

CHALMERS



FlexRay Redundancy in AUTOSAR

An investigation on how FlexRay redundancy can be integrated into AUTOSAR

Master of Science Thesis in the Programmes Secure and Dependable Computer Systems and Networks and Distributed Systems

ALEXANDER SÖDERLUND
MÅRTEN FREDRIKSSON

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, June 2012

The Author grants to Chalmers University of Technology and University of Gothenburg the non-exclusive right to publish the Work electronically and in a non-commercial purpose make it accessible on the Internet.

The Author warrants that he/she is the author to the Work, and warrants that the Work does not contain text, pictures or other material that violates copyright law.

The Author shall, when transferring the rights of the Work to a third party (for example a publisher or a company), acknowledge the third party about this agreement. If the Author has signed a copyright agreement with a third party regarding the Work, the Author warrants hereby that he/she has obtained any necessary permission from this third party to let Chalmers University of Technology and University of Gothenburg store the Work electronically and make it accessible on the Internet.

FlexRay Redundancy in AUTOSAR

An investigation on how FlexRay redundancy can be integrated into AUTOSAR

ALEXANDER SÖDERLUND

MÅRTEN FREDRIKSSON

© ALEXANDER SÖDERLUND, June 2012.

© MÅRTEN FREDRIKSSON, June 2012.

Examiner: **Roger Johansson**

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden June 2012

Abstract

In the past decades the number of ECUs in vehicles has increased rapidly. With this increase comes a higher demand for high speed networks to interconnect the ECUs as well as higher demand on reliability and safety as electronics handle more and more safety critical applications within the vehicles. In order to meet all these demands the automotive industry has developed a serial communication protocol, FlexRay, which has features that makes it suitable for safety related applications and high-speed networking. The industry has also agreed upon a standard automotive software platform, AUTOSAR, to make software hardware independent, more scalable, more modular and easier to maintain.

This report describes an investigation of how the FlexRay redundancy concept can be integrated into the AUTOSAR software architecture as a way to increase the reliability and safety of an electronic system. The chosen solution design has been implemented in a prototype steer-by-wire system running AUTOSAR 3.1.4. The result shows that it is possible to do this with some minor modifications of the AUTOSAR software architecture. The result also shows that the redundancy related parts in the system introduce an execution time overhead less than 100 percent in the effected parts. The implemented solution should be forward compatible with newer versions with minor or no modifications.

Preface

This master thesis was a proposal made by Mecel AB to investigate the possibility of integrating the FlexRay redundancy concept into the AUTOSAR software architecture as a way to increase the reliability and safety of a system.

First of all we would like to thank Mathias Fritzson, our supervisor at Mecel, for his help and support during the course of the project. We would also like to thank Roger Johansson, our examiner at Chalmers, for his valuable inputs and feedback which helped us in the work.

We would also like to thank Mecel for this opportunity, especially Karin Denti, Erik Hesslow and Peter Lööf.

Table of Contents

Abstract	iii
Preface	iv
List of abbreviations	vii
1 Introduction	1
1.1 Background	1
1.2 Purpose	1
1.3 Objective	1
1.4 Scope	2
2 Theoretical Framework	3
2.1 FlexRay	3
2.1.1 Communication	3
2.1.2 Frame format	5
2.1.3 Topologies	6
2.2 AUTOSAR	8
2.2.1 Background	8
2.2.2 Software Architecture	8
2.2.3 Communication modes	13
3 Design	15
3.1 Redundant communication	15
3.1.1 Transmission	15
3.1.2 Reception	16
3.2 Status reporting	18
3.2.1 Determine status	18
3.2.2 Report status to application	19
3.3 Final design	23
3.3.1 Gateway	24
4 Implementation	25
4.1 Prototype system description	25
4.1.1 Software	25
4.1.2 Hardware	26
4.2 Development tools	26
4.2.1 Picea	26

4.2.2	CANalyzer	26
4.2.3	Lauterbach TRACE32	27
4.2.4	Kvaser CanKing	27
4.3	Development process	27
4.4	Design implementation	28
4.4.1	Redundant communication	28
4.4.2	Status reporting	30
4.5	Testing and measuring	32
4.5.1	Redundant communication and status reporting	32
4.5.2	Performance test	32
4.5.3	Electromagnetic radiation test	35
5	Discussion and conclusion	36
5.1	Discussion	36
5.2	Conclusion	37
	References	39
	Appendix A – Performance test results	41

List of abbreviations

μC	Microcontroller
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basic Software
CAN	Controller Area Network
CC	Communication Controller
COM	Communication
CRC	Cyclic Redundancy Check
DFEA2020	Dependable Flexible Electric Architecture 2020
ECU	Electronic Control Unit
E/E	Electrics/Electronics
FR	FlexRay
FRIF	FlexRay Interface
FUSS	Functional Systems Safety
ICD	In-Circuit Debugger
I-PDU	Interaction Layer Protocol Data Unit
LIN	Local Interconnect Network
L-PDU	Link Layer Protocol Data Unit
LPT	Line Print Terminal / Local Print Terminal
PDU	Protocol Data Unit
PDUR	Protocol Data Unit Router
RTE	Run Time Environment
SW-C	Software Component
TDMA	Time Division Multiple Access
USB	Universal Serial Bus
VFB	Virtual Functional Bus

1 Introduction

In the past decades the number of electronic control units (ECUs) within vehicles has increased rapidly. With the increasing number of ECUs the demand on high speed networks to interconnect the ECUs has increased as well.

As the electronic systems within vehicles grow, more and more of the mechanical/hydraulic systems are replaced by x-by-wire systems (e.g. steer-by-wire and brake-by-wire). These x-by-wire systems often control safety critical tasks which places high demands on reliability and safety of these systems. An important part of these, often distributed, x-by-wire systems is that the communication between different nodes are reliable, i.e. the correct data is received. At the same time increasing amount of electronic systems, such as electric powertrains and drive systems, give rise to increasing electromagnetic interference on in-vehicle components and communication cables which may affect signal integrity. [1] [2]

In order to meet the demands above the automotive industry has developed a serial communication protocol, FlexRay. FlexRay has a number of features that makes it suitable for both safety related applications and high speed networking. One of those features is the support for two channels for communication, which can be used to send either redundant data for reliability/safety or for increased bandwidth. [3] [4] [5]

The industry has also created a standard automotive software platform, AUTOSAR, which aims to make the software hardware independent, more scalable, more modular and easier to maintain. AUTOSAR has incorporated FlexRay as one of the communication technologies. [6]

1.1 Background

Mecel AB is a company which specializes in developing and consulting for the automotive industry with focus on vehicle communication technologies [7]. Mecel is currently taking part in the government founded research project DFEA2020 which task is to investigate what kind of on-board electronics architecture a Volvo car, produced in the years 2017-2025, should have to fulfill the core values of “Green”, “Safe” and “Connected” [8]. Within the DFEA2020 project Mecel has turned the focus on the “Safe” part with the sub-project Functional Systems Safety (FUSS) and is conducting research in the areas of AUTOSAR, FlexRay and ISO26262. This thesis is a project within the scope of the FUSS project at Mecel.

1.2 Purpose

The purpose of this thesis work is to investigate how the FlexRay redundancy concept can be integrated into an AUTOSAR environment as a way to increase the level of reliability for safety-critical systems.

1.3 Objective

The objective of the investigation is to find where in the AUTOSAR communication stack the application signal shall be duplicated for transmission on the two FlexRay channels and where the signals shall be merged at reception. Another part is to

investigate how the status of the received signals can be reported to the receiving application to allow it to act accordingly. A proof of concept will be implemented, based on the results of the investigation, into an existing prototype system developed as a part of the FUSS project. Suitable tests and measurements shall be made to verify the function and performance of the implementation.

1.4 Scope

The thesis work does not aim to produce a general production ready solution, rather a result to aid future work and development in the area. The proof of concept is integrated into a prototype system running on AUTOSAR 3.1.4. The FlexRay CC used implements FlexRay 2.1 revision A.

The prototype system given is assumed to be correct and no further testing will be done to ensure the correctness of it. Tests performed as a part of this thesis will only ensure that the system still functions as intended with the modifications made.

2 Theoretical Framework

2.1 FlexRay

FlexRay is a communication protocol designed to provide a fast and reliable network for next-generation automotive applications. The development of FlexRay started as a response to meet the future demands on high speed networks and reliability in automotive applications. Current protocol standards like CAN and LIN will not be able to meet all these demands [4]. The main feature of FlexRay is that it offers high bandwidth (up to 10 Mbit/s per channel), determinism and fault-tolerance. The FlexRay consortium was founded in 2000 by leading companies from the automotive industry to develop the protocol. With the release of the latest version of the FlexRay specification, 3.0.1, the consortium was ended. Although the FlexRay standard is still active and maintained by the industry that currently are working on a FlexRay ISO specification (ISO 17458). [3] [9]

2.1.1 Communication

The FlexRay protocols offer nodes to communicate on two different channels; channel A and channel B. As illustrated in Figure 1 below this feature can be used to either increase the data rate by sending different data on each channel or for redundancy by sending the same data on both channels.

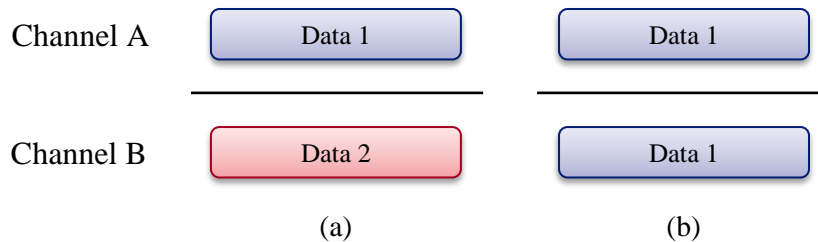


Figure 1: FlexRay channel A and B. In (a) different data are sent on the channels for increased data rate. In (b) the same data is sent on both channels for redundancy.

2.1.1.1 Media access control

The FlexRay protocol offers nodes to access the bus and send data according to a time-triggered communication schedule defined in a recurring FlexRay cycle, see Figure 2. The FlexRay cycle consist of 64 communication cycles. Each communication cycle is further divided into a static segment, a dynamic segment, a symbol window and network idle time.

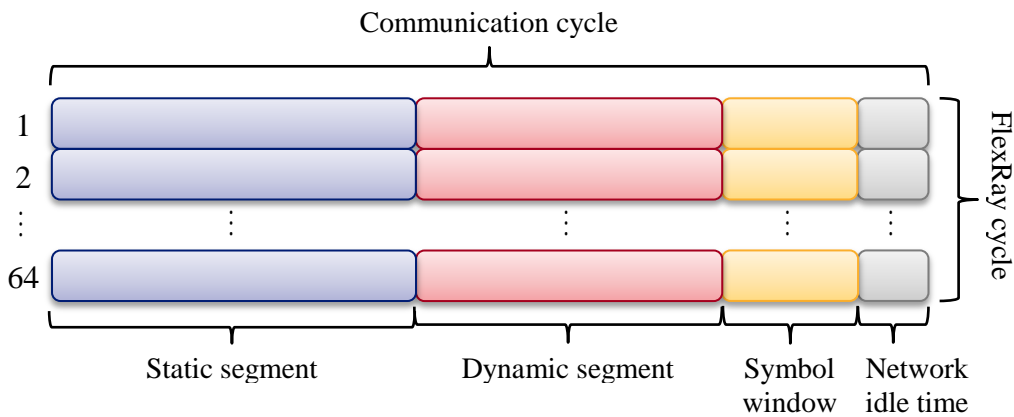


Figure 2: A FlexRay cycle and its parts.

The static segment of a communication cycle provides a static time division multiple access (TDMA) scheme to coordinate access to the bus. Each static segment consists of a pre-configured number of static communication slots, see Figure 3. Each static slot gives a node exclusive right to send data on the bus. This scheme offers a guaranteed deterministic service and is therefore suitable for safety critical data.

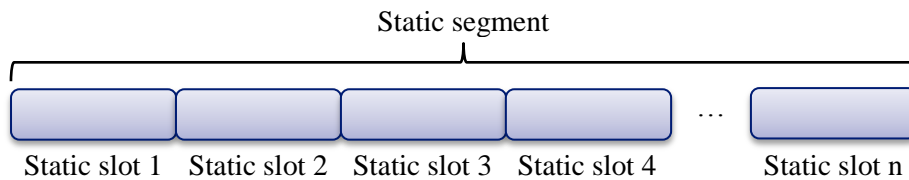


Figure 3: A static segment and its static slots.

The dynamic segment of a communication cycle provides a dynamic event-triggered scheme. Each dynamic segment consists of a pre-configured number of mini slots, see Figure 4a. A mini slot gives a node exclusive right to send data on the bus. The data is sent in dynamic slots, which size depends on whether data is being sent or not. If no data is sent the dynamic slot only consists of one mini slot, see Figure 4a, and if data is sent the dynamic slot spans multiple mini slots to fit the size of the data, see Figure 4b. This scheme offers a “best-effort” service and will not be able to give any guarantees that all nodes will be able to send their data. Although with appropriate mini slot assignments some nodes may be guaranteed to send their data, for example the node assigned to mini slot 1 will always be allowed to send if it has data to send. The dynamic segment should therefore be used with care for safety-critical data.

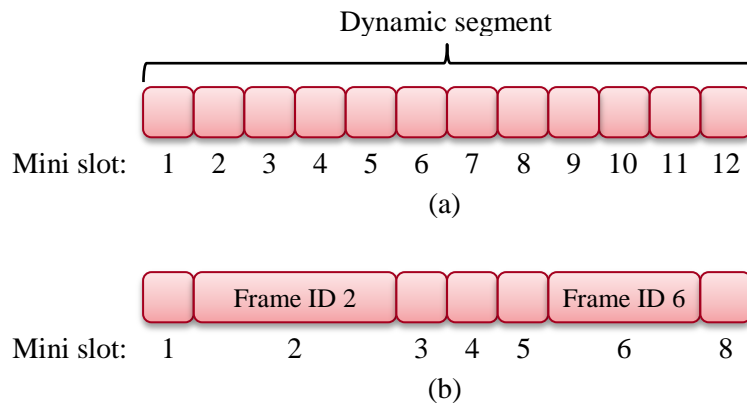


Figure 4: A dynamic segment and its mini slots. In (a) no data is sent. In (b) data is sent in mini slot 2 and 6.

