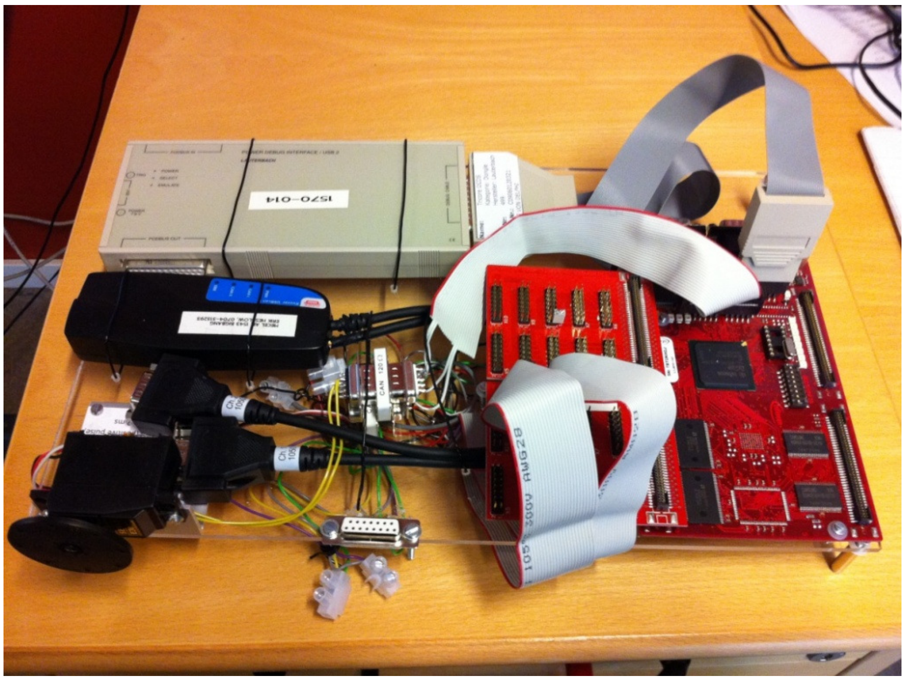
CHALMERS



Methodology and Design Pattern for Converting AUTOSAR Simulink Models from SIL to HIL

Master of Science Thesis at Computer Science and Engineering

Olsson Erik

Pham Daniel Trung

Chalmers University of Technology

University of Gothenburg

Department of Computer Science and Engineering

Göteborg, Sweden, August 2011

Methodology and Design Pattern for Converting AUTOSAR Simulink Models from SIL to HIL

DANIEL PHAM

ERIK OLSSON

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

Department of Computer Science and Engineering
Göteborg, Sweden August 2011.

Abstract

This project is a part of the Architecture For Future Electric-vehicles (AFFE) project. AFFE project is a research project funded by VINNOVA to bring out the next generation electric vehicles based on AUTomotive Open System Architecture (AUTOSAR). Modern vehicles are containing many electronic devices therefor the system complexity is also growing. Therefor this thesis project aims to find a methodology to develop software applications with instantiation possibility using Model-based Design environment, then do code generation and finally integrate into hardware without modifying the software applications configurations between different phases. For this purpose Simulink from Mathworks has been used as a Model Based Development tool for developing software applications, simulating and generating AUTOSAR compliant code. Mecel Picea Workbench has been used to make configuration for simulation in PC environment and hardware platform. The project ends with analysis of compatibility of different tools and configuration possibilities in different phases. The outcome of this project shows that the AUTOSAR standard is young and therefor compliant tools have limitations.

Preface

This master thesis proposal was made by **Mecel AB** where they want to develop a methodology using Model Based Development for developing AUTOSAR compliant software applications. We want to thank Mecel for the opportunity to do the thesis work and be a part of the AFFE project with a great team.

We want to thank **Erik Hesslow** and **Marcus Johansson** at Mecel who help us out during the thesis work. Erik was our supervisor whom gave us all necessities and guided us through the project. Marcus supported us with technical expertise and helped us to configure the basic platform for the system using Mecel Picea Workbench.

We also want to thank our examiner **Roger Johansson** at Chalmers. He supported us through the project and gave feedback on the planning and the thesis report.

Table of Contents

List of Abbreviations

ADC	Analog to Digital Conversion
AFFE	Architecture For Future Electric Vehicles
BSW	Basic Software
CAN	Control Area Network
DLL	Dynamic Link Library
ECU	Electronic Control Unit
GUI	Graphical User Interface
HIL	Hardware in the loop
IDE	Integrated Development Environment
MBD	Model Based Development
MCAL	Micro Controller Abstraction Layer
MCU	Micro Controller
MIL	Model in the loop
RTE	Runtime Environment
PWM	Pulse With Modulation
PSP	Pilot Support Package
SDK	Software Development Kit
SIL	Software in the loop
SW-C	Software Component
SW-CD	Software Component Description
TCP/IP	Transmission Control Protocol / Internet Protocol
USB	Universal Serial Bus
VFB	Virtual Function Bus
WPF	Windows Presentation Foundation
XML	Extensible Markup Language

1 Introduction

This chapter introduces the thesis with the background, scope and objective of the project.

1.1 Background

Vehicles today have a large impact on the global environment and the automotive industry wants to reduce emissions [1]. Therefore the automotive industry is forced to make a shift from the traditional designed power train solutions to more environment friendly, efficient and long lasting solutions. Mecel is part of the AFFE (Architecture For Future Electric vehicles) project funded by VINNOVA, which aims to develop the next generation of hybrid vehicles. By developing a new architectural design it is possible to improve the hybrid technology to be more efficient.

A new standardized automotive software architecture called AUTOSAR (AUTomotive Open System Architecture) has been developed by a joint group of automotive manufacturers and suppliers. The concept of AUTOSAR is to decouple the functionality from the hardware and to simplify communication between applications [2].

Furthermore simulation of the system has become increasingly important as the system complexity grows, the earlier simulations are done the earlier are errors detected and the cost associated with correcting the errors are reduced significantly [3]. By introducing different phases of simulation; model-in-the-loop (MIL), software-in-the-loop (SIL) and hardware-in-the-loop (HIL) the functionality of the system can be verified in each phase and any errors can be detected as early as possible. The software developed should be easy to move between the different phases in order to reduce the cost of modifying an application.

Model based development (MBD) tools such as Simulink can be used to model the functionality of the system and to generate AUTOSAR compliant code and descriptions from these models. However there are still uncertainties on how well different tools support the AUTOSAR standard and the compatibility within the tool chain.

1.2 Purposes

The purpose of this thesis is to find a methodology on how to develop software components (SW-C) in Simulink and how to go from MIL to SIL to HIL simulations of the system.

The purpose is also to investigate how SW-Cs can be instantiated multiple times and how each instantiation can be configured to provide a unique behaviour when developing the SW-C in Simulink.

1.3 Problem

The first part of the main problem of this thesis is to investigate how a SW-C developed in Simulink can be;

- Instantiated multiple times
- Simulated in the MIL phase
- Generated with AUTOSAR compliant code

The second part of the problem consists of finding what is necessary to configure in order to go from one phase to the next, i.e. from MIL to SIL or from SIL to HIL, without the need to change the behaviour of the models used during the MIL phase or manually modify the generated code in the SIL and HIL phases.

1.4 Delimitations

The software components and models developed should be kept very simple. Since this project focuses on model based design and methodology we will not elaborate on parts of AUTOSAR that is not related to software components.

Authoring tools will be used but will not be covered since it is out of the scope for this project.

In automotive system CAN is often used for communication between ECUs, which will be used in this project and FlexRay, LIN communication interfaces will be left out.

2 Theory

In this chapter the theory and technical background regarding AUTOSAR standard and MBD will be described.

2.1 AUTOSAR

AUTOSAR is a common agreement of a software architecture standard developed by the automotive industry; manufacturers, suppliers and tool developers. The number of electronic systems in vehicles increases each year and the industry as a whole agreed to address the issue of developing and maintaining larger systems. Previously no standard existed to allow different parts of a system to efficiently be reused and developed separately. [4]

The goals of AUTOSAR, easier development and verification, reuse and maintenance of software is achieved by adding additional layers on top of the hardware. These layers provides an interface to the applications that can be used to send messages to other applications, read sensor values or write actuator values. This allows the software to be developed independently of the hardware and can more easily be reused. Each application also specifies its own interface, according to the AUTOSAR standard, which simplifies the integration of applications developed by different suppliers. Also AUTOSAR makes it easier to move the applications between different ECUs. [5]

The AUTOSAR standard consists of many specifications and concepts, in this chapter the basic concepts of AUTOSAR is covered. To create software applications according to the AUTOSAR standard it is necessary to use an AUTOSAR Authoring Tools (e.g. Mecel Picea).

2.2 AUTOSAR Concepts

There are a few key concepts in AUTOSAR [6]

- Software Component (SW-C)
- AUTOSAR Interface
- Virtual Function Bus (VFB)
- Runtime environment (RTE)
- Basic Software (BSW)

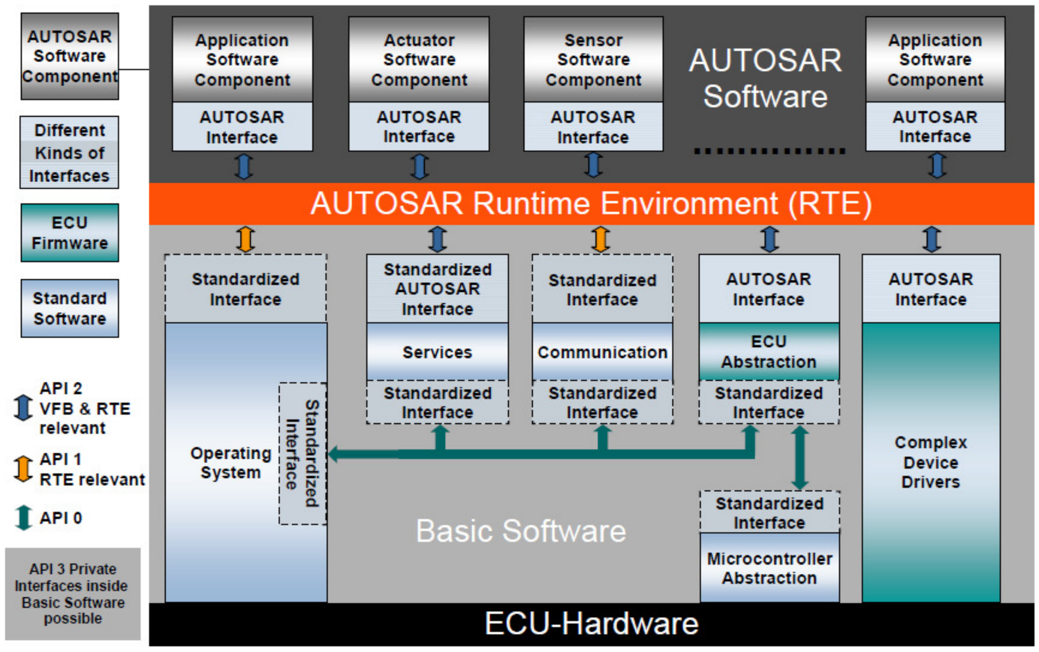

AUTOSAR Software Component
Different Kinds of Interfaces
ECU Firmware
Standard Software
API 2 VFB & RTE relevant
API 1 RTE relevant
API 0
API 3 Private Interfaces inside Basic Software possible
Application Software Component
Actuator Software Component
Sensor Software Component
AUTOSAR Software
AUTOSAR Interface
AUTOSAR Runtime Environment (RTE)
Standardized Interface
Standardized AUTOSAR Interface
Services
Communication
ECU Abstraction
Operating System
Microcontroller Abstraction
Complex Device Drivers
Basic Software
ECU-Hardware

Figure 1 – An overview of AUTOSAR and the usage of interfaces

2.2.1 Software Component

A SW-C in AUTOSAR is an application that performs some duty (e.g. a brake controller) with a well-defined interface and internal behaviour. The interface is the connection point between different SW-Cs and there are several port types available in AUTOSAR. Internally the SW-C consists of one or more runnables that can be executed to perform a task.

SW-Cs that read the input of sensors or outputs to actuators needs to be located on the same ECU as the sensor or actuator in order to use them. The communication between sensor/actuator and SW-C is handled by drivers loaded onto the ECU and because the implementation details are very hardware dependent only the interface of the drivers are specified by AUTOSAR.

A SW-C can also be a composition of other SW-Cs (e.g. the vehicle itself can be seen as a SW-C composition of all SW-Cs in the vehicle). A SW-C composition can be distributed on many different ECUs but a SW-C that is not a composition must be located on a single ECU.

2.2.1.1 Runnables

As mentioned before a SW-C consists of one or more runnables which are executed based on events the runnable receives from the RTE. The event for a periodically executed runnable is a TimingEvent but there are also other events, ex. DataReceivedEvent where incoming data triggers the execution of a runnable (e.g. used for servers). Each runnable in a SW-C has its own triggers and can be triggered by different types of events.

Data can be shared between the runnables of a SW-C by using interrunnable variables. These variables are visible to all runnables within the same SW-C and can be accessed implicit or explicit, same semantics as the Sender/Receiver interface applies.

2.2.2 AUTOSAR Interface

To achieve scalability and transferability of SW-Cs between different vehicle platforms AUTOSAR standardized the communication interface used by the components, the use of standardized interfaces can be seen in Figure 1 – An overview of AUTOSAR and the usage of interfaces. There are two different port types in AUTOSAR, provide port (P-Port) and receive port (R-Port). These ports implements one of two possible interfaces, either Sender/Receiver or Client/Server. The Sender/Receiver interface is used for one way communication while the Client/Server interface is used to perform an operation and return the result to the caller. [6]

A P-port implementing Sender/Receiver provides a signal whereas an R-Port implementing the Sender/Receiver interface requires a signal. When implementing the Client/Server interface a P-Port provides a service to a client and an R-Port requires a service.

2.2.2.1 Sender/Receiver interface

To determine if a P-port and an R-port, both implementing the Sender/Receiver interface, are compatible the R-Port must accept a subset of the data elements provided by the connected P-Port. E.g. if the P-Port provides an 8-bit integer and a 16-bit integer then any R-Port connected to it must accept either one or both of the 8-bit and 16-bit integer. The data elements accepted by the R-Port must also have a name that matches the data elements of the P-Port. If these conditions are not meet the ports are not compatible.

The Sender/Receiver interface allows for queued and unqueued communication, this means that data sent to a receiver can either be queued, handled one by one, or unqueued, the most recent data received is used. Also the communication can be of two different modes, explicit or implicit. Explicit indicates that the data should be sent immediately and the receiver always reads the latest data while implicit means that the data will be buffered when the runnable starts and it will use the same data throughout its execution. When implicit is used queued data is not allowed.

