

Bachelor Thesis



**Czech
Technical
University
in Prague**

F3

**Faculty of Electrical Engineering
Department of Control Engineering**

EV manager development

Jiří Jiráček

**Supervisor: doc. Ing. Tomáš Haniš, Ph.D.
Field of study: Cybernetics and Robotics
May 2022**

I. Personal and study details

Student's name: **JirákJi í** Personal ID number: **492369**
Faculty / Institute: **Faculty of Electrical Engineering**
Department / Institute: **Department of Control Engineering**
Study program: **Cybernetics and Robotics**

II. Bachelor's thesis details

Bachelor's thesis title in English:

EV manager development

Bachelor's thesis title in Czech:

Vývoj manageru elektromobilu

Guidelines:

The goal of the thesis is to develop and implement a Matlab & Simulink based framework for electric vehicle high level functionality management system. The thesis results will be implemented and tested on full-scale vehicle dynamics verification platform.

Following points will be addressed:

- 1) Review EV manager systems.
- 2) Define system requirements and test requirements.
- 3) Implement EV manager system for experimental platform.
- 4) Test and validate the implementation with respect to system requirements and test specification.

Bibliography / sources:

- [1] Hans-Leo Ross, Functional Safety for Road Vehicles – Subtitle New Challenges and Solutions for E-mobility and Automated Driving, Springer, Cham, ISBN: 978-3-319-33360-1
- [2] Wei Liu, Hybrid Electric Vehicle System Modeling and Control, 2nd Edition, Wiley, ISBN: 978-1-119-27932-7
- [3] Dieter Schramm, Manfred Hiller, Roberto Bardini – Vehicle Dynamics – Duisburg 2014
- [4] Robert Bosch GmbH - Bosch automotive handbook - Plochingen, Germany : Robet Bosch GmbH ; Cambridge, Mass. : Bentley Publishers

Name and workplace of bachelor's thesis supervisor:

doc. Ing. Tomáš Haniš, Ph.D. Department of Control Engineering FEE

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **28.01.2022** Deadline for bachelor thesis submission: **20.05.2022**

Assignment valid until:

by the end of summer semester 2022/2023

doc. Ing. Tomáš Haniš, Ph.D.
Supervisor's signature

prof. Ing. Michael Šebek, DrSc.
Head of department's signature

prof. Mgr. Petr Páta, Ph.D.
Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

Date of assignment receipt

Student's signature

Acknowledgements

I would like to express my thanks to all people who helped me during my bachelor studies. Especially to my family and friends for their support, to doc. Ing. Tomáš Haniš, Ph.D. for supervising this thesis and for regular consultations, to Ing. David Vošahlík and Bc. Tomáš Veselý for giving me valuable pieces of advice concerning the matter of car systems.

Declaration

I hereby declare that this thesis and the work presented in it are all my own, and all used sources are listed in the bibliography section.

Prague, 20. May 2022

Abstract

This thesis describes development of an electric vehicle high-level functionality management system for a project led by Smart Driving Solutions - a research center at the Department of Control Engineering at FEE, CTU. It focuses mainly on designing the management system with respect to functional and safety requirements of an electric vehicle, implementing a Matlab & Simulink based framework, and testing and validating the developed system.

Keywords: electric vehicle, high-level management system, Model-Based Design approach

Supervisor: doc. Ing. Tomáš Haniš, Ph.D.

Abstrakt

Tato práce se zabývá návrhem vysokoúrovňového řídicího systému elektromobilu pro projekt vedený výzkumným střediskem Smart Driving Solutions na Katedře řídicí techniky na FEL, ČVUT. Zaměřuje se především na tři části - vlastní návrh systému s ohledem na funkční a bezpečnostní požadavky elektromobilu, jeho implementaci v programu Matlab & Simulink a na následné otestování a validaci vyvinutého systému.

Klíčová slova: elektromobil, vysokoúrovňový řídicí systém, Model-Based Design přístup

Překlad názvu: Vývoj manageru elektromobilu

Contents

Acronyms	1	2.7 Steer-by-wire and Camber-by-wire System	12
1 Introduction	3	2.8 Powertrain	13
1.1 Smart Driving Solutions	3	2.9 Battery Management System	13
1.2 Drive-by-wire Concept	3	2.10 Power Supply Architecture	14
1.3 Why Use an Electric Vehicle	4	3 Developed EV Manager	17
1.4 What EV Manager Is	5	3.1 An Important Note	18
1.5 Commercial EV Managers	5	3.2 Model-based Design Approach	18
1.6 Problem Formulation	6	3.3 Stateflow	18
1.7 Objectives of Thesis	6	3.4 Important Signals and their Meaning	20
2 Full-scale Vehicle Dynamics Verification Platform	7	3.5 EV Manager Design	23
2.1 Architecture	8	3.5.1 State Communication	24
2.2 System Bus - CAN	9	3.5.2 State Diagnostics	25
2.3 Human-Machine Interface	10	3.5.3 States BBW and SBW	28
2.4 Electronic Control Unit	11	3.5.4 State Warning	29
2.5 Vehicle Measurement System	12	3.6 State MainStateMachine	30
2.6 Brake-by-wire System	12	3.6.1 State Start	33
		3.6.2 State LV	34

3.6.3 State Battery	36
3.6.4 State Motors	38
3.6.5 State CarRunning	40
3.6.6 State HV_SD	42
3.6.7 State Charging	44
3.6.8 State Error	45
4 Testing the EV Manager	55
4.1 Test Harness	55
4.2 Testing Process	56
5 Conclusion	61
A Bibliography	63

■ Acronyms

BBW Brake-by-wire

BMS battery management system

CBW Camber-by-wire

DBW Drive-by-wire

ECU electronic control unit

EV electric vehicle

EVM electric vehicle manager

HMI Human-Machine Interface

IMU inertial measurement unit

MBD model-based design

FSVDVP full-scale vehicle dynamics verification platform

SBW Steer-by-wire

SDS Smart Driving Solutions

SOC state of charge

VMS vehicle measurement system



Chapter 1

Introduction



1.1 Smart Driving Solutions

Smart Driving Solutions (SDS) is a research center at the Department of Control Engineering at FEE, CTU. According to [9], its main objectives are:

- Developing new vehicle control concepts that would provide full separation of the driver from the car in tasks of dealing with vehicle dynamics.
- Building a full-scale vehicle dynamics verification platform (FSVDVP) on which all newly developed concepts will be tested.

Currently, building FSVDVP is in progress and **Drive-by-wire** concept is being worked on.



1.2 Drive-by-wire Concept

Drive-by-wire (DBW) is a concept inspired by Fly-by-wire system developed in 1960. It is based on mechanical decoupling of the driver from motors, brakes and wheels. Instead of it, **a signal provides the coupling**. The driver still

sets the speed and direction of travel by sending control signals to electronic control unit (ECU) via Human-Machine Interface (HMI) and ECU controls wheels and brakes in the best way possible, concerning the current state of vehicle dynamics. As mentioned in [6], this could improve the safety of car systems because ECU can process much more information in a shorter period than a human driver. DBW is composed of **Motor-by-wire** for setting speed of travel, **Brake-by-wire (BBW)** for braking, **Steer-by-wire (SBW)** for setting the direction of travel, and **Camber-by-wire (CBW)** for controlling camber of wheels.

1.3 Why Use an Electric Vehicle

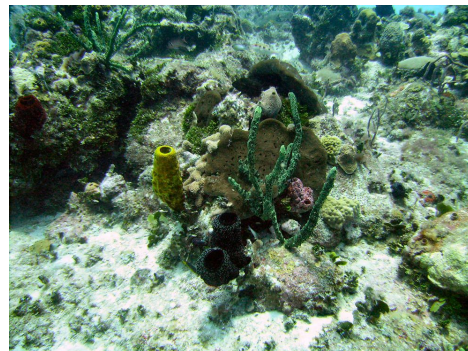
Receding glaciers, bleached corals, acidifying oceans, killer heat waves, fauna and flora in the process of extinction - all those are consequences of **global warming**. How to improve the situation?

“Even in the worst-case scenario, an electric car with a battery produced in China and driven in Poland still emits 22 % less CO₂ than diesel and 28 % less than petrol, the tool shows. In the best-case scenario, an electric car with a battery produced in Sweden and driven in Sweden can emit 80 % less CO₂ than diesel and 81 % less than petrol.”[5]

According to [5], **replacing petrol and diesel cars with electric ones might make a significant change**. That is the main reason why one should use electric cars. Except for the ecological benefits, they provide **decent acceleration and pleasant driver experience**.



(a) : Receding glaciers



(b) : Bleached corals

Figure 1.1: Consequences of global warming[7]

■ 1.4 What EV Manager Is

According to [8], automobiles are becoming less mechanical devices and more electronic appliances every day, resulting in modern electric cars having, on average, between 25 and 50 central processing units (CPUs). How to merge all those CPUs, each determined for one single purpose, into one system, providing the functionality of an electric vehicle (EV)?

The solution is to use a **central management system that builds the overall functionality of an electric vehicle by joining together lower-level functional system blocks - electric vehicle manager (EVM)**. It is not trivial to design such a management system, for it must provide not only the overall functionality but safe behavior of EV defined by ISO 26262 as well.

Someone might ask: “I worked in the automotive industry back in the ’80s, and I have never heard of such a management system, how come there did not have to be any?” The answer is simple: the number of electronics in cars then was much smaller, making the driver such a management system.

■ 1.5 Commercial EV Managers

Since different systems are used in every vehicle and the overall functionality is highly dependent on requirements specified by the customer, there is no general method for designing EVM. Nevertheless, there is a need for such a management system in every vehicle. Some companies that offer high-level management systems will be listed below:

- **Porsche Engineering** - according to [1], Porsche Engineering offers development of high-level management systems providing energy management and cooperation of electronics in the vehicle as well.
- **Valeo** - according to [2], Valeo offers interior management systems that use a camera in combination with artificial intelligence algorithms to improve driver’s safety.
- **Elaphe** - according to [3], Elaphe offers EV propulsion management systems.

1.6 Problem Formulation

In SDS group, EV described in Chapter 2 is being developed. It includes many systems which have to cooperate to provide the overall car functionality. For such a case, there is a need to develop EVM, a central control management system that merges lower-level functional blocks into one system, providing the overall car functionality and safe, deterministic behavior of FSVDVP as well.

1.7 Objectives of Thesis

To create an electric car with many concepts new to automotive industry, many challenges must be overcome. Crucial is to obtain safe and deterministic behavior of the vehicle. The main objective of this thesis is to **develop EVM** - a central control management system that builds the overall car-like functionality by joining together lower-level functional system blocks, and ensure safe, deterministic behavior of FSVDVP.

According to [20], to develop a safety-related system such as EVM is, the development at system-level has to cover following points:

- Creating a technical safety concept, both on hardware and software level.
- System integration and testing.
- Safety validation.

Accordingly with the points above, objectives of this thesis were defined as the following:

1. Designing EV manager for FSVDVP.
2. Implementing a Matlab & Simulink based framework.
3. Testing and validating the implementation.

Chapter 2

Full-scale Vehicle Dynamics Verification Platform

There is a need to build FSVDVP in order to be able to demonstrate the Drive-by-wire concept described in Section 1.2 on a real-world hardware system. This chapter aims to provide an overview of the current FSVDVP design from the point of view of EVM development. In Figure 2.1, expected appearance of FSVDVP may be seen.



Figure 2.1: Expected appearance of FSVDVP[10]

2.1 Architecture

Current FSVDVP architecture may be seen in Figure 2.2

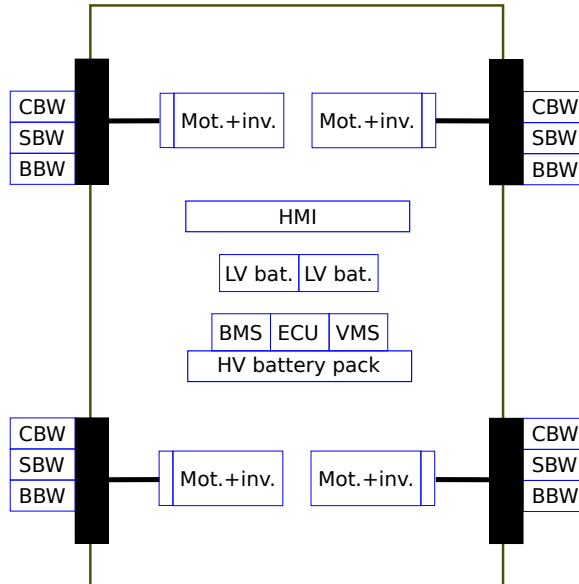


Figure 2.2: FSVDVP architecture

There may be seen the following building blocks in Figure 2.2:

1. **Human-Machine Interface - HMI**
2. **Electronic control unit - ECU**
3. **Vehicle measurement system - VMS**
4. **Brake-by-wire system - BBW**
5. **Steer-by-wire system - SBW**
6. **Camber-by-wire system - CBW**
7. **Powertrain - motors + inverters**
8. **Battery management system - BMS**
9. **Power supply - high-voltage battery pack + low-voltage batteries**

CAN Bus is used for communication between ECU and other systems. Description of the main building blocks may be seen in following sections.

2.2 System Bus - CAN

CAN (Controller Area Network) protocol provides efficient and reliable communication between sensors, actuators, controllers, and other network nodes.[15] CAN has become an automotive standard thanks to its advantages such as electromagnetic noise resistance and wiring simplicity. Due to this fact, many components already have a CAN interface from the manufacturer, which is highly beneficial when building FSVDVP.

Current version of Can Bus architecture is composed from three CAN networks:

1. **CAN1** - connects battery management system (BMS) and ECU.
2. **CAN2** - connects motors (inverters) and ECU.
3. **CAN3** - connects low-voltage systems (BBW, SBW, CBW), HMI, vehicle measurement system (VMS) and ECU.

