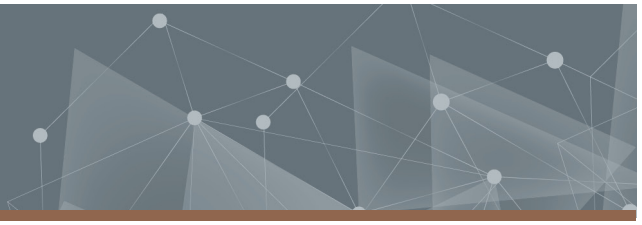
CHALMERS
UNIVERSITY OF TECHNOLOGY



Automated Testing of Hardware-in-the-Loop Systems for Electrified Vehicles

Master's thesis in System, Control and Mechatronics

Karl Fransson and Manuel Alejo Gago

Department of Electrical Engineering

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2022
www.chalmers.se

Master's thesis 2022

Automated Testing of Hardware-in-the-Loop Systems for Electrified Vehicles

Karl Fransson and Manuel Alejo Gago

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2022

Automated Testing of Hardware-in-the-Loop Systems for Electrified Vehicles
Karl Fransson and Manuel Alejo Gago

Supervisor: Johan Lidén Eddeland, Volvo Cars
Examiner: Knut Åkesson, Electrical Engineering

Master's Thesis 2022
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2022

Abstract

The increase of interest in electrical vehicles has been rapidly increasing in recent years. Modern day vehicles are equipped with more complex systems to meet with technological standards and safety aspects. A part of those are autonomous driving functionalities to improve the safety of the vehicle as well as the people using it. The automation of the vehicles as the end-product goes hand-in-hand with the development of the vehicle down to component level.
In this thesis we present a framework on how automated test generation of Hardware-In-the-Loop systems for electrified vehicles can be performed. The utility of Hardware-In-the-Loop systems comes in stages after simulations in the other test integration levels Software-In-the-Loop and Model-In-the-Loop, with a key difference being the presence of hardware which is configured and tested upon as test objects.
By traditional means, the tests are running in a test environment so the test objects are emulated as if running in the real-world environment in the use of the end product. Interfacing with the test object can be done by manually actuating and alter the inputs in real-time to produce test results. Besides the manual means it is possible to run automated tests where the interface use test cases with instructions on how the test is performed. When test cases are available they can be used for the future and run on several systems. The issue is the creation and maintainability of test cases since that is performed manually by test engineers. Resources which could otherwise be spent on other areas and time spent by the test engineers is spent on implementing new test cases as new functionalities while the need for testing is still growing. To complement this way of testing we provide a framework for automated test case generation and execution. The primary method of performing this automation is called falsification. With falsification it is possible to find counterexamples of specifications where requirements are no longer met. Using falsification, it is possible to automate the process of finding new specifications to be used in test cases, thus creating new ones. With the generation of test cases it is possible to execute the test on the Hardware-In-the-Loop system to produce test results and use the test results as input data and interface that to a falsification problem and create new test cases. The framework presented in the thesis makes use of this and interface the data to perform both test generation and execution, providing another layer of automation to testing.

Keywords: Cyber-Physical Systems, Hybrid systems, Temporal logic, Falsification, Software-in-the-Loop, Hardware-in-the-Loop, Automated test generation.

Acknowledgements

We would like to thank our supervisor Johan Lidén Eddeland who presented us with the idea of this project and worked with us throughout the project. We would also like to thank Volvo Cars for providing us with the resources and equipment to carry out this project.

Karl Fransson & Manuel Alejo Gago, Gothenburg, June 2022

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

ACC	Adaptive Cruise Control
ADAS	Advanced Driver-Assistance Systems
BECM	Battery Electronic Control Module
CI	Compute-Intensive
CPS	Cyber-Physical System
CPU	Central Processing Unit
ECU	Electronic Control Unit/Engine Control Unit
EDM	Electric Drive Module
HW	Hardware
HIL	Hardware-In-the-Loop
I/O	Input/Output
LTL	Linear Temporal Logic
MBD	Model Based Development
NSF	National Science Foundation
PWM	Pulse-Width Modulation
RAM	Random-Access Memory
SIL	Software-In-the-Loop
STL	Signal Temporal Logic
SUT	System Under Test
SW	Software

Contents

List of Figures

List of Tables

1
Introduction

Cyber-Physical Systems (CPSs) are systems that integrate networks, computations and physical processes. They can be simple, e.g., a heater, or include multiple components, e.g., autonomous automotive systems or medical monitoring. This novel understanding of systems has resulted in several challenges, including their validation and verification. This master's thesis tackles the problem of verifying and validating CPSs in an industrial environment. In particular, the challenge of implementing automated testing in the Hardware-in-the-loop (HIL) environment to reduce additional work for engineers and increase confidence in the final components. Such complex systems are usually developed using the so-called Model-Based Development (MBD) approach. It is a common design paradigm that uses mathematical models to simulate the behavior of the CPSs. In addition, to perform tests on these models there is a set of activities contained under the name of Model-Based Testing (MBT). One of these activities is Hardware-in-the-Loop, which allows evaluating complex systems in real-time, combining real hardware with emulated components. Since HIL does not require testing on the final assembled product, it helps to reduce risks. For example, HIL testing minimizes late detection of errors, which could lead to an overhaul of the entire system and the discovery of malfunction at late stages that could drive to failures that destroy expensive equipment.
Among all different techniques for testing CPSs, this master's thesis considers falsification, which consists of validating a system by monitoring its response to input signals. More precisely, the goal of falsification is finding inputs signals which prove the system does not fulfil a certain specification. The implementation of this technique is performed using a toolbox called Breach [1]. In addition, this tool generates automated tests, as it computes, investigates, and selects input parameters automatically, without the collaboration of test engineers.

1.1 Cyber-physical systems

The term Cyber-Physical System was first coined at the National Science Foundation (NSF) around 2006 [2]. Since then, the interest in its analysis and study has increased, a fact that is reflected in the rise of funds awarded by the scientific institutions throughout the world to research projects related to CPSs in recent years [3]. These systems can boost progress in many different fields, which could generate big economic impacts [4]. For example, transportation systems could benefit from better embedded intelligence in cars and in this way improve safety and efficiency. Another example is that the power grid could integrate distributed micro

power generation precisely and safely. However, despite the progress made in the last years, the field of researching CPSs is still a discipline with a great amount to be explored. Furthermore, several recent trends have increased interest in CPSs even further: the proliferation of low-cost and increased-capability sensors; the availability of low-cost, high-capacity and small-form-factor computing devices; or the wireless communication revolution.
Several definitions of CPSs have been proposed by the scientific community. In [5] CPSs are defined as physical and engineered systems whose operations are monitored, coordinated, controlled, and integrated by a computing and communication core. Another work [4] describes them as the integration of computational with physical processes, monitored and controlled by embedded computers and networks. Liu et al. state in [6] that they are multidisciplinary systems to conduct feedback control on widely distributed embedded systems by the combination of computation, communication, and control technologies. Gathering all formal definitions together, we can conclude that CPSs are complex and multidisciplinary systems that integrate the dynamics of the physical processes with those of the software and communication.
To better understand what a CPS is, the system's components and structure are introduced. As seen in Figure 1.1, a CPS can be divided into three different layers: the physical layer, the information layer and the cyber layer. The physical level represents the physical phenomena to be monitored. The information layer refers to the communication network and other devices. It is primarily in charge of the transmission and collecting of the data and is responsible for linking the physical and cyber layers. Last, the cyber level exemplifies the embedded instruments, sensors, and actuators that deal with the acquisition and sampling of data.

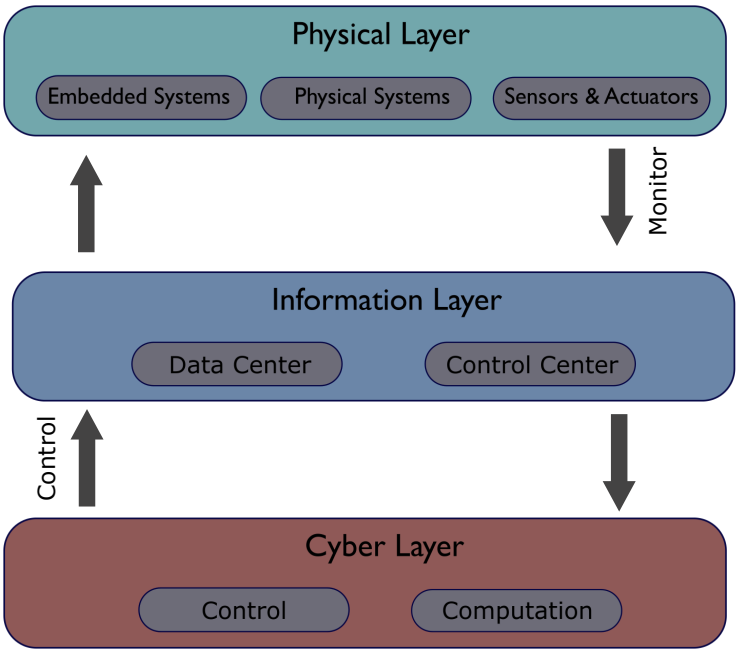


Figure 1.1: Representation of the components and structure of a CPS. Figure reproduced from [7].

1.1.1 Characteristics

CPSs have some features that differentiate them from both traditional embedded and real-time systems. The most important of these features are as follows:

1. **Importance of physical systems**: as opposed to traditional embedded systems, physical systems play a key role in CPSs. Tasks regarding hardware design and size, energy management, or system testing should be taken deeply into account. Some models have been discussed within the related literature. A methodology for automatically abstracting models of CPSs can be found in [3]. Also, this work shows how analytical abstractions from physical systems can be converted into executable simulation code.
2. **Information system as the integrating element**: to join the physical and the cyber systems an information system is needed. In this way, the information in the physical environment can be transformed into the rules and models of software systems. The network must satisfy two basic requirements to facilitate this integration and ensure proper results. First, data transmission must be secure against possible network attacks. Second, this information system must be capable of transmitting data in real-time.
3. **Integration of heterogeneous systems**: as mentioned before, CPSs are heterogeneous distributed systems with high integration and interaction of information and physical systems. To achieve this integration, cyber capability in every physical component needs to be installed.
4. **Real-time capability**: CPSs transmit collected information or instructions in real-time, so that the requirements of task processing are met.

1.1.2 Challenges

CPSs have and will continue to transform how we interact with the world around us. This new paradigm has resulted in progress in multiple fields and its consequent benefits. However, all this does not come for free. Even though great progress has been done in recent years, CPSs still have some challenges. According to [6], the main difficulty in the design of CPSs lies in the lack of a theoretical framework of network and physical resources. Unlike embedded systems, where there is a great deal of experience and know-how, the modeling of physical systems and networks is still at an early stage of development. This leads to problems such as the inability to transform the requirements of physical systems or the overlooks of fundamental aspects of computer engineering. A summary of the most important challenges faced by CPSs is presented below from [2, 3, 5, 6, 8]:

1. **Abstraction and architectures**: for the correct design of CPSs it is necessary to accurately capture the physical properties through programming abstractions. Also new architectures must emerge to integrate physical systems, cyber systems and networks . These must ensure the integration and interoperability of the heterogeneous systems that make up CPSs.
2. **Hybrid systems and control**: a hybrid automaton is needed to describe through a mathematical model, systems where digital computational processes interact with analog physical processes in order to perform feedback control. newpage

3. **Robustness, reliability, safety, and security**: it is a key challenge because uncertainties in the physical world, cyber-attacks, and errors make it necessary to pay close attention to these requirements.
4. **Verification and validation**: the gap between formal methods and testing needs to be bridged. Compositional verification and testing methods that explore the heterogeneous nature of CPS models are essential.

As will be discussed later in Section 1.3, CPSs cannot be validated and verified using traditional formal methods. Instead, testing needs to be used. This master's thesis focuses on taking a step further on the path of providing a tool to overcome the challenge of verifying and validating CPSs.

1.2 Model-based development

Model-based development is a mathematical method based on models that represent the dynamic behavior of the system and its main goal is to provide developers with a framework which establishes a series of steps to follow for designing, analyzing, verifying, and validating dynamic systems [9, 10]. As systems become more and more complex involving both hardware and software, as with CPSs, engineers need to separate the development and testing phases from the hardware availability, since dependence on the hardware may lead to delays and difficulties. Model-Based Development is one approach to overcome this issue.
There are several advantages to using MBD. Firstly, equation-based models and simulations of these models allow engineers to an early understanding in the design phase. Moreover, MBD also reduces time to market through component reuse and reduces costs by simulating and testing systems before implementation [11].
Jensen et al. detail in [10] how the MBD process for CPSs can be broken up into ten codependent steps. Staying within these steps, which means follow them carefully and correctly, it is possible to successfully develop and design a CPS and satisfactorily track the evolution of the process. It should be kept in mind that it is a iterative methodology, which makes a continuous review of the phases necessary until the requirements are fulfilled.

1.2.1 Model-based testing

The term Model-based testing has different meanings, depending on the scope of application and use. Generally speaking, it can be described as all testing activities in the context of Model-based development projects [12]. More precisely, MBT is a variant of testing, which is based on models that simulate the behavior of the System Under Test (SUT) and/or the behavior of the environment [13]. Its main activities are focused on building the model, defining test specifications, generating tests, and executing tests.
The advantages of following this methodology are numerous [10, 12, 13]. The most interesting ones for the purpose of the project are discussed next. As already mentioned in Section 1.2, MBD, and hence MBT, enables engineers to understand and detect failures at an early stage. In addition, one of its most attractive features is the capability for automated test generation. This is very useful in the case of systems

as complex as the CPSs considered here, where an iterative process is necessary to ensure that the required specifications are met. Finally, as will be discussed in Section 1.3, this approach also allows engineers to operate in virtual environments.

1.3 Verification and validation

There are many analysis techniques that can be classified into two main groups: formal verification and testing [9]. A short insight into how to verify CPSs, comparing formal verification and testing, and also the approach followed to implement the project is given below.

1.3.1 Formal verification

According to [9], formal verification provides formal proof of the correctness of a system for a set of parameters and inputs. In other words, it is the procedure to prove that a system is safe to all disturbances [14]. Formal methods are the techniques that underlie this approach. They are mathematically based techniques, usually assisted by tools, that help to describe the properties of a system and verify them. Among its advantages are the ability to analyze and verify the models under study at any part of the development cycle and the ability to reveal inconsistencies, ambiguities, and bugs. These techniques have proven to be useful for validating CPSs under very specific conditions, as in [15]. However, formal verification in CPS is very complex. This is because it is based on the calculation of the reachable space of a hybrid automata, which in the case of very large CPSs may not converge even under very specific constraints, as in [16]. This is, there is no terminating algorithm that can answer whether a CPS ever enters a set of bad states.

