Dynamic Architectural Simulation Model of YellowCar in MATLAB/Simulink Using AUTOSAR System

Master Thesis

Submitted for the Fulfillment Requirements for the Academic Degree M.Sc.

Dept. of Computer Engineering Faculty of Computer Science

Submitted by: Saeed Soltani
Student ID: 335395
Date: 20.07.1987

Supervisors: Prof. Dr. W. Hardt
M. Sc. Owes Khan
Abstract

The YellowCar at the professorship of computer engineering of TU Chemnitz is a demonstration vehicle. The car is equipped with multiple networked Electronic Control Unit (ECU)s. There are regular software and hardware updates. Before introduction of any new update, it is essential to test the behavior of the car. This can be done through simulation. Since the majority of the ECU in YellowCar are AUTOSAR based, several AUTOSAR simulation tools can be used to do so. However non-AUTOSAR ECU applications can still not be simulated in these tools. Moreover simulating with such tools need the whole application to be implemented and also very expensive.

Simulink is one of the most powerful tools for the purpose of Model-in-the-Loop (MIL) testing which is a popular strategy in the embedded world. The scope of this Master thesis is analyzing the YellowCar and its architecture to develop a dynamic Simulink architectural model that can be modified and extended to facilitate future updates.

The outcome of this thesis is an implementation of a model for the YellowCar which allows both AUTOSAR and non-AUTOSAR ECUs to be simulated as one system. Also the model supports extension by easy addition of new modules like ECU or sensor through a graphical user interface.

Keywords: Simulation, MATLAB/Simulink, AUTOSAR, MIL
# Contents

Contents ................................................................. ii

List of Figures .......................................................... iv

List of Abbreviations ................................................. vi

1. Introduction ......................................................... 1
   1.1. Simulation ................................................... 1
       1.1.1. Background in Automotive Industry ................. 2
       1.1.2. Advantages and Disadvantages ....................... 2
   1.2. YellowCar ................................................... 3
       1.2.1. Technical Points ..................................... 3
       1.2.2. Construction .......................................... 4
   1.3. AUTOSAR .................................................... 5
       1.3.1. Purpose ................................................ 6
       1.3.2. AUTOSAR Architecture ............................... 6
   1.4. Model-in-the-Loop .......................................... 9
       1.4.1. Advantages ............................................ 9
       1.4.2. MATLAB ............................................... 11
       1.4.3. Simulink .............................................. 11
   1.5. Problem Statement ......................................... 11

2. State of the Art ...................................................... 13
   2.1. TargetLink .................................................. 14
   2.2. ASCET ........................................................ 15
   2.3. VAP ............................................................. 16
   2.4. MATLAB/Simulink .......................................... 16

3. Concept ............................................................... 19
   3.1. Simulation of the YellowCar .............................. 20
   3.2. Model Analyzer/Generator ................................. 29
       3.2.1. Objective .............................................. 29
       3.2.2. Algorithm Conceptualization ....................... 30
4. Implementation and Evaluation of the Solution .......................... 33
   4.1. Implementation of the YellowCar MBD in Simulink .................. 33
      4.1.1. Sensors .................................................. 34
      4.1.2. Actuators ................................................. 36
      4.1.3. ECUs ..................................................... 39
      4.1.4. YellowCar Library ....................................... 52
      4.1.5. Online/Offline Mode ..................................... 53
   4.2. Implementation of Model Developer Tool in MATLAB ............... 54
      4.2.1. Sensor Creation ......................................... 55
      4.2.2. ECU Creation ............................................ 55

5. Testing and Results ............................................................. 61
   5.1. Offline Mode (MIL test) ......................................... 61
   5.2. Online Mode (Verification of Implementation) ...................... 65

6. Conclusion ............................................................................... 68
   6.1. Summary .............................................................. 68
   6.2. Advantages ............................................................ 69
   6.3. Future Work ................................................................ 70
      6.3.1. Automatic Port Connection .................................. 70
      6.3.2. Communication with Other Popular Networks ............... 71
      6.3.3. Increasing Speed of Simulation .............................. 71

Bibliography ............................................................................. 73

A. APPENDIX A- Integration of existing C/C++ code into Simulink model 75
   A.1. Manual MEX file .................................................. 75
   A.2. Legacy Code Tool .................................................. 75
List of Figures

1.1. YellowCar Picture [29] .................................................. 3
1.2. YellowCar Construction [29] ......................................... 5
1.3. AUTOSAR Architecture [3] ........................................... 8
1.4. MIL in the Simulation process [25] ................................. 10

2.2. AUTOSAR tool chain between TargetLink and SystemDesk [6] . 15
2.3. AUTOSAR Standardard Supports by MathWork [8] ................ 17

3.1. Illustration of Concept .................................................. 20
3.2. AUTOSAR Bottom-Up workflow [8] ................................. 24
3.3. AUTOSAR Top-Down workflow [8] ................................... 25
3.4. AUTOSAR Round-Trip workflow [8] .................................. 25
3.5. Inter-ECU communication example [14] .............................. 26
3.6. Interaction of VNT between MATLAB/Simulink and CAN bus [17] . 27
3.7. Conceptualized internal architecture of ECUs and their connectivity to the sensors and actuators in Simulink model ...................... 28
3.8. Conceptualization diagram for process of ECU creation ............. 32

4.1. Implemented model of YellowCar in Simulink environment ........ 34
4.2. Random Integer Generator block and its dialog box ............... 37
4.3. Table of implemented actuators ....................................... 39
4.4. AUTOSAR compliant workflows for enabling Round-Trip engineering [20] ................................................................. 42
4.5. Internal view of an imported AUTOSAR SWC (DistMgr) of YellowCar 44
4.6. Data transmission between AUTOSAR based ECUs of YellowCar . 46
4.7. Internal view of Assistant ECU architecture .......................... 49
4.8. Internal view of Function-Call Light ECU 1, of Assistant ECU ...... 50
4.9. CAN Unpack dialog box .................................................. 51
4.10. YellowCar Library, internal components of Function-Call Light ECU . 53
4.11. Model Developer Tool in detail ........................................ 55
4.12. Process of ECU Creation ................................................ 60

5.1. Offline Mode, Assistant ECU recognized an obstacle ............... 62
5.2. I/O analyzing, ultrasonic sensors and Assistant ECU output ........ 63
5.3. Result of Feature ECU, all lights are on ............................... 64
5.4. I/O analyzing, light sensor and Feature ECU output .................. 65
LIST OF FIGURES

5.5. Online Mode, Result of Assistat ECU with steering functions . . . . . 67
List of Abbreviations

VFB  Virtual Function Bus
RTE  Run-Time Environment
BSW  Basic Software
SWC  Software Component
MIL  Model-in-the-Loop
HIL  Hardware-in-the-Loop
PIL  Processor-in-the-Loop
SIL  Software-in-the-Loop
MBD  Model-Based Design
MBT  Model-Based Testing
AUTOSAR  AUTomotive Open System ARchitecture
AAT  AUTOSAR authoring tool
Arxml  AUTOSAR XML
XML  Extensible Markup Language
ASCET  Advanced Simulation and Control Engineering Tool
VAP  Virtual Verification Platform
CAN  Controller Area Network
dbc  Database Container
LIN  Local Interconnect Network
LDF  LIN Description File
MCAL  Micro Controller Abstraction Layer
I/O  Input/Output
List of Abbreviations

GUI  Graphical User Interface

E/E  Electric/Electronic

ECU  Electronic Control Unit

VNT  Vehicle Network Toolbox

USB  Universal Serial Bus

MATLAB  MATrix LABoratory

MEX  MATLAB Executable

API  Application Programming Interface

XCP  Universal Measurement and Calibration Protocol (The ‘X’ stands for the variable and interchangeable transport layer)
1. Introduction

This chapter gives an introduction to following topics and discusses about each comprehensively:

- Simulation
- YellowCar
- AUTOSAR
- Model-in-the-Loop
- Problem Statement

All the domains mentioned above are prerequisites to understand the main scenario of this Master thesis.

1.1. Simulation

In today’s automotive software and electronics, there are many goals that have to be reached to achieve a competitive system. For instance, progressing the electronic technology causes changing of any network components to bring the system up to date. The story of regular hardware update is true for updating of software as well. In any automotive system, it is essential to test the behavior of the system, before introduction of any new update. In this case simulation of the system is the solution.

The word ’Simulation’, based on Shannon (1998) is described as, the process of designing a model of a real system and conducting experiments with this model for the purpose either of understanding the behavior of the system or of evaluating various strategies (within the limits imposed by a criterion or set of criteria) for the operation of the system [28]. In other words simulation is the discipline of designing model of a physical system, either theoretical or actual, to analyze the output of system. Simulation is used in all cases that examination of real system would be impossible or impractical. Such situations can be caused by many reasons like: high cost of prototyping and testing, impractical process duration of the experiment in real time, high risk or danger of testing on actual systems, or impossible extensive tests in fragile systems [26].

There are variety of fields in which simulation is being used, from video games, training and education to high level scientific modeling of natural system. In the following a brief background of simulation in automotive industry and its objectives are discussed.
1. Introduction

1.1.1. Background in Automotive Industry
Now a days, the strong worldwide competition among the automobile manufacturers, suppliers and related companies in automotive industries result a large scale of variety of automobiles that have to be developed in the best possible way in less possible time [30]. It is obvious that the classical methods of automobile development from designing to production is no longer feasible in such a great competition. Therefore the solution could be development of simulation model of a vehicle in any different field of automotive like electronic, mechanic and etc.

1.1.2. Advantages and Disadvantages
Simulation has proven to be a low-cost analysis tool for experimentation in low-risk environment. Although simulation is helpful for scientists, researchers, engineers and managers to make decisions faster, cheaper and more reliable, but obviously they also come with a set of advantages and disadvantages [28]. The following parts discuss the general advantages and disadvantages of simulation categorized in three different broad areas of technology, process and socialization.

Technology
Simulation has a great advantage over data analysis methods which are limited to forecasting effects of similar events that are already happened in the past. But the problem in this case is that high level of theory and causal hypothesis is needed [5].

In research phase there are variety of questions which can be answered by highly flexible techniques and approaches provided by simulation. Although simulation is incredibly creative and in variety of contexts can be applied, but often no standardized approach will be achieved because a formalized set of best practice is not readily available [5].

Process
In order to solve the problems in case of data limitation availability, simulation can be used as an excellent tool. However in comparison to data analytics the process of validation of the given answers is a disadvantage for the simulation, because to achieve great high degree reliable answers usually multiple data sources needed. Therefore often no standardized approach will be achieved [26].

The main goal of simulation can be testing and experimenting a system in a controlled and low-risk environment, before the production phase. Simulation is able to run variety of what-if scenarios but because of time and resources limitation, it is important to be aware of potential of scope creep in the project [5].
Socialization

In comparison to the real world experimenting, the cost of simulation is extremely less in both time and resources needed, but on the other hand there is still high level of skepticism in simulation. However this skepticism can be a result of novelty of simulation. Developers believe that, by achieving more success and validated forecasts day by day, this disadvantage of simulation will subside [5] [15].

1.2. YellowCar

The YellowCar is developed as part of a research work at the professorship of computer engineering in TU Chemnitz. The YellowCar is a demonstrator vehicle in field of Automotive Software Engineering. The car is equipped with multiple ECUs, sensors and actuators. Through multiple networked ECUs, various simple vehicle functions can be implemented. Basically the main focus of this project is not on the actual functions of cars, but it is generally on the automobile software architecture. On this demonstration vehicle real software architectures can be examined and tested [29].

1.2.1. Technical Points

The main technical points of the YellowCar are listed as:

- Hardware setup: three networked ECUs
1. Introduction

- Communication medium: CAN bus
- Software system architecture: AUTOSAR
- Sensors and actuators
- Non-AUTOSAR boards
- Car2Car communication
- Remote controller

However in this report, the last two of the above mentioned technical features of the YellowCar are not described in detail. Although the Car2Car communication and remote controller are completely developed, but since we are focusing on hardware and software architecture, the underlying technologies are not relevant. Thus, these two features are not on the focus of this project and just mentioned as technical points which are already developed and available on the YellowCar.

The main domain of this Master thesis is mostly on basic architecture of YellowCar including, sensors and actuators and networked ECUs (AUTOSAR and non-AUTOSAR based) communicating through CAN bus.

1.2.2. Construction

The YellowCar consists of three main control units (ECUs):

- Processing ECU
- Feature ECU
- Assistant ECU

All controllers boards are type Freescale S12XEP100. Howbeit, they are newly updated to version SPC560P. The software part of all three ECUs is programmed and developed based on AUTOSAR standard. AUTOSAR is a software architecture standard in automotive industry and is explained in the next section exclusively.

In the YellowCar each ECU has a clearly defined tasks and communicate with network of ECUs via CAN bus (125 Kbit/s). As it is shown in Figure 1.2, the Processing ECU is responsible for reading the sensor signal values and distributing them to the Feature ECU and Assistant ECU. The Feature ECU controls the indicator signals (left, alarm, right) and also the dipped headlights. The Assistant ECU is responsible for the implementation of drive control processes and also it controls the motors for steering and drive [29].

Although all three mentioned ECUs are programmed based on AUTOSAR standard (version 2.1 and the newer version which is 3.1), there are also two other non-AUTOSAR based ECUs available on YalloeCar named Planet boards and Raspberry
1. Introduction

Figure 1.2.: YellowCar Construction [29]

Pi. Existing both kind of standardized and non-standardized ECUs in the YellowCar limits the choices for simulation tools. The chapter 'State of the Art' discusses this issue exclusively and compare the advantages and disadvantages of available well-known automotive simulation tools.

