Réf : …../…………   AU : 2024-2025

University of Sousse

National Engineering School of Sousse

Report of Summer Intership

[ ] Initiation [✓]Engineering

Conducted by:

MHAMED Mohamed

Field

Industrial Electronics

Internship completed within the Company

Laboratory of Advanced Technology and Intelligent Systems

Ecole Nationale d'Ingénieurs de Sousse

BP 264 Sousse Erriadh 4023

Internship Period : 01/07/2024 - 31/08/2024

Acknowledgments

I would like to express my deepest gratitude to **Professor Adel Bouallegue** for his invaluable guidance, support, and encouragement throughout the duration of this project. His expertise and insights have been instrumental in the successful completion of my work. I am deeply thankful for the knowledge and experience I have gained under his mentorship.

I would also like to extend my sincere thanks to the entire team at **Latis Laboratory** for providing me with this incredible opportunity. The resources, collaborative environment, and innovative spirit at Latis have greatly contributed to my personal and professional growth during this project. The experience has been immensely rewarding, and I am grateful for the chance to be part of such a dynamic and supportive community.

Summary

Figures:

Tables:

Internship summary sheet

Company Name	**Latis**
City	**Sousse**
Name of the industrial supervisor	**Adel Bouallegue**
Start date and end date of the internship	**01/07/2024-31/08/2024**
Number of effective days spent in the company's premises	**3.0**
The main tasks performed during the internship (maximum 3)	**- Studied AUTOSAR concepts from Classic to Adaptive platforms.** **- Developed and integrated various automotive lighting applications.** **-Implemented a 3D simulation of the integrated application.**
Does the internship include an observation period? If yes, how many days did it last?	**No**
Does the internship include a design phase? If yes, how many days did it take?	**Yes 15 days**
Does the internship include a development phase? If yes, how many days did it take?	**Yes 20 days**
What are the prerequisites for this internship in terms of technical skills?	**- Understanding of AUTOSAR architecture and concepts** **- Familiarity with embedded systems and ECU programming** **- Proficiency in MATLAB Simulink and AUTOSAR Blockset**
Tools and software used during the internship	**-Altova XMLSpy** **- MATLAB Simulink** **- AUTOSAR Blockset** **- Fujitsu MB96340 ECU software tools**
Equipment and machinery (other than PCs) used during the internship	**Fujitsu MB96340 ECU**
Methodologies used during the internship	**-AUTOSAR development process** **-ECU exploration and programming** **-ISO26262 & V Model**
The two main behavioral qualifications/skills acquired through this internship	**- Problem-solving and critical thinking**
The two main technical qualifications/skills acquired through this internship	**- Proficiency in AUTOSAR architecture and application development** **- Experience with ECU programming and simulation tools**

Abstract

Over the last few years automotive electronics and software has grown in importance and so vehicle systems have become more complex. Today the challenge for automotive engineers is not to optimize individual components but to optimize the entire systems. As vehicles must meet many different customer requirements the need for flexibility in product updates becomes critical. To reduce development time and costs you must eliminate repetitive tasks by creating standardized, reusable software components across different vehicle types. This need for standardization led to the creation of the AUTOSAR (Automotive Open System Architecture) standard which was first released in 2005 and the latest version is AUTOSAR v4.0.
AUTOSAR facilitates the development of complex, scalable, and modular software systems, enabling greater flexibility and efficiency in automotive applications. As vehicles become increasingly sophisticated, incorporating advanced driver-assistance systems (ADAS), autonomous driving technologies, and connectivity features, AUTOSAR's role becomes ever more critical. Looking ahead, the continued evolution of AUTOSAR promises to unlock unprecedented opportunities for innovation and enhancement in automotive systems. The standard's adaptability to emerging technologies and its ability to streamline development processes will drive the next wave of automotive advancements, setting the stage for a future where vehicles are not only more intelligent and interconnected but also more aligned with the evolving needs of consumers and regulatory requirements. The future of AUTOSAR and the automotive industry is poised to be extraordinary, with the potential to redefine mobility and pave the way for groundbreaking advancements in vehicle technology.
Key words : AUTOSAR / ECUs / CAN / MATLAB

Chapter 1: Host Organization and Addressed Problem

1.1. Introduction

This chapter provides an overview of the host organization where the internship took place and outlines the problem that was addressed during the internship. The chapter also covers the company's general structure, the specific tasks carried out, and the methodologies employed.

1.2. Presentation of the Company

1.2.1. General Information

LATIS (Laboratory of Advanced Technologies and Innovative Systems) was founded in January 2017 as a research facility at the École Nationale d'Ingénieurs de Sousse (ENISo) which's part of the University of Sousse in Tunisia. As an institution within the university framework without its own legal entity or revenue generating model like a typical business setup. LATIS receives funding from research grants university funds and partnerships, with industry players. Operated under the auspices of the University of Sousse LATIS is not owned by shareholders. Maintains strong ties with various academic and industry partners. The lab is run by a team of over 100 members, which includes 4 professors, 6 associate professors, 2 senior technologists, 28 assistant professors, 13 contractual staff members, 1 administrative technician, 16 postdoctoral researchers, 40 PhD students and 5 masters students. LATIS primarily contributes to the research community through publications, patents and collaborative projects rather, than focusing on product exports. With a focus on cutting edge research in Electrical Engineering and Computer Science fields LATIS specializes in areas, like Intelligent Transport Systems (ITS) Renewable Energies and Smart Grids (RESGs) Artificial Intelligence, Computer Vision and Document Analysis.

1.2.2. Products and Services

The LATIS laboratory engages in multidisciplinary research within the broad field of Electrical Engineering. Currently, its research focuses on four main areas:

- SID (Signal, Image, and Document): Research in this area covers the analysis and processing of signals, images, and documents, with applications in various fields, including document preservation and security.

- CEM-SdF (Electromagnetic Compatibility): This research focuses on understanding and managing electromagnetic interference, with particular attention to integrating Electromagnetic Compatibility (EMC) during virtual prototyping stages.
- Smart Grid & Renewable Energies: This area explores advancements in smart grid technologies and the integration of renewable energy sources to enhance energy management and sustainability.
- Diagnostics & Monitoring: Research in this domain is concerned with the development of innovative diagnostic methods and fault-tolerant control systems, particularly in complex embedded systems.

Research Themes and Axes

The various projects undertaken by LATIS are aligned with globally relevant research themes. The laboratory is involved in developing approaches to address significant challenges in the following areas:

- Analysis and Processing of Images from Ancient Document Heritage:
- Security of Individuals and Virtual Worlds:
- Analysis and Interpretation of Biomedical Signals:
- Medical Image Processing:
- Development of New Methods for Characterizing Electromagnetic Disturbances:
- Integration of Electromagnetic Compatibility in Virtual Prototyping:
- Formal Verification of Safety and Reachability in Hybrid Dynamic Systems:
- Fault Diagnosis and Fault-Tolerant Control with Applications in Complex Embedded Systems:

1.2.3. General Organization of the Company

The research teams are supported by an administrative staff and are led by the head of the laboratory, Prof. Jaleleddine Ben Hadj Slama. The structure is hierarchical, with clearly defined roles and responsibilities among researchers, ensuring focused and efficient progress in each research area.
LATIS is structured into two main research teams, each focusing on distinct specialized areas that allows LATIS to effectively manage its research projects, collaborations, and academic contributions, fostering an environment of innovation and excellence.

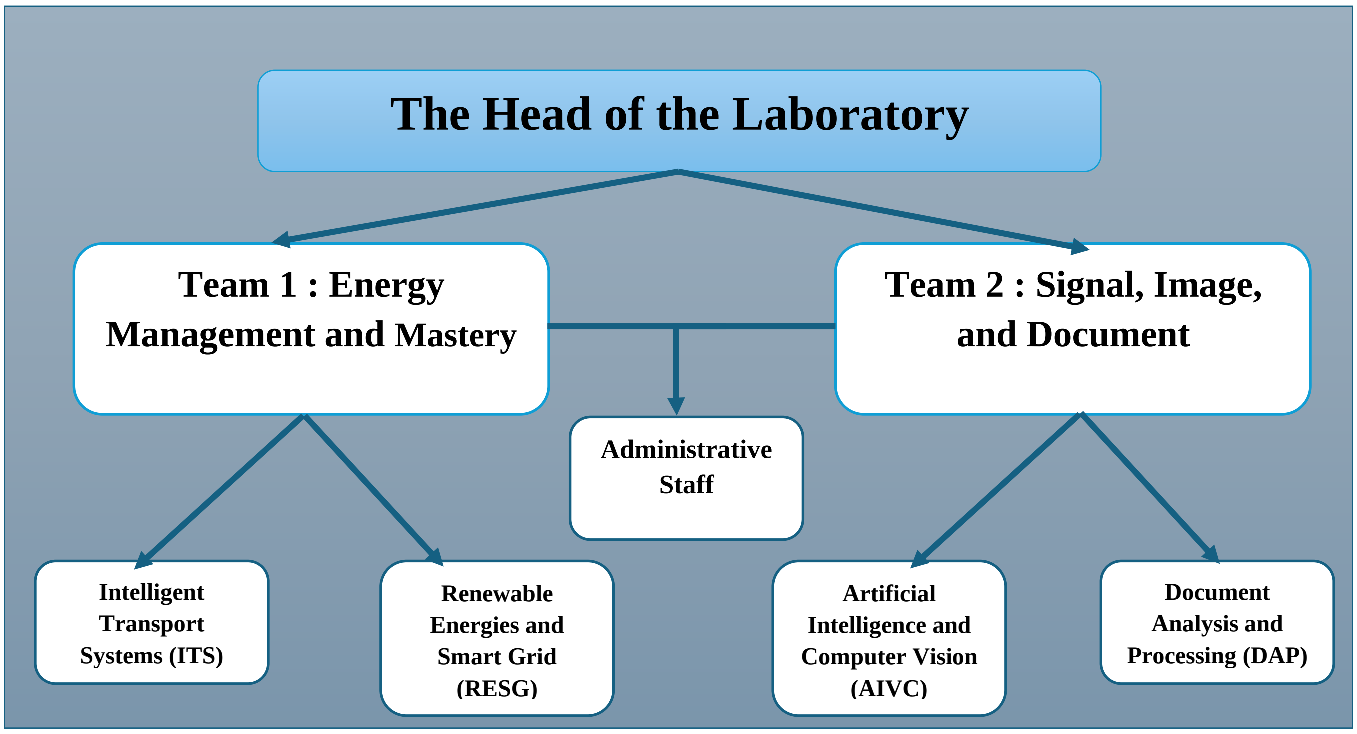

The Head of the Laboratory
Team 1 : Energy Management and Mastery
Team 2 : Signal, Image, and Document
Administrative Staff
Intelligent Transport Systems (ITS)
Renewable Energies and Smart Grid (RESG)
Artificial Intelligence and Computer Vision (AIVC)
Document Analysis and Processing (DAP)

Fig.1: Organizational Structure

1.3. Addressed Problem and Assigned Work

During my internship, I focused on understanding and working with a provided Electronic Control Unit (ECU) in the context of automotive applications. I started by getting to know the ECU's architecture and figuring out how it operates. This was important because I needed a solid grasp of the ECU to analyze its capabilities. After that, I looked into its technical features to see if it could be used effectively in automotive settings, particularly for controlling things like exterior lights.

