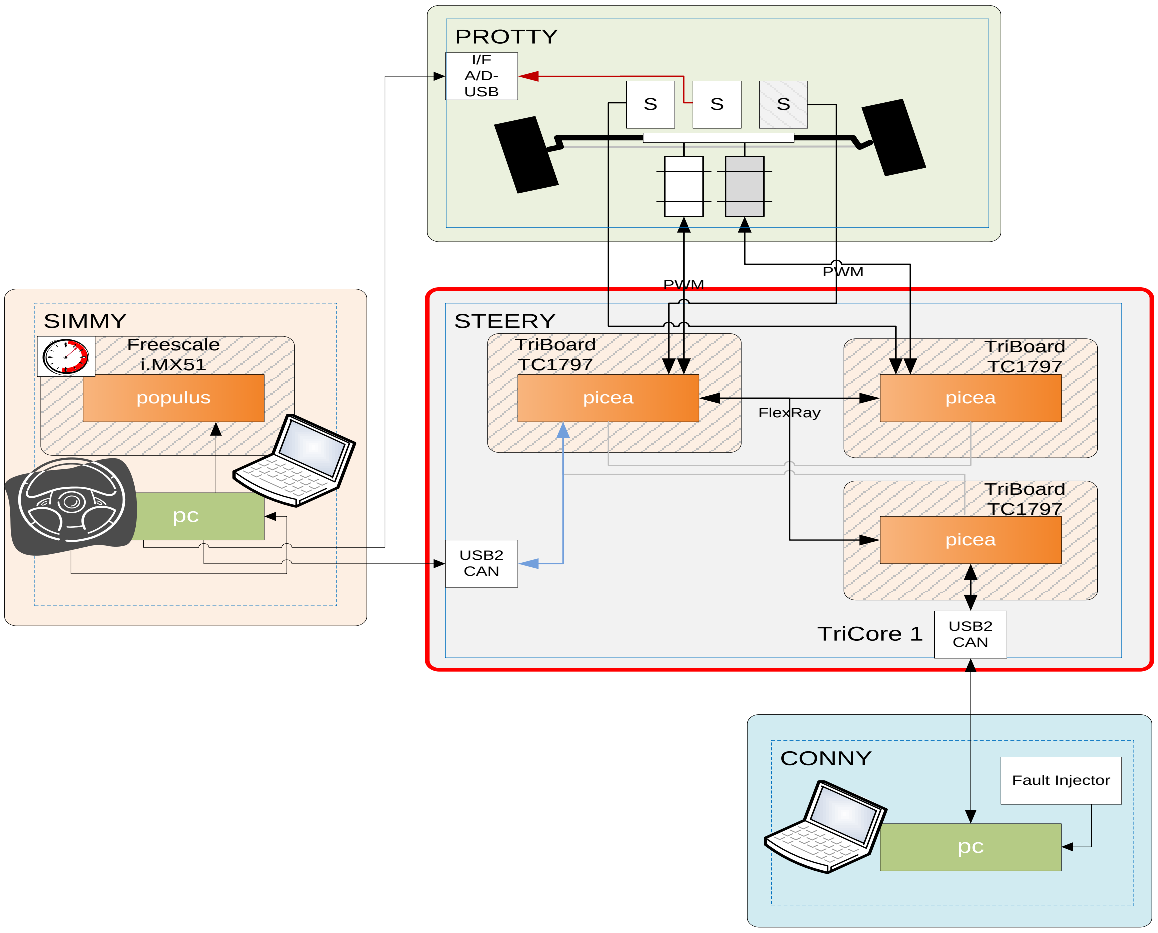
CHALMERS



Development of a Steer-By-Wire application in MATLAB/Simulink

Master of Science Thesis in Computer Science Engineering

Julien Loire

Ajay Kumar Reddy Neelakantam

Chalmers University of Technology
Department of Computer Science Engineering
Göteborg, Sweden, August 2011

Development of a Steer-By-Wire application in Matlab/Simulink

Julien Loire
Networks and Distributed Systems
Department of Computer Science and Engineering
Chalmers University of Technology

Ajay Kumar Reddy Neelakantam
Communication Engineering
Department of Signals and Systems
Chalmers University of Technology

Examiner: Lena Peterson

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Preface

The thesis proposal was given to us by Mecel AB. The purpose was to develop a Steer-By-Wire system using Simulink based on AUTOSAR and ISO 26262 standards in order to learn more about the concept and to prepare the company for early compliance. We are very grateful to Mecel for having us as a part of this project and for letting us to work with one of the most experienced teams. The development environment is a car simulation, which gives a realistic dimension to this project. At the same time it has a nice entertainment value to display during seminars and fairs. Once again we would like to express our gratitude to Mecel for supporting us throughout our work and having provided us with the necessary equipment to do a good thesis project.

In particular, we would like to thank Håkan Sivencrona, Johan Svärd, Pontus Henningsson, Erik Hesslow and Marcus Larsson at Mecel. They gave us great guidance throughout the project and were always enthusiastic to help us with any questions that we might have.

We are grateful to our examiner at Chalmers, Lena Peterson for her invaluable assistance during the entire project from project planning to feedback of the report.

Last but not least, we would like to thank our family members and friends who have been in great support in the time of need.

Abstract

This report deals with the idea of Functional System Safety (FUSS), which is a part of the DFEA2020 project. This DFEA2020 project is a joint effort between several actors of the automotive industry, including Mecel, to oversee car technologies by 2020. FUSS is building knowledge on the basis of AUTOSAR – an open automotive software architecture and ISO 26262 – an ISO safety related standard for passenger cars. The project aims to develop a steer-by-wire application for a car simulator. To achieve that, MATLAB/Simulink has been chosen by Mecel as the application development tool, in which we designed the steer-by-wire application. We used the state-of-the-art safety mechanisms from the latest AUTOSAR release for data flow protection within the steer-by-wire application. Model-based design allowed us to test the logic of our steer-by-wire application very early in the development process. A first comprehensive set of tests was done in the Simulink simulation environment with the help of recent Mathworks tools; a second test was done post – code generation when the developed software had been integrated to the AUTOSAR platform. This two level testing process proved to be a valuable development asset with respect to ISO 26262 because it allows for early error detection and correction. We documented the lessons learnt regarding the impact of ISO 26262 and AUTOSAR development on the Mecel development processes. Some limitations that we were able to point out include the fact that the ISO 26262 standard leaves room for interpretation. Companies cannot rely on standard metrics to evaluate the relevance of their processes with regard to ISO 26262. Moreover, the tools around ISO 26262, recently pushed to the market, lack maturity. Combined with the fact that the abstraction layer of AUTOSAR brings a lack of global control over the overall system, we felt that a significant effort is needed from the automotive companies in order to tailor their existing processes to be in line with ISO 26262 specifications.

List of abbreviations

AAT	AUTOSAR Authoring Tool
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basic SoftWare
CAN	Controller Area Network
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
DFEA	Dependable and Flexible Electrical Architectures
E/E	Electrics/Electronics functionalities or Components or Architecture
E2E	End to End
ECU	Electronic Control Unit
FSR	Functional Safety Requirements
FUSS	Functional System Safety
GUI	Graphical User Interface
IOCTL	Input / Output Control System calls
MCDC	Modified Condition/Decision Coverage
OEM	Original Equipment Manufacturer
PWM	Pulse Width Modulation
RTE	Run Time Environment
SBW	Steer-By-Wire
SDK	Software Development Kit
SWC	Software Component
TLC	Target Language Compiler
TSR	Technical Safety Requirements
USB	Universal Serial Bus
VFB	Virtual Function Bus

Table of Contents

List of Figures

1. Introduction

This chapter gives an overview of the project background, purpose and objectives along with its scope.

1.1 Background

To improve the driver's safety and comfort, car manufacturers are increasing the usage of electric and electronic (E/E) functionalities (or components) in the place of mechanical applications. Until recently, each manufacturer and automotive tool developer were using different standards and implementation procedures in designing these E/E architectures. This remained challenging to test and integrate different standards by one OEM. To overcome this problem, leading OEMs and Tier 1 suppliers formed a group called AUTOSAR, an open standard for automotive E/E architecture [1] that led the way to innovative electronic systems with safety and environmental friendliness.

From the past few years, one of the intensively developing fields at many OEMs has been Drive-By-Wire systems. Among Drive-By-Wire systems, Steer-By-Wire has been the most challenging integration for the automotive industry. Although this technology has been integrated in airplanes for more than a decade, integration on cars and trucks is just rising up. This is due to the fact that the automotive industry faces the need to deploy low cost E/E systems to remain competitive in their market segment.

Numerous benefits that come with Steer-By-Wire over classical mechanical steering systems include: improved space utilization near engine compartment, reduced manufacturing costs, better shock absorption in case of a crash and the possibility to implement driving assistance mechanisms.

On the other hand, undetected systematic faults and hardware failures can potentially cause significant harm to the driver and his environment. To override this, the ISO 26262 standard has been designed to guide actors of the automotive industry throughout their product's development processes in order to improve driver's safety and reach an appropriate level of functional safety by handling safety requirements in the early phase of development.

''The increased complexity of the electrical system is a consequence from the evolving functional growth of the vehicles'' [2]. The research project DFEA2020 funded by

VINNOVA addresses this issue from different perspectives. Mecel’s quick adaptation to AUTOSAR and ISO 26262 standard, made them to be a partner company for DFEA 2020 project. Volvo Car Corporation leads the project and cooperates with partner companies and academia. The partner companies are Mecel, Semcon EIS, and KnowIT Technology Management. Chalmers University of Technology and KTH Royal Institute of Technology are the academic partners in the DFEA2020 project [2]. As part of this project, this master thesis ‘Steer-By-Wire applications in Simulink’ was proposed by Mecel and we are working closely with that application using the guidelines of AUTOSAR and ISO 26262 safety concepts under the supervision of Håkan Sivencrona, Johan Svärd, Pontus Henningsson, Erik Hesslow and Marcus Larsson.

1.2 Purpose

The purpose of this thesis work is to investigate how the new standard for vehicle safety-related applications, ISO 26262 affects the development within Mecel. Evaluation on how Mecel’s development processes can best be adapted to MATLAB/Simulink for the integration, verification and validation of a distributed hardware and software platform. Suggestions will be made on how the MATLAB / Simulink platform can serve as a step to improve chances for early verification and simulation in accordance with ISO 26262.

1.3 Objective

The main goal of the thesis work is to develop a Steer-By-Wire system in accordance with the requirements of AUTOSAR and ISO 26262 standards. In order to achieve this, the following tasks should be performed:

- Designing the functionalities of a Steer-by-Wire system using Simulink
- Verification and Validation of the Steer-by-Wire application using Simulink Verification Tools
- Designing an End-to-End (E2E) protection wrapper for the SWCs as per AUTOSAR standard.
- Code generation and Integration on Mecel’s AUTOSAR product PICEA.
- Verification and Validation of the Steer-By-Wire application in the Virtual Function Bus (VFB)

- Documentation of conclusions and lessons regarding the impact of ISO 26262 on Mecel development methodology and process

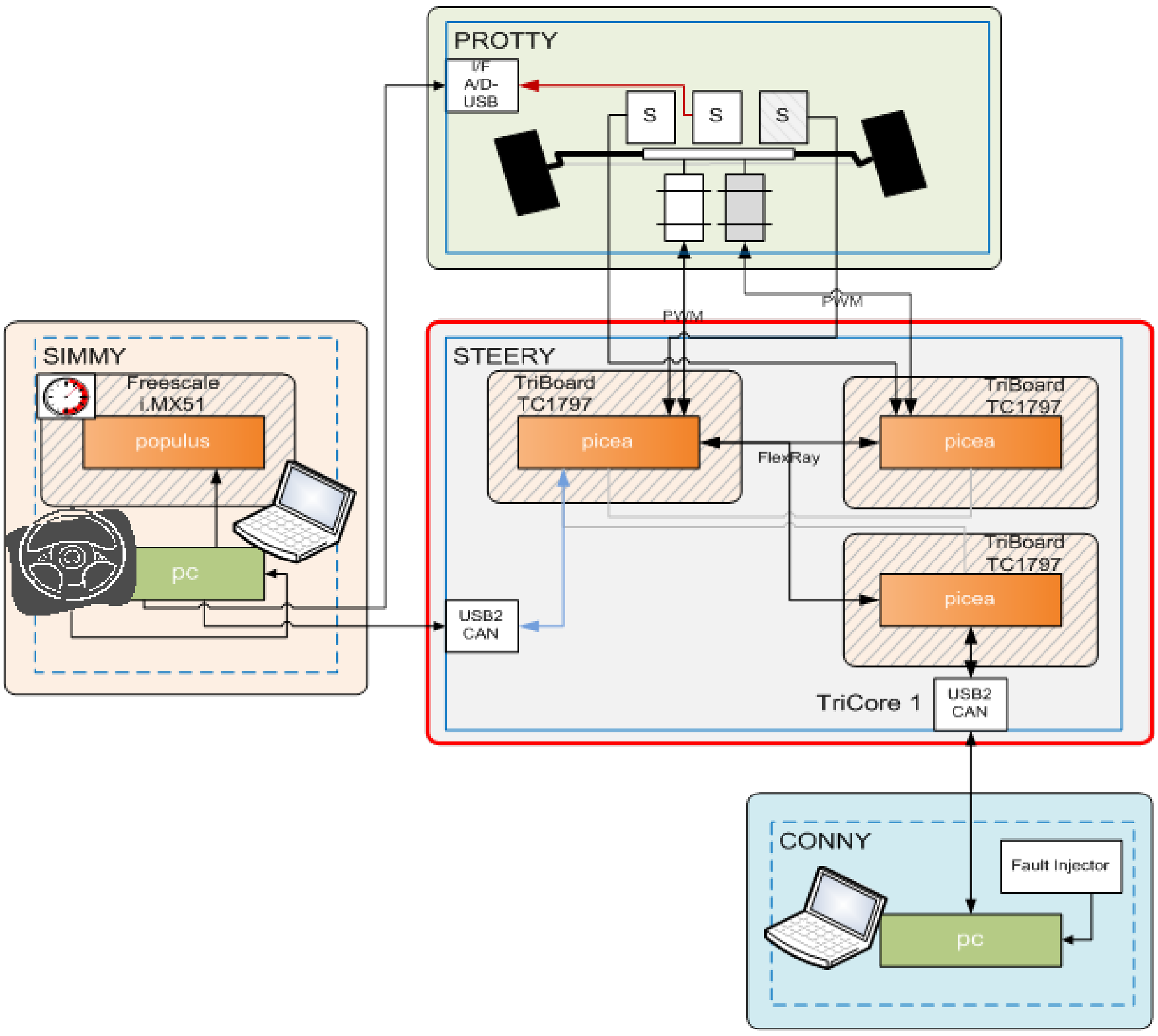


Figure 1: System Overview of the FUSS Project.
Source: Mecel AB internal docs
This master thesis will focus on the "Steery" subsystem that is the control subsystem. Other parts include:
-"Simmy", the simulation environment
-"Protty" that contains front wheels, actuators and sensors
-"Conny" that handles system monitoring and faults injection

1.4 Scope

Mecel’s SBW prototype consists of four subsystems that interact with the simulator – Live for Speed, to provide real sense of driving environment. The four subsystems are:

- “Simmy” – contains complete driving simulator including steering wheel, pedals and driver display.
- “Steery” – contains ECUs that manages the steering function
- “Conny” – provides fault injection and system monitoring
- “Protty” – contains the actuators to control the front wheel prototype (Fig.18) and also provides corresponding sensor information to Steery and Simmy.

The work within this thesis will focus on the “Steery” safety-related subsystem of the overall simulator. Fig.1 gives a clear picture of the scope.

2. Overview of the Standards

This project was developed in combination of two new automotive standards that will be used in commercial passenger cars in a very near future. A brief introduction of those standards will be seen in this section.

2.1 AUTOSAR

AUTOSAR is an open standardized automotive software architecture, developed in cooperation with different automobile manufacturers, suppliers and tool developers. It is a vital technology to cope with the growing electrical/electronic complexity in future vehicles. It aims to be prepared for the upcoming technologies and to improve cost-efficiency without making any compromise with respect to quality. One of the main purposes in implementing this standard was to “Cooperate on standards, compete on implementation” [3]. This jointly developed standard has led to four releases. Based on the most recent specifications (AUTOSAR Release 4.0) our application on steer-by-wire is developed.

2.1.1 Motivation

AUTOSAR’s strategy is to standardize the software architecture to encourage the exchangeability and reuse of software components. This solution effectively reduces development cost, makes the management of complexity of E/E systems smoother, reduces time to market and enhances the quality, flexibility, scalability and reliability of E/E systems. The exchangeability and reuse of software can be found on several levels, namely between supplier’s solutions, manufacturer’s applications and between vehicle platform. In the defined standardized software architecture, hardware and software parts are widely independent of each other as explained in the following section [4].

2.1.2 Key concepts

AUTOSAR defines four key concepts [4]:

- *Software components (SWC): A piece of software to be run in an AUTOSAR system*
- *Virtual Function Bus (VFB): High – level communication abstraction*
- *Run Time Environment (RTE): Implements the VFB on one ECU*

- *Basic Software (BSW):* Standard software for standard ECU functionality (OS, communication, memory, hardware drivers, diagnostics)

A SWC has no means of knowing where in the system it is located, i.e. in which ECU. The RTE is thus the only interface for the SWCs to communicate with each other. The RTE uses CAN, LIN and Flex-Ray buses for inter-ECU communication via the BSW as depicted in Figure 2.

