

Ph.D. Thesis



**Czech
Technical
University
in Prague**

F3

Faculty of Electrical Engineering
Department of Measurement

Methods for Verification and Validation of Automotive Distributed Systems

Ing. Jan Sobotka

Electrical Engineering and Information Technology
Branch of study: Measurement and Instrumentation

August 2017

Supervisor: doc. Ing. Jiří Novák, Ph.D.

Acknowledgement / Declaration

Many people helped me during a long time of writing of this thesis. I would like to thank all of them. Some of them are my supervisor for the guidance, our lab crew for fruitful discussions, and a number of great students which helped me a lot with the implementation part of this work. Also, I would like to thank my girlfriend Radka for the patience and my family for the support, since the time of my studies was quite longer than expected.

The thesis is dedicated to my grandmother Anna, because she always said, that I will be a great doctor. *(smile)*

I declare that I carried out this doctoral thesis independently, and only with the cited sources, literature and other professional sources.

In Prague 30. 8. 2017.

.....

Abstrakt /

Stále narůstající složitost automobilových elektronických systémů vytváří poptávku po vhodných validačních a testovacích metodách. Směrování automobilového průmyslu k plně autonomním vozidlům tento trend dále podporuje. Cílem práce je rozšířit množinu dostupných testovacích metod v několika oblastech. První takovou oblastí jsou měřicí metody pro komunikační sběrnici FlexRay. V porovnání se staršími standardy CAN a LIN je radič této sběrnice konfigurován mnohem větším množstvím parametrů. Pro zajištění správné funkce systému je nutné ověřit, že aktuální hodnoty těchto parametrů odpovídají hodnotám požadovaným. Tento úkol vyžaduje návrh měřicích metod schopných identifikovat skutečné hodnoty parametrů. Další oblastí vyžadující doplnění stávajících testovací možností je integrační testování automobilové elektroniky. S rostoucím počtem elektronických řídicích jednotek začíná být stávající způsob testování pomocí ručně navržených a implementovaných testovacích sekvencí nedostatečný.

Práce se zabývá automatizací generování těchto testovacích sekvencí s využitím principů Model-based testování. Testovací sekvence jsou generovány z modelů specifikovaných časovanými automaty. Součástí práce je návrh tohoto inovativních testovacího konceptu. Řešení je následně implementováno ve formě testovacího nástroje Taster a také je představena nová HIL testovací platforma založená na modulárním hardwaru firmy National Instruments. Ověření této metody je provedeno formou dvou případových studií. První demonstruje metodu na problému testování systému bezklíčového zapalování, druhá potom testuje reálný systém otvírání pátých dveří automobilu.

Klíčová slova: Testování, FlexRay, Parametry, Automobilová, Elektronika

Abstract /

The growth of complexity together with the severity of the potential failure of electronic systems in modern vehicles create pressure on development of powerful validation and testing techniques. This thesis is trying to fill few identified gaps in this field. One of the gaps is the absence of suitable measurement methods for relatively new automotive communication system called FlexRay. In comparison with older communication standards CAN and LIN, each FlexRay communication controller is parametrized by almost hundred of parameters. From the validation point of view, it is necessary to evaluate that the parameters programmed into communication controller are in conformance with the specified parameters values. Such evaluation is not possible without suitable methods capable of measuring the actual parameters values. Another blank space which this thesis is trying to fill is related to the Integration testing of automotive electronics systems. Due to increasing number of electronic control units and number of implemented functions, traditional approach incorporates, the manual test case design is reaching its limits.

This thesis proposes several improvements for this specific domain. Namely it is testbed platform based on National Instruments products, a novel approach based on the Model Based Testing techniques using the Timed Automata specification, several test generation algorithms, and implementation of the proposed concept. The presented solutions were experimentally evaluated. The FlexRay measurement methods were validated on the real communication network. Moreover, in the case of Integration testing, the implemented concept was proved by two case studies. The objective of the first one was the car keyless access system. The second study validated the described solution on the automatic trunk opening system.

Keywords: Automotive, Model, Based, Testing, FlexRay, Parameter, Evaluation

Contents /

1 Introduction	1
2 State of the Art	2
2.1 FlexRay.....	2
2.1.1 Of the Shelf Available Solutions	3
2.1.2 Related Work	4
2.1.3 Risk of Incorrect Parameterization	5
2.2 Integration Testing and MBT..	6
2.2.1 Weaknesses of Traditional Approach	7
2.2.2 Terminology.....	7
2.2.3 Taxonomy.....	9
2.2.4 Embedded System Modeling	9
2.2.5 Test Selection Criteria ..	12
2.2.6 Test Generation.....	12
2.2.7 Existing Tools	13
2.2.8 Hardware-in-the-Loop.....	14
2.2.9 Related Work	14
2.3 Summary	15
3 Thesis Objectives	17
3.1 FlexRay Objectives	17
3.2 Integration Testing Objectives.....	18
4 FlexRay	20
4.1 Overview of FlexRay Communication System	20
4.2 Parameter Set Analysis	21
4.3 Wakeup Parameters	23
4.3.1 pWakeupChannel	23
4.3.2 gdWakeupTxIdle, gdWakeupTxActive, pWakeupPattern.....	24
4.3.3 gdWakeupRxLow, gdWakeupRxIdle, gdWakeupRxWindow...	24
4.4 Startup Parameters	25
4.4.1 Type of Node	25
4.4.2 gColdStartAttempt	25
4.4.3 Collision Avoidance Symbol	25
4.4.4 pdListenTimeout	26
4.5 Evaluation of Clock Synchronization Parameters	26
4.5.1 Cycle Length Influencing and Measurement	26
4.5.2 Offset Correction Measurement.....	27
4.5.3 pdMicrotick	27
4.5.4 pClusterDrift-Damping	28
4.5.5 pRateCorrectionOut....	28
4.5.6 pOffsetCorrectionOut ..	29
4.6 Validation on Real FlexRay Network.....	30
4.6.1 FPGA FlexRay Controller	30
4.6.2 FlexRay Hardware – EUT.....	31
4.6.3 Experiments	32
4.7 Measurement Accuracy and Speed	34
4.8 Summary	35
5 Integration Testing	36
5.1 Selection of Formal Model ...	36
5.1.1 Passenger Car Inner Light Model.....	38
5.2 Timed Automata Theory	40
5.2.1 Finite-State Machine...	40
5.2.2 Automata for Infinite Input	40
5.2.3 Büchi Automaton.....	41
5.2.4 Timed Automaton	41
5.3 Testing Workflow Proposal ...	42
5.4 Testbed for Comfort Systems.....	43
5.5 System Modeling	44
5.6 Test Generation Theory.....	45
5.6.1 Graph terminology	45
5.6.2 Selected Graph Algorithms.....	47
5.6.3 Discussion of Timed tours.....	48
5.7 Algorithms.....	48
5.8 Evaluation Metrics	50
5.9 Taster	52

5.9.1	Taster Architecture	52
5.9.2	Model Parser	52
5.9.3	Model Execution	53
5.9.4	Test Adapter	53
5.9.5	Trace Logger	54
5.9.6	Implementation	54
5.9.7	User Interface	54
5.10	Case Study – KESSY	56
5.10.1	Specification	57
5.10.2	Models	57
5.10.3	SUT Implementation	59
5.10.4	Results	61
5.11	Case Study – Trunk	62
5.11.1	Specification	62
5.11.2	Experiment Plan	64
5.11.3	Models	65
5.11.4	Original Test Suite	69
5.11.5	Results	70
5.11.6	Conclusion	71
5.12	Summary	74
6	Future Work	76
7	Conclusion	78
	References	81
A	Author’s Publications and Grants	87
A.1	Publications Related to the Thesis	87
A.1.1	Publications in Journals with Impact Factor	87
A.1.2	International Conference Proceedings	87
A.2	Selected Grants Related to the Thesis	88
B	Abbreviations	89
C	Taster User Guide	91

Tables / Figures

4.1. Parameter set summary	22	2.1. Wrong <code>pdListenTimeout</code> value	6
4.2. An example of real parameterization.....	23	2.2. Possible taxonomy of MBT	7
4.3. Wakeup parameters summary	24	2.3. Possible taxonomy of MBT ...	10
4.4. Startup parameters summary	25	2.4. PN example – Button model..	11
4.5. List of measured parameters ..	26	3.1. FlexRay objectives	18
4.6. Test Network Configuration...	32	3.2. Integration testing objectives	19
4.7. Selected experimental results ..	33	4.1. FlexRay comm. overview	21
4.8. <code>pOffsetCorrectionOut</code> measurement (all values in μT) ...	34	4.2. Wakeup pattern in context wakeup window	24
4.9. Overview of measurement duration.....	35	4.3. <code>pdListenTimeout</code> measurement.....	26
5.1. Selection of modeling environment	36	4.4. Cycle length measurement	27
5.2. Supported subset of UP-PAAL modelling language	45	4.5. Cycle difference accumulation.....	28
5.3. Metrics suitable to TA	51	4.6. Offset correction affected by <code>pClusterDriftDamping</code>	28
5.4. Kessy model summary	58	4.7. Offset correction affected by <code>pClusterDriftDamping</code>	29
5.5. Kessy – fault injection results	61	4.8. Offset correction measurement limitation	30
5.6. Kessy performed test runs	62	4.9. Measurement Setup	31
5.7. Trunk ECU Inputs	63	4.10. Structure of FlexRay SoC.....	31
5.8. Trunk ECU Outputs	63	5.1. Inner Light Controller.....	39
5.9. Trunk model summary	69	5.2. Driver model.....	39
5.10. EXAM Test Suite	70	5.3. MBT concept for Integration testing.....	42
5.11. Performed test runs	71	5.4. MBT Workflow in details	43
5.12. Test runs – SUT int. 1	72	5.5. NI HIL platform	44
5.13. Test runs – SUT int. 2	73	5.6. TA coverage enumeration	51
		5.7. Taster architecture	53
		5.8. Taster Viewer	55
		5.9. Syntax error window	55
		5.10. Taster Run Screen	56
		5.11. Trace Viewer	56
		5.12. Start button model.....	58
		5.13. Lock button model	58
		5.14. Key position model.....	59
		5.15. Observer model – Lock System	59
		5.16. Observer model – Ignition System	60
		5.17. Lock able signal	60
		5.18. Unlock able signal	60

5.19.	Start able signal	60
5.20.	A part of SUT implemen- tation model	61
5.21.	Input – SoftTouch button	65
5.22.	Input – Button on inner door’s side	65
5.23.	Input – Button on Key	66
5.24.	Input – Button on Dash	66
5.25.	Input – Close by hand	67
5.26.	Input – Virtual Pedal	67
5.27.	Input – Unlock button	67
5.28.	Environment – A Driver Model	68
5.29.	Trunk Observer – Basic	68
5.30.	Trunk Observer – Full	69
5.31.	Trunk Node Coverage	71
5.32.	Trunk Edge Coverage	72
C.1.	Input Model	93
C.2.	Model Viewer	94
C.3.	Taster Runtime Screen	95
C.4.	Trace Viewer	96



Chapter 1

Introduction

The length of a development cycle of new car models is continually decreasing. Aside of question if it is meaningful or not. Such situation makes a lot of pressure for research and development of appropriate validation, verification and testing methods, which would ensure the quality of final products (i.e. cars). The trend is noticeable for several decades and still it has not reached the peak. Rather, then it accelerates together with the expansion of a number of electronics systems equipped in vehicles [1].

In modern vehicles, multiple communication buses are involved. Most common ones are Controller Area Network (CAN) and Local Interconnect Network (LIN). Due to theirs limitations, newer communication standards find more and more applications and are slowly replacing the old ones. One of those newer communication systems is called FlexRay [2]. The FlexRay communication system is intended for reliable data transfer with speed up to 10 Mb/s. Its intended application is x-by-wire systems (e.g. steer or brake by wire). More common real-world application is drive train and chassis stability control in high-class Audi, BMW, and Mercedes-Benz vehicles. The focus on reliability makes FlexRay much more complicated in comparison to other automotive communication systems. For example, FlexRay configuration set contains tens of parameters in comparison with several for CAN. The difference between intended and actual parameters can significantly affect system reliability. Some of the configuration parameters are easy to measure. For example, incorrect setup of communication cycle length will not allow integration into running communication or, in the case of cold start node, the initialization of communication. A different situation occurs in the case of incorrect values of clock synchronization parameters. Under common conditions, the FlexRay communication can operate without any observable variance.

In connection with the area mentioned above, there is highly actual and exciting area of Model-Based Testing. Term Model-Based Testing (MBT) covers large research area, and its meaning can differ across disciplines. In this thesis, the MBT will be denoted a testing method based on an executable model. Much work was done on this field in past few decades [3]. Despite this, a significant number of challenges await for a solution. MBT technologies are not sufficiently widespread in automotive development. To help to MBT dissemination, it is necessary to continue with inventing new and adapting existing methods in a form meeting specific car industry requirements. Deployment of an MBT oriented processes can solve multiple challenges in automotive electronics testing.

An integral part of the testing of an automotive electronics system is the Integration testing. It is a high-level function oriented examination. The purpose of this testing is to examine if newly developed Electronic Control Units (ECU) can work in a group as a distributed system. Moreover, many functions are not implemented only in single ECU. The Integration testing is essential for a tryout of such features. The MBT approach can address challenges such as the needful amount of human work, increase the test coverage or decrease time to test.

Chapter 2

State of the Art

Passenger car manufacturers play a role of a system integrator today, as a substantial part of vehicle subsystems is supplied by their contractors. This is especially true for vehicle electronics, where particular ECUs are supplied by different manufacturers. Nevertheless, the ECUs have to seamlessly collaborate together. The ECUs collaboration utilizes vehicle communication network technologies like CAN, LIN or FlexRay, thus expected network functionality is vital for reliable and safe vehicle operation.

In the case of FlexRay ECU, communication behavior is affected by tens of parameters that must be set according to the vehicle manufacturer specification to ensure FlexRay cluster robustness and reliability. The vehicle manufacturer may not rely on ECU manufacturer declaration of conformity (in terms of correct parameter values) and has to measure the actual parametrization instead. This approach is common for CAN and LIN networks, where the number of critical parameters is lower, and measurement methods and instruments are widely available. As far as I know, such measurement methods are not available for the FlexRay technology at all.

2.1 FlexRay

Deployment of a communication system starts with a bus (technology) selection. In general, it requires the availability of proper testing and analysis tools and methods. A project continues with fundamental network design such as network topology, communication speed, and selection of basic parameters. Network design results in a complete parameter set (parameterization) for each network controller. The complexity of this parameterization depends on chosen communication technology. In the case of researched FlexRay, it is tens of parameters. This section maps work related to FlexRay configuration parameters evaluation.

Conformance of a FlexRay communication controller – part of System on a Chip with FlexRay protocol specification [2] is ensured by conformance testing specified by [4]. This document provides exact instructions for chip manufacturers how to test their newly implemented FlexRay devices. From the parameterization point of view, it defines couple of tests for each parameter. The objective of the tests is to examine if parameters influence communication controller behavior as it is defined in the standard.

A car manufacturer (system integrator) starts at best with tested FlexRay silicon chips. Verification of everything else is its own business. One of these responsibilities is validation if the parameters specified by a network designer conform with parameters programmed into registers of communication controllers. This task is usually done by some network analyzer connected to a communication bus. For well-adopted communication protocols these devices are commonly available on the market as development support tools.

2.1.1 Of the Shelf Available Solutions

The section provides an overview of analysis tools and devices available on the market and their features. The list is not comprehensive due to the limited information provided on companies websites. The intention of this section is an outline of production solution capabilities.

The first representatives are Digital Oscilloscopes. Producers, such as Agilent Technologies, Tektronix or Teledyne LeCroy offer oscilloscopes with a frame decoding capability. Teledyne LeCroy probably provides the most advanced solution with its product called Trigger, Decode, Measure/Graph and Physical Layer (TDMP), which is capable of measuring some physical layer characteristics as propagation delay or jitter. No one can extract data link layer parameters.

The second group of representatives contains Specialized FlexRay Interfaces. For instance supplier of modular hardware for rapid prototyping, National Instruments offers two types of FlexRay interface card. As the label suggests, NI PXI-8517/2 designed for PCI eXtensions for Instrumentation (PXI) platforms and NI PCI-8517/2 is for conventional PCI. Programming resources are unified in NI-XNET driver. Application development is possible in LabView or ANCI C/C++ environment. The simple FIBEX editor is also included under the name Database Editor. The platform provides a flexible environment for high-level tests implementation. Low-level features necessary for parameter evaluation are not accessible.

From the list of specialized bus tools manufacturers for the automotive Industry, Eberspächer, and Vector were selected. Eberspächer Electronics portfolio contains hardware interfaces e.g. FlexCard USB and FlexCard PMC II. The difference is in a number of physical interfaces and its combination – FlexRay, CAN/CAN FD, Ethernet. There is no difference in analysis capabilities related to this work. Hardware is based on Bosch E-Ray IP core. The analysis is possible by Caromee software. In product manual is stated capability to display of unspecified synchronization information. The benefit of Eberspächer software is multivendor hardware support including Vector or National Instruments.

Vector Informatik GmbH is state of the Art company in automotive communication systems development support tools. Vector offers several FlexRay network interfaces which differ on a number of channels and PC connection type. One of the highest spec options, VN8900 is available in our department, and it can be used for the thesis purposes. Vector interfaces are based on same Bosch E-Ray IP core as Eberspächer products. Available analysis software is CANoe. Vector also provides FIBEX editor with some useful advanced functionality like parameters constraints check. The CANoe can show some parameters such as an actual value of rate and offset correction. Features can be extended by CAPL scripting.

Of the shelf available solutions can be summarized in following way. Large variety of FlexRay development support tools products is available on the market. Many of them support active bus transmission to provide synchronization node ability. Tools are usually connected with complex software environment. Bundled features are frequently extensible by scripting or programming. Nevertheless, capabilities are oriented to the application layer of International Organization for Standardization (ISO) / Open Systems Interconnection (OSI). Configuration parameters evaluation is data link layer problem. It requires ability to influencing of frame transmission e.g. transmit time variation. This kind of features no one of available devices can offer.

2.1.2 Related Work

The growing complexity of automotive embedded systems requires new measurement and validation techniques [5]. The similar research focused on other vehicle distributed systems (mostly the CAN) was conducted in the past, e.g. in [6]. The CAN interface configuration is much simpler than for the FlexRay, since only a few parameters are used, such as time quantum, a length of particular bit segments and synchronization jump width. One parameter, in particular, is critical for the ECU with CAN interface deployment – the sample point position within the bit time. The risk of incorrect setting and the measurement method was described in [6].

Book [7] is an overview of Time-Triggered Communication Systems. It is not only focused on FlexRay, but also other Time-Triggered protocols are discussed with their pros and cons. An example might be Time-Triggered variants of CAN (TTCAN) and Ethernet (TTEthernet). Remarkable is the description of core Time-Triggered Communication principles. In the context of this work clock synchronization study in section [8] can be helpful. Some general verification recommendations are also mentioned in [9].

Authors of [10] have focused on the extraction of FlexRay cluster global parameters. Their primary goal was to develop an FPGA device with the capability of automated parameters identification from ongoing FlexRay communication. The solution is based on passive bus monitoring. A subset of Global parameters marked with *g* prefix in FlexRay standard is investigated. Analysis starts with bitrate detection for proper sampling frequency selection. Consequently, duration of communication segments is extracted. Direct measurements, as well as calculations using protocol constraint, are performed. The work can be viewed as automation of Time Division Multiple Access (TDMA) constraints identification using an oscilloscope. Clock synchronization related parameters are not investigated under this work.

Paper [11] is also focused on global cluster communication parameters. The motivation for configuration parameters evaluation is discussed in details. Also, it is discussed in context of intended application – automotive safety-critical control systems. The parameter set is analyzed and divided into three classes according to its scope. Similar to paper described in the previous paragraph, the approach is based on passive bus observation. Experimental results using five different network setups are shown. Implementation of a health-monitoring node used for measurement is not covered in many details. The impact of bus traffic density to parameter identification is presented. Clock synchronization related parameters are categorized in parameter set analysis, but no experimental results are included.

In summary works [10] and [11] targets to extraction of FlexRay cluster global parameters. They rely on the passive bus communication monitoring approach only. The limitation of this approach is its inability to reveal the values of all parameters, especially the local node-specific parameters that define node-specific behavior within the wakeup, startup and synchronization. These parameters apply to boundary conditions that are not usually reached during normal operation.

Work [12] studies the behaviour of local oscillators used as a clock source in individual FlexRay nodes. FlexRay specification allows usage of the ordinary crystal oscillator with deviation from nominal frequency up to 1500 *ppm*. Authors present the interesting method for local clock frequency measurement. The method works remotely using the connection to a FlexRay bus. In comparison with the above-mentioned works, it

requires active bus transmission to influence clock synchronization algorithm. Despite the paper is not focused directly on configuration parameters measurement, it shows the interesting method for revealing actual value of offset correction in a FlexRay node. Knowledge of actual offset correction is basis for evaluation of *pOffsetCorrectionOut* parameter – offset correction limit.

■ 2.1.3 Risk of Incorrect Parameterization

A significant FlexRay node parameter is *pRateCorrectionOut*, which determines the maximal possible rate correction value the node is allowed to apply. Let's consider the following situation: the desired value of *pRateCorrectionOut* parameter in ECU specification is 601 μT (unit Microtick defined in FlexRay standard and abbreviated μT), which is the maximal value for communication speed 10 Mb/s and communication cycle length 5 ms. This value allows the communication controller to correct maximal permitted oscillator deviation, which is defined by standard [2] as 1500 ppm of oscillator's nominal frequency.

Let us assume that in the supplier delivered ECU the configured *pRateCorrectionOut* value is not 601, but only 300 μT instead. Such a violation of node parameter specification would not influence the ECU's behaviour under most conditions, unless the local oscillator's frequency reaches a deviation higher than 750 ppm from the nominal value (more precisely from the cluster average value). At this moment the ECU ends communication and goes to the halt state, while it would continue operating with correct parametrization. Considering a crystal oscillator natural behaviour [13], this probably happens after a long time (several years) due to crystal aging in combination with high or low temperatures [14]. To detect this kind of specification breach special testing methods are needed, allowing the car manufacturer to ask the ECU supplier to fix the parametrization during the pre-production phase.

A wrong value of *pdListenTimeout* is another example of a manufacturer specification breach. The *pdListenTimeout* value specifies the time spent in a coldstart listen phase. According to specification [2] the *pdListenTimeout* value has to be calculated according equation (2.1):

$$pdListenTimeout[\mu T] = 2 * (pMicroPerCycle[\mu T] + pRateCorrectionOut[\mu T]) \quad (2.1)$$

Depending on the communication cycle schedule, lower as well as higher parameter values can cause problems with FlexRay cluster startup (e.g. leading coldstart node change or startup phase extension). Three possible scenarios are shown in Figure 2.1. All cases consider two coldstart nodes in a communication network.

The first case labelled I. shows a situation when the gap between Collision Avoidance Symbol (CAS) and the startup frame in a static slot (actual *pdListenTimeout* of node 2) is shorter than the required *pdListenTimeout*. In this case the startup behaviour of the cluster is not seriously affected. The second scenario shows the change of leading coldstart node. The *pdListenTimeout* timer of node 2 expires before node 1 sends its first startup frame (assigned key slot is later in static segment of communication cycle) and cluster startup is thus delayed. Case III. means that *pdListenTimeout* parameters in both nodes are too small; the nodes are alternating in coldstart leading and the

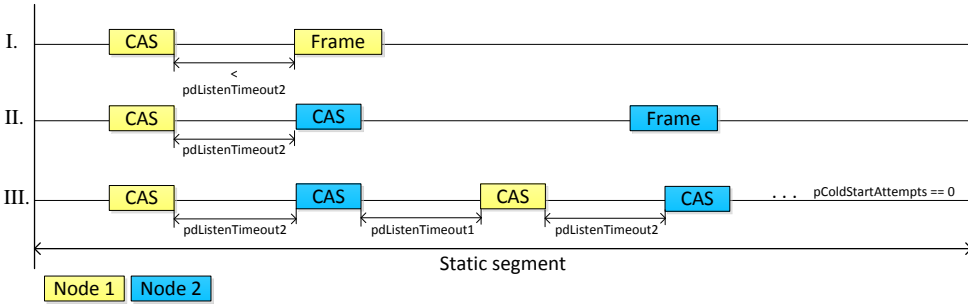


Figure 2.1. Effects of wrong `pdListenTimeout` value

startup delay is significant. To prevent all these problems, an evaluation of the actual value of the `pdListenTimeout` parameter is necessary.

The examples presented above show that the problems originating from incorrect ECU parametrization may occur later (caused by components ageing), under specific operating conditions (e.g. extreme temperature), under the spare part change or under the simultaneous influence of aforementioned effects. This kind of problem is hard to detect because of its sporadic nature. Evaluation of configuration parameters can dramatically increase confidence in the fault-free operation of FlexRay networks during the car life cycle.

2.2 Integration Testing and MBT

Integration testing is widely used term covering various methods and processes in testing of software or electronic systems. In general, Integration testing denotes a phase after some parts (modules, components, code fragments, etc.) are joined. Further in this work by name Integration testing is denoted Automotive Integration testing as described below. Integration testing phase is placed close to the end of a new car electronic system development. An example of a development process with corresponding testing methods is depicted in the Figure 2.2. Application area of this work is marked by a yellow background. By nature it is a high level, function oriented testing of almost finished car electronics system or part of that system (a network of ECUs). The objective of the testing is to examine whether the whole system is able to operate in synergy and functions distributed over multiple ECUs work as expected. Automotive Integration testing is being performed on Hardware-in-the-Loop basis on a testbed. An industrial experience says that implementation of test sequences is usually done by test engineers. In general, this area has great potential for research and real word application of MBT solutions.

Test management, development, and execution are usually performed by some software tool. For the purpose of this work EXAM is considered. It is complex testing tool co-developed and used by Volkswagen AG. Graphical user interface provides the environment for individual test cases development. A test case is implemented as sequences or activity diagrams. These test cases are organized in test suites and test campaigns. The test suite is executed by test runner in a sequence of test cases. Finally test results are stored in the database for various assessments.

Current test suites are mostly implemented using typical driver use cases. Time to test of complex systems is limited and thus number of uncovered situations in test suites

is significant. Considered systems have multiple inputs and outputs and particular function behavior is typically constrained by other distributed system parts. I.e. behavior of car inner light depends on the state of doors locking, ignition state, switch position, etc. Frequently basic and low-end functionality test cases are preferred in test suites from high-end variants.

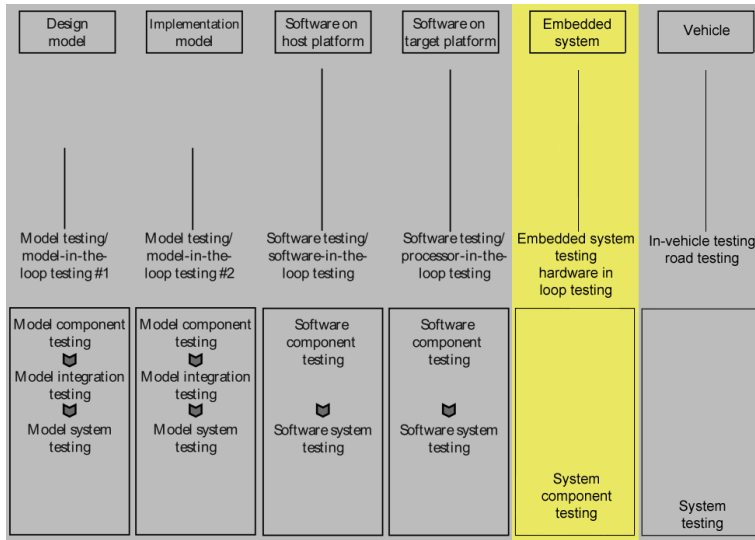


Figure 2.2. An automotive distributed system development process [15]

2.2.1 Weaknesses of Traditional Approach

Manual design and implementation of test cases is very laborious and demanding task. The quality of this approach strongly depends on the quality of test engineers work. Like every human activity, it is prone to errors. Good test rule policy and its enforcement can be beneficial. Weaknesses of manual test case design and implementation can be viewed in a similar way as problems in ordinary coding. Produced code can be great, but there is no guarantee. There are numerous coding as well as software projects management recommendation which can help to ensure the desired level of quality. However, the result is in hands of coding/testing team.

Specifically, in intended Integration testing, supplementary cons are identifiable. One of them is fixation to the behavior of an average driver. It means that test cases mostly acts like the average driver and some less common, but still, valid scenarios remain uncovered. Another con example – a Modern car is offered in various trim levels. Some optional extras can be selected across defined trim levels. It creates a significant number of combinations. The manual combination of test cases covering particular features is practically impossible. Last but not least during a vehicle life cycle, electronics system is often changed. Every change requires an update to a test suite to fit exact system specification. Maintenance of test suites is also very demanding and error prone task.

2.2.2 Terminology

Terminology in fields related to this thesis – Integration testing (defined in 2.2) and Model-Based testing might be confusing. Differences in the meaning of individual

terms are caused by various aspects. One of these aspects is liveness of the scientific areas. Especially the domain of MBT is only a few tens of years old, and research is still active. Another reason is in different conventions in target application area. An example is testing in software development and testing in the automotive industry. To make the work clearly understandable, used terminology is briefly described.

Model-based testing (MBT) covers very extensive research area. The reason of area extensiveness is that in general, it is possible to classify as Model-Based every test method, which using some model (including non-executable or mental model). For the purpose of this work, a testing method is denoted as Model-Based if the method directly uses some executable model to performing a testing.

System under Test (SUT) is a system which is the object of a Testing. In this thesis primarily a car electronics system or its part.