I also spent time learning about AUTOSAR, which is a standard framework used in the automotive industry to develop software that's modular and scalable. I went through the different stages of the AUTOSAR development process, such as designing the architecture, setting up configurations, and generating code. To put this knowledge into practice, I used tools like MATLAB/Simulink and AUTOSAR Coder to model systems and generate code that aligns with AUTOSAR standards.

Finally, I applied what I'd learned to develop an automotive application for controlling the car's exterior lighting system, including the low beams, high beams, turn signals, and brake lights. I created models and simulations using the ECU, wrote the embedded software based on AUTOSAR standards, and then tested everything to make sure it worked as expected in different conditions.

1.4. Approach and Methodology Used

During my internship at LATIS, at first i was mostly exploring research papesr on the automotive industry standards , tools and the ECU, which helped me on developping an AUTOSAR application for the car exterior lights.

- **The Learning Phase:**

  The first thing I tackled was AUTOSAR (AUTomotive Open System ARchitecture). I really dug into understanding how it works, especially how it organizes automotive software development through its modular approach, architecture, configuration, and code generation. Getting a solid grasp of these concepts was crucial since AUTOSAR compliance is a big deal in automotive systems.

  After that, I delved into MATLAB and Simulink, focusing on how to use the tools for AUTOSAR projects like the autosar blockset . I learned how to model SWC and composition for software architecture, generating the necessary code and arxml files, and run simulations to see how these systems would behave in real-world scenarios.

  I also took the time to study ISO 26262, the standard that deals with the functional safety of automotive electrical and electronic systems. With that, I focused on the V-Model its well known for automotive application development, this part emphasizes rigorous testing and validation at each stage of the project. By following this model helped making sure the software I worked on would be safe, reliable, and up to industry standards.

- Applications Development Phase:

  After gaining a solid understanding of these tools and concepts, I moved on to the actual project work. Using MATLAB Simulink, I modeled the exterior lighting control system for a vehicle. I made sure to integrate AUTOSAR principles into this model to ensure it would meet industry standards. Simulating the system helped me catch and fix any issues early on.

  After that, I generated AUTOSAR-compliant C code from the model and tested it to make sure it met the functional requirements and safety standards, especially those outlined by ISO 26262.

  To wrap up, I used the V-Model framework to guide the validation process, ensuring that each development stage was thoroughly tested before moving on. This iterative approach helped me fine-tune the system and make sure it was reliable before considering implementation.

  This structured approach not only helped me understand and apply the relevant tools and standards but also led to the successful development of a solid and compliant automotive control system.

1.5. Timeline of Tasks Completed During the Internship

The following timeline outlines the various tasks I completed during my internship. These tasks ranged from initial studies and exploration to the development, integration, and testing of automotive applications. The timeline excludes weekends and holidays and is based on the detailed entries recorded in my summer internship journal.

Table.1:Tasks during the internship

Task	**July**	**August**
Study of AUTOSAR Concepts	**M01-M05**	**-**
Working with ARXML Files	**M08-M12**	**-**
Exploration of the Fujitsu MB96340 ECU	**M21-M31**	**-**
Development of the Low Beam Light Application	**-**	**M08-M15**
Development of the Turn Signal Light Application	**-**	**M16-M20**
Development of the Brake Light Application	**-**	**M21-M25**
Integration of Applications and ARXML File Exportation	**-**	**M26-M28**
Simulation of the Developed Application	**-**	**M29-M31**

This timeline captures the sequential progression of the tasks, highlighting the structured approach taken throughout the internship to ensure each phase of the project was completed efficiently and effectively.

1.6. Conclusion

In this chapter, I have provided a comprehensive overview of the host organization, LATIS, and detailed the specific problem addressed during my internship. Through a systematic approach, I familiarized myself with critical industry standards and tools, such as AUTOSAR and MATLAB Simulink, and applied this knowledge to the development of automotive control applications. The timeline presented highlights the structured nature of the tasks completed, from initial exploration to the final validation of the developed systems.

The key takeaway from this chapter is the importance of a methodical approach to both learning and application, which enabled me to effectively contribute to the research and development efforts at LATIS. This process not only enhanced my technical skills but also provided valuable insights into the complexities of automotive software development within an academic research environment.

Chapter 2: BACKGROUND AUTOSAR STANDARD

2.1. Introduction

AUTOSAR changes the relationship between software and hardware in automotive systems making them more independent of each other. This independence allows software to be reused across different ECUs, simplifying the development and reducing costs. This reuse also improves software quality and development efficiency for OEMs and suppliers.

This chapter gives an overview of the AUTOSAR Standard and focuses on three areas: AUTOSAR architecture, AUTOSAR methodology and the theoretical knowledge for specific BSW modules and ISO 26262.

2.2. AUTOSAR Layered Software Architecture

The layered architecture style organizes a system into a set of layers each of which provides a set of services to the layer above. The decoupling mechanism of the layers above from the layers below hides the unnecessary details and minimizes the complexities of each layer thereby imparting abstraction and encapsulation properties.

In the AUTOSAR architecture, adopting the layered architecture style, enables decoupling application development process from the underlying hardware. This property allows a software developer to develop an application for an ECU without being concerned about the hardware architecture, hence making the whole process function-centric rather than ECU-centric. This approach also enhances software re-usability for OEMs because of the standardization of software modules. Figure 2 shows the skeletal framework of the AUTOSAR architecture which mainly includes application layer, Run-Time Environment (RTE) layer, Basic Software (BSW) layer, all built on top of the underlying micro-controller hardware.

The application layer mainly consists of software components and BSW layer consists of system software modules as shown in Figure 2. There are two ways to classify these modules, viz., vertical and horizontal. In vertical classification, BSW modules are classified as system stack, memory stack, communication stack, I/O stack and complex drivers. Horizontal classification (which is color coded) includes Services Layer (SL) (modules in purple), ECU Abstraction Layer (ECUAL) (modules in green) and Micro-controller Abstraction Layer (MCAL) (modules in red). RTE layer conjoins the application layer and the BSW layer and enables communication. Micro-controller is the hardware board on top of which the AUTOSAR architecture layers are implemented.

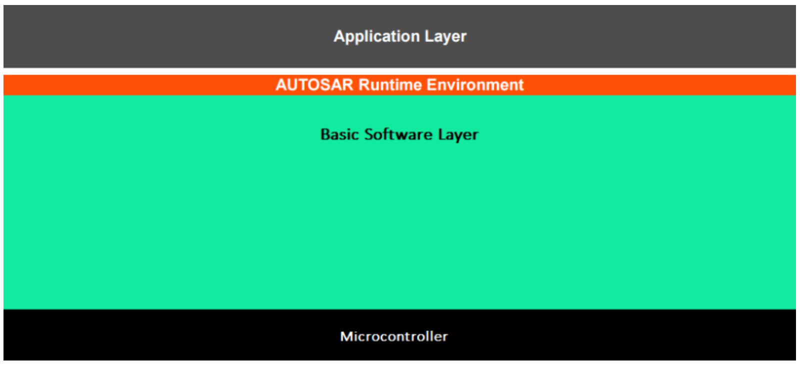


Fig.1: Layered software architecture for AUTOSAR

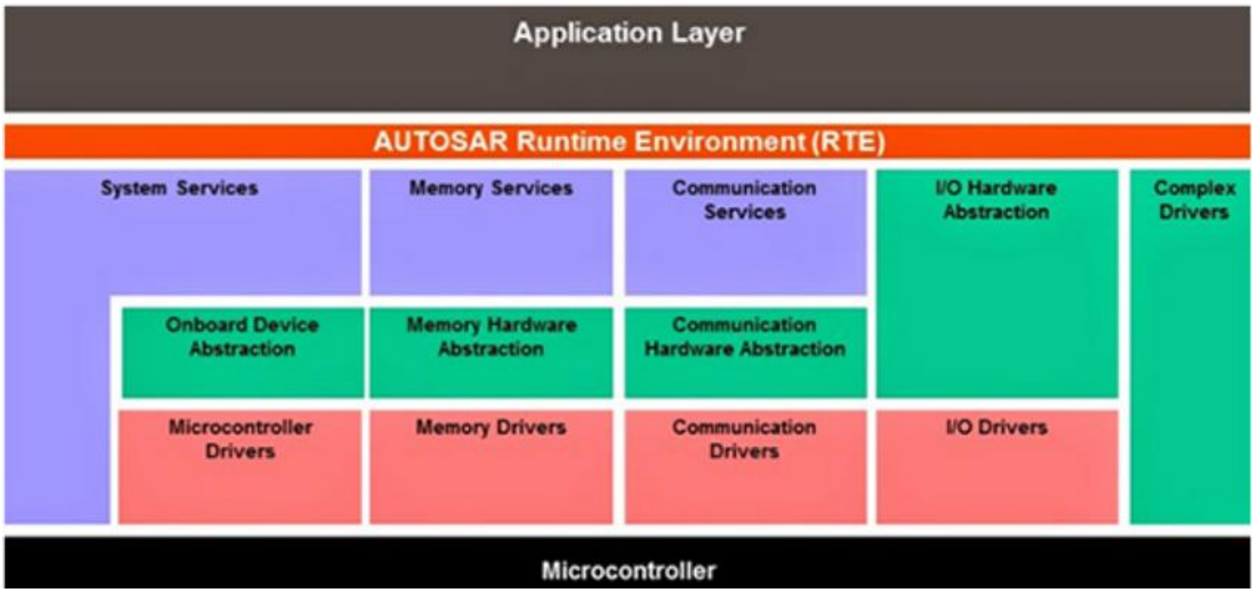


Fig.2: Classification of BSW layers. (Red - MCAL, Green - ECUAL, Purple - SL)

2.2.1. Application Layer

Application software component is an atomic piece of software which is interconnected to other SWCs and BSW modules. Unlike the layered architecture style of an overall AUTOSAR framework, the application layer has a component architecture style. Adopting component style architecture enhances scalability and re-usability of SWCs.