1.3.2 Testing

When the complexity of systems, such as those covered in this master's thesis, makes it impossible to use formal verification techniques, testing becomes the main alternative. According to [13], the objective of testing is to verify that the actual and expected behavior differ, or to increase confidence that they do not. Its goal is failure detection. However, this way of validating systems has two limitations that it is good to keep in mind [9]. On the one hand, the conditions under which testing is performed may not reproduce the actual conditions under which the system will be used once implemented. On the other hand, due to the underlying nature of testing, no matter how many times a system is tested, we cannot prove the absence of errors. Nevertheless, as previously stated, it allows to increase the confidence in the correct validation of the final system.
According to how they are generated, i.e, how inputs signal are defined and parameterized, and executed, i.e, how the code is executed and how the expected and actual results are compared, there are three level of test automation. Table 1.1 shows the generation and execution characteristics of each of the three testing levels. According to Table 1.1, there are:

- **Manual testing**: both the generation and the execution are manual tasks. This requires the engineer to take the role of the end-user to ensure proper operation.
- **Automated execution testing**: the execution is automated through specialized software that allows creating test cases. However, the generation is still manual, so test engineers must parameterize and define the inputs each time they want to run a test case.
- **Automated generation testing**: this testing method allows automated generation and execution. As in automated execution testing, test engineers develop test cases that are automatically executed through a specific software. Furthermore, test engineers are not anymore in charge of defining and parameterizing the inputs signal each time a test case is run, instead a software carries out this task.

	Automated Execution	Automated Generation
Manual Testing	×	×
Automated Execution Testing	✓	×
Automated Generation Testing	✓	✓

Table 1.1: Execution and generation characteristics of the three automation levels of testing.

As the reader may imagine, manual testing is the traditional method used since the beginning of testing techniques. In recent times, a lot of work has been done to complement manual testing with automated execution testing. In fact, this last method is consolidated in the validation and verification of systems in Volvo Cars. Because of the proliferation of CPSs, traditional testing approaches, manual and automated execution testing, are no longer scalable and they need to be gradually complemented with automated generation testing methods.
MBT described in Section 1.2.1, allows testing at different levels of integration. The most relevant to the master's thesis are [12, 17]:

- **Model-in-the- Loop (MIL)**: both the software components and the physical components of the systems are simulated, without any hardware component. Models created from tools such as Simulink are used for this purpose. This is the first level of integration.
- **Software-in-the-Loop (SIL)**: the main difference between MIL and SIL is that in the latter the model is compiled and run with a fixed time-step. That is, the software is compiled and tested within a simulated environment but without any hardware component.
- **Hardware-in-the-Loop (HIL)**: the software runs on Electronic Control Units (ECUs), Battery Electronic Control Module (BECM) and the Electric Drive Module (EDM) in the case of the thesis, i.e., hardware components are used, but the environment with which it interacts is still simulated. In this integration level, a real-time simulation is used.

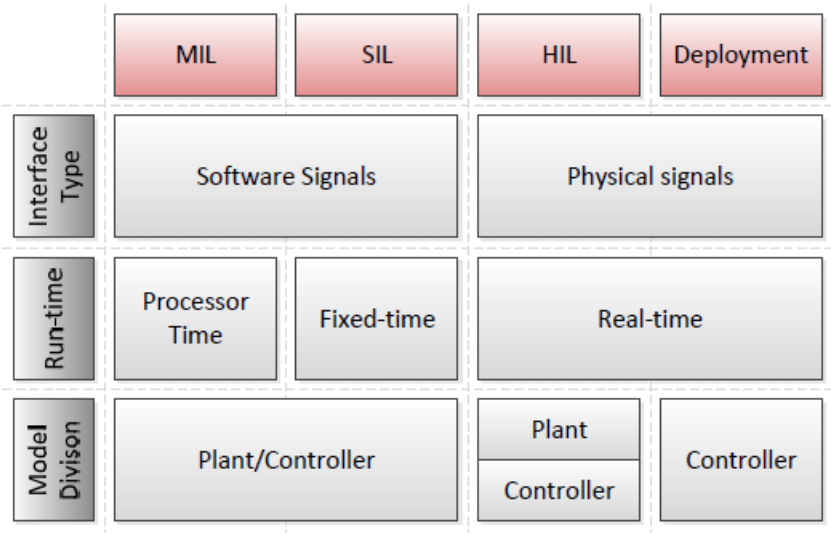


Figure 1.2: Testing characteristics according to each integration level. Figure obtained from [17].

Figure 1.2 shows the similarities and differences between the three integration levels. As can be seen, both MIL and SIL use software interfaces for the signals, while the HIL approach uses a physical interface since it works with hardware. Simulations are run differently: MIL at processor time, SIL at fixed-time, and HIL at real-time. Finally, it is verified that both MIL and SIL simulate the plant and the controller, and HIL only simulates the controller since the plant is the hardware.
In this master's thesis, tests are carried out at the last level of integration described, HIL. In addition, as mentioned above, because of the complexity of the systems, our work is based on an automated test generation approach, which is implemented in the HIL environment with the use of falsification technique and Breach toolbox.

1.4 Objective and scope

As discussed in Section 1.1.2, one of the main challenges of CPSs is validation and verification. Because of their complexity, formal techniques cannot be applied, and instead, testing must be used. One approach to overcome this limitation and increase the automation level is MBT. Among the three testing integration levels described in Section 1.3.2, two (SIL and MIL) have already been implemented at Volvo Cars with positive results, as shown in [18]. However, the last integration level HIL, where hardware elements are used, has not yet been proven.
The master's thesis has the overall aim of implementing a specific method of automated test generation, falsification, for the electric drive and/or the battery control module in the HIL integration level for electrified vehicles in development at Volvo Cars. The motivation is to provide Volvo Cars with a method to optimize testing tasks, that detects failures at earlier stages of development.

Within this framework, the three main objectives of this master's thesis are established:

1. Analysis of the framework, i.e, set of guidelines used for creating and designing test cases, followed at Volvo Cars for HIL environment testing.
2. Development of a general test case framework to implement automated testing generation using falsification in the HIL environment.
3. Development of the necessary tools for the implementation of the general framework.

1.5 Method

The purpose of the master's thesis was to explore an under-researched topic: the implementation of automated test generation in an industrial environment. The research was descriptive in nature, as not much is known yet about the topic. Also, this research was based on qualitative and primary data, since test cases developed in Volvo Cars were used as the main data source. This qualitative and descriptive nature is the one that best allows us to answer the research questions stated in Section 1.6.

As discussed above, the data needed to develop this master's thesis has been mainly the existing test cases in Volvo Cars, i.e., the data existed as pre-developed test cases when the master's thesis was initiated. To gain insight into the possibilities of implementing automated test generation, a general review of these test cases was performed with the help of the test engineers working on each of the analyzed hardware systems. The aim was to gather those test cases that best suited our purpose, i.e., that could be automated using falsification. As a result, a couple of test cases were selected from each of the two ECUs: BECM and EDM.

Once the test cases were selected, the method, the structure, and specifications were analyzed to adapt them to the new automated test generation approach. Again, this study was carried out in collaboration with the test engineers and was based on the knowledge gained from the initial literature review.

The falsification technique was chosen to further develop automated generation testing in an industrial environment for HIL systems, since it has already been successfully implemented in SIL and MIL at Volvo Cars, as shown in [19]. It is therefore logical to continue the development process already started. The same holds for the tools, Automation Desk and Breach have been selected because they are the ones used at Volvo Cars.

- **Chapter 2** explains the theoretical foundation on which the project is based. First, the concept of temporal logic and its two main variants, Linear Temporal Logic and Signal Temporal Logic, are introduced. This is followed by a detailed description of the falsification process and tools.
- **Chapter 3** is about Hardware-in-the-Loop testing. The HIL testing method, the HIL setup, and the main components that make it up are explained. Then, it is described how the HIL approach is implemented in Volvo Cars.
- In **Chapter 4** the proposed framework to combine falsification technique with the HIL testing in Volvo Cars is presented.

- **Chapter 5** addresses the analysis of the work presented in the master's thesis and the future work to be accomplished. It also includes a discussion about ethics and sustainability and the answer to the research question.

1.6 Research questions

In addition to the objectives at hand, academical aspects of the thesis are presented. To connect the work of the project and represent the academical value we define and present research questions. These questions are associated to the project content as a whole, meaning that they are stated so that they connect how the literature and background can be utilized towards the application. Throughout the thesis, the content and relevancy of the questions are considered and at the later stage, answered upon. They are stated as the following:

Question 1 - How can falsification of CPSs be used in an industrial environment, for automated test generation of electrical automotive HIL systems?

Question 2 - How should automated test generation based on falsification be used in conjunction with traditional manual testing?

Question 3 - What kind of specifications are suitable to use for the kind of automotive HIL systems considered in the thesis as compared to typical simulation-based testing?

The first question takes into account how the main method of automating and testing, namely falsification, be used on CPSs and then applied to a practical aspect. It is stated so that the intent is to answer upon how this method of automation can be used for testing on HIL systems. Thus the question is formed such that it takes into account of how the theoretical background can be applied in practice.
The second question has the intent of stating comparisons between traditional manual testing and automated test generation based on the falsification method. By providing the information of how both manual testing and automated test generation works, one can then understand when it is suitable and when it is not suitable to use one or the other.
The third question might seem to be stated more complex than the other ones. Definitions on specifications have to be made. It take into account what specific requirements are of use in testing. Such specifications vary and therefore there may be those that are more suitable for HIL systems than simulation-based testing. The difference of HIL systems and simulation-based testing also have to be stated. The advantages and disadvantages of both ways of testing depends on the testing itself. Such testing should be clearly explained so that it is possible to apprehend what specifications are suitable.

2

Falsification of Cyber-Physical Systems

In this chapter we provide a theoretical background on the concepts of Temporal Logic and the interdisciplinary branches of it. In particular we present an explanation on Signal Temporal Logic (STL), which is the language of the specifications considered. We also present the method of approach to automate the testing generation, called falsification, and give an insight on how the the falsification problem is solved and what falsification tools exists.

2.1 Temporal Logic

Temporal logic is a logic for specifying properties over time, meaning that it is possible to make arguments of *what is*, *what was* and *what will be* in terms of propositions. It is a formalism widely used for specifying desired behaviours of a system and predict certain results. Originally the usage of temporal logic was to reason about Input/Output (I/O) systems. An I/O system is a system where the input and output is the communication between information processing systems and the real world, which can be a human or another system. Usually inputs and outputs come and goes to physical devices and would then be called input and output devices. An example of an I/O system is a computer. The inputs would come from a keyboard as the acting input device and the output would go to a monitor acting as the output device. The intention is to evaluate well-defined sequences of states and events found in said systems. The I/O systems uses Boolean, discrete-time signals to focus on verification, specification and synchronous systems. It is often described as a branch of modal logic [20].

There exists several varieties of temporal logics and expansions on those. One of the original temporal logics described for reasoning on formal systems is Linear Temporal Logic (LTL) [21]. Over the years, there has been extentions to reason about specifications of properties of real-valued signals defined over time. Such extensions are Metric Temporal Logic (MTL) and Signal Temporal logic (STL) [22]. These logics can express requirements for safety and liveness, that is, running a system safely without hazardous behaviour and avoiding deadlocks.

2.1.1 Linear Temporal Logic

One of the original varieties is LTL. It was described by Pnueli in 1977 [21]. It is a logic for execution paths of systems and is one of the most important logics for verification of software and hardware. In system design LTL is used as a formalism for specifying desirable and acceptable behaviors or reactive systems. As a turning point in verification it is putting a ongoing sequential behavior of a system at the center stage of the verification process. A basic intuition would be to consider execution paths of a system into the future. An execution is the sequence of states to which the system goes. From each point of time during the execution, one is looking into the future.

2.1.2 Discrete-time signals

As will be later discussed in Section 2.2, falsification is based on the evaluation of specifications described in temporal logic. Falsification can be defined either in continuous time or in discrete-time. For this master's thesis, we have selected discrete-time since we work with computer devices. A formal definition of discrete-time is given below [19].

Definition 1. *A discrete-time signal $x[k]$ is a function $x : I \to \mathbb{R}$ from a finite subset of $I \subset \mathbb{Z}$ to $\mathbb{R}$, where $k \in I$. The set I labels the time instants of the signal, and the signal takes on continuous values at each of those time instants.*

2.1.3 Signal Temporal Logic

The language of specification which is widely used for CPSs is STL. It is usually defined in terms of continuous-time signals, however it also works for discrete-time signals [23].
STL was introduced to monitor the temporal properties of the signals as real-valued signals. [24]. Lately, it has been extended to the specification of properties of said real-valued signals and applied to various domains. The combination of dense time modalities with numerical predicates allows for deciding satisfaction of properties.
STL formulas consist of atomic predicates connected by Boolean and temporal operators. They are defined as

$$\varphi := \mu \mid \neg\mu \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \Box_{[a,b]}\psi \mid \Diamond_{[a,b]}\psi \mid \mathcal{U}_{[a,b]}\psi, \tag{2.1}$$

where φ and ψ are STL formulas and μ is a predicate [18]. The predicate is also a formula which can consist of other operators, though it yields a Boolean value. The operators $\neg, \wedge$, and $\vee$ (*not*, *and*, *or*) are propositional connectives. The operators $\Box, \Diamond$, and $\mathcal{U}$ (*always*, *eventually*, *until*) are temporal connectives. The subscript notation $[a, b]$ represents the intervals of the time frame in which the execution is being held. Notating without a time interval, like $\mathcal{U}$, is used as a short for the infinite time interval $\mathcal{U}_{[0,\infty]}$. Some of the operators used in STL are similar or the same as in earlier logics such as LTL. Making use of De Morgan's law it is possible to define logical *or* $\varphi \vee \psi$ as $\neg(\neg\varphi \wedge \neg\psi)$. Similarly, it is possible to define *eventually* $\Diamond_{[a,b]}\varphi$ as

$\neg(\Box_{[a,b]}\neg\varphi)$. The semantics of a STL formula φ relative to a signal $x = (x_1, ..., x_m)$ is immediate. This means that the semantics of x is the signal x and the semantics of a constant is the constant signal c.
An example from [25] of a specification using STL properties for a case in the automotive can be described as

$$\varphi_i = \Box_{[0,30]}((v < 120) \wedge (\omega < 4000)), \tag{2.2}$$

where v is the vehicle speed (km/h) and ω is the engine speed (rpm). The interpretation of the requirement is that "during the first 30 seconds, the speed of the vehicle should always be less than 120 km/h and the speed of the engine will always be less than 4000 rpm."
It is imperative to make sure STL formulas are validated for use. specifications which can only end up at one certain result after testing is not valid for testing. The requirements shall be written so that there can be different outcomes. Like in [26], the validity of a formula φ with respect to the discrete-time signal x at time instant k is defined as

$$\begin{aligned}
&(x,k) \models \mu && \Leftrightarrow\ \mu(x[k]) > 0\\
&(x,k) \models \neg\mu && \Leftrightarrow\ \neg((x,k) \models \mu)\\
&(x,k) \models \varphi \wedge \psi && \Leftrightarrow\ (x,k) \models \varphi \wedge (x,k) \models \psi\\
&(x,k) \models \varphi \vee \psi && \Leftrightarrow\ (x,k) \models \varphi \vee (x,k) \models \psi\\
&(x,k) \models \Box_{[a,b]}\varphi && \Leftrightarrow\ \forall k' \in [k+a, k+b], (x,k') \models \varphi\\
&(x,k) \models \Diamond_{[a,b]}\varphi && \Leftrightarrow\ \exists k' \in [k+a, k+b], (x,k') \models \varphi\\
&(x,k) \models \varphi\ \mathcal{U}_{[a,b]}\psi \Leftrightarrow && \exists k' \in [k+a, k+b]\ \ (x,k') \models \psi\\
& && \wedge \forall k'' \in [k, k'), (x,k'') \models \varphi.
\end{aligned}$$

With the definitions of how to validate the formulas it is possible to review criterions. Regard the satisfactions as criterions and upholding the satisfactions shall make sure that the specifications are valid and eventually be of use in testing.

