com essas informações , foi obtido o diagrama *Matlab/Stateflow* da Figura 17:

Figura 17 – Diagrama *Stateflow* da função *Trava Elétrica*.



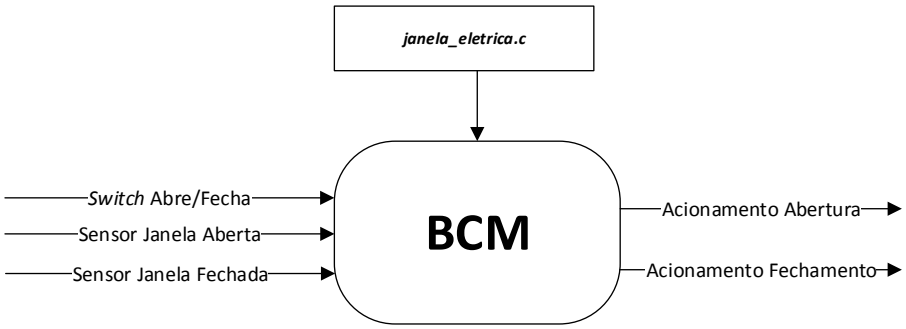
Fonte: Autoria Própria.

3.4.2 Janela Elétrica

Este sistema foi selecionado porque, embora esteja presente em modelos mais simples, é comumente utilizado em metade de suas possibilidades – apenas nas portas frontais do veículo. Seu uso é interessante pois devido a alta utilização de dispositivos de entradas e saídas de dados, causa um sensível impacto na definição da capacidade da *BCM*.

O diagrama de blocos da função é mostrado na Figura 18 :

Figura 18 – Diagrama de blocos do sistema e janela elétrica



Fonte: Autoria Própria

Para acionamento da janela elétrica de uma porta foram usadas 4 entradas e 2 saídas (Tabela 4).

Tabela 4 – Entradas e saídas do sistema de Janela Elétrica

Nome	Tipo
sensor_up	Entrada Digital
sensor_down	Entrada Digital
abre	Entrada Digital
fecha	Entrada Digital
motor_up	Saida Digital
motor_down	Saída Digital

Fonte: Autoria Própria

A entrada *sensor_up* pode assumir 2 valores: 1 indica que a janela está totalmente fechada, 0 indica abertura parcial.

A entrada *sensor_down* pode assumir 2 valores: 1 indica abertura total da janela, 0 indica abertura parcial.

A entrada *abre* pode assumir 2 valores: 1 é o comando para abrir a janela, 0 é o estado ocioso.

A entrada *fecha* pode assumir 2 valores: 1 é o comando para fechar a janela, 0 é o estado ocioso.

A entradas *motor_up* e *motor_*down podem assumir 2 valores: 0, para desligado, e 1 para acionamento.

A execução da função deve obedecer a seguinte lógica:

1) O veículo deve estar ligado;

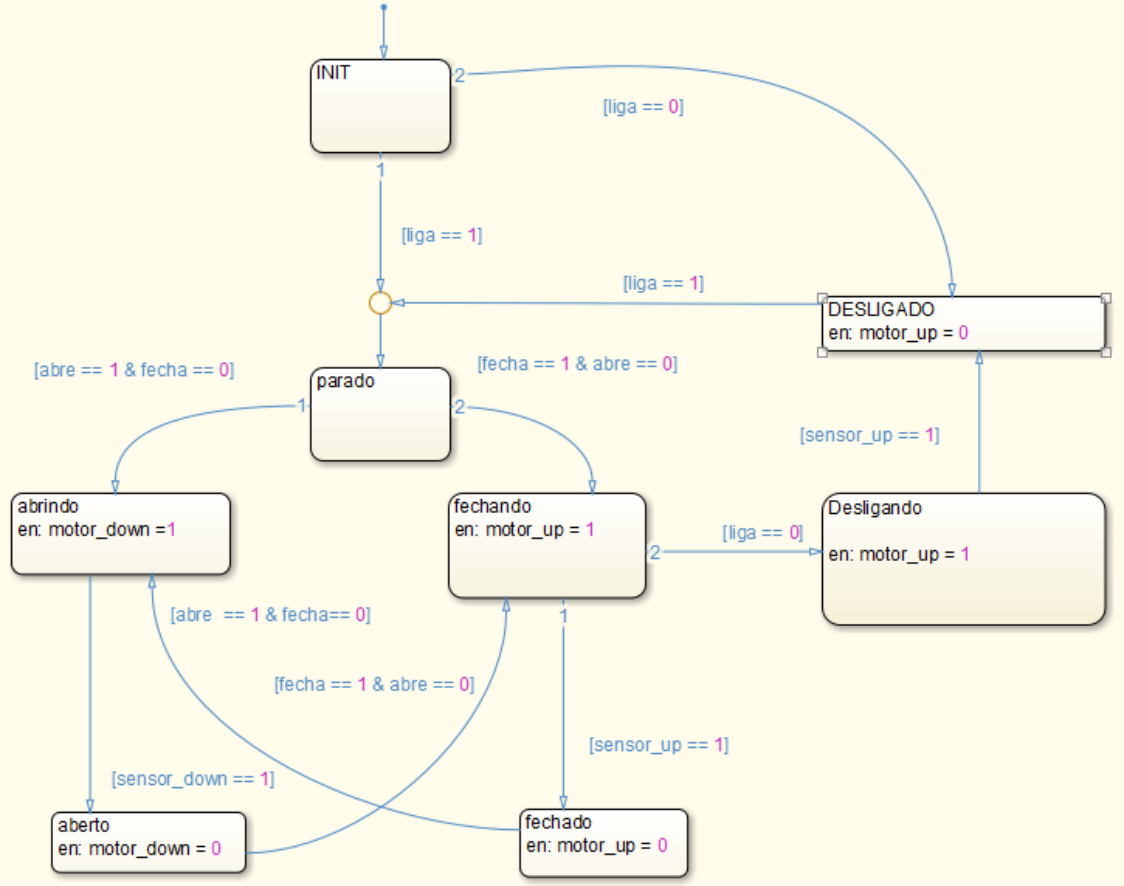
2) Se for pressionado o botão para abertura o vidro (entrada *abre* = 1), a saída para ativação do motor para abrir a janela (saída *motor_down)* permanece ligada até que o sensor (*saída sensor_down)* de fim de curso indique a abertura máxima;

3) Se for pressionado o botão para fechar o vidro (entrada *fecha*= 1), a saída para ativação do motor para fechar a janela (saída *motor_up)* permanece ligada até que o sensor (*saída sensor_up)* de fim de curso indique que a janela está fechada;

4) Se o carro for desligado (entrada *liga == 0)*, o motor para fechamento da janela permanece acionado até a indicação do sensor que indica o fechamento.

Com essas informações, foi obtido o diagrama *Matlab/Stateflow* da Figura 19:

Figura 19 – Diagrama *Stateflow* da função *Janela Elétrica*



Fonte: Autoria Própria

3.4.3 Limpador

O limpador acionado por comando de usuário é uma função presente em todos os veículos. Contudo, este trabalho considera o uso de um sistema automatizado que detecta presença de chuva, e que está disponível apenas em modelos mais sofisticados. Apesar do baixo acréscimo de apenas uma entrada, esta modalidade requer maior poder computacional para leitura de dados e tomada de decisão.

Para acionamento do limpador de vidros foram utilizadas 1 entrada e 1 saída em seu modelo mais simples, e 2 entradas e 1 saída em seu modelo automatizado (Tabela 5).

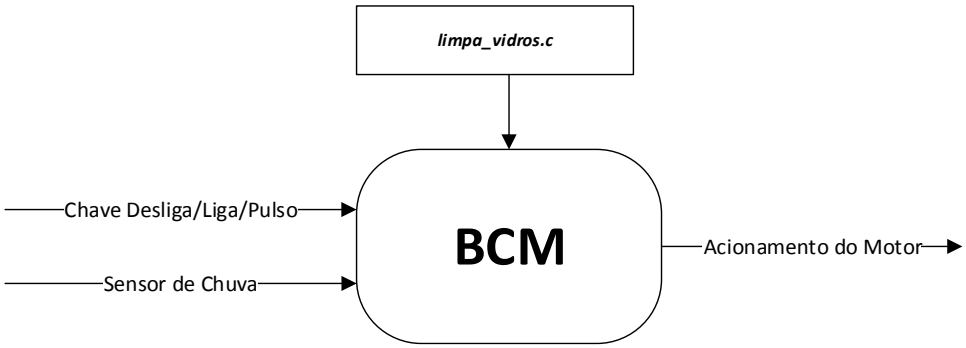
Tabela 5 – Entradas e saídas do sistema de de limpador

Nome	Tipo
In	Entrada Analógica
Sensor	Entrada Digital
Motor	Saída Digital

Fonte : Autoria Própria

O diagrama de blocos do sistema de Limpador é mostrado na Figura 20:

Figura 20: Diagrama de Blocos do sistema de limpador;



Fonte: Autoria Própria

A entrada *in* pode assumir os valores 0 (desligado), 1 (modo pulso) e 2 (modo contínuo), sendo que os modos pulso e contínuo devem acionar a saida *motor;*

A entrada *sensor* pode assumir valores 1, que indica presença de chuva, e 0.

A saída *motor* assume nível 1, para acionar o motor, e 0, para desligá-lo.

A execução da função obedece a seguinte lógica:

1) O carro precisa estar ligado para funcionar;

2) O limpador pode ser acionado de 3 formas:

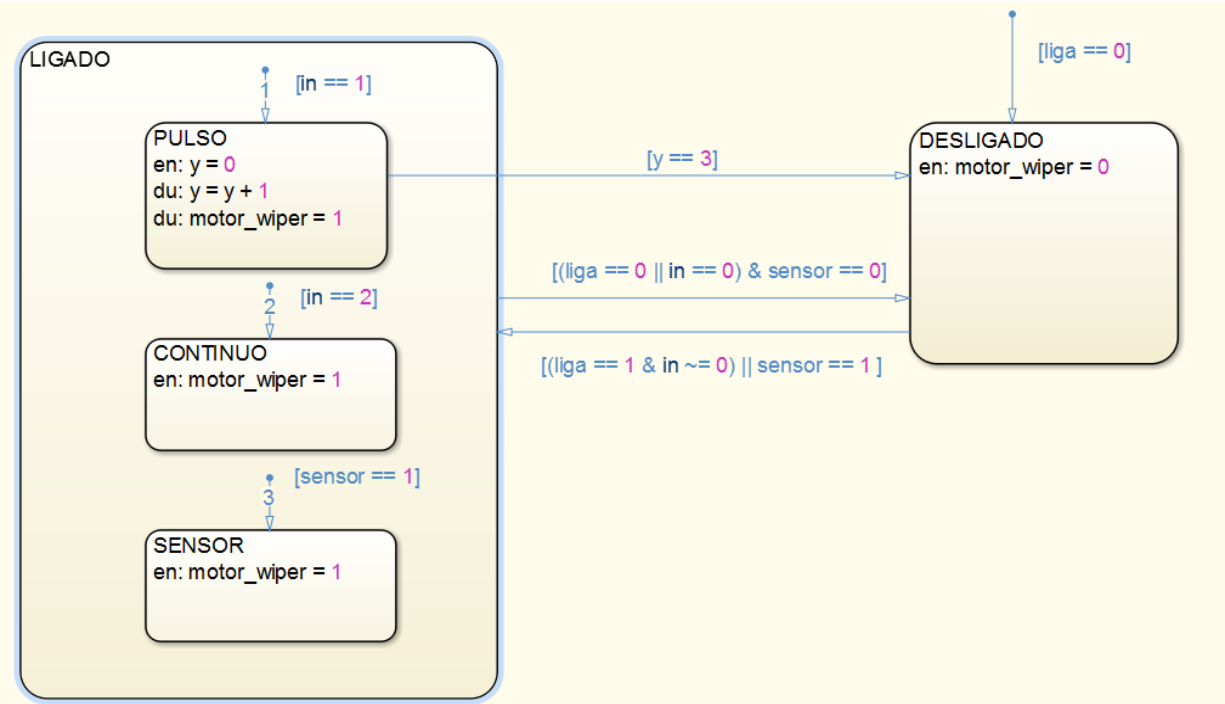
    a. Entrada *in = 1* ativa apenas 1 ciclo do limpador;

    b. Entrada *in = 2* aciona modo de funcionamento contínuo;

c. Entrada *sensor = 1* (sensor de chuva) ativa modo contínuo de forma independente do usuário;

Com essas, foi obtido o diagrama *Matlab/Stateflow* da Figura 21:

Figura 21 – Diagrama *Stateflow* da função *Limpador*



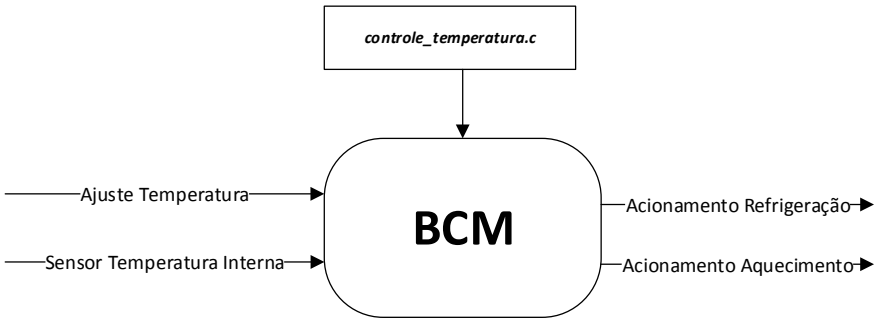
Fonte: Autoria Própria

3.4.4 Climatizador

O climatizador é uma função presente apenas em modelos mais sofisticados. Requer a presença da função de refrigeração de ar, que também não é encontrada em todos os modelos, além de fazer uso de sensores de temperatura espalhados pelo veículo e de necessitar de poder computacional para interpretação de dados e atuação nas saídas.

O diagrama de blocos da função é mostrado na Figura 22:

Figura 22 – Diagrama de Blocos do sistema climatizador.



Fonte: Autoria Própria

Para o funcionamento do climatizador foram utilizadas 2 entradas e 2 saídas: (Tabela 6).

Tabela 6 – Entradas e saídas do sistema de climatizador

Nome	Tipo
T	Entrada Analógica
tref	Entrada Analógica
heater	Saída Digital
Cooler	Saída Digital

Fonte: Autoria Própria

A entrada *T* lê a entrada do sensor que indica a temperatura interna do carro.

A entrada *tref* lê o valor de temperatura desejado declarado na interface do usuário.