Application SWCs makes use of ports and interfaces for communication. Two main kinds of ports are used i.e., Provide Port (PPort) and Request Port(RPort). Also, two main interface types are normally used, which are, client-server interface and sender receiver interfaces. The port notations are as shown in the Table 2. Port notations depends on the placement of a SWC. A SWC can be placed in any layer (application layer or BSW layer) depending on the functionality it imparts. For example, if a SWC has client-server interface and is placed in BSW layer, then the PPort and RPort are represented accordingly by notations shown in Table 2.

Table.2:Port notations

Component Location	Port Interface	Port	Port Icon
Application Layer	Sender Receiver	RPort	◁
		PPort	▷
Basic Software Layer	Sender Receiver	RPort	◀
		PPort	▶
Application Layer	Client Server	RPort	(
		PPort	◎
Basic Software Layer	Client Server	RPort	◖
		PPort	●

Application SWC can be of multiple types. Few important ones are as listed below.

1. Application SWC - An application SWC is a basic building block of an AUTOSAR application, as shown in Figure 3. It is used to carry out a particular application task within the system. It is a part of application layer and has no direct communication with the underlying BSW layer.

2. Parameter SWC - A parameter SWC is used to store the parameter values like calibration data, variables, fixed data etc required by the application and BSW. The main purpose of this SWC is to only provide data when requested and hence has only PPort as shown in Figure 4.

3. Sensor / Actuator SWC - Sensor / Actuator SWC (Figure 5) is used to interact with the ECU abstraction SWC directly in order to read the sensor values and access the actuators. A sensor component is placed in application layer and has a client port to request sensor data from the underlying ECU abstraction SWC.

4. ECU abstraction SWC - An ECU abstraction SWC, unlike other SWCs, is present in the ECU abstraction layer (i.e within the IO hardware abstraction module in Figure 2). Port notations changes accordingly since this component is placed within BSW layer. This component can access ECUs I/O directly. It is the only SWC that has a direct access to BSW modules, as shown in Figure 6.

5. Composition SWC - A composition SWC (Figure 7) encapsulates one or more SWCs thereby providing abstraction from multiple applications on the same ECU. It encapsulates the SWCs of one application thereby providing a separation from SWCs of another application present on the same ECU.

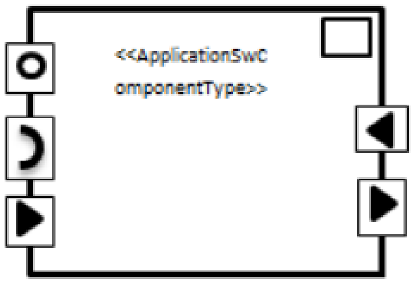


Fig.3: Application SWC type

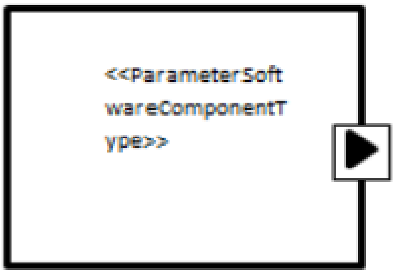


Fig.4: Parameter SWC type

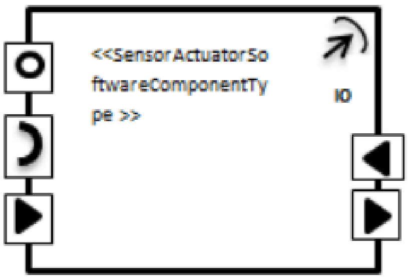


Fig.5: Sensor SWC type

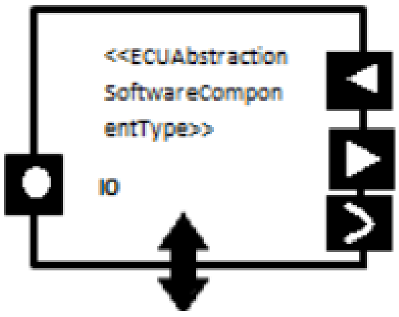


Fig.6: ECU abstraction SWC type

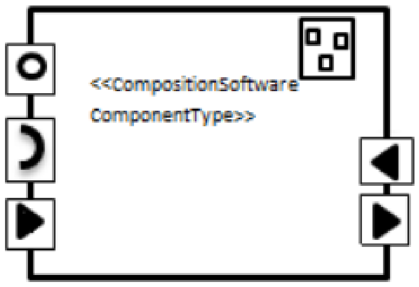


Fig.7: Composition SWC type

Each application component defines an internal behavior for a particular component. An internal behavior specifies how a software component behaves with the rest of the architecture. It includes runnables that specifies the functionality of a SWC.

A runnable is a small software block that implements a certain function. This function is triggered by certain events called RTE Events. An RTE Event activates a runnable entity thereby addressing timing events, sending and receiving data (sender receiver) events, invoking operations, client server events, mode switching and other external events. For example, when the data is received over a port then data received event is generated by RTE that triggers the runnable entity which is responsible to receive the data. The data within a SWC is mapped to ports and the RTE layer establishes communication to other components and/or to BSW layer via ports and interfaces.

2.2.2. VFB layer

Virtual Function Bus (VFB), is the communication concept that separates the communication mechanism between SWCs and the infrastructure beneath the application layer and amongst SWCs themselves. Communication amongst SWCs is done through the same mechanisms as between SWCs and the lower layers in the AUTOSAR stack. Figure 9 demonstrates an arbitrary number of SWCs and the communication interfaces between them. From the SWC point of view, a communication is done via the VFB, thus, a SWC has no awareness of the communication end-point location, (i.e the receiving port) providing mobility among SWCs. In Figure 8 port F writes to two ports, D and B, establishing a 1:2 relation.

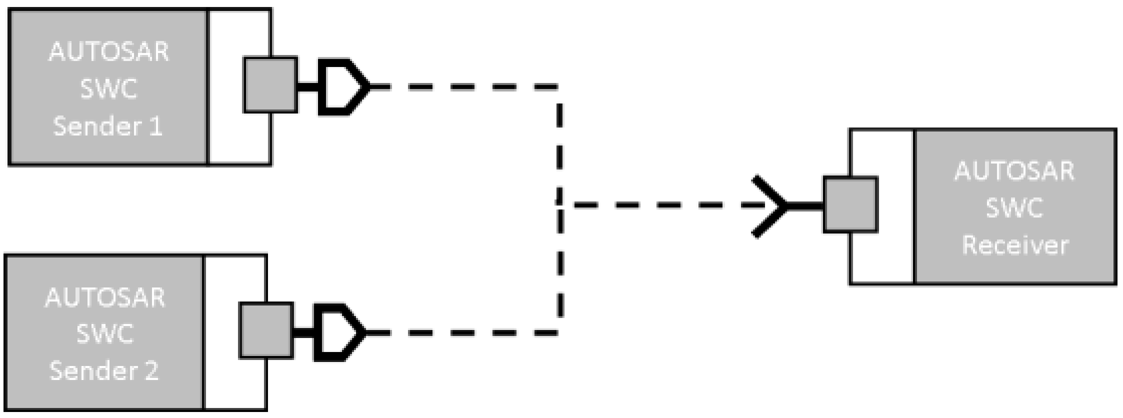


Fig.8: High level illustration of multiple SWCs writing to 1 Receiver demonstrating a 2 to 1

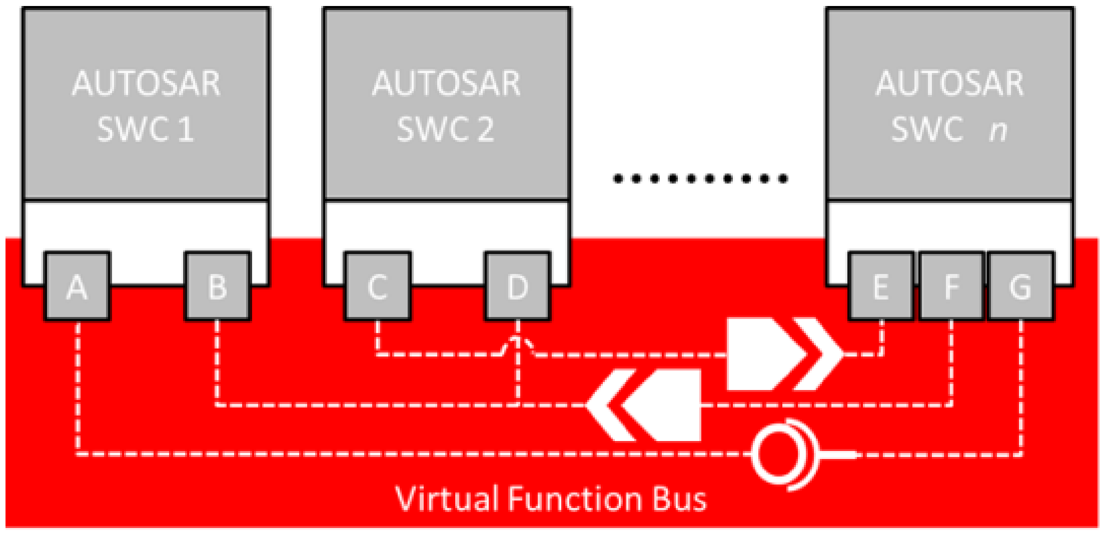


Fig.9: High level illustration of Virtual Function Bus, (VFB), with multi SWCs.

2.2.3. BSW Layer

Basic Software Layer provides core system functionality which consists of modules that imparts specific functionalities for communication, memory, IO etc. Sub-layers within the BSW layer (Figure 2) are explained as follows.