2.2.2.2 Client/Server interface

Ports implementing the Client/Server interface are compatible if the server P-Port provides all the operations required by an R-Port with matching names and arguments (name, direction and data type). The call to a server can be either synchronous, client will block when calling, or asynchronous, client will not block when calling.

2.2.3 Virtual Function Bus

The Virtual Function Bus (VFB) is an abstraction of the communication between SW-Cs. It connects different SW-Cs without knowledge of their actual assignment to ECUs as is done in Figure 2, this allows for simpler transferring and integration of SW-Cs. [4]

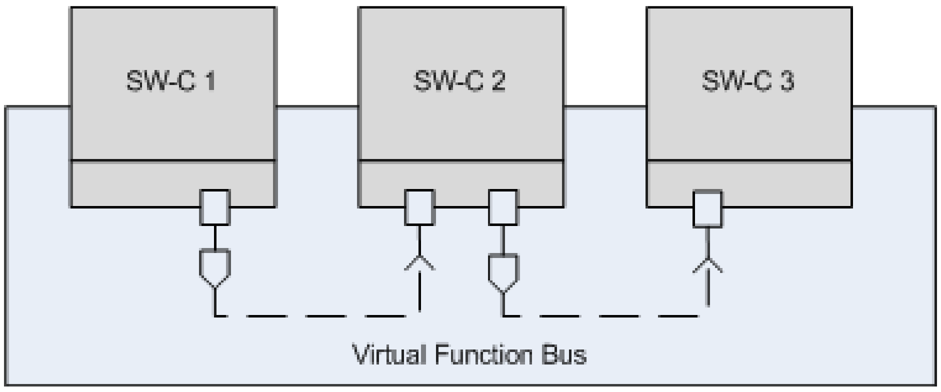


Figure 2 – The Virtual Function Bus abstracts communication between Software Components

A VFB simulator can be used to simulate the system in a PC-environment and perform system testing without the need of ECUs to be available.

2.2.4 Runtime environment

The RTE implements the VFB and provides communication between SW-Cs both internally (SW-Cs on the same ECU) and externally (SW-Cs on different ECUs). Also it decouples the SW-C from the underlying hardware and software, making the SW-C independent of the hardware (Figure 3). The RTE is generated for each ECU and implements the necessary parts which the SW-Cs assigned to the ECU requires.

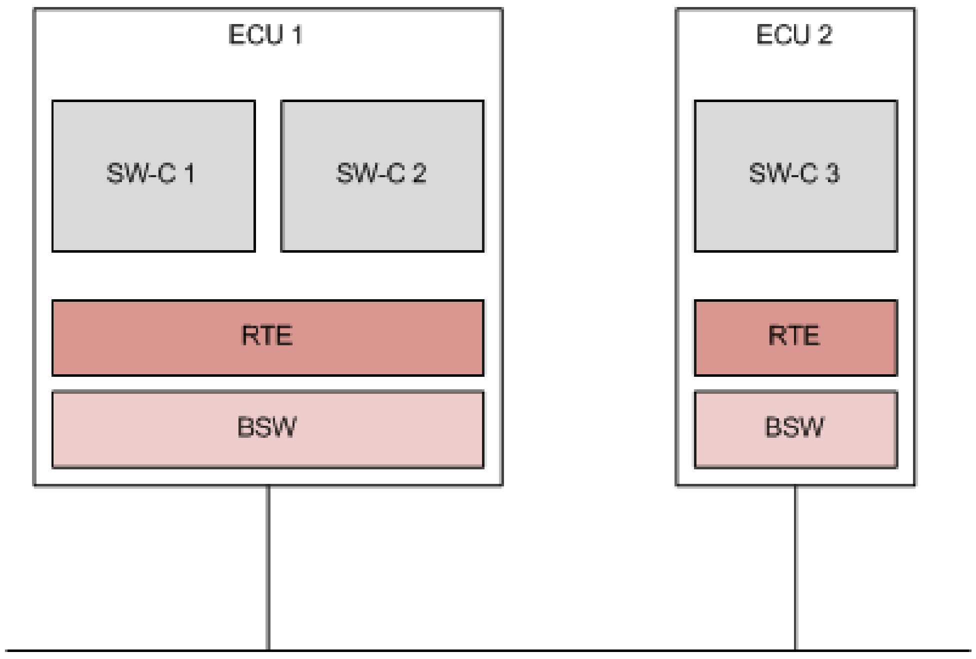


Figure 3 – The RTE layer separates the SW-Cs on an ECU from the BSW layer.

2.2.5 Basic Software

The Basic Software (BSW) is implemented below the RTE layer on each ECU (figure 2). It abstract details of the hardware and allows the RTE to communicate with other ECUs over a bus network (CAN, FlexRay etc.). The BSW is divided into different layers with different responsibilities and it contains both hardware dependent and independent software. E.g. the

Service Layer manages operating system, diagnostic protocols and other services used by the RTE and BSW. There are layers that abstract the details of the microcontroller and ECU layout, ECU Abstraction Layer and Microcontroller Abstraction Layer, and there is a complex driver layer that contains drivers for hardware dependent features that are not covered by the AUTOSAR standard. The implementation of the modules in the complex driver layer are not part of the standard, because it can be very specific implementations with strong timing requirements, but their interfaces are part of the AUTOSAR standard. [4]

The operating system located in the BSW is responsible for scheduling and trigger execution of the runnables in the SW-Cs located on the same ECU.

2.2.6 Multiple instantiation

Multiple instantiation can be done in different ways, either the code of the SW-C is shared among the instantiations and it is required of the SW-C to support re-entrancy, that shared code can be executed simultaneously by different threads, or the code should be duplicated in which case each thread executes its own copy.

In AUTOSAR multiple instantiation of a SW-C is done by code sharing and requires all instances to be located on the same ECU, it is not supported to instantiate a SW-C on multiple ECUs because support for code duplication is not yet included in the AUTOSAR standard [7].

2.2.7 Calibration parameters

Calibration parameters can be used to dynamically control the behaviour of a SW-C, e.g. the parameters of a SW-C can during simulation have different values to find which value is the most suitable. Instances of a SW-C can either share the calibration parameters or have instance specific calibration parameters to make it possible to differentiate the behaviour of multiple instances of the SW-C.

2.2.8 Methodology

Development of SW-Cs can be done using two different workflows, either top-down or bottom-up [8].

In the top-down approach the AUTOSAR system architecture is specified in an authoring tool and each SW-C has its interfaces, enumerations, events specified. These details of a SW-C are called a SW-C description (SW-CD). The SW-CD can be exported and used to specify the skeleton of the SW-C which has to be completed with the implementation details.

The bottom-up approach is used for when there are existing software models that are not AUTOSAR compatible and needs to be modified to be used in an AUTOSAR platform. Then a SW-CD, containing the same information as in the top-down approach, should also be created along with the implementation details for the SW-C.

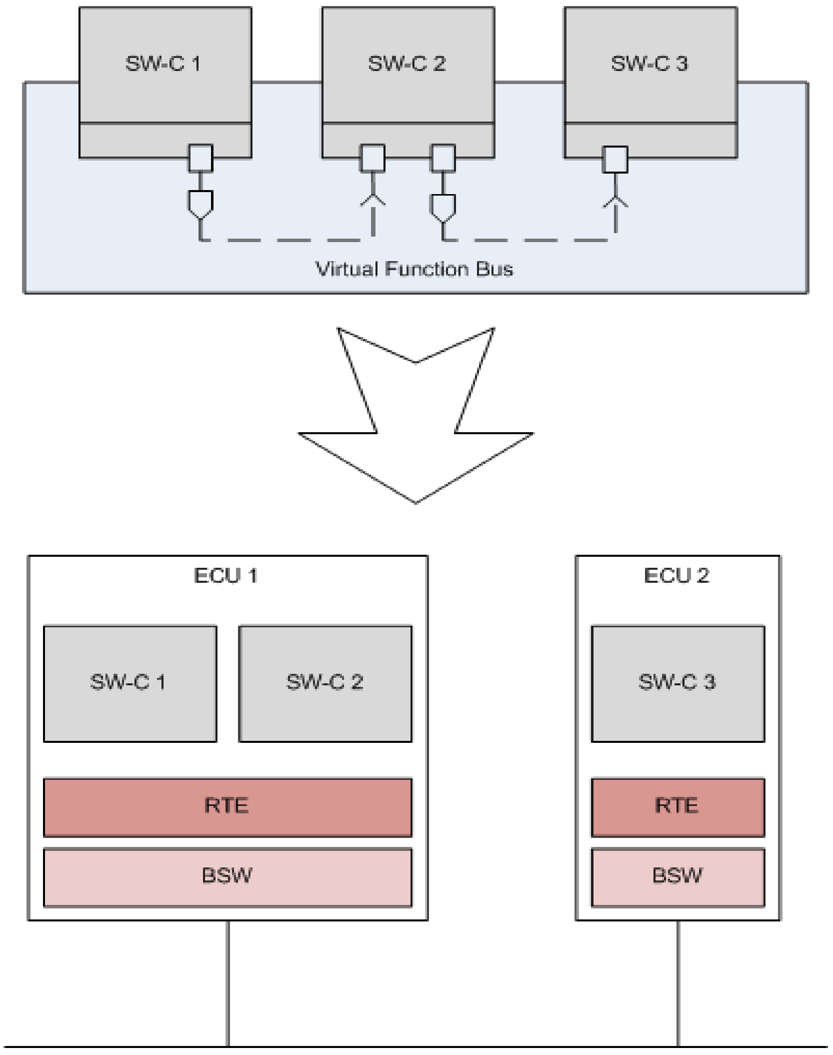


Figure 4 - The mapping of SW-Cs to ECUs

Once the SW-Cs has been specified and implemented a VFB simulator can be used to simulate the system. This early simulation results in errors of the SW-Cs or compatibility errors between SW-Cs are detected earlier when the cost for correcting any error found is still low [8].

The SW-Cs are then mapped to the ECUs in the system as is shown in Figure 4. An ECU description can be derived from all the SW-Cs located on one ECU and it specifies the content of that ECU. Once all SW-Cs are known for one ECU and the configuration of the ECU and hardware drivers is complete the source code can be generated, built and loaded into the hardware.

2.3 Model Based Development

Model based development is a development process where a system model acts as the center point and the system model contains sub-models which contain additional models or implementation details, creating a complete system in one system model. Benefits with this design concept, the system could be developed faster, more cost effective, less error prone and work can be split into separate tasks [8]. Simulation of the integration of different sub-models can be done early, as soon as the implementations of these models are completed.

The development process can be split up into three phases as in Figure 5. These phases are model-in-the-loop, software-in-the-loop and hardware-in-the-loop.

2.3.1 MIL Phase

The MIL phase consists of developing the models as well as simulating the behaviour of the system. This is done in a modelling tool e.g. Simulink. As development progresses more sub-models are completed and more parts of the system can be included in the simulations to verify the functionality of each sub-model. [9]

2.3.2 SIL Phase

In the SIL phase the code which was generated from the models in the MIL phase are simulated. The results of the simulation are compared to the result of the simulation done in the MIL phase, since the functionality of the system should be the same independent of which phase the development process is in [9]. If any errors are detected in the generated code from the MIL phase during the simulations, the developer should return to the MIL phase and correct the erroneous models. [9]

2.3.3 HIL Phase

During the HIL phase the system is integrated to and executed on the target hardware. Again the system is simulated and the functionality compared to that of the previous steps.

With these three phases the system errors can be detected and solved earlier, this decreases the cost of the development process. Further the models provide a better overview of the system even as complexity grows since the details can be abstracted to sub-models. [9]

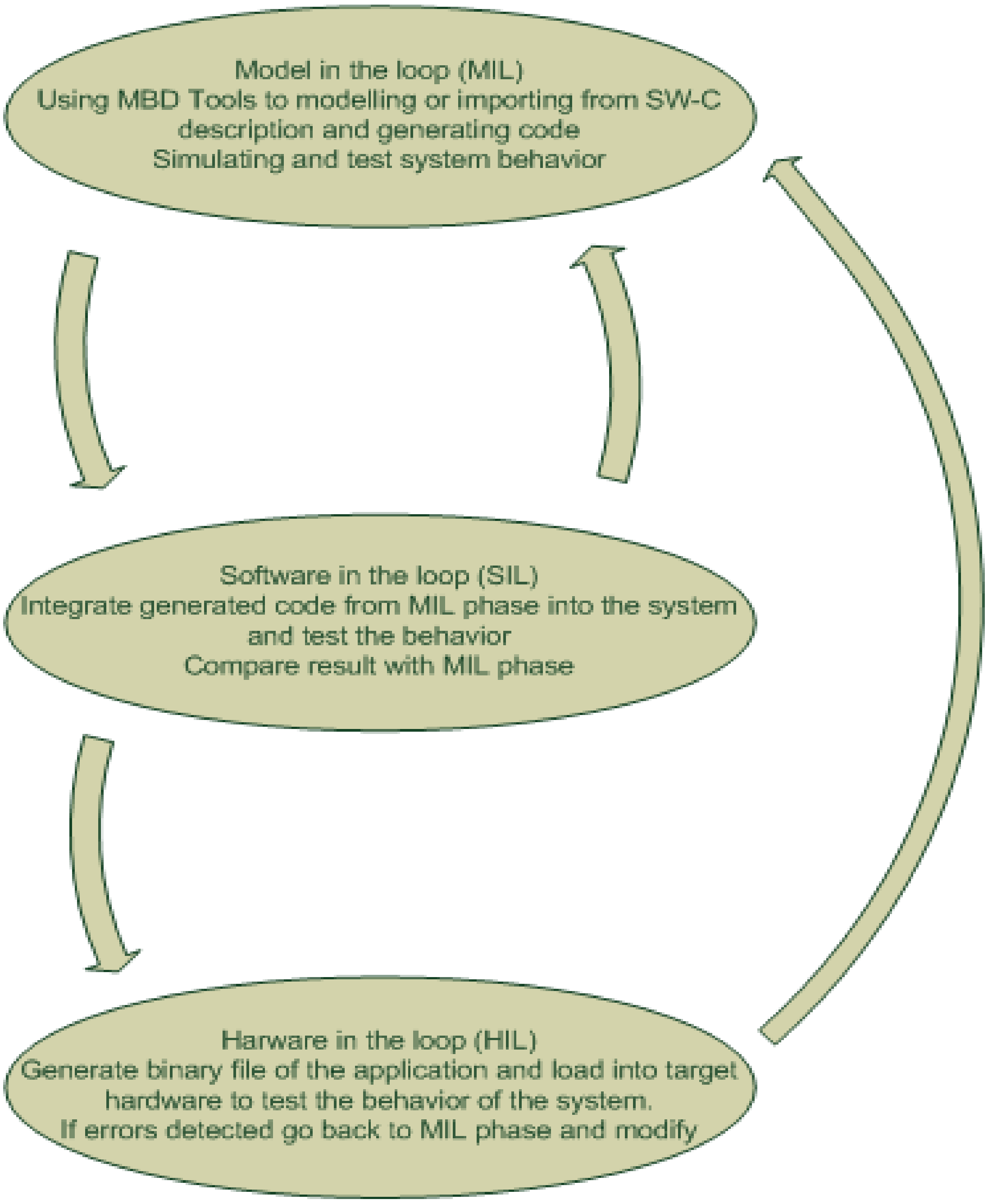

Model in the loop (MIL)
Using MBD Tools to modelling or importing from SW-C description and generating code
Simulating and test system behavior
Software in the loop (SIL)
Integrate generated code from MIL phase into the system and test the behavior
Compare result with MIL phase
Harware in the loop (HIL)
Generate binary file of the application and load into target hardware to test the behavior of the system.
If errors detected go back to MIL phase and modify

Figure 5 - Model Based Development process

3 Developments tools

3.1 Software

3.1.1 MATLAB/Simulink

One of the main tools which have been used in this project is MATLAB/Simulink, version 7.7, from Mathworks. Simulink provides an MBD environment which allows modelling with building blocks. In this project the Embedded Coder 6.0 was used to generate AUTOSAR compliant code from the models and a pilot support package was used for importing SW-CD into the Simulink environment. [10]

3.1.2 Mecel Picea Workbench

Picea Workbench is an AUTOSAR Authoring Tool developed by Mecel. This tool is used for efficient development of AUTOSAR compliant ECUs. With Picea Workbench it is possible to configure, generate, validate and integrate RTE and BSW modules based on AUTOSAR methodology. [11]

3.1.3 Visual C++ Express & Visual C# Express

Visual C++ Express and Visual C# Express are two integrated development environment (IDE) tools for developing and testing software available for free by Microsoft. These tools had been used in the project for developing a simulation and test application and for building/compiling the VFB simulator in the SIL phase.

