Department of Computer Science, Chair of Computer Engineering

In concurrence with

Master Thesis

In

Analysis and Specification of an AUTOSAR based ECU in compliance with ISO 26262 Functional Safety Standard

For Fulfilment of the Academic Degree

M.Sc. in Automotive Software Engineering

Submitted by: Vibhu Layal
Matriculation No.: 386681

Internal Supervisors: Prof. Dr. Wolfram Hardt
Mr. Mirko Lippmann
External Supervisors: Mr. Ulrick Mikesky
Mr. Matthias Specht

Date of Submission: XX Qwertz, 2016

Vibhu Layal

Analysis and Specification of an AUTOSAR based ECU in compliance with ISO 26262 Functional Safety Standard

Master Thesis, Department of Computer Science, Chair of Computer Engineering

Technische Universität Chemnitz, Qwertz 2016

Declaration

I hereby declare that this master thesis on the topic “Analysis and Specification of an AUTOSAR based ECU in compliance with ISO 26262 Functional Safety Standard” has been written by me and is entirely the product of work done by my own. I thereby also certify that the research work on this thesis is studied and refined by me independently. The efforts of gathering facts and figures for this master thesis from different sources and literatures have been faithfully and properly cited.

Neutraubling, XX Qwertz, 2016	
Place, Date	Vibhu Layal

Acknowledgment

My utmost gratefulness towards all those who have not only helped me with tips and tricks of the trade but also for directing me towards the solutions to some intermediate problems that I had to face while executing this master thesis. They further boosted me to a whole new level by appreciating and motivating me during those ups and downs. The feeling of stress or workload during this whole period was almost unheard of. Which was all due to the support and friendly nature of the staff at MBtech Group GmbH & Co. KGaA and that made this an enormous positive experience overall.

My first obligation goes towards my supervisor at MBtech Group GmbH & Co. KGaA, Mr. Ulrich Mikesky, firstly for inviting me for such an interesting thesis topic and then supporting me throughout the course. I would also like to extend my gratitude towards Mr. Matthias Specht, as he guided me from time to time and also gave exceptionally innovative ideas on how to approach different tasks assigned to me in various successful ways. He has been very comforting to me with all my problems during the coursework of this thesis. In the best of my honesty they have been an integral part of my thesis which finally led me to a successful completion.

Lastly, I would like to thank my Professor from Technische Universität Chemnitz, Prof. Dr. rer. nat. Wolfram Hardt and Mr. Mirko Lippmann from the Department of Computer Science for consenting to my decision to take this opportunity of the thesis offered to me at MBtech Group GmbH & Co. KGaA.

A heart-warming genuine thanks to everyone for helping me successfully complete my master thesis into a reality.

Place: Neutraubling Vibhu Layal

Date: XX Qwertz, 2016

Abstract

Safety has been always been an important part, irrespective of the field of work that it accounts for. The functional safety standard that is currently being used in the automotive domain is the ISO 26262. This is an adaptation of the IEC 61508 safety standard. It is directed as a basic functional safety standard for a variety of industries. The version of ISO 26262 that is used in this thesis is the final draft released in January, 2011.

In this thesis, various parts of the ISO 26262 functional safety standard are considered in order to understand the differences and interdependencies between them. The parts of ISO 26262 that are treated are as follows; Part 1: Vocabulary, Part 3: Concept phase, Part 4: Product development at the system level, Part 6: Product development at the software level and Part 9: Automotive Safety Integrity Level (ASIL)-oriented and safety-oriented analysis. During the entire course of this thesis the ISO 26262 standard is evaluated and the experience gained from it is jotted down.

The understanding gained during this thesis about the ISO 26262 can be applied to ongoing or new development processes. As safety can never be overlooked, the wisdom that belongs to the ISO 26262 can be generously used into embedded systems that demand certain levels of safety.

Keywords: ISO 26262, ASIL, software

Contents

List of Figures

List of Tables

List of Abbreviations

ABS	Anti-lock Braking System
ADAS	Advanced driver assistance system
WHO	World Health Organization
E/E	Electrical and/or electronic
OEM	Original Equipment Manufacturer
IC	Inter-Integrated Circuit
AUTOSAR	Automotive Open System Architecture
ASIL	Automotive Safety Integrity Level
E/E/PE	Electrical, electronic and programmable electronic
IEC ACOS	International Electro-technical Committee Advisory Committee of Safety
IEC	International Electro-technical Commission
PLC	Programmable logic controller
SIL	Safety integrity level
SIS	Safety Instrumented System
SIF	Safety Instrumented Function
ISO	International Organization for Standardization
ECU	Electronic Control Unit
SWC	Software component
RTE	Run Time Environment
BSW	Basic Software
OS	Operating System
MISRA	Motor Industry Software Reliability Association
QA	Quality assurance
UML	Unified Modelling Language
H&R	Hazard analysis and risk assessment
CAN	Controller Area Network
CRC	Cyclic Redundancy Check
FIT	Failures In Time
FMEA	Failure Mode and Effects Analysis

FTA	Fault Tree Analysis
HAZOP	HAZard and OPerability analysis
MMU	Memory Management Unit
MPU	Memory Protection Unit
ECC	Error-Correcting Code
PCB	Printed circuit board
RTOS	Real-time operating system
STE	Segment table entry
EROS	Extremely Reliable Operating System
PTE	Page table entry

1. Introduction

This is an introductory chapter to this master thesis. It explains the background, the aims and the problematic areas that were taken into account during the entire period of this thesis. The chapter also includes the channels that will be used for completing the aims and an elaborate perspective towards the expected outcomes.

The automotive domain has defined certain phases that are involved in any product lifecycle. These phases not only give the product lifecycle a structured approach but also helps in the traceability of different aspects in the later stages of the lifecycle. The figure 1.1 gives an outline to these phases that make up the 'V-Model' along with the hierarchy in which they lie.

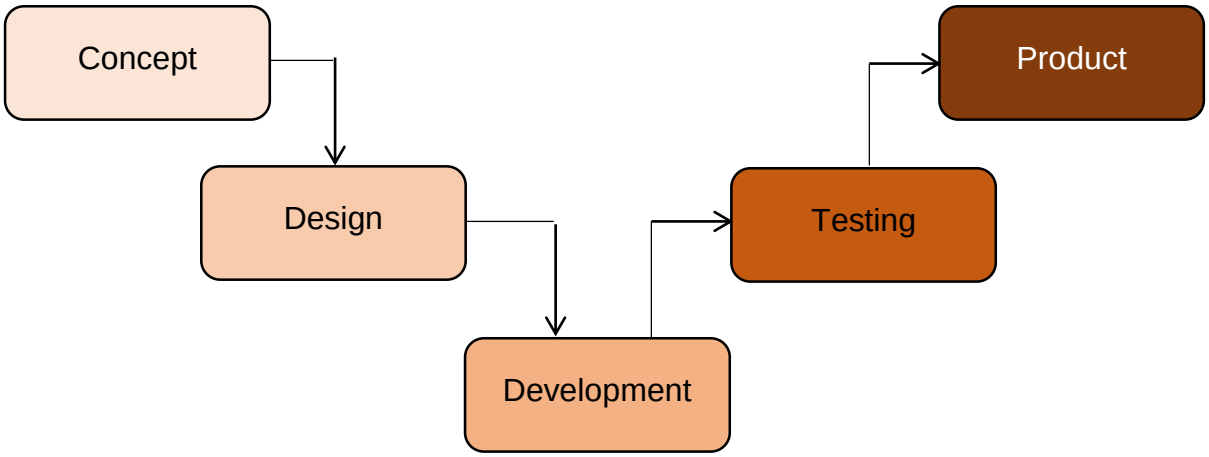


Figure 1. 1 Product development lifecycle

1.1 Motivation

The focal point in the automotive domain lies upon safety-critical applications. Although not all the applications are safety-critical but the former ones are significantly growing, bringing new and added pressures on engineers. Their work towards solving these safety challenges is not an easy task at hand. As we grow day-by-day, the automotive industry is under immense pressure to supply with new and improved vehicle safety systems, ranging from basic anti-lock braking systems (ABS) to extremely complex advanced driver assistance systems (ADAS) with accident prediction and avoidance capabilities.

An estimated 1.2 million people are part of fatal road accidents every year, according to the World Health Organization (WHO) [1]. Thus, the foremost important feature that should be a part of every future vehicle development is safety. New space-age safety systems in the vehicle are being developed where the obvious focus is on safety. But meanwhile, there is an even higher amount of software, sensors and actuators in a vehicle than ever before. This corresponds to a higher risk of having mostly software bugs along with a few hardware failures as well. Hence, it becomes an inevitable issue in the automotive industry to take this risk seriously. Which should result in proper, structured and supervised working methods and products. This leads to a strong urge for a process or a standard that can guide through the complete product lifecycle and at the same time guarantee proofs towards all security, safety and functional objectives of the system.

There is no denying to the fact that with their direct impact on human well-being, the electronic safety systems are experiencing increasingly stringent requirements. It becomes a challenging job for system designers to design safety systems while keeping the state-of-the-art functional safety requirements securely in the picture. Especially with the time to market urgency and managing increased application complexity combined, this becomes a very challenging task. Putting it in a nutshell the system engineers have to architect a system in a way that it should surely prevent dangerous failures or at least sufficiently control them whenever they occur. These fatal failures may arise from various points [2]:

- Irregular hardware failures
- Systematized hardware failures
- Systematized software failures

With the rise of development of driverless technology and "connected" cars, safety has become a different ballgame altogether. Some massive failure examples are cited in figure 1.2 [3].

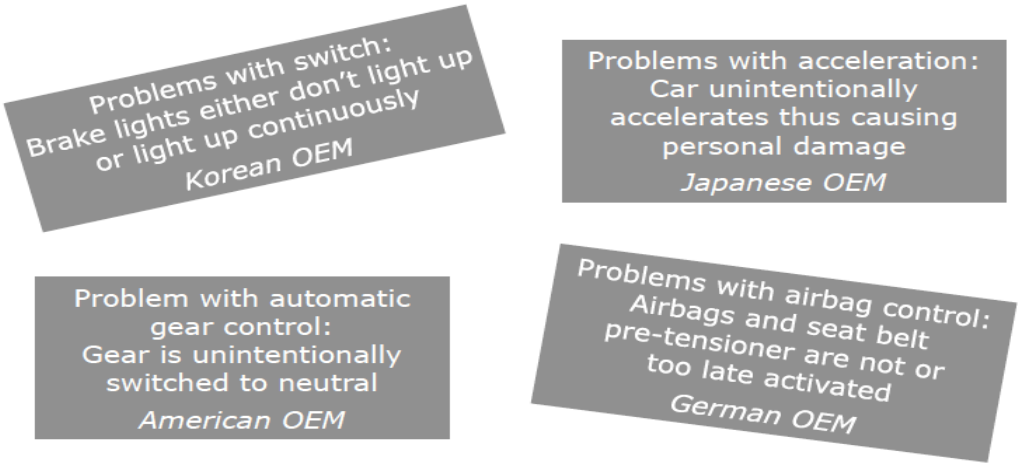


Figure 1. 2 Recent call-backs

1.2 Problem Statement

ISO 26262 which was published in 2011, now recognized as the state-of-the-art functional safety standard to address the increasing complexity of safety-relevant electrical and

electronic systems. This standard covers all those safety-related systems that are equipped in production passenger cars, and consists of one or more electrical and/or electronic (E/E) system.

The product and process related requirements dealing with the specific safety features in the final product are defined as properties in the ISO 26262 framework, which reduces the probability of systematic faults but cannot directly be observed as a feature of the product. Though, few automotive Original Equipment Manufacturers (OEM) and suppliers will argue on the importance of this standard. However, the main challenge lies in interpreting and implementing ISO 26262 the way it is intended in and not in the perspective that one takes it in. This is clearly due to its complexity and the lack of functional safety experts as this standard is relatively young within the industry.

Interpreting the standard differently depending upon the requirements by different people is a big issue. Figure 1.3 shows the parameters that revolve around ISO 26262 and which are usually misunderstood or sometimes too complex to handle during a development process. These parameters are well versed in the following sections of this thesis.

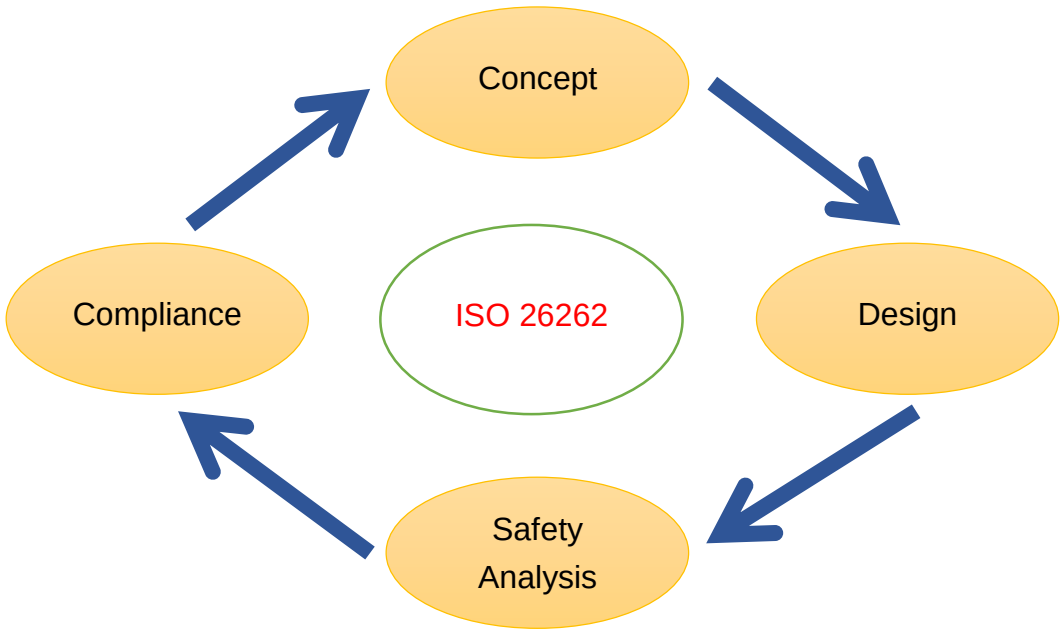


Figure 1. 3 Essential parameters of ISO 26262

1.3 Outline of the thesis

The orientation of this thesis is organized as described in the following chapters:

- Chapter 2 – This chapter will describe all the aspects of research involved during this thesis. It will include all the safety standards that were analysed in order for the better understanding of the ISO 26262. IEC 61508 was initially looked into to identify and build up the face of ISO 26262 as the later was derived from IEC 61508. Then the ISO 26262 will be given a concise description. Followed by Automotive Open System Architecture (AUTOSAR) 3.2. This chapter would also include all the tools, technologies and languages that were used for the completion of this master thesis.

- Chapter 3 – This chapter includes all the parts of ISO 26262 that were analysed during this thesis. Starting from the techniques being currently used in the industry and how can they be altered and improved further so as to benefit the whole development lifecycle of the product. The ISO 26262 Part 1: Vocabulary, deals with the common jargon that is used over the entire spread of parts of the ISO 26262. This is then followed by ISO 26262 Part 3: Concept phase that involves the concept evolution for the final product. The ISO 26262 Part 4: Product development at the system level consists of all the activities involved during development of the product at a system level. Going a notch deeper, the ISO 26262 Part 6: Product development at the software level deals with the development methods carried out for the product at the software level. Lastly, the ISO 26262 Part 9: Automotive Safety Integrity Level (ASIL)-oriented and safety-oriented analysis that has to do with the ASIL levels and other safety oriented activities for the respective product.

- Chapter 4 – This chapter involves diving deep into the ISO 26262 Part 6: Product development at the software level. Which gives way to the complete product development lifecycle at the software phase. The different mechanisms by which errors can be detected/handled are also pondered upon in this chapter. Various mechanisms are described and analysed with respect to their use in the automotive domain application and how well do they comply with the sixth part of the ISO 26262 functional safety standard.

- Chapter 5 – This chapter involves the testing and verification aspect as per the ISO 26262 Part 6: Product development at the software level. It gives an insight on the dos and don’ts for different testing techniques listed under the ISO 26262. It also includes the test specification and the respective work products obtained from the same.

- Chapter 6 – This chapter deals with the results obtained from this master thesis. It also broadens the perspective about this thesis if it needs to be taken into different dimensions. These dimensions can further be narrowed down according to one’s interest.

2. Literature Research

This chapter explains in detail all the areas that were researched for the completion of this thesis. The topics are as follows:

- IEC 61508
- ISO 26262
- AUTOSAR 3.2
- MISRA
- Tools
- Technologies

The above mentioned topics are highly essential in order to dive deep into this thesis topic and hence, their background along with their importance in the automotive domain is mentioned in this chapter.

2.1 IEC 61508

The international standard for electrical, electronic and programmable electronic (E/E/PE) equipment is the IEC 61508 functional safety standard. It started in the mid 1980's by the International Electro-technical Committee Advisory Committee of Safety (IEC ACOS). They cumulated a group of people so as to solve the standardization issues raised by the programmable electronic systems. This was the first time in history that these combined group of people treated safety as a system issue [4].

Figure 2. 1 Logo of IEC 61508

As every other standard, even this standard was disgraced at the beginning for its extensive documentations and use of not proven statistical methods for hardware failures, although it was a major step forward in many industries. Resulting in even more cost-effective implementations, this standard primary aim lies with risk-based safety-related system design. The standard also alike other standards, require the attention to detail that is highly important for any safe system design. It is considered a major leap for the technical world as far as the standards as concerned due to its large degree of international acceptance and other features.

2.1.1 Introduction

IEC 61508 is basically a safety publication by the International Electro-technical Commission (IEC). In other words, one can consider it as an "umbrella" document covering many industries and industry-related applications. The sole aim of this standard is to aid industries develop themselves by specifically tailoring the IEC 61508 standard. A secondary goal lies with this standard to enable the development of E/E/PE safety-related systems.

Although now several such industry specific standards have now been developed and with more on the way.

2.1.2 The standard

Safety-related systems that incorporates mechanical and/or E/E/PE devices are considered under the IEC 61508. They may include anything right from ball valves, electrical relays and switches all the way to programmable logic controllers (PLC). When failures of the safety functions performed by E/E/PE safety-related systems happen resulting into hazards, this standard covers these hazards under itself. Thus, "functional safety" is the overall program to insure that the safety-related E/E/PE system always turnabout into a safe state.

This standard does not cover problems like electric shock, long-term exposure to a toxic substance, hazardous falls, etc. But it has a term called as the safety integrity levels (SIL) that depicts a measure of the probable residual risks of the product under consideration. It also does not take care of low safety E/E/PE systems where a single E/E/PE system is enough to support the necessary risk reduction and the required safety integrity. This is usually for all the E/E/PE systems whose safety integrity level is less than 1, i.e., the system is only available for 99% or less of its total operational time [5].

IEC 61508 is mostly importantly concerned with the E/E/PE safety-related systems whose failures could adversely affect people and/or the environment. The environment may also include business loss and asset protection cases. The total IEC 61508 standard is divided into seven distinct parts, namely:

- Part 1: General requirements. This part is necessary for compliance
- Part 2: Requirements for E/E/PE safety-related systems. This part is necessary for compliance

- Part 3: Software requirements. This part is necessary for compliance

- Part 4: Definitions and abbreviations. This part is necessary for the information supporting its cause

- Part 5: Examples of methods for the determination of safety integrity levels. This part is necessary for the information supporting its cause

- Part 6: Guidelines on the application of Part 2 and Part 3. This part is necessary for the information supporting its cause

- Part 7: Overview of techniques and measures. This part is necessary for the information supporting its cause

The standard is spread upon on two fundamental concepts, namely: the safety life cycle and SILs. The safety life cycle is defined as an engineering process that describes all of the steps in order to achieve the aimed functional safety. Its philosophy is to develop a safety plan, execute it, document it and continue to follow it till its decommissioning. The decommissioning should also be aided with appropriate documentation as well. Hence, the outcome is a proper structured approach for the entire lifetime of the system. Keeping in mind that phases of planning, execution, validation, and documentation should also be similarly handled.

Steeping into the safety integrity levels. These represent magnitude levels of risk reduction. There are four SILs stated in the IEC 61508. SIL1 has the lowest level of risk reduction whereas SIL4 has the highest level of risk reduction. The IEC 61508 has two modes mentioned in it and they are differentiated as follows [5]:

1. Low demand mode – where the frequency of demands for safety-related system operations are not greater than twice the proof test frequency. Proof test frequency tells about the times the system is tested and assured so as to be completely operational.

2. High demand or continuous mode – where the frequency of demands for the operation of safety-related systems is greater than twice the proof test frequency.

Safety Integrity Level	Probability of failure on demand, average (Low Demand mode of operation)	Risk Reduction Factor
SIL 4	$>=10^{-5}$ to $<10^{-4}$	100000 to 10000
SIL 3	$>=10^{-4}$ to $<10^{-3}$	10000 to 1000
SIL 2	$>=10^{-3}$ to $<10^{-2}$	1000 to 100
SIL 1	$>=10^{-2}$ to $<10^{-1}$	100 to 10

Figure 2. 2 Safety integrity levels for low demand mode

Figure 2.2 depicts the SILs for the low demand mode. It helps in deciding a SIL depending upon the average of the probability of failure over the complete operational time of the product in an year along with the factor dedicated for risk reduction. Figure 2.3 depicts the SILs for the continuous mode. It helps in deciding a SIL depending upon the probability of failure as per an hour.

Safety Integrity Level	Probability of dangerous failure per hour (Continuous mode of operation)
SIL 4	$>=10^{-9}$ to $<10^{-8}$
SIL 3	$>=10^{-8}$ to $<10^{-7}$
SIL 2	$>=10^{-7}$ to $<10^{-6}$
SIL 1	$>=10^{-6}$ to $<10^{-5}$

Figure 2. 3 Safety integrity levels for continuous mode

It may seem that the continuous mode is by far more stringent compared to the low demand mode, but the continuous mode is calculated per hour. On the other hand the low demand mode assumes a time interval of roundabout one year/definition. Although the modes are approximately the same in terms of safety metrics. So basically, the aim for functional safety is solved by properly designing a Safety Instrumented System (SIS) to execute a Safety Instrumented Function (SIF) which has a dedicated SIL.

2.1.3 Utilization

The IEC 61508 standard states: “To conform to this standard it shall be demonstrated that the requirements have been satisfied to the required criteria specified and therefore, for each clause or sub-clause, all the objectives have been met.” [5].

During actual application of this standard, the proof of compliance often demands listing all the IEC 61508 requirements along with an explanation of how each of them has been achieved. This sticks to both, products developed to meet IEC 61508 and specific application projects that would want to adhere themselves according to this standard.

As IEC 61508 is just a standard and not a law, adhering to its compliance is not always accompanied with a legal document. However, compliance in best practices is cited in most liability cases, for e.g. in contracts. The global acceptance of this standard led to many countries hard-wiring this standard directly into their safety codes. Moving forward with the goal of safety there are ample industry and government contracts for safety equipment, systems, and services that purposefully require specific compliance with the IEC 61508 standard.

2.2 ISO 26262

The safety standard accepted widely in the automotive domain is provided by the International Organization for Standardization (ISO) which is the ISO 26262 functional safety standard. In general, ISO 26262 talks about:

- Provides an automotive safety lifecycle through the stages of management, development, production, operation, service until decommissioning. It also supports tailoring these steps according to the requirements during the lifecycle

- Provides an automotive specific risk-based idea to classify risks. This is termed under Automotive Safety Integrity Levels

- The ASIL levels are used for further specifying necessary safety requirements of a product so as to attain the acceptable residual risk

- It also provides requirements for validation, verification and confirmation measures to assure an acceptable and sufficient amount of safety is being reached

2.2.1 Introduction

Although it is not stated under this standard but it is a good practice to first determine the gap between the standard and the status of the current project. This leads to a clearer picture that lies between the existing processes and products with the requirements of the standard. The ISO 26262 is so well built that its stages of safety life cycle are very successful in identifying and assessing hazards/safety risks. They then establish safety specific requirements to minimize those risks to acceptable levels. This also aids the product lifecycle in the future as it becomes easy to manage and track those safety specific requirements.

2.2.2 The standard

The ISO 26262 functional safety standard talks about certain stages of the respective product on which it is applied. Amongst these stages, the first stage is termed as definition of the product under consideration. This involves identifying the required safety functions. Followed by this a risk assessment process is carried out, which is in accordance with the principles of risk assessment. This aids to differentiate between the safety and non-safety relevant requirements.

Thereafter, a safety goal is assigned for every hazard. Hence, the outcome of the risk assessment is one of the core values of the ISO 26262 standard and that is the ASIL level. They are a parameter that defines the potential risk and the methods required to solve them. The ASIL level is calculated on a combination of the probable severity and the probability of controllability with exposure.

The ASIL is differentiated into four levels, starting from ASIL A to ASIL D, where A to D gives ascending order of risk involved. The ASIL being a core component of the ISO 26262 standard has to be determined at the beginning of the product development process for each and every item. Figure 2.4 [6] gives an abstract view of all the different parts of the ISO 26262 and their coherence with the V-model.

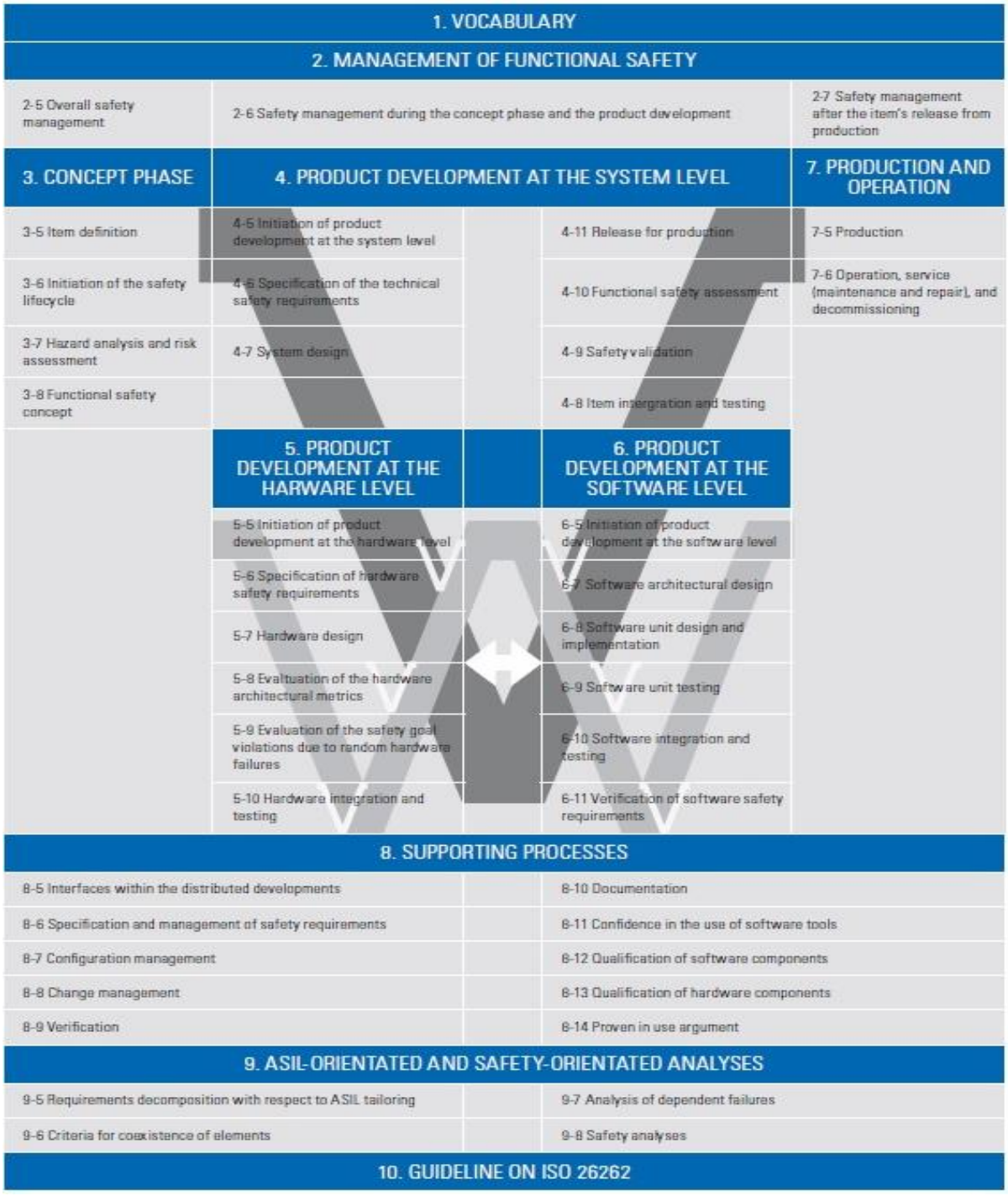


Figure 2. 4 Overview of the structure of ISO 26262

For failures from underlying hardware and systematic faults, the next theoretical stage lies to define the safety requirements. These must also be implemented by the appropriate ASIL level along with suitable methods and processes. Which is carried out by first, defining a functional safety concept that is supported by the technical safety concept. This is further

assisted with the safety architecture that is responsible for the software and hardware safety requirements.