2.2.3.1 Services Layer (SL)

The Service Layer of basic software provides top level services to application software components. These services include operating system functionality, communication services, management services, memory services, ECU state management, mode management, diagnostic services to name a few.

2.2.3.2 ECU Abstraction Layer (ECUAL)

The ECU Abstraction Layer provides abstraction for drivers present in Micro-controller Abstraction Layer (MCAL). It also contains drivers for the external or on-board devices (off chip drivers). ECUAL masks the position of drivers (on chip or off chip) and provides an abstraction to the layers above. It offers an API for accessing the peripherals and devices regardless of their location and connection to the micro-controller and thus makes higher software layers independent of the hardware layout. ECUAL also includes the Complex Device Driver (CDD) layer. CDD is used for deploying functionality that is not available in other modules (e.g. proprietary software). CDD layer connects to the underlying hardware directly as shown in the Figure 10. Hence, applications that have hard deadlines can also be incorporated in this layer. The drivers implemented in this layer do not navigate through the AUTOSAR BSW layers and accesses the micro-controller directly.

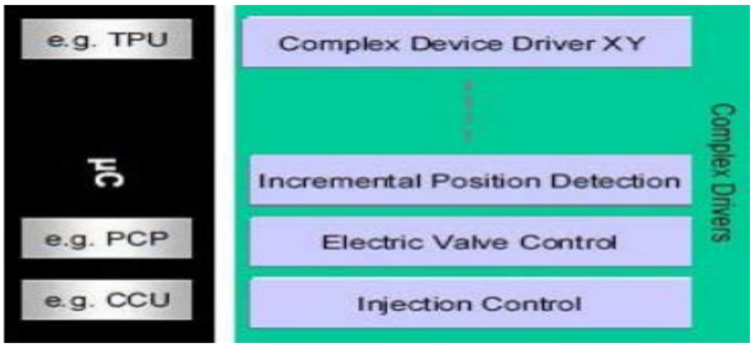


Fig.10: Complex Device Driver Layer

2.2.3.3 Micro-controller Abstraction Layer (MCAL)

Microcontroller Abstraction Layer is the lowest layer of abstraction. It contains internal drivers, which are driver modules that accesses the underlying micro-controller and internal peripherals directly, as shown in Figure 11. Internal devices are located inside the micro-controller like the internal Electrically Erasable Programmable Read-Only Memory (EEPROM), internal CAN driver etc. MCAL provides abstraction to the higher software layers and masks the underlying hardware details.

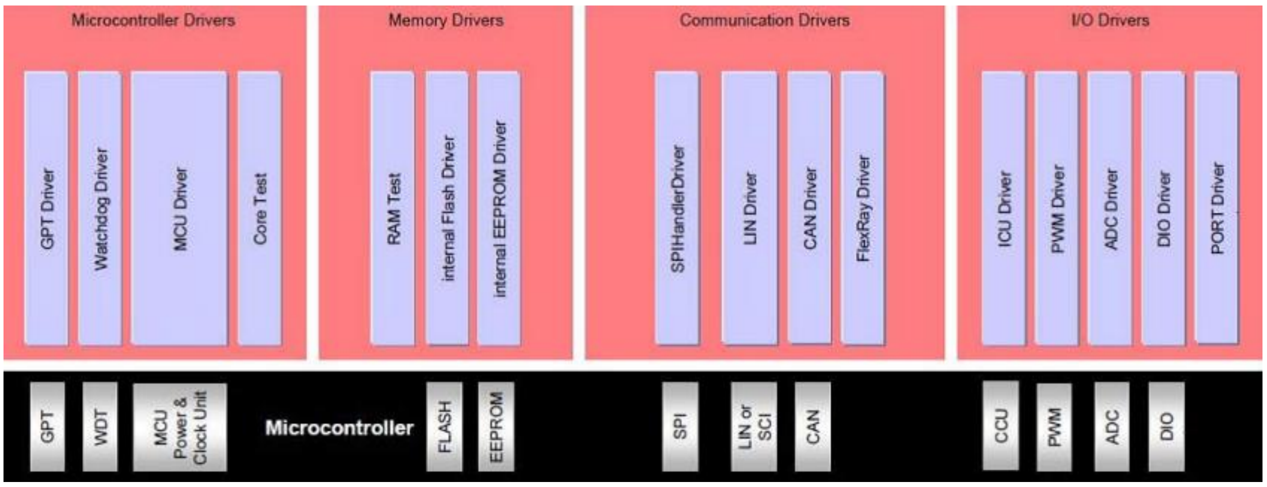


Fig.11: Microcontroller Abstraction Layer

2.2.4. RTE Layer

The Run-Time Environment (RTE) layer facilitates communication between the SWCs. It also creates a join between the application software and the underlying BSW layer via standardized interfaces. The RTE layer decouples the application layer from the hardware architecture. RTE in the realm of software development phase is known as Virtual Function Bus (VFB). VFB is the virtual implementation of RTE which provides similar features as a real RTE layer. VFB aids a developer in early testing of the application software as it provides standard services of the underlying system. RTE also maps the runnables to Operating System (OS) tasks. These runnables are triggered by events called the RTE Events. On occurrence of a certain RTE event (for example, timing event that triggers the runnable every 0.1s), the runnable gets executed and the necessary function is performed.

2.3. AUTOSAR Interfaces

The AUTOSAR interfaces provide standardized APIs between the application layer and BSW layer and between functional units within the BSW layer. These standardized interfaces facilitates software re-usability and interoperability. Three basic types of interfaces constitutes the AUTOSAR architecture as shown in the Figure 12, AUTOSAR Interface, Standardized AUTOSAR Interface, Standardized Interface . Black arrow indicates standardized interfaces which are relevant to VFB and RTE, usually required during the development of application layer and testing (virtual implementation). Yellow arrow indicates the interfaces relevant to RTE and are mainly used while configuring ECU and green arrows indicate the interfaces relevant to BSW modules that provides standardized APIs between modules.

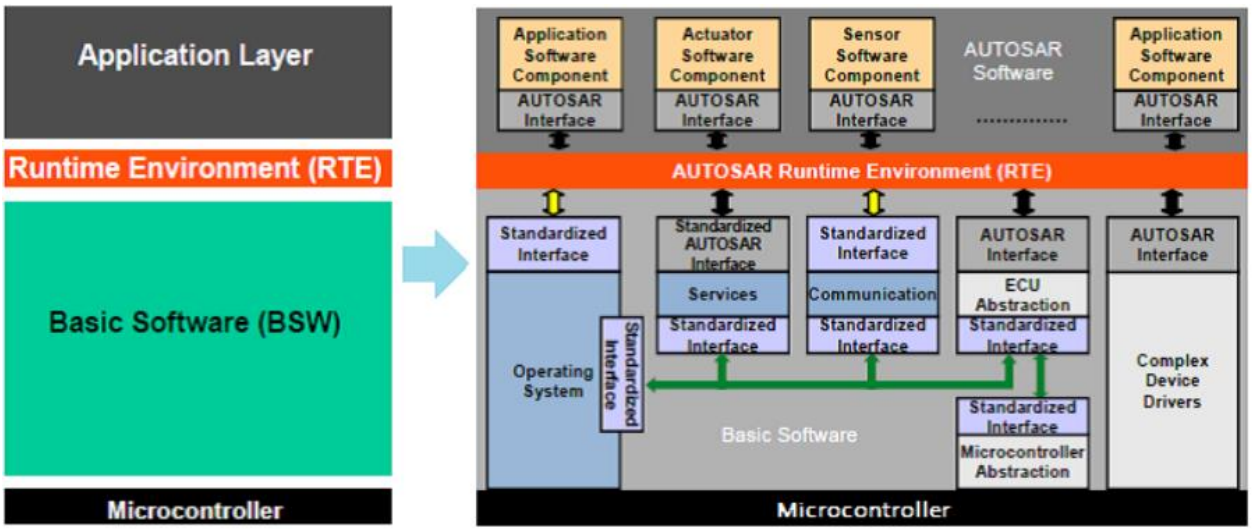


Fig.12: AUTOSAR Interfaces

1. AUTOSAR Interface:

An AUTOSAR interface, as shown in Figure 3, defines the information exchanged between software components and/or BSW layer. Client Server and Sender Receiver are the two interfaces that are commonly used and is independent of any programming language, ECU or network technology. AUTOSAR interfaces communicate via ports in SWCs. AUTOSAR makes it possible to implement this communication between software components and/or BSW modules either locally or via a network.

2. Standardized AUTOSAR Interface:

A Standardized AUTOSAR Interface is an AUTOSAR Interface whose syntax and semantics are standardized in AUTOSAR. The ”Standardized AUTOSAR Interfaces” are typically used to define AUTOSAR Services, which are standardized services provided by the AUTOSAR Basic Software to the application Software-Components.

3. Standardized Interface:

A Standardized Interface are APIs which are standardized within AUTOSAR BSW layer. These Standardized Interfaces are typically defined for a specific programming language (like ”C”). Because of this, standardized interfaces are typically used between software modules which are always on the same ECU.

