technische universität dortmund

H&D International Group

Master Thesis

Usefulness of Model-Based Testing Within Simulink Platform

Filip Chocholak

23.03.2018

Supervisors:
Prof. Dr. Jian-Jia Chen
Dr. Stefan Schneider

Technische Universität Dortmund
Fakultät für Informatik
Lehrstuhl Informatik 12 (Eingebettete Systeme)
http://ls12-www.cs.tu-dortmund.de

Acknowledgements

The following Thesis was done in cooperation with H&D Services for Engineering and the Design Automation of Embedded Systems Group of the Department of Computer Science at TU Dortmund. I would like to thank my company supervisor Dr. Stefan Schneider for his support in developing the basic principles for this Thesis and his meaningful feedback. My special gratitude goes to my university supervisor Prof. Dr. Jian-Jia Chen for giving me freedom to explore the different topics in this Thesis, while guiding me with insightful comments and helping me to shape it in every aspect. I would also want to thank both MathWorks Inc. and PikeTec GmbH. by providing us the relevant tools.
Lastly, a special thanks goes to my family, friends and my girlfriend for supporting me during my time of writing this Thesis.

Abstract

The trends of drastic increase of software features in relation to popular electric and self-driving cars pushes the industry to think more vast. To think more software features in vehicles, require more attention, towards safety and high product quality error-free aspects. To ensure such manners, more effort regarding testing must be considered. To test as early as possible, starting from the component Level through the integration Level up to the system Level while performing Model-Based Development (MBD). One of the approaches that has gained attention in the context of the Model-Based Development is the Model-Based Testing (MBT).

In this Thesis, an overview of the usefulness of Model-Based Testing within a standard Model-Based Development platform Simulink for automotive applications will be introduced. Firstly, a detailed investigation regarding MBT approaches covering functional perspective will be performed by exploring two MBT tools: the Simulink Design Verifier from Mathworks Inc. and the TPT from PikeTec GmbH. The objectives are to look deeply inside of the MBT methods used for each tool by analyzing the modeling, execution and evaluation of the test cases from the different system levels as well as from the different requirement categories points of view. For these reasons, the categorization of requirements based on the concurrent level of the system as well as based on its characteristics have to be established. Moreover, based on these investigations, conclusions regarding the testing activities performing at various system level will be conducted. Secondly, the analysis based on the model complexity, scalability and applicability for each tool will be performed.

To fulfill these objectives a case study model was selected. The model introduced basic functionality for Power Window Control System in vehicles modeled using Simulink Functional Block Diagrams. Due to the simplicity of the case study model, the results are limited with respect to its applicability.

Contents

Nomenclature

Mathematical Notation	Meaning
F	Function F
K_I=(i_1,...,i_n)	Set of input channels from i_1 to i_n
K_O=(o_1,...,o_n)	Set of output channels from o_1 to o_n
t_{stop}	Termination Time
∞	Infinity
T	Non-empty set of Functions T
t	Time
$\wedge$	Boolean *AND*
$\neg$	Boolean *NEGATION*
N	Unit *Newton*
A	Unit *Amper*
M	Model
p	Property
k	Bound
s	State Variable
$I(s)$	Initial States
$T(s)$	Transition relation

Abbreviation	**Meaning**
MBD	Model-Based Development
MBT	Model-Based Testing
TPT	Time Partition Testing
SLDV	Simulink Design Verifier
SUT	System Under Test
MiL	Model-in-the-Loop
SiL	Software-in-the-Loop
PiL	Processor-in-the-Loop
HiL	Hardware-in-the-Loop
PWCS	Power Window Control System
SRE	Software Requirement Engineering
TPT-VM	TPT Virtual Machine
TSLT	Time Step List Testlet
TPTT	Time Partition Testing Testlet
TSLA	Test Step List Assessment
DED	Design Error Detection
BMC	Bounded Model Checking
SAT	Boolean Satisfiability Problem
CNF	Conjunctive Normal Form
SMCC	Structural Model Coverage Criteria
DC	Decision Coverage
CC	Condition Coverage
MCDC	Modified Condition/Decision Coverage

1 Introduction

1.1 Motivation

With increasing global competition, companies in automotive industry face the challenge of constantly developing new technological solutions with respect to electronic units. According to [7] electronics and software represent about 90 percent of all the current car innovations. The demand for more complex but reliable software solutions for vehicle electronics strikes new dimensions. A Model-Based Development (MBD) approach supported with the standard industrial MBD tool Simulink enables companies in precise graphical design, simulation and code generation for various automotive applications. In order to rapidly speed up a development process, the verification against the functional requirements is usually done after the automatic code generation [31]. Which although might be too late, considering that software covers approximately 20 percent of the total classical vehicle costs and approximately 50 percent of the total costs for hybrid vehicles [18], finding and fixing them during the later stage can become extremely costly.

According to [15], the costs of discovering defects during the implementation phase is approximately 5 times higher than that in the design phase. For these reasons, the better approach is to consider MBT activities during the modeling phase, the so called: "test as you go" phase. This approach enables companies to start testing as early as possible, starting from the component level through the integration level up to the system level while performing Model-Based Development, which in the end might have a positive impact on the time and costs savings as well as the general product quality improvement. However, a necessary question might arise: Is it worth to invest time and extra costs into early testing? According to [26] companies allocate more than 40 percent of their product development time to testing, which results in extreme time consumption and an overall increase in costs. In contrast, the results of MBT survey show that by using MBT methods generally will decrease the costs of testing by approximately 23 percent and the time duration by approximately 20 percent [25].

1.2 Goals of the Thesis

The general goal of this Thesis is to analyze the usefulness of the Model-Based Testing methods within the Simulink platform for automotive applications by covering the following objectives: The first objective is to perform the detailed analysis of the selected model in order to define and categorize all of the functional requirements into specific defined categories based on functionality as well as to categorize based on the different levels of the system: *component level*, *integration level* and *system level*. After the categorization of the functional requirements a detailed investigation regarding MBT approaches covering functional perspective will be covered by exploring two MBT tools: the Simulink Design Verifier from MathWorks Inc. and the TPT from PikeTec GmbH. The goals are to look deep inside of the MBT methods used for each tool by analyzing the modeling, execution and evaluation of the test cases from the different system levels as well as from the different requirement categories' point of view. In the last part the goal is to examine the applicability of the approaches for both tools and a possible comparison of the tools.

1.3 Road-map of the Thesis

The road-map of the Thesis is structured as follows: in the first Chapter a definition of MBT as well as a general overview of MBT Taxonomy is introduced. Furthermore, an overview of the MBT tools for Simulink all together with the limitations are also discussed in Chapter 1. To support the analysis, a Literature Review is conducted and summarized in Chapter 2. Chapter 3 provides a detailed description of the model used for the Case Study followed with requirements categorization based on its functionality and the level of the system. A detailed investigation of the first MBT tool for Simulink (the TPT tool) covering modeling, execution, evaluation and reporting is summarized in Chapter 4. A detailed investigation of the second MBT tool for the Simulink platform (the SLDV tool) from MathWorks Inc. focusing on Design Error Detection, Property Proving and Test Case Generation is described in Chapter 5. Finally, the conclusion of this analysis is summarized in Chapter 6.

2 Model-Based Testing

This chapter gives insights of Model-Based Testing (MBT) approaches in general. In the first part the definition of MBT used in this Thesis will be introduced. Then the generic dimensions for testing in the automotive sector will be deeply discussed followed by a brief overview of the MBT Taxonomy with respect to the Simulink platform. Based on the MBT Taxonomy, a detailed review of available MBT Tools used with relation to the Simulink platform will be presented.

2.1 Definition of Model-Based Testing

Generally, giving a single precise definition of MBT can become quite challenging. This section will present some of the existed definitions of MBT from the academic field. The authors of the MBT Taxonomy [49] define MBT as the way of automatically deriving concrete test cases from abstract formal models and their execution. Another definition introduced in [48] states that MBT is the automation of black-box test design. Another MBT approach explains that the test cases are derived from an environment model that describes an expected environment of a System Under Test (SUT) [11]. According to [53] the most generic definition states that test specifications are derived directly from both system requirements and a model that describes SUT from functional and non-functional aspects. Due to the fact that the scope of this Thesis is restricted only to functional aspects applied for applications in automotive industry, the most suitable definition of MBT is based on the definition from [14] reporting that MBT serves to describe all testing activities in context of MBD and is comprised of several different testing methods gathered from the model [27]. Moreover, selected MBT methods and tools are based on their applicability to the Simulink platform. The Simulink platform is directly connected to Model-Based Development, which is one of the most popular development processes in the automotive industry [14].

In terms of Simulink, the testing is usually done after an auto-generated code as represented in the following flow diagram shown in Figure 2.1.

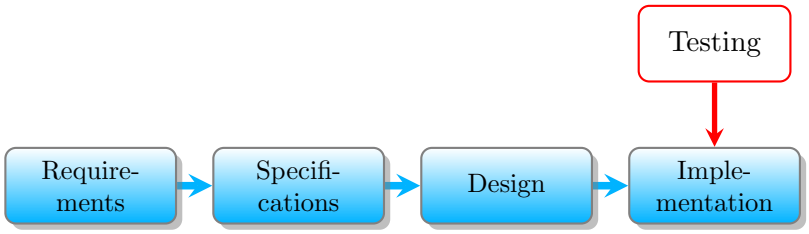

Figure 2.1: Diagram representing development flow associated with traditional testing

On the other hand, using MBT activities already during the modeling phase gives an additional possibility to eliminate design errors that might lead to faulty functional behavior. Such a flow is represented in the following diagram shown in Figure 2.2.

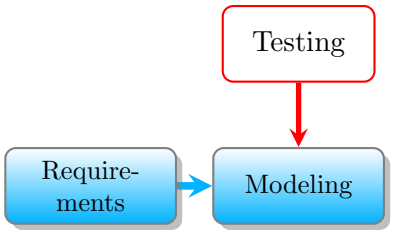

Figure 2.2: Diagram representing development flow associated with MBT

2.2 Generic Dimensions for Testing in the Automotive Sector

This section briefly explains generic dimensions for testing activities involved in the automotive sector and its relation to this Thesis. For this purpose a categorization of the different dimensions in the automotive sector according to [54] will be used. The exact definitions for each category are retrieved from various sources. These are:

1. **Test Goal** explains the general purpose of the testing. These perspectives are categorized as follows:
   - **Static Tests:** serve to detect the errors within a source code of the system requirements.
   - **Dynamic Tests:** aim through the execution to ensure the accuracy of its functionalities [13].
     a) *Structural Tests:* are also known as "white-box" tests, where the interior model behavior is known.

b) *Functional Tests:* are known as "black-box" tests. The internal behavior is not known and the objective is to focus on the outputs generated in response to selected inputs and execution conditions [51].

c) *Non-functional Tests:* are also black-box tests with an objective to focus on non-functional behavior (for example: performance).

2. **Test Abstraction:** defines the user-friendliness of the test cases.
   - *Abstract:* test specifications are readable and understandable when abstraction is used [54].
   - *Non-abstract* test specifications are with the lack of readability and understandability.

3. **Test Execution Platform:** defines various execution platforms used through the MBD.
   - *Model-in-the-Loop (MiL):* is the type of testing with the goal of verifying a model implementation from the structural and the behavioral perspectives [46].
   - *Software-in-the-Loop (SiL):* this type of testing is done after automatic code generation in a closed or open loop.
   - *Processor-in-the-Loop (PiL):* is the next stage of testing, where the goal is to perform testing on the target processor.
   - *Hardware-in-the-Loop (HiL):* testing is performed on the actual target hardware. With HiL simulation the physical part of a system - hardware is replaced by a simulation in order to reduce an overall risk of personal injuries, damaging equipment and delays [24].

4. **Test Reactiveness:** emerges when test cases are dependent on the system behavior [54].
   - *Reactive:* means that dependency is based on the interaction between the system and the environment.
   - *Non-reactive:* there is no interaction between the system and the environment considered.

5. **Test Scope:** considers the categorization based on different levels of the system. Due to the various definitions for the levels of the system, the following definitions were separately defined for the purpose of this Thesis.
   - *Component/Unit level:* is considered to be the smallest possible unit. In the Simulink platform it would be the simplest functional block diagram.
   - *Integration level:* is considered whenever two or more components interact with each other. As an example: several functional block diagrams in Simulink are merged into one Atomic unit.
   - *System level:* is considered the complete system, as the unit.

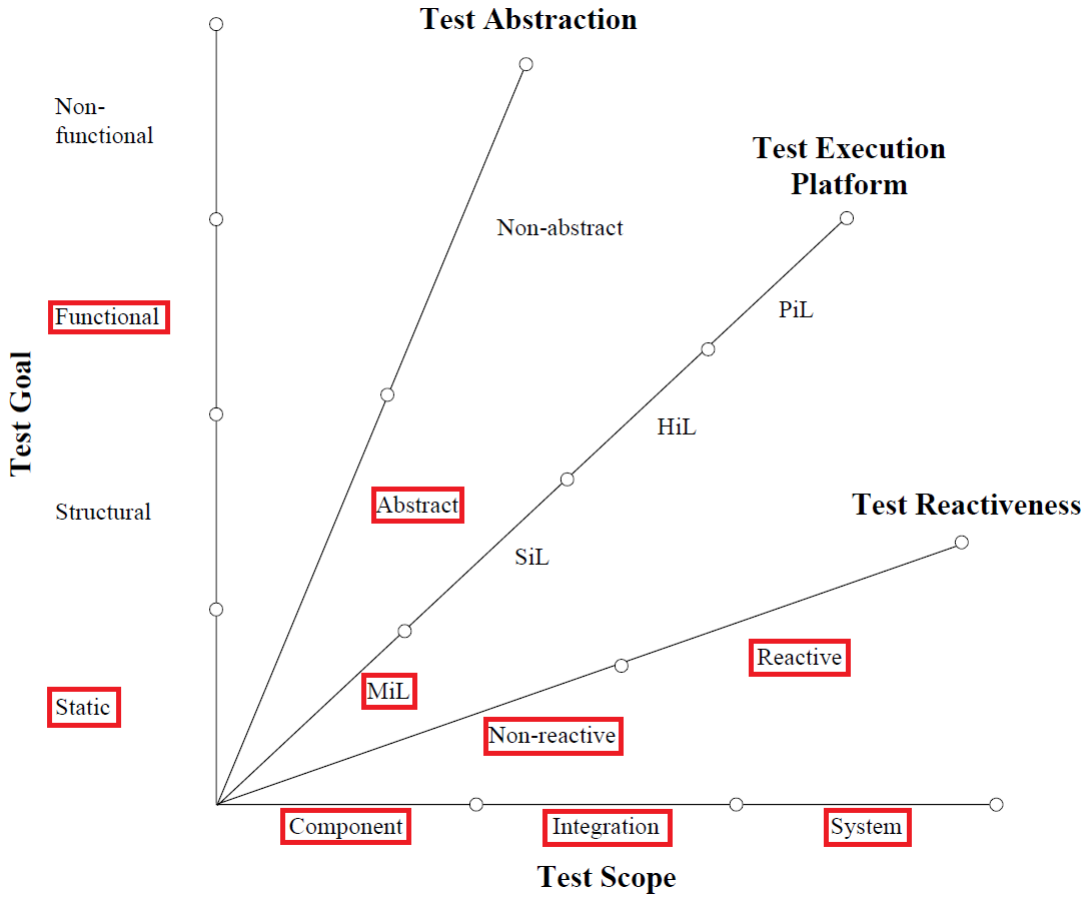

Figure 2.3: Graphical representation of the Test Dimensions

Figure 2.3 represents graphically the test dimensions and the relationship between them. The ones that are inside of red frames are the ones considered to be relevant in this Thesis.

2.3 Brief Overview of the MBT Taxonomy

The first version of the MBT taxonomy was introduced in [49], [48], [26] and [11] where the authors defined three general divisions: **Classes** followed by their various possible **Categories** and **Options**, which give various degrees of freedom to different categories. The first division consists of three classes: *Model*, *Test Generation* and *Test Execution*. This general structure of the MBT taxonomy was in [54], [10] extended with one more class called: *Test Evaluation*. For the purposes of this Thesis, an additional extension needed to be performed by excluding the entire *Model* class, due to the reason that we are strictly focusing only on one specific model category represented in Simulink/StateFlow. Moreover, we have introduced a new category in the *Test Execution* class called *Test Scope*, it consists of three options: *component*, *integration* and *system*. Note that the red frames point out the options that cover the selected MBT tools in this Thesis. The modified version is represented graphically in the diagram (Figure 2.4) below followed by a detailed description for each section. An overview of the selected available MBT tools that are suitable for the Simulink platform are presented in Section 2.4. The selection criteria of the MBT Tools were strictly based on its applicability to perform functional testing at the different levels of the system, mainly at the component level. Another criterion was an industrial use in automotive industry. For these reasons, the following two MBT tools fitting our selection criteria were chosen: the TPT tool from PikeTec GmbH and the Simulink Design Verifier from MathWorks Inc.

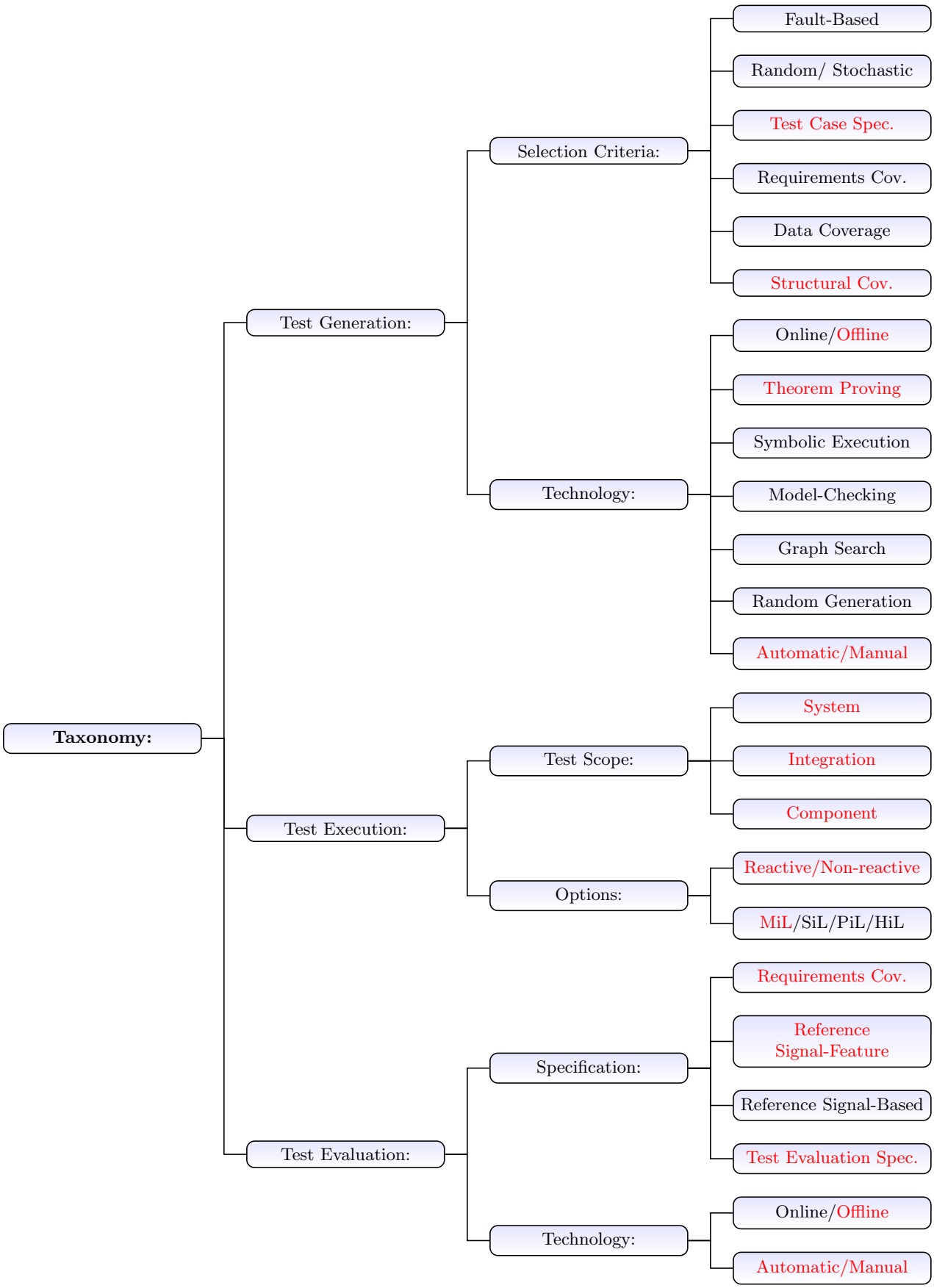

Figure 2.4: Diagram representing the modified MBT Taxonomy

1. **Test Generation:**
   - **Selection Criteria:**
     a) *Fault-Based:* the criterion that uses a fault model directly to hypothesize potential faults that might occur in System Under Test (SUT) [16]. According to [48], this criterion applies directly into the SUT model, because the aim is to find faults in the SUT.
     b) *Random/Stochastic:* the criterion that is typically applied with relation to environment models. According to [54] a typical approach of how to specify the expected SUT is with the use of Markov Chains.
     c) *Test Case Specification:* is defined in some formal notation. For example: the temporal logic formulas or the constraints.
     d) *Requirements Coverage:* is applied in cases of direct association of some elements of model with the informal requirements of the SUT [10].
     e) *Data Coverage:* is a criterion that specifies the intended test coverage for input data [26].
     f) *Structural Coverage:* explores the structure of the model in order to select test cases [43].
   - **Technology:**
     a) *Online/Offline:* in case of *online*, the test cases are generated during execution. When *offline*, the test cases are generated before the actual run of the execution.
     b) *Theorem Proving:*is the method of varying the model design against the functional requirements by specifying a property.
     c) *Symbolic Execution:* explores different execution paths of a program based on the path condition; each path condition encodes conditions over the variable for the corresponding test case [39].
     d) *Model-Checking:* generally serves to automatically verify correctness of properties of safety-critical reactive systems [17].
     e) *Graph Search:* applies the graph theory that allows us to use the behavioral information stored in modes to generate new test cases [44].
     f) *Random Generation:* uses the randomization to explore the entire space of a system [53].
     g) *Automatic/Manual:* manual test cases are generated by hand through the modeling languages like automatons.