Robustness degree function

One useful property of STL formulas, which makes their utilization very interesting, is the possibility of quantifying the satisfaction or violation of a specification. The concept was described in [27] under the name of robustness degree (or just robustness; terms will be used interchangeably hereinafter). In addition, other works, such as those in [28] and in [29] have introduced methods for monitoring this parameter. Its value lies in providing more descriptive feedback than a simple Boolean response, i.e., *true* or *false*, as to whether a requirement has been met or violated, which allows for better learning of the system under test.
A basic characteristic of robustness ρ is that it is a real-valued function, whose sign indicates whether the STL specification φ has been met or not: positive sign means satisfied, negative one means not. Meanwhile, the magnitude of ρ shows how far a STL specification is from being violated or fulfilled. The robustness is a function of a specification φ, a signal x, and a discrete-time k at which the robustness is

evaluated. It is defined as follows from [19] and [26]

$$\begin{aligned}
\rho(\mu, x, k) &= \mu(x[k]) && (2.3)\\
\rho(\neg\mu, x, k) &= -\mu(x[k])) && (2.4)\\
\rho(\varphi \wedge \psi, x, k) &= \min(\rho(\varphi, x, k), \rho(\psi, x, k)) && (2.5)\\
\rho(\varphi \vee \psi, x, k) &= \max(\rho(\varphi, x, k), \rho(\psi, x, k)) && (2.6)\\
\rho(\Box_{[a,b]}\varphi, x, k) &= \min_{k' \in [k+a, k+b]} \rho(\varphi, x, k') && (2.7)\\
\rho(\Diamond_{[a,b]}\varphi, x, k) &= \max_{k' \in [k+a, k+b]} \rho(\varphi, x, k') && (2.8)\\
\rho(\varphi \,\mathcal{U}_{[a,b]}\psi, x, k) &= \max_{k' \in [k+a, k+b]} (\min(\rho(\psi, x, k'), && (2.9)\\
&\quad \min_{k'' \in [k, k']} \rho(\varphi, x, k''))).
\end{aligned}$$

A example follows. Returning to the specification defined in Equation 2.2 but simplifying it for a better understanding, only the first part is included. Consequently, the specification in natural language is: the vehicle speed, v, should always be less than 120 km/h during the first 30 seconds of simulation. This way the specification in STL is $\varphi_1 = \Box_{[0,30]}(v < 120)$. For specification φ_1, $\mu(x[k])$ is defined as $120 - x[k]$ and the robustness at time 0 is $\rho(\varphi_1, x, 0) = 120 - x[0]$. Consider two simulations with results x_1 and x_2 that meet the specification, i.e., v is always less than 120 km/h during the first 30 seconds of simulation, and let us say that $x_1[0] = 110$ and $x_2[0] = 80$. Intuitively, $x_1[0]$ is closer to not satisfying the specification, and thus should have lower robustness value. Indeed, the robustness value of x_1 is 10 and the robustness value of x_2 is 40.

2.2 Falsification

As seen in Section 1.3, due to the complexity of the systems addressed, we will make use of testing techniques for their validation. Figure 2.1 shows a classification of the different testing and verification approaches, according to their degree of scalability and formalism. Among all those shown, we will focus on the method we are trying to implement: falsification. Its position in Figure 2.1 makes clear, on the one hand, that it is a testing technique. On the other hand, it reflects that it is a method that automatically chooses the operating conditions, i.e., inputs and their parametrization, and that is suitable for medium and high complexity systems.
Having contextualized what falsification is, we will now introduce the falsification problem, based on [9]:
Given a system $\mathcal{M}$, a set of parameters $\mathcal{P}$, a set of inputs $\mathcal{U}$, and a property φ that must be satisfied by the system, the falsification problem consists of finding a $p \in \mathcal{P}$ and a $u \in \mathcal{U}$ such that the behaviour of system $\mathcal{M}$ under parameter p and input u, $\Phi(\mathcal{M}, p, u)$, does not satisfy the property φ which is expressed as $\Phi(\mathcal{M}, p, u) \nvDash \varphi$. In other words, the objective of the falsification problem is to find the parameters and inputs that lead the system to the violation of a requirement. Such parameters and inputs are known as a counterexample [30].

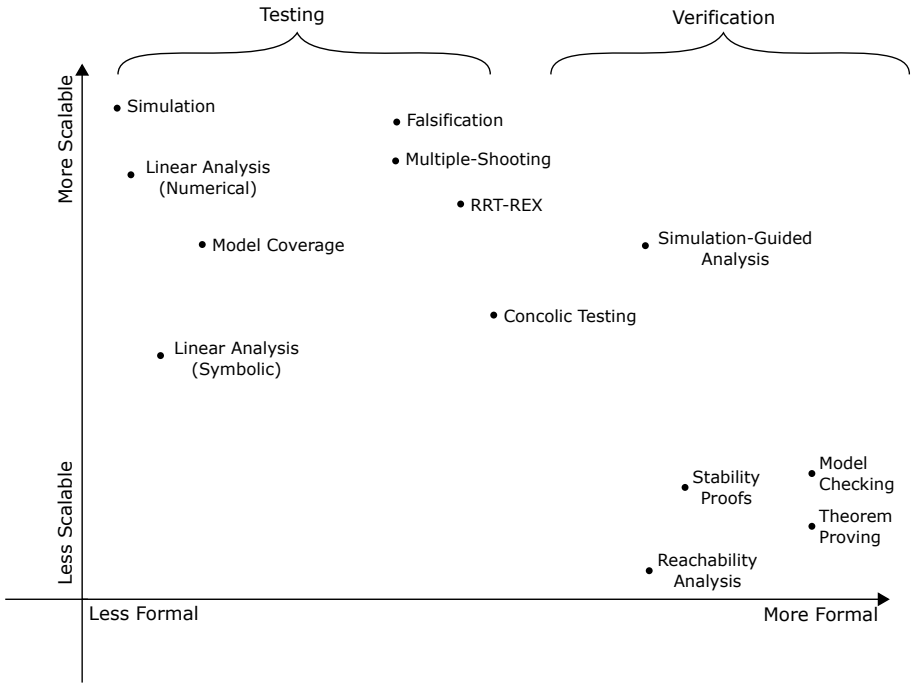


Figure 2.1: Set of validation techniques, classified according to their scalability and formalism. Figure reproduced from [9].

This problem, i.e., the search of parameters and inputs that lead the system to violate a specification, can be solved in different ways [14]. In our case, we will use an approach based on optimization techniques, which its main advantage is the minimal number of restrictions placed on the simulator of the system and environment. The standard form of an optimization problem is

$$\min_{x} \quad f(x), \tag{2.10}$$

where $f : \mathbb{R} \to \mathbb{R}$ is the objective function to be minimized.
For our purpose, where the specifications are written in signal temporal logic (see Section 2.1.3), the robustness function ρ is used as the objective function, so that the general falsification problem is

$$\min_{u(p)\in P} \quad \rho(\varphi, \mathcal{M}(u(p)), 0). \tag{2.11}$$

Figure 2.2 shows an overview of the process described above. The generator provides, based on the input parameters, inputs to the system under test. The simulator represents the SUT, in our case the HIL simulation environment. Once the simulation is completed, the simulation trace together with the specification is used to evaluate the robustness function. This value is analyzed to determine if the specification is falsified or not. If it is not falsified, new parameters are sampled and the process is repeated. The parameter optimizer is a global optimizer which attempts to find new input parameters that are closer to falsifying the specification, i.e., parameters that lead to a lower robustness value.

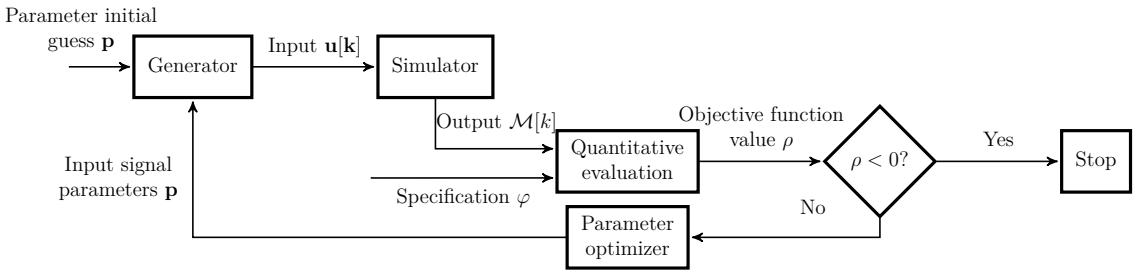


Figure 2.2: Flowchart that describes the falsification process. Figure obtained from [19].

Input generation

Before the input generation, the parameter vector must be correctly defined. For example, when varying the temperature under which a system operates, the values must always be within the range where the system operates safely [19]. This requires a thorough knowledge of the SUT, since CPSs involve complex dynamics. Also, for a better performance of the optimization problem it is better to use as few parameters as possible, since an excessive use would increase the search space in the optimization problem uncontrollably.

Robustness

The robustness value comes from the robustness function introduced in Equation 2.3. It is a measure of the degree to which the output $\mathcal{M}(u(p))$ satisfies the specification φ [14]. This value is used by the evaluation function to check if the specification has been falsified, and decide whether to terminate the algorithm or try to find a new set of parameters. As earlier stated, it also provides very useful information. A negative value means that the system has violated the specification and therefore the algorithm must terminate. Conversely, a positive value indicates that the specification has not been violated at any point. Within the positive values of robustness, high values indicate that the system is not close to violating the specification and low ones that the specification is close to be violated.

2.2.1 Example of falsification procedure

The following is a simplified example of the falsification process. The aim is to help the reader better understand the process and, in particular, how the input generator and the robustness function work. It has been obtained from [19].
The SUT is a model of the automatic transmission of a vehicle [25]. It consists of two inputs: the accelerator and the brake. The system also has two outputs: the engine speed ω [RPM] and the vehicle speed v [mph]. The first input, the accelerator, can take values in the range of [0,100] and the brake within the interval [0,500]. The system has been simulated in three different scenarios. Figure 2.3 shows the throttle and brake curves for each of the three scenarios.

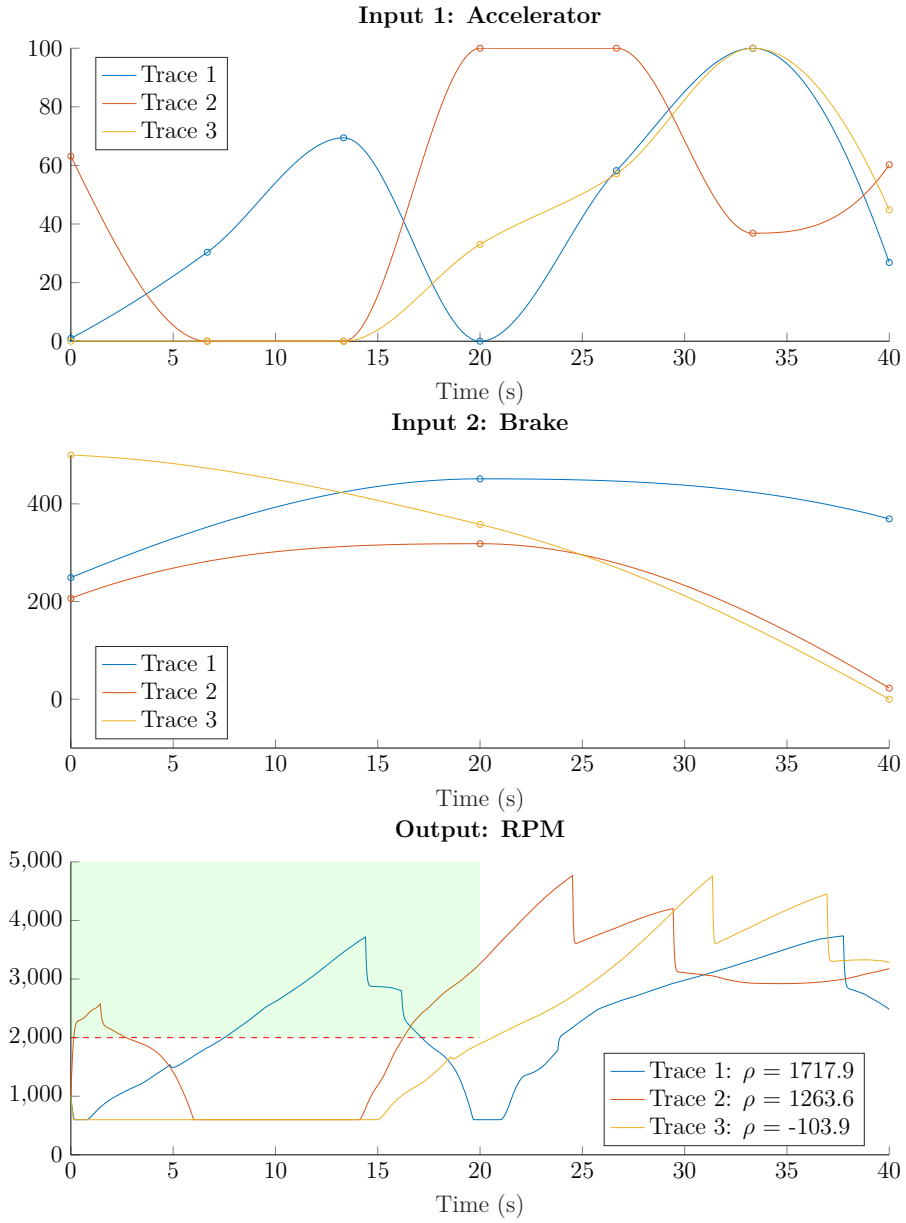


Figure 2.3: Signals of the example proposed in 2.2.1 for the three simulated scenarios. (Input 1) Trace of the accelerator input signal. (Input 2) Plot of the brake input signal. (Output) Output signal of the engine speed ω and the robustness value in each of the three scenarios. Figure obtained from [19].

