UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

DIRETORIA DE PESQUISA E PÓS-GRADUAÇÃO

PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

JOÃO HENRIQUE ZANDER NEME

UM ESTUDO DA APLICAÇÃO DO PADRÃO AUTOSAR PARA DESENVOLVIMENTO DE FUNÇÕES EM SISTEMAS EMBARCADOS AUTOMOTIVOS

DISSERTAÇÃO

PONTA GROSSA

2017

JOÃO HENRIQUE ZANDER NEME

UM ESTUDO DA APLICAÇÃO DO PADRÃO AUTOSAR PARA DESENVOLVIMENTO DE FUNÇÕES EM SISTEMAS EMBARCADOS AUTOMOTIVOS

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

PONTA GROSSA

2017

Ficha catalográfica elaborada pelo Departamento de Biblioteca
da Universidade Tecnológica Federal do Paraná, Câmpus Ponta Grossa
n.70/17

N433 Neme, João Henrique Zander

Um estudo da aplicação do padrão AUTOSAR para desenvolvimento de funções em sistemas embarcados automotivos. / João Henrique Zander Neme. 2017.
95 f.; il. 30 cm

Orientador: Prof. Dr. Max Mauro Dias Santos

Dissertação (Mestrado em Engenharia Elétrica) - Programa de Pós-Graduação em Engenharia Elétrica. Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2017.

1. Indústria automobilística. 2. Sistemas embarcados (Computadores). 3. Software de sistemas. I. Santos, Max Mauro Dias. II. Universidade Tecnológica Federal do Paraná. III. Título.

CDD 621.3

Elson Heraldo Ribeiro Junior. CRB-9/1413. 04/12/2017.

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

Título de Dissertação Nº **28/2017**

UM ESTUDO DA APLICAÇÃO DO PADRÃO AUTOSAR PARA DESENVOLVIMENTO DE FUNÇÕES EM SISTEMAS EMBARCADOS AUTOMOTIVOS

por

João Henrique Zander Neme

Esta dissertação foi apresentada às 13 horas do dia **03 de maio de 2017** como requisito parcial para a obtenção do título de MESTRE EM ENGENHARIA ELÉTRICA, com área de concentração em Controle e Processamento de Energia do Programa de Pós-Graduação em Engenharia Elétrica. O candidato foi argüido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

Prof. Dr. Álvaro Costa Neto (USP)

Prof. Dr. Marco Aurélio Wehmeister (UTFPR)

Prof. Dr. Ângelo Marcelo Tusset (UTFPR)

Prof. Dr. Frederic Conrad Janzen (UTFPR)

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

Prof. Dr. Ângelo Marcelo Tusset (UTFPR)
Coordenador do PPGEE

- A FOLHA DE APROVAÇÃO ASSINDADA ENCONTRA-SE ARQUIVADA NA SECRETARIA ACADÊMCIA -

Aos meus pais por sempre acreditarem e investirem em mim. À minha esposa por ser o motivo e inspiração da minha constante busca por ser uma pessoa melhor e mais capacitada.

AGRADECIMENTOS

Primeiramente agradeço aos meus pais pelo apoio absoluto durante todos os anos de minha vida.

À minha esposa pelo amor e apoio incondicional, por estar ao meu lado nas horas que chorei e nas horas que sorri nas horas que me lamentei e nas horas em que de uma forma ou de outra demonstrei total alegria, não me deixando nunca desanimar frente às dificuldades.

Às minhas sobrinhas por serem a alegria da minha vida.

Aos meus colegas do Laboratório de Sistemas Automotivos por compartilharem comigo ideias e conhecimentos.

Ao meu orientador Prof. Dr. Max Mauro Dias Santos, por ser um mestre e amigo que estimulou com ensinamentos e conselhos que contribuíram de forma incomensurável ao longo da pesquisa desenvolvida.

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

A Deus por me conferir saúde e força para superar as dificuldades do caminho.

A mágica de um homem é a engenharia
de outro. (Heinlein, Robert A.)

RESUMO

NEME, João Henrique Zander. **Um estudo da aplicação do padrão AUTOSAR para desenvolvimento de funções em sistemas embarcados automotivos.** 2017. 98 f. Dissertação (Mestrado em Engenharia Elétrica) - Programa de Pós-Graduação em Engenharia Elétrica, Universidade Tecnológica Federal do Paraná, Ponta Grossa, 2017.

A indústria automotiva necessita de constante aumento de qualidade e inovação para se diferenciar das concorrentes, porém sem impactar os custos de produção. No desenvolvimento tradicional geralmente o aumento da complexidade de um sistema é diretamente proporcional ao custo da sua produção. Para inverter esta relação é necessário que se implementem técnicas e padrões modernos de desenvolvimento. Além disso, é estratégico que os engenheiros, hoje em dia, estejam preparados para trabalhar com ferramentas modernas. Este trabalho tem a intenção de apresentar algumas técnicas modernas de desenvolvimento e demonstrar o fluxo de trabalho necessário para a criação de funções automotivas modernas, disponibilizando uma bibliografia que demonstra conceitos e práticas da cadeia de ferramentas específicas para este fim. Inicialmente é apresentada uma visão geral do desenvolvimento de softwares automotivos e uma comparação entre a metodologia tradicional e a abordagem X-in-the-Loop. Na sequência do trabalho o padrão AUTOSAR é apresentado para ser aplicado após o desenvolvimento da função de um painel de instrumentos automotivo.

Palavras-chave: *Software* automotivo. Design baseado em modelos. AUTOSAR, Sistemas embarcados. Padronização de *software*.

ABSTRACT

NEME, João Henrique Zander. **A study of the application of the AUTOSAR standard for function development in automotive embedded systems.** 2017. 98 p. Dissertation (Master Degree in Eletric Engineering) - Programa de Pós-Graduação em Engenharia Elétrica, Federal Technology University - Parana. Ponta Grossa, 2017.

The automotive industry needs constant increase of quality and innovation to differentiate from competitors, but without impacting production costs. In the traditional development increasing complexity of a system directly increases the cost of production. To reverse this relationship it is necessary to implement modern development techniques and standards. This work intends to present some modern development techniques demonstrating the workflow necessary for the creation of modern automotive functions, providing a bibliography that demonstrates concepts and practices of the tool chain for this purpose. Initially, an overview of the development of automotive software and a comparison between the traditional methodology and the X-in-the-Loop approach is presented. Following the AUTOSAR standard is presented to be applied after the design of an automotive dashboard function.

Keywords: Automotive software. Model based design. AUTOSAR. Embedded systems. Software standardization.

LISTA DE FIGURAS

Figura 1 - Crescente demanda por componentes elétricos, eletrônicos e Software para veículos automotores. 12
Figura 2 - Fluxo de desenvolvimento de um produto automotivo com integração de OEM, Fornecedores e Desenvolvedores de Ferramentas. 13
Figura 3 - Diagrama do Modelo-V para sistemas embarcados automotivos 21
Figura 4 - Setup genérico para os testes xIL....26
Figura 5 - Sequência de estágios para xIL. 26
Figura 6 - Diagrama em blocos do estágio MIL. 28
Figura 7 - Diagrama em blocos do estágio SIL. 30
Figura 8 - Diagrama em blocos do estágio PIL. 31
Figura 9 - Diagrama em blocos do estágio HIL. 33
Figura 10 - Método de desenvolvimento tradicional. 34
Figura 11 - Desenvolvimento Baseado em Modelo (MBD). 37
Figura 12 - Estrutura em camadas do padrão AUTOSAR....43
Figura 13 - Módulos de *Software* para MCAL 45
Figura 14 - Esboço de metodologia AUTOSAR. 48
Figura 15 - Arquitetura de Software base em componentes de Software. 50
Figura 16 - Distribuição de componentes de *Software* e *Runnables* nas ECU. 52
Figura 17 - Software arquitetura base em componentes de Software. 52
Figura 18 - A metodologia padrão AUTOSAR....54
Figura 19 - Metodologia AUTOSAR considerando o Modelo-V....55
Figura 20 - Esquemático da fase de desenvolvimento....57
Figura 21 - Arquitetura distribuída física para disponibilização da velocidade rotação do motor no painel de instrumentação. 60
Figura 22 - Arquitetura distribuída com suporte à função de controle de lâmpada nível de combustível. 61
Figura 23- Operações de chaveamento de uma lâmpada indicadora de nível baixo de combustível 62
Figura 24 - Gráfico de estado/transição de uma lâmpada indicadora de nível baixo de combustível 62
Figura 25 - Lógica de funcionamento do grupo Alerts em Simulink/Stateflow....64
Figura 26 - Funcionamento do grupo *Lights* em Simulink/Stateflow....65
Figura 27 - Grupo *TPMS* em Simulink/Stateflow 65
Figura 28 - Lógica de funcionamento dos mostradores de ponteiro em Simulink/Stateflow 66
Figura 29 - Tabela verdade com a lógica de habilitação do sistema....67
Figura 30 - Sinais simulados do sistema de habilitação do painel de instrumentos ..69
Figura 31 - Sinais de entrada e saída do grupo *Lights* para verificação da lógica 69
Figura 32 - Sinais de entrada e saída das funções de Piscas e abertura da porta ...70

Figura 33 - Sinais de entrada e saída das de alerta e fluido do motor e aviso do cinto de segurança.....70
Figura 34 - Sinais de entrada e saída das funções de temperatura do motor e nível de combustível 71
Figura 35 - Sinais de entrada e saída das funções referentes a tensão da bateria...71
Figura 36 - Sinais de entrada e saída das funções de velocidade e RPM 71
Figura 37 - Fluxo de trabalho para a validação virtual em AUTOSAR. 73
Figura 38- Fluxo de trabalho utilizando ferramentas Mathworks® para criação dos arquivos do padrão AUTOSAR. 75
Figura 39 - Visão geral dos arquivos utilizados pelo padrão AUTOSAR. 76
Figura 40 - Cenário para uma aplicação Intra-ECU em AUTOSAR (Arquitetura independente) 77
Figura 41 - Topologia dos SWC 79
Figura 42 - Sinais de saída referente as funções pisca esquerdo e direito além do alerta 82
Figura 43 - Sinais de saída referente as funções luz de posição e luz alta 83
Figura 44 - Sinais de saída referente as funções dos alertas de bateria, cinto de segurança e de abertura de porta 84
Figura 45 - Sinais de saída referente as funções dos alertas de fluido de freio, fluido do motor e mal funcionamento do motor 85
Figura 46 - Sinais de saída referente as funções do alerta da pressão dos pneus e do aviso do acionamento do freio de mão.....85
Figura 47 - Sinais de saída referente as funções de alerta da temperatura do motor e do nível de combustível.....86
Figura 48 - Sinal de saída referente a função do nível de combustível.....87
Figura 49 - Sinal de saída referente a função de temperatura do motor 87
Figura 50 - Sinal de saída referente a função de temperatura do motor 88
Figura 51 - - Sinal de saída referente a função da velocidade 88

SUMÁRIO

1 INTRODUÇÃO 11
1.1 TEMA 14
1.1.1 Delimitação do Tema 14
1.2 PROBLEMA 16
1.3 OBJETIVOS 17
1.3.1 Objetivo Geral 17
1.3.2 Objetivo Específico 17
1.4 MÉTODO DA PESQUISA 18
1.5 ORGANIZAÇÃO DA DISSERTAÇÃO 18
2 DESENVOLVIMENTO DE *SOFTWARE* AUTOMOTIVO 20
3 DESENVOLVIMENTO BASEADO EM MODELOS E O X-IN-THE-LOOP 25
3.1 MÉTODOS DE TESTES PARA XIL 26
3.1.1 Model-In-the Loop (MIL) 27
3.1.2 *Software*-In-the Loop (SIL) 28
3.1.3 Processor-In-the Loop (PIL) 30
3.1.4 Hardware-In-the Loop (HIL) 32
3.2 VANTAGENS DO DESENVOLVIMENTO XIL 33
3.3 TRABALHOS CORRELATOS 38
4 O PADRÃO AUTOSAR 41
4.1 ARQUITETURA DE *SOFTWARE* 42
4.2 METODOLOGIA AUTOSAR 47
4.2.1 Projeto AUTOSAR 49
4.2.2 AUTOSAR e o Desenvolvimento baseado em modelos 53
4.3 VALIDAÇÃO VIRTUAL DE UM SWC 55
4.3.1 Desenvolvimento virtual e testes 56
5 PAINEL DE INSTRUMENTOS AUTOMOTIVO 59
5.1 DESENVOLVIMENTO DAS FUNÇÕES DO PAINEL DE INSTRUMENTOS AUTOMOTIVO BASEADO EM MODELOS 63
5.1.1 Model-in-the-Loop para a função de um Painel de Instrumentos Automotivo. 64
5.1.2 Simulação e teste para verificação da função em MIL 68
6 MIGRAÇÃO DOS MODELOS PARA O PADRÃO AUTOSAR 73
6.1 COMUNICAÇÃO INTRA ECU 77
6.2 GERAÇÃO DOS COMPONENTES DE SOFTWARE 78
6.2.1 Criação dos arquivos para o SWC 78
6.2.2 Design da Arquitetura do Sistema pelo SystemDesk 78
6.2.3 Configuração do experimento 80
7 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS 90
REFERÊNCIAS 91

1 INTRODUÇÃO

O setor automotivo do Brasil se destaca mundialmente por ser o oitavo maior produtor de veículos automotores, ocupando a quarta posição em volume de veículos produzidos entre o grupo dos países emergentes, 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).

Para o domínio de aplicação automotivo, sistemas de controle embarcados é, atualmente, o foco principal de inovações e melhoria na qualidade destes produtos. Isto se dá em função da crescente demanda 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.

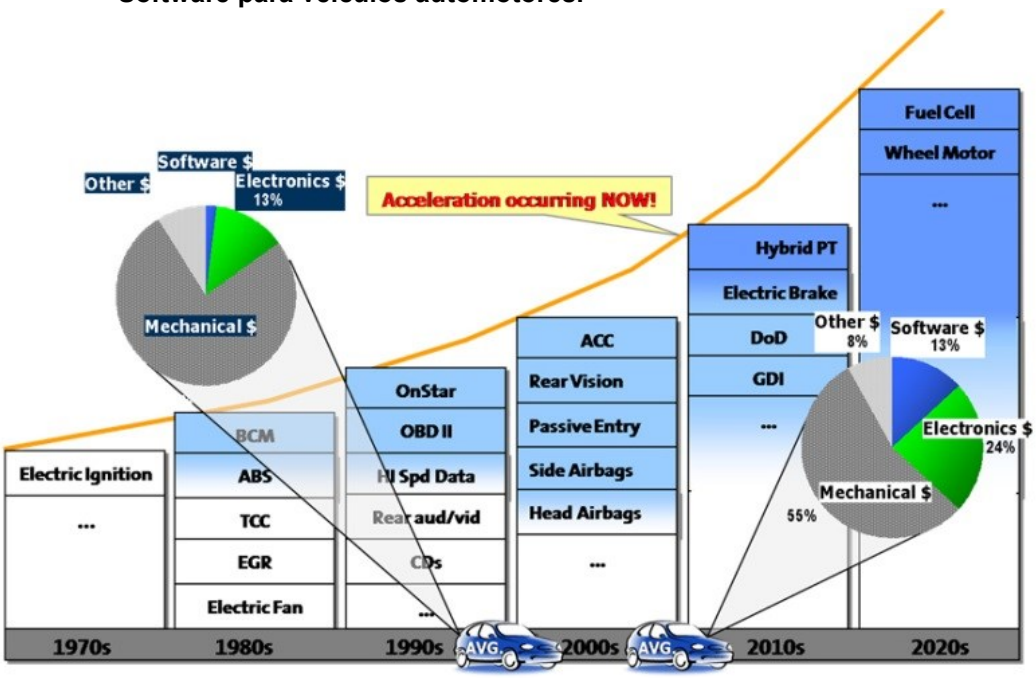
A primeira mudança significativa da migração de componentes mecânicos, hidráulicos e pneumáticos por componentes eletroeletrônicos é observada por um estudo feito por Mercer et al. (2011 apud Broy, 2004). O estudo foca em uma análise de como os fatores de custo no desenvolvimento de um veículo mudou até o ano de 2015 em comparação ao ano de 2002. Em 2015, estimou-se que, dos custos totais envolvidos em um produto automotivo, 35% serão destinados aos componentes de elétrica e eletrônica. Previu-se ainda que, no espaço de análise de 12 anos, enquanto o custo relativo às 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, demandam a um grau de complexidade exponencial para o desenvolvimento de Software embarcado automotivo em arquiteturas E/E.

A Figura 1 apresenta uma estimativa da evolução de conteúdos eletroeletrônicos e Software em automóveis ao longo do tempo e sua participação no custo total do veículo.

Figura 1 - Crescente demanda por componentes elétricos, eletrônicos e Software para veículos automotores.

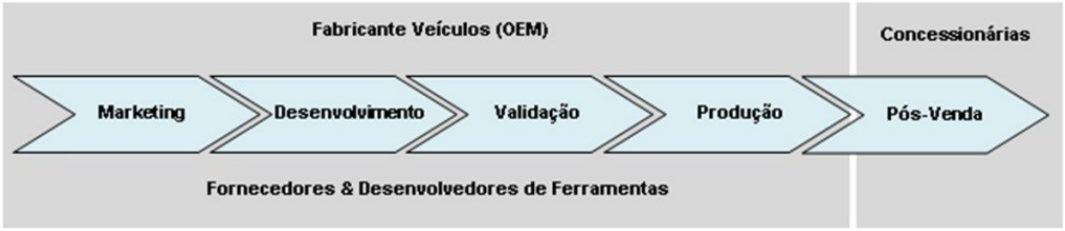


Fonte: Adaptado de Vincentelli (2006).

Este cenário mercadológico é de interesse nacional por ser um vetor que contribui para a economia. Destaca-se ainda que o setor automotivo seja 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. A ligação direta com outros setores da economia impulsiona ao desenvolvimento social e econômico nacional.

O processo de desenvolvimento de um produto automotivo consiste em fases sequenciais obedecendo à seguinte ordem de forma genérica: Marketing, Desenvolvimento, Validação, Produção e Pós-Venda. Os Fabricantes de Automóveis (OEM), Fornecedores e Desenvolvedores de Ferramentas devem estar em constante interação para o projeto e integração dos componentes e conteúdos automotivos. A Figura 2 - Fluxo de desenvolvimento de um produto automotivo com integração de OEM, Fornecedores e Desenvolvedores de apresenta as etapas de desenvolvimento de produto automotivo e seus atores e interfaces.

Figura 2 - Fluxo de desenvolvimento de um produto automotivo com integração de OEM, Fornecedores e Desenvolvedores de Ferramentas.



Fonte: Autor

Em todas estas fases do desenvolvimento, têm-se metodologias e tecnologias de diversos domínios da ciência como química, mecânica, produção, elétrica, eletrônica, computação, logística entre outros, fazendo com que a cadeia automotiva seja uma das mais complexas e dotadas de desafios constantes e de dinâmica acelerada pelas exigências do mercado consumidor (regional e global) e órgãos regulamentadores.

Estas características induzem em ações constantes e inovadores de novas tecnologias e metodologias para desenvolvimento de produtos automotivos com melhor qualidade, segurança e confiabilidade para o mercado consumidor, podendo ser verificado pelos referenciados salões e feiras de automóveis realizados anualmente nas principais cidades do mundo, onde montadoras em parceria com fornecedores mostram as tendências globais em veículos automotores.

Os conteúdos e a complexidade em Eletrônica, Controle e Software para sistemas automotivos estão aumentando mais rápido que os negócios possam ser gerenciados. Os métodos, processos e ferramentas utilizadas atualmente para desenvolvimento de componentes eletroeletrônicos automotivos, são oriundos dos requisitos de conteúdos limitados que os automóveis possuem por volta dos anos de

1990s, onde se tinham menos conteúdos em ECU com arquiteturas distribuídas menores em volume e quantidade.

Para atender a esta demanda futura, a indústria automotiva em conjunto com fornecedores, estão caminhando em parceria estabelecendo padrões abertos para o desenvolvimento de Software em arquitetura E/E.

1.1 TEMA

O presente trabalho consiste em estudar e oferecer bibliografia atualizada de ferramentas específicas e utilizada no desenvolvimento de Software automotivo. O desenvolvimento tradicional apresenta diversos problemas para os envolvidos no processo de criação de uma nova função automotiva. Desta forma, busca-se demonstrar os problemas existentes e apresentar o fluxo prático de trabalho de duas ferramentas que buscam diminuir problemas relacionados à distribuição de recursos em um projeto e simplificar toda a cadeia de desenvolvimento.

A potencialidade da utilização destas ferramentas é destacada durante o trabalho, proporcionando aos interessados a possibilidade de adquirir um entendimento mais prático das ferramentas e conceitos envolvidos. São apresentados também limitações da metodologia e um comparativo com outras técnicas existentes. O intuito é oferecer de forma simples e funcional uma bibliografia sobre este assunto que possui pouca referência bibliográfica atualmente

.

1.1.1 Delimitação do Tema

As ferramentas que serão utilizadas neste trabalho dão balizamento para o desenvolvimento de *Software* na área automotiva. A primeira ferramenta trata-se do desenvolvimento *X-in-the-Loop* (xIL)), derivado do conceito do Desenvolvimento Baseado em Modelos ou *Model Based Design* (MBD) que define um fluxo de trabalho de constante desenvolvimento e verificação. Este fluxo de trabalho consiste no desenvolvimento da lógica de funcionamento de forma virtual, baseando-se em um modelo lógico desenvolvido em Matlab/Simulink.

A utilização do xIL permite que os desenvolvedores se preocupem com o desenvolvimento do controlador do sistema sem necessariamente atrelar o desenvolvimento a um protótipo ou *hardware* específico. Após o desenvolvimento e consequente verificação da funcionalidade do sistema, o xIL permite a geração automática de código para uma plataforma de *hardware* específico. Isso permite uma grande economia de tempo e recursos.

A utilização de diagrama de blocos para a construção da lógica de funcionamento permite uma interação constante entre desenvolvimento e verificação, permitindo que o desenvolvedor altere parâmetros de projeto, resolva problemas encontrados e implemente melhorias na lógica do sistema de maneira fácil e utilizando a mesma interface de *Software* (Matlab/Simulink).

A segunda ferramenta trata-se de uma metodologia de *Software* comumente utilizada no projeto e implementação de sistemas de *Software* automotivos, conhecida como AUTOSAR (AUTomotive Open System ARchitecture). Ele fornece alto nível de abstração entre *Software* e *hardware* através dos conceitos de componentes de *Software* (SWC) e o barramento virtual de comunicação ou *Virtual Function Bus* (VFB). AUTOSAR é um plataforma de padronização de desenvolvimento de *Software* automotivo desenvolvido pelos fabricantes de automóveis europeus e seus fornecedores primários (*Tiers*).

O objetivo do AUTOSAR é permitir que o *Software* automotivo seja composto de componentes desenvolvidos de forma independente um dos outros (Heinecke, 2006) e do *hardware* que será utilizado. No desenvolvimento em AUTOSAR a aplicação é desenvolvida e configurada sem a direta preocupação com a estrutura e configuração da Unidade Central de Controle (ECU). Cada desenvolvedor limita-se a desenvolver, verificar e simular o comportamento dos Componentes de *Software* (SWC) e verificar qual configuração é a mais adequada às suas necessidades de aplicação (Shatat, 2015).

O padrão AUTOSAR é resultado de uma parceria entre as OEM e as Tier, trabalhando juntas para desenvolver e estabelecer uma indústria aberta e padronizada para a arquitetura E/E a qual serve como estrutura básica para gerenciar tanto as novas e futuras funções de veículos e módulos de *Software*.

Discussões iniciais, com desafios e objetivos comuns, foram providenciadas por BMW, Bosch, Continental, DaimlerChrysler e Volkswagen em Agosto de 2002 e se juntaram logo após a primeira reunião com a Siemens VDO.

Uma junta técnica foi formada em Novembro de 2002 para estabelecer a estratégia técnica de implementação. A parceria entre os parceiros iniciais, ou *Core Partners*, foi formalmente assinada em Julho de 2003. A Ford Motor Company se juntou como Core Partner em novembro de 2003. Foi seguida por Peugeot Citroën Automobiles S.A. e a Toyota Motor Corporation também como *Core Partners* em Dezembro de 2003 e a General Motors em Novembro de 2004. Em Fevereiro de 2008 a Siemens VDO se tornou parte da Continental e desde então deixou de ser um *Core Partner* da parceria.

Para o desenvolvimento com as ferramentas citadas serão utilizados Matlab, Simulink e Stateflow da MathWorks para o desenvolvimento da lógica do controlador. Para gerar os arquivos e códigos necessários para o padrão AUTOSAR será utilizada a geração automática de código por Embedded Coder da MathWorks. Finalmente para geração das interfaces AUTOSAR e a arquitetura do sistema de *Software* para simulação serão utilizados as ferramentas SystemDesk e VEOS da dSpace.

1.2 PROBLEMA

Atualmente os veículos automotivos trazem uma gama extensa de sistemas eletrônicos, aumentando sensivelmente a necessidade de sistemas embarcados mais complexos e seguros. O conhecimento e maturidade de métodos, processos e ferramentas adequados, são fatores diferenciais para o sucesso dos sistemas desenvolvidos.

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). Assim como o x-In-the-Loop (xIL), que é uma das metodologias derivadas do MBD, onde x pode ser *software*, modelo ou *hardware*, torna-se substancialmente significativo porque o processo de verificação e validação tem sido cada vez mais difícil devido ao aumento do tamanho do código do controlador.

AUTOSAR é uma parceria internacional de desenvolvimento formada por diversos fabricantes de veículos automotivos, fornecedores primários e fornecedores de ferramentas, que definem conceitos e fluxo de trabalho para o desenvolvimento de *Software* automotivo.

O fluxo de trabalho destas duas ferramentas é bem conhecido e explorado fora do Brasil, no entanto poucas experiências e referências nacionais existem demonstrando a utilização prática da metodologia AUTOSAR no desenvolvimento de componentes de *Software.*

1.3 OBJETIVOS

1.3.1 Objetivo Geral

Demonstrar o fluxo de trabalho de uma estratégia de controle baseada em eventos para um painel de instrumentos automotivos usando a metodologias de desenvolvimento xIL. Disponibilizar bibliografia da padronização AUTOSAR, assunto pouco mencionado em trabalhos nacionais e disponibilizar o fluxo de trabalho necessário para o desenvolvimento de *software* dentro deste padrão. Além de validar as funções do sistema através de simulações em *Software*s utilizados atualmente em ambiente industrial.

1.3.2 Objetivo Específico

- Estabelecer o fluxo de trabalho e as ferramentas computacionais necessárias para desenvolvimento de função para *Software* automotivo.
- Demonstrar o desenvolvimento anterior do sistema do painel de instrumentos automotivo.
- Demonstrar os conceitos que regem o desenvolvimento xIL.
- Apresentar o padrão AUTOSAR.
- Demonstrar o fluxo de trabalho real para o desenvolvimento de *Software* no padrão AUTOSAR

- Confrontar e analisar os sinais de teste e as respostas esperadas.