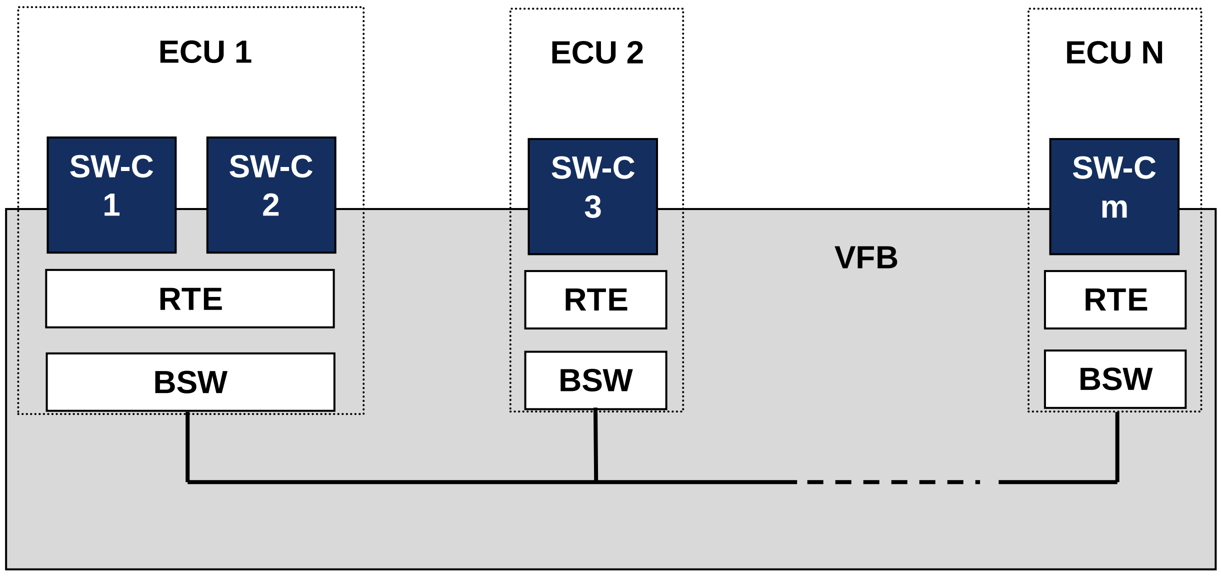


Figure 2: View of inter – ECU communication
Source: Mecel AB internal Docs
The SW-C 1 situated in ECU 1 communicates to SW-C 3 situated on ECU 2 through the RTE and BSW layer. There is no communication setup on SW-C level (as there would be in a client/server HTTP session). The communication is handled by RTE+BSW which form an abstract communication layer called VFB.

The implementations of the RTE and BSW modules are specified by AUTOSAR but they must be configured by the integrator. In the context of this master thesis, this has been the responsibility of Marcus Larsson, member of Mecel System team. This task includes for example the configuration of the CAN frames and channels, the flash memory blocks and the OS tick time of the ECU power states.

The SWC developer has some degree of configuration too, for example data types on input and output ports, communication signals, scheduling and inter-task communication, but these properties mainly focuses on the logic of the SWC. It is thus a radically different approach from a classical waterfall development process since the SWC developer does not care about how data travel outside the boundaries of the AUTOSAR interfaces and does not need to implement any hardware support or bus communication setup. The RTE will provide the necessary APIs. The source code for the SWCs can be implemented manually (C/C++) or can be automatically generated from a Simulink model, as is the case in this master thesis. Finally, to specify SWCs, connect them and integrate them with the BSW layer an AUTOSAR Authoring Tool (AAT) must be used. For this entire tool chain, one can choose between different vendors (Vector, Greensoft, dSpace, Mecel). In this project a product from Mecel, "Mecel Picea Suite" [5] was used.

2.2 ISO 26262

2.2.1 Motivation

Until recent times, all automotive industries were following the IEC 61508 standard, which was not fully adapted to the automotive E/E systems. In order to make a standard dedicated to automotive industry, ISO 26262 was created. This ISO 26262 is Functional Safety standard which came to presence with the title ‘‘Road vehicles – Functional safety’’. This standard is an adaption of IEC 61508 that provides an automotive Safety lifecycle, functional safety aspects of the entire development process, scaling of risk based approach using ASIL's (Automotive Safety Integrity Level) and measures to ensure a sufficient and acceptable level of safety [6]. Because of these incentives many automotive industries are adapting their development process to this new standard in order to be competitive in the market. As for ISO 9000, which relates to quality management systems, that is a must have certification required by OEMs nowadays, we foresee that ISO 26262 certification will open up an entirely new business opportunities for automotive tiers when it comes to safety – related development. Automotive tier suppliers therefore understand that an early adoption of this standard means a technological edge within the market.

2.2.2 Safety Process

To achieve an acceptable residual risk for each of the E/E vehicle functions regardless of the initial potential risk, ISO 26262 defines a safety process. This process addresses the prevention of systematic faults, the detection, control and handling of the inevitable remaining faults or failures. All the necessary activities to ensure the required functional safety of an E/E system are defined in the standard. The basic flow is as follows [7]:

- Identify the potential hazards and risks of a vehicle function
- Define acceptable risk goals and establish safety requirements
- Define and implement measures to avoid risks

- Verify that the safety goals are met and validate that they are satisfying an acceptable level of safety

The following figure gives an overview of the ISO 26262 safety process.

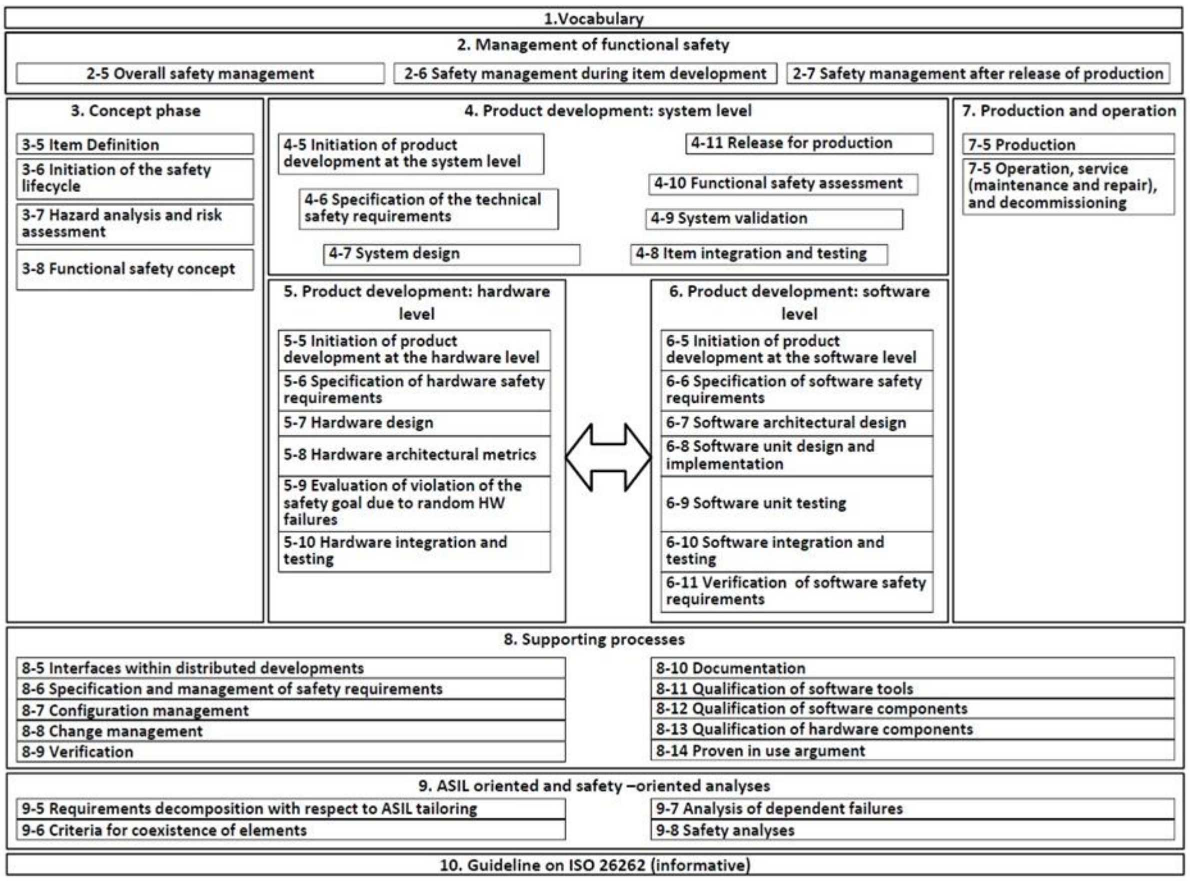

1.Vocabulary
2. Management of functional safety
2-5 Overall safety management
2-6 Safety management during item development
2-7 Safety management after release of production
3. Concept phase
3-5 Item Definition
3-6 Initiation of the safety lifecycle
3-7 Hazard analysis and risk assessment
3-8 Functional safety concept
4. Product development: system level
4-5 Initiation of product development at the system level
4-6 Specification of the technical safety requirements
4-7 System design
4-8 Item integration and testing
4-9 System validation
4-10 Functional safety assessment
4-11 Release for production
7. Production and operation
7-5 Production
7-5 Operation, service (maintenance and repair), and decommissioning
5. Product development: 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 Hardware architectural metrics
5-9 Evaluation of violation of the safety goal due to random HW failures
5-10 Hardware integration and testing
6. Product development: 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 Qualification 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 (informative)

Figure 3: Overview of ISO 26262 Safety Process, spreading over all the common phases of SW development [8]
In this master thesis our focus was on part 6: Product development: software level

A first impression one has when looking at this process is that it seems quite complex and heavy. We need to remember that ISO 26262 leaves room to tailoring of the company established processes. It does not aim at defining a whole new complex process with which companies are not familiar with, rather it more generally gives guidelines on what activities to perform and how to adapt the company processes to satisfy ISO 26262 Functional Safety requirements.

During the *concept phase*, the *item definition* (part 3 of Figure 3) is performed. An item represents a system, an array of systems or a function to which ISO 26262 is applied. Based on the particular item, a hazard analysis and risk assessment is done in which each hazard is classified and assigned an Automotive Safety Integrity Level (ASIL) depending on its severity [8]. This hazard analysis needs to treat the system as a whole, particularly the system within its environment and the worst case failures need to be identified [9]. Identifying all the possible failures of an E/E function under every single driving scenario and environment could take months and is therefore not an easy task to manage. The challenge here is thus to keep the analysis realistic without scarifying safety of the vehicle.

Based on the Hazard Analysis, safety goals are determined and each safety goal is assigned the same ASIL level as the corresponding hazard. Functional Safety Requirements (FSRs) are then derived from the safety goals inheriting the same ASIL level and allocated to different parts of the system based on a preliminary architectural assumption of the item [8].

During further development Technical Safety Requirements are specified based on the refinement of the hardware and software modules. After specification, implementation and integration on hardware and software level, hardware and software components are integrated step by step [8].

In the product development phase, TSRs help to describe how to implement the functional safety concept on a technical level. During this phase, the system gets more and more refined by partitioning into sub-systems structures. TSRs must be specified on each level of system and sub-system granularity [8].

Finally, the compliance of the developed system to ISO 26262 lies in the responsibility of the safety manager. ISO 26262 gives guidelines on what safety activities to conduct in order to achieve a reasonable residual risk. Proof and relevance of such activities must be defended to an external audit authority in order to fully comply with the standard.

We have seen that ISO 26262 leaves room to interpretation and tailoring of the company established processes. This is of course a good thing as it is not too invasive but on the other hand it opens up for misinterpretation. Consequently, the functional safety management team must make a real effort here to successfully perform the required activities and be able to motivate their choices when the audit is performed. If an X-by-wire system were to fail the consequences can be disastrous in real life. The impact on the society is thus great and the reputation of the automotive industry is at stake when it comes to such safety critical systems. The standard is a first step to safety – critical system development, processes and use of the tool chain (especially the AUTOSAR platform) must be refined according to it and maturity must be gained before we see such applications as X-by-wire being commercialized.

3. System Architecture and Design Decisions

This section briefly introduces basic system architecture of this project and also its data flow is explained.

Before starting this section, we recommend readers to have a look in APPENDIX D – Software and Hardware Specifications for a better understanding of the simulator setup.

3.1 Existing Design

Before this project was started, the system architecture of the simulator resembled the one shown in Figure 4. Here, the FUSS application will receive the steering wheel data, which is also called a steering angle, through DirectInput drivers and passes it to the Steery model through a CAN interface. Originally, both the FUSS application and the Steery Model were Microsoft VC++ project modules, which in other words, mean that they were hand – written C/C++ code. The output of the Steery Model is then passed to the Basic Stamp microcontroller, which converts the Wheel Angle Command into a timing signal, PWM. Based on this timing sequence the servo motor moves the front wheel position with respect to the steering wheel. As in Figure 4, the sensor (S1) detects the position of the front wheels and passes it to the Velleman Board, which converts Analog data to the discrete values. These discrete values represent the Wheel Position of the front wheels, which will be retrieved by FUSS application. Based on those values, by using DirectInput drivers in FUSS application force feedback was sent to the Steering Wheel. In order to change the Wheel Position in the game (*Live for Speed*), PPJoy- a virtual joystick is used by the FUSS application.

The main function here to be modeled into Simulink is the Steery Model, transforming a steering angle into a steering wheel command to the front wheel’s actuator. At this point of the project, after having explored the existing system and understood the different communication mechanisms, we decided to model Steery and explored the possibilities on how to connect the resulting model with the existing interfaces of the simulator. Our findings are presented in the following subsection and highlight the pros and cons of each method as well as justifying the retained design. It is to be noted that this modeling and integration was not intended to be kept as a final design but was done as an initial way to explore the tool chain, understanding the flow of the whole project and give us positive feedbacks on how to proceed with the final design.

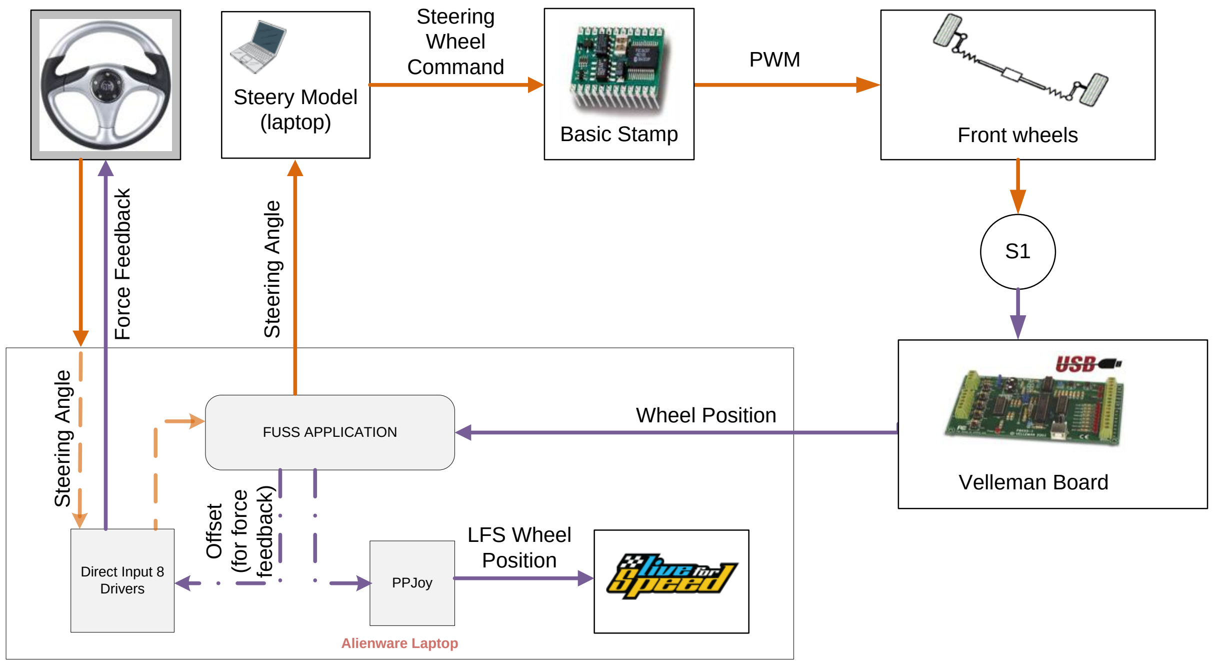


Figure 4: Simulator Architecture used in PC Environment in the early stage of the project.
Simulink model design was done for Steery model and integrated into the above architecture to have a first look of the system behavior. Later on it is updated to advanced Arch as like in Section 3.3

3.2 Interfacing the first Simulink model with the Simulator: Exploration of methods and Design Decisions.

We will see in more details in Section 4 how the exact implementation is done, but so far we can assume that it is rather straightforward to implement the transfer function in MATLAB/Simulink® to convert the steering angle received from the Logitech G25 Race wheel to a timing to control the servo motor in the front wheels. However, when dealing with the external environment of the model, namely receiving the steering angle through a CAN bus and commanding the Basic Stamp microcontroller through a serial communication, things get a little more complex. This section will describe some of the common methods used in MATLAB/Simulink© to integrate a model to its environment and present their pros and cons in relation to the thesis project.

xPC Target

xPC Target® is a Mathworks® product that enables users to execute Simulink models on a target computer for rapid control prototyping and Hardware-In-the-Loop (HIL) testing. It comes with a library of I/O device drivers, a real-time kernel, and an interface for real-time monitoring, parameter tuning, and data logging.

When searching for a solution to deploy our Simulink model to the simulator described in Section 3.1, xPC Target® appeared to us as a de-facto response to our needs. It would allow us to manage external communication with the simulator using the I/O device driver library and run our model in real-time on a target computer connected to the simulator. This solution is very flexible, effectively allowing to tune up the model (could it be changing a parameter or testing an algorithm over another) and directly check the performance in real time on the target computer.

However, the Kvaser© USB to CAN adapters are used to set up the CAN bus between the computer running the FUSS application and the computer running the Steery model. But Kvaser USB to CAN adapter is not supported by xPC Target. xPC Target currently supports only CAN-AC2-ISA and CAN-AC2-PCI boards from Softing® GmbH (Germany) [26]. The fact that PCI and ISA communication buses are not available on laptops used in this project was a first argument against using xPC target® for that matter. Moreover, one of the technical requirement states that the USB to CAN adapters should support multi-threaded applications, a characteristic that is not supported by Softing CAN cards. Consequently, even if xPC Target® appeared like a very good approach at first sight we decided to drop that option for lack of devices support.