The specification to verify is: does the motor reach a speed equal to or higher than 2000 RPM during the first 20 seconds of simulation? In STL language this corresponds to

$$\varphi = \Diamond_{[0,20]}(\omega \geq 2000). \tag{2.12}$$

In Figure 2.3, compliance with the specification is represented by a green rectangle that encompasses speeds equal to or greater than 2000 RPM during the first 20 seconds of simulation. Therefore, if the output trace enters at least once that area, the specification will have been fulfilled. If it does not enter the green space during the first 20 seconds, the system has violated the specification.
The standard robustness function used to solve the optimization problem is

$$\rho(\varphi, x, 0) = \max_{k' \in [0,20]}(\omega[k'] - 2000). \tag{2.13}$$

It follows that the robustness value for each of the three output traces is the maximum difference between the motor speed ω and the target speed 2000 during the first 20 seconds. In the lower-right margin of Figure 2.3, we can observe the robustness values for each case. Trace 1 takes the maximum robustness value, which means the scenario is the farthest from falsifying the specification. Trace 2 has an intermediate value of 1263.6. Finally, trace 3 has a negative value of -103.9, which, as can be seen in the graph, means it does not enter the green area and therefore the specification is falsified.

2.3 Falsification tools

Different tools allow the implementation of falsification. The main ones are presented below to have a general view but without going into detail [30]:

- **falsify**: an experimental tool that solves falsification problems using a reinforcement learning approach. It can learn the behavior of the system by observing the outputs of the system during the simulation.
- **ARIsTEO**: MATLAB toolbox for tests generation that evaluates specifications only written in STL. Its purpose is to validate and verify a specific type of CPS, known as compute-intensive (CI), which is characterized by requiring up to hours to complete its validation.
- **FalCAuN**: experimental tool based on automata learning and model checking, for testing systems in Simulink. It is designed to falsify a Simulink model against multiple specifications through previous learning.
- **Breach**: it is a MATLAB toolbox that allows test case generation, formal specifications monitoring, and optimization-based falsification implementation. It is characterized by its great flexibility in input generation and modularity for requirements evaluation.
- **ForeSee**: it introduces a new concept of robustness to address the scale problem, i.e., when the input signals have different units, so that one could be masked by the other. The strategy followed for solving the falsification problem is based on a Monte Carlo Tree Search over the structure of the formal specification.

- **S-TaLiRo**: MATLAB toolbox for monitoring and generating tests for the evaluation of specifications presented in STL. Like Breach, it uses optimization techniques to solve the falsification problem through the different algorithms it supports.

The two most popular tools are Breach and S-TaLiRo [31]. This master's thesis uses the first, since it has already been used to implement falsification in MIL and SIL integration levels in Volvo Cars. In addition, it allows solving the falsification problem from an optimization perspective, as explained in Section 2.2, using one of the different algorithms it supports. In Breach, the input signals are defined and parameterized, as explained in Section 4.2.2, and also the requirements are defined in STL, as detailed in Sections 4.2.3 and 4.2.4.

3

Automated Testing of Hardware-in-the-Loop Systems

In this chapter we provide the methodology used in practice on HIL systems and testing. We present the definitions of the HIL system of use and the model of it. Moving on to the test automation as it is of today. Then providing HIL falsification as it is defined from the background on HIL systems and falsification together.

3.1 HIL Introduction/Presence in automotive industry

While the future is moving towards electrifying vehicles [32], HIL simulations have historically been used for design and testing of (combustion) engine-control systems in the automotive industry [33]. In later years the focus of the development of engine-control systems in particular is more keen on electric machines rather than combustion engines [34, 35]. With the development of said systems, rigorous testing is necessary for various reasons. One reason is the considerable weakness of electrical vehicles, their batteries. Although the development has been increasing in recent years, the driving range is still considered low. To improve the driving range, more development is needed and thus increased testing. This in turn also affects investments and development costs. Motivations of why electrical vehicles as an alternative to fossil fuel driven vehicles is more clear as of recent times. However, the means of developing and improving the vehicle systems goes in relation with that of improving manners of development and in turn testing. That is where HIL testing is a preferable approach for many of the manufacturers and suppliers in the automotive industry. With HIL it is possible to test the software and hardware at the same time. The goals of the testing are to test the objects in a controlled environment, and detect and resolve faults before the object is integrated into the vehicle. The relevant test objects of the thesis are ECUs. The overview process of how the test and development of ECUs is performed is shown in Figure 3.1.

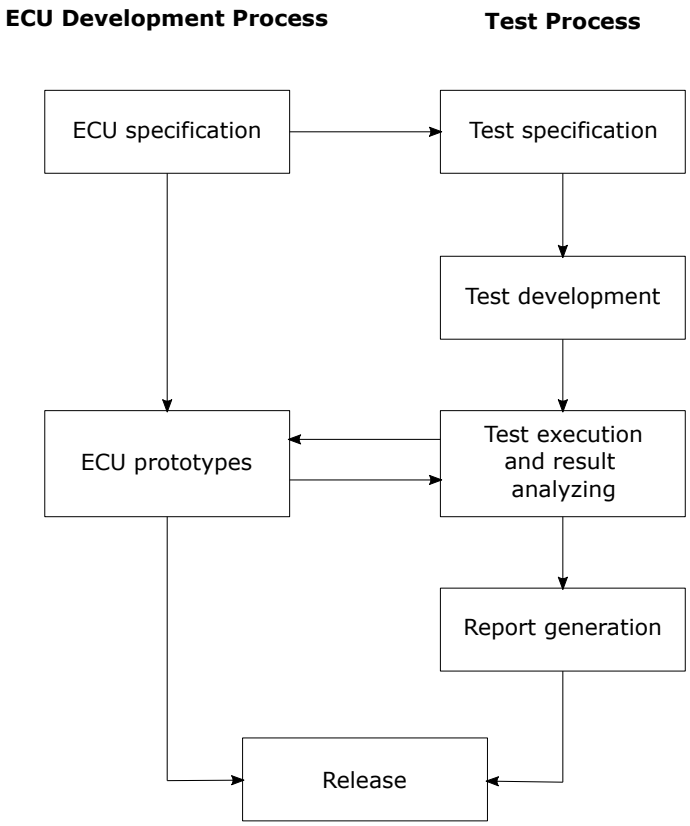


Figure 3.1: Test and development process of ECUs.

3.1.1 Test objects

With the more complex sub-systems and functionality of modern day vehicles, the number of ECUs present in the vehicles are increasing. A vehicle can contain more than 100 ECUs and according to [36] it is on the rise. There is not only the mechanical propulsion of having the vehicle move or perform work, but also many other functionalities like entertainment services or Advanced Driver-Assistance Systems (ADAS) [37]. That is where different types of ECUs do different kinds of work and computations. The ECUs are acting as a collection of computer nodes on a bus network. This bus network (or topology) is traditionally using Controller Area Network (CAN) as the common protocol between ECUs [38]. Depending on the ECU, there might also be other protocols such as FlexRay or Ethernet, though CAN is the most common protocol [39]. An ECU contains the following elements:

- **Microcontroller** - The core of the ECU consists of the microcontroller which is the acting computing device. The microcontroller itself contains one or more Central Processing Units (CPUs).
- **Memory** - The memory consists of Random-Access Memory (RAM) and Flash. As the RAM is static it is volatile, meaning that the stored information is lost if power is removed. The Flash memory is instead non-volatile so it can retain stored information when power is removed. With Flash, the ECU can be reprogrammed and erased electrically.

- **Software** - The software on the ECU is embedded. Mainly, the software makes use of routines for functionality. Along with this there is configuration data to connect the hardware and software, i.e. which physical resources to use. There is also certain boot loaders which is the program that handles booting of the computer device.
- **Inputs/Outputs** - The I/O of the ECU is fairly simple to its purpose. One input is the power supply voltage to power the ECU. Then there is the input signals which can be either analog or digital. The outputs are thereafter used for actuation, for example to handle relays or valves, and logical outputs to a third party service, for example lights.
- **Communication** - The communication is linked via bus transceivers. Such transceivers are made for the earlier mentioned CAN protocol and others.

The BECM and EDM are no exceptions from this architecture and are considered amongst other ECUs, vital of an electrical vehicle. They handle a lot of functionality related to the main aspects of a vehicle, that is, propulsion of the vehicle and does not only focus on secondary functionality, like entertainment services. As such, those ECUs are acting as appropriate test objects for HIL system environments and convenient examples for the thesis to show how falsification can be used to automate tests for said systems.

3.2 HIL setup at Volvo Cars

HIL simulation is a technique that is used in the development and test of real-time embedded systems, in this case the test objects. As has been stated, the hardware considered in this thesis is the EDM and BECM. These hardware components are the acting test objects of the HIL simulation. The test objects are physically connected to a HIL simulator. The HIL simulator is built up by essential parts to perform the simulation technique and be shown in Figure 3.2.

3.2.1 Environment model

Environment models contain mathematical representations of the environment around the test object. Environment models includes physical components, other ECUs and the emulation of the electrical interface with the test object. The (variable) architecture of the environment follows a model structure which contain the following elements:

- **Plant** - Plant is a word which is widely used in control-system language. The plant is the control object that is being controlled. The plant itself contain the models of the physical components and these components interact with the test object. When handling variables in the HIL system, the plant is the element that contains the variables of physical quantity. These variables are measured and handled in the controllers. The inputs to the plant model are actuation signals from the controllers and the outputs from the plant model are physical signals that are measured by the controllers. Some examples of plant models of physical hardware are Electrical machines, gearboxes, or inverters.

- **Controller** - The controller is that which controls the plant. Input to the controller is a reference signal, a desired value and output from the controller is the signal to the controlled object. The controller contains models of software functionality and makes use of said input and outputs to handle said functionality. A controller is designed and used in a way that it corresponds to the ECU topology of the vehicle and so signals are grouped and transmitted via CAN bus to to the ECU. Some signals are also transmitted to communicate with other ECUs in the architecture and not just the solemn one which the controller belongs to.
- **I/O** - The I/O handles the emulation of the electrical interface with the test object. The inputs and outputs are analog and digital signals which are converted to and from voltages and digital levels. Other signals are Pulse-width modulation (PWM) signals. PWM is the manner of reducing the average power carried out by a electrical signal. It does this by actively dividing the signal into into discrete parts. PWM is, in a sense, a way of digitally encoding analog signal levels. PWM signals can be used to monitor or control functionality, for example monitoring and controlling a contactor. The serial communication carried out by signals is then CAN. There are thus several different I/O's divided into different boards and different connectors.

As for the thesis, the relevant plant models are those where the EDM and BECM are the control objects. The controller handles the functionality towards those objects and the I/O defines the electrical interface between the HIL and the objects.

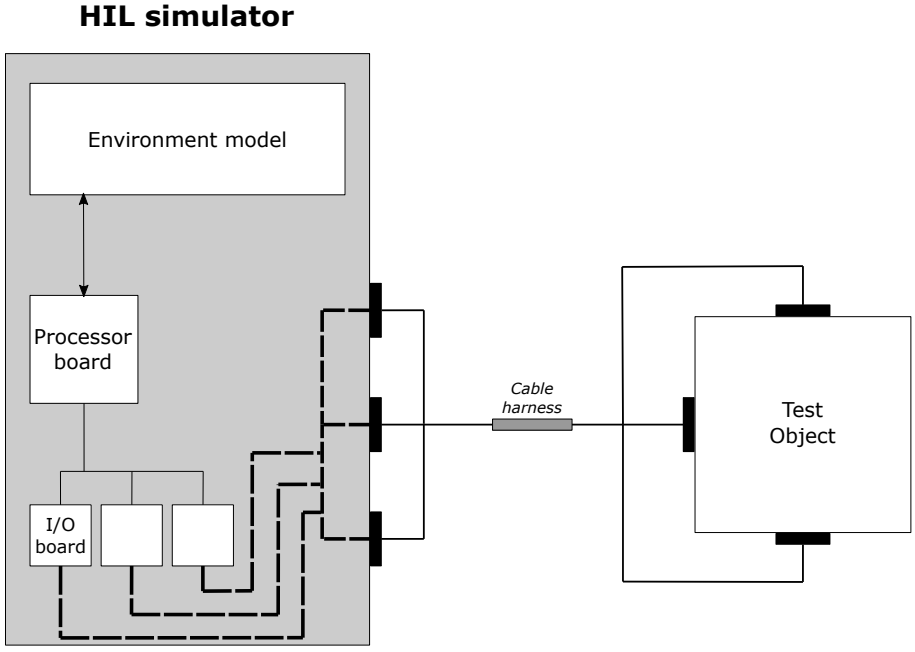


Figure 3.2: HIL - Test environment. The HIL simulator and connection to the test object.

3.2.2 Real-time system

Simulations performed on the ECUs are in real-time. A computer model of a physical system that can execute at the same rate as actual real time, like a wall clock or a wristwatch, is simulated in real-time. In practice it means that the model runs at the same rate as the physical system. For example, if it takes 4.9 seconds for a vehicle to reach 100 km/h in the real world, the real-time simulation would take just as much time. There are different types of simulations and usages [40], though the common simulations running in HIL environments are executed in discrete-time and the performed time steps are constant. Just like with real-time simulation, the time-steps in discrete-time simulation progress in equal duration. This is known as fixed time-step simulation.
For application in HIL, a physical controller is connected to a virtual plant executed on a real-time simulator, that is, not a physical plant. The HIL simulator contain a real-time simulator and which used the environment model as the computer model and the ECU as the physical system. The ECU is connected via the I/O's to the real-time simulating computing device. Thus the simulations performed on the ECU are in real-time.