Testing is an interaction with some system. The purpose of the testing activity is to achieve some level of confidence that behavior of the system under test meets the defined objectives. A reasonable definition of test objective is essential for successful test implementation. Demonstration of general test objective is conformance testing. The goal is to find out if the behavior of the system under test is in compliance with system specification.

Hardware-in-the-Loop (HIL) is the testing method with roots in aircraft industry suitable for testing complex electronic control systems like an electronic control unit or whole electronics system. The object of testing is an embedded system, but sensors and actuators are simulated or stimulated by testbed facilities. SUT works identically as in its final deployment, but interaction with the environment is controlled by some test equipment.

Reactive system is a system which reacts to inputs from environment by producing some outputs. Reactive system are usually non-terminating. Behavior of this kind of systems can be non-deterministic and it is usually based on internal system state. For this reason, testing of reactive systems is more challenging in comparison to transformative systems. Transformative systems deterministically produce output after input and some portion of time.

Real-Time Systems is system with responds to an event within defined time window.

Test Objective or test purpose is definition what a test case should test. Test objective can be very different from a formal property, e.g. system does not contain deadlock, to some performance or durability criterion of an SUT.

Test Case is sequence of steps to examine some SUT property or properties. It can be viewed as implementation of one or more test objectives.

Test Suite is a collection of test cases.

Test Campaign denotes higher level entity containing an arbitrary number of test suites. It is used by some test tools to improve test management options.

Offline Testing means the test case or whole test suite is generated before it is executed. Test generation and test execution are strictly separated. The process is similar to manual test case development when tester designs and implements test case. Afterward, test case is executed on a target platform.

Online Testing In opposite to offline testing the border between test generation and test execution is not sharp. Online testing tool interacts with SUT in command and

response way. This approach allows reacting to SUT non-deterministic behavior. The technique is especially suitable for slowly interacting system, e.g. car and driver.

■ 2.2.3 Taxonomy

World of MBT can be categorized in different ways. A possible taxonomy suitable to thesis goals is depicted in Figure 2.3. Firstly it divides MBT into four classes which correspond with consequent steps during testing. Each class is subdivided into multiple categories and options. Titles used for options designate specific methods and technologies. Only this last column refers to specific technology. Classes and categories have a general meaning.

Class models are distinguished on the basis of model purpose and its relation to SUT. Model directly related to the SUT is considered as System Model. An abstract model for deriving test cases only is classified as Test Model. As usual the most common in practice is a combined approach.

With some model, three consequent steps (classes) can be performed. Performing testing in MBT way does not necessarily contain all depicted phases. Test generation class covers test selection (objective). The second part of the class is technology capable to achieve desired test objective. Moreover, final step is rank test generation by a format of obtained results.

Third class categorizes MBT according to execution options. It differentiates what is tested – Model, Software, Hardware or Processor. The ordinary testing process can start with testing of SUT model and continues until it is possible to test final hardware with software. Systems related to this thesis are tested after hardware and software are available. No model or processor in the loop simulation is performed.

Test evaluation refers different approaches to the test result processing. In the simplest way it can be done manually – only test input is generated. In presented case, a model providing test oracle capability is supposed.

■ 2.2.4 Embedded System Modeling

There are numerous options for modeling of automotive embedded systems. The researched system class has two specific features. It is Real-Time and Reactive behavior. Selected model language should be able to capture these attributes. These requirements limit the number of applicable formalisms, but the list of possible languages still contains at least hundreds of options. E.g. paper [17] present a survey on Timed Automaton (TA), and it contains almost eighty variants. The problem is not lack of suitable formalism, rather the opposite – too many options. Enumeration can begin with process algebras like Communicating Sequential Processes (CSP). As the name suggest, the systems are represented by process and interaction between them. Properties can be specified by some modal temporal logic. A known example is Linear Temporal Logic (LTL). The problem of this theoretical description possibilities is that they are not well accepted by practitioners. Graphical representation based on an Finite-state machine (FSM) is much better adopted to a wider audience. Some of them significant in the field of interest are outlined in rest of the section.

Petri net

The first example of an embedded system representation is Petri net. The net consists of places, arcs, and transitions. Arcs can be coupled with weights. There is a huge number of Petri net variants. One of the Petri net definition given by [18], is:

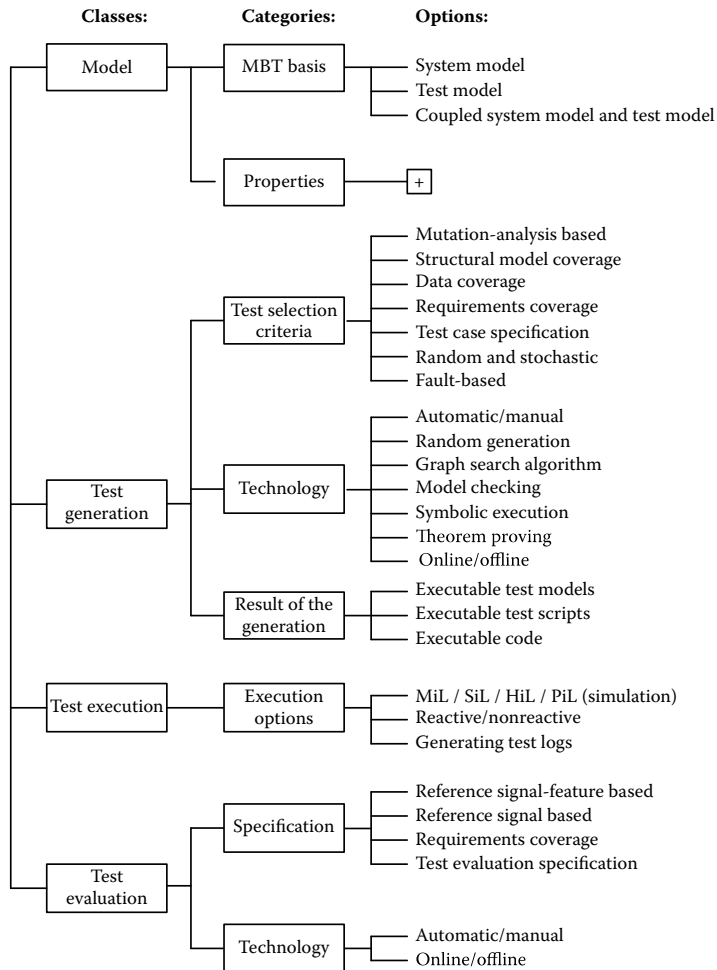


Figure 2.3. Overview of the taxonomy for Model-Based Testing[16]

- Petri net is quintuple (S, T, F, M_0, W) where
- S is a finite set of places.
- T is a finite set of transitions.
- F is a finite set of arcs $F \subseteq (S \times T) \cup (T \times S)$.
- $M_0 : S \rightarrow \mathbb{N}$ is initial marking, where a place $s \in S$ contains $n \in \mathbb{N}$ dots.
- $W : F \rightarrow \mathbb{N}^+$ is set of arc weights, where for every $f \in F$ weight is $n \in \mathbb{N}^+$.
Weight denotes how many dots are consumed by a transition, or it is a count of dots produced in output places.

Petri net according to the definition can also be viewed as weighted bipartite graph. Execution is nondeterministic and depends on marking. Firstly it is necessary to fulfill a minimal number of dots(tokens) on some transition. The enabled transition can be consequently fired. According to weights, tokens are consumed by a transition, and an

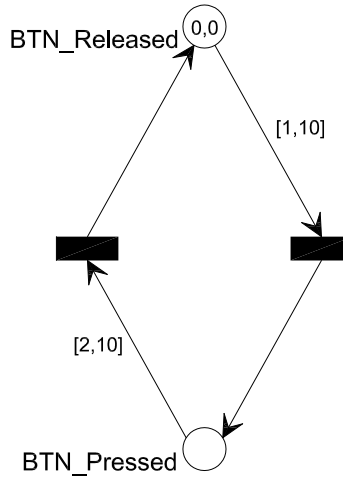


Figure 2.4. PN example – Button model

arbitrary number of tokens is inserted into output place. A timed-arc Petri net button model is depicted in Figure 2.4.

Timed Automata

Theory of timed automata was originally published by Alur and Dill [19]. Actually the term timed automata usually means slightly modified version called Timed Safety Automata [20] which uses local invariant conditions to ensure automaton progress. Formal definition of a single timed automaton, given by [21], is:

- A timed automaton A is a tuple $A = (N, l_0, E, I)$ where
- N is a finite set of locations (or nodes),
- $l_0 \in N$ is an initial location,
- $E \in N \times \mathcal{B}(\mathcal{C}) \times \Sigma \times 2^{\mathcal{C}} \times N$ is the set of edges and
- $I : N \rightarrow \mathcal{B}(\mathcal{C})$ assigns invariants to locations
- We shall write $l \xrightarrow{g,a,r} l'$ when $\langle l, g, a, r, l' \rangle \in E$

Local invariant is constraint in form $x < n, x \leq n$ where n is natural number.

Informally, the timed automaton is an oriented graph which contains states and transitions. One of the states is called initial. Transitions are labeled by guards which enable relevant transition. One of the key concepts of timed automata theory is parallel composition of individual automata to the network of timed automata. The theoretical background for this operation provides parallel composition operator know from CSP. Timed Safely Automata are implemented e.g. in UPPAAL [22] tool. Comparison of TA and Petri net (PN) is provided by [23].

UML

In comparison to formalisms presented above, Unified Modeling Language (UML) is a bit more complex. TA and PN are compact modeling systems with strong mathematical background. UML was born for practice. UML application is not limited to model a

system behavior, but it is capable of capturing most of the aspects of a software system including business processes. MBT tools usually implement a subset of UML. I.e. system structure is expressed by UML block diagram, and system behavior is modeled by FSM. With the presumption of usage of an FSM variant for behavior modeling, it is UML as MBT basis very similar to TA.

■ 2.2.5 Test Selection Criteria

An integral part of a test generation are test selection criteria. It is specification what the generated cases should or should not contain. Definition of reasonable test criteria is work for a test engineer. It is similar to manual test case design, just in the more general way. Unfortunately it is not possible to select single best test selection criteria for any SUT. Based on the work [24] test selection criteria suitable for transition based models is overviewed. Also, test generation can be directly influenced by form of an environment model.

Structural Model Coverage Criteria

They use model structure to expresses test objectives. In case of FSM based models criteria are in form which nodes and edges should be incorporated in test case. Obviously, full node or full edge coverage are very commons criteria. Another criterion is a coverage of all automaton cycles. Instead of full/all entities a certain level of coverage can be chosen. E.g. test case has to cover at least 70% of edges. As not so straightforward options the isomorphism-checking methods [25] can be mentioned. Similarly to code testing, Modified condition/decision coverage (MC/DC) can be applied to an FSM.

Data Coverage Criteria

Industrial strength formalism works with variables as well as complex data structures. It makes case structural coverage criteria hardly applicable. Due to state space magnitude, it is necessary to pick significant variable values. Data coverage criteria are based on some partition of value space into classes or intervals of boundary values. As a representative Classification Tree Method [26] is listed.

Fault-based Criteria

Evaluation of test cases or entire suites bases on effectiveness to reveal pre-defined bugs in SUT. Instead of model oriented metrics – e.g. node coverage is Fault-based criteria expressed as a count of revealed injected bugs to an SUT. A well-known member of this group is Mutation Testing [27]. The key idea is to test a test suite by small variation (a mutant) of SUT implementation. A mutation example is replacing operator such as $<$ to $<=$. Same tests are executed over original and mutated SUT. The significant disadvantage of Mutation Testing is a required access to an SUT code, which is not possible in many cases.

■ 2.2.6 Test Generation

This is a first phase of the testing process automation. After the model development and specification of test selection criteria is time to release the power of MBT. With all necessary information in machine-readable form, rest of testing process can be completely automated. Suitable algorithms run over the model and generate test cases which fulfill given criteria.

Graph search techniques

Taking into account graph nature of many modeling formalisms, application of graph search algorithms act naturally. Graph traversing is possible in various ways. Some suitable for MBT are for instance random search, Breadth-first search (BFS), and Depth-first search (DFS) respectively its modified versions according to the used modeling formalism. Model semantics bring some constraint into the exploration. For example, it is not possible to pick any edge if some enabling condition (guard) is not fulfilled. Optimized test generation using shortest path algorithms is also widely used.

Symbolic execution

Instead of specific (single) variable values, a model is executed with a valuation of variables by a set of values (intervals). Alternatively, another analogy with fixed-step discrete simulation, symbolic execution is like a discrete simulation for multiple steps at one time. Model is executed using constraints instead actual values. The result of execution are symbolic test cases. One symbolic test case can represent a group of real test cases. Symbolic as well as real test cases have to fulfill test selection criteria. Instantiation of symbolic to real test cases is done by a sampling of given intervals. The examples of this approach from TA world are Region and Zone graphs [17][20].

Model Checking

MC or property checking is decision making if a system meets given property (deadlock-free, all states are reachable). As a side effect, model checkers can work as test generators. In this case, the test selection criterion is expressed as reachability property. The idea of usage of a model checker to produce test cases for a real system seems very logical as there are a lot of high-level Model Checker (MC) tools and methods [28]. Also, a synergy of model formal analysis with testing of its implementation can be beneficial [29]. Deployment of UPPAAL MC engine for test case generation was objective of the dissertation theses [30][31].

2.2.7 Existing Tools

TRON

Uppaal TRON (UPPAAL for Testing Realtime systems ONline) is MBT tool developed as a part of Marius Mikucionis Ph.D. project [31]. The tool is available from webpage [32]. TRON is intended to online conformance testing of system specified by timed automata network. The modeling language is the same for both SUT and environment models. Models are explored using UPPAAL model checking engine. Inputs for testing are chosen randomly. Test objective is specified by environment model – fully permissive model performs full conformance test. For real SUT connection multiple examples how to implement test adapter are provided. Presentation [33] provides further information.

CoVer

Uppaal CoVer is a command line extension [34] for offline test generation based on UPPAAL model checking engine. Test suite generation is controllable by Observer and Property files. Syntax is described in Backus-Naur Form. Format of abstract test cases can be modified by Extensible Markup Language (XML) config file. Model coverage criteria are specified by observer automata [35]. Tool generates abstract test suite which has to be converted to an executable form. Detailed CoVer description can be found in Anders Hessel Ph.D. thesis [30].

Yggdrasil

Simple offline test-case generation tool integrated in UPPAAL since version 4.1.19. Yggdrasil derives traces with edge coverage criteria. It is necessary to fulfill some conditions (deterministic, deadlock free models ...), otherwise tests might not be generated. To make generated traces executable it is possible to add a test code into the model. Test code can be added to nodes and edges of the model.

Features:

- Offline test generation
- Depth or Breadth search order selection
- Load a trace into UPPAAL simulator
- Export of outputs to a file

■ 2.2.8 Hardware-in-the-Loop

Testing can be performed from a very beginning stage of the development as it is shown in Figure 2.2. Whichever is available it can be distinguished as Model-in-the-loop, Software-in-the-loop, Processor-in-the-loop and finally Hardware-in-the-loop. Integration testing works with pre-production versions of an electronic car system. The intelligent part of this system is usually implemented as electronic units called an ECU. ECU is Microcontroller Unit (MCU) controlled module with various input and output types. Individual ECUs are interconnected by a communication link.

The main purpose of Hardware-in-the-loop (HIL) facilities is to handle SUT inputs and monitor its outputs. It creates the interface between real hardware and software testing tools. The method serves inputs to an SUT and observes its reaction by observing outputs. Identically as testing of a computer program, but Inputs/Outputs (I/O) are physical. In-the-loop postfix indicates cyclic nature of the process. Input values generation, as well as the evaluation between right and wrong outputs, is the task of test generator.

Working in desired time window is required since investigated systems have Real-time properties. It means that a new input has to be served within specified time and similarly output is valid only within a given time interval. Handling time constraints is also dedicated to an HIL platform. Simulation of complex subsystems such as sensors, actuators, and communication interfaces is frequently necessary. To conclude the HIL method allows performing MBT with real hardware.

HIL test execution has its specifics [36]. The mapping between abstract test cases to the real ones have to be solved. For instance in a step of an abstract test case is input value specified by equivalence class. However, in execution time is necessary to choose the certain value. In a simplified example, it can be done by selecting a random value from given interval. Instantiation of test sequences is usually done by a software module called test adapter. The extensiveness of the adapter depends on input and output formats compatibility.

■ 2.2.9 Related Work

In previous text, an introduction to the large world of MBT was outlined. This section analyses few papers, which are the most relevant for intended work. The referred works present results of an application of different MBT approaches. Target SUT class are embedded Real-Time systems. The thesis is focused on the same system class.

Paper [29] proposes a scenario of usage UPPAAL tools in the Automotive domain. An object of interest is a car turn indicator system. This system is modeled as TA network, and consequent formal analysis and testing are driven by the model. Formal properties can be checked by symbolic MC or Statistical Model Checker (SMC). Both are incorporated in UPPAAL tool. SMC offers speed in exchange of confidence level. The formal analysis is used to check model (system requirements) in the early design phase and some safety standards, e.g. ISO 26262 requires usage of formal techniques. The same model can be later used for test generation by Yggdrasil for implemented SUT. Yggdrasil is offline symbolic test case generator included in UPPAAL. In conclusion, it is stated that TA modeling is acceptable for the industry but still requiring the reasonable amount of training.

Work [37] is focused to timed test traces generation from TA model. The novelty of presented approach is in the employment of meta-heuristic algorithm for producing of traces. The method was evaluated on an Anti-Slip Regulation (ASR) / Anti-lock Braking System (ABS) model. The system was designed in SystemC and TA was generated from it. Resulting automaton is quite large to analyze by a model checking engine. It contains 261 nodes and 229 transitions. UPPAAL MC can't compute nodes reachability with 4 GB of RAM. Presented test generation is done by evolution algorithm. Proposed algorithm try to achieve the best possible transition coverage. Result for the ABS example was 95.45 % coverage in comparison with 45.45 % for MC and randomized approach.

Comparison of various MBT solutions is a difficult task. A freely available benchmark model is offered by [38]. The model captures a complete turn indicator system on observation level. SUT outputs are monitored in detail including Pulse-width modulation (PWM) duty cycle. The benefit is that the modeled SUT is the real-world system as it is used by Daimler. The only difference is pure UML2 format instead modified UML used by Daimler for some HIL connection reasons. A reference test generation solution is based on the calculation of symbolic test cases represented as logic formulas. Criteria for benchmarking are clearly described. Unfortunately, to harmonize result of two MBT tools to comparable form can require inconsiderable work.

2.3 Summary

Practice in the automotive industry was outlined. The current situation requires a wide range of testing and validation techniques. The first area waiting for new testing methods is parameterization of FlexRay Communication Controller (CC). Key principles of FlexRay communication system were presented. Features and types of available solutions were overviewed. Available tools can be divided into two group. The first group is smart oscilloscopes with FlexRay bus decoding function. The second group uses programable network interfaces. Tools from the second group are more suitable for automotive electronics development but still are not disposable for low-level FlexRay testing. In the corresponding related work, the published results are summarized. Results are mostly based on passive bus observation. Also, parameters are estimated for the whole cluster and not precisely measured for a single node. Furthermore, the motivation for the evaluation configuration parameters is stated. Despite the long list of market solutions as well as huge amount of publications, it can be concluded that measurement methods for a single FlexRay node parameters evaluation are unavailable.

In the second part of this chapter, the scenario with carmaker and supplier was described on a higher level. The Integration testing is introduced as a particular development phase. In comparison with FlexRay parameter evaluation, the bus communication is tested indirectly at the system function level. The ECUs are connected by their communication interfaces. The communication is usually monitored, but its malfunction is backward identified only if some function fails.

A common Integration testing procedure with manually developed test cases was sketched. Weaknesses of this approach bringing motivation for this work were discussed. A modern MBT approach was summarized in the rest of the chapter. As the MBT terminology is not exactly uniform, used notion was defined. Hereafter whole testing process was discussed according to MBT approach, starting from modeling of the intended class of automotive electronics systems and continuing with the test selection specification. At this point, human (i.e. test engineer) work ends and the rest can be fully automatized. The test generation phase prepares test cases and whole test suites or campaigns. Finally, the test execution on an HIL platform was discussed. Arising thesis objectives are specified in the consecutive section.

Chapter 3

Thesis Objectives

In section 2.3, two prospective research areas are identified. Evaluation and measurement of the FlexRay configuration parameters is the first of them. The second part of the objectives targets to the area of high-level function oriented Integration testing. Respectively, the goal is an improvement of the Integration testing using MBT principles. According to the testing practice, both fields belong to different part of the testing process. Measurement of a bus configuration parameters is classified as lower level testing while high-level testing of the nearly finished distributed system as an Integration testing case. For this reason, the thesis objectives, as well as entire thesis, is divided into two parts.

3.1 FlexRay Objectives

The aim of this part of the dissertation thesis is to fill the gap in the area of available FlexRay communication system measurement and validation techniques. State of the art analysis in section 2.1 shows the lack of evaluation techniques focused on a single FlexRay node parameters. Paper [11] supports the conviction that the approach based on the application of active stimuli is needed. This work is thus an expected extension of communication parameter extraction methods based on passive bus observation. Existing work is mostly focused on an analysis of whole FlexRay cluster parameters. Marginalization evaluation of individual nodes might cause unsafe behavior described e.g. in 2.1.3. The goal of this work is expressed in Figure 3.1.

The set of the FlexRay communication protocol standards describes Physical [39] and Data Link [2] layers according to ISO/OSI model. Configuration parameters mentioned in this work influence Data Link Layer behavior. A network designer creates a configuration based on some requirements. A configuration tool can be deployed in the design process. The set of configuration parameters is used for configuration of individual FlexRay controllers deployed in ECUs. This work is not intended for verification of proper network design, but it is intended to validate if parameters values specified by car manufacturer conform with actual values programmed into a communication controller. The validation is based on the active bus communication, no access to ECU firmware is required.

Firstly, the parameterization (set of tens of parameters) of a FlexRay controller will be analyzed with the focus on potential risks in case of the wrong setup. It is presumable, that not all of them are critical, due to the robust protocol design. Based on this analysis, measurement methods will be designed and developed. Finally, these methods will be implemented and evaluated on a real FlexRay network.

Highlights of particular objectives:

- Analysis of the configuration parameters set.
- Evaluation of single FlexRay node parameters – design of measurement algorithms.
- Implementation and evaluation of the measurement methods.
- Characterization of measurement methods accuracy and time requirements.

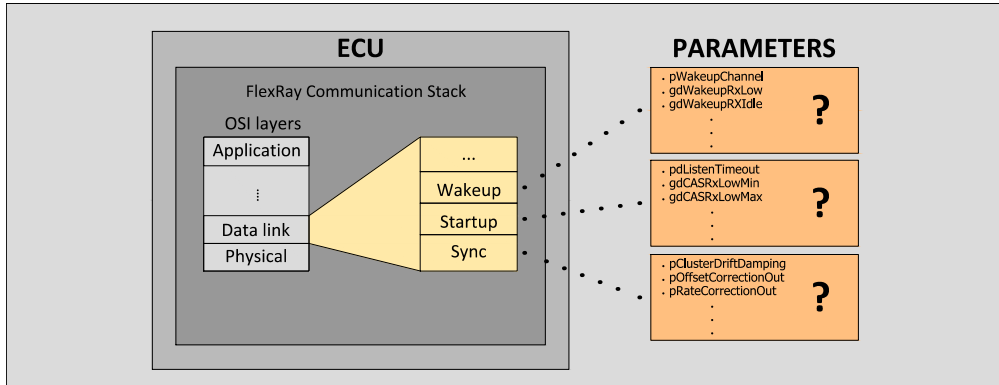


Figure 3.1. FlexRay objectives

3.2 Integration Testing Objectives

The starting point for this part of the thesis is test suite design and implementation by test engineers without automation as sketched in section 2.2. This area has excellent research and development potential. In general, the idea is to improve the Integration testing using the MBT techniques. Slightly wider beginning formed specific particular objectives. The work goes from the selection of applicable theory, through its implementation to results evaluation by a case study. Effort transfer from test cases coding to model development is expressed by Figure 3.2.

First of all, it is necessary to choose suitable modeling formalism for intended system class. Fundamental options with required attributes were also overviewed in 2.2. Selected model type proceeds to the suitable MBT concept proposal. Based on the proposed concept, the relevant theory is studied in detail. The MBT solution implementation is described including target HIL platform. The concept is proven on a case study.

Highlights of particular objectives:

- Selection of suitable modeling formalism.
- Overview of relevant theory.
- Proposal of suitable MBT concept.
- Development of HIL test place.
- MBT solution implementation.
- Knowledge-based test generation.
- Evaluation by a case study.

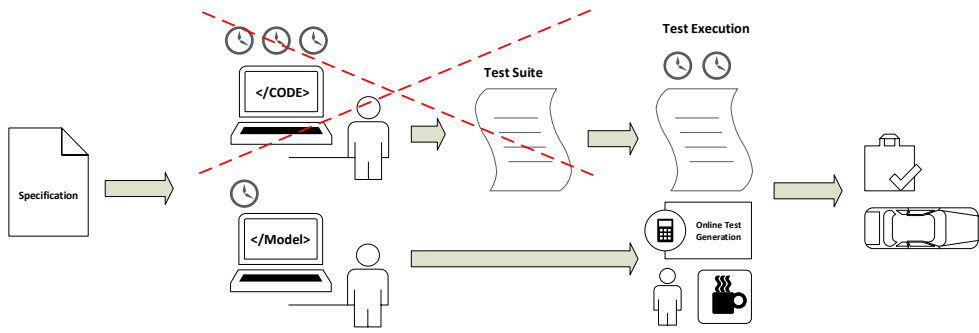


Figure 3.2. Integration testing objectives

Chapter 4

FlexRay

The FlexRay communication stack within an ECU complies with the OSI model. The Data-link layer entities and protocols are implemented by the FlexRay communication controller – the chip provided by the semiconductor manufacturer. However, the Data-link layer behavior is also significantly influenced by parametrization of its protocols and entities. Values of particular parameters are set by the ECU firmware, and thus it is the responsibility of the ECU manufacturer to set the required parameter values. The ECU is a black (or at best a gray) box from the car manufacturer point of view. The manufacturer knows what parameter values should be set (they have specified them), but they have no access to the ECU firmware to verify them. Therefore, the car manufacturer needs measurement methods and instruments suitable for measuring values of these parameter without having access to internal ECU data.

4.1 Overview of FlexRay Communication System

Description of FlexRay Communication System can be found in [40] or directly in protocol specification [2], which is the primary information source. The following paragraphs are focused on relevant principles only. Briefly, FlexRay is high speed (up to 10 Mbit/s) communication technology primarily intended for automotive applications. Both time-triggered and event-triggered communication is possible and redundant physical layer topology can be used for increased reliability.

The wakeup procedure serves to drive the network from low power mode (e.g. after a car is unlocked) [41] to normal operation mode. The procedure starts with ECUs in either standby or sleep mode, where only bus drivers are powered. The process is controlled at application layer by the ECU firmware. An ECU initiating the cluster wakeup starts by sending a wakeup pattern to the bus on one of the redundant channels. Another ECU, which recognizes the wakeup pattern, wakes up and continues the process by sending the wakeup pattern to the second FlexRay channel (if it is present). After the wakeup is finished and all the ECUs are woken up, they continue with a startup phase. Below, the methods for evaluation of proper wakeup pattern transmission and recognition are provided.

The communication startup phase is intended to initialize the communication cycle and clock synchronization. The start of the FlexRay network depends on the network type. FlexRay specification distinguishes three network types: TT-L, TT-D and TT-E, according to the clock synchronization principle. In a TT-L network the nodes are synchronized using a single clock master. A TT-D cluster uses a distributed clock synchronization mechanism, which will be discussed in next paragraph. Finally, a TT-E type uses an external time gateway. This work is focused on TT-D FlexRay network type, which is the most widespread. For the startup process, the network is

divided into coldstart and noncoldstart nodes (ECUs). Only the coldstart ones are active during the communication startup. The first coldstart node, which does not detect bus communication, becomes a leading coldstart node. This node immediately sends a CAS and then starts sending startup frames (normal frames with a startup flag set) according to the TDMA communication schedule. The following coldstart node initializes its clock synchronization and joins communication in communication cycle No. 4. Finally, the noncoldstart nodes join the communication. Below, the methods for evaluation of proper ECU startup parameter values are provided.

FlexRay normal operation phase is based on the modified TDMA communication cycle structure, as depicted in Figure 4.1. The individual ECU clocks are synchronized using the distributed clock synchronization algorithm. ECUs labelled as synchronization nodes are sending frames with the sync flag set within the static segment slots. Other (non-synchronization) ECUs only correct their clock according to the synchronization nodes. All nodes measure an arrival time of synchronization frames locally. Based on the deviations in arrival time, the clock corrections are calculated using the Fault-tolerant midpoint (FTM) algorithm [42] and applied in particular nodes.

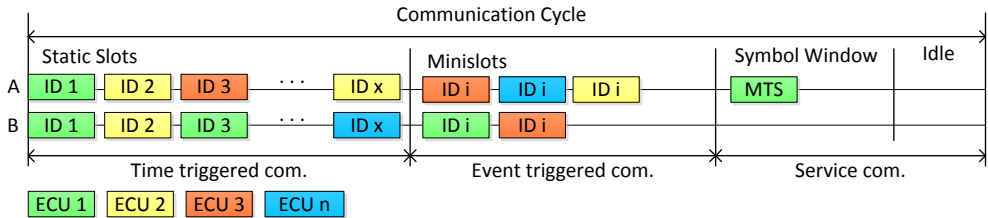


Figure 4.1. FlexRay communication overview.