If the project requirements do allow the use of xPC Target-friendly I/O device we strongly recommend using that concept since it is a fast, flexible and reliable mean test the behavior of a model. It is to be noted that it is a great tool for development of fast prototypes but is unsuitable for production code since it runs on a computer as opposed to an embedded system and that easiness of the concept does restrict the performance.

Integrating C code for the communication purpose

The Kvaser USB to CAN adapter is provided with a CANLIB SDK that allows us to create our own applications. Since xPC Target proved unsuitable for the thesis's project we anticipated the possibility to integrate C/C++ code from the kvaser CANLIB SDK to manage the CAN communication within Simulink. Some of the common methods to do so are described in more details in APPENDIX A. To put it in a nutshell, the handbook methods of associating the CANLIB files with Simulink were proven too much of manual work and deep knowledge of the core Matlab programming is required, which is not the focus of our thesis work.

We later discovered that MathWorks' Vehicle Network Toolbox fully supports Kvaser Controller Area Network (CAN) hardware interfaces, enabling direct connection between MATLAB or Simulink and the CAN bus [17]. However this tool-box is not included in the licensed version of Matlab/Simulink that is assigned to our thesis project.

Simulink Coder and manual tweaking

Simulink Coder generates C/C++ code from Simulink and Stateflow models. This generated code can be used for real-time and non-real-time applications including simulation acceleration, rapid prototyping, and hardware-in-the-loop testing. **This is the solution we retained** as we could generate C/C++ code for the model presented in Section 4.1.2 and tweaked that code so that it can be integrated in the Steery Model application developed during the previous Master Thesis on the simulator. The Steery Model application already implemented the methods to manage the CAN bus from Kvaser's CANLIB SDK and the serial communication to the Basic Stamp microcontroller. More details on the generated code and the tweaking are in Section 4.1.3.

Embedded coder

Embedded Coder™ generates readable and compact C/C++ code to use on embedded processors, on-target rapid prototyping boards, and microprocessors used in mass production. Embedded Coder enables additional Simulink Coder™ (Real-Time-Workshop) configuration options and advanced optimizations for fine-grain control of the generated code's functions, files, and data. These optimizations improve code efficiency and facilitate integration with legacy code, data types, and calibration parameters used in production [26].

Furthermore, Embedded Coder offers built-in support for AUTOSAR software platform and will be used in generating the codes for Steery application with Tricore hardware, where we split the model into two Software Components (SWCs) or ECUs.

3.3 Improved Design

In the improved design of system architecture as shown in Figure 5, most of the data flow remains identical as before (Figure 4). The changes are explained below. Instead of using a laptop for the execution of the Steery application, the Tricore Hardware is used. In addition, previously, the force feedback was calculated in the FUSS application; instead, the formerly designed Simulink model is extended to calculate the offset, which is used for force feedback. The input for this force feedback was obtained from sensor S1, and the output of the force feedback is given to DirectInput via FUSS application. Another sensor S2 is used to update the wheel position in the game; *Live for speed* through PPJoy. The reason for an extra sensor in this design was that we wanted to be able to simulate errors in the system where the actual position of the front wheel are not aligned with the steering wheel command thus the position of the front wheel in the car racing simulation game *Live for speed*. Overall, in this format, the Steery application was designed using standard AUTOSAR interfaces.

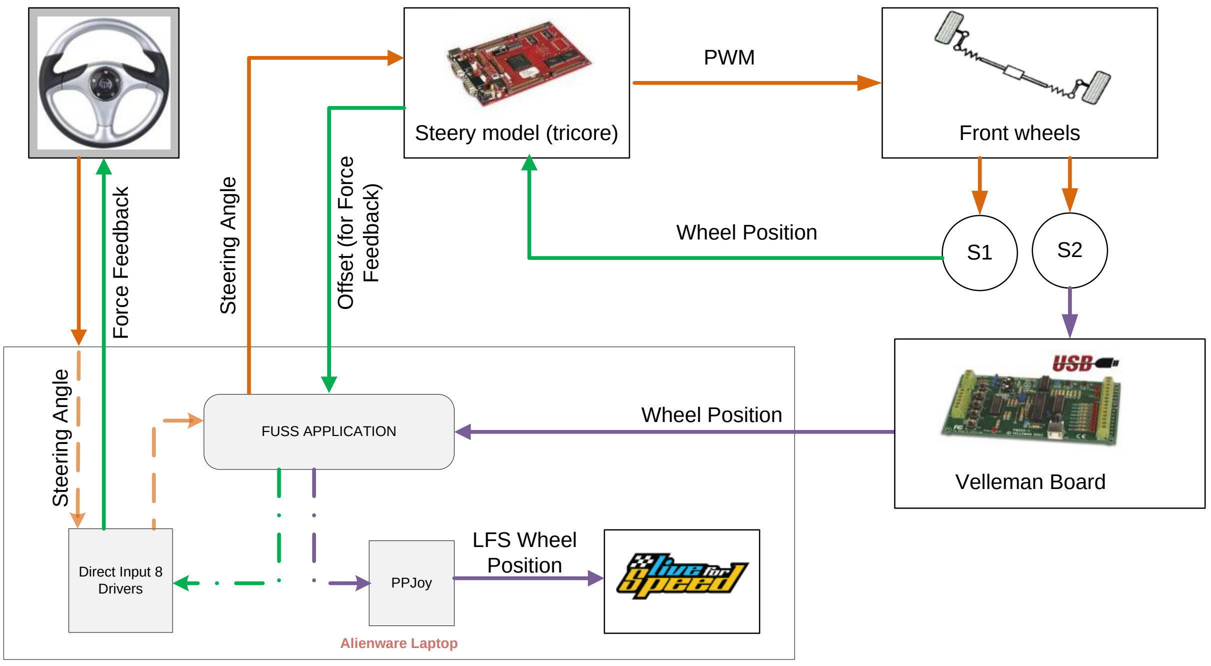


Figure 5: Simulator Architecture for Real Time Environment (using Tricore Board Hardware)
Here we can see the newly added sensor as well as the control algorithm being now implemented in an embedded system. Not only can the generated code from the Simulink model be optimized for this embedded system, the overall platform is now based on standardized AUTOSAR interfaces.

4. Implementation

This chapter describes the exploration and modeling of the SBW system as when the master thesis was started (without support for the Tricore hardware), the extended and improved model to be deployed on the Tricore hardware and implementation of AUTOSAR End-To-End library (E2E) for safety-related data.

In the following Sections 4.1 and 4.2, i.e., in the Steery application with and without Tricore hardware (TriBoard), the Steery application receives a steering angle from the FUSS application through a CAN bus (it also receives the speed of the vehicle which is not used in the present work). The Steering angle is polled from the Logitech G25 Racing wheel that has a numerical value range from -10,000 to 10,000. The received steering angle must then be converted to a wheel angle command that will be sent as an input to the microcontroller.

4.1 Steery Application without Triboard

This section explains the study of previous design and its corresponding modeling for the existing system with code generation methodology.

4.1.1 Exploration of previous system

This master thesis is an extension of a previous Chalmers University thesis work at Mecel – "Functional System Safety – Simulator Environment" [29] by Petter Gustavsson and Henrik Roslund. Therefore the first step of the project was to explore the formerly designed system for the complete understanding of the system architecture. As a result, various project requirements and their constraints of implementation were categorized in the early stage. Though there were few changes on the project specification, corresponding changes were made in the design and its documentation was updated subsequently.

4.1.2 Simulink Modeling

As explained in Section 3.1, a simple Steery model/application was designed using Simulink, which converts the input steering angle into a wheel angle command that is given to the Basic Stamp microcontroller. This is shown in the following Figure 6, with *SteeringAngle* as input port and *WheelAngleCommand* as output port, which feeds the basic stamp.

The main functions to be performed by this Steery application are:

- Implementing the Transfer Function that calculates the wheel angle command for a given steering angle.
- Managing the CAN communication with the FUSS application.
- Managing the serial communication with the Basic Stamp.

The Transfer Function that calculates the wheel angle command is passed to the Basic Stamp, which is the core function of the Steery model and is rather straightforward to implement in MATLAB/Simulink. However, managing CAN and serial communications within or outside MATLAB/Simulink was not an obvious decision and each of the methods has its own constraints that are explained in section 3.2.

In order to develop the application with the available resources, a decision was made to design the Transfer Function alone in the Simulink by managing the CAN and serial communications outside the boundaries of the Steery model.

The detailed reasoning of the logic implemented in the transfer function is discussed in APPENDIX B, where as its Simulink design is shown in the following Figure 6.

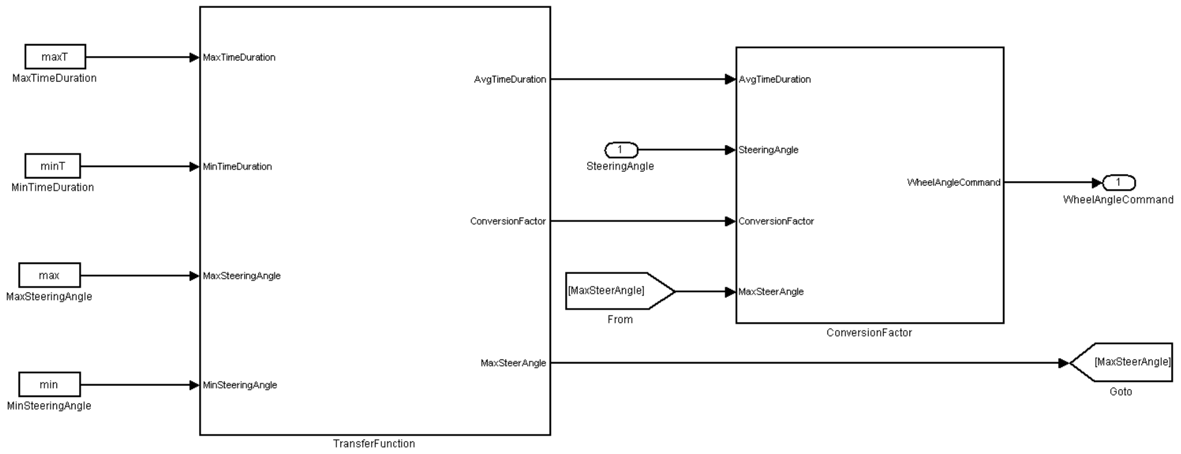
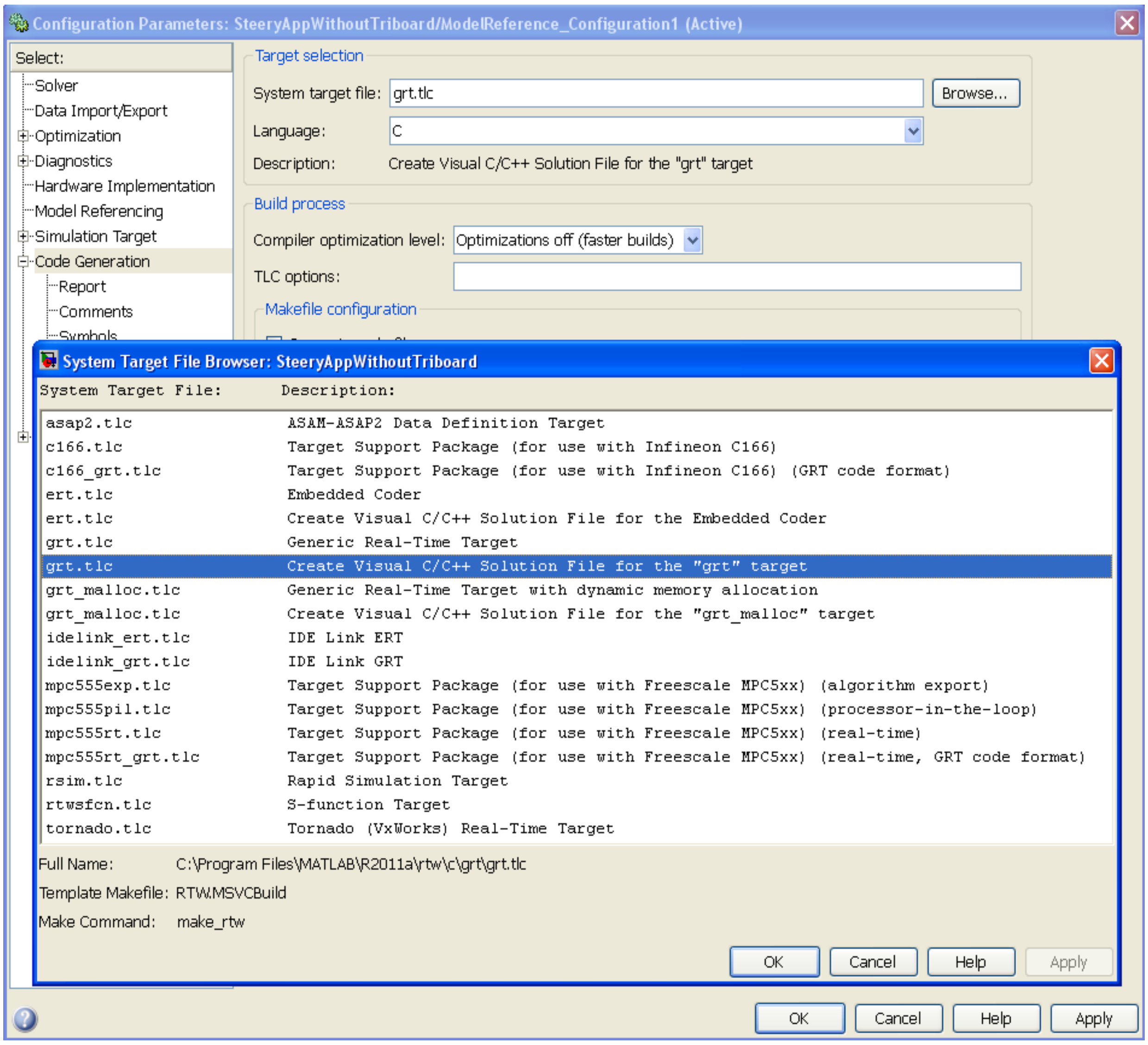


Figure 6: Steery Application without TriBoard

This view represents the top view of the simulink model. We can see the main input being the steering angle coming from the steering wheel being converted to a wheel angle command based on several different parameters.

4.1.3 Code Generation and Tweaking

The code generation of the model presented in Section 4.1.2 starts with the choice of a Target Language Compiler (TLC) file. The TLC files are ASCII files that explicitly control the way code is generated by Simulink Coder or Embedded Coder. Simulink comes with some generic TLC files for generic real-time target (grt.tlc) that will generate code (as in Figure 7). This runs on any computer that has an Intel x86 architecture. As stated before, the generated code will be integrated to the Visual C++ project containing a Graphical User Interface (GUI) and the code to manage the communications with the rest of the prototype (more details in section


Configuration Parameters: SteeryAppWithoutTriboard/ModelReference_Configuration1 (Active)

Select:
- Solver
- Data Import/Export
- Optimization
- Diagnostics
- Hardware Implementation
- Model Referencing
- Simulation Target
- Code Generation
  - Report
  - Comments
  - Symbols

Target selection

System target file: grt.tlc

Language: C

Description: Create Visual C/C++ Solution File for the "grt" target

Build process

Compiler optimization level: Optimizations off (faster builds)

TLC options:

Makefile configuration

System Target File Browser: SteeryAppWithoutTriboard

System Target File:	Description:
asap2.tlc	ASAM-ASAP2 Data Definition Target
c166.tlc	Target Support Package (for use with Infineon C166)
c166_grt.tlc	Target Support Package (for use with Infineon C166) (GRT code format)
ert.tlc	Embedded Coder
ert.tlc	Create Visual C/C++ Solution File for the Embedded Coder
grt.tlc	Generic Real-Time Target
grt.tlc	Create Visual C/C++ Solution File for the "grt" target
grt_malloc.tlc	Generic Real-Time Target with dynamic memory allocation
grt_malloc.tlc	Create Visual C/C++ Solution File for the "grt_malloc" target
idelink_ert.tlc	IDE Link ERT
idelink_grt.tlc	IDE Link GRT
mpc555exp.tlc	Target Support Package (for use with Freescale MPC5xx) (algorithm export)
mpc555pil.tlc	Target Support Package (for use with Freescale MPC5xx) (processor-in-the-loop)
mpc555rt.tlc	Target Support Package (for use with Freescale MPC5xx) (real-time)
mpc555rt_grt.tlc	Target Support Package (for use with Freescale MPC5xx) (real-time, GRT code format)
rsim.tlc	Rapid Simulation Target
rtwsfcn.tlc	S-function Target
tornado.tlc	Tornado (VxWorks) Real-Time Target

Full Name: C:\Program Files\MATLAB\R2011a\rtw\c\grt\grt.tlc

Template Makefile: RTW.MSVCBuild

Make Command: make_rtw

Figure 7: Simulink Coder Configuration Parameter Settings for auto code generation
The different system target files (.tlc) represent the different variants for which the generated code can be oprimized.

3.2). Consequently, for compatibility purposes, Visual C++ compiler is used by Simulink Coder while generating the code.

Unlike Embedded Coder, Simulink Coder does not offer optimized code depending on the target the user chooses. Consequently, the generated code contains methods for all the functionality that a user might use in the design of a Simulink model. That leads to a somehow non optimized generic code. The generated code basically simulates an embedded version of the Simulink environment.

Thus, the first step was to go through the generic code and identify methods that were important for steer-by-wire application. Those included mainly the function related to the initialization of the model and the loop through the Step and Update functions. The Step and Update functions implement together one execution of the whole model. The second step was to make the code lighter, and effectively removing portions of code that were not relevant for the Steery application. A third step was to modify the structure declarations so that:

- Steery could access the result of the Transfer Function outside the boundaries of the generated code i.e. retrieve it in the VC++ project that implements communication methods
- Pass the steering angle input from the VC++ project (received through CAN) to update the step function of the generated code.

