Architecture Design for Distributed Mixed-Criticality Systems based on Multi-Core Chips

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

vorgelegt Dissertation von:

Mohammed Abuteir

eingereicht bei der Naturwissenschaftlich-Technischen Fakultät der Universität Siegen
Siegen– Dezember 2016

Promotionskommission:
Prof. Dr. Roman Obermaisser
Prof. Dr. Roland Wismüller
Prof. Dr. Madjid Fathi
Prof. Dr. Kristof Van Laerhoven

Tag der mündlichen Prüfung:
4 Mai 2017

Lehrstuhl Embedded Systems

Architecture Design for Distributed Mixed-Criticality Systems based on Multi-Core Chips

This dissertation is submitted for the degree of

Doctor of Engineering

by

Mohammed Abuteir

Submitted to the Faculty of Science and Technology of
the University of Siegen
Siegen– December 2016

Examination commission:
Prof. Dr. Roman Obermaisser
Prof. Dr. Roland Wismüller
Prof. Dr. Madjid Fathi
Prof. Dr. Kristof Van Laerhoven

Chair for Embedded Systems

Defense Date: 4 May 2017

This dissertation is dedicated to my parents, my wife, and my family.

Acknowledgements

I would like to express my special appreciation and thanks to my advisor Professor Dr. Roman Obermaisser, you have been a tremendous mentor for me. I would like to thank you for encouraging my research and for allowing me to grow as a research scientist. Your advices on both research as well as on my career have been priceless.

Special thanks to my parents and my family for their endless love and support through my life, and for my lovely wife for her unconditional love, support, and understanding.

To all my friends and colleagues thank you for your motivation and encouragement.

Kurzfassung

In vielen Anwendungsbereichen wie beispielsweise der Avionik, industriellen Kontrollsystemen und dem Gesundheitswesen gewinnen sogenannte Mixed-Criticality Systeme, in denen Anwendungen mit unterschiedlicher Wichtigkeit sowie unterschiedlichen sicherheitskritischen Anforderungen auf einer gemeinsamen Rechenplattform implementiert werden, immer größere Bedeutung. Die Hauptanforderung an solche Systeme ist ein modularer Sicherheitsnachweis, der eine unabhängige Zertifizierung von Anwendungen anhand der zugehörigen Sicherheitsebenen unterstützt. Um dieses Ziel zu erreichen fehlt im Stand der Technik jedoch eine Mixed-Criticality Architektur für vernetzte Multi-Core-Chips mit Echtzeitunterstützung, Fehlereingrenzung und Sicherheit. Die Dissertation befasst sich mit dieser Problematik und bietet einen Lösungsansatz auf Basis von Architekturmodellen, selektiver Fehlertoleranz, Scheduling-Techniken und einer Simulationsarchitektur.

Die Basis dieser Integration sind Mechanismen für die zeitliche und räumliche Partitionierung, die die Sicherheit der Anwendungen mit verschiedenen Kritikalitätsstufen sicherstellen, so dass keine gegenseitige Beeinflussung entsteht. Die zeitliche Partitionierung wird über den Einsatz von autonomer zeitlicher Kontrolle basierend auf einem zeitgesteuerten Schedule mit definierten Zeitpunkten aller Kommunikationsaktivitäten in Bezug auf eine globale Zeitbasis realisiert. Diese Zeitpunkte der periodischen Nachrichten verbessern die Vorhersehbarkeit und ermöglichen eine rigorose Fehlererkennung und Fehleranalyse.

Zeitgesteuerte Schedules erleichtern zudem die Beherrschung der Komplexität von Fehlertoleranzmechanismen und die Erstellung analytischer Zuverlässigkeitsmodelle. Ferner wird eine Partitionierung der Netzwerkbandbreite verwendet um verschiedene Zeitmodelle (z.B. periodisch, sporadisch und aperiodisch) zu kombinieren.

Ein weiterer Beitrag dieser Arbeit ist die selektive Fehlertoleranz für Mixed-Criticality Systeme. Ein Hauptmerkmal der Fehlertoleranz in Kommunikationsprotokollen wie Time-Triggered Ethernet (TTEthernet) und ARINC 664 ist die Bereitstellung redundanter Kommunikationskanäle zwischen Netzwerkknoten über mehrere unabhängige Netzwerkkomponenten. Die Datenflüsse zwischen den Netzwerkknoten sind gegen Fehler der verschiedenen Netzwerkkomponenten, wie beispielsweise Links oder Switches, geschützt. Der Hauptnachteil replizierter Netzwerke in großen Systemen sind jedoch die zusätzlichen Kosten,

insbesondere wenn die Netzwerke ihre Dienste für mehrere Subsysteme, nämlich nicht-sicherheitskritische und kritische Subsysteme, bereitstellen. Diese Arbeit stellt eine neuartige Systemarchitektur vor, welche die Redundanz in Mixed-Criticality Systemen basierend auf einer Ring-Topologie unterstützt. Diese Architektur erfüllt die Anforderung der sicherheitskritischen Systeme und ist gleichzeitig auch für nicht-sicherheitskritische Systeme wirtschaftlich einsetzbar. Das Hauptmerkmal der vorgeschlagenen Architektur ist die Fehlereingrenzung, so dass Fehler keinen Einfluss auf Subsysteme mit höherer Kritikalität aufweisen. Außerdem garantiert die vorgeschlagene Architektur die Bereitstellung von Nachrichten mit begrenzten Verzögerungen und begrenztem Jitter.

Basierend auf den in dieser Arbeit vorgestellten Architekturansätzen werden effiziente Scheduling-Algorithmen für große Mixed-Criticality Systeme mit verschiedenen Zeitmodellen eingeführt. Die Architekturmodelle werden auch mit Hilfe eines Simulations-Frameworks evaluiert, welches hierarchische Mixed-Criticality Systeme mit vernetzten Multi-Core-Chips unterstützt. Ferner wird dieses Framework verwendet um die vorgeschlagenen Scheduling-Algorithmen zu verifizieren. Diese Evaluation wird zudem um analytische Modelle der End-to-End-Kommunikation für verschiedene Kritikalitätsstufen ergänzt.

Abstract

In many domains such as avionics, industrial control, or healthcare there is an increasing trend to mixed-criticality systems, where applications of different importance and criticality are implemented on a shared computing platform. The major requirement of such a system is a modular safety case where each application is certified to the respective assurance level. A mixed-criticality architecture for networked multi-core chips with real-time support, fault isolation and security is missing in the state-of-the-art. In this dissertation, we advance the state-of-the-art by providing solutions to research gaps towards such an architecture for networked multi-core chips, which include the architecture models, selective fault-tolerance concepts, scheduling techniques, and a simulation framework.

The foundations for this integration are mechanisms for temporal and spatial partitioning, to ensure that applications of different criticality levels are protected so they cannot influence each other. We establish temporal partitioning using autonomous temporal control based on a time-triggered schedule containing the instants of all message exchanges with respect to a global time base. The predetermined instants of the periodic messages improve predictability and enable rigorous error detection and fault isolation. The time-triggered schedules facilitate managing the complexity of fault-tolerance and analytical dependability models. In addition, we use network bandwidth partitioning to support different timing models (i.e., periodic, sporadic and aperiodic traffic). We introduce an architectural model for mixed-criticality systems based on networked multi-core chips, which describes both the physical system structure as well as a logical system structure of the application.

Another contribution of the dissertation is a selective fault-tolerance concept for mixed-criticality systems. One of the key features of existing fault-tolerant communication protocols such as Time-Triggered Ethernet (TTEthernet) and ARINC 664 is providing redundant channels for the communication between nodes over multiple independent network components. The data flows between the nodes are protected against the failure of any network component such as a link or a switch. However, the main drawback of replicated networks in large systems is the extra cost, in particular, if the networks provide their services for non safety-critical subsystems alongside with the critical subsystems. We introduce a novel system architecture supporting redundancy in mixed-criticality systems based on a ring topology,

which fulfills the requirements of high-critical systems while also being economically suitable for low-critical systems. The main characteristic of the proposed architecture is fault isolation so that a failure of a low-critical subsystem cannot reach subsystems of higher criticality. Moreover, the proposed architecture supports the delivery of messages with bounded delays and bounded jitter.