3.1.4 Tasking

Tasking is an embedded software development tools from Altium Limited. Tasking provides embedded development environment, with C compiler, assembler, debugger and Real Time Operating System (RTOS) for microcontroller development such as Triboard TC1797. This tool was included with the hardware and is used for compiling code for the target hardware in the HIL phase.

3.1.5 Trace32

Trace32 is a set of microprocessor development tools that includes multi processors emulators and disassemblers from German Lauterbach. With Trace32 it is possible to load the binary executable code into the target hardware, debug and run the application. It makes it easier to step through the assembler instructions in the application running on the hardware and tracing if errors occur.

3.2 Hardware

3.2.1 TriBoard TC1797

TriBoard TC1797 (see Figure 6) from Infineon includes a high performance 32bits Tricore - based microcontroller for automotive system.

It has support for communication interfaces such as CAN, FlexRay and also DAP, PWM signals and ADC. Applications can be developed with corresponding tools that was included and loaded to the TriBoard TC1797 and tested with a debugger, such as Trace32 from Lauterbach as mentioned in section 3.1.5.

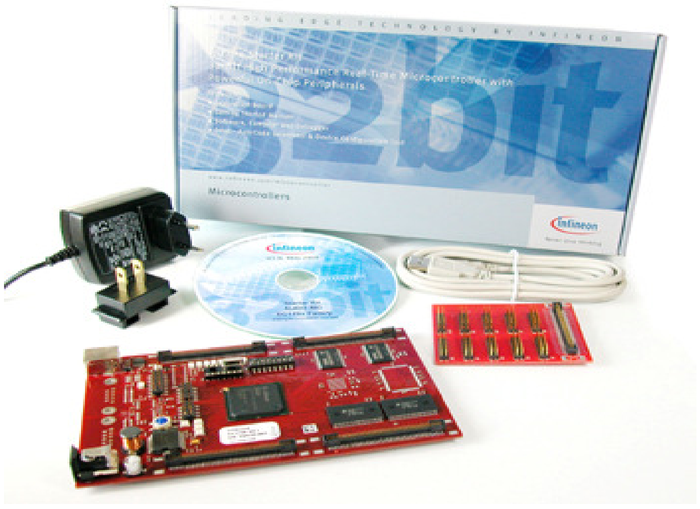


Figure 6 - TriBoard TC1797 from Infineon

3.2.2 Kvaser USB to CAN

To allow for CAN communication between a PC and the TriBoard an adapter, called Kvaser USB to CAN (USBcanII HS/HS) (Figure 7), for converting CAN to USB was used. To be able to use Kvaser USBtoCAN with the software in this project a library called CANLIB SDK were downloaded from Kvaser homepage.

Figure 7 - Kvaser USB to CAN adapter

3.2.3 Lauterbach Debugger

For loading the application in binary code to the target hardware and debugging a Power Debug Interface / USB 2 from Lauterbach were used (Figure 8). It connects to a PC by USB and is used by the Trace32 software mentioned in section 3.1.5 for operating and debugging purposes. This debug interface supports different target processors by changing the debug cable and the software.

Figure 8 - Lauterbach Power Debugger

3.2.4 Wheel Prototype

The wheel prototype used in this project consists of two separate hardware modules, the steering servo and the speed servo.

3.2.4.1 Steering servo

The steering servo consists of two small electric servos for controlling the steering axis of the wheel and is shown in Figure 9. There are three potentiometer sensors which are used to determine the position of the wheel axis. The steering servo is controlled by PWM signals at 50Hz with a duty cycle of 1-2ms. The range 1-2ms duty cycle is to determine how turning angle is going to be. The largest turning angle to the left is equal to 1ms and the opposite applies for the turning angle to the right which is 2ms pulse and straight forward is 1.5ms.

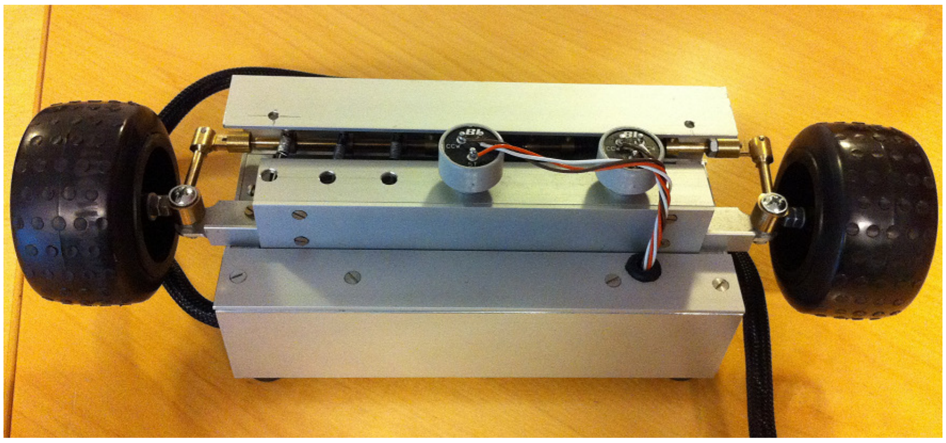


Figure 9 - The steering servo

3.2.4.2 Speed servo

Speed servo (Figure 10) is a prototype in miniature of an independent wheel node in a vehicle. The concept is to make all wheel nodes driveline separated and independent of each other. The wheel nodes are controlled by a PWM signal at same frequency as the steering servo, 50Hz and 1-2ms duty cycles and the power input is 5V. The wheel is driving forward or backward depending on the duty cycle of the PWM signal controlling the servo.

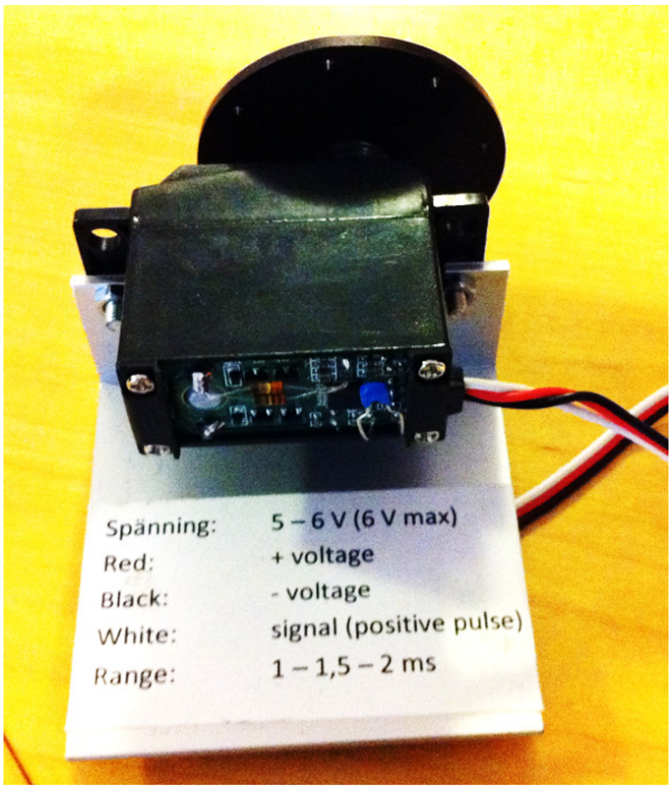


Figure 10 - The speed servo

4 Method

The project behind this report is divided into three phases.

(1) MIL - Make a design pattern and methodology for Simulink models

(2) SIL - Simulation and integration into VFB

(3) HIL – Integration into target hardware

Implementation and configuration possibilities are evaluated in each phase to find the most suitable methodology for the AFFE project.

4.1 MIL

In this section the Model in the Loop methodology and the choices that has been made to be able to meet the design requirements will be described.

4.1.1 Study of AUTOSAR, AFFE project and Simulink MBD

A study on AUTOSAR concepts and the AFFE project was needed in order to obtain a better understanding of the problem. Specifically more knowledge was required regarding the AUTOSAR concept, why AUTOSAR is needed in the automotive industry and how AUTOSAR makes it possible to reuse SW-Cs. Also the purpose of the AFFE project and possible requirements from the different companies involved in AFFE needed to be better understood. This information was provided by Mecel and the AUTOSAR [2].

Since Simulink was used as the modelling environment better knowledge of the capabilities of Simulink and how it supports AUTOSAR was also needed. The material used was retrieved from Mathworks [15] and also provided by the Simulink support team at Mathworks.

4.1.2 Modelling in Simulink

According to the requirements of the AFFE project [16] the architecture of the vehicle uses four separate wheel nodes and two control units (Figure 11). To simplify the decision was made to instead use two wheel nodes to represent the front wheels of a vehicle and one control unit to convert the input to the system to torque given to each wheel node (Figure 12). This decision was made because the same modelling method should be used disregarding of whether two or four wheel nodes are present in the system and the control unit logic to support multiple control units was not a part of this project.

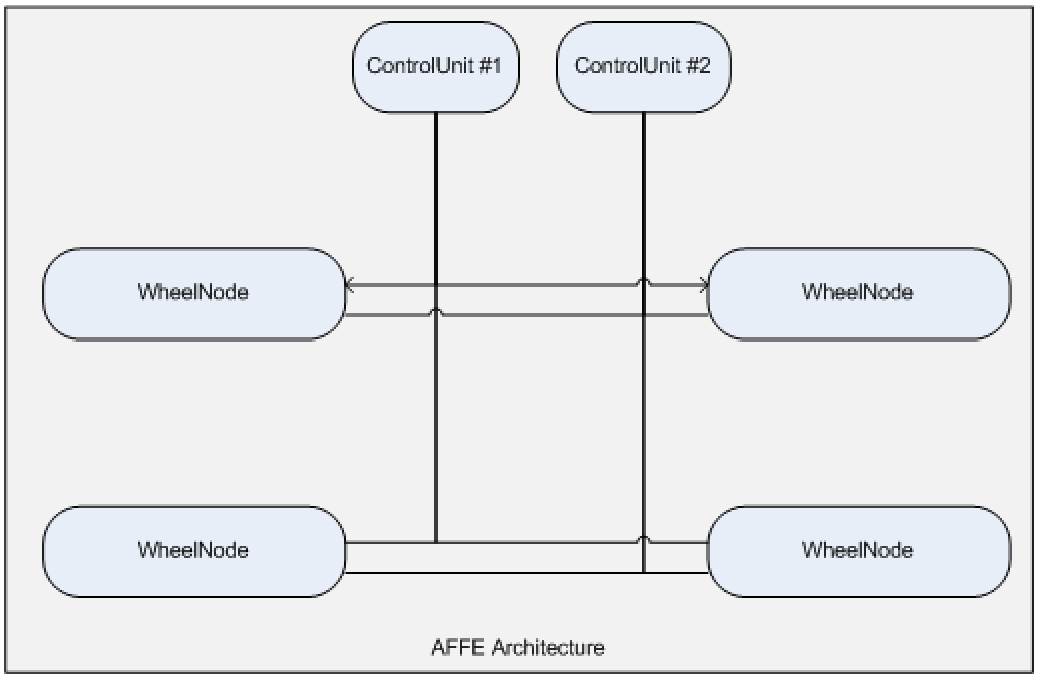


Figure 11 - The planned architecture for the AFFE project

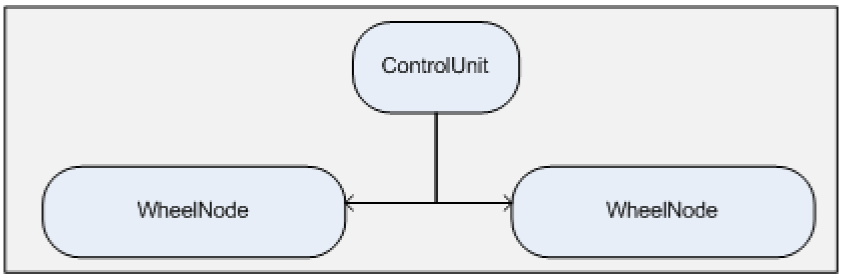


Figure 12 - The architecture used for this thesis

The different modelling methods examined during the MIL phase were model referencing and libraries. The focus was to determine how well they supported multiple instantiation, calibration of the different instances, and simulation of the system and code generation.

4.1.3 Evaluating the models and methods

As mention before there are different design choices when developing models in Simulink that fulfils the AUTOSAR requirements. Different modelling methods generate different system designs and therefor the methods mentioned in section 4.1.2 needed to be evaluated.

Evaluating modelling methods consist of parts such as; how different methods can be used when modelling, configuration possibilities, SW-C code generation for AUTOSAR requirements and how multiple instantiation can be done and how calibration parameters are supported.

These properties need to be evaluated to determine which method is appropriate for the AFFE system.

4.1.4 Simulate models in Simulink

Simulation of the models in Simulink is done to verify the functionality of the included SW-Cs and the system. By connecting sinks to the different signals of the model and analysing the output window it is possible to observe how different parts of the model behave. Simulating a model early in the development process allows for earlier detection of errors and the functionality of the model can be verified before code is generated.