3.2.3 Signals in HIL

A signal specifies a variable's value over time, i.e. it describes how one parameter varies with another parameter, $x(t)$ in continuous time and $x[k]$ in discrete time. In systems, signals keep required information and are used to store and replay measured variable values, and to specify references to evaluate measured signals. Practically, a signal is implemented and represented as a variable in the tools and programming languages of use in HIL systems.
A real-valued signal is a signal which takes values that take on the real values of $\mathbb{R}$. The real-valued signals can be expressed in both continuous and discrete time and values. When describing signals and their usage with testing in the thesis, the signals of use are said real-valued and mostly discrete. An example of a real-valued continuous time signal is an analog signal of a power supply reading, that is, electrical voltage $v(t)$ where the voltage v takes on real values in continuous time t. As such the voltage is the physical measurement and the quantities of the signal can be represented at any time in the sequence. The shape of a signal like this can be sinusoidal.
An example of a real-valued discrete time signal is a digital signal of a temperature reading from a thermometer $T[k]$ where a temperature takes on real values in discrete time k. The measurements are specified at specific time intervals and so the quantity of the signal can be represented only at those certain times instances of the time sequence. The shape of a signal like this can be a square wave.
Some signals can also be divided into segments which is the case for Signal-based testing. In Signal-based testing, the test is stimulated by input signals and the reactions are captured as output signals. The output signals are then compared to reference signals which are specified beforehand in segments. Each segment can be defined on its own with unique properties, thus configuring an arbitrary signal shape.

3.2.4 Test case definition

Test cases follow certain structures and guidelines in order to keep them formal. The intention is to keep the test cases thorough and easy to comprehend. Definitions on test cases and what they contain are described and with that information it is possible to structure and design test cases. Fundamentally, a test case is a collection of one or more actions to test the correct functionality and features of a certain objective. Most often an expected outcome or expected result is given before hand for the objective.
A test case contains several building blocks to act as the collection of actions. A test case consists of the following elements:

- **Block** - A block is commonly defined as an action or activity. An action or activity is the operating part, an instruction, leading to a change of the input data. It is described as operating on the input, for example perform logical operations, computations or comparisons. A block can contain a single or multiple actions or activities. It can also be a structural object that groups a number of activities together so that one activity can call upon and make use of other activities.
- **Test step** - A test step is a collection of blocks which performs an action and produces a result. A test step can be that of performing actions on a referenced signal so that an initial value changes to attain a desired value, an action performed for an expected outcome. When a test case is designed it is customary to not use too many blocks so that it is comprehensible.
- **Test section** - A test section is a collection of test steps that are used together. Test sections can contain one or more test steps. The intention is to increase readability and gather the information of the part of the test.

A test case is then a collection of test sections, containing the elements as components down to the fundamental level of actions or activities. This is visualized in Figure 3.3. The test sections built up and executed in a sequence so that the conclusion of the whole test case is the conclusion of all the test sections together, broken down to its smallest component. The test case can then correspond to a requirement or an expected outcome towards verification and validation.

3.2.5 Block-based testing

As described, the smallest building block of a test case is just that, a block. Not only is it named that way because of framing the functionality, but it is also named so when working with it in graphical programming. The blocks program code is text based, but visualised as blocks in a sequence. Said blocks thus contain data objects which consists of referenced signals and variables.
The test cases are created in a sequential structure and executed in a tool called AutomationDesk. AutomationDesk is a test- and automation tool used for HIL testing of ECUs. It is used to make specific steps of an automation process more efficiently.
Within this tool there are two substantial ways of testing, Control flow testing and Signal-based testing. Control flow testing is when one consider a test as an algorithm that is first implemented according to its control flow and then parameterized via

data objects before it is executed. Signal-based testing is when one consider a test scenario as something that is stimulated by input signals. Its reactions are captured in the form of output signals. For evaluation, the output signals are compared to reference signals.

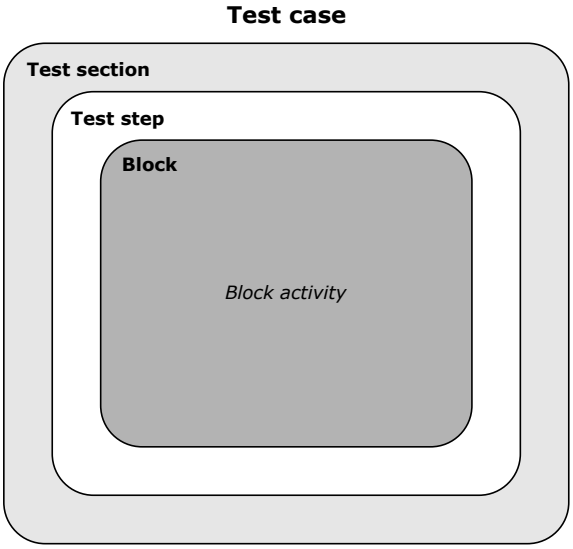


Figure 3.3: Test case depth on fundamental level. Note that a block can contain multiple activities, a test step can contain multiple blocks, and a test section can contain multiple test steps.

3.3 Traditional HIL testing

The following section describes the general framework followed in Volvo Cars for implementing HIL testing. The aim is to get some insight into how HIL tests are performed in Volvo Cars so that we can adapt them to a HIL falsification approach. For this purpose, we have analyzed various test cases related to two ECUs, our CPSs: the Battery Control Module and the Electric Drive Module. The information that we have worked with at Volvo Cars, i.e., the specific test cases, the data, and the knowledge provided by the engineers, is sensitive information and therefore proprietary to Volvo Cars. For this reason, a simplified, tailored, and unclaimed example is used in the following sections to explain the current implementation status and the changes that should be considered to move towards an automated generation strategy using falsification. Although we cannot use the exact information we have worked with, we believe that this example allows us to illustrate and explain the most important concepts, since the structure and general method are similar to the one implemented in Volvo Cars.

3.3.1 Use case description

Here we introduce a use case to illustrate a typical test case. The use case is inspired by previously published examples [41] and [42]. Consider the Adaptive Cruise Control (ACC) of cars, which maintains the speed selected by the driver and, in addition, automatically adjusts the speed to keep the driver-selected distance to the cars in front. The ECU that controls these tasks is known as the ACC controller, which in turn also communicates with other ECUs, such as: the engine controller to accelerate or decelerate, the brake system to actively brake the vehicle, or the restraint system, in case a situation is unavoidable so that it activates the airbags. There are two inputs to the system, the desired *Set_Speed*, which enables selecting the requested speed to hold, and the desired *Set_Distance*, which allows specifying the desired distance to the car in front. There are also two outputs from the System Under Test: *Ego_Speed* and *ECUs_Communication*. The first signal records the real speed of the vehicle and the second the communication between the ACC ECU and the brake ECU. In addition, the system has two auxiliary signals that stimulate the test object but cannot be varied by the engineer: *Car_Ahead* and *Steep_ Slope*. The first one warns the detection of a car in front at a lower speed than the actual set-speed, and the second warns of a slope greater than 10 degrees. The system requirements to be verified, expressed in natural language, are:

- The braking system should activate within the first 10 seconds after a steep downhill slope (>10 degrees) is detected.
- The actual speed must be equal to the speed of the car in front after 5 seconds of the detection of the car ahead and during the following 20 seconds.

3.3.2 Test case structure

The recurrent structure of test cases developed using the HIL traditional technique is shown in Figure 3.4. It is a very general view of the structure of a real test case, the intention of which is to define and explain simply the main characteristics based on the example introduced in Section 3.3.1. As shown in Figure 3.4, the general framework is mainly organized into three distinct sections.
Section 1, *Test Initialization*, initializes the simulated environment, i.e., this section is in charge of executing the mathematical models that emulate the virtual vehicle, which include the vehicle motion and its surroundings. It provides information on the simulator status, allows setting the desired values to boot the simulated environment, and clear parameters that could have been modified in previous tests so that the test object operates under normal state. *Operations* blocks included in the *Simulator Setup* test step follow a standardized sequence that must be executed each time a test case is run. In addition, there is not much room to vary parameters, since, as mentioned above, the method to start the simulator is standardized and varying parameters and/or the order of operations may cause the mathematical models not to run or even simulator safety problems.
In Section 2, *Test Steps and Evaluation*, the test object is stimulated and verification of the requirements is performed. This means, input and internal signals are sent and compliance with the specifications is checked. It should be noted that neither is a trivial procedure. The *Input Generation* block generates the input

signal– for the use case described above, it generates *Set_Speed* and *Set_Distance* signals. A closer look at this block shows that for various reasons, such as test object safety or software limitations, the action of generating a signal also follows a standardized and strict method compromising many write, read, and check steps. The *Recording* test step allows to record signals, generally those used to check specifications. The signals recorded in the suggested example are the *Ego_Speed* and the *ECUs_Communication* signals. The internal signals, in our case *Car_Ahead* and *Steep_Slope*, are triggered by the *Stimulation* blocks, which make it possible, for example, to communicate to the ACC controller that the car is on a slope steeper than 10 degrees by setting the value *true* for the Boolean signal *Steep_Slope*. As described above, these blocks are also complex and require several different steps. Last, Figure 3.4 shows the blocks used to check the specifications, *Evaluation* blocks. They are used to check if the desired signal value range is within the constraints of the specification during a time period. For example, going back to the first specification of Section 3.3.1, it is checked whether the ACC controller has communicated with the brake controller at least once during the first 10 seconds after detecting that the car is driving on a steep slope. Here, the *ECUs_Communication* signal is monitored during 10 seconds to verify that it reaches the high level, i.e., *true*, at least once. Another important feature to consider when checking specifications is that there are two types of *Evaluation* blocks. On the one hand, in cases where requirements do not jeopardize the test object, we compare only the output signals of interest with the specifications, and afterwards continue the test case. On the other hand, if non-compliance with the specification endangers the ECU, a specific block is used that terminates the execution of the test case if the specification is not met.
Finally, section 3, namely *Test Clean up* is dedicated to clean-up operations of everything previously configured.
The execution of the test cases follows a sequential structure, so a block is not executed until the previous one finishes. First, the instructions of the initial section are executed. In this way, the simulation environment will be set up. Then, those commands whose purpose is to stimulate the system and evaluate its outputs are performed. The evaluation of the output signals is not performed once the entire test case ends, but simultaneously to the case execution, since, as mentioned above, for safety it is often necessary to check a specification just after having varied the value of one of the system inputs. Finally, and following a logical order, the blocks that perform a cleanup of the entire system are executed. Also, the data is not saved until the test case is finished or a safety block abruptly stops its execution.

3.3.3 Characteristics

It is important to understand and define the main characteristics of the analyzed HIL implementation. This facilitates the transition from a traditional HIL testing setup to one based on falsification. The main features from the study are presented below:

- **Automated execution testing**: analyzed tests are not fully automated. This means, as stated in 1.3.2, the execution is run automatically but the input signals are defined by testers manually every time a test case is run. The limitations of this approach, among others, are: it requires testers to have a good insight into the SUT, it is time-consuming, and it is not suitable for testing complex systems that require varying multiple input parameters as test engineers cannot test all combinations of parameters and many behaviors are left untested. Another disadvantage of manual testing is the limited variety of test cases. By relying on people, the development of test cases will be conditioned by their way of working and will cause unimaginative test cases, which may also leave interesting behaviors untested.
- **Safety**: in HIL testing, unlike MIL and SIL approaches, safety is a decisive factor. Working with non-simulated components, in our case the Battery Electric Control Module and the Electric Drive Module, requires a thorough understanding of the hardware and its limits. Improper use or lack of knowledge of the hardware operation can lead to system damage.
- **Boolean specifications**: system requirements are usually Boolean conditions, e.g., checking whether a given safety system has been activated or not, which means that the signal can only take the value of *true* or *false*. For example, for the proposed use case, one specification is to check that the braking system activates within the first 10 seconds after a steep downhill slope (>10 degrees) is detected. This means that it is usually not possible to obtain a robustness value for the quality of the system's compliance. This is an issue since the falsification problem is supposed to be solved with an optimization solver, but when the robustness does not vary at all, the optimization solver has no useful information to act upon.

The HIL systems of use in the industry are described in the first sections of this chapter in Section 3.1 and Section 3.2. The hardware along with the test case structure and characteristics described further in Section 3.3 make up the available resources to be of use in combination with the theoretical background of Cyber-physical systems and falsification described in Chapter 2 to form the resulting HIL falsification framework described in Chapter 4.

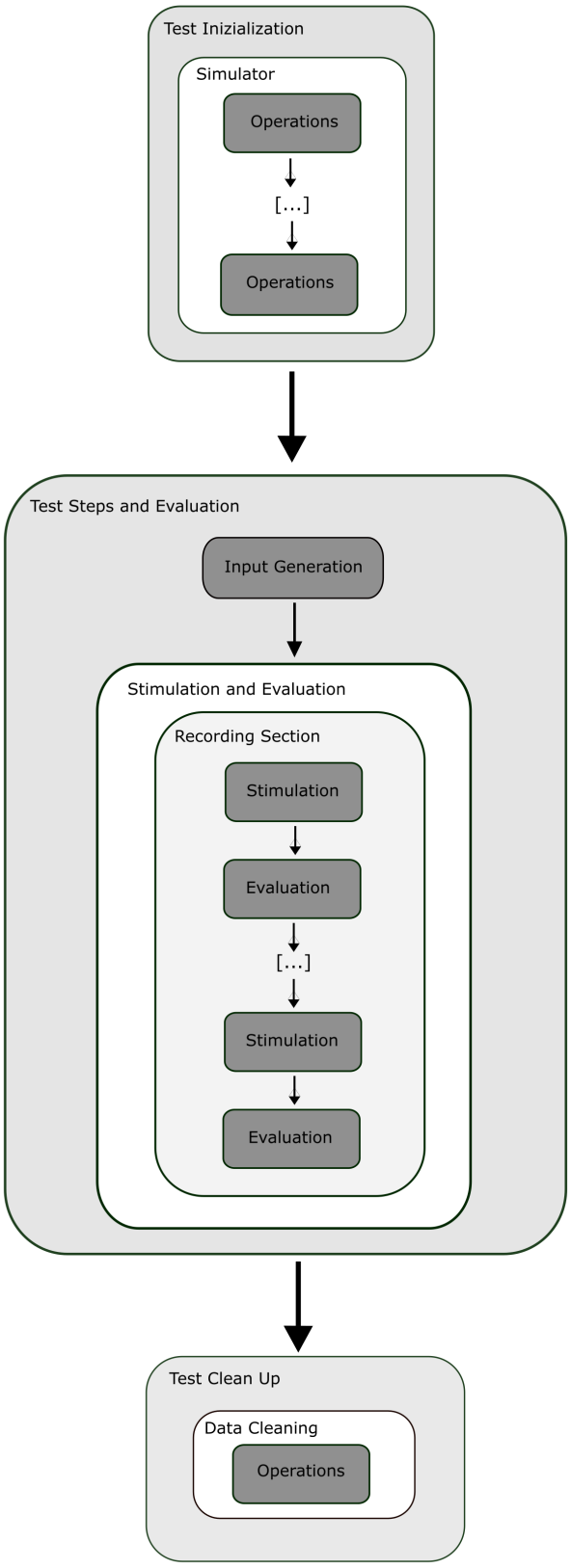


Figure 3.4: Test case flowchart for traditional HIL approach.

