UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DIRETORIA DE PESQUISA E PÓS-GRADUAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

GILSON NATAL DALLA STELLA

APLICANDO A METODOLOGIA DE DESENVOLVIMENTO BASEADO EM MODELOS PARA FUNÇÕES DE SOFTWARE AUTOMOTIVO

DISSERTAÇÃO

PONTA GROSSA
2015

GILSON NATAL DALLA STELLA

APLICANDO A METODOLOGIA DE DESENVOLVIMENTO BASEADO EM MODELOS PARA FUNÇÕES DE SOFTWARE AUTOMOTIVO

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre em Engenharia Elétrica do Programa de Pós-Graduação em Engenharia Elétrica, da Universidade Tecnológica Federal do Paraná – Campus Ponta Grossa. Área de Concentração: Instrumentação e Controle.

Orientador: Prof. Dr. Max Mauro Dias Santos

Co-orientador: Prof. Dr. Sergio Luiz Stevan Jr.

PONTA GROSSA

2015

Ficha catalográfica elaborada pelo Departamento de Biblioteca
da UTFPR Câmpus Ponta Grossa
n.35/15

S824 Stella, Gilson Natal Dalla

Aplicando a metodologia de desenvolvimento baseado em modelos para funções de software automotivo. / Gilson Natal Dalla Stella. Ponta Grossa, 2015.
123 f.: il.; 30 cm.

Orientador: Prof. Dr. Max Mauro Dias Santos
Co-orientador: Prof. Dr. Sergio Luiz Stevan Jr.

Dissertação (Mestrado em Engenharia Elétrica). Programa de Pós-Graduação em Engenharia Elétrica. Universidade Tecnológica Federal do Paraná, Ponta Grossa, 2015.

1. Automóveis - Equipamento eletrônico. 2. Sistemas embarcados (Computadores). 3. Engenharia elétrica. I. Santos, Max Mauro Dias. II. Stevan Jr., Sergio Luiz. III. Universidade Tecnológica Federal do Paraná. IV. Título.

CDD 621.3

Universidade Tecnológica Federal do Paraná
Campus Ponta Grossa
Diretoria de Pesquisa e Pós-Graduação
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

FOLHA DE APROVAÇÃO

Título da Dissertação Nº 15/2015

APLICANDO A METODOLOGIA DE DESENVOLVIMENTO BASEADO EM MODELOS PARA FUNÇÕES DE SOFTWARE AUTOMOTIVO

por

Gilson Natal Dalla Stella

Esta dissertação foi apresentada às 8 horas e 30 minutos do dia 15 de maio de 2015 como requisito parcial para a obtenção do título de MESTRE EM ENGENHARIA ELÉTRICA, com área de concentração em Instrumentação e Controle, Programa de Pós-Graduação em Engenharia de Produção. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo citados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

Prof. Dr. Francisco Vasques de Carvalho (FEUP)

Prof. Dr. João Francisco Justo Filho (USP)

Prof. Dr. Abraham Elias Ortega Paredes (UTFPR-PG)

Prof. Dr. Sérgio Luiz Stevan Jr. (UTFPR-PG)
Co-orientador

Prof. Dr. Max Mauro Dias Santos (UTFPR)
Orientador

Prof. Dr. Claudinor Bitencourt Nascimento (UTFPR)
Coordenador do PPGEE

A FOLHA DE APROVAÇÃO ASSINADA ENCONTRA-SE NO DEPARTAMENTO DE REGISTROS ACADÊMICOS DA UTFPR – CÂMPUS PONTA GROSSA

Este trabalho é dedicado a minha esposa Valdevina Terezinha e aos meus filhos Grazielle e Gabriel, sendo as minhas principais razões para persistir e prosseguir.

AGRADECIMENTOS

À esta universidade, seu corpo docente, direção e administração que oportunizaram o apoio e estrutura necessária para adquirir conhecimentos necessários para desenvolver este trabalho.

Ao meu orientador Prof. Max Mauro Dias Santos, pelo estímulo ao desenvolvimento desta dissertação e fundamentais ensinamentos que contribuíram de forma incomensurável ao longo da pesquisa desenvolvida.

Ao meu co-orientador Prof. Sergio Luiz Stevan Jr. pelo suporte e ensinamentos durante todo este processo de pesquisa.

Aos meus pais (in memoriam), pelo amor, ensinamentos e exemplos vida que me proporcionaram.

Aos meus amigos Celso, Dahge e Mauricio que, na realidade são muito mais que amigos, são como irmãos verdadeiros, pessoas muito importantes em diversos momentos de minha vida.

Aos meus companheiros de estudos João Henrique Neme, Felipe Franco, José Jair Mendes Jr (JJ).

A Anacom e OpenCadd por dar suporte e disponibilizarem ferramentas que foram utilizadas durante o desenvolvimento deste trabalho.

A Deus por ter me dado saúde e força para superar as dificuldades.

E a todos os colegas e amigos que direta ou indiretamente fizeram parte da minha formação, o meu muito obrigado.

“Seja você quem for, seja qual for a posição social que você tenha na vida, a mais alta ou a mais baixa, tenha sempre como meta muita força, muita determinação e sempre faça tudo com muito amor e com muita fé em Deus, que um dia você chega lá. De alguma maneira você chega lá”.

SENNA, Ayrton,1994.

RESUMO

STELLA, Gilson Natal Dalla. **Aplicação do Método de Desenvolvimento Baseado em modelos para uma Função de Software Automotivo**. 2015. 123 folhas. Dissertação (Mestrado em Engenharia Elétrica) – Programa de Pós-Graduação em Engenharia Elétrica, Universidade Tecnológica Federal do Paraná, Ponta Grossa, 2015.

Os veículos automotivos possuem diversas funcionalidades em que, para ser melhoradas ou inovadas, é mandatório que se garanta melhor desempenho, segurança e confiabilidade. Para isto, é necessário criar, ou aperfeiçoar os já existentes, métodos e processos de desenvolvimento de software embarcado automotivo. As metodologias de desenvolvimento tradicionais não atendem aos requisitos e complexidade destes sistemas. Desta forma, a metodologia de desenvolvimento de software baseado em modelos (MBD – *Model-Based Design*) pode contribuir grandemente, por tornar possível a otimização com recursos de análise e testes. Por estes motivos, este trabalho busca demonstrar como se pode aplicar esta metodologia desenvolvimento baseado em modelos para as funções de software automotivo, considerando as etapas deste processo como MIL (*Model-In-the-Loop*), SIL (*Software-In-the Loop*), PIL (*Processor-In-the Loop*) e RCP (*Rapid Control Prototyping*) e comprovar a sua eficácia. Considera-se ainda que foram utilizadas ferramentas compatíveis e essenciais para o processo de desenvolvimento, tais como para definição dos requisitos, elaboração da planta física, projeto controlador, verificação e testes.

Palavras-chave: Desenvolvimento Baseado em Modelo. Software Automotivo. *Model-in-the-Loop. Software-in-the-Loop. Processor-in-the-Loop. Rapid Control Prototyping.*

ABSTRACT

STELLA, Gilson Natal Dalla. **Model Based Design for Automotive Software Function**. 2015. 123 pages. Dissertação (Mestrado em Engenharia Elétrica) – Programa de Pós-Graduação em Engenharia Elétrica, Universidade Tecnológica Federal do Paraná, Ponta Grossa, 2015.

The vehicles has many functions wherein, to be improve or innovated, is essential that the warranty of a better performance, safety and reliability. For this, is necessary to create, or improve the already existing, developing methods and processes of automotive embedded software. However, the traditional development methodologies do not meet the requirements and complexity of these systems. Therefore the methodology of model-based design (MBD) may contribute greatly, by making possible the optimization using analysis resources and tests. For these reasons, this work seeks to demonstrate how you can apply this development methodology based on models for automotive software functions, considering the steps in this process as MIL (Model-In-the-Loop), SIL (Software-In-the Loop) , PIL (Processor-In-the Loop) and RCP (Rapid Control Prototyping) and prove its effectiveness. It is further considered that were used compatible and essential tools for the development process, such as for requirements definition, development of physical plant, controller design, verification and testing.

Keywords: Model Based Design. Automotive Software. Model-in-the-Loop. Software-in-the-Loop. Processor-In-the-Loop. Rapid Control Prototyping.

LISTA DE FIGURAS

Figura 1: Crescente demanda por componentes elétricos, eletrônicos e *software* para veículos automotores. 18
Figura 2: Fluxo de dados em ECUs e tipos de ECUs automotivas 26
Figura 3: Sequência de desenvolvimento de software automotivo....28
Figura 4: Exemplo de diagrama do modelo em V 29
Figura 5: Método de desenvolvimento tradicional 34
Figura 6: Desenvolvimento Baseado em Modelo (MBD)....35
Figura 7: Sequência de métodos para verificação de sistema embarcado 36
Figura 8: Fluxo de controle para o MIL mostrando telas genéricas dos programas utilizados 37
Figura 9: Fluxo de controle para o SIL demonstrando telas genéricas dos programas utilizados 37
Figura 10: Fluxo de controle para o PIL demonstrando telas genéricas dos programas utilizados e imagens ilustrativas das plataformas 38
Figura 11: Fluxo de controle para o HIL demonstrando telas genéricas dos programas utilizados e imagens ilustrativas das plataformas 39
Figura 12: Plataforma de *hardware MicroAutoBox*® modelo DS1401 com os módulos DS1505 e DS1507, da *dSpace*® 41
Figura 13: Diagrama do blocos da plataforma de *hardware MicroAutoBox*®, da *dSpace*® 41
Figura 14: Comparação do desenvolvimento tradicional com o desenvolvimento do *Rapid Control Prototyping* 42
Figura 15: Estratégia de controle e estágios de verificação 42
Figura 16: Diagrama de controlador projetado em Simulink para o hardware RCP. .43
Figura 17: Fullpass, o sistema prototipagem substitui completamente a ECU existente. 44
Figura 18: Bypass, novas funções rodam no sistema de prototipagem, enquanto algoritmos sem mudanças continuam na ECU existente. 44
Figura 19: B*ypass*, exemplo de função desviada para o MABX usado no RCP....45
Figura 20: Diagrama em blocos do MCU *Kinetis* KL25Z. 47
Figura 21: Imagem da plataforma FRDM-KL25Z. 48
Figura 22: Sequência de desenvolvimento deste trabalho e ferramentas utilizadas durante o processo 49
Figura 23: Distribuição básica das ECUs automotivas. 50
Figura 24: Diagrama de blocos de uma ICM 50
Figura 25: Diagrama de blocos de uma ECM....51
Figura 26: Diagrama de blocos de uma BCM....52
Figura 27: Sistema de iluminação externo automotivo 56
Figura 28: Diagrama *Simulink*® para o sistema do controlador da função ELS 58
Figura 29: Diagrama *Matlab*/*Simulink*® dos Subsistemas da função ELS 59
Figura 30: Diagrama de estados do *Matlab*/*Stateflow*® referente a posição da chave de ignição "*Ignition SS*" 60
Figura 31: Gráfico dos sinais de entrada e saída do subsistema *Ignition_SS* do controlador de iluminação 61
Figura 32: Diagrama *Matlab*/*Stateflow*® para a lógica das luzes de posição....63
Figura 33: Gráfico das entradas e saídas para a lógica das luzes de posição....64
Figura 34: Diagrama *Matlab/Stateflow*® para a lógica da luz baixa. 65

Figura 35: Gráfico dos sinais de entrada e saída para a lógica da luz baixa 66
Figura 36: Diagrama *Matlab/Stateflow*® para lógica da luz alta.....67
Figura 37: Gráfico dos sinais de entrada e saída para a lógica da luz alta. 68
Figura 38: Diagrama *Matlab/Stateflow*® para a lógica da luz de freio.....69
Figura 39: Gráfico dos sinais de entrada e saída para a lógica da luz de freio. 70
Figura 40: Diagrama *Matlab/Stateflow*® para lógica a luz de indicação direita.....71
Figura 41: Diagrama *Matlab/Stateflow*® lógica para a luz de indicação esquerda....72
Figura 42: Diagrama *Matlab/Stateflow*® referente intermitência das luzes de indicação 73
Figura 43: Diagrama do subsistema de sincronismo das luzes de indicação 73
Figura 44: Gráfico dos sinais de entrada e saída para a lógica da luz de indicação direita, esquerda e alerta.....74
Figura 45: Diagrama *Stateflow*® para a lógica da luz de ré. 75
Figura 46: Gráfico dos sinais de entrada e saída para a lógica da luz de ré.....76
Figura 47: Diagrama para validação do controlador de iluminação com os sinais de entrada gerados pelo *Signal Builder*®.....77
Figura 48: Sinais de entradas utilizados para o caso de teste da função ELS.....78
Figura 49: Entradas dos sinais de falha para o caso de teste da função ELS. 79
Figura 50: Saídas dos sinais referentes ao caso de teste da função ELS. 80
Figura 51: Saídas de erros referente ao caso de teste da função ELS. 80
Figura 52: Diagrama *Matlab/Simulink*® da *S-Function* gerada a partir do modelo inicial do controlador da função ELS.....82
Figura 53: Gráfico comparativo das saídas dos sinais das etapas MIL e SIL. 83
Figura 54: Gráfico comparativo das saídas de erros das etapas MIL e SIL. 84
Figura 55: Módulo do painel de instrumentos.....86
Figura 56: Diagrama *Matlab/Simulink*® do controlador da função IC.....88
Figura 57: Diagrama *Matlab/Simulink*® dos blocos do controlador da função IC.....89
Figura 58: Diagrama *Matlab/Simulink*® do bloco Ignition_Position_TT.....89
Figura 59: Diagrama *StateFlow*® do bloco *Instrument_Cluster_SS*.....91
Figura 60: Diagrama *Matlab/StateFlow*® do bloco *Default_Set*, *Default_Reset* e *Ignition*.....92
Figura 61: Diagrama parcial *Matlab/StateFlow*® do Painel de instrumentos.....93
Figura 62: Gráfico para validação do bloco *Ignition Position_SS*.....95
Figura 63: Gráfico para validação dos sinais *Park_Position_Out*, *Lights_Alert_Out*, *Position_Beam_Out* e *High_Beam_Out*.....95
Figura 64: Gráfico para validação dos sinais *Blinker_Right_Out*, *Blinker_Left_Out*, *Blinker_Alert_Out*, e *Door_Alert_Out*. 96
Figura 65: Gráfico para validação dos sinais *Engine_Alert_Out*, *Seat_Belt_Alert_Out*, *Engine_Fluid_Alert_Out* e *Brake_Fluid_Alert_Out*.....96
Figura 66: Gráfico para validação dos sinais *Engine_Temp_Alert*, *Engine_Temp_Out*, *Fuel_Level_Alert* e *Fuel_Level_Out*.....97
Figura 67: Gráfico para validação do sinal *Battery_Alert*. 97
Figura 68: Gráfico para validação dos sinais *Velocity_Out* e *RPM_Out*,.....97
Figura 69: Diagrama *Matlab/Simulink*® do controlador da função IC para o SIL 98
Figura 70: Gráfico para validação da etapa SIL dos sinais *Park_Position_Out*, *Lights_Alert_Out*, *Position_Beam_Out* e *High_Beam_Out*. 99
Figura 71: Gráfico para validação da etapa SIL dos sinais *Blinker_Right_Out*, *Blinker_Left_Out*, *Blinker_Alert_Out*, e *Door_Alert_Out*.....99
Figura 72: Gráfico para validação da etapa SIL dos sinais Engine_Alert_Out, Seat_Belt_Alert_Out, Engine_Fluid_Alert_Out e Brake_Fluid_Alert_Out.....100

Figura 73: Gráfico para validação da etapa SIL dos sinais *Fuel_Level_Alert*, *Fuel_Level_Out*, *Engine_Temp_Alert*, *Engine_Temp_Out* e *Battery Charge*. .100
Figura 74: Gráfico para validação da etapa SIL dos sinais *TPMS_Alert*, *Velocity_Out* e *RPM_Out*, ...101
Figura 75: Função, em *Labview®,* que gera os sinais na entrada do PIL. 102
Figura 76: Função, em *Labview®*, que captura sinais nas saídas do PIL. 103
Figura 77: Fluxograma de atividades para as funções PIL. 103
Figura 78: Foto do myRIO. 104
Figura 79: Foto da montagem para validação do PIL das funções ELS e IC. 105
Figura 80: Diagrama *Simulink*® do controlador da função ELS para o PIL. 106
Figura 81: Gráfico para validação da etapa PIL dos sinais entradas para o ELS. ..107
Figura 82: Gráfico para validação da etapa PIL dos sinais saídas para o ELS. 107
Figura 83: Diagrama *Simulink*® do controlador da função IC para o PIL. 108
Figura 84: Gráfico para validação da etapa PIL dos sinais entradas para o IC.....108
Figura 85: Gráfico para validação da etapa PIL dos sinais saídas para o IC. 109
Figura 86: Foto do painel de instrumentos físicos. 109
Figura 87: Entradas para a posição da chave de ignição. 110
Figura 88: Ligações feitas entre o MABX e a DAQ 111
Figura 89: Fluxo de trabalho desenvolvido para o RCP 111
Figura 90: Entradas criadas para o caso de teste do RCP 113
Figura 91: Sinais de emulação da posição da chave de ignição e do nível de bateria 113
Figura 92: Gráfico dos sinais de entrada para o Caso de Teste 1. 115
Figura 93: Gráfico dos sinais de saída para o Caso de Teste 1. 115
Figura 94: Gráfico dos sinais de entrada para o Caso de Teste 2. 116
Figura 95: Gráfico dos sinais de saída para o Caso de Teste 2. 116
Figura 96: Gráfico dos sinais de entrada para o Caso de Teste 3. 117
Figura 97: Gráfico dos sinais de saída para o Caso de Teste 3. 118
Figura 98: Gráfico dos sinais de entrada para o Caso de Teste 4. 118
Figura 99: Gráfico dos sinais de saída para o Caso de Teste 4. 119

LISTA DE TABELAS

Tabela 1: Tabela das etapas desenvolvidas para cada função.....46
Tabela 2: Definição dos requisitos de funcionamento. 57
Tabela 3: Tabela de sinais do *Matlab/Stateflow*® referente ao Ignition Position. 61
Tabela 4: Tabela de sinais do *Matlab/Stateflow*® referente ao Position Lights SS. ..63
Tabela 5: Tabela de sinais do *Matlab/Stateflow*® referente ao Low Beam Light SS. 65
Tabela 6: Tabela de sinais do *Matlab/Stateflow*® referente ao Full Beam Light SS..67
Tabela 7: Tabela de sinais do *Matlab/Stateflow*® referente ao *Brake Light SS*. 69
Tabela 8: Tabela de sinais do *Matlab/Stateflow*® referente ao Right Light SS. 71
Tabela 9: Tabela de sinais do *Matlab/Stateflow*® referente ao Left Light SS.....72
Tabela 10: Tabela de sinais do Matlab/Stateflow® referente ao Reverse Light SS. .75
Tabela 11: Requisitos da função do painel de instrumentos. 86
Tabela 12: Estados de saída do *Stateflow*® do bloco *Ignition_Position_SS*. 90
Tabela 13: Requisitos da função de vidros elétricos. 114

LISTA DE SIGLAS E ACRÔNIMOS

BCM	*Body Control Module*
CAN	*Controller Area Network*
CCM	*Central Control Module*
CSV	*Comma Separated Value*
CTM	*Central Time Module*
DAQ	*Data Aquisition*
E/E	Elétrica e Eletrônica
EBCM	*Electronic Brake Control Module*
ECM	*Engine Control Module*
ECU	*Electronic Control Unit*
GEM	*General Electronic Module*
HIL	*Hardware-in-the-Loop*
IC	*Instrument Cluster*
ICM	*Instrument Control Module*
IP	*Instrument Panel*
IPC	*Instrument Panel Control*
LIN	*Local Interconnect Network*
MABX	*MicroAutoBox®*
MBD	*Model Based Design*
MCU	*Microcontroler Unit*
MIL	*Model-in-the-Loop*
OEM	*Original Equipment Manufacturer*
PCM	*Powertrain Control Module*
PIL	*Processor-in-the-Loop*
PWM	*Pulse Width Module*
RCP	*Rapid Control Prototyping*
SCI	*Serial Communication Interface*
SCM	*Suspension Control Module*
SIL	*Software-in-the-Loop*
SPI	*Serial Peripheral Interface*
TCM	*Transmission Control Module*
TCU	*Transmission Control Unit*
UPA	*Ultrasonic Park Assist*

SUMÁRIO

1 INTRODUÇÃO 17
1.1 TEMA 20
1.1.1 Delimitação do Tema 20
1.2 PROBLEMA 22
1.3 OBJETIVOS 22
1.3.1 Objetivo Geral 22
1.3.2 Objetivos Específicos 22
1.4 JUSTIFICATIVA 23
1.5 MÉTODO DA PESQUISA 23
2 METODOLOGIA DE DESENVOLVIMENTO BASEADO EM MODELOS . 25
2.1 MODELO V PARA DESENVOLVIMENTO DE SOFTWARE AUTOMOTIVO 28
2.2 DESENVOLVIMENTO BASEADO EM MODELO 32
2.3 DESENVOLVIMENTO TRADICIONAL VERSUS DESENVOLVIMENTO BASEADO EM MODELOS 33
2.4 MÉTODOS DE TESTE PARA MBD 36
2.4.1 Model-In-The-Loop (MIL) 36
2.4.2 Software-In-The-Loop (SIL) 37
2.4.3 Processor-In-The-Loop (PIL) 38
2.4.4 Hardware-In-The-Loop (HIL) 38
2.5 Rapid Control Prototyping (RCP) 39
3 DESENVOLVIMENTO BASEADO EM MODELOS: MIL, SIL E PIL 46
3.1 ARQUITETURA DAS ECUs AUTOMOTIVAS 49
3.2 PROJETO DA FUNÇÃO DO SISTEMA DE ILUMINAÇÃO EXTERNA AUTOMOTIVA 52
3.2.1 Controle Eletrônico das Funções de Iluminação Externa 53
3.2.2 Model-In-the-Loop para um Sistema de Iluminação Externa Automotiva ... 53
3.2.2.1 Definição dos requisitos da função sistema de iluminação externa automotiva 57
3.2.2.2 Projeto da função sistema de iluminação externa automotiva 58
3.2.2.3 Detalhamento dos subsistemas do sistema de iluminação externa automotiva 61
3.2.2.4 Teste e validação da função sistema de iluminação externa automotiva ... 76
3.2.3 Software-In-The-Loop para um Sistema de Iluminação Externa Automotiva 81
3.3 PROJETO DA FUNÇÃO DO PAINEL DE INSTRUMENTOS AUTOMOTIVO 85
3.3.1 Características da Função do Painel de Instrumentos Automotivo 85
3.3.2 Model-In-The-Loop para um Painel de Instrumentos Automotivo 85
3.3.2.1 Descrição da planta física do painel de instrumentos automotivo 86
3.3.2.2 Definição dos requisitos do painel de instrumentos automotivo 86
3.3.2.3 Projeto do controlador do painel de instrumentos automotivo 87
3.3.2.4 Detalhamento do projeto dos subsistemas do painel de instrumentos automotivo 88
3.3.2.5 Teste e validação da função painel de instrumentos automotivo 94
3.3.3 Software-In-The-Loop para um Painel de Instrumentos Automotivo 98
3.4 PROCESSOR-IN-THE-LOOP 101
3.4.1 Descrição do Hardware para o PIL 101

3.4.2 Processor-In-The-Loop para um Sistema Externo de Iluminação Automotiva 105
3.4.3 Processor-In-The-Loop para um Painel de Instrumentos Automotivo 107
4 RAPID CONTROL PROTOTYPING (RCP) PARA UM SISTEMA DE ILUMINAÇÃO EXTERNA AUTOMOTIVA 110
4.1 RAPID CONTROL PROTOTYPING PARA UM SISTEMA DE ILUMINAÇÃO EXTERNA AUTOMOTIVA 110
4.2 RAPID CONTROL PROTOTYPING PARA UM SISTEMA DE VIDRO ELÉTRICO AUTOMOTIVO 114
5 CONSIDERAÇÕES FINAIS 120
REFERÊNCIAS 121

1 INTRODUÇÃO

Atualmente, o setor automotivo do Brasil se destaca mundialmente por ser o oitavo maior produtor de veículos automotores e ter o quarto maior mercado consumidor. Ocupando a quarta posição em volume de veículos produzidos entre o grupo dos países emergentes (BRIC – Brasil, Rússia, Índia, China e adicionalmente África do Sul e México),segundo dados da Organização Internacional de Produtores de Veículos Automotores, a OICA (2014).

Deve-se considerar também que o setor automotivo no Brasil contribui significativamente de forma positiva para a economia do Brasil desde os anos 1960, segundo a Associação Nacional dos Fabricantes de Veículos Automotores, tendo atualmente uma participação no PIB Industrial de aproximadamente 23% e 5% no PIB do país. Contempla um complexo de mais de 5 mil empresas entre fabricantes de automóveis, autopeças e concessionárias em todo território nacional (ANFAVEA, 2015).

Destaca-se ainda que um setor automotivo é composto por produtos de mobilidade e transporte como veículos de passeio, caminhões, ônibus, motocicletas, máquinas agrícolas, máquinas de construção e outros que utilizam direta ou indiretamente a infraestrutura de transporte disponibilizada pela malha rodoviária do Brasil ou para atividades oriundas da produção de bens e serviços. Este cenário mercadológico é de interesse nacional por ser um vetor importante da economia e PIB. A ligação direta com outros setores da economia impulsiona o desenvolvimento social e econômico nacional.

Para o domínio de aplicação automotivo, sistemas de controle embarcados são, atualmente, o foco principal de inovações e melhoria na qualidade destes produtos. Isto se dá em função da crescente demanda dinâmica dos usuários de veículos automotivos deste mercado.

O uso intensivo de componentes elétricos e eletrônicos na indústria automotiva tem impulsionado inovações no produto e na produção que visam diminuir custos, tempo de produção e melhoria da qualidade do produto final, proporcionando maior conforto e segurança.

A primeira mudança significativa da migração de componentes mecânicos, hidráulicos e pneumáticos por componentes eletroeletrônicos é observada por um

estudo feito por Mercer *et al*. (2011 apud BROY, 2004). O estudo foca em uma previsão de como os fatores de custo no desenvolvimento de um veículo mudarão até o ano de 2015 em comparação ao ano de 2002. Segundo a ANFAVEA (2015), estimou-se que, dos custos totais envolvidos em um produto automotivo, 35% serão destinados a componentes de elétrica e eletrônica. Ela ainda previu que, no espaço de análise de 12 anos, enquanto o custo relativo à funções de motorização dos carros terá um pequeno aumento, os custos relacionados com funções elétricas e eletrônicas triplicarão.

Segundo Ploss (2008) de 1997 a 2008 a produção de veículos mundial aumentou 44%, enquanto que a utilização de elementos de eletrônicos (E/E) cresceu 155% e de semicondutores 355%.

Pode-se considerar que os conteúdos de eletrônica, controle e software em automóveis, correspondem de 30% a 40% do custo total de um veículo, e a demanda por novos conteúdos para os veículos da próxima geração, necessitam de um grau de complexidade exponencial para o desenvolvimento de software embarcado automotivo em arquiteturas E/E. A Figura 1 ilustra a participação de componentes no custo total do veículo entre 2010 e 2020 no mundo (Mecânico: 55%, Eletrônica: 24%, Software: 13%, Outros: 8%) (VINCENTELLI, 2006).

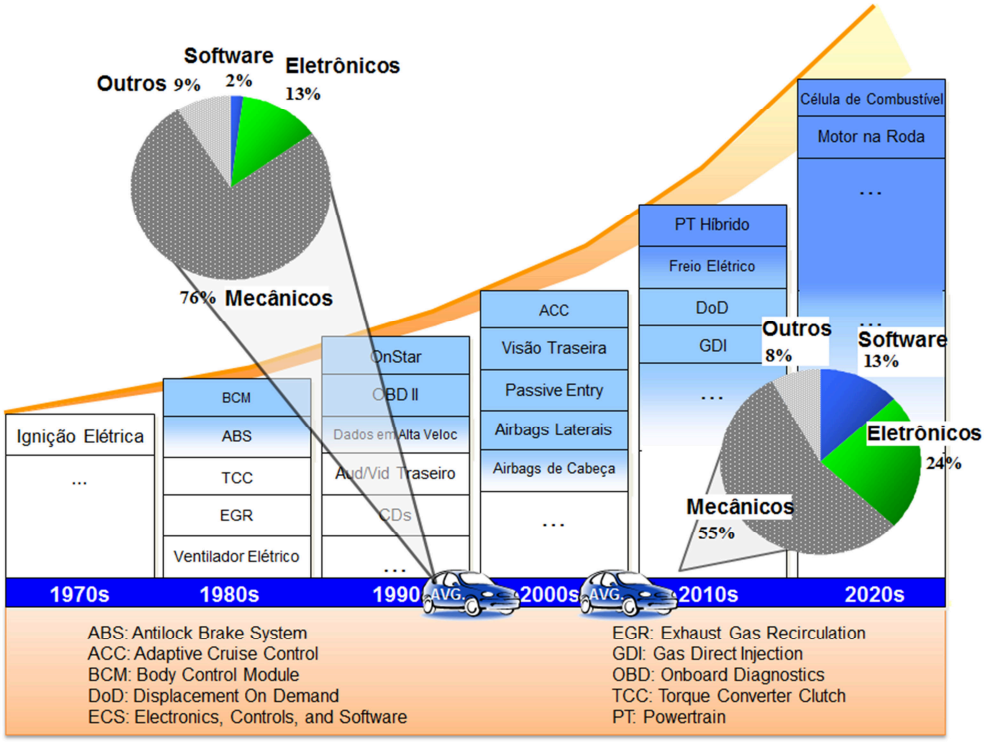


Figura 1: Crescente demanda por componentes elétricos, eletrônicos e *software* para veículos automotores.
Fonte: Matt Tsien (VINCENTELLI, 2006) (adaptado pelo autor, 2015).

O motivo destas mudanças foi em função do menor custo, tamanho e maior desempenho que os componentes eletroeletrônicos alcançam, possibilitando a sua melhor aplicação no controle de sistemas mecatrônicos. Como resultado disso, o conteúdo dos sistemas de *software* embarcado em automóveis cresceu rapidamente nos últimos anos.

No domínio automotivo os subsistemas automotivos são gerenciados por unidades de controle eletrônicas denominadas por ECU (*Electronic Control Unit*). Considerando que um veículo é um conjunto de subsistemas, este possui diversos tipos de ECUs capazes de gerenciar funções principais como controle do motor e transmissão, e também outras espalhadas nas diferentes funções de carroceria, como vidro elétrico, trava elétrica, retrovisores elétricos, teto solar elétrico, dentre outros. 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).

Deve-se considerar que o *software* é parte integrante essencial de um sistema mecatrônico para o qual é considerado como um componente de desenvolvimento.

A engenharia de *software* automotiva consiste na adoção de métodos, processos, ferramentas e padronizações adequados que assegurem aos produtos automotivos um nível de qualidade, confiabilidade, segurança, satisfação às regulamentações e conforto, perceptíveis e mensuráveis aos clientes e a todos os envolvidos.

Este processo consiste em três etapas: desenvolvimento; produção e serviços. Para o caso de sistemas embarcados automotivos, levou-se em consideração apenas o processo de desenvolvimento de *software* de controle para um fim específico.

O objetivo deste trabalho é modelar, validar e avaliar o desempenho do desenvolvimento de um equipamento com um *software* específico em um dispositivo disponível no mercado. Com isto se reduz drasticamente os custos e esforços em relação ao método tradicional que exige a confecção de um protótipo com um *software* e *hardware* dedicados.

Este trabalho tem um custo inicial elevado pela necessidade de ferramentas e equipamentos específicos para esta aplicação o qual foi adquirido e (gentilmente) disponibilizado pela UTFPR. Também foram disponibilizadas algumas ferramentas temporárias pelas empresas OpenCadd (www.opencadd.com.br) e Anacom (www.anacom.com.br).

1.1 TEMA

O presente trabalho consiste em estudar e compreender melhor a metodologia, processo e ferramentas do desenvolvimento baseado em modelos aplicado a software embarcado automotivo, considerando como funções protótipos: sistema de iluminação externa de um veículo e painel de instrumentos de sistemas veiculares. Desta forma, busca-se destacar e apresentar as vantagens e limitações por adotar-se a potencialidade dos métodos e ferramentas envolvidas no processo, proporcionando aos interessados a capacidade de compreensão deste assunto. Este tema foi considerando por existir poucas referências bibliográficas com descritivos simples, funcionais e objetivos.

1.1.1 Delimitação do Tema

Neste trabalho é aplicada a metodologia de desenvolvimento baseado em modelo, a qual consiste na utilização de modernas ferramentas de desenvolvimento gráfico para modelar o sistema físico a ser controlado. Desta forma, pode-se desenvolver a estratégia de controle fazendo uso de testes virtuais nas fases preliminares do projeto. O desenvolvimento da estratégia de controle virtual em ferramentas computacionais permite a geração de código automático destinado a embarcar em um sistema final previamente determinado.

Desta maneira, não é necessário se preocupar em desenvolver o funcionamento do sistema de maneira textual e com codificação manual, algo que pode consumir muito tempo e recursos. Ao contrário, todo o funcionamento pode ser idealizado de maneira gráfica e através de diagramas de blocos, que resultam em desenvolvimentos de fácil alteração, ajuste e/ou manutenção (uma interface muito mais fácil de ser utilizada).

Estes modelos são desenvolvidos em ferramentas de *software* específicas e podem ser facilmente simuladas sem necessariamente testar em uma plataforma física. Na abordagem tradicional, os testes e simulações do código eram complicados e por vezes impossíveis, pelo alto custo dos testes, pela dificuldade de integração dos protótipos, e, principalmente, porque podiam gerar muitos erros.

Em busca deste objetivo, este trabalho será balizado pela abordagem do MBD, para desenvolver e testar duas funções desejadas, considerando três etapas tradicionais de verificação de sistemas, *Model-in-the-Loop* (MIL) e *Software-in-the-Loop (SIL), Processor-in-the-Loop (PIL)* (KELEMENOVÁ, 2013).

Para validar as funções, será realizado o *Rapid Control Prototyping* (RCP) que permite que a validação seja feita em um *hardware*, com uma máquina em tempo real. Com o RCP, é possível verificar como as interfaces e a execução da função se comporta no mundo externo.

Estas abordagens apresentam inúmeras vantagens para o projeto, como: redução de tempo, economia dos recursos, facilidade de controle e eliminação de erro de codificação. Isto ocorre pelo fato do MIL e SIL se basearem na modelagem gráfica do funcionamento do *software* em contraste com o modo textual de programação.

Com o PIL e o RCP tem-se grande economia de tempo que normalmente é gasto para o desenvolvimento do código a ser embarcado em um *hardware* para a validação. Estas técnicas permitem que um modelo seja diretamente embarcado em uma plataforma a fim de ser verificado.

Para o desenvolvimento destes modelos serão utilizadas ferramentas específicas. No caso do MIL e SIL serão utilizadas as ferramentas da *MathWorks*® Matlab/*Simulink*® e *Stateflow*®. O *Stateflow*® é um ambiente de simulação de decisões combinacionais e sequenciais baseadas em máquinas de estados e fluxogramas. Para a realização do PIL e RCP será utilizado além destes programas o *LabVIEW*® da *National Instruments*® para simular as entradas de testes destas fases. Durante o processo foi também utilizada a ferramenta da *dSPACE*® chamada *ControlDesk*® que auxiliou na verificação dos modelos desenvolvidos.

1.2 PROBLEMA

Atualmente existe uma ampla utilização de sistemas embarcados automotivos. O nível e volume de novas funções automotivas estão em constante crescimento e com isso a complexidade destes sistemas segue o mesmo caminho. O conhecimento e maturidade de métodos, processos e ferramentas adequados, são fatores diferenciais a este domínio de aplicação.

Sabe-se que a metodologia MBD é extensivamente utilizada para desenvolvimento de *software* automotivo no mercado internacional, principalmente para desenvolver aplicações de forma rápida e confiável (KELEMENOVÁ, 2013). Entretanto, no cenário nacional não existem referências que possam proporcionar pesquisas e desenvolvimento nesta área. Isto se deve ao fato de existirem poucos profissionais que conheçam totalmente este assunto. Este contexto resulta em um motivador para o desenvolvimento deste trabalho.

1.3 OBJETIVOS

1.3.1 Objetivo Geral

Desenvolver uma estratégia de controle baseada em eventos para o sistema de iluminação externa e painel de instrumentos automotivos usando as metodologias de desenvolvimento baseadas em modelo (MBD) e validar as funções do sistema de iluminação externa e uma função, já desenvolvida anteriormente, do sistema de vidro elétrico, ambas pelo método de *Rapid Control Prototyping* (RCP).

1.3.2 Objetivos Específicos

- Estabelecer o fluxo de trabalho e *as* ferramentas computacionais necessárias para desenvolvimento de função para *software* automotivo.
- Definir os requisitos de funcionamento do subsistema de controle.
- Modelar e validar a planta física a controlar.
- Desenvolver a estratégia de controle baseada no tempo ou evento.
- Definir o conjunto de testes virtuais para validar os requisitos.

- Aplicar as fases de MIL, SIL, PIL e RCP.
- Confrontar e analisar os sinais de teste e as respostas esperadas.

1.4 JUSTIFICATIVA

O setor automobilístico é extremamente dinâmico e exigente. De um lado o consumidor procura cada vez mais melhorias em conforto, segurança, qualidade e incremento de novas tecnologias, fazendo com que o ciclo de vida dos sistemas embarcados seja curto e a sua complexidade aumente.

Por outro lado a indústria pressiona os desenvolvedores e fornecedores pela diminuição de custos e tempo de trabalho para conseguir manter-se neste mercado dinâmico e agressivo, além de ter que satisfazer as restrições de regulamentação. Paralelamente, o aumento constante na demanda de automóveis, a necessidade de que a produção aumente concomitantemente é mandatória.

O desenvolvimento de *softwares* automotivos utiliza métodos, conceitos e ferramentas específicas que são definidos por grandes grupos formados por montadoras e seus fornecedores. Porém, estes conceitos e conhecimentos vêm para nosso país com determinada limitação e atraso, visto que os veículos com elevado nível de conteúdos eletroeletrônico ainda são desenvolvidos, fabricados e consumidos fora de nosso território nacional. Essa realidade justifica a baixa demanda de conhecimento dos profissionais nacionais que esses produtos requerem.

Desta forma, tem-se um motivador para estudos na área de desenvolvimento em engenharia de *software* automotiva com intuito em reduzir esta defasagem tecnológica, melhorando e equalizando a mão de obra nacional ao nível internacional de forma competitiva, com benefícios consideráveis e mensuráveis no contexto mercadológico regional e nacional.

1.5 MÉTODO DA PESQUISA

Inicialmente, foram definidos os requisitos para a função de controle e, para isso, foram analisadas as necessidades do usuário. Depois de mapeadas as coordenadas do projeto, iniciou-se a modelagem do sistema físico a ser controlado.

O passo seguinte foi o desenvolvimento do controlador para toda a planta. Para tal, foram utilizadas ferramentas computacionais específicas da *MathWorks*®, como o *Matlab*®, *Simulink*®, *Stateflow*® e *LabVIEW*®

Após esta etapa, foram definidos sinais de entrada para todas as etapas, de modo a poder simular e observar os sinais de saída de cada caso do sistema de controle. Estes sinais não foram gerados aleatoriamente. Como alguns desses sinais tiveram seu funcionamento interligado, os sinais de entrada foram idealizados de modo a comprovar que não há nenhum erro de lógica no desenvolvimento do modelo.

Inicialmente estas entradas de teste foram utilizadas no modelo desenvolvido no *Simulink*®, para verificar a etapa de MIL. Em seguida, a fase de SIL, consiste em utilizar este modelo do *Simulink*® para gerar automaticamente o código em linguagem C. Utilizando as mesmas entradas de testes, foi feita a verificação do funcionamento do código. A fase de RPC foi executada embarcando o código diretamente em um *hardware* denominado *MicroAutoBox*® (MABX) da *dSPACE*®. Novamente, exatamente as mesmas entradas de teste foram utilizadas, desta vez no MABX, de modo a verificar a estratégia de controle.

2 METODOLOGIA DE DESENVOLVIMENTO BASEADO EM MODELOS

Nesta seção, são abordados alguns conceitos relevantes e fundamentais ao desenvolvimento da pesquisa em relação às tecnologias envolvidas, explorando metodologias, equipamentos, componentes e também *softwares* utilizados.