Based on these contributions, we address the scheduling algorithms for large scale mixed-criticality systems where different criticality levels of the subsystem as well as high numbers of nodes and applications lead to a steady increase of the complexity of scheduling the events associated with such systems.

The architecture models have also been evaluated using a simulation framework. This simulation framework is established for hierarchical mixed-criticality systems based on networked multi-core chips. Additionally, this framework is used to verify the proposed scheduling algorithms. This evaluation is accompanied by analytical models of end-to-end communication for different criticality levels.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The use of multi-core processors in embedded systems enables new applications with high-performance requirements such as embedded vision systems for autonomous vehicles [KG14]. In addition, the computational power of a multi-core processor facilitates a higher physical integration, where several electronic functions can be implemented on a single chip. In large electronic systems, e.g. the distributed in-vehicle electronic system of a car, this higher integration allows providing given services with fewer ECUs compared to systems with single-core processors. Benefits include a reduction of cabling, lower hardware cost, less weight and easier installation.

The physical integration often leads to mixed-criticality systems, if the functions of the multi-core processor exhibit different safety assurance levels. In this case, mechanisms for temporal and spatial partitioning [Rus01, Rus99a] are required, which establish fault containment and the absence of unintended side-effects between functions.

At the same time, multi-core processors introduce significant challenges for safety-critical systems and mixed-criticality systems. An example of such a challenge is the sharing of resources (e.g., caches, buses and inputs/outputs), which can lead to temporal interferences precluding the assurance of the real-time requirements. Therefore the use of multi-core processors in safety-critical systems is a cause of concern to certification authorities. For example, avionic certification authorities point out that the features of multi-core processors could cause a loss of integrity, a loss of availability or non-deterministic behavior [Cer14].

In order to overcome these challenges, the integration of functions with different criticality using time and space partitioning has been introduced at various integration levels in prior research. Operating systems and execution layers that provide these services based on task scheduling, memory protection and suitable hardware abstractions are available as products (e.g. PikeOS [Sys10], Deos [DI11]).

Temporal and spatial partitioning were also addressed in communication networks at a chip level. Deterministic multi-core platforms use message-based Network-on-a-Chips (NoCs) with Time Division Multiple Access (TDMA) to avoid the temporal unpredictability and the potential for fault propagation of architectures with shared memories and memory hierarchies. Examples of these architectures are the GENESYS MPSoC [SEH+12] and Æthereal [GH10].

However, a single multi-core chip is insufficient in many embedded applications. Therefore, hierarchical networks including off-chip and on-chip networks are required. Likewise, hierarchical networks are required to achieve a system reliability beyond the reliability of a single chip and to satisfy resource requirements exceeding the capacity of a single chip. As a consequence, hierarchical platforms emerge in which cores inside a multi-core chip interact by on-chip networks whereas multi-core chips are interconnected by off-chip networks.

At present, there is, however, a significant gap between the mixed-criticality integration at chip-level and off-chip level, which is a challenge for upcoming mixed-criticality systems with multi-core chips. This dissertation introduces multi-core platforms for a hierarchical system perspective of mixed-criticality applications combining the chip and off-chip level. This combination is established through a gateway to enable vertical integration and seamless communication in hierarchical networks respecting mixed-criticality safety requirements.We support message-based NoCs and off-chip networks with different timing models, while also establishing real-time guarantees, fault isolation and protocol transformations.

Moreover, hierarchical networks with different timing models including time-triggered communication, event-triggered communication with rate-constraints and best-effort communication require new scheduling and allocation algorithms in order to establish the connectivity between nodes, while satisfying the application requirements with respect to timeliness, performance, safety and availability.

This dissertation focuses on the design of a hierarchical architecture that is suitable for mixed-criticality services. This architecture requires configurations that depend on the particular set of applications that is deployed in the system. Therefore, new scheduling algorithms are introduced to guarantee the correct temporal behavior of the applications in the system. Figure 1.1 gives an overview of the main contributions of the dissertation.

1.1 Objectives

A major contribution of this dissertation is the design of a mixed-criticality architecture with a hierarchical platform comprised of networked multi-core chips. We consider communication resources with the respective timing properties. This system model consists of switches and

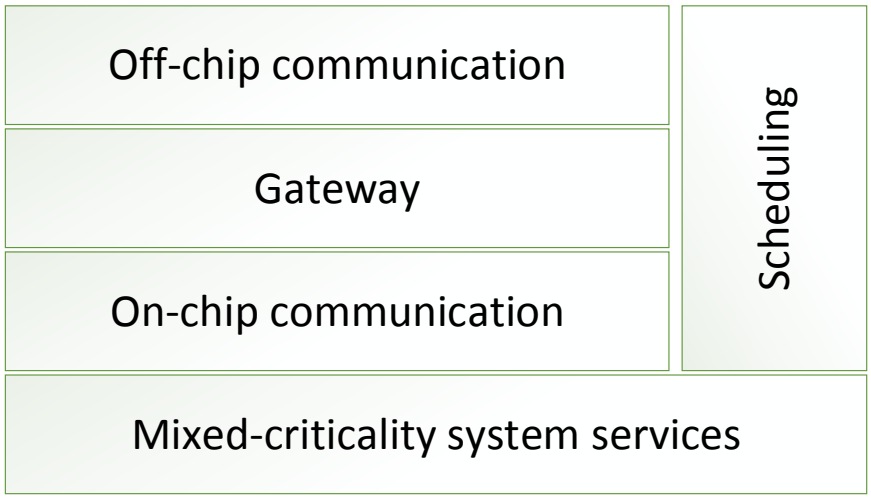


Figure 1.1 Dissertation Overview

nodes that can be implemented as multi-cores. Each multi-core node communicates with other nodes through the off-chip/on-chip gateway. This system supports full flexibility for designing complex network topologies with different applications as well as varying numbers of nodes and switches.

Moreover, the proposed system architecture ensures a predictable timing with bounded latency and jitter based on periodic time-triggered and sporadic rate constrained messages. Fault-tolerance using redundant switches and multiple stars is supported to preserve the communication services despite the failure of individual physical links or switches. However, other communication topologies are required in order to achieve scalability to large-scale systems and to support mixed-criticality systems. We introduce multi-ring topologies with corresponding switches for large-scale mixed-criticality systems. The architecture supports redundant channels using heterogeneous paths for periodic time-triggered and sporadic rate constrained messages, where switches autonomously perform the duplication and deduplication of redundant messages. Differences with respect to latencies on the redundant paths are hidden to ensure an unchanged network timing in the case of failures.

Furthermore, multi-core processors promise improved performance and a higher physical integration by combining functions of different criticality levels in one platform. Networked multi-core chips are required to achieve a system reliability beyond the reliability of a single chip and to satisfy resource requirements exceeding the capacity of a single chip. As a consequence, hierarchical platforms emerge in which cores inside a multi-core chip interact by on-chip networks whereas multi-core nodes are interconnected by off-chip networks. We present gateways for establishing such a hierarchical platform. We support message-based

NoCs and off-chip networks with different timing models, while also supporting real-time guarantees, fault isolation and protocol transformations as follows:

- *Gateways supporting different timing models.* The proposed gateways support three types of timing models for the redirection of messages between on-chip and off-chip networks: (1) periodic time-triggered messages, (2) sporadic event-triggered communication with rate-constraints, and (3) aperiodic, best-effort communication.

- *Temporal partitioning.* The presented gateways enforce temporal specifications including periods and phases of time-triggered messages and rate-constraints of an event-triggered communication. Untimely messages are blocked, thereby preventing fault propagation between chip level and off-chip level.

Moreover, we present a scheduling algorithm that performs the allocation of computational and communication activities to nodes and networks, as well as the scheduling of the communication and execution times. Both periodic time-triggered activities as well as event-triggered rate-constrained activities are supported. Moreover, we introduce a simulation and validation environment, which supports the automatic generation of test cases based on scenario parameters and the testing of scheduling algorithms using these test cases. Thereby, a large number of test cases can be analyzed in order to gain a deeper understanding of the behavior of scheduling algorithms under different scenarios. We present the generic simulation environment and use it for the evaluation of the presented neighborhood scheduling and allocation algorithm. The simulation environment is also an effective foundation for the comparison of different scheduling algorithms. Thereby, we enable a comprehensive evaluation of the scheduling algorithm for use cases of varying complexity. In addition, the simulation and verification framework is a foundation for the systematic comparison of different scheduling algorithms including the evaluation of schedulability and run-time for different types of scenarios.