Schematic drawing of CAN Bus architecture may be seen in Figure 2.3. CAN1

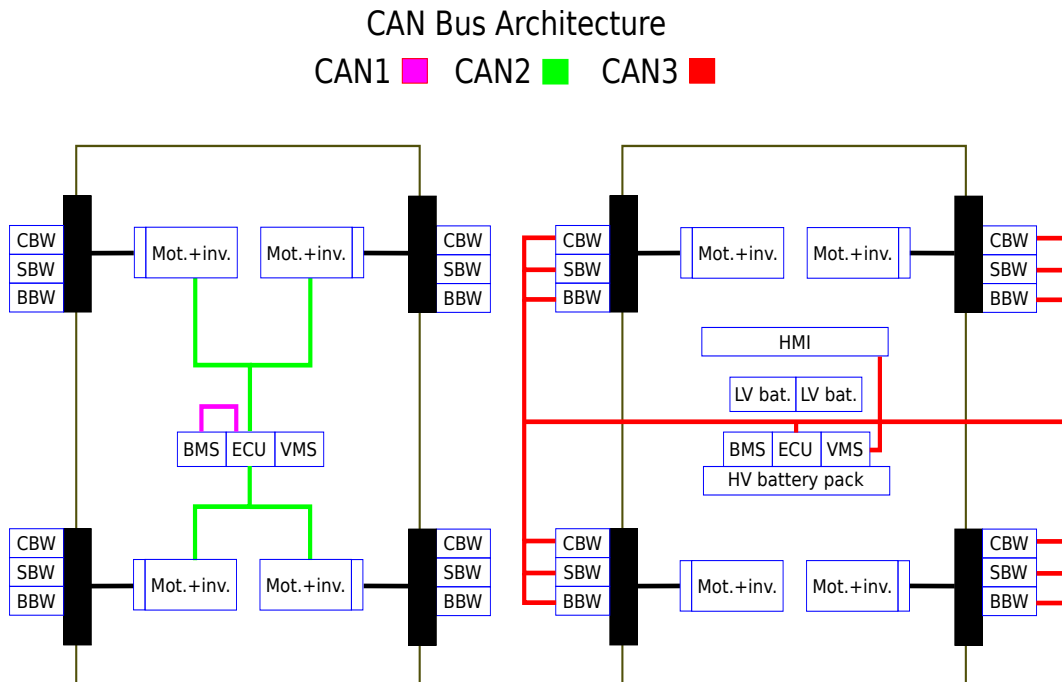


Figure 2.3: Schematic drawing of CAN Bus architecture

and CAN2 dbc files may be found at [4]. CAN3 dbc is not available yet because low-voltage systems are in the process of development.

2.3 Human-Machine Interface

Human-Machine Interface enables the operator to express their intention to ECU. A schematic drawing how HMI could look like when FSVDVP is finished may be seen in Figure 2.4.

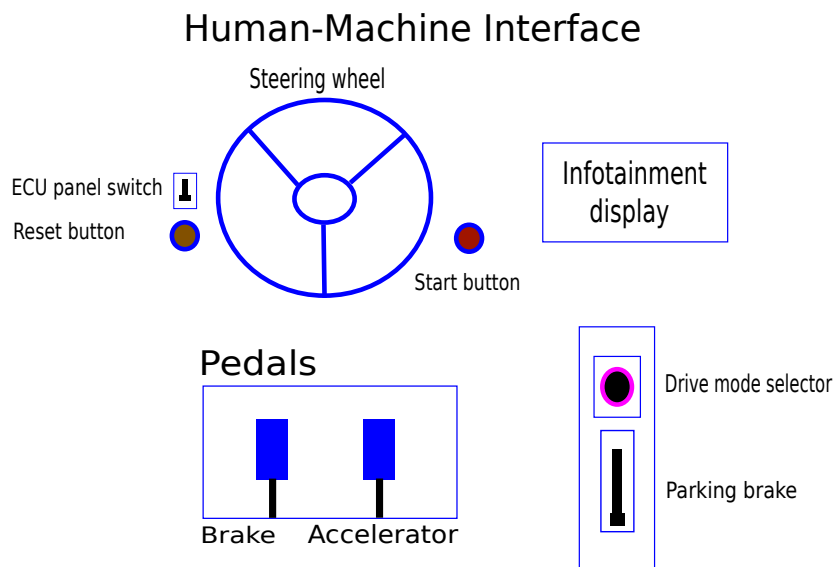


Figure 2.4: Schematic drawing of HMI

HMI contains the following essential control elements:

- **Steering wheel** - for setting the direction of travel of FSVDVP.
- **Drive mode selector** - enables the operator to choose among three modes:
 - **Neutral mode** - tires are allowed to spin freely, pushing down the accelerator pedal has no effect.
 - **Drive mode** - this mode enables the operator to move FSVDVP forward by pushing down the accelerator pedal.
 - **Reverse mode** - this mode enables the operator to move FSVDVP backward by pushing down the accelerator pedal.

- **Accelerator and brake pedal** - for setting the speed of FSVDVP.
- **Parking brake** - for keeping FSVDVP securely motionless.
- **ECU panel switch** - for turning ON/OFF FSVDVP low-voltage power supply, and therefore switching ON/OFF ECU.
- **Start button** - for starting the motors.
- **Reset button** - for resetting the system after an error occurs, to fully understand its meaning, see Chapter 3.

On top of that, HMI contains an infotainment display to provide the operator all necessary information about the state of FSVDVP.

2.4 Electronic Control Unit

An electronic control unit is a device responsible for controlling the main car systems. Used ECU is VTC 7230. According to [18], it is an Intel i3 dual-core-based embedded computer. RTLinux is be used as the operating system of ECU, for it is suitable for real-time applications.

Electric vehicle manager, developed as the main objective of this thesis, will be running on this ECU and controlling the vehicle behavior when the car is assembled. In order to be able to run EVM on ECU, Simulink coder will be used to generate C/C++ code from Matlab & Simulink based framework developed as a part of this thesis. The embedded computer may be seen in Figure 2.5.



Figure 2.5: VTC 7230 embedded computer [18]

2.5 Vehicle Measurement System

For obtaining information about vehicle dynamics, inertial measurement unit (IMU) + GPS needs to be used. The used system will be VBOX 3iS.

2.6 Brake-by-wire System

Building BBW system is in progress. When finished, it will provide the functionality of braking with no mechanical linkage between brake pedal and brakes. A signal with braking intensity is sent from the pedal to ECU which calculates optimal input to brake actuators with respect to the current state of vehicle dynamics. A system prototype may be seen in Figure 2.6.

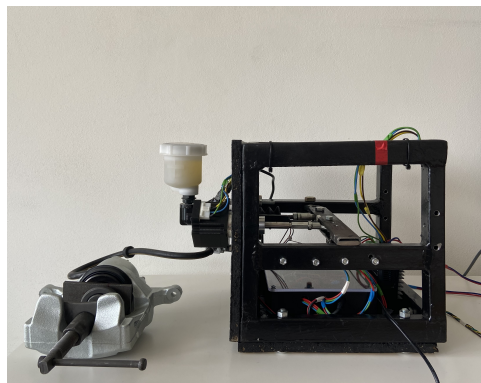


Figure 2.6: A BBW system prototype [22]

2.7 Steer-by-wire and Camber-by-wire System

Building SBW and CBW systems are planned for the next phase of FSVDP development, so no details about it will be presented in this thesis. The basic principle of the concepts may be seen in Section 1.2.

2.8 Powertrain

As may be seen in Figure 2.2, there will be four electric motors with inverters in FSVDVP, providing an option to set speed or torque reference for each wheel separately. HyPer 9HV motors with HyPer-Drive SRIPM-X144 controller/inverter will be used. Motor peak power is 90 kW at 170 V and efficiency peaks at 95 %. For more details about the motors and inverters, see [17]. SRIPM-X144 controller/inverter may be seen in Figure 2.7a, HyPer 9HV motor may be seen in Figure 2.7b.



(a) : SRIPM-X144 controller/inverter



(b) : HyPer 9HV motor

Figure 2.7: FSVDVP powertrain

2.9 Battery Management System

According to [16], using an intelligent battery management system is desirable for the following reasons:

- BMS provides monitoring of battery parameters in real time.
- BMS ensures safe and reliable use of battery.

EMUS G1 BMS with CAN interface is used in FSVDVP. Used configuration may be seen in Figure 2.8.

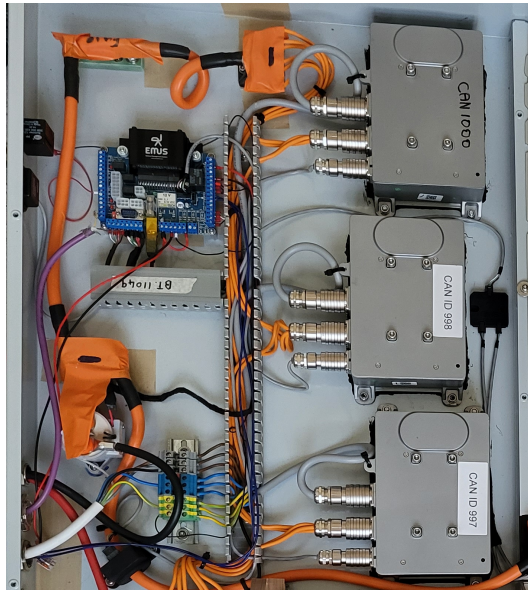


Figure 2.8: EMUS G1 BMS used configuration

2.10 Power Supply Architecture

One of the most crucial things in FSVDVP is power supply architecture - almost nothing can work without electricity in EV. The current version of power supply architecture may be seen in Figure 2.9. HV battery pack provides 170 V power supply for the powertrain system described in Section 2.8 and for recharging LV batteries with the help of DC/DC converters. LV batteries provide 12 V power supply necessary for DBW systems. BMS contactor can be controlled via BMS.

Low-voltage Batteries

There will be used two VARTA BLUE Dynamic E11 batteries as a power supply for DBW systems, BMS, HMI, VMS and ECU in FSVDVP.

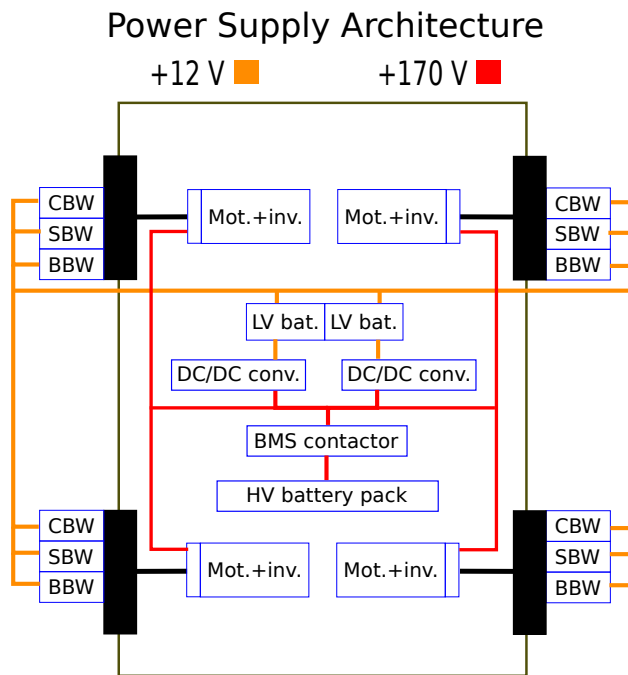


Figure 2.9: Current version of FSVDVP power supply architecture

■ High-voltage Battery Pack

There will be used seven Tesla battery modules based on Panasonic NCR-18650B Lithium-Ion rechargeable batteries in FSVDVP. According to [19], each battery nominal capacity (at 25 °C) is minimally 3250 mAh and nominal voltage is 3.6 V. HV battery pack may be seen in Figure 2.10.



Figure 2.10: HV battery pack used in FSVDVP



Chapter 3

Developed EV Manager

As mentioned in Section 1.6, EVM is a high-level central control management system that merges lower-level functional blocks into one system, providing the overall car functionality and safe, deterministic behavior of EV as well.

Developing such a system is not trivial. It is not only necessary to have at least basic knowledge about all FSVDVP architecture building blocks described in Chapter 2 but also the exact behavior of FSVDVP must be defined to obtain a safe, deterministic system.

According to [20], the development should cover the following points:

- Creating a technical safety concept, both on hardware and software level.
- System integration and testing.
- Safety validation.

In this chapter, EVM developed as the main objective of the thesis will be presented. In the beginning, model-based design (MBD) approach will be explained, emphasizing the benefits of using such an approach when developing EVM. Then all signals important for developed EVM will be presented. And then, as the main part of this chapter, developed EVM in the form of a finite-state machine will be described in detail, including system and test requirements.

■ 3.1 An Important Note

Before describing developed EVM, an important note must be stated. Due to the early phase of FSVDVP development, **current version of EVM does not include CBW**, yet integration of CBW would be analogical to the way BBW and SBW are integrated in EVM . Accordingly with the thesis specification, only a **framework without concrete implementations of using lower-level functions was developed**, yet the design provides their easy integration when the lower-level functions are finished.

■ 3.2 Model-based Design Approach

According to [14], model-based design is an approach where engineers define a model of the system with advanced functional characteristics using building blocks in an integrated software environment, where the model may be immediately simulated without any extra work. That can lead to effective prototyping, easy software testing, and simple verification of the developed system.

Stateflow from Matlab & Simulink family was used for EVM development. That is highly beneficial since all lower-level functional control blocks will be developed in Matlab & Simulink, simplifying the integration of various functional blocks into one system. Models developed in Simulink may be easily deployed on hardware via Simulink coder, which is also advantageous. Using Stateflow for MBD of EVM provides merging theoretical design and software implementation together.

■ 3.3 Stateflow

Stateflow is part of Matlab & Simulink family and provides modeling systems in the form of finite-state machines. Its basics will be described in this section to provide enough knowledge to understand the developed EVM.

Every state has three types of actions:

1. **entry** - entry actions are performed only in the simulation step when the state is entered.
2. **during** - during actions start being executed one step later the state was entered and end being executed one simulation step after the state is marked for exit (i. e., their last execution is in the simulation step when the state is marked for exit).
3. **exit** - exit actions are performed one simulation step after the state was marked for exit, i. e. one step after the last execution of during actions of the state. Entry actions of the new state and transition actions are executed in the same simulation step as exit action of the state that is being exited.

After executing states' actions, outgoing transition conditions are evaluated, and states are marked for exit if transition conditions were evaluated true. If a state has a substate, actions of both are performed - first actions of the state and then of the substate. Evaluating outgoing transitions is performed after it, resulting in executing the actions of the substate even when the state is marked for exit in the same simulation step.

There are two types of decomposition in Stateflow:

1. **Exclusive** - during one simulation step, actions of only one state are executed. When the simulation is started, a state with default transition (the arrow with blue dot, see Figure 3.1) is entered.
2. **Parallel** - during one simulation step, actions of all parallel states are executed depending on the chosen execution order. First, actions of all parallel states are executed. After it, all states are tested for exit accordingly with the chosen execution order.

At the beginning of a new simulation step after the state was marked for exit, transition actions are performed, exit actions of the state marked for exit are performed. The new state is entered, and entry actions of the new state are executed. A demonstration of Stateflow basics may be seen in Figure 3.1.

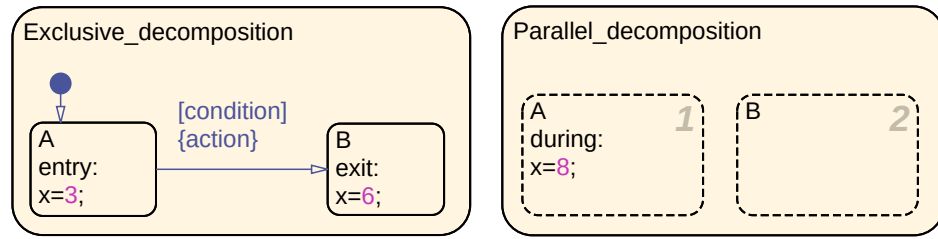


Figure 3.1: Demonstration of Stateflow basics

3.4 Important Signals and their Meaning

In this section, all signals important for the current version of EVM will be listed, and their meaning will be explained. Constants are typeset in capital letters. Their value will not be presented if not necessary for the description of EVM.

