UNIVERSITÄT SIEGEN

Naturwissenschaftlich Technische Fakultät

TIMEA: Time-Triggered Message-based Multicore Architecture for AUTOSAR

DISSERTATION
zur Erlangung des akademischen Grades
eines Doktors der Ingenieurwissenschaften (Dr.-Ing.)

vorgelegt Dissertation von:
Moisés Ignacio Urbina Fuentes

eingereicht bei der Naturwissenschaftlich-Technischen Fakultät der
Universität Siegen
Siegen– Mai 2020

Gedruckt auf alterungsbeständigem holz- und säurefreiem Papier.

Betreuer und erster Gutachter:
Prof. Dr. Roman Obermaisser, Universität Siegen

Zweiter Gutachter:
Prof. Dr. Roland Wismüller, Universität Siegen

Promotionskommission:
Prof. Dr. Roman Obermaisser
Prof. Dr. Roland Wismüller
Prof. Dr. Markus Lohrey
Prof. Dr. Madjid Fathi (Vorsitz der Prüfungskommission)

Tag der mündlichen Prüfung: 04. Dezember 2020

UNIVERSITÄT SIEGEN

Naturwissenschaftlich Technische Fakultät

TIMEA: Time-Triggered Message-based Multicore Architecture for AUTOSAR

DISSERTATION
to obtain the degree of
Doctor of Science Engineering

Submitted by
Moisés Ignacio Urbina Fuentes

Submitted to the Faculty of Science and Technology of
the University of Siegen
Siegen– May 2020

This dissertation is specially dedicated to my beloved mother and brother . . .

Acknowledgements

I would like to thank all my family, school friends, university friends and the rest of the people who have been part of my life till this really important moment. Particularly I want to express my special gratitude to two people whose participation was crucial so I could accomplish this goal: **1)** My Adviser, the Professor Dr. Roman Obermaisser who has been a tremendous mentor for me during this phase of my life. Your incredible work passion and outstanding professionalism were always a source of motivation to me to carry out this PhD work. **2)** My beloved friend Lisa Villioth who was the person that stood with me at the beginnings of this journey and pushed me to believe in myself and to take this exciting challenge in this beautiful country.

Additionally, I gratefully acknowledge the funding received towards my PhD from the German Academic Exchange Service (DAAD). Without your financial support it would have been extremely difficult. Furthermore, gratitude to the dSpace Company for their software support.

Abstract

Multi-Processors System-on-a-Chips (MPSoCs) are becoming a preferred option for the development of embedded system applications. They provide the possibility to execute different software components in parallel on different cores. In the last years several MPSoC architectures have been developed for specific application domains (e.g., by intel, powerpc, etc). However, commercial MPSoCs are the cause of major concern to certification authorities. The paradigm of message-based Networks-on-a-Chip (NoC) with support for time-triggered communication provides significant advantages with respect to temporal predictability, fault isolation and energy efficiency in comparison to the common shared memory approaches implemented for the development of multicore systems. Therefore predictable multicore platforms (e.g., COMPSOC, GENESYS MPSoC) provide message-based on-chip networks as a solution.

At present, in the automotive domain, multicore processors are deployed that use the paradigm of shared memory for the interaction between the cores. The AUTOSAR (Automotive Open System Architecture) standard introduces a multicore version of its Electronic Control Unit (ECU) software architecture since the version 4, defining a multicore operating system that controls the execution of AUTOSAR Software Components (SWCs) allocated to different cores with a shared memory for the inter-core communication. However, AUTOSAR does not provide any approach for the mapping of its ECU software architecture to a NoC-based MPSoC.

In order to combine the benefits of NoC-based MPSoCs with the AUTOSAR standard, this dissertation presents a novel system architecture which maps the AUTOSAR single-core ECU software architecture to a message-based multicore platform. The so-called TIMEA (TIme-triggered MEssage-based multicore platform for AUTOSAR) defines a message-based NoC as the only physical medium for the communication between the cores and introduces autonomous application cores which function as AUTOSAR Micro-ECUs (μECUs) on the MPSoC. Each μECU acts as a unit of abstraction where the SWCs are provided with a Run-Time Environment (RTE) and a lightweight implementation of the AUTOSAR Basic Software (BSW), exploiting the advantages of message-based NoC in contrast to a shared memory approach (e.g., fault isolation, temporal predictability).

Furthermore, computationally expensive functionality of the basic software is delegated to system cores, which serve as hardware accelerators for the application cores. TIMEA supports fault-tolerance mechanisms by the integration of new BSW modules for health monitoring services and proxy functionalities for accessing the dedicated system cores offering SWC redundancy at the core level and at the MPSoC level.

TIMEA was prototypically implemented and evaluated using a simulation framework. The simulation framework consists of an AUTOSAR simulator and on-chip simulator for the implementation of the models and algorithms. Automotive use cases based on an anti-lock braking system and a light indicator system served for the evaluation.

The obtained results demonstrate a better fault isolation for the AUTOSAR system due to the use of an on chip network for the inter-core communication. TIMEA supports stringent temporal guarantees for the SWC interaction between different cores. Moreover, the reliability of the AUTOSAR multicore system was improved considerably. Faults at the SWC level and at the core level are detected and recovery solutions based on SWC redundancy are exploited. Finally, the proposed architecture supports, for the first time, an AUTOSAR multicore platform with SWC communication through a message-based NoC.

Kurzfassung

System-on-a-Chips mit mehreren Prozessoren (MPSoC) werden zu einer bevorzugten Option für die Entwicklung eingebetteter Systemanwendungen. Sie bieten die Möglichkeit, unterschiedliche Softwarekomponenten auf unterschiedlichen Kernen parallel auszuführen. In den letzten Jahren wurden mehrere MPSoC-Architekturen für bestimmte Anwendungsbereiche (z.B., Intel, PowerPC usw.) entwickelt. Kommerzielle MPSoCs geben den Zertifizierungsstellen jedoch Anlass zu großer Sorge. Das Paradigma der nachrichtenbasierten Netzwerke auf einem Chip (NoC) bietet signifikante Vorteile hinsichtlich der zeitlichen Vorhersagbarkeit, der Fehlerisolierung und der Energieeffizienz im Vergleich zu den für die Entwicklung von Multicore-Systemen implementierten gemeinsamen Speicheransätzen. Daher stellen vorhersagbare Multi-Core-Plattformen (z.B., COMPSOC, GENESYS MPSoC) nachrichtenbasierte On-Chip-Netzwerke als Lösung bereit.

Zur Zeit werden im Automobilbereich Multicore-Prozessoren eingesetzt, die das Paradigma des gemeinsamen Speichers für die Interaktion zwischen den Kernen verwenden. Mit dem AUTOSAR-Standard (Automotive Open System Architecture) wird seit Version 4 eine Multi-Core-Version der ECU-Softwarearchitektur eingeführt, die ein Multi-Core-Betriebssystem definiert, das die Ausführung der zugewiesenen AUTOSAR-Softwarekomponenten (SWCs) steuert und verschiedene Kerne mit einem gemeinsamen Speicher unterstützt. AUTOSAR bietet jedoch keinen Ansatz für die Zuordnung seiner ECU-Softwarearchitektur zu einem NoC-basierten MPSoC.

Um die Vorteile von NoC-basierten MPSoCs mit dem AUTOSAR-Standard zu kombinieren, wird in dieser Dissertation eine neuartige Systemarchitektur vorgestellt, die die AUTOSAR-Einkern-ECU-Softwarearchitektur auf eine nachrichtenbasierte Multi-Core-Plattform abbildet. Die sogenannte TIMEA (TIme-triggered MEssage-based Multi-Core-Plattform für AUTOSAR) definiert ein nachrichtenbasiertes NoC als einziges physikalisches Medium für die Kommunikation zwischen den Kernen und führt autonome Anwendungskerne auf dem MPSoC ein, die als AUTOSAR Micro-ECUs (µECUs) funktionieren. Jede µECU fungiert als Abstraktionseinheit, bei der die SWCs mit einer Laufzeitumgebung (RTE) und einer einfachen Implementierung der AUTOSAR Basis Software (BSW) ausgestattet sind,

wobei die Vorteile von nachrichtenbasiertem NoC im Gegensatz zu einem gemeinsamen Speicheransatz genutzt werden (z.B., Fehlerisolation, zeitliche Vorhersagbarkeit).

Darüber hinaus wird die rechenintensive Funktionalität der Basis Software an Systemkerne delegiert, die als Hardwarebeschleuniger für die Anwendungskerne dienen. TIMEA unterstützt Fehlertoleranzmechanismen durch die Integration neuer BSW-Module für Health Monitoring Services und Proxy-Funktionalitäten für den Zugriff auf die dedizierten Systemkerne, die SWC-Redundanz auf Kerne-Ebene und auf MPSoC-Ebene bieten.

TIMEA wurde prototypisch implementiert und mit einem Simulationsframework evaluiert. Das Simulationsframework besteht aus einem AUTOSAR-Simulator und einem On-Chip-Simulator zur Implementierung der Modelle und Algorithmen. Zur Auswertung dienten Automotive Use Cases auf Basis eines Antiblockiersystems und eines Lichtanzeigesystems.

Die erhaltenen Ergebnisse zeigen eine bessere Fehlerisolierung für das AUTOSAR-System aufgrund der Verwendung eines On-Chip-Netzwerks für die Inter-Core-Kommunikation. TIMEA unterstützt strenge zeitliche Garantien für die SWC-Interaktion zwischen verschiedenen Kernen. Darüber hinaus wurde die Zuverlässigkeit des AUTOSAR-Multicore-Systems erheblich verbessert. Fehler auf der SWC-Ebene und auf der Kerne-Ebene werden erkannt und Wiederherstellungslösungen basierend auf der SWC-Redundanz werden ausgenutzt. Schließlich unterstützt die vorgeschlagene Architektur erstmals eine AUTOSAR-Multicore-Plattform mit SWC-Kommunikation über ein nachrichtenbasiertes NoC.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

For the development of embedded systems, such as those in the automotive industry, suitable programming models are needed that support a high degree of concurrency and sensitivity to the needs of embedded systems in terms of reliability, real-time capability, resource efficiency and support for heterogeneous computing cores. Today the paradigm of shared memory is predominant in multicore architectures. However, a shared memory typically leads to temporal unpredictability, since the access of the cores is not planned and simultaneous memory accesses are resolved dynamically. In addition, memory hierarchies and cache coherence protocols contribute significantly to the temporal unpredictability [LSL+09]. In contrast, a time-triggered message-based Network-on-a-Chip (NoC) has significant advantages [OKP10] [PPB+07] for embedded real-time systems such as improved fault isolation, real-time support and energy efficiency. As explained in [HO09] a message-based NoC is superior to a shared memory in case of a high computation/communication ratio, which is typical of automotive electronics. Compared with shared-memory architectures, message-based NoCs eliminate the overhead and the hardware complexity of a protocol for cache coherency [LAS+07], offer a better temporal predictability, support the seamless integration of autonomous cores and exhibit higher reliability and energy efficiency. These advantages are also achieved by time-triggered off-chip communication systems in the automotive domain such as FlexRay [Fle05].

Due to the increasing importance of Multi-Processor System-on-a-Chips (MPSoCs), in 2012 the automotive industry introduced support for MPSoCs in the development of automotive embedded systems with the publication of the version 4 of the Automotive Open System Architecture (AUTOSAR) standard. The last release of the AUTOSAR platform (4.3) defines a multicore operating system [AUT16c] for managing an MPSoC with multiple cores, which are used for parallel execution of AUTOSAR Software Components (SWCs). Furthermore, the functionalities provided by the AUTOSAR Basic Software (BSW) are

separated in multiple master and slave modules to support the AUTOSAR SWCs in the application layer on different cores. Thus, in order to allow the communication between the master and the slave BSW modules, and between SWCs located in different cores, an Inter-OS-Application Communicator (IOC) was introduced as a new service in the BSW of each core using a shared memory as the medium for the inter-core communication.

Nevertheless, the multicore version of AUTOSAR lacks support for message-based MPSoC architectures, therefore the mentioned advantages of message-based NoCs over a shared memory cannot be exploited. In particular, temporal predictability has been identified as a weak point in AUTOSAR [ROH+09]. In previous works MPSoC platforms based on message-based NoCs [OK09] [HGBH09] were introduced that support stringent fault isolation and temporal predictability. A major scientific challenge is the extension of the AUTOSAR architecture for such an MPSoC platform.

1.1 Contributions

The thesis presents the TIme-triggered MEssage-based multicore platform for AUTOSAR (TIMEA) with support for reliability and real-time requirements. Autonomous application cores serve as AUTOSAR Micro-Electronic Control Units (µECUs). The interaction occurs only using messages on the Virtual Functional Bus (VFB), which are mapped to the on-chip network. Each µECU is a unit of abstraction, where the timely provision of message-based services can be analyzed and understood independently from the other µECUs. Each µECU has its own BSW and there are no hidden interactions between the BSW of different µECUs. Additionally, specific system cores (e.g., off-chip gateway, I/O gateway, etc) are defined which serve as hardware accelerators for the AUTOSAR software running on the µECUs. Appropriate system models and algorithms are defined for extending the AUTOSAR BSW towards a hierarchical platform comprising:

- *Communication between SWCs on the same application core.* Implementing the RTE as defined by the AUTOSAR single-core architecture.

- *Message-based on-chip communication.* Enabling communication between application cores on the same MPSoC.

- *Message-based off-chip communication.* Enabling inter-communication between different MPSoCs.

The contributions of the dissertation are summarized as follows.

- ***Autonomous application cores in the role of µECUs.*** Each application core uses a local AUTOSAR operating system, which is solely responsible for the management of the resources of the application core. The elimination of hidden dependencies between cores avoids error propagation and facilitates the integration of independently developed and validated cores. Therefore, the AUTOSAR BSW on the application cores is adapted to support the communication through the message-based NoC. Additionally, the BSW on the µECUs is extended with new health monitoring service modules to provide failure recognition and recovery actions to the AUTOSAR software.

- ***Interaction between cores using a message-based NoC.*** The paradigm of time-triggered control leads to significant advantages [OKP10, PPB+07]. The NoC of the proposed AUTOSAR MPSoC platform realizes these benefits at the chip level and provides high temporal predictability, inherent fault isolation and a global time base. Thus, an interface module for the NoC is defined for connecting higher layers of the AUTOSAR BSW (e.g., PDU router) with the NoC.

- ***AUTOSAR-specific system cores.*** The AUTOSAR MPSoC platform delegates costly functions of the AUTOSAR BSW to system cores to improve the performance of the execution environment by acceleration through dedicated hardware and reduction of the operating system overhead on the application cores. Specific specialized system cores such a off-chip network gateway cores, input/output cores and memory cores are defined.

- ***Simplified AUTOSAR BSW.*** A reduced AUTOSAR BSW is used on the application cores that uses the functions of the system cores. In contrast, the traditional AUTOSAR multicore operating system leads to higher complexity of the AUTOSAR BSW, since more cores are managed and additional synchronization and communication mechanisms are implemented via shared memory.

- ***Efficient implementation of drivers.*** For I/O hardware, sensors, actuators and in particular for realizing complex drivers an efficient implementation with system cores is supported. For instance, for dedicated peripherals that have stringent time constraints a hardware acceleration of a device driver is possible. The choice whether a driver would be accelerated in hardware is transparent to the RTE and the AUTOSAR applications.

- ***Simulation Framework.*** A novel co-simulation framework supporting the integration of the AUTOSAR architecture with NoC-based platforms is presented. We describe a simulation model for application cores playing the role of AUTOSAR µECUs on

the MPSoC platform. The framework introduces an interface for the co-simulation of simulation models for the AUTOSAR-based software (μECUs), the natural environment and the NoC behavior. This co-simulation interface describes multiple simulation building blocks and local simulation coordinators for the synchronization and the data exchange between the simulators hosting the simulation models.

- ***Evaluation and Experiments.*** A set of experiments were carried out to evaluate the performance of the system using several automotive use cases in a simulation scenario under failure occurrences. The obtained results demonstrate how the TIMEA platform remains operational in the presence of failures.

1.2 Thesis Organization

The remainder of the thesis is structured as follows.

- *Chapter 2* introduces the basic concepts and provides the background knowledge used in this work.

- *Chapter 3* discusses the existing state-of-the-art for domain-specific embedded system architectures followed by the existing multicore platforms that use message-based NoCs for the communication between the cores. Additionally, several I/O core implementations are presented and the chapter compares different approaches for the acceleration and the employment of dedicated-core solutions for multicore applications. Moreover, the Health monitoring service of the avionic domain is presented. Finally, the original AUTOSAR multicore operating system with a shared memory approach is studied and compared to the proposed message-based NoC architecture.

- *Chapter 4* maps an AUTOSAR system to a multicore platform with a message-based NoC. Application cores are described as autonomous μECUs, each containing a lightweight AUTOSAR operating system and a RTE. μECUs provide meaningful units of abstraction and ensure freedom of inference from other cores. Computationally expensive functionality of the BSW is delegated to system cores, which serve as hardware accelerators for the application cores.

- *Chapter 5* presents a framework for the simulation of multicore processors hosting AUTOSAR software, which employs message-based NoCs for the inter-core communication. The simulation framework introduces models for the simulation of the AUTOSAR μECUs, the physical environment and the on-chip communication network.

Additionally, local coordinators are described for the communication between the simulations tools, which implement the Functional Mock-Up Interface (FMI) standard as a key part of the synchronization of the simulation systems.

- *Chapter 6* provides a detailed process for the development of the AUTOSAR message-based multicore system. The dSpace AUTOSAR development tools are used for the configuration and the programming of the AUTOSAR application cores, using the standard AUTOSAR development process of AUTOSAR single-core ECUs. Furthermore, the implementation of the previously defined system cores is carried out employing different tools for the simulation of message-based multicore systems.

- *Chapter 7* describes a set of automotive use cases that are implemented for the evaluation of the performance and fault isolation of the AUTOSAR message-based multicore platform. Several experiments are carried out which reveal the ability of the presented multicore architecture for the recognition and recovery of software and hardware failures in the AUTOSAR system.

- *Chapter 8* concludes the thesis through a discussion of the overall results of the presented solutions.

Chapter 2

Background and Basic Concepts

In this chapter the background and main concepts required for the full understanding of the work presented in the dissertation are explained.

It starts with the definition of real time embedded systems, their different classifications and the concept of timing. Thereafter, the definition of dependability, its attributes and means are established which are needed to describe a dependable system. At third, the architecture paradigms for the development of real time embedded systems are presented.

Finally, the AUTOSAR software architecture is presented together with the description of its hierarchical abstraction layers and the definitions of the AUTOSAR software components.

2.1 Real-time Embedded Systems

An entity formed by the interaction of multiple dependent components whose spatial and timing constrains can be different is considered a system. These interactions and relationships between the components together with the overall behavior of the system interacting with the environment define the functionalities of the system. The particular service delivered by the system is seen as the behavior of the system perceived by the system user. The user of the system can be a human operator as well as a second system.

Furthermore, an embedded system is a system resulting from the combination of hardware components and software components in order to perform a particular task or several tasks. An embedded system is defined by [Kam11] and [Shi09] as a system composed by two main components: a particular function-purpose hardware and the embedded software. These two components are combined to accomplish a specific application or as part of a larger system to provide a specific function. Additionally, [Hea02] defines an embedded system as a microprocessor-based system consisting of an instruction control unit and an arithmetic

control [Sta96], which is built to perform a function or several functions and differently to a personal computer is not designed be programmed by the end-user.

Embedded systems are provided with memories and input/output ports for the storage of the program code and the data, and the interaction with the environment and the system user respectively. Nowadays different types of memories are used which involve different storage purposes. A non-volatile memory (i.e., ROM) retains the stored information even after the system is powered off, so this memory is used for the storage of the program code and the configuration information. A volatile memory (i.e., RAM) serves for the storage of the application data which is used by the system during the execution. Another kind of memory is the cache which is located closer to the microprocessor unit so it increases the performance and reduces the memory access delay providing a direct and faster access to the currently used application data [LS11].

Peripherals are required by the system for the exchange of information with the external environment. Input and output ports are used for collecting significant data used by the system for the correct execution of the instructions and for the transmission of the resulting actions through system actuators or other output devices.

An embedded system is seen as a real time system when the correctness of its output behavior depends not only on the logical result of the computation but also on the physical time when the output behavior was produced by the system with respect to a global time base [Kop13]. For this, the system must maintain a continuous and timely interaction with its environment.

2.1.1 Classification of Real-time Systems

As explained in [Kop13] real-time embedded systems can be classified from different perspectives. In this section we introduce the relevant classifications used within the scope of the dissertation. If a system is evaluated based on its temporal behavior with respect to a deadline, it is classified as a soft or hard real-time system. In case the evaluation is performed based on its reaction after a fault occurrence the system is classified as a fail-safe or fail-operational system. When the system is observed in relation with the control of its actions and the transmission of its signals it is classified as an event-triggered or a time-triggered system.

2.1.1.1 Based on its Temporal Constrains

A real-time system has to calculate its outputs within a bounded interval of time after the collection of its inputs so the time instants when its outputs must be delivered can be reached. The time instant for the delivery is known as the deadline of the output. In case of stringent

timing constrains an output deadline is defined as a hard deadline. This means, in case the deadline is not met, this would lead to severe consequences as costly environmental damages or human life loss.

A system is considered a hard real-time system in case at least one of its deadline outputs is classified as a hard deadline. The design of this kind of systems requires guaranteed temporal behavior and temporal predictability to provide full deadline compliances, so the safety and reliability of the system can be ensured. In the automotive domain, an anti-lock braking system is an example of a hard real-time system. A hard real-time system represents a safety critical system.

On the other hand, systems which do not declare any hard deadline to be met are considered as soft real-time systems. This kind of systems has less stringent timing constraint so they are able to tolerate a certain amount of deadline misses without catastrophic consequences rather than the degradation of their output results. A navigation system is an example of a soft real-time system within the automotive domain.

2.1.1.2 Based on its Behavior after Failure Occurrences

A safety critical system can also be classified depending of its reaction under failure occurrences. For this, two types of systems are distinguished: fail-safe systems and fail-operational systems.

In case of the appearance of a failure, a fail-safe system provides one or more safe states that can lead to stop delivering its original functionality but serve to avoid major consequences. These systems are designed to have high error detection coverage. A traffic light which switches always to red in case of a component failure is an example of a fail-safe system.

A fail-operational real time system is a system that must stay operational providing an acceptable level of the services regardless the occurrence of a failure. In order to achieve this, a fail-operational system is designed to support fault tolerance mechanisms and component failure masking so recovery actions can be triggered in case of failure occurrences. For instance, an airplane computer control system is an example of a fail-operational system since this is supported with component redundancy [OKS08] in order to provide full operation under a failure appearance.

2.1.1.3 Based on its Action Trigger Approach

In [Kop13], an event that causes the starting of one or more actions is declared as a trigger. In this context, the nature of the trigger serves to classify the systems according to two different triggering mechanisms: event-triggered systems and time-triggered systems.

In an even-triggered system processing activities and communication signals are triggered by events that can be originated within the system itself or from changes of states in the environment which are perceived through a specific message (e.g., an alarm condition indicated by a sensor). An event-triggered system implements interruptions in order to serve the events accordingly.

On the other hand, in a time-triggered system the execution of the communication and processing activities is bound to particular points in time which are synchronized based on a global time base. In [Kop92] this global time base is defined as a sparse time that enables the temporal coordination of the actions that are initiated periodically based on prior known clock ticks according to a synchronized clock reference. Real-time information about the job executions and message transmissions is contained in a pre-defined schedule table.

2.1.2 Timing Concepts

The duration between two consecutive micro-ticks of a digital physical clock is defined as the granularity of the clock. In a real-time system the synchronization of the system actions is possible due to the mapping of these to specific time stamps which preserve the time information of each activity. An absolute synchronization is obtained in case all the time stamps of a system are synchronized with an external reference clock that serves as an observing timekeeper.

In a distributed real time system with multiple nodes, every node is provided with its own local physical clock. The maximum offset between the micro ticks of two different clocks is known as the time precision between the clocks. The system is synchronized when all local clocks have the same granularity and are synchronized with the same precision. Thus, a global tick of the system global time is described as a set of micro ticks of each local clock.

The time instant when a task or action within a component becomes ready to be executed is called the release time. The computation time or execution time is the time required by a task for its execution completion after the task was started and without interruptions [Zur09]. The estimation of the execution time depends on the available system resources and possible inter-task dependencies. The response time is calculated from the difference between the release time of the task and its time completion. For hard real-time tasks the bounding of the response time is mandatory, so the analysis of the Worst Case Execution Time (WCET) must be performed assuming all possible inter-task dependencies and interruptions.

The end-to-end delay in a real-time system consists of the related time between the occurrence of an event and the occurrence of a second event. In a message system the end-to-end delay is the time required by the transmission of a message from its sender component

till its receiving destination. The message transmission may involve multiple communication networks such as on-chip and off-chip networks.

2.2 Dependability

The property of computer systems to measure their ability to provide justifiably trusted services to the final users is known as dependability [Dub13]. This is a very important non-functional term for the description of real-time systems which consists of the following three classifications [UAcLR01]: attributes, means and threats.

2.2.1 Attributes of Dependability

In [LAK92] the following dependability attributes must be taken into consideration for the delivery of a dependable system:

Safety

This dependability attribute refers to the ability of the system to avoid catastrophic consequences in case of the failure of a component. This property involves the behavior of the whole system rather than the behavior of a single component [Sto96].

Reliability

Reliability is defined as the ability of the system to maintain the correctness of the service delivery in a specific environment and period of time. This property denotes the probability of the system to stay failure-free operational with respect to a specific time unit. In safety-critical systems embedded system services are required to have a reliability in the order of 10^{-9} failures/hour [LHSC10]. In the example described in 2.1.1.2, the traffic light system exposes a poor reliability since the original service function is not delivered in case of a component failure.

Maintainability

The ability of the system to repair itself after the occurrence of a system failure is known as maintainability. This property measures the time required by the system to become operational after stopping the delivery of the services due to a failure.

Availability

Availability refers to the property of the system to stay ready for delivering its functionality. This property is directly related to the reliability and maintainability of the system. Thus, having high reliability and high maintainability increases the availability of the system.

Security

This attribute is concerned with the ability of the system to avoid unapproved access to the information and services handled by the system. It aims to reduce espionage vulnerabilities and to increase the protection against threats and cyber attacks.

2.2.2 Means of Dependability

The means to attain dependability are grouped into the four following classes according to [ALRL04]:

Fault Prevention

This term refers to the set of quality control methods and techniques which are employed during the design phase of the hardware and software. These methods and techniques are applied to reduce the introduction of faults to the system during the design phase.

Fault Tolerance

This mean refers to the ability of the system to stay operational and continuing to provide correct services in the presence of faults. This property includes error detection mechanisms and recovery solutions to move the system from a faulty state to an operational state without errors. To achieve this, a solid fault hypothesis of the system is required which provides a clear understanding of the fault assumptions.

Fault Removal

This term points to the set of techniques employed during the design phase for the verification of the system in order to detect any system behavior which does not comply with the specified functional conditions. Additionally, it also includes diagnostic and correction mechanisms to eliminate these faults.

Fault Forecasting

The estimation and evaluation of the possible future fault occurrences and the analysis of their consequences is known as fault forecasting. This evaluation can be qualitative, by identifying, classifying and ranking the faults, as well as quantitative providing a probabilistic evaluation with respect to the dependability attributes.

2.2.3 Threats of Dependability

Threats represent any failure, error or fault that can occur during the execution of the system functionalities.

Failure

A failure occurs when the system behavior is not consistent with its specification. This can be due to a system output which does not comply with the expected behavior of the system or because the system specification does not describe the functionality of the system adequately.

Error

An error is a system state that could lead to a failure of the system. A failure occurs when an error is propagated to the interfaces of the systems.

Fault

The adjudged or hypothesized cause of an error is denoted as a fault. There are different classifications of faults with respect to various criteria such as the nature of the fault, the domain, the persistence of the fault, the location and the frequency of appearance.

2.3 Architecture Paradigms in Real-time Systems

In the last years the increasing employment of embedded systems for the construction of different real-time applications has caused the design of multiple control subsystems to fill the various and controversial domain requirements. These subsystems are required to meet stringent specifications to accomplish the dependability attributes. From this evolution, the following embedded system architectures can be distinguished:

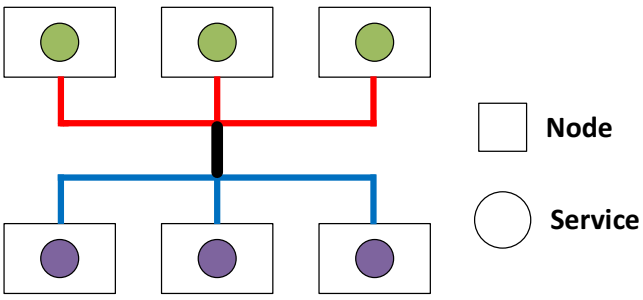

Figure 2.1: Federated Architecture

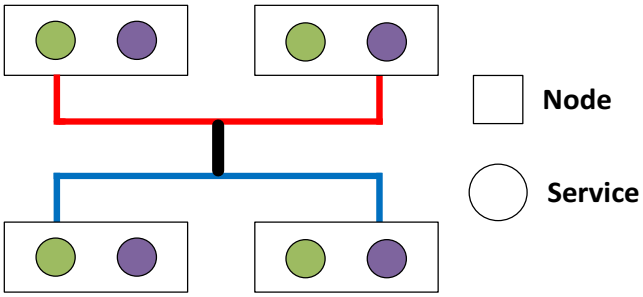

Figure 2.2: Integrated Architecture

2.3.1 Federated Architecture

In this architecture each node is responsible of the execution of a maximum of one service, as illustrated in Figure 2.1. In this figure a distributed system is constituted by two applications, wherein each application service is allocated to a single node. Moreover, a service can run several tasks within its own node.

The federated architecture is highly used for the implementation of distributed systems performing applications with different criticality, because of the loose coupling between the application nodes. As a disadvantage, the federated architecture implies a high number of resources, and therefore, high hardware costs, size and weight.

2.3.2 Integrated Architecture

Oppositely to the federated architecture, the integrated architecture allocates different services to the same node, which share hardware resources (input and output interfaces, external memory, etc) and use the same physical communication channel for the communication with services in other nodes (see Figure 2.2).

A drawback of the integrated architecture is potential interference between services due to the sharing of resources. On the other hand, such an architecture reduces the hardware costs, as well as the weight and size of the distributed system.

2.4 AUTOSAR

In 2003 the automotive industry has founded the development partnership AUTomotive Open System ARchitecture (AUTOSAR). This partnership is intended to counteract the skyrocketing costs, as well as the growing complexity and versatility of the manufacturing tools used to create automotive Electronic Control Unit (ECU) software. The aim of this consortium was to provide a uniform software architecture for the cost reduction, the quality improvement and the re-usability of existing developed components. This section presents the reasons for the introduction of this new software architecture. Subsequently, the components of AUTOSAR and the ECU architecture are explained.

2.4.1 Motivation

The number and the complexity of the faced challenges in the development of ECU software have been growing steadily in the recent years. More and more functionalities for deeply embedded automotive systems such as door, tail, light, engine and sun roof control devices are required. These controllers form a highly distributed and networked system with the required installed functionalities.

Additionally, the development of the ECU software used to be centralized and not function-based. A function-based approach means that, for example, a warning flasher control function requires a hazard warning flasher ECU. However, as the number of required functionalities grows extremely fast, a function-based approach leads to an unmanageable system of ECUs, where some of them perform similar tasks. In a centralized ECU approach, for instance a turn signal system in the car, the ECU is the subject of various functions, e.g., remote control for the central locking, warning flash button, etc., and providing an own ECU for each one of these functions makes no sense.

However, a centralized ECU approach leads to complex software with a high potential for errors. In addition, there is a large variety of different hardware components of the ECUs, with different processors, computing powers and memories. Each Original equipment manufacturer (OEM) also requires the employment of its own operating system (basic software) for the ECUs it orders from the Tier1 suppliers. All this leads to requirement changes being complex and expensive.

Another major problem in the production of the ECU software is the strong mixing of functional code (application logic, algorithms) with technical (hardware-specific) code. This makes the reuse of already created application logic very difficult, time-consuming and expensive.

Objectives

The automotive market is a mass market with an enormous competitive pressure. Consequently, the top priority is the cost reduction. Following their motto *"Cooperate on standards, compete on implementation"*, AUTOSAR achieves a strict separation of functional and basic software code. AUTOSAR precisely specifies the implementation of the basic software and its modules for the ECUs.

The technical portion of the ECU software is standard and is specified by AUTOSAR, while the application logic or functional code expresses the competition between the various manufacturers. Further goals of AUTOSAR are the easy interchangeability of modules and the reuse of software components of the functional code. These goals should lead to a simplification of the work processes of all participating companies, which should then be reflected in a cost reduction.

Furthermore, the strict separation of technical and functional code can also reduce hardware costs. AUTOSAR avoids unnecessary code resulting from mixing functional and technical code, and thus, allows the use of an ECU with the next smallest memory.

Advantages

AUTOSAR introduces a standardized software architecture that guarantees clear hardware independence as a layered model. This immensely facilitates the reuse of application logic and the implementation of requirement changes. The modularity improves the quality of the software, since individual module tests can be performed. The loose coupling of the modules makes it easy to use them in various software projects. If a module with a certain algorithm is already used correctly in a software project for a long time, it can be used with little effort in another software project. The error-free running time of the module in the previous project represents a strong quality statement. Furthermore, the functional or architecture-based view is used via AUTOSAR. First, all the functions to be realized are collected and developed and only after this, the application logic is assigned to the various ECUs. Moreover, AUTOSAR uses exactly one basic software whose modules are described precisely. This has several advantages:

- The interchangeability of modules is relatively easy.
- Suppliers who serve several different OEMs no longer need to maintain their own basic software and development tools for each OEM.
- Already developed application logic is easy to reuse.

- The system integration on the part of the OEM is simplified.
- Due to the late assignment of the application software to the ECU, there is a high degree of design flexibility.

Another advantage for suppliers is the ability to establish new business models. Suppliers can offer finished libraries to software components with pure functional code, or develop and sell combined hardware-software products in the field of sensors-actuators. Distributed development with offshore forces is also conceivable, as the AUTOSAR standard enables a clearly defined division of labor.

2.4.2 System View

In the system view, the automaker has the overall view of the vehicle type and all functions to be created for that vehicle. Now the *"System Configuration Input"* file is created in the first step. This is a predefined AUTOSAR template, in which the planned software and hardware components for realizing the functions are recorded. The step *"Configure System"* decides which software components are used. The development of the software components can be carried out by the OEM or given to others. Only after successful completion of this step, it is decided on which ECU the software components are to run. This decision also defines the required communication network between the various ECUs and the actuators and sensors required based on the software components running on them. The result of this work is stored in the *"System Configuration Description"* file. From this file, in the *"Extract ECU specific Information"* step, the car manufacturer individually extracts the *"ECU Extract of System Configuration"* file for each ECU and gives it to the suppliers, who can then produce it in the ECU view.

2.4.3 AUTOSAR Software Components

The AUTOSAR Software Components (SWCs) serve as a modeling tool for structuring and arranging the functional application. The actual code is encapsulated in so-called *"runnables"* within the SWCs (see Figure 2.3). A distinction is made between different SWCs, with special mention being made of the SWCs for actuators and sensors. They are the only ones that are not hardware-independent, as they are tied to actuators and sensors of specific manufacturers and the corresponding ECUs.

The exchange of information between SWCs takes place via so-called ports. The AUTOSAR standard defines ports as the interconnection points between SWCs in the AUTOSAR architecture to indicate the data flow between these SWCs [AUT16d]. The

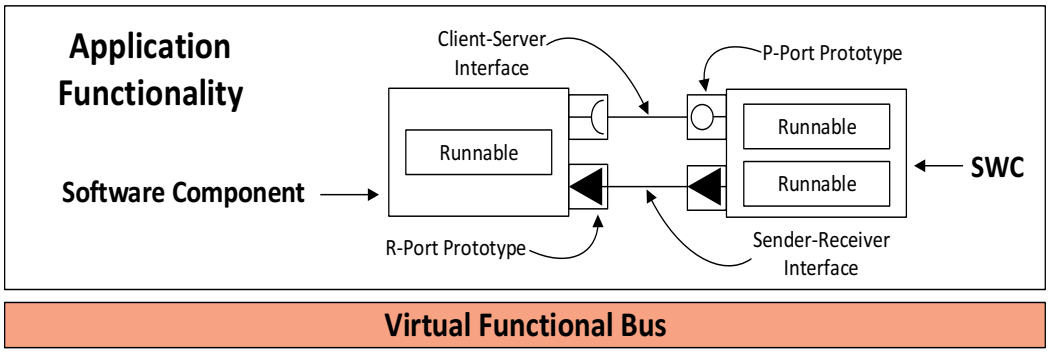

Figure 2.3: Application View

interconnections between the SWCs through the ports conform to the so-called AUTOSAR Virtual Funcitonal Bus (VFB), which represents the communication relations between different SWCs, independently of the ECU mapping of the SWCs.

Port interfaces define the information that is transported through the ports. Ports, which will be connected to each other, must have compatible interfaces. According to [AUT16d] three different port interfaces are distinguished:

- *Sender/receiver interface,* for sending and receiving data values, the so-called variable data prototypes.
- *Client/server interface,* for invoking and deploying operations. The argument data values handled by these operations are called argument data prototypes.
- *Calibration interface,* for the provision and the use of static calibration values.

Application data types represent data types linked to the variable data prototypes and the argument data prototypes. Computation methods define the scaling conversion between the physical and the internal representation of the data [AUT16d]. Data-constraint elements restrict the physical range of values. Units represent the physical dimensions. Constant specifications of the application data types are assigned to the SWC input ports in order to define initial values [AUT16d].

An internal behavior provides means for formally defining the behavior of a SWC [AUT16d]. It is characterized by the runnables, exclusive areas and per instance memories. Exclusive areas represent critical sections that may not be interrupted by other runnables, while per instance memories are used for the exchange of array-based or structure-based variables between two runnables [AUT16d].

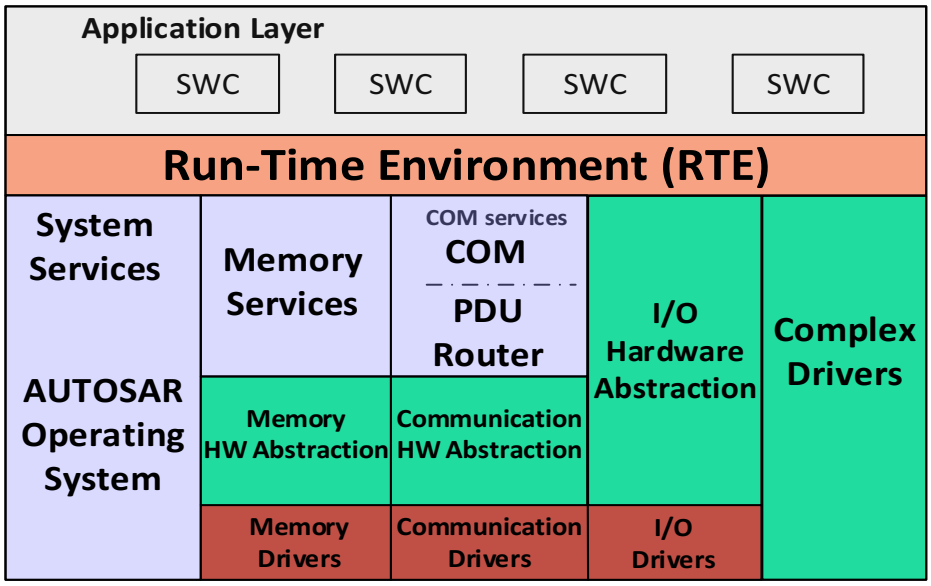

Figure 2.4: AUTOSAR ECU Architecture

2.4.4 AUTOSAR ECU Architecture

The AUTOSAR architecture is a layered model in which each layer is further abstracted from the underlying hardware. Thus, this architecture layer model exhibits typical properties. The layers are loosely coupled. Each layer knows only the underlying layer. Exceptions to the rule are the *"complex drivers"*, which will be discussed later. Figure 2.4 shows the AUTOSAR architecture with its different layers.

2.4.4.1 Application Layer

In this layer, we find application SWCs with their functional code. The development of this takes place here independent of the vehicle bus and the hardware used. The SWCs for sensors and actuators are the exception here, as mentioned previously.