To evaluate the mixed-criticality architecture for the hierarchical systems and to validate the scheduling algorithm, the simulation framework is instantiated for the proposed architecture. The main results of the simulation framework are generic building blocks of the infrastructure elements of the proposed system, which can be configured and extended to create an application-specific simulation model:

- *Generic model of a switch.* We have developed a generic simulation model of a switch supporting periodic time-triggered, sporadic rate-constrained and aperiodic best-effort communication. In order to construct the overall simulation model, the user can perform multiple instantiations of the generic switch, establish connections

to nodes and other switches, and assign to each switch instantiation a corresponding configuration. The switch configuration defines the message timing including a periodic time-triggered communication plan.

- *Generic model of a node.* The user can perform instantiates of the generic node and connect each instantiation to switches. Nodes can be configured to produce messages according to application-specific parameters (e.g., interarrival time distributions of sporadic messages, periods of periodic messages). In addition, nodes can be extended with the application behavior (e.g., C++ application code).

- *Generic fault injectors.* Building blocks for fault injection allow investigating the system behavior in the presence of component failures. Generic fault injectors can be instantiated and configured to inject specific failure modes (e.g., babbling idiot, masquerading failure, ...).

- *Generic model of a switch with fault-tolerance.* We extended the switch simulation building block to integrate and evaluate the proposed fault-tolerance mechanism.

- *Generic gateway*: We have developed a generic simulation model of the gateway to couple the chip and off-chip simulations. The proposed gateways are realized as simulation components based on GEM5 and the GARNET NoC models. The simulation environment demonstrates the timely redirection and fault isolation of periodic time-triggered, sporadic rate-constraint, and aperiodic messages.

1.2 Overview

The dissertation is structured as follows:

- Chapter 2 contains definitions and detailed information about the main concepts and terms that are used throughout the dissertation. It starts by explaining the dependability concepts with their main classifications, and the fault hypothesis concept with its main components. The chapter continues with the real-time systems concepts with the concentration on distributed systems. Then the chapter explains different architecture paradigms ranging from federated to integrated and mixed-criticality architectures. Then the partitioning at various levels such as processor, memory, I/O and communication is illustrated. Finally, the chapter ends with certification concepts and modular certification.

- Chapter 3 gives an overview of the state-of-the-art in the areas of on-chip communication, off-chip communication, fault-tolerance, gateways, and scheduling algorithms. The chapter closes with an overview of the research gap in the state of the art and the proposed architectures.

- Chapter 4 presents the architecture for the hierarchical system that supports the mixed-criticality services. A conceptual and a concrete system model of a platform that consists of networked multi-core chips are introduced. In the conceptual system model, we describe the physical and logical system structure, the platform services of the system model and the fault hypothesis.

- Chapter 5 introduces multi-ring topologies with corresponding switches for large-scale mixed-criticality systems. The architecture supports redundant channels using heterogeneous paths for periodic time-triggered and sporadic rate constrained messages, where switches autonomously perform the duplication and deduplication of redundant messages. Differences with respect to latencies on the redundant paths are hidden to ensure an unchanged network timing in the case of failures.

- Chapter 6 focuses on the gateway model and services of a bridging both on-chip and off-chip networks that address the requirements for a system perspective of mixed-criticality applications by combining both networks and by performing protocol transformations between heterogeneous networks.

- Chapter 7 presents a scheduling algorithm for the mixed-criticality systems. We support the allocation and scheduling of periodic time-triggered and sporadic rate constrained applications to nodes and communication links.

- Chapter 8 provides the detailed description of the implementation of the simulation framework. The architecture and the proposed models are evaluated using a simulation framework in different use-cases. it also describes a simulation and verification framework for the hierarchical system. The simulation and verification framework supports the automatic generation of test cases based on generic scenario parameters including the connectivity degree as well as the number of networks, nodes, switches and services. Thereby, we enable a comprehensive evaluation of the scheduling algorithm for use cases of varying complexity.

- Chapter 9 concludes the dissertation and gives an outlook for future work on mixed-criticality systems.

Chapter 2

Concepts and Terms

This chapter contains definitions and detailed information about the main concepts and terms that are used throughout this dissertation.

2.1 Dependability

Dependability [LAK92] is the property of a computer system that reliance can justifiably be placed on the service it delivers. The service delivered by a system is its behavior as it is perceived by its user(s). A user is another system (physical, human) which interacts with the former. A systematic exposition of the concepts dependability consists of three parts: the threats to, the attributes of, and the means by which dependability is attained.

Threats

The threats to dependability are faults, errors, and failures, which can affect a system and cause a drop in dependability. Threats are expected to occur or to be part of any system since no system can be designed or operated perfectly.

- **Failure:** It occurs when the actual behavior of the component is no longer consistent with to the specification, either because the component does not comply with the specification, or because the specification did not adequately describe its function [LAK92].
- **Error:** It is the part of the state of the system that may cause a subsequent failure. A failure occurs when an error reaches the service interface.
- **Fault:** It is the adjudged or hypothesized cause of an error. A fault is a concept that is introduced to stop recursion [UAcLR01]. As stated in [LAK92], the fault can be classified according to various criteria such as the phenomenological cause, the intent,

the domain, the phase of creation or of occurrence, the location with respect to the system boundaries, and the persistence.

Attributes

According to [LAK92], the following attributes characterize a dependable system:

- **Reliability:** It is the probability that a system will provide the correct behavior (specified service) during the period of a mission. The reliability denotes the probability that the system functions properly and continuously in the period of a mission.
- **Safety:** It is the ability of the system to avoid the occurrence of a catastrophic failure for a given application in a given environment.
- **Security:** It is concerned with the authenticity and integrity of information, and the ability of a system to prevent unauthorized access to information or services.
- **Integrity:** It is defined as the absence of improper alterations of information.
- **Maintainability:** It is related to the time interval that the system needs to repair itself after the occurrence of a benign failure, and it can be defined as the ability of the system to undergo repairs.
- **Availability:** It is measured by the fraction of time that the system is ready to provide the service. The availability is determined by the reliability and maintainability of the system. High availability of a system can be achieved either by high reliability or by short repair times.

Means

The concept of the means of dependability consists of the following techniques, to achieve the various attributes of dependability [ALRL04].

- **Fault prevention:** It is a set of techniques attempting to eliminate or reduce the introduction or occurrence of faults in the system during the design and manufacturing of hardware and software. These techniques are the quality control techniques that are implemented in the process of manufacturing, and development of software or hardware systems.
- **Fault tolerance:** It targets techniques and methods to keep the system providing its service in the presence of faults. It includes error detection, recovery and fault handling, which mask faults or prevent faults from being activated again.

- **Fault removal:** It is a set of techniques targeting the reduction of the number of faults which are present in the system. It includes verification to test if the system‘s service is within the specifications. It also includes diagnosis that finds mistakes in the system implementation and deploys corrective actions.

- **Fault forecasting:** It is the technique used to estimate how many faults are present in the system. Fault forecasting also analyses possible future occurrences of faults and the consequences of faults. It is done by performing an evaluation of the system behavior regarding to the fault occurrences or activation. It can be qualitative by identifying, classifying and ranking failures, or it can be quantitative which is basically a probabilistic evaluation of the satisfaction regarding the dependability attributes.

2.2 Fault Hypothesis

The fault hypothesis specifies assumptions that describe the types of faults, the rate at which components fail and how components may fail [Pow92]. The fault hypothesis is a central part in any safety-relevant system and provides the foundation for the design, implementation and test of the fault-tolerance mechanisms [Obe12].

A Fault Containment Region (FCR) is a collection of components or a subsystem that operates correctly regardless of any arbitrary logical or electrical fault outside the region [Pow92]. A FCR is a set of subsystems that share one or more common resources that one single fault may affect [Kop11]. An FCR limits the immediate impact of a fault, but fault effects manifested as erroneous data can propagate across FCR boundaries. Therefore, the system must also provide error containment [OP06] to avoid error propagation by the flow of erroneous messages.

An Error Containment Region (ECR) is defined as a subsystem that is encapsulated by error-detection interfaces such that there is a high probability that the consequences of an error that occurs within this subsystem will not propagate outside this subsystem without being detected and/or masked [Kop11]. The error detection mechanisms must be part of different FCRs than the message sender. Otherwise, the error detection mechanism may be impacted by the same fault that caused the message failure.

Part of the fault hypothesis is a specification of the failure rate of FCRs. In general, different failure rates with respect to different failure modes and failure persistence are necessary. Related to the failure rates in industrial communication the residual error rate needs to be calculated according to IEC 61784-3 [IEC10d]. The residual error rate needs to stay below 1% of the probability of dangerous failures per hour (PFH) of the target

Safety Integrity Level (SIL) according to IEC 61508. Furthermore, failure persistence (i.e., permanent or transient) is an important factor in the differentiation of failure rates.

2.3 Concept of Component, Service and Behavior

We use the concept of a task for the process of executing an algorithm. The tasks are executed in components and a component is considered to be a self-contained hardware/software unit that communicates via a communication service that enables components to interact with their environment exclusively by the exchange of messages. The timed sequence of output messages that a component produces is called the behavior of the component. The intended behavior of a component is called its service.

2.4 Concept of State

The concept of the state is fundamental for the investigation of complex systems. The state is introduced in order for the systems description to separate the past from the future behavior [Kop11]. This definition is based on the idea of Mesarovic and Takahara [MT89] if one knows what state a deterministic system is in and the future inputs, he could with assurance ascertain what the output will be. Hence, the state of a system accumulates the history and captures only what is relevant for the future behavior of the given system. In a deterministic system, future outputs just depend on the current state and the future inputs.

2.5 Real-Time Systems

Real-time computing systems [Kop11] are systems in which the correctness of the system behavior depends not only on the logical result of the computation but also on the time at which the results are produced.

The real-time system usually has inputs that correspond to entities in the physical world, and outputs that also relate to physical entities. Most of these entities are connected to controlling processes. The most stringent temporal requirements come from control loops where all the functions for controlling the physical environment are included, e.g., controller computing a set value of a controlled entity such as an automotive engine. The lag time between inputs and outputs must be sufficiently small to ensure the stability of control. The time interval when a result (output) must be produced is called a deadline. The deadline is classified as soft, if a result has utility even after the deadline has passed, otherwise it is firm. Deadlines are called hard deadlines if severe consequences can result from missing a

deadline. In fact, the real-time system is a system that maintains a continuous and timely interaction with the environment (cf. Figure 2.1).

Figure 2.1 Real-Time System

2.5.1 Classification of Real-Time Systems

2.5.1.1 Concept of Hard and Soft Real-Time System

Real-time systems can be classified into two categories: hard real-time systems and soft real-time systems. Hard real-time systems have strict temporal constraints, in which missing the specified deadline could have a dramatic impact on human life and on the environment. The system damage when missing a deadline can be orders of magnitude higher than the utility of the system under normal operation. Hard real-time systems are used in many domains such as military applications, space missions, and automotive applications. Automobile engine control systems and anti-lock brakes are examples of hard real-time systems.

Soft real-time systems also have temporal constraints but these constraints are not as strict. In other words, the missing of deadlines does not lead to a catastrophic failure of the system. Examples of soft real-time applications are call admittance in voice over internet and cell phones, multimedia services and augmented reality systems.

2.5.1.2 Resource Adequacy for Hard Real-Time Systems and Best-Effort for Soft Real-Time Systems

Resource adequacy is related to the provision of enough computing and communication resources to handle the specified fault- and load-hypothesis. The system that supports resource adequacy needs careful planning and extensive analysis during the design phase. The system that does not support guarantees is called best effort, which is only suitable for non safety-critical applications. There are two reasons to support such kinds of systems:

- **Economic viability**: The provision of sufficient resources to handle every possible situation involves high cost.
- **Dynamic systems and flexibility**: A dynamic resource allocation strategy based on resource sharing and probabilistic arguments about the expected load and fault scenarios is suitable for highly dynamic systems with dynamically changing compositions of components.

2.5.1.3 Concept of Time-Triggered and Event-Triggered Control

According to [Kop11], a trigger is *an event that causes the start of some action, e.g., the execution of a task or the transmission of a message*. Two different approaches in the design of real-time systems can be distinguished according to the triggering mechanisms for the processing and communication activities:

Time-Triggered System:

In the time-triggered approach, the communication and processing activities are initiated at a particular point in time of a synchronized *global time base*. The global time base is a sparse time [Kop92] and enables the temporal coordination of actions by providing a system wide clock reference. In the time-triggered system, each process activation or message communication is done based on a static schedule table built offline.

Event-Triggered System:

In event-triggered systems, event triggers serve as control signals for communication and computational activities. According to [Obe05], the event can originate either from activities within the computer system (e.g., termination of a task) or from state changes in the natural environment (e.g., alarm condition indicated by a sensor element).

2.5.1.4 Fail-Safe and Fail-Operational Systems

Two different approaches can be distinguished in the realization of a safe real-time system.

Fail-Safe System:

Fail-safe systems have one or more safe states that can be reached in case of a system failure. In other words, the system will not endanger lives or property when it fails. This system must, however, have a high error-detection coverage. An example of a fail-safe system is a

railway signaling system. A safe state for this system might be setting all signals to red and thus stopping all the trains.

Fail-Operational System:

Fail-operational means that the system must continue to operate correctly in case of a failure. This system requires a technique to mask component failures and continue the provision of the correct service. One of the possible techniques that can be used is active redundancy with voting (see [OKS08] for more details). A flight control system aboard an airplane is an example of a fail-operational system [Kop11].

2.6 Architecture Paradigms

In the last decades, the use of embedded systems in many domains such as the automotive and avionic industry has rapidly increased. Sensors and control subsystems with different criticalities are becoming more complex, where these subsystems should meet stringent specifications for safety, reliability, availability and other attributes of dependability. Additionally to that, the requirement for small size suitable for mobility an extremely low production costs require small and controlled resource consumption with limited hardware capacity.

2.6.1 Federated Architecture

Each dedicated node implements at most one service, and the applications are loosely coupled [OESHK09, Rus99b]. Figure 2.2 shows an example of a federated architecture where we have three applications of different criticality levels implemented on a distributed system. In case an application consists of several services, each service can comprise several tasks in its own node that is only loosely coupled to the nodes of another service.

This architecture has remarkable advantages from the point of view of complexity management, fault isolation, and fault containment. In the federated architecture, the fault containment units are clearly defined and due to separate physical resources, a faulty task cannot affect the rest of the system. The evident drawback of the federated architecture is its profligate use of resources, which means that the number of nodes in the system is as high as the number of services. This also increases the associated wiring, hardware cost, size, weight and power. Another drawback of the federated architecture is the limited sharing of hardware and communication resources.

To overcome these drawbacks of the federated architecture, the trend in real-time systems is towards integrated architectures.

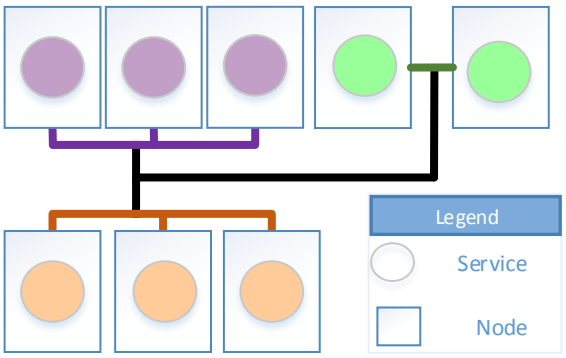


Figure 2.2 Federated Architecture

2.6.2 Integrated Architecture

The integrated architecture describes a system that integrates different services in each node and sharing a single physical communication channel [Kop04b]. An example of the integrated architecture is shown in Figure 2.3.

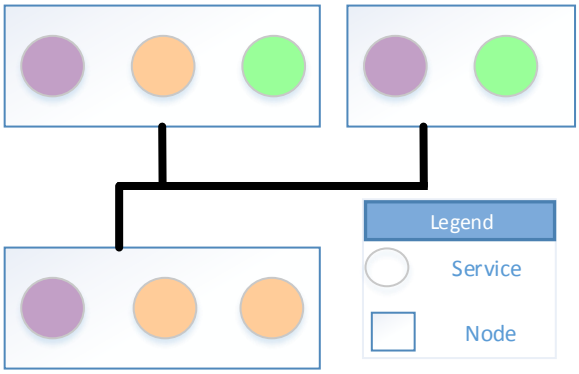


Figure 2.3 Integrated Architecture

Such an architecture reduces cost, increases dependability, and weight by reducing the number of nodes and cables. A drawback of integrated architectures is the complexity increase of the system due to the possible potential for interference through shared resources.

2.6.3 Mixed-Criticality Architecture

A mixed-critical system is an integrated suite of hardware, operating system and middleware services and application software that supports the execution of safety-critical, mission-critical, and non-critical software within a single, secure embedded platform [BBB+09]. In addition to the integration of multiple services with different certification assurance levels using a shared platform, the *modular certification* (see section 2.9) is an important aspect

in a mixed-criticality architecture to limit certification costs. *Partitioning* (see section 2.7) is a prerequisite to enable this *modular certification*, where each component is certified to the respective level of criticality. An example of a mixed-criticality architecture is shown in Figure 2.4.

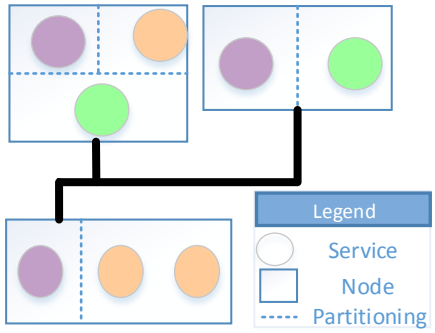


Figure 2.4 Mixed-Criticality Architecture

2.7 Partitioning

A partitioned architecture provides partitioning mechanisms at the platform level for isolating the functional and temporal behavior of each node, in order to avoid the propagation of functional and timing faults. As stated in [Rus99b], the goal of such an architecture is providing fault containment equivalent to an idealized system in which each partition is allocated to an independent processor where associated peripherals and all inter-partition communications are carried on dedicated lines.

In this architecture, two categories of partitions can be distinguished according to their domains: spatial and temporal partitions. Spatial partitioning ensures that a service in a partition will not overwrite memory elements of other partitions (i.e., protection of data and code), and will prevent interference between partitions [Rus99b]. Temporal partitioning guarantees the temporal properties of shared resources (e.g, processor or communication channel) even in the case of faulty services in other partitions.

An example of such an integrated architecture is Integrated Modular Avionics (IMA) [Pri92] in the aerospace domain. Each partition in IMA is associated with dedicated resources such as processor time, memory, I/O and communication.

The partitioning at the processor level can be realized using the hardware or by virtualization in software. An example of a hardware mechanism is a Memory Management Unit (MMU) to ensure spatial partitioning by preventing a running application on different partitions from the overwriting memory of other partitions. An example for temporal

partitioning is a predetermined partition table for granting access of applications to the processor. Virtualization is a technology that introduces a software abstraction layer between the underlying platform and the operating system. This layer is called a Virtual Machine Monitor (VMM) or a hypervisor [BK10, RG05, SML10]. The main responsibilities of the hypervisor are hardware abstraction, robust resource and time partitioning, inter-partition communication, partition management, and the provision of an intra-partition Real-Time OS API.

Examples of virtualization solutions at processor level are the Linux Kernel Virtual Machine (KVM), XtratuM [CRM10], PikeOS [Sys10], VxWorks [PK07] and LynxOS [Lyn02].

The partitioning at memory level can be realized by splitting the memory into segments. Each partition is forced to use only the memory segment which it owns. The memory partitioning can be attained using a hardware approach or a software approach [BCSM08]. Dividing the memory into N segments with fixed size is an example of the hardware approach [KS90]. In the software approach, the code and data of an application are logically restricted to specific memory portions which are done by mapping instructions and assigning data only to certain parts of the addressing space [BCSM08].

The partitioning at I/O level is realized by assigning I/O devices to partitions and extending the protection and isolation properties of partitions for I/O operations. To deal with the interrupts from multiple I/O devices, an application is used, which can configure these interrupts to either be handled directly by a partition (e.g. Intel's VT-x [UNR+05]) or to invoke the hypervisor (e.g. Intel's virtualization [Int14]) which then routes the IRQ to the corresponding partition.

The partitioning at communication level is realized using communication protocols that ensure the spatial and temporal partitioning (e.g. FlexRay, ARINC 664, TTEthernet).

FlexRay [Fle04] is a time-triggered communication protocol with a static segment that supports segregation of the messages from different components in the time and value domains and with a dynamic segment that can be used for event messages.

ARINC 664 [Com05] guarantees the timely delivery of messages by dividing the total bandwidth on the network between the nodes that share the network. Each type of data is assigned a Minimum Inter-Arrival Times (MINT) that limits the frequency with which that type of data can be sent.

TTEthernet [Com11] is a real-time Ethernet extension that supports the exchange of messages with bounded transmission delays, low jitter and high channel utilization. TTEthernet ensuring the deterministic behavior of the time-triggered messages using pre-scheduled communication.

2.8 Certification

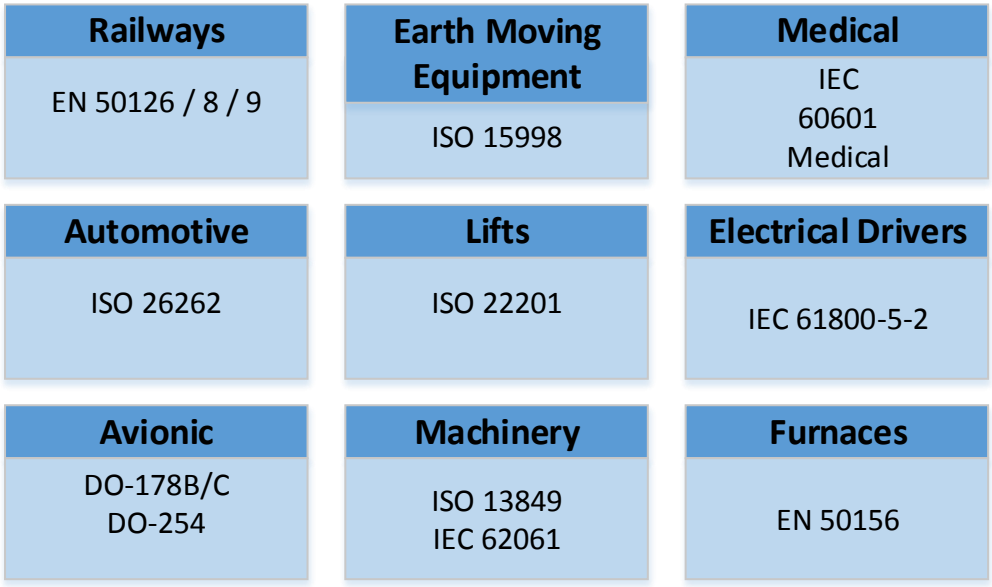


Figure 2.5 Safety Relevant Standards in Different Areas

Certification is a third-party assurance of a product or, system ensuring that the system conforms to defined requirements. There is a large number of corresponding standards that have been established by different organizations like ISO or IEC.

The current approach to certification practice is standards based, which requires that the product and the development processes fulfill the requirements and satisfy the objectives of a certain certification standard, depending on the application area [Rus07]. Figure 2.5 shows examples of safety standards in different areas.

Most of the domain-specific safety standards are derived from the generic safety standard IEC 61508 [IEC10a, IEC10b, IEC10c] (e.g., ISO 26262 [cit09]), or use similar approaches (e.g., DO-178C [oR11]).

The IEC 61508 standard defines four Safety Integrity Levels (SILs) based on the risk of a critical failure, ranging from SIL 4 (most critical) to SIL 1 (least critical). The SIL is determined by the consequences of software failures (systematic failures) and hardware failures (systematic and random failures). Table 2.1 lists the target failure measures for a safety function according to IEC 61508 [IEC10a]. DO-178C defines five assurance levels, rated by the criticality of the software functionality. Level A, the highest criticality level, is assigned for software whose anomalous behavior causes a catastrophic failure condition whereas Level E, the lowest level, is required for software whose anomalous behavior has no effect on the system's operational capacity with respect to safety.

Table 2.1 Safety Integrity Levels - Target Failure Measures for a Safety Function Operating in High Demand Mode of Operation or Continuous Mode of Operation [IEC10a]

Safety Integrity Level, (SIL)	**Average frequency of dangerous failure of the safety function [h^{-1}], (PFH)**
4	$\geq 10^{-9} PFH < 10^{-8}$
3	$\geq 10^{-8} PFH < 10^{-7}$
2	$\geq 10^{-7} PFH < 10^{-6}$
1	$\geq 10^{-6} PFH < 10^{-7}$

2.9 Modular Certification

Modular certification means that a system is certified not as a monolithic piece but as a set of modules using a certification strategy that promises a massive reduction in certification cost through modularization and reuse of certification arguments.

In fact, certification is a significant cost factor in the development of safety-criticality systems. For example, the cost of the certification and validation ranges between 60% and 70% of the cost of an avionics component [ASS$^+$08]. Moreover, the cost of the certification is increasing based on the criticality level (e.g. the development cost of avionic software according to the safety level increases by 300–500% [ASS$^+$08]). Consequently, there is a need for an architectural approach, which enables the certification of small, reusable modules and applications.

The main challenge in modular certification is the need for a component and interface model, which is also valid in the presence of faults. Therefore, fault containment and encapsulation services from an integrated architecture are required.

Although real-time communication protocols such as time-triggered networks can not solve the analysis of such hazards at the application level, they provide an important base line for modular certification due to the temporal and spatial partitioning.

Chapter 3

State of the Art in Mixed-Criticality Systems

This chapter analyses architectures and communication services as well as their suitability for mixed-criticality systems.

One of the main challenges in mixed-criticality systems based on multi-core processors is the safety certification. This is because sufficient evidence must be provided to demonstrate that the resulting system is safe for its purpose. Therefore, the designer of the mixed-criticality system shall consider the following safety aspects for the system [25]:

- No interference between the applications executing simultaneously on the separate cores may occur.
- The code or private data of one application cannot be altered by any other application.
- Applications will have no effect on each other's ability while sharing a common resource such as memory, cache, data buses and peripheral devices of the entire multi-core systems.
- No collision between messages in the communication channels.

The encapsulation concept is the realization of these aspects. To ensure that the mixed-criticality system is free of interferences, the system shall ensure the spatial and temporal independence between applications.

An equally important challenge in mixed-criticality systems based on multi-core processors is reliability. For instance, the increasing rates of transient faults are more significant in highly integrated chips such as modern multi-core processors. Since the mixed-criticality system combines applications with different safety requirements, services such as fault containment and fault tolerance are required.

Furthermore, mixed-criticality systems differ not only in the safety assurance levels, but typically also exhibit varying temporal requirements for the underlying platform. Many applications of the highest criticality levels realize cyclic control services. For example, time-triggered communication protocols are well suited for these safety-critical control applications [Obe11]. Functions of lower criticality often employ less restrictive timing models. Sporadic communication with rate-constraints offers higher potential for resource sharing and bounded latencies, but higher variability of communication latencies. Aperiodic communication activities without temporal guarantees are suitable for non safety-relevant functions.

Another important challenge in mixed-criticality systems is the support for a real-time. Therefore, architectures and communication systems shall guarantee that the messages meet the specified deadlines in all situations. Establishing multiple priorities may not suffice for assuring determinism, since the collisions between messages of the same priority may cause unacceptable delays. Therefore, a deterministic behavior is required to guarantee the end-to-end latency for the real-time messages. To ensure the deterministic communication behavior, a pre-defined schedule and predefined paths are required.

3.1 State of the Art: Communication

This section gives an overview of the state of the art for the on-chip communication as well as the off-chip communication. In addition, we discuss how the mixed-criticality challenges are addressed in different protocols.

3.1.1 On-chip Communication

The use of multi-core processors in embedded systems enables new applications with high performance requirements such as embedded vision systems for autonomous vehicles [KG14]. The uses of multi-core processors also leads to interest in NoCs. A NoC is an interconnection network that transports data between cores, which provides a solution for scalability, parallelism and system modularity, high frequency operation and power efficiency.

Shared resources of a multi-core processor such as caches, buses and inputs/outputs are a source of indeterminism in execution time analysis. Therefore, researchers proposed various resource reservation and priority-based mechanisms to achieve Quality of Service (QoS), i.e., to provide guarantees in latency and bandwidth.

Æthereal [GH10] and Nostrum [LTMJ05] adopt the resource-reservation mechanism to offer guaranteed services for throughput and latency in conjunction with best effort

services. Both architectures employ a Time-Division Multiplexing (TDM) Circuit-Switching mechanism to provide these guarantees. Æthereal uses the concept of open-ended *virtual circuits* while Nostrum uses the concept of closed-loop *virtual circuits*. A detailed discussion of both methods is available in [LJ07]. A virtual circuit is a technique where resources are reserved in both space and time using an explicit time division multiplexing mechanism called temporally disjoint networks [BDM06].

Moreover, the Time-Triggered Network-on-a-Chip (TTNoC) [Pau08, WEK10] uses also TDM with simple routers to provide predictable communication for real-time systems. The TTNoC with a pseudo-static communication schedule allows for a high bandwidth interconnect with inherent fault isolation for heterogeneous components of possibly different criticalities.

Resource-reservation mechanisms require the connection establishment between the source and target components before starting the data transmission. The connection is established by using configuration information to reserve the path between the respective routers. The path reservation avoids the establishment of other conflicting connections in the path. Some NoCs, such as Æthereal, use table data to store the required bandwidth of the guaranteed throughput (GT) flows, but such a table increases the router area significantly.

The advantage of using these NoCs is scalability due to their modular structure and the provision of a guaranteed and predictable performance. Likewise, this method guarantees a lossless and deterministic communication. On the other hand, the resource and the space consumption for the hardware implementation is significantly high.

In the priority-based mechanism two approaches can be distinguished to enable control over communication flows; Static Priority and Dynamic Priority. The static priority approach aims to use a resource allocation mechanism based on static priorities for providing differentiated services to the flows. Each priority has its lane to serve its messages. In contrast, the dynamic priority approach aims to assign the priority to the communication flows as opposed to the static priority approach, where priorities are assigned to lanes. This gives the system more flexibility, which allows sending the transmitted messages through any lane, transmitting messages through different lanes along the message path, and using the time division multiplexing by transmitting messages with the same priority through different lanes in the same physical link.

The priority-based mechanism is adopted in different NoC architectures (e.g. Mango NoC [BS05], STNoC [CGL+08]). Mango NoC uses the concept of *virtual channels* for message-passing over open core protocol (OCP) interfaces to achieve the QoS in an asynchronous network. The virtual channel is implemented using separate physical buffers in each switch that contend for access to the shared physical link. STNoC is a flexible and a

scalable packet-based on-chip network where the router comprises multiple interconnected input and output ports and dynamic arbitration mechanisms that resolve any output port conflicts based on the messages priorities. STNoC also uses the concept of virtual channels.

The priority-based mechanism can only guarantee the QoS for a small number of virtual channels. From the implementation point of view, the router area increases approximately with the square of the number of the virtual channels [MTCM05].

The comparison among NoCs with respect to mixed-criticality requirements is summarized in Table 3.1.

Protocol	Encapsulation	Real-Time (Bounded Latency and Jitter)	Timing Models	Global Time	Fault Containment	Error Containment
Aethereal	Yes	Yes	Periodic and aperiodic	NO	Yes	NO
Nostrum	NO	Yes	Periodic	No	NO	NO
TTNoC	Yes	Yes	Periodic	Yes	Yes	Yes
Mango	NO	Yes	Periodic and aperiodic	NO	NO	NO
STNoC	NO	Yes	Sporadic and aperiodic	NO	NO	NO

Table 3.1 Comparison of Mixed-Criticality Requirements

3.1.2 Off-Chip Networks

This section provides a brief discussion on the off-chip communication networks for mixed-criticality systems including the support for fault tolerance.

Ethernet has evolved to offer higher bandwidths and support improved media access control methods. It is now widely used in many application areas. According to [Car16], there is a market share of 38% for industrial Ethernet and an annual growth rate of 20%. Ethernet has become attractive in embedded applications because it is an open standard with many Commercial-Off-The-Shelf (COTS) components on the market. Furthermore, Ethernet-based embedded systems can be seamlessly connected to higher network levels (e.g., business planning and logistics) and integrated into the Internet-of-Things [VD10]. Therefore, Ethernet-based networks are pervasive in many domains such as automotive, avionics and industrial control.

In this dissertation, we will focus on the off-chip communication in a hierarchical architecture using Ethernet-based networks addressing mixed-criticality requirements.

3.1.2.1 Off-chip Communication

Since Ethernet did not support the real-time and the mixed-criticality requirements, Ethernet was extended with several approaches. Examples are audio video bridging based on an IEEE Standard [AVB16], ARINC 664 and TTEthernet. The architecture proposed in this dissertation was inspired by these communication protocols.

Ethernet based on IEEE Standard and Extensions for Time Sensitive Networking

In the 1980s, the IEEE project 802 generated standards for the design and compatibility of hardware components that operated within the Open Systems Interconnection (OSI) physical and data link layers. This standard is the IEEE 802.3 specification. Over time, the family of IEEE 802 standards was extended and changed according to new communication demands including share media (e.g. hubs, switches, physical links).

The family of IEEE 802 standards nowadays uses Virtual Local Area Networks (VLANs) to divide one physical network into multiple broadcast domains based on IEEE 802.1Q [80211]. VLANs are a core protocol required for different systems such as Audio/Video Bridging (AVB). AVB aims to enhance Ethernet with QoS using the priorities of outgoing queues.

This mechanism allows network traffic to be separated from each other without changing physical connections or including additional devices. Furthermore, it is possible that for each virtual network the optimal route between two end nodes is defined. Moreover, the failure of a node or a connection does not necessarily cause the failure of the entire network traffic between two end nodes, because AVB supports the reconfiguration of the virtual networks by using the Multiple VLAN Registration Protocol (MVRP).

Hard real-time traffic is not sported by AVB. The upcoming Ethernet standard time sensitive networking (IEEE 802.1Qbv) [IEE13] will introduce scheduled traffic based on time-triggered communication plans, while also offering run-time reconfigurability and management capabilities.

ARINC 664

Due to the growing complexity of avionic systems, the data transmission in the network has increased. Boeing and Airbus developed a next-generation avionics data bus using the COTS components on the market. This step has resulted in the development of Avionics

Full-Duplex Switched Ethernet (AFDX) based upon IEEE 803.2 Ethernet technology. The AFDX switch is extended with specific functionality to provide a deterministic network with guaranteed services in order to comply with the stringent requirements of Aircraft Data Networks (ADNs). Aeronautical Radio, Inc. (ARINC) standardized it based on Ethernet technology as the standard ARINC 664.

The ARINC 664 standard aims to provide dedicated bandwidth to each communication path in the network and allows the specification of the Quality of Service (QOS) available to each node in the system.

The ARINC 664 standard supports redundant channels, namely two channels transmitting the same data stream at the same time to improve system reliability. Therefore the end node supports the redundancy management to forward only one data stream to the upper layers, and automatically excludes an erroneous data stream from being forwarded.

The ARINC 664 standard ensures a BER as low as 10^{-12} while providing a bandwidth up to 100 Mbps, thereby fulfilling the requirements of new generations of avionics in terms of reliability and available bandwidth.

TTEthernet

Since Ethernet does not support applications with real-time and safety requirements, Ethernet extensions with predictable timing were developed. TTEthernet [Com11] is a real-time Ethernet extension that was standardized by the Society of Automotive Engineers (SAE). It supports message exchanges with bounded transmission delays, low jitter and high channel utilization. TTEthernet establishes a global time base through clock synchronization and provides fault containment for failures of switches, communication links and nodes. In particular, mixed-criticality applications are supported, where safety-critical subsystems (e.g., alarm monitoring functions, active safety functions) and non safety-critical subsystems (e.g., multimedia functions) are combined in a single system [SBct]. For these different types of subsystems, TTEthernet includes suitable communication mechanisms ranging from best-effort messaging with a high channel utilization to predictable real-time messaging based on a time-triggered communication schedule.

A TTEthernet [Obe11] network consists of a set of nodes and switches, which are interconnected using bi-directional communication links. TTEthernet combines different types of communication on the same network. A service layer is built on top of IEEE 802.3, thereby complementing layer two of the Open System Interconnection (OSI) model [Obe11].

TTEthernet supports synchronous communication using so-called *time-triggered frames*. Each participant of the system is configured offline with pre-assigned time slots based on a global time base. This network access method based on TDMA offers a predictable

transmission behavior without queuing in the switches and achieves low latency and low jitter.

The bandwidth that is either not assigned to time triggered frames or assigned but not used is free for asynchronous frame transmissions. TTEthernet defines two types of asynchronous frames: rate-constrained and best-effort frames. *Rate-constrained frames* are based on the AFDX protocol and intended for the transmission of data with less stringent real-time requirements [s0911]. Rate-constrained frames support bounded latencies but incur higher jitter compared to time-triggered frames. *Best-effort frames* are based on standard Ethernet and provide no real-time guarantees.

The different types of frames are associated with priorities in TTEthernet. Time-triggered frames have the highest priority, whereas best-effort frames are assigned the lowest priority. Using these priorities, TTEthernet supports three mechanisms to resolve collisions between the different types of frames [Obe11, Se09]:

- **Shuffling**. If a low priority frame is being transmitted while a high priority frame arrives, the high priority frame will wait until the low priority frame is finished. That means that the jitter for the high priority frame is increased by the maximum transmission delay of a low-priority frame. Shuffling is resource efficient but results in a degradation of the real-time quality.

- **Timely Block.** According to the time-triggered schedule, the switch knows the in advance the transmission times of the time-triggered frames. Timely block means that the switch reserves so-called guarding windows before every transmission time of a time-triggered frame. This guarding window has a duration that is equal to the maximum transmission time of a lower priority frame. In the guarding window, the switch will not start the transmission of a lower priority frame to ensure that time-triggered frames are not delayed. The jitter for high priority frames will be close to zero. Timely block ensures high real-time quality with a near constant delay. However, resource inefficiency occurs when the maximum size of low-priority frames is high or unknown [Ste06].

- **Preemption**. If a high priority frame arrives while a low priority frame is being relayed by a switch, the switch stops the transmission of the low priority frame and relays the high priority frame. That means that the switch introduces an almost constant and a priori known latency for high priority frames. However, the truncation of frames is resource inefficient and results in a low network utilization. Also, corrupt frames result from the truncation, which can be indistinguishable to the consequences of hardware faults. The consequence is a diagnostic deficiency.

The TTEthernet frame format is fully compliant to the Ethernet frame format. However, the destination address field in TTEthernet is interpreted differently depending on the traffic type. In best-effort traffic, the format for destination addresses as standardized in IEEE 802.3 is used. In time-triggered and rate-constrained traffic, the destination address is subdivided into a constant 32-bit field and a 16-bit field called the *virtual-link identifier*. TTEthernet communication is structured into virtual links, each of which offers a unidirectional connection from one node to one or more destination nodes. The constant field can be defined by the user but should be fixed for all time-triggered and rate-constrained traffic. This constant field is also denoted as the *CT marker* [s0911]. The two least significant bits of the first octet of the constant field must be equal to one, since rate-constrained and time-triggered frames are multicast messages.

3.1.3 Fault-Tolerance Networks

Redundant communication architectures based on time-triggered networks and different topologies have been introduced in previous work. Bus topologies with local guardians offer low cost, but limited independence of fault containment regions due to the spatial proximity between host computers and local guardians [ASe03]. A star topology has the advantage of a higher level of independence, since guardians are located at a physical distance from nodes. Furthermore, guardians reshape signals and support additional monitoring services. Ring topologies do not require a central node to manage the connectivity between the nodes. Also, the point-to-point connection between devices with immediate neighbors has advantages w.r.t. installation, reconfiguration as well as identification and isolation of faulty nodes [For07, p. 136]. For example, a braided-ring architecture with superior guardian functionality and complete Byzantine fault-tolerance were introduced [HDPDB05] to achieve high integrity and availability levels similar to SAFEbus but at significantly lower cost.

