Institutionen för datavetenskap
Department of Computer and Information Science

Examensarbete

Development of an ISO 26262 ASIL D compliant verification system

Daniel Carlsson

LIU-IDA/LITH-EX-A-13/005-SE
Linköping 2013

Linköpings universitet
TEKNISKA HÖGSKOLAN

Institutionen för datavetenskap
Linköpings universitet
581 83 Linköping

Institutionen för datavetenskap
Department of Computer and Information Science

Examensarbete

Development of an ISO 26262 ASIL D compliant verification system

Daniel Carlsson

LIU-IDA/LITH-EX-A-13/005-SE
Linköping 2013

Handledare: **Breeta SenGupta**
IDA, Linköpings universitet
Viacheslav Izosimov
Semcon Caran AB
Urban Ingelsson
Semcon Caran AB

Examinator: **Zebo Peng**
IDA, Linköpings universitet

Institutionen för datavetenskap
Linköpings universitet
581 83 Linköping

Avdelning, Institution
Division, Department

Software and Systems
Department of Computer and Information Science
SE-581 83 Linköping

Datum
Date

2013-03-01

Språk Language	**Rapporttyp** Report category	**ISBN** —
☐ Svenska/Swedish	☐ Licentiatavhandling	**ISRN** LIU-IDA/LITH-EX-A-13/005-SE
☒ Engelska/English	☒ Examensarbete	
	☐ C-uppsats	**Serietitel och serienummer** Title of series, numbering
	☐ D-uppsats	**ISSN** —
☐ ________	☐ Övrig rapport	
	☐ ________	

URL för elektronisk version

http://www.ep.liu.se

Titel
Title

Development of an ISO 26262 ASIL D compliant verification system

Författare
Author

Daniel Carlsson

Sammanfattning
Abstract

In 2011 a new functional safety standard for electronic and electrical systems in vehicles was published, called ISO 26262. This standard concerns the whole lifecycle of the safety critical elements used in cars, including the development process of such elements. As the correctness of the tools used when developing such an element is critical to the safety of the element, the standard includes requirements concerning the software tools used in the development, including verification tools. These requirements mainly specify that a developer of a safety critical element should provide proof of their confidence in the software tools they are using. One recommended way to gain this confidence is to use tools developed in accordance to a "relevant subset of [ISO 26262]".

This project aims to develop a verification system in accordance to ISO 26262, exploring how and what specifications should be included in this "relevant subset" of ISO 26262 and to which extent these can be included in their current form. The work concludes with the development of a single safety element of the verification system, to give an demonstration of the viability of such a system.

Nyckelord
Keywords

ISO 26262, Verification Systems, Tool Qualification, Automotive Embedded Systems, Functional Safety

Abstract

In 2011 a new functional safety standard for electronic and electrical systems in vehicles was published, called ISO 26262. This standard concerns the whole lifecycle of the safety critical elements used in cars, including the development process of such elements. As the correctness of the tools used when developing such an element is critical to the safety of the element, the standard includes requirements concerning the software tools used in the development, including verification tools. These requirements mainly specify that a developer of a safety critical element should provide proof of their confidence in the software tools they are using. One recommended way to gain this confidence is to use tools developed in accordance to a "relevant subset of [ISO 26262]".

This project aims to develop a verification system in accordance to ISO 26262, exploring how and what specifications should be included in this "relevant subset" of ISO 26262 and to which extent these can be included in their current form. The work concludes with the development of a single safety element of the verification system, to give an demonstration of the viability of such a system.

Acknowledgments

This thesis concludes my studies in Electrical Engineering at Linköping University. The project described herein was performed at Semcon Caran AB in Linköping in 2012.

I would like to thank my supervisors and colleagues, Urban Ingelsson and Viacheslav Izosimov, for their supervision and a lot of interesting discussions during the run of the project, I have learned a lot from you both, not only on the subject of functional safety. I would also like to thank my examiner and my supervisor at Linköping University, Professor Zebo Peng and Breeta Sen-Gupta, for their comments during the project. My colleague Andreas Wallin at Semcon also deserves thanks for his help during the project, especially his involvement in our discussions on how to interpret the more complicated clauses of ISO 26262.

I would also like to thank Semcon in Linköping, primarily for giving me the opportunity to do this thesis project but also for making me feel very welcome from day one. A special thanks is extended to Jan-Erik Mathisen, who as my manager during and after the project have provided me with interesting opportunities.

Special thanks to my family for supporting me during my studies, helping me out whenever possible in different ways.

Thanks also the Yi class of 2012, making my five years at Linköping University more enjoyable. I've really appreciated spending time at the university with all of you, and it would not have been the same without you.

Contents

List of figures

List of tables

Abbreviations

The following abbreviations have been used in the project and are subsequently used in the report.

ADC	Analog-to-Digital Converter
ASIL	Automotive Safety Integrity Level
AWG	Arbitrary Waveform Generator
CAN	Controller Area Network
CNL	Constrained Natural Language
DFMEA	Design Failure Mode and Effect Analysis
ECU	Electronic Control Unit
E/E	Electrical and Electronic
FMEA	Failure Mode and Effect Analysis
FPGA	Field Programmable Gate Array
FSC	Functional Safety Concept
FSR	Functional Safety Requirement
FTA	Fault Tree Analysis
HARA	Hazard Analysis and Risk Assessment
HazOP	Hazard and Operability (study)
HDL	Hardware Description Language
HSI	Hardware-Software Interface
HWSR	Hardware Safety Requirement
ISO	International Organization for Standardization
IEC	International Electrotechnical Commission
LUT	Lookup Table
OEM	Original Equipment Manufacturer

RAM	Random-Access Memory
ROM	Read-Only Memory
SIL	Safety Integrity Level
SG	Safety Goal
SPI	Serial Peripheral Interface
SUT	System Under Test
SWSR	Software Safety Requirement
TCL	Tool Confidence Level
TCX	Test Case X (project specific)
TCY	Test Case Y (project specific)
TD	Tool error Detection
TI	Tool Impact
TMR	Triple Modular Redundancy
TSC	Technical Safety Concept
TSR	Technical Safety Requirement
UART	Universal Asynchronous Receiver/Transmitter
UML	Unified Modeling Language
VHDL	VHSIC Hardware Description Language

Chapter 1

Introduction

This report is the outcome of the Master of Science thesis project "Development of an ISO 26262 ASIL D compliant verification system", performed as an internal project at the company Semcon Caran in Linköping.

1.1 Background

The automotive industry, one of the largest industries in the world, has as most large industries an interest in the latest research and modern technology. In the automotive industry this interest stems from a desire to either to add value to the manufactured vehicles, by adding new functionality such as parking help systems or entertainment systems, or to lower manufacturing costs of the vehicles, by using newer and possibly cheaper and/or smaller components. These two aspects have led to an increasing use of embedded systems in vehicles in the last decades, a development also seen in many other industries.

This development has an important safety aspect, as the shift towards more and more functionality implemented on embedded systems includes the shifting of safety critical functionality, such as steering and braking, to these new embedded systems. Additionally, modern vehicles contain an increasing number of active safety systems, often implemented on embedded systems, such as electronic stability control and collision avoidance systems.

To help make this shift to increased use of embedded systems in modern vehicles safe and not accompanied with increased risks of failures, a functional safety standard for safety critical Electrical and Electronic (E/E) systems in cars was released in 2001. The release of this standard, named ISO 26262, being one of the backgrounds to this project.

Taking a step back and looking at general functional safety, it is simple to see that good and reliable software tools are an important part of developing these safety critical embedded systems. A badly designed software tool, such as a code generation software with unknown bugs, may introduce errors to the developed system. This in turn can make hazards arise in the completed product, if the error is not found and identified in the verification process of the product. The same thing applies to verification tools, as an error let through in the testing means an erroneous product could ship, with risk of creating major hazards.

ISO 26262 therefore includes a chapter about establishing confidence in software tools, tools used to develop safety critical embedded systems. However, this chapter does not differentiate between the different types of tools, such as code generation tools or verification tools, and it mainly consists of a description of the classification scheme used for establishing the level of confidence in the tool. It ends with recommendation that tools in which the confidence is low should be "qualified", using one of four methods. Most of these methods are rather weakly described, leaving much up to interpretation.

One of these qualification methods mentioned for software tools to be used in development of a safety critical system, is to use tools themselves developed with a safety standard. Since, however, no functional safety standards for software tools exist, the standard recommends following itself or a similar standard to the best effort. This is a rather vague specification after motivating the importance of reliable tools earlier in the chapter. It continues with recommending itself as a partially applicable standard, meaning that ISO 26262 should at least be partially applicable to software tools.

This project therefore aims to study this idea of qualification of a software verification system by developing the software verification system in accordance to ISO 26262, to find out if and how ISO 26262 is applicable to software verification tools.

1.2 Objectives

The objectives of the project described in this report are as follows:

1. Identify if a verification system used to test safety critical automotive E/E-systems should be classified as safety critical itself according to ISO 262626, as required by the standard.

2. Perform the requirement elicitation and system design activities of the ISO 26262 process model on such a verification system.

3. Develop and implement an element of such a verification system, such that it satisfies one or several of the elicited safety requirements.

4. Test and evaluate the performance of the implemented element.

All in all these objectives will contribute to provide some answers to the following questions:

- Is a verification system used to test a safety critical automotive E/E-system actually safety critical itself, based on the initial safety analyses of ISO 26262?

- Is it possible to apply ISO 26262 directly to the development activities of a verification tool? If not, what safety activities needs adjustments and how could they be adjusted? How could the possible non-applicable safety activities be handled?

- How should an element of such a verification system be developed to satisfy the ISO 26262 safety requirements?

As a result of the focus on the above objectives and some additional time constraints, mainly originating from the project being the work of a 30 ECTS Master's thesis, some limitations of scope were introduced to the project. These limitations of scope are foremost limitations on the amount of requirements elicited in each step of the elicitation process, with the project being more focused on depth than width. This results in the project aiming to follow the safety requirement elicitation process of ISO 26262 as far as possible, instead of trying to create an as exhaustive collection of high level safety requirements as possible. Some limitations of scope also follow naturally from the objectives of the project, such as the specifications concerning validation of the system. Since validation only can be performed on a complete implementation of a system, which lies well outside the objectives of this project, most validation activities are not performed in this project. This also applied to activities related to the planning and preparation of the verification activities.

1.3 Overview

Since the objective of the project is related to the "first half" of ISO 26262 process model, it is a good idea to start with an overview of ISO 26262 to explain the basic structure of this report. Such an overview of the standard is included in [10], and also included in this report as Figure 1.1. As hinted in the figure, the standard is mainly centered around Part 3 to Part 7 of ISO 26262, parts which in earlier drafts of the standard were called the “core processes”. A close-up of this part of the figure is therefore shown in Figure 1.2, with the clauses least related to the project being grayed out, to show what clauses this project will focus on.

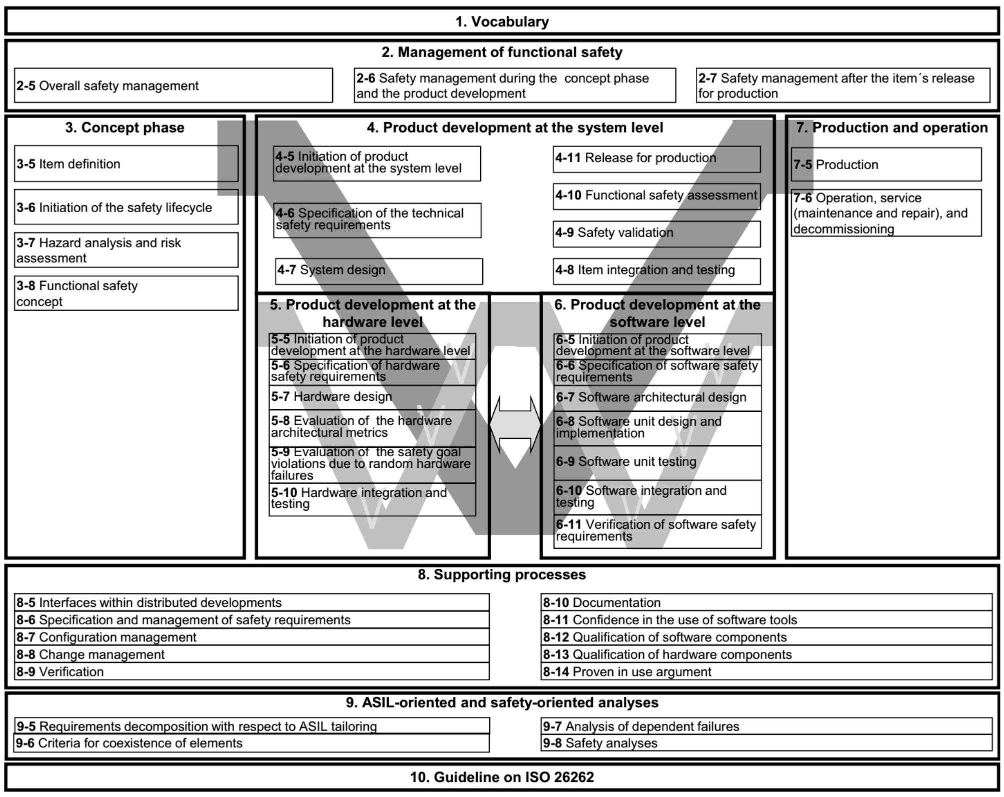

1. Vocabulary
2. Management of functional safety
2-5 Overall safety management
2-6 Safety management during the concept phase and the product development
2-7 Safety management after the item´s release for production
3. Concept phase
3-5 Item definition
3-6 Initiation of the safety lifecycle
3-7 Hazard analysis and risk assessment
3-8 Functional safety concept
4. Product development at the system level
4-5 Initiation of product development at the system level
4-6 Specification of the technical safety requirements
4-7 System design
4-11 Release for production
4-10 Functional safety assessment
4-9 Safety validation
4-8 Item integration and testing
7. Production and operation
7-5 Production
7-6 Operation, service (maintenance and repair), and decommissioning
5. Product development at the hardware level
5-5 Initiation of product development at the hardware level
5-6 Specification of hardware safety requirements
5-7 Hardware design
5-8 Evaluation of the hardware architectural metrics
5-9 Evaluation of the safety goal violations due to random hardware failures
5-10 Hardware integration and testing
6. Product development at the software level
6-5 Initiation of product development at the software level
6-6 Specification of software safety requirements
6-7 Software architectural design
6-8 Software unit design and implementation
6-9 Software unit testing
6-10 Software integration and testing
6-11 Verification of software safety requirements
8. Supporting processes
8-5 Interfaces within distributed developments
8-6 Specification and management of safety requirements
8-7 Configuration management
8-8 Change management
8-9 Verification
8-10 Documentation
8-11 Confidence in the use of software tools
8-12 Qualification of software components
8-13 Qualification of hardware components
8-14 Proven in use argument
9. ASIL-oriented and safety-oriented analyses
9-5 Requirements decomposition with respect to ASIL tailoring
9-6 Criteria for coexistence of elements
9-7 Analysis of dependent failures
9-8 Safety analyses
10. Guideline on ISO 26262

Figure 1.1: Structure of ISO 26262, adapted from Figure 1 found in all parts of ISO 26262.

A more straightforward overview of all activities performed in the project and their corresponding clause(s) in ISO 26262 can also be found in Table 1.1, showing all the activities performed in the project. The table also includes columns showing which phase each activity is performed in, and in which chapter and section in this report the activity is described and the relevant clauses in ISO 26262. The last column uses the notation [Part #]-[Clause #].

As seen in the table, this project follows a different grouping of activities than the parts of the standard. This is the case as the grouping of the standard aim to differentiate between different abstraction levels of a project,

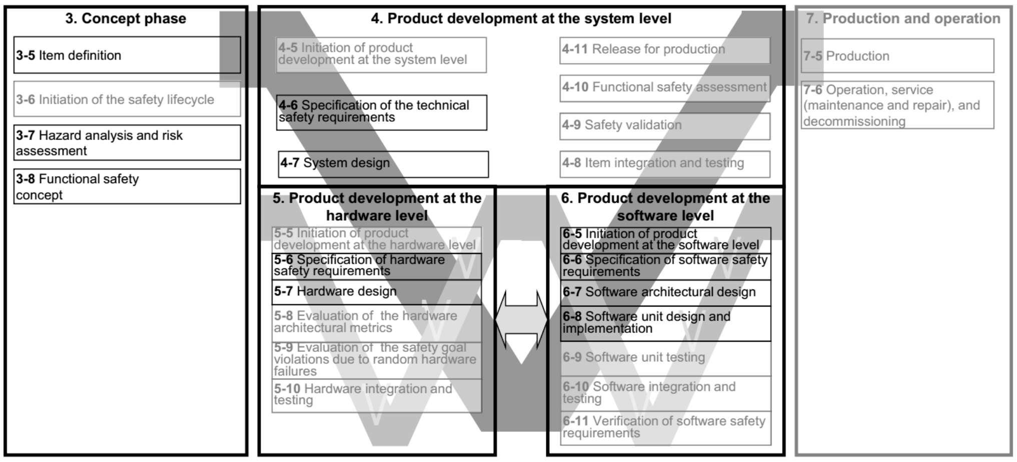

3. Concept phase
3-5 Item definition
3-6 Initiation of the safety lifecycle
3-7 Hazard analysis and risk assessment
3-8 Functional safety concept
4. Product development at the system level
4-5 Initiation of product development at the system level
4-6 Specification of the technical safety requirements
4-7 System design
4-11 Release for production
4-10 Functional safety assessment
4-9 Safety validation
4-8 Item integration and testing
5. Product development at the hardware level
5-5 Initiation of product development at the hardware level
5-6 Specification of hardware safety requirements
5-7 Hardware design
5-8 Evaluation of the hardware architectural metrics
5-9 Evaluation of the safety goal violations due to random hardware failures
5-10 Hardware integration and testing
6. Product development at the software level
6-5 Initiation of product development at the software level
6-6 Specification of software safety requirements
6-7 Software architectural design
6-8 Software unit design and implementation
6-9 Software unit testing
6-10 Software integration and testing
6-11 Verification of software safety requirements
7. Production and operation
7-5 Production
7-6 Operation, service (maintenance and repair), and decommissioning

Figure 1.2: ISO 26262 V-model, adapted from Figure 1 found in all parts of ISO 26262. Clauses not directly related to this report grayed out.

because different people often only work with their assigned level. Since all work in this project will be performed by the same person and as the aim of the project lies in the requirement elicitation activities and the concluding partial implementation, the grouping seen in Table 1.1 was therefore chosen instead.

As seen in Table 1.1, this project has three phases. The first phase, the preparation phase, includes activities being prerequisites for the requirement elicitation and system design activities in the next phase. The activities in the preparation activities therefore include:

- Creating an initial safety plan, including planning for the initial safety activities.
- Gathering the information about the so called "item", i.e. the verification system, to be included in the item definition, employing basic concepts of requirement engineering to gather and present the information.
- Performing an initial safety analysis to find out if the item contains safety critical elements or not. In this project this corresponded to performing a Hazard and Operability (HazOP) study on the basic functionality of a verification system. Specifying the initial Safety Goals (SGs), the initial safety requirements of the verification system, as well as unacceptable or unwanted behavior.

Table 1.1: List of project activities, with their section in the report and their corresponding clauses in ISO 26262 included.

Phase	**Section**	**Activity**	**Clause(s)**
Preparation	3.1	Safety plan	2-6
	3.2	Item definition	3-5
	3.3	Hazard analysis and risk assessment	3-7
Requirement	4.1	Elicitation of functional safety requirements	3-8
	4.2	Elicitation of technical safety requirements	4-6
	4.4	System design	4-7
	4.5	Elicitation of hardware safety requirements	5-6
	4.5	Elicitation of software safety requirements	6-6
Implementation	5.2 & 5.3	Communication monitor design	(5-7, 6-7 & 6-8)
	5.4	Communication monitor implementation	(6-8)
	5.5	Communication monitor integration and verification	-

These steps should, when finished, be able to tell what needs to be developed and if it should be developed as a safety critical item in accordance to ISO 26262.

The second phase of the project is called the requirement phase, as most of the activities performed during this phase are related to the elicitation and management of safety requirements. These activities are (as specified by the standard):

- Eliciting the implementation-independent Functional Safety Requirements (FSRs), requirements specifying the functionality needed to satisfy the SGs.