1.3. AUTOSAR

This section introduces the AUTOSAR standard from the basic level of understanding. Based on the scope of this Master thesis the following aspects of AUTOSAR are explained more comprehensively:

- Purpose
- AUTOSAR Architecture
- Application Layer
- Virtual Function Bus

As in previous section discussed, the main ECUs of YellowCar project are developed with an automotive standard called AUTOSAR. The word AUTOSAR stands for AUTomotive Open System ARchitecture, developed with partnership of different automobile manufacturers, suppliers and many other hardware and software developer companies.
1. Introduction

1.3.1. Purpose

In today’s automotive industry mastering with growing complexity of Electric/Electronic (E/E) architecture is becoming a big challenge and needs a technological breakthrough. For instance, increasing the number of network components in an automotive system leads to a level of complexity which may be impossible to handle applying traditional development processes. In addition, to adapt existing solutions to different environment, much more resources and efforts have to be spent [13].

In order to cope with these problems, AUTOSAR standard has been formed. The main objective of this cooperation was to establish an open industry standard for the automotive software architecture for suppliers and manufacturers. The main goals of AUTOSAR according to the motto can be concluded as fulfillment of following points:

- Definition of an open architecture supporting various functional domains [2]
- Standardization of basic software functionality of ECUs [2]
- Quality and reliability improvement of E/E systems [23]
- Maintainability and software updates and upgrades over whole product life cycle of a vehicle
- Flexibility for product modification and upgrading [23]
- Availability, safety, transferability, scalability and reusability of functions (software)
- Cost optimization of scalable systems [23]

The main goals of AUTOSAR partnership in a nutshell can be considered as modularity, transferability, scalability and re-usability of functions [2].

1.3.2. AUTOSAR Architecture

In order to achieve the above mentioned goals, AUTOSAR provides a layered architecture with standardized interfaces for different layers.

According to Figure 1.3, AUTOSAR software architecture is a layered architecture distinguishes between three main layers including: application layer, runtime environment and basic software, running on an ECU hardware (microcontroller) [13]. These layers encapsulate the functionalities which can be used by all AUTOSAR applications. This whole combination is offering all mechanism needed for independency of software in automotive systems. To say in simple words, the goal of layered architecture of AUTOSAR is to abstract software components [2].

In this report, based on the project-scope of this Master thesis, some sections of AUTOSAR architecture are discussed more than the others. The main activity of
1. Introduction

this thesis work is done mostly on the Application layer of AUTOSAR. Thus, this AUTOSAR layer and its relative aspects are more on the focus of the report and the other layers are just in the next two paragraphs shortly introduced.

The layers of AUTOSAR architecture have different specifications and responsibilities. According to Figure 1,3, between the upper most layer (AUTOSAR software) and the hardware (microcontroller), there are two other layers:

- AUTOSAR Basic Software (BSW)
- AUTOSAR Run-Time Environment (RTE)

Basically these two layers are responsible for the abstraction of hardware and the software layer [2]. The RTE as the central connecting element, provides capability for communication between the components of software layer of the same ECU or software components of different ECUs [3]. In other words, all interaction between AUTOSAR Software Components (SWCs) or atomic SWCs is routed through RTE. The BSW layer, located below the RTE, contains ECU specific modules and allows the external communication of components between different ECUs over a bus networks like CAN, FlexRay or LIN [3] [4].

AUTOSAR Application Layer

The upper most layer in AUTOSAR architecture is the AUTOSAR Application layer. As previously mentioned, this layer of AUTOSAR architecture is mainly related to the scope of the implementation of the Master thesis.

The application layer contains all AUTOSAR SWCs which are mapped on an ECU [23]. This section describes three important related parts of AUTOSAR Application layer including: the AUTOSAR Software Components, AUTOSAR Ports and Runnables. At the end of this subsection the Virtual Function Bus (VFB) is introduced briefly.

AUTOSAR Software Components

AUTOSAR standard organized a common software infrastructure based on standardized interfaces. An application software in this standard is defined as self-contained unit called AUTOSAR Software Component (SWC) [2]. Basically each SWC as an atomic piece of software, encapsulates an application running on the AUTOSAR infrastructure [23]. In other words, they contain the implementation of their functionality and behaviors.

There are different kinds of AUTOSAR SWCs. Application SWCs are location independent and Sensor-Actuator SWCs are location dependent because they are dependent on ECU hardware. Hence, instances of Application SWCs can be simply deployed to any different ECUs. However to design AUTOSAR application for Sensor/Actuator SWCs, software designers have to know which hardware specification is used for sensors and actuators [23].

7
1. Introduction

AUTOSAR Ports

In order to connect the SWCs, AUTOSAR expose well defined ports through which communication with other software component or with basic software layer can happen. This connection points which communicate over the Virtual Functional Bus (VFB) to outside world called PortPrototypes. The duty of communication is implemented by the Run-Time Environment [2]. There are two different types of port; a Pport (Provider port) and Rport (Receiver port) [23].

Runnables

In AUTOSAR standard, the SWCs cannot directly access the underlying hardware or operating system. Therefore the functionalities of SWCs are wrapped into so-called runnables and each SWC can consist of arbitrary number of these runnables.
1. Introduction

Basically runnables are piece of functionality which has to be executed during the runtime of related SWC. These executable codes can be called upon by RTE to perform a certain task [23] [2].

Virtual Function Bus

AUTOSAR defines a layered software architecture to omit the dependencies of hardware and software. This abstraction is required for all the communication of in and out of SWCs as well. To fulfill this goal, the Virtual Function Bus provides a standardized interface for all the SWC communication. The VFB facilitates the concept of transferability within the AUTOSAR software architecture. This approach allows the SWC developers to concentrate just on the implementation of the application without caring about the specification of underlying hardware [22].

1.4. Model-in-the-Loop

During recent years, automotive embedded industry has experienced a great methodological revolution by changing from an electrical and mechanical engineering discipline to a combination of these two with software engineering and it has happened by integrating the modeling as an engineering and development tool[7]. As a result of this revolution and growth of software, model based technologies (design and test) became a part of development process in automotive industry [31][27].

In development environment, use of modeling and simulation facilities enables developers to check the result of development faster and perform the adjustments and modifications simpler.

As it is shown in Figure 1.4, automotive industry applies test and simulation environments at several levels of abstraction [25]. MIL is a fast and cost-effective simulation strategy in embedded world which can be performed in an early development phase [7]. In other words, Model-Based Testing (MBT) is system-level testing using simulation of models where no hardware exists.

The simulation of YellowCar as the main focus of this Master thesis is done in a MIL simulation. Using this systematic method enables us to define and generate test cases, evaluate the requirements and verify our design algorithm.

1.4.1. Advantages

MIL test has a number of potential advantages which depend directly on the application. Firstly MIL test tools using graphical models and simulation of vehicle models allow a better understanding of the system and enables more agile testing and error correction. Therefore MIL is time saving. Additionally since high-level models can be developed in very early design phase and very large account of tests can be generated (system validation), it reduces costs as well [12][7]. The advantage
1. Introduction

Figure 1.4.: MIL in the Simulation process [25]

in savings by using model-based development over hand written code can be as much 36 percent according to some studies [24].

Moreover MBD allows users to improve efficiency by [18]:

- Using a common design environment across project teams
- Linking designs directly to requirements
- Refining algorithms through multi-domain simulation
- Generating embedded software code
- Developing and reusing test suites
- Generating documentation
- Reusing designs to deploy systems across multiple processors and hardware targets
- Integrating testing with design to continuously identify and correct errors

Regarding development time and costs, the YellowCar project is not an exception. There are regular software and hardware updates in YellowCar and before introduction of any new update, it is essential to test the behavior of the car. The process of testing and validation of system before any real hardware device can be done faster and cheaper through MIL test on simulated model.

In the following a brief introduction to MATLAB and Simulink as a tool for MIL test and chosen for implementation of this Master thesis. In the next Chapter, State
of the Art, some of well-known Model-Based Design (MBD) tools in automotive industry are discussed and compared with MATLAB/Simulink and the main reasons of using MATLAB/Simulink for this project are outlined.

1.4.2. MATLAB

The name MATLAB stands for MATrix LABoratory and introduced by MathWorks Company [11]. MATLAB is matrix-based language of technical computing for algorithm development, data analysis, visualization and numeric computation. MATLAB is a worldwide popular language among millions of developers and scientists. It provides world’s most natural way to express computational mathematics [21]. Also some MATLAB’s built-in tools like Simulink enables developers to visualize the system and gain insight of data [21].

1.4.3. Simulink

The main chosen tool for implementation of this project and basically simulation of YellowCar is Simulink. It is an add-on integrated with MATLAB and provides an interactive block diagram environment for multi-domain simulation and MBD. Simulink facilitates continuous test and verification of embedded systems, modeling, simulating, automatic code generation and analyzing of dynamic systems. Since it is integrated to MATLAB, the data can be easily shared between to programs and this enables user to incorporate variety of MATLAB algorithms and vast library of prebuilt toolboxes into models in Simulink and the exported result of simulation can be imported to MATLAB for further analysis. Simulink provides an environment for developers as a laboratory which enables them to model, simulate and analyze systems that could be impossible or impractical in real word environment [21][10][24].

This environment is extensively used in many different application domains. According to REF designing and modeling of behavioral part of more than 50 percent of automotive control units are done in Simulink. The combination of MATLAB and Simulink provides a powerful tool with lot of beneficial abilities for many different areas. But the question ’Why is MATLAB/Simulink chosen for implementation of this project?’ still does exist. In order to clarify the reasons, more related capabilities and strength of this tool needs to be outlined. In the next chapter, State of the Art, the special features of Simulink are discussed precisely and in the chapter 'Concept’ it will be cleared why Simulink is the best choice for the objectives of this Master thesis [10][24].

1.5. Problem Statement

As the subject name of this project suggests, main goal of this master thesis is to design a model of YellowCar to simulate its architecture/behavior and apply MIL test on it. The main reason of doing that is the regular software and hardware
updating on YellowCar and an essential need for testing the behavior of the car before introduction of any new update.

Thus, there is a need of a simulation tool for all aspects of the YellowCar with the least possible effort. Since in this approach it is tried to provide the simulation model as generic as possible, this strategy can also be applied to normal automotive applications. In other words, the YellowCar as a demonstration vehicle acts only as a representation of the automotive sector and this trend can be expanded to the normal automotive applications.

Nevertheless, till now there has been no MIL test project especially on the YellowCar and this Master thesis is the first attempt to simulate architecture of the YellowCar in a MBD tool.

There are lot of available MBD and MBT application/tools providing powerful environment for simulation of an AUTOSAR based automotive system. However these applications are not able to simulate the non-AUTOSAR ECU application. Although the YellowCar is also an AUTOSAR based system and the majority of the ECUs are AUTOSAR based, but there are still non-AUTOSAR ECU applications available on this vehicle which cannot be simulated in those tools. Moreover working with such tools need high degree of knowledge about AUTOSAR standardized systems. In addition simulating with such tools need the whole application to be implemented and also very expensive.

Therefore there is a need to simulate the YellowCar in a MBD environment like Simulink supporting both kinds of ECUs with and without standardized software architecture. Also simulation with a graphical environment like Simulink is not only cheap but also its Integration with MATLAB provides a powerful tool. In the next two chapters it will be cleared how the combination of these two tools could be helpful for the implementation of this thesis work.
2. State of the Art

The main domain of this Master thesis can be defined as development of a MBD/MBT of an automotive system supporting AUTOSAR and CAN standards as the main automotive standards used in YellowCar.

Since there are AUTOSAR and non-AUTOSAR based ECUs on the YellowCar, a tool supporting only AUTOSAR systems would not be sufficient. Therefore the corresponding tool must support both sides correctly. Also easy introduction of new Sensors and actuators are important. In addition, in close future there would be some changes in communication standard of YellowCar. Therefore the chosen tool not only have to support CAN communication but also must provide some other possibilities in case of any future model update. Totally, a significant factor which has to be satisfied is flexibility of the model for addition of network, hardware and software features.

The Figure 2.1 illustrates the AUTOSAR software architecture which has been briefly introduced previously. The marked part (with green rectangle) addressed as application software component implementation, shows the level of abstraction in which we are going to talk about. This AUTOSAR layer is Application layer in which the MBD tools will be used to model the behavior of AUTOSAR SWCs, provide implementation and test in MIL simulation [6].

![Figure 2.1.: Field of Application SWC Implementation/Development [6]](image-url)
2. State of the Art

There are several tools available for performing MIL test in the automotive systems with variety of features and capabilities. In the following, three most powerful MIL test tools in field of automotive supporting AUTOSAR standard are discussed. At the end, the advantages and disadvantages, comparison of these tools, the reason of choosing MATLAB/Simulink as the simulation tool for this Master thesis is concluded.

2.1. TargetLink

One of the most popular tools for MIL in automotive area is TargetLink developed by dSPACE Company. TargetLink is basically a code generator tool which generates code directly from an existing model of MATLAB/Simulink/Stateflow for mass production [6]. This code generator tool provide very efficient C code for implanting control functions. It can be also integrated into another existing development environment. Therefore it enables developers to exchange the models between development departments. For instance, the exchange of model between software production and rapid prototyping team can be done without any need to model conversion.

According to the AUTOSAR methodology, the architecture part of an AUTOSAR based system and the behavior part are designed in two different tools. An AUTOSAR authoring tool (AAT) like dSPACE SystemDesk is where developers (software architect) can design the basic architecture of AUTOSAR system and behavior modeling tool like TargetLink is where the function designer (software developer) define the behavior for the corresponding architecture. These two AUTOSAR tools work together by exchanging standardized AUTOSAR system description file called AUTOSAR XML (Arxml).