A saída *cooler* assume nível 1 para acionar o refrigeramento, quando a temperatura *tref* é menor do que a *T,*do contrário, permanece em nível 0;

A execução da função obedece a seguinte lógica:

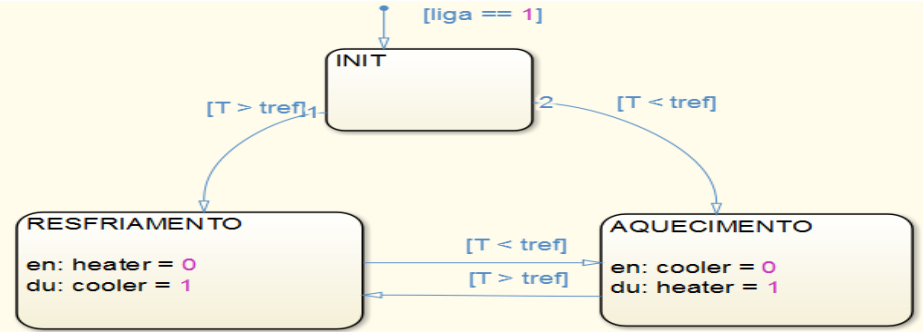
1) O carro precisa estar ligado para o climatizador funcionar;

2) Se a temperatura medida internamente (entrada *T)* for menor do que a temperatura definida como referência pelo usuário (entrada *tref*), o aquecedor (saída *heater*) é acionado;

3) Se a temperatura medida internamente (entrada *T)* for maior do que a temperatura definida como referência pelo usuário (entrada *tref*), a refrigeração (saída *cooler*) é acionado;

Com essas informações, foi obtido o diagrama *Matlab/Stateflow* da Figura 23.

Figura 23 – Diagrama *Stateflow* da função *Climatizador*

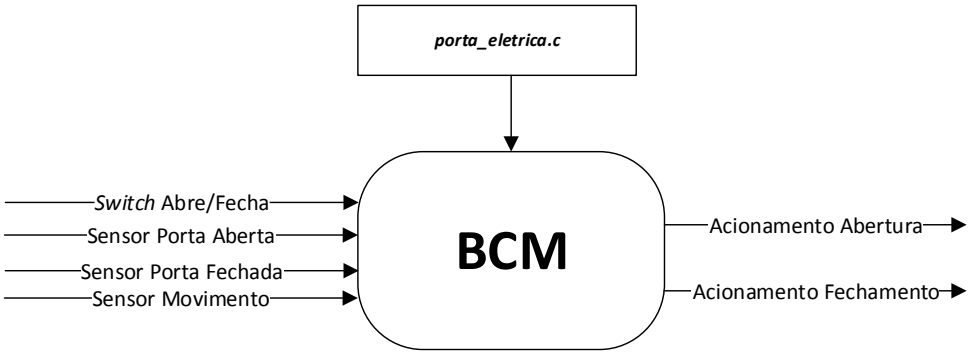


Fonte: Autoria Própria

3.4.5 Porta Elétrica

Esta função está presente em uma gama específica de carros mais espaçosos, considerados de primeira linha. Além da necessidade de motores elétricos para execução da tarefa, precisa de saídas para ativação dos motores e sensores de segurança e para controlar a posição da porta. O diagrama de blocos do sistema de porta elétrica é mostrado na Figura 24 :

Figura 24 – Diagrama de Blocos do sistema de porta elétrica;



Fonte: Autoria Própria

Para acionamento da porta elétrica de uma porta foram usadas 5 entradas e 2 saídas (Tabela 7).

Tabela 7 – Entradas e saídas do sistema de porta motorizada

Nome	Tipo
sensor_closed	Entrada Digital
sensor_open	Entrada Digital
sensor_mov	Entrada Digital
abre	Entrada Digital
fecha	Entrada Digital
motor_close	Saida Digital
motor_open	Saída DIgital

Fonte: Autoria Própria

A entrada *sensor_closed* pode assumir 2 valores: 1 indica que a porta está totalmente fechada, 0 indica que está parcialmente aberta.

A entrada *sensor_open* pode assumir 2 valores: 1 indica abertura total da porta , 0 indica que está parcialmente aberta.

A entrada *abre* pode assumir 2 valores: 1 é o comando para abrir a porta, 0 é o estado ocioso;

A entrada *fecha* pode assumir 2 valores: 1 é o comando para fechar a porta, 0 é o estado ocioso.

A entradas *motor_close* e *motor_*open podem assumir 2 valores: 0, para desligado, e 1 para acionamento.

A execução da função obedece a seguinte lógica:

1) O veículo deve estar ligado e parado;

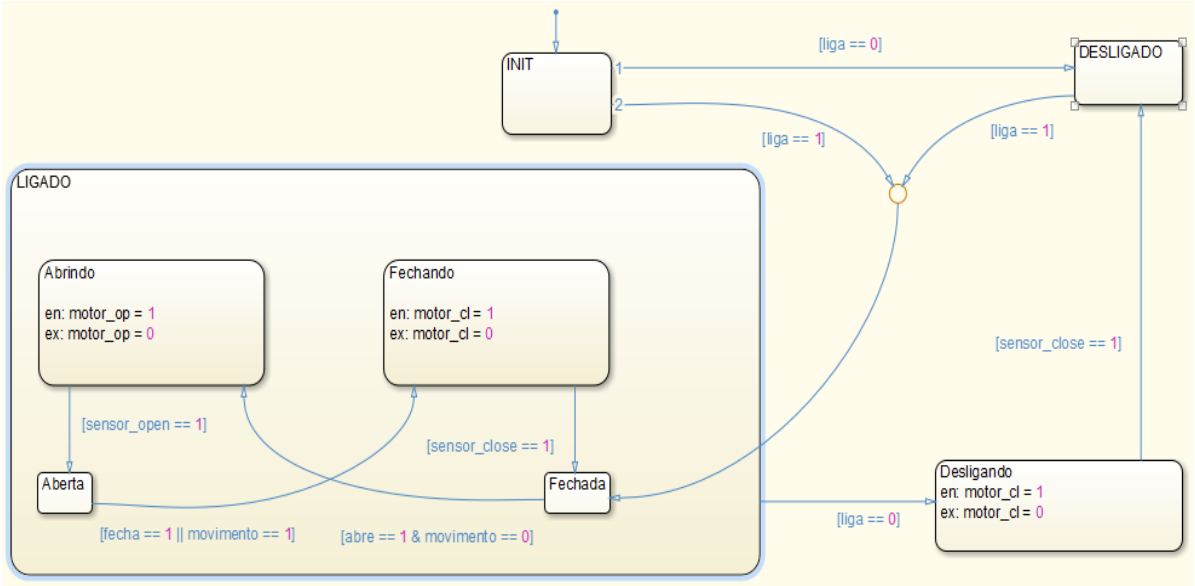
2) Se for pressionado o botão para abrir a porta(entrada *abre* = 1), a saída para ativação do motor de abertura (saída *motor_open)* permanece ligada até que o sensor de fim de curso (entrada *sensor_down)* indique a abertura máxima;

3) Se for pressionado o botão para fechar a porta (entrada *fecha*= 1), a saída para ativação do motor de fechamento (saída *motor_close)* permanece ligada até que o sensor de fim de curso (entrada *sensor_up)* indique que a porta está fechada;

4) Se o carro for desligado (entrada *liga == 0)*, o motor para fechamento da janela é acionado e permanece assim até a indicação do sensor que indica o fechamento.

Com essas informações e as presentes na seção 3.5.4, foi obtido o diagrama *Matlab/Stateflow* da Figura 25:

Figura 25 –Diagrama *Stateflow* da função *Porta Elétrica*



Fonte : Autoria Própria

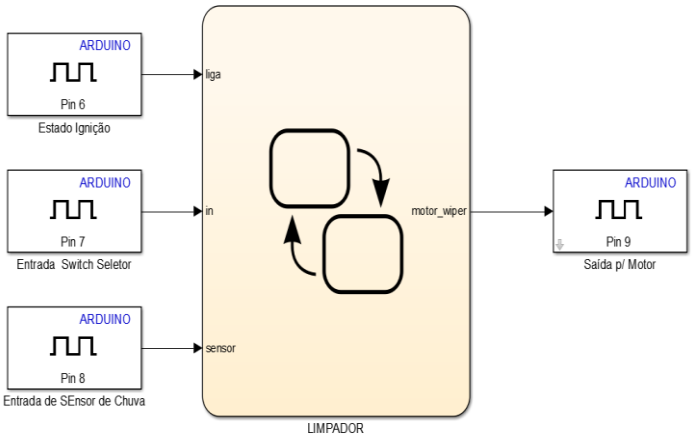
3.5 – *Upload* do código para o Microcontrolador

O pacote de suporte ao *Arduino* disponibilizado pela *Mathworks*® possui blocos funcionais para o ambiente *Simulink*® que possibilitam a leitura e escrita nas portas do microcontrolador. Com a ajuda do pacote, pode-se realizar a comunicação do *Matlab*® com o *Arduino* de duas formas. A primeira delas, mais simples, é baseada em um servidor rodando no microcontrolador que recebe comandos do *Matlab*® através da porta *serial*, e após executar esses comandos, retorna uma resposta ao usuário. Isso permite a visualização dos dados adquiridos e a atuação em portas de saída do Arduino de forma instantânea no *Matlab*®, sem a necessidade de compilação.

A segunda forma consiste em converter o modelo de *Simulink*® em um código que será executado no microcontrolador, permitindo que o mesmo possa ser

desconectado do computador e executar a aplicação de forma independente. A Figura 26 mostra uma das funções desenvolvidas na seção 3.5 com os blocos funcionais de entrada e saídas para implementação em um *Arduino.*

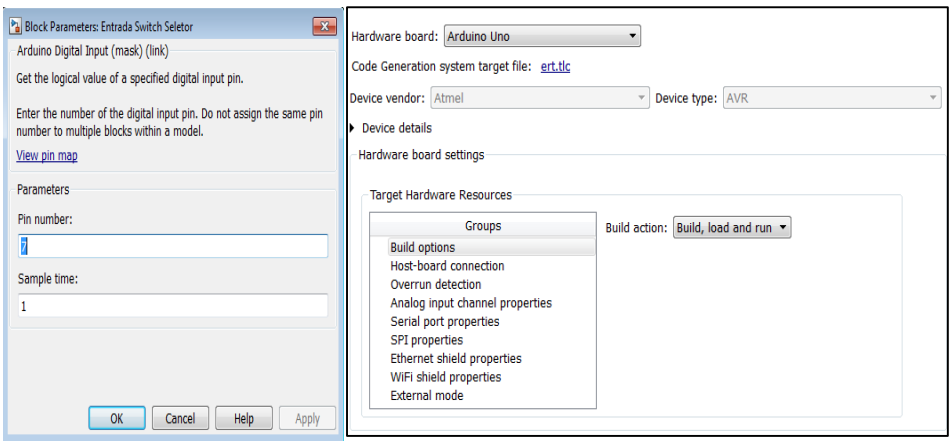
Figura 26 – Bloco funcional com entradas e saídas dedicadas ao uso do *Arduino*



Fonte: Autoria Própria

A configuração dos blocos de entradas e saídas é semelhante, sendo necessário o ajuste de apenas 2 parâmetros: tempo de amostragem (*Sample Time)* o pino do microcontrolador a ser utilizado (*Pin Number).* 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 27).

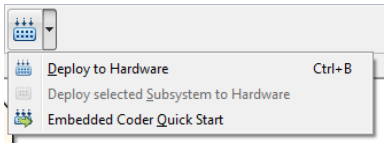
Figura 27: Configuração dos pinos (à esquerda) e configuração do dispositivo (à direita)



Fonte: Autoria Própria

Com o dispositivo configurado, a opção de Implementação em *Hardware* (*Deploy to Hardware)* realiza o upload do código para a plataforma (Figura 28).

Figura 28 – Menu para implementação em *hardware* do modelo



Fonte: Autoria Própria

4 RESULTADOS OBTIDOS

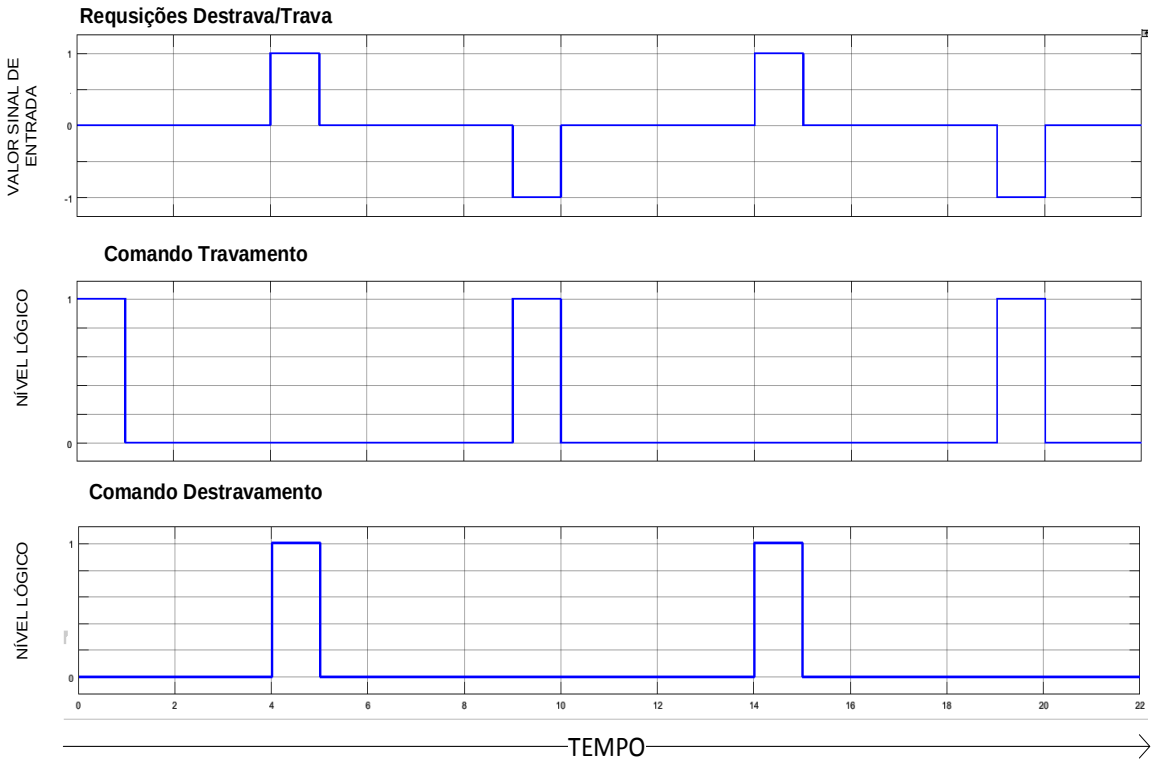
À seguir, são mostrado os resultados da fase de testes *Model in the Loop,* comparando entradas e saídas com o esperado.