1.4 MÉTODO DA PESQUISA

Nos capítulos seguintes serão abordados desde a metodologia de desenvolvimento de *Software*, apresentando ferramentas e métodos utilizados atualmente como estado da arte. Serão apresentadas técnicas como a metodologia baseada em simulações xIL e toda a arquitetura AUTOSAR e demonstrado na prática como é possível desenvolver, a partir dos pré-requisitos iniciais, o sistema de um controlador para um painel de controle.

Após a conceituação, o desenvolvimento iniciará com uma função previamente desenvolvida em um trabalho do próprio autor de um sistema menor que alimenta um painel de instrumentos com informações relativas à temperatura do motor e o nível de combustível no tanque. A partir deste modelo será feita a migração para o padrão AUTOSAR e através de sinais de testes será verificado a eficácia da migração.

Após a verificação deste sistema simples, passará a se discutir o sistema do Painel de Instrumentos automotivo e, novamente, a partir de um controlador já desenvolvido em trabalhos anteriores do próprio autor será feita a migração deste sistema complexo e sua verificação.

1.5 ORGANIZAÇÃO DA DISSERTAÇÃO

Este trabalho divide-se da seguinte forma:

- O capítulo 2 apresenta resumidamente uma visão geral do desenvolvimento de softwares automotivos.

- O capítulo 3 fala sobre o conceito do desenvolvimento baseado em modelos e o *X-in-the-Loop* como ferramenta de testes baseados em simulações.

- O capítulo 4 explica conceitos estruturais do padrão AUTOSAR.

- O capítulo 5 explica resumidamente o funcionamento do painel de instrumentos e demonstra rapidamente como foi concebido em ambiente Matlab/Simulink.

- O capítulo 6 mostra a migração do modelo de Simulink para o padrão AUTOSAR e mostra a validação das funções.

- O capítulo 7 apresenta considerações finais e trabalhos futuros.

2 DESENVOLVIMENTO DE *SOFTWARE* AUTOMOTIVO

O desenvolvimento de *Software* embarcado automotivo sob uma Arquitetura E/E é baseado na metodologia de desenvolvimento V-Model, que já possui considerável grau de maturidade. Definido e especificado em conjunto e coordenação pelas matrizes de OEMs, fornecedores e fornecedores de ferramentas, o *know-how* deste conhecimento vem para nosso país com determinado atraso, visto que os veículos com elevado nível de conteúdos eletroeletrônico ainda são desenvolvidos fora do território nacional.

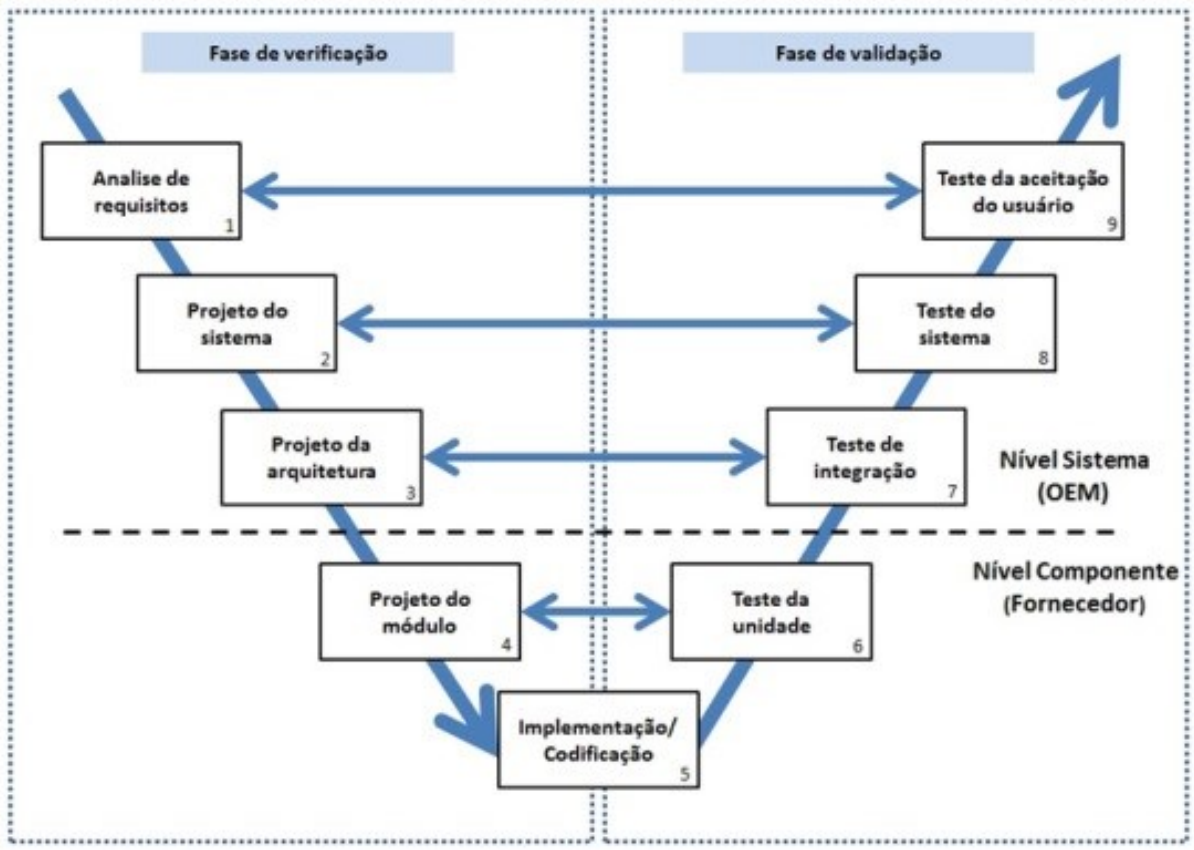
Isto implica em dizer que podemos motivar nossos engenheiros em conhecer novos métodos, processos e ferramentas que possam auxiliá-los a ter condições suficientes para desenvolvimento de sistemas automotivos complexos.

A composição da metodologia V-Model ou Modelo-V consiste em uma estrutura de testes que pode ser usada durante todas as etapas de desenvolvimento, enfatizando pela qualidade e controle do *Software* de forma a detectar erros antecipadamente.

O desenvolvimento de *Software* embarcado automotivo sob uma Arquitetura Eletroeletrônica (E/E) é baseado na metodologia de desenvolvimento Modelo V (Ramachandran, 2009). A especificação é definida em conjunto e coordenadamente entre as matrizes dos fabricantes originais de equipamentos (*Original Equipment Manufacturer* ou OEM), fornecedores e fornecedores de ferramentas.

A Figura 3 apresenta as atividades, fluxo e responsabilidade do ciclo de desenvolvimento baseado no Modelo V.

Figura 3 - Diagrama do Modelo-V para sistemas embarcados automotivos



Fonte: Stella (2015).

A composição do Modelo-V pode ser explicada observando seu ciclo de desenvolvimento na Figura 3. 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:

1. **Análise de requisitos**: 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. Entretanto não é possível determinar a forma como o *Software* será projetado ou construído.

2. Projeto do Sistema: É a fase em que os desenvolvedores de sistemas analisam e compreendem o negócio do sistema proposto por estudar o documento de requisitos do usuário. Podem descobrir possibilidades e técnicas pelas quais as necessidades do usuário podem ser implementadas. Se algum dos requisitos não são exequíveis, o usuário é informado sobre a questão e a solução é encontrada e o documento de requisitos do usuário pode ser editado em conformidade.

3. Projeto de Arquitetura: A fase de concepção e arquitetura do *hardware* e de *Software* pode também ser referida como uma concepção de alto nível. A linha de base na seleção da arquitetura é que ele deve considerar tudo o que normalmente consiste de uma lista de módulos, breve funcionalidade de cada módulo, suas relações de interface, dependências, tabelas de banco de dados, arquitetura diagramas, detalhes da tecnologia etc.

4. Projeto do Módulo: A fase de projeto do módulo também pode ser referida como o projeto de baixo nível. O sistema concebido está dividido em pequenas unidades ou módulos e cada um deles é explicado para que o programador possa iniciar a codificação diretamente. O documento ou programa de especificações de projeto de baixo nível irá conter uma lógica funcional detalhada do módulo, em pseudocódigo.

Já a fase de validação é composta pelo desenvolvimento das seguintes ações:

5. Implementação ou Codificação: A implementação das funções em alguma linguagem de programação podendo-se codificar manualmente as funções ou recorrer à geração automática de código através de ferramentas específicas.

6. Teste de Unidade: O plano de testes de unidade são desenvolvidos durante a fase de projeto do módulo sendo executadas para eliminar erros a nível do código ou nível de unidade. Uma unidade é a menor entidade que pode existir, independentemente, por exemplo, um módulo de programa ou uma unidade de controle eletrônica (ECU – *Electronic Control Unit*). O teste de unidade verifica se a menor entidade 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 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 são desenvolvidos durante a fase de projeto do sistema e ao contrário do plano de testes da unidade e de integração, o plano de testes do sistema é composto por equipe de negócios do cliente. O teste do sistema assegura que as expectativas de aplicação desenvolvida 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 desenvolvidos durante a fase de análise de requisitos sendo compostos por usuários de negócios. É realizada 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 sistema está pronto para uso em tempo real.

Através do desenvolvimento de uma arquitetura física, que pode ser considerada a 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.

3 DESENVOLVIMENTO BASEADO EM MODELOS E O X-IN-THE-LOOP

A crescente demanda por novas funcionalidades e melhorias das já existentes em veículos, tem elevado o nível de complexidade para os sistemas embarcados automotivos tendo em vista uma constante preocupação por garantir maior desempenho, segurança e confiabilidade. Sendo assim, as metodologias de desenvolvimento de *Software* tradicionais não garantem o atendimento aos requisitos e restrições para as inovações em veículos da próxima geração, sendo mandatório que se criem novas metodologias para o processo de desenvolvimento desses sistemas embarcados com qualidade. Não sendo isto suficiente, deve-se ainda considerar a inserção de novas tecnologias de sistemas elétricos e eletrônicos.

Segundo Wehrmeister (2011), os sistemas embarcados em tempo real devem fornecer um conjunto crescente de serviços, enquanto o seu tempo de desenvolvimento e custo deve permanecer o mais baixo possível.

A metodologia de desenvolvimento de *Software* baseado em modelo encaixa exatamente neste conceito, com uma abordagem que pode atender as demandas para inovações em sistemas embarcados automotivos através de 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 físico em que se deseja controlar e seu controlador utilizando a mesma ferramenta computacional. Isto facilita a visualização e o entendimento do sistema de controle como um todo e assim desenvolver um projeto com melhor qualidade. O resultado de tudo isto é caracterizado por um sistema funcional e de fácil verificação, diminuindo significativamente a possibilidade de que os componentes individuais não se encaixem de maneira otimizada.

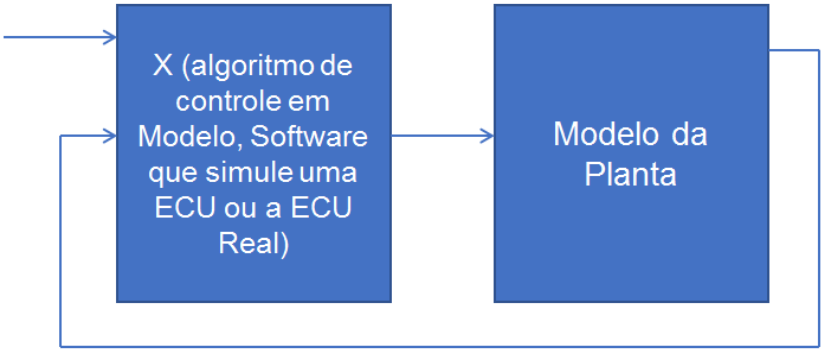
Dentro do desenvolvimento de *Software* baseado em modelos existe a metodologia de testes baseados em simulação (*simulation based testing*) (Iwagaya, 2013) de onde deriva o conceito do desenvolvimento xIL. O método utiliza técnicas de validação da função de *Software* sequenciais em que para cada estágio têm-se bem definidos as etapas do projeto a seguir. Adicionalmente, o processo de testar e otimizar funções e estratégias de controle em uma ECU protótipo, é denominado por

prototipagem rápida do controlador pode ser considerado como um mecanismo adicional para o desenvolvimento de *Software* embarcado automotivo, porém de forma rápida acelerando o processo.

3.1 MÉTODOS DE TESTES PARA XIL

No contexto do Projeto Baseado em Modelos no domínio automotivo o processo de testes de uma ECU pode ser dividido em níveis de teste (Tibba, 2017). Uma imagem geral da abstração trazida por esses testes pode ser vista na Figura 4.

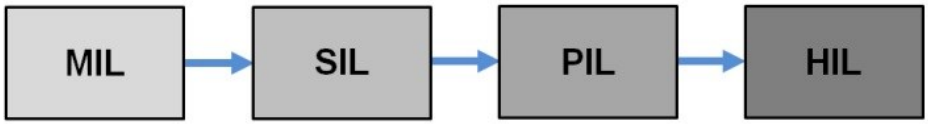
Figura 4 - Setup genérico para os testes xIL



Fonte: Tibba (2017) (Adaptado pelo Autor)

Geralmente a metodologia xIL é caracterizada por utilizar as tradicionais técnicas de validação e simulação. Os estágios 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 demonstrado na Figura 5.

Figura 5 - Sequência de estágios para xIL.



Fonte: Neme (2014)

A sequência de estágios para o desenvolvimento de *Software* baseado em modelos tem como finalidades:

- MIL – Especificação e projeto do controlador funcional;

- SIL – Teste do gerador e da função em código C ou C++;
- PIL – Geração do código executável para testes no microcontrolador usado na ECU;
- HIL – Teste do controlador já embarcado na ECU em que já possui os componentes de entrada/saída, em que a ECU pensa já estar controlador a planta física.

3.1.1 Model-In-the Loop (MIL)

É o primeiro estágio de um projeto em xIL e caracteriza-se por completo como a fase de simulação da planta física e do controlador os quais não operam tempo real. Normalmente, este estágio do processo permite que os engenheiros possam estudar e analisar o desempenho do sistema e conceber o algoritmo de controle, em um ambiente virtual, através de simulações por computador de execução do sistema completo ou do subsistema.

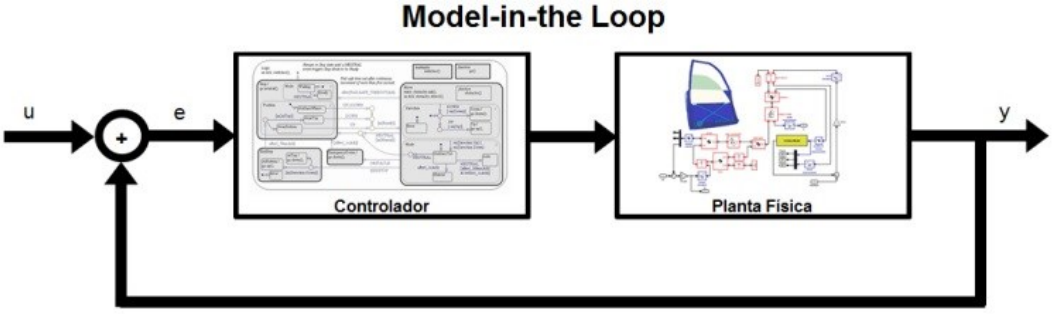
Uma ferramenta computacional é utilizada em um mesmo ambiente de trabalho, o qual se tem a planta física simulada virtualmente e a função de controle a qual pode ser realizado utilizando mecanismos de programação gráfica como diagrama em blocos e máquinas de estados finitos, capaz de representar um controlador baseado no tempo e em eventos. A estratégia de controle e a planta física são desenvolvidos de forma extremamente rápida, possibilitando realizar mudanças e testes de verificação dos requisitos de forma a possibilitar encontrar possíveis erros em fases preliminares ao desenvolvimento.

A simulação em MIL é feita em aritmética de ponto-flutuante e serve como uma referência para os estágios subsequentes fornecendo os valores mínimos e máximos de variáveis como uma base para aplicação posterior de aritmética em ponto-fixo se requerido. Nesse caso, não é necessário ter-se em mãos protótipos físicos disponíveis para desenvolvimento do controlador, como no caso de uma válvula borboleta ou vidro elétrico. Estes componentes podem ser representados virtualmente em um ambiente de simulação.

A Figura 6 ilustra em diagrama de blocos na forma de um sistema de controle realimentado a representação do estágio MIL, onde se tem o modelo da planta física e do controlador realizados em um ambiente de simulação virtual como

MATLAB/Simulink/StateFlow e seus conjuntos de Toolbox disponíveis. Com isto é possível realizar testes de verificação iniciais.

Figura 6 - Diagrama em blocos do estágio MIL.



Fonte: Neme (2014)

Na Figura 6:

- Controlador – Representado por diagrama de blocos ou máquina de estados finitos possível de ser simulado;
- Planta Física – Modelo da planta física a ser simulado.

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

No estágio de SIL, o código de *Software* para produção real é gerado e testado num computador para execução permitindo a inclusão de funcionalidades de *Software* para o qual não existe nenhum modelo (quando se deseja incluir outra rotina) ou para permitir simulações mais rápidas. Apenas o código em linguagem C ou C++ é gerado e não o executável, em que para isto precisaríamos de um processador adequado. Com a simulação SIL sob o PC, é possível comparar os dados referentes ao comportamento do controlador a partir do código gerado com os dados obtidos a partir do modelo em simulação MIL.

Consiste num estágio onde a ferramenta de geração automática de código fornece a estratégia de controle estabelecida no estágio MIL automaticamente em código em linguagem C ou C++ onde o modelo do controlador é um pouco mais real. É considerado como um estágio essencial para testar o sistema de geração de código (seja feito de maneira automática ou manual). A interação do modelo diminui um pouco com relação ao estágio MIL, mas já é possível enxergar erros de lógica. O diagrama de blocos desta fase pode ser observado na Figura 7. Conforme a figura

ilustra, o controlador é representado pelo código em linguagem C com extensão .c e a biblioteca que possui extensão .h, e a planta segue sendo representada por um diagrama de blocos como no Simulink®.

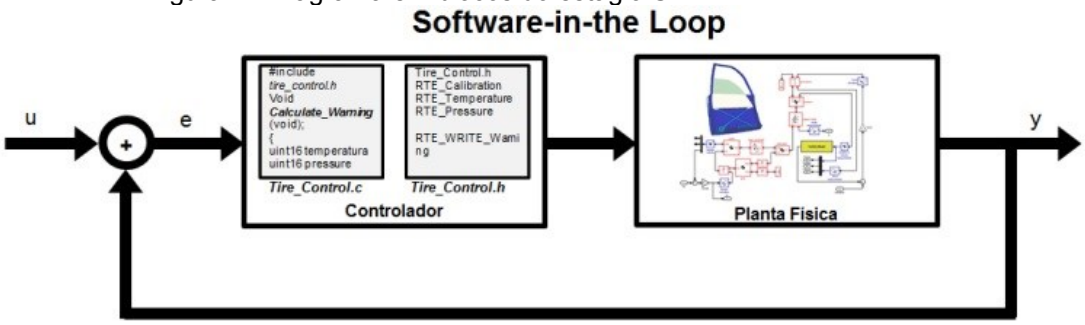
O SIL consiste assim na inclusão de código compilado em *Software* de produção em um modelo de simulação. Como principais finalidades têm-se:

- Permitir a inclusão da funcionalidade de algoritmos de controle para o qual não existe modelo;
- Acelerar a velocidade de simulação, incluindo o código compilado no lugar de modelos interpretativos;
- Verificar que o código gerado a partir de um modelo irá funcionar de forma idêntica ao modelo garantir que um algoritmo no ambiente de modelagem vai funcionar de forma idêntica à que mesmo algoritmo de execução, de um controlador de produção;
- A detecção precoce de bugs e falhas de projeto, especialmente no Software de controle;
- Dimensionamento, aplicação e otimização de parâmetros de Software de controle. Como resultado, os projetos mais maduros são transferidos para projeto mais tarde e fases de teste.

Além disto, caracteriza-se ainda em função dos efeitos do ponto-fixo a possibilidade em avaliar os erros de quantização, saturação e overflow e alternativas de implementação.

- Controlador – Representado por uma linguagem de programação onde o programa principal consiste em um arquivo com extensão .c e o arquivo de cabeçalho com extensão .h. Este código pode ser executado em um computador do tipo IBM-PC;

Figura 7 - Diagrama em blocos do estágio SIL.



Fonte: Neme (2014)

Na Figura 7:

- Planta Física – Modelo da planta física continua a ser simulado.
- Controlador – Representado por uma linguagem de programação onde o programa principal consiste em um arquivo com extensão .c e o arquivo de cabeçalho com extensão .h. Este código pode ser executado em um computador do tipo IBM-PC;
- Planta Física – Modelo da planta física continua a ser simulado.

Neste estágio é gerado o código C ou C++, que é executado sob um computador do tipo IBM-PC. Não existe ainda a compilação do código gerado para ser embarcado em uma EVB (*Evaluation Board*) ou ECU (*Electronic Control Unit*).

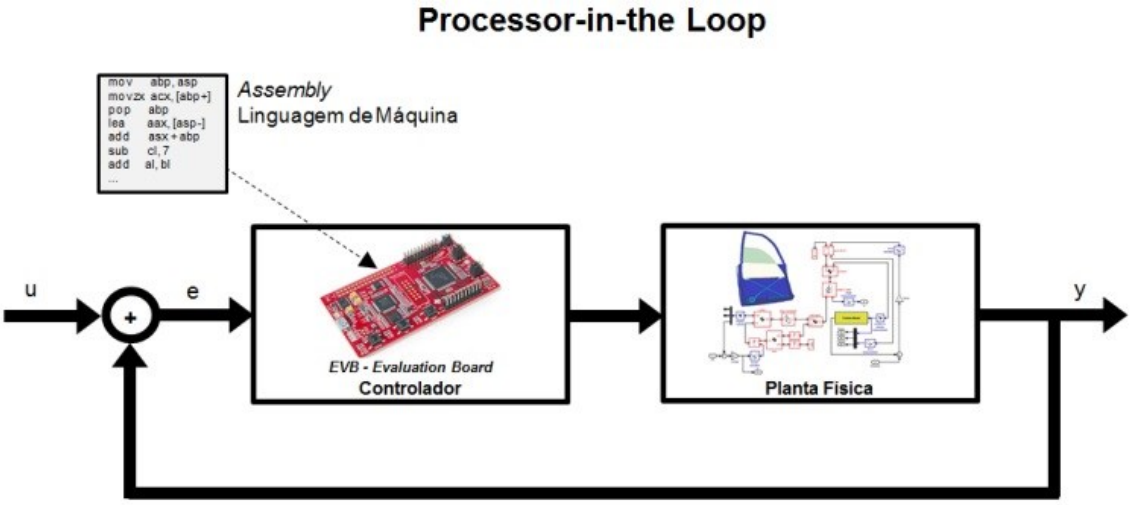
3.1.3 Processor-In-the Loop (PIL)

Nesta fase não se roda mais o código do sistema em simulação, ao invés disso ele é implementado em um microcontrolador podendo ser numa EVB. O controlador é compilado gerando o executável e baixados em uma placa de avaliação equipada com o mesmo microcontrolador que será utilizado na ECU embarcada e se comunica diretamente com o modelo da planta através de interfaces de comunicações padrão, tais como Ethernet. Neste caso, a placa com microcontrolador não possui ainda os dispositivos de entrada e saída, então não podemos dizer que estamos testando uma ECU. O teste da ECU é de fato realizado apenas no próximo estágio que é o HIL, sendo assim a ECU final teria o microcontrolador e os dispositivos de entrada e saída que as compõe.

A verificação bem sucedida do PIL com o MIL e SIL assegura a qualidade do Software gerado. Enquanto o código do controle está rodando no microcontrolador, a planta física ainda está sendo representada por um diagrama de blocos como em Simulink®.

Figura 8 ilustra uma representação em diagrama de blocos de um sistema PIL. Neste estágio os testes são projetados para expor problemas com a execução em um sistema embarcado e tem como principais tarefas a validação do código de forma a verificar as propriedades do tempo de execução (runtime) do código através da análise do código exato, medição do tamanho e consumo da pilha (stack), dimensão da RAM e ROM, e verificações finais.

Figura 8 - Diagrama em blocos do estágio PIL.



Fonte: Neme (2014)

Na Figura 8

- Controlador – Uma placa de avaliação contendo o microcontrolador (EVB - Evaluation Board), é usada para hospedar e executar o controlador que agora é representado em linguagem de máquina, depois de ser compilado a partir da linguagem C (arquivos .c e .h);
- Planta Física – Modelo da planta física continua a ser simulado.

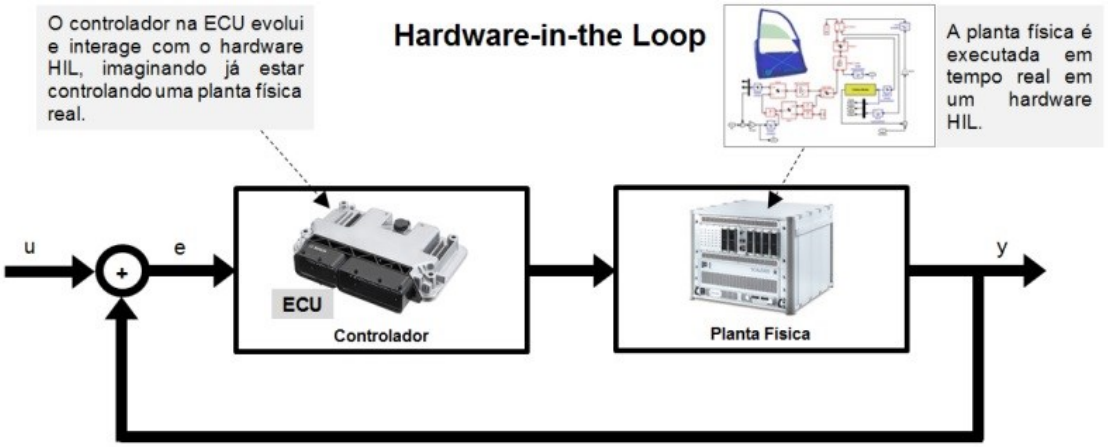
Enquanto o código do controle está rodando no microcontrolador, a planta física ainda é simulada.

3.1.4 Hardware-In-the Loop (HIL)

O estágio HIL consiste na combinação de um modelo de simulação matemática de um sistema com *hardware* físico real, de tal modo que o *hardware* funciona como se fosse integrado no sistema real. Para os testes e desenvolvimento de controladores eletrônicos embarcados, o controlador de *hardware* e *Software* associado estão ligados a uma planta de simulação matemática do sistema, que é executado em um computador em tempo real. Para ligar o modelo em tempo real ao *hardware* do controlador, o computador em tempo real recebe sinais elétricos a partir do controlador quando o atuador de comandos para dirigir a planta, e converte estes sinais para as variáveis físicas ligadas ao modelo da planta. O modelo da planta calcula as variáveis físicas que representam as saídas da planta, que são convertidos em sinais eléctricos que representam as tensões produzidas pelos sensores que alimentam o controlador.

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 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 clara a maior parte dos problemas que poderão ser encontrados. A fase de HIL é ilustrada pela Figura 9.

Figura 9 - Diagrama em blocos do estágio HIL.



Fonte: Neme (2014).