Figure 2 illustrates an AUTOSAR tool chain in which TargetLink is taken as behavior modeling tool in combination of an AAT (SystemDesk in this example). As it is shown, the SWC description based on the standardized AUTOSAR XML format can be exported from the AAT. TargetLink enables function designing, function simulation, function implementation and subsequently AUTOSAR compliant code generation. Typically, the process of SWCs development is an iterative process. It means, the Arxml files can be exchanged several times between both tools to accommodate changes [6].

TargetLink is known as a proven software system and has been used in many vehicle production projects in the world. Its powerful software design facilities provide highly efficient, readable, configurable and traceable class code for developers. It is not only a strong partner tool for software verification of MIL simulation, but also it supports Software-in-the-Loop (SIL) and Processor-in-the-Loop (PIL) concepts in automotive industries as well [6].

However TargetLink generates very efficient C code but it is important to mention that it particularly supports the AUTOSAR based system. In means TargetLink can be only useful in models which are implemented in form of AUTOSAR SWC
2. State of the Art

Another well-known tool for automotive software development and especially MBD is Advanced Simulation and Control Engineering Tool (ASCET). This tool is a product software family produced by ETAS Company and has been used successfully since 1997. This software system provides development of MBD of automotive software with real-time requirements. Also like other tools like TargetLink, production of code generation is one of the main aspects of this tool. ASCET as a fast and reliable tool generates highly efficient and configurable ECU production codes from the models. This MBD tool supports importing Simulink MBD environment and development of AUTOSAR SWC as well as integration of existing C code. The long term experience of ASCET in more than 68 million ECUs all around the world ensures the reliability of this tool among the automotive software developers. ASCET is specifically developed to meet special requirements of automotive safety-
sensitive software which requires more efficiency and real-time restrictions like brake or steering systems [1].

2. State of the Art

2.3. VAP

Another approach for applying MIL test in field of automotive embedded systems is called Virtual Verification Platform (VAP) introduced by MBtech group. VAP is a software tool mostly used for integration, testing and validation of ECU software for hardware independent systems. This platform is able to execute and test AUTOSAR software long before availability of targeted ECU [16].

The VAP-system consists of a Linux based PC and an AUTOSAR basic software stack. The communication part supports automotive communication standards like CAN bus or FlexRay. Depending on the communication standard, it can be connected to vehicle via corresponding interfaces PC cards. In VAP-system, implementing a Micro Controller Abstraction Layer (MCAL) with compatible AUTOSAR BSW enables using the cards in the AUTOSAR environment [16].

MBtech introduces another product integrated to VAP called PROVEtech:RP (rapid). This is basically a PC-based system and a simple and flexible software solution for rapid prototyping AUTOSAR based systems. PROVEtech:RP enables users to integrate, test and validate AUTOSAR SWCs on PC platforms. Additionally it provides real-time integration to replace one or more ECUs either in the Hardware-in-the-Loop (HIL) system or even directly in the test vehicle. It is also possible to integrate AUTOSAR SWCs on PROVEtech:RP [16].

Generally VAP is a software system for early phase development process and it well suited for testing new algorithms mixing with new technologies. VAP-system provides variety of advantages and values for developers. Supporting AUTOSAR standard and hardware dependent drivers (MCAL) with AUTOSAR conform interfaces are the most known characteristics of this platform. Due to easy configuration, this software framework has the benefits of fast integration to existing on-board networks like CAN and FlexRay. It is based on industrial PC with Linux operation system and pre-emptive real-time Linux kernel supporting digital and analog Input/Output (I/O) [16].

2.4. MATLAB/Simulink

There are several facts which proves the strength of Simulink as one of the most powerful tool for multidomain simulation and MBD. By providing a graphical editor for building block diagram models, Simulink allows developers to design the system as simple as possible and since the models are hierarchical, models can be built using both top-down and bottom-up approaches. Additionally it includes a comprehensive libraries in variety of fields of automotive area.

One of the most distinguished aspects of Simulink is because of its integration to
2. State of the Art

MATLAB. Hence, the developers can switch easily between both sides. The shared data between MATLAB and Simulink enables developers to incorporate variety of MATLAB algorithms and vast library of prebuilt toolboxes into models in Simulink and also the exported result of simulation can be imported to MATLAB for further analysis. This integration of Simulink and MATLAB is not only useful for the modeling but also for simulation. The dynamic behavior of the designed system can be simulated using a choice of mathematical integration methods. This testing can be done either by entering commands in the MATLAB or directly in Simulink [18][24].

The result of simulation can be easily monitored using scopes or display blocks.

In case of development of AUTOSAR systems, Embedded Coder support package of MATLAB/Simulink provides facilities to model and simulate AUTOSAR SWCs. In order to realize, test and implement AUTOSAR SWCs, they can be either imported from AUTOSAR system description file (Arxml file) coming from an AAT like SystemDesk or it can be developed from a basic model in Simulink. As a result the SWCs can be exported from Simulink as XML file as well [8].

Embedded Coder support package also enables automatic generation of AUTOSAR production code using SIL and PIL simulations. In other words, this support package enables developers to generate a description for SWC as an algorithmic code. The generated code can be used for either testing in Simulink or integration into the AUTOSAR RTE [9].

In Figure 2.3, main approaches regarding AUTOSAR standard introduced by Mathworks are listed [8].

![AUTOSAR Standard Supports by MathWork](image-url)

Figure 2.3.: AUTOSAR Standard Supports by MathWork [8]
2. State of the Art

In comparison to the other introduced tools for MIL simulation, one of the most specific characteristic of the Simulink environment which makes it more particular for this project is capability of implementation both AUTOSAR and none-AUTOSAR systems in one model. In case of automotive systems with both AUTOSAR and non-AUTOSAR ECUs like YellowCar, there would be lack of support for implementation of normal ECUs in the other tools like TargetLink. However Simulink provides an environment for both kind of ECUs simply.

Moreover, MIL simulation with such tools like TragetLink, ASCET or VAP, need the whole application to be implemented and also very expensive. However in Simulink a model can be partially designed, implemented and tested separately. This capability provides a great possibility to run the simulation in each phase of simulation and debug the errors and change the design as intended. Correcting errors in very early phase of design reduces huge amount of human efforts and extra costs as well.

Additionally in this Master thesis the integration of Simulink with such a powerful tool like MATLAB provides a great facility to implement a graphical user interface (acsGUI tool) with several capabilities like creating ECUs, Sensors, importing AUTOSAR SWCs and so on. The acsGUI tool written in MATLAB language can be implemented using GUIDE which is a graphical user interface development environment of MATLAB.
3. Concept

According to topic of this Master thesis Dynamic Architectural Simulation Model of YellowCar in MATLAB/Simulink Using AUTOSAR System, the basic idea of this project consists of two main parts:

- Simulation of the YellowCar
- Model Analyzer/Generator

First part as the main concept is modeling and simulation of the architectural model of YellowCar in Simulink and the second one is implementation of a model analyzer/generator using MATLAB functions to make the simulated model easy extendable.

The Figure 3.1 illustrates the main concept of this project generally. On one hand there is YellowCar model in Simulink environment (right side). On the other hand, a combination of a so-called model analyzer and a model generator is placed under control of a Graphical User Interface (GUI) tool. As it is illustrated, some configuration and description files like: sensor configurations, system architecture description, hardware information and network database file can be imported to this part. Each of these files contain their own generic system specifications and will be utilized in an especial part of the model. For instance, in order to simulate the original architecture of the YellowCar, it is necessary to use its original system architecture which is based on AUTOSAR standard. In this case, the AUTOSAR description file (exported from an AAT) has to be imported to the model to implement and develop the ECUs based on the AUTOSAR system architecture. As another example, to build the communication system, the original messages and signals which have to be transferred through bus are needed. These configurations can be achieved by importing network database file of the relative communication standard which in case of YellowCar, it is CAN database file.

After processing the inputs by model analyzer, the model generator creates some models and add them to the YellowCar model. The main production of model generator can be ECUs (both AUTOSAR and non-AUTOSAR) and sensors. Additionally, the model generator must provide a facility to import the original AUTOSAR SWCs of YellowCar as well. They can be imported directly into the model, either in an existing ECU in the YellowCar or in a new ECU being generated. Also as it is shown the models can be imported to the YellowCar model through user-defined Simulink library. At the end, the production of AUTOSAR code and XML code generation from the Simulink model can be the final result of the Simulation. In the following parts, these two main concepts are fully described in detail separately.
3. Concept

![Figure 3.1.: Illustration of Concept](image)

### 3.1. Simulation of the YellowCar

As it is mentioned in the introduction part about the YellowCar, this demonstration vehicle is equipped with several ECUs, sensors and actuators. Although the ECUs are mainly AUTOSAR based, but there are also some other boards like Planet boards and Raspberry Pi available in YellowCar having no standard software architecture. Simulation of MBD for the YellowCar must be done in a way which supports both AUTOSAR and non-AUTOSAR based ECUs. Additionally the model developer must be able to decide easily whether to import the standards into an ECU or just leave it as no standardized application.

According to the YellowCar architecture, it contains three basic parts categorized as:

- Sensors
- Actuators
- ECUs

In the following sections, the basic structure required for simulation of three above mentioned main modules of YellowCar are going to be discussed.

#### 1. Sensors

There are variety of sensors that can be used in an automotive system like, infrared sensor, ultrasonic sensor, light sensor and many other types of sensors.
Each group of these sensor’s types depending on their features and characteristics can be responsible for different tasks in a vehicle.

As YellowCar in the introduction part introduced, there are overall 6 ultrasonic sensors, 3 installed on front bumper of the car and 3 on rear side. They are able to recognize the pedestrian or any other kinds of obstacle on the road. Based on their values, AssistantECU can decide if it is necessary to brake and stop the motor. Apart from these ultrasonic sensors, there is a light sensor responsible for lighting of the vehicle. According to the light sensor, the FeatureECU can recognize whether the outside light is enough to drive without lights or it is needed to turn the lights of the car on. However both of the ECUs receives their corresponding commands from ProcessingECU which is directly connected to all sensors.

In order to simulate the functionality of these sensors in the model, Simulink provides several possibilities. Although, one of these possibilities is to connect the Simulink model to the external sensors and feed the system with real data, but it is preferred to utilize the other possibilities to implement virtual sensors inside the Simulink model. Such virtual sensors in Simulink will enable the use to change the sensor configurations as intended to test the model with variety of data with different bit rate, data types, time steps and frequency.

Considering the basic sensor’s signal data types of YellowCar, just 10 bit integer signals from 0 to 1024 are need to be provided to feed the model. Such requirements can be provided either by preparing a preconfigured message data file form MATLAB or directly from a Simulink block. In order to apply such data feeding, in the implementation part both of these possibilities are explained, compared with each other and at the end the reasons of chosen method are realized.

2. **Actuators**

In simple words, an actuator in an automotive system is a component responsible for receiving a control signal from the controllers (ECUs) and applying corresponding reaction. An actuator can be a motor or even lights giving alarm to driver. As it is previously mentioned, in the YellowCar there are four actuators: motor, lights, steering and break. They receive control signals from ECU and their acts can be for example turning the motor/light on and off or changing steering right, left and straight.

Such functionalities in Simulink can be implemented in several ways. Since the main goal is just to monitor the output of ECUs the controlling signal can be simply shown on a scope or display blocks. While simulation is running, both of these blocks are able to show the signal values real-time. The display block can be used by user to check and monitor of data transmission of each
3. Concept

signal and scope is the best choice for comparing the signal behavior, resulting and documentation.

Additionally, the controlling signals coming out of the ECUs can be also illustrated more in a realistic and visual perspective. This can be done by changing the color of actuator blocks. Also Simulink provides possibility to set Image as mask icon on the blocks. By changing each actuator block’s mask icon programmatically it would be possible to illustrate the corresponding ECU commands for the user.

While simulation is running using the combination of scope, display and mask, the user can be clearly informed about the reaction of each actuator and monitor the result of simulation at each step.

3. ECUs

ECUs as the main decision maker or controller part in a vehicle consider as the most important part of an automotive system. The complete set of software in an ECU consists of an operation system, hardware drivers, control algorithms and scheduling software. Basically an ECU model in Simulink must contain at least two main parts:

- Function
- Communication

The function part is like the brain of the ECU, where all the decisions must be made and the communication part includes all the configuration needed for sending and receiving data between ECUs.

Function

In the YellowCar there are both AUTOSAR based and non-AUTOSAR based ECUs available. Depending on type of ECU, the implementation of function part differs. In case of normal ECU (without any software standard), the function part only contains the behavior. But if the expected ECU is an AUTOSAR based one, the behavior must be located in an AUTOSAR SWC. However the software components can be either created in Simulink or directly imported from an Arxml file containing whole AUTOSAR standard descriptions. Thus, the function part of an ECU can contain behavior and alternatively software architecture standard like AUTOSAR. The idea, requirements and possibilities for implementation of both kind of ECUs (standardized and non-standardized) are going to discussed in following and the chosen approach and rest details will be completely discussed in the implementation chapter.
3. Concept

Behavior

Behavior is the permanent member of all ECUs which contains decisions (control functions). In order to define the behavior in MATLAB/Simulink, there are several ways. Simulink typically provides facilities to model the systems control algorithms with a basic scheduler. The combination of several Simulink blocks allow users to model variety of scheduling behavior [18]. Also Stateflow is another possibility for implementation of the behaviors. This module can be implemented, tested and verified separately in another Simulink model and then integrated to an ECU in Simulink model [8].

Simulink also provides simulation of behavioral part of ECUs through MATLAB codes. The codes can be written in separate MATLAB file and called via a Simulink block or they can be located directly into a Simulink block called MATLAB Function block. Although the MATLAB function block accepts only MATLAB language syntaxes, but MATLAB provides possibilities to integrate existing codes in C/C++ or Fortran in our simulation. More information about the different possibilities to use existing C/C++ codes in MATLAB are explained in APPENDIX A comprehensively.