Os sistemas eletrônicos automotivos consistem em diversas ECUs, as quais controlam subsistemas automotivos, que são interligados através de barramentos de comunicação como *Controller Area Network* (CAN) e *Local Interconect Network* (LIN), satisfazendo uma arquitetura computacional distribuída.

Uma arquitetura de computação distribuída é um sistema caracterizado pela descentralização de funções. Estas funções trocam informações dentro do barramento de comunicação através de mensagens, sendo que cada um destes sistemas computacionais (ECUs) suporta funções de controles embarcados específicos.

Dentre os tipos de ECUs, genericamente falando, os principais são: módulo de gerenciamento do motor (ECM), módulo de gerenciamento de *powertrain* (*Powertrain Control Module,* PCM*),* módulo de gerenciamento da transmissão (*Transmission Control Unit,* TCU), módulo de controle do freio (*Electronic Brake Control Module,* EBCM), módulo de controle central (*Central Control Module,* CCM), módulo de sincronismo central (*Central Timing Module,* CTM*),* módulo eletrônico geral (*General Electronic Module,* GEM), módulo de controle da carroceria (*Body Control Module*, BCM), o módulo de controle dos instrumentos (*Instrument Control Module* – ICM), o módulo de controle de suspensão (*Suspension Control Module,* SCM), módulo de assistência ao estacionamento (*Ultrasonic Park Assist,* UPA), unidade de controle, ou controle de módulo, entre outros. De forma conjunta, estes sistemas são muitas vezes referidos como o sistema computacional do veículo.

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 implementado em *software* determina a ação que precisa ser realizada controlando ou regulando as saídas do veículo;
- Interfaces de Saída (Atuadores e Saídas de comunicação): 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.

A Figura 2 ilustra o fluxo de dados e as ECUs automotivas que são empregadas em veículos, tais como o módulo de controle do motor ou PCM (*Powertrain Control module*), módulo eletrônico geral ou GEM (*General Electronic module*), módulo de controle de carroceria ou BCM (*Body Control Module*), módulo de controle do painel de instrumentos ou IPC (*Instrument Panel Control*) e a unidade de assistência de estacionamento ou UPA (*Universal Park Assist*).

Figura 2: Fluxo de dados em ECUs e tipos de ECUs automotivas
Fonte: Elaborado pelo Autor (2015).

Para o projeto de arquitetura eletroeletrônica distribuída automotiva, existe uma forte separação entre o sistema lógico funcional e o sistema físico, onde o sistema funcional abstrai qualquer aspecto físico e foca puramente sob as dependências lógicas entre as funções de um veículo.

A abstração da forma e características do *hardware* são fatores importantes que proporcionam ao projetista a capacidade de mapear aquelas funções definidas preliminarmente em entidades executáveis sob as entidades computacionais físicas (ECUs distribuídas) e determinam assim as características da rede de comunicação e *hardware* envolvidos. Assim, tem-se a necessidade em caracterizar:

- Mapeamento das ECUs sob a arquitetura elétrica;
- Descrição da especificação técnica do componente (ECM, EBCM, IPC, BCM, etc.);
- Determinação das relações de comunicação entre as ECUs (ex.: especificação do dicionário de dados);
- Especificações de diagnósticos das ECUs.

A Figura 3 apresenta a sequência de desenvolvimento de *software* embarcado automotivo. Na parte superior tem-se inicialmente a determinação dos subsistemas que compõe a função global do veículo, em seguida à definição das funções que serão distribuídas e por último como são descritas em tarefas de forma que possam ser alocadas e executadas fisicamente em ECUs distribuídas.

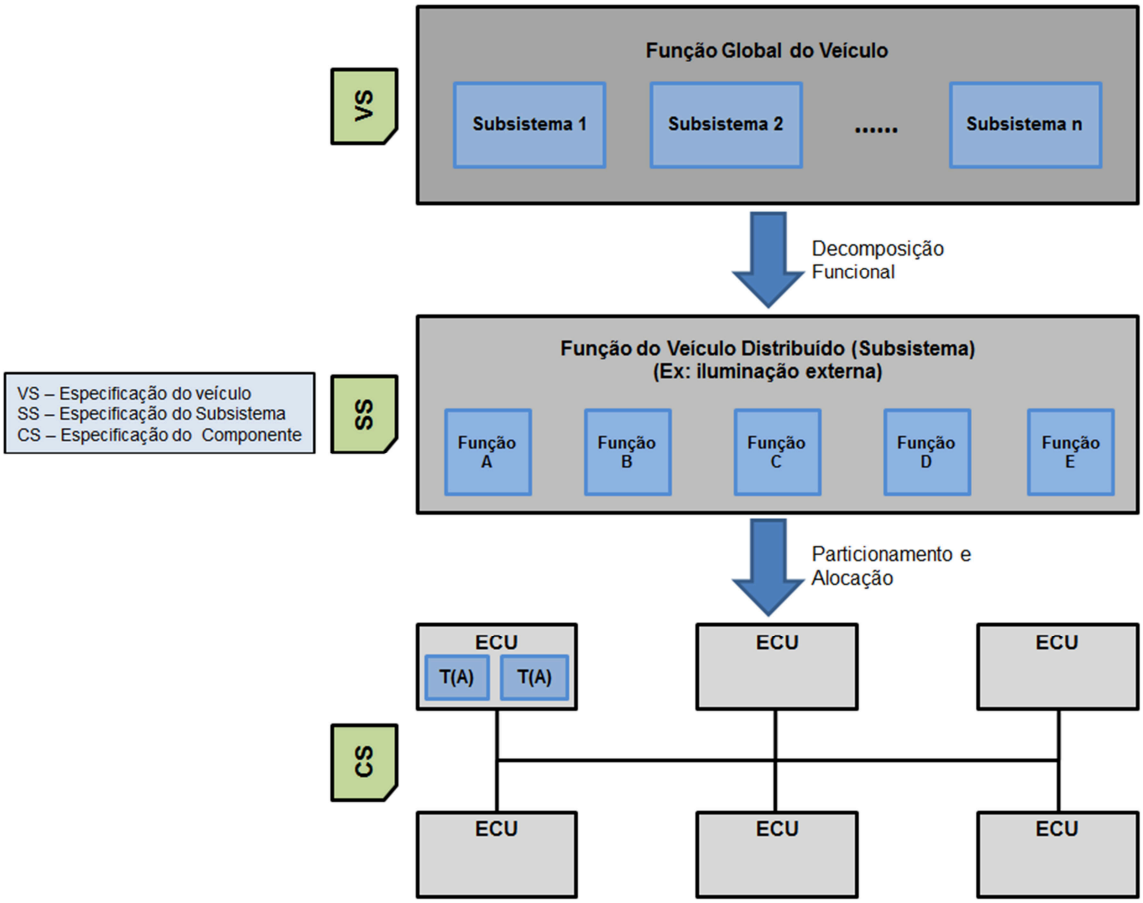


Figura 3: Sequência de desenvolvimento de software automotivo
Fonte: SANTOS, 2010 (adaptado pelo autor, 2015).

Desta forma, buscou-se com o supracitado apresentar uma visão geral de sistemas eletroeletrônicos automotivos de forma genérica com o intuito de ilustrar como a engenharia de *software* automotivo é empregada e como ainda possui diversos desafios para investigação futura. A seguir é apresentado o tradicional método de desenvolvimento de *software* embarcado automotivo denominado *V-Model* (ou Modelo V).

2.1 MODELO V PARA DESENVOLVIMENTO DE SOFTWARE AUTOMOTIVO

O desenvolvimento de *software* embarcado automotivo sob uma arquitetura Eletroeletrônica (E/E) é baseado na metodologia de desenvolvimento Modelo V (RAMACHANDRAN 2009). As especificações são definidas em conjunto e coordenadamente entre as matrizes dos fabricantes originais de equipamentos (*Original Equipment Manufacturer* ou OEMs), fornecedores de produtos e fornecedores de ferramentas.

O modelo V está divido em verificação e validação. Segundo Figueiredo (2012) essas fases podem ser definidas como:

- Verificação: objetivo é verificar se o software atende aos requisitos funcionais e não funcionais especificados. Verificação inclui da realização de testes para encontrar erros.
- Validação: A validação procura assegurar que o sistema atenda as expectativas e necessidades do cliente.

A Figura 4 apresenta as atividades, fluxo e responsabilidade do ciclo de desenvolvimento baseado no Modelo V.

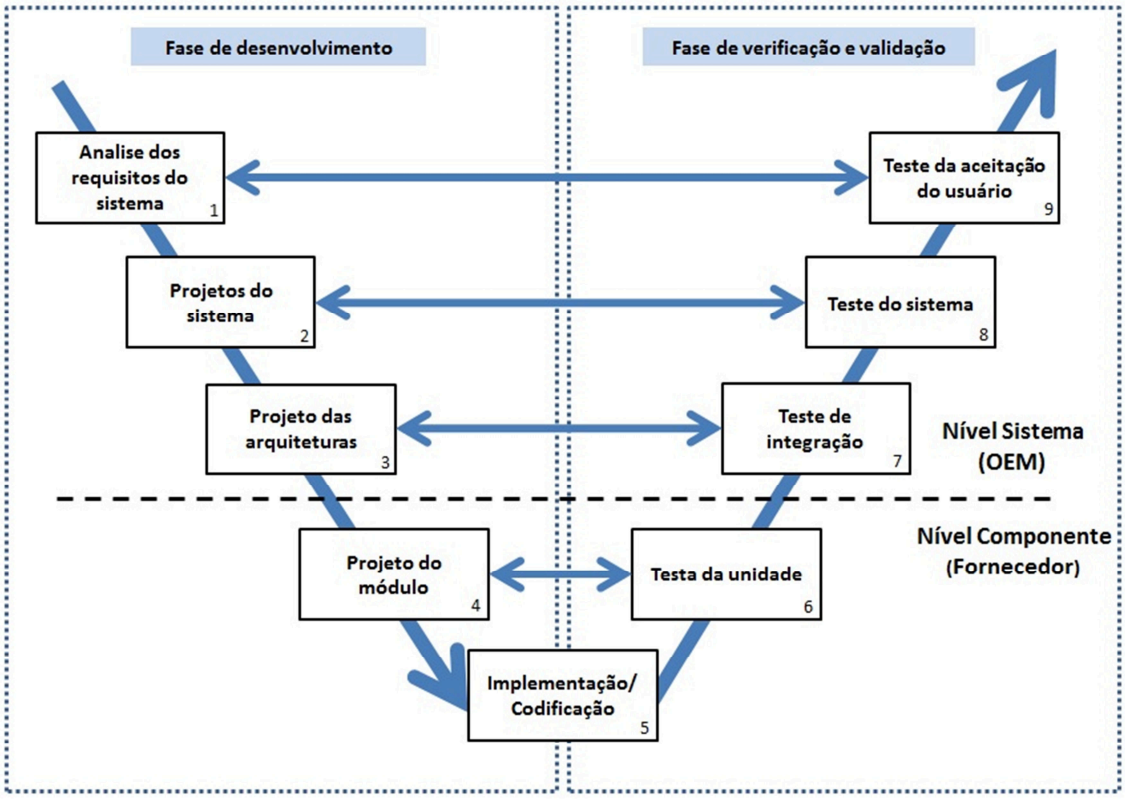


Figura 4: Exemplo de diagrama do modelo em V
Fonte: Elaborado pelo Autor (2015)

A composição do Modelo V pode ser explicada observando seu ciclo de desenvolvimento na Figura 4. Do lado esquerdo são definidos os requisitos, especificações, projeto e implementação, que pode ser chamada, também, de fase de verificação. Já no lado esquerdo têm-se os testes para verificação e validação de componentes, sistemas e conteúdos, que podem ser utilizados durante todas as etapas de desenvolvimento. O processo tem como objetivo enfatizar a qualidade e o

controle do *software* de forma a prover a detecção antecipada dos possíveis erros durante o desenvolvimento do projeto.

Nesse modelo, cada etapa da fase de verificação tem uma ação correspondente na fase de validação. A seguir são descritas as ações da fase de verificação (enumeradas de 1 a 4):

1. **Análise de requisitos:** É o primeiro passo no processo de verificação. Os requisitos do sistema são determinados de acordo com as necessidades do usuário de forma que se tem a preocupação em estabelecer o que o sistema ideal tem de realizar. Nesta etapa, não é possível determinar a forma como o software será projetado ou construído.
2. **Projeto do Sistema:** É a etapa em que os desenvolvedores de sistemas analisam e compreendem o funcionamento do sistema proposto, estudando os documentos dos requisitos do usuário. Podem descobrir possibilidades e técnicas pelas quais as necessidades do usuário podem ser implementadas. Se algum dos requisitos iniciais não são exequíveis, o usuário é informado sobre a questão e uma solução é proposta, consequentemente os documentos de requisitos do usuário podem ser editados adequadamente.
3. **Projeto de Arquitetura:** É a etapa de concepção da arquitetura do *hardware* e do *software,* pode também ser referida como projeto de alto nível. Na seleção de uma arquitetura é que deve-se considerar: lista de módulos: resumo das funcionalidade de cada módulo, suas interface, dependências, tabelas de banco de dados, diagramas da arquitetura, detalhes da tecnologia, etc.
4. **Projeto do Módulo:** Esta etapa também pode ser chamada como o projeto de baixo nível. O sistema projetado está dividido em pequenas unidades ou módulos e cada um deles é explicado para que o programador possa iniciar a codificação diretamente. As especificações dos documentos ou programa do projeto de baixo nível irá conter uma lógica funcional detalhada do módulo

Já a fase de validação é composta pelo desenvolvimento das seguintes ações (enumeradas de 5 a 9):

5. **Implementação ou Codificação:** Refere-se à implementação das funções em alguma linguagem de programação, as quais podem ser codificadas manualmente ou recorrerem à geração automática de código através de ferramentas específicas.
6. **Teste de Unidade:** O plano de testes de unidade são desenvolvidos durante a etapa de projeto do módulo (etapa 4). São executadas para eliminar erros do código ou da unidade. Uma unidade é a menor entidade independente que pode existir, por exemplo, um módulo de programa ou uma unidade de controle eletrônica (ECU – *Electronic Control Unit*). O teste de unidade deve ser capaz de verificar se uma unidade é capaz de funcionar corretamente quando isolada do resto dos códigos ou unidades.
7. **Teste de Integração:** O plano de testes de integração são desenvolvidos durante a fase de projeto da arquitetura (etapa 3) e verificam se as unidades criadas e testadas de forma independente podem coexistir e se comunicar entre si. Os resultados dos testes são compartilhados com a equipe do cliente.
8. **Teste de Sistema:** O plano de testes para o sistema é desenvolvido durante a etapa de projeto do sistema (etapa 2).. O teste do sistema assegura que as expectativas de aplicação desenvolvidas sejam atendidas. Toda a aplicação é testada quanto à sua funcionalidade, interdependência e comunicação em que se verifica se os requisitos funcionais e não funcionais foram cumpridos. São caracterizados: testes de carga, desempenho, stress, regressão, etc.
9. **Teste de Aceitação do Usuário:** Os planos de testes de aceitação do usuário são determinados durante a etapa de análise de requisitos (etapa 1) sendo executados por usuários treinados. São realizados num ambiente de utilização a que se assemelhe ao ambiente de produção, utilizando dados realistas. O plano de testes verifica que o sistema entregue atende aos requisitos do usuário e do sistema, e que está pronto para uso em tempo real.

Baseado nestes modelos verifica-se que utilizando a arquitetura física, que pode ser considerada à planta física do modelo, é possível 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.

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.

2.2 DESENVOLVIMENTO BASEADO EM MODELO (MBD)

Segundo Kelemenová (2013), atualmente as indústrias estão sob constante pressão para reduzir o tempo de desenvolvimento de novos produtos. Trabalhar de forma eficiente é indispensável para o sucesso em um mundo globalizado, principalmente para empresas de alta tecnologia como a automobilística, aeroespacial e de comunicação. Nestas empresas, controles eletrônicos são uma parte vital de cada produto desenvolvido. O desenvolvimento baseado em modelo é uma abordagem que pode cortar custos e diminuir o tempo de um projeto, pois possibilita que se trabalhe com apenas um modelo de uma função ou com um sistema completo em um ambiente integrado de *software*.

A principal característica e vantagem do processo de desenvolvimento baseado em modelo é o desenvolvimento em uma única plataforma, permitindo que se crie a planta do sistema desejado e seu controlador utilizando a mesma ferramenta computacional. Isto facilita a visualização e o entendimento do sistema assim como possíveis erros. O resultado é em um sistema funcional e de fácil verificação, diminuindo significativamente a possibilidade de que os componentes individuais não se encaixem de maneira otimizada.

Segundo a empresa *MathWorks*®, no desenvolvimento baseado em modelo, um modelo de sistema está no centro do processo de desenvolvimento. Parte desde a definição dos pré-requisitos, passa pelo projeto, passa pela implementação áte a

realização dos testes. Esta metodologia está transformando o modo que engenheiros e cientistas trabalham, tirando as tarefas do projeto do laboratório e do campo e levando para ferramentas computacionais.

Segundo Lennon (2007) o MBD simplifica o desenvolvimento de sistemas, proporcionando um ambiente comum para design e comunicação entre diferentes áreas de engenharia.

Algumas principais vantagens que o MBD oferece em comparação com as abordagens tradicionais são (FANTUZZI, 2014):

a) Possibilidade de um ambiente de projeto comum, o que facilita a comunicação, análise de dados e verificação do sistema entre os grupos envolvidos no desenvolvimento;
b) Os engenheiros podem localizar e corrigir erros no início do projeto do sistema, quando o tempo e custo para alguma modificação no sistema são ainda pequenos;
c) Expansão das oportunidades de reutilização de projetos para atualizações de sistema ou para sistemas derivativos.

2.3 DESENVOLVIMENTO TRADICIONAL VERSUS DESENVOLVIMENTO BASEADO EM MODELOS

No processo desenvolvimento tradicional, a elaboração dos requisitos, o desenvolvimento físico do protótipo, o desenvolvimento de códigos, o processo para embarcar e posteriores testes são realizados sequencialmente em ambientes diferentes e com muitos passos manuais. Os requisitos são descritos textualmente, utilizando ferramentas de edição de texto. Os projetos são muitas vezes desenvolvidos em ferramentas de linguagem de domínio específico, o que impede testes do sistema até a fase de implementação em *software* ou *hardware*, dependendo do caso. Os projetos são então traduzidos manualmente em código, o que normalmente significa em um processo que consome muito tempo e propenso a falhas. Em cada fase, defeitos podem surgir, porém estas falhas só serão observadas na fase de implementação.

O resultado disso é que a fase final se torna em uma etapa maçante, demandando muito tempo e recursos. A Figura 5 demonstra a separação que este processo causa no projeto

Figura 5: Método de desenvolvimento tradicional
Fonte: MathWorks, adaptado pelo autor (2015)

O MBD se inicia com o mesmo conjunto de requisitos do processo tradicional. Porém, ao invés de servir para desenvolver especificações de modo textual, os requisitos são utilizados para desenvolver um modelo executável. Os envolvidos no projeto utilizam este modelo para esclarecer e facilitar o entendimento dos requisitos e especificações. Utilizando ferramentas computacionais é possível simular o sistema, descobrindo falhas e defeitos muito antes da implementação. Com o modelo finalizado e verificado, é possível gerar automaticamente o código e refazer testes a partir destes códigos.

Este fluxo de trabalho permite que se permaneça sempre no mesmo ambiente de desenvolvimento, minimizando o volume de trabalho necessário. Além disso é possível iniciar os testes nas fases iniciais, já nos modelos recém projetados e verificar se os requisitos estão sendo alcançados. Como resultado, falhas são encontradas e corrigidas mais cedo do que no modelo de desenvolvimento tradicional, diminuindo o tempo de desenvolvimento total e reduzindo a utilização de recursos.

A Figura 6 ilustra o fluxo de trabalho facilitado do MBD. Todas as etapas se conectam com facilidade, permitindo que se acesse e modifique características do das fases iniciais de mesmo nas últimas etapas de implementação.

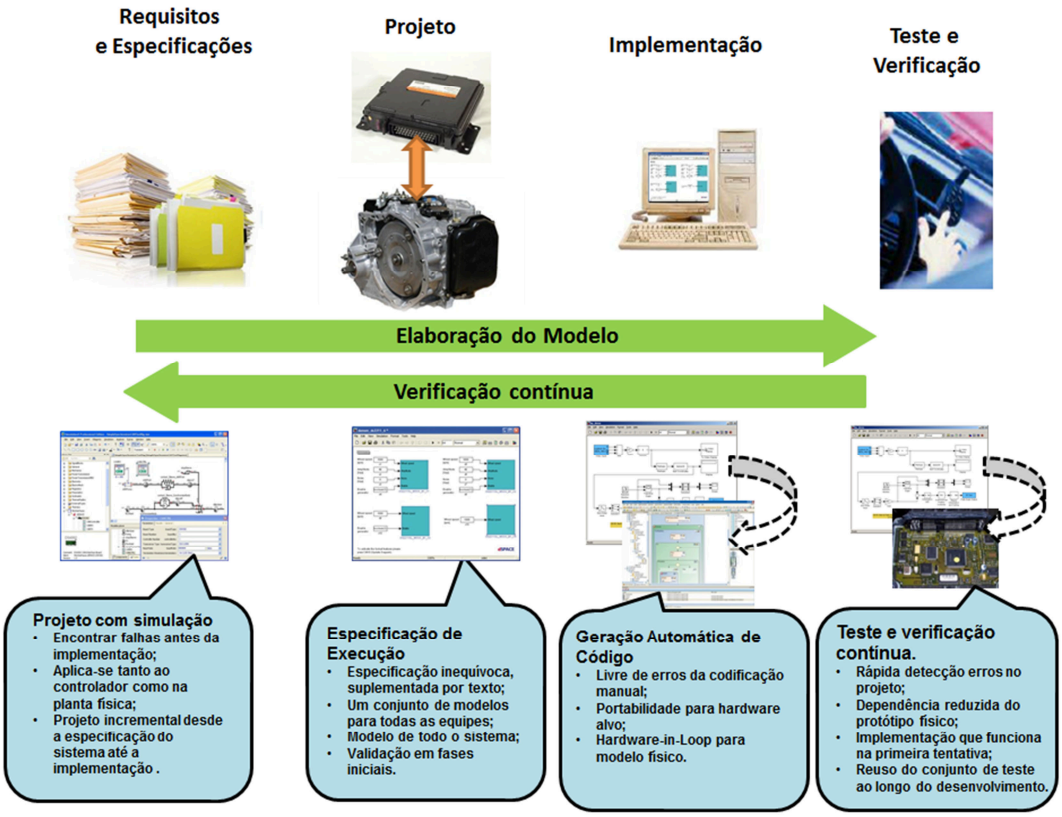


Figura 6: Desenvolvimento Baseado em Modelo (MBD)
Fonte: MathWorks, adaptado pelo autor (2015)

No MBD de sistema de controles, o desenvolvimento segue basicamente as mesmas etapas demonstradas no diagrama do *Modelo V*, abaixo listados:

a) Definição dos Requisitos;
b) Projeto da Arquitetura Funcional;
c) Projeto da Arquitetura Física;
d) Codificação;
e) Testes de Componente (Unidade);
f) Testes de Integração;
g) Verificação e Validação.

Inicialmente definem-se todos os requisitos necessários para o funcionamento do controle do sistema conforme esperado. Estes requisitos são descritos textualmente e permitem que se desenvolva a arquitetura funcional do sistema, que normalmente consiste em um modelo desenvolvido em ferramentas

computacionais. Após esta etapa é possível desenvolver fisicamente o modelo, construindo um protótipo do sistema a ser controlado. Com o protótipo pronto é possível, utilizando o modelo funcional gerar um código na linguagem desejada que será embarcado em outras fases em um *hardware* escolhido. Com o modelo desenvolvido em *software* e o código gerado é possível verificar o funcionamento do projeto utilizando entradas de teste já desenvolvidas. Esta verificação pode ser via *software* ou *hardware*. Com todas as funções verificadas é possível embarcar o código no sistema alvo final e validar o sistema como um todo.

2.4 MÉTODOS DE TESTES PARA MBD

No MDB, as tecnicas tradicionamente utilizadas para validação são: *Model-In-the-Loop* (MIL), *Software-In-the-Loop* (SIL), *Processor-In-the-Loop* (PIL) e *Hardware-In-the-Loop* (HIL). A utilização destes métodos geralmente segue cronologicamente esta sequência, conforme mostrado na Figura 7.

Figura 7: Sequência de métodos para verificação de sistema embarcado
Fonte: Elaborado pelo Autor (2015)

Neste trabalho são utilizados apenas os métodos de MIL, SIL e o PIL por não estarem disponíveis os hardwares (ECU final e o Emulador da planta) necessários para realizar o método HIL. A seguir são descritos um a um os principais métodos utilizados para o MBD, que podem muitas vezes serem encontradas outras denominações na bibliografia.

2.4.1 *Model-In-the-Loop* (MIL)

É o primeiro estágio de simulação que serve como referência para os estágios seguintes e nele estão contidos os valores mínimos e máximos das variáveis. A estratégia de controle e o modelo físico são de desenvolvimento extremamente rápido, possibilitando mudanças e testes rápidos no sistema. Neste

trabalho esta fase foi desenvolvida em ambiente *Matlab*/*Simulink*®. A Figura 8 ilustra o ambiente em que o MIL geralmente são aplicados. O modelo da planta e do controlador, assim como todos os testes para verificação são desenvolvidos em *Matlab*/*Simulink*®.

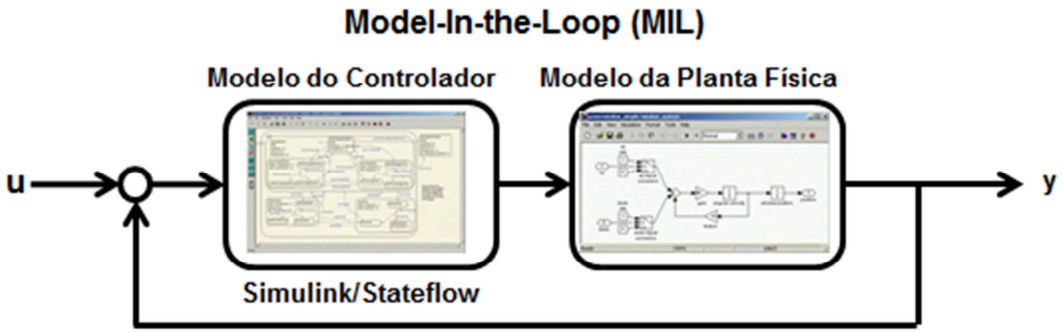


Figura 8: Fluxo de controle para o MIL mostrando telas genéricas dos programas utilizados
Fonte: Elaborado pelo Autor (2015)

2.4.2 *Software-In-the Loop* (SIL)

Esta simulação é o estágio onde a ferramenta de geração automática de código fornece a estratégia de controle estabelecida no MIL automaticamente em código em linguagem C ou C++. Aqui o modelo de controle é um pouco mais real. Este 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 MIL, mas já é possível enxergar erros de codificação. O fluxograma desta fase pode ser observado na Figura 9. Conforme a figura ilustra, o controlador é representado pelo código em linguagem C e a planta segue sendo representada por um diagrama de blocos de *Matlab*/*Simulink*®.

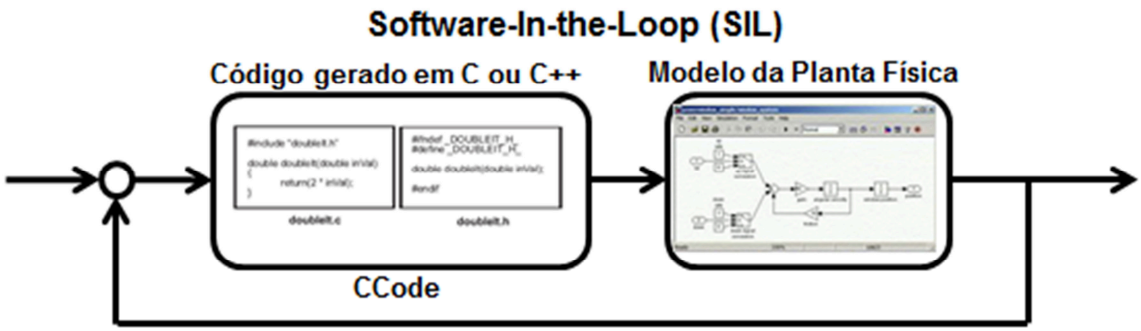


Figura 9: Fluxo de controle para o SIL demonstrando telas genéricas dos programas utilizados
Fonte: Elaborado pelo Autor (2015)

2.4.3 *Processor-In-the Loop* (PIL)

Nesta fase não se roda mais o código do sistema em simulação. Ao invés disso, ele é implementado em um micro controlador. Enquanto o código do controle está rodando no micro controlador, a planta ainda está sendo representada por um diagrama de *Matlab*/*Simulink*®. Este teste é projetado para expor se existem 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 10 ilustra as plataformas envolvidas nesta etapa.

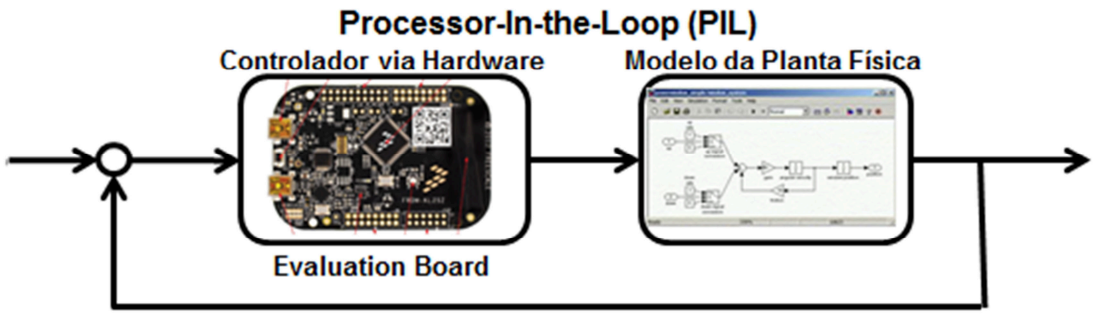


Figura 10: Fluxo de controle para o PIL demonstrando telas genéricas dos programas utilizados e imagens ilustrativas das plataformas
Fonte: Elaborado pelo Autor (2015)

2.4.4 *Hardware-In-the Loop* (HIL)

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 está rodando em um computador em tempo real com entradas e saídas simuladas para fazer com que o controlador acredite que está instalado na planta real. Neste caso a única diferença entre a aplicação final e o ambiente HIL é a fidelidade do modelo da planta e os vetores teste que estão sendo utilizados. O HIL é geralmente utilizado para validação de *software* ao invés de desenvolvimento, pois a interação do modelo é muito lenta. Apesar disso, 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 HIL é ilustrada pela Figura 11.

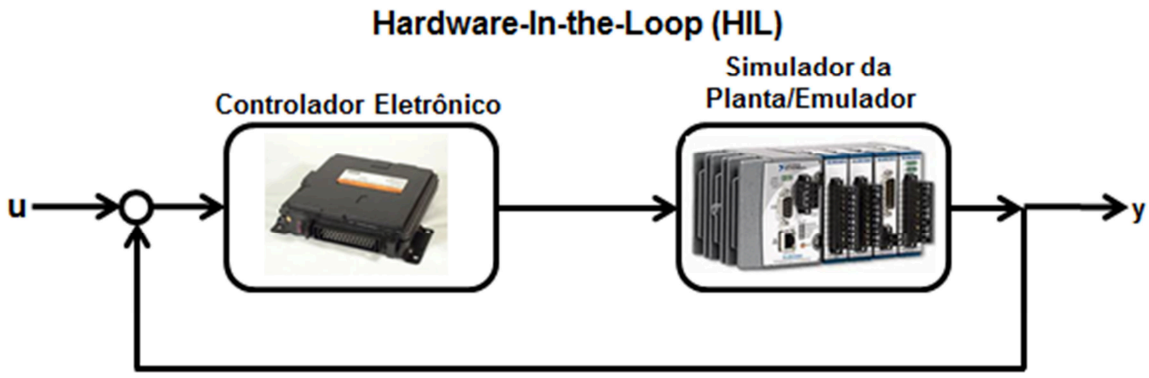


Figura 11: Fluxo de controle para o HIL demonstrando telas genéricas dos programas utilizados e imagens ilustrativas das plataformas
Fonte: Elaborado pelo Autor (2015)

2.5 RAPID CONTROL PROTOTYPING (RCP)

Apesar de não ser um método do MBD, pois o MBD é usado para desenvolver novas funções, o *Rapid Control Prototyping* (RCP) é um importante e valioso método de validação para *softwares* automotivos, e que será também utilizado neste trabalho.

O principal desejo de engenheiros que trabalham com projetos automobilísticos é poder se concentrar completamente em funções das ECU e executar rapidamente testes em seus sistemas alvo. Não apenas testes *off-line*, como também em veículos reais ou em bancos de teste. Neste cenário é preciso de uma rápida confirmação de que o projeto está indo no caminho certo.

O RCP é um processo de calibração dos algoritmos de controle em um protótipo de *hardware*. Este processo permite que testes sejam feitos antes que haja uma ECU disponível para o processo.

O RCP permite aos engenheiros testar e interagir suas estratégias de controle rapidamente. Consequentemente, as funções de controle são importadas automaticamente com *Matlab*/*Simulink*® em uma máquina em tempo real com interfaces físicas de entradas e saídas para se conectar aos sistemas do mundo real.

Além disso, esse processo diminui o tempo de desenvolvimento, permitindo que correções sejam feitas no início do processo produto. Essa oportunidade de conseguir verificar o processo em fases iniciais do projeto permite que erros sejam corrigidos e mudanças serem feitas enquanto o custo ainda é pequeno.

Segundo Dong (2008), no RCP as plantas e o controlador são implementados em uma placa de processamento de sinal digital, permitindo um fácil ajuste de vários parâmetros da planta e do controlador.

De acordo com Fang *et al.*(2009) o principal conceito do RCP é entender os requisitos do sistema e traçar um plano geral de desenvolvimento, levando em consideração a eficiência e rapidez, e então selecionar ou desenvolver um projeto do modelo de controlador e uma plataforma de *hardware*.

Seguindo esses requisitos, engenheiros podem otimizar os projetos até que o resultado seja satisfatório. Falhas de projeto podem ser encontradas e as correções podem ser realizadas imediatamente, visando a eficiência de tempo e a confiabilidade do sistema.

Cada vez mais empresas estão oferecendo soluções para o desenvolvimento de RCP. O *software* da *MathWorks*®, *Matlab*/*Simulink*® é talvez a ferramenta mais conhecida e amplamente usada para modelagem e simulação de sistemas. O *hardware MicroAutoBox*® (MABX) modelo DS1401 com os módulos DS1505 e DS1507 da *dSpace*®, que pode ser visto na Figura 12 e seu diagrama de de blocos na Figura 13, é uma ferramenta poderosa, com diversas entradas e saídas digitais e analógicas que permitem a simulação de sistemas complexos. Usando *Matlab*/*Simulink*® com o *MicroAutoBox*® da *dSpace*® pode-se facilmente realizar testes para validar qualquer modelo.

Os blocos e diagramas de estado do *Matlab*/*Simulink*® e *Stateflow*® são os pontos de partida para o desenvolvimento da função para prototipagem. Segundo Fang *et al* (2009) eles representam o controlador e o seu ambiente de forma gráfica e podem ser simulados *off-line* para uma validação inicial e não minuciosa. Para realizar o RCP, os diagramas de bloco criados em ambiente *Matlab*/*Simulink*® são implementados diretamente em um sistema de prototipagem e rodados em tempo real, sem a necessidade de qualquer programação manual. O *hardware* de prototipagem geralmente possui um poder de processamento maior do que o ECU real, portanto, não há necessidade de se preocupar com limitações de *hardware*.

Figura 12: Plataforma de *hardware MicroAutoBox*® modelo DS1401 com os módulos DS1505 e DS1507, da *dSpace*®
Fonte: *dSpace*® (2015)

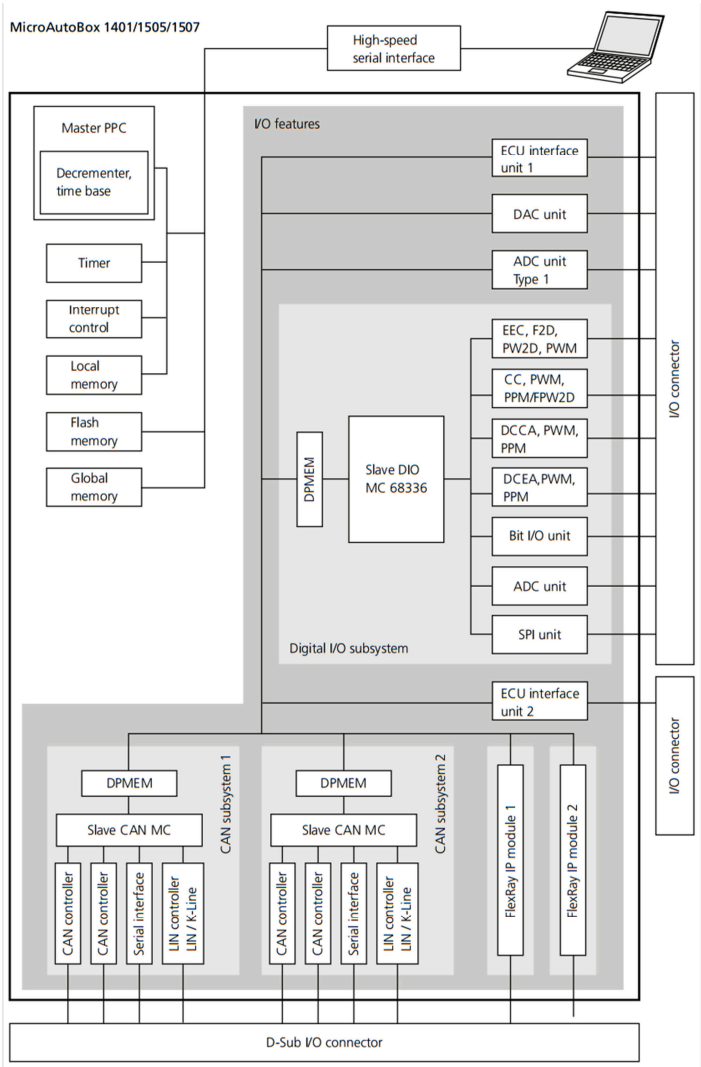


Figura 13: Diagrama do blocos da plataforma de *hardware MicroAutoBox*®, da *dSpace*®
Fonte: *dSpace*® (2015)

Neste ambiente de prototipagem rápida, o desenvolvimento pode avançar muito mais rápido, já que o profissional responsável pelo design de controle pode facilmente executar funções que, em um ambiente de desenvolvimento tradicional,

normalmente seria executado por três departamentos. A Figura 14 compara os procedimentos do RCP com os procedimentos tradicionais.

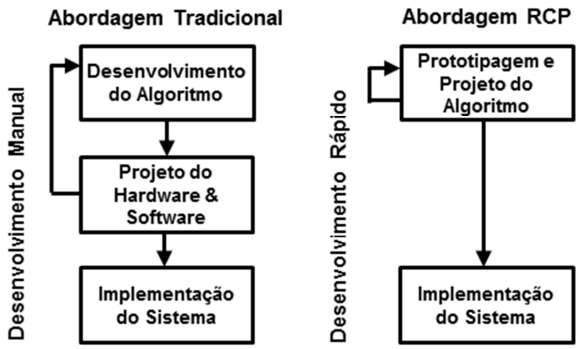


Figura 14: Comparação do desenvolvimento tradicional com o desenvolvimento do *Rapid Control Prototyping*
Fonte: Elaborado pelo Autor (2015)

Atualmente os *softwares* controle automobilístico são submetidos a quatro etapas de verificação (Figura 15), a saber:

a) Simulação *Offline* (como MIL).
b) Simulação RCP.
c) Simulação HIL.
d) Validação na ECU.