4.1.5 Importing SW-C description into Simulink

One way of reusing SW-Cs is to import the SW-C description into the MBD environment such as Simulink. With Mathworks PSP toolbox it is possible to import and create a model skeleton of the SW-C to Simulink. It is then possible to reconfigure, modify and test the models to new requirements.

To be able to test the functionality of the PSP and how well it works an already existing SW-C description was used and a MATLAB script was developed.

4.2 SIL

This section describes the workflow and methodology in the SIL phase

4.2.1 Configuration and code generation of SW-Cs in Simulink

Before exporting and code generating the models in Simulink there are configurations needed to be examined e.g. how behaviours, settings should be set to be compliant with AUTOSAR. Therefor the models need to be configured with the AUTOSAR toolbox in Simulink. How parameters, instantiation of SW-Cs, runnables, samples time should be set and which level of the model can be generated together. These properties need to be explored for generating code correctly.

For code generation of the SW-Cs with AUTOSAR configuration a toolbox called Simulink Coder in MATLAB version r2011a needed. Within this toolbox it is possible to configure the SW-Cs for AUTOSAR requirements.

4.2.2 Generating configuration files in Picea Workbench

To able to test the generated code for a SW-C a basic platform consisting of the RTE and some of the BSW modules are required. These modules can be generated by AUTOSAR authoring tool such as Mecel Picea Workbench mentioned in section 3.1.2.
In Picea Workbench it is possible to set up signals, signals routing, communication interfaces and configuration of the modules for the required system. How signals should be routed between the modules and how the SW-Cs communicates with each other.

4.2.3 Simulation and Testing

To verify the functionality of the AFFE system an application was created which inputs data to the system and displays the output. From here this application will be referred to as the test application and the test application is presented in section 5.4.

Also an analysis was done to show the possibility of using external tools, such as Simulink, to model the environment or behaviour.

4.2.3.1 Test application for the AFFE system

The test application was written in C# because of the simplicity to include different form of graphical displays using WPF. The focus was not to have the highest performing test application but rather show how the input and output can be generated and displayed.

The source of the input for testing in the SIL phase can vary, e.g. the gas pedal value might be from; a physical gas pedal connected to a PC, a gas pedal in the test application or values entered manually. To simplify two methods of input was chosen in the test application for AFFE, a simulated gas pedal in the GUI as well as the possibility to enter values manually.

There are also different options for how to present the output data; list the output signals and their values or visualize the signals graphically, e.g. a speedometer or in a graph. The test application uses two methods to present the output; a speedometer for the different wheel nodes and a list of signals and their values received from the VFB simulator.

Depending on the functionality of the system there might be a need to simulate the behaviour in some way, e.g. the AFFE system outputs torque for each wheel node but the test application displays the speed which is calculated depending on the torque and previous speed. This was done by generating code from the behaviour model that was used during the MIL phase. Simulink generated C code so it needed to be converted to C#.

To transmit and display the input and output frames the signal names, bit size, bit position in the frame and if the signal is signed or unsigned needed to be extracted from the configuration exported from Picea Workbench.

4.2.4 Extending VFB for calibration parameters

The VFB simulator was extended to support calibration parameters to verify that Simulink supports generating code that uses calibration parameters. To simplify implementation the SW-Cs does not use the RTE layer but rather directly calls the created calibration parameters manager. Support for loading the VFB with configuration parameters was also included in the test application.

4.3 HIL

In this section the methodology of the HIL phase will be described.

4.3.1 Hardware testing with sample software

When running an application on the target hardware for the first time there could be numerous issues. To verify that the hardware and cabling works as intended a reference software, which has previously been proven to work, can be loaded and executed. Any errors encountered when running the reference software could be assumed to be related to the hardware or setup, once any errors are taken care of the newly developed software can be loaded and executed. This time any errors encountered could be assumed to be software or configuration related.

When going from the SIL phase to the HIL phase this approach was used to verify the functionality of both the hardware and the AFFE application. The reference software used was included in the Picea suite.

4.3.2 Simulate the environment in HIL

As in the SIL phase a PC based test application which simulates the environment and gives input to the AFFE application was required. The test application created in the SIL phase communicated over TCP/IP, but the target hardware only allows for communication using CAN, FlexRay and LIN.

To allow for communication with the target hardware the Kvaser USB to CAN adapter was used. This adapter connects to the hardware using the CAN interface and to a PC using the USB interface and it also had several libraries the CANLIN SDK which could be used in different programming languages to support sending and receiving CAN frames. The best option in the AFFE project was determined to be the integration of the C# dlls from CANLIB to extend the test application to also support communication over USB.

4.3.3 Integrate AFFE application to hardware

Similar to the SIL phase the source code and configuration needed to be generated from Picea Workbench. However the HIL phase requires more of the BSW layer to be included and also configuration for the drivers of the hardware which are intended to be used, e.g. interrupt handling.

The configurations used by the TriBoard drivers were the same as the ones used in the Picea reference application. This was because the AFFE application did not, at this point, use the hardware any differently than the reference application which already supported CAN communication. By using the same configuration as the reference application the cause of any error would be easier to locate.

4.3.4 Testing in HIL phase

For the functionality of the AFFE application to be verified the test application was run and used in the same way as in the SIL phase.

4.3.5 Adding support for PWM signals and AD conversion

In order for the SW-Cs to control an electric engine the system needs to support generating PWM signals, the PWM drivers had to be configured and included in the build process.

Also sensors are used to provide information regarding the surrounding and environment and therefore an analog to digital converter is necessary to input the sensor values. The ADC driver had to be configured and included in the build as well.

Since both PWM and ADC is very hardware dependent only the driver interfaces is part of the AUTOSAR standard, not the actual implementation. Therefore the actual configuration for the PWM and ADC drivers is less important but rather how the SW-Cs can interact with them.

5 Analysis and Results

In this section the result of the constructed models in Simulink will be discussed. How these models have been designed and which setting/configuration has been chosen due to AUTOSAR implementation requirements. The different methods have been analysed and compared to be able to point out which settings/configurations are recommended for particular issue.

5.1 Modelling in Simulink - MIL

During the MIL phase the system was modelled in Simulink and simulated to verify the functionality. Different methods for how to instantiate and configure the SW-Cs were evaluated.

5.1.1 Modelling methods

As mentioned in section 3.2.3 there are different ways to model in Simulink. The two methods that has been evaluated and tested are model referencing and using libraries. The results of the simulation are identical for both methods.

There is another method to create models where subsystem is used at top level model. But with this method multiple instantiation of one SW-C is not possible. In order to have different instantiation of on SW-C the model block must be duplicated, which is not optimal. Therefor Subsystem method will not be used in this project.

5.1.1.1 Model Reference

It is possible to include one model in another model using Model blocks in Simulink. A Model block is one building block in a model that can reference to another model. Each Model block at the top level represents an instance of another model, called sub model or referenced model, and is not containing any building blocks itself. The model that contains the referenced model is called parent model. All sub models and parent models together define a hierarchy of modelling.

The referenced model block show the input and output ports of the sub model and these ports can be connected to other blocks to create a large scale model. [12]

5.1.1.2 Libraries

If one block is repeatedly used in a model there is a method to create a template and reuse it everywhere it’s needed. This template is called a library block and can be created as a normal model and stored as a library. Once the library has been created it can be locked and imported into the Library browser.

When a library block is used a reference block to it is created. The reference block is an instance of the library block and the contents of the library block are not stored in the model, the reference block only stores a library link which contains a path to the library block. To be able to modify the reference block the link between the library block and the reference block must be disabled. However if only the values of the reference block are modified then the link becomes parameterized. [13]

5.1.2 System overview

The system (Figure 13) contains one control unit, called ControlUnit, and two wheel nodes, called LeftWheelNode and RightWheelNode. The two wheel nodes should be instances of a common base model, so that the only difference between them is the configuration parameters applied.

In addition there is a simulation environment consisting of LeftWheelSpeed and RightWheelSpeed whose tasks are to simulate the speed of each wheel based on the torque that has been calculated at each wheel node and the current speed of the wheel.

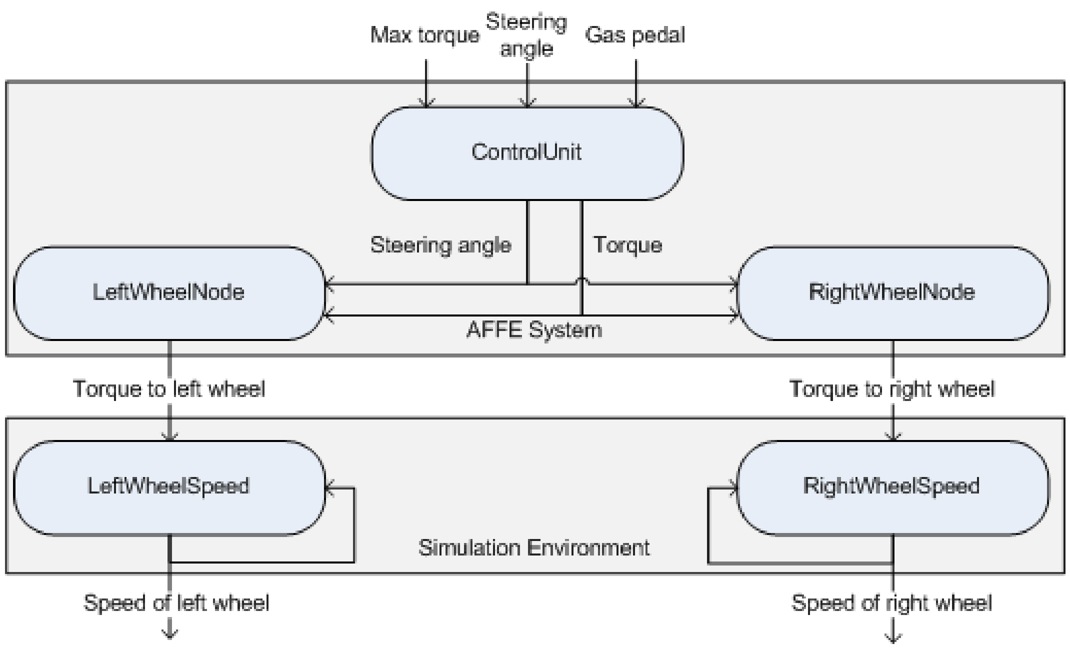


Figure 13 - An overview of the system developed during the MIL phase

5.1.2.1 ControlUnit

The control unit is responsible for the dynamic control of the vehicle. In this very simplified system the control unit is only converting a gas pedal input to a torque request. From the max torque and gas pedal inputs the torque is calculated and sent to the wheel nodes, and the steering angle is passed on to the wheel nodes without modification.

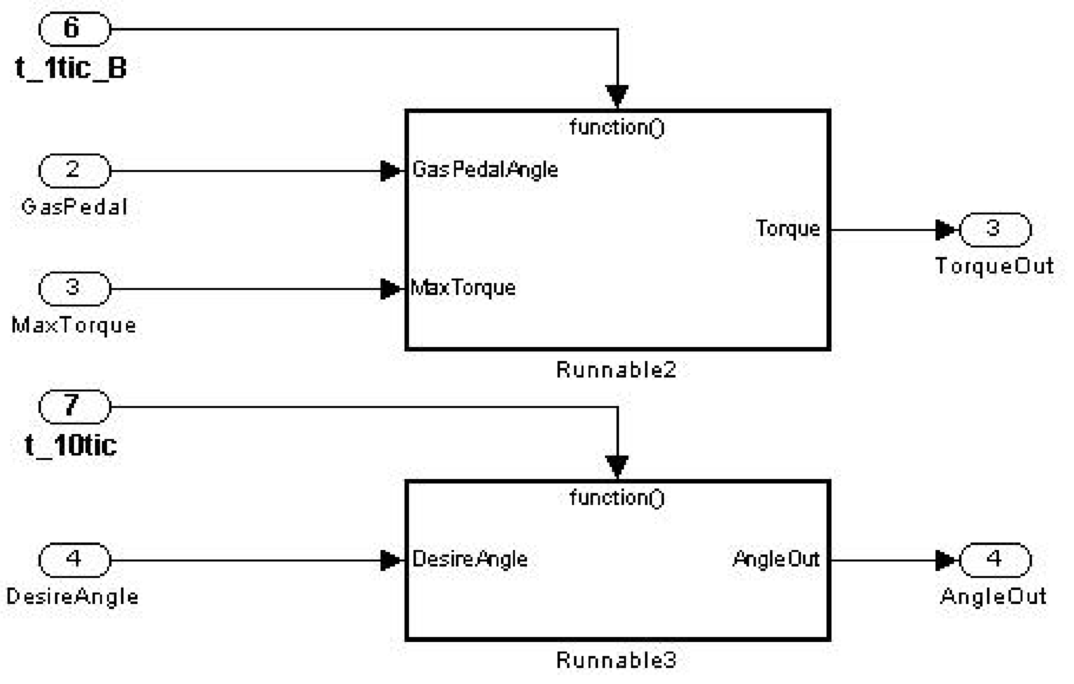


Figure 14 - The ControlUnit SW-C

As can be seen in Figure 14 there are two runnables in the control unit SW-C, each with their respective tasks. The first runnable (Runnable2 in Figure 14) is reading the max torque and gas pedal inputs, uses a lookup table to convert the gas pedal value to a percentage of the max torque that should be applied. Once this is done the torque is written to the torque output port of the control unit SW-C. The second runnable (Runnable3 in Figure 14) reads the steering angle input and writes the value to the steering output of the SW-C. The execution period for each runnable is set to 100ms.

5.1.2.2 WheelNode

A wheel node SW-C (Figure 15) has one runnable which calculates the torque that is applied to the wheel associated with the wheel, the model of the runnable is shown in Figure 16.

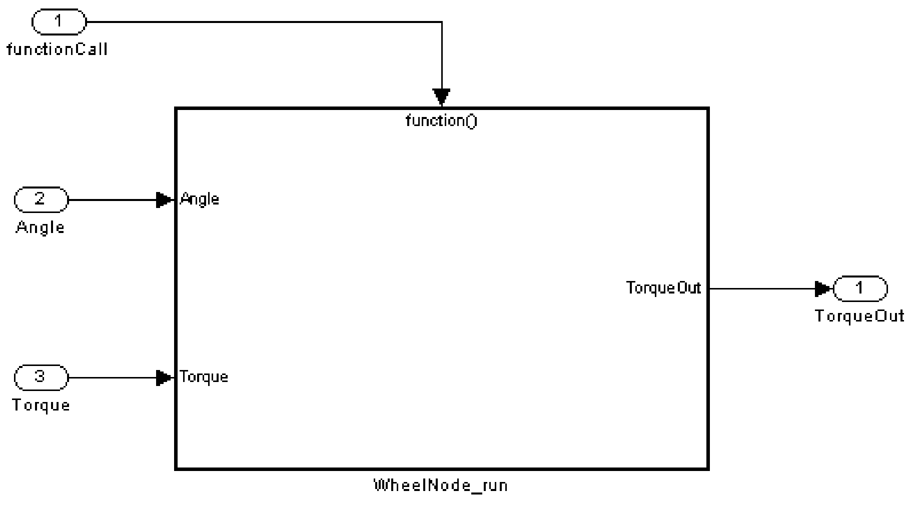


Figure 15 - The WheelNode SW-C

The torque is calculated based on the current steering angle and the torque received from the control unit. If the vehicle is turning the inside wheel node should apply less torque because it has a shorter distance to go than the outer wheel, i.e. if the vehicle is turning right, the right wheel node should output less torque than the left wheel node and if the vehicle is turning left the left wheel node should output less torque than the right wheel node.