Na Figura 9:

- Controlador – Enfim, o controlador é embarcado em uma ECU que possui os módulos de entrada e saída com microcontrolador embarcado. A ECU interage com o HIL de forma a simular em tempo real a planta física;

- Planta Física – Modelo da planta física agora executa em uma plataforma de *hardware* HIL em tempo real. Esta configuração pode emular todas as cargas de entrada e saída, fazendo com que a ECU imagine estar controlando seu sistema final.

3.2 VANTAGENS DO DESENVOLVIMENTO XIL

No processo desenvolvimento de *Software* usando as metodologias de tradicionais, 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 possíveis de não serem descobertos possíveis falhas, pois muitas vezes não se cobrem todos os casos de testes.

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* e *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 ou estágio, defeitos podem surgir, porém estas falhas só serão observadas na fase de implementação podendo onerar o projeto ou até mesmo inviabilizá-lo.

A Figura 10 apresenta os quatro estágios de desenvolvimento, em que tem-se a definição dos requisitos e especificação em que são utilizados documentos textuais para esta atividade. Nesta etapa de projeto, necessita-se de protótipos físicos que em alguns casos não estaria ainda disponível para desenvolvimento do *Software* de controle, tornando-se assim incompleto e caro. No estágio de implementação manual, a quantidade de ferramentas sem integração pode levar a erros de projeto e humanos e no estágio de teste e verificação os erros encontrados podem ser custosos ou terem sido detectados nas fases preliminares.

Figura 10 - Método de desenvolvimento tradicional.

Fonte: Neme (2014)

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, controle eletrônico é uma parte vital de cada produto desenvolvido.

O desenvolvimento baseado em modelo é uma abordagem que pode reduzir custos, tempo de um projeto e garantir a melhoria na qualidade do produto, 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 dentro do conceito do xIL é que inicialmente possa utilizar uma única plataforma durante a maior parte do desenvolvimento, permitindo que se crie a planta física do sistema desejado a controlar e seu controlador utilizando uma mesma ferramenta computacional. Isto facilita a visualização e uma melhor compreensão do comportamento do sistema como um todo assim como possíveis erros e consequentemente tendo como resultado um sistema funcional e de fácil verificação. Wehrmeister (2012) afirma que a verificação em etapas iniciais de um projeto de sistemas embarcados em tempo real desempenha um papel importante na diminuição dos custos de concepção, pois o custo para correção dos erros detectados em estágios avançados é muito alto.

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.

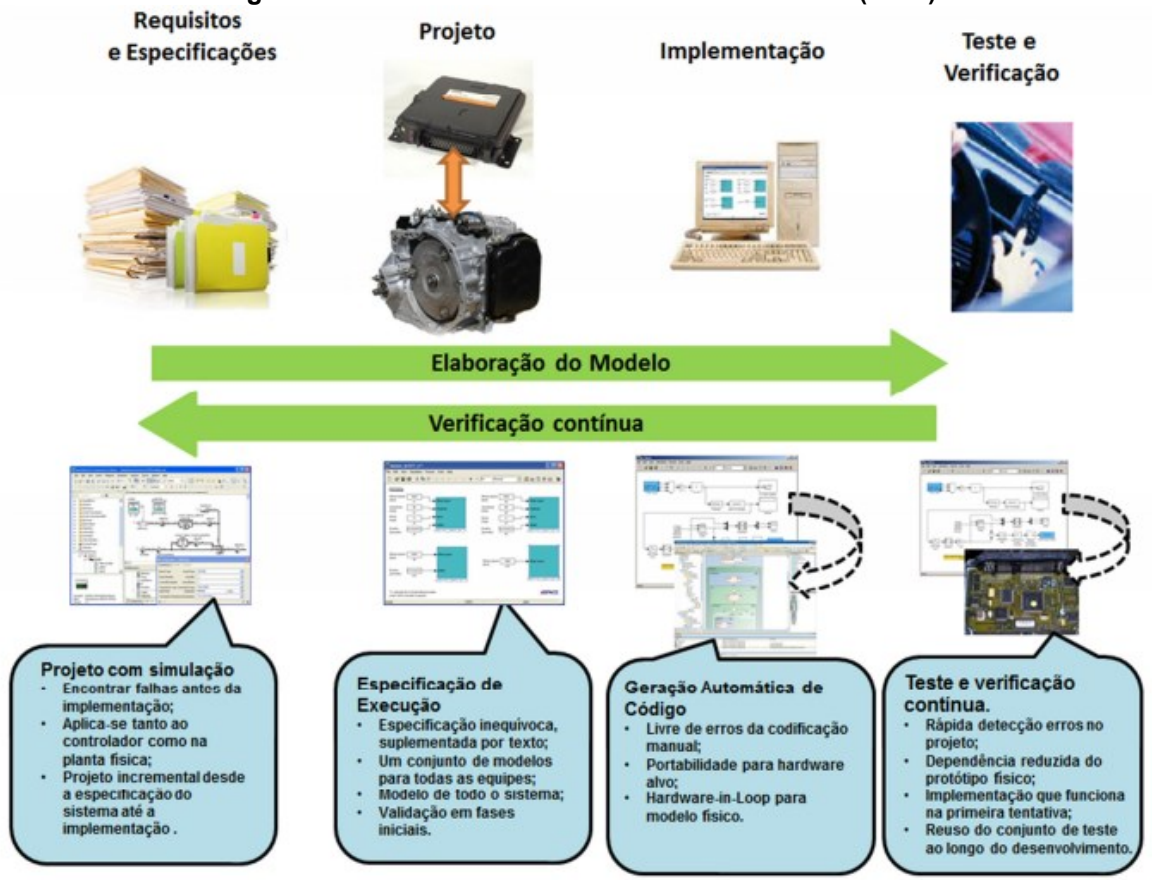
A **Erro! Fonte de referência não encontrada.** ilustra o fluxo de trabalho facilitado do MBD. Todas as etapas se conectam com facilidade, permitindo que se acesse e modifique características do das fases iniciais de mesmo nas últimas etapas de implementação.

Na metodologia MBD o processo de desenvolvimento segue basicamente as mesmas etapas demonstradas na metodologia de desenvolvimento baseado no Modelo-V. 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:

- Análise de requisitos;
- Projeto do sistema;
- Projeto de arquitetura;
- Projeto do módulo;
- Implementação ou codificação;
- Teste de unidade;
- Teste de integração;
- Teste de sistema;
- Teste de aceitação ao usuário;

Figura 11 - Desenvolvimento Baseado em Modelo (MBD).



Fonte: Neme (2014)

Como em MBD a planta física é o centro do processo de desenvolvimento do controlador, deve-se primeiramente ter o modelo ou identificação da planta física inicialmente para as etapas subsequentes. Desta forma, definem-se todos os requisitos necessários para o funcionamento do controle do sistema conforme esperado. Estes requisitos são descritos textualmente através de uma ferramenta integrável e permite 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 seja 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.

3.3 TRABALHOS CORRELATOS

De acordo com a MathWorks®, no desenvolvimento baseado em modelo, modelo de uma planta física é o centro do processo de desenvolvimento MBD. O processo consiste sequencialmente a partir da definição dos requisitos, modelagem ou identificação da planta física, desenvolvimento do projeto do controlador, testes de verificação, implementação e testes de validação. Esta metodologia está transformando o modo que engenheiros e cientistas trabalham, tirando as tarefas do projeto do laboratório e do campo e levando as atividades para serem realizadas em ferramentas computacionais.

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

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

- 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 (montadora e fornecedores);

- 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 considerados pequenos e o impacto sob o projeto é mínimo;

- Expansão das oportunidades de reutilização de projetos para atualizações de sistema ou para sistemas derivativos.

Tibba (2017) diz que os testes e validação representam mais de metade dos custos de desenvolvimento do sistema em camadas. Ao mesmo tempo, a eficiência do processo e a qualidade do produto dependem da maturidade de teste, validação e calibração. Quanto mais cedo os erros forem identificados no processo de desenvolvimento, maiores serão as economias em relação à correção e ao custo de ajustes de falhas.

Störmer (2015) afirma que a criação de instâncias no desenvolvimento de software automotivo é perceptível como a tentativa de substituir artefatos reais, como por exemplo, motores elétricos ou de combustão, ou atividades realizadas sobre esses artefatos reais, por exemplo, testes e validação de veículos reais,

através de artefatos virtuais, ou seja, representações virtuais chamados de modelos, e atividades realizadas nesses artefatos virtuais, por exemplo, verificação e validação baseada em modelos. Em seu trabalho ele conclui que fechar as lacunas entre as fases do desenvolvimento em um ambiente XiL de forma apropriada, permite que os desenvolvedores conduzam análises de forma homogênea usando o ferramental apropriado durante todo o processo de desenvolvimento.

Broy (2012) apresenta um estudo feito pela empresa Altran Technologies que aponta, entre outros dados, relatos de experiências positivas e negativas com o uso do desenvolvimento baseado em modelos, conforme demonstrado no Quadro 1.

Quadro 1 - Experiências positivas e negativas do uso de desenvolvimento baseado em modelos

Experiências Positivas	Experiências Negativas
• Comunicação simplificada por causa do uso de um modelo funcional no desenvolvimento de SW. • Mesmo colegas de outros departamentos ou domínios, que não estão familiarizados com o desenvolvimento de software, podem estar envolvidos no desenvolvimento de software, por causa do uso de modelos. • Facilidade em incluir know-how extra no desenvolvimento de software. • Possibilidade de simulação em estados iniciais do modelo funcional. Revisões de modelos, verificadores de diretrizes, prototipagem de controle rápido (RCP) e Testes em MIL ajudam a encontrar erros já na fase de projeto.	• Alto custo para redefinir o processo de desenvolvimento necessário para implementar o desenvolvimento baseado em modelos. • Além do custo com ferramentas, relatou-se gastos com treinamento de pessoal e novo desenvolvimento de funções antigas dentro do padrão novo. • Alta dependência dos fornecedores de ferramentas, pois geralmente os arquivos gerados com uma ferramenta não necessariamente funcionam em outra plataforma. • Se uma empresa decide por uma ferramenta de modelagem, é aconselhável comprar o resto da cadeia de ferramentas do mesmo fornecedor de ferramentas, para maximizar a homogeneidade no processo de desenvolvimento.
• Possibilidade de automatizar é vista como um efeito positivo adicional e é um fator chave para um desenvolvimento mais eficiente.	• Mesmo se existirem interfaces padronizadas, os especialistas ainda são pessimistas em relação à dependência de fornecedores de ferramentas.

Fonte: Broy, 2012 (adaptado pelo autor)

O estudo de conclui que o design baseado em modelo tem um enorme benefício uma vez que pode trazer economias de custo e tempo no desenvolvimento e na manutenção, a qualidade de artefatos de desenvolvimento único e a qualidade do produto e, além disso, os modelos de função desenvolvidos podem ser facilmente reutilizados em linhas diferentes de carros.

4 O PADRÃO AUTOSAR

O AUTOSAR (AUtomotive Open System Architecture) é uma arquitetura de *Software* automotivo aberto e padronizado, que é baseado no conceito de componentes de *Software* o qual é desenvolvido e mantido por fabricantes de automóveis, fornecedores e desenvolvedores de ferramentas que podem fornecer o melhor caminho para suportar as aplicações da próxima geração de sistemas embarcados automotivos (AUTOSAR Basics, 2016).

O principal objetivo desta norma é criar e estabelecer um padrão aberto de *Software* embarcado para ECU automotiva, com base na metodologia específica para sistemas elétrico e eletrônico automotivos, que irá fornecer uma infraestrutura básica para apoiar o desenvolvimento de *Software* veicular, interfaces de usuário e de gestão para todos os domínios de aplicações para a próxima geração de sistemas automotivos.

A padronização proporciona benefícios consideráveis que inclui aspectos relevantes do *Software* automotivo ao considerar que as funções básicas dos sistemas, escalabilidade para suportar capacidade diferente modelos de veículos e variantes da plataforma, para realizar a transferência em toda a rede, integração de múltiplos fornecedores de ECU, capacidade de manutenção ao longo das atualizações de todo o ciclo de vida do produto e *Software* e atualizações sobre a vida útil do veículo (serviços) como alguns dos principais objetivos.

Motivação do padrão AUTOSAR (Nishikawa, 2006):

- Gestão da complexidade E&E (Elétrica e Eletrônica) automotiva por causa do crescimento de novas funções;
- Flexibilidade para modificação de um produto, melhorias e atualizações;
- Escalabilidade de soluções dentro e através de linhas de produtos;
- Melhoria da qualidade e confiabilidade dos sistemas E&E automotivos.

Objetivos do padrão AUTOSAR:

- Cumprimento dos requisitos de veículos futuros, como, disponibilidade e segurança, atualizações e manutenção de *hardware* e *Software* embarcados;

- Aumento da escalabilidade e flexibilidade para integração e transferência de funções;
- Inserção de componentes de *hardware* e *Software* do tipo COTS (Produtos de Prateleiras - *Commercial Off the Shelf*) em todas as linhas de produtos;
- Contenção do produto, complexidade do processo e o risco melhorado;
- Otimização de custos de sistemas escaláveis.

4.1 ARQUITETURA DE *SOFTWARE*

No padrão AUTOSAR, a arquitetura de Software é totalmente modular e possibilita reutilização de código e modelos. Na Figura 12 - Estrutura em camadas do padrão AUTOSAR é possível ver, de cima para baixo, a Camada de Aplicação (APL - Application Layer), Ambiente de Execução (RTE - Runtime Environment) e Software Básico (BSW - Basic Software).

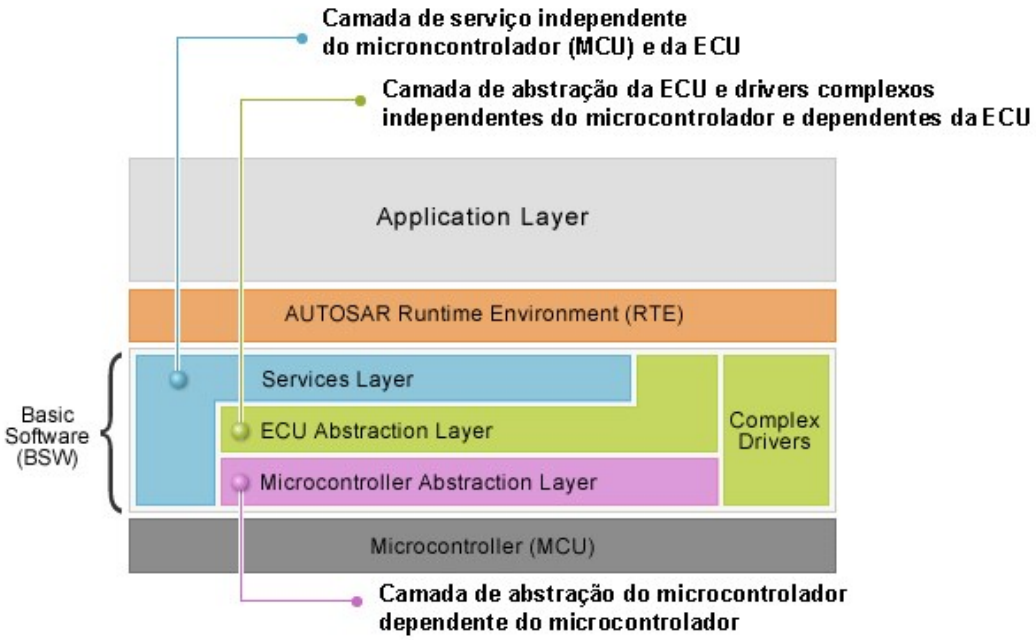
Em tempo de projeto tem-se o Barramento Funcional Virtual (VFB - Virtual Functional Bus) que é convertido em RTE em tempo de execução. Note que o BSW possui quatro camadas que são respectivamente a Camada de Serviços (Services Layer), Camada de Abstração da ECU (ECUAL – ECU Abstraction Layer), Drivers Complexos (Complex Drivers) e Camada de Abstração do Microcontrolador (MCAL – Microcontroller Abstraction Layer).

A arquitetura de *Software* é a estrutura de alto nível composta por componentes de Software, que compreende nos elementos e suas relações. O componente de Software é uma unidade de composição com interfaces contratualmente especificadas e dependências de contexto explícitas únicas que pode ser implantado de forma independente e está sujeita a composição por terceiros.

Assim, a fim de fazer um sistema embarcado baseado em componente de Software possível, o padrão AUTOSAR usa uma arquitetura em camadas que garante a dissociação entre a funcionalidade de componentes com interfaces bem feitas a partir dos serviços de *Hardware* e *Software* de apoio. A disposição de uma arquitetura de *Software* básica pode ser denotada de forma genérica pela camada de aplicação, ambiente de execução e camada de Software básico.

- Camada de aplicação: A camada de aplicação é uma camada que não é composta de Software normalizado, que também é a camada em que a funcionalidade de acordo com os requisitos do usuário são implementadas de forma real. Ela é composta por componentes de Software de aplicativos que interagem com o ambiente abaixo em tempo de execução.
- Ambiente de Execução (Runtime Environment): Ele lida com a troca de informações entre os componentes de Software de aplicação e conecta os componentes de Software aplicativo para o hardware certo. Esta camada desacopla os componentes de Software aplicativo do hardware, bem como os componentes de Software aplicativo de si mesmos. Esta abordagem proporciona maior interoperabilidade independentemente do hardware.
- Camada de Software Básico: É uma camada caracterizada por um Software padronizado que não tem qualquer funcionalidade, mas oferece serviços dependentes do hardware e independente de hardware para a camada acima (Runtime Environment), que é realizada através do uso de interfaces de programação de aplicativos. Isso por si só, faz com que esta camada não seja totalmente independente de hardware, mas faz com que o as camadas superiores sejam independentes do hardware.

Figura 12 - Estrutura em camadas do padrão AUTOSAR



Fonte: Renesas (acessado em 2016 e adaptado pelo autor).

O Quadro 2 apresenta a descrição característica e finalidade de cada uma das camadas apresentadas na Figura 12 - Estrutura em camadas do padrão AUTOSAR.

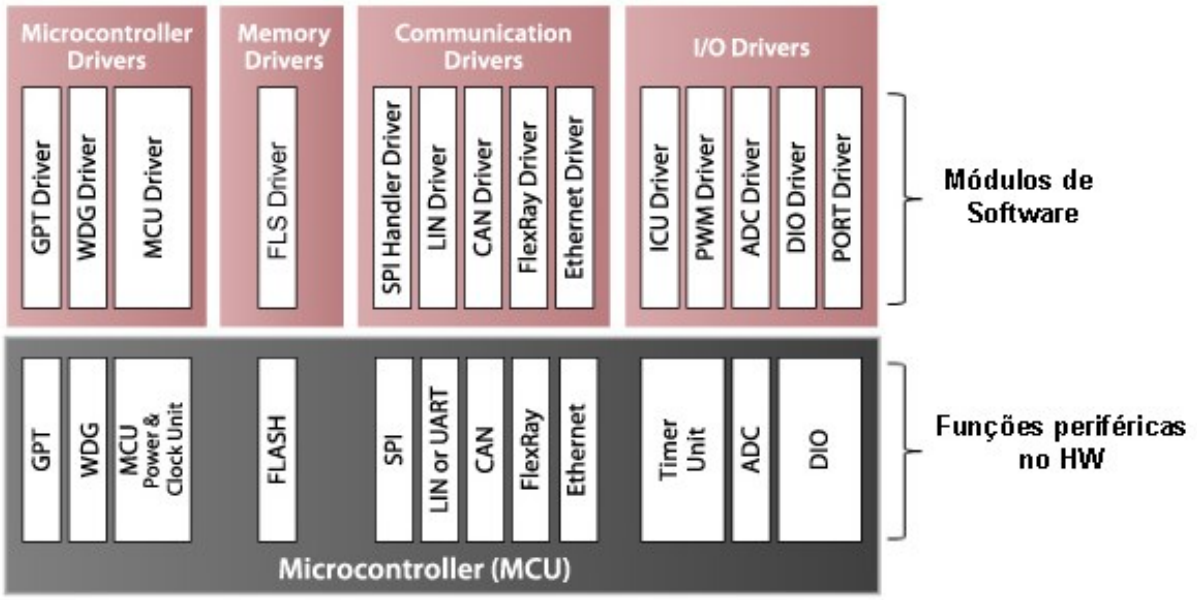
Quadro 2 - Descrição das camadas do padrão AUTOSAR

Nome Camada		Descrição	Característica	Finalidade	
				Montagem	Interfaces
Runtime Environment (RTE)		Camada de middleware para prover serviços de comunicação aos SWC AUTOSAR e aplicações que contenham partes de sensor e atuador AUTOSAR.	Faz com que o SWC AUTOSAR seja independente do mapeamento para uma ECU específica.	Específico para a ECU e aplicação gerada para cada ECU.	A interface é completamente independente da ECU.
Basic Software (BSW)	*Services Layer*	Camada superior do *Software* básico (BSW) que fornece as seguintes funções ・Sistema operacional (OS); ・Comunicação e gerenciamento da rede veicular; ・Serviços de memória (gerenciamento NVRAM);	Fornece serviços básicos e módulos básicos para aplicações.	Específico ao MCU, parte do *hardware* ECU e aplicação.	A interface é independente do MCU e *hardware* ECU.
	ECU Abstraction Layer	Interface com MCAL (incluindo os drivers de dispositivos externos); pode fornecer o seguinte: ・Acesso a periféricos e dispositivos,	Faz com que a camada de *Software* superior seja independente do layout do *hardware* ECU.	A montagem é independente do MCU e dependente do *hardware* ECU.	A interface é independente do MCU e jardware ECU.
	Complex Drivers	Camada de *Software* usada para funções complexas que não são encontradas em outras camadas. Esta camada acessa diretamente o MCU.	Cumpre as funções especiais e requisitos de tempo necessários para operação de sensores e atuadores complexos.	Altamente dependente do MCU, ECU e aplicação.	A interface é padronizada e montada de acordo com AUTOSAR (Interface AUTOSAR).
	Microcontroller Abstraction Layer (MCAL)	Módulo de *Software* que acessa diretamente o MCU, módulos periféricos e dispositivos externos que são mapeados pela memória.	Faz com que a camada de *Software* superior seja independente do MCU.	Dependente do MCU.	Interfaces independentes do MCU padronizado.

Fonte: Autoria própria

O MCAL é um módulo de *Software* que acessa diretamente os módulos periféricos de um MCU, responsável para fornecer acesso direto aos periféricos de microcontroladores e dispositivos externos que fazem partes do *hardware* ECU, e faz com que a camada de *Software* superior seja independente da MCU. Esta camada de *Software* cobre a descrição de drivers de comunicação (FlexRay, SPI, CAN, LIN, etc.), módulos de I/O (ADC, DAC, I/O digitais, PWM, etc.), drivers de memória (Flash, EEPROM, etc.), os dispositivos externos (Memórias externas, sensores e atuadores externos, etc.) e drivers diversos do micro controlador (*Watchdog*, temporizador de propósito geral, etc.). A configuração e nomes de módulos de *Software* diferem dependendo MCU utilizada e a estrutura em camadas do módulo de *Software* MCAL é mostrada na Figura 13.

Figura 13 - Módulos de *Software* para MCAL



Fonte: Renesas (acessado em 2016 e adaptado pelo autor).

O Quadro 3 apresenta a descrição dos módulos de *Software* para MCAL de cada uma das camadas apresentadas na Figura 1312.

Quadro 3 - Descrição dos módulos de Software para MCAL

Grupo Módulo	Módulo	Explicação
Microcontroller Drivers	GPT Driver	GPT (General Purpose Timer) Driver Device driver using on-chip MCU timer Initializes GPT, performs timer count
	WDG Driver	WDG (Watchdog) Driver On-chip WDG device driver Initializes WDG, performs WDG mode settings, etc.
	MCU Driver	MCU (Micro Controller Unit) Driver Device driver that performs MCU settings Initializes clock, performs power mode settings
Memory Drivers	FLS Driver	FLS(Flash) Driver Initializes FLS, reads/writes to FLS memory
	FEE Driver*	FEE (Flash EEPROM) Driver Flash EEPROM emulation driver Initializes FEE, reads/writes to FEE memory *Not supported for RH850
Communication Drivers	SPI Handler/Driver	SPI (Serial Peripheral Interface) Handler/Driver Device driver for on-chip clock serial function Initializes SPI, performs SPI input/output and SPI I/O buffer settings
	LIN Driver	LIN (Local Interconnected network) Driver Initializes LIN, performs LIN input/output
	CAN Driver	CAN (Controller Area Network) Driver Initializes CAN, performs CAN input/output
	FlexRay Driver	FlexRay Driver Initializes FlexRay, performs FlexRay input/output
	Ethernet Driver	Ethernet Driver Initializes Ethernet Driver, performs Ethernet Driver input/output
I/O Drivers	ICU Driver	ICU (Input Capture Unit) Driver Device driver using on-chip MCU timer Initializes ICU, measures PWM waveforms, etc.
	PWM Driver	PWM (Pulse Width Modulation) Driver Device driver using on-chip MCU timer Initializes PWM, outputs PWM waveforms,

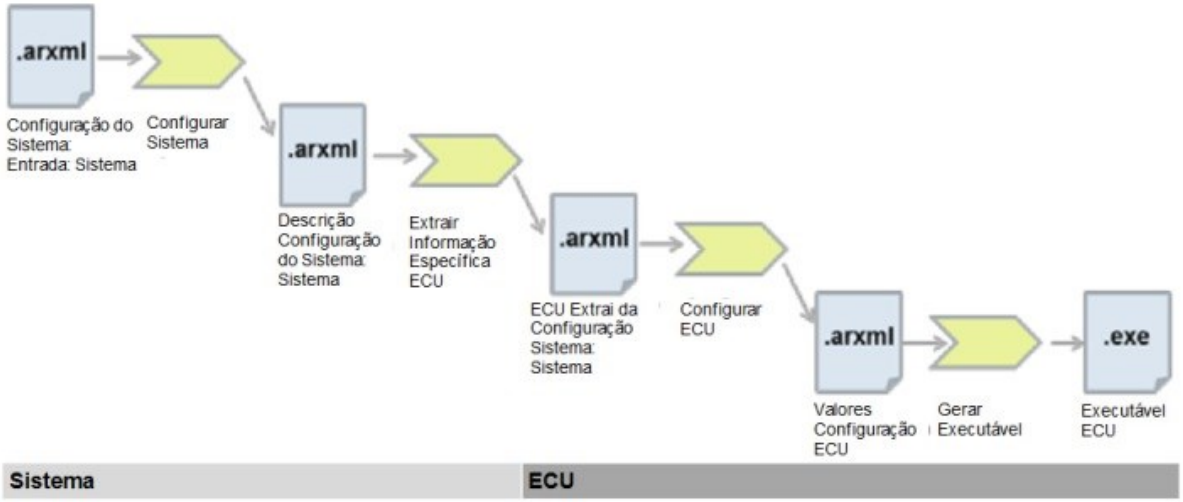
Grupo Módulo	Módulo	Explicação
		etc.
	ADC Driver	ADC (Analog Digital Converter) Driver Device driver for on-chip ADC Initializes ADC, starts/stops AD conversion, sets AD conversion result buffer, reads AD conversion results, etc.
	DIO Driver	DIO (Digital Input/Output) Driver MCU port device driver Performs port signal input/output, etc.
	PORT Driver	PORT Driver MCU port device driver Performs MCU pin settings (I/O, shared functions)

Fonte: Autor

4.2 METODOLOGIA AUTOSAR

A metodologia AUTOSAR descreve as atividades e uso de ferramentas para desenvolvimento de aplicações. Assim, a metodologia não é uma descrição do processo completo, mas sim uma abordagem técnica comum para algumas etapas de desenvolvimento do sistema embarcado automotivo. Ela pode definir atividades e fluxo de trabalho, exceto papel e responsabilidade dos envolvidos. Um meta-modelo AUTOSAR define o conteúdo do produto de trabalho e é uma descrição formal de toda a informação que é produzida ou consumida na metodologia AUTOSAR. A Figura 14 - Esboço de metodologia AUTOSAR. mostra o esboço da metodologia AUTOSAR em cada etapa, onde o *Software* baseado em AUTOSAR é desenvolvido de acordo com um procedimento a ser seguido.