Definitions of signals with the same meaning, even when related to different systems, are below.

Definition 3.1. Let `buttonSignal` be a signal obtained from a button. Then

$$\text{buttonSignal} = \begin{cases} 1, & \text{if button is being pressed} \\ 0, & \text{otherwise.} \end{cases}$$

Definition 3.2. Let `statusSignal` be a status signal of any system. Then

$$\text{statusSignal} = \begin{cases} 1, & \text{if the system is ON} \\ 0, & \text{otherwise.} \end{cases}$$

Definition 3.3. Let `faultSignal` be a fault signal of any system. Then

$$\text{faultSignal} = \begin{cases} 1, & \text{if the system reports an error} \\ 0, & \text{otherwise.} \end{cases}$$

All important signals are stated below:

- **heartbeatMsgsCame** - denotes if all periodically broadcast messages have come since the last time they came.

$$\text{heartbeatMsgsCame} = \begin{cases} 1, & \text{if all periodically broadcast messages came} \\ 0, & \text{otherwise} \end{cases}$$

■ **HMI**

- **ParkingBrakeStatus** - state of parking brake.

$$\text{ParkingBrakeStatus} = \begin{cases} 1, & \text{if parking brake is engaged} \\ 0, & \text{otherwise} \end{cases}$$

- **DriveMode** - a signal from Drive mode selector, for meaning of the drive modes see Section 2.3.

$$\text{DriveMode} = \begin{cases} \text{N_MODE}, & \text{if neutral mode is requested} \\ \text{D_MODE}, & \text{if drive mode is requested} \\ \text{R_MODE}, & \text{if reverse mode is requested} \end{cases}$$

- **Reset** - state of Reset button, see Definition 3.1.
- **buttStart** - state of Start button, see Definition 3.1.
- **status** - see Definition 3.2.
- **fault** - see Definition 3.3.

■ **IMU**

- **speed** - the current speed of FSVDVP.
- **status** - see Definition 3.2.
- **fault** - see Definition 3.3.

■ **BBW**

- **status** - see Definition 3.2.
- **fault** - see Definition 3.3.

■ **SBW**

- **status** - see Definition 3.2.
- **fault** - see Definition 3.3.

- **BMS** - all those signals are based on G1 Unit CAN Protocol described in [12].

- **contactorStatus** - state of BMS contactor, schematic drawing of FSVDVP power supply architecture may be seen in Figure 2.9.

$$\text{contactorStatus} = \begin{cases} \text{CONT_CLOSED}, & \text{if contactor is closed} \\ \text{CONT_OPEN}, & \text{if contactor is open} \end{cases}$$

- **Charging related signals:**

- **chargingInterlock**

$$\text{chargingInterlock} = \begin{cases} 1, & \text{if charger is connected} \\ 0, & \text{otherwise} \end{cases}$$

- **chargingStage** - charging stage, for details see [13] and [12]. For purpose of developed EVM, only value CHARGER_ERROR is important. This value denotes error in charging.
- **SOC** - state of charge of HV battery pack.
- **Protection flags** - BMS protection status signals. Their definition is taken from [12]. If value of any of those signals is 1, problem defined by the description of the signal occurred.
 - **underVoltage** - some cell is below critical minimum voltage.
 - **overVoltage** - some cell is above critical maximum voltage.
 - **dischargeOverCur** - discharge current (negative current) exceeds the critical discharge current setting.
 - **chargeOverCur** - charge current (positive current) exceeds the critical charge current setting.
 - **cellModOverheat** - cell module temperature exceeds maximum critical temperature setting.
 - **Leakage** - leakage signal was detected on leakage input pin.
 - **noCellCom** - loss of communication to cells.
 - **cellOverheat** - cell temperature exceeds maximum cell temperature threshold.
 - **noCurSens** - no current sensor.
 - **packUnderVoltage** - HV battery pack under-voltage.
- **Warning flags** - BMS warning status signals. If value of any of those signals is 1, problem defined by the description of the signal occurred.
 - **lowVoltage** - some cell is below low voltage warning setting.
 - **highCurrent** - discharge current (negative current) exceeds the current warning setting.
 - **highTemp** - cell module temperature exceeds warning temperature setting.
- **Motors** - definitions of motor signals were taken from [21].
 - **MainsState** - status of a motor is determined by value of this signal.

$$\text{MainsState} = \begin{cases} \text{ALARMED}, & \text{if motor is OFF} \\ \text{PWR_RDY}, & \text{if motor is ON} \\ \text{START_UP}, & \text{if motor is ready to be switched ON} \end{cases}$$
 - **FaultLevel** - anomalous working conditions are indicated by different alarm levels, classified by Table 3.1, depending on their effects on the system.

FaultLevel	Motor main contactor	Motor	Motor outputs
BLOCKING	Opened	Disabled	Disabled
STOPPING	Closed	Stopped	Enabled
LIMITING	Closed	Limited	Enabled
WARNING	Closed	Enabled	Enabled
READY	Closed	Enabled	Enabled

Table 3.1: Table of motor alarm levels.

3.5 EV Manager Design

In this section, the implemented finite-state machine model of EVM will be presented, including defined system and test requirements. The overall design may be seen in Figure 3.2. All states will be explained thoroughly in this Chapter. In the figure, Blue boxes with a number denote the execution order of the parallel states.

The overall design is divided into several parallel state machines, their main tasks are the following:

1. **Communication** - decides, if communication between ECU and other systems in FSVDVP works.
2. **Diagnostics** - decides, if any error occurred.
3. **MainStateMachine** - provides the overall FSVDVP functionality.
4. **BBW** - provides the BBW functionality.
5. **SBW** - provides the SBW functionality.
6. **Warning** - to provide a piece of information to human operator if any system is reporting a warning.

It is highly recommended to take a look at figures relating to the state which is being described, to understand the state's function and principle of implementation fully.

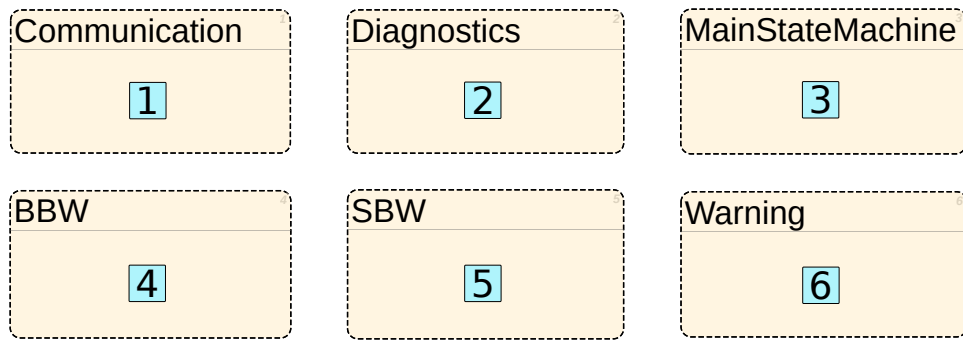


Figure 3.2: Overall EVM design

3.5.1 State Communication

The main task of this state is to decide if the communication between ECU and all other systems works. Communication is declared working, if variable `comm_status` equals to ON. Otherwise, it is declared not working.

The decision is made based on the value of `heartbeatMsgsCame` signal which denotes if all periodically broadcast messages have come since the last time they came. Communication is checked, and a new decision is made every `TO_BROADCAST_MS`, which is a certain timeout until all status and heartbeat messages were supposed to come since the last time they came. State Communication may be seen in Figure 3.3.

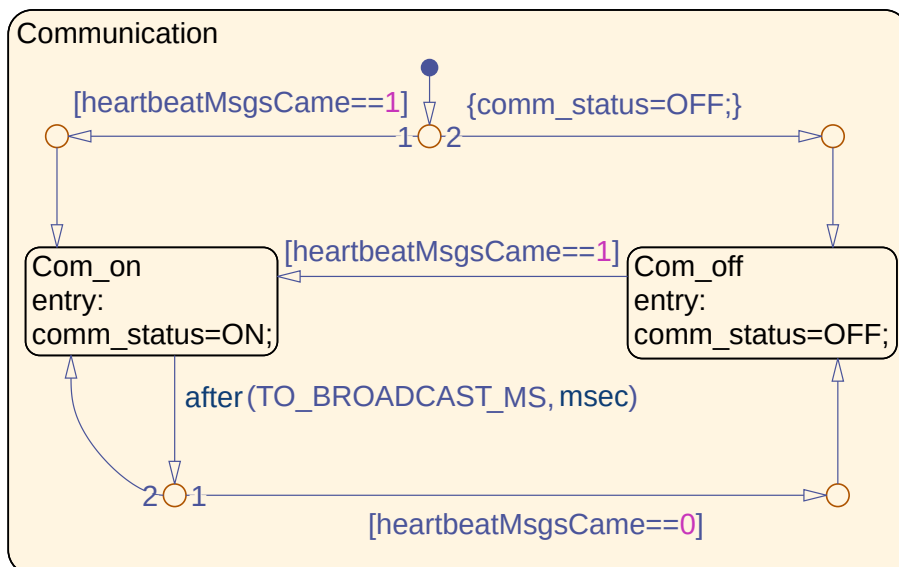


Figure 3.3: State Communication

■ 3.5.2 State Diagnostics

The main task of this state is to decide if any error occurs. For such a case, the following states of diagnostics were defined.

Definition 3.4. Let diagnostics be a variable defining state of the system. Then its value can be one of the following:

$$\text{diagnostics} = \begin{cases} \text{BLOCKING} = 1 \\ \text{STOPPING} = 2 \\ \text{LIMITING} = 3 \\ \text{NORMAL} = 5 \end{cases}$$

depending on the current state of FSVDVP.

BLOCKING and STOPPING value of diagnostics variable denotes a severe error (see Subsection 3.6.8), LIMITING denotes a minor error, which leads only to limiting motors power (see Subsection 3.6.5) and NORMAL value of diagnostics variable means all systems in FSVDVP work without any error.

This state also decides if any system reports a warning, storing the decision into variable warning. The following variables play a crucial role in determining values of diagnostics and warning:

- **charging_allowed** - for determining if connecting a charger leads to an error.
- **contactor_allowed** - for determining if CONT_CLOSE state of contactorStatus leads to an error.
- **diagnose_comm** - for determining if communication is supposed to be diagnosed.
- **diagnose_BMS** - for determining if BMS is supposed to be diagnosed.
- **diagnose_lv** - for determining if LV systems (SBW, BBW, HMI, IMU) are supposed to be diagnosed.
- **diagnose_motors** - for determining if motors are supposed to be diagnosed.

Values of these variables, and therefore the decision, what systems are supposed to be diagnosed, are determined in MainStateMachine (see Section 3.6).

Before functions crucial for determining values of diagnostics and warning variables will be presented, a definition must be stated.

Definition 3.5. Let C be set of conditions. $\text{Any}(C)$ is *true* when any of the conditions in C is *true*. $\text{None}(C)$ is *true* when none of the conditions in C is *true*.

All the functions are characterized by sets of conditions stated in their description and return two values - *diag*, *warn* - accordingly with the following:

$$\text{diag} = \begin{cases} \text{BLOCKING}, & \text{if Any}(B) \\ \text{STOPPING}, & \text{if Any}(S) \text{ and None}(B) \\ \text{LIMITING}, & \text{if Any}(L) \text{ and None}(S) \text{ and None}(B) \\ \text{NORMAL}, & \text{otherwise,} \end{cases}$$

$$\text{warn} = \begin{cases} 1, & \text{any}(W) \\ 0, & \text{otherwise,} \end{cases}$$

where B , S , L , W are sets of conditions. Those return values are assigned to diagnostics and warning variables every simulation step, for details see the implementation in Figure 3.4. The functions are the following:

- **LV_check** - checks if there is an error concerning LV systems. Sets of conditions characterizing the function are:

$$\begin{aligned} B &= \{\text{any from LV systems is OFF; any LV system's fault signal equals to 1}\} \\ S &= \emptyset \\ L &= \emptyset \\ W &= \emptyset. \end{aligned}$$

- **BMS_check** - checks if there is an error or warning concerning BMS. Sets of conditions characterizing the function are:

$$\begin{aligned} B &= \{\text{any from BMS Protection flags signal equals to 1;} \\ &\quad \text{BMS contactorStatus equals to CONT_OPEN}\} \\ S &= \{\text{BMS SOC} < \text{SOC_STOP_THRESHOLD}\} \\ L &= \emptyset \\ W &= \{\text{any BMS Warning flags signal's value equals to 1;} \\ &\quad \text{SOC_STOP_THRESHOLD} < \text{BMS SOC} < \text{SOC_WARN_THRESHOLD.}\} \end{aligned}$$

- **MOT_check** - checks if there is an error or warning concerning motors.

Sets of conditions characterizing the function are:

$$B = \{\text{any motor is OFF; any motor's FaultLevel equals to BLOCKING}\}$$

$$S = \{\text{any motor's FaultLevel equals to STOPPING}\}$$

$$L = \{\text{any motor's FaultLevel equals to LIMITING}\}$$

$$W = \{\text{any motor's FaultLevel equals to WARNING}\}$$

- **LV_BMS_check** - checks if an error or warning concerning LV systems or BMS occurred. It is characterized by:

$$\text{diag} = \min\{\text{diag from LV_check, diag from BMS_check}\}$$

$$\text{warn} = \max\{\text{warn from LV_check, warn from BMS_check}\}$$

- **BMS_MOT_check** - checks if an error or warning concerning BMS or motors occurred. It is characterized by:

$$\text{diag} = \min\{\text{diag from BMS_check, diag from MOT_check}\}$$

$$\text{warn} = \max\{\text{warn from BMS_check, warn from MOT_check}\}$$

- **LV_BMS_MOT_check** - checks if an error or warning concerning LV systems or BMS or motors occurred. It is characterized by:

$$\text{diag} = \min\{\text{diag from LV_check, diag from BMS_check, diag from MOT_check}\}$$

$$\text{warn} = \max\{\text{warn from LV_check, warn from BMS_check, warn from MOT_check}\}$$

State Diagnostics may be seen in Figure 3.4. Transition conditions and actions were removed from the figure and are the following:

1. **transition:** $[\text{diagnose_lv} == \text{ON} \ \&\& \ \text{diagnose_BMS} == \text{ON} \ \&\& \ \text{diagnose_motors} == \text{ON}] \{\text{diagnostics, warning} = \text{LV_BMS_MOT_check};\}$
comment: LV systems, BMS and motors are being diagnosed.
2. **transition:** $[\text{diagnose_lv} == \text{ON} \ \&\& \ \text{diagnose_BMS} == \text{ON}] \{\text{diagnostics, warning} = \text{LV_BMS_check};\}$
comment: LV systems and BMS are diagnosed.
3. **transition:** $[\text{diagnose_lv} == \text{ON} \ \&\& \ \text{diagnose_motors} == \text{ON}] \{\text{diagnostics} = \text{BLOCKING};\}$
comment: Not supposed to happen, motors are not supposed to be diagnosed without BMS being diagnosed.
4. **transition:** $[\text{diagnose_BMS} == \text{ON} \ \&\& \ \text{diagnose_motors} == \text{ON}] \{\text{diagnostics, warning} = \text{BMS_MOT_check};\}$
comment: BMS and motors are diagnosed, may happen when LV systems have been switched OFF before motors are switched OFF and before BMS contactor is opened, for details see Subsection 3.6.8.