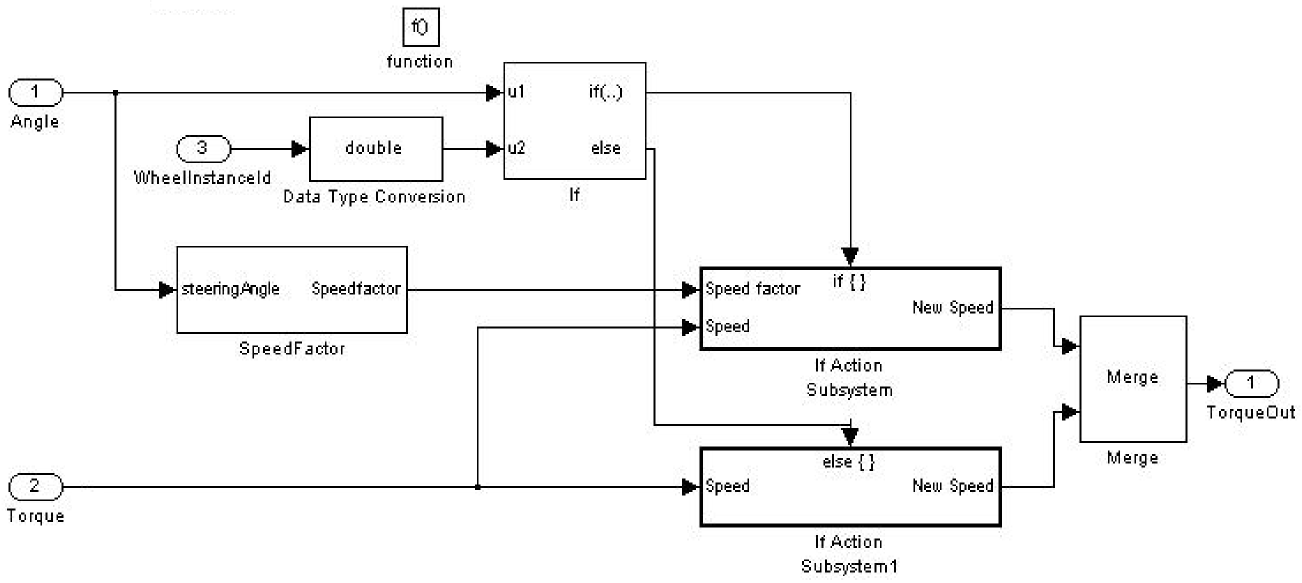


Figure 16 - Model of the Runnable in the WheelNode SW-C. The torque is calculated based on the angle and torque values received from the control unit.

To know how much torque to apply the wheel node contains an algorithm that calculates a speedfactor, this is multiplied with the input torque to determine the torque to apply.

The speedfactor is calculated for the inner wheel using the wheel base and wheel track distances. Wheel base is the distance between the front and rear wheels (assuming there is four wheels) and the wheel track is the distance between the front wheels which are the only wheels modelled in the AFFE system.

The parameters that can be configured in the wheel node are; the wheel id which determines if the wheel node is on the left (0) or right (1) side of the vehicle, the wheel track distance and the wheel base distance. Only the wheelID parameter is instance specific, wheel track and wheel base are the same for all instantiations.

5.1.2.3 WheelSpeed

The WheelSpeed model contains the simulation logic to, given a torque and the previous speed of a wheel, calculate the current speed of the wheel. The torque input is read from the wheel node to which the WheelSpeed model is connected, the previous speed is stored locally from the previous execution.

The speed calculation is done by accounting for the weight of the vehicle (kg), existing counter forces (N), e.g. friction and wind, and the wheel radius (m).

5.1.2.4 Model configuration file

To simplify the instantiation and applying parameters to the different instances and blocks in the system a configuration file was used, this file configures the models with the appropriate parameters and is executed in the MATLAB environment. The file used in this project is included in Appendix X.

5.1.3 General observations when modelling

To model and generate code for AUTOSAR there are a few general guidelines to use.

Mask parameters can be used to create instance specific parameters for SW-Cs in both Model Reference and Libraries methods. However the mask parameters cannot be used as AUTOSAR calibration parameters when generating code, this limits the usefulness of these and therefore it is not a feasible option.

5.1.3.1 Simulink workspaces

The different parameters, types and variables available in Simulink can be located in different workspaces. There is the base workspace which is the MATLAB environment; there are also workspaces for each model. Using the model explorer the parameters can be created in the appropriate workspace and the properties of the parameters can be modified (e.g. initial values, sizes etc.). To generate code that includes calibration parameters and per instance memory the parameters has to be located in the base workspace when generating otherwise the calibration parameters will be regarded as constants by the code generator.

5.1.3.2 Type of parameters to use

There are different types of parameters used when modelling for AUTOSAR, most common are MATLAB variable, Simulink signal, Simulink parameter, AUTOSAR signal and AUTOSAR parameter.

The MATLAB variables, Simulink signals and Simulink parameters will all be converted to constants with their current value when the code is generated.

The AUTOSAR signal type is used for per-instance-memory to store data between executions of runnables. For example in the WheelSpeed model discussed in 5.1.2.3 the previous speed is stored using the AUTOSAR signal. To have calibration parameters generated the parameter must be of the AUTOSAR parameter type. The difference between a parameter and a signal is that the value of the signal is expected to vary throughout the execution, i.e. similar to a line that connects blocks when modelling. A parameter will be configured and will remain constant throughout the simulation.

5.1.4 Modelling using Model Reference

With model reference the model was created in a separate model file. Parameters and calibration parameters needed to be created in the models workspace in order to have instance specific calibration parameters and have each instance of the model act differently. The initial values will be used if the parameters are not set to something else before simulation either in MATLAB or Simulink environment.

The system model which includes all parts of the system can be seen in Figure 17. The values of the instance specific calibration parameters are set by providing the Model blocks with arguments, either through the GUI or by executing a command in the MATLAB environment. The arguments can be configured in a MATLAB script file and the script can be executed to automatically assign the values to the different Model blocks. The order of the arguments are specified in the Model Explorer and can be specified at the same time as the instance specific calibration parameters are created in the model's workspace.

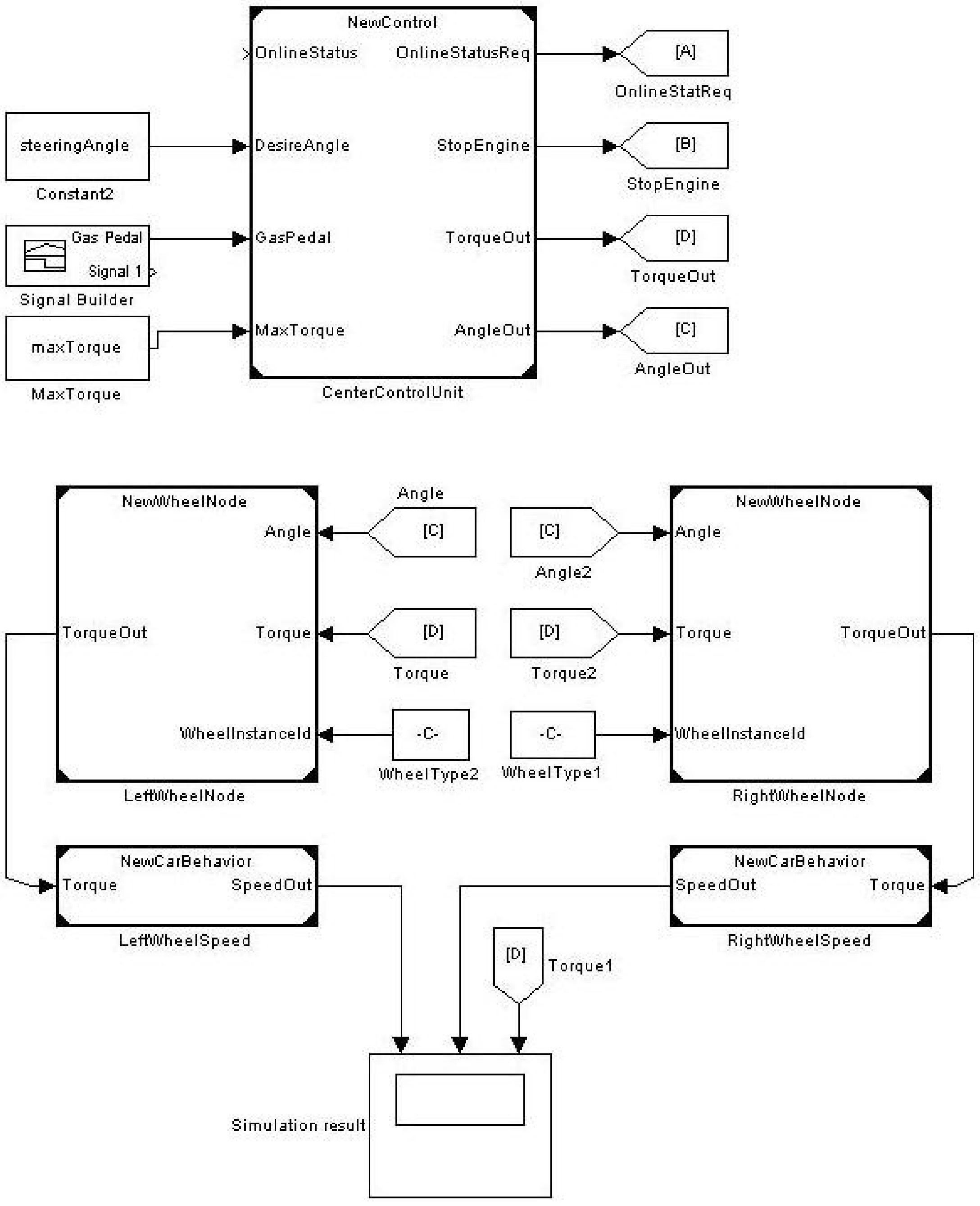


Figure 17 - System model when using model referencing

An issue discovered was that the function call cannot cross the boundaries of the Model block; this is because the data type must be specified for all the input ports of a Model block but it does not exist a data type for a function call. Since the function calls cannot be given as input to the referenced model the Function-Call Generator or Stateflow chart must be on the same level as the SW-C within the referenced model (Figure 18). This introduces extra work of having both the ports of the Model block and the ports of the SW-C well specified and matching, but it is necessary if model referencing is to be used.

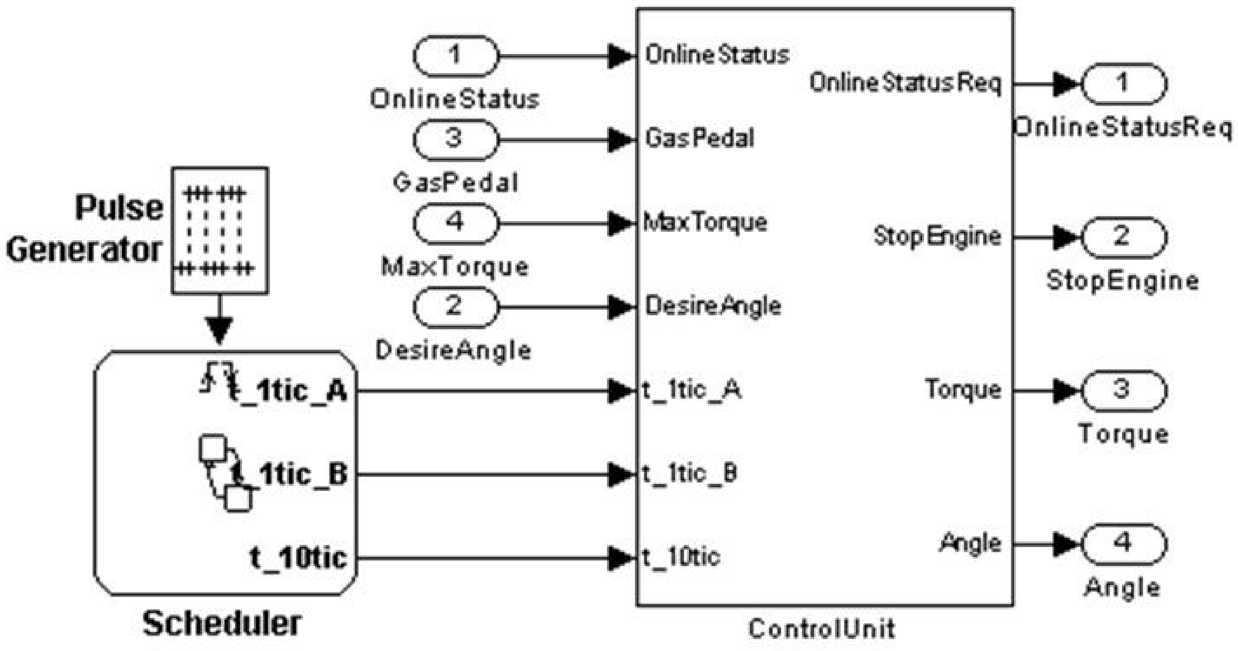


Figure 18 - The referenced model contains a chart and the SW-C

Model Reference also makes it possible to develop models separately and combine them once completed. Although it has a slow initial build time and is time consuming for prototyping because if the model that is being referenced is modified then it needs to be rebuilt, but once it has been built and is not modified it offers fast simulation time. Therefore model reference is appropriate for large models with many blocks since it does not need to keep all the referenced models in memory, only the top model [14].

A list of advantage and disadvantage regarding Model Reference can be viewed in Appendix section Modelling method.

5.1.5 Modelling using Libraries

When it comes to instantiation of the wheel nodes, separate workspaces for each wheel node do not exist for Libraries; therefor it is not possible to have the same parameter in the different instantiations if the instances need to have different values of their parameters. Instantiation using Libraries can be done in two ways to support simulation; either masking the subsystems or having unique names of the parameters for each instance, e.g. wheelIdLeft and wheelIdRight.