4.1 Trava Elétrica

As combinações de entradas/saídas é mostrada no Gráfico 1 e obedece aos critérios mostrados na seção 3.4.1.

Para facilitar a visualização do resultado, as entradas *lock_c* e *unlock_c* foram colocadas no mesmo gráfico. Na relação *Requisições Destrava/Trava* o valor é igual a 1 quando a entrada *unlock_c* assume o valor 1 (requisição para destravamento), e igual a -1 quando a entrada *lock_c* assume valor 1 (requisição para travamento). O valor 0 representa a situação onde não há requisição do usuário. Esta adaptação tem efeitos apenas para a visualização do modelo, não afetando o código da função.

Gráfico 1 – Relação entradas/saídas da função *Trava Elétrica;*



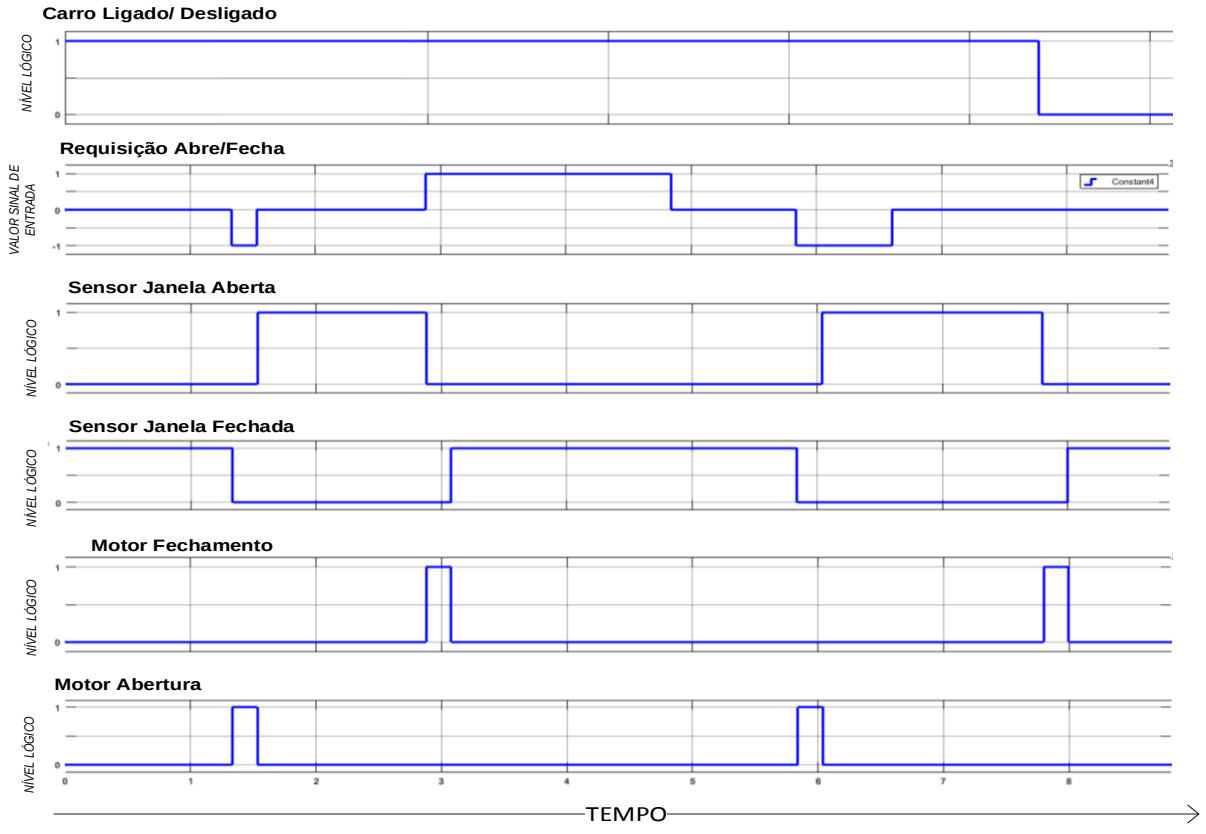
Fonte: Autoria Própria

4.2 Janela Elétrica

No Gráfico 2, é possível verificar a relação de entradas/saídas cumprindo os requisitos de funcionamento: os motores ativados apenas quando sensores fim de curso estão desligados e quando há o comando do usuário.

Para facilitar a visualização do resultado, as entradas *abre* e *fecha* foram colocadas no mesmo gráfico. Na relação *Requisições Abre/Fecha* o valor é igual a 1 quando a entrada *fecha* assume o valor 1 (requisição para fechamento), e igual a -1 quando a entrada *abre* assume valor 1 (requisição para abertura). O valor 0 representa a situação onde não há requisição do usuário. Esta adaptação tem efeitos apenas para a visualização do modelo, não afetando o código da função.

Gráfico 2 - Relação de entradas/saídas da função *Janela Elétrica*

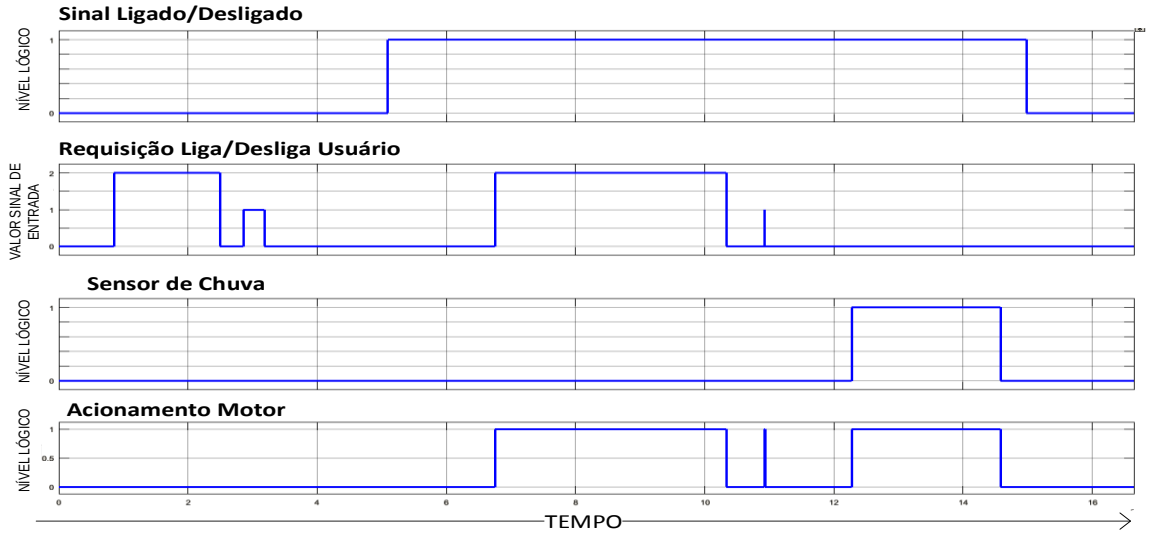


Fonte: Autoria Própria

4.3 Sistema de Limpador

No Gráfico 3, é possível verificar a relação de entradas/saídas cumprIndo os requisitos de funcionamento: o motor é acionado pela ação do usuário e pela ação da chuva. O sistema não funciona com o carro desligado.

Gráfico 3 – Relação de entradas / saídas função limpador



Fonte: Autoria Própria

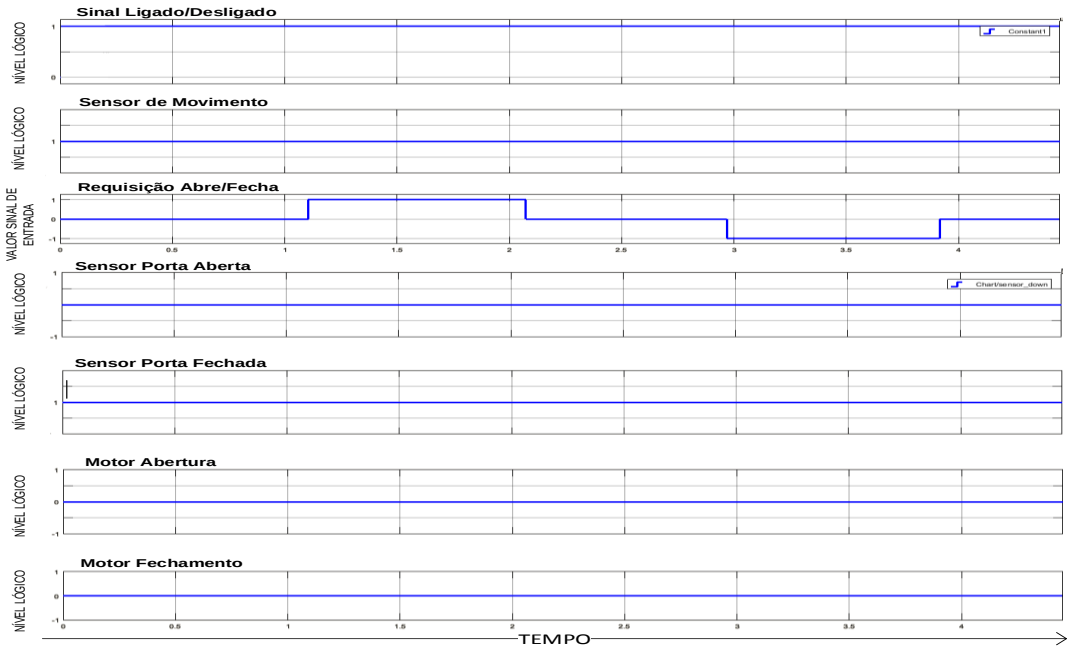
4.4 Porta Elétrica

Para facilitar a visualização do resultado, as entradas *abre* e *fecha* foram colocadas no mesmo gráfico. Na relação *Requisições Abre/Fecha* o valor é igual a 1 quando a entrada *abre* assume o valor 1 (requisição para abertura), é igual a -1 quando a entrada *fecha* assume valor 1 (requisição para fechamento). O valor 0 representa a situação onde não há requisição do usuário. Esta adaptação tem efeitos apenas para a visualização do modelo, não afetando o código da função.

Na Gráfico 4, é possível verificar o funcionamento da função na situação onde o carro está em movimento: não há ativação dos motores quando o sensor indica movimento mesmo com o pedido do usuário, por questões de segurança.

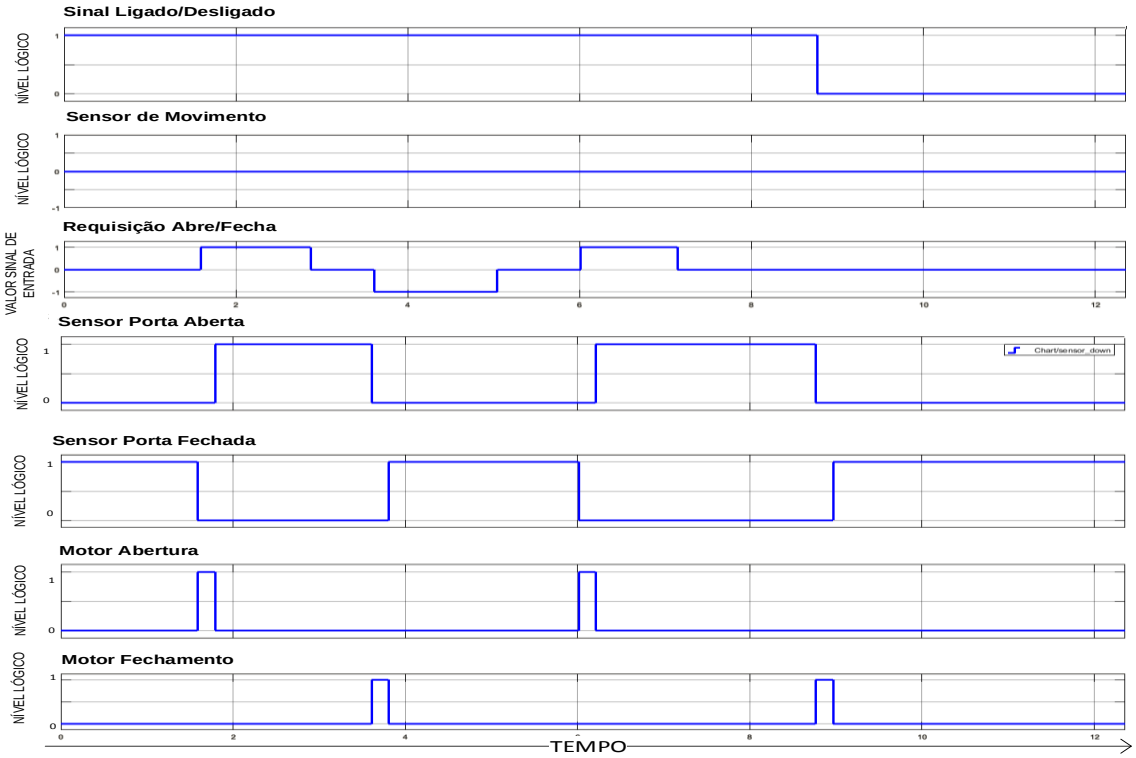
No Gráfico 5, que ilustra a situação onde não há movimento, ou seja, quando não há restrições de segurança, é possível ver a saída de acionamento do motor quando há o pedido do usuário.

Gráfico 4 : Relação de entradas/saídas da função *Porta Elétrica* com o carro em movimento.



Fonte: Autoria Própria

Gráfico 5 : Relação de entradas/saídas da função *Porta Elétrica* com o carro parado.