The latest version of the specification is [2]. It can be viewed as the final version. FlexRay consortium submitted them to ISO to become technical normative. Documents are not longer maintained by FlexRay consortium. Probably only significant problems will cause publication of revised documents in the future. From the application point of view version [43] is important. The reason is that the most of the today's application is based on 2.1 version. Moreover for the purpose of this work a repetitive survey of available FlexRay controllers integrated into an MCU was done (last in November 2016). All founded products have CC v 2.1. None of the known silicon manufacturers had CC supports 3.0.1 in the portfolio.

Version 3.0.1 in comparison to 2.1 brings reasonable number of improvements. Lot of them are related to text of the specification only. A variety of applications is extended by the introduction of 2.5 and 5 Mbit/s communication speed. Basic TT-D clock synchronization architectures were extended by TT-L and TT-E options. Despite the changes, presented work is valid for both mentioned variants. Just parameter *pdMaxDrift* was replaced by *pRateCorrectionOut*.

4.2 Parameter Set Analysis

One of the first goals of the thesis was to analyze parameter space of FlexRay protocol. In Table 4.1 are parameters divided into groups according to specification. First are protocol constants – definition of fundamentals such as highest static slot ID, the length

of mandatory communication segments or maximal payload length. Two parameters in second section performance constants defining how long time synchronization parameters calculation have to take.

Major part – configurable parameters are divided into Global cluster and Node parameters. They are easily distinguished by g or p prefix. Among all categories subsidiary prefix d is used to designate time duration. This convention is followed by silicon chip manufactures in datasheets. Global parameters are the same for the whole cluster, and node local ones can differ between individual nodes. Division to subgroups of protocol relevant and protocol related means only that related parameters are used in Specification and Description Language (SDL) diagrams.

The physical layer group contains values related to a bus driver. In the context of this work are similar to protocol constants as they are not configurable. Last part is auxiliary parameters which are used in configuration constraints definition. It is set of equations which verify formal correctness of a FlexRay parameterization. Total parameter count according to the latest version 3.0.1 is one hundred and eighty-one.

Group	Count	Prefix	Section
Protocol constants	40	c	A.1
Performance constants	2	cd	A.2
Global cluster			
Protocol relevant	25	g	B.3.1.1
Protocol related	11	g	B.3.1.2
Node parameters			
Protocol relevant	29	p	B.3.2.1
Protocol related	4	p	B.3.2.2
Physical layer	34		B.3.3
Auxiliary parameters	36	a	B.3.4
Total	181		

Table 4.1. Parameter set summary

In the case of automotive application domain is important sixty-nine cluster global and node local parameters. This set is programmed into each CC connected to the vehicle network. The correctness of chosen values, as well as conformance of desired and actual values, should be verified.

Parameterization of available CCs is not an exact copy of parameterization mentioned in the specification. In this work two MCUs with integrated FlexRay CC were analyzed. The first representative is 16-bit MCU MC9S12XF manufactured by Freescale Semiconductor. Configuration of this device contains fifty-four parameters according to datasheet[44]. The second device was Texas Instruments TMS570LS31. Beside firstly mentioned MCU it is a generation newer more powerful with 32-bit architecture. CC registers are described in [45] and it is possible to configure fifty-one values. Detailed structure of parameterization is shown in Table 4.2.

The difference between some parameters in specification and number of parameters in registers of real CCs is not caused by the limited implementation of these specific controllers. Most of missing parameters are from both protocol related groups. Cluster global related contain for example $gdBit$ which defines bit time. Bit time seems like

Group	Freescale MC9S12XF	Texas Instruments TMS570LS31
Global cluster		
Protocol relevant	22	21
Protocol related	1	3
Node parameters		
Protocol relevant	29	26
Protocol related	2	1
Total	54	51

Table 4.2. An example of real parameterization

required parameter, but practically it is defined as multiplication of constant $cSamplesPerBit$ and $gdSampleClockPeriod$. Another example is $gdMaxInitializationError$. It specifies maximal timing error for a node following integration. This kind of values can be viewed as configuration constraint, as it is not possible to assure this timing deadline by a configuration of CC. Some parameters are derivable from other ones – the specification is redundant in some parts for the better readability.

The set of approximately fifty important parameters is very heterogeneous. It varies from transmission channel enable by $pChannels$ to limit of the rate correction $pRateCorrectionOut$. Although $pChannels$ should not be omitted in validation, the test case design and implementation are straightforward and do not required any special knowledge nor equipment. Methods for automatic identification of fundamental communication cycle schedule are published in [10].

The situation is more complicated than for static schedule in the case of varying parameters e.g. time synchronization or parameters which influence only short time behavior during some distinctive event. An example of these events is cluster Wakeup and Startup. Parameters that were identified important together with the assumption for non-trivial measurement are researched in following sections. Parameters are categorized according to Protocol Operational Control (POC) automaton states.

4.3 Wakeup Parameters

Bus communication starts with a wakeup procedure, which is intended to power up and force the FlexRay POC automaton of each connected node to the ready state. Responsibility for the proper wakeup of a node is divided among the bus driver, host (an MCU) and a FlexRay communication controller. The bus driver should be able to recognize the wakeup pattern and to wakeup other components including the host MCU and communication controller. Remaining steps are driven by the host with support of a communication controller. A summary of relevant parameters is given in Table 4.3

Parameter $gdBit$ expresses nominal bit time, for bit rate 10 Mbit/s it is equal to 100 ns.

4.3.1 pWakeupChannel

$pWakeupChannel$ denotes a channel where the wakeup pattern is transmitted by the node. Evaluation is simple and based on a bus observation only. If the node does

Parameter	Range
<i>pWakeupChannel</i>	Channel A or B
<i>gdWakeupRxLow</i>	8 – 59 gdBitt
<i>gdWakeupRxIdle</i>	8 – 59 gdBitt
<i>gdWakeupRxWindow</i>	76 – 485 gdBitt
<i>gdWakeupTxIdle</i>	45 – 180 gdBitt
<i>gdWakeupTxActive</i>	15 – 60 gdBitt
<i>pWakeupPattern</i>	0 – 63

Table 4.3. Wakeup parameters summary

not detect a wakeup pattern, it sends a wakeup pattern on the channel defined by *pWakeupChannel*.

4.3.2 *gdWakeupTxIdle*, *gdWakeupTxActive*, *pWakeupPattern*

These three parameters define the wakeup pattern waveform. *gdWakeupTxIdle* defines the duration of the bus idle state. *gdWakeupTxActive* is a time period of low bus state. A wakeup symbol consists of one *gdWakeupTxActive* followed by *gdWakeupTxIdle*. The wakeup pattern is a sequence of several wakeup symbols specified by *pWakeupPattern*. Evaluation of these parameters is possible by an oscilloscope or bus sampling with a reasonable sampling period (lower than gdBitt/2 period).

4.3.3 *gdWakeupRxLow*, *gdWakeupRxIdle*, *gdWakeupRxWindow*

Complementary to wakeup pattern transmission, reception of wakeup pattern is controlled by *gdWakeupRxLow*, *gdWakeupRxIdle* and *gdWakeupRxWindow*. Relationship between parameters is shown in Figure 4.2. For the proper node wakeup it is crucial to test its ability to recognize wakeup pattern on the bus.

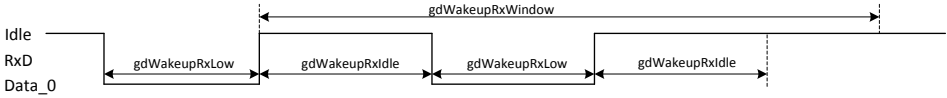


Figure 4.2. Wakeup pattern in context wakeup window

Testing of wakeup pattern recognition is based on the assumption that if a node has recognized the wakeup pattern, it does not send a wakeup pattern itself. The selected wakeup pattern parameter is being decreased from the maximal permitted length until the wakeup pattern is not detected by the ECU under test (EUT). This principle is used for *gdWakeupRxLow* and *gdWakeupRxIdle*. The third parameter is evaluated by changing the right side of inequality (4.1). Modification of *gdWakeupRxLow* value can be used.

$$gdWakeupRxWindow \geq 2 * gdWakeupRxIdle + gdWakeupRxLow \quad (4.1)$$

Evaluation of the wakeup parameters is straightforward, but also an integral part of the exhaustive validation of FlexRay controller parametrization. A similar principle is used for LIN cluster wakeup testing [46]. On the contrary, the LIN wakeup signal timing is fully defined by the standard.

4.4 Startup Parameters

Startup is a key process intended to initialize time synchronization for a whole cluster. Critical parameters influencing a startup procedure are summarized in Table 4.4. A correct setting of FlexRay startup parameters is necessary to ensure fault-free cluster startup.

Parameter	Range
<i>vColdstartInhibit</i>	True or False
<i>pdListenTimeout</i>	1926 – 2567692 μ T
<i>cdCASRxLowMin</i>	29 gdBIt
<i>gdCASRxLowMax</i>	28 – 254 gdBIt
<i>gColdStartAttempt</i>	2 – 31 gdBIt

Table 4.4. Wakeup parameters summary

4.4.1 Type of Node

Test of the startup related parameters has to be distinguished by type of FlexRay node. Four node types are considered in the following test. They are TT-D and TT-L, both variants of coldstart or noncoldstart. Coldstart or noncoldstart node is determined by *vColdstartInhibit* parameter. The parameter is of Boolean type. True denotes the ability to start communication (coldstart node) while False defines a noncoldstart node. A coldstart node starts sending a startup frame, which could be detected by the tester.

4.4.2 *gColdStartAttempt*

This test is relevant for the TT-D coldstart node only, because a TT-L coldstart node never terminates a coldstart attempt (it sends two startup and synchronization frames). Evaluation of *gColdStartAttempt* is possible by the counting of received startup frames according to equation (4.2).

$$gColdStartAttempt = \frac{N_{RSF}}{N_{FCA}} \quad (4.2)$$

Where N_{RSF} is the number of received Startup frames and N_{FCA} is the number of startup frames per coldstart attempt defined by standard, equals to 5.

4.4.3 Collision Avoidance Symbol

Collision avoidance symbol length shall be between *cdCASRxLowMin* and *gdCASRxLowMax*, otherwise it does not have to be recognized. The same idea as for the measurement of wakeup pattern parameters is used. If the collision avoidance symbol is not detected, the tested node tries to send a collision avoidance symbol by itself. The iterative algorithm is used, where the tester starts sending the collision avoidance symbol with a slightly lower length than the minimal permitted value for *cdCASRxLowMin* discovering. Next, the tester continues increasing collision avoidance symbol length to *gdCASRxLowMax*. The range of recognized symbol lengths is finally evaluated.

4.4.4 pdListenTimeout

Measurement of $pdListenTimeout$ uses the property that the $pdListenTimeout$ timer is restarted when the idle state is recognized on the bus. After the timeout expiration the ECU sends a collision avoidance symbol. The time interval measured between collision avoidance symbols should be corrected by $cChannelIdleDelimiter$ and $cdCASActionPointOffset$ values (standard defined constants) that affect bus idle recognition and collision avoidance symbol transmission. The principle is shown in Figure 4.3 and expressed by equation (4.3), where ts_CAS means timestamp of CAS.

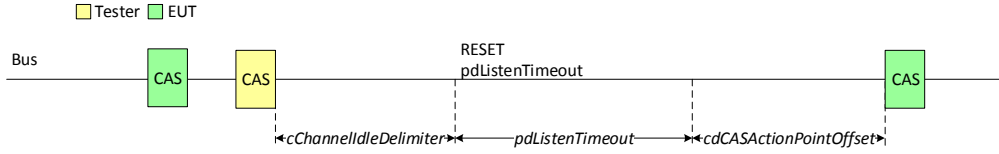


Figure 4.3. $pdListenTimeout$ measurement

$$pdListenTimeout = ts_CAS_{EUT} - ts_CAS_{Tester} - cChannelIdleDelimiter - cdCASActionPointOffset \quad (4.3)$$

4.5 Evaluation of Clock Synchronization Parameters

Functionality of a FlexRay synchronization mechanism is influenced by four parameters. The first parameter is local time unit microtick, the nominal value of which is specified by $pdMicrotick$. Limit values for the offset and rate part of clock correction are $pOffsetCorrectionOut$ and $pRateCorrectionOut$. Rate correction is additionally reduced by the $pClusterDriftDamping$ parameter. Incorrect parametrization is difficult to reveal under normal operating conditions. An example of such a violation and its consequence is provided in section *Motivation for evaluation of configuration parameters*. Parameter names and their ranges are recapitulated in Table 4.5.

Parameter	Range
$pdMicrotick$ (μT)	12.5 ns, 25 ns, 50 ns
$pClusterDriftDamping$	0 – 10 μT
$pOffsetCorrectionOut$	15 – 16082 μT
$pRateCorrectionOut$	3 – 3846 μT

Table 4.5. List of measured parameters

4.5.1 Cycle Length Influencing and Measurement

All presented methods are based on knowledge of communication cycle length (duration time). Cycle length is measured using a principle depicted in Figure 4.4. All incoming frames are marked by a timestamp.

EUT cycle length is synchronized by the FTM algorithm. For three synchronization frames (two of them with the same cycle length are generated by the tester) two of the

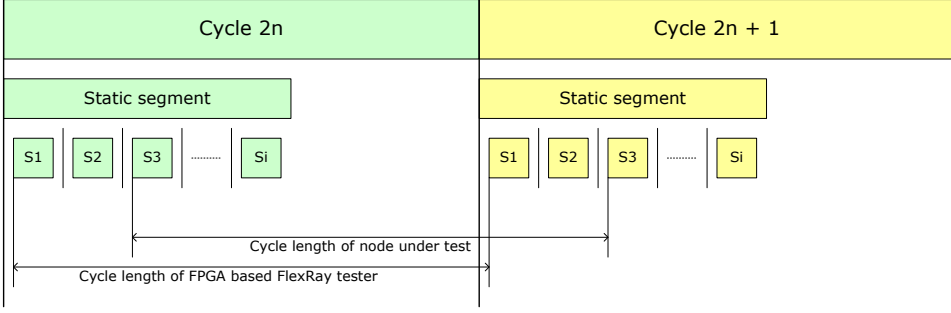


Figure 4.4. Cycle length measurement

three measured time deviations (the highest and the lowest one) are discarded. The EUT frame deviation is always zero and therefore it is either the highest or the lowest one (other two are the same). The remaining deviation value is always that generated by the tester, and the EUT is thus forced to follow its communication cycle length.

■ 4.5.2 Offset Correction Measurement

Some methods presented later require the value of the offset correction in a particular communication cycle. The method for offset correction measurement published in [12] can be utilized. In my opinion, there is a mistake in the formula in section III.C of [12], where the even and odd communication cycles are swapped. Offset correction is applied in the odd communication cycle. To obtain an offset correction with the correct sign, it is necessary to subtract the even cycle length from the odd cycle length. Presumption for the proper use of the method is short-term oscillator stability (constant rate correction). The magnitude of offset correction can be calculated according to equation (4.4).

$$OC(2n + 1) = CL(2n + 1) - CL(2n) \quad (4.4)$$

Where $OC(2n + 1)$ is the offset correction applied in the odd communication cycle ($2n + 1$). $CL(2n + 1)$ is the length of the odd communication cycle. $CL(2n)$ is the length of the previous even communication cycle.

■ 4.5.3 pdMicrotick

Usually pdMicrotick is directly derived from a local clock source. Depending on the desired communication speed, three nominal values are possible – 12.5 ns, 25 ns and 50 ns. The complete communication cycle schedule is derived from this parameter. The *pdMicrotick* parameter represents the minimal possible change in the FlexRay node timing. The change is observable by a frame transmission transmit time for frames transmitted in a static slot. The proposed method deals with the presumption that clock synchronization works with microtick resolution and two FlexRay nodes are never synchronized absolutely. The cycle length is affected by components from equation (4.5).

$$CL(2n + 1) = pMicroPerCycle + RC(2n + 1) + OC(2n + 1) \quad (4.5)$$

A small difference between EUT and tester communication cycle lengths (denoted Δ) always exists. This small difference is accumulated over a few communication cycles until it exceeds magnitude of $1 \mu\text{T}$ (as shown in Figure 4.5). Within the following odd communication cycle the EUT synchronization mechanism corrects the difference by adding $1 \mu\text{T}$ to the actual offset correction value. $p\text{dMicrotick}$ value can thus be evaluated by precise measurement of EUT communication cycle lengths. The minimal distance in the communication cycles histogram is equal to $p\text{dMicrotick}$.

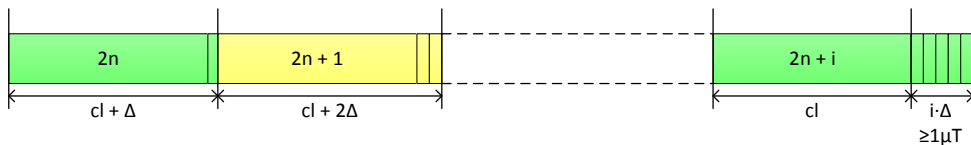


Figure 4.5. Cycle difference accumulation

4.5.4 $p\text{ClusterDriftDamping}$

The parameter value is subtracted from the actual calculated value of rate correction. Thus, $p\text{ClusterDriftDamping}$ can be interpreted as the insensitivity zone of rate correction. A clock frequency difference below this limit has to be corrected by an offset correction (red squares in Figure 4.6). For example, if $p\text{ClusterDriftDamping}$ is equal to $5 \mu\text{T}$ and the clock frequency difference is higher than $5 \mu\text{T}$ per cycle, the minimal applied offset correction is $10 \mu\text{T}$ per each odd communication cycle.

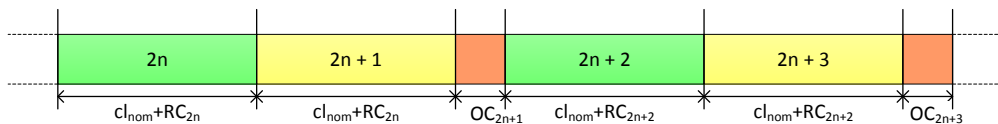


Figure 4.6. Offset correction affected by $p\text{ClusterDriftDamping}$

For measurement of the parameter it is necessary to assure that the tester and the EUT clock difference is higher than maximal permitted $p\text{ClusterDriftDamping}$ value, which is $10 \mu\text{T}$. Afterwards, the value can be extracted from measured cycle lengths according to equation (4.6) (it does not reflect sporadic $1 \mu\text{T}$ corrections). Division by two reflects the accumulation of the parameter value over two communication cycles.

$$p\text{ClusterDriftDamping} = \frac{CL_{EUT}(2n + 1) - CL_{EUT}(2n)}{2} \quad (4.6)$$

Where $CL(2n + 1)$ is the length of the odd communication cycle and $CL(2n)$ is the length of the previous even communication cycle.

4.5.5 $p\text{RateCorrectionOut}$

The idea for $p\text{RateCorrectionOut}$ measurement is to slightly push the EUT synchronization mechanism to its limits. After the limit is reached, EUT stops sending frames. The measurement takes place in even communication cycles, since only even cycles are not affected by the offset correction. Maximal and minimal measured cycle lengths

determine the value of $pRateCorrectionOut$ according to equation (4.7). Either a positive or negative Rate correction value is applied (lengthening or shortening of communication cycle, respectively). To obtain the actual value of the parameter it is necessary to divide the measured difference of cycle lengths by two.

$$pRateCorrectionOut = \frac{CL_{max} - CL_{min}}{2 * pdMicrotick} \quad (4.7)$$

Figure 4.7 depicts this measurement step by step. The first cycles are of the same length (EUT and tester are synchronized). Next, the tester starts forcing EUT to shorten its communication cycle using the principle described in section Cycle length control and measurement. When the lower limit is discovered, the tester starts with increasing the communication cycle length and the upper limit is discovered consequently. Missing communication is expressed by white bars in Figure 4.7.

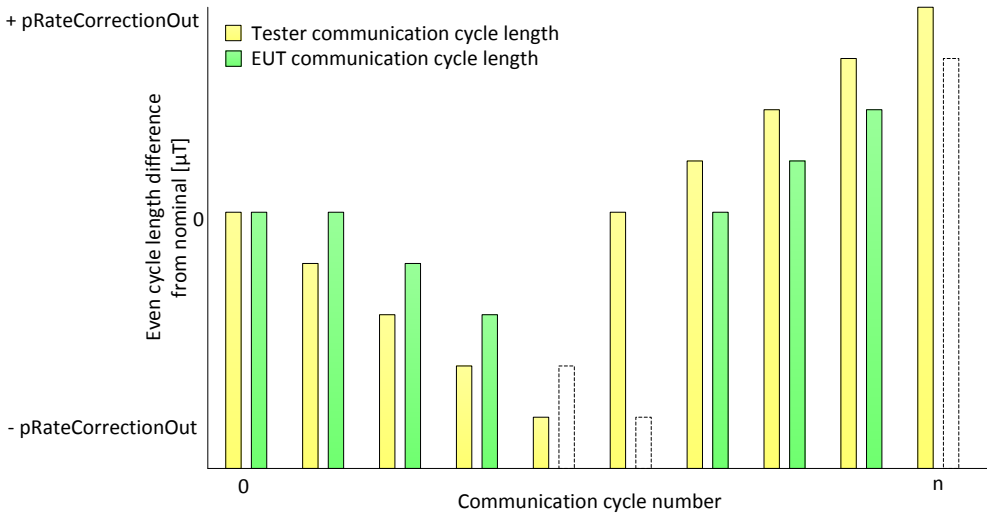


Figure 4.7. Rate correction affected by $pClusterDriftDamping$

4.5.6 pOffsetCorrectionOut

The offset correction is calculated using the FTM algorithm from the deviations measured in each odd communication cycle. This value is applied at the end of the same communication cycle. Thus the offset correction value evaluation is only possible within the corresponding odd communication cycle. EUT offset correction is induced by a fast shift of the tester’s synchronization frames time position. This shift magnitude should be corrected by the EUT offset correction, whose value is measured using the method described in section *Offset correction measurement*. If the required offset correction value is higher than the $pOffsetCorrectionOut$ limit, only this limit value is applied. This method is further limited by the actual duration of the static slot. The shifted test frame may not violate the static slot boundaries. The area of possible synchronization frames shifting is expressed by green shaded area in Figure 4.8.

The maximal negative offset correction value (frame shift towards start of static slot) that can be measured is limited by $gdActionPointOffset$ and is denoted by the minus superscript in equation (4.8).

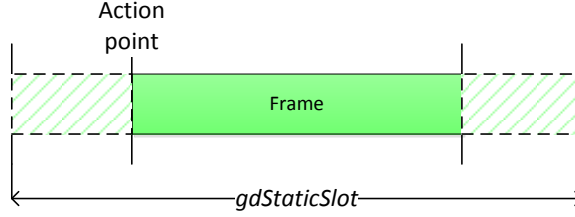


Figure 4.8. Offset correction measurement limitation

$$OC^- [\mu T] = gdActionPointOffset * (gMacroPerCycle \div pMicroPerCycle) \quad (4.8)$$

The maximal positive offset correction value (a frame shift towards the end of static slot) that can be measured is limited by the frame length, $gdStaticSlot$ and $gdActionPointOffset$, and is denoted by the plus superscript in equation (4.9).

$$OC^+ [\mu T] = (gdStaticSlot - gdActionPointOffset) * (gMacroPerCycle \div pMicroPerCycle) - FrameLength \quad (4.9)$$

$$FrameLength [\mu T] = (gdTSSTransmitter + cdFSS + 80gdBit(header + trailer) + 2 * gPayloadLength * 10 + cdFES) * (cSamplesPerBit \div pSamplesPerMicrotick)$$

Definitions of all parameters used in the equations above can be found in FlexRay standard [2].

A frame shifted over the limits is marked invalid by EUT internally and thus not used for synchronization. This behaviour can be used for evaluation of $gdActionPointOffset$ and $gdStaticSlot$ parameters.

4.6 Validation on Real FlexRay Network

Proposed methods were experimentally evaluated on real FlexRay network. Experiments were performed using setup shown in the figure 4.9. The control element is PC. Oscilloscope with FlexRay frame decoding feature was used as an auxiliary monitoring device. The major part is FPGA tester node, and as EUTs two different MCUs were used. Used network hardware (predominantly CTU developed) is described in the following text.

4.6.1 FPGA FlexRay Controller

Due to limited capabilities of production FlexRay CC, a VHSIC Hardware Description Language (VHDL) tester implementation was used as the evaluation platform. The controller is designed in VHDL and implemented on an Altera FPGA as master thesis [47]. Features of this controller were specified to address this research requirement. Especially precise control of frame transmission and receiving time timestamping. The

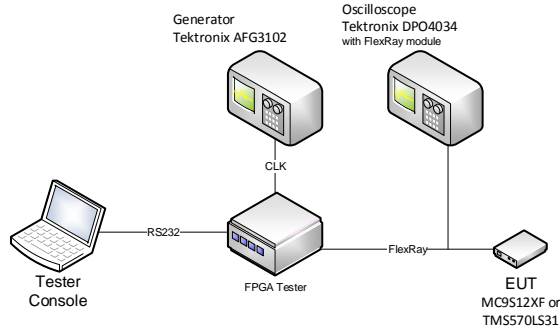


Figure 4.9. Measurement Setup

NIOS II microprocessor is used as a control element and runs testing methods. The synthesis was done for Cyclone II EP2C20F484C7N chip. For integration of NIOS softcore MCU, FlexRay core, and other necessary function blocks (UART, SRAM, PLL ...) Qsys – System Integration Tool was used.

The methods are implemented in C using NIOS II Integrated development environment (IDE) for Eclipse. Tester is operated by a serial terminal connected to a PC by RS232. The menu-based user interface was also implemented. The structure is similar to parameters and attached methods description in sections above. The method is labeled by the parameter name and selectable by numeric keyboard. Internal system structure as it is interconnected by Qsys is depicted in figure 4.10.

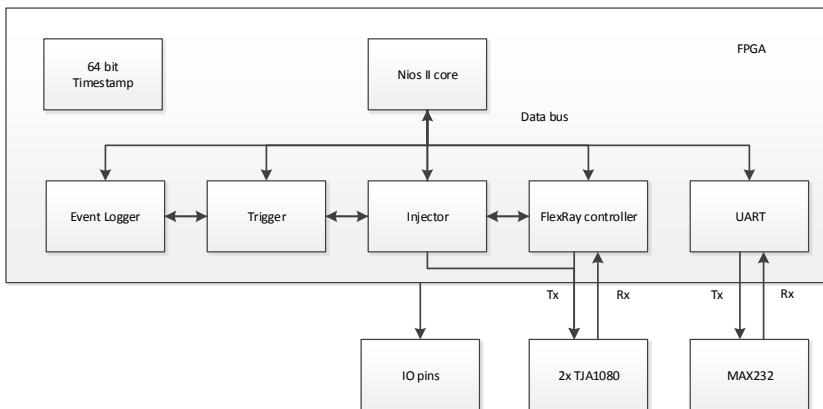


Figure 4.10. Structure of FlexRay System on a chip (SoC) [47]

4.6.2 FlexRay Hardware – EUT

Two EUT types were used for measurement method evaluation. Both controllers are implemented according to specification version 2.1 [43]. Differences in CCs parameterization are mentioned in 4.2. Both EUTs use TJA1080 bus drivers. The first EUT is a FlexRay board developed in CTU in past. It is based on a 16-bit microprocessor MC9S12XF with integrated Freescale FlexRay controller. The board was used as wheel speed sensor unit for a FlexRay ABS prototype. Hardware description can be

found in [48]. Parameterization of CC was modified by MCU programming using CodeWarrior integrated development environment. Except FlexRay parameter set, same firmware as for wheel speed sending unit was used.

The second EUT is Ethernet–FlexRay gateway developed by two bachelor program student as their thesis [49][50]. It is based on a 32-bit microprocessor TMS570LS31 from Texas Instruments with integrated Bosch FlexRay controller called E-Ray [51]. This gateway is the ideal piece of hardware for any FlexRay development. In a nutshell, it is small Printed Circuit Board (PCB) with mentioned MCU, FlexRay and Ethernet physical interfaces accompanied by the necessary electronics. FlexRay communication is user controllable by an Application. Gateway provides sufficient data throughput for Real-time monitoring. FlexRay setting is stored in custom format XML file. Specific features such as node startup, integration to running cluster or just listening are also application controllable.

4.6.3 Experiments

The FlexRay network used for practical evaluation of the presented methods was parameterized according to the Table 4.6. Parameters closely related to the measurement methods are explained in corresponding paragraphs; a detailed description of each parameter can be found in the FlexRay Communication System Specification [2]. The testing network consisted of just two nodes (Tester and EUT). Standard Category 5e cable was used for interconnection.

Parameter	Value
<i>gdActionPointOffset</i>	3 MT
<i>gdDynamicSlotIdlePhase</i>	1
<i>gdMinislot</i>	40
<i>gdStaticSlot</i>	50
<i>gdSymbolWindow</i>	13 MT
<i>gdTSSTransmitter</i>	11 gdBit
<i>gMacroPerCycle</i>	5000 MT
<i>gNumberOfMinislots</i>	22
<i>gNumberOfStaticSlots</i>	60
<i>gOffsetCorrectionStart</i>	4920 MT
<i>pClusterDriftDamping</i>	1 μ T
<i>pDecodingCorrection</i>	56 μ T
<i>pDelayCompensation[A]</i>	1 μ T
<i>pDelayCompensation[B]</i>	1 μ T
<i>pdListenTimeout</i>	401202 μ T
<i>pLatestTx</i>	21
<i>pMacroInitialOffset[A]</i>	5 MT
<i>pMicroInitialOffset[A]</i>	5 μ T
<i>pMacroInitialOffset[B]</i>	23 MT
<i>pMicroInitialOffset[B]</i>	23 μ T
<i>pMicroPerCycle</i>	200000 μ T
<i>pMicroPerMacroNom</i>	40 μ T

Table 4.6. Test Network Configuration

Measured values were compared with values configured in communication controller registers. As far as I know, similar work with comparable results has not yet been published. The results fully correspond with assumptions with respect to clock frequency differences discussed in section 4.7. To eliminate this issue, the validation tests are made using a tester clock frequency very close to the clock frequency of the EUT (driven by arbitrary generator Tektronix AFG3102). The tester clock frequency was 80.0206 MHz for the ECU with E-Ray controller and 80.0011 MHz for the ECU with Freescale controller. Pre-set values of particular measured parameters are chosen from the interval of all possible values. Global FlexRay controller settings are mentioned in Table 4.7 (with exception of measured parameters). Bus monitoring was done using a Tektronix DPO4034 oscilloscope. Regrettably methods were not tested with a CC implemented according to last specification 3.0.1. Since none was on the market in time of work realization.