4

HIL Falsification Framework

In this chapter we present the resulting framework using HIL falsification. We explain the overview of the framework, what is necessary to perform HIL Falsification and the outcome of the automated testing.

4.1 Falsification in HIL testing

We present a framework on how to apply falsification in HIL testing. As an overview, there are a couple of steps considered in the process. The process along with the considerable stages are shown in Figure 4.1 and explained further upon.

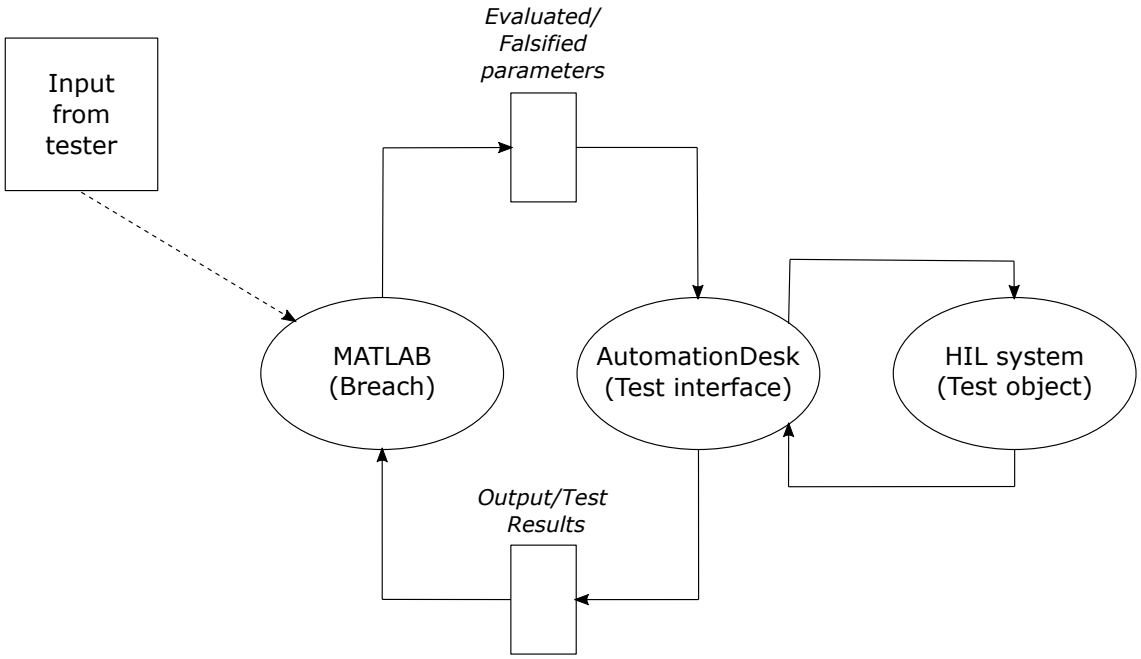


Figure 4.1: Flow chart on how to connect the falsification procedure along with the HIL system.

4.1.1 Input from tester

The outcome which is desired from verifying and validating system behaviour is dependant on the input signals from the very beginning of the test. To be able to emulate as real-world like behaviour as possible it is necessary for the user, most

often a tester, to act with the systems. The test engineer is the one to decide upon appropriate specifications and ranges of signal values. As mentioned in section 3.2.3 the amount of specifications and requirements is also decided by the tester. Customary would be to not create complex specifications just because one can and to create specifications without certain test cases in mind. Proper specifications created should be fairly easy to use in any test case.

4.1.2 Specifications and evaluation

The specifications are written based on the requirements suitable for testing. To suit in this framework they should be implemented as STL formulas in the MATLAB toolbox Breach [1]. Such specifications are thus taken into account depending on how the user designed them. Certain functions in Breach makes it possible to handle the input data and create the requirement objects in MATLAB and pass those as falsification problem objects and solve them. The following steps are done to solve the problem:

1. Set the parameter values and their domains and types
2. Generate the signal with the parameter data
3. Create the requirements with the signals as the input data
4. Create a falsification problem object with the specified requirements
5. Run the object in the solver
6. The solution of the falsification problem is then passed as the evaluated parameters

The parameters which are evaluated using falsification are then sent from this stage to the test interface in a format suitable for both instances. This stage is acting as a pre-processing- and post-processing stage of the acquired data from the test results. It is the act of pre-processing where the very first set of parameters is the prior setup to the test interface. The post-processing is then done when the output parameters from the test results are used as input data for a new falsification problem to be created and solved.

4.1.3 Test interaction

The test is executed using AutomationDesk as the interface and block-based testing for the test cases as described in Section 3.2.5. In traditional HIL, without using falsification, this part would not be dependant on the evaluated parameters from the earlier stage. With HIL falsification, this stage makes use of the evaluated parameters for the reference signals used in the blocks of the executed test case. It is important that this kind of interface is connected properly, so that the correct variables are called upon and loaded onto the corresponding signals. As such, it have to be specified explicitly which parameters are corresponding to which signals. For Signal-based testing, the signals can be determined and created from the parameter values into segments.

4.1.4 Test result from HIL

When the test is executed on the HIL, the environment in which the test object is emulated and produces test results as by traditional means. That is, the output data is stored when the test is completed and if AutomationDesk is used, the output can be directly handled and displayed to the tester using the interface, just like when running automated tests without using falsification. The desired outcome should be an expected test result, meaning it should provide quantities that the user or the system can use to determine whether the test has passed or not. The data for verification and/or validation is part of the output and the user is able to analyze the status in a report generated by AutomationDesk. The output from the test result is sent from the test interface back to Breach to be re-evaluated and eventually goes through the falsification procedure anew, thus closing the loop. As the framework is designed today, the restrictions made are those that are implemented by the testers as the input. The specifications implemented are based upon pre-defined requirements by the tester and there are no implemented requirements making restrictions upon the output as the test result. The test result have to be used as input data for the new specifications and in turn falsification problem as they are fed back through the test interface and then MATLAB.

4.1.5 Practical demands

The equipment which is needed to implement and perform falsification in HIL testing with this particular HIL falsification framework is:

- A computer with MATLAB and the Breach toolbox as well as AutomationDesk installed
- HIL system
- Test object
- Communication (I/O)

Along with the equipment it is necessary for the input to be suitable and relevant to the test. It is therefore crucial for the user to have understanding of both how the test execution works and the test object itself to affect the test automation and create meaningful test cases.

4.2 Framework combining falsification and HIL testing

As mentioned in Section 1.4, MIL and SIL solutions together with falsification technique have been successfully implemented for testing CPSs at Volvo Cars. However, combining HIL solution with falsification to implement automated test generation has not yet been achieved, since moving from the first two integration levels, MIL and SIL, to HIL is a difficult and time-consuming process. Among the main complications of this transition are:

- **Use of hardware**: this use of hardware implies considering aspects that in previous integration levels were not considered, e.g., its safety or the way to stimulate the system, i.e., generate input signals.

- **Change of the working environment**: the addition of hardware to the test environment causes in turn the working environment to change. For example, it is necessary to understand the operation of the HIL simulator and the new way of generating use cases through tools such as Automation Desk.

The following presents a general picture of how HIL testing can be combined with falsification for verifying and validating CPSs in an industrial environment. As specified in Section 3.3, the precise information we have worked with cannot be used because it is proprietary to Volvo Cars. Consequently, the whole discussion is based on the adaptive cruise control example proposed in Section 3.3.1.
The development of the new framework proposed here involves defining the concepts, practices, and criteria necessary to approach CPSs HIL testing from a perspective where falsification is the cornerstone. Therefore, the general structure proposed for the test cases is described in detail next, as well as a new method for the characterization and definition of the input signals to match the presented approach. Likewise, it is described how the specifications have to be expressed and evaluated, considering the new structure. Finally, the necessary process to implement the proposed framework is explained.

4.2.1 Test case structure

Figure 4.2 shows the structure of a general test case proposed for the validation and verification of CPSs using HIL testing in conjunction with falsification technique. As can be verified by going back to Figure 3.4, the structure is similar to the one introduced for the validation and verification of CPSs using a traditional HIL approach. The underlying cause is that the testing process comprises standardized steps that cannot be interchanged or deleted. The proposed structure is also divided into three sections: *Test Initialization*, *Test Step and Evaluation*, and *Test Clean Up*.
Section 1, *Test Initialization*, has the same purpose as previously described: it initializes the simulation environment. Its characteristics match those already explained in Section 3.3.2. Section 2, *Test Step and Evaluation*, has a similar structure to the one analyzed for the traditional HIL testing. This section groups together all operations that stimulate the test object. However, it should be emphasized that all *Evaluation* blocks, i.e., those used to check requirements, have been suppressed. This is because using this new approach specifications are checked at the end of the test case execution. There is only one exception to this: an *Evaluation* block is used to terminate the test case if non-compliance with a specification compromises the test object, e.g., it could be dangerous if the maximum allowable voltage is exceeded and the protections are not triggered. Finally, section 3, *Test Clean Up*, clears all previously settings and sets values back to preset so that the ECU is set up to its default state.
As the reader might have figured out, there are not many changes needed in the general structure of test cases to move from a traditional HIL approach to one where this traditional HIL approach is complemented with the falsification technique. Therefore, it is possible to work with test cases already developed by test engineers at Volvo Cars, subtly modifying them from a falsification perspective.

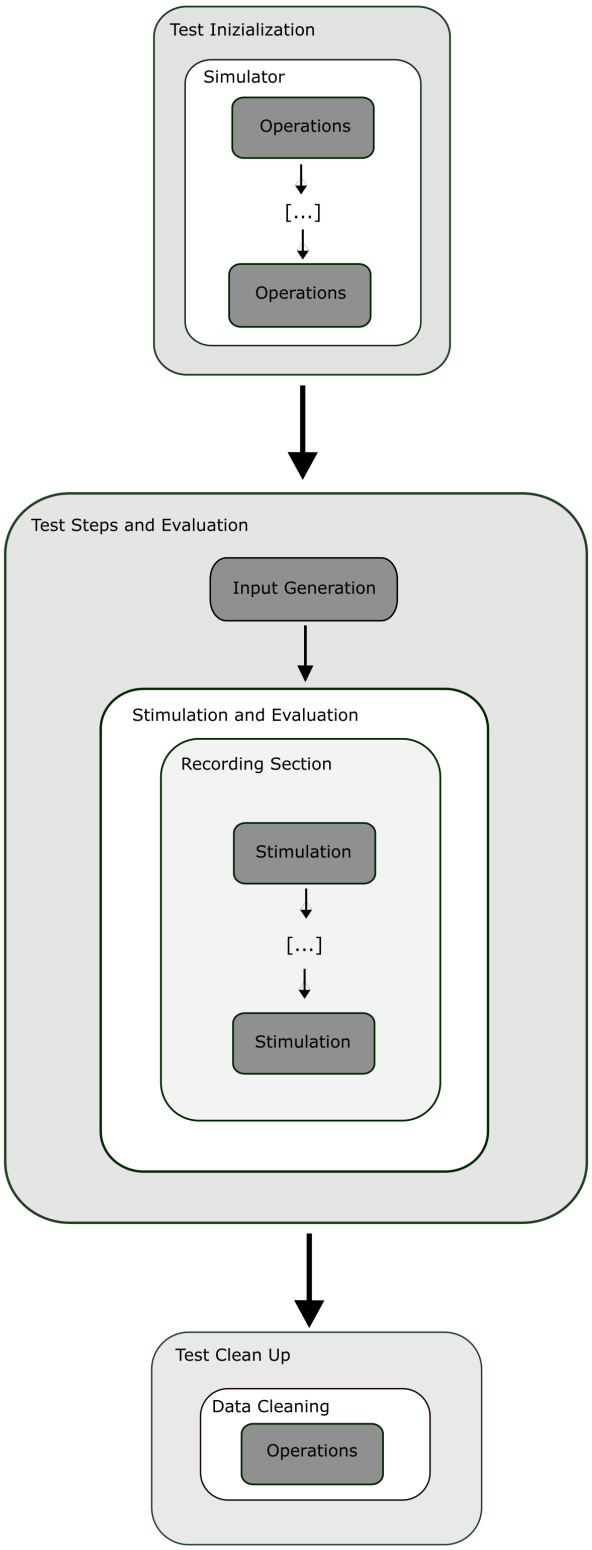


Figure 4.2: Test case flowchart proposed for the combination of falsification technique and HIL environment.

4.2.2 Automated input signal generation

The successful implementation of the proposed framework brings a new challenge: the automated input generation. To overcome this challenge, suggested steps are presented and explained below. Unlike what was previously explained in Section 3.3, where it was stated that test engineers are in charge of parameterizing, i.e., defining input signals, each time a test case is run, the adoption of this new testing approach allows the automated generation of inputs and, consequently, the automated test generation. However, it requires prior work.
It is necessary to correctly parameterize all input signals, which means: defining the type of curve representing the signal, i.e., whether it is, for example, a constant, a sinusoid, or a ramp; deciding which signal parameters can be changed, for example, in the case of a sinusoidal signal the amplitude, period, or offset; and defining the interval within which the parameters can be varied. For this purpose, a thorough knowledge of the system under test is required.
Going back to the example in Section 3.3.1, where the inputs signals are *Set_Speed* and *Set_Distance*, and considering the first one, *Set_Speed* input. Here, the signal is of type constant since the selected cruise speed is desired to remain constant during the whole execution of the test case. Since it is a constant signal, the only parameter that can be varied is the value of the signal. The allowed range of the *Set_Speed* input is [60,130], based both on the hardware knowledge and common sense. The lower bound has been defined considering the advice not to allow the use of the adaptive cruise control on urban roads, where the maximum speed permitted is 50 km/h. The upper bound has been specified based on the maximum speed allowed on freeways. Therefore, it is logical that test cases are executed using values within this range. In this particular case, testing out of range is not a major drawback, as it only results in unnecessary time consumption. However, in more complex use cases, it can cause serious damage to the test object, so a thorough knowledge of the hardware's behavior is required.