- Eliciting the implementation-specific Technical Safety Requirements (TSRs), requirements concerning the safety measures employed to satisfy the FSRs.
- Creating a system design for the verification system and assigning the TSRs to the hardware or software aspects of the design.
- Refining of one of the TSRs into Hardware Safety Requirements (HWSRs) and Software Safety Requirements (SWSRs).

These steps should together create the requirement specification for a basic verification system down to TSRs, including HWSRs and SWSRs based on one TSR, assignable to a single element.

The progression of functions, hazards and safety requirements in the two first phases in relation are included in Figure 1.3, with two "project definitions" included in the figure. The first project definition being the decision of what functions to analyze in the HazOP and the second project definition being the decision of which single TSR should be analyzed further in the Software and Hardware Safety Requirement Elicitation activities and later implementation.

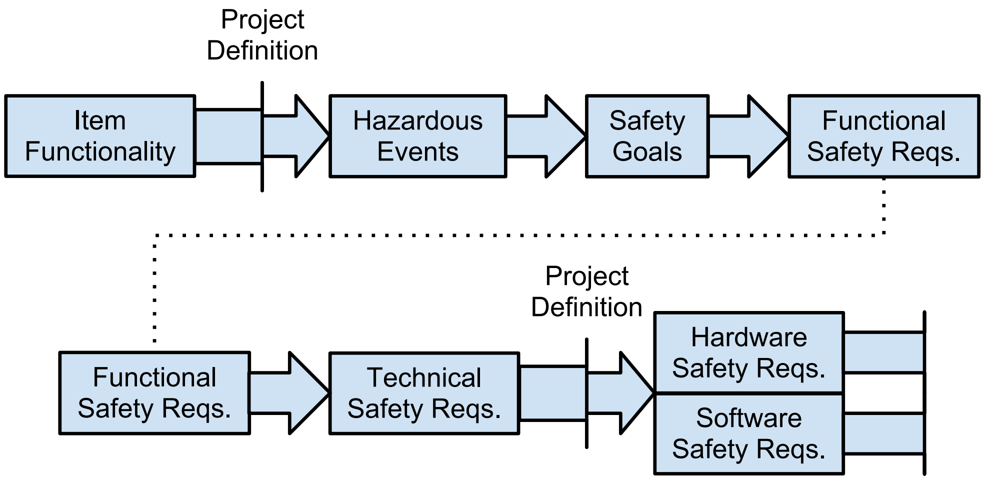


Figure 1.3: Progression of functions, hazards and safety requirements in the first phases of the project.

The third and final phase of the project, the implementation phase, is foremost based on the HWSRs and SWSRs elicited from the single TSR in the previous phase, some of them being implemented into an element of the item, in this case a communication monitor. A communication monitor being an element ensuring safe communication between the verification system and the system to be tested. The reason for a communication monitor being developed is that the chosen TSR and its derived HWSRs and SWSRs mainly concern the safety of communication to the System Under Test (SUT). The activities in this phase include:

- Choice/design of the hardware needed for the implementation, resulting in the use of an FPGA development kit from Avnet, to meet the applicable HSWRs. Creation of a software design implementing the required functionality, to meet the applicable SWSRs.
- Implementation of the design in synthesizable VHDL and deployment of the design on the FPGA development kit.
- Verification and demonstration of the communication monitor.

From the first phase of the project the conclusion is drawn that a verification system for safety critical automotive elements should be considered safety critical itself if developed according to ISO 26262. Additionally, it is found that the risk assessment method proposed in the standard is not directly applicable to development tools, requiring an alternative risk assessment method to be used.

The first and second phase of the work resulted in a safety requirement specification and a system design for a verification system for safety critical automotive elements. The two phases additionally show the difficulties in differentiating between the different types of safety requirements presented in the standard.

In the last phase, a communication monitor is implemented to provide confidence in the communication between the verification system and a vehicle ECU. In particular, it is determined which parts of the standard to follow when developing a safety critical automotive element on an FPGA platform.

1.4 Outline of the thesis

The rest of the thesis is organized as follows.

Chapter 2 gives a brief overview of the standard ISO 26262 and an introduction to the parts most relevant to the project. A short introduction to verification and system testing is also included, as well as a review of related work.

Chapter 3 describes the preparation and planning activities of the project, such as writing a safety plan, gathering information and requirements for an item definition and performing a hazard analysis and risk assessment.

Chapter 4 describes the requirement elicitation and system design activities of the project. In particular, safety requirements and safety-aware system design for the verification system are considered.

Chapter 5 describes the implementation of a communication monitor. The chapter also describes the activities performed to verify the functionality of the monitor.

Chapter 6 concludes the thesis with a summary of the results of the project, reflections on working with ISO 26262 and suggestions for future work.

Chapter 2

Theory

This project is centered around two main areas, the functional safety standard ISO 26262 and a type of testing that, depending on context, is called “verification” or “system testing”. Requirement engineering is also a field related to both these areas that is relevant to the project as several activities of the first half of the ISO 26262 V-model concerns eliciting the safety-related and non safety-related requirements of the element to be developed. This chapter therefore aims to give some background on these three areas and fields and present some related previous work.

2.1 ISO 26262

ISO 26262 [10] is a functional safety standard applicable to all road vehicles with a weight under 3500 kg, i.e. passenger vehicles. The standard was published in November 2011 and is based on the more general functional safety standard IEC 61508[1] [9]. The intention of ISO 26262 is to provide guidelines and best practices to increase the safety of E/E-systems in vehicles. This standard is often combined with laws and directives, like the EU Directive 2001/95/EC that require companies to follow state-of-the-art practices when developing any safety related product. The compliance to the standard both by vehicle manufacturers and automotive OEMs is assumed to increase in the coming years.

[1]Full name: Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems

2.1.1 Brief overview

ISO 26262 is a functional safety standard providing specifications on the safety lifecycle of a safety critical “item”, where an item is a concept in ISO 26262 corresponding to a single safety critical automotive E/E-system. This means the term item refers to everything from the main electric system of a vehicle down to single electrical control units (ECUs), as long as they are not decidedly or necessary a part of a larger system. As the standard covers the whole lifecycle of an item these specifications include specifications concerning the planning of the item to specifications concerning the production and decommission of the item, and everything in between. ISO 26262 is like its parent standard IEC 61508 [9] based on hazards and their risks. This means that the safety analyses of the standard are focused on the hazards of the E/E-systems and their associated risks, which according to the standard all should be found through structured methods. The associated risks are then assigned an Automotive Safety Integrity Level, ASIL, reminding of SIL as of IEC 61508[2]. This ASIL is then recurrent through most of the following steps, specifying to what degree the safety activities of the item or its elements need to be performed. The ASIL classification also ties to the specifications concerning the safety requirements, another important concept of the standard.

To best explain the structure of the standard it is a good idea to start with Figure 1.1. As seen in this figure the standard is composed of 10 parts, each composed of up to 14 clauses, structured around a V-model composed of Part 3 to Part 7 of the standard. As also seen from the figure Part 4 to Part 6 are the parts composing the main body of the large V while Part 3 and Part 7 are only slightly connected to the beginning and end of the “V-model”, mostly by association to clauses in the main parts. Part 5 and Part 6, the two parts corresponding to the of the lower half of the overall V-model, are additionally structured as smaller V-models themselves, a fact hinted at in Figure 1.1 by a lighter V in the background. The other parts, Part 1, Part 2 and Part 8 to Part 10 are not part of the V-model, as seen in the figure, and these parts contain clauses which support or explain the clauses in the central parts.

The clauses later referenced in this report can be found in Table 2.1 and Table 2.2, with the first column specifying clause in a notation sometimes used in the standard, [Part #]-[Clause #], and the second column including the name of the clause and a short description of its content.

[2]It is important to note that even though ASIL and SIL are similar concepts, a SIL cannot be translated into an ASIL, or vice versa.

Table 2.1: The clauses of ISO 26262 relevant to the project and a short description of each.

Clause	**Description**
2-5	"Overall safety management" Specifications concerning proof of the suitability of a organization involved in developing a safety critical item or element.
2-6	"Safety management during the concept phase and the product development" Specifications concerning the initial safety plan of a project, where a safety plan is a document describing the planning of the initial safety activities of the project.
3-5	"Item definition" Specifications concerning how to perform the item definition activity and what should be included in the item definition document.
3-7	"Hazard analysis and risk assessment" (HARA) Specifications on how the HARA activities should be performed and what results are expected from the activity, including SGs.
3-8	"Functional safety concept" Specifications on how to elicit and manage the FSRs.

Table 2.2: The clauses of ISO 26262 relevant to the project and a short description of each. (cont.)

Clause	Description
4-6	“Specification of the technical safety requirements” Specifications on how to elicit and manage the TSRs.
4-7	“System design” Specifications concerning the overall design of the system to be developed and assignment of TSRs to design elements.
5-6	“Specification of hardware safety requirements” Specifications on how to elicit and manage the HWSRs.
5-7	“Hardware design” Specifications concerning the hardware design.
6-5	“Initiation of product development at the hardware level” Second half includes specifications on choice of software tools and software language.
6-6	“Specification of software safety requirements” Specifications on how to elicit and manage the SWSRs.
6-7	“Software architectural design” Specifications concerning the software architectural design.
8-6	“Specification and management of safety requirements” Specifications concerning elicitation and management of safety requirements, i.e. SGs, FSRs, TSRs, HWSRs and SWSRs.
8-11	“Confidence in the use of software tools” Specifications concerning tool classification and tool qualification, as described later in Section 2.1.3.
9-5	“Requirement decomposition with respect to ASIL tailoring” Specifications describing the technique “ASIL decomposition”, as described later in Section 2.1.2.
9-8	“Safety analysis” Specifications concerning the aim and execution of hardware and software safety analyses.

2.1.2 Automotive Safety Integrity Level (ASIL) classification & requirement decomposition

As mentioned in the previous section, when following ISO 26262 [10] the hazardous events of the item to be developed and their associated risks must be identified in a systematic manner. This is for example done by using a (Design) Failure Mode and Effects Analysis ((D)FMEA) or a HazOP study. The risk associated with the identified hazards are then classified with three parameters, as follows:

- Severity, ranging from S0 to S3, based on the level of injuries the hazard may cause. S0 being no injuries and S3 being life-threatening or fatal injuries.

- Exposure, ranging from E0 to E4, based on the probability of the element being in a situation where the hazard could occur, with E0 being an unusual or incredible situation and E4 being a very normal situation, like driving on a dry highway.

- Controllability, ranging from C0 to C3, based on the hazardous situation being controllable or not by a normal driver, with C0 being the hazard only causing a minor distraction and C3 being situations in which the controllability of the vehicle is severely limited, for example brakes failing.

These three parameters are then combined to an Automotive Safety Integrity Level (ASIL), varying from QM, Quality Management, and ASIL A to ASIL D with rising severity, probability and/or controllability, following [10, Part 3, Table 4], included in this report as Table C.1 in Appendix C. The level QM is a special ASIL being only assigned to hazards with very low probability, hazards causing only slight injuries or hazards avoidable by a normal driver. Even though QM in practice means that the hazard does not need to be considered in the following ISO 26262 safety activities, it has some use when performing some requirement-based activities, such as ASIL decomposition, which will be presented shortly.

After this classification the ASIL is recurrent in the subsequent activities of the standard, and all safety requirements elicited to mitigate a risk of a hazard occurring also inherit the ASIL of the risk or its source requirement. This ASIL determines the development effort required by the standard, which means that effort needed to develop a safety critical E/E-system depends on the risk with the highest ASIL associated with the system. This effectively means that if an item or element, called “Product A”, has a single risk whose effects are hard to control and which is associated with a severe outcome in normal driving conditions, and several risks with less severe outcome only appearing in rare driving conditions, the development effort for each part included in Product A would be dictated by the single risk with the more probable and severe outcome which are harder to control.

To help to develop items or elements such as Product A, ISO 26262 provides a technique called “requirements decomposition with respect to ASIL tailoring” [10, Part 9, Clause 5], usually referred to as “ASIL decomposition”. This technique allows the developer of an item with risks of varying degrees, such as the item from the last paragraph, to decompose a high ASIL requirement into two new requirements where at least one of the two new requirements will have a lower ASIL than the source requirement. This decomposition can only be performed as long as the two new requirements are allocated to two “sufficiently independent” elements of the system. With the two elements being sufficiently independent only if no dependent failures violating the source safety requirements can be found, as described in [10, Part 9, Clause 7].

This technique, both applied in this project and explored or used in some earlier work, such as Åström et al. [22], has two main applications. The first application is lowering the development effort of a larger safety critical element by moving the most safety critical functionality out from an element into a new and smaller safety functionality element, and in this way lower the required development effort on the larger element. This element could for example be some kind of safety monitor. The second application is lowering the development effort of an element by splitting or sharing functionality between two redundant or complementary elements, lowering the total ASIL of the initial element by having two lower ASIL elements.

An example of these two decomposition methods can be seen in Figure 2.1, where a requirement R with ASIL D, is decomposed into the requirements R1 and R2 in two ways. In a) R1 retains the original ASIL of ASIL D(D) while R2 is assigned QM(D) and in b) R1 and R2 are both assigned the lower ASIL B(D). An item or element like Product A can with help of decomposition a) therefore move the high ASIL requirement to a new element, lowering the ASIL of the main element to the lower ASIL of the other requirements. It is important to note that the new ASILs are expressed with the ASIL of the initial requirement in parentheses after the actual ASIL, just as seen in the figure, to show that decomposition has been performed.

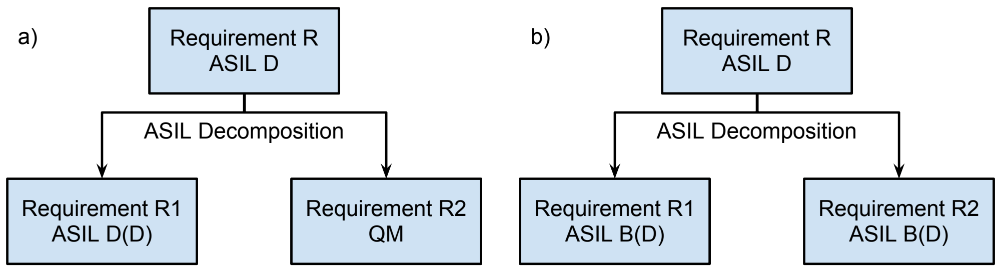


Figure 2.1: Examples of ASIL Decomposition.

2.1.3 Tool qualification

A concept in ISO 26262 that is not explicitly found in its parent standard, IEC 61508, is the qualification of software tools[3]. This concept concerns the reliability and trustworthiness of the software tools used when developing automotive E/E-systems in accordance to ISO 26262 and should be applied to both development tools and verification tools.

Because of this concept, as specified in [10, Part 8, Clause 11], before using a tool in the development of an ISO 26262 item including software elements, the tool has to be classified by two attributes. The first attribute is Tool Impact (TI), corresponding to the possibility that the tool malfunctions could impact an safety-related item, either by introducing errors or failing to detect errors in the item. The second attribute is Tool error Detection (TD), denoting the confidence in that a malfunction in the tool can either be prevented or detected. These two attributes are then combined into a Tool Confidence Level (TCL), which decides if the tool needs to be qualified before its use or not. If the tool needs to be qualified, the TCL combined with the ASIL level of the element to be developed decides the qualification measures needed to be permitted to use the tool in the development process.

For example, if the tool is assigned the lowest TCL, TCL1, it does not actually need any qualification at all. TCL1 is only assigned to tools with no chance of a malfunction impacting the safety-related item, or to tools in which there is a high degree of confidence that malfunctions will be prevented or detected. As the standard specifies that TI and TD should be determined conservatively, this means that TCL1 can be assumed to only be assignable to tools developed with great detail put in analyzing the inner working of the tool. The next and last two TCLs, TCL2 and TCL3, are very similar, with the difference being the TD level that defines if there is some level of

[3]A similar concept is found in the RTCA/DO-178B though, a standard applicable to avionics software.

confidence in the error detection in the tool or not. The results of both these TCLs are therefore very similar, the tool has to be qualified before use using a combination of the methods described in [10, Part 8, Table 4 & Table 5]. All this is included in Figure 2.2, which from left to right gives the classification flow used when assigning a software tool a TCL.

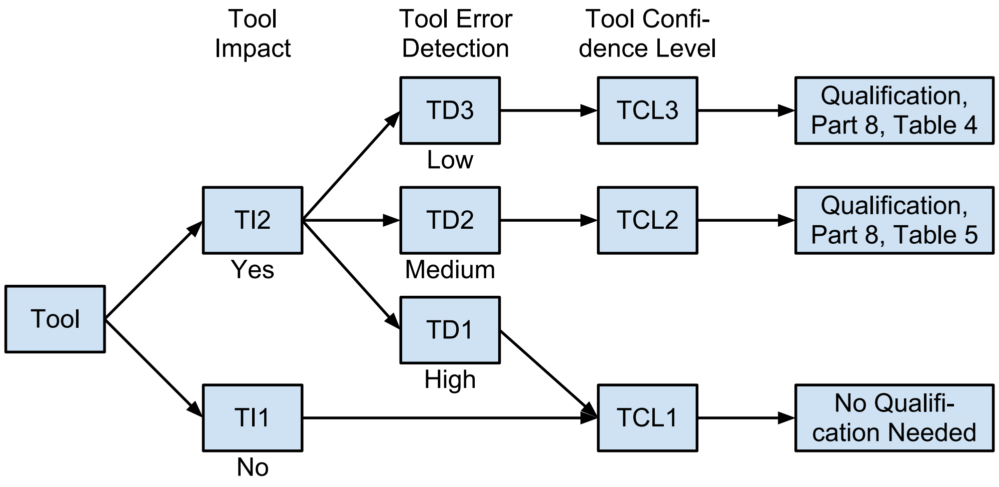


Figure 2.2: Diagram showing the Tool Confidence Level (TCL) assignment flow.

In this project, the goal is to explore the possibility of a verification tool that can be used on all safety critical items, including those with ASIL D requirements. This means that the most recommended methods are “Validation of the software tool in accordance with [subclause] 11.4.9” and “Development in accordance with a safety standard” [10, Part 8, Table 4 & Table 5], where subclause 11.4.9 concerns validation of the tool in relation to its requirements, potential malfunctions and reaction to anomalous operating conditions. As 1c) partially is covered by 1d) if the safety standard used is chosen with some care[4], the most interesting qualification method for this project should be 1d), to develop the tool in accordance to ISO 26262. This can only be performed to the best effort possible however, as the standard notes: “No safety standard is fully applicable to the development of software tools. Instead, a relevant subset of requirements of the safety standard can be selected.” [10, Part 8, Table 4 & Table 5].

[4]As an example, the process model of ISO 26262 implicitly covers the verification and hazard analysis specifications of [10, Part 8, 11.4.9], with clauses like [10, Part 3, Clause 7] and [10, Part 4, Clause 9]

2.2 Verification and system testing

While the tool to be developed foremost will be called a verification system, it could also be considered a system test tool, under the definition of system testing found in the wider concept of software testing [11, 12]. This section therefore aims to give an introduction to verification and system testing, and where in the product development process this types of testing comes in.

The definition of verification is, according to the vocabulary of ISO 26262, "determination of completeness and correct specification or implementation of requirements[...]" [10, Part 1, 1.137]. That means either showing the validity and completeness of the requirements or proving that the final system is following its initial requirements, with the verification system in this project aimed at the latter activity. System testing on the other hand is as noted a part of what is called "software testing", an approach often used when developing embedded systems to prove that the developed system follow its initial requirements. So while these two terms come from different backgrounds, they are very similar in their procedures and goals.

For example, both verification and system testing are different from the so called production testing, or manufacturing testing, which may be what some people generally associate with the word "testing". While production testing is used in production environments, to test each produced product is working as intended, verification and system testing does not really relate much to a finished product, but instead relate more to the development of the product. As a result, verification and system testing instead have the goal of providing proof that a system under development is developed according to the initial wishes of the stakeholders and fulfills the requirements set upon the system. Production testing can in relation therefore be considered as testing performed on copies of an already verified product.

To understand exactly where in the development phase that the verification comes in, it is good to return to the similarity to system testing, as part of the wider concept software testing. Software testing is in itself normally split into four types of tests, namely:

- Unit (or component) testing
- Integration testing
- System testing
- Acceptance testing

While unit testing and integration testing are usually only performed at the software level of an embedded system, system testing is done with more parts of the actual system, often by running the complete software on the actual hardware, or at least similar hardware. Acceptance testing is the final step of software testing, usually performed by letting the actual end user test the complete system [11].