To remove any persistent gaps, a plan also needs to be formulated. This plan should also include all the documentation that directs towards possible future modifications in case there are some defects with the current system. This is a smart step because of the fact that if at all these measures are to be carried in the future; they turn out to be time-consuming and costly. Lastly, verifying the system in accordance to its ASIL level is also a vital stage. This can be achieved by the initial analysis of the concerned safety functions followed by running tests on the same [6].

2.2.3 Utilization

So, all-in-all to reduce the risk in the automotive domain, it is vital to implement the ISO 26262 functional safety standard. The central and most important value of this standard lies with risk-based assessment and to implement the same so as to reduce the residual risk. There are certain issues present in the ISO 26262 functional safety standard that are sublimely new to the automotive domain, for instance, the qualification of software tools. Functional safety assessments and audits are performed to ensure that the desired functional safety of the product is achieved. The outcomes of these are thoroughly checked. To make sure that the systems tolerate faults and that they stay within safe states, diagnostics and monitoring functions are used. This leaps to the final goal of a functionally safe system [7].

2.3 AUTOSAR 3.2

This section introduces AUTOSAR and its different layers. The agenda behind this is to understand the core goal of introducing this platform. A need to organize complex systems in the rapidly growing automotive industry sparked the idea for this platform. This led the leading automotive industries to come together to form a common platform. This platform ensured the easier handling of increasing level of complexity. Keeping all the discomforts caused by the current development model, the roots of AUTOSAR were laid. An international partnership of automotive manufacturers, suppliers of tools/software and the firms involved with electronics and semi-conductor came together to put the AUTOSAR platform for the sole purpose of common growth.

2.3.1 Introduction

As the complexity of vehicles is increasing ever other day, managing this complexity becomes a challenge as numerous Electronic Control Units (ECU) are part of these cars. Alongside, scalability also adds up to the challenge. Whenever there are any modifications in the existing system the corresponding implementation becomes quite an issue. So, in order to solve all the mentioned issues and to compete with the aim of reliability, the introduction of a common method was needed so as to improve the quality of these systems.

And AUTOSAR seemed to be the next logical step towards all those issues. This helps right form the component to the system level. Another major aspect of AUTOSAR is its modularity, which enables independent and separate work on different parts of the software by engineers. The next big thing targeted under AUTOSAR is scalability. Next up, was transferability of smaller systems in order to make much more complex systems out of them. Lastly, the issue of re-usability is also solved under it.

The whole AUTOSAR consortium consists of different types of member. These are basically divided into five categories. These categories are as follows:

- The core members are the founding and the most important member. They consist of nine companies namely; BMW Group, PSA Peugeot Citroën, Continental, Robert Bosch, Daimler, Toyota, Ford, Volkswagen and General Motors.

- Then there are the premium members. They are 47 in total and some of them are TATA, Renault, Infineon, iAV, etc.

- The associate members are a total of 115 of them. They consist of companies like Alps, Caterpillar, CISCO, HCL, Infosys, Nissan, etc.

- There are some development members as well. They are 31 of them namely: Gliwa GmbH, iSYSTEM AG, ESR Labs, Timing-Architects Embedded Systems GmbH.

- There are a certain attendees that support AUTOSAR. There are 16 of them namely; Technische Universität Braunschweig, Universität Duisburg-Essen, Universität Paderborn, Budapest University of Technology and Economics, etc.

2.3.2 The architecture

The basic architecture of AUTOSAR is mainly divided into 4 main layers. The topmost layer where all the software components (SWC) lie is called as the Application Layer. These software components can be re-used and modularized with ease. Lower to the application layer lies the Run Time Environment (RTE). This allows the SWCs to interact with the target platform. Below the RTE lies the Basic Software (BSW) Layer. It consists of the Operating System (OS), ECU Abstraction, Micro-Controller Abstraction and Complex Device Drivers. They aid the SWCs to interact with the hardware usually a micro-controller that lies just below the BSW layer which can be clearly seen in figure 2.5 [8].

AUTOSAR also consists of some key features like modularity and re-configurability. Each of these are separate modules and can also be configured individually. This can be achieved via a standardized interface. As it is vital so a standard platform is thus needed. And to implement the SWCs communication with the hardware through the BSW, the RTE is provided so that everything runs in a synchronized manner.

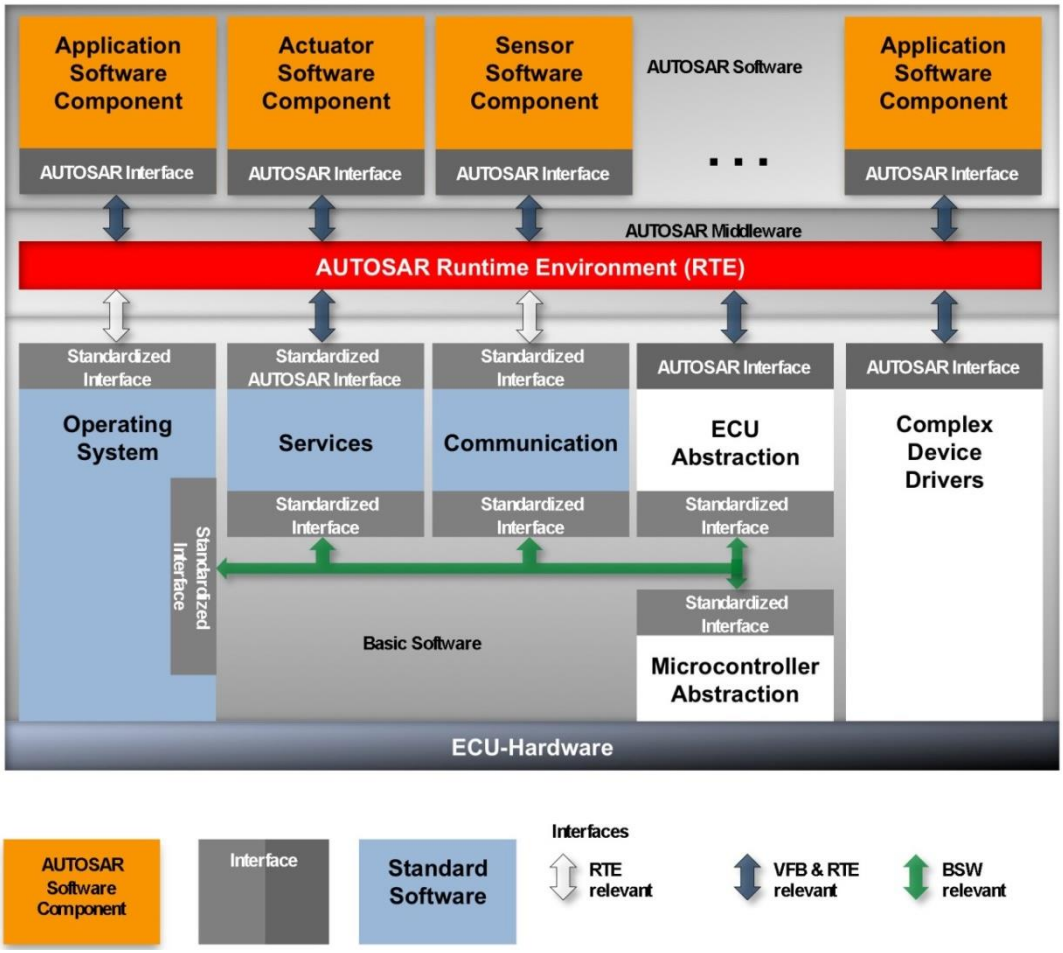


Figure 2. 5 AUTOSAR architecture

As from figure 2.6 [9], it can be seen that AUTOSAR takes maximum advantage from a layered software architecture structure. The SWCs are of various kinds and can be implemented according to the need of the project it is been applied into.

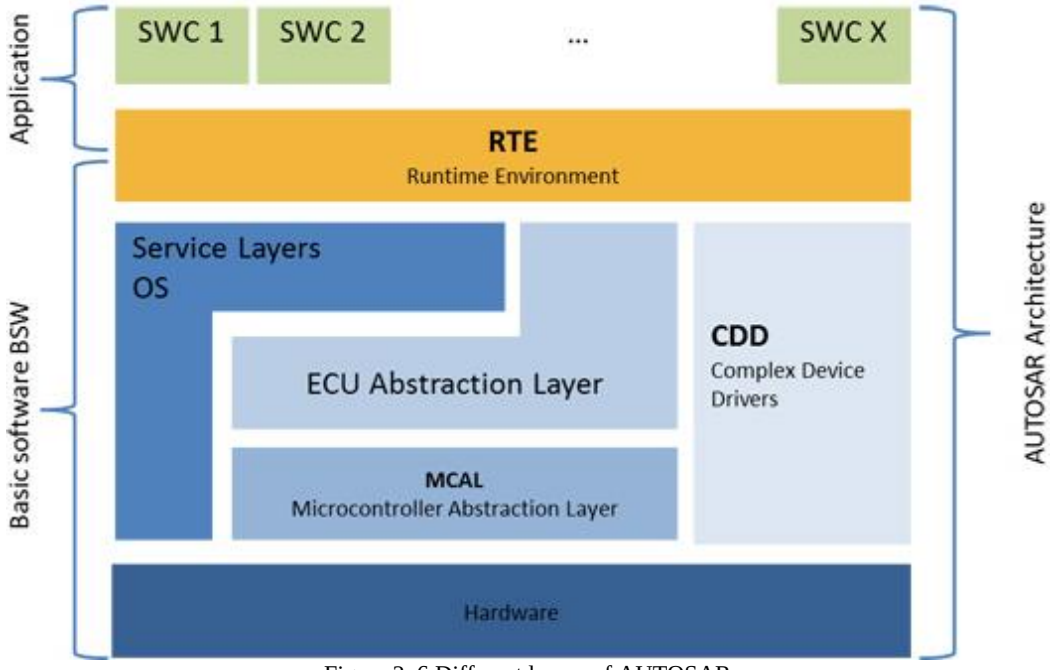


Figure 2. 6 Different layers of AUTOSAR

2.3.3 Utilization

The technical approach to develop a system in AUTOSAR is termed as AUTOSAR methodology. This methodology is a work-product flow with dependencies on past work results. The first step involves designing the architecture of the system under development. This translates into choosing the hardware and the software components along with system restrictions. This step is named as ‘System Configuration’. Which is followed by the activity of ‘Configure System’. This results into mapping of the components by their respective timings and resources. The outcome is termed as ‘System Configuration Description’. This contains the complete system information like topology, bus-mapping and mapping of SWCs. The next steps are processed for each ECU separately. This step is termed as ‘Extract ECU-Specific Information’, which as the name suggests extracts the information from the system configuration description. This information is needed by the ECU. And is then stored in an 'ECU Extract of System Configuration’. This is followed by the step known as ‘Configure ECU’ that feeds all the important information like the basic software modules, task scheduling, configuration of the basic software and allocation from runnables to tasks to the ECU. This information is stored in an ‘ECU Configuration Description’. The last step of this methodology is known as ‘Generate Executable’. This is a flashable file that contains the RTE code, the basic software and the SWCs. It is generated from the formerly generated ‘ECU Configuration Description’. All of these processes can be clearly seen in figure 2.7 [10].

The basic rule behind any AUTOSAR-compliant software development is to develop the SWCs according to the specifications of the ECU it will run on. Then the BSW and other modules are integrated and configured corresponding to the requirements of the SWC. Once this step is repeated for all SWCs, the RTE is generated. The generation of the RTE is done with respect to the interfaces of the SWCs and the basic software modules that communicate with the RTE.

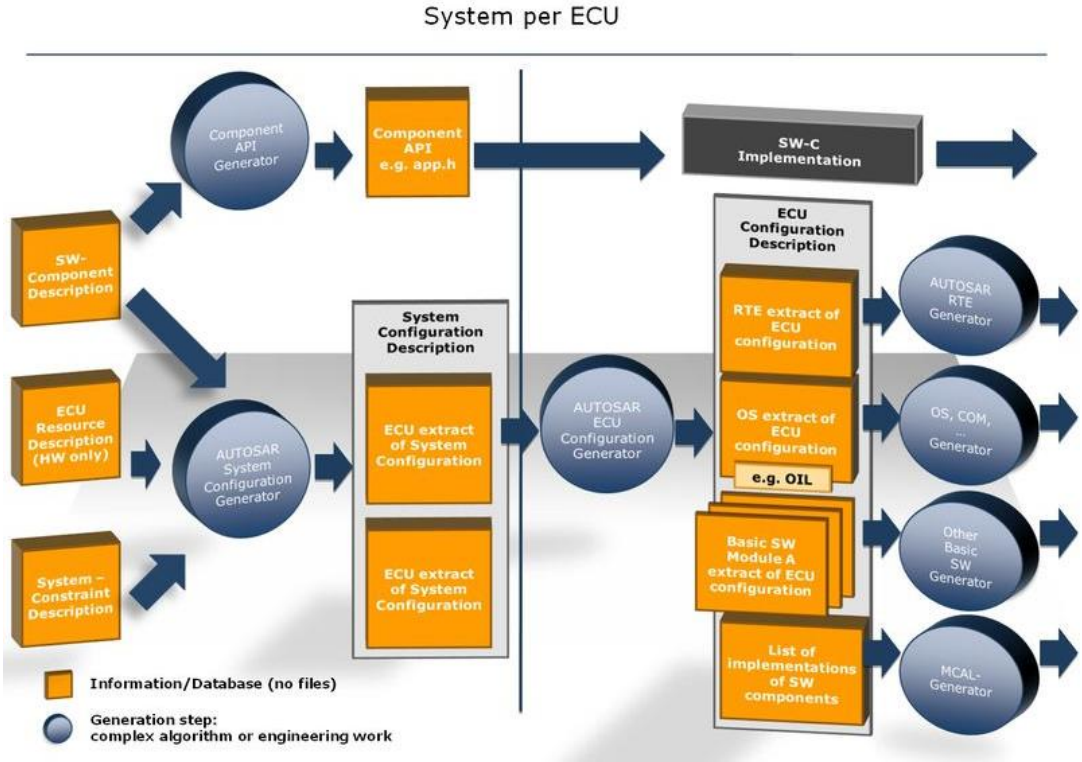


Figure 2. 7 AUTOSAR methodology

The application layer of AUTOSAR consists of atomic software component. These are the absolute primitive SWCs of AUTOSAR. For all SWCs the data elements should be specified. These are based on the requirements coming from the infrastructure of the system. Along with this, the list of resources required by the SWCs should also be provided. This information can be clubbed together as one term and is termed as ‘software component template’ under the AUTOSAR nomenclature.

The development of application for AUTOSAR can basically be divided into two methods. The first one being, Model-based development. This method usually generates robust code but which is not easy to read. The final version consists of an Extensible Markup Language (XML) file, this consists of all the SWC description whereas the code is generated in *.c and *.h format. The second method of development is the "Classic way". This involves development of the code by hand. The output files achieved from this type of software development is same as compared to the first method and the files are of *.xml, *.c and *.h format.

2.4 MISRA C 2012

Motor Industry Software Reliability Association (MISRA) is an organization that aids with guidelines for the software development of electronic components used in the automotive industry. It is an alliance formed between vehicle manufacturers, component and tool suppliers. For the creation of safe and reliable software, this organization works towards it by providing important advice to the automotive industry. As the safety requirements used in automobiles is different from that of areas such as healthcare, industrial automation, aerospace, etc. The core mission of MISRA is "To provide assistance to the automotive industry in the application and creation within vehicle systems of safe and reliable software”. The MISRA was created so as to be in par with the UK Safety Critical Systems Research Programme.

2.4.1 Introduction

MISRA C: 2012 that is used in this thesis supports the C99 version of the C language. It consists of a number of improvements that reduce the complexity and cost of compliance, along with the consistent and safe use of C. The MISRA C: 2012 has higher granularity of rules so as to incorporate precise control, rationales for every guideline stated under it and identified decidability to better interpret the output of checking tools. This further aids the user by giving a number of expanded examples along with a cross reference for ISO 26262.

2.4.2 MISRA 2012 guidelines

The guidelines mentioned under the MISRA 2012 states that whenever a new software project is started, it should use the latest version of the MISRA standard. With the introduction of the latest MISRA C: 2012 a complete new category of guidelines are introduced. These are much more readable, understandable and relates to the procedural matters. It can be used by a wide

spectrum of people from quality assurance (QA) executives to software developer as seen in figure 2.8 [11]. The MISRA guidelines can basically be divided into the following categories:

- Classification – Every guideline stated in it is termed as either; Mandatory, Required or Advisory. They can also be further classified as 'Decidable' or 'Undecidable'. These are expressed as the following *[12]*:

  - Mandatory guidelines should always be complied
  - Required guidelines scan be complied until and unless they exhibit a 'Deviation'
  - Advisory guidelines can be opted out

- Categorization - MISRA C: 2012 rules can be divided into the following categories:

  - Making sure to avoid possible compiler differences
  - Strict use of constructs and functions that may lead to failure
  - Write readable and debuggable code
  - Staying within the complexity limits

- Compliance – So as to claim compliance with MISARA C: 2012 all mandatory rules should be adhered to whereas all required rules should either be implemented or a formal deviation is to be provided along with it. Although, advisory rules can be considered to be ruled out and that too without a formal deviation. But this needs to be mentioned by annotations in the code files.

Figure 2. 8 Target audience of MISRA C

2.4.3 Utilization

The guidelines listed under the MISRA C: 2012 was initially a fruit from collaboration between OEMs, component suppliers and engineering consultancies. The sole aim for this was to promote best practices while developing safety-related specific E/E systems for road vehicles. This is a never ending process and MISRA still provides with information for management executives and engineers. Certain events are also held to share the common best practices currently being used in the industry.

However, due to its proven ethics and global acceptance, MISRA C has recently being used across a lot of industries starting right from military, aerospace, rail and medical sectors. Adding to this cause there are ample number of tools out there that further assist to support the enforcement of the MISRA C: 2012 guidelines.

2.5 Tools and technologies

This section discusses various tools and technologies that were studied and surveyed which might be useful for the implementation of the concept formulated under this master thesis. Each tool and technology is understood from the point of present research work and how can the same be utilized. The explanation to the usage of each is covered under their respective topics.

2.5.1 Tools

1. Enterprise Architect - It is a tool used to manage information as it has substantial capabilities and can be innovatively used in various complex processes and projects. It offers modelling at a high level which is easy to implement and understand as well.

Enterprise Architect can be used by a wide variety of people from the industry, right from business analysts, programmers, testers, etc. It can handle anything from small scale to large repositories based projects and also Cloud based processes due to its commanding standards and efficient scaling techniques. Enterprise Architect also provides End-to-End Traceability as one of its key features. It gives the user traceability over all the important aspects of the industry, ranging from requirements and analysis to designing of models. Thus, it helps all throughout the phases of implementation and deployment.

During the product lifecycle phases such as verification, validation and analysis can be considered through Enterprise Architects features called as Hierarchy View and Relationship Matrix. Due to the efficient resource allocation in this tool it gives the ability to QA teams and Project managers to deliver the processes or projects successfully. Enterprise Architect very well handles the requirements of the given project as this was the reason to use this tool for this master thesis. And modifications can also be taken care of by Enterprise Architect's Impact Analysis feature and to further build the system. Its requirements management features can be further used for the following reasons [13]:

- For an organized and hierarchical requirements model
- To trace the system requirements from implementation to model elements
- Report or search the requirements

2. Understand C - It is a metrics tool used for source code analysis. Some of the features it handles with utmost efficiency are; cross-platform and multi-language development

environment. It also supports the following list of languages; C, C++, C#, Python, VHDL, XML, Ada, JAVA, etc. The below mentioned are its other strong points [14]:

- Editor – It has one of the most powerful modern-day editor with a modern GUI. It is designed in a way so that it can be used with a multi-monitor setup and for the ease of use it has auto-completion, syntax colorization, etc.

- Code knowledge – It gives you all the information about the code at a single glance. This includes information about different classes, functions, variables, etc. along with their usage, calling mechanism and interaction behaviour. The user could also view the different references, trees, control flow graphs, call graphs, metrics and likewise information in perspective of the code.

- Standards testing - Understand also allows the user to check the code. These checks usually include verifying metric requirements, guidelines, best practices or any other project-specific conventions.

3. PRQA·C – It is used by engineers for the development of high integrity code. It can be used to not only prevent and detect defects but also to ensure the code's compliance towards coding standards. It is a easy-to use and non-disruptive tool. It also detects and then reports issues such as: software defects, language implementation errors, dataflow problems, inconsistencies, coding standard violations and dangerous usage of semantics. And these were the reasons to use this tool for this thesis to check the code against coding standards. Some highlights of this tool are as follows [15]:

- Undefined Behaviour - Many constructs are present in the C language that are not explicitly incorrect but may result in unpredictable behaviour. Similar issues ranging from the most familiar to the less well known ones are stated in the ISO standards and QA·C identifies all such problems.

- Constraint Violations – The misuse of the language is something that needs to be taken care about while writing or designing the code. This tool takes care of all such parameters and warns the user about them once identified. It also gives possible solutions with examples on how to approach their solutions.

- Redundancy - QA·C is a smart tool that recognizes the code that is never used or the code that does nothing. It also recognizes unused functions, variables and parameters. Along with detection of unreachable code, redundant assignments and initializations, dead code and redundant operations.

- Coding Standard Rules - QA·C also warns the user about possibly unwise or dangerous usage of the designated language.

4. iSYSTEM winIDEA - winIDEA is iSYSTEM's Integrated Development Environment. This consists of all the tools necessary for embedded software development and testing. It has a project manager, C/C++ source code editor and a high level source code

debugger. It also has support for scripting languages and helps in automation and testing. Other features in this tool are namely [16]:

- Real-time analyser
- Debugging for multi-cores
- Support for various Operating Systems
- Has GNU compilers for ARM, Power Architecture

5. Vector DaVinci Configurator – This is a tool used for configuring, validating and generating the BSW and RTE of an AUTOSAR complaint ECU. Some of its main features include [17]:

- Specific configuration interfaces between the BSW and the RTE
- Controlled overwriting of parameters to correct errors in the system description
- Generation of an HTML report regarding parameters which deviate from the system description
- For re-working on descriptions in ARXML file

6. Vector CANoe – It is a tool for development, analysis and testing of either individual ECUs or an entire ECU network. It does so at all the different phases of the product lifecycle. It helps in the analysis of multi-bus communication of the entire system and the communication data base can be manually or automatically simulated. Because of these points this tool was used in this thesis. It also has the below mentioned features [18]:

- One tool for all development and testing related tasks
- Testing and simulating ECU diagnostics
- Detection and correction of errors during early phases of the development process

CANoe provides various ways to stimulate ECUs. It supports KWP2000 and UDS standards as well. This tool greatly helps in ECU tests, module tests, integration tests, etc.

2.5.2 Technologies

1. C – It is a procedural language. It can be compiled by a very straightforward compiler. This is done so as to provide low-level access to memory and language constructs that adhere to the machine instructions. This also takes the minimum run-time support. As, C became a much more accepted language it was coded in assembly language, for application in system programming. And hence, for the purpose of this master thesis C99 is used.

With the high-level capabilities this language has it can basically do just about everything but it was basically designed for cross-platform programming. A standards C program can be compiled on various compatible compilers with ample number of computer platforms and OS along with minimalistic change in it.

As mentioned about the "system programming" approach of this language, which also talks about it use in OS and embedded system applications. As this language is the best combination of two worlds, namely; code portability and efficiency and with its ability to access hardware has further worked in its favour.

C is sometimes used as an intermediate language by implementations of other languages. This approach may be used for portability or convenience; by using C as an intermediate language, it is not necessary to develop machine-specific code generators. C has some features, such as line-number pre-processor directives and optional superfluous commas at the end of initializer lists, which support compilation of generated code.

2. UML - The Unified Modelling Language (UML) is a language in the field of software engineering used for developmental and modelling. This gives a very standard way to visualize the system under consideration. It consists of various types of diagrams but are mainly classified into two basic categories.

While at one end it has types that talk about structural information and on the other end it has the general types of behaviour. It also includes a few that have various other aspects of interactions. These diagrams can be further categorized in the following hierarchical list of class diagrams:

- Structure diagrams – It talks about the objects that need to be present in the system under consideration. As they mainly show the structure of the system being modelled, it is widely used for documentation of the software architecture. For instance, the component diagram represents all the components and their respective dependencies in the model being developed.

- Behaviour diagrams – As the name suggests, these diagrams talk about the behaviour of the system being modelled. As they talk about how the system functions, they are widely used to understand the functionality of system. For instance, the activity diagram represents the step-by-step activities of the components that need to be followed in order to design the system.

- Interaction diagrams – These being a subset of the behaviour diagrams, lay stress on the control flow and the sharing of data in the system being modelled. For instance, the sequence diagram represents the way different objects communicate amongst each other via sequence of messages.

3. Conceptualization

This chapter ponders upon the different steps involved for building up the concept for this thesis. The steps are as follows:

- State of the art
- Compendium of research question
- Approach towards research question
- ISO 26262 Part 1: Vocabulary
- ISO 26262 Part 3: Concept Phase
- ISO 26262 Part 4: Product development at the system level
- ISO 26262 Part 6: Product development at the software level
- ISO 26262 Part 9: Automotive Safety Integrity Level (ASIL)-oriented and safety-oriented analysis

3.1 Concept

This particular section reads about the current scenario that prevails in the automotive domain when it comes to the question of safety. Leading into the mould into which this thesis tends to take direction and finally the approach that is initiated so as to come in par with the aim of this thesis.

3.1.1 State of the art

The usual trend that has been dominating the present in the automotive domain in terms of safety speaks volumes for itself. It is important to first understand the current process and then find a way to change it. This involves digging into current ongoing processes and gaining

experience about them through rigorous steps. This can be very tricky at times because of the complexity with which these processes are implemented and sometimes also due to the lack of proper documentation for the ISO 26262 functional safety standard. The difficulty takes a whole new level when the perspective in which the reader interprets this standard varies miles from what is actually intended. But summing down to the below listed points, it can aid in order to build the structure of the state of the art. These points will not only help in realizing and analysing the current scenario but also to decide the areas of serious concern. Once these are selected, they can easily be targeted so as to find their solutions. Research on such a structured model is easier and the solutions can also be evaluated directly in competition with its respective concern areas. The problematic areas are as mentioned below:

- The foremost issue lies when the functional safety standards are usually layered upon top of the basic functionalities. This is an issue that leads to conflicts. The layering of these functionalities is firstly, not advisable by the ISO 26262 and secondly, hampers the product lifecycle adversely. The product lifecycle needs to be shifted on a prior timeline just so that the safety features can be implemented on it at later stages. This is not only bad for the management stage but also for the system and testing stages cohesively.

- When it comes to the software phase, the above mentioned issue causes major problems. Once the software is laid upon with the new safety features this not only negatively affects the optimizability of the code but also greatly hampers the readability of it. This clearly sums up to be a costly affair at the end when the process is taken as a whole.

- The next issue lies when the software architects design the functionalities on an abstract level with which the first prototype is developed. Once this is done then the functional safety standards are applied to it which adversely affects the reusability of the software code. There are streams of life where reusability plays a major role and the reusability of the software code is something that is not hidden from anyone. The importance of reusable code is known and appreciated by software minds all over the globe and when this is hampered, it becomes a serious concern on its own.

- The final problem that is targeted in this thesis lies with the after effects once the functional safety standards are applied to a particular ECU. The ECU adheres to the functional safety standards set for its own functionalities and not for the standards set for the complete system. So even though the particular ECU is in order with the functional safety standard the whole system to which this ECU is a subset, does not adhere to the functional safety standards.

3.1.2 Compendium of research question

The scientific challenge that lies in this thesis is contemplating the ISO 26262 functional safety standard. This involves going through the different parts of the ISO 26262. Once the different parts are analysed thoroughly, then their understanding as a whole comes into the picture. The ISO 26262 being such a huge, detailed document doesn't make it any easier. The complexity mainly lies in the interdependencies between different parts and how can they be sufficiently but orderly combined.

After the in-depth analyses of the standard to the point where it can actually be applied to a sample system, the ASIL levels need to be designated. The ASIL levels need utmost attention because they govern the present and future of the system till the end of its product lifecycle. The ASIL levels are very well defined in the ISO 26262 but implementing them after a proper understanding is a very complex aim at hand that is being addressed in this thesis.

The system that is being used in this thesis as a sample will comply with the ISO 26262 standard with the most part of it. This would comprise right from the designing, development to the verification phases. For instance, this translates into writing software requirements for the sole aim of system development so as to better reflect requirements engineering and many other aspects too which will be clearly defined in the following sections of this chapter. All the phases will be deliberately covered so as to avoid any conflicts in the future thus, aiming directly at one of the key problems that we face in our present processes. Followed by implementation and integration of a subset of the sample system on an AUTOSAR compliant ECU is carried out. And finally, verifying the software safety requirements by software testing to prove its conformity to functional safety will lead us into the last chapter of this thesis.

3.1.3 Approach towards research question

The direction which is taken by this thesis drives into deriving the basic structured approach so that the ISO 26262 functional safety standard can be understood with utmost depth and to ease its implementation process into real life processes. This is done initially so that all the parts that comprise of the ISO 26262 functional safety standard can be easily taken into account and be applied to a sample system.

This sample system that is being considered in this thesis is just for a better understanding and simultaneous implementation of the ISO 26262. The system does not completely touch all the sections of all the parts of the ISO 26262 but only the major ones. The parts that are excluded are because of the blown out complexity that it will lead into and also because they are not necessary for the scope of this thesis. Although they were analysed and are well spoken about in this thesis. The different functionalities and non-functionalities of the sample system will be specified with due respect to ISO 26262. The hazardous events of the system will be taken into account to complete the aim for functional safety. This will lead to the specification of software functional safety requirements and software architectural design.

They are then conceptualized in order to build an architecture that adheres to a specified ASIL level which is the main challenge of this thesis. The designation of the ASIL levels becomes challenging as the ASIL level is given to the entire system. Once these are specified then a major important aspect listed under the ISO 26262 comes into the picture and that is regarding ASIL decomposition. This is a tricky but yet very essential because of the fact that this leads the development of any product following into its lifecycle. So, understanding it completely keeping in mind its effects on other parts of the ISO 26262 is alarmingly important. This then leads to the implementation part of the ISO 26262 with respect to the software development of the product. The sample system is then implemented on an AUTOSAR compliant ECU. The AUTOSAR compliant ECU for the purpose of this thesis was borrowed from an existing ongoing project. As the hardware aspect of the ISO 26262

functional safety standard has been ruled out for the purpose of this thesis. Hence, a subset of the sample system is implemented on the AUTOSAR compliant ECU. And in conclusion verifying and validating the actually implemented requirements of the sample system to test its obedience towards functional safety. This covers the final part of the ISO 26262 that is aimed under this thesis title which is testing.

3.2 ISO 26262 Part 1: Vocabulary

This section will talk about the Part 1 listed under the ISO 26262 functional safety standard which consists of all the terminologies that are used in all the remaining parts of the ISO 26262. This section involves understanding all the terms and definitions listed under it. Followed by a list of all the abbreviated terms used in the ISO 26262 functional safety standard.

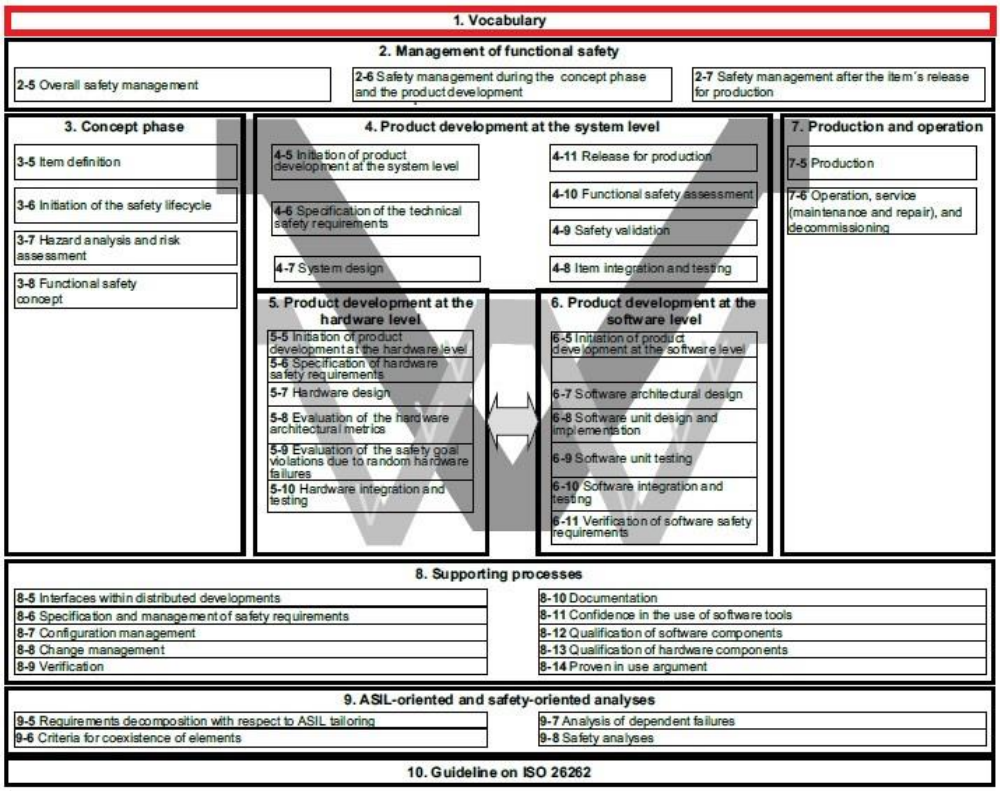


Figure 3. 1 ISO 26262 Part 1: Vocabulary

3.2.1 Scope

The ISO 26262 functional safety standard is designed for the safety related application of electrical and/or electronic (E/E) systems which are installed in passenger cars. These cars can have a maximum gross vehicle mass of up to 3,500 kg. Although the ISO 26262 standard does not address any E/E systems that are unique to vehicles, for instance vehicles designed for people with disabilities.

Systems that are being produced or already under development while the ISO 26262 standard was released, are exempted from this standard. Although for the same systems, their modifications if any should be in compliance with the ISO 26262 standard [19].

The ISO 26262 standard looks upon all the hazards possibly caused by a defect in the behaviour of E/E systems. But it does not specify hazards such as radiation, reactivity, corrosion, toxicity, electric shock, heat, fire, smoke and flammability. So, this is the part of ISO 26262 that specifies the definitions and abbreviated terms that are used in all the following parts of this standard.

3.2.2 Terms and definitions

Keeping in sight the scope of this thesis, some important and essential terms and definitions are listed from the ISO 26262 Part 1: Vocabulary. They are as follows [19]:

- Architecture – It is the representation of the structure of an item which allows the segregation of building blocks and interfaces involved.

- Automotive Safety Integrity Level (ASIL) – This consists of four levels so as to specify the item's safety measures. These are applied to avoid risk.

- ASIL decomposition – This involves bifurcating the safety requirements to independent elements. This is done to reduce the ASIL level of the corresponding elements.

- Cascading failure – This is defined when the failure of an element causes another element to fail.

Figure 3. 2 Cascading failure

- Common cause failure – This is defined when the failure of two or more elements that have a single root cause.

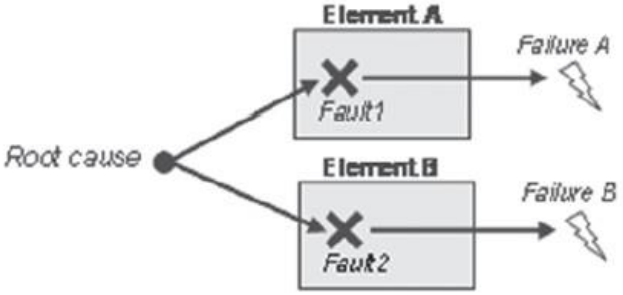


Figure 3. 3 Common cause failure

- Component – It is a level of element that can be logically and technically be separable as it comprises of a number of hardware part and software units.

- Electrical and/or electronic system – These are the systems which consist electrical and/or electronic elements. This may also include programmable electronic elements.

- Error – It is the difference that lies between an observed, computed or measured value against the specified, true or theoretically correct value.

- Failure – It is the inability of an element to execute the required function.

- Fault – these are vivid conditions that may cause an element to fail.

- Fault reaction time – It is the time period taken to detect a fault and to reach the desired safe state.

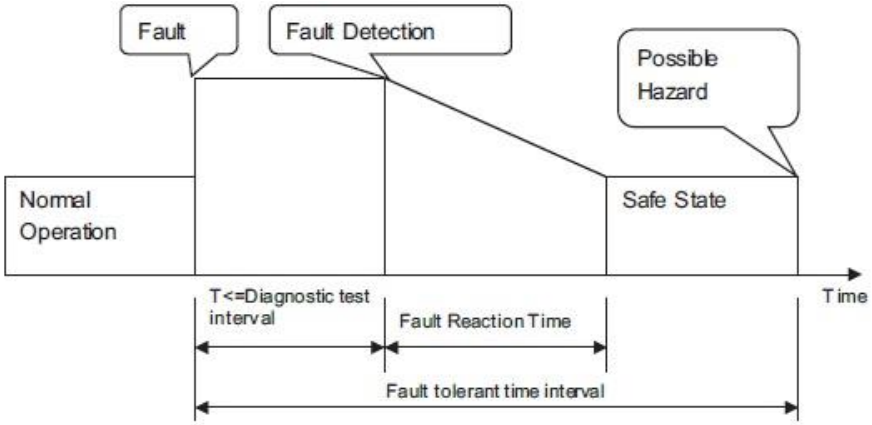


Figure 3. 4 Fault reaction time and fault tolerant time interval

- Functional safety concept – It is the detailed specification of the functional safety requirements. This consists of all the important information, allocation to architectural elements and their interaction. These all are highly necessary to achieve the safety goals.

- Functional safety requirement – It is the specification of implementation-independent safety behaviour/measure so as to achieve the safety goal.

- Hazard analysis and risk assessment (H&R) – It is the step that needs to be followed so as to recognize the hazardous events and then specify the safety goals. This is further extended by assigning an ASIL level to avoid any kind of risks.

- Item – It is the array of systems which is used to implement a function to which the ISO 26262 standard is applied.

3.2.3 Abbreviated terms

This sub-section consists of a list of important terms that are used in the remaining parts of the ISO 26262 functional safety standard. They are as follows [19]:

- ASIL - Automotive Safety Integrity Level
- CAN - Controller Area Network
- CRC - Cyclic Redundancy Check
- FIT - Failures In Time
- FMEA - Failure Mode and Effects Analysis
- FTA - Fault Tree Analysis
- HAZOP - HAZard and OPerability analysis
- H&R - Hazard analysis and Risk assessment
- MMU - Memory Management Unit
- MPU - Memory Protection Unit

3.3 ISO 26262 Part 3: Concept phase

This section describes all the fundamentals that are included in the Part 3 of the ISO 26262 functional safety standard.

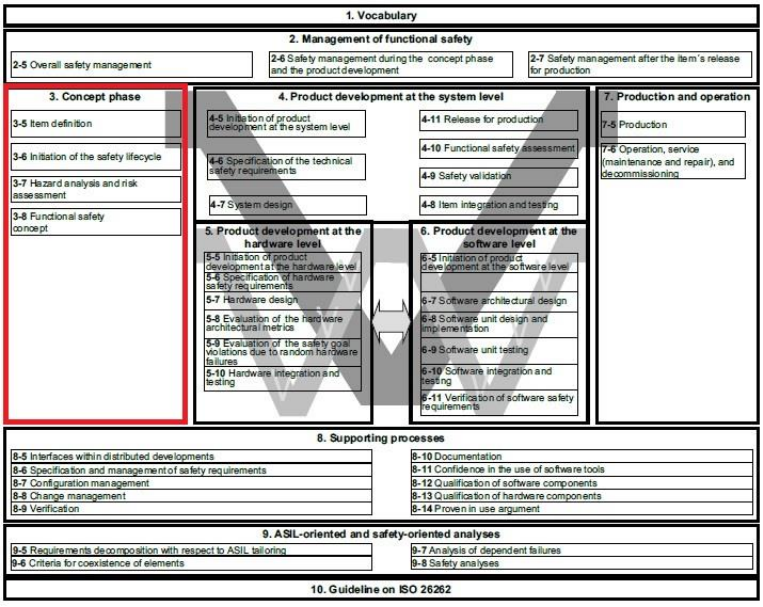


Figure 3. 5 ISO 26262 Part 3: Concept phase

Part 3 deals with all the steps that need to be taken so that the concept stage of the product lifecycle of any process adheres to the ISO 26262 functional safety standard. All the necessary sub-parts of this part along with their explanations are elaborated in this section of the thesis.

3.3.1 Scope

The requirements needed for the concept phase of an automotive application is specified under this part of the ISO 26262 and includes the following:

- Item definition
- Initiation of the safety lifecycle
- Hazard analysis and risk assessment
- Functional safety concept

The first and the foremost task of this sub-section is to first define the item followed by it dependencies and interaction with the environment. Thereafter the second objective lies in understanding of the item which will make the remaining sub-sections easier to implement. This results in the functional and non-functional requirements of the item. They can further be classified as safety-related once their respective ASIL are declared. This information has the following:

- The functional concept
- The environmental and operational constraints
- Behavioural assumptions of the item
- Consequences of hazards and failure modes.

The locality of the item along with its interfaces and interactions are defined considering the following [20]:

- The elements that make up the item
- Behavioural assumptions of the item over the environment
- Interaction of the item with other items/elements
- Functionality of/from other items/elements and the environment
- Allocation/distribution of functions of the item in the system
- Scenarios effecting the functionality of the item

Below mentioned are a few requirements that were used so as to build the sample system. These are important as they act as the building blocks for the remaining parts of the ISO 26262 functional safety standard. The requirements for the locking/unlocking mechanism of the door can be seen in figure 3.6.

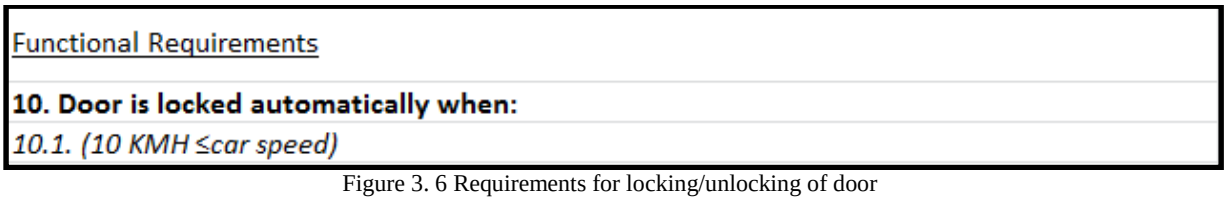


Figure 3. 6 Requirements for locking/unlocking of door

The requirements for the closing/opening mechanism of the window can be seen in figure 3.7.

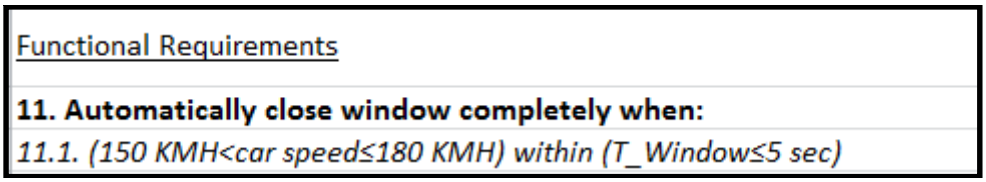


Figure 3. 7 Requirements for closing/opening of window

3.3.2 Initiation of the safety lifecycle

The first objective lies in making a differentiation between a new or a modification of the item. As the sample system under consideration of this thesis is a new system so the activities revolving around modifications of the item are ignored. Although, the activities in-case of a modification are analysed in this thesis. The second objective follows in its footsteps and tries to elaborate the necessary safety lifecycle activities in the case of a modification. For any modification cases, an impact analysis needs to be carried out. These modifications usually include design modifications and its respective implementations. The impact analysis gives information about all the affected areas from the modifications to the item including [20]:

- Operating modes
- Interfaces between the item and the environment
- Environmental conditions

The results from the modifications with respect to the functional safety should be described. But keeping in mind that these safety activities should take place in compliance with the different lifecycle phases. In the advent where there are missing work-products, an attempt to make those requirements with respect to the ISO 26262 standard have to be made.

3.3.3 Hazard analysis and risk assessment

The objective of the hazard analysis and risk assessment is to identify the hazards that may cause malfunctions in the item and falsify the safety goals. So basically, they are used to determine the ASIL levels so as to achieve the safety goals for the item in order to remove the advent of any risks. For this, the item's potential hazardous events are considered. The ASIL is calculated by its parameters; i.e. (severity X probability of exposure and controllability).

The next step is determining the situation analysis along with the hazard identification. The operating modes and the operational situations which lead the item to a hazardous event is described. The operational situation talks about the limits in which the item is in bounds to be safe. Whereas, the hazards are calculated by certain techniques. Techniques such as quality history, FMEA, brainstorming, etc. For the purpose of this these few of these techniques were used so as to determine an ASIL for the requirements of the sample system. Basically, each and every hazard has a variety of potential causes behind it and these need not be considered for H&R. The hazardous events are determined so as to get multiple relevant combinations of different hazards and operational situations. Then there consequences should be identified. If there are hazards which are outside of the scope of ISO 26262 then an attempt need to be made to control them wisely otherwise there classification is not at all necessary.

If the classification of a specific hazard in terms of severity, probability of exposure and controllability is becoming a difficult affair then always the higher ASIL level should be assigned. The severity can be assigned from one of the following severity classes, namely; S0, S1, S2 or S3 as in table 3.1 [20]. The risk assessment main focus is on the harm that each person is potentially at along with risks associated to pedestrians, cyclists, etc. The severity class is based upon the height to which the person is injured. The way to estimate these is via evaluating different situations based upon samples from target markets.

Table 3. 1 Classes of severity

	Class			
	S0	**S1**	**S2**	**S3**
Description	No injuries	Light and moderate injuries	Severe and life-threatening injuries (survival probable)	Life-threatening injuries (survival uncertain), fatal injuries

The severity class S0 is assigned whenever the damage is restricted only to materials and no persons are involved. And for this no ASIL assignment is required.

The probability of exposure is the next parameter for calculating the H&R. The probability classes are as follows; E0, E1, E2, E3 and E4 as in table 3.2 [20]. The exposure determination is based upon the probability of the likeliness of the hazardous event. The way to estimate these is via evaluating different situations based upon samples from target markets.

Table 3. 2 Classes of probability of exposure

	Class				
	E0	**E1**	**E2**	**E3**	**E4**
Description	Incredible	Very low probability	Low probability	Medium probability	High probability

Class E0 is used for extremely unusual situations. A rationale is to be provided for the exclusion of these situations. And in this very case no ASIL level is required.

The controllability of any hazardous event by the driver or any person is to be determined using a rationale. This then needs to be assigned to the controllability classes, namely; C0, C1, C2 and C3 as in table 3.3 [20]. The determination of the controllability is an idea that the driver or the persons probably involved in the risk are able to control and mitigate the same to their fullest capacity. Under this also all the reasonable foreseeable misuses are taken into consideration. In context where the hazardous event is not talking about the vehicle direction and speed, the controllability is judged by the fact that the person at risk is able to mitigate themselves from the hazardous situation.

Table 3. 3 Classes of controllability

	Class			
	C0	C1	C2	C3
Description	Controllable in general	Simply controllable	Normally controllable	Difficult to control or uncontrollable

Class C0 is for hazards that don't affect the safe operation of the vehicle. It can also be assigned in the case where there are dedicated regulations for a particular hazard for a particular item. And in the case of hazards with the C0 controllability need not be assigned any ASIL level.

The final step is determining the safety goals and ASIL levels. The ASIL level is determined by severity, probability of controllability and exposure as seen in table 3.4 [20]. There are four ASIL levels and are defined as: ASIL A, ASIL B, ASIL C and ASIL D. Where, ASIL A is the lowest and ASIL D is the highest safety integrity level. Along with these; quality management (QM) specifies no requirement to be implemented in compliance with ISO 26262 standard.

Table 3. 4 ASIL determination

Severity class	Probability class	Controllability class		
		C1	C2	C3
S1	E1	QM	QM	QM
	E2	QM	QM	QM
	E3	QM	QM	A
	E4	QM	A	B
S2	E1	QM	QM	QM
	E2	QM	QM	A
	E3	QM	A	B
	E4	A	B	C
S3	E1	QM	QM	A
	E2	QM	A	B
	E3	A	B	C
	E4	B	C	D

A safety goal is set for each hazardous event along with an ASIL level. Similar safety goals are combined into one goal. And the highest determined ASIL level is assigned to the respective safety goal. In cases where the safety goal can be reached upon by transitioning/maintaining to one or more safe states, then these safe state have to be defined and specified.

The ASIL level for the requirements of the sample system are mentioned below. The ASIL level along with its respective severity, exposure and controllability for the locking/unlocking mechanism of the door can be seen in figure 3.8.

Functional Requirements	Severity Class	Exposure Class	Controllability Class	ASIL Level
10. Door is locked automatically when:				
10.1. (10 KMH ≤car speed)	**S1**	**E4**	**C2**	**A**

Figure 3. 8 ASIL level determination of door

The ASIL level along with its respective severity, exposure and controllability for the opening/closing mechanism of the window can be seen in figure 3.9.

Functional Requirements	Severity Class	Probability Class	Controllability Class	ASIL Level
11. Automatically close window completely when:				
11.1. (150 KMH<car speed≤180 KMH) within (T_Window≤5 sec)	**S2**	**E3**	**C2**	**A**

Figure 3. 9 ASIL level determination of window

3.3.4 Functional safety concept

The core fundamental behind the functional safety concept is to extract the functional safety requirements. This is done from the safety goals and then assign them to different architectural elements of the item. The functional safety concept consists of parts that majorly focus on goal of achieving safety, namely; safety measures, safety mechanisms, etc. These are defined in the functional safety concept because they are then executed in the architecture of the item. The functional safety concept talks about the below mentioned points:

- Failure mitigation and fault detection
- Safe state transitioning
- Mechanisms for fault tolerance
- Detection of a fault and warning the driver

Post the H&R the safety goals are thought about as in figure 3.10 [20]. And then the functional safety requirements are derived from them.

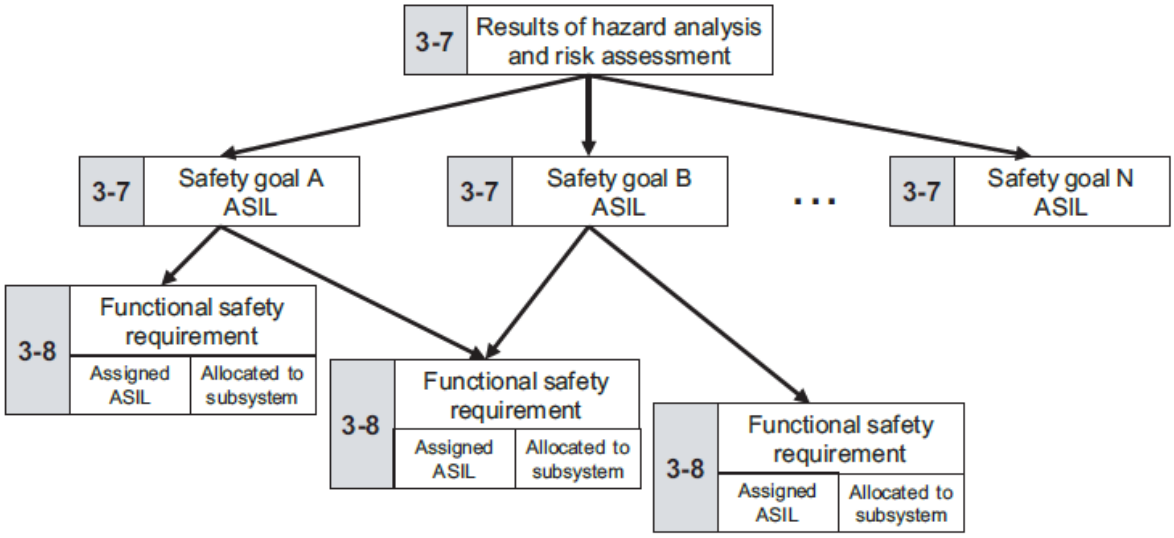


Figure 3. 10 Hierarchy of safety goals and functional safety requirements

The orientation of the safety requirements in prospect to different parts of the ISO 26262 are illustrated in figure 3.11 [20]. And the functional safety requirements are distributed amongst the elements of the architecture.

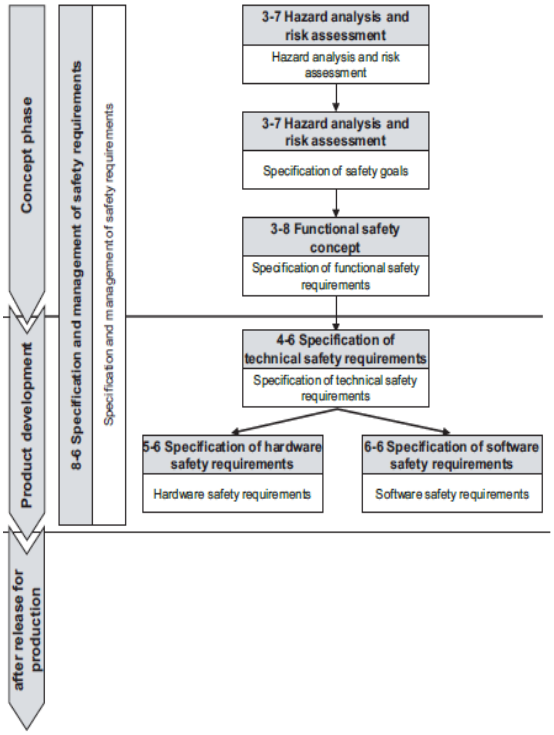


Figure 3. 11 Structure of the safety requirements

The allocations of the functional safety requirements of the elements to the architecture are carried out with the following assumptions:

- All the information regarding the ASIL level should be inherited from its corresponding safety goal

- If a system has a variety of items then their respective interfaces and functional safety requirements should be specified

If there are other technologies involved while considering the functional safety concept then the following should be noted:

- The elements from other technologies should be described and allocated in the architecture

- The elements from other technologies should have their respective functional safety requirements specified

All the extra measure taken into account for the functional safety concept are as follows:

- The external measures should be determined and communicated

- The interfaces determined from external measures should have their respective functional safety requirements specified

3.4 ISO 26262 Part 4: Product development at the system level

This section of the thesis deals with the steps involved in order to proceed with the product lifecycle at the system level.

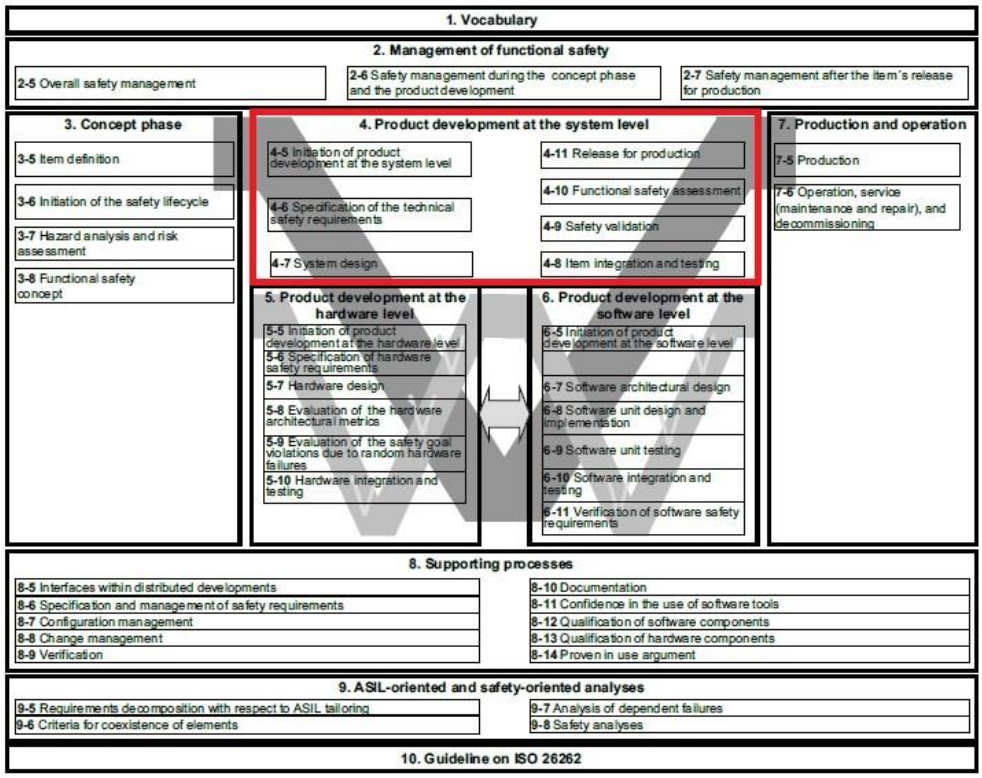


Figure 3. 12 ISO 26262 Part 4: Product development at the system level

This section will explain all the necessary steps mentioned in the Part 4 of the ISO 26262 functional safety standard about the system level development. This phase finds it place between the concept phase and the simultaneous phases of development at hardware and software. Thus, this is an important part of the ISO 26262 as the decisions made for the system for a particular process governs how the system behaves throughout the lifecycle of the product until it is decommissioned.

3.4.1 Scope

The requirements needed for the development of the product at the system level of an automotive application is specified under this part of the ISO 26262 and includes the following [21]:

- Initiation of product development at the system level
- Specification of technical safety requirements
- Technical safety concept
- System design
- Item integration and testing
- Safety validation
- Functional safety assessment
- Product release

3.4.2 Specification of technical safety requirements

The core motive behind initiation of the product development at the system level is to design the activities for functional safety for system development. These activities will be included in the safety plan. All the necessary activities involved during the development of a system are described well in figure 3.13 [21].

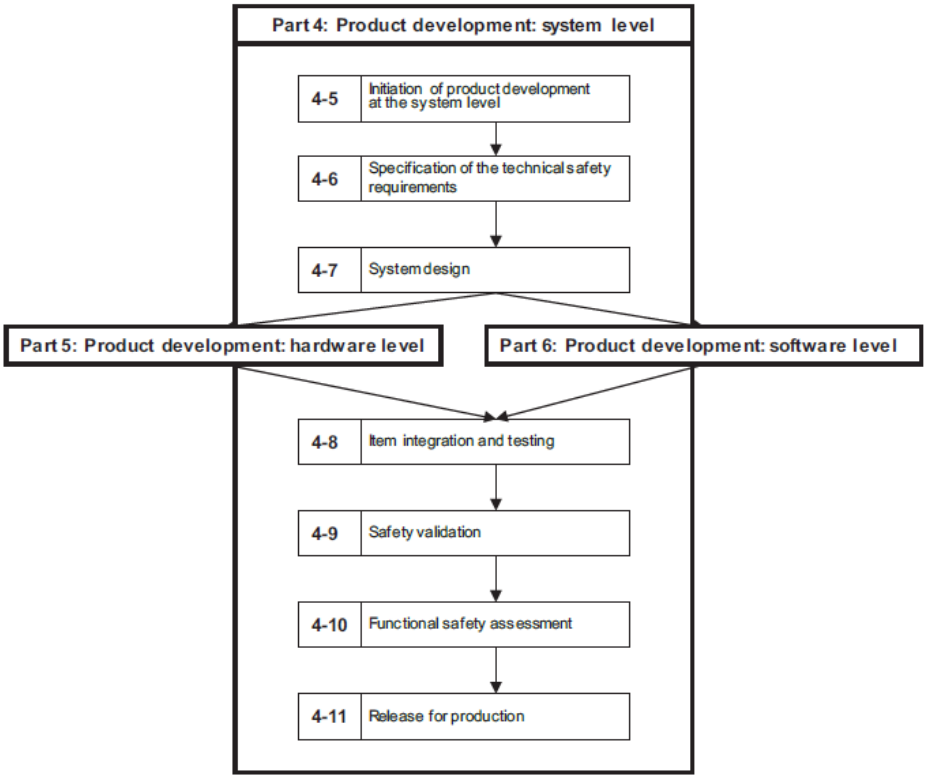


Figure 3. 13 Reference phase model for the development of a safety-related item

In cases where the system consists of multiple levels of integration figure 3.14 [21] provides an outline about its association with different part of the ISO 26262.

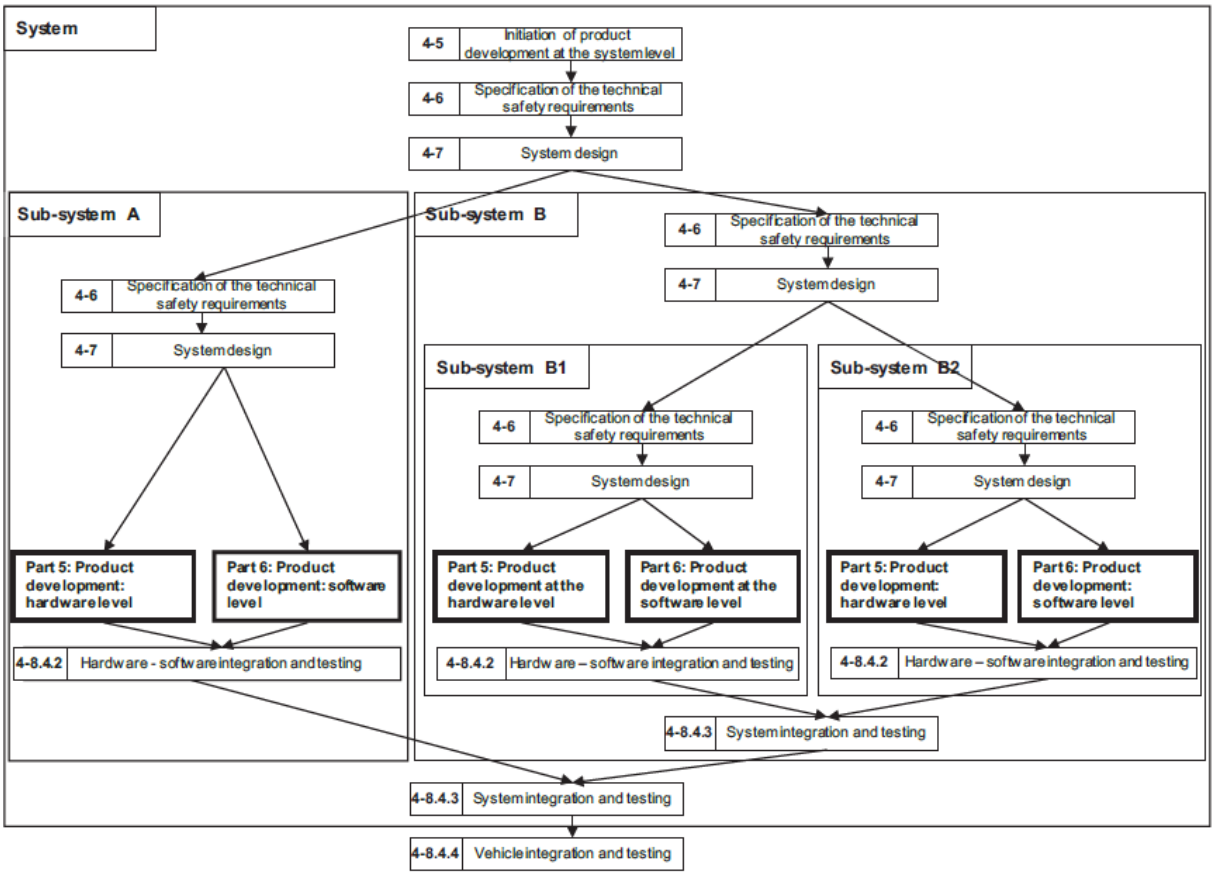


Figure 3. 14 Detailed view of product development at the system level

The first objective is to determine the technical safety requirements. This is refined from the functional safety concept. The second objective is to check whether the technical safety requirements are in par with the functional safety requirements through analysis. As the technical safety requirements are basically the requirements used to implement the functional safety concept, the other goal that lies with it is to assure that the item-level functional safety requirements are parted into the system-level technical safety requirements. While designing the architecture and system properties of the item the following should be taken under consideration:

- All the interfaces required for communication
- Constraints such as functional constraints or environmental conditions
- System configuration requirements

As the technical safety requirements describe the way the elements respond to scenarios that affect the penultimate point of achieving the safety goal, there are certain safety mechanisms to be considered for adhering to the ISO 26262 functional safety standard. These safety mechanisms shall be included in the technical safety requirements along with:

- Measures for detection, direction and control of the system during the event of a fault and/or from external devices
- Measures aiding the system to achieve/maintain a safe state
- Measures for the implementation of the warning and degradation concept
- Measures preventing faults from becoming latent

The measures that aid the system to achieve and then maintain the described safe state the following is to be specified:

- Safe state transition
- Fault tolerant time interval
- Measures to make sure that the safe state is maintained

The technical safety requirements of the requirements for the sample system are mentioned below. The technical safety requirements for the locking/unlocking mechanism of the door can be seen in figure 3.15.

Functional Safety Requirements	ASIL Level	Technical Safety Requirements	Fault Tolerant Time
10. Door is locked automatically when:			
10.1. (10 KMH ≤car speed)	A	1. Door ECU should check Central Locking ECU whether the door is not locked when the car speed≥10 KMH 2. Door ECU should lock the door 3. Door ECU should send signal to Central Locking ECU as the door is now locked	60 ms

Figure 3. 15 Technical safety requirements of door

The technical safety requirements for the opening/closing mechanism of the window can be seen in figure 3.16.

Functional Safety Requirements	ASIL Level	Technical Safety Requirements	Fault Tolerant Time
11. Automatically close window completely when:			
11.1. (150 KMH<car speed≤180 KMH) within (T_Window≤5 sec)	A	1. Door ECU checks the car speed to be greater than 150 KMH from Brake ECU 2. Door ECU should close window completely by operating the window motor 3. Start timer T_Window 4. Door ECU should send signal to Express System ECU as the window is now completely closed	90 ms

Figure 3. 16 Technical safety requirements of window

3.4.3 System design

The first objective lies in developing the design of the system and the technical safety concept. The second objective incorporated in this this sub-phase is to verify the design of the system made and the technical safety concept. So to develop a system architectural design for a particular system there are different activities that need to be covered, namely; technical safety requirements, functional safety requirements and non-safety-related requirements. Therefore, both safety and non-safety-related requirements are taken care of.

The activities on which the system design basically depends upon are the functional concept along with the assumptions made about the technical safety requirements and the architecture. Next activity is regarding the system architectural design. The technical safety requirements along with their respective ASIL level should be adhered while designing of the system and subsystem architecture. Then the ISO 26262 talks about the measure so as to prevent the advent of systematic failures. For this a safety analyses needs to be run on the system design which is further elaborated in table 3.5 [21].

All the tables mentioned in this standard follow a level of recommendation for the respective methods and these are categorized as:

- A "++" means that the method is highly recommended
- A "+" means that the method is recommended
- A "o" means that the method has no recommendation

Table 3. 5 System design analysis

	Methods	ASIL			
		A	B	C	D
1	Deductive analysis[a]	o	+	++	++
2	Inductive analysis[b]	++	++	++	++
[a] Deductive analysis methods include FTA, reliability block diagrams, Ishikawa diagram.					
[b] Inductive analysis methods include FMEA, ETA, Markov modelling.					

Once the identification of the external and internal causes for probable systematic failures is achieved, thereafter the ways in which they should be discarded should be though about. And

for the same the renowned automotive design principles for the system should be taken into account and including the following:

- Using well-renowned technical safety concepts
- Using well-renowned designs for elements
- Using well-renowned mechanisms for the detection of failures and their control
- Using well-renowned standardised interfaces

The results from the implementation of these on the item then need to be analysed. This applies to all the ASIL levels so as to avoid any failures from architectural design and high complexity. And should exhibit the properties like simplicity, granularity and modularity by the principles mentioned in table 3.6 [21].

Table 3. 6 Properties of modular system design

	Properties	ASIL			
		A	B	C	D
1	Hierarchical design	+	+	++	++
2	Precisely defined interfaces	+	+	+	+
3	Avoidance of unnecessary complexity of hardware components and software components	+	+	+	+
4	Avoidance of unnecessary complexity of interfaces	+	+	+	+
5	Maintainability during service	+	+	+	+
6	Testability during development and operation	+	+	++	++

3.4.4 Item integration and testing

This phase basically consists of three phases and two goals: the first phase involves the integration of hardware and software. The second phase talks about the integration process of all the elements to form a complete system. The third phase is about the integration of the item under consideration with various other systems present in the environment of the item.

Looking into the first objective is about the testing of compliancy of every safety requirement with respect to its particular ASIL level. The second objective lies in verifying the system design. The whole integration process is a step-by-step process starting right from hardware-software integration followed by system and vehicle integration. There are certain tests to prove the accordance of the integrations that happen at all the mentioned stages correctly in this sub-phase of the ISO 26262.

Once there is ample development at the hardware and software the integration at the system level can start. Testing the integration to ensure that the system design is in-par with the technical and functional safety requirements is the next step and the following need to be taken care of during the same:

- Precise implementation of functional and technical safety requirements

- Precise implementation of interfaces
- Robustness

Steeping deeper into system integration the following shall be performed:

- Refined hardware-software integration and testing plan
- Specifications of test at system and vehicle level should be a part of the item integration and testing plans
- Interfaces and the environment must be considered in the system and vehicle level item integration and testing plans

Table 3. 7 Methods for deriving test cases for integration testing

Methods		ASIL			
		A	B	C	D
1a	Analysis of requirements	++	++	++	++
1b	Analysis of external and internal interfaces	+	++	++	++
1c	Generation and analysis of equivalence classes for hardware-software integration	+	+	++	++
1d	Analysis of boundary values	+	+	++	++
1e	Error guessing based on knowledge or experience	+	+	++	++
1f	Analysis of functional dependencies	+	+	++	++
1g	Analysis of common limit conditions, sequences, and sources of dependent failures	+	+	++	++
1h	Analysis of environmental conditions and operational use cases	+	++	++	++
1i	Analysis of field experience	+	++	++	++

For targeting the correct implementation of the hardware-software integration and testing table 3.7 [21] gives some feasible test methods.

Table 3. 8 Correct implementation of technical safety requirements at the hardware-software level

Methods		ASIL			
		A	B	C	D
1a	Requirements-based test[a]	++	++	++	++
1b	Fault injection test[b]	+	++	++	++
1c	Back-to-back test[c]	+	+	++	++

These are applied to all the ASIL levels. Table 3.8 [21] talks about the probable effect that the hardware fault detection mechanisms may concur. This also considers the diagnostic coverage of fault models at hardware-software level and can be done by methods present in table 3.9 [21].

Table 3. 9 Effectiveness of a safety mechanism's diagnostic coverage at the hardware-software level

Methods		ASIL			
		A	B	C	D
1a	Fault injection test[a]	+	+	++	++
1b	Error guessing test[b]	+	+	++	++

This sub-section also entertains the possibility of integration and testing at the system and vehicle level. All the necessary test goals along with their test methods are listed clearly in it. This is also supported by ample number of tables to guide through it. But these are not evaluated in this thesis as it jumps out of bounds.

3.4.5 Safety validation

The first objective that is stated under this is to assure that whether the functional safety concept and the safety goals concur towards the functional safety of the item. The second objective talks about these goals being true and fully implemented at the vehicle level. This is to make sure that the results deriving from each activity adhere to their respective requirement.

The validation process of the item assures that it sticks to the functionality it was designed for and that it adheres towards the safety measures assigned to it. This plan for validation should include:

- Configuration of the item
- Specification of procedures, driving manoeuvres, test cases for validation purposes
- Required equipment and environmental conditions

This sub-section also entertains the possibility of validating at the system and vehicle level. All the necessary test goals along with their test methods are listed clearly in it. This is also supported by ample number of tables to guide through it. But these are not evaluated in this thesis as it jumps out of bounds. Although the gist for a few methods involved are mentioned below [21]:

- Repeatable tests with highly specific test procedures along with a fail/pass criteria. For instance; black box testing, fault injection, etc.
- Analyses. For instance; FTA, FMEA, etc.
- Long-term tests
- Real-life condition tests
- Reviews

3.4.6 Functional safety assessment

The objective is to judge the functional safety being led by the item. The entity responsible for the start of this assessment can be the vehicle manufacturer or the supplier. The requirements mentioned in this sub-phase apply to the ASIL levels of ASIL B, C, and D. While conducting the assessment for the functional safety the documentation involved in the same should include the following information [21]:

- Name and signature of the person who is responsible for the release
- Version of the particular item
- Configuration of the particular item
- References to corresponding documents
- Release date

But for the scope of this thesis this sub-section of the ISO 26262 is only analysed within limits. This marks the end of the development at the system level and all the specifications for the same are achieved. With all the work products from various activities in this section of the ISO 26262, Part 6 takes the next theoretical step.

3.5 ISO 26262 Part 6: Product development at the software level

This section gives a slight insight about how the software front is taken care in the ISO 26262 functional safety standard.

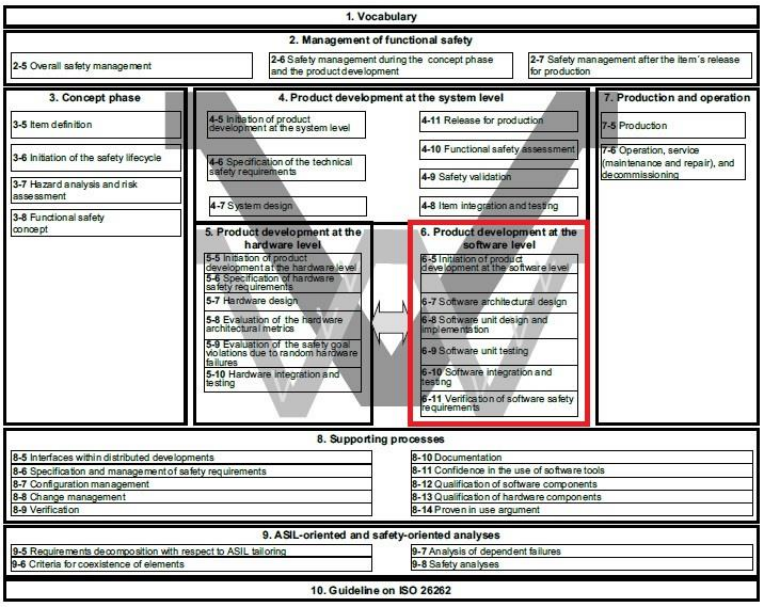


Figure 3. 17 ISO 26262 Part 6: Product development at the software level

This being the Part 6 of the ISO 26262, lying between the parts those talk about the system and the Part 7. Part 7 talks about the steps involved during the lifecycle of the product during hardware development. Noticeably, the development at the software level goes hand-in-hand with the development at the hardware level. As these two parts are very closely wound, it is essential to trace and look at the requirements mentioned in the Part 6 with respect to that of the Part 5. This is essential so as to absorb the right interpretation of the ISO 26262 standard with respect to all dimensions.

3.5.1 Scope

This part of the ISO 26262 specifies the requirements for automotive applications of product development at the software level consisting of [22]:

- Initiation of product development at the software level
- Software safety requirements
- Software architectural design
- Software unit design and implementation
- Software unit testing
- Software integration and testing
- Verification of software safety requirements.

3.5.2 Requirements for compliance

While achieving compliance with ISO 26262 every requirement should obey to it or unless one of the following reasons plays a role:

- the safety activities has been planned in accordance with the ISO 26262
- to accept the non-compliant-ability a rationale is provided and that the rationale is in accordance with ISO 26262

Depending upon ASIL-dependent requirements certain work products may not be needed as prerequisites. If at all ASIL decomposition has been performed at a previous stage of product development then the resulting ASIL level of the decomposition is complied with it. If any ASIL level is adjoined by parentheses, the respective sub-clause will be considered more as a recommendation rather than a requirement. And that this is different from the parenthesis of ASIL decomposition.

The remaining sub-sections of Part 6: Product development at the software level are analysed and elaborated in the following chapters of this thesis so as to maintain a proper workflow.

3.6 ISO 26262 Part 9: Automotive safety integrity level (ASIL)-oriented and safety-oriented analysis

This section is a highly important part of the ISO 26262 as far as safety of the whole system is considered. As this section provides with the different ways that assists in order to decompose the ASIL levels. The decomposition is slightly complex with due respect to the complexity of the system. This part plays a crucial role when it comes to its linkage with other parts of the ISO 26262. Hence, it becomes essential to have a handsome in-depth look at their decomposition and respective recommendations that need to be followed.

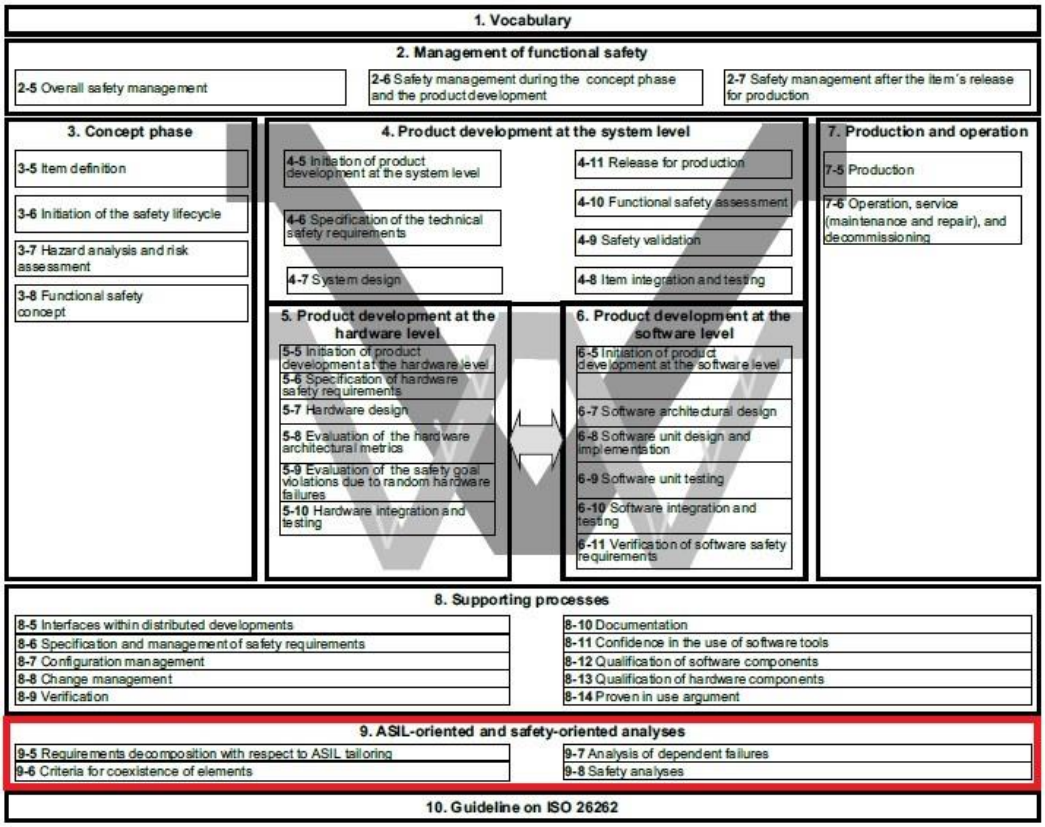


Figure 3. 18 ISO 26262 Part 9: Automotive safety integrity level (ASIL)-oriented and safety-oriented analysis

3.6.1 Scope

This part of ISO 26262 is about the Automotive Safety Integrity Level (ASIL)-oriented and safety-oriented analyses and includes the following [23]:

- requirements decomposition
- coexistence of elements

3.6.2 ASIL decomposition schemes

For the tailoring of the ASIL levels there are a set of procedures that guide through the decomposition of the safety requirements into redundant safety requirements. So the particular ASIL level is inherited by each following safety requirement. Starting with the functional and technical safety requirements.

Hence, this method of tailoring ASIL levels is termed as "ASIL decomposition". While the allocation process is going on there is a benefit if the architectural decisions has sufficient amount of independent architectural elements. This greatly helps in:

- Execution of safety requirements by the independent architectural elements
- By assigning a lower ASIL to the decomposed safety requirements

So in totality the ASIL decomposition imparts its application to a safety requirement amongst several elements which ensure that it is in par with the same safety requirement in respect to the safety goal. Therefore, the initial safety requirement is to be distributed in terms of redundant safety requirements by the use of sufficient elements.

For the ASIL decomposition at the software level, there has to be enough independencies amongst the elements and should be cross-verified at the system. If at all an ASIL decomposition leads to location of decomposed requirements towards a functionality along with an associated safety mechanism, then the following need to be considered [23]:

- The safety mechanism is to be assigned the highest decomposed ASIL level.
- The safety requirement corresponding to the intended functionality should be implemented with respect to the decomposed ASIL level.

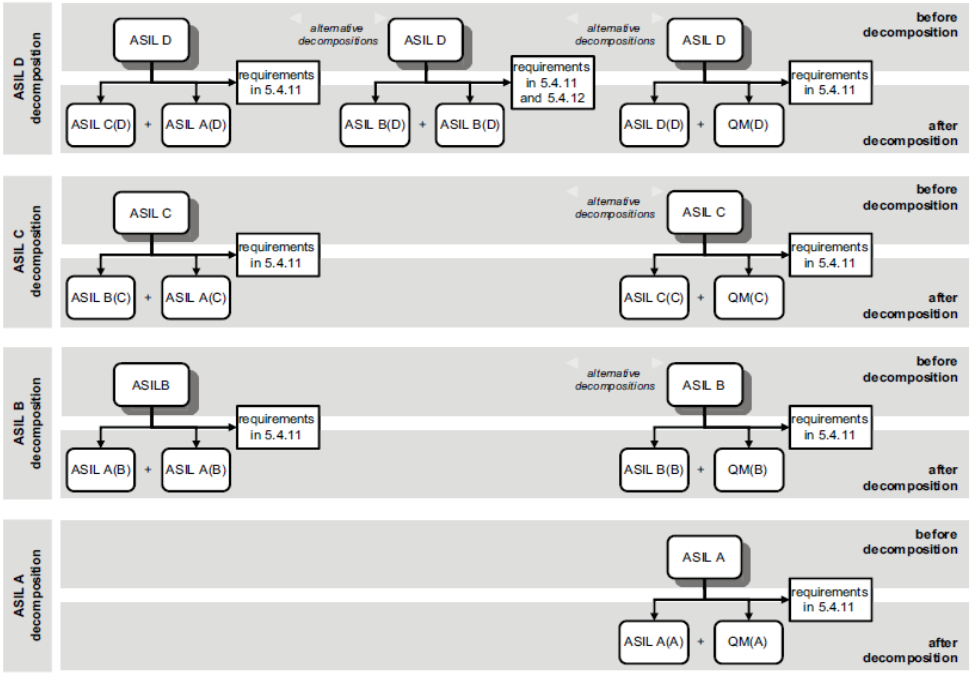


Figure 3. 19 ASIL decomposition schemes

If at all there is an issue with an initial safety requirement then the availability of sufficient independent elements that implement the decomposed safety requirements have to be considered. The following figure 3.19 [23] shows the different decomposition schemes that are mentioned in the ISO 26262 functional safety standard. A single decomposition of the ASIL level is defined when there is a transition from one step of one level to the lower next level and the following always needs to be kept under check:

- To comply the ASIL level with respect to the safety goal
- After decomposition there should be a sufficient independence of the elements

Utmost care need to be taken while using the decomposition scheme for ASIL D. And the following needs to be considered during the same time [23]:

- The decomposed safety requirements should be in par with that of ASIL C requirements so as to avoid any systematic failures
- The same software tools that were used for the development of the decomposed elements should also be used for the development of the ASIL D elements

4. Implementation

As the title suggests this section deals with the aspect of implementation undertaken during this thesis. It mainly highlights the ISO 26262 Part 6, which describes the product lifecycle at the software level. This section contains a detailed elaborate explanation of all the points that were considered for this thesis which also means that some of the points were excluded as they were out of the scope with respect to the thesis.

4.1 ISO 26262 Part 6: Product development at the software level

As this section mainly talks about the Part 6 of the ISO 26262 functional safety standard, it is executed parallely with the product development at the hardware level. This is because of the fact that both these parts of the ISO 262762 are very closely knit and should advisably be run concurrently. After the Part 6, the ISO 26262 Part 7: Production and operation, talks about the steps involved during the production and line of operation of the product. So this being the last part which talks about development, verification and validation at the software level should be interpreted well and as directed.

4.1.1 Initiation of product development at the software level

The primary objective of this sub-phase is to begin the functional safety activities required for software development. These activities in-order to start the software development sub-phases begin by determining the proper methods that need to be executed so as to fulfil the respective ASIL level. These methods are guided in the ISO 26262 by certain tools and guidelines.

The different phases that the product undergoes during its lifecycle in the software phase is described in figure 4.1 [22].

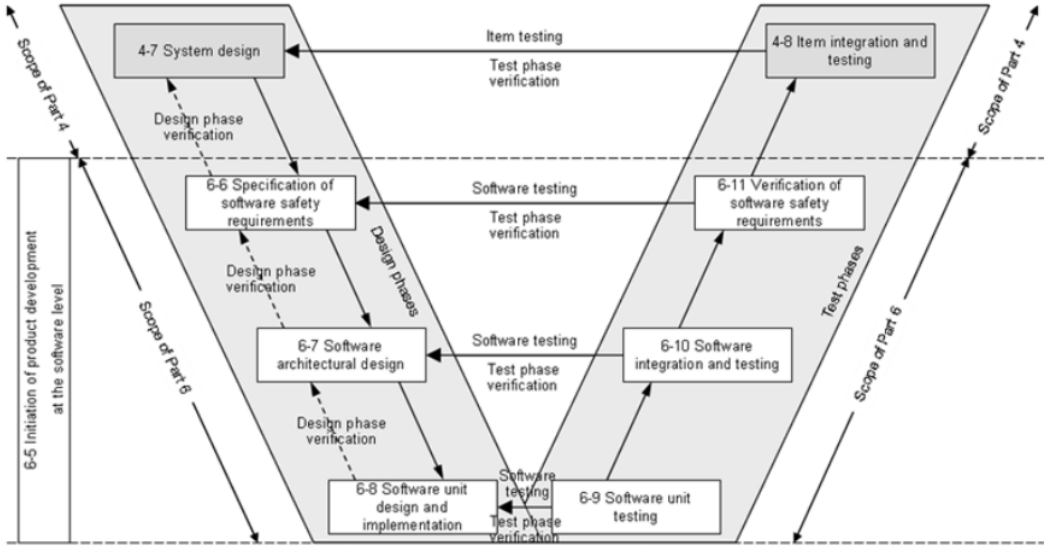


Figure 4. 1 Phase model for software development

For selecting the right programming/modelling language the below mentioned points need to be pondered upon:

- Unambiguous definition
- Should support embedded real-time software and error handling during runtime

Although assembly level languages can also be utilized in certain parts of the software where other high-level languages fail to deliver the desired goal. So to ensure that the programming/modelling languages adhere to the design and implementation, table 4.1 [22] should be looked through thoroughly.

Table 4. 1 Topics to be covered by modelling and coding guidelines

Topics		ASIL			
		A	B	C	D
1a	Enforcement of low complexity[a]	++	++	++	++
1b	Use of language subsets[b]	++	++	++	++
1c	Enforcement of strong typing[c]	++	++	++	++
1d	Use of defensive implementation techniques	o	+	++	++
1e	Use of established design principles	+	+	+	++
1f	Use of unambiguous graphical representation	+	++	++	++
1g	Use of style guides	+	++	++	++
1h	Use of naming conventions	++	++	++	++

4.1.2 Specification of software safety requirements

The first objective lies in specifying the software safety requirements. The second objective lies in a much more detailed specification for the hardware-software interface requirements. The third objective lies in verifying the software safety requirements and the hardware-software interface requirements. During the specification of the software safety requirements,

the restrictions that come with hardware are noted along with its impact on the software. After this the specification of software safety requirements is carried out in this part of the ISO 26262 functional safety standard.

The software safety requirements talks about all the issues that could probably occur leading to a failure and ultimately violating the technical safety requirement. Hence, the below mentioned points need to be considered well in advance while specifying the functions for the software safety requirements:

- Functions that lead the system to a safe state
- Functions assigned with detection and handling of faults
- Functions assigned with on and off-board tests
- Functions assigned with time-critical operations and performance specific

During the specification of the software safety requirements the following points should be considered:

- Configurations for the particular system and hardware under consideration
- Specification of hardware-software interface
- Timing constraints
- Information about external interfaces
- All operating modes for the system that may have an effect on the software

The earlier specified hardware-software interface should now be jotted in a detailed manner involving the extent of hardware usage along with safety dependency between software and hardware. At the end the final version of the hardware-software interface specification has to be verified and validated by persons responsible for the system and software/hardware development processes.

The specifications of the software safety requirements for the sample system are mentioned below. These specifications for the software safety requirements for the locking/unlocking mechanism of the door are seen in figure 4.2.

Functional Safety Requirements	ASIL Level	Technical Safety Requirements	Fault Tolerant Time	Software Safety Requirements
10. Door is locked automatically when:				
10.1. (10 KMH ≤car speed)	A	1. Door ECU should check Central Locking ECU whether the door is not locked when the car speed≥10 KMH 2. Door ECU should lock the door 3. Door ECU should send signal to Central Locking ECU as the door is now locked	60 ms	1. Door ECU should check 'B_CarSpeed' from Brake ECU for the car speed≥10 KMH 2. Door ECU should lock the door 3. Door ECU should send 'D_SER_Locked' to Central Locking ECU as the door is now locked

Figure 4. 2 Software safety requirements of door

These specifications for the software safety requirements for the opening/closing mechanism of the window are seen in figure 4.3.

Functional Safety Requirements	ASIL Level	Technical Safety Requirements	Fault Tolerant Time	Software Safety Requirements
11. Automatically close window completely when:				
11.1. (150 KMH<car speed≤180 KMH) within (T_Window≤5 sec)	A	1. Door ECU checks the car speed to be greater than 150 KMH from Brake ECU 2. Door ECU should close window completely by operating the window motor 3. Start timer T_Window 4. Door ECU should send signal to Express System ECU as the window is now completely closed	90 ms	1. Door ECU checks 'B_CarSpeed' from Brake ECU for car speed being greater than 150 KMH 2. Door ECU should close window completely by operating the window motor 3. Start timer T_Window 4. Door ECU should send 'D_SER_CompUP' to Express System ECU as the window is now completely closed

Figure 4. 3 Software safety requirements of window

4.1.3 Software architectural design

The first objective lies in developing a software architectural design for the system. The second objective is to verify the same. The software architectural design talks about all the present SWCs and their respective interactions. Also all the static sand dynamic aspects are defined. This sub-phase specifies all the necessary steps involved in-order to implement the software safety requirements and also how to manage any complexities that may arise during the software development process.

To assure that all the steps involved in his sub-phase of software development are carried out effectively, the software architectural design is prescribed with apt levels of abstraction via notations that are listed in table 4.2 [22].

Table 4. 2 Notations for software architectural design

Methods		ASIL			
		A	B	C	D
1a	Informal notations	++	++	+	+
1b	Semi-formal notations	+	++	++	++
1c	Formal notations	+	+	+	+

While developing the software architectural design the below mentioned points need to be taken care about:

- Verifiability of the software architectural design
- Suitability for the configurable software
- Feasibility of design and then implementation of the software units
- Testability of the software architecture design

To avoid any failures probable from complexities the software architectural design should confirm with encapsulation, modularity and simplicity by the principles listed in table 4.3 [22].

Table 4. 3 Principles for software architectural design

Methods		ASIL			
		A	B	C	D
1a	Hierarchical structure of software components	++	++	++	++
1b	Restricted size of software components[a]	++	++	++	++
1c	Restricted size of interfaces[a]	+	+	+	+
1d	High cohesion within each software component[b]	+	++	++	++
1e	Restricted coupling between software components[a, b, c]	+	++	++	++
1f	Appropriate scheduling properties	++	++	++	++
1g	Restricted use of interrupts[a, d]	+	+	+	++

The detailing of the software architectural design is in a way that it specifically talks about all the software units. Thus, it should describe:

a) Static design aspects that address the following areas:

- Software structure
- Logical sequence for processing of the data
- External interfaces of SWCs and the software
- Constraints of the architecture including various dependencies

b) Dynamic design aspects that address the following areas:

- Behaviour and functionality
- Concurrency of processes and their control flow
- Data flow amongst SWCs and external interfaces
- Temporal constraints

For determining the dynamic behaviour of the system one has to consider the different operating states of the same. Further describing of the dynamic behaviour includes specifying the communication relationships along with their presence with the system hardware. All the specified safety-related SWC is to be categorized as one of the below:

- Newly developed

- Reused but with/without modifications

If the requirements specify software partitioning so as to avoid faults due to interference between SWCs, then these can be carried out taking the below mentioned points into consideration:

- Shared resources should be utilized so as to assure freedom from interference

- Software partitioning is only allowed under dedicated hardware features

- For implementing a software partitioning usually the same or a higher ASIL level is to be considered

The next step involves the safety analysis of the software architecture. Mechanisms that are used for error detection so as to prove the conformity of the software architectural level so that it does not fail during the safety analysis is listed in table 4.4 [22]. These mechanisms are basically used to mitigate the probable residual risk on the behaviour of the system.

Table 4. 4 Mechanisms for error detection at the software architectural level

Methods		ASIL			
		A	B	C	D
1a	Range checks of input and output data	++	++	++	++
1b	Plausibility check[a]	+	+	+	++
1c	Detection of data errors[b]	+	+	+	+
1d	External monitoring facility[c]	o	+	+	++
1e	Control flow monitoring	o	+	++	++
1f	Diverse software design	o	o	+	++

This sub-clause applies to all the ASIL levels and it states that all the software safety mechanisms imparted at the software architectural level should include error handling techniques as listed in the table 4.5 [22]. This ensures the analysis of the software architecture in regards with probable hazards by hardware as well.

Table 4. 5 Mechanisms for error handling at the software architectural level

Methods		ASIL			
		A	B	C	D
1a	Static recovery mechanism[a]	+	+	+	+
1b	Graceful degradation[b]	+	+	++	++
1c	Independent parallel redundancy[c]	o	o	+	++
1d	Correcting codes for data	+	+	+	+

There has to be an assumption made on the resources that will be required by the embedded software and should consider the execution time, storage space and the resources required for communication. Lastly to verify the software architectural design via different verification methods so as to ensure that they impart their designated properties, table 4.6 [22] should be referred.

Table 4. 6 Methods for the verification of the software architectural design

Methods		ASIL			
		A	B	C	D
1a	Walk-through of the design[a]	++	+	o	o
1b	Inspection of the design[a]	+	++	++	++
1c	Simulation of dynamic parts of the design[b]	+	+	+	++
1d	Prototype generation	o	o	+	++
1e	Formal verification	o	o	+	+
1f	Control flow analysis[c]	+	+	++	++
1g	Data flow analysis[c]	+	+	++	++

4.1.4 Software unit design and implementation

The first objective is specifying the software units. The second objective lies in implementing them in the way they have been specified. The third objective lies in their static verification.

On the basis of the software architectural design, the software units are developed. These can directly be implemented as a model or a source code. All the properties corresponding to the functionality are achieved with manual code development at the source code level. And these properties apply to the model in case of model-based development that is partnered with automatic code generation. For the final implementation of these software units the source code is generated and further translated into an object code.

The following sub-clause is for specific safety-related software unit. So to make sure that all the steps involved for the software unit design are recognized and implemented perfectly as intended, the following notations listed in table 4.7 [22] should be taken into account.

Table 4. 7 Notations for software unit design

Methods		ASIL			
		A	B	C	D
1a	Natural language	++	++	++	++
1b	Informal notations	++	++	+	+
1c	Semi-formal notations	+	++	++	++
1d	Formal notations	+	+	+	+

The principles for designing the software units and their subsequent implementations as a source code level as listed in table 4.8 [22] and should adhere with the following properties:

- Precise order of execution of functions and subprograms
- Consistency of interfaces amongst the software units
- Correct flow of data and control flow between and amongst software units
- Simplicity and testability

Table 4. 8 Design principles for software unit design and implementation

Methods		ASIL			
		A	B	C	D
1a	One entry and one exit point in subprograms and functions[a]	++	++	++	++
1b	No dynamic objects or variables, or else online test during their creation[a,b]	+	++	++	++
1c	Initialization of variables	++	++	++	++
1d	No multiple use of variable names[a]	+	++	++	++
1e	Avoid global variables or else justify their usage[a]	+	+	++	++
1f	Limited use of pointers[a]	o	+	+	++
1g	No implicit type conversions[a,b]	+	++	++	++
1h	No hidden data flow or control flow[c]	+	++	++	++
1i	No unconditional jumps[a,b,c]	++	++	++	++
1j	No recursions	+	+	++	++

For the verification of the software unit design and its respective implementations the following methods in table 4.9 [22] should be taken into account in-order to demonstrate the below mentioned points:

- Compliance with the software-hardware interface specification
- Fulfilment of software safety requirements
- Adherence of source code with its design specification and coding guidelines

Table 4. 9 Methods for the verification of software unit design and implementation

Methods		ASIL			
		A	B	C	D
1a	Walk-through[a]	++	+	o	o
1b	Inspection[a]	+	++	++	++
1c	Semi-formal verification	+	+	++	++
1d	Formal verification	o	o	+	+
1e	Control flow analysis[b,c]	+	+	++	++
1f	Data flow analysis[b,c]	+	+	++	++
1g	Static code analysis	+	++	++	++
1h	Semantic code analysis[d]	+	+	+	+

4.2 Mechanisms for error handling at the software architectural level

This section deals with all the different mechanisms that are analysed during the period of this thesis. These mechanisms can be implemented into existing processes in-order to make sure that the final product adheres to their respective ASIL level according to the ISO 26262 functional safety standard. All these mechanisms are elaborated with terms to the automotive domain and are also categorized into different techniques according to the specification in the ISO 26262.

4.2.1 Table of methods

In order to jot down all the software safety mechanisms at the software architectural level, techniques that are used for error detection are mentioned in table 4.10 [22]. In the case when these are not directed by the technical safety requirements, they are used in the software safety mechanisms at a system level in-order to check on the probable effects on the way it behaves.

Table 4. 10 Mechanisms for error detection at the software architectural level

Methods		ASIL			
		A	B	C	D
1a	Range checks of input and output data	++	++	++	++
1b	Plausibility check[a]	+	+	+	++
1c	Detection of data errors[b]	+	+	+	+
1d	External monitoring facility[c]	o	+	+	++
1e	Control flow monitoring	o	+	++	++
1f	Diverse software design	o	o	+	++

4.2.2 Reciprocal Comparison

This is a very useful mechanism to detect an error at the architectural level during software development. This mechanism not only assures that the value of a particular message on which this mechanism is applied stays error free but in the advent of an error, this mechanism can also be extended as to shift to a safe state or to take counter actions or to just alarm the user. This is successfully implemented during this thesis and following are snippets from the code of the same.

This mechanism first stores the real value of the desired message. It then complements this and saves this as well. Whenever this desired message is required in the source code, the actual and the complement of the complemented values are checked against each other. If they are alike then the desired message is used otherwise one of the above mentioned error detection/correction techniques are used.

Firstly, a structure which carries the real and inverted values is coded. A snippet for the same is as in figure 4.4.

```
/* Struct for Bit Inversion of Variables                    */
typedef struct BitInvert
{
    uint8 BitInv_dir; /* "Normal" */
    uint8 BitInv_inv; /* Inverted */
} BitInv;
```

Figure 4. 4 Structure for bit inversion of variables

Then the function that receives and sets these real and inverted values is coded. A snippet for the same is as in figure 4.5.

```
/*!+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
!   Function Description: Function to store uint8 Variable "Normal" and inverted
!-------------------------------------------------------------------------------
!   Parameter:   Address: Pointer to Address of Variable
!                Value:   New value
!   Returnvalue: None
!   Global:      None
!+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
void Set_BitInvert (BitInv *address, uint8 value)
{
    address->BitInv_dir = value;
    address->BitInv_inv = (uint8) ~value;
}
```

Figure 4. 5 Function to store normal and inverted variable

Followed by the function that compares, returns and in case of an error triggers the watchdog. A snippet for the same is as in figure 4.6.

```
/*!+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
!   Function Description: Function to read uint8 Variable
!                       : Compare "Normal" and inverted value
!-------------------------------------------------------------------------------
!   Parameter:   Address: Pointer to Address of Variable
!   Returnvalue: uint8 "Normal" Variable
!   Global:      None
!+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
uint8 Get_BitInvert (BitInv const *address)
{
    uint8 Direct, Invert;

    /* Read Variable  */
    Direct = address->BitInv_dir;
    Invert = (uint8) (~(address->BitInv_inv));

    /* Compare Variable  */
    if (Direct != Invert)
    {
        WatchdogTrigger_UJA107xA();
    }
    return Direct;
}
```

Figure 4. 6 Function to read normal and inverted variable

The function ‘Set_BitInvert ( )’ and ‘Get_BitInvert ( )’ are used in the source code as shown in figure 4.7.

```
/*****************************************************************************************\
***  FUNCTION:
***      LkCar_WhnMove
***
***  DESCRIPTION:
***      Lock the door when Car starts
***
***  PARAMETERS:
***      Type              Name              Description
***      ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
***
***  RETURNS:
***      boolean
***
***  SETTINGS:
***
\*****************************************************************************************/
static void LkCar_WhnMove(eFzgStat *FzgStat_a, REC_Ign_Stat_Pr2_b0ko81o0x5jgvwtduf99gvzi0 *Ign_Stat_Pr2_a, REC_CLkS_Rq_45sc5jf37s715ovttd2e6tvwg *MT_CLkS_Rq_a)
{
    //static uint8 MT_Ca85_BedienbefehlZV = 0u;
    static uint8 MT_Ca1_Befehlsquelle = 0u;
    static BitInv s_MT_Ca85_BedienbefehlZV;
    static uint8 f_MT_Ca85_BedienbefehlZV;
    //boolean retVal = FALSE;

    //REC_CLkS_Rq_45sc5jf37s715ovttd2e6tvwg MT_CLkS_Rq_a;
    Rte_Read_R_CLkS_Rq_CLkS_Rq(&MT_CLkS_Rq_a);

    if ((*FzgStat_a == FZG_FAEHRT) &&
        (Ign_Stat_Pr2_a->ISw_Stat == I_C03_ISW_STAT_IGN_START) &&
        (MT_CLkS_Rq_a->CLkS_Dr_RR_Rq == 2u)
       )
    {
        //MT_Ca85_BedienbefehlZV = 1u;
        Set_BitInvert(&s_MT_Ca85_BedienbefehlZV, 1u);
        MT_Ca1_Befehlsquelle = 1u;
        Ca16_BedienbefehlTuer = B_TUER_SCHLIESSEN_AUTOMATISCH;
        //retVal = TRUE;
    }
    else if ((*FzgStat_a == FZG_FAEHRT) &&
             (Ign_Stat_Pr2_a->ISw_Stat == I_C03_ISW_STAT_IGN_START))
    {
        //MT_Ca85_BedienbefehlZV = 1u;
        Set_BitInvert(&s_MT_Ca85_BedienbefehlZV, 1u);
        //retVal = TRUE;
    }
    else
    {
        Set_BitInvert(&s_MT_Ca85_BedienbefehlZV, 0u);
        MT_Ca1_Befehlsquelle = 1u;
        Ca16_BedienbefehlTuer = B_TUER_SCHLIESSEN_AUTOMATISCH;
        //retVal = FALSE;
    }

    f_MT_Ca85_BedienbefehlZV = Get_BitInvert(&s_MT_Ca85_BedienbefehlZV);

    Rte_Write_ppBedienbefehlZv_BedienbefehlZv(f_MT_Ca85_BedienbefehlZV);
    Rte_Write_ppBedienbefehlTuer_Befehlsquelle(MT_Ca1_Befehlsquelle);

    //return retVal;
}
```

Figure 4. 7 Source code

4.2.3 Cyclic redundancy check

Cyclic Redundancy Check (CRC) is one of the best checksums available for detecting/correcting errors that happen during communication transmissions. As the modulo-2 arithmetic that can also be used to calculate the CRC is not easy to map onto the software, CRC becomes the next obvious choice.

There are various global and mathematically accepted standardized CRC polynomials that exist. So, these are usually taken into account instead of developing something new cause of its unreliability and due to the fact that these polynomials are proven to their limit.

Along with the generator polynomial, every globally accepted CRC standard comes along with some other parameters. These parameters give an insight about how should the respective CRC standard be computed. Table 4.11 has some of these parameters for three of the most popular CRC standards. These parameters include 'initial remainder' and the 'final XOR value'. The sole aim of these two C-bit constants is likewise to the last bit inversion step added to the sum-of-bytes checksum algorithm. In order to eliminate various ordinarily undetectable differences, the help from these parameters is taken. So basically they add a bulletproof vest to the already strong algorithm behind these checksums [24].

Table 4. 11 CRC standards with their parameters

	CRC-CCITT	CRC-16	CRC-32
Width	16 bits	16 bits	32 bits
(truncated) Polynomial	0x1021	0x8005	0x04C11DB7
Initial Remainder	0xFFFF	0X0000	0xFFFFFFFF
Final XOR Value	0X0000	0X0000	0xFFFFFFFF
Check Value	0x29B1	0xBB3D	0xCBF43926

4.2.4 Multiple reads of received signal

This technique can be utilized in cases where the value of a particular signal should be on point. This technique involves reading the signal multiple but defined number of times before it is used and then averaging all the read values. This would not only give a much more reliable value of the signal but it can also be extended by removing the highest and the lowest values measured and taking the average of the remaining read values.

Once the signal is read multiple defined times, it can be stored is an array. The elements of this array can then be compared so as to figure out the highest and the lowest amongst the measured values. Once these two values are identified, they should be ignored from the remaining averaging process. Then the average of the remaining measured values should be carried out and the final averaged value should be stored in a separate variable. So, whenever this signal is used any further in the program, the variable can be read to obtain the value of that particular signal.

This technique can successfully keep errors at bay and assure to provide the most apt value to the desired signal during the runtime of the program.

4.2.5 Validity of received signal

This mechanism is a very healthy way to assure that the received signal is in bounds of what is desired and if that is not the case various counter-measures can be taken with what is

mentioned in the product requirements. This would result in assurance of the received signal to be in the bounds of the accepted values.

To elaborate further on this, the particular signal that needs to be received should first be checked for certain criterions before it is further utilized in the program. These criterions may include checks like whether the received signal is in bounds of the acceptable values with regards to the current state of the program or whether the signal should even be present at all during that particular state of the program and various other checks which may be further polished and designed keeping in mind the functionalities of the program intended for.

If the received signal does not obey to either one of the above mentioned checks then the program can be designed so as to fall onto a fail state, pick up the last correct value, pick up the default value or continue its degraded running while just alarming the user about the occurred error. These can be tailored with keeping in sight the requirements and the functionalities that the program will be executed for.

4.2.6 Error-correcting code

Error-Correcting Code (ECC) is executed by widening the memory along with the addition of a combinatorial logic between the path of this extra memory as in figure 4.8. The logic behind the ECC encoding comes from a well-tested Hamming algorithm. The ECC has an ECC bit generator. This generates ECC bits from the data that is under consideration and then stores the same along with the regular data. At the output of the memory, an ECC detection/correction logic function is used.

While reading from the memory, this particular function checks the combination consisting of regular and ECC data. If it is error free, then it will allow the regular data through and will keep it unchanged. But even if there is an error of a single bit detected, it would correct this error bit and then allow the regular data to pass through. Although with all these correct bits it would also raise a flag. In the advent where there are two errors, it would initially raise the flag and then would allow the system to react to the event in its own way [25].

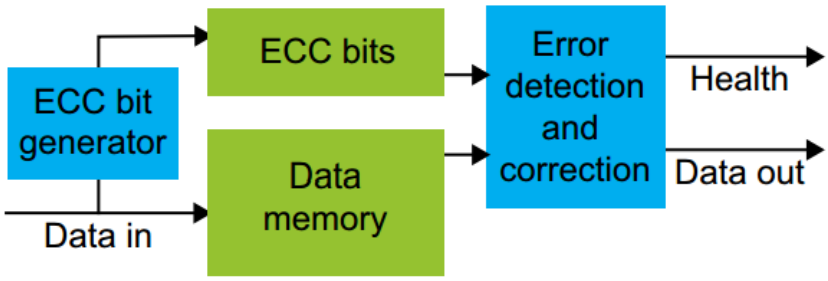


Figure 4. 8 Wider ECC-enabled memories with additional logic

This capability of correcting single errors and detecting double errors is very beneficial. Apart from this it has immunity against other types of errors as well. Single hard errors like a stuck bit line inside a memory, misleading connection on a printed circuit board (PCB) can also be looked upon by ECC. The main point lies that all single bit errors in a word will be corrected. Which means that ECC can handle many issues of the system at the same time as long as they are in a single word and don’t have more than a single bit error in it.

It can also be used to indicate the status of the health of the system. In case of a single bit error in a word, the ECC can also inform the processor about it so that the operator can take the needed and desired measures against it. So this turns the system towards a maintenance task rather than a degraded state, as against a recall towards an unknown fatal system error. By performing heuristic probabilities, a model can be generated that shows the estimation of soft errors with and/or without the ECC logic.

In the modern world of embedded systems, the memories suffer from bit flips cause by cosmic energy injections and errors due to transmission by noise and jitter which usually result in the transfer of incorrect data. Even though these errors are seldom heard off, but with the ever-growing size of the memory, bandwidth and interface frequencies it may be a common scenario sooner than ever. During embedded system design the attention given towards system-level error resilience is increasing day-by-day.

4.2.7 Memory protection unit

The protection of the memory is an important feature of the OS which helps in safety and reliability of real-time systems. There are numerous types of protection methods namely; processing overhead, multiple-thread support, memory consumption, hardware support and region enlargement. There are new methods also being developed for memory protection. One of such methods is called Intermediate-level Skip Multi-Size Paging. This works by skipping all the intermediate-level page tables belonging to the Multi-level Paging that are not used. It also works with multiple page sizes.

Looking at the exponential growth of embedded systems in the automotive domain in recent times, the day is not far when all these systems will be connected to each other via fields networks instead of the old-age wire harnesses. This translates into firstly, a lowered cost of wiring and secondly, a shift from centralized controller systems having high-efficiency microprocessors to a much more distributed type of a control system including small but several microprocessors. With the constant growth in the VLSI technology, the availability of low-cost single-chip small embedded systems will be sooner than ever. And they would consist of high-performance RISC microprocessors along with built-in memory and transceivers for communication, I/O ports and A/D converters. Corresponding to the small size of these embedded systems, the code that runs on them is respectively small as well. *a priori* or fixed-based are their usual memory requirements. Embedded software is usually built in a way that has several modules and because of that it needs multiple-thread service. Thus, any real-time operating system (RTOS) that is being used on an embedded system should obey to the above mentioned requirements and also have a small code size.

So for small embedded systems like an ECU the below mentioned metrics need to be taken care of when determining the feasibility of memory protection [26]:

1. Efficiency of memory usage
   – Support for threads
   – Enlargement of stack and data regions

2. Processing overhead

3. Memory consumption

The memory-protection is basically divided into two methods namely; logical-address-based or physical-address based. Logical-address-based consist of Segmentation and Paging, while Linked List and Bit Map belong to physical-address based. The most prominent memory protection methods in the automotive are of five types. And they are elaborated as follows [26]:

1. Bit Map: This memory protection mechanism divides physical memory into pages. Every process initially verifies the information regarding the protection of the respective page whenever it has to accesses the memory. For instance, to construct memory management information for a page, could be done by information about just 2-bit-size read/write protection.

2. Segmentation: This memory protection mechanism divides the space of the logical address into multiple segments. Each of these segments is then allocated onto the space of the physical address. Although this mechanism needs a memory management table which consists of the segment table entries (STEs) that manage their respective segments.

3. Multi-level Paging: This memory protection mechanism divides the space of the logical address into fixed-size pages. Each of these pages is allocated onto the space of the physical address. Although this consists of memory management tables. These tables are built in multiple levels of hierarchy so as to reduce the cumulative size of tables. Figure 4.9 depicts a page table structure with 3-level Paging.

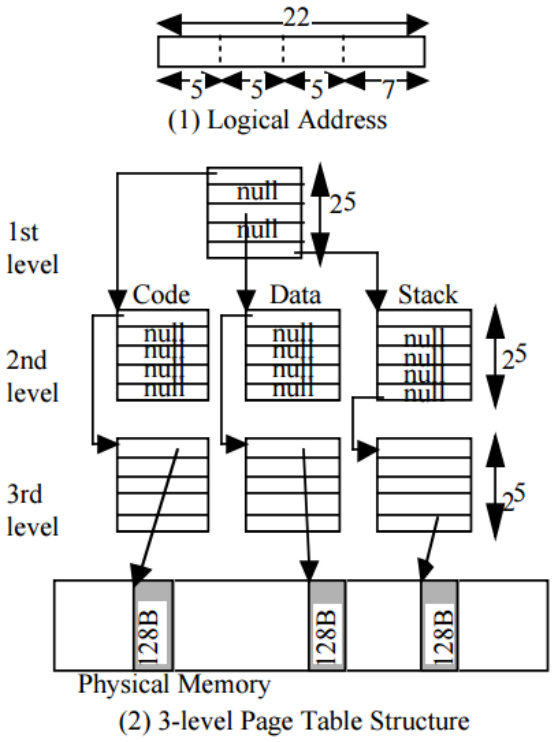


Figure 4. 9 Multi-level paging (3-level)

4. Paged Segmentation: This memory protection mechanism divides each segment of the space of the logical address into pages. Each of these pages is then allocated onto the

space of the physical address. The management of the memory happens by the use of several page tables and a segment table.

5. Short-Circuit Segment Tree: This memory protection mechanism is for Extremely Reliable Operating System (EROS) which was derived from the University of Pennsylvania and basically is a Multi-level Paging. It works by skipping page tables that contain only the first page table entries (PTEs). This results in reduction of the total table size by comparison with Multi-level Paging and the processing overhead.

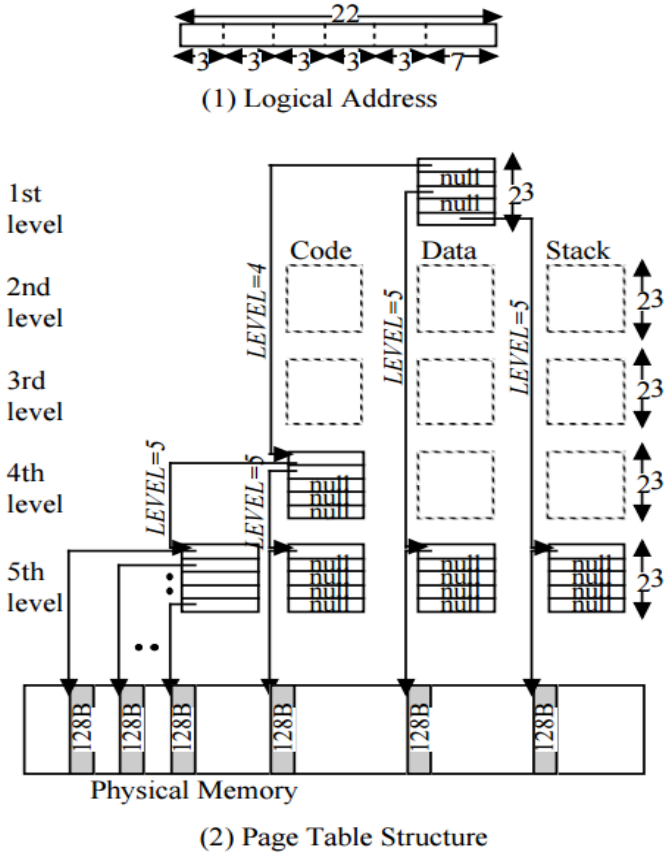


Figure 4. 10 Short-circuit segment tree

Figure 4.10 depicts the structure of the page table for Short-Circuit Segment Tree. In figure 4.10 (1) the logical address format of a 5-level Paging and in figure 4.10 (2) depicts its page table structure. As seen in these figures, the first-level PTE links directly to the fifth (bottom)-level page tables and the second- to the fourth-level page tables of the stack and data areas are skipped. The level information is present in the PTE and directs towards the level of the page table that needs to be linked. It is because of this that it is possible to eliminate intermediate-level page tables.

So while in a multiple-thread support environment efficient use of thread-stack areas become necessary. This is so because of the fact that it is important to handle the thread-stack areas while detaching/attaching them to each process solely on the demand basis. Shrinking/enlarging the stack or data areas dynamically on demand is also a necessary aspect that needs attention. Below mentioned are certain points to target for the efficient use of memory [26]:

- Efficient threads support - In the advent of a Bit Map, a thread-stack area can be used efficiently between processes and by further dividing a section of the *a priori* memory

into bundles of thread stacks along with a pool of free thread stacks. In the advent of a logical-address-based memory protection, involves the memory manager handling the thread stacks. It does so by either pages or segments. While in Segmentation, the use of segment table in order to support numerous threads by allocation of one segment onto one thread stack. Although the issue over here lies with the amount of STEs required in segment table which in turn increases the memory requirements. Looking into the paging-based management including Short-Circuit Segment Tree, Paged Segmentation and Multi-level Paging, involves the memory manager handling the thread stacks in a very flexible fashion per page. The memory manager gets them via the pool of free physical pages. This ultimately makes it an easy-to-use thread-stack areas. With the analysis and the current trend in the automotive domain the paging-based and Bit Map are the most widely used when it comes to multiple-thread support.

- Region enlargement - In Segmentation and Bit Map it becomes a difficult task to grow the stack or data regions dynamically. This is because these regions have to be allocated contiguously in the memory which gives no guarantee of free space in their neighbourhood. Paging based management involves the memory manager getting a new physical page from the pool of free physical pages. It then enlarges this region and manages it in a very flexible manner. As in conclusion, the paging-based managements are best suited when there is a need for region enlargement.

4.2.8 Hamming code

Hamming code is an effective mechanism that provides surety towards error detection. This can be achieved by two types of functions. The first one is by the use of matrices and the second one by the use of tables. Both of these mechanisms are elaborated below and are as follows [27]:

1. Encoding with Matrices - The technique involved in this type of hamming code is to realize all the values that are needed and further pack them into certain justified bytes like unsigned chars. This is done by packing the data that needs to be encoded into a byte. And the columns of the generator matrix are packed into another array of unsigned chars. In this case the *ith* bit corresponding to the code word is equal to the times of data word in the *ith* column of its corresponding generator matrix. This is the *ith* entry of the generator matrix in the array. Corresponding to the modulo-2 arithmetic, this proves to be just a bitwise ANDing of the two bytes which is then followed by an XOR.

2. Encoding with Tables – In this mechanism of hamming code, tables are utilized. For instance, in a hamming code of (7,4) translates into only 4 bits of data to encode at a time and consists of only sixteen possible data values for encoding. So creating a sixteen byte array named as hamming Codes and then defining it in such a way that ‘code word = hamming Codes[data value]’. The downside of taking use of such a lookup table is that it needs to be computed for every possible combination of code word well in advance.

Now decoding the hamming codes implemented by these two functions are done in the below mentioned ways and they are follows:

1. Decoding with Matrices – This is pretty much similar like the encoding with matrices. But the difference is that this time a packed array consisting of indexes representing the bits in a row from the parity check array is multiplied by a byte that contains the code word which was received. This can be implemented again by just XORing and ANDing. The outcome from this matrix multiplication is known as the 'syndrome'. In the case where this syndrome is null, then the least significant four bits of the respective code word is the data that is encoded. In the case where this syndrome is a non-null entity, play with the bit that is at the identical position to that of the column in the parity check matrix which ultimately matches with the syndrome. Moreover an array can be attached to the syndrome. This array acts as its index and is used to provide a mask for avoiding different types of errored bit. Once the error is got rid of then the least significant four bits of the code word represent the encoded data.

2. Decoding with Tables – Like the encoding with tables for the hamming code the decoding also works on similar lines. As a hamming code of (7,4) translates only to a seven bit code word, this results to only 128 possible values to further decode. So a 128 byte array is created called hammingDecodeValues and is defined in such a way that 'data value = hammingDecodeValues[code word]'. As for this example hamming codes of (7,4) decodes to only 4 bit code words, a data space can be used so as to pack the two data values into a single byte. These can be a part of a look-up table as well. The value of this hammingPackedDecodeValues[*code word* / 2] is a byte. This byte is then packed in a way such that the four least significant bits correspond to the data values of an even code word whereas, the four most significant bits correspond to the data value of an odd code word.

The difference between the generation of codes for dealing with packed and unpacked look-up tables is that the packed look-up tables may be smaller but slower compared to the unpacked look-up tables. Although the downside of utilizing such a lookup table lies in the fact that it needs to be computed so as to apply the correction to every possible code word that is received well in advance.

5. Testing and Verification

This section deals with aspect of verifying and testing with respect to the ISO 26262 functional safety standard. The standard defines clear methods and tables of all the necessary steps that are involved in-order to assure that the particular ASIL level is achieved. This section lists the testing for product development at the software level. And some other tests at other different levels in the ISO 26262 Part 6 are neglected as were considered to be out of the scope of this thesis.

5.1 ISO 26262 Part 6: Product development at the software level

This section deals with the different software techniques that should be involved during the product lifecycle mentioned in the ISO 26262 functional safety standard. It covers the software testing at the unit level followed by its integration and testing of the same. These are elaborated in context of this thesis, although minor points are ruled out as they were out of bounds as far as the thesis was concerned.

5.1.1 Software unit testing

The sole purpose of this sub-phase is to ensure that the software units agree to the software unit design specifications and also they should not contain any undesired functionality. So, as to achieve this, a procedure for testing it against the software unit design specifications is written.

In this sub-section the term ‘Safety-related’ refers to the unit that consists of safety requirements. Software unit testing is then designated, specified and executed in accordance with ISO 26262. Although for model-based software development, the parts of the model represent objects for test planning. Hence, depending upon the selected software development process, the test objects may vary from the model itself or the code derived from this model.

The software unit testing methods listed in table 5.1 [22] should be implied so as to assure that the software units have:

- Adherence with software unit design specification

- Adherence with specification of the hardware-software interface
- Robustness

Table 5. 1 Methods for software unit testing

Methods		ASIL			
		A	B	C	D
1a	Requirements-based test[a]	++	++	++	++
1b	Interface test	++	++	++	++
1c	Fault injection test[b]	+	+	+	++
1d	Resource usage test[c]	+	+	+	++
1e	Back-to-back comparison test between model and code, if applicable[d]	+	+	++	++

To ensure the correctness of test cases for software unit testing, test cases should be obtained from the methods mentioned in the table 5.2 [22].

Table 5. 2 Methods for deriving test cases for software unit testing

Methods		ASIL			
		A	B	C	D
1a	Analysis of requirements	++	++	++	++
1b	Generation and analysis of equivalence classes[a]	+	++	++	++
1c	Analysis of boundary values[b]	+	++	++	++
1d	Error guessing[c]	+	+	+	+

To ensure the completeness of test cases and to make sure that there is no unintended functionality, the evaluation of requirements should be judged, whereas the structural coverage should also be measured with respect to the table 5.3 [22]. In the advent that structural coverage is not sufficient, then a rationale shall be provided or additional test cases should be specified. For instance, evaluation of structural coverage can lead to issues like deactivated code, dead code, requirement-based test cases, dead code, unintended functionality or inadequacies in requirements.

Table 5. 3 Structural coverage metrics at the software unit level

Methods		ASIL			
		A	B	C	D
1a	Statement coverage	++	++	+	+
1b	Branch coverage	+	++	++	++
1c	MC/DC (Modified Condition/Decision Coverage)	+	+	+	++

The test environment should be as close as possible to the target environment. If this is not done then the differences between the source and the object code, and between the test and target environment, should be evaluated so as to specify additional tests in the target environment. The different environments are namely:

- model-in-the-loop tests
- software-in-the-loop tests

- processor-in-the-loop tests
- hardware-in-the-loop tests

5.1.2 Software integration and testing

The first goal of this sub-phase is to integrate the software elements. The second goal is to play with the software architectural design. The interfaces between the software elements and the specific integration levels are tested against the architectural design. The steps of the testing and integration correspond to the hierarchical architecture of the software.

The scheduling of software integration describes the steps for integrating the individual software units. This is done in a hierarchical fashion into the software components and is done until the software is completely integrated. This should be carried out making sure to provide ample of support towards functional dependencies related to software and software-hardware integration. The table 5.4 [22] shed some light on the software integration test methods. These methods should aim for:

- The functionality that is specified
- Robustness
- Ample of resources to aid the functionality

Table 5. 4 Methods for software integration testing

Methods		ASIL			
		A	B	C	D
1a	Requirements-based test[a]	++	++	++	++
1b	Interface test	++	++	++	++
1c	Fault injection test[b]	+	+	++	++
1d	Resource usage test[cd]	+	+	+	++
1e	Back-to-back comparison test between model and code, if applicable[e]	+	+	++	++

Table 5.5 [22] refers to all the methods for the appropriate specification of test cases for software integration.

Table 5. 5 Methods for deriving test cases for software integration testing

Methods		ASIL			
		A	B	C	D
1a	Analysis of requirements	++	++	++	++
1b	Generation and analysis of equivalence classes[a]	+	++	++	++
1c	Analysis of boundary values[b]	+	++	++	++
1d	Error guessing[c]	+	+	+	+

In order to ensure that there exists completeness of tests and no unintended functionality, the coverage of requirements shall be determined by test cases. If needed a rationale should be

provided or additional test cases should be specified. This applies to all the ASIL levels. This can be assured by the methods mentioned in table 5.6 [22].

Table 5. 6 Structural coverage metrics at the software architectural level

Methods		ASIL			
		A	B	C	D
1a	Function coverage[a]	+	+	++	++
1b	Call coverage[b]	+	+	++	++

5.2 Testing of mechanisms for error handling at the software architectural level

This section has to do with the testing of the safety mechanisms used at the software architectural level. The mechanism spoken about in this section of this thesis was implemented into an existing project onto an AUTOSAR compliant ECU. And was further tested to check if it reacted as expected. The mechanism that was implemented was the 'reciprocal comparison'. This was done on a particular message which was selected with respect to its importance for a particular SWC.

5.2.1 Test specification

The test specification is an approach towards software or a combination of softwares. This consists of the inputs, expected results and implementation conditions (test procedures) for all the related tests. It jots down a precise description of each test and its corresponding requirements. Each requirement mentioned has a very unique functionality to it. The test procedures very clearly explain all the steps to be taken for testing software in-order to get the expected results out from it.

The mechanism for error handling at the software architectural level that is being tested over here is the reciprocal comparison that is implemented on the AUTOSAR compliant ECU. Its respective test specification are as follows:

1. Connect the ECU to the iSYSTEM iC5000 debugger
2. Open the project workspace in the iSYSTEM winIDEA
3. Open the software component 'Swc_BEK.c'
4. Connect the iSYSTEM winIDEA to the debugger hardware
5. Connect the Vector CANcaseXL to the CAN high and CAN low pins of the ECU
6. Open the simulation for the ECU in Vector CANoe
7. Start the simulation so as to wake up the ECU via CAN messages

8. Add a breakpoint at the function call of ‘LkCar_WhnMove ( )’

9. Add another breakpoint at the function call of ‘Get_BitInvert ( )’ for message ‘s_MT_Ca85_BedienbefehlZV’

10. Supply the ECU with Kl. 30 and Kl. 31

11. Download the software onto the ECU via iSYSTEM winIDEA

12. Run the software

13. If the inverted and direct values of the message ‘s_MT_Ca85_BedienbefehlZV’ are the complement of each other. Then it is a positive test. And the function ‘Get_BitInvert ( )’ should return the program control to the function that writes to the RTE i.e. ‘Rte_Write_ppBedienbefehlZv_BedienbefehlZv ( )’

14. Stop the running of the software and restart it

15. Now falsify the value of the message ‘s_MT_Ca85_BedienbefehlZV’ at the breakpoint of function call of ‘Get_BitInvert ( )’

16. As the inverted and direct value of the message ‘s_MT_Ca85_BedienbefehlZV’ are not each other’s complement anymore. Thus, this is a negative test. And the function ‘Get_BitInvert ( )’ should not return the program control to the function that writes to the RTE i.e. ‘Rte_Write_ppBedienbefehlZv_BedienbefehlZv ( )’. It should trigger the watchdog by calling the function ‘WatchdogTrigger_UJA107xA ( )’

5.2.2 Test work products

The results that are obtained at each step of the test procedure mentioned in the test specification are termed as test work products. These test work products helps to analyse the problematic area if the results are not as expected. Having these test work products for every step of the test procedure also helps to pinpoint the exact location of the error, if any. Otherwise, it’s a good practice to have all the test work products for a particular test specification for future reference.

The test work products for the mechanism of error handling without falsifying the bit at the software architectural level that was implemented on an AUTOSAR compliant ECU by reciprocal comparison is mentioned as follows:

1. The values of ‘BitInv_dir’ and ‘BitInv_inv’ are each other’s complement at the breakpoint of function call of ‘Get_BitInvert ( )’ as in figure 5.1.

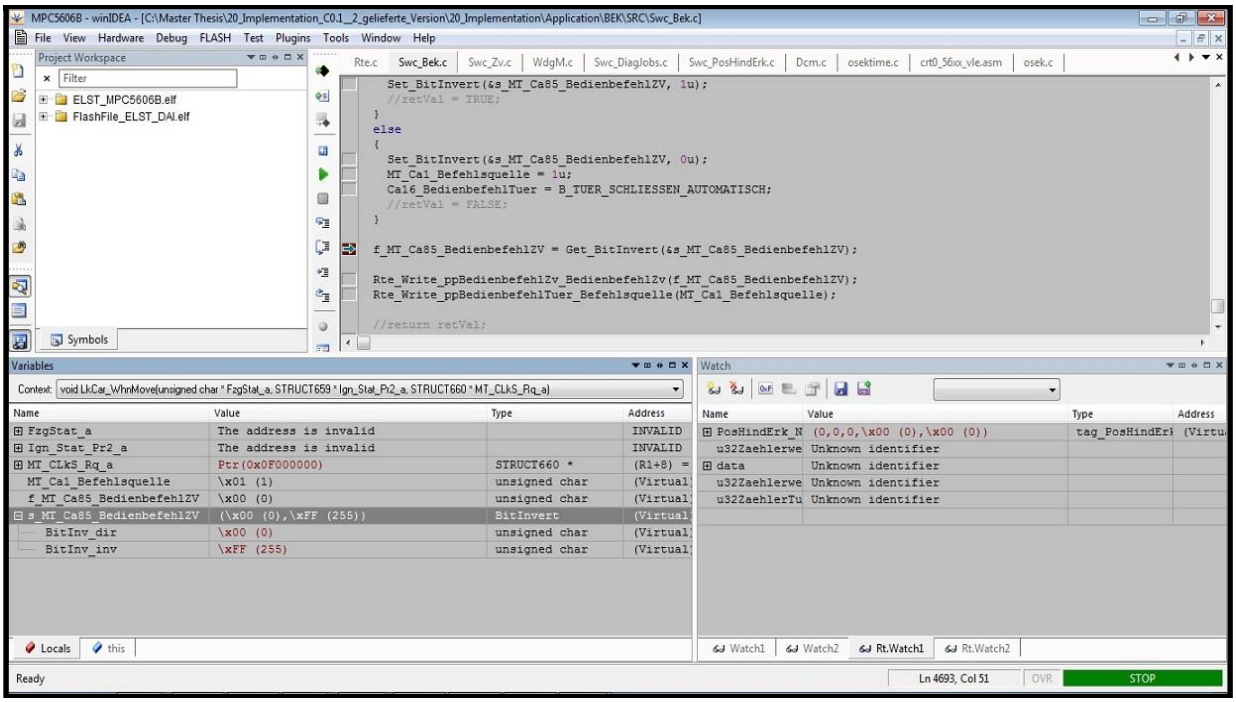


Figure 5. 1 Test work product without falsifying bit-1

2. The program control shifts to the function ‘Get_BitInvert ( )’ as in figure 5.2.

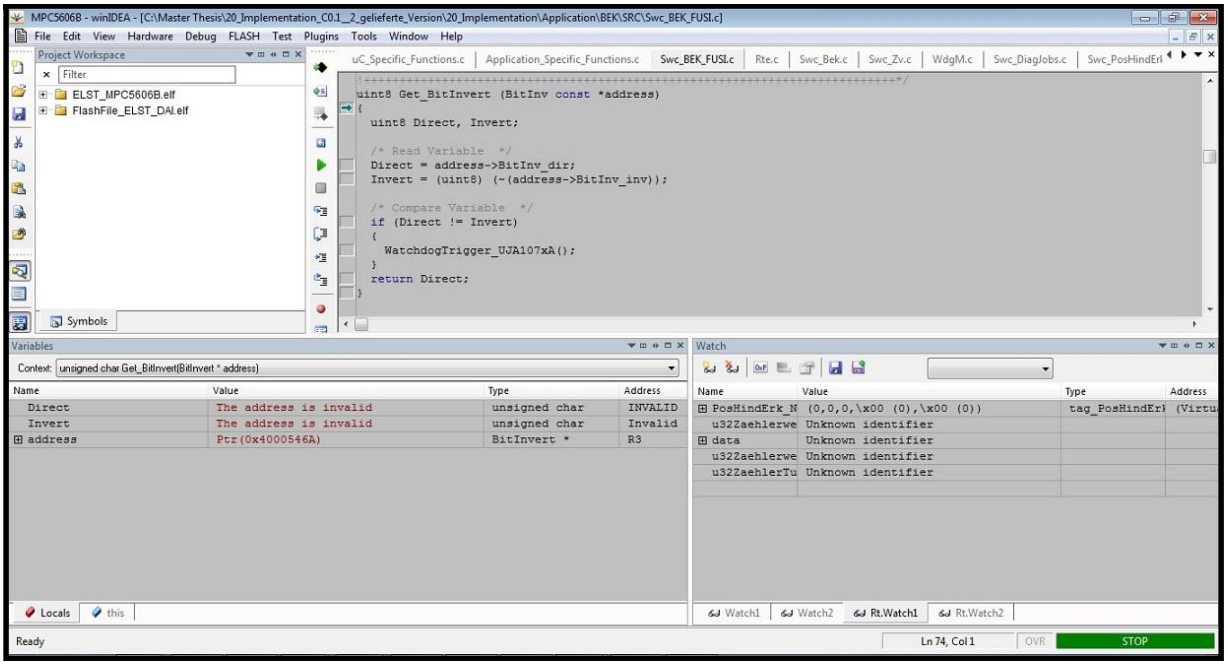


Figure 5. 2 Test work product without falsifying bit-2

3. And it moves out of this function returning the value of the variable ‘Direct’ as in figure 5.3 and figure 5.4.

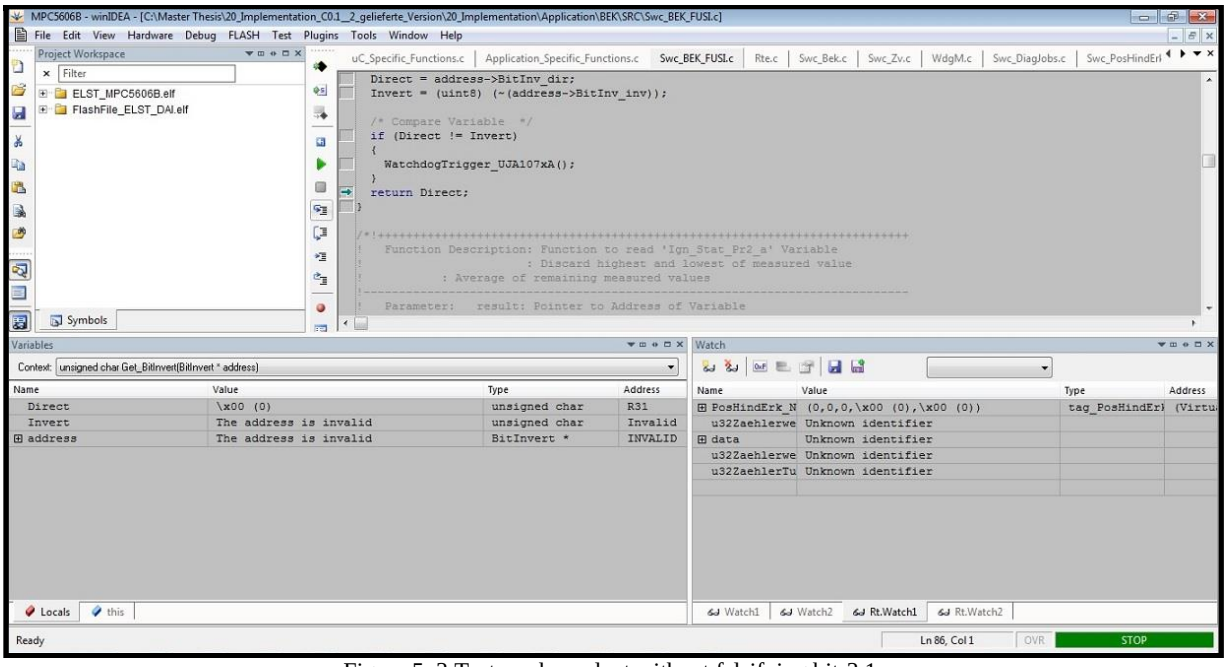


Figure 5. 3 Test work product without falsifying bit-3.1

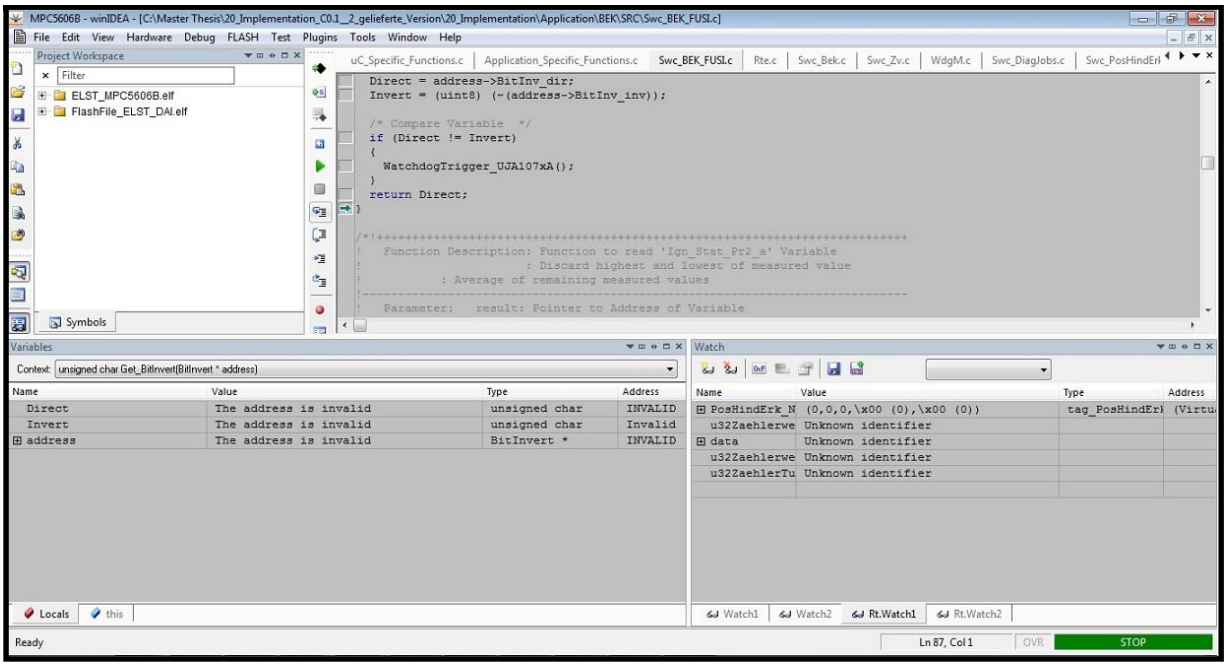


Figure 5. 4 Test work product without falsifying bit-3.2

4. The program control now shifts to the function that writes to the RTE i.e. ‘Rte_Write_ppBedienbefehlZv_BedienbefehlZv ( )’ as in figure 5.5.