Once that is done, the next phase was combining and compiling the two parts of the project. All the header and library files related to the generated code must be referenced in the VC++ project as well as the linkage paths. One of the steps that we failed to understand for a significant amount of time was to include the preprocessor definitions related to the generated code into the VC++ project. Those include the name of the Simulink model, if the application is running in Real-Time (RT) or Non-Real-Time (NRT), the number of sample times (NUMST) and the number of continuous steps (NCSTATES).

This would lead to a fail while building the project as some checks included in generated-code related headers would fail. When it was fixed, the solution was ready to be tested on the simulator. During the first tests, while the result was fruitful, it was noticed that a significant delay was induced by the execution time of the generated code, which lead to the regular loss of steering angle data. The requirement was set so that when a steering angle is received by the application through the CAN bus, it is queued and when the data is going to be treated i.e. sent to the generated code, the timestamp is checked. If it is older than 10ms then the data is ignored and the queue is flushed. This is to prevent an “avalanche effect” where a slight delay would accumulate up to a point where the synchronization between the steering wheel and the front wheels is totally lost.

The delay induced by the generated code caused the receiving queue to be flushed regularly so it was decided to inspect using execution time control markers to find out which portion of the generated code induced the most delay. It was found out that the 6ms delay was induced by the data logging function which was of absolutely no use in our case, so that function was excluded from the project. Later tests verified and validated the model successfully.

4.2 Steery application with Triboard

In this section modeling of updated Simulink model and its code generation process is explained.

4.2.1 Simulink Modeling

In TriBoard the Steery application is designed as one ECU that holds two SWC's (software components) namely: Steering Wheel SWC and Front Wheel SWC, which are shown in Figure 8. The functionality of the Front Wheel SWC is almost the same as the one from Steery application without Tricore (as in Section 4.1.2). Here, in addition, the Front Wheel SWC passes the front wheel sensor information to the Steering Wheel SWC. Previously, in the Steery application without Tricore, this sensor information was maintained only by FUSS application. On the other hand, the Steering Wheel SWC serves two functionalities: It passes the Steering Wheel information (G25 Racing Wheel- Steering angle) to the Front Wheel SWC and calculates the force feedback that will be given to Steering Wheel in order to maintain front wheels position with respect to Steering Wheel.

When the Steery application was moved to Tricore hardware, Steery application will have support from Picea; Mecel's AUTOSAR product. As a result, only the core application is developed with proper interfaces, which later on will be connected to BSW and hardware through RTE by Picea team – this process was defined in AUTOSAR specifications. In this stage of the project, the basic stamp microcontroller is replaced with an inbuilt microcontroller of the TriBoard. During this process to maintain the servo motor resolution with respect to Steering Wheel, both appropriate operating frequency and Time Duration (duty cycle) information should be sent to the microcontroller or an updated Transfer Function should be made in Front Wheel SWC. Since the servo motor which is used in this project was operated around 50Hz with an operating positive duty cycle range from 1.0ms to

2.0ms (front wheel movements from Left to Right), calculation of Time Duration and converted resolution can be done easily. For details of this calculations please refer APPENDIX B.

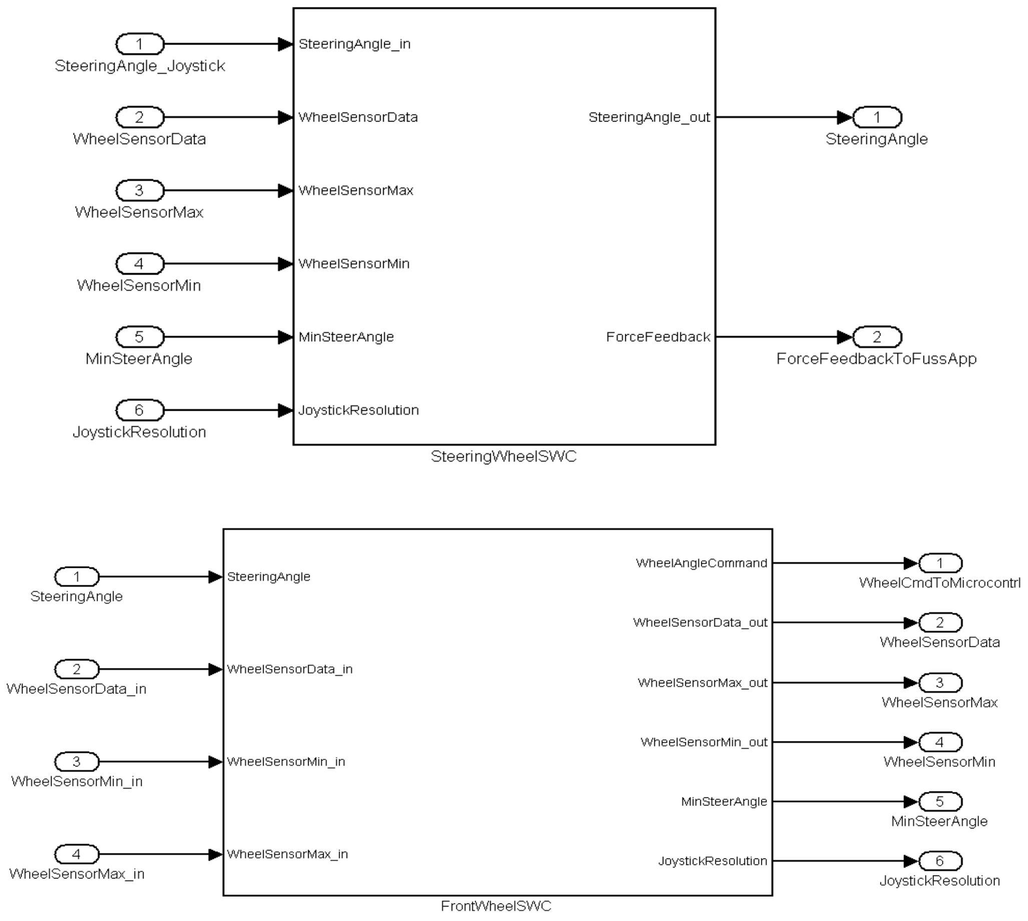


Figure 8: Software Components in Steery Application that can be integrated to the TriBoard Hardware

This again represents the top view of the model. We can see here that the SW-C handles the force feedback fed to the steering wheel. There are additional inputs on FrontWheelSWC which are the sensor data coming from the front wheels that are necessary to calculate the force feedback.

Coming back to the Front Wheel SWC, once the wheel angle command is passed to the microcontroller, microcontroller generates the corresponding PWM signal. Based on that PWM value, the front wheels will respond through the servomotors. Relating to these front wheel moments, the corresponding wheel sensor information is passed to the Steering Wheel SWC. Then this Steering Wheel SWC calculates the force feedback, which will be delivered to FUSS application through a CAN bus. Figure 9, shows the logic of the Force Feedback calculation.

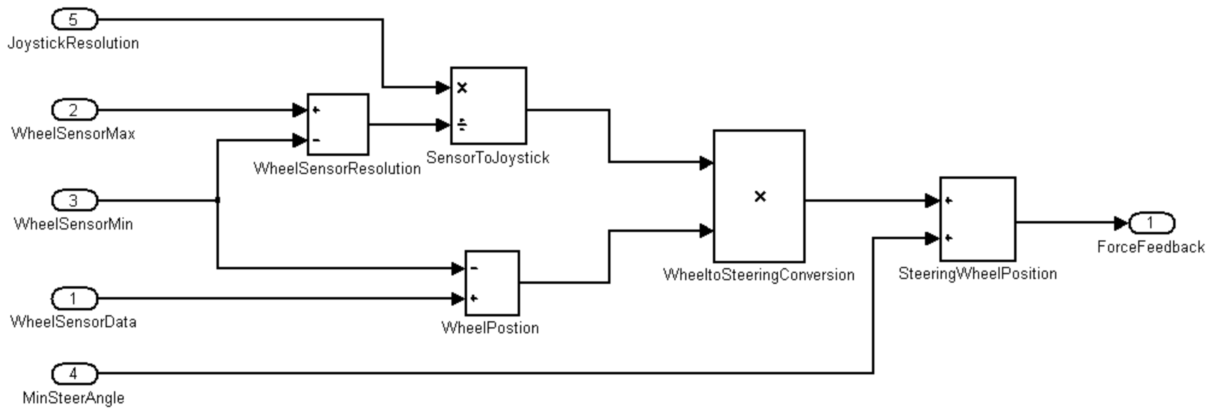
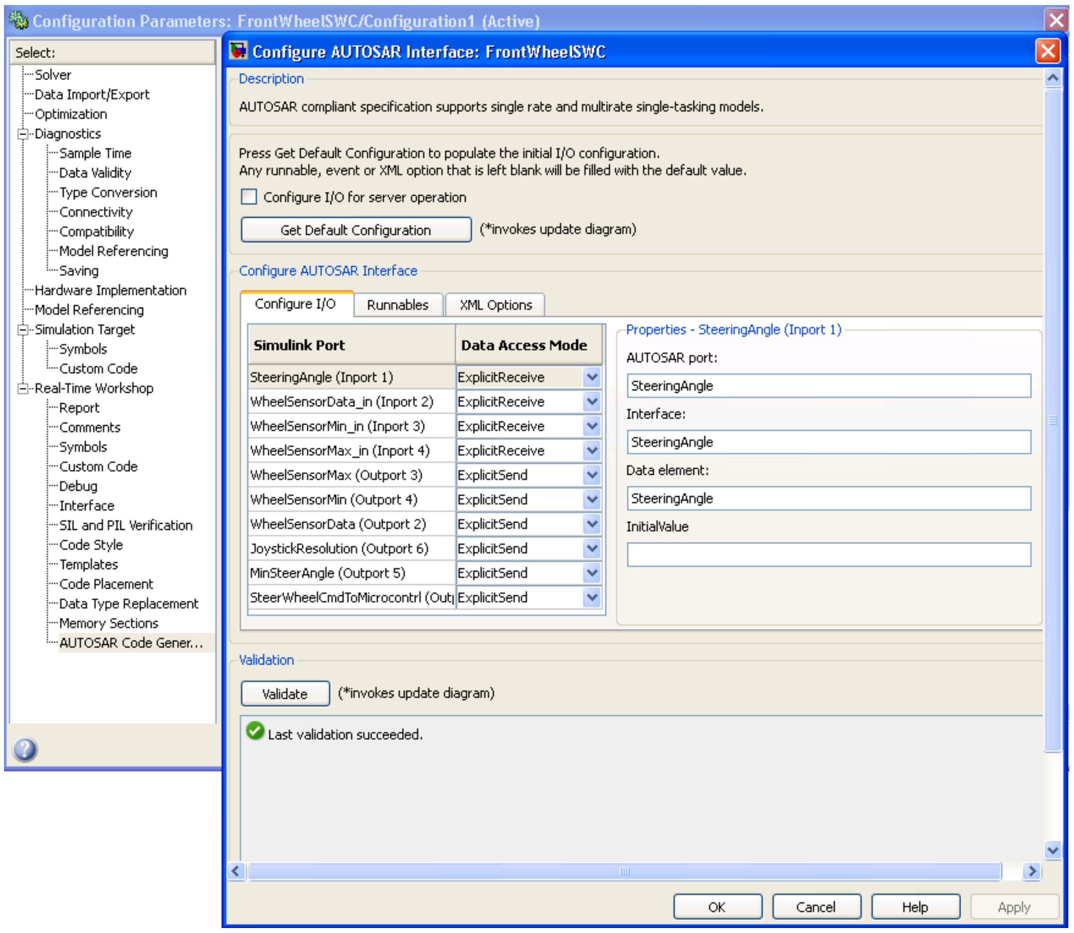


Figure 9: Calculation of Force Feedback
This represents the lower view of the force feedback calculation that is located in the SteeringWheelSWC. This mainly is a quotient of the front wheel range and the steering wheel range.

4.2.2 Code Generation with AUTOSAR interfaces

The Simulink model in Section 4.2.1 is designed using AUTOSAR interfaces. As explained in Section 2.2.2, the input and output data of the SWCs' will be through RTE for which


Configuration Parameters: FrontWheelSWC/Configuration1 (Active)

Select:
- Solver
- Data Import/Export
- Optimization
- Diagnostics
  - Sample Time
  - Data Validity
  - Type Conversion
  - Connectivity
  - Compatibility
  - Model Referencing
  - Saving
- Hardware Implementation
- Model Referencing
- Simulation Target
  - Symbols
  - Custom Code
- Real-Time Workshop
  - Report
  - Comments
  - Symbols
  - Custom Code
  - Debug
  - Interface
  - SIL and PIL Verification
  - Code Style
  - Templates
  - Code Placement
  - Data Type Replacement
  - Memory Sections
  - AUTOSAR Code Gener...

Configure AUTOSAR Interface: FrontWheelSWC

Description

AUTOSAR compliant specification supports single rate and multirate single-tasking models.

Press Get Default Configuration to populate the initial I/O configuration.
Any runnable, event or XML option that is left blank will be filled with the default value.

☐ Configure I/O for server operation

Get Default Configuration (*invokes update diagram)

Configure AUTOSAR Interface

Configure I/O | Runnables | XML Options

Simulink Port	Data Access Mode
SteeringAngle (Inport 1)	ExplicitReceive
WheelSensorData_in (Inport 2)	ExplicitReceive
WheelSensorMin_in (Inport 3)	ExplicitReceive
WheelSensorMax_in (Inport 4)	ExplicitReceive
WheelSensorMax (Outport 3)	ExplicitSend
WheelSensorMin (Outport 4)	ExplicitSend
WheelSensorData (Outport 2)	ExplicitSend
JoystickResolution (Outport 6)	ExplicitSend
MinSteerAngle (Outport 5)	ExplicitSend
SteerWheelCmdToMicrocontrl (Out	ExplicitSend

Properties - SteeringAngle (Inport 1)

AUTOSAR port: SteeringAngle
Interface: SteeringAngle
Data element: SteeringAngle
InitialValue:

Validation

Validate (*invokes update diagram)

✓ Last validation succeeded.

OK | Cancel | Help | Apply

Figure 10: Defining AUTOSAR Interfaces in Embedded Coder
Since the communication setup is not handled on the application level, the SW developper must specify the input and output port (receive and send as well as interface and port) so that the RTE and BSW can route it to the intended recipient.

corresponding interfaces should be defined accordingly. For that reason, as like in Section 4.1.3, instead of generic TLC, an autosar TLC (autosar.tlc) is used here. During this stage of the project, since Simulink Coder does not have support for autosar.tlc, Embedded Coder was used. Figure 10 shows the way of defining the AUTOSAR interfaces for one of the SWC.

All the input ports of the SWC have a data access mode as Explicit Receive, which can also be represented as *RTE_Read* interface. This interface is used to get the input data from the external environment; which can be either intra ECU communication or inter ECU communication through RTE. Similarly, all the output ports have the data access mode as Explicit Send, which can also be represented as *RTE_Write* interface. This interface is used to transmit the data from the application level to lower levels of the AUTOSAR architecture in inter or intra ECU communication (See Figure 2). Figure 11 explains in detail one of the *RTE_Read* and *RTE_Write* data flow from one SWC to another.

The interface ‘a’ refers to Explicit Receive interface, which is the input from the G25 Racing Wheel that transmits the steering angle. This steering angle is received by the Steering Wheel SWC through RTE. Since this steering angle is used to calculate the wheel angle command, which is input to the microcontroller, the received steering angle must be sent to the Front Wheel SWC as it is responsible for the wheel angle command calculations. So, interface ‘b’ is used to send the steering angle data from Steering Wheel SWC via RTE. Then Front Wheel SWC receives this steering angle at interface ‘c’ and transmits the wheel angle command to the microcontroller though interface ‘d’, which is *RTE_Write* or Explicit Send.

Once the AUTOSAR interfaces are configured properly, then by choosing appropriate solver and end target options in the configuration parameters of the Embedded Coder, the corresponding code will be generated for their equivalent SWC’s. Then this generated code can be built along with the Middle Software and be deployed on the suitable hardware.

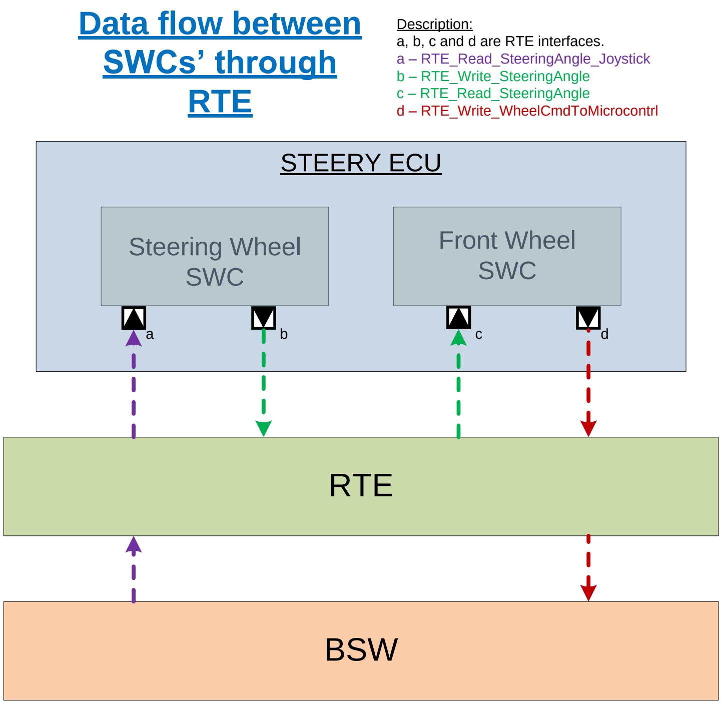

Data flow between SWCs' through RTE
Description:
a, b, c and d are RTE interfaces.
a – RTE_Read_SteeringAngle_Joystick
b – RTE_Write_SteeringAngle
c – RTE_Read_SteeringAngle
d – RTE_Write_WheelCmdToMicrocontrl
STEERY ECU
Steering Wheel SWC
Front Wheel SWC
a
b
c
d
RTE
BSW

Figure 11: Data Flow between SWCs through RTE
Intra-ECU communication does not go down all the way to the Basic SoftWare but CAN communication or communication to sensors/actuators is handled in lower layers of the AUTOSAR topology.