2. **Test Execution:** see Section 2.2.
   - **Test Scope:**
     a) *System*
     b) *Integration*
     c) *Component*
   - **Options:**
     a) *Reactive/Non-reactive*
     b) *MiL/SiL/PiL/HiL*
3. **Test Evaluation:**
   - **Specification:**
     a) *Requirements Coverage:* aims to cover all informal SUT requirements with test evaluation scenario [12].
     b) *Reference Signal-Feature Based:* based on features of the reference signal, assesses the SUT behavior by classifying the SUT outcomes into features and comparing the outcome with previously specified reference values for those features [53].
     c) *Reference Signal-Based:* based on reference signal, assesses the SUT behavior comparing the SUT outcomes with the previously specified references [53].
     d) *Test Evaluation Specification:* uses the specification of expected SUT responses.
   - **Technology:**
     a) *Online/Offline:* online in case of test evaluation during text execution.
     b) *Automatic/Manual*

2.4 Overview of Available MBT Tools Suitable for Simulink

This section will provide an overview of available MBT tools used primarily in relation to Simulink presented in the industry and academia based on the MBT Taxonomy defined in the section 2.3. The full summary of the different categorizations is represented in the following Table 2.1.

	Text Generation:		**Text Execution:**		**Text Evaluation:**	
Selected MBT Tools	*Test Selection Criteria:*	*Technology:*	*Test Scope:*	*Options:*	*Specifications:*	*Technology:*
BTC Embedded Tester	Data Coverage, Requirement Coverage	Automatic Generation	System, Integration	MiL, SiL, PiL, Non-reactive, Automated	Requirements Coverage	Automatic Specification, Offline Evaluation
MES Test Manager	Data Coverage, Requirement Coverage, Test Case Specification	Manual Generation	System	MiL, SiL, PiL, Non-reactive, Automated	Reference Signal-based	Automatic Specification, Offline Evaluation
Reactis Tester	Data Coverage, Structural Model Coverage	Automatic Generation, Model-checking	System, Integration, Component	MiL, SiL, PiL, HiL, Non-reactive, Automated	Test Evaluation Specifications	Automatic Specification, Offline Evaluation
Reactis Validator	Requirement Coverage	Automatic Generation	System	MiL, SiL, Non-reactive, Automated	Test Evaluation Specifications	Manual Specification, Offline Evaluation
TPT	Requirement Coverage, Data Coverage, Test Case Specification	Manual Generation	System, Integration, Component	MiL, SiL, piL, HiL, Reactive, Automated	Reference Signal-based	Manual Specification, Offline Evaluation, Online Evaluation
MiLEST	Requirement Coverage, Data Coverage, Test Case Specification	Manual Generation	Integration, Component	MiL, Reactive, Automated	Reference Signal-feature-based, Requirement Coverage, Test Evaluation Specifications	Automatic Specification, Manual Specification, Online Evaluation
Simulink Test	Simulation-based, Requirement Coverage	Automatic Generation	System	SiL, HiL, PiL, Non-reactive, Automated	Test Evaluation Specifications	Manual Specification, Online Evaluation

T-VEC Test Vector Generation System	Structural Model Coverage, Data Coverage, Requirement Coverage	Automatic Generation	System	Mil, SiL, Non-reactive, Automated	Test Evaluation Specifications	Automatic Specification
ETAS RT2	Test Case Specification	Manual Generation, Automatic Generation	Integration	Mil, SiL, Reactive, Automated	Test Evaluation Specifications, Reference Signal-based	Manual Specification, Offline Specification
Mx-Suite	Structural Model Coverage, Data Coverage	Manual Generation, Automatic Generation	System	Mil, SiL, HiL, Non-reactive, Automated	Test Evaluation Specifications	Automatic Specification

Table 2.1: Selected MBT Tools overview according to the MBT Taxonomy

3 Literature Review: Trends in MBT in relation to the Simulink Platform

In general, Literature Review serves to identify, analyze and interpret all available evidence related to a specific research question [23]. Generally, the main reasons to perform literature reviews, explained in [9] are:

- "To summarize the existing evidence concerning a treatment or technology."
- "To identify any gaps in current research in order to suggest areas for further investigation."
- "To provide a framework/background in order to appropriately position new research activities."

The literature review in this Thesis aims to summarize the existing evidence of the trends in MBT in relation with the Simulink platform. The goal is to analyze an interest to this topic among researchers. The review focuses on the trends in automotive domain within the time span of 2007 - 2017.

3.1 Search Strategy:

In order to achieve relevant results, the search strategy consists of five elements. The following Figure 3.1 illustrates the search strategy conducted.

Figure 3.1: Graphical representation of the search strategy

3.1.1 Search Queries

For our purposes only two search queries were necessary. Both queries contain two search terms, which are divided with help of the Boolean function 'AND'. The first query is the core query for this Thesis. The second query serves to confirm a sufficiency of the search results for the first query. Both are defined as:

1. "*Model-based Testing* ***AND*** *Simulink*"
2. "*MBT* ***AND*** *Simulink platform*"

3.1.2 Search Sources

In this section, the list of eight digital libraries used as a primary search source is presented. Each of the search engines have been designed for the purpose of the software engineering research and therefore are relevant in the case. Most of the listed sources were randomly selected, except IEEE Xplore, ACM Portal and ScienceDirect. These three digital libraries and search engines were considered to be essential for gathering relevant results from the field of software engineering.

- **ScienceDirect**
- **Springler Link**
- **IEEE Xplore Digital Library**
- **Web of Science**
- **ACM Digital Library**
- **Wiley Online Library**
- **SPIE Digital Library**
- **Scopus**

3.1.3 Search Inclusive/Exclusive Criteria

Applying inclusive criteria ensures that all necessary information will be included in the review and consequently, exclusive criteria serve to determine any circumstances that would make a study ineligible to be included in the review [4]. In our case, three inclusive/exclusive criteria were sufficient enough to conclude relevant observations. They are defined as:

- **All the papers must be related to automotive industry**
- **The language must be English**
- **Duplicated publications must be excluded**

3.1.4 Search Results

The following section shows the publications that were obtained by applying the search queries. It is necessary to mention that search results for the second query had no influence on an overall search result for the first query and for this reason, the result for the second query were excluded. Table 3.1 represents the search results obtained for the time period of 2007 - 2017 for selected search resources.

Search Sources:	**2007**	**2008**	**2009**	**2010**	**2011**	**2012**	**2013**	**2014**	**2015**	**2016**	**2017**
Science Direct	7	5	12	4	16	24	17	28	36	42	54
Springler Link	17	5	23	34	21	25	42	54	31	50	45
IEEE Xplore Digital Library	6	11	10	6	10	14	17	22	15	13	10
Web of Science	11	18	15	18	19	21	27	39	38	28	15
ACM Digital Library	8	8	8	11	11	22	20	20	17	18	17
Wiley Online Library	22	20	25	24	41	41	66	53	54	63	65
SPIE Digital Library	15	24	12	21	12	10	14	11	15	17	8
Scopus	13	10	13	17	25	24	26	27	26	19	16

Table 3.1: Search results represented in tabular form

3.1.5 Search Analysis

Figure 3.2 shows graphically that the number of Model-Based Testing publications in relation with the Simulink platform has been increasing radically and the interest among researchers has more than doubled in the past years. More specifically, in 2007 the total number of publications for all digital libraries was 86 and in 2016 it was 231, which has almost tripled during the period of nine years. Another observation shows that the trend during the years of 2014 - 2017 stayed constant with no profound derivations. Furthermore, a small decrease in the trend between 2014 and 2015 can be observed. This might be caused by the time effort of publishing new research papers or the new MBT tools development.

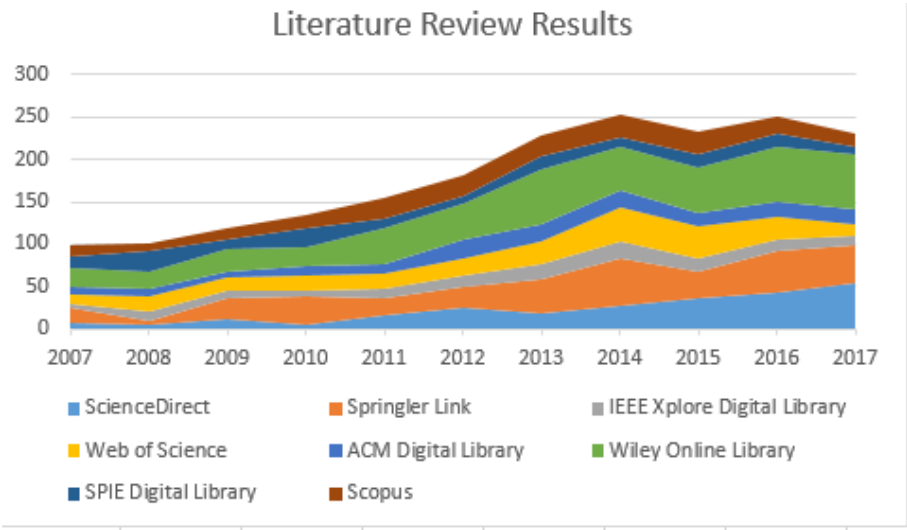

Figure 3.2: Graphical representation of the search results

4 Case Study: Model of PWCS

In this Section, a brief overview of the selected model for the Case Study will be presented by explaining the model features and the actual implementation. The second part contains a detailed requirements categorization.

4.1 Features of the PWCS

In order to fully achieve all the goals defined for this work a case study was established. As an objective of the case study, the Simulink model that models the functionality of the **Power Window Control System (PWCS)** (retrieved from [37]) was conducted. The PWCS controls the window position from either driver's or passenger's perspective with extended obstacle detection functionality. The next Figure 4.1 provides an illustrative representation for the system in reality. The implementation of the PWCS has the me-

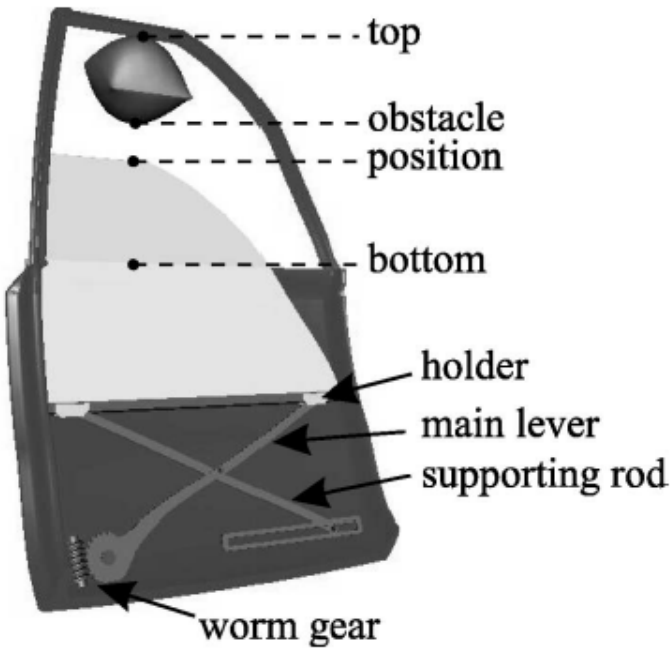

Figure 4.1: An illustrative representation of the PWCS, retrieved from [37]

chanical section that consists of four parts: *holder, main lever, supporting rod and worm gear* and have the function to ensure a correct lift mechanism for the window to be able to move either up or down.

A detailed illustration of the implementation of the mechanical section is shown in Figure 4.2.

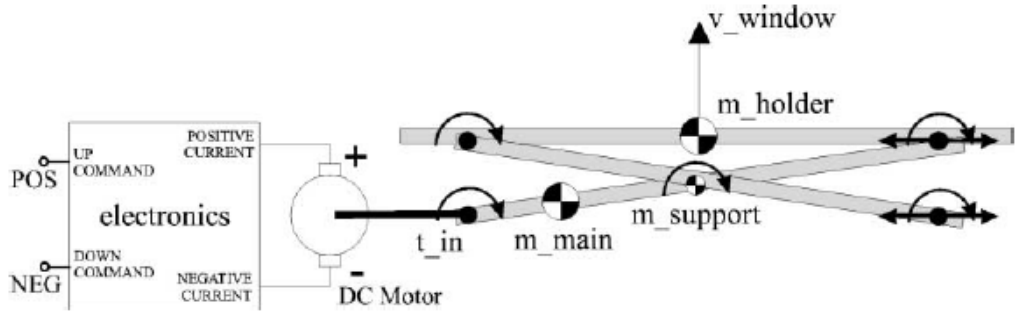

Figure 4.2: A detailed illustrative representation of the mechanical section of PWCS, retrieved from [37]

It might be observed that the mechanical system with all four mechanical parts (*holder, main lever, supporting rod and worm gear*) altogether are connected to *Electronics* block with *DC Motor*. The role is to serve as an intermediate power source to the lift mechanism [37]. The illustrative explanation of the *Electronics* block is shown in Figure 4.3.

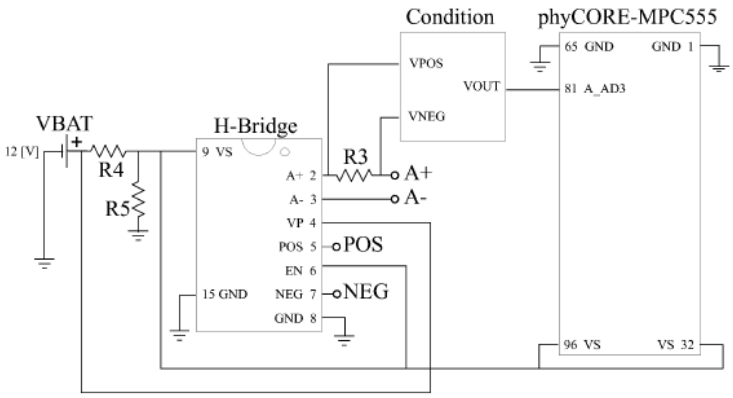

Figure 4.3: Detailed scheme of electronics block, retrieved from [37]

Before the output is sent to the AD converter, the commands up/down must be amplified by an H-Bridge [37] that enables a voltage to be applied in the opposite direction, which is used in relation with DC motor functionality [3]. The measurement process in the implementation resides on the phyCORE-MPC555, which is a highly functional Single Board Computer solution that is normally populated by a MPC555 PowerPC Microcontroller [40].

In the Simulink platform, these functionalities are modeled as followed in Figure 4.4.

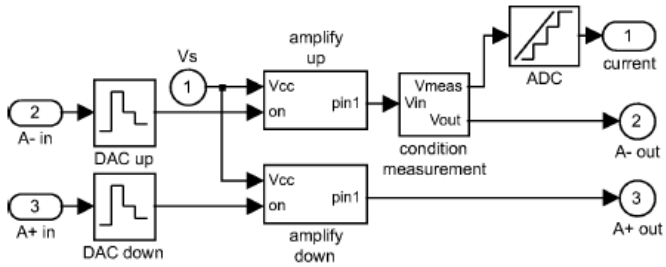

Figure 4.4: Simulink model of the electronics block, retrieved from [37]

Furthermore, the existing Simulink model [37] had defined four system requirements that have been further extended to the total of 6 system requirements listed in the Table 4.1. For example, in the automotive sector, specifically in the area of *Car Interior* and *Passenger Comfort*; there are around 70 features with an average amount of 50 pages per feature that defines its functionalities [50]. Therefore, the listed four requirement specifications for the original model were not sufficient to achieve all goals in this Thesis. For this reason, an additional number of requirement specifications had to be established. A complete list of the requirements specifications is explained in the Section 4.4.

Req. ID:	**Block Diagram Name:**	**Description:**
1.	*power_windows_system*	The window has to be fully opened within 4s.
2.	*power_windows_system*	The window has to be fully closed within 4s.
3.	*power_windows_system*	When an obstacle is present, the window should be lowered by approximately 10 cm.
4.	*power_windows_system*	The force to detect when an obstacle is present should be less than 100N (maximum current must be 1.7A).
5.	*power_windows_system*	If the up command is issued for at least 200ms and at most 1s, the window has to be automatically opened completely.
6.	*power_windows_system*	If the down command is issued for at least 200ms and at most 1s, the window has to be automatically closed completely.

Table 4.1: Table of system requirements for PWCS

4.2 PWCS Model Implementation

The full realization of the PWCS provided by the authors of [37] is demonstrated in Figure 4.5. It consists of different sub-systems that are responsible for different functionalities. Specifically functional block diagrams, namely: *driver_switch*, *passenger_switch*, *power_window_control_system*, *window_system* and *window_world* are concerned. The last sub-system *window_world* is not relevant for our case study and therefore is excluded. Note that each of the system is detailed, analyzed and the list of all functional specifications for each sub-system was conducted and presented in the next section 4.3.

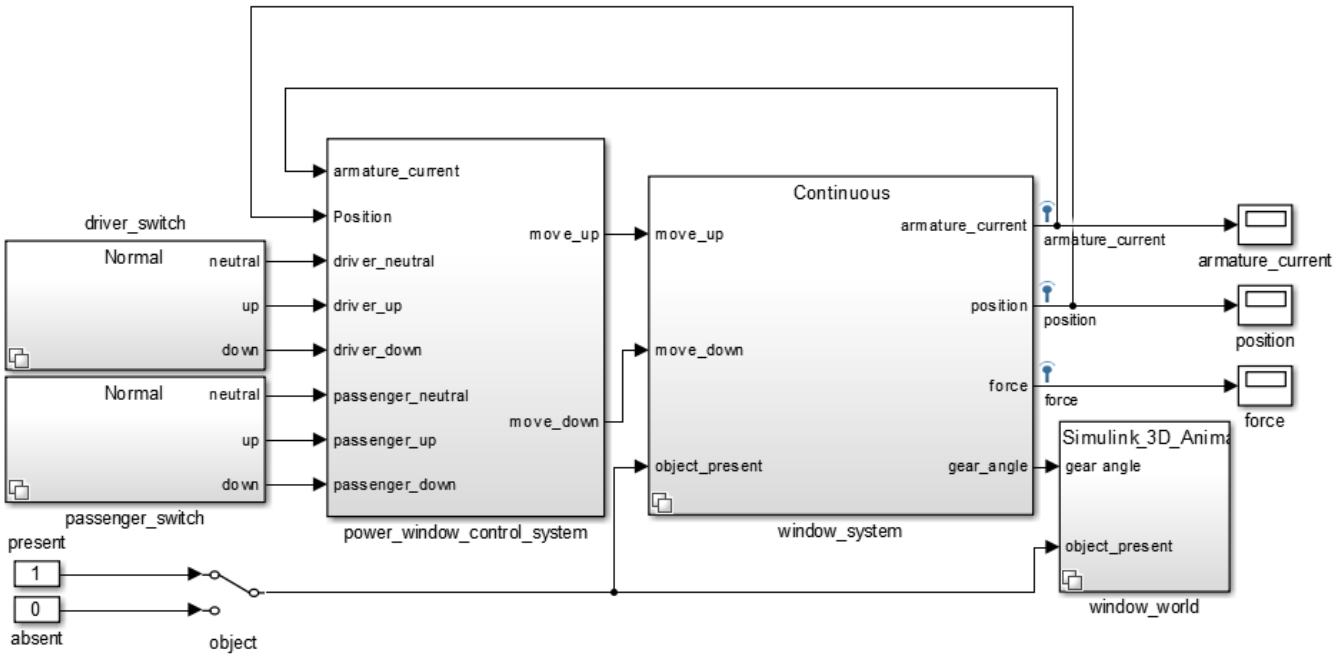

Figure 4.5: General model realization

In Figure 4.6, the insights of the Power Window Control System are shown. Four subsystems: *detect_obstacle_endstop*, *validate_driver_state*, *validate_passenger_state* and *control* are shown.