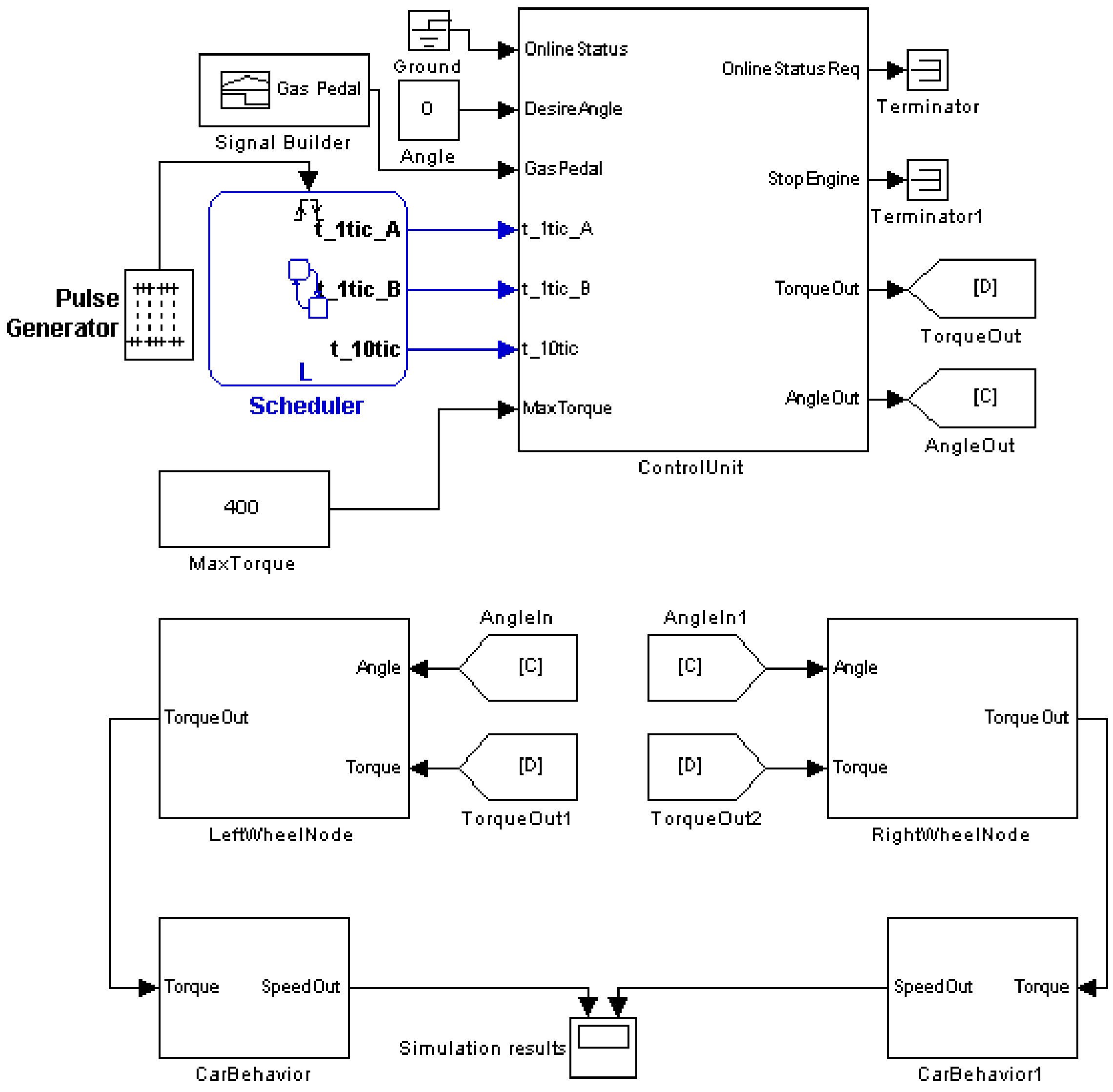


Figure 19 - Modelling using library blocks

Function calls can cross the boundaries of the library block as seen with the control unit block in Figure 19, so it does not need to contain the extra level which was required when using Model Reference and this makes it easier if different instantiations requires different sample times.

Using Libraries it is easy to model as long as the models are small with few blocks. But it becomes complicated to edit the content of an existing block since the link between the reference block and library block must be deleted and the library block must be unlocked. This makes it complicated to integrate SW-Cs that have been developed separately and to update existing SW-Cs.

Since each instantiation of a library block is built separately during simulation [14] it takes longer to run a simulation if the SW-C that has been instantiated contains many blocks.

5.1.6 Importing Software Components to Simulink

To be able to import SW-C XML description file into Simulink environment, an additional MATLAB Toolbox was needed. The package named Pilot Support Package for AUTOSAR (PSP) where arxml.importer class exists was used to import old SW-C which created a skeleton of the model in Simulink model workspace for editing and reconfiguration. In the arxml.importer class there exist a few methods for importing AUTOSAR SW-C. These methods had been tried and evaluated for different purposes to determine which method is appropriate for what kind of purposes. These methods can be found by first creating an importer object from the xml description file. For more details regarding the methods in the PSP see Appendix B – Importing SW-C using PSP.

5.1.6.1 Importing SW-C using a PSP from MathWorks

To import SW-Cs to the Simulink environment for editing and testing a MATLAB script was created where the methods available in the PSP were used.

First define the name of the SW-CD file and then create an ARPSP importer object which imports the SW-CD. The importer object now contains all information regarding the SW-C model. Next is to analyse the SW-Cs dependencies, enumerator classes to determine what can be done with the importer object. Before finally creating SW-C model of the importer object to Simulink all dependencies must be set and enumerator classes needs to be created otherwise it will not be visible in the model.

When steps above are done, then the method createComponentAsModel with the component name holding in the importer object then create a skeleton of the SW-C model. With the R2011 release of the PSP used the internal behaviours of the SW-C will be included, but still not all detail/configuration will be set in the Simulink model. If using a previous release of the PSP then another method called `createComponentWithInternalBehaviorAsSubsystem` should be used to create a skeleton model with internal behaviour included.

5.1.6.2 Bug detection and fixes with PSP importer class

The importer class from the PSP used is still under development and therefore contains bugs and unsupported behaviour which have been detected. One of the bugs was that the importer class does not import all detail and behaviour of the SW-CD. Another bug was Data Type error. MATLAB does not support non-standard data type such UInt4, when UInt4 is used and import to Simulink model it converts automatically to uint8(15), but it should convert the data type UInt4 to FixPoint type instead. Furthermore the data type of the initial value of some of the input ports in the imported model where supposed to be set to uint8(0) but was set to a FixPoint value instead. MathWorks has been notified of these issues.

Also the importer class will create a default Init-runnable even if the SW-CD contains a custom predefined Init-runnable. The custom predefined Init-runnable was converted to a runnable in Simulink model after importing procedure which required a periodic event which was not expected according to the SW-CD.

5.2 Code generation MIL → SIL

From MIL to SIL Simulink Coder has been used for code generation of the models.

5.2.1 Code generation using Simulink Coder

When generating the C code and the SW-CD from the models in the MIL phase the toolbox Simulink Coder was used. In this toolbox there were settings that had to be configured before AUTOSAR compliant code can be generated. The solver needed to be set to discrete with fixed step and the inline parameters option had to be set if calibration parameters should generate correctly, also the target file had to be changed to "autosar.tlc". See Appendix C – Settings in Simulink Coder for instructions on how to set these settings.

To instantiate SW-Cs it is possible to use the AUTOSAR instantiation, where each instantiation must be located on the same ECU, or generate each instantiation as a separate SW-C to allow for them to be assigned to different ECUs. If each instantiation is generated as a separate SW-C and they are using calibration parameters it will not be possible for them to be located on the same ECU unless the instance specific calibration parameters have different names during code generation.

In the AFFE system, each instantiation of a wheel node was expected to run on separate ECUs, therefore the wheel nodes were generated as separate SW-Cs.

As mentioned in 5.2.1, in order to instantiate a SW-C during simulation each instantiation had to have its own parameters; either configured by model arguments when using Model Reference or different parameter names or masked subsystem when using Libraries. But when code generating these settings had to be changed for calibration parameters to be generated correctly since the AUTOSAR parameters have to be located in the base workspace.

To generate a SW-C the triggers of the runnables and the settings of the ports had to be specified, e.g. sender/receiver, implicit/explicit mode, data types etc. Also it is not possible to generate the complete system; each SW-C needs to be generated one at a time.

5.2.1.1 Code generation using Model Reference

The parameters required for instantiation of the SW-Cs which had previously been located in the model workspace needed to be moved to the base workspace in order to generate calibration parameters, otherwise they will be generated as constants.

It is not possible to configure a Model block as an AUTOSAR SW-C. Only a Subsystem block can be configured as a SW-C therefore it was necessary to model the SW-C, when using model reference, as it was discussed in section 5.1.4 to be able to generate code for the SW-C.

5.2.1.2 Code generation using Libraries

To be able to generate code for a SW-C using the Simulink Coder the parameters which was used had to be modified. Because during simulation it was necessary to have either masked parameters or use different names for the parameters of different instantiations the library block had to be modified to use the AUTOSAR parameters located in the base workspace. Otherwise the calibration parameters would not be generated correctly.

5.2.2 Code generation issues

There were issues when trying generating code for SW-Cs. The sample times for runnables and instantiation of SW-Cs are discussed in this section.

5.2.2.1 Inheriting sample time

One of the issues with the code generation part was to inherit the sample time from the Function-Call Generator block or the Stateflow chart. Because the function call generator source cannot be code generated together with the SW-Cs and therefor the sample time for the runnables in SW-Cs needs to inherit from function call generator.

One of the required functionalities was to be able to have different sample times for different runnables in SW-C. It is possible to have different sample time but needed to be changed manually before generating C code.

It is not possible to have several function call generators with different sample time for different runnables in one SW-C. The same issue occurs when using stateflow charts instead of function call generator. The sample time is inherited only from the source connected to the chart, not the actual sample times calculated in the chart.

This issue is confirmed by the support group of MathWorks. They are aware of this problem and currently don't have a solution yet, but in future release they hope to have a solution for this inherit sample time issue.

5.2.2.2 Instantiation of a SW-C

One of the requirements in this project was to be able to have multiple instantiations of one SW-C. This functionality is possible in the MIL phase by using either Model Referencing or Libraries to instantiate. But generating code with instance specific parameters and ports is not possible. This has to be done manually, which mean the SW-C developer has to modify the generated code.
This has been confirmed by MathWorks and currently there is no solution to instantiate a SW-C.

5.3 Virtual Function Bus Simulation – SIL

5.3.1 Configuration in Picea Workbench

As mentioned in section 3.1.2 Picea Workbench can be used to configure a basic platform and generate RTE and BSW modules for the system. In Picea Workbench it is possible to configure each module, how the modules are connected and how the signals are routed through the RTE and BSW layers.

5.3.1.1 Input

Different inputs were given to Picea Workbench to configure the basic platform. These inputs are signal names, data types of the signals, SW-C interface, communication interfaces and connection between different runnables. The input varies depending on the designed system.

5.3.1.2 Output

From the input and configurations made in Picea Workbench the output is the configuration description files for different modules. The configuration files are then processed through Picea code generator for generating code for all modules. It is the .c and .h files which will be built together with the SW-C files generated from Simulink to make a complete test environment. Any IDE software with C-compiler can be used to build the test system.

5.3.2 Integrate SW-C into VFB simulator

With the platform configured the SW-Cs code generated from the Simulink Embedded coder could be integrated. The platform consist of folder structure with configuration files, RTE files, BSW files, integration package and SW-Cs folders. The software components source codes from embedded coder were then copied into the platform folders ad Visual C++ 2008 Express were used to compile and debugging the executable output.

The software components which were integrated into the extended VFB are;

1. CentralControlUnit
2. LeftWheelNode
3. RightWheelNode

5.3.3 Extending VFB Simulator to support calibration parameters

To support calibration parameters some changes were necessary to the VFB simulator, a module was added to the VFB to handle periodic parsing of a xml file to support updating of the calibration parameter values, and provide access to the calibration parameters for the SW-Cs.

5.3.4 Connecting VFB to Simulink

To connect the VFB simulator to Simulink to allow for more advanced simulations different options exist. The TCP/IP receive and send blocks, that is part of the Instrument Control Toolbox, could be used to handle TCP/IP communication, but the toolbox was not available during this thesis to be tested and validated.

Instead a server module was created in Java which were loaded into MATLAB and allowed communication between the VFB simulator and the Simulink models. The server was setup similar to the test application where a callback had to be registered to which the server module forwards any incoming CanFrames. Since MATLAB supports the use of Java objects in its environment the callback function can use the CanFrame datatype to extract signal values and update a Simulink model with the new values, e.g. update the value of a constant block with a signal value each time a new frame is received.

There also exist other options to provide this functionality, e.g. writing the server module in C, C# or using S-functions within Simulink.

5.4 HIL

The HIL phase consists of extracting the extended configuration from Picea Workbench, generating source code from the configuration, building the application and running it on the hardware to test the functionality.

5.4.1 Testing the hardware

Initially the Triboard was loaded with the reference application, which contains the AUTOSAR stack and a SW-C that transmits messages over CAN. The messages were transmitted from the Triboard but could not be received in the PC due to timing errors. It was discovered that the current clock frequency on the Triboard was slower than the clock frequency that the hardware drivers had been configured for. This caused timing errors and the clock had to be changed from 20MHz to 24MHz in order for the timing of the signals to be correct.

5.4.2 Configuration in Picea Workbench

As in SIL step section 4.2.3, configurations needed to be done in Picea Workbench for the HIL step. In HIL step Picea Workbench is used to configure the module drivers for the target hardware, in this case Infineon Triboard TC1797. The outputs is the same as in the SIL process where configuration description files are exported and Picea code generator is used for generating .c and .h code for all modules and configurations of the system.

5.4.3 Integrating SW-Cs into AUTOSAR Picea Integration Package

To build the AFFE application and generate a binary file, which could be loaded on the hardware, the complete system had to be integrated. The configuration from Picea Workbench was used to generate the source code for the different modules used and together with the SW-Cs source code the AFFE application was built. Two issues were found during the build phase and these issues are described in 4.3.3.

5.4.3.1 Error in Math Library

The control unit SW-C used functions from the math library; fmod, ceil, floor and sin, in its implementation but at compile time errors were encountered regarding unknown symbols in the math library. The issue was solved by removing the functions from the library and recompiling them using the same compiler that is used to build the application. The assumption is that the libraries had been built using another version of the compiler which was not compatible with the current compiler.

5.4.3.2 Error in hardware initialization function

The default initialization functions for the hardware can be included by the linker, unless a flag is set to instruct the linker not to include these functions. If the application linked provides its own initialization functions this flag has to be set.