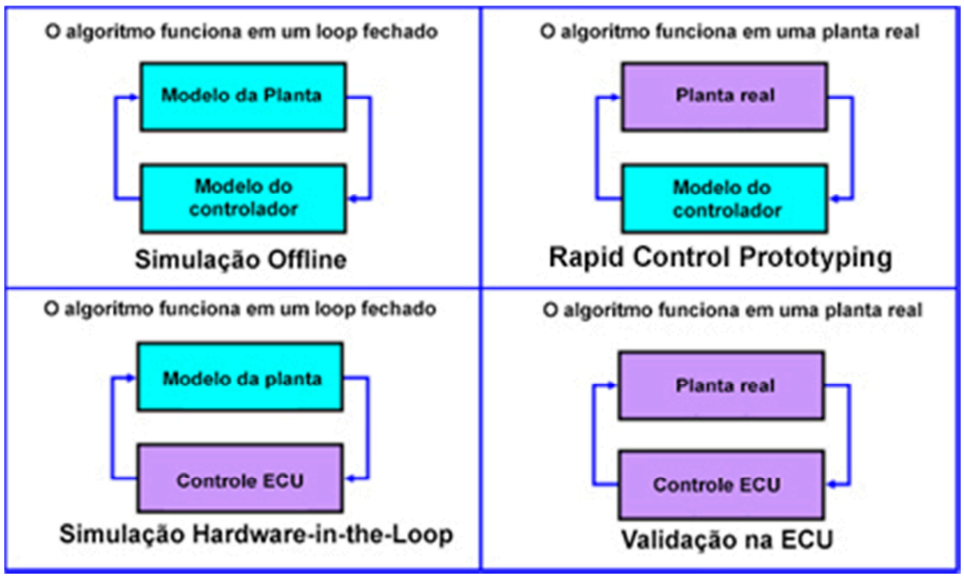


Figura 15: Estratégia de controle e estágios de verificação
Fonte: NEME (2015)

As duas primeiras fases de validação destinam-se a verificar o desempenho funcional do algoritmo de controle. Mudanças no modelo de função podem, portanto, ser feita de forma rápida, e transferidos para o sistema de prototipagem através da geração automática de código em praticamente com um clique. Este procedimento torma o tempo de transição o mais curto possível. As últimas duas fases de validação são executadas em *software* e embarcada na ECU. Parâmetros das funções controle podem ser alterados e os sinais podem ser capturados em tempo real pela ferramenta ControlDesk da dSPACE. Portanto, cada variável interna no controlador pode ser monitorada pelo ControlDesk, que projetou uma interface que permite ao usuário e outros aplicativos vê-las. Na Figura 16 a ilustra forma do foco principal para o modelo com blocos MABX, onde desmonstramos que a MABX atua no sistema de controle, onde suas entradas recebe os sinais do sensores (*Input*) do veiculo, realiza o processamento (*RAM*, *ROM* e *CPU*) e entrega os resultados (*Output*) ao veiculo.

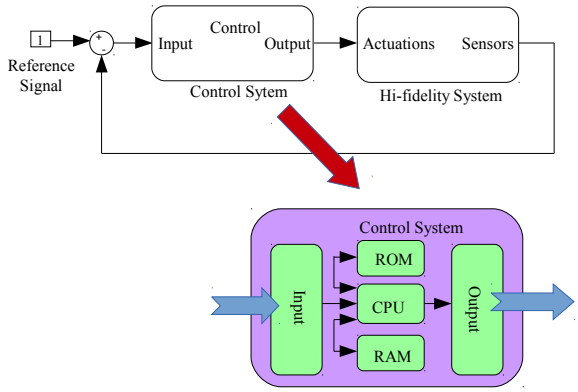


Figura 16: Diagrama de controlador projetado em Simulink para o hardware RCP.
Fonte: Elaborado pelo Autor (2015)

Segundo a dSPACE (2006) existem duas abordagem básicas para RCP que podem ser distintas: *fullpass* e *bypass*.

Com *fullpass*, o sistema da prototipagem substitui completamente a ECU existente. Todos os sensores e atuadores são conectados a MABXe as funções são executadas em tempo real, e tem plena autonomia para controlar a planta física (Figura 17).

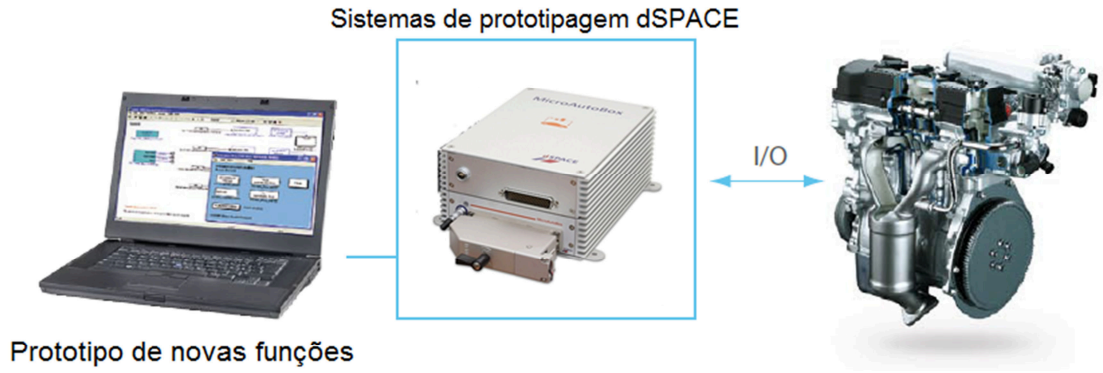


Figura 17: Fullpass, o sistema prototipagem substitui completamente a ECU existente.
Fonte: Elaborado pelo Autor (2015)

No *bypass*, o sistema de prototipagem pode ser conectado à alguns sensores e atuadores, e trabalha em paralelo com a ECU existente, apenas parte das tarefas de controle é feito por ele, conforme ilustrado na Figura 18.

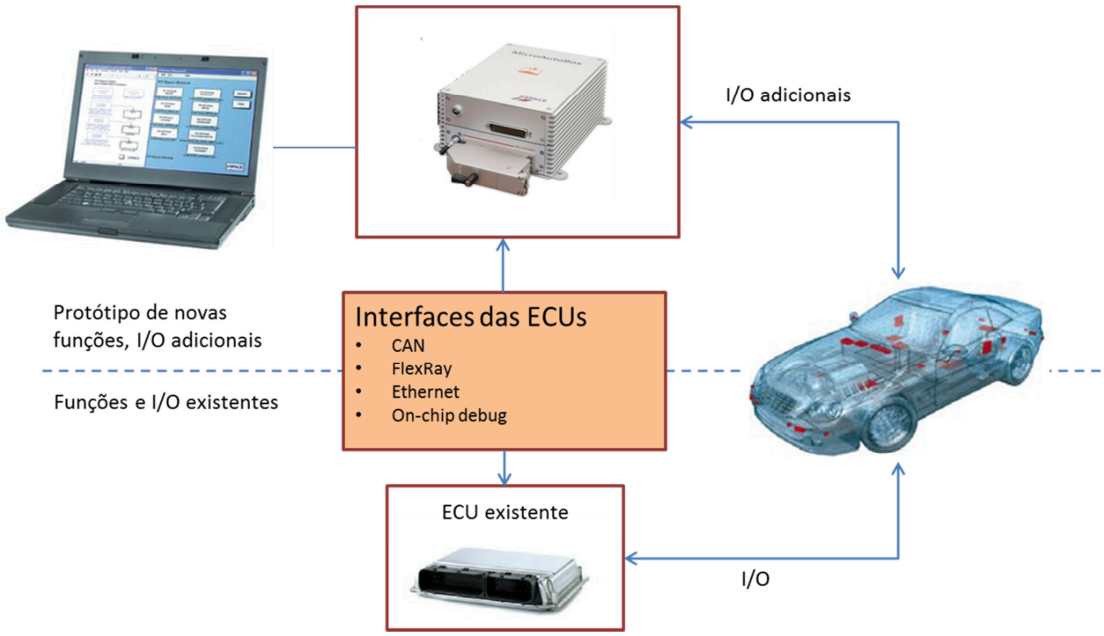


Figura 18: Bypass, novas funções rodam no sistema de prototipagem, enquanto algoritmos sem mudanças continuam na ECU existente.
Fonte: Elaborado pelo Autor (2015)

Para exemplificar este, considere que se deseja desenvolver uma nova função de *software* ou melhora a já existente. O *hardware* de prototipagem que suportará esta nova função em tempo real é colocado em paralelo à ECU, que possui a função original. Esta é desabilitada no ECU, fazendo com que o fluxo de controle para a nova função seja transferido para o controlador de prototipagem

ligado a ele, como apresentado na Figura 19. Isto possibilita de forma rápida testar, avaliar, otimizar e calibrar novas funções.

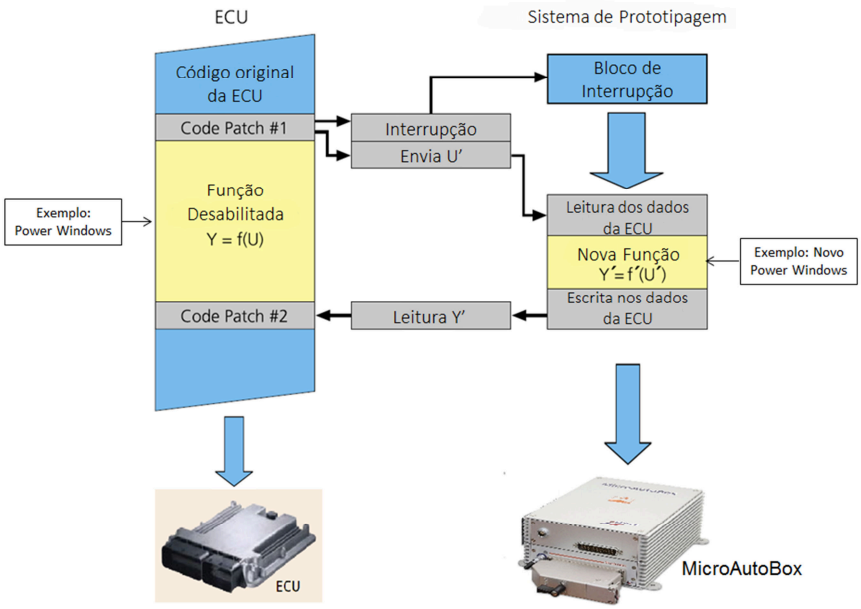


Figura 19: B*ypass*, exemplo de função desviada para o MABX usado no RCP.
Fonte: Elaborado pelo Autor (2015)

O método de RCP desenvolvido no presente trabalho foi o de *fullpass,* que utiliza um sistema de prototipagem para substituir a ECU de maneira completa, ao contrário do modo *bypass* que é utilizado para substituir apenas partes individuais da ECU.

3 DESENVOLVIMENTO BASEADO EM MODELOS: MIL, SIL E PIL

Este trabalho foi fundamentado na metodologia MDB, aplicada ao modelamento de duas funções: sistemas de iluminação exterior (*Exterior Light System* - ELS) e painel do instrumento (*Instrument Cluster* – IC). Esta metodologia organiza-se nas etapas MIL, SIL e PIL, sendo que em cada uma destas realizam-se testes até o desenvolvimento. Também foram desenvolvidos pelo método RCP para as funções do sistema de iluminação externa e do sistema de vidro elétrico (*Power Window System* – PWS).

Como não consideramos as funções desenvolvidas vitais à segurança do veículo, não foi especificado nenhum requisito de segurança funcional. A seguir podemos observar a Tabela 1, que aponta os testes realizados no presente trabalho, para cada uma das funções e onde estão localizados no texto.

Tabela 1: Tabela das etapas desenvolvidas para cada função.

Método / Função	MIL	SIL	PIL	HIL	RCP
Sistema de iluminação externa	✓ Sessão 3.2.2	✓ Sessão 3.2.3	✓ Sessão 3.4.2	✗	✓ Sessão 4.1
Painel de instrumentos	✓ Sessão 3.3.2	✓ Sessão 3.3.3	✓ Sessão 3.4.3	✗	✗
Sistema de vidro elétrico	✗	✗	✗	✗	✓ Sessão 4.2

Fonte: Elaborado pelo autor (2015).

Na etapa MIL, foi realizado o modelamento do controlador, levando em consideração os requisitos da função. Na validação, utilizou-se a simulação para confirmar o funcionamento e validar as características. A ferramenta *Matlab*/*Simulink*® foi utilizada para etapa. Na etapa SIL foi gerar código linguagem "C" equivalente ao controlador modelado na etapa MIL. Novamente recorre-se ao *Matlab*/*Simulink*® para gerar o código automaticamente e realizar a simulação de validação desta etapa.

A plataforma escolhida para os testes na etapa PIL foi uma ferramenta de FRDM-KL25Z desenvolvimento *Freedom Board* projetada pela Freescale, em especial a.

Segundo a Freescale (2015) a FRDM-KL25Z é uma plataforma de desenvolvimento de baixíssimo custo para as MCUs da série Kinetis L (KL1x e KL2x), que são construídas com um processador ARM Cortex™-M0+. As características incluem fácil acesso aos I/O do MCU, vem com possibilidade de uso com bateria, baixo consumo de energia, um fator de forma baseado em padrão com a opção de placa de expansão e uma interface de debug interna para programação da memória Flash e controle de execução. A FRDM-KL25Z é suportada por vários *softwares* de desenvolvimento da Freescale e de terceiros.

Outras características são:

- MCU MKL25Z128VLK4 – 48 MHz, 128 KB flash, 16 KB SRAM, USB OTG (FS), 80LQFP;
- Sensor de toque capacitivo, acelerômetro MMA8451Q, LED RGB;
- Sofisticada interface de debug OpenSDA;
- Interface de programação flash por dispositivo de armazenamento de massa.

Na Figura 20 mostramos o diagrama em blocos do MCU Kinetis KL25Z.

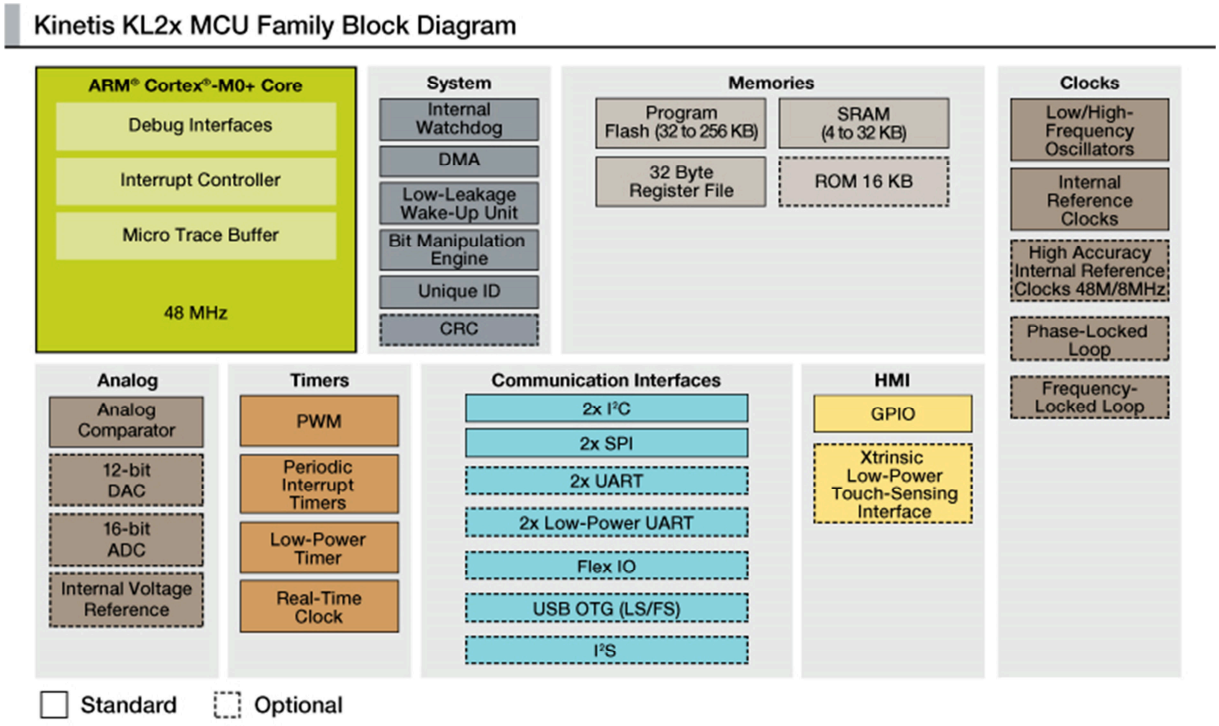


Figura 20: Diagrama em blocos do MCU *Kinetis* KL25Z.
Fonte: Freescale (2015)

Nela está disponível um processador ARM Cortex M0+, tendo uma frequência de *clock* de 48MHz, memória 128KB Flash, com sistemas de gravação *on-board* (*OpenSDA*). Tendo vários periféricos como interfaces de entrada e saída

(I/Os) analógicos e digitais, interfaces seriais como *Serial Peripheral Interface* (SPI) e *Serial Communication Interface* (SCI), Módulo de base de tempo (*Time Base Module* - TBM), interfaces com modulação (*Pulse Width Module* – PWM). Também está presente na placa LED RGB, acelerômetro e USB. O principal fator da escolha deste equipamento foi a disponibilidade de bibliotecas, para o *hardware*, em *Matlab*/*Simulink*®. Foi necessário também o desenvolvimento de bibliotecas para o controle dos servos motores dos mostradores na função Painel de Instrumentos as quais serão apresentadas na seção 3.4.3.

A Figura 21 apresenta uma foto da plataforma FRDM_KL25Z.

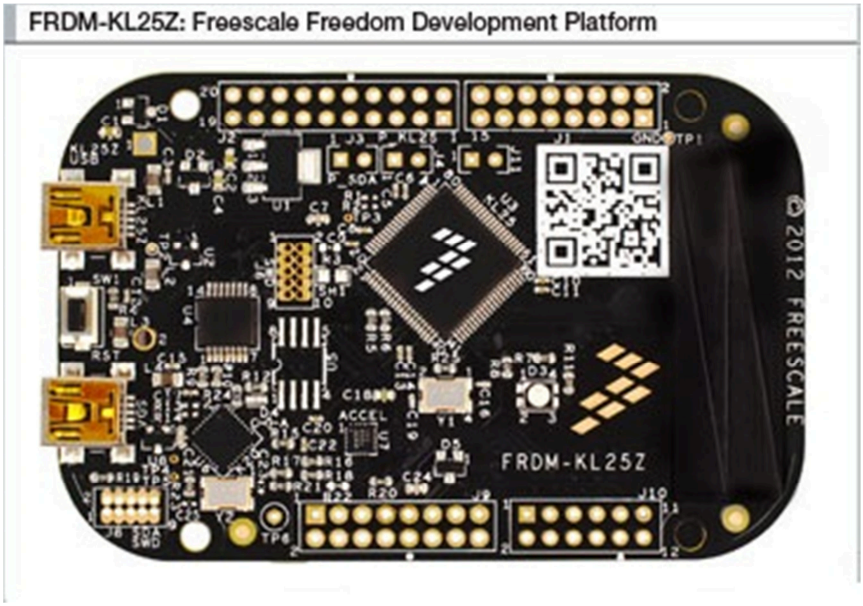


Figura 21: Imagem da plataforma FRDM-KL25Z.
Fonte: Freescale (2015)

Antes de embarcar o *software*, foram incluídos os blocos de entradas e saída para esta placa, visando gerar código para o controlador interagir com o mundo externo, complementando o código previamente gerado nas etapas MIL e SIL.

Para geração dos sinais externos de validação foram desenvolvidas função em *Labview*®. Esta sequência é ilustrada na Figura 22.

Figura 22: Sequência de desenvolvimento deste trabalho e ferramentas utilizadas durante o processo
Fonte: Elaborado pelo Autor (2015)

Nos estudos foram utilizadas as etapas MIL, SIL, PIL e RCP. A etapa HIL não será desenvolvida devido à falta de disponibilidade da ECU e do hardware que emula a planta física.

Foram executadas, para cada função, a sequência de atividades a seguir de acordo com o proposto pela *MathWorks*®.

a) Definição de requisitos;
b) Projeto da Planta Física;
c) Projeto do Controlador;
d) Definição de Testes de Caso
e) Validação;

Neste trabalho apresentado-se o desenvolvimento das funções e todos os processos envolvidos. Nas seção 3.2 e 3.3 serão expostos estudos realizados nas fases MIL e SIL das funções do sistemas de iluminação exterior e do painel de instrumento, passando pelas características, requisitos, modelamento, validação e teste. Nas seções 3.4.2 e 3.4.3 são descritas a fase PIL, transferência do código, validação e comparação dos resultados.

3.1 ARQUITETURA DAS ECUs AUTOMOTIVAS

No projeto das funções levou-se em conta o uso de veículos de baixo custo, onde este pode ter no mínimo três ECUs, sendo elas: ECM, BCM e ICM, mostradas na Figura 23.

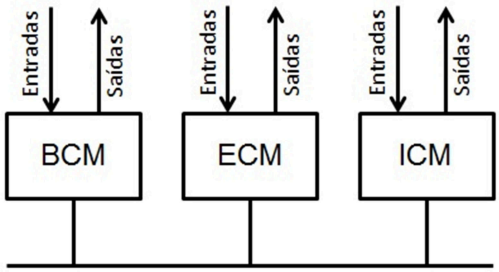


Figura 23: Distribuição básica das ECUs automotivas.
Fonte: Elaborado pelo autor (2015).

Da Figura 24 a Figura 26 ilustram o diagrama de blocos das ECUs, com as principais funções por elas controladas, e entre elas, destacam as funções que foram desenvolvidas nesse trabalho.

Responsável pelas funções do painel de instrumentos (ICM), a Figura 24 mostra o diagrama de blocos da ECU e em destaque as funções desenvolvidas.

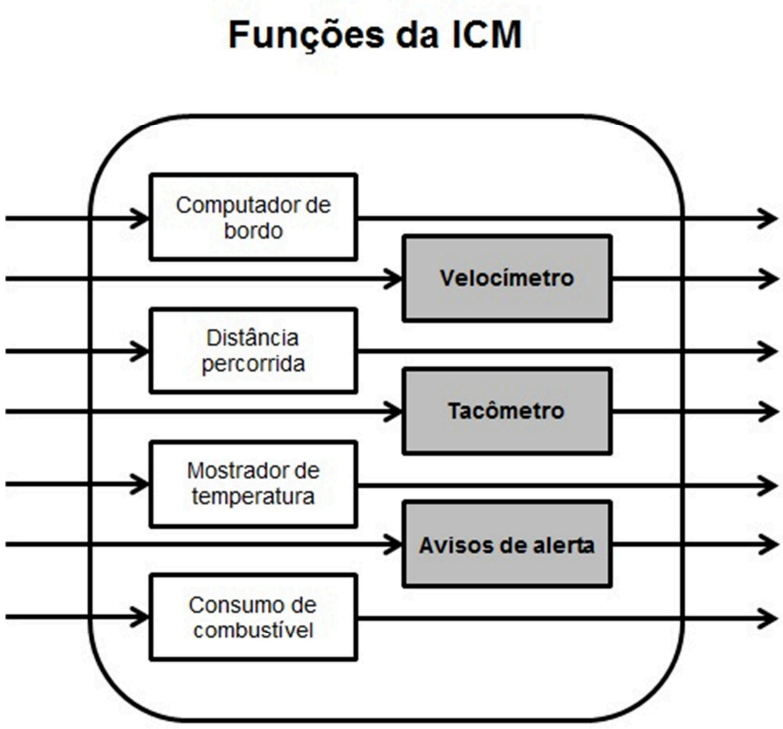


Figura 24: Diagrama de blocos de uma ICM
Fonte: Elaborado pelo Autor (2015)

Mostrada na Figura 25, a ECM controla as funções de *Power train*, neste trabalho não foi estuda nenhuma função desta ECU, estando ilustrada apenas como parte da arquitetura básica do veículo.

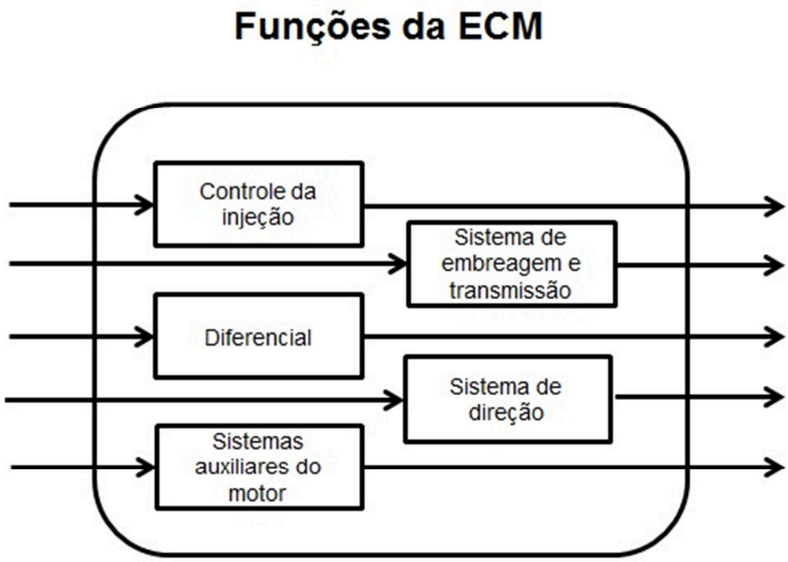


Figura 25: Diagrama de blocos de uma ECM
Fonte: Elaborado pelo Autor (2015)

Na Figura 26 temos BCM, onde estão às funções de carroceria e duas destas funções serão desenvolvidas (ELS e PWS).

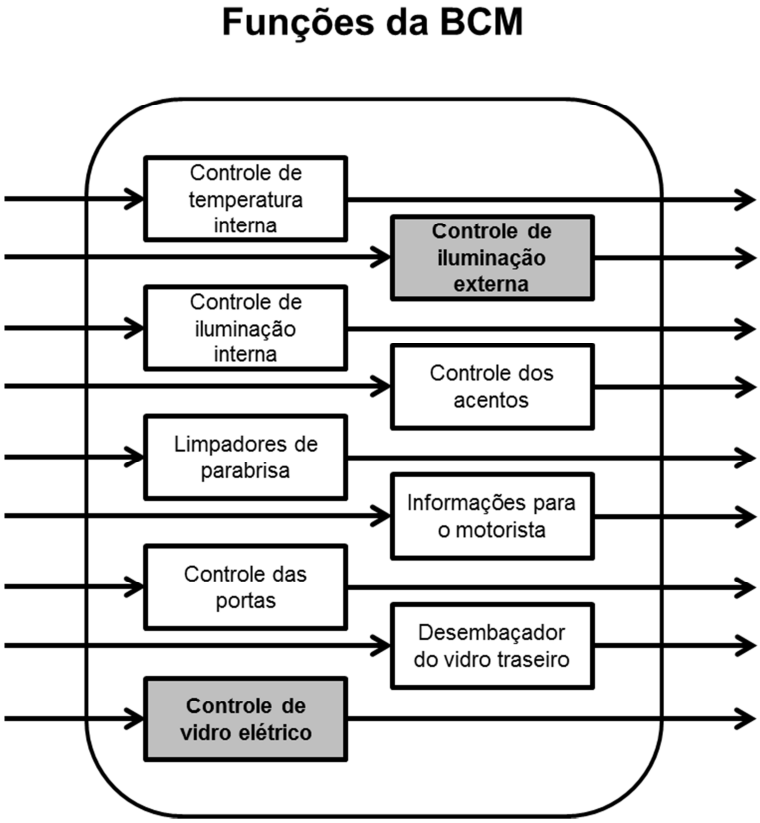


Figura 26: Diagrama de blocos de uma BCM
Fonte: Elaborado pelo Autor (2015)

A seguir será apresentado o desenvolvimento da função iluminação externa presente nas funções do BCM

3.2 PROJETO DA FUNÇÃO DO SISTEMA DE ILUMINAÇÃO EXTERNA AUTOMOTIVA (ELS)

O sistema de iluminação exterior de um automóvel é constituído por dispositivos montados ou integrados em várias partes de um veículo. Estas podem incluir: a frente, as laterais, a traseira e, em alguns casos, a parte superior do veículo. Isso permite a operação segura do veículo aumentando a visibilidade para o motorista. Este sistema também facilita a visualização do veículo por terceiros, indicando sua localização, dimensões e a direção do deslocamento, assim como as intenções de manobras do motorista.

3.2.1 Controle Eletrônico das Funções de Iluminação Externa

As partes eletrônicas de um veículo, desde o sistema de iluminação do carro até as fechaduras das portas, podem ser controladas por módulos. O módulo de controle das funções de carroceria, conhecido como *Body Control Module* (BCM) executa uma função muito complexa e utiliza uma grande gama de dispositivos de entrada e saída.

Segundo Liuguilong (2013) na eletrônica automotiva, a BCM é o principal componente de uma ampla gama de funções que interagem com o usuario. Pode compará-lo com o sistema nervoso central de um ser humano, onde ele executa a coordenação do corpo humano através da transmissão de sinais, assim com, o BCM regula a atividade de coordenação entre as diferentes partes de um automóvel também através de sinais.

A centralização das funções na BCM viabiliza a implementação de atuações complexas utilizando acionadores intuitivos para o usuário. No entanto essa centralização traz alguns efeitos não desejados, tal como o mau funcionamento da BCM causando problemas intermitentes em várias partes do veículo, e em certos casos, estes sistemas podem até mesmo deixar de funcionar inteiramente.

Nos sistemas atuais de iluminação externa automotiva os principais requisitos se baseiam nos seguintes elementos:

- A posição dos interruptores de acionamento de cada um dos casos de iluminação seja ela frontal, de indicação de direção, de freio ou de ré;
- Nível de tensão fornecida pela bateria;
- Posição da chave de ignição (que possui alteração em sentido horário) com as seguintes posições: *off*, acessório (*acc*), *run* e *crank*.

3.2.2 Model-In-the-Loop para um Sistema de Iluminação Externa Automotiva

Neste trabalho, são analisadas apenas as luzes mais tradicionais: luzes laterais, de nevoeiro e de aparência não estão presentes nos estudos realizados. O conjunto de luzes analisadas, agrupadas por dianteiras e traseiras, são:

- **Luzes dianteiras:** luzes indicadoras de direção (esquerda e direita), luz baixa, luz alta e luzes de posição;
- **Luzes traseiras**: luz de ré, luz de freio, luzes indicadoras de direção (esquerda e direita) e luzes de posição.

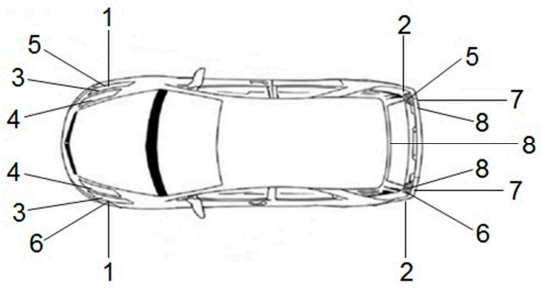
As descrições de cada uma das funções do sistema externo de iluminação segundo o Código de Trânsito Brasileiro no Anexo 1 da Lei 9.503 de 1997, estão descritas abaixo:

- **Luz de posição:** é a "luz do veículo destinada a indicar a presença e a largura do veículo" (Lei 9.503 de 1997, Anexo 1). A luz de posição é obrigatória à noite, quando o veículo estiver parado para carga e descarga de passageiros ou mercadorias e no caso de neblina, chuva forte, ou cerração (Lei 9.503 de 1997, artigo 40, incisos IV e VII). Para ativar as luzes de posição são necessárias algumas condições: a posição da ignição deve estar em “Acessório” ou posterior e o interruptor de luz em “luzes de posição”. Caso a chave de ignição for girada no sentido horário as luzes de posição continuarão acesas, só sendo desativadas caso a chave seja girada no sentido anti-horário para a posição *off*.
- **Luz baixa:** é "o facho de luz do veículo destinada a iluminar a via diante do veículo, sem ocasionar ofuscamento ou incômodo injustificável aos condutores e outros usuários da via que venham em sentido contrário" (Lei 9.503 de 1997, Anexo 1). Ela é obrigatória em dois casos; à noite em qualquer situação e de dia em túneis providos de iluminação pública (Lei 9.503 de 1997, artigo 40, inciso I). Na maioria dos sistemas de iluminação de carros este isso é implementado com a ignição em qualquer posição diferente de *off* e o interruptor de luz em “luz baixa”.
- **Luz alta:** é "o facho de luz do veículo destinado a iluminar a via até uma grande distância do veículo" (Lei 9.503 de 1997, Anexo 1). É mandatório em vias não iluminadas quando não houver outro veículo vindo na direção contrária ou à frente (Lei 9.503 de 1997, artigo 40, inciso II). A luz alta será

ativada quando a posição da ignição estiver em “Acessório” ou posterior e o interruptor estiver em “luz alta”.

- **Luz indicadora de direção:** é a "luz do veículo destinada a indicar aos demais usuários da via que o condutor tem o propósito de mudar de direção para a direita ou para a esquerda" (Lei 9.503 de 1997, Anexo 1). Há duas maneiras de ativar luz indicadora com a ignição em qualquer posição exceto *off* e o interruptor da luz indicadora na posição para cima ou para baixo. Se o interruptor estiver para cima, as luzes de indicação (frontal e traseira) do lado direito serão acesas intermitentemente. Já se o interruptor estiver para baixo, as luzes de indicação (frontal e traseira) do lado esquerdo serão ativadas intermitentemente. Para sinalizar situações de emergência, aciona-se o botão de “alerta” onde todas as luzes de indicação serão ativadas intermitentemente de forma sincronizada, independentemente da posição da chave de ignição.
- **Luz de marcha ré:** é "a luz do veículo destinada a iluminar atrás do veículo e advertir aos demais usuários da via que o veículo está efetuando ou a ponto de efetuar uma manobra de marcha à ré" (Lei 9.503 de 1997, Anexo 1). A luz de marcha ré é ativada sempre que a marcha ré for engrenada e a ignição se encontrar em qualquer posição, exceto *off*.
- **Luz de freio:** é "a luz do veículo destinada a indicar aos demais usuários da via, que se encontram atrás do veículo, que o condutor está aplicando o freio de serviço" (Lei 9.503 de 1997, Anexo 1). A luz de freio é ativada quando o motorista pisar no pedal de freio e a ignição se encontrar em qualquer posição, exceto *off*.

O arranjo de iluminação de um veículo é constituído de sinalizações luminosas (lâmpadas, LED), distribuído conforme apresentada na Figura 27.



1. Luz de posição frontal;
2. Luz de posição traseira;
3. Luz baixa;
4. Luz alta
5. Luz indicadora de posição direita;
6. Luz indicadora de posição esquerda;
7. Luz de ré;
8. Luz de freio

Figura 27: Sistema de iluminação externo automotivo
Fonte: Elaborado pelo Autor (2015)

a) **Luz de posição:** composta de quatro lâmpadas próximas das extremidades laterais da parte frontal e posterior do veículo. São lâmpadas de pouca intensidade luminosas, de cor branca na frente e vermelha na traseira, e com a função de fornecer a outros noção da posição e dimensões do veículo.

b) **Luz baixa:** composta de duas lâmpadas na parte frontal do veículo, de cor branca, com intensidade maior, usada para o condutor visualizar a via sem prejudicar a visão dos outros.

c) **Luz alta:** composta de duas lâmpadas, próxima a luz baixa, de cor branca com semelhante a luz baixa, usada para o condutor uma longa distância da via.

d) **Luz indicadora de direção:** composta de quatro lâmpadas, sendo duas frontais e duas traseiras, de cor amarela, com a função de indicar mudança de direção do veículo ou alguma situação de alerta.

e) **Luz de ré:** composta por uma ou duas lâmpadas na traseira do veículo, de cor branca, para iluminação e indicação de manobra em marcha ré.

f) **Luz de freio:** composta por duas lâmpadas na parte traseira do veículo, de cor vermelha, que indica o acionamento a sistema de freio e consequentemente a mudança de velocidade.

3.2.2.1 Definição dos requisitos da função sistema de iluminação externa automotiva

Para o desenvolvimento do projeto do sitema de iluminação externo automotivo através da metodologia MBD, primeiramente elegem-se os requisitos de projeto.

Para o caso proposto, os 16 requisitos a serem considerados são apresentados na Tabela 2.

Tabela 2: Definição dos requisitos de funcionamento.

Para ativar as luzes de posição:

- **REQ-01**: a posição da ignição deve estar em *accessory*, *run* ou *crank*.
- **REQ-02**: o interruptor de luz deve estar em "luzes de posição".

Para ativar a luz baixa:

- **REQ-03**: a posição da ignição deve estar em *accessory*, *run* ou *crank*.
- **REQ-04**: o interruptor de luz deve estar em "luz baixa".

Para ativar a luz alta:

- **REQ-05**: a posição da ignição deve estar *accessory, run* ou c*rank*.
- **REQ-06**: o interruptor completa do feixe deve ser puxado.

Para ativar as luzes de indicação de posição esquerda:

- **REQ-07**: a posição da ignição deve estar em *accessory*, *run* ou *crank*.
- **REQ-08**: o interruptor da luz de indicação de posição deve ser direcionado para baixo.

Para ativar as luzes de indicação de posição direita:

- **REQ-09**: a posição da ignição deve estar em *accessory*, *run* ou *crank*.
- **REQ-10**: o interruptor da luz de indicação de posição deve ser direcionado para cima.

Para ativar as duas luzes indicadoras de posição ao mesmo tempo:

- **REQ-11**: o interruptor de alerta deve ser pressionado.

Para ativar a luz de ré:

- **REQ-12**: a posição da ignição deve estar em *accessory*, *run* ou *crank*.
- **REQ-13**: a marcha ré deve ser engatada.

Para ativar luzes de freio:

- **REQ-14**: o pedal do freio deve ser pressionado.

Para todos os casos:

- **REQ-15**: o nível de tensão da bateria deve estar entre 9 e 16 Volts.
- **REQ-16**: gerar sinal de erro independente quando as lâmpadas estiverem acionadas e com mal funcionamento.

Fonte: Elaborado pelo Autor (2015)

3.2.2.2 Projeto da função sistema de iluminação externa automotiva

Após a definição dos requisitos é possível desenvolver o controlador sistema desejado. A arquitetura em questão é modelada utilizando-se blocos de *Simulink*®/*Matlab*®. O diagrama principal do sistema, onde são mapeados todas as entradas e saída, é apresentado na Figura 28.

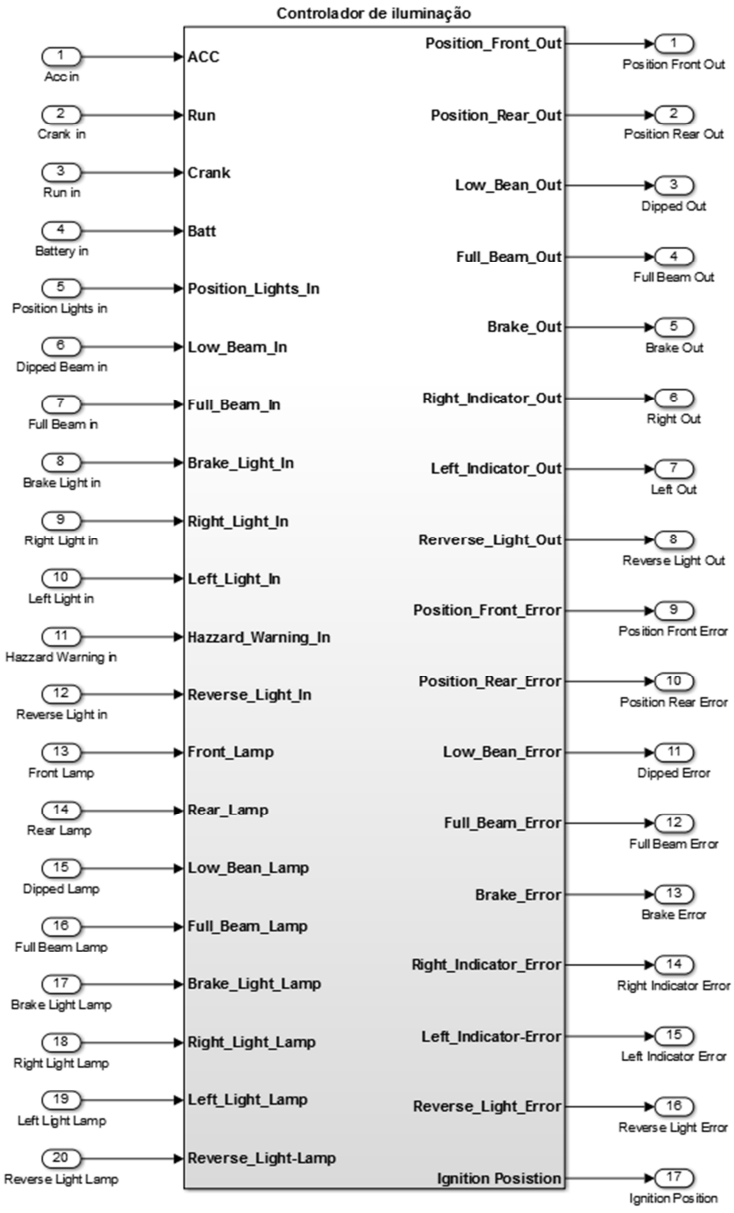


Figura 28: Diagrama *Simulink*® para o sistema do controlador da função ELS
Fonte: Elaborado pelo Autor (2015)

A Figura 29 apresenta o diagrama do principal expandido, ilustrando os blocos dos subsistemas para a função de iluminação propostas neste estudo de

caso. Esse diagrama também representa as conexões dos sinais de entrada e saída dos subsistemas.