Figura 14 - Esboço de metodologia AUTOSAR.



Fonte: Lee (2014) Adaptado pelo autor.

Além de estabelecer uma arquitetura de *Software* automotiva padronizada, o AUTOSAR cria e estabelece uma metodologia de desenvolvimento para a geração líquida de E&E sistemas automotivos. Esta abordagem utiliza quatro passos principais, citados no Quadro 4.

Quadro 4 - Passos para metodologia AUTOSAR.

Passo 1: Descrições das entradas
No primeiro passo, é a descrição de entrada para os componentes de *Software*, o sistema de *hardware* e que podem ser definidos em três descrições, como segue: • Componentes de *Software*: Esta é uma descrição com independência da execução efetiva do componente de *Software* e estão entre os dados que devem ser especificados são as interfaces e os requisitos de *hardware*. • Sistema: A topologia do sistema é uma visão global de todas as ECU sistema e suas interconexões através de barramento de comunicação que descrevem os barramentos de dados disponíveis, protocolos, função de agrupamento e de matriz de comunicação e atributos (por exemplo, taxas de dados, o tempo, latência, etc.). • Hardware: O *hardware* disponível (processadores, sensores, atuadores, etc.) que composse na ECU, precisa ser especificada juntamente com os métodos de processamento de sinal e as capacidades de programação.
Passo 2: Configuração do sistema
Esta etapa distribui as descrições dos componentes de *Software* para a ECU diferente, que é um processo iterativo onde os recursos ECU e as limitações do sistema são tidos em conta. Por exemplo, é necessário que haja uma verificação ou não o necessário, se as velocidades de comunicação sejam atendidas.
Passo 3: Configuração ECU
Nesta etapa, o Basic *Software* (BSW) e do *Runtime Environment* (RTE) de cada Unidade de Controle Eletrônico (ECU) é configurado e esta se baseia na concessão de componentes de *Software* de aplicação para cada ECU específica.

Passo 4: Geração de *Software* executáveis
No fim da configuração do passo anterior, os executáveis de *Software* são gerados. Para esta etapa, é necessário especificar a implementação de cada componente de *Software*.

Fonte: Autor

Esta metodologia, passando por suas etapas até a geração de executáveis que são suportados pela definição de formatos de troca (usando XML) e métodos de trabalho é automatizada usando uma cadeia de ferramentas específicas para o padrão.

Para apoiar a metodologia AUTOSAR, um meta-modelo é desenvolvido. Este modelo é uma descrição formal de todas as informações relacionadas com a metodologia modelada em UML. Isto leva aos seguintes benefícios:

- A estrutura de informação pode ser claramente visualizada;
- A consistência das informações pode ser garantida;
- A XML é um formato de intercâmbio de dados que pode ser gerado automaticamente para fora do meta-modelo e ser utilizado como entrada para a metodologia;
- Facilidade para manutenção dos subsistemas em todo o sistema veicular.

4.2.1 Projeto AUTOSAR

O desenvolvimento de *Software* baseado na arquitetura AUTOSAR é uma abordagem que abrange etapas que serão apresentadas graficamente e textualmente a seguir.

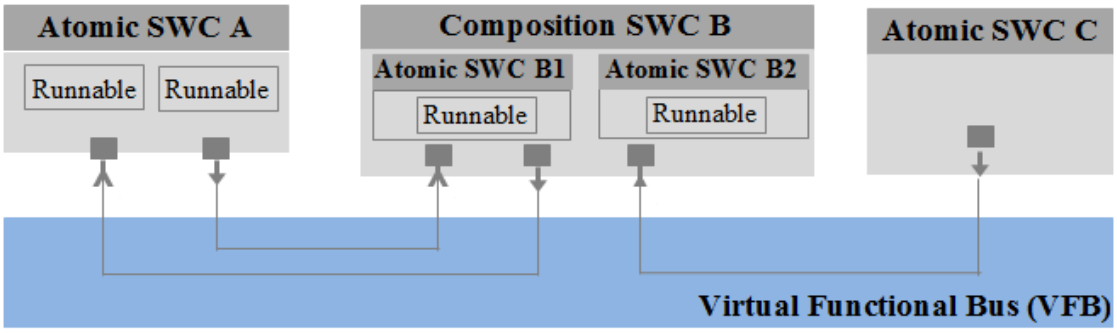
Na primeira etapa é necessário definir a funcionalidade para o *Software* que será aplicado no sistema desejado. Essa funcionalidade é definida pelo *Software Component* ou componente de *Software* (SWC). O SWC é um dos principais elementos da arquitetura AUTOSAR e é a aplicação em *Software* que será executada na ECU. Além da funcionalidade é também necessário definir o modo que este componente de *Software* irá se comunicar. Esta comunicação é chamada de *Virtual Functional Bus* ou barramento funcional virtual (VFB), que tem como função permitir a implementação independente dos componentes de *Software* com a camada subjacente de *hardware*. Através desta abstração providenciada pelo VFB a

interação entre todos os componentes é validada. O protocolo utilizado normalmente é CAN, LIN, FlexRay, MOST ou outros.

A Figura 15 - Arquitetura de Software base em componentes de Software.ilustra uma aplicação que engloba três SWC interligados via VFB e possuem seus comportamentos definidos por entidades executáveis chamadas *runnables*. Um componente de *Software* pode possuir um ou mais *runnables*. Neste nível de abstração, a principal preocupação é desenvolver a função que será encapsulada no SWC e não tem ligação direta com os aspectos relacionados ao *hardware* (ECU) ou barramento de comunicação. Porém esta relação será feita em estágios subsequentes e é parte importante para o entendimento da arquitetura.

O comportamento dos SWC, demonstrado pelo funcionamento dos *runnables*, pode ser desenvolvido através de código em C, Misra-C, tabelas e máquinas de estado ou diagramas de blocos. Isto dá ao desenvolvedor a possibilidade de reutilizar *Software* criado anteriormente facilitando o desenvolvimento de novas funcionalidades.

Figura 15 - Arquitetura de Software base em componentes de Software.



Fonte: Autor

Os SWC são unidades de *Software* que possuem portas de comunicação com outros SWC. Estas portas de comunicação são tipificadas de acordo com a interface, como por exemplo, por um padrão emissor/receptor (*sender/receiver*) para troca de dados, ou uma função cliente/servidor para um tipo de operação. Os SWC são classificados como demonstrado no Quadro 5.

Quadro 5 - Classificação de componentes de Software.

Type	Utilização	Implementado por
Aplicação	*Software* Funcional	Usuário (Baseado em código ou modelo)
Sensor/Atuador	Acesso as I/O (via abstração de I/O de Hardware)	Usuário (Baseado em código ou modelo)
Parâmetro	Acesso a parâmetros calibráveis	Gerador de RTE
Bloco de memória não volátil	Acesso ao bloco de memória não volátil	Gerador de RTE
I/O Abstração de Hardware	Acesso aos canais de I/O	Usuário e parcialmente o gerador de RTE
Drivers Complexos	Acesso a funcionalidades especiais	Usuário ou gerador de RTE
Serviços	Acesso ao serviço de Basic *Software* via RTE	Gerador de Basic *Software*
Composição	Agregação de outros *Software*s	n.d.

Fonte: Autor

O comportamento interno de um *Software* é modelado através dos *runnable*, e a interface de programação (API) do SWC é derivada de uma configuração detalhada dos *runnables*. Para cada entidade *runnable* o acesso a uma porta é definido por uma referência a um subconjunto das portas dos SWC, como demonstrado no Quadro 6.

Quadro 6 - Comunicação de portas.

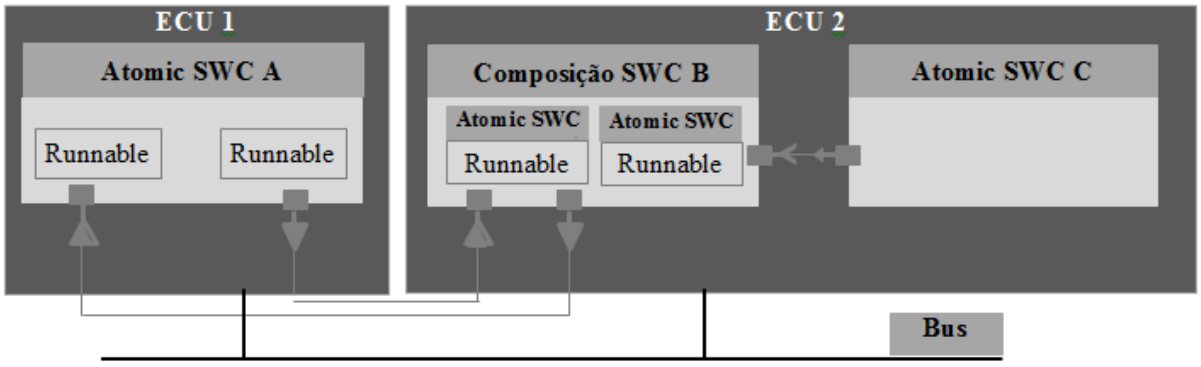
Port Access	Utilização	API
DataReadAccess	Ler elementos de dados que não estão em fila	Rt1_Read, Rte_IRead, Rte_Dread
DataReceivePoint	Ler elementos de dados que estão em fila	Rte_Receive, Rte_IsUpdated
DataWriteAccess	Escreve elementos de dados que não estão em fila	Rte_Write, Rte_IWrite, Rte_IWriteRef, Rte_IInvalide
DataSendPoint	Escreve elementos de dados que estão em fila	Rte_Send, Rte_Invalidate
ServerCallPoint	Invocação síncrona ou assíncrona de uma operação de servidor	Rte_Call
ModeSwitchPoint	Inicia uma mudança de modo através de uma porta modo sender	Rte_Switch
ModeAccessPoint	Ler o modo de uma porta modo *receiver*	Rte_Mode

Fonte: Autor

Assim, os componentes de *Software* definidas na primeira fase devem ser mapeados na segunda fase do projeto, onde SWC A é mapeado na ECU 1 e o SWC B e o SWC C são mapeados na ECU 2. A Figura 16 e Figura 17 demonstram como os SWC se relacionam com as ECU e com o barramento de comunicação.

Depois de implementar os SWC nas ECU, a comunicação da *Virtual Function Bus* (VFB) resulta em comunicação local da ECU ou baseada em comunicação em rede segundo os protocolos CAN, LIN, FlexRay ou Ethernet.

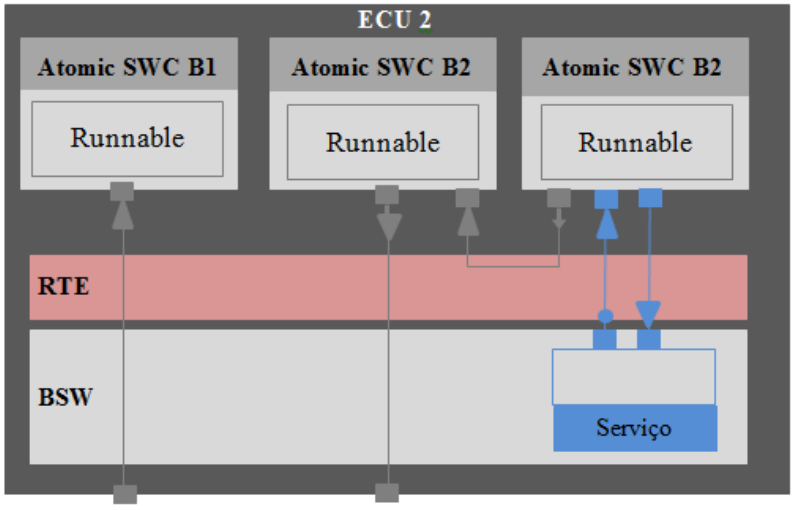
Figura 16 - Distribuição de componentes de *Software* e *Runnables* nas ECU.



Fonte: Autor

Para integração de ECU, o ambiente de tempo de execução (*runtime environment* ou RTE) e os módulos de *Software* básico (*Basic Software* ou BSW) são configurados para atender os requisitos definidos pelos SWC.

Figura 17 - Software arquitetura base em componentes de Software.



Fonte: Autor

4.2.2 AUTOSAR e o Desenvolvimento baseado em modelos

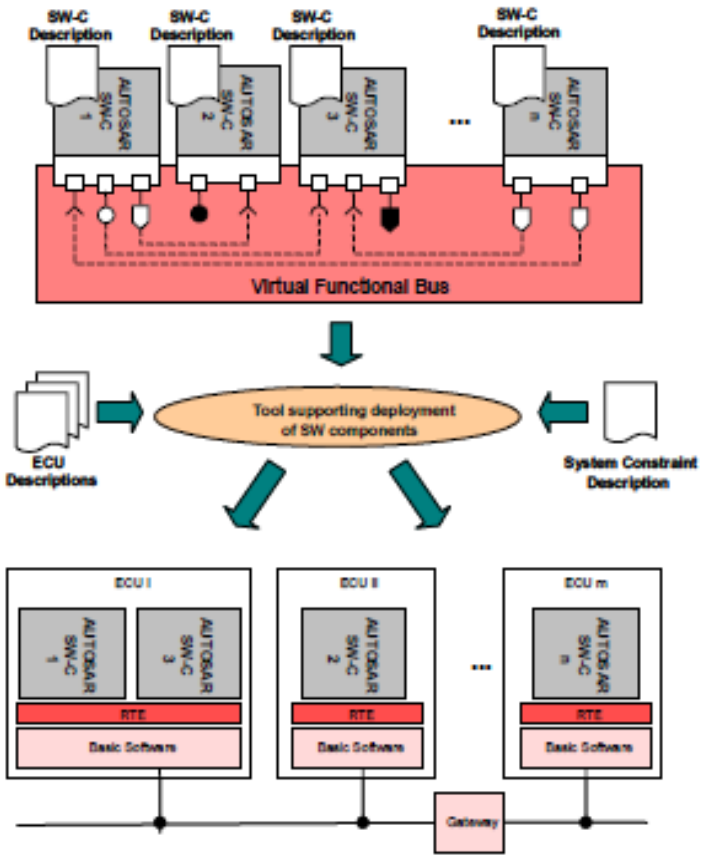
O padrão AUTOSAR permite o uso do Design Baseado em Modelo para desenvolvimento de componentes de *Software* de um sistema. O MBD já é um grande avanço para a padronização do desenvolvimento de *Software* permitindo a reutilização de projetos e garantia de segurança. A utilização do padrão AUTOSAR dentro da metodologia do MBD forma uma ferramenta poderosa para o desenvolvimento de *Software* automotivo.

Um dos fatores que facilita esta integração entre estas ferramentas é o nível de abstração que o AUTOSAR cria entre os SWC e o *hardware*. Isso possibilita que os SWC sejam desenvolvidos por completo seguindo os preceitos do MBD sem necessariamente haver a preocupação de qual *hardware* será utilizado na validação final. A Figura 18 mostra ilustrativamente o padrão da metodologia AUTOSAR, em qual há um desacoplamento entre as etapas de desenvolvimento das aplicações entre o *Software* básico e o *Software* que será utilizado.

O projeto baseado em modelos utiliza aplicações em SWC que irão se conectar via VFB. O VFB permite que sejam executados testes, integração e validação de forma simulada, em *hardware*s simples e até em ECU. Isso resulta em economia de tempo, custos e minimiza problemas que possam ser encontrados no desenvolvimento tradicional de *Software* automotivo.

O SWC é a menor parte funcional de um *Software*, que pode ser formado por vários SWC com diferentes aplicações em componentes. Os SWC podem ser criados separadamente em MBD flexibilizando as aplicações e permitindo a reutilização em futuros desenvolvimentos. A padronização das interfaces para todos os SWC, necessários para a construção de uma aplicação automotiva, são especificados nos padrões AUTOSAR e se encaixam perfeitamente para as etapas de verificação e validação no MBD.

Figura 18 - A metodologia padrão AUTOSAR

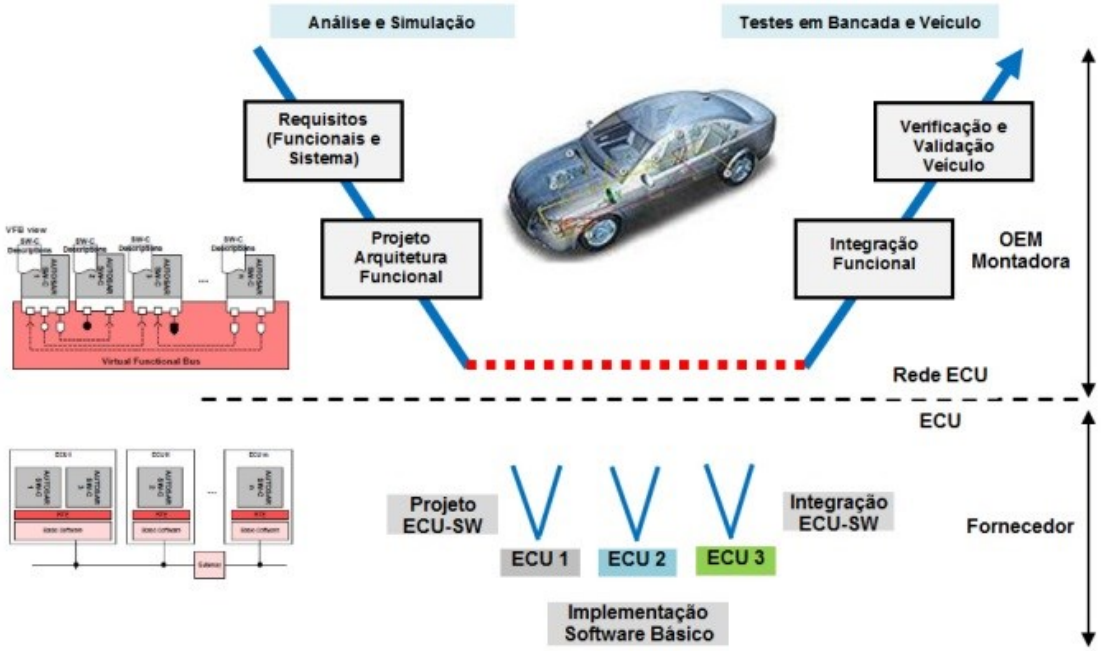


Fonte: Nishikawa (2006)

O VFB conecta os diferentes componentes de *Software* no modelo de um projeto. Este componente abstrato, em fases de verificação, conecta os diferentes componentes de *Software* que forem desenvolvidos em modelos e facilita a troca de informações entre eles. Isso permite que os desenvolvedores se concentrem nos testes e aplicação, em vez de se preocupar precocemente com a validação final.

A metodologia AUTOSAR segue normalmente o Modelo-V de Desenvolvimento, dando condições para validar a integração da ECU fase preliminares através da VFB, onde os componentes de *Software* têm interfaces bem definidas e comunicações padronizadas através das portas. A Figura 19 mostra claramente como a metodologia AUTOSAR dentro do Modelo-V pode trabalhar com maior eficiência.

Figura 19 - Metodologia AUTOSAR considerando o Modelo-V.



Fonte: Nishikawa (2006)

Assim, pode ser visto como o Modelo-V e o design baseado em modelos, para desenvolvimento de *Software* automotivo, através da sua metodologia pode apoiar o padrão AUTOSAR de forma mais eficiente do que o tradicional.

4.3 VALIDAÇÃO VIRTUAL DE UM SWC

Para atender os desafios de segurança e conformidade no desenvolvimento de *Software* automotivo é necessário seguir uma abordagem estruturada. Esta abordagem é caracterizada por uma fase de projeto bem enfatizada.

Esta fase de desenvolvimento irá fornecer uma base inicial confiável para a implementação concreta dos SWC dentro da rede de comunicação do veículo. Esta fase é totalmente pautada nos preceitos do MBD. Este balizamento permitirá a avaliação da conformidade com os requisitos, a adequação da topologia da rede de comunicação e o particionamento das funções com respeito ao número de ECU, gateways, carga e outras particularidades da rede.

A grande vantagem de esta fase ser bem estruturada é a possibilidade de executar a modelagem e a simulação dentro de um protótipo de uma rede e não apenas de funções individuais em *hardware*s independentes da rede.

4.3.1 Desenvolvimento virtual e testes

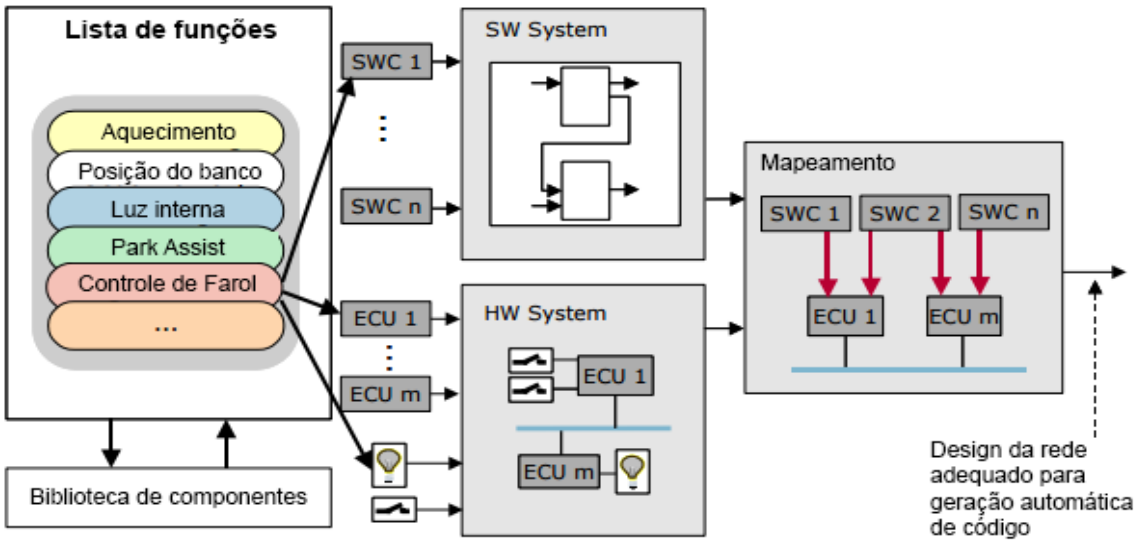
A abordagem que será demonstrada é aplicada para o nível lógico mais alto, que corresponde a parte da empresa que irá desenvolver apenas o *Software* a ser embarcado. A aplicação sequencial dos itens a seguir é importante para que o desenvolvimento das funções em rede seja dominado totalmente:

- Processo de desenvolvimento estruturado iniciado com a analise dos pré-requisitos;
- Desenvolvimento da rede a partir do topo assim como a verificação e validação executada em fases iniciais do desenvolvimento de cada componente.
- Desenvolvimento a partir da interação do Software com o hardware levando em consideração os resultados de testes que ocorram durante o desenvolvimento;
- Mapeamento e documentação de toda a sequencia da abordagem utilizando ferramentas específicas.

A fase da definição dos requisitos também deve definir, além das funções a serem desenvolvidas, testes e cenários de aplicação.

A fase de design da rede inicia-se com a listagem das funções que serão implementadas e a separação de cada função em *hardware*, *Software* e mecanismos de aplicação. As funções implementadas em *Software* podem ser divididas em sub funções resultando em futuros sub SWC. O resultado desta fase é um modelo completo da rede que pode ser simulado após a geração de código, como demonstra a **Erro! Fonte de referência não encontrada.**.

Figura 20 - Esquemático da fase de desenvolvimento



Fonte: Nishikawa (2006)

O SWC é o principal elemento para a modelagem estruturada de funções do veículo. Ele consiste em uma caixa preta, com interfaces de sinal bem definidas e orientadas. A semântica do sinal é indicada pelo nome do sinal. A especificação do tipo de um sinal consiste da largura de bits, fórmula de conversão, valor padrão e constantes simbólicas.

Um componente de *Software* encapsula completamente suas funções intrínsecas. Estas funções intrínsecas são um modelo de comportamento ou uma subestrutura com outros componentes de *Software*. Com isso uma função pode ser mapeada em um único componente de *Software*, podendo ser então hierarquicamente decomposta em sub componentes até que um detalhamento razoável seja atingido.

Além desta abordagem de cima para baixo, uma abordagem de baixo para cima é aplicável, como quando está se desenvolvendo um componente de *Software* de nível superior usando componentes já existentes armazenados em uma biblioteca de componentes, como demonstra o bloco inferior do lado esquerdo da **Erro! Fonte de referência não encontrada.**.

O SWC que já foi devidamente particionado é representado por uma descrição comportamental. Esta descrição pode ser feita, por exemplo, da seguinte forma:

- Um componente de Software com suas interfaces de sinal é implementado como um modelo de comportamento com interfaces equivalentes

utilizando uma ferramenta de modelagem de comportamento, como por exemplo, o Simulink;

- Estes modelos de comportamento podem ser simulados dentro desses ambientes de Software, isto é, eles podem ser testados em cenários de teste correspondentes e avaliados quanto ao seu resultado esperado segundo os pré-requisitos;
- Os geradores de código destas ferramentas podem ser configurados de forma adequada para garantir a consistência do código gerado com a interface de destino.
- O código a ser embarcado no micro controlador é implementado diretamente. Os sinais de entrada e saída do componente podem ser acessados através de uma interface macro. O acesso direto a recursos específicos ECU (por exemplo, acessar a um registo específico) não é necessário

O modelo comportamental descreve a tarefa essencial do SWC, por exemplo, a execução de um algoritmo específico. A independência é uma característica importante do componente de *Software*, seja do *hardware* ou de outros componentes de *Software*. Por esta razão, uma função específica implementado em *Software* pode ser desenvolvida e testada como um SWC autónomo e, em seguida, armazenado na biblioteca de componentes.

Para executar a funcionalidade geral desejada do veículo todas as funções desenvolvidas para este veículo, implementadas em *Software*, são integrados a um sistema de *Software*.