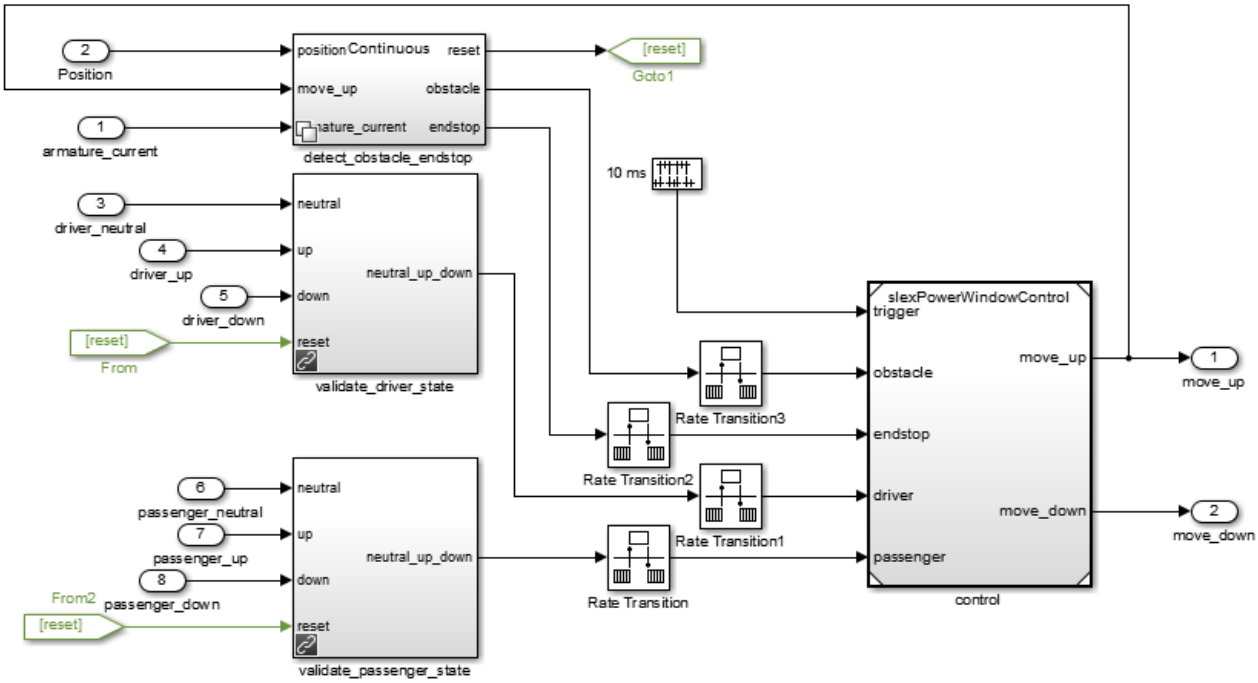

Figure 4.6: Components of the *power_windows_system* sub-system

4.3 Limitations of the Model

The major limitation of this model is the level of simplicity and the lack of dynamic behavior, which could be interesting to explore. The model contains only a few functional block diagrams with a limited level of granularity, which is a very important aspect. Another limitation is that the control system is implemented in the Stateflow chart. For example, there is no such a control system containing PID controller.

4.4 Requirements Categorization

In Software Requirement Engineering (SRE) requirements classification defines two categories: **Functional Requirements** and **Non-functional Requirements** [1]. The most fundamental definition of **Functional Requirements** can be stated by answering the question: What should the system do? A more sophisticated definition is: "A requirement that specifies an action that a system must be able to perform, without considering physical constraints; a requirement that specifies input/output behavior of a system" [22]. In terms of giving an exact definition for **Non-functional Requirements** several complications arrise. In [21], M. Glinz explains that even with a relative large amount of trials, the problem of finding a specific definition is that there is no consensus about the concepts for each term.

The non-functional requirements are not relevant for our goals and therefore will not be further mentioned. On the other hand, the functional requirements are in the range of our interest.

4.5 Description of Requirement Types

After performing extended research, there were no significant findings showing the taxonomy for functional requirements in automotive domain. In the Thesis [32] Liliegard M. and Nilsson V. defined the categories for the functional requirements. Inspired by this classification, a new classification was developed to fit the purpose of the Thesis and is shown in Figure 4.7.

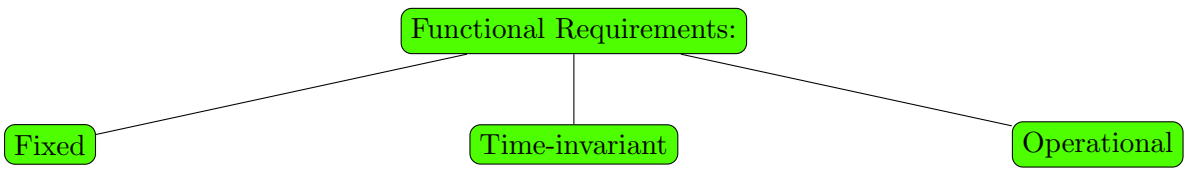

Figure 4.7: A classification tree for the functional requirements category

The total number of the functional requirement specifications categories were obtained and its definition can be found in the following list:

- **Fixed requirement category:** *'The condition for output signal(s) is based only on current input signal(s). Time has no relevance in this case.'*

  Example: "If armature current is greater than 15, endstop is detected"

- **Time-variant requirement category:** *'Defines the condition of output signal(s) that is based on the change within time interval.'*

  Example: "If the up command is issued for at least 200ms and at most 1s, the window has to be automatically opened completely"

- **Operational requirement category:** *'Defines the output signal values for a condition which is not enough to consider the requirement fully tested if the condition is fulfilled.'*

  Example: "When the driver does not issue a command, the passenger may assume control"

4.6 Analysis of Defined Requirements

The following Figure 4.8 shows an overall distribution through all previously defined functional requirement specifications in regards to different categories:

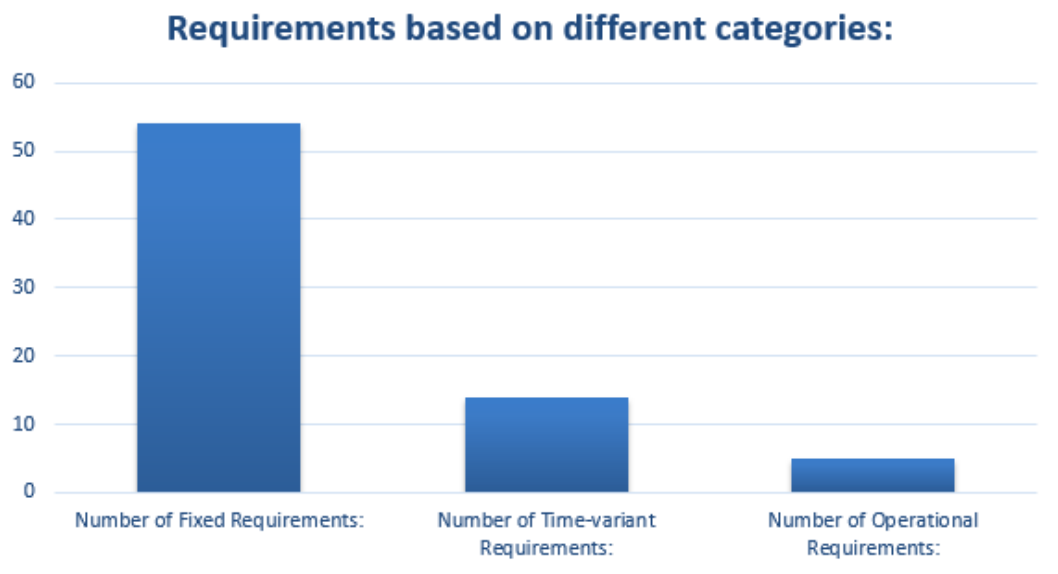

Figure 4.8: Categorization of the functional requirements

The total number of 73 functional requirement specifications were defined for testing purposes. The main category of requirements is under the **Fixed** category with 74 percent following by 20 percent for the **Time-variant** category and 6 percent for the **Operational** category.

Another necessary categorization is based on the levels of the system. Precisely, based on its granularity of the SUT that is determined by Test Scope [53] (see Section 2.2), where three levels of the system are known: *system level*, *integration level* and *component level*. Generally, creating several layers of granularity gives additional problems in terms of explaining exact boundaries between manufacturer and supplier that might have totally different interpretations, which might lead misunderstandings. Thus, the importance of informal and formal reviews comes into account in order to guarantee the compatibility and integrity [50].

This categorization is crucial in order to support one of the objectives, which is to identify at what level of the system is it worth to perform MBT methods within the Simulink platform. In Figure 4.9, the categorization based on the different levels of the system can be found. More than 50 percent of requirements were describing its functionality from the integration level point of view. Note that the integration level in this Thesis is considered to occur when two components are joined together into one sub-system. Furthermore, 33 percent were considered to cover component level and lastly, 15 percent the system level. A complete Table describing all the requirement specifications is available in the attached Appendix CD.

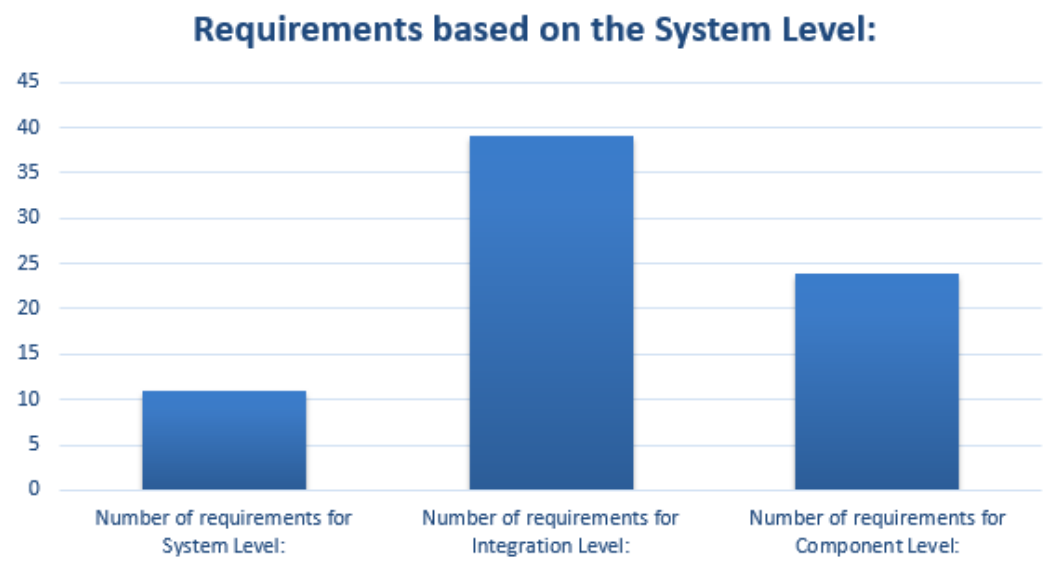

Figure 4.9: Categorization based on the different levels of the system

5 Case Study: TPT Tool

This chapter introduces the first selected MBT tool (the TPT tool). The aim is to focus on explaining how are the different requirement categories modeled, executed, evaluated and reported. In the last part, conclusions regarding to complexity, scalability and applicability will be presented.

5.1 Time Partition Testing Approach

The Time Partition Testing approach is designed to perform systematic functional model-based testing of embedded control systems applications [10]. Functional or black-box testing is a procedure where the selection of data is strictly based on specifications of requirements. Most of the embedded system applications interact with physical environments. For the Power Window Control Systems relevant in this Thesis, the system behaves according to either the driver's or the passenger's decision and therefore one can conclude that a driver/passenger represents the physical environment.

The Time Partition Testing approach is based on the *testlets*. A testlet provides an exact description of the input data used for testing, called test data. In [29] Dr. Lehmann mathematically defines the formal term of test data as a function:

F: $\tilde{i_1} \times \cdots \times \tilde{i_n} \rightarrow Time^{\infty}\ \tilde{o_1} \times \cdots \times \tilde{o_n}$ with K_I=(i_1,...,i_n) and K_O=(o_1,...,o_n)

where K_I and K_O characterize sets of input and output channels. $t_{stop} : Time^{\infty}$ represents the termination time. It can be concluded that this function describes the termination time as well as the signal course for each output channel with respect to signal courses of input channels.

Following the function describing the formal definition of the Time Partition Testing approach and can be defined as follows: "A non-empty set of functions $T \subseteq \tilde{i_1} \times \cdots \times \tilde{i_n} \rightarrow Time^{\infty}\ \tilde{o_1} \times \cdots \times \tilde{o_n}$ to the sets of input and output channels K_I and K_O is called *testlet*, an element from T is called *testlet element*" [28].

Furthermore, Lehmann with his theoretical concept of the Time Partition Testing approach, has defined three major methods that propose the recipe making techniques to develop testlets, as mentioned above, a pillar of the entire TPT approach [28]. These are:

1. **Direct Definition:** Used for the most fundamental scenarios, where no further contingency is required in terms of timing. For example the following specification: "Whenever the driver presses the command *up*, the window should *move_up*" could be modeled with no more than five elements and thus data selection technique: *Direct Definition* is the most suitable one [28].

2. **Time Partitioning:** In this method the test data is divided into several phases with respect to time. This provides the flexibility to model even more complicated test cases. For instance: "Driver presses *up* command, after three seconds the *obstacle* is detected, then the window should be lowered by 5 cm.". In this case, the temporal course of each single test data will be partitioned into three stages: pressed switch, obstacle hit, emergency routine of lowering window. *Time partitioning* extends the single course of *Direct Definition* technique.

3. **Data Partitioning:** This technique can be characterized by a classification tree, where each class represents certain test data.

Graphical representation of the techniques are shown in Figure 5.1.

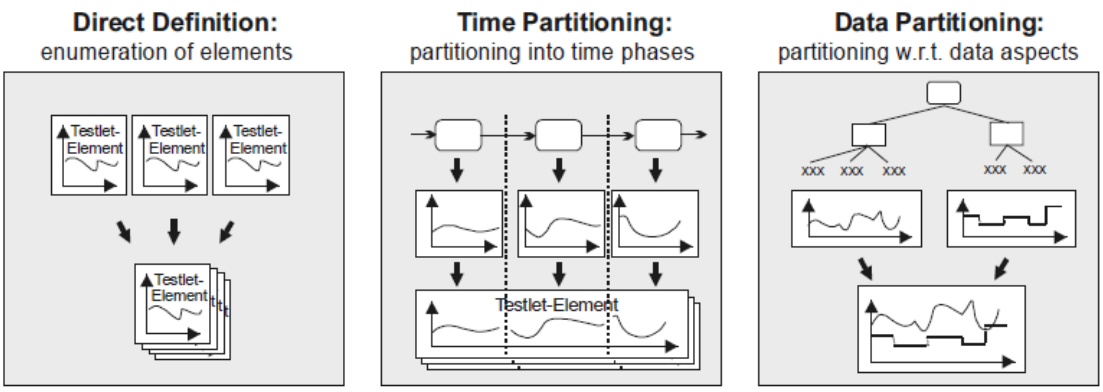

Figure 5.1: Graphical representation of TPT techniques, retrieved from [28]

5.2 The TPT Tool

In this section the TPT tool, which is based on the Time Partition Testing approach described in section 5.1, will be presented. According to [14] the objectives of TPT can be characterized by the following:

- supports a test modeling technique that allows the systematic selection of test cases.
- provides a precise, formal, portable, and simple representation of test cases for the Model Based Design.
- provides an infrastructure for automated test execution and evaluation.

A summarizing description of the tool was retrieved from [54] where the author describes the TPT tool as: "automaton-based approach to model the test behavior and associates with the states pre- and post- conditions on the properties of the tested system (including the continuous signals) and on the timing." In other words, the TPT tool is a signal-based oriented tool, which performs functional testing at every level of the system.

The following diagram shown in Figure 5.2 was developed to explain the importance of the entire testing process of the TPT tool.

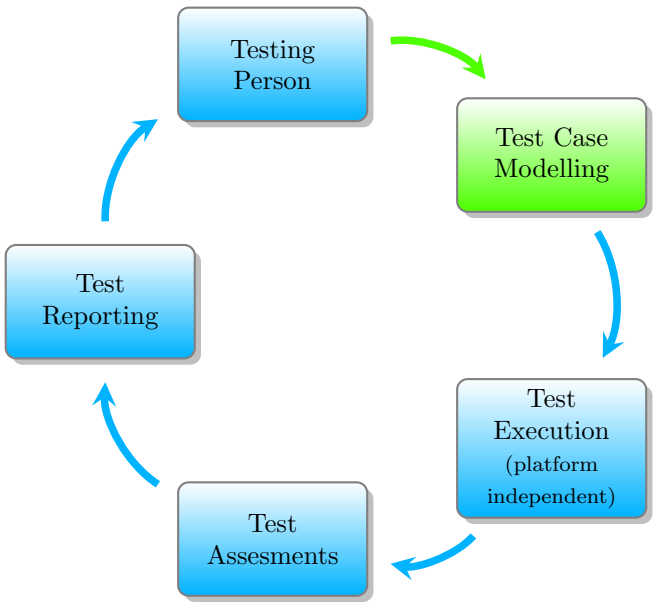

Figure 5.2: Diagram representing testing process in TPT

Figure 5.3 represents the Test Case Modeling general procedure. This process starts with a precise model selection used for testing purposes. Under these prerequisites, one must load all the parameters into the Simulink work space in order to proceed model import successfully. In the next step, after the model is successfully imported into the TPT platform, the user is required to select a subsystem. This option is crucial due to the fact that one of the goals is to examine a testing effort performed at the lowest system level (at the component level). Consequently, several problems might occur, which could cause the rejection of the subsystem selection. For example, if the selected subsystem contains some characters that are not supported by the TPT platform, the subsystem is rejected and the user has to fix the resulted issues in order to proceed into the next step. If succeed, all the input/output channels, parameters and constants might be defined and imported regarding to let the TPT tool automatically generate test platform.

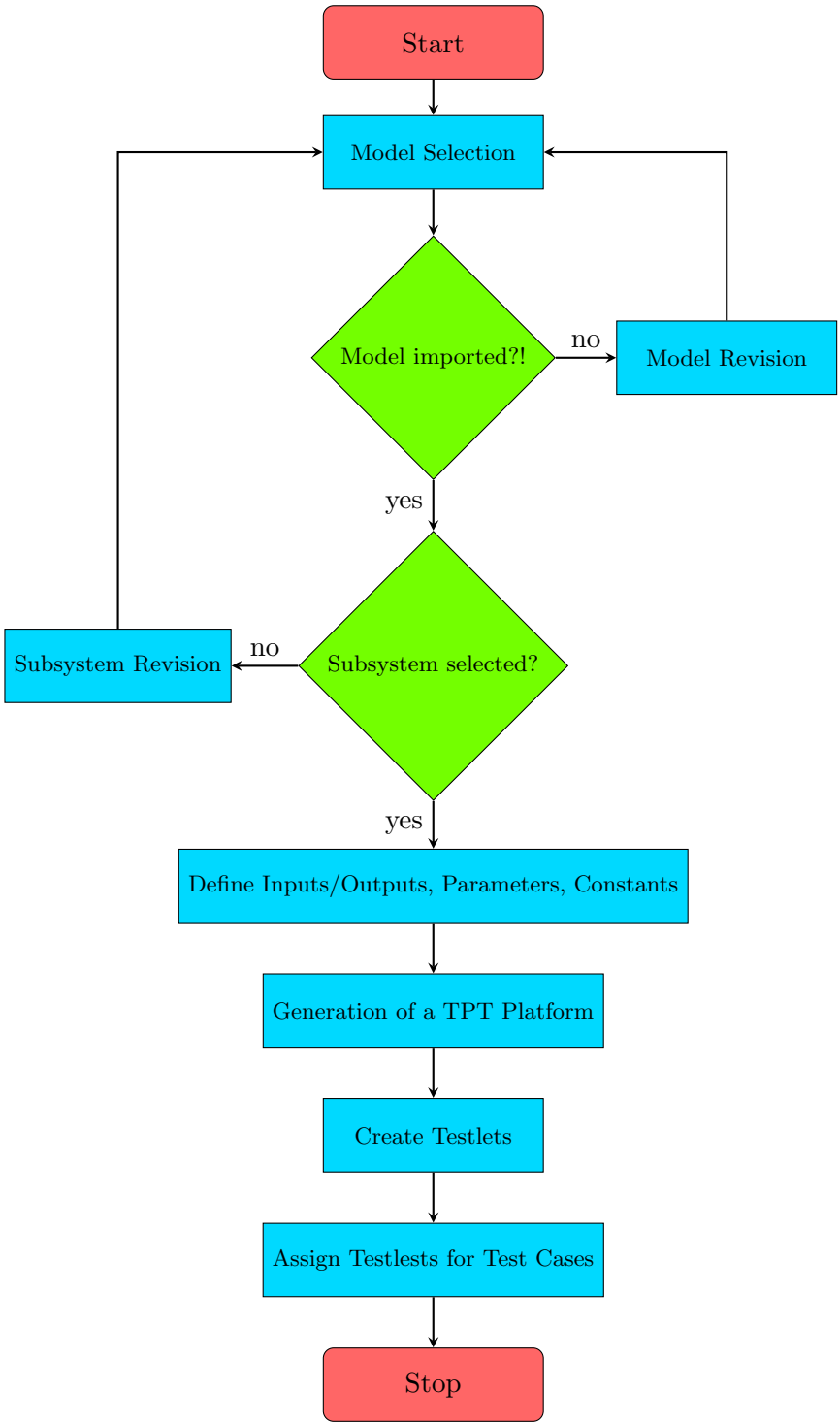

Figure 5.3: Flowchart representing Test Case Modeling general procedure

The automatically generated the TPT test platform shown in Figure 5.4 contains the following elements: first, the functional block diagram: *detect_obstacle_endstop* is the original block diagram that is desired to be tested. The inputs/outputs from this block diagram are connected with Bus Creator block, where the function of the Bus Creator Simulink Block combines the set of signals into a bus. Moreover, the last functional block diagram *TPT_SFun_frame* represents the actual SUT. The TPT tool then converts the original Simulink block diagram into corresponding C code that can be easily executed in the platform independent TPT Virtual machine. TPT-VM is implemented in ANSII-C and has the response time in microseconds [41]. It can be observed that the outputs from the original block diagram (*detect_obstacle_endstop*) are inputs for TPT SFun frame and outputs from TPT SFun frame are inputs for the original block diagram (*detect_obstacle_endstop*). This provides an additional flexibility to separate the test execution and the test evaluation. More details are given in the next section.