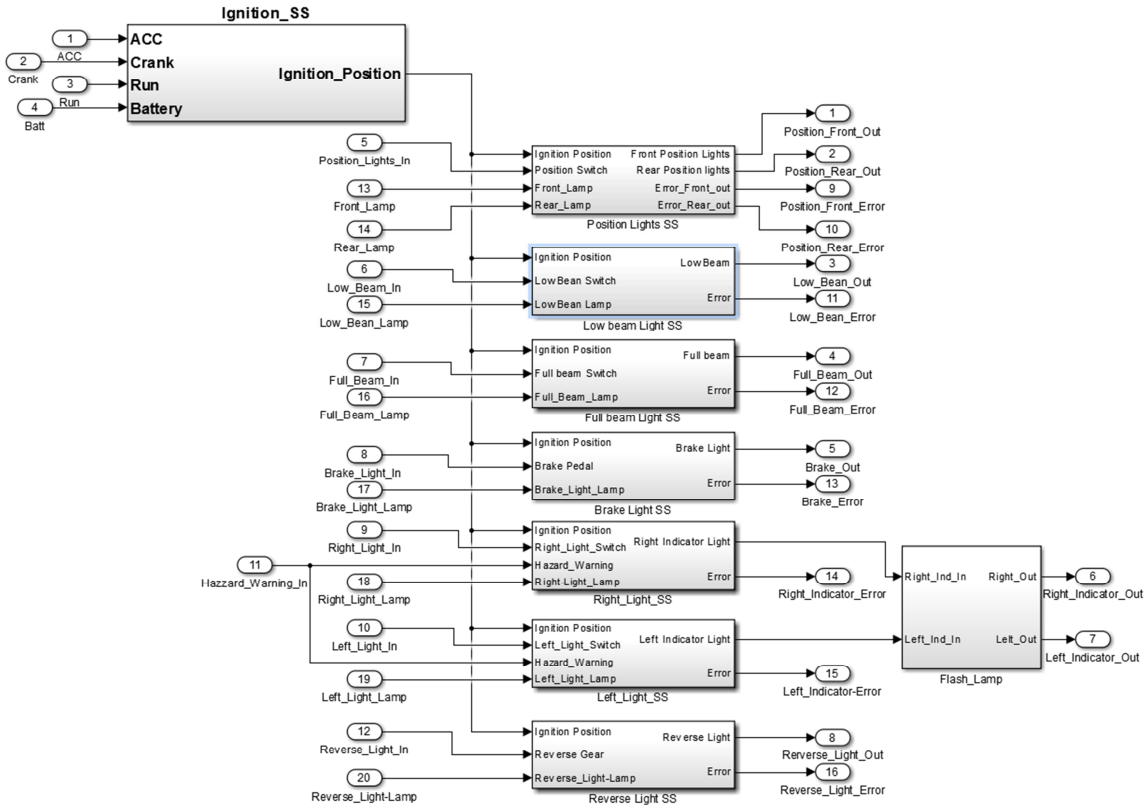


Figura 29: Diagrama *Matlab*/*Simulink*® dos Subsistemas da função ELS
Fonte: Elaborado pelo Autor (2015)

Nesta figura é possível observar que os subsistemas *Position Light SS, Low Beam Light, Full Beam Light SS, Brake_Light SS, Right_Light_SS, Left_Light_SS* e *Reverse Light SS* se referem a cada um dos casos de iluminação considerados (luz de posição, luz baixa, luz alta, etc.). Subsistema "*Ignition_SS*" gera o sinal que habilita os outros subsistemas. Este subsistema é dependente das condições da bateria e da posição da chave de ignição. Este sinal será alto apenas quando a tensão da bateria estiver dentro dos níveis aceitáveis (9 a 16 volts) e a chave de ignição esteja em *acc*, *run* ou *crank e* por último o subsistema *Flash_Lamp* é responsável intermitência e sincronismos das lâmpadas de indicação.

Para controlar estes estados foi criado um diagrama *Matlab*/*Stateflow*® que foi colocado no controlador que pode ser visto na Figura 30 com os sinais:

Acc – entrada, ativado quando a chave de ignição estiver nas posições *Acc* ou *Run* ou *Crank*.

Run – entrada, ativado quando a chave de ignição estiver nas posições *Run* ou *Crank*.

Crank – entrada, ativado quando chave de ignição estiver na posição *Crank*.

Batt – entrada, representa a tensão da bateria, onde estará ativa se a tensão estiver entre 9 à 16V.

Ignition_Pos – saída, ativada quando *Acc*, *Run* ou *Crank* e *Batt* estiverem em nível 1, habilitando os outros subsistemas.

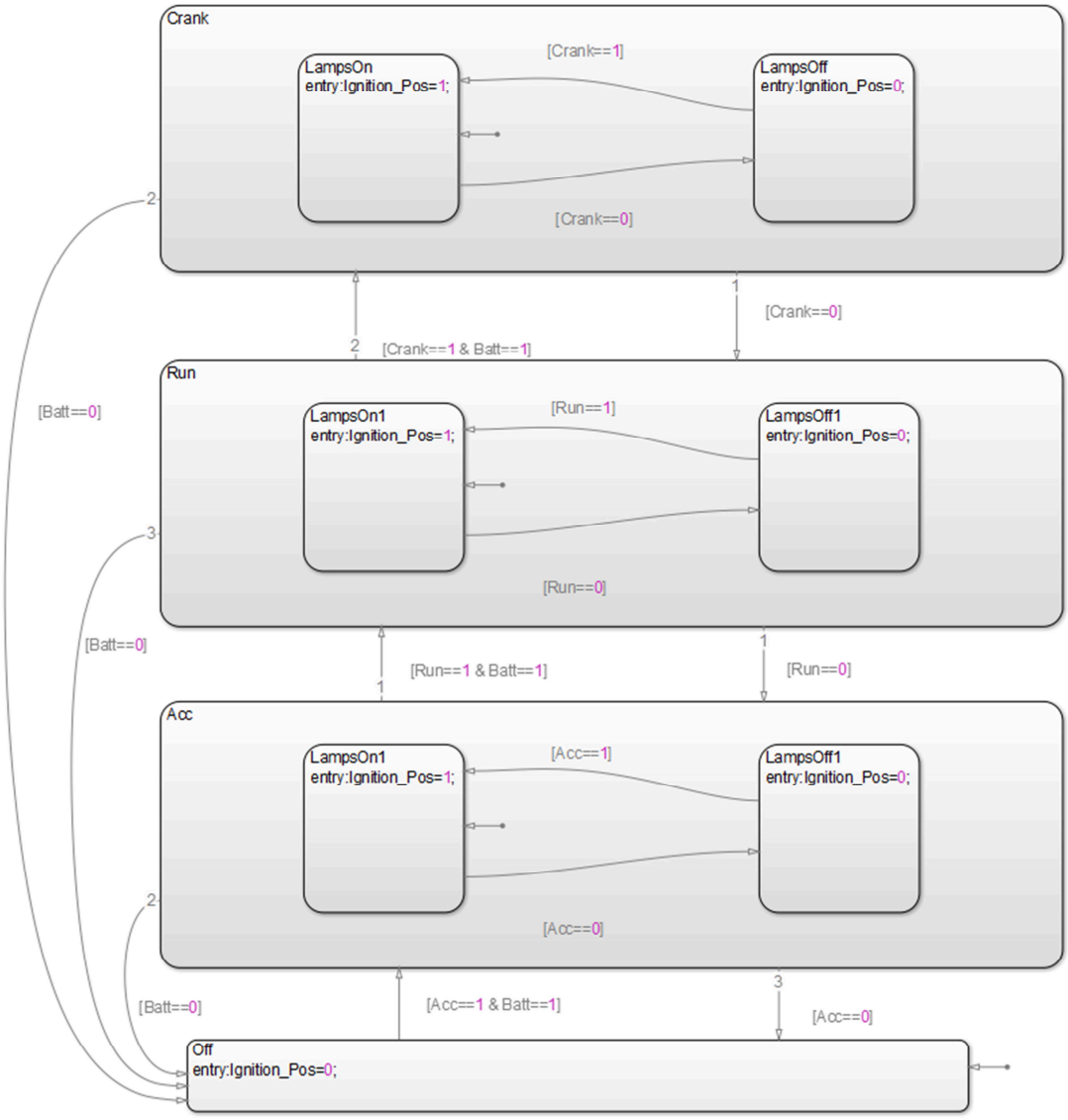


Figura 30: Diagrama de estados do *Matlab*/*Stateflow*® referente a posição da chave de ignição "*Ignition SS*"
Fonte: Elaborado pelo Autor (2015)

Tabela 3: Tabela de sinais do *Matlab/Stateflow*® referente ao Ignition Position.

Sinal	Porta	Tipo
ACC	Acc	Entrada
Crank	Crank	Entrada
Run	Run	Entrada
Battery	Batt	Entrada
Ignition Position	Ignition Pos	Saída

Fonte: Elaborado pelo autor (2015).

Os sinais de entrada e saída destes sistemas são exemplificados no diagrama temporal na Figura 31, onde é apresentado o caso de teste que foi usado para validar, posteriormente, todos os subsistemas.

Como apresentado no gráfico da Figura 31, observa-se que os sinais *Acc*, *Run*, *Crank*, são sinais sequenciais gerados pela rotação da chave de ignição. A curva *Batt* representa o comportamento da carga da bateria, que neste teste decresce a fim de demonstração de uma possível falha de bateria. O resultado, *Ignition Position*, demostra que o subsistema dependente da posição da chave e do nível da bateria.

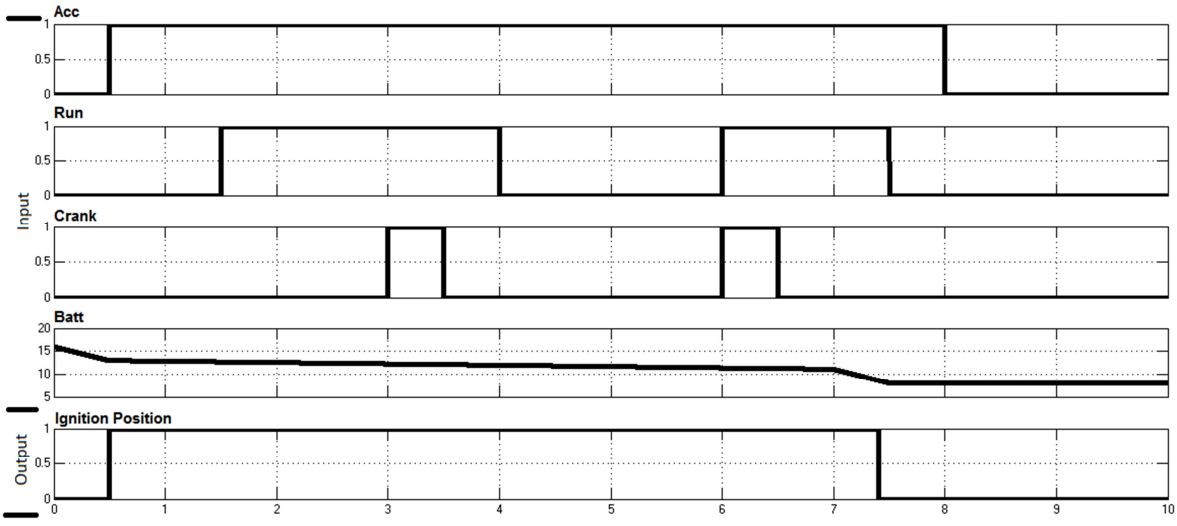


Figura 31: Gráfico dos sinais de entrada e saída do subsistema *Ignition_SS* do controlador de iluminação
Fonte: Elaborado pelo Autor (2015)

3.2.2.3 Detalhamento dos subsistemas do sistema de iluminação externa automotiva

Todos os subsistemas a seguir foram desenvolvidos em diagramas *Matlab*/*Stateflow*®. Para ilustrar o funcionamento destes sistemas, da Figura 32 até a Figura 46 demonstram os diagramas e gráficos de *Matlab*/*Stateflow*® desenvolvidos e os sinais de entrada e saídas, na seguinte ordem: Luz de posição, luz baixa, luz alta, luz de freio, luz de indicação direito, luz de indicação esquerda, intermitência da luz de indicação e luz de ré. Para facilitar o entendimento dos gráficos serão apresentados antes de cada caso os requisitos necessários para seu funcionamento.

a) Subsistema Luz de Posição

Para o acionamento das luzes de posição são necessários os seguintes requisitos:

- O interruptor da luz de posição deve ser acionado (PLS=1);
- A chave de ignição não esteja na posição *off* e que haja carga suficiente na bateria (IP=1);

Desta forma, o sinal de ativação de luzes de posição será alto (PL=1), caso contrário ele terá valor baixo (PL=0).

Para o acionamento do sinal de erro é necessário:

- O sinal de funcionamento da lâmpada esteja baixo (PL1=0);
- O sinal de ativação de luzes de posição esteja alto (PL=1);

Observando estes requisitos foi possível desenvolver os diagramas em *Matlab/Stateflow*®. O diagrama referente ao funcionamento das luzes de posição é demonstrado na Figura 32.

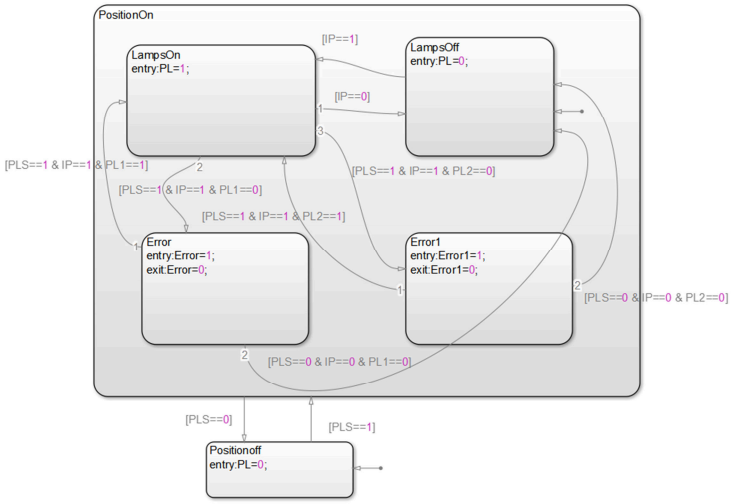


Figura 32: Diagrama *Matlab*/*Stateflow*® para a lógica das luzes de posição
Fonte: Elaborado pelo Autor (2015)

Tabela 4: Tabela de sinais do *Matlab/Stateflow*® referente ao Position Lights SS.

Sinal	Porta	Tipo
Ignition Position	IP	Entrada
Rear Lamp	PL2	Entrada
Position Switch	PLS	Entrada
Front Lamp	PL1	Entrada
Error Front Out	Error	Saída
Front/Rear Position	PL	Saída
Error Rear Out	Error1	Saída

Fonte: Elaborado pelo autor (2015).

Como apresentado no gráfico da Figura 33, os sinais de entrada do subsistema são *Ignition Position, Position_Lights_in, Front_Lamp e Rear_Lamp*.

O *Ignition Position* representa a saída do subsistema *Ignition SS*, sendo o sinal de habilitação desse subsistema.

O *Position_Lights_in* é o sinal proveniente da chave de luz que apresenta três posições, *Off, Luz de Posição e Luz Baixa*. Na posição *Off* não apresenta nível alto em nenhuma entrada. Na posição Luz de Posição gera nível alto na entrada Luz de Posição. Na posição Luz Baixa gera nível alto na entrada *Low_Bean*.

O *Front_Lamp* e *Rear_Lamp* indicam o funcionamento ou não da lâmpada dianteira e traseira da luz de posição, respectivamente. Quando em alto indica correto funcionamento e quando em baixo indica falha.

O sinais de saída desse subsistema são o *Position_Front_out*, *Position_Rear_out*, *Position_Front_error* e *Position_Rear_error*.

Os sinais *Position_Front_out* e *Position_Rear_out* são os sinais que comandam a ativação e desativação para lâmpadas de posição frontais e traseiras. Esses sinais seguem o sinal de entrada *Position_Lights_in*, enquanto o sinal *Ignition Position* estiver ativado.

Os sinais de erros *Position_Front_error* e *Position_Rear_error* são ativados quando há falha nas lâmpadas de posição frontal e traseira.

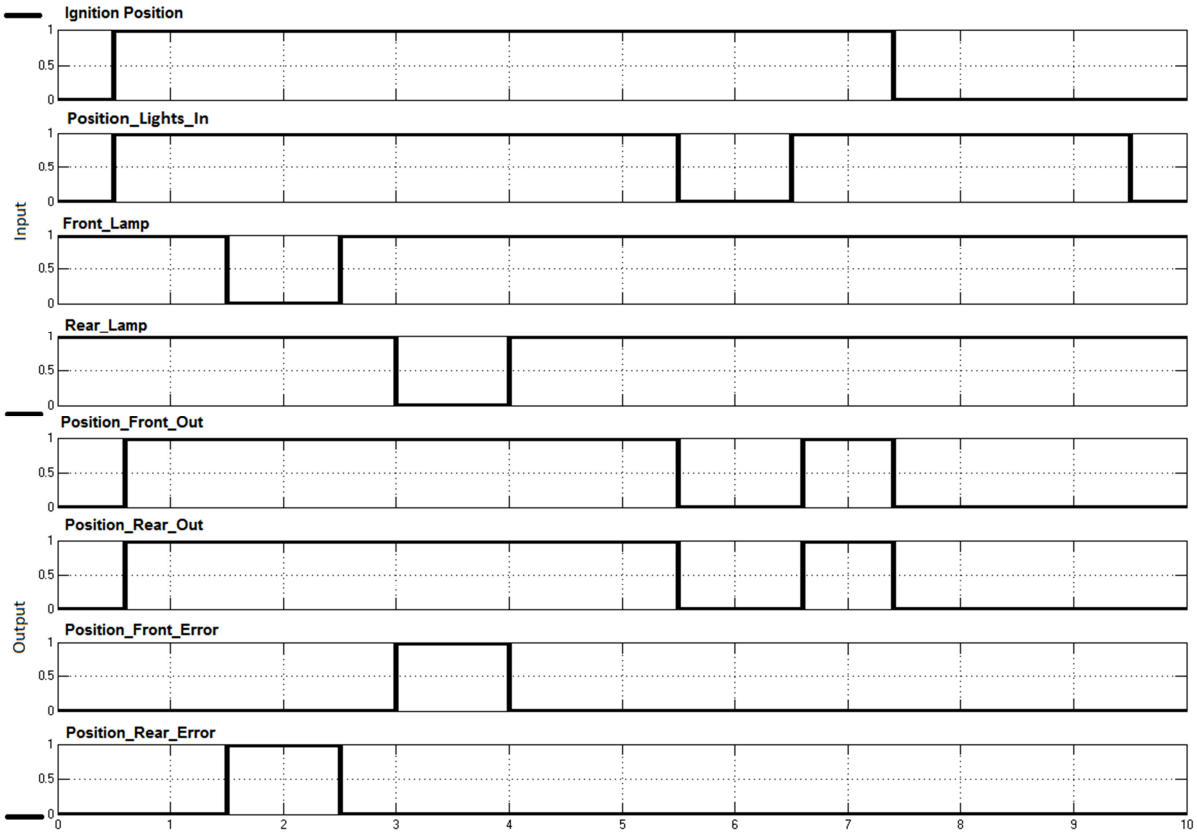


Figura 33: Gráfico das entradas e saídas para a lógica das luzes de posição
Fonte: Elaborado pelo Autor (2015)

b) Subsistema Luz Baixa

Para o acionamento da luz baixa são necessários os seguintes requisitos:

- O interruptor da luz baixa deve ser acionado (LBS=1);
- A chave de ignição não esteja na posição *off* e que haja carga suficiente na bateria (IP=1);

Desta forma, o sinal de ativação de luz baixa será alto (LB=1), caso contrário ele terá valor baixo (LB=0).

Para o acionamento do sinal de erro é necessário:

- O sinal de funcionamento da lâmpada esteja alto (LBL=0);
- O sinal de ativação de luz baixa em alto (LB=1);

A partir dos requisitos listados foi possível desenvolver o diagrama *Matlab/Stateflow*® demonstrado na Figura 34.

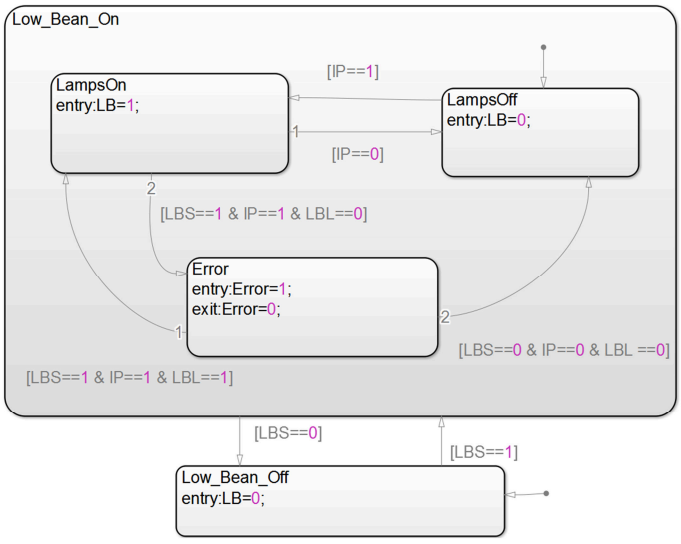


Figura 34: Diagrama *Matlab/Stateflow*® para a lógica da luz baixa.
Fonte: Elaborado pelo Autor (2015)

Tabela 5: Tabela de sinais do *Matlab/Stateflow*® referente ao Low Beam Light SS.

Sinal	Porta	Tipo
Ignition Position	IP	Entrada
Low Beam Switch	LBS	Entrada
Low Beam Lamp	LBL	Entrada
Error	Error	Saída
Low Beam	LB	Saída

Fonte: Elaborado pelo autor (2015)

O diagrama temporal dos sinais que validam o funcionamento da luz baixa é mostrado na Figura 35.

O sinal *Ignition Position* é a saída da função descrita nas Figura 30 e Figura 31.

O sinal *Low_Beam_in* é o sinal proveniente da chave de luz e seu comportamento é conforme descrito anteriormente.

O sinal *Low_Beam_Lamp* é o sinal representa o funcionamento ou não da Luz Baixa.

O sinal *Low_Beam_out* representa o comportamento da Luz Baixa do veículo. Esse sinal depende das entradas *Low_Beam_in* e *Ignition Position*.

O sinal *Low_Beam_error* representa o mal funcionamento da lâmpada de Luz Baixa e depende dos sinais *Low_Beam_in* e *Low_Beam_Lamp*.

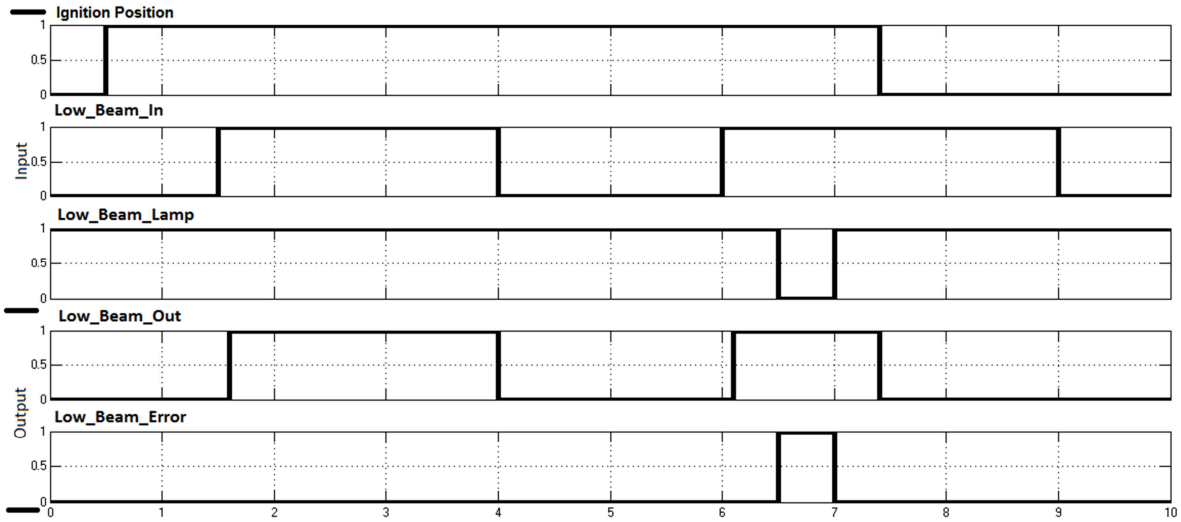


Figura 35: Gráfico dos sinais de entrada e saída para a lógica da luz baixa
Fonte: Elaborado pelo Autor (2015)

c) Subsistema Luz Alta

Em seguida iniciou-se o desenvolvimento do subsistema de luz alta. Os requisitos necessários são:

- O interruptor da luz alta deve ser acionado (FBS=1);
- A chave de ignição não esteja na posição *off* e que haja carga suficiente na bateria (IP=1);

Desta forma, o sinal de ativação da luz alta será alto (FB=1), caso contrário ele terá valor baixo (FB=0).

Para o acionamento do sinal de erro é necessário:

- O sinal de funcionamento da lâmpada esteja alto (FBL=1);
- O sinal de ativação da luz alta em alto (FB=1);

Com os requisitos apresentados foi possível gerar o gráfico em *Matlab/Stateflow*® que controla o funcionamento do subsistema de luz alta. Este gráfico é demonstrado na Figura 36.

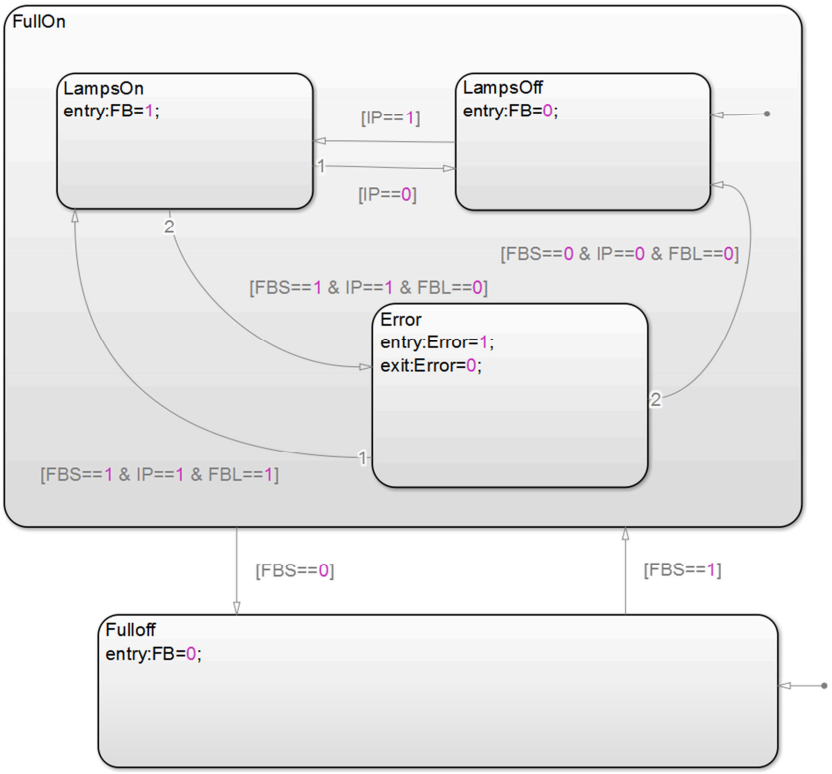


Figura 36: Diagrama *Matlab/Stateflow*® para lógica da luz alta.
Fonte: Elaborado pelo Autor (2015)

Tabela 6: Tabela de sinais do *Matlab/Stateflow*® referente ao Full Beam Light SS.

Sinal	Porta	Tipo
Ignition Position	IP	Entrada
Full Beam Switch	FBS	Entrada
Full Beam	FBL	Entrada
Error	Error	Saída
Full Beam	FB	Saída

Fonte: Elaborado pelo autor (2015)

O diagrama temporal dos sinais que validam o funcionamento da luz alta são mostrados na Figura 37.

O sinal *Ignition Position* é a mesma função já descrita anteriormente.

O sinal *Full_Beam_in* é o sinal proveniente da chave de luz alta.

O sinal é o sinal representa o funcionamento ou não da Luz Alta.

O sinal *Full_Beam_out* representa o comportamento da Luz Alta do veículo. Esse sinal depende das entradas *Full_Beam_in* e *Ignition Position*.

O sinal *Full_Beam_error* representa o mal funcionamento da lâmpada de Luz Alta, e depende dos sinais *Full_Beam_in*. e *Full_Beam_Lamp*.

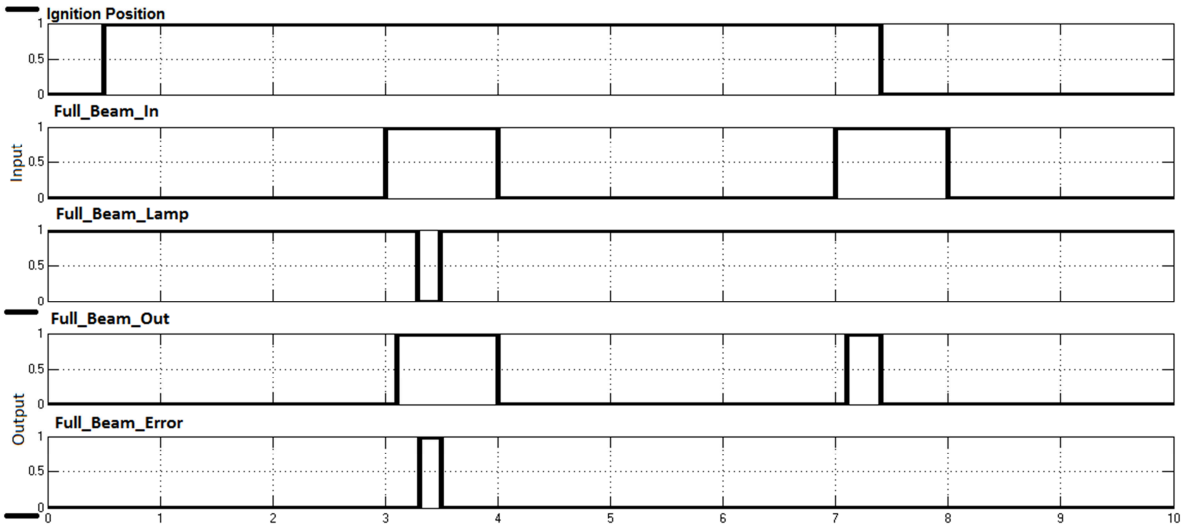


Figura 37: Gráfico dos sinais de entrada e saída para a lógica da luz alta.
Fonte: Elaborado pelo Autor (2015)

d) Subsistema Luz de Freio

O próximo sistema a ser demonstrado é o da luz de freio. Para o seu acionamento é necessário o cumprimento dos seguintes requisitos:

- O pedal de freio seja acionado (BLS=1);
- A chave de ignição não esteja na posição *off* e que haja carga suficiente na bateria (IP=1);

Desta forma, o sinal de ativação de luz de freio será alto (BL=1), caso contrário ele terá valor baixo (BL=0).

Para o acionamento do sinal de erro é necessário:

- O sinal de funcionamento da lâmpada esteja baixo (BLL=0);

- A ativação da luz de freio em alto (BL=1);

A partir disso, foi possível desenvolver o diagrama *Matlab/Stateflow*® de funcionamento deste sistema. A Figura 38 demonstra este diagrama.

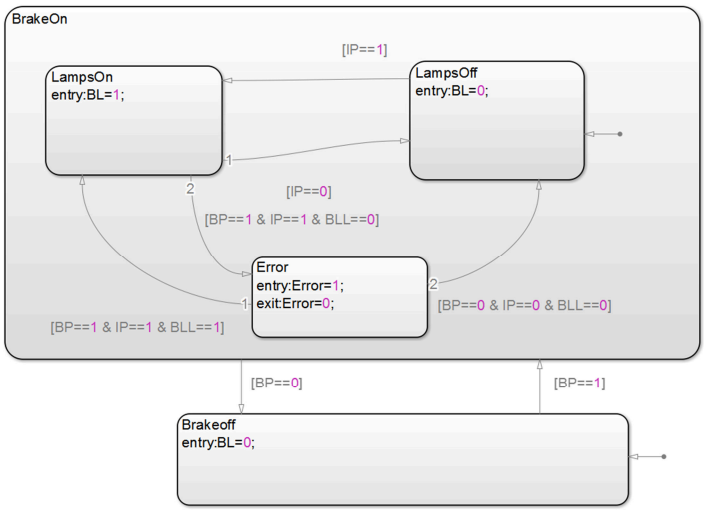


Figura 38: Diagrama *Matlab/Stateflow*® para a lógica da luz de freio
Fonte: Elaborado pelo Autor (2015)

Tabela 7: Tabela de sinais do *Matlab/Stateflow*® referente ao *Brake Light SS*.

Sinal	Porta	Tipo
Ignition Position	IP	Entrada
Brake Pedal	BP	Entrada
Brake Light Lamp	BLL	Entrada
Error	Error	Saída
Brake Light	BL	Saída

Fonte: Elaborado pelo autor (2015)

O diagrama temporal dos sinais que validam o funcionamento do subsistema luz de freio são mostrados na Figura 39.

O sinal *Ignition Position* é a mesmo sinal de saídas da função descrita na figura 30 e 31.

O sinal *Brake_Light_in* é o sinal proveniente do pedal de freio do veículo.

O sinal *Brake_Light_Lamp* é o sinal que representa o funcionamento ou não da Luz de Freio.

O sinal *Brake_out* representa o comportamento da Luz de Freio do veículo. Esse sinal depende das entradas *Brake_Light_in* e *Ignition Position*.

O sinal *Brake_error* representa o mau funcionamento da lâmpada de Luz Freio, e depende dos sinais *Brake_Light_in* e *Brake_Light_Lamp*.

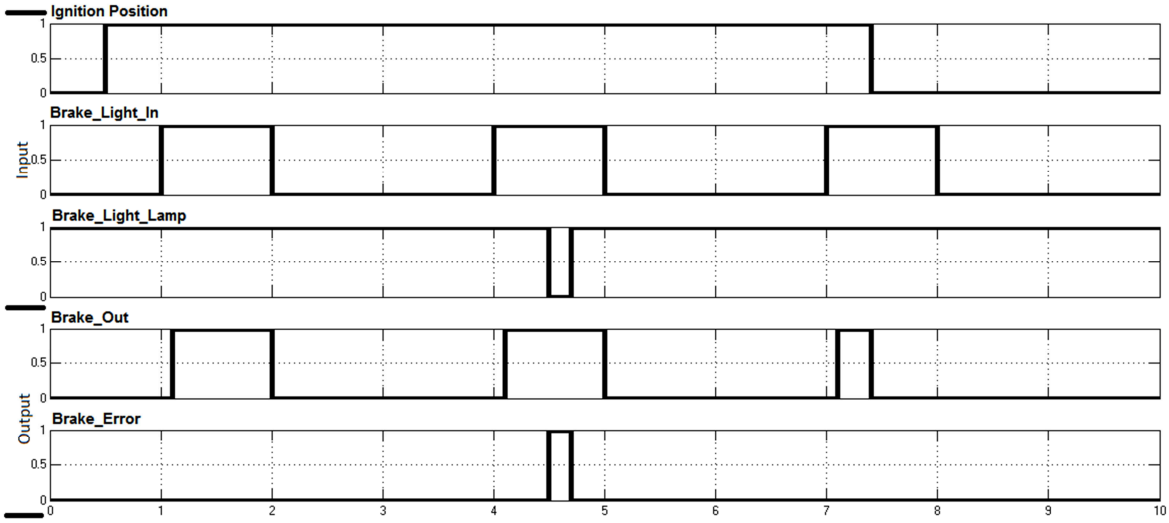


Figura 39: Gráfico dos sinais de entrada e saída para a lógica da luz de freio.
Fonte: Elaborado pelo Autor (2015)

e) Subsistema Luz de Indicação de Direção

Ambas as luzes de indicação têm seu funcionamento idêntico, entre si, sendo diferenciadas apenas pela posição da alavanca. Além disso, caso o botão de alerta seja acionado, ambos os casos deverão ser ativados. Portanto, para a luz de indicação direita é possível listar os seguintes requisitos:

- A alavanca da luz de indicação deve estar deslocada para cima (RLS=1);
- A chave de ignição não esteja na posição *off* e que haja carga suficiente na bateria (IP=1);
- Caso o sinal de alerta seja ativado, o sistema deve funcionar independentemente de outros requisitos (HW=1).

Desta forma, o sinal de ativação de luz de indicação direita será alto (RL=1), caso contrário ele terá valor baixo (RL=0).

Para o sinal de erro, os requisitos são:

- O sinal de funcionamento da lâmpada esteja baixo (RLL=0);
- A luz de indicação direita deve está acionada (RL=1);

Com estes requisitos definidos, foi possível desenvolver o diagrama *Matlab/Stateflow*® da Figura 40.

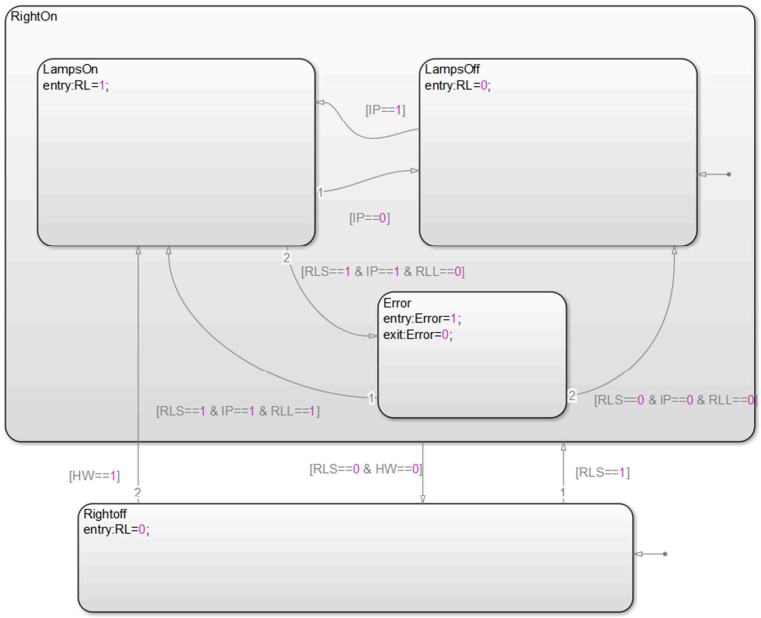


Figura 40: Diagrama *Matlab/Stateflow*® para lógica a luz de indicação direita
Fonte: Elaborado pelo Autor (2015)

Tabela 8: Tabela de sinais do *Matlab/Stateflow*® referente ao Right Light SS.

Sinal	Porta	Tipo
Ignition Position	IP	Entrada
Right Light Switch	RLS	Entrada
Right Light Lamp	RLL	Entrada
Hazard Warning	HW	Entrada
Error	Error	Saída
Right Indicator Light	RL	Saída

Fonte: Elaborado pelo autor (2015)

Considerando o funcionamento da luz de indicação esquerda é possível listar os seguintes requisitos:

- A alavanca da luz de indicação deve estar deslocada para baixo (LLS=1);
- A chave de ignição não esteja na posição *off* e que haja carga suficiente na bateria (IP=1);
- Caso o sinal de alerta seja ativado, o sistema deve funcionar independentemente de outros requisitos (HW=1).

Desta forma, o sinal de ativação de luzes de indicação esquerda será alto (LL=1), caso contrário ele terá valor baixo (LL=0).

Para o sinal de erro os requisitos são:

- O sinal de funcionamento da lâmpada esteja baixo (LLL=0);
- A luz de indicação direita deve está acionada (LL=1);

Com estes requisitos definidos, foi possível desenvolver o diagrama *Matlab/Stateflow*® da Figura 41.

