Karlsruher Institut für Technologie

Institut für Technik der
Informationsverarbeitung (ITIV)

Institutsleitung
Prof. Dr.-Ing. Dr. h. c. J. Becker (Sprecher)
Prof. Dr.-Ing. E. Sax
Prof. Dr. rer. nat. W. Stork

Masterarbeit Nr. ID-3621

von Herrn B.Sc. Mustafa Burak **Cin**

Future Automotive Gateways: A Paradigm Shift from MCU Based AUTOSAR Implementations to FPGAs

Beginn:	11.12.2024
Abgabe:	11.09.2025
Betreuer:	Dipl.-Ing. Benjamin Schüler Daimler Truck AG
Korreferent:	Prof. Dr.-Ing. Dr. h.c. Jürgen Becker Institut für Technik der Informationsverarbeitung

Hauptreferent: Dr.-Ing. Tanja Harbaum

Ich versichere wahrheitsgemäß, die Arbeit selbstständig verfasst, alle benutzten Quellen und Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde, sowie die Satzung des KIT zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet zu haben.

Karlsruhe, den 04.09.2025

B. Sc. Mustafa Burak Cin

For Jülide

Acknowledgements

I would like to express my sincere gratitude to my academic supervisor and examiner, Dr.-Ing. Dipl.-Inform. Tanja Harbaum, for her constructive and detailed feedback, her thorough and insightful reviews, as well as for generously sharing her knowledge and scientific perspective. I am also grateful to Prof. Dr.-Ing. Dr. h. c. Jürgen Becker, who first inspired me with the idea of conducting my master's thesis at ITIV.

I could not have undertaken this journey without the support of Benjamin Schüler, my manager at Daimler Truck AG and supervisor for this thesis.

I would be remiss not to mention the Daimler Truck Academic Programs, which made my master's studies possible by providing the necessary framework to pursue my degree while working.

Words cannot express my gratitude to my colleagues:

- Alican Yalçın, for his excellent soldering expertise and support in assembling the CAN extension boards, that I designed for this thesis.
- Deepak Naik, for his insightful reviews on the AUTOSAR related chapters and our in-depth discussions on the topic.
- Kevin Schmidt, for listening to my concept in early stages, his valuable insights and inspiring discussions on the latest advancements in IT data processing.
- Lucas Mauser, for his valuable guidance and knowledge sharing on how to conduct a systematic literature review.
- Matthias Schulz, for providing valuable feedback on the AUTOSAR-related subjects and our engaging discussions on this at the office.
- Sushmita Sahana, for her support and collaborative brainstorming during the debugging sessions of the AUTOSAR implementation in this thesis.
- Tayte Daniel Waterman, for sharing valuable perspectives during our discussions about hardware accelerated routing concepts and implementations.
- Wolfgang Griebel, also known as the first driver operating a vehicle with passengers equipped with a CAN XL network [1], for his thorough review and valuable feedback on in-vehicle networks, CAN, and many other chapters of this thesis.

Lastly, I wish to express my heartfelt thanks to my fiancée Jülide, my family and my friends for their patience and support throughout this thesis.

Abstract

Automotive Gateways are fundamental components of In-Vehicle Networks (IVNs) that interconnect different networks, performing routing and protocol translations between them. The majority of today's gateways are processor based solutions implemented on microcontrollers utilizing the Automotive Open System Architecture (AUTOSAR) Classic software architecture. AUTOSAR introduces significant complexity, often requiring long development cycles and large development teams working over several years. Therefore, it is associated with high development costs as well as high initial investments due to required tooling, licenses and basic software stacks. While such complexity and associated costs may be justified for application centric, high end Electronic Control Units (ECUs), in majority of AUTOSAR Gateway projects these substantial efforts and resources fail to deliver adequate performance. These Gateway implementations often suffer from high Central Processing Unit (CPU) loads and in some cases even data loss. Although, recent CPU offloading solutions introduced by silicon vendors have improved the hardware capabilities to address some of the previously mentioned challenges, they remain insufficient in meeting non-functional requirements, particularly flexibility, complexity and cost efficiency. In parallel, emerging trends such as Software Defined Vehicles (SDVs) and Zonal Electric/Electronic (E/E) Architecture are anticipated to significantly redefine the requirements for Automotive Gateways. These trends introduce opportunities for new Gateway designs that were previously constrained by the existence of high number of ECUs and networks in a vehicle. In this thesis, the key requirements for future Automotive Gateways, in particular for Zonal Gateways, are first identified. Subsequently, a critical analysis of AUTOSAR based Gateway solutions is conducted, including its implementation on a microcontroller. Finally, the thesis proposes and demonstrates a novel, hardware centric Gateway architecture realized in the Programmable Logic (PL) of an Field Programmable Gate Array (FPGA), offering a new approach to meet the challenges and opportunities of future Zonal Gateways.

Zusammenfassung

Fahrzeug-Gateways sind wesentliche Komponenten von fahrzeuginternen Netzwerken, die unterschiedliche Netzwerke miteinander verbinden und dabei das Routing sowie die Protokolübersetzung zwischen ihnen realisieren. Der Großteil der heutigen Gateways sind prozessorbasierte Lösungen, die auf Mikrocontrollern implementiert und unter Verwendung der Automotive Open System Architecture (AUTOSAR) Classic Softwarearchitektur realisiert werden. AUTOSAR bringt eine erhebliche Komplexität mit sich, die häufig lange Entwicklungszyklen sowie große Entwicklungsteams erfordert, die über mehrere Jahre an solchen Projekten arbeiten. Daher ist es mit hohen Entwicklungskosten sowie beträchtlichen Anfangsinvestitionen verbunden, unter anderem aufgrund der erforderlichen Werkzeuge, Lizenzen und Basissoftwarestacks. Während sich eine derartige Komplexität und die damit verbundenen Kosten bei applikationzentierten, leistungsstarken Steuergeräten möglicherweise rechtfertigen lassen, führen sie in der Mehrzahl der AUTOSAR-Gateway-Projekte trotz des erheblichen Aufwands und Ressourceneinsatzes nicht zu einer adäquaten Leistungsfähigkeit. Viele dieser Gatewayimplementierungen leiden unter einer hohen CPU-Auslastung und in einigen Fällen sogar unter Datenverlusten. Obwohl, von Halbleiterherstellern eingeführte CPU-Offloading-Lösungen die Hardwarefähigkeiten zur Bewältigung einiger der zuvor genannten Herausforderungen verbessert haben, sind sie nach wie vor unzureichend, um nicht-funktionale Anforderungen wie Flexibilität, Komplexität und Kosteneffizienz zu erfüllen. Parallel dazu wird erwartet, dass aufkommende Trends wie Software Defined Vehicle (SDV) und die zonale elektrische/elektronische (E/E)-Architektur die Anforderungen an Fahrzeug-Gateways erheblich neu definieren werden. Diese Entwicklungen eröffnen Möglichkeiten für neue Gateway-Designs, die zuvor durch die hohe Anzahl von Steuergeräten und Netzwerken in einem Fahrzeug eingeschränkt waren. In dieser Arbeit werden zunächst die Anforderungen für zukünftige Fahrzeug-Gateways, insbesondere zonaler Gateways, identifiziert. Anschließend wird eine kritische Analyse von AUTOSAR-Basierten Gateway-Lösungen, einschließlich ihrer Implementierung auf einem Mikrocontroller, durchgeführt. Abschließend wird eine neuartige, hardwarezentrierte Gateway-Architektur vorgeschlagen und in der Programmierlogik eines Field-Programmable-Gate-Arrays (FPGA) implementiert, um die zuvor genannten Herausforderungen und neu entstehenden Chancen für zukünftiger zonaler Gateways zu adressieren.

Contents

1 Introduction

Automotive Gateways serve as the fundamental network processing elements within the In-Vehicle Network (IVN) [2], enabling the routing of data across multiple vehicular networks. Their importance has grown significantly and is expected to further increase, particularly as in-vehicle data traffic is projected to exceed a Gbit/s rates [2].

Currently, most Gateway implementations are based on microcontrollers running the Automotive Open System Architecture (AUTOSAR) Classic software architecture. The automotive industry, however, is undergoing a transformation in Electric/Electronic (E/E) architectures, driven by advancements in IVN technologies and the shift towards the Software Defined Vehicle (SDV). These trends introduce new requirements for gateway design and may enable the adoption of more hardware-centric solutions.

This thesis examines the requirements and Key Performance Indicatorss (KPIs) relevant to both current and emerging Automotive Gateway architectures. It evaluates the advantages and limitations of microcontroller-based Gateways using the AUTOSAR Classic architecture, including a practical implementation on a microcontroller. Furthermore, the thesis proposes and implements a hardware-centric alternative using an Field Programmable Gate Array (FPGA) platform, and benchmarks this solution against the traditional microcontroller-based approach.

1.1 Motivation

Microcontroller-based Gateways implementing the AUTOSAR Classic architecture often struggle to meet emerging requirements for flexibility, scalability, latency, and determinism, particularly as demanded by trends in SDVs and zonal E/E architectures [2]. Moreover, these gateways present notable challenges even in current applications, such as high initial investment, increased system complexity [3], elevated development costs, and lengthy development cycles.

Despite these drawbacks, AUTOSAR remains as the mainstream standard for automotive software architectures [4, 5], offering advantages such as software portability and support for multi-supplier development. These benefits are most evident in complex Electronic Control Units (ECUs) projects, in which complex application design is present. However, Gateway ECUs typically do not host application-level functionality, thereby limiting the practical advantages of using AUTOSAR in this context.

From a systems engineering perspective, Automotive Gateways primarily enable other ECUs to deliver differentiating vehicle functions, rather than providing such functions themselves. Nonetheless, under current microcontroller-centric AUTOSAR Classic paradigms, the development costs and timelines for Gateways are comparable to those for feature rich ECUs. The expansion of routing tables, which must be statically defined in AUTOSAR, further contributes to ongoing development and maintenance challenges.

The emergence of Zonal E/E Architectures and the shift towards SDVs have the potential to fundamentally reshape Automotive Gateway design [6]. In traditional distributed E/E architectures, the Central Gateway (CGW) served as the principal network processor. In contemporary Domain Centralized E/E Architectures, the CGW continues to play a central role, while domain controllers increasingly combine Gateway and application functionalities.

In both architectural models, the CGW often performs additional tasks such as diagnostics, security, and global time synchronization, frequently on the Central Processing Unit (CPU) [7]. As in-vehicle data traffic grows [2], reliance on the CPU for both communication and system services increases computational load, which can degrade throughput and reliability.

Recently, automotive silicon vendors have introduced accelerated networking solutions for microcontrollers [8, 9], designed to offload the CPU in Gateway applications. While these solutions may be suitable for High Performance Computers (HPCs) in Zonal E/E Architectures, where Gateway functionality remains critical, the inherent drawbacks of complexity, cost, and development time associated with microcontroller-based AUTOSAR solutions are likely to persist. These trade-offs are particularly challenging for domain controllers and low-end vehicles that do not employ HPCs.

A clear gap exists in both the academic literature and industry practice regarding the architecture of future Zonal and low-maintenance Gateways. The primary aim of this thesis is to define first fundamental requirements for future Gateways, identify the limitations of microcontroller-based AUTOSAR Classic Gateways and to address these limitations by proposing and implementing a hardware-centric alternative based on FPGA technology for future Zonal Gateways.

1.2 Contribution and Structure

This thesis makes three principal contributions to the field:

First, it identifies and analyzes the functional and structural requirements for future gateways, with a particular focus on Zonal Gateways.

Next, it presents a complete implementation of a microcontroller-based AUTOSAR Classic Gateway using the Infineon TC399XX, illustrating both the strengths and limitations of this approach.

Finally, it proposes and implements a hardware-centric Gateway architecture on a Xilinx (AMD) Zynq UltraScale+™ MPSoC ZCU102, utilizing only the Programmable Logic (PL). This prototype includes the design of a Controller Area Network (CAN)-to-Intellectual Property (IP) router.

The thesis is organized as follows:

- Chapter 2 (*Background*) presents the theoretical foundations and prerequisite knowledge underlying this research.
- Chapter 3 (*Literature Review*) surveys related work, with a focus on AUTOSAR, E/E architectures, and Gateways.
- Chapter 4 (*Concept*) introduces the conceptual framework of the thesis, discusses key drivers of Gateway requirements, and examines the impact of SDV and Zonal E/E Architectures.
- Chapter 5 (*Implementation*) details the realization of both microcontroller-based and FPGA-based Gateway solutions.
- Chapter 6 (*Evaluation*) presents experimental results and provides a comparative analysis with respect to the requirements defined in Chapter 4.
- Chapter 7 (*Conclusion*) summarizes the main findings, outlines limitations, and suggests directions for future research.

2 Background

2.1 Communication Components

This section provides definitions and explanations of technical terms such as Nodes, interfaces, ECUs and networks that will be referenced frequently throughout the remainder of this thesis.

2.1.1 Node

A Node [10] consists of a processing unit that performs the data exchange with processing units in other devices through a communication protocol controller and a transceiver. The processing unit in a Node is commonly referred to as the host [10]. An illustration of a Node is given in the Figure 2.1 (a).

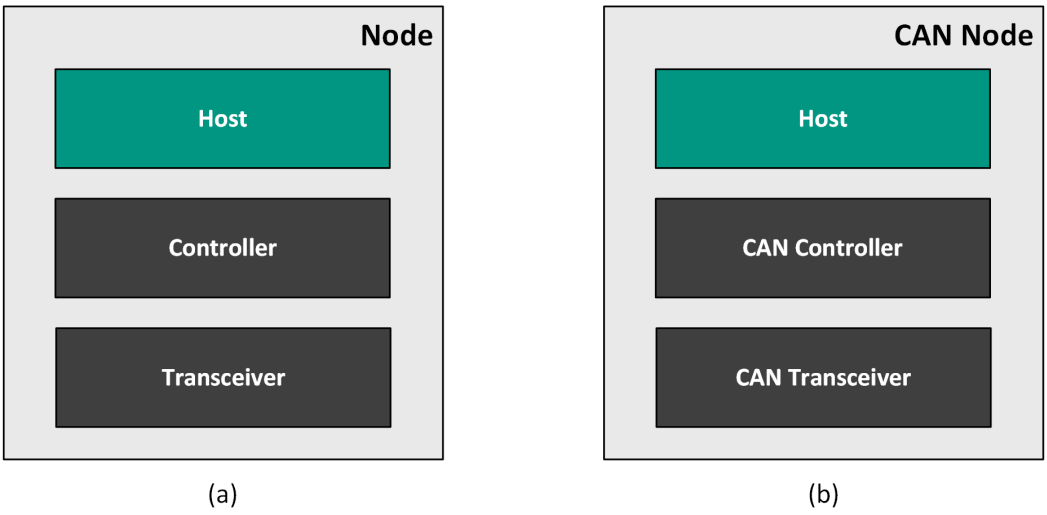

Figure 2.1: Representation of a Node and a CAN Node [10]

In this thesis, the CAN (refer to the Section 2.3) is the main communication protocol being used. When a Node is implemented with the CAN communication capability , it is referred to as a CAN Node and typically comprises a host, a CAN controller and a CAN transceiver. An illustration of a CAN Node is given in the Figure 2.1 (b).

2.1.2 Electronic Control Unit (ECU)

ECUs are essential computing systems in vehicles, originally developed to manage the operation of sensors and actuators to enable specific vehicle functions [2]. In the context of this thesis, an ECU is considered as comprising a host along with its communication interfaces.

In the automotive industry, the host of an ECU is typically implemented using a microcontroller [10]. However, in certain scenarios, it may also be realized using a FPGA[11], an Application Specific Integrated Circuit (ASIC) [11] or a microprocessor [12].

The communication interfaces of an ECU facilitate the data exchange with other ECUs, commonly referred to as the inter-ECU communication. The combination of a CAN controller and a CAN transceiver within a CAN Node constitutes a CAN interface.

An ECU may have multiple communication interfaces. Consequently, an ECU equipped with several CAN interfaces can connect to multiple CAN channels and functions as a CAN Node on various CAN networks within a vehicle. Within the scope of this thesis, the terms CAN Node and ECU may be used interchangeably. Figure 2.2 illustrates a representative ECU with two CAN interfaces.

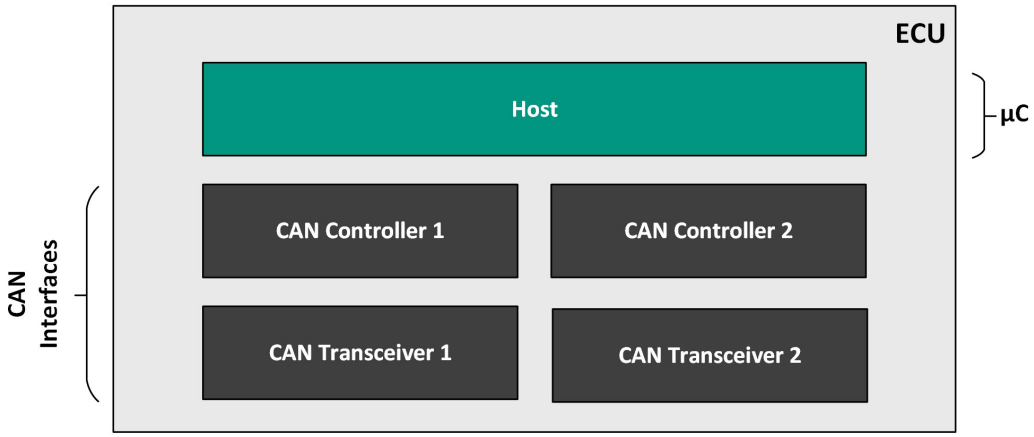

Figure 2.2: Representation of an ECU with two CAN interfaces [10]

The CAN controller in an ECU may either be integrated into the host or implemented as an external CAN controller as a hardware. Both configurations can be found later in this thesis in the Figure 2.12. On the other hand, although some CAN controllers are available in the industry with multi-channel support, for the sake of simplicity, the illustrations in this thesis depict each CAN channel with a dedicated CAN controller and a CAN transceiver.

2.1.3 Network

A network, specifically a CAN network consists of two or more ECUs typically connected through a linear bus topology. The CAN interfaces of these ECUs are connected to the CAN network through physical layer components, namely the CAN High (CAN-H) and CAN Low (CAN-L) wires.

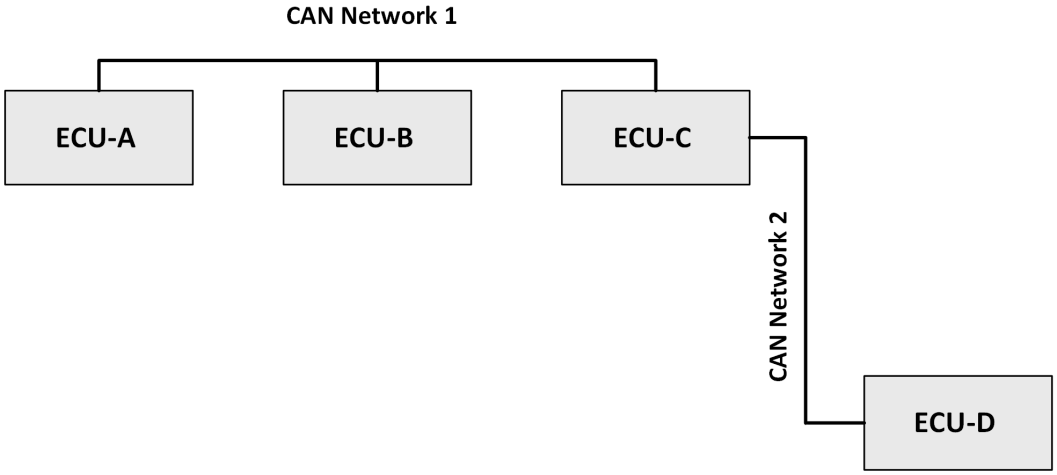

Figure 2.3: A representative E\E Architecture

A representative E/E architecture with four ECUs and two CAN networks is given in the Figure2.3, where there are three ECUs (ECU-A, ECU-B, ECU-C) connected to the CAN Network 1, while two ECUs (ECU-C and ECU-D) connected to the CAN Network 2. Although not explicitly stated, it can be deduced from the Figure 2.3, that the ECU-C is equipped with two CAN interfaces, as it is connected to both CAN Network 1 and CAN Network 2. The presence of multiple communication interfaces does not inherently qualify an ECU as a gateway. The actual network configuration can only be determined through analysis of the ECU's circuit diagram and the software configuration. However, ECUslike ECU-C, typically functions as gateway ECUs and enables the data exchange between two distinct CAN networks.

2.2 Electric/Electronic (E/E) Architecture

This section provides the foundational concepts of Electric/Electronic Architectures, with particular focus on both Domain Centralized and Zonal E/E Architectures.

2.2.1 Domain Centralized E/E Architecture

E/E Architecture is the structural organization of a vehicle's electrical and electronic systems including ECUs, sensors, actuators, wiring and networks that is designed to achieve functional objectives [13].

The initial E/E Architecture trend was the Distributed E/E Architecture , in which each function was realized by a dedicated ECU [14, 2]. With rapidly increasing functions and amount of the software, this methodology became difficult to sustain and manage especially due to large number of ECUs.

The second generation of E/E Architecture is the Domain Centralized E/E Architecture, in which ECUs are distributed into networks according to their functionalities [14, 2], such as Powertrain Domain or Advanced Driver Assistance Systems (ADAS) Domain [15].

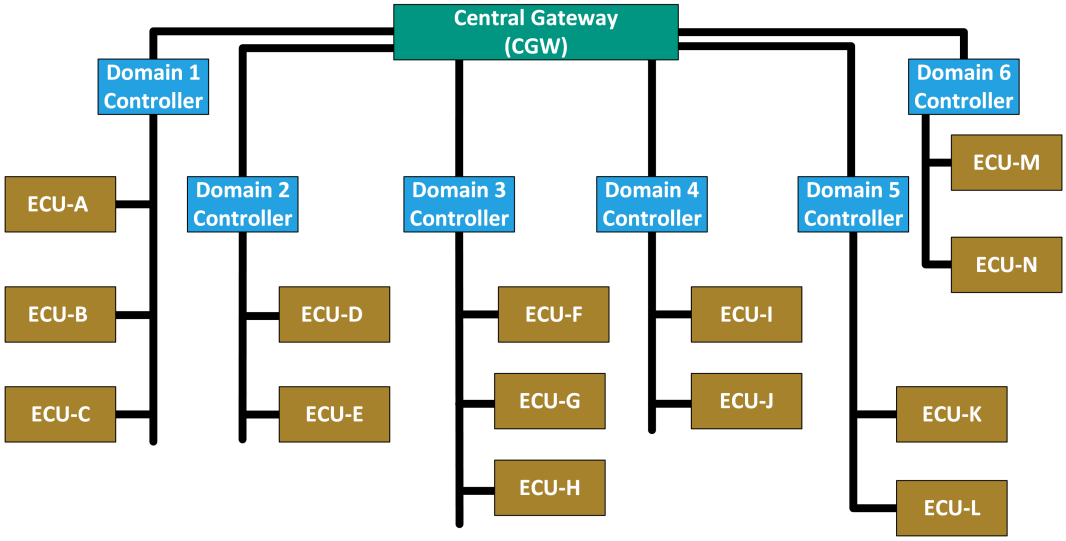

Figure 2.4: Domain Centralized E/E Architecture [14, 16, 13, 6]

Within this architecture, functions are grouped together [6] and each domain is managed by a dedicated domain controller ECU. Sub-domain ECUs are integrated under these domain controllers' sub networks. The Domain Controllers are connected to the CGW, which not only handles the protocol conversion and routing but also typically is responsible for roles such as diagnostics, security and network management coordination [13].

An illustration of the Domain Centralized E/E Architecture is given in the Figure 2.4.

The Domain Centralized E/E Architecture remains the mainstream E/E architecture used in today's vehicles [6]. Since ECUs are allocated to networks based on their functional roles rather than their physical placement within the vehicle, this architecture introduces challenges for Original Equipment Manufacturers (OEMs), particularly due to the long

wiring harnesses required [6, 13]. The wiring harness complexity might be even more challenging for heavy duty vehicle manufacturers, where vehicles are significantly longer than the passenger cars. Given that the wiring constitutes one of the heaviest and most costly elements of a vehicle [17], this challenge represents a major limitation.

Although the transition from the Distributed E/E Architecture to Domain Centralized E/E Architecture sought to move away from the One-ECU-per-function concept [14], the continuous expansion of vehicle functions, mainly due to naturally increased by electrification, autonomy, ADAS features, have nevertheless resulted in a significant increase in the number of ECUs in recent years.

2.2.2 Zonal E/E Architecture

The challenges outlined in the Subsection2.2.1 have driven the industry to seek alternative E/E Architecture solutions. In this context, the Zonal E/E Architecture has been proposed, wherein ECUs are distributed in the vehicle according to their physical locations. This approach aims to address aforementioned wiring harness challenges, as well as associated cabling costs [2, 6].

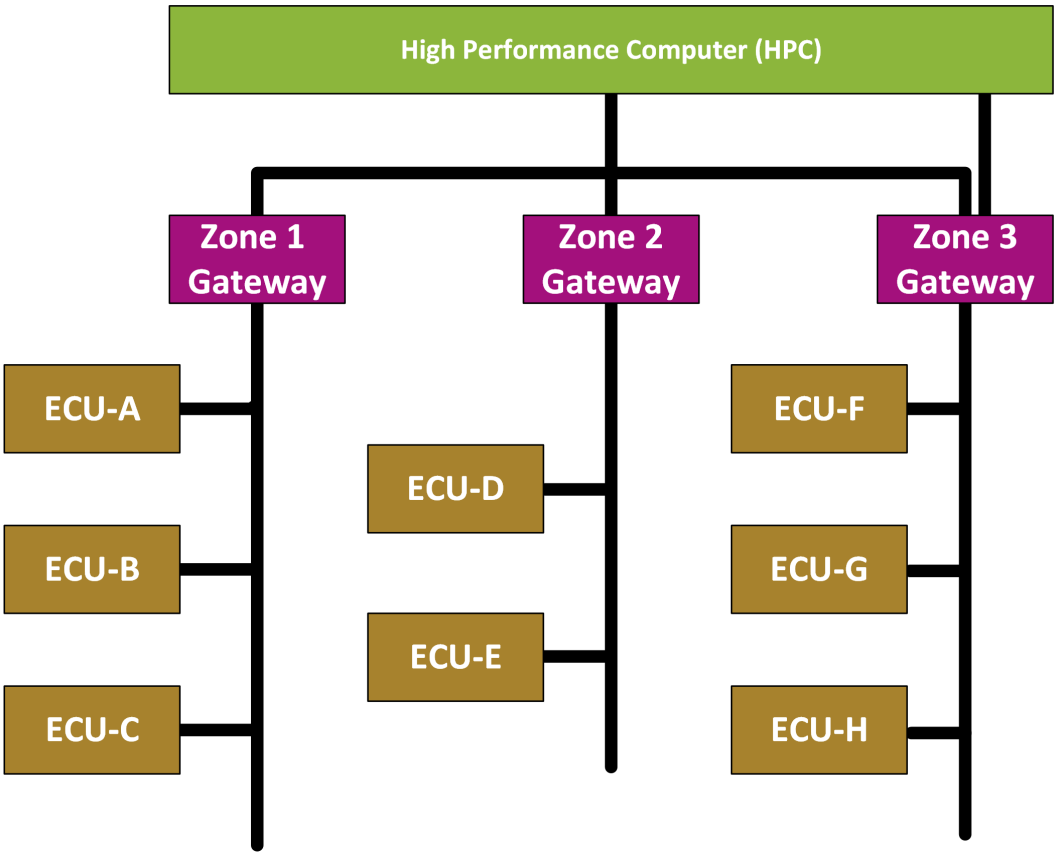

Figure 2.5: Zonal E/E Architecture [16, 13, 6]

In the Zonal E/E Architecture, a central vehicle computer, also referred as HPC, is responsible for implementing the majority of vehicle functions [6]. The physically distributed zones are equipped with dedicated Zonal Gateways, which manage the data traffic between the HPC and end nodes (Zonal ECUs).

An illustrative representation of the Zonal E/E Architecture for a heavy-duty vehicle is given in the Figure 2.5.

By consolidating most of vehicle functions within the HPC, this architecture significantly reduces the overall number of ECUs [13]. Since the network distribution is based on physical rather than functional, the number of networks are typically limited to four in passenger cars [2, 6] and three for heavy-duty vehicles where the vehicle length is considerably greater.

Through these design principles, the Zonal E/E Architecture aims to reduce overall system complexity and costs [2].

In this thesis, the transition toward Zonal E/E Architecture is regarded as an opportunity to redefine the existing gateway designs, with a particular focus on Zonal Gateways. The opportunities and potential implications of this architectural shift on Zonal Gateway designs will be examined in detail in the Subsections 4.2.1 and 4.2.2.

2.3 Controller Area Network (CAN)

This section provides an overview of the fundamentals of CAN Protocol, its generations, and its history.

With the increasing number of ECUs and the rising volume of the data exchange in vehicles, the use of point-to-point connections between ECUs has become inefficient. For instance, it is common for a single signal to be required by multiple ECUs, with point-to-point connection this necessitates duplicating the transmission over separate dedicated lines, leading to redundancy and complexity. Additionally, introduction of new features or improvements often requires significant modifications to the physical hardware and wiring harnesses. To overcome these challenges, the adoption of a bus system has become essential in the design of the in-vehicle networks.[10]

2.3.1 History of CAN

The transition to bus systems began several decades ago in the automotive industry. In 1983, the development of the CAN communication protocol has been started at Robert Bosch GmbH to address the evolving communication requirements of automotive industry at the time. During the development of CAN, a particular attention was given to followings:

- The protocol should be able to operate reliably in environments where the electromagnetic interference is high.
- The protocol should support rapid recovery following the error detection
- A non-destructive arbitration mechanism is required to guarantee that only one node transmits at a time.

Due to these features, the CAN protocol is capable of meeting the stringent hard real time requirements demanded by automotive applications. [18]

While the reduction of wiring complexity and harness weight represents one of the most substantial advantages of the CAN protocol compared to point-to-point connection, this emerged as a by product rather than a primary design criterion [19].

The development of the CAN protocol involved collaborative efforts between Bosch experts, Mercedes-Benz engineers and academic institutions culminating in its introduction in 1986 at the SAE Congress in Detroit [19]. The following year, in 1987, the first CAN controller chip 82526 is being produced and delivered by Intel [19, 18]. The first series application of the CAN protocol was introduced by Mercedes-Benz in the S-Class model in 1991 [10]. In 1993, formal standardization of the CAN protocol was achieved through the publication of ISO 11898 [19]. The initial implementation -the first generation of CAN- commonly referred to as Classical CAN or CAN 2.0 and it supports data rates up to 1 Mbit/s and allows for up to 8 data bytes in a CAN frame [20, 21].

The CAN protocol continues to evolve continuous enhancements. Following the initial ISO standard in 1993, two subsequent generations of CAN protocol has been introduced:

- The development of CAN Flexible Data-Rate (CAN FD) was initiated in 2011 by General Motors and Bosch with the objective of achieving higher data throughput [19]. Prior to this, several academic studies proposed extensions to the CAN protocol to enhance its capability [20]. Bosch has released the official specification of CAN FD in 2012 [22] and the protocol was formally introduced at the 13th international CAN Conference. CAN FD was incorporated into the ISO 11898 standard in 2015. This advancement increased the maximum payload size from 8 to 64 bytes and enabled data rates up to 8 Mbit/s with CAN FD Signal Improvement Capability (SIC) transceivers [23, 19, 24, 21].
- CAN extended data-field length (CAN XL) is the third and the latest generation of the CAN protocol. Its development began in 2018, initiated by a request from Volkswagen to address increased bandwidth requirements with the zonal architecture and high performance computing [19, 25]. In 2024, CAN XL was formally standardized under ISO 11898-1 and ISO 11898-2. This third generation CAN Protocol, CAN XL, introduces significant enhancements including an increased payload capacity up to 2048 bytes and support for data rates up to 20 MBit/s with CAN SIC XL transceivers. [19, 24, 26, 21].

2.3.2 Comparison of CAN Protocol Generations

In the Table 2.1 a summary and comparison of these can generations is being shown.

CAN Protocol	Number of Data Bytes	Maximum Data Rate	Year of Introduction
CAN 2.0	8 Byte	1 Mbit/s	1986
CAN FD	64 Byte	8 Mbit/s	2012
CAN XL	2048 Byte	20 Mbit/s	2024

Table 2.1: Comparison of different CAN protocol versions

The study in [21] compares different CAN protocol generations and proposes an optimal choice based on the payload size. In practical scenarios, the adoption and implementation of different CAN protocols by OEMs are influenced by a range of factors. A key consideration is that not all vehicle functions require data rates exceeding 1 Mbit/s which can be sufficiently supported even by the classical CAN. Furthermore, the E/E architectures developed by OEMs differ significantly across vehicle types and product lines. Typically, the latest in-vehicle network technologies are first implemented in the high-end models, while cost-sensitive products continue to be manufactured using comparatively older technologies. Although the classical CAN was introduced in 1986, it remains widely in use in in-vehicle networks and is expected to continue playing a prominent role, particularly among commercial vehicle OEMs, where the production volume is lower than passenger car OEMs and the cost sensitivity is a more critical factor.

Although Automotive Ethernet based in-vehicle network protocols -such as 10BASE-T1S, 100BASE-T1 and 1000BASE-T1- are increasingly being adopted in the automotive industry, particularly in the context of software defined vehicles and emergence of zonal architectures where Automotive Ethernet serves as the backbone of the E/E architecture, CAN based protocols are still expected to remain prevalent within subnetworks connecting zonal gateways and mechatronic ECUs.

The second generation, CAN FD protocol along with CAN FD SIC transceivers has been widely adopted by OEMs since its introduction. In contrast, CAN XL is still in its early stages of adoption with only a limited selection of CAN SIC XL transceivers and controllers currently available in the market. Nonetheless, several OEMs and Tier1 suppliers plan to implement the latest generation of CAN protocol, CAN XL within zonal architectures [24, 25].

In 2023, a proof of concept project led by Daimler Truck AG, in collaboration with Bosch, NXP, Rohde & Schwarz and Vector Informatik GmbH, involved replacing an existing CAN FD network in a Mercedes-Benz Citaro bus with a CAN XL network. This setup achieved a data rate of 14.5 Mbit/s over 60 meter cables while retaining the original CAN wiring harness used for CAN FD communication [1]. This research project

represented the first real-world application of CAN XL technology in a vehicle carrying passengers; however, its adoption for series production has not yet been confirmed and remains under evaluation.

The choice and usage of CAN protocol generation influences the design of a gateway ECU and impacts KPI such as CPU load and routing latency. However, the primary objective of this thesis is independent of the specific CAN protocol employed, and the findings can be generalized across different CAN protocol generations. Due to hardware availability for the implementation and comparability with existing gateway ECUs, the classical CAN protocol with 29-bit identifier is utilized for the implementation of this thesis.

2.3.3 Physical Layer of CAN

The physical layer of the CAN protocol is standardized in International Organization for Standardization (ISO) 11898-2 [27, 28] and except the Physical Coding Sublayer (PCS), it is implemented in the CAN transceiver hardware.

According to the definition in the Open Systems Interconnection (OSI) Model [29], the physical layer provides the mechanical, electrical, functional and procedural means to activate, maintain and deactivate physical connections for bit transmission between data-link-entities. Physical layer entities are interconnected by means of a physical medium. In the context of CAN, the physical layer entities correspond to CAN transceiver and a part of CAN controller -PCS-. The CAN Physical layer is standardized in ISO 11898-1 and ISO 11898-2. The OSI Layer model for a CAN communication is illustrated in this thesis in Figures 2.6 and later 2.12.

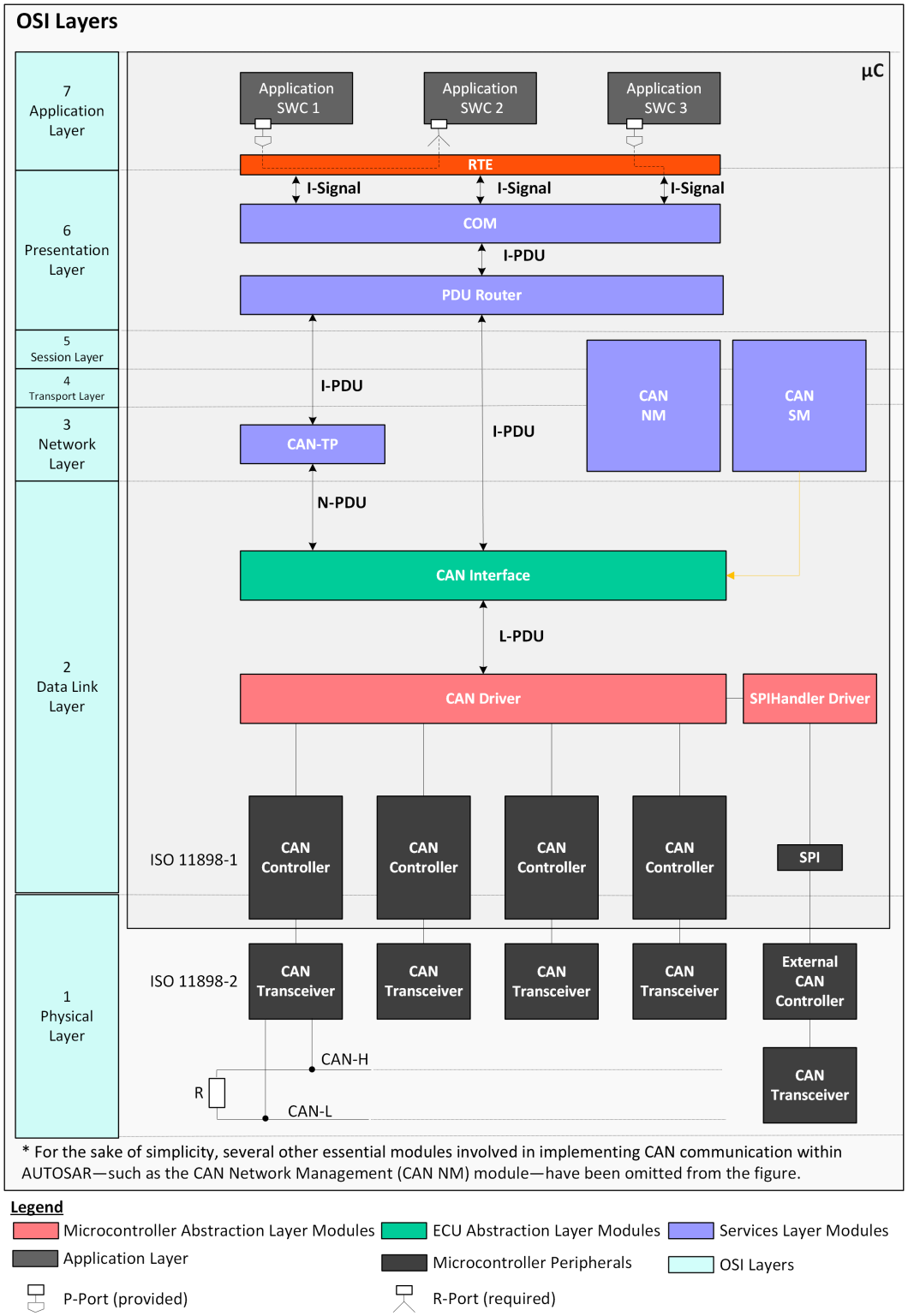

Figure 2.6: CAN Communication with Layered Architecture acc. to AUTOSAR versions >= 4.3 and <24-11

ISO 11898-2 [30] defines the bus state in CAN protocol. According to this standard, the dominant bus state represents the logical 0 (where the voltage difference between CAN-H and CAN-L cables are around 2 Volts) and the recessive bus state represents the logical 1 ($V_{\text{CAN-H}} - V_{\text{CAN-L}} \approx 0\,\text{V}$).

2.3.4 Data Link Layer of CAN

The data link layer of CAN protocol is standardized in ISO 11898-1 [27, 28] and implemented in a CAN controller.

According to the definition in the OSI Model [29], the data link layer is in charge of transferring the data link service data units. The data link service data unit corresponds to the CAN frame entity of the data link layer. The OSI Layer model for a CAN communication is illustrated in Figures 2.12 and 2.6 later in this thesis.

The CAN protocol standardizes several CAN frame formats [28, 27]. In this thesis, only CAN Classical Extended Frame Format (CEFF) for the data frame will be considered. An illustration of CEFF with 8 byte data field is given in the Figure 2.7.

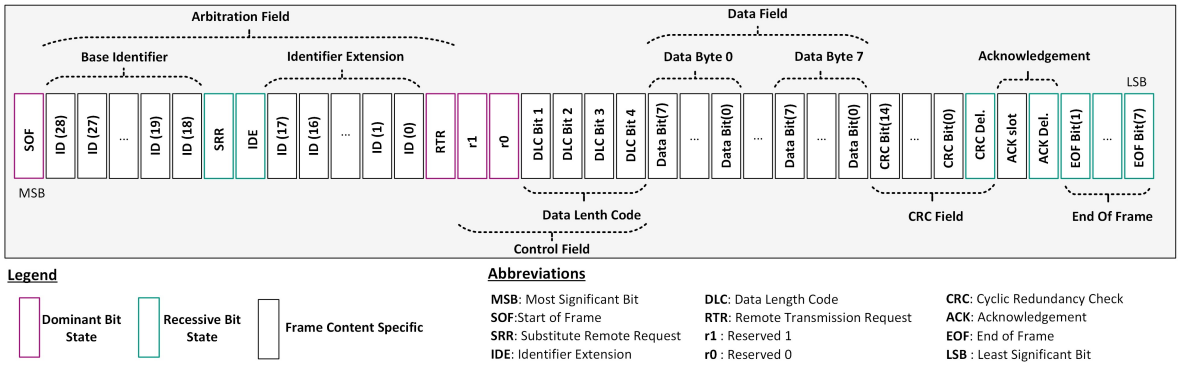

Figure 2.7: CEFF with 29 Bit Identifier [28]

In CEFF, there are fields with predefined bit states (dominant or recessive bit state), which will result into the bus state.

The transmission on the bus begins with the dominant Start of Frame (SOF) bit. Within each field, the Most Significant Bit (MSB) is transmitted first. In the data field, bytes are sent sequentially from Byte 0 to Byte 7 and with each byte, the bits are transmitted from bit 7 down to bit 0 [28].

The CAN protocol defines the bus access method for the CAN protocol by specifying a bitwise arbitration mechanism based on the CAN frame Identifier (ID). This arbitration

mechanism ensures that, in scenarios where multiple CAN nodes attempt to transmit frames simultaneously, only one frame is transmitted on the bus.

Lower CAN frame ID values have higher priority, as the CAN protocol designates the dominant bit state as logical '0'. Frames with higher priority -i.e. those with lower CAN frame ID values- win the arbitration and are allowed to continue being transmitted on the CAN bus.

All CAN Nodes, including those currently transmitting, continuously monitor the CAN bus as well as their own transmission [31]. If a CAN Node attempts to transmit a recessive bit but detects a dominant bit on the CAN bus, it recognizes that it has lost the arbitration. As a result, it ceases transmission and the node which sends the higher priority frame continues uninterrupted.

The arbitration mechanism is non destructive, meaning that the node which lost the arbitration will attempt to retransmit its frame once the bus is in idle state. The idle bus state can be identified via seven consecutive recessive bits marking the end of frame field, followed by the intermission field with three recessive bits.

2.4 Communication Design Entities

2.4.1 Communication Matrix

Distributed functionalities are realized by ECUs in a vehicle. ECUs require certain information from other ECUs to perform their own functions. This naturally results in a database created and maintained by the OEM, which stores the data transmitted and received between ECUs. This database is commonly referred to as communication matrix (a.k.a K-Matrix or CAN-Matrix). The data between ECUs are being transmitted via frames.

Typically frames and their attributes are defined by OEMs along with Protocol Data Units (PDUs) and signals. The communication matrix generated by the OEM results into an ECU Extract for an individual ECU. Further information about the ECU Extract is provided in the Section 2.5.

A representative design of a communication matrix is depicted in the Table 2.2 :

Although adherence to AUTOSAR standards is not mandatory for implementing the CAN protocol, AUTOSAR strictly conforms to the ISO CAN standardization. Furthermore, many OEMs opt to develop their communication matrices based on the constraints defined in the AUTOSAR System Template [32] . A comprehensive overview of the communication matrix, its relation and usage in AUTOSAR framework, and ECU Extract is provided in the Section 2.5.

Frame Name	Frame ID	Length (Byte)	Type	Launch Type	Timing	ECU-A	ECU-B	ECU-C	Diag Tester
ADAS_01	CF08B00h	8	Application	Cyclic	50 ms	s	-	r	-
BRAKE_01	0F04C01h	8	Application	Cyclic	10 ms	r	-	s	-
DIAG_RQ	10DA4CFFh	8	Diagnostic	Spontaneous	-	r	r	r	s

s = send, **r** = receive

Table 2.2: A Representative Design of a Communication Matrix on One Network

2.4.2 Frame

In the design of a communication matrix, a frame serves as a container for PDUs. The data field length of a CAN frame is restricted to 8 bytes according to the ISO 11898-1 standard [28, 27].

2.4.3 Protocol Data Unit (PDU)

PDUs encapsulate communication signals. OEMs may define multiple PDUs in a frame -if the communication protocol allows- and multiple signals in a PDU.

In the context of classical CAN, each CAN frame is limited to containing a single PDU [32]. A representative CAN frame, a PDU and signals with their hierarchical layout are provided in the Figure 2.8.

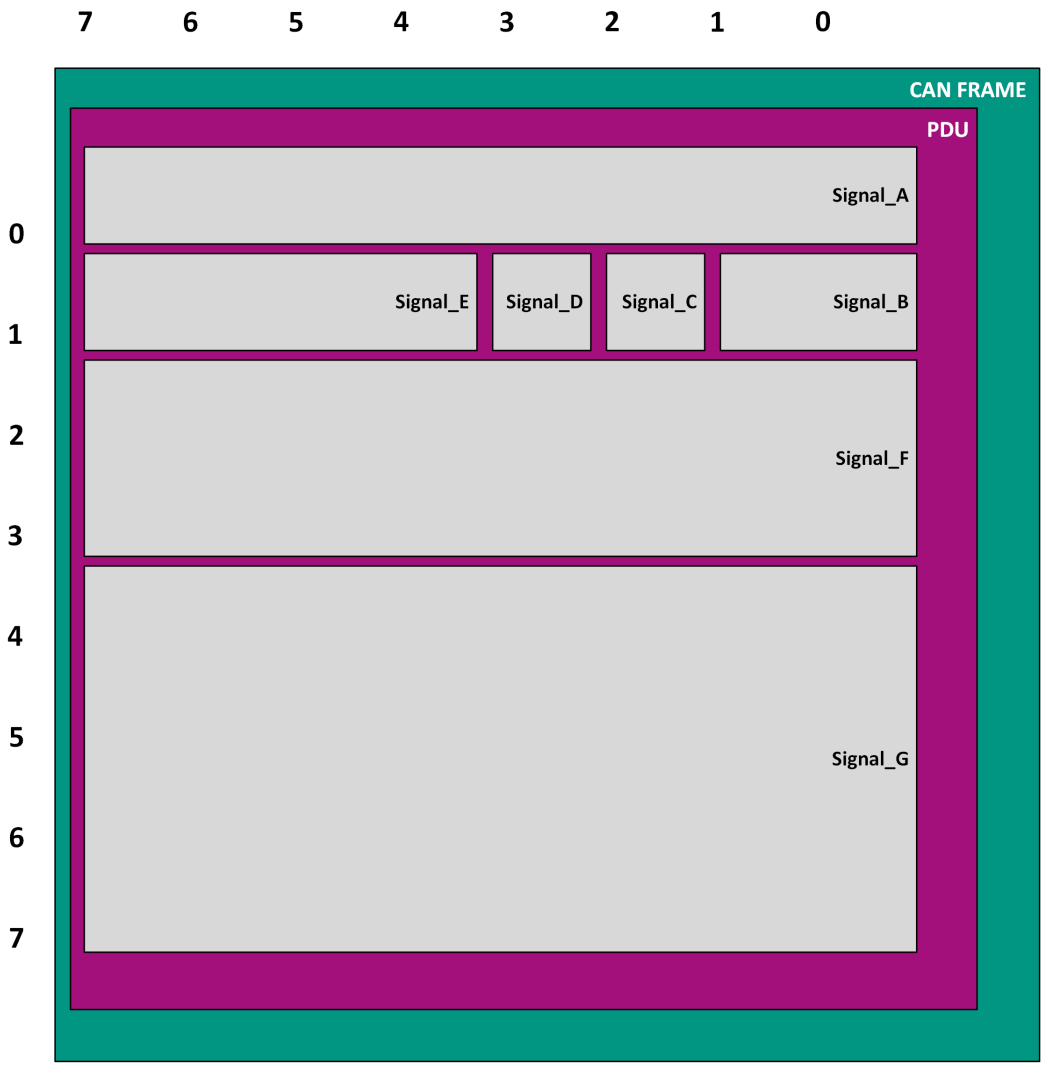

Figure 2.8: Representation of a Data Field of a CAN Frame

In the OSI Model [29], the CAN protocol [32] and the layered AUTOSAR Architecture [28], a PDU consists of two components:

1. Service Data Unit (SDU)
2. Protocol Control Information (PCI).

The SDU represents the actual payload transferred by the upper layer, while PCI contains essential information required for transferring the SDU between layers. The process of transferring the SDU across layers is depicted in the Figure 2.9

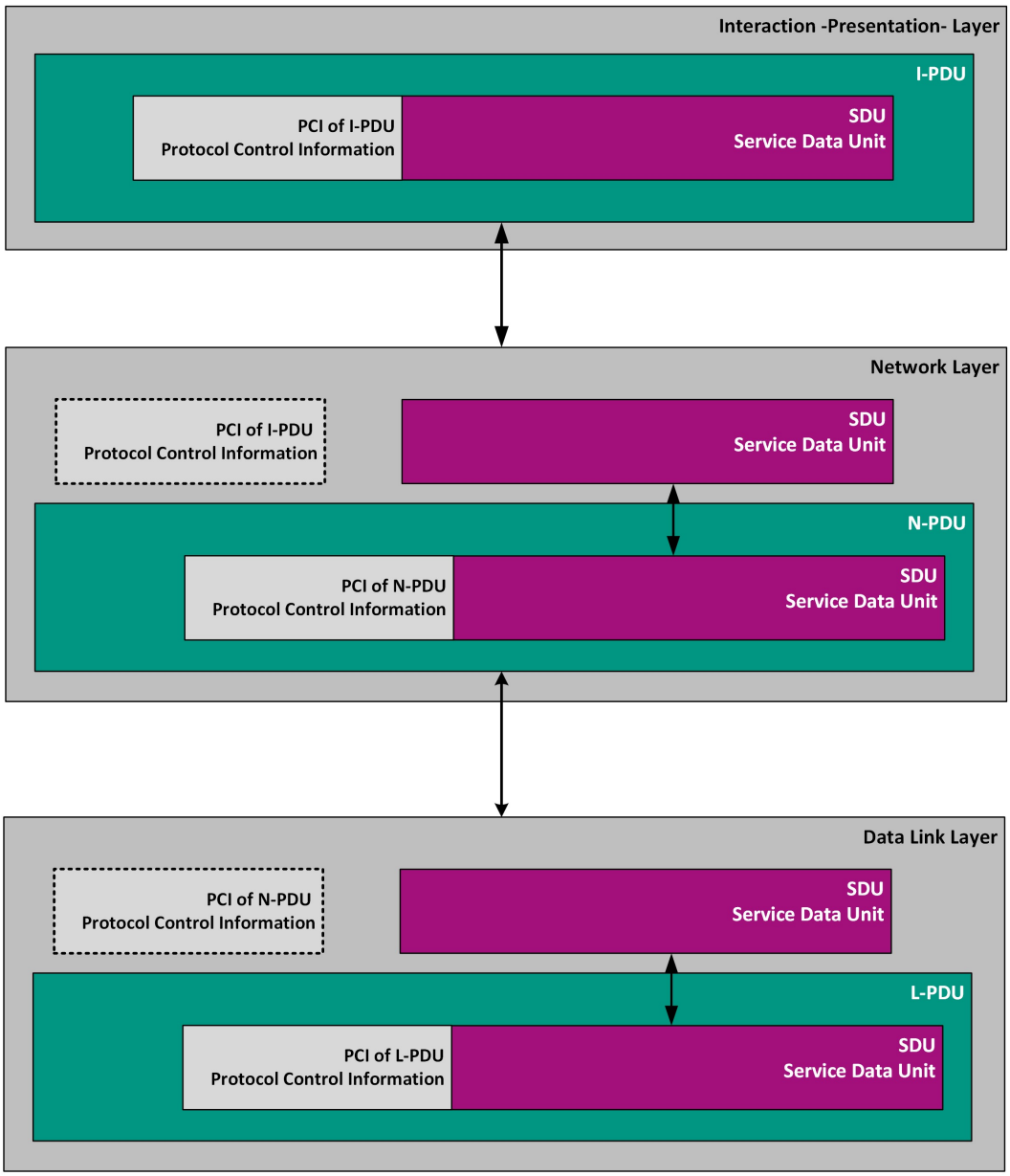

Figure 2.9: Illustration of PDU, SDU and PCI [32, 29]

In the AUTOSAR Layered architecture, PDUs are assigned specific prefixes that reflect the OSI layer responsible for their handling. These prefixes are:

- Interaction -Presentation- Layer PDU (I-PDU)
- Network Layer PDU (N-PDU)
- Data Link Layer PDU (L-PDU)

The L-PDU corresponds to the CAN frame mentioned in the Section 2.3.4. A comprehensive depiction of OSI layers along with their associated PDUs is represented later in this thesis in Figures 2.6 and 2.12.

2.4.4 Signal

A signal is the smallest unit that is used in the communication to transfer information. They are encapsulated within PDUs and are characterized by various attributes in the communication matrix. A representative design of signals inside a PDU is provided in the Table 2.3.

Level	Object Name	Signal Length Bit(s)	Data Type	Signal Bit Start Position
Frame	ADAS_01			
PDU	ADAS_01_PDU			
Signal	Signal_A	8	SCALED_08_DEG	0
Signal	Signal_B	2	ENUM_02_STAT	8
Signal	Signal_C	1	BOOLEAN	10
Signal	Signal_D	1	BOOLEAN	11
Signal	Signal_E	4	ENUM_04_MODE	12
Signal	Signal_F	16	SCALED_16_YAW	16
Signal	Signal_G	32	SCALED_32_POS	32

Table 2.3: An Example Design of Signals in a PDU

2.5 Automotive Open System Architecture (AUTOSAR)

This section aims to explain fundamentals of AUTOSAR and important pillars of it in the context of gatewaying.

2.5.1 AUTOSAR Partnership and Releases

AUTOSAR is a partnership founded in 2003 that brings automotive OEMs, suppliers, service and tool providers together in order to standardize the software development methodology for an open E/E architecture [19, 33].

Thanks to its layered architecture (refer to chapter 2.5.5), AUTOSAR provides the possibility to decouple the development of the differentiator application software from

the hardware. With the usage of AUTOSAR standardization in the ECU development process, significant improvements are expected to be yielded [34]. Through AUTOSAR, standardization of non-competitive software, re-usability and reduction of overall software development costs are outcomes anticipated by OEMs. While these benefits are expected and mentioned by the partnership, their realization in practice may depend on various factors and remains subject to an evaluation.

The partnership standardizes the AUTOSAR Classic Platform and AUTOSAR Adaptive Platform and releases their specification documents annually along with the Foundation standards in order to ensure the interoperability between the two platforms. Adaptive Platform is being developed to address use cases that require high-performance computing. Its implementations are based on microprocessors using Portable Operating System Interface (POSIX) operating systems [35]. The relevant AUTOSAR platform for this thesis is the AUTOSAR Classic platform, where the near hard real-time requirements and safety constraints for automotive embedded ECUs are aimed to be met [36]. Implementations of this platform are typically carried out on microcontrollers.

As the partnership releases new versions of its standards, tool and stack providers typically align their development strategies to support these updates. However, the adoption of a new AUTOSAR release often requires several years to be seen in vehicles on the road. Currently, the majority of ECU development in the automotive industry is based on AUTOSAR stacks version R4.3.0 or later. A common industry practice is for stack providers to deliver OEMs and Tier 1 suppliers their most recent AUTOSAR-compliant stack available at the start of a development cycle, with subsequent updates integrated as needed throughout the project. The selection of stack features and AUTOSAR versions is to a great extent influenced by the Extensible Markup Language (XML) schema version specified by the OEM. A more detailed discussion of the XML schema versions will be presented in the chapter 2.5.2.

In this thesis two key AUTOSAR release versions are referenced: the Classic Platform Release R4.4.0 and latest Classic Platform Release R24-11. While R4.4.0 is widely used in the automotive industry, R24-11 introduces new features such as Link Layer SDU (L-SDU) Router which is relevant to thesis topic.

2.5.2 AUTOSAR Exchange Format

With the software-defined vehicle trend, OEMs seek to expand their in-house software development capabilities. However, at present, the predominant portion of software continues to be developed in collaboration with suppliers [37] .

AUTOSAR specifies a standardized exchange format in order to facilitate and streamline this collaboration. In AUTOSAR Classic Platform, the content of the exchange format is primarily specified by the system template and software component template documents, both created and maintained by the AUTOSAR Working Group Methodology and Templates (WG-MT). These documents build upon the AUTOSAR meta-model which is

written using the Unified Modeling Language (UML)2.0 language. The chosen exchange format in AUTOSAR is XML, with its production rules also standardized by the partnership and the specific file name extension used is .arxml (AUTOSAR XML (ARXML)), which will be referred to as ARXML and ECU Extract throughout this thesis [32, 38, 39]. The standardized AUTOSAR facilitates the interoperability across various development tools [40].

Each AUTOSAR release is accompanied by a corresponding XML schema version for ARXML. As outlined in the chapter 2.5.1, this thesis focuses on AUTOSAR Releases R4.4.0 and R24-11, which are associated with the XML schema versions AUTOSAR_00046 and AUTOSAR_00053 respectively [35].

2.5.3 AUTOSAR Methodology and ARXML from OEMs' Perspective

In the automotive software development process, suppliers may be responsible for one or multiple ECUs, while OEMs are responsible for the overall system. Consequently, prior to the generation of an ECU Extract, OEMs shall complete several pre-requisite tasks related to the system level design. These tasks include, but are not limited to, the following :

1. Creating E/E Architecture
2. Assigning ECUs to Specific In-Vehicle Networks
3. Distributing Functions into ECUs
4. Creating Communication Design of ECUs in a Database
5. Creating Software Component Design of an ECU in a Database

The sequence or the existence of the above tasks may vary depending on the ECU being developed and the strategic approach adopted by the OEM.

In scenarios where the ECU's functionality does not serve as a differentiating factor (e.g. gateway ECUs) or when OEMs do not possess sufficient in-house expertise for a specific functionality (e.g. steering angle sensor ECUs), OEMs may opt not to define the software component architecture in terms of atomic software components and their internal behaviors. Instead, OEMs might generate a top level software compositions that are structurally defined but do not include any internal content. These empty software compositions are called root software compositions and specify all provided and required ports, along with their communication attributes (e.g. data element reference, init value). ECU Extracts of this nature are being generated with the system category of SYSTEM_EXTRACT [32].

The AUTOSAR XML schema is very detailed and has numerous constraints, therefore, manually creating ARXML files can be a cumbersome and error prone process, even for relatively simple ECU projects. Hence, the usage of a tool that might partially automate

the repetitive tasks and provides built-in consistency checks is generally preferred in ARXML generation.

For the creation of ECU Extracts, there are several well established tools in the automotive industry. Among these are the PREEvision -provided by Vector Informatik GmbH- [41] and ARTOP -accessible exclusively to ARXML partners- [42]. In addition, there are proprietary tools such as XDIS -a tool used by Daimler Truck AG and Mercedes-Benz AG- [43, 5]. These tools possess varying strengths and differ in their approaches to communication and software component design. However, a comparison of their advantages falls outside the scope of this thesis. While the tools mentioned represent some of the solutions, it should be noted that additional tools exist beyond those discussed in this thesis.

A legacy practice that sometimes being followed in the AUTOSAR ECU development process involves the usage of non-AUTOSAR exchange formats such as Database for CAN (DBC) files. However, relying on such formats limits the ability to fully leverage the capabilities and standardization benefits offered by AUTOSAR methodology.

In this thesis, the ARXML files which will be used later in the implementation phase will be generated by the tool XDIS [43, 5].

An exemplary development process of an AUTOSAR ECU in a collaboration with a Tier 1 supplier is being shown in the Figure 2.10. The tasks assigned to the OEM in this figure are outlined with the assumption that the process begins at the initial phase of the project, commonly referred to as ECU creation. However, in the automotive industry, most ECUs continue to receive updates even after the Start of Production (SOP). During such post-SOP update cycles, updating communication and software component is still necessary, but the first three tasks originally designated for the OEM at project initiation may no longer be required and can be omitted.

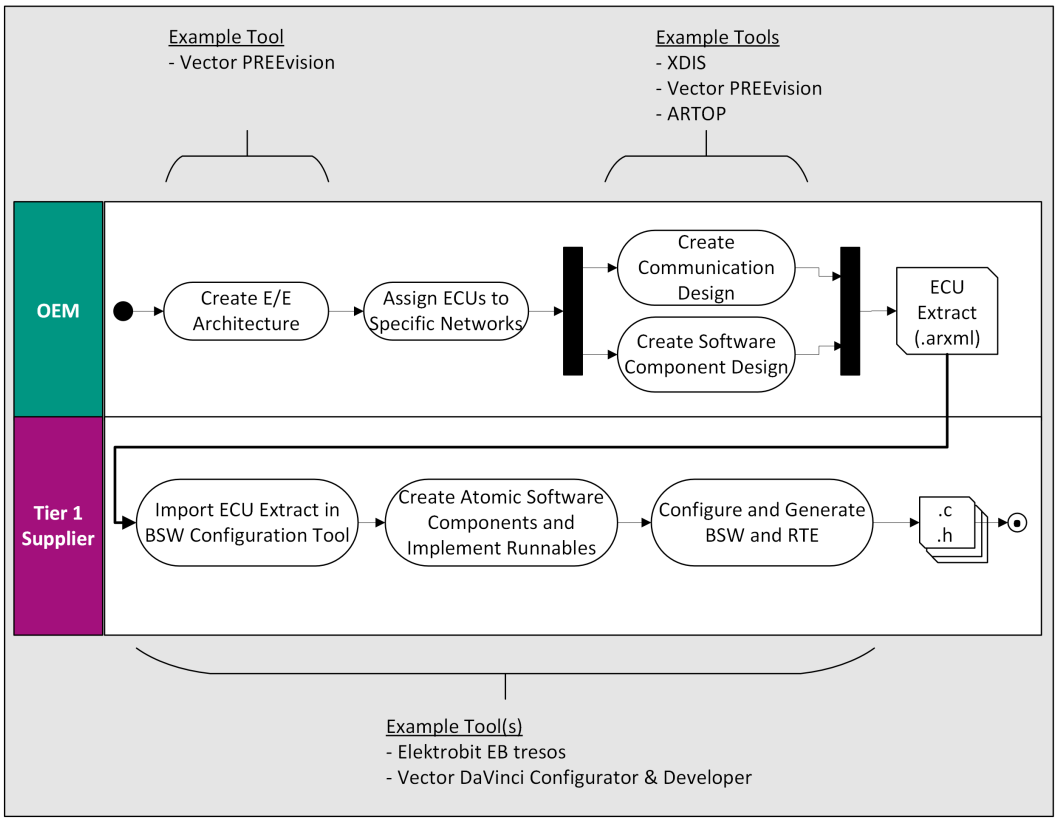

Figure 2.10: Development Process of an AUTOSAR ECU with a Tier 1 Supplier

2.5.4 AUTOSAR Methodology from Tier 1 Supplier's Perspective

The AUTOSAR Basic Software (BSW) stack constitutes the real-time embedded software [44] that provides services to application software components [45]. It comprises a range of standardized modules, inter alia, Network Management (NM) module, diagnostic module, Operating System (OS) module, or End to End (E2E)-Protection module [46]. These BSW modules facilitate communication, timing, diagnostics, safety and many other fundamental functions for the ECU. Typically, the BSW stack is procured from a third-party stack provider such as Vector Informatik, Elektrobit, ETAS.

OEMs might follow different strategies regarding the procurement and provision of BSW stacks, at times these strategies may form a partnership with a BSW stack provider. In that case, the OEM may have licenses and rights to distribute BSW stacks to their Tier 1 suppliers. However, for the purposes of this thesis, scenarios are considered in which the Tier 1 supplier independently procures the BSW stack. In this context, the supplier selects an appropriate stack provider and chooses required BSW modules based on the specific requirement of the ECU project.

BSW stack providers usually offer dedicated BSW configuration toolchains along with their BSW stack. Consequently, the selection of the configuration tool is highly linked to the choosen BSW stack. Commonly used tools include Vector DaVinci Configurator [47] and Elektrobit EB Tresos [48], among others.

Following the delivery of the ECU Extract, the supplier imports it into BSW configuration tool. The tool automatically derives a significant portion of the BSW parameters from the ECU Extract directly such as, bus timings, frame lengths, among other parameters. However, some parameters, especially those dependent on the hardware or OS specific should be manually configured by the BSW integrator for instance baudrate prescaler, number of cores and OS tasks. Since ECU Extracts are mostly generated without having a prior knowledge about the hardware or the OS in this type of collaboration, these inputs should be given by the supplier.

As depicted in the Figure 2.10, the supplier will develop or integrate its own atomic software components to provide the intended ECU functionality. These atomic components may then be used to define one or multiple runnable entities [49] with their internal behaviors. These runnable entities are the functions which contain the real code implementation, i.e. C or Rust code. This software component development or integration can be carried out in tools like Vector DaVinci Developer or ETAS ASCET [47].

The modular approach regarding software component integration highlights one key advantage of AUTOSAR for Tier 1 suppliers. For instance, a Tier 1 supplier developing a steering angle sensor may deliver its solution to multiple OEMs with minimal modification to the application software component design, since the application software design is decoupled and independent from the hardware. In such cases, previously developed atomic software components can be reused across projects, eliminating the need to create them from scratch.

2.5.5 AUTOSAR Layered Architecture

This subsection aims to summarize the AUTOSAR layered architecture and mostly refers to the AUTOSAR Layered Software Architecture document [36].

AUTOSAR Architecture is mainly being split into three software layers:

1. Application Layer
2. Runtime Environment (Rte)
3. BSW

The BSW is also further divided into four layers:

- Services Layer
- ECU Abstraction Layer

- Microcontoller Abstraction Layer (MCAL)
- Complex Device Driver (CDD)

Figure 2.11 illustrates layered architecture of AUTOSAR, with particular emphasis on the communication functional group.

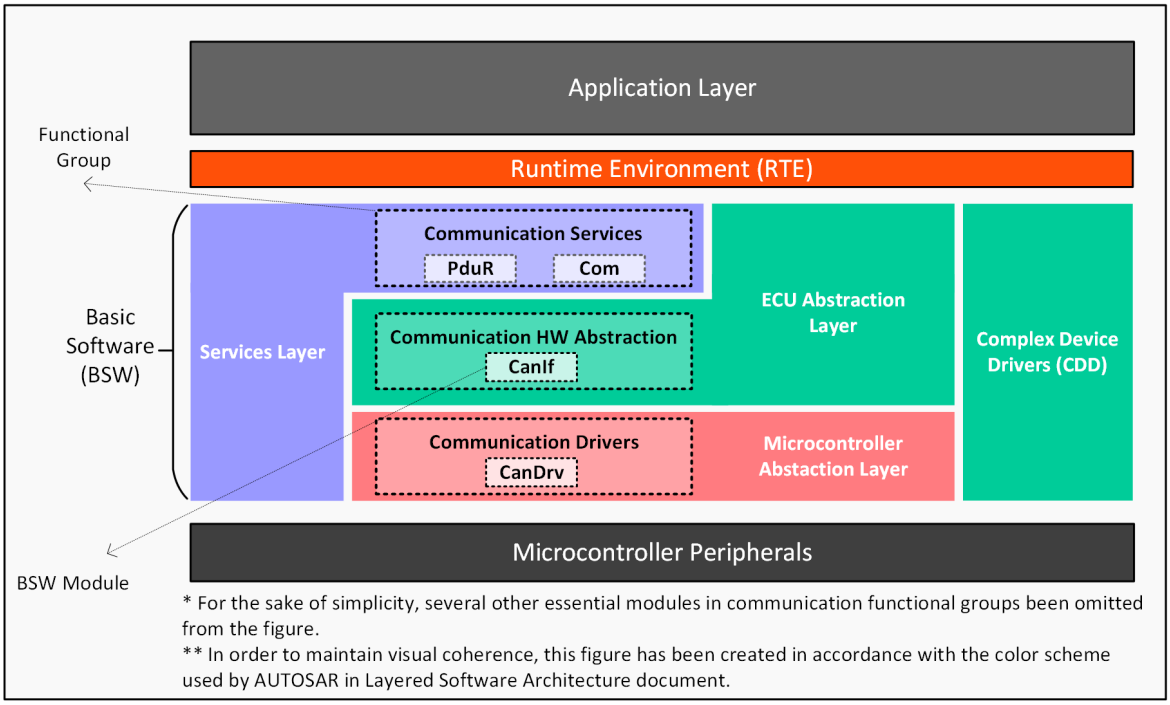

Figure 2.11: AUTOSAR Layered Architecture [32]

CDD aims to implement enhanced services, protocols or maintains legacy functionality (e.g. Offene Systeme und deren Schnittstellen für die Elektronik in Kraftfahrzeugen (OSEK) NM) which are not standarized in AUTOSAR [36], therefore goes beyond the scope of this thesis. The remaining three layers are in detail divided into functional groups, which are further subdivided into modules, commonly referred to as BSW modules. The important functional group for this thesis would be related to communication:

- Communication Services
- Communication Hardware Abstraction
- Communication Drivers

In the subsection 2.5.6 a detailed explanation of the purpose of the relevant BSW modules and their interactions is provided.

2.5.6 An Example of CAN Communication with AUTOSAR Layered Architecture

Although the layered architecture remains largely similar between AUTOSAR release versions 24-11 and earlier versions (<24-11), a notable addition in version 24-11 is the introduction of the L-SDU router.

The operational flow of AUTOSAR - from the application software layer to the point at which a CAN frame appears on the bus- is illustrated in Figures 2.12 and 2.6.

In this section, the working principles of AUTOSAR flow will be explained using an example, with reference to the AUTOSAR version 24-11 and the Figure 2.12.

In this subsection, the ECU under consideration corresponds to the configuration shown in the Figure 2.12 : an ECU equipped with a microcontroller, five CAN controllers (four integrated and one external), and five CAN transceivers. As a scenario, the transmission of a CAN frame on the bus will be carried out using the leftmost CAN controller and its associated transceiver.

The interactions between AUTOSAR modules, as illustrated in Figure 2.12, are bidirectional.

In this scenario, it is assumed that Application Software Component (SWC) - 3 is responsible for processing the steering angle value obtained from a sensor. The development of the atomic software component corresponding to Application SWC-3, including its runnables and the logic for calculating the steering angle, falls under the responsibility of the Tier 1 as illustrated in the Figure 2.5.4 in Section 2.5.4.

According to our assumed collaboration model in this thesis, the communication signal, its associated data type, the PDU and CAN frame design along with other configuration details are provided by the OEM in the ECU Extract.

In the described scenario, it is assumed that the runnable of the steering angle sensor is a cyclic runnable executed every ten miliseconds. Similarly, the transmission trigger defined for the corresponding signal, PDU and CAN frame is set to 10 ms. This implies that the Application SWC-3 calculates and writes the signal value for steering angle using the Rte_Write() Application Programming Interface (API) every 10 ms. Then, the AUTOSAR Rte handles the conversion from the application data type to the implementation data type and forwards the signal to the Communication (COM) Module.

The COM subsequently stores it in an internal buffer. Based on the defined I-SIGNAL-TO-I-PDU-MAPPING in the ECU Extract - consequently reflected in the module configuration - the COM maps the signal into the appropriate I-PDU. The COM then invokes the PduR_ComTransmit() API to pass the I-PDU to the PDU Router (PduR).

The PduR is responsible for determining the destination modules for each I-PDU. In this case, it calls the LSduR_Transmit() API to forward the I-PDU to the newly

introduced L-SDU Router module. The routing paths for both PDU and the L-SDU Router are statically defined in the configuration based on parameters derived from the ECU Extract.

The L-SDU Router converts the I-PDU into an L-PDU (refer to Figure 2.9) and subsequently calls the CanIf_Transmit() API. The CAN Interface module then encapsulates the L-PDU into the appropriate CAN frame and invokes the Can_Write() API to forward the L-PDU to the CAN Driver.

If the CAN hardware available, the CAN Driver copies data into the CAN hardware object and sets the transmit request flag within the CAN controller. The CAN transceiver then converts the digital data coming from the CAN controller into the physical signals, which are transmitted over the CAN bus via the CAN-H and CAN-L cables.

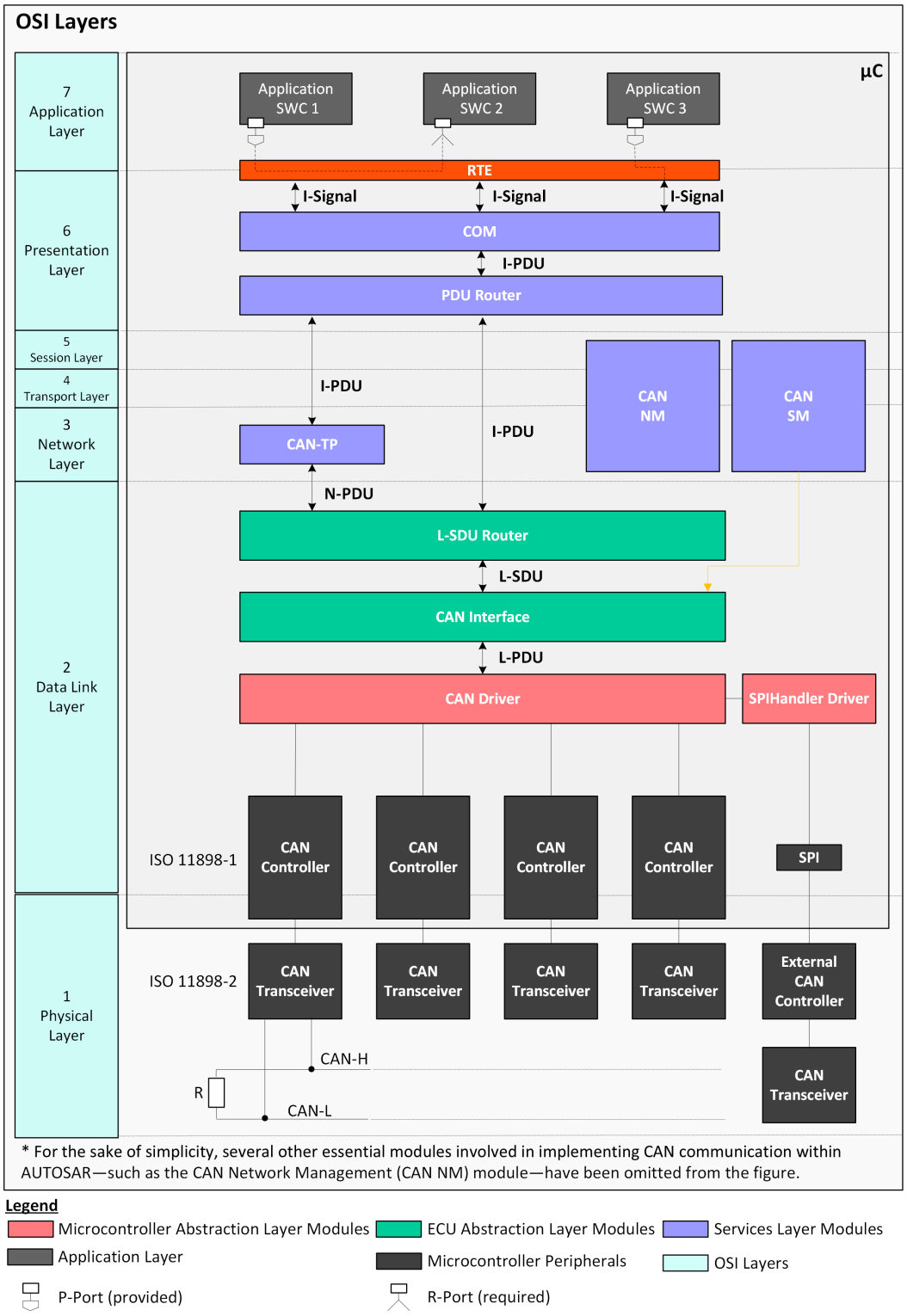

Figure 2.12: CAN Communication with Layered Architecture acc. to AUTOSAR 24-11 [32]

2.6 Gatewaying

This section aims to explain fundamentals of gatewaying in the context of automotive in-vehicle networks.

2.6.1 Terminology: Gateway vs. Router

In principle, a gateway refers to a device that connects different network technologies (e.g. CAN to Automotive Ethernet) and performs protocol translation between them. In contrast, the term router typically denotes a device that forwards data between networks adopting the same communication protocol (e.g. Automotive Ethernet to Automotive Ethernet). [2]

However, in practice, both gatewaying and routing functionalities are often integrated together into a single unit [2], which commonly results into a Gateway ECU.

As outlined in Section 2.3, this thesis focuses on a single in-vehicle network communication protocol: CAN. From a theoretical standpoint, this would imply the use of the term router rather than a gateway, since no protocol translation is required between CAN networks. Nonetheless, the CAN protocol exists in multiple generations (refer to Section 2.3.2), and a Gateway ECU is expected to handle communication between these different CAN protocols. Therefore, even though the analyzed and proposed implementations in this thesis primarily forwards data between CAN networks, they will be referred to as a Gateway throughout this thesis.

2.6.2 Gatewaying in AUTOSAR

This section provides an overview of the gatewaying mechanisms within the AUTOSAR standard.

In today's vehicles, AUTOSAR is the predominant software architecture for ECUs, including Gateway ECUs with microcontroller based implementations [2].

The main gatewaying mechanism provided by AUTOSAR is known as the PDU based Gateway [32] which is within the PDU Router module of the AUTOSAR BSW [50, 51]. With PDU based Gateways, the payload stays intact but the PDU ID used in different communication interfaces are different. Different PDU IDs are required to ensure Tx_Confirmation mechanisms in the AUTOSAR.

The Signal Gateway refers to an implementation in which routed signals inside a routed PDU (from source) can be placed into different PDUs in the destination. In AUTOSAR, such functionality can be achieved using AUTOSAR COM Module, as the PDU Router does not support this mechanism. As a result, configuring signal routings requires

additional effort during integration and introduces increased runtime load and latencies additional efforts during the runtime.

In this thesis, signal gatewaying mechanism are excluded from the conceptual work and implementation, as they constitute only a small percentage of overall routings in a vehicle, based on the research results given in the Chapter 4. Accordingly, throughout this thesis, the term AUTOSAR Gateway refers exclusively to the PDU based Gateway.

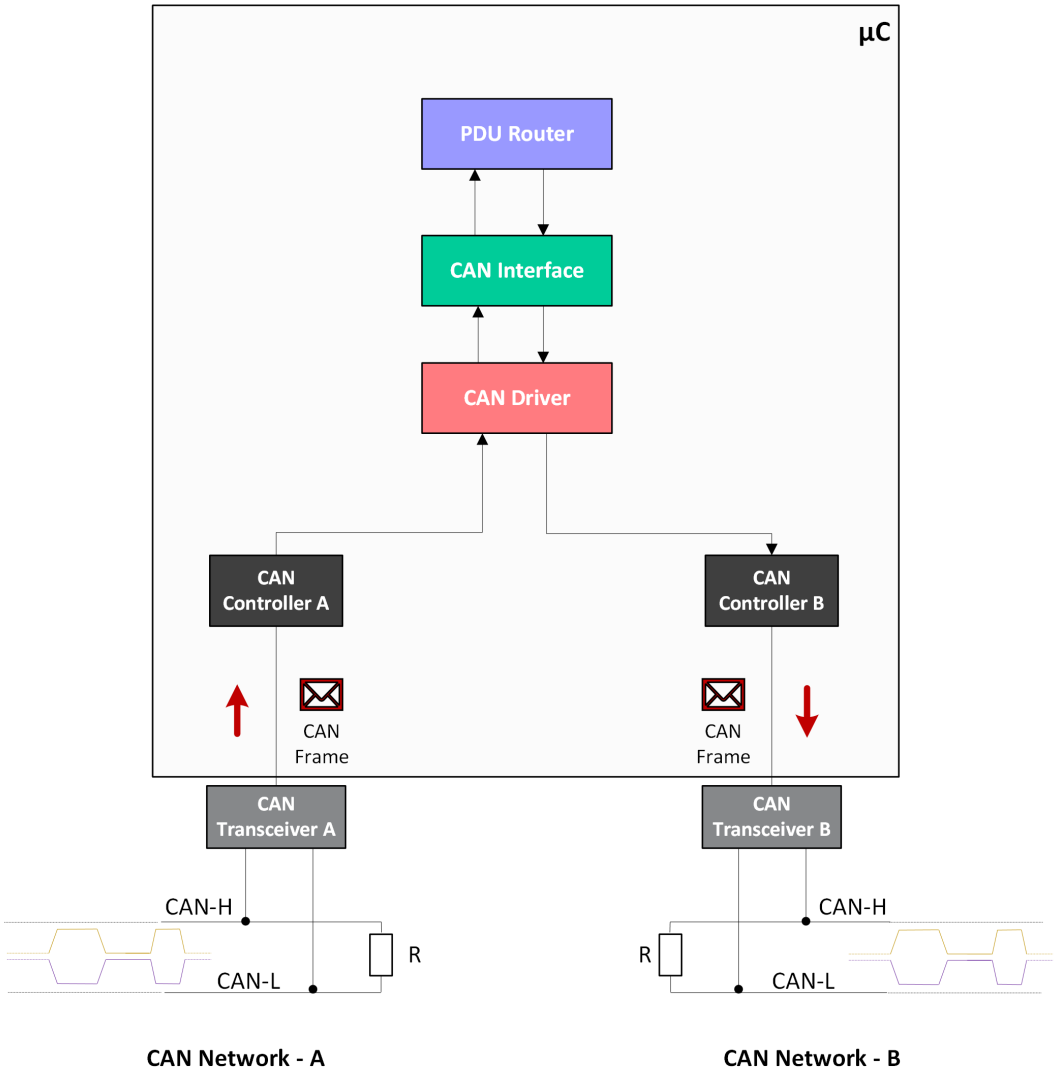

Figure 2.13: PDU Routing in AUTOSAR versions <24-11

Figure 2.13 illustrates the flow of a PDU routing for AUTOSAR versions prior to the release 24-11.

PDU Router Module is the highest level BSW module required in PDU Routings and consist of the PDU Router routing paths and the PDU Router Engine [50, 51].

The routing paths of PDUs, including both their source and destination, are defined statically [50, 51]. Consequently, I-PDUs are routed dynamically at the runtime.

The static routings configurations between communication interfaces of an ECU are usually specified in the ECU Extract by the OEM and can be derived into the PDU Router module upon importing the ECU Extract into the BSW Configuration tool. The ECU Extract and the Methodology has been discussed in the Subsection 2.5.3 in detail.

The PDU Router Engine is responsible for executing the actual routing from source to the destination. With the PDU Router, the data payload stays intact [50, 51].

As illustrated in the Figure 2.13, the PDU Router represents the highest level BSW Module in gatewaying,interfacing with lower level communication modules.

I-PDUs can be routed from a source communication multiple communication interfaces (1:n), such as CAN Interface (CanIf), Ethernet Interface (EthIf), LIN Interface (LinIf), FlexRayInterface. Although, the PDU Router supports gatewaying between different network technology interfaces, the focus of this thesis is the CAN Protocol. Therefore, in this thesis only the gatewaying between two CanIfs will be considered further.

The sequence diagram of a PDU Routing between two CAN networks is given in the Figure 2.14.

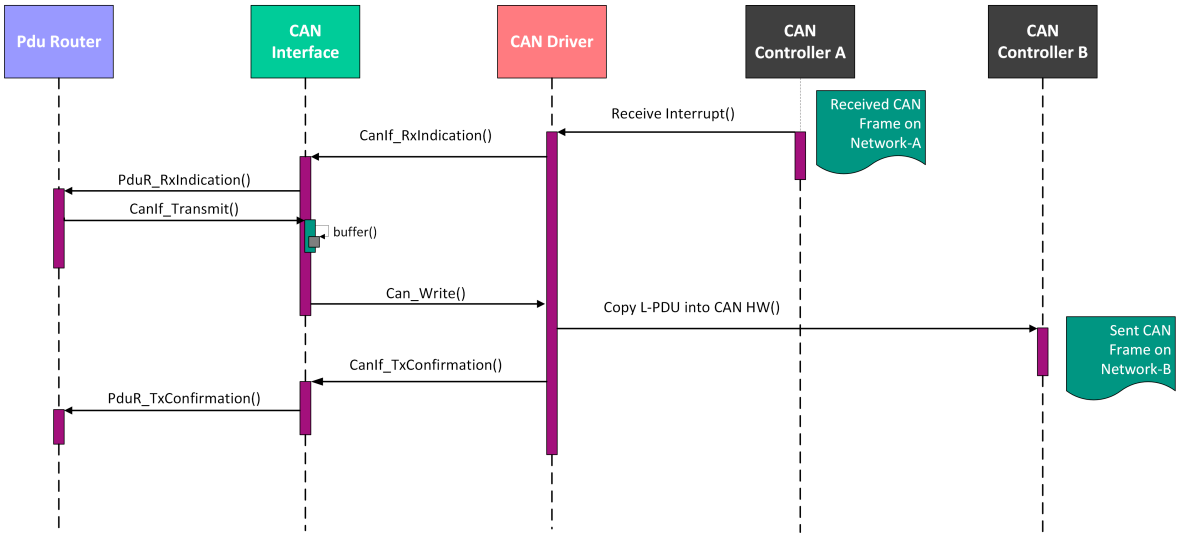

Figure 2.14: Sequence Diagram of the Routing between CAN-Network-A and Network-B

In the AUTOSAR release 24-11, the routing flow has been updated to include newly introduced L-SDU Router module. This revised routing flow is depicted in the Figure 2.15.

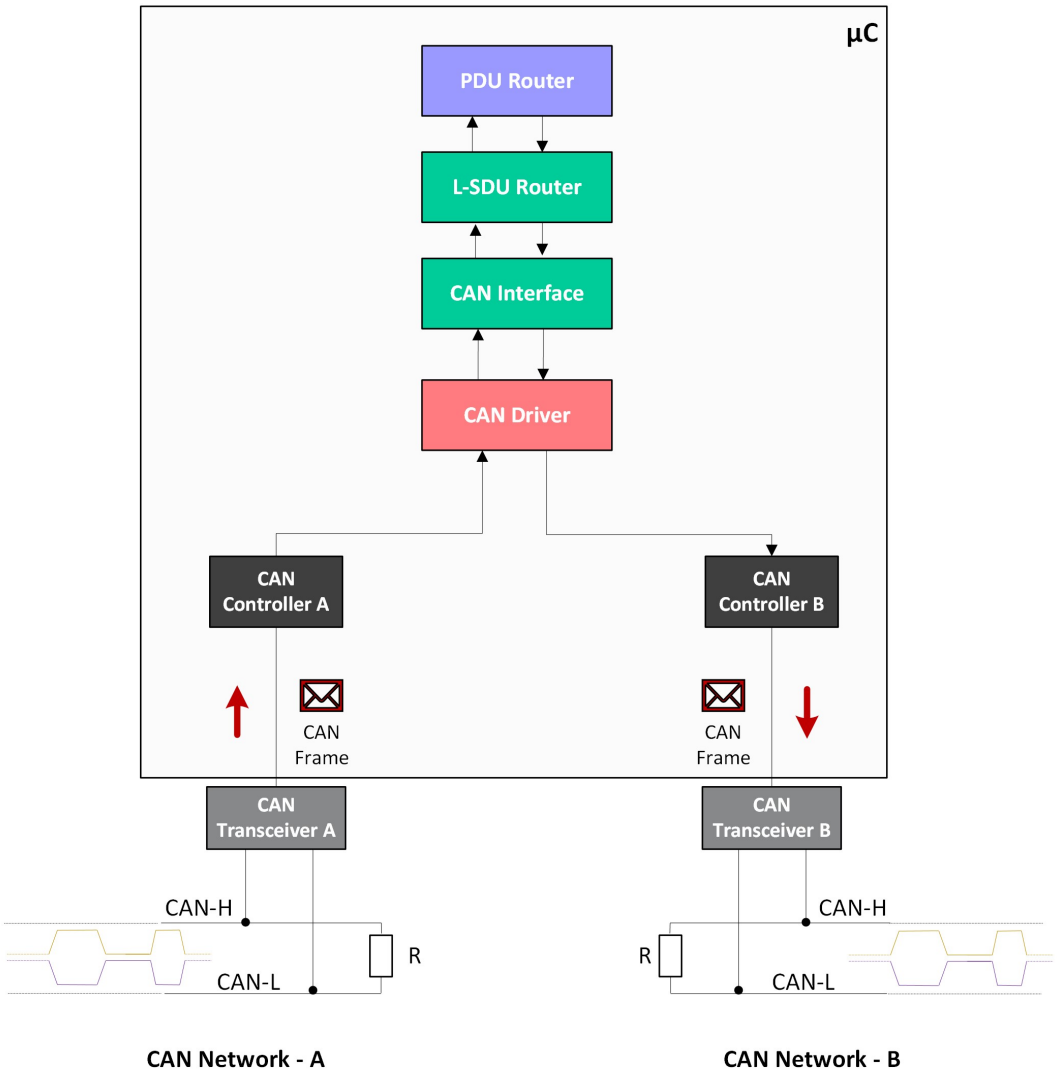

Figure 2.15: PDU Routing in AUTOSAR versions = 24-11

The L-SDU Router module was introduced in AUTOSAR release R24-11 [32] and has become the mandatory upper layer for all communication interfaces [52].

The main motivation behind introducing the L-SDU Router module is to enable Institute of Electrical and Electronics Engineers (IEEE) 1722 use cases across all communication interfaces, and facilitate data exchange from non-Ethernet communication interfaces to Ethernet based ones [52].

As the L-SDU Router module is an recent addition, there is currently no pbulished research on this topic, nor is the funtionality yet available as a BSW module from BSW vendors. Conseuquently, its impact on the standard PDU Routing remains subject to a further evaluation.

In this thesis, the L-SDU Router module is described and included in figures to illustrate the state-of-the-art AUTOSAR Layered Software Architecture. However, this module will not be used in the implementation of the AUTOSAR Gateway developed as part of this thesis.

2.6.3 Gatewaying with Hardware Acceleration

Recently, most silicon vendors have introduced communication hardware acceleration solutions for gatewaying [8, 9, 53, 54]. These solutions aim to offload the CPU by performing low level routing in hardware with a low latency.

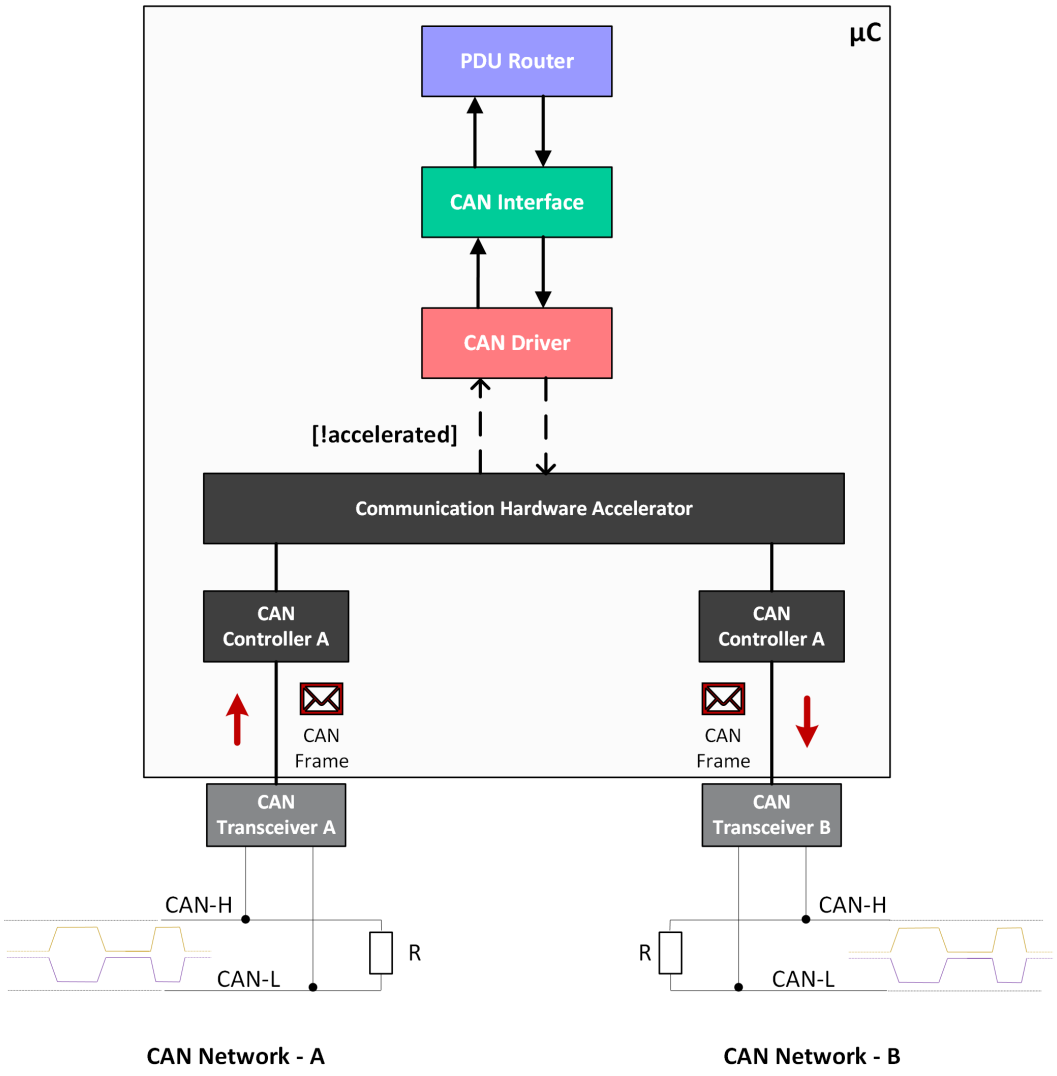

Figure 2.16: Routing with Hardware Acceleration [9, 53]

Communication hardware acceleration solutions are proprietary and lack standardization [55, 9, 54]. The specific implementations and mechanisms used differ among hardware vendors. Nonetheless, as a common practice (similar to AUTOSAR PDU Router), these solutions retain statically defined routing configurations and do not support dynamic routing mechanisms.

As illustrated in the Figure 2.16, while internal mechanisms of communication hardware accelerators may vary, they generally process incoming CAN frames within their hardware itself. This approach reduces both CPU and interrupt load compared to the traditional AUTOSAR PDU Routing methodology. Frames are only forwarded to upper layers (CAN Driver) if they are not configured to be routed with hardware accelerators.

Due to the absence of the standardized hardware acceleration mechanisms, integrating AUTOSAR Classic Gateway solutions with proprieatry hardware acceleration from different vendors also remains a non-standardized practice. Despite these challanges, communication hardware accelerated Gateway solutions have demonstrated promising performance in proof-of-concept studies[54].

Given these promising results and the increasing availability of this feature in next generation microcontrollers and BSW Stacks, communication hardware acceleration solution may represent a viable approach for HPCs. However, they are generally considered costly, inflexible and complex in this thesis, making them less suitable for Zonal Gateways, which might face specific challenges as outlined in the Section 1.1.

This thesis explores an alternative hardware centric gateway architecture and does not consider existing communication hardware acceleration solutions in its scope.

2.7 CAN Bit Rate and Bus Timing

CAN is an asynchronous communication protocol that does not synchronize the clocks of the CAN Nodes on a network [56]. To achieve the desired bit rate and minimize the clock drift and its effects, each CAN node shall implement a bit timing logic as per ISO 11898-1 which synchronies all CAN Nodes [28, 27].

According to the ISO 11898-1 [28], the nominal bit time is divided into four different segments. In the Figure 2.17 these four segments are depicted.

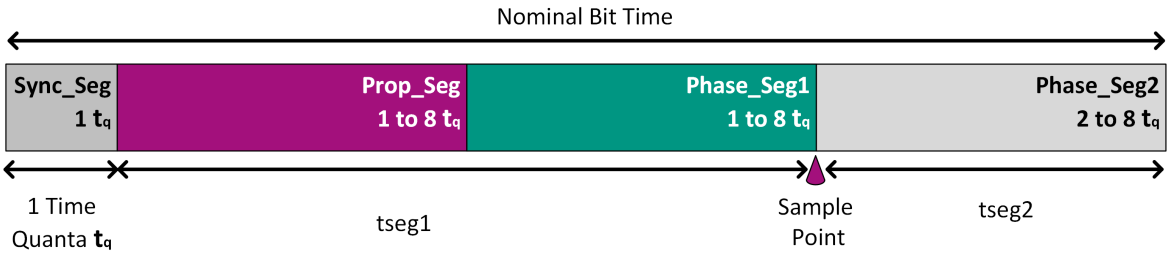

Figure 2.17: CAN Bit Time Segments [28, 27, 57, 58]

The Sync_Seg is the nominal start of each bit [57] synchronization segment which is used to synchronize the CAN Nodes [28]. The Prop_Seg is the propagation time segment which is used to compensate for physical delay times which is consisting of the signal propagation time on the bus and internal delay time of CAN Nodes [28]. In other words, Prop_Seg is being used to delay the earliest possible sample of the bit by a node until the transmitted bit values from all transmitting nodes have reached all of the nodes in order to keep the arbitration mechanism sage [57]. The Phase_Seg1 and Phase_Seg2 are buffer segments to compensate edge phase errors.

There are two different synchronization defined in the ISO 11898-1 [28]:

1. Hard Synchronization : It is performed only at the beginning of each CAN frame once every CAN Node aligns the Sync_Seg of its current bit time to the recessive to dominant edge of the transmitted SOF bit.
2. Resynchronization: It is performed during the remainder of the CAN frame whenever a change of bit value from recessive to dominant occurs [57].

The time quantum or also called time quanta (t_{q}) is the fixed unit of time derived from the CAN node clock period. The CAN Node clock frequency (f_{CAN}) is dependent on the CAN clock frequency ($f_{\mathrm{SystemCAN}}$) and Baudrate Prescaler (BRP) and can be calculated as the following:

$$f_{\mathrm{CAN}} = \frac{f_{\mathrm{SystemCAN}}}{BRP} \tag{2.1}$$

Also the equation 2.2 is being deduced from the Figure 2.17 which shows the total period for Nominal Bit Time (NBT).

$$t_{\mathrm{NBT}} = t_{\mathrm{Sync\_Seg}} + t_{\mathrm{Prop\_Seg}} + t_{\mathrm{Phase\_Seg1}} + t_{\mathrm{Phase\_Seg2}} \tag{2.2}$$

As per the definition from the ISO 11898-1[28] the bit timings shall be calculating considering following constraints:

$$8 \leq t_{\mathrm{NBT}} \leq 25 \tag{2.3}$$

$$1 \leq BRP \leq 32 \tag{2.4}$$

Some other constraints are coming from the ECU Extract due to OEM design and requirements and can be depicted as followings:

$$75\% \leq \text{Sample Point} \leq 81.25\% \tag{2.5}$$

$$10 \leq t_{\text{NBT}} \leq 20 \tag{2.6}$$

In the CAN protocol, since no additional modulation is applied and each symbol corresponds directly to a single bit, the bit time and symbol time are effectively equivalent. As a result, the terms bit rate and baud rate can be used interchangeably. For consistency, the term bit rate will be used throughout this thesis, with the understanding that it also refers to the baud rate.

The chosen bit rate for the CAN bus implementation in this thesis is 500 kbit/s, as this value is commonly used not only in Classical CAN, but also serves as arbitration phase bit rate in other CAN generations. The system clock for classical CAN used in the CAN IPs is usually up to 24 MHz [59], and therefore this value will be taken as a reference. From these values and considering the equation 2.1 we can derive:

$$\text{Bit Time} = \frac{1}{500\,\text{kbit/s}} = 2\,\mu\text{s} \tag{2.7}$$

$$t_{\text{q}} = \frac{1}{f_{\text{CAN}}} \tag{2.8}$$

$$t_{\text{NBT}} = \frac{\text{Bit Time}}{t_{\text{q}}} \tag{2.9}$$

If we use the value found from the equation 2.7 with the equation 2.9 we can derive a new definition:

$$t_{\text{NBT}} = \frac{2\,\mu\text{s}}{\frac{1}{f_{\text{CAN}}}} = 2\,\mu\text{s} \times f_{\text{CAN}} \tag{2.10}$$

If we combine equation 2.10 and the constraint 2.6:

$$10 \leq 2\,\mu\text{s} \times f_{\text{CAN}} \leq 20 \tag{2.11}$$

From the equation 2.11 we can derive a range for our CAN clock frequency (f_{CAN}):

$$5\,\text{MHz} \leq f_{\text{CAN}} \leq 10\,\text{MHz} \tag{2.12}$$

In this thesis, BRP has been chosen as 3 to comply with the constraint 2.12:

$$f_{\mathrm{CAN}} = \frac{24\,\mathrm{MHz}}{3} = 8\,\mathrm{MHz} \tag{2.13}$$

Based on this assumption and the equations 2.8 and 2.10 the t_{NBT} can be calculated:

$$t_{\mathrm{q}} = \frac{1}{8MHz} = 125\,\mathrm{ns} \tag{2.14}$$

$$t_{\mathrm{NBT}} = \frac{2\,\mu\mathrm{s}}{125\,\mathrm{ns}} = 16\,\mathrm{Time\ Quanta} \tag{2.15}$$

The t_{NBT} value as 16 Time Quanta satisfies the constraint given in the 2.6.

Lastly, time quanta (t_{q}) values for different segments shall be selected.

The sample point can be calculated with following equation:

$$\mathrm{Sample\ Point} = \frac{t_{\mathrm{NBT}} - t_{\mathrm{Phase\_Seg2}}}{t_{\mathrm{NBT}}} \tag{2.16}$$

If we combine the equation 2.16 with the constraint 2.5, a new constraint can be obtained:

$$12 \leq t_{\mathrm{NBT}} - t_{\mathrm{Phase\_Seg2}} \leq 13 \tag{2.17}$$

Based on ISO 11898-1 [28] the Sync_Seg shall be 1. The rest 15 t_{q} will be divided by Prop_Seg, Phase_Seg1 and Phase_Seg2. Considering this requirement and the constraint 2.17 the time quanta values of segments selected as follows in the table 2.4.

Parameter	**Value**
Sync_Seg	1 t_{q}
t_Seg1	11 t_{q}
t_Seg2	4 t_{q}
BRP	3
Bit Rate	500000 bit/s
System CAN Clock	24 MHz

Table 2.4: Bit Timing Related Values

3 Literature Review

This chapter provides an overview of relevant research and state-of-the-art developments in the field of Automotive Gateways.

The progression of research on Gateways and its evaluation over the years is apparent from the existing literature. Researches have approached Gateways using different methodologies. To reflect this diversity, the literature review is structured into subsections based on methodologies adopted in the relevant studies and each subsection presented in chronological order.

3.1 FPGA-based Gateway Literature

Traub, in his diploma thesis [11], evaluates various gateway architectures, with particular attention to the Institut für Technik der Informationsverarbeitung/DaimlerChrysler (ITIV/DC) Gateway. In this architecture, a dedicated routing engine and message Random Access Memory (RAM) are interfaced with multiple communication channels, including CAN, through the central gateway component, the Gateway-Network-on-Chip (GNoC). Traub's implementation achieved sub-10 μs latency at a system clock of 24 Megahertz (MHz).

Building upon this work, Sander et al. [60] further analyzed the mechanisms of the ITIV/DC Gateway, highlighting improvements such as transitioning from a purely hardware-based message RAM module to the adoption of a soft processor. This change was motivated by considerations of resource consumption and the need for reduced iterative development cycles. Additionally, Sander et al. introduced a tool chain for generating routing information, emphasizing the centrality of this process within their workflow. A Java-based parser was developed to utilize DBC files for creating routing definitions, which are then converted into .hex files to initialize the Block RAM (BRAM) blocks in the FPGA bit stream. With these enhancements, the team achieved an 8 μs latency at a 24 MHz system clock. Notably, their solution was benchmarked against contemporary high-performance 32-bit microcontrollers operating at 70 MHz, which exhibited latencies around 70 μs.

In his subsequent dissertation [61], Sander emphasized the advantages of FPGA-based implementations, particularly their hardware design flexibility. His architecture employed a hardware/software co-design approach, leveraging parallelization and pipelining as

key strategies. The dissertation provides further detail on the generation of routing configurations from DBC files and documents the successful integration and in-vehicle testing of the proposed gateway, which operated seamlessly in comparison to existing solutions. Sander also noted that AUTOSAR was primarily designed for microcontrollers, with gateway functionality only implicitly supported and dependent on other modules, potentially leading to latency challenges.

Traub, in his dissertation [62], presented an evaluation of a microcontroller-based AUTOSAR Classic gateway running at 80 MHz. Despite the increased system clock, no significant improvement in latency was observed. The focus of this work was on methodology, tooling, and the AUTOSAR framework, with an emphasis on the need to consider execution times and latency throughout the development process.

Lee et al. [63] proposed a CAN/FlexRay gateway implemented using a hardware/software co-design methodology on an FPGA. The gateway architecture included receive, send, and conversion modules, with the CAN controller realized in the Processing System (PS) section of the FPGA. Their implementation achieved a gateway latency of 10 μs. The results were compared with microcontroller-based and earlier FPGA-based implementations, although the exact system clock frequency was not specified. No substantial improvement was noted over previous FPGA implementations.

Shreejith et al. [64] introduced VEGa, an FPGA-based vehicular Ethernet Gateway architecture that leverages a hybrid FPGA to closely couple software control on a processor with a dedicated switching circuit on the reconfigurable fabric. Their architecture integrated both the PS and PL, and featured a binary search lookup table for header extraction from incoming frames. Notably, VEGa was among the first architectures to implement heterogeneous IVN technologies within an FPGA. Operating at a 125 MHz system clock, VEGa achieved CAN-to-CAN routing latencies below 5 μs, surpassing the performance of prior work.

3.2 Processor-based and SoC Gateway Literature

Xie et al. [65] conducted an analysis of the worst-case response time (Worst Case Response Time (WCRT)) associated with message processing in multi-core Gateway processors. Their findings indicate that utilizing a four-core configuration almost entirely eliminates interference between processing nodes. Furthermore, they observed that increasing the core count beyond four yields diminishing returns, as the degree of parallelization in gateway processing is inherently limited. Notably, their approach improved WCRT values by up to 74% compared to dual-core Gateway implementations.

Jo et al. [66] introduced a multi-core Gateway architecture alongside a novel scheduling algorithm. In their design, one core is dedicated to frame reception, while two additional cores handle the delivery of payloads and associated information to the appropriate

destination channels. This approach aims to minimize the number of frames stored within a single core, addressing limitations present in existing storage algorithms.

Kane et al. [67], together with Mariño et al. [68, 69], proposed SoC based Elastic Gateway solutions as an alternative, hardware-centric strategy. Their architecture demonstrates superior flexibility and scalability compared to existing solutions and seeks to introduce a Software Defined Networking (SDN) paradigm to automotive systems by decoupling data handling from the underlying hardware.

3.3 Gateway Review Articles

This subsection reviews state-of-the-art research in the form of review articles, which primarily evaluate existing works without introducing new architectural concepts for Gateways.

Jiang [13] examined E/E architectures and assessed recent development trends. While Gateways were not the primary focus, the analysis facilitated insights into the evolution of Gateway requirements in response to changes within the E/E architecture. Consequently, this study provides supporting evidence for several propositions advanced in this thesis (see Chapter 4), particularly regarding anticipated changes in Gateway requirements brought about by the adoption of Zonal E/E Architectures. Similar review articles by Bandur et al. [15] and Alparslan et al. [16] also considered Gateways in the context of E/E Architectures. However, these studies did not conduct a thorough analysis of the limitations and challenges facing current gateway architectures.

In contrast, Mariño et al. [2] provided an in-depth overview of the evolution of IVNs from distributed to zonal E/E architectures, with a specific focus on Gateways. The novelty of their contribution lies in the systematic review and classification of major Gateway architectures, including microcontroller-based solutions, CPU offloading, communication hardware accelerators, and Protocol Independent Switch Architecture (PISA). These architectures were compared against a set of functional and structural requirements, and the analysis revealed a significant gap: none of the surveyed Gateway architectures fully met all necessary criteria. The authors highlighted the promise of communication hardware accelerators and custom processor designs as important avenues for future research, underlining the need for more hardware-centric studies in this domain. They also advocated for the parallel evolution of functional and structural requirements. Although many of the requirements identified are more relevant to high-performance computers (HPCs) and may not be entirely realistic for Zonal Gateways, their analysis remains instructive. Additionally, while the study acknowledged the coexistence of Ethernet-based and traditional automotive communication technologies such as CAN, the evaluation of key functional requirements, such as latency, throughput, and performance, was primarily based on Ethernet technologies. It is also important to note that their selected metric for latency, end-to-end (E2E) latency, serves as a crucial system-level KPI for network

design but does not transparently capture the performance of the Gateway processing time itself, as E2E latency can be heavily influenced by network parameters such as busload and arbitration.

Despite these limitations, the work of Mariño et al. [2] provides a robust foundational basis for the research presented in this thesis.

4 Concept

This chapter aims to provide information on the requirements set for Gateways, concept of a Gateway and explains why the selected IVN technology CAN

4.1 Targeted Communication Technology and Routing Types

Based on an analysis of an existing Gateway, the distributions shown in Figure 4.1 and Figure 4.2 are obtained.

Figure 4.1 illustrates the distribution of routing entries in a Gateway across the three main routing types : PDU Routing, Transport Protocol (TP) Routing and Signal Routing.

The majority of TP Routing entries arise from diagnostic request and response frames, both on Ethernet CAN. Further details regarding the Signal Routing can are discussed in the Subsection 2.6.2.

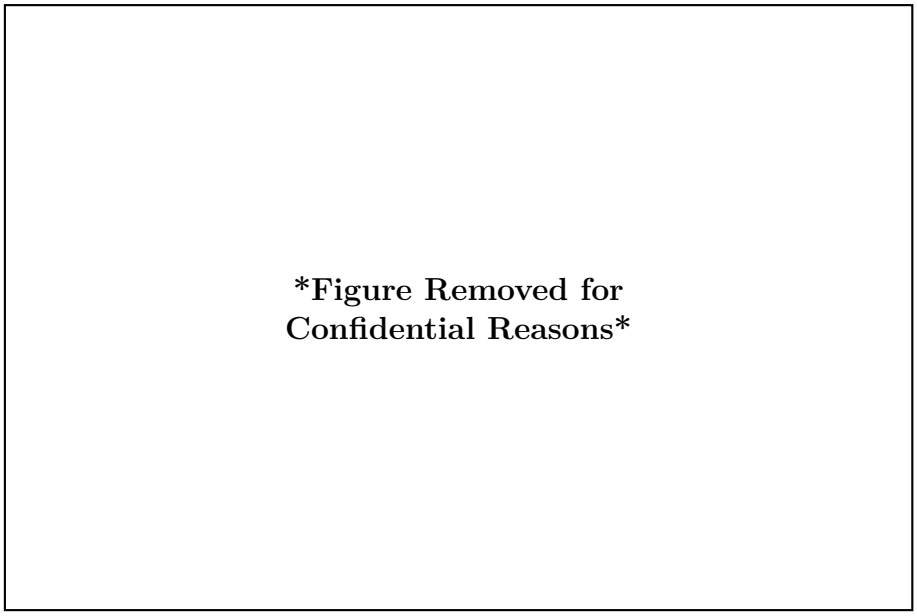

Figure 4.1: Distribution of Different Routing Entries in a Gateway

Although TP and Signal Routings are supported by Gateways, this thesis assumes that the Signal Routings will primarily be implemented in HPCs rather than Zonal Gateways. This is essential to increase the flexibility of Zonal Gateways while reducing the OEM and specific implementation dependency. Therefore, this thesis assumes that Signal Routing will not be implemented in Zonal Gateways. Diagnostic TP Routings, while relevant, is beyond the scope of this thesis and is suggested as a topic for future research of Zonal Gateways.

The focus of this thesis is exclusively on PDU Routings, which represents approximately 70% of all routing entries in a Gateway. Furthermore, diagnostic TP Routings typically occurs only during diagnostic or flashing sessions and not during normal vehicle operation. Therefore, PDU Routings account for almost all routings performed during the vehicle runtime.

Figure 4.2 presents the distribution of PDU Routings among the CAN2CAN, CAN2Ethernet and Ethernet2CAN Routings. As evident from the figure, despite the increasing adoption fo Automotive Ethernet in IVN architectures, CAN remains the predominant communication technology especially for heavy duty vehicle manufacturers. This continued prevalence is supported by established standards such as Society of Automotive Engineers (SAE) J1939, among other factors, suggesting that CAN will remain relevant to IVN for the foreseeable future.

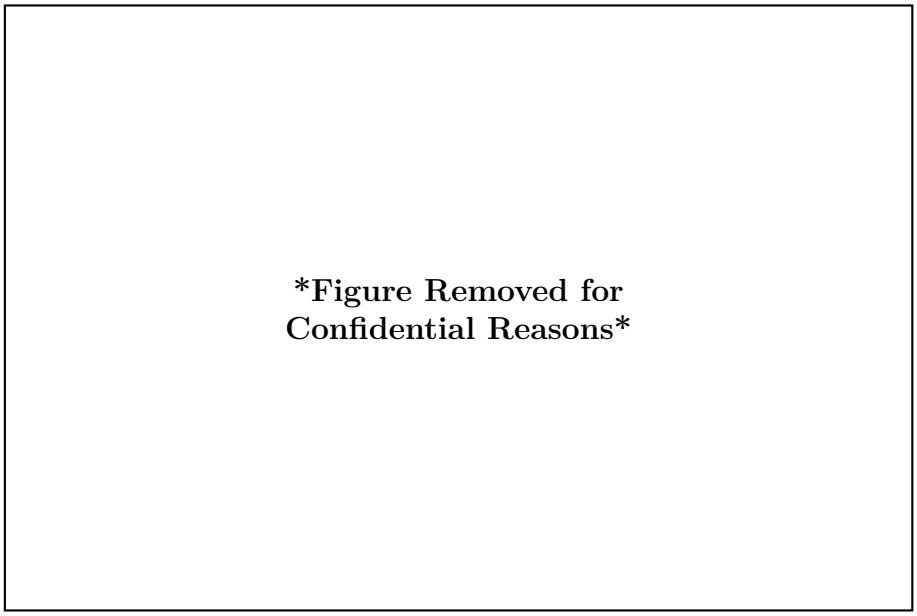

Figure 4.2: Distribution of PDU Routings Across Different Communication Technologies

Consequently, the conceptual framework developed in this thesis is primarily based on CAN technology, while also acknowledging the potential of future integration of Ethernet-based communication solutions.

4.2 Drivers of the Concept: Challenges and Opportunities

4.2.1 Current Challenges

The vast majority of traditional Automotive Gateways are implemented on microcontrollers using AUTOSAR Classic Software Architecture [2].

Despite the considerable complexity and significant initial investment [3] associated with AUTOSAR, many of these projects continue to experience high CPU loads and in some cases data loss.

In AUTOSAR-based microcontrollers, elevated CPU load is often attributable to the BSW [70]. Within BSW modules, both cyclic main functions and Interrupt Service Routines (ISRs) are key contributors to the CPU load. Increased communication load further exacerbates the frequency of ISRs, posing particular challenges for Automotive Gateways.

To address high CPU loads, the industry is currently pursuing two primary solutions: Multicore BSW Concept [71, 72] and the usage of Communication Hardware Accelerators [9, 55, 8].

While splitting BSW across multiple cores can reduce the CPU load on the main core, it also introduces additional integration complexity and presents new challenges in other architectural areas [71, 72].

Recent reports highlights emerging CPU offloading technologies from silicon vendors (refer to the Chapter 2.6.3), such as routing engines that offload the CPU and use hardware accelerators for routing activities [8, 9]. However, there is currently a lack of comprehensive industry research and few completed projects utilizing these products, meaning that vehicles employing such technologies are not yet common. According to the available reports [8], these solutions appear promising in terms of reducing both latency and the CPU load. Nevertheless, these solutions seem particularly applicable to the complex Gateway solutions, such as HPCs, due to flexibility and cost effectiveness making them less suitable for Zonal Gateways, where multiple units are required in one vehicle.

Another challenge coming with AUTOSAR-based Gateway ECUs is the statically defined routing paths [50, 51].

Given that OEMs such as Daimler Truck operate multiple brands across various continents, it is common for the same ECUs utilized for different products such as trucks and busses. This practice may face with a challenging situation with the introduction of the Zonal E/E Architecture (refer to the Subsection 2.2.2), as ECUs will be assigned to networks based on their physical location within a vehicle.

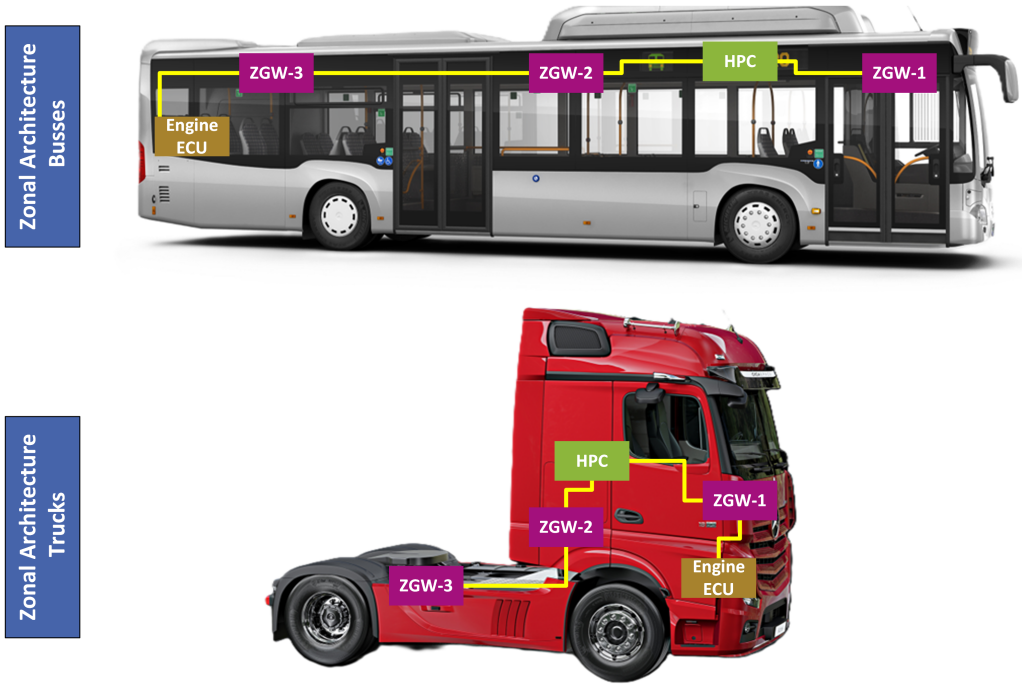

Figure 4.3: Hypothetical Engine ECU Placement in Trucks and Buses with Zonal Architecture

For example, as illustrated in the Figure 4.3, the engine related ECU may be connected to the rear Zonal Gateway in busses, due to rear engine placement, while in trucks, it may be connected to the front Zonal Gateway for the same reason. Although, this scenario is hypothetical, it highlights the potential challenges that may arise as heavy duty vehicle manufacturers adopt Zonal E/E Architecture.

In such scenarios, statically defined routings within both front and rear Zonal Gateway would necessitate the creation of software variants for these gateways (namely, a front Zonal Gateway with and without Engine ECU, and similarly for the rear Zonal Gateway). Managing these software variants presents additional challenges for OEMs, increasing development costs, complexity in communication matrix design (refer to the Subsection 2.4.1) and complexity in project management.

4.2.2 Opportunities Arising from Trends

There are two trends impacting the way Gateways are designed : SDV and Zonal E/E Architecture.

While SDVs primarily influence HPCs, the introduction of the Zonal E/E Architecture affects not only HPCs but also Zonal Gateways.

As outlined in the Subsection 4.2.1, the Zonal E/E Architecture introduces new challenges for existing gateway designs, particularly in case they are used in Zonal Gateways. Nevertheless, it also presents significant opportunities.

One of the most notable changes associated with the Zonal E/E Architecture is a substantial reduction in the overall number of networks (refer to Subsection 2.2.2). Additionally, since Zonal Gateways are typically connected to two channels, this configuration may enable dynamic routing, which eliminates the need for statically defined routing paths.

The possibility of dynamically routing paths might shift the complexity from the design of Gateways to network level design parameters such as busloads. But dynamic routings may be the way we face with the challenge of the need of software variants outlined in the Subsection 4.2.1.

Another change associated with the Zonal E/E Architecture is the fact that there are three or four Zonal Gateways required per vehicle depending on the vehicle type (refer to the Subsection 2.2.2). This might seem to be a challenge but from a big picture this can be considered as an opportunity which will push OEMs to find a generic solution for Zonal Gateways with less or no software updates required pointing to a more hardware-centric approach.

Given aforementioned challenges in existing Gateway architectures and new challenges expected with trends, this thesis focuses on a new hardware-centric Gateway architecture concept given in the Subsection 4.4.

4.3 Important Requirements for Gateways

In this thesis, the requirements derived from the identified concept are presented in this subsection. The requirements are categorized into two groups, following the methodology outlined by Mariño et al.[2]: Functional Requirements (FRs) and Structural Requirements (SRs).

These key requirements will serve as the basis for evaluation the implementations in this thesis.

4.3.1 Requirement 1: Latency

In this thesis, the latency of the routing is defined as follows and illustrated in the Figure 4.4.

$$t_{\text{latency}} = t_{\text{Start of Frame, Network B}} - t_{\text{End of Frame, network A}}$$

This definition more clearly isolates the time required for routing processing within the Gateway itself, independent of network level parameters, in contrast to the E2E Latency.

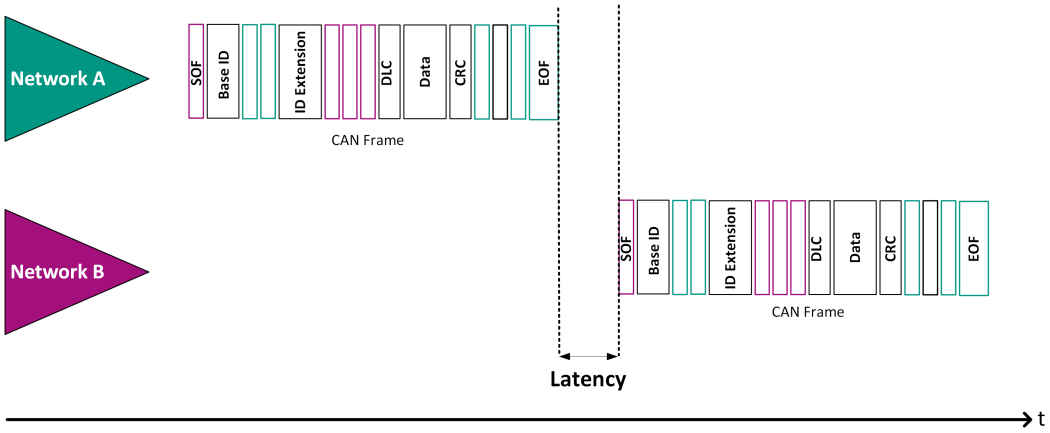

Figure 4.4: Latency Definition

4.3.2 Requirement 2: Flexibility

Flexibility, as derived from the discussions in Subsections 4.2.1 and 4.2.2, is defined as the ability of a Gateway to perform routing independently of static routing definitions or the need for software updates when new frames are introduced into the communication matrix

This requirements also reflects the capability of the Gateway concept to handle different sets of ECUs across multiple networks without necessitating hardware or software variants.

Furthermore, this parameter encompasses the potential to deploy the same Gateway in various zones without significant modifications to its software, thereby contributing to the enhanced scalability.

4.3.3 Requirement 3: Complexity

Complexity is intended to reflect the level of difficulty involved in implementing the Gateway concept. As complexity can be challenging to quantify, this thesis considers factors such as the number of different tools required for the implementation, associated expertise needed, the amount of configuration parameters and the number of lines of C or VHSIC Hardware Description Language (VHDL) code that must be written.

4.3.4 Requirement 4: Availability

Availability assesses whether necessary tools and expertise are readily accessible regardless of cost considerations, or whether additional, concept specific steps must be developed from scratch to complete the implementation.

This parameter also indicates the extent to which the concept can be realized using standardized, mainstream industry methods.

4.3.5 Requirement 5: Development Cost

Development Cost represents the total estimated expenses associated with the required tools, licenses and engineering hours, as well as any additional development efforts and software variants needed, taking into account the approximate project timeline.

4.3.6 Requirement 6: Adaptation

Adaptation, in this thesis, refers to the ability to accommodate rapidly changing requirements from OEMs, such as the introduction of a new security algorithm after the SOP.

4.4 Concept of the Gateway

Based on rationale presented in Subsections 4.2.1, 4.2.2, 4.1, the Gateway concept proposed in this thesis envisions a Gateway architecture that prioritizes gatewaying functions such as routing and protocol translation. An illustration of this approach, alongside the conventional Gateway approach for comparison is provided in the Figure 4.5 (b) and (a) respectively.

As shown in the Figure 4.5 (b) and discussed in Subsection 4.1, this thesis primarily focuses on the CAN to CAN (CAN2CAN) routings.

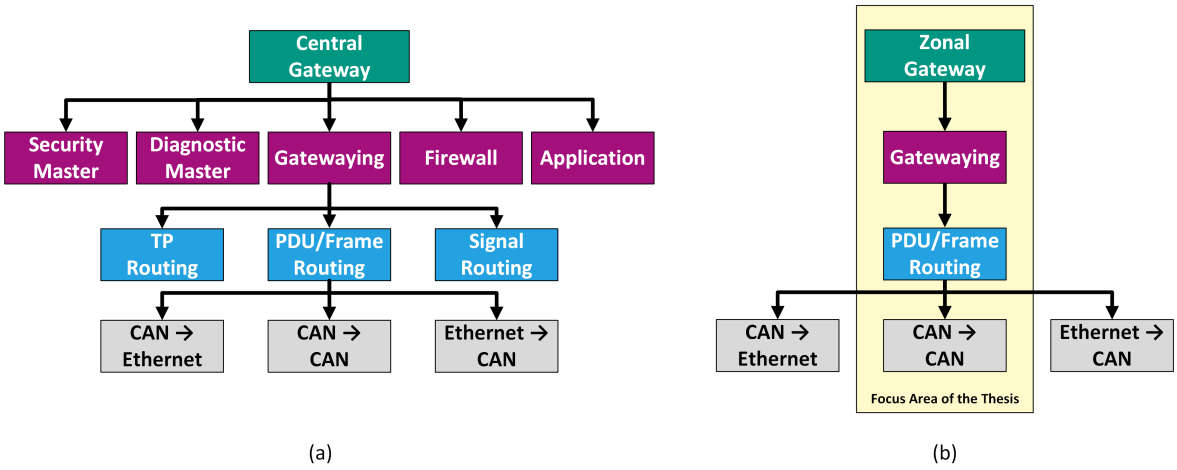

Figure 4.5: From Current Gateways to Projected Gateway Concept

With the objective of establishing CAN2CAN routing set aside for now, the remaining task is how this can actually be achieved conceptually.

To address this, the concept begins with a thought experiment aimed at identifying the least complex approach to implement a Gateway.

The primary goal of this thought experiment is to forward (route) traffic from one network to another, assuming as an example from Network A to Network B, as illustrated in the Figure 4.6.

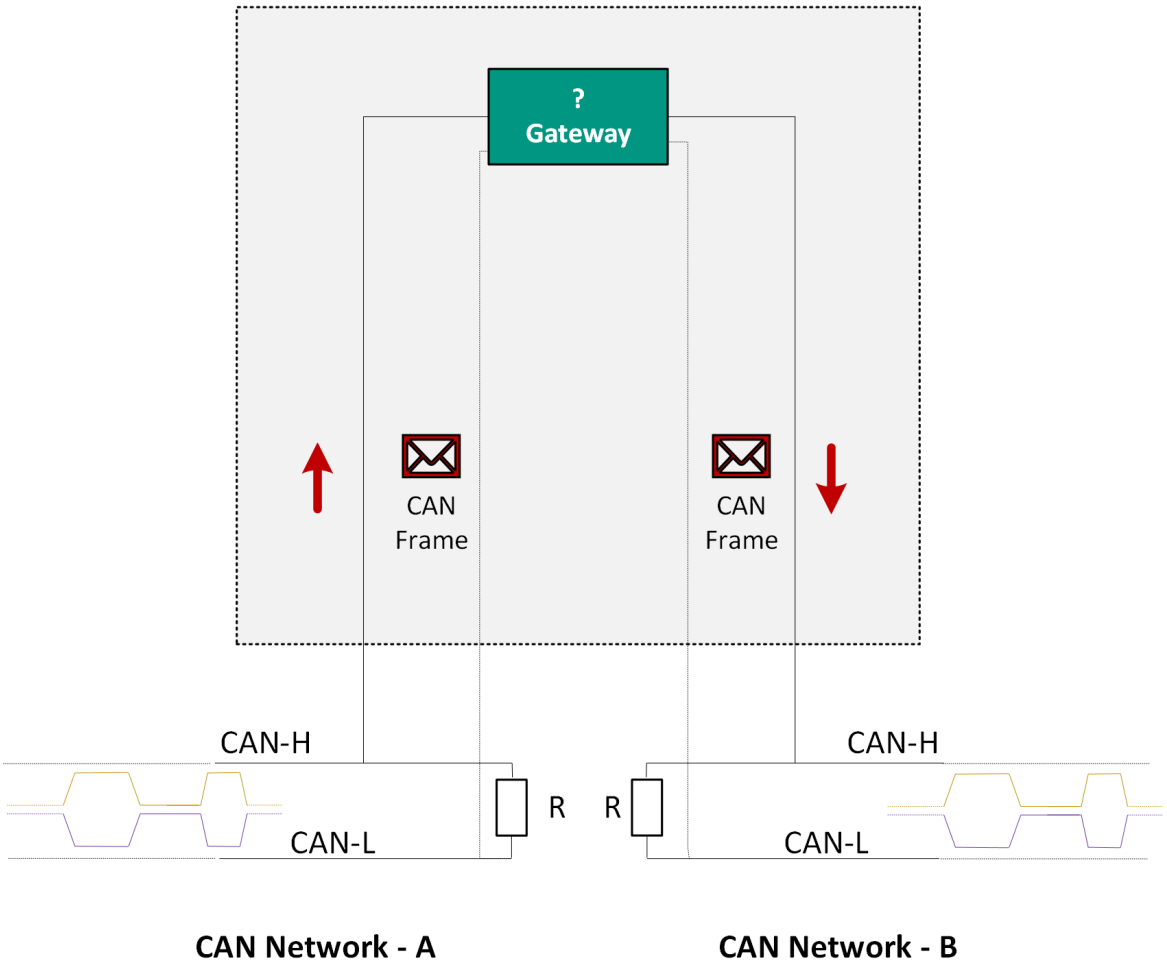

Figure 4.6: Gateway Concept Thought Experiment

The simplest approach would be to directly mirror analog signals from Network A to Network B. This method would require neither logic nor software, and could be implemented entirely in the hardware, resulting in a robust and highly cost-effective solution.

However, this approach is not feasible due to the architectural characteristics of CAN Networks. CAN Networks operate as multi-master [27] networks, where each node is allowed to start sending a frame when the CAN bus is idle. In case bus is not idle, the node which is intended to send the new frame should wait until the bus is idle again. On the other hand, if multiple nodes would like to send CAN frames simultaneously the one with the higher priority Frame ID is allowed to continue the transmission while other shall retrieve due to carrier sense multiple access/collision resolution [27]. Consequently, the ongoing traffic on Network B can prevent the direct mirroring from Network A, making purely hardware based mirroring solution unworkable.

Eliminating the hardware mirroring option necessitates the inclusion of some form of a memory in the Gateway. Incoming traffic from Network A may need to be temporarily stored until Network B is ready to accept new frames (when the bus is idle). Therefore, memory elements are essential for the Gateway.

Furthermore, since analog signals cannot be directly stored in the memory, the Gateway must include entities capable of translating analog signals on CAN Networks A and B into digital values. These components can be grouped as CAN interfaces.

With the ability to read and write frames thanks to CAN interfaces and store them in the memory, the system now requires a logic to manage post reception activities. After receiving a CAN Frame, this logic must decode the frame into its fields, such as CAN Frame ID, Data Length Code (DLC) and Data Payload.

The logic must then determine whether to store this information in memory (WriteMem) or to forward it directly to the destination CAN interface. If the frame must be stored in the memory, or if there are already frames waiting in the memory, the logic must identify the appropriate time to retrieve the data (ReadMem), reconstruct CAN frame and send it to the destination interface.

The final aspect of the thought experiment involves determining whether the Gateway logic should process every received frame or operate based on predefined settings that specify which frames to process and which to ignore. This consideration is closely linked to the underlying E/E Architecture and the type of Gateway employed. In a Distributed E/E Architecture, specifically with a CGW (refer to Subsection 2.2.1), a filtering mechanism, commonly referred to as a routing table is necessary due to large number of networks connected to the CGW.

However, as outlined in the Subsection 4.2.2 the adoption of the Zonal E/E Architecture presents an opportunity to reduce overall number of networks and introduces a new Gateway type, the Zonal Gateway. In the context of this thought experiment, a Zonal Gateway is considered, which does not require filtering of incoming traffic, as there is only a single target network within the Zonal Architecture.

While this approach may increase the complexity of the communication matrix design, it reduces the complexity of the Zonal Gateway itself. This opens a possibility to generalize a Zonal Gateway solution, particularly with regard to SRs. As a result, the implementation of lookup tables or routing tables within the Gateway logic becomes unnecessary.

Based on thought experiment described above, the resulting Gateway concept is illustrated in the Figure 4.7.

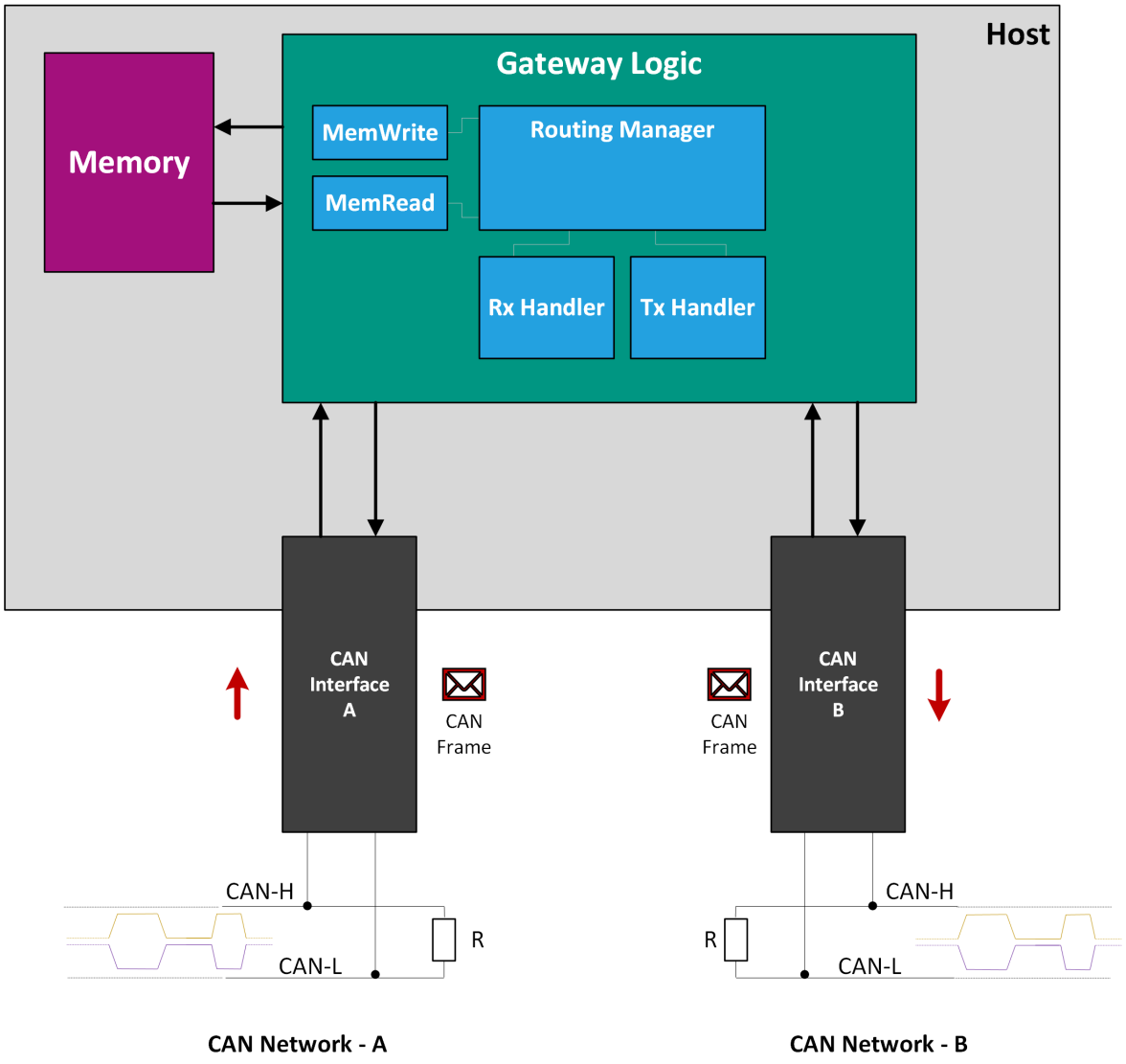

Figure 4.7: Concept of a Gateway

5 Implementation

This section details the implementations carried out in this thesis on both the microcontroller and FPGA platforms.

The microcontroller implementation has been carried out using AUTOSAR Classic Software Architecture in order to create a basis to compare with the implementation carried out based on the concept given in the Subsection 4.4. Additionally, carrying out this implementation from scratch helps to evaluate especially SRs given in the Subsection 4.3. Lastly, this implementation has been realized to find the limitations of AUTOSAR Classic Software Architecture for Automotive Gateway as outlined in the Subsection 1.1.

For the concept implementation, an FPGA platform has been choosen. There are several reasons why an FPGA platform has been chosen for realizing the concept. First of all, in the concept and the motivation of this thesis a hardware-centric Gateway architecture is thought. Secondly, FPGA platforms provide necessary parallelism, customization and flexibility [73, 60] required for the concept implementation.

5.1 CAN Extension Board

In this thesis, a dedicated CAN Extension Printed Circuit Board (PCB) was developed to facilitate accurate measurements and minimize errors from faulty cabling during testing with both oscilloscopes and CAN measurement tools equipped with DSUB-9 connectors.

The CAN Extension Board includes slots for two NXP TJA1041AT CAN Transceivers, enabling support for both CAN Network A and Network B in the FPGA implementation.

The CAN Extension Board provides the option to enable or disable the CAN termination resistors.

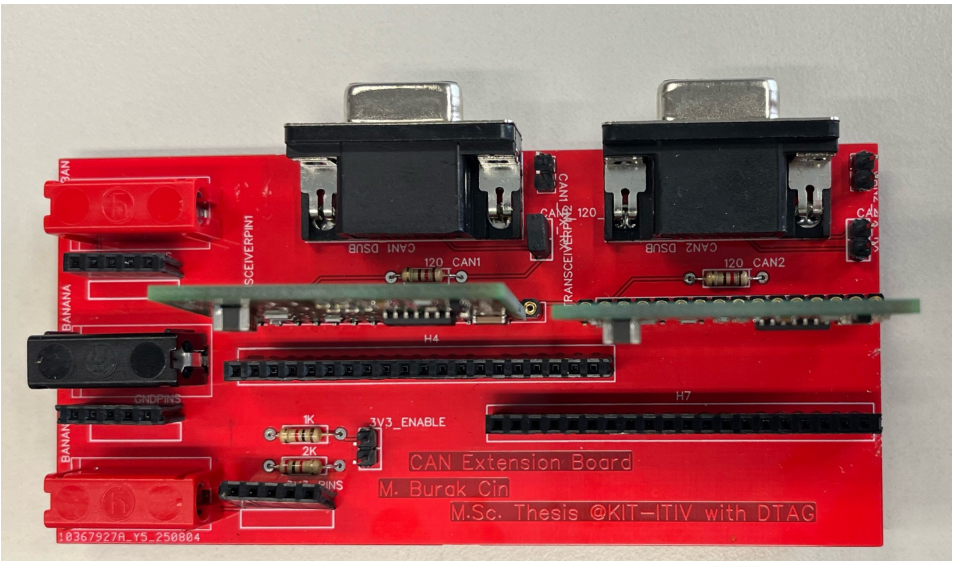

Figure 5.1: CAN Extension Board

For the microcontroller implementation, these external transceivers are not required, as the microcontroller board is already equipped with onboard CAN transceivers. Nevertheless, the CAN extension board can be utilized by connecting the CAN-H and CAN-L cables to appropriate female pins, which are routed to the DSUB-9 connectors on the extension board.

5.2 Microcontroller Implementation with AUTOSAR

For the microcontroller implementation, the Infineon Triboard is used. This board is equipped with an Infineon TC399XX-256 F300S BD microcontroller and features onboard Infineon TLE9251 CAN FD transceivers.

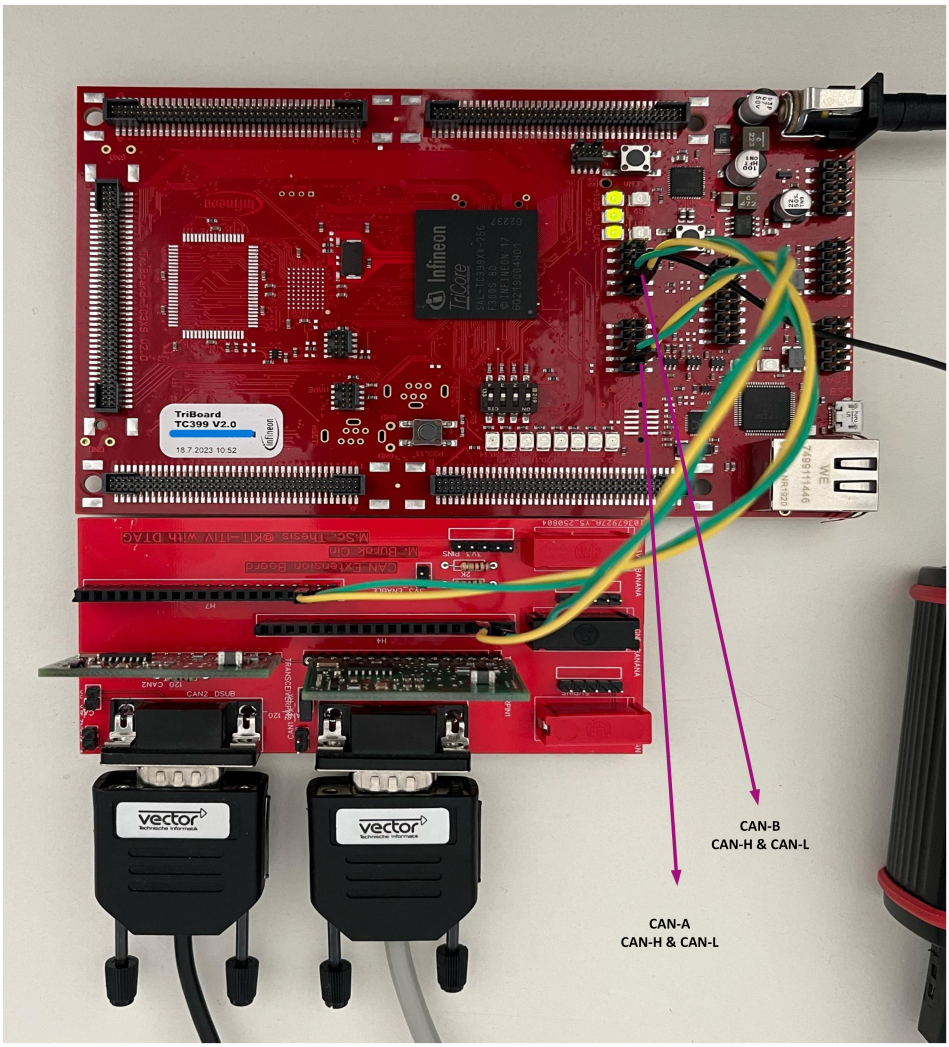

Figure 5.2: Microcontroller Board in Combination with CAN Extension Board

The implementation process in AUTOSAR typically begins with the creation of ECU Extract (ARXML), as described in the Subsection 2.5.3.

For this thesis, a dedicated ARXML file was created using schema version AUTOSAR_00046 for this thesis. This ARXML file contains numerous essential design parameters required for software generation, most importantly for this thesis, including definitions of PDU routing paths, as illustrated for a representative single PDU routing in Figure 5.3.

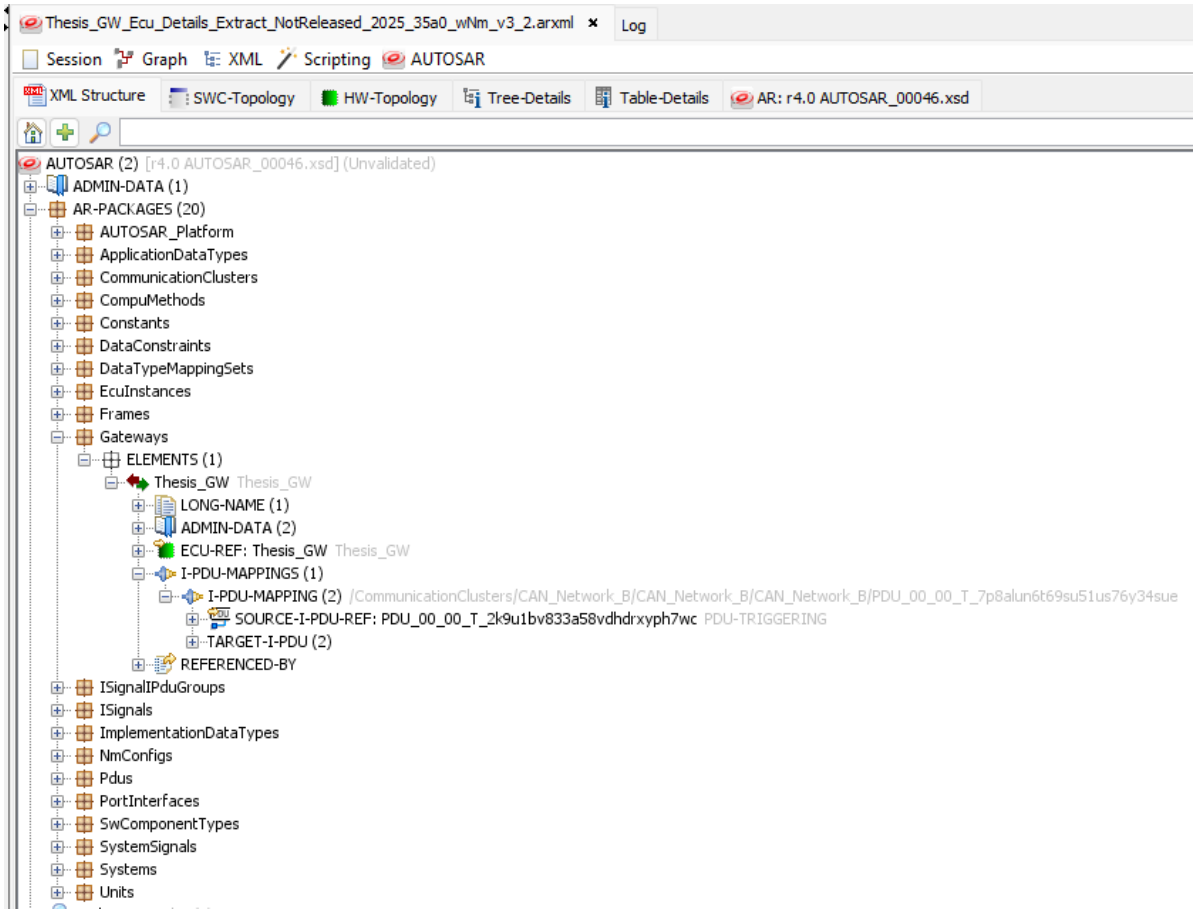

Figure 5.3: ECU Extract (ARXML) in ARXML Visualizer Tool with One Representative Routing

The subsequent step in the implementation is to create a project within the BSW Configuration tool. In this thesis, the Vector MICROSAR Classic BSW Stack is employed. The default BSW Configuration tool for this stackis the DaVinci Configurator tool, into which the ARXML is imported.

MICROSAR conforms to the Implementation Conformance Class (ICC)-3 [44, 36], where each BSW module is treated as a separate entity. During the implementation, a total of 25 different BSW modules needed to be configured, including standard AUTOSAR modules such as PduR, as well as specific modules to the MICROSAR BSW Stack.

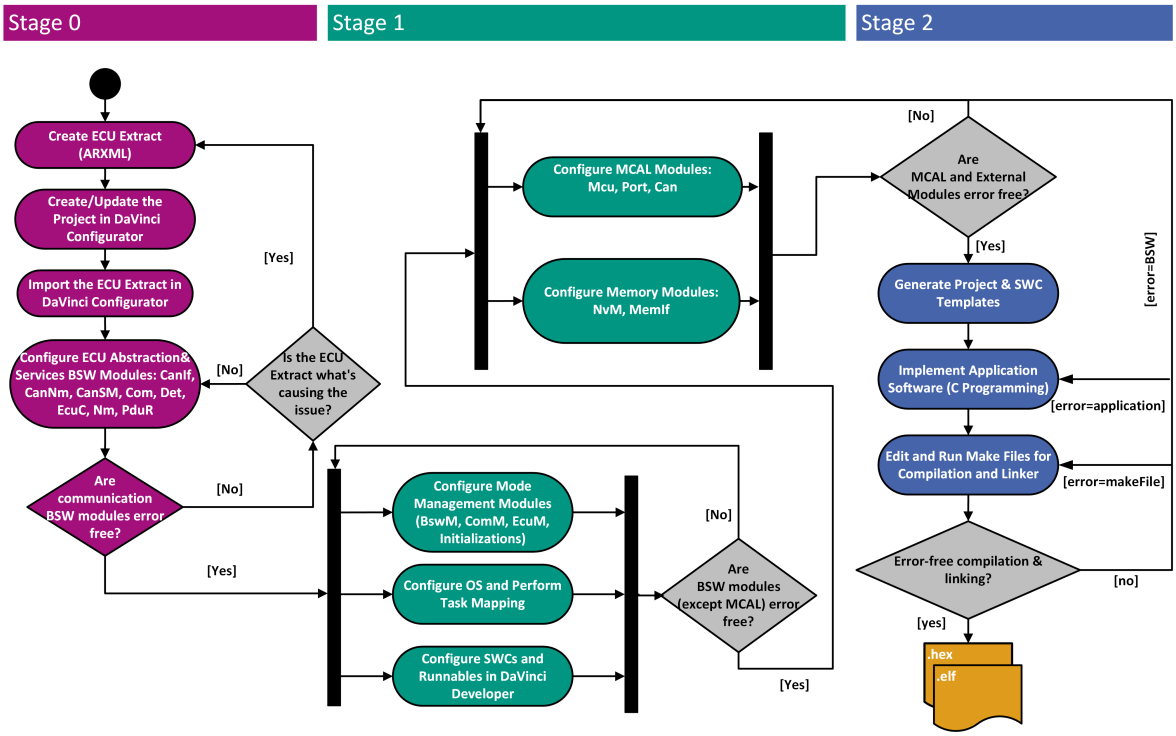

Figure 5.4: Activity Diagram of Microcontroller Implementation in AUTOSAR

There is no standardized procedure for configuring the BSW modules. However, the methodology adopted in this thesis begins with configuring BSW modules that depend on the ARXML content such as, PduR, NM and COM. As depicted in the activity diagram in Figure 5.4, this step is referred as Stage 0.

Stage 1 involves configuring the mode management modules and setting fundamental OS parameter. To ensure ECU remains awake independently of the NM for a convenient test procedure, the DaVinci Developer tool is used to create access points for necessary Communication Manager (ComM) request calls. The corresponding Rte_Call APIs are then written into the .c file of the relevant dummy atomic SWC. However, beyond this, the DaVinci Developer tool is not required, as the Gateway does not host any SWC design.

The second part of the Stage 1 focuses on configuring hardware dependent MCAL modules. The required MCAL modules and licenses are provided by Infineon and are configured using the Elektrobit EB Tresos tool. Once the MCAL modules are configured, the memory modules can be set up in the DaVinci Configurator tool.

Stage 2 encompasses compiling, linking, debugging and flashing activities. In this thesis, the Tasking compiler is used for compilation, while debugging carried out with the iSystems iC5700 debugger and the WinIDEA tool.

5.3 Implementation with FPGA

For the implementation of the concept presented in this thesis, the Xilinx (AMD) Zynq UltraScale+™ MPSoC ZCU102 FPGA Board is used. In this thesis, only PL side of the FPGA utilized and no PS functionalities incorporated.

The implementation on FPGA begins with configuring the clocking wizard. Two separate clock domains are required: one for the system and another for CAN modules. The Advanced eXtensible Interface (AXI) CAN IP [59] used in this thesis supports a maximum clock frequency of 24 MHz, which is insufficient for the overall system requirements. For the system clock, both 100 MHz and 300 MHz options were evaluated and for the final design 300 MHz kept.

Following the clock configuration, a system reset IP was integrated to facilitate the system reset when the CPU reset button on the FPGA board is pushed.

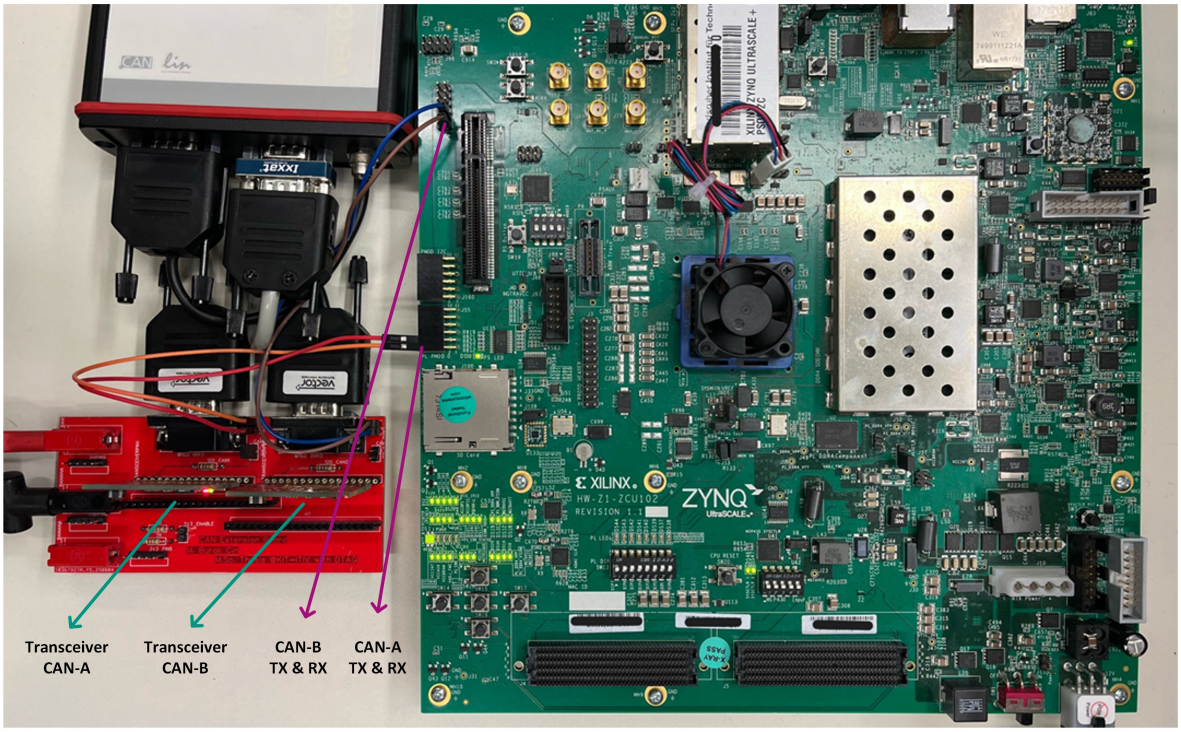

Figure 5.5: FPGA Board in Combination with CAN Extension Board and CAN Measurement Equipment

To establish necessary CAN channels, two AXI CAN IPs are instantiated. These, in conjunction with the physical CAN transceivers provided by CAN Extension Board, comprise the required CAN interfaces as outlined in the concept. Each AXI CAN IP

requires a connection to the CAN clock, as well as a system clock to support its glsAXI4-Lite [74] slave interface. Furthermore, the active-low reset signal from the system reset IP is connected to CAN IPs. The CAN-Rx and CAN-Tx signals were made external pins, an through constraint files, these were mapped to physical Peripheral Module Interface (PMOD) pins on the board, as shown in the Figure 5.5. Additionally, both the transmit and receive First In First Out (FIFO) depths for AXI CAN modules configured to 64.

To satisfy the memory requirements described in the Subsection 4.4, an AXI BRAM Controller and Block Memory Generator were added. The AXI BRAM module operates with an AXI4-Lite slave interface. The data width of 32 bits and depth of 2048 were configured for the BRAM. The BRAM is also connected to the system clock and the active low reset signal for its AXI slave interface.

With necessary peripheral and fundamental modules such as clocks and memory in place, the central component, the gateway logic was implemented. For this purpose, a dedicated Gateway_Logik IP was developed. Owing the previously described peripherals, the Gateway_Logik IP requires three AXI4-Lite master interfaces in order to control both CAN and BRAM modules.

The remainder of the implementation is conducted within the Gateway_Logik IP, which is developed using VHDL. The initial responsibility of the Gateway_Logik IP is to configure and initialize the CAN channels by writing appropriate values into the designated registers, as specified in the AXI CAN user manual [59]. Additionally, the Gateway_Logik reads relevant status registers from the CAN modules to determine whether they are operating in the normal mode, indicating that CAN modules are ready to process (send and receive) CAN frames on the bus. Once the modules are confirmed to be enabled, two General Purpose Input Output (GPIO) Light Emitting Diodes (LEDs) are activated on the board. The visual indication eases the debugging by making it clear when the CAN channels are operational.

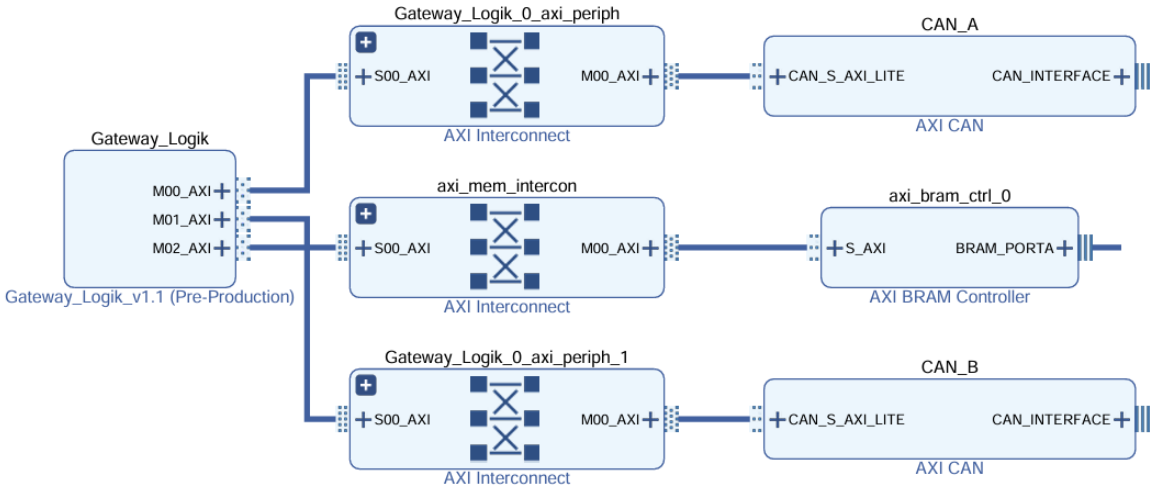

Figure 5.6: Vivado Block Design for FPGA Implementation

The Gateway_Logik implements a state machine comprising three primary states: idle, read and write. Upon the reception of a CAN frame on either channel, an interrupt is generated, prompting Gateway_Logik to receive the frame and decode it into elements of : Frame ID, DLC and Data Payload. Subsequently, Gateway_Logik verifies the status of the Tx buffers on the target interface. If Tx buffers are full, the decoded frame information, along with the target network ID (0 for CAN-A, 1 for CAN-B), is stored in the BRAM. If the buffers are not full, the frame is written directly to the CAN Tx buffer for the transmission. Additionally, Gateway_Logik periodically checks for pending data in the BRAM by utilizing an external counter. When the CAN Tx buffers become available, the module initiates the transmission of stored frame.

Table 5.1 presents the resource consumption for a single CAN Controller and the Gateway_Logik IP.

Function	**Registers**	**Lookup Tables (LUTs)**	**BRAMs**
CAN (1 Controller)	689	607	1.5
Gateway_Logik	427	151	–

Table 5.1: Resource Consumption FPGA

6 Evaluation

In this chapter, the test setup is first described. Following this, the results of both microcontroller-based AUTOSAR Gateway implementation and the FPGA-based Gateway implementation are discussed, based on previously defined FRs and SRs presented in the Subsection 4.3.

6.1 Test Setup

The CAN-A channels of both microcontroller and FPGA boards are connected in daisy chain form via CAN Extension Boards, forming the CAN Network-A.

To evaluate the routing functionality of both microcontroller and FPGA implementations, the first channel of Measurement Box 1 was also connected to Network-A as well and served as the dummy sender. This setup allowed the transmission of multiple CAN frames with various configuration through the Vector CANoe tool. Frames could be sent either manually by specifying parameters such as Frame ID and cycle time or by selecting predefined Frames from the the ARXML file generated for the microcontroller implementation.

As outlined in the Subsection 5.3, the FPGA implementation is capable of routing all Frames transmitted by the dummy sender on Network-A to Network-B. In contrast, the microcontroller implementation can only route specific Frames based on statically defined PDU routings in the ARXML.

The test setup is illustrated in the Figure 6.1.

An additional measurement box, Vector 1630A (referred to as Measurement Box 2), is used to capture measurements from the microcontroller. Its first channel is also connected to the Network-A for monitorung purposes and accurately assess the number of dropped routings later.

Channel 2 of the Measurement Box 1 is connected to the CAN-B channel of the FPGA via the CAN Extension Board to monitor routed Frames via FPGA from Network-A to Network-B. This network is referred to as Network B_FPGA.

Similarly, channel 2 of the Measurement Box 2 connected to the CAN-B channel of the microcontroller via the second CAN Extension Board, forming the Network B_µC.

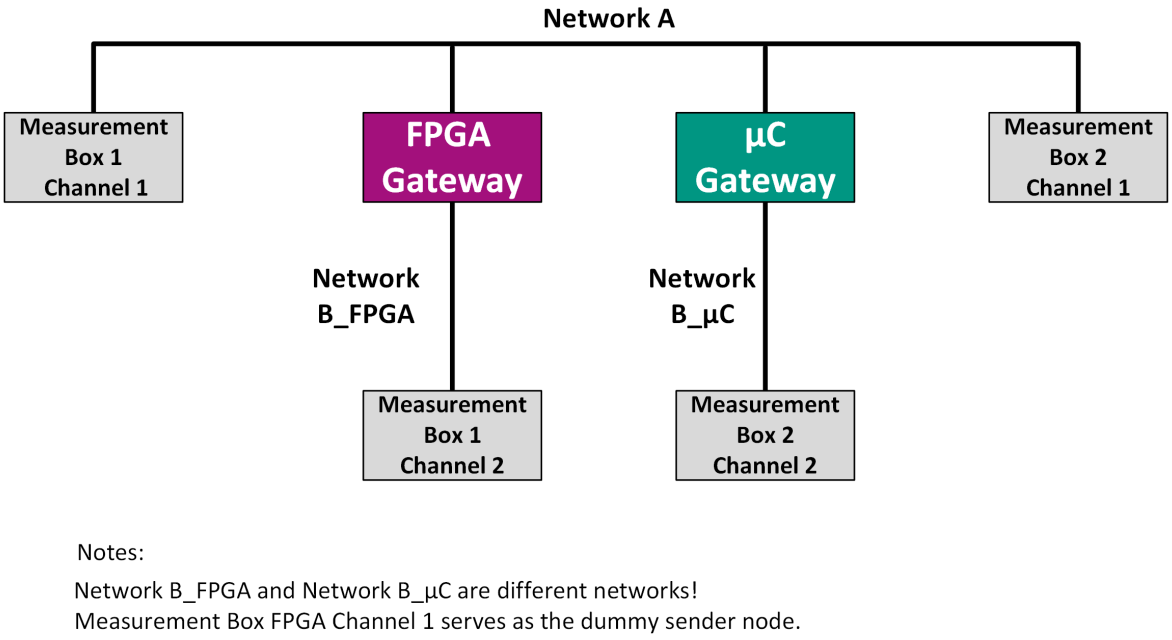

Figure 6.1: Test Setup

As indicated in the Figure 6.1, Network B_FPGA and Network B_µC are distinct networks.

The realization of the test setup has been showed in the Figure 6.2.

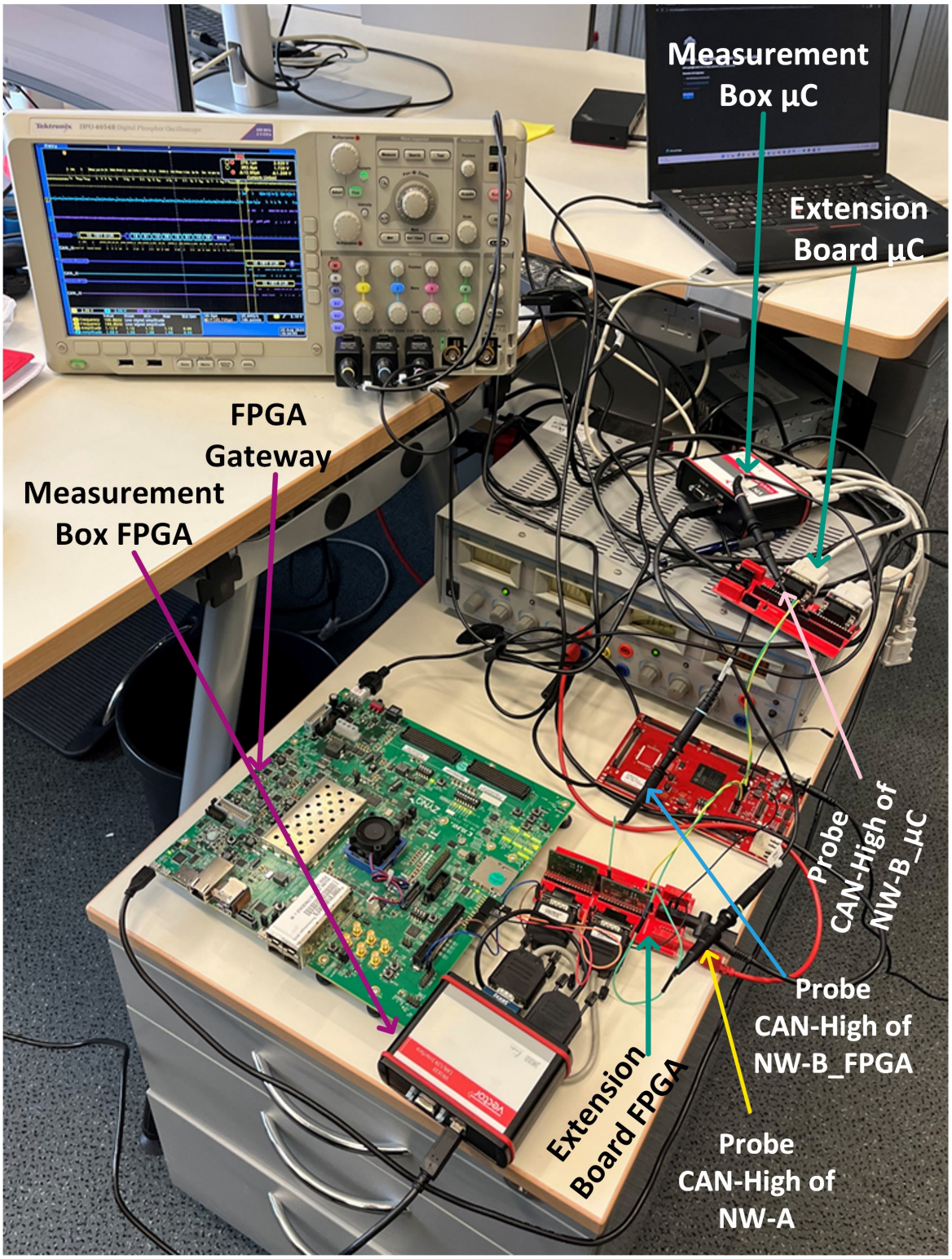

Figure 6.2: Measurement Setup

6.2 Evaluation of Requirement 1 : Latency

Latencies are being measured according to the method described in Figure 4.4 of Subsection 4.3.1, using an oscilloscope for precise measurement.

Figure 6.3 presents an oscilloscope screenshot illustrating routing via the microcontroller implementation running at 300 MHz, from Network-A to Network-B.

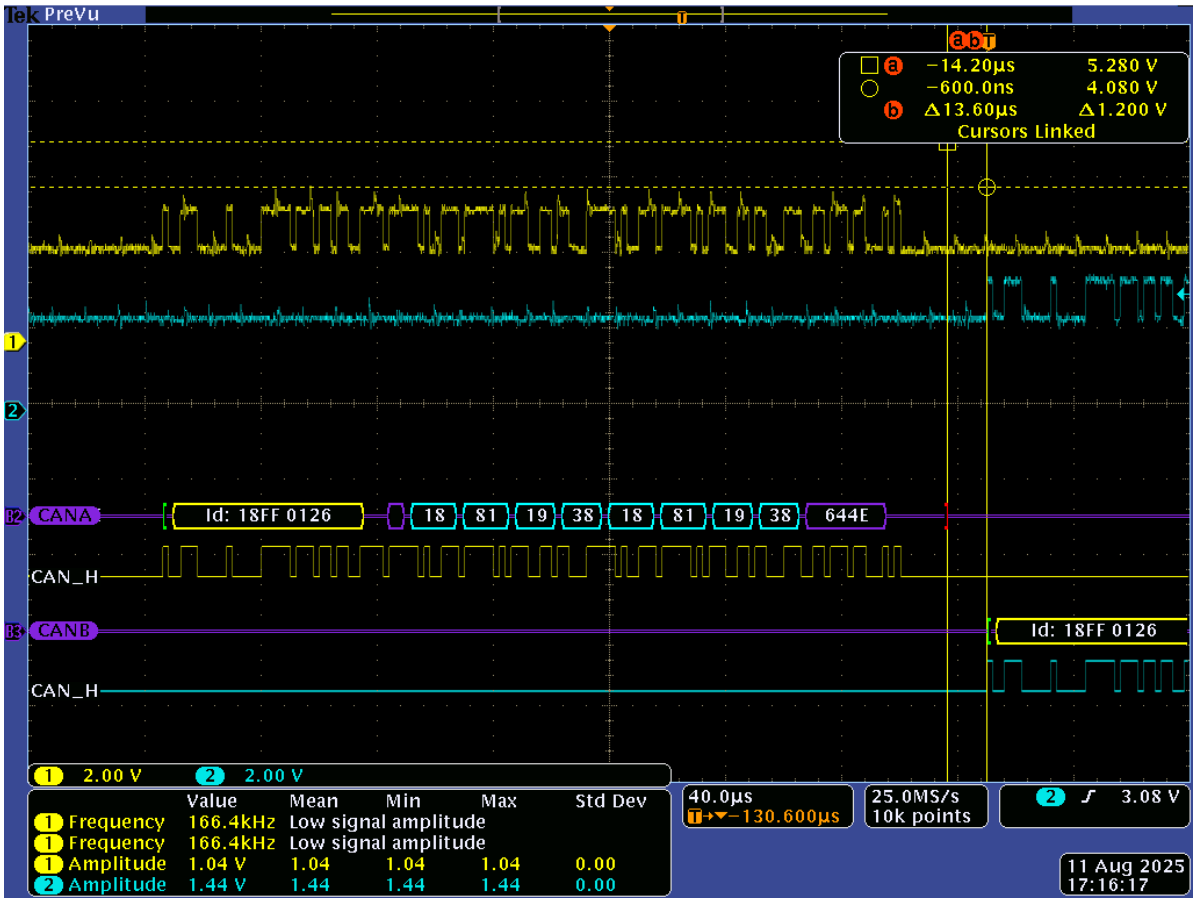

Figure 6.3: Latency of a PDU Routing in AUTOSAR & Microcontroller

Oscilloscope measurements with FPGA Gateway under identical conditions demonstrate that routing latencies in nanosecond range can be achieved. This is shown in the Figure 6.4.

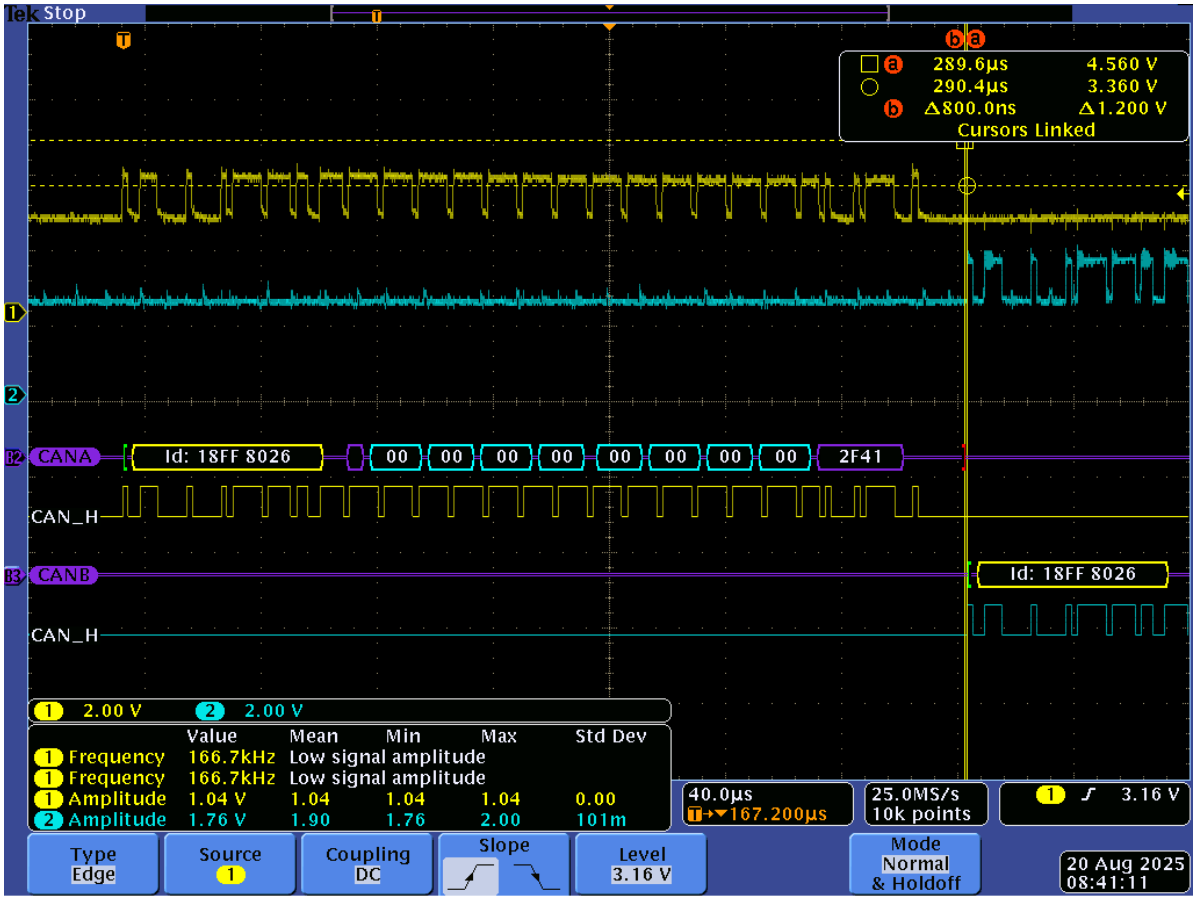

Figure 6.4: Latency of a Routing with FPGA

To facilitate an easy comparison, the oscilloscope results from the measurement setup shown in Figure 6.2 are presented in the Figure 6.5, in which microcontroller and FPGA connected to the system simultaneously.

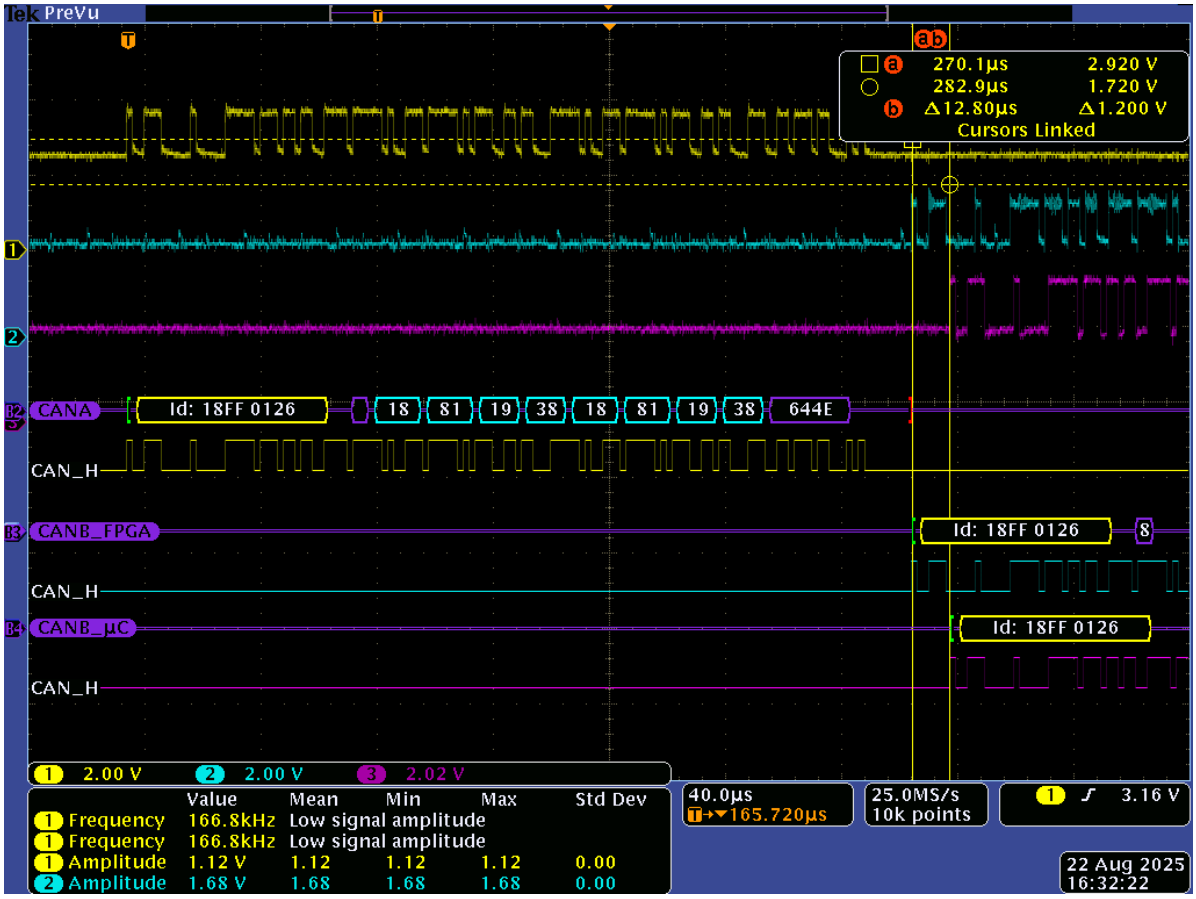

Figure 6.5: Latency of a Routing both Microcontroller and FPGA

In Figure 6.5, the routing latency between CAN-A and CAN-B_µC can be detected easily as 12.8 µs , representing the microcontroller latency. However, the routing latency between CAN-A and CAN-B_FPGA, representing FPGA latency cannot be easily distinguished, appearing almost instantaneous in this time/div setting. Therefore, a zoomed-in (decreased time/div) screenshot is provided in Figure 6.6, where latency is measured at 760 nanoseconds.

Figure 6.6: Latency of FPGA Zoomed in

More than hundred different routing measurements were conducted, with the resulting data summarized in the Table 6.1.

	Min (µs)	**Avg (µs)**	**Max (µs)**
Microcontroller 300 MHz	12.4	13.9	14.8
FPGA 100 MHz	1.80	1.90	2.10
FPGA 300 MHz	0.66	0.80	0.89

Table 6.1: Comparison of routing latencies for different hardware platforms

A comparison of the results obtained in this thesis with those reported in the literature is presented in Figure 6.7. This figure clearly illustrates that the FPGA implementation developed in this thesis, operating at 300 MHz, surpasses previous implementations and stands out the only architecture achieving nanosecond-level routing latency.

While 500 kbit/s may be considered relatively low, especially considering the new generation CAN XL enables data rates up to 20 Mbit/s, such high data rates are

generally utilized only after the arbitration phase. Therefore, although there are no restrictions in the standards, even in next generation CAN implementations, arbitration is expected to remain at Classical CAN data rates level.

Due to above mentioned reason with arbitration phase, beyond achieving nanosecond-level latency, it is also noteworthy that the latency observed with the FPGA implementation in this thesis is less than the duration of a single bit time at 500 kbit/s on CAN bus.

The Gateway characteristic of having latency less than a duration can therefore provide a significant advantage for future Gateway Nodes, allowing them to maximize throughput without being substantially affected by overall busload.

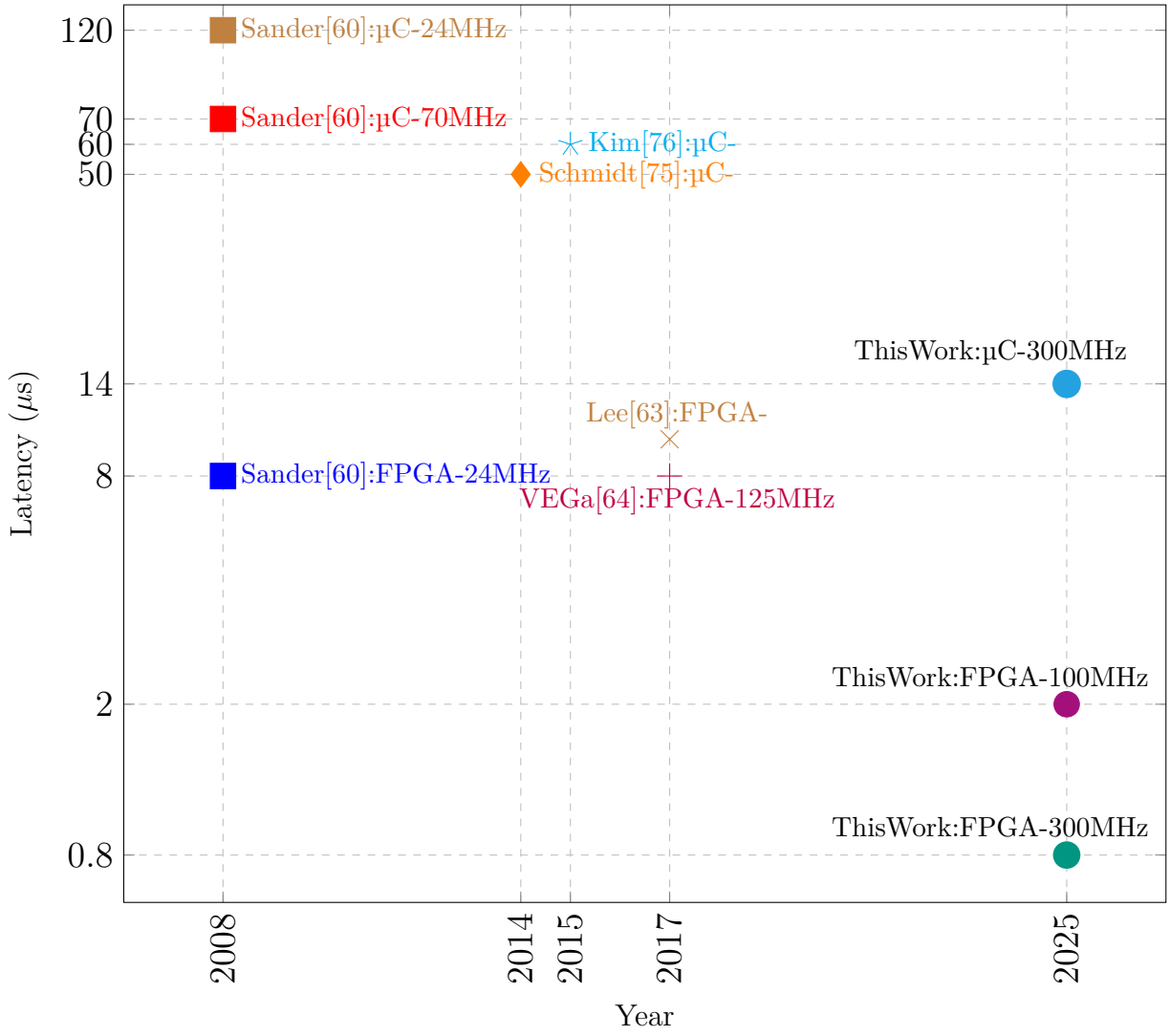

Figure 6.7: Latency Comparison Over the Years

Another notable observation from Figure 6.7 is that, although the microcontroller runs at 300 MHz, the achieved latency is comparable to that of FPGA-based implementations running considerably lower clock rates. Nevertheless, a significant improvement due to enhanced hardware performance is evident in the microcontroller implementation presented in this thesis when compared to previous works in the literature [60, 75, 77].

In summary, the evaluation presented above supports the idea of adopting more hardware-centric Gateway architectures to achieve improved latency performance.

6.3 Evaluation of Requirement 2 : Flexibility

Based on the definition given in the Subsection 4.3.2, FPGA implementation addresses the challenge described in the Subsection 4.2.1 through its capability of dynamic routing.

In comparable scenarios, microcontroller implementation with AUTOSAR would require additional software variants or post-build selectable options, both of which increase complexity in the software development process and the design of the communication matrix.

In conclusion, the FPGA-based Gateway implementation with dynamic routing capability offers enhanced flexibility, which is particularly beneficial for applications in Zonal Gateways.

6.4 Evaluation of Requirement 3 : Complexity

The complexity involved in both of microcontroller based AUTOSAR Gateway and FPGA-based Gateway implementations has been evaluated by listing the required tools and licenses in the Table 6.2. This comparison reflects the necessary expertise and potential team size may be required for a real-world Gateway project.

Additionally, the use of ARXML significantly facilitates and automates the configuration of numerous BSW parameters. While the availability of ARXML is advantageous, bringing a microcontroller board into operation with AUTOSAR Software Architecture still requires the definition of more than 10,000 parameters in the BSW configuration tool, particularly those related to the hardware.

In contrast, the FPGA-based Gateway was developed using the AMD Vivado Design Tool, which provides a more straightforward experience from design till bit stream generation. Features such as block design and the IP generator further contribute to a degree of automation in the development process.

	μC-based AUTOSAR Gateway	**FPGA-based Gateway**
Tool 1	XDIS (System Design)	AMD Vivado™ Design Suite
Tool 2	ARXML Visualizer (System Design)	
Tool 3	Vector DaVinci Configurator (SW Configuration)	
Tool 4	Vector DaVinci Developer (SW Configuration)	
Tool 5	Elektrobit EB Tresos (HW Configuration)	
Tool 6	TASKING Compiler (Compiler)	
Tool 7	WinIDEA (Debugger IDE)	
License 1	Tasking Compiler License	AXI CAN IP License
License 2	DaVinci Configurator License	
License 3	DaVinci Developer License	
License 4	Infineon MCAL License incl. EB Tresos License	
License 5	Vector MICROSAR BSW Software Package License	
Hardware 1	Infineon TC399 V2.0 Triboard (incl. CAN Transceivers)	AMD (Xilinx) Zynq UltraScale+ MPSoC ZCU102 ZCU102 Board
Hardware 2	iSystem Debugger (Debugger Hardware)	NXP TJA1041AT CAN Transceivers

Table 6.2: Tool, License, and Hardware Comparison for Microcontroller Based AUTOSAR Gateway and FPGA Gateway

In summary, although both implementations require a certain level of expertise, the FPGA-based Gateway implementation is perceived as less complex and can typically be accomplished with a smaller team compared to AUTOSAR Gateway.

6.5 Evaluation of Requirement 4 : Availability

AUTOSAR, as the mainstream software architecture in the automotive industry [2], offers a high availability. MCAL solutions support a wide range of microcontrollers, and a variety of vendors provide comprehensive BSW stack solutions.

The standardized methodology provided by AUTOSAR, which is continually advanced through efforts of the AUTOSAR partnership, ensures readiness for both legislative and technology driven requirement. More importantly, it offers a standardized methodology and thanks to the AUTOSAR partnership it is being further developed which makes it ready for the legislative or technology driven features ready. Also there are various companies offer BSW Stack solutions.

In contrasts, FPGA platforms have not yet achieve standardization within the automotive industry. While fundamental soft IP cores such as CAN IPs are available, certain essential functionalities such as NM are not currently offered as off-the-shelf solutions and may be require project specific implementation.

In summary, microcontroller-based Gateways equipped with AUTOSAR Classic software architecture offers higher availability compared to current FPGA-based Gateways.

6.6 Evaluation of Requirement 5: Development Cost

Considering required tools´and licenses listed in the Table 6.2, the initial investment for a microcontroller-based AUTOSAR Gateway is substantially higher.

Furthermore, the necessity for a larger development team, extended development timelines,a on ongoing software updates after the SOP contribute to the perception of microcontroller-based AUTOSAR Gateways as high cost architectures.