2.4.4.2 Run-time Environment

The AUTOSAR Run-Time Environment (RTE) functions as the VFB at the ECU level. The RTE ensures the communication between the SWCs depending on the assignment of the SWCs to the ECUs. If two SWCs communicating with each other are located on the same ECU, the RTE establishes the communication link directly between the SWCs. If the SWCs are located on different ECUs, the connection to the other SWC is realized by the RTE via the basic software and the vehicle communication bus.

2.4.4.3 Basic Software

The AUTOSAR Basic Software (BSW) layer itself is divided into three layers:

- *The service layer* contains the operating system and system services for the application layer. This layer abstracts the RTE from a direct access to the underlying layers of the BSW. The operating system is an OSEK [Joh98] operating system that adds features such as memory protection or extended counters. Additionally, system services include diagnostic and communication functions as well as memory management.
- *The ECU abstraction layer* abstracts upper layers from the underlying ECU hardware, for example a Controller Area Network (CAN) controller mounted onboard for accessing the communication bus. Furthermore, this layer provides an I/O abstraction with direct access to the RTE so sensor and actuator SWCs can be serviced.
- *The Micro-Controller Abstraction Layer (MCAL)* is directly above the hardware and depends on the installed micro-controller. It allows the initialization and configuration of the micro-controller. If the micro-controller is replaced, then this layer must be completely replaced.

The complex drivers completely bypass the layer logic established by the AUTOSAR standard. This section is only used for the implementation of functionalities that are not contained in the AUTOSAR standard yet. Moreover, in case of time-critical features, a complex driver allows a faster access to the hardware.

Furthermore, the AUTOSAR BSW is divided into five vertical areas, the so-called stacks. The stacks and the layers overlap and form the so-called functional blocks. Within a functional block, the AUTOSAR standard defines so-called modules. The modules within a function block have similar tasks or work together to complete a task. Figure 2.4 illustrates the stacks and the functional blocks.

- *System Service Stack:* It is responsible of the provision of basic network, diagnostic services and software arbitration services. Also, the AUTOSAR operating system is located here .
- *Memory Stack:* It allows the storage in non-volatile memory. It consists of the blocks *"Memory Services"*, *"Memory Hardware Abstraction"* and *"Memory Drivers"*.
- *Communication Stack:* This stack provides communication services for exchanging data with other ECUs for the application layer and the BSW layer. This stack includes the *"Communication Services"*, the *"Communication Hardware Abstraction"* and the

"Communication Drivers" functional blocks. The communication services support the routing of messages (i.e., COM module, Protocol Data Unit (PDU) module), the multiple use of a channel by multiplexing and the provision of the transport protocols for the off-chip network systems, e.g., CAN, Local Interconnect Network (LIN) and FlexRay. Moreover, the communication hardware abstraction abstracts the bus-specific hardware (e.g., CAN, FlexRay, Ethernet). The interfaces provide functions for accessing the available channels of the respective bus system.

- *I/O hardware stack:* This stack concentrates all functional blocks for setting and reading digital input and output values. The stack contains the functional blocks *"I/O Hardware Abstraction"* and *"I/O Drivers"*.

Chapter 3

Analysis of the State-of-the-Art

Over the years, the construction of embedded systems in different application domains has resulted in the development of several domain-specific system architectures. For example, the AUTOSAR standard, which is the predominant standard in the automotive domain, Integrated Modular Avionic (IMA) [Aer15] for the aerospace domain and the Network on Terminal Architecture (NoTa) in the mobile domain [KKOE07].

Furthermore, different Multi-Processor System-on-a-Chip (MPSoC) architectures have been developed for specific application domains (e.g., CellBE [IST06], Sonics [Son02] and Nostrum [MNT+04]). All of them focus on a shared memory approach for the communication between the cores.

Within the European initiative ARTEMIS [ART06] common challenges were identified on specific-domain architectures (e.g., composability, predictability, robustness, security). Driven by these domain-independent challenges, the European research project GENESYS [OK09] developed a MPSoC reference architecture blueprint that can be universally implemented on different application domains as automotive, avionic, industrial control, mobile and consumer electronic systems. Moreover, during the European ARTEMIS project ACROSS [SEH+12] a MPSoC architecture was developed which was specifically designed for safety-critical embedded system applications. These two architectures introduce a Time-Triggered Network-on-a-Chip (TTNoC) for the communication between the cores in order to provide the advantages of this king of networks to the multicore system. The benefits of the implementation of such a message-based MPSoC architecture specifically for the automotive domain are well discussed in [HO09].

In the rest of this chapter we present an example of a comercial multicore platform highly used at the moment in the automotive industry for the development of ECUs, the so-called Aurix TC3xx from the Infinieon multicore family. Thereafter, as an example of the on-going state-of-the-art in the area of Network-on-Chip (NoC)-based MPSoC platforms the

results of the MESARA and parMERASA projects in the context of automotive applications are analyzed. Furthermore, a discussion of available I/O multicore solutions is presented followed by the description of the health monitoring service developed by the avionic domain. Finally, the multicore solution developed by the AUTOSAR Consortium is presented together with the research gap, which compares this existing AUTOSAR multicore version with the AUTOSAR message-based multicore architecture proposed in this dissertation.

3.1 AURIX TC3XX

The Aurix TC3xx family of microcontrollers is developed by Infinieon technologies specifically focused for electric and automated vehicles. The multicore architecture of the Aurix TC3xx family contains up to six independently operating 32-bit TriCore processor cores. Together with the enhanced features for vehicle communication, data security and functional safety, the TC3xx microcontrollers are currently used for the development of many vehicle applications, e.g., the control of motors, transmission systems and hybrid and electric drives. Especially the domain controllers for hybrid drives are been benefited from TC3xx microcontrollers as well as the battery management and AC / DC converter. TC3xx microcontrollers are suitable for safety-critical applications such as airbags, brake systems and power steering, as well as radar- or camera-based driver assistance systems. The high real-time capability and the comprehensive security functions make the TC3xx family suitable for data fusion, and thus, for the implementation of automated vehicle applications [DMK+18].

The Aurix TC3xx microcontrollers offer up to 16 MB of embedded flash memory and more than 6 MB of built-in RAM. Four of the six TriCore cores have an additional Lockstep core. Up to 2,400 DMIPS of computing power is available to design systems with the highest level of safety, i.e., ASIL-D of the ISO26262 standard [Int11]. The high computing power and the reusability of existing safety concepts enable automotive system manufacturers to save a grand part of development time. In addition, multiple feature can be implemented on a single microcontroller, e.g., a powertrain domain control together with vehicle dynamic applications.

An improved Hardware Security Module (HSM) is offered by every TC3xx microcontroller. The HSM makes on-board communication more secure and hampers hardware manipulation such as engine tuning. It integrates new features to support asymmetric encryption mechanisms according to the EVITA project requirements [KZK+14]. This allows Aurix software update over-air and helps to prevent software hijacking.

The Aurix TC3xx supports the latest communication interfaces, which allows it to host complex gateway and telematic applications. The Aurix is provided with a Gigabit Ethernet

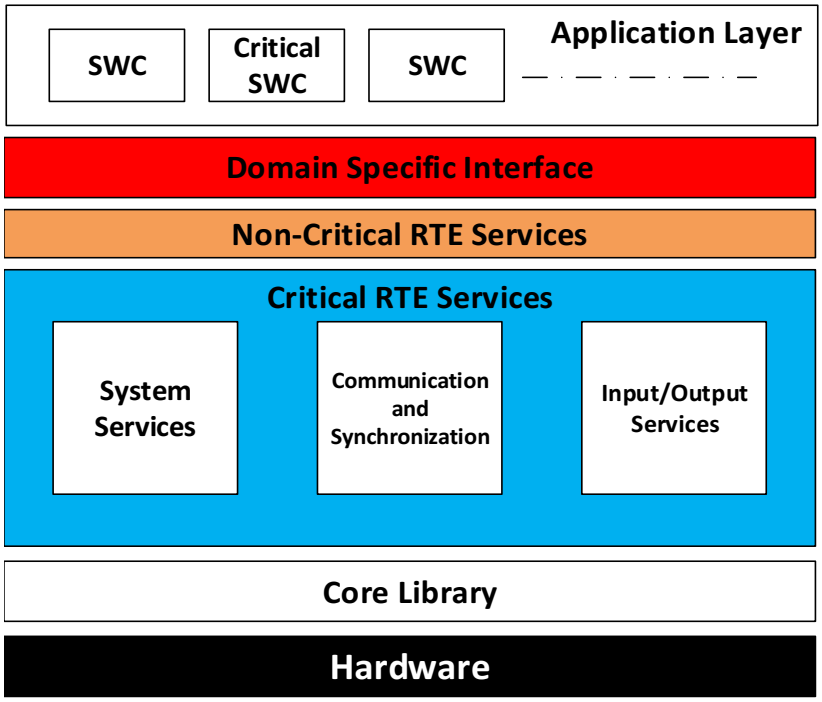

Figure 3.1: parMESARA Software Architecture

interface, up to 12 CAN with Flexible Data-Rate (CAN FD) channels according to ISO 11898-1 [Int15] and up to 24 LIN channels. Additionally, an integrated eMMC interface for external flash interface enables local data storage.

3.2 MERASA and parMERASA European Projects

The Multi-Core Execution of Parallelised hard Real-Time Applications Supporting Analysability (parMERASA) EU project [UBG+13] was carried out to investigate industrial real-time programs for their possible performance enhancement by means of suitable parallelization together with user companies from the fields of aircraft electronics, automotive technology and construction machinery. The focus was on moving from a sequential to a parallel real-time program that still meets real-time requirements despite the simultaneous execution of the parallel control threads on a multicore processor. In order to achieve this goal, a corresponding software design process was developed, suitable parallel software structures were found, and different hardware structures were examined for their suitability.

The parMERASA was developed on the resulting basis obtained from the initial Multi-Core Execution of hard Real-Time Applications Supporting Analysability (MERASA) project [UCS+10]. Both projects were coordinated by Prof. Ungerer, being the first one

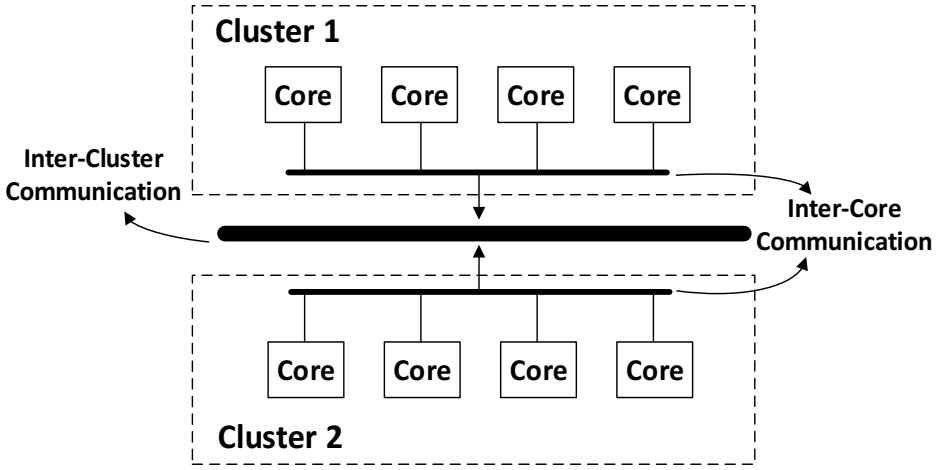

Figure 3.2: parMESARA Clustered System Architecture

focused on developing real-time multicore processors with two to eight cores, which are, on an order of magnitude, the today already being used general-purpose processors for PCs and servers. The parMERASA followed-up project continued this investigation to multicore processors with up to 64 cores and with other new connection structures. While the focus of research and development work in MERASA was on the hardware development, in parMERASA it shifted to user programs and their parallelization as well as the support of the system software.

Figure 3.1 depicts the software architecture approach introduced by the parMERASA project. The aim of this architecture is to ensure that timing assumptions performed on the application level development are full filled. For this, the architecture differentiates between no critical and critical services providing protection environments to avoid access across partition boundaries except if it is explicitly allowed (e.g., memory mapping). Thus, only critical services can influence other partitions.

Depending of the application domain these protection environments are defined to comply with specific domain requirements. For instance, Figure 3.1 introduces AUTOSAR tiny RTEs for critical services and for no critical services, abstracting the application software from the BSW functionalities, e.g., system services (scheduling, protection), communication and synchronization services and I/O interfaces.

Additionally, the project proposes to build the parMERASA architecture on top of a clustered processor architecture, where cores in the same cluster are connected through a NoC and cores on different clusters are connected using a inter-cluster NoC (see Figure 3.2). Thus, inter-core communication running within the same cluster does not interfere with applications running on a different cluster (intra-cluster communication). Only in case of communication between cores allocated on different clusters (inter-cluster communication)

messages are routed through the inter-cluster NoC to the application running in the specific cluster.

3.3 State-of-the-Art of I/O Multicore Solutions

In the last years I/O management in combination with multicore processors has been investigated. A review of different solutions for maintaining coherency between caches and the data generated or consumed by I/O devices is presented in [Ber09]. This work compares different approaches for data and I/O coherence with solutions trading off hardware versus software complexity depending on the application and the system characteristics.

A high performance multicore I/O manager for the Glasgow Haskell Compiler (GHC) is introduced in [VWHY13]. The so-called Mio manager eliminates the bottlenecks originating from a typical GHC I/O manager when implemented on a multicore processor. In [JGJ+09] a virtual regionalized NoC is used to optimize the performance of the peripheral devices. This work presents an architecture consisting of a NoC with a mesh topology wherein the network is divided into several virtual regions taking advantage of the characteristics of the applications and their communication patterns to adapt to the I/O communication requirements.

A heterogeneous multicore embedded system with virtualization to improve security and isolation among virtualized environments is presented in [KGC12]. This multicore embedded architecture consists of an I/O management unit that enables the virtualization and provides support for a global coherent address space, flow isolation, security, resource management and runtime monitoring. In [KYBS14] a partition scheduler providing conflict-free I/O for multicore avionic systems is introduced. This work proposes a heuristic algorithm that prevents conflicts among I/O transactions from applications running in different cores.

3.4 ARINC 653 Health Monitoring

The Aeronautical Radio, Incorporated (ARINC) 653 [Aer15] is an avionic standard for integrated modular avionics which defines an execution environment based on time partitioning for safety-critical avionic Real Time Operating System (RTOS). It describes a general purpose application interface between the operating system and the application software.

This standard introduces a health monitoring service [ZW+13] which provides a framework to raise and handle alarms in a system consisting of three levels in a hierarchical fashion: process level, partition level and module level (see Figure 3.3). A system health monitoring table defines the level of an error (module, partition, process) based on the error and the state

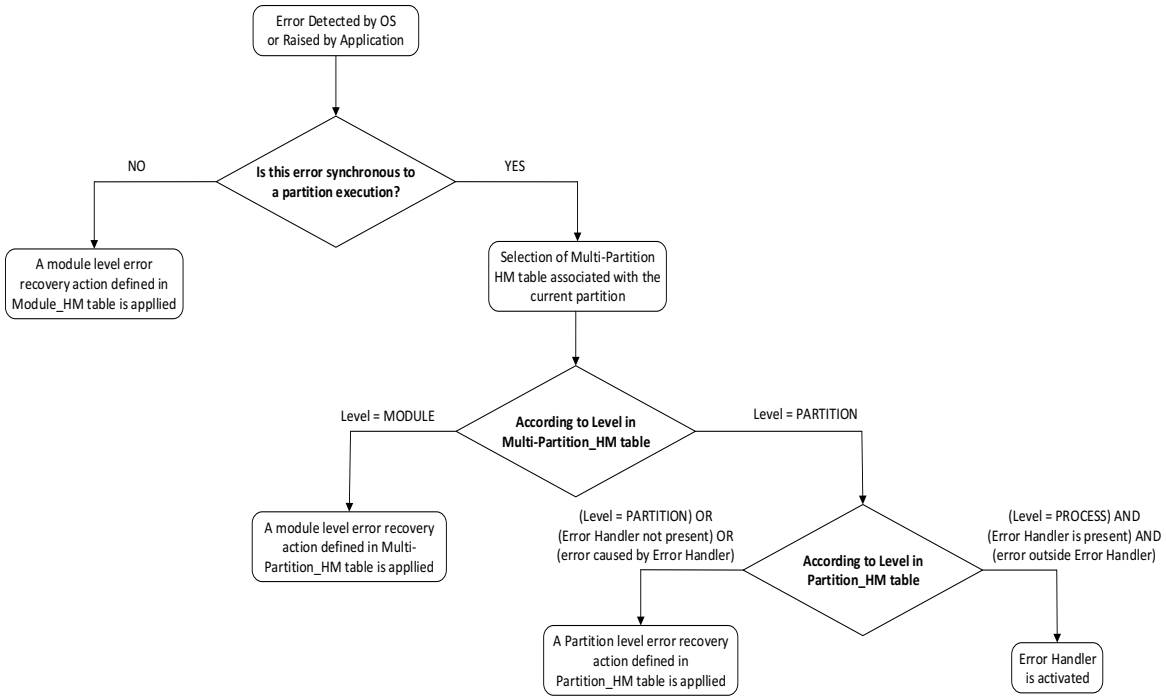

Figure 3.3: Health Monitoring Decision Tree

of the system. Additionally, fault responses and recovery actions are defined depending on the error level.

Recovery actions for process level errors are defined by the application programmer, while recovery actions for the partition and module level are specified in the health monitoring configuration tables. The partitions can have separated configuration tables or share a common table.

3.5 Multicore Approach of AUTOSAR

The version 4 of the AUTOSAR standard specifies the first (and still up to date) version with multicore support for its ECU architecture. With this first multicore support, AUTOSAR found a path to a multicore architecture that makes minimal changes to its original system architecture. As a result, the cost of conversion should be kept relatively low. One crucial aspect of the AUTOSAR design was exploited: all SWCs in a vehicle network can be freely distributed to the ECUs in the network at design time, as the VFB provides an abstraction from the actual ECU used. In a multicore system, the VFB now also provides an abstraction of the cores. As a result, SWCs can also be used in a multicore system without any further

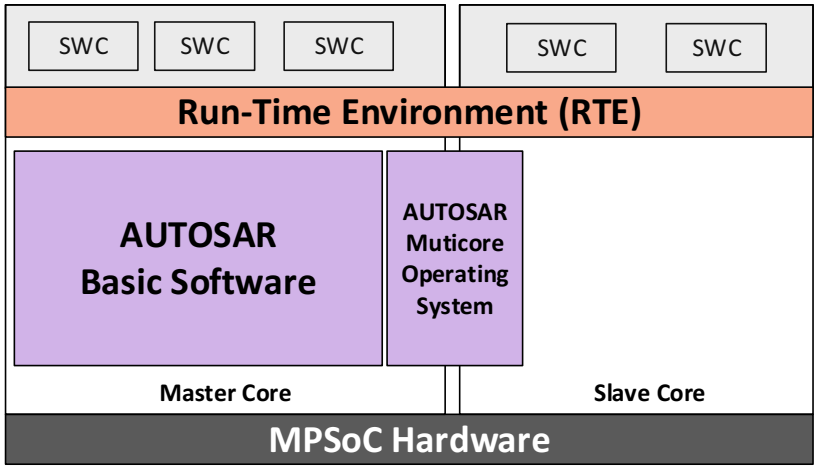

Figure 3.4: The AUTOSAR Multicore System

adaptation, so in other words, there is no migration effort on the application development side.

Even the BSW does not have to be completely reworked. The operating system and the RTE are both adapted. The operating system must be able to execute tasks on multiple cores and the RTE must now implement the VFB in such a way that, in addition to the well-known local and external communication, a third form is supported, namely the cross-core communication within an ECU. However, the rest of the BSW does not need to be specially adapted to the multicore system according to AUTOSAR. This is achieved by using the BSW only on one, previously determined, master core. This ensures that any implicit or explicit synchronization within the BSW will continue to work just as well as the one-core system does. The resulting architecture is shown in Figure 3.4.

The largest amount of customization affects the operating system [AUT16c]. This AUTOSAR multicore operating system can execute tasks on all cores of a multicore processor. The task-core assignment is done statically as part of the configuration by always assigning an operating system application to exactly one core. Since synchronization using existing OSEK resources only applies to mutually exclusive tasks and not to parallel operations, resources are assigned to a core and may only be assigned to tasks on the same core.

To synchronize tasks on multiple cores, AUTOSAR spinlocks are introduced. Here, AUTOSAR uses a term different from its usual definition. The term spinlock usually refers to an application-level synchronization mechanism wherein the involved processes synchronize based on the value of a memory location. Waiting processes actively check the value of the memory location and therefore occupy the processor once it is their turn. Spinlocks are favored where low competition and short block times are expected, as they bring very little overhead. In return, AUTOSAR spinlocks are provided entirely as a unit independent service

to the operating system. To get the semantics of spinlocks, a task that requests an occupied AUTOSAR spinlock is blocked by the operating system. However, executable low-priority tasks are not assigned to the processor in order to give the impression that the blocking task would still occupy the processor. The reason for this heavy weight approach is to try to avoid deadlocks. Therefore, the assignment of an AUTOSAR spinlock is linked to several conditions, for example, a certain order must be adhered to when using several spinlocks. These conditions are checked by the operating system before the AUTOSAR spinlock is actually allocated. If the conditions are violated, the operating system returns with an error message without occupying the AUTOSAR spinlock.

Most operating system services are available across the cores. Examples include activating tasks or setting events. The exception are the functions for interrupt locks, which always only affect the calling core. The AUTOSAR operating system is also enhanced with a subsystem for signal-based communication across core and memory protection boundaries, the so-called Inter OS-application Communicator (IOC). The IOC is primarily designed tu be used by the RTE, offering communication channels with or without queues. Each channel must be configured with the data type and, if necessary, the length of the queue. The AUTOSAR operating system then manages corresponding memory areas.

As another module of the AUTOSAR system, the RTE is adjusted, but here the changes are less extensive. During the generation phase of the RTE, it must be checked for each local communication whether it exceeds the limits of an AUTOSAR operating system application. If this is the case, the communication should be routed through a suitable IOC channel. If the calling SWC is assigned to another core, then the function call is not allowed take place from its own task. The RTE therefore implements all BSW calls from such SWCs as remote calls. For this, the RTE sends the call parameters to a proxy task on the first core. This task then invokes the appropriate function. This procedure is shown in Figure 3.5 for SWC B. If the function has a return value, it is sent back to the calling core. Task B is in the meantime blocked and waits for the answer. This provides the synchronous semantics of a function call to the calling SWC.

3.6 Limitations of the existing AUTOSAR Multicore Version

In the AUTOSAR multicore version the cores share the same configuration and code, while different data is processed. The complete BSW is provided only on the master core, while the slave cores have a subset of the AUTOSAR BSW to support SWCs and complex device drivers. The described IOC service is used for the communication between the cores. This

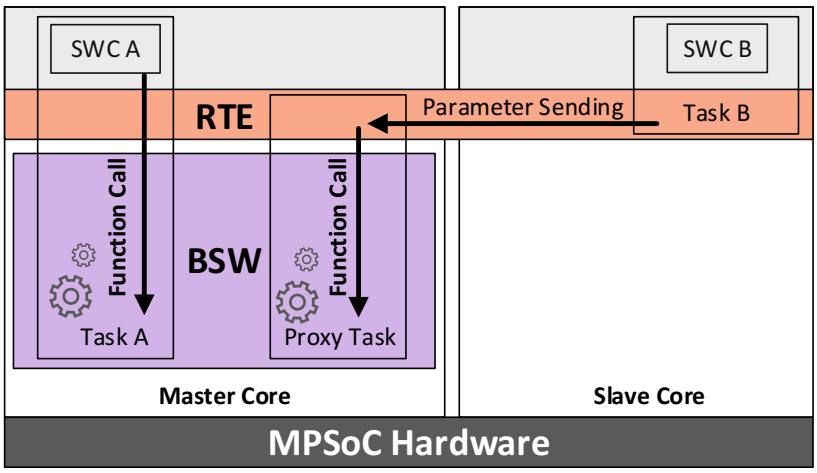

Figure 3.5: The IOC Functional Concept

type of communication is implemented by shared memory [AUT16c] and is used by the AUTOSAR RTE for the communication between SWCs on different cores.

This AUTOSAR multicore version gives a limited autonomy to the cores, which are managed by a common operating system. Dependencies between cores, for example, by the possibility of activating tasks, task chaining or setting alarms/events on other cores, can lead to error propagation between cores in case of permanent and transient hardware errors (e.g., soft errors) and design errors. These dependencies require the analysis of the temporal behavior of each core under the influence of the other cores on the MPSoC. The starting and stopping of individual cores is not supported, and only the complete MPSoC can be put into "*sleep mode*". Moreover, the master core represents a single point of failure and provides limited scalability.

Finally, the sender/receiver communication is the only communication type supported, while the client/server communication is not provided. Also, "*1:N*" communication is not directly supported but implemented with multiple interactions. For synchronization between cores busy-waiting with spinlocks is used, so configuration tools need to check offline that no deadlocks occur when using the spinlocks.

3.7 Research Gap of the State-of-the-Art

The AUTOSAR multicore operating system lacks support for message-based MPSoC architectures, whereby the advantages of this architecture model for temporal predictability, fault isolation, low overhead, energy efficiency and scalability can not be realized for AUTOSAR-based systems. The development of such an AUTOSAR NoC platform represents a major

scientific challenge, where several requirements must be fulfilled: ***(1)*** A requirement is the mapping of the VFB to a message-based NoC. ***(2)*** Interfaces that can also be bypassed for performance reasons within the AUTOSAR BSW and the possibility of direct access through cross-layer shortcuts represent a risk to the realization of the AUTOSAR BSW functions on different cores. ***(3)*** The limited local memory resources within the cores are another technical risk. ***(4)*** Finally, the complete compatibility with existing interfaces and specifications (e.g., AUTOSAR RTE, AUTOSAR operating system) is a challenge of central importance.

Furthermore, the state-of-the-art does not provide I/O management as part of an AUTOSAR message-based multicore platform. Moreover, the support for remapping between input/output cores and application cores is required for the integration of a health monitoring service and recovery actions to the AUTOSAR multicore system.

In this Dissertation we propose a time-triggered multicore architecture for AUTOSAR that offers significant advantages in comparison with the existing AUTOSAR multicore operating system version. The followed advantages are validated in the results presented in Chapter 7.

- *Better reliability.* The message-based NoC and autonomous AUTOSAR application cores provide fault isolation in the time and value domains. Fault isolation is a weak point in the AUTOSAR multicore operating system [DBT09].

- *High performance with low overhead.* The parallel execution of SWCs on multiple cores, system cores with hardware support for gateways, I/O and memory bus and simplified BSW increase the performance and reduce the resource requirements in the AUTOSAR layer model.

- *Temporal predictability.* The message-based NoC and the decoupling of the cores lead to deterministic temporal behavior and suitability for real-time applications.

- *Smooth integration.* Autonomous Micro-Electronic Control Units (μECUs) (application cores) lead to clear responsibility and support for seamless integration of independently developed SWCs.

- *Seamless communication with SWCs on other MPSoCs.* Gateway system cores support seamless communication between the chip-level and distributed systems, especially with the support of time-triggered off-chip networks such as FlexRay [Fle05].

Chapter 4

Message-based Multicore Architecture for AUTOSAR

In this chapter we present a novel message-based multicore architecture for AUTOSAR which combines the AUTOSAR software with a multi processor NoC platform. Thus, the benefits of message-based NoC architectures discussed in Chapter 1 in the context of temporal predictability and fault isolation can be exploited by the AUTOSAR system.

The architecture defines a message-based NoC as the only physical medium for the communication between the cores and introduces autonomous application cores which function as AUTOSAR μECUs on the MPSoC. Each μECU acts as a unit of abstraction where the SWCs are provided with a RTE and a lightweight implementation of the AUTOSAR BSW. Additionally, we extend the AUTOSAR BSW on the μECUs with new BSW modules to connect them with the on-chip network and dedicated system cores that serve as hardware accelerators to the AUTOSAR application. Moreover, a health monitoring service is integrated which provides recovery actions to the AUTOSAR software in case of software failures in the time domain or value domain.

4.1 Overview of the AUTOSAR Multicore System

The proposed system architecture, denoted from now on as TIme-triggered MEssage-based multi-core architecture for AUTOSAR (TIMEA) [UO15] [Urb17], describes a message-based multicore architecture for the mapping of the single-core AUTOSAR ECU software architecture to a message-based multi/many-core system. As depicted in Figure 4.1, this architecture introduces a message-based NoC as the communication interface between the cores. Two types of cores can be distinguished: application cores and system cores. While the application cores, so-called AUTOSAR μECUs, are in charge of the execution of the

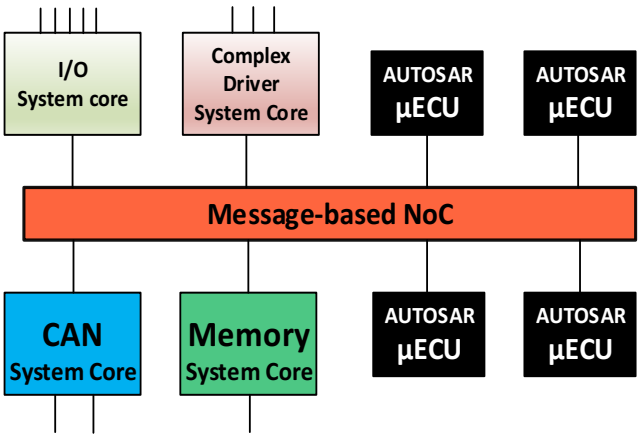

Figure 4.1: TIMEA Platform

specific automotive application functionality, system cores perform hardware acceleration services needed by the μECUs.

A time-triggered NoC supports bandwidths of several *Gbps* and provides communication plans with precise phase positions of messages in the range of a few *ns*. Since most of the functionalities realized by the AUTOSAR BSW are designed for latency requirements with orders of magnitude no lower than 1*ms*, a NoC for inter-core communication allows the introduction of dedicated system cores for replacing computationally expensive functionalities originally handled by the BSW modules in the μECUs. In Figure 4.1 we propose system cores dedicated to I/O functionalities (e.g., PWM, ADC functionality, etc) [UO17], for replacing special complex driver implementations, to provide memory services [OO16] and to implement off-chip communication networks for supporting inter-MPSoC communication [UO16].

4.1.1 Message-based Network-on-a-Chip

The proposed message-based NoC provides temporal predictability and fault isolation to the AUTOSAR μECUs and the system cores. There is a considerable trend towards multiple timing models in on-chip communication systems to overcome trade-offs in terms of predictability and flexibility [Edi12]. We assume that the TIMEA NoC provides support for these timing models to fulfill the different and partially contradicting communication requirements of different application subsystems in real-time embedded systems such as those in the automotive industry. Different available NoCs satisfy this assumption, e.g., AEtheral [GD$^+$05] and DREAMS-NoC [AO15].

Time-Triggered (TT) communication: this kind of communication specifies messages whose transmission times are stored in a static communication schedule, where each message

is assigned to a specific time slot with respect to a global time base. Time-triggered messages ensure temporal predictability since resource conflicts with other time-triggered messages are eliminated at development time.

Rate-Constrained (RC) communication: A sufficient bandwidth allocation for each message transmission is guaranteed with a defined Minimum INter-arrival Time (MINT) and temporal deviations. Rate-constrained messages provide flexibility and a high utilization of resources. Priorities are used to resolve contention between multiple rate-constrained messages and contention with time-triggered messages. Time-triggered messages are assigned higher priority than rate-constrained ones.

Best-Effort (BE) communication: It supports the transmission of messages that are triggered by the occurrence of significant events in the environment or inside the system. Priorities are also used to resolve contention but in comparison with time-triggered and rate constrained messages, this kind of messages are assigned lower priority and they may be lost.

The message-based NoC is composed of Network Interfaces (NIs) and routers. Each μECU and system core of the MPSoC is provided with a NI. These NIs serve as the interfaces for the μECUs and the system cores to access the NoC by injecting the messages from the cores into the NoC as well as delivering the received messages from the NoC to the cores. Routers relay the fixed-length fractions of messages from the sender NI to the destination NI. Physical links serve for the interconnection between the NIs and the routers.

The message-based NoC provides support for different configurable topologies (e.g., mesh, ring) with a global time base for the temporal coordination of the message transmissions between the μECUs and the system cores. For the execution of the NoC, the configuration parameters of the required topology have to be defined at design time. This a priori knowledge is used later to setup the NoC and define its NIs and the static communication schedule.

4.1.2 Application Cores and System Cores

Application cores implement the required automotive functionality by hosting one or more AUTOSAR SWCs. The application cores represent AUTOSAR μECUs in the MPSoC platform.

Each application core is provided with a middleware implementing the AUTOSAR RTE. The purpose of the RTE as defined by AUTOSAR is to provide an interface to the SWCs that makes them independent from the underlying platform and from the mapping to a specific ECU hardware [HBS+06]. SWCs can access via the RTE the following AUTOSAR services [AUT16b]: memory services (nvram manager), system services, communication services (e.g., CAN, LIN, FlexRay), input/output hardware, and complex drivers.

In the proposed AUTOSAR MPSoC, the RTE in the application cores offers access to system services (i.e., AUTOSAR Operating System (OS)) and communication services. The communication services are mapped to the NoC transparently to the application software. This provided version of the AUTOSAR RTE serves as the only interface for the interaction between SWCs on different application cores. The application cores require only a reduced realization of the AUTOSAR BSW, wherein BSW modules for memory services, I/O hardware abstraction and complex drivers are replaced by stubs to dedicated system cores. The motivation behind this approach are the improved performance and lower overhead for AUTOSAR in the μECUs since the system cores act as hardware accelerators.

Furthermore, BSW modules in the "*ECU Communication Abstraction Layer*" are added for supporting access to the message-based NoC. In contrast, BSW communication modules for network off-chip accesses are also allocated to dedicated system cores.

The VFB defined by the AUTOSAR standard is available for the AUTOSAR SWCs through a hierarchical platform. The communication hierarchy is performed as follows:

- ***Inner-core communication:*** the exchange of messages between SWCs on the same μECU is performed by the RTE of the μECU.

- ***Inter-core communication:*** the communication between SWCs on different μECUs in the same MPSoC is available through the extended COM service modules for on-chip communication and the NoC, based on the pre-defined on-chip communication schedule of the network.

- ***Off-chip communication:*** The passing of messages between SWCs in different MPSoCs is possible by using a dedicated system core supporting any off-chip communication network.

The advantages of this approach are the reduced requirements on operating system functionality within a single application core as well as the inherent encapsulation of the μECUs using the message-based NoC.

The system cores are deployed to implement parts of the AUTOSAR BSW. System cores implement particular AUTOSAR functions such as gateways or input/output functions. The system cores perform virtualization of specific resources, e.g., ensure absence of interference on I/O or NVRAM. The access to the system cores is mapped to message-based on-chip communication. At the system cores a modified implementation of the system services layer is required to process those requests from the BSW on the application cores, integrating a NI to interact with the deterministic on-chip-network. The introduction of these system cores in the multicore AUTOSAR platform radically simplifies the AUTOSAR BSW in the μECUs.

The advantage of these dedicated system cores are simpler application cores, since they are tailored to the needs of the actual application only, and the potential of reuse for such system cores in different designs.

The identified system cores are described as follows.

- *Off-chip network gateway core.* This gateway core supports the data exchange between the message-based NoC and the off-chip automotive networks. The gateway core has, in addition to the communication hardware abstraction for the NoC, also software modules and interfaces for accessing off-chip networks such as FlexRay, LIN or CAN. Thus, the communication between the SWCs in different MPSoCs is possible. For instance, Figure 4.1 depicts the realization of the off-chip gateway functionality in a specific system core supporting the CAN communication protocol instead of a communication middleware in each application core.

- *Input/output core.* AUTOSAR adopts a layer model to provide a uniform access to input/output devices, where at the microcontroller abstraction level, basic drivers for analog and digital input/output (e.g., pulse width modulation, capturing of input signals and analog to digital conversion) are provided. An input/output core provides hardware support for the realization of the input/output hardware abstraction services.

- *Memory access core.* The NVRAM management functionality is delegated to a dedicated system core. This system core is connected to a chip-external memory component (e.g., flash memory) on the ECU. The memory access core implements the functionality of the memory hardware abstraction of the ECU abstraction layer. It provides an abstraction from the type and location of the specific memory. A flash or EEPROM memory is accessible via the same interface.

- *Complex drivers core.* AUTOSAR defines the purpose of complex drivers as an interface for complex sensors or actuators, which have stringent requirements on timing and are implemented for specific designs. A complex driver (e.g., for a fast ADC) implemented as a system core highly increases the efficiency of the specific driver functionality.

4.1.3 Fault Hypothesis

In this section we define the fault hypothesis for the TIMEA platform. This fault hypothesis specifies the assumptions about the kinds of failures that system components may experience and, as a consequence, could lead to the failure of the entire system. Based on these failure assumptions, fault tolerance mechanisms are defined. In case a fault occurs, which has not

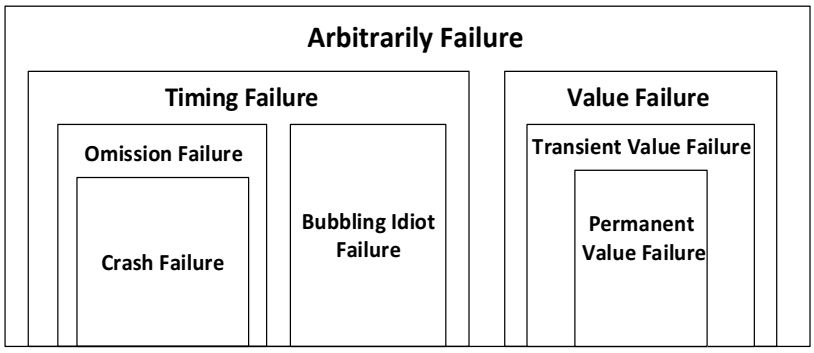

Figure 4.2: Failure Modes

been included in the failure assumptions, the complete system might fail. The fault hypothesis includes the specification of the Fault Containment Regions (FCRs) and the failure mode assumptions [OP07] [UO18].

4.1.3.1 Fault Containment Regions

The FCRs represent isolated units of failures that operate independently from each other and perform correct outputs regardless of any arbitrary logical or electrical fault that may occur in the system outside of their delimited regions. This independence of the FCRs can be compromised if something happens at runtime that was not assumed as part of the fault assumptions (e.g., massive explosion).

We consider AUTOSAR SWCs and AUTOSAR μECUs as FCRs for the AUTOSAR multicore architecture. In each μECU, the AUTOSAR software architecture guarantees a high level of fault independence between SWCs which is assured by the AUTOSAR operating system with memory protection and temporal partitioning (task scheduler). Furthermore, μECUs are identified as independent units of failures due to the employment of the message-based NoC for the communication between each other, which provides fault isolation to each μECU. Thus, two layers of defense can be distinguished: ***(1)*** SWCs as FCRs with lower containment coverage (only addressing software faults) ***(2)*** μECUs as FCRs with higher containment coverage (also addressing hardware faults).

4.1.3.2 Failure Mode Assumptions

Assumptions of the failures are determined independently of the actual cause of the failure (e.g., logical error, hardware error) but from the perspective of the service user and can be classified as a failure in the time domain or in the value domain (see figure 4.2). Based on these assumptions, different failure modes are specified which allow to determine the

degree of redundancy required to provide correct error processing. For the development of the TIMEA platform the following failure assumptions are considered.

1) Time Domain:

- ***Timing Failure.*** A timing failure is a kind of failure where a FCR does not comply with its temporal specification. The FCR outputs are delivered too early or too late. In case no prior knowledge about the message periods (time-triggered messages) or the MINT (rate-constraint messages) is available to the system, the detection of a timing failure is not possible.

  A time-triggered SWC that does not send a message according to its temporal run-time configuration would be an example of a timing failure.

- ***Omission Failure.*** This kind of failure represents a transient failure, where a FCR fails sending a message at its respective time slot and, as a consequence, the receiver FCR does not respond to any input. In contrast to a general timing failure in an omission failure the output is never delivered by the FCR. An example of an omission failure would be a μECU that fails sending a message in its respective time slot through the on-chip network.

- ***Crash Failure.*** In contrast to the omission failures, a crash failure represents a permanent failure that can remain undetected by the system. In case a specific FCR is experiencing a crash failure this FCR would stop producing any outputs. In case a SWC suffers a crash failure this SWC would stop sending messages.

- ***Babbling Idiot Failure.*** This failure represents a special kind of timing failure where a FCR starts sending numerous messages without a minimum time interval between each other, so the communication medium is monopolized by the FCR.

  An example would be an AUTOSAR SWC which constantly sends messages generating delays because of contention in the communication network.