Parameter	Set Value	MC9S12XF	TMS570LS31
<i>gdCASRxLowMin</i> [gdBit]	29	29	29
<i>gdCASRxLowMax</i> [gdBit]	64	64	64
	83	83	83
	120	120	120
<i>pdListenTimeout</i> [μ T]	1926	1934	1933
	401202	401211	401211
	800000	800010	800009
<i>pdMicrotick</i> [ns]	25	25	25
<i>pClusterDriftDamping</i> [μ T]	1	1	1
	3	3	3
	10	10	10
<i>pRateCorrectionOut</i> [μ T]	500	500	499
	600	599	600
	700	700	699

Table 4.7. Selected experimental results

Real measurement of parameters *gdCASRxLowMin* and *gdCASRxLowMax* shows that designed methods are able to identify actual values with bit level resolution precisely. Measurement of the *pdListenTimeout* parameter after subtraction of CAS symbol length according to equation (4.3) contains a maximal error of 10 μ T or 250 ns. This error is caused by a combination of three factors. They are the remaining difference in tester and EUT clock frequencies, the delay in the receiving path of the controller and the tester time stamping implementation. The practical impact of this error on result usability is nevertheless negligible. Methods for identification of *pdMicrotick* and *pClusterDriftDamping* work fully within expectations. Vital evaluation *pRateCorrectionOut* works according to presumptions.

Results of offset correction limits evaluation are summarized in Table 4.8. Due to the limits represented by equations (4.8) and (4.9), two FlexRay network schedules were used. The first schedule, labelled as default setup, is the setup previously mentioned in Table 4.6. The modified setup enables the full range measurement of permitted offset correction. The schedule of the static segment was changed to the static slot length 100 MT, 30 static slots per communication cycle and the action point offset 10 MT.

Set Value	Expected	MC9S12XF	TMS570LS31
<i>pOffsetCorrectionOut</i>	$OC^+ OC^-$	$OC^+ OC^-$	$OC^+ OC^-$
1201 Default setup	224 120	230 128	230 130
1201 Modified setup	1201 400	1201 410	1200 410
300 Modified setup	300 300	300 300	300 300

Table 4.8. pOffsetCorrectionOut measurement (all values in μT)

Results correspond with presumptions with the exception of static slot boundaries violation. Experiments demonstrate that the used controllers consider received frames valid approximately 10 μT before and after the defined static slot boundaries. All experimental results indicate that the presented methods are able to evaluate critical parameters of a single FlexRay node.

4.7 Measurement Accuracy and Speed

Tester and EUT clock frequencies are never equal. There is always a small difference between frequencies, usually a few microticks per communication cycle in terms of FlexRay. Clock frequency differences must be taken into account in order to evaluate the presented parameters. It is assumed that the difference of oscillator frequencies is at worst 1500 ppm from the nominal value. This is the maximal permitted clock deviation according to standard[2]. For a nominal clock frequency of 80 MHz it is equal to a deviation of up to 120 kHz. The quality of an used oscillator should be subject of special testing.

The test objective is to reveal EUT parameter values from communication controller registers; therefore, it is necessary to know the actual EUT clock frequency or minimize clock frequency difference between the tester and the EUT. Our experiments were done using the second approach. Before the measurement, the clock source (Tektronix AFG3102) was set as close to the EUT actual clock frequency as possible. The tester clock frequency setting was done using observation of tester rate and offset correction actual values. The achieved clock difference was better than 1 μT per communication cycle. The explanation can be found in the paragraph about *pdMicrotick* measurement. According to [13] the short term stability the measurement methods rely on is usually not the issue.

In the context of the intended application, measurement time is not critical. Moreover, it is short – typically in the range of a few communication cycles. An overview of measurement times is stated in order to provide complete characteristics of the presented methods. Results are summarized in Table 4.9. The measurement time always depends on the parameter value and mostly on global network setup. Presented methods can be divided into two groups. Measurement algorithms from the first group work in a fixed number of steps. In the second group there are iterative algorithms, where the number of steps depends on the parameter value and the network setup. Measurement time estimation is provided in Table 4.9. in the column Approximate Time.

Parameter	Algorithm – steps	Approximate Time
pdListenTimeout 1926 – 2567692 μ T	fixed	2 * pdListenTimeout
cdCASRxLowMin cdCASRxLowMax 29 – 254 gdBit	Depends on cdCASRxLowMax	gdCASRxLowMax – cdCASRxLowMin + pdListenTimeout
gColdStartAttempt 2 – 31	Depends on gColdStartAttempt	(5 * pMicroPerCycle + pdListenTimeout) * gColdStartAttempt
pdMicrotick 12.5 25 50 ns	Depend on clock difference	1.25 s – worst case for setup in Table 4.6
pClusterDriftDamping 0 – 10 μ T	Depend on pClusterDriftDamping	2 * pMicroPerCycle * pClusterDriftDamping
pOffsetCorrectionOut 15 – 16082 μ T	Fixed	2 * pMicroPerCycle
pRateCorrectionOut 3 – 3846 μ T	Depend on pRateCorrectionOut	2 * pMicroPerCycle * pRateCorrectionOut

Table 4.9. Overview of measurement duration

4.8 Summary

The aim of this work was to fill the gap in the area of (available) FlexRay communication system measurement and testing techniques. The novel complex set of methods was proposed for measurement of the Data-link layer parameters of a FlexRay network node. Presented methods cover the parameters related to all three FlexRay node operational control states, i.e. the cluster wakeup, startup and normal operation. Individual measurement methods are described in detail; they were designed with respect to usability in a real-world testing by means of a straightforward implementation and short execution time. Compared to the cited publications, the new methods are based not only on passive communication monitoring, but active stimuli are used as well. The validity of the presented measurement methods is checked by experiments. Experiments have been conducted on the real FlexRay ECUs based on two different FlexRay controller implementations (Freescale and Bosch E-Ray) to avoid implementation specific results. Detailed experimental results with measurement setup and discussion of measurement accuracy are included. The experimental results prove the validity of all new methods. For *pOffsetCorrectionOut* measurement method, they simultaneously show its fundamental dependence on the static slot configuration.

Chapter 5

Integration Testing

The general objective of this part is the improvement of the Integration testing process by utilization of the Model-Based Testing techniques. Objectives in details are covered by section 3.2. The work related to the automatic test generation for Integration testing has started let say from scratch. Thus, the significant amount of work is dedicated to the built background for the scientific work. Namely, it includes selection of the formal model as the basis for consequent automation. The proposed concept was implemented. Moreover, the implementation is divided into software tool Taster and new HIL platform based on NI products. Without this playground, there would not be any platform for evaluation of proposed testing methods. At the beginning, several benefits of MBT approach are stated to explain why this general idea was chosen.

- System specification in executable form (a model) opens ways to various computer processing.
- From one model, a vast number of test with different objectives can be generated.
- Model development can reveal bugs in a specification.
- Work amount reduction – when it is done in right way.
- Maintenance of a model is usually easier than the maintenance of the original test suite.
- Cost reduction – additional tests can be generated with zero costs.
- Reduced work is the boring one, saved time could be invested in the better way.

5.1 Selection of Formal Model

Planned approach strongly depends on the suitable formal model. A number of existing formal specification languages is relatively high. The question is which is the best suitable for the application in the context of Integration testing. Tools and formal languages can be further divided by many aspects; from theoretical properties to purely practical issues like if the tool is user-friendly. Some of the considered options with characteristic features are summarized in Table 5.1.

Model	RT	Reactiveness	Concurrency	Tool
Aldebaran	○	●	○	JTorX
AsmL	○	●	●	Spec Explorer
Network of TA	●	●	●	UPPAAL
Timed-Arc PN	●	●	●	TAPAAL
UML/SysML	●	●	●	RT-Tester

Table 5.1. Selection of Formal Model

Choice of formal model is comparable to a selection of programming language for a project. There are some options. Some of them are not suitable. Some of them are. From the rest, technically suitable options, it is hard to choose right one considering the only list of features. Pros and cons will appear together with some experience with chosen language. Also, success not only depends on the selected language, different IDEs, compilers and other connected tools may produce different results. Based on this perspective following requirements were written.

Requirements for modeling formalism

Selected formalism should be capable to model complex distributed system with multiple inputs and outputs (analog and digital). These kinds of systems are usually referred as Reactive Systems. Time dependencies should be easily expressible. Possibility to model multiple time domains could be beneficial. Formalism should be able to deal with some level of parallelism. Model of SUT should be executable. Some reasoning about model should be beneficial (completeness, reachability, etc.). The specification should have appropriate data representation for further processing (e.g. no proprietary binary formats).

A tool experience

With specified requirements, academics and companies web sites are searched for downloadable MBT tools. The intention was to find a tool, which can create a good starting point for this dissertation project. No such a tool was available to thesis author on the department at the time of project start. First attempt was made with Aldebaran (.aut) specification format and tool JTorX [52] v. 1.9.4. JTorX is JAVA reimplementation of original TorX. Aldebaran describes a system in form of transitions. One line per one transition. The tested implementation is not able to handle RT properties. Also, complex model creation seems difficult in given format.

Another remarkable tested tool was SpecExplorer [53]. It is the plugin for Microsoft Visual Studio developed by Microsoft Research division. Input for the tool is not a model in the traditional view; it works with annotated C# code or Cord scripts. A wide range of examples provides a good knowledge base for a start. The tool is highly matured and usable. Regrettably, handling of RT properties and model creation with the absence of C# code (SUT is not probably implemented in C# and is not available at all) makes Spec Explorer not suitable.

Few other tools - review articles [17][54] sometimes contain up to hundreds of tools, were tested with vague results. The maturity, as well as documentation level, excluded them from further work.

Next tried tool was UPPAAL. It is Timed Automata based model checker. The software is multi platform and requires only Java Runtime Environment. Although *verifyta* engine is not controllable outside the tool. It provides very usable Timed Automata model editor. The user can design a model in the graphical environment. The model is stored in UPPAAL 4 XML file format. TA can handle RT, concurrency, and other required features as they were designed for it. On this view, Timed Automata and their implementation in UPPAAL tool were selected for experimental evaluation of their suitability as a model editor for further work.

Arguments in behalf of Timed Automata:

- Well accepted FSM based notation
- Multiple clocks domains
- Boolean and integer data types
- Automata can be connected into network
- Synchronization of Automata by channels
- Available tool in high maturity level
- The tool can be used as model editor
- Model is stored in XML file format suitable for further software processing

■ 5.1.1 Passenger Car Inner Light Model

The example of natural language specification of a part of SUT was modeled in UPPAAL. The example describes the behavior of a passenger car inner light. The inner light state (on/off) is determined by a couple of input signals. Behavior depends on the three-position inner light switch (on/on if a doors are open/off) together with signals derived from doors positions (open/closed), the state of central locking (lock/unlock) and ignition state (position of ignition key).

The simplest case is if the interior light switch position is off. No matter on other signals, the state of the light must be off. A similar situation is if the inner light switch position is on. The light must be switched on with an exception when the car is locked. After the car has been locked, the light must turn off within three seconds (slow dimming). When the car has been unlocked, light must turn on immediately (within one second). The most complicated behavior of the inner light is associated with a switch position labeled “on if a door is open.” Previously described reaction on central locking stays valid, and the light is switched on when the car has been unlocked. If the doors are not opened within 30 seconds interval, the light is turned off. In opposite case, a 30 seconds interval counter is restarted, and light is switched off after another 30 seconds interval is elapsed, or the ignition key position is changed to ignition on. Turning an engine off (ignition key position off) switches the inner light on for another 30 seconds. The interval counter is restarted by any door activity (open/close). Selected specification language should be capable of model similar kind of SUT, but real automotive distributed systems are usually much more complex. Need for expression of time constraints is clear from the previous paragraphs.

Example specification was modeled by timed automata. The situation corresponds to expected workflow because in our real situation formal models are not available. Passenger car inner light is modeled by two automata. First automaton describing the behavior of inner light controller (typically part of an ECU) is depicted in the figure 5.1. Model states represent situations that determine the controller behavior (locked/unlocked car, on/off ignition and switch position). Initial transition happens when the user unlocks the car. After the car was locked automata return to the initial state.

Second automaton depicted in the figure 5.2 expresses behavior of car driver. E.g. driver unlocks the car, opens the door, closes the door, and turns the ignition on. Eventually, changes switch position, turns the ignition off, opens a door, closes a door and locks the car. System states are expressed by variables (Integer and Boolean type). Models interactions are synchronized by urgent channels. The second model is

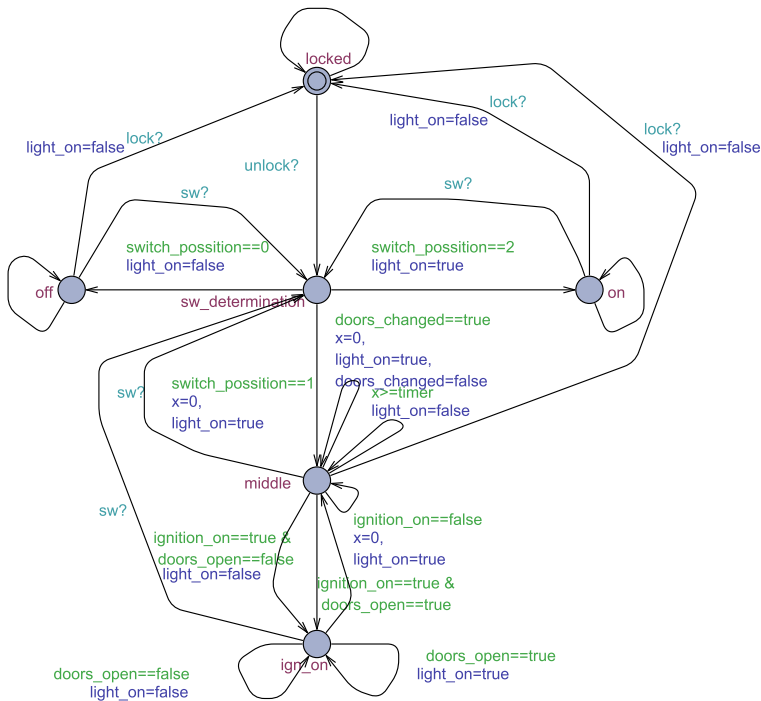


Figure 5.1. Inner Light Controller

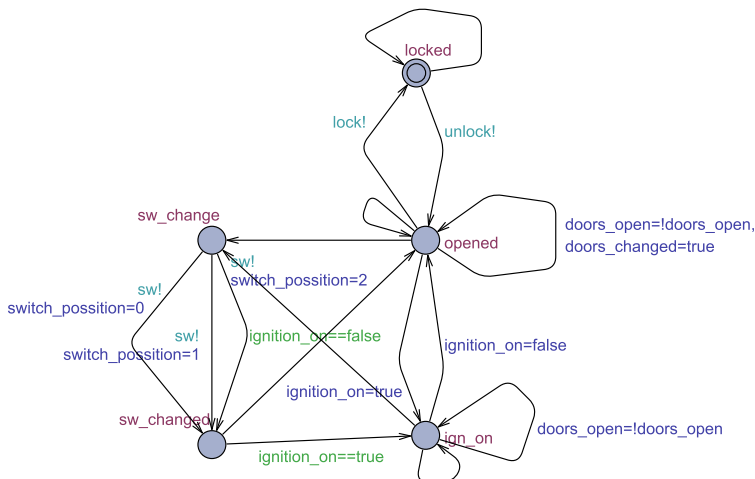


Figure 5.2. Driver model

important for the presented concept. The test can be driven using the model. Time constraints (inner light timer) are expressed by clock variable x .

Gained experience with modeling the light controller as the network of two timed automata is concluded following way. The behavior of the controller as well as the behavior of a driver is easily expressible. Despite the first one is electronic system and the second one is human being. To keep model well-arranged UPPAAL provides some construct from ANSI C. Progress of individual automata is synchronizable by

channels. Model is stored in XML format, which is employable for further processing. Models developed in UPPAAL environment are the promising beginning for following objectives.

5.2 Timed Automata Theory

Origin of the Timed Automaton is in the rich theory of finite state machines. The way from FSM to Timed Safely Automata (known as Timed Automata) is described. It provides a valuable perspective for the rest of the work. In a nutshell theory is connected in this direction: FSM \rightarrow ω -automaton \rightarrow Büchi automaton \rightarrow Timed Automaton. The cause of different symbols used for the same topic is evoked by an effort to ensure consistency with source papers. Therefore, for instance, the initial state is marked by a different symbol for each automaton type.

5.2.1 Finite-State Machine

FSM is the computational model which contains states and transitions (arrows). One state is initial, and one or more states are final. Transitions are labeled by symbols which enable relevant transition – formally transition function. Usual representation of FSM is in graphical form or by transition table. If more than one states are in a column of transition table, the FSM is named nondeterministic.

Acceptor deterministic FSM is commonly mathematically defined [55] by quintuple $(Q, \Sigma, \sigma, q_0, F)$:

- Q is a non-empty set of automaton states,
- Σ is a finite set called the alphabet (input symbols),
- $\sigma : Q \times \Sigma \rightarrow Q$ is state transition function (would return next state),
- $q_0 \in Q$ is initial state,
- $F \subseteq Q$ is set of final (accepting) states.

5.2.2 Automata for Infinite Input

FSM runs on finite input. A reactive system is usually considered non-terminating. A theoretical solution to this problem is ω – automaton or stream automaton, which is a modification of FSM for infinite input sequences. Instead of accepting states, accepting condition is used. Stream automaton types are further distinguished by acceptance condition. List of the most known conditions variants follows.

Acceptance conditions

- Büchi condition
- Rabin condition
- Streett condition
- Parity condition
- Muller condition

■ 5.2.3 Büchi Automaton

Stream automaton version, which uses Büchi accepting condition is named Büchi automaton after his inventor Julius Richard Büchi.

Büchi automaton formal definition [56] is quintuple $(Q, \Sigma, \sigma, q_0, F)$:

- Q is a non-empty set of automaton states,
- Σ is a finite set called the alphabet (input symbols),
- $\sigma : Q \times \Sigma \rightarrow Q$ is state transition function (would return next state),
- $q_0 \in Q$ is initial state,
- $F \subseteq Q$ is the acceptance condition.

Büchi automaton accepts exactly runs when F is non-empty. At least one of accepting states have to occur infinitely often in a run. Transferability of LTL formulae to Büchi automaton is one of important feature used in MC. It is the direct ancestor for Automata for Real-Time Systems modeling which were introduced by Alur and Dill [19].

■ 5.2.4 Timed Automaton

The original [19] version of Timed automaton is defined as follows:

A timed automaton is a tuple $\langle \Sigma, S, S_0, C, E \rangle$, where

- Σ is a finite set called the alphabet (input symbols),
- S is a non-empty set of automaton states,
- $S_0 \in S$ is initial state,
- C is a finite set of clocks,
- $E \subseteq S \times S \times [\Sigma \cup \{\varepsilon\}] \times 2^C \times \Phi(C)$ is the set of edges (transitions).

In couple with a set of accepting states $F \subseteq S$ it creates Timed Büchi automaton. Accepting condition works in the same way as for Büchi automaton. The accepted run is run in which at least one of accepting states occurs infinitely often.

Commonly used is the modified definition introduced by [21]. Instead of accepting condition the local invariants are used. Local invariant is the downward closed condition on clock variable. This condition limits time, which automaton can spend in a location. The original name is Time Safety Automata. Currently, is by term Timed Automaton almost certainly mentioned this definition (for the text consistency repeated from 2.2.4):

A timed automaton [21][20] A is a tuple $A = (N, l_0, E, I)$, where

- N is a finite set locations (or nodes),
- $l_0 \in N$ is initial location,
- $E \in N \times \mathcal{B}(C) \times \Sigma \times 2^C \times N$ is the set of edges and
- $I : N \rightarrow \mathcal{B}(C)$ assigns invariants to locations
- We shall write $l \xrightarrow{g,a,r} l'$ when $\langle l, g, a, r, l' \rangle \in E$

Local invariant is constraint in form $x < n, x \leq n$ where n is natural number. Invariant is equivalent to accepting condition.

Semantics

Semantics of Timed Automaton operation is defined as Timed Transition System with following rules [20]:

1. $\langle l, u \rangle \xrightarrow{d} \langle l, u + d \rangle$ if $u \in I(l)$ and $(u + d) \in I(l)$ for a non-negative real $d \in \mathbb{R}_+$
2. $\langle l, u \rangle \xrightarrow{a} \langle l', u' \rangle$ if $l \xrightarrow{g, a, r} l', u \in g, u' = [r \mapsto 0] u$ and $u' \in I(l')$

Letter l denotes location; u denotes clock assignment mapping function; g is a guard, a is an action and r is clock reset operation. The first rule allows idle in a state for a time bounded by location invariant. The second rule is applied if an allowed transition is taken.

Parallel Composition

Modeling of a real system is almost impossible with a single TA model. Therefore a system is often composed by multiple automata. Decomposition to several models with an adequate number of states is also helping clarity of whole model. Parallel composition operator from Calculus of Communicating Systems (CCS) is used in theory of TA. The composition of two TA is denoted by pipe $|$ character. The theoretical background for this operation provides parallel composition operator know from Communicating Sequential Processes (CSP) [57]. Deeper explanation of the theory is provided e.g. in [22][58]. Synchronization between automata is done synchronously by synchronization channels or asynchronously by shared variables.

5.3 Testing Workflow Proposal

Considering all the fact discussed in the previous text, MBT concept suitable for the Integration testing is proposed. Schematically it is shown in the picture 5.3. In the beginning, there is some textual document that specifies SUT. Instead of static test sequences, the model-driven approach is used. The key element is dual purpose TA model derived from a specification. Correct SUT behavior has to be captured by the model. This part is called System Model. It is complemented by environment section which specifies input behavior. Syntactically are both part same. Differentiation is only on by purpose.

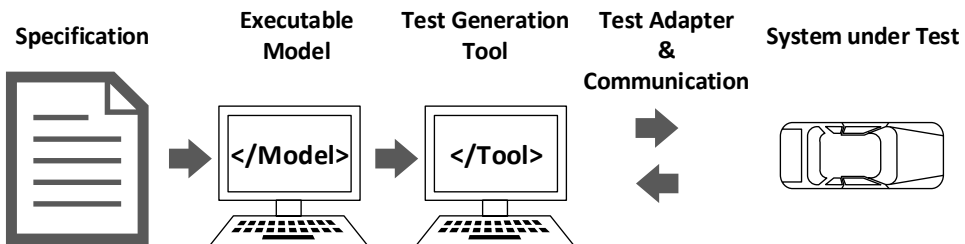


Figure 5.3. MBT concept for Integration testing

Prepared TA model is loaded into a software tool. The dominant part of the testing process is performed in this program. All activity is the model-driven. An input file is syntactically analyzed. The online testing approach is chosen because it suits loop nature of HIL method. Test inputs are generated by TA network exploration. Moreover, these inputs periodical feed SUT. After each step, SUT outputs are compared with simulated ones to determine conformance between model and SUT behavior. Exploration strategies, which directly influenced input generation are covered in the special section.

Targeted deployment of this work is HIL test place. The connection between MBT software tool and testing facility is necessary. In MBT world it is usually realized by a piece of software called test adapter. Our industrial cooperation results in two types of test adapter. First test adapter directly communicates using .NET Application Programming Interface (API) [59] with NI VeriStand. It allows direct interconnection with a broad range of NI modular hardware suitable for performing HIL test. Direct connection to test performing hardware preserve straightforward control on input and output timing. The second variant counts with an indirect connection to the testbed. EXAM is utilized as an additional layer between MBT testing tool and testing hardware. It becomes harder to control timing properties. The advantage is the opening of access to all testbeds on industrial partner side, which are based on various hardware platforms. The concept implementation is depicted in Figure 5.4. Instead of abstract variant 5.3, specific model, tool and adapter are presented.

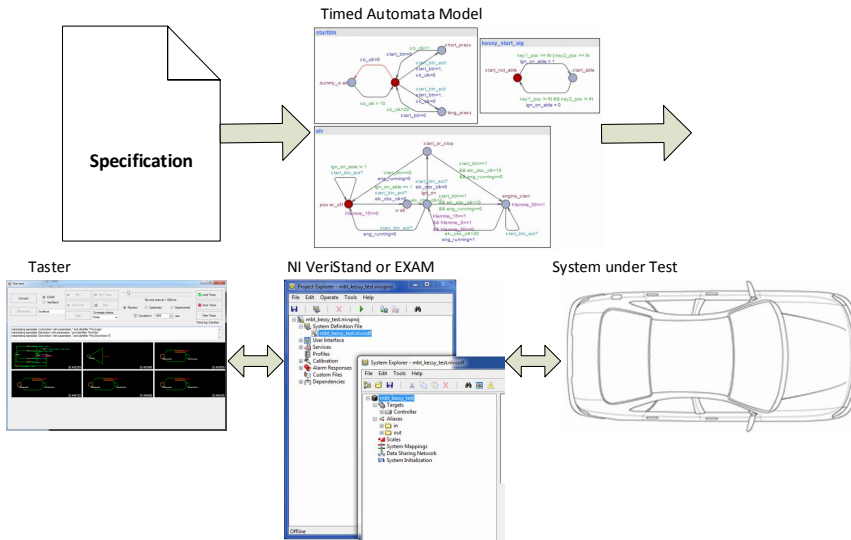


Figure 5.4. MBT Workflow in details

5.4 Testbed for Comfort Systems

The target platform for the work is HIL test place for testing automotive distributed systems. The testbed is based on National Instruments hardware and software solutions. Test purpose is Integration testing. Typical SUT is a bunch of automotive

Electronic Control Units (ECU). The system is capable of handling its input and outputs, which are predominantly digital. Handling of analog I/O is also possible. One or multiple CAN buses are monitored and, if required, a part of CAN traffic is simulated. Diagram is depicted on the picture 5.5.

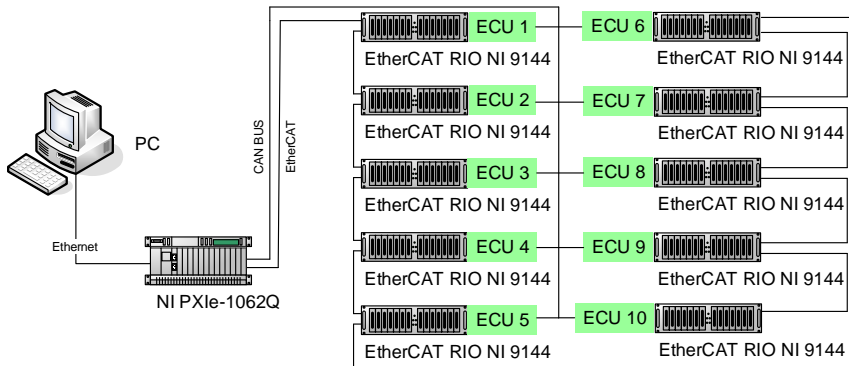


Figure 5.5. NI HIL platform

The central element is industrial controller NI PXIe-8133 RT in chassis NI PXIe-1062Q. Similar to the tested system, testbed architecture is also distributed. The controller communicates with NI 9144 8-Slot EtherCAT Slave Chassis. Each EtherCAT slave handles one ECU. The communication link between master and slaves is Ethernet with EtherCAT protocol to maintain synchronization over the system. A slave is equipped with C series modules according to corresponding ECU electrical connection. Specific functions, behind NI hardware capabilities, are handled by a custom hardware solution. An example of this kind of function is CAN message manipulation on data link layer.

The software can be divided into two parts. The first is non-RT on standard PC with Windows operation system. The second part is RT engine which runs on PXIe. It is NI proprietary version of Pharlap RT OS. The main loop frequency is 1 kHz. Therefore testbed can feed SUT inputs and monitor outputs with 1 ms resolution. The actual software platform is based on NI product version 2015 or 2015 SP1. The system is based on NI VeriStand, and its functionality is extensible in LabView programming environment. PC runs Windows 7 operation system. Tests are implemented and executed by EXAM. More software architecture detail is in section 5.9.

5.5 System Modeling

The concept of Timed Automata is implemented in UPPAAL tool, which is used as a modeling environment for the proposed solution. Beyond theoretically described time and transition properties, the UPPAAL implementation offers usage of variables, conditions, synchronization channels, and other language construct to provide a certain level of expressivity (in theory summarized by term action). The used modeling

language is a subset of UPPAAL modeling language summarized in Tab 5.2. The subset is chosen concerning tested system class. Supported data types are *bool*, *int*, *clock* and synchronization type *chan*. Edges can contain *guard*, *sync*, and *update* expressions.