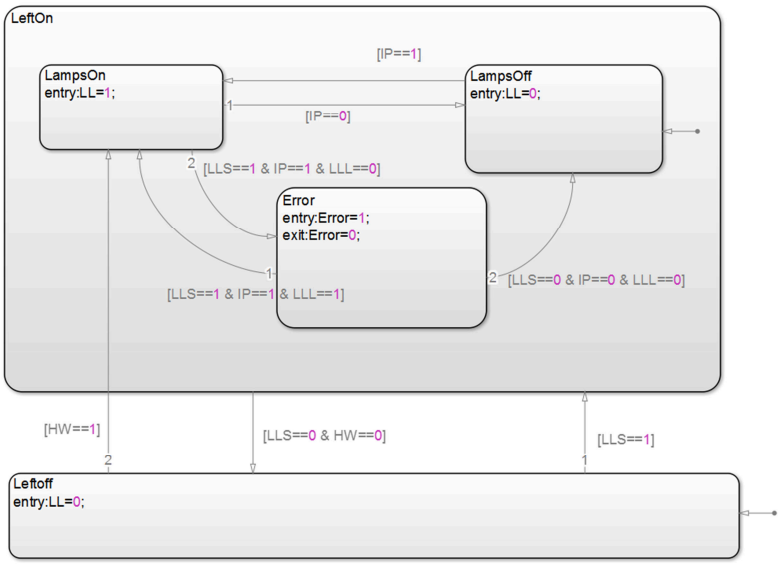


Figura 41: Diagrama *Matlab/Stateflow*® lógica para a luz de indicação esquerda
Fonte: Elaborado pelo Autor (2015)

Tabela 9: Tabela de sinais do *Matlab/Stateflow*® referente ao Left Light SS.

Sinal	Porta	Tipo
Ignition Position	IP	Entrada
Left Light Switch	LLS	Entrada
Left Light Lamp	LLL	Entrada
Hazard Warning	HW	Entrada
Error	Error	Saída
Left Indicator Light	LL	Saída

Fonte: Elaborado pelo autor (2015)

As luzes de indicação ainda necessitam de funções de intermitência e de sincronização:

i. **Intermitência:** gera um sinal cadenciado mesmo com um sinal alto continuo nas entradas, descrito no diagrama da Figura 42: Diagrama *Matlab/Stateflow*® referente intermitência das luzes de indicação.

ii. **Sincronização:** se houve dois sinais altos na entrada a saída serão dois sinais cadenciados e síncronos, descrito no diagrama da Figura 43.

Com esse objetivo foram desenvolvidos os blocos a seguir:

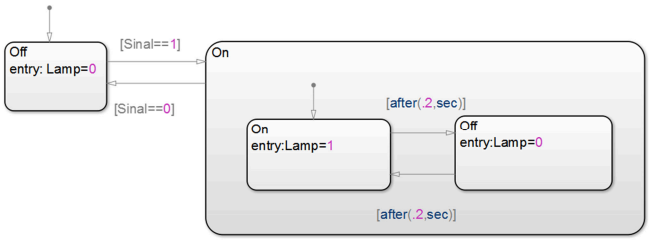


Figura 42: Diagrama *Matlab/Stateflow*® referente intermitência das luzes de indicação
Fonte: Elaborado pelo Autor (2015)

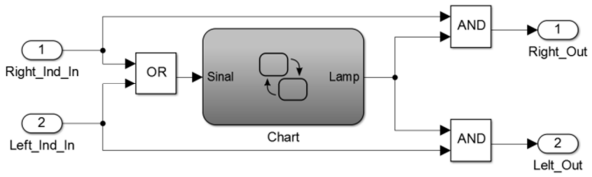


Figura 43: Diagrama do subsistema de sincronismo das luzes de indicação
Fonte: Elaborado pelo Autor (2015)

O diagrama temporal dos sinais que validam o funcionamento das luzes direita, esquerda e alerta são mostrados na Figura 44.

O sinal *Ignition Position* é a mesma função já descrita anteriormente.

Os sinais *Right_Light_in* e *Left_Light_in* são os sinais provenientes das chaves de setas direita e esquerda respectivamente do veículo.

O sinal *Hazard_Warning_in* é o sinal proveniente da chave de alerta do veículo.

O sinal *Brake_out* representa o comportamento da Luz de Freio do veículo. Esse sinal depende das entradas *Brake_Light_in* e *Ignition Position*.

O sinal *Brake_error* representa o mau funcionamento da lâmpada de Luz Freio, e depende dos sinais *Brake_Light_in* e *Brake_Light_Lamp*.

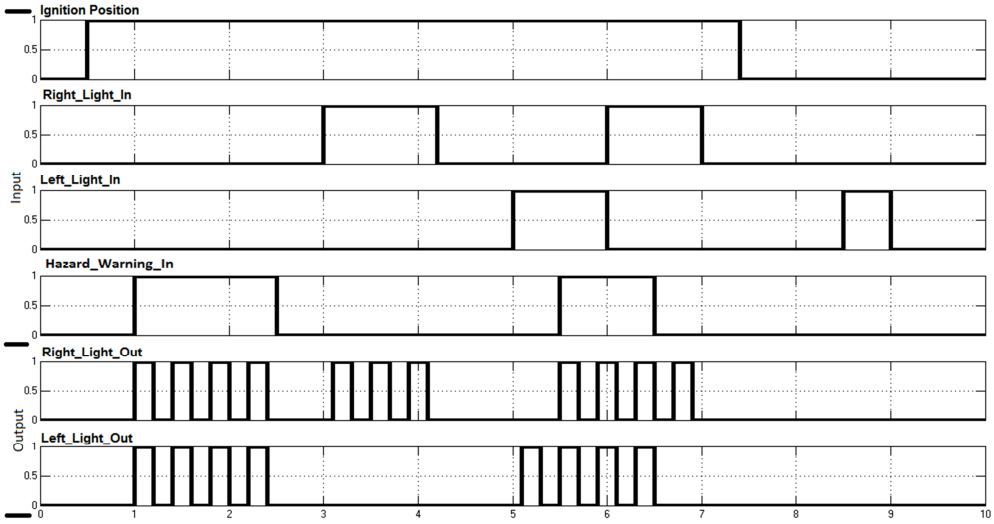


Figura 44: Gráfico dos sinais de entrada e saída para a lógica da luz de indicação direita, esquerda e alerta.
Fonte: Elaborado pelo Autor (2015)

f) Subsistema Luz de ré

Por último, foi criado o controlador da luz de ré. Para que seu funcionamento seja habilitado é necessário o cumprimento dos seguintes requisitos:

- O câmbio deve estar na posição de marcha ré (RGS=1);
- A chave de ignição não esteja na posição *off* e que haja carga suficiente na bateria (IP=1);
- O sinal de funcionamento da lâmpada esteja alto (RGL=1);

Desta forma, o sinal de ativação de luz de ré será alto (RG=1), caso contrário ele terá valor baixo (RG=0).

Para o sinal de erro os requisitos são:

- O sinal de funcionamento da lâmpada esteja baixo (RGL=0);
- A luz de indicação direita deve está acionada (RG=1);

Com estes requisitos foi possível desenvolver o diagrama *Matlab/Stateflow*® de funcionamento deste sistema. A Figura 45 demonstra este diagrama.

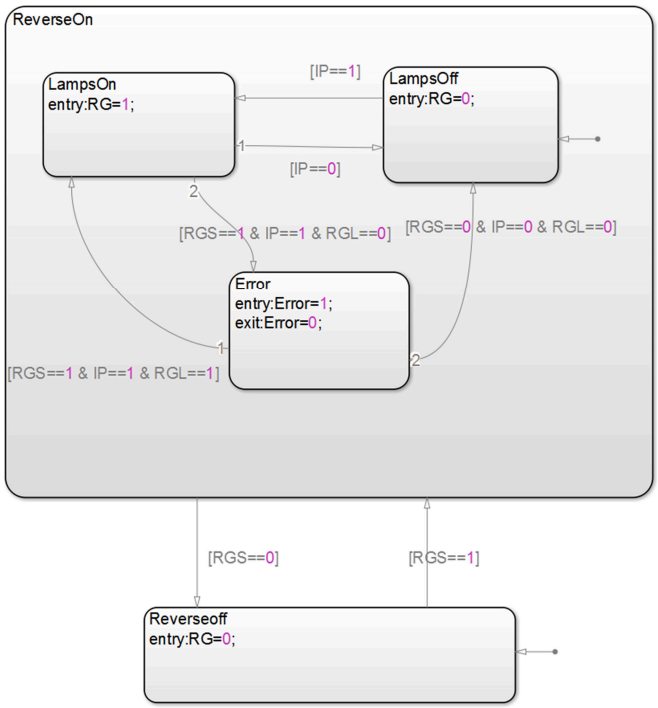


Figura 45: Diagrama *Stateflow*® para a lógica da luz de ré.
Fonte: Elaborado pelo Autor (2015)

Tabela 10: Tabela de sinais do Matlab/Stateflow® referente ao Reverse Light SS.

Sinal	Porta	Tipo
Ignition Position	IP	Entrada
Reverse Gear	RGS	Entrada
Reverse Light Lamp	RGL	Entrada
Error	Error	Saída
Reverse Light	RG	Saída

Fonte: Elaborado pelo autor (2015)

Os sinais que validam o funcionamento da luz de freio são mostrados no diagrama temporal da Figura 46.

O sinal *Ignition Position* é o sinal de saída da função descrita nas Figura 30 e Figura 31.

O sinal *Reverse_Light_in* é o sinal proveniente do sistemas de transmissão que indica se a marcha ré está engrenada.

O sinal *Reverse_Light_Lamp* é o sinal representa o funcionamento ou não da Luz Freio.

O sinal *Reverse_Light_out* representa o comportamento do pedal de freio do veículo. Esse sinal depende das entradas *Brake_Light_in* e *Ignition Position*.

O sinal *Reverse_error* representa o mau funcionamento da lâmpada de Luz Freio e depende dos sinais *Reverse_Light_in* e *Reverse_Light_Lamp*.

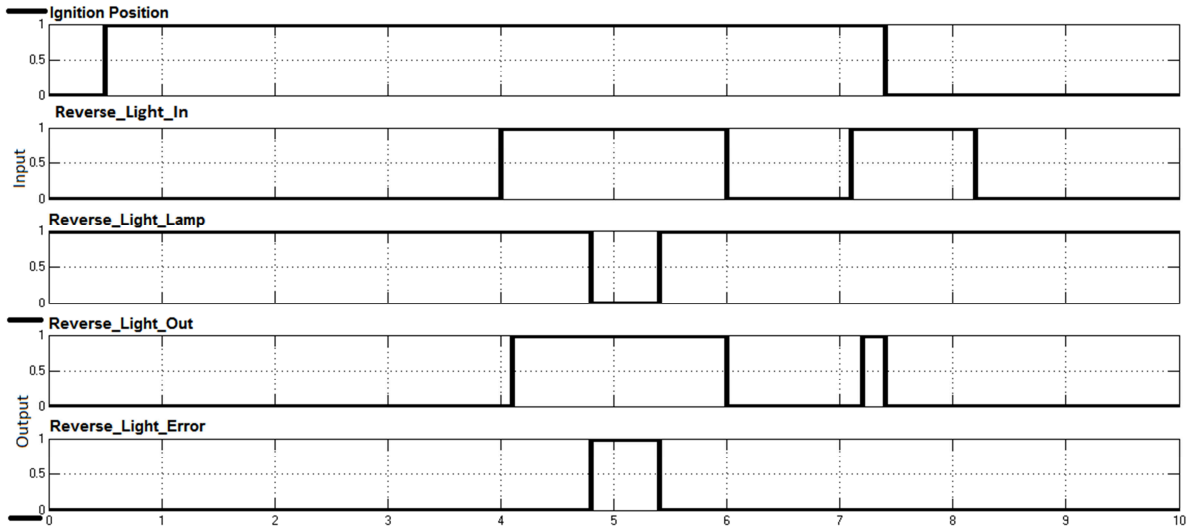


Figura 46: Gráfico dos sinais de entrada e saída para a lógica da luz de ré.
Fonte: Elaborado pelo Autor (2015)

3.2.2.4 Teste e validação da função sistema de iluminação externa automotiva

Seguindo a metodologia, o projeto do controlador da função ELS é validado utilizando o diagrama mostrado na Figura 47, onde observa-se a inclusão de blocos de geração de sinal (*Matlab*®/*Signal Builder*®) utilizado para simular os valores de entrada encontrados em um ambiente real.

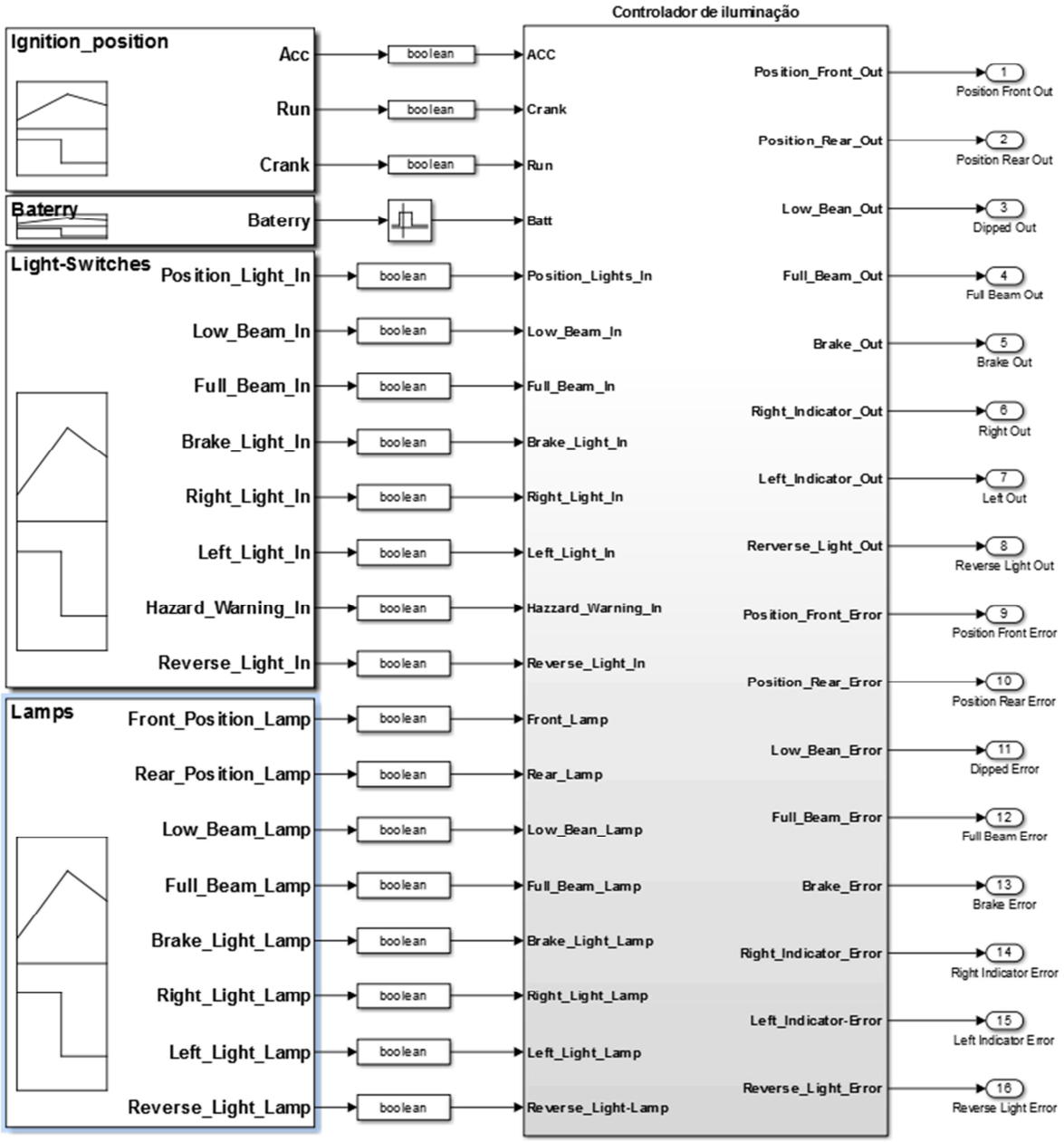


Figura 47: Diagrama para validação do controlador de iluminação com os sinais de entrada gerados pelo *Signal Builder*®.
Fonte: Elaborado pelo Autor (2015)

No sistema de iluminação externa existem saídas que podem ser ativadas por sinais diferentes, como é caso das luzes indicadoras. Esta saída pode ser ativada tanto pela sua própria chave ou pelo sinal de alerta.

Procurando validar as diversas combinações de entrada que podem gerar possíveis falhas na função, foi elaborado um conjunto de sinais prevendo essas situações, garantindo dessa forma o perfeito funcionamento do sistema. Esses sinais são mostrados nos diagramas temporais da Figura 48 e Figura 49.

Na Figura 48, os três primeiros sinais correspondem às posições da chave de ignição. O sinal *Batt* representa o nível de carga da bateria.

Os demais sinais são as entradas do sistema de iluminação exterior, onde os sinais *Position_light_in*, *LowBeam_In*, *Full_Beam*, *Brake_Light_in* e *Reverse_Light_in* foram determinados de maneira que pode-se observar se a saída seguiam a variação da entrada e seu comportamento em uma situação de problema (bateria baixa após 7,5s). Os sinais *Right_Light_In*, *Left_Light_In* e *Hazerd_Warning_In* foram escolhidos, além dos motivos citado acima, para verificar a interação e sincronismo dos sinais na saída, por isso tem-se períodos onde mais de um desses sinais estão ativo.

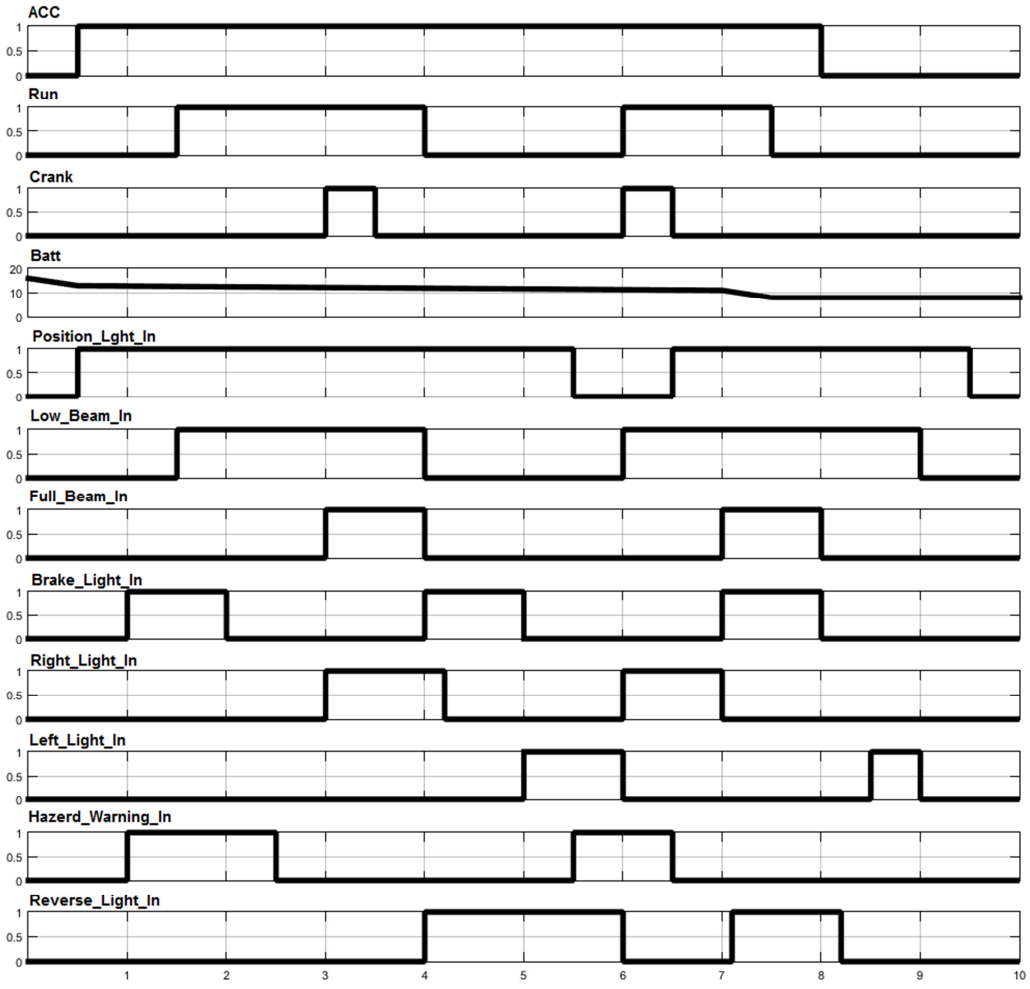


Figura 48: Sinais de entradas utilizados para o caso de teste da função ELS.
Fonte: Elaborado pelo Autor (2015)

Na Figura 49 tem-se diagramas temporais dos sinais de entrada que descrevem o funcionamento das lâmpadas, onde são simulados vários casos de falha, em períodos para comprovar o correto funcionamento dos subsistemas.

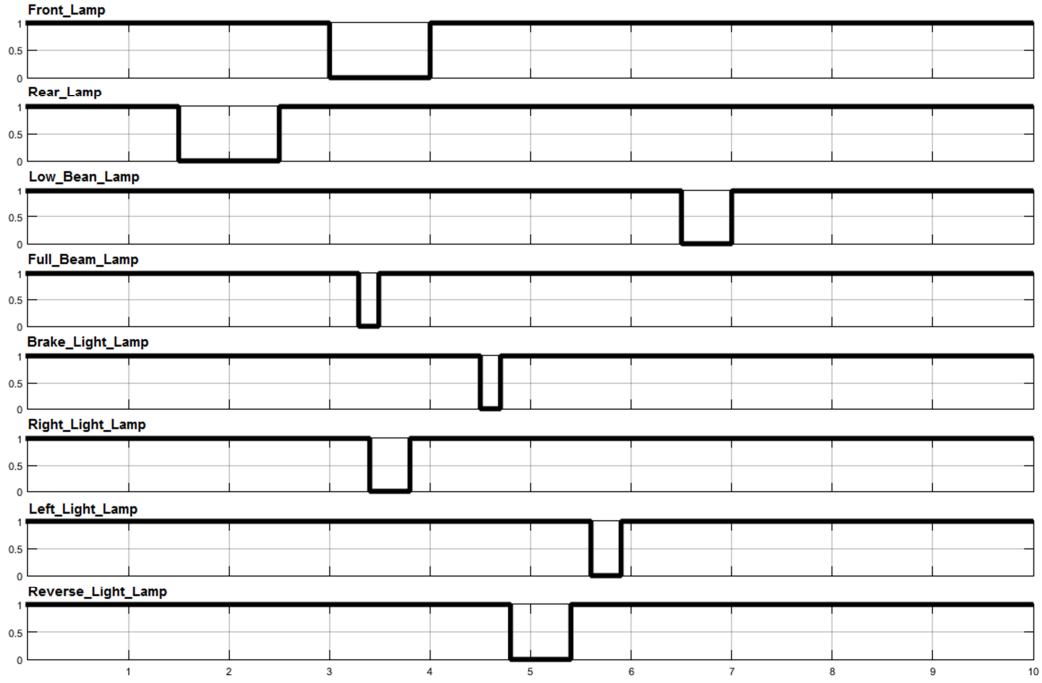


Figura 49: Entradas dos sinais de falha para o caso de teste da função ELS.
Fonte: Elaborado pelo Autor (2015)

No diagrama temporal da Figura 50, mostra-se os sinais de saída do controlador. Todos os subsistemas são ativados quando a chave de ignição estiver na posição *Acc* (próximo de 0,5 segundos), após isso os sinais de saída seguem as entradas e são desativados quando a bateria atinge nível mínimo do 9V que ocorre próximo de 7,5 segundos, sendo os sinais Acc e o nível de bateria (*Batt*) mostrados na Figura 48, bem como os demais sinais de entrada do controlador.

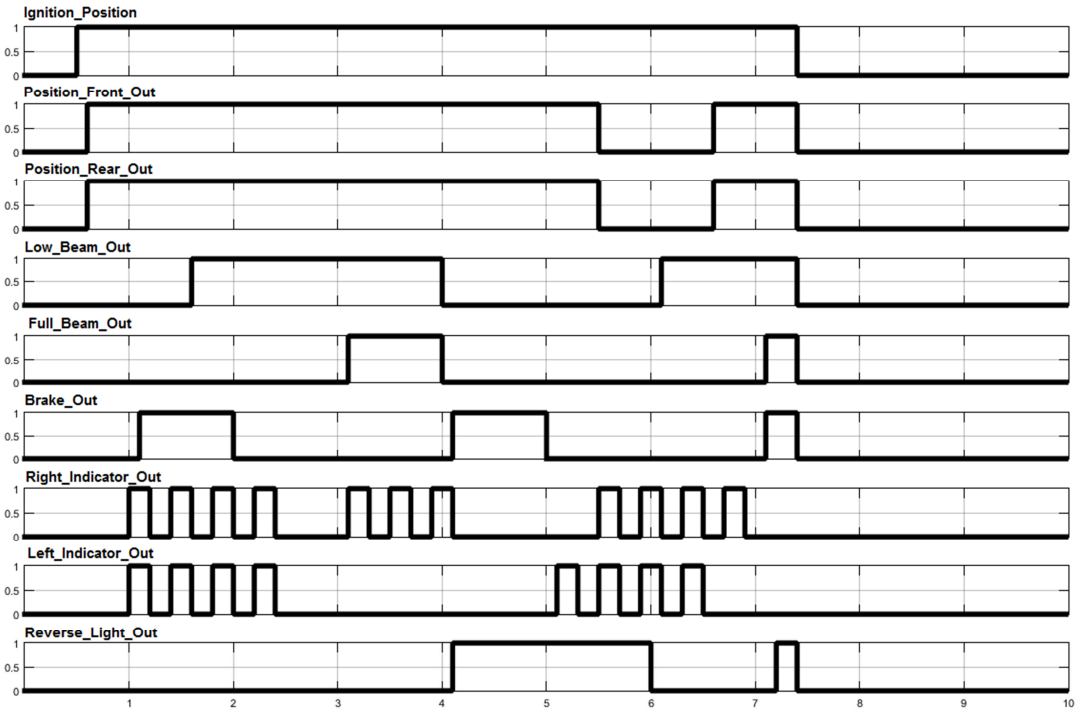


Figura 50: Saídas dos sinais referentes ao caso de teste da função ELS.
Fonte: Elaborado pelo autor (2014)

No diagrama temporal da Figura 51, tem-se os sinais de erros que são utilizados para sinalização no painel, validando os sinais de saída de erros do controlador.

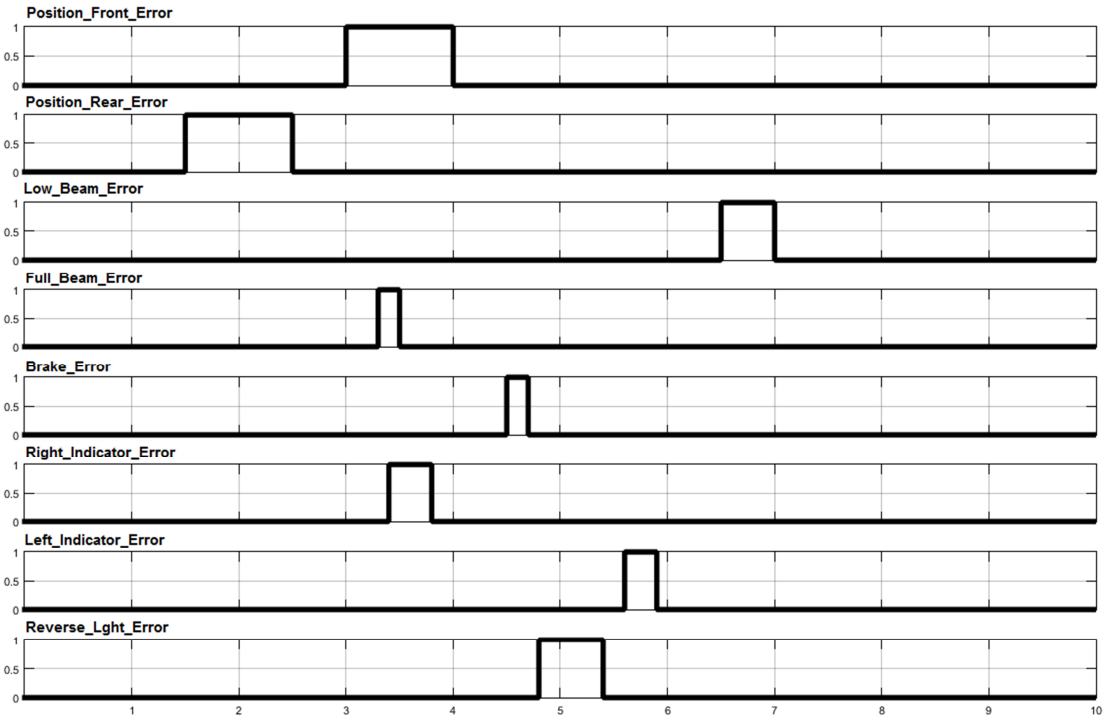


Figura 51: Saídas de erros referente ao caso de teste da função ELS.
Fonte: Elaborado pelo autor (2015)

3.2.3 *Software-In-The-Loop* para um Sistema de Iluminação Externa Automotiva

O método *de Software-in-the-loop* combina um simulador flexível e de baixo custo com a fidelidade de um emulador de *hardware*. Segundo Demers (2007), o SIL pode ser utilizado tanto no *desenvolvimento* do projeto, como também na fase de testes e, potencialmente, oferece uma enorme economia de custos, maximizando a reutilização de código e minimizando o esforço necessário para o desenvolvimento de *software*.

Na simulação do SIL, parte do modelo já existe na ferramenta utilizada para esta simulação, no caso deste trabalho o *Matlab*/*Simulink*®. Esta ferramenta já oferece a opção de se criar o código em linguagem C do modelo desenvolvido na fase de MIL.

Acessando a função de geração de uma função chamada de *S-Function,* automaticamente, o *Matlab*/*Simulink*® cria o código referente ao funcionamento do modelo e o insere em um bloco do próprio programa.

Com este bloco, é possível rodar novamente a simulação, com os mesmos valores de entrada e verificar se as saídas estão idênticas ao teste realizado para o MIL. A Figura 52 ilustra o bloco *S-Function* gerado a partir do modelo inicial.

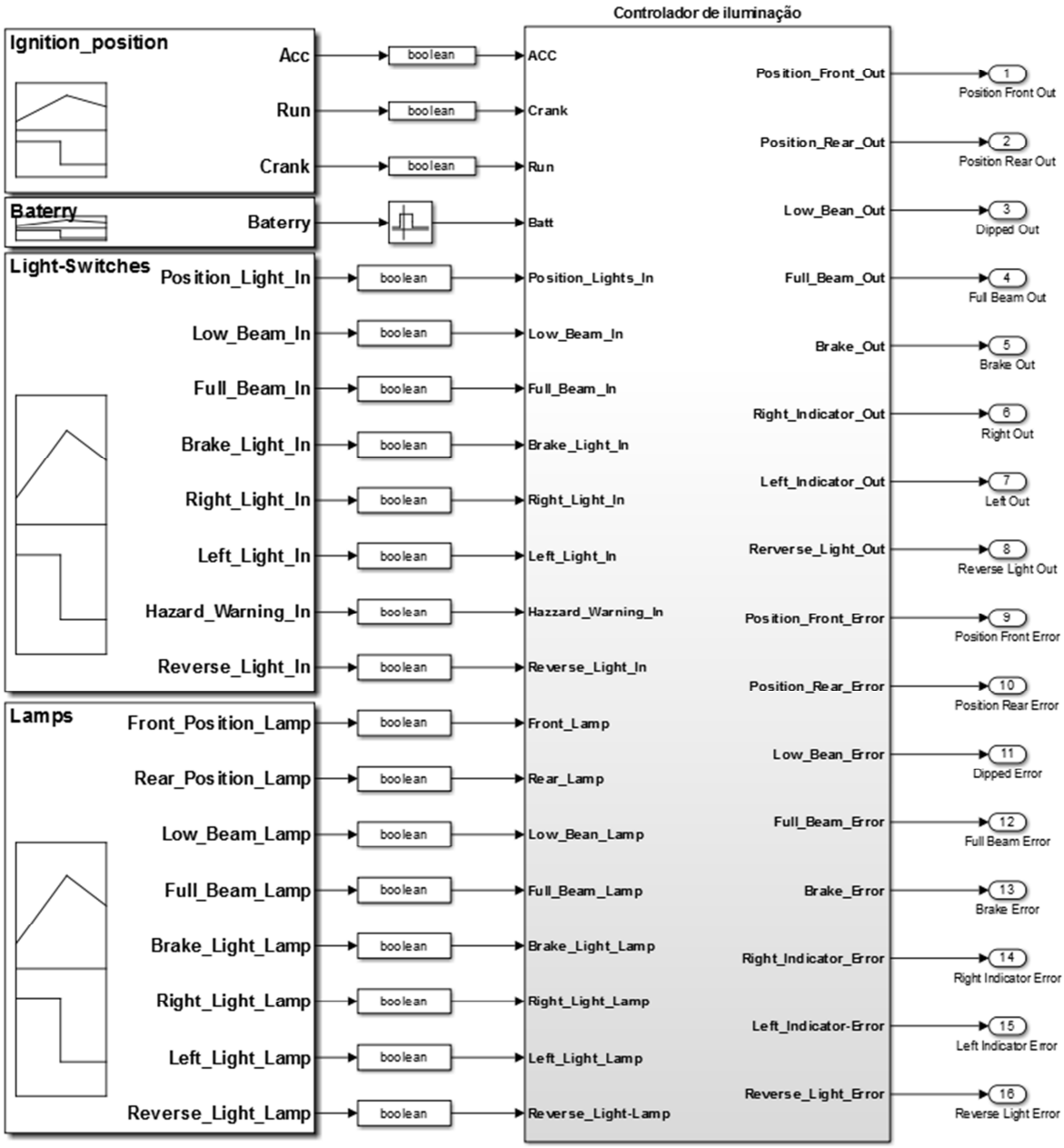


Figura 52: Diagrama *Matlab/Simulink*® da *S-Function* gerada a partir do modelo inicial do controlador da função ELS.
Fonte: Elaborado pelo Autor (2015)

É possível observar que esse modelo no *Matlab/Simulink*® fica muito parecido com o modelo da fase de MIL. No entanto o bloco principal não é mais formado por outros subsistemas. É em si um bloco com o código em linguagem C que funciona exatamente como o modelo utilizado na fase MIL.

Para provar que este o código reage de forma idêntica ao subsistema da fase MIL, uma nova simulação foi realizada com as mesmas entradas, utilizando o

controlador com o código gerado. As Figura 53 e Figura 54 demonstram os sinais envolvidos nas duas etapas, confrontando os resultados.

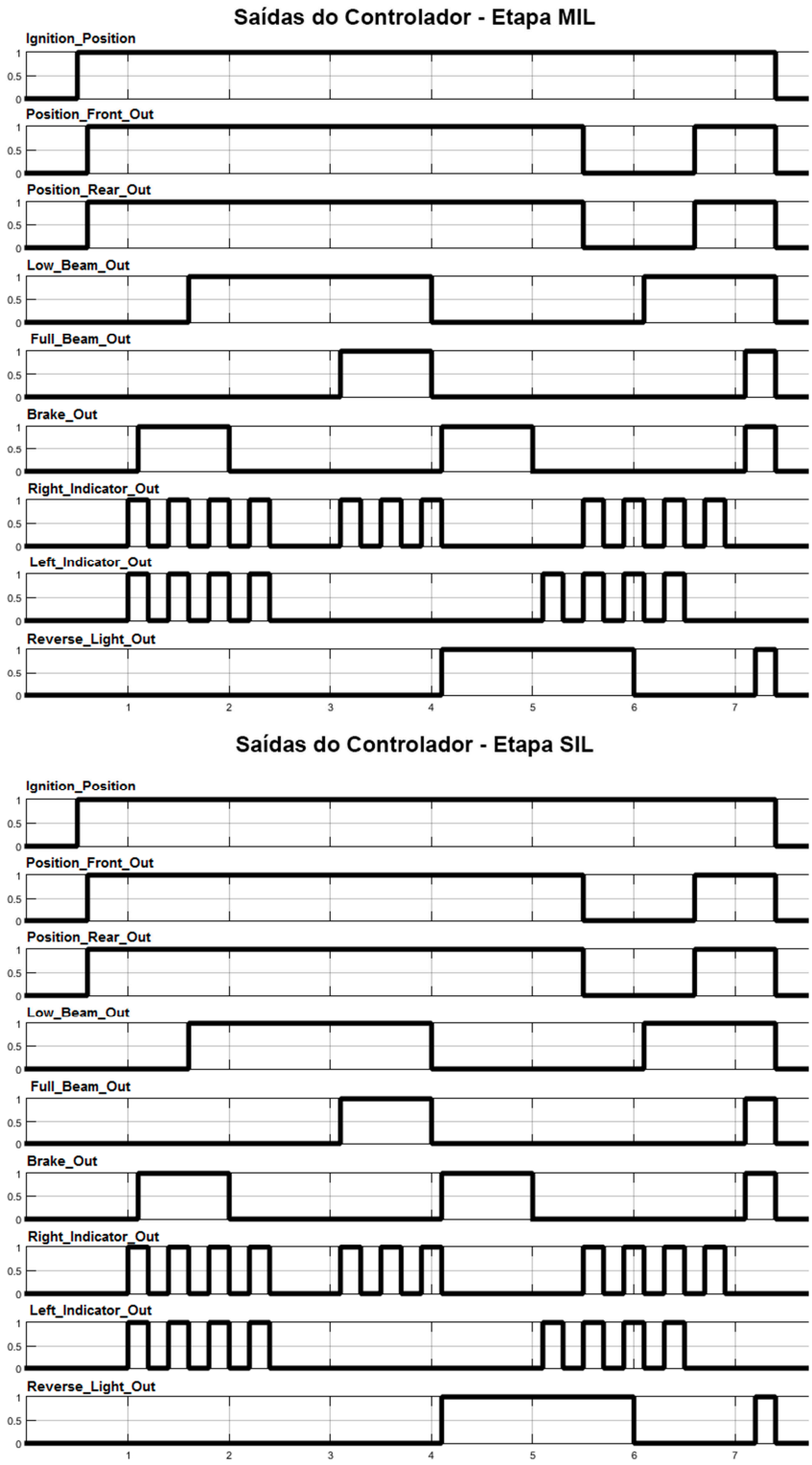


Figura 53: Gráfico comparativo das saídas dos sinais das etapas MIL e SIL.
Fonte: Elaborado pelo Autor (2015)

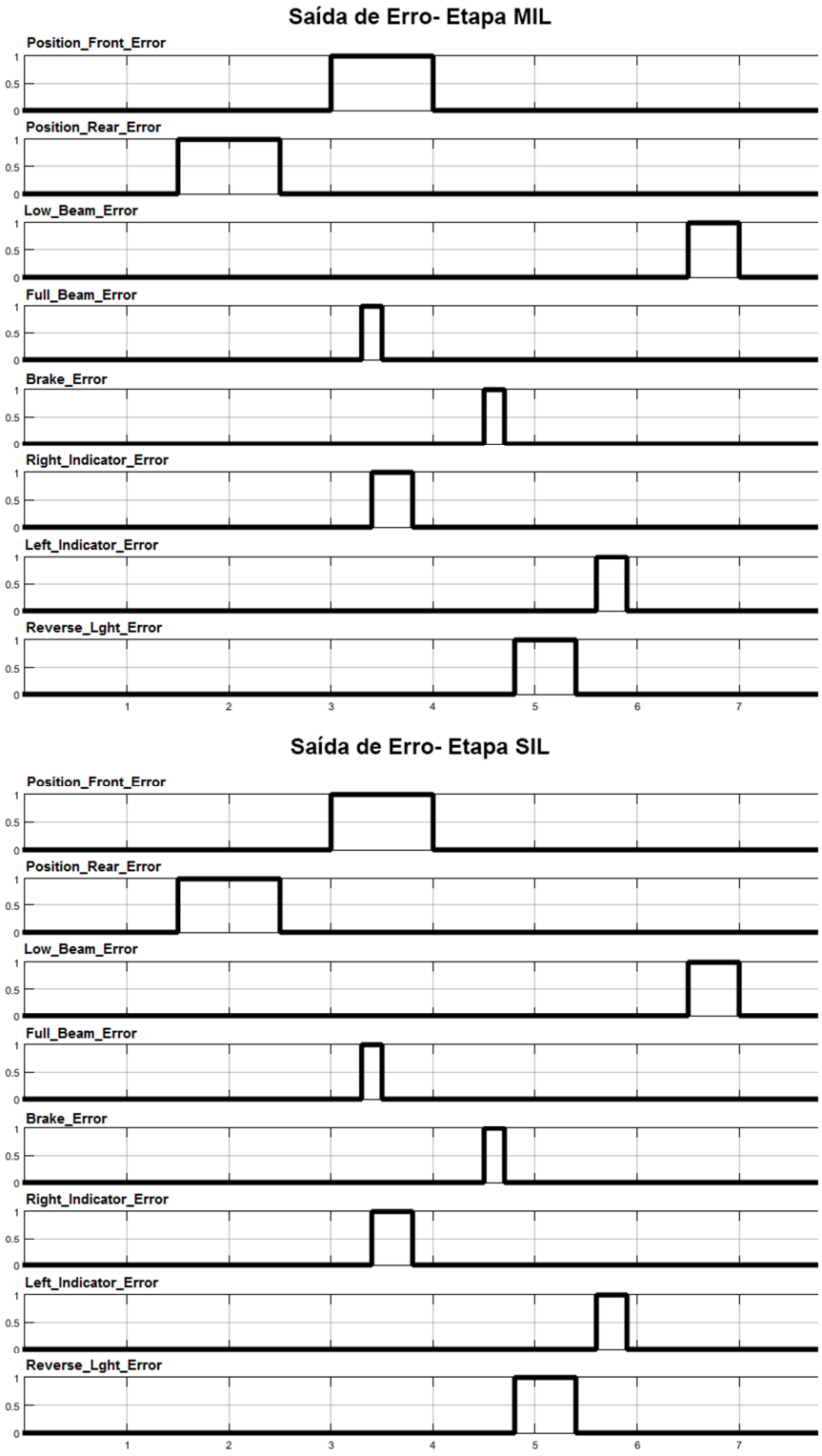


Figura 54: Gráfico comparativo das saídas de erros das etapas MIL e SIL.
Fonte: Elaborado pelo Autor (2015)

O código que foi gerado automaticamente e inserido dentro do bloco de *Matlab/Simulink*® está apresentado no Apêndice A.