2) Value Domain:

- ***Transient/Permanent Value Failure.*** It represents a kind of failure where the content of the message sent by an FCR is corrupted. In case the failure has a sporadic behavior it is called a transient value failure, otherwise if the failure remains it is called a permanent value failure. In the context of this dissertation we use AUTOSAR message specifications to identify value failures. Thus, a SWC which sends random message values that do not comply with its AUTOSAR specification (e.g., data constraint element, constant specifications) would be an example of a value failure.

4.2 Architecture of an AUTOSAR Micro-ECU

In this section we explain in details the software architecture for an AUTOSAR μECU. Figure 4.3 illustrates the μECU software architecture.

The BSW of the μECUs is extended with new BSW modules in order to support message-based NoC communication. Special communication service modules (COM module, PDU Router) [UO15] serve as the interface between the RTE and the NI for connecting the μECU to the on-chip network. Thus, μECUs are able to use the NoC for the communication between each other.

Additionally, we propose an efficient implementation of the I/O drivers by replacing the original AUTOSAR-defined BSW modules of the I/O abstraction layer by an input/output core (I/O abstraction core in figure 4.3) dedicated to these functionalities. Instead of having an implementation of the whole I/O abstraction layer in the BSW of each AUTOSAR μECU, an I/O proxy module is defined. The I/O proxy serves to forward data prototypes [AUT16d] (sender/receiver interface) or argument prototypes [AUT16d] (client/server interface) sent by the sensor/actuator SWCs to the PDU router and thus, to the input/output core through the message-based NoC. Additionally, ECU signals captured by the input/output core can be received by the μECUs and forwarded to the sensor/actuator SWCs using this I/O proxy module. Moreover, external peripherals requiring stringent time constraints that would need a complex driver realization in the BSW of the μECU can be accelerated by delegating the device driver to a dedicated core. In this case, a proxy module for matching ECU signals of the specific input/output core with the data/argument prototypes of their related sensor/actuator SWCs must be integrated into the BSW of the μECUs that require this service. As an example, Figure 4.3 illustrates a Fast Fourier Transform (FFT) proxy service module which allows the interaction of a μECU with an input/output core dedicated to the FFT functionality. Also, the decision whether a device driver is accelerated by hardware is kept transparent to the RTE and the AUTOSAR application. From now on the FFT example will be used as an accelerated complex driver use case.

Furthermore, a health monitoring module is added to the BSW in the μECUs. This module provides error detection mechanisms based on AUTOSAR pre-defined parameters, e.g., data constraint element and constant specifications, and recovery solutions in case of the failure of a SWC consisting of SWC redundancy in the same μECU or in a different μECU in the MPSoC. Thus, if a sensor/actuator SWC fails, the health monitoring service can activate a replicated SWC located in the same μECU which replaces the failed SWC. If there is no replica available in the same μECU, the health monitoring service makes aware the proxy module to send a notification message to the specific input/output core (e.g., I/O

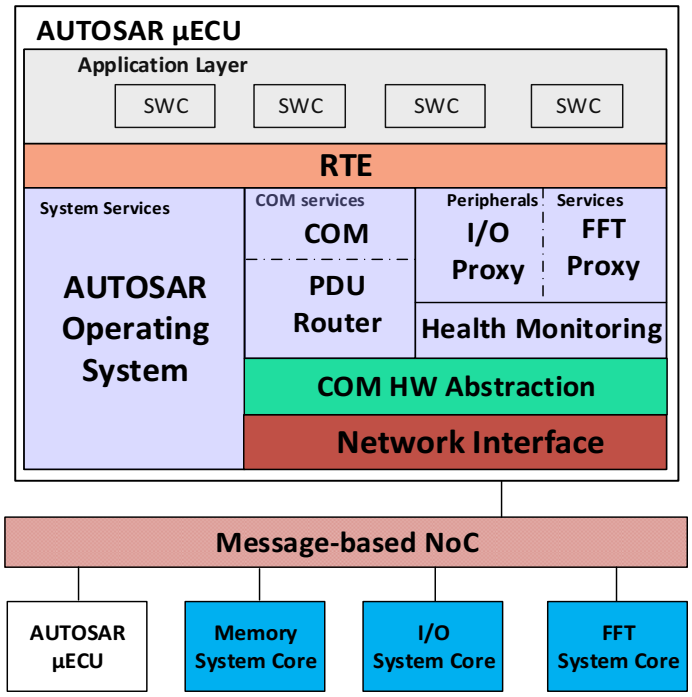

Figure 4.3: AUTOSAR µECU Architecture

abstraction core) in order to indicate that a replica of the failed SWC located on another μECU must take over the access to the I/O functionality. In the scenario depicted in picture 4.3 the I/O system core represents a single point of failure. To avoid this, core redundancy can be also implemented to provide recovery solutions in case of software failures or hardware failures in the system cores. However, the extension of replicated system cores is not within the scope of the thesis and it can be object of study for further investigations.

4.2.1 Communication Stack for NoC support

The AUTOSAR BSW on each μECU is extended to support message-based NoC communication. Figure 4.3 shows the NoC modules added in the AUTOSAR BSW. New communication modules are integrated in the *"ECU communication layers"* for accessing the NoC. A NoC driver module for the MCAL, a NoC hardware abstraction interface and extended COM service modules (COM module and PDU router) are integrated.

As explained in Chapter 2 the RTE is in charge of the inter-SWC communication functionality exposed by the VFB. The RTE provides signals mapped to the variables handled by the SWC ports. For SWC ports hosting a sender-receiver interface, the data transferred by the interface is mapped to a RTE signal. For client-server interfaces, RTE signals are mapped to the function arguments of the SWC ports. The COM module is in charge to

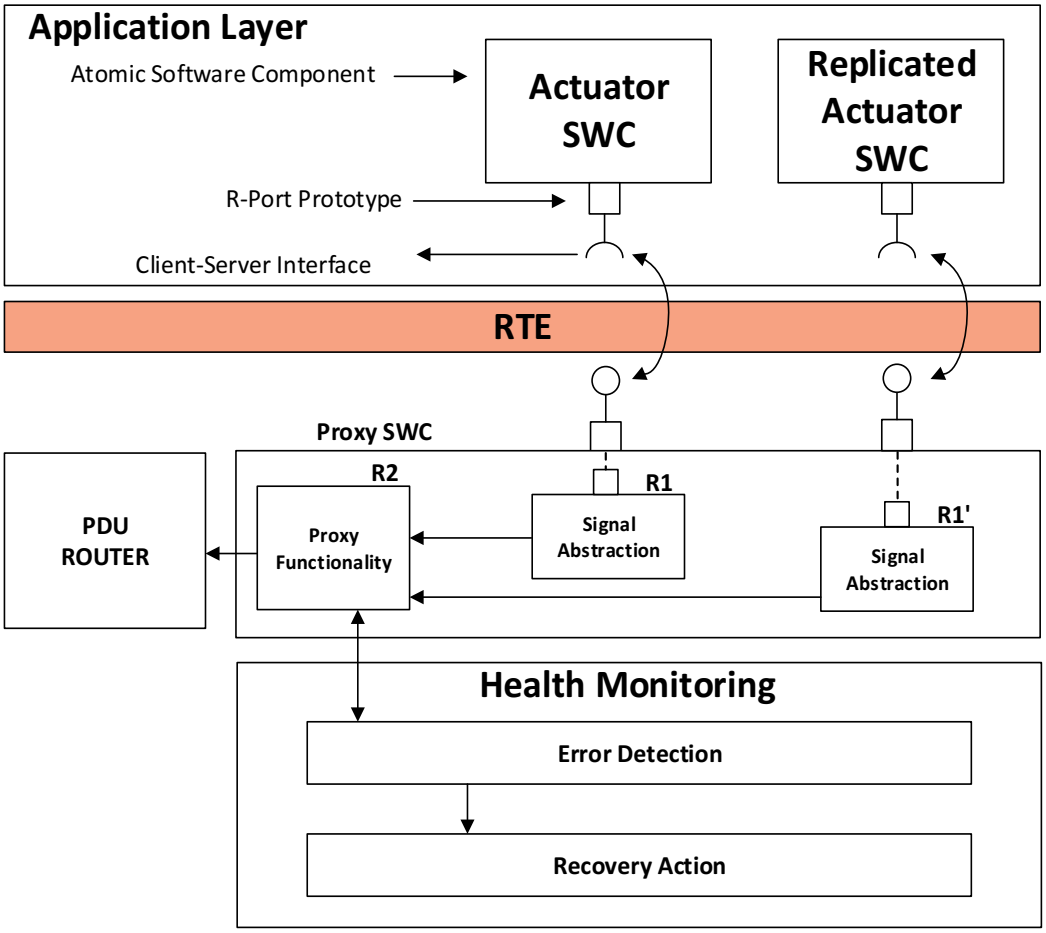

Figure 4.4: BSW Modification

store the RTE signals to outgoing on-chip PDUs and the mapping of these to one of the communication models supported by the NoC (time-triggered messages, rate constrained messages and event-triggered messages). Additionally, the COM module maps incoming on-chip PDUs to the RTE signals. The PDU router contains routing tables to forward NoC PDUs to the COM module and the re-direction of COM PDUs to the NI.

4.2.2 I/O Proxy Functionality

In the AUTOSAR standard [AUT16a] the RTE is in charge of the communication between sensor/actuator SWCs and the BSW modules of the I/O hardware abstraction layer. The I/O abstraction layer consists of BSW SWCs with ports hosting AUTOSAR interfaces (sender/receiver or client/server). Thus, sensor/actuator SWC ports and I/O BSW SWC ports are connected though the RTE layer.

In order to make the application layer and the RTE independent from the acceleration of a device driver, the integration of a proxy module is realized as a BSW SWC, which also implements AUTOSAR interfaces to interact with the application layer. For instance, Figure 4.4 shows an actuator SWC that is connected with a proxy SWC using a client/server interface. In this way, the decision whether a device driver is accelerated or not is kept transparent to the RTE and the application layer.

The internal behavior of the proxy SWC consists of two AUTOSAR runnables. One runnable (R1 in Figure 4.4) is used for the signal abstraction. The aim of this runnable is to abstract in the data handled by the SWC ports from the physical layer values, mapping the proxy SWC ports to ECU signals. Thus, application designers do not have to be aware about implementation details of the device driver APIs and the units of the physical layer values. Thereafter, ECU signals are forwarded to the second runnable of the proxy SWC (R2 in Figure 4.4).

The second runnable matches the ECU signals to PDUs, and invokes the error detection service of the health monitoring module before forwarding the PDU signal to the PDU router. For any value failure recognized by the error detection service the data value is dropped. In case the error detection mechanisms determine that a crash failure or a permanent value failure occurred, a recovery function of the health monitoring service is called. Thus, the two following scenarios are possible:

- In case a replicated actuator SWC depending on the same I/O functionality is located in the same μECU, the proxy function selects this ECU signal (coming from runnable R1' in Figure 4.4) to be packed into the PDU instead of the original one.

- If no redundancy is available on the same μECU, the proxy functionality configures the PDU with a notification message to indicate to the input/output core that the specific μECU is not able any more to control the I/O functionality.

Additionally, the previously defined PDU router is extended with new routing tables to forward PDUs from the proxy module to the NI and vice versa.

4.2.3 Health Monitoring Service

In this section we explain the functionality of the introduced health monitoring module. As mentioned previously, the health monitoring module serves to detect software failures in the time or value domain and to enable recovery solutions in the application software. This module consists of an error detection block and a list of recovery actions that cover the kinds of failures described by the fault hypothesis (see Figure 4.4).

In the TIMEA architecture, we use spare SWCs which are activated upon the detection of a fault. This means, safety-critical SWCs are developed with multiple implementations that can be activated in case the pre-set SWC fails. For this, callback functions are used to resume (at run time) operating system tasks that host the replicated SWC. As defined by AUTOSAR [AUT16c], callback functions provide the capability to trigger or stop SWCs that are outside of the AUTOSAR BSW. Thus, a replica of the SWC does not increase the operating system overhead, since the tasks are set up in suspended state [AUT16c] as defined at compile time.

Omission and crash failure recognition is processed by the error detection block using algorithm 1, which defines the crash failure processing procedure. Moreover, for the handling of transient and permanent value failures the value failure processing procedure is presented in algorithm 2. Also, algorithm 1 defines the procedure for activating replicas, which is used by both algorithms for the exploitation of the SWC redundancy.

The following sets and entities are used to explain the health monitoring algorithm:

- $RTEswc_{ij}$: This represents the obtained status of a SWC data/argument prototype returned by its specific RTE function, with $i \in [1,I]$ and $j \in [1,J]$, where I is the total number of AUTOSAR SWCs in the μECU, i is the ID of the requester SWC, J represents the total number of data/argument prototypes handled by the SWC and j is the data/argument prototype ID. This returned RTE function status means: ***(1)*** $RTE\_E\_OK =$ the job processing finished correctly, ***(2)*** $RTE\_NOT\_OK =$ the job processing finished with error.

- $Dswc_{ij}$: It represents a data/argument prototype of a specific SWC. Each $Dswc_{ij}$ is associated with a data constraint element $DC_{ij} \in [DCij_{min}, DCij_{max}]$ and has a specific unit $Unit_{ij}$, both set up by the data/argument prototype AUTOSAR specification.

- ρ_i: This threshold parameter represents the maximum number of consecutive omission failures of a single SWC that can be tolerated by the μECU.

- ϕ_i: This threshold parameter represents the maximum number of consecutive value failures of a single SWC that can be tolerated by the μECU.

- φ_i: This threshold parameter represents the maximum number of omission failures of a single SWC that can be tolerated by the μECU within an interval of κ_i executions.

- γ_i: This threshold parameter represents the maximum number of value failures of a single SWC that can be tolerated by the μECU within an interval of c_i executions.

- N_i: It represents the actual number of consecutive omission failures of a single SWC.

- R_i: It represents the actual number of consecutive value failures of a single SWC.
- n_i: It represents the actual number of omission failures of a single SWC within an interval of κ_i executions.
- r_i: It represents the actual number of value failures of a single SWC within an interval of c_i executions.
- $SDswc_{ij}$: This variable serves to indicate whether the data/argument prototype complies with its AUTOSAR specification.
- G_i: It represents the actual grade of redundancy provided for the specific safety-critical SWC_i. This means, the number of SWC replicas available on the μECU.
- $Rswc_{iz}$: It represents the SWC replica of a safety-critical SWC_i, with $z \in [0, G]$.
- ξ_i: It represents the actual number of executions of a single SWC within an interval of κ_i executions.
- χ_i: It represents the actual number of executions of a single SWC within an interval of c_i executions.

The following expressions are used to determine the presence of a crash failure occurrence and a permanent value failure occurrence.

$$\begin{cases} N_i \geqslant \rho_i + 1 & \text{Crash failure detected} \\ N_i < \rho_i + 1 & \text{No crash failure present} \end{cases} \tag{4.1}$$

Permanent Omission Failures

$$\begin{cases} n_i \geqslant \varphi_i + 1 & \text{Crash failure detected} \\ n_i < \varphi_i + 1 & \text{No crash failure present} \end{cases} \tag{4.2}$$

Multiples Omission Failures within κ_i executions

$$\begin{cases} R_i \geqslant \phi_i + 1 & \text{Permanent value failure detected} \\ R_i < \phi_i + 1 & \text{No permanent value failure} \end{cases} \tag{4.3}$$

Permanent Value Failures

$$\begin{cases} r_i \geqslant \gamma_i + 1 \text{ Pemanent value failure detected} \\ r_i < \gamma_i + 1 \text{ No permanent value failure} \end{cases} \quad (4.4)$$

Multiple Value Failures within c_i executions

Algorithm 1 HM Algorithms - Crash Failure Processing

```
1:  procedure CrashFailureRecognition(RTEswc_ij)
2:      ξ_i = ξ_i + 1
3:      if RTEswc_ij == True then
4:          N_i = 0
5:          if ξ_i == κ_i then
6:              ξ_i = 0
7:              n_i = 0
8:          end if
9:          ValueFailureRecognition(Dswc_ij)
10:     else
11:         N_i = N_i + 1
12:         n_i = n_i + 1
13:         if N_i ⩾ ρ_i + 1 || n_i ⩾ φ_i + 1 then
14:             if G_i > 0 then
15:                 ActivateReplica(Rswc_iz)
16:                 ProxyIndication
17:                 G_i = G_i − 1
18:             else
19:                 ProxyIndication
20:             end if
21:         else
22:             if ξ_i == κ_i then
23:                 ξ_i = 0
24:                 n_i = 0
25:             end if
26:         end if
27:     end if
28: end procedure
29: procedure ActivateReplica(Rswc_iz)
30:     RTE_SWC_iz(OFF)
31:     RTE_Rswc_iz(ON)
32: end procedure
```

Once a safety critical SWC runnable is triggered by the AUTOSAR operating system, the status of its specific RTE function for the internal communication is observed by the integrated

health monitoring functionality. This functionality uses the algorithm 1 for monitoring the status value of the RTE function.

As explained earlier, algorithm 1 is responsible for the recognition and processing of crash failures. The procedure "$CrashFailureRecognition(RTEswc_{ij})$" is called for each safety-critical SWC triggered by the operating system. This procedure distinguishes between two kinds of crash failure possibilities: ***(1)*** Reaching a pre-defined threshold of consecutive omission failures ρ_i tolerated by the μECU, ***(2)*** Reaching a pre-defined threshold of total omission failures φ_i tolerated by the μECU within an interval of κ_i executions.

The algorithm starts updating the ξ_i counter since a new SWC execution was performed. This counter is checked every single execution and serves to keep the n_i counter updated within a delimited interval of κ_i executions. In case ξ_i reaches κ_i and no crash failure was recognized, this counter, as well as the n_i counter, are set to 0.

The passed RTE function status $RTEswc_{ij}$ is checked for the recognition of omission failure occurrences. In case $RTEswc_{ij}$ is $True$, the N_i counter is set to 0 since no consecutive omission failure was detected and the "$ValueFailureRecognition(Dswc_{ij})$" procedure of algorithm 2 is called for value failure processing. This procedure will be explained later for algorithm 2.

In case the $RTEswc_{ij}$ is $False$, this means that an omission failure occurred and therefore the N_i and n_i counters are increased. After this, the expression 4.1 serves for determining the existence of a crash failure due to consecutive omission failures and, based on this assumption, the "$ActivateReplica(Rswc_i)$" procedure is called if a crash failure was detected and another SWC is available. In case a crash failure was detected and no SWC replica is available, the proxy module is notified. The "$ActivateReplica(Rswc_i)$" procedure is detailed later.

If no crash failure was determined by the expression 4.1, then expression 4.2 is used for determining the existence of a crash failure due to multiple omission failures within an interval of κ_i executions. At this moment, in case no crash failure was detected, the value failure recognition is initiated and the "$ValueFailureRecognition(Dswc_{ij})$" procedure of algorithm 2 is called.

The algorithm 1 also defines the "$ActivateReplica(Rswc_i)$ procedure. This procedure implements callback functions to resume the tasks running $Rswc_i$ and pause the tasks hosting the original SWC_i. Callback functions are implemented according to the $Rte\_Call\_<p>\_<o>$ API [AUT16i] of the RTE in order to enable safe configuration of the AUTOSAR services.

The algorithm 2 is in charge of the processing of the value failures. It defines the $ValueFailureRecognition(Dswc_{ij})$ procedure, which serves to distinguish a permanent value

Algorithm 2 HM Algorithms - Permanent Value Failure Processing

1: **procedure** $ValueFailureRecognition(Dswc_{ij})$
2: $\chi_i = \chi_i + 1$
3: **if** $Dswc_{ij} < DCij_{min} \parallel Dswc_{ij} > DCij_{max} \parallel Dswc_{ij}.Unit \neq [Unit_{ij}]$ **then**
4: $SDswc_{ij} = False$
5: **else**
6: $SDswc_{ij} = True$
7: **end if**
8: **if** $SDswc_{ij} == False$ **then**
9: $ProxyIndication$
10: $R_i = R_i + 1$
11: $r_i = r_i + 1$
12: **if** $R_i \geqslant \phi_i + 1 \parallel r_i \geqslant \gamma_i + 1$ **then**
13: **if** $G_i > 0$ **then**
14: $ActivateReplica(Rswc_{iz})$
15: $ProxyIndication$
16: $G_i = G_i - 1$
17: **else**
18: $ProxyIndication$
19: **end if**
20: **else**
21: **if** $\chi_i == c_i$ **then**
22: $\chi_i = 0$
23: $r_i = 0$
24: **end if**
25: **end if**
26: **else**
27: $R_i = 0$
28: **if** $\chi_i == c_i$ **then**
29: $\chi_i = 0$
30: $r_i = 0$
31: **end if**
32: **end if**
33: **end procedure**

failure from a transient value failure. Alike the crash failure processing, permanent value failures are differentiated in two types: ***(1)*** Reaching a pre-defined threshold of consecutive value failures ϕ_i tolerated by the μECU, ***(2)*** Reaching a pre-defined threshold of total value failures γ_i tolerated by the μECU within an interval of c_i executions.

The permanent value failure recognition starts by updating the χ_i counter. Comparable to the "$CrashFailureRecognition(RTEswc_{ij})$" procedure, the χ_i counter is checked every single execution to keep the r_i counter updated within a delimited interval of c_i executions. In case χ_i reaches c_i and no permanent value failure was recognized, this counter, as well as the r_i counter, are set to 0. For any value failure recognition a proxy module indication function is called, so the data value can be dropped.

The passed data/argument prototype $Dswc_{ij}$ is checked for the recognition of a transient value failure. For this, the data constraint element and the data unit linked to this data/argument prototype are used. Since the data/argument prototype must be bounded by its data constraint element, it is compared with the maximum and minimum values of the constraint element. Additionally, the data/argument prototype unit is compared with its original specification. In case $Dswc_{ij}$ conforms to its AUTOSAR specification, $SDswc_{ij}$ is set to $True$ (no value failure), otherwise it is set to $False$. If a transient value failure occurred, the R_i and r_i counters are increased and expressions 4.3 and 4.4 are used to detect the presence of a permanent value failure. When a permanent value failure is recognized, the "$ActivateReplica(Rswc_i)$ procedure is called to execute the replica $Rswc_i$ and to turn off the original SWC in case SWC redundancy is available, otherwise the proxy module is notified that no recovery action was triggered.

4.3 Architecture of the I/O Gateway Core

Input/output cores replace the functions normally provided by the I/O BSW abstraction layer [AUT16a] and the complex driver abstraction layer of AUTOSAR (see Figure 4.5). The input/output cores support bidirectional communication via the NI that connects the core to the on-chip network. Additionally, a virtualization layer abstracts the I/O functionalities from their underlying AUTOSAR μECUs and sensor/actuator SWCs that control the I/O ports. This layer uses the input port and output port provided by the corresponding NI for the exchange of data with the network supporting fault isolation. ECU signals handled by the I/O ports (e.g., ADC converter) are mapped to time-triggered messages as part of the pre-configured time schedule of the network. The virtualization layer injects input signals via time-triggered buffers with a pre-defined time slot in order to be sent by the NI through the NoC. Additionally, each message received from the NI by the virtualization layer is matched

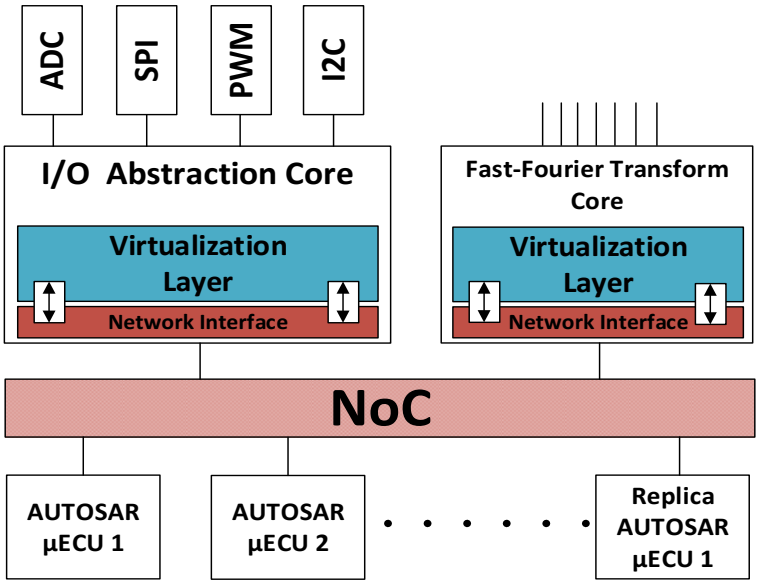

Figure 4.5: Efficient AUTOSAR Multicore Platform based on I/O Gateway Cores

to an output signal of a specific driver device, for example an output pin connected to a light switch.

The virtualization layer allows the use of redundancy for sensor/actuator SWCs at the MPSoC level. The algorithm depicted in Figure 4.6 is implemented by the virtualization layer for the detection of hardware crash failures and software permanent failures on the μECUs. The following entities serve to explain the error detection mechanism:

- ∂_i = This threshold parameter represents the maximum number of consecutive hardware omission failures of a single μECU that can be tolerated by the system core, with $i \in [1, I]$, where I is the total number of μECUs using the system core and i is the ID of the requester μECU.
- ς_i = This threshold parameter represents the maximum number of hardware omission failures of a single μECU that can be tolerated by the system core within an interval of τ_i executions.
- U_i = It represents the actual number of consecutive omission failures of a single μECU.
- u_i = It represents the actual number of omission failures of a single μECU within an interval of τ_i executions.
- G_i: It represents the actual grade of redundancy provided for the specific μECU_i. This means, the number of μECU replicas available on the MPSoC.
- $R\mu ecu_{iz}$: It represents the μECU replica of a safety-critical μECU_i, with $z \in [0, G_i]$.

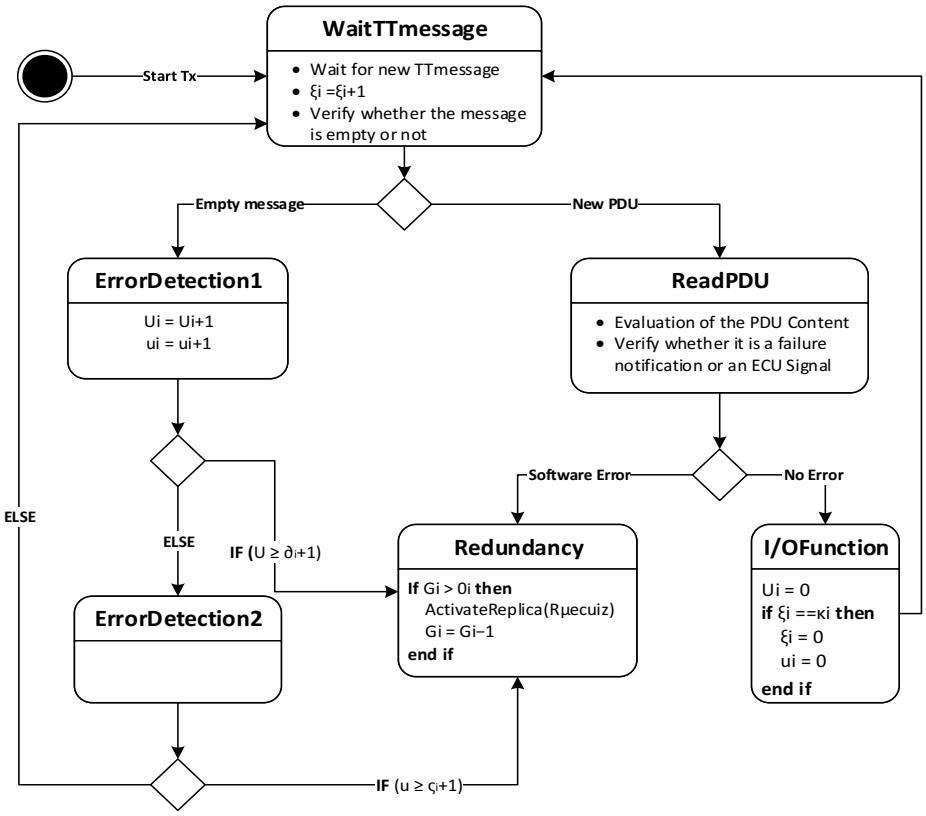

Figure 4.6: Error Detection Algorithm employed by the Virtualization Layer in the I/O Gateway Core

- ξ_i: It represents the actual number of executions of a single μECU within an interval of τ_i executions.

The detection of a hardware crash failure is determined by the following expressions:

$$\begin{cases} U \geqslant \partial_i + 1 & \text{Hardware crash failure detected} \\ U < \partial_i + 1 & \text{No hardware crash failure present} \end{cases} \tag{4.5}$$

Permanent Hardware Omission Failures

$$\begin{cases} u \geqslant \varsigma_i + 1 & \text{Hardware crash failure detected} \\ u < \varsigma_i + 1 & \text{No hardware crash failure present} \end{cases} \tag{4.6}$$

Multiples Hardware Omission Failures within τ_i executions

A no received message at its pre-scheduled point in time is perceived by the virtualization layer as a hardware omission failure. In case a hardware crash failure is detected by the

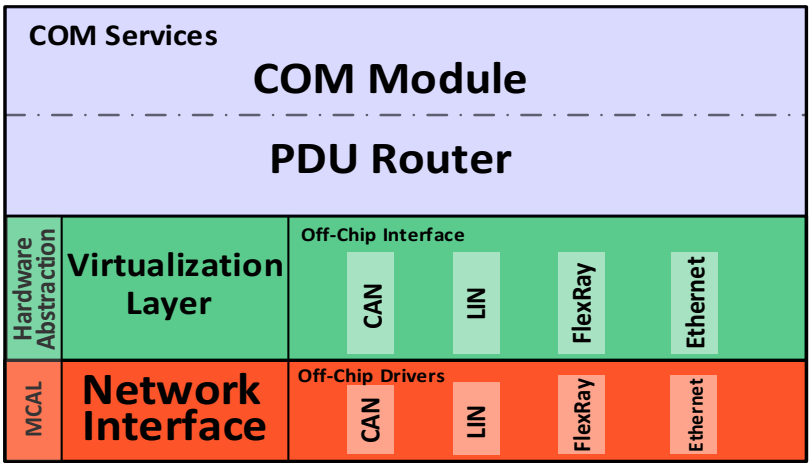

Figure 4.7: Architecture of the Off-Chip Network Gateway Core

virtualization layer using the expressions 4.5 and 4.6 (e.g., the μECU1 in Figure 4.5), this layer provides access to another μECU hosting a replica of the sensor/actuator SWC which takes over interacting with the specific I/O functionality (Replica of μECU 1 in Figure 4.5). Likewise, if a μECU sends a notification message indicating a software permanent failure of a sensor/actuator SWC (explained previously), the μECU redundancy is exploited, otherwise the specific ECU signal is forwarded to the specific I/O functionality.

Additionally, in case multiple μECUs need to access the same service provided by an input/output core dedicated to a complex driver (for example the FFT core of Figure 4.5), the virtualization layer also supports the sharing of the I/O core among different μECUs based on static priorities defined at compile time according to ISO26262 [Int11]. For instance, if μECU 1 and μECU 2 require to trigger a FFT of an ECU signal during a certain time period, based on the criticality assigned to the each μECU, the virtualization layer will forward their requests to the FFT service in a specific order. Hence, the FFT of the μECU signal with the highest criticality finishes first.

4.4 Architecture of the Off-Chip Network Gateway Core

The off-chip gateway system core provides support for off-chip communication to the μECUs in the TIMEA platform. In Figure 4.7 the software architecture of the presented off-chip network gateway core is depicted. Three layers compose the gateway core architecture: a communication service layer, a hardware-abstraction layer and the MCAL for the communication drivers.

As shown Figure 4.7, the gateway core requires an implementation of the NI at the MCAL level in order to be able to access the message-based NoC for on-chip communica-

tion. Besides the NI, communication hardware-abstraction modules for supporting off-chip communication are also integrated. Depending on the off-chip communication network to be supported by the gateway, specific AUTOSAR BSW hardware-abstraction modules of the specific communication bus are selected for the implementation.

Additionally, a virtualization layer is integrated on the hardware abstraction layer, so SWC redundancy on the different μECUs can be exploited by the off-chip network gateway core.

4.4.1 COM module

In the AUTOSAR architecture the COM module [AUT16e] works as an interface between the RTE and the PDU router. Its main function is the mapping of the RTE signals to PDUs and vice versa. Moreover, The implementation of the COM module is independent of the communication protocol used by the ECU for off-chip communication.

For the off-chip gateway core, the COM module is in charge of receiving PDUs from the PDU router and to make use of a look up table to assign incoming NoC PDUs with off-chip PDUs and vice versa. The format of the PDUs sent through the off-chip network depends of the implemented communication protocol as explained in [AUT16e], while for NoC PDUs a format description is presented later.

As mentioned previously, a message-based NoC supports bandwidths of several *Gbps* with communication latencies and jitter in the range of *ns*. Since the bit rate of off-chip networks is typically lower by several orders of magnitude in comparison to a message-based NoC, the COM module is also used to change the sending time and cycle time for providing rate conversion support to the gateway. This is done by using the transfer property of the PDUs. This property defines whether the PDU should be transmitted whenever its message content changes or if the PDU is sent periodically with the last message stored to it.

4.4.2 PDU Router

A typical PDU router implements routing tables and a router engine for routing and transferring PDUs from the COM module to the communication hardware-abstraction which transforms them into an actual message that is then forwarded to the MCAL in order to be sent over the communication hardware entity provided by the ECU. As defined in [AUT16h], the purpose of the PDU router is to statically route PDUs based on their PDU identifier and to avoid any dynamic routing at run-time. Just as the COM module, the PDU router has the same design independent of the off-chip communication protocol.

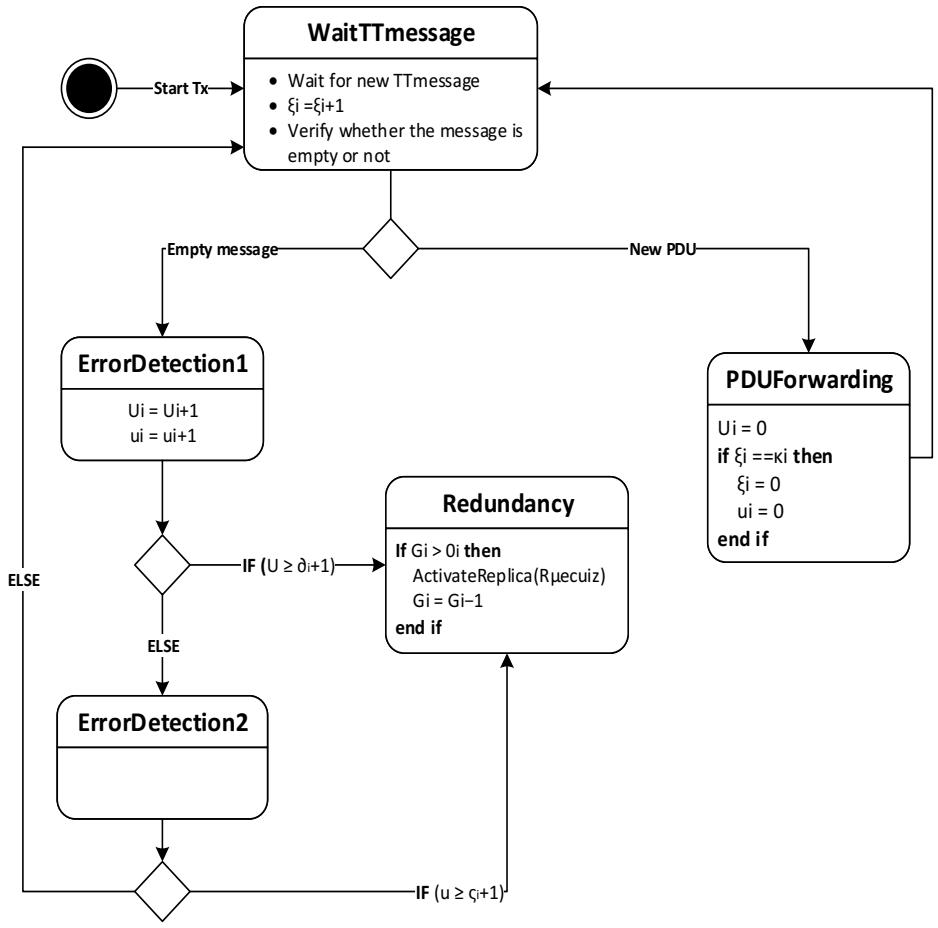

Figure 4.8: Error Detection Algorithm employed by the Virtualization Layer in the Off Chip Network Gateway Core

In the off-chip gateway core the PDU router is extended to support the routing of PDUs coming from the message-based NoC. With this purpose, it contains extended routing tables to forward not just PDUs between the COM module and the off-chip interface in the gateway but also for NoC PDUs between the COM module and the virtualization layer.

4.4.3 Virtualization Layer for Off-Chip Network Gateway Core

The virtualization in the off-chip gateway core allows the TIMEA platform to provide SWC redundancy on the MPSoC level for SWCs that need to interact with SWCs located on a different MPSoC. Thus, if a safety-critical μECU using the off-chip network presents a hardware failure and therefore fails forwarding its functionality, the off-chip network gateway core may pick a different μECU input to maintain providing the correct MPSoC function on a distributed system.

The state machine illustrated in Figure 4.8 performs the error detection employed by the virtualization to recognize μECU failures. The entities defined in Section 4.3 for the virtualization layer in the I/O gateway core are used also here for the error recognition.

In case a safety-critical μECU fails sending its time-triggered message on time, expressions 4.5 and 4.6 are used to differentiate between faults based on the hardware crash failure assumption. Moreover, if a hardware crash failure was detected a different time-triggered message from the μECU redundancy is selected by the virtualization layer to be forwarded to the PDU router.

4.5 Architecture of the Memory Gateway Core

As mentioned previously, a system core serving as a memory gateway allows the μECUs to access an external memory using the message-based NoC. This gateway core is provided with a NI so it can access the message-based NoC to receive memory transactions and to send memory replies to the μECUs. An architecture picture of the memory gateway is depicted in Figure 4.9, which illustrates an example of a distributed system consisting of multiple networked TIMEA platforms. As shown in this figure, each MPSoC is provided with its own memory gateway and off-chip network core.

The memory gateway core consists of a Distributed Mixed-criticality Transactional Controller (DMTC) algorithm [OUOA16] and a basic memory controller. Memory transactions are first processed by the DMTC based on their criticalities according to ISO26262 and thereafter they are queued in the basic memory controller to be executed. Additionally, memory replies are forwarded from the external memory to the receptive μECU through the on-chip network using the NI.

4.5.1 Distributed Mixed-Criticality Transaction Controller

The DMTC encompasses a version container, a global page table and a set of locally stored memory pages (see Figure 4.9). It is responsible to manage address versioning and memory page exchanges between different MPSoCs and μECUs.

The version container is in charge of the tracking and handling of the address versioning of all addresses that are locally stored in the MPSoC. At the other hand, the global page table serves to locate required pages within a MPSoC of a distributed system. This global page table is globally shared and synchronized by all MPSoCs connected to the distributed system based on the requests for memory pages.

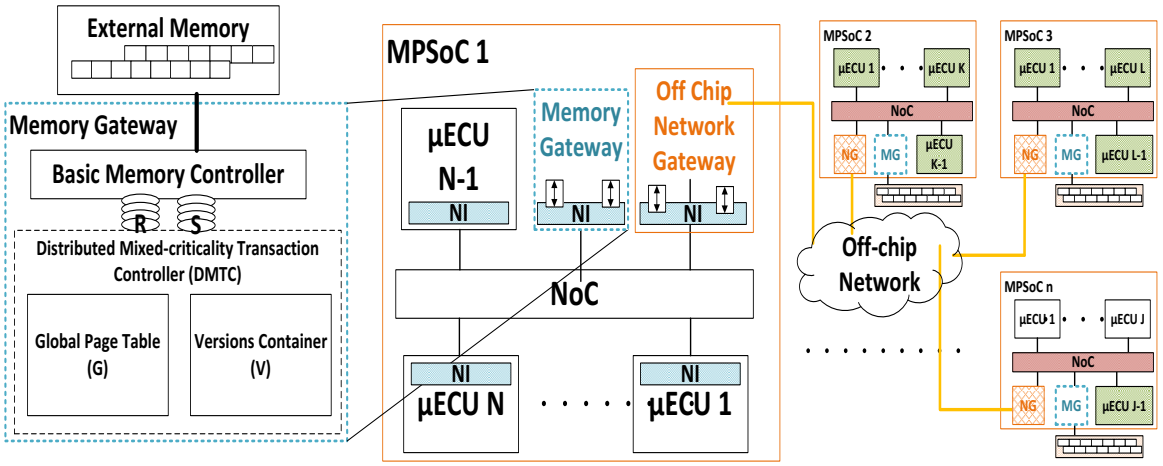

Figure 4.9: Memory Gateway Architecture

Additionally, the algorithm presented by [Owd16] uses the version container and the global page table to execute criticality-aware conflict resolution. The DMTC allows the processing of the following 3 different cases for the executing of memory operations and their required memory pages:

- *Locally at the µECU.* In case the required memory page is in the same requester µECU this page can directly be accessed.