Each of the levels in software testing also has a testing approach associated with it. The lower levels of testing, which are unit testing and integration testing, most often employ white-box testing. This means testing with knowledge of the complete system, making sure that all the internals are working correctly. The higher levels such as system testing and acceptance testing is usually performed with black-box testing, which means testing without complete knowledge of the inner workings of the system [11]. This effectively means that the only variables during system tests is the response of the SUT and the stimuli applied to its inputs to generate the response.

So, with this in mind, a verification system to be used like a system testing tool has two important features. The first important feature is that it needs to be "feature complete", which means to have functionality to test all the requirements set on the SUT. The second important feature follows from the nature of black box testing and is that the system needs to employ good communication features to communicate with the SUT, as the result of the testing will depend on how well the verification system handles the inputs and outputs of the SUT.

2.3 Related work

While ISO 26262 was released in late 2011, drafts for the standard have been available for a few years. Some research has therefore been performed in the last few years, both in the field of requirement elicitation in relation to the specifications in ISO 26262 [21, 4, 7, 19] and in the field of tool qualification [14, 3, 5, 20, 22]. While most of the research on tool qualification has been performed by tool manufacturers wanting to prove their tools qualifiable [14, 3, 5, 20], mostly by the qualification method of validation. Åström et al. [22] at Semcon has also authored a relevant article on the subject, proposing a qualification process involving both the qualification-by-validation method and the development according to a safety standard method.

The field of requirement engineering, both with relation to functional safety [1, 6] and without relation to functional safety [17], is also relevant to the project, as most activities in the development-related parts of ISO 26262 concern the safety-related requirements of the item in one way or another. The early item definition activity does for example concern non-safety-related requirement elicitation to some degree, as some information about the item is best found and expressed as requirements on the item and its environment.

2.3.1 Tool qualification

One of the organizations that have been looking on qualifying their tools for use in an ISO 26262 environment is MathWorks, with focus on their software tool Simulink [14]. MathWorks has together with TÜV SÜD developed a workflow to simplify qualification of MathWork's development and verification tools, as explained in Conrad et al. [3]. The approach to tool qualification presented in this white paper is not to provide a pre-qualified tool to the tool user, but instead provide a tool which has been proved as qualifiable and that has a supported qualification process and a supplied "qualification kit". The downside to this approach is that the supplied qualification kit can only be used if the reference workflow as specified by MathWorks is used.

This method, which only corresponds to the validation tool qualification method of ISO 26262, is also used by other companies to provide ISO 26262 compliant tools. For example, the code generation software TargetLink by dSpace [5] and the software testing tool family VectorCAST by Vector Software [20], are both supplied with workflows certified by TÜV SÜD and qualification kits. The only notable difference between the companies is that Vector Software explicitly states that they provide their so called Tool Qualification Data (TQD), which both includes tool qualification test data and results and test scripts for re-execution.

An alternative approach to qualify the tool, both by validation and development according to a safety standard, is presented in Åström et al. [22]. This approach moves part of the qualification effort to the tool manufacturer, instead of putting all the responsibility of tool qualification on the user, as in [3, 5, 20], while lowering the qualification effort by employing a semi-automatic validation. This semi-automatic validation is made possible through the inclusion of two additional modules outside the main test tool, a fault injector and a monitor, both independent of the test tool. This approach is assumed to reduce the development effort for the tool if developing according to ISO 26262, as the critical requirements can be moved to the more stable monitor, instead of remaining on the main verification engine, with the help of ASIL decomposition, as described in Section 2.1.2.

2.3.2 Requirement engineering

Regarding the field of general requirement engineering, Nuseibeh et al. [17] presents an introduction to the area of software system requirement engineering. The fields of software requirement engineering covered in the article are eliciting, modeling, analyzing, communicating, agreeing and evolving requirements. Although it does not address the special case of safety requirements, it gives a general view on how to work with requirements, which is especially helpful in the item definition activity and when working with satisfying the specifications found in [10, Part 8, Clause 6].

One field mentioned in the article that proved to be especially relevant to the project was the section “elicitation techniques”, which presents various techniques that can be used when eliciting requirements. This section was later considered when performing the item definition activity and trying to decide on a suitable elicitation technique for the project. A decision eventually being taken was to use the traditional technique of using a questionnaire, which is a written survey sent to the stakeholders of the project, containing questions concerning the product to be developed.

To understand the relevance of [1, 6] it is important to know that the specifications on safety requirements in ISO 26262, as found in [10, Part 8, Clause 6], specify that all safety requirements have to be traceable to their source, i.e. previous safety requirements, and to their possibly derived safety requirements. While the necessity of this level of traceability between safety requirements has been known for a while, different solutions have been put forth, with Adedjouma et al. [1] proposing the ReMIAS metamodel, based on the Unified Modeling Language (UML) profiles SysML, MARTE, EAST-ADL2 and DARWIN4Req, the last as presented in [6]. The traceability is this way ensured through the UML-relationships, both relationships inherent in the UML specification and relationships introduced in DARWIN4Req and the ReMIAS metamodel.

The use of UML to ensure requirement traceability in Adedjouma et al. [1] and Dubois et al. [6] led to the decision to use UML-modeling to manage the safety requirements in this project, which according to the articles should keep traceability to a satisfactory level, as to satisfy [10, Part 8, Clause 6]. But as the metamodels proposed in both articles also include UML-types for system modeling, system verification and test cases, the metamodels were considered outside the scope of this project. So a simple requirement class and requirement diagram, as the ones found in Enterprise Architect by default, and also used by the requirement management tool RaQuest, were used instead. This way the traceability between requirements was considered to be satisfactory in accordance to the ISO 26262 specifications.

The work of Sternudd [21] is relevant when writing the safety requirements, as it concerns how to fulfill the possibly contradictory requirements from ISO 26262 that all safety requirements should be unambiguous and at the same time fully readable by those working at the adjacent abstraction level. Sternudd proposes either a double set of requirements, one written in natural language and one in formal notation, or a requirement specification written exclusively in a Constrained Natural Language (CNL). Based on [21] a choice was taken in this project to use an internally constructed CNL when writing safety requirements for this project. A brief description of this internally constructed CNL can be found in Appendix B.

Three more works that discuss the specific safety requirement engineering found in ISO 26262 are Dittel et al. [4], Sinha [19] and Gradin et al. [7]. Dittel et al. [4] provides some examples on safety activities like HARA, safety requirement elicitation, safety analyses and verification being performed on an existing active safety system. But as this project foremost concerns the first two subjects and they additionally are performed on a new system, in contrast to the already designed system in the article, its recommendations are not fully applicable on this project. This results in that only some recommendations, foremost on how to formulate and differentiate the different levels of safety requirements and in a slight degree also how to perform the HARA, has been considered in this project. Sinha [19] meanwhile only provides some examples on how to perform the safety analyses of an ISO 26262, including HARA and later safety analyses. However, with most of these analyses being outside the scope of this project, only some of the recommendations on how to perform a HARA have been considered in the project.

The thesis of Gradin et al. [7] describes a project of developing a collision avoidance system in accordance to ISO 26262, and does as this project focus more on the requirement specification and design related activities. But as with Dittel et al. [4] it does so with an existing product in mind, thus making the initial knowledge of the system to be developed much more extensive than in this project. The way of formulating requirements, and the conclusions on the level of detail on each level of the requirement elicitation activities have been considered though. Gradin et al. [7] concludes that a high level of detail on the higher safety requirement levels, such as FSRs, does not only increase the work needed for elicitation, but also could hinder the traceability to the SGs. Another important conclusion of the thesis that has been considered in this project is the recommendation of using a requirement management tool to manage the safety requirements, and to make the required traceability of the requirements simpler to prove.

2.3.3 Summary of related work

There are two areas of related work to which this project needed to position itself. The first is the field of tool qualification, a field arising from the concept of the same name in ISO 26262. While some companies such as

MathWorks [14], dSpace [5] and VectorCast [20] have studied the field to help sell their tools, they have mostly looked at the qualification method of validation, and not the method of developing the tool according to a safety standard, which this project is more based upon. Åström et al. [22] is therefore more interesting, but at the same time it only discusses the concept of a compliant verification tool, employing the proposed qualification scheme. This means that the article does not present any actual results, which subsequently results in the article not affecting the work of this project directly.

The second area is requirement engineering, mostly in relation to ISO 26262. Here one article by Nuseibeh et al. [17] concerning general requirement engineering, including safety and non-safety related requirements, was found helpful for understanding the basics of requirement engineering. This article [17] was additionally later found helpful in the item definition. Two articles by Adedjouma et al. [1] and Dubois et al. [6] concerning the importance of traceability in safety requirement and how to achieve traceability were also studied, resulting in the choice of using UML for the requirement specification.

Concerning requirement engineering in an ISO 26262 environment one article by Dittel et al. [4] and two masters theses, one by Gradin et al. [7] and one by Sternudd [21], were found relevant to the subject. The article [4] and one of the theses [7] provided examples on how to perform the safety analyses of an ISO 26262 project and how to formulate safety requirements and how to differentiate between the different levels of safety requirements. The second thesis [21] mostly discusses the problem with writing unambiguous and understandable requirements. The first two works [4, 7] therefore resulted in choices on how to handle the different levels of requirements in this project, and the last work [21] resulted in the use of CNL to express the requirements, as to avoid ambiguousness. The article by Dittel et al. [4] and an article by Sinha [19] also provided some recommendations for the HARA activity in the project.

Chapter 3

Planning and Preparation

Before starting the requirement elicitation and design process of the verification system, as found in the first half of the V-model of ISO 26262 and seen in Figure 1.2, some planning and preparation are due, as activities performed in what ISO 26262 calls the “concept phase”. The planning, which is started at the beginning of the project and extended after learning more about the system through the initial safety activities, and the preparation activities are performed in accordance to the first clauses of [10, Part 3]. The preparation activities include collecting information about the item to be developed, in our case a verification system, into an item definition and consequently performing a Hazard Analysis and Risk Assessment (HARA) of the item, based on the information in the item definition. The HARA consequently provides two of the initial results to the project. The first result is a conclusion on the matter if the item is safety critical or not and consequently if the item need to be developed in accordance to ISO 26262 or not. The second result is the initial set of safety requirements, the SGs.

3.1 Safety plan

The first normative part of ISO 26262, [10, Part 2], includes the first activity required by ISO 26262 for a project to develop a safety critical automotive product, which is creating a safety plan, a complement to, or part of, the project plan found in most development projects[1]. While some of the specifications in Part 2 have to be fulfilled before continuing to the development parts of the standard, this safety plan is a document which follows the com-

[1]The project plan of this project consisted of a planning report written in the first month of the project, from which some information included in Chapter 1 is taken, and the safety plan itself

plete project from here on. This results in the safety plan being revised and supplemented with more information during all of the subsequent development parts of the standard. The safety plan therefore cannot be fully completed during this initial activity.

The initial safety plan written in this project before the start of the item definition and the HARA activity therefore only included the following parts:

- Evidence of good safety culture, as of [10, Part 2, 5.4.2].
- Evidence of competence, as of [10, Part 2, 5.4.3].
- Evidence of quality management, as of [10, Part 2, 5.4.4].
- Assignment of roles in the project, such as project manager and safety manager, as of [10, Part 2, 6.4.2].
- Planning of item definition and HARA activities, as of [10, Part 2, 6.4.3 & 6.4.4].

The reason for not planning all safety activities from the start, was the fact that the inclusion of safety critical elements in the item to be developed could not be confirmed until after the activities performed in the HARA part of the project. After the conclusion of Section 3.3 that the system did include safety critical elements, which in this project coincided with the start of the second phase of the project, the requirement phase, the safety plan was extended with a "requirement management plan". This requirement management plan mainly concerns the planning of the rest of the activities performed in the project and supporting clauses such as [10, Part 8, Clause 6], which concerns the specification and management of safety requirements.

The safety plan created in this activity is additionally the first part of what the standard calls the "safety case" of the project, which simply is defined as "[the compilation of] work products that are generated during the safety lifecycle" [10, Part 4, 6.4.6.2].

3.2 Item definition

The first activity performed in the ISO 26262 V-model is creating an item definition, as specified in [10, Part 3, Clause 5]. During this activity all information already known about the item to be developed and/or previous or similar items, and the possible expectations of the item to be developed, is collected and compiled into an item definition. The aim of the activity is to create an item definition containing information of the system. This information is subsequently used in the later activities, foremost the HARA. Most of this information could also be found or presented as the requirements on the system to be developed, but while performing this activity it is important to note that nothing is known or assumed about if the item is safety critical or not. So if the item already has any known requirements, these cannot yet be classified as safety-related requirements or non-safety-related requirements during this activity, regardless of their status in earlier projects or items.

Before this project began, Semcon had developed a verification system similar to the verification system to be developed in this project, but this verification system was not aimed at automotive environments and was not developed in accordance to a safety standard, as recommended by ISO 26262. Despite this, the knowledge and expectations from the Semcon engineers having worked with this previous verification system seemed like a good source of information for the item definition. But as some of these engineers were working in other premises than the one this project was performed at, requirement elicitation methods like interviews or group discussion were not preferable to gain the required information for the item definition. The method of using a questionnaire, as mentioned in [17], was therefore seen as more suitable. A questionnaire was therefore constructed and sent to the engineers having working with the earlier verification system, or similar products, to gather their knowledge about the previous and other verification system and their ideas, wishes and requirements on the new verification system.

This questionnaire was constructed by questions either collected from different sources, including a questionnaire from another project at Semcon, or questions specifically written to give the information as required by [10, Part 3, Clause 5]. A more in-depth description of this work can be found in Appendix A. As an example, [10, Part 3, 5.4.1. b)] states that "The functional and non-functional requirements of the item as well as the dependencies between the item and its environment shall be made available [including the] environmental constraints". This specification was then covered in the questionnaire by the question "In what physical environments would the product be used?", which received answers like "The product will be used in the laboratory [...] or inside the vehicle".

In general, the questionnaire gave a good foundation for writing an item definition. There was some critique of the length of the questionnaire, which possibly could have caused the quality of the answers to the last questions to suffer[2], and some of the questions did not get the intended type of answer, thus not providing the information needed for the item definition. But all in all most of the information that was to be included in the item definition, as of [10, Part 3, Clause 5], was covered, giving the project a good foundation for the next activity, the HARA.

3.3 Hazard analysis and risk assessment

With an item definition completed, the next activity in preparation for the ISO 26262 V-model, found in Figure 1.2, is the HARA. This activity could in itself be divided into two subactivities, namely hazard analysis and risk assessment, even though they are closely related to each other. In the hazard analysis subactivity the information collected from the previous activity, the item definition, is used as a foundation for a structured analysis of the hazards of the item. In the risk assessment subactivity the risks associated with the identified hazards are then classified by three variables: severity, exposure and controllability, as described in [10, Part 3, Clause 7], which are used to assign each hazard an Automotive Safety Integrity Level (ASIL). The risk assessment subactivity additionally involves assigning each risk at least one SG. These SGs are the initial safety requirements of the item and are specified with the functionality needed to mitigate each identified hazard in mind.

Concerning the first subactivity, the hazard analysis, ISO 26262 specifies that “The hazards shall be determined systematically by using adequate techniques” [10, Part 3, 7.4.2.2.1], and gives some suggestions of suitable techniques, for example FMEA, field studies and quality history. The examples given are mostly techniques working best in projects where there has been a previous version of the same product or the product already is more defined and as this project did not have either, these techniques could not be applied to this project. A choice of using the well tried systematic hazard analysis technique HazOP was therefore chosen instead. A technique well suitable as it also often is used in the automotive industry for early safety analyses.

However, some deviations to the standard procedure HazOP were made, as the original HazOP is aimed at finding hazards in the chemical industry and therefore is designed for analyzing deviations in parameters such as flow, pressure and temperature in mind. In this project the HazOP instead focuses

[2]A shorter version of the questionnaire with more precise wording of some questions was subsequently prepared, but it was never used in any larger extent as most people who had the time to answer did so rather quickly, and it was based on these persons answers this second and more concise questionnaire was based.

on the deviations in the functionality of the item, something that probably could be called "functional HazOP", a technique similar to the techniques used in Sinha [19] and Dittel et al. [4]. When doing a HazOP study, whether it is a functional HazOP or not, something called guide words are used to help with the analysis. In this project the most used guide words, as found in example DEF STAN 00-58 [16], was used, with the addition of the two extra guide words "Faster" and "Slower" to help find hazards arising from clocking or timing errors. A full list of the guide words and their meanings, as interpreted in this project, can be found in Table 3.1.

Table 3.1: List of HazOP guide words used in the HazOP-session, with description of each guide word.

Guide word	**Description**
No	None of the intended functionality is performed.
As well as	Something additional to the intended functionality is performed.
Part of	Only some of the intended functionality is performed.
Reverse	The intended functionality is performed in the wrong order.
Other	Something different to the intended functionality is performed instead of the intended functionality.
Early	The intended functionality is performed too early in time.
Late	The intended functionality is performed too late in time.
Before	The intended functionality is performed too early in relation to another function.
After	The intended functionality is performed too late in relation to another function.
Faster	The intended functionality is performed too fast.
Slower	The intended functionality is performed too slow.
More	The intended functionality is performed with some related parameter being too large.
Less	The intended functionality is performed with some related parameter being too small.

With the guide words decided upon, the expected and desired functionality as described in the item definition was adapted into a table of potential functions of the verification system, resulting in 27 functions.

Note: In a project aiming for production of the item the HazOP would then be performed on all these 27 functions. But because of the project objective to follow the standard in depth more than width, this table was shortened to a table with the functions considered important for a basic working verification system, as seen in Table 3.2. Out of these eight functions of the verification system, only four were considered to be decisively safety critical and most interesting to analyze, as they all were associated with some complexity, such as supplying stimuli to the SUT. These more complex functions could be assumed to lead to more significant hazards, opposed to simpler functions like turning the verification system off. These four main functions are marked with gray background in Table 3.2. Most of these functions are as seen from the table related to the input/outputs of the system, which is not very surprising as good communication is very important aspect of a verification system, as seen in the paragraph in Section 2.2 concerning black box testing.

Table 3.2: List of main functions of an verification tool, further analyzed functions marked with gray background.

Mode	**State**	**Elem.**	**Process**	**Function**
ON	General	Sys	Always	Initialize
ON	General	Sys	Always	Shut down
ON	Verification	Main	Test case handling	Read test case description
ON	Verification	Main	Verification	Put SUT in mode to be tested
ON	Verification	Main	Verification	Apply stimuli to SUT
ON	Verification	Main	Verification	Read output from SUT
ON	Verification	Main	Verification	Power off SUT
ON	Verification	Main	Reporting	Evaluate output from SUT

A HazOP analysis was then performed on each of these four functions, in a group session with two more experienced safety engineers. The HazOP session process went as follows:

For each function found in Table 3.2:

- Find deviations of intended functionality based on the guide words applied in relation to the function.
- For each deviation of intended functionality found:
  - List the possible consequences of the deviation.
  - If the consequences of the deviation could risk the safety of the item:
    - Find as many possible causes of the hazard as possible
    - List the possible safeguards that could be used to protect the item from reaching the hazardous situation.
    - List the safe states that could prevent the hazard to occur or cause any harm, if any.
    - Formulate a SG, describing the functionality needed or the functionality not accepted in order to avoid the hazard.

The result of this process was that roughly 60 deviations were identified, each complete with possible causes and safeguards, and corresponding safe states when applicable.

As an example of the result of the HazOP analysis the HazOP sheet for the function “Apply stimuli to SUT” can be found in Table 3.3. In this table the deviations found by considering the guide words applied to the functionality is seen in the first column, with the results in form of possible consequences, causes, safeguards and safe states found in their respective columns. The SGs elicited from the hazards are also found in a column of its own. This table shows that a thorough HazOP not only takes time but also results in a great amount of data, and while some of this data can be re-used in later activities, performing a HazOP on all functions of a system will take time.

With a completed hazard analysis in the form of a HazOP the next sub-activity to perform was the risk assessment. As explained in Section 2.1.2, in ISO 26262 the risk assessment is performed by first classifying the risk according to a rather well defined classification scheme with three variables: severity, exposure and controllability. The problem that appeared in this project was that it was found to be hard to classify the hazards of the verification system in this way, without having knowledge about the system on which the verification system will be used, the SUT. Therefore, after careful consideration and searching for possible alternative classification schemes

Table 3.3: Function HazOP of the function “Apply Stimuli to SUT”.