5 PAINEL DE INSTRUMENTOS AUTOMOTIVO

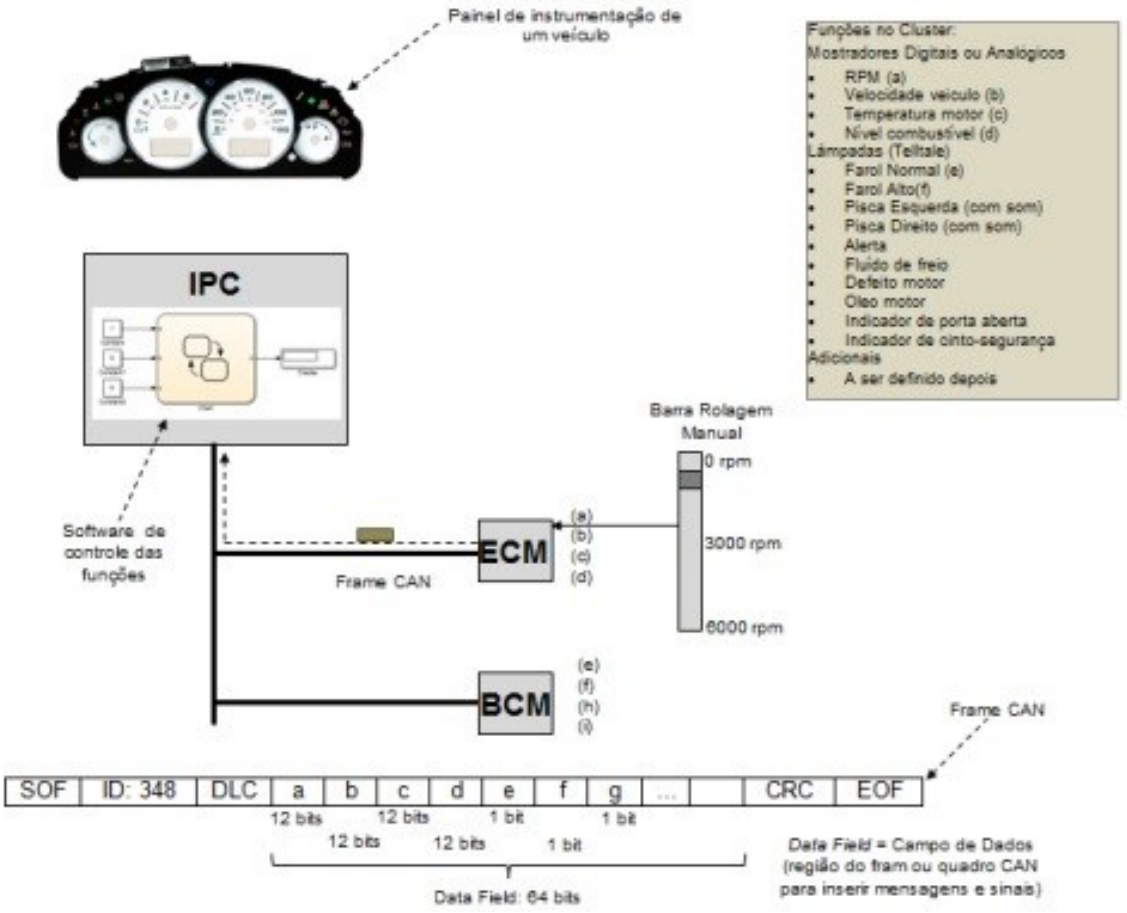
O painel de instrumentos ou *dashboard* é um componente que disponibiliza ao condutor todas as informações relevantes do veículo em tempo real, tais como velocidade do veículo, rotação do motor, nível do combustível, lâmpadas de advertência para quando da ocorrência de alguma anomalia entre outras possíveis. A denominação *Instrument Cluster (*IC) é dada ao componente em si visível ao condutor e a nomenclatura *Instrument Panel Control* (IPC) ou *Instrument Cluster Module* (ICM) como a unidade de controle eletrônica (ECU) que gerencia este componente.

De forma ilustrativa é possível demonstrar os passos para disponibilizar a visualização no IC do valor da rotação do motor em que é processada pelo IPC. Então a sequencia de passos operacionais para que isto ocorra são as seguintes:

1. Sensor de RPM lê sinal e disponibiliza na memória da ECM;
2. O sinal de RPM é colocado no Frame CAN correspondente na função CAN Pack;
3. Deve-se fazer um Event Driven Controller;
4. O Frame que por exemplo ID 381 é transmitido por exemplo periodicamente por exemplo de T = 10 ms com um Baudrate de 250 Kbps;
5. O IPC através do CAN Unpack recebe o frame CAN com o sinal de RPM;
6. Desempacota o sinal e através de outro Event Driven Controller, disponibiliza este sinal no Dashboard como de um veículo;
7. Este deve ser um primeiro teste para validar a lógica.

A arquitetura distribuída para um veículo com *Instrument Cluster* pode ser implementada em uma ferramenta de desenvolvimento virtual como Simulink e StateFlow da MathWorks. A Figura 21 demonstra a arquitetura física de implementação.

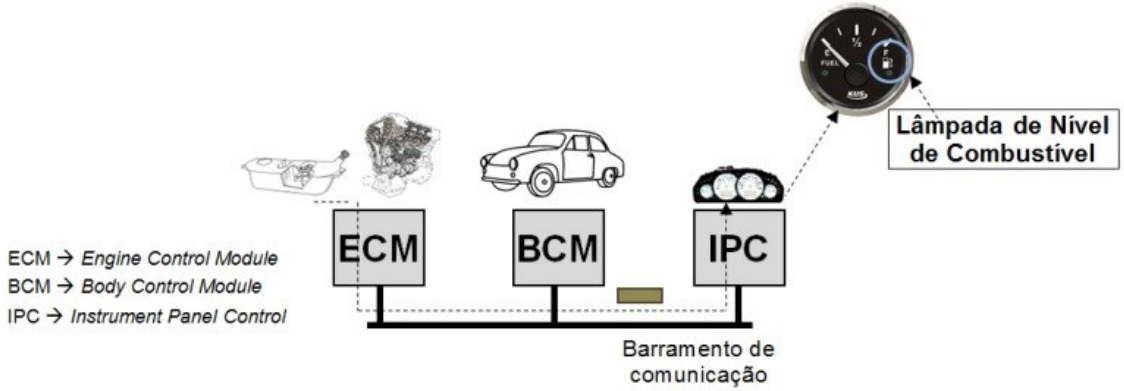
Figura 21 - Arquitetura distribuída física para disponibilização da velocidade rotação do motor no painel de instrumentação.



Fonte: Autor

O sistema de controle de lâmpada de nível de combustível é caracterizado por um controlador baseado em eventos. A Figura 22 apresenta uma descrição de um sistema de controle da lâmpada de nível baixo de combustível. Note que a partir de uma configuração com arquitetura distribuída, tem-se três ECU respectivamente descritas na Figura 22.

Figura 22 - Arquitetura distribuída com suporte à função de controle de lâmpada nível de combustível.



Fonte: Autor

O sensor de nível do tanque está conectado a ECM e gera um sinal elétrico que será convertido na forma binária e encapsulado num quadro de dados para uma rede CAN o qual é transmitido até ao IPC em que tem conectado o *cluster* com a lâmpada indicadora de nível baixo de combustível. O sensor de nível de combustível instalado no reservatório, tem a capacidade de medir o nível exato dentro do tanque de combustível do veículo e gerar um sinal elétrico analógico proporcional ao nível de combustível medido, dentro de uma faixa entre 0 e 10 V.

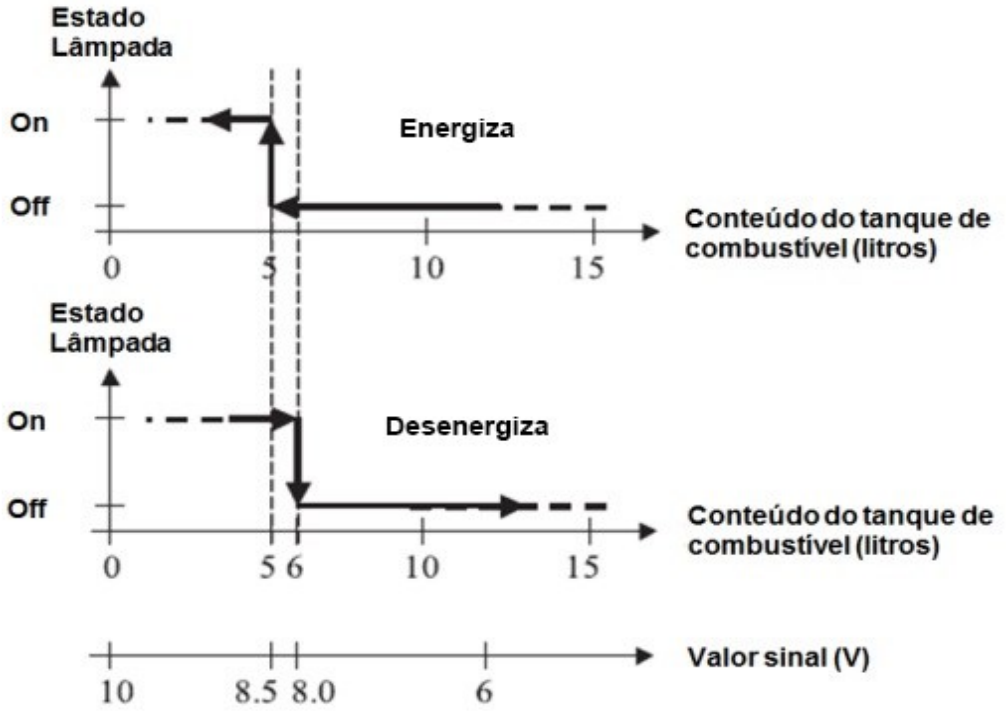
O controlador de lâmpada de nível de combustível (indicação de nível baixo de combustível) é realizado por um *Software* de controle na forma de um controlador baseado em eventos. Considera-se que a ação de controle da lâmpada ocorrerá no intervalo quando o tanque estiver entre 0 a 15 litros.

Este sinal analógico é utilizado como sinal de entrada para controle da lâmpada de indicação de nível baixo de combustível. A Figura 23 mostra a lógica de controle para acendimento da lâmpada de nível baixo de combustível através de chaveamento. Note que quando o sensor disponibilizar um sinal de 10 volts representa que o nível do tanque está em 0 litro e em 8,0 volts tem-se 6 litros. Através de uma regra de três simples, é possível ter-se uma representação através de uma equação de primeiro grau entre nível do tanque e nível do sinal elétrico emitido pelo sensor. A equação que representa este sistema é dada por $y = -1/3x + 10$, sendo que o sensor consegue medir o nível do tanque em até 30 litros, porque para um sinal elétrico de $y = 0$ volt tem-se que a variável $x = 30$ litros.

A histerese da função de controle é definida entre 8,5 a 8,0 volts e 5 a 6 litros. A lâmpada possui dois estados sendo *Off* ou Desligada (estado inicial) e *On* ou Ligada. Para entrar em estado *Off*, o Valor do Sinal deve ser menor do que 8,0 Volts, que representa que o nível do tanque de combustível é maior do que 6 litros e em estado *On* com Valor do Sinal

maior que 8,5 Volts representando que o nível do tanque de combustível é menor do que 5 litros. Para valores entre 5 e 6 litros, o estado da lâmpada é determinado em função do estado anterior em que o sensor de nível realizou a medição.

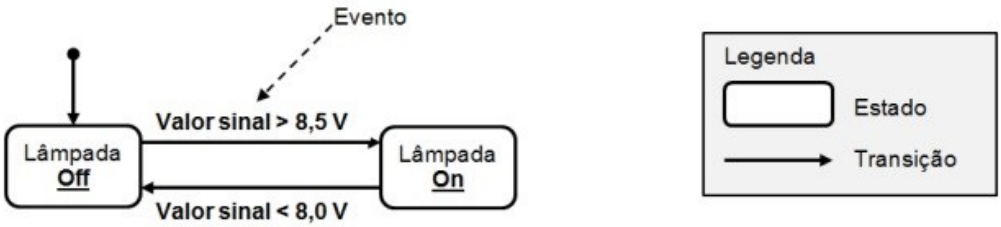
Figura 23- Operações de chaveamento de uma lâmpada indicadora de nível baixo de combustível



Fonte: Schauffele, 2005.

O controlador da lâmpada pode ser implementado através de uma máquina de estados finita representado por um diagrama dos estados finitos e discretos. Este diagrama alternaria o controle da lâmpada entre "*Lamp off*" (Lâmpada *Off*) e "*Lamp on*" (Lâmpada *On*), como pode ser visto na Figura 24, através das possíveis transições entre os estados, atribuindo o evento correspondente.

Figura 24 - Gráfico de estado/transição de uma lâmpada indicadora de nível baixo de combustível



Fonte: Schauffele, 2005.

5.1 DESENVOLVIMENTO DAS FUNÇÕES DO PAINEL DE INSTRUMENTOS AUTOMOTIVO BASEADO EM MODELOS

A primeira etapa para o desenvolvimento do projeto da função do Painel de Instrumentos em modelo é a definição dos pré-requisitos de funcionamento do sistema. Os pré requisitos definidos estão no Quadro 7:

Quadro 7 - Pré-requisitos de funcionamento 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: Autor

Definido os requisitos e levando em conta o funcionamento óbvio do painel, partiu-se para o desenvolvimento do modelo para a etapa de MIL.

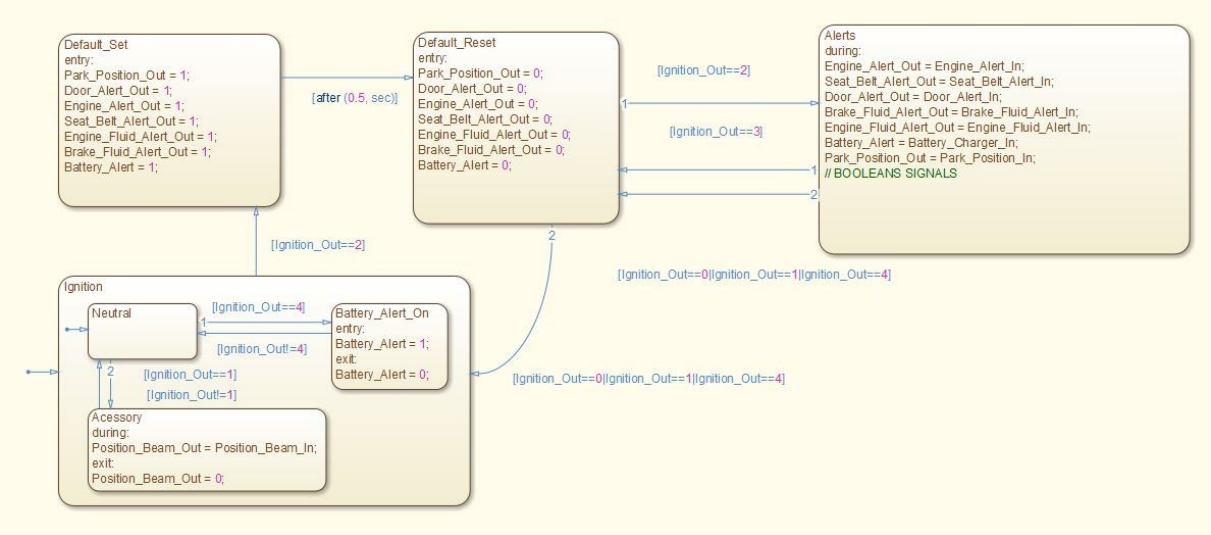
5.1.1 Model-in-the-Loop para a função de um Painel de Instrumentos Automotivo.

As funções do Painel de Instrumentos foram divididas em grupos, a saber:

- Alerts: com os principais alertas do Painel;
- Lights: com informações sobre a situação das luzes do carro;
- TPMS: com informações sobre o estado correto ou não da calibragem
- Gauges: Com as informações demonstradas em medidores de ponteiros;
- Ignition: com a lógica de habilitação ou não do sistema.

Utilizando a ferramenta Stateflow do Simulink (MathWorks), desenvolveu-se a lógica de funcionamento de cada um destes grupos.A Figura 25 demonstra a lógica de funcionamento do grupo Alerts.

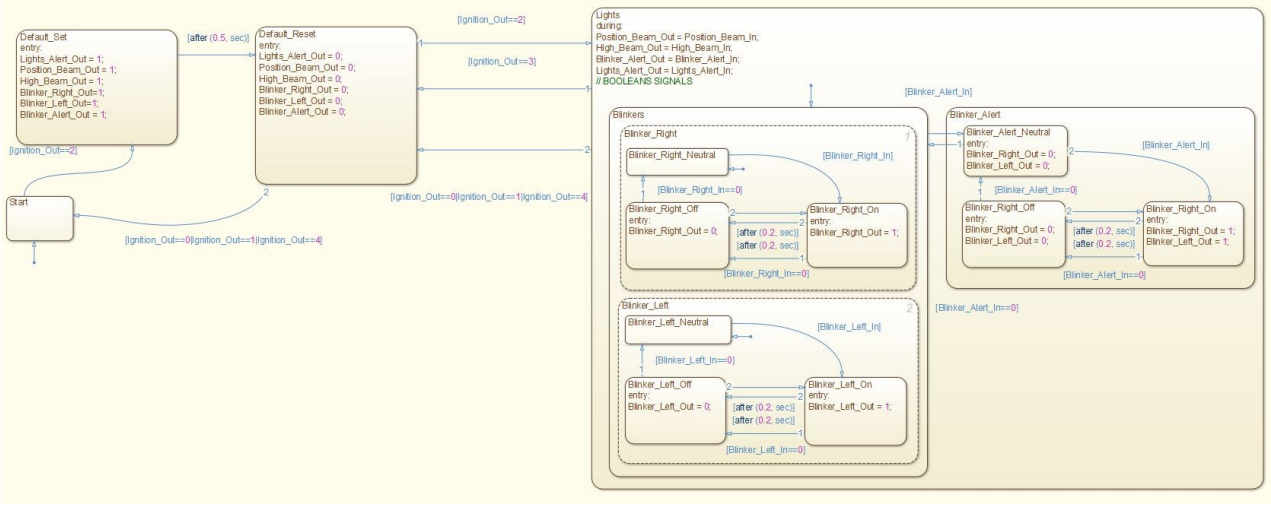
Figura 25 - Lógica de funcionamento do grupo Alerts em Simulink/Stateflow



Fonte: Autor

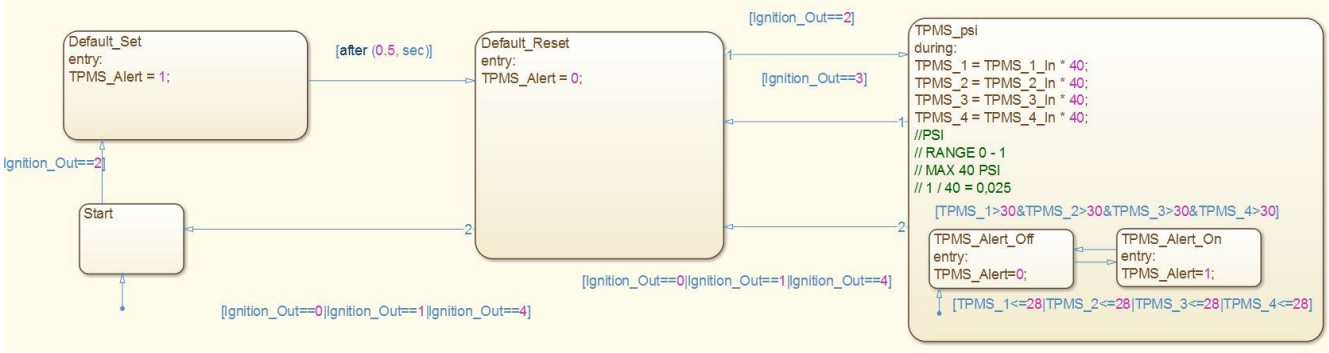
O desenvolvimento destas lógicas é bastante facilitado utilizando Stateflow. A ferramenta possui transições de estados de fácil incorporação e inserção de funções já conhecidas por remeterem a linguagem C. A Figura 26 demonstra a lógica de funcionamento do grupo Lights e a Figura 27 traz a lógica de funcionamento do grupo TPMS:

Figura 26 - Funcionamento do grupo *Lights* em Simulink/Stateflow



Fonte: Autor

Figura 27 - Grupo *TPMS* em Simulink/Stateflow

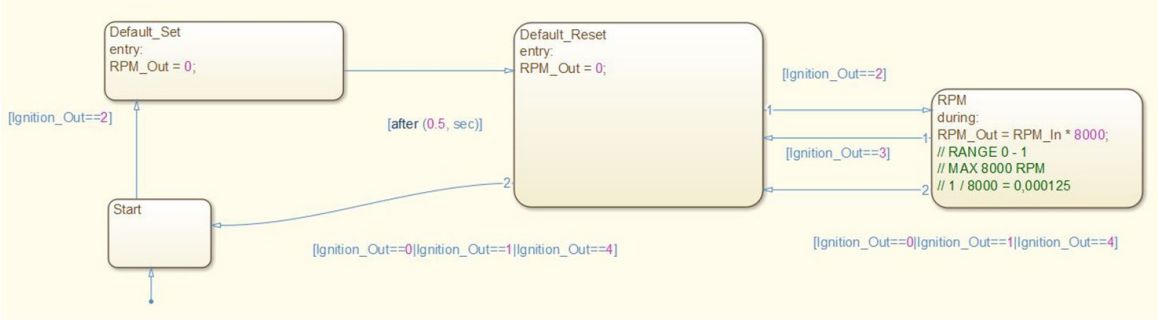


Fonte: Autor

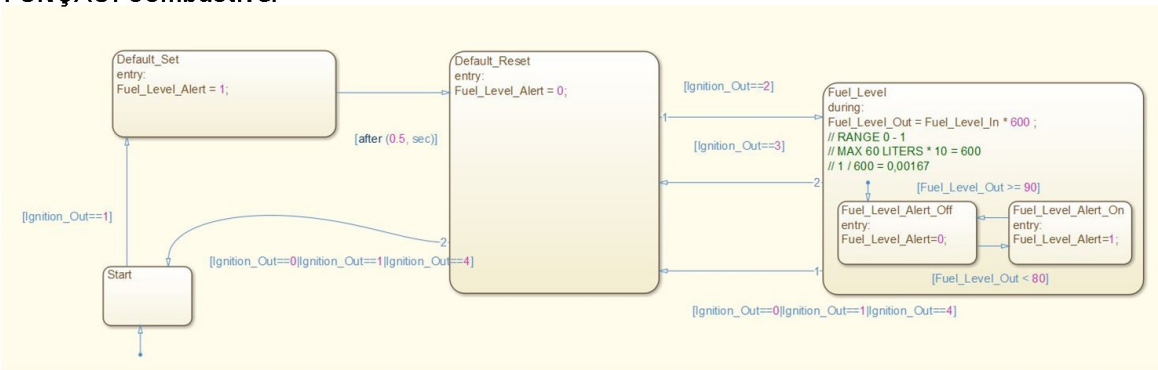
A Figura 28 apresenta a lógica de funcionando de todas as informações no grupo Gauges:

Figura 28 - Lógica de funcionamento dos mostradores de ponteiro em Simulink/Stateflow

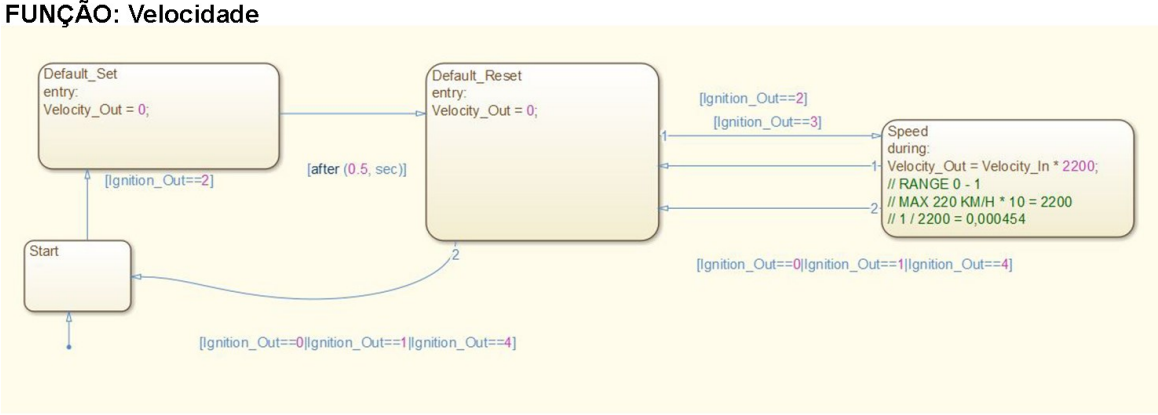
FUNÇÃO: RPM



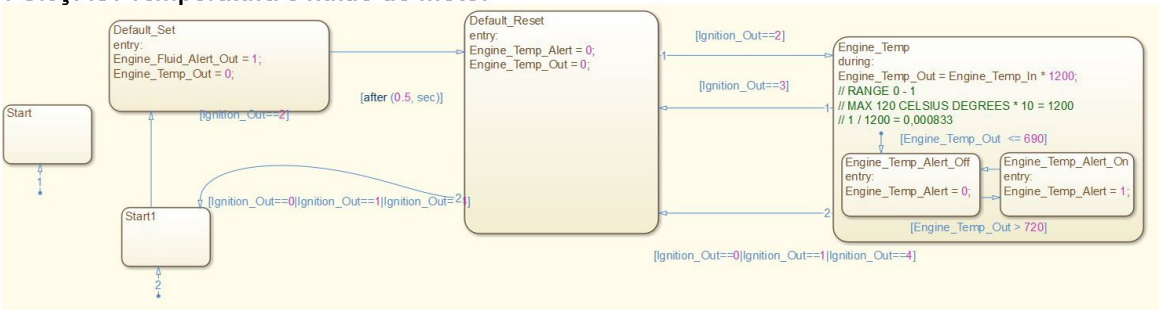
FUNÇÃO: Combustível



FUNÇÃO: Velocidade



FUNÇÃO: Temperatura e fluido do motor



Fonte: Autor

Por último, a função que habilita ou não o sistema foi desenvolvida em forma de uma tabela verdade, porém também dentro do ambiente Simulink/Stateflow, mostrada na Figura 29.

Os grupos foram criados em modelos diferentes no Simulink, pois desta forma é possível criar componentes de software diferentes, permitindo uma total modularização do painel. Caso outro projeto de painel de instrumentos seja desejado com mais ou menos funções, é necessário apenas selecionar qual componente de software faz parte do novo projeto.

Figura 29 - Tabela verdade com a lógica de habilitação do sistema

Condition Table

	Description	Condition	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	D13	D14	D15	D16
1	Acessory	signal_1 == 1	F	F	F	F	F	F	F	F	T	T	T	T	T	T	T	T
2	Run	signal_2 == 1	F	F	F	F	T	T	T	T	F	F	F	F	T	T	T	T
3	Cranck	signal_3 == 1	F	F	T	T	F	F	T	T	F	F	T	T	F	F	T	T
4	Battery	signal_4 == 1	F	T	F	T	F	T	F	T	F	T	F	T	F	T	F	T
		Actions: Specify a row from the Action Table	5	1	5	1	5	1	5	1	5	2	5	1	5	3	5	4

Action Table

#	Description	Action
1	Neutral	Out_signal = 0;
2	Acc	Out_signal = 1;
3	Run	Out_signal = 2;
4	Crank	Out_signal = 3;
5	Battery_Alert	Out_signal = 4;

Fonte: Autor

Os valores assumidos pelo sinal de habilitação (Ignition Out), representado na tabela "*Action Table*" determinam os modos de operação do sistema.

O sistema de ignição modifica o valor da saída Ignition_Out, assumindo valores de 0 a 4. Quando Ignition_Out assume valor 0 o sistema fica desabilitado. No 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. No 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 sistema.

As funções de velocidade, RPM, temperatura e fluido do motor, TPMS e combustível, 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.

A função 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).

Na função *TPMS* 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 94 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.

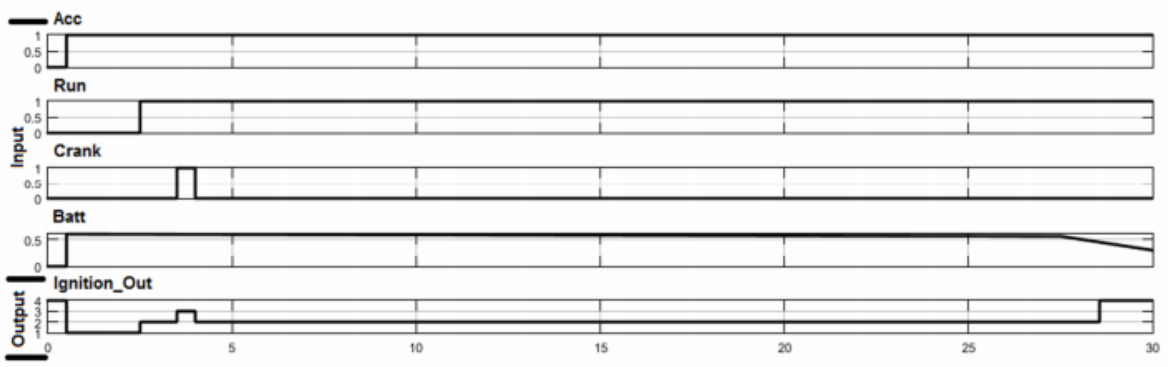
As demais funções consistem em tratamento de sinais booleanos de entrada. A função *Lights* consiste em verificações dos sinais booleanos para a ativação das lâmpadas.

5.1.2 Simulação e teste para verificação da função em MIL

Para se efetuar a verificação das funções modeladas em MIL, aplicam-se sinais de entrada simulando uma situação real de utilização do painel de instrumentos. Observando-se as saídas é possível analisar se os requisitos de projetos foram atendidos.

Na Figura 30 são apresentados os sinais que alimentam a função de habilitação do sistema, *Acc*, *Run* e *Crank* que correspondente à sequência de rotação da chave de ignição. O sinal referente a tensão da bateria faz parte também deste sistema de habilitação. O sinal de saída deste sistema é demonstrado no gráfico “*Ignition_out*”.

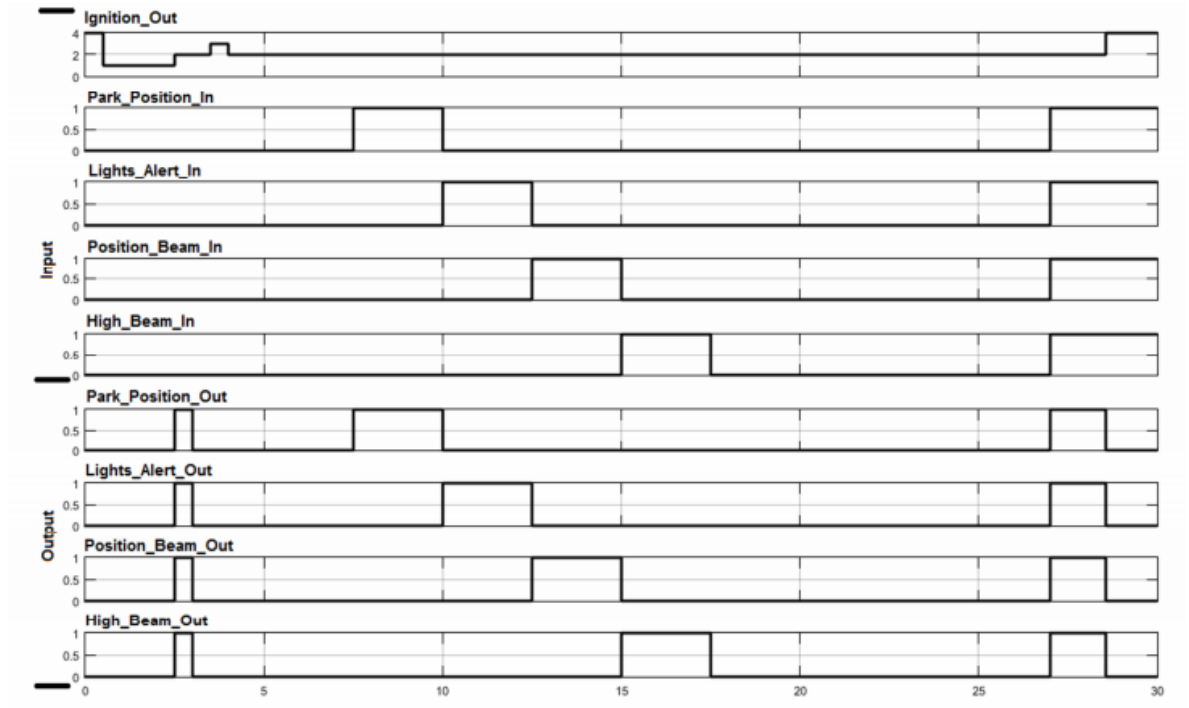
Figura 30 - Sinais simulados do sistema de habilitação do painel de instrumentos



Fonte: Autor

Nos próximos gráficos temporais serão demonstrados outros sinais de entradas seguidos pelos sinais de saída, permitindo a verificação das lógicas ds funções. Inicialmente a lógica observada será do grupo *Lights*, na Figura 31.

Figura 31 - Sinais de entrada e saída do grupo *Lights* para verificação da lógica

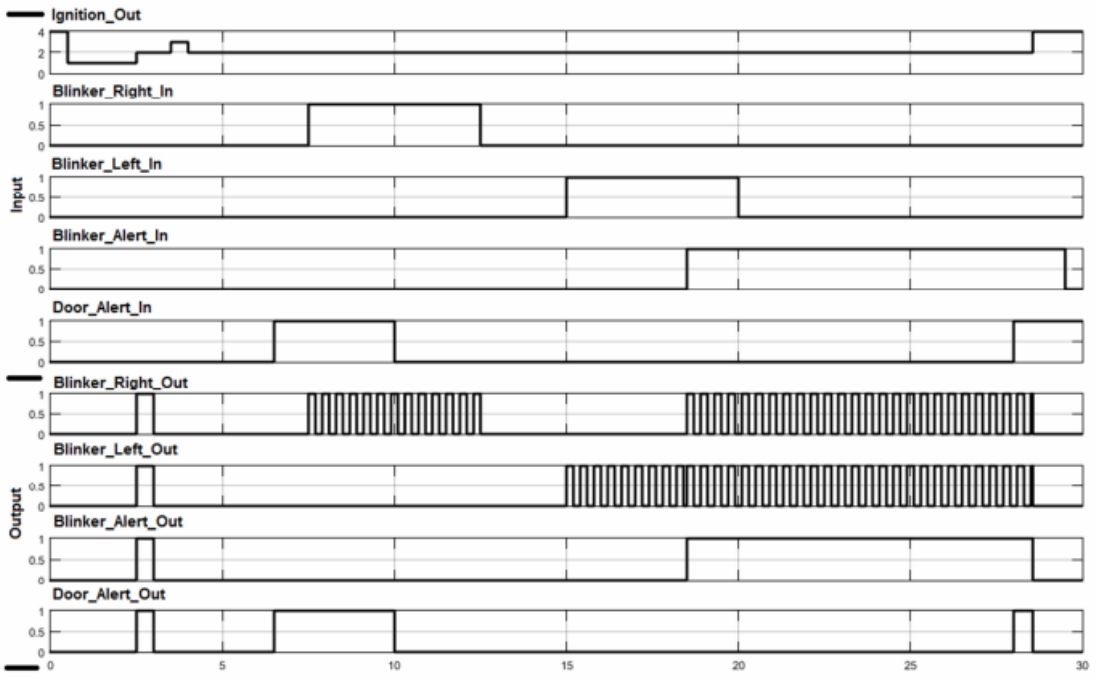


Fonte: Autor

Os gráficos temporais a seguir demonstram, no mesmo raciocínio, os sinais de entrada e saída das funções de alerta e piscas, alertas do uso do cinto de

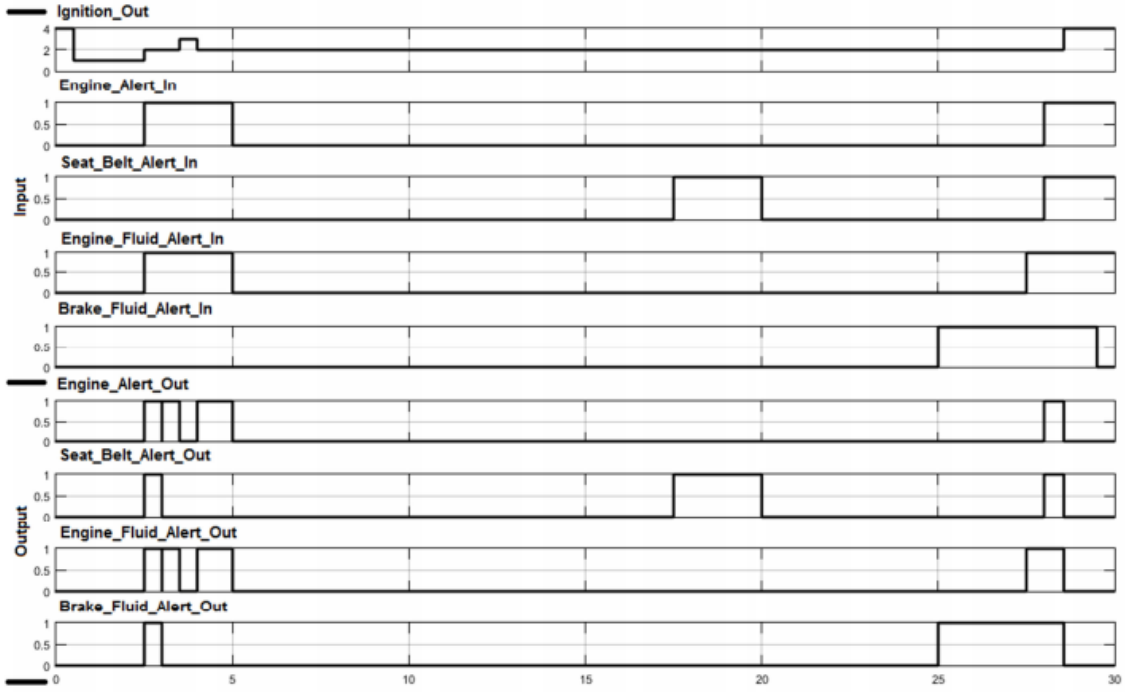
segurança temperatura e fluido do motor, bateria, velocidade e RPM, respectivamente e conforme a legenda.

Figura 32 - Sinais de entrada e saída das funções de Piscas e abertura da porta



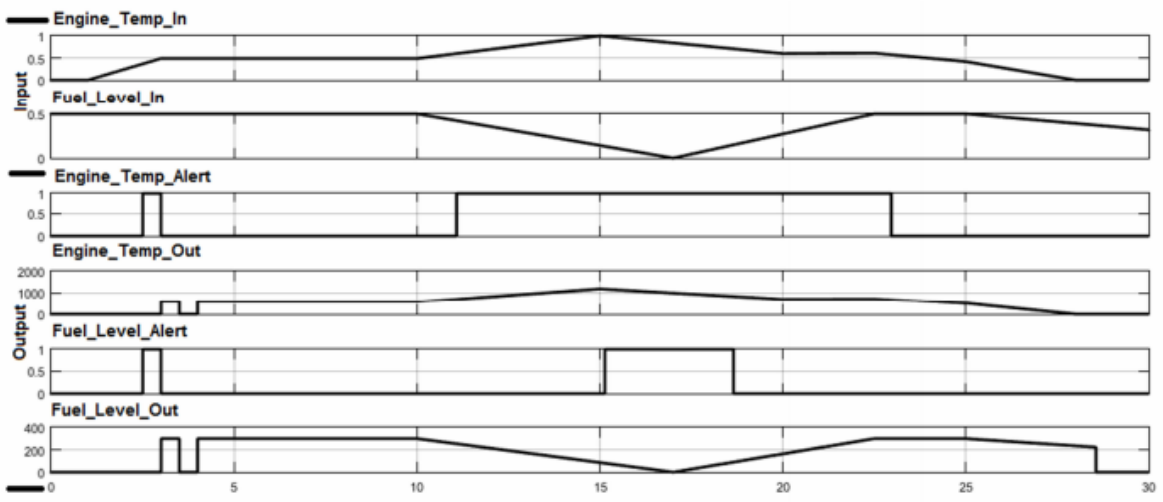
Fonte: Autor

Figura 33 - Sinais de entrada e saída das de alerta e fluido do motor e aviso do cinto de segurança



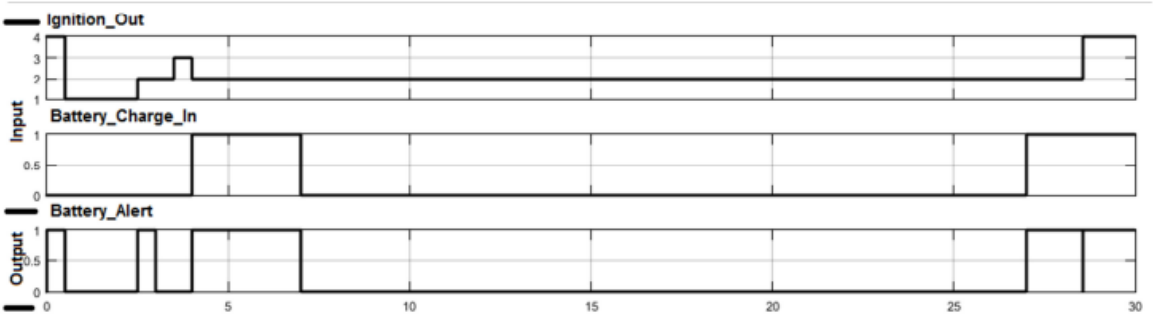
Fonte: Autor

Figura 34 - Sinais de entrada e saída das funções de temperatura do motor e nível de combustível



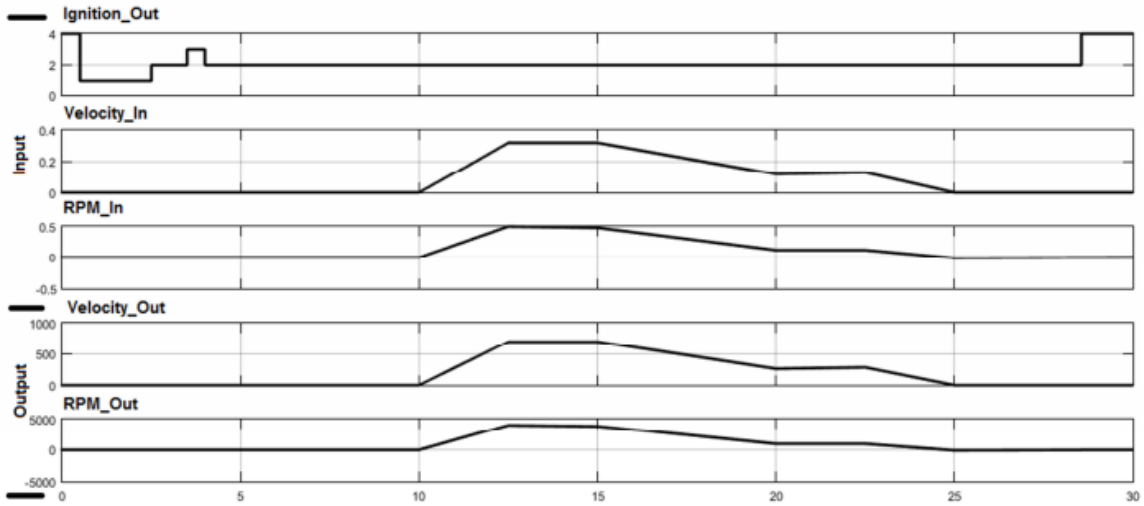
Fonte: Autor

Figura 35 - Sinais de entrada e saída das funções referentes a tensão da bateria



Fonte: Autor

Figura 36 - Sinais de entrada e saída das funções de velocidade e RPM



Fonte: Autor

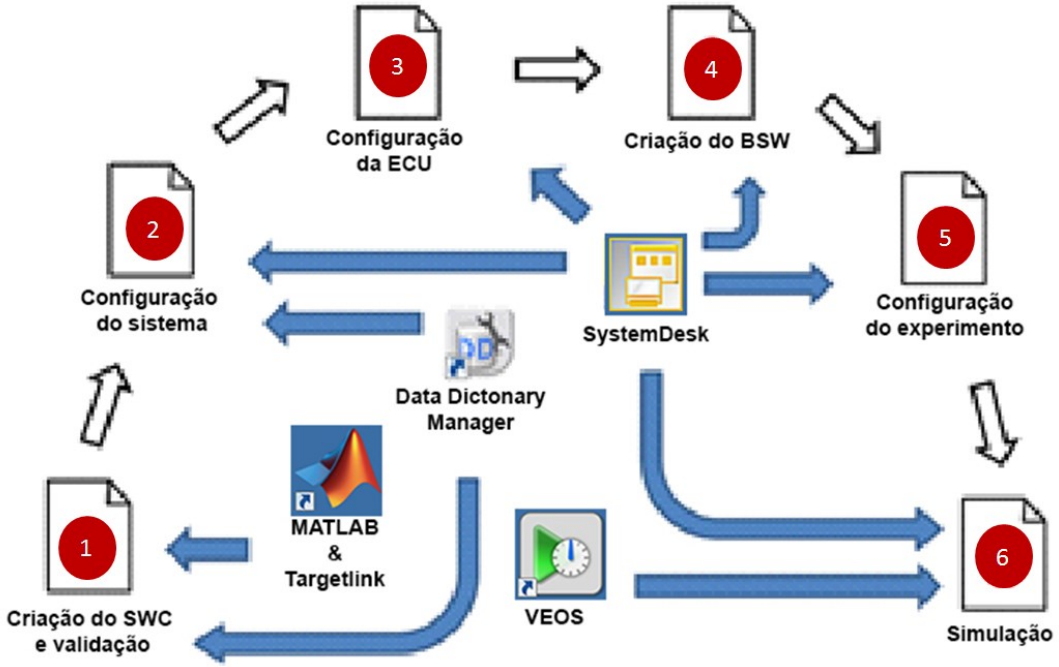
Através da análise dos sinais de saída e confrontando suas ativações com os pré-requisitos do projeto, é possível facilmente verificar que a função satisfaz as condições definidas.

6 MIGRAÇÃO DOS MODELOS PARA O PADRÃO AUTOSAR

A arquitetura de AUTOSAR pode ser feita utilizando-se diferentes níveis de programação, em conjunto ou isolado, para definir todos os parâmetros do código. Há os parâmetros mínimos que devem ser resolvidos para que a arquitetura seja validada como: SWC com seus respectivos códigos e comportamentos internos, portas, elementos de dados, interfaces de comunicação, escalas, tipos de dados e as conexões das SWC entre eles.

Em um fluxo de trabalho AUTOSAR, mostrado na Figura 37 uma variedade de ferramentas oferecidas por diferentes fabricantes podem ser encontrados para executar uma parte específica do processo. Estas soluções podem ser divididas em três classes de ferramentas: Ferramentas de modelagem comportamental ou BMT (Behavior Modeling Tools) e C IDE, ferramentas de design de nível de sistema e ferramentas de configuração de *Software* básico.

Figura 37 - Fluxo de trabalho para a validação virtual em AUTOSAR.



Fonte: Autor

- BMT e C IDEs - correspondem ao desenvolvimento de SWC, utilizando BMT, por exemplo: Simulink, TargetLink ou linguagem C manual. Cada SWC é composto por um arquivo de descrição XML AUTOSAR (.ARXML)

e seu respectivo arquivo .c. Isto corresponde à camada superior no nível AUTOSAR.

- Ferramentas de design de nível de sistema - responsável pelo desenvolvimento dos sistemas e subsistemas. Algumas ações podem ser executadas como ligações SWC e mapeamento ECU, topologia hardware, gestão de comunicação de barramento, mapeamento de elementos de dados, etc. Os resultados desta ferramenta são: Descrição do sistema de arquivos (.ARXML) e arquivos de configuração de ECU (.ARXML) para cada ECU no sistema. A ferramenta SystemDesk é um exemplo de "ferramenta de nível de sistema".
- Ferramentas de configuração de Software básico - estas ferramentas são responsáveis por configurar cada ECU individualmente e gerar o seus arquivos hex que serão gravados nos micro controladores da ECU.

Cada ferramenta de *Software* é responsável por um passo de todo o processo, que é bidirecional, de modo que podem ser feitas alterações de código ou a arquitetura independentemente da etapa do processo.

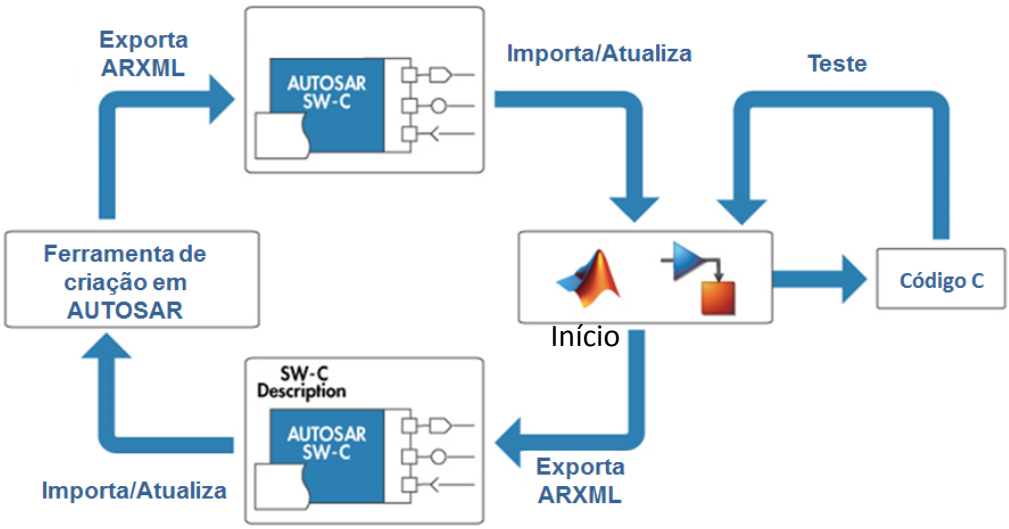
Para implementar o sistema, um conjunto de ferramentas foi utilizado para criar os SWC, a RTE e uma estrutura BSW que permitiu virtualizar uma ECU simples e testar o comportamento do aplicativo. As implementações do SWC foram feitas usando as ferramentas Simulink e Embedded Coder da MathWorks. Para construir a RTE e uma pequena parte da camada de BSW que é necessário para a virtualização da ECU, foi utilizado o SystemDesk e, finalmente, a ferramenta de virtualização VEOS para simular o ECU virtual, ambos fabricados pela dSPACE. A Figura 38 demonstra o fluxo do projeto a partir do modelo até a geração dos arquivos e código em AUTOSAR.

Com a utilização dos modelos em Simulink, a migração para AUTOSAR é facilitada. Os blocos de Simulink são nativamente compatíveis com o padrão e não necessitam de substituição por blocos específicos quando se utiliza um modelo anteriormente desenvolvido para outro propósito.

No ambiente Simulink, todas as aplicações foram construídas para abranger a função específica e necessária para o sistema completo. Os grupos foram divididos como explicado no capítulo anterior.

Cada grupo possui um SWC responsável por ler um valor de um transdutor específico e, em seguida, enviar para o outro componente que controla os mostradores do painel. Todos os SWC comunicam-se com o sua entrada e saída específicos através da infraestrutura da VFB (Virtual Funcional Bus), que será realizada pela RTE (Runtime Environment) e BSW (Software Basic).

Figura 38- Fluxo de trabalho utilizando ferramentas Mathworks® para criação dos arquivos do padrão AUTOSAR.



Fonte: MathWorks (2016) Editado pelo autor

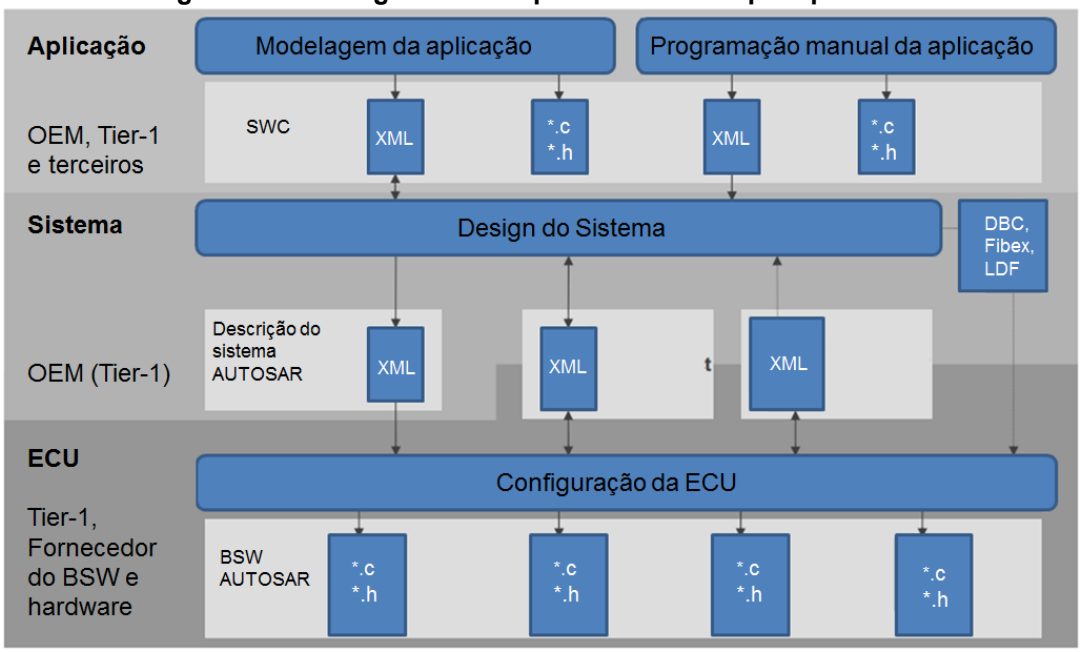
Seguindo este fluxo de trabalho a partir de um modelo são criados os arquivos que contém as configurações de interface em AUTOSAR, a estrutura de comunicação do SWC e a lógica do controlador do sistema em questão.

O arquivo mais importante do padrão é o ARXML, a partir dele a ferramenta escolhida para criação das interfaces em AUTOSAR cria todas as interfaces de entrada e saída (*sender/recievers*). Esses dados são primordiais para que a camada de comunicação RTI seja criada. Além disso, cria-se toda estrutura de comunicação que se faça necessária, seja interna ao SWC, como a externa à lógica de funcionamento do sistema em questão. A Figura 39 demonstra uma visão geral dos arquivos gerados e utilizados em cada fase da aplicação do padrão AUTOSAR.

A Figura 39 demonstra o esquema completo de aplicação do padrão, desde a criação dos SWC até a aplicação em uma ECU. Para a criação do SWC a partir de modelos de Simulink, vamos nos focar nos arquivos da faixa de aplicação,

“Modelagem da Aplicação”. Vemos que os arquivos gerados são o XML (no padrão AUTOSAR é chamado de ARXML) que possuem as características das interfaces, e os arquivos .c e .h que trazem a lógica do controlador desenvolvido.