4.3 AUTOSAR End-To-End protection library

4.3.1 Key Concepts

The concept of End-to-End (E2E) protection assumes that safety-related data exchange shall be protected at runtime against the effects of faults within the communication link. Those include random HW faults (e.g. corrupt registers of a CAN transceiver), interference (e.g. due to EMC), and systematic faults within the software implementing the VFB (e.g. RTE, IOC, COM and network stacks) [18].

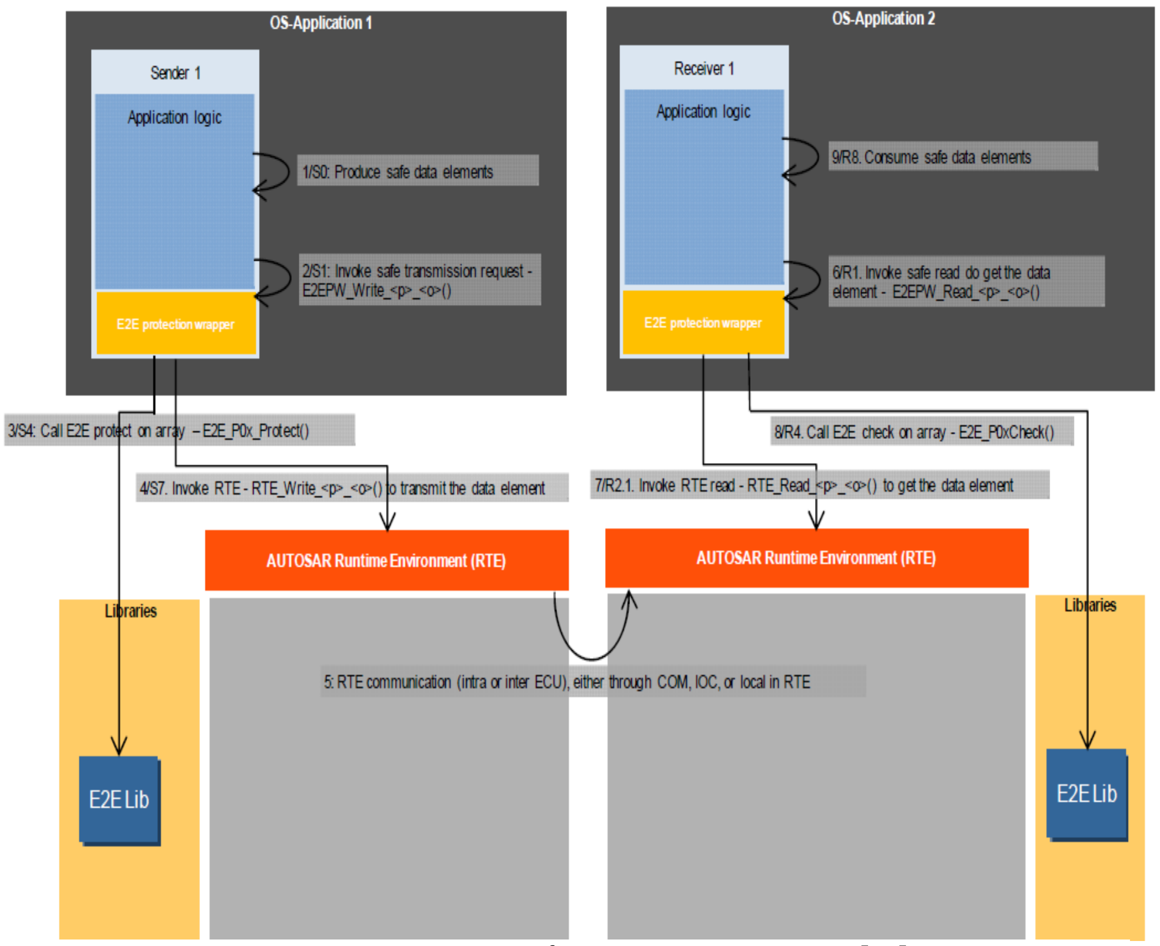
The E2E Library provides mechanisms for E2E protection, adequate for safety-related communication having requirements up to ASIL D (related to ISO26262 standard).

The E2E protection allows the following:

- It protects the safety-related data elements to be sent over the RTE by attaching control data,
- It verifies the safety-related data elements received from the RTE using this control data, and

- It indicates that received safety-related data elements faulty, which then has to be handled by the receiver SW-C.

To provide the appropriate solution addressing flexibility and standardization, this document [18] specifies a set of flexible E2E profiles that implement an appropriate combination of E2E protection mechanisms. Each specified E2E profile has a fixed behavior, but it has some



OS-Application 1
Sender 1
Application logic
1/S0: Produce safe data elements
2/S1: Invoke safe transmission request - E2EPW_Write_<p>_<o>()
E2E protection wrapper
OS-Application 2
Receiver 1
Application logic
9/R8. Consume safe data elements
6/R1. Invoke safe read do get the data element - E2EPW_Read_<p>_<o>()
E2E protection wrapper
3/S4: Call E2E protect on array – E2E_P0x_Protect()
4/S7. Invoke RTE - RTE_Write_<p>_<o>() to transmit the data element
8/R4. Call E2E check on array - E2E_P0xCheck()
7/R2.1. Invoke RTE read - RTE_Read_<p>_<o>() to get the data element
AUTOSAR Runtime Environment (RTE)
AUTOSAR Runtime Environment (RTE)
Libraries
Libraries
5: RTE communication (intra or inter ECU), either through COM, IOC, or local in RTE
E2E Lib
E2E Lib

Figure 12: Overview of E2E Protection Wrapper [18]

This figure represents the communication between two SW-Cs that implement E2E protection library mechanisms. This takes place in four steps, from the production of safe data elements (on the application level) to the invocation of the rte_write() operation. The receiving SW-C will in turn read the data packet, check the content against E2E library mechanisms and consume the data.

configuration options by function parameters (e.g. the location of CRC in relation to the data, which are to be protected).

At runtime each caller (e.g. via E2E protection wrapper or a callout) calls the functions of an E2E profile including their function parameters (the configuration options are not defined statically e.g. at pre-compile time or post-build time). To ease their usage there is a recommended configuration (variant) for each E2E profile.

The E2E Library can be invoked at the level of Data Elements (SWC) or at the level of COM I-PDUs. In the former approach, every safety-related SW-C has its own additional sub-layer (which is a .h/.c file pair) called E2E Protection Wrapper, which is responsible for marshaling of complex data elements into the layout identical to the corresponding I-PDUs (for inter-ECU communication), and for correct invocation of E2E Library and of RTE [18].

As shown in Fig.12, the first step is for the SWC to produce the data element to be sent through the RTE. A call to the wrapper is then made that acts as a transmission request. The wrapper calls a protection library function from the E2E library that essentially attaches the control data to the data. Finally, an *rte_write()* call (as explained in section 4.2) is made transmitting the produced <data, control_data> pair.

4.3.2 Implementation

Each E2E Profile shall use a subset of the following data protection mechanisms [18]:

- A CRC, provided by CRC library;
- A Sequence Counter incremented at every transmission request, the value is checked at receiver side for correct incrementation;
- An Alive Counter incremented at every transmission request, the value checked at the receiver side if it changes at all, but correct incrementation is not checked
- A specific ID for every port data element sent over a port (global to system, where the system may contain potentially several ECUs).
- Timeout detection:
    - Receiver communication timeout
    - Sender acknowledgement timeout
- Double inverse data, as an alternative to CRC.

Depending on the used communication and network stack, appropriate subsets of these mechanisms are defined as E2E communication profiles. Some of above mechanisms are implemented in RTE, COM and/or communication stacks. However, to reduce or avoid an allocation of safety-requirements to these modules, they are not used: E2E Library provides all mechanisms internally (only with usage of CRC library). The E2E Profiles can be used for both inter and intra ECU communication. The E2E Profiles 1 and 2 are optimized for communication over CAN, FlexRay and can be used for LIN whereas the E2E Profile 3 is optimized for intra-ECU communication. Depending on the system, the user selects which

E2E Profile is to be used, from the E2E Profiles provided by E2E Library. For the application developed within the thesis work, FlexRay buses will be used on target. Moreover [19] advise the use of Profile 2 for applications designed with safety requirements of ASIL D so profile 2 was naturally chosen.

The following table summarizes the detectable failure modes with respect to the mechanisms of Profile 2:

Mechanism	**Detected failure modes**
Counter	Unintended message repetition, message loss, insertion of messages, re-sequencing
Data ID	Insertion of messages, masquerading
CRC	Message corruption, insertion of messages (masquerading)
Timeout (detection & handling implemented by SW-C)	Message loss, message delay

Table 1: Detected failure modes using E2E Profile 2 [18]

The mechanism that is the most useful to our application is the CRC checksum because it is very critical for a steer-by-wire application to receive correct steering angle otherwise the front wheel get desynchronized with the steering wheel of the driver. Data ID would be less valuable because we can assume that the automotive manufacturer will isolate the steer-by-wire application on an ECU and a CAN network that is not shared with potential dangerous applications.

The application must call the E2E protection wrapper each time data is to be sent or received. In order to keep an automated tool chain and avoid generating code with Simulink embedded coder and tweak it to include the call to the wrapper, the developer can use custom coding offered by MATLAB/Simulink. Custom coding allows SWC developers to specify a function to call to replace another. As explained in Section 4.2, an output port in Simulink will generate an *rte_write()*. However, implementing E2E library on top, an output port shall now commit a call to the wrapper that will protect the data and then only make a call to *rte_write()*.

5. Results and Tool Chain

In this Chapter, a brief introduction of different MathWorks products is given, which were used during the process of Steery application design. In addition to it, we also discussed about different testing options that are available during this stage and gave the corresponding reasoning for choosing one particular testing method. In the end we documented related test results.

5.1 Testing Methodologies

In this section we describe about different testing methods that were accessible during the time of our project. Then we chose one of the following methods to obtain our final results after doing some analysis as explained below.

5.1.1 Modeling Tools

We designed the Steery application using basic Simulink library blocks. As described in Section 4.2.1, the Steery application consists of two software components, in the sense, two individual Simulink model files designed separately. To test the algorithm of these models, we used model referencing methodology, in which both the models are connected together as shown in Figure 13.

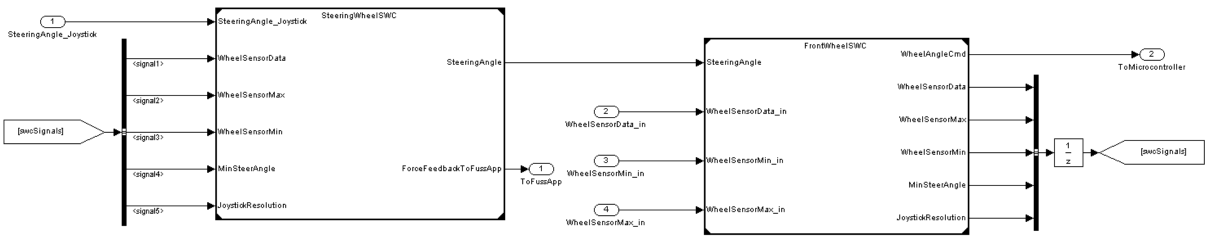


Figure 13: Steery Application with both the SWCs connected together
This view specifies the linkage of the two SW-Cs in Simulink mainly for testing purposes, In that way the SW developer can specifies the input data (by taking real life expected values) and ***assert*** *the outputs to verify that the calculated steering wheel command and force feedback match expected results.*

On the other hand, to test the designed application within Simulink (assuming without using any testing tools), a simulator environment has to be designed with the help of some Mathworks tools that falls under Physical Modeling (Simulink Products). Some of the tools available from the Physical Modeling are Simscape, SimMechanics and SimDriveline. Using these three tools, we made a test environment, which gives sensor information corresponding to front wheel position, and a replica of microcontroller that can activate the servomotors.

Since the simulator environment design is out of the scope in thesis description, the development was not made to larger extent. Instead, a basic study was prepared with the following conclusion: by using the physical modeling tools and Simulink 3D animation, we can model a test environment for the Simulink models that are presented in Section 4.2.1. A direct added benefit is that changes can be easily implemented and tested directly in the Simulink environment without having to generate any code. Once the final application is verified then the corresponding code can be generated and tested in the AUTOSAR VFB or in the target hardware. The first test phase within Simulink gives a positive confidence for smooth integration of generated code into the hardware.

Pros and Cons (Merits and Demerits):

1. Cost of the modeling tools package is lower than for the Design Verifier tool.
2. Once the test environment is done it can be used for different projects that have similar applications.
3. Since the test environment must be designed using Simulink modeling tools it’s a bit more time consuming compared to Design Verifier.
4. Verification of the test environment itself is required in this method before verifying the core application.

5.1.2 Testing Tools

Description of the Tools:

To test the application at the model level, two Simulink products are available from Mathworks: Simulink Verification and Validation and Simulink Design Verifier. A brief summary of these tools are given below:

Simulink verification and validation [20]:

To associate, navigate and review of Simulink models and generated code with the design requirements could be done using this tool along with corresponding report generation. From this generated report highlighting and tagging helps to link the Simulink models or generated code to the requirement documents. For this there are many compatible software’s that are available in the market like: Microsoft Word, Excel, IBM Rational Doors, PDF, etc. In addition to this, compliance checking with modeling standards is available. In addition to that, MAAB (Mathworks Automotive Advisory Board) checks like: Naming Conventions, Model Architecture, Block parameter checks, etc., are possible.

Pros and Cons:

1. Verification of the model can be done just by marking few check boxes in the tool.
2. Logical design of the model can't be verified here.

Simulink Design Verifier [21]:

In this tool Simulink model verification can be done by formal analysis that describes desired and undesired system behavior using different test scenarios and verification objectives. Here, verification of application or system logic at model level can be done using three different modes of the tool. They are: Design Error Detection, Test Generations and Property Proving. Using these three modes we can find dead logics, integer overflow, and division by zero errors and including violations of design properties. Once the model level verification is done, the same test vectors or test cases can also be used for generated code to verify the generated code. In addition to this detail report generation is possible at each level of verification, which can be used for later documentation procedures.

Pros and Cons:

1. Not much test design is required to verify the models. In one of the testing mode; Test generations: Just by providing the input range the tool will generate all the possible test vectors to find the division by zeros and integer overflow errors.
2. The blocks that are used during the testing can be excluded from the code generation automatically.
3. A detailed generated report at each stage of the test gives a clear picture of functionality or test coverage.
4. Though this tool is new to the market, the present price is little expensive than others.

Unit Testing

In this method using C/C++ programming specific entity of the source code can be verified automatically using CppUnit testing methodology.

Pros and Cons:

1. Generated code can be verified by this method.
2. These tests can be implemented only after code generations.
3. For each part of the function or specific entity a CppUnit test is required to verify the functionality.

IEC Certification Kit for ISO 61508 and ISO 26262 [22]

IEC Certification Kit follows an in-context approach to IEC 61508 certification that is based on a specific workflow or set of workflows used when the applicant applies specific tools to develop or verify software for IEC 61508–compliant or IEC 61508–certified applications. The applicant must ensure that the tools are used within the referenced workflows and within the constraints specified in their respective certificates.

5.1.3 Conclusion

It will be harder to use a general tool, such as Simulink Modeling Tools, to develop or to design a test environment for a specific application, than to use a dedicated tool for the verification, such as Simulink Design Verifier. On the other hand, both the Verification and validation tool and the IEC certification kit are limited in their scope to verify some modeling standards and integers overflow errors. These verification tools cannot verify the designed logic of the core application. Similarly, although unit tests are useful for functionality verification, they can be performed only after code generation, which prevents us from testing the functions at the model level. To overcome these limitations we chose the Design Verifier as our main testing tool in this project. The detailed analysis of using Design verifier is explained in Section 5.2.

5.2 Results

As highlighted from the previous section, Simulink Design Verifier is mainly used to test the logic of the Steery application. The following sub-sections explain different modes that are used in Simulink Design Verifier and the corresponding results are shown.

Simulink Design Verifier has three different testing modes, each with its own features that are explained below:

1. Detect Design Errors:

Using this mode of operation, integer overflow and division by zero errors can be identified easily. Based on the warnings generated by the tool, an application designer can rectify corresponding errors before code generation. In APPENDIX C, under the subsections of Front Wheel SWC and Steering Wheel SWC, Design Verifier Report for Design Error Detection is presented. The key point to observe in that report is: based on the Design Min Max Constraints, Design Error Detection objectives were proven valid for both the SWCs'. Based on this, we can verify that the present version of SWCs' is safe from integer over flow errors. In the present version of Simulink Design Verifier, for division by zero errors the tool is not generating test cases to prove in which scenario the error is arising. As per the information from Mathworks support team, this will be updated in the future versions of MATLAB/Simulink (R2011b or above).

2. Generate Tests / Test Generations:

In this mode, Test Generations can be achieved in three ways:

i. By defining Model Coverage Objectives in the configurations parameters
ii. By using *Test Condition* and *Test Objective* blocks from the Simulink Library Browser.
iii. By using the combination of both above techniques.

The generated report in APPENDIX C is based on the third method. In Test Generation, the tool will generate different test cases to show that the given test objective and coverage objectives can be achieved. These test cases will be generated based on the given test conditions by the user. For instance, by considering the case that is used in the generated report: using Model Coverage Objective with MCDC (Modified Condition/Decision Coverage) and Test Condition values with the corresponding SWCs, Design Verifier generated one test case proving that the requested test objective can be achieved. The detailed report of this Test Generation can be seen in APPENDIX C. In many cases, this mode of testing can be viewed as similar to Unit Testing. In unit testing, an output is asserted based on a given well-known input. The expected output is then compared to the asserted output, verifying the proper implementation of the logic. Whereas here, using the verifier tool, unit testing is achieved semi-automatically, the user having to specify a set of input-range as *test conditions* and output-range data as *test objectives*, the corresponding assertion being done by the tool. These assertions are referred as *Test Cases* or *Test Harness Models*. Along with this, a coverage report will be generated at the end, which shows the total coverage of the test objectives. Since for the Steery model, only one test condition was chosen, 100% coverage is achieved and feasible.

