

**UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ**

**GABRIEL RAUL GANDOLFI**

**DESENVOLVIMENTO DE UNIDADE DE CONTROLE VEICULAR PARA  
VEÍCULOS ELÉTRICOS**

**CURITIBA  
2024**

**GABRIEL RAUL GANDOLFI**

**DESENVOLVIMENTO DE UNIDADE DE CONTROLE VEICULAR PARA  
VEÍCULOS ELÉTRICOS**

**DEVELOPMENT OF VEHICLE CONTROL UNIT FOR ELECTRICAL VEHICLES**

Trabalho de conclusão de curso de graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia Elétrica do curso de Engenharia Elétrica da Universidade Tecnológica Federal do Paraná (UTFPR).

Orientador (a): Delvanei Gomes Bandeira Jr.  
Coorientador (a): Carlos Henrique Mariano

**CURITIBA**

**2024**



Esta licença permite remixe, adaptação e criação a partir do trabalho, para fins não comerciais, desde que sejam atribuídos créditos ao(s) autor(es). Conteúdos elaborados por terceiros, citados e referenciados nesta obra não são cobertos pela licença.

**GABRIEL RAUL GANDOLFI**

**DESENVOLVIMENTO DE UNIDADE DE CONTROLE VEICULAR PARA  
VEÍCULOS ELÉTRICOS**

Trabalho de conclusão de curso de graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia Elétrica do curso de Engenharia Elétrica da Universidade Tecnológica Federal do Paraná (UTFPR).

Data de aprovação: 28/agosto/2024

---

Delvanei Gomes Bandeira Junior  
Doutorado  
Universidade Tecnológica Federal do Paraná

---

Adriano Ruseler  
Doutorado  
Universidade Tecnológica Federal do Paraná

---

Marcio Aparecido Batista  
Doutorado  
Universidade Tecnológica Federal do Paraná

**CURITIBA**

**2024**

Dedico este trabalho aos meus amigos e familiares,  
que compreenderam minha ausência e  
sempre me apoiaram, mesmo quando  
o trabalho me afastou de nossa convivência.

## **AGRADECIMENTOS**

Gostaria de expressar minha gratidão ao meu orientador, Delvanei Gomes Bandeira Junior, pela sua disponibilidade e pelo direcionamento assertivo ao longo deste trabalho. Sua orientação foi fundamental para que eu pudesse conduzir esta pesquisa com confiança e clareza.

Agradeço também à equipe Formula UTFPR, pela valiosa oportunidade de aprendizado que me proporcionaram. O conhecimento adquirido ao longo desta experiência foi essencial para o desenvolvimento deste trabalho.

Por fim, meu agradecimento especial à equipe UTForce e-Racing, que gentilmente disponibilizou seu projeto como base para minha pesquisa. Sua colaboração foi crucial para a concretização deste projeto.

## RESUMO

Este trabalho apresenta o desenvolvimento de uma Unidade de Controle Veicular (ECU) para veículos elétricos em competições da Fórmula SAE, focando no gerenciamento de torque e comunicação com o inversor de frequência para melhorar a dirigibilidade e segurança. A metodologia segue o modelo V, cobrindo desde o levantamento de requisitos (como leitura de sinais de acelerador, controle de tração, monitoramento de temperatura e comunicação via CAN) até o desenvolvimento e testes. O firmware foi desenvolvido em Simulink, permitindo simulações eficientes, enquanto o hardware, que tem como componente principal o ESP-32, foi projetado no Altium Designer e fabricado industrialmente. A ECU atende aos requisitos funcionais, mas a validação da rede CAN ainda precisa ser aprimorada, sendo o software adaptável a diferentes veículos da competição, podendo servir como base de desenvolvimento para diferentes equipes.

**Palavras-chave:** ECU; unidade de controle; veículo elétrico; Formula SAE; ESP-32; Simulink; controle de torque.

## ABSTRACT

This work presents the development of a Vehicle Control Unit (ECU) for electric vehicles in Formula SAE competitions, focusing on torque management and communication with the frequency inverter to improve drivability and safety. The methodology follows the V-model, covering everything from requirements gathering (such as throttle signal reading, traction control, temperature monitoring, and CAN communication) to development and testing. The firmware was developed in Simulink, allowing efficient simulations, while the hardware, featuring the ESP-32 as its main component, was designed in Altium Designer and industrially manufactured. The ECU met the functional requirements, but CAN network validation still needs improvement. The software is adaptable to different competition vehicles and can serve as a development foundation for various teams.

**Keywords:** ECU; control unit; electric vehicle; Formula SAE; ESP-32; Simulink; torque control.

## LISTA DE ILUSTRAÇÕES

<b>Figura 1 - Carro de formula SAE fabricado pela equipe UTForce.....</b>	<b>15</b>
<b>Figura 2 - Representação do modelo V de desenvolvimento de software.....</b>	<b>18</b>
<b>Figura 3 - Encoder KY-40.....</b>	<b>19</b>
<b>Figura 4 - Pinout do encoder KY-40.....</b>	<b>20</b>
<b>Figura 5 – Formato dos sinais enviados pelo encoder KY-40.....</b>	<b>20</b>
<b>Figura 6 – Formato dos sinais enviados pelo pedal do acelerador.....</b>	<b>22</b>
<b>Figura 7 – Sensor do pedal do acelerador PF2C/00.....</b>	<b>22</b>
<b>Figura 8 – Sensor de freio analógico 10PP12-03 / 29613942.....</b>	<b>24</b>
<b>Figura 9 – Sensor de freio digital 113.945.515C/H.....</b>	<b>24</b>
<b>Figura 10 – Sensor de freio digital 1J0.927.189.....</b>	<b>24</b>
<b>Figura 11 – Sinal de um sensor de efeito Hall conectado à um disco dentado...<b>25</b></b>	
<b>Figura 12 – Sensor de efeito Hall 1GT101DC.....</b>	<b>26</b>
<b>Figura 13 – Funcionamento do sensor de efeito Hall.....</b>	<b>26</b>
<b>Figura 14 – Sensor de temperatura NTC M12.....</b>	<b>27</b>
<b>Figura 15 – Curva de leitura do sensor de temperatura NTC M12.....</b>	<b>28</b>
<b>Figura 16 – Exemplo de utilização do LM2596 para regulação de 5 V.....</b>	<b>28</b>
<b>Figura 17 – Exemplo de utilização do LM1117T para regulação de 3.3 V.....</b>	<b>29</b>
<b>Figura 18 – Método de geração automatizada de código.....</b>	<b>30</b>
<b>Figura 19 – Exemplo de leitura dos sinais do acelerador .....</b>	<b>31</b>
<b>Figura 20 – Exemplo de tratativa de segurança para pedais de acelerador e freio.....</b>	<b>31</b>
<b>Figura 21 – Exemplo de tratativa dos sinais do acelerador para cálculo do torque.....</b>	<b>32</b>
<b>Figura 22 – Exemplo de implementação para controle de tração .....</b>	<b>33</b>
<b>Figura 23 - Diagrama de blocos inicial para desenvolvimento do hardware.....</b>	<b>36</b>
<b>Figura 24 – Conector TE AMPSEAL 776180-1.....</b>	<b>39</b>
<b>Figura 25 – Esquemático interno ao módulo ESP32-WROOM-32.....</b>	<b>41</b>
<b>Figura 26 – Pinout e periféricos externos ao módulo ESP32-WROOM-32.....</b>	<b>42</b>
<b>Figura 27 – ESP32-WROOM-32 pinout e funcionalidade dos pinos.....</b>	<b>42</b>
<b>Figura 28 – Pinos de protocolo SPI do ESP-32.....</b>	<b>43</b>
<b>Figura 29 – Módulo MCP2515 utilizado para converter protocolo SPI para rede CAN .....</b>	<b>43</b>
<b>Figura 30 – Circuito referente às conexões do módulo MCP2515.....</b>	<b>44</b>
<b>Figura 31 – Alimentação e Conversão de 12 V para 5 V.....</b>	<b>45</b>
<b>Figura 32 – ESP32 Standalone.....</b>	<b>45</b>
<b>Figura 33 – Filtro passa baixa para os sinais analógicos .....</b>	<b>46</b>
<b>Figura 34 – Tratamento de entradas e saídas analógicas e digitais.....</b>	<b>47</b>
<b>Figura 35 – Rede CAN de dois canais.....</b>	<b>48</b>
<b>Figura 36 – Pinout do conector principal.....</b>	<b>48</b>
<b>Figura 37 – Roteamento da Placa – Camada superior.....</b>	<b>49</b>
<b>Figura 38 – Roteamento da Placa – Camada inferior.....</b>	<b>50</b>
<b>Figura 39 – Roteamento da Placa – Modelo 3D superior.....</b>	<b>51</b>
<b>Figura 40 – Roteamento da Placa – Modelo 3D superior.....</b>	<b>51</b>
<b>Figura 41 – Visão geral do sistema desenvolvido para a ECU.....</b>	<b>52</b>
<b>Figura 42 – Função de aquisição de dados .....</b>	<b>53</b>
<b>Figura 43 – Função de segurança.....</b>	<b>54</b>
<b>Figura 44 – APPS Check.....</b>	<b>55</b>

<b>Figura 45 – Brake Check.....</b>	<b>55</b>
<b>Figura 46 – PowerTrain.....</b>	<b>56</b>
<b>Figura 47 – Controle de Tração.....</b>	<b>57</b>
<b>Figura 48 – Refrigeração e Troca de Marcha.....</b>	<b>58</b>
<b>Figura 49 – Saídas digitais e CAN.....</b>	<b>59</b>
<b>Figura 50 – Primeiros testes de hardware realizados com protoboard.....</b>	<b>60</b>
<b>Figura 51 – PCB fabricada através de processos industriais.....</b>	<b>61</b>
<b>Figura 52 – Montagem e teste da placa.....</b>	<b>61</b>
<b>Figura 53 – Painel auxiliar para testes no Simulink .....</b>	<b>62</b>
<b>Figura 54 – Resultados do teste 1 - Dinâmica básica do veículo.....</b>	<b>63</b>
<b>Figura 55 – Resultados do teste 2 - Intervenções de segurança.....</b>	<b>64</b>
<b>Figura 56 – Resultados do teste 3 – Acionamento da refrigeração.....</b>	<b>65</b>
<b>Figura 57 – Resultados do teste 4 – Troca de marcha.....</b>	<b>66</b>
<b>Figura 58 – Resultados teste 5 – Mapas de Torque e Modos de Direção.....</b>	<b>67</b>
<b>Figura 59 – Resultados do teste 6 – Controle de Tração.....</b>	<b>68</b>
<b>Figura 60 – Setup dos sinais utilizados nos testes finais.....</b>	<b>69</b>
<b>Figura 61 – Primeira medição do microcontrolador tratando os dados sem calibração prévia .....</b>	<b>70</b>
<b>Figura 62 – Medição feita após calibração dos sinais analógicos.....</b>	<b>71</b>
<b>Figura 63 – Medição real do acionamento do aviso de troca de marcha.....</b>	<b>72</b>
<b>Figura 64 – Teste inicial da Rede CAN.....</b>	<b>73</b>
<b>Figura 65 – Montagem final da ECU .....</b>	<b>75</b>

## **LISTA DE QUADROS**

<b>Quadro 1 - Levantamento de requisitos funcionais .....</b>	<b>35</b>
<b>Quadro 2 - Levantamento de requisitos não funcionais.....</b>	<b>35</b>
<b>Quadro 3 - Levantamento de requisitos de hardware.....</b>	<b>37</b>
<b>Quadro 4 - Sensores lidos pela ECU .....</b>	<b>38</b>
<b>Quadro 5 – Comparativo entre microcontroladores (parte 1/2) .....</b>	<b>40</b>
<b>Quadro 6 – Comparativo entre microcontroladores (parte 2/2) .....</b>	<b>40</b>
<b>Quadro 7 - Teste 1 - Dinâmica básica do veículo.....</b>	<b>63</b>
<b>Quadro 8 - Teste 2 - Intervenções de segurança.....</b>	<b>64</b>
<b>Quadro 9 - Teste 3 – Acionamento da refrigeração.....</b>	<b>65</b>
<b>Quadro 10 - Teste 4 – Troca de marcha.....</b>	<b>66</b>
<b>Quadro 11 - Teste 5 – Mapas de Torque e Modos de Direção.....</b>	<b>67</b>
<b>Quadro 12 - Teste 6 – Controle de Tração.....</b>	<b>68</b>

## LISTA DE ABREVIATURAS E SIGLAS

ABNT	Associação Brasileira de Normas Técnicas
ADC	<i>Analog-to-Digital Converter</i>
CAN	<i>Controller Area Network</i>
CAN FD	<i>Controller Area Network Flexible Data-Rate</i>
CCP	<i>CAN Calibration Protocol</i>
CPU	<i>Central Processing Unit</i>
DTC	<i>Diagnostic Trouble Code</i>
ECM	<i>Engine Control Module</i>
ECU	<i>Electronic Control Unit</i>
EEPROM	<i>Electrically Erasable Programmable Read-Only Memory</i>
EV	<i>Electric Vehicle</i>
FET	<i>Field-Effect Transistor</i>
FMEA	<i>Failure Mode and Effects Analysis</i>
GPIO	<i>General Purpose Input/Output</i>
HVAC	<i>Heating, Ventilation, and Air Conditioning</i>
I2C	<i>Inter-Integrated Circuit</i>
ISO	<i>International Organization for Standardization</i>
NBR	<i>Normas Brasileiras</i>
OBD	<i>On-Board Diagnostics</i>
PID	<i>Proportional-Integral-Derivative</i>
PWM	<i>Pulse Width Modulation</i>
RAM	<i>Random Access Memory</i>
ROM	<i>Read-Only Memory</i>
SAE	<i>Society of Automotive Engineers</i>
SOC	<i>State of Charge</i>
SPI	<i>Serial Peripheral Interface</i>
UART	<i>Universal Asynchronous Receiver-Transmitter</i>
UTFPR	<i>Universidade Tecnológica Federal do Paraná</i>
VDC	<i>Vehicle Dynamics Control</i>

## SUMÁRIO

<b>1 INTRODUÇÃO .....</b>	<b>13</b>
<b>1.1      Tema .....</b>	<b>14</b>
<b>1.2      Delimitação do tema.....</b>	<b>14</b>
<b>1.3      Problema e premissas .....</b>	<b>14</b>
<b>1.4      Objetivos .....</b>	<b>15</b>
<b>1.4.1      Objetivo Geral.....</b>	<b>16</b>
<b>1.4.2      Objetivos específicos.....</b>	<b>16</b>
<b>1.5      Justificativa.....</b>	<b>16</b>
<b>1.6      Metodologia de pesquisa.....</b>	<b>17</b>
<b>1.7      Cronograma de realização da pesquisa .....</b>	<b>18</b>
<b>2 REFERENCIAL TEÓRICO.....</b>	<b>19</b>
<b>2.1      Referências para Desenvolvimento do Hardware .....</b>	<b>19</b>
<b>2.1.1      Seletor de modos de direção .....</b>	<b>19</b>
<b>2.1.2      Ready to Drive.....</b>	<b>21</b>
<b>2.1.3      Pedal do Acelerador .....</b>	<b>21</b>
<b>2.1.4      Pedal de Freio .....</b>	<b>23</b>
<b>2.1.5      RPM e Velocidade.....</b>	<b>25</b>
<b>2.1.6      Temperatura.....</b>	<b>27</b>
<b>2.1.8      Regulação de tensão.....</b>	<b>28</b>
<b>2.2      Referencias para Desenvolvimento do software .....</b>	<b>29</b>
<b>3 DESENVOLVIMENTO .....</b>	<b>34</b>
<b>3.1      Levantamento de requisitos .....</b>	<b>34</b>
<b>3.2      Desenvolvimento do Hardware .....</b>	<b>36</b>
<b>3.2.1      Escolha dos componentes .....</b>	<b>38</b>
<b>3.2.1.1      Sensores .....</b>	<b>38</b>
<b>3.2.1.2      Conector.....</b>	<b>39</b>
<b>3.2.1.3      Microcontrolador.....</b>	<b>39</b>
<b>3.2.1.4      Rede CAN .....</b>	<b>43</b>
<b>3.2.2      Esquemático da placa .....</b>	<b>44</b>
<b>3.2.3      Roteamento da placa .....</b>	<b>49</b>
<b>3.3      Desenvolvimento do Software .....</b>	<b>52</b>
<b>3.3.1      Aquisição de Dados.....</b>	<b>53</b>
<b>3.3.2      Checagem de Segurança.....</b>	<b>54</b>

3.3.3	<u>Powertrain .....</u>	56
3.3.4	<u>Funções Auxiliares .....</u>	58
3.3.5	<u>Saídas do sistema .....</u>	59
<b>4 TESTES E VALIDAÇÕES.....</b>		<b>60</b>
4.1	<b>Validação do Hardware .....</b>	<b>60</b>
4.2	<b>Validação do Software .....</b>	<b>62</b>
4.3	<b>Validação integrada do Software e Hardware .....</b>	<b>69</b>
<b>5 CONCLUSÃO .....</b>		<b>74</b>
<b>REFERÊNCIAS.....</b>		<b>76</b>
<b>APÊNDICE A: Extrato do código gerado através do Simulink Embedded Coder.....</b>		<b>79</b>

## 1 INTRODUÇÃO

Assim como diversos avanços tecnológicos advindos das pesquisas feitas durante o período de Guerra Fria (conflito político-ideológico que foi travado entre Estados Unidos e União Soviética, entre 1947 e 1991. - Silva, Daniel. Guerra Fria. Mundo Educação), pode-se destacar um em especial que será o principal objeto de estudo deste trabalho: os sistemas embarcados.

Um sistema embarcado é basicamente um conjunto de hardware e software que são desenvolvidos especificamente para uma determinada aplicação, de maneira que, após implementado, o sistema dificilmente sofre alterações. Outro ponto característico dessa tecnologia é o fato de, na maioria dos casos, ele ser instalado em corpos móveis e, por conta disso, ele precisa ser compacto e consumir pouca energia.

Atualmente existem sistemas embarcados dos mais variados tipos - os que utilizam microcontroladores ou, os mais potentes, com microprocessadores, por exemplo -, porém o primeiro sistema nesse modelo surgiu na década de 60 no programa espacial Apollo. Desde então, a tecnologia tem avançado a passos largos e esses equipamentos de com software/hardware dedicados e com processamento em tempo real têm dominado cada vez mais áreas distintas, sendo uma delas o setor automotivo.

Os sistemas embarcados em automóveis são de extrema importância atualmente, pois além de todas as funções básicas, como ativar limpadores de para-brisa, acender os faróis e abaixar os vidros, agora todo o sistema de *powertrain* (sistema responsável por fornecer potência para o veículo) é controlado via injeção eletrônica (que é um sistema embarcado). Quando falamos então de veículos elétricos, temos ainda mais centrais eletrônicas para gerenciar todos os subsistemas do veículo, e de forma semelhante aos veículos à combustão interna, os EVs (*Electric Vehicles*) precisam de uma central eletrônica que gerencie as demandas de torque e potência que serão exigidas do motor. Dito isso, vem a proposta deste estudo: desenvolver uma ECU (do inglês “*Electronic Control Unit*”) para controle de veículos elétricos.

## 1.1 Tema

Visto que a demanda por sistemas de controle com respostas em tempo real tem sido alvo de intenso desenvolvimento ao longo dos últimos anos no mundo automotivo e, de forma semelhante, tem-se investido no desenvolvimento de tecnologias que viabilizem ainda mais a utilização dos veículos elétricos no lugar de veículos à combustão interna, esse universo de sistemas embarcados em veículos elétricos foi tomado como tema principal deste estudo.

## 1.2 Delimitação do tema

Dentro do tema amplo “sistemas embarcados para veículos elétricos” a busca aqui será direcionada para o desenvolvimento de uma Unidade de Controle Eletrônica para gerenciar alguns sinais de um veículo elétrico. A principal função deste módulo será o controle do *Powertrain* (cálculos de torque e comunicação com o inversor de frequência para acionamento do motor elétrico) e o monitoramento de alguns sensores e sinais de outros módulos que garantem a segurança do veículo.

O objetivo secundário do trabalho será a calibração dessa ECU para que a mesma possa ser aplicada em um veículo elétrico de uma equipe de competição da Fórmula SAE (competição estudantil organizada pela *Society of Automotive Engineers*, em que equipes universitárias projetam e fabricam veículos com características definidas pelo regulamento da competição e os apresentam para uma banca de jurados, através de provas dinâmicas e estáticas).

## 1.3 Problema e premissas

Como já foi introduzido neste estudo, a arquitetura elétrica dos veículos tem ficado cada vez mais complexas e tem exigido centrais eletrônicas com capacidade de gerenciamento e tomada de decisão envolvendo vários subsistemas do veículo, incluindo o controle do *powertrain*. Para os veículos à combustão interna essa demanda vem sendo suprida com os módulos de injeção eletrônica e, de semelhante modo, os veículos elétricos também precisam de uma “injeção eletrônica”, porém não para controlar os bicos injetores do motor e sim para se comunicar com inversor de frequência responsável pelo acionamento do motor.

O suprimento dessa necessidade do controle refinado dos motores elétricos em aplicações veiculares é o alvo desse estudo em questão.

#### 1.4 Objetivos

Para atingir o objetivo de entregar um controle refinado do *powertrain* de um veículo elétrico, tem-se como pontos focais o desenvolvimento de um hardware (a ECU) e um software (firmware da ECU) que serão os produtos finais deste projeto.

O software será desenvolvido de maneira que seja calibrável, possibilitando ser implementado em diferentes veículos. Porém, para fins de validação, a ECU será aplicada ao veículo elétrico que está sendo desenvolvido pela equipe UTForce e-Racing, da UTFPR. Portanto, será tomado como base os sensores, atuadores e módulos de comunicação CAN presentes nesse veículo.

Na figura 1 pode-se observar o último protótipo fabricado pela equipe:

**Figura 1 - Carro de formula SAE fabricado pela equipe UTForce**



**Fonte: foto fornecida pela equipe**

#### 1.4.1 Objetivo Geral

Desenvolver uma unidade de controle veicular calibrável para controle refinado do *powertrain* de veículos elétricos, tendo como principal foco os veículos de pequeno porte, como protótipos de Fórmula SAE.

#### 1.4.2 Objetivos específicos

Os objetivos específicos que foram mapeados para direcionar o desenvolvimento da ECU estão listados cronologicamente a seguir:

##### Primeira etapa:

- Definição e esboço do tema/proposta.
- Seleção de orientador e levantamento de fontes.
- Elaboração e submissão da proposta de pesquisa.

##### Segunda etapa:

- Pesquisa e levantamento de informações técnicas.
- Definição e simulação de arquitetura do sistema.
- Desenvolvimento e testes preliminares do hardware.
- Estudo de ferramentas de simulação.

##### Terceira etapa:

- Documentação do desenvolvimento de hardware e software.
- Desenvolvimento e testes de software básico.
- Prototipagem e validação de funções de software.

##### Quarta etapa:

- Aperfeiçoamento e calibração do software.
- Testes estáticos e dinâmicos no veículo.
- Revisão final e documentação.

### 1.5 Justificativa

A utilização desta ECU que será desenvolvida trará muitos benefícios para as equipes de Fórmula SAE que a implementarem, pois, ela ofertará um controle mais refinado e com diferentes possibilidades para requisição de torque. As equipes que ainda não possuem um módulo com essa funcionalidade, geralmente utilizam o sinal do pedal do acelerador conectado diretamente no inversor de frequência, ficando limitado aos perfis de torque do inversor. A equipe UTForce e-Racing utiliza um inversor CVW 300 fornecido pela WEG, que pode operar com até 400A (por até 2

minutos) à uma frequência de chaveamento de 10kHz e frequência de saída de 0 a 300Hz.

A utilização de uma ECU entre o pedal do acelerador e o inversor é muito mais eficaz, pois, se o sinal de do pedal passar por um processamento, juntamente com outros sinais do veículo, pode-se escolher diferentes curvas de aceleração e requisição de torque para o inversor. Isso se traduz em, por exemplo, funções de controle de arranque, aumento de torque em inclinação negativa, reversão do sentido de torque para acionamento do modo regenerativo de energia, entre outras possibilidades.

A ECU em si não é um produto inovador no mercado, pois já existem versões comerciais desse tipo de equipamento, porém será inovadora para pequenos desenvolvedores, como por exemplo as equipes de Fórmula SAE, Baja SAE, e entusiastas do automobilismo, pois o projeto será disponibilizado de forma gratuita e com código aberto, possibilitando a melhoria contínua e desenvolvimento conjunto da comunidade.

## 1.6 Metodologia de pesquisa

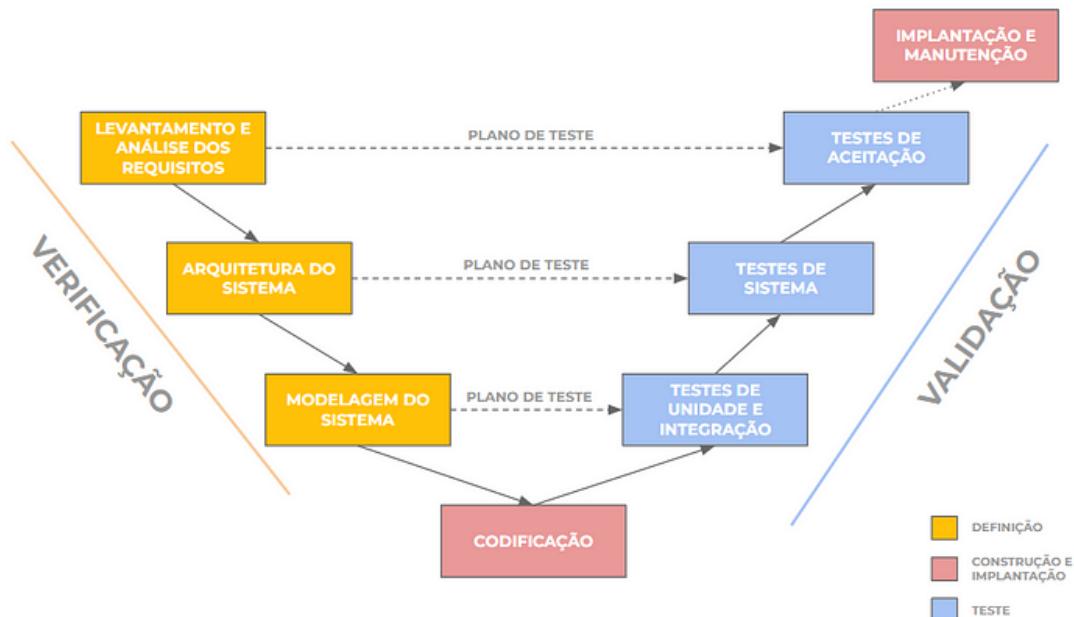
Após estudo dos métodos de Azevedo & Ensslin (autores que apresentam uma metodologia para classificação de trabalhos), percebe-se que esse trabalho será, uma pesquisa aplicada, tendo em vista que o objetivo final é a entrega de um equipamento funcional e com objetivo prático.

Apesar de não ser novidade no mercado automobilístico, a ECU, da forma que será apresentada, será novidade para as equipes de Fórmula SAE de modo geral, o que traz também o caráter exploratório para o trabalho, além do óbvio que é o caráter experimental, visto todos os passos de desenvolvimento, testes e validação que extrapolam a esfera teórica de pesquisa.

Os testes para validação do sistema serão realizados, em primeiro momento, através de simulações virtuais (com ferramentas de simulação de circuitos, com Proteus por exemplo) e de simulações de sinais físicos (por meio de geradores de função, potenciômetros, fontes de tensão e plataformas de desenvolvimento como arduino/raspberry para simulação de comunicação). Em um segundo momento serão realizadas validações com a ECU embarcada de fato em um veículo, podendo ou não

passar por testes em um dinamômetro dependendo da necessidade e disponibilidade dos envolvidos. No geral, para o desenvolvimento do software será utilizada uma metodologia conhecida como “Modelo V”, que determina a sequência de ações a serem seguidas para garantir a entrega de um software seguro, funcional e revisado ao final do processo.

**Figura 2 - Representação do modelo V de desenvolvimento de software**



Fonte: MODELO V - Ciclo de Vida de Desenvolvimento de Software - [link](#)

## 1.7 Cronograma de realização da pesquisa

O trabalho foi desenvolvido durante quatro semestres letivos, levando em conta o período decorrido nas matérias de Metodologia Aplicada ao TCC, TCC1 e TCC2, além da utilização do período de férias, para adiantar o desenvolvimento do hardware que foi o objeto de estudo no TCC1.

Levando em conta tal calendário, foi levantado os principais pontos de estudo e entregas que seriam trabalhadas até a finalização da pesquisa (objetivos específicos) e distribuídas entre o início da matéria de Metodologia Aplicada ao TCC (na primeira semana de agosto de 2022) e o final previsto para a matéria de TCC2 (em meados de 2024).

## 2 REFERENCIAL TEÓRICO

Este capítulo abordará toda a pesquisa necessária para o chegar na definição do sistema e definir materiais e ferramentas a serem utilizadas.

A pesquisa referencial por exemplos de aplicação de hardware e de software para sistemas embarcados em veículos elétricos, com foco em Formula SAE.

### 2.1 Referências para Desenvolvimento do Hardware

Com base em projetos de Formula SAE (destacados ao decorrer do capítulo), alguns componentes eletrônicos foram selecionados para estudo, visando a posterior possibilidade de utilização na placa da ECU.

#### 2.1.1 Seletor de modos de direção

É comum, nos veículos de passeio mais atuais, botões no painel de controle para que o motorista selecione entre diferentes modos de direção, como por exemplo modo “Eco” e modo “Sport”. No Fórmula SAE também é interessante ter essa funcionalidade, visto que existem diferentes provas dinâmicas durante a competição (aceleração, *skid pad*, *autocross* e enduro) e cada uma delas exige um comportamento diferente do veículo. Pensando nisso, foi planejado implementar alguns botões no painel, para que o motorista escolha uma estratégia de toque diferente para cada situação.

A responsável por receber esses inputs dos botões e selecionar um mapa de torque diferente é a ECU. Pensando na melhor maneira de fazer essa comunicação, foi decidido utilizar um *encoder* digital de 20 posições, ao invés de vários botões, pois, desta forma, pode-se economizar entradas digitais do microcontrolador.

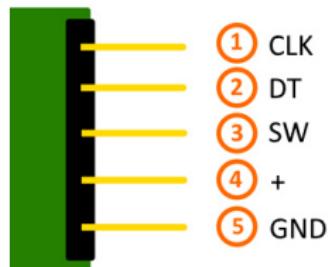
Nas figuras 5, 6 e 7 segue informações sobre o *encoder* KY-40, que foi escolhido para a aplicação:

**Figura 3 - Encoder KY-40**



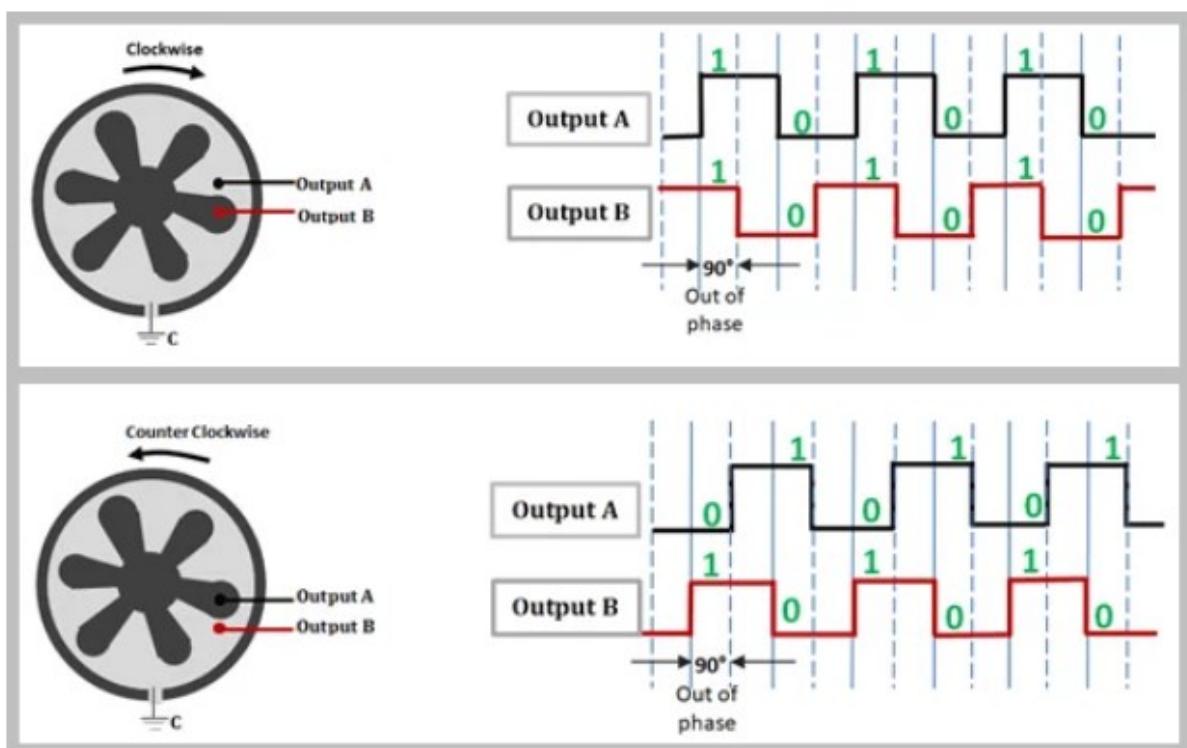
**Fonte: KY-040 Rotary encoder datasheet**

Figura 4 - Pinout do encoder KY-40



Fonte: KY-040 Rotary encoder datasheet

Figura 5 – Formato dos sinais enviados pelo encoder KY-40



Fonte: site [electricalibrary.com](http://electricalibrary.com) - [link](#)

O sensor trabalha com um nível de tensão de 5 V para alimentação e sinal lógico.

### 2.1.2 Ready to Drive

O *Ready to Drive* é uma função que será executada dentro da ECU. Ela é definida pelo regulamento da competição (*Fórmula SAE® Rules 2023 - Version 1.0, 1 setembro de 2022*) e consiste basicamente em conferir se alguns sinais como pedal do freio, pedal do acelerador, comunicação com inversor, entre outros, estão funcionando de acordo. Além desses sinais, mais um input deve ser analisado – o botão de “*Ready to Drive*”. Este botão é posicionado na cabine do piloto (pode ser no volante ou no painel), e ele deve ser acionado quando o piloto estiver pronto para dirigir.

No caso em estudo, o modelo do botão ainda não foi definido pela equipe, mas ele pode ser qualquer tipo de botão autotravante que enviará continuamente 12 V ou 5 V para a ECU quando acionado.

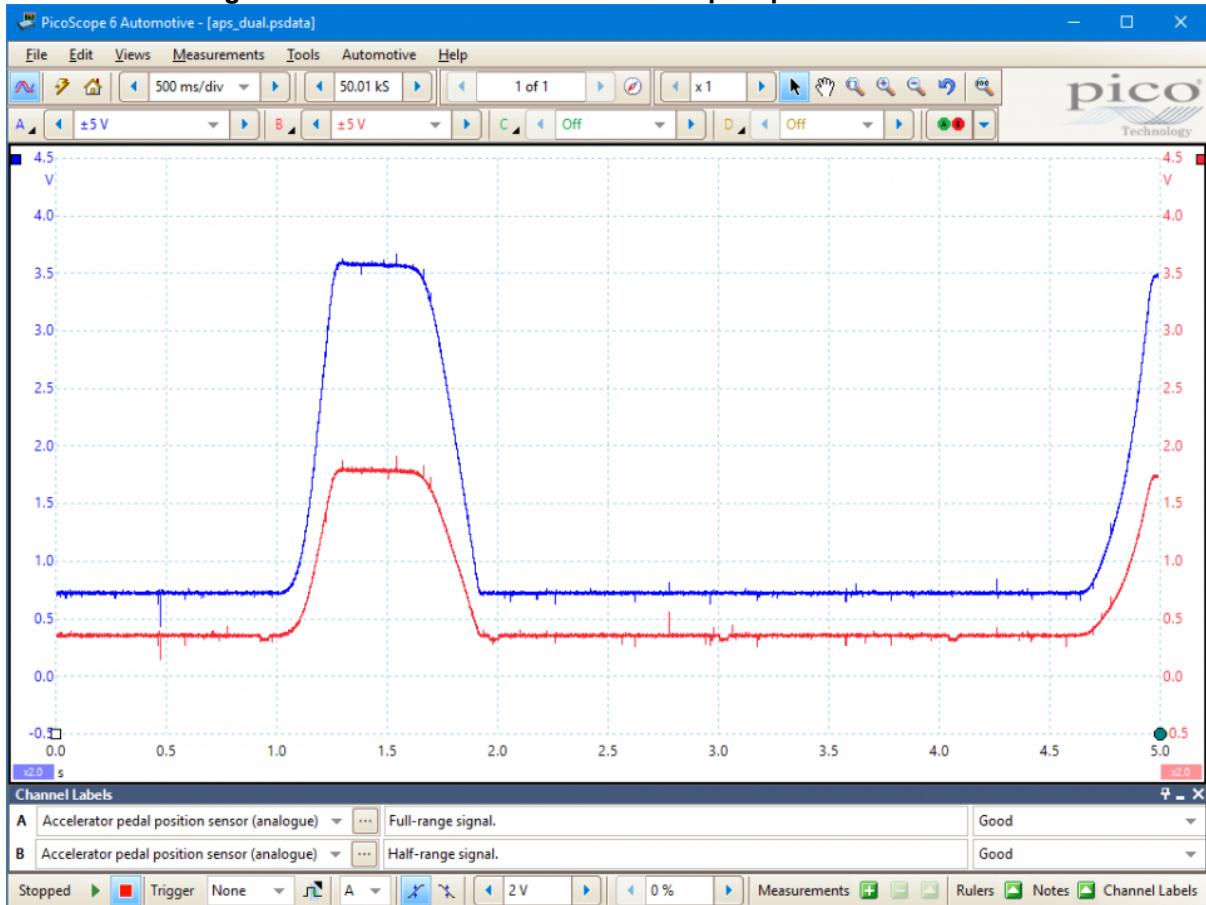
### 2.1.3 Pedal do Acelerador

O pedal do acelerador é, sem dúvidas, o principal input para a ECU, visto que é através dele que o motorista irá requisitar torque para o veículo. Por conta da criticidade deste input, é necessário ter uma plausibilidade do mesmo, de forma que se possa identificar no software caso haja falha de chicote, mal contato ou qualquer leitura indevida do pedal.

Para isso, foi utilizado dois sensores analógicos (potenciômetros) fazendo a leitura do pedal em paralelo e, geralmente, com escalas de tensão diferentes. Internamente à ECU esses sinais são lidos em entradas analógicas distintas e é feito o cálculo da porcentagem de inclinação do pedal, obtendo-se dois valores (cada um advindo de um potenciômetro). Ao final, esses valores são comparados e devem estar similares, respeitando um valor máximo de erro. Exemplos de aplicação para este tipo de sinal (*dual pot*) podem ser encontrados em dois dos TCCs já citados anteriormente como fonte de pesquisa: *Desenvolvimento do Controle Eletrônico do Acelerador* (ROSSO, Marco Antônio Zolett, UFSC, 2017) e *Desenvolvimento do Sistema Eletrônico de um Veículo do Tipo Formula Student Elétrico Baseado nas Regras da Fórmula SAE* (OLIVEIRA, Gabriel Luis de, UFRGS, 2020).

Veja na figura 6 um exemplo real de sinais coletados de um pedal do acelerador:

**Figura 6 – Formato dos sinais enviados pelo pedal do acelerador**



Fonte: site picoauto.com - [link](#)

Veja na figura 7 o sensor de APP (Accelerator Pedal Position) que será utilizado pela equipe na próxima competição:

**Figura 7 – Sensor do pedal do acelerador PF2C/00**



Fonte: site sigapeca.com - [link](#)

O sensor PF2C/00 possui 3 pinos (Vcc, GND e sinal) e trabalha com um range entre 0.4 e 4.5 V.

### 2.1.4 Pedal de Freio

O sistema de freio pode enviar tanto sinais analógicos quanto digitais para a ECU. Ambos os sinais (analog ou digital) podem ser coletados através do pedal de freio ou através do fluido de freio. No caso do pedal, o dado pode ser advindo de um sensor potenciômetro, coletando o curso do pedal (como também é feito para o pedal do acelerador), ou através de um sensor fim-de-curso que deve ser posicionado de forma muito próxima do curso inicial do pedal, de forma que o mínimo de deslocamento já acione o sinal digital. No caso do fluido de freio, geralmente é instalado um sensor de pressão que emite um sinal digital toda vez que a pressão passa de um valor calibrado ou, no caso de um sensor analógico, ele envia constantemente uma tensão correspondente ao valor da pressão atual do fluido. Um exemplo de aplicação deste sinal digital via sensor de pressão pode ser encontrado na monografia *Implementação da Arquitetura Eletrônica em Protocolo Controller Area Network para Veículo Off-Road Tipo BAJA* (COSTA, Bianca Fernandes, Universidade São Judas Tadeu, 2021).

A informação do acionamento do freio será usada pela ECU como input de segurança e para realização de plausibilidade com outros circuitos de segurança já implementados no veículo. Um exemplo de aplicação deste sinal na ECU é a utilização do sinal do pedal de freio e do pedal do acelerador para garantir que o veículo não seja acelerado e freado ao mesmo tempo. Quando o freio está acionado, qualquer sinal de requisição de torque não é repassado ao inversor de frequência.

Também é interessante requisitar uma leitura inicial do pedal de freio antes de liberar as demais funções da ECU, de modo que o sistema só entra em total funcionamento após a garantia de que o freio está funcionando e se comunicando adequadamente com a ECU.

Nas figuras 9,10 e 11 temos exemplos de sensores (digital e um analógico) que poderão ser utilizados pela equipe na próxima competição.

**Figura 8 – Sensor de freio analógico 10PP12-03 / 29613942**



Fonte: site genebraauopecas.com - [link](#)

**Figura 9 – Sensor de freio digital 113.945.515C/H**

**113.945.515C/H**

**VW:** Fusca Sedan, Brasilia, Kombi, Gol, Voyage, Parati, Saveiro, Passat, Santana, Quantum, todos os modelos.



M10 x 1,0  
CON.



4.5 bar

Fonte: catálogo 3RHO - [link](#)

**Figura 10 – Sensor de freio digital 1J0.927.189**

**1J0.927.189**

**VW:** Gol, Golf, Fox, todos os modelos bipolar, embreagem do Gol Geração 3.

**Audi:** A3.



BOTÃO AUTO  
REGULÁVEL



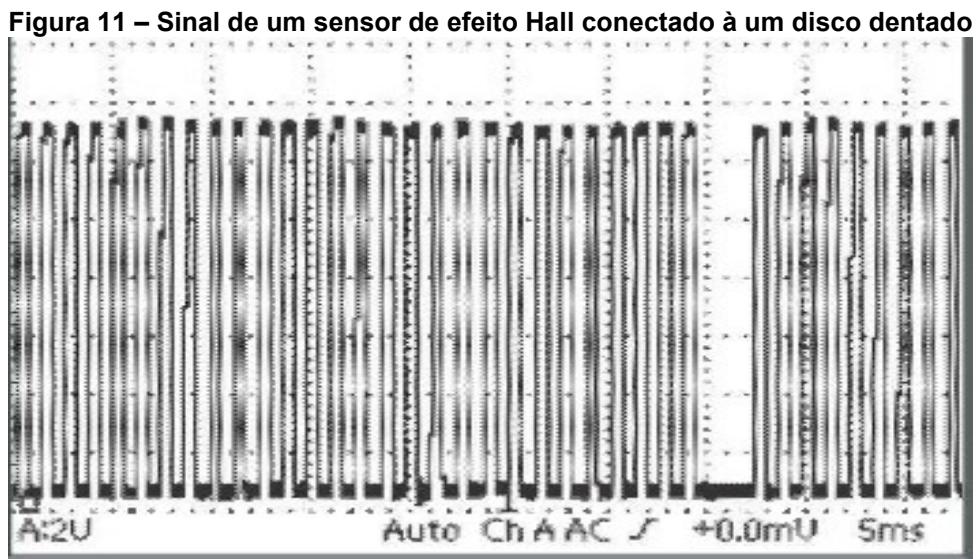
Fonte: catálogo 3RHO - [link](#)

### 2.1.5 RPM e Velocidade

Os sinais de RPM e de velocidade são de extrema importância para o controle de torque do veículo, que é a função principal da ECU. Estes sinais podem ser aquisitados de duas formas principais quando falamos de Fórmula SAE: via sensor digital ou via rede CAN (quando a máquina elétrica já tem algum sensor interno e o inversor de frequência faz a leitura e transmissão da informação).

Na maior parte dos casos, o que ocorre é a utilização de um único sensor de efeito Hall para obter tanto o RPM (rotação do motor ou da roda) quanto a velocidade linear do veículo (a velocidade linear pode ser calculada com os dados de RPM e circunferência da roda). Na aplicação alvo deste desenvolvimento, este será o caso.

Ao ler o livro *Hall Effect Sensing and Application* (HONEYWELL, 2010) mais especificamente os capítulos 5 (*Hall-based Sensing Devices*) e 6 (*Applying Hall-effect Sensing Devices*), pode-se entender o funcionamento e utilização do sensor de efeito Hall. No geral, ele deve ser conectado próximo à alguma estrutura circular dentada que acompanhe a rotação do motor ou da roda. Essa estrutura dentada geralmente tem 1 ou dois dentes removidos, de modo que o sensor não induz corrente quando esta região “sem dentes” está passando. Assim, ao olhar o sinal emitido pelo sensor, pode-se identificar toda vez que a estrutura completa uma volta. Veja o exemplo na figura 11:



Fonte: Site doutorie.com - [link](#)

Na figura 11, observa-se uma leitura real dos sinais de um sensor Hall. Perceba que há uma região do gráfico que fica sem pulsos positivos por um instante. Esta região sem pulsos identifica o início/término de uma revolução do sistema.

Obtendo o RPM, através da estratégia descrita acima, pode-se calcular a também a velocidade se baseando na circunferência do pneu.

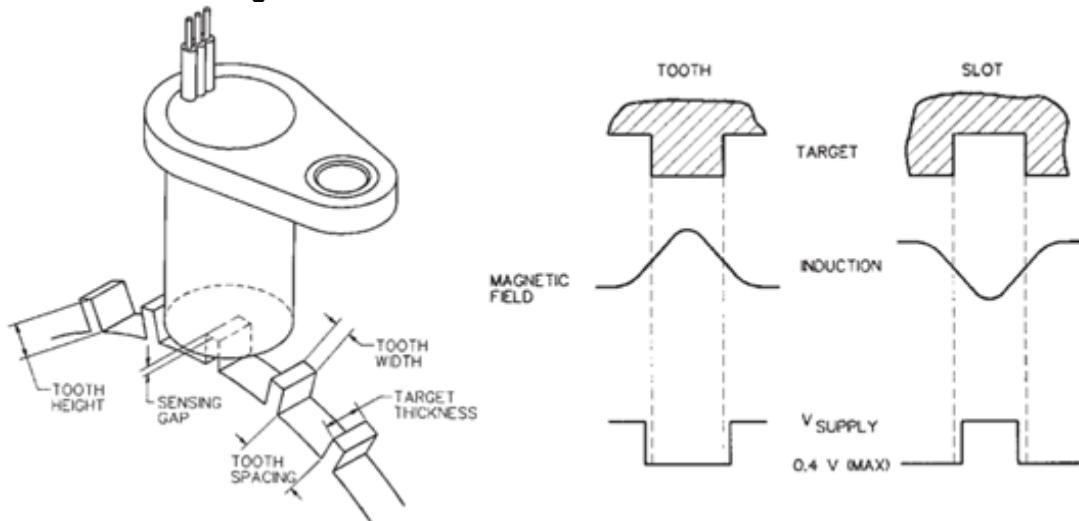
Nas figuras 13 e 14, segue mais informações sobre o sensor de efeito hall que poderá ser utilizado pela equipe na próxima competição:

**Figura 12 – Sensor de efeito Hall 1GT101DC**



Fonte: datasheet do componente - [link](#)

**Figura 13 – Funcionamento do sensor de efeito Hall**



Fonte: datasheet do componente - [link](#)

O sensor pode ser alimentado de 4.5 V a 24V. Seu sinal lógico “High” é correspondente à tensão de alimentação, enquanto o nível lógico “Low” pode variar entre 0 e 0.4V.

## 2.1.6 Temperatura

O sensor de temperatura é um item opcional para a ECU, mas caso as equipes queiram utilizar a ECU para controle do arrefecimento, terá uma entrada analógica separada para isto.

Um dos modelos mais comuns de sensores de temperatura no ramo automotivo é o NTC, que varia sua resistência de acordo com a temperatura. Como a ECU não faz leitura de resistência diretamente, é necessário acrescentar um circuito divisor de tensão para converter a variância de resistência em variância de tensão, que será lida em uma porta analógica do microcontrolador.

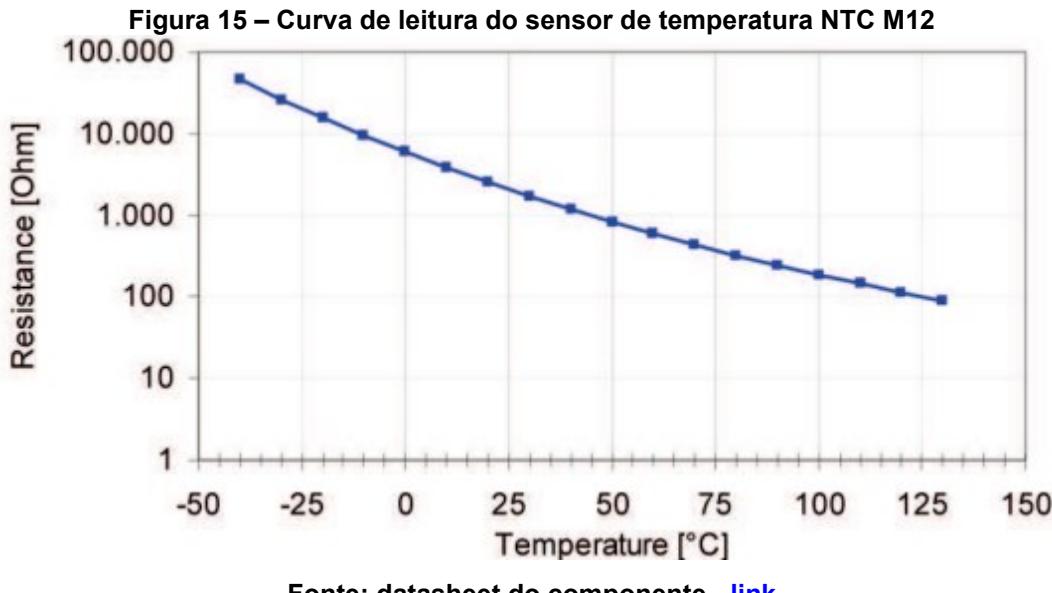
No artigo *Comparação entre Três Tipos de Sensores de Temperatura em Associação com Arduino* (MARTINAZZO; ORLANDO, 2016), é possível encontrar um exemplo de conexão e leitura de um sensor NTC com um Arduino. O processo se assemelha muito com o que será feito na ECU.

Nas figuras 15 e 16 encontra-se informações sobre o sensor que poderá ser utilizado pela equipe na próxima competição e sua curva de acionamento:

**Figura 14 – Sensor de temperatura NTC M12**



Fonte: datasheet do componente - [link](#)



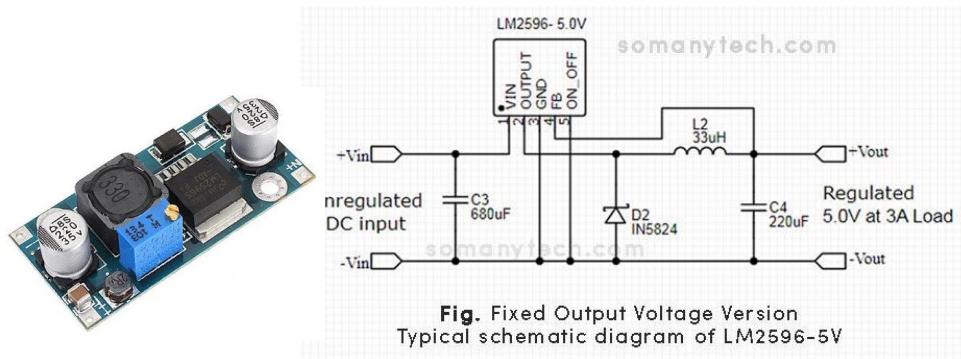
### 2.1.8 Regulação de tensão

No carro que foi utilizado como referência para este trabalho, utiliza-se uma alimentação de 12 V para o sistema de baixa tensão. Porém, os componentes periféricos da *ECU* utilizam entre 5 V e 3.3 V.

Desta forma, faz-se necessária a regulação de tensão, tanto para alimentação dos componentes quanto para a comunicação com os atuadores do veículo.

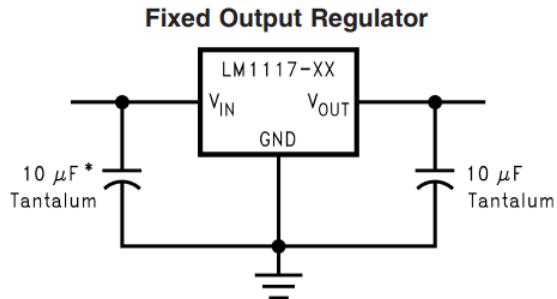
Para a regulação de 12 V para 5 V, pode ser utilizado um circuito Buck, que tem como componente principal o CI LM2596 – 5.0V, como pode-se observar na figura 16:

**Figura 16 – Exemplo de utilização do LM2596 para regulação de 5 V**



Para a regulação de 5 V para 3.3 V, pode-se utilizar o CI LM1117T – 3.3 V, como pode-se observar na figura 17:

**Figura 17 – Exemplo de utilização do LM1117T para regulação de 3.3 V**



Fonte: datasheet do componente

## 2.2 Referencias para Desenvolvimento do software

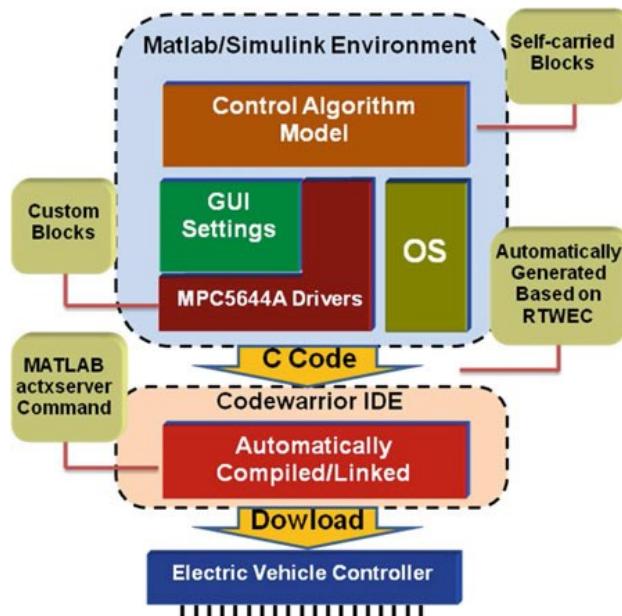
Os referenciais teóricos para desenvolvimento do Software foram diversos, desde tutoriais para utilização das ferramentas do Simulink (como *Embedded Coder*, configurações de Hardware, geração de código, entre outros) até soluções complexas para controle de torque em veículos elétricos com tração nas quatro rodas, por exemplo. Abaixo seguem algumas das principais referências que me guiaram para o desenvolvimento do software autoral.

O firmware da placa foi desenvolvido através das ferramentas do Matlab/Simulink R2024a para geração de código C/C++ a partir de um modelo de blocos do Simulink.

Este firmware tem como objetivo servir com um passo inicial para que as equipes de Formula SAE possam, de acordo com cada necessidade, se aprofundar e desenvolver funções mais complexas e refinadas.

O artigo *Automated Code Generation for Development of Electric Vehicle Controller* (GENG; OUYANG; JIANQIU; LIANGFEI, 2013), trouxe uma visão essencial sobre a automatização da geração de código automotivo a partir modelagens feitas em linguagem de mais alto nível (como os blocos do Simulink), bem como suas vantagens e desafios. No Artigo é exemplificado de maneira prática como desenvolver uma aplicação para um controlador MPC5644A, perpassando todos os pontos do Modelo V citado na introdução deste trabalho. Veja uma síntese da geração automatizada de código representada na figura 18:

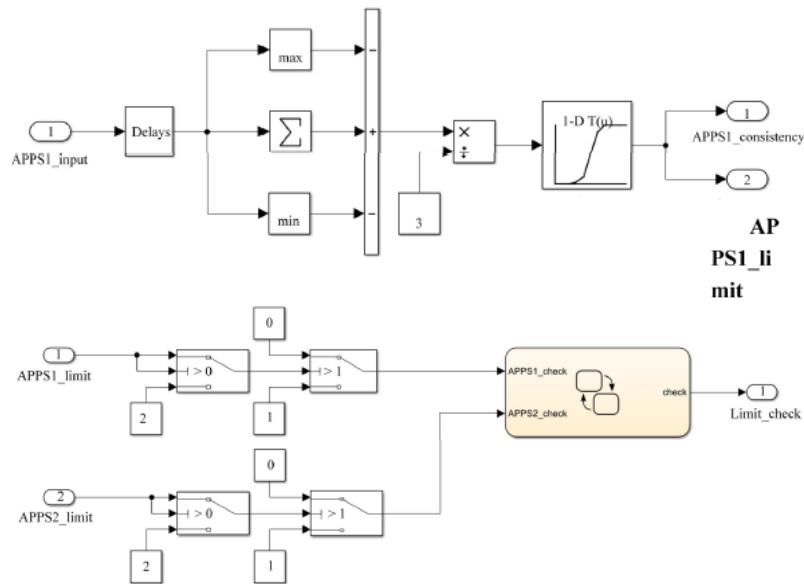
**Figura 18 – Método de geração automatizada de código**



Fonte: artigo “Automated Code Generation for Development of Electric Vehicle Controller”

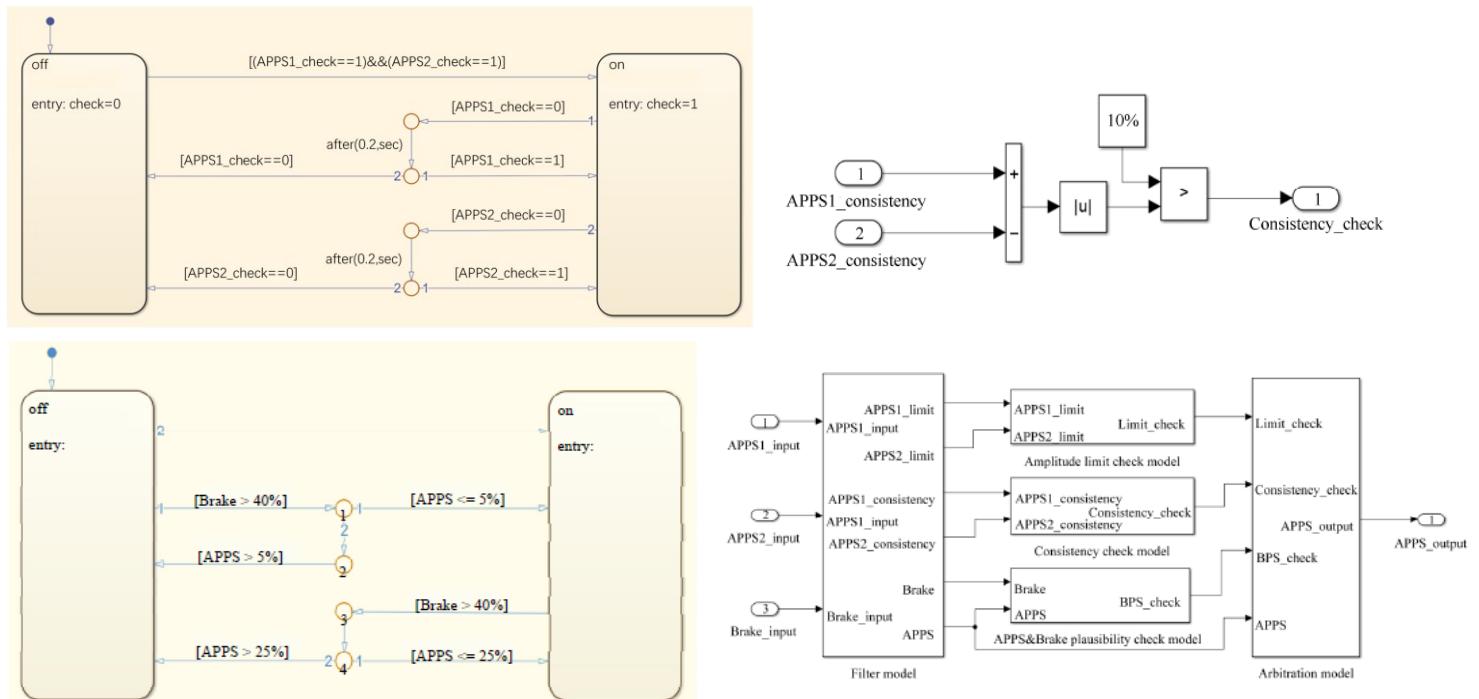
Já o artigo *Reliability Control of Electric Racing Car’s Accelerator and Brake Pedals* (YUXING; HONG; D’APOLITO, 2020), traz uma exemplificação de como tratar o sinal do pedal de freio e do acelerador de maneira segura. Veja os exemplos nas figuras 26 e 27.

**Figura 19 – Exemplo de leitura dos sinais do acelerador**



Fonte: artigo Reliability Control of Electric Racing Car's Accelerator and Brake Pedals

**Figura 20 – Exemplo de tratativa de segurança para pedais de acelerador e freio**

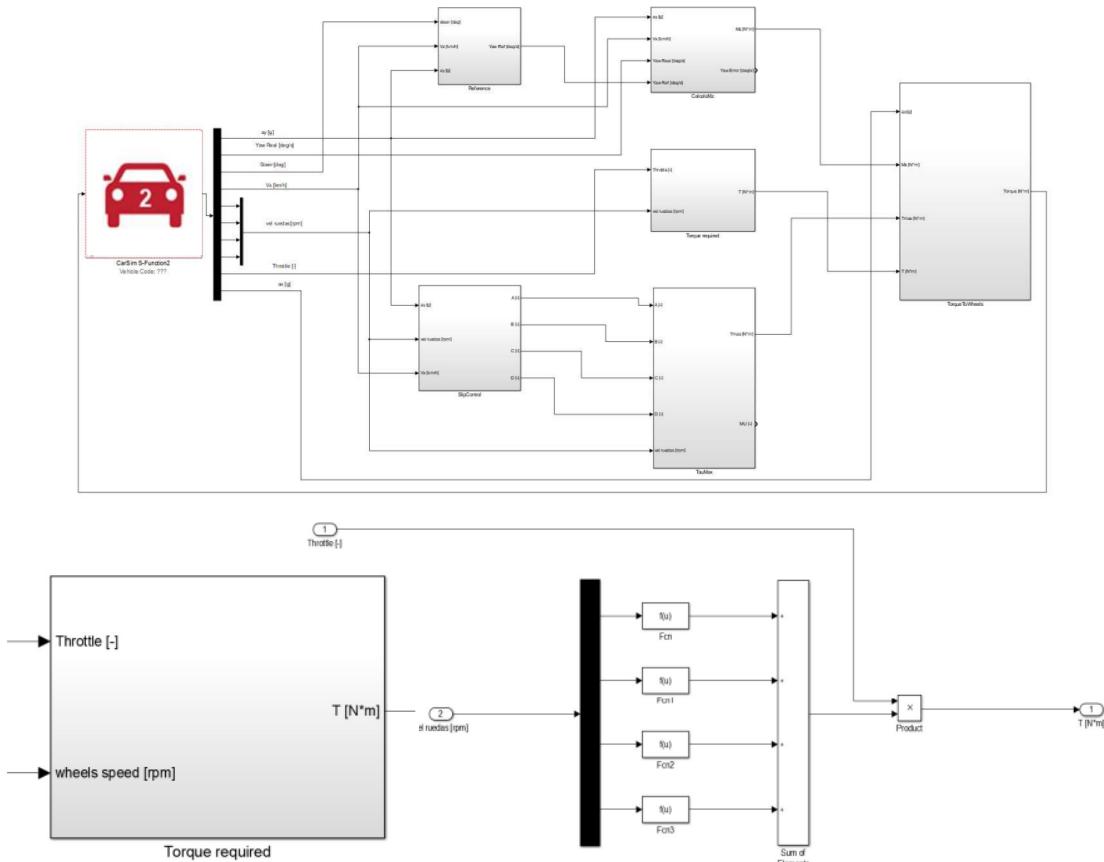


Fonte: artigo Reliability Control of Electric Racing Car's Accelerator and Brake Pedals

Uma das principais funções da ECU é o controle do torque. Dessa forma, busquei também referenciais teóricos para ajudar na elaboração da “cadeia de torque” do projeto.

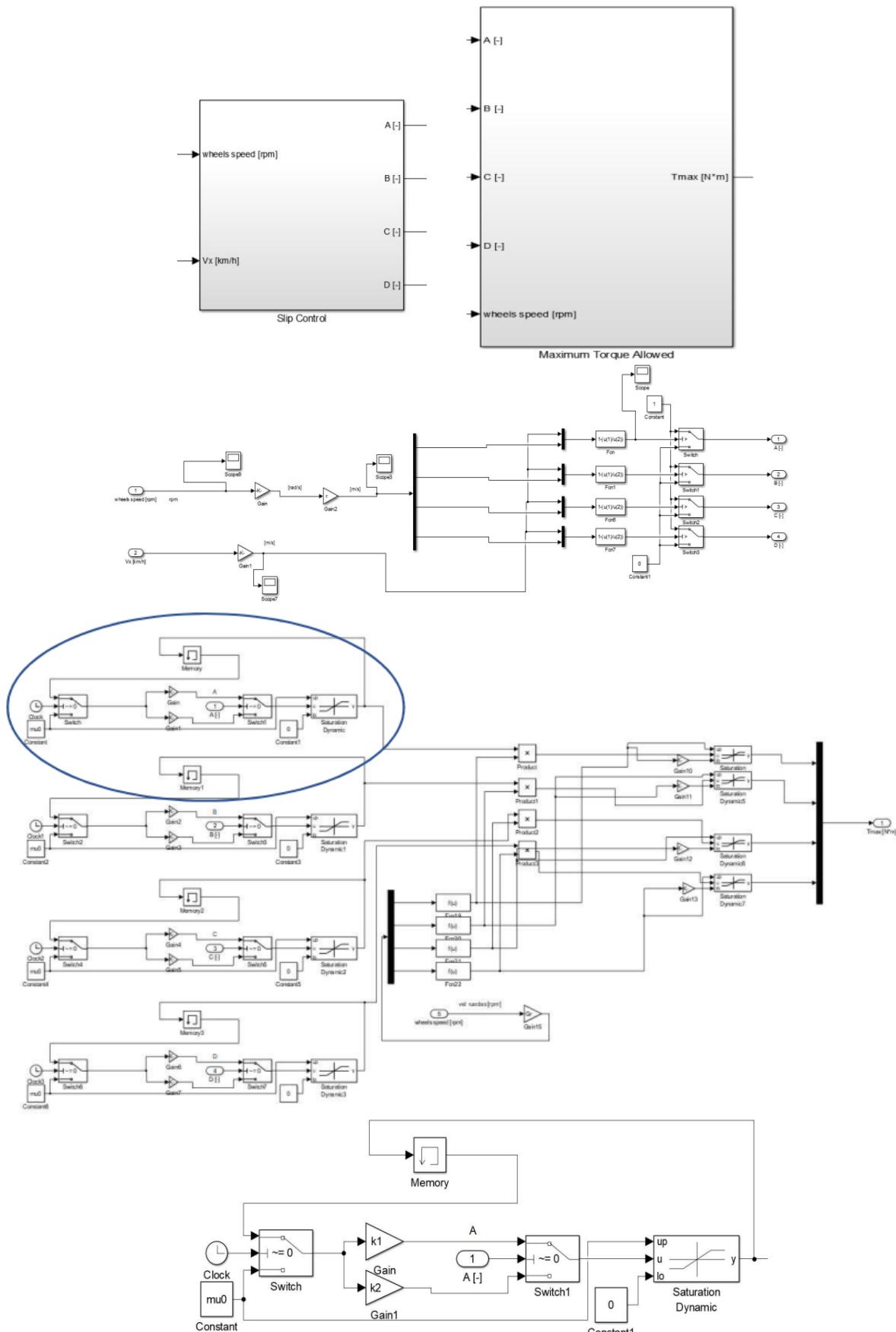
Uma das principais fontes para elaboração da cadeia de torque foi a tese de mestrado *Design of Torque Vectoring system for Formula SAE electric vehicle* (FRICANO, 2019). Desta leitura pude extrair ideias para o cálculo do torque básico requisitado pelo pedal e principalmente, como implementar um controle de tração. Veja o exemplo implementado nas figuras 28 e 29.

**Figura 21 – Exemplo de tratativa dos sinais do acelerador para cálculo do torque**



**Fonte:** artigo *Design of Torque Vectoring system for Formula SAE electric vehicle*

**Figura 22 – Exemplo de implementação para controle de tração**



**Fonte:** artigo Design of Torque Vectoring system for Formula SAE electric vehicle

### 3 DESENVOLVIMENTO

Neste capítulo, abordaremos o desenvolvimento da ECU, conforme os conceitos estabelecidos no referencial teórico. O desenvolvimento foi dividido em três partes: a primeira foca no levantamento de requisitos, a segunda apresenta o processo de criação do hardware, incluindo a escolha de componentes e o esquemático da placa; a terceira parte se concentra no desenvolvimento do software que será embarcado no microcontrolador.

#### 3.1 Levantamento de requisitos

Na etapa de levantamento de requisitos, é realizada uma pesquisa geral sobre a arquitetura eletroeletrônica dos carros elétricos de fórmula SAE, afim de entender qual o papel exato da ECU no sistema e definir suas responsabilidades.

Neste momento, é necessário, baseando-se nos requisitos funcionais, definir os sensores e/ou atuadores mínimos para que se possa fazer as intervenções adequadas no sistema do veículo. Com isso, é possível mapear as entradas e saídas digitais/analógicas, protocolos de comunicação, processamento do sistema, entre outros detalhes, a fim de direcionar a escolha do microcontrolador e dos componentes periféricos para a placa.

Além das interfaces de hardware, também é necessário entender quais as funcionalidades básicas que o software embarcado precisa apresentar.

O entendimento dos requisitos de software e hardware é de extrema importância para garantir a para que a comunicação com os outros módulos do veículo ocorra de maneira apropriada e que a ECU seja, ao máximo possível, flexível entre diferentes projetos de Fórmula SAE.

Baseado em uma pesquisa informal com participantes de Fórmula SAE de diferentes equipes, juntamente com pesquisa em fóruns de Fórmula SAE, regulamento da competição e trabalhos realizados por colegas de outras universidades, como *Desenvolvimento do Controle Eletrônico do Acelerador* (ROSSO, Marco Antônio Zolett, UFSC, 2017), *Telemetria de um Veículo Baja SAE através de Rede CAN* (NUNES, Tomaz Filgueira, UFRN, 2016), *Implementação da Arquitetura Eletrônica em Protocolo Controller Area Network para Veículo Off-Road Tipo BAJA* (COSTA, Bianca Fernandes, Universidade São Judas Tadeu, 2021) e *Desenvolvimento do Sistema Eletrônico de um Veículo do Tipo Formula Student*

*Elétrico Baseado nas Regras da Fórmula SAE* (OLIVEIRA, Gabriel Luis de, UFRGS, 2020), foram levantados os seguintes requisitos funcionais e não funcionais para a ECU, pensando na aplicação de um veículo de Fórmula SAE elétrico:

**Quadro 1 - Levantamento de requisitos funcionais**

<b>Requisitos Funcionais</b>	
<b>Espera-se da ECU</b>	<b>Não se espera da ECU</b>
Controle do toque aplicado no motor	Controlar carga e descarga do pack de baterias
Disponibilização de diferentes mapas de toque para cada modo de direção	Controlar sistema de desligamento do veículo (shutdown system)
Tratativas adicionais de segurança (além das medidas obrigatórias, que são externas à ECU)	Controlar sistemas de segurança obrigatórios definidos no regulamento da competição
Redundância dos sistemas de segurança principais	Datalogger dos sensores
Controle adicional do arrefecimento	Transmissão de informações por telemetria
Comunicação via rede CAN (dois barramentos)	Controle do sistema de frenagem (ABS)

**Fonte: autoria própria**

**Quadro 2 - Levantamento de requisitos não funcionais**

<b>Requisitos Não Funcionais</b>
Confiabilidade: deve resistir a trepidações (ambiente de pista asfaltada) e ambientes levemente úmidos
Processamento: deve ter respostas em tempo real, de forma que não prejudique a dirigibilidade do veículo
Tratativa de falhas: a ECU deve identificar falhas internas e indicar "erro" de alguma forma que fique visível ao piloto e/ou time de manutenção.

**Fonte: autoria própria**

Através do levantamento inicial dos requisitos funcionais e não funcionais, foi possível definir o escopo mínimo da Hardware e Software, que foram a base para o início do design da placa e do firmware, apresentados nos subcapítulos a seguir.

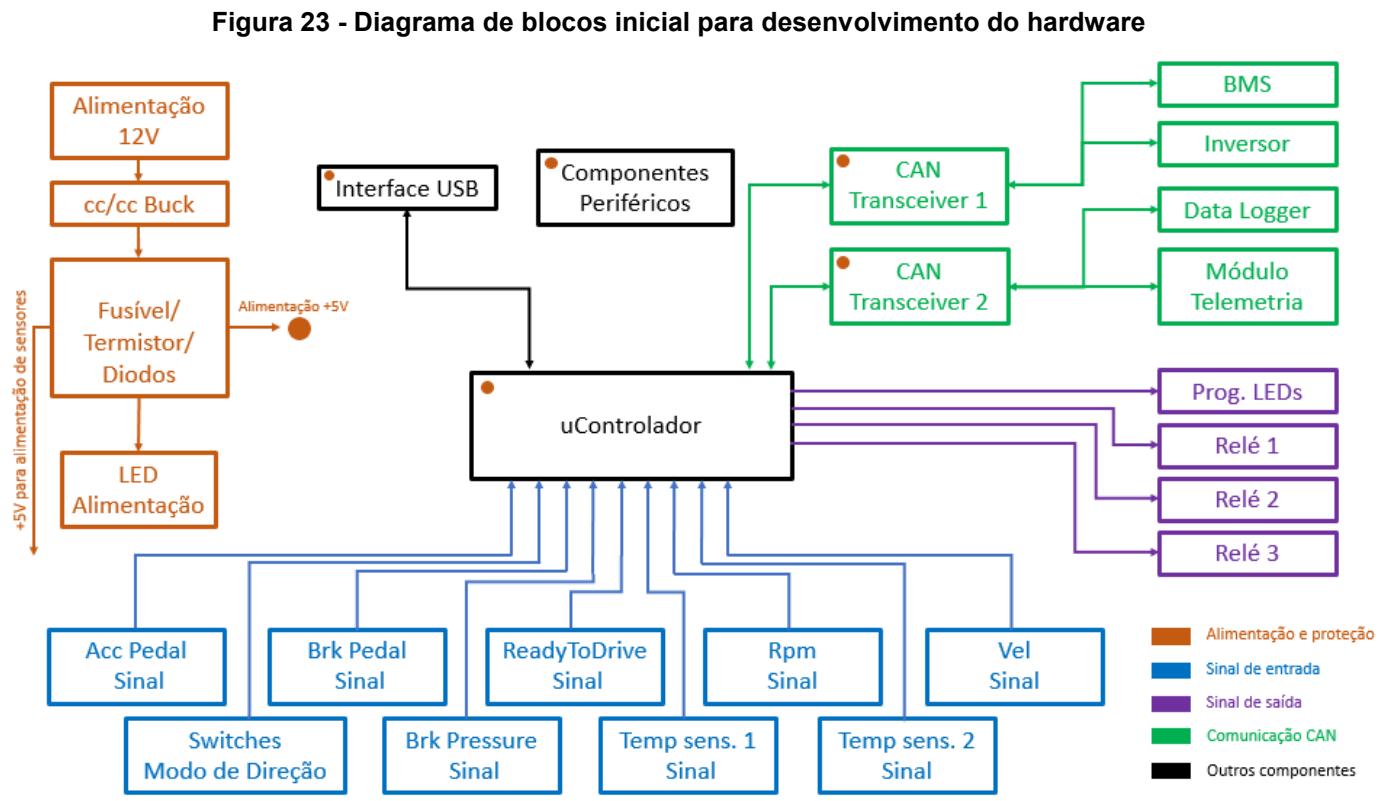
### 3.2 Desenvolvimento do Hardware

O Hardware foi desenvolvido com a ferramenta Altium Designer e softwares de simulação como o Proteus, por exemplo.

O circuito foi composto por componentes como: microcontrolador, *transceiver* de comunicação CAN, componentes de segurança (fusíveis, termistores, diodos, etc), conversor cc/cc buck, leds indicadores, entre outros componentes periféricos a serem definidos.

A placa de circuito impresso foi fabricada na china (com duas faces e furos metalizados) e a soldagem dos componentes será feita de maneira manual nos laboratórios da faculdade.

Na figura 23 está representado, em forma de diagrama de blocos, uma ideia inicial das interfaces de hardware e software propostos para o projeto:



Fonte: autoria própria

Tomando como base os requisitos previamente levantados, foi possível mapear os requisitos mínimos de hardware, incluindo quantidade de entradas e saídas analógicas e digitais, protocolos de comunicação, níveis de tensão, tipo de conectores a serem usados, entre outros.

Dessa forma, foi feita a escolha do microcontrolador (ponto de partida do projeto de hardware) e, posteriormente, todos os componentes periféricos, como: reguladores de tensão, *transceivers* SPI-CAN, circuito integrado para comunicação serial (USB), indutores, capacitores, diodos, leds e resistores em geral.

No quadro 3 encontra-se o desdobramento dos requisitos de Hardware:

**Quadro 3 - Levantamento de requisitos de hardware**

Requisitos de Hardware		
Requisito	Qtdd	Descrição
Digital Inputs	6	Pedal de Freio digital
		RTD ( <i>Ready to Drive</i> - circuito definido no regulamento)
		Switch entre modos de direção (2 inputs = 4 possíveis modos)
		Sensor de RPM
		Sensor de velocidade
Digital Outputs	3	LED programável 1
		Relé para lâmpada indicadora de falha
		PWM (ativação de bomba de refrigeração ou FAN)
Analog Inputs	3	Pedal do acelerador (potenciômetro duplo - 2 pinos analog)
		Pedal de Freio analog
		Sensores de temperatura
Analog Outputs	0	Nenhuma saída analog necessária
Proteção da PCB	-	Conectores com travas e isolamento de água
Regulação de tensão	3	Buck conversor de 12 V para 5 V
		Regulador de 5 V para 3.3 V
		<i>Level shifter</i> bidirecional (3.3 V - 5 V) para comunicação
Supply outputs (Pos and GND)	4	4 pares de alimentação 5 V para sensores (output)
Interface de comunicação	4	Serial - micro USB
		Serial Tx/Rx
		SPI
		CAN (2 nós independentes)
Outros	1	LED indicador de alimentação da ECU

**Fonte: autoria própria**

### 3.2.1 Escolha dos componentes

A escolha dos componentes que foram utilizados no projeto foi baseada primariamente nos projetos referenciados durante a pesquisa. O segundo critério para escolha dos componentes foi a disponibilidade dos mesmos no mercado brasileiro.

#### 3.2.1.1 Sensores

Observando os requisitos de hardware levantados, percebe-se que alguns deles estão relacionados com a leitura de determinados sensores analógicos e digitais. No quadro 4, estão listados os modelos de sensores que foram utilizados como base de configuração da ECU. Esses modelos foram escolhidos com referência no carro da UTForce e-Racing e na pesquisa bibliográfica realizada previamente.

**Quadro 4 - Sensores lidos pela ECU**

<b>Sensores</b>			
<b>Sensor</b>	<b>Tipo</b>	<b>Quantidade</b>	<b>Part Number</b>
Switch entre modos de direção	Digital	1	KY-40
<i>Ready to Drive</i>	Digital	1	Fabricado pelas equipes
Pedal de Freio digital	Digital	1	113.945.515C/H ou 1J0.927.189
Pedal do acelerador	Analógico	2	PF2C/00 magneti marelli
Pedal de Freio analog	Analógico	1	29613942 - 5 V
Sensor de RPM / velocidade	Digital	2	1GT101DC
Sensor de temperatura	Analógico	1	NTC M12

**Fonte:** autoria própria

Para preparar as entradas da ECU, se fez necessário conhecer um pouco melhor cada um desses sensores, a fim de entender as principais características de cada sinal. Isso foi decisivo para a escolha do microcontrolador, e também para mapear se havia necessidade que algum tipo de filtro adicional via hardware.

Posteriormente, na construção do software, esses dados também foram consultados para definir as estratégias de decodificação, calibração, e manipulação de cada sinal.

### 3.2.1.2 Conector

O conector principal da placa é o TE AMPSEAL 776180-1, representado na figura 24. Ele foi escolhido por conta da robustez, praticidade para fazer a conexão com o chicote do veículo e resistência à humidade (vedação).

**Figura 24 – Conetor TE AMPSEAL 776180-1**



**Fonte: Catálogo da TE Connectivity**

### 3.2.1.3 Microcontrolador

Após realizar o levantamento dos requisitos funcionais e não funcionais, definir os sensores, mapear as entradas analógicas e digitais, nós já temos informações suficientes para escolher um microcontrolador e definir os circuitos periféricos necessários.

Para a escolha do microcontrolador foi levado em conta alguns fatores como: quantidade de entradas e saídas analógicas e digitais, tensão de trabalho, processamento, memória, protocolos de comunicação integrados, preço, disponibilidade de compra e disponibilidade de informação.

Foi realizado um estudo comparativo entre alguns modelos distintos de microcontroladores que poderiam se adequar ao projeto:

**Quadro 5 – Comparativo entre microcontroladores (parte 1/2)**

Nome	ESP WROOM 32	Stm32F103C8T6	Stm32f427 lqfp100
Digital I/Os	34	37	82
Analog I/Os	18	10	48
Tensão lógica	2,2V ~ 3,3V	3,3V ~5 V	1.7 ~ 3.6 V
Processamento	Xtensa® Dual-Core, 32-bit LX6	Cortex-M3 ARM 32 bits	32b Arm® Cortex®-M4 MCU+FPU, 225DMIPS
Memória	4 MB Flash e 520kB SRAM	64 kB Flash e 20kB SRAM	up to 2MB Flash/256+4KB RAM
CAN	1 CAN integrada	1 CAN integrada	2 CANs integradas
Disponibilidade de info	Fácil	Médio	Difícil
Preço médio	R\$ 50,00	R\$ 45,00	R\$ 100,00
Facilidade de compra	Fácil	Médio	Difícil

**Fonte:** autoria própria**Quadro 6 – Comparativo entre microcontroladores (parte 2/2)**

Nome	NXP LPC1768	Arduino MEGA
Digital I/Os	70	54
Analog I/Os	8	16
Tensão lógica	3.3 V	5 V
Processamento	32b ARM Cortex-M3	ATmega2560 RISC com até 16 MIPS
Memória	up to 512 kB flash e 64 kB SRAM	256 KB Flash e 8kB SRAM
CAN	2 CANs integradas	Não possui CAN integrada
Disponibilidade de info	Difícil	Fácil
Preço médio	R\$ 42,00	R\$ 110,00
Facilidade de compra	Difícil	Fácil

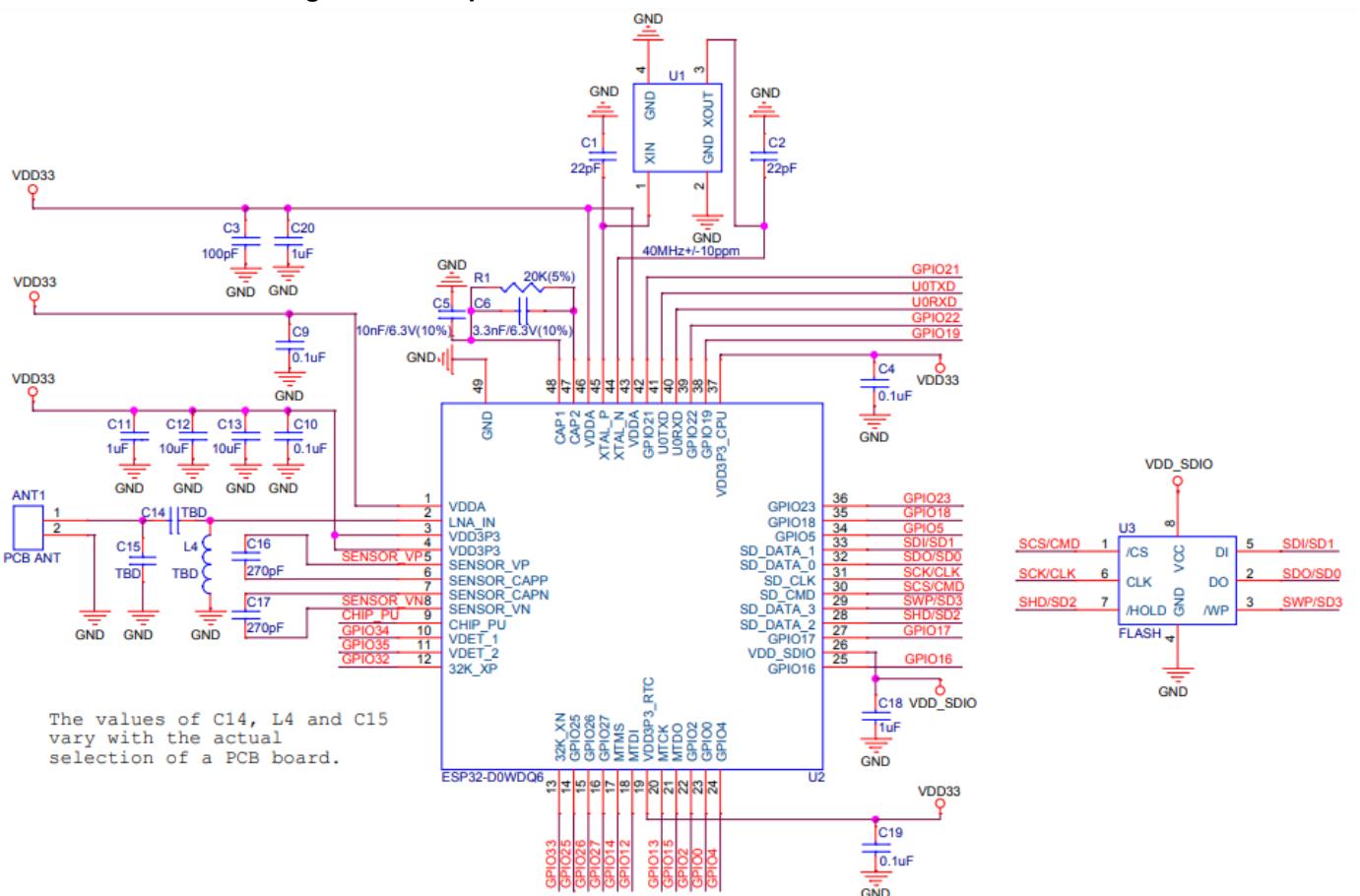
**Fonte:** autoria própria

Considerando que o ESP WROOM 32 atinge os requisitos mínimos de hardware (I/Os, rede CAN, memória), possui um processamento muito bom, se destaca no quesito memória, e tem uma acessibilidade muito favorável (preço acessível, encontrado facilmente em lojas nacionais e muita informação disseminada na internet), ele foi a escolha para a aplicação da ECU.