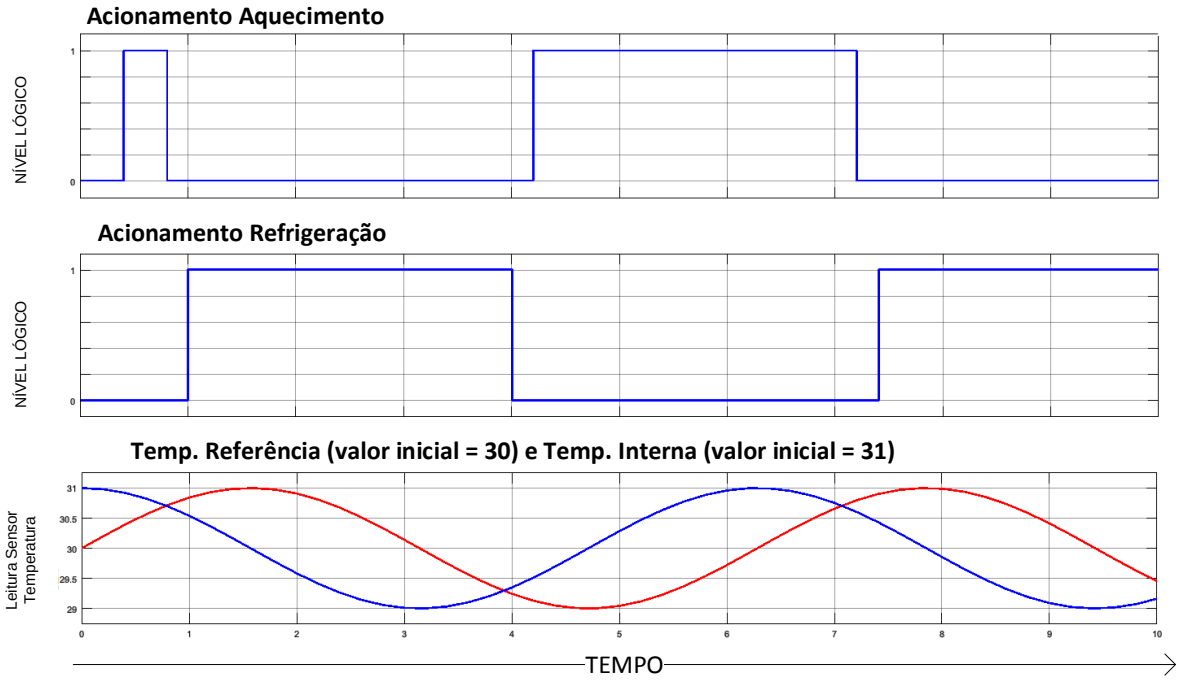


Fonte: Autoria Própria

4.5 Climatizador

No Gráfico 6, é possível verificar a relação de entradas/saídas cumprindo os requisitos de funcionamento: o aquecedor é ligado quando a temperatura de referência (definida pelo usuário) é maior do que o que registra o sensor de temperatura interna do veículo. No caso contrário, há a ativação da refrigeração.

Gráfico 6 – Relação de entradas/saídas da função *Climatizador*



Fonte: Autoria Própria

Com estes resultados, verificou-se o comportamente do modelo definidos para cada função. As saídas respondem às entradas da forma desejada

4.6 – Códigos Gerados

Com a ajuda do pacote de integração da plataforma Arduino com o *Matlab/Simulink*®, os códigos foram verificados no microcontrolador e seu comportamento condiz com os resultados da fase *Model in the Loop*.

Durante a fase de implementação em *hardware* foram gerados 2 arquivos: um com a extensão *elf* e um com a extensão *hex.* O de extensão *hex* é um arquivo compilado para a arquitetura específica do microcontrolador contendo as saídas em linguagem de máquina em formato hexadecimal que serão interpretadas e executadas pelo processador.

O arquivo *elf* contém informações sobre o programa e referências a bibliotecas utilizadas. À partir dele é possível extrair as informações sobre o uso de memória do programa. A Tabela 8 mostra o uso de memórias *Flash* e *SRAM* do microcontrolador de cada função.

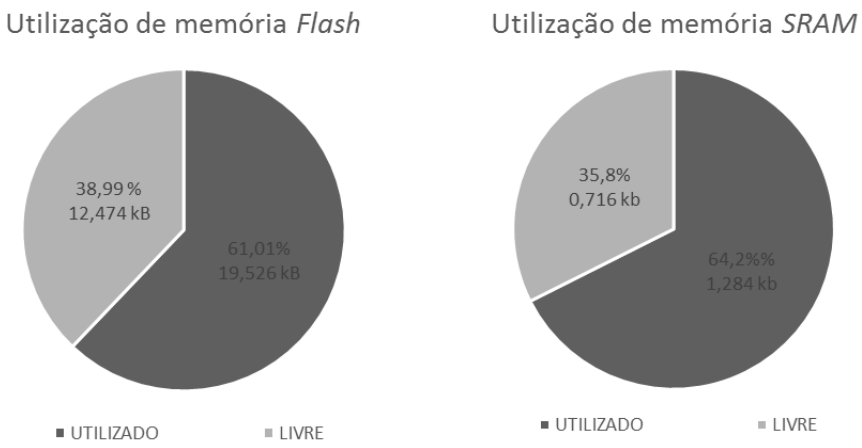
Tabela 8 - Uso de memória das funções

Uso de Memória do Microcontrolador		
Função	***Flash (Bytes)***	***SRAM (Bytes)***
Trava Elétrica	3780	250
Janela Elétrica	3862	262
Limpador	4120	251
Porta Elétrica	3926	267
Climatizador	3838	254
TOTAL	19526	1284

Fonte: Autoria Própria

O Gráfico 7 mostra a parcela dos 32 kB de memória *Flash* e dos 2 kB de memória *SRAM* disponíveis utilizada por cada uma das funções A memóra *flash* é a memória permanente utilizada para gravação do programa de controle e do *bootloader,* que são parâmetros de configuração carregados quando o microcontrolador é ligado. A memória *SRAM* é uma memória volátil que armazena estados das variáveis controladas de cada processo.

Gráfico 7 – Utilização de memórias do Microcontrolador

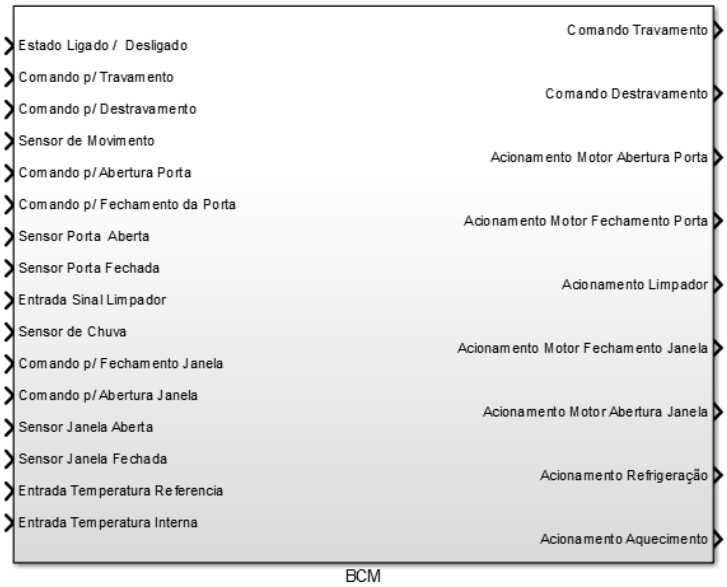


Fonte: Autoria Própria

4.7 – Considerações Finais

É conveniente destacar o alto número de entradas e saídas necessárias para o controle das 5 funções escolhidas (Figura 36).

Figura 29 - Diagrama final de entradas e saídas da BCM proposta



Fonte: Autoria Própria

Para controlar apenas as funções geradas, seriam necessárias 16 entradas e 9 saídas. Seriam necessárias 32 entradas e 23 saídas se o controle de janela (mais 12 entradas e 6 saídas), de travas (mais 6 saídas) e o de porta elétrica (mais 4 entradas e 2 saídas) fossem implementados para todas as posições possíveis.

CONCLUSÃO

Os códigos obtidos à partir da programação gráfica no ambiente *Matlab/Simulink* foram implementados com sucesso no microcontrolador escolhido. Embora não ofereça a robustez desejada para o uso contínuo e a confiabilidade desejadas para implantação em um veículo em condições reais, possibilita a verificação prática dos códigos gerados e permite o avanço do desenvolvimento.

A implementação integrada do ambiente de desenvolvimento *Matlab/Simulink*® com a plataforma *Arduino* realiza a compilação otimizada para a plataforma, proporcionando a possibilidade de uma integração com *hardware* em um tempo curto. Contudo, o código gerado à partir do modelo poderá ser usado em outra arquitetura

O modelo de padronização para criação e reutilização se mostra eficiente, pois elimina esforços repetitivos nas fases iniciais do desenvolvimento e permite maior foco nas fases de testes e validação, possibilitando que todas as rotinas criadas tenham mais confiabilidade e resultando em um sistema mais seguro.

A escolha por trabalhar com funções controladas pela *BCM (Body Control Module)* se mostrou correta, uma vez que as funções nela presentes são mais flexíveis em relação a tempos de resposta e tratam, em sua maioria, de sinais discretos que não necessitam de sensores mais robustos, e dessa forma, de maior custo. Embora muitas funções possam utilizar a mesma entrada, como a de estado do veículo, por exemplo, também é sensível a necessidade mais dispositivos de entrada e saídas conforme se dá a adição de funções à *BCM* , fato que impacta diretamente nos custos do módulo e, assim, no valor do veículo.

5 REFERENCIAS

BROY, Manfred; KIRSTAN, Sascha; KRCMAR, Helmut. **What is the benefit of a model-based design of embedded software systems in the car industry?** 2012. Disponível em: https://wwwbroy.in.tum.de/~schaetz/papers/Benefit_of_MBEES.pdf>. Acesso em: 2 maio 2017

CUNHA, Alessandro F. **O Que São Sistemas Embarcados**. Saber Eletrônica, 2007.

DA SILVA, B.S.S. **Desenvolvimento De Software Embarcado Automotivo Aderente Ao Padrão Autosar.** 2014. Tese (Graduação) – Curso de Engenharia Eletrônica, UnB, Brasília

FARINES, J.M.; FRAGA, J.S ; OLIVEIRA, RS de. **Sistemas De Tempo Real**. Escola de Computação, v. 2000, p. 201, 2000

GANSSLE, J., BARR, M. **Embedded Computer Systems Dictionary.** CMP BOOKS , 2003.

GUIMARÃES, A. A. **Eletrônica Embarcada Automotiva**. 1. ed. [S.l.]: Érica, 2007.

KIENCKE, U., NIELSEN, L. – **Automotive Control Systems – For Engine, Driveline And Vehicle**. 2 ed. Germany, 2005.

LIUGUILONG. **The Design of BCM in Automotive Base on CAN/LIN Bus**. 2013. 132 f. Tese (Mestrado) - Curso de Control Engineering, East China University Of Science And Technology, Xangai, 2013

LI, Q.; YAO, C. Real-time Concepts for Embedded Systems. CRC Press, 2003.

MACÊDO, Raimundo et al. **Tratando A Previsibilidade Em Sistemas De Tempo-Real Distribuídos: Especificação. Linguagens, Middleware E Mecanismos Básicos (Minicurso**). **XXII Simpósio Brasileiro de Redes de Computadores.** Gramado, RS, Brasil, 2004

MARQUES, M. R. S.; SIEGERT, E.; BRISOLARA, L**. Integrating UML, MARTE and SysML to Improve Requirements Specification and Traceability in the Embedded Domain**. 12th IEEE International Conference on Industrial Informatics (INDIN), IEEE, p. 176–181, 2014.

NAVET, N., SIMONOT-LION, F. **Automotive Embedded Systems Handbook**. CRC PRESS, 2009.

NEME, J.H.Z. **Aplicação Do Método De Desenvolvimento Baseado Em Modelos Para Uma Função De Software Automotivo: Sistema De Iluminação Externa**. 2014. 132 f. Tese (Graduação) - Curso de Engenharia Eletrôncia, University, Ponta Grossa.

POLBERGER, D. **Component technology in an embedded system**. Master's thesis in Computer Science. ISSN, 2009.

SANGIOVANNI-VINCENTELLI, A.; NATALE, M. D. **Embedded System Design for Automotive Applications**. Computer, IEEE, v. 40, n. 10, 2007.

SANTOS, M. M. D**.. Redes de Comunicação Automotiva: Características, Tecnologias e Aplicações**. 1. ed. São Paulo: Editora ERICA, 2010. v. 1. 224p

SOUZA, F. Arduíno Uno. Disponível em: <http://www.embarcados.com.br/arduino-uno/> Acesso em 5 de Abril de 2017.

TANENBAUM, A. S. **Sistemas Operacionais Modernos.** 2.ed. São Paulo: Livro Técnico, Prentice Hall, 2003.

ZURAWSKI, R. **Embedded Systems Handbook, 2-Volume Set**. Taylor and Francis Group: CRC Press, Inc, 2009.

Site oficial da AUTOSAR

Site oficial da RENESAS

Site oficial da INFINEON

Site oficioal da ARDUINO

.

ANEXO A – CÓDIGOS GERADOS

No Anexo A, constam os códigos C gerados pelo *Embedded Coder* do *Matlab,* que regem o controle das funções criadas. É digno de nota que por se tratarem de funções complexas, o código de cada uma delas pode incluir outros arquivos de cabeçalhos ou configurações. Neste anexo, constam apenas o código principal da lógica de cada função elaborada.