3.3 PROJETO DA FUNÇÃO DO PAINEL DE INSTRUMENTOS AUTOMOTIVO (IP)

O painel de instrumentos (IP) é composto por indicadores que permitam ao motorista obter informações e alertas relevantes para a condução do veículo. Os tipos de instrumentos podem variar de acordo com a grandeza representada. A quantidade pode variar dependendo do modelo do veículo, sendo que os mais simples são compostos de velocímetro e indicador de combustível e mais completos sendo integrados com centrais que possuem visores gráficos apresentando informações mais dinamicamente. As indicações mais importantes devem ser visualizadas rapidamente, utilizando o indicador mais adequado para cada situação.

3.3.1 Características da Função do Painel de Instrumentos Automotivo

Como principais características dessa função, pode-se destacar:

a) A habilitação do painel ocorrerá somente quando a chave de ignição estiver na posição *acc*, *run* e *crank*.
b) Na transição de *off* para *acc*, o painel realizará um auto teste acionando todas as indicações por 2 segundos. Após esse período, entra automaticamente em operação normal.
c) A função somente atuará se o valor de entrada da bateria estiver entre 9 e 16 Volts.
d) As constantes de tratamento dos sinais devem fornecer precisão satisfatória para os ponteiros e mostradores.

3.3.2 *Model-In-The-Loop* para um Painel de Instrumentos Automotivo

Semelhante à função anterior foi seguido o método MBD e realizados os passos de descrição da planta física, definição dos requisitos, projeto e detalhamento do projeto.

3.3.2.1 Descrição da planta física do painel de instrumentos automotivo

A planta física é composta pelos sinais de entradas correspondentes às chaves e sinais provenientes de outras ECUs e tem como principais saídas o painel de instrumentos composto de sinalizadores (LEDs e lâmpadas) e mostradores (ponteiros, *displays* digitais).

Os subsistemas dependem da posição da chave de ignição, sendo que a maioria dos subsistemas é ativada a partir da posição *Acc*.

Existem dois tipos básicos de ICMs, a centralizada onde todas as entradas são feitas através de conexões individuais, e a distribuída onde existe um barramento CAN onde trafegam vários sinais sem a necessidade de conexões individuais para cada sinal.

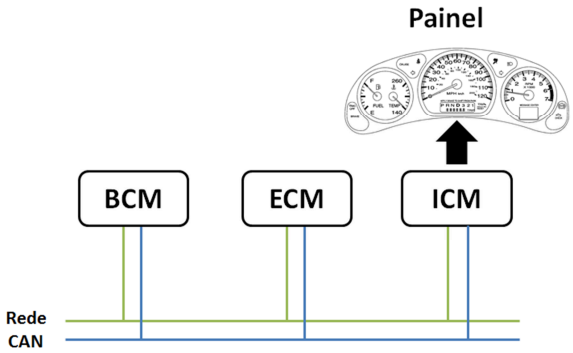


Figura 55: Módulo do painel de instrumentos.
Fonte: Elaborado pelo Autor (2015)

3.3.2.2 Definição dos requisitos do painel de instrumentos automotivo (IP)

Para a o projeto do IP foram considerados os seguintes requisitos:

Tabela 11: Requisitos da função do painel de instrumentos.

Para desligar todo o painel:
• **REQ-01**: a posição da ignição deve estar em *off*;
• **REQ-02**: a tensão da bateria estiver fora da faixa de 9 a 16 volts;
Para ativar auto teste:
• **REQ-03**: a tensão da bateria estiver na faixa de 9 a 16 volts;
• **REQ-04**: houver uma transição na chave de ignição de *off* para *Acc*;
Para ativar lâmpadas de sinais abaixo:

Engine_Alert_Out,
Seat_Belt_Alert_Out,
Door_Alert_Out,
Brake_Fluid_Alert_Out,
Engine_Fluid_Alert_Out,
Battery_Alert.

- **REQ-05**: o IC em operação e a entrada correspondente em nível alto;

Para *Engine_Temp_Alert*:

- **REQ-06**: ativar quando a temperatura atingir 72º Celsius;
- **REQ-07**: desativar quando a temperatura baixa de 69º Celsius;

Para ativar as luzes de indicação de posição esquerda:

- **REQ-08**: a posição da ignição deve estar em *Acc*, *Run*, ou *Crank*.

Fonte: Elaborado pelo autor (2015).

3.3.2.3 Projeto do controlador do painel de instrumentos automotivo

O modelamento dessa função utiliza blocos de *Matlab/Simulink*®, apresentados na Figura 56.

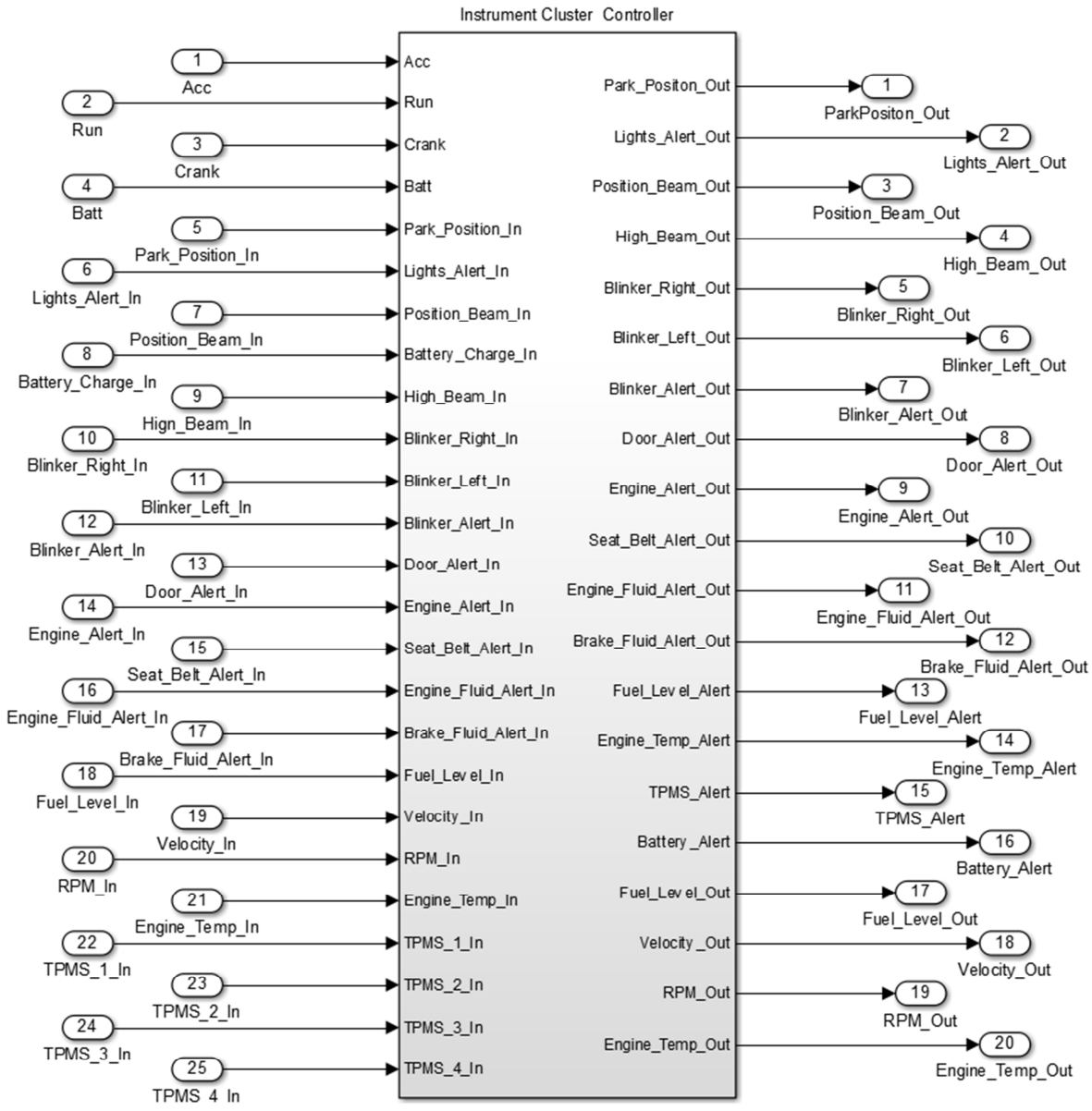


Figura 56: Diagrama *Matlab/Simulink*® do controlador da função IC.
Fonte: Elaborado pelo Autor (2015)

3.3.2.4 Detalhamento do projeto dos subsistemas do painel de instrumentos automotivo

Observa-se no diagrama principal que o controlador é composto por dois *charts* do *Matlab/StateFlow*® mostrados na Figura 57, sendo um deles (*Ignition_Position_TT*) uma tabela verdade e o outro a (*Instrument_Cluster_SS*) máquina de estados do controlador.

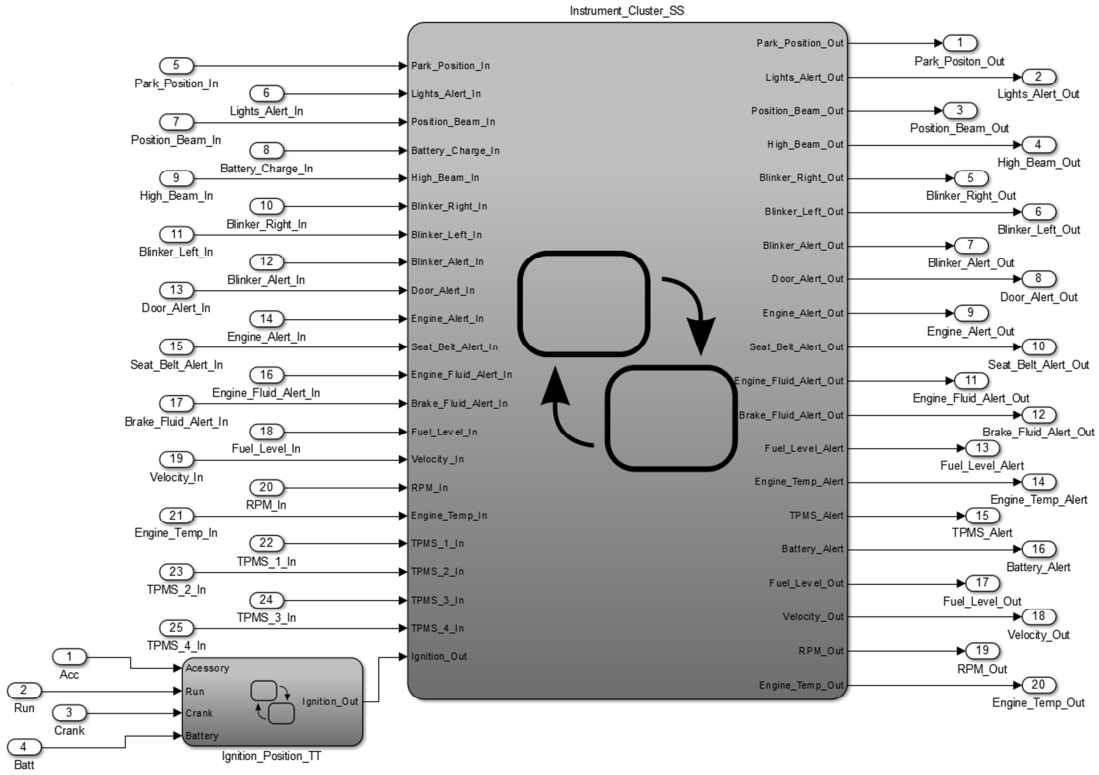


Figura 57: Diagrama *Matlab/Simulink*® dos blocos do controlador da função IC.
Fonte: Elaborado pelo Autor (2015)

No Bloco *Ignition_Position_TT* está a verificação das posições da chave de ignição e o nível da bateria usou-se um blocos *StateFlow*® do *Simulink*® com as seguintes entradas: *Accessory*, *Run*, *Crank* e *Battery*, e saída: *Ignition_Out*, como apresentado na Figura 58.

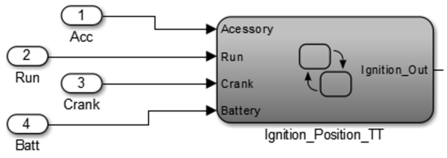


Figura 58: Diagrama *Matlab/Simulink*® do bloco Ignition_Position_TT
Fonte: Elaborado pelo Autor (2015)

Para o sinal de saída (*Ignition_Out*) é considerada a Tabela 12: Estados de saída do *Stateflow*® do bloco *Ignition_Position_SS*., onde é representada a relação entre os sinais de entrada e os estados resultantes na saída.

Tabela 12: Estados de saída do *Stateflow*® do bloco *Ignition_Position_SS*.

Acc	Run	Crank	Voltage Sensor	Neutral - [0] Acc - [1] Run - [2] Crank - [3] Battery_Alert - [4]
0	0	0	0	4
0	0	0	1	0
0	0	1	0	4
0	0	1	1	0
0	1	0	0	4
0	1	0	1	0
0	1	1	0	4
0	1	1	1	0
1	0	0	0	4
1	0	0	1	1
1	0	1	0	4
1	0	1	1	0
1	1	0	0	4
1	1	0	1	2
1	1	1	0	4
1	1	1	1	3

Fonte: Elaborado pelo Autor (2015)

Os valores assumidos por esse sinal (*Ignition Out*, representado na quarta coluna da Tabela 12) determinam os modos de operação do controlador.

O sistema de ignição modifica o valor da *Ignition_Out*, assumindo valores de 0 a 4.

Quando *Ignition_Out* assume valor 0 o sistema fica desabilitado.

Em valor 1 habilita o acionamento da *Position_Beam* e a alimentação dos acessórios.

Em valor 2 habilita a verificação das luzes do painel (apenas nesta passagem ocorrerá a verificação das luzes, valor 1 para 2); todas as lâmpadas acendem e após 0,5 segundos são apagadas, e posteriormente o painel é habilitado.

Em valor 3 ocorre o acionamento do motor de partida.

Durante esses processos se houver decaimento do nível de carga da bateria, fora do intervalo de funcionamento, a saída *Ignition_Out* assume valor 4 e desabilita todo o subsistema do bloco *Instrument_Cluster_SS*.

O diagrama *Matlab/StateFlow*® completo do bloco *Instrument_Cluster_SS* é apresentado na Figura 59, e suas partes serão detalhadas a seguir.

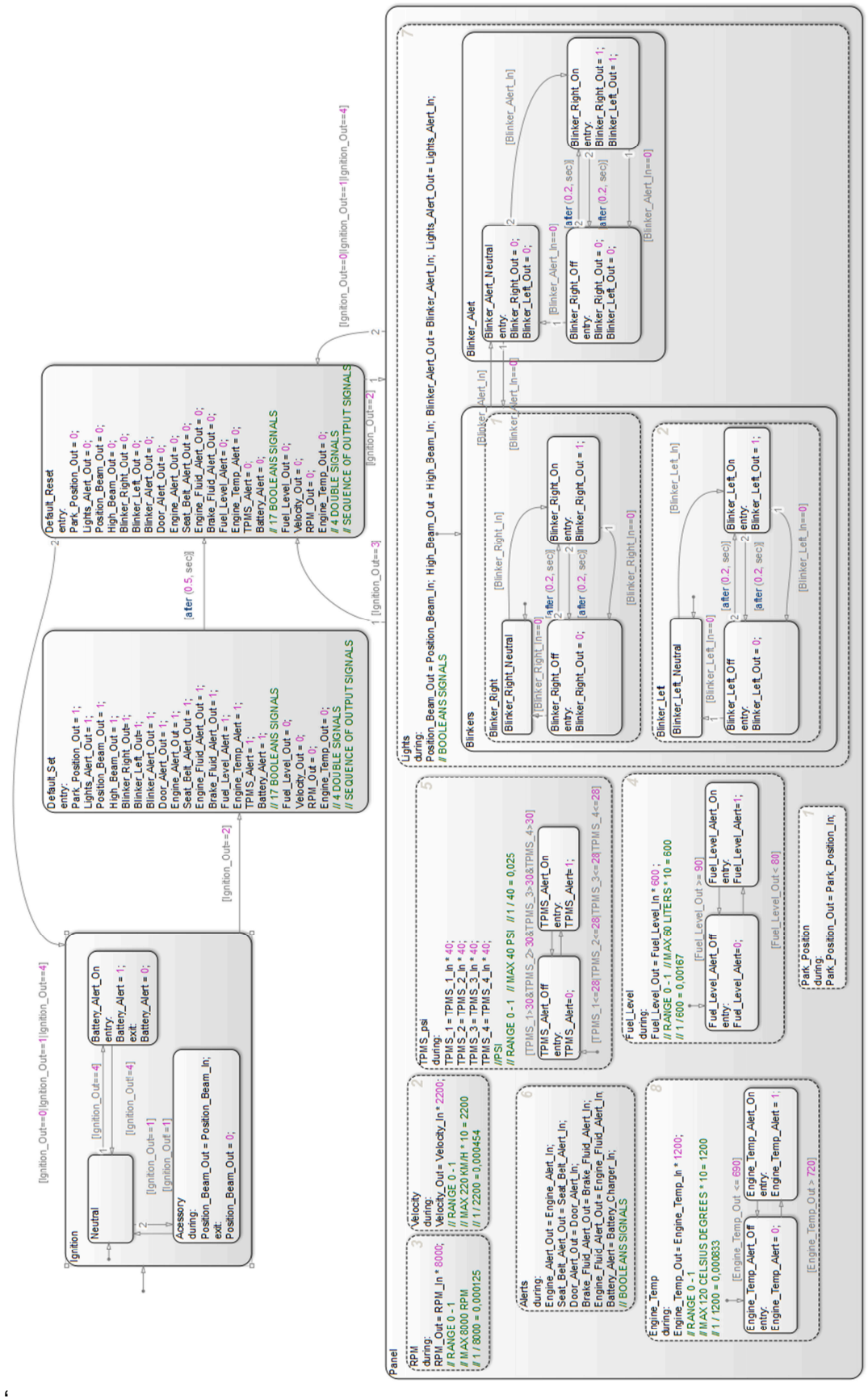


Figura 59: Diagrama *StateFlow*® do bloco *Instrument_Cluster_SS*
Fonte: Elaborado pelo Autor (2015)

O blocos da Figura 60 são responsáveis principalmente pelo gerenciamento do sinal *Ignition_Out*, que é controlado pela posição da chave de ignição e nível de carga da bateria. O estado *Ignition* é ativado quando o sinal *Ignition_Out* está em 0, 1 e 4, onde 0 e 1 representam, respectivamente, as posições *off* e *Acc* da chave de ignição, 4 representa tensão de bateria fora da faixa.

Figura 60: Diagrama *Matlab/StateFlow*® do bloco *Default_Set*, *Default_Reset* e *Ignition*
Fonte: Elaborado pelo Autor (2015)

Para o funcionamento adequado do processo há dois estados fundamentais para o gerenciamento das variáveis, o *Default_Set* e o *Default_Reset*. Esses estados gerenciam o teste das luzes do painel e a inicialização das variáveis.

Quando ativado, o painel habilita os sub estados *Panel*, que consistem nos módulos: *Velocity*, RPM, *Alerts*, *Engine_Temp*, *TPMS_psi*, *Fuel_Level*, *Park_Position*, *Lights*. O diagrama do painel de instrumentos é apresentado na Figura 61.

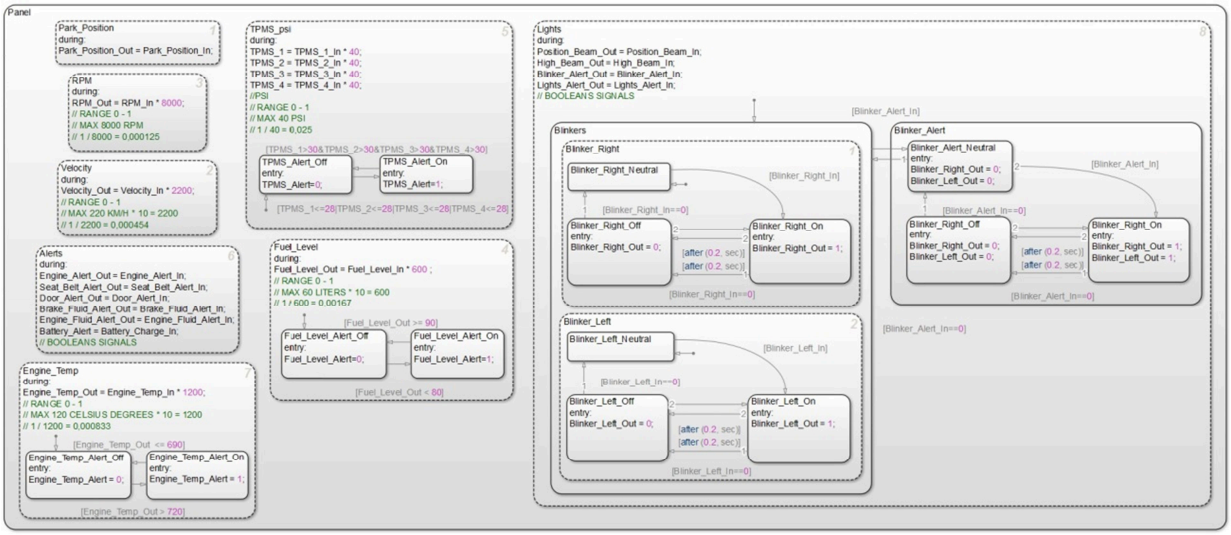


Figura 61: Diagrama parcial *Matlab/StateFlow*® do Painel de instrumentos
Fonte: Elaborado pelo Autor (2015)

Os módulos de *Velocity*, RPM, *Engine_Temp*, *TPMS_psi* e *Fuel_Level*, realizam o condicionamento dos sinais de entrada originados dos sensores, que são convertidos em valor real de 0 a 1, sendo que o valor 1 representa o maior valor possível para o sinal da entrada.

O módulo de velocidade tem seu sinal multiplicado pela constante 2200, que representa dez vezes a velocidade máxima que é de 220 Km/h. Essa constante foi determinada objetivando suavizar o deslocamento do ponteiro.

O módulo de RPM é idêntico ao de Velocidade, sendo seu sinal multiplicado pela constante 8000, que representa o valor máximo de saída (8000 rpm). Neste caso não foi necessário o método de suavização de movimentação de ponteiro utilizado no módulo de Velocidade.

No módulo de *Engine_Temp* houve a necessidade de melhorar a movimentação do ponteiro, assim a constante de tratamento do sinal é 1200, e o máximo do valor de temperatura é 120°C. Neste módulo foi incrementado um identificador de limites para acionar o alerta de quando a temperatura exceder 72ºC. Para evitar transições excessivas e incertas no acionamento do alerta de superaquecimento, colocou-se um intervalo para a desativação com valor igual ou abaixo de 69ºC.

No módulo de *TPMS_psi* não houve a necessidade de aumentar a constante de tratamento pois o módulo apenas gerenciará um alerta sem a utilização de ponteiros no painel. Assim a constante foi fixada como 40, o máximo sinal de pressão dos pneus em psi. O alerta de baixa pressão dos pneus será ativado

quando o valor de um dos quatro sensores de pressão ficar igual ou abaixo de 28 psi e será desativado quando todos os quatro sinais forem maiores que 30 psi.

No módulo de *Fuel_Level* foi necessário suavizar a movimentação do ponteiro do painel, assim a constante de tratamento foi definida como 600. O alerta de pouco combustível será ativado quando ficar abaixo de 80 (8L) e será desativado quando ficar igual ou maior a 90 (9L). O valor máximo do sinal de saída é 600, que representa 60 litros.

Os demais módulos consistem em tratamento de sinais booleanos de entrada. Compreendem nos módulos de: *Alerts*, *Park_Position*, *Lights*.

Os módulos *Alerts* e *Park_Position* apenas verificam os sinais booleanos e repassa-os para os sinais de saída.

O módulo *Lights* consiste em verificações dos sinais booleanos para a ativação das lâmpadas. O estado *Blinker_Alert* necessita de sincronia das quatro lâmpadas de alerta. Assim, este estado controla os dois sinais de saída *Blinker_Right_Out* e *Blinker_Left_Out*.

Os demais estados *Blinker_Right* e *Blinker_Left* controlam os sinais das lâmpadas da direita e da esquerda isoladamente.

3.3.2.5 Teste e validação da função painel de instrumentos automotivo

Na etapa de validação são aplicados sinais de entrada semelhante a casos reais para ser possivel analisar as saídas, verificando se atendem aos requisitos das funções. Devido ao grande número de entradas e saídas, os sinais foram divididos em grupos para facilitar a verificação.

No diagrama temporal da Figura 62 são apresentados os sinais *Acc*, *Run* e *Crank* que correspondente à sequência de rotação da chave de ignição, o sinal *Batt* proveniente da tensão da bateria, com os valores do sinal de saída *Ignition_Out*, pode-se validar a tabela verdade do bloco *Ignition_Position_SS*. Apesar desse sinal ser interno ao controlador, foi necessário a repetição nos próximos gráfico para validar os sinais dependentes dele.

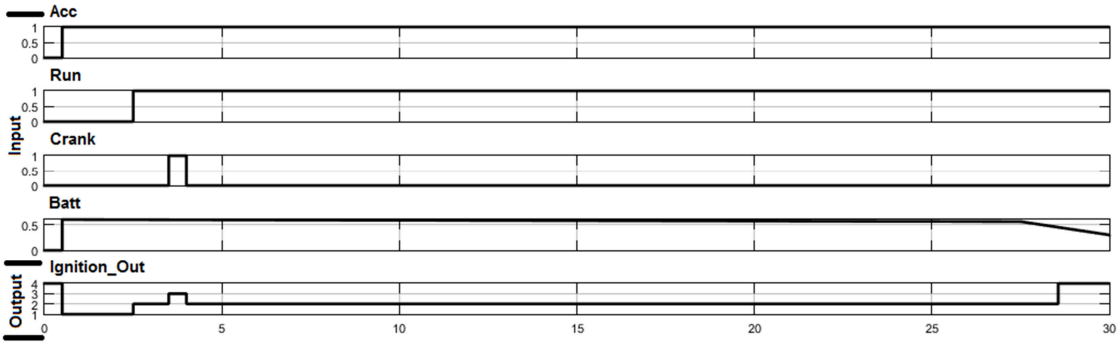


Figura 62: Gráfico para validação do bloco *Ignition Position_SS*.
Fonte: Elaborado pelo Autor (2015)

Nos próximos diagramas temporais serão apresentados os vários sinais de saída para validação, entre eles os sinais de lâmpada, onde observa-se que existe um nível alto após a transição de zero para um do sinal *Ignition_Out*, e com isso pode-se validar o teste de lâmpada do IC.

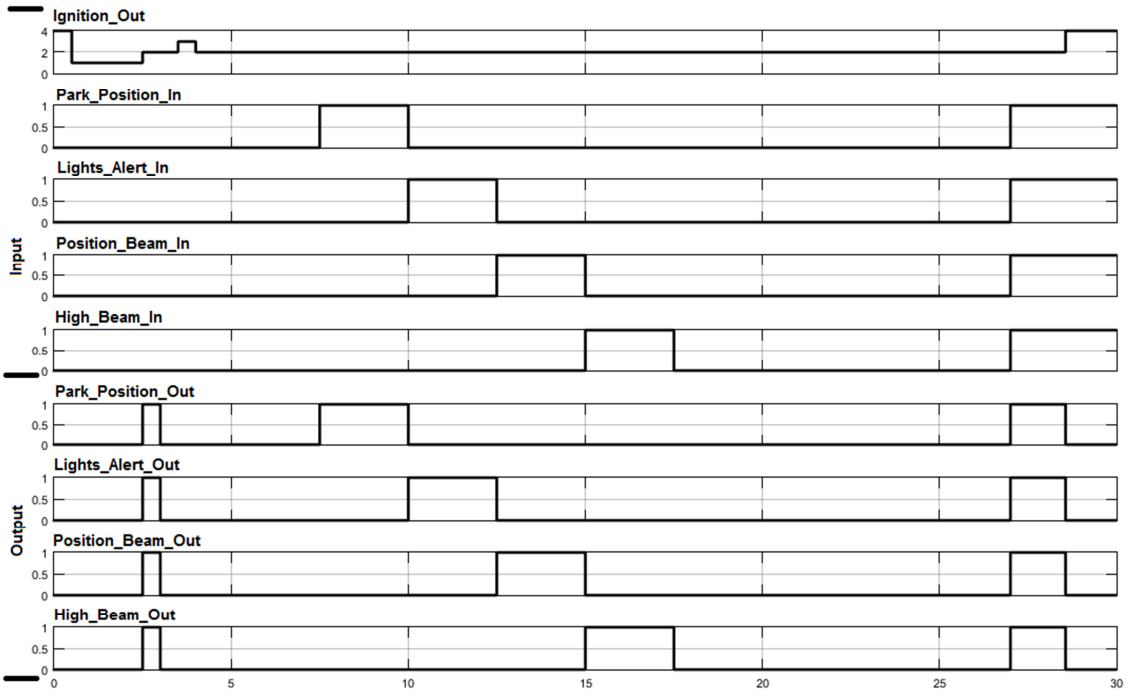


Figura 63: Gráfico para validação dos sinais *Park_Position_Out*, *Lights_Alert_Out*, *Position_Beam_Out* e *High_Beam_Out*.
Fonte: Elaborado pelo Autor (2015)

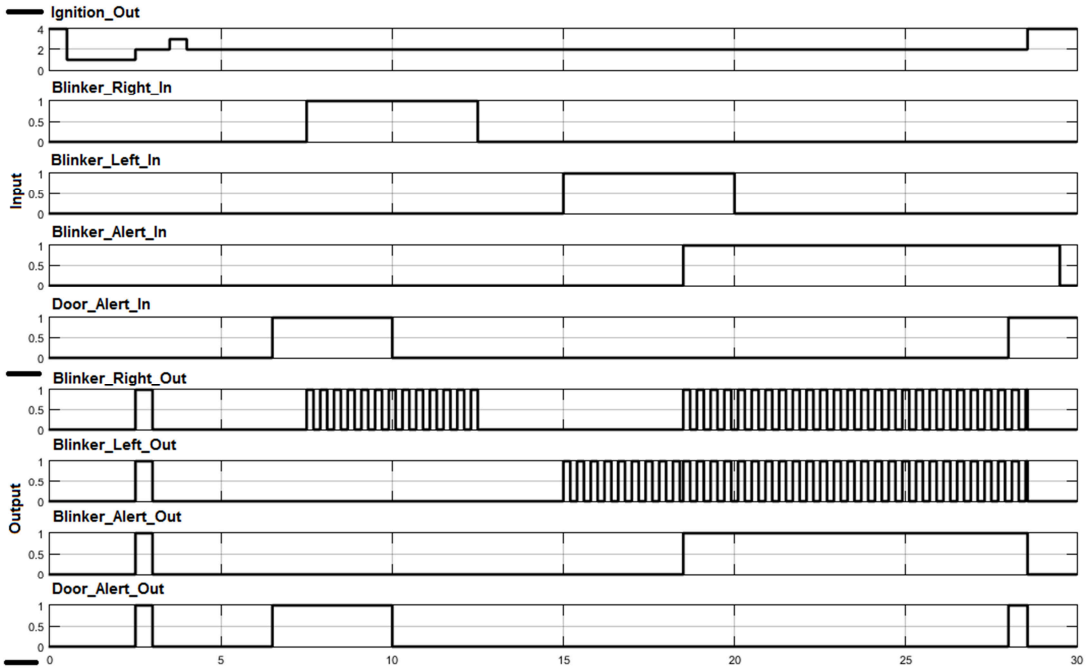


Figura 64: Gráfico para validação dos sinais *Blinker_Right_Out*, *Blinker_Left_Out*, *Blinker_Alert_Out*, e *Door_Alert_Out.*
Fonte: Elaborado pelo Autor (2015)

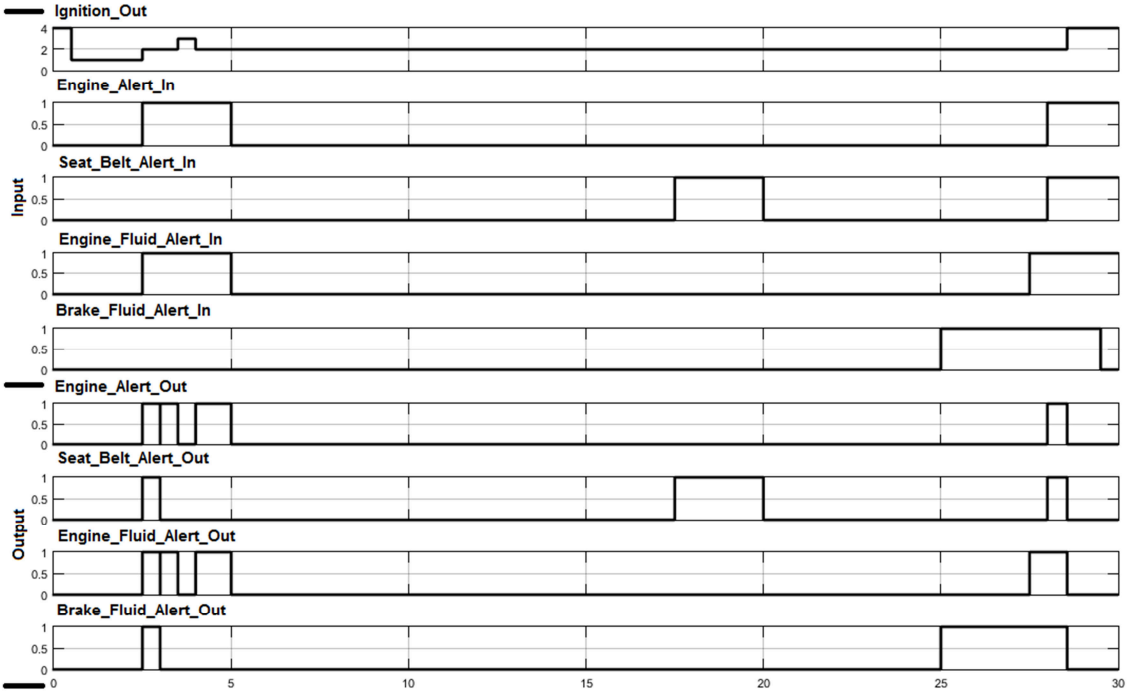


Figura 65: Gráfico para validação dos sinais *Engine_Alert_Out*, *Seat_Belt_Alert_Out*, *Engine_Fluid_Alert_Out* e *Brake_Fluid_Alert_Out*.
Fonte: Elaborado pelo Autor (2015)

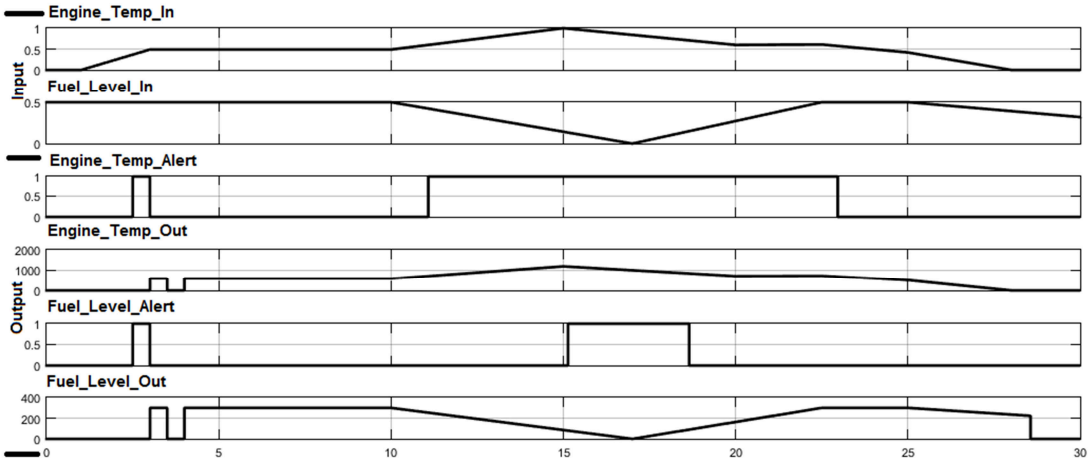


Figura 66: Gráfico para validação dos sinais *Engine_Temp_Alert*, *Engine_Temp_Out*, *Fuel_Level_Alert* e *Fuel_Level_Out*.
Fonte: Elaborado pelo Autor (2015)

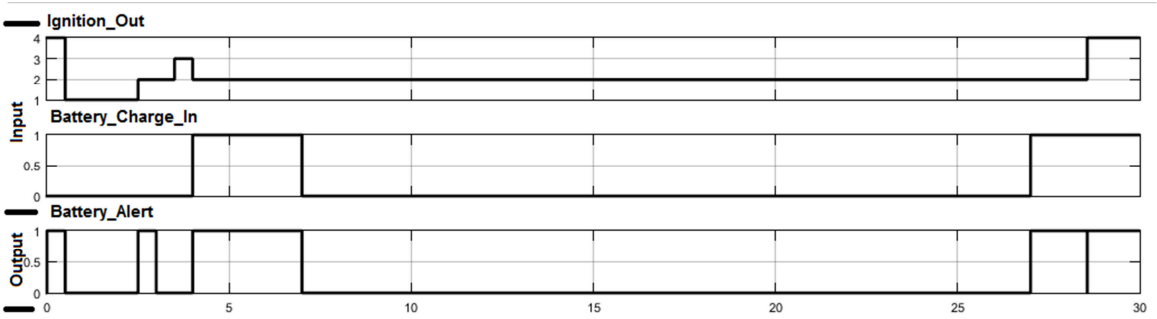


Figura 67: Gráfico para validação do sinal *Battery_Alert*.
Fonte: Elaborado pelo Autor (2015)

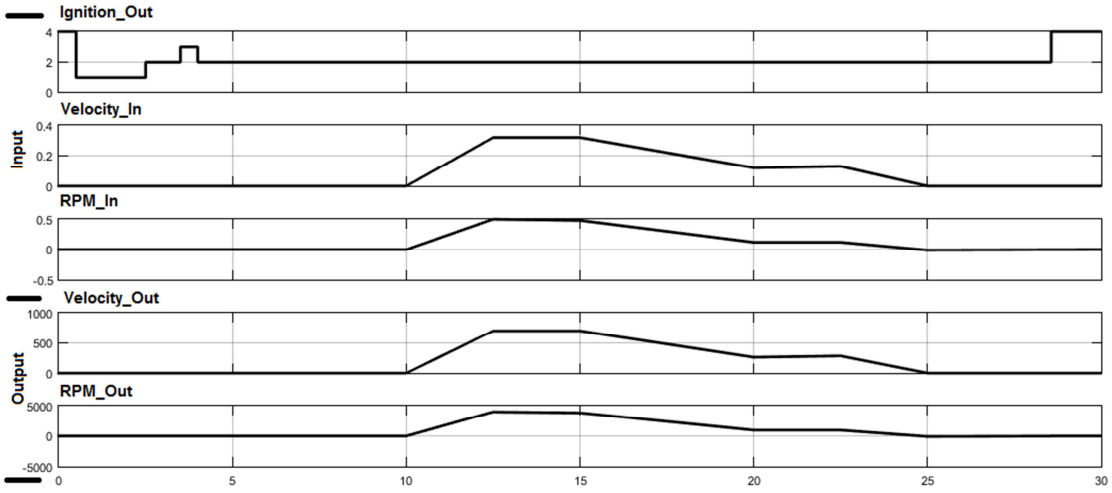


Figura 68: Gráfico para validação dos sinais *Velocity_Out* e *RPM_Out*,
Fonte: Elaborado pelo Autor (2015)

3.3.3 *Software-In-The-Loop* para um Painel de Instrumentos Automotivo

O modelo validado na etapa anterior é utilizado, para gerar-se o código para etapa SIL que ficará embutido em um bloco do *Simulink*® mostrado na Figura 69, onde também foi incluído os blocos para geração dos sinais de validação.