5. **transition:** `[diagnose_LV==ON] {diagnostics, warning = LV_check;}`
comment: Only LV systems are being diagnosed.
6. **transition:** `[diagnose_BMS==ON] {diagnostics, warning = BMS_check;}`
comment: BMS is diagnosed, may happen when motors and LV systems have been switched OFF before BMS contactor is opened, for details see Subsection 3.6.8.
7. **transition:** `[diagnose_motors==ON] {diagnostics = BLOCKING;}`
comment: Not supposed to happen, motors are not supposed to be diagnosed without BMS being diagnosed.
8. **transition:** `[(HMI status==ON && HMI fault==1) || (IMU status==ON && IMU fault==1)] {diagnostics = STOPPING;}`
comment: May happen when LV systems except HMI and IMU were shut down in Blocking error (see Subsection 3.6.8) and HMI or IMU reports an error.
9. **transition:** `[charging_allowed==0 && BMS chargingInterlock==1] {diagnostics = STOPPING;}`
comment: Charger has been connected when not supposed to be.
10. **transition:** `[diagnose_comm==ON && comm_status==OFF] {diagnostics = BLOCKING;}`
comment: Communication stopped working, see Subsection 3.5.1.
11. **transition:** `[contactor_allowed==0 && BMS contactorStatus==CONT_CLOSED] {diagnostics = BLOCKING;}`
comment: BMS contactor is closed when not supposed to be.
12. **transition:** `[entry action]{warning = 0; charging_allowed = 0; contactor_allowed = 0; diagnostics = NORMAL; diagnose_comm = OFF; diagnose_lv = OFF; diagnose_BMS = OFF; diagnose_motors = OFF;}`
comment: Initialize variables.

■ 3.5.3 States BBW and SBW

State BBW provides the BBW functionality. Since BBW lower-level system functions have not been finished yet, only empty substates where those functions will be inserted are present in the state. When bbw system is ON and reports no error, its functionality will be being used. State SBW provides the SBW functionality. The principle of the state is analogical to state BBW, therefore will not be commented again. To fully understand the principle of states BBW and SBW, see Figure 3.5.

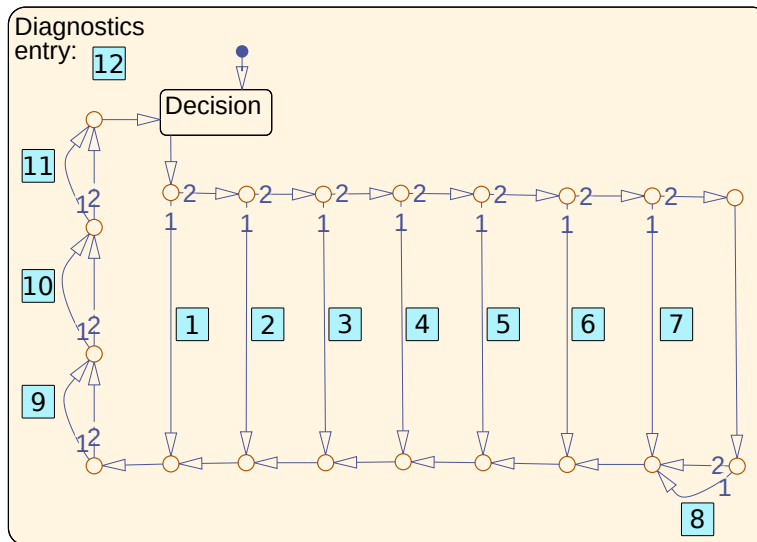


Figure 3.4: State Diagnostics

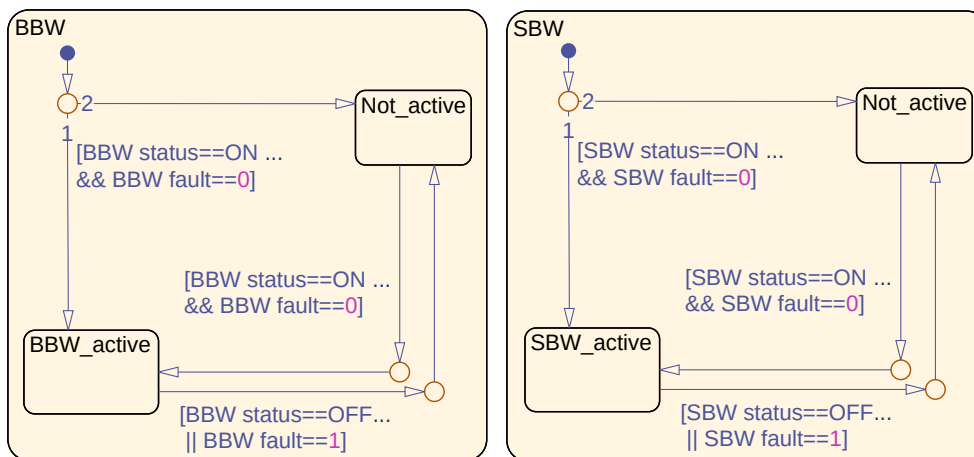


Figure 3.5: States BBW and SBW

3.5.4 State Warning

When a system reports a warning, this state will provide information about the warning to the human operator. The exact form of the information has not been chosen yet. It will probably be displaying a warning message on the Infotainment display in HMI (see Subsection 2.3).

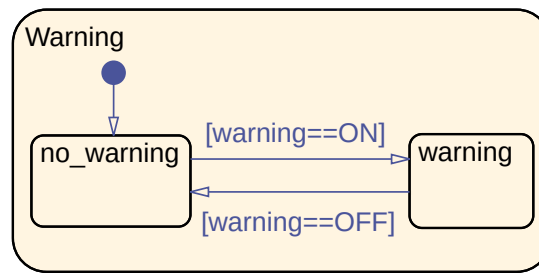


Figure 3.6: State Warning

3.6 State MainStateMachine

As the name indicates, State MainStateMachine is the main state machine that determines the behavior of FSVDVP. Its design may be seen in Figure 3.7. All substates of MainStateMachine will be described in this section. Error detection is performed in state Diagnostics, parallel to MainStateMachine, for details see Subsection 3.5.2. Transition conditions and actions were removed from the figure and are the following:

1. **transition:** {limited_mode = OFF; state = NO_VALUE;}
comment: Default transition. Variable limited_mode is for limiting motors power by software.
2. **transition:** after(TO_COM_ON_MS, msec) {diagnostics = BLOCKING; }
comment: Communication has not been established until defined timeout.
3. **transition:** [diagnostics==BLOCKING || diagnostics==STOPPING]
comment: An error occurred.
4. **transition:** Empty.
comment: An empty transition. Communication has been established. See state Start (Subsection 3.6.1) for details.
5. **transition:** [diagnostics==BLOCKING || diagnostics==STOPPING]
comment: An error occurred.
6. **transition:** after(PERIOD_LV_ON_MS, msec) [count==N_TRIES] {diagnostics=STOPPING;}
comment: Request to switch ON LV systems has not been followed until PERIOD_LV_ON_MS timeout for N_TRIES times. For details see state LV (Subsection 3.6.2).

7. **transition:** [BMS chargingInterlock == 1]
comment: A charger has been connected.
8. **transition:** [BMS chargingInterlock == 0]
comment: A charger has been disconnected.
9. **transition:** Empty.
comment: An empty transition. LV systems have been switched ON. See state LV (Subsection 3.6.2) for details.
10. **transition:** [diagnostics==BLOCKING || diagnostics==STOPPING]
comment: An error occurred.
11. **transition:** after(PERIOD_CLOSE_CONT_MS, msec) [count==N_TRIES]
{diagnostics=STOPPING;}
comment: Request to close BMS contactor has not been followed until PERIOD_CLOSE_CONT_MS timeout for N_TRIES times. For details see state Battery (Subsection 3.6.3).
12. **transition:** Empty.
comment: An empty transition. BMS contactor has been closed. See state Battery (Subsection 3.6.3) for details.
13. **transition:** [diagnostics==BLOCKING || diagnostics==STOPPING]
comment: An error occurred.
14. **transition:** Empty.
comment: An empty transition. Request to reset motors or to start motors has not been followed for N_TRIES times. For details see state Motors (Subsection 3.6.4).
15. **transition:** Empty.
comment: An empty transition. Motors have been started. See state Motors (Subsection 3.6.4) for details.
16. **transition:** [diagnostics==BLOCKING || diagnostics==STOPPING]
comment: An error occurred.
17. **transition:** [HMI buttStart==1 && IMU speed < STOPPED_THRESHOLD && HMI parkingBrakeStatus == 1]
comment: Follow a request to shut down motors from the human operator.
18. **transition:** Empty.
comment: An empty transition. Request to shut down motors or to open BMS contactor has not been followed for N_TRIES times. For details see state HV_SD (Subsection 3.6.6).
19. **transition:** [diagnostics==BLOCKING || diagnostics==STOPPING]
comment: An error occurred.

- 20. **transition:** Empty.
comment: An empty transition. Motors have been shut down and BMS contactor has been opened. See state HV_SD (Subsection 3.6.6) for details.

- 21. **transition:** [BMS chargingStage == CHARGER_ERROR || HMI parkingBrakeStatus == 0 || IMU Speed > STOPPED_THRESHOLD] {diagnostics=STOPPING;}
comment: BMS reports an error in charging process or FSVDVP is not standing or the parking brake is not engaged.

- 22. **transition:** [diagnostics==BLOCKING || diagnostics==STOPPING]
comment: An error occurred.

- 23. **transition:** [error_resolved==1] {diagnostics=NORMAL;}
comment: Reset of FSVDVP. See state Error (Subsection 3.6.8) for details.

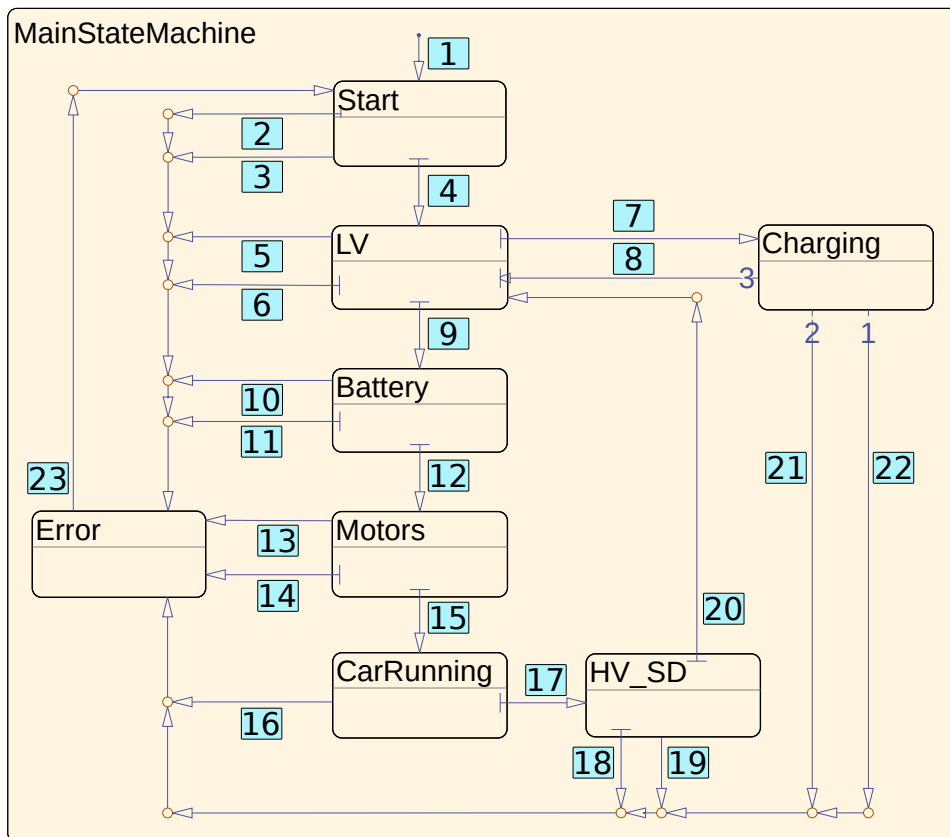


Figure 3.7: State MainStateMachine

3.6.1 State Start

State Start is determined for establishing communication between ECU and all systems in FSVDVP meant to communicate with it - SBW, BBW, IMU, HMI, BMS, and motors (inverters).

Start is entered when:

1. ECU is switched ON from OFF.
2. Reset from state ERROR is performed.

After entering Start, wait until the communication is established. Change state to LV when the communication is established.

Change state to Error when any of the following conditions is met:

- Charger is connected.
- BMS contactor is closed.
- Communication is not established until defined timeout.
- All LV systems are supposed to be ON (after reset from Stopping error - see Subsection 3.6.8) and any is OFF or any is reporting an error.
- HMI is ON and reports an error or IMU is ON and reports an error (HMI and IMU may be ON without other LV systems being ON after reset from Blocking error - see Subsection 3.6.8)

Implementation of state Start may be seen in Figure 3.8. Due to the parallel state Diagnostics, its execution order (actions of state Diagnostics are performed before actions of MainStateMachine) and Stateflow workflow for state chart execution, Go_to_LV substate had to be inserted into state Start to ensure communication is diagnosed (thanks to `diagnose_comm = ON;`) before entering state LV. If `diagnose_comm = ON;` were performed as transition action when exiting Start and entering LV, communication would not be diagnosed until the next simulation step. The analogical principle will be used in some of the other states and not commented again.

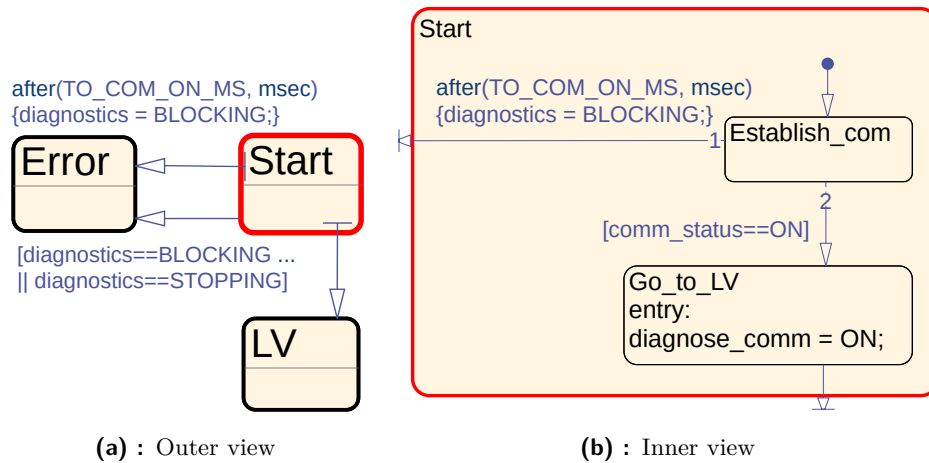


Figure 3.8: State Start

3.6.2 State LV

State LV is determined for switching LV systems ON and then waiting for further action requested by the human operator.