A.1 – CÓDIGO DO SISTEMA DE TRAVAS

```
1  /*
2  * File: testetravaportas.c
3  *
4  * Code generated for Simulink model 'testetravaportas'.
5  *
6  * Model version : 1.201
7  * Simulink Coder version : 8.10 (R2016a) 10-Feb-2016
8  * C/C++ source code generated on : Tue Jun 06 20:34:12 2017
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 "testetravaportas.h"
17 #include "testetravaportas_private.h"
18
19 /* Named constants for Chart: '<Root>/TRAVAS PORTAS' */
20 #define testetravapo_IN_NO_ACTIVE_CHILD ((uint8_T)0U)
21 #define testetravaportas_IN_IN ((uint8_T)1U)
22 #define testetravaportas_IN_INIT ((uint8_T)2U)
23 #define testetravaportas_IN_LOCKED_I ((uint8_T)1U)
```

```
24 #define testetravaportas_IN_LOCKED_O ((uint8_T)1U)
25 #define testetravaportas_IN_OUT ((uint8_T)3U)
26 #define testetravaportas_IN_UNLOCKED_O ((uint8_T)2U)
27
28 /* Block signals (auto storage) */
29 B_testetravaportas_T testetravaportas_B;
30
31 /* Block states (auto storage) */
32 DW_testetravaportas_T testetravaportas_DW;
33
34 /* Real-time model */
35 RT_MODEL_testetravaportas_T testetravaportas_M_;
36 RT_MODEL_testetravaportas_T *const testetravaportas_M = &testetravaportas_M_;
37
38 /* Model step function */
39 void testetravaportas_step(void)
40 {
41 boolean_T rtb_DigitalInput_0;
42 boolean_T rtb_DigitalInput1_0;
43 uint8_T tmp;
44
45 /* S-Function (arduinodigitalinput_sfcn): '<Root>/Digital Input' */
46 rtb_DigitalInput_0 = MW_digitalRead(testetravaportas_P.DigitalInput_p1);
47
48 /* S-Function (arduinodigitalinput_sfcn): '<Root>/Digital Input1' */
49 rtb_DigitalInput1_0 = MW_digitalRead(testetravaportas_P.DigitalInput1_p1);
50
51 /* Chart: '<Root>/TRAVAS PORTAS' incorporates:
52 * S-Function (arduinodigitalinput_sfcn): '<Root>/Digital Input'
53 * S-Function (arduinodigitalinput_sfcn): '<Root>/Digital Input1'
54 */
55 /* Gateway: TRAVAS PORTAS */
```

```
56  /* During: TRAVAS PORTAS */
57  if (testetravaportas_DW.is_active_c2_testetravaportas == 0U) {
58  /* Entry: TRAVAS PORTAS */
59  testetravaportas_DW.is_active_c2_testetravaportas = 1U;
60
61  /* Entry Internal: TRAVAS PORTAS */
62  /* Transition: '<S2>:104' */
63  testetravaportas_DW.is_c2_testetravaportas = testetravaportas_IN_INIT;
64
65  /* Entry 'INIT': '<S2>:103' */
66  /* '<S2>:103:1' lock_on = 1 */
67  testetravaportas_B.lock_on = 1.0;
68  } else {
69  switch (testetravaportas_DW.is_c2_testetravaportas) {
70  case testetravaportas_IN_IN:
71  /* During 'IN': '<S2>:97' */
72  if (testetravaportas_DW.is_IN == testetravaportas_IN_LOCKED_I) {
73  /* During 'LOCKED_I': '<S2>:98' */
74  /* '<S2>:101:1' sf_internal_predicateOutput = ... */
75  /* '<S2>:101:1' lock_p == -1; */
76  /* '<S2>:105:1' sf_internal_predicateOutput = ... */
77  /* '<S2>:105:1' lock_c == 1; */
78  if (rtb_DigitalInput_0) {
79  /* Transition: '<S2>:105' */
80  testetravaportas_DW.is_IN = testetravapo_IN_NO_ACTIVE_CHILD;
81  testetravaportas_DW.is_c2_testetravaportas = testetravaportas_IN_OUT;
82  testetravaportas_DW.is_OUT = testetravaportas_IN_UNLOCKED_O;
83
84  /* Entry 'UNLOCKED_O': '<S2>:87' */
85  /* '<S2>:87:1' lock_on = 0 */
86  testetravaportas_B.lock_on = 0.0;
87  }
```

```
88  } else {
89  /* During 'UNLOCKED_I': '<S2>:99' */
90  /* '<S2>:100:1' sf_internal_predicateOutput = ... */
91  /* '<S2>:100:1' lock_p == 1; */
92  if (rtb_DigitalInput1_0) {
93  /* Transition: '<S2>:100' */
94  testetravaportas_DW.is_IN = testetravaportas_IN_LOCKED_I;
95
96  /* Entry 'LOCKED_I': '<S2>:98' */
97  /* '<S2>:98:1' lock_on = 1 */
98  testetravaportas_B.lock_on = 1.0;
99  } else {
100 /* '<S2>:102:1' sf_internal_predicateOutput = ... */
101 /* '<S2>:102:1' lock_c == -1; */
102 }
103 }
104 break;
105
106 case testetravaportas_IN_INIT:
107 /* During 'INIT': '<S2>:103' */
108 /* '<S2>:93:1' sf_internal_predicateOutput = ... */
109 /* '<S2>:93:1' lock_c == 1; */
110 if (rtb_DigitalInput_0) {
111 /* Transition: '<S2>:93' */
112 testetravaportas_DW.is_c2_testetravaportas = testetravaportas_IN_OUT;
113 testetravaportas_DW.is_OUT = testetravaportas_IN_UNLOCKED_O;
114
115 /* Entry 'UNLOCKED_O': '<S2>:87' */
116 /* '<S2>:87:1' lock_on = 0 */
117 testetravaportas_B.lock_on = 0.0;
118 } else {
119 /* '<S2>:94:1' sf_internal_predicateOutput = ... */
```

```
120  /* '<S2>:94:1' lock_c == -1; */
121  }
122  break;
123
124  default:
125  /* During 'OUT': '<S2>:69' */
126  if (testetravaportas_DW.is_OUT == testetravaportas_IN_LOCKED_O) {
127  /* During 'LOCKED_O': '<S2>:90' */
128  /* '<S2>:92:1' sf_internal_predicateOutput = ... */
129  /* '<S2>:92:1' lock_c == 1; */
130  if (rtb_DigitalInput_0) {
131  /* Transition: '<S2>:92' */
132  testetravaportas_DW.is_OUT = testetravaportas_IN_UNLOCKED_O;
133
134  /* Entry 'UNLOCKED_O': '<S2>:87' */
135  /* '<S2>:87:1' lock_on = 0 */
136  testetravaportas_B.lock_on = 0.0;
137  }
138  } else {
139  /* During 'UNLOCKED_O': '<S2>:87' */
140  /* '<S2>:95:1' sf_internal_predicateOutput = ... */
141  /* '<S2>:95:1' lock_p == 1; */
142  if (rtb_DigitalInput1_0) {
143  /* Transition: '<S2>:95' */
144  testetravaportas_DW.is_OUT = testetravapo_IN_NO_ACTIVE_CHILD;
145  testetravaportas_DW.is_c2_testetravaportas = testetravaportas_IN_IN;
146  testetravaportas_DW.is_IN = testetravaportas_IN_LOCKED_I;
147
148  /* Entry 'LOCKED_I': '<S2>:98' */
149  /* '<S2>:98:1' lock_on = 1 */
150  testetravaportas_B.lock_on = 1.0;
151  } else {
```

```
152  /* '<S2>:91:1' sf_internal_predicateOutput = ... */
153  /* '<S2>:91:1' lock_c == -1; */
154  }
155  }
156  break;
157  }
158  }
159
160  /* End of Chart: '<Root>/TRAVAS PORTAS' */
161
162  /* DataTypeConversion: '<S1>/Data Type Conversion' */
163  if (testetravaportas_B.lock_on < 256.0) {
164  if (testetravaportas_B.lock_on >= 0.0) {
165  tmp = (uint8_T)testetravaportas_B.lock_on;
166  } else {
167  tmp = 0U;
168  }
169  } else {
170  tmp = MAX_uint8_T;
171  }
172
173  /* End of DataTypeConversion: '<S1>/Data Type Conversion' */
174
175  /* S-Function (arduinodigitaloutput_sfcn): '<S1>/Digital Output' */
176  MW_digitalWrite(testetravaportas_P.DigitalOutput_pinNumber, tmp);
177  }
178
179  /* Model initialize function */
180  void testetravaportas_initialize(void)
181  {
182  /* Registration code */
183
```

```
184  /* initialize error status */
185  rtmSetErrorStatus(testetravaportas_M, (NULL));
186
187  /* block I/O */
188  (void) memset(((void *) &testetravaportas_B), 0,
189  sizeof(B_testetravaportas_T));
190
191  /* states (dwork) */
192  (void) memset((void *)&testetravaportas_DW, 0,
193  sizeof(DW_testetravaportas_T));
194
195  /* Start for S-Function (arduinodigitalinput_sfcn): '<Root>/Digital Input' */
196  MW_pinModeInput(testetravaportas_P.DigitalInput_p1);
197
198  /* Start for S-Function (arduinodigitalinput_sfcn): '<Root>/Digital Input1' */
199  MW_pinModeInput(testetravaportas_P.DigitalInput1_p1);
200
201  /* Start for S-Function (arduinodigitaloutput_sfcn): '<S1>/Digital Output' */
202  MW_pinModeOutput(testetravaportas_P.DigitalOutput_pinNumber);
203
204  /* SystemInitialize for Chart: '<Root>/TRAVAS PORTAS' */
205  testetravaportas_DW.is_IN = testetravapo_IN_NO_ACTIVE_CHILD;
206  testetravaportas_DW.is_OUT = testetravapo_IN_NO_ACTIVE_CHILD;
207  testetravaportas_DW.is_active_c2_testetravaportas = 0U;
208  testetravaportas_DW.is_c2_testetravaportas = testetravapo_IN_NO_ACTIVE_CHILD;
209  }
210
211  /* Model terminate function */
212  void testetravaportas_terminate(void)
213  {
214  /* (no terminate code required) */
215  }
```

```
216
217 /*
218 * File trailer for generated code.
219 *
220 * [EOF]
221 */
```