An issue that occurred during the building of the application was that the default initialization functions had unknown symbol errors, similar to those with the math library. The application provided its own initialization functions in the Mcal (Microcontroller abstraction layer) module and the default function was not used. The issue was solved by removing the functions from the library since it was not needed. Again it is assumed that the library had been compiled using a compiler with a different version and therefor caused the unknown symbol errors.

5.4.4 Running application on hardware

When the AFFE application was loaded and executed on the target hardware another issue was encountered, the ECU manager transitioning into sleep mode (section 5.3.4.1). Once this issue was taken care of the functionality of the AFFE application was tested and confirmed to be the same as in the SIL phase.

5.4.4.1 Issue with ECU manager going into sleep mode

The application was running as expected but did not transmit any CAN signals and after debugging it was discovered that the ECU manager module transitioned from run mode into sleep mode before the application could request communication. But in order for the application to request communication the ECU had to be in the run mode.

This problem was caused by the runnables in the AFFE application having a higher sample rate, 100ms compare to the 10ms in the reference application. The issue can be solved in two ways, either increasing the sample rate of any runnable requesting communication, or increase the duration before the ECU manager transitions from run mode to sleep mode. These modifications are done when the configuring the application in Picea Workbench.

5.4.5 Extend application to support PWM signals and ADC

For the purpose of creating a demo application of the system the SW-Cs needed to not only transmitting CAN frames with the output but also control servos and read sensor values.

5.4.5.1 Pulse Width Modulation

A PWM signal is used to control an electric servo. In the AFFE application there are two electric motors to control, a steering servo which changes the steering angle and a wheel servo which causes a wheel to rotate.

The configuration of the MCU, PWM and PORT drivers, which are necessary for generating PWM signals, was included from a different project at Mecel. The PWM signal is generated from the General Purpose Timer Array whose settings have been configured in the MCU drivers.

The control unit SW-C controls the steering servo and sets the duty cycle and period to reflect the input steering angle. The wheel node SW-Cs controls one PWM signal each that is used to output the current torque for that wheel node to the wheel servos.

5.4.5.2 Analog to Digital Conversion

The steering servo used provides sensors that can read the actual steering angle, which might not always be the same as the angle output by the control unit SW-C.

To read sensor values the ADC drivers needed to be configured and activated on the hardware. The resolution of the conversion was chosen to be 12-bits.

5.5 Simulation and Test Application

To test and simulate the generated code from MATLAB/Simulink an application was created using C# and WPF.

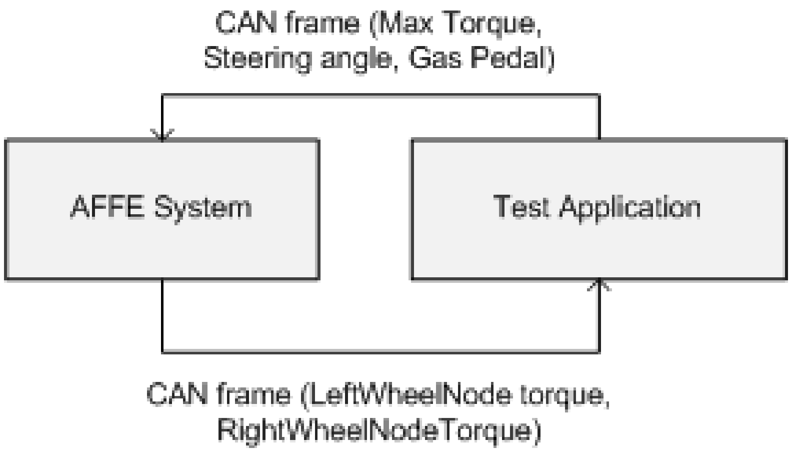

CAN frame (Max Torque, Steering angle, Gas Pedal)
AFFE System
Test Application
CAN frame (LeftWheelNode torque, RightWheelNodeTorque)

Figure 20 - How the test application is used with the AFFE system

The test application provides the max torque, steering angle and gas pedal input to the AFFE system and receives the torque output from each wheel node (Figure 20).

5.5.1 Test application overview

The test application consists of three modules as shown in Figure 21, the GUI that has two different options for how to give input and display output from the system; the graphical interface which provides graphical components and the generic table view which list all signals in tables and allows the user to input values by hand. The other two modules are the communication modules, one for the SIL phase when CAN frames was sent over TCP/IP and one for the HIL phase when CAN frames was sent over USB.

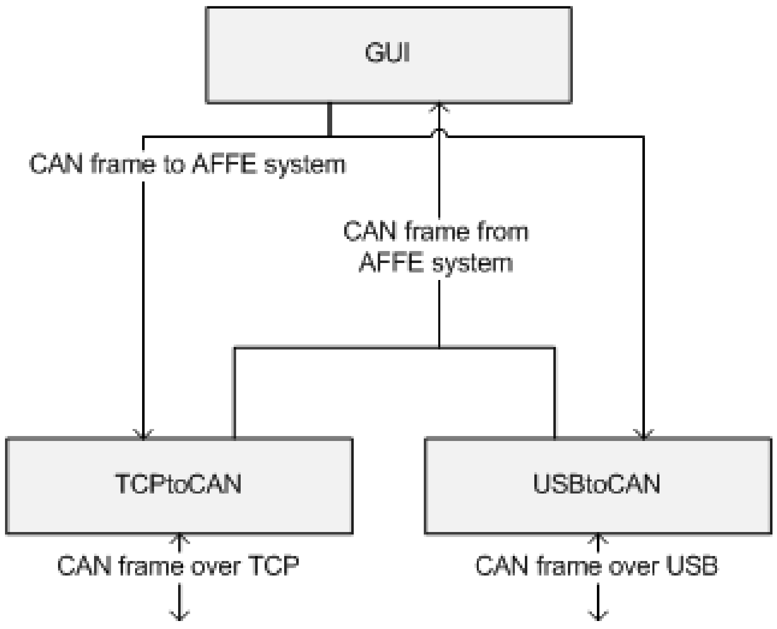


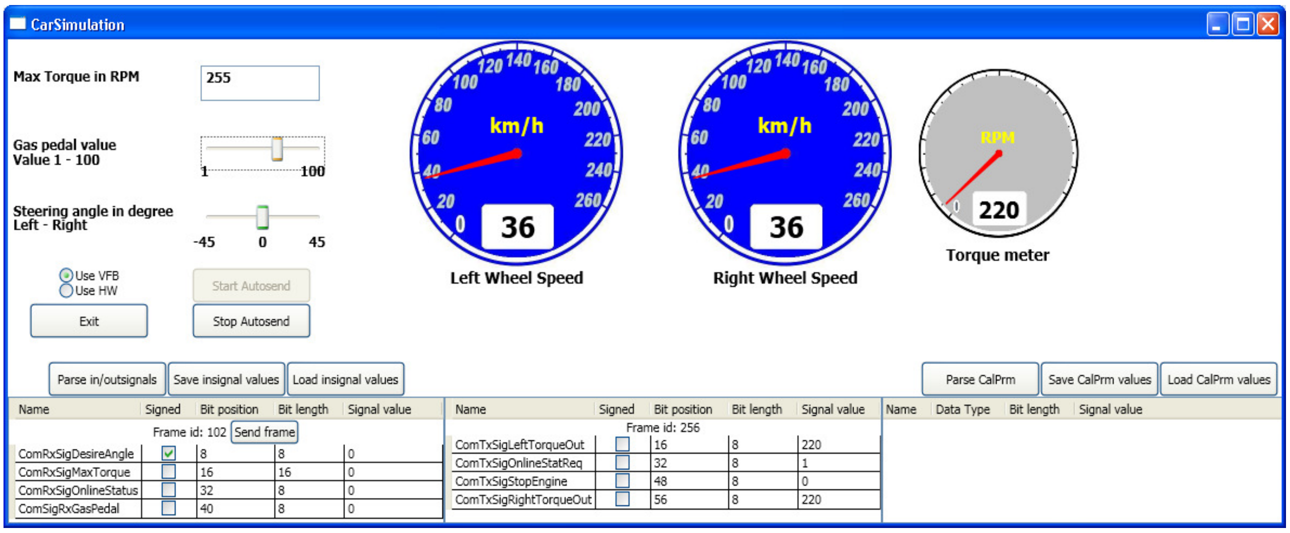
Figure 21 - The three modules of the test application

5.5.2 GUI

In this section the two different options to input data to the system are presented.

5.5.2.1 Graphical Display

In Figure 22 the graphical display is at the top of the window, the input to the system is visualized using two sliders which give the gas pedal and steering angle values. The maximum torque input is also displayed in a textbox.



Frame id: 102 Send frame

Name	Signed	Bit position	Bit length	Signal value
ComRxSigDesireAngle	✓	8	8	0
ComRxSigMaxTorque	☐	16	16	0
ComRxSigOnlineStatus	☐	32	8	0
ComSigRxGasPedal	☐	40	8	0

Frame id: 256

Name	Signed	Bit position	Bit length	Signal value
ComTxSigLeftTorqueOut	☐	16	8	220
ComTxSigOnlineStatReq	☐	32	8	1
ComTxSigStopEngine	☐	48	8	0
ComTxSigRightTorqueOut	☐	56	8	220

Name	Data Type	Bit length	Signal value

Figure 22 - The GUI of the test application for the AFFE project

To transmit the input to the system the values of the sliders will continuously be read and sent by a separate thread once the autosend feature is activated.

Since the output from the AFFE system is the torque of each wheel it needs to be put through a simulation to determine the actual speed of the wheels. Once the simulation is done the calculated speed will be displayed in the speedometers, there is one speedometer for each wheel and also one to display the torque.

5.5.2.2 Generic table view

The different signals in the system are presented to the user by two different tables; see the lower part of Figure 22, the input table and the output table. In each table the signals are grouped by the CAN frame they belong to, i.e. in the AFFE system there is one CAN frame with id 102, which is an input to the system, and the signals available in that frame are:

- ComRxSigDesireAngle
- ComRxSigMaxTorque
- ComRxSigOnlineStatus
- ComSigRxGasPedal

The properties for each signal are; the name, whether the signal is signed or unsigned, which bit position the signal has in the payload of the CAN frame, the size in bits of the signal and the value to send to the AFFE system. The output table displays the same properties.

The third table available, the rightmost, displays the calibration parameters which needs to be parsed separately, once parsed the values can be changed and then saved to the location where the extended VFB simulator looks for a new file. Calibration parameters are not supported during the HIL phase.

5.5.3 Communication modules

The test application has been designed to support different modules that allows for output and input during the SIL and HIL phases. Currently there are two modules, TCPtoCAN and USBtoCAN, which supports CAN communication over both TCP/IP and USB, see figure Y.

The modules for both TCP/IP and USB are loaded as dynamic-link libraries (dll) and share the same architecture. They are initialized with either port (for TCP/IP) or channel (for USB) and both require a Signal Group Manager object (see section Signal handling) which contains all information regarding the signals in the system. A callback is also necessary to specify so the modules know where to forward any incoming CAN frames.

Each module has its own receive thread which starts to execute once the module has been initialized. When a packet arrives it is parsed and forwarded to the callback.

5.5.4 Test application internals

Internally the test application uses several threads to provide better performance and to simulate the speed of the wheels. In addition to the threads in the communication modules (section 5.5.3) the GUI is using four threads (Figure 23) to perform updating of the graphics, simulating the speed and auto transmit values.

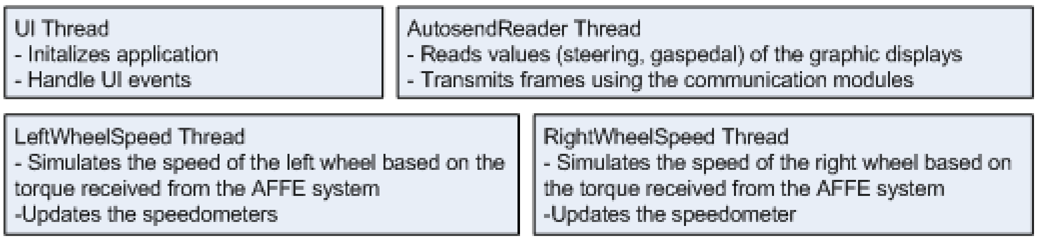

UI Thread
- Initalizes application
- Handle UI events
AutosendReader Thread
- Reads values (steering, gaspedal) of the graphic displays
- Transmits frames using the communication modules
LeftWheelSpeed Thread
- Simulates the speed of the left wheel based on the torque received from the AFFE system
-Updates the speedometers
RightWheelSpeed Thread
- Simulates the speed of the right wheel based on the torque received from the AFFE system
-Updates the speedometer

Figure 23 - The different threads used in the GUI module of the test application

The UI thread is the default thread in C# / WPF. It handles user interaction, e.g. button clicks and selections as well as updating the graphics.

The AutosendReader thread reads the values off the graphic controls, steering angle slider and the gas pedal slider. These values are then put into a CAN frame and sent to the AFFE system.

The two simulation threads, LeftWheelSpeed and RightWheelSpeed, uses the last received torque values and calculates the speed based on the torque and the previous speed of the wheel. The execution period is 100ms for both of these threads.

5.5.4.1 Parsing of signals and calibration parameters

The signals and calibration parameters used in the AFFE system is parsed from the configuration generated by the authoring tool.

5.5.4.2 Signal handling

For the test application to handle the different frames present in a system it needs to know the mapping of signals to the payload of the frame, i.e. which bits of the payload belongs to which signal.