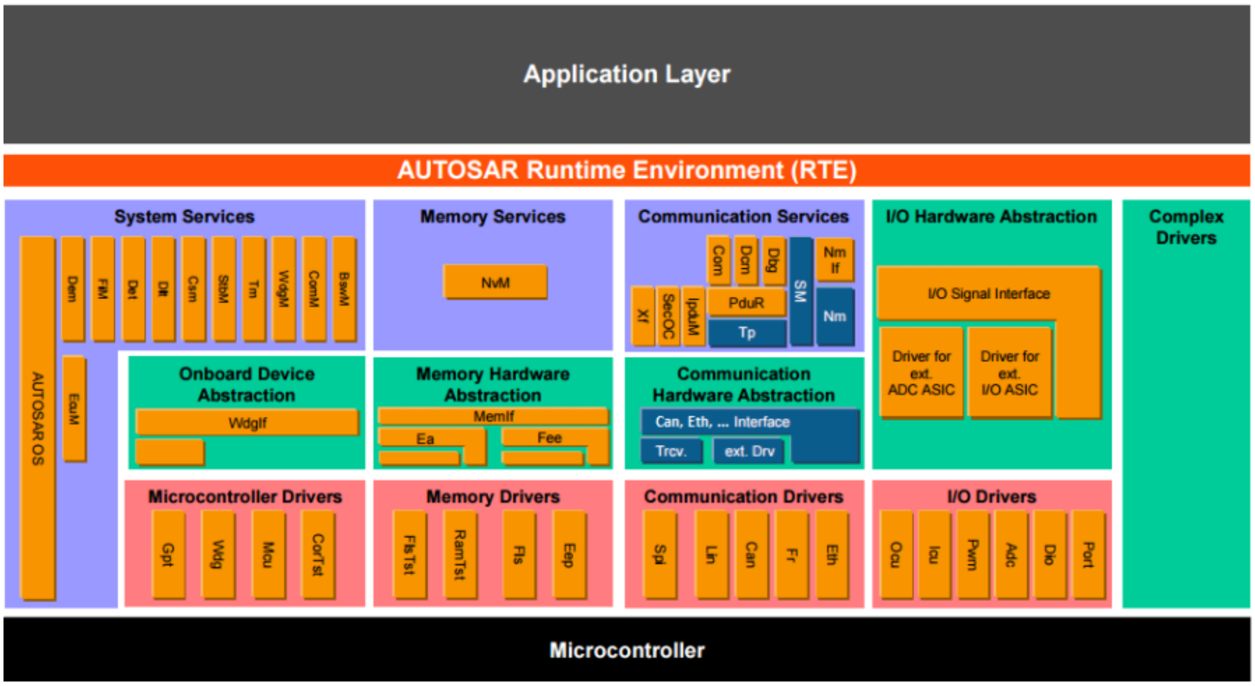


Fig.13: AUTOSAR BSW modules

2.4. AUTOSAR Methodology

AUTOSAR methodology, describes various steps followed in the process of implementing AUTOSAR architecture on an ECU. AUTOSAR adopts a uniform work-flow for the system development. All steps that are required to implement AUTOSAR architecture from system description to the generation of binaries are shown in Figure 14. System configuration input constitutes a system level design both for software, hardware and network architecture and other

system level constraints (for example, number of cores, memory capacity etc.). This serves as the first artefact / input for developing an AUTOSAR system. Once the system is configured, the output is generated (System Configuration Description) and this artifact (.XML file) serves as an input to the next phase in the AUTOSAR methodology.

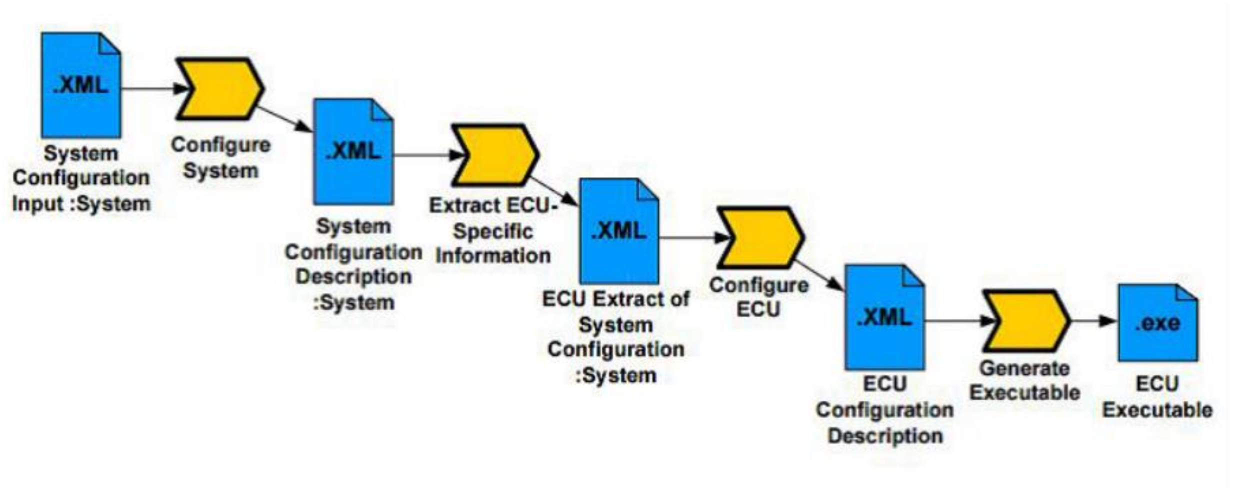

.XML
System Configuration Input :System
Configure System
.XML
System Configuration Description :System
Extract ECU-Specific Information
.XML
ECU Extract of System Configuration :System
Configure ECU
.XML
ECU Configuration Description
Generate Executable
.exe
ECU Executable

Fig.14: AUTOSAR Methodology

System Configuration Description contains the information necessary to configure an entire system. In the next phase, configuration details required for only one ECU is extracted which is termed as ECU Extract. An ECU extract includes a system configuration description for a specific ECU. This implies that a one to one mapping of the system configuration description for a particular ECU is made. It also includes application software configuration description (part of System Configuration Description). The output of this phase is an XML file (ECU Extract of System Configuration) and is the input for next phase of development.

Next, the required BSW modules and RTE layer are configured. In this phase, OS tasks are configured and the runnables are mapped to tasks. The configuration information of all modules along with RTE are stored in XML files (ECU configuration description). At this stage, the RTE generator also generates configuration files (.c and .h files).

Finally, the compiler compiles all the artifacts (.c and .h configuration files and source files) to generate an executable or a binary file. As a result, a .out / .elf file is generated as an executable that can be flashed on the micro-controller.

2.5. ISO 26262

Safety is an essential concept for automotive industry, carmakers aim to security as a key selling point to take advantage of the competition. It is important to have an absence of risk and good measures to manage them. Comes into play here the ISO 26262, an international standard for E/E systems that defines the safety-related requirements covering the entire vehicle life cycle process:

Requirements specification-Design -Implementation-Integration -Verification -Validation

The standard indicates all the steps to follow in every phase in order to ensure the avoid of control systematic failures. The most important aspect of the standard is the concept of Functional Safety that is defined as “the part of the overall safety of a system that depends on the system operating correctly in response to its inputs, including the safe management of likely operator errors, hardware failures and environmental changes. It should be emphasized that functional safety does not implies the total absence of risks of incorrect operation, but it implies the absence of risks that cannot be accepted due to the malfunctioning of E/E systems.

The standard ISO 26262 is divided in 10 parts:

1. Vocabulary
2. Management of functional safety
3. Concept phase
4. Product development at the system level
5. Product development, hardware level
6. Product development, software level
7. Production and operation
8. Supporting processes
9. ASIL-oriented and safety-oriented analysis
10. Guideline on the safety standard

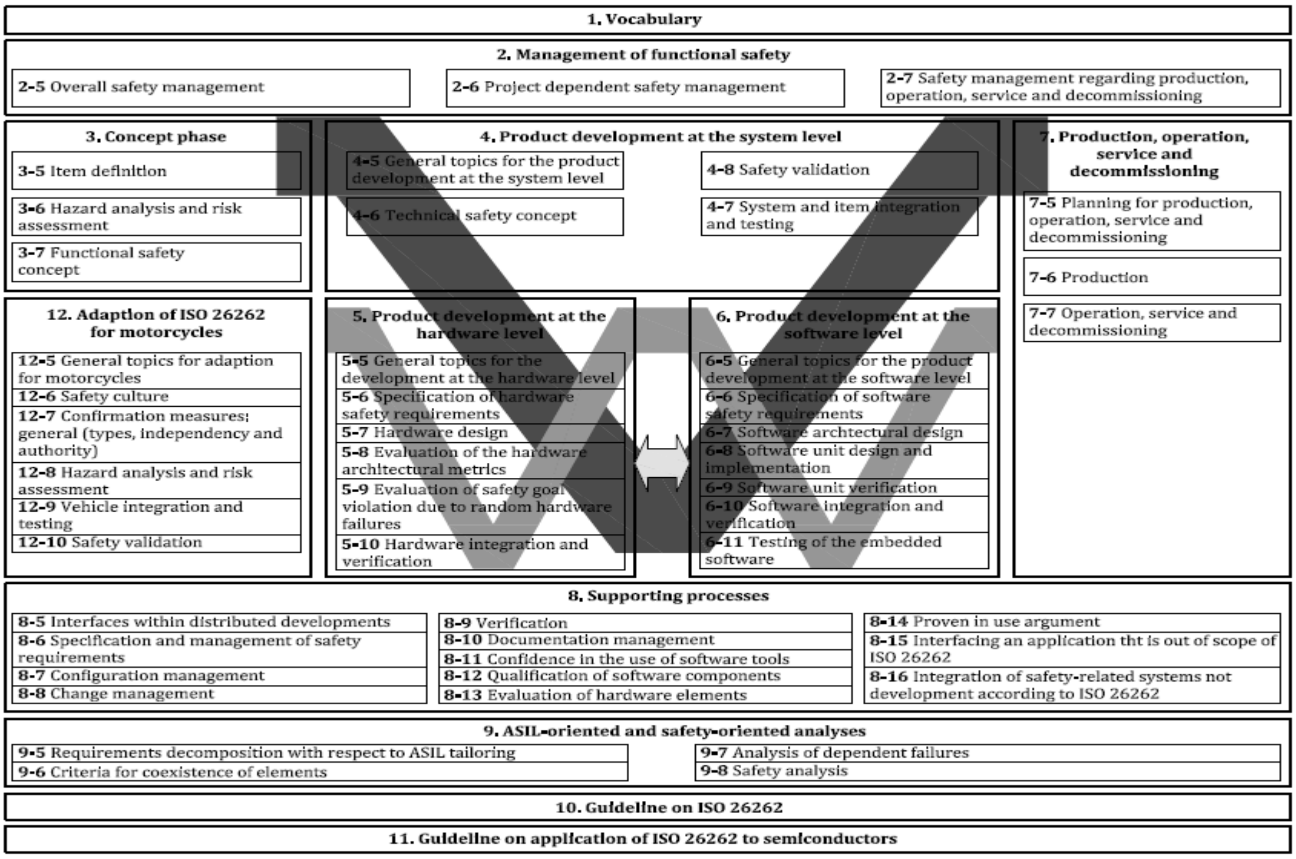

1. Vocabulary
2. Management of functional safety
2-5 Overall safety management
2-6 Project dependent safety management
2-7 Safety management regarding production, operation, service and decommissioning
3. Concept phase
3-5 Item definition
3-6 Hazard analysis and risk assessment
3-7 Functional safety concept
4. Product development at the system level
4-5 General topics for the product development at the system level
4-6 Technical safety concept
4-8 Safety validation
4-7 System and item integration and testing
7. Production, operation, service and decommissioning
7-5 Planning for production, operation, service and decommissioning
7-6 Production
7-7 Operation, service and decommissioning
12. Adaption of ISO 26262 for motorcycles
12-5 General topics for adaption for motorcycles
12-6 Safety culture
12-7 Confirmation measures: general (types, independency and authority)
12-8 Hazard analysis and risk assessment
12-9 Vehicle integration and testing
12-10 Safety validation
5. Product development at the hardware level
5-5 General topics for the development at the hardware level
5-6 Specification of hardware safety requirements
5-7 Hardware design
5-8 Evaluation of the hardware architectural metrics
5-9 Evaluation of safety goal violation due to random hardware failures
5-10 Hardware integration and verification
6. Product development at the software level
6-5 General topics for the product development at the software level
6-6 Specification of software safety requirements
6-7 Software archtectural design
6-8 Software unit design and implementation
6-9 Software unit verification
6-10 Software integration and verification
6-11 Testing of the embedded software
8. Supporting processes
8-5 Interfaces within distributed developments
8-6 Specification and management of safety requirements
8-7 Configuration management
8-8 Change management
8-9 Verification
8-10 Documentation management
8-11 Confidence in the use of software tools
8-12 Qualification of software components
8-13 Evaluation of hardware elements
8-14 Proven in use argument
8-15 Interfacing an application tht is out of scope of ISO 26262
8-16 Integration of safety-related systems not development according to ISO 26262
9. ASIL-oriented and safety-oriented analyses
9-5 Requirements decomposition with respect to ASIL tailoring
9-6 Criteria for coexistence of elements
9-7 Analysis of dependent failures
9-8 Safety analysis
10. Guideline on ISO 26262
11. Guideline on application of ISO 26262 to semiconductors

Fig.15: ISO 26262 Structure

2.6. Conclusion

This chapter has highlighted the critical role of AUTOSAR in managing the increasing complexity of automotive software. By separating software from hardware, AUTOSAR allows for greater flexibility, reusability, and efficiency across vehicle systems. While the standard simplifies development, it also requires careful configuration and strict adherence to safety protocols like ISO 26262. Despite its challenges, AUTOSAR is essential for modern automotive engineering, offering a structured approach to developing reliable, scalable software solutions.

Chapter 3: Testing and Exploration

3.1. Introduction

This chapter covers the development of the Exterior Light Control Application for the car, a key automotive safety feature that ensures brake lights respond accurately to braking inputs. We begin by outlining the application's core functions and proceed to describe the steps involved in creating the Simulink model, configuring AUTOSAR compliance, and generating the corresponding C code. The chapter also discusses the testing methods used to validate the application, ensuring its reliability before implementation in the vehicle.

3.2. Developing AUTOSAR Systems in Simulink

In this section, we will discuss the process of developing AUTOSAR-compliant systems using MathWorks tools, specifically focusing on the integration of Simulink with the AUTOSAR standard. MathWorks, as a premium member of the AUTOSAR consortium, actively contributes to the evolution of the AUTOSAR standard, with an emphasis on applying Model-Based Design in automotive Electronic Control Unit (ECU) development.

3.2.1. Overview of AUTOSAR Support in Simulink

Simulink provides native support for the AUTOSAR standard, allowing engineers to design and simulate both AUTOSAR Classic and Adaptive platforms. By using the Simulink environment along with specialized toolboxes, such as the AUTOSAR Blockset, developers can seamlessly create models that adhere to AUTOSAR specifications.

<u>Key Tools and Workflows :</u>

1. Simulink and AUTOSAR Blockset
2. System Composer
3. Embedded Coder

3.2.2. Round-Trip Workflow for AUTOSAR Development

Simulink, together with AUTOSAR Blockset and Embedded Coder, supports a round-trip workflow for developing AUTOSAR software. The process is as follows:

- **ARXML File Creation**:

The process begins with the creation of an ARXML file, which can be done using either an AUTOSAR Authoring Tool (AAT) or directly within Simulink®. This file defines the structure and behavior of the AUTOSAR software components.

- **Model Generation and Simulation**:

The ARXML file is imported into Simulink to generate a corresponding model. This model can be simulated, and C code can be generated for software-in-the-loop (SIL) or processor-in-the-loop (PIL) testing.

- **Model Updates and ARXML Export**:

Any changes made to the model can be reflected back into the AUTOSAR architecture by exporting an updated ARXML file. This ensures synchronization between the model and the software components defined in the AUTOSAR architecture.

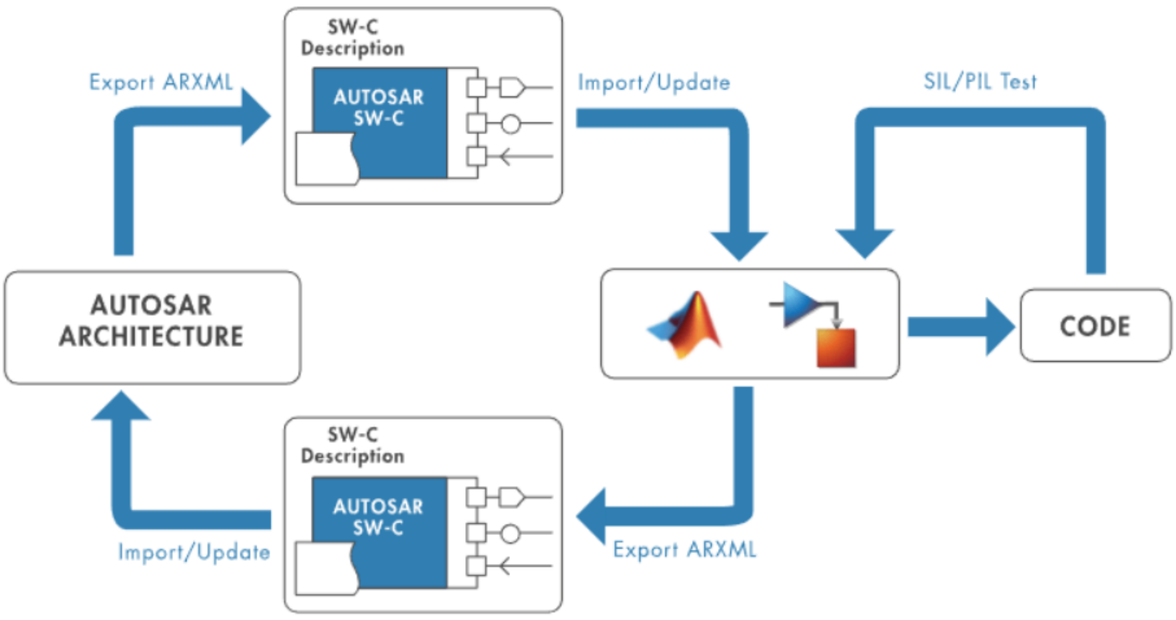


Fig.16: Round-Trip Workflow

3.2.3. Compliance with ISO 26262

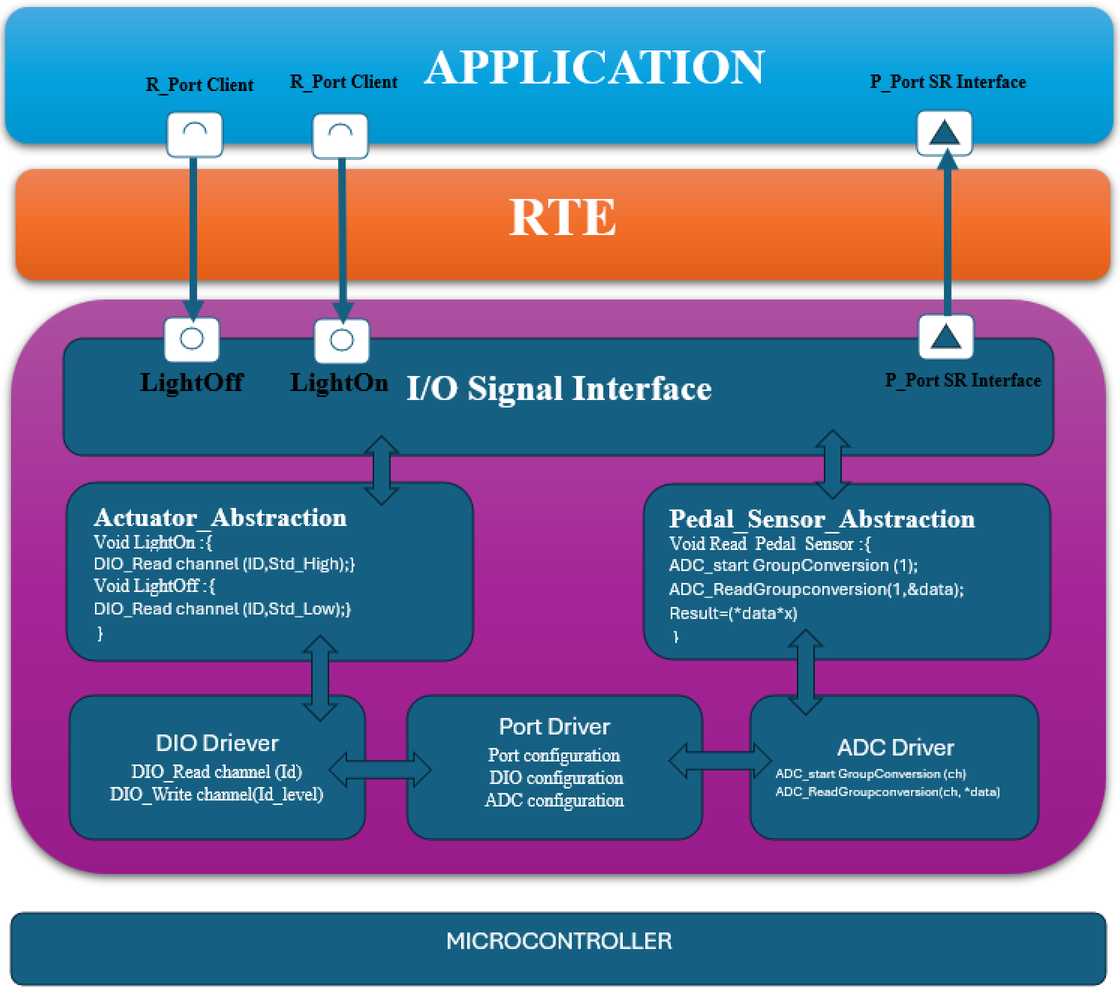
AUTOSAR Blockset is qualified for use with the ISO 26262 standard, which is critical for functional safety in automotive applications. Developers can utilize the IEC Certification Kit to qualify the generated AUTOSAR code according to ISO 26262. This compliance ensures that the software developed using these tools meets the stringent safety requirements needed for automotive ECUs.

By leveraging these tools and workflows, engineers can efficiently develop, simulate, and validate AUTOSAR software, ensuring a robust and compliant integration into automotive systems.

3.3. Development of the Brake Light Control Application

3.3.1. Basic Working Principle

The Brake Light Control Application operates by monitoring key inputs such as brake pedal position and vehicle speed to determine when to activate the brake lights. It ensures that the brake lights are engaged whenever the vehicle's braking system is activated, providing a clear signal to other road users. This system is integrated within the vehicle's AUTOSAR (AUTomotive Open System ARchitecture) framework, which standardizes software development, ensuring reliable and efficient control of brake light operations.



```
Void LightOn :{
DIO_Read channel (ID,Std_High);}
Void LightOff :{
DIO_Read channel (ID,Std_Low);}
}
```

```
Void Read  Pedal  Sensor :{
ADC_start GroupConversion (1);
ADC_ReadGroupconversion(1,&data);
Result=(*data*x)
}
```

```
DIO_Read channel (Id)
DIO_Write channel(Id_level)
```

```
Port configuration
DIO configuration
ADC configuration
```

```
ADC_start GroupConversion (ch)
ADC_ReadGroupconversion(ch, *data)
```

Fig.17: Architecture demo of the braking system

3.3.2. Simulink Model Development

<u>Step 1: Creation of Simulink Data Dictionary File (SLDD):.</u>

To organize and manage the data efficiently, a Simulink Data Dictionary (SLDD) file was created. This file contains all the input and output signals, as well as the calibration parameters used in the model. The steps involved include linking the model to the data dictionary, adding the relevant signals to the design data box, and ensuring that the SLDD file is properly integrated into the Simulink model.

Step 2: Requirement Tagging:

To ensure traceability and compliance with the requirements, each requirement was tagged directly within the Simulink model. This involved linking specific requirement statements from the documentation to the corresponding subsystems within the model. This process enhances the validation and verification stages by providing clear documentation references.

1. *Pedal Sensor Requirement:*
   This subsystem processes the BrakePedalInput to monitor the pedal position. The output signal indicates the level of braking force applied, which is crucial for determining brake light activation.
2. *Throttle Sensor Requirement:*
   This subsystem monitors the ThrottlePosition to assess the vehicle's speed and acceleration. The output signal helps in adjusting the brake light activation based on the vehicle's speed during braking.
3. *Actuator Requirement:*
   This subsystem receives control signals from the controller and activates the brake lights. It ensures the correct implementation of BrakeLightActivation by converting the control signals into physical actions.

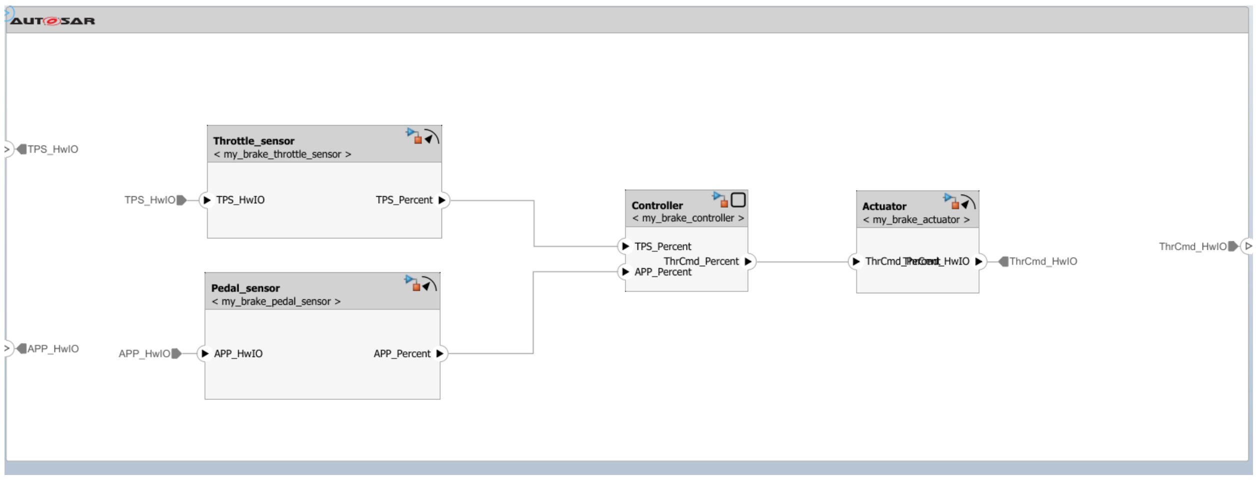


Fig.18: Software Architecture of the braking system

Step 3: Configuration Parameter Update:

To optimize the model for AUTOSAR code generation, the configuration parameters were updated. The solver type was set to Fixed-step with a discrete solver (no continuous states) and a sample time of 0.01 seconds. These settings ensure that the model operates efficiently within the AUTOSAR framework.

3.3.3. AUTOSAR Integration and Code Generation

Step 4: Model Configuration for AUTOSAR Code Generation:.

For seamless integration into the vehicle's ECU, the Simulink model was configured for AUTOSAR code generation. The solver settings were adjusted to Fixed Step Discrete with an automatic step size, and autosar.tlc was selected as the system target file. This configuration ensures that the generated code is compliant with AUTOSAR standards, which facilitates integration and interoperability within the vehicle's software architecture.

Step 5: Mapping Simulink Model Elements to AUTOSAR Software Components:

The next step involved mapping the elements of the Simulink model to AUTOSAR software component elements. This process was carried out using the AUTOSAR Component Designer app, which provides a user-friendly interface for mapping model elements to AUTOSAR components such as runnables, receiver ports, sender ports, and S-R interfaces.

- *AUTOSAR Dictionary:*

  The AUTOSAR dictionary was utilized to define the XML options, interfaces, and ports required for code generation. Specific prefixes were assigned to inports, outports, and data elements to maintain consistency and clarity in the generated code:

  - XML Options: Edit XML options to create a data file for code generation.
  - S-R Interface: Create interfaces for inports and outports with prefixes SIRF_ and DE_.
  - Receiver Ports: Create input ports with prefix Rp_.
  - Sender Ports: Create output ports with prefix Pp_.
  - Runnables: Create runnables for initialization and time events, including init event and TimingEvent.

- Port Mapping:

  Simulink entry-point functions were mapped to AUTOSAR runnables, and data transfer lines were mapped to inter-runnable variables (IRVs). This mapping ensures that the application functions correctly within the AUTOSAR framework, with all necessary data transfers and operations being accurately represented in the generated code:

  1. Map Simulink entry-point functions to AUTOSAR runnables.
  2. Map Simulink inports and outports to AUTOSAR receiver and sender ports, and sender-receiver data elements.
  3. Map Simulink data store and block signals to AUTOSAR variables.
  4. Map Simulink data transfer lines to AUTOSAR inter-runnable variables (IRVs).
  5. Map Simulink function callers to AUTOSAR client ports and client-server operations.

Step 6: AUTOSAR Quick Start:

Finally, an AUTOSAR Quick Start was performed to automatically convert the model into generated code. This process updates all configuration properties and identifies any mistakes, ensuring that the generated code is ready for deployment on the vehicle's ECU.

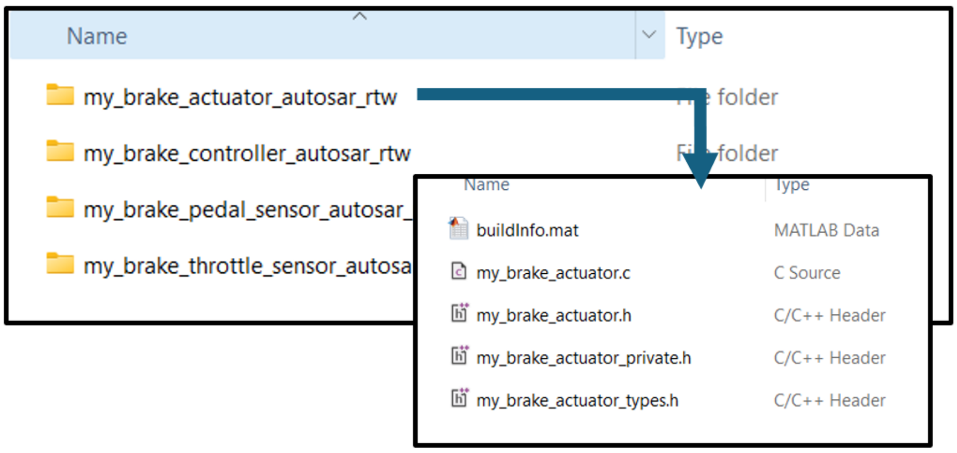
Key C Code Generated:

During the code generation process for each software component—Throttle Sensor, Pedal Sensor, Controller, and Actuator—the following key C code files were produced:

- **ComponentName.c**: This file contains the implementation of the core functions for each software component. For example, it includes functions that handle sensor data processing, control logic, and actuator commands.

- **ComponentName.h**: The header file declares the functions and data structures used within the component, allowing for modular design and easier integration with other components.
- **ComponentName_private.h**: This file defines internal functions and variables that are not exposed to other components, ensuring encapsulation and maintaining the integrity of the component's functionality.
- **ComponentName_types.h**: This header file contains type definitions, such as custom data types or structures, that are essential for the proper functioning of the component's code.

These generated files are crucial for the correct execution and integration of each software component within the overall system, ensuring that the throttle, pedal, controller, and actuator components work seamlessly together in the automotive control application.



