TECHNISCHE UNIVERSITÄT
CHEMNITZ

# Concept and Implementation of AUTOSAR compliant Automotive Ethernet stack on Infineon Aurix Tricore board

Completed at Bertrandt Ingenieurbüro GmbH, Ingolstadt

## Master Thesis


Author:
Sreenath Krishnadas
Matric.Nr. 368366

Supervisor:
Prof. Dr. Wolfram Hardt
Mr. Markus Gerngroß


A Master Thesis Report submitted in fulfilment of the requirements
for the degree of M.Sc. in Automotive Software Engineering


Faculty of Computer Science
Department of Computer Engineering


July 5, 2016

# Declaration of Authorship

I, Sreenath Krishnadas, declare that this Master thesis titled, 'Concept and Implementation of AUTOSAR compliant Automotive Ethernet stack on Infineon Aurix Tricore board' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this internship has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this Master thesis is entirely my own work.
- I have acknowledged all main sources of help.

Signed: _____

Date: _____

# *Abstract*

Automotive Ethernet is a newly introduced in-vehicle bus that allows unicast communication between ECUs. It is based on the OSI model of Ethernet, with a few modifications on the physical layer and newly introduced application protocols. AUTOSAR, a consortium of automotive OEMs, Tier-1 suppliers and tool vendors has defined a standard software architecture that simplifies the ECU software development with its well defined software specifications and APIs. The Automotive Ethernet stack is now an integral part of the latest AUTOSAR specification release 4.2. Infineon Aurix TriCore TC27x microcontroller is a popular board used in ADAS applications. The board has support for Fast Ethernet.

This thesis investigates the setting up of an Ethernet communication on the TriCore board running under AUTOSAR software architecture. The various modules of the AUTOSAR Ethernet stack are familiarized and configured. This is followed up by validating the implementation on the Ethernet physical layer. The validation is based on a real Ethernet communication between the TriCore board and the Vector VN5610 network interface card. TCP and UDP based connections between the AUTOSAR compliant board and the VN5610 are tested and validated.

A test suite for evaluating the protocol conformance of the AUTOSAR Ethernet stack exists at Bertrandt. The final step of this thesis involved the execution strategies for this test suite.

# *Acknowledgements*

Firstly I would like to thank the Almighty Lord for giving me the strength to complete this thesis successfully and also to my parents for their constant support and inspiration without which this would have not been possible.

I now take this opportunity to express my sincere gratitude to Prof. Dr. Wolfram Hardt and all of the Department of Computer Engineering faculty members for their help and support. I would also like to express my sincere thanks to my industrial supervisor Mr. Markus Gerngroß, Bertrandt Ingenieurbüro GmbH, for providing me the wonderful opportunity along with the necessary facilities to execute this thesis work in his team. Besides my supervisor, I would like to thank Mr. Arun Nagraj a colleague at Bertrandt for his technical inputs. I am indebted to all my colleagues and friends, who have directly or indirectly contributed towards the successful completion of this thesis

# Contents

# 1 Introduction

## 1.1 Motivation

In recent years, the number of electronic control units (ECUs) in cars have increased by manifold. Electronic systems have gained more prominence with the recent advances in the fields of driver assistance systems. In the 90s, the number of ECUs used in a vehicle were limited to a few couple of devices. But the conventional modern vehicles come equipped with at-least more than 40 ECUs (European market)[KT15]. The premium luxury models even boast around 100 ECUs these days. The figure 1.1 shows the advances in the average number of ECUs in a car.

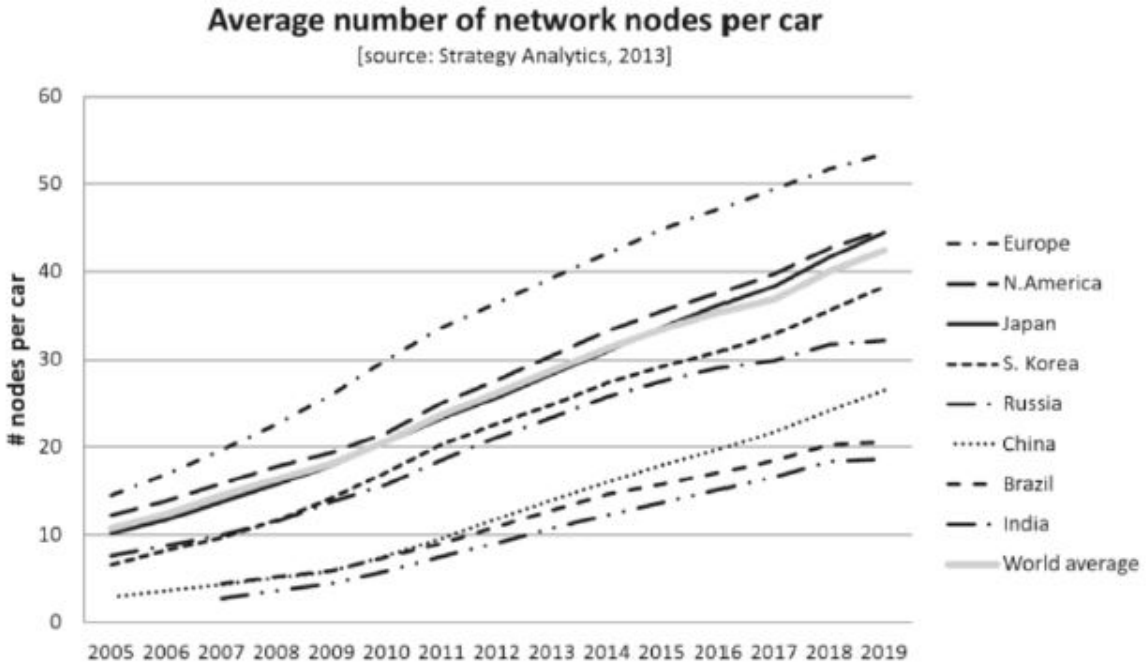



Figure 1.1: Increase in average number of networked ECUs per car [KT15]

ECUs communicate with each other by means of networking bus systems exclusive

to the automotive domain. The major bus sytems in use these days are the CAN, FLEXRAY, LIN and MOST networks. Figure 1.2 illustrates the in-vehicle bus systems and their application domain inside the cars. With exception to the MOST network, ECUs belonging to the other networks run under the standard AUTOSAR software.

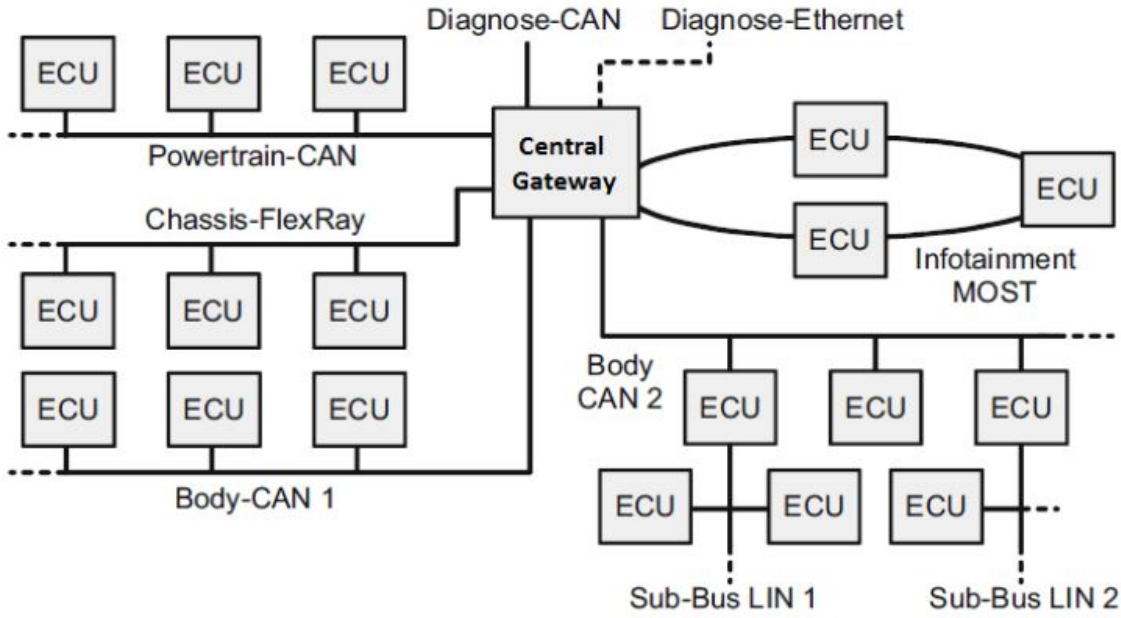



Figure 1.2: Overview of in-vehicle networks [WR14]

The latest innovations in ADAS technologies require very high bandwidths. The conventional in-vehicle networks cannot support these requirements. Moreover all these networks are based on a broadcast messaging mechanism. With these issues in mind, the commonly known Ethernet network is being integrated into the automotive domain. In order to meet the automotive grade requirements, minor adaptations were made to the Ethernet model. The new model came to be known as Automotive Ethernet.

## 1.2 Problem statement

The thesis executed at Bertrandt AG, is concerned with the implementation of the Automotive Ethernet stack with AUTOSAR support on the Infineon AURIX TriCore embedded target. An AUTOSAR version 4.2 Ethernet stack evaluation bundle from Vector was available at Bertrandt. This thesis tries to investigate the suitability of this software for the TriCore board.

The problem statement is categorized into the following aspects:

- Understand the AUTOSAR Ethernet communication stack and configure it suitably for the Infineon Aurix TriCore board
- Implement an Ethernet rest bus simulation on Test PC
- Establish and investigate the Ethernet communication on TriCore board by setting up an Ethernet link with the Test PC
- A basic test specification is available at Bertrandt AG for the AUTOSAR Ethernet stack. The final step of the thesis is concerned with the implementation of a test bench for executing this test specification. This may serve as the platform for evaluating the AUTOSAR Ethernet stack.

# 2 State of Art

As we have seen in the previous chapter, Ethernet as a major bus system is slowly getting integrated into the automotive domain. It cannot be seen as yet another bus network being introduced into cars, but it has the potential to drive the consolidation even further. Here it is also important to note that Automotive Ethernet will not be the only in-car bus system in the future. In this chapter a literature review of the current state of the art for Automotive Ethernet is presented.

## 2.1 Basics of In-Vehicle Networking

In-Vehicle networking systems have evolved over the years. Present day vehicles are built with several different bus systems. ECUs inside the car exchange messages between each other. For example, an ECU which contains sensors needs to send its data to another ECU responsible for controlling an actuator. Each bus system is utilized for a specific domain or purpose. These features could be execution of time-critical tasks, exchange of control signals or more recently the high-bandwidth data like audio/video signals from high resolution cameras used in ADAS sensor fusion.

The most common bus systems currently used in a vehicle are the following:

- Controller Area Network (CAN)
- Flexray
- Local Interconnect Network (LIN)
- Media Oriented Systems Transport (MOST)

A brief summary of properties of these bus protocols is given in Table 2.1. As already mentioned, each of these technologies have a fixed application domain in mind. In the following, a basic overview of these four bus systems is given.

Table 2.1: In-vehicle bus systems [Rei10]

| **Attribute** | **CAN** | **FLEXRAY** | **LIN** | **MOST** |
|---|---|---|---|---|
| **Transmission rate** | CAN-H: <125 Kbps<br>CAN-L: <1 Mbps | <10 Mbit/s | <20 Kbps | <150 Mbps |
| **Topology** | star | star | linear | ring |
| **Control mechanism** | Event | Time/event | Time | Time/event |
| **Wiring** | Copper twisted-pair | Copper twisted-pair | Optical fiber or copper | Copper single wire |
| **Application domain** | Engine management system,<br>body electronics | Driver assistance systems | Connection of sensors and actuators | Infotainment |

The Controller Area Network, introduced in a vehicle series in 1988, is the oldest bus system and still in use in every car. Its prevalence is due to the major requirement for robust ECU communication. On the other hand, CAN networks carry the disadvantage of is its low bandwidth rate, restricted to 1 Mbit/s. A single CAN frame can carry a maximum payload of 8 bytes. These two factors results in its inability to transmit large data payloads that are required to implement the latest advances in ADAS domain.

Development for Flexray was started in the early 2000s with the major goal of replacing the conventional mechanical functions by electrical functions, with the so called X-by-wire (e.g *Brake-by-Wire* or *Steer-by-Wire*). High requirements with respect to safety in these application fields also led to the introduction of this new bus protocol. Another improvement is the bandwidth rate (upto 10Mbit/s) compared to CAN. This protocol is implemented with Time Division Multiple Access (TDMA) principle, compared to the event triggered *Carrier Sense Multiple Access/Collision Resolution* (CSMA/CR) for CAN.

A drawback with the bus arbitration method of CAN was its non-deterministic behavior under certain circumstances. ECUs in CAN networks broadcast their messages. and messages with a higher priority always win the arbitration on the bus. Thus there could be a situation whereby, only the high priority messages can be transmitted with a maximal latency. Low-priority messages cannot be guaranteed to be transmitted at a specific time under a high bus load. This issue is solved in Flexray with synchronous TDMA with static time slots for each node. Flexray also incorporates event based

triggering with the use of dynamic slots in its frame format. Static slots are followed by dynamic slots.

The other conventional bus system is the Local Interconnect Network (LIN). This is a one-wire bus. It is cheaper alternative to CAN and is mainly used in smaller environments e.g. for interconnecting all sensors and actuators of a door or a seat (sub-systems). The main consideration for LIN was the cost factor and simplicity in its design, thus the transmission rate was less important at the time of its consideration.

The Media Oriented Systems Transport (MOST) has been designed for multimedia applications in 1998. Upto 64 ECUs can be connected with each other using a ring topology[COO]. The recent MOST150 standard supports transmission rates up to 150 Mbit/s. Nearly all the major car manufacturers use MOST inside their cars for the infotainment and telematics applications. Compared to CAN and Flexray, there are far less ECUs inside the vehicles using the MOST bus network. MOST is a proprietary technology invented by SMSC (now taken over by MICROCHIP) and currently maintained by the MOST cooperation (AUDI, BMW, SMSC, Harman & Becker). For this reason, it is a very expensive technology facing several issues in terms of interoperability and licensing. This reason also prevents MOST from being used for in-vehicle communication outside the infotainment network, irrespective of its high bandwidth capability.

Finally, Ethernet is the newest bus system to be introduced into the in-vehicle networking domain. Ethernet, though has been a sturdy communication network over the years as a comprehensive office and home network. It was a never choice due to its expensive cabling and strict EMC requirements for the automotive domain. Now though with increasing demands for higher bandwidth, and with limitations of existing bus systems with regards to in-vehicle gateway networks, suddenly Automotiv Ethernet has gained a lot of foothold. It was first introduced for diagnostics of ECUs with external tools and for ECU flashing purposes due to its high bandwidth. During the second generation (current phase), it has gained more presence in the ADAS domain as well as into in-car networking between ECUs. It is predicted that in the coming years, Automotive Ethernet becomes the de facto standard for in-vehicle networking whereby all the several application domains such as Powertrain, Chassis, Multimedia,

ADAS are interconnected via Ethernet cable [IXI].

## 2.2 OSI model from the Ethernet perspective

To establish communications across a network several models have been prescribed over the years. One of the most prominent among them is the Open Systems Interconnection model(OSI model). It was originally conceived by the International Organisation for Standardization (ISO) in the early 1980s [Wikb] . It is a conceptual model that explains the communication between two systems connected over a network. The entire communication is abstracted into seven different layers. Each layer has distinct functions and interfaces. Figure 2.1 shows the seven layers of the OSI model. All intermediate layers have interfaces for communication in either direction [Rei10]. Ethernet has a typical implementation of the OSI model and its implementation slightly varies when it comes to the automotive domain.

The lowermost layer is called the *Physical layer*. It defines all electrical, mechanical and functional parameters of the transmission link. Some typical parameters include the wiring (electrical, optical or wireless) details, transmission rate, signal encoding/decoding schemes or even the type of connectors (RJ45, DSUB-9) for the Ethernet interface on the ECU. The most well known implementation of this layer for Ethernet is 100Base-TX. Recently the automotive industry adopted the BroadR-Reach standard as the physical layer for in-vehicle use cases.

The second layer is commonly known as the *Data Link Layer*. It is responsible for implementation of the Medium Access Control (MAC) technique for handling the bus arbitration of the several devices on the network. Error correction/detection techniques are also applicable in the Data Link Layer. Popular error detection methods include the use of a Cyclic Redundancy Check (CRC) check for payload data. When an error is detected, the data link layer could signal the bus about an error, in which case, there would be a retransmission of the data from the sender. The most well known implementation of this layer is the Ethernet MAC.

The third layer is the *Network Layer*. It handles routing of packets between different networks. Internet Protocol (IP) is a common implementation of this layer. Nowadays the IP layer is available in two different implementations, ipV4 and ipV6. These

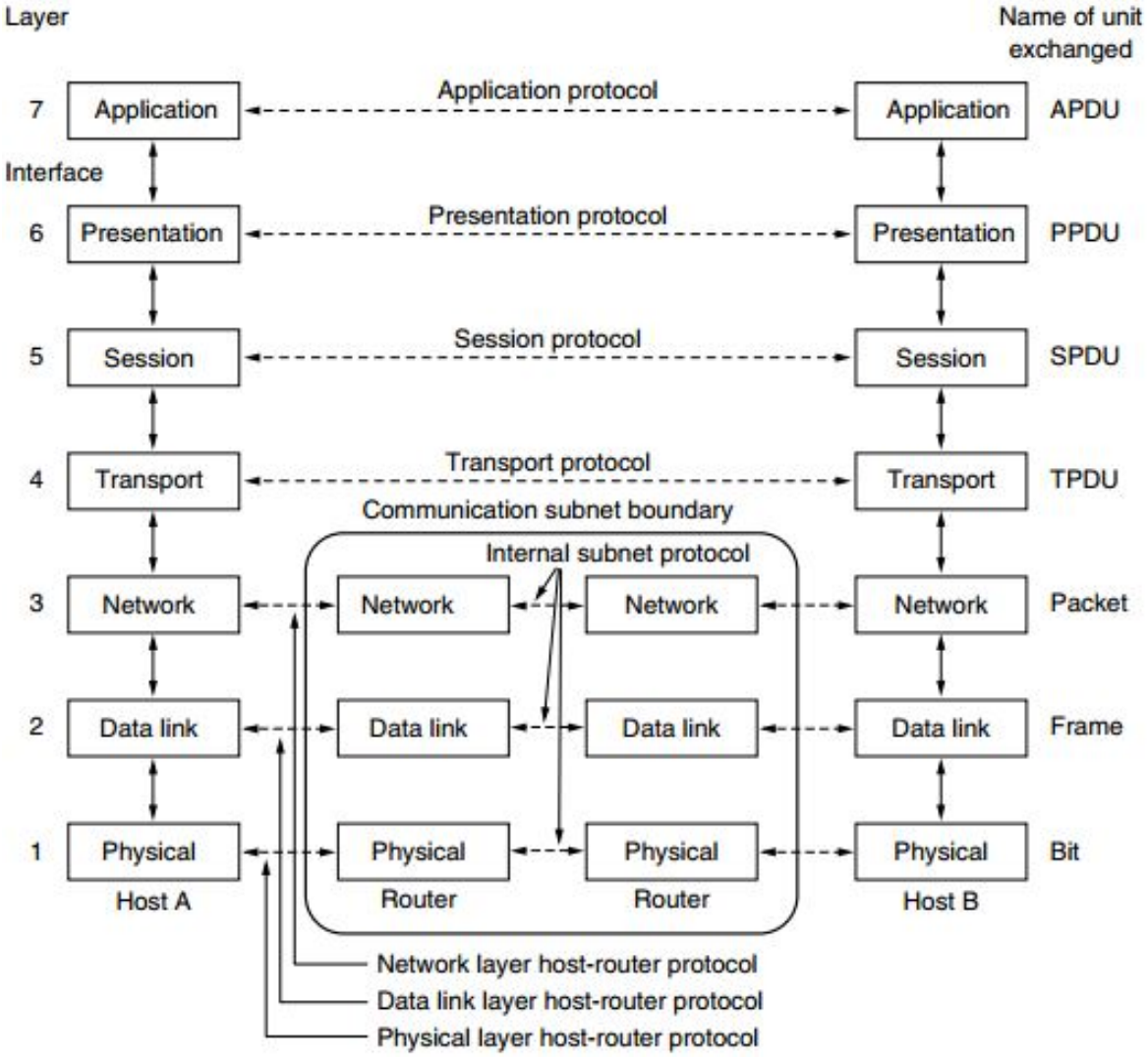



Figure 2.1: OSI model [AD10]

variants are distinguished by the size of the network.

The fourth layer is called the *Transport Layer*. Application unaware of the payload capability of the underlying bus, may send down excess data. The Transport layer is responsible for splitting the data into smaller packets that fit the bus capability while also managing the sequence order of the packets during its transmission/reception on the physical layer. The most well known implementations of this layer are TCP and UDP.

The upper layers (layers five to seven) are always combined together as they are very

closely interconnected with each other and as it is difficult to distinguish them in complex system implementations. They are identified as the *Session layer*, *Presentation Layer*, and the *Application Layer* respectively. Generally these are summarized together as the *Application Layer*. Even though the most frequent implementations of this layer are HTTP , FTP and SMTP, their relevance in the automotive domain is negligible. Instead tailor made applications for the automotive use cases have gained relevance. These include SOME/IP, DOIP, XCP or the typical AUTOSAR SWCs [VEC].

The OSI layer model is distributed across the hardware-software boundary. Except physical layer, all the other OSI layers are implemented in software. The data link layer usually resides on the Ethernet MAC controller. Each software layer is called as a protocol. A collection of such different protocols is known as the *Protocol Stack*. The term *Ethernet Stack* shall refer to a particular protocol stack in this thesis.

During transmission, the payload passes from the application layer, to the physical layer, undergoing a series of transformations at each layer enroute. During reception, the opposite happens, with each layer stripping away header data etc, before the actual payload is available to the application layer.

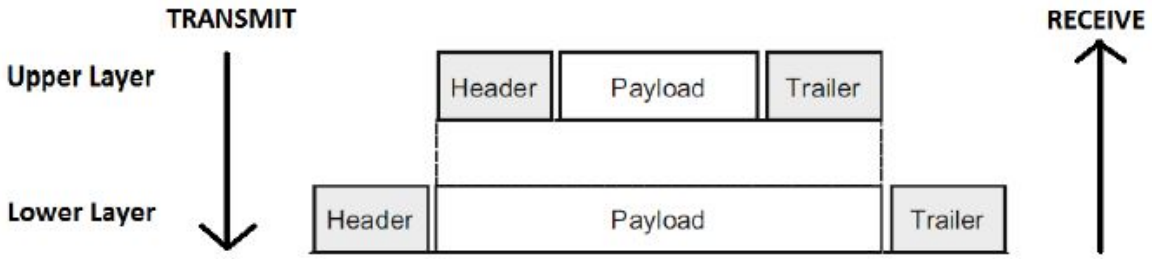



Figure 2.2: Data exchange across the OSI layers [WR14]

An application layer payload is referred to as a Service Data Unit (SDU). When it passes onto the layer below it, a header and an optional trailer data is added to the SDU. This frame is collectively referred to as a Protocol Data Unit (PDU). For each layer, the PDU received from the upper layer is the SDU. Thus every layer adds its own specific header and trailer to the frame. Similarly during reception, each layer removes the corresponding header and trailer and the SDU is passed onto the upper layer. This concept is illustrated in the figure 2.2.

## 2.3 Ethernet frame layout

In order to apply the knowledge from last section for Ethernet, this section shall elaborate on Ethernet and the format of an Ethernet frame. Ethernet itself only defines layers 1 and 2 of the OSI model. It is the most popular in LAN networks with 90% [Kiz14] share in the market. It was introduced in 1983 as IEEE standard 802.3 and is continuously enhanced ever since, Although Ethernet describes the physical Layer, it does not strictly specify the exact wiring schemes and the transmission rates of the hardware used. This is because it describes only the Media-Independent Interface (MII). As a result of this, there are many technologies available with different data rates and transmission mediums like twisted-pair or fiber optic cables. It is considered that there are 24 different standards available today. The next section will give an overview of two technologies for physical transmission (layer 1) which are used in the automotive domain. Despite that, the Ethernet protocol defines a common channel coding method, media access control technique (CSMA/CD) and frame-based serial transmission. The frame format according to IEEE 802.3 is displayed in the figure 2.3. In this definition, an "Ethernet 2" frame is used; "Ethernet 1" uses the length of the entire frame instead of the payload used in the upper layer.

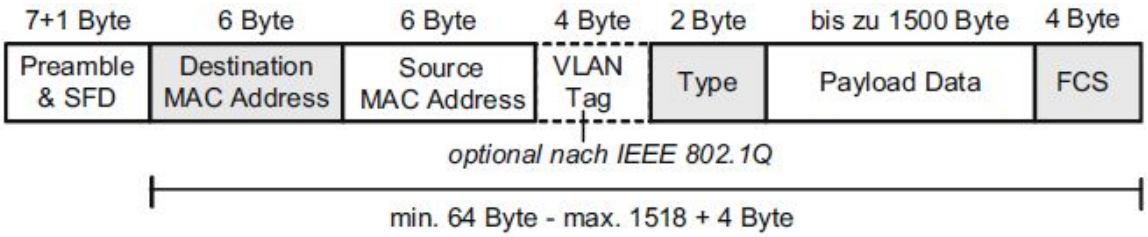



Figure 2.3: Ethernet Frame Format [WR14]

As explained in the previous section, the upper layer PDU's are wrapped inside the data field of an Ethernet frame. A short overview of the various fields of an Ethernet frame are explained below.

- **Preamble & Start of Frame delimiter (SFD)**: It consists of 7 bytes and every byte has the value 0x55. It is used for synchronization purposes and marks the start of a frame.
- **Destination MAC address**: It consists of 6 bytes and contains the MAC address of the recipient.

- **Source MAC address**: It consists of 6 bytes and contains the MAC address of the sender.
- **VLAN Tag**: It is a 4 byte field and is used to distinguish the VLAN frames from the normal frames. The first 16 bits of the VLAN tag carries a standard value of 0x8100 which labels it as a VLAN frame. Of the remaining 16 bits, the first 3 bits represent the frame priority, the next 1 bit represent if the frame could be dropped or not. The last 12 bits carry a unique ID to represent the VLAN network it belongs to.
- **Type**: It consists of 2 bytes and contains the type of protocol used in layer 3.
- **Payload Data**: It consists of a minimum of 46 and a maximum of 1500 bytes and contains the payload data (PDU of layer 3)
- **FCS**: It consists of 4 bytes and contains the CRC-32 checksum of the header ad payload.

## 2.4 Physical Layer

In the automotive domain, the physical layer standard has undergone several iterations over the years. These include the 100BASE-TX, BroadR-Reach (OABR) and the Gigabit 1000BASE-T wiring schemes [KT15]. Ethernet as a vehicle communication network was first adopted for diagnostic purposes with external devices. It used the standard offce cabling schemes of 100BASE-TX. But these physical layer technology could not be used for in-vehicle networking due to the environmental restrictions it posed. To tackle this, the Open Alliance SIG consortium of companies consisting of BMW, Broadcom, Freescale, Harman, Hyundai, NXP and STMicroelectronics came up with the solution of broadR-reach wiring scheme [All]. Both 100BASE-TX and OABR have a transmission rate of 100 MBit/s. A general comparison of these protocol properties is given in Table 2.2.

The major difference between these two protocols is the amount of twisted-pair wires inside a cable. In Figure 2.4 the section of an Ethernet cable as used for 100BASE-TX is shown. This technology uses only two out of four pairs for transmission and reception though. Each pair can only transmit data in one direction, so two pairs are needed for duplex communication. This is called half-duplex transmission mode. Full-duplex transmission in comparison can transmit data in both directions on the same medium.

Thus only one twisted-pair is sufficient. This is the idea behind OABR. Furthermore, the cable does not need to be shielded anymore. This results in a serious reduction of cabling weight. Broadcom estimates the reduction in cabling weight by 30% whereas the reduction of connectivity cost shall go down by 80% [Broa]. As the complexity rises with OABR, the signals must be better de-correlated. For this purpose, the DSP in the connector has to use an optimized scrambler compared to 100BASE-TX. Another improvement is the bandwidth efficiency. Whereas 100BASE-TX has 65 MHz bandwidth in total (both twisted-pairs), OABR only has 33.3 MHz which is only half the original bandwidth. As a result, crosstalk and loss are reduced. OABR also has the feature of echo cancellation compared to 100BASE-TX. The most important development criteria for OABR was the fulfillment of the strict Electromagnetic compatibility (EMC) criteria for vehicles though. BroadR-Reach starts to find its way into vehicles with the development of Automotive Ethernet and has already been implemented in the 2014 series of the BMW X5 [Blo].

Table 2.2: Comparison of Automotive Ethernet physical layers [EFl15]

| **Attrribute** | **100BASE-TX** | **BroadR-Reach** |
|---|---|---|
| **Standard** | IEEE 802.3 Clause 25 | OPEN Alliance IEEE802.3 |
| **Encoding** | 4B5B, MLT-3, half-duplex, two-shielded twisted pair | 4B3B, PAM-3, full-duplex, one unshielded twisted pair |
| **Signal level** | three levels | three levels |
| **Transmission Bandwidth** | 62.5MHz | 33.3MHz |

## 2.5 TCP/IP

This refers to the combined layers 3 and 4 of the Ethernet OSI model. Similar to the normal LAN applications, the TCP/IP suite is used on top of the Ethernet physical layer for the various automotive use cases. The various protocols of the TCP/IP suite is explained below.

### 2.5.1 Internet Protocol (IP)

The Internet Protocol is responsible for routing between different networks. Whereas it is possible to send Ethernet frames in a certain network via a switch, packets have to pass a router when the destination host lies in a remote network. For this purpose, every network node is assigned a distinct IP address which is independent from the

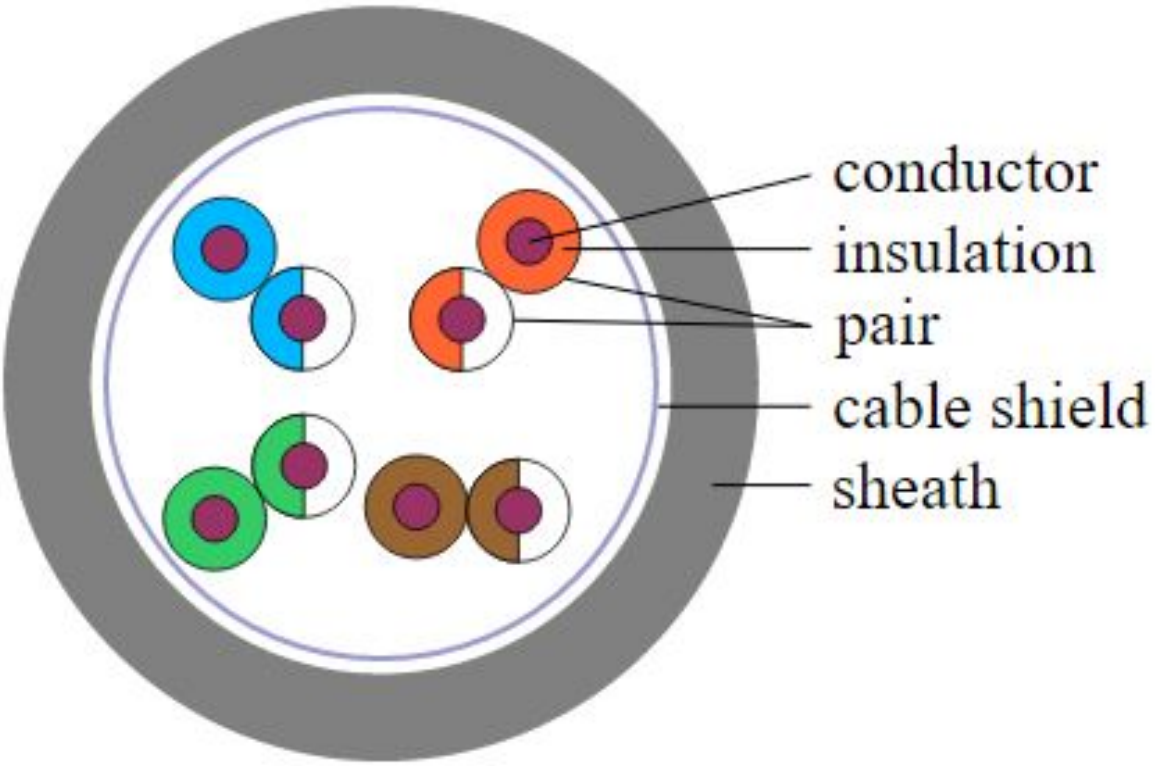



Figure 2.4: Section of an Ethernet cable [Com]

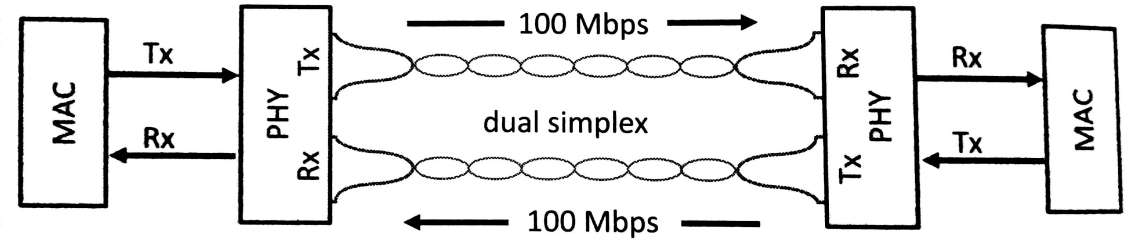



Figure 2.5: Key elements of the 100Mbps over 100BASE-TX [KT15]

physical address (MAC address) and thus acts as a logical address. All nodes in a network share a subnet mask which is important to calculate the network address. A logical AND operation of a random IP address in that network and the subnet mask results in the network address. As a result, all nodes belonging to a network can be identified. When a packet is sent to a host not lying in the same network, the router can look up the network this node lies in and can thus route the packet to the correct network. For automotive use, routing is not used today as all ECUs supporting Ethernet lie in a switched network. Anyway every ECU must possess a distinct IP

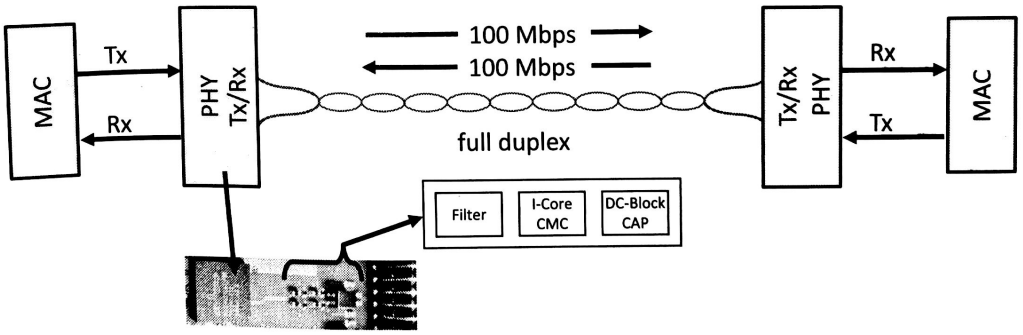



Figure 2.6: Key elements of an OABR true full duplex network link [KT15]

address as it is a vital parameter in the layer 3 header (IP header). At this point, two important protocols concerning the Internet Protocol shall be introduced: ARP and ICMP. The Address Resolution Protocol (ARP) allows a host to identify the MAC address of a recipient when its IP address is known. The Internet Control Message Protocol (ICMP) contains error information about packets which are discarded, e.g. because a host is not reachable or the Maximum Transmission Unit of a link is smaller than the frame size. Nowadays, two versions of the Internet Protocol are available: IPv4 and IPv6. The major difference between them is, that IPv4 addresses are 32 bit long compared to 128 bit for IPv6. Again, in the automotive industry, only IPv4 addresses are important, as with IPv4, $2^{32} = 4.294.976.296$ ECUs can be addressed and this is far more than the amount of ECUs which will ever be used inside vehicles. The IPv4 header is 20 bytes long and is illustrated in Figure 2.7.

### 2.5.2 Transmission Control Protocol (TCP)

On layer 4, there are two major transport protocols. Their main goal is to open a socket connection between two hosts determined by two tuples of an IP address and a port number. The more commonly used transport protocol is called TCP. Its main functions include the following [CKo]:

- Data transfer
- Connection Establishment, Management and Termination
- Prioritization of connections
- Flow Control

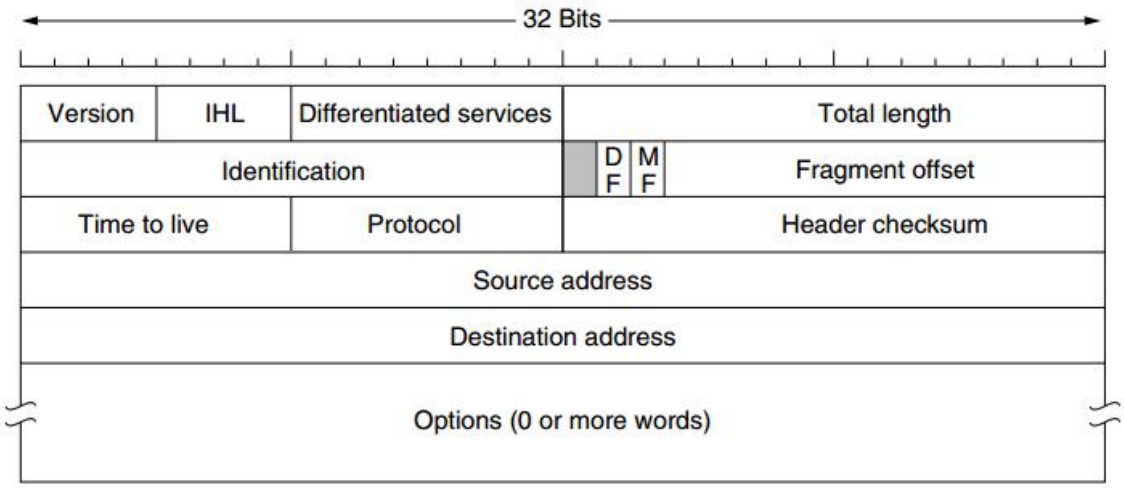



Figure 2.7: IPv4 Header [AD10]

- Congestion Avoidance

A TCP connection is established by a three-way-handshake. After the connection has been established, data can be transmitted between the two nodes over a point-to-point connection. As a result, this packet switching method is connection-oriented. Furthermore, TCP acknowledges all successfully sent segments. It also contains checksums and timers to detect errors during transmission. It retransmits segments which are not transmitted correctly. Segments are then put together at the recipient. For this purpose, TCP uses segment numbers. As a result of the retransmission, there is an increase in network traffic. This can be controlled by the flow control and congestion avoidance functionalities of TCP. The socket can be closed anytime after data does not need to be transmitted anymore. The TCP header is 40 bytes long and displayed in Figure 2.8

### 2.5.3 User Datagram Protocol (UDP)

UDP is another transport protocol, but much simpler than TCP as it is connectionless and does not guarantee the correct transmission of segments. This is a major disadvantage as a packet is either sent or discarded. But an advantage of UDP is the ability of multicasts and broadcasts. With TCP, several point-to-point sockets would have to be opened first which is time and resource consuming. Furthermore, the header size is only 8 bytes and thus there is little overhead. The header is shown in Figure 2.9

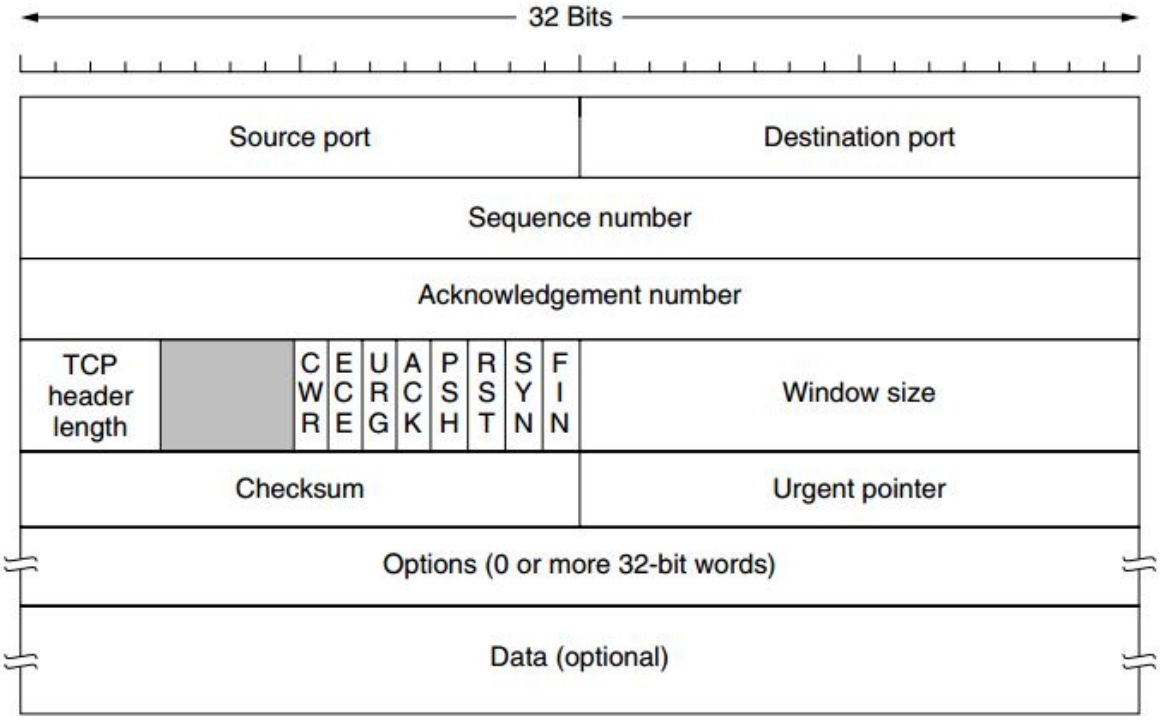



Figure 2.8: TCP Header [AD10]

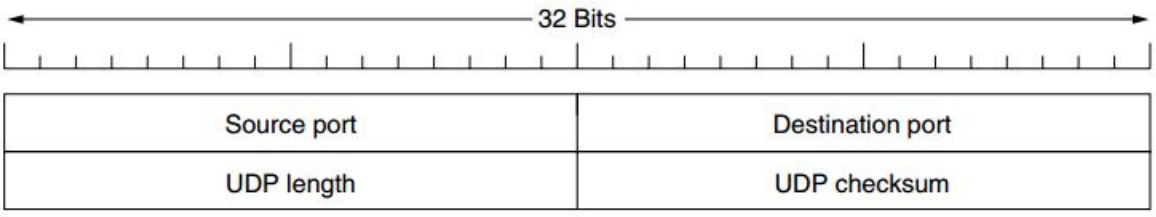



Figure 2.9: UDP Header [AD10]

# 3 Background

This chapter introduces the reader to the fundamental concepts of AUTOSAR, with a detailed explanation of its software architecture. Furthermore the rationale behind the chosen software framework for the AUTOSAR Ethernet stack and its tool support for this thesis is given. The last part of this chapter gives an overview of the microcontroller board used in this thesis work.

## 3.1 AUTOSAR

A timeline capturing the feature-centric evolution of cars is shown in figure 3.1. With new concepts being introduced on a car on a regular basis, the complexity of their implementation has also increased and the E/E architectures also demand constant revision in order to realize these features. OEMs rely on a lot of tool vendors and service-based companies that can offer software solutions for realizing these features. The various ECUs on board a vehicle exchange data but the software architecture in the ECUs depended on the supplier offering the solutions. Thus there did not exist a certain standard for the implementation of the same. Main issues faced by many of the OEMs were as follows:

- Compatibility of the interfaces
- The standards and architecture followed by these vendors were different
- Re-usability of the software
- Increasing complexity of the software
- Licensing and IP related issues

To overcome all these issues, OEMs, Tier-1 suppliers and tool vendors jointly set up a technical team in 2002. In July 2003 a new partnership was signed between them

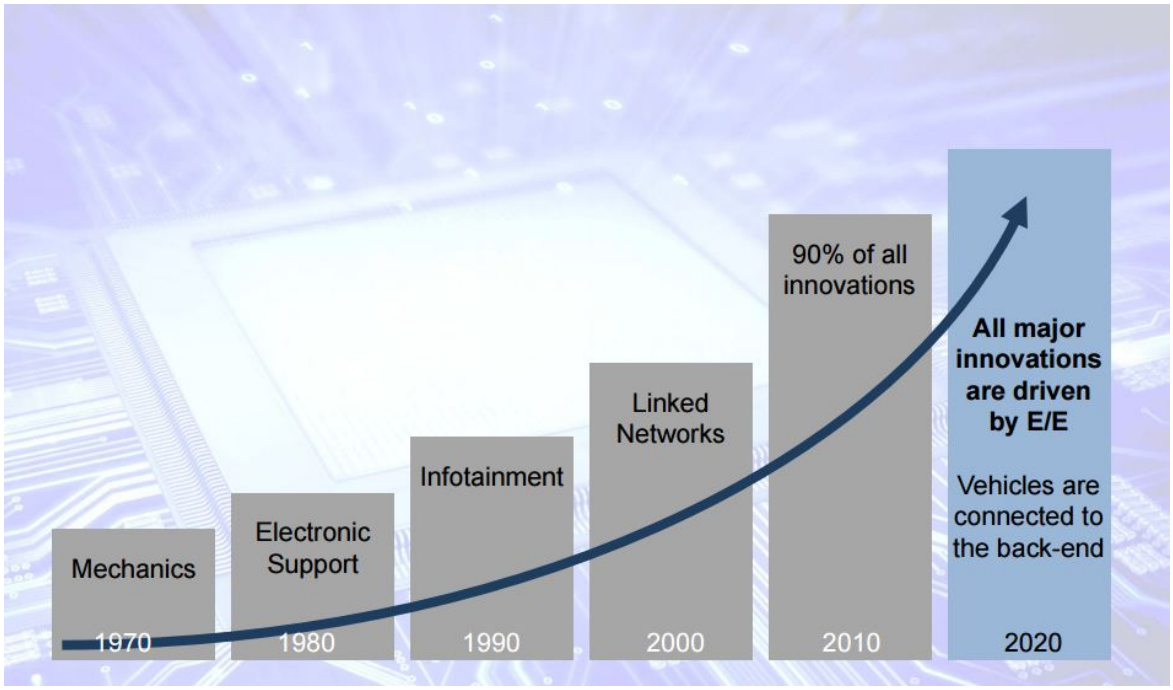



Figure 3.1: Increasing complexity in vehicles [Für15]

and their solution was to adapt a new open standard architecture. This led to the rise of the Automotive Open System architecture (AUTOSAR). This standard was responsible for a basic infrastructure for the management of future applications and standard software modules to be deployed inside a car.

"Co-operate on standards – compete on Implementation" [AUTc]

AUTOSAR is a global consortium that brings together all the OEMs, automotive ECU suppliers and the tool vendors so that they can create work on a standard architecture, application interfaces and a methodology which the automobile industry can follow. The AUTOSAR group specifies a three tier structure for development and each tier has to follow the specified rights and duties. The top tier which is also known as core partners consists of the OEMs and the suppliers who were part of the original founding team. These companies control and organise the AUTOSAR development partnership. Working alongside the core partners are the premium, development and associate partners. This group consists of the semiconductor vendors, service oriented suppliers. This group has the right to use AUTOSAR standards and Intellectual property (IP) for commercial purposes. These groups are also entitled to take forward the standards further and also organise the AUTOSAR projects [AUTa]

.

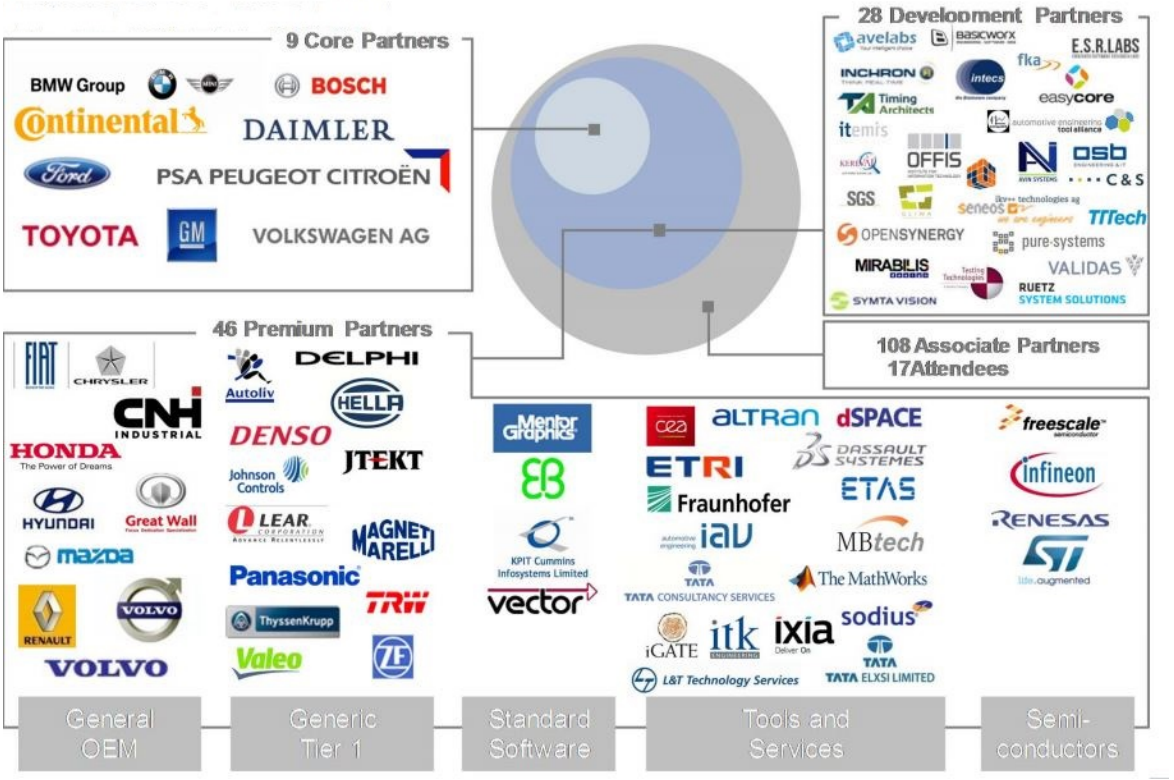



Figure 3.2: AUTOSAR partners [Für15]

The primary goal of the AUTOSAR development cooperation is the standardization of basic system functions and functional interfaces [AUTb]. AUTOSAR organisation not only looks at standardising the basic interface and methodologies but also encourages the information exchange between the partner companies. This co-operation leads to reusability of the existing software and reduces the expenditures for the update of the existing software and hardware over the life-time of the vehicle.

AUTOSAR also looks at reducing the overall complexity of the electronic hardware by providing a standardized layered architecture.

### 3.1.1 Phases in AUTOSAR standard development

AUTOSAR also looks at reducing the overall complexity of the electronic hardware by providing a standardized layered architecture. This architecture is not only limited to be used in automobile industry but also can be used in the automation in defence industry.

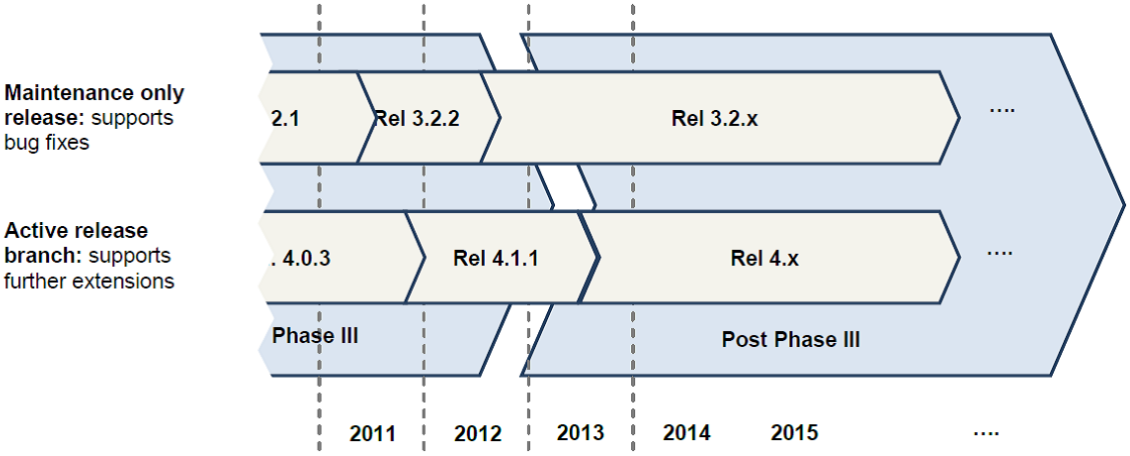



Figure 3.3: Phases in AUTOSAR Development[Sch]

After the formation of the AUTOSAR group, the first version of the AUTOSAR standard v1.0 was released in 2005. The phase 1 of the development which continued till 2006 had a basic development of the standards. Phase 2 of the release that continued till the year 2009 where new additions were made to the existing standards and the existing standards were revised after a careful review of the same. Phase 3 which concluded in 2013 went into a maintenance phase, maintaining the concepts of the architecture developed before and making selected improvements to the same [Sch]. The thesis is compliant with AUTOSAR release 4.2.1 version, the latest version available with Bertrandt during the start of the thesis.

With the advancement in technology, new ideas and concepts are being integrated into the automobile industry in order to increase safety and more comforts to the passengers. So keeping in view of these advancements, AUTOSAR group is releasing versions standardising these newer technologies while improving the quality of the standards for the old legacy systems.

For instance, to incorporate higher data transmission rates in the vehicle communication system, CANFD and Ethernet communication standards are included in the latest release(v4.2). To improve the processing speed in the ECUs the multicore core architecture OS standards are included in v4.0 release. New functional safety implementation, diagnostic and safety regulations are being developed and standardized by the AUTOSAR group as well as co-ordinating with the other standardization bodies

like Car-to-X or GENIVI to provide an infrastructure that supports these standards.

### 3.1.2 Main Features of AUTOSAR

The software in the non- AUTOSAR era didn't have a clear demarcation between the layers and hence it was difficult to re-use the software delivered. The interfaces were also not clearly defined and it became increasingly difficult to integrate these software components. It also took lot of effort to update the software which led to the wastage of time and money.

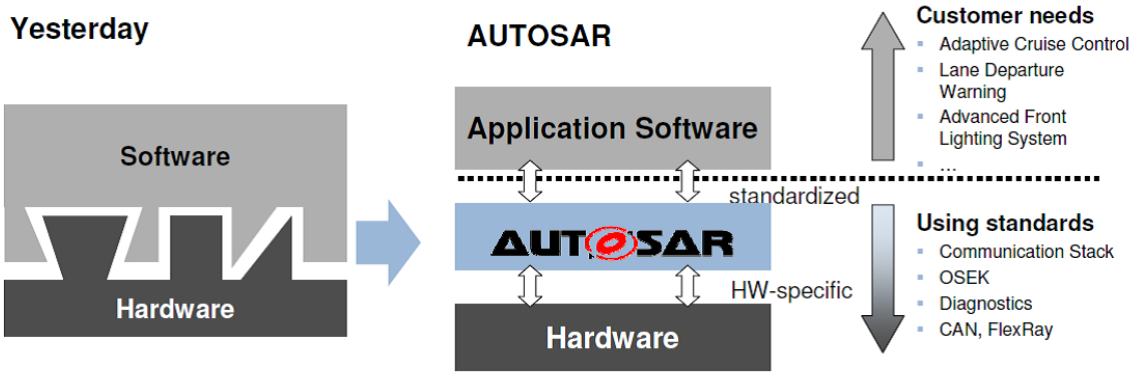



Figure 3.4: Comparison between AUTOSAR and Non- AUTOSAR model [Für15]

While developing the AUTOSAR concepts the main aim was to utilise the already proven concepts rather than start it from the scratch. It emphasises mainly on 4 points

- Modularity
- Scalability
- Transferability
- Re-usability

AUTOSAR defines a well-structured architecture with standard interfaces. It distinguishes the application layer from the hardware and provides an interface via the Virtual functional bus (VFB) through which it communicates with all the other layers. VFB is an abstract layer provided by the AUTOSAR with a standard interfaces that

provides a virtual integration before the implementation itself. The realization of the Virtual Functional bus is done by using the RTE (Run Time Environment) layer in AUTOSAR. This realization is ECU specific.

Application developers of AUTOSAR conformant software need not be aware of the under laying hardware. Well-defined interface is available to communicate with the other underlying layers. This also helps in increasing in modularity and scalability of the developed software. It also increase the portability of the application software, as it may be completely re-used in the other ECUs, and hence it promotes re use of the developed software with little or no modifications.

### 3.1.3 AUTOSAR Layered Architecture

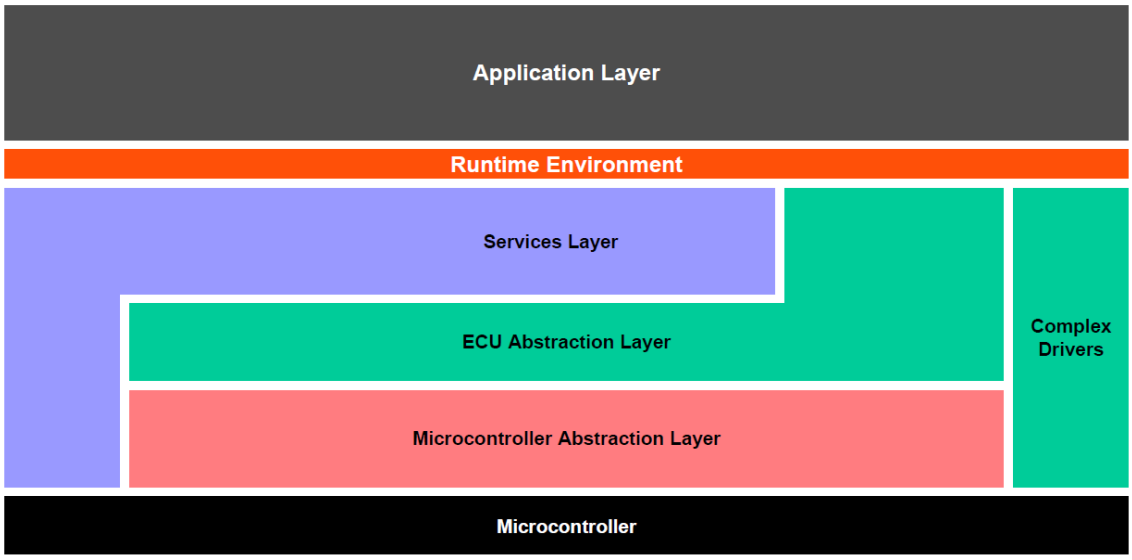



Figure 3.5: AUTOSAR Layered Architecture[AUTd]

Figure 3.5 shows the layered architecture of the AUTOSAR. The architecture is divided mainly into 4 main layers. The architecture provides a hardware abstraction and encapsulation for all the application functions. All the layers interact with one-another using a standard set of interfaces provided at all the layers.

#### Microcontroller layer

This is the lowest layer in the architecture. It is the hardware on which the entire AUTOSAR architectural layers are built upon. It contains the input / output peripherals,

communication peripherals, the memory interfaces and microcontroller chip itself that make up the ECU.

**Basic Software (BSW)**

Basic software is the layer that lies in-between micro-controller and the RTE layer in the AUTOSAR. It is the layer that realises the services and the communication protocols to be used by the application layer. It also consists of the operating system that schedules and controls the working of the ECU software. The Basic software can be broadly classified into three parts:

**Microcontroller Abstraction Layer (MCAL)**

MCAL provides the hardware abstraction for the upper layers in the AUTOSAR. This is the unit of the AUTOSAR architecture that is dependent upon the hardware and it directly interacts with the underlying hardware. All the higher layers in AUTOSAR interacts with the hardware using the standard interface provided by this layer, i.e., it prevents all the upper layers from directly accessing the hardware and hence encapsulates the underlying hardware. It consists of drivers for communication, input or output, storage interfaces in the microcontroller which provides services to the ECU abstraction layer.

**ECU Abstraction Layer (ECUAL)**

ECUAL is the layer that is sandwiched between MCAL and services layer. It is the hardware independent layer which provides interfaces to the drivers in the MCAL. It also contains drivers for the external peripherals connected to the system and also offers API calls regardless of the location on the microcontroller (external or internal). For e.g., certain memory devices may be available as external peripherals connected to the microcontroller via a SPI interface. It provides encapsulation to the underlying layers and makes the higher software layers independent of the hardware.

**Services Layer**

This is the highest layer in the BSW and provides access for the application layer to utilize the services using the standard interface. Apart from providing access to the encapsulated ECUAL it also provides services like Operating system functionality, Vehicle network communication and management, Diagnostic services, ECU state management and memory services.

#### Complex Driver

It is the layer in spanning from RTE layer to the microcontroller which does not utilize any services from the BSW layer. This layer becomes when it is required to implement the drivers that are not specified by AUTOSAR and also if the application layer has to directly access the ECU hardware for performance issues. For example dosing valve or injection valve for which drivers are built by supplying vendors themselves.

#### RTE Layer

RTE is the layer that provides interface for the software components in the application layer to communicate with one another. It not only maintains the communication between the two SWCs from the same ECUs but also between different ECUs by sending the message through the underlying communication stack. This RTE layer makes the SWCs in the application layer independent from the ECU.

#### Application Layer

From the top of the RTE layer the layered architecture changes to the component type. The application layer consists of the Software components which are interconnected with one another using the ports. They are also connected to the RTE to establish communication with the SWCs in other ECUs or to utilise the various services provided by the BSW. There are various guidelines for developing the SWCs so that they have standard interfaces and the so developed SWCs can also be reused.

### 3.1.4 Modelling in AUTOSAR

SWC is the component that encapsulates software that runs on the AUTOSAR architecture. It contains a standard set of interfaces to interact with one another or to utilise the communication services provided via the RTE layer. Application layer is developed in the AUTOSAR as an abstract concept and the granularity to which the application can be broken down is not defined by AUTOSAR. It is up-to the designer to develop a piece of application SWC that can be small and re-used or a large application block that includes that entire system.

There are different types of SWC prototypes available for development of the application layer and these types are chosen depending upon the type of application that has to be developed. Broadly there are three types of SWC types

The atomic software component, as the name itself suggests it's a standalone type of the SWC and it is the smallest type of application component can be developed and cannot be divided into any further smaller components. Each instance of the AUTOSAR SWC is typically assigned to a single ECU.

The composition SWC, as denoted by its name it's made up of the many software components that are related. These are usually used as a model design but it does not have any impact on the program code.

The sensor or the actuator SWC, as the name implies this prototype is used to encapsulate the dependencies of application on specific sensors or actuators. In this way only the algorithm that is independent of the hardware remains in the application layer.

**Runnables**

Runnable entity is a sequence of instructions that can be started by RTE. Runnable is the smallest component to which the application can be broken down into. It is small piece of code that is developed to fulfil a particular function desired. Again no granularity is provided by the AUTOSAR on the development of runnables. It is up-to the designer to decide upon the level to which he can divide the application SWCs.

Not all runnables are started by the AUTOSAR software. There are certain runnable which are run depending upon the occurrence of an event and some SWCs contain the runnables that are to be run periodically. Other than the runnables that are initially started by the AUTOSAR software, other runnables are triggered by RTE events. The periodic runnables are triggered by special kind of events called timing events and other runnables are triggered by using the asynchronous events set by the RTE events.

In order to communicate between the runnables in the same SWCs we have IRV (Inter Runnable Variable). These variables are of primitive data types and are used for information exchange within the SWC. It acts like a global variable carrying data in the SWC.

### Communication ports

The SWCs in the application layer communicates with one another or with the BSW layer using the interfaces and these interfaces are called communication ports or port interfaces. AUTOSAR has defined two types of port prototypes for communicating .They are Sender-Receiver and Server-Client prototype. The SWCs can send the data or provide the features using these port by using the Provide port interface (PPort) and receive the data or demand the feature using the Receive port interface (RPort) interface.

Via these Sender Receiver ports (S/R) SWCs can send and receive many types of data elements, from the primitive datatypes to the complex ones (even the struct data types). This is a model for the asynchronous distribution of data where a sender distributes the data (1:1) to one or several receivers (1: n) or several senders can send to one receiver (n: 1). The S/R supports only a unidirectional communication. The sender does not know the identity of the receivers and it broadcasts the information over the channel. The sender does not expect to receive an acknowledgement for the reception of the messages. The sending component can refresh the values that are to be sent there-by overwriting the current value. A "queued" semantics is used where the consecutive elements are stored in a queue as the information to be sent arrives at the sender port. The receiver site can implement a "last is best" buffer so that it always has access to the latest data element that is communicated. The receivers that receive these messages autonomously knows what to do with the information and how to utilise the data so received. It is the responsibility of the communication infrastructure to provide a channel to distribute the data and also to ensure that data receives the receiver site.

Via the Client Server communication (C/S) there is a possibility of invocation of the operations or features. The server is a provider of the service and client is the recipient of the service. The client is the one who initiates the request for a service while passing some of the operation parameters if necessary via the client-server interface. The server is the one who receives the request for the service along with the parameters and dispatches a response to the client after performing the necessary operations. If the valid operation is requested by the client a valid response is got by the client or an error response will be received by the client. Hence as we can see it is a bi-directional communication in the client-server interface. The direction of initiation of the service request depicts whether SWC is a client or server or both. For the C/S interface only

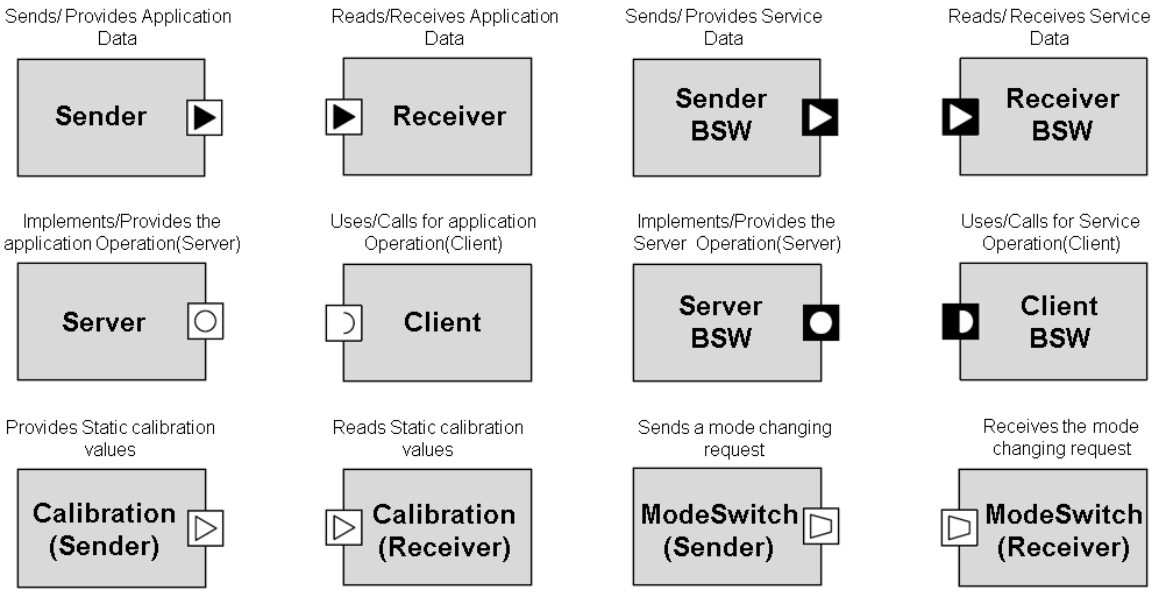



Figure 3.6: Communication Ports [Fri]

"n:1" (n clients, n greater than 0, 1 server) communication is supported . There are two types of client server communication that are supported.

The synchronous/blocked communication: In this type of communication the client is not allowed to invoke a specific operation on the RPort until it receives the response from the server for the previous operation that it had invoked. The synchronous method helps in getting fast response from the server.

The asynchronous/non-blocked communication: In this type the client is allowed to make an invocation of a different operation on the RPort before receiving the response from the server from the previous different operation invocation. However in this case the VFB does not guarantee the order in which the operations are invoked and also on the ordering of the responses received by the client.

## 3.2 Vector AUTOSAR Software Suite

The Ethernet stack was first introduced in the AUTOSAR specification release 4.0. With newer revisions and updates, the AUTOSAR release 4.2 is fully compliant with the software standards of Automotive Ethernet. Inorder to implement an AUTOSAR compliant Ethernet stack, the entire range of AUTOSAR software framework must be available, which include the services layer packages, OS, RTE and the ECUAL packages as well as the MCAL packages for Ethernet driver and transceiver.

Microsar is a complete AUTOSAR software solution available from Vector Informatik GmbH. It consists of the entire AUTOSAR suite that covers all the BSW modules that are part of the AUTOSAR specification as well as the RTE. Additionally the hardware dependant packages such as the OS is also made available.[Gmb].

The MICROSAR suite available at Bertrandt was tailor made for the Infineon AURIX TriCore board. This suite is in accordance with the latest AUTOSAR specification 4.2. The suite also comes bundled with the AUTOSAR tooling solutions for the BSW and RTE configuration, DaVinci Configurator PRO. The application SWC tool used in thesis is the DaVinci Developer. The DaVinci Developer tool is not part of the MICROSAR suite and was procured externally.

## 3.3 Infineon TriCore Hardware

Infineon TriCore TC27x with the actual C-step, which comes from the AURIX family of microcontrollers, was used for development of this project. This is a high-performance 32-bit microcontroller with three CPUs, program and data memories, interrupt systems and a powerful set of on-chip peripherals. The TC27 series of microcontrollers is built on RISC load/store architecture that is necessary for providing high computational bandwidth with low system cost chip. Its mainly used in application areas of Powertrain, ADAS features such as Advanced Anti-Lock Braking Systems, Multi purpose camera configuration and in Telematics applications. [Tec]

The microcontroller is housed in a development board (TriBoard) through which, it provides an interface to connect to the automotive bus systems as seen in the below figure 3.7. This board is used as an interface to provide quick access to the capabilities of the TriCore processor. It comes with a varieties of memories and peripherals for

connection to the environment and also provides an On Chip Debugging Feature. This also provides a provision for replacing the processor when a newer and powerful processor becomes available. Micro-USB connector provides connection from laptop to the board for flashing and debugging.

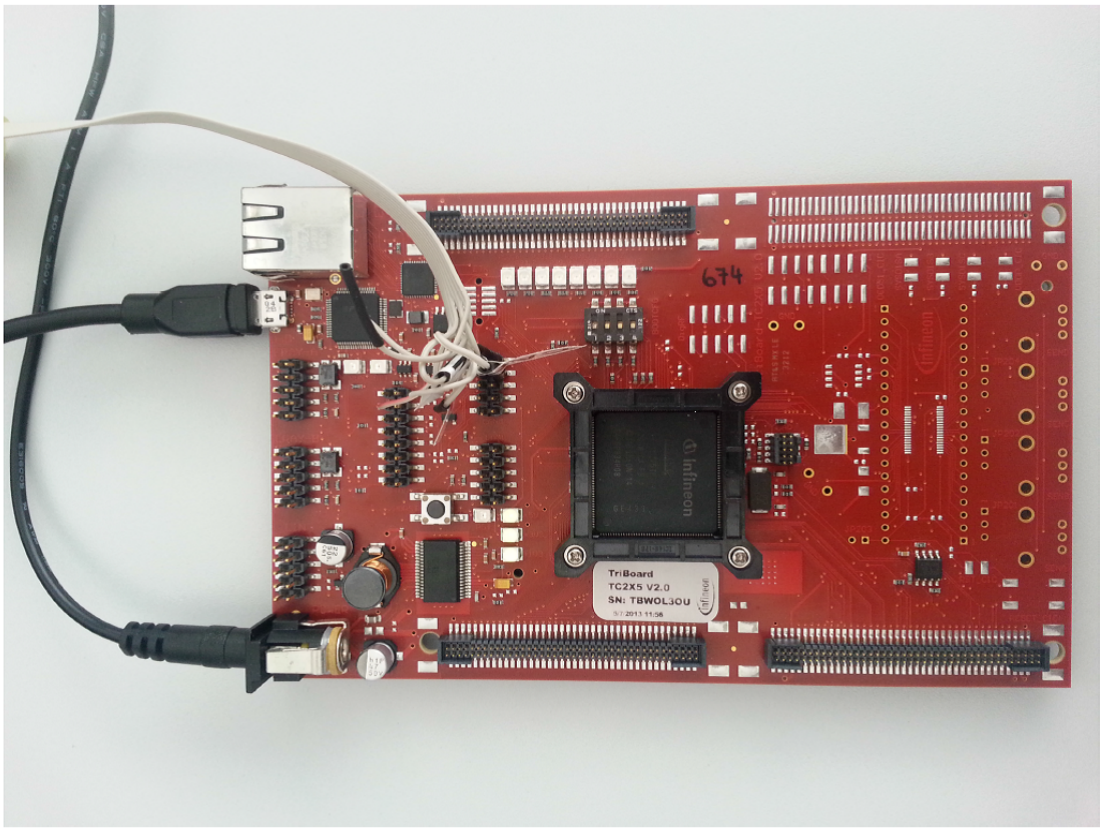



Figure 3.7: Infineon Tricore Hardware

Main features of the Infineon Triboard:

- 32-bit multicore processor
- ETHERNET MAC with 10/100/1000 Mbps capable Transceiver
- 2 * FLEXRAY Transceiver
- 2 * Highspeed CAN Transceiver
- 1* LIN Transceiver
- CPU Clock of 200 MHz

- USB miniWiggler JDS for easy debugging
- 8-DIP switches for configuration.

# 4 Concept

This chapter deals with the fundamental building blocks for this thesis. It introduces the concept behind the implementation of Ethernet compliant AUTOSAR software running on the Infineon board, the rest bus simulation required for verification of Ethernet communication and the test approaches conceived for the evaluation of the available AUTOSAR Ethernet software package.
The first section in this chapter is devoted to the AUTOSAR methodology which is needed to develop the software for the board. The second section explains the concept of rest bus simulation. The third section illustrates the overall architectural setup for this thesis. Finally the various approaches that were devised to setup a suitable test platform for evaluation of the AUTOSAR Ethernet package are explained in the fourth section.

## 4.1 AUTOSAR methodology

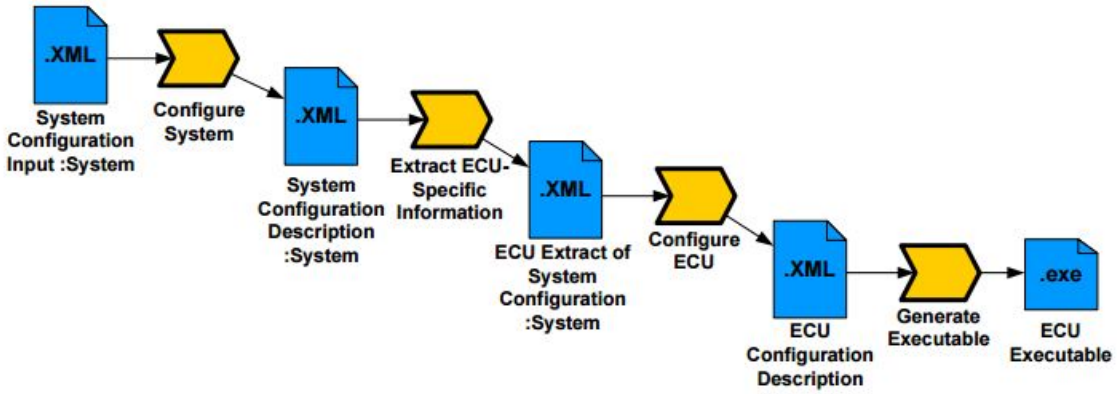



Figure 4.1: Generic AUTOSAR workflow [Heb]

AUTOSAR adopts a uniform workflow for the system development. All steps that are required for the development of the software from the system description to the generation of the executable are shown in the figure 4.1 below. The output all steps

except the generation of the executable step, is in the ARXML format. It is a special XML format for the description of AUTOSAR components. For the generation of an AUTOSAR compliant software for an ECU, these steps need not be necessarily executed. This is discussed in detail in the coming sections.

During the overall design of the system, the software and the hardware components are selected and the overall system description is identified. This is done with the *System Configuration Input*. The AUTOSAR system configuration tools help in configuring the software by identifying the ECU constraints. The communication matrix which carries data regarding the frames exchanged on the network and the related timing information, is one of the important output of this activity. The *System Configuration*: system XML file maps these data to the ECUs with resources and timing requirements.

In the next step *Extract ECU Specific Information*, the system configuration description for the specific ECU will be derived from the System configuration and a one to one mapping of the system configuration description for a particular ECU is done. The *Configure ECU* deals with the configuration of the BSW layer and RTE. It adds the Operating System task related information and the timing configuration and mapping of runnables to the task. These details are extracted into the *ECU configuration description* XML file. It contains the information relevant to the ECU, collections of SWC implementations and BSW configurations.

In the last stage *Generate Executable*, the ECU system configuration consisting of SWC and BSW layer extract is generated into the actual source code. The underlying functionality of runnables are implemented and all the modules are made available for compilation. The code is now compiled using a suitable compiler, linked and finally an executable (.exe) that can be finally flashed on to the microcontroller is produced.

Since only a single ECU (Infineon Aurix board) is under consideration, some of the steps of the AUTOSAR methodology can be overlooked. These include the System Configuration Description and the Extraction of ECU specific information from the System Configuration description. This is illustrated in the figure 4.2.

The ECU extract information in the form of an ARXML file can be generated from a

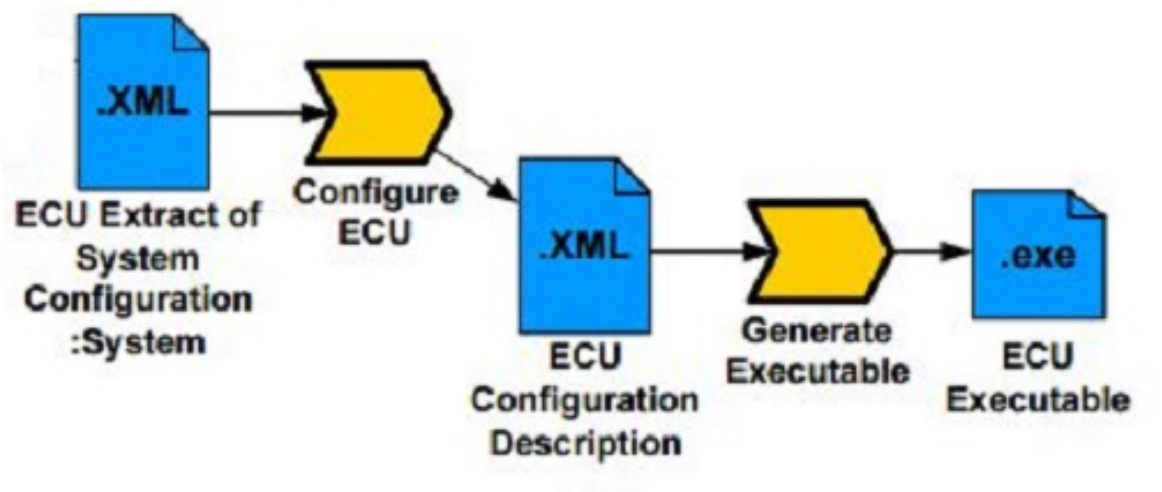



Figure 4.2: AUTOSAR Workflow transformation

tool such as the Vector AUTOSAR System Description Network Explorer. All the ECU specific information regarding the frames, PDU and signals could be input into the tool and the resultant output file in ARXML format is fed into the ECU configuration step.

An example of ECU extract definition is illustrated in the figures 4.3, 4.4 and 4.5.

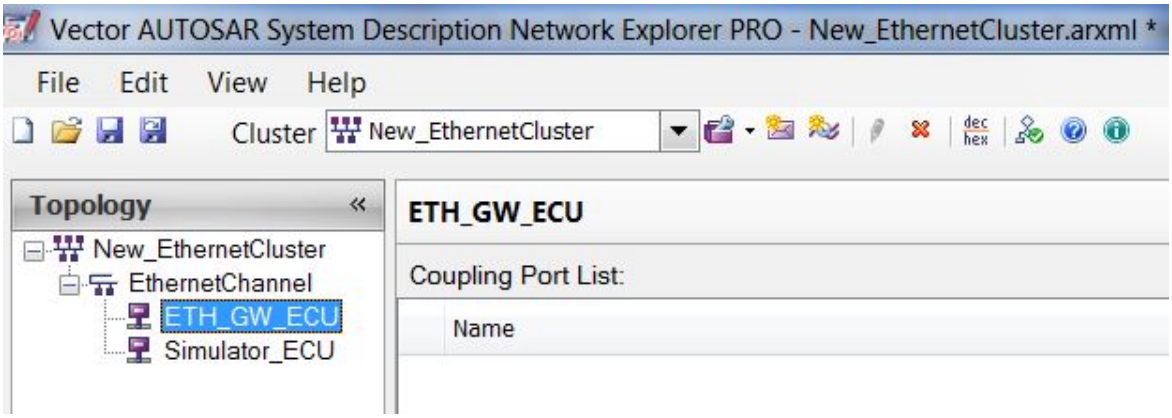



Figure 4.3: Topology definition

In the topology definition, the simulator ECU is shown only for illustrative purpose. The simulator ECU is not part of the AUTOSAR worklow model. The role of the simulator is explained in the coming section.
With respect to Ethernet, the above ECU extract definition is incomplete. Unlike the other bus networks such as CAN and FLEXRAY, Ethernet is a point to point

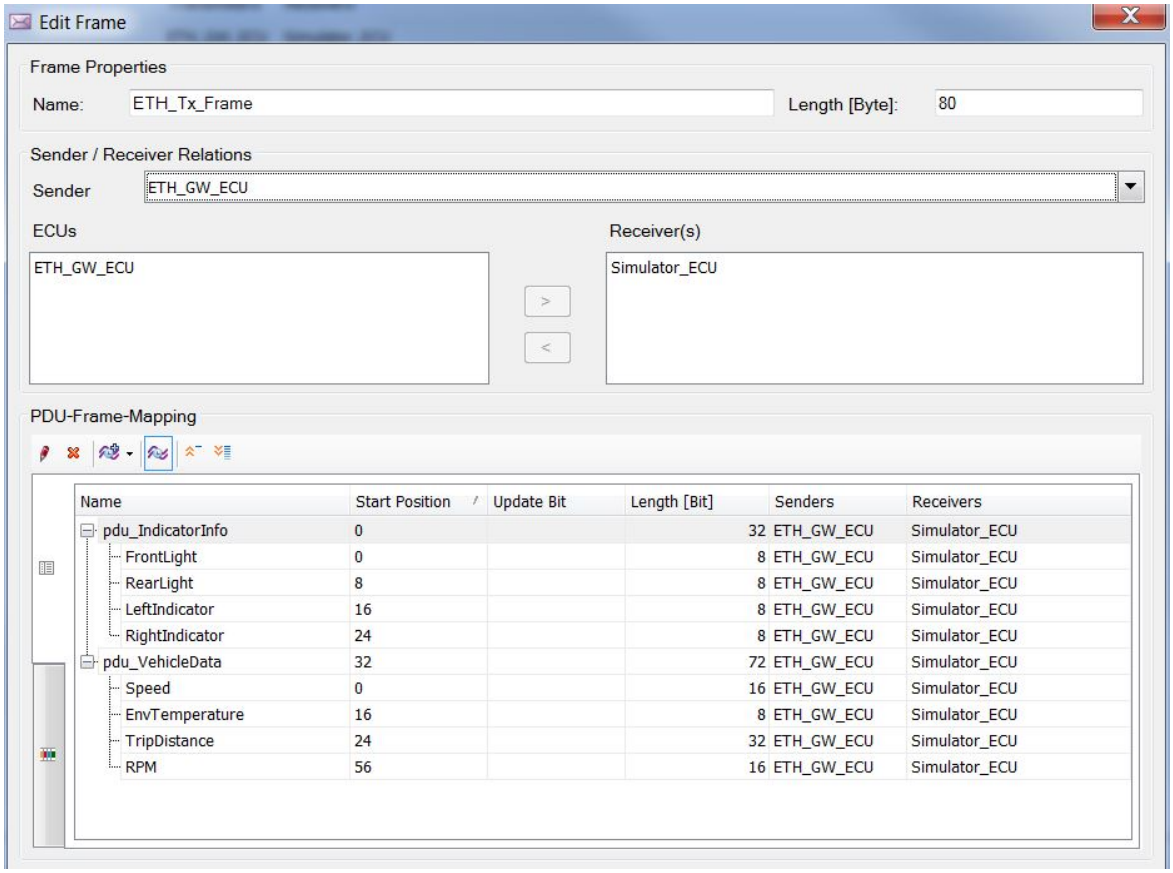



Figure 4.4: Transmission Frames, PDUs and Signals

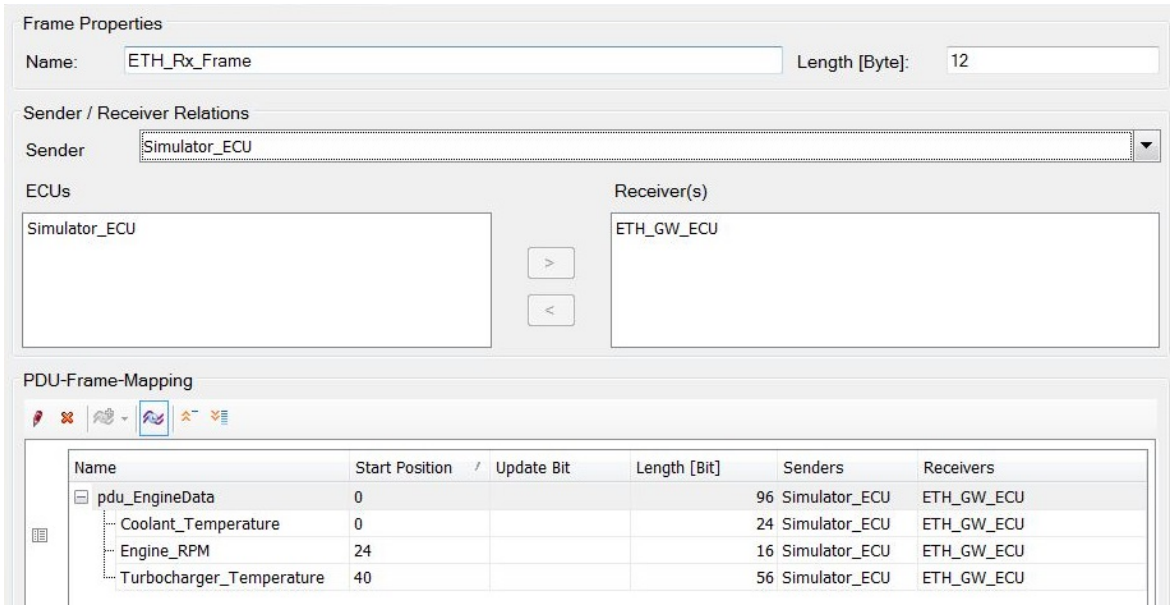



Figure 4.5: Reception Frames, PDUs and Signals

communication system. Hence the sender and reciever frames also carry information regarding the IP and MAC address of their counterparts. Within the state of art, definition of ethernet address data is not possible. This is to be done internally during the ECU configuration process.

## 4.2 Rest bus simulation

Once the AUTOSAR compliant Ethernet software on the board is ready for execution, the logical next step involves the validation of the software running on the board. Here validation consists of testing the correctness of Ethernet communication with regards to the data frames to be sent and received by the board. For this purpose alone, rest bus simulation on a test PC is used.

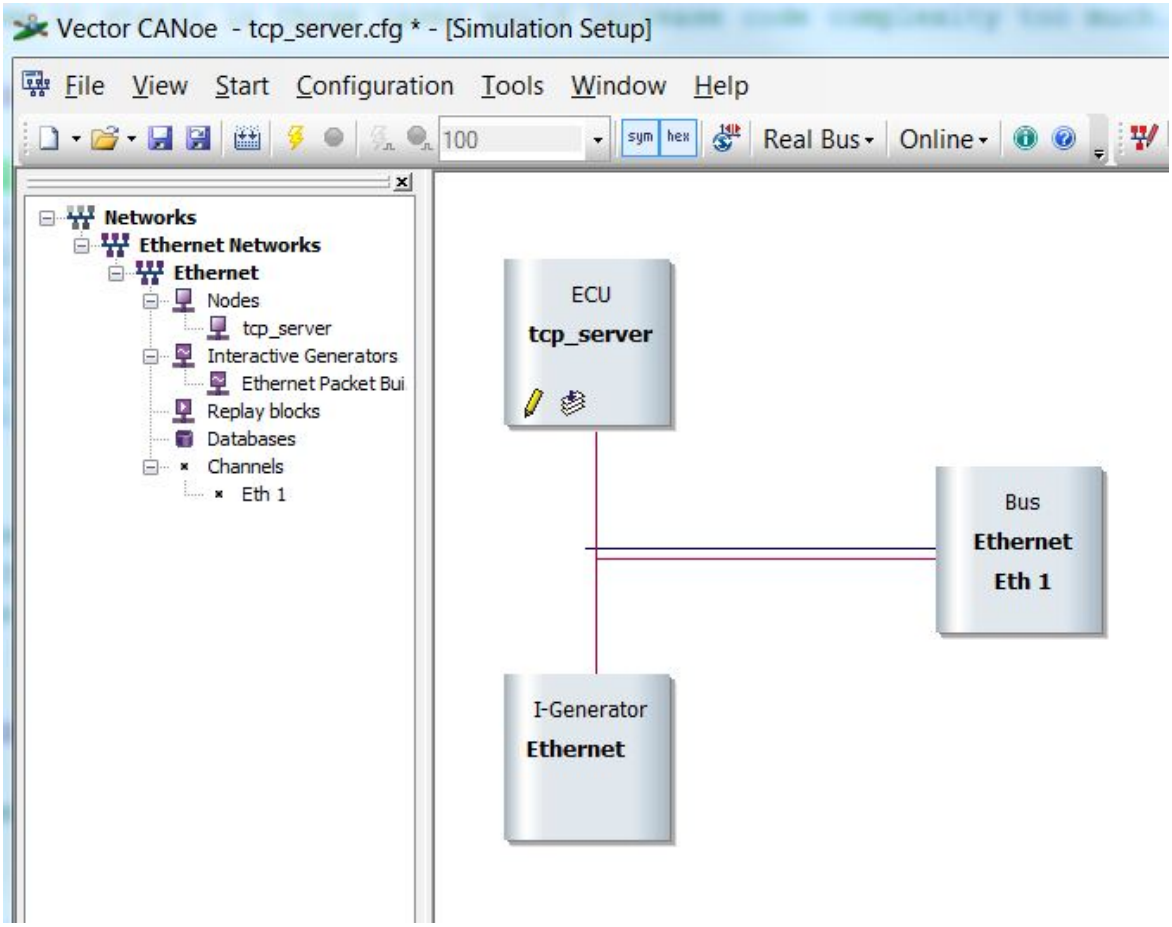



Figure 4.6: Rest bus Simulation on Vector CANoe

The Vector CANoe software tool is used to perform simulation of an ECU on the system. Figure 4.6 shows an Ethernet node being simulated on the network. The nodes for simulation are programmed with CAPL code. CAPL is an event driven

programming interface. With its rich set of available API's, various scenarios could be simulated and validated. Analogous to the ECU extract information passed on during the AUTOSAR workflow, a similar data needs to be passed on to the individual nodes of the rest bus simulation. The frames and signal data (receive and transmit) are mapped onto the nodes via a Database definition. The figure 4.7 illustrates this idea. This is the concept behind the software simulation of the Simulator ECU.

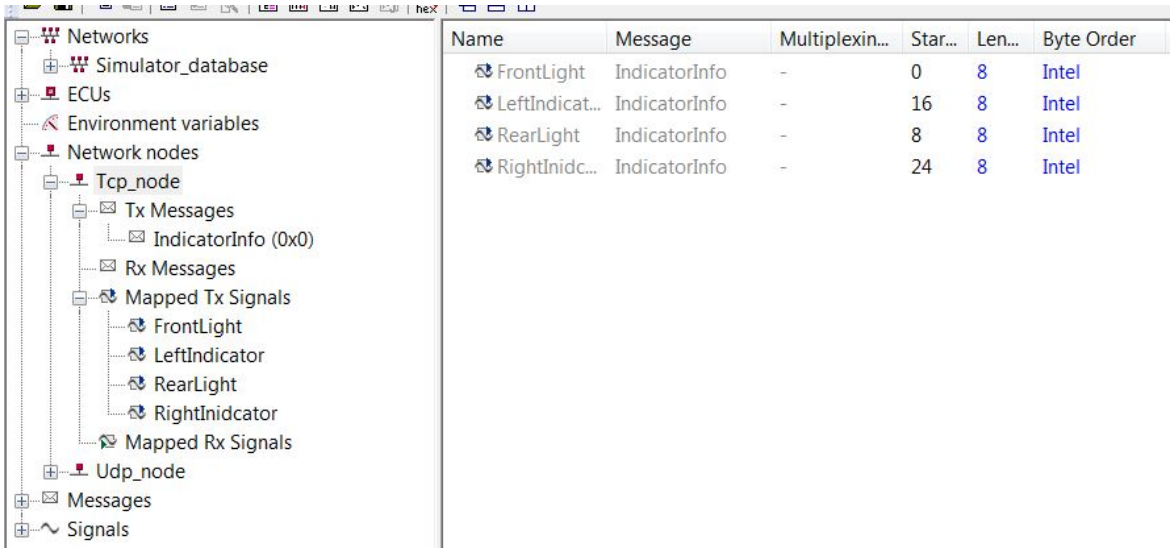



Figure 4.7: CANoe Database definition which is analogous to the AUTOSAR ECU extract

It is possible to simulate an ECU's communication channel such as Ethernet, CAN, Flexray etc. This is done with the help of network interface hardware solutions available with Vector. The VN5610 is one such network interface with two independent Ethernet channels. Depending on the use-case, either of the channels may be configured for our use.

## 4.3 System Architecture

The figure 4.8 illustrates the overall conceptualization.

The thesis aim of an implementation of AUTOSAR compliant Automotive Ethernet stack is realized with the design concept shown in figure 4.8. As a first step, a system configuration description is created. This is analogous to the first step of the AUTOSAR methodology. ECU extract information is created from the system description for both the TriCore board and the simulator ECU configuration. The Database phase of the

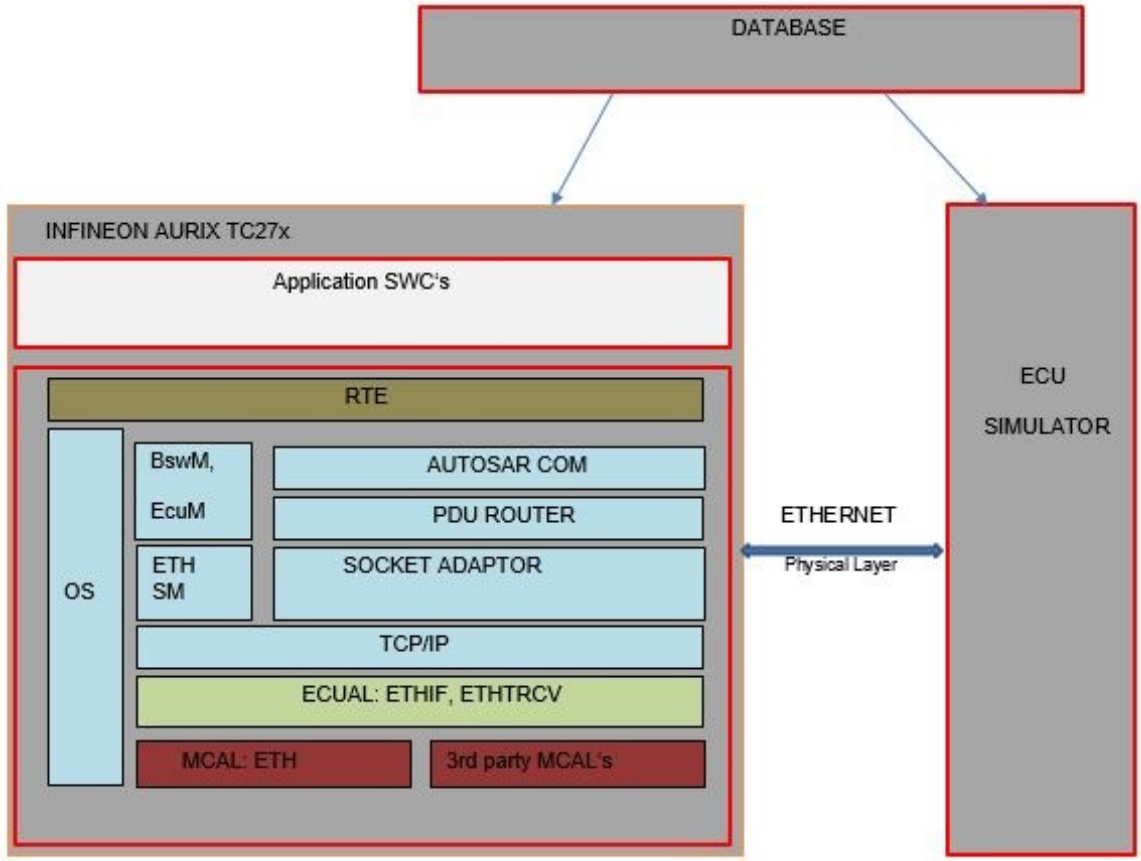



Figure 4.8: Overview of the system concept

figure 4.8 corresponds to this step of ECU extract information.

The next step involves the configuration of the AUTOSAR Ethernet stack. During this phase, the various modules of the Ethernet stack are familiarized and configured suitably. Along with the TriCore ECU configuration, the simulator ECU is realized by rest bus simulation on Vector CANoe. The simulator ECU can provide different TCP or UDP socket based connection setup.

The AUTOSAR configuration on the TriCore board is then subjected to code generation and finally compiled and flashed on the board. In the next step,the TriCore board is connected via an Ethernet link to the Simulator ECU. This connection can be validated by testing the Ethernet frames being exchanged between the board and the test PC.

# 5 Implementation

This chapter deals with the actual realization of the AUTOSAR compliant Ethernet communication stack on the TriCore board. The software package provided by Vector needed to be familiarized before proceeding with the configuration of the various AUTOSAR BSW modules with respect to the Ethernet communication stack. These are explained in detail in this chapter. Additionally other aspects such as Integration of the third party MCAL modules into the Vector software package and the setting up of the compilation environment are also discussed in this chapter.

## 5.1 Integration of 3rd party MCAL modules

The AUTOSAR software package provided by Vector does not contain the MCAL drivers specific to the micro controller. Instead these are made available directly by the semi conductor manufacturer. In this case, the MCAL files are provided by Infineon. As a result these MCAL modules need to be integrated into Vector's software package.

The MCAL modules provided by Infineon are by default, designed for use with the AUTOSAR tooling solutions (namely EB tresos) available from Elektrobit. The package consists of the source codes and the relevant *BSWMD* files for each MCAL module. The MCAL modules can be configured from within the EB tresos tool. The configured modules are then exported into an *ECUC* file for use with other tooling solutions such as Vector's DaVinci Configurator. However this practice was not followed as it became evident it had various other drawbacks. These are summarized as follows.

- Additional knowledge and know-how of a third party tool is required. This simply increases the overhead from multiple tool workflows.
- Configuration from 3rd party tool and then exporting the *ECUC* file may result in loss of data during the import/export process. This data will then have to

be manually coded into the *arxml* files that are used by Vector tool i.e Manual synchronization [Refa].

- With multiple tool based configurations, it may also be possible that the dependencies between *BSW* and MCAL files needs to be solved manually using stubs and callout functions if in case the 3rd party code generator is not fully AUTOSAR compliant. Thus all the non AUTOSAR related code will have to be ported into AUTOSAR format.

The Vector tools come along with code generators as well as the makefiles for the MCAL modules. Hence the option of integrating the 3rd party MCAL modules into the Vector package has far more benefits than working on a separate tool for the MCAL modules alone and removing the dependencies at a later stage.

Integration involves moving the MCAL source files and the *BSWMD* files into the Vector software package. The steps involved are enumerated as follows.

- Within the project working directory, setup a directory path for the MCAL modules and copy the (*.c, *.h files) respective MCAL modules into this new directory.
- Copy the *BSWMD* files for each 3rd party MCAL module into the *BSWMD* folder of Vector software package.
- Add the path to the MCAL *BSWMD* files from within the DaVinci Configurator tool

## 5.2 Configuration of MCAL modules

**MCU Driver**

Though several different MCAL modules are available, only the modules relevant to the use case need to be selected and configured. Of these the MCU (Microcontroller Unit) is common to all applications. It provides the necessary services for the microcontroller initialization, initialization of the MCU clock, initialization of the RAM memory section and other application requirements such as polling the reason for ECU reset. The MCU driver implementation is based on AUTOSAR specification [42c].

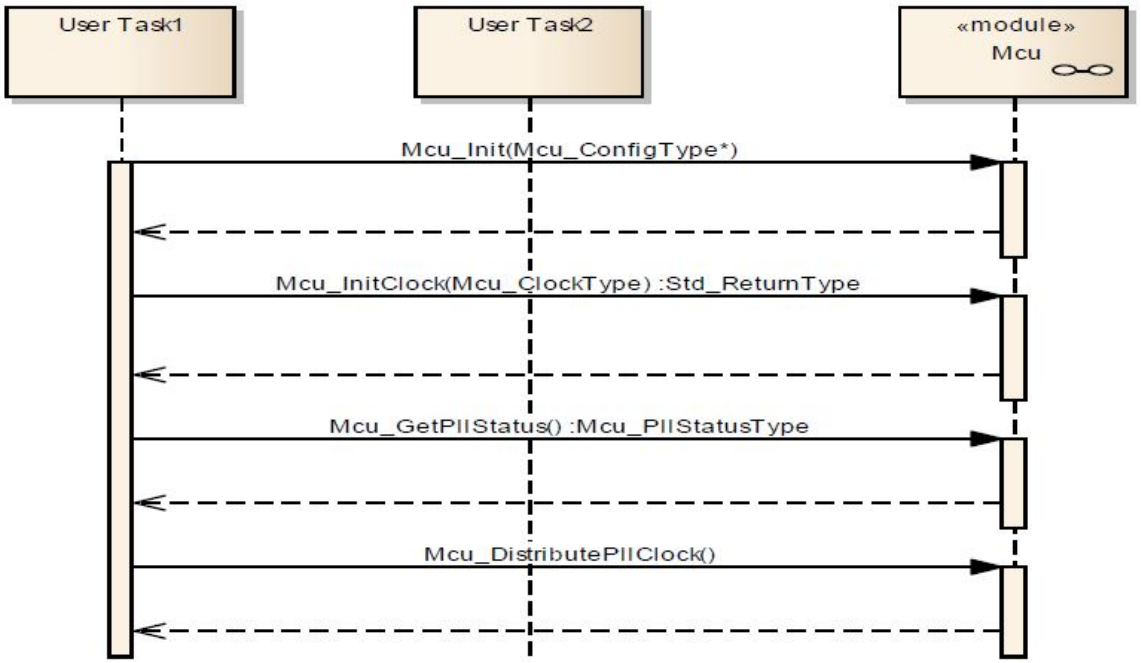



Figure 5.1: Initialization sequence [AUR15b]

Figure 5.1 shows the expected initialization sequence for the MCU driver. The different function calls are implemented within the MCU driver. These are to be invoked during the system startup phase. This configuration is done within the EcuM module, as part of the *EcuM DriverInit list* section. This is responsible for the routines to be executed before the call to start the operating system [42a]. During the code generation phase, these function sequences are generated inside the EcuM callout file which are then linked with the MCU driver module during the compilation phase.

A summary of the configured items for the MCU module is given below.

| | |
|---|---|
| **Main Oscillator freq:** | 20MHz |
| **RAM sector base address:** | 0x70000000 |
| **RAM section variable default value:** | 0x0 |
| **Ethernet clock source:** | ERAY PLL |

### PORT Driver

The port driver is used to initialize the different pins used for a particular application. These pins are configurable via the port registers of the microcontroller. These pins are combined in hardware to form groups called ports [AUR15c]. The port pins may be configured as output or input depending on its functionality.

For the application under study (i.e Ethernet), only a select few ports were needed to be configured. These correspond to the Ports 11 and 21 on the chip. Additionally for purposes of debugging during the initial system integration phase, port 33 which corresponds to the on-board LEDs was also configured.

The Ethernet interface on the TriCore board consists of the Ethernet MAC, Ethernet Transceiver module and the RJ45 connector for external communication[AUR13]. The MAC and the Transceiver modules need to be interfaced by MII(Media Independent Interface) or RMII(Reduced Media Independent Interface) connection.This is illustrated in the figures 5.2 and 5.3.

The MII interface consists of 18 pins while the RMII interface uses only 9 pins. [Wika]. With reduced pin configuration, the RMII interface is chosen as the interface between the MAC and Transceiver modules.

These pins are available on the Ports 11 and 21[AUR13]. Accordingly these were configured as input or output pins.

Table 5.1: Port-Pin configuration for Ethernet

| **Port.Pin** | **Pin Function** | **I/O Control** |
|---|---|---|
| 21.0 | MDC | Output |
| 21.1 | MDIO | Output |
| 11.2 | TxD1 | Output |
| 11.3 | TxD0 | Output |
| 11.6 | TxEN | Output |
| 11.9 | RxD1 | Input |
| 11.10 | RxD0 | Input |
| 11.11 | CRS | Input |
| 11.12 | PHY CLK | Input |

The MDC and MDIO pins are management signals used to exchange control signals between the MAC and Transceiver modules. Tx(D0, D1) are the transmitter signals with D0 corresponding to Bit 0 and D1 to Bit 1. D0 is always transmitted first. TxEN is the Transmitter clock. Transmission happens during the clock's high phase. Rx(D0, D1) are the reciever signals from the Transceiver to the MAC with D0 cor-

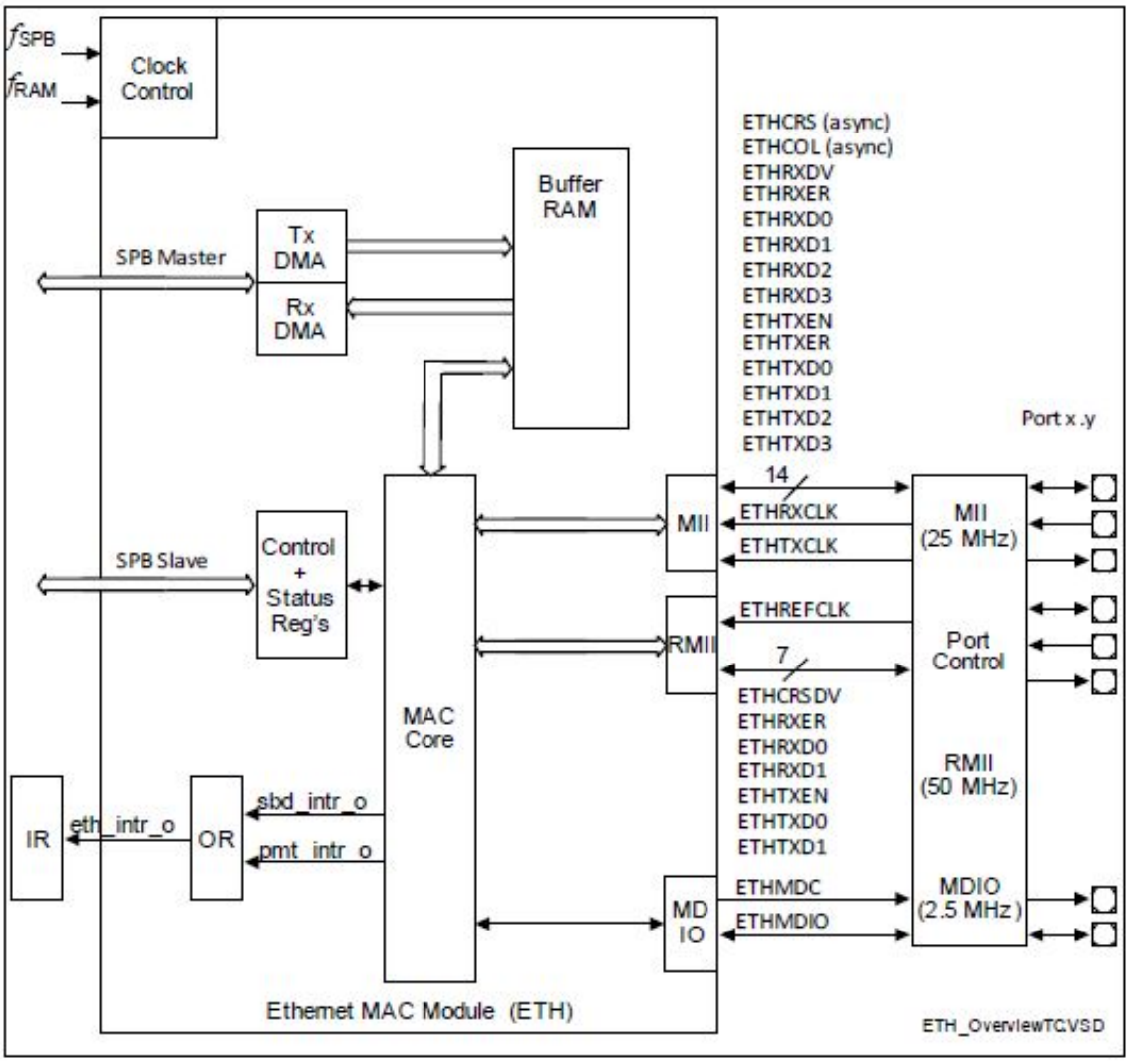



Figure 5.2: Ethernet MAC module overview [AUR13]

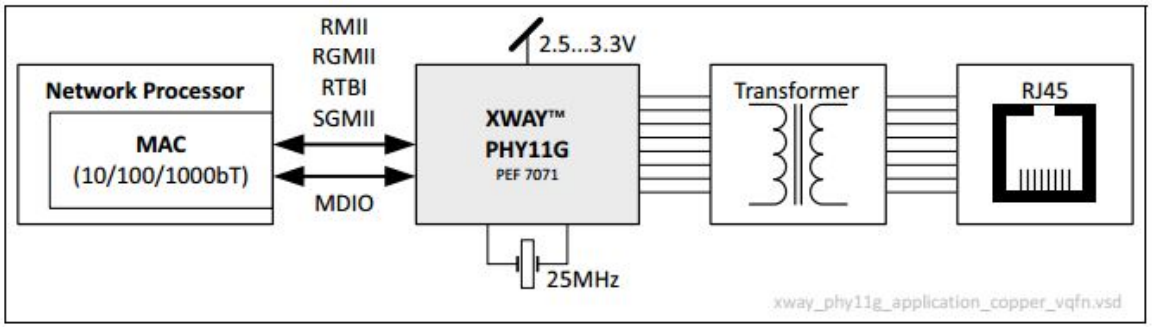



Figure 5.3: Ethernet MAC module overview II[LAN12]

responding to Bit 0 and D1 to Bit 1. CRS is the Carrier Sense signal for collision detection. PHY CLK corresponds to the clock input for the Transceiever module[Wika].

#### DIO Driver

The DIO driver provides direct read and write access to the port pins of the microcontroller. Only pre condition is that the port pins must be configured before access via the Port driver [AUR15a]. The DIO driver was used in this thesis during the initial test phases to verify the running state of the ECU via setting up a counter operation that turns ON and OFF a set of On-Board LEDs. There are a total of 8 available LED Pins. These are mapped to Port 33. Upon configuration of these pins in the Port driver, each pin is configured as an independent channel on the DIO driver container. Upper layer applications can invoke the *DIO_WriteChannel(ChannelNumber, Value)* or *DIO_ReadChannel(ChannelNumber)* to perform read or write operations on the pin. The Dio module initialization is also configured by adding the *Dio_Init()* code to the BswM configuration.

## 5.3 Configuration of Ethernet Communication stack

The Ethernet communication stack consists of modules from the various layers of the AUTOSAR software architecture. These include the Ethernet(Eth) driver from the MCAL layer, the Ethernet Transceiver(EthTrcv) and Ethernet Interface (EthIf) modules which are part of the ECUAL layer, as well as the Services layer modules such as the TCP/IP, Socket Adaptor (SoAd), Ethernet State Manager (EthSM) and application layer protocols DoIP, Service Discovery (SD) and SOME/IP modules. In this thesis, the communication was done in the classical AUTOSAR method of application SWCs whereby PDUs and Signals were being received and transmitted between the SWCs and the Ethernet communication stack. Hence the DoIP, SD and SOME/IP modules were not used in this thesis. The following sections deal with the configuration of the various modules of the Ethernet communication stack.

### 5.3.1 Ethernet Driver (Eth)

The Ethernet driver provides a hardware independent interface to the upper layer. This interface is uniform to all controllers so that the upper layers may access the underlying hardware in a uniform manner. The Ethernet driver provides functionality for initialization, configuration and data transmission[42b].

The Eth module initialization is configured by adding the *Eth_Init()* code to the BswM

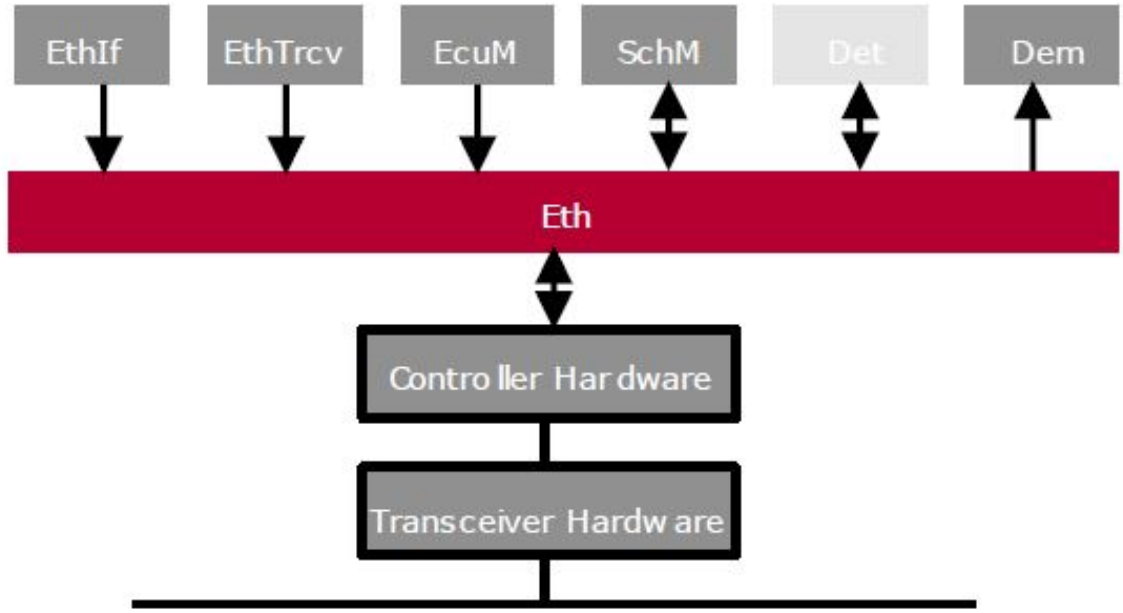



Figure 5.4: Ethernet Driver Interfaces [Refb]

configuration. On ECUs with multiple Ethernet controller hardwares, the same driver is used though of a different instance. Only requirement here being that the controllers must be of the same type. The configuration of the Eth module is grouped into general parameters as well into a controller specific parameter group.

The Ethernet controller may be configured as interrupt driven or polling driven for reception indication and Transmission confirmation. Here the device is configured as interrupt driven. With interrupt driven mechanism, whenever an interrupt is triggered, the corresponding callback functions gets invoked and the notification is passed on to the upper layers. For example, during a transmission confirmation interrupt from the bus, the corresponding EthIf module function gets invoked. A typical sequence is shown in the figure 5.5.

The interrupts are configured as Category 2 ISR. This allows the OS to handle the ISRs. The above said parameters are general parameters that are common to all the controllers for this driver. In addition, there are the controller specific parameters. These are explained below.

As explained in the Port Driver section, the interface between the controller and the

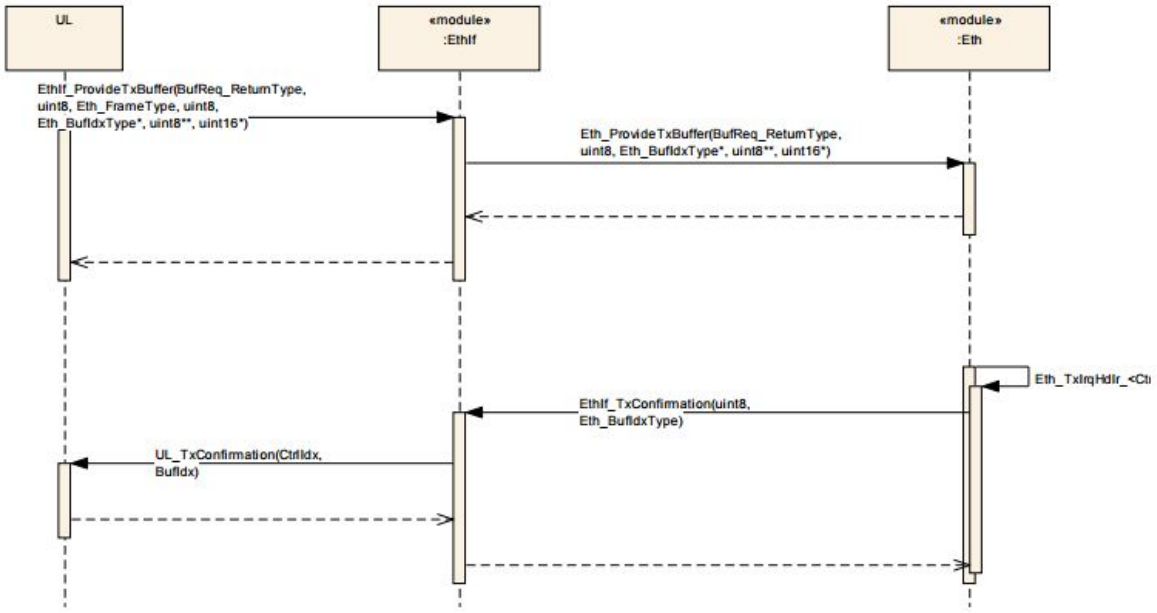



Figure 5.5: Frame Transmission in Interrupt Mode [Refc]

transceiver is chosen to be RMII interface. The same information is also passed onto the ETH driver configuration via a parameter, by setting RMII as the option instead of MII.

MAC address of the controller is configured in the ETH driver. Since this parameter is a part of the controller specific group, multiple controllers may be configured with distinct MAC address. Also the controller specific hardware register base address are enabled here. The register base address values were obtained from the hardware documentation [AUR13].

The controller is also configured with the transmission and reception memory buffers. The number of buffers configured for Transmission or reception must equal the number of nodes with which the ECU is expected to communicate. The size of each buffer is also configured. The size is governed by the minimum and maximum length of the Ethernet frame. These are 46 and 1500 bytes respectively. In addition to this the 14 byte header and 4 byte CRC should also be taken into account. Thus if the maximum expected payload for transmission is 500 bytes, then the size of the transmission buffer would be $500 + 14 + 4 = 518$ bytes.

IF VLAN tagged frames are used, the length of the header will be 18 bytes.

### 5.3.2 Ethernet Transceiver Driver (EthTrcv)

Similar to the Eth driver, the Transceiver driver provides a hardware independent access for upper layer modules. The Transceiver driver can also handle multiple transceivers under the condition that they are all of the same device type.

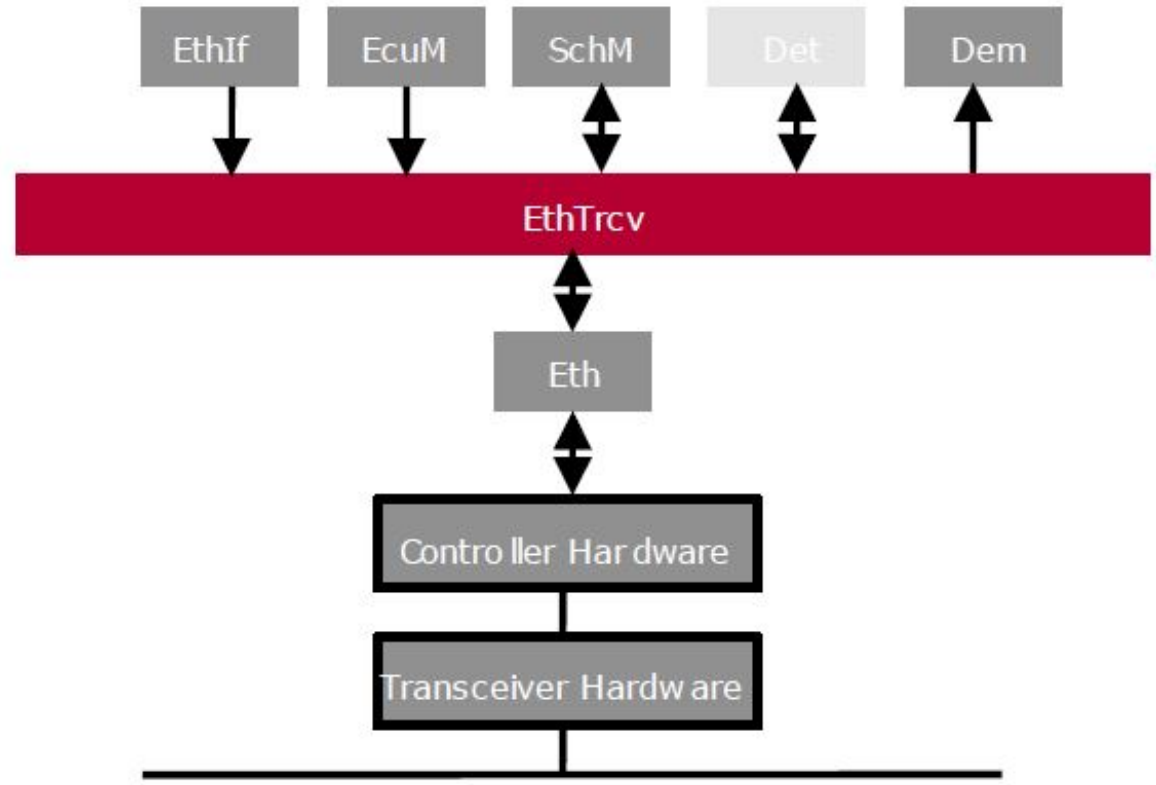



Figure 5.6: Ethernet Transceiver Driver Interfaces [Refd]

The EthTrcv module initialization is configured by adding the *EthTrcv_Init()* code to the BswM configuration. The interface with the Ethernet controller is selected as RMII. In this way, both the controller and transceiver chips are notified of the desired interfacing. The speed of the transceiver link can also be configured. Speeds of 10Mbit/s, 100Mbit/s and 1000Mbit/s are available for this particular transceiver chip. The configured value is 100Mbit/s.

The module offers features for the upper layer modules to perform the basic activation tasks such as setting the Transceiver active mode, starting the Auto-negotiation process as well as reading the transceiver link state for the Ethernet State manager processing.

These features are enabled so that the EthIf higher layer has access to these functions and no explicit function calls from the user level are required.

### 5.3.3 Ethernet Interface (EthIf)

It is designed to provide a uniform interface to the upper layer protocols such as TCP/IP, Ethernet State Manager, etc. On the other hand, based on the configuration, it is able to handle communication requests with multiple Ethernet controllers or transceivers. In this thesis, only a single controller and transceiver are needed and configured as explained in the above sections. The configuration tool provides a parameter that links a particular controller and transceiver configuration to the EthIf module. The actual handling part is then left to the code generation mechanisms.

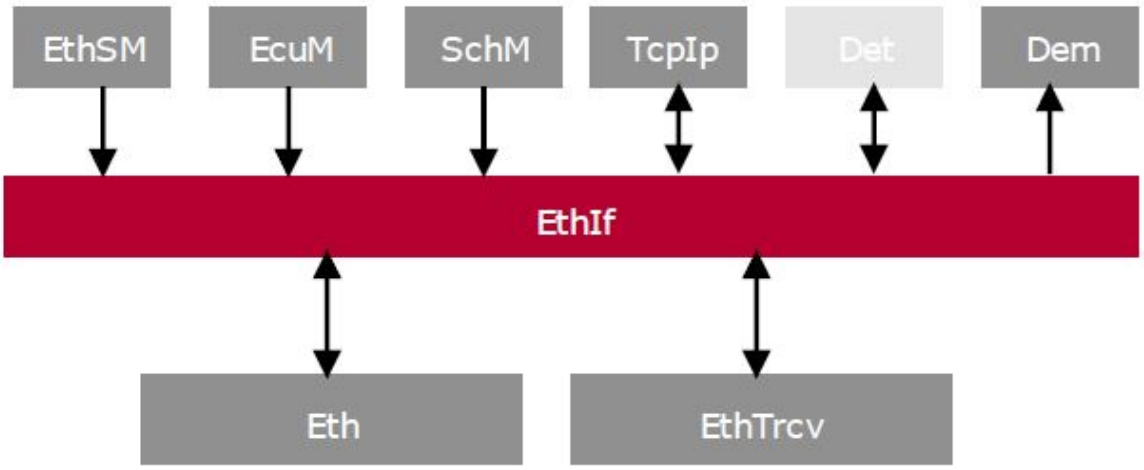



Figure 5.7: Interfaces of EthIf [Refc]

The AUTOSAR specification allows for zero copy extensions during the data exchange between the higher layer applications and the underlying Ethernet driver. Thus the higher layer modules may allocate a buffer, copy data into it and pass its pointer to the Ethernet driver module, which then simply de-references the pointer and copies the data into the hardware registers. Unfortunately the Vector software framework has no implementation for this feature. As a result, every time data needs to be exchanged between the driver and higher layer, the higher layer modules have to explicitly ask the driver for a buffer and copy the data into it. This results in an additional data copy. Due to no support for Zero copy extension, the feature is disabled in the EthIf configuration.

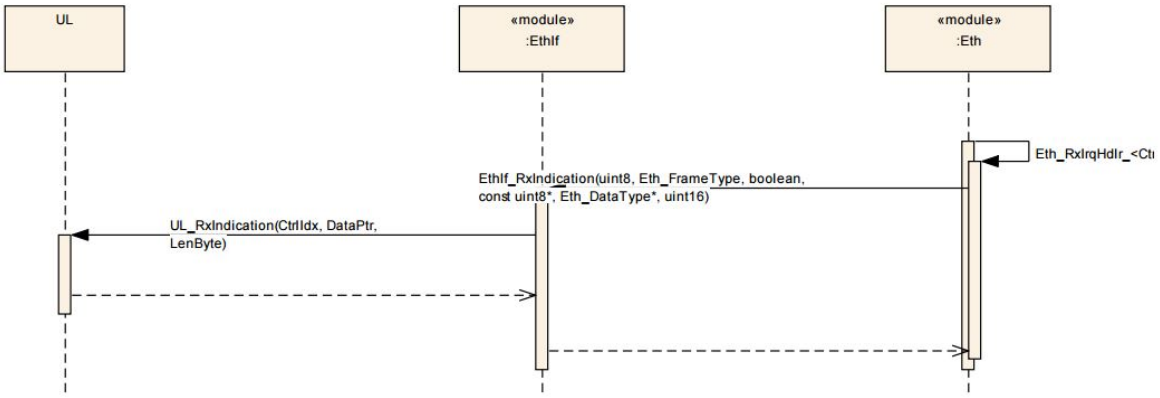



Figure 5.8: Frame Reception in interrupt mode and subsequent callback to upper layer [Refc]

At the point of Ethernet Interface, there are various upper layer protocols to which the received data may be passed on. This is done by reading the protocol tag on the received Ethernet frame header and payload. For example, the received data could be an ARP (Address Resolution Protocol) message or an ipV4 message or even ipV6 message. The EthIf can handle this scenario only if it is configured with the various expected protocols. This shall be known during the system design phase. In this thesis work, only ipV4 and ARP messages are expected from the other node. ipV4 and ARP were configured as the higher layer protocols for the EthIf module. Additionally the module should also be notified of the desired upperlayer call back functions when a particular message is received. The AUTOSAR specification mentions the *IpV4_Ip_RxIndication()*, *IpV4_ARP_RxInidcation()* as the callback functions for ipV4 and ARP messages. These were accordingly configured in the EthIf module. Figure 5.8 illustrates this sequence.

The EthIf module initialization is configured by adding the *EthIf_Init()* code to the BswM configuration

### 5.3.4 Tcp/Ip

The Tcp/Ip module is a services layer module that is located between the upper layer SoAD and lower layer EthIf modules. Figure 5.9 shows the entire family of TcpIp protocols that are included in this module. It is upto the system designer to select the protocols that need to be configured. In this thesis work, the protocols that were configured and implemented include the ARP, IPv4, TCP and UDP protocols. IPv4

implements the Internet Protocol and on top of it, the TCP or UDP protocols are implemented. This software architecture completely agrees with the Ethernet OSI model that was introduced in the State of Art chapter.

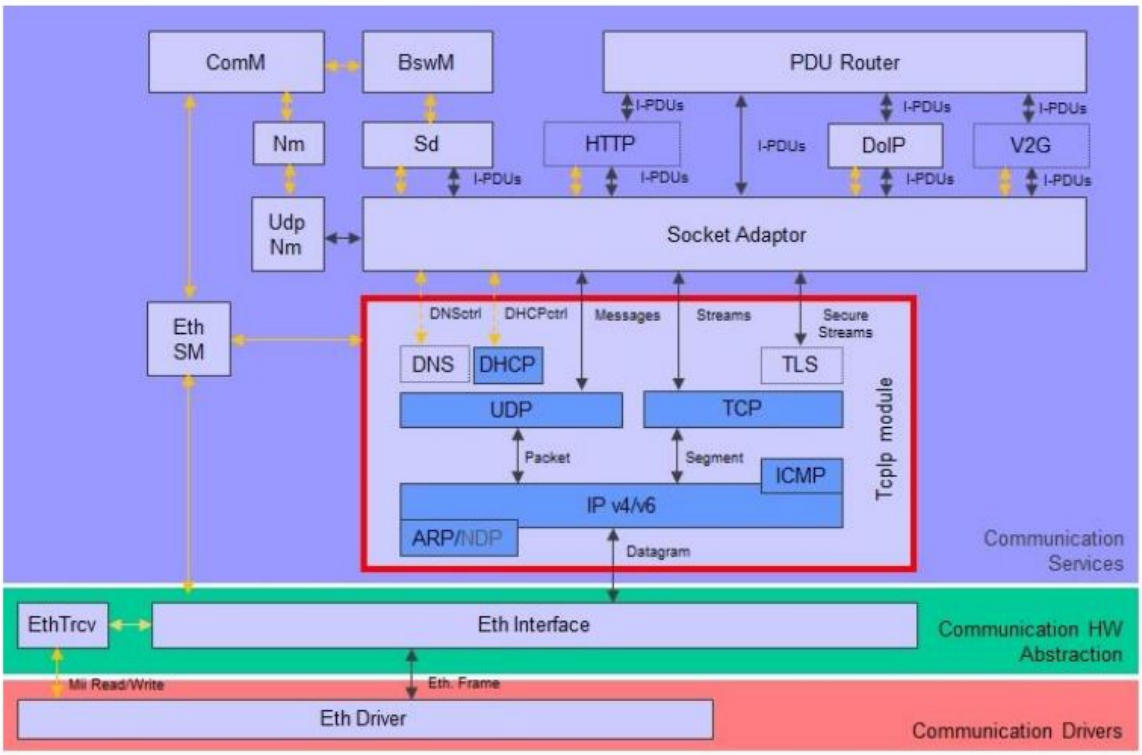



Figure 5.9: TCP/IP Architecture Overview [42e]

In this thesis, static ip addressing scheme was used. This means that the ECU is statically configured with an ip address. This ip address should be within the same network as its counterpart ECU or ECUs with which it intends to exchange Ethernet communication. The chosen ip address for the TriCore ECU was 192.168.1.11. Only point of significance is that the nodes in the network must have a similar ip address with only the least byte being different (for example 192.168.1.x). The corresponding MAC address for this board was already configured in the Ethernet driver module, as explained in that chapter.

If the ECU wants to send a message to another device in the network, it requires the ip address of the counterpart. This is also configured, though only in the SoAD module. Once the ip address of the destination node is known, the device does ARP resolution before starting the actual message transmission. With ARP resolution, the device broadcasts the ip address of the destination node. The destination node on seeing

the ARP request, acknowledges the message with its own MAC address. Only when this handshaking is performed, the actual message exchanges start. However, there is a workaround feasible with the use of Static ARP tables. For each ip address, the corresponding MAC address is also stored alongside. Thus there is no need for ARP resolution and message transmission begins from the sender straightaway.

The static ARP tables with IP, MAC address pairs are configured in the Tcp/Ip module. In this thesis, both the mechanisms (with and without static ARP tables) were configured to investigate the protocol stacks behaviour. ARP retry time and retry interval time were also configured for 5s and 2s respectively. This means that whenever an ARP resolution fails, the device restarts ARP request after waiting for 5s and then repeats the request every 2s.

The above set of parameters cover the configuration of the IPv4 and ARP protocol families. For TCP and UDP protocol configuration, the information regarding the upper layer sockets plays a crucial role. For each TCP socket configured in the upper layer, two corresponding socket buffers must be configured in the TCP module, with its size also configured depending on the expected socket payload in bytes. Each socket must also be configured as a listen or an initiator socket. Listen sockets are initially created during ECU startup and wait for incoming connections. In the case of connector sockets, after ECU startup, they initiate a connection to a destination node.

With UDP, only the number of expected connections are configured. Whenever the device wants to send data or data is received from a destination, the module opens the UDP socket, processes the data and closes the socket.

The Tcp/Ip module initialization is configured by adding the *TcpIp_Init()* code to the BswM configuration

### 5.3.5 Socket Adaptor (SoAd)

The Socket Adaptor module has a generic upper layer, whereby it may be possible to interface with the PDU router, DoIP, or SD. On the other side, the Tcp/Ip module forms the lower layer of the SoAd module. As the name suggests, the module provides the socket based connection demanded by Ethernet communication standards. Figure 5.10 illustrates the role of the SoAd module in the AUTOSAR Ethernet stack.

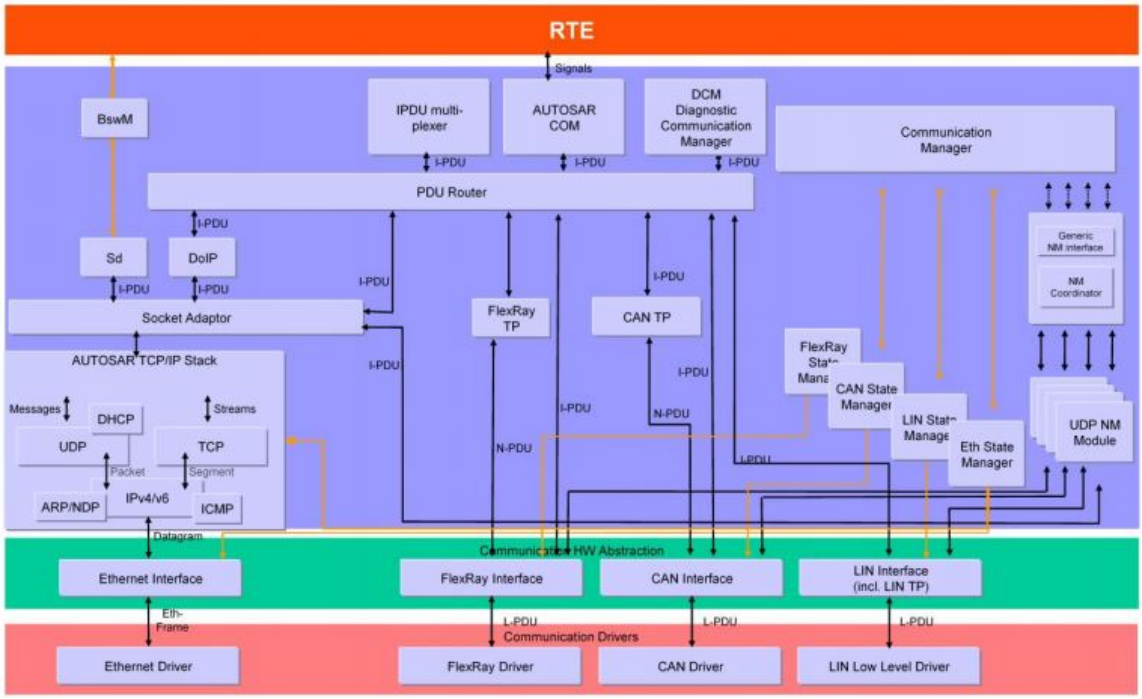



Figure 5.10: Comparison of Ethernet stack architecture with other communication stacks [42d]

A socket inherently carries the ip address and port information of the destination ECU. This could change dynamically. Whereas in the AUTOSAR concept, all the information is pre-determined and the communication can be considered as a form of static communication.

SoAd module tries to bridge the gap between the dynamic concepts of Ethernet with the static concepts in AUTOSAR communication methods. The I-PDUs coming from the upper layers need to be routed to a particular socket connection and viceversa[42d].

The SoAd module configuration is based on mapping the Transmit and Recieve PDUs to two different groups, known as the SoAdPduRoute and SoAdSocketRoute. The SoAdPduRoute corresponds to the Transmit PDUs while the SoAdSocketRoute corresponds to the Receive PDUs.

Each PDU is to be marked with a remote address corresponding to the destination to which it transmits or receives from. Moreover the connection setup also needs to be

pre-configured, that is as TCP or UDP. For this purpose the SoAd module allows the creation of a Socket connection group. A socket connection group is configured as a TCP or a UDP group. The elements of a group are configured with the remote address and port information.
The above information can be summarised as below:

**SoAdPduRoute**: It is a collection of Transmit PDUs
**SoAdSocketRoute**: It is a collection of Receive PDUs

**SoAdSocketConnectionGroupTCP**: A Group of TCP sockets each having a remote address information
**SoAdSocketConnectionGroupUDP**: A Group of UDP sockets each having a remote address information

By linking the PDUs to a particular socket in a group, the SoAd configuration is completed. This has recreated the static concepts in AUTOSAR communication methods.

The sockets of a group may also be pre-configured to enable ip address change notification or socket connection mode change notification. This allows dynamic address change and restart of connection in case of a failed connection.

In this way the static requirements of AUTOSAR and dynamic requirements of Ethernet communication are bridged by the SoAd module.

The SoAd module initialization is configured by adding the *SoAd_Init()* code to the BswM configuration.

### 5.3.6 Ethernet State Manager

The Ethernet State Manager provides an interface to the Communication Manager to startup or shutdown the Ethernet communication channel. Ethernet State Manager needs to be notified of the protocol being used for Ethernet communication. This could be either Tcp/Ip or the Audio/Video Bridging (AVB) protocol. In this thesis work, the Tcp/Ip is the protocol configured for the Ethernet State Manager.

The EthSM module initialization is configured by adding the *EthSM_Init()* code to the

BswM configuration.

## 5.4 Pdu Router

The Pdu Router module lies between the COM layer and the SoAd layer. The Pdu Router module configuration is simplified with the AUTOSAR tools since all the PDUs are automatically mapped based on their direction of transmission or reception.

The PduR module initialization is configured by adding the *PduR_Init()* code to the BswM configuration.

## 5.5 SWC Design and Implementation

The Application SWCs were designed with the DaVinci Developer Tool. For the Ethernet communication, only a basic SWC was used with minimal functionality just to read and write signal data. Only upon this request, the corresponding communication channel Ethernet, is allowed to start transmission and reception of PDUs.

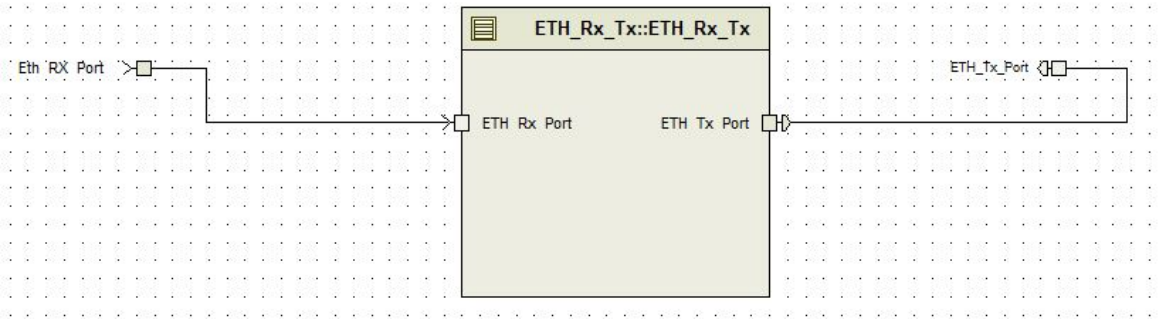



Figure 5.11: SWC Modelling

Figure 5.11 illustrates the design of the application SWC used in this work. The ports were modelled as Sender/Receiver Interface, with Eth_Tx port as a Sender and Eth_Rx as a Receiver. In addition to the Sender and Receiver Port of the SWC, a service Port was added to the SWC. It's role is to request FULL_COM_MODE. Only upon this request, the corresponding communication channel Ethernet, is allowed to start transmission and reception of PDUs.

## 5.6 Operating System Configuration

Figure 5.12 shows the various tasks scheduled on the OS.

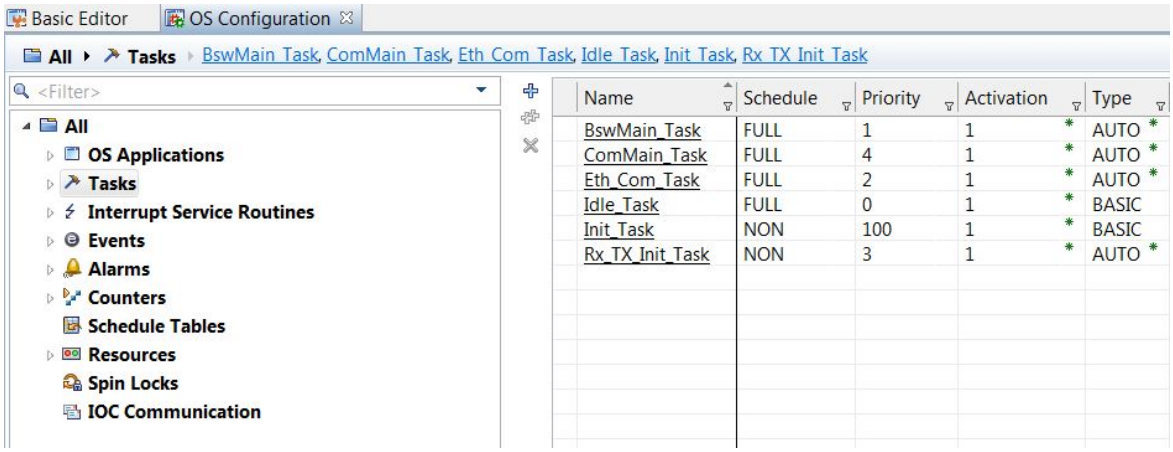



Figure 5.12: OS Tasks

The Init task and idle task are configured as 'AUTOSTART'. Init task has the highest priority of all the tasks. It is responsible for invoking the EcuM_StartupTwo, which in turn invokes the Scheduler (SchM), RTE and the BswM. Idle task as the name suggests does not have any runnables mapped to it. It simply serves to keep the OS alive during the context switches between tasks or at instants when no tasks are scheduled. In such scenarios, the OS is designed to go into shutdown. To prevent this, the idle task is configured which makes sure, at any slice of time, the OS has a scheduled task.

The *BswM_MainFucntion()* and *EcuM_MainFunction()* are mapped to the BswMain-Task. The *Com_MainFunctionRx()*, *Com_MainFucntionTx()* functions are mapped to the ComMainTask. All the schedulable runnables of the Ethernet stack are grouped together in the EthMainTask. These include the SoAd_MainFunction, TcpIp_MainFunction() and the EthSM_MainFunction(). All these are cyclic tasks. As mentioned during the SWC design, a service port is used in the SWC to request for FULL_COM_MODE. The corresponding runnable function needs to be triggered at initialization phase. For this reason, a separate RX_TX_Init_Task is scheduled with the corresponding runnable mapped to it. This task is set for trigger on Init.

## 5.7 Build Environment

Upon successful configuration of the entire AUTOSAR package on a tool chain such as the Vector DaVinci Configurator, the configured system is then transformed into the actual C code using the tool's inbuilt code generator package. This paves the way for the next step in the workflow i.e Code compilation and Linkage to generate the executable file. This is done with the help of a suitable build environment. The Vector AUTOSAR software package comes along with a raw build environment which then needs to be adapted to our specific use case. For instance, since Ethernet COM package is the only communication bus being utilized here, the software related to other bus systems such as CAN, FLEXRAY need to be eliminated. Similarly the Infineon AURIX board's Ethernet controller needs to have its memory sections aligned as per the hardware specifications. Such adaptations were done in the linker file available with the AURIX compiler tool chain [Alt].

Figure 5.14 shows the general workflow of the build process of the whole software. The whole process runs in the context of the Makefile which then invokes other files that carry the project related data (compiler, linker, individual *BSW* modules related data etc.). The build process is started by running a batch script that contains the path to the Makefile and calls the make.exe executable file.
The Makefile was adapted to define the following features.

- The name of the executable file
- Set the compiler path
- Additional compiler options: During the board boot up phase, the start-up code allows features to be enabled or disabled. Protection functionality for accessing *SFR* registers could be turned off with appropriate macro flags being defined. This was done from the makefile environment. Similarly the watchdog functionality was turned off.
- File formats for executable file and linker file were defined as *.elf* and *.lsl* respectively

**Makefile.config**

The board derivative and the oscillator frequency were defined here. This particular board is part of the *TC27x* series with the main oscillator frequency being *20Mhz*

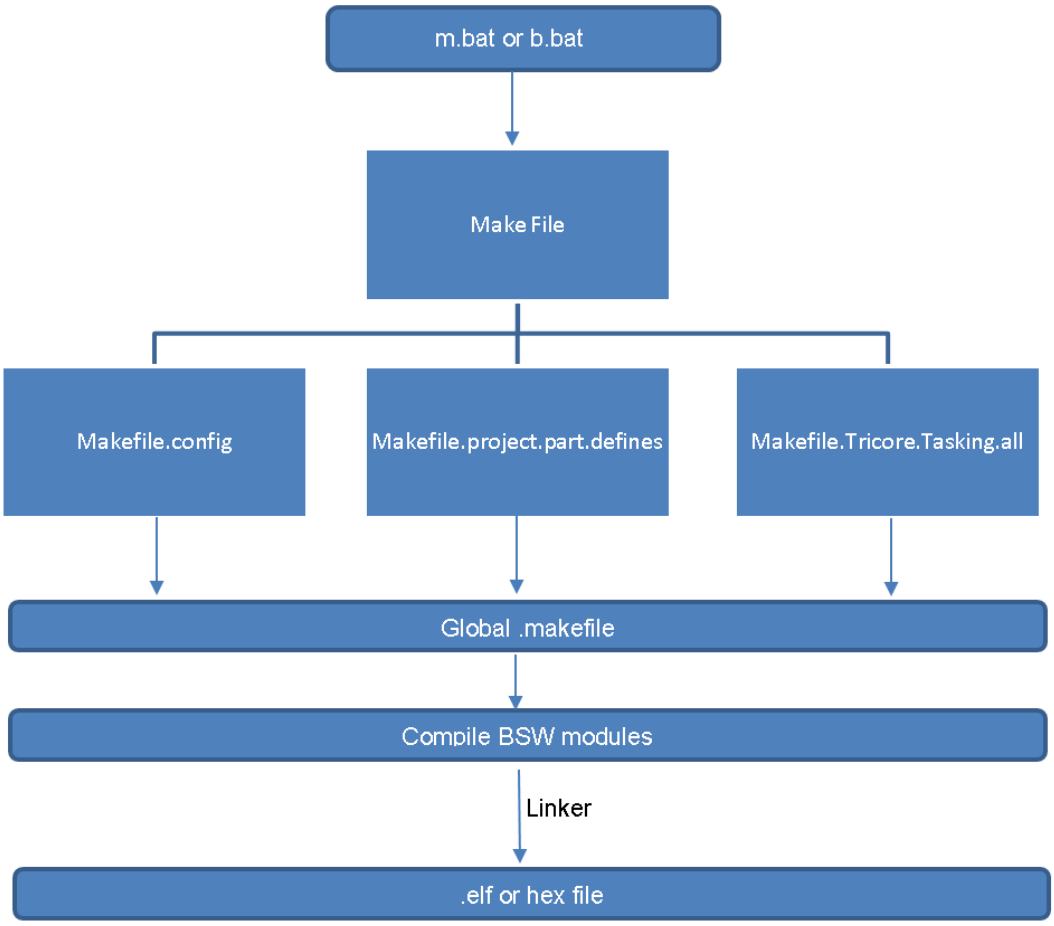



Figure 5.13: Overall Build Environment setup

[AUR13]. Other derivatives for the Infineon Aurix series are the *TC21x*, *TC22x*, *TC23x* and the *TC29x* series.

### Makefile.project.part.defines

*BSW* module specific compilation settings were exclusively defined in this file. In a multi-feature environment such as an AUTOSAR system, only certain modules or certain source files be used for a particular build. For example in my thesis only Ethernet communication is desired for the ECU. So all the software components relevant to other bus systems was to be avoided.

Including all the irrelevant components only serves to increase the compilation time and also the size of the executables. Hence a mechanism was devised for a selective

build environment with which it is possible to pick and choose source files or modules. With the use of macros we can enable/disable the relevant *BSW* modules.

- Step 1: Set the path to the root of source files directory
- Step 2: Set the path to the generated source files from the configuration process
- Step 3: Enable/Disable the macros for each *BSW* module.
- Step 4: Set the path to the individual *BSW* module makefiles.In the case of the third party *MCAL* modules, a path to the central *MCAL* makefile was defined. This makefile then does the selection of *MCAL* modules.
- Step 5: Set the path to the additional files for necessary for compilation. These are the generated files which are not related to any of the *BSW* modules. These consist of the *RTE* file, bootstrapping file and the *SOME/IP* transformer source file. The RTE file contains all the Operating system related information such as the tasks, runnables, alarms and events. The bootstrapping file contains the source code whose instructions are executed immediately upon power-on of the ECU. The *SOME/IP* transformer is a generated file that converts the payload data from the applications into the *SOME/IP* data that is required as per the AUTOSAR standards for Ethernet communication.
- Step 6: Set the path to the *SWC* files and *BSW* callout files for board bring up.

**Makefile.Tricore.Tasking.All.Make**

This is responsible for generating the hardware related linker options. Individual address space for system reset, interrupt vector table, stack and heap size were configured here.

**Global.Makefile.target.make**

This file invokes the compilation of individual *BSW* modules. It makes use of the data available in the *Makefile.project.part.defines* file for selection of *BSW* modules. This specific file was already provided by Vector and need not be adapted. For each *BSW* module, AUTOSAR standardised individual build environment is available. These were invoked from this particular file.

## 5.8 Execution strategies for Existing Test Specifications

At Bertrandt AG, a basic test specification to evaluate the AUTOSAR Ethernet stack is available. It covers all the relevant modules of the stack. The final step of this thesis investigates the strategies for execution of the test specification. The nature of the test specification is such that all the test cases cannot be covered with a unique approach. Some tests require detailed analysis of the ongoing Ethernet communication, while some others require the intervention of a tester to examine the current state by temporarily interrupting the communication. Another group of test cases does not even require manual testing, since a proper communication exchange indicates the successful execution of the test. Based on this analysis of the entire test specification, the execution strategies can be classified into three methods:

**Source code analysis:**

In certain test cases, analysis of the source code reveals that a successful communication exchange between the TriCore board and Simulator ECU as an indication of a successfully passed Test case. For example consider the following test case for the module Tcp/Ip:

Test Case: *Request ip address assignment*; Expected result: Initiates local ip address assignment for ip address specified by LocalAddrId.

Analysis of Tcp/IP module implementation source code reveals that the ip address assignment is one of the initial steps during the activation of the Tcp/Ip stack. If the request fails, then there will not be any communication incoming from the TriCore board to the Simulator PC. Evidence of communication between the two devices is thus a clear indication of this test being passed.

**CAPL code analysis of the received messages from the TriCore board**

Rest bus simulation with CAPL also allows the validation of incoming messages using a suitable assert mechanism against the expected values. This is done with CAPL programming, to implement a suitable function for checking the recieved values. In this way the communication exchanges between the board and test PC is validated.

### Using code debuggers

Code debuggers allow users to simply iterate through each line of the code or jump through function calls and inserting breakpoints. They also allow the user to view the call stack of a function as well as checking the current state of a variable when a certain condition has occurred. One such debugger is the UDE debugger. It is an open source tool compatible with the TriCore board [Dev]. For example consider the following test case for the module Ethernet State Manager:

Test Case: Transition from *EthSM_Wait_Trcvlink* to *EthSM_State_Online*; Expected result: The state must transition from *Wait_TrcvLink* to *State_Online*. The State manager transitions can be easily tracked with breakpoints in the source code and checking the current state value every time a transition occurs.

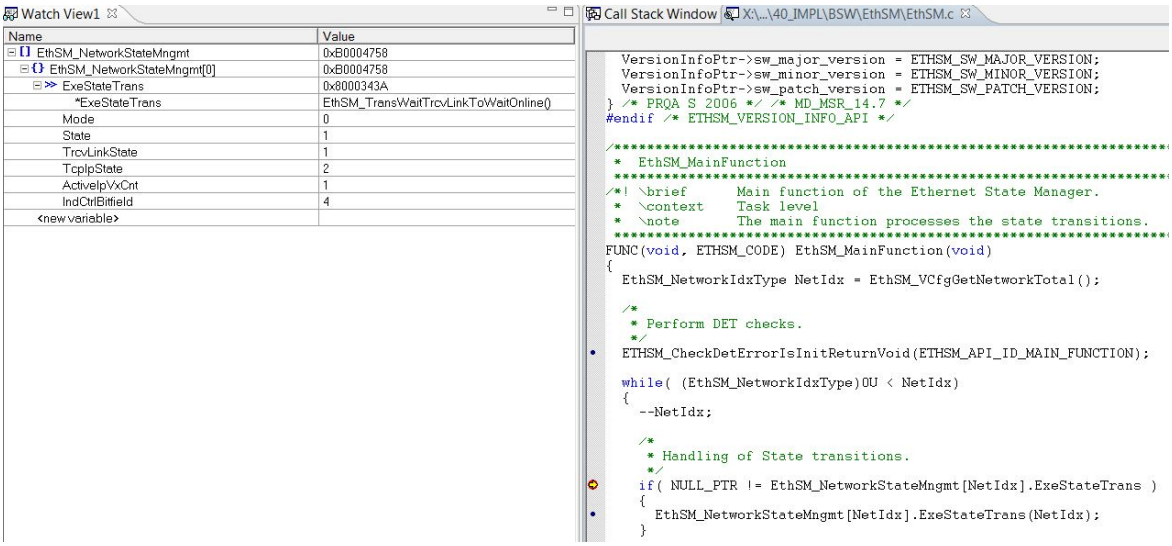



Figure 5.14: Validation of Ethernet State Manager transitions with the UDE Debugger

# 6 Results

This chapter focuses on the expected outputs from this thesis work. They are organized into two sections. The first section deals with the realization and the subsequent validation of Ethernet communication on the physical layer between the Infineon board running under AUTOSAR software and the Vector VN5610 simulator ECU. The different test approaches for the existing Bertrandt test specification for the AUTOSAR 4.2 Ethernet stack were evaluated and these are illustrated in the final section of this chapter.

## 6.1 Validation of Ethernet communication

With the Vector CANoe software, it is possible to analyze the Ethernet traffic between the connected devices on a test PC. The incoming data from the Infineon board as well as the outgoing data from the simulator are captured in a tracefile. The resulting data was analyzed both visually and using offline analysis techniques in the CANoe software with CAPL codes. For instance, the received payload can be validated with the offline analysis method, by comparing the payload with the expected values and using assertion methods to notify the result of validation to the user.

The validation of the communication was done with the visual analysis of the tracefile for the various connection mechanisms (TCP, UDP) between the Infineon board and the simulator ECU. The Infineon board was configured with a MAC address of *EE:EE:EE:EE:EE:EE*. The simulator ECU for TCP connection was configured with a MAC address of *02:58:31:5E:06:00* and for UDP connection with *02:62:1E:7A:11:00*.

### TCP connection

To achieve a TCP connection between the two devices, corresponding configurations were created for the AUTOSAR compliant ECU and the simulator ECU. TCP connections involve a three-way handshake mechanism between the devices before data communication is possible. In the following illustrations, the Infineon board sends a connection request to the awaiting simulator ECU.

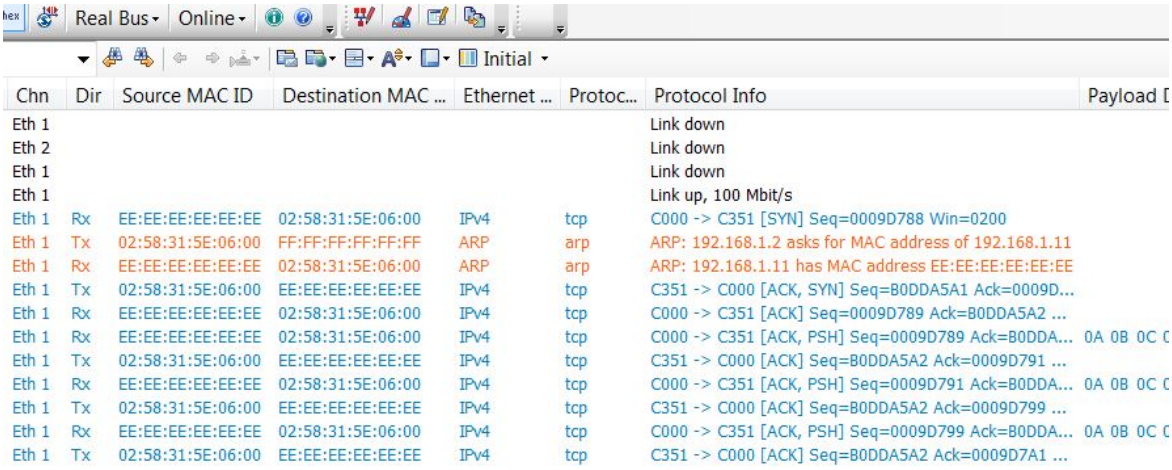



Figure 6.1: TCP connection with half-duplex communication

Figure 6.1 shows the tracefile captured from CANoe for a TCP half duplex connection between the Infineon board and the simulator. The TriCore board was configured with static ARP table with socket configured as Initiator. Hence it straight away sends a connection request to the TCP node of simulator ECU. The simulator does not have static ARP tables. Hence upon receiving a message, it replies with an ARP packet. The TriCore board upon seeing this ARP message from the simualtor replies with its MAC address. As ARP resolution has turned succesful for the simulator ECU, it acknowledges the previously recieved connection request from the TriCore board. The board then sends its acknowledgement back. Now the TCP 3 way handshake mechanism is complete and the two devices start communication exchanges. Figure 6.2 shoes the message sequence chart for the TCP 3 way handshake between two devices.

For the communication validation, the TriCore board was setup to transmit messages as soon as the socket connection is established. The simulator ECU was not configured to transmit any messages, Hence it keeps sending acknowledgement whenever a message from the board is received.

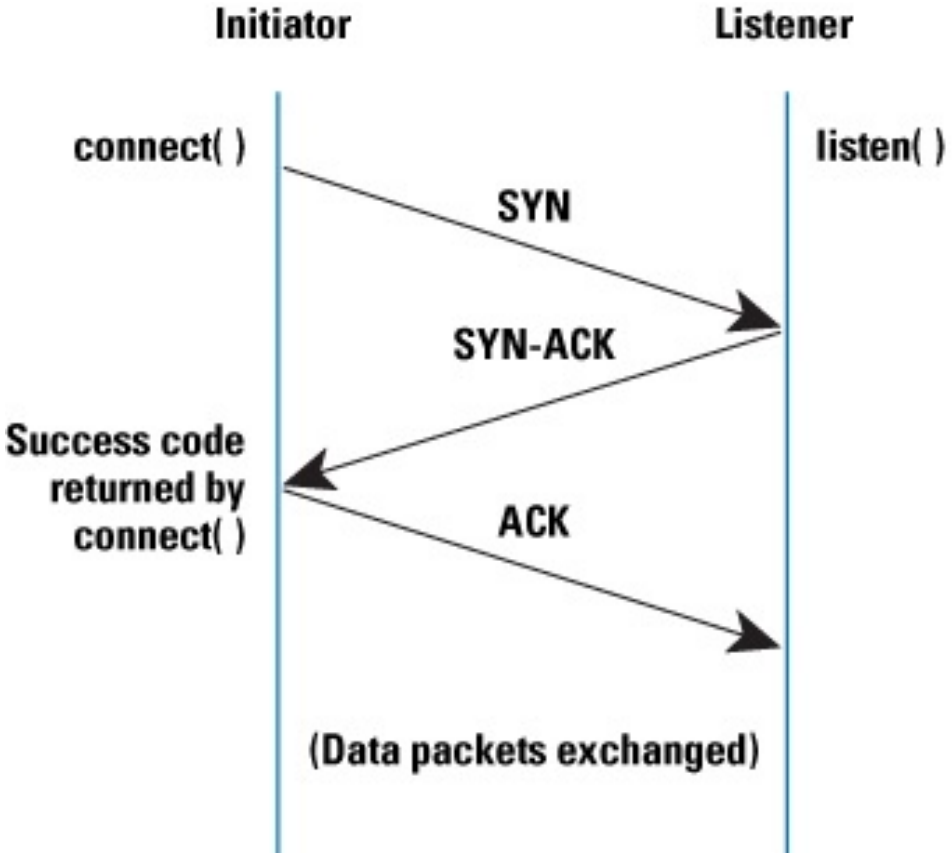
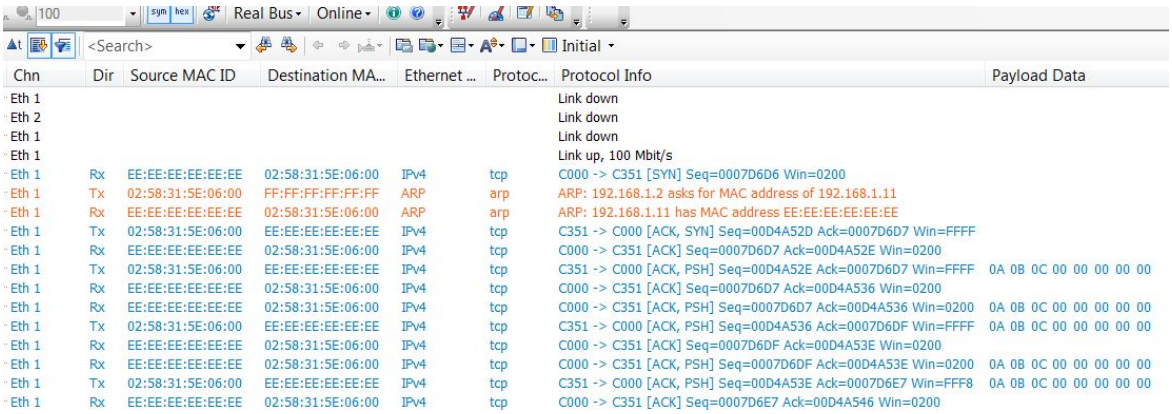



Figure 6.2: TCP 3 Way Handshake

Figure 6.3 shows the tracefile captured from CANoe for a TCP full duplex connection between the Infineon board and the simulator.

This is a similar situation as the previously explained validation scheme. In this particular case, the simulator ECU was setup to transmit messages as soon as a connection request with the TriCore board is established. The application code on the board was designed in such a way that as soon as a message is received from the Simulator, a loopback implemented on the TriCore's SWC, whereby the received message is immediately transmitted back to Simulator. The loopback was implemented with a simple SWC design that reads the data from the Receiver port, copies it into the Sender Port and triggered for communication.

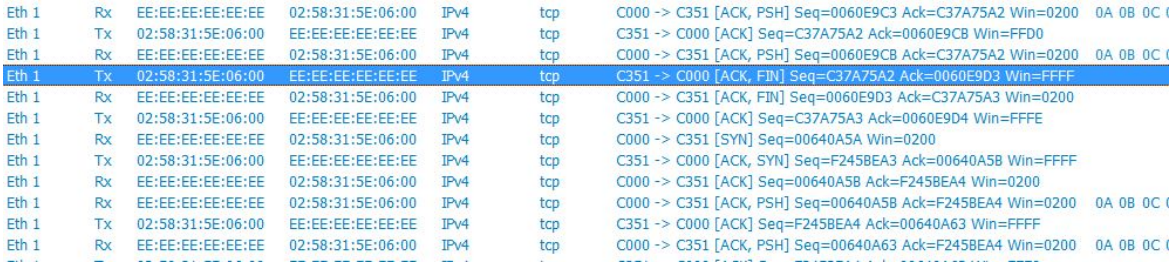
The validation concept involved a CAPL code function on the Simulator ECU that reads the incoming data from the board and compares it with the expected values.



| Chn | Dir | Source MAC ID | Destination MA... | Ethernet ... | Protoc... | Protocol Info | Payload Data |
|---|---|---|---|---|---|---|---|
| Eth 1 | | | | | | Link down | |
| Eth 2 | | | | | | Link down | |
| Eth 1 | | | | | | Link down | |
| Eth 1 | | | | | | Link up, 100 Mbit/s | |
| Eth 1 | Rx | EE:EE:EE:EE:EE:EE | 02:58:31:5E:06:00 | IPv4 | tcp | C000 -> C351 [SYN] Seq=0007D6D6 Win=0200 | |
| Eth 1 | Tx | 02:58:31:5E:06:00 | FF:FF:FF:FF:FF:FF | ARP | arp | ARP: 192.168.1.2 asks for MAC address of 192.168.1.11 | |
| Eth 1 | Rx | EE:EE:EE:EE:EE:EE | 02:58:31:5E:06:00 | ARP | arp | ARP: 192.168.1.11 has MAC address EE:EE:EE:EE:EE:EE | |
| Eth 1 | Tx | 02:58:31:5E:06:00 | EE:EE:EE:EE:EE:EE | IPv4 | tcp | C351 -> C000 [ACK, SYN] Seq=00D4A52D Ack=0007D6D7 Win=FFFF | |
| Eth 1 | Rx | EE:EE:EE:EE:EE:EE | 02:58:31:5E:06:00 | IPv4 | tcp | C000 -> C351 [ACK] Seq=0007D6D7 Ack=00D4A52E Win=0200 | |
| Eth 1 | Tx | 02:58:31:5E:06:00 | EE:EE:EE:EE:EE:EE | IPv4 | tcp | C351 -> C000 [ACK, PSH] Seq=00D4A52E Ack=0007D6D7 Win=FFFF | 0A 0B 0C 00 00 00 00 00 |
| Eth 1 | Rx | EE:EE:EE:EE:EE:EE | 02:58:31:5E:06:00 | IPv4 | tcp | C000 -> C351 [ACK] Seq=0007D6D7 Ack=00D4A536 Win=0200 | |
| Eth 1 | Rx | EE:EE:EE:EE:EE:EE | 02:58:31:5E:06:00 | IPv4 | tcp | C000 -> C351 [ACK, PSH] Seq=0007D6D7 Ack=00D4A536 Win=0200 | 0A 0B 0C 00 00 00 00 00 |
| Eth 1 | Tx | 02:58:31:5E:06:00 | EE:EE:EE:EE:EE:EE | IPv4 | tcp | C351 -> C000 [ACK, PSH] Seq=00D4A536 Ack=0007D6DF Win=FFFF | 0A 0B 0C 00 00 00 00 00 |
| Eth 1 | Rx | EE:EE:EE:EE:EE:EE | 02:58:31:5E:06:00 | IPv4 | tcp | C000 -> C351 [ACK] Seq=0007D6DF Ack=00D4A53E Win=0200 | |
| Eth 1 | Rx | EE:EE:EE:EE:EE:EE | 02:58:31:5E:06:00 | IPv4 | tcp | C000 -> C351 [ACK, PSH] Seq=0007D6DF Ack=00D4A53E Win=0200 | 0A 0B 0C 00 00 00 00 00 |
| Eth 1 | Tx | 02:58:31:5E:06:00 | EE:EE:EE:EE:EE:EE | IPv4 | tcp | C351 -> C000 [ACK, PSH] Seq=00D4A53E Ack=0007D6E7 Win=FFF8 | 0A 0B 0C 00 00 00 00 00 |
| Eth 1 | Rx | EE:EE:EE:EE:EE:EE | 02:58:31:5E:06:00 | IPv4 | tcp | C000 -> C351 [ACK] Seq=0007D6E7 Ack=00D4A546 Win=0200 | |

Figure 6.3: TCP connection with full-duplex communication

Figure 6.4 shows the tracefile captured from CANoe for a TCP connection termination between the Infineon board and the simulator.
The Simulator ECU was configured to close all socket connections after a few seconds. This connection termination is done by Simulator by sending the (FIN,ACK) message to the TriCore board. The board replies it with its own (FIN,ACK). This is then acknowledged by the Simulator. The connection is now terminated. The board upon seeing a terminated connection, resends a connection request and again a three way TCP handshake begins, at the end of which connection is once again established.



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Eth 1 | Rx | EE:EE:EE:EE:EE:EE | 02:58:31:5E:06:00 | IPv4 | tcp | C000 -> C351 [ACK, PSH] Seq=0060E9C3 Ack=C37A75A2 Win=0200 | 0A 0B 0C |
| Eth 1 | Tx | 02:58:31:5E:06:00 | EE:EE:EE:EE:EE:EE | IPv4 | tcp | C351 -> C000 [ACK] Seq=C37A75A2 Ack=0060E9CB Win=FFD0 | |
| Eth 1 | Rx | EE:EE:EE:EE:EE:EE | 02:58:31:5E:06:00 | IPv4 | tcp | C000 -> C351 [ACK, PSH] Seq=0060E9CB Ack=C37A75A2 Win=0200 | 0A 0B 0C |
| Eth 1 | Tx | 02:58:31:5E:06:00 | EE:EE:EE:EE:EE:EE | IPv4 | tcp | C351 -> C000 [ACK, FIN] Seq=C37A75A2 Ack=0060E9D3 Win=FFFF | |
| Eth 1 | Rx | EE:EE:EE:EE:EE:EE | 02:58:31:5E:06:00 | IPv4 | tcp | C000 -> C351 [ACK, FIN] Seq=0060E9D3 Ack=C37A75A3 Win=0200 | |
| Eth 1 | Tx | 02:58:31:5E:06:00 | EE:EE:EE:EE:EE:EE | IPv4 | tcp | C351 -> C000 [ACK] Seq=C37A75A3 Ack=0060E9D4 Win=FFFE | |
| Eth 1 | Rx | EE:EE:EE:EE:EE:EE | 02:58:31:5E:06:00 | IPv4 | tcp | C000 -> C351 [SYN] Seq=00640A5A Win=0200 | |
| Eth 1 | Tx | 02:58:31:5E:06:00 | EE:EE:EE:EE:EE:EE | IPv4 | tcp | C351 -> C000 [ACK, SYN] Seq=F245BEA3 Ack=00640A5B Win=FFFF | |
| Eth 1 | Rx | EE:EE:EE:EE:EE:EE | 02:58:31:5E:06:00 | IPv4 | tcp | C000 -> C351 [ACK] Seq=00640A5B Ack=F245BEA4 Win=0200 | |
| Eth 1 | Rx | EE:EE:EE:EE:EE:EE | 02:58:31:5E:06:00 | IPv4 | tcp | C000 -> C351 [ACK, PSH] Seq=00640A5B Ack=F245BEA4 Win=0200 | 0A 0B 0C |
| Eth 1 | Tx | 02:58:31:5E:06:00 | EE:EE:EE:EE:EE:EE | IPv4 | tcp | C351 -> C000 [ACK] Seq=F245BEA4 Ack=00640A63 Win=FFFF | |
| Eth 1 | Rx | EE:EE:EE:EE:EE:EE | 02:58:31:5E:06:00 | IPv4 | tcp | C000 -> C351 [ACK, PSH] Seq=00640A63 Ack=F245BEA4 Win=0200 | 0A 0B 0C |

Figure 6.4: TCP termination and re-connection

Figure 6.5 shows the message sequence chart for the TCP connection termination procedure.

Figure 6.6 shows the tracefile captured from CANoe for a connection without statically configured ARP between the Infineon board and the simulator. The Tricore board does not have a static ARP table. Thus before sending a connection request, it issues a broadcast message asking for the MAC address of the Simulator ECU. The simulator

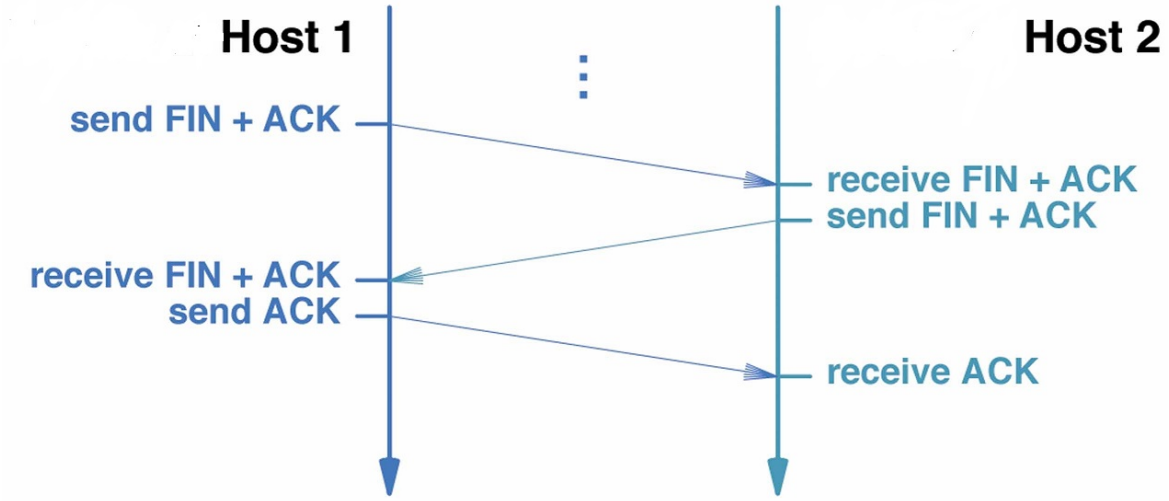



Figure 6.5: TCP connection termination

then responds to the ARP request, with its MAC address. Now ARP resolution is successful for the TriCore board. Hence it immediately issues a connection request to the simulator. Now 3 way handshake begins and upon its completion, normal communication is established.

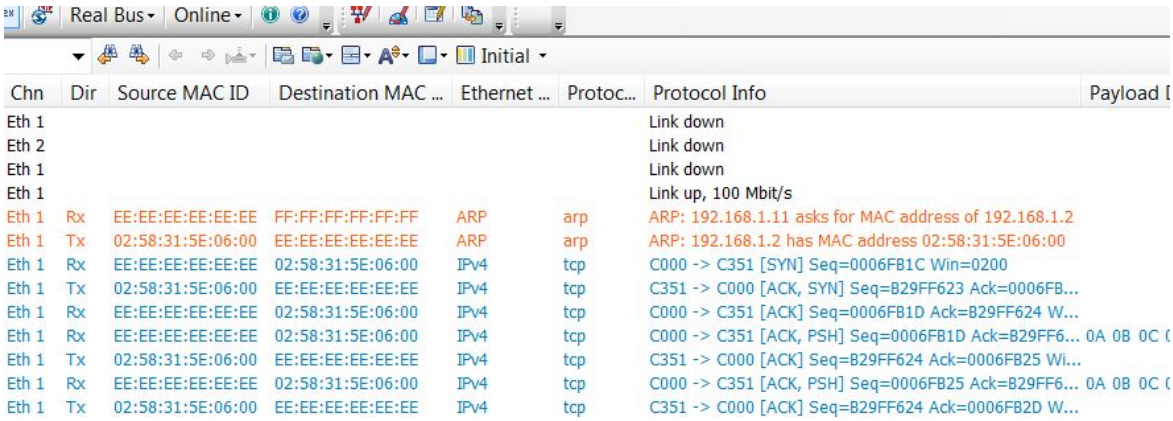



Figure 6.6: ARP request before connection initiation

Figure 6.7 shows the tracefile captured from CANoe for a UDP connection between the TriCore board and the simulator. The UDP connection is initiated by the TriCore board. ARP static table is not configured on the board for this configuration. Hence it first issues ARP message. Upon its successful resolution, connection is established between TriCore board and simulator.

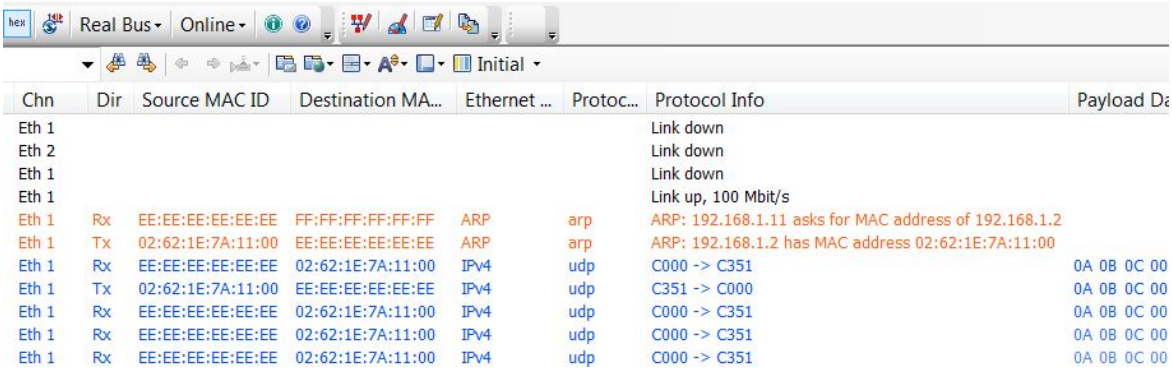



Figure 6.7: UDP connection

In this way, various configurations of TCP and UDP have been tested and validated to ensure the correctness of the implementation on the TriCore board.

## 6.2 Execution of Test Specification

The Test specification was successfully passed with the test execution being done with various approaches as explained in the previous chapter. The strategies for evaluation of the AUTOSAR Ethernet stack is laid with this result. In the following sections, for each Ethernet stack module, the test cases are grouped under a feasible test execution methodology

### 6.2.1 Ethernet Driver

Methodology: **Source code Analysis**

1. **Eth Controller Initialization**: Initialization of the indexed Ethernet Controller
2. **Write Mii**: Writes a specific transceiver register with the given value through the MII of the indexed controller
3. **Read Mii**: Reads a specific transceiver register through the MII of the indexed controller
4. **Provide TX Buffer**: Provides a transmit buffer resource and lock the resource until a subsequent call of the Transmit service
5. **Transmit**: Builds the Ethernet header with the given target MAC address and triggers the transmission if a previously filled transmit buffer
6. **Receive**: Triggers frame reception and indicates if there are more frames to receive

Methodology: **Code debugger**

1. **TX Confirmation**: Triggers frame transmission confirmation.

### 6.2.2 Ethernet Interface

Methodology: **Code Debugger**

1. **Eth Controller Initialization**: Forward of the Call for Initialization of the Eth Controller during Communication Initialization

2. **Set Controller Mode**: Forward of the Call for setting the Controller mode of the respective Eth Controller

3. **Provide TX Buffer**: Forward of the Call for providing access to a transmit Buffer to the respective Eth Controller

4. **Transmit**: Forward of the Call for triggering the transmission of a transmit Buffer to the respective Eth Controller

5. **Transceiver Initialization**: Forward of the Call for Initialization of the Eth Transceiver during Communication Initialization

6. **RX Indication**: Handles a received frame received by the respective controller and indicates the reception of a frame to the Upper layer.

7. **TX Confirmation**: Confirms the frame transmission by the respective controller and confirms the transmission to the Upper layer.

8. **Link State Change**: Indicates the change of a transceiver state

### 6.2.3 Tcp/Ip

Methodology: **Source code Analysis**

1. **Bind**: Binds a TCP or UDP socket to a local resource

2. **Close**: Closes socket and releases all related resources.

3. **TCP Connect**: Establishes a TCP connection to configured peer

4. **Request IP Address Assignment**: Initiates local IP address assignment for IP address specified by LocalAddrId

5. **TCP Transmit**: Requests transmission of data via TCP to a remote node

6. **UDP Tranmsit**: Transmits data via UDP to a remote node

Methodology: **Code Debugger**

1. **TCP Received**: Confirms reception of socket data to TcpIp stack.

2. **Request Com Mode**: Changes TcpIp state of communication network identified by EthIf controller index

### 6.2.4 Socket Adaptor

Methodology: **Source code Analysis**

1. **Socket Connection Open**: SoAd shall try to open each socket connection.

2. **Socket Connection Close**: SoAd shall try to close each socket connection.

3. **PDU Transmission via IF-API**: For the transmission of a PDU requested by an upper layer using the IF-API.

4. **PDU Reception via IF-API**: For the reception of a PDU requested by an upper layer using the IF-API.

Methodology: **CAPL analysis**

1. **PDU FanOut**: Multi-client handling requires a flexible PDU fan-out in the Socket Adaptor, i.e. one PDU is dynamically sent on multiple Socket Connections

### 6.2.5 Ethernet State Manager

In the case of the Ethernet State manager, all the tests are concerned with the verification of the state diagram and the state change transitions of the module. These are possible only with the Code Debugger method.

# 7 Conclusion and Future Outlook

This chapter summarizes the contents of this thesis work and also looks into the potential enhancements that may be implemented in the future along with any shortcomings that could not be addressed during the course of this work.

## 7.1 Conclusion

This thesis covers the implementation of Ethernet communication running under AUTOSAR specification on an Infineon Aurix Tricore board. With Ethernet being a crucial future technology in vehicle systems, several new technologies have been introduced in it in order to pass the requirements of the automotive domain. The conventional bus systems only allow the implementation of a broadcast messaging mechanism with data sent on a bus being available to all the ECUs connected to it. Ethernet has the potential to rewrite the broadcast paradigms with its exclusive point -to- point addressing schemes. It also introduces new application layer protocols such as SOME/IP, DoIP and AVB. Vector had made available the Automotive Ethernet stack corresponding to the latest AUTOSAR specification 4.2. This stack was custom built for the Infineon Aurix TriCore Tc27x board. The TriCore board supports Fast Ethernet. This thesis has successfully realized the AUTOSAR compliant Automotiv Ethernet communication on the TriCore board. The classic case of AUTOSAR application SWC triggered communication with signals and PDUs has been covered in this thesis. The existing Ethernet stack test suite at Bertrandt covers the various layers of the Ethernet OSI model. Three different approaches were examined and these were sufficient to execute the test suite. This thesis has thus laid the groundwork for evaluation of future releases or improvements in Vector's AUTOSAR Ethernet stack made available to Bertrandt.

## 7.2 Future Outlook

- Higher layer application protocols such as SOME/IP, DoIP or even AVB could now be evaluated since a working AUTOSAR compliant Automotive Ethernet software was realized for the Infineon board during the course of this thesis.

- 100BASE-TX formed the physical layer for the Ethernet communication on the TriCore board. The board does not have current support for use with the BroadR-Reach physical layer. The Lantiq PEF 7071 Fast Ethernet Transceiver on the TriCore board is not compliant with the Open Alliance BroadR-Reach physical layer specification. This situation can be improved in the future by replacing the existing transceiver on the board with the readily available BroadR-Reach compliant transceivers such as the TJA1100HN PHY from NXP semiconductors or the BCM89810 PHY supplied by Broadcom [Brob], [NXP]. On the software side, the existing AUTOSAR Ethernet stack may be reused with suitable modifications to the Ethernet Transceiver driver in its initialization routine.

- Ethernet switch driver is now becoming a part of the AUTOSAR specification. With future predictions on Ethernet becoming the backbone architecture in cars, switched networks may play a big role in cars. It may be possible to control the bandwidths of the physical layer by connected the various sub-domains to the Ethernet switches. Thus the feasibility of the switch driver for use on the TriCore board can be evaluated

# Abbreviations

| | |
|---|---|
| **ADAS** | Advanced Driver Assistance System |
| **ARXML** | AUTOSAR XML |
| **AUTOSAR** | AUTomotive Open System ARchitecture |
| **AVB** | Audio Video Bridging |
| **BSWMD** | Basic Software Module Description |
| **CAN** | Controller Area Network |
| **CANFD** | CAN with Flexible data Rate |
| **CAPL** | CAN Access Programming Language |
| **CRC** | Cyclic Redundancy Check |
| **CSMA/CD** | Carrier Sense Multiple Access with Collision Detection |
| **DOIP** | Diagnostics over Internet Protocol |
| **ECU** | Electronic Control Unit |
| **ECUAL** | ECU Abstraction Layer |
| **ECUC** | ECU Configuration |
| **E/E** | Electric/Electronic |
| **EMC** | Electro Magnetic Compatibility |
| **ETH** | Ethernet |
| **ETHTRCV** | Ethernet Transceiver |
| **ETHIF** | Ethernet Interface |
| **FTP** | File Transfer Protocol |
| **HTTP** | HyperText Transfer Protocol |
| **IP** | Intellectual Property |
| **ISO** | International Organization for Standardization |
| **ISR** | Interrupt Service Routine |
| **LIN** | Local Interconnect Network |
| **MAC** | Media Access Control |
| **MCAL** | Microcontroller Abstraction Layer |
| **MII** | Media Independent Interface |
| **MOST** | Media Oriented System Transport |
| **OEM** | Original Equipment Manufacturer |
| **Open Alliance SIG** | OPEN (One-Pair Ether-Net) Alliance Special Interest Group |

| | |
|---|---|
| **OS** | Operating System |
| **OSI** | Open Systems Interconnection |
| **PDU** | Protocol Data Unit |
| **RMII** | Reduced Media Independent Interface |
| **RTE** | Run Time Environment |
| **SDU** | Service Data Unit |
| **SFR** | Special Function Register |
| **SD** | Service Discovery |
| **SMTP** | Simple Mail Transfer Protocol |
| **SOAD** | Socket Adaptor |
| **SOME/IP** | Scalable service-Oriented MiddlewarE over IP |
| **SWC** | Software Component |
| **TCP** | Transmission Control Protocol |
| **TDMA** | Time Division Multiple Access |
| **UDP** | User Datagram Protocol |
| **VFB** | Virtual Function Bus |
| **XML** | EXtensible Markup Language |

# List of Figures

# List of Tables

# Bibliography

[AD10] A.Tanenbaum and D.Wetherall. *Computer Networks 5th edition*. Pearson, 2010.

[Rei10] Konrad Reif. *Batterien, Bordnetze und Vernetzung 1.Auflage*. Vieweg + Teubner Verlag, 2010.

[LAN12] LANTIQ. *Single Port Gigabit Ethernet PHY (10/100/1000 Mbit/s)PEF 7071, Version 1.5*. Lantiq Deutschland GmbH, 2012.

[AUR13] INFINEON AURIX. *TC27x C-Step 32-bit Single-Chip Microcontroller User's Manual v1.5*. Infineon Technologies AG, 2013.

[Kiz14] J.M. Kizza. *Computer Network Security and Cyber Ethics*. McFarland and Company, Inc., 2014.

[WR14] W.Zimmermann and R.Schmidgall. *Bussysteme in der Fahrzeugtechnik*. Springer Fachmedien, 2014.

[AUR15a] INFINEON AURIX. *AURIX Microcontroller Infineon Software Architecture User's Manual: DIO Driver v8.1*. Infineon Technologies AG, 2015.

[AUR15b] INFINEON AURIX. *AURIX Microcontroller Infineon Software Architecture User's Manual: MCU Driver v4.1*. Infineon Technologies AG, 2015.

[AUR15c] INFINEON AURIX. *AURIX Microcontroller Infineon Software Architecture User's Manual: PORT Driver v4.3*. Infineon Technologies AG, 2015.

[EFl15] E.Flemming. "Ethernet". In: *Elektronik automotive* (2015).

[Für15] Simon Fürst. "AUTOSAR the Next Generation – The Adaptive Platform". In: (2015).

[KT15] K.Matheus and T.Koenigseder. *Automotive Ethernet*. Cambridge university Press, 2015.

[42a] AUTOSAR Release 4.2. *AUTOSAR SWS ECUStateManager*. URL: http://www.autosar.org/fileadmin/files/releases/4-2/software-architecture/system-services/standard/AUTOSAR_SWS_ECUStateManager.pdf.

[42b] AUTOSAR Release 4.2. *AUTOSAR SWS EthernetDriver*. URL: https://www.autosar.org/fileadmin/files/releases/4-2/software-architecture/communication-stack/standard/AUTOSAR_SWS_EthernetDriver.pdf.

[42c] AUTOSAR Release 4.2. *AUTOSAR SWS MCUDriver*. URL: http://www.autosar.org/fileadmin/files/releases/4-2/software-architecture/peripherals/standard/AUTOSAR_SWS_MCUDriver.pdf.

[42d] AUTOSAR Release 4.2. *AUTOSAR SWS SocketAdaptor*. URL: https://www.autosar.org/fileadmin/files/releases/4-2/software-architecture/communication-stack/standard/AUTOSAR_SWS_SocketAdaptor.pdf.

[42e] AUTOSAR Release 4.2. *AUTOSAR SWS TcpIp*. URL: https://www.autosar.org/fileadmin/files/releases/4-2/software-architecture/communication-stack/standard/AUTOSAR_SWS_TcpIp.pdf.

[All] Open Alliance. *One-Pair Ether-Net Special Interest Group (SIG)*. URL: https://www.opensig.org/home/.

[Alt] Altium. *Tasking VX-toolset for TriCore User Guide v4.2*.

[AUTa] AUTOSAR. URL: http://www.autosar.org/partners/current-partners/http://www.autosar.org/.

[AUTb] AUTOSAR. URL: http://www.autosar.org/about/faq/general/.

[AUTc] AUTOSAR. *AUTOSAR Home Page*. URL: http://www.autosar.org/.

[AUTd] AUTOSAR. *Layered Software Architecture*. URL: http://www.autosar.org/fileadmin/files/releases/4-2/software-architecture/general/auxiliary/AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf.

[Blo] Connected Blog:Broadcom. *BroadR-Reach Ethernet: Enterprise-Level Security for Connected Cars*. URL: http://www.broadcom.com/blog/automotive-technology-2/broadr-reach-ethernet-enterprise-level-security-for-connected-cars/.

[Broa] Broadcom. *Automotive Market Trends and Challenges*. URL: http://jrjcommunications.com/wp-content/uploads/2012/08/Broadcom_Automotive_Media_Presentation.pdf.

[Brob] Broadcom. *BroadR-Reach Single-Port Automotive Ethernet Transceiver*. URL: https://www.broadcom.com/products/ethernet-communication-and-switching/physical-layer/bcm89810.

[CKo] C.Kozierok. *TCP Functions. What TCP does*. URL: http://www.tcpipguide.com/free/t_TCPFunctionsWhatTCPDoes.htm.

[Com] Wikimedia Commons. *S/UTP cable*. URL: https://commons.wikimedia.org/wiki/File:S-UTP-cable.svg.

[COO] MOST COOPERATION. URL: http://www.mostcooperation.com//.

[Dev] pls Development Tools. *Universal Debug Engine*. URL: http://www.pls-mc.com/.

[Fri] Mario Kindel Olaf ; Friedrich. "Softwareentwicklung mit AUTOSAR". In: *dpunkt.verlag GmbH* (), p. 2009.

[Gmb] Vector Informatik GmbH. *Product Information MICROSAR, v1.4, May 2015*.

[Heb] Regina Hebig. *Methodology and Templates in AUTOSAR*. URL: http://hpi.de/fileadmin/user_upload/fachgebiete/giese/Ausarbeitungen_AUTOSAR0809/Methodology_hebig.pdf.

[IXI] IXIACOM. *Automotive Ethernet: An Overview*.

[NXP] NXP. *TJA1100HN: OPEN Alliance BroadR-Reach Automotive Ethernet PHY*. URL: http://www.nxp.com/products/interface-and-connectivity/wired-connectivity/ethernet/open-alliance-broadr-reach-automotive-ethernet-phy:TJA1100HN.

[Refa] Vector Technical Reference. *MCAL Integration*.

[Refb] Vector Technical Reference. *MICROSAR Ethernet Driver*.

[Refc] Vector Technical Reference. *MICROSAR Ethernet Interface*.

[Refd] Vector Technical Reference. *MICROSAR Ethernet Transceiver Driver*.

[Sch] Dr. Stefan Schmerler. "AUTOSAR – Shaping the Future of a Global Standard". In: ().

[Tec] Infineon Technologies. *Highly integrated and performance optimized 32-bit microcontrollers for automotive and industrial applications*. URL: http://www.infineon.com/dgdl/Infineon-New+Tricore+Family+Brochure-BC-v01_00-EN.pdf?fileId=db3a30431f848401011fc664882a7

[VEC] VECTOR. *Introduction to Ethernet and IP in automotive vehicles*. URL: http://vector.com/portal/medien/cmc/events/Webinars/2015/Vector_Webinar_IP-Ethernet_Basics_20150506_EN.pdf.

[Wika] Wikipedia. *Media-independent interface*. URL: https://en.wikipedia.org/wiki/Media-independent_interface.

[Wikb] Wikipedia. *OSI model*. URL: https://en.wikipedia.org/wiki/OSI_model.