LV is entered when:

1. Previous state was Start and communication was established.
2. Previous state was Charging and the charger was disconnected.
3. Previous state was HV_SD and motors were shut down and BMS contactor opened.

When LV is entered, switch ON all LV systems (SBW, BBW, HMI, IMU) that are OFF. Then wait until a charger is connected - change state to Charging - or the Start button is pressed - change state to Battery.

Change state to ERROR when any of the following conditions is met:

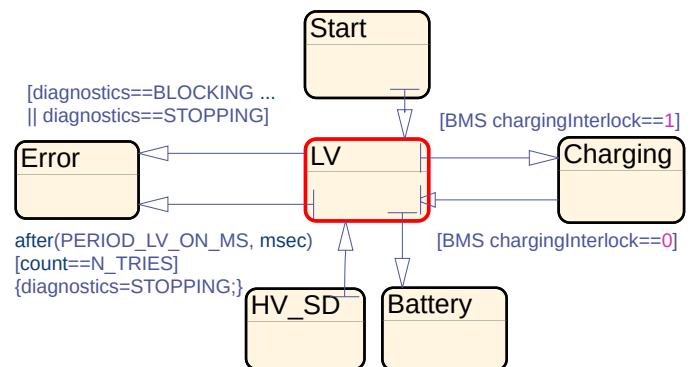
- BMS contactor is closed.
- Communication stopped working.
- LV systems have not been switched ON yet and charger is connected.

- Request for switching ON LV systems has not been followed until defined timeout for defined number of times.
- Before switching ON LV systems, only HMI pedal unit and IMU are ON (after reset from Blocking error - see Subsection 3.6.8) and HMI or IMU reports an error.
- After switching ON LV systems, any reports an error or any is OFF.

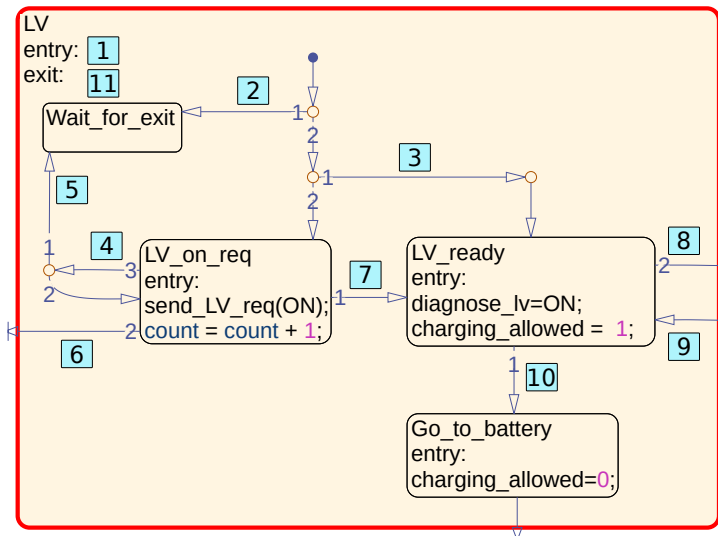
State LV may be seen in Figure 3.9. Transition conditions and actions were removed from the figure and are the following:

1. **transition:** [Entry action] {state = STATE_LV; count = 0;}.
comment: Variable count is for determining how many times a request has been sent.
2. **transition:** [diagnostics==BLOCKING || diagnostics==STOPPING]
comment: Do not send request request if an error is diagnosed in the same simulation step as LV is entered. See Section 3.3 for details about Stateflow substates and their execution order.
3. **transition:** [all_LV_ON(IN) == ON]
comment: Do not switch LV systems ON if they are all ON yet.
4. **transition:** after(PERIOD_LV_ON_MS, msec)
comment: Request to switch on LV systems has not been followed until timeout PERIOD_LV_ON_MS. The request has been sent less than N_TRIES times yet so send it again if no error occurred. If an error occurred, wait for exit and do not send a new request.
5. **transition:** [diagnostics==BLOCKING || diagnostics==STOPPING]
comment: Do not send a request if an error is diagnosed in the same simulation step. See Section 3.3 for details about Stateflow substates and their execution order.
6. **transition:** after(PERIOD_LV_ON_MS, msec) [count==N_TRIES]
{diagnostics=STOPPING;}
comment: Request for switching ON LV systems has not been followed until PERIOD_LV_ON_MS timeout for N_TRIES times so change state to Error.
7. **transition:** [all_LV_ON(IN) == ON]
comment: LV systems have been switched ON.
8. **transition:** [BMS chargingInterlock == 1]
comment: A charger has been connected.

- 9. **transition:** [BMS chargingInterlock == 0]
comment: A charger has been disconnected.
- 10. **transition:** [HMI buttStart == 1]
comment: HMI Start button has been pressed. Go close BMS contactor - change state to Battery.
- 11. **condition:** [Exit action] {count = 0;}
comment: Variable count is for determining how many times a request has been sent.



(a) : Outer view



(b) : Inner view

Figure 3.9: State LV

3.6.3 State Battery

This state is determined for closing BMS contactor. State Battery is entered after waiting in state LV and pressing Start button. After entering Battery,

close BMS contactor. After it is closed, change state to Motors.

Change state to ERROR when any of the following conditions is met:

- Charger is connected.
- Communication stopped working.
- Request to close BMS contactor has not been sent so far, yet the contactor is closed.
- Any LV system reports an error or any is OFF.
- Request to close BMS contactor has not been followed until defined timeout for defined number of times.

State Battery may be seen in Figure 3.10. Transition conditions and actions were removed from the figure and are the following:

1. **transition:** [entry action] {count = 0;}
comment: Variable count is for determining how many times a request has been sent.
2. **condition:** [diagnostics==BLOCKING || diagnostics==STOPPING]
comment: Do not send request request if an error is diagnosed in the same simulation step as Battery is entered. See Section 3.3 for details about Stateflow substates and their execution order.
3. **transition:** after(PERIOD_CLOSE_CONT_MS, msec)
comment: Request to close BMS contactor has not been followed until timeout PERIOD_CLOSE_CONT_MS. The request has been sent less than N_TRIES times yet, so send it again if no error occurred. If an error occurred, wait for exit and do not send a new request.
4. **transition:** [diagnostics==BLOCKING || diagnostics==STOPPING]
comment: Do not send request request if an error is diagnosed in the same simulation step. See Section 3.3 for details about Stateflow substates and their execution order.
5. **transition:** after(PERIOD_CLOSE_CONT_MS, msec)[count==N_TRIES]
{diagnostics=STOPPING;}
comment: Request for closing BMS contactor has not been followed until PERIOD_CLOSE_CONT_MS timeout for N_TRIES times. Change state to Error.

- 6. **transition:** [BMS contactorStatus==CONT_CLOSED]
comment: BMS contactor has been successfully closed, go to state Motors.
- 7. **transition:** [exit action] {count = 0;}.
comment: Variable count is for determining how many times a request has been sent.

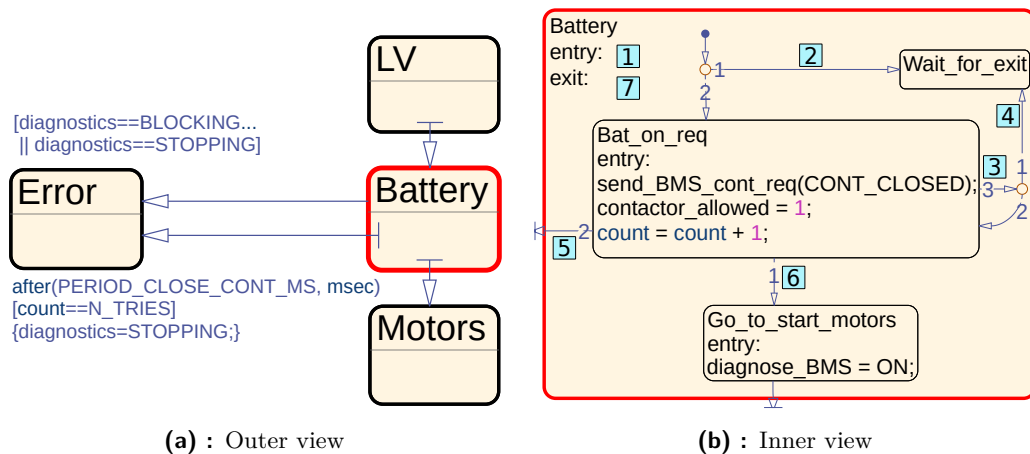


Figure 3.10: State Battery

3.6.4 State Motors

This state is determined for switching ON motors. It is entered after closing BMS contactor in Battery. After entering Motors, start motors. Motors MainsState must be START_UP to be able to start them. If it is not, resetting motors is necessary to be able to start them. Change state to CarRunning when all motors are successfully switched ON.

Change state to Error when any of the following conditions is met:

- Charger is connected.
- Communication stopped working.
- Any LV system reports an error or any is OFF.
- BMS reports an error.
- Very low state of charge (SOC) of battery.
- BMS contactor is open.

- Request to reset motors has not been followed until defined timeout for defined number of times.
- Request to switch motors ON has not been followed until defined timeout for defined number of times.

State Motors may be seen in Figure 3.11. Transition conditions and actions were removed from the figure and are the following:

1. **transition:** [entry action] {count = 0;}
comment: Variable count is for determining how many times a request has been sent.
2. **transition:** [diagnostics==BLOCKING || diagnostics==STOPPING]
comment: Do not send request request if an error is diagnosed in the same simulation step as state Motors is entered. See Section 3.3 for details about Stateflow substates and their execution order.
3. **transition:** [all_motors_mainsState(IN)==START_UP]
comment: MainsState of all motors is START_UP, so motors are ready to be started.
4. **transition:** after(PERIOD_MOT_ON_MS, msec)
comment: Request to start motors has not been followed until timeout PERIOD_MOT_ON_MS. The request has been sent less than N_TRIES times yet, so send it again if no error occurred. If an error occurred, wait for exit and do not send a new request.
5. **transition:** [diagnostics==BLOCKING || diagnostics==STOPPING]
comment: Do not send request request if an error is diagnosed in the same simulation step. See Section 3.3 for details about Stateflow substates and their execution order.
6. **transition:** [all_motors_mainsState(IN)==START_UP] {count = 0;}
comment: MainsState of all motors is START_UP, so motors are ready to be started.
7. **transition:** after(PERIOD_MOT_RESET_MS, msec)
comment: Request to reset motors has not been followed until timeout PERIOD_MOT_RESET_MS. The request has been sent less than N_TRIES times yet, so send it again if no error occurred. If an error occurred, wait for exit and do not send a new request.
8. **transition:** [diagnostics==BLOCKING || diagnostics==STOPPING]
comment: Do not send request request if an error is diagnosed in the same simulation step. See Section 3.3 for details about Stateflow substates and their execution order.

9. **transition:** `after(PERIOD_MOT_RESET_MS, msec) [count==N_TRIES]`
`{diagnostics=STOPPING;}`
comment: Request for resetting motors has not been followed until PERIOD_MOT_RESET_MS timeout for N_TRIES times.
10. **transition:** `after(PERIOD_MOT_ON_MS, msec) [count==N_TRIES]`
`{diagnostics=STOPPING;}`
comment: Request for switching motors ON has not been followed until PERIOD_MOT_ON_MS timeout for N_TRIES times.
11. **transition:** `[all_motors_mainsState(IN)==PWR_RDY]`
comment: MainsState of all motors is PWR_RDY, so all motors are ON.
12. **transition:** `[exit action] {count = 0;}`.
comment: Variable count is for determining how many times a request has been sent.

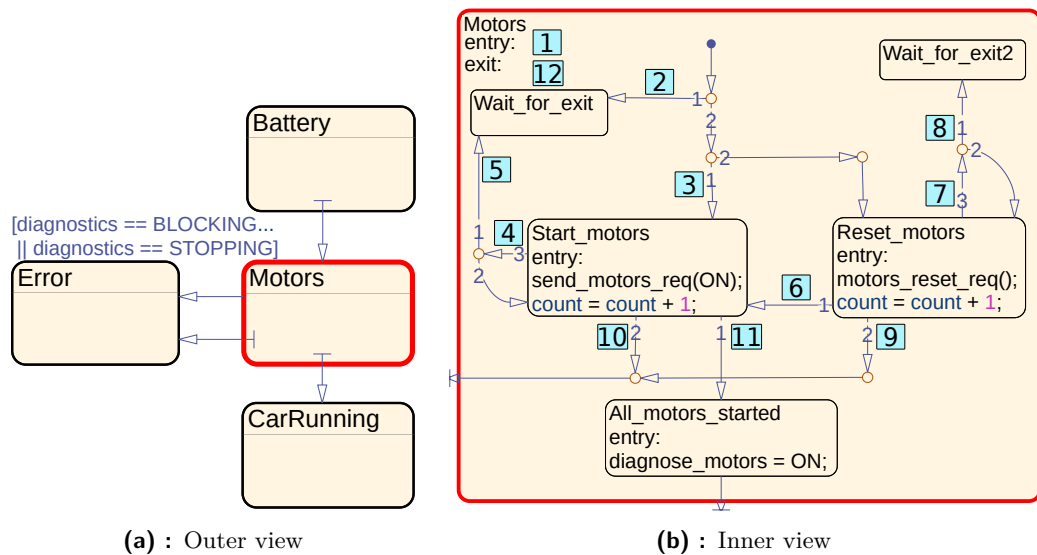


Figure 3.11: State Motors

3.6.5 State CarRunning

This state is determined for providing full car functionality of FSVDVP. It is entered after all motors are successfully turned ON in state Motors. After entering state CarRunning, FSVDVP starts in neutral mode.

Purposes of the modes are the following:

- **Neutral mode** - this mode allows tires to spin freely and might come in handy when there is a need to push the car or have it towed.
- **Drive mode** - this mode allows the driver to move FSVDVP forward, setting its speed via accelerator pedal. The power of all motors may be limited in this mode when the state of motors requires it (diagnostics == LIMITING).
- **Reverse mode** - this mode allows the driver to move FSVDVP backward, setting its speed via accelerator pedal. The power of all motors is limited in this mode.

Switching mode may be performed only when driver sends request to do so via drive mode selector and FSVDVP is standing.

Change state to HV_SD when in neutral mode, FSVDVP is standing and the human operator presses Start button.