Para evidenciar melhor os resultados, os sinais de saídas foram divididos em grupos e apresentados da Figura 70 à Figura 74, comparando os sinais do MIL e SIL, validando esse passo da metodologia.

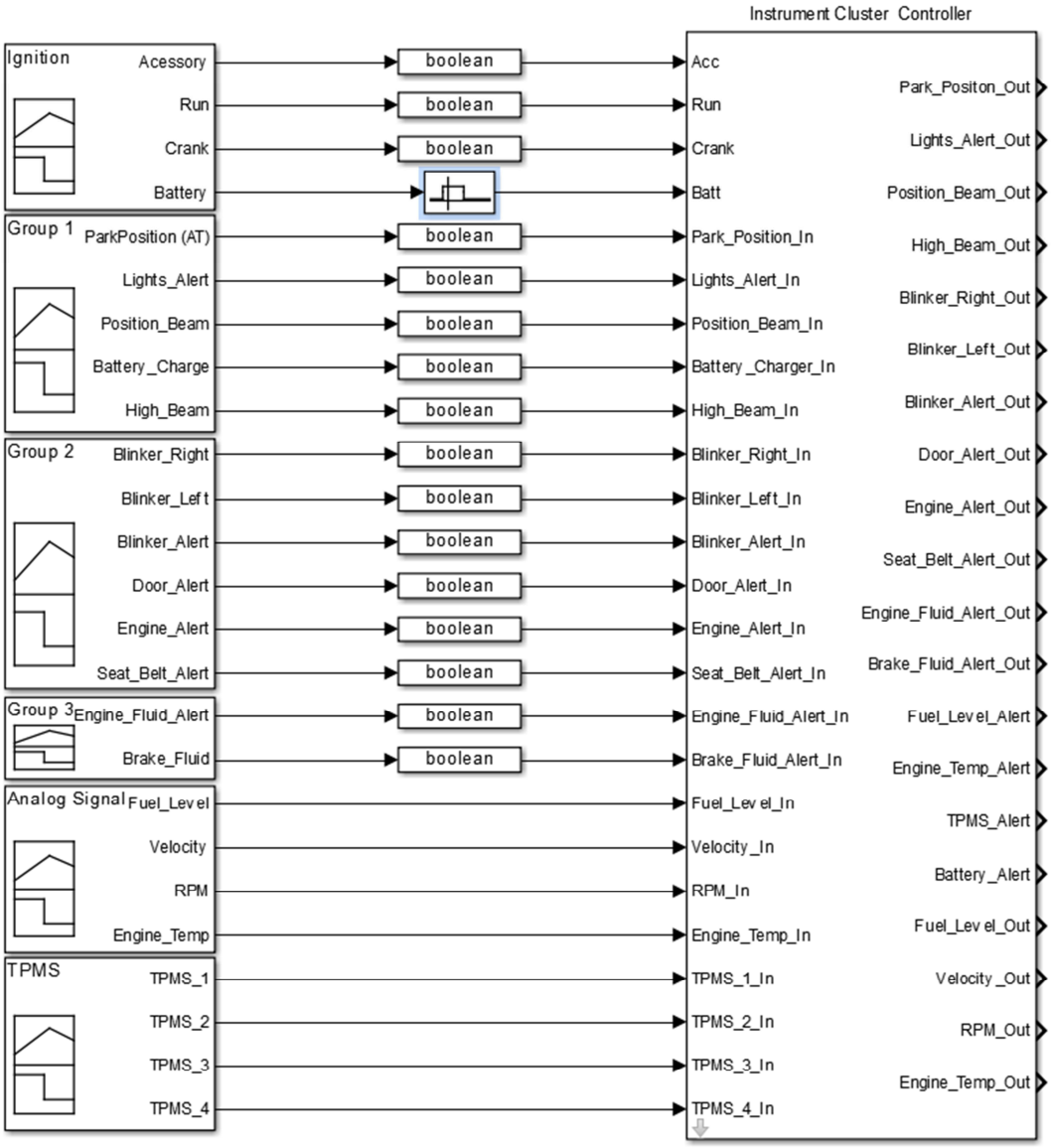


Figura 69: Diagrama *Matlab/Simulink*® do controlador da função IC para o SIL
Fonte: Elaborado pelo Autor (2015)

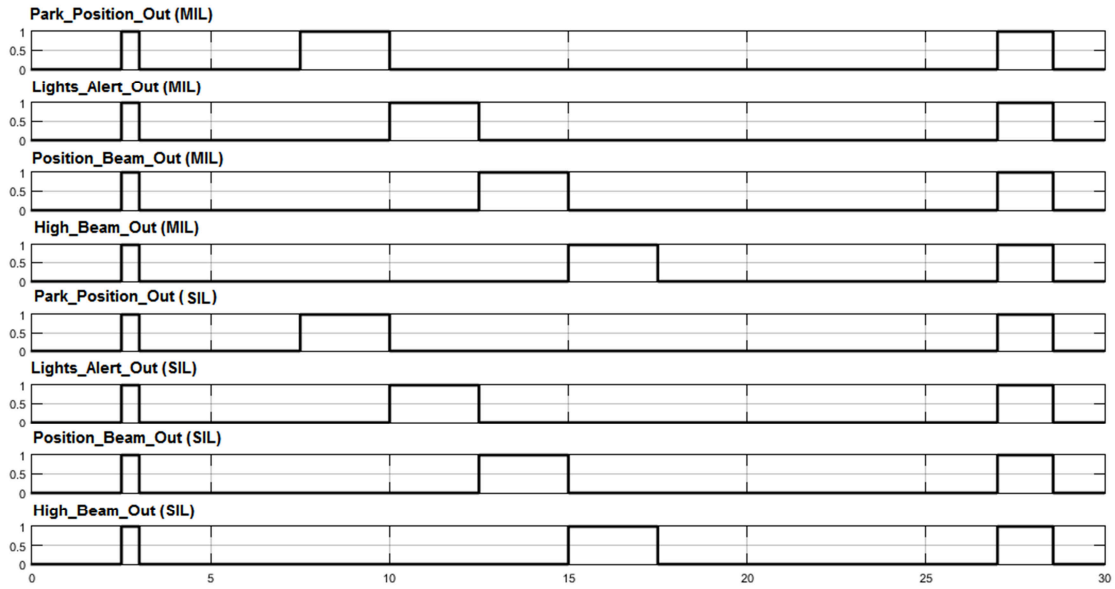


Figura 70: Gráfico para validação da etapa SIL dos sinais *Park_Position_Out*, *Lights_Alert_Out*, *Position_Beam_Out* e *High_Beam_Out*.
Fonte: Elaborado pelo Autor (2015)

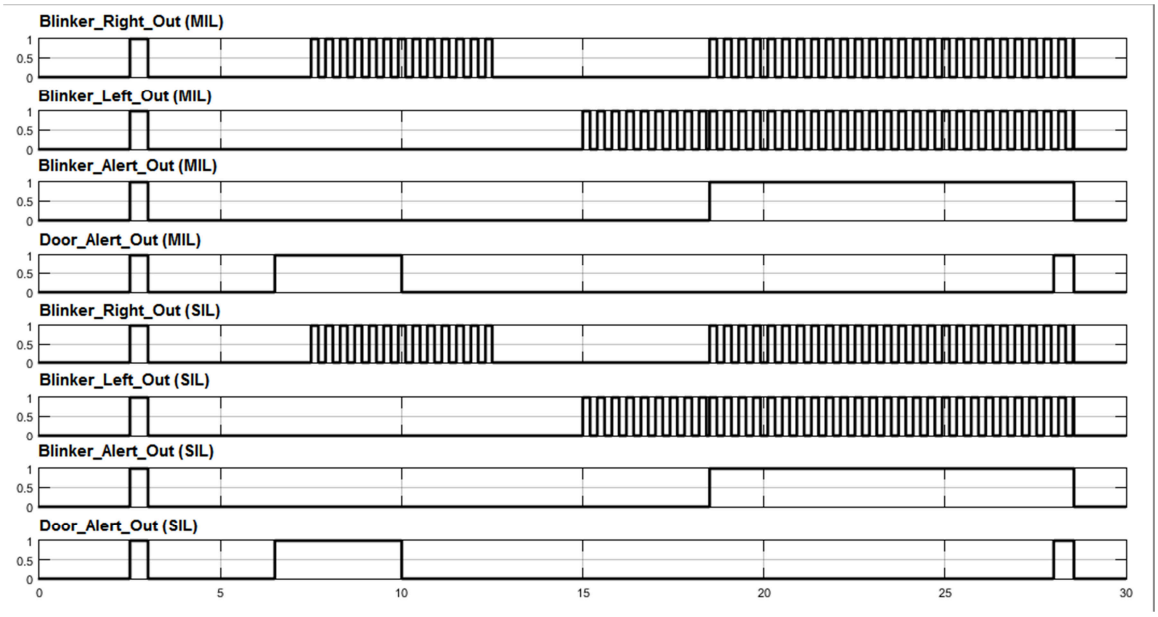


Figura 71: Gráfico para validação da etapa SIL dos sinais *Blinker_Right_Out*, *Blinker_Left_Out*, *Blinker_Alert_Out*, e *Door_Alert_Out*.
Fonte: Elaborado pelo Autor (2015)

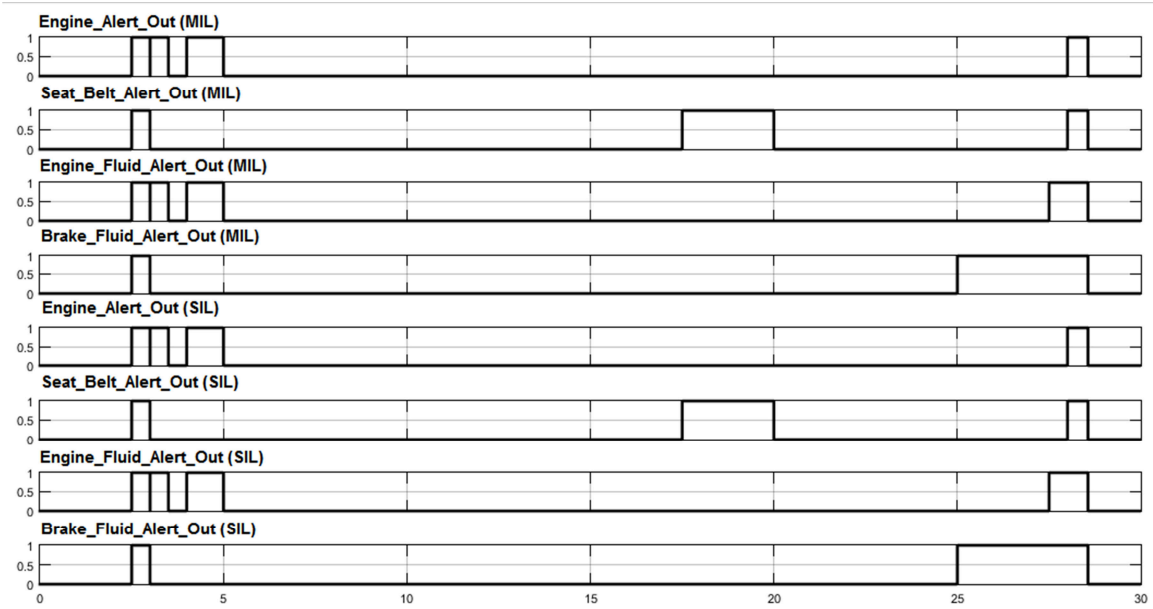


Figura 72: Gráfico para validação da etapa SIL dos sinais Engine_Alert_Out, Seat_Belt_Alert_Out, Engine_Fluid_Alert_Out e Brake_Fluid_Alert_Out.
Fonte: Elaborado pelo Autor (2015)

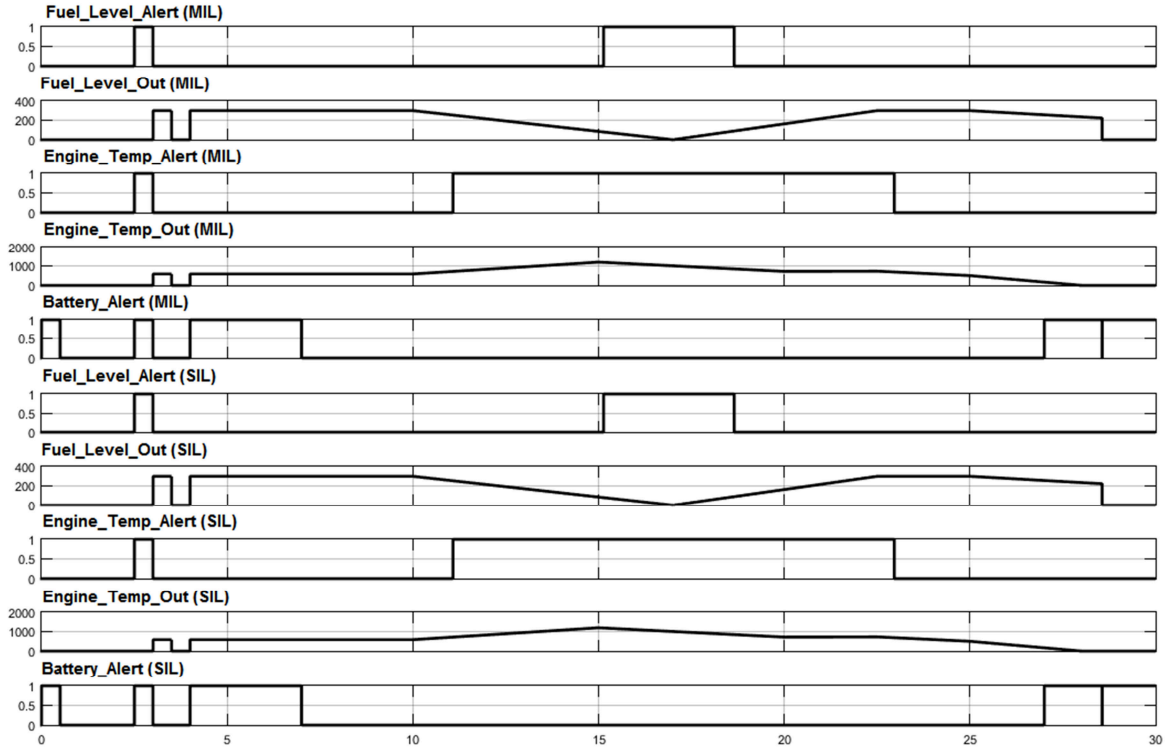


Figura 73: Gráfico para validação da etapa SIL dos sinais *Fuel_Level_Alert*, *Fuel_Level_Out*, *Engine_Temp_Alert*, *Engine_Temp_Out* e *Battery Charge*.
Fonte: Elaborado pelo Autor (2015)

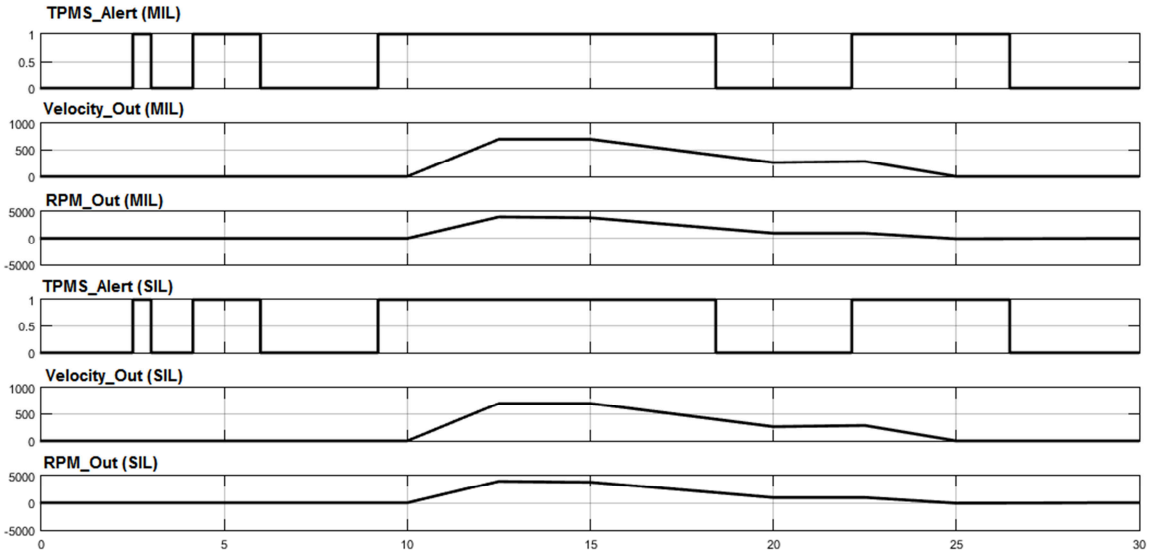


Figura 74: Gráfico para validação da etapa SIL dos sinais *TPMS_Alert*, *Velocity_Out* e *RPM_Out*,
Fonte: Elaborado pelo Autor (2015)

3.4 *PROCESSOR-IN-THE-LOOP*

Prosseguindo com os termos propostos pela metodologia, passa-se para etapa PIL, onde é necessária a escolha de um *hardware* para embarcar o modelo desenvolvido e validado nas etapas anteriores. Foram analisadas várias possibilidades e entre elas foi escolhida a placa de avaliação FRDM-KL25Z da Freescale pelos seguintes motivos: disponibilidade no mercado - garantida pelo fabricante; baixo custo de aquisição; desempenho do *hardware*; e existência de biblioteca para *Matlab®/Simulink®.*

No decorrer dos estudos foi necessário o desenvolvimento de novas bibliotecas do *Matlab®/Simulink®* para o controle de servos motores dos mostradores do painel e conversão de escala.

3.4.1 Descrição do Hardware para o PIL

Para o teste desta etapa, é necessário gerar estímulos elétricos de entrada e captar os resultados na saída em um *hardware*. Para poder-se gerar resultados que condizem com os passos anteriores, foi necessária a geração e captação de vários sinais elétricos simultâneos. Por essa razão, as ferramentas de *software* do *Matlab®/Simulink®* (*Signal Builder* e *Scope*) sozinhas não foram suficientes. Portanto

foram utilizados adicionalmente equipamentos como: osciloscópio com analisador lógico, DAQ e *myRIO* da *National Instruments*®. Os dois últimos tornam possível gerar e ler os sinais ao mesmo tempo.

Na coleta dos dados, para gerar os gráficos de validação dessa etapa, foi usado o equipamento myRIO para digitalização dos sinais junto com *Simulink*® para produzir as formas de ondas necessárias, geração automática dos códigos necessários e gravação na flash da ECU.

Usando o *Matlab*/*Simulink*®, foram construídos os códigos em linguagem C/C++, com *Embedded Coder*®, bibliotecas *ARMmbed* e CMSIS, e compilador *gcc arm GNU*. Após o código ser compilado é gravado na memória flash da ECU destino através de interface *OpenSDA* disponível na placa.

Na geração dos sinais de teste recorreu-se ao *Signal Builder*® do *Simulink*® para exportar a forma de onda para um arquivo no formato de valores separados por vírgula (*Comma Separed Values* – CSV). Com a ferramenta *Labview*® da *National Instruments*®, esses arquivos são lidos pela função (mostrada na Figura 75) e convertidos em sinais elétricos equivalentes os mais próximos possíveis aos usados nas etapas anteriores.

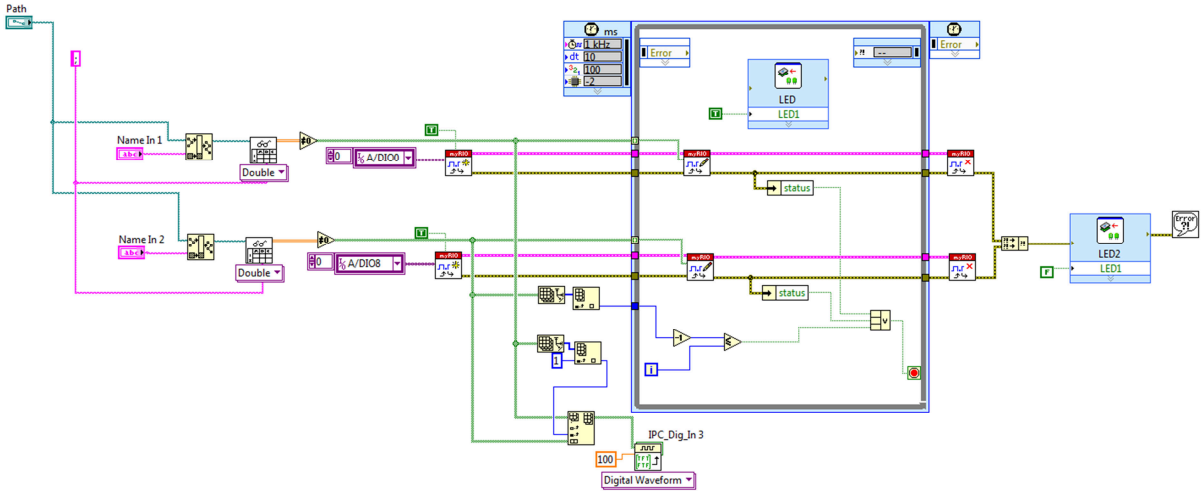


Figura 75: Função, em *Labview*®, que gera os sinais na entrada do PIL.
Fonte: Elaborado pelo Autor (2015)

A aquisição dos sinais de saídas foi realizada pela função *Labview*® mostrada na Figura 76, que são lidos todos simultaneamente. A função que faz a aquisição de sinal é comandada por um sinal proveniente da função que gera o sinal de entrada. Esse sinal permanece em nível alto durante todo o processo de geração.

Após o período de leitura, os dados são gravados em um arquivo CSV e gerados os gráficos dos sinais de saída.

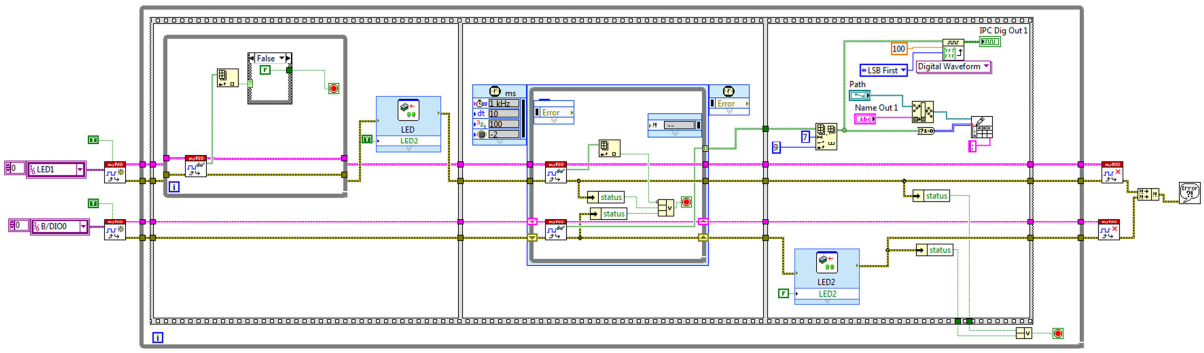


Figura 76: Função, em *Labview*®, que captura sinais nas saídas do PIL.
Fonte: Elaborado pelo Autor (2015)

A Figura 77 mostra o fluxograma das atividades realizadas para a elaboração dos testes dos dois modelos estudados.

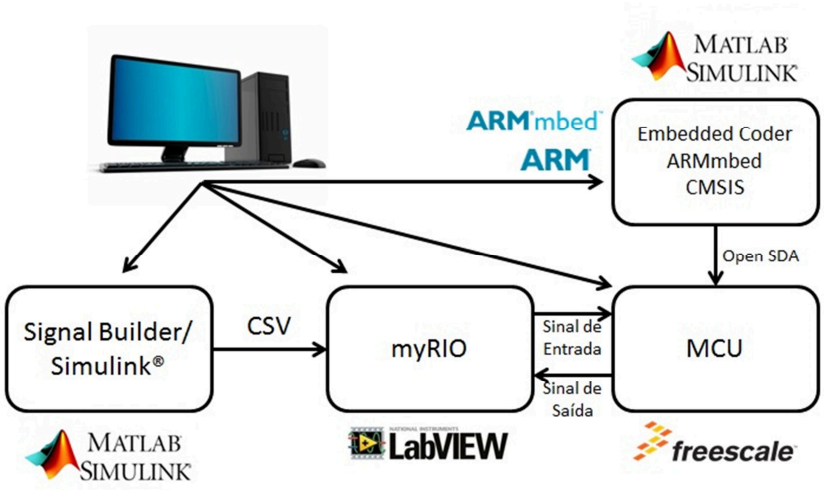


Figura 77: Fluxograma de atividades para as funções PIL.
Fonte: Elaborado pelo Autor (2015)

O *hardware* usado para esta conversão foi o NI myRIO, baseado em um processador *ARM*® *Cortex*™-A9 dual-core e as E/S customizáveis do Xilinx FPGA.

Figura 78: Foto do myRIO.
Fonte: National Instruments (2015).

Nos trabalhos realizados foi necessário desenvolver bibliotecas para o *Matlab/Simulink*®, que são compostas de funções em linguagem nativas do *Matlab/Simulink*® responsáveis por fazer as chamadas das rotinas na ECU de destino e o código das novas rotinas em linguagem C ou C++ que serão executadas na ECU de teste. No *Matlab/Simulink*®, o uso de bibliotecas é representado pela inclusão de um bloco no diagrama, que faz a chamada das funções da biblioteca. Nesse trabalho, foram desenvolvidas bibliotecas de conversão de escala e de controle de servo-motores para a função do painel de instrumentos, que serão apresentadas na seção 3.4.3.

Para conversão de escala a biblioteca foi escrita em linguagem nativa *Matlab*®, onde temos como entradas o ângulo máximo de mostrador, podendo variar de 0 a 180 graus, o valor máximo do sinal e o sinal de representa a grandeza a ser mostrada, e na saída a largura de pulso que será aplicada no controle do servo-motor.

Como a biblioteca de controle do servo-motor envolve a configuração e controle de componentes (temporizador, pino de saída) internos da MCU foi escrita em duas partes:

- Em linguagem *Matlab*®, a rotina para fazer a configuração e chamadas das rotinas em linguagem C/C++ que serão executadas na MCU, esse rotina é executada na geração automática do código;
- Em linguagem C/C++, as rotinas que fazem a configuração e controle dos componentes da MCU, no caso temporizador e pinos de I/O.

Na Figura 79 pode-se observar a montagem dos equipamentos utilizados para captação dos dados, onde aparecem a FRDM-KL25Z e o myRIO.

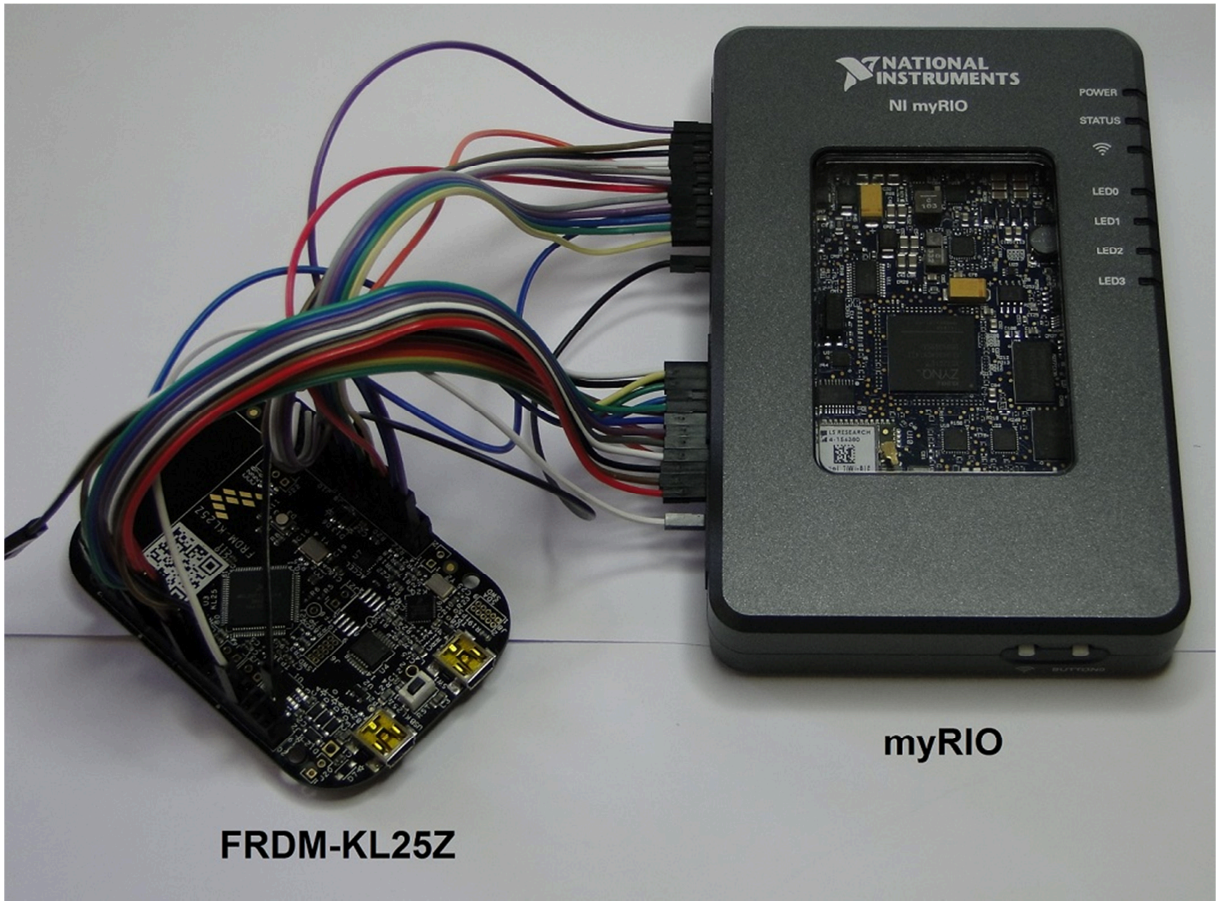


Figura 79: Foto da montagem para validação do PIL das funções ELS e IC.
Fonte: Elaborado pelo Autor (2015)

3.4.2 *Processor-In-The-Loop* para um Sistema Externo de Iluminação Automotiva

A Figura 80 mostra o diagrama do controlador das etapas anteriores (SIL) onde foram adicionados os blocos referentes às entradas e saídas da ECU alvo, visando gerar os códigos complementares para os testes físicos. Esses blocos de entradas e saídas foram desenvolvidas por Marc Herniter do Rose-Hulman Institute of Technology

Por motivo das limitações impostas pela ECU alvo (FRDM-KL25Z), como o número de pinos I/O, falta de DAC e número reduzido de ADC, não foi possível implementar todos os sinais de entrada e saídas. Esses sinais foram fixados para não gerar erros indesejáveis no teste.

O bloco *StateLED* foi adicionado para sinalizar o funcionamento do dispositivo através de um LED intermitente.

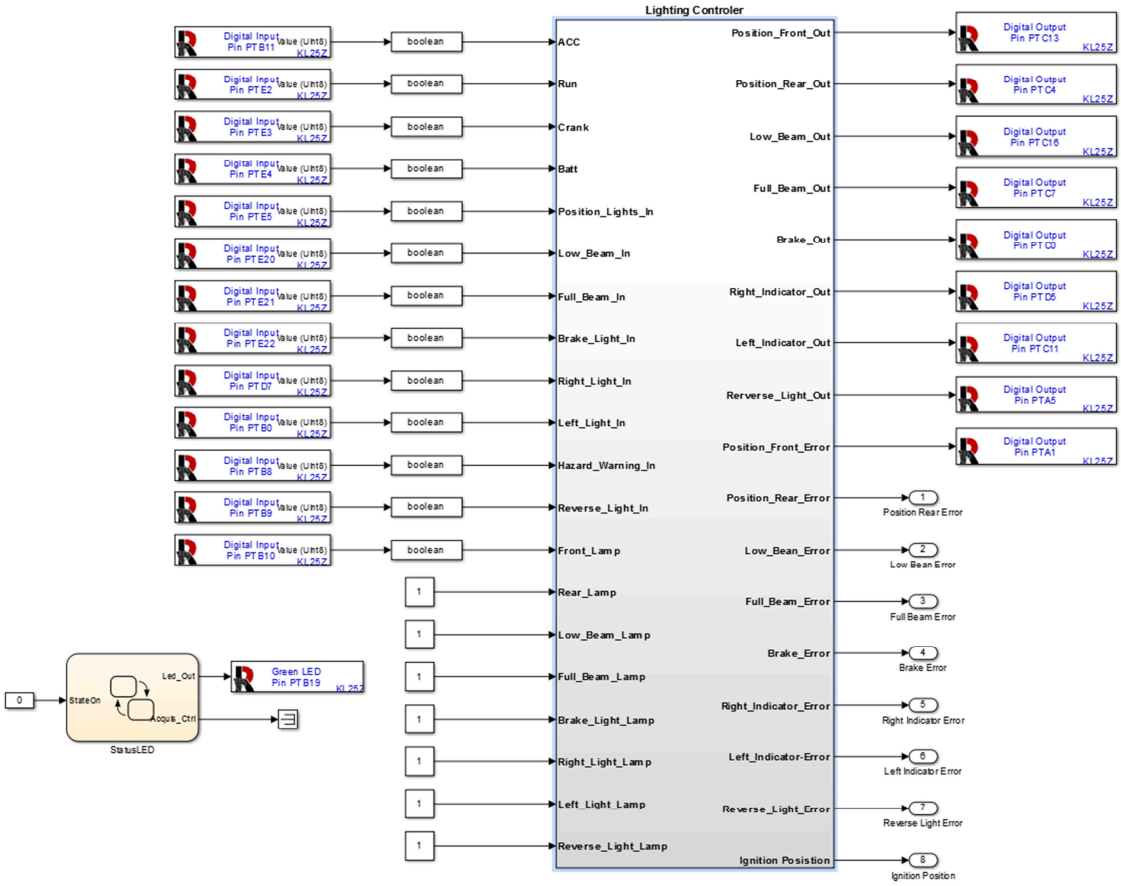


Figura 80: Diagrama *Simulink*® do controlador da função ELS para o PIL.
Fonte: Elaborado pelo Autor (2015)

A Figura 81 mostra os sinais de entrada gerados pelo myRIO para estimular a ECU alvo. É baseado nos sinais das etapas anteriores com as adequações impostas pelo *hardware*.

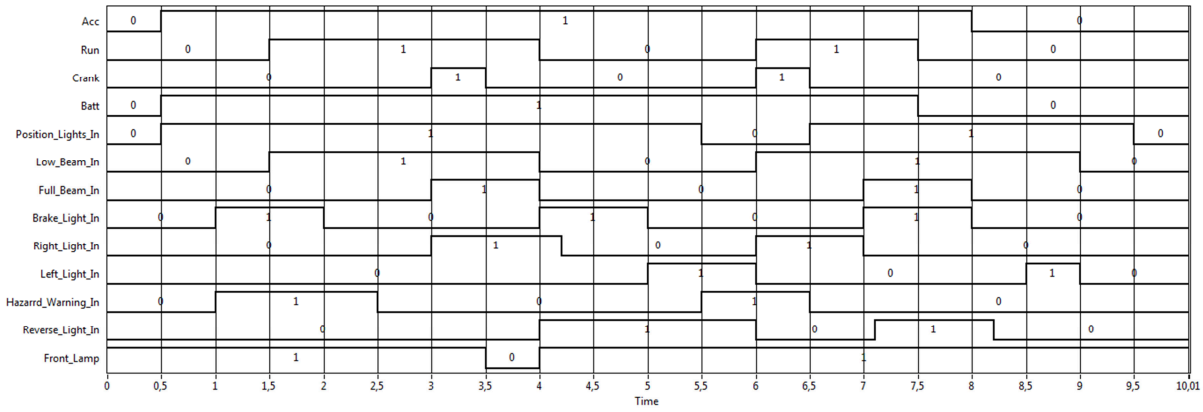


Figura 81: Gráfico para validação da etapa PIL dos sinais entradas para o ELS.
Fonte: Elaborado pelo Autor (2015)

A Figura 82 demonstra os sinais de saída da ECU destino. Os sinais seguem os sinais de entrada correspondentes, descrevendo o comportamento das lâmpadas.

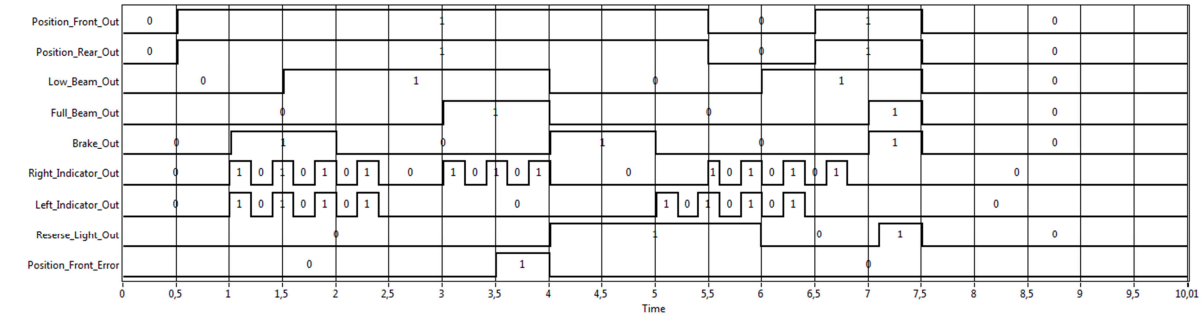


Figura 82: Gráfico para validação da etapa PIL dos sinais saídas para o ELS.
Fonte: Elaborado pelo Autor (2015)

Com o que foi apresentado acima, valida-se a função sistema de iluminação exterior automotiva.

3.4.3 *Processor-In-The-Loop* para um Painel de Instrumentos Automotivo

Visando gerar os códigos para os testes físicos, foram adicionados blocos complementares no diagrama do controlador, como mostra a Figura 83.

Da mesma forma o bloco *StateLED* foi adicionado para sinalizar o funcionamento do dispositivo através de um LED intermitente.

Os blocos sinalizados com GDS foram desenvolvidos como bibliotecas, as quais foram descritas na seção 3.4.1, sendo indicações visuais controlando os servo-motores dos mostradores. Houve uma limitação no teste devido a frequência do PWM ser maior que a frequência de captura.