4.2.3 Definition of requirements

As already discussed in Chapter 2, to apply falsification, system requirements need to be expressed in STL. For this reason, it is essential to transform the specifications from natural language to STL. It is noteworthy that this conversion and, in short, writing good specifications in STL is difficult, as derived from [43]. For this reason, it requires practice and the help of experts in the field.
The use case introduced in Section 3.3.1 is used to exemplify how this natural language to STL transformation process is carried out. To do so, it is necessary to previously define which signals are used as inputs and outputs and give them a unique name. In our case the inputs signals are *Set_Speed* and *Set_Distance* and the outputs are *ECUs_Communication* and *Ego_Speed*. As a reminder, specifications written in natural language are introduced again:

1. The braking system should activate within the first 10 seconds after a steep downhill slope (>10 degrees) is detected.
2. The actual speed must be equal to the speed of the car in front after 5 seconds of the detection of the car ahead and during the following 20 seconds.

Following Section 2.1.3, specifications in STL language are

1.

$$\varphi_1 = \Diamond_{[0,10]}(ECUs\_Communication == 1), \tag{4.1}$$

Specification 1, φ_1 , corresponds to the fact that eventually, $\Diamond$, i.e., at least once, during the interval from 0 to 10 seconds of simulation, [0,10], the output signal, *ECUs_Communication*, takes the desired value, 1. And

2.

$$\varphi_2 = \Box_{[5,25]}(Ego\_Speed == Speed\_Ahead), \tag{4.2}$$

Specification 2, φ_2 , captures the fact that always, $\Box$, during the interval from 5 to 25 seconds of simulation after detecting a car ahead at a lower speed, [5,25], the output signal, *Ego_Speed*, has to be equal to the desired value, *Speed_Ahead*.

4.2.4 Evaluation of requirements

The requirements evaluation is not done online, i.e., during the test case execution, as in the traditional HIL approach. The suggested framework requires, first, recording the data of the signals of interest. Then, when the test case has finished, the specifications are evaluated offline using the recorded data. For this reason, the evaluation blocks proposed in Section 3.3.2 have been suppressed from the test case structure. However, this implies a new challenge, the synchronization of the evaluation process, which needs to be addressed.

Again, for an easier understanding by the reader, we return to the example proposed in Section 3.3.1. Specification 1 checks whether the ACC controller communicates with the brake controller within the first 10 seconds after detecting a steep slope. The first part of the statement is captured in Equation 4.1. However, the current definition would lead to evaluating the specification in the first 10 seconds of the test when, in fact, it is required to check the specification within the first 10 seconds after detecting a steep downhill slope. Since the evaluation is offline, it is necessary to specify when the system is excited, so the analysis of the signal of interest, in this case, *ECUs_Communication*, begins at that moment. For this purpose, an auxiliary signal is used to monitor when the signal warning of a steep downhill slope is detected. In our case, it is *Aux_Slope*. With this, the specification completely defined in STL is

1.

$$\varphi_1 = \Box(Aux\_Slope == 1) \Rightarrow \Diamond_{[0,10]}(ECUs\_Communication == 1), \tag{4.3}$$

Specification 1, φ_1, details that always, $\Box$, the auxiliary signal, *Aux_Steep_Slope*, is equal to the desired value, 1, it will be evaluated that eventually, $\Diamond$, the output signal, *ECUs_Communication*, is equal to the desired value, 1, during the first 10 seconds after detecting a steep downhill slope, [0,10].

The same holds for specification 2. In this case, the auxiliary signal is *Aux_Ahead* and informs when a car ahead is detected at a speed slower than the set cruise speed. Specification 2 in STL is

2.

$$\varphi_2 = \Box(Aux\_Ahead == 1) \Rightarrow \Box_{[5,25]}(Ego\_Speed == Speed\_Ahead), \tag{4.4}$$

Specification 2, φ_2, indicates that always, $\Box$, the auxiliary signal, *Aux_Ahead*, is equal to the desired value, 1, it will be evaluated that the output signal, *Ego_Speed*, is always, $\Box$, equal to the desired value, *Speed_Ahead*, starting 5 seconds after detecting the car in front, and finishing after 20 seconds of evaluation, [5,25].

4.2.5 Implementation

As discussed in Section 4.1.5, implementing this new framework requires a HIL simulator, a test object, an I/O communications module, and some software. In our case, AutomationDesk and Breach toolbox for developing test cases and implementing falsification, respectively.
Following instructions given in Section 4.2.2, falsification tools allow defining the inputs sent to the HIL simulator. The signal proposed in Section 4.2.2 is characterized as follows. First, the signal to be defined is specified through the unique name previously chosen. Next, the parameter to be varied and its data type are specified and, finally, the interval in which the parameter is to be varied.
According to rules described in Section 4.2.3 and 4.2.4 for defining specifications, falsification tools, introduced in Section 2.3, are used to evaluate these specifications. Based on what has been previously stated in Section 2.1.3 and Equation 2.3 these tools compute their robustness value.
By defining the inputs and specifications and their subsequent evaluation, these tools solve the falsification problem. For this purpose, as discussed in Section 2.2, an approach based on optimization techniques is used through different algorithms that these tools incorporate. Based on the optimization problem result, they automatically generate parameters that allow minimizing the robustness value so a counterexample can be found.
Finally, a tool that communicates the software used to solve the falsification problem, in our case the Breach toolbox, and the program used to develop the test cases, Automation Desk, is needed. Therefore, a MATLAB script has been developed at Volvo Cars to serve as an interface between these two tools. With this implementation of the framework in mind, conclusions of the HIL framework and the use of the HIL systems for automated test generation and execution are presented in Chapter 5.

5
Conclusion and Future Work

As systems become more complex and larger, combining both cyber and physical properties, their validation and verification turn into a growing challenge. An optimal validation and verification of such systems would reduce time and costs and increase the confidence in the systems. This master's thesis proposes a new framework for combining the falsification technique and HIL testing to implement automated test generation in an industrial environment.
The first part of the study carried out in this master's thesis examines the verification and validation method followed by Volvo Cars. From this analysis, it is clear that a case-by-case basis research is necessary to decide which use cases are optimal for the framework presented. Especially, aspects related to the hardware safety and the use case adaption to an automated test generation approach have to be considered. Furthermore, it is inferred that it is not necessary to make significant changes to the test cases developed by Volvo Cars test engineers. Hence, they can be used as a basis for the new implementation approach. It has also been found that systems requirements evaluated in Volvo Cars are mostly Boolean. This means that no interesting robustness value can be obtained from their evaluation, which makes the final implementation difficult.
The second part of this master's thesis consists of developing a new framework to verify and validate CPSs. This new framework supports automated test generation by integrating falsification and HIL techniques. The results presented above and the theoretical knowledge explained in Chapter 2 have been taken as a reference for this purpose. A general structure that should be followed to develop test cases in this new framework has been proposed. The result is a similar structure to that of the method implemented in Volvo Cars but slightly changed with a falsification perspective, only evaluation blocks have been removed. Guidelines for a correct definition of the input signals are also provided. This rules explain how to correctly parameterize the signals and mention the precautions to consider when defining them. The most significant advances are presented in the definition and evaluation of the requirements. It is explained how to transform specifications into STL and how to synchronize their evaluation. The result is a theoretical guide that allows to implement this new automated test generation technique in an industrial environment.

5.1 Answers to research questions

With the help of the approach and methodology in chapter 3 and the project results in Chapter 4 we provide answers to the research questions stated in Section 1.6.

Question 1 - How can falsification of CPSs be used in an industrial environment, for automated test generation of electrical automotive HIL systems?

Answer to Question 1 - The theoretical background of falsification as well as CPSs are described in chapter 2. In chapter 3 the HIL systems of use is described and how testing works traditionally on the HIL setup in the automotive industry. From the information provided in these parts it is possible to grasp how manual and automated testing can be used in an industrial environment. However, with the work done in this thesis, the results in chapter 4 provide foundation of how automated test generation of electrical automotive HIL systems can be performed. Regard the HIL system as an example of a CPS since it contains the collections of computer components and physical components that are integrated with each other to operate the test process. The test objects are the embedded systems used in a electrical automotive application. The substantial work here being falsification which is performed with the help of the Breach toolbox. With the addition of falsification it is used as a helpful method to generate automated test cases and potentially even optimize current ones.

Question 2 - How should automated test generation based on falsification be used in conjunction with traditional manual testing?

Answer to Question 2 - In Chapter 3, the description of HIL systems is provided and how they are used for traditional manual testing. In Chapter 4 the resulting framework shows how automated test generation can be done. The intention of this question is to provide comparisons between both ways of testing and establish how they should be used together. As one, might have recognized from the thesis work, one way of testing does not necessarily neglect the other. That means that there is still a value to use both manual testing as well as automated test generation and the choice of which test method is dependant on complexity of the test.

In terms of tests where there is simply one operation or activity to be made and then compare it with a Boolean outcome *True* or *False* (verified *OK* or *not OK*), falsification does not necessarily improve anything. With the framework, it is still possible to generate new tests, but the optimization will not produce anything particular. It can essentially be simple mode handling and expectations are then just based on a threshold. It does not necessarily have to be dependant on the time it takes for the test to be executed. An example can be where a ECU is simply powered on and set to its default standby mode, then do a test where it is put to sleep mode and immediately back again to its standby mode.

On the other hand, besides the tests that are just mode handling, there are those that are dependant on functions of time and changing signal values. The falsification problem is constructed to find values that lead the system to the violation of a requirement. Tests which contain functions of changing values until a requirement is met, is suitable for falsification. An example is a test where a variable of a signal is not just set and then the variable attains that value, but where the value is increasing or decreasing. It can have ramp-up/ramp-down, sinusoidal, or periodic behaviours. The value after increasing

or decreasing is time dependent and since it is executed in real-time, the time it takes is equal to the time it would take in the real world. Requirements should be written for this and processed with falsification to find out at which value and also when it violates.

To summarize, automated test generation based on falsification should be used for tests with some more dynamics to them in terms of increasing or decreasing values dependant on time. It can also be used for comparisons between two signals to find out at which value and when they no longer hold according to the requirement. Manual testing is well-founded for simple mode handling where the actual value is not of interest, just if it passed a threshold or not.

Question 3 - What kind of specifications are suitable to use for the kind of automotive HIL systems considered in the thesis as compared to typical simulation-based testing?

Answer to Question 3 - (The answer might be a bit coherent with the one in RQ2)

In general, the specifications have to be written in accordance to the test object and the HIL system. The purpose of the specifications is to create tests which can be executed on the test object to simulate and produce test results of desired real world behaviours. One key difference her between the HIL system testing and typical simulation-based testing is the presence and use of hardware. In simulation-based testing, there is not necessarily an exact model of physical hardware to consider. It is possible to experiment to another extent and create arbitrary environments. There can therefore be difficult to create simulation conditions in which the outcomes of the test is observable correct in accordance with the real world expectations. For the HIL systems, there are other prerequisites. The test object is specifically modelled and connected to with the I/Os. The actual connection is not important here, but rather to understand that the test object of the HIL system affects what kind of specifications are suitable for the application.

One should thus take the test object into account when writing specifications for the test which will be executed on the HIL system. Consider the role of the test object and try to make it mimic real world behaviours in order to produce proper test results. Specifications which affect the test to only produce results which are verified good are not suitable, and vice versa, specifications which affect the test to only produce results which are verified *not* good are also not suitable. Suitable specifications are those that are written to cope with the systems and contain requirements after behaviours which can occur in the real world.

The considered specifications in the thesis are written in STL and implemented in the Breach toolbox. They contain sets of parameters which are used in the test cases and so that connection between the computer model and the test object is upheld. Such specifications are suitable and are of use in the HIL framework provided from this thesis.

5.2 Ethics and sustainability

The interest of manufacturing electrical vehicles is spread globally and on international levels. The development of the end-products, for which the test objects of the HIL systems are tested, are making ethical impacts on different levels.

On one side it is a questions of automation depth of tasks in the job of testing with HIL systems. Questions may arise if automated test generation and execution is able to replace human actions. It is necessary to take into account the tasks of the testing and if they can be replaced by systems instead of human labour, eventually leading to reducing human actions. The intent is to generate test cases so that they do not have to be manually created and handled. The automated test generation should as such be deemed as a support for test engineers in their role of testing and developing. Time can be saved for the test engineers to spend on other tasks to improve testing and development of the electrical vehicles. As inputs are still needed from test engineers or users, the automated test generation of HIL systems is not considered a full-scale replacement of engineering duties. Automated testing should be used as a complementary to present ways of testing, which the framework provided by the thesis does. Other than comparing if the HIL system is able to impact the test engineers tasks, it is also noteworthy to think of costs and the energy consumed for running tests in a HIL system. No such study has been made in the thesis of determining whether it is cheaper or not to change and run a HIL system instead of manually performing tests. The HIL systems of use are already in place and are not modified in this thesis, though with time such HIL systems can also be updated and so affect costs. It is worth to consider the uptime of the HIL systems in comparison to human working hours. The uptime of testing where a human is needed to run test would be affected by numerous factors, for example working hours. The HIL system with automated test generation and execution can in practice run just as well during uncommon work hours and weekends. In the long run, this may in turn lead to more extensive testing and eventually produce more data to fix found errors and bugs at a more rapid pace.

In another chain of events is the increasing volumes of electrical vehicles to be developed and manufactured which the testing using HIL system indirectly impacts. Although electrical vehicles are seen as another alternative to fossil fuel driven vehicles there are differences between these products in terms of materials and resources, noticeably the material for batteries. Some resources used in modern day vehicles are limited. There may also be a question of where the resources are gathered and how, that is what methods and if it affects people at or around the source of the resource. It is noteworthy to take into account if HIL systems using automated test generation can replace other test equipment or complete vehicles, acting as complete test objects. There might be a chance of decreasing development costs and usage of limited resources.