Change state to Error when any of the following conditions is met:

- Charger is connected.
- Communication stopped working.
- Any LV system reports an error or any is OFF.
- BMS reports an error.
- Very low SOC of battery.
- BMS contactor is open.
- Any motor reports an error or any is OFF.

State CarRunning may be seen in Figure 3.12. Transition conditions and actions were removed from the figure and are the following:

1. **transition:** [HMI DriveMode == R_MODE && IMU speed < STOPPED_THRESHOLD]
comment: Switch to reverse mode.
2. **transition:** [HMI DriveMode == N_MODE && IMU speed < STOPPED_THRESHOLD]
comment: Switch to neutral mode.

3. **transition:** [HMI DriveMode == D_MODE
&& IMU speed < STOPPED_THRESHOLD]
comment: Switch to drive mode.
4. **transition:** [diagnostics==LIMITING || limited_mode==1]
comment: Switch to drive mode with limited power of motors.
5. **transition:** [diagnostics==LIMITING || limited_mode==1]
comment: Switch to drive mode with limited power of motors.
6. **transition:** [HMI DriveMode == N_MODE
&& IMU speed < STOPPED_THRESHOLD]
comment: Switch to neutral mode.
7. **transition:** [HMI DriveMode == N_MODE
&& IMU speed < STOPPED_THRESHOLD]
comment: Switch to neutral mode.
8. **transition:** [HMI buttStart==1
&& IMU speed < STOPPED_THRESHOLD]
comment: Go shut down motors and open BMS contactor.

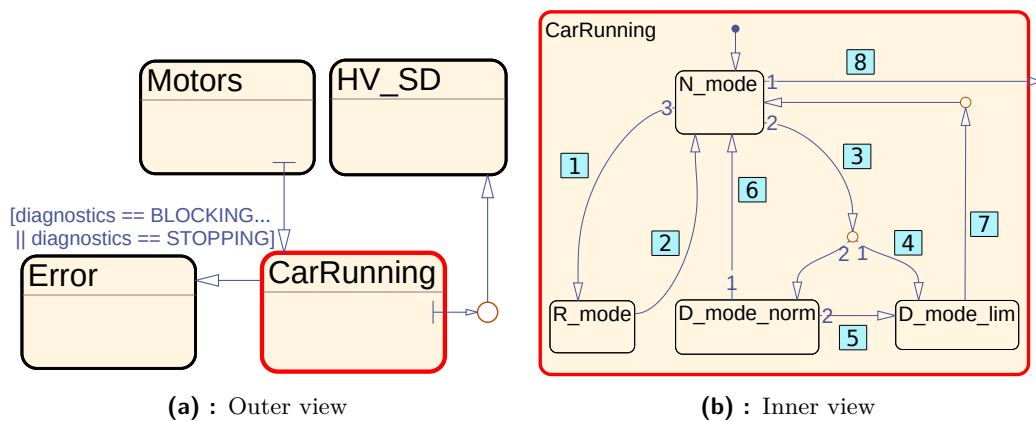


Figure 3.12: State CarRunning

3.6.6 State HV_SD

This state is determined for shutting down HV systems - i. e., turning OFF motors and opening BMS contactor. It is entered after the human operator requests turning OFF motors when in State CarRunning via pressing Start button. Change state to LV when motors are OFF, and BMS contactor is closed.

Change state to Error when any of the following conditions is met:

- A charger is connected.
- Communication stopped working.
- Any LV system reports an error or any is OFF.
- Before a request to open BMS contactor is sent, BMS contactor is open or BMS reports an error.
- Before a request to switch OFF motors is sent, any motor reports an error or any is OFF.
- Any LV system reports an error or any is OFF.
- Request to open BMS contactor has not been followed until defined timeout for defined number of times.
- Request to shut down motors has not been followed until defined timeout for defined number of times.

State HV_SD may be seen in Figure 3.13. Transition conditions and actions were removed from the figure and are the following:

1. **transition:** [entry action] {count = 0;}
comment: Variable count is for determining how many times a request has been sent.
2. **transition:** [all_motors_mainsState(IN)~=ALARMED]
comment: Not all motors are OFF, shut down the ones that are ON.
3. **transition:** [all_motors_mainsState(IN)==ALARMED] {count = 0;}
comment: All motors are OFF, open BMS contactor.
4. **transition:** after(PERIOD_MOT_OFF_MS, msec) [count==N_TRIES]
{diagnostics=STOPPING;}
comment: Request for shutting down motors has not been followed until PERIOD_MOT_OFF_MS timeout for N_TRIES times.
5. **transition:** after(PERIOD_MOT_OFF_MS, msec)
comment: Request to start motors has not been followed until timeout PERIOD_MOT_OFF_MS. The request has been sent less than N_TRIES times yet so send it again if no error occurred. If an error occurred, wait for exit and do not send a new request.
6. **transition:** BMS contactorStatus == CONT_CLOSED
comment: Send request to open BMS contactor.

7. **transition:** `after(PERIOD_OPEN_CONT_MS, msec) [count==N_TRIES]`
`{diagnostics=STOPPING;}`
comment: Request for opening BMS contactor has not been followed until `PERIOD_OPEN_CONT_MS` timeout for `N_TRIES` times. Change state to Error.
8. **transition:** `after(PERIOD_OPEN_CONT_MS, msec)`
comment: Request to open BMS contactor has not been followed until timeout `PERIOD_OPEN_CONT_MS`. The request has been sent less than `N_TRIES` times yet so send it again if no error occurred. If an error occurred, wait for exit and do not send a new request.
9. **transition:** `[BMS contactorStatus == CONT_OPEN]`
comment: BMS contactor has been opened, HV is shut down, go to state LV.
10. **condition:** `[exit action] {count = 0;}`
comment: Variable `count` is for determining how many times a request has been sent.

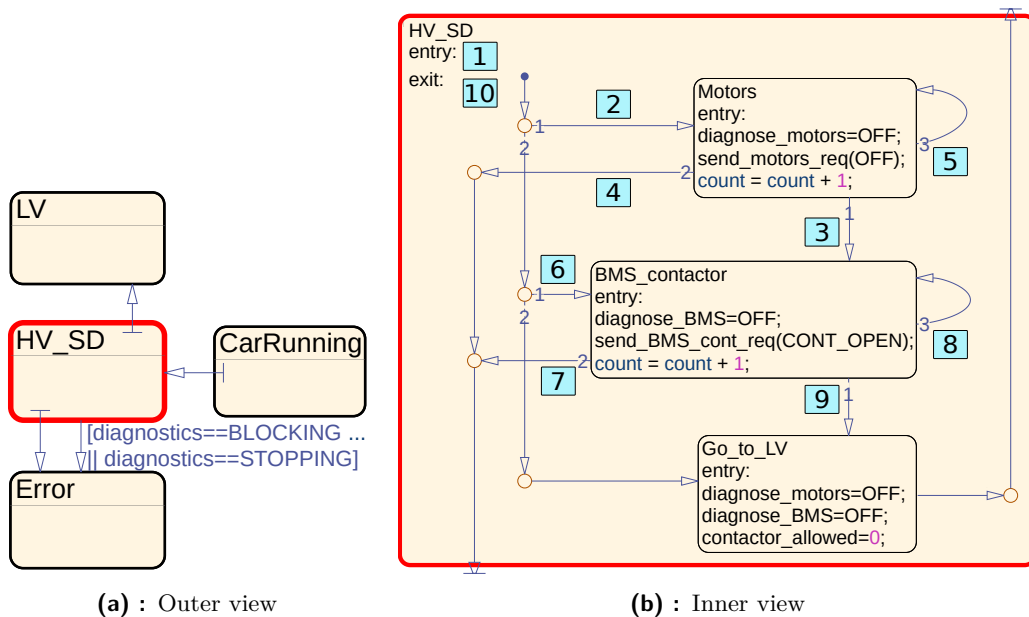


Figure 3.13: State HV_SD

3.6.7 State Charging

State Charging is determined for charging FSVDVP. It is entered if a charger is connected while waiting in state LV. Parking brake must be engaged, and FSVDVP must be standing while in this state. When the charger is disconnected, change state to LV.

Change state to Error when any of the following conditions is met:

- BMS contactor is closed.
- BMS reports an error in the charging process.
- Communication stopped working.
- Parking brake is disengaged.
- FSVDVP is moving.
- Any LV system reports an error or any is OFF.

State Charging may be seen in Figure 3.14.

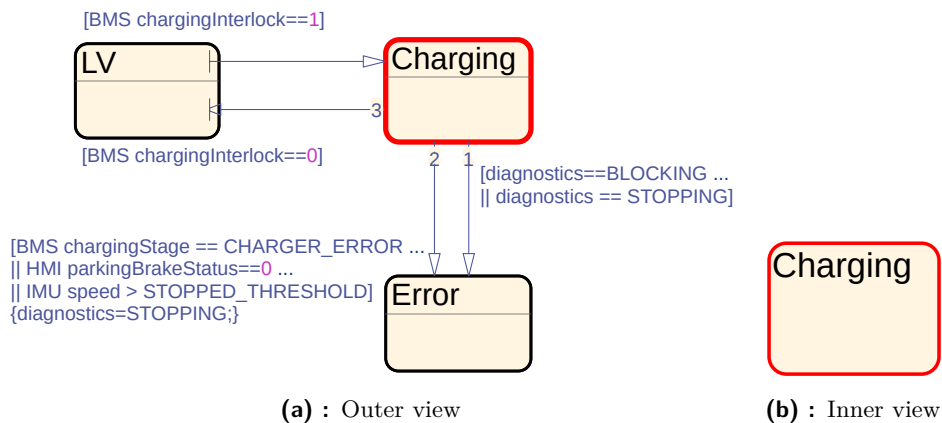


Figure 3.14: State Charging

■ 3.6.8 State Error

State Error is entered after an error occurs in any other state. Parallel state Diagnostics (see Subsection 3.5.2) is used for error detection. When a request to switch a system ON/OFF is not repeatedly followed until a defined timeout for a defined number of times, state is also changed to Error. See previous parts of this section for details.

Two types of errors were defined:

- **Blocking errors** - when such an error occurs, SBW, BBW, and motors must be shut down, BMS contactor must be opened.
- **Stopping errors** - when such an error occurs, motors must be shut down and BMS contactor must be opened.

When all systems that are supposed to be shut down (depending on the error severity - described above) are shut down, FSVDVP was stopped, parking brake is engaged and no charger is connected, change state to Start on pressing Reset button. State Error may be seen in Figure 3.15. Substates Blocking_error and Stopping_error provide resolution of the error depending on its type.

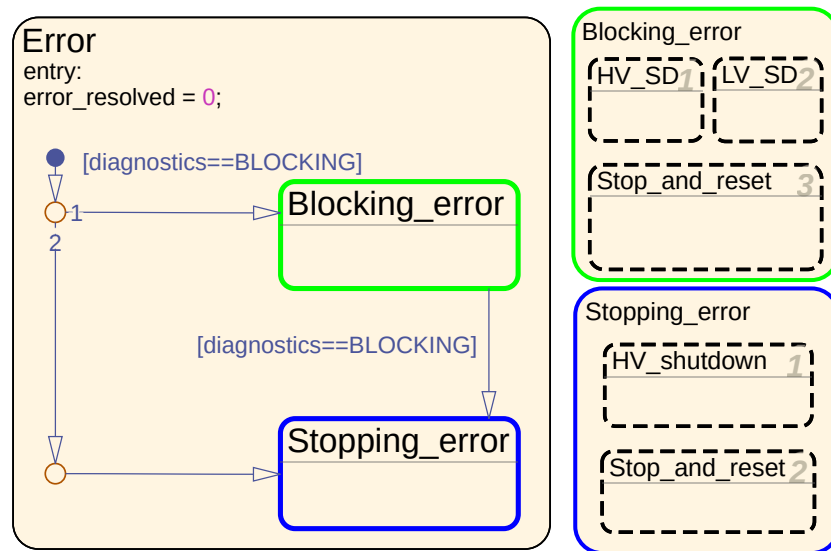


Figure 3.15: State Error

■ State Blocking_error

When a blocking error occurs, SBW, BBW and motors must be shut down, BMS contactor must be opened. That provides state Blocking_error which may be seen in Figure 3.16.

It contains three parallel substates:

1. **HV_SD** - switches motors OFF and opens BMS contactor.
2. **LV_SD** - switches SBW and BBW systems OFF.

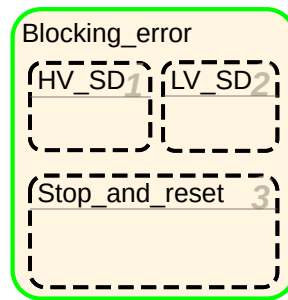


Figure 3.16: State Blocking_error

3. **Stop_and_reset** - waits until BMS contactor is opened, SBW and BBW systems are switched OFF, the car is stopped (slowing down is performed by the human operator pressing brake pedal - back-up mechanical brakes), and parking brake is engaged. Then state may be changed to Start on pressing Reset button if no charger is connected, SBW and BBW systems are OFF, BMS contactor is open and parking brake is engaged.

Substate HV_SD may be seen in Figure 3.17. Transition conditions and actions were removed from the figure and are the following:

1. **transition:** [entry action] {count = 0; diagnose_motors=OFF; diagnose_BMS=OFF; diagnose_lv=OFF; diagnose_comm=OFF;}
comment: Variable count is for determining how many times a request has been sent. Switch OFF diagnosing of all systems, error can not be more severe.
2. **transition:** [all_motors_mainsState(IN)~=ALARMED]
comment: At least one of the motors is not OFF. Go switch it OFF.
3. **transition:** [all_motors_mainsState(IN)==ALARMED] {count=0;}
comment: All motors have been switched OFF.
4. **transition:** after(PERIOD_MOT_OFF_MS, msec)[count==N_TRIES] {count=0;}
comment: Switching motors OFF has been unsuccessful for N_TRIES times. Go open BMS contactor.
5. **transition:** after(PERIOD_MOT_OFF_MS, msec)
comment: Try again to switch motors OFF.
6. **transition:** after(PERIOD_OPEN_CONT_MS, msec) [count==N_TRIES && all_motors_mainsState(IN)~=ALARMED] {count=0;}
comment: Switching motors OFF had been unsuccessful for N_TRIES

times, opening BMS contactor was unsuccessful for N_TRIES times, so try again to switch motors OFF.

7. **transition:** [BMS contactorStatus==CONT_CLOSED]
comment: Go open BMS contactor.
8. **transition:** [BMS contactorStatus==CONT_OPEN]
comment: BMS contactor has been successfully opened.
9. **transition:** after(PERIOD_OPEN_CONT_MS, msec)
comment: Try again to open BMS contactor.
10. **transition:** [BMS contactorStatus==CONT_OPEN] {count=0;}
comment: BMS contactor is open, go close it.
11. **transition:** [BMS contactorStatus==CONT_CLOSED] && all_motors_mainsState(IN)~=ALARMED {count=0;}
comment: Motors are not OFF, go turn them OFF.
12. **transition:** [exit action] {count = 0;}
comment: Variable count is for determining how many times a request has been sent.