Figura 39 - Visão geral dos arquivos utilizados pelo padrão AUTOSAR.



Fonte: Autor

No momento da migração do modelo através do Embedded Coder, estes arquivos são automaticamente gerados em uma pasta específica. Com estes arquivos faz-se o uso da ferramenta SystemDesk da dSpace para a criação do ambiente virtual do SWC. Este ambiente virtualiza a lógica do controlador e demonstra também em blocos a disposição das interfaces de entrada e saída distribuída em subsistemas.

Esta disposição separa as lógicas de funcionamento de maneira clara e facilita o entendimento, pois não apresenta ao desenvolvedor apenas um código sequencial. Através do SystemDesk é possível virtualizar o funcionamento de uma ECU com diversos SWC, basta importa-los dentro de um mesmo projeto.

O trabalho a ser feito no SystemDesk é focado em como compor os SWC migrados do Simulink. Cria-se uma composição (*Composition*) e define-se como as entradas e saídas irão se relacionar.

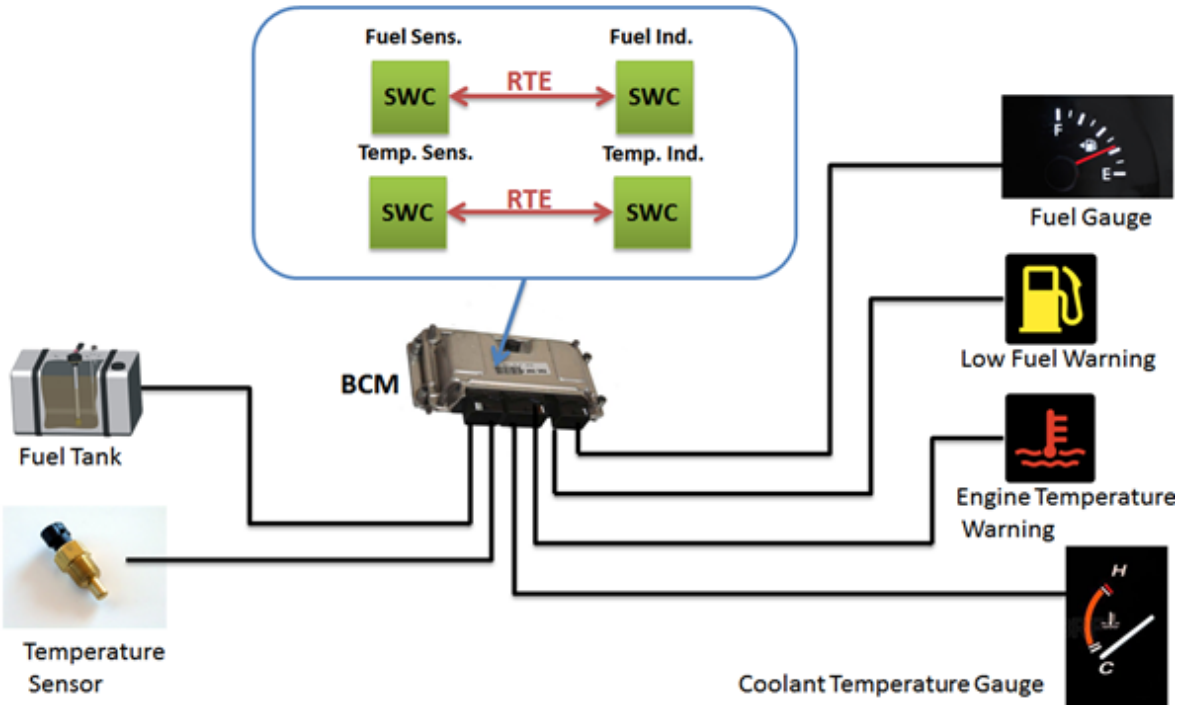
Há duas maneiras dos SWC se comunicarem na arquitetura AUTOSAR, uma é Intra-ECU e outra é chamada de comunicação Inter-ECU. Na arquitetura do sistema do veículo, as duas formas de implementação trabalham juntos,

compartilhando dados para as aplicações. A aplicação do painel foi implementada utilizando apenas a comunicação Intra-ECU por questões de disponibilidade de licença de ferramentas de software.

6.1 COMUNICAÇÃO INTRA ECU

Em uma comunicação Intra-ECU, todos os SWC que precisam se comunicar uns com os outros estão disponíveis sobre o mesmo processador (ECU). A troca de dados entre aplicações ocorre por meio de regiões dentro do chip (memória RAM). A implementação usando a arquitetura Intra-ECU foi realizada colocando todos SWC (Sensor de combustível., Indicador de Combustível, Sensor de Temperatura e Indicador de Temperatura) em uma única ECU, conectando todos os aplicativos via VFB (RTE + BSW). O sensor de temperatura, sensor de nível, medidores e luzes de advertência são conectados a uma ECU única. A Figura 40 mostra a aplicação Intra-ECU da função de temperatura do motor e nível de combustível. Todas as outras funções seguem a mesma lógica em sua estrutura.

Figura 40 - Cenário para uma aplicação Intra-ECU em AUTOSAR (Arquitetura independente)



Fonte: Autor

6.2 GERAÇÃO DOS COMPONENTES DE SOFTWARE

Para implementar a virtualização do sistema, alguns passos precisam ser executados para construir os elementos necessários que, juntos, podem propiciar a visualização do comportamento.

6.2.1 Criação dos arquivos para o SWC

O Simulink é responsável pela definição dos parâmetros dos componentes usando um compilador interno chamado Embedded Coder. Cada componente do modelo no Simulink é referenciado e relacionado com as especificações de variáveis, tipos de dados, elementos de dados e funcionalidades. Todo o sistema foi projetado em ambiente Matlab/Simulink.

Cada grupo desenvolvido será definido como um SWC, definindo as portas do SWC com a respectiva interface de comunicação que trás consigo as especificações de elementos de dados, escalas, tipos de dados e variáveis.

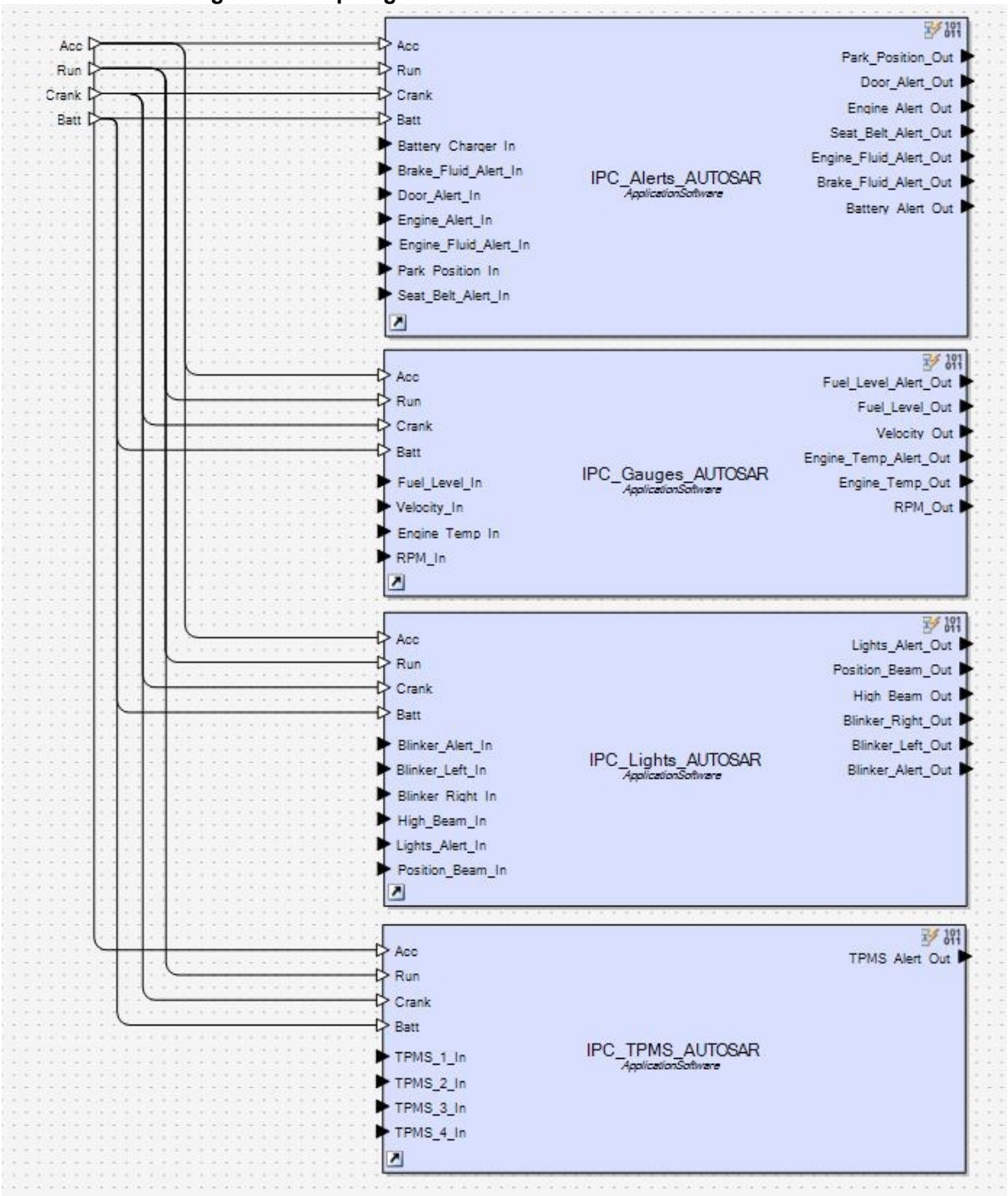
O Embedded Coder gera todos os arquivos necessários para a criação do SWC no SystemDesk. Após o SWC ser criado, cada aplicação pode ser testada individualmente para verificar, no início do processo, se o algoritmo desejado funciona adequadamente como esperado.

Depois de terminado o desenvolvimento de todos SWC, o próximo passo é a exportação destes parâmetros para o SystemDesk, que será o gerente das conexões entre os SWC e topologias de *hardware* e *Software*.

6.2.2 Design da Arquitetura do Sistema pelo SystemDesk

O SystemDesk é responsável por criar a estrutura do sistema, em outras palavras, o modelo do sistema, permitindo o funcionamento correto da ECU. Aqui, as principais definições serão: Arquitetura de *Software*, Topologia de Hardware, Comunicação pelo barramento e, finalmente, a simulação. Um projeto foi criado e todos os SWC desenvolvidos no passo anterior foram importados. A Figura 41 mostra os SWC importados e relacionados com a respectiva aplicação.

Figura 41 - Topologia dos SWC



Fonte: Autor

Após a criação do projeto e as inclusões dos SWC, o próximo passo é configurar a topologia de *hardware*. Para a implementação Intra-ECU, apenas uma única configuração de ECU precisa ser criada (topologia de *hardware*), seguindo

pela fase de definição do sistema, onde a colocação do SWC é realizada e todos os aplicativos serão locados na mesma ECU.

Para executar uma simulação virtual da ECU, é necessário ter uma pequena implementação da camada de BSW para gerenciar a ação da estrutura AUTOSAR (tarefas do SO, periféricos de comunicação, etc.). O SystemDesk é uma ferramenta de nível de sistema, mas pode oferecer uma pequena pilha de BSW que é suficiente para construir uma ECU virtual a ser utilizada na etapa de simulação.

6.2.3 Configuração do experimento

Esta etapa é a simulação da interação do sistema que inclui todas as partes do modelo, incluindo a ECU, o barramento de comunicação e outras partes. Para a experiência é necessário especificar um conjunto de variáveis de medição e de estímulo.

As variáveis de estímulo têm como opção empregar como sinais de entrada no formato de ondas, rampas e constantes. Os valores destes sinais são escolhidos para garantir a melhor cobertura para o teste do sistema. As variáveis de medições têm a opção de medir um elemento de dados de uma porta dos SWCs, o dado de uma abstração de I/O da V-ECU (ECU Virtual) ou uma variável de estímulo em sua origem.

A configuração da simulação é responsável por toda a configuração do ambiente de teste. Nesta configuração os parâmetros como StopTime (tempo de parada), BreakTime (tempo de pausa) e HistorySignal (histórico do sinal) devem ser definidos. Depois de todos esses parâmetros serem resolvidos é possível gerenciar a simulação, gerar os sinais que vão alimentar as entradas dos SWC, quantas vezes a simulação irá ser executada e analisar os sinais de saídas.

Depois das topologias e configurações serem definidas no SystemDesk, já é possível executar a simulação de um V-ECU usando as topologias de *Software* e *hardware*. Para executar a simulação, SystemDesk deve ser conectado com VEOS que é executado em segundo plano para executar a simulação.

Na fase de verificação, um conjunto de parâmetros no ambiente de teste precisa ser configurado para oferecer condições mais próximas possíveis de uma situação real. Como resultado, os sinais de saída podem ser demonstrados em

gráficos para análises de verificação de comportamento. O Quadro 8 lista o conjunto de sinais de entrada utilizados para verificação dos componentes de software.

Quadro 8 - Sinais de entrada para verificação da função em AUTOSAR.

Descrição	Formato do Sinal
Crank	Sinal constante em 0
Run	Sinal degrau de 0 para 1 em 0,5s
Acc	Sinal degrau de 0 para 1 em 0,3s
Bateria	Sinal rampa de 8.5 para 12 em 10s
TMPS 1	Sinal degrau de 1 para 0 em 6s
TMPS 2	Sinal constante em 1
TMPS 3	Sinal constante em 1
TMPS 4	Sinal degrau de 0 para 1 em 2s
Alerta (ambos os piscas)	Sinal degrau de 0 para 1 em 8s
Pisca Esquerdo	Sinal degrau de 1 para 0 em 3s
Pisca Direito	Sinal degrau de 0 para 1 em 7s
Farol Alto	Sinal degrau de 0 para 1 em 8,5s
Farol de posição	Sinal degrau de 1 para 0 em 5s
Combustível	Sinal rampa de 1 para 0 em 8.5s
Velocidade e RPM	Sinal rampa de 0 para 1 em 10s
Temperatura do Motor	Sinal rampa de 0 para 1 em 6s
Alerta de fluido de freio	Sinal degrau de 0 para 1 em 7.5s
Alerta de fluido do motor	Sinal degrau de 0 para 1 em 7.5s
Alerta de Falha no Motor	Sinal degrau de 0 para 1 em 7.5s
Alerta de porta aberta	Sinal degrau de 1 para 0 em 2.5s
Alerta do cinto de segurança	Sinal degrau de 1 para 0 em 2,5s

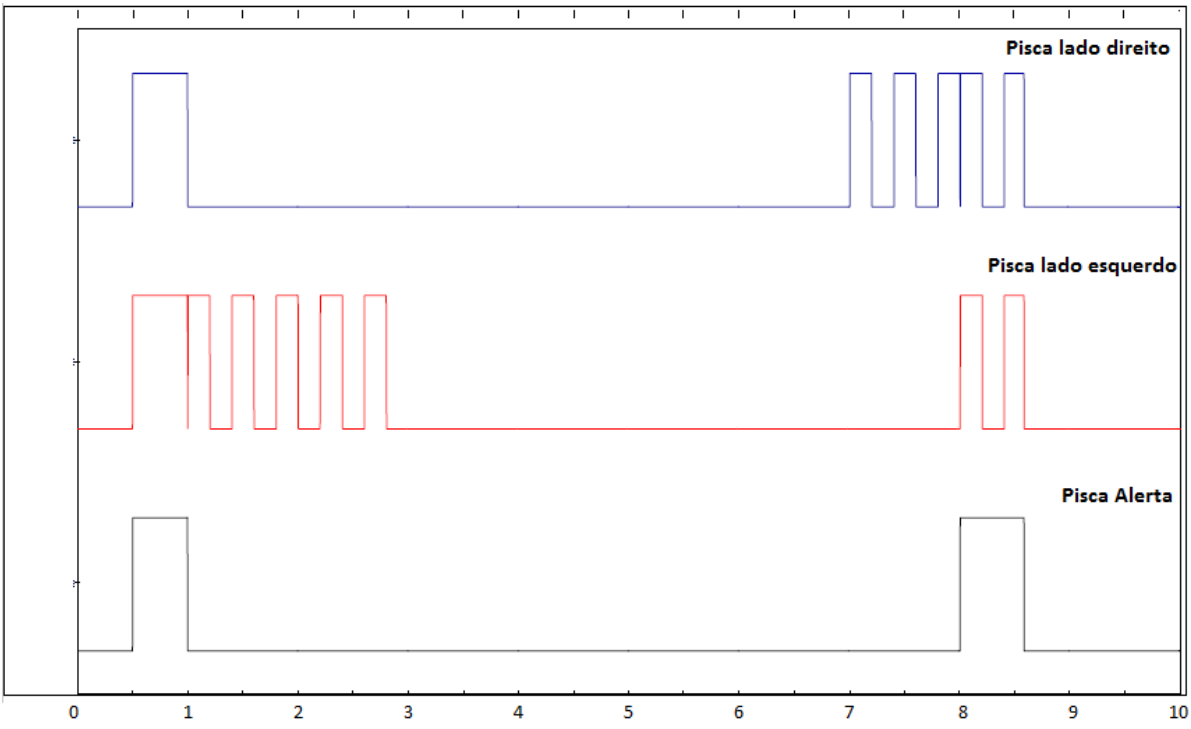
Fonte: Autor

Após a criação destes perfis de sinal de entrada, foi feita a simulação do sistema completo. A partir da observação dos sinais de saída pode-se verificar se a lógica está correta após a migração e que todas as interfaces de entrada e saída foram corretamente criadas.

Observando os sinais de saída na Figura 42 podemos notar que o sistema iniciará seu funcionamento após 0,5 segundos, instante que a ignição alcançar a posição “Acc”. Portanto nenhum sinal antes deste tempo pode apresentar variação. De acordo com a lógica criada no Simulink, após esta ativação, os sinais booleanos deverão todos ser ativados por 0,5 segundos para um teórico teste de lâmpadas. Observando o final da simulação, todos os sinais devem ser desabilitados em aproximadamente 8,7 segundos devido ao comportamento do sinal que simula a tensão da bateria. Este comportamento foi observado em todos os sinais, como demonstrarão os gráficos a seguir.

O primeiro gráfico, mostrado na Figura 42 demonstra o funcionamento das luzes indicadoras de direção (piscas) e o alerta.

Figura 42 - Sinais de saída referente as funções pisca esquerdo e direito além do alerta



Fonte: Autor

Vemos que conforme o esperado, após 0,5 segundos o sistema é ativado, faz a verificação das lâmpadas e aí inicia o funcionamento normal. O pisca direito é ativado em 7 segundos pois o sinal de entrada referente a sua ativação é um degrau iniciado em 7 segundos até o instante 10 segundos. Devido a condição da bateria

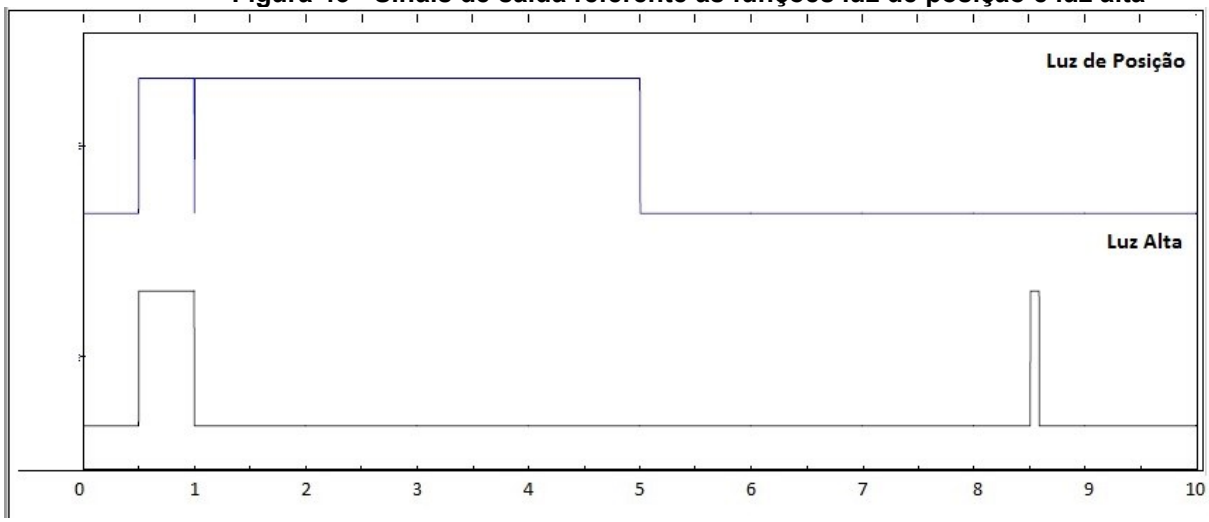
estar fora do range aceito pelo requisito de projeto no instante aproximado de 8,7 segundos, notamos que o pisca direito é desativado neste instante.

Outra observação interessante é que o seu funcionamento não é atrapalhado pelo acionamento do sinal de alerta (degrau positivo em 8 segundos). Porém observamos que a lógica do alerta funciona conforme o definido pois o sinal de pisca alerta esquerdo é ativado logo após a verificação inicial (pois seu sinal de ativação é uma rampa que inicia-se em 0 segundos) desativado em 3 segundos, quando o sinal passa a ser 0, porém reativado em 8 segundos, junto com o sinal de alerta. Novamente a desativação é notada em aproximadamente 8,7 segundos, devido à bateria.

O sinal de alerta é de fácil observação pois é ativado no momento da verificação das lâmpadas e ativado de 8 a 8,7 segundos.

O próximo gráfico (Figura 43) mostra o comportamento dos sinais de saída referente a luz de posição e luz alta.

Figura 43 - Sinais de saída referente as funções luz de posição e luz alta

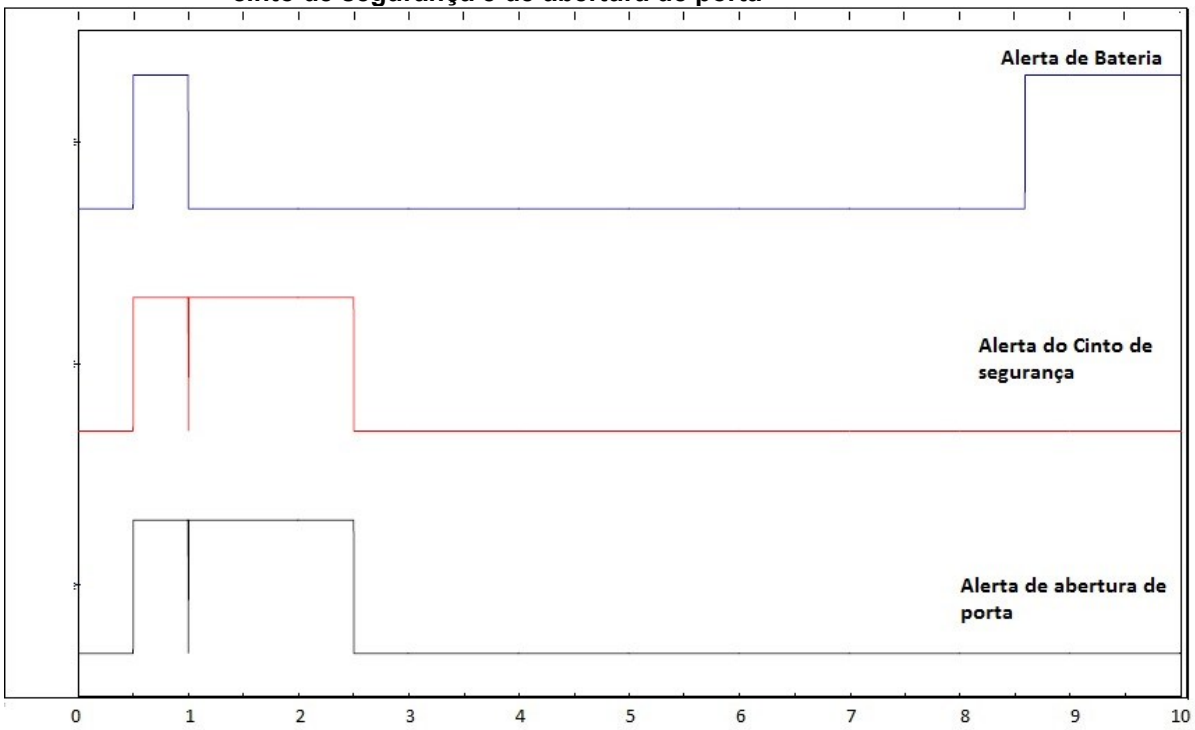


Fonte: Autor

No gráfico é possível observar que a luz de posição, após a verificação inicial, é acionada até o instante 5 segundos, pois ele é acionado por um sinal degrau que passa de 1 para 0 após 5 segundos. A luz alta é acionada por um sinal degrau que passa de 0 para 1 em 8,5 segundos, portanto é acionado apenas na verificação inicial e no instante de 8,5 segundos até 8,7 segundos.

A Figura 44 mostra a função dos alertas de bateria, cinto de segurança e abertura de porta. O sinal de bateria, conforme explicado em lógicas anteriores deve ser acionado no instante aproximado de 8,7 segundos. O sinal do cinto de segurança é acionado por um sinal degrau que passa de 1 para 0 em 2,5 segundos, conforme verificado pelo sinal de saída. Por último vemos que a função do alerta da abertura da porta tem o funcionamento de acordo com a sua entrada, um sinal degrau que passa de 1 para 0 em 2,5 segundos.

Figura 44 - Sinais de saída referente as funções dos alertas de bateria, cinto de segurança e de abertura de porta

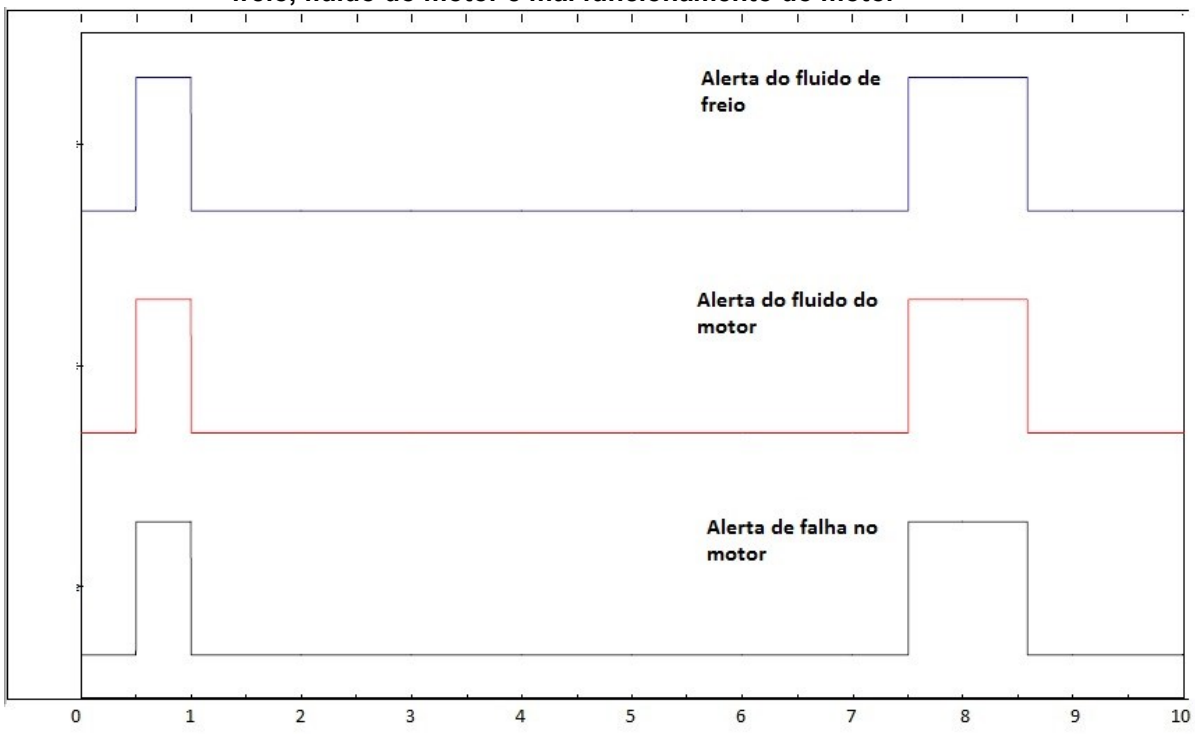


Fonte: Autor

Os sinais de alerta de falta de fluido de freio, falta de fluido do motor e mal funcionamento do motor são acionados por sinais idênticos. Os três devem ser acionados por um sinal degrau que passa de 0 para 1 em 7,5 segundos. A Figura 45 verifica esse requisito.