A proposta será de incluir o módulo ESP32-WROOM-32 diretamente na PCB do projeto, juntamente com um circuito *standalone* (círcuito básico para funcionamento do microcontrolador, composto apenas pelos periféricos necessários) que garante o acesso às I/Os do microcontrolador bem como os pinos de comunicação. O módulo consiste num circuito que liga o chip do microcontrolador com alguns circuitos periféricos básicos, definidos pelo fabricante, encapsulados numa estrutura metálica.

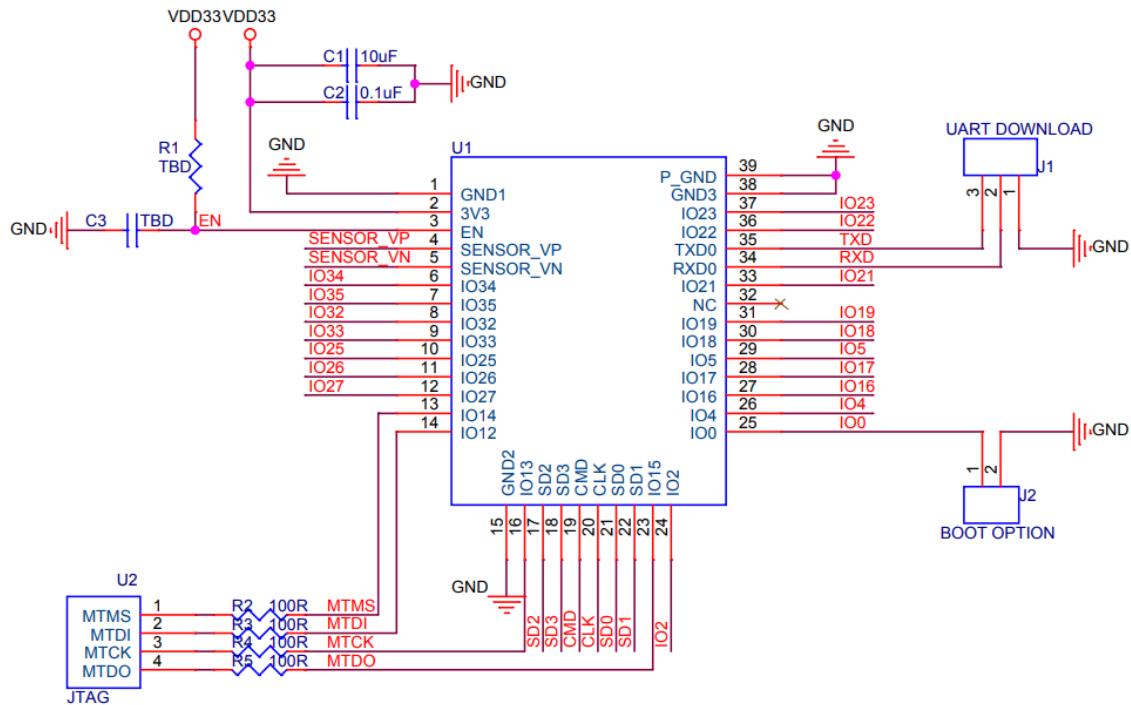
Nas figuras 25, 26 e 27 pode-se verificar a circuitaria interna do módulo ESP32-WROOM-32 fornecida pelo *datasheet* do componente e, de semelhante modo, o *pinout* externo ao módulo (o que será de fato utilizado no esquemático da ECU).

**Figura 25 – Esquemático interno ao módulo ESP32-WROOM-32**



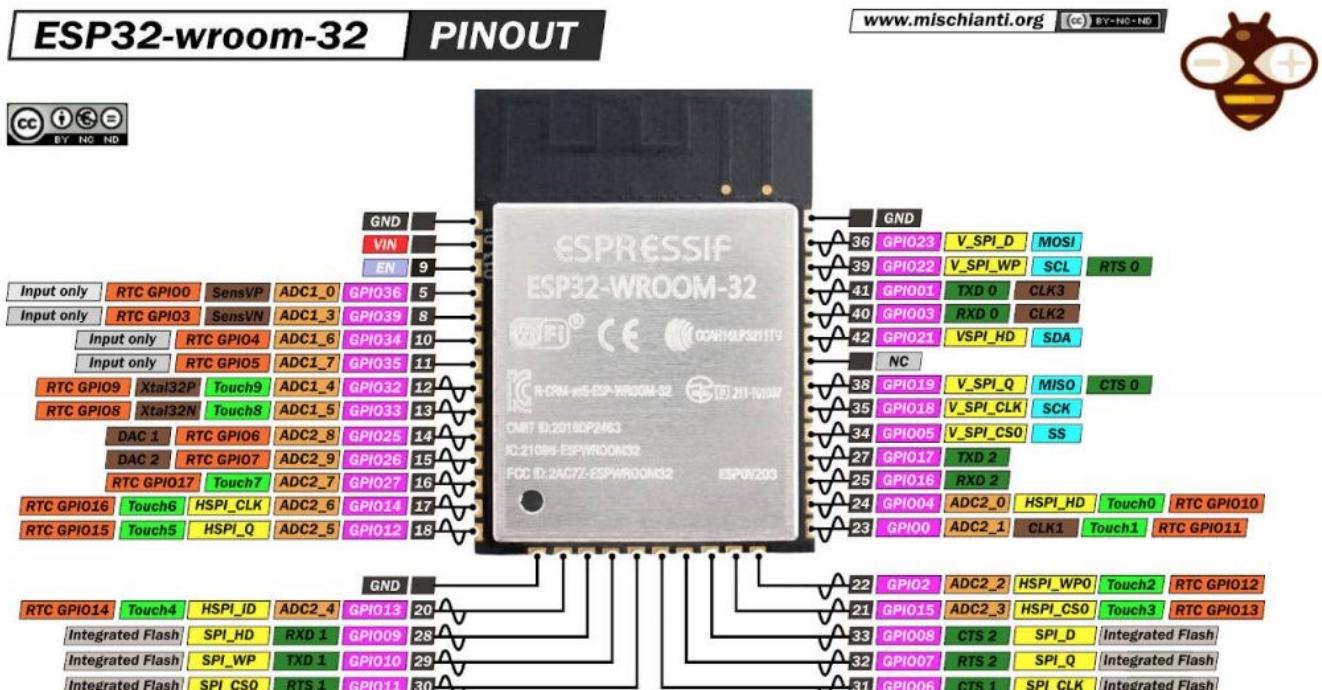
**Fonte:** [esp32-wroom-32 datasheet](#)

**Figura 26 – Pinout e periféricos externos ao módulo ESP32-WROOM-32**



**Fonte:** esp32-wroom-32 datasheet

**Figura 27 – ESP32-WROOM-32 pinout e funcionalidades dos pinos**

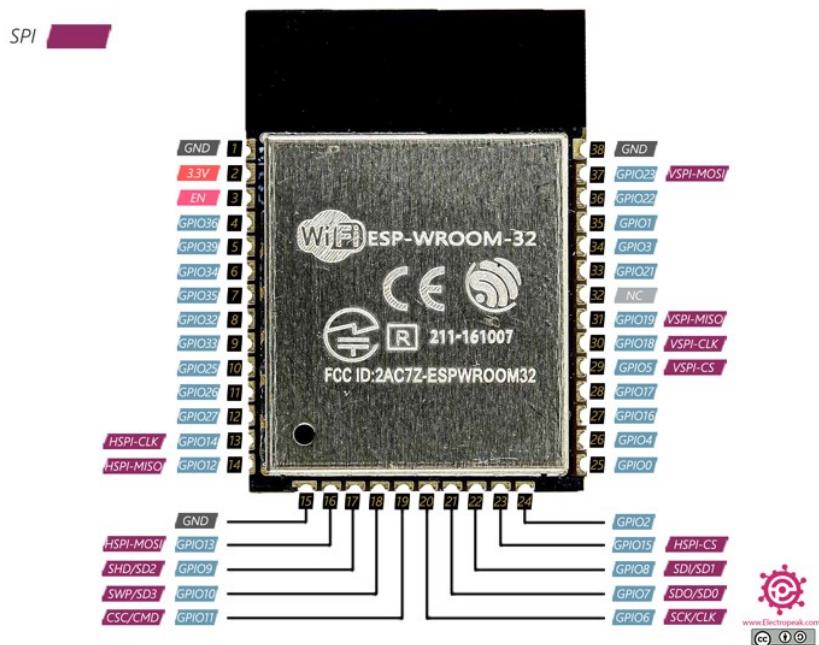


**Fonte:** [mischianti.org](http://mischianti.org) - [link](#)

### 3.2.1.4 Rede CAN

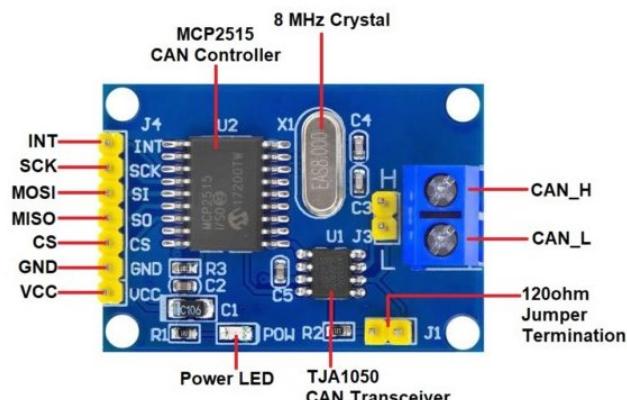
O projeto tem como um dos requisitos a possibilidade de comunicação CAN com dois barramentos distintos. Porém, o ESP32 não possui protocolo CAN integrado. Para viabilizar a comunicação CAN, foram utilizados os pinos de protocolo SPI (*Serial Peripheral Interface*) do microcontrolador, para estabelecer comunicação com módulos *tranceivers* CAN MCP2515, como pode-se observar nas figuras 22, 23 e 24:

**Figura 28 – Pinos de protocolo SPI do ESP-32**



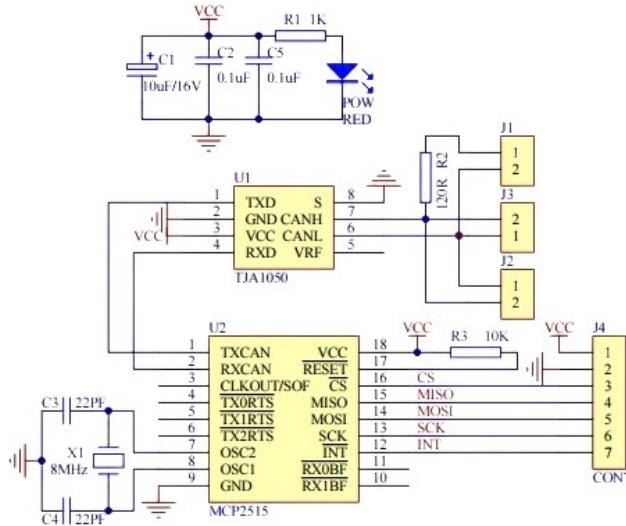
Fonte: electropeak.com

**Figura 29 – Módulo MCP2515 utilizado para converter protocolo SPI para rede CAN**



Fonte: tronisoft.com

**Figura 30 – Circuito referente às conexões do módulo MCP2515**



Fonte: [tronisoft.com](http://tronisoft.com)

### 3.2.2 Esquemático da placa

Para unificar todos os circuitos necessários, decidiu-se elaborar uma placa de circuito impresso, de modo que a ECU ficasse mais compacta e robusta devido à redução de conexões por cabos entre os módulos.

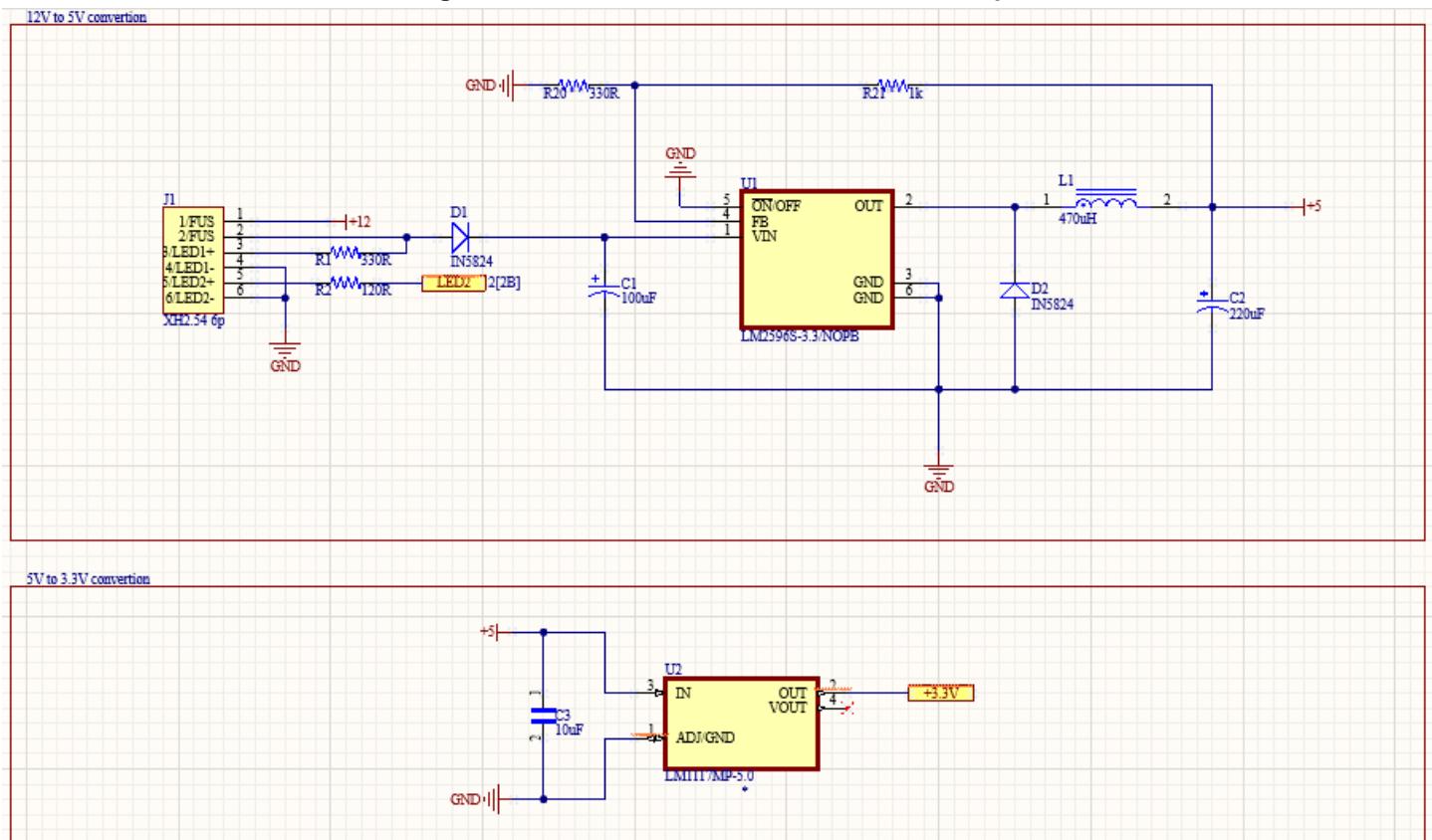
Para o desenvolvimento da placa, foi utilizado o software Altium Designer Professional (versão 23.7.1).

Com base nas pesquisas apresentadas, foi elaborado um esquemático completo das soluções estudadas, que pode ser organizado nos seguintes tópicos:

- Alimentação e Conversão de 12 V para 5 V
- Conversão de 5 V para 12 V
- *ESP32 Standalone*
- Tratamento de entradas e saídas analógicas e digitais
- Rede CAN
- Pinout do conector principal

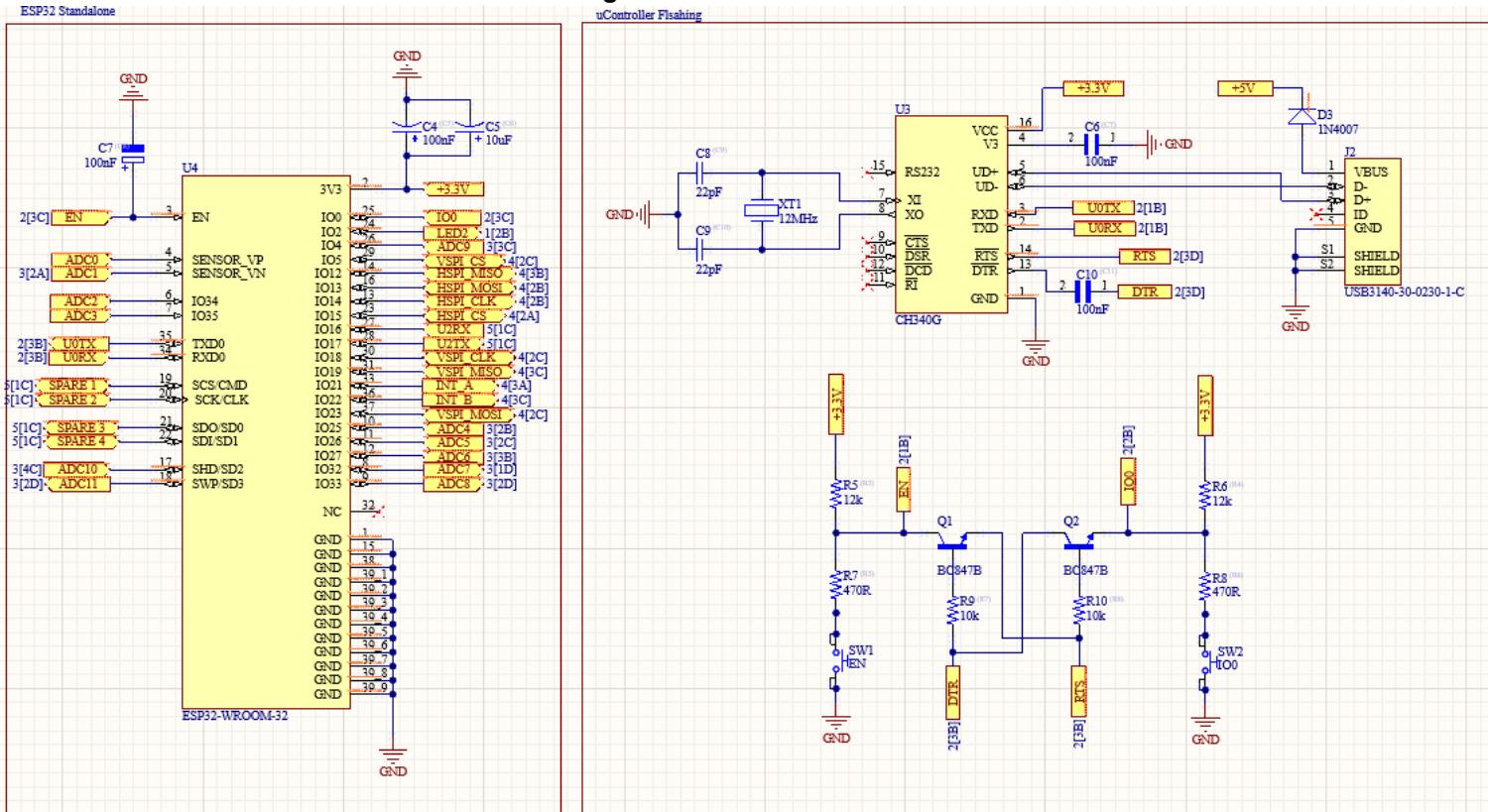
O esquemático completo pode ser observado nas figuras 31 a 36:

**Figura 31 – Alimentação e Conversão de 12 V para 5 V**



Fonte: autoria própria

**Figura 32 – ESP32 Standalone**



Fonte: autoria própria

Para os sinais analógicos, foi adicionado um filtro passa baixa para diminuir os ruídos na leitura do sinal.

Cálculo do filtro:

$$F_c = \frac{1}{2\pi RC}$$

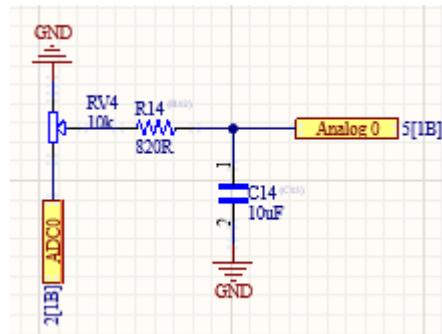
Adotando uma frequência de 20Hz e um capacitor de 10uF (valores arbitrados), obtém-se:

$$R = \frac{1}{2\pi CF_c} \approx 795\Omega$$

O valor comercial aproximado para R seria de 820 ohms.

Confira o circuito na imagem a seguir:

**Figura 33 – Filtro passa baixa para os sinais analógicos**

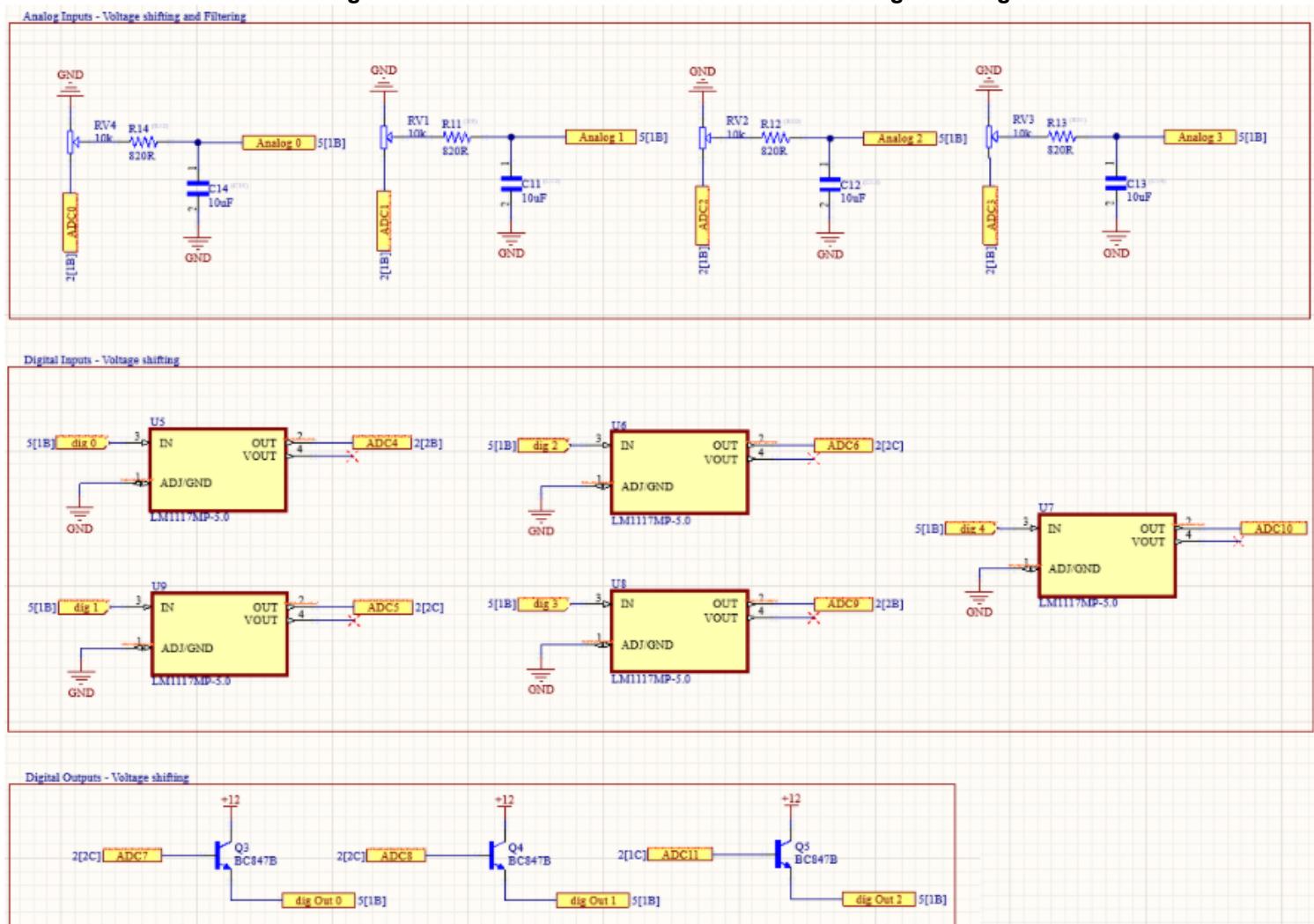


**Fonte: autoria própria**

Neste exemplo acima, “Analog 0” é o sinal não filtrado (pode ser 12 V ou 5 V). E o “ADC0” é o sinal com tensão regulada (3v3) e filtrado (20Hz).

Na figura 34 pode-se observar as conexões de todas as entradas analógicas e digitais da ECU. As entradas analógicas contém o filtro demonstrado anteriormente, as entradas digitais tem uma conversão para 3.3 V e as saídas digitais são convertidas para 12 V através de transistores.

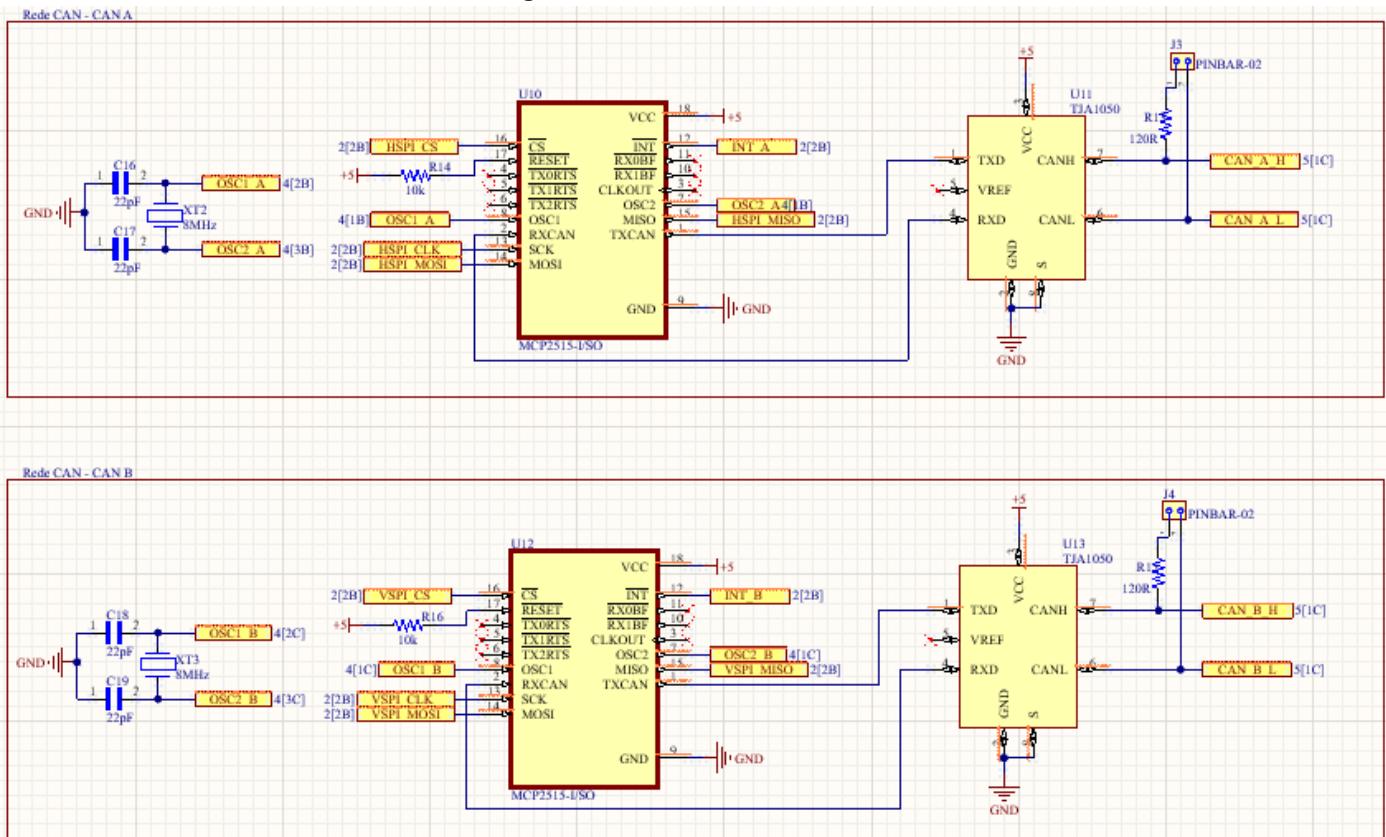
**Figura 34 – Tratamento de entradas e saídas analógicas e digitais**



Fonte: autoria própria

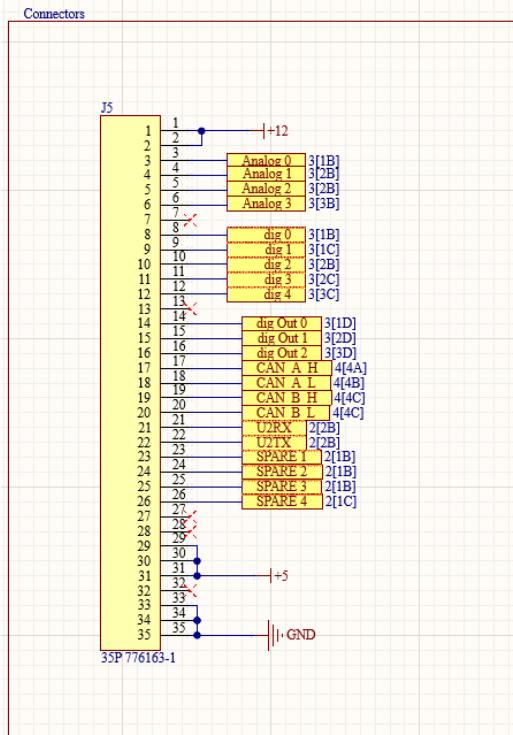
Na figura 35, pode-se observar as conexões necessárias para fazer a conversão de protocolo SPI para CAN, visto que o microcontrolador não possui CAN integrada. O circuito foi duplicado para possibilitar a utilização de diferentes *baudrates* em paralelo.

**Figura 35 – Rede CAN de dois canais**



Fonte: autoria própria

**Figura 36 – Pinout do conector principal**

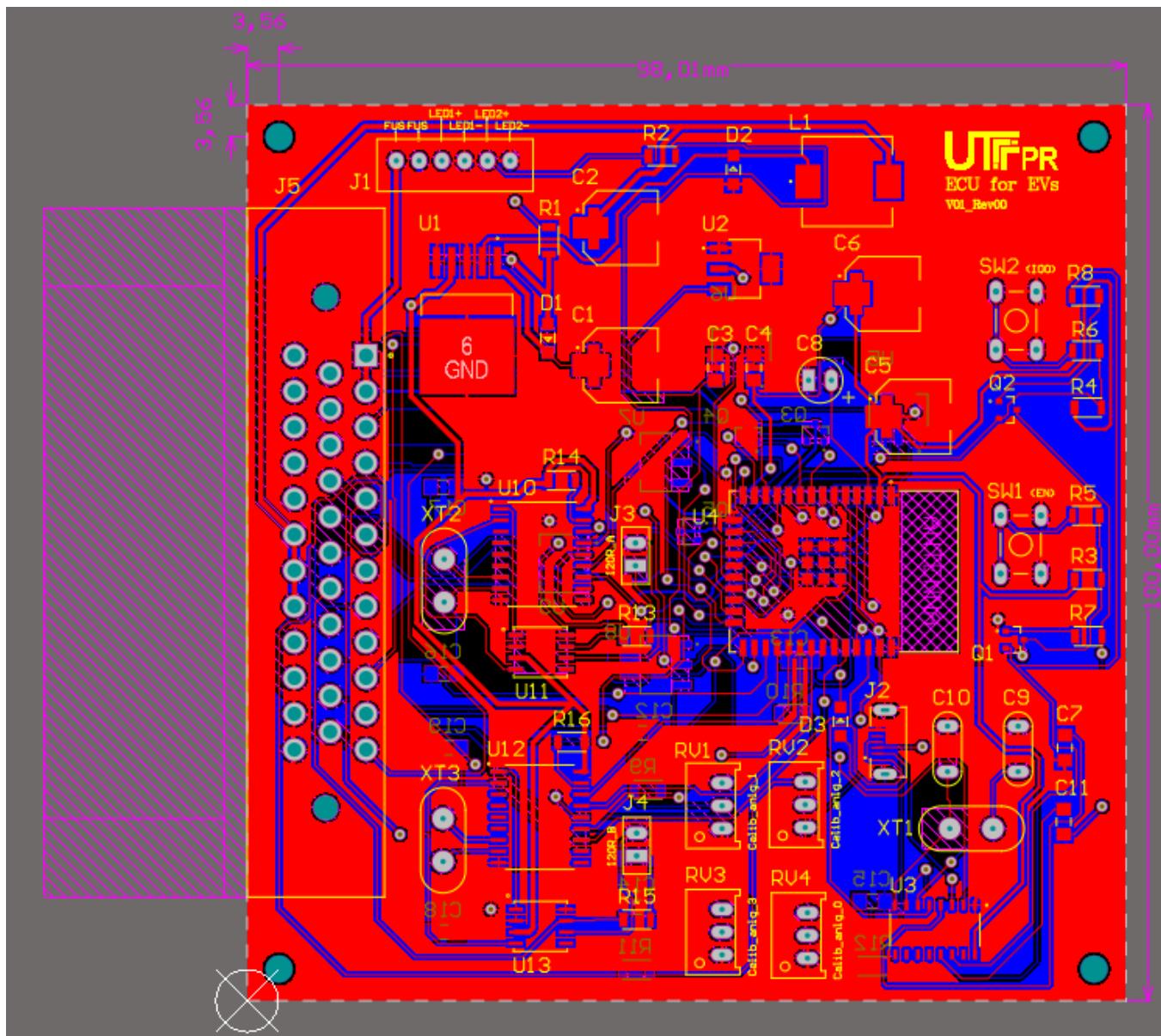


Fonte: autoria própria

### 3.2.3 Roteamento da placa

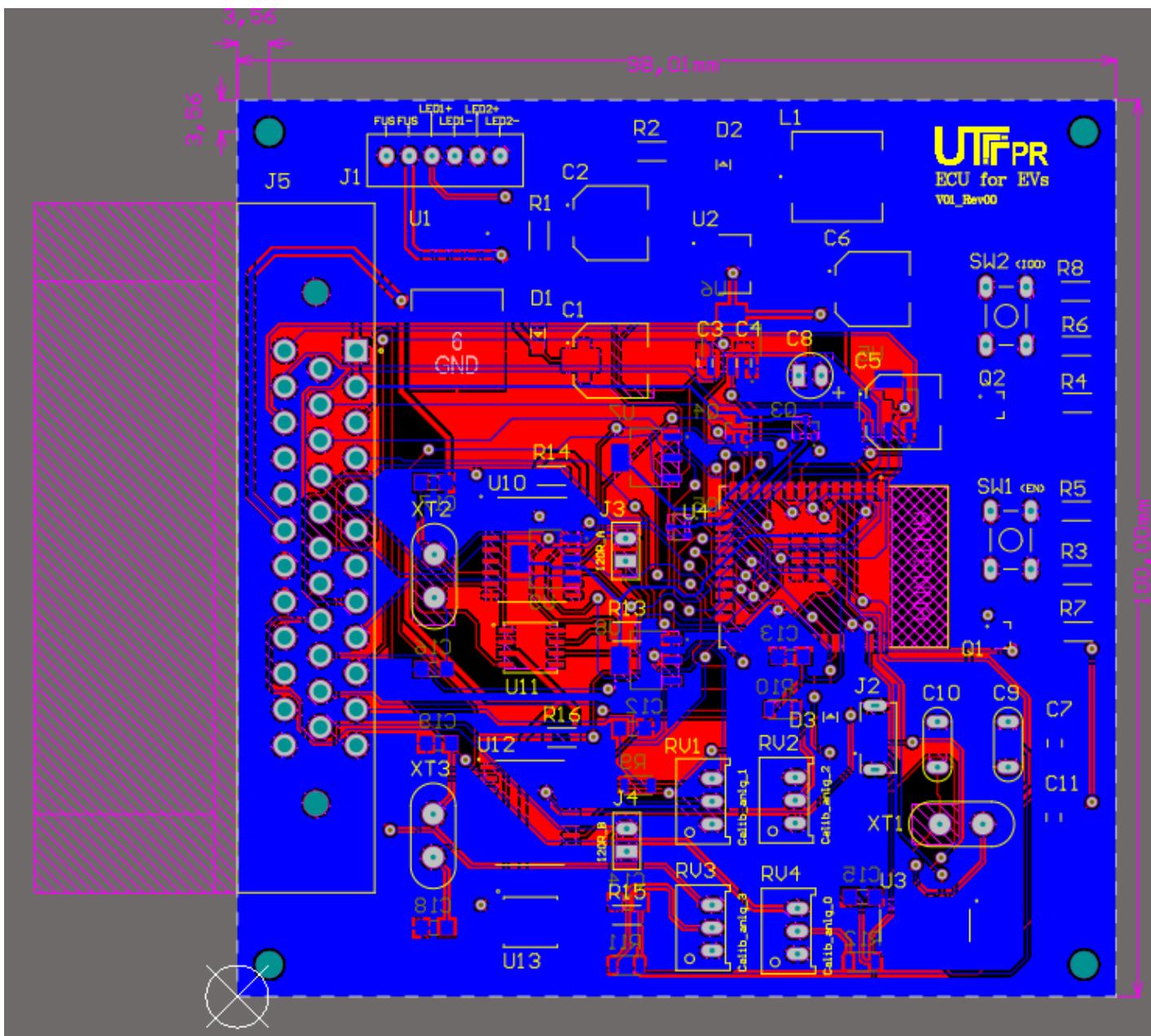
A placa foi roteada no Altium Designer, a partir do esquemático supracitado no trabalho. Foram utilizadas duas camadas, com furos metalizados, para conseguir uma placa mais compacta e com aspecto profissional. Veja o resultado nas figuras 37, 38, 39 e 40.