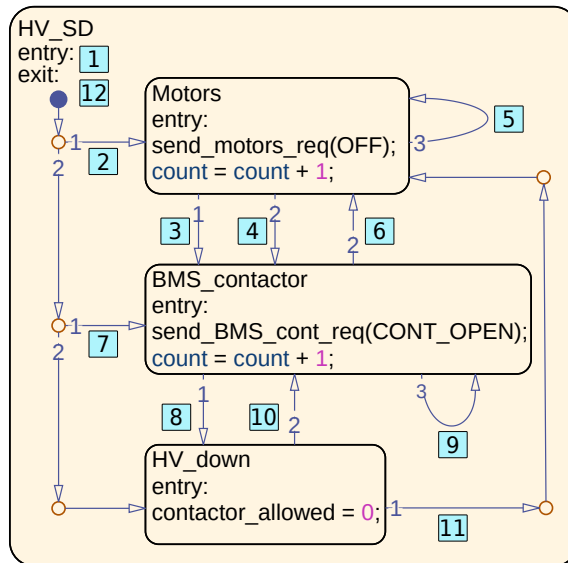


Figure 3.17: State HV_SD

Substate LV_SD may be seen in Figure 3.18. Transition conditions and actions were removed from the figure and are the following:

1. **transition:** [all_LV_toGoOFF_OFF(IN) == 0]
comment: There is a LV system that needs to be shutdown.

2. **transition:** $[all_LV_toGoOFF_OFF(IN) == 1]$
comment: All LV systems supposed to be switched OFF have been switched OFF.
3. **transition:** $after(PERIOD_LV_OFF_MS, msec)$
comment: Send request to shut down LV systems again.
4. **transition:** $[all_LV_toGoOFF_OFF(IN) == 0]$
comment: Some of the LV systems supposed to be OFF has been switched ON, go switch it OFF.

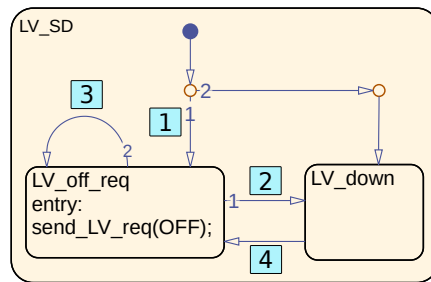


Figure 3.18: State LV_SD

Substate Stop_and_reset may be seen in Figure 3.19. Transition conditions and actions were removed from the figure and are the following:

1. **transition:** $[BMS\ contactorStatus == CONT_OPEN \ \&\&\ all_LV_toGoOFF_OFF(IN) == 1]$
comment: BMS contactor is open, SBW and BBW is OFF, go wait until the car is standing and parking brake is engaged.
2. **transition:** $[IMU\ speed < STOPPED_THRESHOLD \ \&\&\ HMI\ parkingBrakeStatus == 1]$
comment: FSVDVP is standing and parking brake is engaged.
3. **transition:** $[BMS\ chargingInterlock == 1]$
comment: A charger is connected, wait until it is disconnected.
4. **transition:** $[HMI\ parkingBrakeStatus == 0]$
comment: The parking brake has been disengaged.
5. **transition:** $[BMS\ chargingInterlock == 0]$
comment: The charger has been disconnected.
6. **transition:** $[BMS\ contactorStatus == CONT_CLOSED \ ||\ all_LV_toGoOFF_OFF(IN) == 0]$
comment: BMS contactor has been closed or SBW or BBW has been switched ON, go wait until it is resolved by states HV_SD and LV_SD.

- 7. **transition:** [HMI parkingBrakeStatus==0]
comment: The parking brake has been disengaged.
- 8. **transition:** [BMS chargingInterlock==1]
comment: A charger has been connected.
- 9. **condition:** [HMI Reset==1]
comment: SBW, BBW and motors are OFF, BMS contactor is open, parking brake is engaged, no charger is connected and Reset button has been pressed. Therefore perform reset - change state to Start.

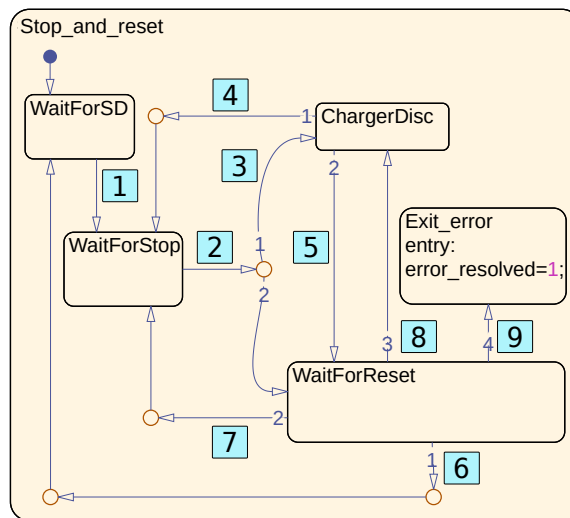


Figure 3.19: State Stop_and_reset

State Stopping_error

When a stopping error occurs, motors must be shut down and BMS contactor must be opened. That provides state Stopping_error. It may be seen in Figure 3.20.

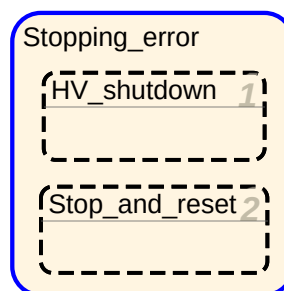


Figure 3.20: State Stopping_error

It contains two parallel substates:

1. **HV_shutdown** - switches ON safestop mode to slow down FSVDVP, after the vehicle is standing, switches motors OFF (and therefore safestop mode as well) and opens BMS contactor.
2. **Stop_and_reset** - waits until BMS contactor is opened, car is stopped (slowing down is performed by motors safestop mode and human operator pressing brake pedal - BBW system) and parking brake is engaged. Then state may be changed to Start on pressing Reset button if no charger is connected, BMS contactor is open and parking brake is engaged.

Substate HV_shutdown may be seen in Figure 3.21. Transition conditions and actions were removed from the figure and are the following:

1. **transition:** [entry action] {count = 0;}
comment: Variable count is for determining how many times a request has been sent.
2. **transition:** [all_motors_mainsState(IN)~=ALARMED]
comment: At least one motor is not OFF.
3. **transition:** [IMU speed > STOPPED_THRESHOLD
&& all_motors_mainsState(IN)==PWR_RDY]
comment: FSVDVP is moving and safestop mode may be switched ON on all motors.
4. **transition:** [all_safestop_ON(IN)==1] {count=0;}
comment: Safestop mode has been successfully switched ON on all motors.
5. **transition:** after(PERIOD_SFSP_ON_MS, msec)[count==N_TRIES]
{count=0;}
comment: Failed to turn ON safestop mode until PERIOD_SFSP_ON_MS timeout for N_TRIES. Go switch motors OFF.
6. **transition:** after(PERIOD_SFSP_ON_MS, msec)
comment: Try again to turn safestop mode ON.
7. **transition:** [IMU speed < STOPPED_THRESHOLD]
comment: FSVDVP has been stopped, switch motors OFF.
8. **transition:** [all_motors_mainsState(IN)==ALARMED] {count=0;}
comment: All motors have been switched OFF.

9. **transition:** after(PERIOD_MOT_OFF_MS, msec)[count==N_TRIES]
{count=0;}
comment: Failed to switch motors OFF until PERIOD_MOT_OFF_MS timeout for N_TRIES times.
10. **transition:** after(PERIOD_MOT_OFF_MS, msec)
comment: Try again to switch motors OFF.
11. **transition:** [BMS contactorStatus == CONT_OPEN]
comment: BMS contactor has been successfully opened.
12. **transition:** after(PERIOD_OPEN_CONT_MS, msec)[count==N_TRIES
&& all_motors_mainsState(IN)~=ALARMED] {count=0;}
comment: Switching motors OFF had been unsuccessful for N_TRIES times, opening BMS contactor was unsuccessful for N_TRIES times, so try again to switch motors OFF.
13. **transition:** after(PERIOD_OPEN_CONT_MS, msec)
comment: Try again to open BMS contactor.
14. **transition:** [BMS contactorStatus == CONT_CLOSED]
comment: BMS contactor is closed, go open it.
15. **condition:** [BMS contactorStatus==CONT_CLOSED
&& all_motors_mainsState(IN)~=ALARMED] {count=0;}
comment: BMS contactor must have been closed and at least one motor switched ON, go turn motors OFF again.
16. **transition:** [BMS contactorStatus==CONT_CLOSED] {count=0;}
comment: BMS contactor has been closed, go open it again.
17. **Transition:** [exit action] {count=0;}
comment: Variable count is for determining how many times a request has been sent.

Substate Stop_and_reset of Stopping_error is same as the one of Blocking_error (see Figure 3.19) except for transitions no. 1 and 6, those are the following:

- **transition no. 1:** [BMS contactorStatus == CONT_OPEN]
comment: BMS contactor is open, go wait until FSVDVP is stopped and parking brake is engaged.
- **transition no. 6:** [BMS contactorStatus == CONT_CLOSED]
comment: BMS contactor has been closed, wait until it is opened again.

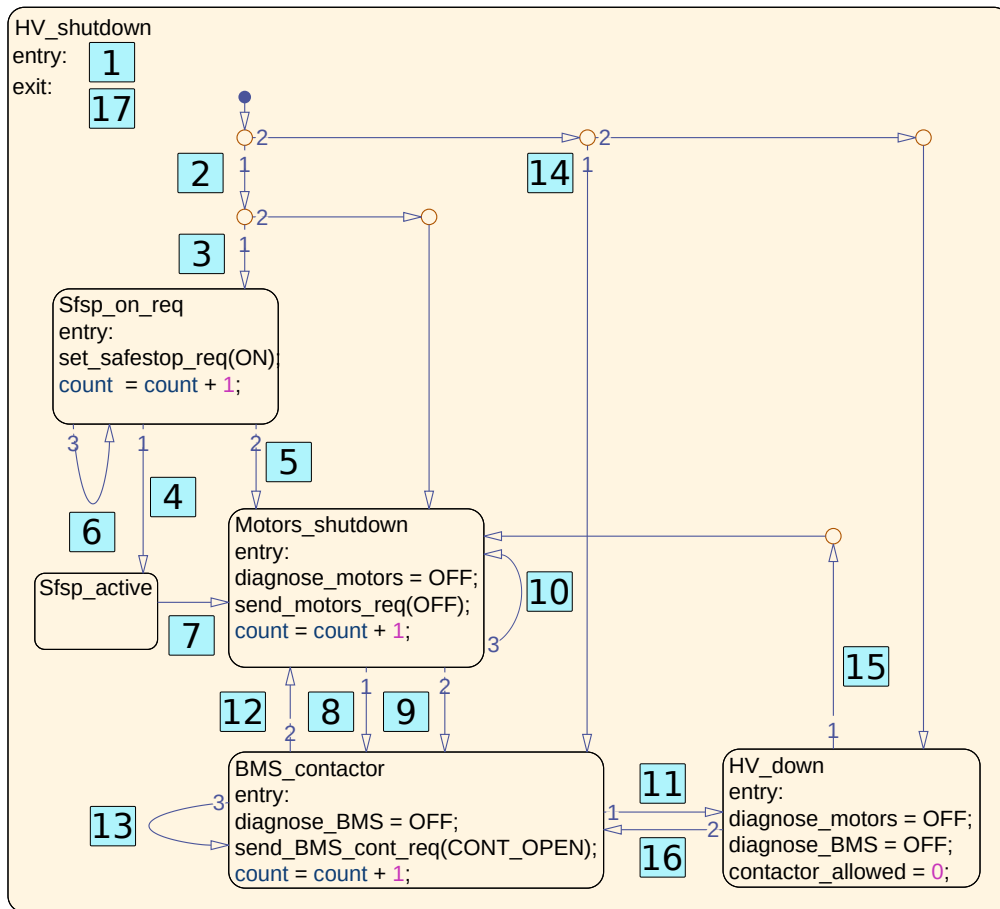


Figure 3.21: State HV_shutdown



Chapter 4

Testing the EV Manager

According to [20], there is need to test the developed EVM to reveal possible bugs and fix them. Also EVM design must be validated to be sure it behaves accordingly with the system requirements described in Chapter 3.

Matlab & Simulink Test Harness was used for software testing and validation of the implementation. Tests were also automated with the help of Matlab & Simulink Test Manager.

In this chapter, EVM testing process will be described. Designed test series will be presented, and an example will be explained in detail.



4.1 Test Harness

According to [11], a test harness is a test-specific simulation environment where models may be isolated for unit testing. Inputs may be easily designed with many tools. I used Signal Editor Block for such a case, for it provides a user-friendly graphical interface for creating and analyzing input signals. The user interface of Signal Editor Block may be seen in Figure 4.1.