Standardized Software Architecture

As it is discussed in the beginning, the software architecture standard used in all three main YellowCar’s ECUs is AUTOSAR. According to MATLAB/Simulink supports, for modeling and simulation of these AUTOSAR based ECUs in Simulink, there are different possible AUTOSAR workflows between an AAT and Simulink MBD environment.

AUTOSAR Workflows

In order to configure Simulink representation of an AUTOSAR based model for MBD/MBT and generate AUTOSAR-compliant code, MATLAB/Simulink approach follows a transparent process. Mathworks introduced three typical and intuitive different workflows for AUTOSAR including [10]:

- Top-Down
- Bottom-Up
- Round-Trip

As the name of above mentioned workflow suggests, Round-Trip can cover most possible facilities between both sides, MATLAB/ Simulink and AAT. The other two AUTOSAR workflows are somehow a part of Round-Trip approach.
3. Concept

**Bottom-Up**

The so-called bottom-up workflow, or Simulink originated approach, starts from an existing MBD that originated in Simulink model to configure and generate AUTOSAR compliant code. The final production of this workflow is exporting C code and AUTOSAR XML file for integrating into an AAT [8][10].

![AUTOSAR Bottom-Up workflow](image)

Figure 3.2.: AUTOSAR Bottom-Up workflow [8]

**Top-Down**

But top-down approach starts from AUTOSAR component description. It means, in this workflow, after providing the software architecture and final AUTOSAR SWC descriptions from exported Arxml (from AAT), the function designer can develop the behavior of SWCs directly in MATLAB/Simulink [10][8].
3. Concept

Figure 3.3.: AUTOSAR Top-Down workflow [8]

Round-Trip

The Figure 3.4 shows diagram of Round-Trip workflow. This AUTOSAR workflow is the most complete approach in which AUTOSAR SWCs created by an AUTOSAR system architecture tool can be imported into Simulink model, and afterwards both XML descriptions and C code can be exported from the model for integrating into the AUTOSAR system architecture tool [8][10].

Figure 3.4.: AUTOSAR Round-Trip workflow [8]
Communication

The other main part of an ECU in Simulink is the communication module. As previously mentioned, in the YellowCar the communication between ECUs is truly done via CAN bus.

The communication of ECUs has nothing to do with the function part. In other words, in a standardized system like AUTOSAR based application a SWC has no means of knowing where in the system or in which ECU it is located [14].

As depicted in Figure 3.5, the only interface for communication of SWCs internally (inter-ECU) is RTE and they communicate with each other externally (between ECUs) via BSW [14]. As it is introduced at the beginning, the combination of RTE and BSW forms an abstract layer called VFB [14]. The Implementations of YellowCars VFB are already specified by AUTOSAR standard. The only problem is getting this information from the model in AAT and utilize them for communication of modules in Simulink.

Therefore the information of both internal and external communication of SWCs must be extracted from the AUTOSAR description file. This communication information is also called port mapping information and can be achieved either manually from AAT design or programmatically from Arxml.

Moreover in development process of an automotive system with distributed ECU network on CAN bus, the communication descriptions are saved in the form of Database Container (dbc) file. The dbc file contains description about: properties of the CAN network, the connected ECUs to the bus, and the CAN messages and signals. Hence, for modeling and simulation of communication system, using the original dbc file of YellowCar is necessary.
In case of any future updates in the communication system of YellowCar to the other communication standards, the corresponding database files like LIN description file (LDF) or ASAM MCD-2 D (FIBEX) should be imported to the model.

MATLAB/Simulink supports CAN standard by introducing Vehicle Network Toolbox (VNT). The toolbox provides CAN communication possibility either as MATLAB function from command line or Simulink blocks for sending, receiving, encoding and decoding CAN messages. The Figure 3.6 illustrates how this toolbox interacts with its components and is conduit between MATLAB/Simulink and CAN bus [17]. In this illustration one CAN device is attached to the toolbox, built on the MATLAB technical computing environment. After attaching the database file to the configured CAN channel, we will be able to receive and transmit from and to the CAN bus correspondingly.

Figure 3.6.: Interaction of VNT between MATLAB/Simulink and CAN bus [17]
The concept of implementing CAN communication in Simulink needs at least four main Simulink blocks of VNT in each ECU: Receive, Unpack, Pack and Transmit. The Figure 3.7 depicts the internal view of an ECU using above mentioned four blocks. In this concept, an ECU is able to read (Receive) a message from the bus and send (Transmit) a message to the bus. The process is like all ECUs wait and do nothing until their related message IDs appear on the bus. As soon as message arrival, the ECU receives the message with the receive block, decodes (Unpack) the message based on the CAN dbc file and pass the corresponding signals to the function block. After the decision making, depending on the task of ECU, the function block has the possibility to either write any message on the bus for the other ECUs or send command signal to the actuators directly. In case of writing on the bus, it packs the signals in a message (again based on the CAN dbc file). Then Transmit block writes the message with its special ID on the bus.

![Conceptualized internal architecture of ECUs and their connectivity to the sensors and actuators in Simulink model](image)

In the implementation part, we discuss more about the VNT and the facilities provided by this tool for construction of networked ECU via CAN bus. Additionally, the Figure 3.7 shows that the communication architecture of all three available ECUs in the YellowCar are the same. The main differences between them are the functional parts (including all SWCs, ports and etc.) and also how they are connected to the internal architecture of car. According to the specification of the YellowCar, since the Processing ECU is responsible for analyzing the sensor signals it is connected to the sensors (Green/smaller rectangle) and the other two ECUs, Feature and Assistant ECU, are connected to the actuators (blue/bigger rectangle).
3. Concept

3.2. Model Analyzer/Generator

The second concept suggested for this project is implementing a so-called model analyzer/generator in form of GUI next to the generic model of YellowCar. As it is clear from its name, one part is responsible for analyzing the inputs and the other one for generating outputs (models).

The integration of Simulink with MATLAB provides lot of possibilities. The most important one is to access Simulink model programmatically. In other words, instead of working on Simulink blocks manually, MATLAB lets us to write code for whatever a model developer wants to do, for instance: add blocks, add connections, set parameters and etc. This interesting facility motivated us to conceptualize a dynamic GUI tool, able to provide what a model developer in future may need for model extension.

3.2.1. Objective

The main objective of this part is to provide supports for:

- Automatic generic model creation and port connection via a dynamic GUI

It means that not only every model (ECU, sensor) must be added to YellowCar model automatically but also they must be connected to the other available components. Moreover, these all have to be developed in a generic way. In other words, the GUI must be responsible for any updating of files or changes in original YellowCar architecture.

According to the main concept illustrated in Figure 3.1, there are several different files being imported to the model. Basically, the model analyzer must be able to read and analyze information of these files separately. After parsing the files, the corresponding required data must be extracted for generating the model. This data can be any configuration, database, system description, communication and/or system standard file. The model generator based on the extracted content of these files must be able to create each model or component and locate them directly into the YellowCar model.

As it is depicted in Figure 3.1, the model analyzer/generator must be under control of a GUI tool in which the developer can decide and choose which files must be imported to Simulink and which models with which specifications is needed to add to the model. The hierarchical process of importing files, analyzing them and creating model must be implemented in a clear way. In other words the process of the generic model creation must not be confusing for the model developer.

It is obvious that both model analyzer and model generator are integrated together as one unit, but for more understanding functionality of each step, in this Master thesis concept they are illustrated as two different sections with different responsibilities.

The product of this unit can be automatic creation of sensor and ECU (AUTOSAR or non AUTOSAR ECU) with their specification and configuration based on the files.
These automatic model productions will be done based on the generic structure of models. This module will use the extracted data to configure and form the basic model of ECUs and sensors. For example in case of an AUTOSAR based ECU creation, the required AUTOSAR SWCs for this ECU with their corresponding CAN communication blocks with specific configuration based on CAN dbc file will be imported automatically. Moreover, by extracting extra information like port mapping from the system description file, automatic port connection can be also applied. It means, not only the AUTOSAR SWCs can be selected and imported to an ECU, but also the configured CAN modules will be added next to it and all ports will be connected to the model at the same time.

Since development of the whole idea must be done through a GUI tool, next part, discuss the complete process flow of this conceptualized algorithm on a flow chart diagram.

3.2.2. Algorithm Conceptualization

To clarify the aimed process of ECU generation (both AUTOSAR based and non-AUTOSAR based ECU) based on interaction between model developer tool (model analyzer/generator and graphical user interface), a conceptualization diagram is demonstrated in Figure 3.8. In this diagram the green boxes are the actions which needs to be asked from the user to type or select, and orange rectangles illustrate the corresponding process behind of selected user requests. The blue Parallelograms are inputs of system in each stage. Since the outputs of this process are production of ECU and/or SWC in the Simulink model, they are demonstrated as process (orange rectangle) and not output (blue parallelogram).

According to the diagram, first part of ECU creation process by graphical user interfaces starts by clicking on a bottom like ECU creation. Then user is asked to choose a name for the ECU. Afterwards, the corresponding ECU with all default specification of a normal ECU (without any standardized software architecture) will be appeared in the Simulink model of YellowCar. This process will be completely done by model generator. Moreover, name of this ECU will be saved to a text file containing whole available ECU in the Simulink model for further uses. The reason of that will be cleared later in the implementation part.

In the second part the user will be asked if he/she wants to add AUTOSAR SWCs to the ECU. By selecting no, the ECU remains as a normal ECU in the model for further architectural or behavioral development. But by selecting yes, firstly the name of ECU needs to be changed in MATLAB code to run the GUI functions based on the name of new ECU and secondly the user must be dropped to another window for importing the AUTOSAR standard descriptions. By browsing the Arxml file, model analyzer must parse and extract whole information about AUTOSAR SWCs and provide a list for user, in which he/she can select SWCs to import. By choosing each SWC to import, not only the SWC must be imported and saved in separated Simulink file, but also it must be imported directly into the corresponding ECU block being created. At the same time, there are two more
3. Concept

tasks that must be done. One is importing the related CAN modules with correlative configuration and the other is parsing and extracting the port mapping information of the SWCs. In order to fulfill the automatic port connection of SWCs these two tasks have to be performed. As it is shown in diagram, at the end of process by closing file, the MATLAB file containing GUI function has to be reset.

It is obvious that the process of sensor creation will be more like creation of non-AUTOSAR ECUs (first part of chart). Hence, an extra diagram is not prepared for that. The details of both concepts and the chosen implementation process will be discussed in the next chapter.
3. Concept

Figure 3.8.: Conceptualization diagram for process of ECU creation
4. Implementation and Evaluation of the Solution

This chapter provides a complete explanation of each phase of implementation step by step. Based on the conceptualization of the project, the implementation process is divided into two main separated sections.

- Implementation of the YellowCar MBD in Simulink
- Implementation of Model Developer Tool in MATLAB

In the first section 4.1, as the main task of the thesis, simulation of the YellowCar MBD in Simulink and all applied methods are provided. Afterwards, in the second section 4.2, implementation of a model developer tool supporting extension of YellowCar model by providing easy adding modules (like ECU and/or sensor) is realized. In each section all the facilities and supports provided by MATLAB and Simulink and also mechanism, functions and methods which has been used are discussed.

4.1. Implementation of the YellowCar MBD in Simulink

In the concept, the basic functionality and requirements of each module has been discussed. The Figure 4.1 is a screenshot of the simulated model of YellowCar in Simulink containing all available modules like: different sensors, variety of actuators and three main ECUs with different tasks. In this model, the three colorful boxes placed in the middle of the mode are simulated YellowCar’s ECUs connected to each other via CAN bus (long gray box). There are three blocks in front and three in the rear of model as ultrasonic sensors and one block as light sensor. Two black boxes located in front and two in the rear are simulated as car’s lights. Other actuators like Motor, Steering, LightsOnSymbol and BreakLight are demonstrating using four white boxes in the model. Each of these actuators will show their corresponding reaction by getting commands from ECUs. They are also some other actuators like display and scope blocks used in several parts of model.

In order to achieve the architectural model of YellowCar, different possible approaches to provide the requirements and simulate the functionality of each module are needed to be realized.
4. Implementation and Evaluation of the Solution

4.1. Sensors

Considering the functionality of YellowCar’s sensors, 10 bit integer signals with tolerance from 0 to 1024 are needed to feed the model. In order to prepare such data feeding there are two different approaches. One is by preparing a preconfigured message data file from MATLAB and the other is using available Simulink signal generator blocks like Random Integer Generator. However both of these methods are simulated and tested in this work, but final version of model is implemented with Simulink Random Integer Generator block. In the following both methods are discussed to make it clear which facilities have been provided by each method.

Preconfigured CAN Message file

The first possibility is providing a prewritten file containing an array of messages in MATLAB and replaying them.

Since the data provided by simulated sensors, will be sent to the controllers which communicate with each other via CAN bus, it is necessary to provide messages compatible with CAN standard. In other words, in this method the messages generated by sensors are in form of CAN message from the beginning. Therefore the preconfigured message file contain an array of messages and must have properties of basic elements of CAN standards as following:

- Message ID
4. Implementation and Evaluation of the Solution

- Extended
- Name
- Database
- Error
- Remote
- Timestamp
- Data
- Signals
- UserData

Each property can be directly accessed and messages can be sent through CAN bus programmatically. This approach is basically being used to simulate networked system communicating via CAN bus. This is basically how CAN communication is supported in MATLAB. This support is also possible in a graphical way provided by Simulink blocks and will be discussed in the communication part.