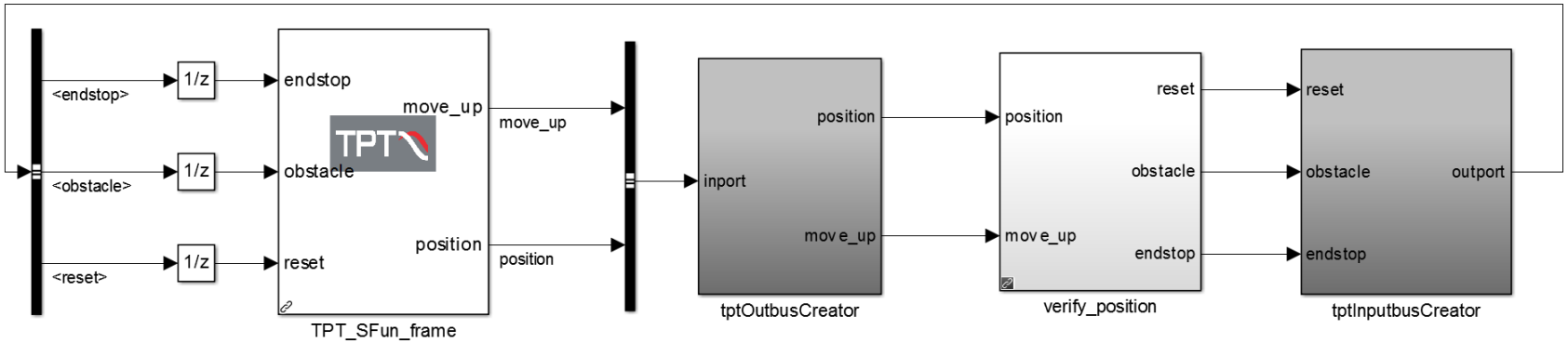

Figure 5.4: The example of TPT SUT scheme representation

Furthermore, the TPT tool has selected process automatons as the modeling language. Specifically, the TPT automaton is a hierarchical automaton, which allows to be decomposed into several sub-automatons and thus to cover even highly complex test cases [41]. Figure 5.5 shows an example of the TPT automaton with a short explanation of each term involved.

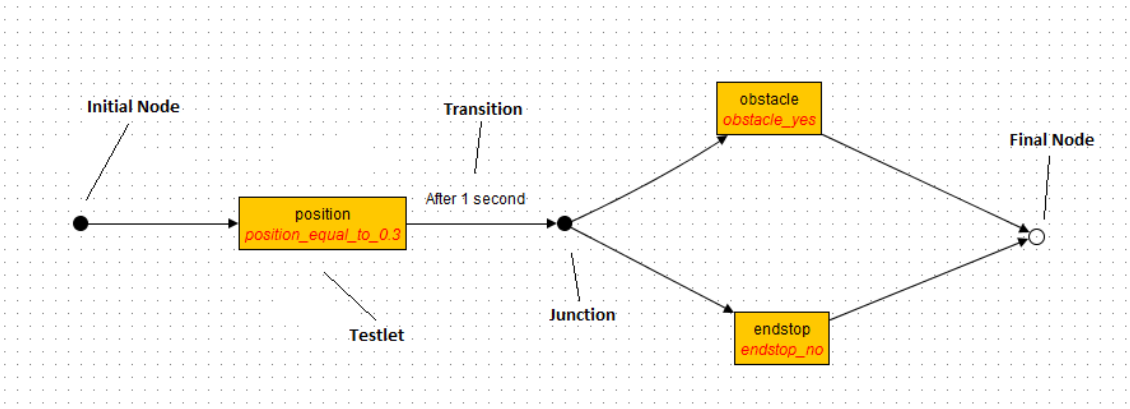

Figure 5.5: Example of TPT Automaton

- **Initial Node** describes an entry into the test execution. The transition at the initial node starts from $t = 0$ and only one transition from the initial node is possible in order to avoid non-determinism.
- **Junctions** serve in case of path branching.
- **Transition** defines a condition in which the change of continuous behavior takes place
- **Parallelization Line** serves to separate parallel automatons for the purpose of simultaneous execution (not shown in Figure 5.5).
- **Testlet(State)** determines the behavior of an automaton.

With the use of these elements, the TPT automaton might be modeled. However, it is necessary to specify the behavior of such an automaton. In other words, to specify the test data selection. The TPT tool provides four different categories for defining testlets:

1. **Test Step List Testlet** this technique has the same characteristics as the *Direct Definition* method from Section 5.1.
2. **Time Partition Testing Testlet** is an equivalent to the *Time Partitioning* method from Section 5.1.
3. **Reference Testlet** allows to use another testlet several times in the same automaton of a project. [41]
4. **Library Testlet** allows to use another testlet several times in the same automaton of different projects. [41]

5.3 Test Modeling in the TPT Tool

This chapter covers the following research question:

"How can these requirements be modeled in the TPT tool?"

This question will be answered by using the two different testlet categories presented in the previous section and the identified requirement specifications. Operational category could not be modeled with the TPT tool and will not be discussed further in this section.

5.3.1 Time Step List Testlet: Fixed Category

The majority of all functional requirements were able to be modeled with the Time Step List Testlet (TSLT) technique. As an example, requirement *number 5* is introduced:

"*If move_up pressed and position greater than 0.3, obstacle is detected*"

This specification describes the functionality of one of the sub-block named: *verify_position*, which is shown in Figure 5.6. This sub-block covers functionality at the lowest level (**component level**) and it is a part of an overall *power window control system* functional block diagram.

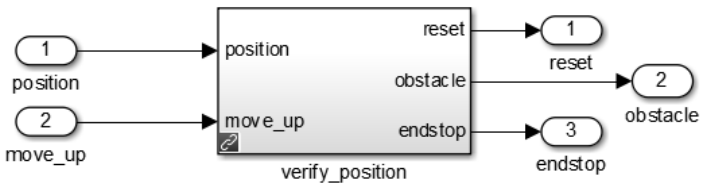

Figure 5.6: TPT req5 Functional Diagram

The first step is to define the number of testlets needed to perform testing activities. Due to the simplicity of this requirement a single TSLT testlet is sufficient. The testlet is shown in Figure 5.7.

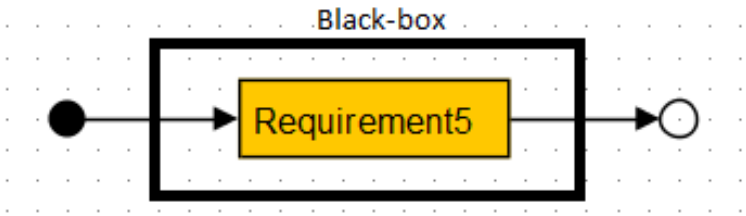

Figure 5.7: TPT req5 testlet

The testing procedure is modelled by enabling the input channel *move_up* to become true. The change of the *position* is demonstrated with the TPT command **Ramp**. Then the desired value of the *position* is checked by the command **Wait Until**. Lastly, with the help of the TPT command **Compare**, the output channel of *obstacle* will be compared and concluded to whether the value of the signal channel *obstacle* became true or false. The graphical representation of this modeled test case is shown in Figure 5.8.

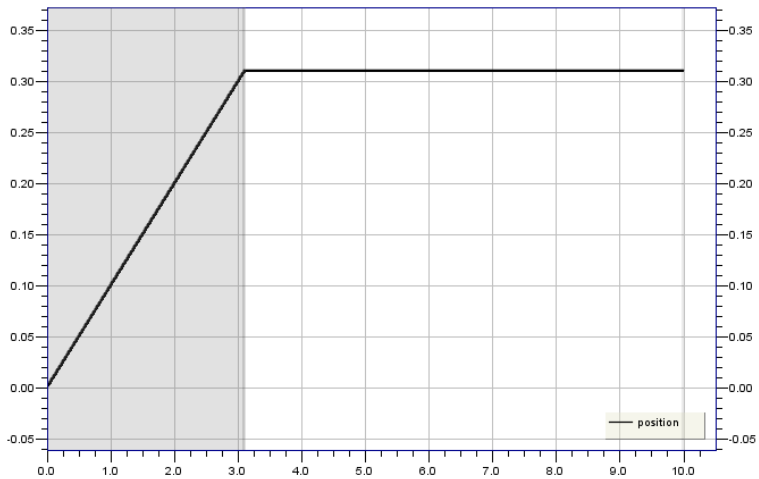

Figure 5.8: TPT req5 graph

The second example for the fixed requirement category covers the functional requirement that defines the functionality at the **integration level**. This means that several components are combined together into one functional block diagram. Precisely, the functional block at the integration level named *validate_passanger_state* (Figure 5.9) consists of three small component level functional block diagrams (*check_up*, *check_down* and *mutually_exclusive*) and are represented in Figure 5.10.

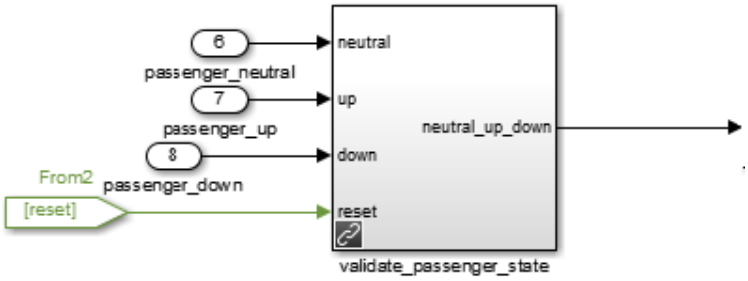

Figure 5.9: TPT req32 Functional Diagram - integration level

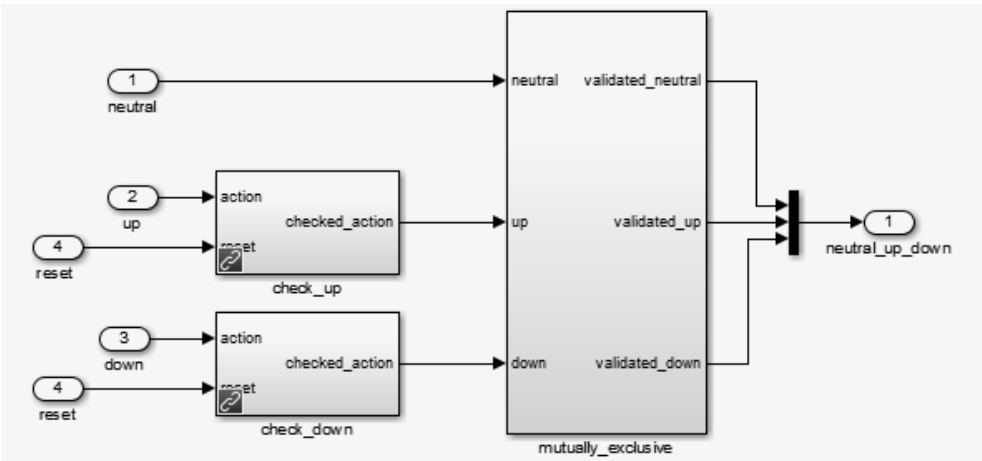

Figure 5.10: TPT req32 Functional Diagram - component level

For this purpose, the specification of requirement *number 32* will be demonstrated. The aim of this requirement is to validate the passenger state based on the passenger's choice (commands: *up, down, neutral*). This requirement was also modeled as a single automaton. With the help of these three variants (**channel**, **wait** and **compare**), all possible test cases were conducted in order to verify the functionality of *validate_passenger_state* functional block.

5.3.2 Time Partition Testing Testlet: Time Variant Category

This section will demonstrate the modeling of **Time-variant** requirement as categorized by the help of **Time Partition Testing Testlet (TPTT)**. The TPTT method models small sub-systems within one single automaton. The following example defines the functional requirements *number 64* and the *number 67*. It is formulated as:

"*The window has to be fully opened and fully closed within 4s*"

This requirement is considered to be at the **system level** and the goal is to describe an overall functionality of the functional block called *power_window_control_system*, which is shown in Figure 5.11.

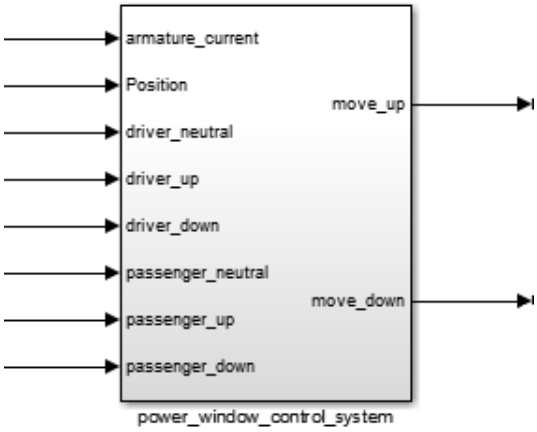

Figure 5.11: TPT req6467 Functional Diagram - system level

The aim was to model this functional requirement with the help of **Time Partition Testing Testlet**. For this purpose, two of the TPTT were created. The first is for opening the window within 4 seconds and the second one is for closing the window within 4s. A graphical representation is shown in Figure 5.12.

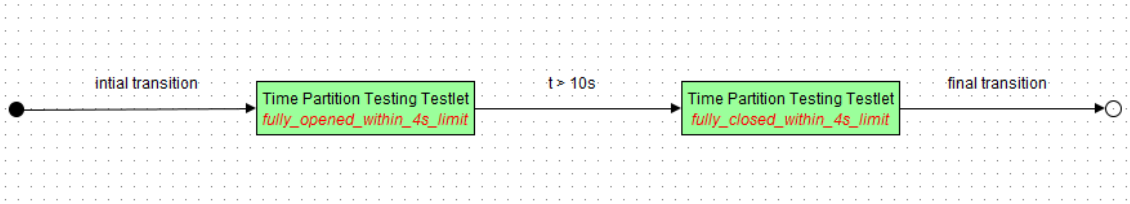

Figure 5.12: TPT req6467 Time Partition Testing Testlet main

In order to define the behavior for each TPTT, additional testlets were defined. Inside of the TPTT, two consequent **Time Step List Testlets** were modeled (Figure 5.13) and inside of the TSLT the actual behavior with the help of variants: **Channel**, **Wait** and **Compare** was modeled.

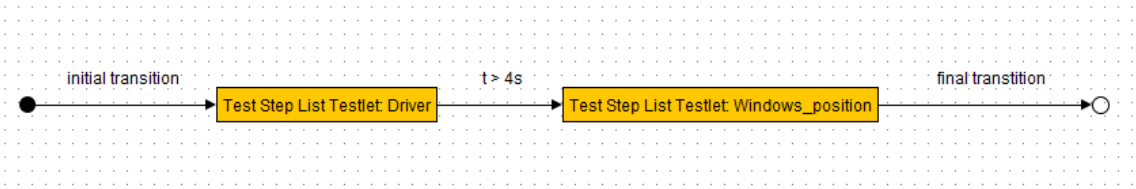

Figure 5.13: TPT req6467 Time Step List Testlet

In conclusion, **Time Partition Testing Testlet** provide a large degree of freedom in terms of modeling possibilities. The Time Partition Testing Testlet gives the possibility to extend the range up to several hierarchies. In terms of modeling of different requirements, the Time Partition Testing Testlet is suitable to model both Fixed and Time-variant functional requirements categories.

5.4 Test Execution in the TPT Tool

This sub-section deals with the second element of the loop that represents TPT tool process (Figure 5.14): **Test Execution**

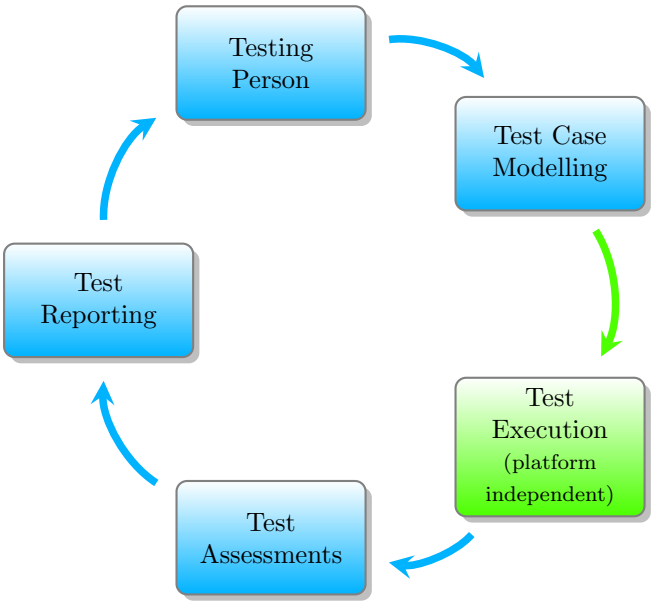

Figure 5.14: Diagram representing testing process in TPT

Generally, test execution means the execution of test cases and test scenarios. In the TPT tool, test execution is performed in completely platform independent environment, called TPT Virtual Machine (TPT-VM). Due to this reason, every test case that has been created during Test Case Modeling has to be compiled first into a highly compacted byte code that has been specifically designed for internal purposes of TPT [14]. Precisely, this byte code contains information about the formal semantics of testlets that allows the process to be fully automated.

The question might arise, why is there no automation already included during Test Case Modeling?

The reason for that is that all the specifications of requirements cannot be interpreted in a general matter. That means that one person might have a different interpretation of a certain specification of the requirement compared to another person. Moreover, the automation is only possible during the test execution and the test evaluation steps, where the test drivers perform the execution of testlet elements [28]. The process is graphically shown in Figure 5.15.

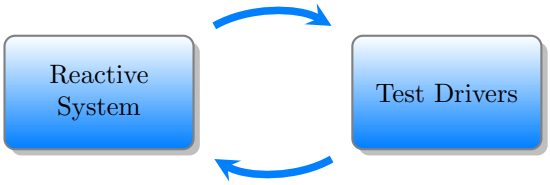

Figure 5.15: Diagram representing interaction between the system and test drivers during automatic text execution

Generally, test drivers can be explained as small executable programs that store test cases in executable form [34]. Lehmann explains that due to the fact that the testlets are based on a dense temporal model, time discretization is crucial [28]. Moreover, as the TPT-VM runs platform-independently, additional degrees of freedom is considered. That means that selected test cases can be executed on MiL, SiL, PiL and HiL platforms. To summarize, the test cases are automatically executed but there is no evaluation that would confirm the results of the SUT. Therefore, Test assessments are necessary and explained in the next section.

5.5 Test Assesments in the TPT Tool

Moving forward in our TPT process diagram, into: **Test Assessments**.

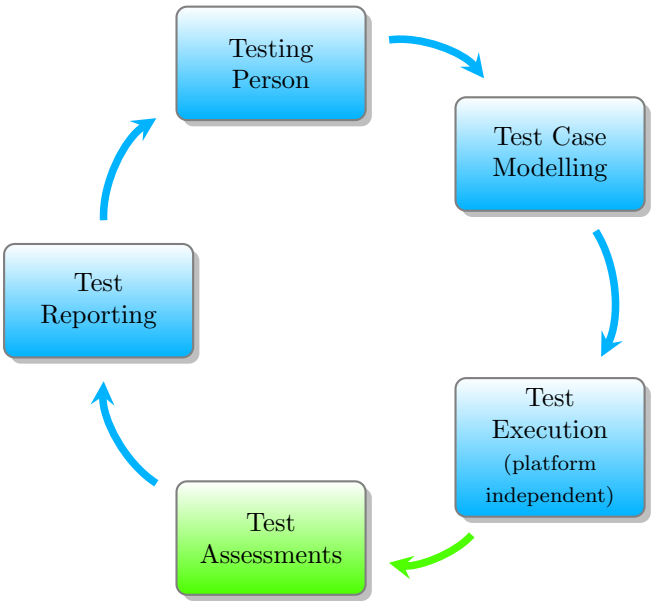

Figure 5.16: Diagram representing testing process in TPT

Test Assessments are test evaluation techniques in the TPT tool. The TPT Test Assessments use **Assesslets** as watchdogs to determine failure [28]. An assesslet is based on the Python programming language and the TPT tool provides a large amount of available assesslets. The following list represents all possible predefined test assessments within the TPT tool:

1. Check Log Entries
2. Equivalence Classes
3. Import Measurements
4. M-script Assesslet
5. **Min/Max Comparison**
6. Script
7. Sequence Check
8. Signal Comparison

9. **Test Step List Assessment**
10. Timeout
11. **Trigger Rule**
12. Variable Definition

For our case, the following assesslets were considered:

- **Test Step List Assessment:** is the most fundamental assesslet and mandatory when the TPT command **Compare** is used. An example is demonstrated with the help of the requirement specification *number 34* defined as:

  "$validated\_down = down \wedge \neg\ up$"

  After executing and performing Test Step List Assessment against this requirement, it could be concluded that the test case was performed successfully. The success of the assessment is highlighted with a green color and the status: PASSED. Graphical representation is shown in Figure 5.17.