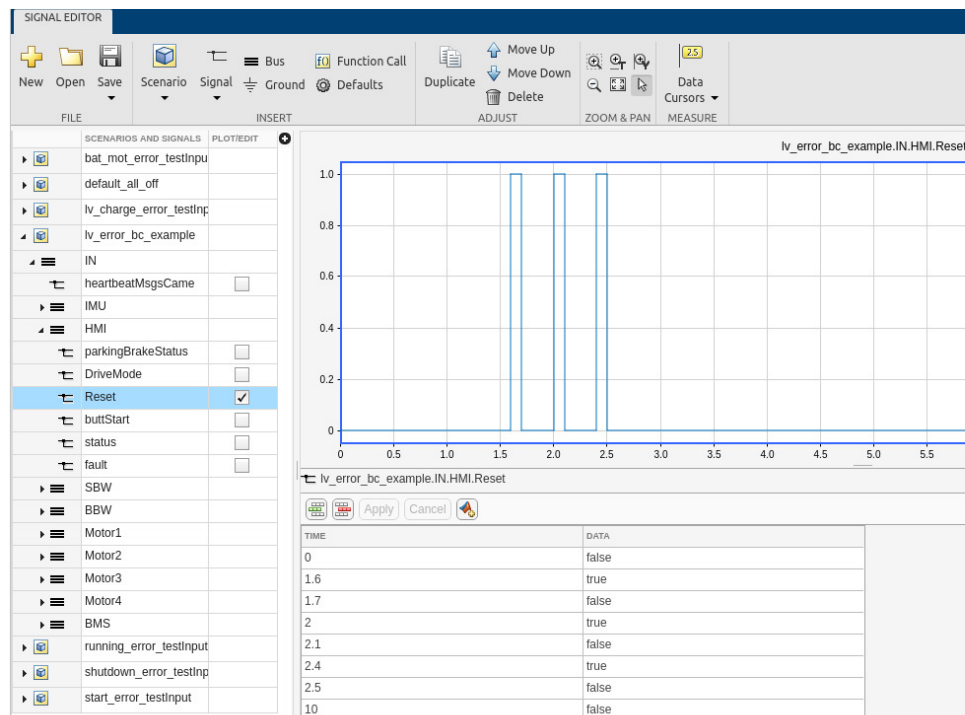


Figure 4.1: User interface of Signal Editor Block

4.2 Testing Process

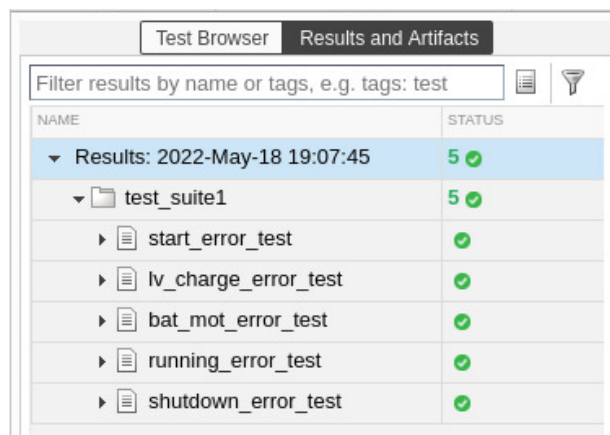
I designed five test series, each covering a part of EVM design, to validate the overall functionality of EVM specified in Chapter 3:

1. **Start_error_test** - for verifying state is changed from Start to Error when any error associated with state Start occurs (described in Subsection 3.6.1), and state is changed from Error to Start when all conditions for system reset are fulfilled (see Subsection 3.6.8).
2. **Lv_charge_error_test** - for verifying state is changed from Start to LV when communication is established, state is changed from LV to Error when any error relating to state LV occurs (see Subsection 3.6.2), state is changed from LV to Charging when all LV systems are ON and a charger is connected, state is changed from Charging to Error when any error relating to state Charging occurs (see Subsection 3.6.7) and state is changed from Charging to LV when the charger is disconnected. Also, for verifying state is not changed from Error to Start when Reset button is pressed and SBW or BBW is ON.
3. **Bat_mot_error_test** - for verifying state is changed from LV to Battery when LV systems are ON and Start button is pressed, state is

changed from Battery to Motors after BMS contactor is closed, state is changed from Battery to Error when any error associated with state Battery occurs (see Subsection 3.6.3), state is changed from Motors to Error when any error relating to state Motors occurs (see Subsection 3.6.4). Also, for verifying state is not changed from Error to Start when Reset button is pressed, and BMS contactor is closed.

4. **Running_error_test** - for verifying state is changed from Motors to CarRunning when all motors are successfully started, state is changed from CarRunning to Error when any error relating to state CarRunning occurs (see Subsection 3.6.5). Also, for verifying requests to switch mode are followed only when FSVDVP is standing, switching from Drive mode to Reverse mode and conversely might be performed only through switching to Neutral mode first. Also, for verifying that the mode is switched from normal Drive mode to limited Drive mode when diagnostics is LIMITING or variable limited_mode equals to 1, and switching back is not performed when diagnostics equals to NORMAL but Neutral mode is not switched to first.
5. **Shutdown_error_test** - for verifying state is changed from CarRunning to HV_SD when in Neutral mode, FSVDVP is standing and Start button is pressed, and it is not changed on pressing Start button when any of those conditions is not fulfilled. Also, for verifying state is changed from HV_SD to Error when any error associated with state HV_SD occurs (see Subsection 3.6.6) and state is not changed from Error to Start when motors are ON and BMS contactor is closed.

Those test series were also automated with the help of Simulink Test Manager. How it looks in Test Manager when all those tests are passed may be seen in Figure 4.2.



The screenshot shows the 'Test Browser' window in Simulink Test Manager, displaying the 'Results and Artifacts' for a test suite. The interface includes a search filter and a table of test results.

NAME	STATUS
Results: 2022-May-18 19:07:45	5 ✓
test_suite1	5 ✓
start_error_test	✓
lv_charge_error_test	✓
bat_mot_error_test	✓
running_error_test	✓
shutdown_error_test	✓

Figure 4.2: Simulink Test Manager - automated tests results

A part of `Lv_charge_error_test` will be described thoroughly here. Input signals of LV systems and signal `heartbeatMsgsCame` may be seen in Figure 4.3, HMI reset signal may be seen in Figure 4.4, and state of `MainStateMachine` may be seen in Figure 4.5.

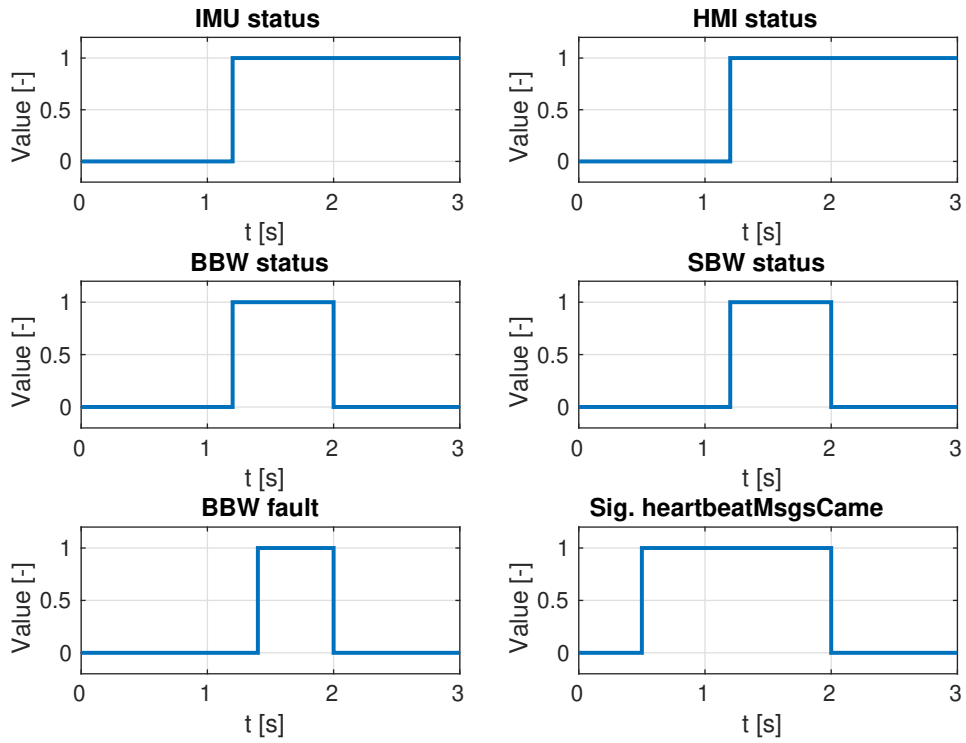


Figure 4.3: A part of `Lv_charge_error_test` - various input signals

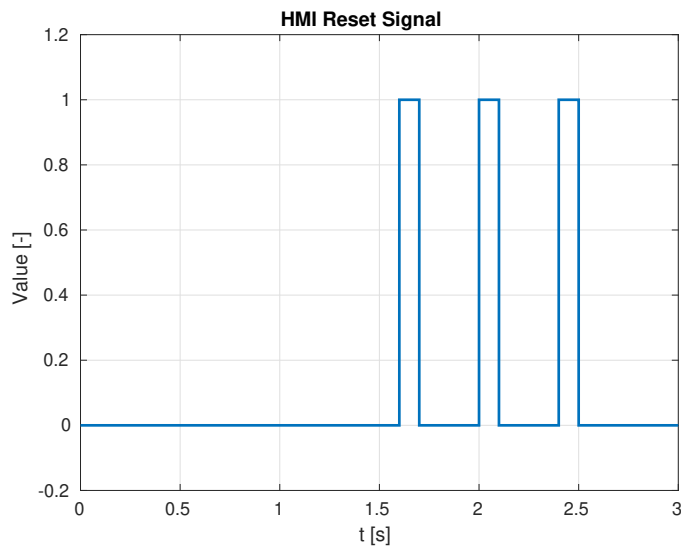


Figure 4.4: A part of `Lv_charge_error_test` - HMI Reset signal

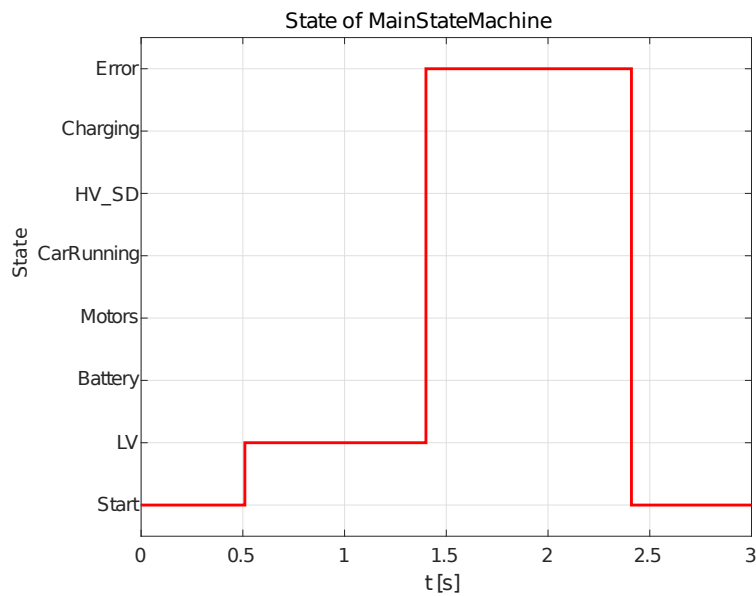


Figure 4.5: A part of Lv_charge_error_test - state of MainStateMachine

Communication is established at 0.5 s, therefore state is changed from Start to LV in the next simulation step. IMU, HMI, BBW and SBW are switched ON. BBW reports an error at 1.4 s, therefore state is changed to Error. Pressing Reset button has no effect until BBW and SBW are switched OFF - that happens at 2 s. At 2.4 s, Reset button is pressed and BBW and SBW are OFF, therefore state is changed to Start where EVM waits for communication to be established until TO_BROADCAST_MS timeout, which was defined as 2 s for the test.

The designed test series validate the developed EVM. It provides the full EV functionality and also deterministic behavior specified in Chapter 3. Nevertheless, testing process never ends. New test series will have to be made when EVM design is modified.



Chapter 5

Conclusion

Within this thesis, a Matlab & Simulink based framework for electric vehicle high-level functionality management system was developed, implemented, and tested.

Model-based design approach was chosen for the development because it provides an easy option to simulate the behavior of the developed system without any extra work and combines overall concept design and implementation. EV manager systems were reviewed in Chapter 1. In Chapter 3, system and test requirements were defined, and implementation and the overall concept of developed EVM were presented. The testing and validation process of the implementation was described in Chapter 4.

All of the objectives were met in this thesis, yet the current design of EVM is not the final one since FSVDVP design will be updated many times in the future. Even though it has to be altered when new systems are added to FSVDVP, it provides a solid design that is easy to modify.

As a future work proposition, hardware-in-the-loop testing should be performed.



Appendix A

Bibliography

- [1] <https://www.porscheengineering.com/peg/en/services/engineeringservices/electronics/>. Accessed: 13. 5. 2022.
- [2] <https://www.valeo.com/en/interior-cocoon/>. Accessed: 13. 5. 2022.
- [3] <https://in-wheel.com/en/solutions-2/vehicle-electrification-and-prototyping/>. Accessed: 13. 5. 2022.
- [4] https://gitlab.fel.cvut.cz/jirakji1/bc_thesis_doc/-/tree/main/Communication.
- [5] Does an electric vehicle emit less than a petrol or diesel? <https://www.transportenvironment.org/discover/does-electric-vehicle-emit-less-petrol-or-diesel/>. Accessed: 4. 5. 2022.
- [6] Drive-by-wire is born. <https://sds.fel.cvut.cz/home>. Accessed: 29. 4. 2022.
- [7] Greenhouse effect. <https://www.nationalgeographic.org/encyclopedia/greenhouse-effect/#on-thin-ice>. Accessed: 4. 5. 2022.
- [8] How cars have become rolling computers. <https://www.theglobeandmail.com/globe-drive/how-cars-have-become-rolling-computers/article29008154/>. Accessed: 4. 5. 2022.
- [9] Research motivation. <https://sds.fel.cvut.cz/research>. Accessed: 29. 4. 2022.

- [10] Stavby aut a motorek. <http://www.ponda.cz/stavby-aut-a-motorek>. Accessed: 1. 5. 2022.
- [11] Test harnesses. <https://www.mathworks.com/help/sltest/test-harnesses.html>. Accessed: 11. 5. 2022.
- [12] EMUS. EMUS G1 Control Unit CAN Protocol. https://gitlab.fel.cvut.cz/jirakji1/bc_thesis_doc/-/blob/main/Communication/CAN_protocol_datasheets/EMUS-G1-BMS-CAN-Protocol-v2.0.12.pdf, 2021. Original document from EMUS.
- [13] EMUS. G1 Battery Management System - User Manual. <https://emusbms.com/wp-content/uploads/2020/12/G1-BMS-User-Manual-2.7.0-2.pdf>, 2021. Original document from EMUS.
- [14] T. Kelemenová, M. Kelemen, L. Miková, V. Maxim, E. Prada, T. Lipták, and F. Menda. Model based design and HIL simulations. *American Journal of Mechanical Engineering*, 1(7):276–281, 2013.
- [15] R. Li, C. Liu, and F. Luo. A design for automotive CAN bus monitoring system. *2008 IEEE Vehicle Power and Propulsion Conference*, pages 1–5, 2008.
- [16] K. Liu, K. Li, Q. Peng, and C. Zhang. A brief review on key technologies in the battery management system of electric vehicles. *Frontiers of Mechanical Engineering*, 14:47–64, 2019.
- [17] Net Gains Motors, Inc. HyPer 9HV IS. https://www.go-ev.com/PDFs/HyPer_9HV_Sales_Sheet.pdf, 2021. Original document from Net Gains Motors, Inc.
- [18] NEXCOM International Co., Ltd. VTC 7230. http://files.nexcom.com/Driver/VTC72xx_Series/User_Manual_VTC72xx-BK_170303.pdf, 2017. Original document from NEXCOM International Co., Ltd.
- [19] Panasonic. NCR18650B. <https://www.tme.eu/Document/3e0170a1e089819f286f7066e69035b4/NCR18650B.pdf>, 2021. Original document from Panasonic.
- [20] H. L. Ross. *Functional Safety for Road Vehicles*. Springer, 2016.
- [21] TAU. TAU Generic Slave Interactive Documentation for Asynchronous Motors. https://gitlab.fel.cvut.cz/jirakji1/bc_thesis_doc/-/blob/main/00_Doc/Data_sheets/EV-europe/TAU_ACGSL_HELP.chm, 2021. Original document from TAU.
- [22] T. Veselý. Brake-by-wire system development. Master’s thesis, Czech Technical University in Prague, Czechia, 2022.