Figura 37 – Roteamento da Placa – Camada superior



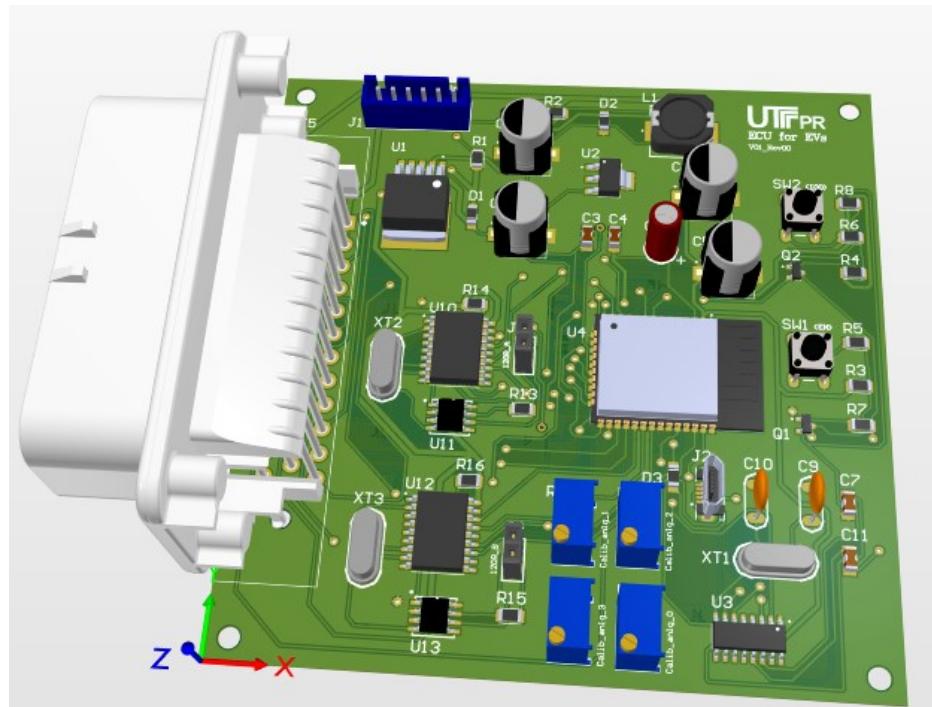
Fonte: autoria própria

**Figura 38 – Roteamento da Placa – Camada inferior**



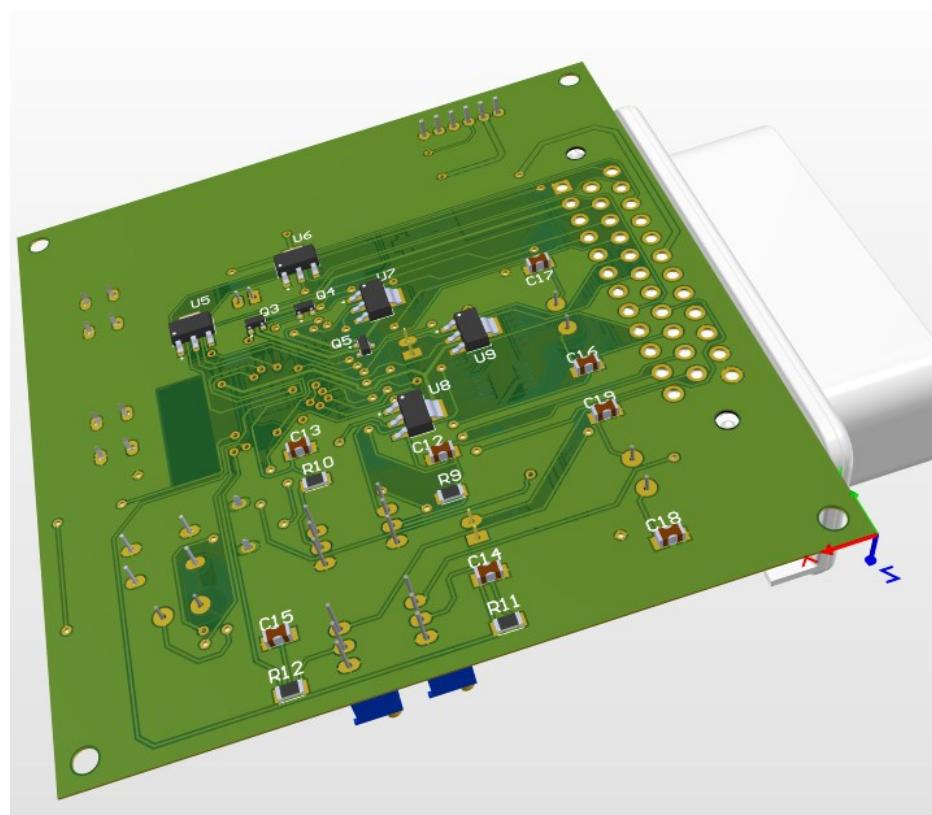
**Fonte:** autoria própria

Figura 39 – Roteamento da Placa – Modelo 3D superior



Fonte: autoria própria

Figura 40 – Roteamento da Placa – Modelo 3D superior

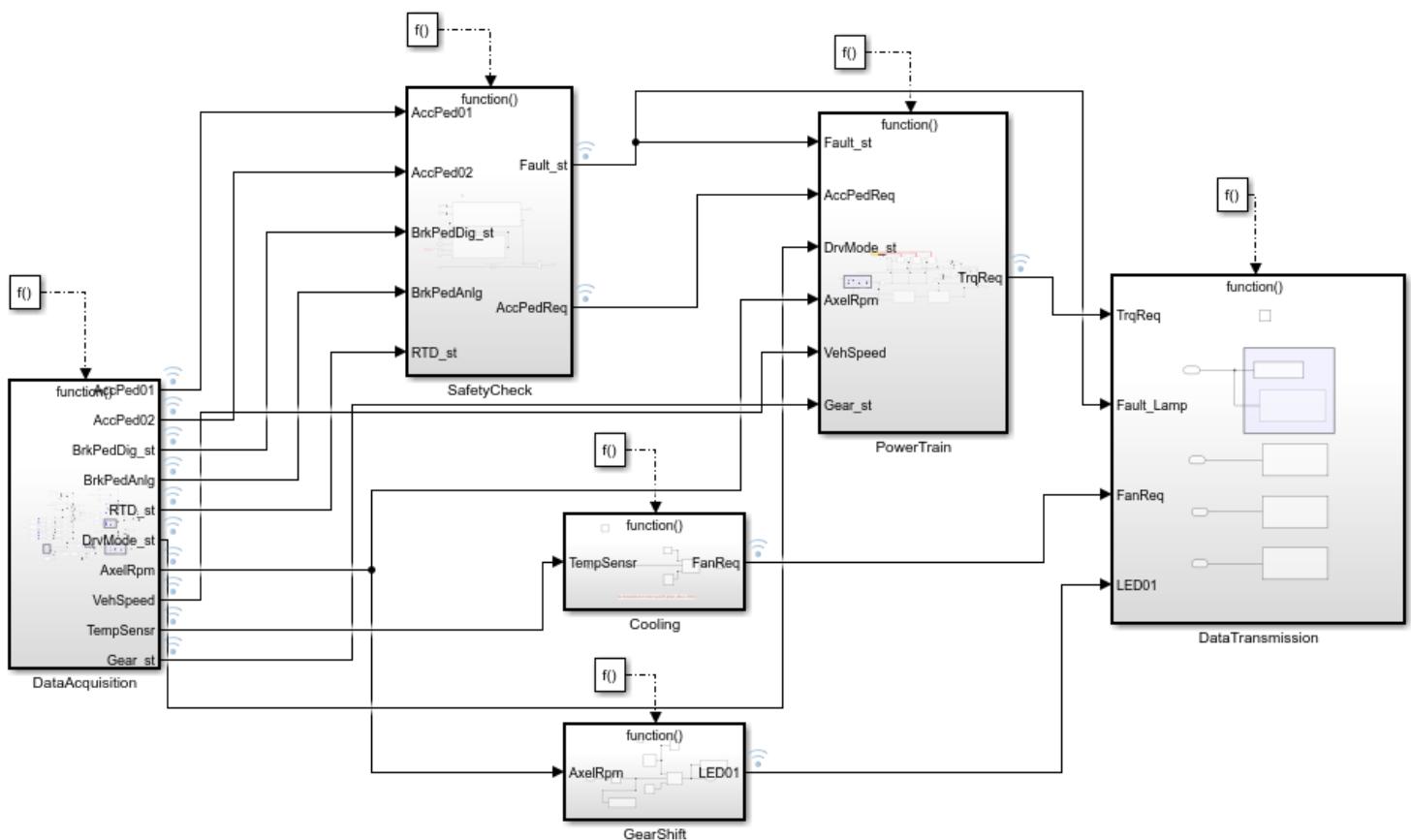


Fonte: autoria própria

### 3.3 Desenvolvimento do Software

Baseando-se nos requisitos levantados no início do projeto e nas referências de software estudadas durante todo o período de desenvolvimento, foi elaborado um software que, além de demonstrar como pode-se fazer o setup dos pinos da placa, traz exemplos de aplicações como: Checagens de segurança, aviso para troca de marcha, ativação do sistema de resfriamento e cálculo do torque requisitado com aplicação de diferentes modos de direção e controle de tração. Veja a arquitetura geral na figura 41:

**Figura 41 – Visão geral do sistema desenvolvido para a ECU**

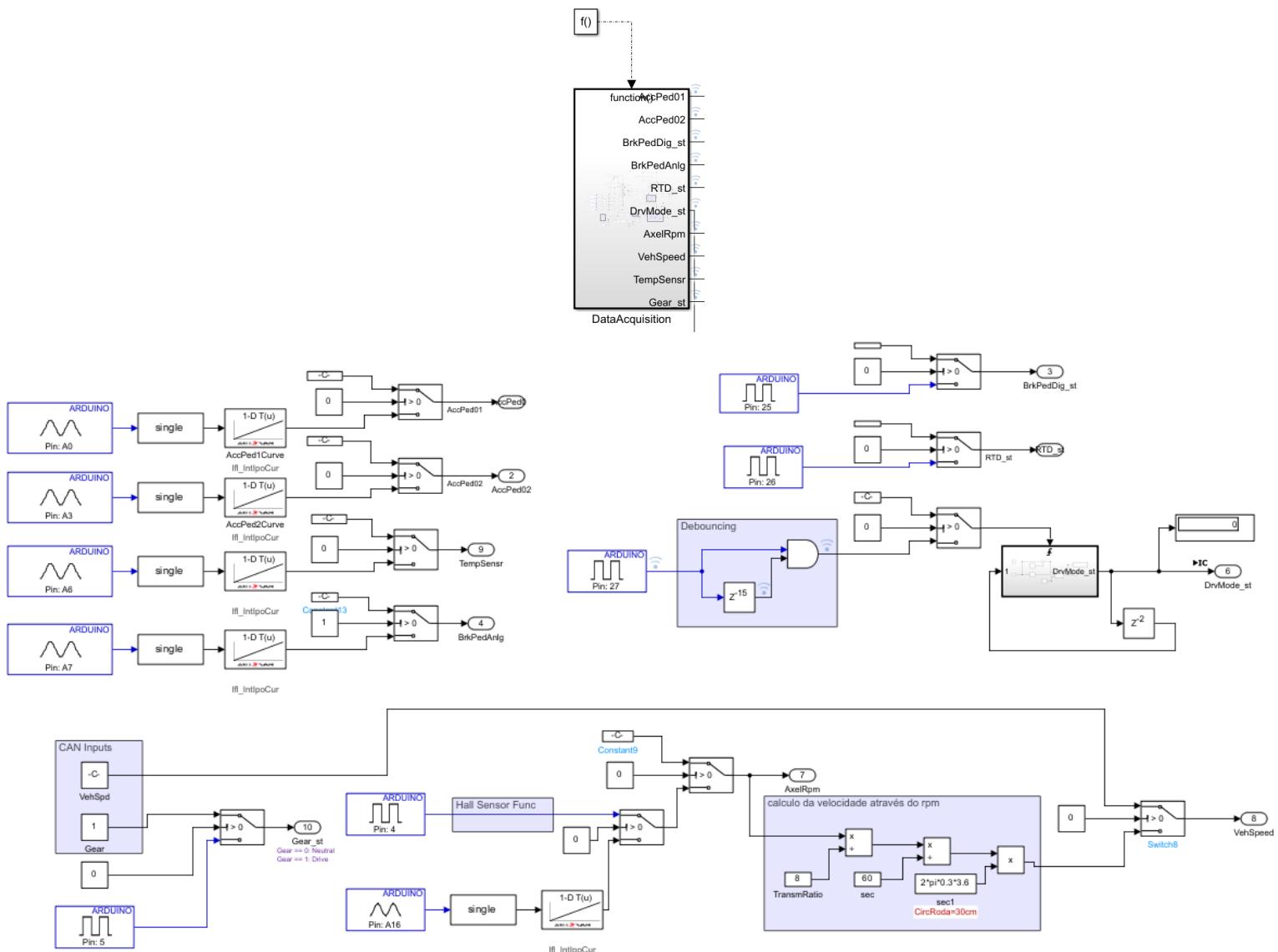


**Fonte: autoria própria**

### 3.3.1 Aquisição de Dados

A primeira função do sistema nomeada “*DataAcquisition*” é onde ocorre a entrada dos sinais através dos pinos da ECU.

**Figura 42 – Função de aquisição de dados**



**Fonte:** autoria própria

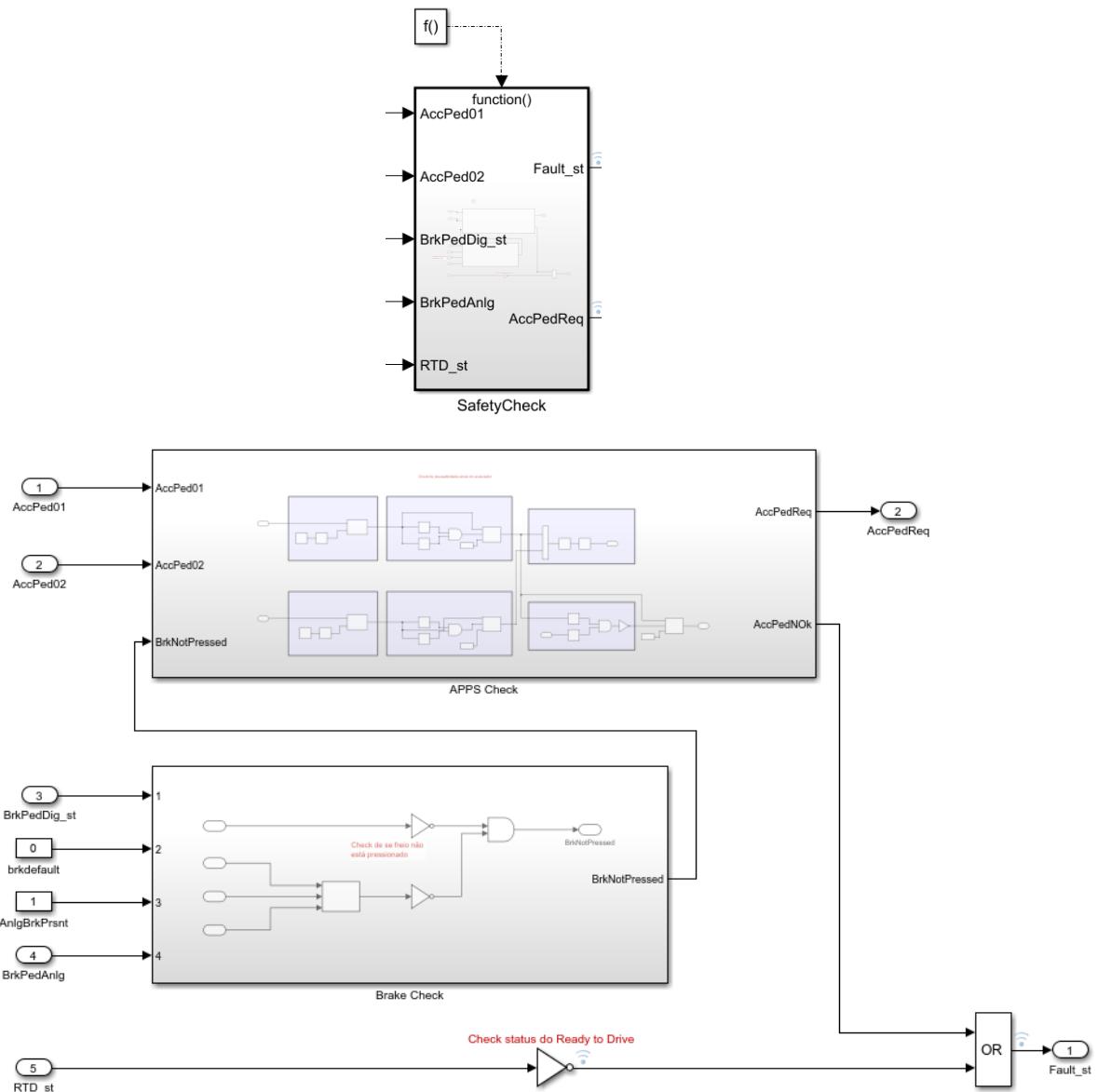
Nesta função os sinais analógicos passam por uma conversão para serem visualizados em porcentagem. A velocidade do veículo pode ser calculada através do sinal de rpm ou ser lido através da CAN. O sinal digital do seletor de modo de direção passa por um tratamento (*debouncing* e lógica de incremento).

Após essas tratativas, os sinais são exportados para as demais funções do Software.

### 3.3.2 Checagem de Segurança

Na função “SafetyCheck“ é feita a verificação dos sinais de pedais e “Ready To Drive“ (um sinal de segurança obrigatório na competição de Formula SAE):

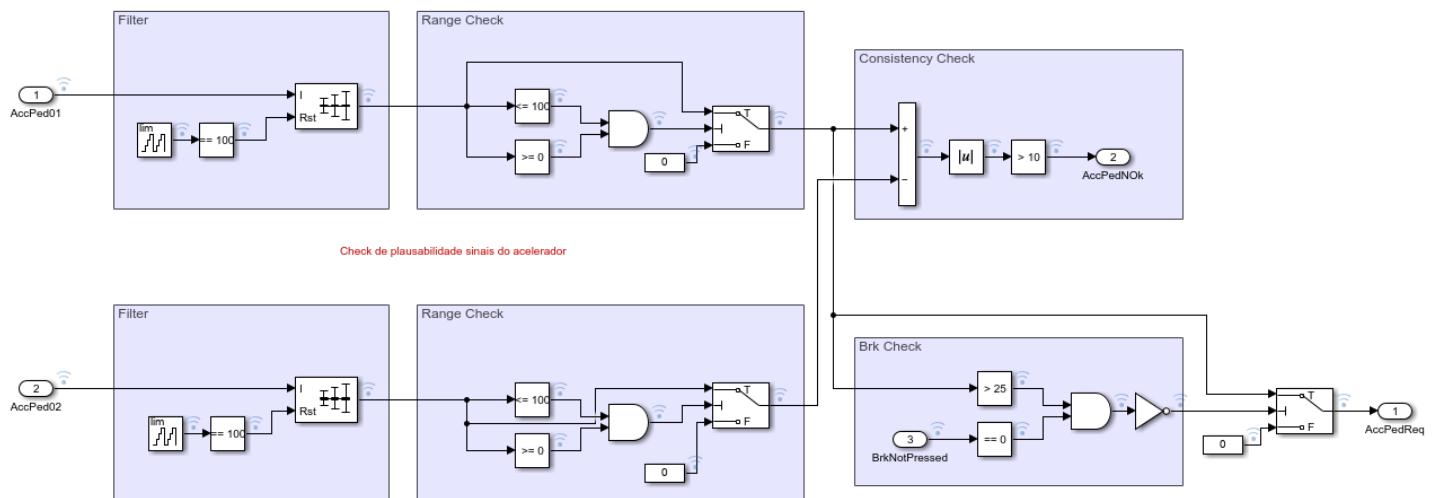
**Figura 43 – Função de segurança**



**Fonte: autoria própria**

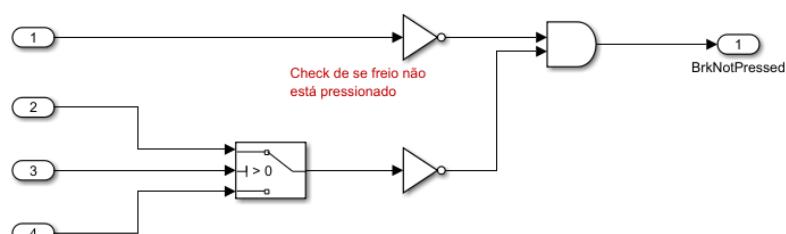
Dentro desta função de *Safety*, existem duas outras sub-funções. A “*Brake Check*“ que monitora se algum dos sinais de freio está acionado e exporta essa informação para a “*APPS Check*“ que, por sua vez, filtra os sinais do acelerador, confere o range e a consistência entre os sinais e, caso o freio não esteja acionado, libera o sinal do pedal para utilização no cálculo de torque.

**Figura 44 – APPS Check**



**Fonte:** autoria própria

**Figura 45 – Brake Check**

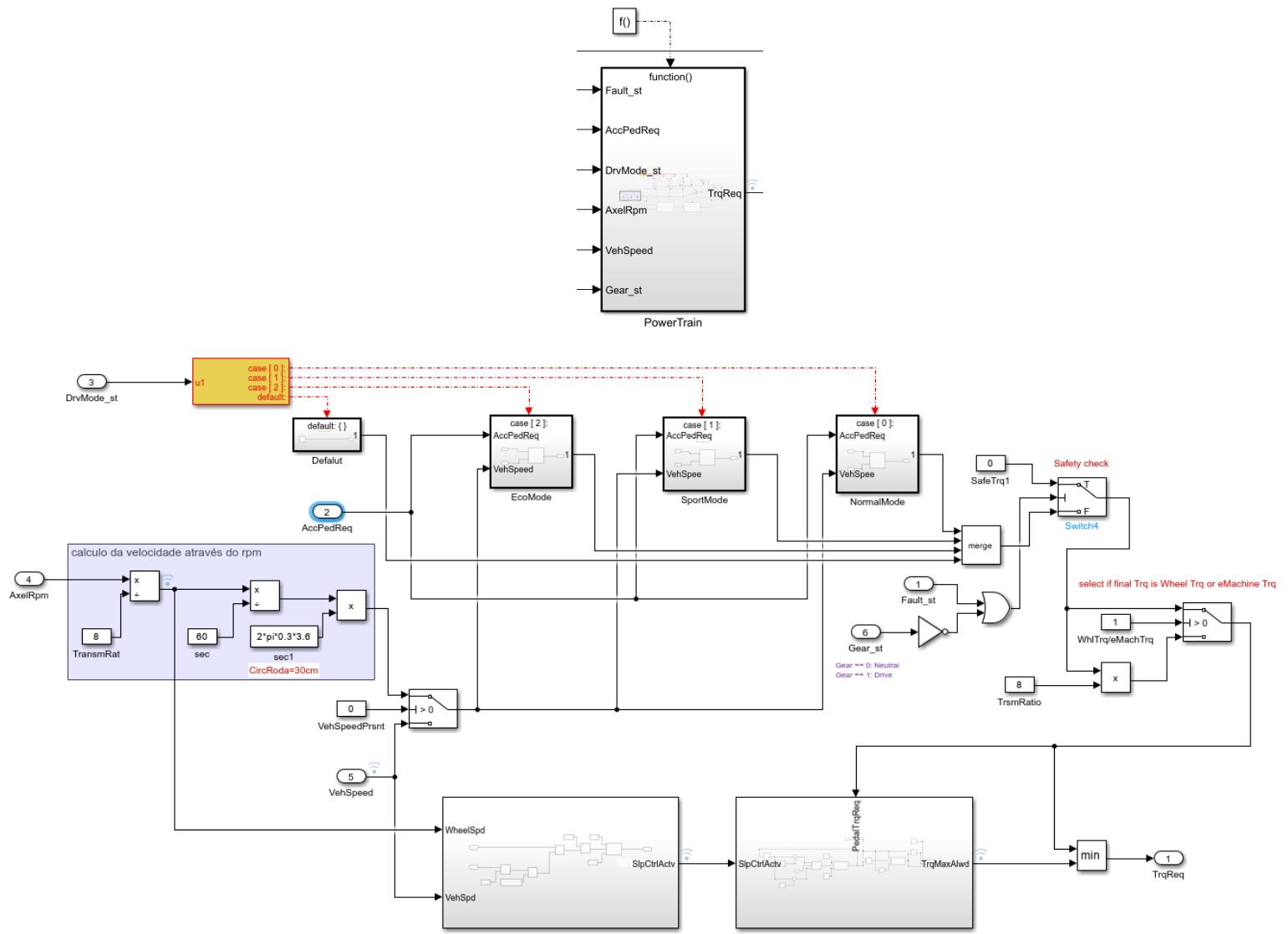


**Fonte:** autoria própria

### 3.3.3 Powertrain

Após a checagem de segurança, o sinal do acelerador é mais confiável e pode então ser utilizado para o cálculo do torque. Esse cálculo é feito dentro da função “PowerTrain”, utilizando não somente o pedal do acelerador, mas também: velocidade do veículo, rpm, modo de direção e marcha.

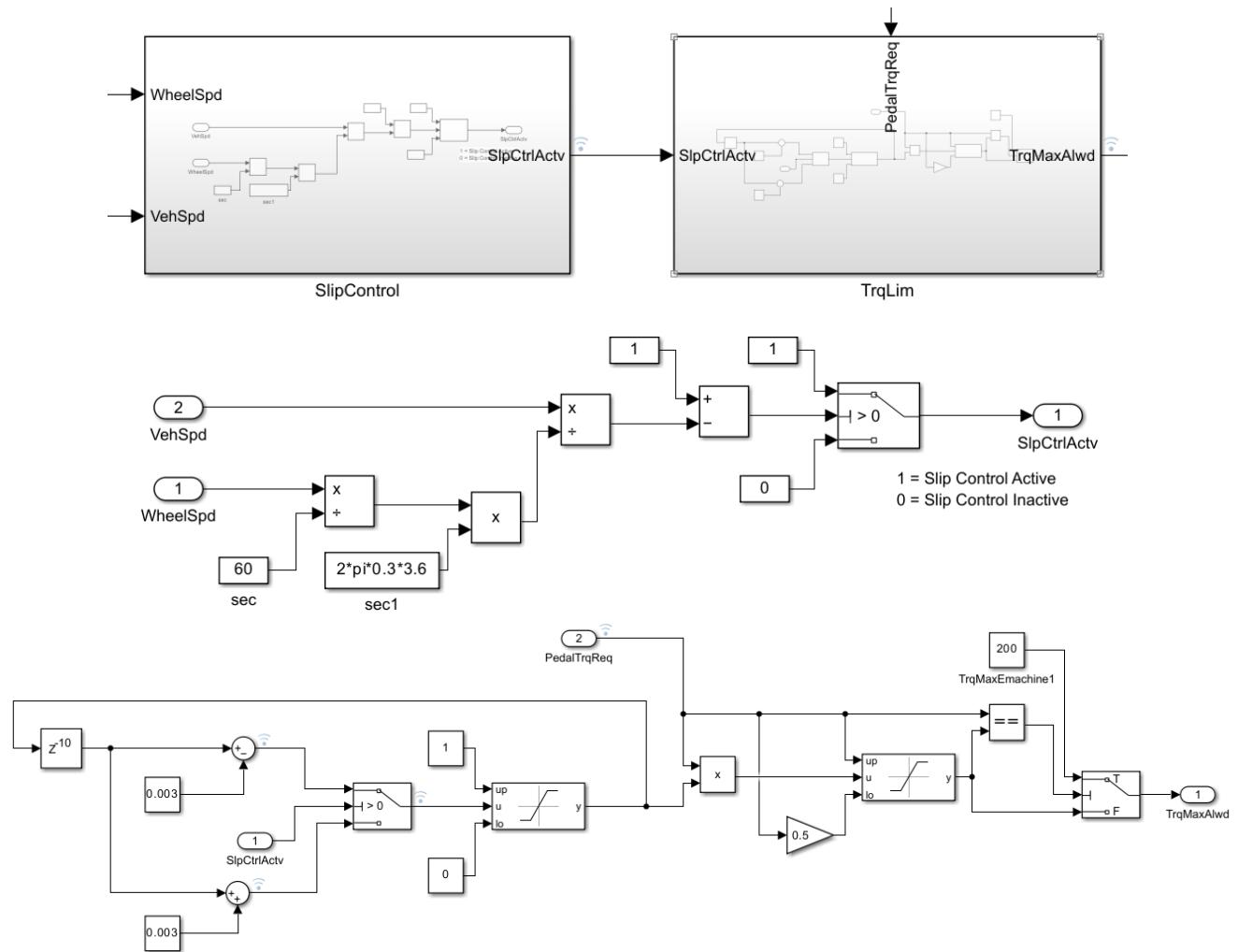
**Figura 46 – PowerTrain**



**Fonte: autoria própria**

Nesta função, existem três mapas calibráveis de toque para que a dinâmica do veículo seja diferente de acordo com o modo de direção escolhido: “*EcoMode*”, “*SportMode*” ou “*NormalMode*”. Após selecionar o mapa de torque, a função confere se o veículo não está em Neutro ou se há alguma falha ativa. Se o caminho de torque estiver sem erros, o torque segue a diante, passando por uma limitação que pode ser um valor calibrável ou definido pela estratégia de controle de tração (subfunções “*SlipControl*” e “*TrqLim*”).

**Figura 47 – Controle de Tração**



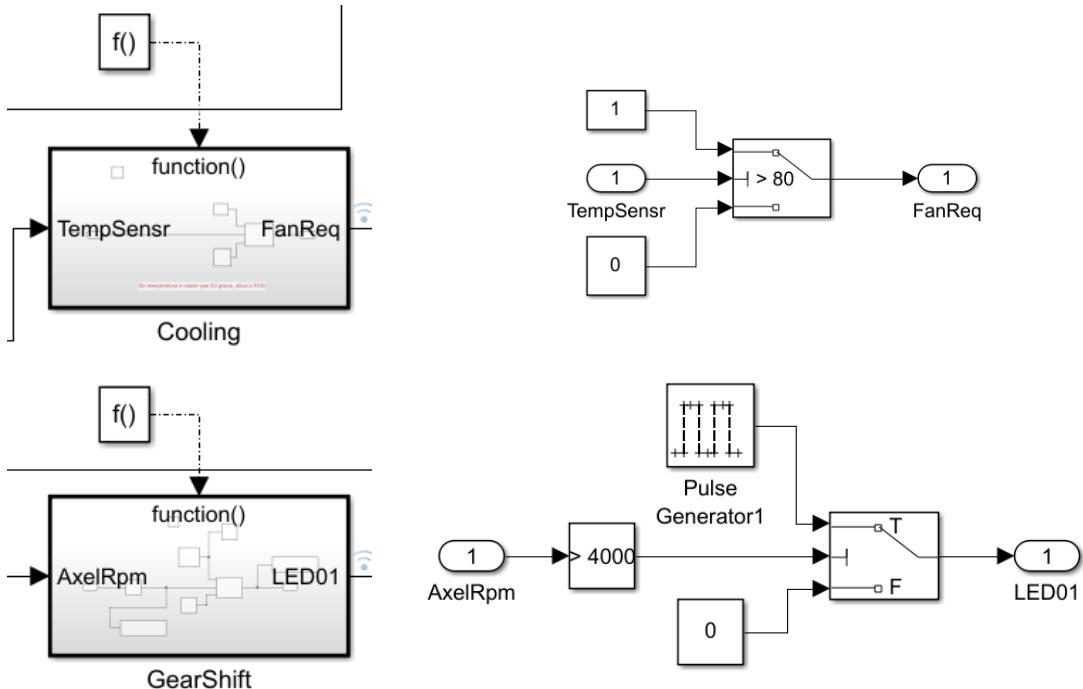
**Fonte:** autoria própria

A lógica do controle de tração se baseia em comparar a velocidade da roda com a velocidade linear do veículo. Caso a velocidade da roda esteja acima do esperado, ativa-se uma limitação de torque (em rampa para evitar solavancos) até que o veículo recupere a tração e as velocidades sejam compatíveis.

### 3.3.4 Funções Auxiliares

Há ainda duas funções menores, para controle da refrigeração e para indicar o momento em que o motorista deve trocar a marcha:

**Figura 48 – Refrigeração e Troca de Marcha**



**Fonte:** autoria própria

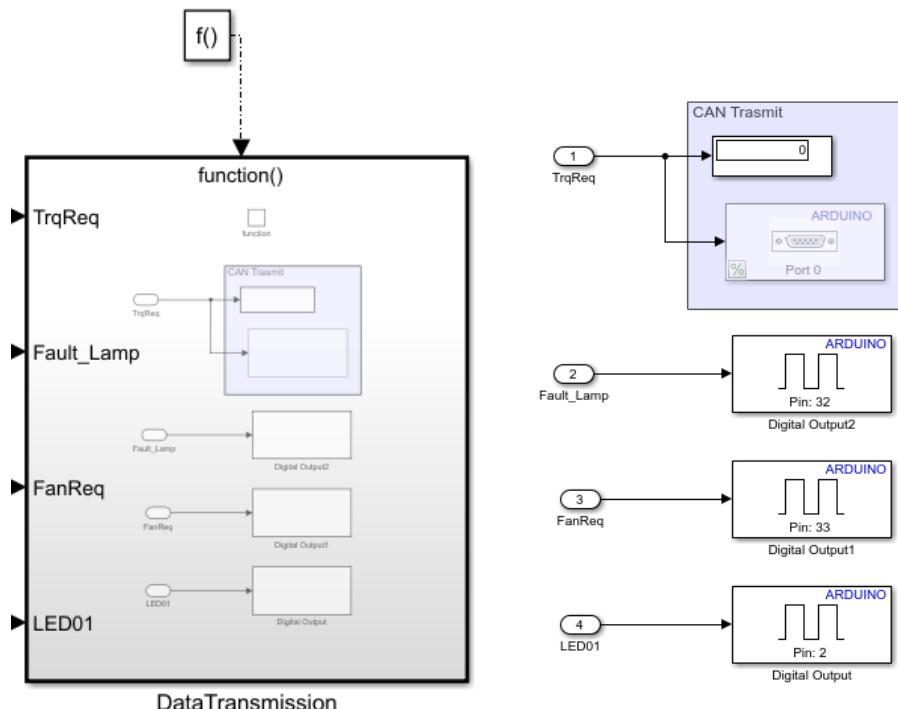
Quando o sensor de temperatura passa de um valor calibrável, é enviado um sinal digital “HIGH” que pode ser utilizado para, através de um relé, ativar uma ventoinha.

Quando o rpm passa de um valor calibrável, é gerado um sinal pulsante que fará um LED piscar rapidamente na cabine do motorista, indicando a necessidade da troca de marcha.

### 3.3.5 Saídas do sistema

Ao final de toda a cadeia do software, a função “*DataTransmission*“ faz o envio dos sinais digitais e CAN, para requisitar o torque ao inversor de frequência.

**Figura 49 – Saídas digitais e CAN**



**Fonte: autoria própria**

## 4 TESTES e VALIDAÇÕES

Foram realizados testes de software através de simulação, testes puramente de hardware (durante e após a montagem da placa) e, por fim testes em bancada, com o firmware rodando no microcontrolador.

Foi retirado do escopo do projeto os testes em veículo, pois o carro não estava disponível para rodagem no período em que a ECU ficou pronta e a previsão para disponibilidade do veículo era incerta.

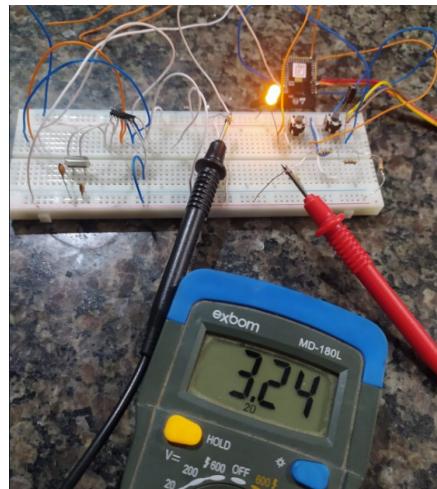
### 4.1 Validação do Hardware

Antes de fabricar a placa de circuito impresso, foram realizadas simulações dos circuitos principais, bem como a montagem parcial dos circuitos em uma *protoboard*.

Durante esse processo, foram identificados alguns erros de projeto, como por exemplo a conexão errada do pino 4 de todos os Cls LM1117 presentes no projeto (estavam ligados no GND, quando na verdade deveriam ser N.C). Outro erro identificado nesta fase, foi a falta de um divisor de tensão para gerar um sinal de feedback no regulador de tensão LM2596.

Os testes em *protoboard* tiveram uma certa limitação, visto que grande parte dos componentes utilizados no projeto eram SMD, e nem todos tinham versão PTH para facilitar a montagem em bancada.

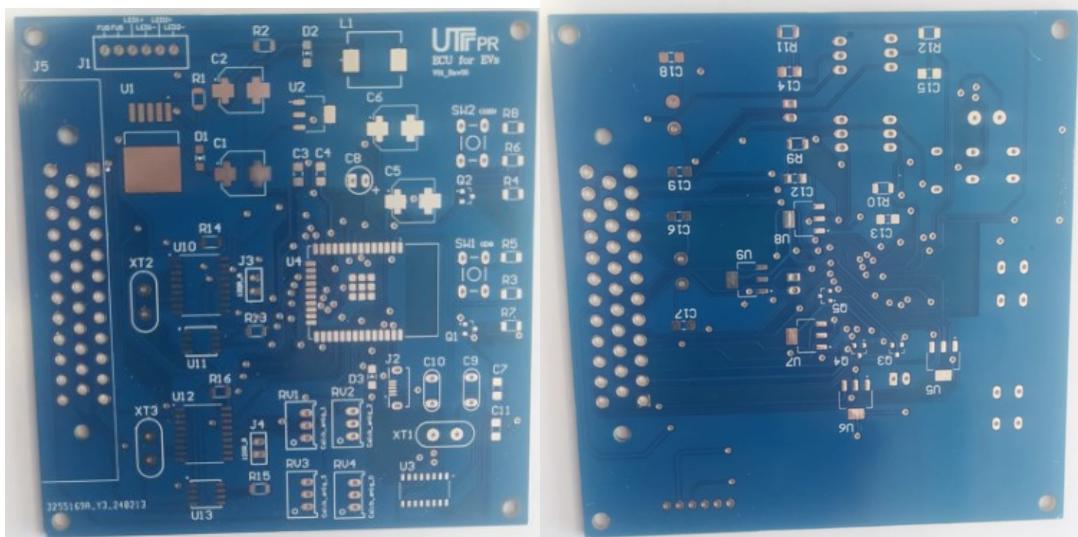
**Figura 50 – Primeiros testes de hardware realizados com protoboard**



Fonte: autoria própria

Após essa validação inicial, o projeto da PCB foi enviado para fabricação em uma empresa chinesa:

**Figura 51 – PCB fabricada através de processos industriais**

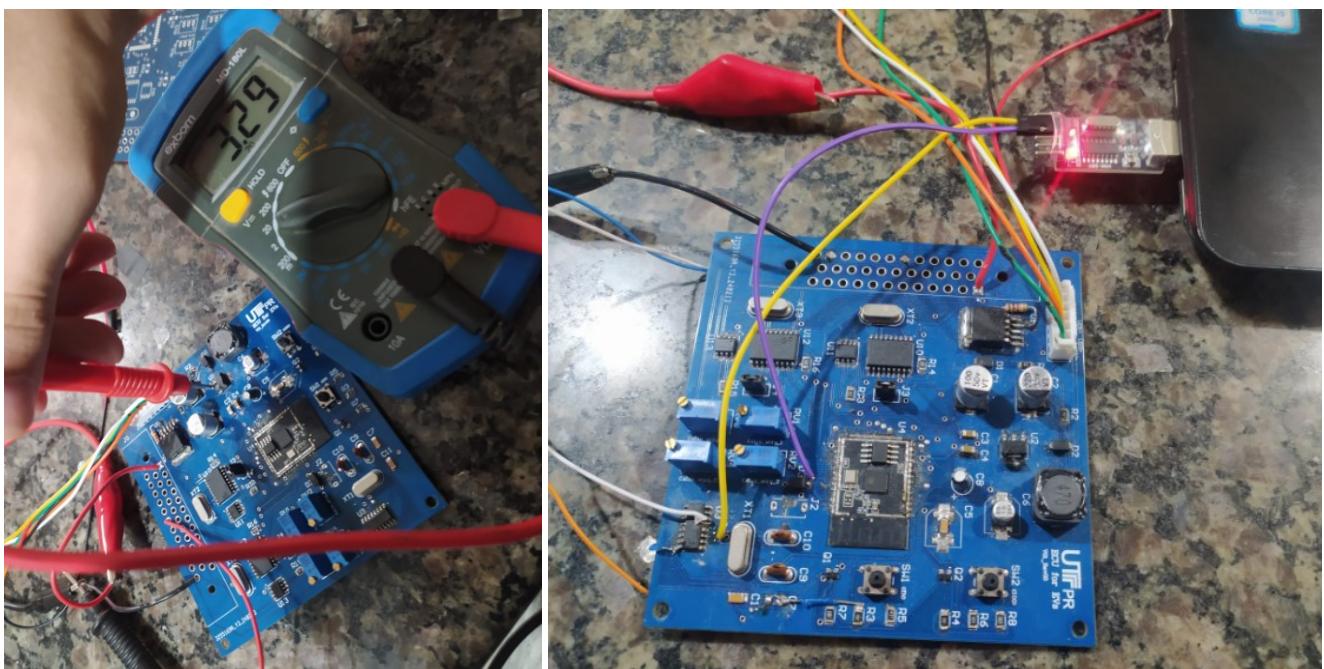


**Fonte:** autoria própria

A soldagem dos componentes foi realizada manualmente em laboratório.

Para reduzir o risco de queima de componentes, a montagem da PCB foi feita de maneira gradativa, iniciando pelo circuito de regulação de tensão e, após validação deste circuito, foi-se acrescentando e validando os demais componentes.

**Figura 52 – Montagem e teste da placa**



**Fonte: autoria própria**

Para a validação do circuito *standalone* do ESP32, neste primeiro momento, foi utilizando um código simples para inicialização das portas analógicas, digitais e envio de mensagem serial através do Arduino IDE 2.3.2.

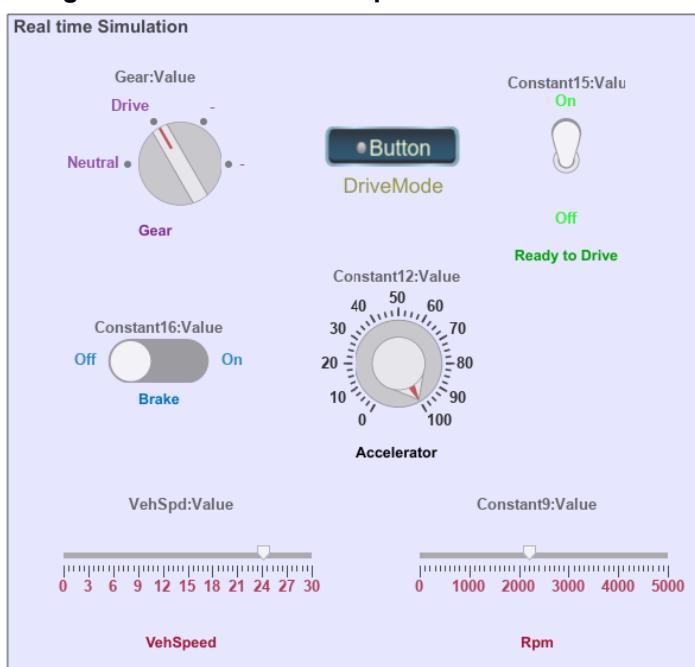
Durante os testes, foram identificados mais dois erros de projeto, relacionados ao desenho do esquemático no Altium. Em certo ponto do esquemático, foram utilizadas duas nomenclaturas diferentes para a *Label* de 5 V, o que gerou uma descontinuidade na trilha da PCB, impossibilitando que o regulador de 3.3 V recebesse o input de 5 V. Este problema foi corrigido adicionando um jumper na parte inferior da placa. O outro erro detectado, foi a inversão das *label/s* de entrada analógica antes de depois do filtro passa-baixa. Esse problema também foi resolvido interrompendo algumas trilhas e adicionando jumpers na parte inferior da placa.

## 4.2 Validação do Software

Para a validação da lógica e codificação do software, foram elaborados 6 casos de teste específicos à serem seguidos.

Para auxiliar na manipulação dos sinais de entrada do sistema, foi criado um painel no Simulink, representado na figura 53:

**Figura 53 – Painel auxiliar para testes no Simulink**



Fonte: autoria própria

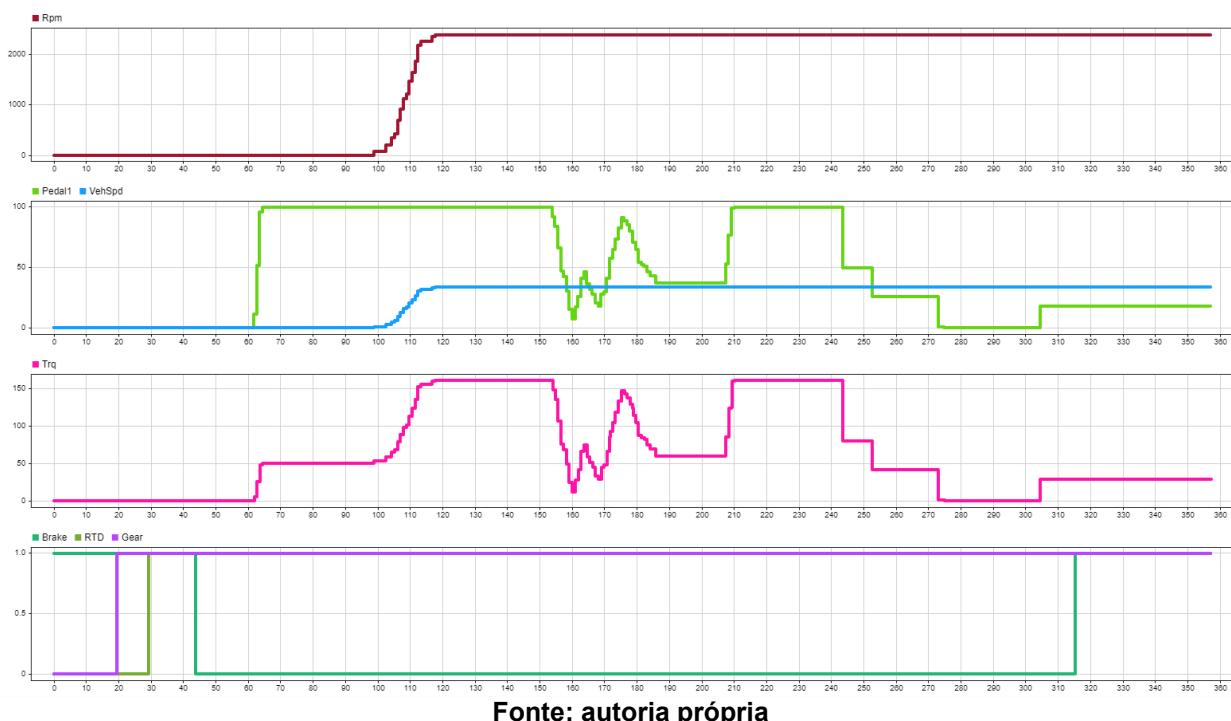
Nos quadros 7 a 12 e nas figuras 54 a 59, encontra-se a compilação dos testes realizados e seus respectivos resultados:

**Quadro 7 - Teste 1 - Dinâmica básica do veículo**

Teste	Ação	Resultado Esperado	Resultado Obtido
1) Dinâmica básica do veículo	<p>Status inicial Gear: N Drive Mode: Normal RTD: Off APPS: 0 % Brake: Pressed VehSpd: 0 m/s Rpm: 0</p> <p>Passo 1) Gear: D RTD: On Brake: Unpressed APPS: 100 % VehSpd: 0, 1.41, 7.07, 14.14, 21.21, 28.28 m/s Rpm: 0, 100, 500, 1000, 1500, 2000</p> <p>Passo 2) Variar aleatoriamente a % do pedal</p> <p>Passo 3) APPS: 100, 50, 25, 0, 20 % Brake: Pressed</p>	<p>Passo 1) Deve-se observar o torque aumentando no início e variando (pra mais ou para menos) de acordo com a velocidade do veículo.</p> <p>Passo 2) Deve-se observar o torque variando (pra mais ou para menos) de acordo com a variação do pedal.</p> <p>Passo 3) Deve-se observar o torque diminuindo, até chegar em para 0 Nm (quando o acelerador é zerado). Se o pedal estiver com menos de 25% e o freio for acionado, o torque não deve ser zerado.</p>	Conforme esperado

**Fonte: autoria própria**

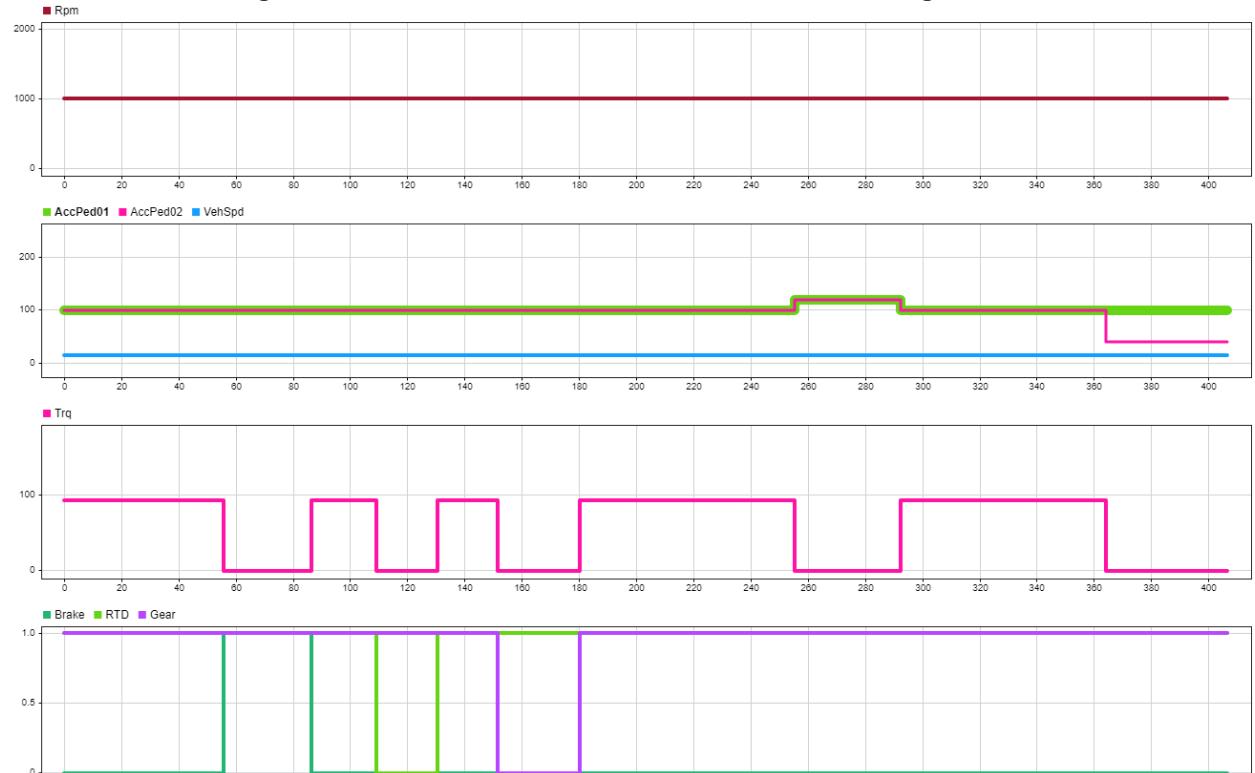
**Figura 54 – Resultados do teste 1 - Dinâmica básica do veículo**



**Fonte: autoria própria**

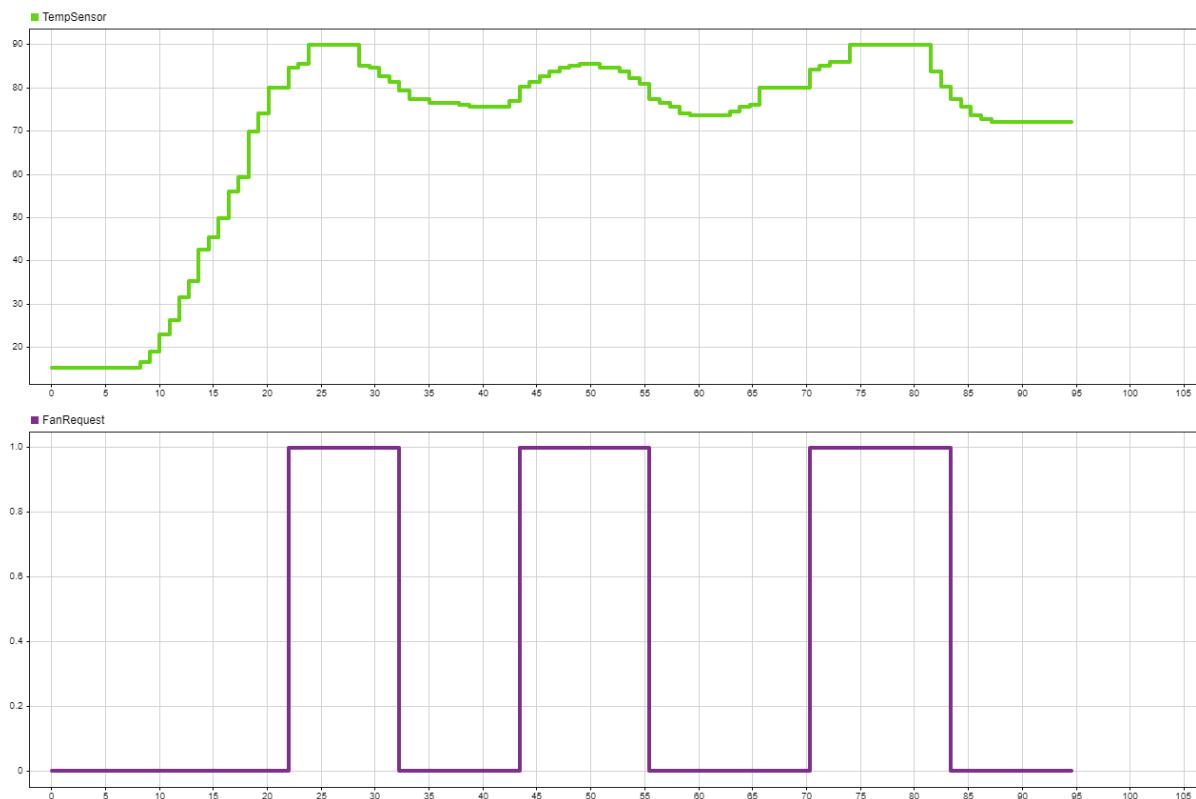
**Quadro 8 - Teste 2 - Intervenções de segurança**

Teste	Ação	Resultado Esperado	Resultado Obtido
2) Intervenções de segurança	<p>Status inicial Gear: D Drive Mode: Normal RTD: On Brake: Unpressed APPS: 100 % VehSpd: 14.14 m/s Rpm: 1000</p> <p>Passo 1) Brake: Pressed</p> <p>Passo 2) Voltar às condições iniciais e em seguida: RTD: Off</p> <p>Passo 3) Voltar às condições iniciais e em seguida: Gear: N</p> <p>Passo 4) Voltar às condições iniciais e em seguida: Enviar sinal do acelerador acima de 100%</p> <p>Passo 5) Voltar às condições iniciais e em seguida: Descasar os sinais do Acelerador em mais de 10%</p>	Para todos os passos (1 à 5), deve-se observar o torque caindo para 0Nm imediatamente após a falha ocorrer.	Conforme esperado

**Fonte:** autoria própria**Figura 55 – Resultados do teste 2 - Intervenções de segurança****Fonte:** autoria própria

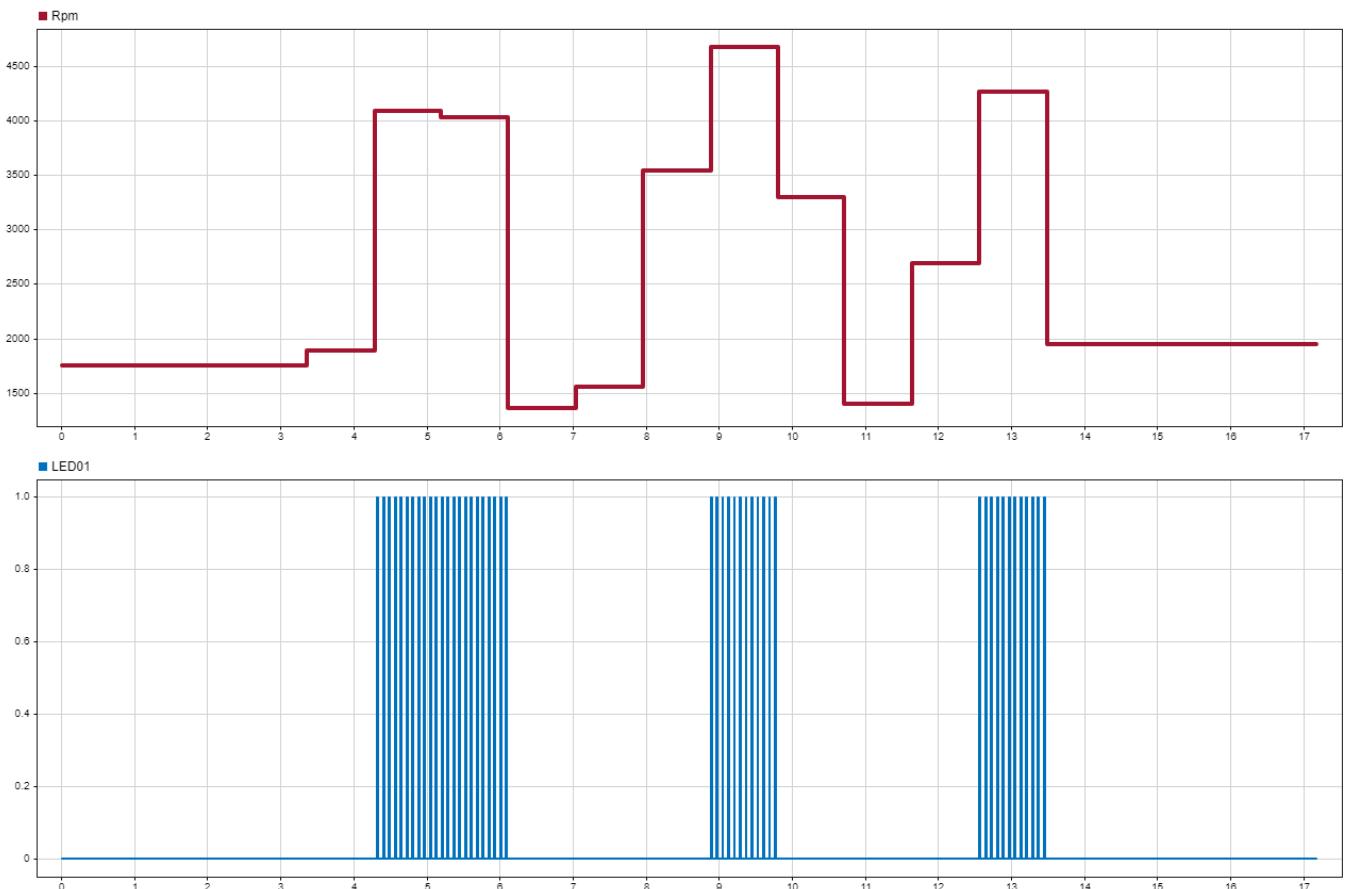
**Quadro 9 - Teste 3 – Acionamento da refrigeração**

Teste	Ação	Resultado Esperado	Resultado Obtido
3) Acionamento de cooling	Status inicial: TempSensor < 80  Passo 1) TempSensor > 80  Passo 2) TempSensor < 80	Passo 1) Deve-se observar o sinal digital da FAN mudando de Low para High  Passo 2) Deve-se observar o sinal digital da FAN mudando de High para Low	Conforme esperado

**Fonte: autoria própria****Figura 56 – Resultados do teste 3 – Acionamento da refrigeração****Fonte: autoria própria**

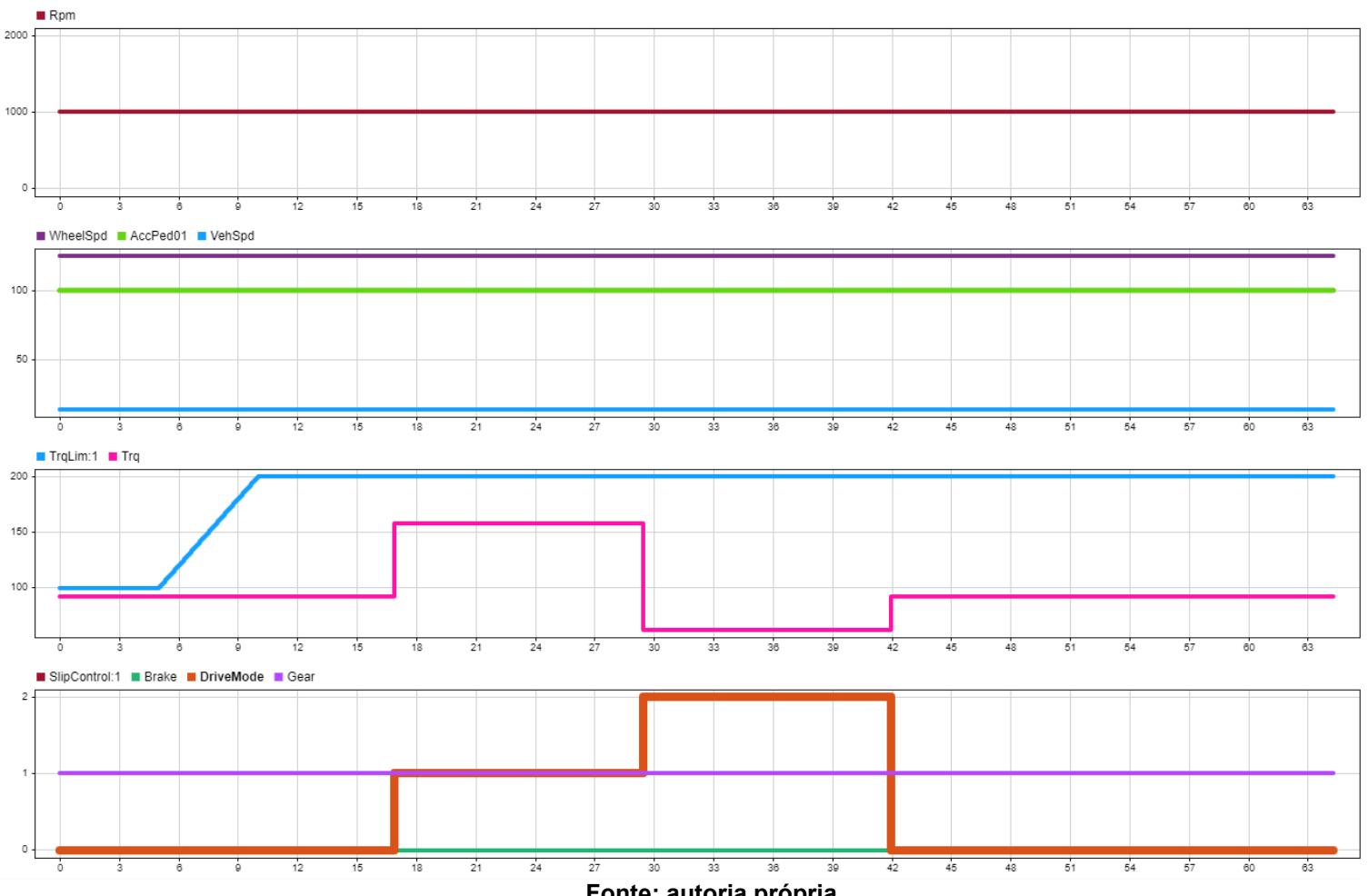
**Quadro 10 - Teste 4 – Troca de marcha**

Teste	Ação	Resultado Esperado	Resultado Obtido
4) Troca de marcha	Status inicial AxelRpm = 2000  Passo 1) AxelRpm > 4000  Passo 2) AxelRpm < 4000	Passo 1) Deve-se observar o sinal digital do LED01 variando entre High e Low (piscando)  Passo 2) Deve-se observar o sinal digital do LED01 em Low	Conforme esperado

**Fonte: autoria própria****Figura 57 – Resultados do teste 4 – Troca de marcha****Fonte: autoria própria**

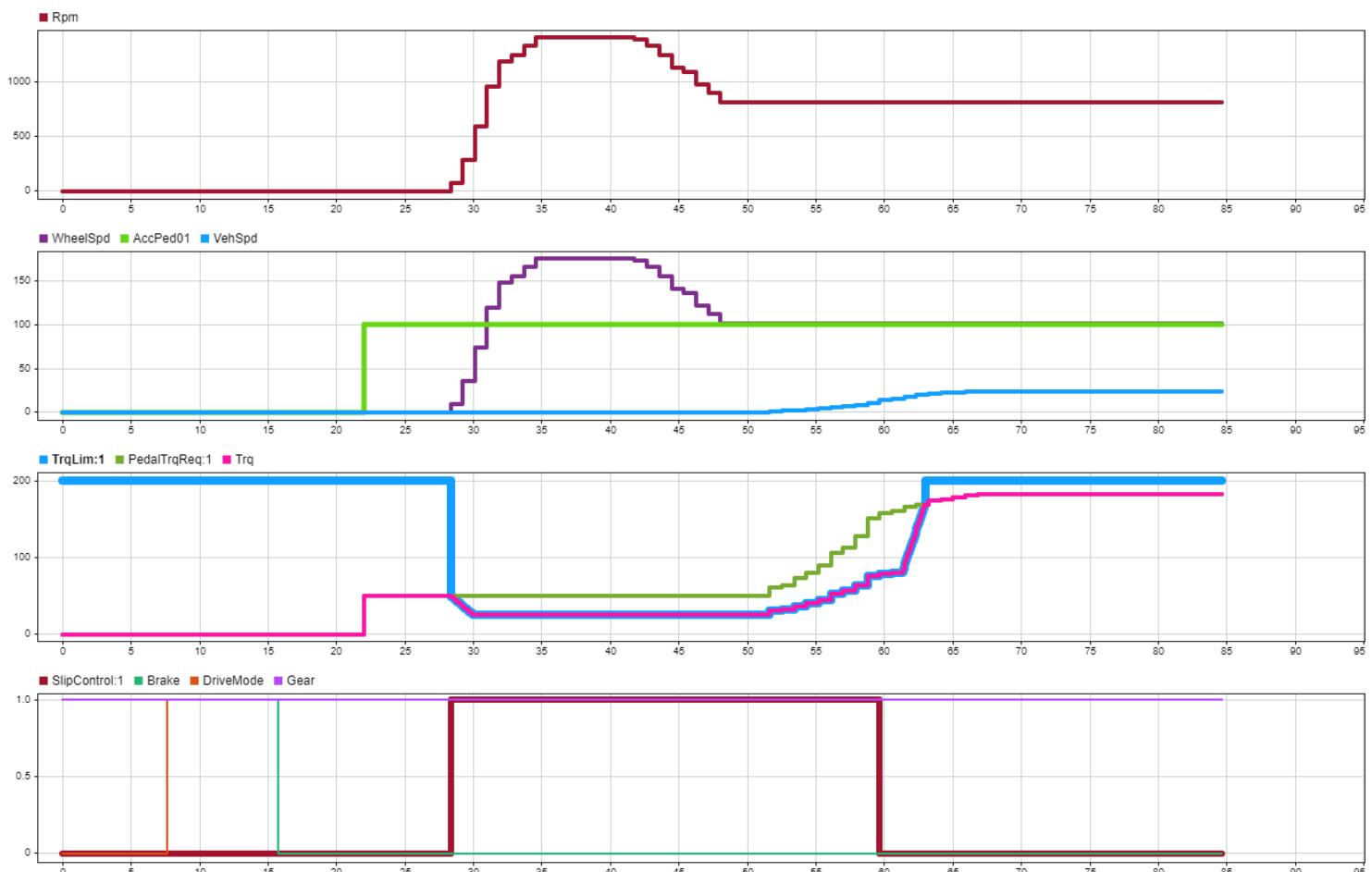
**Quadro 11 - Teste 5 – Mapas de Torque e Modos de Direção**

Teste	Ação	Resultado Esperado	Resultado Obtido
5) Mapas de Torque e Modos de Direção	<p>Status inicial Gear: D Drive Mode: Normal RTD: On APPS: 100 % Brake: Unpressed VehSpd: 14.14 m/s Rpm: 1000</p> <p>Passo 1) Drive Mode: Normal (0), Sport(1), Eco(2)</p>	<p>Passo 1) Deve-se observar o torque variando com a mudança entre os modos de direção (mesmo mantendo as condições de pedal e velocidade)</p>	Conforme esperado

**Fonte: autoria própria****Figura 58 – Resultados teste 5 – Mapas de Torque e Modos de Direção****Fonte: autoria própria**

**Quadro 12 - Teste 6 – Controle de Tração**

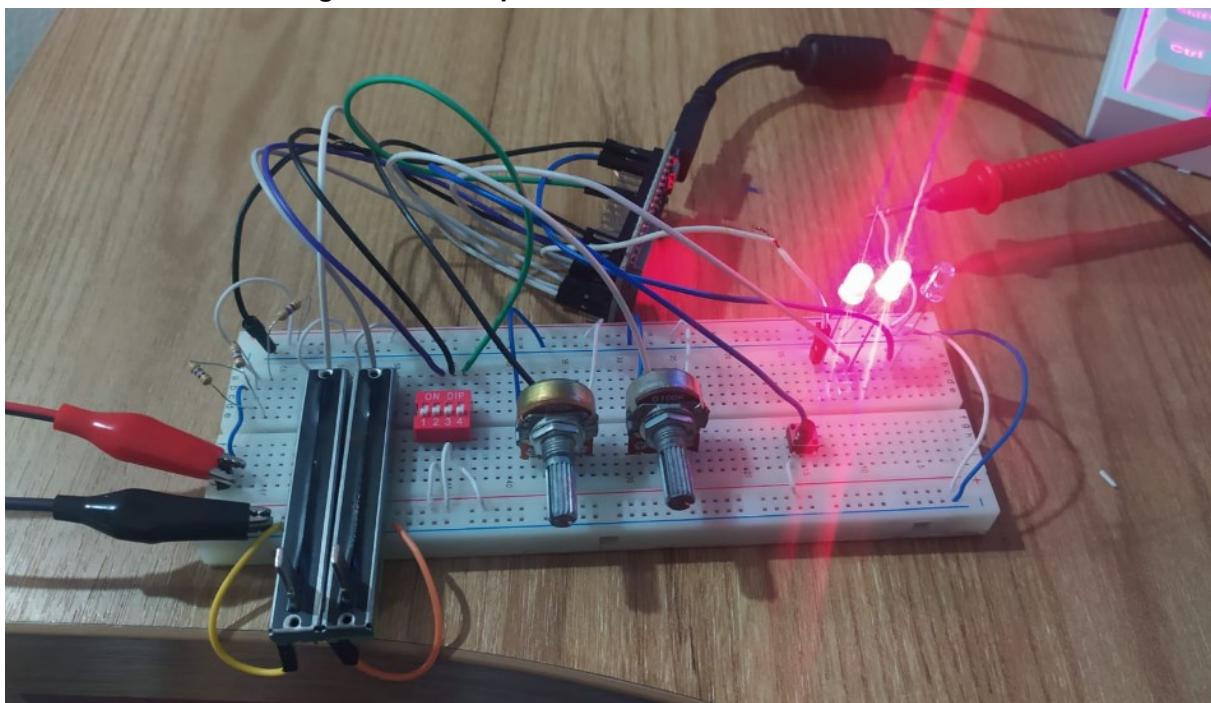
Teste	Ação	Resultado Esperado	Resultado Obtido
6) Controle de Tração	<p>Status inicial Gear: D Drive Mode: Sport RTD: On APPS: 0 % Brake: Pressed VehSpd: 0 m/s Rpm: 0</p> <p>Passo 1) Brake: Unpressed APPS: 100 % Rpm: 100, 300, 400,..2000, 1000, 700 VehSpd: 0, 2, 5, 10 m/s</p>	<p>Passo 1) deve-se observar o torque sendo limitado em uma curva decrescente até saturar. Após a velocidade do veículo atingir o mínimo esperado para determinada rotação, deve-se observar o torque sendo retomado em uma curva ascendente até chegar ao valor original.</p>	Conforme esperado

**Fonte: autoria própria****Figura 59 – Resultados do teste 6 – Controle de Tração****Fonte: autoria própria**

### 4.3 Validação integrada do Software e Hardware

Para realizar os testes finais do projeto, o software, previamente testado via simulação, foi gravado no microcontrolador e os inputs que antes também foram gerados através da simulação (painel interativo do simulink), agora foram substituídos por sinais reais gerados em bancada.

**Figura 60 – Setup dos sinais utilizados nos testes finais**



**Fonte:** autoria própria

Os sinais utilizados ficaram organizados da seguinte maneira:

Entradas:

**Pedal do acelerador:** dois potenciômetros deslizantes;

**Freio:** switch chaveando 3.3 V

**Ready to Drive:** switch chaveando 3.3 V

**Marcha:** switch chaveando 3.3 V

**Sensor de temperatura:** potenciômetro linear rotativo

**RPM:** potenciômetro linear rotativo

**Modo de direção:** Push button

Saídas:

**Lâmpada de Falha:** LED 8mm

**Ativação da ventoinha:** LED 8mm

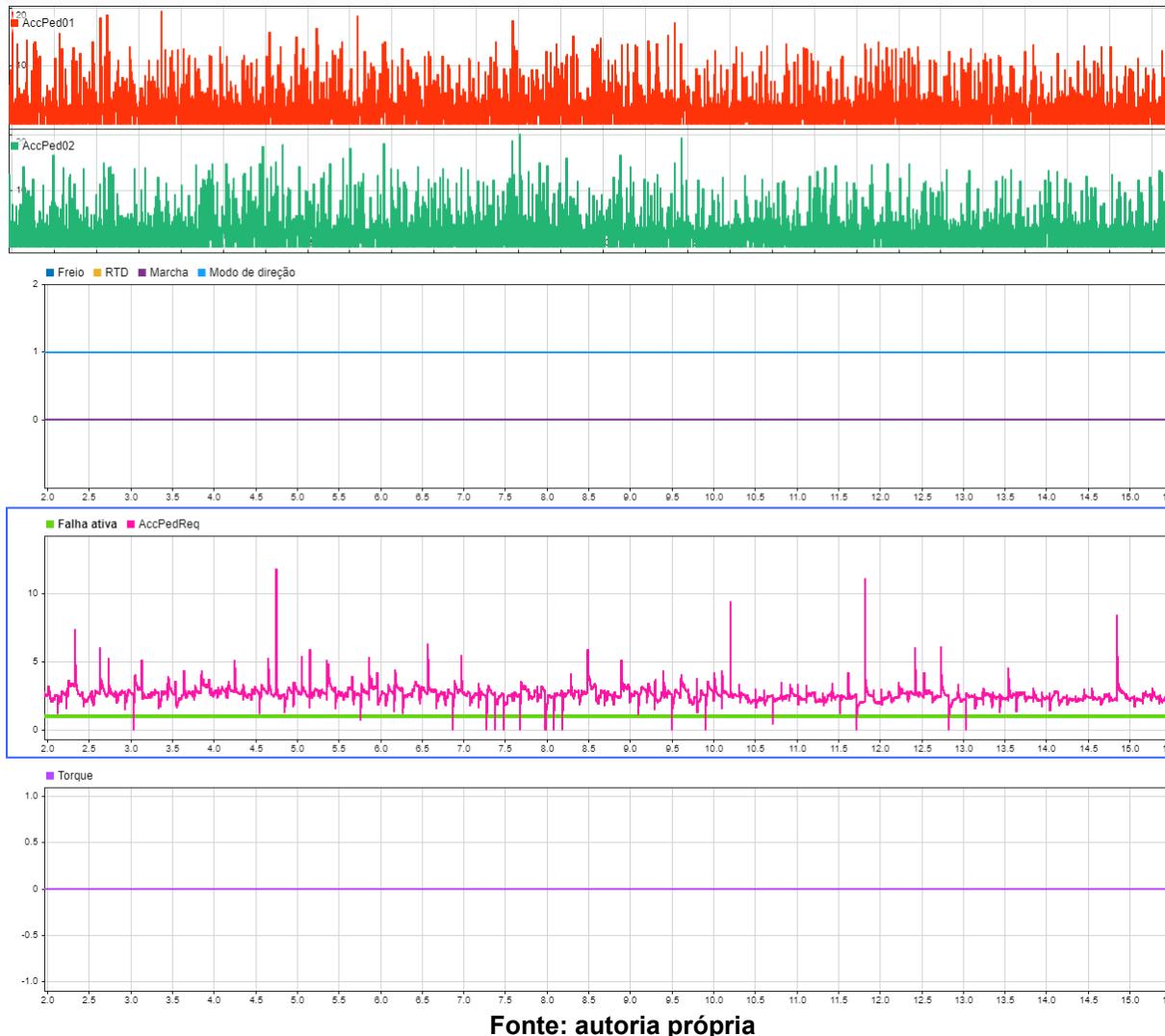
**Aviso troca de marcha:** LED 8mm

**Torque requisitado:** Mensagem serial

Foram realizados alguns testes básicos para analisar o comportamento dos sinais no microcontrolador e comparar com os resultados obtidos anteriormente.

Na figura 61, pode-se observar que houve uma variação muito grande dos sinais analógicos, mesmo com os potenciômetros parados. Há uma diferença grande entre os dois potenciômetros do acelerador, de forma que o sinal de segurança está sempre ativo e o torque final fica zerado.

**Figura 61 – Primeira medição do microcontrolador tratando os dados sem calibração prévia**

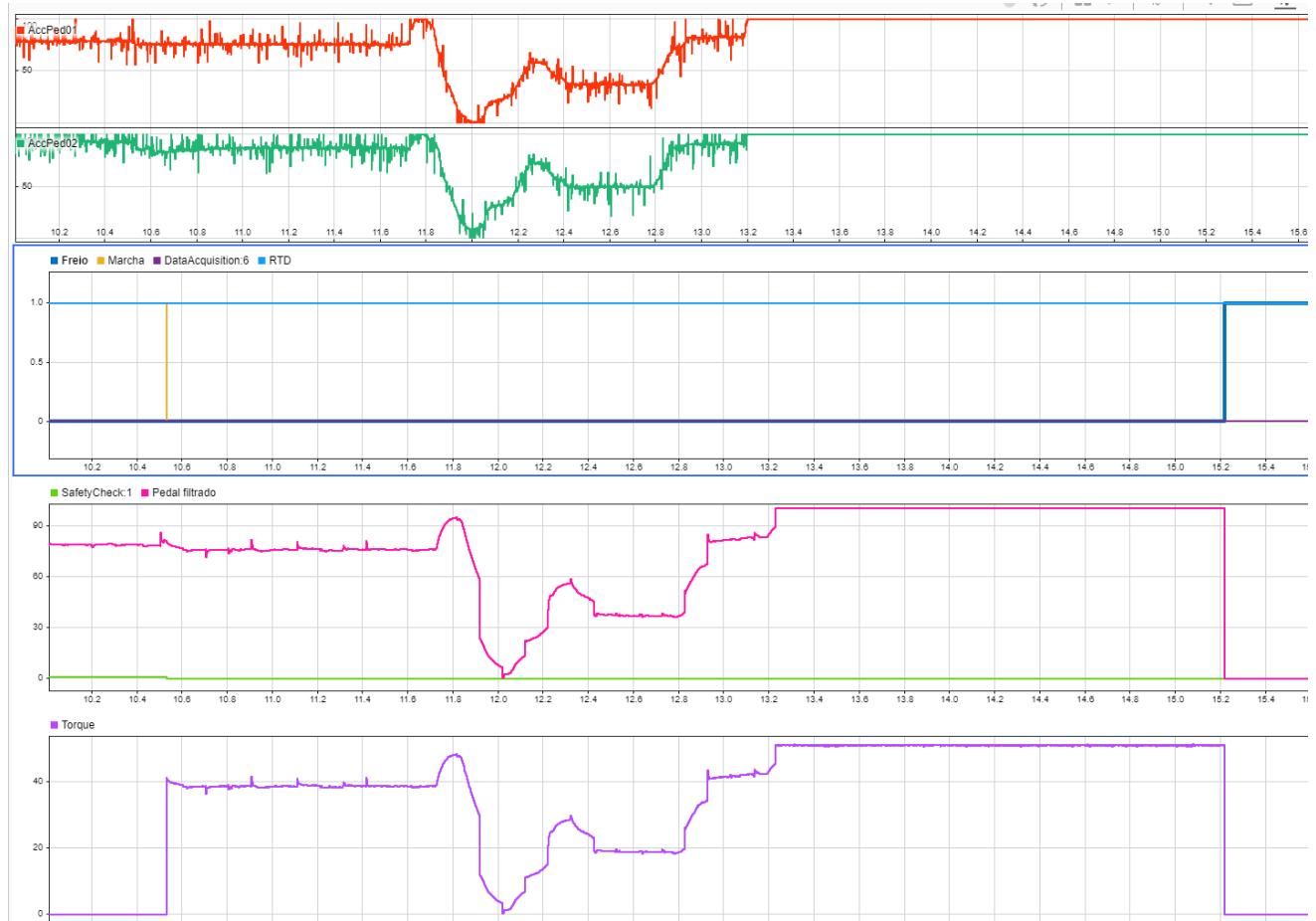


Fonte: autoria própria

Com esse teste pode-se observar a necessidade de calibrar o software para ignorar essa tensão flutuante dos sinais analógicos e aumentar um pouco as margens de segurança para checar os sinais do pedal.

Feito isso, os sinais ficaram mais estáveis e pode-se simular uma requisição de torque funcionando de acordo com o esperado, como pode-se observar na figura 62:

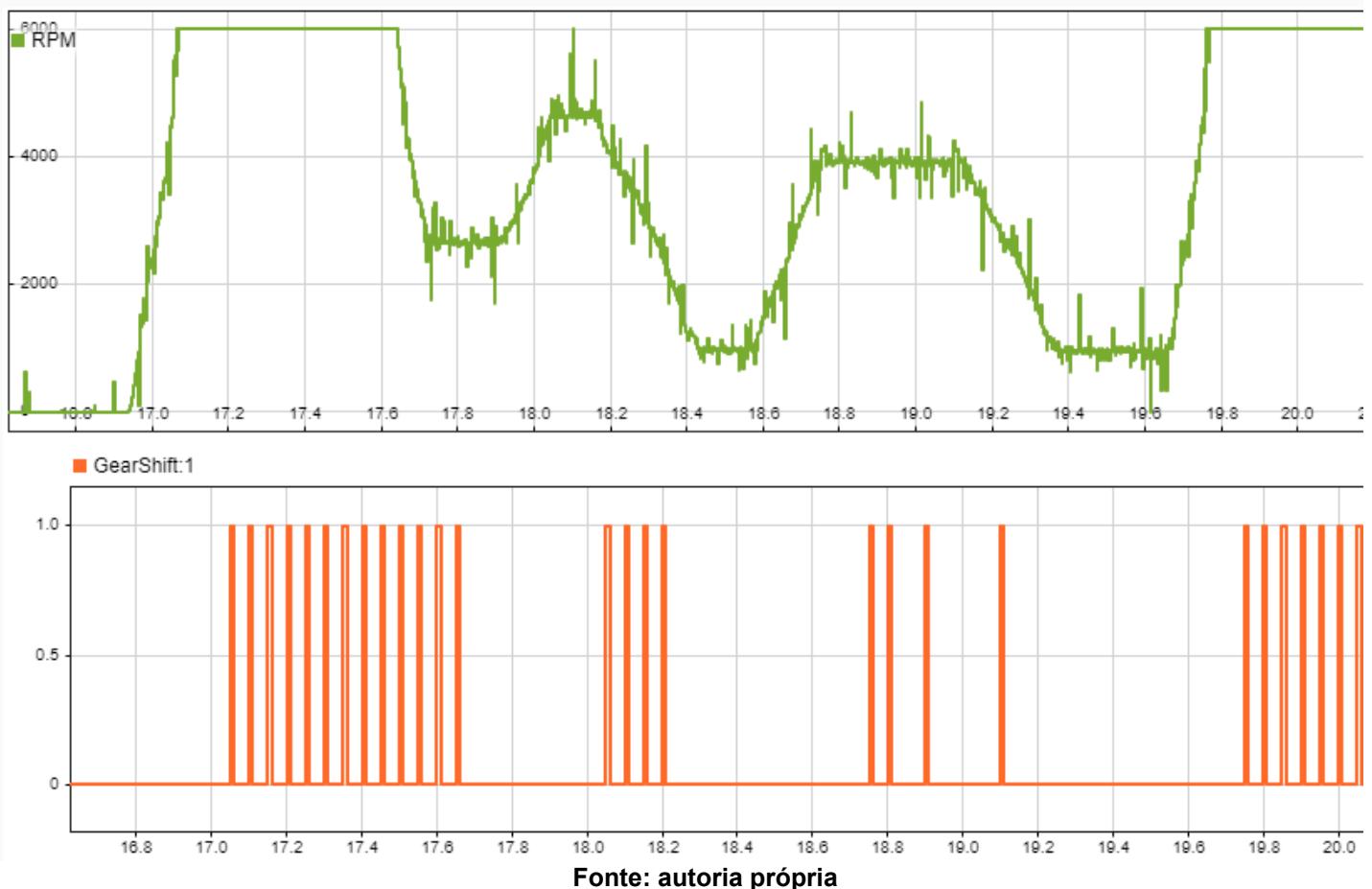
**Figura 62 – Medição feita após calibração dos sinais analógicos**



**Fonte: autoria própria**

Para obter o comportamento esperado do sensor de temperatura e o sinal de RPM, também foi necessária uma calibração prévia, com ranges específicos para cada sinal e valores de aceitáveis de flutuação.

**Figura 63 – Medição real do acionamento do aviso de troca de marcha**



**Fonte:** autoria própria

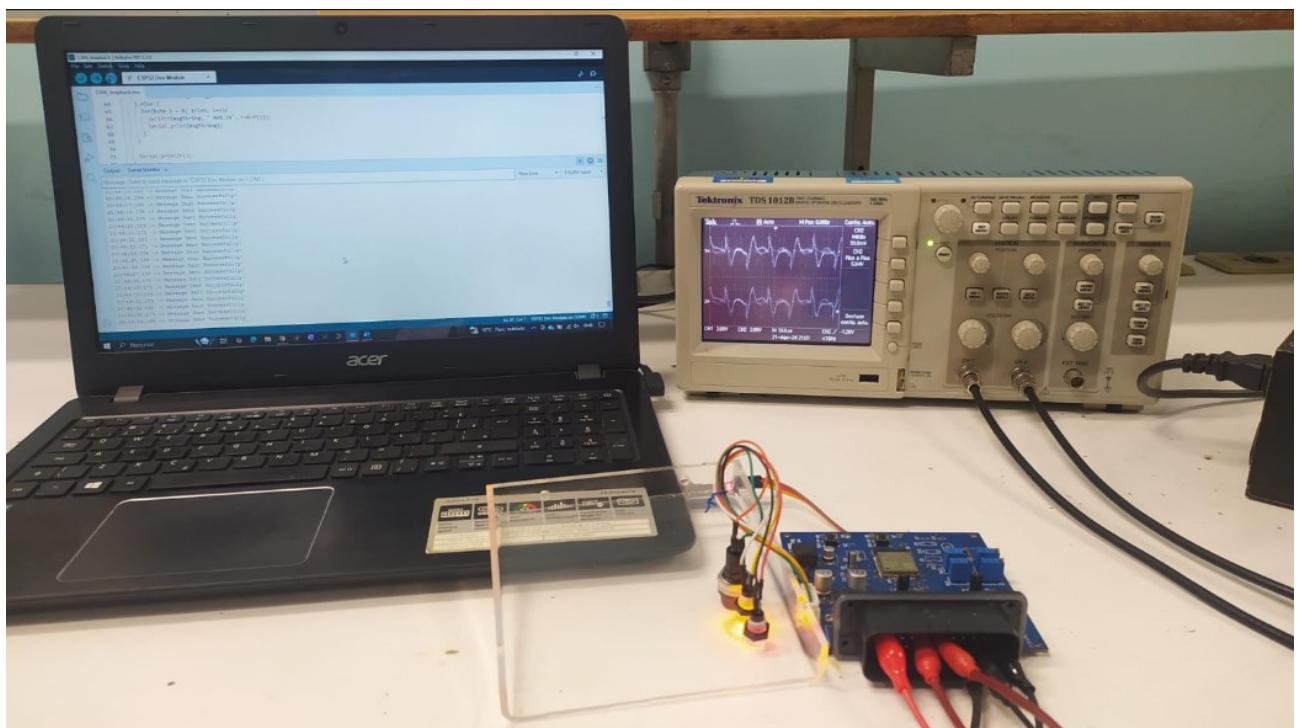
O sinal do RPM ainda pode ser melhorado adicionando um filtro via software, da mesma forma que foram adicionados filtros na função de segurança do pedal do acelerador. Mas ainda com as variações observadas, pode-se considerar o resultado satisfatório.

Um teste adicional foi realizado para verificar o status da rede CAN, porém o resultado inicial não foi satisfatório. Foi utilizado um script básico para testar o envio de mensagens CAN. Nesse script, quando o *tranceiver* CAN é inicializado e uma mensagem CAN é enviada, ele envia também uma mensagem serial para debug. Pode-se observar que as mensagens estavam sendo enviadas de acordo com o

planejado, porém o sinal lido no osciloscópio não reflete o esperado para o conteúdo da CAN.

Desse modo, pode-se entender que há um problema na conexão física da CAN e não no software implementado. A análise desse problema físico não pode ser realizada devido a limitação de tempo. Porém, há a hipótese de que estava faltando um resistor de 120ohm no lado do osciloscópio, o que pode ter interferido na leitura do sinal da CAN *High* e CAN *Low*.

**Figura 64 – Teste inicial da Rede CAN**



**Fonte:** autoria própria

## 5 CONCLUSÃO

Em conclusão, o desenvolvimento da ECU atendeu com sucesso à necessidade de um gerenciamento refinado de torque dos veículos elétricos no contexto das competições da Fórmula SAE. Este projeto não apenas cumpre os requisitos específicos da equipe UTForce e-Racing, mas também estabelece um padrão para que outras equipes possam utilizar e derivar seus próprios projetos.

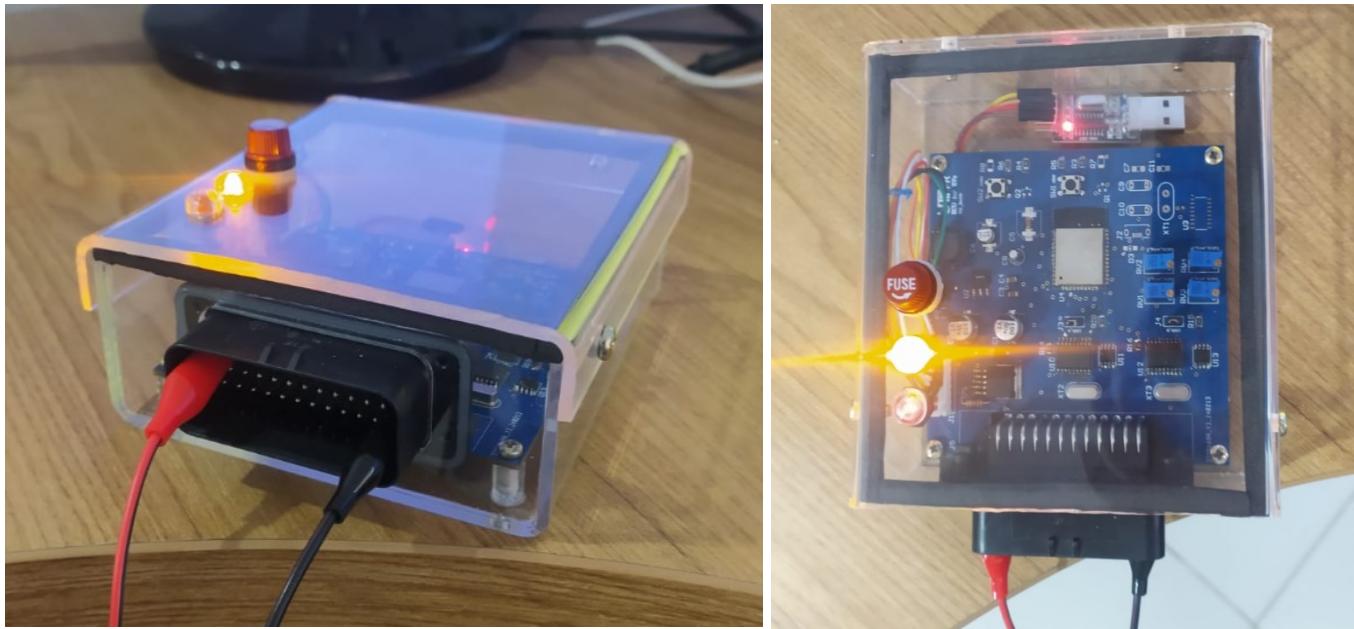
A metodologia empregada, baseada no modelo V, garantiu uma abordagem estruturada para o projeto, facilitando o levantamento minucioso de requisitos, o desenvolvimento sistemático e os testes rigorosos. A utilização do Simulink para o desenvolvimento do firmware mostrou-se muito útil para facilitar a simulação e validação dos algoritmos de controle, permitindo a detecção precoce de potenciais problemas e assegurando que a ECU opere de forma confiável sob diversas condições.

A ECU foi projetada para atender a requisitos funcionais essenciais, incluindo a leitura de sinais de acelerador, controle de tração, monitoramento de temperatura e comunicação via rede CAN. Embora os resultados iniciais indiquem que a ECU atende à maioria das especificações funcionais, ainda é necessário um trabalho adicional para validar completamente a integração da rede CAN, que é crucial para garantir uma comunicação fluida entre a ECU e outros componentes do veículo.

O design do hardware, executado com o Altium Designer e fabricado por meio de processos industriais, resultou em um produto com boa robustez, que foi finalizado com a confecção de uma caixa impermeável, garantindo a durabilidade da ECU no ambiente do automobilismo.

Na figura 65 pode-se observar a ECU finalizada com a caixa impermeável:

**Figura 65 – Montagem final da ECU**



**Fonte: autoria própria**

Antes de ser embarcada em um veículo para testes dinâmicos, a ECU necessita passar por um processo de calibração dos sensores e adequação dos valores utilizados para cálculos internos do Software, como por exemplo a circunferência das rodas e relação de transmissão.

Como sugestão de trabalhos futuros, pode-se considerar: O aprimoramento do software, implementando mais funcionalidades e refinando o controle de torque; O aprofundamento na implementação da Rede CAN e estabelecimento de um protocolo proprietário para o veículo; Desenvolvimento de um sistema de teste em loop fechado, para simular a dinâmica do veículo juntamente com a atuação do hardware em tempo real.

## REFERÊNCIAS

FARRUGIA, Mario; FARRUGIA, Michael e SANGEORZAN, Brian. **ECU Development for a Formula SAE Engine**. Oakland University, 2005.

DE OLIVEIRA, Gabriel Luis. **Desenvolvimento do sistema eletrônico de um veículo do tipo Formula Student elétrico baseado nas regras da Fórmula SAE**. UFRGS, 2020.

NUNES, Tomaz Filgueira. **Telemetria de um veículo Baja SAE através de rede CAN**. UFRN, 2016.

ROSSO, Marco Antônio Zolett. **Desenvolvimento do controle eletrônico do acelerador**. UFSC, 2017.

BOCCI, Emanuel Damasceno. **Projeto de um sistema embarcado de aquisição de dados com implementação e testes de sistema de telemetria**. UFSCAR, 2020.

MARTINAZZO, Claodomir Antonio e ORLANDO, Tailan. **Comparação entre três tipos de sensores de temperatura em associação com arduino**. URI Erechim, 2016.

GENG, Peng, OUYANG, Minggao, JIANQIU, Li e XU, Liangfei. **Automated Code Generation for Development of Electric Vehicle Controller**. Proceedings of the FISITA 2012 World Automotive Congress, Beijing, China, 2013.

LI, Yuxing, HONG, Hanchi e D'APOLITO, Luigi. **Reliability Control of Electric Racing Car's Accelerator and Brake Pedals**. World Electric Vehicle Journal, 2021.

LENZO, Basilio, DE PASCALE, Valentina, FARRONI, Flavio e TIMPONE, Francesco. **Torque Vectoring Control for fully electric Formula SAE cars**. Sheffield Hallam University, 2019.

FRICANO, Andrea. **Design of Torque Vectoring system for Formula SAE electric vehicle**. University Institute of Automobile Research (INSIA), 2019.

Ji, Pengcheng. **Development and Simulation of a Traction Control System Using Individual Motor Direct Torque Control Techniques for a Formula SAE Electric Racing Vehicle.** University of Tasmania, 2020.

HONEYWELL. **Hall Effect Sensing and Application.** Disponível em:  
[http://www.rsp-italy.it/Electronics/Databooks/Honeywell/\\_contents/Honeywell%20-%20Hall%20Effect%20Sensing%20and%20Application%201998.pdf](http://www.rsp-italy.it/Electronics/Databooks/Honeywell/_contents/Honeywell%20-%20Hall%20Effect%20Sensing%20and%20Application%201998.pdf)  
(Acesso em 19 jun 2023).

HUDAK, Zoltan. Home made mbed board with LPC1768 microcontroller. Disponível em:  
<https://os.mbed.com/users/hudakz/notebook/home-made-mbed-board-with-lpc1768/>  
(Acesso em 01 jun 2023).

ESPRESSIF, Systems. **ESP32 Hardware Design Guidelines.** Disponível em:  
[https://www.espressif.com/sites/default/files/documentation/esp32\\_hardware\\_design\\_guidelines\\_en.pdf](https://www.espressif.com/sites/default/files/documentation/esp32_hardware_design_guidelines_en.pdf)  
(Acesso em 05 jun 2023).

ESPRESSIF, Systems. **ESP32-WROOM-32 Datasheet.** Disponível em:  
[https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32\\_datasheet\\_en.pdf](https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf)  
(Acesso em 05 jun 2023).

SAE International. **Formula SAE Rules 2023 Version 1.0.** Disponível em:  
<https://www.fsaeonline.com/cdsweb/gen/DownloadDocument.aspx?DocumentID=d2404288-c121-4242-9944-90aa0b1d6a5f>  
(Acesso em 20 maio 2023).

ESPINHA, Roberto. **Gráfico de Gantt: o que é, para que serve e como montar do zero no Excel.** Artia, 2022. Disponível em: <https://artia.com/blog/grafico-de-gantt-o-que-e-para-que-servir-e-como-montar-o-seu/>  
(Acesso em 27 nov. 2022).

AZEVEDO, Rogério C; Ensslin. **Metodologia da pesquisa para engenharias** Disponível em:  
[file:///C:/Users/chmar/Dropbox/Metodologia%20para%20o%20TCC/Bases%20do%20conhecimento/Outras%20apresenta%C3%A7%C3%B5es%20complementares/e-book\\_metodologia\\_pesquisa\\_para\\_engenharias.pdf](file:///C:/Users/chmar/Dropbox/Metodologia%20para%20o%20TCC/Bases%20do%20conhecimento/Outras%20apresenta%C3%A7%C3%B5es%20complementares/e-book_metodologia_pesquisa_para_engenharias.pdf)  
(Acesso em 28 nov. 2022).

MARIANO, Carlos. **Tutorial para redação da proposta de pesquisa para o trabalho de conclusão de curso.** Disponível em:  
[https://drive.google.com/drive/folders/1sTEAxFPcteV5zHXwW1GQP\\_Cm-Ux2ptON](https://drive.google.com/drive/folders/1sTEAxFPcteV5zHXwW1GQP_Cm-Ux2ptON)  
(Acesso em 22 nov. 2022).

ROCHA, Camila. **Estudo da qualidade de software na Metodologia V-model e sua interação com metodologias ágeis (SCRUM).** Disponível em:  
<http://www.fatecsp.br/dti/tcc/tcc0028.pdf>  
(Acesso em 29 nov. 2022).

O'NEILL, Erin. **McGill Formula Electric Torque Controller.** Disponível em:  
<https://www.mathworks.com/matlabcentral/fileexchange/72848-mcgill-formula-electric-torque-controller>  
(Acesso em 06 abr. 2024).

**APÊNDICE A: Extrato do código gerado através do Simulink Embedded Coder**

```

1 /*
2 * ert_main.c
3 *
4 * Academic License - for use in teaching, academic research, and meeting
5 * course requirements at degree granting institutions only. Not for
6 * government, commercial, or other organizational use.
7 *
8 * Code generation for model "Torque_Control_ESP32_V6".
9 *
10 * Model version : 3.6
11 * Simulink Coder version : 24.1 (R2024a) 19-Nov-2023
12 * C source code generated on : Fri Aug 23 05:11:05 2024
13 *
14 * Target selection: ert.tlc
15 * Embedded hardware selection: ARM Compatible->ARM Cortex
16 * Code generation objectives: Unspecified
17 * Validation result: Not run
18 */
19
20 #include <stdio.h>
21 #include <stdlib.h>
22 #include "Torque_Control_ESP32_V6.h"
23 #include "rtwtypes.h"
24 #include "limits.h"
25 #include "MW_ArduinoHWInit.h"
26 #include "mw_freertos.h"
27 #define UNUSED(x) x = x
28 #define NAMELEN 16
29
30 /* Function prototype declaration*/
31 void exitFcn(int sig);
32 void *terminateTask(void *arg);
33 void *baseRateTask(void *arg);
34 void *substrateTask(void *arg);
35 volatile boolean_T stopRequested = false;
36 volatile boolean_T runModel = true;
37 SemaphoreHandle_t stopSem;
38 SemaphoreHandle_t baserateTaskSem;
39 mw_thread_t schedulerThread;
40 mw_thread_t baseRateThread;
41 void *threadJoinStatus;
42 int terminatingmodel = 0;
43 void *baseRateTask(void *arg)
44 {
45 runModel = (rtmGetErrorHandler(Torque_Control_ESP32_V6_M) == (NULL));
46 while (runModel) {
47 mw_osSemaphoreWaitEver(&baserateTaskSem);
48 Torque_Control_ESP32_V6_step();
49
50 /* Get model outputs here */
51 stopRequested = !((rtmGetErrorHandler(Torque_Control_ESP32_V6_M) == (NULL)));
52 runModel = !stopRequested;
53 }
54
55 runModel = 0;
56 terminateTask(arg);
57 mw_osThreadExit((void *)0);
58 return NULL;
59 }
60
61 void exitFcn(int sig)
62 {
63 UNUSED(sig);
64 rtmSetErrorHandler(Torque_Control_ESP32_V6_M, "stopping the model");
65 }
66
67 void *terminateTask(void *arg)

```

```

68 {
69 UNUSED(arg);
70 terminatingmodel = 1;
71
72 {
73 runModel = 0;
74 }
75
76 /* Terminate model */
77 Torque_Control_ESP32_V6_terminate();
78 mw_osSemaphoreRelease(&stopSem);
79 return NULL;
80 }
81
82 int app_main(int argc, char **argv)
83 {
84 init();
85 MW_Arduino_Init();
86 rtmSetErrorStatus(Torque_Control_ESP32_V6_M, 0);
87
88 /* Initialize model */
89 Torque_Control_ESP32_V6_initialize();
90
91 /* Call RTOS Initialization function */
92 mwRTOSInit(0.001, 0);
93
94 /* Wait for stop semaphore */
95 mw_osSemaphoreWaitEver(&stopSem);
96
97 #if (MW_NUMBER_TIMER_DRIVEN_TASKS > 0)
98
99 {
100 int i;
101 for (i=0; i < MW_NUMBER_TIMER_DRIVEN_TASKS; i++) {
102 CHECK_STATUS(mw_osSemaphoreDelete(&timerTaskSem[i]), 0,
103 "mw_osSemaphoreDelete");
104 }
105 }
106
107 #endif
108
109 return 0;
110 }
111
112 /*
2 * Torque_Control_ESP32_V6.c
3 *
4 * Academic License - for use in teaching, academic research, and meeting
5 * course requirements at degree granting institutions only. Not for
6 * government, commercial, or other organizational use.
7 *
8 * Code generation for model "Torque_Control_ESP32_V6".
9 *
10 * Model version : 3.6
11 * Simulink Coder version : 24.1 (R2024a) 19-Nov-2023
12 * C source code generated on : Fri Aug 23 05:11:05 2024
13 *
14 * Target selection: ert.tlc
15 * Embedded hardware selection: ARM Compatible->ARM Cortex
16 * Code generation objectives: Unspecified
17 * Validation result: Not run
18 */
19
20 #include "Torque_Control_ESP32_V6.h"
21 #include <math.h>
22 #include "rtwtypes.h"
23 #include "zero_crossing_types.h"

```

```

24 #include <stddef.h>
25 #include "solver_zc.h"
26 #ifndef slZcHadEvent
27 #define slZcHadEvent(ev, zcsDir) (((ev) & (zcsDir)) != 0x00 )
28 #endif
29
30 #ifndef slZcUnAliasEvents
31 #define slZcUnAliasEvents(evL, evR) (((slZcHadEvent((evL), (SL_ZCS_EVENT_N2Z)) &&
32 slZcHadEvent((evR), (SL_ZCS_EVENT_Z2P))) || (slZcHadEvent((evL), (SL_ZCS_EVENT_P2Z))
33 && slZcHadEvent((evR), (SL_ZCS_EVENT_Z2N)))) ? (SL_ZCS_EVENT_NUL) : (evR)))
32 #endif
33
34 #define NumBitsPerChar 8U
35
36 /* Block parameters (default storage) */
37 P_Torque_Control_ESP32_V6_T Torque_Control_ESP32_V6_P = {
38 /* Mask Parameter: CompareToConstant_const
39 * Referenced by: '<S8>/Constant'
40 */
41 1.0F,
42
43 /* Mask Parameter: CompareToConstant_const_b
44 * Referenced by: '<S20>/Constant'
45 */
46 100.0F,
47
48 /* Mask Parameter: CompareToConstant1_const
49 * Referenced by: '<S21>/Constant'
50 */
51 0.0F,
52
53 /* Mask Parameter: CompareToConstant2_const
54 * Referenced by: '<S22>/Constant'
55 */
56 100.0F,
57
58 /* Mask Parameter: CompareToConstant3_const
59 * Referenced by: '<S23>/Constant'
60 */
61 0.0F,
62
63 /* Mask Parameter: CompareToConstant5_const
64 * Referenced by: '<S25>/Constant'
65 */
66 25.0F,
67
68 /* Mask Parameter: CompareToConstant4_const
69 * Referenced by: '<S24>/Constant'
70 */
71 20.0F,
72
73 /* Mask Parameter: CompareToConstant_const_a
74 * Referenced by: '<S9>/Constant'
75 */
76 4000.0F,
77
78 /* Mask Parameter: CompareToConstant6_const
79 * Referenced by: '<S26>/Constant'
80 */
81 false,
82
83 /* Mask Parameter: WrapToZero_Threshold
84 * Referenced by: '<S32>/FixPt Switch'
85 */
86 100U,
87
88 /* Mask Parameter: WrapToZero_Threshold_b

```

```
89 /* Referenced by: '<S34>/FixPt Switch'
90 */
91 100U,
92
93 /* Mask Parameter: CompareToConstant7_const
94 * Referenced by: '<S27>/Constant'
95 */
96 100U,
97
98 /* Mask Parameter: CompareToConstant8_const
99 * Referenced by: '<S28>/Constant'
100 */
101 100U,
102
103 /* Expression: -1
104 * Referenced by: '<S2>/Analog Input1'
105 */
106 -1.0,
107
108 /* Expression: -1
109 * Referenced by: '<S2>/Analog Input4'
110 */
111 -1.0,
112
113 /* Expression: -1
114 * Referenced by: '<S2>/Analog Input5'
115 */
116 -1.0,
117
118 /* Expression: -1
119 * Referenced by: '<S2>/Analog Input6'
120 */
121 -1.0,
122
123 /* Expression: -1
124 * Referenced by: '<S2>/Analog Input7'
125 */
126 -1.0,
127
128 /* Expression: -1
129 * Referenced by: '<S2>/Digital Input'
130 */
131 -1.0,
132
133 /* Expression: -1
134 * Referenced by: '<S2>/Digital Input1'
135 */
136 -1.0,
137
138 /* Expression: -1
139 * Referenced by: '<S2>/Digital Input2'
140 */
141 -1.0,
142
143 /* Expression: -1
144 * Referenced by: '<S2>/Digital Input3'
145 */
146 -1.0,
147
148 /* Expression: -1
149 * Referenced by: '<S2>/Digital Input4'
150 */
151 -1.0,
152
153 /* Expression: 0
154 * Referenced by: '<S2>/Constant13'
155 */
```

```
156 0.0,
157
158 /* Expression: 1
159 * Referenced by: '<S2>/Constant10'
160 */
161 1.0,
162
163 /* Expression: 60
164 * Referenced by: '<S2>/Constant14'
165 */
166 60.0,
167
168 /* Expression: 0
169 * Referenced by: '<S2>/Constant16'
170 */
171 0.0,
172
173 /* Expression: 1
174 * Referenced by: '<S2>/Constant15'
175 */
176 1.0,
177
178 /* Expression: 0
179 * Referenced by: '<S2>/Constant19'
180 */
181 0.0,
182
183 /* Expression: 5000
184 * Referenced by: '<S2>/Constant9'
185 */
186 5000.0,
187
188 /* Expression: 0
189 * Referenced by: '<S2>/Constant11'
190 */
191 0.0,
192
193 /* Expression: 0
194 * Referenced by: '<S2>/Switch9'
195 */
196 0.0,
197
198 /* Expression: 24.11483253588517
199 * Referenced by: '<S2>/VehSpd'
200 */
201 24.114832535885171,
202
203 /* Expression: 2*pi*0.3*3.6
204 * Referenced by: '<S2>/sec1'
205 */
206 6.785840131753953,
207
208 /* Expression: 60
209 * Referenced by: '<S2>/sec'
210 */
211 60.0,
212
213 /* Expression: 8
214 * Referenced by: '<S2>/TransmRatio'
215 */
216 8.0,
217
218 /* Expression: 0
219 * Referenced by: '<S2>/Constant1'
220 */
221 0.0,
222
```

```
223 /* Expression: 0
224 * Referenced by: '<S2>/Constant2'
225 */
226 0.0,
227
228 /* Expression: 0
229 * Referenced by: '<S2>/Switch'
230 */
231 0.0,
232
233 /* Expression: 0
234 * Referenced by: '<S2>/Switch2'
235 */
236 0.0,
237
238 /* Expression: 0
239 * Referenced by: '<S2>/Constant4'
240 */
241 0.0,
242
243 /* Expression: 0
244 * Referenced by: '<S2>/Switch4'
245 */
246 0.0,
247
248 /* Expression: 1
249 * Referenced by: '<S2>/Constant'
250 */
251 1.0,
252
253 /* Expression: 0
254 * Referenced by: '<S2>/Switch1'
255 */
256 0.0,
257
258 /* Expression: 0
259 * Referenced by: '<S2>/Constant5'
260 */
261 0.0,
262
263 /* Expression: 0
264 * Referenced by: '<S2>/Switch5'
265 */
266 0.0,
267
268 /* Expression: 0
269 * Referenced by: '<S2>/Constant6'
270 */
271 0.0,
272
273 /* Expression: 0
274 * Referenced by: '<S2>/Switch6'
275 */
276 0.0,
277
278 /* Expression: 0
279 * Referenced by: '<S2>/Constant7'
280 */
281 0.0,
282
283 /* Expression: 0
284 * Referenced by: '<S2>/Constant8'
285 */
286 0.0,
287
288 /* Expression: 0
289 * Referenced by: '<S2>/Switch7'
```

```

290 */
291 0.0,
292
293 /* Expression: 0
294 * Referenced by: '<S2>/Switch8'
295 */
296 0.0,
297
298 /* Expression: 0
299 * Referenced by: '<S2>/Constant3'
300 */
301 0.0,
302
303 /* Expression: 0
304 * Referenced by: '<S2>/Switch3'
305 */
306 0.0,
307
308 /* Expression: 0
309 * Referenced by: '<S18>/Constant2'
310 */
311 0.0,
312
313 /* Expression: 0
314 * Referenced by: '<S18>/Constant4'
315 */
316 0.0,
317
318 /* Expression: 0
319 * Referenced by: '<S18>/Constant'
320 */
321 0.0,
322
323 /* Expression: 50
324 * Referenced by: '<S4>/Pulse Generator1'
325 */
326 50.0,
327
328 /* Expression: 50
329 * Referenced by: '<S4>/Pulse Generator1'
330 */
331 50.0,
332
333 /* Expression: 10
334 * Referenced by: '<S4>/Pulse Generator1'
335 */
336 10.0,
337
338 /* Expression: 0
339 * Referenced by: '<S4>/Pulse Generator1'
340 */
341 0.0,
342
343 /* Expression: Table
344 * Referenced by: '<S2>/Curve3'
345 */
346 { 0.0F, 100.0F },
347
348 /* Expression: BreakpointsForDimension1
349 * Referenced by: '<S2>/Curve3'
350 */
351 { 200.0F, 4095.0F },
352
353 /* Expression: Table
354 * Referenced by: '<S2>/AccPed2Curve'
355 */
356 { 0.0F, 100.0F },

```

```

357
358 /* Expression: BreakpointsForDimension1
359 * Referenced by: '<S2>/AccPed2Curve'
360 */
361 { 200.0F, 4095.0F },
362
363 /* Expression: Table
364 * Referenced by: '<S2>/Curve2'
365 */
366 { 0.0F, 200.0F },
367
368 /* Expression: BreakpointsForDimension1
369 * Referenced by: '<S2>/Curve2'
370 */
371 { 200.0F, 4095.0F },
372
373 /* Expression: Table
374 * Referenced by: '<S2>/Curve1'
375 */
376 { 0.0F, 6000.0F },
377
378 /* Expression: BreakpointsForDimension1
379 * Referenced by: '<S2>/Curve1'
380 */
381 { 200.0F, 4095.0F },
382
383 /* Expression: Table
384 * Referenced by: '<S2>/AccPed1Curve'
385 */
386 { 0.0F, 100.0F },
387
388 /* Expression: BreakpointsForDimension1
389 * Referenced by: '<S2>/AccPed1Curve'
390 */
391 { 200.0F, 4095.0F },
392
393 /* Computed Parameter: Constant_Value_l
394 * Referenced by: '<S7>/Constant'
395 */
396 1.0F,
397
398 /* Computed Parameter: Constant1_Value_h
399 * Referenced by: '<S7>/Constant1'
400 */
401 0.0F,
402
403 /* Computed Parameter: DrvMode_st_Y0
404 * Referenced by: '<S7>/DrvMode_st'
405 */
406 0.0F,
407
408 /* Computed Parameter: AccPed01_Y0
409 * Referenced by: '<S2>/AccPed01'
410 */
411 0.0F,
412
413 /* Computed Parameter: AccPed02_Y0
414 * Referenced by: '<S2>/AccPed02'
415 */
416 0.0F,
417
418 /* Computed Parameter: BrkPedDig_st_Y0
419 * Referenced by: '<S2>/BrkPedDig_st'
420 */
421 0.0F,
422
423 /* Computed Parameter: BrkPedAnlg_Y0

```

```
424 /* Referenced by: '<S2>/BrkPedAnlg'  
425 */  
426 0.0F,  
427  
428 /* Computed Parameter: RTD_st_Y0  
429 * Referenced by: '<S2>/RTD_st'  
430 */  
431 0.0F,  
432  
433 /* Computed Parameter: AxelRpm_Y0  
434 * Referenced by: '<S2>/AxelRpm'  
435 */  
436 300.0F,  
437  
438 /* Computed Parameter: VehSpeed_Y0  
439 * Referenced by: '<S2>/VehSpeed'  
440 */  
441 0.0F,  
442  
443 /* Computed Parameter: TempSensr_Y0  
444 * Referenced by: '<S2>/TempSensr'  
445 */  
446 0.0F,  
447  
448 /* Computed Parameter: Gear_st_Y0  
449 * Referenced by: '<S2>/Gear_st'  
450 */  
451 300.0F,  
452  
453 /* Computed Parameter: Constant12_Value  
454 * Referenced by: '<S2>/Constant12'  
455 */  
456 100.0F,  
457  
458 /* Computed Parameter: Delay_InitialCondition  
459 * Referenced by: '<S2>/Delay'  
460 */  
461 0.0F,  
462  
463 /* Computed Parameter: Constant_Value_i  
464 * Referenced by: '<S1>/Constant'  
465 */  
466 1.0F,  
467  
468 /* Computed Parameter: Constant1_Value_i  
469 * Referenced by: '<S1>/Constant1'  
470 */  
471 0.0F,  
472  
473 /* Computed Parameter: Switch_Threshold_m  
474 * Referenced by: '<S1>/Switch'  
475 */  
476 80.0F,  
477  
478 /* Computed Parameter: brkdefault_Value  
479 * Referenced by: '<S6>/brkdefault'  
480 */  
481 0.0F,  
482  
483 /* Computed Parameter: AccPedReq_Y0  
484 * Referenced by: '<S6>/AccPedReq'  
485 */  
486 0.0F,  
487  
488 /* Computed Parameter: AnlgBrkPrsnt_Value  
489 * Referenced by: '<S6>/AnlgBrkPrsnt'  
490 */
```

```

491 1.0F,
492
493 /* Computed Parameter: Switch1_Threshold_f
494 * Referenced by: '<S19>/Switch1'
495 */
496 0.0F,
497
498 /* Expression: Table
499 * Referenced by: '<S12>/NormalMode'
500 */
501 { 0.0F, 5.0F, 10.0F, 15.0F, 20.0F, 25.0F, 30.0F, 35.0F, 40.0F, 45.0F, 50.0F,
502 0.0F, 8.0F, 16.0F, 24.0F, 32.0F, 40.0F, 48.0F, 56.0F, 64.0F, 72.0F, 80.0F,
503 0.0F, 11.0F, 22.0F, 33.0F, 44.0F, 55.0F, 66.0F, 77.0F, 88.0F, 99.0F, 110.0F,
504 0.0F, 15.0F, 30.0F, 45.0F, 60.0F, 75.0F, 90.0F, 105.0F, 120.0F, 135.0F,
505 150.0F, 0.0F, 18.0F, 36.0F, 54.0F, 72.0F, 90.0F, 108.0F, 126.0F, 144.0F,
506 162.0F, 180.0F, 0.0F, 20.0F, 40.0F, 60.0F, 80.0F, 100.0F, 120.0F, 140.0F,
507 160.0F, 180.0F, 200.0F, 0.0F, 20.0F, 40.0F, 60.0F, 80.0F, 100.0F, 120.0F,
508 140.0F, 160.0F, 180.0F, 200.0F, 0.0F, 18.0F, 36.0F, 54.0F, 72.0F, 90.0F,
509 108.0F, 126.0F, 144.0F, 162.0F, 180.0F, 0.0F, 16.0F, 32.0F, 48.0F, 64.0F,
510 80.0F, 96.0F, 112.0F, 128.0F, 144.0F, 160.0F, 0.0F, 16.0F, 32.0F, 48.0F,
511 64.0F, 80.0F, 96.0F, 112.0F, 128.0F, 144.0F, 160.0F },
512
513 /* Expression: BreakpointsForDimension1
514 * Referenced by: '<S12>/NormalMode'
515 */
516 { 0.0F, 10.0F, 20.0F, 30.0F, 40.0F, 50.0F, 60.0F, 70.0F, 80.0F, 90.0F, 100.0F
517 },
518
519 /* Expression: BreakpointsForDimension2
520 * Referenced by: '<S12>/NormalMode'
521 */
522 { 0.0F, 10.0F, 20.0F, 30.0F, 40.0F, 60.0F, 80.0F, 100.0F, 125.0F, 150.0F },
523
524 /* Expression: Table
525 * Referenced by: '<S14>/SportMode'
526 */
527 { 0.0F, 5.0F, 10.0F, 15.0F, 20.0F, 25.0F, 30.0F, 35.0F, 40.0F, 45.0F, 50.0F,
528 0.0F, 15.0F, 30.0F, 45.0F, 60.0F, 75.0F, 90.0F, 105.0F, 120.0F, 135.0F,
529 150.0F, 0.0F, 17.0F, 34.0F, 51.0F, 68.0F, 85.0F, 102.0F, 119.0F, 136.0F,
530 153.0F, 170.0F, 0.0F, 20.0F, 40.0F, 60.0F, 80.0F, 100.0F, 120.0F, 140.0F,
531 160.0F, 180.0F, 200.0F, 0.0F, 20.0F, 40.0F, 60.0F, 80.0F, 100.0F, 120.0F,
532 140.0F, 160.0F, 180.0F, 200.0F, 0.0F, 20.0F, 40.0F, 60.0F, 80.0F, 100.0F,
533 120.0F, 140.0F, 160.0F, 180.0F, 200.0F, 0.0F, 20.0F, 40.0F, 60.0F, 80.0F,
534 100.0F, 120.0F, 140.0F, 160.0F, 180.0F, 200.0F, 0.0F, 18.0F, 36.0F, 54.0F,
535 72.0F, 90.0F, 108.0F, 126.0F, 144.0F, 162.0F, 180.0F, 0.0F, 16.0F, 32.0F,
536 48.0F, 64.0F, 80.0F, 96.0F, 112.0F, 128.0F, 144.0F, 160.0F, 0.0F, 16.0F,
537 32.0F, 48.0F, 64.0F, 80.0F, 96.0F, 112.0F, 128.0F, 144.0F, 160.0F },
538
539 /* Expression: BreakpointsForDimension1
540 * Referenced by: '<S14>/SportMode'
541 */
542 { 0.0F, 10.0F, 20.0F, 30.0F, 40.0F, 50.0F, 60.0F, 70.0F, 80.0F, 90.0F, 100.0F
543 },
544
545 /* Expression: BreakpointsForDimension2
546 * Referenced by: '<S14>/SportMode'
547 */
548 { 0.0F, 10.0F, 20.0F, 30.0F, 40.0F, 60.0F, 80.0F, 100.0F, 125.0F, 150.0F },
549
550 /* Expression: Table
551 * Referenced by: '<S11>/EcoMode'
552 */
553 { 0.0F, 3.0F, 6.0F, 9.0F, 12.0F, 15.0F, 18.0F, 21.0F, 24.0F, 27.0F, 30.0F,
554 0.0F, 5.0F, 10.0F, 15.0F, 20.0F, 25.0F, 30.0F, 35.0F, 40.0F, 45.0F, 50.0F,
555 0.0F, 8.0F, 16.0F, 24.0F, 32.0F, 40.0F, 48.0F, 56.0F, 64.0F, 72.0F, 80.0F,
556 0.0F, 12.0F, 24.0F, 36.0F, 48.0F, 60.0F, 72.0F, 84.0F, 96.0F, 108.0F, 120.0F,
557 0.0F, 13.0F, 26.0F, 39.0F, 52.0F, 65.0F, 78.0F, 91.0F, 104.0F, 117.0F,

```

```

558 130.0F, 0.0F, 15.0F, 30.0F, 45.0F, 60.0F, 75.0F, 90.0F, 105.0F, 120.0F,
559 135.0F, 150.0F, 0.0F, 15.0F, 30.0F, 45.0F, 60.0F, 75.0F, 90.0F, 105.0F,
560 120.0F, 135.0F, 150.0F, 0.0F, 13.0F, 26.0F, 39.0F, 52.0F, 65.0F, 78.0F,
561 91.0F, 104.0F, 117.0F, 130.0F, 0.0F, 13.0F, 26.0F, 39.0F, 52.0F, 65.0F,
562 78.0F, 91.0F, 104.0F, 117.0F, 130.0F, 0.0F, 12.0F, 24.0F, 36.0F, 48.0F,
563 60.0F, 72.0F, 84.0F, 96.0F, 108.0F, 120.0F },
564
565 /* Expression: BreakpointsForDimension1
566 * Referenced by: '<S11>/EcoMode'
567 */
568 { 0.0F, 10.0F, 20.0F, 30.0F, 40.0F, 50.0F, 60.0F, 70.0F, 80.0F, 90.0F, 100.0F
569 },
570
571 /* Expression: BreakpointsForDimension2
572 * Referenced by: '<S11>/EcoMode'
573 */
574 { 0.0F, 10.0F, 20.0F, 30.0F, 40.0F, 60.0F, 80.0F, 100.0F, 125.0F, 150.0F },
575
576 /* Computed Parameter: Constant_Value_d
577 * Referenced by: '<S10>/Constant'
578 */
579 0.0F,
580
581 /* Computed Parameter: sec1_Value_o
582 * Referenced by: '<S5>/sec1'
583 */
584 6.78584F,
585
586 /* Computed Parameter: sec_Value_e
587 * Referenced by: '<S5>/sec'
588 */
589 60.0F,
590
591 /* Computed Parameter: TransmRat_Value
592 * Referenced by: '<S5>/TransmRat'
593 */
594 8.0F,
595
596 /* Computed Parameter: TrqReq_Y0
597 * Referenced by: '<S5>/TrqReq'
598 */
599 0.0F,
600
601 /* Computed Parameter: VehSpeedPrsnt_Value
602 * Referenced by: '<S5>/VehSpeedPrsnt'
603 */
604 0.0F,
605
606 /* Computed Parameter: Switch1_Threshold_e
607 * Referenced by: '<S5>/Switch1'
608 */
609 0.0F,
610
611 /* Computed Parameter: Merge_InitialOutput
612 * Referenced by: '<S5>/Merge'
613 */
614 0.0F,
615
616 /* Computed Parameter: Constant_Value_ij
617 * Referenced by: '<S4>/Constant'
618 */
619 0.0F,
620
621 /* Computed Parameter: NormalMode_maxIndex
622 * Referenced by: '<S12>/NormalMode'
623 */
624 { 10U, 9U },

```

```

625
626 /* Computed Parameter: SportMode_maxIndex
627 * Referenced by: '<S14>/SportMode'
628 */
629 { 10U, 9U },
630
631 /* Computed Parameter: EcoMode_maxIndex
632 * Referenced by: '<S11>/EcoMode'
633 */
634 { 10U, 9U },
635
636 /* Computed Parameter: Delay1_InitialCondition
637 * Referenced by: '<S2>/Delay1'
638 */
639 false,
640
641 /* Computed Parameter: FanReq_Y0
642 * Referenced by: '<S1>/FanReq'
643 */
644 false,
645
646 /* Computed Parameter: Fault_st_Y0
647 * Referenced by: '<S6>/Fault_st'
648 */
649 false,
650
651 /* Computed Parameter: LED01_Y0
652 * Referenced by: '<S4>/LED01'
653 */
654 false,
655
656 /* Computed Parameter: Constant_Value_c
657 * Referenced by: '<S32>/Constant'
658 */
659 0U,
660
661 /* Computed Parameter: Constant_Value_o
662 * Referenced by: '<S34>/Constant'
663 */
664 0U,
665
666 /* Computed Parameter: Output_InitialCondition
667 * Referenced by: '<S29>/Output'
668 */
669 0U,
670
671 /* Computed Parameter: Output_InitialCondition_o
672 * Referenced by: '<S30>/Output'
673 */
674 0U,
675
676 /* Computed Parameter: FixPtConstant_Value
677 * Referenced by: '<S31>/FixPt Constant'
678 */
679 1U,
680
681 /* Computed Parameter: FixPtConstant_Value_g
682 * Referenced by: '<S33>/FixPt Constant'
683 */
684 1U
685 };
686
687 /* Block signals (default storage) */
688 B_Torque_Control_ESP32_V6_T Torque_Control_ESP32_V6_B;
689
690 /* Block states (default storage) */
691 DW_Torque_Control_ESP32_V6_T Torque_Control_ESP32_V6_DW;

```

```

692
693 /* Previous zero-crossings (trigger) states */
694 PrevZCX_Torque_Control_ESP32__T Torque_Control_ESP32_V6_PrevZCX;
695
696 /* Real-time model */
697 static RT_MODEL_Torque_Control_ESP32_T Torque_Control_ESP32_V6_M_;
698 RT_MODEL_Torque_Control_ESP32_T *const Torque_Control_ESP32_V6_M =
699 &Torque_Control_ESP32_V6_M_;
700 static real32_T look2_iflf_linlcapw(real32_T u0, real32_T u1, const real32_T
701 bp0[], const real32_T bp1[], const real32_T table[], const uint32_T maxIndex[],
702 uint32_T stride);
703 static real32_T look1_iflf_linlcapw(real32_T u0, const real32_T bp0[], const
704 real32_T table[], uint32_T maxIndex);
705 static ZCEventType rt_ZCFcn(ZCDirection zcDir, ZCSigState *prevZc, real_T
706 currValue);
707
708 #define NOT_USING_NONFINITE_LITERAL 1
709
710 extern real_T rtInf;
711 extern real_T rtMinusInf;
712 extern real_T rtNaN;
713 extern real32_T rtInfF;
714 extern real32_T rtMinusInfF;
715 extern real32_T rtNaNF;
716 static void rt_InitlffAndNaN(size_t realSize);
717 static boolean_T rtIsInf(real_T value);
718 static boolean_T rtIsInfF(real32_T value);
719 static boolean_T rtIsNaN(real_T value);
720 static boolean_T rtIsNaNF(real32_T value);
721 typedef struct {
722 struct {
723 uint32_T wordH;
724 uint32_T wordL;
725 } words;
726 } BigEndianIEEEDouble;
727
728 typedef struct {
729 struct {
730 uint32_T wordL;
731 uint32_T wordH;
732 } words;
733 } LittleEndianIEEEDouble;
734
735 typedef struct {
736 union {
737 real32_T wordReal;
738 uint32_T wordLuint;
739 } wordL;
740 } IEEESingle;
741
742 real_T rtInf;
743 real_T rtMinusInf;
744 real_T rtNaN;
745 real32_T rtInfF;
746 real32_T rtMinusInfF;
747 real32_T rtNaNF;
748 static real_T rtGetInf(void);
749 static real32_T rtGetInfF(void);
750 static real_T rtGetMinusInf(void);
751 static real32_T rtGetMinusInfF(void);
752 static real_T rtGetNaN(void);
753 static real32_T rtGetNaNF(void);
754
755 /* Detect zero crossings events. */
756 static ZCEventType rt_ZCFcn(ZCDirection zcDir, ZCSigState *prevZc, real_T
757 currValue)
758 {

```

```

759 slZcEventType zcsDir;
760 slZcEventType tempEv;
761 ZCEventType zcEvent = NO_ZCEVENT; /* assume */
762
763 /* zcEvent matrix */
764 static const slZcEventType eventMatrix[4][4] = {
765 /* ZER POS NEG UNK */
766 { SL_ZCS_EVENT_NUL, SL_ZCS_EVENT_Z2P, SL_ZCS_EVENT_Z2N, SL_ZCS_EVENT_NUL },/* ZER */
767
768 { SL_ZCS_EVENT_P2Z, SL_ZCS_EVENT_NUL, SL_ZCS_EVENT_P2N, SL_ZCS_EVENT_NUL },/* POS */
769
770 { SL_ZCS_EVENT_N2Z, SL_ZCS_EVENT_N2P, SL_ZCS_EVENT_NUL, SL_ZCS_EVENT_NUL },/* NEG */
771
772 { SL_ZCS_EVENT_NUL, SL_ZCS_EVENT_NUL, SL_ZCS_EVENT_NUL, SL_ZCS_EVENT_NUL }/* UNK */
773 };
774
775 /* get prevZcEvent and prevZcSign from prevZc */
776 const slZcEventType prevEv = (slZcEventType)((uint8_T)(*prevZc)) >> 2;
777 const slZcSignalSignType prevSign = (slZcSignalSignType)((uint8_T)(*prevZc))
778 & (uint8_T)0x03;
779
780 /* get current zcSignal sign from current zcSignal value */
781 const slZcSignalSignType currSign = (slZcSignalSignType)((currValue) > 0.0 ?
782 SL_ZCS_SIGN_POS :
783 ((currValue) < 0.0 ? SL_ZCS_SIGN_NEG : SL_ZCS_SIGN_ZERO));
784
785 /* get current zcEvent based on prev and current zcSignal value */
786 slZcEventType currEv = eventMatrix[prevSign][currSign];
787
788 /* get slZcEventType from ZCDirection */
789 switch (zcDir) {
790 case ANY_ZERO_CROSSING:
791 zcsDir = SL_ZCS_EVENT_ALL;
792 break;
793
794 case FALLING_ZERO_CROSSING:
795 zcsDir = SL_ZCS_EVENT_ALL_DN;
796 break;
797
798 case RISING_ZERO_CROSSING:
799 zcsDir = SL_ZCS_EVENT_ALL_UP;
800 break;
801
802 default:
803 zcsDir = SL_ZCS_EVENT_NUL;
804 break;
805 }
806
807 /* had event, check if zc happened */
808 if (slZcHadEvent(currEv, zcsDir)) {
809 currEv = (slZcEventType)(slZcUnAliasEvents(prevEv, currEv));
810 } else {
811 currEv = SL_ZCS_EVENT_NUL;
812 }
813
814 /* Update prevZc */
815 tempEv = (slZcEventType)(currEv << 2);/* shift left by 2 bits */
816 *prevZc = (ZCSigState)((currSign) | (tempEv));
817 if ((currEv & SL_ZCS_EVENT_ALL_DN) != 0) {
818 zcEvent = FALLING_ZCEVENT;
819 } else if ((currEv & SL_ZCS_EVENT_ALL_UP) != 0) {
820 zcEvent = RISING_ZCEVENT;
821 } else {

```

```

822 zcEvent = NO_ZCEVENT;
823 }
824
825 return zcEvent;
826 } /* rt_ZCFcn */
827
828 /*
829 * Initialize the rtInf, rtMinusInf, and rtNaN needed by the
830 * generated code. NaN is initialized as non-signaling. Assumes IEEE.
831 */
832 static void rt_InitInfAndNaN(size_t realSize)
833 {
834 (void)(realSize);
835 rtNaN = rtGetNaN();
836 rtNaNF = rtGetNaNF();
837 rtInf = rtGetInf();
838 rtInfF = rtGetInfF();
839 rtMinusInf = rtGetMinusInf();
840 rtMinusInfF = rtGetMinusInfF();
841 }
842
843 /* Test if value is infinite */
844 static boolean_T rtIsInf(real_T value)
845 {
846 return (boolean_T)((value==rtInf || value==rtMinusInf) ? 1U : 0U);
847 }
848
849 /* Test if single-precision value is infinite */
850 static boolean_T rtIsInfF(real32_T value)
851 {
852 return (boolean_T)((value)==rtInfF || (value)==rtMinusInfF) ? 1U : 0U);
853 }
854
855 /* Test if value is not a number */
856 static boolean_T rtIsNaN(real_T value)
857 {
858 boolean_T result = (boolean_T) 0;
859 size_t bitsPerReal = sizeof(real_T) * (NumBitsPerChar);
860 if (bitsPerReal == 32U) {
861 result = rtIsNaNF((real32_T)value);
862 } else {
863 union {
864 LittleEndianIEEEDouble bitVal;
865 real_T fltVal;
866 } tmpVal;
867
868 tmpVal.fltVal = value;
869 result = (boolean_T)((tmpVal.bitVal.words.wordH & 0x7FF00000) == 0x7FF00000 &&
870 ((tmpVal.bitVal.words.wordH & 0x000FFFFF) != 0 ||
871 (tmpVal.bitVal.words.wordL != 0));
872 }
873
874 return result;
875 }
876
877 /* Test if single-precision value is not a number */
878 static boolean_T rtIsNaNF(real32_T value)
879 {
880 IEEESingle tmp;
881 tmp.wordL.wordLreal = value;
882 return (boolean_T)((tmp.wordL.wordLuint & 0x7F800000) == 0x7F800000 &&
883 (tmp.wordL.wordLuint & 0x007FFFFFF) != 0 );
884 }
885
886 /*
887 * Initialize rtInf needed by the generated code.
888 * Inf is initialized as non-signaling. Assumes IEEE.

```

```

889 */
890 static real_T rtGetInf(void)
891 {
892 size_t bitsPerReal = sizeof(real_T) * (NumBitsPerChar);
893 real_T inf = 0.0;
894 if (bitsPerReal == 32U) {
895 inf = rtGetInfF();
896 } else {
897 union {
898 LittleEndianIEEEDouble bitVal;
899 real_T fltVal;
900 } tmpVal;
901
902 tmpVal.bitVal.words.wordH = 0x7FF00000U;
903 tmpVal.bitVal.words.wordL = 0x00000000U;
904 inf = tmpVal.fltVal;
905 }
906
907 return inf;
908 }
909
910 /*
911 * Initialize rtInfF needed by the generated code.
912 * Inf is initialized as non-signaling. Assumes IEEE.
913 */
914 static real32_T rtGetInfF(void)
915 {
916 IEEESingle infF;
917 infF.wordL.wordLuint = 0x7F800000U;
918 return infF.wordL.wordLreal;
919 }
920
921 /*
922 * Initialize rtMinusInf needed by the generated code.
923 * Inf is initialized as non-signaling. Assumes IEEE.
924 */
925 static real_T rtGetMinusInf(void)
926 {
927 size_t bitsPerReal = sizeof(real_T) * (NumBitsPerChar);
928 real_T minf = 0.0;
929 if (bitsPerReal == 32U) {
930 minf = rtGetMinusInfF();
931 } else {
932 union {
933 LittleEndianIEEEDouble bitVal;
934 real_T fltVal;
935 } tmpVal;
936
937 tmpVal.bitVal.words.wordH = 0xFFFF0000U;
938 tmpVal.bitVal.words.wordL = 0x00000000U;
939 minf = tmpVal.fltVal;
940 }
941
942 return minf;
943 }
944
945 /*
946 * Initialize rtMinusInfF needed by the generated code.
947 * Inf is initialized as non-signaling. Assumes IEEE.
948 */
949 static real32_T rtGetMinusInfF(void)
950 {
951 IEEESingle minfF;
952 minfF.wordL.wordLuint = 0xFF800000U;
953 return minfF.wordL.wordLreal;
954 }
955

```

```

956 /*
957 * Initialize rtNaN needed by the generated code.
958 * NaN is initialized as non-signaling. Assumes IEEE.
959 */
960 static real_T rtGetNaN(void)
961 {
962 size_t bitsPerReal = sizeof(real_T) * (NumBitsPerChar);
963 real_T nan = 0.0;
964 if (bitsPerReal == 32U) {
965 nan = rtGetNaNF();
966 } else {
967 union {
968 LittleEndianIEEEDouble bitVal;
969 real_T fltVal;
970 } tmpVal;
971
972 tmpVal.bitVal.words.wordH = 0xFFFF800000U;
973 tmpVal.bitVal.words.wordL = 0x00000000U;
974 nan = tmpVal.fltVal;
975 }
976
977 return nan;
978 }
979
980 /*
981 * Initialize rtNaNF needed by the generated code.
982 * NaN is initialized as non-signaling. Assumes IEEE.
983 */
984 static real32_T rtGetNaNF(void)
985 {
986 IEEESingle nanF = { { 0.0F } };
987
988 nanF.wordL.wordLuint = 0xFFC00000U;
989 return nanF.wordL.wordLreal;
990 }
991
992 static real32_T look2_iflf_linlcapw(real32_T u0, real32_T u1, const real32_T
993 bp0[], const real32_T bp1[], const real32_T table[], const uint32_T maxIndex[],
994 uint32_T stride)
995 {
996 real32_T fractions[2];
997 real32_T frac;
998 real32_T y;
999 real32_T yL_0d0;
1000 uint32_T bplIndices[2];
1001 uint32_T bpldx;
1002 uint32_T offset_1d;
1003
1004 /* Column-major Lookup 2-D
1005 Search method: 'linear'
1006 Use previous index: 'off'
1007 Interpolation method: 'Linear point-slope'
1008 Extrapolation method: 'Clip'
1009 Use last breakpoint for index at or above upper limit: 'on'
1010 Remove protection against out-of-range input in generated code: 'off'
1011 */
1012 /* Prelookup - Index and Fraction
1013 Index Search method: 'linear'
1014 Extrapolation method: 'Clip'
1015 Use previous index: 'off'
1016 Use last breakpoint for index at or above upper limit: 'on'
1017 Remove protection against out-of-range input in generated code: 'off'
1018 */
1019 if (u0 <= bp0[0U]) {
1020 bpldx = 0U;
1021 frac = 0.0F;
1022 } else if (u0 < bp0[maxIndex[0U]]) {

```

```

1023 /* Linear Search */
1024 for (bpldx = maxIndex[0U] >> 1U; u0 < bp0[bpldx]; bpldx--) {
1025 }
1026
1027 while (u0 >= bp0[bpldx + 1U]) {
1028 bpldx++;
1029 }
1030
1031 frac = (u0 - bp0[bpldx]) / (bp0[bpldx + 1U] - bp0[bpldx]);
1032 } else {
1033 bpldx = maxIndex[0U];
1034 frac = 0.0F;
1035 }
1036
1037 fractions[0U] = frac;
1038 bplIndices[0U] = bpldx;
1039
1040 /* Prelookup - Index and Fraction
1041 Index Search method: 'linear'
1042 Extrapolation method: 'Clip'
1043 Use previous index: 'off'
1044 Use last breakpoint for index at or above upper limit: 'on'
1045 Remove protection against out-of-range input in generated code: 'off'
1046 */
1047 if (u1 <= bp1[0U]) {
1048 bpldx = 0U;
1049 frac = 0.0F;
1050 } else if (u1 < bp1[maxIndex[1U]]) {
1051 /* Linear Search */
1052 for (bpldx = maxIndex[1U] >> 1U; u1 < bp1[bpldx]; bpldx--) {
1053 }
1054
1055 while (u1 >= bp1[bpldx + 1U]) {
1056 bpldx++;
1057 }
1058
1059 frac = (u1 - bp1[bpldx]) / (bp1[bpldx + 1U] - bp1[bpldx]);
1060 } else {
1061 bpldx = maxIndex[1U];
1062 frac = 0.0F;
1063 }
1064
1065 /* Column-major Interpolation 2-D
1066 Interpolation method: 'Linear point-slope'
1067 Use last breakpoint for index at or above upper limit: 'on'
1068 Overflow mode: 'portable wrapping'
1069 */
1070 offset_1d = bpldx * stride + bplIndices[0U];
1071 if (bplIndices[0U] == maxIndex[0U]) {
1072 y = table[offset_1d];
1073 } else {
1074 yL_0d0 = table[offset_1d];
1075 y = (table[offset_1d + 1U] - yL_0d0) * fractions[0U] + yL_0d0;
1076 }
1077
1078 if (bpldx == maxIndex[1U]) {
1079 } else {
1080 bpldx = offset_1d + stride;
1081 if (bplIndices[0U] == maxIndex[0U]) {
1082 yL_0d0 = table[bpldx];
1083 } else {
1084 yL_0d0 = table[bpldx];
1085 yL_0d0 += (table[bpldx + 1U] - yL_0d0) * fractions[0U];
1086 }
1087
1088 y += (yL_0d0 - y) * frac;
1089 }

```

```

1090
1091 return y;
1092 }
1093
1094 static real32_T look1_iflf_linlcawp(real32_T u0, const real32_T bp0[], const
1095 real32_T table[], uint32_T maxIndex)
1096 {
1097 real32_T frac;
1098 real32_T y;
1099 real32_T yL_0d0;
1100 uint32_T bpldx;
1101
1102 /* Column-major Lookup 1-D
1103 Search method: 'linear'
1104 Use previous index: 'off'
1105 Interpolation method: 'Linear point-slope'
1106 Extrapolation method: 'Clip'
1107 Use last breakpoint for index at or above upper limit: 'on'
1108 Remove protection against out-of-range input in generated code: 'off'
1109 */
1110 /* Prelookup - Index and Fraction
1111 Index Search method: 'linear'
1112 Extrapolation method: 'Clip'
1113 Use previous index: 'off'
1114 Use last breakpoint for index at or above upper limit: 'on'
1115 Remove protection against out-of-range input in generated code: 'off'
1116 */
1117 if (u0 <= bp0[0U]) {
1118 bpldx = 0U;
1119 frac = 0.0F;
1120 } else if (u0 < bp0[maxIndex]) {
1121 /* Linear Search */
1122 for (bpldx = maxIndex >> 1U; u0 < bp0[bpldx]; bpldx--) {
1123 }
1124
1125 while (u0 >= bp0[bpldx + 1U]) {
1126 bpldx++;
1127 }
1128
1129 frac = (u0 - bp0[bpldx]) / (bp0[bpldx + 1U] - bp0[bpldx]);
1130 } else {
1131 bpldx = maxIndex;
1132 frac = 0.0F;
1133 }
1134
1135 /* Column-major Interpolation 1-D
1136 Interpolation method: 'Linear point-slope'
1137 Use last breakpoint for index at or above upper limit: 'on'
1138 Overflow mode: 'portable wrapping'
1139 */
1140 if (bpldx == maxIndex) {
1141 y = table[bpldx];
1142 } else {
1143 yL_0d0 = table[bpldx];
1144 y = (table[bpldx + 1U] - yL_0d0) * frac + yL_0d0;
1145 }
1146
1147 return y;
1148 }
1149
1150 /* Model step function */
1151 void Torque_Control_ESP32_V6_step(void)
1152 {
1153 int32_T idxDelay;
1154 real32_T den;
1155 uint32_T iterStart;
1156 uint16_T b_varargout_1;

```

```

1157 boolean_T c_value;
1158 ZCEventType zcEvent;
1159
1160 /* S-Function (fcgen): '<Root>/Function-Call Generator1' incorporates:
1161 * SubSystem: '<Root>/DataAcquisition'
1162 */
1163 /* MATLABSystem: '<S2>/Analog Input4' */
1164 if (Torque_Control_ESP32_V6_DW.obj_ac.SampleTime != 
1165 Torque_Control_ESP32_V6_P.AnalogInput4_SampleTime) {
1166 Torque_Control_ESP32_V6_DW.obj_ac.SampleTime =
1167 Torque_Control_ESP32_V6_P.AnalogInput4_SampleTime;
1168 }
1169
1170 Torque_Control_ESP32_V6_DW.obj_ac.AnalogInDriverObj.MW_ANALOGIN_HANDLE =
1171 MW_AnalogIn_GetHandle(36U);
1172 MW_AnalogInSingle_ReadResult
1173 (Torque_Control_ESP32_V6_DW.obj_ac.AnalogInDriverObj.MW_ANALOGIN_HANDLE,
1174 &b_varargout_1, MW_ANALOGIN_UINT16);
1175
1176 /* MATLABSystem: '<S2>/Analog Input4' */
1177 Torque_Control_ESP32_V6_B.AnalogInput4 = b_varargout_1;
1178
1179 /* MATLABSystem: '<S2>/Analog Input5' */
1180 if (Torque_Control_ESP32_V6_DW.obj_a.SampleTime != 
1181 Torque_Control_ESP32_V6_P.AnalogInput5_SampleTime) {
1182 Torque_Control_ESP32_V6_DW.obj_a.SampleTime =
1183 Torque_Control_ESP32_V6_P.AnalogInput5_SampleTime;
1184 }
1185
1186 Torque_Control_ESP32_V6_DW.obj_a.AnalogInDriverObj.MW_ANALOGIN_HANDLE =
1187 MW_AnalogIn_GetHandle(39U);
1188 MW_AnalogInSingle_ReadResult
1189 (Torque_Control_ESP32_V6_DW.obj_a.AnalogInDriverObj.MW_ANALOGIN_HANDLE,
1190 &b_varargout_1, MW_ANALOGIN_UINT16);
1191
1192 /* MATLABSystem: '<S2>/Analog Input5' */
1193 Torque_Control_ESP32_V6_B.AnalogInput5 = b_varargout_1;
1194
1195 /* Switch: '<S2>/Switch' incorporates:
1196 * Constant: '<S2>/Constant1'
1197 */
1198 if (Torque_Control_ESP32_V6_P.Constant1_Value >
1199 Torque_Control_ESP32_V6_P.Switch_Threshold) {
1200 /* Switch: '<S2>/Switch' incorporates:
1201 * Constant: '<S2>/Constant12'
1202 */
1203 Torque_Control_ESP32_V6_B.AccPed01 =
1204 Torque_Control_ESP32_V6_P.Constant12_Value;
1205 } else {
1206 /* DataTypeConversion: '<S2>/Data Type Conversion4' */
1207 Torque_Control_ESP32_V6_B.DataTypeConversion4 =
1208 Torque_Control_ESP32_V6_B.AnalogInput4;
1209
1210 /* Lookup_n-D: '<S2>/AccPed1Curve' incorporates:
1211 * DataTypeConversion: '<S2>/Data Type Conversion4'
1212 */
1213 Torque_Control_ESP32_V6_B.AccPed1Curve = look1_iflf_linlcaw
1214 (Torque_Control_ESP32_V6_B.DataTypeConversion4,
1215 Torque_Control_ESP32_V6_P.AccPed1Curve_bp01Data,
1216 Torque_Control_ESP32_V6_P.AccPed1Curve_tableData, 1U);
1217
1218 /* Switch: '<S2>/Switch' */
1219 Torque_Control_ESP32_V6_B.AccPed01 = Torque_Control_ESP32_V6_B.AccPed1Curve;
1220 }
1221
1222 /* End of Switch: '<S2>/Switch' */
1223

```

```

1224 /* Switch: '<S2>/Switch2' incorporates:
1225 * Constant: '<S2>/Constant2'
1226 */
1227 if (Torque_Control_ESP32_V6_P.Constant2_Value >
1228 Torque_Control_ESP32_V6_P.Switch2_Threshold) {
1229 /* Product: '<S2>/Product' incorporates:
1230 * Constant: '<S2>/Constant10'
1231 * Constant: '<S2>/Constant12'
1232 */
1233 Torque_Control_ESP32_V6_B.Product =
1234 Torque_Control_ESP32_V6_P.Constant12_Value *
1235 Torque_Control_ESP32_V6_P.Constant10_Value;
1236
1237 /* Switch: '<S2>/Switch2' */
1238 Torque_Control_ESP32_V6_B.AccPed02 = (real32_T)
1239 Torque_Control_ESP32_V6_B.Product;
1240 } else {
1241 /* DataTypeConversion: '<S2>/Data Type Conversion3' */
1242 Torque_Control_ESP32_V6_B.DataTypeConversion3 =
1243 Torque_Control_ESP32_V6_B.AnalogInput5;
1244
1245 /* Lookup_n-D: '<S2>/AccPed2Curve' incorporates:
1246 * DataTypeConversion: '<S2>/Data Type Conversion3'
1247 */
1248 Torque_Control_ESP32_V6_B.AccPed2Curve = look1_ifl_linlcawp
1249 (Torque_Control_ESP32_V6_B.DataTypeConversion3,
1250 Torque_Control_ESP32_V6_P.AccPed2Curve_bp01Data,
1251 Torque_Control_ESP32_V6_P.AccPed2Curve_tableData, 1U);
1252
1253 /* Switch: '<S2>/Switch2' */
1254 Torque_Control_ESP32_V6_B.AccPed02 = Torque_Control_ESP32_V6_B.AccPed2Curve;
1255 }
1256
1257 /* End of Switch: '<S2>/Switch2' */
1258
1259 /* MATLABSystem: '<S2>/Digital Input' */
1260 if (Torque_Control_ESP32_V6_DW.obj_ny.SampleTime != 0) {
1261 Torque_Control_ESP32_V6_P.DigitalInput_SampleTime = 1;
1262 Torque_Control_ESP32_V6_DW.obj_ny.SampleTime =
1263 Torque_Control_ESP32_V6_P.DigitalInput_SampleTime;
1264 }
1265
1266 c_value = readDigitalPin(25);
1267
1268 /* MATLABSystem: '<S2>/Digital Input' */
1269 Torque_Control_ESP32_V6_B.DigitalInput = c_value;
1270
1271 /* Switch: '<S2>/Switch4' incorporates:
1272 * Constant: '<S2>/Constant4'
1273 */
1274 if (Torque_Control_ESP32_V6_P.Constant4_Value >
1275 Torque_Control_ESP32_V6_P.Switch4_Threshold) {
1276 /* Switch: '<S2>/Switch4' incorporates:
1277 * Constant: '<S2>/Constant16'
1278 */
1279 Torque_Control_ESP32_V6_B.Switch4 = (real32_T)
1280 Torque_Control_ESP32_V6_P.Constant16_Value;
1281 } else {
1282 /* Switch: '<S2>/Switch4' */
1283 Torque_Control_ESP32_V6_B.Switch4 = Torque_Control_ESP32_V6_B.DigitalInput;
1284 }
1285
1286 /* End of Switch: '<S2>/Switch4' */
1287
1288 /* MATLABSystem: '<S2>/Analog Input7' */
1289 if (Torque_Control_ESP32_V6_DW.obj.SampleTime != 0) {
1290 Torque_Control_ESP32_V6_P.AnalogInput7_SampleTime = 1;
1291 }
```

```

1291 Torque_Control_ESP32_V6_DW.obj.SampleTime =
1292 Torque_Control_ESP32_V6_P.AnalogInput7_SampleTime;
1293 }
1294
1295 Torque_Control_ESP32_V6_DW.obj.AnalogInDriverObj.MW_ANALOGIN_HANDLE =
1296 MW_AnalogIn_GetHandle(35U);
1297 MW_AnalogInSingle_ReadResult
1298 (Torque_Control_ESP32_V6_DW.obj.AnalogInDriverObj.MW_ANALOGIN_HANDLE,
1299 &b_varargout_1, MW_ANALOGIN_UINT16);
1300
1301 /* MATLABSystem: '<S2>/Analog Input7' */
1302 Torque_Control_ESP32_V6_B.AnalogInput7 = b_varargout_1;
1303
1304 /* Switch: '<S2>/Switch1' incorporates:
1305 * Constant: '<S2>/Constant'
1306 */
1307 if (Torque_Control_ESP32_V6_P.Constant_Value >
1308 Torque_Control_ESP32_V6_P.Switch1_Threshold) {
1309 /* Switch: '<S2>/Switch1' incorporates:
1310 * Constant: '<S2>/Constant13'
1311 */
1312 Torque_Control_ESP32_V6_B.Switch1_b = (real32_T)
1313 Torque_Control_ESP32_V6_P.Constant13_Value;
1314 } else {
1315 /* DataTypeConversion: '<S2>/Data Type Conversion1' */
1316 Torque_Control_ESP32_V6_B.DataTypeConversion1 =
1317 Torque_Control_ESP32_V6_B.AnalogInput7;
1318
1319 /* Lookup_n-D: '<S2>/Curve3' incorporates:
1320 * DataTypeConversion: '<S2>/Data Type Conversion1'
1321 */
1322 Torque_Control_ESP32_V6_B.Curve3 = look1_iflf_linlcaw
1323 (Torque_Control_ESP32_V6_B.DataTypeConversion1,
1324 Torque_Control_ESP32_V6_P.Curve3_bp01Data,
1325 Torque_Control_ESP32_V6_P.Curve3_tableData, 1U);
1326
1327 /* Switch: '<S2>/Switch1' */
1328 Torque_Control_ESP32_V6_B.Switch1_b = Torque_Control_ESP32_V6_B.Curve3;
1329 }
1330
1331 /* End of Switch: '<S2>/Switch1' */
1332
1333 /* MATLABSystem: '<S2>/Digital Input1' */
1334 if (Torque_Control_ESP32_V6_DW.obj_g.SampleTime !=
1335 Torque_Control_ESP32_V6_P.DigitalInput1_SampleTime) {
1336 Torque_Control_ESP32_V6_DW.obj_g.SampleTime =
1337 Torque_Control_ESP32_V6_P.DigitalInput1_SampleTime;
1338 }
1339
1340 c_value = readDigitalPin(26);
1341
1342 /* MATLABSystem: '<S2>/Digital Input1' */
1343 Torque_Control_ESP32_V6_B.DigitalInput1 = c_value;
1344
1345 /* Switch: '<S2>/Switch5' incorporates:
1346 * Constant: '<S2>/Constant5'
1347 */
1348 if (Torque_Control_ESP32_V6_P.Constant5_Value >
1349 Torque_Control_ESP32_V6_P.Switch5_Threshold) {
1350 /* Switch: '<S2>/Switch5' incorporates:
1351 * Constant: '<S2>/Constant15'
1352 */
1353 Torque_Control_ESP32_V6_B.RTD_st = (real32_T)
1354 Torque_Control_ESP32_V6_P.Constant15_Value;
1355 } else {
1356 /* Switch: '<S2>/Switch5' */
1357 Torque_Control_ESP32_V6_B.RTD_st = Torque_Control_ESP32_V6_B.DigitalInput1;

```

```

1358 }
1359 /* End of Switch: '<S2>/Switch5' */
1360
1361
1362 /* MATLABSystem: '<S2>/Digital Input2' */
1363 if (Torque_Control_ESP32_V6_DW.obj_n.SampleTime !=  

1364 Torque_Control_ESP32_V6_P.DigitalInput2_SampleTime) {  

1365 Torque_Control_ESP32_V6_DW.obj_n.SampleTime =  

1366 Torque_Control_ESP32_V6_P.DigitalInput2_SampleTime;  

1367 }
1368
1369 c_value = readDigitalPin(27);
1370
1371 /* MATLABSystem: '<S2>/Digital Input2' */
1372 Torque_Control_ESP32_V6_B.DigitalInput2 = c_value;
1373
1374 /* Delay: '<S2>/Delay1' */
1375 Torque_Control_ESP32_V6_B.Delay1 = Torque_Control_ESP32_V6_DW.Delay1_DSTATE[0];
1376
1377 /* Delay: '<S2>/Delay' */
1378 Torque_Control_ESP32_V6_B.Delay = Torque_Control_ESP32_V6_DW.Delay_DSTATE[0];
1379
1380 /* Switch: '<S2>/Switch6' incorporates:
1381 * Constant: '<S2>/Constant6'
1382 */
1383 if (Torque_Control_ESP32_V6_P.Constant6_Value >  

1384 Torque_Control_ESP32_V6_P.Switch6_Threshold) {
1385 /* Switch: '<S2>/Switch6' incorporates:
1386 * Constant: '<S2>/Constant19'
1387 */
1388 Torque_Control_ESP32_V6_B.Switch6 =
1389 Torque_Control_ESP32_V6_P.Constant19_Value;
1390 } else {
1391 /* Logic: '<S2>/AND' */
1392 Torque_Control_ESP32_V6_B.AND_a = Torque_Control_ESP32_V6_B.DigitalInput2 &&  

1393 Torque_Control_ESP32_V6_B.Delay1;
1394
1395 /* Switch: '<S2>/Switch6' */
1396 Torque_Control_ESP32_V6_B.Switch6 = Torque_Control_ESP32_V6_B.AND_a;
1397 }
1398
1399 /* End of Switch: '<S2>/Switch6' */
1400
1401 /* Outputs for Triggered SubSystem: '<S2>/Triggered Subsystem' incorporates:
1402 * TriggerPort: '<S7>/Trigger'
1403 */
1404 zcEvent = rt_ZCFcn(RISING_ZERO_CROSSING,  

1405 &Torque_Control_ESP32_V6_PrevZCX.TriggeredSubsystem_Trig_ZCE,  

1406 (Torque_Control_ESP32_V6_B.Switch6));
1407 if (zcEvent != NO_ZCEVENT) {
1408 /* RelationalOperator: '<S8>/Compare' incorporates:
1409 * Constant: '<S8>/Constant'
1410 */
1411 Torque_Control_ESP32_V6_B.Compare_j = Torque_Control_ESP32_V6_B.Delay <=
1412 Torque_Control_ESP32_V6_P.CompareToConstant_const;
1413
1414 /* Switch: '<S7>/Switch' */
1415 if (Torque_Control_ESP32_V6_B.Compare_j) {
1416 /* Sum: '<S7>/Sum' incorporates:
1417 * Constant: '<S7>/Constant'
1418 */
1419 Torque_Control_ESP32_V6_B.Sum = Torque_Control_ESP32_V6_B.Delay +
1420 Torque_Control_ESP32_V6_P.Constant_Value_l;
1421
1422 /* Switch: '<S7>/Switch' */
1423 Torque_Control_ESP32_V6_B.Switch_h = Torque_Control_ESP32_V6_B.Sum;
1424 } else {

```

```

1425 /* Switch: '<S7>/Switch' incorporates:
1426 * Constant: '<S7>/Constant1'
1427 */
1428 Torque_Control_ESP32_V6_B.Switch_h =
1429 Torque_Control_ESP32_V6_P.Constant1_Value_h;
1430 }
1431
1432 /* End of Switch: '<S7>/Switch' */
1433 }
1434
1435 /* End of Outputs for SubSystem: '<S2>/Triggered Subsystem' */
1436
1437 /* MATLABSystem: '<S2>/Analog Input1' */
1438 if (Torque_Control_ESP32_V6_DW.obj_h.SampleTime !=
1439 Torque_Control_ESP32_V6_P.AnalogInput1_SampleTime) {
1440 Torque_Control_ESP32_V6_DW.obj_h.SampleTime =
1441 Torque_Control_ESP32_V6_P.AnalogInput1_SampleTime;
1442 }
1443
1444 Torque_Control_ESP32_V6_DW.obj_h.AnalogInDriverObj.MW_ANALOGIN_HANDLE =
1445 MW_AnalogIn_GetHandle(14U);
1446 MW_AnalogInSingle_ReadResult
1447 (Torque_Control_ESP32_V6_DW.obj_h.AnalogInDriverObj.MW_ANALOGIN_HANDLE,
1448 &b_varargout_1, MW_ANALOGIN_UINT16);
1449
1450 /* MATLABSystem: '<S2>/Analog Input1' */
1451 Torque_Control_ESP32_V6_B.AnalogInput1 = b_varargout_1;
1452
1453 /* MATLABSystem: '<S2>/Digital Input3' */
1454 if (Torque_Control_ESP32_V6_DW.obj_c.SampleTime !=
1455 Torque_Control_ESP32_V6_P.DigitalInput3_SampleTime) {
1456 Torque_Control_ESP32_V6_DW.obj_c.SampleTime =
1457 Torque_Control_ESP32_V6_P.DigitalInput3_SampleTime;
1458 }
1459
1460 c_value = readDigitalPin(4);
1461
1462 /* MATLABSystem: '<S2>/Digital Input3' */
1463 Torque_Control_ESP32_V6_B.DigitalInput3 = c_value;
1464
1465 /* Switch: '<S2>/Switch7' incorporates:
1466 * Constant: '<S2>/Constant11'
1467 * Constant: '<S2>/Constant7'
1468 * Switch: '<S2>/Switch9'
1469 */
1470 if (Torque_Control_ESP32_V6_P.Constant7_Value >
1471 Torque_Control_ESP32_V6_P.Switch7_Threshold) {
1472 /* Switch: '<S2>/Switch7' incorporates:
1473 * Constant: '<S2>/Constant9'
1474 */
1475 Torque_Control_ESP32_V6_B.Switch7 = (real32_T)
1476 Torque_Control_ESP32_V6_P.Constant9_Value;
1477 } else {
1478 if (Torque_Control_ESP32_V6_P.Constant11_Value >
1479 Torque_Control_ESP32_V6_P.Switch9_Threshold) {
1480 /* Switch: '<S2>/Switch9' */
1481 Torque_Control_ESP32_V6_B.Switch9 =
1482 Torque_Control_ESP32_V6_B.DigitalInput3;
1483 } else {
1484 /* DataTypeConversion: '<S2>/Data Type Conversion5' incorporates:
1485 * Switch: '<S2>/Switch9'
1486 */
1487 Torque_Control_ESP32_V6_B.DataTypeConversion5 =
1488 Torque_Control_ESP32_V6_B.AnalogInput1;
1489
1490 /* Lookup_n-D: '<S2>/Curve1' incorporates:
1491 * DataTypeConversion: '<S2>/Data Type Conversion5'

```

```

1492 /* Switch: '<S2>/Switch9'
1493 */
1494 Torque_Control_ESP32_V6_B.Curve1 = look1_iflf_linlcaw
1495 (Torque_Control_ESP32_V6_B.DataTypeConversion5,
1496 Torque_Control_ESP32_V6_P.Curve1_bp01Data,
1497 Torque_Control_ESP32_V6_P.Curve1_tableData, 1U);
1498
1499 /* Switch: '<S2>/Switch9' */
1500 Torque_Control_ESP32_V6_B.Switch9 = Torque_Control_ESP32_V6_B.Curve1;
1501 }
1502
1503 /* Switch: '<S2>/Switch7' */
1504 Torque_Control_ESP32_V6_B.Switch7 = Torque_Control_ESP32_V6_B.Switch9;
1505 }
1506
1507 /* End of Switch: '<S2>/Switch7' */
1508
1509 /* Switch: '<S2>/Switch8' incorporates:
1510 * Constant: '<S2>/Constant8'
1511 */
1512 if (Torque_Control_ESP32_V6_P.Constant8_Value >
1513 Torque_Control_ESP32_V6_P.Switch8_Threshold) {
1514 /* Switch: '<S2>/Switch8' incorporates:
1515 * Constant: '<S2>/VehSpd'
1516 */
1517 Torque_Control_ESP32_V6_B.Switch8 = (real32_T)
1518 Torque_Control_ESP32_V6_P.VehSpd_Value;
1519 } else {
1520 /* Product: '<S2>/Divide2' incorporates:
1521 * Constant: '<S2>/TransmRatio'
1522 */
1523 Torque_Control_ESP32_V6_B.Divide2 = Torque_Control_ESP32_V6_B.Switch7 /
1524 Torque_Control_ESP32_V6_P.TransmRatio_Value;
1525
1526 /* Product: '<S2>/Divide' incorporates:
1527 * Constant: '<S2>/sec'
1528 */
1529 Torque_Control_ESP32_V6_B.Divide = Torque_Control_ESP32_V6_B.Divide2 /
1530 Torque_Control_ESP32_V6_P.sec_Value;
1531
1532 /* Product: '<S2>/Divide1' incorporates:
1533 * Constant: '<S2>/sec1'
1534 */
1535 Torque_Control_ESP32_V6_B.Divide1 = Torque_Control_ESP32_V6_B.Divide *
1536 Torque_Control_ESP32_V6_P.sec1_Value;
1537
1538 /* Switch: '<S2>/Switch8' */
1539 Torque_Control_ESP32_V6_B.Switch8 = (real32_T)
1540 Torque_Control_ESP32_V6_B.Divide1;
1541 }
1542
1543 /* End of Switch: '<S2>/Switch8' */
1544
1545 /* MATLABSystem: '<S2>/Analog Input6' */
1546 if (Torque_Control_ESP32_V6_DW.obj_j.SampleTime !=
1547 Torque_Control_ESP32_V6_P.AnalogInput6_SampleTime) {
1548 Torque_Control_ESP32_V6_DW.obj_j.SampleTime =
1549 Torque_Control_ESP32_V6_P.AnalogInput6_SampleTime;
1550 }
1551
1552 Torque_Control_ESP32_V6_DW.obj_j.AnalogInDriverObj.MW_ANALOGIN_HANDLE =
1553 MW_AnalogIn_GetHandle(34U);
1554 MW_AnalogInSingle_ReadResult
1555 (Torque_Control_ESP32_V6_DW.obj_j.AnalogInDriverObj.MW_ANALOGIN_HANDLE,
1556 &b_varargout_1, MW_ANALOGIN_UINT16);
1557
1558 /* MATLABSystem: '<S2>/Analog Input6' */

```

```

1559 Torque_Control_ESP32_V6_B.AnalogInput6 = b_varargout_1;
1560
1561 /* Switch: '<S2>/Switch3' incorporates:
1562 * Constant: '<S2>/Constant3'
1563 */
1564 if (Torque_Control_ESP32_V6_P.Constant3_Value >
1565 Torque_Control_ESP32_V6_P.Switch3_Threshold) {
1566 /* Switch: '<S2>/Switch3' incorporates:
1567 * Constant: '<S2>/Constant14'
1568 */
1569 Torque_Control_ESP32_V6_B.Switch3 = (real32_T)
1570 Torque_Control_ESP32_V6_P.Constant14_Value;
1571 } else {
1572 /* DataTypeConversion: '<S2>/Data Type Conversion2' */
1573 Torque_Control_ESP32_V6_B.DataTypeConversion2 =
1574 Torque_Control_ESP32_V6_B.AnalogInput6;
1575
1576 /* Lookup_n-D: '<S2>/Curve2' incorporates:
1577 * DataTypeConversion: '<S2>/Data Type Conversion2'
1578 */
1579 Torque_Control_ESP32_V6_B.Curve2 = look1_ifff_linlcawp
1580 (Torque_Control_ESP32_V6_B.DataTypeConversion2,
1581 Torque_Control_ESP32_V6_P.Curve2_bp01Data,
1582 Torque_Control_ESP32_V6_P.Curve2_tableData, 1U);
1583
1584 /* Switch: '<S2>/Switch3' */
1585 Torque_Control_ESP32_V6_B.Switch3 = Torque_Control_ESP32_V6_B.Curve2;
1586 }
1587
1588 /* End of Switch: '<S2>/Switch3' */
1589
1590 /* MATLABSystem: '<S2>/Digital Input4' */
1591 if (Torque_Control_ESP32_V6_DW.obj_p.SampleTime !=
1592 Torque_Control_ESP32_V6_P.DigitalInput4_SampleTime) {
1593 Torque_Control_ESP32_V6_DW.obj_p.SampleTime =
1594 Torque_Control_ESP32_V6_P.DigitalInput4_SampleTime;
1595 }
1596
1597 c_value = readDigitalPin(5);
1598
1599 /* MATLABSystem: '<S2>/Digital Input4' */
1600 Torque_Control_ESP32_V6_B.DigitalInput4 = c_value;
1601
1602 /* Update for Delay: '<S2>/Delay1' */
1603 for (idxDelay = 0; idxDelay < 14; idxDelay++) {
1604 Torque_Control_ESP32_V6_DW.Delay1_DSTATE[idxDelay] =
1605 Torque_Control_ESP32_V6_DW.Delay1_DSTATE[idxDelay + 1];
1606 }
1607
1608 Torque_Control_ESP32_V6_DW.Delay1_DSTATE[14] =
1609 Torque_Control_ESP32_V6_B.DigitalInput2;
1610
1611 /* End of Update for Delay: '<S2>/Delay1' */
1612
1613 /* Update for Delay: '<S2>/Delay' */
1614 Torque_Control_ESP32_V6_DW.Delay_DSTATE[0] =
1615 Torque_Control_ESP32_V6_DW.Delay_DSTATE[1];
1616 Torque_Control_ESP32_V6_DW.Delay_DSTATE[1] =
1617 Torque_Control_ESP32_V6_B.Switch_h;
1618
1619 /* End of Outputs for S-Function (fcgen): '<Root>/Function-Call Generator1' */
1620
1621 /* S-Function (fcgen): '<Root>/Function-Call Generator3' incorporates:
1622 * SubSystem: '<Root>/SafetyCheck'
1623 */
1624 /* UnitDelay: '<S29>/Output' */
1625 Torque_Control_ESP32_V6_B.Output = Torque_Control_ESP32_V6_DW.Output_DSTATE;

```

```

1626
1627 /* RelationalOperator: '<S27>/Compare' incorporates:
1628 * Constant: '<S27>/Constant'
1629 */
1630 Torque_Control_ESP32_V6_B.Compare_a = Torque_Control_ESP32_V6_B.Output == 
1631 Torque_Control_ESP32_V6_P.CompareToConstant7_const;
1632
1633 /* S-Function (sdspstatfcns): '<S18>/Mean' */
1634 if (Torque_Control_ESP32_V6_B.Compare_a) {
1635 Torque_Control_ESP32_V6_DW.Mean_Iteration = 0U;
1636 }
1637
1638 iterStart = Torque_Control_ESP32_V6_DW.Mean_Iteration;
1639 Torque_Control_ESP32_V6_DW.Mean_Iteration = iterStart;
1640 Torque_Control_ESP32_V6_DW.Mean_Iteration++;
1641 if (Torque_Control_ESP32_V6_DW.Mean_Iteration > 1U) {
1642 Torque_Control_ESP32_V6_DW.Mean_AccVal += Torque_Control_ESP32_V6_B.AccPed01;
1643 den = (real32_T)Torque_Control_ESP32_V6_DW.Mean_Iteration;
1644
1645 /* S-Function (sdspstatfcns): '<S18>/Mean' */
1646 Torque_Control_ESP32_V6_B.Mean = Torque_Control_ESP32_V6_DW.Mean_AccVal / 
1647 den;
1648 } else {
1649 if (Torque_Control_ESP32_V6_DW.Mean_Iteration == 0U) {
1650 Torque_Control_ESP32_V6_DW.Mean_Iteration = 1U;
1651 }
1652
1653 Torque_Control_ESP32_V6_DW.Mean_AccVal = Torque_Control_ESP32_V6_B.AccPed01;
1654
1655 /* S-Function (sdspstatfcns): '<S18>/Mean' */
1656 Torque_Control_ESP32_V6_B.Mean = Torque_Control_ESP32_V6_B.AccPed01;
1657 }
1658
1659 /* End of S-Function (sdspstatfcns): '<S18>/Mean' */
1660
1661 /* RelationalOperator: '<S20>/Compare' incorporates:
1662 * Constant: '<S20>/Constant'
1663 */
1664 Torque_Control_ESP32_V6_B.Compare_h = Torque_Control_ESP32_V6_B.Mean <=
1665 Torque_Control_ESP32_V6_P.CompareToConstant_const_b;
1666
1667 /* RelationalOperator: '<S21>/Compare' incorporates:
1668 * Constant: '<S21>/Constant'
1669 */
1670 Torque_Control_ESP32_V6_B.Compare_g = Torque_Control_ESP32_V6_B.Mean >=
1671 Torque_Control_ESP32_V6_P.CompareToConstant1_const;
1672
1673 /* Logic: '<S18>/AND' */
1674 Torque_Control_ESP32_V6_B.AND = Torque_Control_ESP32_V6_B.Compare_h &&
1675 Torque_Control_ESP32_V6_B.Compare_g;
1676
1677 /* UnitDelay: '<S30>/Output' */
1678 Torque_Control_ESP32_V6_B.Output_p =
1679 Torque_Control_ESP32_V6_DW.Output_DSTATE_n;
1680
1681 /* RelationalOperator: '<S28>/Compare' incorporates:
1682 * Constant: '<S28>/Constant'
1683 */
1684 Torque_Control_ESP32_V6_B.Compare_ge = Torque_Control_ESP32_V6_B.Output_p ==
1685 Torque_Control_ESP32_V6_P.CompareToConstant8_const;
1686
1687 /* S-Function (sdspstatfcns): '<S18>/Mean1' */
1688 if (Torque_Control_ESP32_V6_B.Compare_ge) {
1689 Torque_Control_ESP32_V6_DW.Mean1_Iteration = 0U;
1690 }
1691
1692 iterStart = Torque_Control_ESP32_V6_DW.Mean1_Iteration;

```

```

1693 Torque_Control_ESP32_V6_DW.Mean1_Iteration = iterStart;
1694 Torque_Control_ESP32_V6_DW.Mean1_Iteration++;
1695 if (Torque_Control_ESP32_V6_DW.Mean1_Iteration > 1U) {
1696 Torque_Control_ESP32_V6_DW.Mean1_AccVal += 
1697 Torque_Control_ESP32_V6_B.AccPed02;
1698 den = (real32_T)Torque_Control_ESP32_V6_DW.Mean1_Iteration;
1699
1700 /* S-Function (sdspstatfcns): '<S18>/Mean1' */
1701 Torque_Control_ESP32_V6_B.Mean1 = Torque_Control_ESP32_V6_DW.Mean1_AccVal /
1702 den;
1703 } else {
1704 if (Torque_Control_ESP32_V6_DW.Mean1_Iteration == 0U) {
1705 Torque_Control_ESP32_V6_DW.Mean1_Iteration = 1U;
1706 }
1707
1708 Torque_Control_ESP32_V6_DW.Mean1_AccVal = Torque_Control_ESP32_V6_B.AccPed02;
1709
1710 /* S-Function (sdspstatfcns): '<S18>/Mean1' */
1711 Torque_Control_ESP32_V6_B.Mean1 = Torque_Control_ESP32_V6_B.AccPed02;
1712 }
1713
1714 /* End of S-Function (sdspstatfcns): '<S18>/Mean1' */
1715
1716 /* RelationalOperator: '<S22>/Compare' incorporates:
1717 * Constant: '<S22>/Constant'
1718 */
1719 Torque_Control_ESP32_V6_B.Compare_c = Torque_Control_ESP32_V6_B.Mean1 <=
1720 Torque_Control_ESP32_V6_P.CompareToConstant2_const;
1721
1722 /* RelationalOperator: '<S23>/Compare' incorporates:
1723 * Constant: '<S23>/Constant'
1724 */
1725 Torque_Control_ESP32_V6_B.Compare_h2 = Torque_Control_ESP32_V6_B.Mean1 >=
1726 Torque_Control_ESP32_V6_P.CompareToConstant3_const;
1727
1728 /* Logic: '<S18>/AND1' */
1729 Torque_Control_ESP32_V6_B.AND1 = Torque_Control_ESP32_V6_B.Compare_c &&
1730 Torque_Control_ESP32_V6_B.Compare_h2;
1731
1732 /* Switch: '<S18>/Switch' */
1733 if (Torque_Control_ESP32_V6_B.AND) {
1734 /* Switch: '<S18>/Switch' */
1735 Torque_Control_ESP32_V6_B.Switch = Torque_Control_ESP32_V6_B.Mean;
1736 } else {
1737 /* Switch: '<S18>/Switch' incorporates:
1738 * Constant: '<S18>/Constant'
1739 */
1740 Torque_Control_ESP32_V6_B.Switch = (real32_T)
1741 Torque_Control_ESP32_V6_P.Constant_Value_f;
1742 }
1743
1744 /* End of Switch: '<S18>/Switch' */
1745
1746 /* RelationalOperator: '<S25>/Compare' incorporates:
1747 * Constant: '<S25>/Constant'
1748 */
1749 Torque_Control_ESP32_V6_B.Compare_d = Torque_Control_ESP32_V6_B.Switch >
1750 Torque_Control_ESP32_V6_P.CompareToConstant5_const;
1751
1752 /* Logic: '<S19>/NOT1' */
1753 Torque_Control_ESP32_V6_B.NOT1 = !(Torque_Control_ESP32_V6_B.Switch4 != 0.0F);
1754
1755 /* Switch: '<S19>/Switch1' incorporates:
1756 * Constant: '<S6>/AnlgBrkPrsnt'
1757 */
1758 if (Torque_Control_ESP32_V6_P.AnlgBrkPrsnt_Value >
1759 Torque_Control_ESP32_V6_P.Switch1_Threshold_f) {

```

```

1760 /* Switch: '<S19>/Switch1' incorporates:
1761 * Constant: '<S6>/brkdefault'
1762 */
1763 Torque_Control_ESP32_V6_B.Switch1_p =
1764 Torque_Control_ESP32_V6_P.brkdefault_Value;
1765 } else {
1766 /* Switch: '<S19>/Switch1' */
1767 Torque_Control_ESP32_V6_B.Switch1_p = Torque_Control_ESP32_V6_B.Switch1_b;
1768 }
1769
1770 /* End of Switch: '<S19>/Switch1' */
1771
1772 /* Logic: '<S19>/NOT2' */
1773 Torque_Control_ESP32_V6_B.NOT2 = !(Torque_Control_ESP32_V6_B.Switch1_p != 0.0F);
1774
1775 /* Logic: '<S19>/AND1' */
1776 Torque_Control_ESP32_V6_B.AND1_g = Torque_Control_ESP32_V6_B.NOT1 &&
1777 Torque_Control_ESP32_V6_B.NOT2;
1778
1779 /* RelationalOperator: '<S26>/Compare' incorporates:
1780 * Constant: '<S26>/Constant'
1781 */
1782 Torque_Control_ESP32_V6_B.Compare_o = Torque_Control_ESP32_V6_B.AND1_g ==
1783 Torque_Control_ESP32_V6_P.CompareToConstant6_const;
1784
1785 /* Logic: '<S18>/AND2' */
1786 Torque_Control_ESP32_V6_B.AND2 = Torque_Control_ESP32_V6_B.Compare_d &&
1787 Torque_Control_ESP32_V6_B.Compare_o;
1788
1789 /* Switch: '<S18>/Switch1' */
1790 if (Torque_Control_ESP32_V6_B.AND1) {
1791 /* Switch: '<S18>/Switch1' */
1792 Torque_Control_ESP32_V6_B.Switch1_a = Torque_Control_ESP32_V6_B.Mean1;
1793 } else {
1794 /* Switch: '<S18>/Switch1' incorporates:
1795 * Constant: '<S18>/Constant2'
1796 */
1797 Torque_Control_ESP32_V6_B.Switch1_a = (real32_T)
1798 Torque_Control_ESP32_V6_P.Constant2_Value_g;
1799 }
1800
1801 /* End of Switch: '<S18>/Switch1' */
1802
1803 /* Sum: '<S18>/Add2' */
1804 Torque_Control_ESP32_V6_B.Add2 = Torque_Control_ESP32_V6_B.Switch -
1805 Torque_Control_ESP32_V6_B.Switch1_a;
1806
1807 /* Abs: '<S18>/Abs' */
1808 Torque_Control_ESP32_V6_B.Abs = (real32_T)fabs(Torque_Control_ESP32_V6_B.Add2);
1809
1810 /* RelationalOperator: '<S24>/Compare' incorporates:
1811 * Constant: '<S24>/Constant'
1812 */
1813 Torque_Control_ESP32_V6_B.Compare_i = Torque_Control_ESP32_V6_B.Abs >
1814 Torque_Control_ESP32_V6_P.CompareToConstant4_const;
1815
1816 /* Sum: '<S31>/FixPt Sum1' incorporates:
1817 * Constant: '<S31>/FixPt Constant'
1818 */
1819 Torque_Control_ESP32_V6_B.FixPtSum1 = (uint8_T)((uint32_T)
1820 Torque_Control_ESP32_V6_B.Output +
1821 Torque_Control_ESP32_V6_P.FixPtConstant_Value);
1822
1823 /* Switch: '<S32>/FixPt Switch' */
1824 if (Torque_Control_ESP32_V6_B.FixPtSum1 >
1825 Torque_Control_ESP32_V6_P.WrapToZero_Threshold) {
1826 /* Switch: '<S32>/FixPt Switch' incorporates:

```

```

1827 /* Constant: '<S32>/Constant'
1828 */
1829 Torque_Control_ESP32_V6_B.FixPtSwitch =
1830 Torque_Control_ESP32_V6_P.Constant_Value_c;
1831 } else {
1832 /* Switch: '<S32>/FixPt Switch' */
1833 Torque_Control_ESP32_V6_B.FixPtSwitch = Torque_Control_ESP32_V6_B.FixPtSum1;
1834 }
1835
1836 /* End of Switch: '<S32>/FixPt Switch' */
1837
1838 /* Sum: '<S33>/FixPt Sum1' incorporates:
1839 * Constant: '<S33>/FixPt Constant'
1840 */
1841 Torque_Control_ESP32_V6_B.FixPtSum1_b = (uint8_T)((uint32_T)
1842 Torque_Control_ESP32_V6_B.Output_p +
1843 Torque_Control_ESP32_V6_P.FixPtConstant_Value_g);
1844
1845 /* Switch: '<S34>/FixPt Switch' */
1846 if (Torque_Control_ESP32_V6_B.FixPtSum1_b >
1847 Torque_Control_ESP32_V6_P.WrapToZero_Threshold_b) {
1848 /* Switch: '<S34>/FixPt Switch' incorporates:
1849 * Constant: '<S34>/Constant'
1850 */
1851 Torque_Control_ESP32_V6_B.FixPtSwitch_o =
1852 Torque_Control_ESP32_V6_P.Constant_Value_o;
1853 } else {
1854 /* Switch: '<S34>/FixPt Switch' */
1855 Torque_Control_ESP32_V6_B.FixPtSwitch_o =
1856 Torque_Control_ESP32_V6_B.FixPtSum1_b;
1857 }
1858
1859 /* End of Switch: '<S34>/FixPt Switch' */
1860
1861 /* Logic: '<S18>/NOT' */
1862 Torque_Control_ESP32_V6_B.NOT = !Torque_Control_ESP32_V6_B.AND2;
1863
1864 /* Switch: '<S18>/Switch2' */
1865 if (Torque_Control_ESP32_V6_B.NOT) {
1866 /* Switch: '<S18>/Switch2' */
1867 Torque_Control_ESP32_V6_B.Switch2 = Torque_Control_ESP32_V6_B.Switch;
1868 } else {
1869 /* Switch: '<S18>/Switch2' incorporates:
1870 * Constant: '<S18>/Constant4'
1871 */
1872 Torque_Control_ESP32_V6_B.Switch2 = (real32_T)
1873 Torque_Control_ESP32_V6_P.Constant4_Value_d;
1874 }
1875
1876 /* End of Switch: '<S18>/Switch2' */
1877
1878 /* Logic: '<S6>/NOT' */
1879 Torque_Control_ESP32_V6_B.NOT_f = !(Torque_Control_ESP32_V6_B.RTD_st != 0.0F);
1880
1881 /* Logic: '<S6>/Logical Operator' */
1882 Torque_Control_ESP32_V6_B.LogicalOperator =
1883 Torque_Control_ESP32_V6_B.Compare_i || Torque_Control_ESP32_V6_B.NOT_f;
1884
1885 /* Update for UnitDelay: '<S29>/Output' */
1886 Torque_Control_ESP32_V6_DW.Output_DSTATE =
1887 Torque_Control_ESP32_V6_B.FixPtSwitch;
1888
1889 /* Update for UnitDelay: '<S30>/Output' */
1890 Torque_Control_ESP32_V6_DW.Output_DSTATE_n =
1891 Torque_Control_ESP32_V6_B.FixPtSwitch_o;
1892
1893 /* End of Outputs for S-Function (fcgen): '<Root>/Function-Call Generator3' */

```

```

1894
1895 /* S-Function (fcgen): '<Root>/Function-Call Generator2' incorporates:
1896 * SubSystem: '<Root>/Cooling'
1897 */
1898 /* Switch: '<S1>/Switch' */
1899 if (Torque_Control_ESP32_V6_B.Switch3 >
1900 Torque_Control_ESP32_V6_P.Switch_Threshold_m) {
1901 /* Switch: '<S1>/Switch' incorporates:
1902 * Constant: '<S1>/Constant'
1903 */
1904 Torque_Control_ESP32_V6_B.Switch_a =
1905 Torque_Control_ESP32_V6_P.Constant_Value_i != 0.0F;
1906 } else {
1907 /* Switch: '<S1>/Switch' incorporates:
1908 * Constant: '<S1>/Constant1'
1909 */
1910 Torque_Control_ESP32_V6_B.Switch_a =
1911 Torque_Control_ESP32_V6_P.Constant1_Value_i != 0.0F;
1912 }
1913
1914 /* End of Switch: '<S1>/Switch' */
1915 /* End of Outputs for S-Function (fcgen): '<Root>/Function-Call Generator2' */
1916
1917 /* S-Function (fcgen): '<Root>/Function-Call Generator6' incorporates:
1918 * SubSystem: '<Root>/GearShift'
1919 */
1920 /* RelationalOperator: '<S9>/Compare' incorporates:
1921 * Constant: '<S9>/Constant'
1922 */
1923 Torque_Control_ESP32_V6_B.Compare = Torque_Control_ESP32_V6_B.Switch7 >
1924 Torque_Control_ESP32_V6_P.CompareToConstant_const_a;
1925
1926 /* DiscretePulseGenerator: '<S4>/Pulse Generator1' */
1927 Torque_Control_ESP32_V6_B.PulseGenerator1 =
1928 Torque_Control_ESP32_V6_DW.clockTickCounter <
1929 Torque_Control_ESP32_V6_P.PulseGenerator1_Duty &&
1930 Torque_Control_ESP32_V6_DW.clockTickCounter >= 0 ?
1931 Torque_Control_ESP32_V6_P.PulseGenerator1_Amp : 0.0;
1932
1933 /* DiscretePulseGenerator: '<S4>/Pulse Generator1' */
1934 if (Torque_Control_ESP32_V6_DW.clockTickCounter >=
1935 Torque_Control_ESP32_V6_P.PulseGenerator1_Period - 1.0) {
1936 Torque_Control_ESP32_V6_DW.clockTickCounter = 0;
1937 } else {
1938 Torque_Control_ESP32_V6_DW.clockTickCounter++;
1939 }
1940
1941 /* Switch: '<S4>/Switch' */
1942 if (Torque_Control_ESP32_V6_B.Compare) {
1943 /* Switch: '<S4>/Switch' */
1944 Torque_Control_ESP32_V6_B.Switch_i =
1945 Torque_Control_ESP32_V6_B.PulseGenerator1 != 0.0;
1946 } else {
1947 /* Switch: '<S4>/Switch' incorporates:
1948 * Constant: '<S4>/Constant'
1949 */
1950 Torque_Control_ESP32_V6_B.Switch_i =
1951 Torque_Control_ESP32_V6_P.Constant_Value_ij != 0.0F;
1952 }
1953
1954 /* End of Switch: '<S4>/Switch' */
1955 /* End of Outputs for S-Function (fcgen): '<Root>/Function-Call Generator6' */
1956
1957 /* S-Function (fcgen): '<Root>/Function-Call Generator4' incorporates:
1958 * SubSystem: '<Root>/PowerTrain'
1959 */
1960 /* Switch: '<S5>/Switch1' incorporates:

```

```

1961 /* Constant: '<S5>/VehSpeedPrsnt'
1962 */
1963 if (Torque_Control_ESP32_V6_P.VehSpeedPrsnt_Value >
1964 Torque_Control_ESP32_V6_P.Switch1_Threshold_e) {
1965 /* Product: '<S5>/Divide3' incorporates:
1966 * Constant: '<S5>/TransmRat'
1967 */
1968 Torque_Control_ESP32_V6_B.Divide3 = Torque_Control_ESP32_V6_B.Switch7 /
1969 Torque_Control_ESP32_V6_P.TransmRat_Value;
1970
1971 /* Product: '<S5>/Divide' incorporates:
1972 * Constant: '<S5>/sec'
1973 */
1974 Torque_Control_ESP32_V6_B.Divide_m = Torque_Control_ESP32_V6_B.Divide3 /
1975 Torque_Control_ESP32_V6_P.sec_Value_e;
1976
1977 /* Product: '<S5>/Divide1' incorporates:
1978 * Constant: '<S5>/sec1'
1979 */
1980 Torque_Control_ESP32_V6_B.Divide1_j = Torque_Control_ESP32_V6_B.Divide_m *
1981 Torque_Control_ESP32_V6_P.sec1_Value_o;
1982
1983 /* Switch: '<S5>/Switch1' */
1984 Torque_Control_ESP32_V6_B.Switch1 = Torque_Control_ESP32_V6_B.Divide1_j;
1985 } else {
1986 /* Switch: '<S5>/Switch1' */
1987 Torque_Control_ESP32_V6_B.Switch1 = Torque_Control_ESP32_V6_B.Switch8;
1988 }
1989
1990 /* End of Switch: '<S5>/Switch1' */
1991
1992 /* SwitchCase: '<S5>/Switch Case1' */
1993 den = Torque_Control_ESP32_V6_B.Switch_h;
1994 if (den < 0.0F) {
1995 den = (real32_T)ceil(den);
1996 } else {
1997 den = (real32_T)floor(den);
1998 }
1999
2000 if (rtlsNaNF(den) || rtlsInff(den)) {
2001 den = 0.0F;
2002 } else {
2003 den = (real32_T)fmod(den, 4.294967296E+9);
2004 }
2005
2006 switch (den < 0.0F ? -(int32_T)(uint32_T)-den : (int32_T)(uint32_T)den) {
2007 case 0:
2008 /* Outputs for IfAction SubSystem: '<S5>/NormalMode' incorporates:
2009 * ActionPort: '<S12>/Action Port'
2010 */
2011 /* Merge: '<S5>/Merge' incorporates:
2012 * Lookup_n-D: '<S12>/NormalMode'
2013 * Switch: '<S18>/Switch2'
2014 * Switch: '<S5>/Switch1'
2015 */
2016 Torque_Control_ESP32_V6_B.Merge = look2_iflf_linlcawp
2017 (Torque_Control_ESP32_V6_B.Switch2, Torque_Control_ESP32_V6_B.Switch1,
2018 Torque_Control_ESP32_V6_P.NormalMode_bp01Data,
2019 Torque_Control_ESP32_V6_P.NormalMode_bp02Data,
2020 Torque_Control_ESP32_V6_P.NormalMode_tableData,
2021 Torque_Control_ESP32_V6_P.NormalMode_maxIndex, 11U);
2022
2023 /* End of Outputs for SubSystem: '<S5>/NormalMode' */
2024 break;
2025
2026 case 1:
2027 /* Outputs for IfAction SubSystem: '<S5>/SportMode' incorporates:

```

```

2028 /* ActionPort: '<S14>/Action Port'
2029 */
2030 /* Merge: '<S5>/Merge' incorporates:
2031 * Lookup_n-D: '<S14>/SportMode'
2032 * Switch: '<S18>/Switch2'
2033 * Switch: '<S5>/Switch1'
2034 */
2035 Torque_Control_ESP32_V6_B.Merge = look2_ifff_linlcawp
2036 (Torque_Control_ESP32_V6_B.Switch2, Torque_Control_ESP32_V6_B.Switch1,
2037 Torque_Control_ESP32_V6_P.SportMode_bp01Data,
2038 Torque_Control_ESP32_V6_P.SportMode_bp02Data,
2039 Torque_Control_ESP32_V6_P.SportMode_tableData,
2040 Torque_Control_ESP32_V6_P.SportMode_maxIndex, 11U);
2041
2042 /* End of Outputs for SubSystem: '<S5>/SportMode' */
2043 break;
2044
2045 case 2:
2046 /* Outputs for IfAction SubSystem: '<S5>/EcoMode' incorporates:
2047 * ActionPort: '<S11>/Action Port'
2048 */
2049 /* Merge: '<S5>/Merge' incorporates:
2050 * Lookup_n-D: '<S11>/EcoMode'
2051 * Switch: '<S18>/Switch2'
2052 * Switch: '<S5>/Switch1'
2053 */
2054 Torque_Control_ESP32_V6_B.Merge = look2_ifff_linlcawp
2055 (Torque_Control_ESP32_V6_B.Switch2, Torque_Control_ESP32_V6_B.Switch1,
2056 Torque_Control_ESP32_V6_P.EcoMode_bp01Data,
2057 Torque_Control_ESP32_V6_P.EcoMode_bp02Data,
2058 Torque_Control_ESP32_V6_P.EcoMode_tableData,
2059 Torque_Control_ESP32_V6_P.EcoMode_maxIndex, 11U);
2060
2061 /* End of Outputs for SubSystem: '<S5>/EcoMode' */
2062 break;
2063
2064 default:
2065 /* Outputs for IfAction SubSystem: '<S5>/Defalut' incorporates:
2066 * ActionPort: '<S10>/Action Port'
2067 */
2068 /* Merge: '<S5>/Merge' incorporates:
2069 * Constant: '<S10>/Constant'
2070 * SignalConversion generated from: '<S10>/Out1'
2071 */
2072 Torque_Control_ESP32_V6_B.Merge = Torque_Control_ESP32_V6_P.Constant_Value_d;
2073
2074 /* End of Outputs for SubSystem: '<S5>/Defalut' */
2075 break;
2076 }
2077
2078 /* End of SwitchCase: '<S5>/Switch Case1' */
2079 /* End of Outputs for S-Function (fcgen): '<Root>/Function-Call Generator4' */
2080
2081 /* S-Function (fcgen): '<Root>/Function-Call Generator' incorporates:
2082 * SubSystem: '<Root>/DataTransmission'
2083 */
2084 /* MATLABSystem: '<S3>/Digital Output2' */
2085 writeDigitalPin(32, (uint8_T)Torque_Control_ESP32_V6_B.LogicalOperator);
2086
2087 /* MATLABSystem: '<S3>/Digital Output1' */
2088 writeDigitalPin(33, (uint8_T)Torque_Control_ESP32_V6_B.Switch_a);
2089
2090 /* MATLABSystem: '<S3>/Digital Output' */
2091 writeDigitalPin(2, (uint8_T)Torque_Control_ESP32_V6_B.Switch_i);
2092
2093 /* End of Outputs for S-Function (fcgen): '<Root>/Function-Call Generator' */
2094 }

```

```

2095 /* Model initialize function */
2096 void Torque_Control_ESP32_V6_initialize(void)
2097 {
2098 /* Registration code */
2099
2100 /* initialize non-finites */
2101 rt_InitInfAndNaN(sizeof(real_T));
2102
2103 /* initialize error status */
2104 rtmSetErrorStatus(Torque_Control_ESP32_V6_M, (NULL));
2105
2106 /* block I/O */
2107 (void) memset(((void *) &Torque_Control_ESP32_V6_B), 0,
2108 sizeof(B_Torque_Control_ESP32_V6_T));
2109
2110 /* states (dwork) */
2111 (void) memset((void *)&Torque_Control_ESP32_V6_DW, 0,
2112 sizeof(DW_Torque_Control_ESP32_V6_T));
2113
2114
2115 {
2116 int32_T i;
2117 Torque_Control_ESP32_V6_PrevZCX.TriggeredSubsystem_Trig_ZCE =
2118 UNINITIALIZED_ZCSIG;
2119
2120 /* SystemInitialize for S-Function (fcgen): '<Root>/Function-Call Generator1'
incorporates:
2121 * SubSystem: '<Root>/DataAcquisition'
2122 */
2123 /* InitializeConditions for Delay: '<S2>/Delay1' */
2124 for (i = 0; i < 15; i++) {
2125 Torque_Control_ESP32_V6_DW.Delay1_DSTATE[i] =
2126 Torque_Control_ESP32_V6_P.Delay1_InitialCondition;
2127 }
2128
2129 /* End of InitializeConditions for Delay: '<S2>/Delay1' */
2130
2131 /* InitializeConditions for Delay: '<S2>/Delay' */
2132 Torque_Control_ESP32_V6_DW.Delay_DSTATE[0] =
2133 Torque_Control_ESP32_V6_P.Delay_InitialCondition;
2134 Torque_Control_ESP32_V6_DW.Delay_DSTATE[1] =
2135 Torque_Control_ESP32_V6_P.Delay_InitialCondition;
2136
2137 /* SystemInitialize for Triggered SubSystem: '<S2>/Triggered Subsystem' */
2138 /* SystemInitialize for Switch: '<S7>/Switch' incorporates:
2139 * Outport: '<S7>/DrvMode_st'
2140 */
2141 Torque_Control_ESP32_V6_B.Switch_h = Torque_Control_ESP32_V6_P.DrvMode_st_Y0;
2142
2143 /* End of SystemInitialize for SubSystem: '<S2>/Triggered Subsystem' */
2144
2145 /* Start for MATLABSystem: '<S2>/Analog Input4' */
2146 Torque_Control_ESP32_V6_DW.obj_ac.matlabCodegenIsDeleted = false;
2147 Torque_Control_ESP32_V6_DW.obj_isempty_n = true;
2148 Torque_Control_ESP32_V6_DW.obj_ac.SampleTime =
2149 Torque_Control_ESP32_V6_P.AnalogInput4_SampleTime;
2150 Torque_Control_ESP32_V6_DW.obj_ac.isInitialized = 1;
2151 Torque_Control_ESP32_V6_DW.obj_ac.AnalogInDriverObj.MW_ANALOGIN_HANDLE =
2152 MW_AnalogInSingle_Open(36U);
2153 Torque_Control_ESP32_V6_DW.obj_ac.isSetupComplete = true;
2154
2155 /* Start for MATLABSystem: '<S2>/Analog Input5' */
2156 Torque_Control_ESP32_V6_DW.obj_a.matlabCodegenIsDeleted = false;
2157 Torque_Control_ESP32_V6_DW.obj_isempty_d = true;
2158 Torque_Control_ESP32_V6_DW.obj_a.SampleTime =
2159 Torque_Control_ESP32_V6_P.AnalogInput5_SampleTime;
2160 Torque_Control_ESP32_V6_DW.obj_a.isInitialized = 1;

```

```

2161 Torque_Control_ESP32_V6_DW.obj_a.AnalogInDriverObj.MW_ANALOGIN_HANDLE =
2162 MW_AnalogInSingle_Open(39U);
2163 Torque_Control_ESP32_V6_DW.obj_a.isSetupComplete = true;
2164
2165 /* Start for MATLABSystem: '<S2>/Digital Input' */
2166 Torque_Control_ESP32_V6_DW.obj_ny.matlabCodegenIsDeleted = false;
2167 Torque_Control_ESP32_V6_DW.objisempty_a = true;
2168 Torque_Control_ESP32_V6_DW.obj_ny.SampleTime =
2169 Torque_Control_ESP32_V6_P.DigitalInput_SampleTime;
2170 Torque_Control_ESP32_V6_DW.obj_ny.isInitialized = 1;
2171 digitalIOSetup(25, 0);
2172 Torque_Control_ESP32_V6_DW.obj_ny.isSetupComplete = true;
2173
2174 /* Start for MATLABSystem: '<S2>/Analog Input7' */
2175 Torque_Control_ESP32_V6_DW.obj.matlabCodegenIsDeleted = false;
2176 Torque_Control_ESP32_V6_DW.objisempty_e = true;
2177 Torque_Control_ESP32_V6_DW.obj.SampleTime =
2178 Torque_Control_ESP32_V6_P.AnalogInput7_SampleTime;
2179 Torque_Control_ESP32_V6_DW.obj.isInitialized = 1;
2180 Torque_Control_ESP32_V6_DW.obj.AnalogInDriverObj.MW_ANALOGIN_HANDLE =
2181 MW_AnalogInSingle_Open(35U);
2182 Torque_Control_ESP32_V6_DW.obj.isSetupComplete = true;
2183
2184 /* Start for MATLABSystem: '<S2>/Digital Input1' */
2185 Torque_Control_ESP32_V6_DW.obj_g.matlabCodegenIsDeleted = false;
2186 Torque_Control_ESP32_V6_DW.objisempty_g = true;
2187 Torque_Control_ESP32_V6_DW.obj_g.SampleTime =
2188 Torque_Control_ESP32_V6_P.DigitalInput1_SampleTime;
2189 Torque_Control_ESP32_V6_DW.obj_g.isInitialized = 1;
2190 digitalIOSetup(26, 0);
2191 Torque_Control_ESP32_V6_DW.obj_g.isSetupComplete = true;
2192
2193 /* Start for MATLABSystem: '<S2>/Digital Input2' */
2194 Torque_Control_ESP32_V6_DW.obj_n.matlabCodegenIsDeleted = false;
2195 Torque_Control_ESP32_V6_DW.objisempty_l = true;
2196 Torque_Control_ESP32_V6_DW.obj_n.SampleTime =
2197 Torque_Control_ESP32_V6_P.DigitalInput2_SampleTime;
2198 Torque_Control_ESP32_V6_DW.obj_n.isInitialized = 1;
2199 digitalIOSetup(27, 0);
2200 Torque_Control_ESP32_V6_DW.obj_n.isSetupComplete = true;
2201
2202 /* Start for MATLABSystem: '<S2>/Analog Input1' */
2203 Torque_Control_ESP32_V6_DW.obj_h.matlabCodegenIsDeleted = false;
2204 Torque_Control_ESP32_V6_DW.objisempty_nl = true;
2205 Torque_Control_ESP32_V6_DW.obj_h.SampleTime =
2206 Torque_Control_ESP32_V6_P.AnalogInput1_SampleTime;
2207 Torque_Control_ESP32_V6_DW.obj_h.isInitialized = 1;
2208 Torque_Control_ESP32_V6_DW.obj_h.AnalogInDriverObj.MW_ANALOGIN_HANDLE =
2209 MW_AnalogInSingle_Open(14U);
2210 Torque_Control_ESP32_V6_DW.obj_h.isSetupComplete = true;
2211
2212 /* Start for MATLABSystem: '<S2>/Digital Input3' */
2213 Torque_Control_ESP32_V6_DW.obj_c.matlabCodegenIsDeleted = false;
2214 Torque_Control_ESP32_V6_DW.objisempty_m = true;
2215 Torque_Control_ESP32_V6_DW.obj_c.SampleTime =
2216 Torque_Control_ESP32_V6_P.DigitalInput3_SampleTime;
2217 Torque_Control_ESP32_V6_DW.obj_c.isInitialized = 1;
2218 digitalIOSetup(4, 0);
2219 Torque_Control_ESP32_V6_DW.obj_c.isSetupComplete = true;
2220
2221 /* Start for MATLABSystem: '<S2>/Analog Input6' */
2222 Torque_Control_ESP32_V6_DW.obj_j.matlabCodegenIsDeleted = false;
2223 Torque_Control_ESP32_V6_DW.objisempty_j = true;
2224 Torque_Control_ESP32_V6_DW.obj_j.SampleTime =
2225 Torque_Control_ESP32_V6_P.AnalogInput6_SampleTime;
2226 Torque_Control_ESP32_V6_DW.obj_j.isInitialized = 1;
2227 Torque_Control_ESP32_V6_DW.obj_j.AnalogInDriverObj.MW_ANALOGIN_HANDLE =

```

```

2228 MW_AnalogInSingle_Open(34U);
2229 Torque_Control_ESP32_V6_DW.obj_j.isSetupComplete = true;
2230
2231 /* Start for MATLABSystem: '<S2>/Digital Input4' */
2232 Torque_Control_ESP32_V6_DW.obj_p.matlabCodegenIsDeleted = false;
2233 Torque_Control_ESP32_V6_DW.objjisempty = true;
2234 Torque_Control_ESP32_V6_DW.obj_p.SampleTime =
2235 Torque_Control_ESP32_V6_P.DigitalInput4_SampleTime;
2236 Torque_Control_ESP32_V6_DW.obj_p.isInitialized = 1;
2237 digitalIOSetup(5, 0);
2238 Torque_Control_ESP32_V6_DW.obj_p.isSetupComplete = true;
2239
2240 /* SystemInitialize for Switch: '<S2>/Switch' incorporates:
2241 * Outport: '<S2>/AccPed01'
2242 */
2243 Torque_Control_ESP32_V6_B.AccPed01 = Torque_Control_ESP32_V6_P.AccPed01_Y0;
2244
2245 /* SystemInitialize for Switch: '<S2>/Switch2' incorporates:
2246 * Outport: '<S2>/AccPed02'
2247 */
2248 Torque_Control_ESP32_V6_B.AccPed02 = Torque_Control_ESP32_V6_P.AccPed02_Y0;
2249
2250 /* SystemInitialize for Switch: '<S2>/Switch4' incorporates:
2251 * Outport: '<S2>/BrkPedDig_st'
2252 */
2253 Torque_Control_ESP32_V6_B.Switch4 =
2254 Torque_Control_ESP32_V6_P.BrkPedDig_st_Y0;
2255
2256 /* SystemInitialize for Switch: '<S2>/Switch1' incorporates:
2257 * Outport: '<S2>/BrkPedAnlg'
2258 */
2259 Torque_Control_ESP32_V6_B.Switch1_b =
2260 Torque_Control_ESP32_V6_P.BrkPedAnlg_Y0;
2261
2262 /* SystemInitialize for Switch: '<S2>/Switch5' incorporates:
2263 * Outport: '<S2>/RTD_st'
2264 */
2265 Torque_Control_ESP32_V6_B.RTD_st = Torque_Control_ESP32_V6_P.RTD_st_Y0;
2266
2267 /* SystemInitialize for Switch: '<S2>/Switch7' incorporates:
2268 * Outport: '<S2>/AxelRpm'
2269 */
2270 Torque_Control_ESP32_V6_B.Switch7 = Torque_Control_ESP32_V6_P.AxelRpm_Y0;
2271
2272 /* SystemInitialize for Switch: '<S2>/Switch8' incorporates:
2273 * Outport: '<S2>/VehSpeed'
2274 */
2275 Torque_Control_ESP32_V6_B.Switch8 = Torque_Control_ESP32_V6_P.VehSpeed_Y0;
2276
2277 /* SystemInitialize for Switch: '<S2>/Switch3' incorporates:
2278 * Outport: '<S2>/TempSensr'
2279 */
2280 Torque_Control_ESP32_V6_B.Switch3 = Torque_Control_ESP32_V6_P.TempSensr_Y0;
2281
2282 /* End of SystemInitialize for S-Function (fcgen): '<Root>/Function-Call
Generator1' */
2283
2284 /* SystemInitialize for S-Function (fcgen): '<Root>/Function-Call Generator3'
incorporates:
2285 * SubSystem: '<Root>/SafetyCheck'
2286 */
2287 /* InitializeConditions for UnitDelay: '<S29>/Output' */
2288 Torque_Control_ESP32_V6_DW.Output_DSTATE =
2289 Torque_Control_ESP32_V6_P.Output_InitialCondition;
2290
2291 /* InitializeConditions for S-Function (sdspstatfcns): '<S18>/Mean' */
2292 Torque_Control_ESP32_V6_DW.Mean_Iteration = 0U;

```

```

2293 Torque_Control_ESP32_V6_DW.Mean_AccVal = 0.0F;
2294
2295 /* InitializeConditions for UnitDelay: '<S30>/Output' */
2296 Torque_Control_ESP32_V6_DW.Output_DSTATE_n =
2297 Torque_Control_ESP32_V6_P.Output_InitialCondition_o;
2298
2299 /* InitializeConditions for S-Function (sdspstatfcns): '<S18>/Mean1' */
2300 Torque_Control_ESP32_V6_DW.Mean1_Iteration = 0U;
2301 Torque_Control_ESP32_V6_DW.Mean1_AccVal = 0.0F;
2302
2303 /* SystemInitialize for Logic: '<S6>/Logical Operator' incorporates:
2304 * Outport: '<S6>/Fault_st'
2305 */
2306 Torque_Control_ESP32_V6_B.LogicalOperator =
2307 Torque_Control_ESP32_V6_P.Fault_st_Y0;
2308
2309 /* SystemInitialize for Switch: '<S18>/Switch2' incorporates:
2310 * Outport: '<S6>/AccPedReq'
2311 */
2312 Torque_Control_ESP32_V6_B.Switch2 = Torque_Control_ESP32_V6_P.AccPedReq_Y0;
2313
2314 /* End of SystemInitialize for S-Function (fcgen): '<Root>/Function-Call
Generator3' */
2315
2316 /* SystemInitialize for S-Function (fcgen): '<Root>/Function-Call Generator2'
incorporates:
2317 * SubSystem: '<Root>/Cooling'
2318 */
2319 /* SystemInitialize for Switch: '<S1>/Switch' incorporates:
2320 * Outport: '<S1>/FanReq'
2321 */
2322 Torque_Control_ESP32_V6_B.Switch_a = Torque_Control_ESP32_V6_P.FanReq_Y0;
2323
2324 /* End of SystemInitialize for S-Function (fcgen): '<Root>/Function-Call
Generator2' */
2325
2326 /* SystemInitialize for S-Function (fcgen): '<Root>/Function-Call Generator6'
incorporates:
2327 * SubSystem: '<Root>/GearShift'
2328 */
2329 /* InitializeConditions for DiscretePulseGenerator: '<S4>/Pulse Generator1' */
2330 Torque_Control_ESP32_V6_DW.clockTickCounter = 0;
2331
2332 /* SystemInitialize for Switch: '<S4>/Switch' incorporates:
2333 * Outport: '<S4>/LED01'
2334 */
2335 Torque_Control_ESP32_V6_B.Switch_i = Torque_Control_ESP32_V6_P.LED01_Y0;
2336
2337 /* End of SystemInitialize for S-Function (fcgen): '<Root>/Function-Call
Generator6' */
2338
2339 /* SystemInitialize for S-Function (fcgen): '<Root>/Function-Call Generator4'
incorporates:
2340 * SubSystem: '<Root>/PowerTrain'
2341 */
2342 /* SystemInitialize for Merge: '<S5>/Merge' */
2343 Torque_Control_ESP32_V6_B.Merge =
2344 Torque_Control_ESP32_V6_P.Merge_InitialOutput;
2345
2346 /* End of SystemInitialize for S-Function (fcgen): '<Root>/Function-Call
Generator4' */
2347
2348 /* SystemInitialize for S-Function (fcgen): '<Root>/Function-Call Generator'
incorporates:
2349 * SubSystem: '<Root>/DataTransmission'
2350 */
2351 /* Start for MATLABSystem: '<S3>/Digital Output2' */

```

```

2352 Torque_Control_ESP32_V6_DW.obj_d.matlabCodegenIsDeleted = false;
2353 Torque_Control_ESP32_V6_DW.obj_isempty_c = true;
2354 Torque_Control_ESP32_V6_DW.obj_d.isInitialized = 1;
2355 digitalIOSetup(32, 1);
2356 Torque_Control_ESP32_V6_DW.obj_d.isSetupComplete = true;
2357
2358 /* Start for MATLABSystem: '<S3>/Digital Output1' */
2359 Torque_Control_ESP32_V6_DW.obj_o.matlabCodegenIsDeleted = false;
2360 Torque_Control_ESP32_V6_DW.obj_isempty_f = true;
2361 Torque_Control_ESP32_V6_DW.obj_o.isInitialized = 1;
2362 digitalIOSetup(33, 1);
2363 Torque_Control_ESP32_V6_DW.obj_o.isSetupComplete = true;
2364
2365 /* Start for MATLABSystem: '<S3>/Digital Output' */
2366 Torque_Control_ESP32_V6_DW.obj_ds.matlabCodegenIsDeleted = false;
2367 Torque_Control_ESP32_V6_DW.obj_isempty_fg = true;
2368 Torque_Control_ESP32_V6_DW.obj_ds.isInitialized = 1;
2369 digitalIOSetup(2, 1);
2370 Torque_Control_ESP32_V6_DW.obj_ds.isSetupComplete = true;
2371
2372 /* End of SystemInitialize for S-Function (fcgen): '<Root>/Function-Call
Generator' */
2373 }
2374 }
2375
2376 /* Model terminate function */
2377 void Torque_Control_ESP32_V6_terminate(void)
2378 {
2379 /* Terminate for S-Function (fcgen): '<Root>/Function-Call Generator1' incorporates:
2380 * SubSystem: '<Root>/DataAcquisition'
2381 */
2382 /* Terminate for MATLABSystem: '<S2>/Analog Input4' */
2383 if (!Torque_Control_ESP32_V6_DW.obj_ac.matlabCodegenIsDeleted) {
2384 Torque_Control_ESP32_V6_DW.obj_ac.matlabCodegenIsDeleted = true;
2385 if (Torque_Control_ESP32_V6_DW.obj_ac.isInitialized == 1 &&
2386 Torque_Control_ESP32_V6_DW.obj_ac.isSetupComplete) {
2387 Torque_Control_ESP32_V6_DW.obj_ac.AnalogInDriverObj.MW_ANALOGIN_HANDLE =
2388 MW_AnalogIn_GetHandle(36U);
2389 MW_AnalogIn_Close
2390 (Torque_Control_ESP32_V6_DW.obj_ac.AnalogInDriverObj.MW_ANALOGIN_HANDLE);
2391 }
2392 }
2393
2394 /* End of Terminate for MATLABSystem: '<S2>/Analog Input4' */
2395
2396 /* Terminate for MATLABSystem: '<S2>/Analog Input5' */
2397 if (!Torque_Control_ESP32_V6_DW.obj_a.matlabCodegenIsDeleted) {
2398 Torque_Control_ESP32_V6_DW.obj_a.matlabCodegenIsDeleted = true;
2399 if (Torque_Control_ESP32_V6_DW.obj_a.isInitialized == 1 &&
2400 Torque_Control_ESP32_V6_DW.obj_a.isSetupComplete) {
2401 Torque_Control_ESP32_V6_DW.obj_a.AnalogInDriverObj.MW_ANALOGIN_HANDLE =
2402 MW_AnalogIn_GetHandle(39U);
2403 MW_AnalogIn_Close
2404 (Torque_Control_ESP32_V6_DW.obj_a.AnalogInDriverObj.MW_ANALOGIN_HANDLE);
2405 }
2406 }
2407
2408 /* End of Terminate for MATLABSystem: '<S2>/Analog Input5' */
2409
2410 /* Terminate for MATLABSystem: '<S2>/Digital Input' */
2411 if (!Torque_Control_ESP32_V6_DW.obj_ny.matlabCodegenIsDeleted) {
2412 Torque_Control_ESP32_V6_DW.obj_ny.matlabCodegenIsDeleted = true;
2413 }
2414
2415 /* End of Terminate for MATLABSystem: '<S2>/Digital Input' */
2416
2417 /* Terminate for MATLABSystem: '<S2>/Analog Input7' */

```

```

2418 if (!Torque_Control_ESP32_V6_DW.obj.matlabCodegenIsDeleted) {
2419 Torque_Control_ESP32_V6_DW.obj.matlabCodegenIsDeleted = true;
2420 if (Torque_Control_ESP32_V6_DW.obj.isInitialized == 1 &&
2421 Torque_Control_ESP32_V6_DW.obj.isSetupComplete) {
2422 Torque_Control_ESP32_V6_DW.obj.AnalogInDriverObj.MW_ANALOGIN_HANDLE =
2423 MW_AnalogIn_GetHandle(35U);
2424 MW_AnalogIn_Close
2425 (Torque_Control_ESP32_V6_DW.obj.AnalogInDriverObj.MW_ANALOGIN_HANDLE);
2426 }
2427 }
2428
2429 /* End of Terminate for MATLABSystem: '<S2>/Analog Input7' */
2430
2431 /* Terminate for MATLABSystem: '<S2>/Digital Input1' */
2432 if (!Torque_Control_ESP32_V6_DW.obj_g.matlabCodegenIsDeleted) {
2433 Torque_Control_ESP32_V6_DW.obj_g.matlabCodegenIsDeleted = true;
2434 }
2435
2436 /* End of Terminate for MATLABSystem: '<S2>/Digital Input1' */
2437
2438 /* Terminate for MATLABSystem: '<S2>/Digital Input2' */
2439 if (!Torque_Control_ESP32_V6_DW.obj_n.matlabCodegenIsDeleted) {
2440 Torque_Control_ESP32_V6_DW.obj_n.matlabCodegenIsDeleted = true;
2441 }
2442
2443 /* End of Terminate for MATLABSystem: '<S2>/Digital Input2' */
2444
2445 /* Terminate for MATLABSystem: '<S2>/Analog Input1' */
2446 if (!Torque_Control_ESP32_V6_DW.obj_h.matlabCodegenIsDeleted) {
2447 Torque_Control_ESP32_V6_DW.obj_h.matlabCodegenIsDeleted = true;
2448 if (Torque_Control_ESP32_V6_DW.obj_h.isInitialized == 1 &&
2449 Torque_Control_ESP32_V6_DW.obj_h.isSetupComplete) {
2450 Torque_Control_ESP32_V6_DW.obj_h.AnalogInDriverObj.MW_ANALOGIN_HANDLE =
2451 MW_AnalogIn_GetHandle(14U);
2452 MW_AnalogIn_Close
2453 (Torque_Control_ESP32_V6_DW.obj_h.AnalogInDriverObj.MW_ANALOGIN_HANDLE);
2454 }
2455 }
2456
2457 /* End of Terminate for MATLABSystem: '<S2>/Analog Input1' */
2458
2459 /* Terminate for MATLABSystem: '<S2>/Digital Input3' */
2460 if (!Torque_Control_ESP32_V6_DW.obj_c.matlabCodegenIsDeleted) {
2461 Torque_Control_ESP32_V6_DW.obj_c.matlabCodegenIsDeleted = true;
2462 }
2463
2464 /* End of Terminate for MATLABSystem: '<S2>/Digital Input3' */
2465
2466 /* Terminate for MATLABSystem: '<S2>/Analog Input6' */
2467 if (!Torque_Control_ESP32_V6_DW.obj_j.matlabCodegenIsDeleted) {
2468 Torque_Control_ESP32_V6_DW.obj_j.matlabCodegenIsDeleted = true;
2469 if (Torque_Control_ESP32_V6_DW.obj_j.isInitialized == 1 &&
2470 Torque_Control_ESP32_V6_DW.obj_j.isSetupComplete) {
2471 Torque_Control_ESP32_V6_DW.obj_j.AnalogInDriverObj.MW_ANALOGIN_HANDLE =
2472 MW_AnalogIn_GetHandle(34U);
2473 MW_AnalogIn_Close
2474 (Torque_Control_ESP32_V6_DW.obj_j.AnalogInDriverObj.MW_ANALOGIN_HANDLE);
2475 }
2476 }
2477
2478 /* End of Terminate for MATLABSystem: '<S2>/Analog Input6' */
2479
2480 /* Terminate for MATLABSystem: '<S2>/Digital Input4' */
2481 if (!Torque_Control_ESP32_V6_DW.obj_p.matlabCodegenIsDeleted) {
2482 Torque_Control_ESP32_V6_DW.obj_p.matlabCodegenIsDeleted = true;
2483 }
2484

```

```
2485 /* End of Terminate for MATLABSystem: '<S2>/Digital Input4' */
2486 /* End of Terminate for S-Function (fcgen): '<Root>/Function-Call Generator1' */
2487
2488 /* Terminate for S-Function (fcgen): '<Root>/Function-Call Generator' incorporates:
2489 * SubSystem: '<Root>/DataTransmission'
2490 */
2491 /* Terminate for MATLABSystem: '<S3>/Digital Output2' */
2492 if (!Torque_Control_ESP32_V6_DW.obj_d.matlabCodegenIsDeleted) {
2493 Torque_Control_ESP32_V6_DW.obj_d.matlabCodegenIsDeleted = true;
2494 }
2495
2496 /* End of Terminate for MATLABSystem: '<S3>/Digital Output2' */
2497
2498 /* Terminate for MATLABSystem: '<S3>/Digital Output1' */
2499 if (!Torque_Control_ESP32_V6_DW.obj_o.matlabCodegenIsDeleted) {
2500 Torque_Control_ESP32_V6_DW.obj_o.matlabCodegenIsDeleted = true;
2501 }
2502
2503 /* End of Terminate for MATLABSystem: '<S3>/Digital Output1' */
2504
2505 /* Terminate for MATLABSystem: '<S3>/Digital Output' */
2506 if (!Torque_Control_ESP32_V6_DW.obj_ds.matlabCodegenIsDeleted) {
2507 Torque_Control_ESP32_V6_DW.obj_ds.matlabCodegenIsDeleted = true;
2508 }
2509
2510 /* End of Terminate for MATLABSystem: '<S3>/Digital Output' */
2511 /* End of Terminate for S-Function (fcgen): '<Root>/Function-Call Generator' */
2512 }
2513
```