The first disadvantage of this method for simulating sensors is that sensors are not visualized in the Simulink model. But the main reason that it is not chosen in the final version of simulation, is generating CAN messages. Whereas in the real model of YellowCar ultrasonic sensors and light sensor are directly connected to the Processing ECU by normal wire without any standard. It means if we wanted to use this approach, we had to either change the real architecture of YellowCar in which Processing ECU has to have extra CAN configuration to receive sensor signals or transmit the corresponding commands directly to the other to ECUs (Feature and Assistant ECU). First way is what is exactly in contrast to the architecture of the YellowCar and in the second one the role of Processing ECU is completely ignored. However sending CAN messages to the ECUs to test their functionality can be a good idea for test system partially, but for running the whole simulation is not a good idea.

It is obvious that creating preconfigured message file with no CAN property is also possible. But sending these signals to the Processing ECU and also no visual manner in the Simulink model still can be considered as design problem.

Random Integer Generator

The second approach which has been chosen for the simulation of sensors in final version of YellowCar model is using Simulink Random Integer Generator block. This block can play the roles of both ultrasonic sensors and light sensor in YellowCar truly. In addition it enables us to generate variety of integer data types, manipulate and
4. Implementation and Evaluation of the Solution

arrange the timing and the range of generating signals to test behavior of system in different test cases.

This approach not only provides a graphical block diagram in the Simulink model, but also it provides a configurable dialog box which let us to generate random integer signals in variety of data types of both sample-based or framed-based output. This block generates uniformly distributed random integer.

According to the YellowCar model (Figure 4.1), three Random Integer Generator blocks as ultrasonic sensors are located in front and three in rear of YellowCar model. Also there is one block as light sensor located in the middle of model. All of the blocks representing sensors are connected to the Processing ECU. In order to realize how this block is utilized to simulate the behavior of different sensors, it is needed to check the different configuration possibilities of dialog box of this block.

The Figure 4.2 shows the Random Integer Generator block and its corresponding dialog box in default state. According to the dialog box, the integer numbers can be distributed in the range $[0, \text{M-1}]$. The parameter M is the M-ary number defined in the dialog box and can be either a scalar or a vector. According to the current specification of YellowCar’s sensors, the range of sensor signals must be set from 0 to 1024 bit. This block also provides possibility to change the period of each sample based vector or each row of a frame-based matrix by sample time panel. However in our implementation process it remains 1. The Initial seed parameter, as its name suggests, is the initial seed value of random numbers. It can be either a constant value or a vector. In case of constant value the resulting generated signal is repeatable. Therefore, it is better to defined different vector value as initial seeds of each sensor to get variety of test cases. The output data type of this block is set to double by default. However it can be changed and specified as a Boolean, int8, uint8, int16, uint16, int32, uint32 or single as well. Based on the data types of sensor signals which are all 16 bit integer, the output data type must be set to int16.

It is obvious that by changing the sensor configuration file, this configuration can be simply manipulated and updated as well. Moreover, the configuration of random integer block not only can be updated manually, but also programmatically. This is true for the all Simulink blocks. MATLAB enables us to access each Simulink block and set different properties with MATLAB code either through MATLAB command line or running a MATLAB file. This facility is one of the powerful aspects of combination of MATLAB and Simulink and has been used in implementation of several different parts of this thesis work.

4.1.2. Actuators

As it is discussed previously there are several different actuators on the YellowCar and there are several ways to visualized them in the Simulink. Since the scope of this Master thesis is just to simulate the architecture of YellowCar to test the system, only using display and scope blocks could be enough to analyze the output and results. But moreover, to visualize the output for user in a more realistic way,
4. Implementation and Evaluation of the Solution

According to the YellowCar mode (Figure 4.1), there are several displays used in different part of model. However, there are also more displays used inside of each ECU. Basically displays have been utilized to show available data on each signal of the model. They were mostly useful in the process of completing and testing the implementation. For instance to check if a CAN message with an especial ID provided by an ECU is received by another ECU or not.

The scopes are basically used for the final result of simulation. The real-time scope blocks enables us to monitor and compare the I/O signals of each ECU. It provides more possibilities like: adding several number of axes in one scope figure with different colors or other styling properties, displaying signals in different time ranges, time labels and sampling. The testing and evaluating the final simulation results documented in this thesis work are done using this block. This is not necessary to mention, that scope block provides several other settings options which are not used in this work.

Although, scope and display blocks are enough to test and evaluate the I/O signals, but we decided to search more about the other possibilities to display the result of Simulink model more realistic. For instance in a real car, if the light sensor recognizes the darkness the lights as actuators turn on. The only possibility for such illustration in Simulink is so-called Dashboard blocks and only available from MATLAB version 2015a. These blocks consisting Knobs, Gauges, Switches, Lamp,

Figure 4.2.: Random Integer Generator block and its dialog box
4. Implementation and Evaluation of the Solution

Dashboard scope help users to control and visualize Simulink model.

As this thesis work is implemented in MATLAB version 2013b, we had to think about improvise another possibility performing similar actuator functionalities like car’s light or motor.

In order to turning a light on or illustrating something similar, it is possible to change the color of a block. For example, we can change them from black to yellow for demonstration of the front lights and/or from black to red for the back lights. However this functionality is not created for this purpose originally, but could be helpful in this case. The only thing is needed is to provide a MATLAB command changing the color property of a block representing a light. This piece of code can be located in a MATLAB function block in the relative ECU block in Simulink.

For getting even more realistic performance for actuators, demonstration of image on the blocks (representing actuators) can be a better idea. This method has been implemented using Masking method. Basically, masking in Simulink is a user-defined interface enabling developer to create a customized appearance. Masking a block has variety of advantages like encapsulating the logic of block diagram and hiding data for subsystems and custom blocks.

Instead of changing the color of an actuator block, we set different images on the block by changing block’s mask icon. This approach not only enables us to display lights but also several other actuators like motor, braking and steering and etc.

As it is shown at the beginning of this chapter in Figure 4.1, there are two black blocks in front and two in back side of the model representing front and back lights correspondingly. Their initial image is set to black as lights-off situation. Also there are four white blocks in the middle of model named as: LightOnSymbol, Steering, Motor, BreakLight. By running the simulation, each of these actuator blocks will get the corresponding command for setting an image. For instance, the motor block will show an ’Engine Start’ symbol by running the simulation and as soon as it receives the stopping command from Assistant ECU, it image will change to ’Engine Stop’. Or for example steering block shows a straight arrow, unless it receives left, right or reverse commands. This is true for the all other actuator blocks with different functionalities.

In Figure 4.3, a table of current visual actuators available on the YellowCar model is provided.

This approach is somehow an improvisation to illustrate realistic actuators in Simulink and obviously it may have disadvantages. The only drawback found for this method is that this method causes reduction of simulation speed. There are two main reasons realized. First reason is loading images in several blocks for a real-time system is really time consuming process, nevertheless the size of images are reduced as less as possible. But the second and the main reason is because of using function ’set-param’ command in the MATLAB function block to set different either color or image on the actuator blocks. Since the final result of simulated model containing MATLAB function block can be production of automatic code generation and ’set-param’ is not supported, it is necessary to exclude this function by adding coder.extrinsic (“set-param”) at the top of the function to bypass code
4. Implementation and Evaluation of the Solution

Figure 4.3.: Table of implemented actuators

generation and it reduces the speed of simulation.

4.1.3. ECUs

According to the simulation model of YellowCar in Figure 4.1, there are three implemented ECUs: Processing, Feature and Assistant ECU. All three blocks representing ECUs in the model are masked with an image containing a part named S-Function communicating with so-called Receive and Transmit interfaces and connected to CAN bus. The S-Function is where the functions and behaviors are located in Simulink model of ECU. Transmit and Receive represent the communication modules in an ECU block. The implementation procedure of both function and communication parts of ECUs in Simulink are discussed in the following.
4. Implementation and Evaluation of the Solution

Function

The function part of an ECU containing behavior is responsible for the decision making tasks. In Simulink, the behavior of an ECU can be simulated by one or several Simulink model, flowcharts or function blocks. The function of an ECU model differs depending on which type of ECU is intended to build and whether it has any standard software architecture or not. Therefore the implementation of function part of ECUs categorized to:

- Non-Standardized ECUs
- AUTOSAR based ECUs

In the Simulink model, the main difference between an AUTOSAR based ECU and a non-Standardized one is having AUTOSAR SWCs in its function section, where the behavior of an ECU will be located.

1. Non-Standardized ECUs

In normal ECUs the function part is defined by normal MATLAB function blocks. We are able to put one or more MATLAB function block in an ECU. This block allows us to add MATLAB functions to Simulink model and is useful for the complicated algorithms that can be better stated in textual form than graphical form in Simulink. This module can also call functions defined in Simulink function block like Stateflow functions.

However, in case of implementation of any function the combination of other Simulink blocks and/or Stateflow could be used as well. The main reason for using MATLAB function block this thesis is that implementation of original behavior of YellowCar ECUs are done programmatically (C code). Therefore simulating these functions in a coding style has seemed more similar to the original model. Nevertheless, the function codes have be written in MATLAB language.

As the YellowCar project is not a new project and a team of programmers have already worked on it, there are huge reference of prewritten ECU functions in C language available. The positive point of having such source code is that we already had the verified and tested functions. But the negative part was that there is no possibility to call C/C++ code directly in MATLAB. Therefore we have got a big dilemma using the prewritten functions of ECUs. In one hand, writing the whole functions is a huge effort and they are already available with the verified stamp. On the other hand importing or wrapping codes into MATLAB readable version was another great challenge as well. Since writing the whole functionality of the ECUs is not the main focus of this thesis, a research on different methods is done to make the available codes usable in MATLAB. At the end, because of complexity and huge effort of wrapping
whole existing C code into MATLAB readable version, the basic required
functions have been written in MATLAB language.

More information about the different possibilities to use existing C/C++ codes
in MATLAB are explained in APPENDIX A comprehensively.

2. AUTOSAR ECUs

In case of AUTOSAR based ECUs the MATLAB function block can be lo-
cated into the SWCs and codes or behaviors will play the role of AUTOSAR
runnable. As all three main available ECUs of YellowCar consist AUTOSAR
application standard, providing the AUTOSAR software architecture contain-
ing SWCs is necessary. There are two possibilities two possibilities for AU-
TOSAR component development in Simulink:

a) Creating a Simulink model representation of AUTOSAR SWC from an
existing Simulink design

b) Importing AUTOSAR SWCs from the Arxml file to Simulink as a model

Although Simulink enables us to create and develop Simulink model represen-
tation of AUTOSAR SWCs from basic, but we attended to utilize the original
AUTOSAR SWCs of YellowCar’s ECUs. Both of these approach are provided
by AUTOSAR support of Embedded Coder software package.

Embedded Coder

Embedded Coder support package of MATLAB/Simulink provides facilities
for AUTOSAR standard. This support package enables user to model and
simulate AUTOSAR SWCs, generate AUTOSAR production code, and ver-
ify AUTOSAR generated code using SIL and PIL simulations. The imported
AUTOSAR compliant code can be used for real-time and non-real-time appli-
cations including simulation acceleration, rapid prototyping, and HIL testing
[19].

Furthermore, for simulation of YellowCar’s architecture, it was necessary to
utilize the original AUTOSAR SWC descriptions. Embedded Coder support
package lets us to import the AUTOSAR Arxml file containing whole AU-
TOSAR SWCs into Simulink model [19]. In the concept chapter, three dif-
ferent possible approaches (AUTOSAR workflows) for importing and export-
ing the AUTOSAR SWC description supported by MATLAB/Simulink have
been discussed as: Top-Down, Bottom-Up and Round-Trip workflows. Each
of these workflows involves another AUTOSAR AAT in which the SWCs can
be exported and/or imported and integrated with the rest of system. In this
work, dSPACE SystemDesk has been used as AAT for exporting AUTOSAR
SWC description, because the basic software architecture of YellowCar has
been designed in this tool.
4. Implementation and Evaluation of the Solution

**Chosen Approach**

In order to fulfill all possible facilities regarding development of an AUTOSAR based system in Simulink, the Round-Trip workflow has been chosen to configure Simulink representation of original AUTOSAR SWCs of YellowCar for MIL test and consequently production of code generation. The only point about the Round-Trip approach is that this workflow is more suggested for the projects with one clear owner of data. Howbeit, in the domain of this thesis, work continued just to exporting the XML file and C code from the model. The integration of these two files back into SystemDesk is not in the project-scope.

![ Simulink Model-Based Design Environment ]

According to Figure 4.4, illustrating the implementation process of Round-Trip workflow, the following three tasks have been accomplished accordingly.

Figure 4.4.: AUTOSAR compliant workflows for enabling Round-Trip engineering [20]

According to Figure 4.4, illustrating the implementation process of Round-Trip workflow, the following three tasks have been accomplished accordingly.
4. Implementation and Evaluation of the Solution

a) Importing Arxml

First step is importing previously specified AUTOSAR SWCs of Yellow-Car into Simulink (including definitions of calibration parameters). This can be done via AUTOSAR Arxml importer tool of Embedded Coder. This tool parse the Arxml file which has been produced by SystemDesk (containing whole AUTOSAR system description) and imports SWC information into a Simulink model for further configuration and MBD. The Arxml importer tool provides possibility to import a subset of the elements and objects from Arxml representing an AUTOSAR SWC. This subset contains all AUTOSAR elements relevant for MBD of an automotive application in Simulink. For instance this subset consists AUTOSAR SWCs and their ports, interfaces, data types, and packages. The Arxml importer tool is able to create Simulink model from the parsed atomic SWCs.

The Figure 4.5 shows, a screenshot image of one of imported SWCs of YellowCar called DisMgr. This SWC will be saved and opened as a Simulink model in a new Simulink file.

b) SWC Development

The imported SWC must be developed using Simulink MBD. In this phase, the behavior of the SWCs have been implemented and located into them as Runnables. As in function section mentioned, we have located the MATLAB function block into the SWCs and put the related function codes in MATLAB language into each block.