In contrast, FPGA-based implementations offer higher scalability and owing to their flexibility enabled by dynamic routing, they may not require sofware updates, resulting in a more cost-effective solution in terms of development costs.

6.7 Evaluation of Requirement 6: Adaptation

Thanks to the higher availability evaluated in the Subsection 6.5, microcontroller-based AUTOSAR Gateway solutions offer greater adaptability in scenarios where OEMs require additional features after the project start or after SOP. Such requirements often arise from new legislative demands or rapidly evolving OEM specifications.

For instance, new legislative requirements related to the cyber security can be addressed using AUTOSAR based solutions such as Secure Onboard Communication (SecOC) [78].

In contrast, automotive specific, ready to deploy solutions are not yet common for FPGA platforms, making rapid adaption more challenging in these cases.

6.8 Evaluation Summary

Figure 6.8 provides a summary of the evaluation. From this figure, it can be deduced that availability and adaptation capability are the two requirements where the FPGA-based Gateway do not reach the levels of the microcontroller-based AUTOSAR Gateway.

In all other areas, the FPGA-based Gateway demonstrates superior performance compared to the microcontroller-based AUTOSAR Gateways.

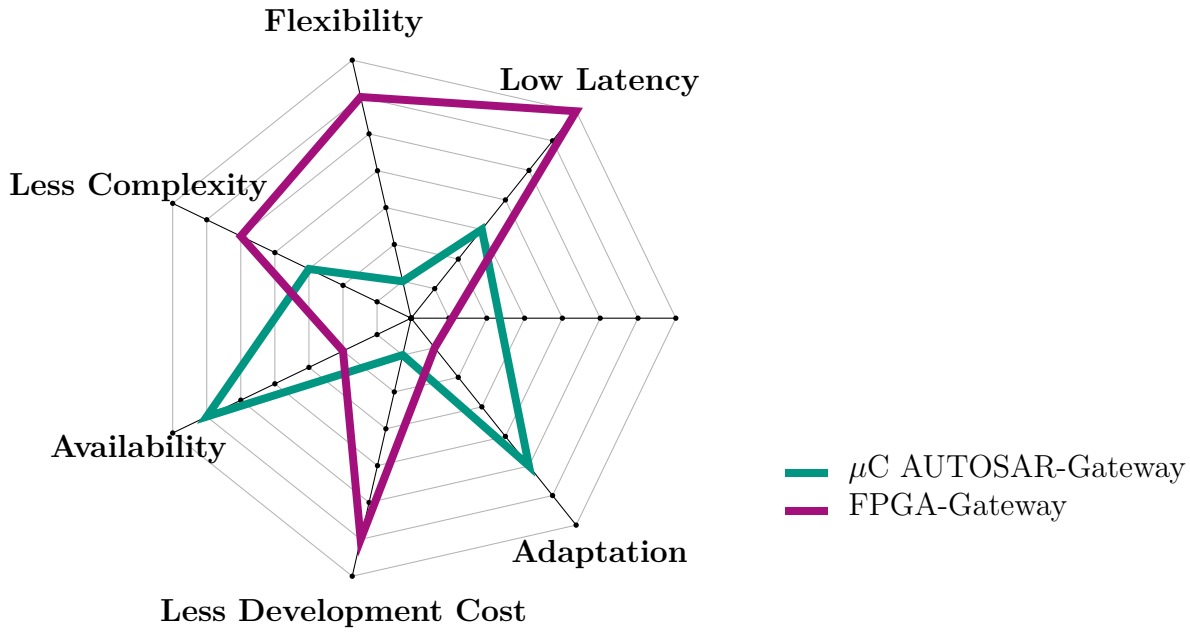

Figure 6.8: Evaluation Summary Chart

7 Conclusion and Future Work

In this final chapter, the thesis work is summarized. The chapter begins by highlighting the main contributions, followed by a review of results and conclusions. Finally, the limitations are discussed and potential directions for future research are proposed.

7.1 Summary

Zonal E/E architectures and SDVs are poised to transform numerous aspects of the automotive industry, including the design and functionality of gateways.

This thesis begins by providing a detailed foundational background on AUTOSAR, covering aspects that are often insufficiently addressed in existing literature, such as PDU Routing, the updated layered software architecture with the L-SDU Router, and the transition from domain-centralized to zonal E/E architectures. The impact of this shift on gateways, particularly on Zonal Gateways, is also examined.

A comprehensive literature review is conducted, revealing a historical emphasis on hardware-centric gateway solutions over the past two decades. The review identifies a recent research trend focused on Ethernet communication technologies, with a declining interest in CAN-related studies. However, a significant gap is found in the literature: there is a lack of fundamental performance data regarding gateways implemented with modern hardware, despite today's platforms being considerably more capable than those from twenty years ago. As a result, the implications for latency and other key parameters remain largely unexplored. Another limitation identified is that current industrial solutions, such as CPU offloading and Multicore architectures, are tailored to complex high-performance computers (HPCs) but do not adequately address, or may not be feasible for Zonal Gateways.

Following this, statistical data derived from a real-world gateway project is presented, highlighting the prominence of PDU Routing and CAN within current systems. These findings underscore the continued relevance of CAN, particularly for heavy-duty vehicle manufacturers, suggesting that a complete transition to fully Ethernet-based E/E architectures is unlikely to occur immediately and will likely be gradual.

The thesis proceeds to analyze the key drivers for new concepts, beginning with the current challenges faced by microcontroller-based AUTOSAR Gateways. While AUTOSAR has established itself as a mainstream software architecture in the automotive industry for

over two decades, offering significant benefits for complex, application-centric ECUs such as ADAS ECUs, it is argued here that AUTOSAR was not specifically designed for gateway applications. The absence of a dedicated basic software (BSW) module for gateways means that gateway functions are incorporated into the general PDU Router module, which introduces significant overhead and requires numerous API calls for routing tasks. Consequently, gateway projects often inherit the complexity and costs associated with AUTOSAR but do not fully benefit from its strengths. In addition, the high CPU load associated with AUTOSAR-based gateways is a major concern. Although solutions such as hardware acceleration (CPU offloading) and multicore implementations are being explored in both academia and industry, these approaches are often complex and primarily applicable to HPCs, rendering them impractical for zonal gateways due to cost and complexity constraints. Furthermore, the requirement for statically defined routings in AUTOSAR gateways hinders the flexibility necessary for Zonal E/E Architectures.

In addition to identifying current challenges, this work views SDVs and Zonal E/E Architectures as opportunities to fundamentally redefine Gateway architectures. These trends not only demand greater flexibility but also foresee an E/E architecture with fewer ECUs, as HPCs assumed to take over more critical vehicle functions. Zonal E/E Architectures also reduce the number of networks compared to Domain Centralized E/E Architectures. Since ECUs are now positioned based on physical location rather than function, information previously confined to a domain may now need to be routed across different zones, underscoring the need for dynamic routing solutions, a central focus of this thesis.

Based on these observations, the essential requirements for future gateways, especially for Zonal Gateways, are articulated. Key requirements include increased importance of latency due to dynamic routing, enhanced flexibility, reduced complexity and costs, and the ability to rapidly adapt to evolving OEM and regulatory requirements.

The proposed concept in this thesis originates from a thought experiment aimed at developing the least complex yet high performance Gateway solution, considering that at least three Zonal Gateways will be deployed per vehicle. The rationale is that a simpler architecture will enable better standardization and scalability. Given the high degree of parallelism afforded by independent operations and the limited number of required states thanks to the nature of routing, FPGAs are identified as an optimal platform for implementation.

During the implementation phase, both a conventional microcontroller-based AUTOSAR gateway and the proposed FPGA-based gateway (using only the PL) are realized. The microcontroller-based gateway serves as a functional baseline to transparently assess the advantages and disadvantages of the AUTOSAR approach.

The evaluation results indicate that the microcontroller-based AUTOSAR Gateway exhibits latencies exceeding 10 μs, while the FPGA-based Gateway achieves nanosecond-level latencies, with an average of 800 ns. Furthermore, the FPGA-based Gateway demonstrates greater flexibility and lower complexity and projected development costs. Although

it currently does not match the availability and adaptability offered by microcontroller-based AUTOSAR Gateways, the FPGA-based Gateway architecture proposed in this thesis outperforms existing solutions and introduces a novel concept for zonal gateways.

7.2 Limitations and Future Work

This work is based primarily on the classical CAN protocol, however, in the context of evolving vehicle E/E Architectures, the adoption of the latest generation, CAN XL, is anticipated to become increasingly prevalent. Accordingly, future studies should consider applying the methodologies developed in this thesis to CAN XL.

A further limitation identified is that the proposed concept is expected to significantly influence busload and network communication design. Currently, there is a lack of studies in the literature that evaluate the feasibility of such an architecture, particularly with respect to dynamic routing capabilities and Zonal E/E Architecture.

Another notable limitation concerns the omission of fundamental services that are typically required by almost all ECUs as mandated by OEMs, such as network management and diagnostics. In this thesis, these functions have not been incorporated into the implementation of the FPGA-based Gateway. Moreover, no suitable IP cores for network management were identified during the research.

To address these limitations, this work recommends the development of additional automotive-related IPs for FPGA platforms and encourages further research into the effects of Zonal Architectures on network communication design and busload.

Abbreviations

ADAS	Advanced Driver Assistance Systems
API	Application Programming Interface
ARXML	AUTOSAR XML
ASIC	Application Specific Integrated Circuit
AUTOSAR	Automotive Open System Architecture
AXI	Advanced eXtensible Interface
BRAM	Block RAM
BRP	Baudrate Prescaler
BSW	Basic Software
CAN	Controller Area Network
CAN FD	CAN Flexible Data-Rate
CAN XL	CAN extended data-field length
CAN-H	CAN High
CanIf	CAN Interface
CAN-L	CAN Low
CDD	Complex Device Driver
CEFF	Classical Extended Frame Format
CGW	Central Gateway
COM	Communication
ComM	Communication Manager
CPU	Central Processing Unit
DBC	Database for CAN
DLC	Data Length Code
E/E	Electric/Electronic
E2E	End to End
ECU	Electronic Control Unit
EthIf	Ethernet Interface
FIFO	First In First Out
FPGA	Field Programmable Gate Array
FR	Functional Requirement
GNoC	Gateway-Network-on-Chip
GPIO	General Purpose Input Output
HPC	High Performance Computer
ICC	Implementation Conformance Class
ID	Identifier

IEEE	Institute of Electrical and Electronics Engineers
IP	Intellectual Property
I-PDU	Interaction -Presentation- Layer PDU
ISO	International Organization for Standardization
ISR	Interrupt Service Routine
ITIV/DC	Institut für Technik der Informationsverarbeitung/DaimlerChrysler
IVN	In-Vehicle Network
KPI	Key Performance Indicators
LED	Light Emitting Diode
LinIf	LIN Interface
L-PDU	Data Link Layer PDU
L-SDU	Link Layer SDU
LUT	Lookup Table
MCAL	Microcontoller Abstraction Layer
MHz	Megahertz
MSB	Most Significant Bit
NBT	Nominal Bit Time
NM	Network Management
N-PDU	Network Layer PDU
OEM	Original Equipment Manufacturer
OS	Operating System
OSEK	Offene Systeme und deren Schnittstellen für die Elektronik in Kraftfahrzeugen
OSI	Open Systems Interconnection
PCB	Printed Circuit Board
PCI	Protocol Control Information
PCS	Physical Coding Sublayer
PDU	Protocol Data Unit
PduR	PDU Router
PISA	Protocol Independent Switch Architecture
PL	Programmable Logic
PMOD	Peripheral Module Interface
POSIX	Portable Operating System Interface
PS	Processing System
RAM	Random Access Memory
Rte	Runtime Environment
SAE	Society of Automotive Engineers
SDN	Software Defined Networking
SDU	Service Data Unit
SDV	Software Defined Vehicle
SecOC	Secure Onboard Communication
SIC	Signal Improvement Capability
SoC	System on Chip
SOF	Start of Frame
SOP	Start of Production

SR	Structural Requirement
SWC	Software Component
TP	Transport Protocol
UML	Unified Modeling Language
VHDL	Very High Speed Integrated Circuit Program (VHSIC) Hardware Description Langua
VHSIC	Very High Speed Integrated Circuit Program
WCRT	Worst Case Response Time
WG-MT	AUTOSAR Working Group Methodology and Templates
XML	Extensible Markup Language

Bibliography

[1] Can in Automation (CiA). *CAN XL Proof of concept by Daimler Buses*. 2023.

[2] Angela Gonzalez Marino, Francesc Fons, and Juan Manuel Moreno Arostegui. "The Future Roadmap of In-Vehicle Network Processing: A HW-Centric (R-)evolution". In: *IEEE Access* 10 (2022), pp. 69223–69249.

[3] Silverio Martínez-Fernández et al. "A Survey on the Benefits and Drawbacks of AUTOSAR". In: *Proceedings of the first international workshop on automotive software architecture*. 2015, pp. 19–26.

[4] Springer Fachmedien Wiesbaden. ""AUTOSAR has Become Mature and Accepted"". In: *ATZextra worldwide* 18.9 (Oct. 2013). Publisher: Springer Science and Business Media LLC, pp. 13–15.

[5] AUTOSAR. *AUTOSAR 20TH ANNIVERSARY*. 2023.

[6] Shugang Jiang. "Vehicle E/E Architecture and Key Technologies Enabling Software-Defined Vehicle". In: *SAE Technical Paper Series*. WCX SAE World Congress Experience. ISSN: 0148-7191, 2688-3627. Detroit, Michigan, United States: SAE International, Apr. 9, 2024.

[7] Brian Carlson. *The Rise and Evolution of Gateways and Vehicle Network Processing NXP*. 2019.

[8] Thomas Boehm. *Welcome to the next generation AURIX TC4x Infineon*. 2022.

[9] Arthur Mackay and Chikita Nangia. *PFE/LLCE Communications Offload Engines NXP*.

[10] Mathias Rausch. *Kommunikationssysteme im Automobil: LIN, CAN, CAN FD, CAN XL, FlexRay, Automotive Ethernet*. München: Carl Hanser Verlag GmbH & Co. KG, Oct. 10, 2022.

[11] Matthias Traub. *Evaluierung von Gateway-Architekturen für Netzwerke im Kraftfahrzeug und Implementierung eines FPGA-basierten CAN-FlexRay-Gateways*.

[12] Markus Oertel and Bastian Zimmer. "More Performance with Autosar Adaptive". In: *ATZelectronics worldwide* 14.5 (May 2019), pp. 36–39. ISSN: 2524-8804. DOI: 10.1007/s38314-019-0049-x. URL: http://link.springer.com/10.1007/s38314-019-0049-x (visited on 06/16/2025).

[13] Shugang Jiang. "Vehicle E/E Architecture and Its Adaptation to New Technical Trends". In: WCX SAE World Congress Experience. Apr. 2, 2019.

[14] Dr. Andreas Lock. “Trends of Future E/E-Architectures Robert Bosch GmbH”. 2019.

[15] Victor Bandur et al. “Making the Case for Centralized Automotive E/E Architectures”. In: *IEEE Transactions on Vehicular Technology* 70.2 (Feb. 2021). Publisher: Institute of Electrical and Electronics Engineers (IEEE), pp. 1230–1245. ISSN: 0018-9545, 1939-9359. DOI: 10.1109/tvt.2021.3054934. URL: https://ieeexplore.ieee.org/document/9337216/ (visited on 05/01/2025).

[16] Onur Alparslan, Shin'ichi Arakawa, and Masayuki Murata. “Next Generation Intra-Vehicle Backbone Network Architectures”. In: *2021 IEEE 22nd International Conference on High Performance Switching and Routing (HPSR)*. 2021, pp. 1–7.

[17] Jun Huang et al. “In-Vehicle Networking: Protocols, Challenges, and Solutions”. In: *IEEE Network* 33.1 (Jan. 2019), pp. 92–98. ISSN: 0890-8044, 1558-156X. DOI: 10.1109/MNET.2018.1700448. URL: https://ieeexplore.ieee.org/document/8315204/ (visited on 08/21/2025).

[18] Uwe Kiencke and Timo Kytölä. “CAN, a ten years' anniversarial review”. In: *3th international CAN Conference, ICC*. Vol. 96. 1996, pp. 2–2.

[19] *History of CAN technology*. URL: https://www.can-cia.org/can-knowledge/history-of-can-technology (visited on 05/18/2025).

[20] T. Ziermann, S. Wildermann, and J. Teich. “CAN+: A new backward-compatible Controller Area Network (CAN) protocol with up to 16x higher data rates.” In: *2009 Design, Automation & Test in Europe Conference & Exhibition*. 2009 Design, Automation & Test in Europe Conference & Exhibition (DATE'09). Nice: IEEE, Apr. 2009, pp. 1088–1093.

[21] Aldin Berisa et al. “Comparative Evaluation of Various Generations of Controller Area Network Based on Timing Analysis”. In: *2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA)*. 2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA). Sinaia, Romania: IEEE, Sept. 12, 2023, pp. 1–8. URL: https://ieeexplore.ieee.org/document/10275549/ (visited on 05/18/2025).

[22] Florian Hartwich. “Bit time requirements for can FD”. In: *Proceedings of the 14th International CAN Conference*. 2013.

[23] Robert Bosch GmbH. *CAN with Flexible Data-Rate Specification Version 1.0*.

[24] Can in Automation (CiA). *CAN-XL CAN Newsletter 1/2025 31*.

[25] Karsten Schanze and Volkswagen AG. *Future of CAN from the prospective of an OEM*. 2020.

[26] Jo Laufenberg, Thomas Kropf, and Oliver Bringmann. “CAN Simulation Framework - From Classic CAN to CAN XL”. In: *2023 IEEE 26th International Conference on Intelligent Transportation Systems (ITSC)*. 2023 IEEE 26th International Conference on Intelligent Transportation Systems (ITSC). Bilbao, Spain: IEEE, Sept. 24, 2023, pp. 3343–3348. (Visited on 05/21/2025).

[27] ISO 11898-1:2024. *Road vehicles — Controller area network (CAN) — Part 1: Data link layer and physical coding sublayer*. 2024.

[28] ISO 11898-1:2015. *Road vehicles — Controller area network (CAN) — Part 1: Data link layer and physical signalling*. 2015.

[29] ISO/IEC 7498-1. *Information technology - Open Systems Interconnection - Basic Reference Model: The Basic Model*.

[30] ISO 11898-2:2024. *Road vehicles — Controller area network (CAN) — Part 2: High-speed physical medium attachment (PMA) sublayer*.

[31] Texas Instruments Steve Corrigan. *Introduction to Controller Area Network (CAN)*.

[32] *AUTOSAR Layered Software Architecture R24-11*.

[33] *A general Introduction to the AUTOSAR organisation (Part1)*.

[34] *AUTOSAR_EXP_Introduction_Part1*. URL: https://www.autosar.org/fileadmin/user_upload/AUTOSAR_Introduction_PDF/AUTOSAR_EXP_Introduction_Part1.pdf.

[35] *Foundation Release Overview AUTOSAR FO R24-11*.

[36] *AUTOSAR Complex Driver design and integration guideline CP R24-11*.

[37] Andrea Szalavetz. "In-house software development for software-defined vehicles: major changes ahead in automotive value chains". In: *International Journal of Automotive Technology and Management* 24.4 (2024), p. 10065215.

[38] *AUTOSAR SW-C and System Modeling Guide R24-11*.

[39] *ARXML Serialization Rules AUTOSAR FO R24-11*.

[40] Uwe Honekamp. "The Autosar XML Schema and Its Relevance for Autosar Tools". In: *IEEE Software* 26.4 (July 2009). Publisher: Institute of Electrical and Electronics Engineers (IEEE), pp. 73–76. ISSN: 0740-7459. DOI: 10.1109/ms.2009.104. URL: http://ieeexplore.ieee.org/document/5076463/ (visited on 05/01/2025).

[41] Kevin Neubauer. "Skalierbarkeit einer Szenarien-und Template-basierten Simulation von Elektrik/Elektronik-Architekturen in reaktiven Umgebungen". PhD thesis. Dissertation, Karlsruhe, Karlsruher Institut für Technologie (KIT), 2022, 2023.

[42] Stefan Voget. "AUTOSAR and the automotive tool chain". In: *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*. 2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010). Dresden: IEEE, Mar. 2010, pp. 259–262.

[43] Thomas Ringler, Christian Dziobek, and Florian Wohlgemuth. "Chancen und Herausforderungen bei der virtuellen Absicherung verteilter Body&Comfort-Funktionen auf Basis von AUTOSAR". In: *Tagungsband des Dagstuhl-Workshops*. Citeseer, p. 83.

[44] Vector Informatik GmbH. *MICROSAR Classic Fact Sheet*.

[45] Huang Bo et al. “Basic Concepts on AUTOSAR Development”. In: *2010 International Conference on Intelligent Computation Technology and Automation*. 2010 International Conference on Intelligent Computation Technology and Automation (ICICTA). Changsha, China: IEEE, May 2010, pp. 871–873.

[46] Thomas Arts et al. “Testing AUTOSAR software with QuickCheck”. In: *2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. 2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW). Graz, Austria: IEEE, Apr. 2015, pp. 1–4. (Visited on 05/02/2025).

[47] Mohamed Ali Shajahan et al. “AUTOSAR Classic vs. AUTOSAR Adaptive: A Comparative Analysis in Stack Development”. In: *Engineering International* 7.2 (Dec. 31, 2019), pp. 161–178. ISSN: 2409-3629. (Visited on 05/02/2025).

[48] Th. M. Galla and R. Pallierer. “AUTOSAR – challenges and solutions from a software vendor’s perspective”. In: *e & i Elektrotechnik und Informationstechnik* 128.6 (June 2011), pp. 234–239. ISSN: 0932-383X, 1613-7620. DOI: 10.1007/s00502-011-0006-8. URL: http://link.springer.com/10.1007/s00502-011-0006-8 (visited on 05/02/2025).

[49] Nico Naumann. “AUTOSAR Runtime Environment and Virtual Function Bus”. In: 2009.

[50] AUTOSAR. *Specification of PDU Router AUTOSAR CP R24-11*.

[51] AUTOSAR. *Specification of PDU Router AUTOSAR CP Release 4.4.0*.

[52] AUTOSAR. *Specification of Linklayer Sdu Routing Module AUTOSAR CP R24-11*. 2024.

[53] NXP. *S32G2 - LOW LATENCY COMMUNICATION ENGINE: LLCE*. 2021.

[54] Illia Safiulin. *Elektrobit Webinar Achieving hardware-accelerated communication in software defined vehicles*. 2024. URL: https://www.elektrobit.com/tech-corner/achieving-hardware-accelerated-communication-in-software-defined-vehicles/.

[55] Infineon. *DRE Data Routing Engine*. 2024.

[56] Murat Akpinar, Klaus Werner Schmidt, and Ece Guran Schmidt. “Improved Clock Synchronization Algorithms for the Controller Area Network (CAN)”. In: *2019 28th International Conference on Computer Communication and Networks (ICCCN)*. 2019 28th International Conference on Computer Communication and Networks (ICCCN). Valencia, Spain: IEEE, July 2019, pp. 1–8. (Visited on 06/12/2025).

[57] Stuart Robb. *CAN Bit Timing Requirements AN1798*.

[58] Meenanath Taralkar. *Computation of CAN Bit Timing Parameters Simplified*.

[59] AMD. *CAN v5.1 LogiCORE IP Product Guide*. 2023.

[60] Oliver Sander et al. “Reducing latency times by accelerated routing mechanisms for an FPGA gateway in the automotive domain”. In: *2008 International Conference on Field-Programmable Technology*. 2008 International Conference on Field-Programmable Technology. Dec. 2008, pp. 97–104. (Visited on 02/09/2025).

[61] Oliver Sander. *Skalierbare adaptive System-on-Chip-Architekturen für Inter-Car und Intra-Car Kommunikationsgateways*. Section: 377. KIT Scientific Publishing, 2011. DOI: 10.5445/KSP/1000021079.

[62] Matthias Traub. *Durchgängige Timing-Bewertung von Vernetzungsarchitekturen und Gateway-Systemen im Kraftfahrzeug*.

[63] Trong-Yen Lee, Chia-Wei Kuo, and I-An Lin. “High performance CAN/FlexRay gateway design for in-vehicle network”. In: *2017 IEEE Conference on Dependable and Secure Computing*. 2017, pp. 240–242.

[64] Shanker Shreejith et al. “VEGa: A High Performance Vehicular Ethernet Gateway on Hybrid FPGA”. In: *IEEE Transactions on Computers* 66.10 (Oct. 1, 2017), pp. 1790–1803.

[65] Guoqi Xie et al. “WCRT Analysis and Evaluation for Sporadic Message-Processing Tasks in Multicore Automotive Gateways”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* PP (Mar. 2018), pp. 1–1.

[66] ChangYoung Jo, JaeWan Park, and JaeWook Jeon. “Multi-Core Gateway Architecture and Scheduling Algorithm for High-Performance Gateway Implementation”. In: *2020 IEEE International Conference on Consumer Electronics - Asia (ICCE-Asia)*. 2020 IEEE International Conference on Consumer Electronics - Asia (ICCE-Asia). Seoul, Korea (South): IEEE, Nov. 1, 2020, pp. 1–6. (Visited on 08/09/2025).

[67] Abdoul Aziz Kane et al. “Elastic gateway functional safety architecture and deployment: a case study”. In: *IEEE Access* 10 (2022). Publisher: IEEE, pp. 91771–91801.

[68] Angela Gonzalez Marino, Francesc Fons, and Juan Manuel Moreno Arostegui. “Elastic gateway SoC proof of concept: Experiments design and performance evaluation”. In: *Vehicular Communications* 43 (2023). Publisher: Elsevier, p. 100636.

[69] Angela González Mariño, Francesc Fons, and Juan Manuel Moreno Arostegui. “Elastic Gateway SoC design: A HW-centric architecture for inline In-Vehicle Network processing”. In: *Vehicular Communications* 45 (2024). Publisher: Elsevier, p. 100721.

[70] Vector. *Making Full Use of Multi-Core ECUs with AUTOSAR Basic Software Distribution*. 2018.

[71] Vector Informatik GmbH. “Making Full Use of Multi-Core ECUs with AUTOSAR Basic Software Distribution”. Webinar, 2018.

[72] Florin Radu Isidor. “Multicore BSW distribution”. MICROSAR and DaVinci User Days, 2022.

[73] Francisco Fons and Mariano Fons. "FPGA-based automotive ECU design addresses AUTOSAR and ISO 26262 standards". In: *Xcell journal* 78 (2012), p. 20.

[74] ARM Limited. *AMBA AXI Protocol Specification ARM IHI 0022*. 2023.

[75] E. G. Schmidt et al. "Performance evaluation of FlexRay/CAN networks interconnected by a gateway". In: *International Symposium on Industrial Embedded System (SIES)*. 2010, pp. 209–212.

[76] Jin Ho Kim et al. "Gateway Framework for In-Vehicle Networks Based on CAN, FlexRay, and Ethernet". In: *IEEE Transactions on Vehicular Technology* 64.10 (2015), pp. 4472–4486. DOI: 10.1109/TVT.2014.2371470.

[77] Seung-Han Kim et al. "A gateway system for an automotive system: LIN, CAN, and FlexRay". In: *2008 6th IEEE International Conference on Industrial Informatics*. 2008, pp. 967–972.

[78] Katsumi Ebina, Rei Ueno, and Naofumi Homma. "Side-Channel Analysis Against SecOC-Compliant AES-CMAC". In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 70.10 (2023), pp. 3772–3776. DOI: 10.1109/TCSII.2023.3288278.

List of Tables

List of Figures