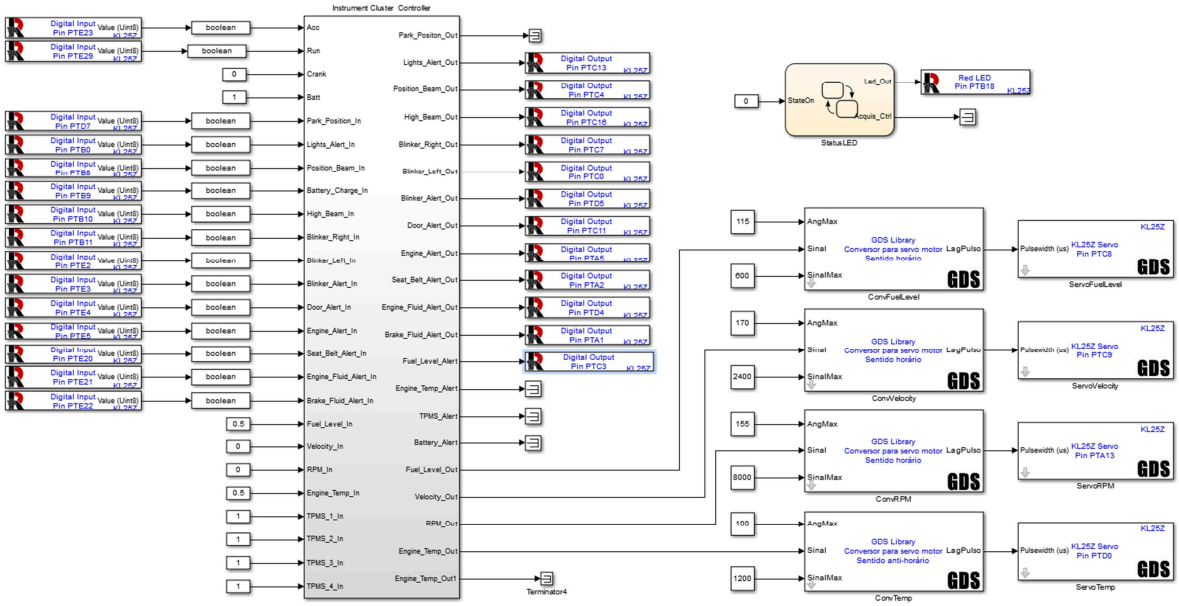


Figura 83: Diagrama *Simulink*® do controlador da função IC para o PIL.
Fonte: Elaborado pelo Autor (2015)

A Figura 84 mostra os sinais de entrada gerados pelo myRIO para estimular a ECU destino com as adequações impostas pelo *hardware*.

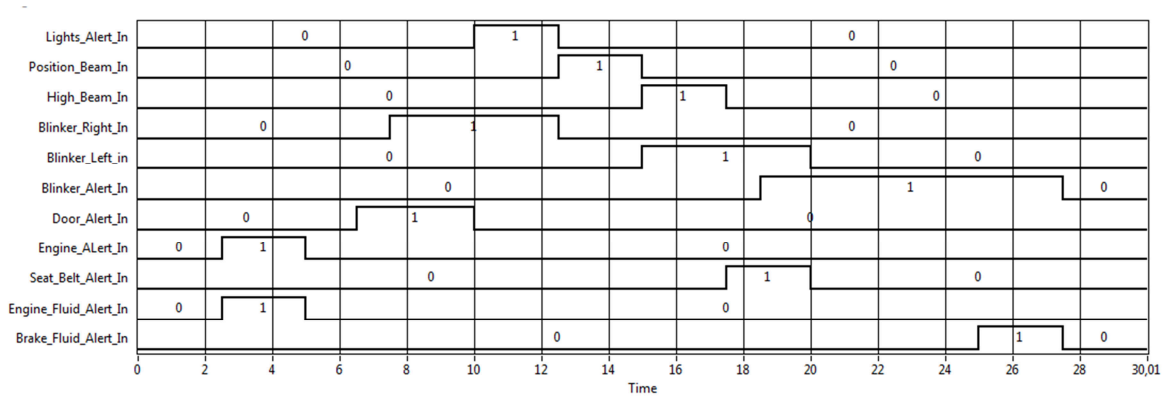


Figura 84: Gráfico para validação da etapa PIL dos sinais entradas para o IC
Fonte: Elaborado pelo Autor (2015)

A Figura 85 demonstra os sinais de saída da ECU destino. Os sinais seguem os sinais de entrada correspondentes, descrevendo o comportamento das lâmpadas.

Foi montado um painel de automóvel com as respectivas lâmpadas e mostradores para validar os sinais. A Figura 86 mostra o painel.

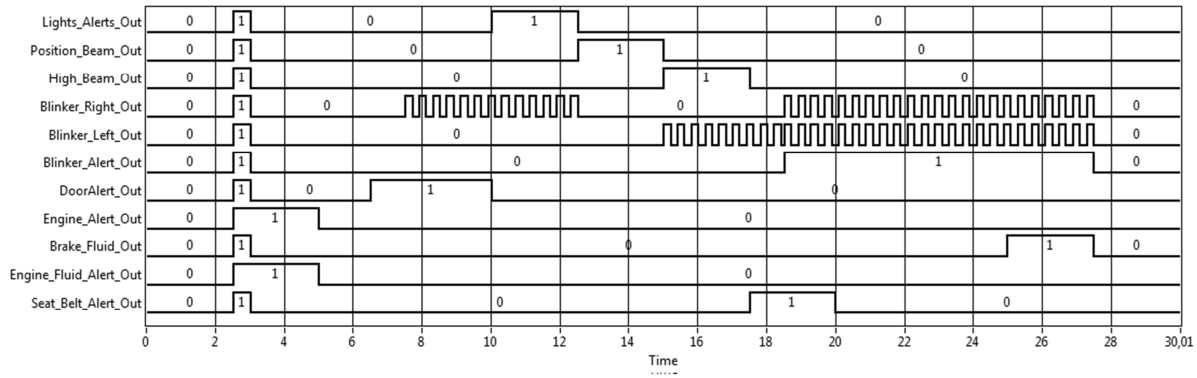


Figura 85: Gráfico para validação da etapa PIL dos sinais saídas para o IC.
Fonte: Elaborado pelo Autor (2015)

Não foi possível a coleta dos dados dos servos motores, devido a não existir a função de medida de largura do pulso no instrumento de aquisição.

Devido às dificuldades na geração dos sinais analógicos e aquisição dos dados relativos ao servo-motores foi construído um painel de instrumentos físico (Figura 86), que foi utilizado para validar visualmente o funcionamento dessa função.

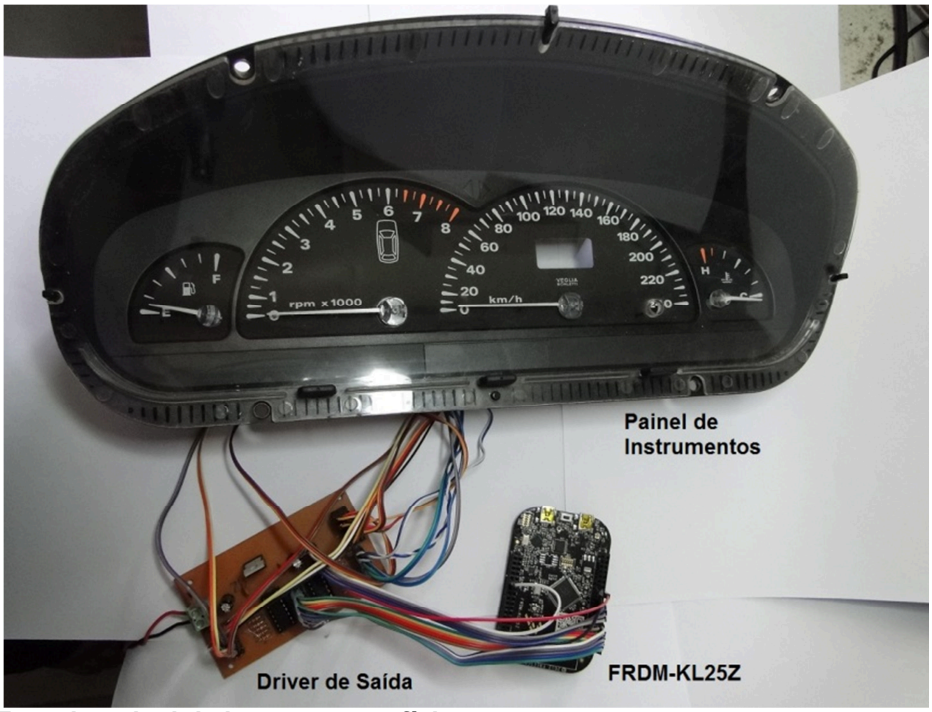


Figura 86: Foto do painel de instrumentos físicos.
Fonte: Elaborado pelo Autor (2015)

Com os testes apresentados acima, valida-se o painel de instrumentos automotivo.

4 RAPID CONTROL PROTOTYPING (RCP)

A etapa RCP é uma sequência das etapas MIL e SIL, já apresentadas anteriormente. Da mesma forma que etapa PIL, é necessária a aquisição de sinais elétricos. Nesta etapa é necessário embarcar o código em um *hardware* específico que reaja como uma ECU real. O *hardware* utilizado neste trabalho foi o *MicroAutoBox*® da *dSpace*®.

Também como na etapa PIL, foi necessário adicionar os blocos de entrada e saídas digitais do *MicroAutoBox*®, modelo DS1401, e os módulos 1505 e 1507. Adicionalmente, para realizar esta etapa foi utilizada o *software TargetLink* e o *ControlDesk*.

4.1 RAPID CONTROL PROTOTYPING PARA UM SISTEMA DE ILUMINAÇÃO EXTERNA AUTOMOTIVA

A Figura 87 mostra o diagrama do controlador com os blocos de entradas e saídas alteradas.

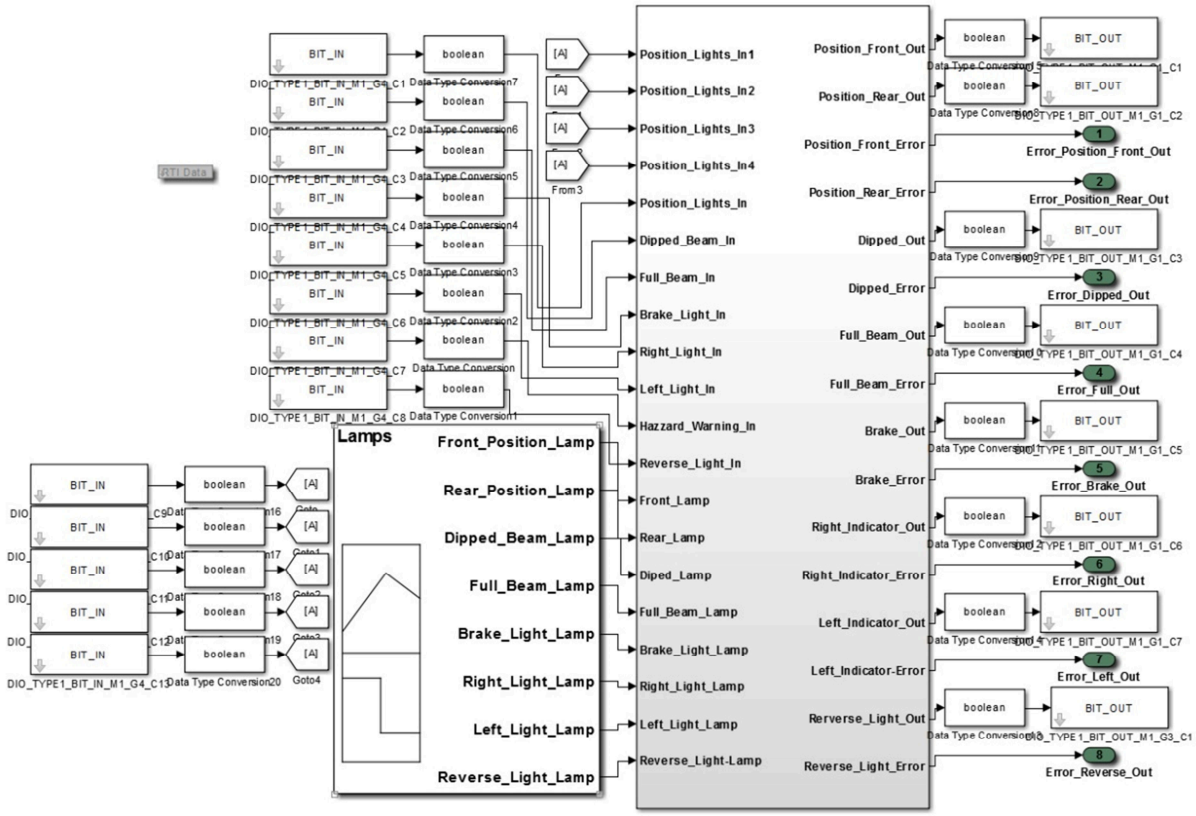


Figura 87: Entradas para a posição da chave de ignição.
Fonte: Elaborado pelo Autor (2015)

Na etapa RCP necessita-se de sinais reais, e não simulados como nas etapas MIL e SIL. Para emular esses sinais, utiliza-se o *Data Aquisition* (DAQ) e *software LabVIEW*® da *National Instruments*.

A Figura 88 mostra a montagem realizada para os testes.

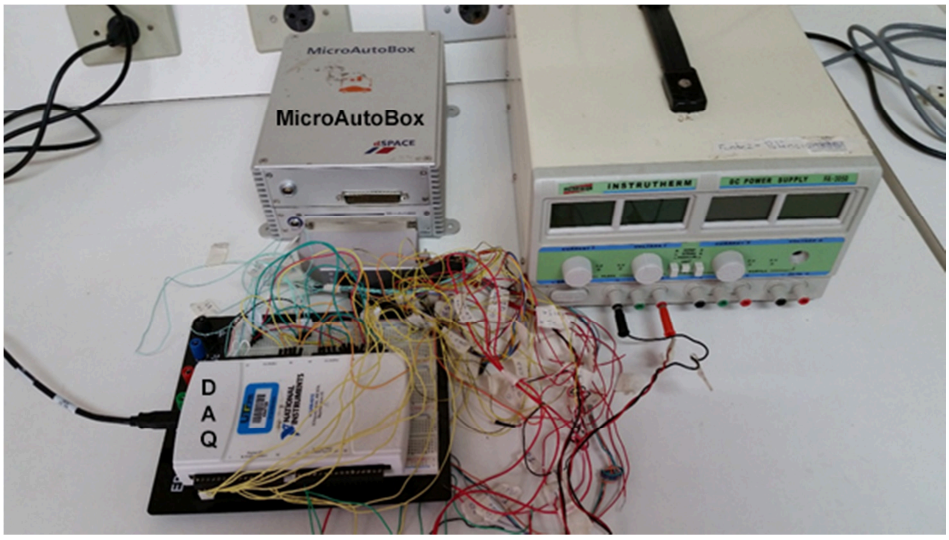


Figura 88: Ligações feitas entre o MABX e a DAQ
Fonte: Elaborado pelo Autor (2015)

Para entendimento do RCP, a Figura 89 mostra o diagrama em blocos do fluxo de informações, bem como as interligações entre seus elementos.

Figura 89: Fluxo de trabalho desenvolvido para o RCP
Fonte: Elaborado pelo Autor (2015)

Foi necessária a criação de duas funções do *LabVIEW*®, uma sendo responsável pela geração (“Gerador”) dos sinais de teste e a outra de captura de sinais, denominada de “Aquisição”. Esses sinais correspondem às entradas e saídas do *MicroAutoBox*®.

Com esses resultados, pode-se comparar o comportamento das saídas com a das outras etapas, validando o RCP.

As funções do *LabVIEW*®, foram criados de uma maneira que possam ser reutilizada, respeitando as seguintes características.

Gerador:

- Máximo de 10 saídas digitais simultâneas;
- Tempo mínimo de atualização de 100ms;
- Os sinais podem ser alterados no tempo de acordo com a configuração em um arquivo de texto.

Aquisição:

- Máximo de 16 entradas digitais simultâneas;
- Tempo de amostragem de 10ms
- Os sinais de saída podem ser exportados para arquivos com extenção “bmp” ou “CSV”.

Utilizaram-se os mesmos sinais de teste criados nas etapas MIL e SIL.

A Figura 90 mostra o diagrama temporal do comportamento desses sinais.

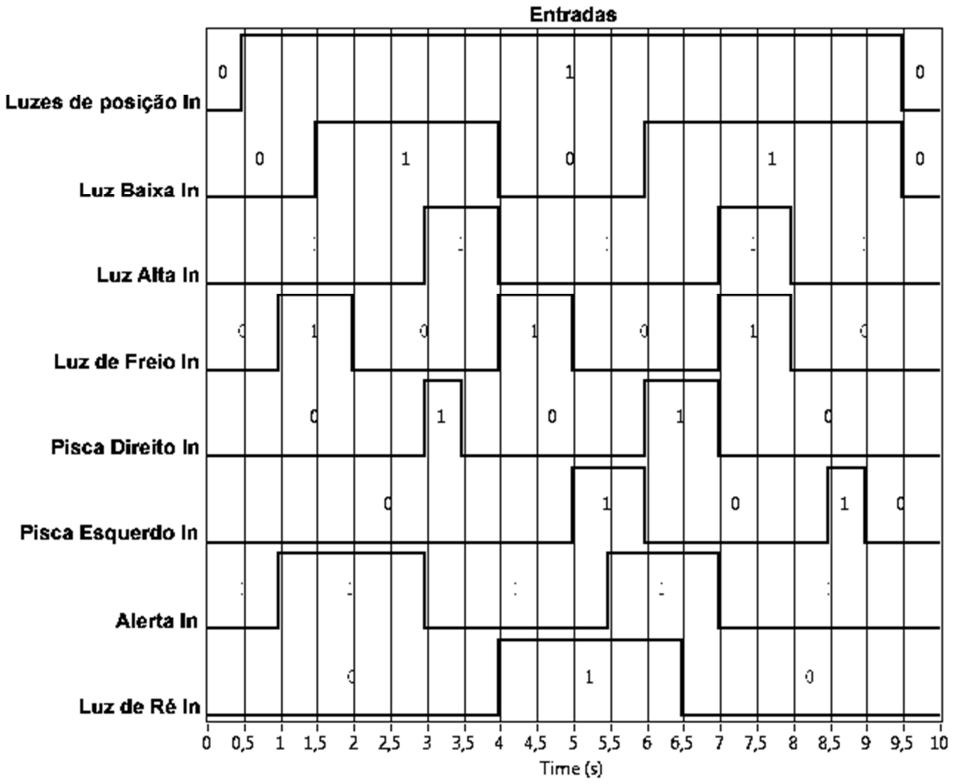


Figura 90: Entradas criadas para o caso de teste do RCP
Fonte: Elaborado pelo Autor (2015)

A Figura 91 descreve os sinais que simulam o nível da bateria e da posição da chave de ignição, necessários para habilitar o funcionamento dos outros subsistemas internos.

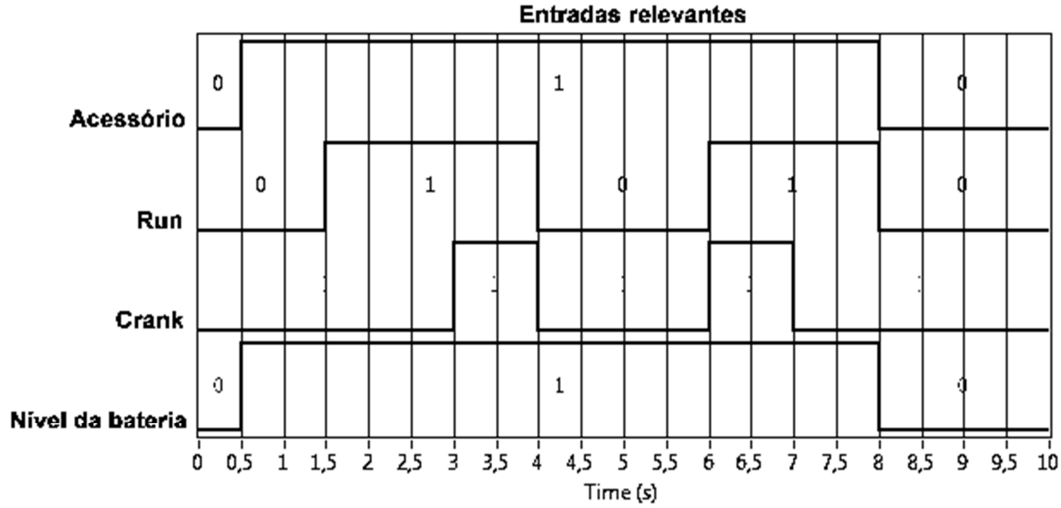


Figura 91: Sinais de emulação da posição da chave de ignição e do nível de bateria
Fonte: Elaborado pelo Autor (2015)

Com essas considerações, os dados da etapa RCP foram validados.

4.2 RAPID CONTROL PROTOTYPING PARA UM SISTEMA DE VIDRO ELÉTRICO AUTOMOTIVO

O sistema de vidro elétrico de um veículo é um subsistema constituído por um conjunto de componentes eletro-mecâncios para controlar as posição dos vidros do veículo. Esta função está alocada na BCM e pode controlar a abertura e fechamento dos vidros, contrapondo ao uso das manivelas manuais.

Esta etapa trabalho tomou por base o artigo *"Model-Based Functional Safety for the Embedded Software of Automobile Power Window"* (FRANCO et al., 2014), onde foram executadas a etapa MIL e sua validação, os requisitos dessa função estão listados.

Tabela 13: Requisitos da função de vidros elétricos.

REQ-01: O vidro deve abrir ou fechar completamente em 4 segundos.
REQ-02: O vidro deve iniciar o movimento após 200ms do comando ser acionado.
REQ-03: Após 4 segundos de movimento ininterrupto em uma única direção, o motor deve ser desligado.
REQ-04: A função de vidro elétrico é habitado quando a tensão da bateria estiver entre 12,5V e 14,5V.
REQ-05: Se o comando de abertura/fechamento do vidro elétrico estiver pressionado entre 0,2 segundos e 1 segundo, o vidro deve ser completamente fechado ou aberto.
REQ-06: Caso um objeto obstrutor seja detectado, o vidro deve descer aproximadamente 10 cm.
REQ-07: A detecção de obstáculos tem prioridade superior ao dos botões de comando.
REQ-08: O comando do motorista tem prioridade sobre os demais.
REQ-09: Quando o sistema de segurança é acionado, os vidros devem ser completamente fechados, exceto se encontrar algum obstáculo.
REQ-10: O sistema de vidro elétrico funciona somente se a chave estiver na posição *On*, *Accessory* ou *Run*.

Fonte: Elaborado pelo autor (2015)

Utilizamos os mesmos casos testes descritos naquele trabalho e descreveremos abaixo os estudos e os resultados dos testes realizados utilizando o método RCP.

- **Caso de Teste 1**

Este caso de teste procura validar as situações do REQ-01 ao REQ-05.

A Figura 92 mostra os sinais do comando de abertura (*passenger down*) e fechamento dos vidros (*passenger up*) e a tensão da bateria (*battery*).

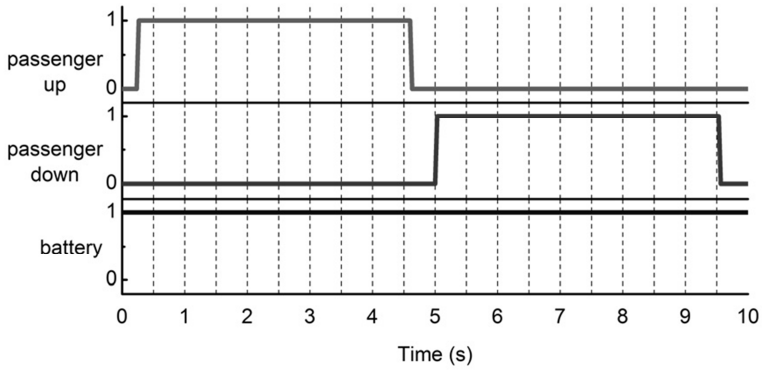


Figura 92: Gráfico dos sinais de entrada para o Caso de Teste 1.
Fonte: Elaborado pelo autor (2015)

A Figura 93 mostra a saída da função, sinalizando a posição do vidro no decorrer do tempo.

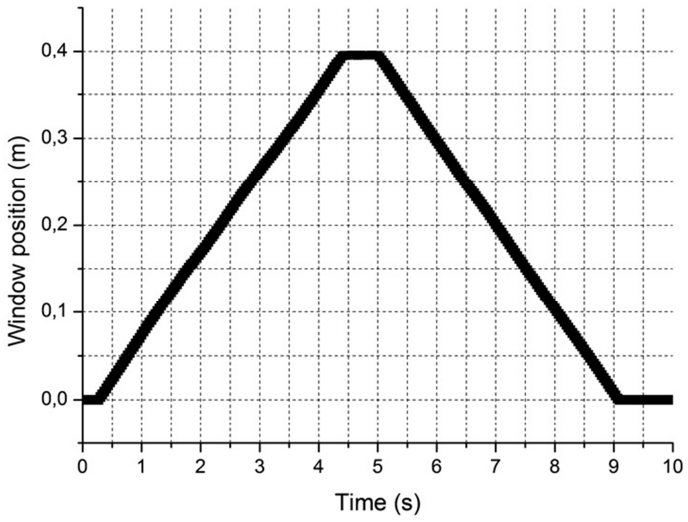


Figura 93: Gráfico dos sinais de saída para o Caso de Teste 1.
Fonte: Elaborado pelo autor (2015).

Analisando esse gráfico, verifica-se que as condições de testes descritas acima foram validadas.

- **Caso de Teste 2**

Para este caso, os sinais de entrada foram aplicados de maneira a verificar se satisfaz a situação do REQ-05 (que foi descrita no caso de teste 1), ilustrados na Figura 94. Ao analisar esta figura, pode-se verificar que o primeiro gráfico representa o comando *passenger up*, e o segundo representa o comando *passenger down* e no último o sinal da bateria.

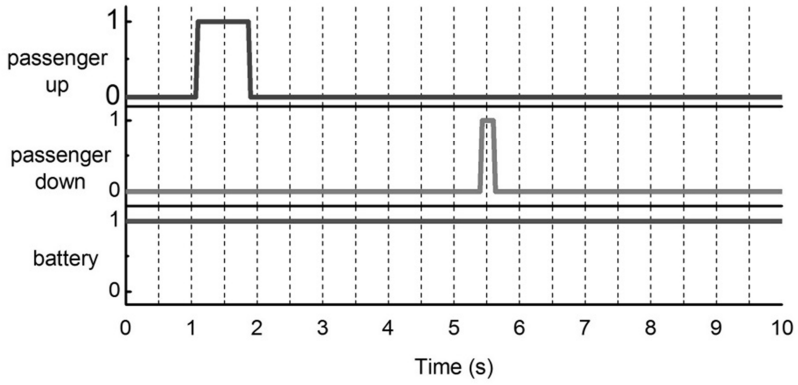


Figura 94: Gráfico dos sinais de entrada para o Caso de Teste 2.
Fonte: Elaborado pelo autor (2015)

A Figura 95 mostra a posição da janela em relação ao tempo.

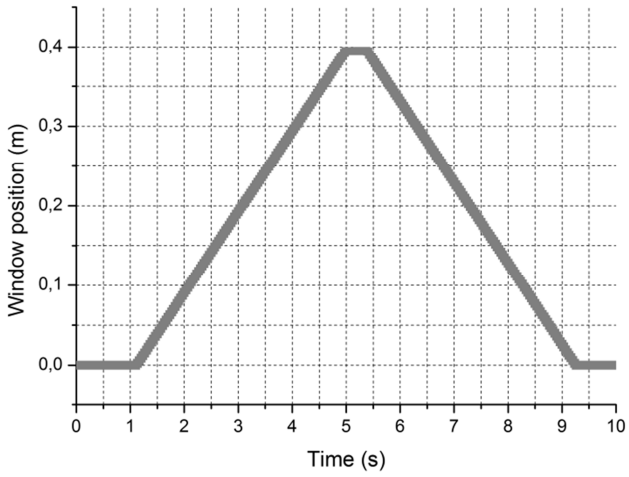


Figura 95: Gráfico dos sinais de saída para o Caso de Teste 2.
Fonte: Elaborado pelo autor (2015)

Ao verificar os gráficos de ambas as figuras, pode-se notar que o requerimento REQ-05 foi satisfeito.

- **Caso de Teste 3**

No caso 3, os gráficos de entrada são mostrados na Figura 96, a fim de confirmar se os requerimentos, apresentados a seguir, são satisfeitos. Os requisitos são: REQ-06, REQ-07 e REQ-08

Na figura a seguir, o primeiro e o terceiro sinal representam o comando de subir o vidro, o segundo e o quarto, o comando de descer o vidro, e o sexto, um obstáculo.

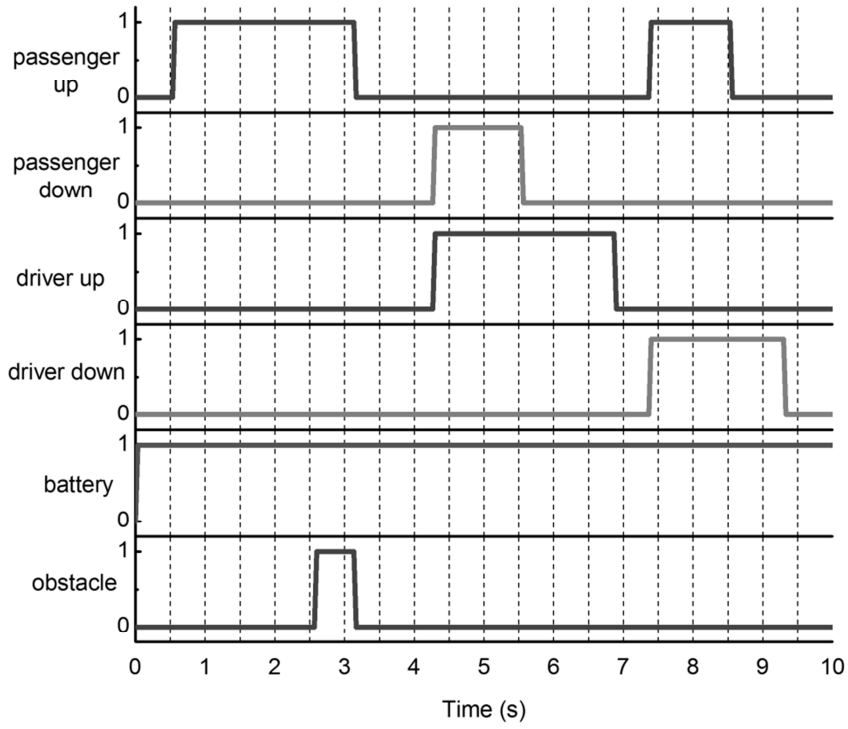


Figura 96: Gráfico dos sinais de entrada para o Caso de Teste 3.
Fonte: Elaborado pelo autor (2015)

Na Figura 97 observa-se a posição do vidro em função do tempo.

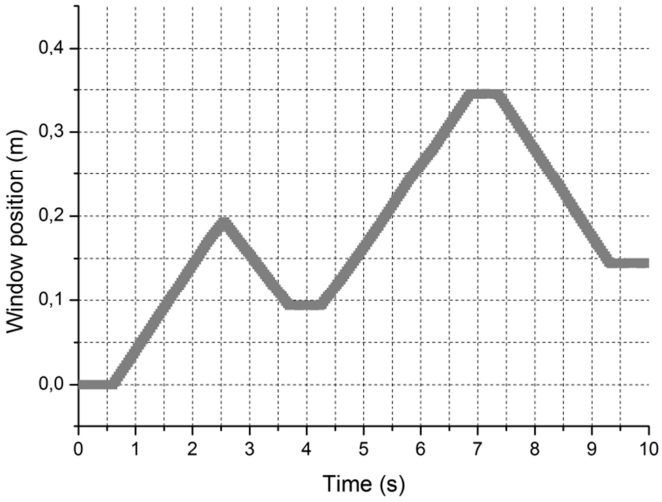


Figura 97: Gráfico dos sinais de saída para o Caso de Teste 3.
Fonte: Elaborado pelo autor (2015).

Na figura acima pode-se notar que, na presença de um obstáculo, o vidro desce, mesmo que o passageiro comande o contrário, assim satisfazendo os requisitos do REQ-06 ao REQ-08.

- **Caso de Teste 4**

Neste caso, a Figura 98 mostra as entradas dos testes para verificar se os requisitosREQ-09 e REQ-10 são satisfeitos.

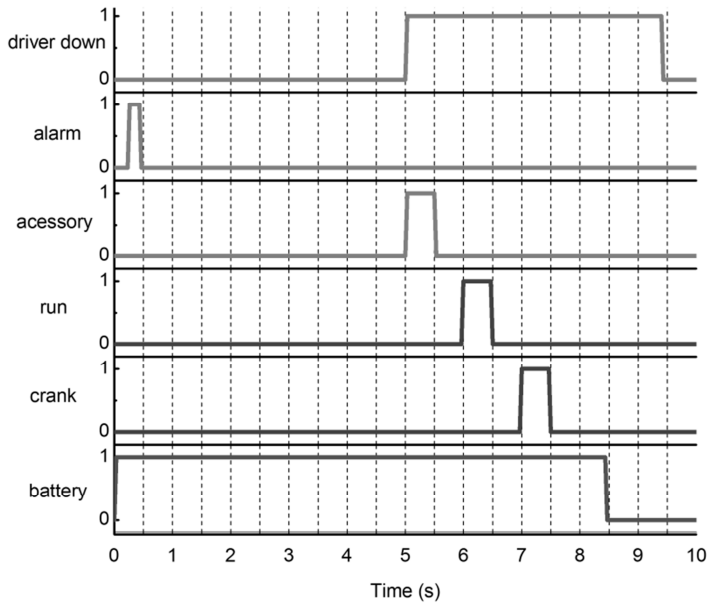


Figura 98: Gráfico dos sinais de entrada para o Caso de Teste 4.

Fonte: Elaborado pelo autor (2015).

Na Figura 99, pode-se notar que o sinal de alarme faz os vidros subirem até fechar e assim validando o REQ-09. Também pode ser observado que os vidros só se movem com a presença dos sinais *Accessory* ou *Run* e com isso valida-se o REQ-10.

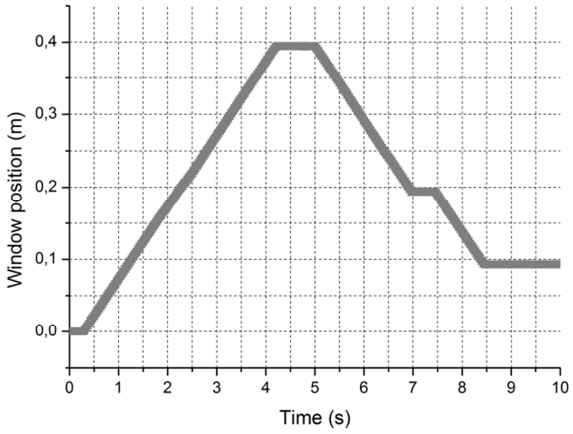


Figura 99: Gráfico dos sinais de saída para o Caso de Teste 4.
Fonte: Elaborado pelo autor (2015).

Após todos esses testes, pode-se considerar validada a função do sistema de vidros elétricos.

5 CONSIDERAÇÕES FINAIS

Através do desenvolvimento deste trabalho foi possível observar a importância da aplicação da metodologia de desenvolvimento baseada em modelos (MBD) nas funções de software automotivo.

Pode-se observar que a metodologia auxilia significativamente na produtividade dos desenvolvimentos de aplicações automotivas, de forma padronizada, otimizando o tempo dispendido com a modelagem e validação das funções do software automotivo.

As ferramentas gráficas permitiram um rápido desenvolvimento, gerando automaticamente os códigos necessários, facilitando também a manutenção dos modelos.

A metodologia se mostrou bastante eficaz na detecção antecipada dos possíveis problemas que poderiam surgir, por permitir validar cada etapa do desenvolvimento, aproveitando os dados levantados em cada etapa. Isso também permite ter a conclusão que a metodologia utiliza de forma bastante eficaz o conceito de reutilização, a confiabilidade e a segurança dos modelos.

A organização da solução apresentada no trabalho potencializa a evolução contínua das funções desenvolvidas, ou até mesmo o desenvolvimento de novas soluções, tomando como base os blocos já desenvolvidos, reutilizando-os, sendo que essa premissa é essencial para o cenário das indústrias automotivas.

Essa pesquisa pode ser expandida para o desenvolvimento destas funções utilizando também os padrões AUTOSAR e ASAM-MCD, que além das vantagens já mencionadas, adiciona uma maior padronização, abstração e interconectividade de *hardware*.

REFERÊNCIAS

ANFAVEA. **Anuário da indústria automobilística brasileira**- 2015. São Paulo, 2015.

BRASIL. Lei nº 9.503., de 23 de dezembro de 1997. **Código de Trânsito Brasileiro.**. Brasília, Disponível em: <http://www.planalto.gov.br/ccivil_03/leis/l9503.htm>. Acesso em: 25 maio 2015

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: 12 maio 2014

DEMERS, Stephanie; GOPALAKRISHNAN, Praveen; KANT, Latha. **A Generic Solution to Software-in-the-Loop**. Military Communications Conference, Orlando, Fl, EUA, p.1-6, 29 out. 2007.

FANG, Zheng et al. **Design and implementation of a low cost dsp based rapid control prototyping system**. Industrial Electronics And Applications, Xi'an, p.1838-1843, 25 maio 2009.

FANTUZZI, Cesare. **Chapter 02-Modeling:** Modena: Università Degli Studi di Modena e Reggio Emilia, 2014. 55 slides.

FIGUEIREDO, Eduardo. **Verificação e Validação**. Disponivel em: <http://homepages.dcc.ufmg.br/~figueiredo/disciplinas/aulas/verificacao-validacao_v01-1.pdf> Acesso em 04 jun 2015.

FRANCO, F. ; SANTOS, M. M. D.; STEVAN, S. L.; LUGLI, A. B.; TORRES, W. M. . **Model-Based Functional Safety for Power Window System**. In: XI IEEE/IAS International Conference on Industry Applications, 2014, Juiz de Fora. IEEE/IAS INDUSCON 2014, 2014.

G1, Globo. **Brasil perde para México e agora é 8º em ranking de produção de veículos**. Disponível em: <http://g1.globo.com/carros/noticia/2015/03/brasil-perde-para-mexico-e-cai-para-8-em-ranking-de-producao-de-veiculos.html> Acesso em 26 de abril de 2015.

HERNITER, M. E. **Instrumentation and Microcontrollers using Automatic Code Generation (Freescale KL25Z Version)**. Disponível em: <http://wiki.ece.rose-hulman.edu/herniter/index.php/Instrumentation_and_Microcontrollers_using_Automatic_Code_Generation_(Freescale_KL25Z_Version)> Acesso em 26 de abril de 2015.

HOLTMANN, Jorg; MEYER, Jan; MEYER, Matthias. **A Seamless Model-Based Development Process for Automotive Systems.** 2011. Disponível em: <http://www.cs.uni-paderborn.de/uploads/tx_sibibtex/HMM11.pdf>. Acesso em: 25 mai. 2015.

INSTRUMENTS, National (Org.). **Shortening the Embedded Design Cycle with Model-Based Design.** 2012. Disponível em: <http://www.ni.com/white-paper/4074/en/>. Acesso em: 04 jun. 2015.

KELEMENOVÁ, T. KELEMEN, M. MIKOVÁ, L. PRADA, E. LIPTÁK, T. MENDA, F. MAXIM, V. "**Model Based Design and HIL Simulations.**" 2013. American Journal of Mechanical Engineering, 2013.

LENNON, Tony. **Model-based design for mechatronics systems.** 2007. Disponível em: <http://machinedesign.com/archive/model-based-design-mechatronics-systems>. Acesso em: 23 maio 2015.

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.

MATHEW, S. **Rapid Control Prototyping of Radar-based Active Safety Applications Using dSPACE MicroAutobox**. dSPACE User Conference 2012. September 14, 2012. India.

MATHWORKS. **Why Use Model-Based Design?** Disponível em: <http://www.mathworks.com/model-based-design/>. Acesso em: 04 jun. 2015.

MITSCHANG, Jonas. **Evaluation of a Model-Based Development Process for Automotive Embedded Systems:** Model-Based Development of an Adaptive Cruise Control System. 2009. 115 f. Tese (Graduação) - Curso de Software Engineering,Departamento de Department Of Computer Sciences, University Of Kaiserslautern, Kaiserslautern, 2009.

RAMACHANDRAN, M. CARVALHO, R. A. de. **Handbook of Research on Software Engineering and Productivity Technologies: Implications of Globalization**. 2009.

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

Site oficial de Anacom®.

Site oficial de Delphi® Automotive LLP.

Site oficial de dSPACE® GmbH.

Site oficial de National Instruments® Corporation.

Site oficial de The Mathworks® Inc.

Site oficial de OpenCADD®.

VINCENTELLI, A. S. **New Vistas on Automotive Embedded Systems**. Chess Review. October 4, 2006. Alexandria, VA.

X. Dong, M. Yu, C. Liao, W. Chen. “**Rapid Control Prototyping Development of Intelligent Control System of Vehicle Semi-active Suspension**” World Congress on Intelligent Control and Automation, Junho, 2008, pp 1971-1975.