Name	Type
my_brake_actuator_autosar_rtw	File folder
my_brake_controller_autosar_rtw	File folder
my_brake_pedal_sensor_autosar_	
my_brake_throttle_sensor_autosa	

Name	Type
buildInfo.mat	MATLAB Data
my_brake_actuator.c	C Source
my_brake_actuator.h	C/C++ Header
my_brake_actuator_private.h	C/C++ Header
my_brake_actuator_types.h	C/C++ Header

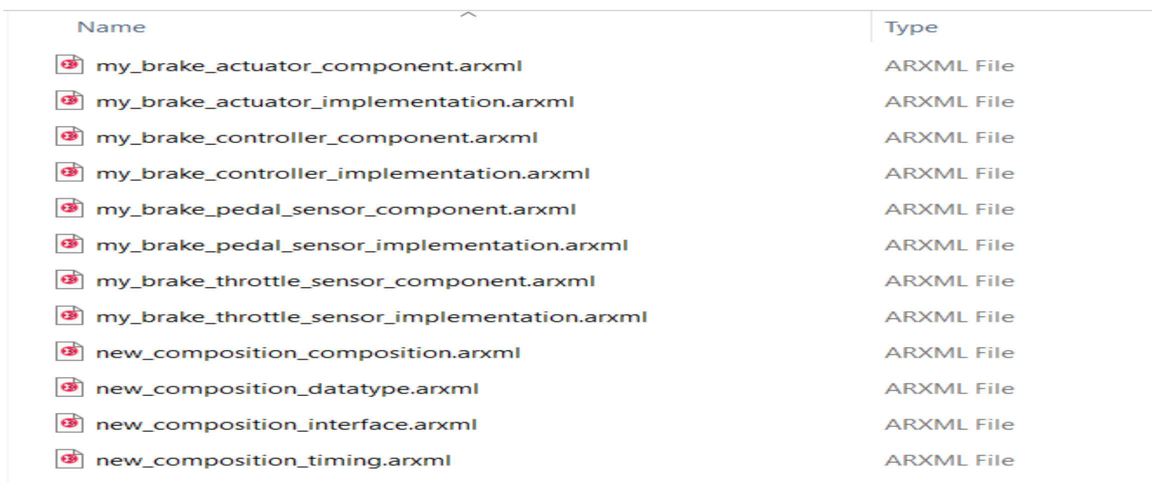
Fig.19: Code Generation for the Software architecture of the braking system

Key ARXML Files:

During the development and code generation process, several ARXML files were generated, each playing a critical role in defining the structure and behavior of the Brake Light Control Application within the AUTOSAR framework:

- **new_composition_composition.arxml**: Defines the overall composition of the system, detailing how individual software components are integrated to form the complete Brake Light Control Application.
- **new_composition_datatype.arxml**: Specifies the data types used across the application, ensuring consistent data handling and communication between components.
- **new_composition_interface.arxml**: Defines the interfaces between components, detailing how they communicate and exchange data within the system.
- **new_composition_timing.arxml**: Contains timing information, specifying the scheduling and timing constraints for the various tasks and functions within the application.

These ARXML files are essential for ensuring that each component of the Brake Light Control Application is correctly defined and integrated within the AUTOSAR architecture, facilitating seamless communication and operation within the vehicle’s overall electronic system.



Name	Type
my_brake_actuator_component.arxml	ARXML File
my_brake_actuator_implementation.arxml	ARXML File
my_brake_controller_component.arxml	ARXML File
my_brake_controller_implementation.arxml	ARXML File
my_brake_pedal_sensor_component.arxml	ARXML File
my_brake_pedal_sensor_implementation.arxml	ARXML File
my_brake_throttle_sensor_component.arxml	ARXML File
my_brake_throttle_sensor_implementation.arxml	ARXML File
new_composition_composition.arxml	ARXML File
new_composition_datatype.arxml	ARXML File
new_composition_interface.arxml	ARXML File
new_composition_timing.arxml	ARXML File

Fig.20: ARXML files of the braking system

3.3.4. Testing and Validation Model-in-the-Loop (MIL)

The initial phase of testing was conducted within the Simulink environment using Model-in-the-Loop (MIL) testing. The control logic for the Brake Light Control Application was validated by simulating vehicle behavior under different braking conditions. The incorporation of the simulation tool allowed for a more comprehensive evaluation by providing a realistic visual representation of the vehicle's response, including how the brake lights would be activated in real-time based on the simulated driving scenarios. This step ensured that the model behaved as expected before moving on to more detailed testing phases.

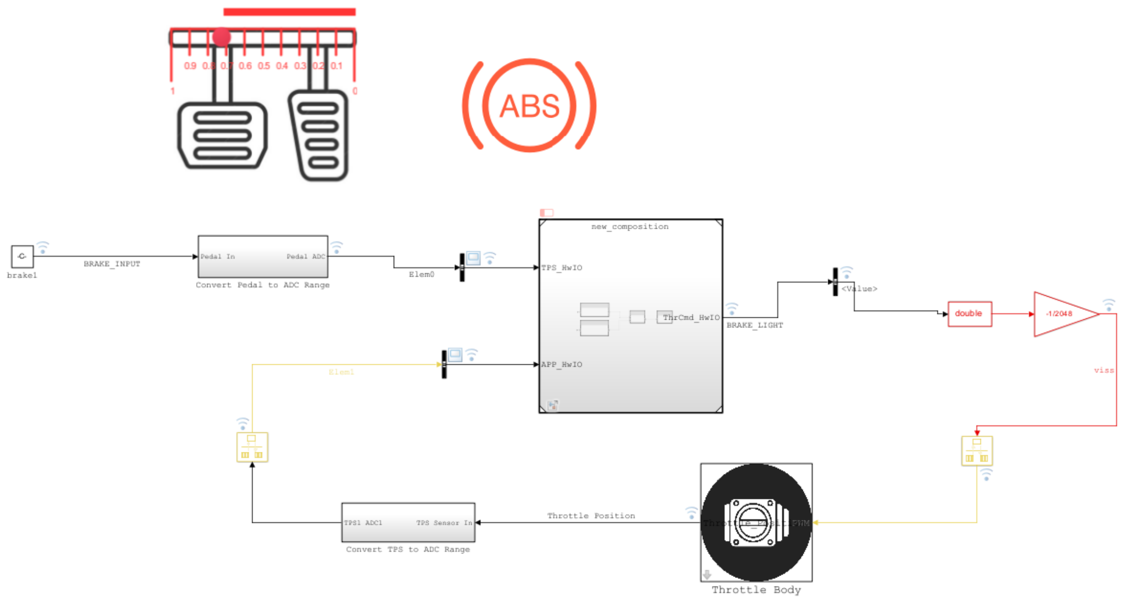


Fig.21: Simulink test of the braking system

3.4. Conclusion

The development of the Application demonstrates the complexity of automotive software engineering, from modeling to code generation and testing. While the application performs effectively in simulations, implementing it on the Fujitsu MB96340 hardware introduces challenges such as processing limitations and integration issues. These factors highlight the difficulties of ensuring optimal performance in a real-world automotive environment.

Conclusion

This report addresses the core problem of Automotive embedded systems, for which a structured and methodical approach was adopted. The methodology involved ISO26262 and V model, which allowed for a systematic exploration of the project goals.

The second part of this report highlights the functionalities that were successfully designed and implemented, including the car exterior lights . However, due to time constraints and other challenges, certain features were only partially completed, such as the hardware implementation, while others could not be realized at all. A significant obstacle encountered was the inability to implement the work on the Fujitsu hardware due to the lack of appropriate software tools. This limitation significantly impacted the overall scope of the project. In retrospect, the solution and the methodology could be further refined by finding a combatible software for the hardware or changing to a Raspberry. Continuous advancements in technology and resources could enable a more robust and comprehensive implementation in the future.

From a personal and professional perspective, this project has been immensely valuable in developing new technical skills, particularly in developing SWC ,Matlab Simulink and the AUTOSAR Blockset. Moreover, the experience has provided insights into the professional sector, deepening my understanding of Automotive Embedded Systems. This Intership has not only reinforced my career ambitions but also introduced me to new avenues and potential career paths I had not previously considered. Although there were challenges, including the Hardware limitations, the overall experience has been positive and constructive.
Looking forward, my expectations for the upcoming PFE internship include continuing to build on these technical and professional competencies, with a particular focus on Automotive systems. I aim to target companies that are at the forefront of Cars Industry, where I can further refine my skills and contribute meaningfully to innovative projects.

My Skills Assessment Overview

What engineering professions do I intend to pursue in the future?

Automotive software development

Embedded systems engineering

What engineering professions would I not want to pursue?

industrial engineering

Fiche bilan de compétences

	Compétences totalement ou partielle acquises			**Targeted Competency (1st Priority)**		**Targeted Competency (2nd Priority)**	
	La compétence	Mastery Level	Practice Opportunity	Targeted Competency	Means to Achieve	Targeted Competency	Means to Achieve
Technical Practical Skills	Programming in C/C++	75%	Developed various software applications and algorithms.	ARXML	Exploring with SWC	AUTOSAR Diagnostics.	Engage in projects focused on diagnostics, take specialized training.
	MATLAB/Simulink Modeling	80%	Developed models for various automotive systems.	Real-time operating systems (RTOS).	Study RTOS concepts, implement in projects.		
	Embedded Systems Programming	65%					
Socioprofessional (Soft Skills)	Time Management	70%	Managed deadlines for multiple projects.	Delegation	Participate in team projects, take on leadership roles.		
	Communication Skills	75%	Presented project reports and technical documentation.	Public speaking.	Attend workshops, practice with peers,	Technical writing.	Write detailed project documentation

Références

https://www.autosar.org/standards/classic-platform

https://www.mathworks.com/videos/develop-autosar-classic-adaptive-applications-with-model-based-design-1671654585175.html

https://github.com/leduynguyen/My_AUTOSAR_Project

https://www.autosartoday.com/posts/types_of_interfaces_and_ports

https://autosar.readthedocs.io/en/latest/autosar4_api/component/composition_swc.html

https://github.com/omerfaruktekin13/ThrottlePositionControlwithSimscapeandAUTOSARStandards

https://webthesis.biblio.polito.it/secure/27762/1/tesi.pdf