Besides the original implementation, Taster additionally utilizes labeling of timed automata states by relevance. *Relevance* is a natural number assigned to an automaton state. The parameter expresses the importance of a model state summarized by one number. The higher value implies, the higher priority. *Relevancies* are assigned by SUT expert knowledge. It is determined manually or automatically and originates in for example safety impact, the impact on rest of the system operability, and previously revealed issues (bugs). Kessy system testing example presented later shows *Relevancies* usage on environment models of start and door buttons. They are used to influence the frequency of short and long button press and button inactivity. Start button environment model is depicted in Figure 5.12. Relevancies are expressed as comments in the form $REL = X;$.

The success of proposed solution strongly depends on the reasonability of used model. In this work, two types of models are used. The first category is an environment model which simulation produces input for SUT. The second category is observer model which has Oracle function – they check expected SUT output by invariant conditions. The model division into environment and observer is only imaginary, and it is possible to combine input and output actions to single timed automaton. Nevertheless, partitioning of entire model to these two model types is recommended to keep clarity.

Data type	Action		
	Guard	Update	Sync
clock	●	reset	○
chan	○	○	●
bool	●	●	○
int	●	●	○

Table 5.2. Supported subset of UPPAAL modelling language

5.6 Test Generation Theory

A brief overview of test generation theory is presented in State of Art section 2.2.6. Now promising theoretical works will be analyzed deeper. To remind Timed Automaton can be viewed as an oriented graph. Guard condition and receiving synchronization channels might prevent taking an edge in certain time. Simulation of TA with the purpose of production of test stimuli can be considered as a special case of graph traversing as it knows from graph theory. Due to the possibility to restrict usage of an edge at the time, well-known graph algorithms can not be used directly. However, they can create a good starting point for the design of modified version which can handle TA specifics.

5.6.1 Graph terminology

Predominantly terms and names in graph theory are the same for undirected and directed graphs. Usually, the designation for a directed graph is expressed by preceding

word directed. In the following survey, all described terms and algorithms are for directed graphs if not stated otherwise.

Eulerian Cycle

Tour over a graph visiting all edges exactly once. Tour starts and ends in the same node. Edge coverage criterion is equivalent to the finding of Eulerian Cycle. This tour exists iff a graph (automaton) has nodes with the same number of input and output edges.

Eulerian graph

The graph in which Eulerian Cycle exist. It was named after Leonhard Euler which solved Konigsberg Bridge Problem.

Eulerian Path

The path over a graph where start node is not equal to end node. Again all edges in the graph are visited, and every edge is visited exactly ones.

Hamiltonian Cycle

It is the cycle over a graph where every node is visited exactly once. Similar to the situation with Eulerian cycle and edge coverage criterion Hamiltonian Cycle can be considered as a solution of node coverage criterion. The first and the last node is the same.

Traveling Salesman Problem

Finding of an optimal (in the distance) Hamiltonian Cycle is called Traveling Salesman Problem.

Hamiltonian graph

A graph which contains Hamiltonian Cycle. The problem of finding the cycle in dodecahedron is called Hamiltonian Game or Icosian Game.

Hamiltonian Path

Path or in another word tour visiting every graph nodes exactly ones. The starting point is not equal to finish point.

Chinese Postman Tour

Not all graphs are Eulerian. In non-eulerian case, a tour visiting all edges of a graph is called Chinese Postman Tour. Obviously, some edges have to be traversed multiple times.

Optimal Chinese Postman Tour

Chinese Postman Tour with minimal cost. It means that the tour has the minimum possible sum of weights. For unweighted graph is problem reduced to the sum of edges contained in a path. Notation of the tour by word optimal is often omitted, and Optimal Chinese Postman Tour is referred simply as Chinese Postman Tour.

5.6.2 Selected Graph Algorithms

Solving of the Chinese Postman as well as Traveling salesman problem is not a trivial task. Some remark on this topic follows. The text is not intended as the comprehensive overview of these problems. The purpose is to give a basic indication of this extensive area.

Chinese Postman Problem (CPP)

Let's start with computationally less intensive problem according to complexity classes [60]. The CPP problem is solvable in polynomial time. It is true for directed and undirected CPP variant. A mixture of oriented and bidirectional edges is considered as NP-hard and is called Rural Postman Problem.

The idea of CPP algorithmic solution follows. Some edges are added to a strongly connected graph until it is not Eulerian. Afterward, a Eulerian cycle can be constructed using i.e. Hierholzer's algorithm [61] which works in linear time.

The graph completion is inserting additional edges between unbalanced vertexes (nodes). Because result should be optimal, firstly is necessary to find shortest paths between all pairs of nodes. Floyd-Warshall [62] algorithm can be employed for that part.

With the knowledge of shortest paths nodes are divided into two sets. The first group contains nodes with the more incoming edges than outgoing. In the second group are oppositely nodes with more outgoing than incoming arcs. Both sets are interconnected with preserving usage of the shortest possible cost of additional edges.

Although CPP is quite familiar, implementation with code fragment is rarely described in graph theory books. Paper [63] tries to fix it and contains detailed implementation. Also, a variant for finding an open path across all edges not ending in the same node is presented. Open variant perfectly fits e.g. for testing when return to start point is not required.

Traveling Salesman Problem (TSP)

A solution of TSP is considered NP-hard. Which is mean, no polynomial time algorithm is known. The original problem statement is to found a shortest possible route between all given cities (nodes). Usually, a graph is complete and undirected. For the purpose of this thesis better suitable directed variant is called Asymmetric TSP.

Intuitive solution – Nearest Neighbour algorithm works as the name suggests. Some node is selected as start city. In the second step the nearest city - lowest cost edge is picked. Choosing of the nearest city ends when a tour is closed. Unfortunately, this approach not guaranteed an optimal solution.

From the long list of reputable algorithms, the optimal solution can be calculated by exact algorithms, but time requirement is high and becomes impractical for large problem instances let's say hundred of nodes. Using some tricks and parallel computing it is possible to achieve outstanding results, but invested work and costs are also corresponding.

Another big class of TSP solvers is heuristic algorithms. The output is not optimal but is very close to it. For practical usage in Testing, it could be the right way. Deviation from the best possible tour is in the range 2 - 3 %.

In connection with TSP, it is possible to mention Czech mathematician V. Chvátal, who deals with this problem and achieve some remarkable results [64]. Luckily Eulerian Cycle or Path, which is easier to found is probably sufficient for most of the testing challenges.

5.6.3 Discussion of Timed tours

This section intends to outline problems linked with the construction of desired tours or paths in given TA (graph). Traversing of TA is constrained by guards, synchronization, and it is distinguished by clock valuation. The first two aspects restrict edges, which are possible picked up at the time. So TA traversing gives different results based on variable variation and clock valuation. In other words, the challenge which is hard for an ordinary graph is yet complicated by additional restriction came from TA semantics.

Mentioned issues are now discussed in detail. The first case is synchronization. To remind in this work only basic synchronization channels - a pair of transmitting edge with postfix ! and receiving edge with postfix ? is reflected. In the case of construction of Eulerian tour sequence of edges have to respect synchronization. E.g. if a graph contains edge with exclamation mark corresponding edge with questioner have to be taken.

The second case is guards. Guard is a condition which fulfillment enables the edge to be taken. Conditions are defined over boolean, integer, and clock data types used in formulas with the common logic conjunctions. Relation of traces with clock valuation is discussed in special section. Boolean and integer variables can be modified by an update expression connected with an edge and by interaction with SUT by test adapter. Test adapter reads input variables before a test step and writes output variables afterward. Similar to the previous case with synchronization pair of edges the challenge is how to ensure that edge is pickable in desired time during a graph traversing.

The third case deals with the key element of TA – clocks. A clock variable can be used in guard condition, in invariant condition and the variable can be reset. Intuitive view says two traces are same when clock valuation in a particular state is the same (it is not so straightforward, comparison of clocks can be found, i.e. in [58]). Two problems can be identified. The first one is how to found clock valuation valid for a Eulerian tour. The second one is how many different time traces are necessary to cover all remarkable clock intervals. Covering of all values is not possibles as clock are real numbers. Distribution clocks values into intervals are solved by two main theories - theory of clock regions and theory of clock zones. An introduction to this topic with further reading references can be found in [58].

5.7 Algorithms

Suggested test generation is based on the model exploration using graph search techniques. Timed automata model is simulated in Real-time. SUT inputs and outputs are linked to model variables. Variable assignment on model edges produces test stimuli. SUT outputs are observed using variables utilized in location invariant conditions. Time is simulated discretely with an arbitrary step. Used algorithms are described in pseudo code. The first algorithm 1 describes the overall testing process.

Algorithm 1

```

DoTesting (Model):
  while invariantsSatisfied and coverageCriterionNotSatisfied
    ReadInputs
    DoStep (random, prioritized random, systematic)
    UpdateClocks
    WriteOutputs

```

Three different strategies are used to choose next step (edge). First step common for all strategies is the creation of a list of allowed edges in that time. Allowed edge is an edge with satisfied guard condition. If the edge contains synchronization, corresponding receiving edge also has to be allowed. In case the list of allowed edges is empty, no edge is taken, clocks are incremented, and the loop continues to next iteration. The loop is stopped in case the invariant is violated, coverage criterion is satisfied, or test is stopped by the test operator. Algorithms 2 and 3 take next edge randomly from the list of allowed edges. Algorithm 2 works with discrete uniform distribution – probability of pickup is the same for all edges in the list.

Algorithm 2

```

DoStepRandom (Model):
  for each ActiveNode in Model Templates:
    for each OutEdge in OutEdges:
      if GuardIsSatisfied(OutEdge)
        listOfAllowedEdges.Add(OutEdge)
    nextEdge = Random(listOfAllowedEdges)

```

Algorithm 3 modifies discrete uniform distribution using *relevance* numbers defined in the previous section. Probability of taking for an edge i from the list of allowed edges is:

$$P(E_i) = \frac{rel_{E_i}}{\sum rel_E} \quad (5.1)$$

Implicit *relevance* is equal to one. *Relevance* is assigned to an edge from its target node. It may be confusing, but reason is to preserve compatibility with UPPAAL model format. *Relevance* is stored as a node comment, which is not possible with an edge.

Algorithm 3

```

DoStepPrioritizedRandom (Model):
  for each ActiveNode in Model Templates:
    for each OutEdge in OutEdges:
      if GuardIsSatisfied(OutEdge)
        listOfAllowedEdges.Add(OutEdge)
    nextEdge = PrizedRandom(listOfAllowedEdges)

```

Algorithm 4 favorites edge with the lowest takes count from a list of allowed edges. Model exploration is more uniform than for random strategy. This behavior may speed up structural model coverage if it is desired.

Algorithm 4

```

DoStepSystematic (Model):
for each ActiveNode in Model Templates:
  for each OutEdge in OutEdges:
    if GuardIsSatisfied(OutEdge)
      listOfAllowedEdges.Add(OutEdge)
    nextEdge = PickLowestTakenEdge(listOfAllowedEdges)

```

Concerning black box testing approach only model coverage criteria are feasible. Condition `coverageCriterionNotSatisfied` unify coverage of all timed automata model nodes or edges.

5.8 Evaluation Metrics

To express some quantitative measure of produced test cases is important to choose some coverage criteria suitable to TA models. Despite SUT code coverage is useful complementary metric, it is not used in the thesis. The code and appropriate tools for this kind of coverage calculation are not commonly available in targeted area. For this reason, code coverage criteria are not considered. To unify terminology used terms coverage criteria, test selection criteria, as well as metrics, have the similar meaning. They expressed the same thing in different context. Metrics and coverage criteria rates result of a test generation. Test selection criterion is objective for this test generation expressed in same quantities.

If we leave SUT code coverage measures, according to workflow proposed in 5.3, a metric can be referenced to specification and model. A specification is a textual document. Despite the fact that it is structured, it is not directly computer processable. Still, it is possible to make an expert analysis. Automation of the process is the separate problem. Usually, requirements or specification are formalized, and some requirements \leftrightarrow model relation is established. Once a model is traceable to a requirement, it is possible to evaluate coverage of individual statements in a document. A specification based metric is proposed in [65].

Intuitive and natural measure a state based model is coverage of its states (nodes) and transitions (edges). This criterion directly expressed how many nodes or edges are visited from total count during a test run. Naturally, if all edges were visited full node coverage is also fulfilled. Full edge coverage of a TA network also guaranteed all guards were satisfied and taken because conditions (guards) are coupled with edges. On the other hand, it not ensures, that case with not fulfilled guard was tested.

Data coverage is another important family of coverage criteria. In our case, we work with boolean and integer variables. For logical variables is, of course, reasonable to examine both states (true and false). For one variable it is not an issue. With increasing number of booleans number of combination grows. Second used variable type are integers up to 64-bit range in Taster and 16-bit range in UPPAAL. Signed in both instances.

Even for one 16 bit integer is covering all possible values practically impossible – one test step takes with SUT in our case is in the range of seconds. The combination of multiple integers makes state space unexplorable. It is referred as state explosion

problem [66]. In many cases, only several values are used from the variable range. E.g. an enumeration of some switch states. Explicit bounding of variable values as it is possible in UPPAAL 4 model format is beneficial. A common solution is partitioning of (bounded) integer in domains (significant intervals). Classification tree method in embedded system variant can be mentioned as a representative [67]. Coverage is consequently expressed as coverage of individual domains.

The most complex problem is connected with time. As it was discussed in 5.6.3 two TA tour are not same if clock valuation is different. The clock is float (theoretically real number) downward closed variable. The design of intelligent timed traces generation, as well as its evaluation, is out of the scope of this thesis. It creates a possibility for future research and will be cover in the corresponding section.

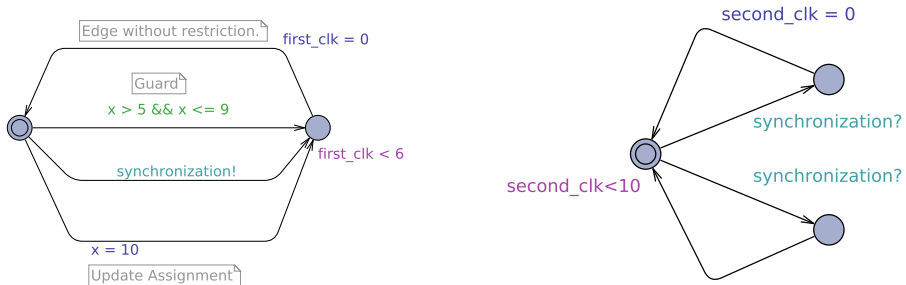


Figure 5.6. TA coverage enumeration

To show possibilities of coverage calculation on Timed Automata situation is depicted in Figure 5.6. Also, criteria applicable in the context of the presented work are highlighted in Table 5.3. To recapitulate previous paragraphs produces test traces can be evaluated based on following constructs. They are *Nodes*, *edges*, *guards*, *synchronization edges*, *update assignments* and *clock operations*. A comprehensive overview of evaluation criteria is stated in [68].

Criterion coverage	Family	Related to
All-requirements	Requirements-Based	Specification
Nodes	Structural	Model
Edges	Structural	Model
Booleans values	Data	Model
Integers values	Data	Model
Synchronization edges	Structural	Model
Guards	Decision	Model

Table 5.3. Metrics suitable to TA

Fault-based criteria are trying to show, that system not containing some predefined set of bugs. One way how to demonstrate the strength to reveal these faults original system is a mutation of original SUT behavior. Usually is defined a group of mutation operators, which are applied to an SUT code. An example of the operator can be the exchange of plus sign to minus sign or less or equal to less. Regrettably, modification of SUT code is not possible in our area of interest. On the other hand, we can mutate models. A nine member set of mutation operator proposed by [69] follow:

1. Change action - name of output in assignment is changed to another output.
2. Self loop - edge target location is replaced by source location.
3. Change target - target location is replaced by another one.
4. Change source - same as Change target with source location.
5. Change guard - quality/inequality sign in a guard is changed randomly to a different operator.
6. Negate guard - change of the guard by its negation.
7. Change invariant - change of invariant by adding one to its right side.
8. Sink location - replacing target location by new don't care state.
9. Invert reset - replacement of clock reset by a different clock operation.

Each of these mutation operators should mimic one common fault. Suitable application of model mutation is to validate model correctness. This fits intended application because straightforward fault injection is not possible.

5.9 Taster

The proposed concept with MBT theory and algorithms results in the implementation of a testing tool named Taster. The first version was implemented by master thesis [70] under the thesis author supervision. Taster works with models stored in UPPAAL 4 format. After a model is loaded, it is possible to performed testing as proposed. SUT is connected by a test adapter. In following sections, the Taster is described concerning testing workflow. First is the model parser and last is the result viewer. Taster provides those main features:

- TA model syntax check
- TA model viewer
- Online MBT testing driven by Env/SUT (Observer) model
- Multiple coverage criteria Node/Edge or infinite run for stress testing
- Model exploration algorithm selection from Random/Systematic/Experimental
- Complete trace with visualization for comfortable debugging
- Replay function from a test run
- Test run statistics

5.9.1 Taster Architecture

The software implementation is represented in Figure 5.7. It can be viewed as two separated parts. The first part is TA Model Analyzer, which converts a model into suitable data structures. This part can be substituted by another one for different model formalism (SysML, Petri nets). The second element – Run Time is testing engine itself. The component labels used in Figure 5.7 corresponds with programming classes.

5.9.2 Model Parser

TA model prepared in UPPAAL tool is input for the model parser. This model is pre-processed before Taster can perform a test run. First of all syntactical correctness is checked. It guarantees usage of only supported language constructs in a model. For this task syntactical analyzer code generator utility ANTLR [71] was employed. Then model templates are instantiated. Each automaton is represented as a list of nodes

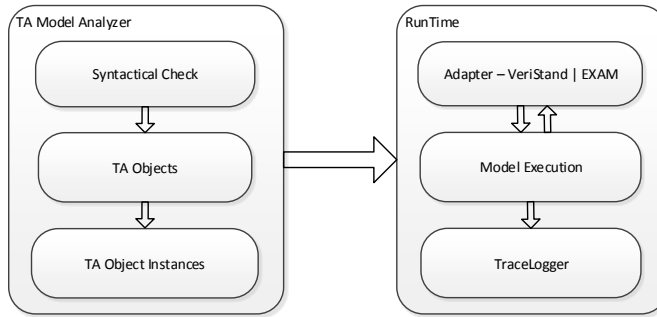


Figure 5.7. Taster architecture

connected with edges. Global and local variables lists are also created. Parser job is done when all objects are constructed and initialized.

■ 5.9.3 Model Execution

Model execution lays on continuous graph traversing using algorithm explained in 5.7. Simulation runs in steps. The time between steps is adjustable. In each step list of allowed edges is created and one of the allowed edges is chosen (depends on selected strategy). Allowed edge means that guard condition satisfied. In the case of edge with synchronization channel is picked, a corresponding receiving edge is also chosen. If no edge is allowed, simulation goes to next step (waits until progress is possible). Guard, invariants, and update expression are evaluated using Shunting-yard algorithm [72].

■ 5.9.4 Test Adapter

Taster offers two types of test adapter. The first uses direct interconnection to National Instruments hardware platform through NI VeriStand Real-Time testing software. The adaptor is based on NI provided .NET API [59]. Each step starts with reading of input variables using *ReadVariables* function.

```
void ReadVariables(Dictionary<string, Variable> variables)
{
    foreach (var v in _outputVariables)
    {
        _ws.GetSingleChannelValue(v.Value, out double val);
        variables[v.Key].Value = (int)val;
    }
}
```

Similarly, each step ends with writing output values using *WriteVariables* function.

```
void WriteVariables(Dictionary<string, Variable> variables)
{
    foreach (var v in _inputVariables)
        _ws.SetSingleChannelValue(v.Value, variables[v.Key].Value);
}
```

To help to the deployment of presented work into automotive industry practice, an EXAM adapter was created [73]. As EXAM does not provide suitable API, the architecture of this adapter is more complicated. Client (Taster) – server (EXAM) communication model is employed. The adapter is executed as a test step in EXAM test sequence and delegates Taster full control on SUT. Edges of Timed Automata model contain names of EXAM packages which are executed with edge pickup during model simulation. Technically Taster with EXAM speaks by the following set of predefined messages.

```
# Supported messages.
MSG_STOP = 'stop'
MSG_LOAD = 'load'
MSG_ACKNOWLEDGE = 'acknowledge'
MSG_EXECUTE = 'execute'
MSG_FAILURE = 'failure'
MSG_READ = 'read'
MSG_WRITE = 'write'
MSG_END = 'end'
```

5.9.5 Trace Logger

To make further analysis possible test steps are recorded. A captured trace is saved in an XML file. Trace viewer provides comfortable walkthrough over test run. Simulation progress is visualized by an active node change. Every saved step also contains variables snapshot. Additionally, is provided a replay function for reconstruction of interaction with SUT. Recording of a trace is implemented as separated class *Tracelogger* with characteristic operation *SaveStep* which is called after a step to store it. Class *Tracelogger* also provides statistical calculations over a trace.

5.9.6 Implementation

The Taster is developed using Microsoft .NET Framework. Programming language is C#. The technology was not chosen randomly; the reason is National Instruments product are mostly designed for Windows platform, and NI VeriStand is available for Windows platform only. Programming language was chosen with the knowledge that NI Veristand offers simple, but powerful API. The code is written in C# using .NET 4.5 and Microsoft Visual Studio 2013 IDE. The program is designed with Object-oriented programming (OOP) principles. The executable is ordinary Windows Forms application. It is runnable on common Windows 7 machine. Requirements for the machine are stated in Appendix C.

5.9.7 User Interface

User interaction with the Taster is implemented in the form of Graphical user interface (GUI). Window depicted in Figure 5.8 is the model viewer. This window is shown after the program launch. The user selects a model in open file dialog. In the case of syntactical error warning 5.9 is shown. Afterward, can browse over model templates and check initialization script. The main purpose of the script is an instantiation of model templates.

All template instances are visible on run screen which is displayed in Figure 5.10. A major part of the tool features as well as test execution is controlled from this window.

A syntactically correct model have to be opened before run screen launch. Instead of a description of individual buttons test control is presented as a sequence of steps.

First, required action is a connection to test adapter. As it was mentioned in the previous text, in the context of the thesis two types (EXAM and VeriStand) of test adapters are considered. Windows in Figure 5.10 shows EXAM variant. Well, by clicking Connect to EXAM a connection is established, and fundamental parameters are checked. Without the connection, test run execution is not possible.

With a connection to an SUT, a test run can be started by Run button. In this case, default test parameters are used. Otherwise, the user can select coverage criteria, step time and the test generation strategy. Maximal testing time can be limited by a timeout. A complete test run is stored and can be inspected in Trace Viewer. Trace viewer is commanded by Open Trace, Save Trace, and View Trace buttons. The screen is illustrated in 5.11. The viewer is linked with the automata in the run screen, and active states are in relation to highlighted trace line.

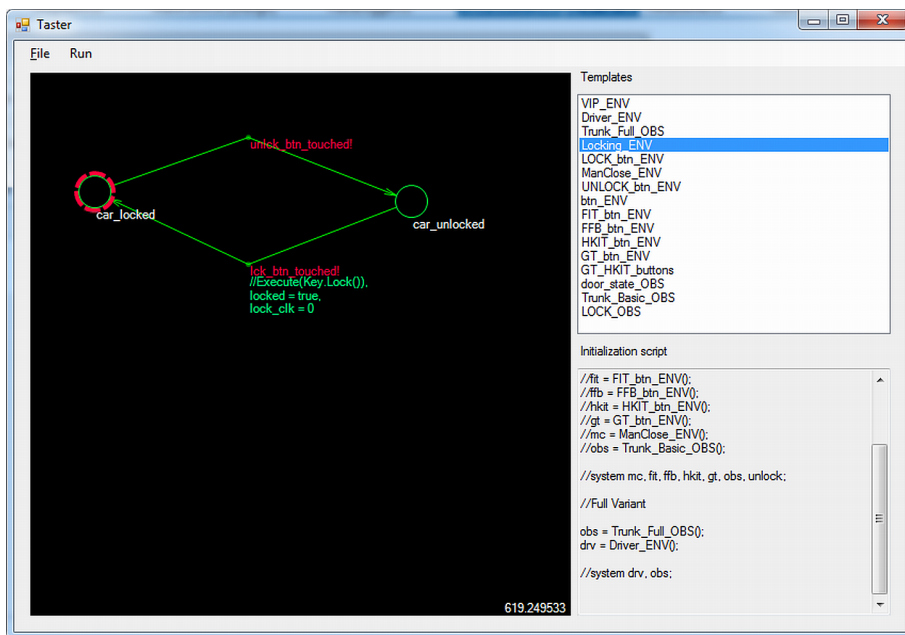


Figure 5.8. Taster Viewer

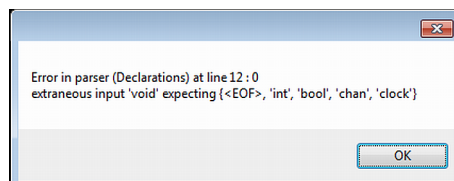


Figure 5.9. Syntax error popup window

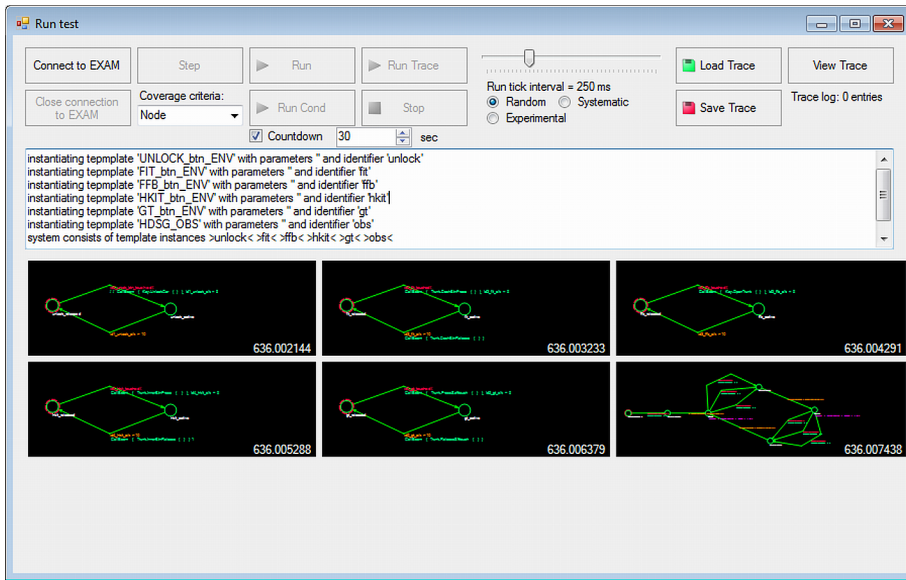


Figure 5.10. Taster Run Screen

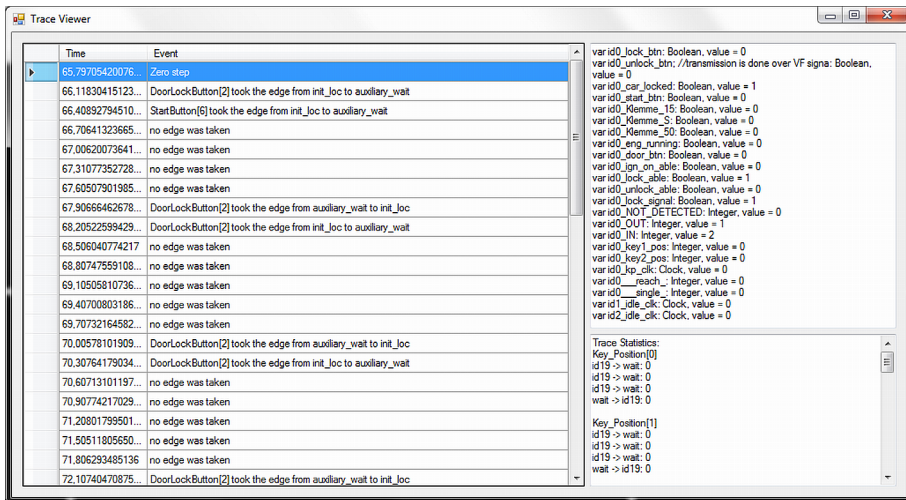


Figure 5.11. Trace Viewer

5.10 Case Study – KESSY

Keyless access system (KESSY) is a system allowing control a car without inserting a key to lock or ignition switch. The presence of a key is technically represented by a detection of key Radio-frequency identification (RFID) tag by the corresponding receiver. A model implementation for National Instruments PXIe platform was chosen for a tryout of proposed and implemented novel Integration testing concept. Firstly is stated SUT specification. Based on the specification SUT model and its implementation were developed. The purpose of the study is a preliminary evaluation of the suitability of the testing concept.

■ 5.10.1 Specification

The example system function (textual specification) is captured by this description. Door locking system is controlled by lock button built in the driver side door handle. Start/stop button works as the ignition switch. The short press is designated to turn the ignition on, and the button long press is the command for engine start operation. Button press longer than one second is considered long. Ability to lock and unlock the door, as well as start or stop the engine, is determined by detected key position. The system contains two equal keys RFID tags marked Key 1 and Key 2. The system recognizes following states of each key: detected outside the car, detected inside the car and not detected. The door unlocking is not possible if no key is detected outside the car. The door locking is not allowed if a key is detected inside the car. The engine start requires a key to be detected inside a car.

In modern vehicles KESSY, the system is implemented as a distributed system. Let's consider a system composed from three ECUs. First ECU is the Kessy system itself which cooperates with a Body Control Module (BCM) and Engine Control Unit. All ECUs are interconnected by CAN bus [74]. Kessy ECU is responsible for keys and button status monitoring. Based on its command BCM controls power supply system and individual door locks. Engine start and stop procedures are driven by Engine Control Unit. The system inputs are described in the previous paragraph. The system is observable by four digital outputs. Door locking system has locked and unlocked states. Start button controls ignition system which controls three power supply branches. German language prefix Klemme for individual clamps is preserved as is standardized by DIN 72552 standard [75] and ISO equivalent does not exist. The system controls Klemmen 15, 50 and S. Klemme 15 is active if the ignition is on. Klemme 50 is active during engine startup only, and in this work, it is used for engine start monitoring. Klemme S is turned on by switching ignition on, and it is active until the car is locked. Common usage is for audio system powering.