3. Property Proving:

Property proving is another testing mode to investigate the logic of the Simulink models. In this mode by providing *Proof Objective* and *Assumption blocks* from the library browser and the type of strategy to use for proving the property in the configuration parameters, Design Verifier can perform Property Proving operations. Three types of strategies can be used here: *Prove*, *Find Violation* and *Prove with Violation Detection*. For this model, *Prove with Violation Detection strategy* was used. This strategy helps in two ways:

1. It proves the values given at proof objective are valid

2. It will try to generate test cases to show different possibilities that may objective as falsified.

For example, as in APPENDIX C under Front Wheel SWC of Property Proving subsection, based on Analysis Constraints and Design Min Max Constraints, the Objective was fulfilled. To see it in detail, the values ranging from -10,000 to 10000 under Analysis Constraints are the input to Front Wheel SWC. For this input range the output at Wheel Angle Command should be in the range of 1500 to 2400. So, this range is given as *Proof Objective* at the corresponding output port. Now, the tool will try to generate test cases to prove the range of values between 1500 and 2400 can be invalid at some input. If that is the case, then it can be consider as if the logic of the Simulink model is wrong for some input condition. Since the tool was not able to find any test case for that scenario it can be marked as verified system.

5.3 Virtual Function Bus

The previous section presented activities conducted during the model development phase to ensure that the model complies with the requirements from the item definition. These activities represent a first mean to verify the logic and robustness of the model within the Simulink environment. However, within Simulink, the model is isolated from its environment, meaning that all the inputs and outputs are managed within the simulating environment. For example, the interface of the Simulink model corresponding to the receipt of the steering angle from steering wheel via a CAN message is defined in Simulink as an input port. All the CAN communication is handled by the VFB so one can easily see that further verification is needed.

To further verify the integration of the generated code for the SWCs, the developer must make a build of the SWCs code generated from Simulink along with the middle software (The VFB). This effectively verifies the interaction between the SWCs and the RTE.

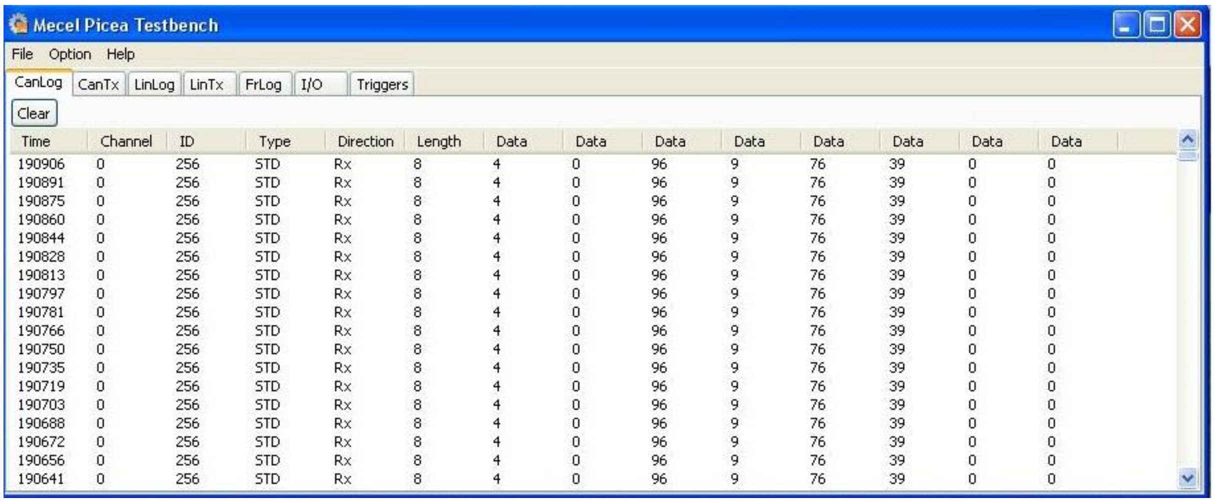
For testing purposes, inputs and outputs of the system have been configured by the VFB integrator as one CAN message respectively. This is because full integration of the PWM signal generator and A/D converter was not available at the time. The byte positioning of the different inputs and outputs, is presented in the following table.

Input	
Byte position	Signal
0	Not used in FUSS
1	SteeringAngleJoystick
2	SteeringAngleJoystick
3	WheelSensorMax_in
4	WheelSensorMin_in
5	WheelSensorData_in
Output	
Byte position	Signal
0	Not used in FUSS
1	Not used in FUSS
2	SteeringWheelCommandToMicrocontroller
3	SteeringWheelCommandToMicrocontroller
4	ForceFeedbackToFussApp
5	ForceFeedbackToFussApp

Table 2: Byte positioning of I/O as implemented in the VFB

Based on this implemented CAN message structure, the proper behavior of the generated code from the Simulink model was verified in the Mecel Picea TestBench. This tool is part of the Mecel Picea Suite [5] and allows interaction with the application (the final build of VFB and SWCs) through inter-process communication in the PC environment. This allows setting up a CAN message with some well-known input values *SteeringAngleJoystick, WheelSensorMax_in, WheelSensorMin_in* and *WheelSensorData_in* for which expected ouptut values *SteeringWheelCommandToMicrocontroller* and *ForceFeedbackToFussApp* are verified.

Figure 14 shows the CAN log within Picea TestBench for input values set up to simulate the vehicle steering in the maximum right direction.



Time	Channel	ID	Type	Direction	Length	Data	Data	Data	Data	Data	Data	Data	Data
190906	0	256	STD	Rx	8	4	0	96	9	76	39	0	0
190891	0	256	STD	Rx	8	4	0	96	9	76	39	0	0
190875	0	256	STD	Rx	8	4	0	96	9	76	39	0	0
190860	0	256	STD	Rx	8	4	0	96	9	76	39	0	0
190844	0	256	STD	Rx	8	4	0	96	9	76	39	0	0
190828	0	256	STD	Rx	8	4	0	96	9	76	39	0	0
190813	0	256	STD	Rx	8	4	0	96	9	76	39	0	0
190797	0	256	STD	Rx	8	4	0	96	9	76	39	0	0
190781	0	256	STD	Rx	8	4	0	96	9	76	39	0	0
190766	0	256	STD	Rx	8	4	0	96	9	76	39	0	0
190750	0	256	STD	Rx	8	4	0	96	9	76	39	0	0
190735	0	256	STD	Rx	8	4	0	96	9	76	39	0	0
190719	0	256	STD	Rx	8	4	0	96	9	76	39	0	0
190703	0	256	STD	Rx	8	4	0	96	9	76	39	0	0
190688	0	256	STD	Rx	8	4	0	96	9	76	39	0	0
190672	0	256	STD	Rx	8	4	0	96	9	76	39	0	0
190656	0	256	STD	Rx	8	4	0	96	9	76	39	0	0
190641	0	256	STD	Rx	8	4	0	96	9	76	39	0	0

Figure 14: Overview of the Mecel Picea TestBench tool

The CanLog captures the input and output data going to the SW-Cs. This is a good tool to verify the functionality, this time not on the model level but on the system level. The application is still not deployed to the embedded system but simulated in a PC environment.

The described process during this chapter highlights one of the advantages of model-based design which is that a first verification and validation is made in the Simulink environment, validating the logic of the model. Then a second verification and validation is done when the final build of the application (VFB+SWCs) is performed thus improving the chances of error detection which is obviously a good thing with regard to ISO 26262.

6. Discussions

6.1 Model-based design in relation to ISO 26262

Model-Based Design (MBD) is a visual method of designing complex applications, which are presently used in many areas where embedded software is mostly used. The main aim is to have an agile design process during the entire development cycle. In MBD, early error detection and correction is more feasible than in a traditional approach, which is one of the critical aspects with respect to ISO 26262. Unlike traditional text – based designs, MBD is less prone to errors and much easier to debug. Since the MBD tools provide a standard environment where even the complex logics or models can be broken down into small hierarchal systems (we referred to them previously as views, i.e top view in Figure 6 and Figure 8), it helps to conceptualize the entire system. In other words, it helps understanding the system architecture from a very high level (i.e two SW-Cs communicating through several given interfaces) down to a very low level (calculating a force feedback based on sensor values and steering wheel command).

As outlined in Section 1.2, one of the purposes of this master thesis was to make suggestions on how MATLAB/Simulink can serve as a step to improve chances for early verification and simulation in accordance with ISO 26262. To fulfill this goal we used a testing tool, Design Verifier, introduced in Section 5.2. In this section, we present our thoughts gathered throughout the duration of the project on how design verifier tool can be used to fulfill the ISO 26262 requirements in some aspects.

Originally, it was intended for this master thesis project to receive early inputs regarding Technical Safety Requirements, as explained in Section 2.2. However, due to an early change in the supervising team at Mecel that has not been possible. Instead we focused on the AUTOSAR E2E library for safety-related data.

Nevertheless, during the duration of the project we gained insights on the understanding of the ISO 26262 standard and on how MATLAB/Simulink tools could be used accordingly. Two TSRs are presented here to support the concept. Those TSRs are made up but realistic.

TRS1 - The wheel angle command to the PWM signal generator shall never have a value out of the range [1500; 2400].

For TSR1, the MATLAB/Simulink tool chain actually provides different complementary mechanisms to ensure the Technical Safety Requirements with a high degree of confidence. First, the modeling tool itself can be used as a first barrier with the saturation block. This block will ensure that when a value out of range is received (may it be via a CAN message or from another sub-system within the same SWC), said values are saturated to the desired range.

Secondly, as described in Section 5.2.2, Simulink Design Verifier presents an interesting approach named property proving. This tool actually tries to violate a strong design property such as the one here and presents violation examples to the developer for analysis and debugging purposes. Since the tool is based on a strong, robust algorithm [21], if the tool is not able to violate the tested TSR, the developer can be fairly confident that no out-of-range value is possible within the model. Finally, if the steering angle was calculated in SWC1 and passed through a Flex-Ray bus to SWC2, errors can be induced by the communication link. By using an adequate E2E profile as described in Section 4.3.2 implementing particularly a CRC checksum, the developer can detect an erroneous value on the receiving SWC and take action accordingly (retransmit or drop value for instance).

TSR2 – The end-to-end timing of the system shall never exceed 100ms.

Unfortunately, for this TSR the MATLAB / Simulink tool chain is of little help. If one knows the execution time of the system, the main delay that can be induced [23] is through network failures or congestion. In AUTOSAR development, this is out of scope for the SWC developer as communication is handled by the VFB. However, the recent FlexRay digital serial bus is deterministic and [24] presents a SBW using FlexRay communication between nodes that proves to be accurate and real-time. Attention to the setup of the FlexRay bus must then be paid by the VFB integrator to validate that particular TSR.

6.2 Lessons learnt

We have seen that model-based design can effectively help development of safety critical systems in accordance to ISO 26262. However, the standard is recent and the tools previously described are quite young. Mathworks felt the business opportunity and pushed those tools into the market. We lack feedback on the robustness of the tools consequently one must be aware that the tool chain helps deploying the safety process but using the tool is far from being enough when it comes to ISO 26262 requirements. The standard does not present any metric one can use to justify ISO 26262 compliance. Focus must therefore be put on systematic functional safety planning, tailoring of well proven company processes and extensive testing procedures to achieve a residual acceptable risk for a given safety related E/E function.

ISO 26262 brings a new complexity layer to the development process. The most important challenge that we were able to recognize is the impact of the AUTOSAR platform on the functional safety concept defined by ISO 26262. AUTOSAR splits the functionality into different independent layers. There is therefore a lack of global control of the overall system and even if focus is put on safety in each layer (namely BSW, RTE and SWC); our conclusion is that this split makes it gets harder to predict what happens when everything is plugged back together. This particular feature of AUTOSAR is there to promote software reuse and interoperability between different vendors. However, small incompatibilities that were acceptable for non-safety related applications become really critical for system such as X-by-wire.

6.3 Future work

The design of the Steer-By-Wire application developed within the context of this master thesis could be refined with respect to one or several Technical Safety Requirements. This would more than likely involve the development of some new logic within the Simulink model based on the TSR themselves.

In the original item description of the Steer-By-Wire application appears a sub-system called “Conny” [25]. The “Conny” sub-system is described as an application running on PC that has fault-injection capabilities to the “Simmy” sub-system. This was thought for “Simmy” to handle such faults thus showing the integration of a safe state within the Simulator. The handling of faults is closely related to the implementation of the Technical Safety Requirements. Once the TSRs are defined by the Safety Team at Mecel and further implemented into the existing logic of the Simulink model, “Conny” could be designed in order to verify and validate the proper implementation of the TSRs, effectively highlighting the ISO 26262 functional safety concept.

Finally, as explained in the Section 5.3, some configuration work is still to be done in the Virtual Function Bus to integrate support for PWM signal generation (signal command of the front wheel) and the A/D converter (sensor position of the wheel to calculate the force feedback). This step is necessary in order to fully integrate the application on the Tricore hardware. However, during the time lap of this master thesis, lot of interaction took place with Daniel and Erik, two Chalmers students doing their master thesis at Mecel on a topic very close to this one (although more hardware oriented) and they were able to successfully use the PWM and A/D modules of the Tricore hardware. The next step is thus the configuration of those modules using Mecel Picea Workbench [5].

7. Conclusion

E/E systems integration on personal cars is on the rise. It started out with systems such as Electronic Stability Control (ESC) and modern versions of the anti-lock braking system (ABS) and has proven not only to increase driver's safety and comfort but to create extra value to personal cars for manufacturers to compete around. Although the advantages offered by Steer-By-Wire are numerous and attractive, its implementation on personal cars has been limited so far for reliability issues.

We believe the new ISO 26262 standard for functional safety on road vehicles is a great cradle for a technology such as SBW to emerge; it will without a doubt impose itself as soon as car manufacturers gain enough maturity in their development process in accordance to ISO 26262.

Throughout this thesis, insight on model-based development for the AUTOSAR platform was gained. We believe that model-based development is not just a trend and is about to impose itself for the automotive industry. In Simulink, the model is directly linked to the generated code, which permits making real-time off-line simulations. Due to these real-time simulations, early error detection makes the process of debugging and correcting defects smoother and more efficient which goes hand – in – hand with the safety activities of ISO 26262. The tool chain itself is continuously evolving and aims to be closely coupled to the AUTOSAR platform and more recently to the ISO 26262 standard. However, both the concepts of the standard and related tools used for these areas are quite new. Consequently, we observed a lack of feedback on the maturity and robustness of the tools and how to use them. We believe that it is a major challenge for the automotive companies to tailor and refine their development processes in accordance to ISO 26262 especially since we felt that the standard leaves a lot of room for interpretation. However, similarly to ISO 9000 standard on quality, we believe that ISO 26262 will become a must-have certification when it comes to safety related system development to OEMs. This naturally explains the recent craze from the automotive companies to early embrace ISO 26262 functional safety concept to their processes. As Mecel is looking to stay at the "forefront of the automotive industry" and to keep a competitive edge on its competitors this master thesis was put together as mean of early investigation along with leading experts on the field.

8. References

[1] AUTOSAR - *Automotive Open System Architechture.* (2003, Sep). March 2011: http://www.autosar.org/download/AUTOSAR_Press_Release_250903_V1_5_f.pdf

[2] Henningsson Pontus, Mecel AB, from DFEA2020 – Mecel internal documents, March 2011.

[3] AUTOSAR – a Standardized Automotive Software Architecture, May 2011, http://www.autosar.org

[4] AUTOSAR presentation Material, Mecel AB internal documents, March 2011.

[5] Mecel Picea Suite, April 2011 http://www.mecel.se/products/mecel-picea/mecel-picea

[6] ISO26262 Wikipedia page, February 2011, http://en.wikipedia.org/wiki/ISO_26262

[7] Dr. Alex Dold (Daimler AG), "Implementation of requierements from ISO 26262 in the development of E/E components and systems", June 2008

[8] Hillenbrand, Heinz, Adler, Müller-Glaser, "An Approach for Rapidly Adapting the Demands of ISO/DIS 26262 to Electric/Electronic Architecture Modeling", Research work.

[9] Torsten Dittel & Hans-Jörg Ayrus, "How to "Survive" a Safety Case According to ISO 26262", Springer-Verlag Berlin Heidelberg 2010.

[10] Live for Speed, Online racing simulator, June 2011, www.lfs.net

[11] MSDN Windows Developer Center, March 2011, http://msdn.microsoft.com/en-us/library/ee416842%28VS.85%29.aspx

[12] Deon van der Westhuysen, PPJoy Documentation, 2010, Documentation received with downloading the application.

[13] Parallax Inc, March 2011, http://www.parallax.com/

[14] Hitec 55.2g HS-645MG Technical specifications, April 2011, http://www.rctoys.com/rc-toys-and-parts/HIT-HS645MG/RC-PARTS-HITEC-SERVOS.html

[15] Damien Walsh, Velleman, May 2011, http://www.velleman.eu/distributor/products/view/?country=be&lang=en&id=351346

[16] Infineon TC1797 Data sheet, March 2011, http://www.infineon.com/dgdl/TC1797_DS_v1.1.pdf?folderId=db3a3043156fd5730115b892639c0e97&fileId=db3a30431ed1d7b2011efeaa4ad16b6d

[17] Kvaser Leaf Professional CAN to USB adapter, April 2011, http://www.kvaser.com/index.php?option=com_php&Itemid=258&eaninput=7330130002432

[18] AUTOSAR, "Specification of SW-C End –to-End Communication Protection Library", V1.0.0, R4.0 Rev 1, December 2009, http://www.autosar.org/download/R3.2/AUTOSAR_SWS_E2ELibrary.pdf

[19] Jonas Svärd, on E2E profile choice, Mecel internal document

[20] Simulink Verification and Validation, May 2011, http://www.mathworks.com/products/simverification/?s_cid=global_nav

[21] Simulink Design Verifier, May 2011, http://www.mathworks.com/products/sldesignverifier/?s_cid=global_nav

[22] IEC certification Kit, May 2011, http://www.mathworks.com/products/iec-61508/

[23] Zhang, Wang and Liao ,"Reliability Research for Steer-By-Wire System of Electric Vehicle" IEEE 2009.