5.3 Future work

Further development of this project could lead to certain improvements, culminating in a well-defined framework that can be followed by test engineers at Volvo Cars. Possible future work is presented below:

1. **Framework implementation**: This work could not be implemented on the HIL simulators, as initially thought. In the future, it will be necessary to carry out this experimental implementation, which in turn will allow the proposed framework to be completed and optimized. For this, it will be necessary to work together with the test engineers and receive their feedback.
2. **Framework Design**: As the framework is designed today, it consists of the three main stages including falsification and evaluation of specifications performed in Breach, Test interaction performed in AutomationDesk, and test execution on the test object connected to the HIL system. In addition to these we presented an initial stage where input from the tester is required in Section 4.1.1. It is required from the tester to implement the restrictions in form of requirements from the very beginning of initializing test case generation. It is mentioned in Section 4.1.4 that no restrictions are made upon the output from the test execution. A future improvement could be to consider restraining the outputs and check for valid constraints of the test output before it is sent back to the first stage of specification evaluation and falsification. This would be an act of safety, ensuring from the test results if the test has passed with parameters staying within safe thresholds.
3. **Broaden the study**: The analysis of the method implemented in Volvo Cars and the subsequent development of the new framework have been carried out based on four use cases of two systems: EDV and BECM. More use cases are necessary to study the method implemented in Volvo Cars.
4. **Signal Based Testing**: up to now, input signals have been defined following a rigid style, i.e., the type of curve representing the signal is always the same for the whole simulation time. However, with this novel Signal-Based Testing approach, the input signals can be defined by dividing them into segments. Each of these segments can be parameterized independently so each can have a different type of curve, different type of parameters to be varied, and different values for those parameters. Implementing this new variant would be a step forward in the validation and verification of CPSs since it would allow the generation of very creative test cases.

Bibliography

[1] Alexandre Donzé. "Breach, A Toolbox for Verification and Parameter Synthesis of Hybrid Systems". en. In: *Computer Aided Verification*. Ed. by Tayssir Touili, Byron Cook, and Paul Jackson. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2010, pp. 167–170. ISBN: 978-3-642-14295-6. DOI: 10.1007/978-3-642-14295-6_17.

[2] Volkan Gunes et al. "A Survey on Concepts, Applications, and Challenges in Cyber-Physical Systems". eng. In: *KSII Transactions on Internet and Information Systems (TIIS)* 8.12 (2014). Publisher: Korean Society for Internet Information, pp. 4242–4268. ISSN: 1976-7277. DOI: 10.3837/tiis.2014.12.001. URL: https://www.koreascience.or.kr/article/JAKO201403760397435.page (visited on 02/07/2022).

[3] Jianhua Shi et al. "A survey of Cyber-Physical Systems". In: *2011 International Conference on Wireless Communications and Signal Processing (WCSP)*. Nov. 2011, pp. 1–6. DOI: 10.1109/WCSP.2011.6096958.

[4] Edward A. Lee. "Cyber Physical Systems: Design Challenges". In: *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*. ISSN: 2375-5261. May 2008, pp. 363–369. DOI: 10.1109/ISORC.2008.25.

[5] Ragunathan (Raj) Rajkumar et al. "Cyber-physical systems: the next computing revolution". In: *Proceedings of the 47th Design Automation Conference*. DAC '10. New York, NY, USA: Association for Computing Machinery, June 2010, pp. 731–736. ISBN: 978-1-4503-0002-5. DOI: 10.1145/1837274.1837461. URL: https://doi.org/10.1145/1837274.1837461 (visited on 02/10/2022).

[6] Yang Liu et al. "Review on cyber-physical systems". In: *IEEE/CAA Journal of Automatica Sinica* 4.1 (Jan. 2017). Conference Name: IEEE/CAA Journal of Automatica Sinica, pp. 27–40. ISSN: 2329-9274. DOI: 10.1109/JAS.2017.7510349.

[7] Naoufel Boulila. *Guidelines for Modeling Cyber-Physical Systems – A Three-Layered Architecture for Cyber Physical Systems*. Jan. 2017. DOI: 10.13140/RG.2.2.21599.30881.

[8] Teodora Sanislav and Liviu Miclea. "Cyber-Physical Systems - Concept, Challenges and Research Areas". In: *Journal of Control Engineering and Applied Informatics* 14.2 (June 2012). Number: 2, pp. 28–33. ISSN: 1454-8658. URL: http://www.ceai.srait.ro/index.php?journal=ceai&page=article&op=view&path%5B%5D=1292 (visited on 02/07/2022).

[9] James Kapinski et al. "Simulation-Based Approaches for Verification of Embedded Control Systems: An Overview of Traditional and Advanced Modeling,

Testing, and Verification Techniques". In: *IEEE Control Systems Magazine* 36.6 (Dec. 2016). Conference Name: IEEE Control Systems Magazine, pp. 45–64. ISSN: 1941-000X. DOI: 10.1109/MCS.2016.2602089.

[10] Jeff C. Jensen, Danica H. Chang, and Edward A. Lee. "A model-based design methodology for cyber-physical systems". In: *2011 7th International Wireless Communications and Mobile Computing Conference*. ISSN: 2376-6506. July 2011, pp. 1666–1671. DOI: 10.1109/IWCMC.2011.5982785.

[11] Rajeev Alur. "Formal verification of hybrid systems". In: *2011 Proceedings of the Ninth ACM International Conference on Embedded Software (EMSOFT)*. Oct. 2011, pp. 273–278. DOI: 10.1145/2038642.2038685.

[12] Eckard Bringmann and Andreas Krämer. "Model-Based Testing of Automotive Systems". In: *and Validation 2008 1st International Conference on Software Testing, Verification*. ISSN: 2159-4848. Apr. 2008, pp. 485–493. DOI: 10.1109/ICST.2008.45.

[13] Mark Utting, Alexander Pretschner, and Bruno Legeard. "A taxonomy of model-based testing approaches". en. In: *Software Testing, Verification and Reliability* 22.5 (2012). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/stvr.456, pp. 297–312. ISSN: 1099-1689. DOI: 10.1002/stvr.456. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/stvr.456 (visited on 03/03/2022).

[14] Anthony Corso et al. "A Survey of Algorithms for Black-Box Safety Validation of Cyber-Physical Systems". In: *Journal of Artificial Intelligence Research* 72 (Oct. 2021). arXiv: 2005.02979. ISSN: 1076-9757. DOI: 10.1613/jair.1.12716. URL: http://arxiv.org/abs/2005.02979 (visited on 04/22/2022).

[15] A. Hall. "Seven myths of formal methods". In: *IEEE Software* 7.5 (Sept. 1990). Conference Name: IEEE Software, pp. 11–19. ISSN: 1937-4194. DOI: 10.1109/52.57887.

[16] R.A. Decarlo et al. "Perspectives and results on the stability and stabilizability of hybrid systems". In: *Proceedings of the IEEE* 88.7 (July 2000). Conference Name: Proceedings of the IEEE, pp. 1069–1082. ISSN: 1558-2256. DOI: 10.1109/5.871309.

[17] Jonathan Nibert, Marc E. Herniter, and Zachariah Chambers. "Model-Based System Design for MIL, SIL, and HIL". en. In: *World Electric Vehicle Journal* 5.4 (Dec. 2012). Number: 4 Publisher: Multidisciplinary Digital Publishing Institute, pp. 1121–1130. ISSN: 2032-6653. DOI: 10.3390/wevj5041121. URL: https://www.mdpi.com/2032-6653/5/4/1121 (visited on 03/11/2022).

[18] Johan Lidén Eddeland et al. "Enhancing Temporal Logic Falsification With Specification Transformation and Valued Booleans". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39.12 (Dec. 2020). Conference Name: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, pp. 5247–5260. ISSN: 1937-4151. DOI: 10.1109/TCAD.2020.2966480.

[19] Lidén Eddeland. "Falsification of Signal-Based Specifications for Cyber-Physical Systems". en. In: (), p. 66.

[20] James Garson. “Modal Logic”. In: (Feb. 2000). Last Modified: 2018-09-08. URL: https://plato.stanford.edu/entries/logic-modal/?ref=https://githubhelp.com (visited on 04/25/2022).

[21] Amir Pnueli. “The temporal logic of programs”. In: *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*. ISSN: 0272-5428. Oct. 1977, pp. 46–57. DOI: 10.1109/SFCS.1977.32.

[22] Alexandre Donzé et al. “On Temporal Logic and Signal Processing”. en. In: *Automated Technology for Verification and Analysis*. Ed. by Supratik Chakraborty and Madhavan Mukund. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2012, pp. 92–106. ISBN: 978-3-642-33386-6. DOI: 10.1007/978-3-642-33386-6_9.

[23] Alexandre Donzé and Oded Maler. “Robust Satisfaction of Temporal Logic over Real-Valued Signals”. en. In: *Formal Modeling and Analysis of Timed Systems*. Ed. by Krishnendu Chatterjee and Thomas A. Henzinger. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2010, pp. 92–106. ISBN: 978-3-642-15297-9. DOI: 10.1007/978-3-642-15297-9_9.

[24] Oded Maler and Dejan Nickovic. “Monitoring Temporal Properties of Continuous Signals”. en. In: *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*. Ed. by Yassine Lakhnech and Sergio Yovine. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2004, pp. 152–166. ISBN: 978-3-540-30206-3. DOI: 10.1007/978-3-540-30206-3_12.

[25] Bardh Hoxha, Houssam Abbas, and Georgios Fainekos. “Benchmarks for Temporal Logic Requirements for Automotive Systems”. en. In: (), p. 6.

[26] Vasumathi Raman et al. “Reactive synthesis from signal temporal logic specifications”. en. In: *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*. Seattle Washington: ACM, Apr. 2015, pp. 239–248. ISBN: 978-1-4503-3433-4. DOI: 10.1145/2728606.2728628. URL: https://dl.acm.org/doi/10.1145/2728606.2728628 (visited on 05/08/2022).

[27] Georgios E. Fainekos and George J. Pappas. “Robustness of Temporal Logic Specifications”. en. In: *Formal Approaches to Software Testing and Runtime Verification*. Ed. by Klaus Havelund et al. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2006, pp. 178–192. ISBN: 978-3-540-49703-5. DOI: 10.1007/11940197_12.

[28] Alexandre Donzé, Thomas Ferrère, and Oded Maler. “Efficient Robust Monitoring for STL”. en. In: *Computer Aided Verification*. Ed. by David Hutchison et al. Vol. 8044. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 264–279. ISBN: 978-3-642-39798-1 978-3-642-39799-8. DOI: 10.1007/978-3-642-39799-8_19. URL: http://link.springer.com/10.1007/978-3-642-39799-8_19 (visited on 05/08/2022).

[29] Jyotirmoy V. Deshmukh et al. “Robust online monitoring of signal temporal logic”. en. In: *Formal Methods in System Design* 51.1 (Aug. 2017), pp. 5–30. ISSN: 0925-9856, 1572-8102. DOI: 10.1007/s10703-017-0286-7. URL: http://link.springer.com/10.1007/s10703-017-0286-7 (visited on 05/08/2022).

[30] Gidon Ernst et al. “ARCH-COMP 2021 Category Report: Falsification with Validation of Results”. en-US. In: *EPiC Series in Computing*. Vol. 80. ISSN: 2398-7340. EasyChair, Dec. 2021, pp. 133–152. DOI: 10.29007/xwl1. URL: https://easychair.org/publications/paper/F4kf (visited on 02/23/2022).

[31] Yashwanth Annapureddy et al. *S-TaLiRo: A Tool for Temporal Logic Falsification for Hybrid Systems*.

[32] Amela Ajanovic. “The future of electric vehicles: prospects and impediments”. en. In: *WIREs Energy and Environment* 4.6 (2015). _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/wene.160, pp. 521–536. ISSN: 2041-840X. DOI: 10.1002/wene.160. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/wene.160 (visited on 05/02/2022).

[33] R. Isermann, J. Schaffnit, and S. Sinsel. “Hardware-in-the-loop simulation for the design and testing of engine-control systems”. en. In: *Control Engineering Practice* 7.5 (May 1999), pp. 643–653. ISSN: 0967-0661. DOI: 10.1016/S0967-0661(98)00205-6. URL: https://www.sciencedirect.com/science/article/pii/S0967066198002056 (visited on 02/23/2022).

[34] A. Bouscayrol. “Different types of Hardware-In-the-Loop simulation for electric drives”. In: *2008 IEEE International Symposium on Industrial Electronics*. ISSN: 2163-5145. June 2008, pp. 2146–2151. DOI: 10.1109/ISIE.2008.4677304.

[35] Bo Li et al. “Study on HIL system of electric vehicle controller based on NI”. en. In: *IOP Conference Series: Materials Science and Engineering* 382 (July 2018), p. 052033. ISSN: 1757-8981, 1757-899X. DOI: 10.1088/1757-899X/382/5/052033. URL: https://iopscience.iop.org/article/10.1088/1757-899X/382/5/052033 (visited on 03/18/2022).

[36] *Number of automotive ECUs continues to rise eeNews Automotive*. en-US. May 2019. URL: https://www.eenewsautomotive.com/en/number-of-automotive-ecus-continues-to-rise/ (visited on 05/09/2022).

[37] Ryosuke Okuda, Yuki Kajiwara, and Kazuaki Terashima. “A survey of technical trend of ADAS and autonomous driving”. In: *Technical Papers of 2014 International Symposium on VLSI Design, Automation and Test*. Apr. 2014, pp. 1–4. DOI: 10.1109/VLSI-DAT.2014.6834940.

[38] H.F. Othman et al. “Controller Area Networks: Evolution and Applications”. In: *2006 2nd International Conference on Information Communication Technologies*. Vol. 2. Apr. 2006, pp. 3088–3093. DOI: 10.1109/ICTTA.2006.1684909.

[39] Ashwini S. Shinde and Prof Vidhyadhar B. Dharmadhikari. *Controller Area Network for Vehicle Automation*.

[40] J Bélanger and P Venne. “The What, Where and Why of Real-Time Simulation”. en. In: (), p. 13.

[41] Herbert Schuette and Peter Waeltermann. “Hardware-in-the-Loop Testing of Vehicle Dynamics Controllers – A Technical Survey”. In: *SAE Transactions* 114 (2005). Publisher: SAE International, pp. 593–609. ISSN: 0096-736X. URL: https://www.jstor.org/stable/44682469 (visited on 05/09/2022).

[42] Jiaqi Ma et al. “Hardware-In-The-Loop Testing of Connected and Automated Vehicle Applications: A Use Case For Cooperative Adaptive Cruise Control”.

In: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. ISSN: 2153-0017. Nov. 2018, pp. 2878–2883. DOI: 10.1109/ITSC.2018.8569753.

[43] Adel Dokhanchi, Bardh Hoxha, and Georgios Fainekos. "Metric interval temporal logic specification elicitation and debugging". en. In: *2015 ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE)*. Austin, TX, USA: IEEE, Sept. 2015, pp. 70–79. ISBN: 978-1-5090-0237-5. DOI: 10.1109/MEMCOD.2015.7340472. URL: http://ieeexplore.ieee.org/document/7340472/ (visited on 05/30/2022).

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY