Adrià Cusidó i Mas

Development of a standard and reusable real time clock AUTOSAR complex device driver

Master's thesis

Supervised by Mr. Axel Palomino

Master degree on Electric Vehicle Technologies

UNIVERSITAT ROVIRA i VIRGILI
Escola Tècnica
Superior d'Enginyeria

Tarragona
2024

Abstract

In the ever-evolving landscape of automotive software, the AUTOSAR (AUTomotive Open System ARchitecture) standard has become instrumental in ensuring the scalability, reliability, and maintainability of embedded software systems. This thesis delves into the comprehensive process of developing a standard AUTOSAR Complex Device Driver (CDD) using Vector's Generator development kit, with a specific focus on implementing a Real-Time Clock (RTC) as a practical example. The research presents a detailed exploration of the methodologies, tools, and best practices involved in the development process, addressing both theoretical and practical aspects. Emphasis is placed on developing a CDD that can be tailored and reused within projects, adhering to AUTOSAR standards, ensuring compatibility, and optimizing performance within an automotive context.

Contents

List of Tables

List of Figures

Acronyms

AUTOSAR	AUTomotive Open System ARchitecture
CDD	Complex Device Driver
SWC	Software Component
BSW	Basic Software
BSWMD	Basic Software Module Definition
ECU	Electronic Control Unit
ECUC	ECU Configuration
ECUAL	ECU Abstraction Layer
MCAL	Microcontroller Abstraction Layer
RTC	Real-Time Clock
RTE	Runtime Environment
VFB	Virtual Functional Bus
HAL	Hardware Abstraction Layer
BCD	Binary Coded Decimal
I2C	Inter-Integrated Circuit
SDA	Serial Data
SCL	Serial Clock
SPI	Serial Peripheral Interface
ACK	Acknowledgement
NACK	Negative Acknowledgement
XML	Extensible Markup Language
ARXML	AUTOSAR XML
IDE	Integrated Development Environment
DDD	Detailed Design Description
GenDevKit	Generator Development Kit

1 Introduction

The automotive industry has witnessed a significant transformation in recent years, with the increasing integration of complex electronic systems in modern vehicles. This evolution has necessitated the development of standardised software architectures to manage the growing complexity of automotive software. The Automotive Open System Architecture (AUTOSAR) has emerged as a leading standard in this domain, providing a framework for the development of modular and reusable software components.

Within the AUTOSAR architecture, Complex Device Drivers (CDDs) play a crucial role in interfacing with specialized hardware components that are not adequately supported by standard AUTOSAR modules. The development of CDDs requires a deep understanding of both the AUTOSAR specification and the specific hardware being interfaced. This thesis focuses on the development of a CDD for a Real-Time Clock (RTC) using Vector's Generator Development Kit, a comprehensive suite of tools designed to facilitate AUTOSAR-compliant software development.

The choice of an RTC implementation as the focus of this study is motivated by the critical importance of accurate timekeeping in automotive systems. RTCs are essential for various functions, including timestamping of diagnostic data, scheduling of periodic tasks, and synchronization of distributed systems. By examining the development process of an RTC driver, this research aims to provide valuable insights into the broader challenges and methodologies associated with CDD development in the AUTOSAR context.

2 Objectives

The primary objective of this thesis is to establish a standardised methodology for the development of a Complex Device Driver (CDD) within the AUTOSAR (Automotive Open System Architecture) framework, specifically tailored for an automotive manufacturing company utilizing Vector's generator development kit. This research aims to address the current challenges faced by automotive manufacturers in the development and integration of CDDs in AUTOSAR-compliant systems, which are critical for the efficient and reliable operation of complex automotive electronic systems.

Firstly, the thesis seeks to establish a comprehensive framework for the development of AUTOSAR CDDs. This involves identifying and defining the necessary steps and methodologies that ensure consistent and efficient driver development across different teams and projects within the organization. By achieving this, the thesis aims to reduce variability and enhance the reliability of the software components developed.

Secondly, the thesis aims to optimize the use of Vector's Generator Development Kit by developing guidelines and best practices tailored for the automotive manufacturer's specific needs. This includes an in-depth analysis of the GenDevKit's capabilities, identifying how these can be leveraged to improve the development process, and ensuring that the resulting CDDs are both compliant with AUTOSAR standards and aligned with the manufacturer's operational requirements.

Furthermore, the thesis seeks to integrate the standardised development process into the existing software development lifecycle of the company. This objective involves tailoring the process to fit seamlessly with current workflows, tools, and practices while ensuring minimal disruption. The integration is expected to facilitate better collaboration among software engineers and other stakeholders, ultimately leading to improved project management and delivery timelines.

The third objective is to validate the proposed methodology through a series of case studies or pilot projects within the company. These case studies will involve the development of specific CDDs using the proposed standardised process, with the outcomes being evaluated in terms of performance, reliability, and compliance with AUTOSAR specifications. The results of these case studies will provide empirical evidence to support the effectiveness of the standardised methodology and will offer insights into potential areas for further refinement.

3 AUTOSAR

AUTomotive Open System ARchitecture (AUTOSAR) is a development partnership of automotive interested parties founded in 2003. It is focused on creating and establishing an open and standardised software architecture for automotive electronic control units (ECUs). Goals include the scalability to different vehicle and platform variants, transferability of software, the consideration of availability and safety requirements, a collaboration between various partners, sustainable use of natural resources, and maintainability during the product lifecycle.

AUTOSAR was formed in July 2003 by Bavarian Motor Works (BMW), Robert Bosch GmbH, Continental AG, Mercedes-Benz Group AG (formerly Daimler-Benz, then DaimlerChrysler), Siemens VDO, and Volkswagen AG to promote an open industry standard for automotive electrical-electronic (E/E) architecture. In November 2003, Ford Motor Company joined as a core partner, and in December, Groupe PSA (formerly PSA Peugeot Citroën) and Toyota Motor Corporation joined. The following November, General Motors also became a core partner. After Siemens VDO was acquired by Continental in February 2008, it ceased being an independent core partner.

AUTOSAR provides specifications of basic software modules, defines application interfaces and builds a common development methodology based on standardised exchange format. Basic software modules made available by the AUTOSAR layered software architecture can be used in vehicles of different manufacturers and electronic components of different suppliers, thereby reducing expenditures for research and development.

The AUTOSAR architecture is characterized by a layered structure that promotes modularity, scalability, and reusability of software components across diverse vehicle platforms and electronic control units (ECUs). At the highest level, the Application Layer houses Software Components (SWCs) that implement specific vehicle functions. These SWCs are designed to be hardware-independent, facilitating their reuse across different ECUs. Examples of such functions include engine management, transmission control, and advanced driver assistance systems (ADAS).

Beneath the Application Layer lies the Runtime Environment (RTE), which serves as a communication middleware. The RTE provides a virtual function bus that enables seamless communication between SWCs, irrespective of their physical location within the vehicle network. This layer is responsible for implementing the AUTOSAR COM stack and managing the execution of runnables based on timing and event triggers.

The Basic Software (BSW) Layer forms the foundation of the AUTOSAR architecture, offering standardised software modules for hardware abstraction, system services, and communication. This layer is further subdivided into the Services Layer, ECU Abstraction Layer, and Microcontroller Abstraction Layer (MCAL). These sublayers provide operating system functionality, hardware-independent interfaces to ECU-specific peripherals, and direct access to microcontroller peripherals, respectively.

3.1 Classic platform

The AUTOSAR classic platform is the standard for embedded real-time ECUs based on OSEK. Its main deliverable is specifications.

The architecture distinguishes between three software layers that run on a microcontroller: application, runtime environment (RTE) and basic software (BSW). The application software layer is mostly hardware independent. Communication between software components and access to BSW happens via RTE, which represents the full interface for applications.

The BSW is divided in three major layers and complex drivers:

- Services
- Electronic control unit (ECU) abstraction
- Microcontroller abstraction
- Services are divided further, into functional groups representing the infrastructure for system, memory and communication services.

One essential concept of the Classic Platform is the Virtual Functional Bus (VFB). This virtual bus is an abstract set of RTEs that are not yet deployed to specific ECUs and decouples the applications from the infrastructure. It communicates via dedicated ports, which means that the communication interfaces of the application software must be mapped to these ports. The VFB handles communication within the individual ECU and between ECUs. From an application point of view, no detailed knowledge of lower-level technologies or dependencies is required. This supports hardware-independent development and usage of application software.

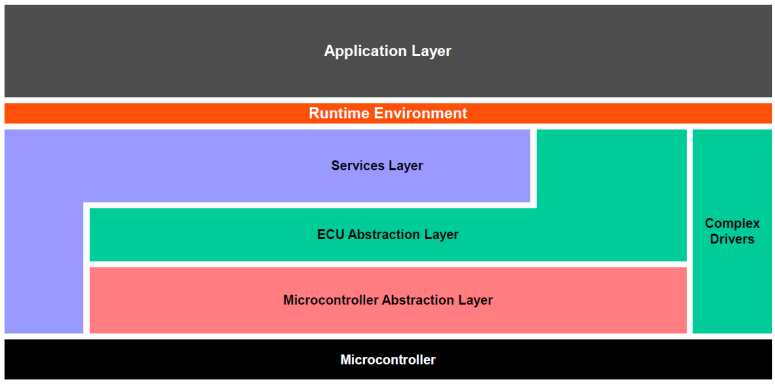

Figure 1: Layers' overview of AUTOSAR architecture

3.2 Complex device drivers

Complex Device Drivers are a specialized category within the AUTOSAR architecture that provide a flexible and efficient means to manage and control hardware devices that exhibit non-standard or highly complex behaviour. Unlike standard device drivers in the AUTOSAR BSW, which are designed to handle common hardware interfaces and functionalities in a standardised manner, CDDs are employed when the interaction with the hardware cannot be sufficiently encapsulated by the standard BSW modules. This situation typically arises in cases where the hardware exhibits unique characteristics, requires highly specific configurations, or demands performance optimizations that cannot be achieved through the generic BSW interfaces.

The role of CDDs within the AUTOSAR architecture is to bridge the gap between the standardised software layers and the unique requirements of specific hardware components. These drivers are typically implemented in scenarios where the hardware peripherals, such as sensors, actuators, or communication interfaces, require direct and detailed control that is not feasible through the abstraction provided by the BSW. For instance, a CDD might be necessary for handling non-standard communication protocols, managing complex timing requirements, or interfacing with hardware that requires precise real-time control beyond the capabilities of the standardised AUTOSAR services.

The implementation of a CDD involves direct interaction with hardware registers, interrupt service routines, and other low-level operations that are not abstracted by the AUTOSAR BSW. This low-level interaction is necessary to meet the stringent performance and timing requirements of certain automotive applications. Moreover, CDDs often include custom algorithms and logic

tailored to the specific hardware component, enabling the software to exploit the full potential of the hardware's capabilities. Such custom implementations are critical in achieving the desired functionality and performance, especially in safety-critical systems where precise hardware control is paramount.

Despite their importance, the use of CDDs in an AUTOSAR-based system introduces several challenges. One of the primary challenges is ensuring that the CDDs are integrated seamlessly with the rest of the AUTOSAR stack. This integration requires careful design to maintain compatibility with the standardised interfaces and services defined by AUTOSAR. Moreover, the development of CDDs demands a deep understanding of both the hardware being controlled and the AUTOSAR framework to ensure that the CDD does not inadvertently compromise the overall system's integrity, safety, or reliability.

Another significant challenge associated with CDDs is their impact on the system's portability and reusability. Since CDDs are typically tailored to specific hardware, they are not as easily portable across different platforms as standard AUTOSAR modules. This lack of portability can complicate the reuse of software components when different hardware platforms are involved, potentially increasing development time and costs. To mitigate this issue, the design of CDDs often involves a trade-off between achieving the desired hardware-specific functionality and maintaining a level of generality that allows for some degree of portability across different hardware configurations.

Furthermore, the development and validation of CDDs require rigorous testing to ensure that they meet the stringent requirements of automotive systems, particularly in terms of safety and reliability. Given that CDDs operate at a low level and often handle critical system functions, any errors or faults in their implementation can have severe consequences. Therefore, extensive validation processes, including hardware-in-the-loop testing, are typically employed to verify the correct operation of CDDs under various conditions. This testing is crucial to ensure that the CDDs perform reliably in real-world automotive environments, where they must operate within the constraints of real-time processing, limited computational resources, and strict safety standards.

In addition to their technical challenges, the use of CDDs also presents organizational and process-related considerations. The development of CDDs often requires collaboration between different teams, including hardware designers, software developers, and systems engineers. This collaboration is essential to ensure that the CDDs are developed in alignment with the overall system architecture and meet the specific requirements of the hardware components.

Moreover, the integration of CDDs into the AUTOSAR-compliant system must adhere to the stringent processes and guidelines defined by the automotive industry, particularly those related to functional safety standards such as ISO 26262.

4 Methodology

This chapter provides an in-depth methodology for developing a standard AUTOSAR CDD using Vector's Generator Development Kit (GenDevKit) and EParm. The process encompasses the entire software development lifecycle, from requirements elicitation to the final integration and testing phases, ensuring compliance with AUTOSAR standards while leveraging the tools provided by Vector Informatik.

4.1 Requirements elicitation and analysis

The development process begins with a comprehensive analysis of the requirements for the target complex device. This involves gathering detailed functional and non-functional requirements, which are typically provided by the OEM (Original Equipment Manufacturer) or derived from the system-level specifications. The requirements are then documented in a structured format, often using a requirements' management tool that supports traceability.

The analysis phase also includes the identification of any specific constraints or limitations imposed by the environment. This may involve interfacing with specialized hardware components, real-time performance considerations, or safety-critical requirements. The output of this phase is a well-defined set of requirements and an appropriate hardware capable to handle these, which will guide the subsequent stages of development.

4.2 Software architecture design

The CDD is typically placed within the BSW layer and interacts with both the hardware abstraction layer (HAL) and the RTE. During the architectural design, the CDD's interfaces, internal structure, and interaction with other AUTOSAR components are defined. This involves specifying the software components (SWCs), their ports, and communication mechanisms, such as sender-receiver or client-server communication.

The architectural design is documented using standardised modelling tools compatible with AUTOSAR, such as Vector's DaVinci Developer or similar tools, which support the generation of AUTOSAR-compliant descriptions (e.g., ARXML files).

4.3 Implementation phase

Once the architecture is defined, the implementation phase begins. This phase involves the actual coding of the Complex Device Driver, adhering to the specifications provided by the AUTOSAR standard and the architectural design. The implementation is carried out using the C programming language, which is the de facto standard for AUTOSAR-based development due to its efficiency and close-to-hardware capabilities.

Vector's Generator Development Kit (GenDevKit) plays a crucial role in this phase. The GenDevKit provides a set of tools and libraries designed to facilitate the development of AUTOSAR-compliant software components. It includes code generators, configuration tools, and a rich set of APIs that abstract the complexities of direct hardware interaction.

4.3.1 User-configurable section

This phase involves the use of Vector's EParm tool, which provides an interface for generating AUTOSAR-compliant configuration files (e.g., `.arxml` files), which are then integrated into the project. The configuration files are essential for the correct operation of the CDD, as they define how the driver interacts with the hardware and other software components.

These configuration files will contain a set of user-configurable parameters that will provide an easy and user-friendly way to tailor the integration of the CDD to each project, by means of checkboxes, drop-down menus, text boxes, etc.

4.3.2 Generator section

The GenDevKit provides a Java framework to build a C code generator. The set of APIs and classes of the framework are used to build and access the configurations defined by the user in the `.arxml` files and generate, dynamically, the C source code of the CDD using templates and substitution methods. It is also capable to generate SWC models and the RTE interfaces.

The implementation process is iterative, with frequent compilation and testing to ensure that the code meets the required functionality and performance criteria. The use of Vector's tools allows for automatic generation of boilerplate code and configuration files, reducing the likelihood of human error and speeding up the development process. The CDD source code is written in compliance with MISRA C guidelines, which are widely adopted in the automotive industry to ensure safety and reliability.

4.4 Integration and testing

After the implementation and configuration phases, the CDD is integrated into the complete AUTOSAR environment. This involves linking the CDD with other software components, configuring the RTE, and ensuring that the driver functions as expected within the broader system.

Testing is a crucial part of the integration process. The CDD is subjected to a series of tests to verify its functionality, performance, and compliance with AUTOSAR standards. These tests include unit tests, integration tests, and system-level tests.

The testing process ensures that the CDD meets all the requirements specified at the beginning of the project and that it integrates seamlessly with the rest of the system. Any issues identified during testing are addressed through iterative debugging and refinement of the code and configuration.

5 Case study

This section has been removed due to confidentiality reasons.

5.1 System requirement analysis

This section has been removed due to confidentiality reasons.

5.2 Hardware selection and analysis

This section has been removed due to confidentiality reasons.

5.2.1 Setting and reading the time

This section has been removed due to confidentiality reasons.

5.2.2 Alarm function

This section has been removed due to confidentiality reasons.

5.3 I2C protocol

This section has been removed due to confidentiality reasons.

5.4 Vector's Generator Development Kit

This section has been removed due to confidentiality reasons.

5.4.1 Basic software module definition

This section has been removed due to confidentiality reasons.

5.5 Code generation and integration

5.5.1 SW Component Static Description

This section has been removed due to confidentiality reasons.

5.5.2 SW Component Dynamic Description

This section has been removed due to confidentiality reasons.

5.5.3 File System Structure

This section has been removed due to confidentiality reasons.

5.5.4 BSW code

This section has been removed due to confidentiality reasons.

5.5.5 RTE integration

This section has been removed due to confidentiality reasons.

6 Verification and validation

Testing and validation are essential to ensure that the CDD operates correctly and meets all specified requirements. This phase involves a combination of unit testing, integration testing, and system testing.

Unit testing focuses on verifying the correctness of individual components within the CDD. Each component is tested in isolation to ensure that it performs its intended function correctly. Unit tests are designed to cover all possible scenarios, including edge cases and error conditions.

An extensive unit test suite, with eighty-six different test cases, has been developed to ensure that each unit of the RTC Driver performs as expected.

Each test case aims to validate the correctness of a unit of the software in isolation from the rest of the system. It also has been assured that the tests cover the maximum amount of lines of code possible, achieving a code coverage of 100 %.

On the other hand, Integration testing verifies that the CDD interacts correctly with other AUTOSAR components, including the BSW modules and the RTE. This phase involves simulating the interaction between the CDD and other software components, such as communication with the I2C driver or synchronization with the system clock. Integration tests are designed to ensure that the CDD functions correctly within the context of the entire ECU, with a focus on timing accuracy, communication reliability, and power management.

Last but not least, System testing involves validating the CDD within the context of the complete automotive system. This phase includes testing the CDD in a real or simulated ECU environment, using tools such as CANoe to simulate vehicle conditions and monitor the system's behaviour. System tests cover a wide range of scenarios, including normal operation, power cycles, and fault conditions.

The goal is to ensure that the CDD meets all functional and non-functional requirements and that it integrates seamlessly with the rest of the vehicle's software.

This involves reviewing the test results to ensure that all functional and non-functional requirements have been met. The validation process includes a thorough comparison of the test outcomes with the expected results to confirm that the CDD behaves as intended under all specified conditions.

If the CDD is intended for use in safety-critical systems, it may be subject

to compliance with standards such as ISO 26262. Documentation should include any necessary certifications, test results, and compliance reports that demonstrate the CDD's adherence to these standards. This documentation is crucial for achieving regulatory approval and ensuring the CDD's suitability for use in safety-critical applications.

All the test suites have been developed with the aim to be integrated and executed automatically in the Continuous Integration system of the project, giving the possibility to detect and fix bugs when they are introduced in the system.

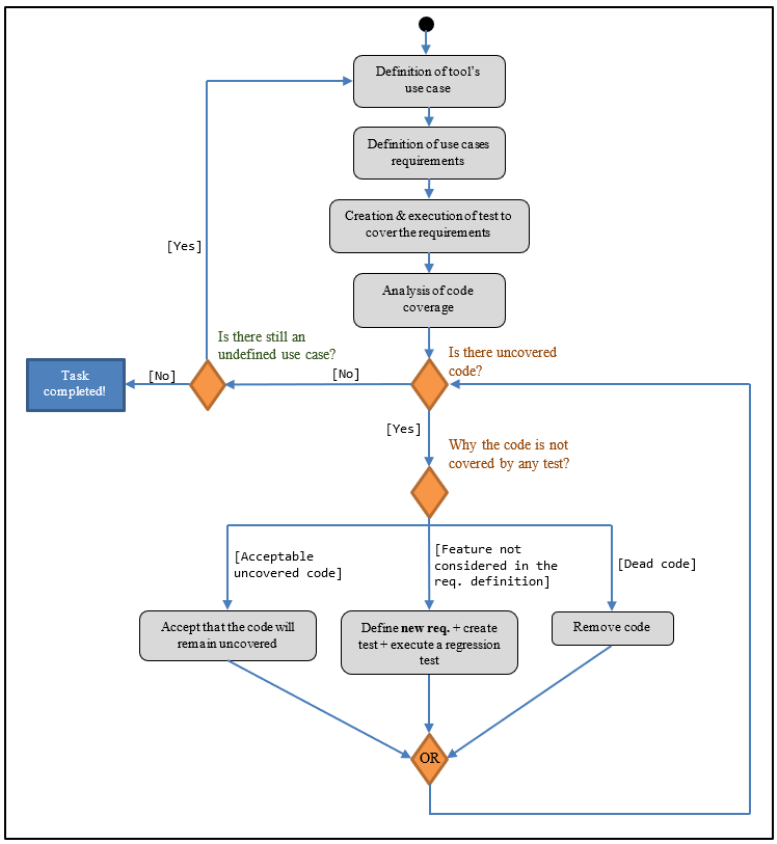

Figure 2: Testing development schema

7 Project deployment

The development of the component has been treated as a standalone project, instead of being part of a customer specific project. This means it has its own version control repository where the source code and the testing is stored. Therefore, this has opened the possibility to enable a Continuous Integration system, where builds and tests are run periodically to assure that no bugs are introduced to the system.

When a release of the repository is triggered, the RTC generator is built automatically with the latest changes available. The resulting output files are converted to a NuGet packet that is uploaded to the artefact repository of the company.

Following this path has given the opportunity to introduce the package management system, a novelty in the company. With this method, the project that requires this Software Component doesn't need to have the component directly copied to its repository, but it can download the specific version required from the artefact repository during the build phase of the project. This increases the speed of integration new components, as well as, decreases drastically the introduction of man-made errors into the SWC.

8 Conclusions

The development of a standard AUTOSAR Complex Device Driver (CDD) for a Real-Time Clock (RTC) using Vector's development kit represents a comprehensive process that spans requirement analysis, system design, implementation, testing, documentation, and release. Each phase of the development process is critical to ensuring that the CDD not only meets the stringent requirements of the automotive industry but also integrates seamlessly into the AUTOSAR framework, providing reliable and efficient operation within the target system.

This thesis has outlined the detailed steps involved in the development of the CDD, emphasizing the importance of adhering to AUTOSAR standards and leveraging the powerful tools provided by Vector's development kit. The process highlights key challenges, such as ensuring timing accuracy, managing power consumption, and achieving reliable communication between the CDD and other system components. Through a methodical approach to design, implementation, and testing, these challenges can be effectively addressed, resulting in a robust and reliable CDD that meets the needs of modern automotive systems.

The final product, supported by comprehensive documentation and a well-defined maintenance plan, ensures that the CDD can be effectively integrated into automotive systems, maintained over its lifecycle, and adapted to future requirements. The methodologies and best practices outlined in this thesis provide a valuable framework for the development of other CDDs within the AUTOSAR architecture, contributing to the ongoing evolution of standardised automotive software.

9 References

- AUTOSAR Layered Software Architecture:
  AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf
- AUTOSAR Methodology:
  AUTOSAR_TR_Methodology.pdf
- General Requirements on Basic Software Modules:
  AUTOSAR_SRS_BSWGeneral.pdf
- Requirements on Basic Software Module Description Template:
  AUTOSAR_RS_BSWModuleDescriptionTemplate
- General Specification of Basic Software Modules:
  AUTOSAR_SWS_BSWGeneral.pdf
- Basic Software Module Description Template:
  AUTOSAR_TPS_BSWModuleDescriptionTemplate.pdf
- Complex Driver design and integration guideline:
  AUTOSAR_EXP_CDDDesignAndIntegrationGuideline.pdf
- MISRA C 2012 Standard: Homepage:
  http://www.misra.org.uk
- RTC PCA2131 data sheet Rev. 2.0:
  https://www.nxp.com/docs/en/data-sheet/PCA2131.pdf
- I2C-bus specification and user manual Rev. 7.0:
  https://www.nxp.com/docs/en/user-guide/UM10204.pdf
- DaVinci Configurator Generator Development:
  GenDevKitDocumentation_Ext.pdf

10 Annex

10.1 Unit Test Report

This section has been removed due to confidentiality reasons.

10.2 Coverage Report

This section has been removed due to confidentiality reasons.

10.3 Requirements Test Report

This section has been removed due to confidentiality reasons.