Function:		Apply Stimuli to SUT				
Guide Word	**Deviation**	**Conse-quences**	**Causes**	**Safeguards**	**Safety Goals**	**Safe State**
No	No stimuli applied to SUT.	Potential false negative.	Communication error. Memory error.	Message/signal counter. Log output to SUT (Log checker). Memory protection.	Failure to apply stimuli shall not lead to silent false negatives.	No test result.
As well as (more than)	Apply stimuli as well as put SUT in mode to be tested	1. Potential false negative. 2. Item damage.	Control flow problem (Scheduling). Interruption from operating system.	No undefined states. Control flow monitoring. Adequate scheduling algorithm.	System shall not put SUT in mode to be tested while applying stimuli.	-
Part of	Only part of stimuli applied.	Not fully executed test case. Potential false negative.	Communication error. Memory error.	Message/signal counter. Log output to SUT (Log checker). Memory protection.	Failure to only apply part of stimuli shall mot lead to silent false negatives.	No test result.
Reverse	Reverse order of stimuli data.	Potential false negative.	Communication error. Memory error. Logical problem.	Message/signal counter. Log output to SUT (Log checker). Memory protection. Logical check/diversified implementation.	Reverse order of stimuli data shall not lead to silent false negatives.	No test result.
Other	Wrong stimuli applied to SUT.	1. Potential false negative. 2. Item damage.	Communication error. Memory error. Logical problem.	Message/signal counter. Log output to SUT (Log checker). Memory protection. Logical check/diversified implementation. Calibration mechanism for stimuli.	Wrong stimuli shall not be applied to SUT.	-
Early	Apply stimuli too early.	Potential false negative.	Communication error. Logical problem.	Message/signal counter. Log output to SUT (Log checker). Logical check/diversified implementation.	Stimuli shall not be applied to SUT too early.	No test result.
Late	Apply stimuli too late.	Potential false negative.	Communication error. Logical problem.	Message/signal counter. Log output to SUT (Log checker). Logical check/diversified implementation.	Stimuli shall not be applied to SUT too late.	No test result.
Before	Stimuli applied before put SUT in mode to be tested.	1. Potential false negative. 2. Item damage.	Logical problem. Control flow problem (Scheduling).	Logical check/diversified implementation. No undefined states. Control flow monitoring. Adequate scheduling algorithm.	Stimuli shall not be applied before SUT is put in mode to be tested.	No test result. -

Table 3.4: Function HazOP of the function “Apply Stimuli to SUT”. (cont.)

Function:		Apply Stimuli to SUT				
Guide Word	**Deviation**	**Conse-quences**	**Causes**	**Safeguards**	**Safety Goals**	**Safe State**
After	Stimuli applied after read response.	Potential false negative.	Communication error. Logical problem. Control flow problem (Scheduling).	Message/signal counter. Log output to SUT (Log checker). Logical check/diversified implementation. No undefined states. Control flow monitoring. Adequate scheduling algorithm.	Stimuli shall not be applied after read response.	No test result. Unvalidated test results.
Faster	Stimuli supplied to SUT applied too quickly.	1. Potential false negative. 2. Item damage.	Communication error.	Message/signal counter. Log output to SUT (Log checker).	Stimuli shall not be applied with the wrong timings.	No test result.
Slower	Stimuli supplied to SUT applied too slow.	Potential false negative.	Communication error.	Message/signal counter. Log output to SUT (Log checker).	Stimuli shall not be applied with the wrong timings.	No test result.
More	Stimuli is longer than specified.	1. Potential false negative. 2. Item damage.	Communication error. Logical problem.	Message/signal counter. Log output to SUT (Log checker). Logical check/diversified implementation.	Stimuli shall not be applied with the wrong timings.	No test result.
More	Stimuli supplied to more interfaces than specified.	Item damage.	Communication error. Logical problem.	Message/signal counter. Log output to SUT (Log checker). Logical check/diversified implementation.	Stimuli shall only be supplied to the number of interfaces specified in the test case.	No test result.
More	Data in stimuli beyond threshold.	Item damage.	Communication error. Logical problem.	Message/signal counter. Log output to SUT (Log checker). Logical check/diversified implementation.	Stimuli levels shall not exceed stimuli threshold.	No test result.
Less	Stimuli is shorter than specified.	Potential false negative.	Communication error. Logical problem.	Message/signal counter. Log output to SUT (Log checker). Logical check/diversified implementation.	Stimuli shall not be applied with the wrong timings.	No test result.
Less	Stimuli supplied to less interfaces than specified.	Potential false negative.	Communication error. Logical problem.	Message/signal counter. Log output to SUT (Log checker). Logical check/diversified implementation.	Stimuli shall only be supplied to the number of interfaces specified in the test case.	No test result.

without finding any good result, a decision was made to not perform any parameter classification. Instead more focus was put on the subsequent risk assessment, as the parameter classification foremost is an activity to simplify the ASIL assignments to the risks. This effectively removed the severity, exposure and controllability parameters from the project altogether.

With no parameters to base the ASIL assignment to the risks on, an alternative approach to the ASIL assignment had to be found. Some further discussion around the subject of the required safety of the verification system, led to the conclusion that the required level of safety on the verification tool would be dependent on the required level of safety on the SUT. Therefore it would make sense to classify the ASIL of the hazards of the tool to the same ASIL as the hazards in the SUT, which for a general SUT would be the highest ASIL on the SUT, as per the note to [10, Part 3, 7.4.4.2], "If classification of a given hazard [...] is difficult to make, it is classified conservatively, i.e. whenever there is any doubt, a higher ASIL classification is given rather than a lower". As the goal of the project was to research a general verification system for all kinds of automotive E/E-systems, this means that the tool should be able to verify items with requirements assigned ASIL D. This in turn resulted in all the risks of the verification system having to be assigned ASIL D, to be able to verify these ASIL D requirements in a safe way.

Concerning the elicitation of the SGs, an initial set of SGs was defined already at the HazOP-stage. As the risk assessment did not really affect this set of SGs in any way, they were kept as they were and just added to the complete safety requirement specification. An example of SGs, more specifically those which later proved to be the most relevant to the implementation phase, can be found in Table 3.5. In this table only one of the three safe states identified in the HazOP is found, namely "No test results given". The other two safe states, "Invalidate test results" and "No test performed", were assigned to SGs more related to scheduling errors or errors arising in the reading and interpretation of the test case. In total 28 SGs were identified from the 60 hazardous events found in the HazOP study.

Table 3.5: An example of SGs, derived during the HazOP study.

ID	Summary	Assigned to	Safe State	Destination Req. ID	ASIL
SG004	Failure to apply stimuli shall not lead to silent false negatives.	Verif. eng.	No test result given	FSR033 FSR039	D
SG006	Only applying part of stimuli shall not lead to silent false negatives.	Verif. eng.	No test result given	FSR036 FSR039	D
SG007	Reverse order of stimuli data shall not lead to silent false negatives.	Verif. eng.	No test result given	FSR037 FSR039	D
SG008	Wrong stimuli shall not be applied to SUT.	Verif. eng.	-	FSR038 FSR039	D
SG013	Stimuli shall not be applied to the SUT at the wrong speed.	Verif. eng.	No test result given	FSR042 FSR062 FSR039	D

3.4 Result and discussion

With all the preparatory safety activities finished, the preparation phase of the project was finished with the following results;

- A safety plan, describing how safety should be handled in the following activities of the project.
- An item definition, describing the item to be developed, i.e. the verification system.
- A HARA complete with the project SGs, describing the possible risks of the verification system and an initial set of requirements concerning how to mitigate the risks.

Based on these activities and their results it could now definitely be decided that the system should be considered a safety critical item according to ISO 26262, answering the first question in Section 1.2.

With the exception of the difficulties when trying to classify the hazardous events with the three parameters in the second part of the HARA, no major problems arose while following the standard during this phase. No major deviations were therefore made from the standard, with the exception of the omission of the parameter classification subactivity in the HARA and the adjustments in the subsequent ASIL determination. As the early parts of the standard present a rather well defined development process, followed during this phase of the project, this was also as expected. This also partially gave answers to the second question in Section 1.2, with the rest of the answers having to wait until the subsequent two phases had been completed.

However, the problems with classifying the hazardous events to the ISO 26262 parameters is an important result for the project. Looking back at the software tool qualification methods, as found in [10, Part 8, Table 4 & Table 5] of the standard, one of the two highly recommended tool qualification methods for tools to be used in development of ASIL D elements are “Development in accordance with a safety standard”. The standard even proposes itself as an example of a good standard to follow if choosing this approach to tool qualification. But as this project shows, some clauses quite central to ISO 26262 are hard to apply to software tools, for example the parameter classification of the risks. However, the standard does note that a “relevant subset of requirements of the safety standard can be selected” [10, Part 8, Table 4 & Table 5], to which the conclusion in our case must be drawn that a good compromise is to omit the parameter classification of the risks of the verification system. Instead a conservative classification of the risks based on the ASIL of the SUT was proposed. With this project aiming to find the requirements for a verification system being able to verify systems with ASIL D requirements, this brings us to the conclusion that all requirements on the verification system should be classified as ASIL D, if nothing else can be proved.

The second issue during this phase came up during the HazOP-session and was the question of the “safety goal” (SG) definition. As the vocabulary part of ISO 26262 gives the very basic explanation that a SG is a “top-level safety requirement as a result of the hazard analysis and risk assessment” [10, Part 1, 1.108] it was not fully clear on how to formulate the SGs. After some discussions and studying of other works, such as Dittel et al. [4] and Gradin et al. [7], a decision was made to treat SG as functional requirements specifying either what functionality is required by the system to avoid a hazard, or what functionality the system should not be permitted to perform, in order to prevent a hazard. Examples of these SGs are given in Table 3.5.

With the preparatory activities finished and the results complete and compiled, the project could move onto eliciting the safety requirements for the safety requirement specification for the verification system.

Chapter 4

Requirement elicitation and system design

The requirement phase mainly consists of four safety requirement specification activities and one design activity, each corresponding to a clause of ISO 26262, as seen in Table 1.1. These activities are all based on the SGs from the preparation phase of the project, and are performed in sequential order, with the exception of the last two activities which are performed in parallel. An additional activity was performed before the first requirement specification activity, that is the extension of the safety plan to include a requirement management plan, as mentioned in Section 3.1. Since the content of this extension to the safety plan is very closely related to the execution of each safety requirement related activity, the most important information included in this plan is included in the description of each activity, instead of being collected in a single section of this chapter.

4.1 Elicitation of functional safety requirements

The first activity of the requirement phase was eliciting the FSRs. This activity stems from [10, Part 3, Clause 8] and results in a collection of FSRs, a collection called the Functional Safety Concept (FSC). The FSRs are defined as "specification[s] of *implementation-independent* safety behavior, or *implementation-independent* safety measure[s], including [their] safety-related attributes" [10, Part 1, 1.53] (emphasis added). FSRs are therefore in this project considered to be the basic safety requirements specifying the safety functionalities of the item, independent from any assumptions about the implementation of the item.

Concerning the process of finding and identifying these FSRs in this project one of the most important specifications considered was [10, Part 3, 8.4.2.2]: “At least one functional safety requirement shall be specified for each safety goal”. With this clause in mind, the FSRs were identified from the SGs, with each FSR being created from at least one SG. Similar FSRs were also combined in such a way that the ISO 26262 specifications specifying each FSR to be externally consistent[1] with the other FSRs were satisfied, while keeping the FSRs atomic[2].

ISO 26262 additionally specifies in [10, Part 3, 8.4.2.3] five attributes that should be considered when specifying each FSR:

1. Operating modes
2. Safe states
3. Fault tolerant time interval
4. Emergency operation interval[3]
5. Functional redundancies.

Considering the verification system, only the first two attributes are applicable, as the faults covered in the HazOP only need to be detected at evaluation of the test result, at which all functionality should be finished, and therefore do not have any safety-related time constraints. Further, as the project only focused on functionality of the system when running a test case, and not functionality like calibration or test result evaluation, the operating mode attribute could also be omitted in the requirement specification. This left safe states as the only important attribute for consideration. But as not all FSRs had an associated safe state, and the relationship between the requirement description and the safe state could be unclear if the safe states were added as properties to the requirements, a choice was then taken to make the safe states into their own FSRs. This decision was also supported by the work of Gradin et al. [7], having incorporated the safe states of their SGs into the description of their FSRs.

[1]“[E]xternal consistency means that multiple safety requirements do not contradict each other.” [10, Part 8, 6.4.3.1 d)].

[2]“Safety requirements at one hierarchical level are atomic when they are formulated in such a way that they cannot be divided into more than one safety requirement at the considered level.” [10, Part 8, 6.4.2.4 b)].

[3]Emergency operation is the state which would occur between the system detects a fault to when it reaches a safe state, if the transition cannot be done immediately. Emergency operation interval is therefore the time in this state as needed by the system such as the safety of the system is not violated. [10, Part 1, 1.34 & 1.35]

ISO 26262 additionally requires all safety requirements to be assigned to a part of the preliminary architecture of the system [10, Part 8, Clause 6]. In the case of this project this was trivial since looking at the FSRs all of them were so far related to the main functional element of the verification system, as no design activities yet had been performed. This main functional element was therefore called verification engine, to differentiate it from future elements of the verification system.

The elicitation activity eventually resulted in 32 identified FSRs, a majority of which were closely related to the detection of various types of erroneous signals, to or from the verification system. However, this was expected, as most of the important functions as decided in Section 3.3, were related to the communications of the verification system. An example of FSRs, namely the FSRs most related to the later communication monitor, can be seen in Table 4.1.

Table 4.1: An example of the elicited FSRs.

ID	Summary	Assigned to	Destination Req. ID	Source Req. ID	ASIL
FSR033	The verification system shall detect failure to apply any stimuli to SUT.	Verif. eng.	TSR078 TSR074 TSR065 TSR064	SG004	D
FSR036	The verification system shall detect failure to apply complete stimuli to SUT.	Verif. eng.	TSR078 TSR074 TSR065 TSR064	SG006	D
FSR037	The verification system shall detect when stimuli to the SUT is applied out of order.	Verif. eng.	TSR078 TSR074 TSR065	SG007	D
FSR038	The verification system shall detect when wrong stimuli is applied to the SUT.	Verif. eng.	TSR078 TSR074 TSR065	SG008	D
FSR039	The verification system shall only supply a test result if no error is detected when supplying stimuli	Verif. eng.	TSR077 TSR066	SG004 SG006 SG007 SG008 SG009 SG010 SG013 SG014 SG015	D
FSR042	The verification system shall detect when stimuli is applied to the SUT too fast.	Verif. eng.	TSR078 TSR065	SG013	D
FSR062	The verification system shall detect when stimuli is applied to the SUT too slow.	Verif. eng.	TSR078 TSR065	SG013	D

Note 1: Some SGs included in the table cannot be found in Table 3.5.

Note 2: Some TSRs included in the table cannot be found in Table 4.2.

4.2 Elicitation of technical safety requirements

The next activity in the project was the elicitation of the TSRs.

This activity corresponds to [10, Part 4, Clause 6], “Specification of technical safety requirements”, and concerns specifying the TSRs, safety-related requirements specific for the implementation based on the previous implementation-independent FSRs.

The process of deriving the TSRs is similar to the process of deriving the FSRs, as the specifications from [10, Part 8, Clause 6] also apply to the TSRs. However, there is no explicit specification in ISO 26262 that each FSR needing to be the source of at least one TSR, as in the case with the SGs and the FSRs. Instead it is only specified that “[the] technical safety requirements shall be specified in accordance with the functional safety concept” in [10, Part 4, 6.4.1.1].

This activity was not entirely trivial, as the project so far principally had concerned the requirement specification for a general verification system, which resulted in that the knowledge about the preliminary architecture or implementation was limited at best. TSRs related to planned and prospective safety measures were therefore elicited first, with the rest of TSRs being elicited to fulfill the goal that all FSRs should be covered by at least one TSR each.

Five of the 18 TSRs elicited during this activity can be seen in Table 4.2. These five TSRs are the requirements that are related to the implementation in the final implementation phase of the project. One of these TSRs, TSR078 was later decomposed into two new TSRs, leading the number of communication monitor-related TSRs to rise to seven out of a total of 20 TSRs. The reason for this decrease in number of TSRs compared to FSRs could mainly be attributed to the implementation of some safety measures covering several safety functions. An example of such safety measure is monitoring of the communication channel, as a good monitor should cover both the error of sending no signal or the error of only sending a partial signal.

Table 4.2: An example of the elicited TSRs.

ID	Summary	Assigned to (element)	Source Requirement ID	ASIL
TSR065	The verification system shall log all output sent to the SUT to a storage medium.	Verif. eng.	FSR061 FSR048 FSR040 FSR042 FSR043 FSR038 FSR037 FSR036 FSR033 FSR044 FSR062 FSR045	D
TSR066	The verification system shall use memory protection to ensure that the memory integrity is preserved.	Verif. eng.	FSR030 FSR056 FSR048 FSR039 FSR061	D
TSR074	The verification system shall perform a self-test on its outputs before supplying stimuli to the SUT.	Verif. eng.	FSR044 FSR033 FSR036 FSR037 FSR038 FSR043	D
TSR077	If any error is detected during the self-tests, the verification system shall not run the test case.	Verif. eng.	FSR039 FSR050	D
TSR078	The verification system shall monitor its inputs and outputs for erroneous signals.	Verif. eng.	FSR043 FSR033 FSR036 FSR038 FSR037 FSR044 FSR042 FSR062 FSR047 FSR051 FSR052	D

Note: Some FSRs included in the table cannot be found in Table 4.1.

4.3 Implementation phase preparation

Before continuing onto the system design it had to be decided what to implement in the concluding implementation phase, as that decision would affect what needed to be designed in the system design activity.

4.3.1 Preparation for the implementation phase

With the SGs, FSRs and TSRs elicited, most projects following ISO 26262 would be ready for the system design activity. If the final goal of this project would have been to implement a complete and general verification tool, this project would also had continued directly into the system design activity. But as the objectives of this project only included a limited implementation and limited verification activities, it was time to find the interesting functionality which would be implemented later.

Therefore, looking closer at the elicited safety requirements, it was concluded that providing safe communications to the SUT from the verification system, and vice versa, was very important if a verification system and its verification should be considered safe. This being consistent with the nature of system testing and the related black-box testing, as described in

Section 2.2. A decision was therefore taken to focus on an element of the verification system that could help provide confidence in the communications from the verification system. This element would preferably verify the communications from the verification system being correct, in relation to a test case specification. This element could then later, in theory, be extended to also handle the communications to the verification system from the SUT.

After the decision to focus on the communication aspect of the verification system, a few approaches on how to provide confidence in communications were investigated, with two solutions considered feasible to implement in this project. The first solution involved creating an element inside the main verification engine, combined with some kind of loop-back support, so that the verification engine could verify its own outputs. The other solution involved creating a monitor outside the main verification engine, which would monitor the communication channels and report if any signal on the channels would deviate from the specifications in the test case.

Considering the two alternatives, the concept of ASIL decomposition, as described in Section 2.1.2, was brought up, which was considered to work well with the external monitor approach, as ASIL decomposition would allow the required ASIL D safety measures concerning the reliable communication to be moved out of the main verification engine. This could additionally provide the possibility of simplifying the future design and implementation of the verification engine by moving even more safety functionality from the engine to other independent elements, and in this way change the ASIL of the verification engine to a lower ASIL. Additionally, the safety standard IEC 61508, upon which ISO 26262 is based, mentions a similar safety measure, under the name of "diverse monitor" [9, Part 7, C.3.4], which lends credibility to monitoring as way to ensure safe operation.

4.3.2 Choice of hardware

One important issue when performing ASIL decomposition is the requirement that the "decomposed safety measures", i.e. the two elements to which the decomposed requirements are allocated to, must be proved to be "sufficiently independent" to prevent dependent failures risking the safety of the combined system [10, Part 9, 5.4.3]. As this independence is simpler to prove if the two elements does not share the same hardware, the choice of hardware for the communication monitor was slightly limited, as the platform being considered for the future versions of the verification engine is the National Instruments CompactRIO [8] industrial controller. At the same time it was clear that a communications monitor for a verification system with future possibilities of multiple parallel interfaces require support for a high level of parallelism and speed. Therefore a conclusion was made that an FPGA would be suitable as a platform for a communication monitor.

4.4 System design

System design [10, Part 4, Clause 7] is the clause and activity in ISO 26262 which follows after the “Elicitation of the technical safety requirements”-clause, and the goal of the system design activity is to produce a design for the complete system, to which the TSRs then can be allocated. The system design activity is also the first activity in the project where the communication monitor, as derived in Section 4.3 and later further developed in Chapter 5, comes into the process in this project.

The system design activity covers the following subactivities:

1. Design of the system to be developed.
2. Allocation of TSRs to system design elements.
3. Specification of internal and external interfaces and hardware-software interfaces.
4. Design of measures for identification and elimination/mitigation of systematic failures.
5. Design of measures for detection and control/mitigation of random hardware failures.
6. Specification of requirements concerning production, operation, service and decommissioning.

While an ISO 26262 project should cover most of these subactivities and their specifications in ISO 26262, Subactivity 5 and Subactivity 6 were not covered in this project, in accordance to the limitations put upon the project from the objectives. Subactivity 5 was omitted because of the extra work needed to perform this analysis was not seen as proportional to the results it would provide, as it would mainly consist of contacting the hardware manufacturers and figuring out the causes and probabilities of the random hardware failures. Subactivity 6 was omitted because the project currently do not have any plans for any production or large scale operation. Subactivity 4 was also limited to a deductive analysis of the communication monitor, as another fault analysis for the whole system falls outside of the objectives of the project, as many of the systematic failures of the verification engine would already be covered in the previous HARA. The result of the systematic failure analysis of the communication monitor was therefore moved to the later hardware and software design activities, which means it will be presented later in this report, in Chapter 5, instead of here.

As Subactivity 1 and Subactivity 2 from the list above both cover the overall design of the system, they should both include the communication monitor as an element in the overall system design. ASIL decomposition was therefore as planned performed on TSR078, "The verification system shall monitor its inputs and outputs for erroneous signals.", as found in Table 4.2, decomposing the TSR into TSR080, "The communication monitor shall monitor the communication channels to and from the verification engine for erroneous signals.", and TSR094, "The verification engine shall provide communications to the SUT." This resulting in the new TSR080 being allocated to the verification engine and TSR094 being allocated to the communication monitor. The initial TSR078 and the two new TSRs can be seen in Table 4.3, a table also including references to the later SWSRs and HWSRs, as found in Table 4.6 and Table 4.5. In accordance to ASIL decomposition TSR080 was assigned ASIL D(D) and TSR094 was assigned QM(D).

As explained in Section 4.3.2 and in Section 2.1.2, an ASIL decomposition has to be followed with proof of sufficient independence to prevent dependent failures, which includes both systematic failures and random hardware failures. While the independence from dependent systematic failures is the most important independence to prove, the system design had not yet been performed at the stage, which means it would yet needed to be proved after the system design activity. However, proving that the elements are sufficiently independent to prevent the same random hardware failures occurring in both elements can easily be made. As the verification engine and the communication monitor will be implemented on two different architectures, a National Instruments CompactRIO and a Xilinx FPGA, the risks of both elements exhibiting the same random hardware failures are minimal.

After the ASIL decomposition the first subactivity of the system design activity could be performed. The result from this subactivity can be seen in Figure 4.1, showing the system when being divided into two main elements, a verification engine and a communication monitor. A voltage divider element is also included between the two main elements, as explained in the later Section 5.2. The SUT is also shown in the figure, to show how the verification system would be connected to the SUT during verification.

The finished system design subactivity in combination with the ASIL decomposition also resulted in the second subactivity being possible, the allocation of the TSRs to the elements of the system, resulting in the so called Technical Safety Concept (TSC). This allocation resulted in all TSRs,

Table 4.3: The decomposed TSR078 and the two new TSRs, from the ASIL decomposition.

ID	Summary	Assigned to (element)	Destination Requirement ID	Source Requirement ID	ASIL
TSR078	The verification system shall monitor its inputs and outputs for erroneous signals. [DECOMPOSED]	Verif. eng.	TSR094 TSR080	FSR043 FSR033 FSR036 FSR038 FSR037 FSR044 FSR042 FSR062 FSR047 FSR051 FSR052	D
TSR080	The communication monitor shall monitor the communication channels to and from the verification engine for erroneous signals.	Comm. monitor	SWSR095 HWSR093 SWSR092 HWSR091 HWSR090 SWSR089 SWSR088 SWSR086 SWSR085 SWSR084 SWSR083 HWSR082 HWSR081 SWSR087 SWSR098 HWSR097 HWSR096	TSR078	D(D)
TSR094	The verification engine shall provide communications to the SUT.	Verif. eng.		TSR078	QM(D)

Note: Some FSRs included in the table cannot be found in Table 4.1.

with the sole exception of TSR080, being allocated to the verification engine, with TSR080 being allocated to the communication monitor as planned.

Concerning the third subactiviy, specifying the internal and external interfaces, all interfaces and signals of the system can be seen in Table 4.4. This includes what is known of the Hardware-Software Interface (HSI) from the system design. As the actual HSI is specified by the pin-out on the FPGA and the layout of the development kit used in the project, the more minute details of the HSI documentation can be found in the documentations supplied with the development kit at [2]. As seen in Table 4.4 and discussed earlier in the section, as nothing is known or assumed about the internals of the verification engine, the number of related interfaces are rather few.

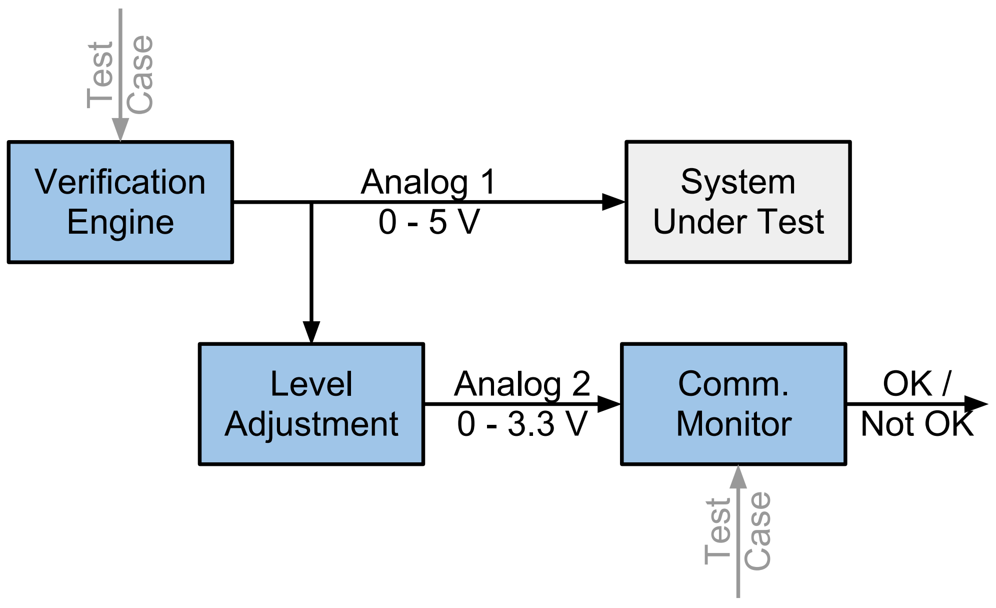


Figure 4.1: Model of the system design with verification engine and communication monitor. A SUT is also included for clarity of connection to system.

Table 4.4: System interfaces and signals.

Name	Type	Description	Software counter-part
Analog 1	Analog output / Analog signal 0 - 5 V	Analog output from verification engine.	N/A
Analog 2	Analog input / Analog signal 0 - 3.3 V	Analog input to communication monitor ADC.	Input samples (SPI-Bus)
Comm. OK / Comm. Not OK	1-bit digital output (LED)	One if no error has been detected on the communication signal Zero if error has been detected.	Monitor-ing result (General I/O)

4.5 Elicitation of hardware safety requirements and software safety requirements

Following the system design activity, the next activities in the project, and consequently also the last activities in the requirement phase of the project, is the elicitation of the HWSRs and the SWSRs.

The elicitation of HWSRs and the elicitation of SWSRs are two activities stemming from one clause each in Part 5 and Part 6 of ISO 26262, namely [10, Part 5, Clause 6] and [10, Part 6, Clause 6]. Both clauses concern the elicitation of the lowest levels of safety requirements. These two types of safety requirements are most easily differentiated in what parts of the system they foremost relate to. The HWSRs are safety requirements related to the physical hardware of the item and its closest environment, while the SWSRs more concern the software functionality and functional reliability of the item.

In this project the communication monitor is the only element for which these two types of safety requirements will be elicited, as the verification engine by earlier choice is not as well known and specified. However, the choice of using an FPGA as platform for the communication platform does complicate elicitation process at this level a little, as some aspects of an FPGA could be considered both software and hardware. The requirements were therefore not as easy to allocate as when using other platforms, which usually has a more distinct division of what aspects of the system are considered hardware and what aspects are considered software. This results in some of the safety requirements being elicited as SWSRs possibly could be reassigned as HWSRs, and vice versa.

The HWSRs and SWSRs can be found in Table 4.5 and Table 4.6 respectively. It is important to note that some of the requirements were not implemented in the communication monitor in the final phase of the project, such as HWSR093, which lies outside the scope of the complete project, and SWSR086, SWSR087, SWSR088 and SWSR098, which lie outside the scope of the planned communication monitor.

Table 4.5: The elicited HWSRs of the communication monitor.

ID	Summary	Assigned to (element)	Source Requirement ID	ASIL
HWSR081	The communication monitor shall be implemented on an FPGA to reduce the chances of the same faults in the test tool and the monitor.	Communication Monitor	TSR080	D
HWSR082	The communication monitor shall be independent of the test system with the exception for the signal inputs and signal outputs.	Communication Monitor	TSR080	D
HWSR090	The communication monitor shall not cause any disturbances on its inputs.	Communication Monitor	TSR080	D
HWSR091	The communication monitor shall not cause any unintended signal on its outputs.	Communication Monitor	TSR080	D
HWSR093	The communication monitor shall be tested in the same environment as the verification engine.	Communication Monitor	TSR080	D
HWSR096	The communication monitor shall be tested in an environment complying with the environmental requirements specified by the manufacturer of the FPGA development kit.	Communication Monitor	TSR080	D

Table 4.6: The elicited SWSRs of the communication monitor.

ID	Summary	Assigned to (element)	Source Requirement ID	ASIL
SWSR095	The communication monitor shall be able to signal when an error is detected in a signal it has received on its inputs.	Communication Monitor	TSR080	D
SWSR083	The communication monitor shall be able to detect when no stimuli is sent.	Communication Monitor	TSR080	D
SWSR084	The communication monitor shall be able to detect when only partial stimuli is sent.	Communication Monitor	TSR080	D
SWSR085	The communication monitor shall be able to detect when there is an error in the stimuli.	Communication Monitor	TSR080	D
SWSR086	The communication monitor shall be able to detect if the SUT does not supply any reply to the stimuli.	Communication Monitor	TSR080	D
SWSR087	The communication monitor shall be able to detect if the SUT does only supply a partial reply to the stimuli.	Communication Monitor	TSR080	D
SWSR088	The communication monitor shall be able to detect if the SUT gives an unfeasible reply to the stimuli.	Communication Monitor	TSR080	D
SWSR098	The communication monitor shall check all signals sent with error detecting codes for errors.	Communication Monitor	TSR080	D
SWSR092	The communication monitor shall be able to monitor signals used in automotive environments.	Communication Monitor	TSR080	D

4.6 Result and discussion

This phase of the work has two main results; a requirement specification and a system design. The requirement specification consists of four categories of safety requirements:

1. Implementation-independent FSRs allocated to general elements of the system.
2. Implementation-specific TSRs allocated to the elements of the system design.
3. HWSRs allocated to the hardware aspect of the communication monitor.
4. SWSRs allocated to the software aspect of the communication monitor.

The system design describes the overall design of the item/system, and with the HWSRs and SWSRs the foundation for the following phase, the design and implementation of a communication monitor. With this phase completed, the limited safety case of the project, as started in Section 3.1 is also finished, containing the safety plan, the item definition, the requirement specification with FSRs, TSRs, HWSRs and SWSRs and the system design.

While there were no large issues in this phase, as with the HARA in the preparation phase, there was some uncertainty concerning the definitions of and differences between FSRs, TSRs and HWSRs/SWSRs. Looking at the previous works this is not an issue unique to this project. In the work of Gradin et al. [7] for example, their initially elicited FSRs were very detailed, at least compared to this project. This in turn made it hard for them to differentiate between their FSRs and TSRs, which made the elicitation of the TSRs harder. In this project the difference was made at the *implementation-independent functional* safety requirement and *implementation-specific technical* safety requirements. The first set of requirements, FSRs, describing an implementation-independent safety function planned to mitigate the hazard and the second set of requirements, TSRs, describing specific technologies or safety measures used to provide the mitigation functionality, such as memory protection.

Related to the management of safety requirements, another minor issue lies in the "specification of the technical safety requirements"-clause [10, Part 4, Clause 6], as it could be interpreted to stand in odds with the specification and management of safety requirements clause [10, Part 8, Clause 6]. While [10, Part 8, Clause 6] require that all safety requirements always should be assigned to an element, this assignment to an element can not, and should not according to [10, Part 4, Clause 7], be performed until after the system design activity is completed. This creates a gap between eliciting the TSRs and finishing the system design, where the TSRs are not assigned to elements of the system as required. While this is a minor issue, as in the end, all TSRs are assigned to design elements, it is an example of the inconsistency sometimes found between different parts of ISO 26262.

Another example of inconsistency in the standard from this phase of the project is how different parts of ISO 26262 describe similar activities and specifications in different ways. This phase includes two parts of ISO 26262, Part 3 and Part 4, and even though the elicitation of FSRs activity found in Part 3 and the elicitation of TSRs activity found in Part 4 are similar it is sometimes apparent that they are written by different authors with different focus. As an example, the functional safety concept clause [10, Part 3, Clause 8] begins with stating “The functional safety requirements shall be specified in accordance with ISO 26262-8:2011, Clause 6” as first specification before continuing onto the specifics of the elicitation activities. In contrast to this, the specification of the technical safety clause [10, Part 4, Clause 6] do not mention this important clause, that is [10, Part 8, Clause 6], until in its sixth subclause, and then only in passing; “The technical safety requirements shall specify the necessary safety mechanisms (see ISO 26262-8:2011, Clause 6) [...]” [10, Part 8, 6.4.2.2]. Thus giving reason to think that [10, Part 8, Clause 6], is less important when eliciting the TSRs.

Chapter 5

Implementation

With the system design completed and the safety requirements for the communication monitor elicited in the activities of the previous phase, the work on designing and implementing the communication monitor could be started. In accordance to the earlier phases of the projects, the emphasis of this phase was put on the design and implementation of the communication monitor. The monitor was also put through a limited verification and validation (V & V) process for investigation of ISO 26262 requirements on the V & V process.

5.1 Signal monitoring and syntactic pattern recognition

The design of the communicator monitor started with four questions:

1. What kind of monitor is preferable to use in a verification system?
2. What kind of signal would the monitor need to monitor?
3. What aspects of the signal would be interesting to monitor?
4. How could this monitoring be implemented?

Trying to answer the first question, the two kinds of monitoring normally used for runtime monitoring is either an on-line monitor or a data logger. The on-line monitor would in this case correspond to monitoring the communication channels at runtime and simultaneously evaluating if the signal follows its specification or not. A data logger would instead correspond to only logging the signals during the runtime, and evaluating the logged data later, possibly on another platform. While the latter approach may be appealing, as it once again moves safety functionality out from the communication monitor into some later evaluation system, the on-line monitor approach also provides some advantages. One advantage is that an on-line monitor could be extended with a simple output connected to the verification engine, and in this way report back to the verification engine when an error is detected, reducing the time needed when running tests with the unreliable verification engine. Another advantage of an on-line monitor is that a data logger possibly would require a large amount of storage space to save all the incoming data, while an on-line monitor could provide better possibilities to filter out unwanted data with its increased capabilities. Because of these two advantages, a decision was taken to design and implement an on-line monitor, planned to be used in tandem with a verification engine.

Continuing with the second question a previous project at Semcon came to mind, where an existing verification system[1] was expanded to improve its ability to test a safety critical steer-by-wire Electronic Control Unit (ECU). In this previous project two different types of signals were used in the communication to and from the ECU, messages over a CAN-bus and an aperiodic analog voltage signal. With this verification system readily available, complete with test cases for testing the steer-by-wire ECU, focusing on one of these two inputs of the ECU should make a demonstration of the communication monitor easier. Especially as using the previous verification system for demonstration purposes would remove the need to procure a completely new verification engine for this project. As the CAN-bus by definition is employing fairly reliable communication by itself, including error-detecting codes in the sent messages, it was clear that there was a greater need to provide monitoring for the aperiodic analog signals. This led to the decision to create an analog signal communication monitor.

More specifically, the analog input of this steer-by-wire ECU represented the front wheel angle of a vehicle as a continuous aperiodic analog signal, with a voltage range varying between 0.307 and 1.768 V, corresponding to a front wheel angle between -45° and 45°, as seen in Figure 5.1. This also leads into the answer to the third question, what features of the signal would be interesting to monitor? While the only single unique feature of a one-dimensional aperiodic analog signal is its value over time, it would be more interesting to identify the shape of the signal around this value, as for example seen in the derivative of the signal at the same time. This way it would be easier to identify the current state of the analog signal and evaluate if the signal is correct or not.

[1]This is the same verification system as mentioned in Section 3.2.

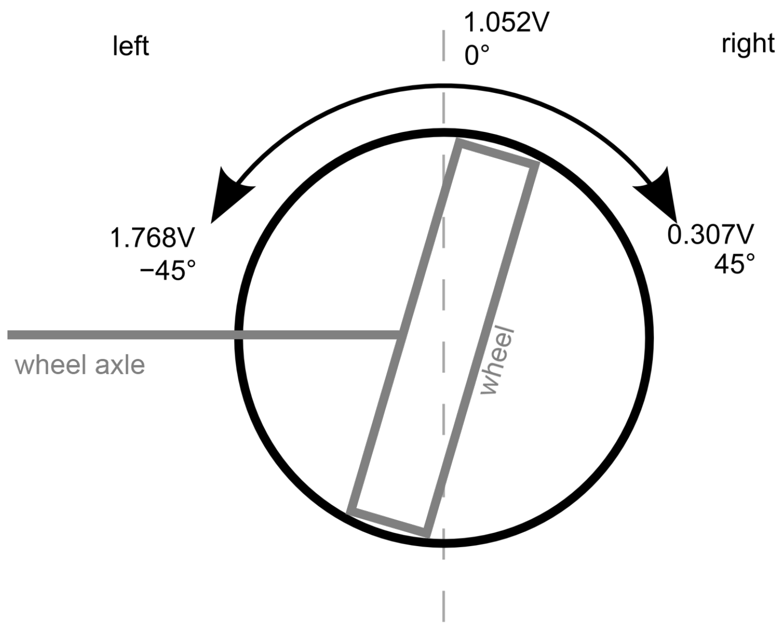


Figure 5.1: Wheel angle and corresponding voltage level for a steer-by-wire ECU.

With these three answers ready, the final question remained, how to perform a monitoring of this analog signal? There are two kinds of solutions to this problem depending on how well the input signal is known. If the input signal is well known, the best way to identify and subsequently monitor the signal is to compare the actual input signal with the signal you would expect from the other system. However, if the input signal is not well known, it is best to perform the identification and monitoring based on special features of the signal instead. An example of the latter type of solution is looking at a periodic signal in the frequency domain, where for example a periodic square wave would create a higher amount of high frequency components than a sinusoidal signal of the same frequency, making the signals distinguishable from each other, even if features like amplitude or (common) frequency is uncertain[2].

In the case of our verification system, the signals sent from the verification engine to the SUT would be specified by a test case specification, so the first type of solutions should fit this project best. This approach is most often called pattern recognition. Usually pattern recognition is performed by comparing a signal with several pre-defined patterns, classifying the input

[2]The monitoring of periodic signals are not further explored in this project.

signal to one of the available patterns. But in this project an added level of determination was needed to the matching, so a special implementation of pattern recognition was employed. The difference is that only one concurrent pre-defined pattern is used for the pattern matching each given time. However, this pre-defined pattern is exchanged to another pattern, at each distinct change of the input signal pattern.

The concept of pattern recognition covers several types of techniques, which meant that another choice needed to be taken for the following question. What pattern recognition technique should be used in the communication monitor? A short study of the subject resulted in the four techniques as found in Jain et al. [13], an introductory article to statistical pattern recognition, and Schalkoff [18], a book on the general subject of pattern recognition.

These four types of pattern recognition methods are presented in Table 5.1, with much of the included data based on [13, 18]. This table includes the names of the four kinds of pattern recognition and a short description on how they match an unknown signal to a pre-defined pattern, and the pros and cons of each method. The result of this study was eventually the conclusion that the syntactic pattern recognition technique would fit this project best, as the other three types each had disadvantages making them unsuitable. Standard template matching is inflexible and sensitive to jitter, statistical pattern matching often proves to need complex pre-processing of the signal to be efficient, and neural networks have a uncertain suitability in safety critical systems.

To explain the basics of the syntactic pattern recognition used in this project, a continuous syntactic pattern recognition based on the derivative of the signal, an example has been included below and in Figure 5.2. In Figure 5.2 the input signal is divided into smaller sections of equal size, a division in this project naturally occurring by the sampling of the signal. Each of these sections is then assigned a code, based on the angle/slope of the signal between the start and end of each section. In the figure there are three codes, A for a positive slope, B for a small or no slope, and C for a negative slope. The signal in the figure can therefore be classified, or interpreted, as a string of codes, "ABCCBB" based on the slope of each section. In normal syntactic pattern recognition this string of codes would have been searched for a number of pre-defined patterns, for example "BB" or "CC", , and return the placement of the pattern in the signal, in the case of "BB" or "CC" at the end or the middle of the input signal. In this project though, the pattern recognition is always performed at the current input signal, and the input is only compared to a single pattern, which corresponds to a sequence of the same code. In the example this means that the syntactic pattern recognition first expects an A, then a B, then two C's, then two B's and so forth, and if

Table 5.1: Comparison of pattern recognition methods.

Name	Description	Pros	Cons
Template matching	Compares the input signal with a stored copy of the expected signal. Makes decision on whether the input signal matches the stored signal or not based on the size of the difference of the two signals in a specified time interval.	Simple to implement.	Heavy on computation and memory. Not as flexible as other methods.
Statistical	Extracts features from the input signals and statistically decides which pattern the input signal best corresponds to based on the extracted features.	Versatile with skill as many features can be extracted.	Can be hard to find usable features. Sometimes requires complicated features to give acceptable results. Complex features computationally heavy to extract.
Syntactic/ structural	Divides the input signal into smaller parts which are classified into "shapes". Given the classification the syntactic pattern recognition the searches for sequences of these shapes in the signal.	Simple. Unimportant features can be filtered away by choice of shapes.	Choice of shapes could lead to loss in detail in comparison between actual signal and expected signal.
Neural networks	Uses a network with layers ofğ weighted nodes, where the weight is set by training the system with already known patterns and their expected result.	Advanced method. Have a history of being used in redundant monitoring.	Unclear suitability in safety critical systems.

the two B's would come before the two C's the matching would fail and the result would be an indication of an incorrect signal.

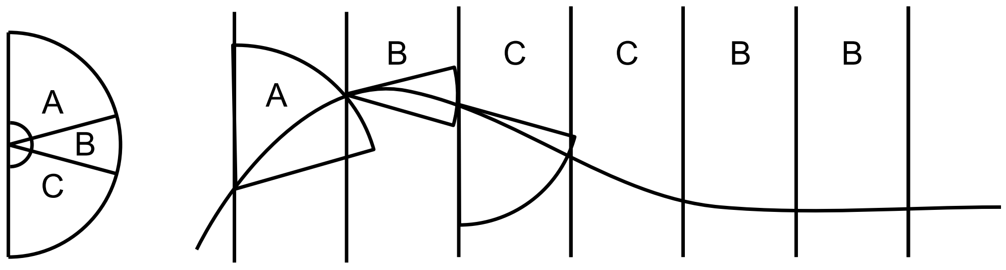


Figure 5.2: Example of the coding in a syntactic pattern recognition approach.

As the signal monitored in the project is a piecewise-linear signal some further adjustments were also performed to the pattern recognition employed in the implementation. The syntactic encoding of a piecewise-linear signal results in each linear *piece* of the signal being encoded into a sequence consisting of only a single *code*, with the length of this sequence being directly related to the length of the piece in the original signal. The expected signal was therefore saved as expected code and the number of samples expected to result in the same code, simply called a *sequence*. An example of this is included in Figure 5.3, where the definition of a *piece* of the input signal and the difference between *code* and *sequence* are also shown.

The implementation also has another minor difference from the example of Figure 5.2. It uses a finer granularity when encoding the signal into codes, resulting in the input signals being encoded into one of 29 logarithmically spaced codes, instead of one of only 3 codes as in the example.

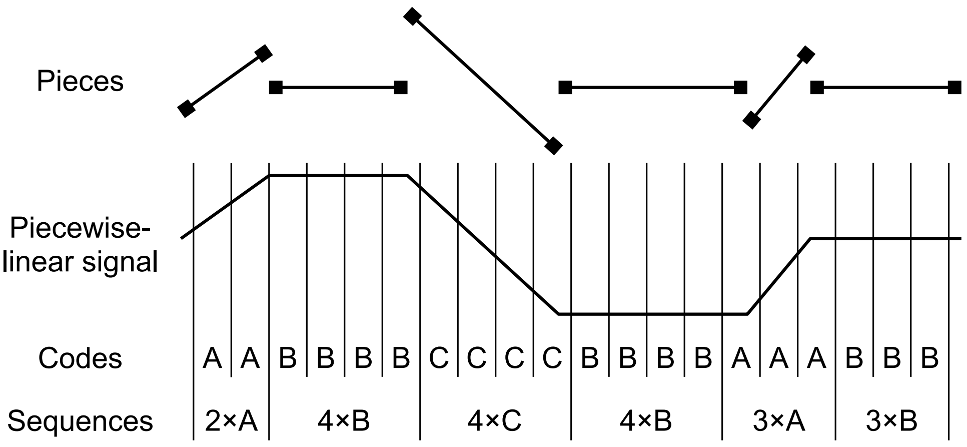


Figure 5.3: Example of the syntactic pattern coding of a piecewise linear signal, using the same encoding as in Figure 5.2.

5.2 Hardware development

The main hardware used in the project is an FPGA development kit from Avnet called Xilinx Spartan-6 FPGA LX75T Development Kit [2]. In addition to the FPGA-chip the development kit includes an on-board 16-bit Analog-to-Digital Converter (ADC) sampling at 60 samples per second[3], used in the project to receive the analog signals from the verification engine, connected as seen in Figure 4.1, and a RS-232 serial communication chip, later used for verification and demonstration purposes. This means no additions or modifications were done to the board, making the actual hardware development of the main communication monitor minimal. But, since the output of the verification system used for the final demonstration and the Arbitrary Waveform Generator (AWG) used in verification both had a voltage range of 0–5 V and the ADC on the FPGA only is specified to handle a voltage swing of 0–3.3 V, the output voltage from these two systems had to be adjusted to the levels acceptable by the ADC, as to reduce the risk of damaging the ADC if the any of the systems would fail and output voltages over 3.3 V. A voltage divider was therefore designed and constructed to bring the signal voltage down to acceptable levels. The following two sections go into the specifics of the hardware design of these two elements in accordance to [10,

[3]While this may be considered a slow ADC in some implementations, the low sampling speed is no problem in this project, as a signal like a wheel angle should be considered a rather slow signal. The later software design was also made with scaling to faster ADCs in mind.

Part 5, Clause 7].

5.2.1 Hardware design: Communication monitor

As mentioned in the previous section the hardware of the actual communication monitor was an off the shelf FPGA development kit from Avnet [2], with no additions or modifications. Any actual hardware design was therefore not performed for the communication monitor[4]. A finished hardware component like this should instead, according to [10, Part 8, Clause 13], be qualified before use, and the recommended qualification method for complex hardware components is to include the hardware in the integration and verification activities of [10, Part 5]. But as the activities specified in these clauses are never reached in this project, being outside the objectives, this qualification would not be performed. But as the manufacturer of the board, Avnet, is a reputable company, endorsed by the FPGA-chip manufacturer Xilinx, the development kit board should be reliable enough for the monitor implementation in this project. The actual qualification would instead have to wait for later, when or if the monitor would be further developed and the verification activities would be performed.

A picture of the hardware card with the components most important to the project is seen in Figure 5.4.

[4]For more information concerning the board it is instead best to consult the documentation supplied with the card, as found at [2].

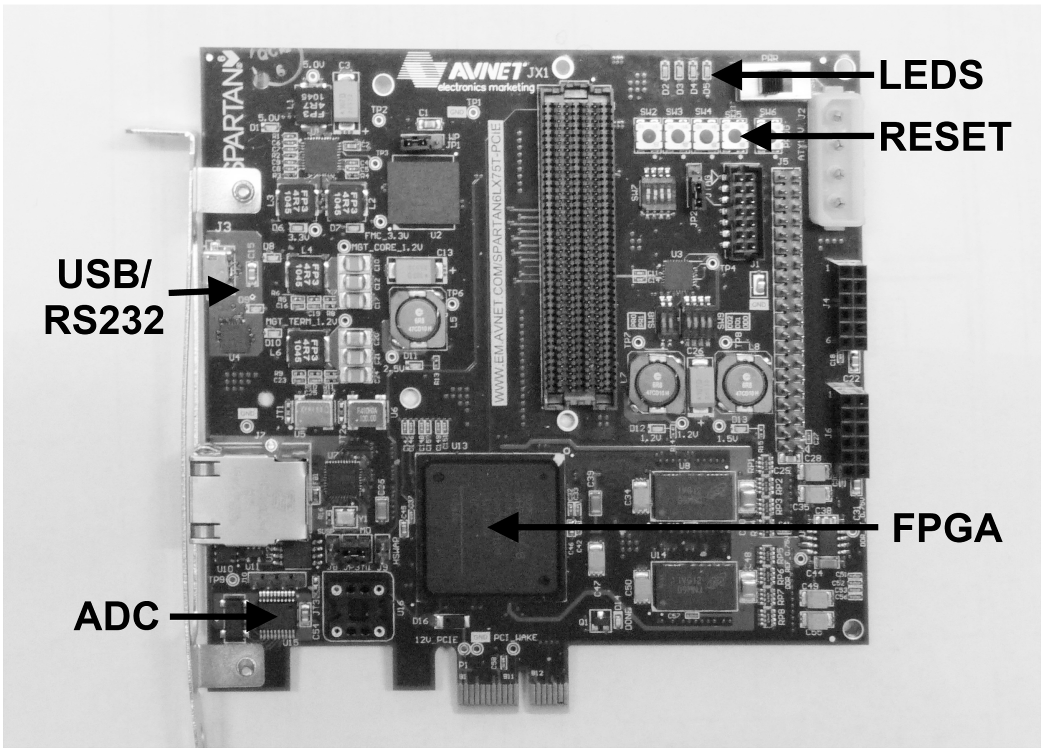


Figure 5.4: Picture of the development board used in the project with the most important components marked.

5.2.2 Hardware design: Voltage divider

The voltage divider used in this project is a simple circuit, consisting of two resistors, R1 and R2, connected in series. R1 has a resistance of 220 Ω and R2 has a resistance of 120 Ω. To adjust the voltage level to the ADC the input voltage is applied over both resistors, while the output voltage is only taken over R1, which results in a maximum voltage of $220\Omega \times 5\text{V}/(220\Omega + 120\Omega) \approx 3.25\text{V}$. A schematic of the circuit can be found in Figure 5.5, with the connections to the signal source shown on the left hand side and the connections to the ADC shown on the right hand side.

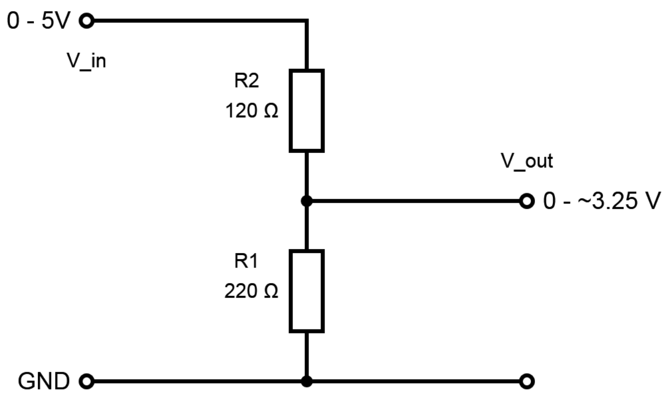


Figure 5.5: Schematic of the voltage divider.

5.2.3 Hardware safety analyses

In accordance to [10, Part 4, 7.4.5.2], which specifies that safety critical elements implemented on FPGAs should be developed with a combination of Part 5 and Part 6, only one safety analysis was performed in the project. The description and results of this analysis can be found later in Section 5.3.3 as it also had to be based on the software design, as described in Section 5.3.2.

5.3 Software development

5.3.1 Initiation of product development at the software level

In contrast to the first clauses of the system level and hardware level development parts of ISO 26262, [10, Part 4 & Part 5], which mainly concern planning activities, the first clause of [10, Part 6] also concerns an activity affecting the actual development work of the item. This activity consists of choosing method, tools and language for the subsequent software development activities. Coding guidelines, either internal or of a standard such as MISRA C, are also to be considered according to the clause. And as FPGA development should be performed with a mix of both Part 5 and Part 6 [10, Part 4, 7.4.5.2] these specifications should be important for a project like this, as choice of Hardware Description Language (HDL) and synthesis tool are two of the elements of software design that could influence the result of

FPGA development significantly.

Two decisions were taken as result of this activity. The first was the decision to use the FPGA development tool PlanAhead, one of the tools developed by Xilinx specifically to be used when developing hardware to be used on their FPGAs, which should lessen the chance of errors stemming from incompatibility or not supporting functionality of the FPGA. The second decision taken was to use VHDL as HDL language for the implementation. This is mainly because VHDL inherently meets the specifications provided in [10, Part 6, 5.4.6], both by providing good support of modularity and by its nature as a HDL providing good "support for embedded real time software", but also as it by definition helps with the development guidelines as of [10, Part 6, Table 1], for example by being inherently strongly typed. The rest of the specifications was either considered to be fulfilled by inherent properties in VHDL or the synthesis tool or was assumed being possible to fulfill by good programming practices and code reviews.

5.3.2 Software architectural design: Communication monitor

Looking at the basics of syntactic pattern recognition, as described in Section 5.1, it can be seen that a syntactic pattern recognition implementation would require three main components:

1. A coder, a component which encodes the analog signal into a sequence of codes based on the shapes of input sections, in this case the derivative of the signal, i.e. the difference of the current and previous sample.
2. A comparison component, a component where the latest code from the coder is compared with the expected code as read from memory and the result if the signal is correct or not is given.
3. A memory, where the expected codes and their expected sequence length are stored.

In addition to these three components the system also needs a Serial Peripheral Interface (SPI) Bus Master component, also implemented in VHDL, to handle the communications to and from the on-board ADC-chip.

The results of the design can be seen in Figure 5.6. The figure also includes the ADC for clarity, even though the ADC functionality is performed on the ADC-chip present on the development kit. Description of all the interfaces and signals seen in the figure can be found in Table 5.2.

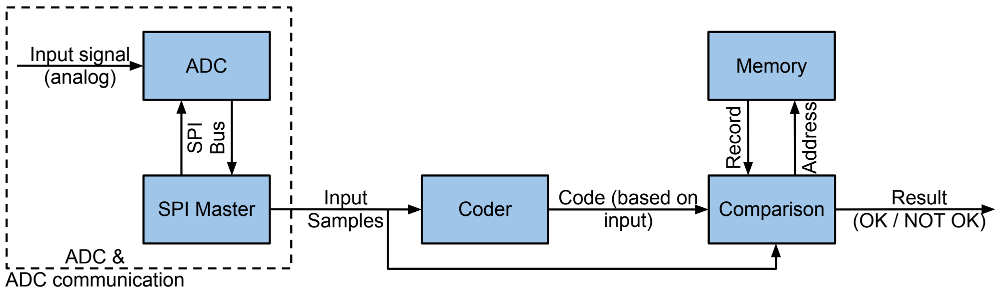


Figure 5.6: High level model of the communication monitor.

Table 5.2: Communication monitor interfaces and signals for Figure 5.6.

Name	Type	Description
Input signal	Analog input	Analog signal from external source. Continuous piecewise-linear with at least 200ms between transitions.
SPI Bus	SPI Bus	Bus handling the communication to and from the on-board ADC.
Input samples	16-bit unsigned	Samples from ADC converted from the analog input signal.
Code	8-bit signed	Code based on log_2 of the derivative of the input signal.
Record	52-bit memory record	A 52-bit memory record from memory as in Table 5.3.
Address	12-bit memory address	A 12-bit memory address for accessing the memory.
Result	4-bit digital output	Result output, outputs an error code in binary encoding as of Table 5.4.

5.3.3 Software safety analysis

As mentioned in Section 5.2.3 a safety analysis of the design is recommended both in [10, Part 5, Clause 7] and [10, Part 6, Clause 7], which both apply to programmable hardware such as FPGAs. According to the introduction to [10, Part 9, Clause 8], this safety analysis could for example be a Fault Tree Analysis (FTA), so in accordance to this example FTA was chosen as the method to perform the safety analysis of the communication monitor on the architectural design.

The clause specifying how to perform these safety analyses [10, Part 9, Clause 8] states that the goal of all safety analyses should be either be verifying that the safety requirements are already fulfilled by the design, or to identify mitigation measures needed to ensure that the safety requirements are fulfilled. This means that the safety requirements most related to the communication tool are the requirements to foremost base the safety analysis on, with these two requirements being TSR080 and TSR076, as found in Table 4.2. Based on these requirements the most severe fault in the communication monitor that would risk the safety of the verification system is a type II error in the communication monitor. A type II error of the communication monitor corresponding to the monitor failing to successfully report an error existing in the signal. An FTA of this error and possible countermeasures was therefore performed with the communication monitor in mind. This FTA can be seen in Figure 5.7.

Complying with the aim of the implementation phase of providing a working prototype, not a complete product, only one of these safety measures was implemented in the final design. The safety measure implemented was the reporting of the result of the code comparison with more than a single bit as planned earlier in the design, as seen in Figure 4.1. The design was instead modified to use a four bit signal to output an error code instead of the initial one-bit digital output, as seen in Table 5.2[5].

[5]Support for serial communication was also introduced later into the design to help with the verification of the communication monitor. This support could possibly be used in future versions of the verification system to provide another redundant output for reporting errors to the verification engine.

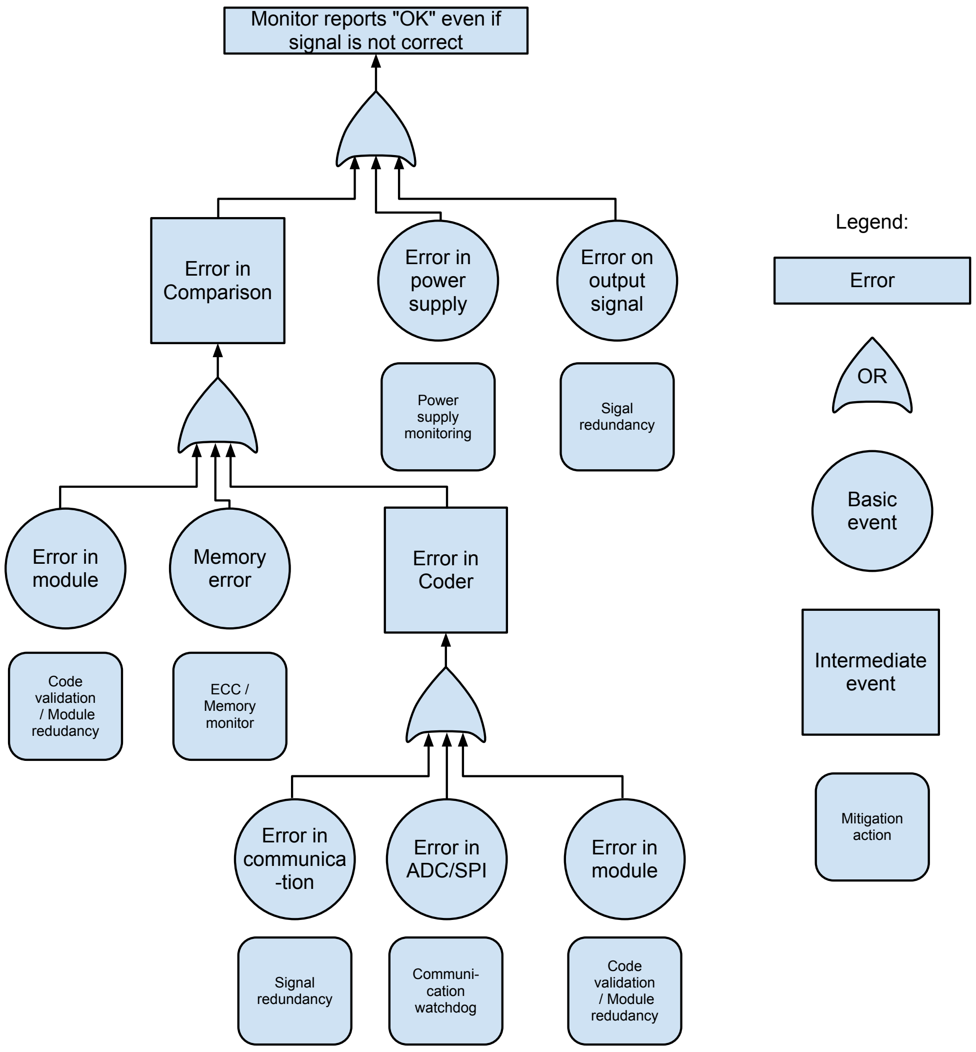


Figure 5.7: FTA of the communication monitor.

5.3.4 Software unit design

With the software architectural design decided upon, some software unit design was performed to simplify the development of each of the components by providing specifications on the functionality of each component. The following sections describe the design of the four most important components and the serial component used for debugging and verification.

Software unit design: Coder

The coder does four operations in the following order:

1. Calculates the difference between the current input (16-bit) and the previous input (16-bit).
2. Saves the sign bit and takes the absolute value of the difference from Step 1.
3. Finds the index of the most significant bit with value one in the highest 14 bits of the difference, returning 0 if no one is found. This corresponds to taking log_2 of the difference from Step 2 divided in half, and then rounding the result down to nearest integer, for differences ≥ 4.
4. If the original difference from Step 1 was negative, i.e. the sign bit saved in Step 2 is one, the value from Step 3 is negated.

Mathematically this could also be described as:

$$code = \begin{cases} \lfloor log_2((in[0] - in[-1])/2) \rfloor & \text{if } in[0] - in[-1] \geq 4 \\ \lfloor log_2((in[-1] - in[0])/2) \rfloor & \text{if } in[0] - in[-1] \leq -4 \\ 0 & \text{else} \end{cases}$$

where $in[0]$ = the most recent sample
and $in[-1]$ = the previous sample

This gives 29 codes for the syntactic pattern recognition, codes with higher granularity of difference at lower differences than higher, which should be good enough for the test cases used in this project.

Memory

The memory used in the design is a single port ROM containing 2^{12} 52-bit records divided into two values as of Table 5.3[6]. To simplify the implementation the dedicated block RAM-cells included on the FPGA was used to implement this memory.

A limitation introduced by the choice of using 12-bits for storing the length of an expected code is the length of each section being at most $2^{12} = 4096$ samples long, which corresponds to about one minute of samples when

[6]The last 32 bits of each memory record was originally intended to hold upper and lower limits of the range of acceptable voltage inputs, a functionality later omitted to provide a better code comparison functionality before extending the monitoring with more functionality.

Table 5.3: Communication monitor memory record description.

Bits	Type	Contains
52-45	Signed	Expected code.
44-33	Unsigned	Length of expected code.
32-1	-	Unused in this project.

sampling with the sample rate of the on-board ADC. As the complete test cases from the previous project being at most 10 seconds long, this is not a limitation relevant to this specific project. For a future implementation the number of bits used to store the length of a sequence could be extended to provide support for longer piece lengths or higher sampling rates.

Comparison

The comparison component has one main function, to evaluate the current input code and report if an error is detected, either by too many consecutive erroneous codes from the coder or if the number of codes in a single sequence is too few or too many.

To implement this functionality the comparison component has two functions, the second occasionally called from the first. The first function is called with each new input sample, and compares the new code with the expected code, as read from the memory. If these codes are the same, a sequence counter is incremented and the component is finished, the second function not being called at all. However, if the codes differ, another comparison is made, between the new code and the expected code to come next, also read from the memory. If these two codes are the same, a second sequence counter is incremented and the component is once again finished, but this time the second function is called afterward, as described in the next paragraph. Although, if this second comparison fails too, this results in an error counter being incremented, before exiting the function. If more than five subsequent wrong codes are detected, that is if the error counter has a value larger than 5, the component outputs the error code “6” and stays in this state until the system is reset.

The reason for choosing more than five subsequent wrong codes before reporting an error in this specific implementation is as follows: 6 samples as 60 Hz, the sampling speed of the on-board ADC, equals 100 ms, which

Table 5.4: Error codes output from the comparison component and complete system.

Error code	Meaning
6	More than five subsequent wrong codes detected.
12	Too many subsequent codes in an odd sequence (sequence number $2n + 1$).
13	Too few subsequent codes in an odd sequence (sequence number $2n + 1$).
14	Too many subsequent codes in an even sequence (sequence number $2n$).
15	Too few subsequent codes in an sequence (sequence number $2n$).

complies with the requirements on a previous tested steer-by-wire-system[7]. However, as a result of this choice a limitation is introduced in the system, making all the pieces of the piecewise-linear signal being required to be at least 100ms long to be able to work correctly, with 200ms being the recommended minimum length.

The second function of the comparison component is executed in the cases where the second counter, counting the amount of expected codes from the subsequent sequence, is high enough[8], at which the component assumes the input signal has moved onto the subsequent code/sequence. The function then compares the first counter with the expected length of the sequence, read from the memory, making the component output an error code and go into the error state if the previous sequence of codes has been too short or too long.

All error codes output from this component can be found in Table 5.4. These codes comply with the decision in Section 5.3.3 to provide redundant digital output signals for the comparison status output, as all errors are reported with at least two of the three most significant bits sets to one, providing basic protection from type II errors stemming from single-bit upset errors.

[7]The requirement specification for this system is unfortunately not available to the public.

[8]In this project this limit was set to five consecutive samples with the same code, as one or two samples could just be disturbances, but five codes should be more reliable.

ADC communication/SPI master

The SPI Master, as seen in Figure 5.6, is not an actual part of the monitoring, but it is still a necessary part of the design as component provides the system with support for communications to the on board ADC. This component is hence a standard SPI Master, with special support for the address-data communication employed by the ADC, a MAX1409, which includes support for 8-bit+8-bit write sequences and two read sequences of 8-bit+8-bit and 8-bit+16-bit respectively, where the first 8-bits always correspond to the intent to read or write and to which address the operation should be performed on and the following bits correspond to the data either read or written by the master.

RS232 UART

A RS232 UART was added to the design during the software integration phase, and is mainly used for debugging the monitor functionality and adding better reporting functionality used in the verification process. As the UART therefore does not affect the actual monitoring capability of the communication monitor in any way, and therefore is easily removed from the rest of the system, it was not added to the system design found in Figure 5.6. This results in the component not being decisively safety critical in its current implementation. Therefore to reduce the development effort of this component, an already finished RS232 UART available from OpenCores was used as basis for the component[9]. However, some adjustments was made to the component before the integration, such as improving the serial communication sampling and changing some indication signals to only remain high for one clock cycle, to increase the compatibility with the rest of the system.

5.4 Unit implementation and software integration

After the design of the communication monitor was finalized, four main activities remained, namely implementing each unit of the software design into synthesizable HDL code, integrating the software units into the complete software design system, syntesizing the HDL code and employing it on the FPGA board and the final demonstration and verification. Each of the first

[9]Project rs232_interface, available at:
`http://opencores.com/project,rs232_interface`.

two activities would also be followed by some kind of verification of the functionality at the development level. While the following activities correspond to some of the clauses in [10, Part 6, Clause 8 to Clause 10] in name, the corresponding clauses in the standard mostly cover verification activities, which by the project definition were outside the scope of the project. The following sections does therefore not include any references to the standard, as no specifications were applicable to the following activities performed in this project.

Unit implementation and testing

The unit implementation activity mainly consisted of implementing the software components in VHDL. The development of each unit was accompanied with development of a test bench for each unit. With the test benches it was easily verified that each unit worked as intended before going forward to the next activity, software integration.

One exception to the implementation by hand process was the creation of the memory used by the comparison component to receive expected codes and the length of each code sequence. The memory was instead created by a block memory generator tool, provided with the main development tool, resulting in the dedicated block RAM-cells available on the FPGA being used in an efficient and correct manner.

Software integration and testing

The software integration foremost consisted of the coding of the two top VHDL-modules for the project, and instantiating the software units from the previous activity as modules in the top VHDL files, resulting in the following file and module hierarchy:

- Communication Monitor Main - main.vhd
    - ADC SPI Master - adc_spi.vhd
    - RS232 UART - uart.vhd
    - Syntactic Pattern Recognition - syntactic.vhd
        - Syntactic Coder - coder.vhd
        - Syntactic Comparison - comparison.vhd
        - Memory - blk_mem_gen_v _1_0.xci (Xilinx IP-core)

As with the unit implementation and testing, the development of the top modules was accompanied with the development of a test bench for each of the two modules. The test bench for the syntactic pattern recognition module verified that the complete syntactic pattern recognition was working as intended, while the test bench for the complete communication monitor was verifying that the inputs and outputs of the system were working correctly to specification.

As the syntactic pattern recognition as noted earlier being a simple pattern recognition technique, the encoding and comparison of the input signal could be performed in less than two clock cycles. This results in that the pattern recognition part of the system being theoretically able to handle samples at half the internal clock speed, which with the FPGA used in the project gives a theoretical limit of $100/2 = 50$ MHz.

5.5 Item integration and testing

With the design implemented and simulations showing the system working as intended, the remaining activities in the implementation phase of the project were the integration of the hardware and software and the subsequent verification. The aim of the verification was to ensure that the implementation worked correctly as its specifications, on the development kit. Because the system was implemented to work on an FPGA, the integration of hardware and software steps were very simple and as follows:

1. Set what pins of the FPGA corresponds to what signal in the top VHDL module.
2. Syntesizing the VHDL files into gate-level netlists.
3. Implementing the netlists into a placed and routed FPGA design.
4. Creating a .BIT-file of the implemented design, which can be used for programming the FPGA.

Even though steps 2 to 4 are performed by different tools, they can all be started from inside the development tool, which automatically calls the tools and adjusts their settings to correspond to project settings, such as target FPGA (mainly for the implementation). The only exception is the first step, which requires a manually created User Constraints File (UCF), specifying which signals in the HDL design correspond to the input and outputs pins on the FPGA-chip. But with the development kit documentation including a complete UCF for the board in this project, this step consisted of editing the available UCF by changing all signal references in the file to the signal names used in the top VHDL module, main.vhd, and removing the references in the file to pins not used in the project. With all steps described above finished the last activity of the project was to program the FPGA with the design and perform the verification of the design.

The communication monitor was tested with one test cases for the steer-by-wire system from the previous project, and one new test case for testing the extremes of the new communication monitor. The first test case, the test case for the steer-by-wire system, henceforth called Test Case X (TCX), was used as it was with only minor adjustments, such as adding an indication of the running test being finished. The test case was then output by an AWG. The shape of the signal based on the test case description and the ADC input samples can both be seen in Figure 5.8. As seen in the figure the shape of the received ADC samples corresponded very well to the shape of the test case specification, and using the measured relationship[10] $In_{Sample} = 11000 * V_{AWG}$ the levels of both signals also corresponded very well. Hence the setup was verified to work as intended and could be used for the verification activities.

After verifying that the previous test case gave the intended result, as in Table 5.5, the previous verification system was thereafter used to verify that the communication monitor would work with an external verification engine. This gave the same result as running the test case on the AWG, proving the possibility of running the communication monitor with an external verification engine as intended.

The test case was thereafter modified in three different ways, one for each type of signal error, to make sure all types of errors were detectable, each test case afterwards being run on the AWG once more. In the first and second modification, TCX2 and TCX3, the third section, corresponding to the middle flat piece in Figure 5.8, was either made longer or shorter, to test the communication monitor's ability to detect the error in the signal of a piece of the signal being too long or too short. In the third and final modification of the test case, TCX4, the test case was modified to after 400 ms of outputting the second section, the negative slope in Figure 5.8, change the slope to a more steep slope. This was used to to verify the communication monitor's ability to detect erroneous shapes in the signal. The result of these three tests can be seen in Table 5.5, with the resulting error codes corresponding to the codes in Table 5.4, and "-" corresponding to the communication monitor having stopped and returning an error signal.

[10] In theory the actual conversion constant should be closer to $Sample_{ADC,Max}/V_{AWG,Max} = 65535/5 \approx 13100$, but because of nonlinearities in the ADC and other factors the constant was measured to a lower value.

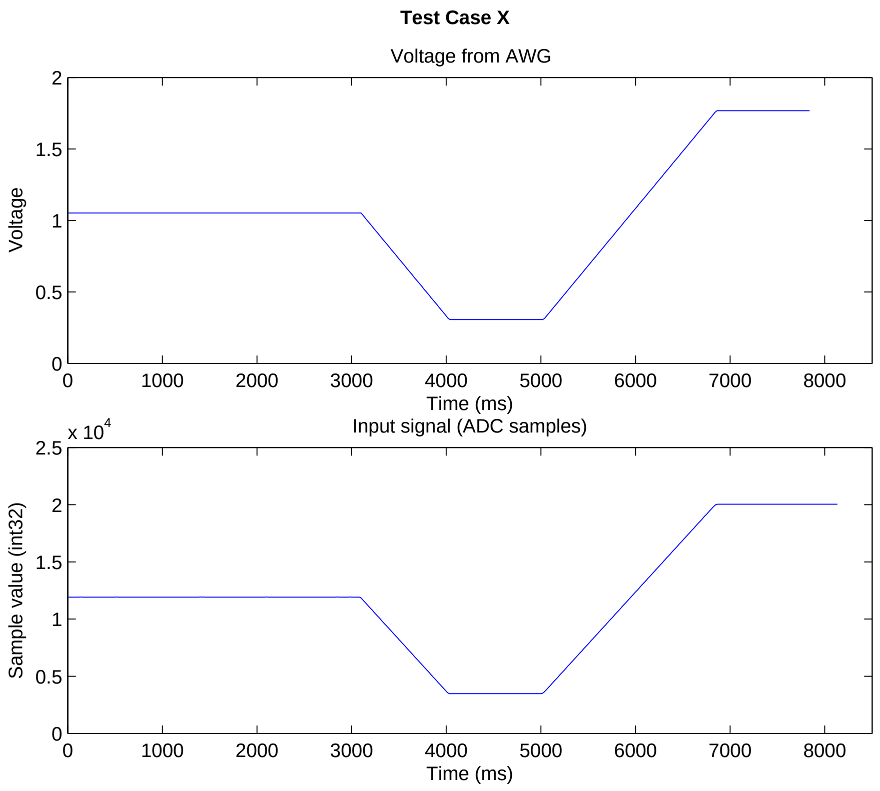


Figure 5.8: ADC input samples and expected signal (based on verification system test case) for Test Case X.

Confirming that the verification system worked as intended with a test case from the previous project, another test case was prepared, called Test Case Y (TCY), as seen in Figure 5.9. This test case, which had no relation to any real use case, was constructed to test the limits of the verification system. An example of a special case verified was that the monitor could handle signals where two neighboring pieces of the signal had similar derivatives, but still with enough difference to result in different codes at the signal syntactic encoding. Another section verified the system worked with pieces as short as 200 ms, as of the limitation introduced in Section 5.3.4.

The results of this test can be found in Table 5.6, showing only deviations of a single sample in section length and thus following the signal well enough even with some extremes in length and derivatives.

A careful reader may have noticed that one earlier specification is not verified, that is that maximum length of a test case and also the maximum length of a piece of the piecewise-linear signal introduced in Section 5.3.4. The reason for this is mainly a limitation in the AWG available for use in

Table 5.5: The results of TCX and the modified test cases based on TCX.

		Sequence Lengths (samples)				
Section	Description	Expected	TCX	TCX2	TCX3	TCX4
1	Flat	180	180	175	179	181
2	Negative slope	56	54	54	55	24
3	Flat	60	57	51	66	-
4	Positive slope	110	108	-	-	-
5	Flat	60	60	-	-	-
	Result:	-	OK	Error 13	Error 12	Error 6

Table 5.6: The results of TCY.

		Sequence Lengths (samples)	
Section	Description	Expected	TCY
1	Flat	54	50
2	Negative slope	48	47
3	Negative slope	30	29
4	Positive slope	120	119
5	Large positive slope	12	12
6	Negative slope	42	42
7	Negative slope	66	65
8	Large positive slope	18	17
	Result:	-	OK

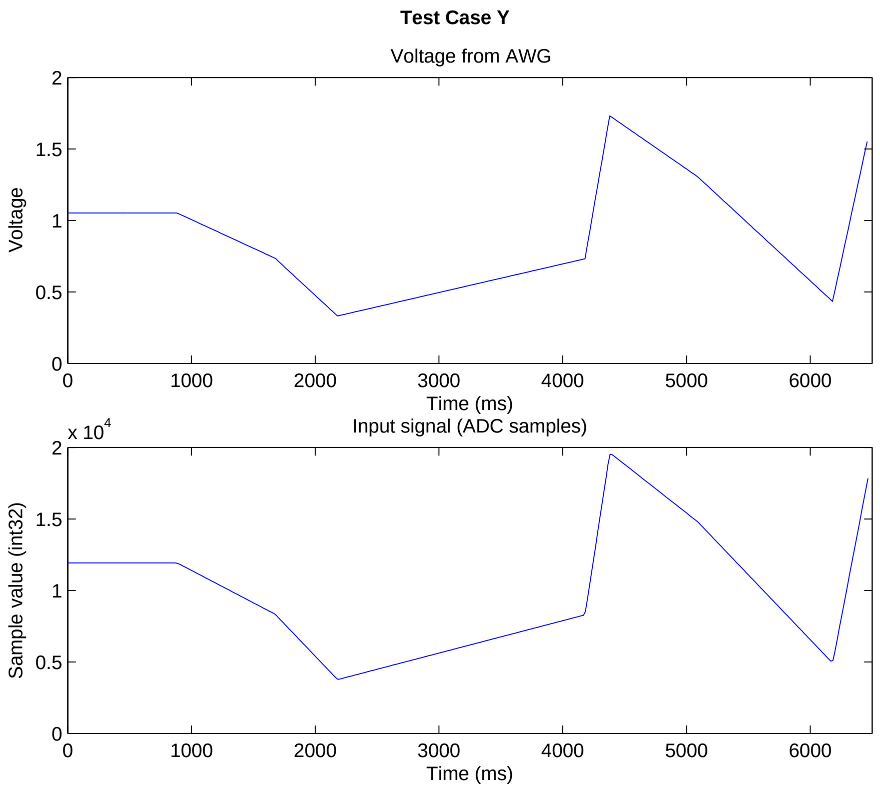


Figure 5.9: ADC input samples and expected signal for Test Case Y.

the project. The AWG was only able to output signals less than 10 seconds in length, and with the longest possible piece handled by the communication monitor being a little more than an minute long, verification of this limit was simply impossible with the available AWG.

An interesting point from the verification can be seen in Table 5.5 and to some degree also in Table 5.6. All measured section lengths are overall shorter than the expected sequence lengths. This was assumed to stem from the fact that the output from the AWG is not perfect when changing between one section of the signal to another, giving one “erroneous code” at the change between two sections. This assumption was verified correct by saving all samples from the ADC during testing and doing offline encoding. This resulted in all erronous codes being found in the transitions between one section to another. This highlights the importance of having some kind of support of error tolerance in the monitoring, as most test cases would have failed otherwise, even though they are following the test case specification very well.

5.6 Results and discussion

The main result of the implementation phase is a working and verified communication monitor. The final report of the implementation performed by the development tool additionally reported only 1.0% of the lookup tables (LUTs) used for creating the gates of the final design being used, which means that the FPGA chip still has much resources left for adding more functionality or improving the safety, for example through Triple Modular Redundancy (TMR). Additionally the generated memory used in the design used only 3% of the dedicated 16-bit RAM blocks on the FPGA chip, and with the implemented memory having a read depth of 4096 providing support for test cases from ∼13 minutes long to ∼75 hours depending on piece length, the memory available should also support further extensions to the design.

Concerning the HWSRs from Table 4.5 the most relevant requirements are satisfied by the implementation. The two requirements HWSR081 and HWSR082 are satisfied by the hardware design and HWSR096 was also satisfied by performing the testing in an indoors environment complying with the operation conditions from Avnet. HWSR090 and HWSR091 were partially verified by the test cases used in the final testing, with no disturbances or unintended signals being observed, either before, during or after the tests. On the other hand HWSR093 could not be verified, as the environment for the verification engine is not yet known.

Similarly, looking at the SWSRs from Table 4.6, SWSR095, SWSR083, SWSR084 and SWSR085 are all satisfied by the software design and verified in the final testing, while SWSR086, SWSR087, SWSR088, SWSR098 and SWSR092 all lie outside the scope of the communication monitor implemented in this specific project. Thus it can be concluded that the communication monitor works as intended.

While the FTA, found in Section 5.3.3, provided many possible safety measures to implement in the remaining resources, only some was implemented as to reach the goal of a functional monitor in the time assigned to the projec. The FTA also results in showing how using a monitor independent to the main system is a good idea. Looking at the FTA it is easy to see as the independent monitor approach, with a monitor architecture less complicated than the complete verification engine architecture, put a limit on how far back an error in the system can be traced, lessening the effort needed to perform a good FTA.

Chapter 6

Project results and future work

This project has two final deliverables and results. The first deliverable is a basic safety case for a verification system, including a safety requirement specification and system design for the complete system and a software and hardware design for an element of the system, namely a communication monitor. The other deliverable is an implementation of a communication monitor and its test results, demonstrating the implementation of a specific safety functionality.

6.1 Safety case

The safety case of an ISO 26262 item is the compiled information concerning the safety aspects of the item, such as the various requirements, SGs, FSRs, TSRs, HWSRs and SWSRs, and the various design documents. The complete result of the first and second phase of the project is therefore the project's safety case.

The safety case includes the following information:

- Safety Plan.
- Item Definition, including 27 main functions of the tool, 4 used in the following HazOP study.
- The results from the HazOP study, including 60 identified hazards of the system.
- Safety Goals, 28 Safety Goals elicited during the HazOP study.
- Functional Safety Concept, including 32 FSRs.
- System Design.

- Technical Safety Concept, including 20 TSRs.
- Hardware Safety Requirements Specification, including 6 HWSRs, all elicited from TSR080.
- Software Safety Requirements Specification, including 9 SWSRs, all elicited from TSR080.

The progression of functions, hazards and safety requirements of the first two phases of the project can be found in Figure 6.1. Included in the figure is also the two "project definitions", the first where four of the main functions of the tool was chosen for the HazOP and the second where one TSR was chosen for the Software and Hardware Safety Requirement Specifications, and later implementation.

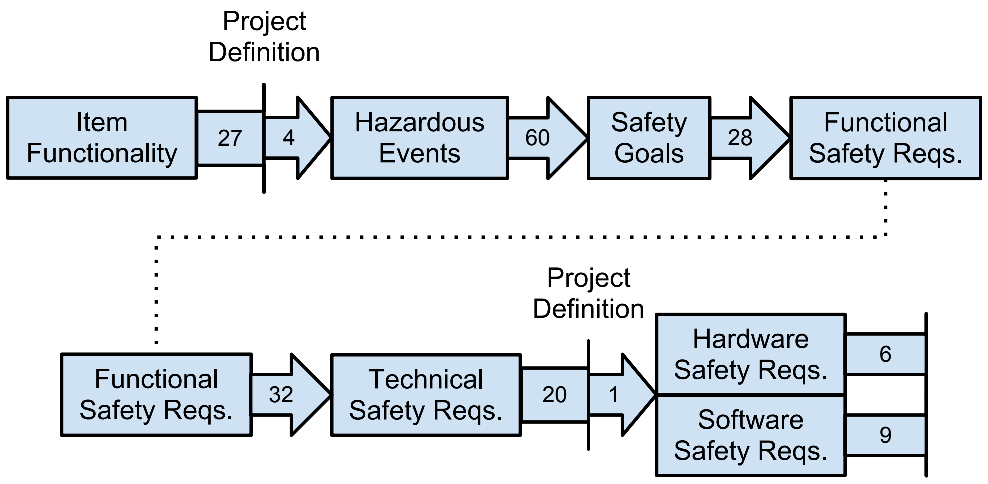


Figure 6.1: Progression of functions, hazards and safety requirements in the first phases of the project. The number of functions, hazards and safety requirements from the project included.

6.2 Reflections on working with ISO 26262

One of the positive aspects of working with ISO 26262 is the well-defined structure of the standard, based around the V-model as seen in Figure 1.1. This structure does for example make it easier to discern what activity follows after which and so on. However, some difficulties with following the structure can appear sometimes, primarily when clauses included in the V-model, i.e. Part 3 to Part 7, reference a supporting clause from Part 8 or Part 9, disrupting the otherwise quite linear flow of information. Sometimes this supporting clause references another supporting clause in itself, which disrupts the flow even more.

The standard also leaves a lot of room for interpretation, as long as you can motivate your interpretation, which is both good and bad. The good aspect is that adjustments can be made to the process as long as the adjustment is justifiable, for example if a clause is not applicable to a specific product. As for somebody who does not have had much experience in safety standards before, it could be hard to know what specifications that are most important to follow in their current form, and which specifications that are easy to fulfill, following standard development principles.

One general issue with ISO 26262 is that it sometimes breaks its own rules, such as the TSRs not being assigned to an element until the safety design activity, while the specifications on requirement management clearly state that all safety requirements must be assigned to an item or element. While being a minor issue it may cause confusion in a project and makes the planning of the project important, as one does not get stuck trying to adjust earlier activities to produce results following naturally from later activities.

Concerning the application of the standard in this project, the largest challenge was of course trying to apply the standard to a verification tool, with some of the problems encountered described earlier in the report. The adjustments to the standard performed in the project ranged from translating specifications regarding the item on a “vehicle level” to specifications regarding the item on a “verification system level”, to the omission of the parameter classification in the HARA.

Another experience from the project was working with an FPGA as platform in Part 5 and Part 6 of ISO 26262. As an FPGA implementation should be considered more deterministic and reliable than software running on a general purpose processor, an FPGA should consequently be a suitable platform for a safety critical element. The standard mentions FPGAs in [10, Part 4, 7.4.5.2], and notes that FPGAs should be considered as both hardware and software. Subsequently development on an FPGA platform should follow clauses from both Part 5 and Part 6 of the standard. But looking at the standard more closely it is clear that without further clarification it is difficult to know what clauses from Part 5 and Part 6 are applicable to an FPGA. As an example, when looking more closely at the requirement elicitation, which requirements should be assigned to what aspect of an FPGA? Is timing constraint a hardware requirement or a software requirement? While the “software implementation” employed on the FPGA decides the timings of the system, the constraints are actually found in the synthesized implementation, which simply is a gate netlist, which could be considered hardware.

6.3 Future work

One of the results of this project was the conclusion that the recommended risk assessment method of classification of each hazardous event with three parameters, severity, exposure and controllability, is hard or impossible to apply to software tools. Finding an alternative risk assessment method for software tools, perhaps from another standard, would therefore be an interesting field of research, as a more suitable risk assessment method possibly could lower the ASIL of some SGs.

Another interesting area that could need more work, which does not follow from this project, is to try to apply other functional safety standards to the development of software tools used in development of safety critical elements. ISO 26262 does for example also mention standards such as IEC 61508 as a standard to follow when developing a software tool according to a safety standard.

A Xilinx FPGA was used in this project for the implementation phase, and while FPGAs should present a good choice of platform for safety critical elements, they are also hard to classify into hardware or software. The standard [10, Part 4, 7.4.5.2] therefore states that a safety critical element implemented on an FPGA should be developed according to "an adequate development process, combining requirements from ISO 26262-5 and ISO 26262-6.", with no further explanation of which requirements that are applicable to an FPGA and which ones are not. Some further research on what clauses from Part 5 and Part 6 should be applied to development of a safety critical element on an FPGA platform would therefore be beneficial before implementing the element.

A logical extension of this project would also be to continue the work on the communication monitor, for example by implementing further safety measures from the FTA into the actual monitor. One example of this would be to evaluate the suitability and use of TMR, as supported in the more expensive Xilinx FPGA-models, for one or more of the functional components of the proposed/implemented communication monitor design.

Chapter 7

References

[1] Morayo Adedjouma, Hubert Dubois, Kamel Maaziz, and François Terrier. A model-driven requirement engineering process compliant with automotive domain standards. In *Proceedings on the 3rd Workshop on Model-Driven Tool and Process Integration (MDTPI 2010)*, 2010.

[2] Avnet. Xilinx spartan-6 fpga lx75t development kit. `http://www.em.avnet.com/en-us/design/drc/Pages/Xilinx-Spartan-6-FPGA-LX75T-Development-Kit.aspx`, 2012. [Online; accessed 07-December-2012].

[3] Mirko Conrad, Patrick Munier, and Frank Rauch. Qualifying software tools according to ISO 26262. `http://www.mathworks.se/automotive/standards/iso-26262.html`, 2010.

[4] Torsten Dittel and Hans-Jörg Aryus. How to "Survive" a safety case according to ISO 26262. In Erwin Schoitsch, editor, *Proceedings of the 29th international conference on Computer safety, reliability, and security (SAFECOMP'10)*, volume 6351 of *Lecture Notes in Computer Science*, pages 97–111. Springer Berlin / Heidelberg, 2010.

[5] dSPACE. dSPACE - TargetLink. `http://www.dspace.com/en/inc/home/products/sw/pcgs/targetli.cfm`, 2012. [Online; accessed 18-June-2012].

[6] Hubert Dubois, Marie-Agnès Peraldi-Frati, and Fadoi Lakhal. A model for requirements traceability in a heterogeneous model-based design process: Application to automotive embedded systems. In *Proceedings of the 15th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 2010)*, 2010.

[7] Petter Gradin and Victor Ortman. Development of a collision avoidance truck system from a functional safety perspective. Master's thesis, Linköping University, 2011.

[8] National Instruments. NI CompactRIO - rugged, high-performance reconfigurable control and monitoring system - National Instruments. `http://www.ni.com/compactrio/`, 2012. [Online; accessed 12-December-2012].

[9] ISO/IEC. IEC 61508:2010: Functional safety of electrical/electronic/programmable electronic safety-related systems. Published, International Electrotechnical Commission, Geneva, Switzerland., 2010.

[10] ISO/IEC. ISO 26262:2011: Road vehicles – functional safety. Published, International Organization for Standardization, Geneva, Switzerland., 2011.

[11] ISTQB. Certified tester foundation level syllabus. Technical report, International Software Testing Qualifications Board, 2011.

[12] ISTQB. Standard glossary of terms used in software testing. Technical report, International Software Testing Qualifications Board, 2012.

[13] Anil K. Jain, Robert P.W. Duin, and Jianchang Mao. Statistical pattern recognition: a review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1), jan 2010.

[14] MathWorks. Simulink - simulation and model-based design. `http://www.mathworks.com/products/simulink/`, 2012. [Online; accessed 26-November-2012].

[15] Joy Matthews. High level requirements gathering questions for new development projects | Pierson Requirements Group blog. `http://blog.piersonrequirementsgroup.com/general/high-level-requirements-gathering-questions-for-new-development-projects/`, 2009. [Online; accessed 12-December-2012].

[16] U.K. Ministry of Defence. Defence standard 00-58: Hazard and operability studies on systems containing programmable electronics, issue 2, 2000.

[17] Bashar Nuseibeh and Steve Easterbrook. Requirements engineering: A roadmap. In *Proceedings of the Conference on The Future of Software Engineering (ICSE 2000)*, 2000.

[18] Robert J. Schalkoff. *Pattern Recognition: Statistical, Structural and Neural Approaches*. Wiley, 1 edition, 1991.

[19] Purnendu Sinha. Architectural design and reliability analysis of a fail-operational brake-by-wire system from ISO 26262 perspectives. *Reliability Engineering & System Safety*, 96(10):1349–1359, oct 2011.

[20] Vector Software. VectorCAST for ISO 26262. `http://www.vectorcast.com/pdf/vectorcast-datasheet-iso26262.pdf`, 2012. [Online; accessed 2-October-2012].

[21] Patrik Sternudd. Unambiguous requirements in functional safety and iso 26262: dream or reality? Master's thesis, Uppsala University, 2011.

[22] Alexander Åström, Viacheslav Izosimov, and Ola Örsmark. Efficient software tool qualification for automotive safety-critical systems. In *Proceedings of the 15th International VDI Congress and Exhibition on Electronic Systems for Motor Vehicles (VDI 2011)*, 2011.

[23] Sparx Systems. Enterprise Architect - UML design tools and UML CASE tools for software development. `http://www.sparxsystems.com/products/ea/index.html`, 2012. [Online; accessed 12-December-2012].

Appendix A

Questionnaire creation process

1. Find and collect groups of questions that are usually included in a requirement elicitation questionnaire or similar process.

2. Find and collect actual questions that are usually included in a requirement elicitation questionnaire or similar process, for example from sources such as Pierson [15].

3. Look at which kind of information are specified in [10, Part 3, 5.4.1 & 5.4.2] to be included in the item definition.

4. Create a number of categories based on group of questions found in Step 1.

5. Mark the categories that have a counterpart in [10, Part 3, 5.4.1 & 5.4.2], and add new categories if needed so that all points in [10, Part 3, 5.4.1 & 5.4.2] have a category.

6. Add each of the questions to a specific category, based on what answers the question is most likely to get.

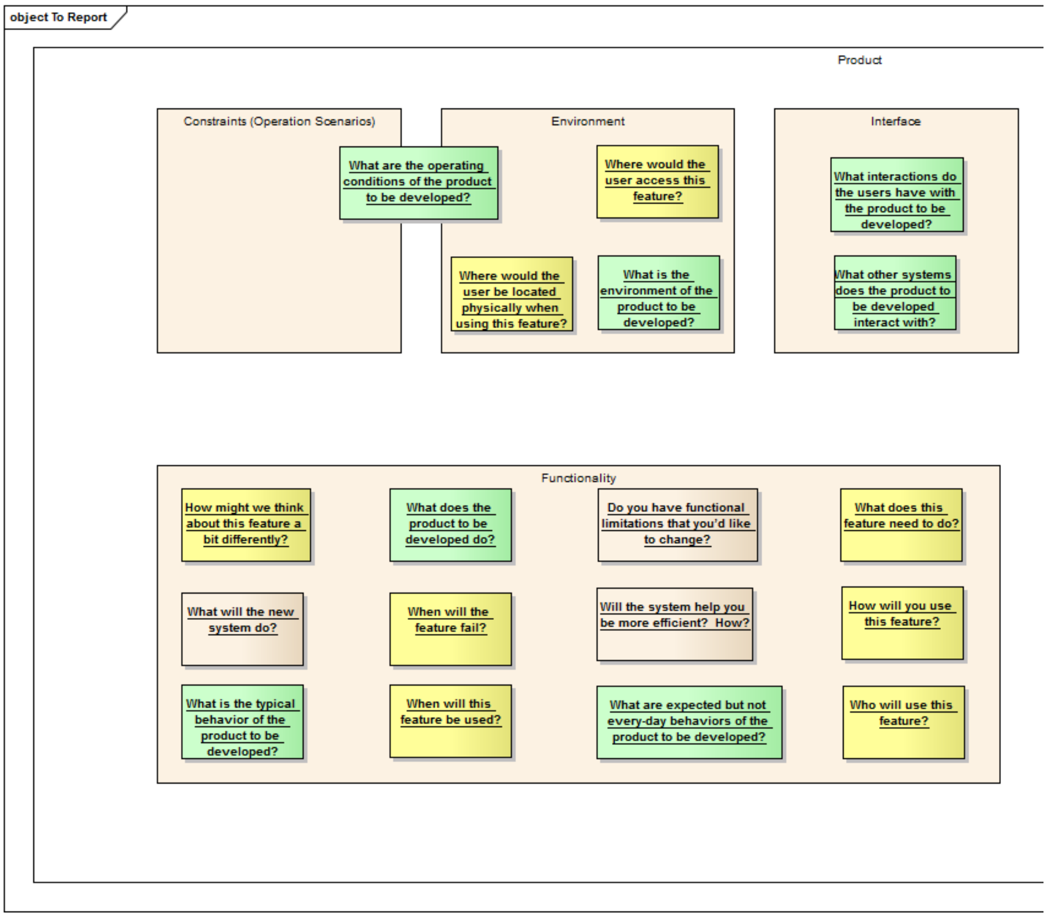
7. Add an initial set of questions to the questionnaire, one from each category that you marked in Step 4 or created in Step 5. Either a question from Step 2 or a composite of several questions from Step 2.

8. Try to figure out if the answers to the questions in the questionnaire should fulfill [10, Part 3, 5.4.1 & 5.4.2].

9. If there is a chance that the answers to the questions in the questionnaire would be sufficient or you are unsure, add more questions until it is probable that [10, Part 3, 5.4.1 & 5.4.2] is fulfilled.

10. Tidy up the questionnaire and try to get similar questions close by so that the recipient of the questionnaire better understands what answers are expected.

In this project the UML modeling tool Enterprise Architect from Sparx Systems [23] was used for the above process. A object diagram was used with questions modeled as objects, color coded based their source, and categories modeled as boundaries, filled in if their counterparts could be found in [10, Part 3, Clause 5]. An example of the result can be seen in Figure A.1. Note that this example only shows a part of the result of the process up to Step 6.



object To Report
Product
Constraints (Operation Scenarios)
Environment
Interface
What are the operating conditions of the product to be developed?
Where would the user access this feature?
What interactions do the users have with the product to be developed?
Where would the user be located physically when using this feature?
What is the environment of the product to be developed?
What other systems does the product to be developed interact with?
Functionality
How might we think about this feature a bit differently?
What does the product to be developed do?
Do you have functional limitations that you'd like to change?
What does this feature need to do?
What will the new system do?
When will the feature fail?
Will the system help you be more efficient? How?
How will you use this feature?
What is the typical behavior of the product to be developed?
When will this feature be used?
What are expected but not every-day behaviors of the product to be developed?
Who will use this feature?

Figure A.1: Example of the questionnaire construction process.

Appendix B

Project-specific Constrained Natural Language

A CNL is usually constrained in two ways, the first being limits on what words that are allowed, usually defined by a vocabulary and the second being various grammatical or structural restrictions.

This project used both types of constraints, a structural restriction based on work of Viacheslav Izosimov and a small vocabulary of "preferred words" (as a complete and exhaustive vocabulary would take too long to develop, even for a relatively small project such as this one).

The structural restriction used in the project was that requirement text of all requirements in the project had to adhere to the following structure:

- <Subject X> shall <Action Y> (applied to) <Object Z> [<Condition>]

Where <Subject X> and <Object Z> specifies an existing elements or parts of the system and <Action Y> should be an observable action for possible verification. A small exception to <Subject X> specifying an element or part of the system was granted for formulation of the Safety Goals (SGs), as the HARA was based on errors found in functionality of the tool. This exception meant that the <Subject X> of the SGs instead had to be errors or faults of the functions of the system.

The vocabulary used can be seen in Table B.1, with the preferred term in the first column, other non-preferred terms to be exchanged to the first term in the second column, and a description of the term in the last column.

Table B.1: Vocabulary of the preferred words used in the requirement text of the safety requirements in the project.

Preferred term	Non-preferred terms	Meaning
Verification system	Verification tool, System test tool, Test tool	The verification system to be developed in the project.
Use	Employ	Specification of a specific safety measure to be used.
Detect	Find	Have the ability to make difference between normal and non-normal operation as described in <Condition>.

Appendix C

ASIL determination table

Table C.1: Table used when determining ASIL, based on the three parameters Severity, Probability and Controllability. Adapted from [10, Part 3, Table 4].

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