A.2 – CÓDIGO DO SISTEMA DE JANELA ELÉTRICA

```
1  /*
2  * File: testejanelas.c
3  *
4  * Code generated for Simulink model 'testejanelas'.
5  *
6  * Model version : 1.205
7  * Simulink Coder version : 8.10 (R2016a) 10-Feb-2016
8  * C/C++ source code generated on : Tue Jun 06 20:37:39 2017
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 "testejanelas.h"
17 #include "testejanelas_private.h"
18
19 /* Named constants for Chart: '<Root>/JANELA ELÉTRICA' */
20 #define testejanelas_IN_DESLIGADO ((uint8_T)1U)
21 #define testejanelas_IN_Desligando ((uint8_T)2U)
22 #define testejanelas_IN_INIT ((uint8_T)3U)
23 #define testejanelas_IN_LIGADO ((uint8_T)4U)
24 #define testejanelas_IN_NO_ACTIVE_CHILD ((uint8_T)0U)
25 #define testejanelas_IN_aberto ((uint8_T)1U)
26 #define testejanelas_IN_abrindo ((uint8_T)2U)
27 #define testejanelas_IN_fechado ((uint8_T)3U)
28 #define testejanelas_IN_fechando ((uint8_T)4U)
```

```
29 #define testejanelas_IN_parado ((uint8_T)5U)
30
31 /* Block signals (auto storage) */
32 B_testejanelas_T testejanelas_B;
33
34 /* Block states (auto storage) */
35 DW_testejanelas_T testejanelas_DW;
36
37 /* Real-time model */
38 RT_MODEL_testejanelas_T testejanelas_M_;
39 RT_MODEL_testejanelas_T *const testejanelas_M = &testejanelas_M_;
40
41 /* Model step function */
42 void testejanelas_step(void)
43 {
44 boolean_T rtb_DigitalInput_0;
45 boolean_T rtb_DigitalInput3_0;
46 boolean_T rtb_DigitalInput2_0;
47 boolean_T rtb_DigitalInput1_0;
48 uint8_T tmp;
49
50 /* S-Function (arduinodigitalinput_sfcn): '<Root>/Digital Input' */
51 rtb_DigitalInput_0 = MW_digitalRead(testejanelas_P.DigitalInput_p1);
52
53 /* S-Function (arduinodigitalinput_sfcn): '<Root>/Digital Input3' */
54 rtb_DigitalInput3_0 = MW_digitalRead(testejanelas_P.DigitalInput3_p1);
55
56 /* S-Function (arduinodigitalinput_sfcn): '<Root>/Digital Input2' */
57 rtb_DigitalInput2_0 = MW_digitalRead(testejanelas_P.DigitalInput2_p1);
58
59 /* S-Function (arduinodigitalinput_sfcn): '<Root>/Digital Input1' */
60 rtb_DigitalInput1_0 = MW_digitalRead(testejanelas_P.DigitalInput1_p1);
```

```
61
62  /* Chart: '<Root>/JANELA ELÉTRICA' incorporates:
63  * S-Function (arduinodigitalinput_sfcn): '<Root>/Digital Input'
64  * S-Function (arduinodigitalinput_sfcn): '<Root>/Digital Input1'
65  * S-Function (arduinodigitalinput_sfcn): '<Root>/Digital Input2'
66  * S-Function (arduinodigitalinput_sfcn): '<Root>/Digital Input3'
67  */
68  /* Gateway: JANELA ELÉTRICA */
69  /* During: JANELA ELÉTRICA */
70  if (testejanelas_DW.is_active_c3_testejanelas == 0U) {
71  /* Entry: JANELA ELÉTRICA */
72  testejanelas_DW.is_active_c3_testejanelas = 1U;
73
74  /* Entry Internal: JANELA ELÉTRICA */
75  /* Transition: '<S3>:48' */
76  testejanelas_DW.is_c3_testejanelas = testejanelas_IN_INIT;
77  } else {
78  switch (testejanelas_DW.is_c3_testejanelas) {
79  case testejanelas_IN_DESLIGADO:
80  /* During 'DESLIGADO': '<S3>:70' */
81  /* '<S3>:74:1' sf_internal_predicateOutput = ... */
82  /* '<S3>:74:1' liga == 1; */
83  if (rtb_DigitalInput_0) {
84  /* Transition: '<S3>:74' */
85  /* Transition: '<S3>:85' */
86  testejanelas_DW.is_c3_testejanelas = testejanelas_IN_LIGADO;
87  testejanelas_DW.is_LIGADO = testejanelas_IN_parado;
88  }
89  break;
90
91  case testejanelas_IN_Desligando:
92  /* During 'Desligando': '<S3>:59' */
```

```
93  /* '<S3>:71:1' sf_internal_predicateOutput = ... */
94  /* '<S3>:71:1' sensor_up == 1; */
95  if (rtb_DigitalInput1_0) {
96  /* Transition: '<S3>:71' */
97  testejanelas_DW.is_c3_testejanelas = testejanelas_IN_DESLIGADO;
98
99  /* Entry 'DESLIGADO': '<S3>:70' */
100 /* '<S3>:70:1' motor_up = 0 */
101 testejanelas_B.motor_up = 0.0;
102 }
103 break;
104
105 case testejanelas_IN_INIT:
106 /* During 'INIT': '<S3>:47' */
107 /* '<S3>:82:1' sf_internal_predicateOutput = ... */
108 /* '<S3>:82:1' liga == 1; */
109 if (rtb_DigitalInput_0) {
110 /* Transition: '<S3>:82' */
111 /* Transition: '<S3>:85' */
112 testejanelas_DW.is_c3_testejanelas = testejanelas_IN_LIGADO;
113 testejanelas_DW.is_LIGADO = testejanelas_IN_parado;
114 } else {
115 /* '<S3>:72:1' sf_internal_predicateOutput = ... */
116 /* '<S3>:72:1' liga == 0; */
117 /* Transition: '<S3>:72' */
118 testejanelas_DW.is_c3_testejanelas = testejanelas_IN_DESLIGADO;
119
120 /* Entry 'DESLIGADO': '<S3>:70' */
121 /* '<S3>:70:1' motor_up = 0 */
122 testejanelas_B.motor_up = 0.0;
123 }
124 break;
```

```
125
126  default:
127  /* During 'LIGADO': '<S3>:49' */
128  /* '<S3>:73:1' sf_internal_predicateOutput = ... */
129  /* '<S3>:73:1' liga == 0; */
130  if (!rtb_DigitalInput_0) {
131  /* Transition: '<S3>:73' */
132  /* Exit Internal 'LIGADO': '<S3>:49' */
133  testejanelas_DW.is_LIGADO = testejanelas_IN_NO_ACTIVE_CHILD;
134  testejanelas_DW.is_c3_testejanelas = testejanelas_IN_Desligando;
135
136  /* Entry 'Desligando': '<S3>:59' */
137  /* '<S3>:59:1' motor_up = 1 */
138  testejanelas_B.motor_up = 1.0;
139  } else {
140  switch (testejanelas_DW.is_LIGADO) {
141  case testejanelas_IN_aberto:
142  /* During 'aberto': '<S3>:57' */
143  /* '<S3>:62:1' sf_internal_predicateOutput = ... */
144  /* '<S3>:62:1' up_down == 1; */
145  if (rtb_DigitalInput3_0) {
146  /* Transition: '<S3>:62' */
147  testejanelas_DW.is_LIGADO = testejanelas_IN_fechando;
148
149  /* Entry 'fechando': '<S3>:56' */
150  /* '<S3>:56:1' motor_up = 1 */
151  testejanelas_B.motor_up = 1.0;
152  }
153  break;
154
155  case testejanelas_IN_abrindo:
156  /* During 'abrindo': '<S3>:54' */
```

```
157  /* '<S3>:58:1' sf_internal_predicateOutput = ... */
158  /* '<S3>:58:1' sensor_down == 1; */
159  if (rtb_DigitalInput2_0) {
160  /* Transition: '<S3>:58' */
161  testejanelas_DW.is_LIGADO = testejanelas_IN_aberto;
162
163  /* Entry 'aberto': '<S3>:57' */
164  /* '<S3>:57:1' motor_down = 0 */
165  testejanelas_B.motor_down = 0.0;
166  }
167  break;
168
169  case testejanelas_IN_fechado:
170  /* During 'fechado': '<S3>:60' */
171  /* '<S3>:67:1' sf_internal_predicateOutput = ... */
172  /* '<S3>:67:1' up_down == -1; */
173  break;
174
175  case testejanelas_IN_fechando:
176  /* During 'fechando': '<S3>:56' */
177  /* '<S3>:61:1' sf_internal_predicateOutput = ... */
178  /* '<S3>:61:1' sensor_up == 1; */
179  if (rtb_DigitalInput1_0) {
180  /* Transition: '<S3>:61' */
181  testejanelas_DW.is_LIGADO = testejanelas_IN_fechado;
182
183  /* Entry 'fechado': '<S3>:60' */
184  /* '<S3>:60:1' motor_up = 0 */
185  testejanelas_B.motor_up = 0.0;
186  }
187  break;
188
```

```
189  default:
190  /* During 'parado': '<S3>:50' */
191  /* '<S3>:55:1' sf_internal_predicateOutput = ... */
192  /* '<S3>:55:1' up_down == -1; */
193  /* '<S3>:69:1' sf_internal_predicateOutput = ... */
194  /* '<S3>:69:1' up_down == 1; */
195  if (rtb_DigitalInput3_0) {
196  /* Transition: '<S3>:69' */
197  testejanelas_DW.is_LIGADO = testejanelas_IN_fechando;
198
199  /* Entry 'fechando': '<S3>:56' */
200  /* '<S3>:56:1' motor_up = 1 */
201  testejanelas_B.motor_up = 1.0;
202  }
203  break;
204  }
205  }
206  break;
207  }
208  }
209
210  /* End of Chart: '<Root>/JANELA ELÉTRICA' */
211
212  /* DataTypeConversion: '<S1>/Data Type Conversion' */
213  if (testejanelas_B.motor_up < 256.0) {
214  if (testejanelas_B.motor_up >= 0.0) {
215  tmp = (uint8_T)testejanelas_B.motor_up;
216  } else {
217  tmp = 0U;
218  }
219  } else {
220  tmp = MAX_uint8_T;
```

```
221  }
222
223  /* End of DataTypeConversion: '<S1>/Data Type Conversion' */
224
225  /* S-Function (arduinodigitaloutput_sfcn): '<S1>/Digital Output' */
226  MW_digitalWrite(testejanelas_P.DigitalOutput_pinNumber, tmp);
227
228  /* DataTypeConversion: '<S2>/Data Type Conversion' */
229  if (testejanelas_B.motor_down < 256.0) {
230  if (testejanelas_B.motor_down >= 0.0) {
231  tmp = (uint8_T)testejanelas_B.motor_down;
232  } else {
233  tmp = 0U;
234  }
235  } else {
236  tmp = MAX_uint8_T;
237  }
238
239  /* End of DataTypeConversion: '<S2>/Data Type Conversion' */
240
241  /* S-Function (arduinodigitaloutput_sfcn): '<S2>/Digital Output' */
242  MW_digitalWrite(testejanelas_P.DigitalOutput_pinNumber_g, tmp);
243  }
244
245  /* Model initialize function */
246  void testejanelas_initialize(void)
247  {
248  /* Registration code */
249
250  /* initialize error status */
251  rtmSetErrorStatus(testejanelas_M, (NULL));
252
```

```
253  /* block I/O */
254  (void) memset(((void *) &testejanelas_B), 0,
255  sizeof(B_testejanelas_T));
256
257  /* states (dwork) */
258  (void) memset((void *)&testejanelas_DW, 0,
259  sizeof(DW_testejanelas_T));
260
261  /* Start for S-Function (arduinodigitalinput_sfcn): '<Root>/Digital Input' */
262  MW_pinModeInput(testejanelas_P.DigitalInput_p1);
263
264  /* Start for S-Function (arduinodigitalinput_sfcn): '<Root>/Digital Input3' */
265  MW_pinModeInput(testejanelas_P.DigitalInput3_p1);
266
267  /* Start for S-Function (arduinodigitalinput_sfcn): '<Root>/Digital Input2' */
268  MW_pinModeInput(testejanelas_P.DigitalInput2_p1);
269
270  /* Start for S-Function (arduinodigitalinput_sfcn): '<Root>/Digital Input1' */
271  MW_pinModeInput(testejanelas_P.DigitalInput1_p1);
272
273  /* Start for S-Function (arduinodigitaloutput_sfcn): '<S1>/Digital Output' */
274  MW_pinModeOutput(testejanelas_P.DigitalOutput_pinNumber);
275
276  /* Start for S-Function (arduinodigitaloutput_sfcn): '<S2>/Digital Output' */
277  MW_pinModeOutput(testejanelas_P.DigitalOutput_pinNumber_g);
278
279  /* SystemInitialize for Chart: '<Root>/JANELA ELÉTRICA' */
280  testejanelas_DW.is_LIGADO = testejanelas_IN_NO_ACTIVE_CHILD;
281  testejanelas_DW.is_active_c3_testejanelas = 0U;
282  testejanelas_DW.is_c3_testejanelas = testejanelas_IN_NO_ACTIVE_CHILD;
283  }
284
```

```
285  /* Model terminate function */
286  void testejanelas_terminate(void)
287  {
288  /* (no terminate code required) */
289  }
290
291  /*
292  * File trailer for generated code.
293  *
294  * [EOF]
295  */
296
```

A.3 – CÓDIGO DO SISTEMA DE LIMPADOR

```
1  /*
2   * File: testelimpador.c
3   *
4   * Code generated for Simulink model 'testelimpador'.
5   *
6   * Model version                  : 1.198
7   * Simulink Coder version         : 8.10 (R2016a) 10-Feb-2016
8   * C/C++ source code generated on : Tue Jun 06 20:47:42 2017
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 "testelimpador.h"
17 #include "testelimpador_private.h"
18
19 /* Named constants for Chart: '<Root>/LIMPADOR' */
20 #define testelimpado_IN_NO_ACTIVE_CHILD ((uint8_T)0U)
21 #define testelimpador_IN_CONTINUO      ((uint8_T)1U)
22 #define testelimpador_IN_DESLIGADO     ((uint8_T)1U)
23 #define testelimpador_IN_LIGADO        ((uint8_T)2U)
24 #define testelimpador_IN_PULSO         ((uint8_T)2U)
25 #define testelimpador_IN_SENSOR        ((uint8_T)3U)
26
27 /* Block signals (auto storage) */
28 B_testelimpador_T testelimpador_B;
```

```
29
30 /* Block states (auto storage) */
31 DW_testelimpador_T testelimpador_DW;
32
33 /* Real-time model */
34 RT_MODEL_testelimpador_T testelimpador_M_;
35 RT_MODEL_testelimpador_T *const testelimpador_M = &testelimpador_M_;
36
37 /* Model step function */
38 void testelimpador_step(void)
39 {
40   boolean_T rtb_DigitalInput_0;
41   boolean_T rtb_DigitalInput1_0;
42   boolean_T rtb_DigitalInput2_0;
43   uint8_T tmp;
44
45   /* S-Function (arduinodigitalinput_sfcn): '<Root>/Digital Input' */
46   rtb_DigitalInput_0 =
   MW_digitalRead(testelimpador_P.DigitalInput_p1);
47
48   /* S-Function (arduinodigitalinput_sfcn): '<Root>/Digital Input1' */
49   rtb_DigitalInput1_0 =
   MW_digitalRead(testelimpador_P.DigitalInput1_p1);
50
51   /* S-Function (arduinodigitalinput_sfcn): '<Root>/Digital Input2' */
52   rtb_DigitalInput2_0 =
   MW_digitalRead(testelimpador_P.DigitalInput2_p1);
53
54   /* Chart: '<Root>/LIMPADOR' incorporates:
55    *  S-Function (arduinodigitalinput_sfcn): '<Root>/Digital Input'
56    *  S-Function (arduinodigitalinput_sfcn): '<Root>/Digital Input1'
57    *  S-Function (arduinodigitalinput_sfcn): '<Root>/Digital Input2'
58    */
59   /* Gateway: LIMPADOR */
```

```
60    /* During: LIMPADOR */
61    if (testelimpador_DW.is_active_c3_testelimpador == 0U) {
62      /* Entry: LIMPADOR */
63      testelimpador_DW.is_active_c3_testelimpador = 1U;
64
65      /* Entry Internal: LIMPADOR */
66      /* '<S2>:40:1' sf_internal_predicateOutput = ... */
67      /* '<S2>:40:1' liga == 0; */
68      if (!rtb_DigitalInput_0) {
69        /* Transition: '<S2>:40' */
70        testelimpador_DW.is_c3_testelimpador =
  testelimpador_IN_DESLIGADO;
71
72        /* Entry 'DESLIGADO': '<S2>:25' */
73        /* '<S2>:25:1' motor_wiper = 0 */
74        testelimpador_B.motor_wiper = 0.0;
75      }
76    } else if (testelimpador_DW.is_c3_testelimpador ==
  testelimpador_IN_DESLIGADO)
77    {
78      /* During 'DESLIGADO': '<S2>:25' */
79      /* '<S2>:39:1' sf_internal_predicateOutput = ... */
80      /* '<S2>:39:1' liga == 1 & ( in ~= 0 || sensor == 1); */
81      if (rtb_DigitalInput_0 && (rtb_DigitalInput1_0 ||
  rtb_DigitalInput2_0)) {
82        /* Transition: '<S2>:39' */
83        testelimpador_DW.is_c3_testelimpador = testelimpador_IN_LIGADO;
84
85        /* Entry Internal 'LIGADO': '<S2>:28' */
86        /* '<S2>:41:1' sf_internal_predicateOutput = ... */
87        /* '<S2>:41:1' in == 1; */
88        if (rtb_DigitalInput1_0) {
89          /* Transition: '<S2>:41' */
90          testelimpador_DW.is_LIGADO = testelimpador_IN_PULSO;
```

```
91
92          /* Entry 'PULSO': '<S2>:27' */
93          /* '<S2>:27:1' y = 0 */
94          testelimpador_DW.y = 0.0;
95        } else {
96          /* '<S2>:42:1' sf_internal_predicateOutput = ... */
97          /* '<S2>:42:1' in == 2; */
98          /* '<S2>:51:1' sf_internal_predicateOutput = ... */
99          /* '<S2>:51:1' sensor == 1; */
100         if (rtb_DigitalInput2_0) {
101           /* Transition: '<S2>:51' */
102           testelimpador_DW.is_LIGADO = testelimpador_IN_SENSOR;
103
104           /* Entry 'SENSOR': '<S2>:50' */
105           /* '<S2>:50:1' motor_wiper = 1 */
106           testelimpador_B.motor_wiper = 1.0;
107         }
108       }
109     }
110   } else {
111     /* During 'LIGADO': '<S2>:28' */
112     /* '<S2>:26:1' sf_internal_predicateOutput = ... */
113     /* '<S2>:26:1' liga == 0 & ( in == 0 || sensor == 0); */
114     if ((!rtb_DigitalInput_0) && ((!rtb_DigitalInput1_0) ||
115          (!rtb_DigitalInput2_0))) {
116       /* Transition: '<S2>:26' */
117       /* Exit Internal 'LIGADO': '<S2>:28' */
118       testelimpador_DW.is_LIGADO = testelimpado_IN_NO_ACTIVE_CHILD;
119       testelimpador_DW.is_c3_testelimpador =
    testelimpador_IN_DESLIGADO;
120
121       /* Entry 'DESLIGADO': '<S2>:25' */
122       /* '<S2>:25:1' motor_wiper = 0 */
```

```
123       testelimpador_B.motor_wiper = 0.0;
124     } else {
125       switch (testelimpador_DW.is_LIGADO) {
126        case testelimpador_IN_CONTINUO:
127         /* During 'CONTINUO': '<S2>:35' */
128         break;
129
130        case testelimpador_IN_PULSO:
131         /* During 'PULSO': '<S2>:27' */
132         /* '<S2>:45:1' sf_internal_predicateOutput = ... */
133         /* '<S2>:45:1' y == 3; */
134         if (testelimpador_DW.y == 3.0) {
135           /* Transition: '<S2>:45' */
136           testelimpador_DW.is_LIGADO =
    testelimpado_IN_NO_ACTIVE_CHILD;
137           testelimpador_DW.is_c3_testelimpador =
    testelimpador_IN_DESLIGADO;
138
139           /* Entry 'DESLIGADO': '<S2>:25' */
140           /* '<S2>:25:1' motor_wiper = 0 */
141           testelimpador_B.motor_wiper = 0.0;
142         } else {
143           /* '<S2>:27:1' y = y + 1 */
144           testelimpador_DW.y++;
145
146           /* '<S2>:27:1' motor_wiper = 1 */
147           testelimpador_B.motor_wiper = 1.0;
148         }
149         break;
150
151        default:
152         /* During 'SENSOR': '<S2>:50' */
153         break;
154       }
```

```
155       }
156     }
157
158     /* End of Chart: '<Root>/LIMPADOR' */
159
160     /* DataTypeConversion: '<S1>/Data Type Conversion' */
161     if (testelimpador_B.motor_wiper < 256.0) {
162       if (testelimpador_B.motor_wiper >= 0.0) {
163         tmp = (uint8_T)testelimpador_B.motor_wiper;
164       } else {
165         tmp = 0U;
166       }
167     } else {
168       tmp = MAX_uint8_T;
169     }
170
171     /* End of DataTypeConversion: '<S1>/Data Type Conversion' */
172
173     /* S-Function (arduinodigitaloutput_sfcn): '<S1>/Digital Output' */
174     MW_digitalWrite(testelimpador_P.DigitalOutput_pinNumber, tmp);
175   }
176
177   /* Model initialize function */
178   void testelimpador_initialize(void)
179   {
180     /* Registration code */
181
182     /* initialize error status */
183     rtmSetErrorStatus(testelimpador_M, (NULL));
184
185     /* block I/O */
186     (void) memset(((void *) &testelimpador_B), 0,
187                   sizeof(B_testelimpador_T));
```

```
188
189   /* states (dwork) */
190   (void) memset((void *)&testelimpador_DW, 0,
191                 sizeof(DW_testelimpador_T));
192
193   /* Start for S-Function (arduinodigitalinput_sfcn): '<Root>/Digital
      Input' */
194   MW_pinModeInput(testelimpador_P.DigitalInput_p1);
195
196   /* Start for S-Function (arduinodigitalinput_sfcn): '<Root>/Digital
      Input1' */
197   MW_pinModeInput(testelimpador_P.DigitalInput1_p1);
198
199   /* Start for S-Function (arduinodigitalinput_sfcn): '<Root>/Digital
      Input2' */
200   MW_pinModeInput(testelimpador_P.DigitalInput2_p1);
201
202   /* Start for S-Function (arduinodigitaloutput_sfcn): '<S1>/Digital
      Output' */
203   MW_pinModeOutput(testelimpador_P.DigitalOutput_pinNumber);
204
205   /* SystemInitialize for Chart: '<Root>/LIMPADOR' */
206   testelimpador_DW.is_LIGADO = testelimpado_IN_NO_ACTIVE_CHILD;
207   testelimpador_DW.is_active_c3_testelimpador = 0U;
208   testelimpador_DW.is_c3_testelimpador =
      testelimpado_IN_NO_ACTIVE_CHILD;
209 }
210
211 /* Model terminate function */
212 void testelimpador_terminate(void)
213 {
214   /* (no terminate code required) */
215 }
216
217 /*
```

```
218   * File trailer for generated code.
219   *
220   * [EOF]
221   */
```