3.1.1.3.1.1. ✔ Variant Req25 (in state 'Testlet')

	No.	Type	From	To	Expression
	1	Channel	0s		up := false (up := false)
	2	Channel	0s		down := true (down := true)
	3	Wait	0s	5s	5s
✔	4	Compare always	5s	59.99s	validated_down == true

Step List Assesslet (from variant 'Req25', testlet: 'Testlet', time: {0s <= t < 59.99s})

Figure 5.17: Example: Test Step List Assessment - PASSED

Consequently, in the case of a negative result, an assessment is highlighted with a red color and the status: FAILED. Figure 5.18 represents an example of requirement *number 40* described as:

"If *passanger* requests *up*, *endstop* is hit, *move_up* should be deactivated"

1.1.1.3.1.1. Variant Req34_2 (in state 'Testlet')

	No.	Type	From	To	Expression
	1	Channel	0s	5s	obstacle := false
	2	Channel	0s	5s	passenger := [true, false, false]
	3	Wait	0s	5s	5s
	4	Channel	5s	59.99s	trigger := false
	5	Channel	5s		passenger := [false, true, false] (passenger := [false, true, false])
	6	Channel	5s	59.99s	endstop := true
	7	Compare always	5s	59.99s	expected moveUp == false always in interval {5s <= t < 59.99s} failure at t=5s: expected 0, got 1

Step List Assesslet (from variant 'Req34_2', testlet: 'Testlet', time: {0s <= t < 59.99s})

Figure 5.18: Example: Test Step List Assessment - FAILED

- **Min/max Comparison:** This assesslet checks whether the value of a channel is within previously specified minimum/maximum range values. The channel value might be able to reach three various values:
  1. **Negative value** *channel(t)* - *min(t)* if *channel(t)* is below bounds (*channel(t)* $<$ *min(t)*)
  2. **Positive value** *channel(t)* - *max(t)* if *channel(t)* is above bounds (*channel(t)* $>$ *max(t)*)
  3. **Neutral value 0.0** in case of *min(t)* $\leq$ *channel(t)* $\geq$ *max(t)* [42]

  In our case study, the min/max assesslet was used in relation with the functional *number 64*:

  "The window has to be fully opened within 4s"

From documentation it is known that the window is considered to be fully opened when the position reaches value 0.4. Based on this condition, two consequent boundaries were set up. The first one covers the minimum boundary defined by position at 0.0 and the second one for maximum boundary with the value of 0.4. The following Figure 5.19 represents the graphical form the application of a min/max assesslet.

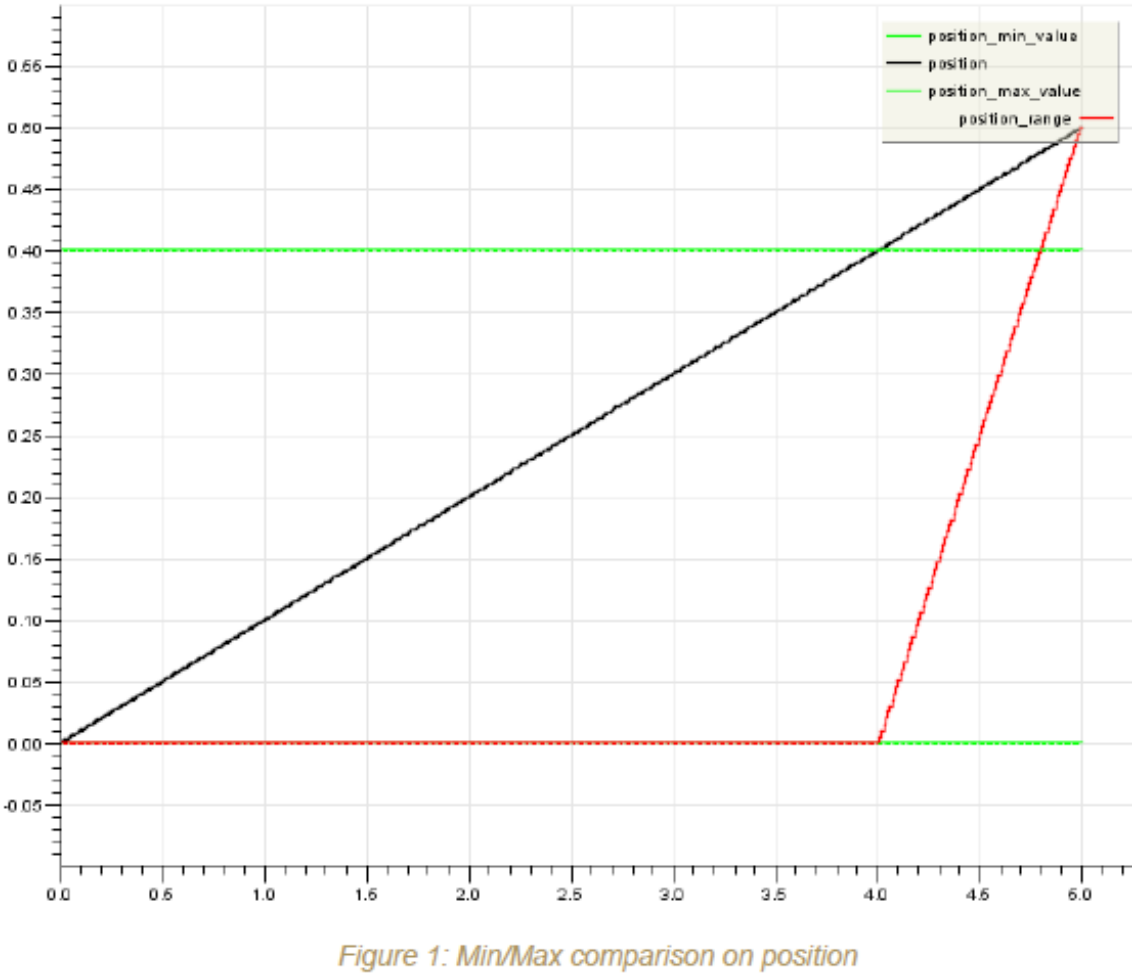

Figure 5.19: Example: Min/Max Assessment

- **Trigger Rule: 'While Condition is true':** The aim is to check the outcomes while the previously specified condition is true. A general TPT tool window, where the user is able to specify conditions, is shown in Figure 5.20.

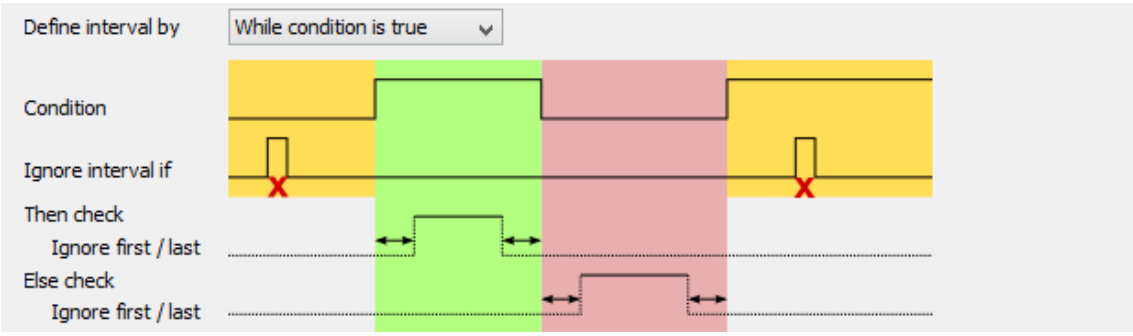

Figure 5.20: General window for describing Trigger rule, retrieved from [41]

Trigger Rule assesslet has been used with the functional requirement *number 71*:

"The force to detect when an *obstacle* is present should be less than 100N (maximum current must be 1.7A)."

The assesslet declares the condition for driver's request to move the window upward and the *amature_current* must be greater than 1.7A. Whenever this condition becomes true, it is desired to check whether the control system reacted correctly and assigned the command *move_up* to be true. A graphical representation is shown in Figure 5.21.

While condition is true

1.1.1.3.2.1. Definition

Type	While condition is true
Condition	driver_up == true and armature_current > 1.7
Then check (once)	move_up != true

1.1.1.3.2. ✔Trigger Rule

	Name	Value	From	To
✔	Trigger Rule	true	0s	59.99s

Figure 5.21: Example: Trigger Rule Assessment

5.6 Test Reporting in the TPT Tool

Reports serve to communicate information of the result of the research and the analysis of data. This information supports further decision making processes [2]. Generally, important aspects are the audience and the purpose of the report. In this section, the focus will be on explaining the role of reports in Model-Based testing with help of the TPT tool.

Test Reporting is the last instance of the TPT process diagram shown in Figure 5.22.

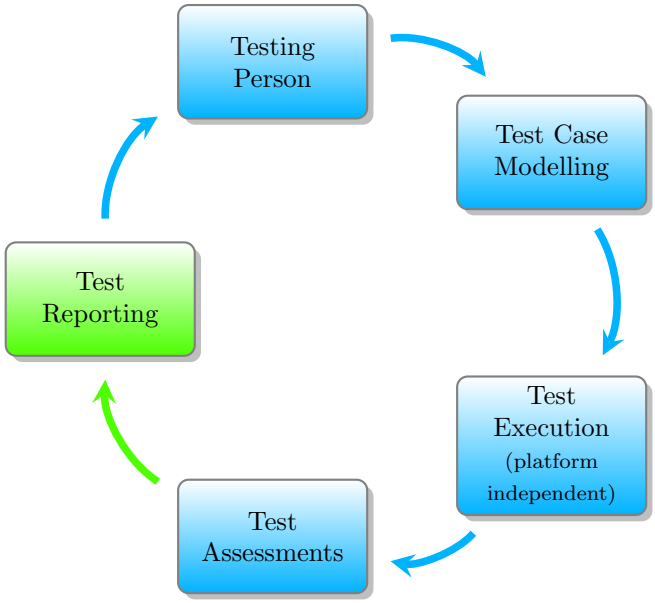

Figure 5.22: Diagram representing testing process in TPT

The key audience for test reports in MBT, respectively MBD is a stakeholder/customer, test engineer, MBD engineer and his team leader. In the TPT tool, test reports are designed to be in human-readable format and therefore a user is able to understand it even with lack of technical background [14].

The report is divided into 4 different sections:

1. **Overview Section:** the aim is to graphically represent information about test execution, whether a test case conquers status: *Passed, Inconclusive, Failed or Execution Error* [41]. This section serves as the primary source of the most fundamental information about the testing activities within the tool. This scope of the information is supposed to target the following audience: a stakeholder/customer.

2. **Variables Section:** gives detailed information about assessment variables and asseslets [41]. The following section gives a deeper insight of the selected variables. It is mostly useful for test or MBD engineer, who needs to see clearly the track of the variables used in MBT.

3. **Requirements Section:** is used for requirement trace-ability, which can be connected to safety norms (for example: ISO 26262) [41]. The section gives information that is targeted to team leader needs.

4. **Individual test case information sorted by platforms:** sorts test case information based on the different platforms: MiL, HiL, SiL or PiL. This is an important aspect, due to the necessity of analyzing each test cases in each environment [41]. This is the most comprehensive section that is useful for all the key audiences.

5.7 Summary and Analysis

In this chapter a detailed investigation of the first MBT tool: the TPT tool was presented. The main focus was to find out how the different requirement specifications at the different level of the system are modeled, executed, evaluated and reported. An example for each different requirement specification based on the different levels of the system as well as different requirements category was introduced. An observation showed that the TPT tool is scalable by looking at the different levels of the system, however it is necessary to mention that it is not negative, because the tool is platform independent from the Simulink. That means that the tool can store all the lower level tests and enables to test from all of the system levels consequentially. The results showed that the TPTT is the most suitable to model requirements that are on the *system level*, the most complex ones. The difference between *Fixed* and *Time-variant* categories was in case of the TPTT not observed. In contrast, the DDT is the most suitable to model requirement specifications at the component/integration level and only *Fixed* requirements category was able to be fully modeled and executed with the DDT.

In terms of applicability, it was confirmed that the TPT tool is a very powerful tool that enables to perform systematic testing from various perspectives. From the analysis of the total number of 63/73 functional requirement specifications that were suitable for testing in the TPT tool can be concluded as following: due to the reason that the TPT is platform independent, modeling and execution (see Figure 5.3 for a better visualization) is done independently from Simulink, each time a new TPT platform configuration must be established. This process takes a large amount of time and for this reason the most effective way of performing MBT activities with the TPT tool is to focus only on the integration and the tpt level testing. The MBT at the component level was found to be rather ineffective compared to MBT performed at the integration level.

5.8 Limitations

The first limitation of the TPT tool with respect to the Simulink platform is the level of integration between Simulink and TPT. The time of connecting the platform with Matlab/Simulink resulted in large delays followed by error messages that the tool does not support the following name types in the model. To change the names of the blocks might become very painful when it comes to large models. Another limitation that was described in [54] states that the TPT tool suffers from lack of automation, which makes it very difficult to learn.

6 Case Study: SLDV Tool

Simulink Design Verifier (SLDV) is the Matlab/Simulink Toolbox created by Mathworks introducing its first version for Simulink 2007a and has all the support to be used in development processes that follows ISO 61508, ISO 26262 and ISO 50128 [6]. SLDV uses a formal analysis of the model to satisfy the following objectives: **Design Errors Detection**, **Property Proving** and **Test Case Generation**. An illustrative representation might be found in Figure 6.1.

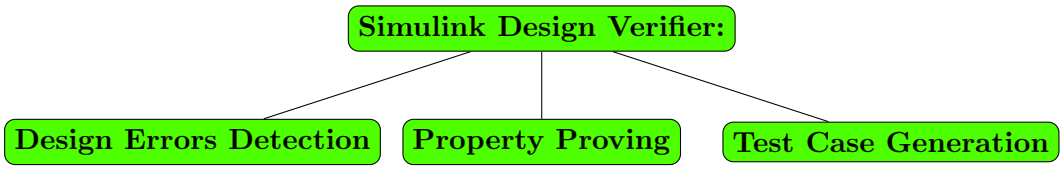

Figure 6.1: Possible objectives to achieve with the SLDV tool

This section will give an investigation for each of the SLDV features. The main focus will be on Property Proving, although the DED analysis will be also performed and examined for our case study. In terms of Test Case Generation, this section was deeply discussed in related works: [36], [5], [52], [32] and [30]. For this reason, only a general overview of this feature will presented. Before performing any of the shown objectives of the SLDV tool an automatic compatibility check must be executed in order to ensure that all the functional block diagrams within the model as well as solver settings are supported for testing purposes. The compatibility check might come up with three outcomes:

1. **Compatible:** all the functional blocks are supported and the model with all internal settings itself is compatible. When the compatibility check has status: *Compatible* the analysis is ready to be performed. A dialog box informing about the positive compatibility check is shown in Figure 6.2.

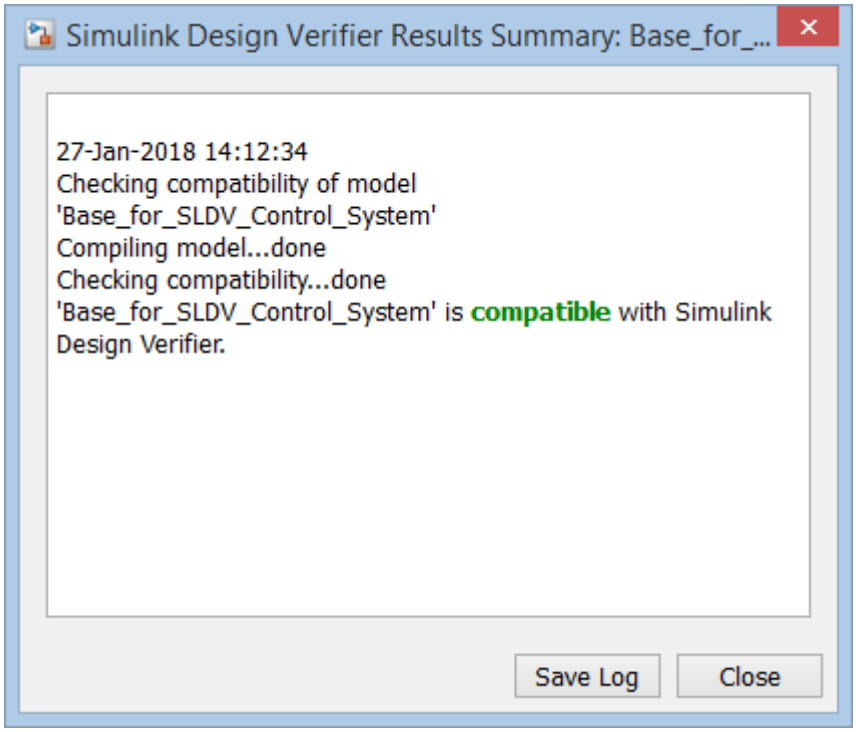

Figure 6.2: Positive result for a compatibility check

2. **Incompatible:** the model itself is incompatible [35]. A dialog box with the negative compatibility check, shown in Figure 6.3, is automatically generated with an additional diagnostic viewer that explains what caused the problem and gives some hints how to fix it (Figure 6.4). Precisely in this case, the negative compatibility check was caused by selecting variable-step solver used for time continuous systems and these systems are strictly restricted in the SLDV tool. Several unsupported blocks were inside of the model, which caused the work to be limited. For more details about the unsupported blocks, refer to the section 6.5.

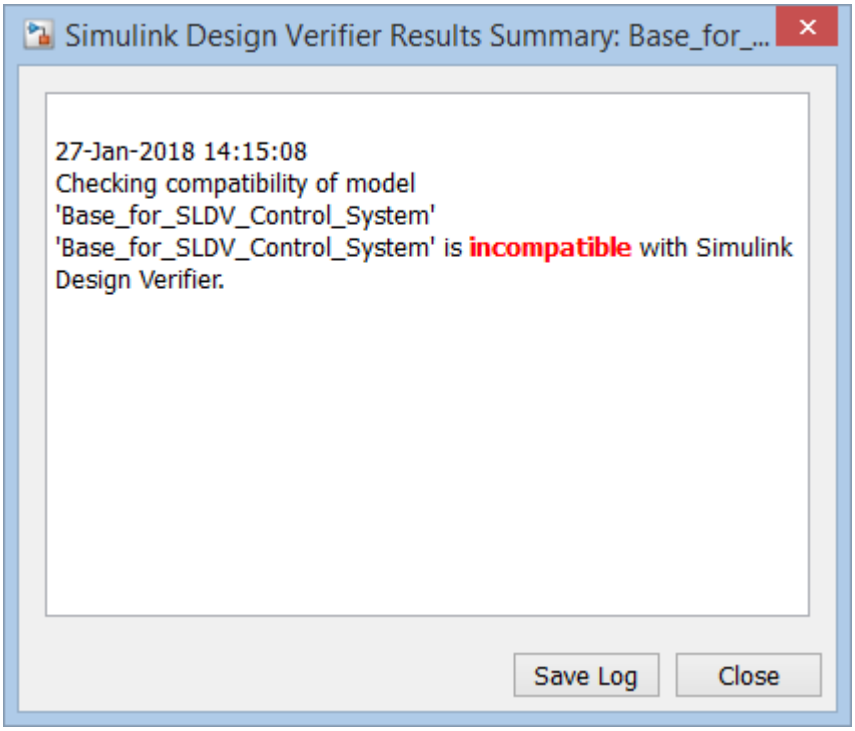

Figure 6.3: Negative result for a compatibility check

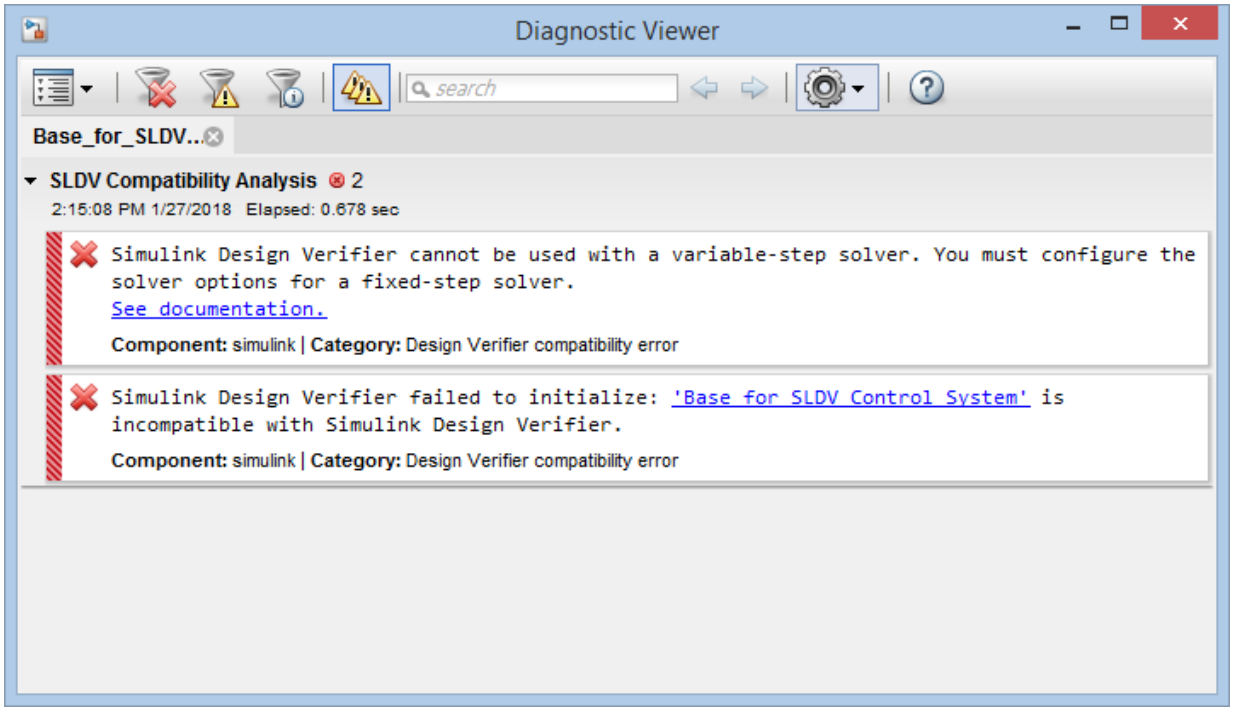

Figure 6.4: Diagnostic viewer in case of negative compatibility check

3. **Partially compatible:** exists when at least one unsupported feature is presented. For demonstrating purposes, one unsupported Simulink function *sqrt* was inserted into the model. As a result, the SLDV tool did not completely confirm an incompatibility, but performed the automatic stubbing of the unsupported block. An automatic stubbing is one of the Simulink features that enables to analyze the model but with the use of the interface of the unsupported object, not their actual behavior [35]. In our case, options of an automatic stubbing for some of the unsupported blocks were also initiated but not used. The reason was that an automatic stubbing effects the accuracy of the results directly. The figure demonstrating such a case is shown in Figure 6.5.