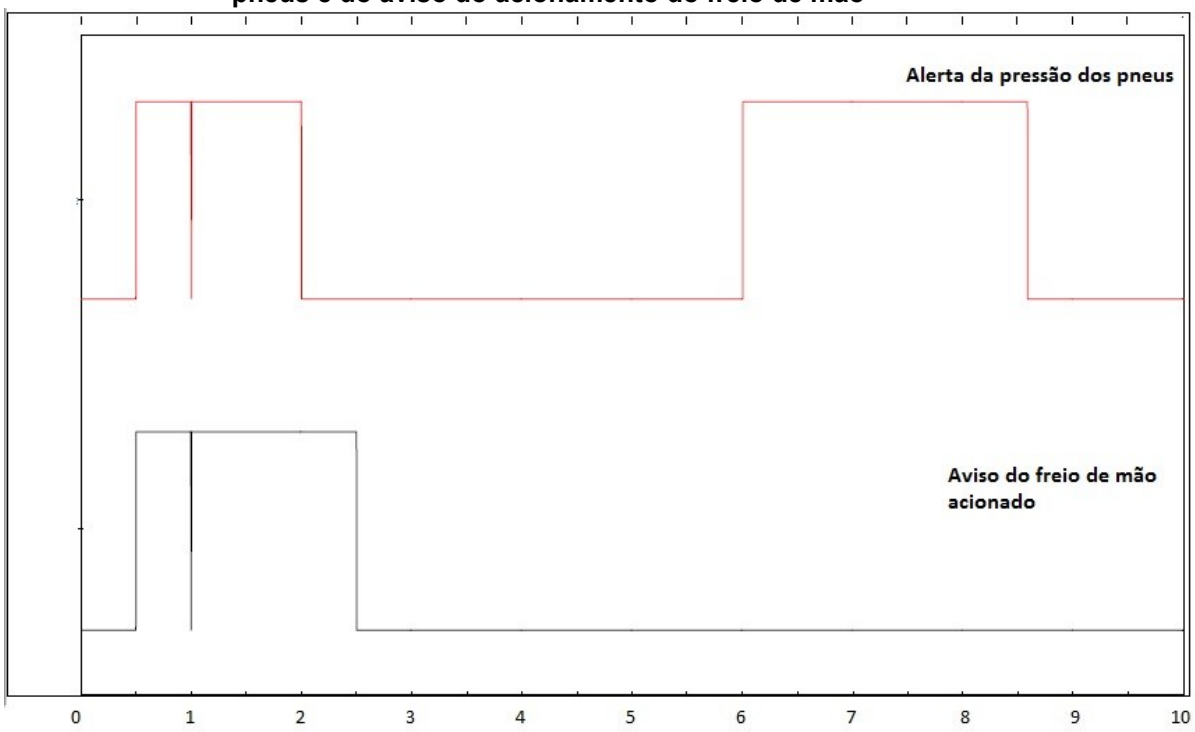
A Figura 46 demonstra os sinais de saída referente a lógica de funcionamento do alerta da pressão baixa em algum dos pneus e a do aviso que o freio de mão está acionado.

Figura 45 - Sinais de saída referente as funções dos alertas de fluido de freio, fluido do motor e mal funcionamento do motor



Fonte: Autor

Figura 46 - Sinais de saída referente as funções do alerta da pressão dos pneus e do aviso do acionamento do freio de mão



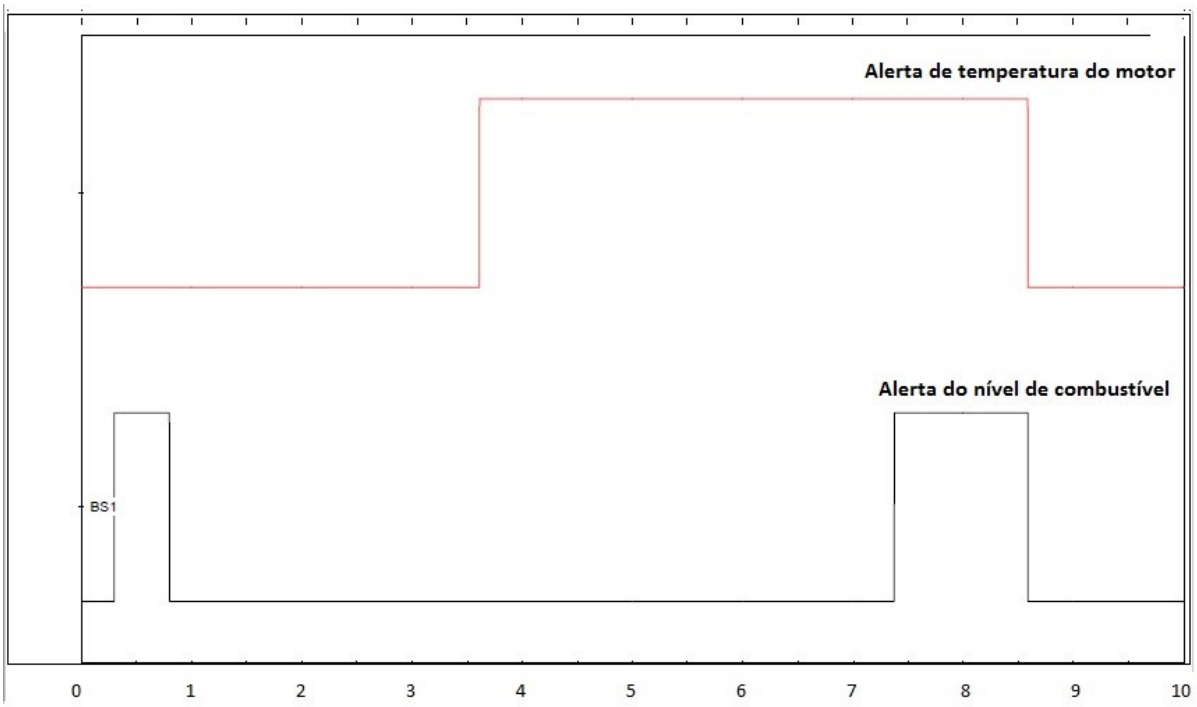
Fonte: Autor

O alerta de pressão baixa nos pneus é ativado em duas situações. Inicialmente pelo pneu 1 no por um sinal degrau de 1 para 0 em 6 segundos e novamente pelo pneu 4 por um sinal degrau de 0 para 1 em 2 segundos. OS dois instantes são verificados na Figura 46. O sinal de aviso do freio de mão é regido por um sinal degrau que passa de 1 para 0 em 2,5 segundos.

A Figura 47 demonstra a lógica de funcionamento dos sinais de saída das funções que alertam que a temperatura do motor foi excedida e que o nível mínimo de combustível foi atingido.

O sinal da temperatura do motor deve ser acionado por um sinal rampa que alcança o valor 1 em 6 segundos, portanto ultrapassando o limite em aproximadamente 3,5 segundos.

Figura 47 - Sinais de saída referente as funções de alerta da temperatura do motor e do nível de combustível

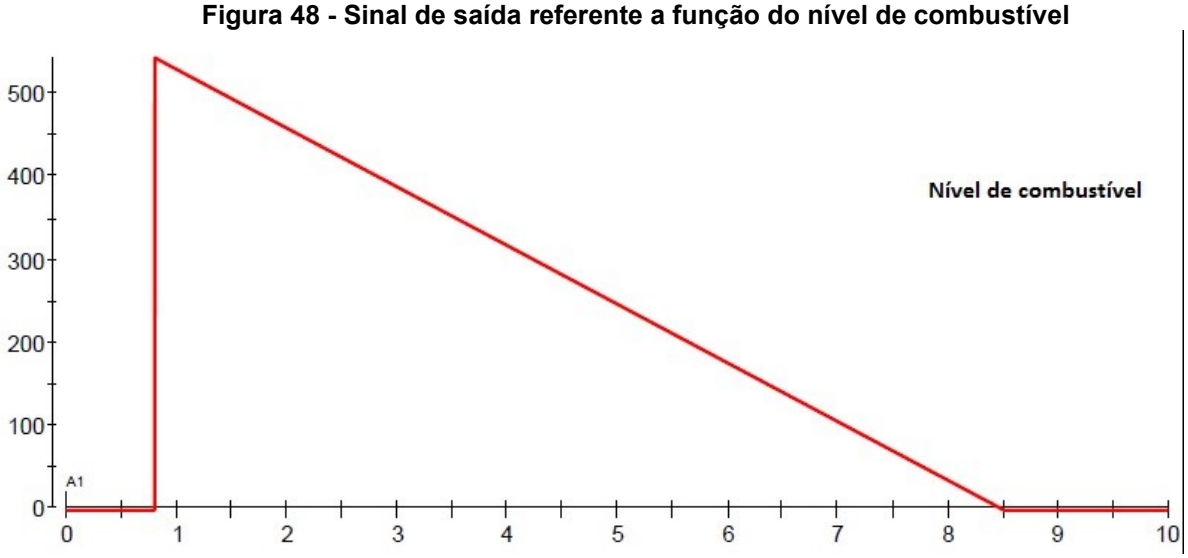


Fonte: Autor

O sinal do alerta do nível de combustível é acionado por um sinal rampa que inicia em 1 e alcança o valor 0 em 8,5 segundos. Portanto o sinal deve ser acionado em aproximadamente 7,4 segundos, conforme verificado na Figura 47.

A Figura 45 demonstra o sinal de saída que controle o ponteiro do nível de combustível. Conforme citado para o gráfico anterior, o comportamento é de uma rampa inicia em 1 e alcança o valor 0 em 8,5 segundos.

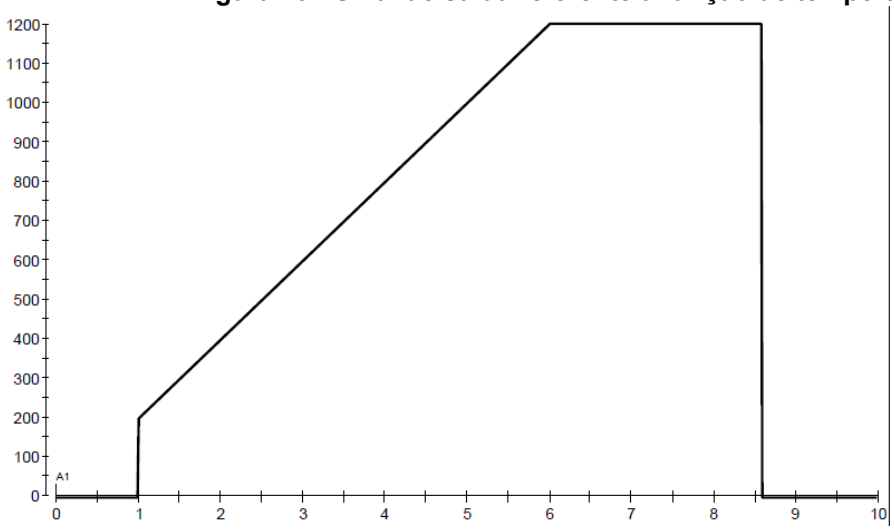
Figura 48 - Sinal de saída referente a função do nível de combustível



Fonte: Autor

O sinal de saída referente a lógica da temperatura do motor, que serve controla o ponteiro de temperatura do motor é demonstrado na Figura 49.

Figura 49 - Sinal de saída referente a função de temperatura do motor



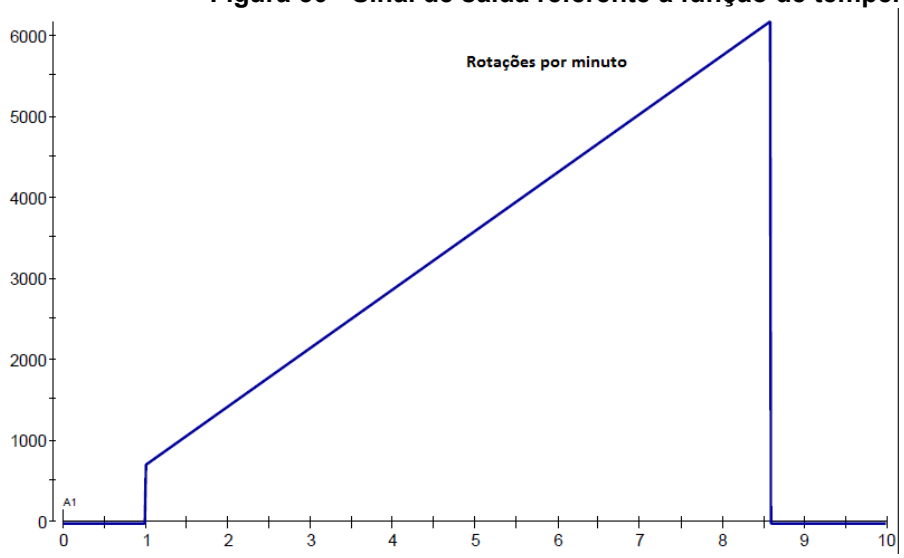
Fonte: Autor

Este sinal de saída é excitado por um sinal de entrada rampa de 0 para 1 em 6 segundos. Por isso nota-se que o valor máximo é alcançado no instante 6

segundos e permanece no máximo até o instante 8,7 segundos, quando a bateria desabilita o sistema.

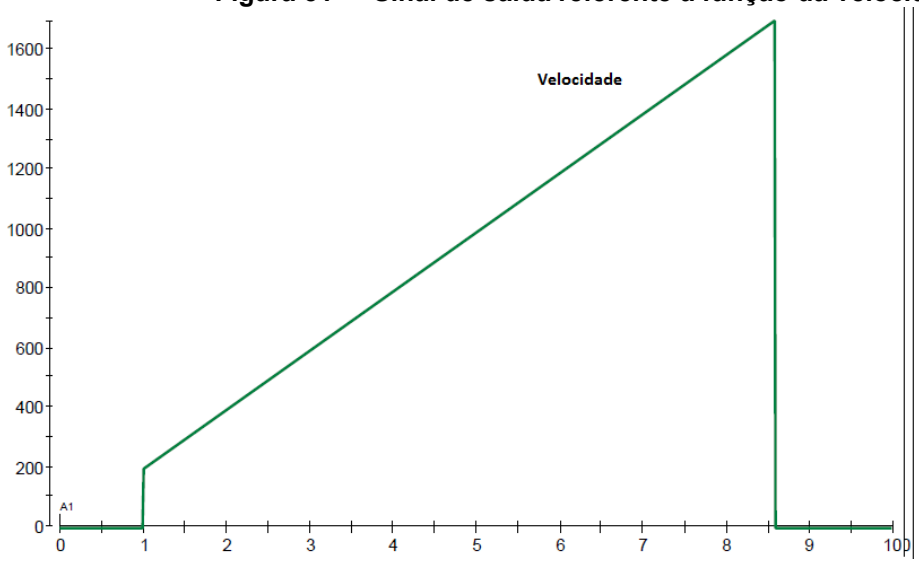
A Figura 50 demonstra o sinal de saída referente ao aumento da rotação por minuto da função que controla o ponteiro de rotação e é excitado por um sinal rampa que varia de 0 até 1 no instante 10 segundos, o mesmo sinal que excita o sinal de velocidade, demonstrado na Figura 51.

Figura 50 - Sinal de saída referente a função de temperatura do motor



Fonte: Autor

Figura 51 - - Sinal de saída referente a função da velocidade



Fonte: Autor

Facilmente observa-se que os sinais responderam corretamente aos requisitos pois crescem até o instante 8,7 segundos, momento que a bateria deixa de permitir o funcionamento do sistema.

Observando que todas as funções foram testadas e validadas, pode-se dizer que a função foi corretamente migrada como software dentro do Padrão AUTOSAR.

7 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

Através do desenvolvimento deste trabalho foi possível observar um fluxo de trabalho possível para desenvolvimento de software no padrão AUTOSAR.

O padrão AUTOSAR vem ganhando espaço no mercado automotivo mundial e será, futuramente, o padrão de todos os sistemas automotivos. Para isso é importante que haja trabalhos nacionais que abordem o tema e principalmente demonstrem a possibilidade de se criar tecnologia de ponta em universidades brasileiras.

Durante o transcorrer deste trabalho, demonstrou-se importância da aplicação da metodologia de desenvolvimento baseada no modelo desenvolvido pela ETAS (Bosch) conhecido como X-in-the-Loop (xIL), baseado no conceito de desenvolvimento baseado em modelos (MBD) nas funções de software automotivo.

Pode-se observar que este modo de verificar as funções durante sua criação auxilia significativamente a produtividade dos desenvolvimentos de aplicações automotivas, otimizando o tempo gasto com a modelagem e validação das funções.

A verificação através de gráficos permite que o desenvolvimento das lógicas seja mais rápido e fácil, permitindo constante simulação do sistema sem impactar o tempo de desenvolvimento.

A depuração antecipada dos possíveis problemas de lógica demonstrou-se facilitada dentro deste fluxo de trabalho, por permitir simulação constante durante qualquer momento do projeto, devido a facilidade das ferramentas.

Provou-se ser possível reutilizar projetos anteriores, diminuindo o tempo de finalização além de permitir melhoria contínua de funções já finalizadas.

Essa pesquisa pode ser expandida com a expansão da função do painel de instrumentos levando em conta todas as funções encontradas em um painel comercial. Além disso, com as ferramentas certas, é possível testar a função em AUTOSAR em um hardware que seja compatível com o padrão.

REFERÊNCIAS

ANFAVEA. **Anuário da Indústria Automobilística Brasileira**. São Paulo, 2015.

AUTOSAR GbR. **AUTOSAR Basics**. Disponível em: <http://www.autosar.org/index.php?p=1&up=1&uup=0>. Acesso em: 10 out. 2016.

AUTOSAR GbR. **AUTOSAR Technical Overview**. Disponível em: <http://www.autosar.org/index.php?p=1&up=2&uup=0>. Acesso em: 10 out. 2016.

AUTOSAR GbR**. AUTOSAR - The Worldwide Automotive Standard for E/E Systems**. Disponível em: < https://www.autosar.org/fileadmin/files/presentations/AUTOSAR_Brochure_EN.pdf>. Acesso em 10 out. 2016.

BARBIERI, G.; FANTUZZI C; BORSARI R. **A Model-Based Design Methodology for the Development of Mechatronics Systems. Elsevier**, v.24, p.833-843, 2014. Disponível em: < http://dx.doi.org/10.1016/j.mechatronics.2013.12.004>.

BROY, M. Challenges in Automotive Software Engineering. ICSE'06 – **Proceedings of the 28th International Conference on Software Engineering**, New York, p.33-42, 2006.

CHARETTE, R. N. This Car Runs on Code. **IEEE Spectrum**, New York, fev. 2009.

CTE Embedded Systems. **Embedded Systems Guide**. Disponível em:< http://www.embedded-systems-portal.com/CTB/AUTOSAR_ECU_Abstraction_Layer,10022.html>. Acesso em: 10 out. 2016.

dSPACE. **MicroAutoBox Overview of Board Revisions Release 2014B** Disponível em: <https://www.dspace.com/files/pdf1/MicroAutoBoxOverviewofBoardRevisions_Release2014B.pdf>. Acesso em: 10 out. 2016.

dSPACE. **System Desk Tutorial.** Disponível em: http://www.dspace.com/ww/en/pub/home/products/sw/system_architecture_software/systemdesk.cfm>. Acesso em: 10 out. 2016.

Electromagnetic Compatibility for Electric Vehicles. **CAN bus: Controller Area Network.** 8 set 2015. Disponível em: <http://www.flexautomotive.net/EMCFLEXBLOG/post/2015/09/08/can-bus-for-controller-area-network>. Acesso em: 10 out. 2016.

EHSANI, M.; GAO, Y.; EMADI, A. **Modem Electric, Hybrid Electric, and Fuel Cell Vehicles:** Fundamentals, Theory and Design.2.ed. Boca Raton: FL, CRC Press, 2009.

HAMMARSTRÖM, J.; NILSSON, J. **A Comparision of Three Code Generators for Models Created in Simulink.** 2006. Dissertação (Mestrado) – Chalmers University of Technology, Department of Computer Science and Engineering. Göteborg, 2006.

HEBIG, R. **Methodology and Templates in AUTOSAR.** Technical Report – Hasso-Plattner – Institut für Softwaresystemtechnik, 2009.

HELLINGRATH, B. **Future Automotive Industry Structure 2015.** Mercer Management Consulting und Fraunhofer Gesellschaft, Stuttgart. 2004.

HEINECKE, H.; et al. **AUTOSAR:** current results and preparations for exploitation. In: 7th EUROFORUM Conference Software in the Vehicle.2006, Stuttgart.

IWAGAYA T.; YAMAGUCHI T. **Speed improvements for xIL Simulation based on Symbolic-Algebraic method,** SICE Annual Conference 2013, p 1338-1343, Set 14-17, 2013, Nagoya University, Nagoya, Japan

KELEMENOVÁ, T.; KELEMEN, M.; MIKOVÁ, L.; PRADA, E.; LIPTÁK, T.; MENDA, F.; MAXIM, V. **Model Based Design and HIL Simulations.** American Journal of Mechanical Engineering, v1, 2013, p.276-281.

LEE, O. **Part 5: AUTOSAR CAN.** 2014. Disponível em: <http://www.autoelectronics.co.kr/article/articleView.asp?idx=1315>. Acesso em: 04 out. 2016.

LENNON, T. **Model-Based Design for Mechatronics Systems**. 2007. Disponível em: <http://machinedesign.com/archive/model-based-design-mechatronics-systems>. Acesso em: 10 oct. 2016.

LEPPLA, G. **Mapping Requirements to AUTOSAR Software Components.** 2008. Dissertação (Mestrado) – Waterford Institute of Technology, Ireland, 2008.

LUO, F.; HUANG, Z. **Embedded C Code Generation and Embedded Target Development Based on RTW-EC.** 3rd IEEE International Conference on Computer Science and Information Technology (ICCSIT 2010). 07, 2010, Chengdu, China. p. 532-536.

NEME, J. H. Z. **Aplicação do Método de Desenvolvimento Baseado em Modelos para Função de Software Automotivo:** Sistema de Iluminação Externa. Trabalho de Conclusão de Curso. Engenharia Eletrônica –EE. Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2014.

NISHIKAWA, K.; KAJIO, K.; LANGE, K.; SCHARNHORST, T.; KUNKEL, B. **Achievements and Exploitation of the AUTOSAR Development Partnership.** CTEA, 2006.

OICA. **Production Statistics 2015.** Disponível em: <http://www.oica.net/category/production-statistics/>. Acesso em: 10 out. 2016.

PLOSS, R.; MUELLER, A.; LETEINTURIER, P. **Solving Automotive Challenges with Electronics.** In: 2008 International Symposium On Technology, Systems and Applications (VLSI-TSA). **Anais of Institute of Electrical and Electronics (IEEE)**, p.327-345, 2008. Disponível em: http://dx.doi.org/10.1109/vtsa.2008.4530772.

PRETSCHNER, A; BROY, M; KRUGER, I. H.; STAUNER, T. **Engineering Automotive Software.** Proceedings of the IEEE, New York, v.5, p.356-373, fev. 2007.

RAMACHANDRAN, M.; CARVALHO, R. A. **Handbook of Research on Software Engineering and Productivity Technologies:** Implications of Globalization. 2009. p. 556.

RENESAS. **AUTOSAR Layered Architecture.** Disponível em: <https://www.renesas.com/en-us/solutions/automotive/technology/autosar/autosar-layerdarch.html>. Acesso em: 10 out. 2016.

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

SCHEID, O. **AUTOSAR Compendium:** Part 1 – Application and RTE. 1 ed. Bruchsal: Createspace Independent Publishing Platform, 2015.

SCHIRRMEISTER, F. **Automotive System & Software Development Challenges – Part 1.** Cadence Design Systems. 05 nov. 2013. Disponível em :< http://www.edn.com/design/systems-design/4423874/1/Automotive-System---Software-Development-Challenges---Part-1>. Acesso em: 10 out. 2016.

SCHNEIDER, A.; SOUZA, F. **Sistemas Embarcados:** Hardware e Firmware na Prática. 2. ed. São Paulo, Editora Érica, 2014.

SHATAT, Tariq S.; ABDULLAH, Bassem A.; SALEM, A. **System C based Simulation of AUTOSAR software components.** In: 10th International Conference On Computer Engineering & Systems (ICCES).2015, Cairo. Anais Institute of Electrical and Electronics Engineers (IEEE), 2015. p. 1-10. Disponível em: http://dx.doi.org/10.1109/icces.2015.7393028.

STELLA, G. N. D. **Aplicando a Metodologia de Desenvolvimento Baseado em Modelos para Funções de Software Automotivo.** 2015.123f. Dissertação (Mestrado) – Programa de Pós-Graduação em Engenharia Elétrica. Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2015.

TIBBA, Ghizlane et al. Testing automotive embedded systems under X-in-the-loop setups. **Proceedings Of The 35th International Conference On Computer-aided Design - Iccad '16,** [s.l.], nov. 2016. ACM Press. http://dx.doi.org/10.1145/2966986.2980076.

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

WARSCHOFSKY, R. **AUTOSAR Software Architecture.** Technical Report – Hasso-Plattner - Institut für Softwaresystemtechnik, 2009.

WEHRMEISTER, Marco A.; PACKER, Joao G.; CERON, Luis M.. Framework to Simulate the Behavior of Embedded Real-Time Systems Specified in UML Models. **2011 Brazilian Symposium On Computing System Engineering,** Florianópolis, p.1-7, nov. 2011. IEEE.

WEHRMEISTER, Marco A.; CERON, Luis M.; SILVA, Johnny L. da. Early Verification of Embedded Systems: Testing Automation for UML Models. **2012 Brazilian Symposium On Computing System Engineering,** Natal, Rn, n. 5, p.119-124, nov. 2012. IEEE.