[24] Jianmin Duan, Hualin Deng and Yongchuan Yu , "Co-Simulation of SBW And FlexRay Bus Based on CANoe_MATLAB", Proceedings of the 2010 IEEE International Conference on Automation and Logistics, August 16-20 2010, Hong Kong and Macau.

[25] FUSS Item definition, Mecel AB internal document

[26] Matlab/ Simulink embedded coder, March 2011, http://www.mathworks.com/products/embedded-coder/index.html

[27] Mathworks Training Services Manual, "Integrating Code with Simulink", 2010

[28] Mathworks, Stateflow, March 2011, http://www.mathworks.com/products/stateflow/

[29] Petter Gustavsson and Henrik Roslund, "Functional System Safety – Simulator Environment" Chalmers University 2010.

APPENDIX A – Manual Code integration into Simulink

In section 3.2 we anticipated the possibility of integrating C/C++ code managing I/O communications directly into the Simulink model for purpose of further integration to the simulator. This section reviews our findings on the code integration methods offered within Matlab/Simulink and rates them in accordance to the relevance, with our project.

Manual CMEX S-function

In Simulink, we have the option to write C-code wrapper that will allow Simulink to call external code i.e C code managing CAN communication in our case. This approach has the benefits of allowing full control over application, including how the external code should be called by Simulink [27]. However, it also requires a much higher degree of knowledge of the inner workings of Simulink and would have been a too much time-consuming task considering time limits of this Master Thesis work. As we are going to see the Legacy Code Tool provides an automated mean of generating S-functions.

Legacy code tool

The Legacy Code Tool is a utility that generates an S-function automatically from existing C code by using the specifications that the user supply using Matlab code. It effectively transforms C functions into C-MEX S-functions for inclusion in a Simulink model. Legacy Code Tool is the quickest method of bringing C code into Simulink models, the S-function generation can even be scripted and the tool will automatically create a S-function block that can be directly used in models.

However there is no automated method of adding code around the external routine call from the Matlab code interface. The tool allows very little control over the appearance of automatically generated S-function, which proved to be a big drawback for our project. Moreover, the tool does not support simulating continuous or discrete states which prevents us from using the mdlUpdate() callback function. This function is an absolute must have for us as it is the function that will be called every time we probe our model with a new steering angle value.

State flow

Stateflow® extends Simulink® with a design environment for developing state charts and flow diagrams [28]. In its recent 2.0 release, Stateflow allows very simple user-defined C routines integration into Stateflow charts. However, that solution was not envisioned simply because the Steery model presented in Section 4.1.2 does not make use of the Stateflow product.

APPENDIX B – Reasoning behind usage of constant values in Steery Application

The steering angle data received from the Logitech G25 steering wheel ranges from -10,000 (min – Steering Wheel value) to 10,000 (max – Steering Wheel value).

In APPENDIX D, the servo motor specifications were given, in which it states that 1.0ms of input timing sequence will gives full scale in one direction and 2.0ms in other direction. Based on those values and input resolution, minimum duty cycle or minimum timing (minT) and maximum duty cycle or maximum timing (maxT) should be considered as per the microcontroller specifications.

To calculate the timing within the boundaries of the servo motor, TimeDuration is defined as:

$$TimeDuration = maxT - minT$$

Steering Wheel input resolution is defined as:

$$Steering\ Wheel\ Resolution = \max - min$$

Thus the conversion factor is calculated as

$$ConversionFactor = \frac{TimeDuration}{Steering\ Wheel\ Resolution}$$

Finally for any steering angle received from the CAN bus through FUSS application, corresponding wheel angle command is send to the microcontroller that provides a timing sequence, which will be given as an input to the servo motor.

$$WheelAngleCommand = Avg\ of\ TimeDuration + steeringAngle * ConversionFactor$$

The average of maxT and minT (Avg of TimeDuration) corresponds to the neutral position (steering rack or front wheels are in 0 degrees position).

Steery Application without Tricore:

Here, based on Basic Stamp microcontroller, for better resolution, a manual test is performed with front wheel prototype and microcontroller to choose minT and maxT as 1500 and 2400. Based on these values, the wheel angle command will get a range of values between 1500 and 2400. By sending these wheel angle command values to the microcontroller it will gives a timing sequence (positive PWM signal) that varies from 1.0ms to 2.0ms. This PWM signal is given as an input to the Servo Motor that helps the Front Wheel Rack/Prototype to react accordingly.

Steery Application with Tricore:

After Steery application is moved to Tricore hardware, instead of basic stamp microcontroller an inbuilt microcontroller is used. Since the hardware specifications are controlled by Picea team, the output of Front Wheel SWC and Steering Wheel SWC are given to the hardware through Picea Work bench. Since the Picea workbench is not ready during the time of our application design, we assumed the microcontroller inside the Tricore will also support as similar to basic stamp microcontroller. Based on this, minT and maxT were chosen same as before. Since this is documented in the M-file of FussSBWconfig.m; the major changes will be just changing the values of minT and maxT in M-fle with the corresponding hardware values. Once microcontroller receives a value from the converted range it generates a corresponding PWM signal that can transmit to the Servo Motor. On the other hand, if the Tricore hardware's microcontroller needs the information of duty cycle and operating frequency instead of passing the values given at minT and maxT frequency and duty cycle values can be transmitted. This can also be done with ease, since the operating frequency is constant (50 Hz), based on that duty cycle can be calculated, since it is just the ratio between duration of the input timing with entire time period (entire time period T is the reciprocal of servo operating frequency). But in this method there will be slight modifications in the Simulink design.

APPENDIX C – Design Verifier Report

This chapter is separated in two parts with the following titles: Steering Wheel SWC and Front Wheel SWC. Under these, there will be three sections named with Detect Design Errors, Test Generations and Property Proving. In each section an auto generated Simulink Design Verifier Report is accessible, whose explanation is given in Section 5.2.2.

Steering Wheel SWC

a)For Detect Design Errors

Simulink Design Verifier Report

SteeringWheelSWC

AjayN

04-Aug-2011 09:51:04

Innehållsförteckning

Kapitel 1. Summary

Analysis Information

Model:	SteeringWheelSWC
Mode:	DesignErrorDetection
Status:	Completed normally
Analysis Time:	25s

Objectives Status

Number of Objectives: 6
Objectives Proven Valid: 6

Kapitel 2. Analysis Information

Innehållsförteckning

Model Information

File:	SteeringWheelSWC
Version:	1.103
Time Stamp:	Thu Aug 04 09:46:02 2011
Author:	AjayN

Analysis Options

Mode:	DesignErrorDetection
Detect integer overflow:	on
Detect division by zero:	on
Maximum Processing Time:	300s
Block Replacement:	off
Parameters Analysis:	off
Save Data:	on
Save Harness:	off
Save Report:	off

Constraints

Design Min Max Constraints

Name	Design Min Max Constraint
WheelSensorMax	[211, 211]
WheelSensorMin	[41, 41]
JoystickResolution	[20000, 20000]
WheelSensorData	[41, 211]
MinSteerAngle	[-10000, -10000]

Kapitel 3. Design Error Detection Objectives Status

Innehållsförteckning

Objectives Proven Valid

#	Type	Model Item	Description	Test Case
2	Overflow	SteeringWheelSWC/Offset_ForceFeedback/WheelSensorR	Overflow	n/a

#	Type	Model Item	Description	Test Case
		esolution		
4	Division by zero	SteeringWheelSWC/Offset_ForceFeedback/SensorToJoystick	Division by zero	n/a
5	Overflow	SteeringWheelSWC/Offset_ForceFeedback/SensorToJoystick	Overflow	n/a
7	Overflow	SteeringWheelSWC/Offset_ForceFeedback/WheelPostion	Overflow	n/a
9	Overflow	SteeringWheelSWC/Offset_ForceFeedback/WheeltoSteeringConversion	Overflow	n/a
11	Overflow	SteeringWheelSWC/Offset_ForceFeedback/SteeringWheelPosition	Overflow	n/a

Kapitel 4. Signal Bounds

Signal	Bounds
SteeringAngle- outport 1	[-32768, 32767]
SteeringWheelSWC/Offset_ForceFeedback/WheelSensorResolution- outport 1	170
SteeringWheelSWC/Offset_ForceFeedback/SensorToJoystick- outport 1	118
SteeringWheelSWC/Offset_ForceFeedback/WheelPostion- outport 1	[0, 170]
SteeringWheelSWC/Offset_ForceFeedback/WheeltoSteeringConversion- outport 1	[0, 20060]
SteeringWheelSWC/Offset_ForceFeedback/SteeringWheelPosition- outport 1	[-10000, 10060]

b) For Test Generations

Simulink Design Verifier Report SteeringWheelSWC AjayN 04-Aug-2011 10:07:50

Innehållsförteckning

Kapitel 1. Summary

Analysis Information

Model:	SteeringWheelSWC
Mode:	TestGeneration
Status:	Completed normally
Analysis Time:	0s

Objectives Status

Number of Objectives: 1
Objectives Satisfied: 1

Kapitel 2. Analysis Information

Innehållsförteckning

Model Information

File:	SteeringWheelSWC
Version:	1.107
Time Stamp:	Thu Aug 04 10:07:37 2011
Author:	AjayN

Analysis Options

Mode:	TestGeneration
Test Suite Optimization:	CombinedObjectives
Maximum Testcase Steps:	500 time steps
Test Conditions:	EnableAll
Test Objectives:	EnableAll
Model Coverage Objectives:	MCDC
Maximum Processing Time:	300s
Block Replacement:	off
Parameters Analysis:	off
Save Data:	on
Save Harness:	on
Save Report:	on

Constraints

Design Min Max Constraints

Name	**Design Min Max Constraint**
WheelSensorMax	[211, 211]
WheelSensorMin	[41, 41]

Name	Design Min Max Constraint
JoystickResolution	[20000, 20000]
WheelSensorData	[41, 211]
MinSteerAngle	[-10000, -10000]

Kapitel 3. Test Objectives Status

Innehållsförteckning

Objectives Satisfied

Simulink Design Verifier found test cases that exercise these test objectives.

#	Type	Model Item	Description	Test Case
1	Test objective	Test Objective1	Objective: [-10000, 10060]	1

Kapitel 4. Model Items

Innehållsförteckning

This section presents, for each object in the model defining coverage objectives, the list of objectives and their individual status at the end of the analysis. It should match the coverage report obtained from running the generated test suite on the model, either from the harness model or by using the sldvruntests command.

Test Objective1

View

#:	Type	Description	Status	Test Case
1	Test objective	Objective: [-10000, 10060]	Satisfied	1

Kapitel 5. Test Cases

Innehållsförteckning

This section contains detailed information about each generated test case.

Test Case 1

Summary

Length:	0 Seconds (1 sample periods)
Objective Count:	1

Objectives

Step	Time	Model Item	Objectives
1	0	Test Objective1	Objective: [-10000, 10060]

Generated Input Data

Time	**0**
Step	**1**
SteeringAngle_Joystick	-
WheelSensorData	128
WheelSensorMax	211
WheelSensorMin	41
MinSteerAngle	-10000
JoystickResolution	20000

Expected Output Inputs that do not affect the test objectives (- in the table above) were given random values.

Time	**0**
Step	**1**
SteeringAngle	-18220
ForceFeedbackToFussApp	266

c) For Property Proving

Simulink Design Verifier Report
SteeringWheelSWC
AjayN
04-Aug-2011 10:20:54

Innehållsförteckning

Kapitel 1. Summary

Analysis Information

Model:	SteeringWheelSWC

Mode:	PropertyProving
Status:	Completed normally
Analysis Time:	1s

Objectives Status

Number of Objectives: 1
Objectives Proven Valid: 1

Kapitel 2. Analysis Information

Innehållsförteckning

Model Information

File:	SteeringWheelSWC
Version:	1.110
Time Stamp:	Thu Aug 04 10:20:45 2011
Author:	AjayN

Analysis Options

Mode:	PropertyProving
Proving Strategy:	ProveWithViolationDetection
Maximum Violation Steps:	20 time steps
Proof Assumptions:	UseLocalSettings
Test Objectives:	EnableAll
Assertions:	UseLocalSettings
Maximum Processing Time:	300s
Block Replacement:	off
Parameters Analysis:	off
Save Data:	on
Save Harness:	on
Save Report:	on

Constraints

Analysis Constraints

Name	**Analysis Constraint**
Assumption	[41, 211]

Design Min Max Constraints

Name	**Design Min Max Constraint**
WheelSensorMax	[211, 211]
WheelSensorMin	[41, 41]
JoystickResolution	[20000, 20000]
WheelSensorData	[41, 211]
MinSteerAngle	[-10000, -10000]

Kapitel 3. Proof Objectives Status

Innehållsförteckning

Objectives Proven Valid

#	**Type**	**Model Item**	**Description**	**Counterexample**
1	Proof objective	Proof Objective	Objective: [-10000, 10060]	n/a

Kapitel 4. Properties

Innehållsförteckning

Proof Objective

Summary

Model Item:	Proof Objective
Property:	Objective: [-10000, 10060]
Status:	Proven valid

Front Wheel SWC

a)Detect Design Errors

Simulink Design Verifier Report
FrontWheelSWC
AjayN
04-Aug-2011 09:22:22

Innehållsförteckning

Kapitel 1. Summary

Analysis Information

Model:	FrontWheelSWC
Mode:	DesignErrorDetection
Status:	Completed normally
Analysis Time:	46s

Objectives Status

Number of Objectives: 6
Objectives Proven Valid: 6

Kapitel 2. Analysis Information

Innehållsförteckning

Model Information

File:	FrontWheelSWC
Version:	1.97

Time Stamp:	Tue Aug 02 09:05:44 2011
Author:	AjayN

Analysis Options

Mode:	DesignErrorDetection
Detect integer overflow:	on
Detect division by zero:	on
Maximum Processing Time:	300s
Block Replacement:	off
Parameters Analysis:	off
Save Data:	on
Save Harness:	off
Save Report:	off

Constraints

Design Min Max Constraints

Name	**Design Min Max Constraint**
SteeringAngle	[-10000, 10000]
WheelSensorData_in	[41, 211]
WheelSensorMax_in	[211, 211]
WheelSensorMin_in	[41, 41]

Kapitel 3. Design Error Detection Objectives Status

Innehållsförteckning

Objectives Proven Valid

#	**Type**	**Model Item**	**Description**	**Test Case**
11	Division by zero	FrontWheelSWC/Microcontroller_Resolution/TransferFunction/SteeringDataToTimeConvertion	Division by zero	n/a
12	Overflow	FrontWheelSWC/Microcontroller_Resolution/TransferFunction/SteeringDataToTimeConvertion	Overflow	n/a
14	Overflow	FrontWheelSWC/Microcontroller_Resolution/ConversionFactor/SteeringPosition	Overflow	n/a
16	Division by zero	FrontWheelSWC/Microcontroller_Resolution/ConversionFactor/DownScaling	Division by zero	n/a
17	Overflow	FrontWheelSWC/Microcontroller_Resolution/ConversionFactor/DownScaling	Overflow	n/a
19	Overflow	FrontWheelSWC/Microcontroller_Resolution/ConversionFactor/Add	Overflow	n/a

Kapitel 4. Signal Bounds

Signal	**Bounds**
FrontWheelSWC/Microcontroller_Resolution/MinTimeDuration- outport 1	[-32768, 32767]
FrontWheelSWC/Microcontroller_Resolution/MaxTimeDuration- outport 1	[-32768, 32767]
FrontWheelSWC/Microcontroller_Resolution/TransferFunction/TotalTime- outport 1	[-32768, 32767]
FrontWheelSWC/Microcontroller_Resolution/TransferFunction/Avg- outport 1	[-128, 127]
FrontWheelSWC/Microcontroller_Resolution/TransferFunction/AvgTime - outport 1	[-32768, 32767]
FrontWheelSWC/Microcontroller_Resolution/TransferFunction/TimeResolution- outport 1	[-32768, 32767]
FrontWheelSWC/Microcontroller_Resolution/MaxSteeringAngle- outport 1	[-32768, 32767]
FrontWheelSWC/Microcontroller_Resolution/TransferFunction/UpScaling- outport 1	[-2147483648, 2147483647]
FrontWheelSWC/Microcontroller_Resolution/MinSteeringAngle- outport 1	[-32768, 32767]
FrontWheelSWC/Microcontroller_Resolution/TransferFunction/SteeringResolution- outport 1	[-32768, 32767]
FrontWheelSWC/Microcontroller_Resolution/TransferFunction/SteeringDataToTimeConvertion- outport 1	[-32768, 32767]
FrontWheelSWC/Microcontroller_Resolution/ConversionFactor/SteeringPosition- outport 1	[-4500000, 4500000]
FrontWheelSWC/Microcontroller_Resolution/ConversionFactor/DownScaling- outport 1	[-451, 450]
FrontWheelSWC/Microcontroller_Resolution/ConversionFactor/Add- outport 1	[1499, 2400]
WheelSensorData- outport 1	[41, 211]
WheelSensorMax- outport 1	211
WheelSensorMin- outport 1	41
MinSteerAngle- outport 1	-10000
JoystickResolution- outport 1	20000

b) Test Generations

Simulink Design Verifier Report
FrontWheelSWC
AjayN
04-Aug-2011 11:08:11

Innehållsförteckning

Kapitel 1. Summary

Analysis Information

Model:	FrontWheelSWC
Mode:	TestGeneration
Status:	Completed normally
Analysis Time:	3s

Objectives Status

Number of Objectives:	**1**
Objectives Satisfied:	1

Kapitel 2. Analysis Information

Innehållsförteckning

Model Information

File:	FrontWheelSWC
Version:	1.102
Time Stamp:	Thu Aug 04 11:07:57 2011
Author:	AjayN

Analysis Options

Mode:	TestGeneration
Test Suite Optimization:	CombinedObjectives
Maximum Testcase Steps:	500 time steps
Test Conditions:	EnableAll
Test Objectives:	EnableAll

Model Coverage Objectives:	MCDC
Maximum Processing Time:	300s
Block Replacement:	off
Parameters Analysis:	off
Save Data:	on
Save Harness:	on
Save Report:	on

Constraints

Design Min Max Constraints

Name	**Design Min Max Constraint**
SteeringAngle	[-10000, 10000]
WheelSensorData_in	[41, 211]
WheelSensorMax_in	[211, 211]
WheelSensorMin_in	[41, 41]

Kapitel 3. Test Objectives Status

Innehållsförteckning

Objectives Satisfied

Simulink Design Verifier found test cases that exercise these test objectives.