■ 5.10.2 Models

The KESSY system is modeled by the network of nine timed automata. Every system input is modeled by a single automaton. Kessy system provides two functions – ignition switch controlled by start button and door locking system control. Correct behavior of each is observed by the separate observer automaton. The relation between keys position and corresponding system behavior is modeled by additional three automatons.

The first pair of models is depicted in Figures 5.12 and 5.13. Door lock button has two states only (pressed and released). Furthermore, in case of the start button, the long and short press are distinguished. Models also contain auxiliary wait states which are used for simulation of user inactivity. Labeling automaton states by *Relevance* parameter are used for preferring inactivity (wait) to button press and short to the long press.

Model in Figures 5.14 describes three possible keys states. These states are “key is not detected,” the “key is detected outside the car,” and “key is detected inside the car.”

The most important part, which distinguishes between expected and incorrect SUT reactions are observer models. Our example uses two observers. Locking System Observer is depicted in Figure 5.15. Ignition System Observer is shown in the Figure 5.16. System behavior is checked by location invariants. Variables in invariant conditions are

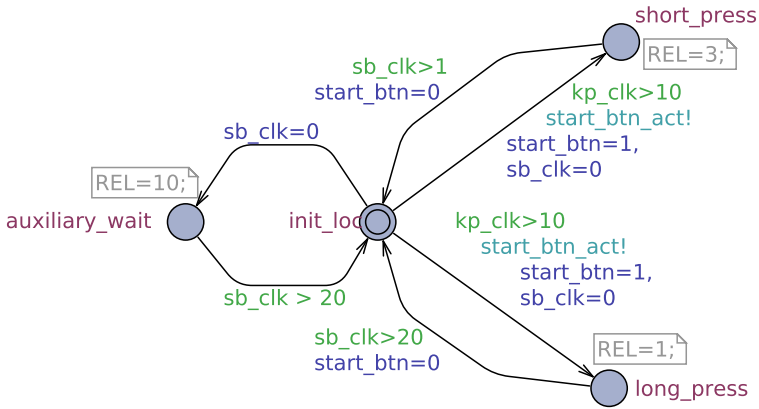


Figure 5.12. Start button model

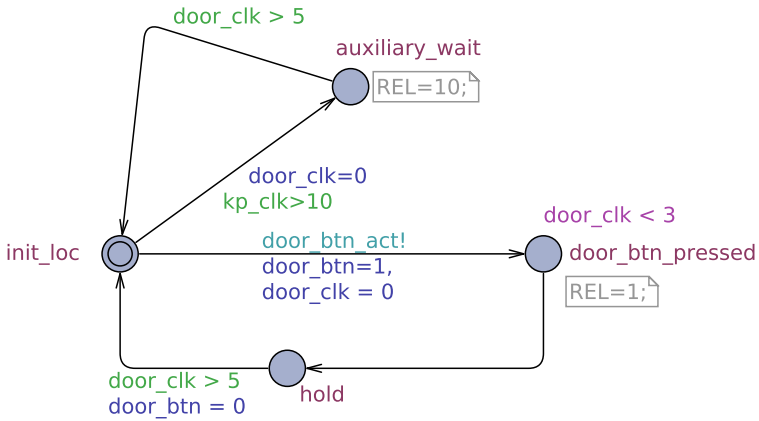


Figure 5.13. Lock button model

Entity	Count
Templates	8
Instances	9
Nodes	30
Edges	39
Clocks	7
Variables	15
In/Out variables	6

Table 5.4. Kessy model summary

mapped to system outputs. Observers are synchronized with button handling models by synchronization channels.

Simple models in Figures 5.17, 5.18, and 5.19 capture relation between keys position and allowed system behavior. Key location determines whether it is possible to unlock, lock, and start the car. The models produce corresponding signals to handle this situation.

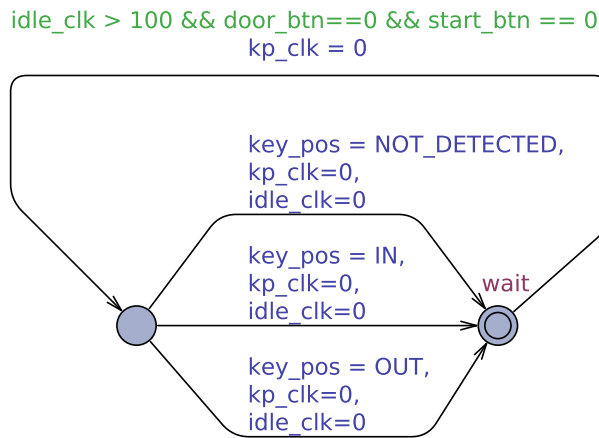


Figure 5.14. Key position model

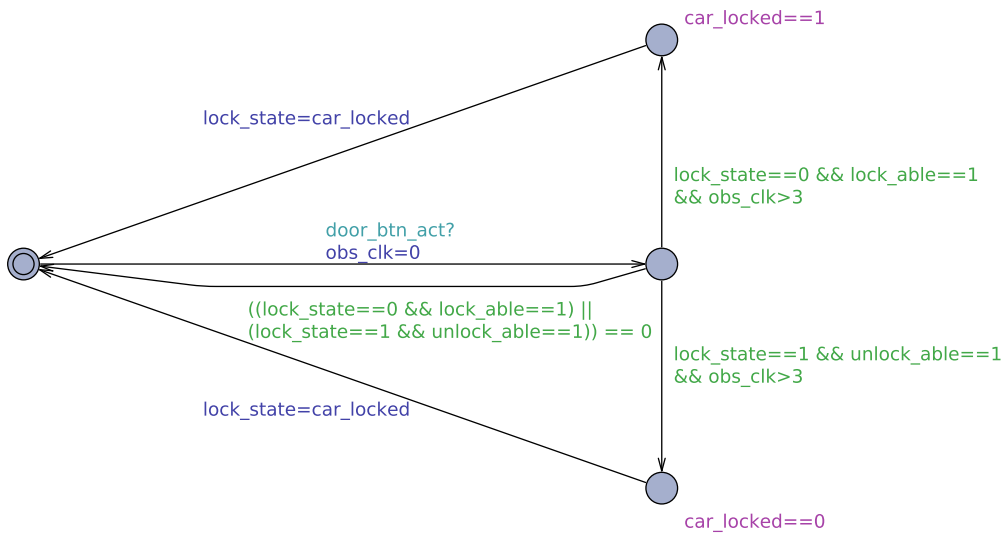


Figure 5.15. Observer model – Lock System

5.10.3 SUT Implementation

The experimental KESSY system was developed in NI VeriStand Real-Time Testing Software. The platform was chosen on the basis of previous experience during development HIL test place with our industrial partner. Real Kessy ECU was replaced by a fully programmable dedicated SUT, as fault injection is much less complicated. The KESSY system described by specification in Section 5.10.1 is implemented as three simulation models executed by NI VeriStand Real-Time engine. SUT partitioning to multiple models better reflects distributed nature of a real automotive system. The interconnection between models is realized by VeriStand channels. The connection of Taster tool to SUT is made by test adapter which uses VeriStand .NET API [59]. Model example implemented in LabView is depicted in Figure 5.20.

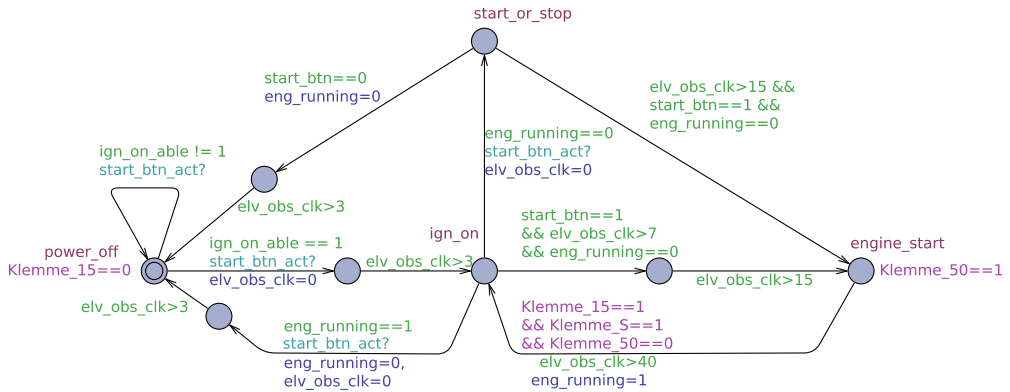


Figure 5.16. Observer model – Ignition System

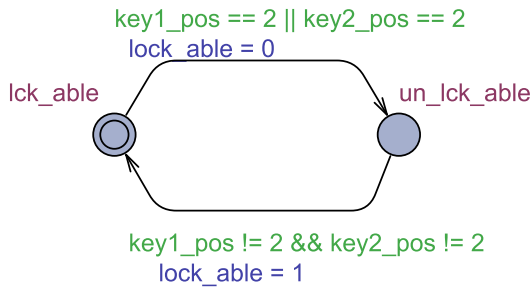


Figure 5.17. Lock able signal

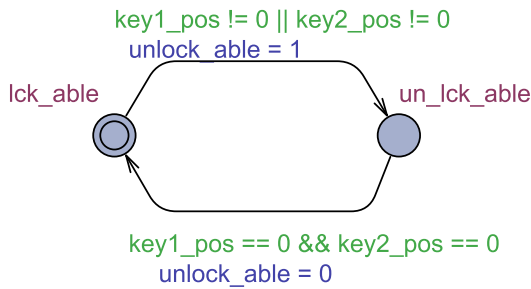


Figure 5.18. Unlock able signal

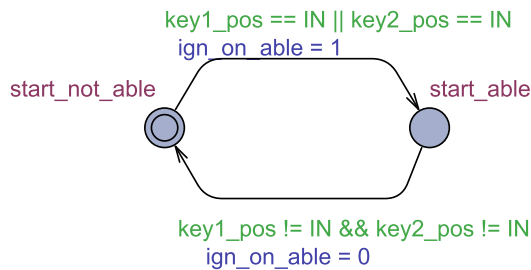


Figure 5.19. Start able signal

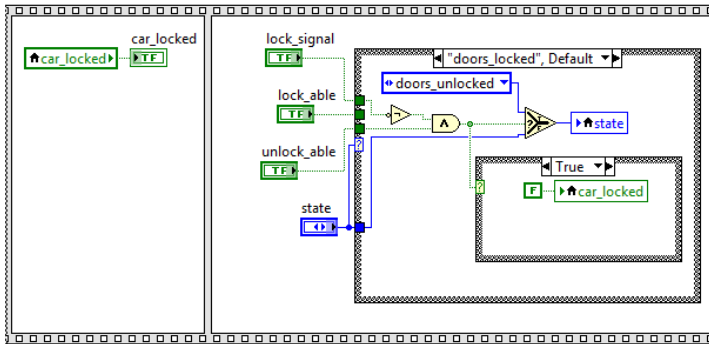


Figure 5.20. A part of SUT implementation model

5.10.4 Results

Created KESSY system specification was implemented in LabView as VeriStand simulation models and modeled in UPPAAL in Taster supported language constructs. The implementation was afterward tested by the Taster. Test step period was 100 ms and frequency of Primary Control Loop of NI VeriStand was 50 Hz. The objective of the first set of tests was an error detection capability, and it was evaluated by injection of three independent faults into SUT.

Fault 1: Klemme S goes off after two seconds from turning on the ignition.

Fault 2: Short start button press is not recognized while the ignition is on – it is not possible to shut down the ignition.

Fault 3: Car is not able to lock if both keys are detected outside the car.

Faults were injected by modification of implementation models. All inserted faults are successfully detected. Detailed results are summarized in Table 5.5. Detection time is lengthened by key position templates; they wait for arbitrary time quanta between key position changes. Fault No. 1 and 2 needs at least one key inside the car to allow switching on the ignition. Otherwise, the fault is not detectable. Fault No. 3 is exposed if Key one and two detected outside the car. Optimization for time or step count is possible in future work and was not objective of presented work. Algorithm 2 – the Random strategy was used for fault detection experiment.

Fault	Detected	Time to detect	Steps	State of detection
F1	•	39 s	392	ign_on
F2	•	28 s	276	power_off
F3	•	99 s	992	check_locked

Table 5.5. Kessy – fault injection results

Correct implementation was evaluated by performing nine test runs summarized in Table 5.6. No fault was revealed. Node coverage of complete model was selected as the stop condition. The SUT model was not optimized for the fastest possible node coverage.

Trace	Strategy	Test Time [s]	Steps	Lock state change	Engine Start–Stop
TR 1	Random	45	446	2	2
TR 2	Random	46	458	10	1
TR 3	Random	195	1950	57	2
TR 4	Relevance R.	152	1525	43	1
TR 5	Relevance R.	36	363	1	1
TR 6	Relevance R.	84	841	25	2
TR 7	Systematic	300*	3005	78	9
TR 8	Systematic	300*	3002	78	9
TR 9	Systematic	300*	3004	79	9

Table 5.6. Kessy – overview of performed test runs

*test run was terminated after 300s because it is not possible to reach node coverage

The testing ability for the specific SUT is expressed by locking state change and engine start and stop included in a test trace. If these two features are examined, the major part of SUT functionality is tested, because they are conditioned by the correct reaction to key position and door and start buttons. Test runs that use Algorithm 4 (Systematic) discover impossibility to achieve node coverage. The reason is in demand of generation of two short start button presses in a row, which is not possible with systematic exploration. Without two consequent short presses, it is not possible to test switching ignition on and off without engine start.

5.11 Case Study – Trunk

Opportunity to examine the proposed concept and testing methods come from our industry cooperation. The object of testing was Trunk opening system controlled by dedicated ECU. It is the system, which is responsible for control of opening and locking system of vehicle trunk door. The system has the manual and automatic variant. Manual variant does not have actuators, and it operated by a human. An automatic variant can operate door on user request by a button or foot gesture. The SUT was chosen concerning reasonable complexity. System specification with its inputs and outputs is captured below.

5.11.1 Specification

Trunk opening system is implemented as dedicated ECU which controls trunk door opening and closing. The ECU cooperates with rest of distributed system by CAN bus. Actuators are directly connected. The buttons located on the door are also wired, and remaining buttons are distributed over the communication bus. The virtual pedal function is processed straightforwardly by Trunk ECU. System behavior is handled by user commands. The operation of the system is constrained by locking system state and vehicle speed.

The automatic variant can be understood as complete system description. Then, the manual variant is its subset. System input are summarized in Table 5.7. Trunk doors are commanded by four buttons. Two of them are placed on trunk doors. One is

Description	Location	States
Open button	Outside trunk door	pressed / released
Close button	Inside trunk door	pressed / released
Open button	Center console	pressed / released
Open button	Car Key	pressed / released
Virtual Pedal	Under bumper	gesture recognized / no
Close by pushing door	Door drive	yes / no
Open position limit	Infotainment	0 – 100 %

Table 5.7. Trunk ECU Inputs

outside and second is inside. Another option is open or close the trunk by the button on car key or from Center console.

The trunk open is launched by one of these buttons. The button on the outer side of the door, the button on the center console, or button on the car key. The action is not launched if the vehicle is locked. The function of outer trunk button is deactivated during a ride.

The closing of the trunk door can be triggered by same buttons as opening except for the button on the center console. The movement can be activated as well by pushing door downward. The closing is stopped if a collision occurs. Either opening process can be stopped by pressing any buttons. In that case, the warning sound is heard. The sound signal is also accompanying opening trunk by center console button. Opening/closing by car key button is also accompanied by the sound signal.

In case of a low ceiling or to help manipulation with trunk smaller persons maximal open position can be set. The position is stored by following step sequence. Users stop door is desired position and hold inner door button until the sound signal is heard. The limit is reset if the door is fully opened by hand and same as in previous case position is stored by the inner button press.

Observable outputs are stated in Table 5.8. In the automatic variant the Actuator is one or two motors – depend on actual configuration. From the Test view, count of motors does not affect system behavior.

Description	States
Position Binary	open / closed
Position in Percent	0 – 100 %
Trunk Lock	locked / unlocked
Warning sound	yes / no

Table 5.8. Trunk ECU Outputs

For the testing purpose, the manual variant is considered as the automatic (full) variant subset. Instead of a linear motor drive, the trunk is opened by releasing of spring. Spring is not able to fully open the door; it opens doors only partially. Closing is purely hand-operated. Instead of the set of four buttons, the trunk is opened exclusively by exterior door button. Car key button is dedicated to trunk unlock. Inner door button, as well as center console button, are not present in this variant. Dependencies with car speed and locking system state are the same.

■ 5.11.2 Experiment Plan

The objective of the experiment is to examine practicability of designed and implemented testing concept on a physical automotive system. The system will be modeled as TA network; this should show the suitability of selected modeling formalism. Taster will be installed into control PC of an HIL testbed. Performing test runs on a targeted platform should indicate the applicability of proposed solution and shows direction for the future work. Initial conditions for the case study realization are the following:

- Functional specification of SUT (textual PDF)
- Original test suite specification
- Trunk opening system with HIL facilities
- EXAM < - > Taster adapter for communication with HIL testbed

At first, it is necessary to develop a model of SUT. Modeling of Trunk actuating system is covered in special section 5.11.3. With the model, it is possible to perform testing using Taster tool. Test adapter needs to define a dictionary of function names used in EXAM and names used in the model. Probably the biggest challenge is the design of quantification of the two different approaches with hardly comparable results.

Research questions can be divided into several categories. The first questions group is focused on modeling SUT by Timed Automata. The intention is to examine if TA is suitable for the area of interest (partially discussed in 5.1). How difficult is model development? Alternatively, is chosen TA subset reasonable or should be extended in future work?

The second group covers chosen online test generation strategies based on graph search techniques. Random, Systematic, and Experimental strategies will be compared according to achieved model coverage on the Trunk System model.

The third group judged implemented adapter according to the provided list of operation. Capabilities of the adapter are summarized in 5.9.4. The question is: Are provided functions comprehensive or should be supplemented?

Finally, the case study should answer overall suitability of proposed concept. Identify its strength and weaknesses, suggest improvements, and indicate future research directions. Researched questions as a listing:

1. Model related questions.
 - a) Which behavior is difficult to model?
 - b) Which behavior is easy to model?
 - c) How labor intensive is model development?
 - d) What are the demands for an expert?
2. Test generation related questions.
 - a) Is random test generation strategy applicable?
 - b) Is systematic test generation strategy applicable?
 - c) Is 'experimental' test generation strategy applicable?
3. Test adapter (EXAM variant) related questions.
 - a) Are the method calls input parameters helpful?
 - b) Are the outputs processing by a method return value sufficient?
 - c) Is the implemented list of operation convenient?
4. Overall concept suitability questions.
 - a) Is designed MBT testing solution suitable to target area?

- b) What are the advantages of the selected solution?
- c) What are the disadvantages of the solution?
- d) Is there any future research directions?

5.11.3 Models

The SUT inputs – buttons and leg gesture are modeled as simple two state automata. The button model captures situation of an input activation, e.g. button press. Delay before return to the initial state is arbitrary and is expressed by a constant. A transition is equipped with the corresponding function which is called by EXAM. These calls provide interaction with a tested system. Also, the variant for an input activation and deactivation was created. It is intended for modeling different button operation like a short and long press. Due to long (hundred of milliseconds) and unpredictable delay in HIL system were not used (probably caused by the low-level implementation).

The first depicted automaton simulates SoftTouch button located outside trunk door. It is depicted in Figure 5.21. The button triggers trunk opening. The closing of the trunk by this button is also possible. Underlying EXAM operation is the parameter of Execute command.

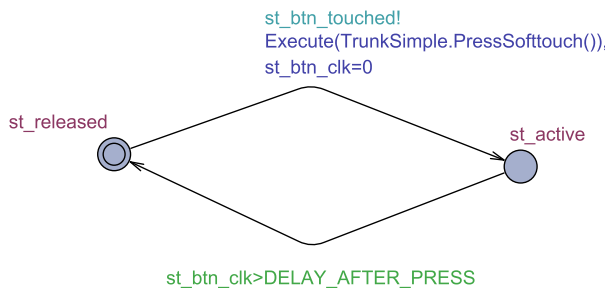


Figure 5.21. Input – SoftTouch button

Complementary to the opening button, the Trunk is closed by the button located on inner side of the door. Model is in Figure 5.22. Delay after button press is defined by constant *DELAY_AFTER_PRESS*. The opening of the trunk by this control is not possible.

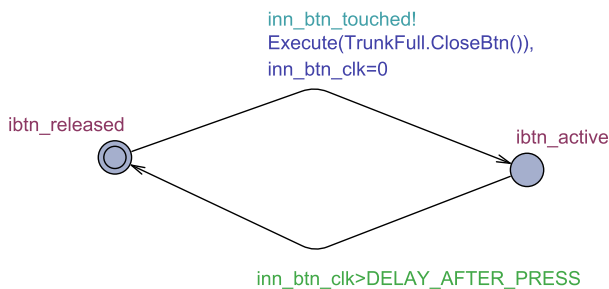


Figure 5.22. Input – Button on inner door's side

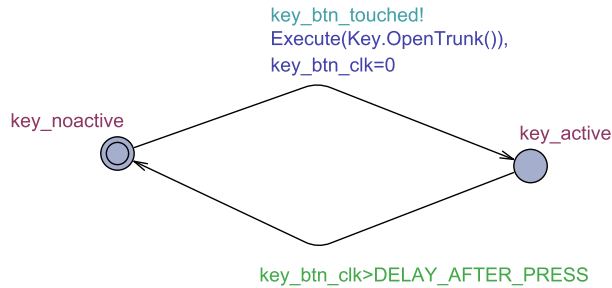


Figure 5.23. Input – Button on Key

Besides commonly located buttons suggesting classical trunk door operating by hand, Open and Close of the door is possible by the button on the car key. TA representing this control element is depicted in Figure 5.23. Due to HIL realization of Key buttons, only press operation is available. Constant `DELAY_AFTER_PRESS` prevents another activation for a defined time.

In some cases opening the trunk from driver's seat can be beneficial. E.g. quick pickup of a passenger on a busy road. For that reason, the system is equipped with the button on the center console. The only open function is linked with this button. TA is shown in 5.24.

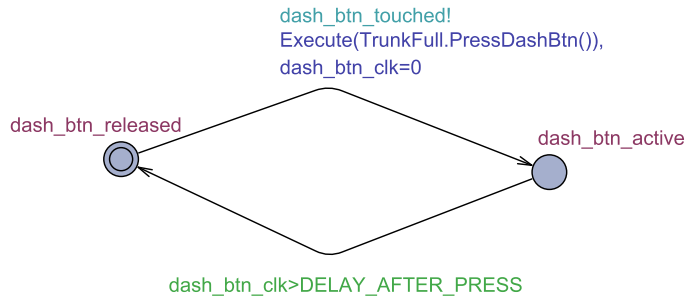


Figure 5.24. Input – Button on dash

Pushing of trunk door down by hand is expressed in Figure 5.25. This action is implemented using worm drive and a DC motor. Control of the testing equipment is hidden for the tester, and it is controlled by a test primitive (step) `CloseManual`. The solution is very suitable for proposed approach because uniform input model can be used for all system inputs.

The last automaton in Figure 5.26 is a little bit special. It is not a button; it is called Virtual Pedal. The operation is triggered by foot gesture. A user swings foot under the bumper. If the gesture is recognized, the system starts opening the door. The input is very different from a user point of view, but from the testing perspective, it can be modeled by similar automaton as an ordinary button.

Pressing of any of button during system operation will cause system stop. The exception is Virtual Pedal which has only trunk opening function. No additional gestures are recognized.

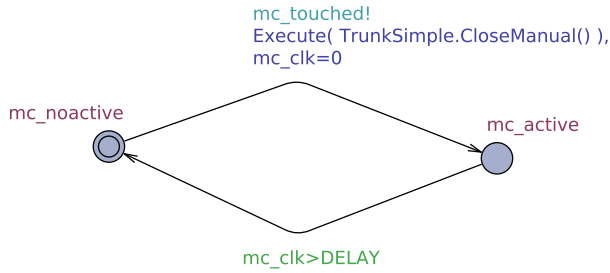


Figure 5.25. Input – Close by hand

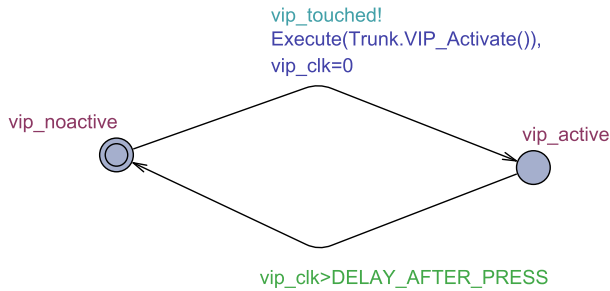


Figure 5.26. Input – Virtual Pedal

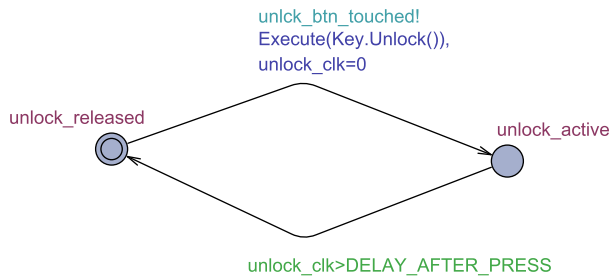


Figure 5.27. Input – Unlock button

The function of Trunk opening system is closely coupled with Locking system. Unlock button model 5.27 can be viewed as an auxiliary model. It makes the SUT operable by unlocking the car.

Input models presented in previous paragraphs can be considered as full permissible. Its simulation allows random activation of all input at almost any time. Pressing of a button is restricted by receiving synchronization edges in observer model. To remind synchronization edge with exclamation mark can be pickup only if a corresponding receiving edge is available.

To show a different option to input (environment) modeling another model was developed. It is depicted in Figure 5.28. In a nutshell, model controls the SUT in following way. The trunk is periodically open and closed by a permitted button. Door opening, as well as closing, can be stopped. Using *Relevancies* proposed in section 5.5 is cancel operation penalized against operation successful finish five times.

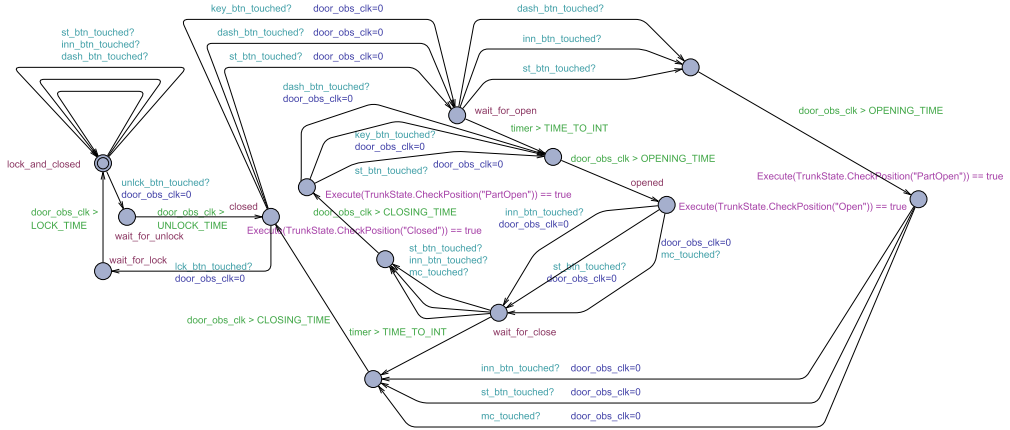


Figure 5.30. Trunk Observer – Full

Entity	Basic Variant	Full Variant
Templates	7	2
Instances	7	2
Nodes	18	22
Invariants	2	4
Edges	22	55
Clocks	7	2
Channels	6	7
Variables	0	1
EXAM commands	8	9

Table 5.9. Trunk model summary

5.11.4 Original Test Suite

Handwritten test sequences used for Trunk Opening System have the origin in a test specification. This document describes individual test cases in a structured form. Those test cases are consequently implemented in EXAM as a sequence diagram. The set creates test-suite. To make this section understandable an example of a test case specification is given.

Test Case Example

Description:

Evaluation of trunk opening no interruption by turning On KL 15 (ignition switched power supply).

Initial Conditions:

- All Doors closed.
- Car is unlocked by the button on key.
- KL S Off.

Actions:

- Action 1: Active trunk opening by center console button.

- Action 2: After trunk position reached 50%, turn KL 15 On.

Expected results:

- Result 1: Tailgate starts opening.
- Result 2: Tailgate continues opening.

The specification contains over one hundred test cases. These test cases are implemented for three different platform, which uses a variant of Trunk Opening System (ECU). Some available characterization of in EXAM developed test sequences is in Table 5.10. A Smaller number of test cases on platform number one is because only manual variant is present.

	Test cases count	Duration
Specification	106	–
Platform nr. 1	40	3:30h
Platform nr. 2	105	7h
Platform nr. 3	116	7h

Table 5.10. EXAM Test Suite

■ 5.11.5 Results

A case study on a real Trunk opening system was the last step of the thesis. It was intended to try out designed MBT solution in the target area – Automotive Integration testing. The experiments were performed in HIL test laboratory on our industrial partner side. Total time spent in the laboratory was five man-days. Overall characteristic of performed test runs is in Table 5.11. Test runs TR 6–TR 10 ends by invariant failure. The cause was in different delay of individual buttons calls (hundred of milliseconds). The delay caused that trunk move did not stop, but the model supposes stop, and a partially opened position was checked. The model constants were adjusted, and simulation step was decreased to 250 ms. Shorter step gives finer resolution, which allowed for precise time delays specification. Test runs TR 11–TR 18 are the final ones with fully debugged model and EXAM operation calls.

Progress of node and edge coverage in time for selected test runs is shown in Figures 5.31 and 5.32. The first graph demonstrates the evolution of node coverage. TR 4 uses the Basic model variant (without the possibility to stop Trunk operation). Observer utilized in this model is not capable of return to the initial state. For this reason, full node coverage was not achieved. The reason why not full node coverage was achieved in the case of TR 15, TR 16, and TR 18 is the minor bug in the model. In driver automaton in Figure 5.28 commands for Trunk Manual Close was switched with the command for open by the button on Car Key. It caused absence of the corresponding synchronization edges pairs, and a small part of the observer model was not traversable.

The second graph 5.32 shows edge coverage evolution. In the case of TR 4 full edge coverage was achieved according to expectation. The situation for TR 15, TR16, and TR 18 has the same cause as for node coverage. In the Driver automaton in Figure 5.28 command for Trunk Manual Close was switched with the command for open by the button on Car Key. It caused absence of the corresponding synchronization edges pairs, and a small part of the observer model was not traversable.

Trace	Strategy	Test Time [s]	Steps	Step Time [ms]	Model Variant
TR 1	Random	374	375	1000	Basic
TR 2	Systematic	408	408	1000	Basic
TR 3	Relevance R.	340	340	1000	Full
TR 4	Random	600	599	1000	Basic
TR 5	Random	340	341	1000	Full
TR 6	Random	189*	758	250	Full
TR 7	Random	132*	531	250	Full
TR 8	Random	62*	249	250	Full
TR 9	Random	369*	1479	250	Full
TR 10	Random	72*	289	250	Full
TR 11	Random	399	1595	250	Full
TR 12	Systematic	610	2442	250	Full
TR 13	Relevance R.	612	2446	250	Full
TR 14	Random	611	2443	250	Full
TR 15	Systematic	1800	7201	250	Full
TR 16	Relevance R.	1800	7200	250	Full
TR 17	Systematic	861	3446	250	Full
TR 18	Random	634	2535	250	Full

Table 5.11. Performed test runs

*invariant failed

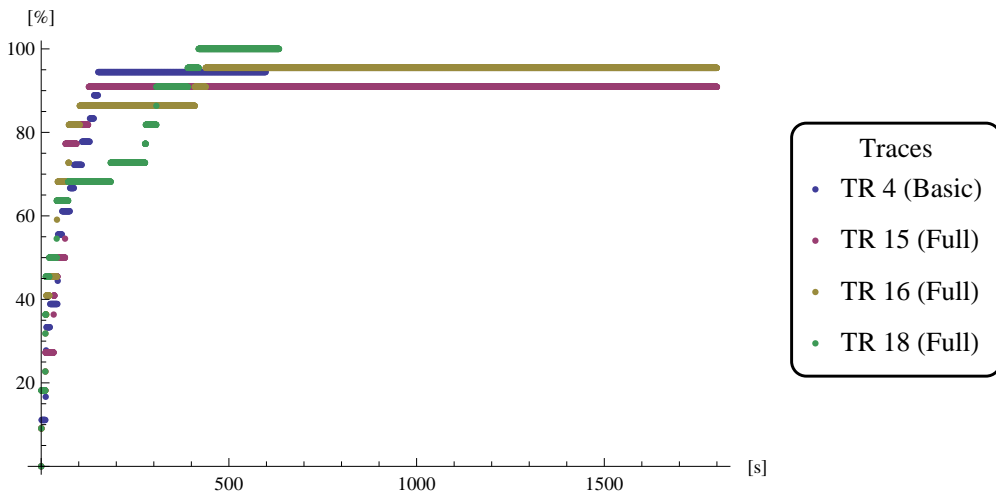


Figure 5.31. Trunk Node Coverage

Characterization of test runs according to performed interaction with SUT is summarized in Tables 5.12 and 5.12. Seven input actions and three output actions were monitored. Data are separated into two tables to achieve reasonable formatting.

5.11.6 Conclusion

Let's start case study conclusion with a discussion of questions stated in section 5.11.2. The discussion is followed by overall analyses of achieved results.

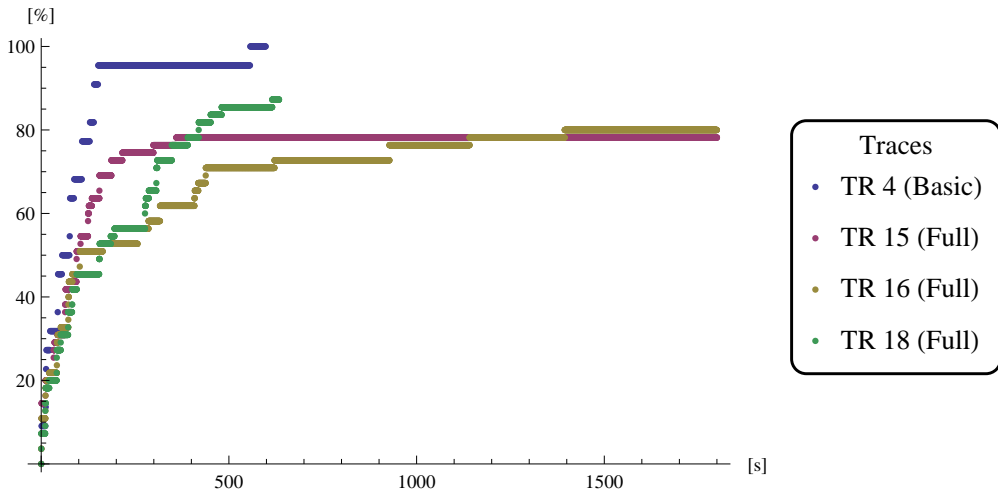


Figure 5.32. Trunk Edge Coverage

Trace	Unlock	Lock	SoftTouch button	Close button	Key Open button
TR 1	1	0	3	3	0
TR 2	1	0	2	2	3
TR 3	1	0	9	4	6
TR 4	1	0	4	4	2
TR 5	4	3	2	6	2
TR 6	5	4	6	6	0
TR 7	1	0	4	2	0
TR 8	1	0	1	2	0
TR 9	1	0	4	2	0
TR 10	1	0	5	1	0
TR 11	2	1	10	10	2
TR 12	4	3	15	10	3
TR 13	2	1	17	8	5
TR 14	5	4	9	12	3
TR 15	10	9	30	29	9
TR 16	1	0	26	18	12
TR 17	6	5	20	15	4
TR 18	5	4	10	13	3

Table 5.12. Test runs – SUT interaction 1

1. Model related questions.

a) Which behavior is difficult to model?

Proper expression of the relations between individual SUT functions modeled by separate automata. Synchronization channels provide powerful tool, but still is necessary to design pair of synchronization edges correctly.

b) Which behavior is easy to model?

All kinds of Trunk inputs were very easy to model.

Trace	Dash button	Manual close	Trunk opened	Trunk closed	Trunk stoped
TR 1	6	3	6	5	0
TR 2	3	2	6	6	0
TR 3	3	0	6	0	3
TR 4	5	4	9	9	0
TR 5	4	0	4	3	3
TR 6	7	0	0	1	2
TR 7	0	0	2	2	1
TR 8	0	0	2	2	1
TR 9	0	0	2	2	1
TR 10	0	0	2	1	0
TR 11	5	0	6	6	9
TR 12	7	0	5	4	9
TR 13	7	0	8	8	3
TR 14	8	0	5	4	8
TR 15	19	0	13	13	26
TR 16	15	0	25	24	10
TR 17	10	0	6	6	12
TR 18	8	0	6	6	12

Table 5.13. Test runs – SUT interaction 2

- c) How labor intensive is model development?
It is hard to quantify, but it seems very reasonable in comparison to manual test sequence implementation. Let's say Basic model variant is one man-day work. Understanding to the specification probably took longer.
- d) What are the demands for an expert?
A technician with knowledge of FSM should be able to create TA model after short training.
2. Test generation related questions.
- a) Is random test generation strategy applicable?
- b) Is systematic test generation strategy applicable?
- c) Is 'experimental' test generation strategy applicable?
All proposed algorithms are suitable for the Trunk opening system. Prioritization by relevancies is very suitable to navigate tool over a state space. In future prioritized systematic algorithm will be developed to achieve best possible coverage in short time and in long term will be able to prefer some model parts.
3. Test adapter (EXAM variant) related questions.
- a) Are the method calls input parameters helpful?
Yes, for example, Trunk position check methods requires parameter, which defines position to evaluate (Open, Close, Partially Open).
- b) Are the outputs processing by a method return value sufficient?
In the case of the Trunk opening system it is. The position check methods return unsigned integer value.
- c) Is the implemented list of operation convenient?
It is. Possibility to define EXAM method aliases, which are used in TA model is helpful. Also, TA models are understandable with rational command names.
4. Overall concept suitability questions.

- a) Is designed MBT testing solution suitable to target area?
With the experience given by the case study it is. Modeling by TA network is very natural and suitable. Software tool Taster fits well to HIL control PC environment. Communication with SUT by EXAM test adapter also works very well.
- b) What are the advantages of the selected solution?
For example with the reasonable amount of work it is possible to make a model for Random stress testing of SUT.
- c) What are the disadvantages of the solution?
If precise sequence of steps is required, the only way how to implemented it, is special model traversable exactly in one path.
- d) Is there any future research directions?
It is discussed in Future Work section.

The object of interest for the case study was the Trunk opening system. The main part of the system is dedicated ECU, which controls Trunk actuators and monitors door position. Buttons located on the door are also monitored by the ECU. The rest of inputs is distributed over CAN bus. The intention was to evaluate proposed and implemented MBT concept on real system of reasonable size. In the first phase, the system specification was analyzed. With the knowledge of system behavior, two models were designed. The first is labeled Basic and is capable of opening or closing trunk by every active button. The second is called Full, and beyond Basic model capabilities, is capable of testing sudden system stop during opening or closing. Each model variant demonstrates a different approach to input modeling. In basic models, inputs are modeled as the set of simple two state automata, and in Full variant is input model designed as a Driver model. System behavior is not covered fully. Only operations available on HIL system at the time of experiment were captured in models. The test are afterward driven by these models. Eighteen independent test runs are recorded. The longest test run took thirty minutes. All three implemented exploration algorithms were examined. Results can be summarized that proposed concept fits well to the targeted area. Amount of work required to model development is reasonable. The weak point of the study is the comparison with the traditional test suite. A technical solution capable of comparing both types of test cases was not available at the time of the experiment. This solution was also not developed due to limited time for this experiments.

5.12 Summary

Chapter Integration Testing proposes complete framework for automatic test case generation intended for Integration testing of automotive electronics. The work starts with vague objective stated as improvement of the known testing process by utilization of formal methods. Suitable methods were sourced in the vital scientific area of Model-Based Testing. From the long list of options, extended Timed Automata were selected for system modeling. Consequently, the workflow proposal with selected formalism was presented. The principle according to the MBT terminology is online test generation from TA network model. Theory of the test generation was overviewed. Based on the overview, the test generation algorithms were designed and described by pseudocode. The theoretical concept was implemented as the testing tool named Taster. The tool offers two independent test adapters providing interconnection with HIL testbed. The adapters are designed for connection to EXAM or NI VeriStand. The result – Taster

tool was evaluated by two case studies. The SUT was laboratory implementation of a KESSY system and real Trunk opening system in the HIL laboratory of industrial partner. Both studies indicate the suitability of proposed solution to the targeted area. During experiments, many topics for future work were identified. For instance, since the method should in the first step complement traditional approach, a metric for comparison both test suites should be developed to avoid overlapping of test cases.

Chapter 6


Future Work

Measurement methods presented in Chapter 4 are focused on the FlexRay node local parameters. Together with existing methods described in 2.1.2, they create the comprehensive set of measurement methods capable of identifying a FlexRay parameter set. Probably there is not much room for further research. The analysis of parameter set in 4.2, which supports this statement, is part of the thesis. However, there is an opportunity to implement all of these algorithms into the FlexRay bus intelligent interface. It could create an effective development tool. Another application of presented work could be online health monitoring of a FlexRay cluster or selected critical nodes. As FlexRay is being used as the communication infrastructure for safety-critical systems.

Results achieved in Chapter 5 create the platform for future research in the area of Integration testing. Opening research possibilities in this promising area was one of the objectives of this dissertation thesis. Prospective topics for future work are divided into several groups. System modeling is first of them. Car inner light, KESSY system, and Trunk opening system were modeled during the dissertation. Although no serious issue was found, system modeling still contains a room for improvement. For example, the committed and urgent locations known in the UPPAAL language can be implemented to the Taster. Support of reasonable subset of a procedural language like C could help to keep model clear. The complete electronics system of a car is quite complex, therefore some hierarchical or layered architecture of TA models could be beneficial to deal with this complexity. Modeling of an environment (inputs generation) requires some level of randomness. Enhancement of the model for Markov chain constructs could help to express a driver behavior.

Proposed algorithms can be viewed as basic versions, which can be enhanced in multiple ways. The systematic strategy does not consider receiving synchronization edges. It could be improved in future. Also, some advanced algorithm, which will consider next step concerning future progress can be beneficial. The concept of weighting nodes by *Relevancies* proposed in the thesis becomes powerful with automatic assignment of these numbers. One of the possible future directions is sourcing of this numbers from a database of previously identified bugs. Using this information, the state space will be partitioned into more and less important parts. Machine learning methods like the Support vector machine can be employed for this task. Optimization of produced traces according to given criterion could be another subject of the future work. Example of such a criterion is producing of time optimal test runs.

Complete presented MBT solution needs a proper method for quantitative evaluation of produced tests according to specified criteria. Without this metric, it is hard to judge the quality of future improvements. In the thesis, the node and edge coverage of TA model were used. The first step in future work should be to define and implement various advanced measures to overcome this problem. Some possibilities are stated in 5.8. Traceability of requirements in the specification with TA model could also be a promising idea.



With the appropriate evaluation metrics, the work can continue by one of this direction. For instance, tests can be generated based on a hypothesis. Combination of the testing with model checking of relevant model parts could bring interesting results. Fault injection into SUT is problematic in the area of interest. Validation of the model by the mutation analysis is the promising approach for systematic model validation. Finally, the Real-Time aspect of the Taster implementation can be mentioned. Taster in the actual version is not capable of working with delays smaller than one hundred milliseconds. In conclusion topics for future work are recapitulated as a list.

- Implementation of the Urgent and Committed locations in the TA model.
- Markov chain based user profile.
- Layered/hierarchical architecture of the model.
- Enhancement of systematic algorithm.
- Assignment of relevancies by Machine learning methods.
- Suitable evaluation metrics.
- Avoiding of test cases covered by the original test suite.
- Combination of testing with model checking.
- Model validation by the mutant analysis.
- Real-Time properties of Taster tool.

Chapter 7

Conclusion

The FlexRay communication controller is configured by almost one hundred parameters. From validation point of view it is necessary to evaluate if actual values conform with values specified, e.g. specified by a network designer. Identification of these parameters usually requires measurements methods. Available methods, as well as solutions available on the market, were overviewed in State of Art section 2.1. In the list of available measurement methods, the blank spaces were identified. In general, no measurement methods focused on individual FlexRay node were available.

The first goal was analysis of the FlexRay parameterization using of FlexRay communication system specification. The parameter set defined in the standard was compared with two independent implementations the FlexRay communication controllers available on the market. The first sample came from Freescale and the second was produced by Texas Instruments. The analysis creates the background for following work – measurement methods for selected parameters.

(Objective: Analysis of the configuration parameters set.)

According to Protocol Operational Control (POC) states, the parameters are divided into three logical groups. They are wakeup, startup, and synchronization. Wakeup phase is intended to power up nodes from low power mode. The startup provides initial communication cycle schedule. Last one – synchronization is one of the key concepts of the protocol. It guarantees that each node transmits its frames in right time. In each group, several parameters requiring validation methods were found. Afterward, new methods were designed, which were in details described in corresponding sections. These measurement methods are in facts algorithms capable of identifying the values from bus communication. According to the POC states, the measurement methods are capable of revealing seven wakeup parameters, five startup parameters, and four synchronization related values.

(Objective: Evaluation of single FlexRay node parameters – design of measurement algorithms.)

The experimental approach was chosen for evaluation of the algorithms. Capabilities of production controllers are limited in the meaning of control over precise frame transmission and reception time. Special FlexRay controller fulfilling these requirements was developed for FPGA and used for these experiments. The algorithms were coded into this controller in C. Two types of EUT were utilized for the evaluation of presented methods. There were Freescale MC9S12XF and Texas Instruments TMS570LS31. The measurement setup and the results are stated in the corresponding section. It can be concluded that all of the presented methods work as expected. Some minor limitations were identified in offset correction measurability.

(Objective: Implementation and evaluation of the measurement methods.)

The last piece related to the FlexRay is the discussion of measurement accuracy and speed. Inaccuracy is caused by the difference in the clock frequencies of the tester and

tested device – each is driven by independent clock source (crystal oscillator). This inaccuracy is explained, and the methods to deal with this issue are presented. All measurement methods need some small time amount to a parameter identification. In general, it takes few communication cycles. A number of each algorithm steps and its dependency on the FlexRay parameters is summarized in Table 4.9.

(Objective: Characterization of measurement methods accuracy and time requirements.)

The challenge, which is hidden under the name Integration testing improvement by adaptation of the Model-Based Testing (MBT) methods starts with the selection of suitable modeling language. The Timed Automata (TA) were selected from the extensive list of options. Labeling of nodes by numbers called *Relevancies* was proposed. Purpose of the *Relevancies* is to provide a way to label more and less important parts of the model.

(Objective: Selection of suitable modeling formalism.)

To built a strong foundation for further work, the relevant theory was summarized. Namely, there is the theory of Timed Automata and analysis of a test generation possibilities from the model. Moreover, relations between presented theoretical concepts is outlined. The background is directly used in the following work.

(Objective: Overview of relevant theory.)

With the selected modeling formalism and the knowledge of the applicable theory, it was possible to propose MBT concept suitable for the area of interest – Integration testing of automotive electronic systems. The proposed concept uses TA model developed in UPPAAL tool. The model is loaded into a software tool, which performs Online test generation. The communication with the SUT is realized by a test adapter, which provides interconnection with HIL test place. The concept is depicted in Figure 5.3.

(Objective: Proposal of suitable MBT concept.)

During the time of realization of this thesis, I was the member of the team, which developed an HIL test place with a novel architecture based on National Instruments modular hardware. This project perfectly fits the thesis objectives and gives us a practical overlook to the researched area. Also, the Taster tool can perform testing on this platform using its NI VeriStand adapter. I was responsible for software part of the project.

(Objective: Development of HIL test place.)

The proposed MBT concept resulted in the implementation of a software utility, which implemented proposed algorithms and methods. The tool is called Taster and is programmed in C# using .NET technology. It offers three testing strategies (Random, Systematic, and Prioritized Random). HIL testbed with SUT is connectable by two types of test adapter. Tool features are highlighted in 5.9, and the user point of view is described in Appendix C. The first version, on which this work is based, was implemented by master thesis [70] under my supervision.

(Objective: MBT solution implementation.)

Three testing strategies are implemented. One of these algorithms uses *Relevancies* numbers to modify discrete uniform distribution. In this thesis these values are assigned by expert knowledge of the model designer. One of prospective direction for the future work is machine extraction of this information from test history database.

(Objective: Knowledge-based test generation.)

Two case studies were made to evaluate presented MBT solution. The SUT for the first study was a keyless access system (KESY). This system was implemented in LabView and simulated by NI VeriStand. The system was afterward modeled as TA network, and several test runs were performed. At first, the ability to detect fault was tested by inserting three artificial errors into the system. The second study was more complex. The SUT was a real Trunk opening system in the HIL laboratory of our industrial partner. The test runs were performed with two different TA models. All three strategies were tried. Test runs were characterized according to the activation of inputs and outputs. Afterwards, the progress of node and edge coverage was analyzed for selected test runs. The results show that the proposed MBT solution is suitable to given problem and opens multiple perspective directions for future research.

(Objective: Evaluation by a case study.)



References

- [1] U. Abelein, H. Lochner, D. Hahn, and S. Straube. *Complexity, quality and robustness - the challenges of tomorrow's automotive electronics*. In: *2012 Design, Automation Test in Europe Conference Exhibition (DATE)*. 2012. 870-871.
- [2] ISO 17458-2:2013(E). *Road vehicles – FlexRay communications system – Part 2: Data link layer specification*. . International Organization for Standardization.
- [3] E. Bringmann, and A. Krämer. *Model-Based Testing of Automotive Systems*. In: *2008 1st International Conference on Software Testing, Verification, and Validation*. 2008. 485-493.
- [4] ISO 17458-3:2013(E). *Road vehicles – FlexRay communications system – Part 3: Data link layer conformance test specification*. . International Organization for Standardization.
- [5] M. Desogus, M.S. Reorda, L. Sterpone, V.A. Avantaggiati, G. Audisio, and M. Sabatini. *Validation and robustness assessment of an automotive system*. 2013.
- [6] Jiří Novák. New measurement method of sample point position in controller area network nodes. *Measurement*. 2008, 41 (3), 300 - 306. Innovative Design and Paradigms in Instrumentation and Measurements.
- [7] Roman Obermaisser. *Time-Triggered Communication*. 1st edition. Boca Raton, FL, USA: CRC Press, Inc., 2011. ISBN 1439846618, 9781439846612.
- [8] M. Paulitsch, W. Steiner, R. Obermaisser, and C. El Salloum. In: *Time-Triggered Communication*, CRC Press, 2011. Core Algorithms. ISBN 978-1-4398-4661-2. <http://dx.doi.org/10.1201/b111155-5>.
- [9] P. Pop, A. Goller, T. Pop, and P. Eles. In: *Time-Triggered Communication*, CRC Press, 2011. Development Tools. ISBN 978-1-4398-4661-2. <http://dx.doi.org/10.1201/b111155-16>.
- [10] M. Heinz, V. Hoss, and K.D. Muller-Glaser. *Physical Layer Extraction of FlexRay Configuration Parameters*. In: *Rapid System Prototyping, 2009. RSP '09. IEEE/IFIP International Symposium on*. 2009. 173-180.
- [11] E. Armengaud, A. Steininger, and M. Horauer. *Automatic Parameter Identification in FlexRay based Automotive Communication Networks*. In: *2006 IEEE Conference on Emerging Technologies and Factory Automation*. 2006. 897-904.
- [12] E. Armengaud, and A. Steininger. *Remote measurement of local oscillator drifts in FlexRay networks*. In: *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09*. 2009. 1082-1087.
- [13] H.-P. Company. *Fundamentals of Quartz Oscillators*. 1997. In HP Application Note 200-2.
- [14] R.W. Johnson, J.L. Evans, P. Jacobsen, J.R. Thompson, and M. Christopher. The changing automotive environment: high-temperature electronics. *Electronics*

- Packaging Manufacturing, IEEE Transactions on.* 2004, 27 (3), 164-176.
DOI 10.1109/TEPM.2004.843109.
- [15] Mirko Conrad, and Ines Fey. In: *Automotive Embedded Systems Handbook*, CRC Press, 2008. Testing Automotive Control Software. ISBN 978-0-8493-8026-6.
<https://doi.org/10.1201/9780849380273.ch11>.
- [16] Justyna Zander, Ina Schieferdecker, and PieterJ Mosterman. In: CRC Press, 2011. A Taxonomy of Model-Based Testing for Embedded Systems from Multiple Industry Domains. ISBN 978-1-4398-1845-9.
<http://dx.doi.org/10.1201/b11321-2>.
- [17] M.T.B. Waez, J. Dingel, and K. Rudie. A survey of timed automata for the development of real-time systems. *Computer Science Review.* 2013, 9 1-26.
DOI 10.1016/j.cosrev.2013.05.001.
- [18] "Desel. In: *"Unifying Petri Nets: Advances in Petri Nets"*. "Berlin: "Springer Berlin Heidelberg", "2001". ISBN "978-3-540-45541-7".
"http://dx.doi.org/10.1007/3-540-45541-8_1" .
- [19] Rajeev Alur, and D. L. Dill. *Automata for Modeling Real-time Systems*. In: *Proceedings of the Seventeenth International Colloquium on Automata, Languages and Programming*. New York, NY, USA: Springer-Verlag New York, Inc., 1990. 322-335. ISBN 0-387-52826-1.
<http://dl.acm.org/citation.cfm?id=90397.90438>.
- [20] Johan Bengtsson, and Wang Yi. *Timed Automata: Semantics, Algorithms and Tools*. Lecture Notes in Computer Science. 2004.
http://dx.doi.org/10.1007/978-3-540-27755-2_3.
- [21] "T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine". "Symbolic Model Checking for Real-Time Systems". *"Information and Computation"*. "1994", "111" ("2"), "193 - 244". DOI "http://dx.doi.org/10.1006/inco.1994.1045". "".
- [22] UPPAAL Team, and others. *UPPAAL 4.0: Small tutorial, November 2009*.
http://www.it.uu.se/research/group/darts/uppaal/small_tutorial.pdf.
Accessed: August 2017.
- [23] "Srba J.". In: "Ciardo, eds. *"Applications and Theory of Petri Nets 2005: 26th International Conference"*. "Berlin: "Springer Berlin Heidelberg", ISBN "978-3-540-31559-9".
"http://dx.doi.org/10.1007/11494744_22" .
- [24] Mark Utting, Alexander Pretschner, and Bruno Legeard. A Taxonomy of Model-based Testing Approaches. *Softw. Test. Verif. Reliab.* 2012, 22 (5), 297-312.
DOI 10.1002/stvr.456.
- [25] David Lee, and Mihalis Yannakakis. Principles and methods of testing finite state machines-a survey. *Proceedings of the IEEE.* 1996, 84 (8), 1090-1123.
- [26] Matthias Grochtmann, Joachim Wegener, and Klaus Grimm. *Test case design using classification trees and the classification-tree editor CTE*. In: *Proceedings of Quality Week*. 1995. 30.
- [27] "Aichernig. In: *"Tests and Proofs: 7th International Conference"*. "Berlin: "Springer Berlin Heidelberg", ISBN "978-3-642-38916-0".
"http://dx.doi.org/10.1007/978-3-642-38916-0_2" .
- [28] Edmund M Clarke, Orna Grumberg, and Doron Peled. *Model checking*. MIT press, 1999.

- [29] "Kim. In: *Formal Methods for Industrial Critical Systems: 20th International Workshop*. "Cham": "Springer International Publishing", "47–61". ISBN "978-3-319-19458-5".
["http://dx.doi.org/10.1007/978-3-319-19458-5_4"](http://dx.doi.org/10.1007/978-3-319-19458-5_4) .
- [30] ANDERS HESSEL. *Model-Based Test Case Generation for Real-Time Systems*. Ph.D. Thesis, Uppsala University. 2007.
- [31] Marius Mikucionis. *Online Testing of Real-Time Systems*. Ph.D. Thesis, Aalborg University. 2010.
- [32] *Uppaal TRON Website*.
<http://people.cs.aau.dk/~marius/tron>. Accessed: 2015-09-09.
- [33] Marius Mikucionis. *UPPAAL TRON: Testing Real-time systems ONLINE*. MoDES project meeting. 2007.
<http://people.cs.aau.dk/~marius/tron/MoDES-2007.pdf>.
- [34] *Uppaal CoVer Homepage*.
<http://www.hessel.nu/CoVer/>. Accessed: 2015-09-09.
- [35] Johan Blom, Anders Hessel, Bengt Jonsson, and Paul Pettersson. *Specifying and Generating Test Cases Using Observer Automata*. Lecture Notes in Computer Science. 2005.
http://dx.doi.org/10.1007/978-3-540-31848-4_9.
- [36] Jürgen Großmann, Philip Makedonski, Hans-Werner Wiesbrock, Jaroslav Svacina, Ina Schieferdecker, and Jens Grabowski. In: CRC Press, 2011. Model-Based X-in-the-Loop Testing. ISBN 978-1-4398-1845-9.
<https://doi.org/10.1201/b11321-13>.
- [37] J. Hänsel, D. Rose, P. Herber, and S. Glesner. *An Evolutionary Algorithm for the Generation of Timed Test Traces for Embedded Real-Time Systems*. In: *2011 Fourth IEEE International Conference on Software Testing, Verification and Validation*. 2011. 170-179.
- [38] Jan Peleska, Artur Honisch, Florian Lapschies, Helge Löding, Hermann Schmid, Peer Smuda, Elena Vorobev, and Cornelia Zahlten. *A Real-world Benchmark Model for Testing Concurrent Real-time Systems in the Automotive Domain*. In: *Proceedings of the 23rd IFIP WG 6.1 International Conference on Testing Software and Systems*. Berlin, Heidelberg: Springer-Verlag, 2011. 146–161. ISBN 978-3-642-24579-4.
<http://dl.acm.org/citation.cfm?id=2075545.2075556>.
- [39] ISO 17458-4:2013(E). *Road vehicles – FlexRay communications system – Part 4: Electrical physical layer specification*. . International Organization for Standardization.
- [40] C. El Salloum, and K. Bilic. In: *Time-Triggered Communication*, CRC Press, 2011. FlexRay. ISBN 978-1-4398-4661-2.
<http://dx.doi.org/10.1201/b11155-7>.
- [41] Christoph Schmutzler, Martin Simons, and Jürgen Becker. *On Demand Dependent Deactivation of Automotive ECUs*. In: *Proceedings of the Conference on Design, Automation and Test in Europe*. San Jose, CA, USA: EDA Consortium, 2012. 69–74. ISBN 978-3-9810801-8-6.
<http://dl.acm.org/citation.cfm?id=2492708.2492726>.

- [42] Jennifer Lindelius Welch, and Nancy Lynch. A New Fault-tolerant Algorithm for Clock Synchronization. *Inf. Comput.*. 1988, 77 (1), 1–36. DOI 10.1016/0890-5401(88)90043-0.
- [43] *FlexRay Protocol Specification V2.1 Rev. A*. 2005.
- [44] *MC9S12XF512 Reference Manual*. 2010. Freescale Semiconductor. Rev.1.20.
- [45] *TMS570LS31x/21x 16/32-Bit RISC Flash Microcontroller*. 2011. Texas Instruments. Preliminary.
- [46] *Road vehicles – Local Interconnect Network (LIN) – Part 6: Protocol conformance test specification*. . International Organization for Standardization.
- [47] Martin Paták. *Methods for testing the flexray start-up mechanism*. Master’s Thesis, CTU in Prague. 2012.
- [48] M. Okrouhly, and D. Waraus. *Anti-lock braking system based on FlexRay protocol*. In: *Proceedings of the 6th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems*. 2011. 283-286.
- [49] Jiří Blecha. *Ethernet/FlexRay Gateway, hardware and firmware*. Bachelor’s Thesis, CTU in Prague. 2012.
- [50] Martin Zeman. *Firmware of Ethernet/FlexRay Gateway*. Bachelor’s Thesis, CTU in Prague. 2012.
- [51] *FlexRay Communication Controller IP*. <http://www.webcitation.org/6mVX1FHki>. Robert Bosch GmbH. 2016. Accessed: 2017-02-18.
- [52] "Axel Belinfante". *JTorX: A Tool for On-Line Model-Driven Test Derivation and Execution*. In: "Javier Esparza, and Rupak Majumdar", eds. *Tools and Algorithms for the Construction and Analysis of Systems*. "Springer Verlag", "2010". "266–270". ISBN "978-3-642-12001-5".
- [53] Margus Veanes, Colin Campbell, Wolfgang Grieskamp, Wolfram Schulte, Nikolai Tillmann, and Lev Nachmanson. *Model-Based Testing of Object-Oriented Reactive Systems with Spec Explorer*. In: *Formal Methods and Testing*. Springer Verlag, 2008. 39-76. ISBN 978-3-540-78916-1. <http://dl.acm.org/citation.cfm?id=1806209.1806211>.
- [54] Arilo C. Dias Neto, Rajesh Subramanyan, Marlon Vieira, and Guilherme H. Travassos. *A Survey on Model-based Testing Approaches: A Systematic Review*. In: *Proceedings of the 1st ACM International Workshop on Empirical Assessment of Software Engineering Languages and Technologies: Held in Conjunction with the 22Nd IEEE/ACM International Conference on Automated Software Engineering (ASE) 2007*. New York: ACM, 2007. 31–36. ISBN 978-1-59593-880-0. <http://doi.acm.org/10.1145/1353673.1353681>.
- [55] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation - international edition (2. ed)*. Addison-Wesley, 2003. ISBN 978-0-321-21029-6.
- [56] "Büchi. In: *The Collected Works of J. Richard Büchi*". "New York: "Springer New York", "1990". ISBN "978-1-4613-8928-6". ["http://dx.doi.org/10.1007/978-1-4613-8928-6_23"](http://dx.doi.org/10.1007/978-1-4613-8928-6_23) .
- [57] "Garavel. In: *Formal Methods for Protocol Engineering and Distributed Systems: FORTE XII / PSTV XIX'99 IFIP TC6 WG6.1 Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE XII) and Protocol Specification*". "Boston: "Springer

- US”, ISBN ”978-0-387-35578-8”.
["http://dx.doi.org/10.1007/978-0-387-35578-8_11"](http://dx.doi.org/10.1007/978-0-387-35578-8_11) .
- [58] ”Bérard. In: *”Control of Discrete-Event Systems: Automata and Petri Net Perspectives”*. ”London”: ”Springer London”, ”2013”. ”169–187”. ISBN ”978-1-4471-4276-8”.
["http://dx.doi.org/10.1007/978-1-4471-4276-8_9"](http://dx.doi.org/10.1007/978-1-4471-4276-8_9) .
- [59] *NI VeriStand™.NET API Help*.
http://zone.ni.com/reference/en-XX/help/372846J-01/vsnetapis/bp_vsnetapis/. Accessed: August 2017.
- [60] Lance Fortnow. The status of the P versus NP problem. *Communications of the ACM*. 2009, 52 (9), 78–86.
- [61] *Hierholzer’s Algorithm for directed graph*.
<http://www.webcitation.org/6rGwoiCEb>. GeeksforGeeks. 2017. Accessed: 2017-06-16.
- [62] Steven S. Skiena. *The Algorithm Design Manual*. Springer Publishing Company, Incorporated, 2008. ISBN 978-1-84800-069-8.
- [63] Harold Thimbleby. The directed Chinese Postman Problem. *Software: Practice and Experience*. 2003, 33 (11), 1081–1096. DOI 10.1002/spe.540.
- [64] David L. Applegate, Robert E. Bixby, Vasek Chvatal, and William J. Cook. *The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics)*. Princeton, NJ, USA: Princeton University Press, 2007. ISBN 0691129932, 9780691129938.
- [65] P. E. Ammann, and P. E. Black. *A specification-based coverage metric to evaluate test sets*. In: *Proceedings 4th IEEE International Symposium on High-Assurance Systems Engineering*. 1999. 239-248.
- [66] Jan Friso Groote, Tim W.D.M. Kouters, and Ammar Osaiweran. *Specification guidelines to avoid the state space explosion problem*. 2015.
<http://dx.doi.org/10.1002/stvr.1536>.
- [67] Mirko Conrad. *Systematic testing of embedded automotive software the classification tree method for embedded systems (CTM/ES)*. In: *Dagstuhl Seminar Proceedings*. 2005.
- [68] *”Chapter 4 - Selecting your tests ”*. In: ”Utting, eds. ”San Francisco”: ”Practical Model-Based Testing, ”2007”. ”107 - 137”. ISBN ”978-0-12-372501-1”.
<https://doi.org/10.1016/B978-012372501-1/50005-3>.
- [69] Bernhard K. Aichernig, Klaus Hörmaier, and Florian Lorber. *Debugging with Timed Automata Mutations*. In: *Proceedings of the 33rd International Conference on Computer Safety, Reliability, and Security - Volume 8666*. New York, NY, USA: Springer-Verlag New York, Inc., 2014. 49–64. ISBN 978-3-319-10505-5.
http://dx.doi.org/10.1007/978-3-319-10506-2_4.
- [70] Tomáš Grus. *Implementation of Integration Testing Test Cases Generation Tool*. Master’s Thesis, CTU in Prague. 2014.
- [71] Terence Parr. *The definitive ANTLR 4 reference*. Pragmatic Bookshelf, 2013.
- [72] ”Edsger W. Dijkstra”. *”Algol 60 translation : An Algol 60 translator for the x1 and Making a translator for Algol 60”*. . ”Mathematisch Centrum
- [73] Michal Veselka. *TaSysTest and EXAM software interface implementation*. Bachelor’s Thesis, CTU in Prague. 2016.

- [74] M. Farsi, K. Ratcliff, and M. Barbosa. An overview of controller area network. *Computing Control Engineering Journal*. 1999, 10 (3), 113-120.
- [75] *NORM, D. I. N. 72552: Klemmenbezeichnungen in Kraftfahrzeugen. 1971.* . Deutsches Institut für Normung.

Appendix A

Author's Publications and Grants

A.1 Publications Related to the Thesis

A.1.1 Publications in Journals with Impact Factor

- [J1] SOBOTKA, J. and NOVÁK, J. FlexRay ECU mission critical parameters measurement. *Measurement*. 2017, **100** pp. 213-222. ISSN 0263-2241.
<http://www.sciencedirect.com/science/article/pii/S0263224116307357>
Co-authorship: 70%

A.1.2 International Conference Proceedings

- [C1] SOBOTKA, J. and NOVÁK, J. FlexRay Controller with Special Testing Capabilities. In: *2012 International Conference on Applied Electronics*. 2012 International Conference on Applied Electronics. Plzeň, 06.09.2012 - 08.09.2012. Pilsen: University of West Bohemia. 2012, pp. 269-272. ISSN 1803-7232. ISBN 978-80-261-0038-6.
Co-authorship: 50%

Cited by:

- Shreejith, S. , Fahmy, S.A. Enhancing communication on automotive networks using data layer extensions. *FPT 2013 - Proceedings of the 2013 International Conference on Field Programmable Technology*, 2013. ISBN 978-1-47992-199-0.
 - Shreejith, S., Fahmy, S.A., Lukaseiwycz, M. Accelerating validation of time-triggered automotive systems on FPGAs *FPT 2013 - Proceedings of the 2013 International Conference on Field Programmable Technology*, 2013. ISBN 978-1-47992-199-0.
 - Shreejith, S., Fahmy, S.A. Extensible FlexRay communication controller for FPGA-based automotive systems *IEEE Transactions on Vehicular Technology*, 2015.
 - Radhiga, R.; Pradeep, J. Design of FlexRay communication controller protocol for an automotive application *Proceedings of 2015 IEEE 9th International Conference on Intelligent Systems and Control (ISCO)*, 2015.
- [C2] SOBOTKA, J. and NOVÁK, J. Methods for Measurement of Flexray Node Basic Timing Parameters. In: *XX IMEKO World Congress 2012 - Proceedings*. XX IMEKO World Congress 2012 - Metrology for Green Growth. Busan, BEXCO, 09.09.2012 - 14.09.2012. Busan: IMEKO. 2012, ISBN 978-1-62748-190-8.
Co-authorship: 50%

- [C3] SOBOTKA, J. and NOVÁK, J. Automation of Automotive Integration Testing Process. In: *IDAACS 2013 - Proceedings of the 2013 IEEE 7th International Conference on Intelligent Data Acquisition and Advanced Computing Systems*. IDAACS 2013 - Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications. Berlin, 12.09.2013 - 14.09.2013. Berlin: IEEE, 2013, pp. 349-352. ISBN 978-1-4799-1426-5.
Co-authorship: 50%
- Cited by:*
- Sagstetter, F.; Waszecki, P.; Steinhorst, S.; Lukasiewicz, M.; Chakraborty, S. Multischedule Synthesis for Variant Management in Automotive Time-Triggered Systems *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 4, pp. 637 - 650, 2016. ISSN 0278-0070.
- [C4] SOBOTKA, J. and NOVÁK, J. Testing Automotive Reactive Systems using Timed Automata. In: *IDAACS 2017 - Proceedings of the 2017 IEEE 9th International Conference on Intelligent Data Acquisition and Advanced Computing Systems*. IDAACS 2017 - Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications. Bucharest, 21.09.2017 - 23.09.2017. (*Accepted.*)
Co-authorship: 80%
- [C5] SOBOTKA, J. and NOVÁK, J. Application of Extended Timed Automata to Automotive Integration Testing. In: *VALID 2017 - The Ninth International Conference on Advances in System Testing and Validation Lifecycle*. Athens, Greece, 8.10.2017 - 12.10.2017. ISSN: 2308-4316 ISBN: 978-1-61208-593-7 (*Accepted.*)
Co-authorship: 80%

A.2 Selected Grants Related to the Thesis

- [G1] Kocourek, P. (2005-2011). Josef Božek Research Center of Engine and Automotive Technology II, *Czech Ministry of Education*, grant No. 1M0568. (*Team Member.*)
- [G2] Ripka, P. (2009-2012). Sensors and intelligent sensor systems, *Czech Science Foundation (GACR)*, grant No. GD102/09/H082. (*Team Member.*)
- [G3] Macek, J. (2012-2017). Josef Božek Competence Centre for Automotive Industry, *Technological Agency, Czech Republic, programme Centres of Competence*, project # TE01020020. (*Team Member.*)
- [G4] Sobotka, J. (2013). Methods for diagnostics and solving of the critical situations in individual transport, *Czech Technical University in Prague, Czech Republic*, SGS13/085/OHK3/1T/13.
- [G5] Sobotka, J. (2016-2017). Model-Based Testing methods for automotive electronics systems, *Czech Technical University in Prague, Czech Republic*, SGS16/171/OHK3/2T/13.



Appendix B

Abbreviations

ABS	Anti-lock Braking System
API	Application Programming Interface
ASR	Anti-Slip Regulation
BFS	Breadth-first search
CAN	Controller Area Network
CAS	Collision Avoidance Symbol
CC	Communication Controller
CCS	Calculus of Communicating Systems
CPP	Chinese Postman Problem
CSP	Communicating Sequential Processes
DFS	Depth-first search
ECU	Electronic Control Unit
EUT	ECU under test
FSM	Finite-state machine
FTM	Fault-tolerant midpoint
GUI	Graphical user interface
HIL	Hardware-in-the-loop
IDE	Integrated development environment
I/O	Input/Output
ISO	International Organization for Standardization
LIN	Local Interconnect Network
LTL	Linear Temporal Logic
LTS	Labelled Transition System
MBT	Model-Based Testing
MC	Model Checker
MC/DC	Modified condition/decision coverage
MCU	Microcontroller Unit

OEM	Original Equipment Manufacturer
OOP	Object-oriented programming
OSI	Open Systems Interconnection
PCB	Printed Circuit Board
PN	Petri net
POC	Protocol Operational Control
PWM	Pulse-width modulation
PXI	PCI eXtensions for Instrumentation
RFID	Radio-frequency identification
RT	Real-time
SDL	Specification and Description Language
SMC	Statistical Model Checker
SoC	System on a chip
SUT	System under Test
TA	Timed Automaton
TDMA	Time Division Multiple Access
TSP	Traveling Salesman Problem
UML	Unified Modeling Language
VHDL	VHSIC Hardware Description Language
XML	Extensible Markup Language

Appendix C

Taster User Guide

In this section, tool Taster described in 5.9, is presented from the user point of view. To provide the best possible readability are screenshots placed in landscape orientation. Since the pictures, come from the wide screen display. Setup of HIL testbed, as well as software installation, is not covered in this guide. The intention is to provide reader look up how the proposed solution works from the user point of view in reasonable detail. It should not be seen as the step by step guide to deploy the presented solution.

Prerequisites:

- Windows 7 on reasonable PC capable run NI VeriStand/EXAM
- Microsoft .NET Framework 4.5
- NI VeriStand 2015 or EXAM Version: 4.3.4
- UPPAAL 4.1.19
- Taster

The first step of the testing process is the design of SUT model in UPPAAL. Supported subset of modeling language is listed in 5.2. The proper design of the model is the key point. Environment part guides input generation, and observer part defines correct and incorrect system behavior. A Timed Automaton example in UPPAAL environment is shown in Figure C.1.

Taster user interface is divided into two main windows. There are viewer and runtime windows. The viewer is depicted in Figure C.2. Model is loaded by selecting File → Open. If the model is syntactically compatible with Taster, it is loaded. Otherwise, the pop-up window with an error message is shown. On the right side is placed a list of available templates. By listing of templates, the user can view the prepared model.

Most of the features are integrated into Runtime window. PrintScreen is shown in Figure C.3. In the screen are shown all templates which create TA system. The size of each automaton depends on number and complexity of templates in the model. Active nodes are dynamically highlighted during a test run. In up, left corner is the pair of Connect and Disconnect buttons. They control connection with SUT by selected test adapted. The adapter is chosen by the radio buttons. EXAM/NI VeriStand can run on a separate machine - IP address or computer name is assigned in the box beside Disconnect button.

After connection to test facilities, it is possible to start a test run. In the middle top section of Runtime window are controls for influencing of test engine behavior. There are Run tick interval (step time) and model exploration strategy (Random, Systematic, Experimental). Minimal reasonable step time is 100 ms as Taster runs on Non-RT operation system, maximal step time is not limited. For the debugging purposes, it is implemented step function, which is similar to common IDEs. Testing can be stopped by one of the following situations. Related to SUT behavior is a violation of invariant

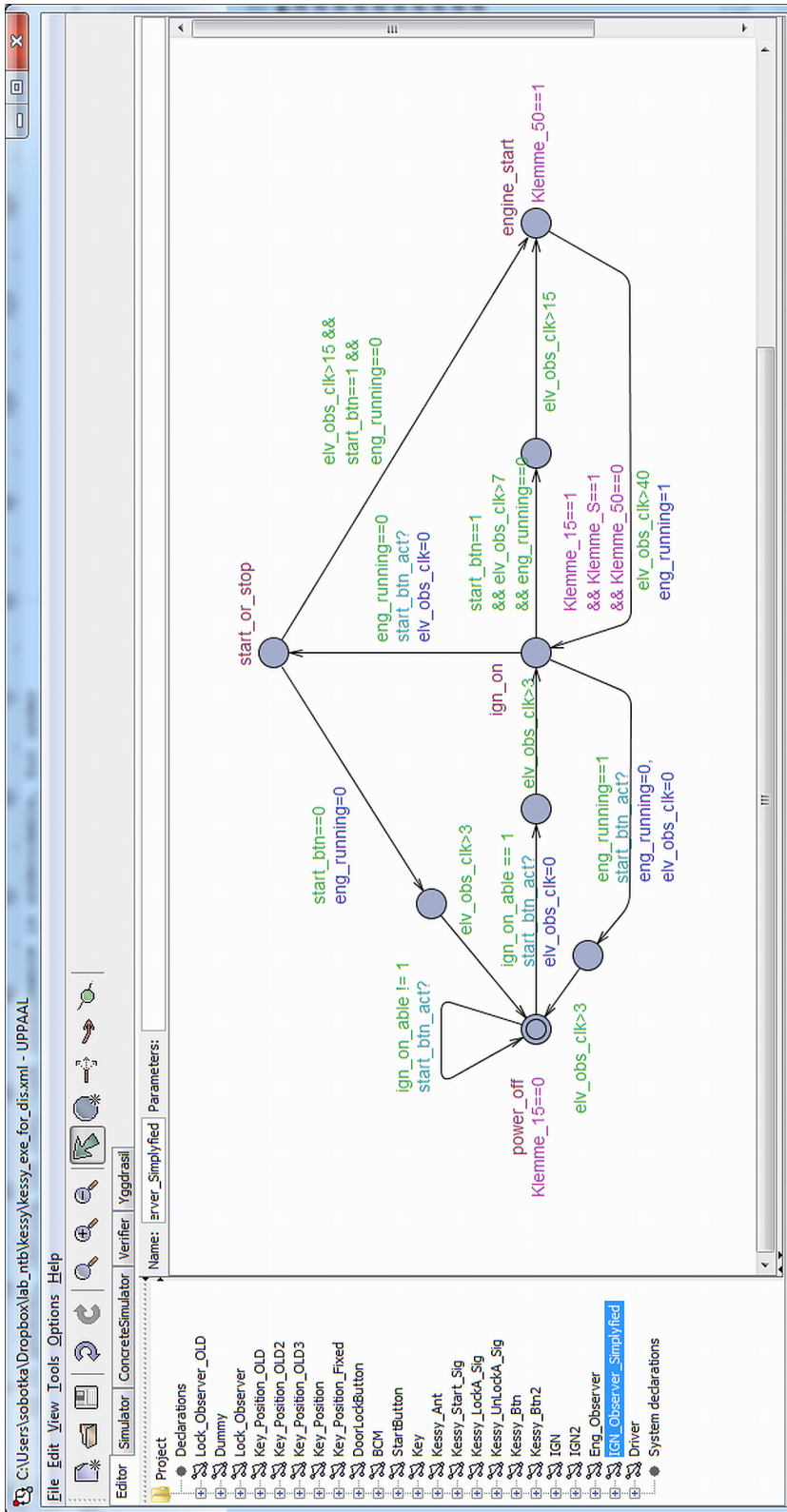


Figure C.1. Input Model

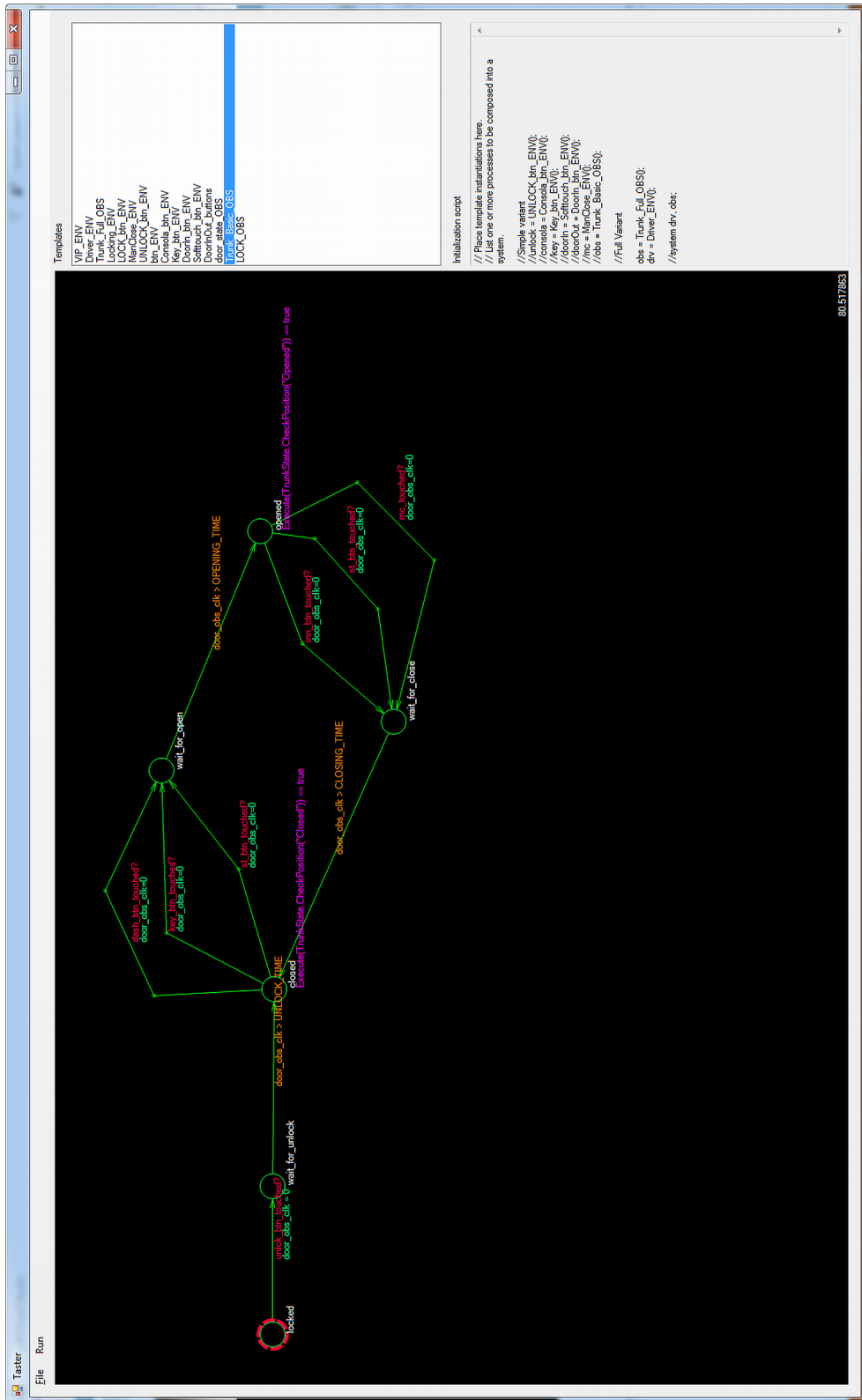


Figure C.2. Model Viewer

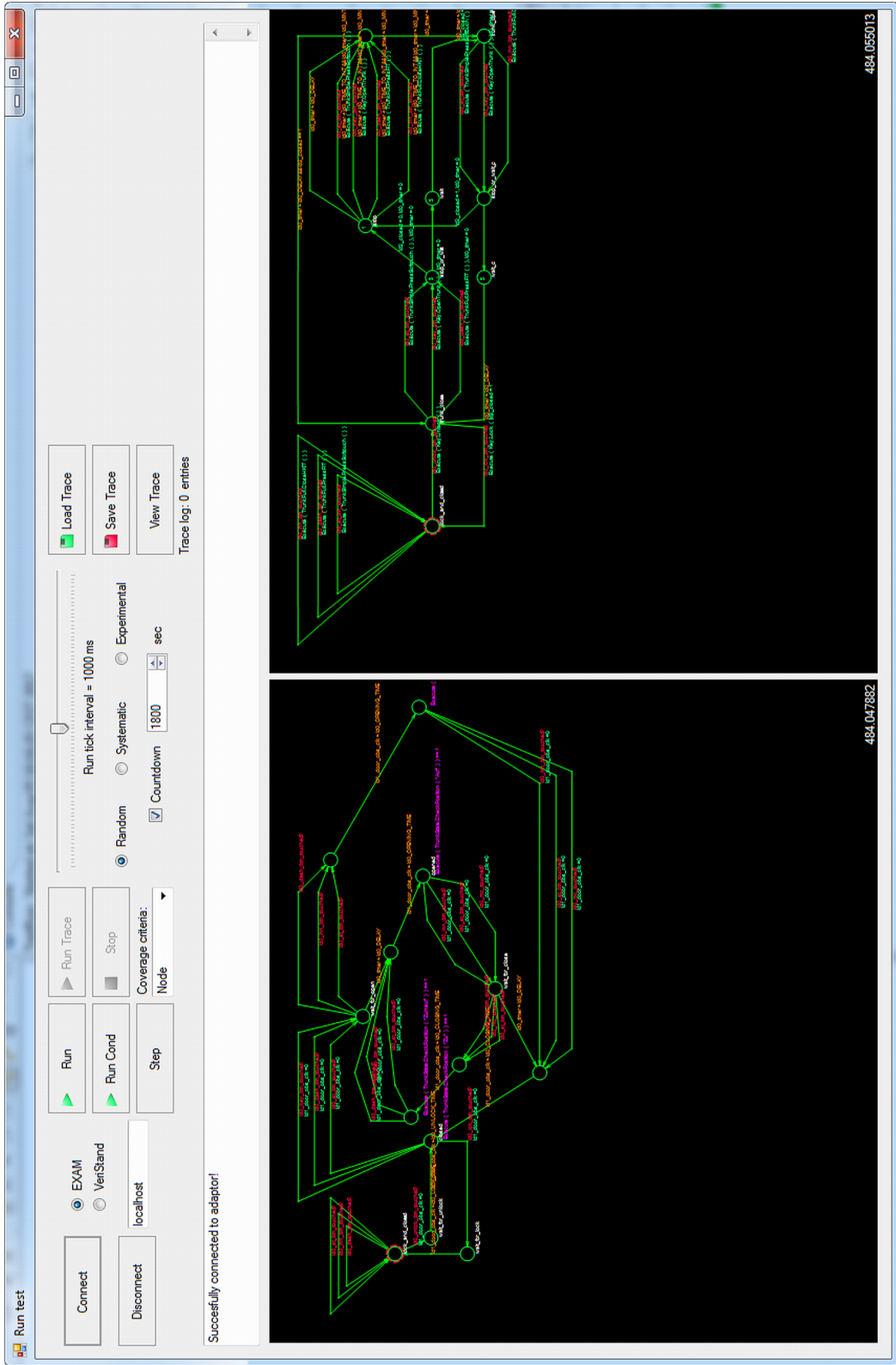


Figure C.3. Taster Runtime Screen

The screenshot shows the Trace Viewer application window. The main area contains a table of events with columns for Time and Event. The first event is highlighted in blue.

Time	Event
2906.745000907...	Zero step
2908.375437851...	UNLOCK_btn_ENV[0] took the edge from unlock_released to unlock_active triggering the sync 1...
2908.745231222...	no edge was taken
2909.7452285056	no edge was taken
2910.745230920...	no edge was taken
2911.7452378631	no edge was taken
2912.745239372...	no edge was taken
2913.745230014...	no edge was taken
2914.745267143...	no edge was taken
2915.745249031...	no edge was taken
2916.745233033...	no edge was taken
2917.7452665393	no edge was taken
2918.745251446...	UNLOCK_btn_ENV[0] took the edge from unlock_active to unlock_released
2919.787066185...	Trunk_Basic_OBS[6] took the edge from wait_for_unlock to closed
2921.857247730...	FIT_btn_ENV[1] took the edge from dash_btn_released to dash_btn_active triggering the sync id...
2921.857621124...	no edge was taken
2922.745248729...	no edge was taken
2923.745241787...	no edge was taken
2924.745248428...	no edge was taken
2925.745239674...	no edge was taken
2926.745242390...	no edge was taken

Below the table, there are two sections of text:

varid0_locked: Boolean, value = 1
varid0_PRESS_TIME: Integer, value = 10
varid0_DELAY_AFTER_PRESS: Integer, value = 10
varid0_DELAY: Integer, value = 20
varid0_OPENING_TIME: Integer, value = 30
varid0_CLOSING_TIME: Integer, value = 30
varid0_UNLOCK_TIME: Integer, value = 10
varid0_LOCK_TIME: Integer, value = 10
varid0_TIME_TO_INT: Integer, value = 3
varid1_unlock_clk: Clock, value = 0
varid2_dash_btn_clk: Clock, value = 0
varid3_key_btn_clk: Clock, value = 0
varid4_inn_btn_clk: Clock, value = 0
varid5_st_btn_clk: Clock, value = 0
varid6_mc_clk: Clock, value = 0
varid7_door_obs_clk: Clock, value = 0

Trace Statistics:
UNLOCK_btn_ENV[0]
unlock_released -> unlock_active: 1
unlock_active -> unlock_released: 1
FIT_btn_ENV[1]
dash_btn_released -> dash_btn_active: 6
dash_btn_active -> dash_btn_released: 6
FFB_btn_ENV[2]
key_noactive -> key_active: 0
key_active -> key_noactive: 0

Figure C.4. Trace Viewer