A.4 – CÓDIGO DO SISTEMA DE PORTAS

```
1  /*
2   * File: testeportaletrica.c
3   *
4   * Code generated for Simulink model 'testeportaletrica'.
5   *
6   * Model version                  : 1.204
7   * Simulink Coder version         : 8.10 (R2016a) 10-Feb-2016
8   * C/C++ source code generated on : Tue Jun 06 21:09:02 2017
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 "testeportaletrica.h"
17 #include "testeportaletrica_private.h"
18
19 /* Named constants for Chart: '<Root>/PORTA ELÉTRICA' */
20 #define testeportale_IN_NO_ACTIVE_CHILD ((uint8_T)0U)
21 #define testeportaletrica_IN_Aberta     ((uint8_T)1U)
22 #define testeportaletrica_IN_Abrindo    ((uint8_T)2U)
23 #define testeportaletrica_IN_DESLIGADO  ((uint8_T)1U)
24 #define testeportaletrica_IN_Desligando ((uint8_T)2U)
25 #define testeportaletrica_IN_Fechada    ((uint8_T)3U)
26 #define testeportaletrica_IN_Fechando   ((uint8_T)4U)
27 #define testeportaletrica_IN_INIT       ((uint8_T)3U)
28 #define testeportaletrica_IN_LIGADO     ((uint8_T)4U)
29
30 /* Block signals (auto storage) */
```

```
31 B_testeportaletrica_T testeportaletrica_B;
32
33 /* Block states (auto storage) */
34 DW_testeportaletrica_T testeportaletrica_DW;
35
36 /* Real-time model */
37 RT_MODEL_testeportaletrica_T testeportaletrica_M_;
38 RT_MODEL_testeportaletrica_T *const testeportaletrica_M =
   &testeportaletrica_M_;
39
40 /* Model step function */
41 void testeportaletrica_step(void)
42 {
43   boolean_T rtb_DigitalInput_0;
44   boolean_T rtb_DigitalInput1_0;
45   boolean_T rtb_DigitalInput2_0;
46   boolean_T rtb_DigitalInput3_0;
47   boolean_T rtb_DigitalInput4_0;
48   uint8_T tmp;
49
50   /* S-Function (arduinodigitalinput_sfcn): '<Root>/Digital Input' */
51   rtb_DigitalInput_0 =
   MW_digitalRead(testeportaletrica_P.DigitalInput_p1);
52
53   /* S-Function (arduinodigitalinput_sfcn): '<Root>/Digital Input1' */
54   rtb_DigitalInput1_0 =
   MW_digitalRead(testeportaletrica_P.DigitalInput1_p1);
55
56   /* S-Function (arduinodigitalinput_sfcn): '<Root>/Digital Input2' */
57   rtb_DigitalInput2_0 =
   MW_digitalRead(testeportaletrica_P.DigitalInput2_p1);
58
59   /* S-Function (arduinodigitalinput_sfcn): '<Root>/Digital Input3' */
60   rtb_DigitalInput3_0 =
   MW_digitalRead(testeportaletrica_P.DigitalInput3_p1);
```

```
61
62    /* S-Function (arduinodigitalinput_sfcn): '<Root>/Digital Input4' */
63    rtb_DigitalInput4_0 =
MW_digitalRead(testeportaletrica_P.DigitalInput4_p1);
64
65    /* Chart: '<Root>/PORTA ELÉTRICA' incorporates:
66     *  S-Function (arduinodigitalinput_sfcn): '<Root>/Digital Input'
67     *  S-Function (arduinodigitalinput_sfcn): '<Root>/Digital Input1'
68     *  S-Function (arduinodigitalinput_sfcn): '<Root>/Digital Input2'
69     *  S-Function (arduinodigitalinput_sfcn): '<Root>/Digital Input3'
70     *  S-Function (arduinodigitalinput_sfcn): '<Root>/Digital Input4'
71     */
72    /* Gateway: PORTA ELÉTRICA */
73    /* During: PORTA ELÉTRICA */
74    if (testeportaletrica_DW.is_active_c1_testeportaletrica == 0U) {
75      /* Entry: PORTA ELÉTRICA */
76      testeportaletrica_DW.is_active_c1_testeportaletrica = 1U;
77
78      /* Entry Internal: PORTA ELÉTRICA */
79      /* Transition: '<S3>:48' */
80      testeportaletrica_DW.is_c1_testeportaletrica =
testeportaletrica_IN_INIT;
81    } else {
82      switch (testeportaletrica_DW.is_c1_testeportaletrica) {
83       case testeportaletrica_IN_DESLIGADO:
84        /* During 'DESLIGADO': '<S3>:13' */
85        /* '<S3>:47:1' sf_internal_predicateOutput = ... */
86        /* '<S3>:47:1' liga == 1; */
87        if (rtb_DigitalInput_0) {
88          /* Transition: '<S3>:47' */
89          /* Transition: '<S3>:27' */
90          testeportaletrica_DW.is_c1_testeportaletrica =
91            testeportaletrica_IN_LIGADO;
92          testeportaletrica_DW.is_LIGADO = testeportaletrica_IN_Fechada;
```

```
93          }
94          break;
95
96        case testeportaletrica_IN_Desligando:
97          /* During 'Desligando': '<S3>:14' */
98          /* '<S3>:29:1' sf_internal_predicateOutput = ... */
99          /* '<S3>:29:1' sensor_close == 1; */
100         if (rtb_DigitalInput4_0) {
101           /* Transition: '<S3>:29' */
102           /* Exit 'Desligando': '<S3>:14' */
103           /* '<S3>:14:1' motor_cl = 0 */
104           testeportaletrica_B.motor_cl = 0.0;
105           testeportaletrica_DW.is_c1_testeportaletrica =
106             testeportaletrica_IN_DESLIGADO;
107         }
108         break;
109
110       case testeportaletrica_IN_INIT:
111         /* During 'INIT': '<S3>:37' */
112         /* '<S3>:39:1' sf_internal_predicateOutput = ... */
113         /* '<S3>:39:1' liga == 0; */
114         if (!rtb_DigitalInput_0) {
115           /* Transition: '<S3>:39' */
116           testeportaletrica_DW.is_c1_testeportaletrica =
117             testeportaletrica_IN_DESLIGADO;
118         } else {
119           /* '<S3>:46:1' sf_internal_predicateOutput = ... */
120           /* '<S3>:46:1' liga == 1; */
121           /* Transition: '<S3>:46' */
122           /* Transition: '<S3>:27' */
123           testeportaletrica_DW.is_c1_testeportaletrica =
124             testeportaletrica_IN_LIGADO;
125           testeportaletrica_DW.is_LIGADO = testeportaletrica_IN_Fechada;
```

```
126       }
127       break;
128
129      default:
130       /* During 'LIGADO': '<S3>:12' */
131       /* '<S3>:28:1' sf_internal_predicateOutput = ... */
132       /* '<S3>:28:1' liga == 0; */
133       if (!rtb_DigitalInput_0) {
134         /* Transition: '<S3>:28' */
135         /* Exit Internal 'LIGADO': '<S3>:12' */
136         switch (testeportaletrica_DW.is_LIGADO) {
137          case testeportaletrica_IN_Abrindo:
138           /* Exit 'Abrindo': '<S3>:15' */
139           /* '<S3>:15:1' motor_op = 0 */
140           testeportaletrica_B.motor_op = 0.0;
141           testeportaletrica_DW.is_LIGADO =
    testeportale_IN_NO_ACTIVE_CHILD;
142           break;
143
144          case testeportaletrica_IN_Fechando:
145           /* Exit 'Fechando': '<S3>:19' */
146           /* '<S3>:19:1' motor_cl = 0 */
147           testeportaletrica_B.motor_cl = 0.0;
148           testeportaletrica_DW.is_LIGADO =
    testeportale_IN_NO_ACTIVE_CHILD;
149           break;
150
151          default:
152           testeportaletrica_DW.is_LIGADO =
    testeportale_IN_NO_ACTIVE_CHILD;
153           break;
154         }
155
156         testeportaletrica_DW.is_c1_testeportaletrica =
```

```
157           testeportaletrica_IN_Desligando;
158
159         /* Entry 'Desligando': '<S3>:14' */
160         /* '<S3>:14:1' motor_cl = 1 */
161         testeportaletrica_B.motor_cl = 1.0;
162       } else {
163         switch (testeportaletrica_DW.is_LIGADO) {
164          case testeportaletrica_IN_Aberta:
165           /* During 'Aberta': '<S3>:16' */
166           /* '<S3>:42:1' sf_internal_predicateOutput = ... */
167           /* '<S3>:42:1' opn_cls == -1 || movimento == 1; */
168           if (rtb_DigitalInput1_0) {
169             /* Transition: '<S3>:42' */
170             testeportaletrica_DW.is_LIGADO =
    testeportaletrica_IN_Fechando;
171
172             /* Entry 'Fechando': '<S3>:19' */
173             /* '<S3>:19:1' motor_cl = 1 */
174             testeportaletrica_B.motor_cl = 1.0;
175           }
176           break;
177
178          case testeportaletrica_IN_Abrindo:
179           /* During 'Abrindo': '<S3>:15' */
180           /* '<S3>:30:1' sf_internal_predicateOutput = ... */
181           /* '<S3>:30:1' sensor_open == 1; */
182           if (rtb_DigitalInput3_0) {
183             /* Transition: '<S3>:30' */
184             /* Exit 'Abrindo': '<S3>:15' */
185             /* '<S3>:15:1' motor_op = 0 */
186             testeportaletrica_B.motor_op = 0.0;
187             testeportaletrica_DW.is_LIGADO =
    testeportaletrica_IN_Aberta;
188           }
```

```
189           break;
190
191          case testeportaletrica_IN_Fechada:
192           /* During 'Fechada': '<S3>:18' */
193           /* '<S3>:43:1' sf_internal_predicateOutput = ... */
194           /* '<S3>:43:1' opn_cls == 1 & movimento == 0; */
195           if (rtb_DigitalInput2_0 && (!rtb_DigitalInput1_0)) {
196             /* Transition: '<S3>:43' */
197             testeportaletrica_DW.is_LIGADO =
    testeportaletrica_IN_Abrindo;
198
199             /* Entry 'Abrindo': '<S3>:15' */
200             /* '<S3>:15:1' motor_op = 1 */
201             testeportaletrica_B.motor_op = 1.0;
202           }
203           break;
204
205          default:
206           /* During 'Fechando': '<S3>:19' */
207           /* '<S3>:33:1' sf_internal_predicateOutput = ... */
208           /* '<S3>:33:1' sensor_close == 1; */
209           if (rtb_DigitalInput4_0) {
210             /* Transition: '<S3>:33' */
211             /* Exit 'Fechando': '<S3>:19' */
212             /* '<S3>:19:1' motor_cl = 0 */
213             testeportaletrica_B.motor_cl = 0.0;
214             testeportaletrica_DW.is_LIGADO =
    testeportaletrica_IN_Fechada;
215           }
216           break;
217         }
218       }
219       break;
220     }
```

```
221   }
222
223   /* End of Chart: '<Root>/PORTA ELÉTRICA' */
224
225   /* DataTypeConversion: '<S1>/Data Type Conversion' */
226   if (testeportaletrica_B.motor_op < 256.0) {
227     if (testeportaletrica_B.motor_op >= 0.0) {
228       tmp = (uint8_T)testeportaletrica_B.motor_op;
229     } else {
230       tmp = 0U;
231     }
232   } else {
233     tmp = MAX_uint8_T;
234   }
235
236   /* End of DataTypeConversion: '<S1>/Data Type Conversion' */
237
238   /* S-Function (arduinodigitaloutput_sfcn): '<S1>/Digital Output' */
239   MW_digitalWrite(testeportaletrica_P.DigitalOutput_pinNumber, tmp);
240
241   /* DataTypeConversion: '<S2>/Data Type Conversion' */
242   if (testeportaletrica_B.motor_cl < 256.0) {
243     if (testeportaletrica_B.motor_cl >= 0.0) {
244       tmp = (uint8_T)testeportaletrica_B.motor_cl;
245     } else {
246       tmp = 0U;
247     }
248   } else {
249     tmp = MAX_uint8_T;
250   }
251
252   /* End of DataTypeConversion: '<S2>/Data Type Conversion' */
253
```

```
254   /* S-Function (arduinodigitaloutput_sfcn): '<S2>/Digital Output' */
255   MW_digitalWrite(testeportaletrica_P.DigitalOutput_pinNumber_o, tmp);
256 }
257
258 /* Model initialize function */
259 void testeportaletrica_initialize(void)
260 {
261   /* Registration code */
262
263   /* initialize error status */
264   rtmSetErrorStatus(testeportaletrica_M, (NULL));
265
266   /* block I/O */
267   (void) memset(((void *) &testeportaletrica_B), 0,
268                 sizeof(B_testeportaletrica_T));
269
270   /* states (dwork) */
271   (void) memset((void *)&testeportaletrica_DW, 0,
272                 sizeof(DW_testeportaletrica_T));
273
274   /* Start for S-Function (arduinodigitalinput_sfcn): '<Root>/Digital
      Input' */
275   MW_pinModeInput(testeportaletrica_P.DigitalInput_p1);
276
277   /* Start for S-Function (arduinodigitalinput_sfcn): '<Root>/Digital
      Input1' */
278   MW_pinModeInput(testeportaletrica_P.DigitalInput1_p1);
279
280   /* Start for S-Function (arduinodigitalinput_sfcn): '<Root>/Digital
      Input2' */
281   MW_pinModeInput(testeportaletrica_P.DigitalInput2_p1);
282
283   /* Start for S-Function (arduinodigitalinput_sfcn): '<Root>/Digital
      Input3' */
284   MW_pinModeInput(testeportaletrica_P.DigitalInput3_p1);
```

```
285
286   /* Start for S-Function (arduinodigitalinput_sfcn): '<Root>/Digital
      Input4' */
287   MW_pinModeInput(testeportaletrica_P.DigitalInput4_p1);
288
289   /* Start for S-Function (arduinodigitaloutput_sfcn): '<S1>/Digital
      Output' */
290   MW_pinModeOutput(testeportaletrica_P.DigitalOutput_pinNumber);
291
292   /* Start for S-Function (arduinodigitaloutput_sfcn): '<S2>/Digital
      Output' */
293   MW_pinModeOutput(testeportaletrica_P.DigitalOutput_pinNumber_o);
294
295   /* SystemInitialize for Chart: '<Root>/PORTA ELÉTRICA' */
296   testeportaletrica_DW.is_LIGADO = testeportale_IN_NO_ACTIVE_CHILD;
297   testeportaletrica_DW.is_active_c1_testeportaletrica = 0U;
298   testeportaletrica_DW.is_c1_testeportaletrica =
      testeportale_IN_NO_ACTIVE_CHILD;
299 }
300
301 /* Model terminate function */
302 void testeportaletrica_terminate(void)
303 {
304   /* (no terminate code required) */
305 }
306
307 /*
308  * File trailer for generated code.
309  *
310  * [EOF]
311  */
312
```

A.5 – CÓDIGO DO SISTEMA CLIMATIZADOR

```
1  /*
2   * File: testeclimatizador.c
3   *
4   * Code generated for Simulink model 'testeclimatizador'.
5   *
6   * Model version                  : 1.1
7   * Simulink Coder version         : 8.10 (R2016a) 10-Feb-2016
8   * C/C++ source code generated on : Tue Jun 06 21:04:08 2017
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 "testeclimatizador.h"
17 #include "testeclimatizador_private.h"
18
19 /* Named constants for Chart: '<Root>/CLIMATIZADOR' */
20 #define testeclimati_IN_NO_ACTIVE_CHILD ((uint8_T)0U)
21 #define testeclimatizad_IN_RESFRIAMENTO ((uint8_T)3U)
22 #define testeclimatizado_IN_AQUECIMENTO ((uint8_T)1U)
23 #define testeclimatizador_IN_INIT      ((uint8_T)2U)
24
25 /* Block signals (auto storage) */
26 B_testeclimatizador_T testeclimatizador_B;
27
28 /* Block states (auto storage) */
29 DW_testeclimatizador_T testeclimatizador_DW;
30
```

```
31 /* Real-time model */
32 RT_MODEL_testeclimatizador_T testeclimatizador_M_;
33 RT_MODEL_testeclimatizador_T *const testeclimatizador_M =
   &testeclimatizador_M_;
34
35 /* Model step function */
36 void testeclimatizador_step(void)
37 {
38   boolean_T rtb_DigitalInput_0;
39   boolean_T rtb_DigitalInput2_0;
40   boolean_T rtb_DigitalInput1_0;
41   uint8_T tmp;
42
43   /* S-Function (arduinodigitalinput_sfcn): '<Root>/Digital Input' */
44   rtb_DigitalInput_0 =
MW_digitalRead(testeclimatizador_P.DigitalInput_p1);
45
46   /* S-Function (arduinodigitalinput_sfcn): '<Root>/Digital Input2' */
47   rtb_DigitalInput2_0 =
MW_digitalRead(testeclimatizador_P.DigitalInput2_p1);
48
49   /* S-Function (arduinodigitalinput_sfcn): '<Root>/Digital Input1' */
50   rtb_DigitalInput1_0 =
MW_digitalRead(testeclimatizador_P.DigitalInput1_p1);
51
52   /* Chart: '<Root>/CLIMATIZADOR' incorporates:
53    *  S-Function (arduinodigitalinput_sfcn): '<Root>/Digital Input'
54    *  S-Function (arduinodigitalinput_sfcn): '<Root>/Digital Input1'
55    *  S-Function (arduinodigitalinput_sfcn): '<Root>/Digital Input2'
56    */
57   /* Gateway: CLIMATIZADOR */
58   /* During: CLIMATIZADOR */
59   if (testeclimatizador_DW.is_active_c6_testeclimatizador == 0U) {
60     /* Entry: CLIMATIZADOR */
61     testeclimatizador_DW.is_active_c6_testeclimatizador = 1U;
```

```
62
63      /* Entry Internal: CLIMATIZADOR */
64      /* '<S1>:23:1' sf_internal_predicateOutput = ... */
65      /* '<S1>:23:1' liga == 1; */
66      if (rtb_DigitalInput_0) {
67        /* Transition: '<S1>:23' */
68        testeclimatizador_DW.is_c6_testeclimatizador =
    testeclimatizador_IN_INIT;
69      }
70    } else {
71      switch (testeclimatizador_DW.is_c6_testeclimatizador) {
72       case testeclimatizado_IN_AQUECIMENTO:
73        /* During 'AQUECIMENTO': '<S1>:8' */
74        /* '<S1>:14:1' sf_internal_predicateOutput = ... */
75        /* '<S1>:14:1' T > tref; */
76        if ((int16_T)rtb_DigitalInput1_0 > (int16_T)rtb_DigitalInput2_0)
    {
77          /* Transition: '<S1>:14' */
78          testeclimatizador_DW.is_c6_testeclimatizador =
79            testeclimatizad_IN_RESFRIAMENTO;
80
81          /* Entry 'RESFRIAMENTO': '<S1>:5' */
82          /* '<S1>:5:1' heater = 0 */
83          testeclimatizador_B.heater = 0.0;
84        } else {
85          /* '<S1>:8:1' heater = 1 */
86          testeclimatizador_B.heater = 1.0;
87
88          /* '<S1>:8:1' temp = T */
89          testeclimatizador_B.temp = rtb_DigitalInput1_0;
90        }
91        break;
92
93       case testeclimatizador_IN_INIT:
```

```
94        /* During 'INIT': '<S1>:22' */
95        /* '<S1>:24:1' sf_internal_predicateOutput = ... */
96        /* '<S1>:24:1' T > tref; */
97        if ((int16_T)rtb_DigitalInput1_0 > (int16_T)rtb_DigitalInput2_0)
    {
98          /* Transition: '<S1>:24' */
99          testeclimatizador_DW.is_c6_testeclimatizador =
100           testeclimatizad_IN_RESFRIAMENTO;
101
102         /* Entry 'RESFRIAMENTO': '<S1>:5' */
103         /* '<S1>:5:1' heater = 0 */
104         testeclimatizador_B.heater = 0.0;
105       } else {
106         /* '<S1>:25:1' sf_internal_predicateOutput = ... */
107         /* '<S1>:25:1' T < tref; */
108         if ((int16_T)rtb_DigitalInput1_0 <
    (int16_T)rtb_DigitalInput2_0) {
109           /* Transition: '<S1>:25' */
110           testeclimatizador_DW.is_c6_testeclimatizador =
111             testeclimatizado_IN_AQUECIMENTO;
112
113           /* Entry 'AQUECIMENTO': '<S1>:8' */
114           /* '<S1>:8:1' cooler = 0 */
115           testeclimatizador_B.cooler = 0.0;
116         }
117       }
118       break;
119
120     default:
121       /* During 'RESFRIAMENTO': '<S1>:5' */
122       /* '<S1>:13:1' sf_internal_predicateOutput = ... */
123       /* '<S1>:13:1' T < tref; */
124       if ((int16_T)rtb_DigitalInput1_0 < (int16_T)rtb_DigitalInput2_0)
    {
```

```
125         /* Transition: '<S1>:13' */
126         testeclimatizador_DW.is_c6_testeclimatizador =
127           testeclimatizado_IN_AQUECIMENTO;
128
129         /* Entry 'AQUECIMENTO': '<S1>:8' */
130         /* '<S1>:8:1' cooler = 0 */
131         testeclimatizador_B.cooler = 0.0;
132       } else {
133         /* '<S1>:5:1' cooler = 1 */
134         testeclimatizador_B.cooler = 1.0;
135
136         /* '<S1>:5:1' temp = T */
137         testeclimatizador_B.temp = rtb_DigitalInput1_0;
138       }
139       break;
140     }
141   }
142
143   /* End of Chart: '<Root>/CLIMATIZADOR' */
144
145   /* DataTypeConversion: '<S2>/Data Type Conversion' */
146   if (testeclimatizador_B.cooler < 256.0) {
147     if (testeclimatizador_B.cooler >= 0.0) {
148       tmp = (uint8_T)testeclimatizador_B.cooler;
149     } else {
150       tmp = 0U;
151     }
152   } else {
153     tmp = MAX_uint8_T;
154   }
155
156   /* End of DataTypeConversion: '<S2>/Data Type Conversion' */
157
```

```
158   /* S-Function (arduinodigitaloutput_sfcn): '<S2>/Digital Output' */
159   MW_digitalWrite(testeclimatizador_P.DigitalOutput_pinNumber, tmp);
160
161   /* S-Function (arduinodigitaloutput_sfcn): '<S3>/Digital Output'
    incorporates:
162    *  DataTypeConversion: '<S3>/Data Type Conversion'
163    */
164   MW_digitalWrite(testeclimatizador_P.DigitalOutput_pinNumber_c,
    (uint8_T)
165                   testeclimatizador_B.temp);
166
167   /* DataTypeConversion: '<S4>/Data Type Conversion' */
168   if (testeclimatizador_B.heater < 256.0) {
169     if (testeclimatizador_B.heater >= 0.0) {
170       tmp = (uint8_T)testeclimatizador_B.heater;
171     } else {
172       tmp = 0U;
173     }
174   } else {
175     tmp = MAX_uint8_T;
176   }
177
178   /* End of DataTypeConversion: '<S4>/Data Type Conversion' */
179
180   /* S-Function (arduinodigitaloutput_sfcn): '<S4>/Digital Output' */
181   MW_digitalWrite(testeclimatizador_P.DigitalOutput_pinNumber_e, tmp);
182 }
183
184 /* Model initialize function */
185 void testeclimatizador_initialize(void)
186 {
187   /* Registration code */
188
189   /* initialize error status */
```

```
190   rtmSetErrorStatus(testeclimatizador_M, (NULL));
191
192   /* block I/O */
193   (void) memset(((void *) &testeclimatizador_B), 0,
194                 sizeof(B_testeclimatizador_T));
195
196   /* states (dwork) */
197   (void) memset((void *)&testeclimatizador_DW, 0,
198                 sizeof(DW_testeclimatizador_T));
199
200   /* Start for S-Function (arduinodigitalinput_sfcn): '<Root>/Digital
    Input' */
201   MW_pinModeInput(testeclimatizador_P.DigitalInput_p1);
202
203   /* Start for S-Function (arduinodigitalinput_sfcn): '<Root>/Digital
    Input2' */
204   MW_pinModeInput(testeclimatizador_P.DigitalInput2_p1);
205
206   /* Start for S-Function (arduinodigitalinput_sfcn): '<Root>/Digital
    Input1' */
207   MW_pinModeInput(testeclimatizador_P.DigitalInput1_p1);
208
209   /* Start for S-Function (arduinodigitaloutput_sfcn): '<S2>/Digital
    Output' */
210   MW_pinModeOutput(testeclimatizador_P.DigitalOutput_pinNumber);
211
212   /* Start for S-Function (arduinodigitaloutput_sfcn): '<S3>/Digital
    Output' */
213   MW_pinModeOutput(testeclimatizador_P.DigitalOutput_pinNumber_c);
214
215   /* Start for S-Function (arduinodigitaloutput_sfcn): '<S4>/Digital
    Output' */
216   MW_pinModeOutput(testeclimatizador_P.DigitalOutput_pinNumber_e);
217
218   /* SystemInitialize for Chart: '<Root>/CLIMATIZADOR' */
```

```
219   testeclimatizador_DW.is_active_c6_testeclimatizador = 0U;
220   testeclimatizador_DW.is_c6_testeclimatizador =
    testeclimati_IN_NO_ACTIVE_CHILD;
221 }
222
223 /* Model terminate function */
224 void testeclimatizador_terminate(void)
225 {
226   /* (no terminate code required) */
227 }
228
229 /*
230  * File trailer for generated code.
231  *
232  * [EOF]
233  */
```