#	Type	Model Item	Description	Test Case
1	Test objective	Test Objective	Objective: [1500, 2400]	1

Kapitel 4. Model Items

Innehållsförteckning

This section presents, for each object in the model defining coverage objectives, the list of objectives and their individual status at the end of the analysis. It should match the coverage report obtained from running the generated test suite on the model, either from the harness model or by using the sldvruntests command.

Test Objective

View

#:	Type	Description	Status	Test Case
1	Test objective	Objective: [1500, 2400]	Satisfied	1

Kapitel 5. Test Cases

Innehållsförteckning

This section contains detailed information about each generated test case.

Test Case 1

Summary

Length:	0 Seconds (1 sample periods)
Objective Count:	1

Objectives

Step	Time	Model Item	Objectives
1	0	Test Objective	Objective: [1500, 2400]

Generated Input Data

Time	**0**
Step	**1**
SteeringAngle	-156
WheelSensorData_in	121
WheelSensorMin_in	41
WheelSensorMax_in	211

c) For Property Proving

Simulink Design Verifier Report
FrontWheelSWC
AjayN
04-Aug-2011 10:38:27

Innehållsförteckning

Kapitel 1. Summary

Analysis Information

Model:	FrontWheelSWC
Mode:	PropertyProving
Status:	Completed normally
Analysis Time:	7s

Objectives Status

Number of Objectives: 1
Objectives Proven Valid: 1

Kapitel 2. Analysis Information

Innehållsförteckning

Model Information

File:	FrontWheelSWC
Version:	1.101
Time Stamp:	Thu Aug 04 10:38:13 2011
Author:	AjayN

Analysis Options

Mode:	PropertyProving
Proving Strategy:	ProveWithViolationDetection
Maximum Violation Steps:	20 time steps
Proof Assumptions:	EnableAll
Test Objectives:	EnableAll
Assertions:	EnableAll
Maximum Processing Time:	300s
Block Replacement:	off
Parameters Analysis:	off
Save Data:	on
Save Harness:	on
Save Report:	on

Constraints

Analysis Constraints

Name	Analysis Constraint
Assumption	[-10000, 10000]

Design Min Max Constraints

Name	Design Min Max Constraint
SteeringAngle	[-10000, 10000]

Name	Design Min Max Constraint
WheelSensorData_in	[41, 211]
WheelSensorMax_in	[211, 211]
WheelSensorMin_in	[41, 41]

Kapitel 3. Proof Objectives Status

Innehållsförteckning

Objectives Proven Valid

#	Type	Model Item	Description	Counterexample
1	Proof objective	Proof Objective	Objective: [1500, 2400]	n/a

Kapitel 4. Properties

Innehållsförteckning

Proof Objective

Summary

Model Item:	Proof Objective
Property:	Objective: [1500, 2400]
Status:	Proven valid

APPENDIX D – Software and Hardware Specifications

This chapter explains concisely the different software applications and hardware products that are used in this project.

Software Specifications

Development Tools

a) MATLAB/Simulink

This project is based on model based design for which MATLAB/Simulink was chosen as developing tool to design Steery application.

b) Microsoft Visual Studio

To tweak the generated code of the Steery application or to link that generated code with the FUSS application, Microsoft Visual Studio was chosen.

Application Software

a) Live for speed

In this project to test the performance of the Steer-By-Wire application a simulator environment, *Live for Speed* is used. Its appearance as a real environment with a good visual quality and tranquil control over its interfaces and vehicle actions complemented the project requirements. In addition, this simulator has a virtuously active community with continuous updates to the product.

Live for Speed uses InSim to communicate with other external interfaces. This interface supports various data to be sent to any external software from simulator and vice-versa [10]. Any information from the external software or applications was fed to the simulator through PPJoy, a virtual joystick.

b) FUSS Application

Fuss application is a Microsoft VC++ project developed by Mecel AB. This application controls three main applications: *Live for speed, DirectInput and Virtual Joystick*.

Direct Input

DirectInput is an add-on to Windows Application Programming Interface (Windows API) that can control the I/O ports of the input device connected externally, in this context it's a game control devices [11]. In this project, we use this application to retrieve data from the external Logitech Racing Wheel and also to introduce the force feedback from the Steery model.

Virtual Joystick

A virtual joystick is used to manipulate the data of the external devices without using hardware. Since it is detectable by the operating system, using IOCTL (input/output control system calls) function calls, it is easy to implement. In this case, PPJoy is used as a virtual joystick, in order to update the front wheel position of the vehicle in the simulator (*Live for Speed* game) [12].

Hardware Specifications

This section explains briefly various significant hardware components that are used in this project work.

Logitech G25 Racing Wheel

Logitech G25 Racing wheel, as in Figure 15, contains three main parts: Steering Wheel, Pedals and Shifter. In the Simulator only Steering Wheel and Pedals are used, since the game is programmed as gears shifts automatically. G25 Racing Wheel supports 810 degrees of wheel rotation with a resolution of 20,000 steering values from -10,000 to 10,000. This resolution can be altered according to the application, for better visualization. In this project; these steering values are considered as steering input. Pedals are used to throttle, brake and clutch.

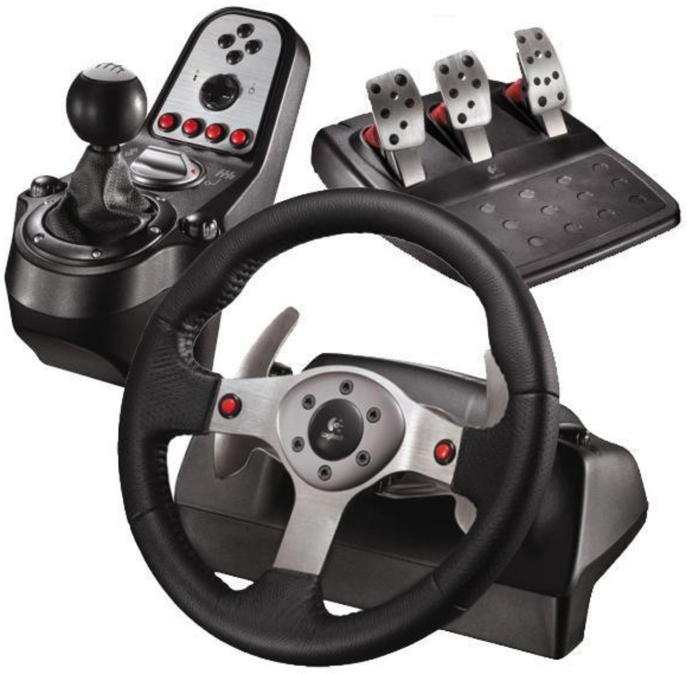


Figure 15: Logitech G25 Racing Wheel with Force Feedback
Source: http://www.hellas.rs/index.php?action=opis&q=5047

Microcontroller

During the first stage of the project (before using Tricore Board), in order to convert the steering wheel data into a control signal for the front wheels, a Parallax Basic Stamp Microcontroller, as in Figure 16, is used [13]. This microcontroller converts any input voltage signal into a timing sequence (PWM). Then, this output timing sequence is given to a servomotor that controls the front wheel position accordingly. Later on, while using the Tricore Board, an inbuilt microcontroller will be used for the same application, such that Basic Stamp Microcontroller can be detached.

Figure 16: BS2e Parallax Basic Stamp
Source:http://www.parallax.com/Store/Microcontrollers/BASICStampModules/

Servo Motor

Figure 17 shows, a Hitec 55.2g HS-645MG High Torque Metal Gear High Performance RC Servo motor. It has high performance servos for radio control of model demo cars. It is controlled by positive Pulse Width Modulated (PWM) train pulse that comes from microcontroller. Its operating frequency is 50 Hz, with 1.0 ms pulse gives full scale in one direction and 2.0 ms pulse gives full scale in other direction, with 1.5 ms as straight position [14]. In other words, the servomotor has a range of 1.0 ms to 2.0 ms that controls the front wheels from one extreme direction to other.

Figure 17: HS-645MG - Three Pole Servo
Source: http://www.servocity.com/html/hs-645mg_ultra_torque.html

Front Wheel Prototype

Figure 18 shows, a front wheel rack system designed and contributed by Torgny Carlsson, a Mecel employee. This prototype is activated and controlled using two servo motors as shown in Figure 17, which is shown in the figure below. It has three potentiometer sensors that detect the position of the front wheels and send it to the FUSS application via the Velleman Board.

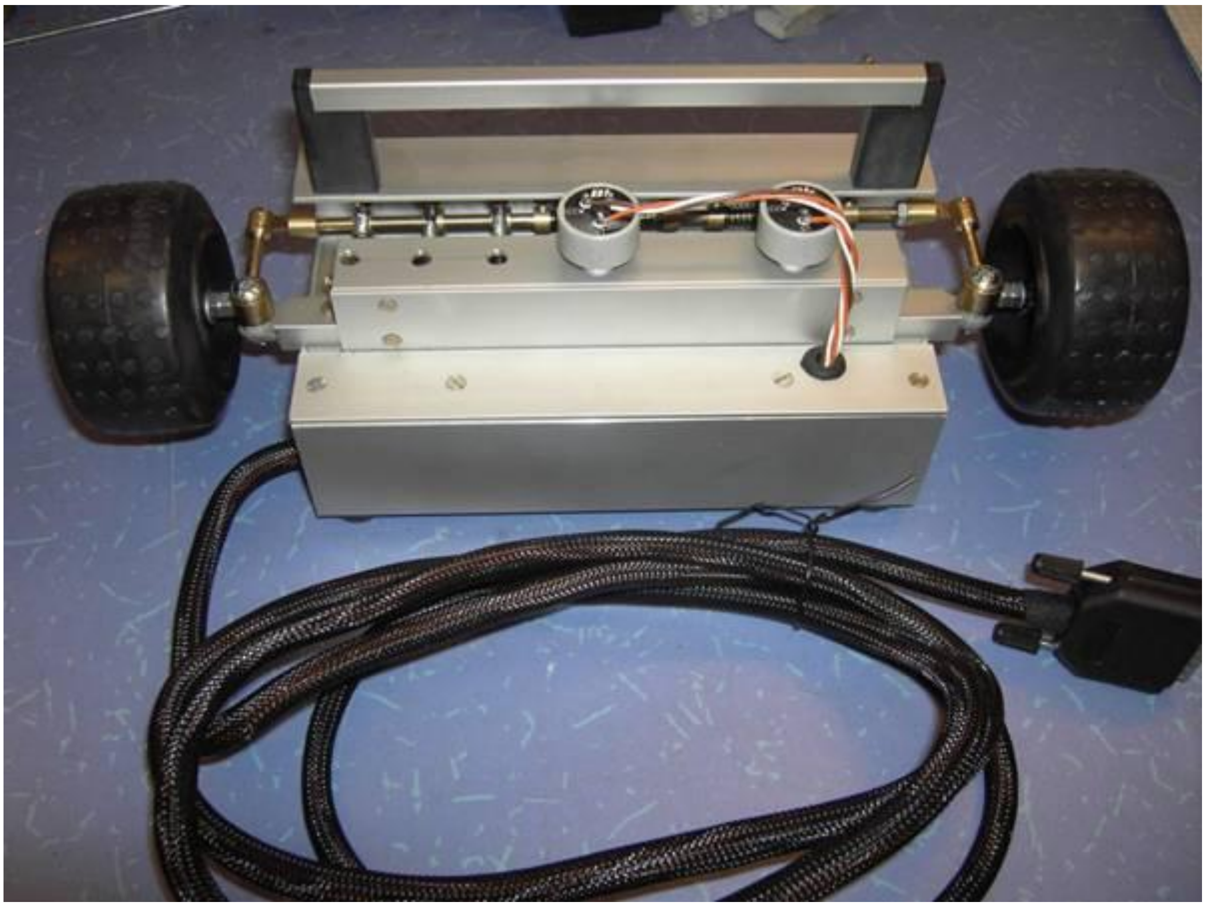


Figure 18: Front Wheel Rack / Prototype

Velleman Board

Shown in Figure 19, is a Velleman USB interface Board, which contains five digital inputs and two analog inputs with attenuation and amplification options. It has eight digital output switches and two analog outputs: one with 0 to 5 V and the other with PWM as output [15]. This board is used to detect the values of the potentiometer sensors that are located on the front wheel prototype. It will quantize or digitize the continuous analog signal that comes from the wheel sensors (potentiometer sensors) and give those discrete values to the FUSS application.

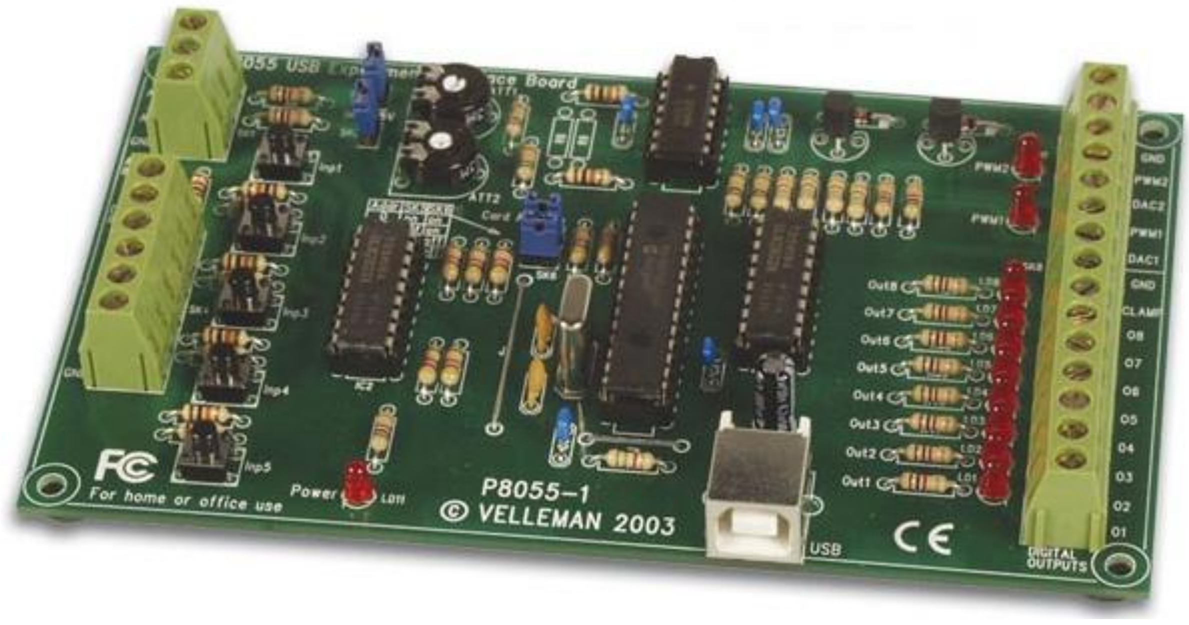


Figure 19: K8055 Interface Board – Velleman
Source: http://www.velleman.eu/distributor/products/view/?country=be&lang=en&id=351346

Infineon TriBoard

The Infineon TriBoard, as shown in Figure 20, has a 32 - bit architecture with 180 MHz CPU clock frequency and 4MB of embedded program on-chip memory along with 4 kB of data cache and 16 kB of instruction cache [16]. In this project this board is part of the Steery sub-system hardware. In other words, the Steery application will run on the TriBoard, which receives an input steering angle through one of the CAN interface. This input is converted into a timing sequence to generate the PWM signal using internal clock on the board. This generated PWM is then fed back to the servo motors that move the front wheels accordingly.

Figure 20: Starter Kit TC 1797 TriBoard
Source: http://www.infineon.com/cms/en/product/microcontrollers/

USB to CAN converter

In common, since CAN interfaces are not available in laptops, a Kvaser Leaf Professional CAN to USB adapter, as in Figure 21, is used to communicate with CAN bus via USB connection. The speed of this CAN bus ranges from 5 Kbps to 1 Mbps. Kvaser delivers all essential drivers and CAN library SDK for creating specific applications [17]. In this project the FUSS application and the Steery model were connected using this Kvaser Leaf Professional USB-to-CAN converter.

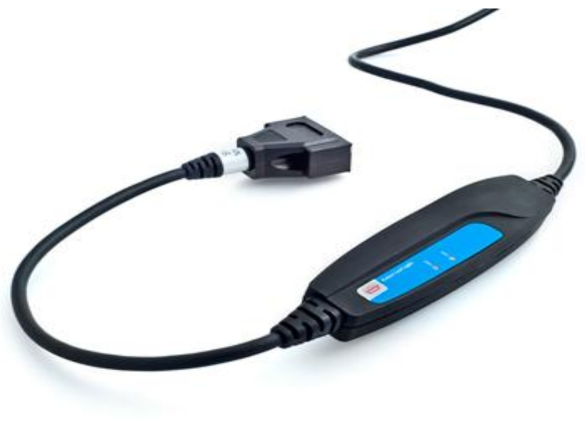


Figure 21: Kvaser Leaf Professional USB to CAN converter
Source:http://www.kvaser.com/index.php?option=com_php&Itemid=258&eaninput=7330130002432