As shown in Figure 4.5, the blue box in the imported SWC model is the place to put the MATLAB function block including function codes. As an example, this SWC contains 10 input ports and 10 output ports all connecting to terminators. After locating the behavior into the blue box, the internal ports of blue box will be updated automatically. Afterwards, we just need to delete all terminators and connect the related ports to the box.
At the end of SWC development phase, the validation of AUTOSAR interfaces must be done before extraction phase. Validation is nothing more than compiling system and solving the errors. In this process also following configuration can happen. Configuration of:

- AUTOSAR interface
- AUTOSAR multiple runnables
- AUTOSAR client-server communication

Additionally, configuration and mapping AUTOSAR components can be applied programmatically [20].

Since the context of this Master thesis is simulation of YellowCar in MATLAB/Simulink using AUTOSAR system, the real required AUTOSAR workflow could end at this step. It means, at the end of this stage, we have already imported the AUTOSAR system description of YellowCar and used them in our Simulink MBD. Also verification and validation of simulation was done. But as it is depicted in process diagram of AU-
4. Implementation and Evaluation of the Solution

TOSAR workflow in Figure 4.4, for future work or system extension, in the next step the developed system can be also exported from Simulink.

c) **Exporting C and XML**

After verification and testing the developed SWC models, it can be exported from Simulink for further configuration or merging with other systems. Simulink provides generating both AUTOSAR compliant C code files and XML description. For extracting both of these code files, there are some special configuration for AUTOSAR exports options that must be accomplished before building the model (generating code) as following [20]:

- Inspection and examination of AUTOSAR configured XML options using the AUTOSAR properties explorer or AUTOSAR property functions
- Selecting an AUTOSAR Schema (by ignoring this configuration, AUTOSAR schema version parameter set to the default value, 4.0)
- Configuration of AUTOSAR Compiler Abstraction Macros
- Examination of AUTOSAR code generation parameter on the AUTOSAR code generation options from code generation tab in Simulink

After configuring our AUTOSAR export options, we could generate AUTOSAR compliant code to export our developed AUTOSAR SWC from Simulink.

According to the Figure 4.4, there is one more task shown in the Round-Trip workflow. Since merging SWC descriptions is not in the scope of this Master thesis, it is not listed in the above mentioned tasks of our chosen AUTOSAR workflow. The last stage is basically merging SWC information back into AAT. The SWC are partitioned to separated file as generated C code and description file (XML) to facilitate merging. Later, these files can be merged with other systems using an AAT for further developments.

**Communication**

As discussed in the specification of the YellowCar, the communication standard used in this vehicle is CAN bus containing the CAN cables and the bus master. According to screenshot image of implemented model (in Figure 4.1), the CAN bus is shown with a long gray box connected all three ECUs together. Though, this is just an empty box used for better understanding and visualization of model and the real CAN modules and blocks are located in each ECU separately. The bus master is also shown with a blue block containing the CAN configuration blocks.
The implemented data flow of networked ECU in YellowCar Simulink is depicted in Figure 4.6. It shows how data transmit between ECUs of YellowCar based on AUTOSAR standardized software architecture. It can be seen that Processing ECU receives sensor signals and sends two group of messages on the CAN bus, one for Feature ECU and one for Assistant ECU. Based on the received messages they react and transmit the corresponding commands to their actuators.

![Data transmission between AUTOSAR based ECUs of YellowCar](image)

Before jumping to the detail of implementation of CAN networked ECUs, it is necessary to introduce MATALB/Simulink support package used for CAN communication.

**Virtual Network Toolbox**

As it is previously pointed, MATLAB/Simulink supports CAN communication by introducing VNT. The toolbox provides CAN communication possibility either as MATLAB function from command line or Simulink blocks for sending, receiving, encoding and decoding CAN messages [17]. The VNT also lets the user to experiment the CAN communication through both virtual and real CAN bus. In other word, VNT provides CAN communication either with virtual network bus or real hardware. For example, an ECU network can be connected to the model to com-
4. Implementation and Evaluation of the Solution

municate via real CAN bus cable. This brings the possibility to monitor, evaluate, filter and analyze the CAN bus data while simulation is running.

Besides, VNT provides connectivity to CAN devices using industry-standard CAN database file. The CAN database (.dbc) file containing all physical messages and signal definitions needed for the project can be uploaded to the system. Using database file, message and signal information can be represented in engineering units and the developer does not need manipulate raw data bytes [17]. Here is the list of features of VNT used in the implementation of this thesis work:

- Simulink blocks for connecting a model to a CAN (through both virtual CAN bus and Vector CAN device)
- Configuring a channel on a CAN device
- Vector CAN database support
- Signal packing and unpacking for simplified encoding and decoding of CAN messages
- Vehicle CAN bus monitor application called CANTool, to configure devices and visualize live CAN network traffic

As previously mentioned, in the implementation of communication, except bus master (CAN configuration block) all other CAN modules are located in the ECUs. Though, there are other blocks available in this tool box like Log, Replay which are not used in this thesis work. Also the Universal Measurement and Calibration Protocol (XCP) package available in this toolbox can be used for future work in case of system update.

According to the conceptualization of CAN network, there are four main Simulink blocks of VNT used for implementation of CAN communication between ECUs:

- CAN Receive
- CAN Unpack
- CAN Transmit
- CAN Pack

This process is true for implementation of all three available ECUs of YellowCar model. The only difference is that based on the specification of YellowCar, the Processing ECU receives the main signals directly from the sensors and only writes the corresponding messages on the bus. Also the other two ECUs, Feature and Assistant, receive the messages from the CAN bus and send command signals to the actuators. Nevertheless, in any case the both ECUs are able to write on the CAN bus as well.
4. Implementation and Evaluation of the Solution

In the following part, 'Implemented ECU Communication Architecture', an overview of configuration of each Simulink VNT block used in this work is provided. As previously mentioned all these configuration can be applied by MATLAB command line programmatically. But since we have only used the Simulink block of VNT, it is not considered in this report of implementation.

Before looking through the ECU internal architecture, it is necessary to introduced CAN configuration block as the bus master and the main module in every CAN network communication.

**Bus Master**

In any CAN network, there is an extra module called bus master. This module is basically responsible for mastering the data traffic on the bus. This has been implemented in Simulink by CAN Configuration block.

The configuration of this block is necessary for any CAN communication in MATLAB/Simulink. It configures parameters for CAN channel (virtual or real) that lets us to interact between Simulink and CAN network. Thus, it is better to configure it before configuration of CAN blocks.

For the whole CAN network, one configuration block is enough. In our implemented model (in Figure 4.1), it is located in the blue block named 'Bus Master'. The main configuration done on this block are setting the bus speed to 500000 and selecting the CAN channel. Depending on online/offline mode, CAN channel can be changed to Vector channel or virtual channel accordingly. These modes will be explained later in section 4.1.4.

**Implemented ECU Communication Architecture**

In order to clarify the facilities provided by VNT block for constructing a CAN network communication between ECUs in Simulink, the implemented model of Assistant ECU as an example is provided in this section. In the following each communication block and its configuration are discussed in detail. It is obvious that the communication architecture of other EUCs have not much difference with Assistant ECU and the main differences are about the configuration of the parameters.

1. **CAN Receive**

This block is responsible for receiving messages with special ID from the CAN network and delivering them to the CAN Unpack block. This block has only one architectural difference with the receive block in conceptualized picture (Figure 3.7). The CAN receive block has two outputs, as depicted in Figure 4.7. The main output of this block (CAN Msg) can be one message or all messages at each time step, depending on the block parameters. The extra output port called 'f()' is a trigger to a Function-Call subsystem. As soon as
4. Implementation and Evaluation of the Solution

Figure 4.7.: Internal view of Assistant ECU architecture

CAN receive block receives a new message, it triggers a Function-Call subsystem from this port. Therefore, it was necessary to create a Function-Call subsystem named ‘Function-Call Light ECU 1’, in which we could put the rest of blocks (CAN Unpack, function blocks and etc).

The configuration panel of this block enables us to select Vector virtual or real channel for communication (two channels for each). For testing the simulation in the offline mode where no CAN device is connected to the model, we have used Virtual channel 1. In the online mode, the Vector channel 1 has been used for testing model with TinyCAN and Vector CAN to USB interface. This is the channel which we test the communication between Simulink model and YellowCar. In the 'Testing and Result’ chapter we discuss more about it.

Also this dialog box provides a facility to filter the messages with standard and extended IDs. As it is shown on its block name, we have filtered the messages with standard ID 52 which represent 'AssistanECUStsMsg' message.

There is also possibility to specify the sampling time of the CAN Receive block during the simulation. In our simulation, it is not changed and the default sample time 0.01 (in second) is used.

2. **CAN Unpack**

This block receives the messages from CAN Receive block and is responsible for unpacking the corresponding CAN message into signal data using the specified
4. Implementation and Evaluation of the Solution

According to the Figure 4.8, the only input of this block is 'CAN Msg'. But its output ports depend on the signals available on the unpacked message. The dialog box of CAN Unpack provides variety of possibilities and configuration. It lets user to select data signal as: row data or manually specified signals or even from CAN database specified signals. As we already had the CAN database file of YellowCar, it enabled us to browse and attach the 'YellowCar.dbc' file. By loading the file, the list of all messages available on ECU network will be updated automatically.

In order to realize which message belongs to which ECU, we had to check the database file manually in 'CANdb++ editor'. Basically, this application produced by Vector Company to let the developers to construct CAN communication including messages and signals in form of dbc file.

The Figure 4.9 shows screenshot image of CAN Unpack dialog box containing 'YellowCar.dbc' file. In this example, one of the messages receiving by Assistant ECU is 'AssistantECUSMsMsg' with standard identifier 52. By selecting the intended message, list of signal or signals available in this message will be appeared in 'Signals' tab. In this example, there is only one signal called 'AssistantEcuStsSig'. The other parameters of this signal can be seen as well (read only).

By selecting any other messages available on database file, depending on the number and name of signals, the output ports of CAN Unpack block will be changed.
3. **CAN Pack**

After receiving and unpacking the messages, the function part of ECU will get the message/s and decide based on the content of their signals. If the corresponding decision (command) has to be sent back on the CAN bus then there must be a CAN Pack block available in the model. Otherwise, it will only send the command to its actuator/s.

The CAN Pack block takes the specified input parameter and packs one or more signals into one message. It has one input and one output port by default. But the number of input ports will be updated based on the specified messages and signals (either manually or from database file). The configuration and dialog box of this block is exactly the same with CAN Unpack. Thus, it is not discussed once again.

As depicted in Figure 4.8, CAN Pack receives a signal from the function block named 'LightsCommand'. This signal is 'HomeCmngAndHomeLvngSig' which...
4. Implementation and Evaluation of the Solution

belongs to 'HomeCmngAndHomeLvngMsg' message.

4. CAN Transmit

This block gets the packed message and transmits it to the CAN network using specified CAN device. It has only one input port getting messages from CAN Pack and no output port.

The configuration dialog box of this block is similar to CAN Receive block, even simpler. Since it has transmission functionality, it consists no filtering option. The only configurable parameters are communication channels and message period. During the implementation, we transmit all the messages through the same channel (depending online/offline mode) with the same sample time (1 message per second)

4.1.4. YellowCar Library

In order to easy access of implemented mode, we have implemented a new Simulink library. This user-defined Simulink library contains the models with basic structures. This library consists of sensor, ECU, CAN blocks with their basic configuration as default. The YellowCar library is exactly the same as the ToolBox module in the YellowCar model. This library like the original Simulink libraries enables model developer to click on the blocks, drag and drop them to the model for further manual model developments. Furthermore, all of internal components of module can be used separately. The Figure 4.10 shows the implemented Simulink library for YellowCar. In this screenshot image we can observe all internal components of a component called 'Function-Call Light ECU' inside of ECU block, like: MATLAB function block, CAN modules, actuators and etc.
4. Implementation and Evaluation of the Solution

4.1.5. Online/Offline Mode

These two different modes are basically defined for applying test on the simulated model and verify the results. In order to test the YellowCar MBD in Simulink there are several ways, but all of them will be categorized to these so-called online mode and offline mode. Actually, the offline mode is when we run the MIL test simulation just in the Simulink without any inputs from outside of the model. However this can be accomplished via either Virtual CAN channel or a CAN device (like vector CAN to USB interface). The online mode will be needed when we want to connect the model to the YellowCar or any other CAN devices. In this mode, it is preferred to switch all the internal CAN message sources off and concentrate only on the data flow between hardware and model. In order to implement such functionality there are several possibilities.

The programmatic method was to define a global value with a switching behavior. In each mode all the ECUs have to check this parameter and continue based on its
value. For example for switch value '0' they have to run some parts of code (online part for example) and for switch value '1' the other part.

Another simpler approach was to put manual switch block on the source signals. In the YellowCar, source signals are always coming from the sensors (both light and ultrasonic sensors). As it can be seen in the Figure 4.1, this approach is also implemented as well. There are two manual switches on the way of both main sensor signal sources. There are also two boxes next to them called 'online mode'. These two just contain a constant value (for light sensor) or an array of values (for ultrasonic sensors) preventing the Processing ECU block to send any message to the CAN bus. The connection of manual switches can be changed by simply a double click on the block.

But the simplest way for switching to the online mode is to comment the Processing ECU out of the model. However this can be done more professional by using a Simulink block called Variant Subsystem. This Simulink block enables us to apply multiple implementations for a subsystem where only one implementation must be active during simulation. This means, we could define two different Processing ECUs in one variant subsystem block, one for online mode and one for offline. As a result we will be able to select online/offline modes more visually.