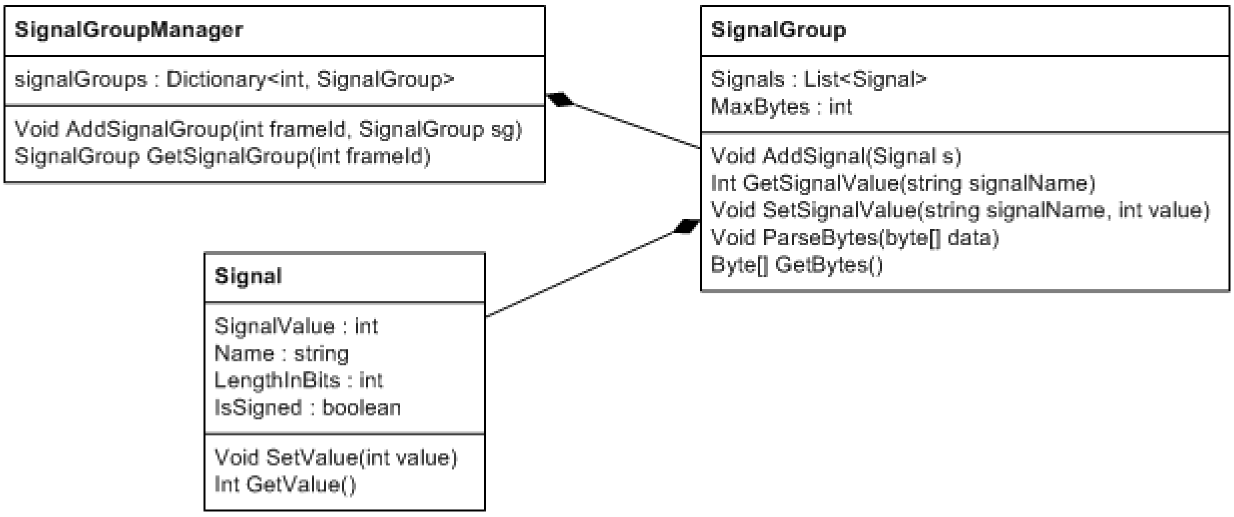

SignalGroupManager
signalGroups : Dictionary<int, SignalGroup>
Void AddSignalGroup(int frameId, SignalGroup sg)
SignalGroup GetSignalGroup(int frameId)
SignalGroup
Signals : List<Signal>
MaxBytes : int
Void AddSignal(Signal s)
Int GetSignalValue(string signalName)
Void SetSignalValue(string signalName, int value)
Void ParseBytes(byte[] data)
Byte[] GetBytes()
Signal
SignalValue : int
Name : string
LengthInBits : int
IsSigned : boolean
Void SetValue(int value)
Int GetValue()

Figure 24 - Class diagram of the relations between SignalGroupManager, SignalGroup and Signal

The signals that are included in a certain CAN frame are stored in a SignalGroup. The SignalGroups are stored in a SignalGroupManager object and each is mapped to a CAN frame

ID. Once a CAN frame is received the SignalGroupManager is queried for the SignalGroup associated with the frame. See Figure 24 for the relations between SignalGroupManager, SignalGroups and Signals.

For the AFFE system there are two different types of frames available, a CAN frame with ID 102 which contains the input signals and a CAN frame with ID 256 which contains the output signals of the AFFE system. These two signal groups are stored in the signal group manager.

The AFFE Vehicle Simulator can handle signals of sizes 1 – 32 bits, both signed and unsigned and requires no special alignment within bytes.

Little endian is used by the AFFE system for all the signals it receives and sends and this requires the test application to use this as well. When the bytes are created for a signal group and sent, all signals are converted to little endian. Similar when frames are received, the signal group expects the signals to be little endian.

5.5.4.3 Calibration Parameters

The calibration parameters parsed are visible in the calibration parameters table in the GUI. To load the parameters to the VFB the signals are saved in an XML file to a location on the disc where the VFB simulator loads the calibration parameters from. The calibration parameters were only used in the SIL phase.

6 Discussion

6.1 Model based development

Using MBD methodology has simplified the development process both in complexity and time. It is less time consuming to create models in Simulink that fulfils the requirements and to run the simulation to view the outputs of a system than by writing the code by hand.

When integration testing of the system is possible early on in the development process errors are detected early on and the cost of correcting errors are reduced. With MBD it is much easier to hide implementation details in subsystem and provide a higher overview of the system. This helps a lot when the system becomes more complex by clearly showing how different parts of the system are connected.

However with MBD the developer is losing the freedom of writing code by hand. Simple tasks for which only a few lines of code is required can take considerably more time to implement in a model, especially if a developer is not used to the MBD environment. Another possible issue is that often the generated code is heavily optimized and hard to read for a developer, e.g. it can be very hard to understand which part of a model that is the cause of some error detected in later phases (SIL or HIL) when the generated code is used.

6.2 Multiple instantiations and AFFE

One issue that was discovered concerning AUTOSAR and the AFFE project was that AUTOSAR only supports multiple instantiations of a SW-C when all instances are located on the same ECU. If the instances will be located on different ECUs it is not possible to use the AUTOSAR feature of multiple instantiation (5.1.). Instead each instance needs to be generated as a separate SW-Cs. The architecture for the AFFE project specified two control units and four wheel nodes, each of these should be located on separate ECUs. Therefor they have to be generated as separate SW-Cs, at least until the AUTOSAR standard supports multiple instantiation across several ECUs.

An issue that arises of having multiple SW-Cs generated from a common model is that the instance specific calibration parameters could be more complicated to provide in the SIL and HIL phases. If the SW-Cs are located on separate ECUs they can have the same name for these instance specific calibration parameters, since each ECU has its local calibration parameters. However when the system is simulated during the SIL phase there could be an issue if the VFB simulator cannot differentiate between each instantiations own calibration parameters. E.g. two SW-Cs, A and B, that have been generated from the same model uses an instance specific calibration parameter X. Even though A and B will be located on different ECUs, it will be an issue during the SIL phase if the VFB simulator only provides one copy of the calibration parameter X and forces both A and B to use the same value. It has to be known before generating code for different instances that uses instance specific calibration parameters how the VFB simulator handles this issue.

This can be viewed more generally as a problem in the SIL phase when all the SW-Cs are included in one system, then the name of e.g. functions, parameters, runnables and data types has to be unique not only within a SW-C but also within the entire system that is simulated.

6.3 MIL to SIL to HIL

6.3.1 MIL > SIL

Before generating code for the models is possible it is necessary to first configure the SW-Cs. The expected result was to find a path from the MIL to the SIL phase without modifying the models. However the results (section 5.2.1) shows that before it is possible to generate code for the SW-Cs the calibration parameters, instantiation of SW-Cs and function calls must be modified. Furthermore from modelling to code generation the internals of the SW-C such as the sample time did not follow automatically and needed to be set manually.

6.3.2 SIL > HIL

The issues encountered when moving from SIL to HIL (see sections 5.4.3.1, 5.4.3.2 and 5.4.4.1) were not related to the actual implementation of the SW-Cs but rather the configuration of the BSW modules. This was not only due to the low complexity of our SW-Cs, but also because the functional errors were discovered and corrected during the MIL simulations. With correct configurations and building properties the system could be built and the simulations in the SIL and HIL phases ran smoothly.

6.3.3 Tool chain

The tool chain that has been used during the three phases did not have support for all needed functionality for AUTOSAR. The MATLAB and Simulink versions which has been used and evaluated in this project does not have full support for all functionality in which is needed, such as code generation for different instantiation of SW-Cs and importing of SW-C description into Simulink MBD. The missing functionality support in the tool chain is because AUTOSAR standard is still young and therefor development tools such as Simulink and its toolboxes are still under development.

6.4 Testing during MIL, SIL and HIL

The earlier errors are detected during the development process the lower are the costs, both in money and time. By using different phases during development more parts of the final system is included at each step, e.g. the MIL phase consist of the SW-Cs, the SIL phase includes part of the RTE and the HIL phase requires both RTE and BSW layers to be included.

The testing process in this project consisted of verifying in each phase (SIL and HIL) that the SW-Cs had the same functionality as we observed during the previous phase. By creating the simulation environment in the MIL phase, it could be reused later on, either by generating the code and implementing in a separate test tool or by allowing for communication between the system and Simulink.

6.4.1 Improving the system test process

To improve the testing process further we would have liked to observe not only the input and output of the system but also the internal communication, e.g. during the SIL phase being able to view what is sent between the different SW-Cs, not only the CAN, Lin or FlexRay frames.

A possible improvement we investigated was to use Simulink during the SIL phase to simulate the input or output of the system. This can be done in different ways. In this thesis we developed a Java server that was executed in the MATLAB environment, it receives IP packets sent by the VFB simulator and triggers a MATLAB function which then updates the model environment. Also blocks can be used in Simulink that provides TCP communication for the models, unfortunately this was not available to us during this thesis and this possible method could not be investigated further.

Another issue that might arise is that the SIL and HIL phase uses different communication protocols to wrap the CAN frames in, SIL uses TCP and the HIL uses USB. Any testing environment would need to support both protocols in order to function. Also during HIL it could be different protocols depending on if FlexRay or Lin was used, that would further complicate the matter. A solution to this problem could be to instead use a proxy to translate the USB communication to the TCP protocol used by the VFB. This way the testing environment would only be required to support one protocol and therefore can be easier to integrate in the different phases of the development process.

7 Conclusion

This project has been profitable in the sense of knowledge in AUTOSAR and model-based development. We have gained more knowledge on how the AUTOSAR standard works and the possibilities with AUTOSAR and MBD.

With MBD the development process has been able to split up in different phases. It has been able to test the models in all phases to detect implementation errors in the model and verify that the model behaves the same in all phases. With MBD the time and cost in the development process can be reduced. The system and its behaviour are the same in all phases, it can therefore be possible to have the same test and simulation environment throughout all phases.

When modelling in Simulink model reference is more appropriate to use to instantiate a SW-C. There are minor issues related to model reference (see Appendix A – Modelling methods pros and cons) but to instantiate it remains the best choice.

It is however not possible to generate code for multiple instantiation of the SW-C and AUTOSAR only supports multiple instantiation if the instances are located on the same ECU using code sharing method. This requires instantiation to multiple ECUs to be done by generating separate SW-Cs for each instance. As discussed in section 6.2 this can cause issues in the SIL phase when the entire system is built into one application and names conflict.

Calibration parameters are supported both in AUTOSAR and when modelling Simulink, however we were unable to generate code for instance specific calibration parameters.

7.1 Future work

To continue on our work done in this project there are additional interesting topics that could be included.

Multiple ECUs can be used in the HIL phase, instead of one as done in this thesis, to better understand the steps required to go from SIL to HIL when the mapping of the ECUs is done.

Further we discussed ways of connecting Simulink to the SIL phase, where we created a Java server to show that it is possible. However there might be other ways to connect a simulation environment in Simulink to the SIL and HIL phases, which could be examined.

CAN is the only communication protocol used in this project, to increase the understanding of going from MIL->SIL->HIL it is possible to use additional protocols such as Lin or FlexRay.

References

[1] Climate change experts propose new measures to cut vehicle emissions [Internet]. [cited 2011 May 2]; Available from: http://www.automotiveindustrydigest.com/2011/07/07/climate-change-experts-propose-new-measures-to-cut-vehicle-emissions/

[2] AUTOSAR Introduction [Internet]. [cited 2011 May 2]; Available from: http://www.autosar.org

[3] Schwalbe, K. *Information Technology Project Management*. 5th ed. Cengage Learning. 2007.

[4] Fürst S. AUTOSAR – A Worldwide Standard is on the Road. 14th International VDI Congress Electronic Systems for Vehicles. 2009.

[5] AUTOSAR. AUTOSAR [homepage on the Internet]. [updated 2010 Feb 26; cited 2011 Jul 10]; Available from: http://www.autosar.org

[6] Technical Overview AUTOSAR [Internet]. [cited 2011 Jul 26]; Available from: http://www.autosar.org

[7] AUTOSAR, AUTOSAR Specification of RTE. 2011. p. 75-79

[8] Development of AUTOSAR Software Components within Model-Based Design [Internet]. [updated 2008 Jan; cited 2011 Aug 3]; Availible from: www.mathworks.com

[9] Model Based Development [Internet]. [updated 2010 Feb 26; cited 2011 July 10]; Available from: http://www.embedded360.com/model_based_development/model_based_development.htm

[10] Simulink – Simulation and Model-Based Design [Internet]. [cited 2011 May 21]; Available from: http://www.mathworks.com/products/simulink/

[11] Mecel Picea Suite [Internet]. [cited 2011 Jul 11]; Available from: http://www.mecel.se/products/mecel-picea/mecel-picea

[12] Simulink Model Reference [Internet]. [cited 2011 Jul 20]; Available from: http://www.mathworks.com

[13] Simulink Libraries [Internet]. [cited 2011 Jul 20]; Available from: http://www.mathworks.com

[14] Walker G, Friedman J, Aberg R. Configuration Management of the Model-Based Design Process. SAE International. Journal of passenger car: Electronic and Electrical systems. 2007; Section 7 - Volume 116

[15] Mathworks [Internet]. [cited 2011 Jun 2]; Available from: http://www.mathworks.com

[16] AFFE Introduction presentation [Mecel internal documents]. [cited 2011 Jul 01] ; Available from : http://www.mecel.se (not official).

Appendix A – Modelling methods pros and cons

Model Reference

Pros:

- Easy to modify sub-content, Hierarchy
- Easy to make different instances of a model block
- Good for model with many blocks
- Support standalone simulation
- Faster simulation time if models are not updated or if several instances exist
- Store component as separate mdl files and not with links
- Support for parallel development

Cons:

- Function call cannot cross the boundaries, bus-interface does not recognize the signal from higher level
- Slow initial builds, time consuming for prototyping
- Needs to have the SW-C as a subsystem inside the model

Libraries

Pros:

- Static – Well defined blocks that does not change alot
- Easy to use when comes to models with few blocks
- Suited for low level utility functions

Cons:

- Hard to edit sub content
- Slower simulation when larger models are used, each instance is built separately and each time simulation is started
- Instantiations is possible but it must be parameterized, which mean different block has different calibration parameters

Appendix B – Importing SW-C using PSP

Follow these steps to import the SW-CD into Simulink

- E.g. xml description file is: sofwarecomponent.xml
- Then type: myObject = arxml.importer('softwarecomponent.xml');
- To get all methods type: methods(myObject);
- If the Pilot Support Package is not installed with MATLAB version 2010a or beyond that then only these methods will be shown:

    - % createCalibrationComponentObjects
    - % createComponentAsModel
    - % createComponentAsSubsystem
    - % createOperationAsConfigurableSubsystems
    - % display
    - % getCalibrationComponentNames
    - % getClientServerInterfaceNames
    - % getComponentNames
    - % getDependencies
    - % getFile
    - % importer
    - % loadobj
    - % saveobj
    - % setDependencies
    - % setFile

- If the Pilot Support Package (PSP) is installed these extra methods can be found also.

    - % addDataType
    - % createComponentWithInternalBehaviorAsSubsystem
    - % createEnumClasses
    - % createOperationAsConfigurableSubsystems
    - % createPerInstanceCalibrationObjects
    - % createSharedCalibrationObjects
    - % getInternalBehaviorNames

- Use % createComponentWithInternalBehaviorAsSubsystem to import the SW-CD with internal behaviour and creating skeleton of the SW-C
    - SW-Cname = getInternalBehaviorNames(myObject)
    - createComponentWithInternalBehaviorAsSubsystem(SW-Cname)

Appendix C – Settings in Simulink Coder

Code generation

When generating code, the variables in the Model workspace must be moved to Based workspace otherwise these variable will not be generated. Furthermore mask parameters created must be removed in order to code generation with AUTOSAR compliant.

Configuration in Configuration Parameters needs to be done before generating code for the SW-Cs.

Configuration options:

- Type: Fixed step
- Solver: Discrete (no continuos states)
- Inline parameters – should be checked
- Target file: autosar.tlc

AUTOSAR Code Generation options:

- Ports should be choose as explicit send and receive
- Check runnable names and set sample time as desired
- Validate configuration