- *Locally at the MPSoC.* If the required memory page is in a different µECU of the same MPSoC the DMTC locates the page and the page ownership is given to the requester µECU. Therefore the corresponding updates are performed to the local global page table and version container.

- *Remotely at another µECU.* In case the required memory page is located at a µECU of a different MPSoC, this page is located in the global page table of the remote MPSoC and moved to the requester µECU, which takes the ownership of the page. Additionally, relevant records of the page are moved from the remote version container to the local version container of the requester MPSoC and the local global page table is updated.

The detailed explanation of the criticality-aware conflict resolution functionality and the description of its algorithms are very well discussed on [OUOA16] and [Owd16] as part of the resulting work presented on a different dissertation.

4.5.2 Basic Memory Controller

In the memory gateway core we assume a compositional real-time memory controller such as [AGR07], which provides support for analytical design-time verification of hard real-time requirements. This memory controller consists on a two-step approach: ***(1)*** starting by defining the memory access groups with known efficiency and latency ***(2)*** a predictable credit-controlled static-priority arbiter [AGR07] is responsible for scheduling these groups dynamically. The dynamic scheduling guarantees the allocated bandwidth and the maximum latency bounds of the memory interactions.

4.6 Fault Tolerance Mechanisms

The TIMEA platform provides a set of fault tolerance mechanisms comprising recovery solutions to counteract the impact of the pre-defined failure modes. Additionally, a single-fault hypothesis is selected, which means the maximum number of failures is 1. The maximum number of failures represents the maximum number of FCR failures that the system may handle at the same time. A single failure occurrence is the prevalent assumption in many present-day safety-critical systems [OP07] (e.g., TTA, FlexRay [Fle05]).

Since a message-based NoC is used for the communication between the cores, communication delays due to the occurrence of a babbling idiot failure can be avoided in safety-critical SWCs. These SWCs use time-triggered messages with pre-defined network scheduling, thus quarantining the core faults since they are isolated by the communication network. Thus, in case a SWC starts sending unlimited messages to the network no delays are perceived at other μECUs.

In case of omission/crash failures, the defined monitoring service allows the TIMEA platform to provide recovery actions based on SWC redundancy on the μECU level. Thus, supposing a safety-critical SWC crashes, the error detection algorithm would determine the presence of a crash failure and the replica of the SWC would be executed. Likewise, for hardware crash failures, as for example a safety-critical AUTOSAR μECU that stops completely its operation, SWC redundancy at the MPSoC level serves to provide a recovery solution to the multicore system.

For timing failures of the AUTOSAR software, we expect time-triggered messages in the NoC to not be affected. This means, in the scenario that a safety-critical SWC does not obey its temporal specification and delays the sending of a message, the resulting communication jitter would not be affected due to the intrinsic fault isolation property of time-triggered messages.

In the same way that the omission/crash failures are prevented, the monitoring service avoids the propagation of value failures through the NoC. For example, in case a safety-critical SWC starts sending messages whose values are not complying with its data constraint element specification, the monitoring service would recognize the fault and use SWC redundancy on the μECU to avoid the persistence of the failure.

The choice of the defined threshold parameters used by the health monitoring service for failure recognition depends on the specific design and implementation use case.

It is important to highlight that in TIMEA the described message-based NoC represents a single point of failure, where failure occurrences are not covered by the presented health monitoring and redundancy methods. For the recovery of these failures other kind of methods may be employed (for example using redundant communication typologies, i.e., double start topology) which are not object of study within the scope of this dissertation. Additionally, other parts of the AUTOSAR software beside the SWCs in the application layer could exhibit failure occurrence. For instance, the defined proxy module could fail propagating the PDU ECU signals to the PDU router. This scenario impedes a two fault hypothesis statement since redundancy in the MPSoC level would serve as the only available recovery action.

Chapter 5

Simulation Framework for Message-based AUTOSAR MPSoC Platforms

Simulation environments are important for the development of automotive applications, helping system architects in exploring different design decisions. Simulation systems enable the early examination of applications on MPSoC platforms, providing environments for performance evaluation before implementing the physical system. In the automotive industry simulation environments play a very important role in the development process of embedded systems since they also provide a framework for reliability examination of the platforms under fault occurrences using fault injection.

However, the simulation of time-triggered message-based multicore processors hosting AUTOSAR-based software and the application behavior for virtual validation scenarios are still missing. There are no full-system simulation tools that support both AUTOSAR-based software and on-chip network simulation. Such a simulation framework is required to quantify in an early stage the advantages obtained by the integration of the AUTOSAR architecture with a message-based NoC and to evaluate the impact of the hardware choices (e.g., multicore topology, NoC configurations) on the AUTOSAR software.

In this chapter a novel simulation framework for AUTOSAR applications running on a multicore platform with NoCs is designed and implemented with the purpose to serve the TIMEA architecture presented in Chapter 4. In the following, available MPSoC simulation tools are compared with respect to their abstraction level, the support for network on-chip and off-chip simulation and the ability for AUTOSAR software execution.

Several simulation tools support multicore processors, software architectures, on/off-chip networks and memory system simulations. For example, OPNET [HGY07] and OMNET++

[Var99] are off-chip network simulation tools that are used for designing and validating networked systems. Furthermore, OPNET provides an interface that is compatible with other simulation tools (e.g., Simulink).

The On-Chip Communication Network (OCCN) [CCG+04] proposes a simulation environment for modeling and simulation of on-chip communication architectures. Additionally, cycle-accurate simulation tools such as SystemC and GEM5 are widely used for on-chip network simulations.

On-chip simulation is supported by full system simulators such as Simics [MCE+02] and Sniper [CHE11]. Also, SystemC and GEM5 support the simulation of system-level architectures.

Simulation tools such as VEOS (dSpace) and CANoe (Vector) provide simulation environments for experimental tests specifically in the automotive domain. These simulation environments support both application simulation as well as simulation of off-chip communication.

However the combination of these simulation tools represents a major scientific challenge. The definition of interfaces for data exchange and synchronization between the simulation tools is required. In the last years, several co-simulation environments have been presented in research and industry communities for the co-simulation of off-chip network simulations, on-chip network simulations, and application functionality simulations.

For co-simulation focusing on the communication behavior of the systems, the co-simulation can be abstracted from the computational functionality of the system. In [ZEK+13] a time-triggered cyber-physical system simulation is presented performing a co-simulation of SystemC with a hardware-in-the-loop automotive simulation tool. In [MRR08], ONNET++ and SystemC are combined in a co-simulation system for the simulation of distributed systems.

On the other hand, for performing full-system co-simulations, virtual platforms are required to support hardware virtualization (e.g., OPVSim, QEMU). In [FMZ+12] the so-called transformer is presented. This is a full-system simulator for multicore simulation based on QEMU. In [CYT11], [NHT+12], [WG14] QEMU and SystemC are combined for the emulation of the CPUs and the communication behavior of a multicore NoC respectively. A co-simulation framework between the OVP virtual platform and QEMU is presented in [CLP14]. This co-simulation framework implements a systemC-based interface for integrating systemC components into the overall simulated system. Moreover, in [BKM13] a design methodology for the integration of heterogeneous SWCs and multicore architectures is presented. The so-called HeroeS is mapped on a SystemC virtual platform framework for

the simulation of embedded real-time architectures, and was evaluated by its integration into an AUTOSAR environment.

In [PHO+14] a GEM5 full system emulation and a CPU-GPU simulator are combined. This GEM5-GPU simulator implements a shared memory interface for the information exchange between the two simulation tools.

Based on the latter discussion, the simulation of AUTOSAR-based systems with on-chip networks for multicore communication is not supported by any existing co-simulation environment. The commercial AUTOSAR multicore tools that are available on the market implement a shared memory approach for multicore simulations (e.g., VECTOR tools [VEC16], ETAS tools [ETA16]).

The presented co-simulation framework [UAO16] supports the simulation of time-triggered message-based multicore processors hosting AUTOSAR-based software and can be easily adapted and implemented based on existing AUTOSAR simulators and on-chip simulation tools. We present the coordination of three different simulation levels: the AUTOSAR software, the physical environment and the NoC behavior. We propose a communication interface for the synchronization and data exchange between the simulation systems. Local simulation coordinators are defined, which determine when a co-simulation step can be performed. Additionally, the extension of AUTOSAR BSW simulation modules for supporting NoC communication is presented.

Furthermore, the implementation of the proposed simulation framework is carried out as part of the work presented in [UOO15]. The AUTOSAR-based system simulator VEOS is used for the simulation at the on-chip application level and two different NoC simulators, GEM5 and SystemC, serve for the simulation at the on-chip communication level.

5.1 Concept of the Co-simulation Framework

The proposed simulation framework consists of one AUTOSAR-based system simulator hosting the simulation of the AUTOSAR μECUs and the physical environment simulation, one on-chip simulator hosting the NoC simulation, and the coordination of the simulation tools.

In the rest of this section we present the simulation model for the message-based NoC, as well as the simulation models for the AUTOSAR μECUs and the physical environment. Additionally, the co-simulation and coupling of the simulation tools is presented. We introduce a socket-based coordination of simulation tools, which uses TCP/IP and integrated local coordinators for interfacing and synchronizing the simulation tools.

5.1.1 Simulation Model of Network-on-Chip

The presented simulation framework gives support for the different timing models described in Chapter 4 (i.e., best-effort, time-triggered and rate-constrained) in a message-based multicore processor running AUTOSAR software.

The simulated MPSoC interconnects several simulated cores supporting different topologies (e.g., Mesh, Spidergon, etc) using the NIs. The interface between the application, running on the core, and the NoC is realized by configurable *Ports*. Based on the configuration, each port contains a FIFO (for event messages) or a buffer with update-in-place semantics (for state messages) to store messages.

Ports are used at the sender core as well as at the destination cores. At the sender core, the NI dequeues the port at the instant, which is determined by the schedule, the rate-constraints and the priority. At the destination cores, ports keep the arrived messages until the core dequeues them. Alternatively, an interrupt can be used to notify the application, once the message arrives at the port.

5.1.2 Environment Simulation

Environment models act as simulation models representing the natural environment of an MPSoC in a virtual validation scenario. These environment models emulate the physical process resulting from the interaction of the μECUs with the physical environment. They are controlled by the AUTOSAR-based system simulator and can be integrated as discrete simulation models as well as continuous simulation models into the AUTOSAR simulation.

5.1.3 Simulation Model of an AUTOSAR Micro-ECU

As explained in Chapter 4, TIMEA describes application cores which play the role of AUTOSAR μECUs using a message-based NoC for the communication between each other. To achieve this, existing simulation models of the AUTOSAR ECUs are extended with simulation building blocks for the simulation of the message-based NoC communication.

The μECUs host the AUTOSAR SWCs and each one of them is provided with a RTE and BSW. In the proposed framework, μECUs are configured as depicted in Figure 5.1 and simulated by the AUTOSAR system simulator.

Since the AUTOSAR standard does not provide support for NoC communication, we extend the BSW on the simulated μECUs with special BSW modules to support the communication through the NoC. An NoC interface module for the hardware abstraction layer is integrated for accessing the NoC. Extended COM service modules are integrated for

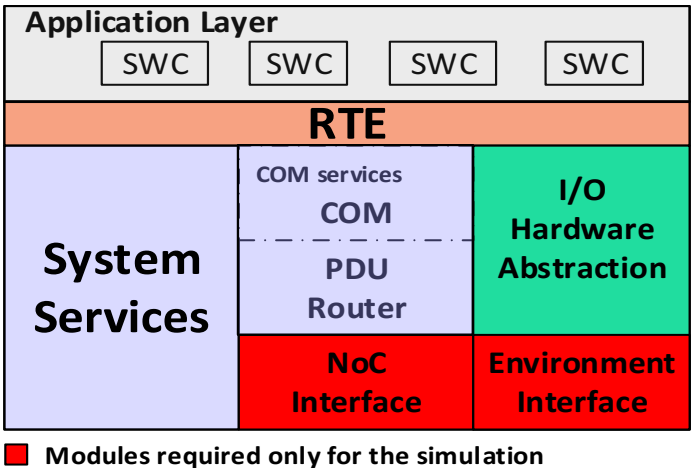

Figure 5.1: Architecture of a simulated AUTOSAR Micro-ECU

routing the messages from the NoC to the application software and vice versa. The integrated NoC interface module allows the access from the BSW of the μECU to the NoC simulation. This module is in charge of the exchange of data between the BSW on the μECU and its corresponding NI in the NoC simulation.

Moreover, as defined by the AUTOSAR standard, the RTE provides signals that are attached to the data handled by the SWC ports, depending on the communication interface hosted by the port (server/client or sender/receiver). The integrated COM module matches these RTE signals to one of the communication types provided by its NI (i.e., best-effort, time-triggered and rate-constrained) in the NoC simulation. This is done by mapping the RTE signals to the sender/receiver ports on the NI. Furthermore, the integrated PDU router [UO15] is in charge of passing the data from the NoC interface module to the COM module and vice versa.

In case of a μECU hosting sensor/actuator SWCs and I/O BSW modules in the hardware abstraction layer an environment interface module is integrated to connect the μECU with a physical environment model imported into the AUTOSAR-based system simulator.

5.1.4 Co-simulation Coordination

The co-simulation of the simulation models described in Sections 5.1.1, 5.1.2 and 5.1.3 results in the proposed simulation framework for AUTOSAR message-based multicore processors. The co-simulation coordination is based on integrated local coordinators for the coupling of the AUTOSAR simulation level and the NoC simulation level. Figure 5.2 depicts an architecture picture of the co-simulation. The TCP/IP protocol was selected for the communication between the simulation tools. The μECUs implement TCP clients for

the exchange of data with their specific NIs on the NoC simulation. Additionally, the local coordinators use the TCP connection to synchronize simulation steps on both simulation systems.

The socket-based communication makes the co-simulation framework independent from the location of the simulation tools and their operating systems. With this approach an existing AUTOSAR-based system simulator is extended for supporting the simulation of μECUs as application cores in a message-based MPSoC.

The following messages are used for the communication between the two simulation tools:

- ***Data Message.*** It represents a message sent from the μECUs to the NoC local coordinator and vice versa. It contains the following fields:

  1. *Message Status:* This parameter indicates whether the data message is empty or the number of PDUs that it contains.
  2. *Sender ID:* This field contains the ID of the μECU sending the message.
  3. *Receiver ID:* This field declares the destination μECU ID.
  4. *SWC Port:* It indicates for which RTE signal the PDU is matched.
  5. *Payload:* It contains the user data.

  Empty data messages are needed because of the time-triggered behavior for the data exchange between simulation systems and the event-triggered execution of the simulation systems. Fields 4 and 5 represent a PDU. Fields 3, 4 and 5 are repeated in the same order depending on the number of PDUs that the data message contains.

- ***Synchronization Message.*** It is used for the time synchronization of both simulation systems. It is defined as follows:

  1. *Creation Time:* It represents the creation time of the synchronization message.
  2. *Indication:* In a synchronization message sent from the AUTOSAR-based system simulator this field indicates that a communication point was reached. Furthermore, if the synchronization message is sent from the NoC simulation this field indicates that the data exchange has finished and the next communication step can be performed.

5.1.4.1 Local Coordinator for AUTOSAR Simulation

In the AUTOSAR simulation system a local coordinator is used for the synchronization of the AUTOSAR simulation system. The local coordinator uses the TCP connection for the

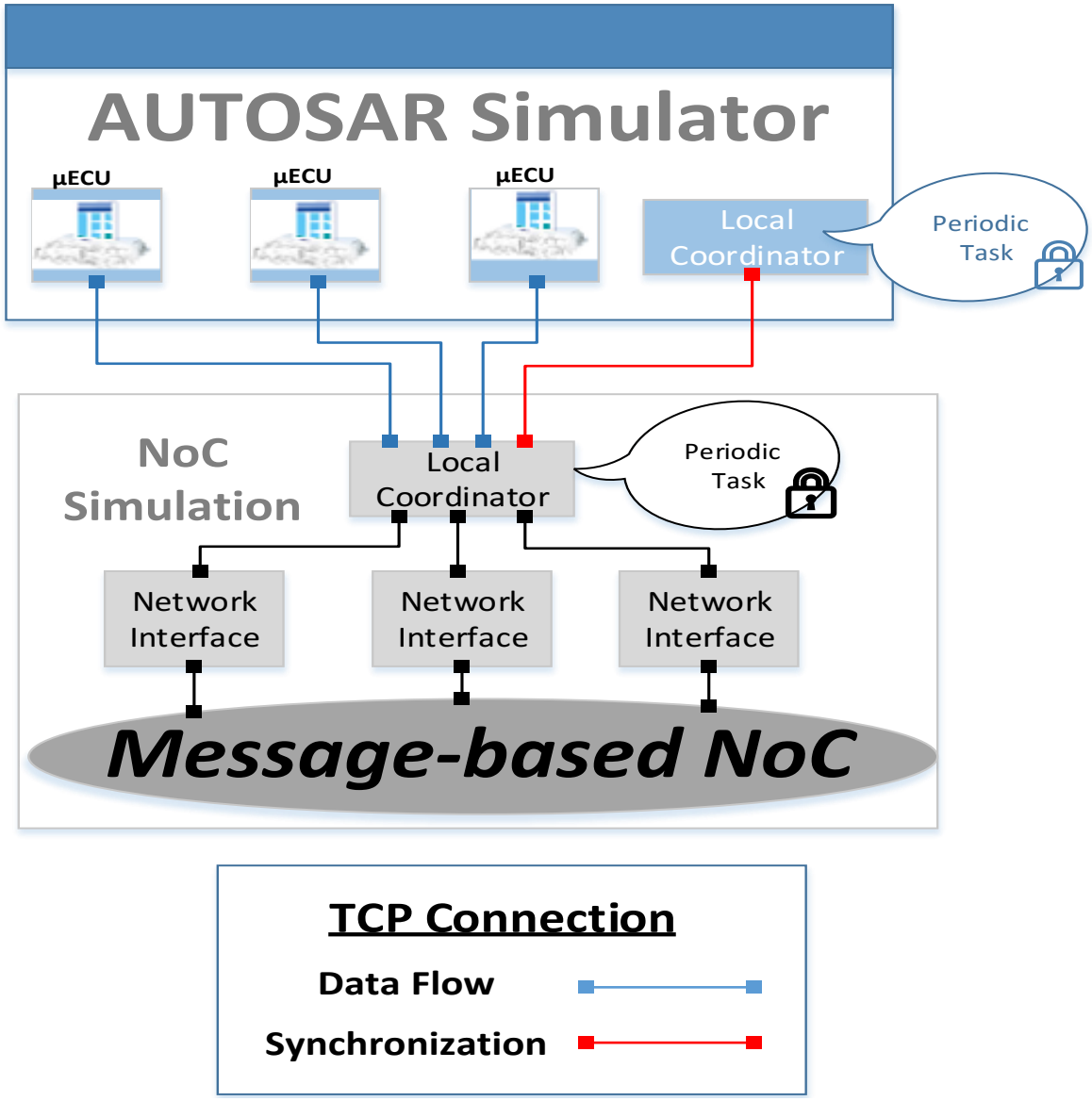

Figure 5.2: Simulation Framework for TIMEA

communication with the message-based NoC simulation. This AUTOSAR local coordinator implements a periodic task with a fixed simulation step size for the co-simulation. Whenever a cycle is finished, the AUTOSAR simulation system is paused and a synchronization message is sent to the NoC simulation. Thereafter the AUTOSAR local coordinator waits for an incoming synchronization message from the NoC simulation in order to run another simulation step on the AUTOSAR simulation. The implementation of the local coordinator for the AUTOSAR simulation will be explained later in details.

5.1.4.2 AUTOSAR µECUs

The μECUs implement the TCP connection for the data exchange with the NoC simulation. Once a communication point is reached in the AUTOSAR-based system simulation, a data message is sent by the NoC interface from each μECU to the NoC simulation. In case of outgoing PDUs from the PDU router on the μECU, these PDUs are integrated into the data message, otherwise the data message is configured to be empty using the message status. Additionally, an incoming data message from the NoC simulation is received by the NoC interface of each μECU, which verifies its message status. If the data message contains a PDU, this is made available to be used by the PDU router of the μECU during the next communication step.

5.1.4.3 Local Coordinator for NoC simulation

The NoC local coordinator implements the TCP connection for the exchange of data with the AUTOSAR simulation and for the synchronization of the NoC simulation system. The NoC local coordinator is used to control the NoC simulation model exactly in the same way as the AUTOSAR local coordinator controls the AUTOSAR simulation. Furthermore, the NoC local coordinator is used to receive the data from the μECUs, to convert it to the adequate message format for the NoC model and to inject it into the respective network interfaces. Also, the NoC local coordinator accepts data from the network interfaces, re-converts the message format and sends it to the μECUs.

The co-simulation is started initiating first the NoC simulation and then the AUTOSAR simulation. The NoC local coordinator waits for the TCP connection of the μECUs and the AUTOSAR local coordinator before triggering the first NoC execution.

Let us discuss how the co-simulation coordination of the NoC model works using the state machine illustrated in Figure 5.3. After starting the simulation in the *initiate* state, the NoC local coordinator configures the parameters to establish the socket-based connection, configures the initial values for the NoC simulation and waits for the connection of the μECUs and the AUTOSAR local coordinator. Thereafter the NoC local coordinator performs

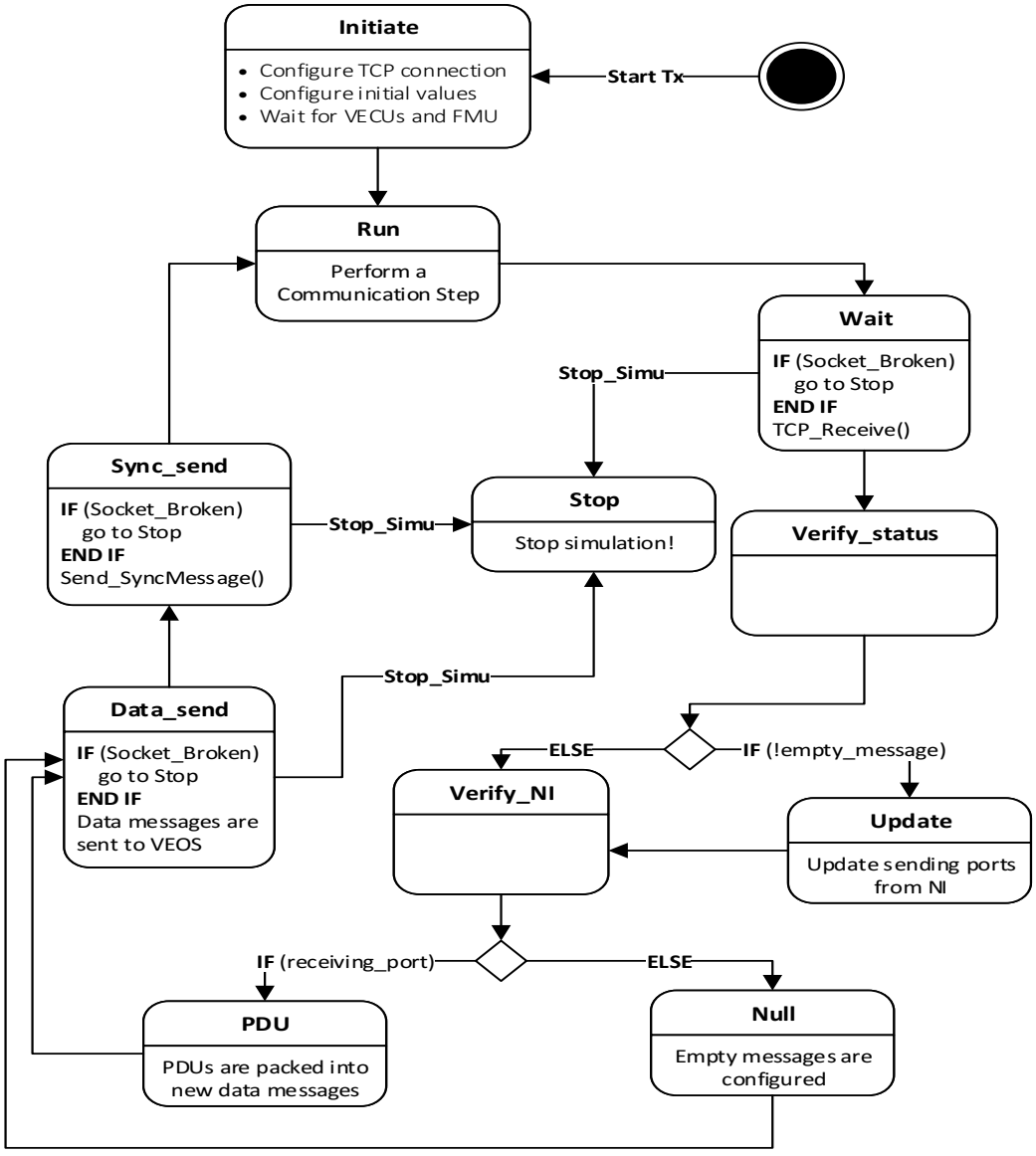

Figure 5.3: State Machine of the NoC Local Coordinator

a communication step on the NoC simulation in the *run* state. In the *wait* state an incoming synchronization message from the AUTOSAR local coordinator is waited for. Once a synchronization message arrives, the status of the data messages received from the μECUs is verified in the *verify_status* state. Thus, if no empty data messages were received, the respective ports of the NIs are updated in the NoC simulation model in the *update* state, otherwise the next state *verify_NI* is directly accessed. In this state, the receiver ports of the NIs are verified and, in case of new data, the *PDU* state is accessed wherein the data is packed into PDUs and these PDUs are stored into new data messages, otherwise these data messages are configured to be empty in the *null* state. In the *data_send* state data messages are sent to the respective μECUs and thereafter a synchronization message is sent to the AUTOSAR local coordinator in the *sync_send* state. Finally, the run state is accessed again and the whole procedure is repeated.

Furthermore, if the co-simulation is stopped by the AUTOSAR-based system simulator the stop state is accessed from *wait*, *data_send* or *sync_send* state. In this state the NoC simulation is stopped.

5.2 Implementation of the Co-simulation Framework

In the implementation of the co-simulation framework, the AUTOSAR system is modelled and simulated with the dSpace AUTOSAR tools including the VEOS simulation framework. Additionally, the SystemC and GEM5 simulators are used for the simulation of the NoC communication behavior.

5.2.1 Simulation System for AUTOSAR Micro-ECUs

The VEOS simulation tool serves for the simulation of the AUTOSAR μECUs. The VEOS environment is the dSpace software for the simulation of ECUs and environment models on a host PC [dSp14d]. The following tools comprise the VEOS software:

- *The VEOS simulator* is the simulation platform for PC-based simulation of simulation systems. The experimental tool ControlDesk can access the VEOS simulator for testing ECU software [dSp14c].

- *The VEOS player* is the software tool for the integration of simulated ECUs and environment models into a simulation system and for the execution control of a simulation running in the VEOS simulator.

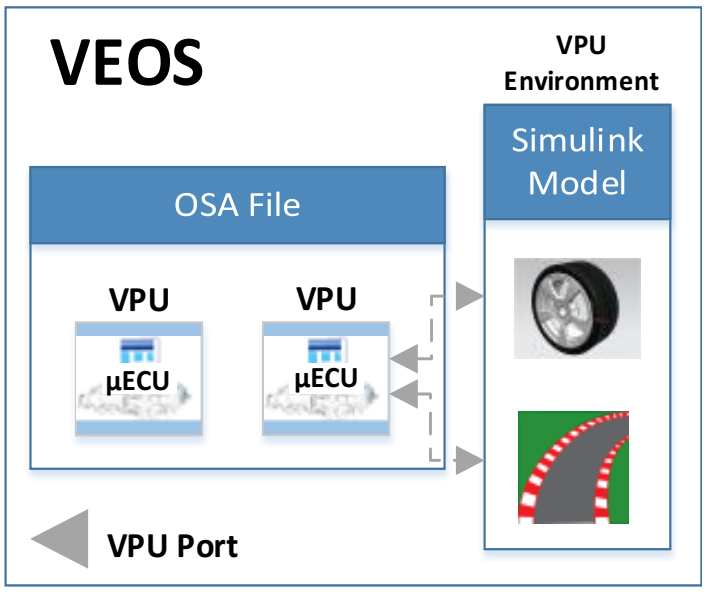

Figure 5.4: VEOS environment

- *dSpace Target for offline simulation* is a Simulink Coder that allows to build environment models for simulation with VEOS. Environment models serve as virtual representations of an ECU environment during virtual validation scenarios.

SystemDesk allows the generation of simulated AUTOSAR ECUs and the integration of them in a simulation system. This simulation system is built with SystemDesk, building an Offline Simulation Application (OSA) file [dSp14a].

The OSA file comprises binary files for each simulated ECU. The OSA file is needed to run the simulation system with the VEOS player. In a running simulation the simulated ECUs can be accessed by an experimental tool like ControlDesk for parameter calibration via the Universal Measurement and Calibration Protocol (XCP). XCP is *"a bus-independent, master-slave communication protocol to connect ECUs with calibration systems"* [ASA17]. A2L files are used to access and interpret the data transmitted via XCP. A2L files describe the variables available for measurement and calibration [ASA17].

Simulated AUTOSAR ECUs running on VEOS are configured according to the software architecture of Figure 5.1, so they serve as the AUTOSAR μECUs on the TIMEA platform. Additionally, the environment models are integrated into the AUTOSAR simulation using the Automotive Simulation Models (ASMs) from dSpace which can be configured using Simulink and the dSpace simulation coder TargetLink as explained earlier.

Each μECU and environment model is represented as an independent Virtual Processing Unit (VPU) by the VEOS player.*"VPU is a generic term for a part of a simulation system that can be run in an offline simulation by VEOS"* [dSp14a]. Using VEOS player, μECUs

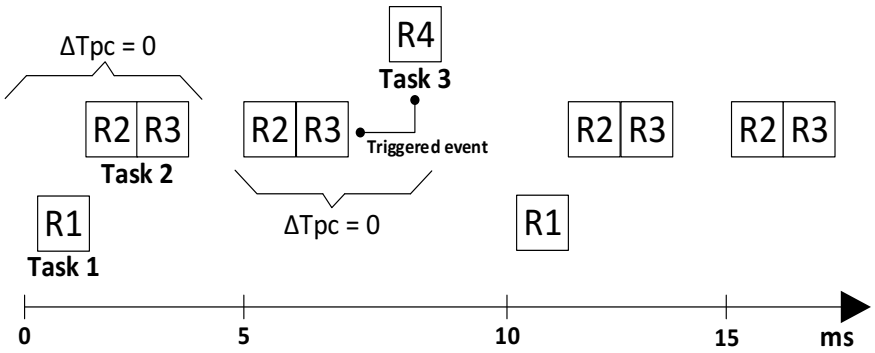

Figure 5.5: VEOS scheduling example with three tasks

and environment models can be interconnected through their VPU ports. VPU ports provide direct communication between VPUs without using off-chip communication [dSp14d]. In the VEOS player VPU ports of the μECUs are represented by the sensor/actuator SWC ports used by the I/O hardware abstraction layer, while VPU ports of the environment models are represented by simple Simulink input or output blocks. An architecture picture of the simulation environment and its components is depicted in Figure 5.4.

During the simulation, an emulated AUTOSAR OS is used to run the simulated μECUs on a host PC [dSp14d]. The main purpose of the OS is to invoke OS tasks. VEOS simulates correctly the order of OS task and function calls. The simulation time is the same for all the μECUs and environment models in the simulation system. VEOS performs a zero execution time assumption for the simulation. The tasks are executed instantly in the virtual simulation time, which means *"all tasks are assumed to run on the μECUs with an execution time of* $\Delta t_{PC} = 0ms$*"* [dSp14d]. This zero execution time assumption is not a problem for the simulation of TIMEA since time-triggered or rate-constrained activation is independent from the execution times, so the focus is on the system-level timing and not on the execution times of the individual tasks.

Figure 5.5 depicts a scheduling example with three tasks. Task 1 is triggered periodically every $10ms$ and calls runnable R1. Task 2 is triggered by the OS every $5ms$ and calls runnables R2 and R3. Task 1 is called before task 2 because of its higher priority. Task 3 is an event-triggered task called sporadically by Task 2. Task 3 calls runnable R4. During the simulation of this scenario, VEOS triggers tasks every 5 ms. At $t = 0ms$, Task 1 is triggered and runnable R1 is called. Then Task 2 is triggered and runnables R2 and R3 are called. After runnable R3 returns, VEOS advances the simulated clock to $t = 5ms$ and triggers Task 2 again. The same example is used later to explain the coordination of the co-simulation.

5.2.2 Simulation System for Network-on-a-chip Communication

For the simulation of the NoC communication behavior two different NoC simulation models are used according to the works presented in [OO15] and [AO15]. The system described in [OO15] provides a SystemC-based NoC simulation which allows us later to simulate the system cores for hardware acceleration in combination with a message-based NoC. On another hand, [AO15] describes a GEM5 NoC simulation system which provides a more accurate simulation time for the analysis of the temporal behavior.

5.2.2.1 SystemC-based TTNoC Simulation

SystemC is a widely used system-level modeling language for event-driven modeling [OG09] [CMM+15]. Timing accuracy in SystemC is ranging from untimed to cycle-accurate where precise temporal specifications and SystemC modules can be simulated to validate the behavior of the platform. Additionally, Transaction Level Models (TLMs) [Ghe06] are used in the simulation frameworks to enhance the overall simulation speed. TLMs capabilities such as interface-based communication, blocking and non-blocking process structures, bidirectional and unidirectional transactions inspired researchers to develop simulation frameworks owing to the adaptability and accuracy that SystemC/TLM provides.

The simulation of the NoC is based on the work presented in [OO15]. TLM/SystemC is used to implement the communication behavior of a TTNoC. Time-triggered channels are expressed using SystemC channels, while message-based communication is managed using interfaces. Interfaces of the TTNoC are inherited from the *sc_interface* class of SystemC. Time-triggered channels are defined as bidirectional transactions and process structures are managed as blocking and non-blocking processes. The overall system model is illustrated in Figure 5.6.

In the Time-Triggered Network Interfaces (TTNIs), a communication interface is required to transmit time-triggered messages by providing procedures to send/receive messages to/from other cores. Moreover, the TTNI is responsible for mapping the outgoing messages to the time-triggered communication according to the Time Division Multiple Access (TDMA) schedule.

The transmission of *time-triggered messages* is based on the following parameters: period, phase, message size, sender core ID and receiver core IDs. The formed time-triggered messages in the TTNI are periodically transmitted based the time-triggered communication schedule. The phase of the message defines the start time with respect to the start of the period. The message size is determined by the message payload, and the routing requires the source and the destination core IDs of the message.

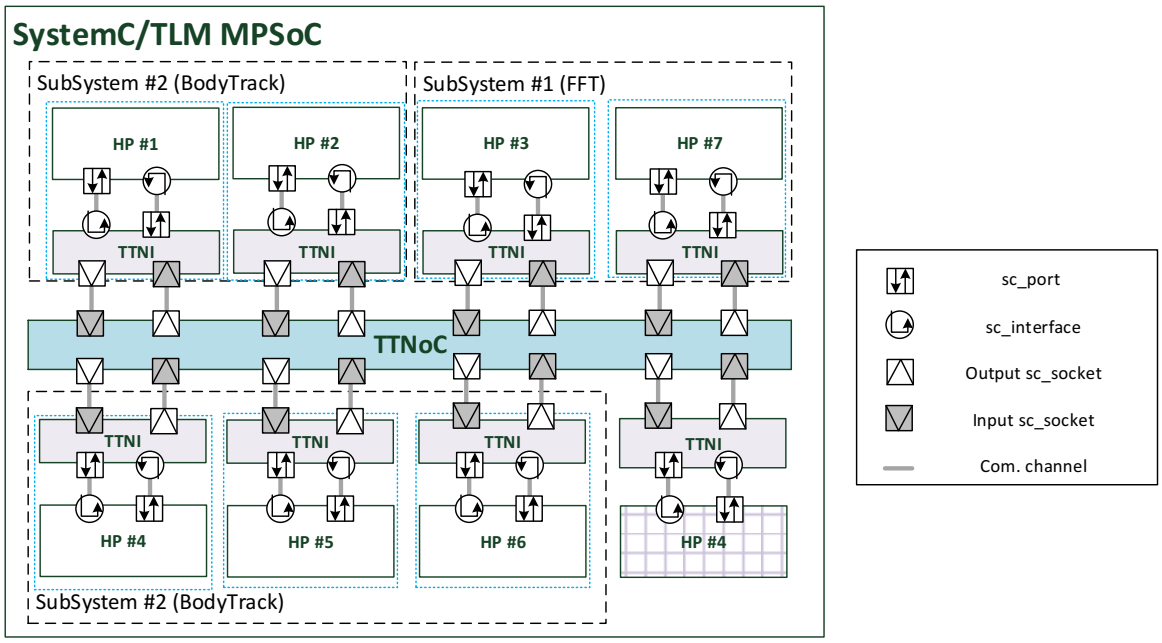

Figure 5.6: SystemC TTNoC simulation Model

During the execution of the framework, a cyclic dispatcher method is called in the network interface to determine whether a time-triggered message has to be sent according to the predefined schedule. In this case, the formed time-triggered message is redirected to the corresponding channel port.

To receive a message at the cores, the network interface ports are notified by the time-triggered channels that a message is received. A receive function is triggered in the core that invokes the incoming message.

The TTNoC establishes the time-triggered schedule and the time-triggered communication channels. The schedule is a static configuration parameter and loaded at the beginning of the simulation. The TTNoC uses time-triggered channels that represent the temporal and spatial allocation of physical links of the simulated TTNoC. The time-triggered channels can simulate communication delays that represents the router delays of a real NoC by defining the configuration and schedule of the simulated use case.

The time-triggered communication table is defined as a Comma-Separated Values (CSV) file that contains the message-based communication configuration.

As shown in the example configuration in table 5.1, time-triggered messages are scheduled to be transmitted periodically. The configuration table defines the time-triggered message period, phase, senderCoreID and ReceiverCoreIDs. The message phase parameter defines the start transmission time in relation to the defined period. In addition to that, the static configuration defines the sender μECU ID and the receiving μECUs.

Message	Period (ms)	Phase (µs)	Sender Core	Receiver Cores	Delay (ns)
Angular speed	1	0	0	1	50
Braking force	1	1	4	1	50
Relative slip	1	2	1	3	50
Slip comparison	1	3	3	4	50

Table 5.1: NoC configuration in SystemC model

Log files provides information about the injection time, receiving time, delay and jitter of the time-triggered messages that are sent through the TTNoC during a SystemC simulation. A detailed explanation of the log file generation and in-depth description of the SystemC-based TTNoC simulation model is provided in [Owd16].

5.2.2.2 NoC simulation with GEM5

The employed simulation environment is based on the GARNET interconnection network [AKPJ09] inside the GEM5 multicore simulator [BBB+11]. The simulated MPSoC models a classic five-stage pipelined router, including input buffers, routing logic, allocators and the crossbar switch, with Virtual Channel (VC) flow control at flit-level. It also supports the three mentioned communication paradigms, i.e., best-effort, time-triggered and rate-constrained, using a Time-Triggered Extension Layer (TTEL) [AO15] which is added to the NI of GARNET (cf. Firgure 5.7). This extension layer takes care of the timely injection of the messages and priorities of the rate-constrained messages and the NI (of GARNET) generates the flits of the messages and injects them into the NoC.

In the simulated MPSoC, *ports* provide the interface to the NoC for the cores. They include a *data area*, *port configuration parameters* and *port status flags*. The data area stores the messages and can be a single buffer, which is overwritten in case of messages with state semantics, or a queue in case of messages with event semantics. The port configuration is implemented as a separate class *PortConfig* and is instantiated by CSV configuration files, once the simulation is started. The generated *PortConfig* object is afterwards used for the instantiation of the ports. This class (*PortConfig*) contains the parameters of the communication channel, e.g., the direction of the port, the path to the destination, temporal parameters such as the period and the phase for time-triggered messages and the MINT for the rate-constrained messages.

Messages are stored at output ports by the sender core. They are dequeued into the NoC (considering the priorities and the schedule) by the extension layer. At the destination side, the messages are stored at input ports to be dequeued by the core. The core can invoke several API routines for sending and receiving the messages. These can be used by the simulated

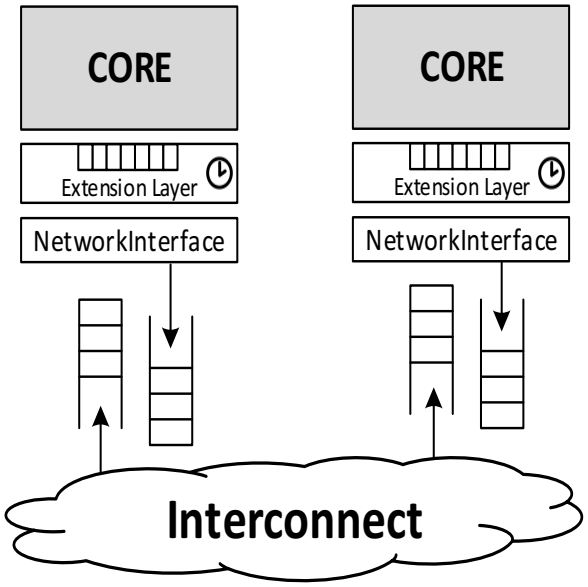

Figure 5.7: Simulated NoC in GEM5

application or by the defined local coordinator which connects two simulation environments in this work.

The time-triggered behavior in GEM5 was technically realized by extending the *Consumer* class and adding a *Scheduled Wake-up* to this class. This class is a virtual base class of all classes that can be the targets of wakeup events. This change enables the classes inherited from *Consumer* to be waked-up not only by the linked consumer, but also by the scheduled wake-up (using *schedule* method).

The NI and the routers in GARNET handle the messages and flits of different types and priorities in the same manner, as the type and the priority of the messages are abstracted from them. This is the task of the scheduler to establish a collision-free communication of the time-triggered messages and to guarantee no impact of the rate-constrained message on the time-triggered messages due to contention at the resources (e.g., buffers at routers, physical links).

The scheduler triggers the injection of time-triggered messages according to an *a priori* defined time-triggered communication schedule. In order to avoid collisions of messages at the NoC level, a scheduler uses the concept of *Timely Block*. Timely block guarantees the absence of collisions between time-triggered and rate-constrained messages by blocking the injection of rate-constrained message during the *guarding windows*. The schedule for opening and closing instants are defined based on the time-triggered communication schedule.

The scheduler employs source-based routing in order to achieve *spatial partitioning*. Source-based routing enables the scheduling tools to determine the path through which, flits

Message	Sender Core	Receiver Core	Message Size (bytes)	Temporal Configuration
Angular speed	0	1	60	TT (period: 1ms)
Braking force	4	1	100	RC (MINT: 3 ms)
Relative slip	1	3	80	RC (MINT: 3 ms)
Slip comparison	3	4	70	RC (MINT: 3 ms)

Table 5.2: NoC configuration in GEM5 model

of the message are traversing. The NI uses a look-up table or computes the path for each message and includes the routing information of the head flits.

The MPSoC simulation system is highly configurable using CSV files. These files enable us to configure a wide range of parameters, such as the topology of the NoC, the number of cores and the allocation and the configuration of ports. Moreover, the schedules for the injection of the time-triggered and the rate-constrained messages are defined using the CSV files. An example table of a NoC configuration in GEM5 is presented in table 5.2

In addition to the configuration parameters and the schedules, log files are also implemented as CSV files. These files provide information about the time behavior of the messages during a GEM5 simulation, so latencies and jitters can be calculated.

5.2.3 Implementation of the Coordination Interface

In this section the implementation of the co-simulation coordination at each simulation system is described. The TCP/IP protocol is used by a server in the NoC simulation and clients in the VEOS simulation. The blocking socket-based communication is used to suspend and resume both simulation tools.

5.2.3.1 Implementation of the AUTOSAR local Coordinator

The Functional Mock-up Interface (FMI) was selected for the implementation of the local coordinator for the AUTOSAR simulation. The FMI is a tool-independent standard [BO+11] that provides support for both model exchange and co-simulation of dynamic models. These models are shipped as Functional Mock-up Units (FMUs) containing an XML description file and the compiled application C-code.

FMI defines interfaces for the data exchange and the time synchronization which are provided by the FMU and used by a master algorithm implemented by the hosting simulator. Although the FMI standard is not just limited to the automotive domain, it is currently highly used by the most popular AUTOSAR simulation tools (e.g., CANoe, AUTOSAR builder, VEOS, etc) [Modls] for the co-simulation of simulation models of different suppliers

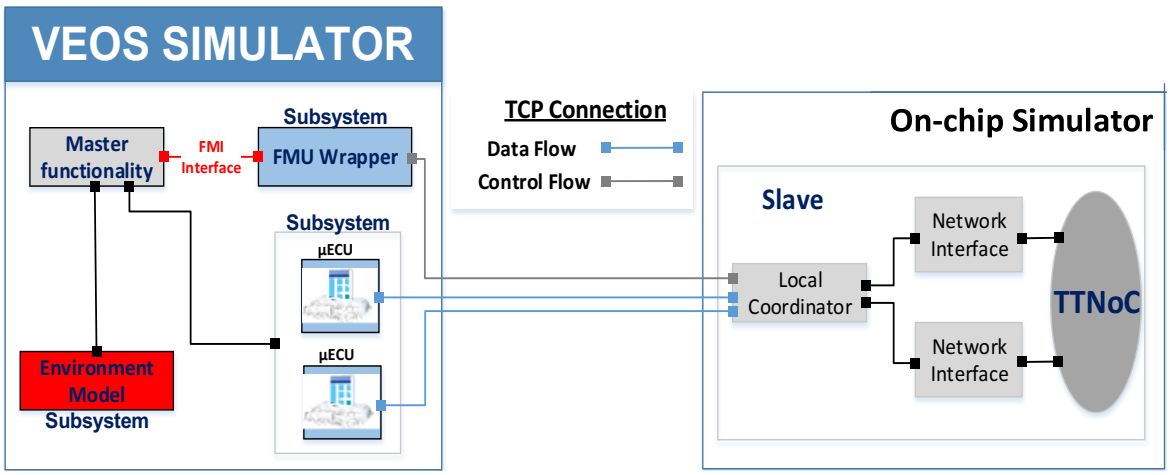

Figure 5.8: Co-simulation Coupling of VEOS and NoC simulation

and OEMs. This allows us to introduce an interface according to FMI as a key part in the implementation of the AUTOSAR local coordinator, using it for the synchronization between the AUTOSAR simulator and the NoC simulation. However, since most of the on-chip simulators (e.g., GEM5 [BBB+11] and SystemC [CMM+15]) do not support FMI, the presented co-simulation coordination is not just limited to an FMI implementation as explained in the previous section.

The FMI standard for co-simulation defines discrete communication points tc_i for the data exchange between simulation models running in different simulation tools. Each communication point is known as a synchronization point between simulation models. The time between two consecutive communication points represents the communication step size:

$$hc_i = tc_{i+1} - tc_i \tag{5.1}$$

A communication step is defined as:

$$tc_i \longrightarrow tc_{i+1} = tc_i + hc_i \tag{5.2}$$

During a communication step the simulation of the models is performed independently in the different simulation tools.

Figure 5.8 illustrates an architecture picture of the co-simulation coordination applying the FMI standard. As depicted in this picture, the AUTOSAR local coordinator is implemented as a wrapper and integrated into the AUTOSAR-based system simulator for the time synchronization with the NoC simulation model running in the on-chip simulator.

The AUTOSAR-based system simulator realizes the master functionality defined by the FMI standard. The FMU wrapper is controlled by the simulator using functions to

synchronize the FMU simulation with the simulation of the μECUs and the environment models, proceeding in communication steps from the initial time $tc_0 = t_{start}$ to the finish time $tc_N = t_{stop}$. Likewise, the FMU wrapper uses the TCP connection to trigger the NoC simulation in the on-chip simulator. It is important to note that FMI functions for data exchange between the μECUs and the NoC model are not provided by the defined FMU wrapper. The data exchange is realized directly between the μECUs and the NoC simulation when a communication point is reached by the FMU wrapper as explained previously.

Since VEOS only allows a fixed communication step size ($hc_i = \mathbb{k}$) for the co-simulation of an FMU with an AUTOSAR simulation, its choice becomes a key parameter for the implementation of the framework. Thus, the choice of the communication step size influences the accuracy and efficiency of the co-simulation framework. Increasing the communication step would reduce the synchronization overhead between the simulation environments and hence reduce the accuracy of their results whilst a short communication step would decrease the simulation performance hence making it less efficient. Since the RTE and the AUTOSAR BSW can only react to an event occurring in the NoC simulation within the interrupt detection latency, we use the minimum interrupt detection latency as a fixed communication step size for the co-simulation with the NoC simulator. We assume $hc_i = 1\mu s$ as the minimum interrupt detection latency. The reason behind this assumption is that the RTE and most of the AUTOSAR BSW are designed for latency and bandwidth requirements that are orders of magnitude higher than $1\mu s$.

VEOS allows the integration of FMUs for co-simulation of subsystem models which have been imported together with their solver and do not require additional tools for their simulation. Using FMI the VEOS simulator is able to control the FMU simulation. A TCP client is integrated into the FMU to establish the communication with the NoC simulator. This approach extends the VEOS simulator for co-simulation of FMUs requiring an extra simulation tool.

The initialization mode of the FMU wrapper is used to configure the socket-based communication. In the start function of the FMU, the socket-based communication parameters are configured to setup the TCP client connection with the NoC simulator.

In the step mode of the FMU wrapper the blocking mechanism provided by the socket-based communication is used to suspend and resume the VEOS simulation with synchronization messages. A function for sending *synchronization messages* is allocated as well as a function that waits for incoming *synchronization messages* from the NoC simulation. Whenever a communication point is reached, the function for sending synchronization messages is called and thereafter the function that receives a synchronization message from the NoC simulation. Thus, the VEOS simulation is suspended till a synchronization message arrives

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<fmiModelDescription  fmiVersion="2.0"  modelName="gem5FMU"
guid="{8c4e810f-3df3-4a00-8276-176fa3c9f008}"  numberOfEventIndicators="0">
  <CoSimulation  modelIdentifier="gem5FMU" canHandleVariableCommunicationStepSize="false">
    <SourceFiles>
      <File name="gem5FMU.c"/>
    </SourceFiles>
  </CoSimulation>
  <LogCategories>
    <Category name="logAll"/>
    <Category name="logError"/>
    <Category name="logFmiCall"/>
    <Category name="logEvent"/>
  </LogCategories>
  <DefaultExperiment    startTime="0"  stepSize="1.0000000000000000e-03"/>
  <ModelVariables> </ModelVariables>
  <ModelStructure>
    <Outputs>
      <Unknown index="1" />
    </Outputs>
  </ModelStructure>
</fmiModelDescription>
```

Figure 5.9: Model Description of the FMU Wrapper

from the NoC local coordinator. During this time the exchange of data messages between the μECUs and the NoC simulation is performed. Once a synchronization message arrives, the FMU simulation is unblocked and the next communication step can be performed on the whole VEOS simulation.

As mentioned previously, the model description is provided as an XML file. The model description file for the FMU wrapper is depicted in Figure 5.9. As shown is this picture, the FMU does not handle a variable communication step size defining a fixed communication step size of $hc_i = 1\mu s$. The start time for the FMU simulation is defined as $tc_0 = 0s$, which is required by the VEOS simulator for the simulation of FMUs.

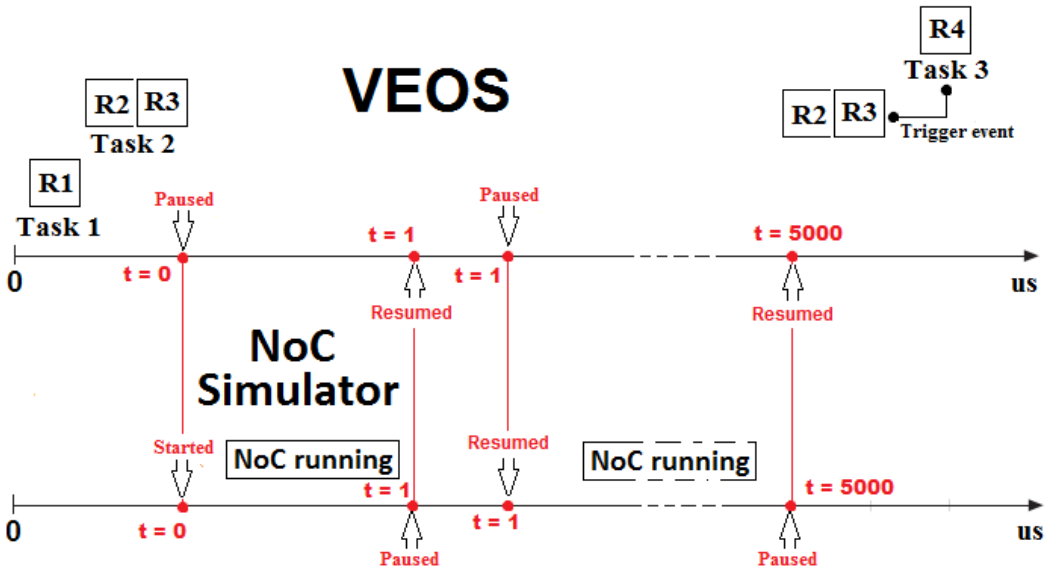

Figure 5.10: Co-simulation Example with Three Tasks

5.2.3.2 Local Coordinator for the NoC simulation

The NoC coordinator is implemented in both, the GEM5 model for on-chip communication and the NoC simulation model based on SystemC. Both of them are realized according to the state machine presented in Section 5.1.4.3 with a task period for loop execution of $hc_i = 1\mu s$ as implemented in the FMU wrapper on VEOS.

1) GEM5 Implementation: The local coordinator is realized in GEM5 using Python and it is implemented as a task controlling the execution of the NoC simulation (implemented in C++) and the exchanging of data between the NIs and the VEOS simulation. A TCP server is integrated into the local coordinator for the data exchange with the μECUs and to establish the co-simulation coordination with the FMU wrapper. The blocking mechanism of the TCP communication is used in the same way and it is implemented in the FMU wrapper in VEOS. Once a communication step was performed on the NoC simulation, the local coordinator invokes the function for receiving synchronization messages from the FMU wrapper, blocking temporally the GEM5 simulation. Thus, the GEM5 simulator is regularly suspended and the data exchange is possible during the co-simulation. This blocking mechanism is controlled based on the execution steps of the state machine described in Section 5.1

2) SystemC Implementation: In the SystemC-TTNoC simulation model a process implemented in C serves as a TTNoC local coordinator and a TCP/server is integrated. A function that waits for the connection of the μECUs and the FMU wrapper is allocated to the TTNoC local coordinator. Thereafter, an infinite loop implements a function that waits for

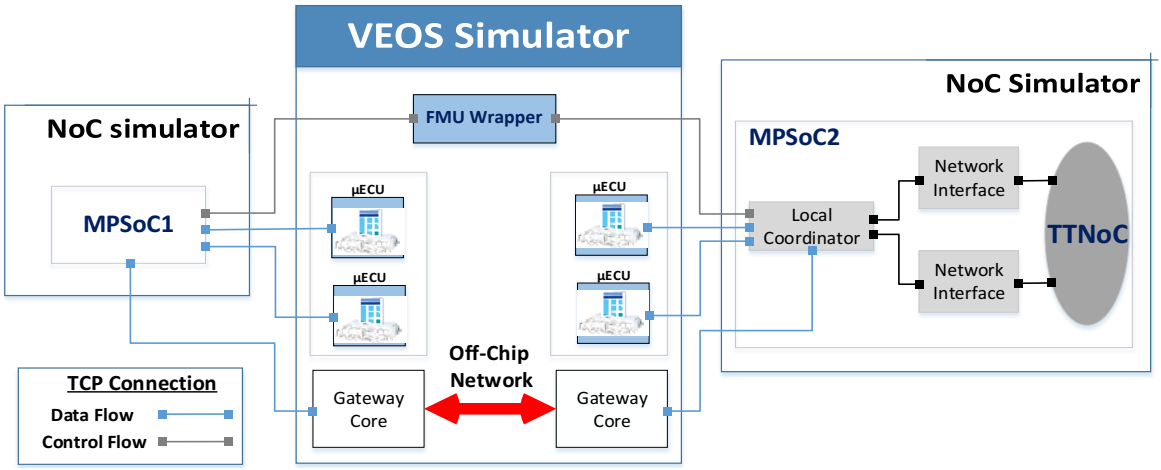

Figure 5.11: Co-simulation of Distributed systems based on TIMEA

the *synchronization messages* from the FMU wrapper and the *data messages* from µECUs. In case of new *PDU messages*, this function updates the *payloads* of the respective network interfaces. After this function the TTNoC model is run for $1\mu s$ and the function for sending *synchronization messages* is called. In addition, this function also takes new payloads in the network interfaces (in case of new data), constructs the *data messages* and sends them to the µECUs. Thus, after one loop execution the simulation is paused till a new *synchronization message* and new *data messages* arrive from the VEOS simulation.

An example of the synchronization of both simulation systems is shown in Figure 5.10, which resembles the scenario from Figure 5.5. Once the NoC simulation model receives a synchronization message from the FMU wrapper and the *data messages* from the µECUs, the NoC local coordinator verifies whether there is new data coming from the µECUs. In case of new data, the *message payloads* of the respective NI are updated (using *sender ID* and *Receiver ID*) and the NoC simulation model is run for one simulation step ($1\mu s$). Thereafter the simulation is paused and, in case of new data in the NIs, *message payloads* are packed into *data messages* and sent to the µECUs. Finally a synchronization message is sent to the FMU wrapper on VEOS to give run permission for a following simulation step on the AUTOSAR simulation.

5.3 Extension of the Co-simulation Coordination

In order to support the simulation of distributed systems based on TIMEA the local coordinator of the AUTOSAR simulation is extended for the synchronization of multiple NoC simulation models. That means, AUTOSAR µECUs are mapped to different MPSoCs using different instances of the NoC simulation. Thus, after one simulation step, the AUTOSAR

local coordinator sends a synchronization message to each NoC simulation and waits for the responses of all of them before performing the next simulation step. In Figure 5.11 an example of a co-simulation of VEOS with two NoC simulation models is presented.

As explained previously (see Section 5.2.3.1), the VEOS local coordinator is implemented as an FMU wrapper using the FMI standard. For the coordination of several NoC simulations, in the initialization mode of the FMU wrapper multiple TCP client connections are established depending on the number of TIMEA platforms that are connected to the distributed system application. In the start function of the FMU, the socket-based communication parameters are configured to setup all the TCP client connections with their respective NoC simulation model.

In the step mode of the FMU wrapper, for instance, whenever a communication point is reached, a synchronization message is sent to each one of the NoC simulation models and thereafter a function that receives synchronization messages from each one of the NoC simulations is called. Thus, the VEOS simulation is suspended till all the synchronization messages arrive from the NoC local coordinators within the co-simulation system. During this time the exchange of data messages between the μECUs and the multiple NoC simulations is performed. Once all the synchronization messages arrive, the FMU simulation is unblocked and the next communication step can be performed on the AUTOSAR μECUs.

Additionally, the *data message* described in Section 5.1.4 is extended for the handling of the DMTC coordination messages and off-chip communication messages. A *data message* for a distributed simulation system of TIMEA platforms includes the following elements:

- *Header:* This field indicates whether the message represents either an on-chip message, an off-chip message or DMTC coordination message.
- *Type:* This parameter is used to distinguish between the different types of the DMTC coordination messages.
- *Status:* Indicates whether the message is empty or not. In case the message is not empty, the number of data and memory operations contained in the message is denoted.
- *Sender ID:* Contains the ID of the μECU sending the data or a memory operation. The destination core is known from the on-/off-chip communication schedules.
- *Payload:* Contains the data or the memory operations.

Chapter 6

Development Process of TIMEA

The realization of the proposed AUTOSAR message-based multicore architecture with the co-simulation framework is presented in this chapter. Based on the system architecture introduced in Chapter 4, the dSpace AUTOSAR development tools are used for the design and configuration of the defined AUTOSAR μECUs. SystemDesk and TargetLink serve for the development of the system architecture based on SWCs and their internal implementation behavior. Additionally, SystemDesk is used for the configuration and generation of simulated μECUs with the integrated I/O proxy module and the health monitoring service module.

The SystemC-based TTNoC simulation model serves for the implementation of the I/O gateway core and memory gateway core as SystemC-based core processors. Moreover, the development and generation of the off-chip network gateway core is carried out with the SystemDesk architecture tool.

6.1 Implementation of the AUTOSAR Micro-ECUs

Simulated AUTOSAR μECUs are developed using the dSpace AUTOSAR solutions (SystemDesk and TargetLink) [dSp14c]. These tools offer a high level of maturity for designing AUTOSAR-based systems and strict compliance to the AUTOSAR workflow [AUT16g]. SystemDesk is a software tool that allows to model the software architecture of distributed automotive electronic systems consisting of AUTOSAR SWCs. The SWCs host the application functionality and are developed independently from each other and the specific ECU technology. For the functional behavior modeling the TargetLink AUTOSAR module is available, which allows the import and export of SWC descriptions. It also supports the AUTOSAR-compliant production code generation for the SWCs from the Simulink-Stateflow graphical environment [dSp14b].

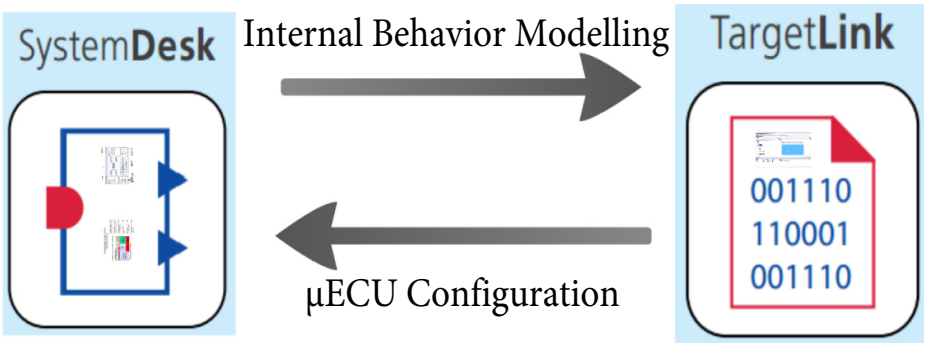

Figure 6.1: Tool´s Interaction in the AUTOSAR µECU development

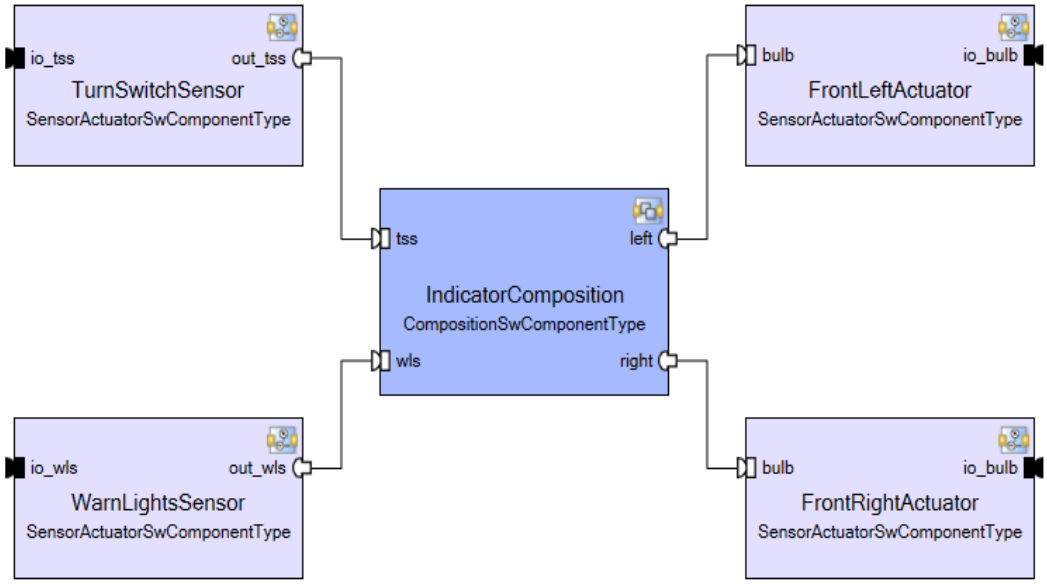

Figure 6.2: Modelling of the AUTOSAR Software Components

Furthermore, SystemDesk is used to generate ECU configurations and for the generation of simulated μECUs for virtual validation scenarios [dSp14c]. A picture pointing out the interaction between the tools in the development process of the AUTOSAR μECUs is depicted Figure 6.1.

6.1.1 Software Architecture Modeling

The modeling of the system architecture is performed using the dSpace system architecture tool SystemDesk. The software architecture is designed in terms of SWCs and the interaction between these SWCs (see Figure 6.2).

A set of SWCs is defined, where each one encapsulates a functionality and the integration of all of them represents the specific automotive system. Once the SWCs are identified,

input and output ports (so-called provided and required ports by AUTOSAR) must be added to each one of these SWCs. After defining the AUTOSAR SWCs and their ports, specific sender/receiver or client/server interfaces are defined. An interface is assigned to every port, describing the data or operations provided or required by a SWC via its ports.

An internal behavior is assigned to each SWC, so their runnables, RTE events, exclusive areas and per instance memories are specified. The initial modeling of the internal behavior of each SWC is performed as follows.

- Several runnable entities are defined for the SWC internal behavior.
- RTE events (e.g., timing event, data received event, etc) are defined, which will trigger the created runnable entities.
- Interrunnable variables are created, which are variables that are just used by the runnables of the specific SWC. A specific variable data type is assigned to each interrunable variable. Also initial constant values for the interrunable variables are defined.
- Data accessed by the defined runnable entities is specified. Interrunnable variables and variable data prototypes defined in the interfaces of the SWC ports are selected.
- A data type mapping set is created, which maps the application data types to implementation data types. Implementation data types specify implementation details such as SW base types or endianness.
- A constant specification mapping set is created and constant specifications of the implementation types must be defined, which maps constant specifications that have application data types to constant specifications that have implementation types.

Once an internal behavior was already assigned to each SWC the descriptions [AUT16g] are exported as AUTOSAR XML files for behavior modeling with TargetLink.

6.1.2 Internal Behavior implementation for the AUTOSAR SWCs

The behavior modeling of the SWCs is performed using Simulink and the dSpace production code generation tool TargetLink. This software tool provides an AUTOSAR module for modeling, simulating and code generation of AUTOSAR SWCs.

The SWC descriptions of the SystemDesk defined SWCs are imported to the TargetLink Data Dictionary Manager (TDDM). A Simulink frame model from the imported AUTOSAR SWC files is generated in a new Simulink model environment using the TargetLink frame

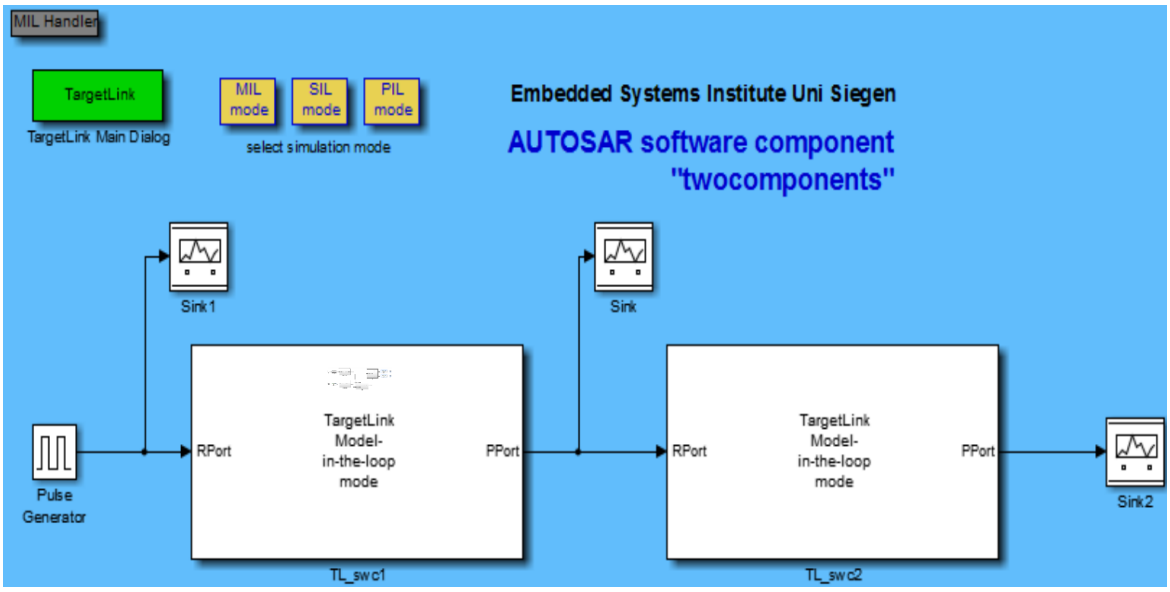

Figure 6.3: Simulink Environment for modeling SWC Behavior

model feature (see Figure 6.3). The generated frame model works as a starting point for modeling the SWC implementation, having systems representing the SWCs and subsystems representing the runnables within the SWCs. The internal behavior of the SWCs is graphically developed using Simulink blocks and the special TargetLink blocks of the AUTOSAR TargetLink module.

Once the SWC behaviors are modeled, it is possible to perform a Model-in-the-loop (MIL) simulation of the graphical Simulink model to verify the model behavior. An AUTOSAR SWC implementation (C code) is generated for each SWC and a Software-in-the-loop (SIL) simulation can be performed. The new AUTOSAR SWC descriptions with the generated SWC implementations are exported from the TDDM for the generation of the ECU configurations using SystemDesk.

6.1.3 Configuration of the AUTOSAR Micro-ECUs

The SWC descriptions provided by TargetLink (with the corresponding SWC implementations) are re-imported to SystemDesk for the completion of the AUTOSAR-based system. SystemDesk allows to define ECU instances that can have communication controllers and connectors for accessing a physical channel of a communication cluster in the context of a system [dSp14a]. Depending of the specific automotive functionality, a set of ECU instances is defined. SWC ports are connected to each other to indicate the data flow between SWCs through the VFB. Once the ECU instances are identified, the SWCs are mapped to them in order to define the automotive system.

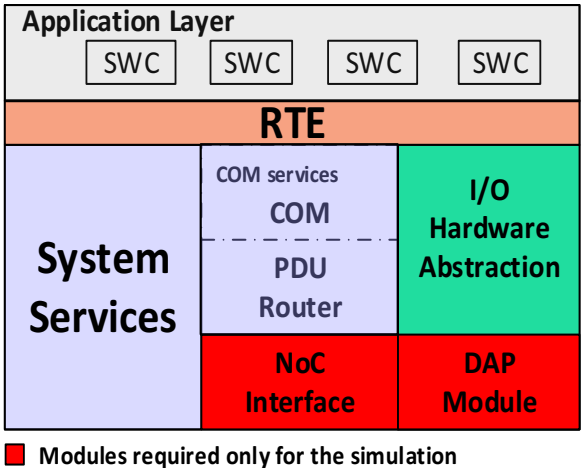

Figure 6.4: Micro-ECU Configuration with I/O BSW Implementation

The BSW and RTE of the ECU instances can be configured and generated using SystemDesk. In the simulation environment proposed in Chapter 5, ECU instances are configured to simulate the μECUs in the TIMEA platform. In this section we present the configuration process for two type of μECUs: ***1)*** for μECUs holding an I/O abstraction implementation as defined by AUTOSAR, ***2)*** for μECUs holding the introduced I/O proxy modules and health monitoring module.

6.1.3.1 Configuration of a µECU with AUTOSAR I/O abstraction implementation

Figure 6.4 presents the BSW structure of this kind of AUTOSAR μECUs. Red modules are required for connecting the μECU with the simulated environment and the NoC simulation. The configuration of the BSW and RTE for each ECU instance is performed as follows.

- *ECU Configuration.* A single reduced ECU configuration (without off-chip network communications, external memory access and special complex driver support) is selected for each ECU instance. An I/O hardware abstraction module and a Data Access Point (DAP) module are added to the ECU configurations. The I/O hardware abstraction module is a BSW module for accessing I/O signals [AUT16b]. The DAP module provides the needed information to integrate the μECUs in a simulation system [dSp14a]. The integration of the I/O abstraction module and DAP module is necessary for μECU hosting sensor/actuator SWCs.

- *Generation of the I/O abstraction layer.* The I/O abstraction layer is configured, selecting the ports (with their corresponding data types) that will be used by the I/O hardware abstraction. A BSW component representation (with its corresponding ports

and interfaces) of the I/O hardware abstraction is generated. The BSW component ports are connected to the application SWC ports. The I/O abstraction layer code is generated.

- *Generation of the DAP module.* The BSW module is configured automatically according to the I/O abstraction module configuration. The ECU I/O signals are mapped to DAP signals. A VPU port (introduced in Section 5.2.1) will be assigned to each DAP signal when a simulation system is run by VEOS player.

- *Configuration of the AUTOSAR operating system.* A reduced configuration of the AUTOSAR OS is used, which just takes care of the tasks to handle application SWC runnables and I/O abstraction runnables. OS tasks are created and the SWC runnables (from the application layer and the BSW abstraction layer) are mapped to them in order to define the execution context of the runnables. The task type, schedule, priority and execution order of the runnables are set for each OS task. A *NI_Task* with $1\mu s$ periodicity is created and the lowest priority is assigned to it. Also, a *COM_Task* with $1\mu s$ periodicity is created and the highest priority is assigned to it. These two tasks will be used later to instantiate COM service functions and NoC interface functions in the RTE. For completing the configuration OS events, OS alarms, OS application modes and the OS counter are configured.

- *Generation of the RTE.* Based on the application software and BSW configuration of the ECU instances a RTE implementation is generated (C code) [AUT16b]. The generated RTE, besides interconnecting the SWCs, connects the application software to the generated I/O abstraction layer and the operating system.

Before building the ECU configurations and the generation of the μECUs developed BSW modules are integrated to the ECU configurations for the completion of the BSW as shown in Figure 6.4. The developed NoC interface module and COM service modules are also integrated to each ECU configuration.

For the simulation of the μECUs, the NoC interface module provides a TCP client to connect the μECU with its corresponding network interface of the NoC simulation as explained in Section 5.1.4. It is responsible for the construction of the *data messages* sent to the NoC simulation. This is made by integrating incoming PDUs from the PDU router to a *data message* or configuring the *message status* to indicate that the *data message* is empty. The NoC interface module provides a sending queue wherein incoming PDUs from the COM module are stored by the PDU router.

The PDU router re-directs available *PDU messages* in the NoC interface module to the COM module and vice versa. The COM module takes incoming PDUs from the PDU router

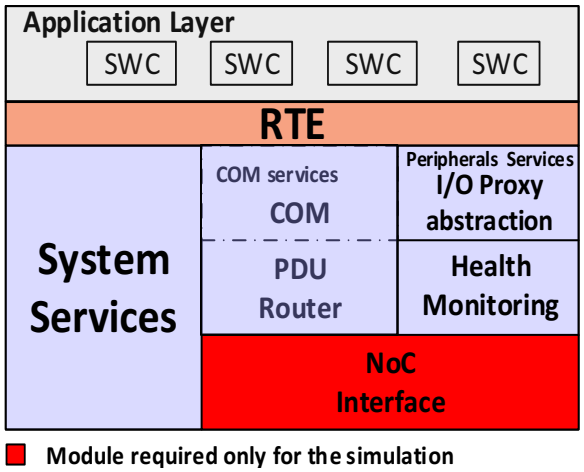

Figure 6.5: Micro-ECU Configuration with I/O Proxy Module and Health Monitoring Service

and puts their *payloads* available to the respective RTE signals. This is made by using the SWC port identifier provided in the *PDU message*. Furthermore, the COM module takes updated data on the RTE signals and constructs PDU messages. The *PDU messages* are passed to the PDU router, which puts them in the sending queue used by the NoC interface module.

The generated RTE implementation (C code) is modified manually in order to adapt the integrated COM service modules and the NoC interface module. The TCP/client connection function of the NoC interface module is allocated to the *Rte_Start* task provided by the RTE. This task is in charge of allocating and initializing system resources and communication resources used by the RTE. The NoC interface function for sending *data messages* to the TTNoC local coordinator is allocated to the *NI_Task*. This means, after an execution of $1\mu s$ a *data message* is sent from the μECU to the NoC simulation. Also, a PDU router function is allocated to the *COM_Task*. This function verifies whether there is a new *PDU* available in the NoC interface. Thus, before an execution of $1\mu s$ a new incoming PDU from the NoC simulation is passed to the PDU router. This is possible because of the configured priorities for *NI_Task* and *COM_Task*.

6.1.3.2 Configuration of an accelerated µECU

For the development of accelerated AUTOSAR μECUs, the simulated μECUs are configured based on the software architecture picture depicted in Figure 6.5. A single reduced ECU configuration without off-chip network communications, external memory access and special complex driver support, is selected for the configuration of each μECU. Additionally, an empty I/O hardware abstraction layer (without c-code implementation) is added to each ECU

configuration. Thus, BSW SWCs are generated automatically together with their interfaces that connect them to the sensor/actuator SWCs at the application layer. This allows us to develop manually the internal behavior (C-code) of these BSW SWCs to implement the proxy modules as defined in Chapter 4.

The OS of each μECU is configured and consists of OS tasks for hosting the SWC runnables that are located in the μECU. Also, OS tasks for hosting the proxy runnables are defined. Moreover, for completing the configuration OS events, OS alarms, OS application modes and OS counters are set up.

Furthermore, based on the application SWCs and the BSW configuration of the μECUs the RTE is automatically generated. This generated RTE interconnects SWCs and connects the application layer with the generated BSW SWCs and the AUTOSAR OS.

Before the generation of the simulated μECUs, the COM service modules (i.e., COM module, PDU router), an NoC interface module, the health monitoring service module and the implementation of the proxy SWCs are integrated into the BSW of the μECUs. The mentioned NoC interface module is a specific simulation module for connecting the μECU with the corresponding NIs of the NoC simulation.

The generated RTE implementation is modified manually in order to integrate the COM module, PDU router, NoC interface, the proxy modules and the health monitoring service. The integration of the COM service modules and the NoC interface module is performed in the same way as described in 6.1.3.1.

The runnables representing the internal behavior of the proxy SWCs are allocated to their specific tasks (defined previously) in the RTE implementation. Additionally, the initialization functions of the proxy implementations are allocated to the start function of the ECU State Manager [AUT16f].

In the implementation of the health monitoring service, the callback functions are used for the recovery actions. These actions are implemented according to the *RTE_Call_* $< p >$ *_* $< o >$ API [AUT16i] of the RTE in order to enable safe configuration of the AUTOSAR services as specified by the standard [AUT16a]. The initialization function of the health monitoring module is allocated to the *Rte_Start* task provided by the RTE implementation. This task is in charge of allocating and initializing system resources and communication resources used by the RTE.

After this, the ECU configurations can be built, generating the μECUs which are integrated to a single simulation system (OSA file) for being run in VEOS.

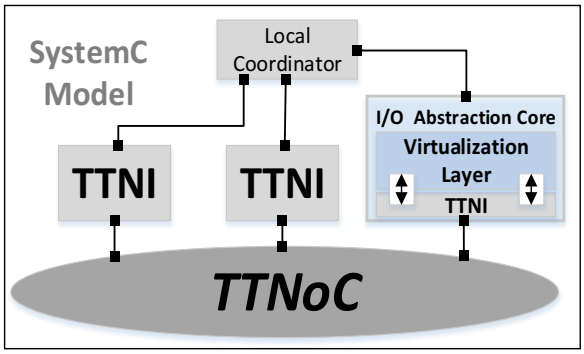

Figure 6.6: Architecture of the Input/Output Gateway Core Simulation

6.2 Implementation of the Input/Output Cores

The SystemC TTNoC simulation presented in [OO15] serves for the implementation of the defined input/output cores (i.e., I/O abstraction core and any input/output core dedicated to complex drivers). Input/output cores are developed as SystemC-based cores that run together with the TTNoC on the same SystemC simulation. For instance Figure 6.6 introduces the I/O abstraction core as a SystemC-based core in the TTNoC simulation.

As mentioned in Section 5.2.2.1, the SystemC-based TTNoC simulation model allows the configuration of the on-chip communication through a pre-defined schedule (CSV file), wherein the period and the phase of each message are set. For the implementation of an input/output core, we extend the on-chip schedule in order to provide information about the priorities of the μECUs that try to access a service of the input/output core (as explained in Chapter 4). Additionally, in order to provide redundancy at the MPSoC level, new time-triggered messages are included in the on-chip schedule to support this specific functionality as well as the configuration of the error threshold parameter ∂ defined in Section 4.3.

The virtualization layer is developed on top of the corresponding TTNI of the input/output core in order to abstract in the driver functionality from the NoC-based multicore platform implementation. Thus, this layer provides variables that map the receiving port of the TTNI directly to ECU signals handled by the input ports of the core (e.g., analog input, ADC, etc), while the data handled by the sending port of the TTNI is mapped to the ECU signals handled by the output ports of the core (e.g., PWM, analog output, etc). Since the extended on-chip schedule provides information about the messages used for supporting redundancy on a different μECU, the virtualization layer allows the mapping of multiple time-triggered messages to a single ECU output-signal. Thus, the algorithm presented in Section 4.3 is implemented by the virtualization layer to determinate the presence of a crash failure on the μECU responsible of the first assigned time-triggered message. In case a crash failure

occurred, the virtualization layer automatically switches to the second time-triggered message from the μECU wherein redundancy is provided.

Furthermore, the criticality assigned to the μECUs based on the Automotive Safety Integrity Levels (ASILs) (according to the ISO 26262 [Int11]) are used by the virtualization layer in order to provide the μECU holding the highest criticality a faster access to an input/output core service.

Environment models of the VEOS simulation generate the ECU input signals which are received by the NoC local coordinator on the SystemC-based simulation and redirected to the input/output abstraction core. Moreover, ECU output signals generated by the input/output abstraction core are redirected by the NoC local coordinator to the VEOS simulation where the FMU wrapper forwards them to the environment simulation models.

6.3 Implementation of the Memory Gateway Core Simulation

In this section the implementation of the memory gateway core together with the external memory simulation are presented. The SystemC-based TTNoC simulation framework is used to carry out the development of the gateway and the simulation of the external memory. An architecture picture of the memory gateway core and the external memory is depicted in Figure 6.7.

6.3.1 Implementation of the External Memory Simulation

For the implementation of the memory gateway core, the DRAMSim2 simulator [RCBJ11] serves for the simulation of the external memory. DRAMSim2 is a widely used cycle-accurate open-source DRAM simulator that models the memory controller, memory channels, ranks, banks and timing constraints [KYL+]. This simulator provides a flexible simulation framework based on dynamic memory models and an interface for the co-simulation with other simulations. Additionally, DRAMSim2 allows the execution of trace-based memory simulations using a *TraceBaseSim* functionality.

In order to connect DRAMSim2 with the SystemC-based TTNoC simulation a TLM/SystemC interface was established based on the work presented in [OO15]. This interface is in charge of the re-direction of incoming memory instructions from the memory gateway to their specific memory controller of the external memory and vice versa.

The configuration of the external memory is described with input files that specify the external memory initialization defining the memory technology, block size and the transaction

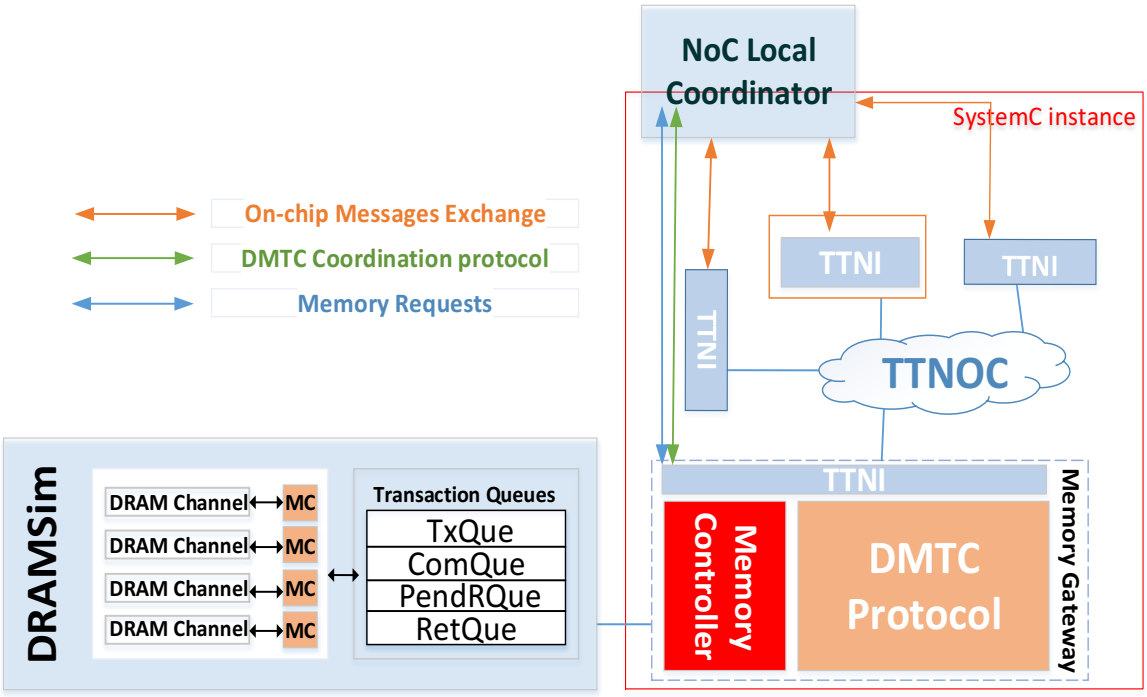

Figure 6.7: Architecture of the Memory Gateway Core Simulation

queue depth. For this, DRAMSim2 provides memory configuration templates that can be used or modified according to the simulation requirements. Additionally, the CSV TTNoC configuration file is extended to provide the configurations of the memory channels, ranks, and banks of DRAMSim2, which are loaded in the memory gateway core. Each channel has its own memory controller and its transaction queues as follows:

- The transaction queue *"TxQue"* receives and stores incoming transactions.
- The queue *"ComQue"* stores the translated commands of each transaction.
- If a read command is dispatched to the memory, then the transaction will be stored into the queue *"PendRQue"* until the data is returned.
- The queue *"RetQue"* is used to store the returned transactions.

6.3.2 Implementation of the Simulated Memory Gateway Core

For the implementation of the memory gateway core a SystemC-based host core is extended to implement and support the hierarchical DMTC protocol and algorithms described in Chapter 4. The memory gateway contains the following building blocks: the TTNI of

the gateway, the DMTC that provides the transactional memory services and the memory controller to the DRAMSim2-based external memory (see Figure 6.7).

The memory gateway core is inherited from the host processor class. Its main functionality is to initialize the external memory according to the configuration parameters of the simulation setup. Additionally, the gateway core is in charge of providing the required interface from the TTNoC to DRAMSim2. It is responsible for sending and receiving the memory transactions and mapping them from the memory transaction format to the DRAMSim2 instructions. API calls serve for invoking the memory transactions of the DMTC. Thus, transaction replies are processed correctly, since the data obtained from a read transaction is sent to the corresponding core.

In addition to the DMTC, the proposed memory gateway assumes a compositional real time memory controller [CG03] that uses a predictable arbiter responsible for scheduling memory access groups dynamically in order to guarantee the allocated bandwidth and the maximum latency bounds.

6.4 Implementation of the Off-Chip Gateway Core

In this Section the off-chip network gateway core is developed using the AUTOSAR architecture tool SystemDesk and is integrated into the AUTOSAR simulation to be run together with the μECUs.

SystemDesk allows the definition of ECU instances where each of them can have a communication controller and a connector to access a physical channel of a communication network of an automotive distributed system. For the simulation of the off-chip network gateway core, depending on the specific automotive distributed system, a number of ECU instances is defined in SystemDesk and configured as depicted in Figure 6.8 without application layer, RTE implementation or AUTOSAR OS. Each gateway core is a system core of an independent TIMEA platform.

The configuration of the off-chip gateway core is performed as follows.

- *Network Description.* A network communication description file is imported into the SystemDesk development environment. This description file contains elements such as system signals, I-Signals/I-PDUs and frames which will be used by the gateway core for the exchange of messages with other MPSoCs. Typically, the network description is based on one of the AUTOSAR description files (e.g., Database Container (DBC), Fieldbus Exchange Format (FIBEX), etc) depending of the automotive off-chip network implemented for off-chip communication.

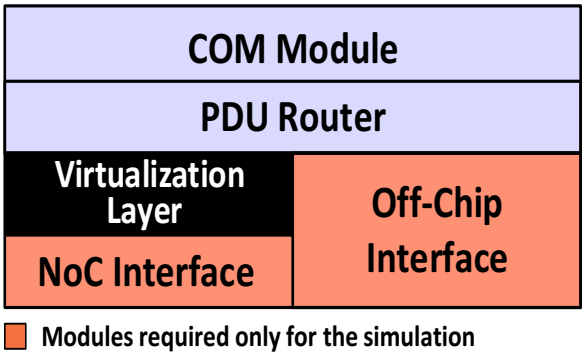

Figure 6.8: Configuration of the simulated Off-chip Gateway Core

- *ECU configuration.* A single empty ECU configuration is selected for each gateway core ECU instance. An off-chip interface module, a COM module and a PDU router are added to the ECU configuration. The off-chip interface module serves as the communication hardware abstraction to connect the gateway core with a simulated off-chip network running in VEOS while the COM module and the PDU router represent the communication service layer.

- *Off-chip Interface.* Depending on the automotive off-chip network the off-chip interface module is required to implement a specific network interface. For CAN and LIN communication protocols SystemDesk provides the specific CAN interface module and LIN interface module respectively. Theses modules allow to connect the gateway core with the simulation of a CAN bus or a LIN bus running in VEOS. For Ethernet or FlexRay [Fle05] communication protocols, SystemDesk provides the *DsIdBusIf module*, which offers idealized bus simulation with VEOS (e.g., not all the PDU properties are taken into account in a FlexRay bus simulation). Independently of the implemented off-chip network, the off-chip module is configured automatically based on the imported network description file.

- *COM module.* The implementation (C code) of the COM module is manually developed. The COM module matches NoC PDUs with off-chip PDUs and vice versa. Using SystemDesk counters are generated to handle cycle and sending times according to the time-triggered schedule of the message-based NoC and the off-chip network. Since the latencies handled by the off-chip network are orders of magnitude slower than the inter-core communication latency of the on-chip network, a Last In, First Out (LIFO) politic is employed for the message exchange between the two communication networks. This is implemented for all kind of messages, independently of their temporal behavior (i.e., rate constrained, time-triggered or event-triggered).

- *PDU Router.* The implementation of the PDU router is automatically generated by SystemDesk based on the imported network description file. As mentioned in Chapter 4, the PDU router in the off-chip network gateway core is in charge of routing the PDUs from the message-based NoC to the off-chip network and vice versa. For this, new routing tables are added to the generated PDU router for handling incoming and outcoming NoC PDUs.

Before building the ECU configurations and the generation of the off-chip gateway cores a developed NoC interface module and the visualization layer are integrated in the ECU configuration. Similar to the μECU configuration, for the simulation of the gateway core the NoC interface uses the TCP/IP protocol to connect the gateway with its corresponding network interface of the NoC simulation. PDUs coming from the virtualization layer are integrated into data messages by the NoC interface and sent to the NoC local coordinator. Additionally, the NoC interface accepts PDUs coming from the NoC simulation and makes them available to the virtualization layer.

Additionally, the virtualization layer implements the algorithm described in Section 4.4 so the μECU redundancy can be exploited by the off-chip network gateway core. This layer maps multiple PDUs coming from the NoC interface to one PDU ID. Thus, in case a crash failure is detected on the first assigned μECU, the virtualization layer switches to the second PDU from the μECU wherein redundancy is provided.

After this, the ECU configurations can be built, generating the gateway cores which are integrated into the simulation system of μECUs (OSA file) for being run in VEOS.

Chapter 7

Evaluation and Results

The proposed message-based multicore architecture for AUTOSAR with the extended BSW modules and the defined system cores is evaluated in this chapter. In the following, five evaluation scenarios are described and discussed based on the experimental results.

This evaluation serves to prove the goals specified in Chapter 1 regarding temporal predictability, enhanced performance, reduced operating system overhead and fault containment, which were expected to be obtained by the combination of the AUTOSAR software architecture with message-based on-chip networks and the integration of the defined system cores.

7.1 Evaluation of the Co-simulation Framework for AUTOSAR Message-based MPSoC Platforms

As introduced in Chapter 5, the presented co-simulation framework is implemented with two different NoC simulation models, one based on the SystemC-based TTNoC simulation and the second based on the GARNET interconnection network in GEM5. We describe the employed use case and the obtained results of the simulation performance using the two message-based NoC simulation models.

These two implementations are used later depending of the required test experiment. The GEM5 NoC implementation provides a more accurate simulation time for the analysis of the temporal predictability and the increased performance of the AUTOSAR system during the simulation scenario, while the SystemC-based NoC implementation allows us to perform experiments based on system cores for I/O services and memory abstraction services for hardware acceleration.

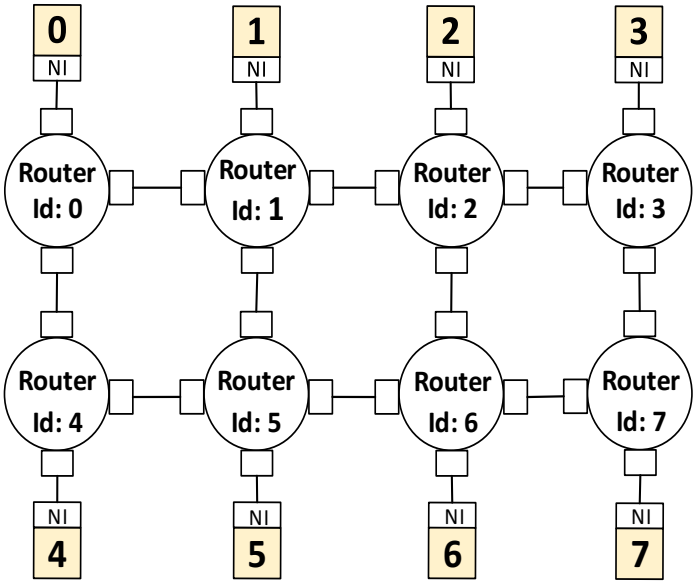

Figure 7.1: Mesh Topology

Message	Period (ms)	Phase (µs)	Sender Core	Receiver Cores	Delay (ns)
Angular speed	1	0	0	1	50
Braking force	1	1	4	1	50
Relative slip	1	2	1	3	50
Slip comparison	1	3	3	4	50

Table 7.1: SystemC-based TTNoC configuration

7.1.1 Use Case-Description

7.1.1.1 Co-simulation of VEOS and the SystemC-based TTNoC Simulation

An Anti-lock Braking System (ABS) serves as an example use case to evaluate the developed AUTOSAR multicore simulation environment. The anti-lock braking system was modelled as an AUTOSAR-based system consisting of five SWCs. The SWCs are distributed on four μECUs (μECU0, μECU1, μECU2 and μECU3), where one of them hosts two SWCs and the other three host one SWC each. Each μECU is mapped to a single core on a message-based MPSoC of 8 cores in a mesh topology (see Figure 7.1).

The scheduling of the inter-core communication through the TTNoC is summarized in Table 7.1. A period of 1*ms* is used for each message core and a unique phase is assigned to each message core. One *ms* is a reasonable period in the example, since the minimum task period in the developed ABS application is 5*ms*. Additionally, in order to inject a crash failure in our MPSoC platform we use the experimental tool ControlDesk to disable the

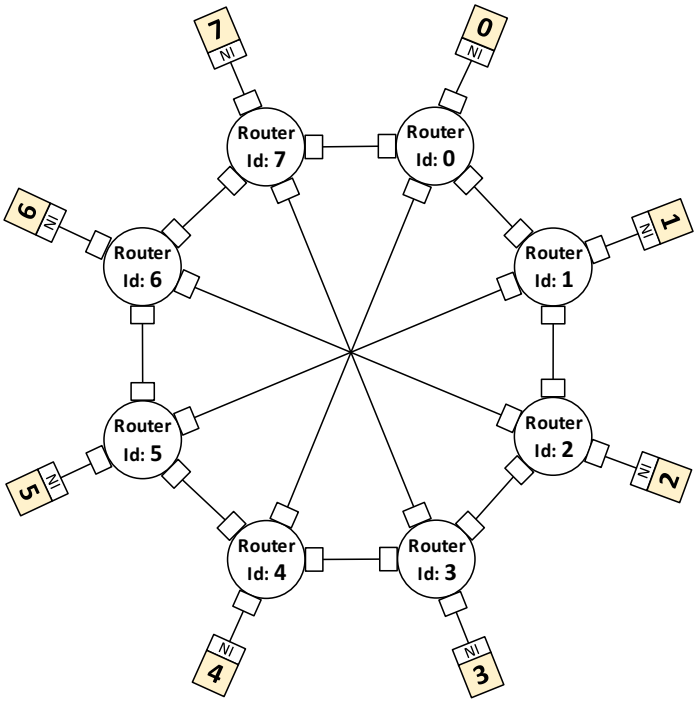

Figure 7.2: Spidergon Topology

μECU3 hosting the ABS SWC controller. In this case, we expect the ABS functionality to stop working.

Furthermore, ASMs are integrated into the AUTOSAR simulation representing the physical environment that interacts with the MPSoC, e.g., road characteristics, dynamics of the brake system hydraulic component, the physical wheel and the human braking behavior of the driver. The wheel has an initial angular speed of $70.4 rad/sec$, which is kept constant till $t = 5s$, when a hard braking is implemented by the driver.

7.1.1.2 Co-simulation of VEOS and GEM5-based NoC Simulation

The ABS use case serves for the evaluation of the co-simulation framework. The use case was modelled as an AUTOSAR-based system consisting of 4 AUTOSAR μECUs, three of them hosting one SWC each and the other one hosting two SWCs. The μECUs are implemented within an MPSoC with 8 cores in a Spidergon topology (see Figure 7.2), where each μECU is mapped to a single core in the MPSoC. This allows us to use different NoC configurations and to evaluate the ABS performance when mapping the μECUs to different cores in the MPSoC. Additionally, since the minimum task period in the ABS application is $5ms$ the accuracy of the co-simulation is guaranteed having a communication step size of $hc_i = 1\mu s$.

Additionally, in order to simulate a crash failure experiment RTE interventions are used to perform fault injections during the simulation scenario. The RTE interventions are defined by the AUTOSAR standard for the testing of ECU code allowing to access the RTE internal communication of sender-receiver and client-server interfaces in order to read and to modify the data elements and operation arguments transmitted by the interfaces. Also, the status return values of RTE API functions can be modified using RTE interventions. Thus, during a simulation SWC ports can be stimulated or error states can be injected to test the behavior of the SWCs. Using SystemDesk a RTE intervention is configured to perform the injection of a crash failure in the ABS SWC controller. In this case, we expect the ABS functionality to fail.

In our simulation scenario a hard braking is implemented by the driver having an initial angular speed of $70.4 rad/sec$.

7.1.2 Results

7.1.2.1 Co-simulation of VEOS and the SystemC-based TTNoC Simulation

Using the dSpace modular experiment and instrumentation software for ECU development, ControlDeskGeneration, the developed time-triggered multicore ABS was tested. A $22s$ co-simulation time was performed during an overall of 6 minutes of real time. The whole co-simulation system is hosted by a 64-bit Windows PC with 4GB of RAM.

Figures 7.3 and 7.4 show the results with enabled ABS and under the influence of the crash failure on μECU5. Figure 7.3 compares the angular speed of the wheel with the angular speed of the car. Figure 7.4 depicts the distance traveled by the car, showing a reduction of the braking distance of $41m$ when the ABS is enabled. Figure 7.5 shows the behavior of the relative slip suffered by the wheel without and under the ABS action.

As a last result, Figure 7.6 compares the distance traveled by the car when changing the values of the message periods in Table 7.1 to $9ms$. In this scenario the TTNoC communication has a lower frequency than the execution of the software in the μECUs. Thereby, we can observe the influence of the TTNoC on the ABS performance. A reduction of the ABS performance is visible, increasing the braking distance to $1.4m$ compared to the previous configuration of the TTNoC.

7.1.2.2 Co-simulation of VEOS and GEM5-based NoC Simulation

A simulation time of $18s$ was selected. Results are obtained using the experimental tool ControlDesk for the application behavior on the μECUs and environment models, and GEM5 for the inter-core communication behavior. Table 7.3 presents the first NoC configuration

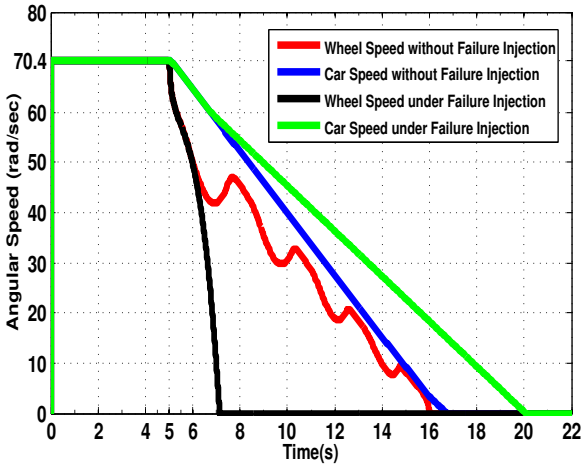

Figure 7.3: Car Speed and Wheel Speed using VEOS-SystemC Co-simulation Environment

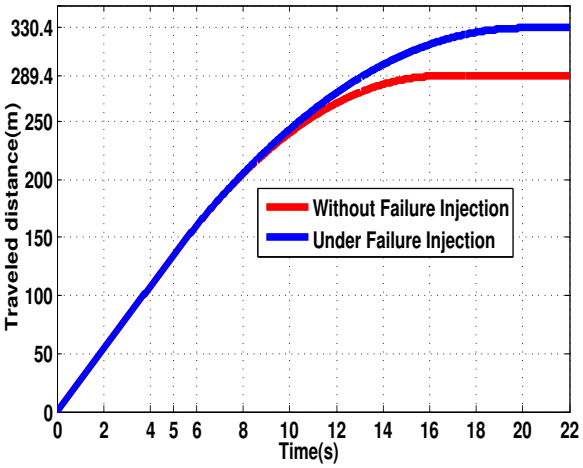

Figure 7.4: Traveled Distance

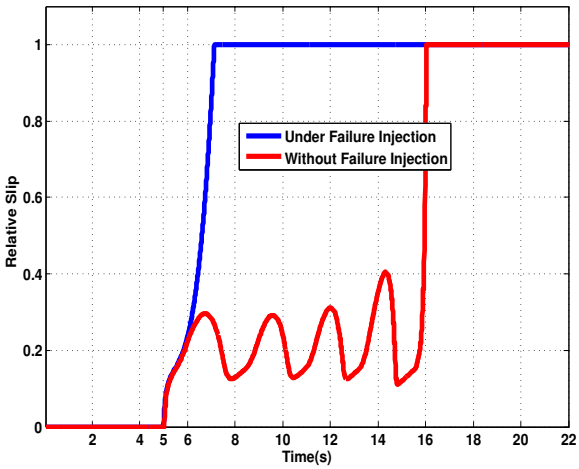

Figure 7.5: Wheel Slip

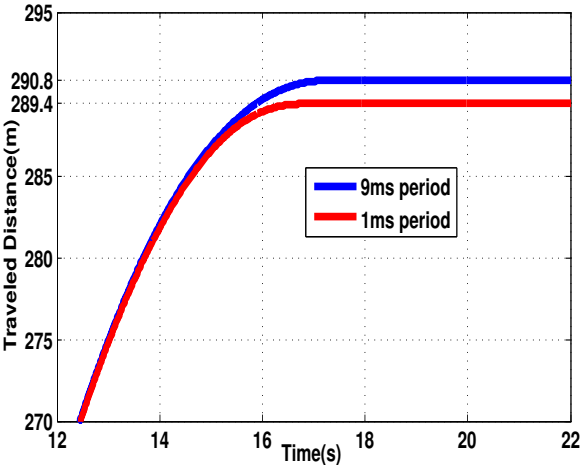

Figure 7.6: ABS Performance Comparison of two different TTNoC Configurations

NoC Configuration 1				Results		
Message	**Sender Core**	**Receiver Core**	**Temporal Configuration**	**Min. E2E Delay (ns)**	**Max. E2E Delay (ns)**	**Jitter (ns)**
Angular speed	µECU7	µECU4	RC (MINT: 3 ms)	232	264	32
Braking force	µECU5	µECU4	TT (period: 8ms)	17	17	0
Relative slip	µECU4	µECU1	RC (MINT: 3 ms)	264	272	8
Slip comparison	µECU1	µECU5	BE	217	226	9

Table 7.2: NoC Configuration 1

NoC Configuration 2				Results		
Message	**Sender Core**	**Receiver Core**	**Temporal Configuration**	**Min. E2E Delay (ns)**	**Max. E2E Delay (ns)**	**Jitter (ns)**
Angular speed	µECU0	µECU1	TT (period: 1ms)	17	17	0
Braking force	µECU4	µECU1	RC (MINT: 3 ms)	232	264	32
Relative slip	µECU1	µECU3	BE	232	272	40
Slip comparison	µECU3	µECU4	BE	217	226	9

Table 7.3: Gem5-based NoC Configuration 2

implemented for the evaluation of our ABS multicore system. In the configuration table, the sender and the receiver cores represent the cores which host the μECUs.

In order to validate the correct operation of the simulation environment, the simulation scenario is analyzed comparing an ideal scenario with a correct behavior of the ABS multicore processor and under the influence of the software crash failure on the ABS controller. Figure 7.7 compares the wheel behaviors during the simulation scenario while Figure 7.8 shows a braking distance reduction of 26m when the ABS is enabled. Also, the configuration table provides results of the communication behavior (end-to-end delay) of each message sent through the NoC. The use case simulation took an overall of 5 minutes having the VEOS simulation on a 64-bit Windows PC with 4GB of RAM, while the Gem5 simulation was executed on a 64-bit Linux PC with 2GB of RAM.

Furthermore, different NoC configurations were implemented in the mapping of the μECUs to different cores in the MPSoC in order to evaluate the influence of the NoC simulation on the ABS performance. Tables 7.2, 7.4 and 7.5 presents three different NoC configurations and their communication results. Figure 7.9 compares the angular speed of the wheel of 4 different NoC configurations, while Figure 7.10 compares the different braking distances. These results demonstrate the ability of the simulation environment to evaluate the impact of different hardware choices (e.g., different NoC configurations) on the high-level system behavior.

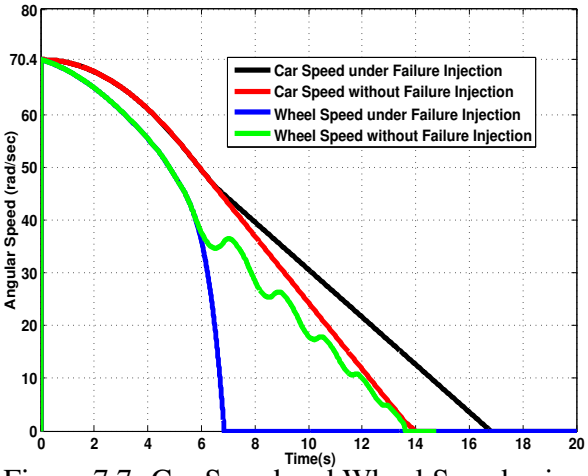

Figure 7.7: Car Speed and Wheel Speed using VEOS-GEM5 Co-simulation Environment

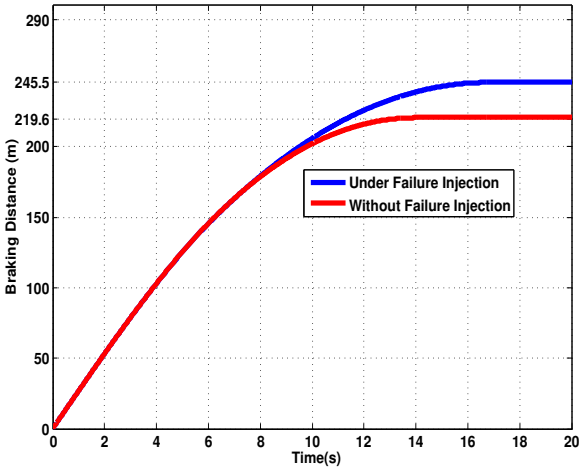

Figure 7.8: Braking Distance

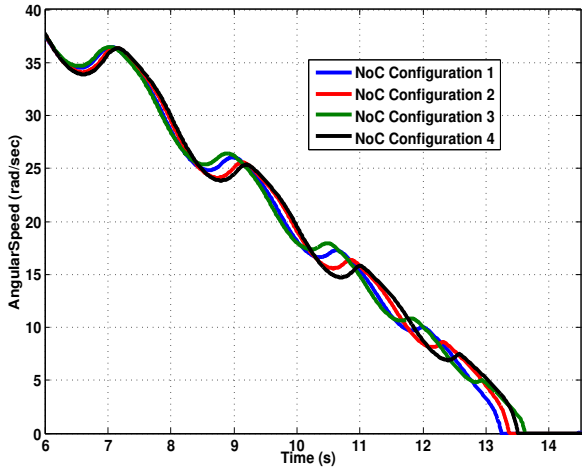

Figure 7.9: Wheel Speed with different NoC Configurations

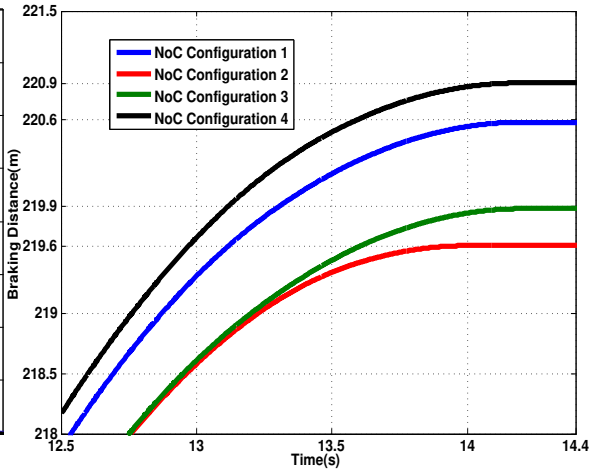

Figure 7.10: Braking Distance with different NoC Configurations

NoC Configuration 3				Results		
Message	**Sender Core**	**Receiver Core**	**Temporal Configuration**	**Min. E2E Delay (ns)**	**Max. E2E Delay (ns)**	**Jitter (ns)**
Angular speed	µECU0	µECU3	RC (MINT: 3 ms)	232	264	32
Braking force	µECU6	µECU3	RC (MINT: 3 ms)	232	272	40
Relative slip	µECU3	µECU2	TT (period: 3ms)	17	17	0
Slip comparison	µECU2	µECU6	BE	217	226	9

Table 7.4: NoC Configuration 3

NoC Configuration 4				Results		
Message	**Sender Core**	**Receiver Core**	**Temporal Configuration**	**Min. E2E Delay (ns)**	**Max. E2E Delay (ns)**	**Jitter (ns)**
Angular speed	µECU3	µECU4	BE	217	264	47
Braking force	µECU1	µECU4	RC (MINT: 3 ms)	232	264	32
Relative slip	µECU4	µECU6	BE	249	295	46
Slip comparison	µECU6	µECU1	TT (period: 12ms)	32	32	0

Table 7.5: NoC Configuration 4

7.1.3 Discussion

The capability of the presented co-simulation environment was evaluated presenting an ABS use case. The influence of the message-based NoC on the AUTOSAR-based system was analyzed. The realistic automotive use case together with the virtual validation scenario demonstrates the ability of the framework for early validation of applications as well as its capability to analyze the performance and the real-time behavior of AUTOSAR applications on different multicore platforms with message-based NoCs under fault interjection experiments.

The presented simulation environment allows the connection with the ControlDesk Generation tool for experimental tests. The simulation can be controlled, e.g., resumed and stopped at any time. Specific simulation steps can be configured and also a limited simulation time is set up. Information from the μECUs can be obtained during execution-time. μECUs can be turned on/off at any time. Furthermore, the simulation results are obtained graphically and numerically.

Additionally, the log files from both message-based NoC simulations provide temporal information of the communication behavior of the network. In the co-simulation with the GEM5-based NoC model the messages configured with a time-triggered temporal behavior hold a 0 communication jitter as expected. This is due to the temporal behavior of the NoC for the injection of this kind of messages in contrast to the rate-constrained and best effort messages whose communication jitters are different to 0.

7.2 Evaluation of Performance and Fault Containment in AUTOSAR Micro-ECUs

The co-simulation environment consisting of VEOS and the GEM5 message-based NoC simulation is selected for the implementation of the use case for the evaluation of the performance and the fault containment in the μECUs. This is due to the ability of the

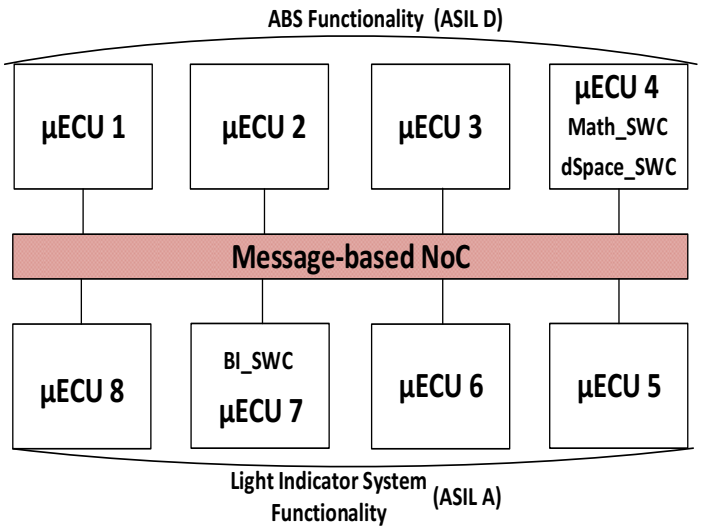

Figure 7.11: Automotive Use Case for Fault Containment Evaluation

NoC in GEM5 to configure different timing models (i.e., rate-constrained, best-effort and time-triggered) for the message exchange between the μECUs.

7.2.1 Use Case-Description

As a realistic automotive use case, we employ the ABS consisting of five AUTOSAR SWCs distributed on 4 AUTOSAR μECUs and a Light Indicator System (LIS) of four SWCs distributed on 4 μECUs. Both application systems are implemented within a multicore platform of 8 cores in Spidergon topology, where each μECU is mapped to a single core in the MPSoC.

Figure 7.11 presents the described use case. The μECU4 contains an AUTOSAR SWC performing the controller of the ABS functionality. Additionally, SWC redundancy is implemented in the μECU4. Thus, two different SWC implementations of the controller are used, one based on the open source ABS simulink implementation of MathWorks [Matml] adapted to an AUTOSAR SWC (**Math_SWC**), and one obtained from the dSpace solution for an AUTOSAR ABS implementation (**dSpace_SWC**). We set the threshold parameters for crash and permanent value failure recognition on μECU4 with a granularity of $\rho = 3$, $\phi = 6$, $\varphi_i = 6$, $\gamma_i = 12$, $\kappa_i = 50$ and $c_i = 50$. Furthermore, to analyze the impact of a babbling idiot failure on our MPSoC platform a SWC (**BI_SWC**) is implemented on the μECU7 of the LIS functionality sending untimely messages to the network.

Additionally, ASMs are integrated into the AUTOSAR simulation representing the physical environment that interacts with the MPSoC as described in the previous section (see Section 7.1.1). In the presented simulation scenario, a hard braking is implemented by the

NoC Configuration				Results					
				Without Fault Injection			Under Fault Injection		
ID	Sender Core	Receiver Core	Temporal Configuration	Min. E2E Delay (ns)	Max. E2E Delay (ns)	Jitter (ns)	Min. E2E Delay (ns)	Max. E2E Delay (ns)	Jitter (ns)
1	µECU4	µECU1	TT (period: 1ms)	17	17	0	17	17	0
2	µECU8	µECU5	RC (MINT: 3 ms)	232	264	32	232	264	32
3	µECU1	µECU2	TT (period: 1ms)	17	17	0	17	17	0
4	µECU2	µECU4	TT (period: 1ms)	32	32	0	32	32	0
5	µECU2	µECU3	TT (period: 1ms)	17	17	0	17	17	0
6	µECU3	µECU4	TT (period: 1ms)	17	17	0	17	17	0
7	µECU5	µECU6	BE	246	272	26	246	286	40
8	µECU6	µECU5	BE	264	284	20	264	284	20
9	µECU6	µECU7	BE	258	286	28	258	304	46
10	µECU1	µECU4	TT (period: 1ms)	17	17	0	17	17	0
11	µECU7	µECU8	RC (MINT: 3 ms)	234	262	28	234	275	41
12	µECU7	µECU8	RC (MINT: 3 ms)	248	289	41	248	298	50

Table 7.6: NoC Configuration in Timing Failure Experiment

driver while the car has an initial speed of $88 km/h$. Moreover, for the evaluation of the TIMEA platform and the presented fault tolerance mechanisms, RTE interventions are used to perform fault injections during the simulation scenario.

In this scenario the ABS functionality represents a safety critical system (ASIL D) while the LIS (ASIL A) is less important in terms of safety.

7.2.2 Results

7.2.2.1 Timing Failure Experiment

For analyzing the behavior of the AUTOSAR message-based multicore system under a timing failure occurrence, RTE interventions are employed to delay the time-triggered execution of the SWCs which send the messages through the NoC, for both, the ABS and the LIS. In this fault injection experiment we use the NoC configuration presented in table 7.6 for the setting of the network. In this NoC configuration messages exchanged by the ABS are set as time-triggered messages since this functionality is safety critical, while messages exchanged by the LIS are set as best-effort and rate-constrained messages.

Additionally, table 7.6 establishes a comparison between the resulting jitter of the NoC messages without faults and in the presence of a timing failure. These communication results demonstrate how time-triggered messages are unaffected by the delay fault occurrence unlike best-effort and rate-constrained messages whose communication jitter is affected.

NoC Configuration 1				Results					
				Without Fault Injection			Under Fault Injection		
ID	Sender Core	Receiver Core	Temporal Configuration	Min. E2E Delay (ns)	Max. E2E Delay (ns)	Jitter (ns)	Min. E2E Delay (ns)	Max. E2E Delay (ns)	Jitter (ns)
1	µECU4	µECU1	TT (period: 1ms)	17	17	0	17	17	0
2	µECU8	µECU5	RC (MINT: 3 ms)	232	264	32	232	293	61
3	µECU1	µECU2	TT (period: 1ms)	17	17	0	17	17	0
4	µECU2	µECU4	TT (period: 1ms)	32	32	0	32	32	0
5	µECU2	µECU3	TT (period: 1ms)	17	17	0	17	17	0
6	µECU3	µECU4	TT (period: 1ms)	17	17	0	17	17	0
7	µECU5	µECU6	BE	246	272	26	246	413	167
8	µECU6	µECU5	BE	264	284	20	264	435	171
9	µECU6	µECU7	BE	258	286	28	258	489	231
10	µECU1	µECU4	TT (period: 1ms)	17	17	0	17	17	0
11	µECU7	µECU8	RC (MINT: 3 ms)	234	262	28	234	319	85
12	µECU7	µECU8	RC (MINT: 3 ms)	248	289	41	248	322	74
13	µECU7 BI_SWC	µECU8	BE				**Fault Injection**		
		µECU1							
		µECU2							
		µECU3							
		µECU4							
		µECU5							
		µECU6							

Table 7.7: NoC configuration 1 for Babbling Idiot Experiment

7.2.2.2 Babbling Idiot Experiment

As a second experiment, we run our simulation scenario having **BI_SWC** on the μECU7, so the response of the platform under a babbling idiot failure is tested. The NoC configuration 1 presented in table 7.7 is used for the configuration of the multicore platform. Additionally, we run our simulation use case using the NoC configurations 2 and (see tables 7.8 and 7.9), where messages exchanged by the ABS functionality are set as rate-constrained messages and best-effort messages respectively, so a comparison can be established.

Figures 7.12, 7.13 and 7.14 represent the distance traveled by the car and the wheel slip with and without babbling idiot failures using each one of the NoC configurations. Figure 7.12 shows how the behavior of the wheel slip stays equal and the distance traveled by the car was not affected, which is due to the time-triggered behavior of the messages sent by the ABS functionality. In contrast, Figures 7.13 and 7.14 exhibit a difference between the curves when the babbling idiot failure is injected. In Figure 7.13 the braking distance is increased by $2.03m$ ($\backsimeq 1\%$), while in Figure 7.14 the distance presents a significant increase of $5.37m$ ($\backsimeq 2.5\%$). Furthermore, tables 7.7, 7.8 and 7.9 compare the resulting maximum and minimum delays in all three NoC configurations with and without babbling idiot injection.

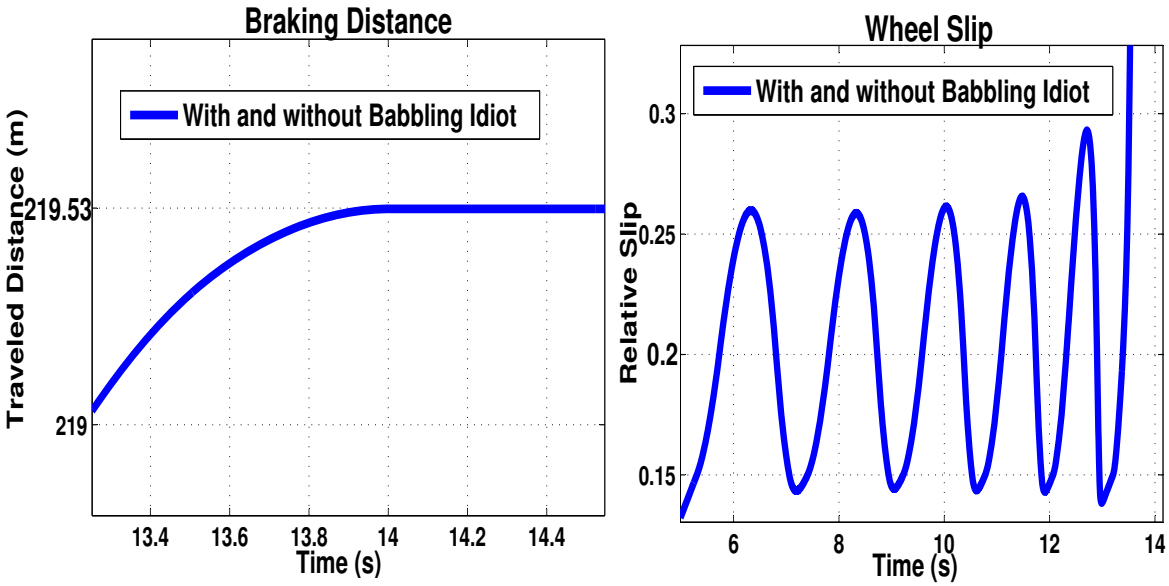

Figure 7.12: Braking Distance and Wheel Slip with NoC Configuration 1

As presented in these tables, NoC configuration 1 shows no difference between maximum and minimum delays on messages sent by the ABS functionality, so a jitter of 0 is kept for all of them. This is not the case for NoC configurations 2 and 3 where the calculated jitter is increased significantly.

7.2.2.3 Omission/Crash Failure Experiment

We run our simulation scenario using RTE interventions for the injection of consecutive omission failures in the ABS SWC controller (**Math_SWC**). In this case we expect the monitoring service to recognize the omission failures by receiving error statuses from the RTE function that triggers this SWC and thus, to determine the presence of a crash failure so **dSpace_SWC** will be activated for fault recovery. The experiment was run with and without having health monitoring service on μECU4. The omission failure injection was initiated after 4s of simulation time.

Figure 7.15 illustrates the comparison of the braking distance behavior when no value failure is injected (curve 1), under fault injection (curve 2) and without health monitoring service (curve 3). This Figure shows how the ABS functionality remains operational under a crash failure occurrence when having the monitoring service module. However, a reduction of $\backsimeq 0.1\%$ $(0.22m)$ in the ABS performance is visible in curve 3 compared with curve 2, which is resulting from the delay in response of the fault recognition algorithm.

NoC Configuration 2				Results					
				Without Fault Injection			Under Fault Injection		
ID	Sender Core	Receiver Core	Temporal Configuration	Min. E2E Delay (ns)	Max. E2E Delay (ns)	Jitter (ns)	Min. E2E Delay (ns)	Max. E2E Delay (ns)	Jitter (ns)
1	µECU4	µECU1	RC (MINT: 3 ms)	17	46	29	17	317	300
2	µECU8	µECU5	RC (MINT: 3 ms)	17	46	29	17	373	356
3	µECU1	µECU2	RC (MINT: 3 ms)	17	52	35	17	347	330
4	µECU2	µECU4	RC (MINT: 3 ms)	32	68	36	32	332	300
5	µECU2	µECU3	RC (MINT: 3 ms)	17	52	35	17	386	369
6	µECU3	µECU4	RC (MINT: 3 ms)	17	46	29	17	366	349
7	µECU5	µECU6	BE	138	172	34	138	613	475
8	µECU6	µECU5	BE	146	184	38	146	795	649
9	µECU6	µECU7	BE	184	196	12	184	589	405
10	µECU1	µECU4	RC (MINT: 3 ms)	17	46	29	17	317	300
11	µECU7	µECU8	RC (MINT: 3 ms)	17	52	35	17	319	302
12	µECU7	µECU8	RC (MINT: 3 ms)	17	68	51	17	362	345
13	µECU7 BI_SWC	µECU8 µECU1 µECU2 µECU3 µECU4 µECU5 µECU6	BE				**Fault Injection**		

Table 7.8: NoC Configuration 2 for Babbling Idiot Experiment

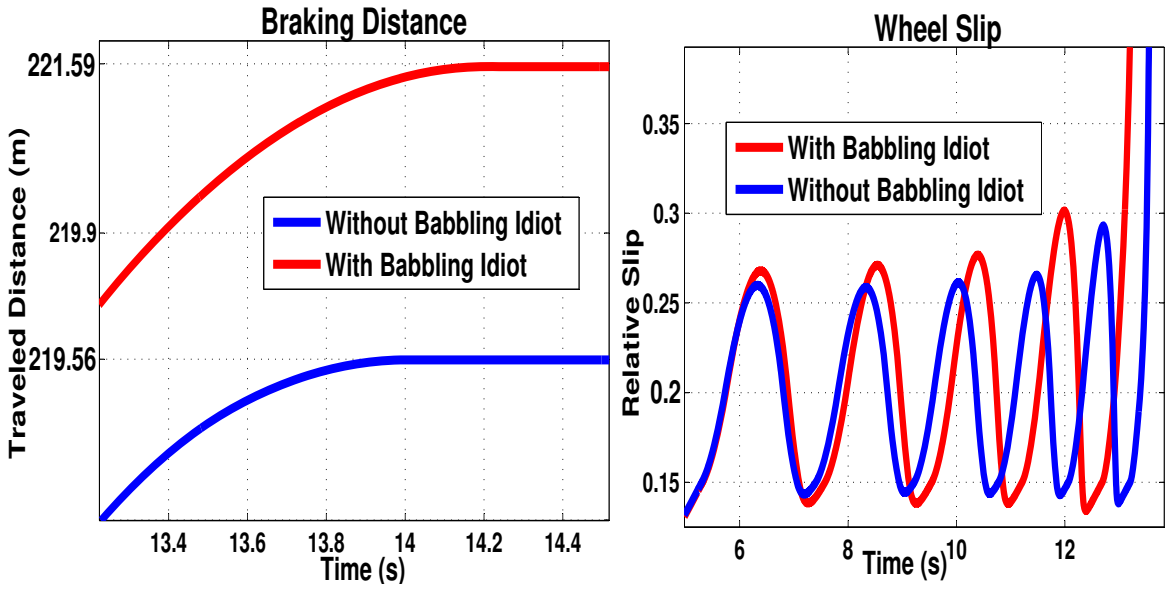

Figure 7.13: Braking Distance and Wheel Slip with NoC Configuration 2

NoC Configuration 3				Results					
				Without Fault Injection			Under Fault Injection		
ID	Sender Core	Receiver Core	Temporal Configuration	Min. E2E Delay (ns)	Max. E2E Delay (ns)	Jitter (ns)	Min. E2E Delay (ns)	Max. E2E Delay (ns)	Jitter (ns)
1	µECU4	µECU1	BE	138	176	38	138	917	779
2	µECU8	µECU5	RC (MINT: 3 ms)	17	26	9	17	173	156
3	µECU1	µECU2	BE	32	62	30	32	632	600
4	µECU2	µECU4	BE	64	88	24	64	587	523
5	µECU2	µECU3	BE	32	52	20	32	648	616
6	µECU3	µECU4	BE	17	46	29	17	936	919
7	µECU5	µECU6	BE	64	72	8	64	832	768
8	µECU6	µECU5	BE	52	84	32	52	795	743
9	µECU6	µECU7	BE	32	96	64	32	917	885
10	µECU1	µECU4	BE	17	46	29	17	1089	1072
11	µECU7	µECU8	RC (MINT: 3 ms)	17	22	5	17	119	102
12	µECU7	µECU8	RC (MINT: 3 ms)	17	28	11	17	162	145
13	µECU7 BI_SWC	µECU8	BE				**Fault Injection**		
		µECU1							
		µECU2							
		µECU3							
		µECU4							
		µECU5							
		µECU6							

Table 7.9: NoC Configuration 3 for Babbling Idiot Experiment

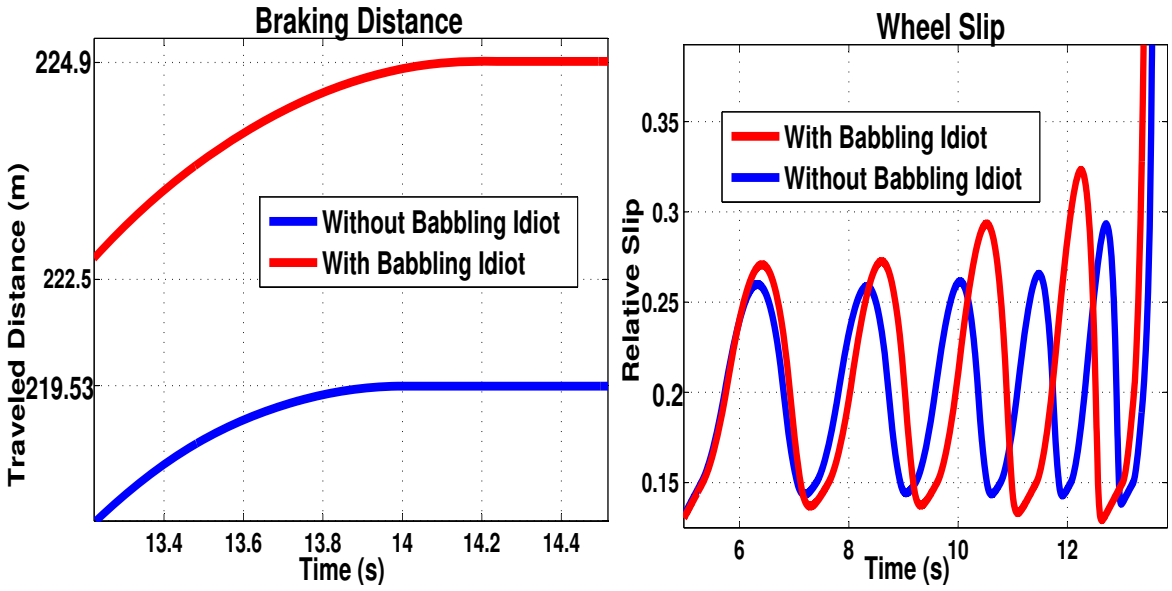

Figure 7.14: Braking Distance and Wheel Slip with NoC Configuration 3

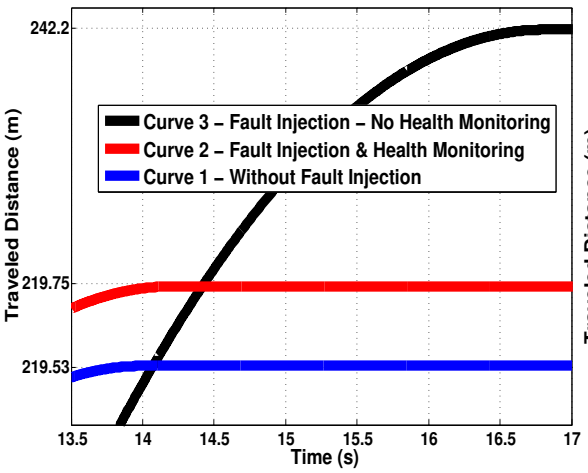

Figure 7.15: Braking Distance in Omission/Crash Failure Experiment

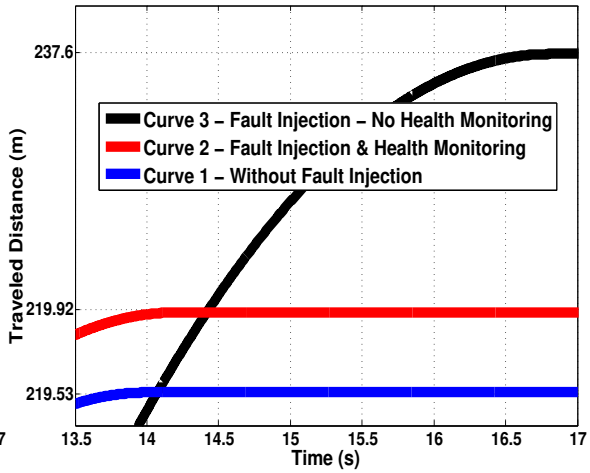

Figure 7.16: Braking Distance in Permanent Value Failure Experiment

7.2.2.4 Value Failure Experiment

In order to test the response of the message-based MPSoC platform to a permanent value failure scenario, we use RTE interventions to inject value failures on **Math_SWC**. The value failure consists of a message value which is not compliant to the data constraint element of the AUTOSAR message.

Just as for the crash failure injection, in this case the expected result is the monitoring service triggering the **dSpace_SWC** once the parameter ϕ is overcome to continue providing the ABS functionality. In Figure 7.16, curves 1 and 2 depict the distance traveled by the car without faults and in the presence of a permanent value failure. As displayed in this Figure, the ABS performance in curve 2 has decreased by $\backsimeq 0.2\%$ $(0.39m)$ compared with curve 1, but still offering a significant difference of the braking distance in comparison with curve 3 when no health monitoring functionality is provided under a permanent value failure occurrence.

7.2.2.5 Evaluation of the Operating System Overhead

In order to measure the impact of the introduced health monitoring service on the operating system overhead the number of task invocations realized by the AUTOSAR operating system in μECU4 is compared. This parameter represents how many times a single task is triggered by the operating system during the whole simulation scenario. We expect a considerable increase of the task invocations when using the health monitoring service since new tasks are handled by the operating system to trigger this functionality.

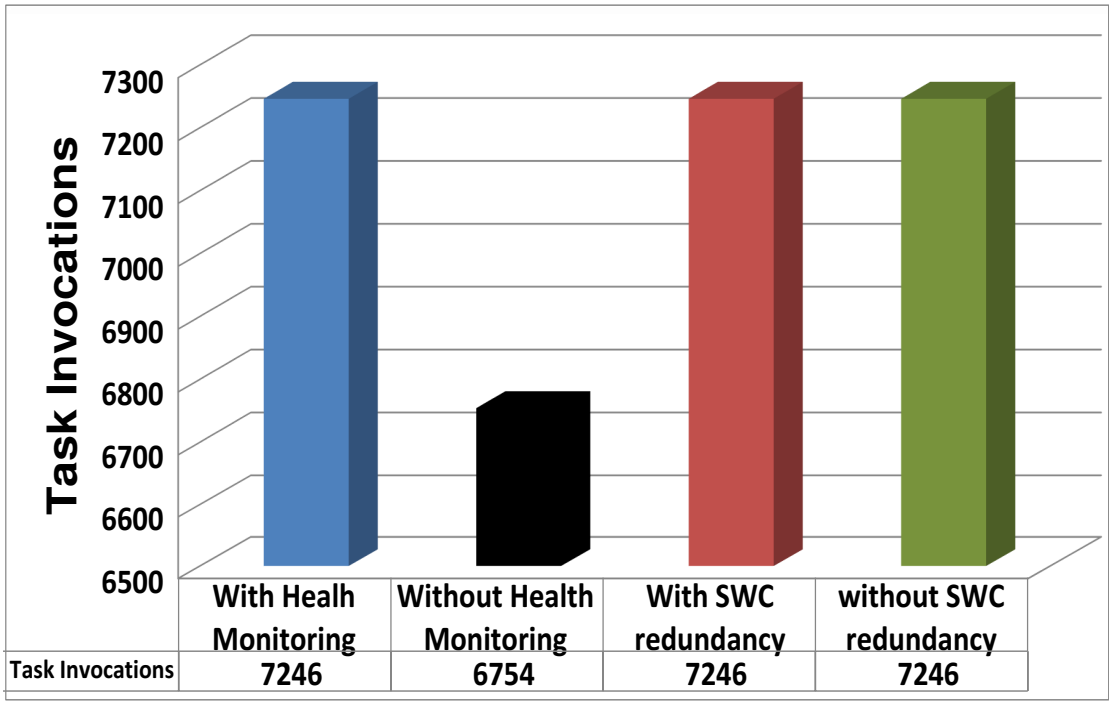

Figure 7.17: Comparison of Task Invocations in AUTOSAR micro-ECUs

Figure 7.17 compares the number of task invocations in μECU4 with the health monitoring implementation, without health monitoring, with SWC redundancy and without SWC redundancy. This figure shows an increase of $\backsimeq 6.8\%$ of the operating system overhead when a health monitoring service is provided by the BSW in the μECU, which can be justified since the reliability of the system has improved significantly. Additionally, the SWC redundancy does not affect the operating system overhead as expected.

7.2.3 Discussion

We presented an evaluation of the performance and fault containment of the AUTOSAR μECUs in the TIMEA platform. A realistic automotive simulation scenario and multiple fault injection experiments were carried out using a co-simulation environment for AUTOSAR message-based multicore systems.

The obtained results demonstrate how μECUs running safety critical automotive functionalities are isolated from faults occurring on a different core due to the intrinsic fault isolation property of the network, since these functionalities are mapped to time-triggered messages with a static communication schedule.

Furthermore, the reliability of the AUTOSAR multicore system has been improved considerably through the integration of the health monitoring service and the SWC redun-

dancy. Software failures on the value domain can be recognized and backup solutions can be activated to overtake the functionality provided by the failed AUTOSAR SWC.

7.3 Evaluation of Performance with an I/O Gateway Core

In this section we perform an experimental use case to evaluate the performance of the TIMEA platform when using the implemented I/O gateway core. Accordingly, the co-simulation environment consisting of VEOS and the SystemC-based TTNoC simulation serves for the examination.

7.3.1 Use Case-Description

For the evaluation, we define a use case consisting of a TIMEA platform of 8 cores in mesh topology. An ABS functionality is distributed over three AUTOSAR μECU*s*, two of them hosting 2 SWCs and one of them hosting just one. Figure 7.18 represents an architecture picture of the use case. In our simulation scenario, a hard braking is implemented by the driver with an initial speed of $88 km/h$. Additionally, we use RTE interventions to perform fault injections in order to change the values of the data prototypes handled by the AUTOSAR SWC ports. Thus, the recovery solutions provided by the health monitoring service together with the I/O proxy module and the I/O gateway core can be tested.

As depicted in Figure 7.18, the μECU1 has a sensor SWC which interacts with the I/O abstraction core receiving an ECU signal from the velocity sensor through an ADC input port. Also, μECU3 hosts an actuator SWC responsible for computing the control signal of the ABS functionality (**dSpace_SWC** in Figure 7.18), which is sent by the NoC to the I/O abstraction core. We inject value failure in the error detection block which consists of an unexpected value for the ABS control signal exceeding the limits assigned by the data constraint element. As a recovery action the health monitoring service in μECU3 must resume tasks for activating a replica of the actuator SWC (**Math_SWC** in Figure 7.18) that is located in the same μECU in order to use redundancy at the μECU level. Additionally, a Breaking Light Indicator (BLI) functionality is performed by one SWC hosted by the μECU4. In order to provide redundancy at the MPSoC level, a replica of the actuator SWC for the ABS control signal is also allocated to μECU4 (**dSpace_SWC'** in Figure 7.18).

Furthermore, μECU*s* 5 and 6 represent automotive functionalities which require a FFT service each for a certain time, which is made available to the μECU*s* through a dedicated input/output core (FFT core in Figure 7.18). The FFT application is based on [FFTft], which performs a 16-point FFT computation of the input signal. Different priorities are assigned to

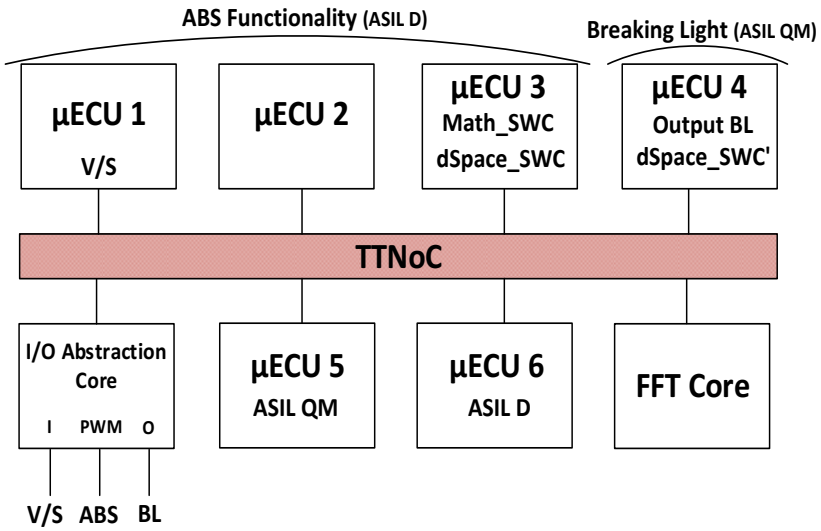

Figure 7.18: Automotive Use Case for Evaluation of the I/O Gateway Core

each μECU in order to avoid conflicts in accessing the FFT core. Since μECU6 possesses a higher criticality level (ASIL D) than μECU5 (ASIL QM), this μECU is pre-configured at compile time in the virtualization layer of the FFT core with a higher priority as explained in Chapter 4. Thus, the values obtained from a specific μECU are queued by the virtualization layer in a 16 point buffer before passing them to the FFT application. The virtualization layer forwards the buffer that corresponds to the higher priority in case two μECU*s* require the FFT service at the same time. For this, trace files allocated in μECU*s* 5 and 6 are used to provide the signals to be sent to the FFT core.

As shown in Figure 7.18, the I/O abstraction core provides one ADC input pin for receiving the velocity sensor signal, a PWM pin for forwarding the ABS control signal and an output pin connected to the BLI functionality. We configure the parameters ∂ and ς (defined in Section 4.3) in the virtualization layer of the I/O abstraction core with a granularity of $\partial = 3$ and $\varsigma = 6$. This seems to be a reasonable selection of both threshold parameters since the minimum task period in the presented use case is $5ms$.

7.3.2 Results

The NoC configuration implemented for the TIMEA platform is presented in table 7.10. Also, this table provides the resulting latencies for each message sent through the TTNoC. The simulated TTNoC does not accept the sending of simultaneous messages at the same time through the NoC, having a processing time of $50ns$ from sender core to receiver core. For this, phases were selected with a difference of more than $50ns$ to avoid delays because of contention in the network.

ID	Period (ms)	Phase (µs)	Sender Core	Receiver Core	Delay (ns)
1	1	2	I/O Core	µECU1	50
2	1	54	µECU1	µECU2	50
3	1	106	µECU2	µECU3	50
4	1	158	µECU2	µECU4	50
5	1	210	µECU3	I/O Core	50
6	1	262	µECU4	I/O Core	50
7	1	314	µECU4	I/O Core	50
8	1	366	µECU5	FFT Core	50
9	1	418	µECU6	FFT Core	50

Table 7.10: TTNoC Configuration for I/O Gateway Core Evaluation

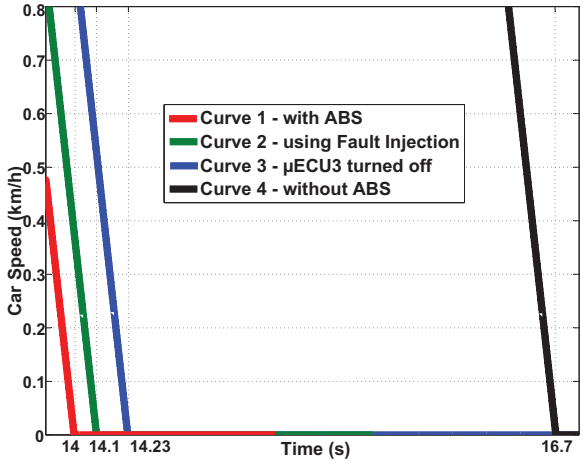

Figure 7.19: Comparison of Car Speeds in I/O Gateway Core Experiment

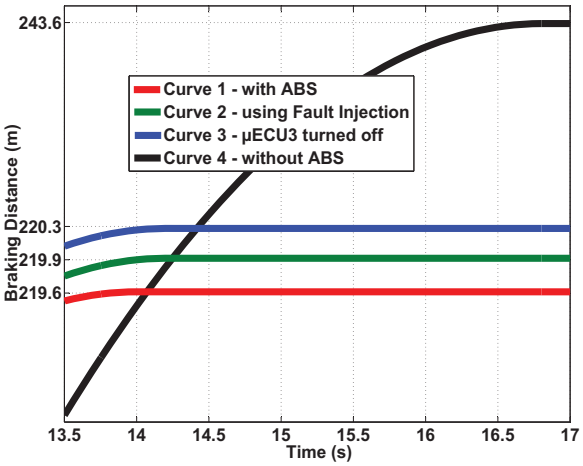

Figure 7.20: Comparison of Braking Distances in I/O Gateway Core Evaluation

Figure 7.19 compares the behavior of the vehicular speed, while Figure 7.20 illustrates the braking distance behavior. In both figures, curve 1 shows the braking behavior of the car when no fault is injected. Curve 2 shows the braking behavior when injecting a value failure on the **dSpace_SWC** of the μECU3 to test the recovery action (activating **Math_SWC**), while curve 3 represents the braking behavior when disabling the μECU3 to simulate a crash failure of the μECU, which means requesting the virtualization layer of the I/O abstraction core to take the ABS control signal from another core where redundancy is provided (**dSpace_SWC'** on μECU4 in this case). As shown in Figure 7.20, the ABS performance in curve 2 and 3 has decreased by $\backsimeq$ 1.3% (0.3m) and $\backsimeq$ 2.8% (0.7m) respectively compared with curve 1 but still offering a significant difference of the braking distance in comparison with curve 4 when no ABS functionality is provided.

	Task Invocations		
Core	With I/O Core	Without I/O Core	Health Monitoring Service
µECU1	4377	7450	----------
µECU3	7576	9327	897
µECU4	7346	12456	----------

Table 7.11: Comparison of Task Invocations in I/O Gateway Core Evaluation

Core	FFT initial Request Time (µs)	Proccessing Time (µs)	Ending Time (µs)
µECU5	5000366	32000	5032366
µECU6	5000418	17000	5017418
µECU5	10000366	17000	10017366
µECU6	11000418	17000	11017418
µECU5	15010366	22000	15032366
µECU6	15000418	17000	15017418

Table 7.12: FFT Timing Accesses

In order to measure the impact of the OS overhead resulting from the implementation of the I/O abstraction core we also run our simulation scenario keeping the I/O functionalities on the BSW of each μECU. Table 7.11 compares the number of task invocations realized by the OS with and without a dedicated input/output core. As shown in table 7.11, in μECU1 the OS overhead decreases by 27.82% when the I/O abstraction core is implemented, while μECU3 and μECU4 exhibit an overhead reduction of 18.77% and 41.02% respectively. In μECU3 the integrated health monitoring service represents 9.84% of the OS overhead, which can be justified since the reliability of the system has being improved significantly.

Furthermore, μECU5 and μECU6 require the service provided by the FFT core at three different times during our simulation scenario. Table 7.12 presents the latency results obtained by the interaction between the μECU*s* and the FFT core. Both μECU*s* access the FFT core using the TTNoC at their pre-defined time slots. The results presented in table 7.12 demonstrate how the μECU6 with the highest criticality level always needs the minimum time (17*ms*) for processing the FFT.

7.3.3 Discussion

We presented an evaluation of performance of the time-triggered multicore architecture for AUTOSAR based on input/output cores. Costly I/O functionalities of the AUTOSAR BSW I/O abstraction layer are delegated to a dedicated input/output core (I/O abstraction core) which is made available to the AUTOSAR SWCs located on different μECU*s* through the NoC. Additionally, an FFT core represents an example for a complex driver input/output core.

The results demonstrate how the OS overhead of the μECU*s* is reduced significantly when the I/O functionalities are delegated to dedicated input/output cores. Furthermore, the presented simulation scenario shows how the performance of the ABS functionality is preserved under the occurrence of core crash failures due to the integrated I/O abstraction core which allows the extension of SWC redundancy on different AUTOSAR μECUs.

7.4 Evaluation of Performance with an Off-chip Network Gateway Core

In this section we carry out an experimental use case to evaluate the performance of the TIMEA platform when using the implemented off-chip network gateway core. The co-simulation environment consisting of VEOS and the GEM5-based NoC simulation model serves to run the use case simulation scenario.

7.4.1 Use Case-Description

The ABS use case consisting of 5 SWCs was distributed on 5 different μECUs, hosting one SWC each. Likewise, the LIS and the BLI use cases were distributed on 5 and 2 μECUs, hosting one SWC each. Figure 7.21 represents an architecture picture of the developed use case. The μECUs are distributed on two TIMEA multicores of 8 cores in Spidergon topology where both MPSoCs are provided with an implementation of the off-chip network gateway core supporting TT-CAN bus communication for the interaction between each other. Moreover, the MPSoC1 holds a replica of the μECU4 (μECU4´) of the ABS functionality while the MPSoC2 is holding a replica of the μECU8 (μECU8´) of the ABS functionality. Parameters ∂_1, ∂_2 and ς_1, ς_2 are configured with a granularity of $\partial = 3$ and $\varsigma = 6$ in the virtualization layer of the off-chip gateway cores 1 and 2 respectively.

We test the developed distributed system in a simulation scenario where the driver performs a hard braking having an initial speed of the car of $88km/h$. Fault injections are

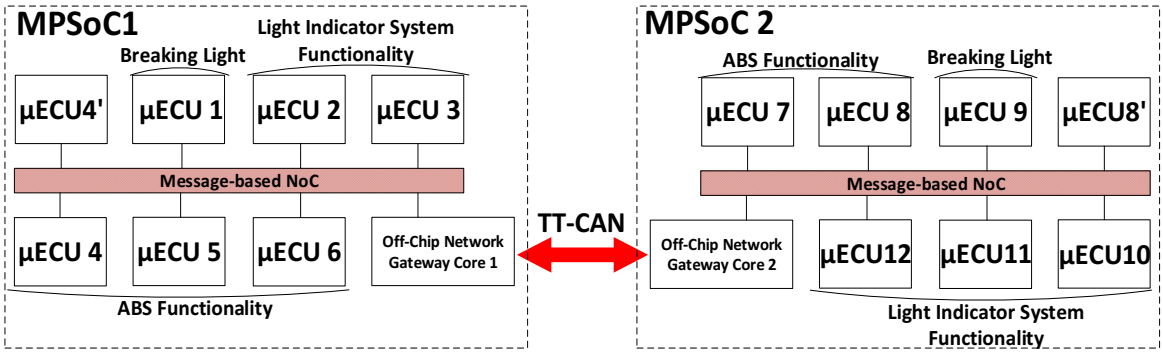

Figure 7.21: Distributed Automotive Use Case for Off-Chip Network Gateway Core Evaluation

ID	NoC Configuration MPSoC1			NoC Configuration MPSoC2		
	Sender Core	Receiver Core	Temporal Configuration	Sender Core	Receiver Core	Temporal Configuration
1	µECU5	µECU6	TT (period: 1ms)	Off-Chip Core2	µECU7	TT (period: 1ms)
2	µECU6	µECU4	TT (period: 1ms)	µECU7	µECU8	TT (period: 1ms)
2	µECU6	µECU4'	TT (period: 1ms)	µECU7	µECU8'	TT (period: 1ms)
3	µECU4	Off-Chip Core1	TT (period: 1ms)	µECU8	Off-Chip Core2	TT (period: 1ms)
4	µECU4'	Off-Chip Core1	TT (period: 1ms)	µECU8'	Off-Chip Core2	TT (period: 1ms)
5	Off-Chip Core1	µECU5	TT (period: 1ms)	µECU9	Off-Chip Core2	RC (MINT: 3 ms)
6	Off-Chip Core1	µECU1	RC (MINT: 3 ms)	Off-Chip Core2	µECU10	BE
8	µECU2	µECU3	BE	µECU10	µECU11	BE
9	µECU3	Off-Chip Core1	BE	µECU11	µECU12	BE

Table 7.13: NoC Configurations in Off-Chip Network Gateway Core Evaluation

used to evaluate the ability of the implemented off-chip gateway core to exploit the μECU redundancy under failure occurrences. The scheduling of the inter-core communication for the two NoCs is summarized in table 7.13. Additionally, table 7.14 presents the scheduling of the TT-CAN bus communication between the MPSoCs.

7.4.2 Results

To validate the correct operation of the off-chip gateway core we compare the speed of the car and the braking distance having enabled and disabled the μECU4 in MPSoC1 and μECU8 in MPSoC2 to simulate a crash failure of the μECUs, so the μECU redundancy should maintain the ABS functionality fully operational. Figure 7.22 compares the car speed while Figure 7.23 compares the braking distance performance. Figure 7.23 presents an increase of the braking distance of $0.78m$ ($\backsimeq 0.35\%$) when the crash failure is injected on both MPSoCs

TT-CAN Configuration				
ID	**Period**	**Phase**	**Sender MPSoC**	**Receiver MPSoC**
1	10ms	1ms	1	2
2	10ms	2ms	2	1
3	10ms	3ms	2	1
4	10ms	4ms	1	2

Table 7.14: TT-CAN Communication Configuration

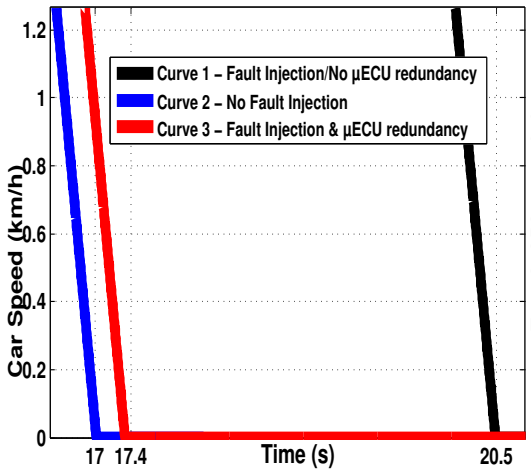

Figure 7.22: Comparison of Car Angular Speeds in Off-chip Network Gateway Core Evaluation

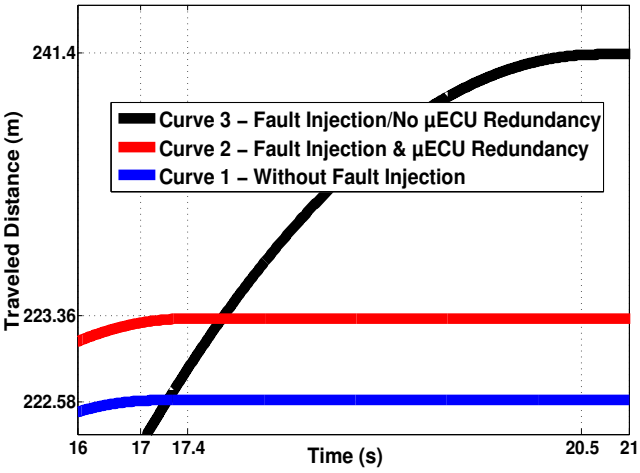

Figure 7.23: Comparison of Braking Distances in Off-chip Network Gateway Core Evaluation

(curve 2) but still meeting a significant performance compared to curve 3 when no μECU redundancy is offered.

In order to quantify the influence of the gateway core on the AUTOSAR multicore system Figures 7.24, 7.25 and 7.26 compare the overall number of task invocations performed by the OS of each μECU for the 3 automotive implemented functionalities. For this, the μECUs where distributed differently on the two multicores, keeping the off-chip network functionality in the own BSW of each μECU. Thus, our described use case was also executed without an implementation of the gateway core.

The μECUs performing the ABS functionality reflects a reduction of the OS overhead of 28.56%, 66.64%, 28.56%, 18.17% and 49.96% when the gateway is used, while μECUs for the LIS and BLI functionalities present a reduction of 49.97%, 33.32%, 49.95%, 49.95% and 22.21%, and 66.62% and 49.98% respectively. These results demonstrate how the developed off-chip gateway core improves the efficiency of the MPSoC platform acting as a hardware accelerator for the AUTOSAR application running in the μECUs.

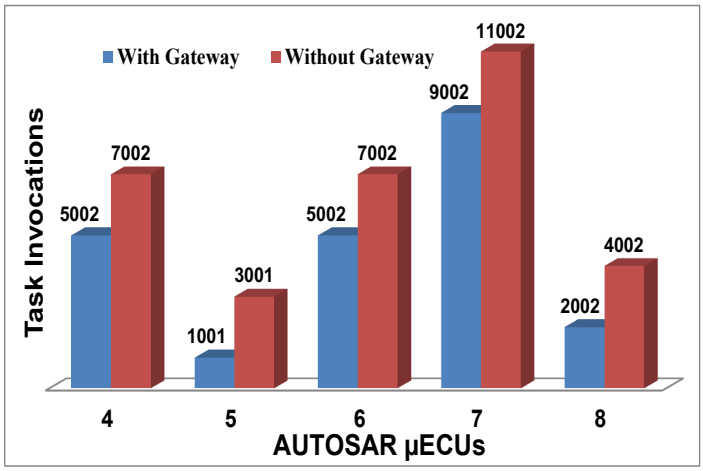

Figure 7.24: Overhead Comparison for ABS functionality

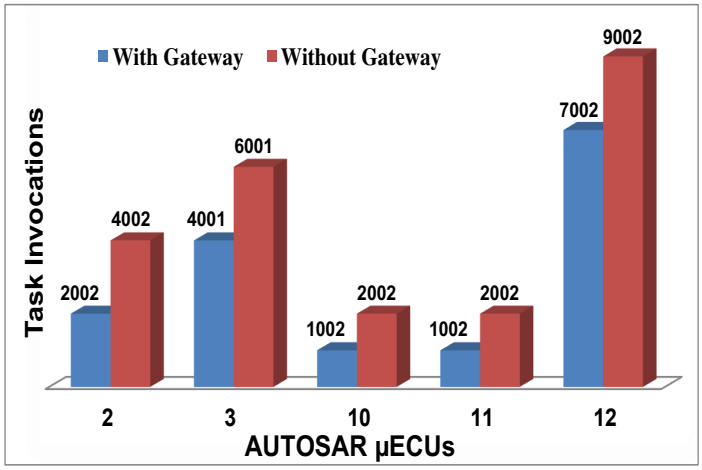

Figure 7.25: Overhead Comparison for LIS functionality

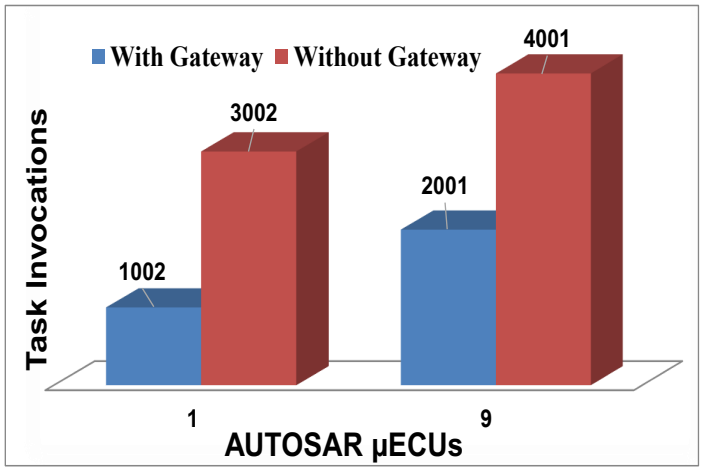

Figure 7.26: Overhead Comparison for BLI functionality

7.4.3 Discussion

The results show that the introduction of an off-chip network gateway core in a message-based multicore platform for AUTOSAR simplifies the AUTOSAR BSW in the μECUs since the off-chip communication functionality is delegated to the dedicated core. The off-chip gateway core allows the communication between SWCs in different MPSoCs which is performed through the off-chip automotive network implemented by the gateway.

Due to the use of the message-base NoC for inter-core communication, SWCs in different μECUs are able to use the gateway for the exchange of data with other MPSoCs. Additionally, the obtained results reflect a significant increase in the performance of the AUTOSAR operating system since it is not any more required to handle expensive off-chip communication functionalities, which are replaced by the integrated off-chip gateway core.

7.5 Evaluation of Performance with a Memory Gateway Core

In this section we carry out an experimental use case to evaluate the performance of TIMEA on a distributed system when employing the described memory gateway core. For this, the co-simulation environment consisting of VEOS and the SystemC-based TTNoC simulation serves for the implementation of the use case.

7.5.1 Use Case-Description

An automotive use case consisting of a Pedestrian Detection Mechanism (PDM) running in parallel with an audio-video streaming system serves for the evaluation of the implemented memory gateway core performing the hierarchical DMTC protocol. The automotive ISO-26262 functional safety standard is used to set the criticality of the applications.

The combination of the following three applications constitutes the PDM functionality: ***(1)*** A Body Track (BT) computer vision algorithm for pedestrian detection (ASIL D) ***(2)*** An FFT (ASIL C) ***(3)*** A Noise Removal (NR) algorithm (ASIL B). Additionally, since the audio-video streaming service is not critical an ASIL QM criticality level is assigned to it.

The *Simlarge* input-sets of the PARSEC benchmarks [BKL08] are used for the generation of trace-files for 12 cores representing the mentioned applications. These trace-file applications are integrated manually on 12 SWCs that serve as the application layer for 12 generated μECUs using SystemDesk. Before the generation of the OSA simulation file the

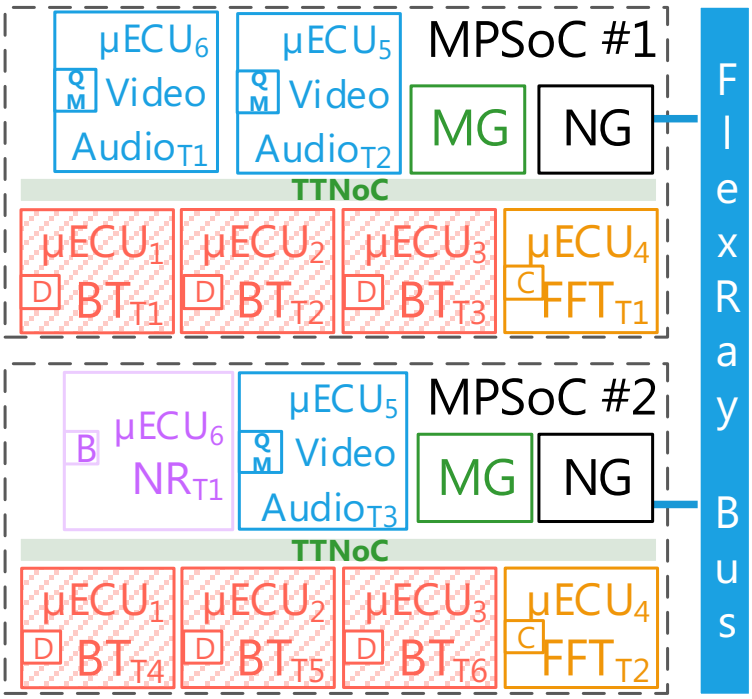

Figure 7.27: Distributed Automotive Use Case for Memory Gateway Core Evaluation

FlexRay Configuration				
ID	**Period**	**Phase**	**Sender MPSoC**	**Receiver MPSoC**
1	1ms	200µs	1	2
2	1ms	300µs	2	1

Table 7.15: FlexRay Bus Communication Configuration

RTE of each μECU is modified to allocate the reading function of the trace-file application in the OS tasks hosting the SWC runnables.

Figure 7.27 represents an architectural picture of the distributed system use case. The μECUs are distributed on two TIMEA platforms in mesh topology with a memory gateway core implementation in both of them, as well as an off-chip gateway core providing access to a FlexRay communication bus for the inter-MPSoC communication. Each MPSoC has an external memory of 4GB micron DDR3. The schedule of the off-chip communication between the two MPSoCs is presented in table 7.15.

7.5.2 Results

The use case was evaluated with respect to the total execution time for each benchmark application core. The execution time of an application core is calculated from the difference between the starting time of the application until the time at which the trace file has finished its

MPSoC 1						
µECU ID	µECU1	µECU2	µECU3	µECU4	µECU5	µECU6
Criticality	ASILD	ASILD	ASILD	ASILC	QM	QM
FOW	2.554s	0.071s	0.049s	4.042s	0.317s	0.021s
DMTC	2.370s	0.037s	0.037s	5.929s	0.424s	0.063s
MPSoC 2						
µECU ID	µECU1	µECU2	µECU3	µECU4	µECU5	µECU6
Criticality	ASILD	ASILD	ASILD	ASILC	QM	ASILB
FOW	0.088s	7.322s	0.103s	6.297s	0.045s	0.237s
DMTC	0.052s	5.430s	0.174s	4.840s	0.054s	0.540s

Table 7.16: Overall Execution Time Per µECU

execution, which means that all messages and memory operations of the trace file have been successfully executed. Additionally, the existing First One Wins (FOW) conflict resolution serves for a comparison. This algorithm allows the first transaction to commit while rolling back all other conflicting transactions.

Table 7.16 compares the overall execution time for each μECU when using the FOW and the DMTC protocols. This table reflects an improvement of the BT application performance of 25.84% when using the DMTC protocol compared to the FOW execution, while the execution time of the FFT application has decreased by 6%. On another hand, the NR and video-audio streaming applications present an increase of their execution time when using the DMTC protocol in comparison to the FOW execution. This is due to the criticality prioritizing of the DMTC execution while the FOW handles non critical applications in a similar way as the critical applications. The DMTC ensures the fastest execution time of the μECUs with higher criticality level since their execution is independent from the execution of the μECUs with a lower criticality.

Additionally, the use case was executed keeping the memory services on the BSW of each AUTOSAR μECU, which means, without an implementation of the dedicated memory gateway core. Thus, a comparison between the OS overhead on the μECUs can be established with and without accelerated system cores.

Figures 7.28 and 7.29 present the number of task invocations executed by the OS of each μECU on both MPSoCs. For μECUs on MPSoC 1 Figure 7.28 shows a reduction of the OS overhead by 41.67%, 34.51%, 33.43%, 27.3%, 29.1% and 53.2% when the memory gateway core is implemented. Likewise, the OS overhead is reduced by 64.2%, 77.8%, 50.2%, 43.1%, 58.8% and 67.13% on the μECUs in MPSoC 2.

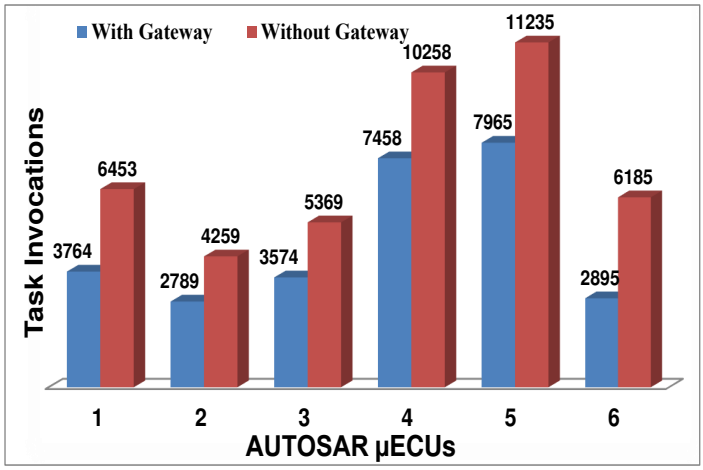

Figure 7.28: Overhead Comparison in MPSoC 1

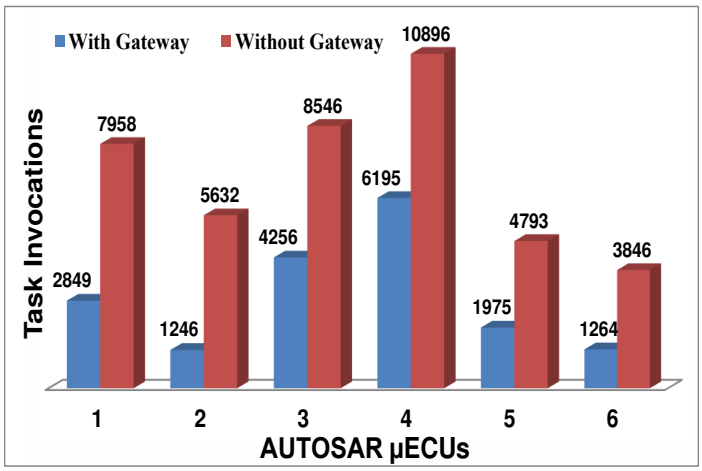

Figure 7.29: Overhead Comparison in MPSoC 2

7.5.3 Discussion

The obtained results demonstrate how the introduced memory gateway core can be accessed by different μECUs running different applications using the TTNoC. The DMTC protocol performed by the memory gateway core imposes no additional execution delay on applications hosting the higher criticality level while transactions required by the μECUs hosting non critical applications are executed with a lower priority.

Furthermore, the operating system overhead has decreased considerably in all μECUs since memory services are not required to be implemented locally on the BSW of the μECUs but they are delegated to the dedicated memory gateway core.

Chapter 8

Conclusion

In the last years MPSoCs have become a suitable option for the development of real-time embedded systems. Specifically in the automotive domain the AUTOSAR standard describes a multicore version of the AUTOSAR operating system for the implementation of automotive systems. However, AUTOSAR just focuses on a shared memory implementation for the inter-core communication for the interaction between SWCs allocated in different cores.

Based on the analysis of the state-of-the-art presented in Chapter 3, this dissertation introduced a novel TIme-triggered MEssage-based multicore architecture for AUTOSAR (TIMEA) together with models and algorithms to provide message-based on-chip communication and health monitoring services to the AUTOSAR system.

The proposed AUTOSAR multicore architecture combines the AUTOSAR software with a message-based NoC platform. This architecture exploits the benefits of NoC architectures achieving a better temporal predictability and stronger fault isolation for the cores running the AUTOSAR SWCs. Application cores play the role of AUTOSAR μECUs in the MPSoC with the extension of new communication service modules in the BSW to support on-chip network communication. Moreover, computationally expensive functionalities of the AUTOSAR BSW are delegated to dedicated system cores that serve as hardware accelerators to the μECUs, so a lightweight implementation of the BSW in the μECUs decreases considerably the overhead of the AUTOSAR operating system.

The presented architecture identifies μECUs as FCRs, which are isolated from timing faults occurring outside their constrained region due to the temporal behavior of the message-based NoC for the communication between the cores. The pre-defined temporal configuration capability of the network allows to set up time-triggered messages for the communication between μECUs running safety critical functionalities. Additionally, the integrated health monitoring service in the BSW of the μECUs provides failure detection to the AUTOSAR software delimiting SWCs as FCRs addressing software failures in the time and value

domains. SWC redundancy at the μECU level, as well as at the MPSoC level serves as recovery actions to counteract SWC failures.

For the implementation of TIMEA a novel framework supporting the integration of the AUTOSAR architecture with NoC architectures was presented (see chapter 5). The framework can be used in conjunction with the available simulation tools and consists of one AUTOSAR-based system simulator, for the simulation of the AUTOSAR software and the physical environment, and one on-chip simulator for the simulation of the NoC communication, implementing local coordinators in combination with additional coordination mechanisms to provide an interface for synchronization and data exchange between the two simulation tools. Furthermore, simulation building blocks for the extended AUTOSAR BSW for NoC communication were introduced.

The simulation framework was implemented using the VEOS simulation tool for the simulation of the AUTOSAR application and environment models while two different simulation models of message-based NoCs served for the simulation of the on-chip communication. The dSpace AUTOSAR development tools were employed for the configuration of the AUTOSAR μECU with the extended BSW modules for health monitoring and proxy functionalities for managing the dedicated system cores.

Several automotive use cases in a realistic simulation scenario were implemented to carry out a set of experiments which validated the capability of the framework and served to evaluate the performance of the proposed multicore architecture under failure occurrences. The obtained results revels how the reliability of the AUTOSAR system is improved significantly by the following aspects:

1. Safety critical μECUs are isolated from failures in the time domain, due to the static communication schedule of the network.

2. The presented AUTOSAR multicore platform continuous providing an outstanding performance of safety critical automotive functionalities in case of fault occurrences in the SWCs. The pre-defined fault assumptions are detected by the health monitoring service which is able to trigger replicated SWCs in the same μECU, as well as in a different μECU.

3. The operating system overhead in the μECUs has decreased considerably as expected due to the introduction of the system cores for hardware acceleration.

Future Work

Future work will continue to focus on the integration of the NoC configuration to the AUTOSAR work flow. For this, the NoC configuration parameters must be mapped to one of the AUTOSAR data/information exchange formats (e.g., FIBEX file). As a first implementation the NoC configuration parameters of the GARNET network inside GEM5 will serve for the definition of the AUTOSAR on-chip communication file. A python-based tool is thought to be developed which will use this AUTOSAR on-chip communication file for the automated generation of the CSV configuration files to set the GEM5-based NoC.

Additionally, tools for the automatic code generation of the defined proxy and health monitoring modules must be developed as well as for the generation of the communication service modules (i.e., COM, PDU router), so support for NoC communication can be provided to the μECUs.

Furthermore, a real prototype implementation of TIMEA is planned to be carried out using a multicore-FPGA platform running a DREAMS-NoC. The challenge behind this real implementation is the lack of support from commercial AUTOSAR tools for the generation of AUTOSAR software that can be deployed in such a platform, specially for the MCAL, where hardware dependencies must be entirely developed.

Bibliography

[Aer15] Aeronautical Radio, Inc., 2551 Riva Road, Annapolis, Maryland 21401. *Specification 653: Avionics Application Software Standard Interface, Overview of ARINC 653*, 2015.

[AGR07] B. Akesson, K. Goossens, and M. Ringhofer. Predator: A predictable sdram memory controller. In *In Proceedings of the 5th IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 251–256, Sept 2007.

[AKPJ09] N. Agarwal, T. Krishna, L. S. Peh, and N. K. Jha. Garnet: A detailed on-chip network model inside a full-system simulator. In *In Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software*, pages 33–42, April 2009.

[ALRL04] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Secur. Comput.*, 1(1):11–33, January 2004.

[AO15] H. Ahmadian and R. Obermaisser. Time-triggered extension layer for on-chip network interfaces in mixed-criticality systems. In *In Proceedings of the Euromicro Conference on Digital System Design*, pages 693–699, Aug 2015.

[ART06] ARTEMIS (Advanced Research and Technology for Embedded Intelligence and Systems). *ARTEMIS Final Report on Reference Designs and Architectures Constraints and Requirements*, 2006.

[ASA17] Association for Standardization of Automation and Measuring Systems (ASAM). *The Universal Measurement and Calibration Protocol Family, Version 1.5.0*, 2017.

[AUT16a] AUTOSAR Consortium. *AUTOSAR Specification of I/O Hardware Abstraction, AUTOSAR Release 4.3*, 2016.

[AUT16b] AUTOSAR Consortium. *Layered Software Architecture, AUTOSAR Release 4.3*, 2016.

[AUT16c] AUTOSAR Consortium. *Operating System, AUTOSAR Release 4.3*, 2016.

[AUT16d] AUTOSAR Consortium. *Software Component Template, AUTOSAR Release 4.3*, 2016.

[AUT16e] AUTOSAR Consortium. *Specification of Communication, AUTOSAR Release 4.3*, 2016.

[AUT16f] AUTOSAR Consortium. *Specification of ECU state manager, AUTOSAR Release 4.3*, 2016.

[AUT16g] AUTOSAR Consortium. *Specification of Interoperability of AUTOSAR Tools, AUTOSAR Release 4.3*, 2016.

[AUT16h] AUTOSAR Consortium. *Specification of PDU Router, AUTOSAR Release 4.3*, 2016.

[AUT16i] AUTOSAR Consortium. *Specification of Run Time Environment, AUTOSAR Release 4.3*, 2016.

[BBB+11] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, August 2011.

[Ber09] T. B. Berg. Maintaining i/o data coherence in embedded multicore systems. *IEEE Micro*, 29(3):10–19, May 2009.

[BKL08] C. Bienia, S. Kumar, and Kai Li. Parsec vs. splash-2: A quantitative comparison of two multithreaded benchmark suites on chip-multiprocessors. In *In Proceedings of the IEEE International Symposium on Workload Characterization*, pages 47–56, Sept 2008.

[BKM13] M. Becker, U. Kiffmeier, and W. Mueller. Heroes: Virtual platform driven integration of heterogeneous software components for multi-core real-time architectures. In *In Proceedings of the Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 2013 IEEE 16th International Symposium on*, pages 1–8, June 2013.

[BO+11] T. Blochwitz, M. Otter, et al. The Functional Mockup Interface for Tool independent Exchange of Simulation Models. In *In Proceedings of the 8th International Modelica Conference*, 2011.

[CCG+04] M. Coppola, S. Curaba, M. D. Grammatikakis, G. Maruccia, and F. Papariello. Occn: a network-on-chip modeling and simulation framework. In *Proceedings Design, Automation and Test in Europe Conference and Exhibition*, volume 3, pages 174–179 Vol.3, Feb 2004.

[CG03] Lukai Cai and Daniel Gajski. Transaction level modeling: An overview. In *In Proceedings of the 1st IEEE/ACM/IFIP International Conference on Hardware/-Software Codesign and System Synthesis*, CODES+ISSS '03, pages 19–24, New York, NY, USA, 2003. ACM.

[CHE11] Trevor E. Carlson, Wim Heirman, and Lieven Eeckhout. Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulations. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pages 52:1–52:12, 2011.

[CLP14] F. Cucchetto, A. Lonardi, and G. Pravadelli. A common architecture for co-simulation of SystemC models in QEMU and OVP virtual platforms. In *In Proceedings of the Very Large Scale Integration (VLSI-SoC), 2014 22nd International Conference on*, pages 1–6, Oct 2014.

[CMM+15] V. Catania, A. Mineo, S. Monteleone, M. Palesi, and D. Patti. Noxim: An open, extensible and cycle-accurate network on chip simulator. In *2015 IEEE 26th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 162–163, July 2015.

[CYT11] Ming-Chao Chiang, Tse-Chen Yeh, and Guo-Fu Tseng. A QEMU and SystemC-Based Cycle-Accurate ISS for Performance Estimation on SoC Development. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 30(4):593–606, April 2011.

[DBT09] S. Faucou D. Bertrand and Y. Trinquet. An analysis of the AUTOSAR OS timing protection mechanism in Emerging Technologies and Factory Automation. *ETFA 2009. IEEE Conference on*, 2009.

[DMK+18] E. Díaz, E. Mezzetti, L. Kosmidis, J. Abella, and F. J. Cazorla. Modelling multicore contention on the aurix™ tc27x. In *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2018.

[dSp14a] dSpace. *SystemDesk 4.x Guide, Release 2014-B*, 2014.

[dSp14b] dSpace. *Target Link AUTOSAR Modeling Guide, Release 2014-B*, 2014.

[dSp14c] dSpace. *VEOS Player Document, Release 2014-B*, 2014.

[dSp14d] dSpace. *Virtual Validation Overview, Release 2014-B*, 2014.

[Dub13] Elena Dubrova. *Fault-tolerant design*. Springer, 2013.

[Edi12] Editted by: Roman Obermaisser. *Time-Triggered Communication*. Embedded Systems. CRC Press, USA, 2012.

[ETA16] ETAS. *Multi-core Automotive ECUs: Software and Hardware Implications*, 2016.

[FFTft] Fast Fourier Transform based on SystemC, https://github.com/systemc/systemc-2.2.0/tree/master/examples/sysc/fft.

[Fle05] FlexRay Consortium. BMW AG, DaimlerChrysler AG, General Motors Corporation, Freescale GmbH, Philips GmbH, Robert Bosch GmbH, and Volkswagen AG. *FlexRay Communications System Protocol Specification Version 2.1*, May 2005.

[FMZ+12] Zhenman Fang, Qinghao Min, Keyong Zhou, Yi Lu, Yibin Hu, Weihua Zhang, Haibo Chen, Jian Li, and Binyu Zang. Transformer: A functional-driven cycle-accurate multicore simulator. In *In Proceedings of the Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*, pages 106–114, June 2012.

[GD+05] K. Goossens, J. Dielissen, et al. Aethereal network on chip: concepts, architectures, and implementations. *Design Test of Computers, IEEE*, 22(5):414–421, Sept 2005.

[Ghe06] Frank Ghenassia. *Transaction-Level Modeling with Systemc: Tlm Concepts and Applications for Embedded Systems*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[HBS+06] H. Heinecke, J. Bielefeld, K. Schnelle, M. Maldener, H. Fennel, O. Weis, T. Weber, L. Ruh, J. Lundh, T. Sandén, P. Heitkämper, R. Rimkus, J. Leflour, A. Gilberg, U. Virnich, S. Voget, K. Nishikawa, K. Kajio, T. Scharnhorst, and B. Kunkel. AUTOSAR Current results and preparations for exploitation. In *In Proceedings of the EUROFORUM "Software in the vehicle"*, 2006.

[Hea02] Steve Heath. *Embedded Systems Design*. Elsevier Science, 2002.

[HGBH09] Andreas Hansson, Kees Goossens, Marco Bekooij, and Jos Huisken. CoMPSoC: A Template for Composable and Predictable Multi-processor System on Chips. *ACM Trans. Des. Autom. Electron. Syst.*, 14(1):2:1–2:24, January 2009.

[HGY07] C. Harding, A. Griffiths, and H. Yu. An interface between matlab and opnet to allow simulation of wncs with manets. In *In Proceedings of the Networking, Sensing and Control, 2007 IEEE International Conference on*, pages 711–716, April 2007.

[HO09] B. Huber and R. Obermaisser. An ARTEMIS Cross-Domain Embedded System Architecture and Its Instantiation for Real-Time Automotive Applications. In *In Proceedings of the 30th IFAC Workshopon Real-Time Programming and 4th International Workshop on Real-Time Software*, Poland, 2009.

[Int11] International Organization for Standardization. Road vehicles-Functional safety-Part 9: Automotive Safety Integrity Level (ASIL)-oriented and safety-oriented analyses. *ISO 26262-9:2011, ICS: 43.040.10*, 2011.

[Int15] International Organization for Standardization. Road vehicles — Controller area network (CAN). *ISO 11898-1:2015, ICS: 43.040.15*, 2015.

[IST06] IBM, Sony, and Toshiba. *Cell broadband engine architecture*. Tech, 2006.

[JGJ+09] C. Jian, J. Guanjun, L. Jingwei, W. Chao, and C. Tianzhou. Optimistic peripheral devices performance by virtual regionalized network-on-chip. In *In Proceedings of the Scalable Computing and Communications; Eighth International Conference on Embedded Computing, 2009. SCALCOM-EMBEDDEDCOM'09. International Conference on*, pages 650–655, Sept 2009.

[Joh98] D. John. Osek/vdx history and structure. In *IEE Seminar on OSEK/VDX Open Systems in Automotive Networks (Ref. No. 1998/523)*, pages 2/1–214, Nov 1998.

[Kam11] Raj Kamal. *Embedded systems: architecture, programming and design*. Tata McGraw-Hill Education, 2011.

[KGC12] G. Kornaros, M. D. Grammatikakis, and M. Coppola. Towards full virtualization of heterogeneous noc-based multicore embedded architectures. In *In Proceedings of the Computational Science and Engineering (CSE), 2012 IEEE 15th International Conference on*, pages 345–352, Dec 2012.

[KKOE07] K. Kronlof, S. Kontinen, I. Oliver, and T. Eriksson. A method for mobile terminal platform architecture development. *Advances in Design and Specification Languages for Embedded Systems*, pages 285–300, 2007.

[Kop92] H. Kopetz. Sparse time versus dense time in distributed real-time systems. In *[1992] Proceedings of the 12th International Conference on Distributed Computing Systems*, pages 460–467, June 1992.

[Kop13] Herman Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Springer Publishing Company, Incorporated, 2nd edition, 2013.

[KYBS14] J. E. Kim, M. K. Yoon, R. Bradford, and L. Sha. Integrated modular avionics (ima) partition scheduling with conflict-free i/o for multicore avionics systems. In *In Proceedings of the Computer Software and Applications Conference (COMPSAC), 2014 IEEE 38th Annual*, pages 321–331, July 2014.

[KYL+] Hyesoon Kim, Sudhakar Yalamanchili, Jaekyu Lee, Nagesh Lakshminarayana, Andrew Kerr, Arun Rodrigues, and Genie Hsieh. Tutorial on ocelot and sst-macsim simulator. ISCA 2012.

[KZK+14] C. T. Kiranoudis, E. Zachariadis, I. Keramitsoglou, K. Saini, O. Kakaliagou, and E. Kleitsikas. Wildfire evacuation trigger buffers for sensitive areas: Evita project. In *2014 Third International Workshop on Earth Observation and Remote Sensing Applications (EORSA)*, pages 121–125, June 2014.

[LAK92] J.C. C. Laprie, A. Avizienis, and H. Kopetz, editors. *Dependability: Basic Concepts and Terminology*. Springer-Verlag, Berlin, Heidelberg, 1992.

[LAS+07] Jacob Leverich, Hideho Arakida, Alex Solomatnikov, Amin Firoozshahian, Mark Horowitz, and Christos Kozyrakis. Comparing Memory Systems for Chip Multiprocessors. In *In Proceedings of the 34th Annual International Symposium on Computer Architecture*, ISCA '07, pages 358–368, New York, NY, USA, 2007. ACM.

[LHSC10] Xin Li, Michael C. Huang, Kai Shen, and Lingkun Chu. A realistic evaluation of memory hardware errors and software system susceptibility. In *Proceedings of the 2010 USENIX Conference on USENIX Annual Technical Conference*, USENIXATC'10, pages 6–6, Berkeley, CA, USA, 2010. USENIX Association.

[LS11] E.A. Lee and S.A. Seshia. *Introduction to Embedded Systems: A Cyberphysical Systems Approach*. Electrical Engineering & Computer Sciences. Lulu.com, 2011.

[LSL+09] Yan Li, V. Suhendra, Yun Liang, T. Mitra, and A. Roychoudhury. Timing Analysis of Concurrent Programs Running on Shared Cache Multi-Cores. In *In Proceedings of the Real-Time Systems Symposium 30th IEEE*, Dec 2009.

[Matml] Mathworks. *Modeling an Anti-Lock Braking System*, www.mathworks.com/help/simulink/examples/modeling-an-anti-lock-braking-system.html.

[MCE+02] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, and B. Werner. Simics: A full system simulation platform. *Computer*, 35(2):50–58, Feb 2002.

[MNT+04] M. Millberg, E. Nilsson, R. Thid, S. Kumar, and A. Jantsch. The Nostrum backbone-a communication protocol stack for Networks on Chip. In *In Proceedings of the VLSI Design, 2004. 17th International Conference on*, pages 693–696, 2004.

[Modls] Modelica Association. *FMI Support in Tools*, https://fmi-standard.org/tools/.

[MRR08] B. Muller-Rathgeber and H. Rauchfuss. A Cosimulation Framework for a Distributed System of Systems. In *In Proceedings of the Vehicular Technology Conference, 2008. VTC 2008-Fall. IEEE 68th*, pages 1–5, Sept 2008.

[NHT+12] K. Nakajima, T. Hieda, I. Taniguchi, H. Tomiyama, and H. Takada. A Fast Network-on-Chip Simulator with QEMU and SystemC. In *In Proceedings of the Networking and Computing (ICNC), 2012 Third International Conference on*, pages 298–301, Dec 2012.

[OG09] R. Obermaisser and P. Gutwenger. Model-based development of mpsocs with support for early validation. In *In Proceedings of the Industrial Electronics, 2009. IECON '09. 35th Annual Conference of IEEE*, pages 2867 – 2873, 2009.

[OK09] R. Obermaisser and H. Kopetz. *GENESYS: A Candidate for an ARTEMIS Cross-Domain Reference Architecture for Embedded Systems*. Südwestdeutscher Verlag für Hochschulschriften, 2009.

[OKP10] R. Obermaisser, H. Kopetz, and C. Paukovits. A Cross-Domain Multi-Processor System-on-a-Chip for Embedded Real-Time Systems. *IEEE Transactions on Industrial Informatics*, 2010.

[OKS08] R. Obermaisser, H. Kraut, and C. Salloum. A transient-resilient system-on-a-chip architecture with support for on-chip and off-chip tmr. In *2008 Seventh European Dependable Computing Conference*, pages 123–134, May 2008.

[OO15] Z. Owda and R. Obermaisser. Trace-based simulation framework combining message-based and shared-memory interactions in a time-triggered platform. In *In Proceedings of the International Conference on Event-based Control, Communication, and Signal Processing (EBCCSP)*, pages 1–8, June 2015.

[OO16] Z. Owda and R. Obermaisser. Mixed-criticality transactional memory controller for embedded systems. In *In Proceedings of the IEEE 14th International Conference on Industrial Informatics (INDIN)*, pages 104–110, July 2016.

[OP07] Roman Obermaisser and Philipp Peti. The fault assumptions in distributed integrated architectures. In *In Proceedings of the SAE AeroTech Congress and Exhibition*, 2007.

[OUOA16] Z. Owda, M. Urbina, R. Obermaisser, and M. Abuteir. Hierarchical transactional memory protocol for distributed mixed-criticality embedded systems. In *In Proceedings of the IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 14th International Conference on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress(DASC/PiCom/DataCom/CyberSciTech)*, pages 334–343, Aug 2016.

[Owd16] Z. Owda. *Predictable Transactional Memory Architecture for Hierarchical Mixed-Criticality Systems*. University of Siegen. Fcaulty of Embedded Systems, 2016.

[PHO+14] J. Power, J. Hestness, M. Orr, M. Hill, and D. Wood. GEM5-GPU: A Heterogeneous CPU-GPU Simulator. *Computer Architecture Letters*, page 1, 2014.

[PPB+07] F. Poletti, A. Poggiali, D. Bertozzi, L. Benini, P. Marchal, M. Loghi, and M. Poncino. Energy-Efficient Multiprocessor Systems-on-Chip for Embedded Computing: Exploring Programming Models and Their Architectural Support. *Computers, IEEE Transactions on*, 2007.

[RCBJ11] P. Rosenfeld, E. Cooper-Balis, and B. Jacob. Dramsim2: A cycle accurate memory system simulator. *IEEE Computer Architecture Letters*, 10(1):16–19, Jan 2011.

[ROH+09] M. Rudorfer, T. Ochs, P. Hoser, M. Thiede, M. Missmer, O. Scheickl, and H. Heinecke. Realtime system design utilizing AUTOSAR methodology. In *elektroniknet*, 2009.

[SEH+12] C.E. Salloum, M. Elshuber, O. Hoftberger, H. Isakovic, and A. Wasicek. The ACROSS MPSoC – A New Generation of Multi-core Processors Designed for Safety-Critical Embedded Systems. In *In Proceedings of the Digital System Design (DSD), 2012 15th Euromicro Conference on*, pages 105–113, Sept 2012.

[Shi09] KV Shibu. *Introduction to Embedded Systems*. Tata McGraw-Hill Education, 2009.

[Son02] Sonics. *Sonics u network technical overview*, 2002.

[Sta96] William Stallings. *Computer Organization and Architecture (4th Ed.): Designing for Performance*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996.

[Sto96] Neil R. Storey. *Safety Critical Computer Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1996.

[UAcLR01] Algirdas Avizienis Ucla, Algirdas Avizienis, Jean claude Laprie, and Brian Randell. Fundamental concepts of dependability, 2001.

[UAO16] M. Urbina, H. Ahmadian, and R. Obermaisser. Co-simulation framework for autosar multi-core processors with message-based network-on-chips. In *In Proceedings of the IEEE 14th International Conference on Industrial Informatics (INDIN)*, pages 1140–1147, July 2016.

[UBG+13] T. Ungerer, C. Bradatsch, M. Gerdes, F. Kluge, R. Jahr, J. Mische, J. Fernandes, P. G. Zaykov, Z. Petrov, B. Böddeker, S. Kehr, H. Regler, A. Hugl, C. Rochange, H. Ozaktas, H. Cassé, A. Bonenfant, P. Sainrat, I. Broster, N. Lay, D. George, E. Quiñones, M. Panic, J. Abella, F. Cazorla, S. Uhrig, M. Rohde, and A. Pyka. parmerasa – multi-core execution of parallelised hard real-time applications supporting analysability. In *2013 Euromicro Conference on Digital System Design*, pages 363–370, Sep. 2013.

[UCS+10] T. Ungerer, F. Cazorla, P. Sainrat, G. Bernat, Z. Petrov, C. Rochange, E. Quiñones, M. Gerdes, M. Paolieri, J. Wolf, H. Cassé, S. Uhrig, I. Guliashvili, M. Houston, F. Kluge, S. Metzlaff, and J. Mische. Merasa: Multicore execution of hard real-time applications supporting analyzability. *IEEE Micro*, 30(5):66–75, Sep. 2010.

[UO15] M. Urbina and R. Obermaisser. Multi-core architecture for autosar based on virtual electronic control units. In *In Proceedings of the IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA)*, pages 1–5, Sept 2015.

[UO16] M. Urbina and R. Obermaisser. A gateway core between on-chip and off-chip networks for an autosar message-based multi-core platform. In *In Proceedings of the AmE 2016 - Automotive meets Electronics; 7th GMM-Symposium*, pages 1–6, March 2016.

[UO17] M. Urbina and R. Obermaisser. Efficient multi-core autosar-platform based on an input/output gateway core. In *In Proceedings of the 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)*, pages 157–166, March 2017.

[UO18] M. Urbina and R. Obermaisser. Evaluation of performance and fault containment in autosar micro-ecus on a multi-core processor. In *2018 IEEE 12th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*, pages 192–200, Sep. 2018.

[UOO15] M. Urbina, Z. Owda, and R. Obermaisser. Simulation environment based on systemc and veos for multi-core processors with virtual autosar ecus. In *In Proceedings of the IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing*, pages 1843–1852, Oct 2015.

[Urb17] Moisés Urbina. A time-triggered message-based multi-core architecture for autosar: Student research abstract. In *In Proceedings of the Symposium on Applied Computing*, SAC '17, pages 1488–1489, New York, NY, USA, 2017. ACM.

[Var99] A. Varga. Using the OMNeT++ discrete event simulation system in education. *Education, IEEE Transactions on*, 42(4):11, Nov 1999.

[VEC16] VECTOR. *AUTOSAR goes Multi-core – the safe way*, 2016.

[VWHY13] Andreas Richard Voellmy, Junchang Wang, Paul Hudak, and Kazuhiko Yamamoto. Mio: A high-performance multicore io manager for ghc. *SIGPLAN Not.*, 48(12):129–140, September 2013.

[WG14] P. Wehner and D. Gohringer. Parallel and distributed simulation of networked multi-core systems. In *In Proceedings of the System-on-Chip (SoC), 2014 International Symposium on*, pages 1–5, Oct 2014.

[ZEK+13] Zhenkai Zhang, E. Eyisi, X. Koutsoukos, J. Porter, G. Karsai, and J. Sztipanovits. Co-simulation framework for design of time-triggered cyber physical systems. In *In Proceedings of the Cyber-Physical Systems (ICCPS), 2013 ACM/IEEE International Conference on*, pages 119–128, April 2013.

[Zur09] Richard Zurawski. *Embedded Systems Handbook, Second Edition 2-Volume Set.* CRC Press, Inc, Boca Raton, FL, USA, 2009.

[ZW+13] H. Zhang, S Wang, et al. Testing method of integrated modular avionics health monitoring. In *CHEMICAL ENGINEERING TRANSACTIONS CET, AIDIC publication*, 2013.

List of Acronyms

TTNI Time-Triggered Network Interface

NoC Network-on-Chip

CAN Controller Area Network

CAN FD CAN with Flexible Data-Rate

LIN Local Interconnect Network

ECU Electronic Control Unit

FCR Fault Containment Region

FIBEX Fieldbus Exchange Format

DBC Database Container

MPSoC Multi-Processor System-on-a-Chip

NI Network Interface

OEM Original equipment manufacturer

TTNoC Time-Triggered Network-on-a-Chip

VC Virtual Channel

AUTOSAR AUTomotive Open System ARchitecture

DMTC Distributed Mixed-criticality Transactional Controller

OS Operating System

RTE Run-Time Environment

SWC Software Component

TIMEA TIme-triggered MEssage-based multi-core architecture for AUTOSAR

BSW Basic Software

PDU Protocol Data Unit

FMU Functional Mock-up Unit

FMI Functional Mock-up Interface

μECU Micro-Electronic Control Unit

LIS Light Indicator System

BLI Breaking Light Indicator

ABS Anti-lock Braking System

FFT Fast Fourier Transform

ASM Automotive Simulation Model

TTEL Time-Triggered Extension Layer

VFB Virtual Funcitonal Bus

MCAL Micro-Controller Abstraction Layer

MINT Minimum INter-arrival Time

VPU Virtual Processing Unit

OSA Offline Simulation Application

XCP Universal Measurement and Calibration Protocol

CSV Comma-Separated Values

MIL Model-in-the-loop

SIL Software-in-the-loop

OCCN On-Chip Communication Network

TDMA Time Division Multiple Access

TLM Transaction Level Model

DAP Data Access Point

TDDM TargetLink Data Dictionary Manager

ASIL Automotive Safety Integrity Level

PDM Pedestrian Detection Mechanism

BT Body Track

NR Noise Removal

FOW First One Wins

IOC Inter OS-application Communicator

LIFO Last In, First Out

IMA Integrated Modular Avionic

NoTa Network on Terminal Architecture

ARINC Aeronautical Radio, Incorporated

TTSoC Time-Triggered System-on-a-Chip

RTOS Real Time Operating System

GHC Glasgow Haskell Compiler

HSM Hardware Security Module

parMERASA Multi-Core Execution of Parallelised hard Real-Time Applications Supporting Analysability

MERASA Multi-Core Execution of hard Real-Time Applications Supporting Analysability

WCET Worst Case Execution Time

Selected Publications

1. M. Urbina and R. Obermaisser, "Evaluation of Performance and Fault Containment in AUTOSAR Micro-ECUs using dedicated System Cores on a Multi-Core Processor", The 13th ACM/IEEE International Workshop on Network on Chip Architectures (NocArc-2020) at the 53rd ACM/IEEE International Symposium on Microarchitecture (MICRO-53). Athens, Greece - October 2020.

2. M. Urbina and R. Obermaisser, "Evaluation of Performance and Fault Containment in AUTOSAR Micro-ECUs on a Multi-Core Processor", The IEEE 12th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC-2018). Hanoi, Vietnam - September 2018.

3. M. Urbina, “A Time-triggered Message-based Multi-core Architecture for AUTOSAR", The ACM 17th Symposium on Applied Computing (SAC-2017). Marrakesh, Morocco - April 2017.

4. M. Urbina and R. Obermaisser, “Efficient Multi-core AUTOSAR-Platform based on an Input/Output Gateway Core", The IEEE 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP-2017). St. Petersburg, Russia - March 2017.

5. Z. Owda, M. Urbina, R. Obermaisser, M. Abuteir, “Hierarchical Transactional Memory Protocol for Distributed Mixed-Criticality Embedded Systems”, **BEST PAPER AWARD** at The 14th IEEE International Conference on Dependable, Autonomic and Secure Computing (DASC-2016). Auckland, New Zealand - August 2016.

6. M. Urbina, H. Ahmadian, R. Obermaisser, “Co-simulation Framework for AUTOSAR Multi-Core Processors with Message-based Network-on-Chips”, The IEEE 14th International Conference on Industrial Informatics (INDIN-2016). Poitiers-Futuroscope, France - July 2016.

7. M. Urbina and R. Obermaisser, "A Gateway Core between On-chip and Off-chip Networks for an AUTOSAR Message-based Multi-core Platform”, The 7th Automotive meets Electronics GMM-Symposium (AmE-2016). Dortmund, Germany - March 2016.

8. M. Urbina, Z. Owda, R. Obermaisser, “Simulation Environment based on SystemC and VEOS for Multi-Core Processors with Virtual AUTOSAR ECUs”, The 13th IEEE International Conference on Dependable, Autonomic and Secure Computing (DASC-2015). Liverpool, England - October 2015.

9. M. Urbina and R. Obermaisser, “Multi-Core Architecture for AUTOSAR based on Virtual Electronic Control Units”, The 20th IEEE International Conference on Emerging Technology & Factory Automation (ETFA-2015). Luxembourg, September - 2015.