4.2. Implementation of Model Developer Tool in MATLAB

This part describes the implementation of the second proposed concept with a GUI which enables developer to interact with a so-called model analyzer/generator indirectly, to add configured model in the Simulink model directly. Before discussing about the implemented model, it is necessary to look into the conceptualized image (Figure 3.1: Illustration of Concept) in chapter 3 deeply. The depicted diagram in Figure 4.11 shows details of an automatic process for:

- Importing, analyzing and parsing the files
- Extracting components and configurations from the files
- Configuration and creation of the model
- Saving and adding the model to Simulink
- Port connection of new model to YellowCar Simulink model

In the following parts, the implementation method of each step of above mentioned process is going to be explained for both sensor and ECU creation.
4. Implementation and Evaluation of the Solution

4.2. Sensor Creation

In case of sensor creation, the model analyzer set the signal generator based on the sensor configurations. According to this configuration information and the basic sensor skeleton from the Toolbox, the model generator can configure a new sensor. The corresponding created sensor will be added to the YellowCar Simulink model automatically, as shown in Figure 4.11.

Since in this MIL test simulation we have used the Random Integer Generator block for both ultrasonic sensor and light sensor, the block used as the basic skeleton of the sensor module in the toolbox is the same with the other sensors. However in any case the configurations can be changed manually and/or programmatically.

It is obvious that implementation and configuration of sensors as signal provider for our MIL test is not the main focus of this thesis work. Therefore, creation of a generic model of ECU as the most important section of an automotive system is discussed in more details.

4.2.2. ECU Creation

As it is illustrated in Figure 4.11, ECU creation process is much more complicated, because an ECU as the brain of a vehicle needs more configurations. Based upon discussed ECU architecture, in case of an ECU creation, both sides (function and communication) must be covered. Each step of this process has been done using...
different features of MATLAB/Simulink. As it previously conceptualized, the main objectives of this part were not only every model had to be configured and added automatically but also it must be connected to the other available components as well. Besides, the implementation of GUI had to be done in a generic way. It means, in case of changing the system architecture of YellowCar or any update of Arxml file the developer wont have to change any part of codes manually. The model developer will only face to a dynamic GUI tool in which any such mentioned changes or updating are considered previously. Therefore this section of implementation has been divided to three subsections:

- Automatic ECU generating
- Port connection
- Dynamic GUI tool

**Automatic ECU Generating**

Basically, the process of generating ECU differs depending on which kind of ECU is intended to build. As it discussed, the main difference of an AUTOSAR based ECU and a normal ECU in YellowCar Simulink model is having AUTOSAR SWCs in its function section, where the behavior of the ECU is located. In development process of normal (non-AUTOSAR based) ECUs this part is defined by one or more MATLAB function blocks. The only thing we need is to implement a basic ECU architecture model of ECU and configure the communication modules and other parameter programmatically. But for creation of an AUTOSAR based ECU, an automatic mechanism for importing the AUTOSAR description file of YellowCar and extracting each SWC separately is required additionally.

According to Figure 4.11, the model analyzer enables us to import Arxml file by using the AUTOSAR Arxml importer tool of Embedded Coder support. As mentioned, this tool is able to parse AUTOSAR component description file of YellowCar produced by SystemDesck. Afterwards we can extract each needed SWCs and save it as Simulink model. This has been done using createComponentAsModel.

After recognizing all SWCs from the file, the intended SWC can be created as a Simulink model and saved in a separated Simulink file. In parallel, created Simulink model of SWC can be located directly either into an existing ECU (Processing, Feature or Assistant ECU) of YellowCar or into an ECU which is being created.

The imported SWC in Simulink can be developed by refining the AUTOSAR configuration and creating algorithmic model content [19].

Finally the ECU generator (inside of model generator) depending on whether an AUTOSAR or non-AUTOSAR based ECU is intended, which SWCs have to be imported and how many CAN blocks needed correspondingly, can form the basic ECU skeleton coming from Toolbox. Similar to the process of generating sensor, the created ECU will be imported directly into Simulink automatically. As it previously
4. Implementation and Evaluation of the Solution

In section 4.1.3 discussed particularly, afterwards the behavior of each ECU can be added via MATLAB function blocks.

**Port Connection**

The integration of Simulink with MATLAB has provided lot of possibilities for implementation of model developer tool. The most important one is accessing Simulink model programmatically. Connecting ports of Simulink blocks is also possible from MATLAB command line. This interesting facility motivated us to add extra option to the process of ECU creation. In this Master thesis, it is tried to connect any SWC and its related configured communication blocks to each other and also to the available components in ECU at the same time. This should be happen as soon as a SWC is added to the model.

The function 'add-line' lets us to connect any ports of any Simulink blocks together. The only requirement is accessing the port mapping information of original system architecture of YellowCar. This information can be achieved by:

- Checking the system architecture in SystemDesk (manually)
- Parsing the Arxml file (manually/programmatically)

Unfortunately at this phase of implementation which is almost the last phase, we noticed that the available version of SystemDesk is expired and until finishing this Master thesis we could not access to the system architecture of YellowCar from SystemDesk.

Hence, there was only the second possible way for extracting port mapping information either programmatically or manually. Since the Embedded Coder package has no tool for parsing Arxml to extract such information, we had to parse the file with normal MATLAB XML parsing syntaxes. The function ‘xmlread’ returns document object model node. The ‘getElementsByTagName’ method returns a deep list that contains information about the child nodes of corresponding pattern.

In order to know which tags have to be parsed for achieving port mapping information, AUTOSAR XML schema has been checked. This file is an XML language definition for exchanging AUTOSAR models and descriptions and defines the language for exchanging AUTOSAR models.

Using this method we could extract some information about the tag 'CONNECTORS' and save them in a separated file in text format. In this phase new problem occurred. We noticed that the extracted data was not completed. Additionally there were some not related port mapping information saved in the file. Therefore, we concluded that extracting port mapping information of internal and external communication of each ECU, definitely needs a stronger XML parser.

Parsing the whole Arxml file of YellowCar manually which contains huge number of information (more than 5000 lines of code) was also not possible. This way not only needs more human effort, but also is really time consuming. Besides, manual parsing such a huge file could increase rate of human mistakes.
4. Implementation and Evaluation of the Solution

After all, based on the scope of this thesis work, we decided to leave the implementation of automatic port connection at this phase till new version of SystemDesk will be available.

However, because of imported SWC structure (shown in Figure 4.5) it is obvious that providing complete automatic port connection is somehow impossible. The main reason is that the ports of SWC are connected to terminators by default and cannot be connected to the function part, unless the related behavior is placed in it. Therefore the internal port connection of the SWCs has been remained manual.

**Dynamic GUI tool**

As mentioned, the whole process of importing files, analyzing them and generating the models is under control of GUI tool in which the developer can be pushed to the right way without any confusion.

The GUIDE, a graphical user interface development environment of MATLAB enables us to develop graphical user interface. The provided GUI enabled us to name intended ECU or sensor, browse files, updating list of ECUs and sensors, monitoring data traffic and etc.

As one of the most important goals of this Master thesis, it is tried to program the GUI as dynamic as possible. For example, in case of changing number of ECUs or sensors in the YellowCar Simulink model, the lists of ECUs/Sensors will be updated as well. Also in each attempt for an AUTOSAR based ECU creation, the model developer is able to browse and select any Arxml file. Afterwards, the list of SWCs will be updated in two different categories, Application SWCs and Sensor/Actuator SWCs dynamically.

Using GUIDE, there are several GUIs implemented which are able to call each other. The chosen programming strategy for calling functions using handles of variable made it even more dynamic. This feature can be realized for instance while updating dynamic list box of SWCs. In addition, text parsing method was really helpful to make the GUI more dynamic. For example, in each attempt to import SWCs, the name of target ECU will be parsed. This approach let us to develop the list of available ECUs dynamic as well.

The Figure 4.12 illustrates combination of all windows which appear in the process of ECU creating and importing AUTOSAR SWCs to them. It also shows the process of standardizing the existing non-AUTOSAR ECUs in the YellowCar. The first dialog box which the user will face contains following button: Create ECU, Create Sensor, Import SWC and CANTool.

By clicking on the Create ECU button, new dialog box will appear in which model developer will be asked to type a name for the new ECU. Afterward the named ECU will be added to the YellowCar model with a default normal ECU architecture. At this stage, the user will be asked if he/she wants to add SWCs (to standardize the created ECU).

If the user intends to import AUTOSAR SWCs, another window (Listbox) appears in which Arxml file can be uploaded to system. The user just needs to browse
4. Implementation and Evaluation of the Solution

the file. Afterward, the AUTOSAR system descriptions will be imported automatically. As soon as the file imported (takes 3-4 seconds) the list of SWCs of the AUTOSAR system (YellowCar in our case) will be updated. There are two separate list boxes for Application SWCs and Sensor/Actuator SWCs available. Therefore, the model developer can click to import them simply. The selected SWC with the whole communication blocks will be added into the created ECU. After importing all needed SWCs to the ECU, the closing panel in this window asks user to end the process with this button. Based on the programming strategy of this GUI implementation for resetting the system to the start stage, it is necessary to close the window with this green button.

It is obvious that user can ignore the process of standardization of ECU. Even though, there is always the possibility to import and add AUTOSAR SWCs to each of existing ECUs. The implemented GUI lets the user to do it by clicking on button Import SWC. In order to choose the target ECU (in which the AUTOSAR architecture must be located) a list of available ECUs can be updated. By selecting each ECU name the system will automatically choose the corresponding ECU as target and lets the user to import and add components to it. As it is shown in Figure 4.12, again the list box window appears and the rest process of importing Arxml file and adding components will be the same as before.
4. Implementation and Evaluation of the Solution

Figure 4.12.: Process of ECU Creation
5. Testing and Results

As the main goal of all simulation project, this thesis work also aimed to test the result of simulated model and proof of concept involves validation of the model. The simulated model of YellowCar is prepared and ready for MIL test.

In this chapter, we are going to get the result of implemented model and check if it matched the requirements. In order to test the model, we have to test two things. First, checking if the model results are correct and second, verifying the correctness of results. Therefore, the results of this MIL test are going to be tested in two modes:

- Offline mode (MIL test)
- Online mode (verification of implementation)

The offline mode, run the model in Simulink environment and test the results of simulation without any external data feeding. In the online mode, the Simulink model can be connected either to a CAN device or directly to the YellowCar. The MIL testing process in this mode is based on the external data transmission through CAN bus.

5.1. Offline Mode (MIL test)

Based on the implementation tasks of last chapter, this possibility is provided to switch between modes. For testing the simulation results in offline mode, both of on/offline switches in the implemented Simulink model must be switched (connected) to the sensor signals (not to the online mode blocks). Also all the ECUs and modules are available in the model and not commented out. This configurations let the sensor signals pass to the Processing ECU and consequently the other two ECUs will be fed by this ECU through the signals of implemented Simulink sensors. Therefore, in this mode the results of Feature ECU and Assistant ECU has been tested and documented separately.

Assistant ECU

According to the specification of the YellowCar, if any obstacle appears in front or rear of car, the ultrasonic sensors (in front and/or rear) will recognize it. The Processing ECU based on the sensor values sends the corresponding command to Assistant ECU. As a result, it turns the motor off and the car will stop.
5. Testing and Results

As mentioned, the functionality of ultrasonic sensors is simulated in the using Random Integer Generator block. In each time frame, the Processing ECU checks each 3 signals in front and 3 signals in the rear separately and simultaneously. At the time a pair of signal values of each group of 3 sensors shows the same value, it means the sensors recognize the obstacle correctly and Processing ECU realizes it as an obstacle signal. Then it releases the corresponding CAN message on the bus. This message is 'AssistantEcuStsMsg' with CAN ID '52'. Afterwards, Assistant ECU recognizes the obstacle message with its special ID on the bus. It changes its first output value from 0 (default) to 5 and second output value from 100 to 400. The first output is the processed signal which will be used in the result analysis and the second one is the command signal for turning the lights (changing the actuator block’s image).

Notice that all output values of ECUs are contractual and just chosen for experimental purposes to test and verify the correctness of model behavior.

As in implementation of actuators discussed, there are also some visualized versions implemented for every actuator. The visual actuators for stopping the car are 'Brake Light' and 'Engine Stop'. The Figure 5.1 shows a screenshot image of model at the moment that two of simulated sensors have generated same number, 17. Consequently, the brake light is turned on ('Brake Light' is shown) and the motor light is switched to 'Engine Stop' red light (form 'Engine Start' green light). Also, the processed value of Assistant ECU on the display shows number 5, as expected.

![Figure 5.1.: Offline Mode, Assistant ECU recognized an obstacle](image)

In order to document the tolerance of all 6 sensor signals and evaluate their changes and responses of actuators, a scope block is located above of Assistant
5. Testing and Results

ECU. The Figure 5.2 shows a piece of our evaluation process based on the signal tolerances in this scope block in almost 3300 time frames. The two lowermost level scopes show input signals from two sensor groups in front and rear of the car (three input signals from three ultrasonic sensors in each group and illustrated with different colors). The scope on the top displays the output of Assistant ECU in each time slot illustrated with yellow color signal.

Note that for the documentation of result we had to reduce the interval signal changes from 1024 to 20. Otherwise, showing the response of ECU to may take a very larger period and consequently could not be shown in a proper scope. With the same reason, the sampling time of scope has been changed to 200. Also in this image, the above scope shows the relative output signal of Assistant ECU changing between 0 and 5. By looking more precisely, it will be clear that this signal is changed from 0 to 5 as soon as any pair of the sensor signal groups took the same signal value. This is exactly what we expected.