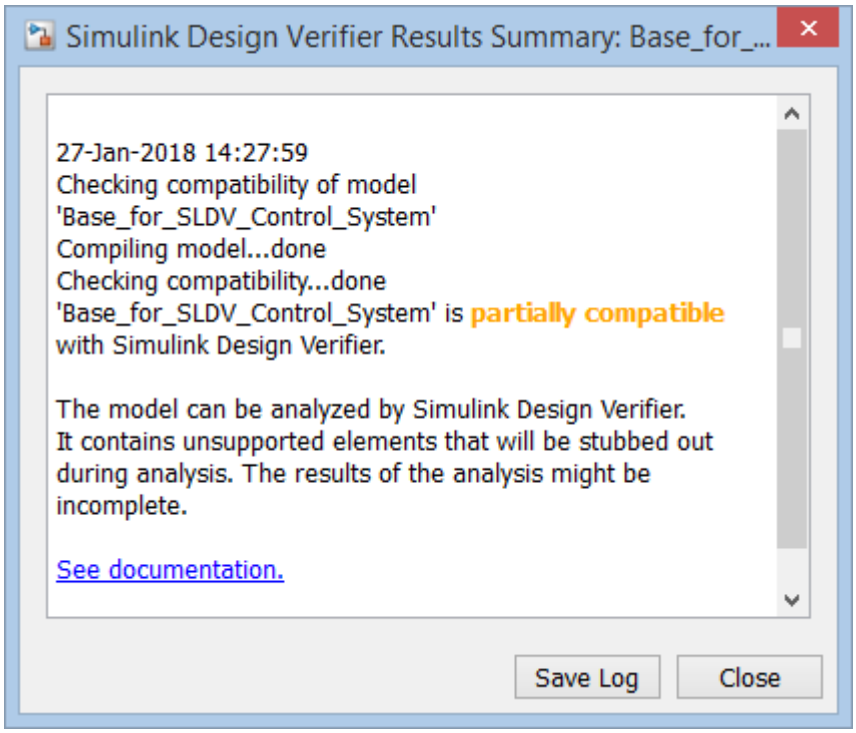

Figure 6.5: Partially positive result for the compatibility check

6.1 Design Errors Detection (DED):

Design Errors Detection (DED): provides an option to automatically identify design inconsistencies within the model. The following list gives an insight of design errors that SLDV is able to detect [35]:

- **Integer/fixed data overflow:**

  Specific input data that generates a non-deterministic behavior.

- **Division by zero:**

  A non-deterministic behavior caused by invalid mathematical operation.

- **Dead logic:**

  In case that designed functionality can never be activated.

- **Intermediate signal values that are outside the specified minimum and maximum values:**

  Specific input data that generates a non-deterministic behavior.

- **Out of bound array access:**

  Specific input data that generates a non-deterministic behavior.

The idea behind that is an attempt to find specific conditions that cause the error. Nevertheless, by performing DED a computation of ranges of signal values that might occur is performed. For our case study model of the PWCS, the DED analysis was performed only on the State-flow Chart for Control System, for the reason of incompatibility of the other functional block diagram in the PWCS model. Necessary separation of DED analysis into two separate runs was required, because the SLDV tool does not support to run all the options for DED analysis at once. Before the actual analysis, the subsystem has to be modified and thus the SLDV treats a selected subsystem as an Atomic Unit. This is a general procedure that must be always proceeded in order to perform DED, Property proving or Test Case Generation for specific subsystems. During the DED analysis, the execution and the evaluation is fully automatic. The result of DED analysis is constrained only to searching for Dead Logic for the PWCS is shown in Figure 6.6 and Figure 6.7.

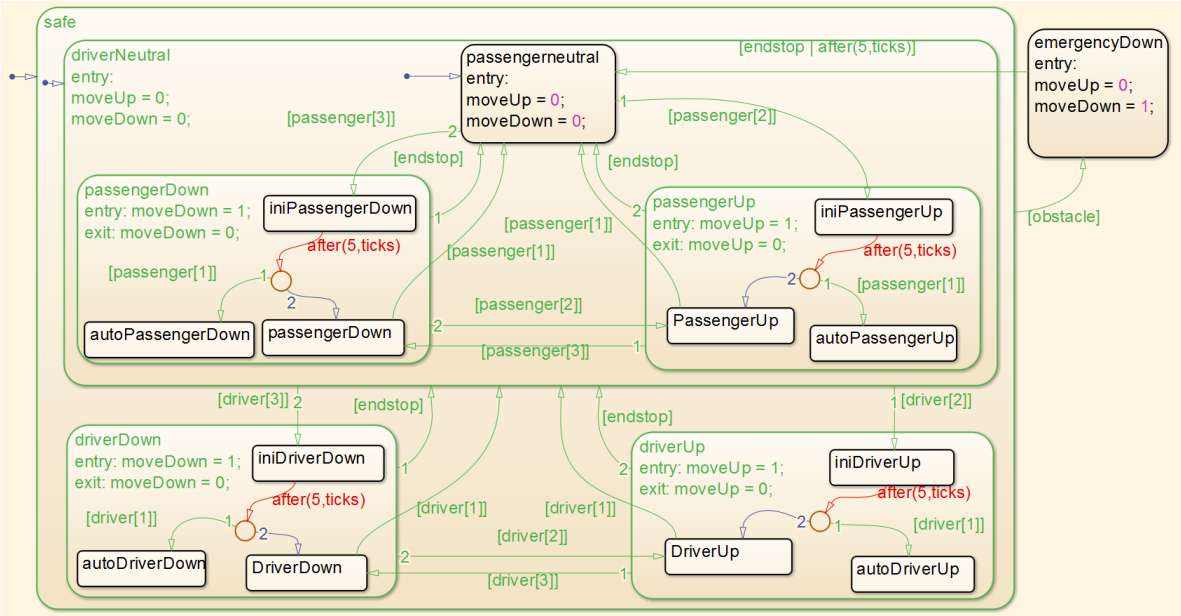

Figure 6.6: Visualization of results for the Dead Logic constraint in DED analysis

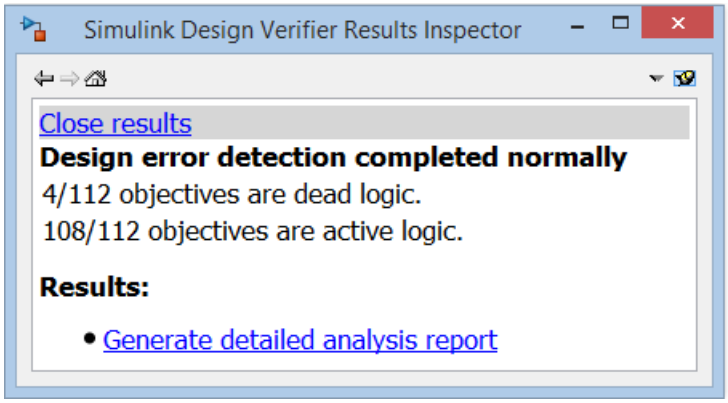

Figure 6.7: Results for the Dead Logic constraint in DED analysis

The SLDV tool allows the user to click on each of the transitions to see the conditions closely that caused the dead logic to occur. This helps to interactively track the dead logic and to possibly support the elimination of the dead logic. An example is shown in Figure 6.8.

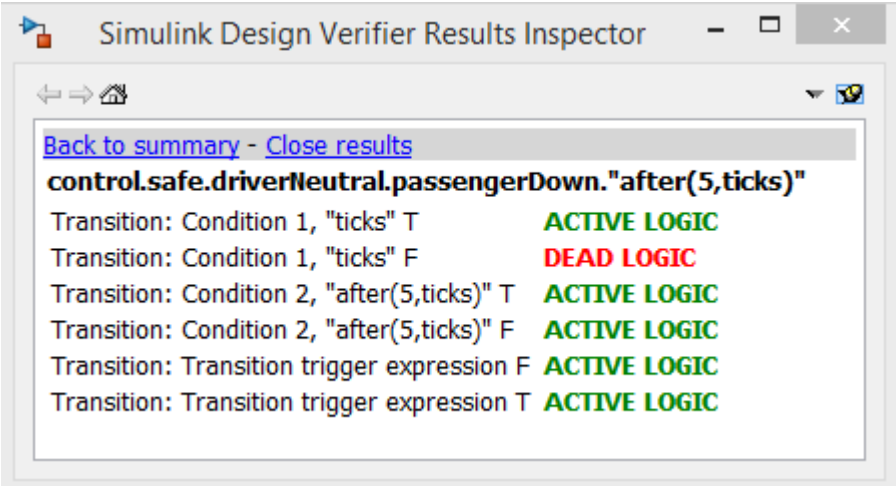

Figure 6.8: Results for the Dead Logic constraint in DED analysis

The objectives for the second part of DED analysis is focusing on detecting any *Integer Overflow*, *Division by zero*, *Intermediate Min/Max values* and *Out of bound array access*. The Figure 6.9 shows the results of the DED analysis, where the total number of eighteen objectives were successfully proven valid.

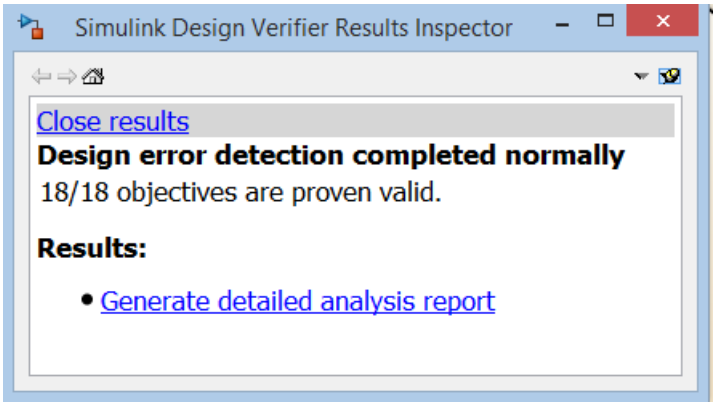

Figure 6.9: Results for the design error detection in DED analysis

Furthermore, in terms of the completion time of the analysis the first case took approximately 4 minutes to be able to gather the results for Dead Logic detection analysis. In the second part, the analysis took approximately 1 minute and 20 seconds due to the smaller number of objectives involved in DED analysis. The impact of the timing issue is further discussed in Section 6.5.

6.2 Property Proving

Property proving is the method of verifying the model design against the functional requirement specifications by specifying a property that needs to be proven for each functional requirement specification. In other words, the functional requirement specification is considered to be verified when the SLDV tool can prove a correctness of the design. A definition of a *property* can be stated as follows: "a requirement that you model in Simulink or Stateflow, or using Function blocks" [35]. Usually, a property is formed out of logical expressions that combines the input values or the range of the input values with the desired output values or the range of the output values. In case of property invalidation, a counterexample is then automatically generated to form a relevant basis for the analysis.

Deeply, in the SLDV tool the *Prover* Plug-In from the Prover Technology Inc., which serves as a model-checker for the verification of embedded system applications in Simulink [47], is incorporated. In more depth, the *Prover* Plug-In is considered to use techniques to solve Boolean Satisfiability problem (SAT) [45] that is consequently implied with "*Black-box*" procedures. According to the authors from [20] the techniques to solve SAT-based problem are **Bounded Model Checking (BMC)** combined with ***k*-Induction Rule**, where the aim is to perform search for a counterexample in the execution that is constrained by defined bounds of the length k. The bound k is usually increasing until the completeness threshold is reached [38]. That means to explore the state space with respect to the bound k. Moreover, the SAT-based BMC combined with k-Induction Rule can be formally represented in the following equation that is further converted into Conjunctive Normal Form (CNF) and solved [8]:

$$BMC(M,p,k) = I(s_0) \wedge \bigwedge_{i=0}^{k-1} \wedge\ T(s_i,s_{i+1}) \bigvee_{i=0}^{k} \neg p(s_i)$$

where M represents a model, p the property and k the length of a bound. In terms of k-Induction Rule, $I(s)$ and $T(s)$ encodes the initial states and transition relation for a system over sets of state variables s [19]. Two cases might be conducted in respect to this formula:

1. **Satisfiable case**: if true, then the defined property is **false** and *Prover* has found a counterexample of length k.

2. **Unsatisfiable case**: if true, no counterexample of length k is available and a proof of unsatisfiability can be obtained.

6.2.1 Modeling and Execution of Properties

Modeling properties in the SLDV tool are directly associated with the Simulink Design Verifier Library shown in Figure 6.10, where the most fundamental and critical blocks are located in *Objectives and Constraints* and are:

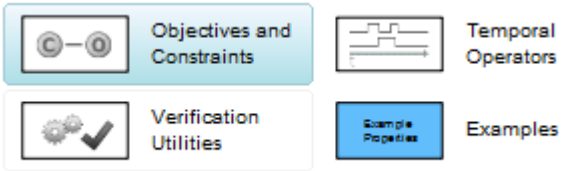

Figure 6.10: Simulink Design Verifier Library

1. **Proof Objective:** has the role to define the values of the signal to prove [35]. Without this block, the property proving is not possible. It is directly placed onto the signal branch within the model. The values of the signal might range from single TRUE/FALSE up to vector forms.

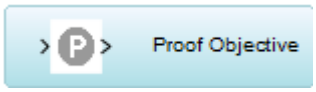

Figure 6.11: Constraints and Objectives: Proof Objective Block

2. **Assumption:** has the role of constraining the values of a signal during a proof [35]. Usually associated with relation to IF/THEN properties. The user assumes the specific behavior that is then proven or falsified with respect to defined the proof objective.

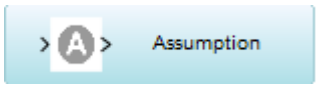

Figure 6.12: Constraints and Objectives: Assumption Block

The first mentioned block *Proof Objective* is usually applied to output signals and the *Assumption* block is usually applied with relation to constrain the input signals with respect to *Proof Objective* block. In other words, one can think of that as a black box, the input signals are constrained with the *Assumption* block and then the property is proven/falsified with the help of the *Proof Objective* block. Furthermore, the following section provides exact examples of how to model different categories of requirement specifications for our case study in PWCS. The first example, shown in Figure 6.13, represents the modeling *Fixed* requirement specification *number 5*:

"*Whenever move up is pressed and position greater than 0.3, obstacle should be detected*"

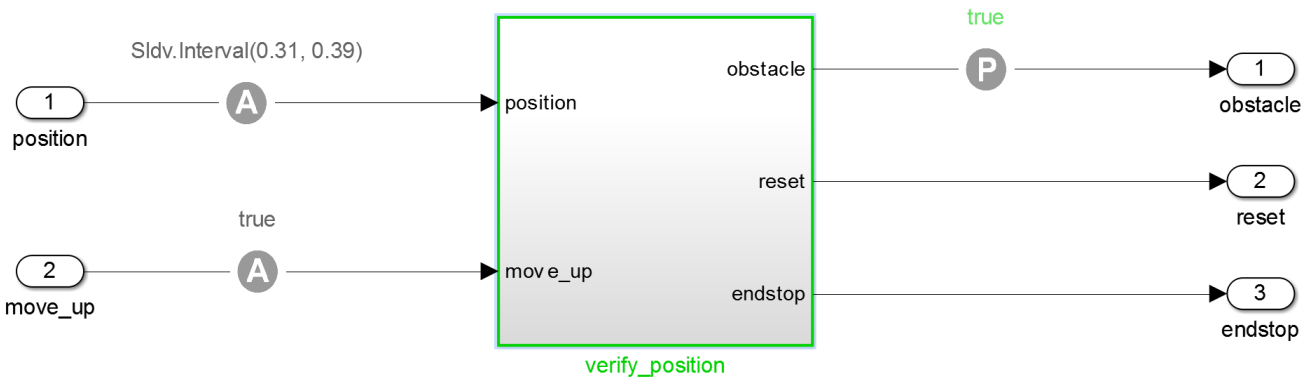

Figure 6.13: Modeling of the Fixed property: PROVEN VALID

The aim was to demonstrate the usage of *Objective and Constraints* blocks at the **integration level** in property proving. Directly placing the objective and constraints blocks into the model is the simplest way of modeling the properties. We assumed that the input range for the position is between 0.31 - 0.39 and at the same time the driver/passenger issues a command to move the window up. As an expected outcome, we want to prove whether the obstacle will be detected or not. The execution of property proving is fully automatic. After the execution, a window introduces the results of the Property Proving analysis with an additional option to generate the detailed report. More information about reporting will be given in the further section 6.2.2.

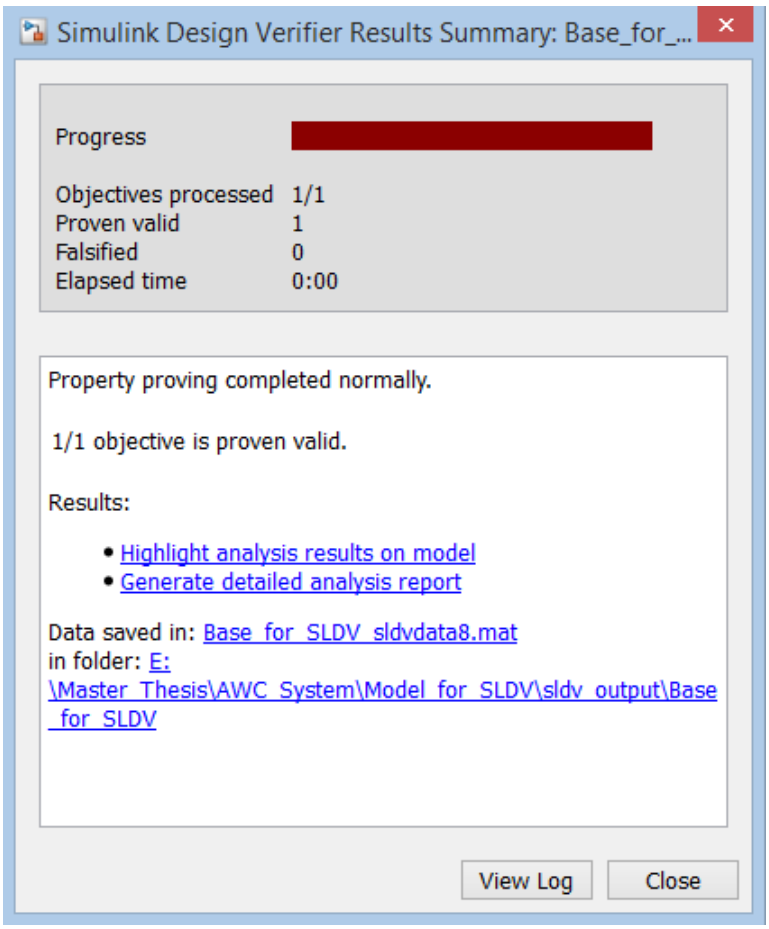

Figure 6.14: Property results dialog box window for PROVEN VALID case

The next example demonstrates the same property in case of property falsification. For this demonstration that input range for the position is set to be between 0.1 - 0.29. The following input range should disprove the defined property. The following Figure 6.15 shows the case of property falsification and Figure 6.16 shows the results of the property proving.

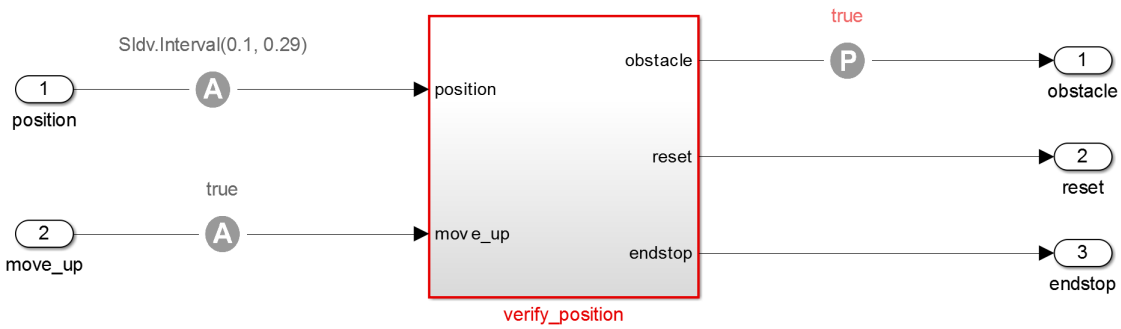

Figure 6.15: Modeling of the Fixed property: PROVEN FALSIFIED

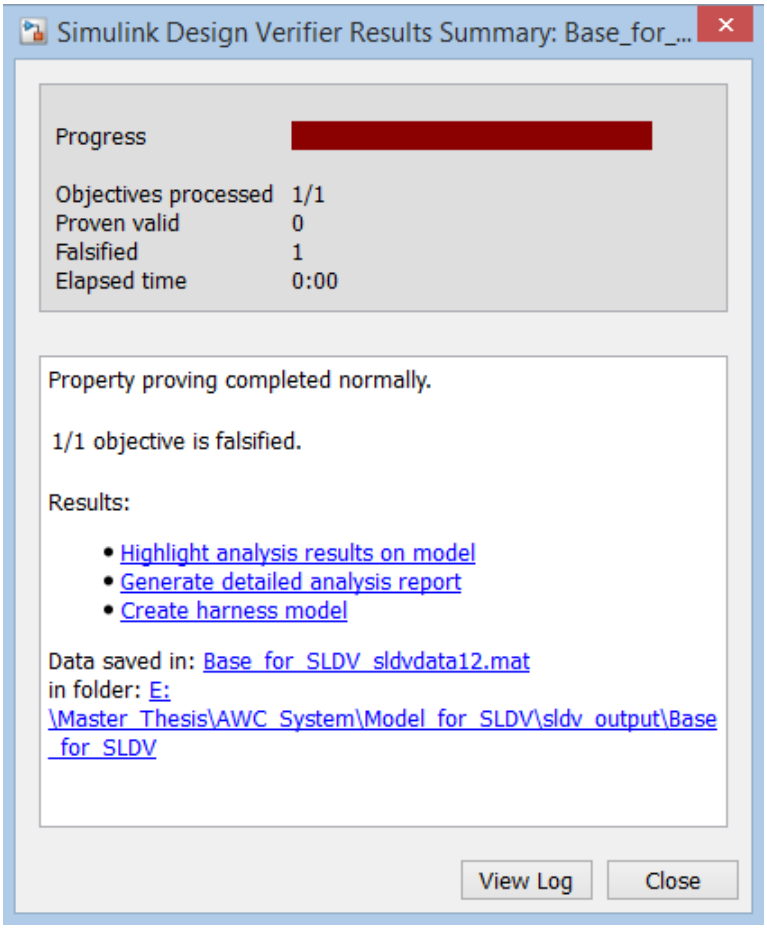

Figure 6.16: Property results dialog box window for PROVEN FALSIFIED case

The first example gave an insight of how to model simple properties by directly placing *Assumption* and *Proof Objectives* blocks into the model. This approach is the most useful to be used with relation to requirement specification defining the system functionality at the component level. It has to be mentioned that sometimes the difference between component and integration level of the system in terms of complexity might be neglected. It varies from case to case. Generally, component level specifications are assumed to be less complex in comparison to the ones defined at the integration/system level. The issue comes into account whenever the property that needs to be proven is more complex. In order to model such a complex property, the *Verification Subsystem* block used for property proving from Simulink Design Verifier Library should be used. The following Figure 6.17 shows an example for the general structure of the *Verification Subsystem* block.

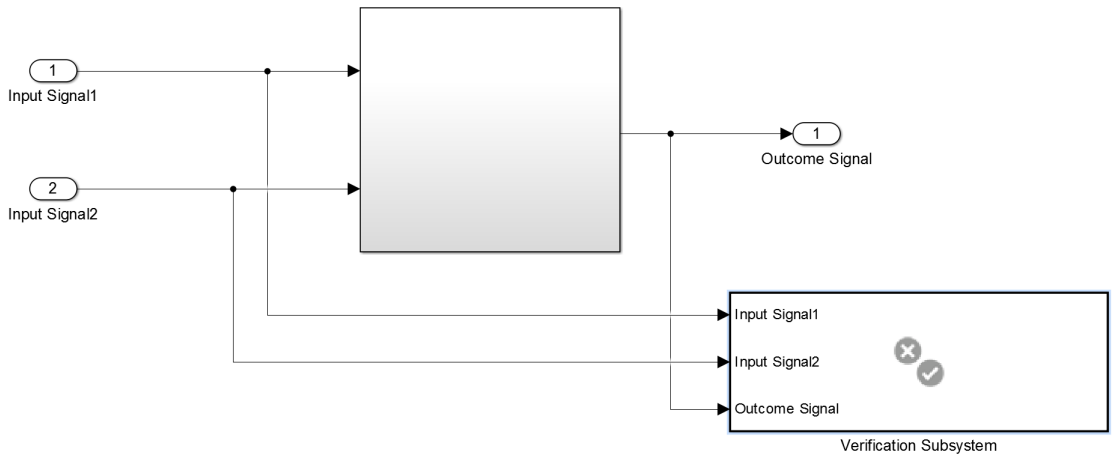

Figure 6.17: An example of the general structure for Verification Subsystem block for Property Proving

The **Assumption** and the **Expected Outcome** are usually modeled by combining several logic and mathematical operators together. The level of complexity of the property determines the complexity in terms of modeling. The Simulink block *Implies* is the general block for Model-Based Testing that serves to test whether the specified assumption triggers the expected outcome. Simply, whether the INPUT A implies the INPUT B. An example of the general implementation structure example of the Verification Subsystem is shown in Figure 6.18

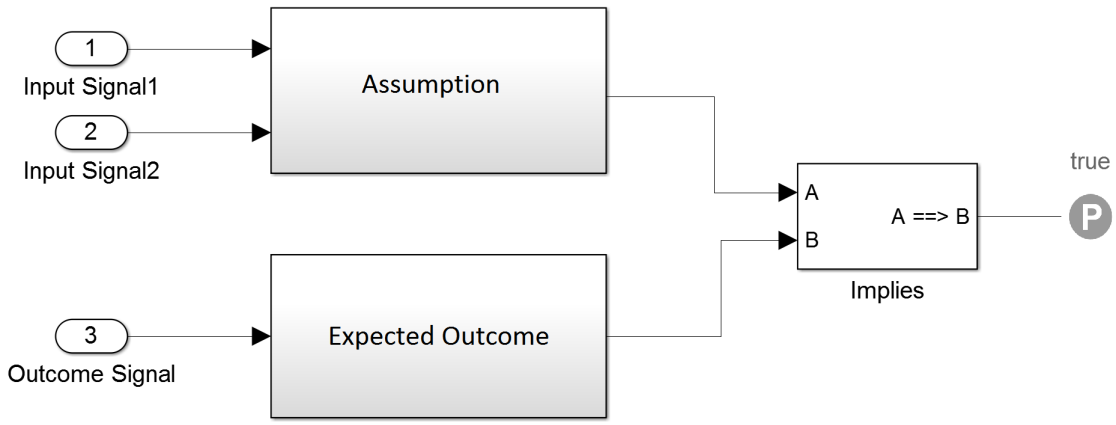

Figure 6.18: An example of the general implementation structure inside of the Verification Subsystem block for Property Proving

Another example that demonstrates the usage of the *Verification Subsystem* for proving more complex properties is introduced. The selected requirement specification *number 70* is categorized under *Time-variant* category and is defined as follows:

"*When an obstacle is present, the window should be lowered by approximately 10 cm (Command Down shall be given for 1 second)*"

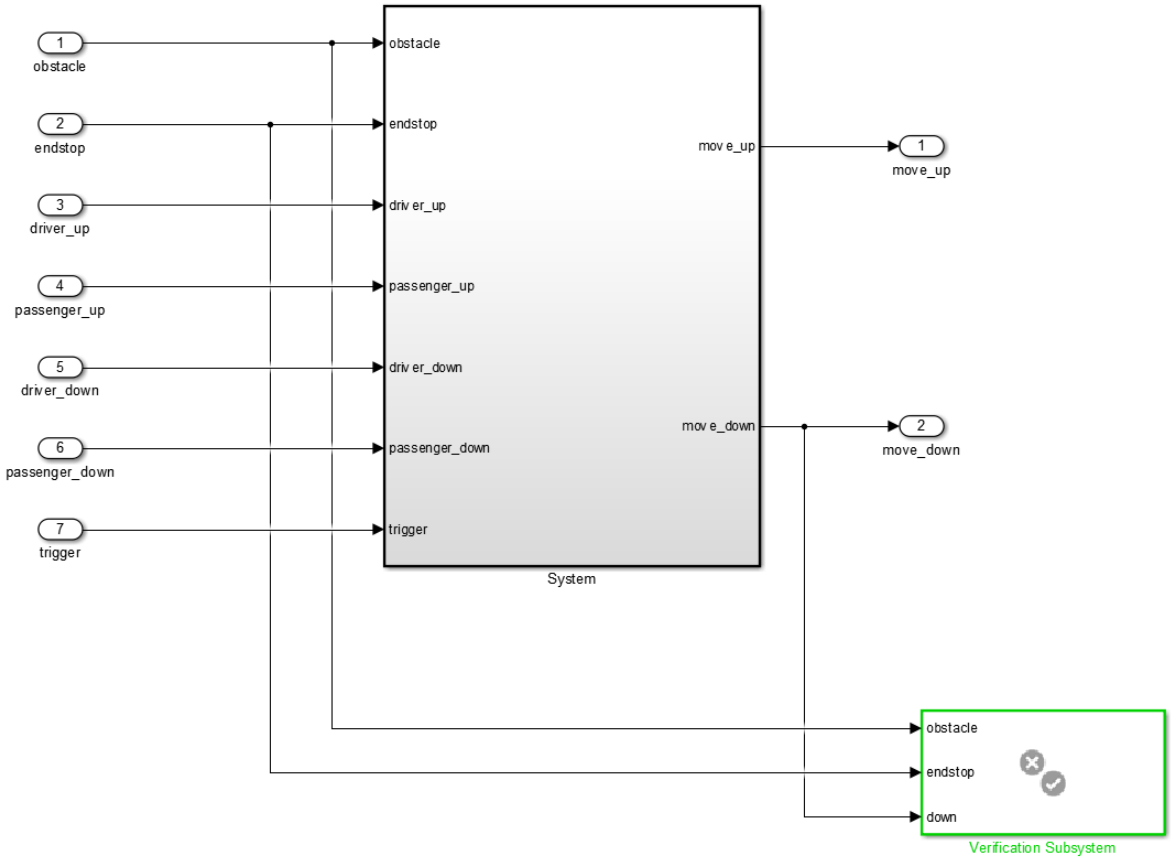

Figure 6.19: Modeling of property with use of Verification Sybsystem

Furthermore, this property represents the functional behavior of the *Control System* functional block that is treated as an atomic unit shown above. This functional block was implemented in State Flow Diagram and falls under the integration level category. A detailed implementation inside of the *Verification Subsystem* for this specific property is graphically shown in Figure 6.20.

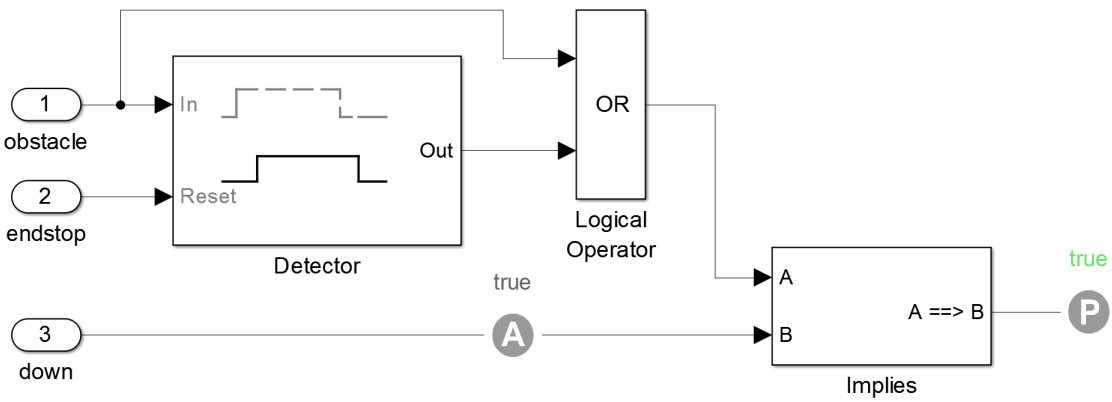

Figure 6.20: Modeling of the Time-variant property

The modeling of this property was done by using the operators: the *Logical Operator* and the *Temporal Operator*. For the temporal operator the block called *Detector* was selected. The purpose of this block is to detect a fixed number of consecutive time steps where the input is true. In the case above, the window should be lowered approximately by 10 cm, which in simulation environment is equal to one second. For this reason, *Detector* block was considered as the most suitable to model this property. However, it is necessary to mention that the block *Detector* is limited to detect only small sample times. Precisely, it was observed that in case of the detecting sample time >100, an overflow error occurs.

Additionally, an execution of above property with the use of Verification Subsystem is fully automated. As in the previous case, dialog box window summarizing the results from the testing emerges automatically. This dialog box for this property is shown in Figure 6.21.

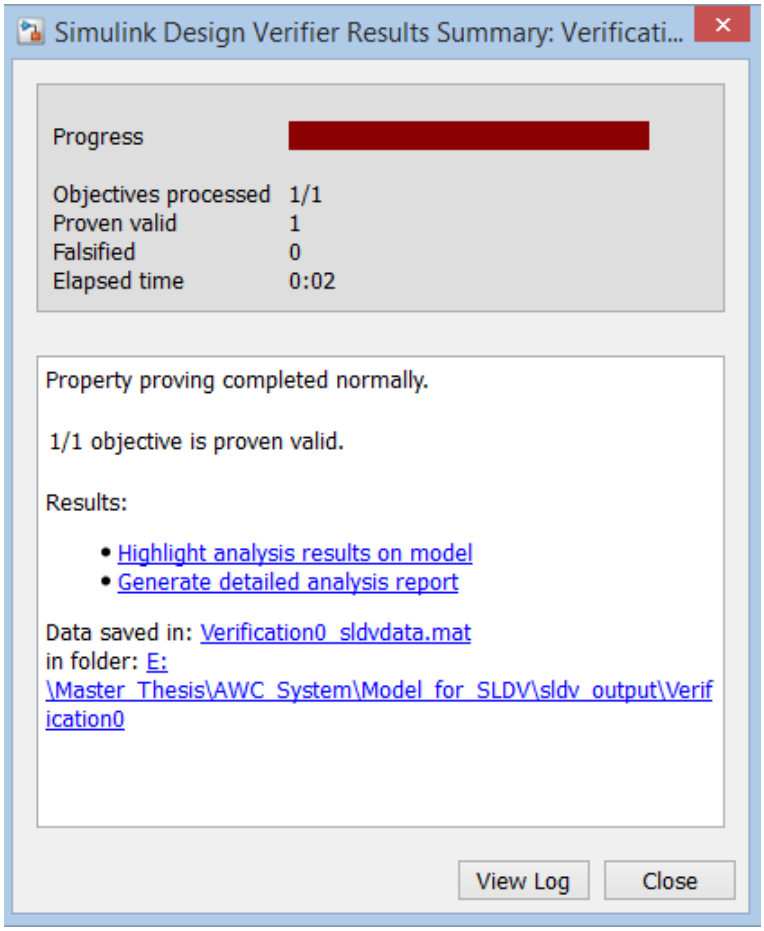

Figure 6.21: Property results dialog box window

6.2.2 Reporting for Property Proving

After the successful execution the detailed report can be automatically generated. The *Property Proving* report contains four sections:

1. **Summary:** describes the first and the most important information conducted from the Property Proving. This contains the information about the analyzed subsystem and an overall status of the execution. An example of such an information given by this chapter is shown in Figure 6.22.

2. **Analysis Information:** gives more precise information regarding to the model that was selected for testing purposes, the analysis options as well as the selected constraints.

Analysis Information

Model:	Base_for_SLDV_Control_System
Analyzed Subsystem:	Base_for_SLDV_Control_System/Verification Subsystem
Mode:	Property proving
Status:	Completed normally
Analysis Time:	2s

Objectives Status

Number of Objectives: 1
Objectives Proven Valid: 1

Figure 6.22: Example of graphical representation of the Summary chapter in SLDV report

3. **Proof Objectives Status:** a table containing detailed data about the proof objectives. An example of such a table is presented in Figure 6.23:

#	Type	Model Item	Description	Analysis Time (sec)	Counterexample
1	Proof objective	Verification Subsystem/Proof Objective	Objective: T	2	n/a

Figure 6.23: Example of the table summarizing Proof Objective Status

4. **Properties:** In the last chapter of the report, information regarding the verification subsystem properties is explained.

6.3 Test Case Generation

Another feature of the SLDV tool is to be able to automatically generate test vectors. By looking at the MBT taxonomy presented in section 2.3 *Test Generation* can be performed based on the six different *Test Selection Criteria*, due to the reason that the mission of the SLDV tool is to exploit the structure of the model, can be concluded that the SLDV tool uses **Structural Model Coverage Criteria (SMCC)**. The test cases are automatically generated to satisfy the SMCC, where the SMCC can be obtained out of three different coverage criterion:

1. **Decision Coverage(DC)**: conducts a percentage for all decision branches throughout the model by executing each of the possible branches from each decision point at least once [36]. The decision coverage can be calculated with help of the following formula:

   $$DC = \frac{\text{\# of decision branches reached}}{\text{The total\# of decision branches}} \times 100\%$$

   A relevant example for *Decision Coverage* would be a switch block in our case study for the PWCS that characterize a driver/passenger's inputs, *UP* or *DOWN*, then in order to fully reach 100% of decision coverage the total number of three test vectors must be generated.

2. **Condition Coverage (CC)**: conducts a percentage of the logical inputs and transitions that are covered by executing each logical guard (*true* and *false*) [32]. By exploiting only logical inputs and transitions, the usage of *Condition Coverage* is mainly related to Stateflow diagrams.

3. **Modified Condition/Decision Coverage (MCDC)**: This coverage criterion is the combination of both *Decision Coverage* and *Condition Coverage* where the aim is to ensure that a change in the outcome of each condition results in changing the outcome of the corresponding decision [36]. The MCDS criterion is usually used for covering highly critical software applications [33].

Creating a model for Test Case Generation is followed with the same SLDV Library section: *Objectives and Constraints* blocks. In comparison with Property Proving, where the blocks *Assumption* and *Proof Objectives* were used, the model for Test Case Generation uses the following blocks:

1. **Test Objective:** obtains signal values in Simulink Design Verifier test cases [35].

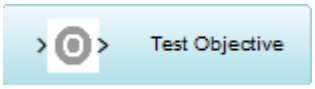

Figure 6.24: Constraints and Objectives: Test Objective Block

2. **Test Condition:** has the role to constrain signal values in SLDV test cases [35].

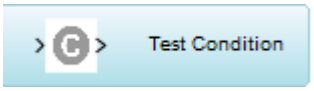

Figure 6.25: Constraints and Objectives: Test Condition Block

The most fundamental situation is whenever the user constraints directly the input/output signal with the *Test Condition* block in order to generate test cases to fully achieve the previously defined model coverage. To model more complex test cases a usage of the *Verification Subsystem* must be used. The next following Figure 6.26 and Figure 6.27 represent examples of the general modeling structure using the *Verification Subsystem* in order to perform Test Case Generation. The actual Test Case Generation execution and its analysis was not part of the objectives in this Thesis, for this reason only the generic model is shown and Test Case Generation will be not mentioned anymore.