The ratio between the static- and the dynamic segment can be configured to fit the needs of a specific application.

The third part of a communication cycle is the optional symbol window. The symbol window is used for network maintenance and during start-up to wake up and synchronize nodes.

The last part of the communication cycle is the network idle time. During the network idle time the network should be “quiet” and the nodes should calculate rate- and offset corrections used in the distributed clock synchronization algorithm to synchronize the nodes. The nodes are synchronized on a global time base called FlexRay global time, which is measured in so called macroticks. Each macrotick consist of a predefined number of microticks, which are derived from the oscillator clock tick. This allows nodes that use different clock frequencies to be synchronized.

2.1.2 Frame format

FlexRay transmit data over the network in frames. The frame format is shown in Figure 5 and consist of three parts; header, payload and trailer.

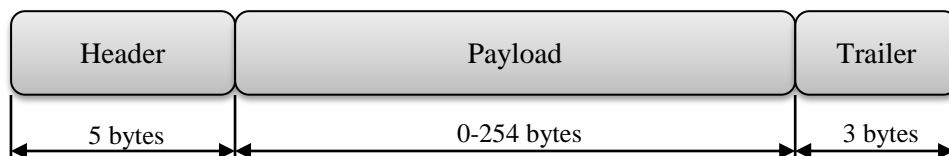


Figure 5: FlexRay frame format.

The header includes the following fields, see also Figure 6:

- **Payload preamble indication:** Indicates that network management data or a message id is available in the payload, see below for more information about this.
- **Null frame indicator:** A frame has to be sent in all static slots. If a node does not have any data to send a null frame (empty frame) is sent and this bit is used to indicate this and the frame is discarded at reception.
- **Synch frame indicator:** Indicates that this frame should be used in the distributed synchronization algorithm.

- **Startup frame indicator:** Indicates that this frame is used and serves a special role during the startup phase.
- **Frame ID:** The ID of the slot in which the frame should be transmitted in.
- **Payload length:** The length of the payload in number of words (2 bytes), which gives a maximum payload length of $2 \times 127 = 254$ bytes.
- **Header CRC:** A CRC over 2 status bits, frame ID and payload length.
- **Cycle count:** The communication cycle in which the frame should be transmitted.

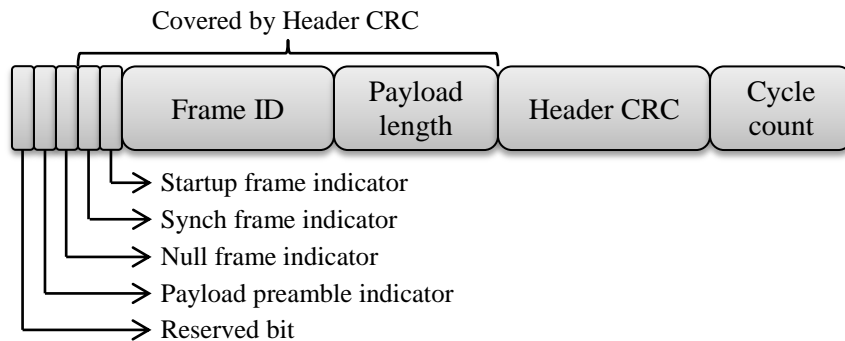


Figure 6: FlexRay header format.

The payload section of the frame contains the actual data to transmit. The length of the payload can be 0-254 bytes. The first 0-12 bytes of the payload may contain data used for network management. The presence of network management data is indicated with the 'Payload preamble indicator' status bit. Network management data can only be sent in the static segment. In the dynamic segment the 'Payload preamble indicator' status bit indicates that a message ID is present in the first two bytes of the payload. The message ID is used at the receiving side to select in which receive buffer the received data should be stored.

The trailer contains a 24-bit frame CRC value that is calculated over the header and payload for error detection.

2.1.3 Topologies

A major feature in FlexRay is that it supports a variety of different network topologies. Figure 7 shows some examples of possible topologies.

Figure 7a shows the bus topology where each node is connected to the same bus. Each channel has its own bus and a node can be connected to both or to only one of the channels. The FlexRay protocol puts a limit on the physical distance between two nodes of 24 meters [10]. This puts a limit on the diameter (the longest distance between two nodes) of a bus network.

Figure 7b shows the star topology where each node is connected in a star. The active star, as shown in the middle of the star, works as a gateway and repeats all messages. The star topology can overcome the problems with limits on the physical distance by adding these repeating stars.

Figure 7c shows a hybrid topology which is a mixture of bus and star topologies. It should be noted that channel A and B does not have to use the same network topology. It is for example possible to have a bus topology on channel A and a star topology on channel B.

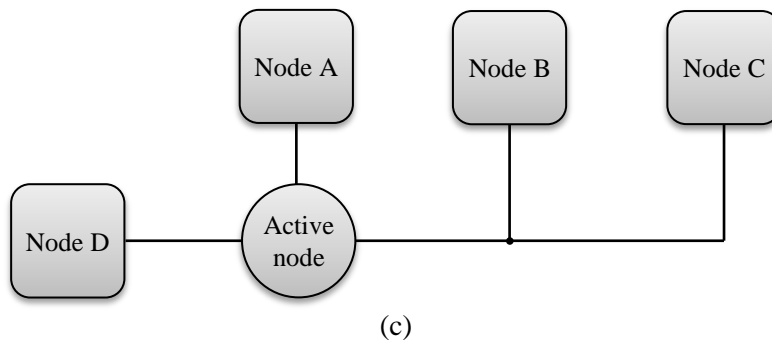
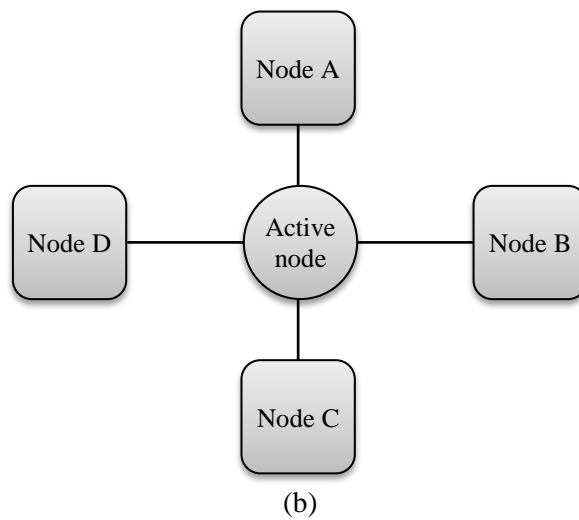
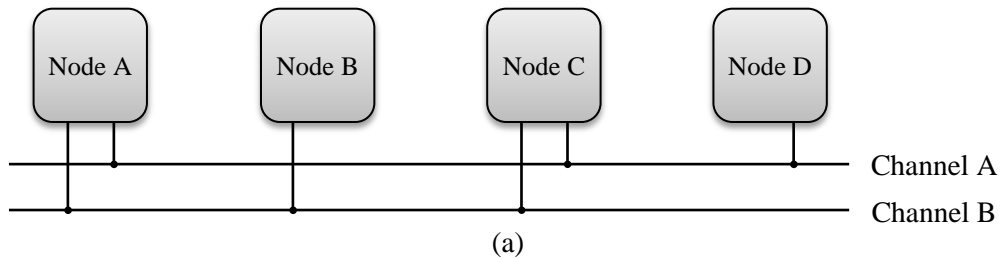


Figure 7: Examples of FlexRay topologies. (a) Bus topology. (b) Star topology. (c) Hybrid topology.

2.2 AUTOSAR

2.2.1 Background

AUTomotive Open System ARchitecture (AUTOSAR) is a standardized software architecture developed by and for the automotive industry. The standard is a result of the combined efforts of automobile manufacturers, automotive suppliers and tool developers to manage the increasing complexity of E/E-architectures in today's cars. The collaboration was started in 2002 with initial discussions between BMW, DaimlerChrysler, Bosch, Continental and Volkswagen, regarding the common problem at hand. Since then several other companies have joined the AUTOSAR project.

To achieve the AUTOSAR goals such as modularity, reusability, scalability and safety of functions in automotive systems, a layered software infrastructure has been developed, see Figure 8. The infrastructure comprises a large number of different modules performing basic system functions such as resource management, inter/intra node communication and network management, diagnostic services and scheduling to name a few. The standard does not specify how these different layers and modules should be implemented; this task is left for the different suppliers to solve. However it does provide well-defined standardized interfaces for interaction between modules, layers and for applications developed on top. "Cooperate on standards, compete on implementation" is a key idea within the AUTOSAR alliance.

The standardized interfaces are where a lot of the power in AUTOSAR lies. They make it possible to build systems with high modularity and scalability, where different modules and applications are implemented by different manufacturers and combined to work together through the AUTOSAR interfaces. The infrastructure also makes large part of the system development independent of the underlying hardware and the actual mapping of applications to the ECUs in the vehicle. It allows reusability and of the shelf applications that are well tested and widely used which makes them more reliable and safe. [6]

2.2.2 Software Architecture

The AUTOSAR architecture introduces an important separation between the application software in a system and the infrastructural functions. This is done to direct the focus and energy of developers away from the basic non-functional parts of an application.

In the top layer of the AUTOSAR architecture applications are built from atomic software components, see Figure 8. An application can comprise several software components, each implementing some specific function(s). The components are called atomic as they cannot be further divided and mapped on different ECUs in the system. However an application may be divided on multiple ECUs by the means of using several atomic software components that communicate with each other over the abstract Virtual Functional Bus (VFB).

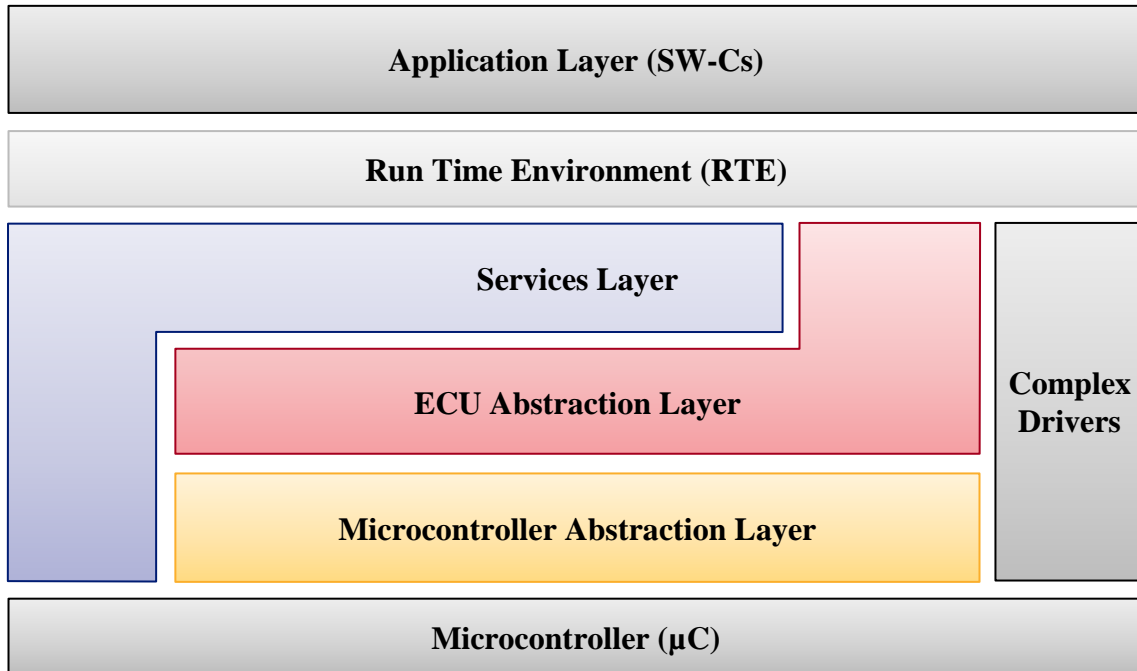


Figure 8: Shows the AUTOSAR layered software architecture [14].

2.2.2.1 VFB

To assure the location and hardware independent properties of application software development, the software components in AUTOSAR are connected to what is called the Virtual Functional Bus (VFB). The VFB is an abstract layer which embodies all interconnections and data exchanges between software components and between components and the system environment. This virtual bus allows integration of applications into the system in an early design phase before the concrete system communication infrastructure is finalized. Thus the VFB acts as a separator between the application functionality and the concrete implementation of the ECUs and system infrastructure. [11]

2.2.2.2 RTE

The Run Time Environment (RTE) together with the modules of the Basic Software, see next section, realizes the abstract services of the VFB. It is generated specifically for each ECU and is the glue between the application layer and the implementation of the VFB. The RTE is located below the application layer and above the Basic Software and provides the interfaces of the VFB to the AUTOSAR SW-Cs.

When the applications are deployed the SW-Cs are mapped to the different ECUs in the system and the virtual connections are mapped to the concrete communication facilities of the system. If two communicating components are mapped to the same ECU then their connection will be intra ECU and most likely handled solely by the RTE of that ECU. If communicating components end up on different ECUs the communication will be mapped on the network. The RTE will delegate communication from these components to the network via the communication stack Basic Software (BSW) modules. [12]

2.2.2.3 BSW

The Basic Software layer is situated below the RTE and contains a number of standardized software modules providing services to the AUTOSAR SW-Cs. The Basic Software layer can be further divided into a couple of sub layers as shown in Figure 8 and a more detailed view of each layer can be seen in Figure 9. [13] [14]

Services Layer

This layer contains modules that provide different kinds of services to the AUTOSAR SW-Cs. Services provided are memory management, network communication and operating system functionality. The Services Layer provides a μ C and ECU hardware independent interface to the AUTOSAR SW-Cs.

ECU Abstraction Layer

The ECU Abstraction Layer contains external device drivers and interfaces drivers located in the Micro Controller Abstraction Layer. The task of it is to provide a μ C and ECU hardware independent interface to upper layers, which allows use of peripherals and devices without any notion about their location.

Micro Controller Abstraction Layer

The lowest layer of the Basic Software is the Micro Controller Abstraction Layer. The task of this layer is to make higher layers μ C independent. The layer contains μ C dependent drivers for internal peripherals and memory mapped external devices.

Complex Drivers

The Complex Drivers offers direct access to the hardware or BSW modules and can be used for certain resource critical applications.

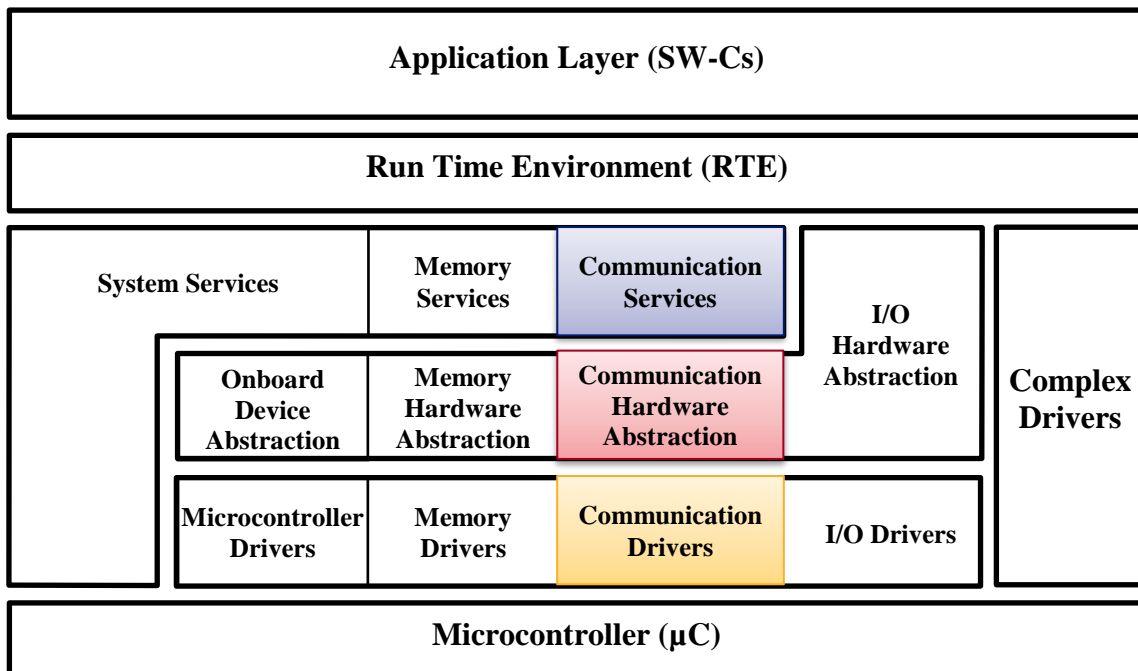


Figure 9: Shows a more detailed view of the types of BSW modules and highlights the most relevant to this thesis [14].

2.2.2.4 AUTOSAR COM STACK

The focus of this thesis lies on the BSW modules that together form the AUTOSAR communication stack. It is in these modules the FlexRay redundancy concept is implemented and integrated in AUTOSAR. The AUTOSAR COM stack provides a uniform interface for use of the different networks (FlexRay, Lin, CAN) and hides message properties and protocol specifics from the application. As mentioned above, when SW-Cs that communicate with each other are located on separate ECU the RTE of those ECUs will call the services of the communication stack to send the data over the network to the receiving ECU. The modules of the communication stack are divided into three different parts; the communication drivers which belongs to the Micro Controller Abstraction Layer, the communication hardware abstraction which belongs to the ECU Abstraction Layer and the communication services which belong to the Service Layer [13]. The most important modules to this project, and their location within the stack, can be seen in Figure 10.

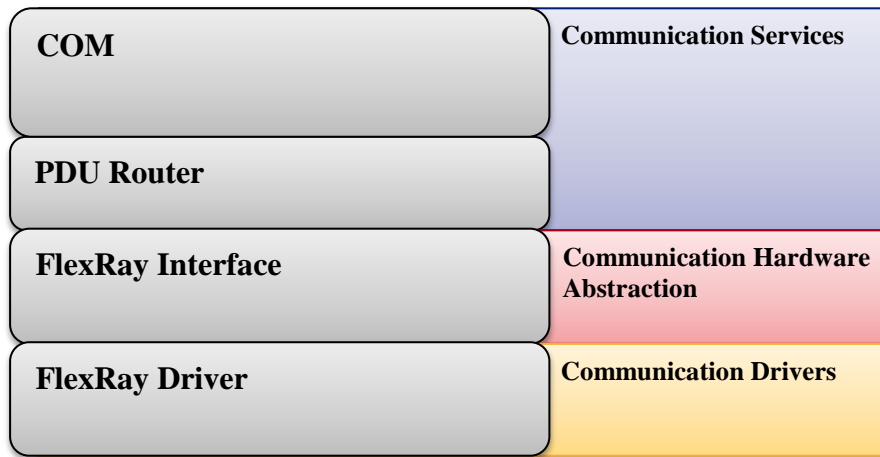


Figure 10: Relevant modules of the AUTOSAR Communication stack.

COM

On the very top of the AUTOSAR Communication Stack between the RTE and the PDUR lies the AUTOSAR COM software module. It is a part of the Communication Services and provides the RTE with a signal based API for transmission and reception of data elements. The main functions of the COM module is packing of AUTOSAR signals and signal groups into Interaction Protocol Data Units (I-PDUs) for transmission and unpacking of signal and signal groups from received I-PDUs, see Figure 11. After the signals or signal groups has been packed into an I-PDU it is delivered to the PDUR module for further processing. When I-PDUs are received from the PDUR the contained signals are unpacked and delivered to the RTE and the application.

The COM module also includes many other functions like deadline-monitoring, notifications, filtering of incoming signals, signal adaption such as sign-extensions and endianness-conversion, transmission control of I-PDU groups, PDU based callouts, and a signal invalidation mechanism to name a few [15].

AUTOSAR 4.0 introduces a feature called communication protection that sends an I-PDU in multiple L-PDUs in order to prevent loss of data or corruption. At the receiver the replicated I-PDUs are compared and a voting mechanism is used in the COM module to decide whether to accept the I-PDU or not [16].

PDUR

The PDU Router module is situated below the COM module and is also a part of the Communication Services. The main function of the PDUR is the service of routing I-PDUs between different BSW modules depending on the type of communication. Down to this point the processing of data is communication protocol independent. The task of the PDUR is to simply forward, without modification, the I-PDUs to the correct communication interface module or transport protocol module depending on the protocol used (CAN, Lin, FlexRay etc.). The routing is static and the PDUR identifies the destination module by an I-PDU ID and static routing tables. For I-PDU transmission the PDUR will transfer the received I-PDU from the upper layer module to the lower layer module according to the static routing tables and correspondingly for I-PDU reception from the lower layer to the upper layer module(s) [17].

FlexRay Interface

The FlexRay Interface module belongs to the ECU abstraction layer or more precisely the Communication Hardware Abstraction. It provides to the upper layers an abstract interface for accessing the FlexRay communication system. For AUTOSAR I-PDUs to be bus independent there is a maximum length of 8 bytes, this is the maximum length of a CAN frame, however a FlexRay frame can carry up to 254 bytes of data in its payload [3]. Therefore the FRIF provides the service of packing several I-PDUs into a Link layer PDU (L-PDU) for transmission, and unpacking the I-PDUs contained in an L-PDU upon reception, see Figure 11. The packing is done according to preconfigured frame construction plans.

To realize the time-triggered communication schedule that the FlexRay protocol uses the FRIF has what is called a Job List. Each job in the job list includes the start time of that job, which communication cycle as well as a macro tick offset within the cycle, and a set of communication operations which shall be performed within the context of that job. The communication operations perform different actions such as receiving and storing frames, transmitting frames, indicate upon reception and deliver I-PDUs to the upper layer modules. The jobs are executed by the FRIF Job List Execution Function which is called cyclically following the FlexRay Global time.

The FRIF does not access the FlexRay hardware and CC directly but do so through the FlexRay Driver module. Synchronous access to the hardware and the communication buffers of the CCs is ensured by the job list. [18]

FlexRay Driver

The FlexRay Driver module is part of the communication drivers. As such its main task is to abstract the hardware specific details (registers, message buffers etc.) of a certain type of CCs and provide an API to the upper layers. If several different types of CCs are used a specific driver is needed for each type. The FlexRay driver API is only used within the context of the FRIF and does not contain any main function which is cyclically executed. It transforms the functional requests, such as receive or transmit, from the FRIF into the correct sequence of hardware access patterns to perform the operation. The FlexRay driver also implements the different CHI commands which are used to control the Protocol Operation Control (POC) state machine of the FlexRay CC. [3] [19]

The PDU based data flow of the communication stack can be seen in Figure 11. This data flow is hidden from the application which only reads and writes data elements. These data elements can either be of primitive type, like integer, float or boolean, or of complex type which is a composition of several data elements. Each primitive type that

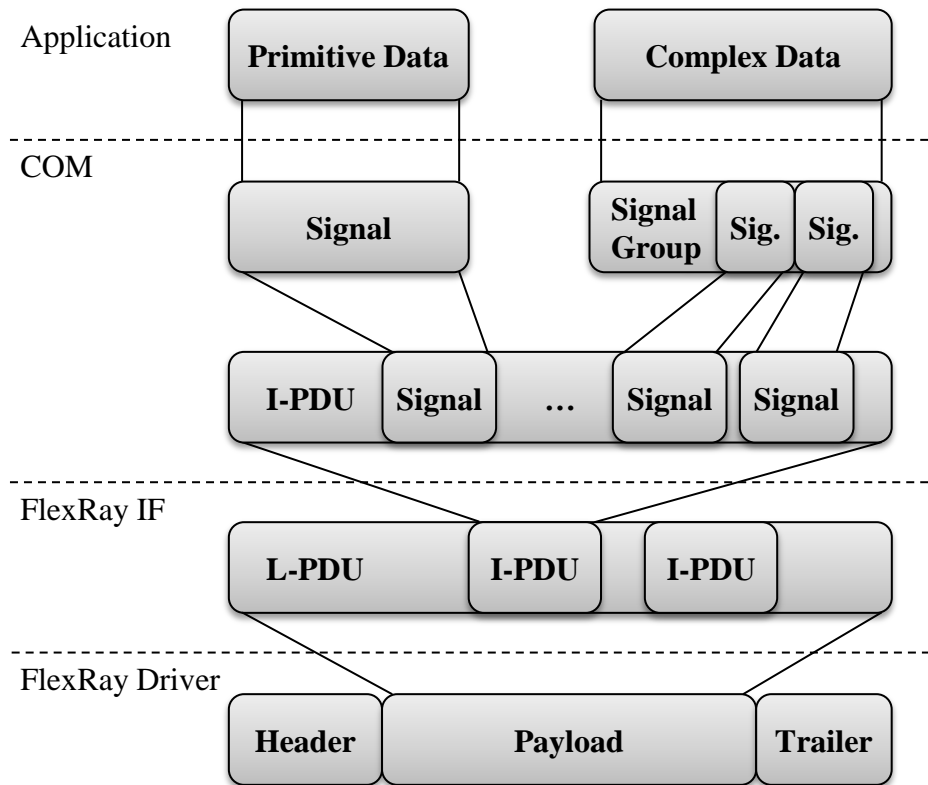


Figure 11: Illustrates the PDU based data flow of the AUTOSAR communication stack.

is supposed to be sent externally is mapped to a signal in the COM module. Each complex type is mapped to a signal group which is a group of signals mapped to each data element included in the composition.

2.2.3 Communication modes

AUTOSAR supports two different communication paradigms for distributed systems; sender-receiver communication and server-client communication. The realization of these two communication modes is located in the RTE [12].

2.2.3.1 Sender-Receiver Communication

Sender-receiver communication provides to AUTOSAR the means of an asynchronous distribution of information among software components. The sender has no knowledge of identity or location of receivers, or even how many receivers there are. Data to be transmitted is just handed to the RTE and then it is up to the communication infrastructure to distribute the data to the different receivers in the system. Each receiver then decides how and when to use the received information. The sender does not expect any response from receivers and will continue its work as soon as the communicated data has been handed to the RTE, thus there is no blocking of the sender. Any responses from receivers will be handled as separate sender-receiver communications. [13]

The sender-receiver communication comes in two different variants in AUTOSAR; data distribution and event distribution. In data distribution a “last is best” semantic is used and any old data will be overwritten by new data even if the old data has not yet been read by the receiver. For event distribution the whole sequence of event is important to the receiver. Thus this data will be queued in the receiving side RTE until it has been read by the receiving component [12].

2.2.3.2 Client-Server Communication

One of the commonly used communication paradigms in distributed systems is the client-server communication and it is the second pattern supported in AUTOSAR. The server is a component that provides a certain service and the client is a component that requires this service. An AUTOSAR SW-C is called a server or a client depending on the direction of the communication initiation. A single component may act as both a client and a server, that is provide a service for some components and require service from others. The client application makes a request, possibly providing a set of parameters, to the RTE and it is then the RTE which synchronously invokes the server. The client’s invocation may be blocking (synchronous) or non-blocking (asynchronous) depending on the nature of the communication. As a server might have several clients requiring its services the RTE must be able to handle concurrent invocations depending on whether the server can accept concurrent service provision or not [12].

3 Design

This chapter presents different design approaches on how to integrate the FlexRay redundancy concept into the AUTOSAR software architecture. The implementation design consists of two parts. The first part of the design deals with where in the AUTOSAR communication stack a signal should be split for transmission on both channel A and channel B at the sending node and where it should be merged at the receiving node. The second part deals with how to report the status of the received data to the application to allow it to act according to whether data was received or not. The goal with the design phase is to find a design that limits the negative effect on execution time and memory usage in comparison with the given prototype system where redundant communication is not used. The design should also be integrated into the AUTOSAR architecture in such a way that it limits the violations of the AUTOSAR specifications. The rest of this chapter will consider positive and negative aspects for a few different design approaches and a design chosen for implementation on the given prototype system will be presented at the end.

3.1 Redundant communication

The redundant communication part of the design consists of two parts; transmission of a frame redundantly on both channels and reception and merging of the two frames.

3.1.1 Transmission

Two different approaches are considered when it comes to where to split a signal for transmission on both channel A and channel B at the sending node. These approaches are presented in Figure 12 below.

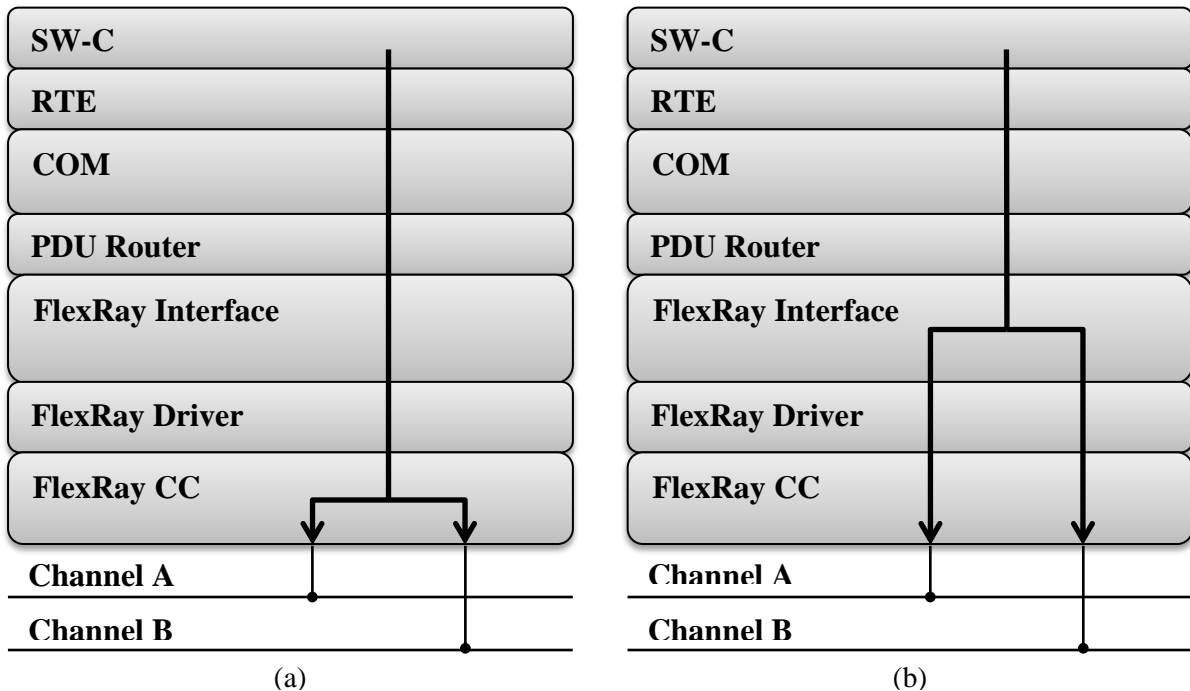


Figure 12: Shows two different approaches on where to split a signal for redundant communication in the AUTOSAR communication stack.

The first approach, Figure 12a, uses a feature available in the FlexRay CC to assign a transmit buffer to transmit a FlexRay frame on both channels in a static slot [3]. This approach works well with respect to performance since FlexRay CCs are implemented in hardware, which will limit the negative impact on the software execution time. AUTOSAR provides configuration parameters to assign a FlexRay CC transmit buffer to transmit on both channels [18]. This means that the redundant transmission becomes transparent to the AUTOSAR communication stack, which can act as if the data was sent only on one channel. This also means that no extra buffer space has to be allocated. Although a negative aspect with this approach is that the FlexRay CC only allows transmit buffers to be assigned to transmit on both channels if they are assigned to transmit in the static segment of a FlexRay communication cycle, which means that this approach does not allow for redundant transmission of frames in the dynamic segment [3].

In order to allow for redundant transmission for frames in the dynamic segment consider the second design approach in Figure 12b. In this approach the task of splitting a signal is handled by the FlexRay Interface module. For each transmission of a FlexRay frame an extra call has to be made in the FRIF to the FlexRay Driver module in order to transmit the redundant frame on the other channel. The positive aspect of this approach is that it supports for redundant transmission both in the static segment and in the dynamic segment. The negative aspects of this approach are that the task of splitting is handled by software which has a negative impact on the software execution time. It will also have a negative impact on the memory usage since extra buffers has to be allocated for the redundant frame. Violations of the AUTOSAR specifications are also needed in order or make it work correctly.

Positive and negative aspects for each of the two approaches above are summarized in Table 1 below.

Approach	Advantage	Disadvantage
(a)	Implemented in hardware Transparent to AUTOSAR No software execution time overhead No memory usage overhead	No support for redundancy in the dynamic segment
(b)	Support for redundancy in dynamic segment	Execution time overhead Memory usage overhead

Table 1: Summary of positive and negative aspects of the two design approaches regarding transmission of redundant signals.

3.1.2 Reception

This section describes two different design approaches on where to receive and merge the redundant frames in the AUTOSAR communication stack. The approaches are illustrated in Figure 13 below.

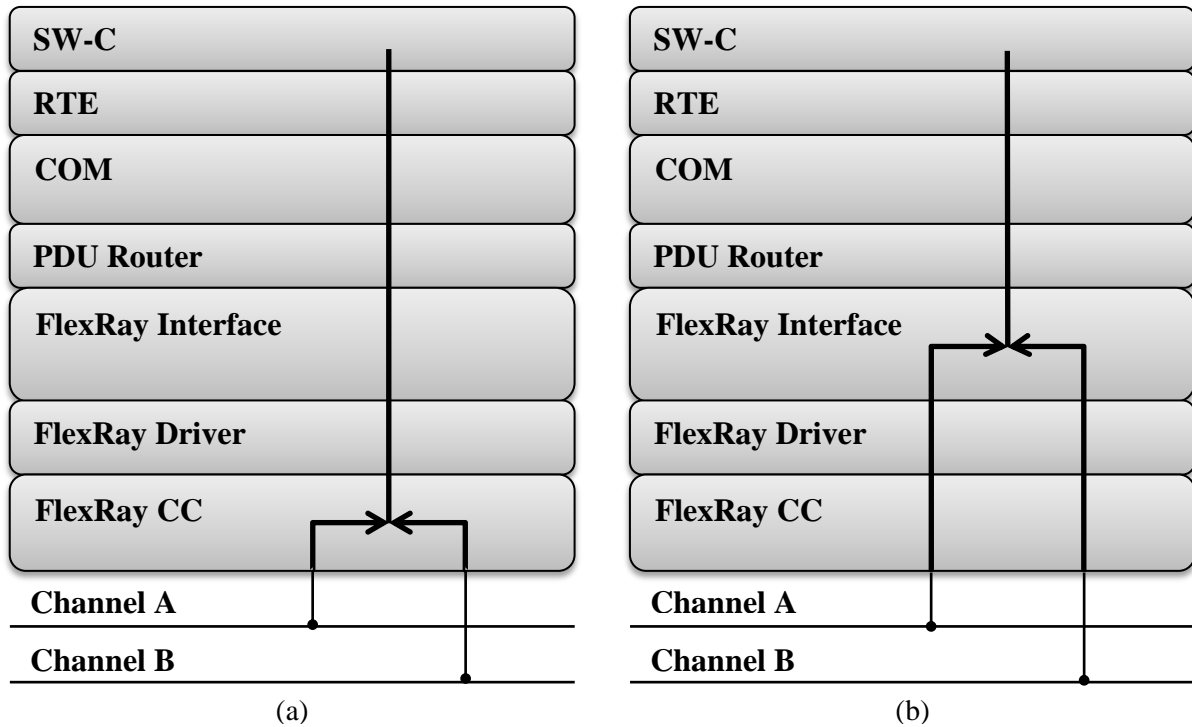


Figure 13: Shows two different approaches on where to receive and merge the two signals in the AUTOSAR communication stack.

In the first approach, as illustrated in Figure 13a, the redundant signals are received and merged in the FlexRay CC. The FlexRay CC allows a receive buffer to be assigned to both channel A and B and at reception the first valid frame received on either channel A or B are stored in the receive buffer [3]. The positive aspect of this approach is that it is totally handled in the FlexRay CC, which means it is handled in hardware. This approach hides the redundant communication from the AUTOSAR communication stack, which will only receive one frame. A negative aspect of this approach is that since only the first valid frame are forwarded to the AUTOSAR communication stack there is no way to examine and reasoning about the other frame if it was received at all. This means there is no guarantee that the same data actually arrived on both channels, which may be required by some applications for integrity reasons.

The second approach, illustrated in Figure 13b, deals with the negative aspect of the first approach. In order to be able to examine and determine the status (received/not received) for both frames they have to be merged at a higher level in the AUTOSAR communication stack. The first candidate would be the module right above the FlexRay CC the FlexRay driver. But since the FlexRay driver implementation are FlexRay CC dependent this is not a good place to merge since the solution would not be hardware independent [19]. The FRIF module is FlexRay CC independent and a better place to merge, and the one chosen for this approach [18]. This means that the two frames have to be handled separately up until the FRIF module. This adds some software execution time overhead and requires some more buffer space to be allocated to be able to handle both frames. When it comes to integrating this approach into the AUTOSAR architecture some specification violations are required as the specification does not support the reception of two frames within the same operation [18].

Positive and negative aspects for the two approaches above are summarized in Table 2 below.

Approach	Advantage	Disadvantage
(a)	Implemented in hardware Transparent to AUTOSAR No software execution time overhead No memory usage overhead	No support for redundancy in dynamic segment No possibility to ensure a higher integrity (two valid frames)
(b)	Determine status of received signals (Report status to application)	Software execution time overhead Memory usage overhead

Table 2: Summary of positive and negative aspects of the two design approaches regarding reception of redundant signals.

3.2 Status reporting

This section describes the second part of the design that deals with how to determine the status of the received data and how to report this status up through the communication stack to the application.

Normally AUTOSAR applies silent failures, which means that if something goes wrong (no data received for example) no data is sent to the application. This means that the absences of “new” data or possibly a time-out indication are the only ways an application is notified when something goes wrong, and the reason for failure is not reported. In the case of redundant communication it may be of interest for an application to know if data was received on both FlexRay channels or if data was received on only one of the channels. It might even be of interest for an application to know if no data was received at all, which could be useful in for example a distributed membership algorithm. Therefore this kind of information has to be reported to the application to allow it to act accordingly.

3.2.1 Determine status

In redundant communication with two channels there are a few possible cases that can occur in the FRIF module depending on whether frames are received or not as shown in Figure 14. The FlexRay CC only forwards frames to AUTOSAR communication stack if they are considered valid, which more or less means they pass all CRCs. The status is determined in the FRIF module after the reception of the two frames. Status simply refers to which of the four cases shown in Figure 14 and described below that is the case for the received data.

High integrity data

In the case where two valid and equal frames are received the data is considered to have high integrity.

Unreliable data

In the case where two valid and non-equal frames are received the data is considered to be unreliable. This case occurs when at least one of the FlexRay frames gets corrupted during transmission but the CRC value is still correct. When this occurs it is impossible for the FRIF module to determine which data is correct if any, therefore it is marked unreliable and any of the received data is sent to the application. The frame could be corrupted due to several reasons, such as noise on the bus causing bit-flips, a faulty transceiver sending an incorrect frame on the bus but with a valid CRC value or a faulty transceiver reading the frame from the bus incorrectly and/or stores an incorrect frame in the receive buffers. The data may also be corrupted in the communication stack between the FlexRay CC and the application SW-C due to transient faults. This fault is caught and marked unreliable if it occurs before the comparison in the FRIF module.

Low integrity data

This is the case where only one valid frame is received either from channel A or channel B. The data is considered to be of low integrity since there is no data to compare it to, to ensure a higher integrity. By allowing this case the availability of the system will increase as it allows the transmission on one of the channels to fail.

No data

This is the case where no valid frames are received.

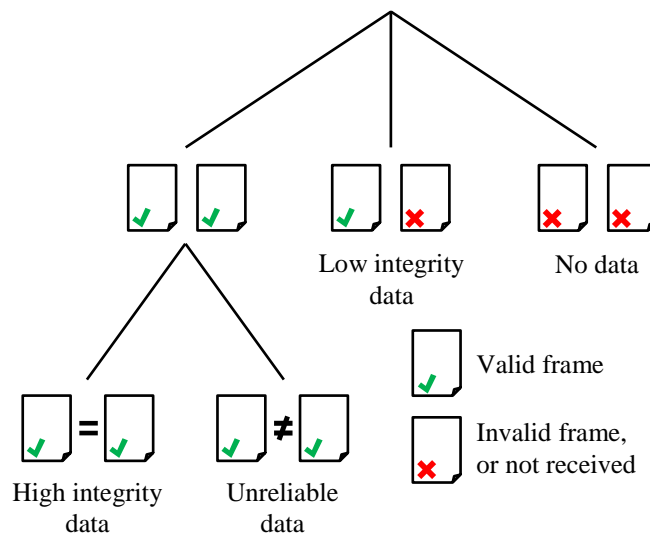


Figure 14: Shows the possible cases of at reception in the FRIF module.

3.2.2 Report status to application

When the status of the received data has been determined it has to be passed through the communication stack along with the data to reach the application. The most important thing to consider at this case is that the connection between the data and its

corresponding status is strong so that the application reads the correct status for each data.

The design can be split into two sub parts. The first sub part deals with how to pass the status from FRIF to COM module, and the second sub part deals with how to pass the status from the COM module to the application/SW-Cs.

The AUTOSAR specification offers no way to use the API calls between the modules to pass this kind of metadata up through the modules of the communication stack. In order to bypass the API the status is written to the PDU buffer and passed along with the PDU from the FRIF module via the PDUR module to the COM module. Two different approaches on where to write the status are considered. The first approach, as illustrated in Figure 15a, attaches the status to the first byte following the PDU and the second approach, as illustrated in Figure 15b, writes the status to unused bits in the PDU. The second approach is based on the existing update-bit mechanism used in AUTOSAR in which a configured bit in a PDU is used to indicate whether the data in the received PDU has been updated or not [18]. The advantages with the first approach are that the PDU can fully utilize its length, which is not the case with the approach using the update-bit mechanism. Although in the first approach extra buffer space has to be allocated for the status for each PDU, which is not the case with the second approach. In the second approach the location of the status within the PDU has to be configured for each separate PDU (if they differ), which requires some extra memory space.

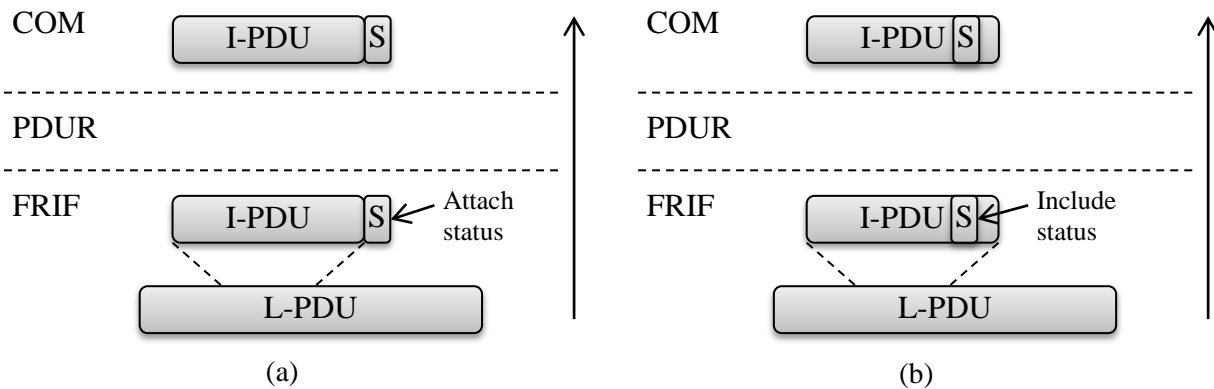


Figure 15: Shows two different approaches on how to pass the status from the FRIF module to the COM module.

Positive and negative aspects for each of the two approaches above are summarized in Table 3 below.

Approach	Advantage	Disadvantage
(a)	PDU can be fully utilized	Extra buffer space required
(b)	No extra buffer space required	PDU cannot be fully utilized Configuration parameters to specify location within PDU

Table 3: Summary of positive and negative aspects of the two design approaches regarding status reporting between FRIF and COM modules.

The second sub part deals with how to pass the status from the COM module to the application. In the COM module the status is available according to any of the two approaches illustrated in Figure 15. Three approaches are considered and illustrated in Figure 16.

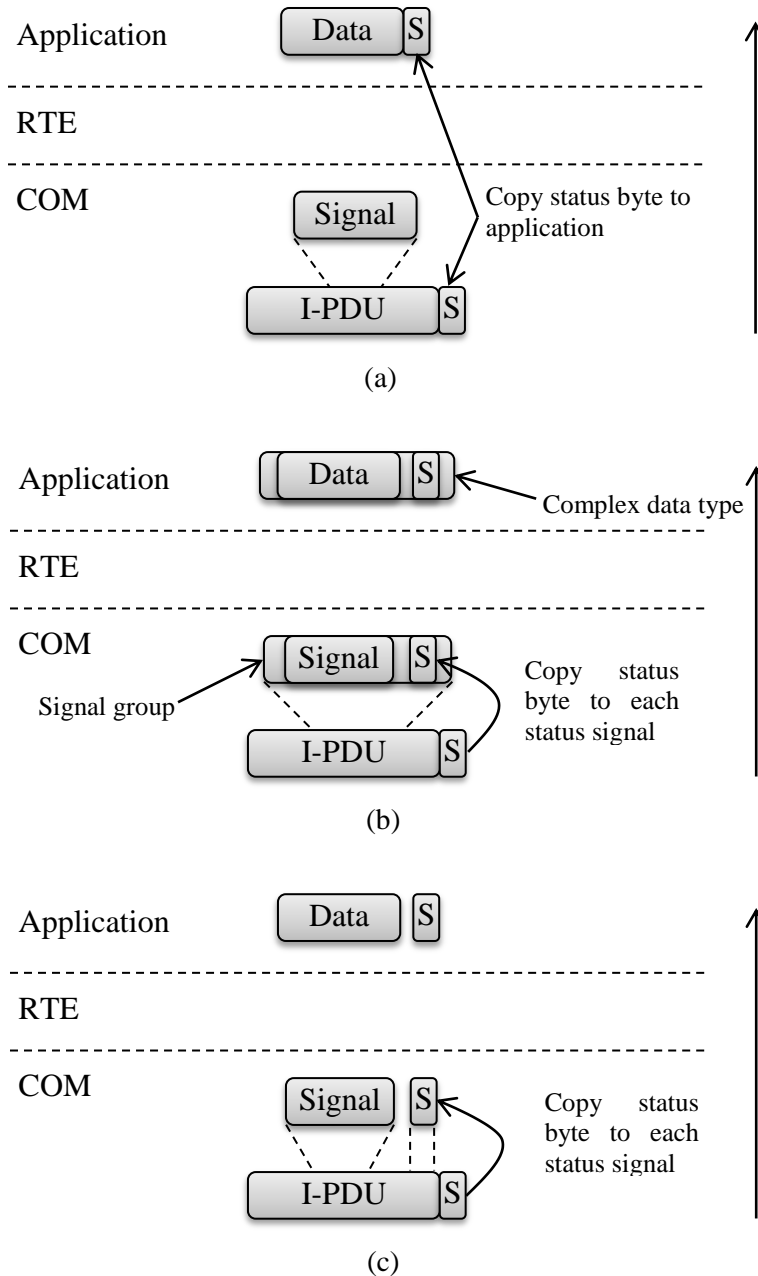


Figure 16: Shows three different approaches on how to pass the status from the COM module up to the application.

In the first approach, illustrated in Figure 16a, the status is written directly into the application when the application via the RTE calls the COM module to read the data. At read the COM module is given a pointer to a location in the application where to write the signal data. This pointer is then used to write the status to the first byte following the

data in the application. The greatest disadvantage with this approach is that the status is handled outside the scope of the AUTOSAR specification. This means that in the case where the data is not passed directly from the COM module to the application but temporarily stored in a queue in the RTE, which is the case when using client-server or event distributed sender-receiver communication, the status will be lost since the RTE is not aware of it [12]. To solve this problem the RTE has to be modified to be able to handle the status in a correct way and since the RTE is automatically generated this would require changes to be made each time the RTE is regenerated. Another disadvantage is that the application has to make sure that the first byte following the location for the received data in the application is allocated in order to be able to store the status without overwriting some other data. However an advantage with this first approach is that all status related work is handled by the receiving node which puts no burden on either the sending node or the network.

The second approach, as illustrated in Figure 16b, uses a complex data type to pass the status to the application. The sending node uses a complex data type including two primitive data types; the actual data to send and a byte to hold the status. If an already complex data type should be sent redundantly the status byte could just be include in that complex data type. The sending node writes the data field of the complex data type and leaves the status field unspecified. The status field is written by the receiving node after the status has been determined. The COM module is the first module in the communication stack, from bottom and up, that have notion about the structure of the complex data type and is therefore responsible to copy the status from the PDU buffer to the status field of the complex data type. This approach offers a tight connection between the data and the status since AUTOSAR handles complex data types as atomic units, which are sent and received atomically [15]. The case where the RTE stores the data in a queue is not a problem since it will store the whole complex data type as one atomic unit including both the data and the status. This means that no modification of the RTE is required. The status field is however only relevant to the receiving node but still handle by the sending node and sent on the bus. This adds an execution time overhead on the sending node and a communication overhead on the bus.

The third approach, as illustrated in Figure 16c, is similar to the second approach but instead of including the status in a complex data type the status is treated as a separate primitive data type and sent in a separate signal. As in the previous approach the status is left unspecified by the sender and written in the COM module of the receiving node. In comparison with the previous approach the tight connection between the data and its corresponding status is no longer guaranteed. To ensure a tight connection mechanism has to be implemented to make sure that the data and the status are synchronized at all time.

Positive and negative aspects for each of the three approaches above are summarized in Table 4 below.

Approach	Advantage	Disadvantage
(a)	Totally handled by the receiver No execution time overhead at sender No communication overhead	Problems with queuing in RTE
(b)	Tight connection between data and status Sent and received atomically	Execution time overhead at sender Communication overhead
(c)	No complex data type needed	Execution time overhead at sender Communication overhead No tight connection between data and status

Table 4: Summary of positive and negative aspects of the two design approaches regarding status reporting between the COM module and the application.

3.3 Final design

For the implementation the chosen design approaches for the different parts are; the first approach for the transmission (Figure 12a), the second for the reception (Figure 13b) and for reporting of the status the approach using the AUTOSAR complex data types (Figure 16b) is used together with first approach regarding placement in the PDU buffer (Figure 15a).

The approach for transmission is chosen because of its minimal overhead in the communication stack and its simplicity in implementation to let the separation for the different channels take place in the hardware at the very bottom of the stack. The disadvantage of not being able to send in the dynamic segment of the FlexRay cycle is a fact, though safety-critical data transmissions do not, in general, belong in a scheme which cannot guarantee send time on the bus.

For reception of data the second approach is the only real option as the other approach does not allow reasoning about the data. It is necessary to bring both received frames to a higher layer for analysis. The approach does present extra overhead in the communication system but these are necessary in order to present a status of the data to the application.

To use a complex data type and signal group to send extra control data together with the application data is a service which has become available in AUTOSAR 4.0 through the End-to-End Communication Protection Library (E2E library) [20]. This library provides an API for adding an end-to-end protection on communicated data. For safety-related sender-receiver communication the E2E can be used to wrap the send request of the SW-C to the RTE. This protection wrapper adds control fields such as counter and CRC on the sender side and on the receiving side the data is verified and faults are indicated to the SW-C. The approach for reporting the status by adding status data together with application data in a complex data type is influenced by the E2E library and integrates

well in AUTOSAR. It is also a more general approach compared to the other methods discussed above as it can handle for example queuing in the RTE for event distribution.

3.3.1 Gateway

The chosen design approach for status reporting requires an application SW-C to be modified in order to be able to use redundant communication. To avoid changing an existing application SW-C and still be able to use redundant communication a gateway SW-C will be used to relay all data to and from the application as illustrated in Figure 17. The application will send all data without any use of complex data types and status. The gateway will receive all outgoing data and pack it into a complex data type and send it over the FlexRay channels. The gateway will receive all incoming traffic over the FlexRay channels and unpack the complex data type and depending on the status send it to the application. This means that the decision whether the data should be used or not is decided by the gateway. The gateway offers to the application a transparent solution to redundant data communication. Unspecified

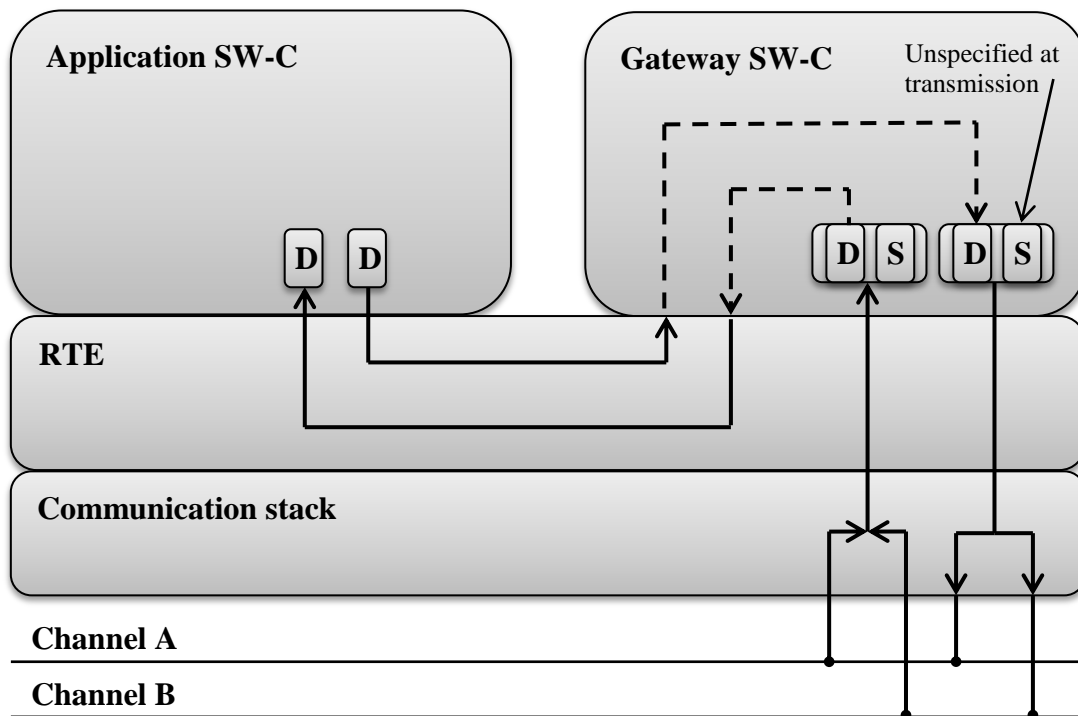


Figure 17: An illustration of how the application SW-C uses the gateway SW-C for redundant communication.

4 Implementation

This chapter explains the implementation and integration of the chosen design into an existing AUTOSAR prototype system. First the prototype system's architecture and function will be described and then the implemented solution and the integration of the same into the prototype system.

4.1 Prototype system description

The chosen design for the FlexRay redundancy has been implemented in a small prototype steer-by-wire system with force feedback. The system works by reading the driver's steering intentions and relay the information to an actuator controlling the front wheels of the vehicle.

The system architecture, which can be seen in Figure 18, contains two separate nodes which host the SW-Cs implementing the system functions. The system also has a steering wheel that is connected to a PC which communicates the angle to node 1 over a CAN bus. In the other end node 2 is connected to a small front wheel prototype which is controlled based on the angle of the steering wheel. Node 1 and node 2 are connected to each other via a FlexRay bus. The original prototype system given only used channel A for communication.

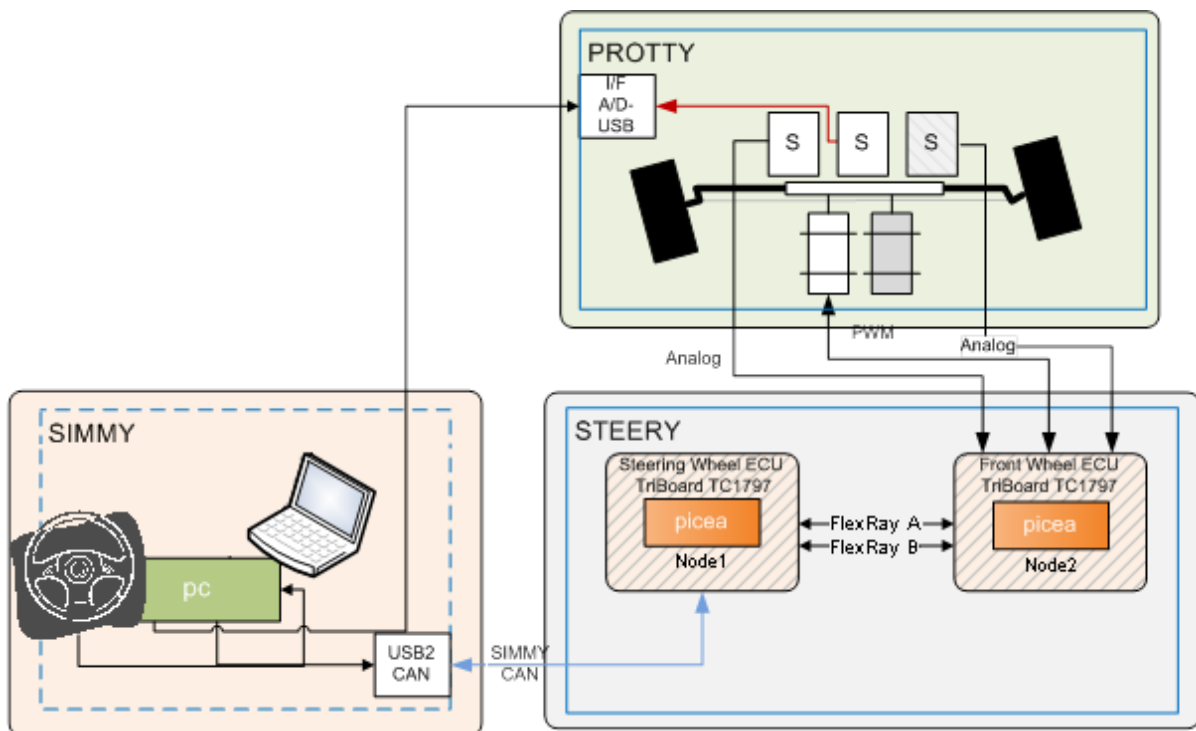


Figure 18: Prototype system architecture.

4.1.1 Software

The system functions are implemented by two different AUTOSAR SW-Cs; the Steering Wheel SW-C which resides on node 1 and the Front Wheel SW-C which resides on node 2. Each SW-C runs on top of the AUTOSAR infrastructure, RTE and

BSW, which provides all necessary services needed by the SW-C on that specific ECU to perform its function. The software architecture can be seen in Figure 19.

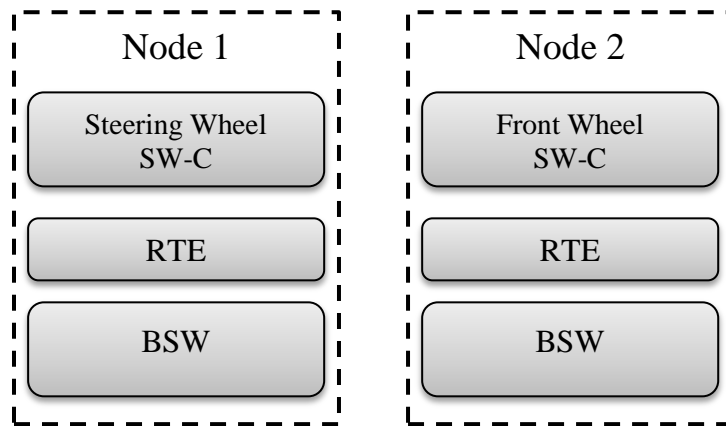


Figure 19: Software architecture of Steering Wheel ECU and Front Wheel ECU.

4.1.2 Hardware

The hardware of the ECUs consists of the Infineon TC1797 single-chip 32-bit microcontroller based on the Infineon TriCore architecture [21]. It comes with an on-chip FlexRay controller IP module called Bosh E-Ray [22], with two connected channels.

4.2 Development tools

This section describes the different development tools used in the implementation phase of the project.

4.2.1 Picea

Picea is a suite for AUTOSAR development developed by Mecel AB. The suite provides several tools further described below. [23]

Picea Workbench is an AUTOSAR ECU configuration tool used to configure the BSW modules and the RTE.

Picea Run Time Environment Generator is a tool for RTE generation. The tool read and validates AUTOSAR configuration files and generates the RTE in C code.

Picea Basic Software provides BSW modules. Each supported BSW comes with a software core and a code generator. The generator reads and validates AUTOSAR configuration files and generates configuration specific C code.

4.2.2 CANalyzer

CANalyzer is a software analysis tool for ECU networks developed by Vector. CANalyzer provides support to observe and analyze the network traffic as well as inserting traffic. CANalyzer supports many different kinds of bus systems like CAN, LIN, FlexRay and Ethernet. [24]

4.2.3 Lauterbach TRACE32

Lauterbach TRACE32 is a set of hardware and software tools used for microprocessor development. The hardware tool TRACE32-ICD, In-Circuit Debugger, provides debugging of microprocessors using an on-chip debug interface. The TRACE32-ICD is connected to a host computer using Ethernet, USB or LPT. In the host a software tool called TRACE32-PowerView is used together with TRACE32-ICD to provide a user interface for debugging, tracing and run-time analysis of microprocessors. [25]

4.2.4 Kvaser CanKing

CanKing is a software tool used to monitor CAN buses. The tool is used together with a CAN interface to connect to a CAN bus. In the software interface it is possible to monitor and log the messages sent over the bus as well as sending messages. [26]

4.3 Development process

This section presents the development process used during the implementation. The process more or less follows the AUTOSAR methodology [27]. Figure 20 shows the development process and its different phases, which are further described below.

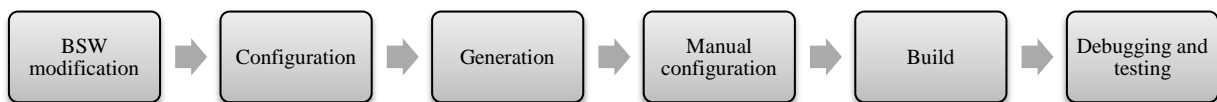


Figure 20: Shows the development process used during the implementation.

BSW modification

The AUTOSAR architecture and its BSW modules have not been designed to support redundant communication. Therefore the BSW modification phase deals with modification of the core software of the BSW modules to support redundant communication according to the design approach presented in the previous chapter.

Configuration

The configuration phase deals with AUTOSAR configuration of the different ECUs and their SW-Cs and BSW modules. The configuration deals with things like setting up interfaces for SW-Cs, adding signals and PDUs for communication and configuration of the FlexRay schedule etc. This phase is carried out using the Picea Workbench.

Generation

During the generation phase the configurations from the previous phase are used as input to generate the RTE layer and configuration files for BSW modules. This step is carried out by Picea RTE and Picea BSW.

Manual configuration

For reason that will be further discussed in the next section not all implementations are compliant with the AUTOSAR specifications. For that reason some manual configuration is needed in some of the generated configuration files from the previous phase in order for implementations not compliant with the AUTOSAR specification to work correctly.

Build

The build phase compiles all files and builds an executable that is loaded into the ECU.

Debugging and testing

In the last phase debugging and testing are carried out to make sure that the implementation works as intended. If changes or corrections are required the process is restarted from a previous phase.

4.4 Design implementation

The implementation of the FlexRay redundancy has been divided into 3 steps. Step 1 explores the method for using both FlexRay channels already available in AUTOSAR 3.1. In step 2, in order to be able to reason about the data, both frames must be received in the FlexRay Interface module. For this there is no support in AUTOSAR 3.1 so the existing module has to be extended with extra functionality to support this. Step 3 deals with the reporting of the status of the received data to the application layer. No specific way to do this is available in AUTOSAR, thus also here some special measures has to be taken. The steps can be divided in two parts; step 1-2 which deals with the spatial communication redundancy and step 3 which adds the functionality of status reporting on top. The steps are implemented and tested one at the time, first step 1 then step 2 and finally step 3 on top.

4.4.1 Redundant communication

Step 1

In implementation step 1 the existing support in AUTOSAR for sending frames on both channels simultaneously is used. This step is done entirely using the Picea workbench to configure the FlexRay Driver module and Interface module for redundant communication. On the transmitting side the transmit buffer of the FlexRay CC is assigned to both channels, thus it is first in the FlexRay CC that the data will be split and sent on separate channels. On the receiving side the receive buffer of the FlexRay CC is configured to receive on both channels. The semantics of this reception is that the first valid frame is accepted and delivered upwards and any redundant frame is simply ignored, see Figure 21. Additionally also the FlexRay cluster must be configured to use both channels and as the FlexRay specification prohibits a single node to wake up both channels [3], the nodes has to be configured to wake one channel each.

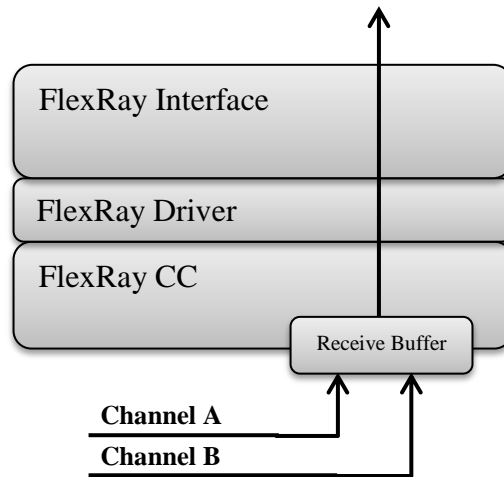


Figure 21: Shows that the FlexRay CC receive buffer is connected to both channel A and B.

To test that step 1 is working properly the bus is monitored using CANalyzer to see that the same data is indeed transmitted on both channels. Furthermore the “first valid” reception is tested by running the system and assure that it is still working as intended even though the two channels are disrupted one at the time.

Step 2

To be able to reason about the status of the communicated data, both frames need to be received and delivered up to the FlexRay Interface for comparison. This is done in implementation step 2. To be able to handle reception of two frames at the same time, allocation of extra resources is needed in the receiver side communication stack for the redundant data, up to the point of the merge. An extra receive buffer is added in the FlexRay CC so that not only the first valid frame can be received but both, see Figure 22. An extra L-PDU configuration structure is added as well as a tight connection between the two frames to be able to access them within the same context. Further, to be able to store the payload of the redundant frame an extra PDU buffer is allocated in the FlexRay Interface. For the actual receiving of the frames up to the FlexRay Interface a special communication operation is also added to the module.

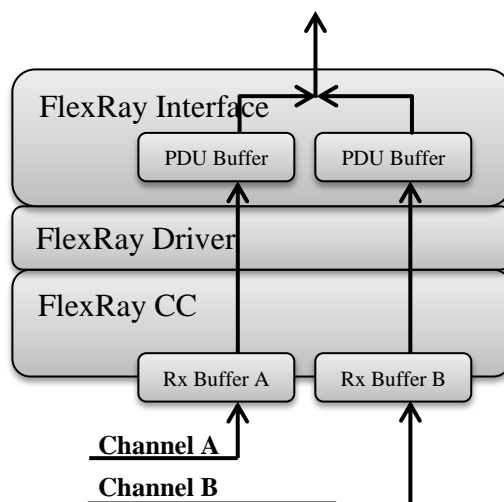


Figure 22: Shows that the received data is handled separately up until the FlexRay interface.

Most of the extra resources are added in the configuration of the ECU using the same tool as above. Though some changes has to be made afterwards in the generated configuration code, changes such as declaring the extra PDU buffer in the generated configuration code of the FlexRay Interface module as well as instructions to call the new communication operation. The actual implementation of the module itself is also extended with the new function `ReceiveRedundantAndIndicate`, a communication operation which is implemented to receive both frames within the same operation.

No comparison of the data is done in step 2. Instead much the same semantics as in the FlexRay CC in step 1 is applied, that “first valid” data is delivered up to the next layer. The receive function of the FlexRay Driver module is called for both frames but as long as there is data received on channel A this data will be forwarded to the upper layer. If no data is received on channel A but data was received from channel B this data will be sent upwards instead. Hence the end result will be the same as in step 1, though the merge now takes place in the FlexRay Interface and not in the receive buffer of the FlexRay CC. A small example of the operation can be seen in Figure 23.

```
...
status <- frDriver.ReceiveRXLpdu(CCIdx, LpduIdx, &pduBuffer);
status_red <- frDriver.ReceiveRXLpdu(CCIdx, LpduIdx_red, &pduBuffer_red);

if(RECEIVED == status)
    upperLayer_RxIndication(&pduBuffer);
else if(RECEIVED == status_red)
    upperLayer_RxIndication(&pduBuffer_red);
...

```

Figure 23: A short pseudo code example of the `ReceiveRedundantAndIndicate` communication operation.

To test the implementation of step 2 a debugger is used to see that both frames are received and their payloads copied to the memory locations of the corresponding PDU buffers. During this test CANalyzer is used to simulate the bus traffic from one of the ECUs to test the reception of the data in the other ECU. Frames with different payloads are sent to be able to tell them apart when they are stored in the PDU buffer. Thus it makes it easier to see that what is sent on one channel is also what is actually stored in the buffer corresponding to that channel in the FlexRay Interface. The system is also run to see that it still works properly even when one of the two channels is disrupted.

4.4.2 Status reporting

Step 3

The third and final step of the implementation takes care of the determining of status of the data and also the delivery of this status information to the application. As mentioned before, there are some different possible cases at the receiver; either no frame is received at all, or only one of the two frames is received, or both frames are received. In the case that both frames are received a comparison of the payloads is performed. This comparison is implemented in the FlexRay Interface and is called by the communication operation `ReceiveRedundantAndIndicate`. When the status has been

determined it is written to a special status byte which is appended to the PDU in the PDU buffer, see Figure 15a. Thus the status of the received PDU can be found by the upper layer in the first byte following the PDU data. Table 5 shows the different values of the status byte.

Value	Description
<code>RED_HIGH_INTEGRITY_DATA</code>	Both frames are received and payloads are equal.
<code>RED_LOW_INTEGRITY_DATA</code>	Only one frame was received.
<code>RED_UNRELIABLE_DATA</code>	Both frames are received but payloads are not equal.
<code>RED_NO_DATA</code>	No frame was received.

Table 5: The different status values for received data.

As the PDUs are packed together in the buffer there is no room for the status byte after each PDU because this byte would then overwrite the first data byte of the following PDU. To solve this issue the order in which the PDUs are processed at reception and copied to the upper layer is reversed. That is the last PDU in the buffer is handled first and then the second last and so on. As it is always the last PDU in the buffer that is processed the status can be written without fear of overwriting data since any PDU data stored there would already have been processed and copied to the upper layer.

In the COM module, in order to get the quality up to the application layer, the status byte is copied to the status signal of each signal group contained within the PDU. For this the ability to register a callout function with a PDU, which is supported in the AUTOSAR COM module, is used [15]. A special callout function is implemented and registered with all the PDUs that are sent redundantly. This function is then called in the COM callback function `RxIndication` each time a redundant PDU is copied from the FlexRay Interface to the COM module. The function loops through all signal groups within the PDU and writes the PDU status to the status signal in each, see Figure 16b. Afterwards the signal processing is called in normal fashion so that the signal group, now containing the status, can be delivered to the application layer in the form of a complex data type structure.

As explained in the design chapter changes to the actual application implementations is avoided by the means of a gateway which acts as a wrapper of the redundant communication towards the application. This redundancy gateway is implemented as a stand-alone SW-C which uses complex data types, in the form of C structs containing the data element(s) together with a status element, for the application data sent over the bus. One gateway is made for the Steering Wheel SW-C and one for the Front Wheel SW-C and they are added to the respective ECU-configuration. Using the ECU-configuration tool the ports of the application SW-C is connected to the gateway SW-C and the two gateways are connected to each other. Thus the two application components now communicate via the redundancy gateways. The two gateways implements the wrapping and unwrapping of the application data sent over the buses. They also analyze the status of the data received and decide whether to forward the data to its application or not. The gateway is scheduled to execute before and after its application in order to be able to provide the application with input from the buses and to send the output from the application to the buses.

4.5 Testing and measuring

This chapter describes different tests and measurements designed to test the implemented prototype system described in the previous chapter.

4.5.1 Redundant communication and status reporting

In order to test the redundant communication a simple function test is performed to see that the system works as intended. The determining of the data status, i.e. that the status byte is set correctly depending on whether data is received on one channel or both, as well as the applications ability to act depending on the status received from the communication system is also tested.

Setup

In order to verify that the redundant communication works as expected (i.e. the system should continue working even if one of the FlexRay channels fails) a construction was made to be able to “kill” one of the FlexRay buses without disrupting the system. It is implemented by a simple switch which, in “on-mode”, short-circuits the bus, thus destroying the transmission on the physical medium.

In the software some conditional code is added in the gateway on the application level which examines the data status and decides whether to forward it to the application SW-C or not. For example the steering angle data in the Front Wheel SW-C is always forwarded even if the status byte indicates `RED_LOW_INTEGRITY_DATA`, while the force feedback data is required to have `RED_HIGH_INTEGRITY_DATA` to be forwarded. This is done only for the purpose of testing the ability to act differently depending on the status of the data. A real steer-by-wire system would most likely not have this kind of functionality. Further two status LEDs are added and controlled by the gateway SW-C to indicate the status of the data received at each ECU.

Execution

The test is executed by running the system and observing the results when the buses are disrupted one at the time.

Results

When one of the channels is disrupted the force feedback is no longer delivered to the Steering Wheel SW-C and the status LED indicating degraded functionality is lit. When the disrupted channel is returned to its working condition the LED indicating full functionality is lit and the force feedback is delivered to the application again. The steering angle is delivered to the front wheel SW-C during both degraded and full functionality, thus continues to work even with one of the buses short-circuit as intended.

4.5.2 Performance test

This section describes a test that compares how the implemented system performs in comparison with the original system where redundant communication is not used. To do this interesting parts of the execution are measured. Interesting parts to measure are those that relates to the redundant communication. The parts chosen are the communication operation `ReceiveRedundantAndIndicate` in the FlexRay

interface and the reception indication function `Com_RxIndication` in the COM module [15].

Setup

In order to perform the test a simple timer is implemented. The timer uses the system clock to measure the time from one point in the code to another. The system clock runs at 64 MHz which gives a resolution of approximately 16 nanoseconds.

The timer is used to measure three different execution times in the implementation. The first time, T1 in Figure 24, is the total execution time for the `ReceiveRedundantAndIndicate` communication operation. The second timer, T2, is used to measure the part where the two frames are received and compared in the communication operation. The last timer is used to measure the time spent in `Com_RxIndication` (involves callout function to copy status byte), which is called as a part of the `ReceiveRedundantAndIndicate` operation. The same times are measured for the corresponding parts of the original implementation.

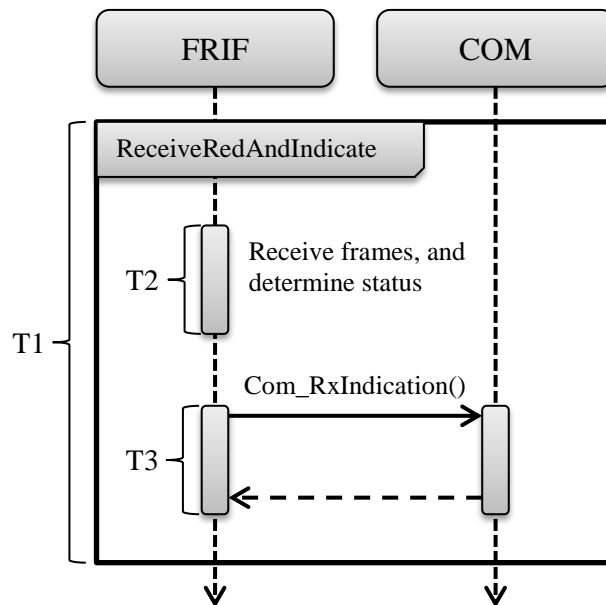


Figure 24: Illustrates where in the implementation time samples are taken.

In order to log and analyze the measured times they are sent over a CAN bus to CanKing where they are logged for further analysis.

Execution

A test is executed by running the system and log the CAN messages sent out from the system using CanKing. A test will run for approximately 25 seconds and as the two ECUs communicate with a frequency of 100 Hz this will give about 2500 samples. Four test runs will be executed; one for each ECU in both the original implementation and the redundancy implementation. These runs are performed twice and the ECUs are restarted in between.

Results

The collected samples show that 96-99% of all samples have the exact same value (the median value). The only samples that differ come in bursts with higher values periodically. These burst are most likely cases where a higher priority tasks interfere with the task executing the code being measured. From this it is assumed that the value appearing in 96-99% of the samples is the undisturbed execution time for the measured parts. It is the undisturbed execution time that is used in the results presented below.

The results from the tests are presented in the figures below and the actual measured values can be seen in appendix A. Figure 26 and Figure 26 shows the result of the different timers for the original and the redundancy implementation. The entire bar represents the time T1 and the three sub parts represents the times T2, T3 and other time not covered by neither T2 nor T3.

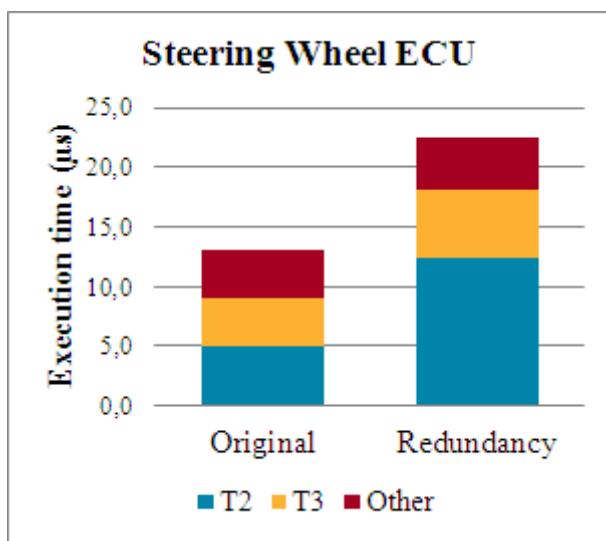


Figure 26: Illustrates the difference in execution time between the original and the redundancy implementation in the Steering Wheel ECU.

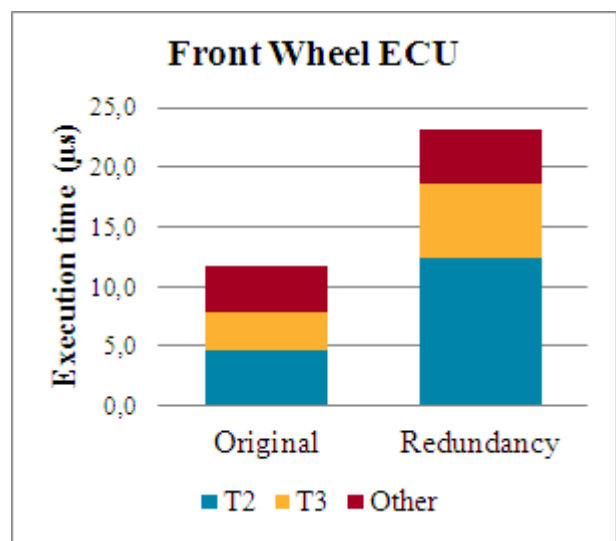


Figure 26: Illustrates the difference in execution time between the original and the redundancy implementation in the Front Wheel ECU.

Looking at the results for the second timer, T2, it adds more than a 100 percent of execution time overhead. Taken into account that the redundancy implementation receives an extra frame and compares the two frames in comparison with the original implementation this seems like a reasonable amount of overhead. The overhead added in the COM module as illustrated by timer T3 is not as high relative the one for T2. The reason for this is that after the FRIF the data flow is merged and only treated as one, which in the COM module limits the amount of overhead. Some overhead can also be seen in the other time related to neither T2 nor T3. The main reason for this overhead is some redundancy related initialization and the fact that the status byte is written to the PDU in this part.

Figure 27 shows the total overhead for both ECUs in percent and how the overhead is distributed between the different parts. As can be seen the time spent to receive and compare the frames in the FRIF module is the major contributor to the overhead.

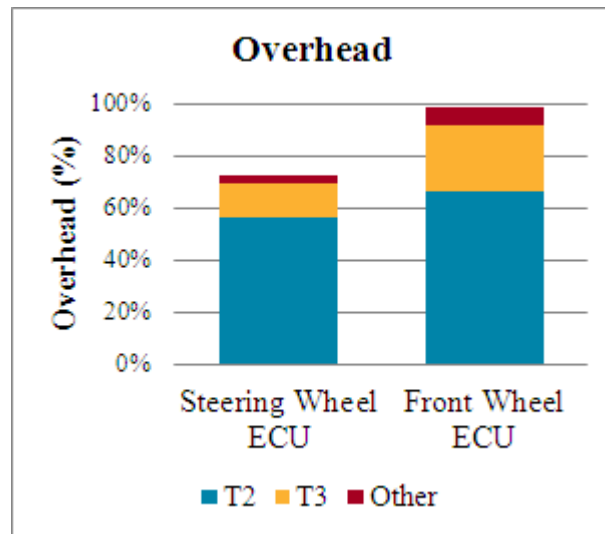


Figure 27: Illustrates where the most of the execution time overhead is found in the redundancy implementation.

4.5.3 Electromagnetic radiation test

As explained in 3.2.1 there are a few possible reasons why a FlexRay frame can be corrupted and possibly cause unreliable data to be flagged. This test aims to show that it is possible to create unreliable data by corrupting frames on the bus by disturbing one of the FlexRay buses with electromagnetic radiation.

The test was carried out at SP Technical Research Institute of Sweden. The system ran for around 200 minutes with one of the FlexRay buses disturbed with electromagnetic radiation. During this time the system sent approximately 2.4 million frames with application data and out of these 4 were logged with the status unreliable data, i.e. non-equal valid frames were received.

These preliminary results show that it is possible to inject faults corrupting the frame in such a way that it is not detected by the CRC. Further analysis of the results is outside the time frame of this thesis, but will probably lead to an additional paper published on the subject in the future in cooperation between Chalmers University of Technology and Mecel AB.

5 Discussion and conclusion

In the following chapter the results of the project are reviewed and discussed, and conclusions and further work are presented.

5.1 Discussion

The resulting implementation into the prototype steer-by-wire system shows that it is possible, with some exceptions, to integrate the concept of FlexRay redundancy into AUTOSAR 3.1.4. What it also shows are which additions that are needed in AUTOSAR in order to fully support comparison of redundant data and provision of quality information to the application.

The major issue which was faced when integrating FlexRay redundancy in AUTOSAR was the absence of the ability to receive both frames from the different channels in the FlexRay interface module and also the lack of support for passing metadata from lower layers to the applications. To cope with this a new communication operation was introduced in the FRIF module. This is a direct addition to the functionality of the FRIF and as such not a part of the AUTOSAR FRIF module specification. Though, the effects of this addition are very much confined within the FRIF module and the important well defined interfaces between the modules are not affected. When it comes to the passing of metadata up through the communication stack, the use of a complex data type allowed full integration of this function. No additional changes to modules are needed as the status is handled as any other signal processed in the stack.

The implementation should also be forward compatible with the new versions of the AUTOSAR specification and should be possible to integrate in AUTOSAR 4.0 with minor or no changes at all. The new additions in the concerned modules' specifications in AUTOSAR 4.0 do not introduce any apparent conflicts with the presented FlexRay redundancy design. In fact AUTOSAR 4.0 introduces the possibility to have user-defined communication operations in the FRIF which might make the integration of this implementation conform even better to the specification [28]. At the very least it shows that the AUTOSAR community acknowledges a need for user-defined FlexRay communication operations in future releases. As mentioned a new feature called communication protection is available in AUTOSAR 4.0 which performs redundant communication in the COM module. This approach introduces an extra overhead at the sender in comparison with the approach presented in this report since two different signal paths has to be processed in the communication stack. Also the communication protection mechanism may not be able to guarantee a tight connection between the replicated I-PDUs as the transmission of the I-PDUs is not atomic.

The prototype system is limited and only includes the sender-receiver communication mode with data distribution. However the use of complex data type to deliver the status to the application should allow for the use of sender-receiver communication with event semantics as discussed earlier in the design chapter. Neither does the system include any client-server communication. Though, the implementation of redundant communication, which is mainly located in the FRIF of the communication hardware abstraction, is communication mode independent and should therefore also work in the client-server case. Extra attention would be needed when it comes to the status reporting in the client-server case. It would need to be incorporated with the invocation, both when parameters

are passed to the server and when no parameters are sent, as well as with the response from the server to the client.

The performance test presented in the implementation chapter shows that the resulting implementation adds some execution time overhead. As shown in Figure 27 the overhead in the receive communication operation in the FRIF module is close to 100% of the measured execution time when the redundancy is used. This is reasonable and expected considering that double amount of work is needed in places where two frames are processed instead of one. Whether the amount of overhead is acceptable is another matter. It will, to a large extent, depend on the system characteristics, such as system load, communication frequency, frame size etc. In a safety-related hard real-time system it must be assured that the introduced overhead in the stack does not cause any deadlines to be missed. Though what is safe to say is that the overhead would be greater if the merge took place at a higher layer of the stack or if the redundancy was added as a protocol at application level. The merge in the FRIF means that a lot of extra signal processing, such as filtering, sign extensions, byte order conversions, for the extra frame is avoided in the COM module.

In this implementation a redundancy manager in the form of a gateway SW-C is used to handle the redundancy for the application. It should be noted that applications using a gateway must be written in such a way that it can handle the event of not receiving any new data from the gateway i.e. cases when the gateway does not forward the data due to a certain status. It should also be noted that the gateway is location dependent as the application and the gateway must reside on the same ECU. The reason for this is simply that the gateway is written to manage a specific SW-C or set of SW-Cs which are located on the same ECU. For the implementation presented in this report the sender-receiver communication mode with data distribution policy allows for the use of a redundancy manager without making changes to the application since it follows a “last is best” semantic on received data. In most cases though one cannot simply add a gateway, to manage an application which is not already adapted to benefit from it, and expect higher reliability.

5.2 Conclusion

The result shows that it is possible to integrate the FlexRay redundancy concept into the AUTOSAR architecture, with some exceptions.

The implemented status reporting feature allows an application to exploit the redundancy in several ways. The redundancy could be used to increase the availability and reliability of a system by accepting data with low integrity status, which means that the system will continue working even if one of the channels fail. It can also be used to increase the integrity level of the received data by only accepting high integrity data.

An extension to the system would be to report more status information to the application. It might be of interest to know why data was not received; was it corrupt (CRC error or syntax error) or was it not received at all. Such information could be used by an application to determine whether for example a FlexRay bus is disconnected or if there is a faulty node sending corrupted FlexRay frames.

Another further development would be to change the AUTOSAR specifications to include the redundancy related extensions discussed in the report. One approach could be to create a redundant communication library similar to the E2E protection library already present in AUTOSAR [20].

References

1. ISO 26262 Software Compliance: Achieving Functional Safety in the Automotive Industry. [Online] [Cited: May 22, 2012.] http://www.parasoft-embedded.com/printables/ISO_26262_Software_Compliance.pdf.
2. *An extended EMC study of an electrical powertrain for transportation systems.* **Chand, B, Keghie, J and Dickmann, S.** Hamburg, Germany : European Association for the Development of Renewable Energies, Environment and Power Quality (EA4EPQ), 2012.
3. *FlexRay Communications System - Protocol Specification Version 2.1 Revision A.* s.l. : FlexRay Consortium, Decemeber 2005.
4. Next Generation Car Network - FlexRay. [Online] June 2006. [Cited: May 22, 2012.] <http://www.fujitsu.com/downloads/CN/fmc/lfi/FlexRay-EN.pdf>.
5. *Research on FlexRay Communication System.* **Luo, Feng, et al.** Harbin, China : IEEE Vehicle Power and Propulsion Conference (VPPC), 2008.
6. AUTOSAR. [Online] [Cited: May 14, 2012.] <http://www.autosar.org/>.
7. Mecel. *At the forefront of automotive technology.* [Online] Mecel AB. [Cited: May 11, 2012.] <http://www.mecel.se>.
8. DFEA2020 - Dependable Flexible Electric Architecture 2020. [Online] Vinnova. [Cited: May 11, 2012.] <http://www.vinnova.se/sv/Resultat/Projekt/Effekta/DFEA2020--Dependable-Flexible-Electric-Architecture-2020/>.
9. FlexRay. *About FlexRay and this website.* [Online] [Cited: February 2, 2012.] <http://www.flexray.com>.
10. *FlexRay Communications System - Electrical Physical Layer Application Notes Version 2.1 Revision B.* November 2006.
11. *Specification of the Virtual Functional Bus, V1.0.1, R3.1, Rev 1.* [PDF] s.l. : AUTOSAR, 2008.
12. *Specification of RTE, V2.2.0, R3.1, Rev 4.* [PDF] s.l. : AUTOSAR, 2010.
13. *Technical Overview, V2.2.1, R3.1, Rev 1.* [PDF] s.l. : AUTOSAR, 2008.
14. *Layered Software Architecture, V2.2.1, R3.1, Rev 1.* [PDF] s.l. : AUTOSAR, 2008.
15. *Specification of Communication, V3.1.0, R3.1, Rev 4.* [PDF] s.l. : AUTOSAR, 2010.
16. *Specification of Communication, V4.2.0, R4.0, Rev 3.* [PDF] s.l. : AUTOSAR, 2011.
17. *Specification of PDU Router, V2.2.2, R3.1, Rev 1.* [PDF] s.l. : AUTOSAR, 2008.
18. *Specification of FlexRay Interface, V3.0.2, R3.1, Rev 1.* [PDF] s.l. : AUTOSAR, 2008.
19. *Specification of FlexRay Driver, V2.2.1, R3.1, Rev 1.* [PDF] s.l. : AUTOSAR, 2008.
20. *Specification of SW-C End-to-End Communication Protection Library, V2.0.0, R4.0, Rev 3.* [PDF] s.l. : AUTOSAR, 2011.
21. *TC1797 User's Manual, V1.1.* [PDF] s.l. : Infineon Technologies AG, 2009.

22. *E-Ray FlexRay IP-Module User's Manual, Rev 1.2.7*. [PDF] s.l. : Robert Bosch GmbH, 2009.
23. Product brief – Mecel Picea Suite. [Online] <http://www.mecel.se/products/mecel-picea/Product.Brief.Mecel.Picea.pdf>.
24. CANalyzer. [Online] Vector Informatik GmbH. [Cited: May 10, 2012.] http://www.vector.com/vi_canalyzer_en.html.
25. Lauterbach Development Tools. [Online] Lauterbach GmbH. [Cited: May 10, 2012.] <http://www.lauterbach.com/>.
26. Kvaser - Advanced CAN Solutions. [Online] Kvaser Inc. [Cited: May 15, 2012.] <http://www.kvaser.com/>.
27. *AUTOSAR Methodology, V1.2.1, R3.1, Rev 1*. [PDF] s.l. : AUTOSAR, 2008.
28. *Specification of FlexRay Interface, V3.3.0, R4.0, Rev 3*. [PDF] s.l. : AUTOSAR, 2011.

Appendix A – Performance test results

This appendix presents the test results from the performance test presented in 4.5.2. The results are presented both in system timer ticks and in microseconds.

Test round 1

Original implementation

	Steering Wheel ECU				Front Wheel ECU			
	T1	T2	T3	Other	T1	T2	T3	Other
Samples	2388	2388	2388	2388	2302	2302	2302	2302
Median (tick)	838	322	254	262	748	299	204	245
Mean (tick)	848	326	257	265	765	306	209	251
Min (tick)	821	305	254	-	243	97	204	-
Max (tick)	2129	1614	1541	-	1239	790	693	-
Median (µs)	13,1	5,0	4,0	4,1	11,7	4,7	3,2	3,8
Mean (µs)	13,3	5,1	4,0	4,1	12,0	4,8	3,3	3,9
Min (µs)	12,8	4,8	4,0	-	3,8	1,5	3,2	-
Max (µs)	33,3	25,2	24,1	-	19,4	12,3	10,8	-

Redundancy implementation

	Steering Wheel ECU				Front Wheel ECU			
	T1	T2	T3	Other	T1	T2	T3	Other
Samples	2307	2307	2307	2307	2263	2263	2263	2263
Median (tick)	1445	791	366	288	1484	793	394	297
Mean (tick)	1446	792	366	288	1499	801	398	300
Min (tick)	1445	791	365	-	1484	793	393	-
Max (tick)	1782	1125	703	-	2633	1937	1543	-
Median (µs)	22,6	12,4	5,7	4,5	23,2	12,4	6,2	4,6
Mean (µs)	22,6	12,4	5,7	4,5	23,4	12,5	6,2	4,7
Min (µs)	22,6	12,4	5,7	-	23,2	12,4	6,1	-
Max (µs)	27,8	17,6	11,0	-	41,1	30,3	24,1	-

Test round 2

Original implementation

	Steering Wheel ECU				Front Wheel ECU			
	T1	T2	T3	Other	T1	T2	T3	Other
Samples	2460	2460	2460	2460	2482	2482	2482	2482
Median (tick)	838	322	254	262	748	299	204	245
Mean (tick)	847	326	257	265	748	299	204	245
Min (tick)	837	322	254	-	243	97	204	-
Max (tick)	2128	1611	1544	-	748	299	204	-
Median (µs)	13,1	5,0	4,0	4,1	11,7	4,7	3,2	3,8
Mean (µs)	13,2	5,1	4,0	4,1	11,7	4,7	3,2	3,8
Min (µs)	13,1	5,0	4,0	-	3,8	1,5	3,2	-
Max (µs)	33,3	25,2	24,1	-	11,7	4,7	3,2	-

Redundancy implementation

	Steering Wheel ECU				Front Wheel ECU			
	T1	T2	T3	Other	T1	T2	T3	Other
Samples	2440	2440	2440	2440	2300	2300	2300	2300
Median (tick)	1445	791	366	288	1484	793	394	297
Mean (tick)	1533	837	389	307	1502	803	400	299
Min (tick)	1445	791	365	-	1472	781	393	-
Max (tick)	2435	1449	1028	-	2629	1938	1539	-
Median (µs)	22,6	12,4	5,7	4,5	23,2	12,4	6,2	4,6
Mean (µs)	24,0	13,1	6,1	4,8	23,5	12,5	6,2	4,7
Min (µs)	22,6	12,4	5,7	-	23,0	12,2	6,1	-
Max (µs)	38,0	22,6	16,1	-	41,1	30,3	24,0	-