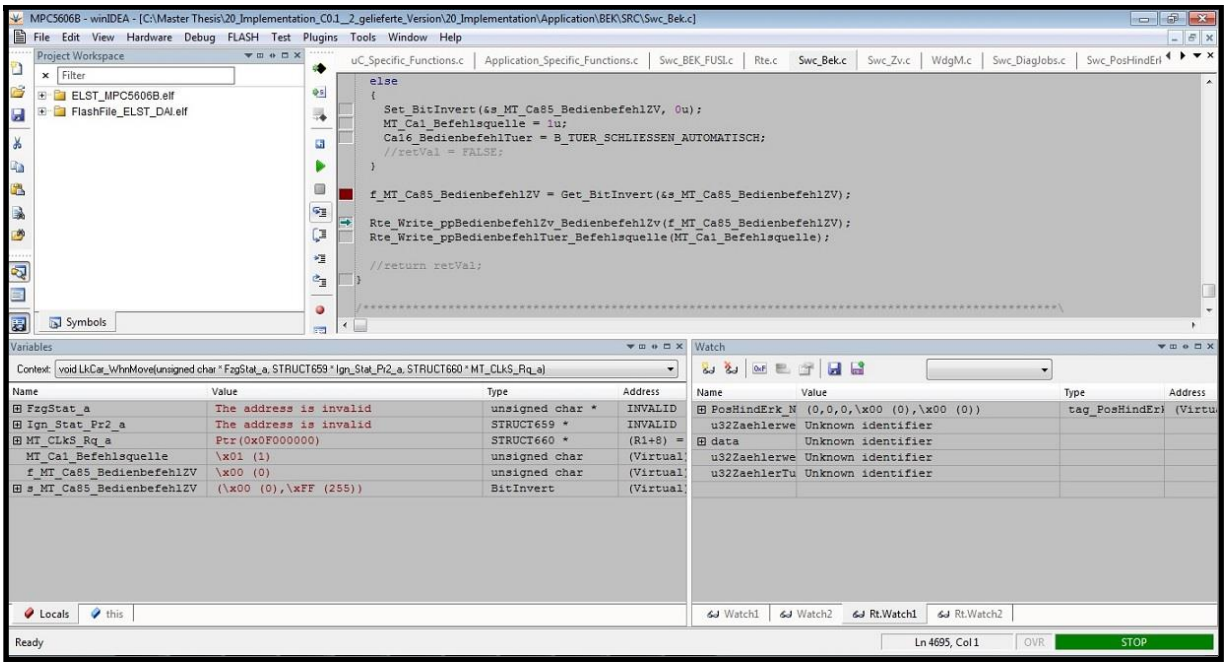


Figure 5. 5 Test work product without falsifying bit-4

The test work products for the mechanism of error handling with falsifying the bit at the software architectural level that was implemented on an AUTOSAR compliant ECU by reciprocal comparison is mentioned as follows:

1. At the breakpoint of function call of ‘Get_BitInvert ( )’ as in figure 5.6.

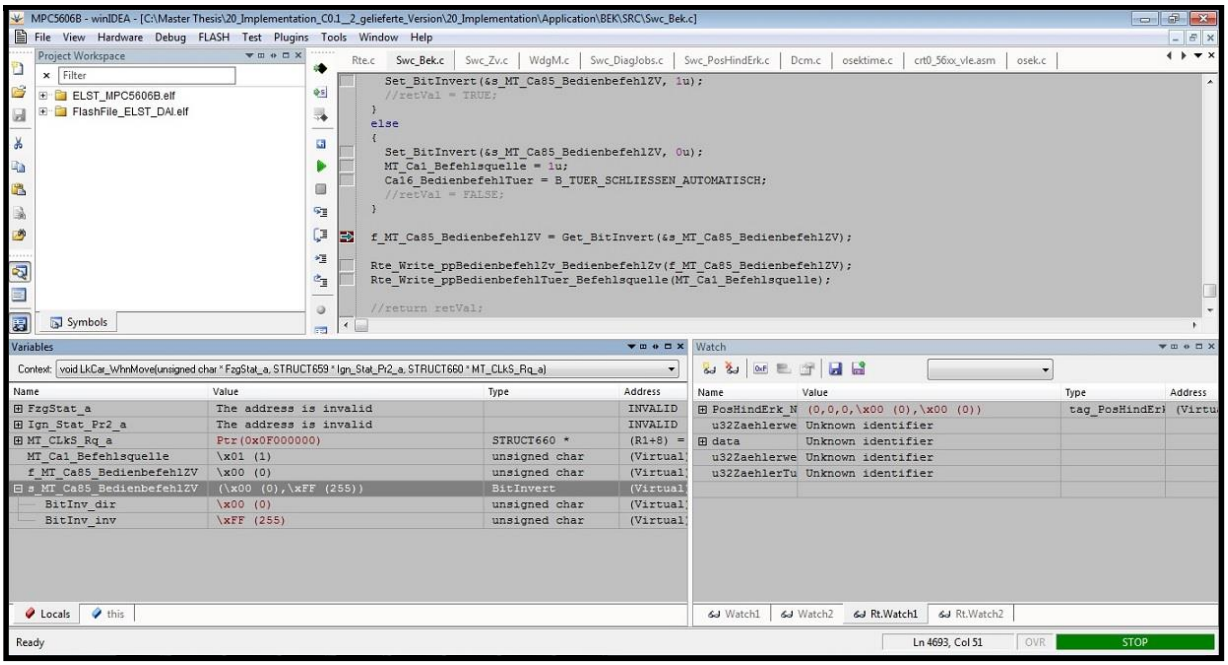


Figure 5. 6 Test work product with falsifying bit-1

2. The value of ‘BitInv_dir’ is deliberately falsified and changed to a random numeral. So now the values of ‘BitInv_dir’ and ‘BitInv_inv’ are no longer each other’s complement as in figure 5.7.

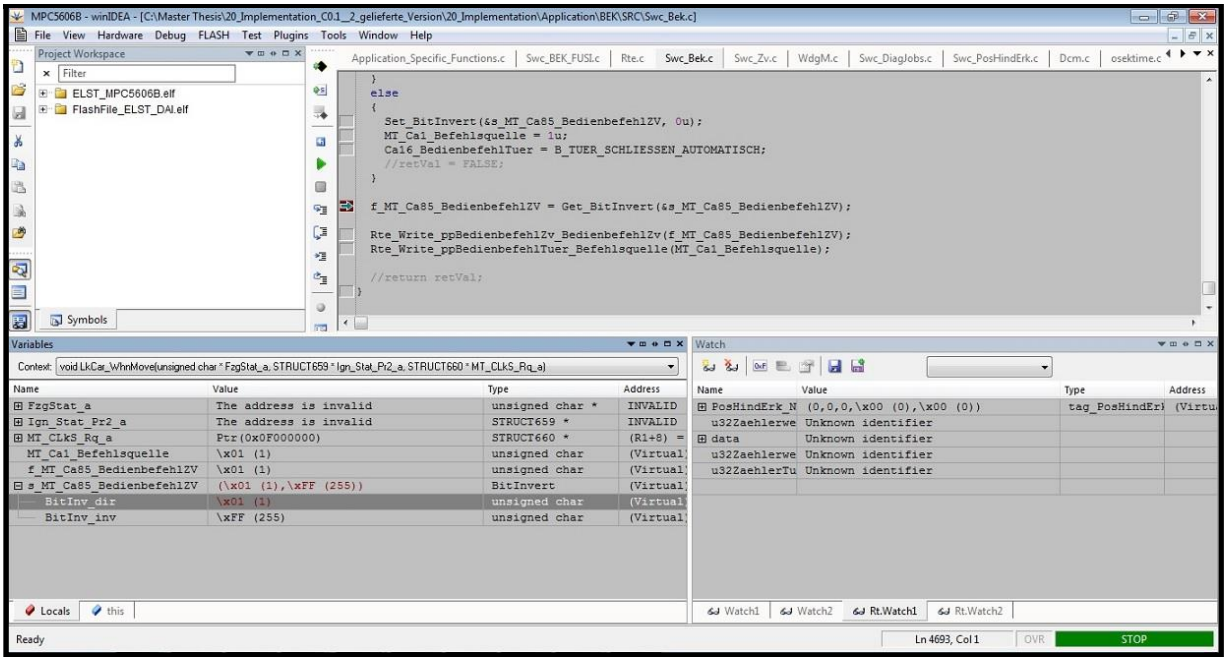


Figure 5. 7 Test work product with falsifying bit-2

3. The program control shifts to the function ‘Get_BitInvert ( )’ as in figure 5.8.

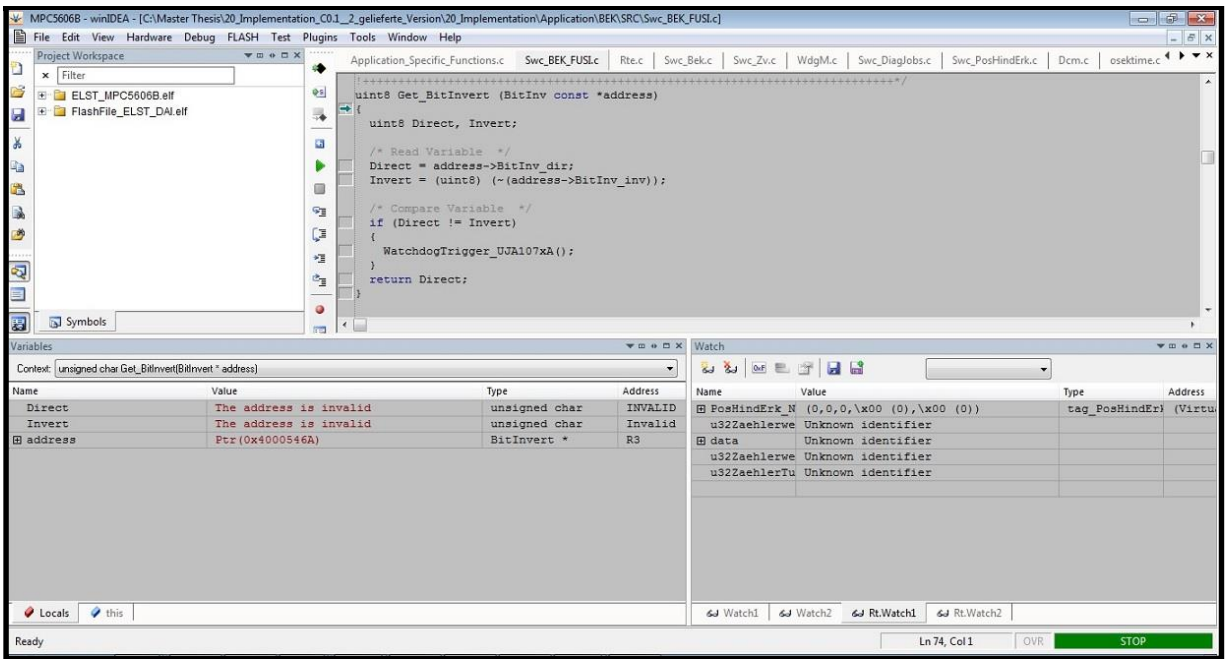


Figure 5. 8 Test work product with falsifying bit-3

4. Now as the values of ‘Direct’ and ‘Invert’ are unequal, the program control shifts to the if statement of the function ‘Get_BitInvert ( )’ as in figure 5.9.

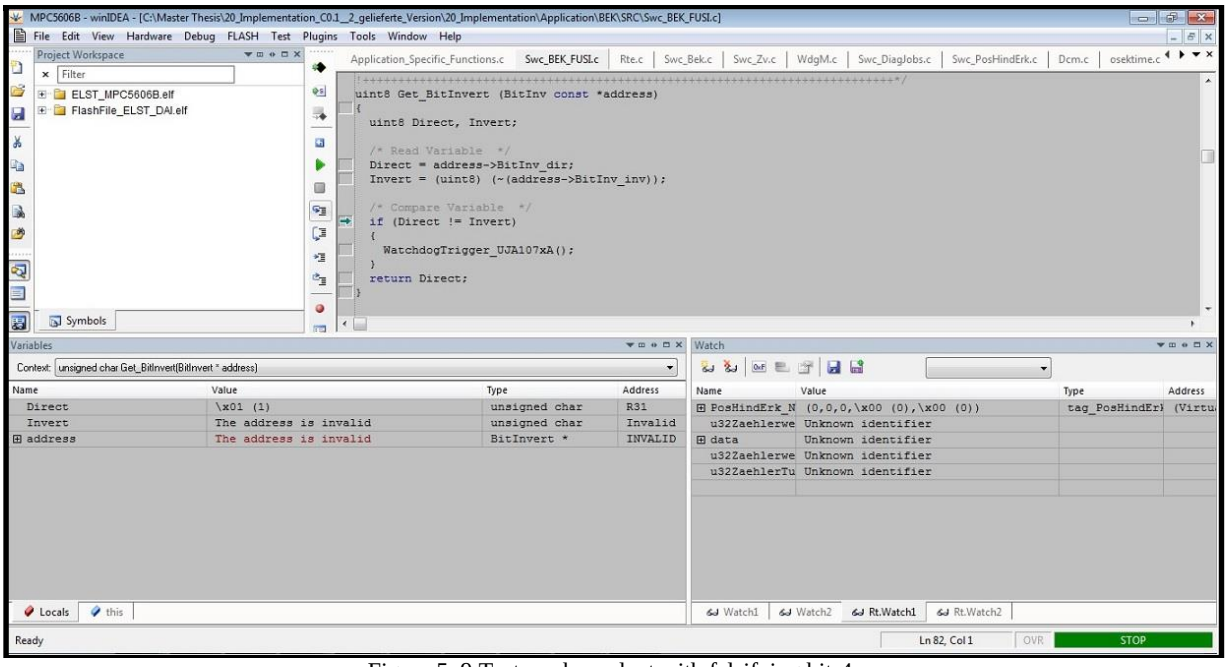


Figure 5. 9 Test work product with falsifying bit-4

5. And as desired it shifts to the function that triggers the watchdog i.e. ‘WatchdogTrigger_UJA107xA ( )’ as in figure 5.10.

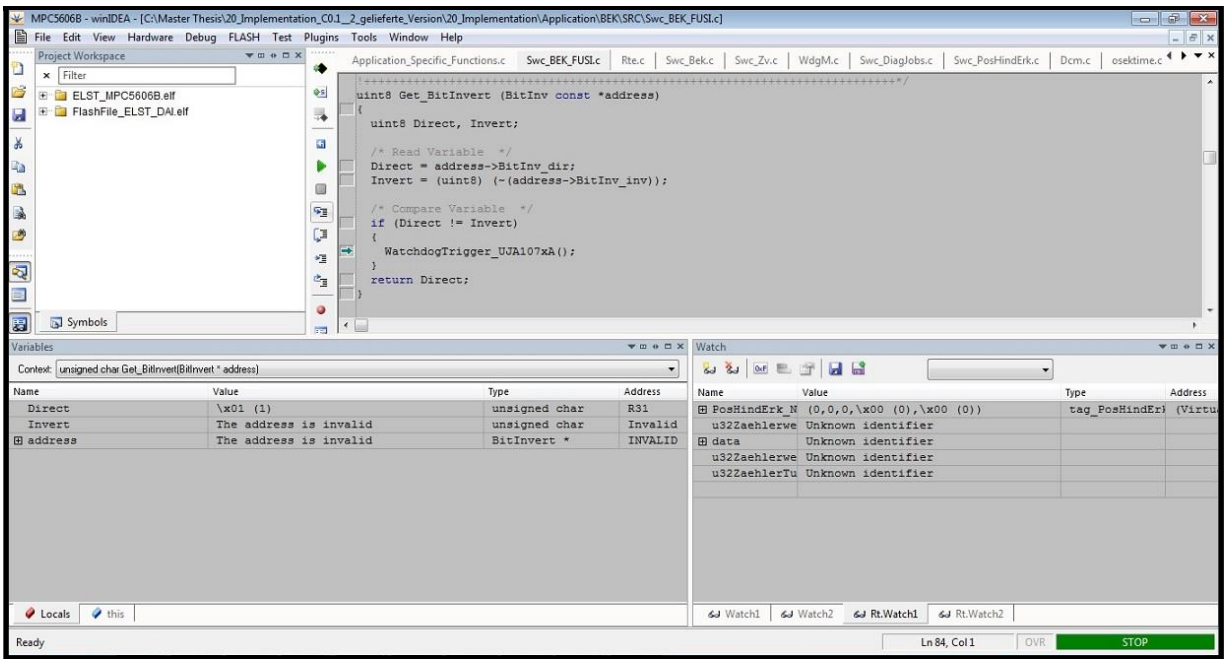


Figure 5. 10 Test work product with falsifying bit-5

6. And finally the program control shifts to the watchdog trigger unction of ‘WatchdogTrigger_UJA107xA ( )’ and the functionality stated under it is performed as in figure 5.11.

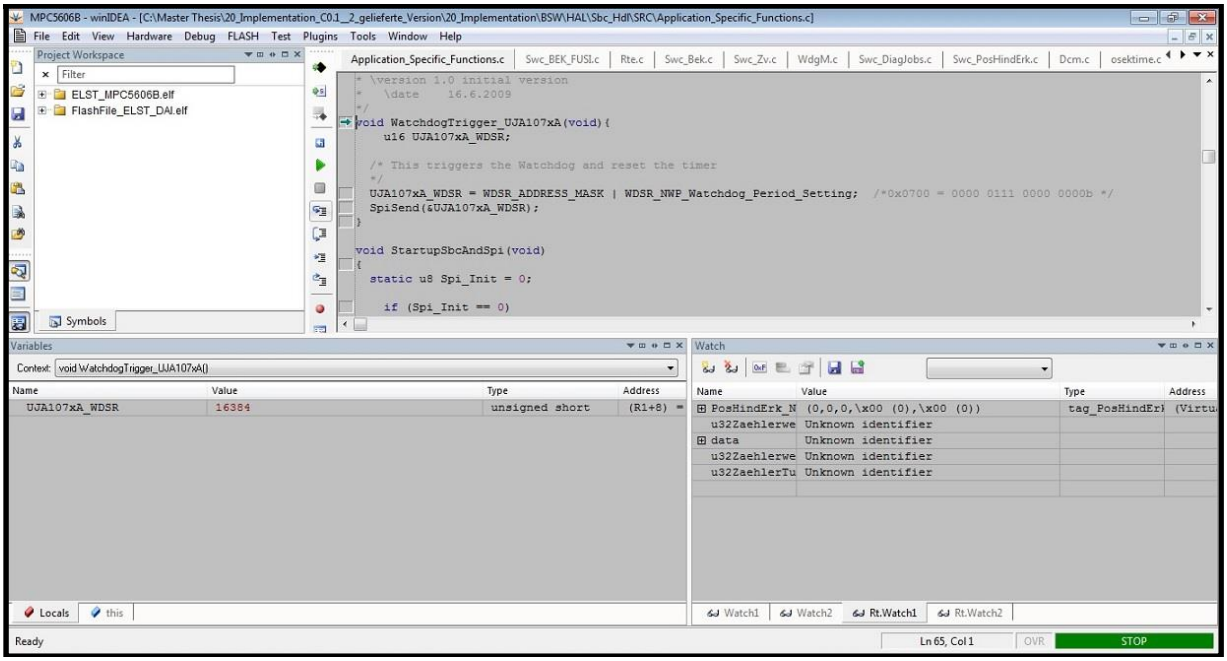


Figure 5. 11 Test work product with falsifying bit-6

6. Results and Potential Directions

The final section of this master thesis gives an insight to the results achieved. It gives a look at all the points that were taken into account to reach the result and the process involved in the same. It serves as a gist to the whole master thesis and can very well stand alone to speak about it. Any research is considered incomplete until and unless a peek into the future is provided. The final section gives you a perspective to all the possible future directions that this thesis can be turned into. It also gives other advancements that could help aid the true purpose of ISO 26262 in an efficient and easier manner.

6.1 Results

Safety in the automotive domain being the call of the hour led to the successful completion of this master thesis. It mainly laid stress on the analysis of the ISO 26262 functional safety standard. But firstly this involved understanding the main source from where the ISO 26262 is derived and that is the IEC 61508. This thesis also contemplates on various tools and technologies that were used during the journey of this thesis. Jumping onto the core motive of understanding the different parts of the ISO 26262 functional safety standard and their links amongst themselves alongside the V-model. The most noticeable part of this was making sure that everything that is mentioned in the ISO 26262 is interpreted in the correct manner and no personal or wrong notions were formed about the same.

The thesis explains the different terms and abbreviations used in the ISO 26262 standard that are mentioned under the Part 1: Vocabulary. Followed by the initial steps that need to be taken at the beginning of the product lifecycle which comes under Part 2: Concept phase. It is highly important to take care of safety right from the beginning because of the simple fact that it becomes easy for the rest of the process to fall into place and to achieve the penultimate goal of safety corresponding to the whole process. The next stage that falls is the Part 4: Product development at the system level, and reads about the design steps that should be necessarily involved during the designing of the system. The next two parts that follow correspond to the major development activities that are carried out at the hardware and software level. One noticeable point of these parts is that, these parts should ideally be worked upon simultaneously. This is because of fact that both of these parts are highly inter-linked and share a lot of work-products. Keeping the conscious of this thesis in mind the Part 6: Product development at the software level was thoroughly analysed. This part speaks volumes about the points to ponder upon when designing, verifying and validation software, that needs to adhere to a particular ASIL level. This clearly involves certain mechanisms that help in achieving the concerned safety level.

The foremost important aspect of determining a particular ASIL level which corresponds to a particular requirement of an ECU comprising of the whole system is all mentioned in the Part 9: Automotive Safety Integrity Level (ASIL)-oriented and safety-oriented analysis. This becomes tricky when big and complex systems are involved but a sustainable way of achieving the same doesn't belong to no-man's land. With diving into all these parts of the ISO 26262 one can be assured to firstly achieve the desired ASIL level, which are the building blocks of this standard and to guarantee the safety which is much needed in today's world of ever-growing technology.

6.2 Potential Directions

Directing this master thesis into a variety of avenues would be extremely beneficial as far as the automotive domain is concerned with safety. The possible ways that have been described are a mixture of present industry beliefs and a personal perspective. Although all of them have not been spoken about with respect to the ISO 26262 functional safety standard keeping in mind the scope of this thesis.

The major issue that lies with the ISO 26262 functional safety standard is the right interpretation of all the parts keeping insight their effects on the previous and the following parts. More rigorous measures need to be taken so as to make sure that the intended is understood and further implemented by the reader. This especially effects the requirements engineering. As requirements engineering is not very experienced when it comes to the automotive domain. This naturally reflects in disability to detect the right quantity and amount of details in the requirements owing full respect to the designated ASIL level. This issue needs to be pondered upon and a solution should be seeped for the same. Which may result in a much more detailed extension of the ISO 26262 with respect to all its parts.

The ISO 26262 functional safety standard is not for prototypes but only for final products. This greatly hampers its usage in the industry because it does not support the proven trend of prototype-to-product transition. So once the prototype is built, introducing the ISO 26262 standard in it becomes a difficult and less efficient process. So, a prequel to the ISO 26262 standard that describes the measures that need to be taken care of while designing a prototype would be very beneficial. The prequel should also talk about how the prototype can transfer itself into the ISO 26262 standard, so as to build the final product. It would encourage the further use of the ISO 26262 safety standard in the automotive industry.

Lastly, with the exponential advancements in autonomous vehicles and as this standard does not comment anything about them. It would be great if the ISO 26262 could have details and activities to be executed when such automobiles are being considered. It would further add up to its core motive which would mean that safety would never be overlooked ever again as far as the automotive domain is concerned.

Bibliography

[1] World Heatlh Organization, „http://www.who.int,“ 2004. [Online]. Available: http://www.who.int/violence_injury_prevention/publications/road_traffic/world_report/en/.

[2] Freescale Semiconductors GmbH, „http://www.freescale.com/SafeAssure,“ 2013. [Online]. Available: http://cache.nxp.com/files/microcontrollers/doc/white_paper/FCTNLSFTYWP.pdf.

[3] C. Ebert und A. Braatz, „http://www.vector.com/safety,“ 2015. [Online]. Available: http://vector.com/portal/medien/cmc/events/Webinars/2015/Vector_Webinar_FunctionalSafety_Principles_and_Practice_20150428_EN.pdf.

[4] IEC ACOS, [Online]. Available: http://storage.googleapis.com/wzukusers/user-17356013/images/564cd8fedae86goXcG5R/IEC61508_200.png.

[5] Exida .Inc, „http://www.exida.com/,“ 2006. [Online]. Available: http://www.win.tue.nl/~mvdbrand/courses/sse/1213/iec61508_overview.pdf.

[6] A. Bärwald, „https://www.tuv-sud.com,“ 2014. [Online]. Available: https://www.tuv-sud.com/home-com/resource-centre/publications/white-papers-e-books/iso-26262-compliance.

[7] D. Crolla, Encyclopedia of Automotive Engineering, 2015.

[8] AUTOSAR, „https://www.autosar.org,“ [Online]. Available: https://www.autosar.org/fileadmin/images/media_pictures/AUTOSAR-components-and-inte.jpg.

[9] Conplement AG, „http://www.cp-embedded.de,“ [Online]. Available: http://www.cp-embedded.de/userdir/cms/images/autosar.jpg.

[10] AUTOSAR, „https://www.autosar.org,“ [Online]. Available: https://www.autosar.org/fileadmin/_processed_/csm_AUTOSAR_Methodology_b_72b1b0cbe8.jpg.

[11] Programming Research Ltd., „http://www.programmingresearch.com,“ [Online]. Available: http://www.programmingresearch.com/wp-content/uploads/2016/04/From-the-Authors-of-MISRA-C.png.

[12] MISRA, „https://www.misra.org.uk,“ [Online]. Available: https://www.misra.org.uk/MISRAHome/MISRAC2012/tabid/196/Default.aspx.

[13] Sparx Systems Pty Ltd., „http://www.sparxsystems.com,“ [Online]. Available: http://www.sparxsystems.com/products/ea/.

[14] Scientific Toolworks, Inc., „https://scitools.com,“ [Online]. Available: https://scitools.com/features/.

[15] Programming Research Ltd., „http://www.programmingresearch.com,“ 2015. [Online]. Available: http://www.programmingresearch.com/content/literature/PRQA-QAC.pdf.

[16] iSYSTEM AG, „http://www.isystem.com,“ [Online]. Available: http://www.isystem.com/products/software/winidea.

[17] Vector Informatik GmbH, „https://vector.com,“ [Online]. Available: https://vector.com/portal/medien/cmc/factsheets/DaVinci_Configurator_Pro_FactSheet_EN.pdf.

[18] Vector Informatik GmbH, „http://vector.com,“ [Online]. Available: http://vector.com/portal/medien/cmc/factsheets/CANoe_FactSheet_EN.pdf.

[19] The International Organization for Standardization, Road Vehicles - Functional safety - Part 1: Vocabulary, 2011.

[20] The International Organization for Standardization, Road Vehicles - Functional safety - Part 3: Concept Phase, 2011.

[21] The International Organization for Standardization, Road Vehicles - Functional safety - Part 4: Product development at the system level, 2011.

[22] The International Organization for Standardization, Road Vehicles - Functional safety - Part 6: Product development at the software level, 2011.

[23] The International Organization for Standardization, Road Vehicles - Functional safety - Part 9: Automotive Safety Integrity Level (ASIL)-oriented and safety-oriented analysis, 2011.

[24] M. Barr, „http://www.barrgroup.com,“ 2000. [Online]. Available: http://www.barrgroup.com/Embedded-Systems/How-To/CRC-Calculation-C-Code.

[25] Micron Technology, „https://www.micron.com,“ 2012. [Online]. Available: https://www.micron.com/~/media/documents/.../esc_2012_wp_ecc_embedded.pdf.

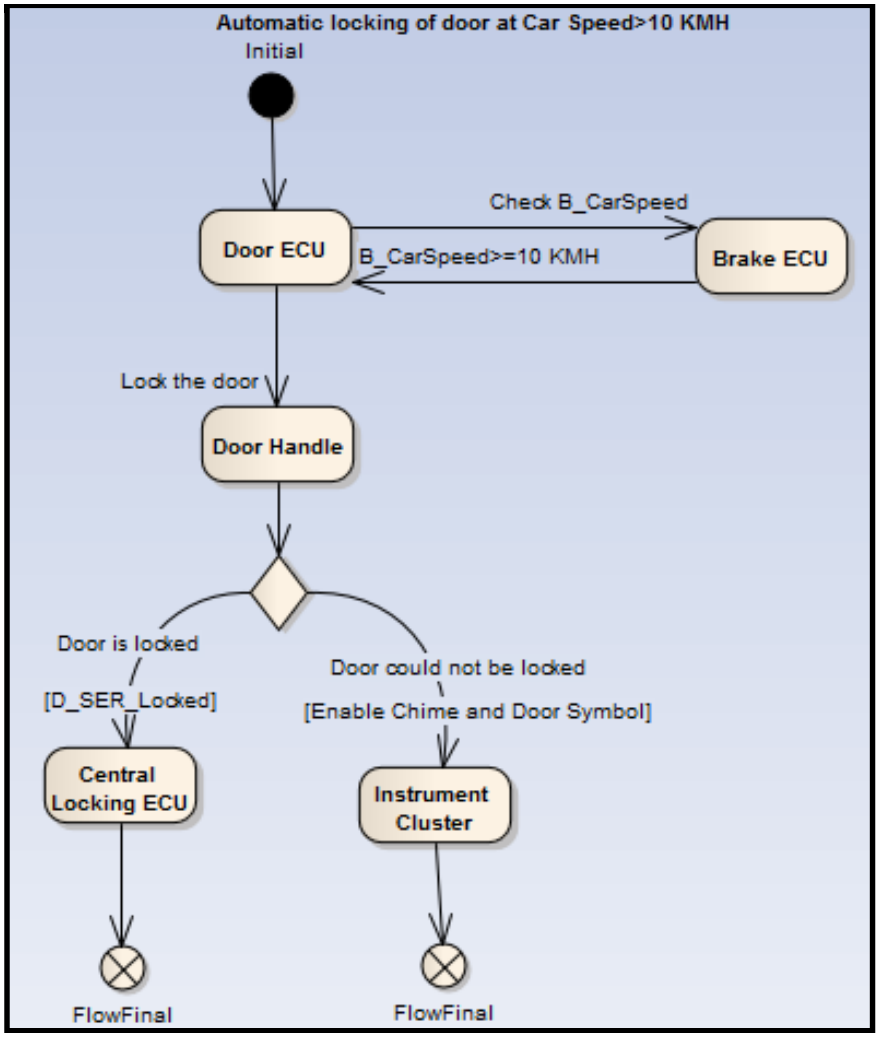
[26] S. Suzuki und K. G. Shin, „On Memory Protection in Real-Time OS“.

[27] M. Dipperstein, „http://michael.dipperstein.com,“ 2014. [Online]. Available: http://michael.dipperstein.com/hamming/.

Appendix A

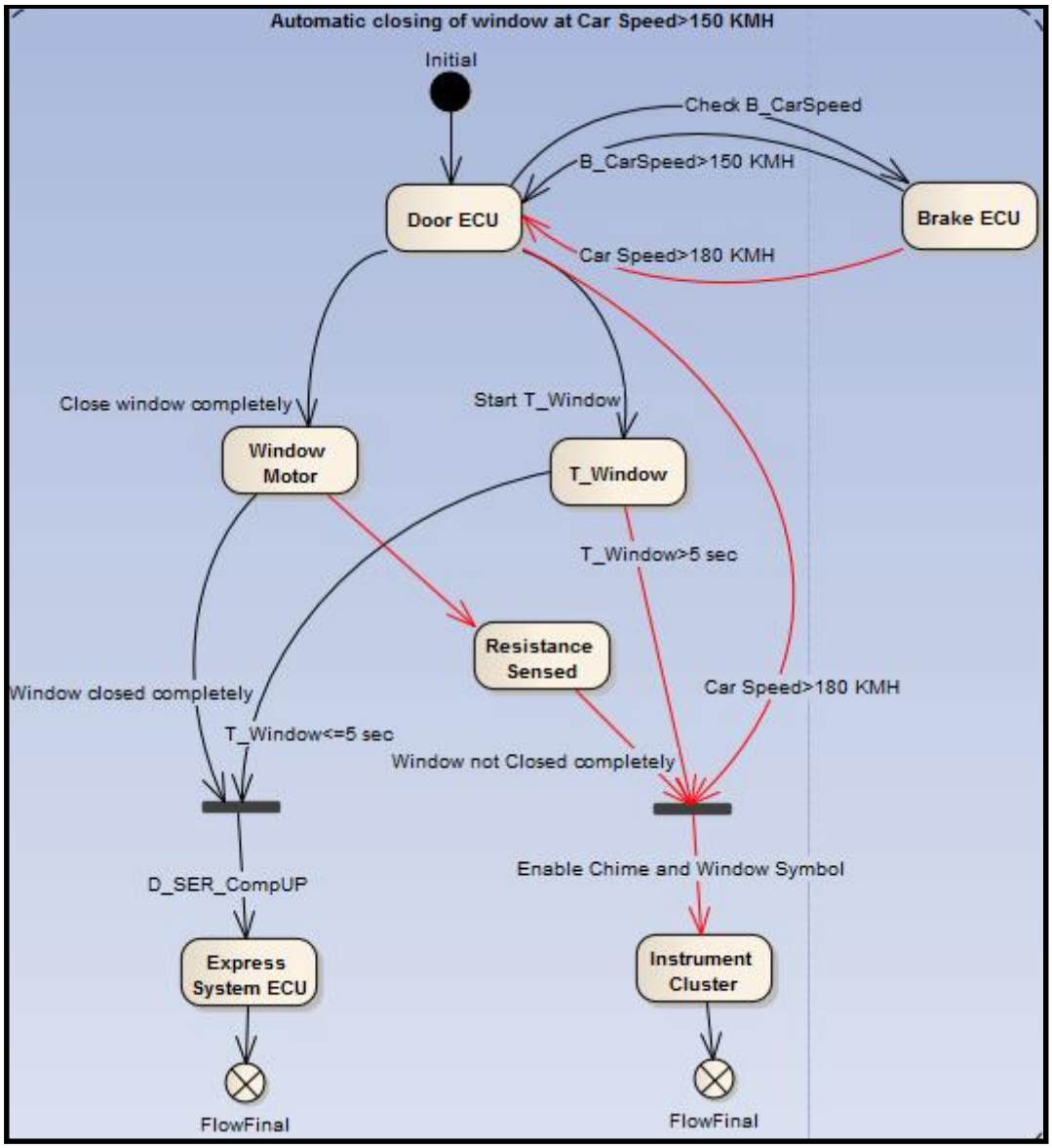
A.1 Activity Diagram

This sub-section of this thesis consists of the activity diagrams built according to the specified requirements for the sample system. One of the activity diagram for the locking/unlocking mechanism of the door is as follows:



A. 1 Activity diagram of door

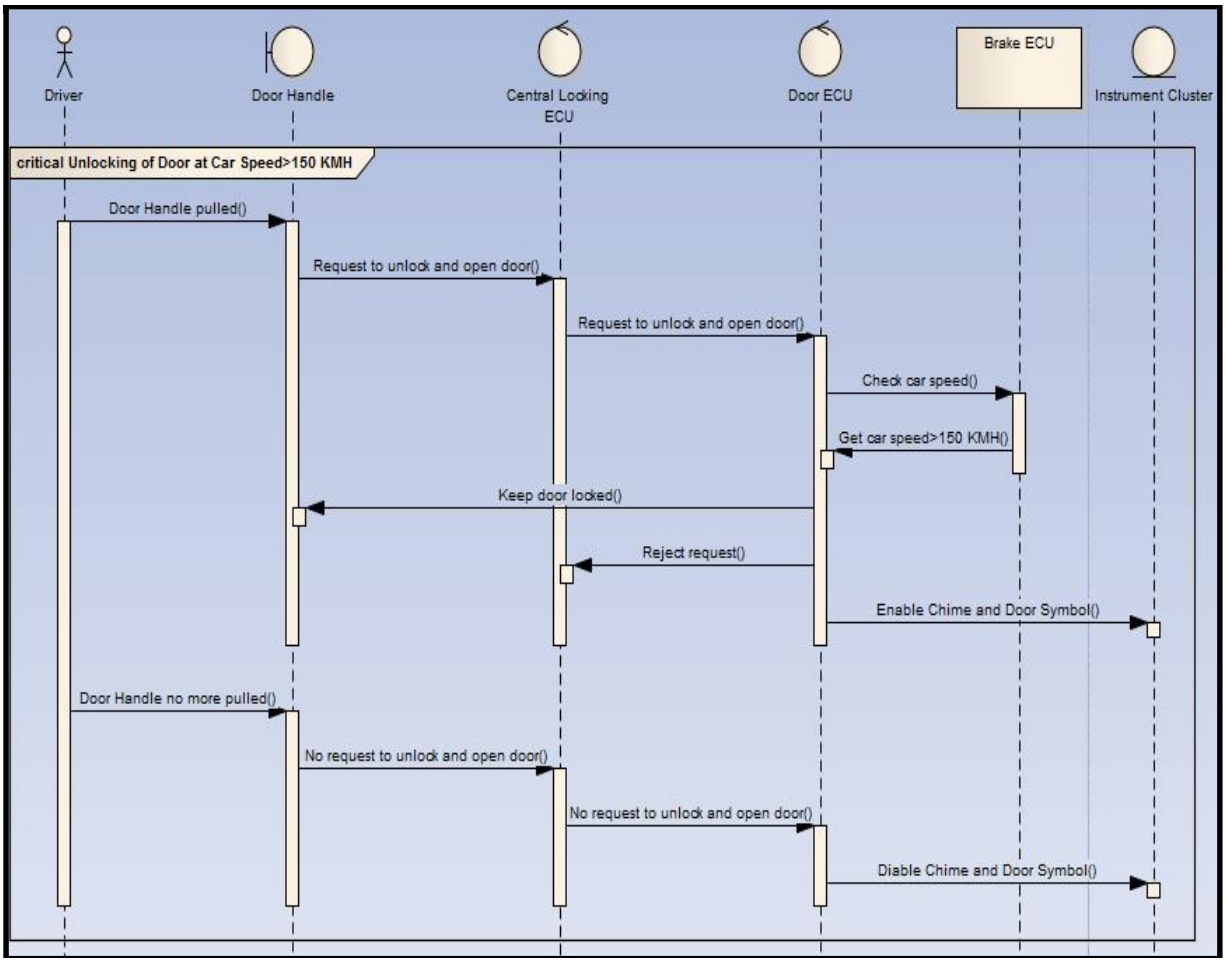
The activity diagram of a requirement for the opening/closing mechanism of the window is as follows:



A. 2 Activity Diagram of window

A.2 Sequence Diagram

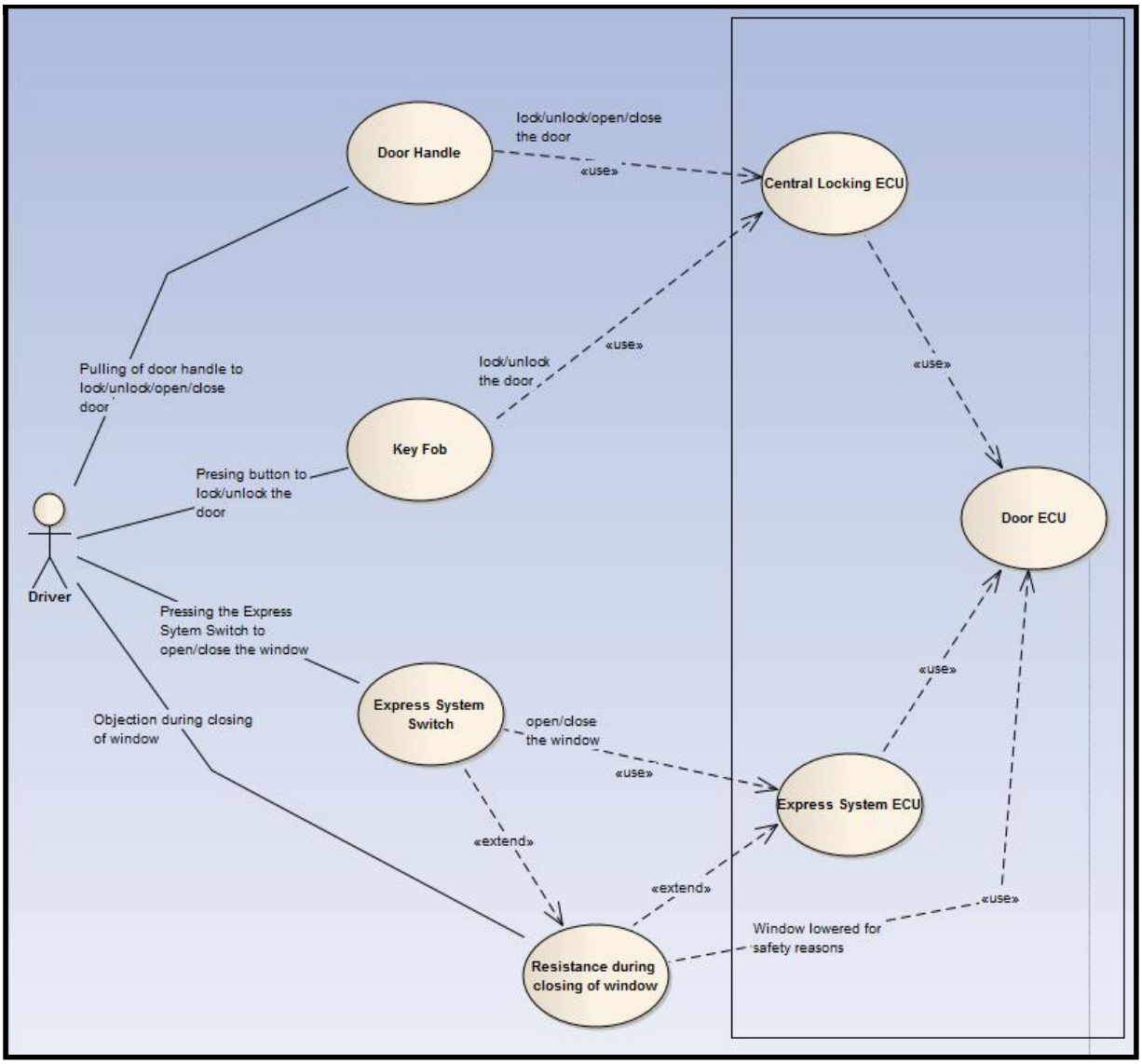
This section of this thesis consists of the sequence diagram built according to the specified requirements for the sample system. One of the sequence diagram for the locking/unlocking mechanism of the door is as follows:



A. 3 Sequence diagram of door

A.3 Use-Case Diagram

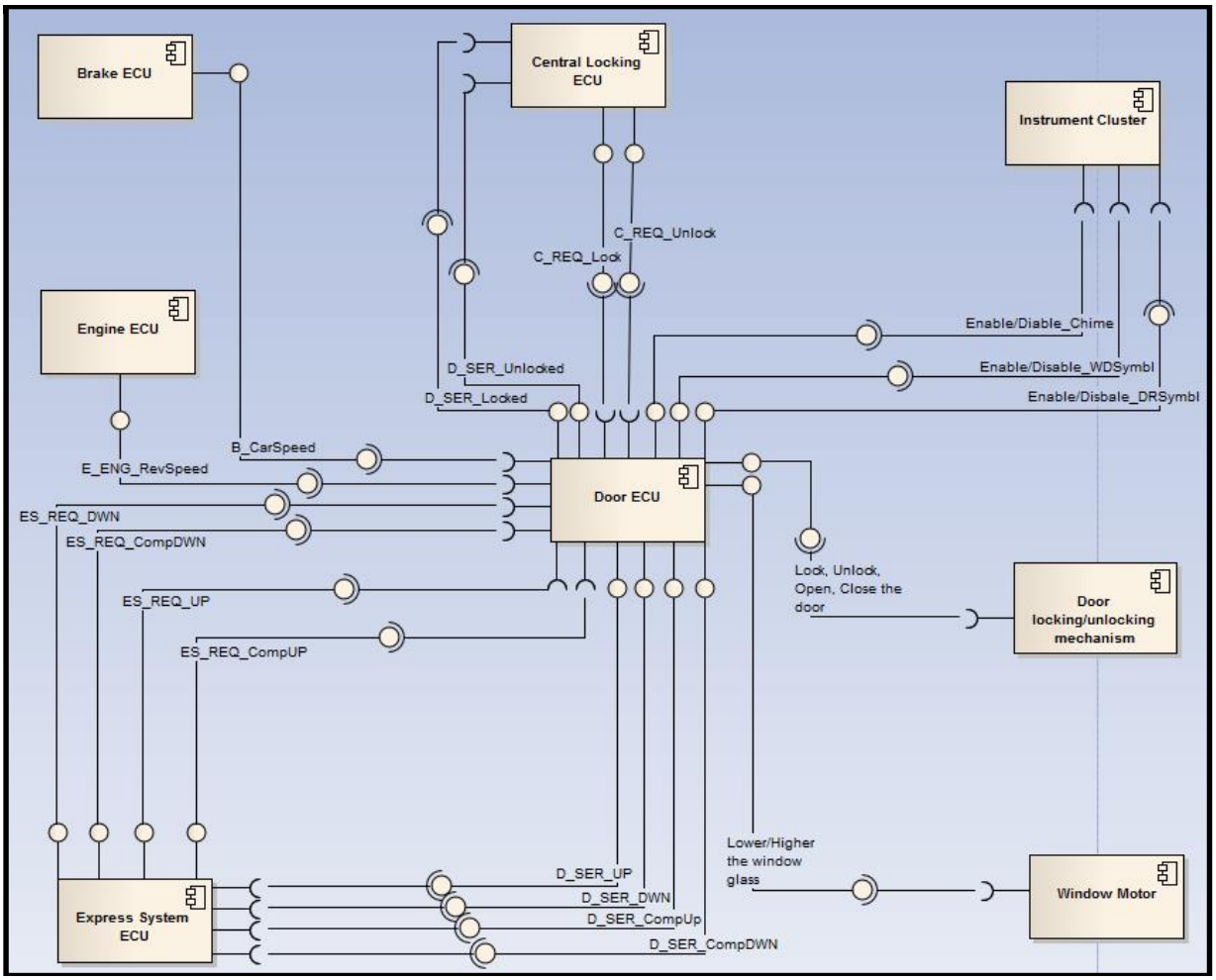
This sub-section of this thesis consists of the use-case diagram built according to the specified requirements for the sample system. The use-case diagram for the sample system is as follows:



A. 4 Use-case diagram

A.4 Component Diagram

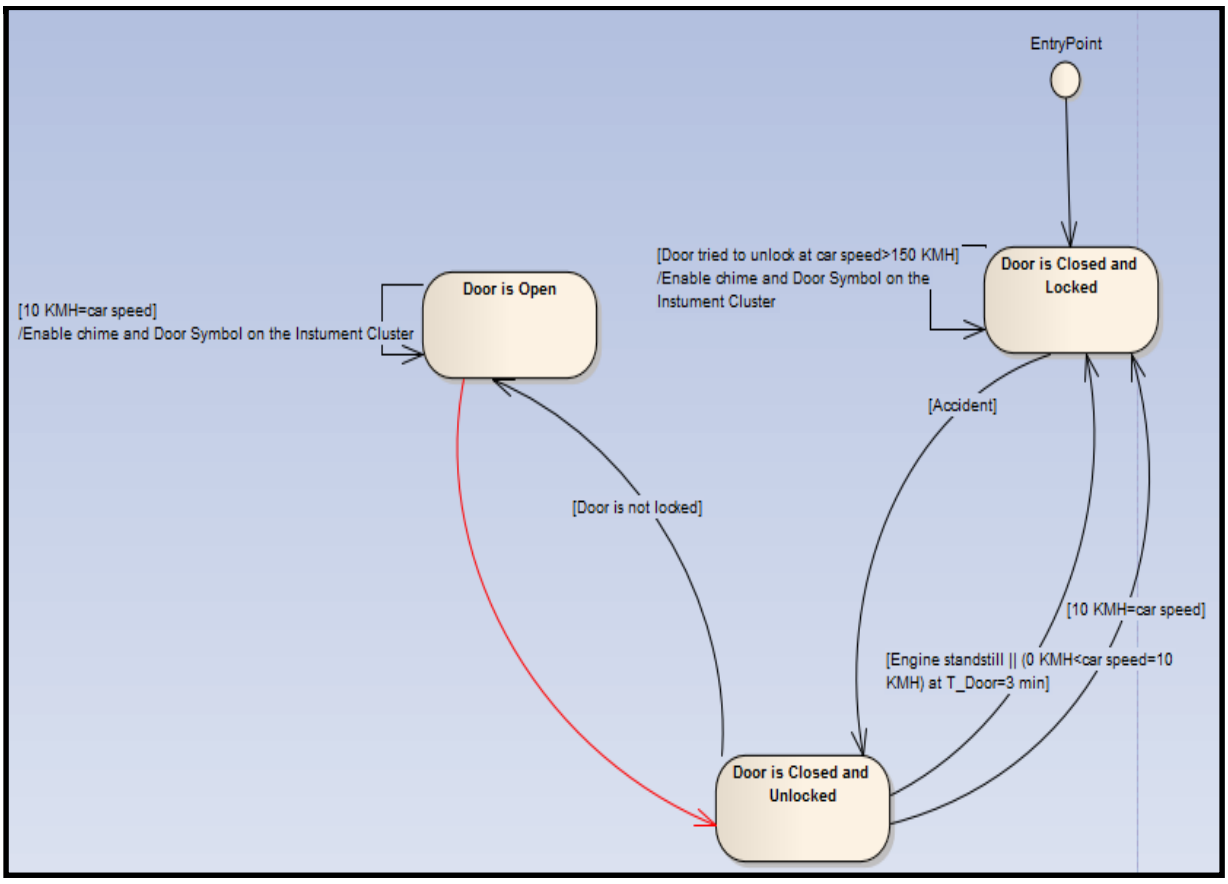
This sub-section of this thesis consists of the component diagram built according to the specified requirements for the sample system. The component diagram for the sample system is as follows:



A. 5 Component diagram

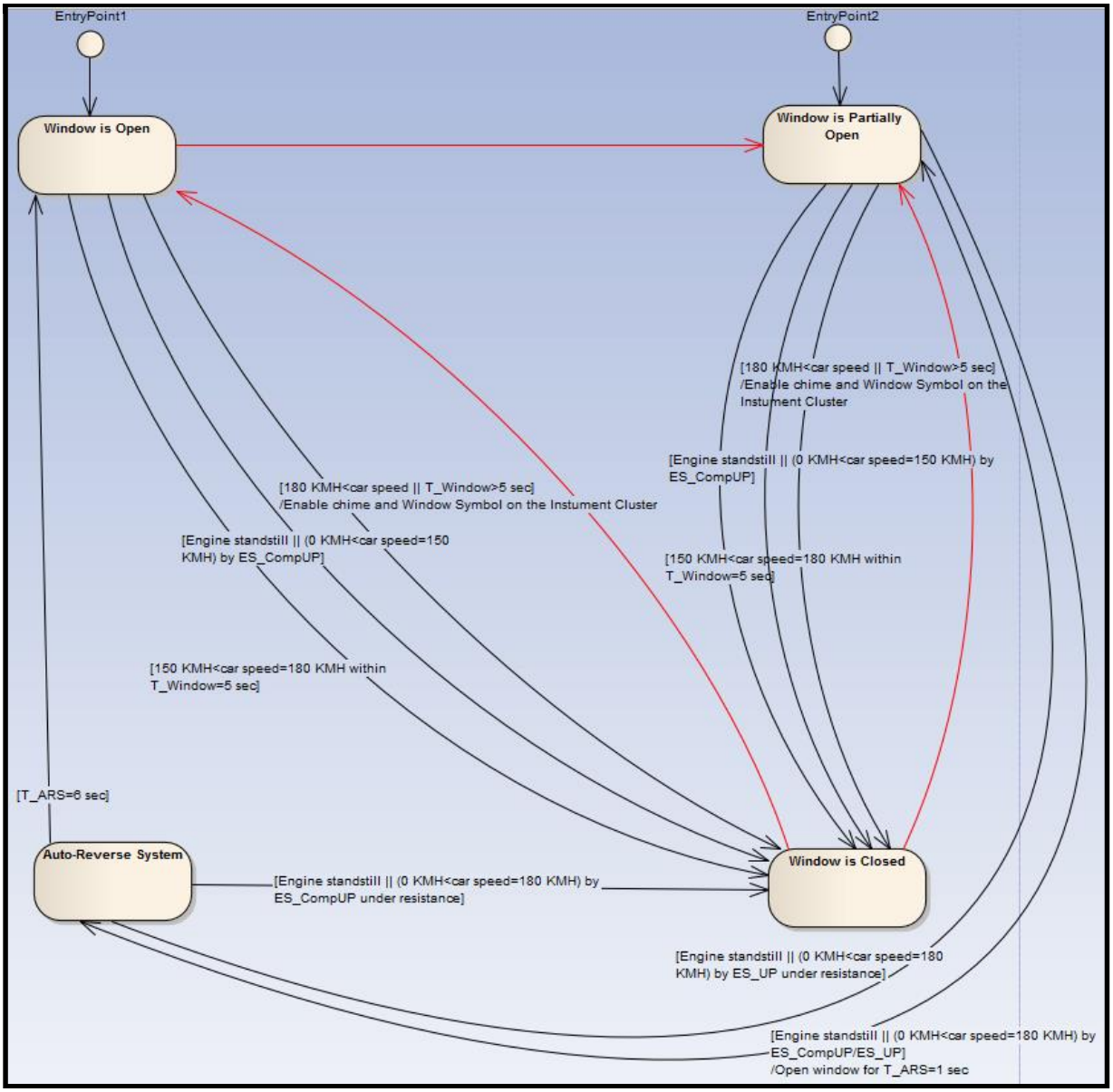
A.5 State Machine Diagram

This sub-section of this thesis consists of the state machine diagrams built according to the specified requirements for the sample system. The state machine diagram of the door for the sample system is as follows:



A. 6 State machine diagram of door

The state machine diagram of the window for the sample system is as follows:



A. 7 State machine diagram of window

Selbstständigkeitserklärung

TECHNISCHE UNIVERSITÄT CHEMNITZ

Studentenservice – Zentrales Prüfungsamt
Selbstständigkeitserklärung

Name: Laval Vorname: Vibhu geb. am: 07.07.1991 Matr.-Nr.: 386681	**Bitte beachten:** 1. Bitte binden Sie dieses Blatt am Ende Ihrer Arbeit ein.

Selbstständigkeitserklärung*

Ich erkläre gegenüber der Technischen Universität Chemnitz, dass ich die vorliegende Masterarbeit selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe.

Die vorliegende Arbeit ist frei von Plagiaten. Alle Ausführungen, die wörtlich oder inhaltlich aus anderen Schriften entnommen sind, habe ich als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch bei keinem anderen Prüfer als Prüfungsleistung eingereicht und ist auch noch nicht veröffentlicht.

Datum: ..... Unterschrift: .....

* Statement of Authorship

I hereby certify to the Technische Universität Chemnitz that this thesis is all my own work and uses no external material other than that acknowledged in the text.

This work contains no plagiarism and all sentences or passages directly quoted from other people's work or including content derived from such work have been specifically credited to the authors and sources.

This paper has neither been submitted in the same or a similar form to any other examiner nor for the award of any other degree, nor has it previously been published.