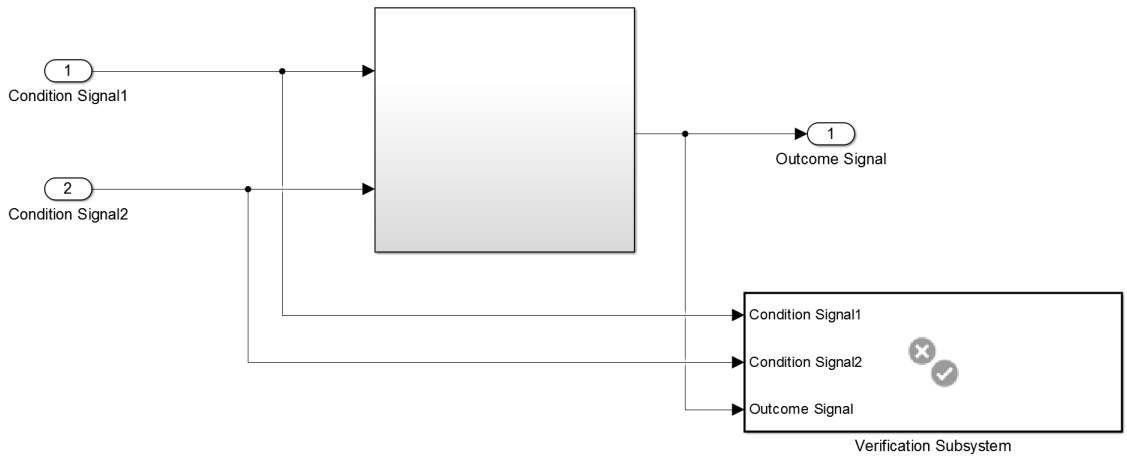

Figure 6.26: General structure for Verification Subsystem block for Test Case Generation

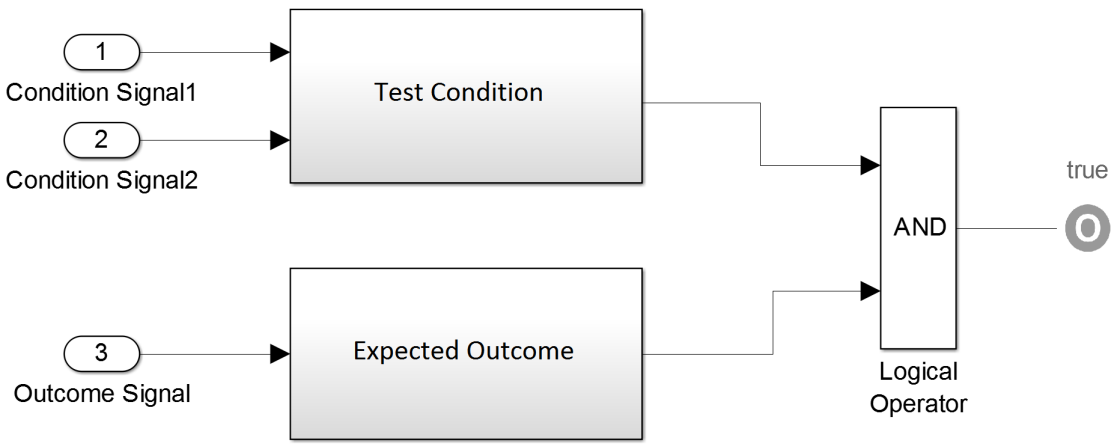

Figure 6.27: General implementation structure inside of the Verification Subsystem block for Test Case Generation

6.4 Summary and Analysis

The second part of the case study was focusing on the SLDV tool that has primary function to perform analysis from the structural point of view. The three key features: Design Error Detection, Property Proving and Test Case Generation were introduced and investigated. The main focus was on the Property Proving feature of the SLDV tool that serves as a verification method against the functional requirements. The analysis of how the different requirement specifications at the different levels of the system are modeled, executed and reported was distinguished. The data gathered from Property Proving resulted in proving the total number of 53/73 requirement specifications listed in the Table in the Appendix CD. The rest of the requirement specifications were not able to be modeled, due to the non-linear behavior, which is restricted to this tool. From the available results, the first milestone for this analysis: scalability could be concluded. The results show that more complex requirement specifications involve to use *Verification Subsystem*, which has a negative effect on scalability. In contrast, by splitting the system into separate smaller sub-sections the overall requirements complexity is reduced and the possibility of avoiding *Verification Subsystem* increases and thus the negative effect on scalability decreases. However, it does not solve the problem of scalability by going further into integration and system level, the scalability exponentially increases. Only by starting from the component level gives the user more confidence in his design and possibility of avoiding errors. The following table 6.1 represents relative rating of scalability based on the granularity of the system:

level of granularity:	**Impact on scalability:**
System level	high
Integration level	high
Component level	low

Table 6.1: Relative rating of scalability based on the granularity of the system

Furthermore, an increasing scalability and complexity of the modeled test cases for property proving inside of the verification subsystem results to increasing time of execution. Due to the simplicity of our case model, the differences were not significant and therefore no further specific analysis was performed. It can be concluded that SLDV is a suitable tool for efficient MBT at the component level, due to the reason of direct integration in the Simulink platform. However, for the systematic testing the tool could not be considered as a suitable one.

Another issue deals with the applicability of this approach. The potential of this MBT tool with respect to Model-Based Testing focusing on the structural perspective is large, due to the main reason that the tool is implemented directly within the Simulink platform and there is no need to use some kind of external platform for test execution. The tool can be used from both End-to-Bottom and Bottom-to-End perspectives, however in order to effectively use all the discussed features of the tool, second approach should be selected. That means to perform verification already during MBD. The major issues for applicability of the SLDV tool due to the tool limitations are further discussed in the section 6.5.

6.5 Limitations

The major limitation of the SLDV tool is that it supports only **Discrete Time Systems**, which means that the usage of solvers are strictly restricted to fixed-step solvers only. That means that all the functional blocks from Continuous Library in Simulink, for example: *Integrator*, *Transfer Fcn*, *Algebraic Loops* or commonly used functional blocks from Sources Library, for example: *Signal Generator* or *Repeating Sequence*, are not supported. The first possibility that the SLDV tool offers is to solve this with Block Replacements, which at some point might be useful, but generally leads potentially into errors by making the changes to the model. Another limitation deals with the computational time during DED analysis. For large models, the computational time might drastically increase.

Moreover, when the user performs many block replacements only to perform the MBT, the possibility of losing the track of the made changes increases. That might lead into future errors. The second possibility is to use the so called *stubbing technique*, which stubs the unsupported block for the use of testing. It must be considered that stubbing will make the block "invisible", which might cause an undesired behavior. From the industrial point of view, applying Model-Based testing techniques to a safety-critical system, where the precise data processing altogether with some filtering techniques is necessary, the SLDV tool is limited to its applicability and not suitable for such processes. Another limitation is the **ambiguity** of the results for Property Proving. For example, in case of the falsified property proving, the user does not know whether the problem has occurred in modeling of the property or it was caused by faulty behavior. The last relevant limitation observed is the **Sample Time Limit**, the Time-variant properties were modeled with the help of *Temporal Operators*, which with large sample times (more than 100) were causing an overflow error.

7 Conclusion

In this work, two different tools for using MBT within the Simulink platform were presented, analyzed and evaluated in order to support the major goal of this Thesis: the usefulness of MBT within Simulink platform. In the first MBT tool (**the TPT tool**) a detailed investigation of how the different requirement specifications categories are modeled, executed, evaluated and reported with respect to different levels of the system. Examples for each different requirement specifications category were introduced in order to demonstrate the theoretical background used in the selected MBT tool and show when and how to use it appropriately. It was shown how are both the *Fixed* and the *Time-variant* categories are modeled with the help of the TPTT and the DDT approach from different levels of the system perspective. From observations a conclusion stating that applicability of each of the approaches with respect to different requirement categories and different levels of the system was presented. Furthermore, a detailed analysis of the various evaluation techniques supported by the TPT tool were discussed in detail and how they were used to evaluate our defined requirement specification categories. From the case study of PWCS it could be confirmed that the TPT tool is the most suitable for systematic MBT activities focusing on tests on the integration and the system level. Although the testing at the component level is supported, the case study concluded that it is inefficient. The criteria for making the conclusion were based on the timing and effort needed for testing through the case study.

The second MBT tool that was under the scope of this analysis was **the SLDV tool**. There are three features of the tool: Design Error Detection Analysis, Property Proving and Test Case Generation were investigated. The special focus was on Property Proving technique that serves as a verification against the requirements and the major focus in this case was on functional requirements. The analysis showed that it is useful to perform component level testing with respect to scalability. That means that by performing component level testing, a scalability will be minimized and the user gains confidence in his MBD, which will result in less effort by performing MBT at the integration/system level. Thus the SLDV tool can be used directly by the user who is performing MBD in Simulink platform in order to verify the design against the requirements. In contrast, the TPT tool is better to be used for complex, systematic testing procedures, due to the strong applicability and availability of defining test cases that could then be easily executed at

the different platforms like SiL, PiL or HiL. In conclusion, even though both tools are suitable to perform testing activities on the component level, the SLDV showed better integrity with respect to the actual platform. Applying the MBT together with MBD within Simulink is an effective way of improving the verification process with respect to the system behavior. Both tools showed an extreme potential in terms of usefulness within Simulink platform. In order to proceed a reasonable effort to generate more safe and reliable software after MBD in Simulink, a combination of both tools together is suggested. As mentioned above, the SLDV tool to be used in order to detect early design errors within a model, starting already while modeling at the component level and the TPT tool to perform more complex and systematic testing at the integration/system level.

List of Figures

List of Tables

Bibliography

[1] *IEEE Recommended Practice for Software Requirements Specifications*. IEEE Computer Society, New York, USA, 1998.

[2] Purpose of reporting. Technical report, University of Wollongong, New South Wales, Australia, 2000 (accessed January 15, 2018). `https://unilearning.uow.edu.au/report/`.

[3] H-bridge. `https://en.wikipedia.org/wiki/H_bridge`, 2018 (accessed January 19, 2018).

[4] Systematic reviews: Creating inclusion/exclusion criteria. `http://guides.library.duq.edu/systematicreviews`, 2018 (accessed January 19, 2018).

[5] S. Ali and M. Sulyman. Applying model checking for verifying the functional requirements of a scania's vehicle control system. Master's thesis, School of Innovation, Design and Engineering Maelardalen University Vasteras, Gteborg, Sweden, 2012. `http://www.diva-portal.org/smash/get/diva2:558435/fulltext01`.

[6] D. Alparslan. Simulink design verifier-product presentation. `http://cmacs.cs.cmu.edu/presentations/verif_csystems/10_DenizhanAlparslan.pdf`, 2011.

[7] H. Altinger, F. Wotawa, and M. Schurius. Testing methods used in the automotive industry: Results from a survey. In *Proceedings of the 2014 Workshop on Joining AcadeMiA and Industry Contributions to Test Automation and Model-Based Testing*, JAMAICA 2014, pages 1–6, New York, NY, USA, 2014. ACM.

[8] N. Amla, X. Du, A. Kuehlmann, R. Kurshan, and K. McMillan. An analysis of sat-based model checking techniques in an industrial environment. In *Proceedings of 14th IFIP WG 10.5 Advanced Research Working Conference on Correct Hardware Design and Verification Methods (CHARME'05)*, pages 254 – 268. Cadence Design Systems, 2005.

[9] M. Babar and H. Zhang. Systematic literature reviews in software engineering: Preliminary results from interviews with researchers. In *2009 3rd International Symposium on Empirical Software Engineering and Measurement*, pages 346–355, Ireland, 2009.

[10] J. Babic. *Model-based Approach To Real-Time Embedded Control Systems Development With Legacy Components Integration.* PhD thesis, University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, Croatia, 2014. `https://bib.irb.hr/datoteka/728042.PhD_JB_print.pdf`.

[11] M. Bagheri. Analysis of model-based testing methods for embedded systems. Master's thesis, Uppsala Universitet, 2016. `https://uu.diva-portal.org/smash/get/diva2:952248/FULLTEXT01.pdf`.

[12] T. Bienmueller, U. Brockmeyer, and G. Sandmann. Automatic validation of simulink/stateflow models - formal verification of safety-critical requirements. In *International Automotive Conference*, Stutgart, Germany, 2004.

[13] F. Bouquet, F. Peureux, and F. Ambert. *Model-Based Testing for Functional and Security Test Generation.* Spingler e-ISBN 978-3-319-10082-1, France, 2014.

[14] E. Bringmann and A. Krmer. Model-based testing of automotive systems. In *2008 1st International Conference on Software Testing, Verification, and Validation*, pages 485–493, April 2008.

[15] K. Briski, P. Chitale, V. Hamilton, A. Pratt, B. Starr, J. Veroulis, and B. Villard. Minimizing code defects to improve quality and lower development costs. Technical report, IBM, 2008.

[16] A. Cavalcanti and A. Simao. Fault-based testing for refinement in csp. *IFIP International Federation for Information Processing 2017.*

[17] E. Clarke and B. Schingloff. Model checking. In *Handbook of automated reasoning*, 2001.

[18] R. Cleaveland. Model-based verification of automotive controllers, 2012.

[19] A. Donaldson, L. Haller, D. Korening, and P. Ruemmer. Software verification using k-induction. In *SAS'11 Proceedings of the 18th international conference on Static analysis*, pages 351 – 368, 2011.

[20] J.-F. Etienne, S. Fechter, and E. Juppeaux. *Using Simulinkg Design Verifier for proving behavioral properties on comples safety critical system in the ground transporation domain.* Springler e-ISBN 978-3-642-15654-0, Paris, France, 2010.

[21] M. Glinz. On non-functional requirements. In *15th IEEE International Requirements Engineering Conference (RE 2007)*, pages 21–26, 2007.

[22] I. Jacobson, G. Booch, and J. Rumbaugh. *The Unified Software Development Process.* ADDISON WESLEY e-ISBN 9780321822000, 1999.

[23] S. Kitchenham and S. Charter. Guidelines for performing systematic literature reviews in software engineering, 2007. `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.117.471`.

[24] C. Kleijn. Introduction to hardware-in-the-loop simulation. Technical report, ControlLab, 2015. `http://www.hil-simulation.com/images/stories/Documents/Introduction`.

[25] A. Kramer, B. Legeard, and R. Binder. Model-based testing user survey: Results. Technical report, Smartesting and Sepp.med, 2017.

[26] A. Kull. *Model-Based Testing of Reactive Systems.* PhD thesis, Faculty of Information Technology, Tallinn University of Technology, Finnland, 2009. `https://www.researchgate.net/publication/234117217_Model-Based_Testing_of_Reactive_Systems`.

[27] K. Lamberg, M. Beine, M. Eschmann, O. Rainer, M. Conrad, and I. Fey. Model-based testing of embedded automotive software using Mtest. Technical report, SAE 2004 Word Congress and Exhibition, 2004.

[28] E. Lehmann. Time partition testing: A method for testing dynamic functional behaviour, 2000. `http://www.systematic-testing.com/documents/test2000.pdf`.

[29] E. Lehmann. *Time Partition Testing: Systematischer Test des kontinuierlichen Verhaltens von eingebetteten Systemen.* PhD thesis, Technische Universität Berlin, Lehrstuhl für Elektrotechnik und Informatik, Berlin, Germany, 2004. `http://time-partition-testing.de/downloads/papers/ELehmannDissertation.pdf`.

[30] F. Leitner. Evaluation of the matlab simulink design verifier versus the model checker spin. Technical report, Department of Computer and Informaition Science, University of Konstanz, 2008.

[31] M. Li and R. Kumar. Automated test generation and error localisation for simulink/stateflow modelled systems using extended automata. *IET Cyber-Physical Systems: Theory & Applications*, 2016.

[32] M. Liliegard and V. Nilsson. Model-based testing with simulink design verifier. Master's thesis, Department of Signals and Systems, Chalmers University of Technology, Goteborg, Sweden, 2014. `http://publications.lib.chalmers.se/records/fulltext/200464/200464.pdf`.

[33] S. Liu, T. Maibaum, and K. Araki. Formal methods and software engineering. In *10th International Conference on Formal Engineering Methods ICFEM 2008*, Japan, 2008.

[34] T. Majchrzak. *Improving Software Testing - 2012*. Springler e-ISBN 9783642274640, 2012.

[35] MathWorks. *Simulink Design Verifier Guide*, 2014. `http://files.matlabsite.com/docs/books/matlab-docs/simulink_design_verifier_sldv_ug_r2014b.pdf`.

[36] R. Matinnejad. *Automated Testing of Simulink/Stateflow Models in the Automotive Domain*. PhD thesis, The Faculty of Sciences, Technology and Communication, Universite du Luxembourg, Luxembourg, 2016. `http://orbilu.uni.lu/handle/10993/28688`.

[37] P. Mosterman, S. Engell, and J. Sztipanovits. Computer-automated multiparadigm modeling in control systems technology. *IEE Transactions on Control Systems Technology*, 12(2):223–234, 2004.

[38] L. Moura, R. Herald, and M. Sorea. Bounded model checking and induction from refutation to verification. In *Proceedings of the 15th International Conference on Computer Aided Verification, CAV 2003, volume 2725 of Lecture Notes in Computer Science*, pages 14–26. SRI International Computer Science Laboratory, 2003.

[39] K. Munakata, S. Tokumoto, and U. Tadahiro. Model-based test case generation using symbolic execution. In *Proceedings of the 2013 International Workshop on Joining Academia and Industry Contributions to testing Automation*, Japan, 2013.

[40] PHYTEC Metechnik GmbH, Meinz, Germany. *phyCORE-MPC555*, 2000. `http://www.phytec.de/fileadmin/user_upload/downloads/Manuals/L-524E.pdf`.

[41] G. PikeTec. TPT - model-based testing of embedded control systems. `http://www.piketec.com/de/2/tpt.html`.

[42] PikeTec GmbH. *PikeTec User Guide*.

[43] S. Rayadurgam and M. Himdahl. Coverage based test-case generation using model checkers. In *Proceedings. Eighth Annual IEEE International Conference and Workshop On the Engineering of Computer-Based Systems-ECBS 2001*, pages 83–91, 2001.

[44] H. Robinson. Graph theory techniques in model-based testing. In *International Conference in Model-Base Testing*, 1999.

[45] J. M. Rushby, N. Shankar, G. Hamon, L. Erkök, J. Matthews, D. Sheridan, D. Cok, P. Bokor, S. Shukla, A. Pataricza, N. Suri, A. Gargantini, J. Kirschenbaum, H. K. Harton, and M. Sitaraman. Afm'08: Third workshop on automated formal methods 14 july 2008 princeton, new jersey. Princeton, New Jersey, 2008. SRI International Computer Science Laboratory.

[46] M. Staron. *Automotive Software Architectures.* Springler ISBN 978-3-319-58610-6, Germany, 2017.

[47] B. Steffen. *Transactions on Foundations for Mastering change.* Spingler ISBN 978-3-319-46508-1, 2016.

[48] M. Utting and B. Legeard. *Practical Model-Based Testing: A Tools Approach.* Elsevier Science and Technology Books ISBN-13: 9780123725011, 2010.

[49] M. Utting, A. Pretchner, and B. Legeard. A taxonomy of model-based testing. *Software Testing, Verification and Reliability*, 2006.

[50] M. Weber and J. Weibrod. Requirements engineering in automotive development-experiences and challenges. In *Proceedings IEEE Joint International Conference on Requirements Engineering*, pages 331–340, 2002.

[51] L. Williams. *A (Partial) Introduction to Software Engineering Practices and Methods (Seventh Edition).* NCSU CSC326 Course Pack, Raleigh, USA, 2011.

[52] Y. Xu. Towards DO-178C compativle tool design. Master's thesis, Iowa State University, Iowa, USA, 2015. `https://lib.dr.iastate.edu/cgi/viewcontent.cgi?article=5747&context=etd`.

[53] J. Zander, I. Schieferdecker, and P. Mosterman. *Model-based Testing for Embedded Systems.* CRC Press ISBN 9781138076457, 2011.

[54] J. Zander-Nowicka. *Model-based Testing of Real-Time Embedded Systems in the Automotive Domain.* PhD thesis, Technische Universität Berlin, Lehrstuhl für Elektrotechnik und Informatik, Berlin, Germany, 2009. `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.427.6806&rep=rep1&type=pdf`.

Eidesstattliche Versicherung

_____ _____

Name, Vorname Matrikelnummer

Ich versichere hiermit an Eides statt, dass ich die vorliegende Bachelorarbeit/Masterarbeit* mit dem Titel

_____

_____

_____

selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäe Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

_____ _____

Ort, Datum Unterschrift

* Nichtzutreffendes bitte streichen

Belehrung:

Wer vorsätzlich gegen eine die Täuschung über Prüfungsleistungen betreffende Regelung einer Hochschulprüfungsordnung verstöt, handelt ordnungswidrig. Die Ordnungswidrigkeit kann mit einer Geldbue von bis zu 50.000,00 geahndet werden. Zuständige Verwaltungsbehörde fr die Verfolgung und Ahndung von Ordnungswidrigkeiten ist der Kanzler/die Kanzlerin der Technischen Universität Dortmund. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Tüschungsversuches kann der Prfling zudem exmatrikuliert werden. (63 Abs. 5 Hochschulgesetz - HG -)

Die Abgabe einer falschen Versicherung an Eides statt wird mit Freiheitsstrafe bis zu 3 Jahren oder mit Geldstrafe bestraft.

Die Technische Universität Dortmund wird gfls. elektronische Vergleichswerkzeuge (wie z.B. die Software turnitin) zur Überprüfung von Ordnungswidrigkeiten in Prüfungsverfahren nutzen.

Die oben stehende Belehrung habe ich zur Kenntnis genommen:

_____ _____

Ort, Datum Unterschrift