![Figure 5.2: I/O analyzing, ultrasonic sensors and Assistant ECU output](image)

It is clear that by running the system continuously, some delayed in the reaction of ECU can be observed. Although, at the beginning in the first reaction there is no delay and in the last reaction (time frame 3200) a delay with around 50 time frame can be observed.

**Feature ECU**

Basically, the role of Feature ECU in the YellowCar is turning the lights of car on or off. Actually, it receives the corresponding signals from the Processing ECU which
5. Testing and Results

is connected to the light sensor.

The light sensor recognizes the amount of outside lights. If the amount of light is less than a threshold, it realizes it as darkness. As mentioned, this behavior is also simulated in our model using a Random Integer Generator block generating integers from 0 to 1024. The Processing ECU checks the value of this block continuously. As soon as it generates a value less than threshold 300, it sends a message to the bus with value '1'. This message is called 'ActionMsg' with CAN ID '50'. As soon as Feature ECU receives this message, it changes its outputs from 0 to 1 and from 200 to 400. Like Assistant ECU, one of these outputs is for analyzing the scope signals and one of them is the corresponding commands for turning the lights on.

The Figure 5.3 shows the moment that the value of lights sensor has been 196 (less than 300) and the Feature ECU has turned the lights on.

![Figure 5.3.: Result of Feature ECU, all lights are on](image)

For evaluating the changes of light sensor signals and output signals of Feature ECU, another scope block is provided. This scope located above this ECU has two inputs, one for light sensor and one for ECU output. As shown in the Figure 5.4, the under located scope with signals changing between 0 or 1 displays ECU output signals and the other one with signal tolerance of 0-1024 shows light sensor signals.

Note that for documenting proper result evaluation, the sampling time of scope is changed to 200, like the scope of Assistant ECU.

This scope block shows that the output result of Feature ECU remains 0, unless the light sensor signals change to a value under 300. At this time, the output is changed to 1 as expected.
5. Testing and Results

5.2. Online Mode (Verification of Implementation)

In order to verify the result of our MBD, there are several possibilities. The simplest way is providing a possibility for connecting the model to the YellowCar. This connection can be happened via CAN bus cable and CAN interface devices like Vector CAN to USB interface. Through this external connection we could feed the model with external device via CAN bus. Thus, this mode is called online mode.

However, it is also possible to connect the model to a CAN device like TinyCAN instead of YellowCar. TinyCAN device is a CAN interface enabling user to send CAN messages to the CAN bus via its especial Application Programming Interface (API) called CANView.

Regarding setting the system in the online mode, in the implementation chapter several possibilities have been discussed. As mentioned, there are two switches located on the route of both sensor signals. If the switches set to the online mode, the Processing ECU won’t be able to react to any sensor signal. Consequently the model cannot be fed internally and is ready to get external messages. In order to do that we have connected the TinyCAN to the model and set different CAN messages on the CANView.

Although all the mentioned outputs (actuators) are tested in thesis work, but in this part we would like to present only the outputs which are not shown in the offline mode like steering functionalities. Howbeit, the signals related to steering any way won’t be provided by simulated sensors internally. In order to test such
5. Testing and Results

functionalities in offline mode, what we did was feeding and testing the system with TinyCAN while we were still remained in the offline mode. This means without switching to the online mode and while the other CAN messages are on the bus, we have sent other messages with their special CAN IDs and tested the functionalities of other actuators at the same time.

According to the specification of YellowCar, the message information of the Feature ECU and corresponding steering position is as follow:

- 0x01 - car steers left
- 0x02 - car drives forward
- 0x03 - car steers right
- 0x04 - car drives backward
- 0x05 - car stops

The left side values are the message values and the right side explanations are steering positions. The corresponding message ID of all signals is 0x35 (decimal: 52).

The Figure 5.5 shows a combination of several screenshots of simulation result for steering in the online mode. In each of them, a different value has been sent to the bus and the corresponding image is changed on the steering block.
5. Testing and Results

Figure 5.5.: Online Mode, Result of Assistat ECU with steering functions
6. Conclusion

A work is incompleted if the differentiation between the scenarios and ideas of conceptualization phase and present achievements of implementation are unable to be made. Therefore, it is important to consolidate all the achievements and conclude all the works have been done.

This chapter provides following sections:

- Summary
- Advantages
- Future work

At first, all research and development done during this thesis has been summarized within the section of summary. It gives a compact overview of two implemented parts of this work separately. Afterwards, in the next section, the advantages of this thesis work have been included concretely. At last, the section future work, discusses about the reasons of an uncompleted attempt of this work and proposes different possible ways for expanding and improving this project in future.

6.1. Summary

Looking to contents of this Master thesis implementation, the main work can be classified to two topics:

- Simulation of architectural model of YellowCar for MIL test
- Automatic generic model creation and port connection via a dynamic GUI

The first topic as the main task of this work was to model all the main available components of YellowCar in Simulink generically. The main aim of this attempt was regular software and hardware updates for this vehicle. This car as a demonstration vehicle at the professorship of computer engineering of TU Chemnitz is equipped with multiple networked ECUs. Therefore before introduction of any new update, it was essential to test the behavior of the car to reduce the human efforts and costs. Hence, we decided to implement a model in Simulink as close as possible to original architecture of YellowCar. In which we are now able to apply MIL test on the YellowCar with basic ECU architecture and behaviors for both AUTOSAR and non-AUTOSAR based ECUs. This model lets us to configure different sensors as
6. Conclusion

input of system, add/subtract models (sensor and/or ECUs), and manipulate the behavior of ECUs, to evaluate different reaction of actuators in different positions on the lights and displays. Moreover, the scopes output a clear comparison of different I/O signals to evaluate the results and document the observations. Also this MIL test can be done in both online and offline modes. In one hand, the offline mode enables us to apply MIL test on the whole system in the Simulink through either virtual CAN channel or a CAN device. On the other hand, online mode provides real-time connection between YellowCar and Simulink model to test and verify the model. Thus, applying several tests on the online mode, could prove our system design.

The second topic was an extra option to our Simulink model. This idea came to our mind after about 2 months of starting project. We noticed that having an extra application next to the model will be really helpful for future model development process. In case of any update, the model developer can add ECU or sensors by clicking one button simply. Although, because of mentioned reasons the attempt for making this process as automatic as possible stops at the point of automatic port connection, but this module can still create and configure models, import AUTOSAR SWCs and corresponding CAN communication blocks to the ECU model and then add models to the YellowCar. Additionally, the tool itself can be extended easily for further development facilities.

6.2. Advantages

The scope of this Master thesis was analyzing the YellowCar and its architecture to develop a dynamic Simulink architectural model that can be modified and extended to facilitate future updates. The main achievements and advantages collected during the implementation of this thesis work, as realized and mentioned in different parts of this report, can be categorized as:

- Simulation of YellowCar using Simulink
- Implementation of both AUTOSAR and non-AUTOSAR ECUs to be simulated as one system
- Applying MIL test on YellowCar model (Offline mode)
- Visualizing lights as output results through masking
- Changing the architecture of YellowCar by adding and subtracting models
- Easy sensor configurations
- Automatic generic model creation (ECU and sensor)
- CAN communication based on original CAN dbc file of YellowCar
6. Conclusion

- Importing AUTOSAR system descriptions of YellowCar automatically
- Automatic and dynamic importing of original SWCs with corresponding CAN communication blocks in the ECU models
- Automatic port connection possibility (not completed)
- Verification and validation of the system design by CAN connection possibility between model and YellowCar (Online mode)
- Connection possibility to the other CAN devices like Tiny-CAN to feed, test and verify system in different positions in each design phase and any mode (Online/Offline mode)
- Extension supports through dynamic GUI tool using MATLAB
- Visualizing and Monitoring live CAN network traffic via CanTool

It is obvious that no system is perfect and there are also some disadvantages in our implementation. Although it has been tried to design the YellowCar model in Simulink as generic as possible, but there are still some restrictions in case of new architectural updates. For example in case of changing communication protocol in the YellowCar, however MATLAB supports other popular communication standards, but it would be necessary to remove all the available CAN blocks in the model manually and redesign the communication protocols based on new standard.

Additionally, the simulation speed with realistic actuators using mask icons is still not satisfying. In this case, there are some solutions suggested in the future work section.

6.3. Future Work

Since, no work can be ended without having an idea for scope of future including improvement and/or extension, this project is also not an exception to giving a perspective for the possible future works. Thus, some of these ideas are proposed in the following sections.

6.3.1. Automatic Port Connection

Only one of our hopes has not happened completely as we conceptualized as automatic port connection. We tried to connect all ports of each component as soon as it is created and added to the model. Although combination of MATLAB and Simulink provides this facility to connect the ports programmatically, but unfortunately this attempt required classification of huge data in Arxml file with more than 5 thousands of code lines. There are several SWCs and each of them have several ports connected internally (inter ECU) or externally with each other. Since parsing
6. Conclusion

the Arxml file and extracting port connections automatically weren’t possible easily (because of unordered port mapping info), thus the most logical way of extracting port connection can be from original system architecture of YellowCar model in SystemDesk. Unfortunately at this phase of this project the available version of SystemDesk at TU Chemnitz was expired and we could not wait for the updates. However this can be easily completed as soon as new SystemDesk is updated.

The only point is that although this could be a solution for connecting the output ports of a SWC to the other components of model (where it is located), but the connection of internal ports cannot be automated simply. The reason as explained previously is because of imported SWC structure (shown in Figure 4.5). In other words, the ports of SWC are connected to terminators by default and cannot be connected to the function part, unless the related behavior is located in it. Therefore the internal port connection of the SWCs remains still manual and will need more research for future improvements.

6.3.2. Communication with Other Popular Networks

In case of changing whole communication protocol of the YellowCar to the other automotive standards, the model can be redesigned according to the new communication protocol. Fortunately MATLAB/Simulink supports variety of popular communication standards like FlexRay and Ethernet. In Simulink there are block sets representing drivers for these communication and I/O protocols, grouped by protocol. Each protocol block set includes utilities to manage data according to that protocol.

As discussed, VNT provides connectivity to CAN devices from MATLAB and Simulink using industry-standard CAN database files. This toolbox not only provides MATLAB functions and Simulink blocks for CAN communication but also it supports J1939 messages. Moreover there are possibilities for stimulus and measurement with XCP in master mode. The XCP stands for Universal Measurement and Calibration Protocol and is a network protocol for connecting calibration systems to ECUs. To receive measurement values and send stimulation data for bypassing ECU code, XCP blocks in Simulink can be used to implement a real-time application that runs in XCP master mode. It means, in case of any updates in the future, the simulated model of YellowCar will be able to connect to the other models via XCP on CAN using A2L description files.

6.3.3. Increasing Speed of Simulation

Currently to visualize the lights in Simulink, we are changing the images of actuator blocks. Since this approach is somehow an improvisation to illustrate realistic actuators in Simulink, it has disadvantages. As discussed in chapter 4, the only realized drawback for this method is that this method causes reduction of simulation speed. There are two main reasons for that. First reason is loading images in several blocks for a real-time system is really time consuming process, nevertheless the size
6. Conclusion

of images are reduced as less as possible. But the second and the main reason is because of using function ‘set-param’ command in the MATLAB function block to set different either color or image on the actuator blocks. Since the ‘set-param’ is not supported for code generation, it is necessary to exclude this function by adding coder.extrinsic (‘set-param’) at the top of the function to bypass code generation. Consequently, this line of code reduces the speed of simulation.

In order to prevent any timing conflict in our final documented results, we have uncommented these commands related to visual lights. But in the future optimization, visualizing light or any actuator could be applied with other ways without any conflicting with code generation or system speed. However, from MATLAB version 2015a, there are new possibilities for such illustration in Simulink is so-called Dashboard blocks. In future, these blocks consisting Knobs, Gauges, Switches, Lamp, Dashboard scope will be help for controlling and visualizing the outputs of YellowCar model.
Bibliography


[25] Olalla, R.S.: Automating real-time testing for embedded software development


A. APPENDIX A- Integration of existing C/C++ code into Simulink model

In chapter of Implementation and Evaluation of the Solution, section 4.1.3, we anticipated the possibility of integrating C/C++ code in Simulink model for purpose of using them in MATLAB function block as behavior of ECUs. Although, because of mentioned reasons we have not used these methods in the final results of simulation, but this section reviews our findings on the code integration methods offered within MATLAB/Simulink.

A.1. Manual MEX file

The term MEX file stands for MATLAB Executable file and is a function created in MATLAB allowing users to call a C, C++, or even Fortran function. In other words these functions gives developers a possibility to program in C or C++ which are ultra-fast languages and also useable on real ECUs, but be able to call and use them in MATLAB-Simulink environment for the simulation’s purposes.

Although this approach allows full control over application, but a high degree of knowledge of the inner functions of codes and all dependencies is required. Considering time limit of Master Thesis, manual creating MEX files from the whole existing files have been too much efforts and time-consuming. In the next part, an efficient approach called Legacy Code Tool as an automated mean of creating MEX file is going to be discussed.

A.2. Legacy Code Tool

In order to integrate existing C or C++ codes directly into Simulink model-based design environment, the Legacy Code Tool can be used.

The function of Legacy Code, based on the specification which the user supplies using MATLAB structure, generates S-function from the existing C or C++ code. In comparison to the manual MEX file creation, this is an effective tool to transform C codes into S-function. The Legacy Code Tool is not only a quickest method to generate, compile the C codes in MATLAB, but also the tool can create a masked block for the specified S-function which can be used directly from the model.
A. APPENDIX A- Integration of existing C/C++ code into Simulink model

Legacy code tool provides an automated mean of generating S-functions. Howbeit, this tool has some disadvantages. In this method there is no automated way to add code around the external routine call from the MATLAB code interface. In addition, Legacy code tool allows very little control over the appearance of automatically generated S-function [14].