Prior work has also introduced redundant communication architectures based on existing protocols. For example, extensive results are available for improving the widely used Controller Area Network (CAN) protocol [ISO93]. Previous work has addressed fault-tolerance by active redundancy (e.g., [Ruf97]), using consistent atomic broadcast mechanisms (e.g., [RVA+98, KL99, Liv99]), redundant channels and redundancy management (e.g. [SP07]). ReCANcentrate [BAP05] and CANbids [PBe12] provide fault-tolerance by using star couplers. Likewise, fault-tolerance extensions were performed for industrial Ethernet networks [F. 03], where redundant paths are used to arrive at each node using a ring or mesh topology. A redundant Ethernet system based on a ring topology without switches or hubs using a new topology adaptation network management protocol was introduced in [YKK+06].

Fault-tolerance extension was also explored in different industrial domains. ARINC 664 part 7 [ari05] introduces redundant Ethernet-based communication networks for avionic systems. The integration of different communication protocols with a redundancy concept was presented in [KsH10] for in-vehicle networks.

The existing solutions of fault-tolerant ring architectures do not specifically address mixed-criticality systems where the trade-off between cost and reliability requires fault-tolerance for safety-related communication only.

Furthermore, a communication architecture for mixed-criticality systems needs to support different traffic types and timing models to satisfy the heterogeneous requirements of application subsystems with different criticalities. Safety-critical control loops typically exhibit a regular periodic timing, which can be fulfilled by time-triggered communication. Non safety-critical applications such as multimedia and comfort systems often use less rigid timing models (e.g., sporadic and periodic activities).

3.2 State of the Art: Gateways

In this section, the state-of-the-art of gateways that bridge different networks are given.

The state-of-the-art provides gateways for hierarchical systems with local networks and wide-area networks. The architectural design and implementation of the multi-level internetworking gateways is presented in [BE83]. [DC97] describes the design and implementation of a gateway that links a consumer electronic bus to the Internet or an equivalent WAN. The main functionality of this gateway is to manage, broker and integrate network traffic between these two distinct categories of networks.

A hardware/software co-designed architecture to support the real-time transcoding between IEEE 1394 based digital video and Ethernet-based MPEG4 streaming are introduced in [JLLK07]. An architecture of gateways with Quality-of-Service (QoS) and traffic forecasting aiming to favor application requests with temporal constraints is proposed in [MMT10]. This gateway classifies traffic into six classes with different priorities using IEEE 801.1q.

[FFR+11] focuses on the development of an embedded time gateway for interfacing Simple Network Time Protocol (SNTP) based clients with a Precision Time Protocol (PTP) synchronization infrastructure. A multi-interface sensor network gateway architecture for home automation and other distributed monitoring applications is introduced in [SZZS08].

In the literature, several gateway approaches for integrating mobile ad-hoc networks and the Internet have been proposed [RK03, AER04, OAOK14]. The approaches are generally classified into two-tier and three-tier architectures.

A cloud networking architecture to achieve dynamic and on-demand cloud computing services using different underlying networking technologies is proposed in [MZZM13]. Moreover, the authors present a cloud networking gateway manager to enable networking of distributed cloud resources by authorized customers and to provide network control and configuration capabilities.

An Internet-of-Things (IoT) architecture that allows real-time interactions between mobile clients and legacy devices (e.g., sensors and actuators) via a wireless gateway is proposed in [DBN14]. This architecture provides to clients the support for dynamic discovery of Machine-to-Machine (M2M) devices and endpoints. It manages connections with non-smart devices connected over Modbus and associated meta-data to sensor measurements using the Sensor Markup Language (SenML).

The gateways presented in this dissertation go beyond the state-of-the-art by providing an off-chip/on-chip gateway architecture for networked multi-core chips that provides a solution for mixed-criticality systems with different timing models, fault isolation and real-time guarantees.

3.3 State of the Art: Distributed Scheduling

Since time-triggered networks are a fundamental technology for the partitioning in a mixed-criticality architecture and since time-triggered networks depend on static schedule tables, distributed scheduling algorithms are presented for computing these tables.

The optimal scheduling of a distributed real-time system is an NP-complete problem [EDPP00]. There are several approaches to tackle the scheduling problem in time-triggered systems. The static scheduling and partitioning of processes, and the allocation of system components are introduced in [Ben96, PP92] based on Mixed Integer Linear Programming (MILP). The main drawback of this approach is that the runtime of solving the MILP model grows quickly with the number of processors and services. Therefore, heuristics have been introduced such as list scheduling heuristics using different priority criteria (e.g., [SS01, KA96]), branch-and-bound algorithms (e.g., [NYC06, EKP$^+$98]) and heuristics based on neighborhood search (e.g., [TSX00, AB06, Esw09]). Popular meta-heuristics in the neighborhood search are simulated annealing, tabu search and genetic algorithms.

Many researchers have provided solutions for offline scheduling of distributed systems with bus-based multi-core processes. The scheduling problem of TTEthernet systems is addressed in a few research results. [Ste10] provides an algorithm to handle periodic traffic over multiple hubs based on TTEthernet using the Satisfiability Modulo Theory (SMT) solver. The static scheduling for integrating time-triggered and event-triggered traffic is introduced

in [Ste11]. In [TSPS12] the authors used the tabu search to schedule periodic traffic and to minimize the end-to-end delay of sporadic messages.

However, none of the previous results for time-triggered networks address the scheduling of the services with dependencies, while at the same time handling periodic and sporadic traffic. Since mixed-criticality systems support different timing models, we present a new scheduling algorithm for periodic and sporadic traffic based on neighborhood search. We minimize transmission delays and execution times, while also supporting service dependencies.

3.4 Research Gap in the State of the Art

A mixed-criticality architecture for networked multi-core chips with support for safety, real-time performance, fault isolation and system integrity is missing in the state-of-the-art. Existing architectures do not support reliable and predictable communication services, such as timing guarantees (e.g., minimal jitter, bounded delay, best effort), using heterogeneous communication systems with different criticality, different timing models and different models of computation.

Moreover, the integration of on-chip and off-chip networks with different protocols into a coherent embedded architecture for networked multi-core chips taking into account complete segregation to establish mixed-criticality support on the network level is not available.

In addition, architectural support for fault-tolerance based on the boundary conditions of mixed-criticality systems (e.g., heterogeneous models of computation, non-deterministic subsystems, and different timing models) is missing.

In addition to that, mixed-criticality scheduling has to be considered at the off-chip network and the on-chip network. Many existing scheduling algorithms focus on the off-chip message scheduling only, without support for integration with the chip-level and end-to-end considerations.

We advance the state-of-art and introduce an integrated off-chip and on-chip communication system with a reliable and temporally predictable communication infrastructure that supports different criticality levels, QoS requirements (e.g., bandwidth, latency, reliability) and communication modes (e.g., streaming, cyclic control messages, event notifications). In addition, we present an off-/on-chip gateway that supports the mixed-criticality services, fault isolation and real-time guarantees.

Likewise, we establish redundant communication paths, traffic shaping and exploit QoS parameters in order to improve reliability and temporal predictability for end-to-end communication.

Furthermore, a fault injection environment is introduced to support the injection of different fault types of the timing and value failures into networks to evaluate the system's reliability. The injected faults are derived from certification standards such as IEC 61508.

Finally, we advance the state-of-the-art of mixed-criticality scheduling to address the scheduling of the services with dependencies, while at the same time handling periodic and sporadic traffic.

Chapter 4

System Model of Multi-Core Chips Interconnected by Real-Time Ethernet

This chapter presents a mixed-criticality architecture based on networked multi-core chips that integrates applications at different levels of criticality on the same platform. The system architecture aims at defining the platform services of the mixed-criticality system and how these services are applied in the architecture models.

The first part of this chapter describes a conceptual system model of a platform that consists of networked multi-core chips. In addition, we describe the platform services of the system model. In addition, we present the platform services of the mixed-criticality architecture. We use fundamental services of a mixed-criticality architecture (global time, communication with heterogeneous traffic types, fault containment, and fault tolerance) as the architecture's platform services. Subsequently, a concrete system model for mixed-criticality systems based on networked multi-core chips is described.

4.1 Conceptual Architecture Model

The system model illustrated in Figure 4.1 consists of a physical and a logical model as well as a mapping to platform services which are described in the following subsections.

4.1.1 Physical and Logical System Models

The overall system (cf. Figure 4.1) is physically structured into a set of nodes that are interconnected by a real-time communication network in a corresponding network topology (e.g., mesh, star, redundant star, ring). An off-chip/on-chip gateway serves as the bridging point between chip-level and off-chip communication.

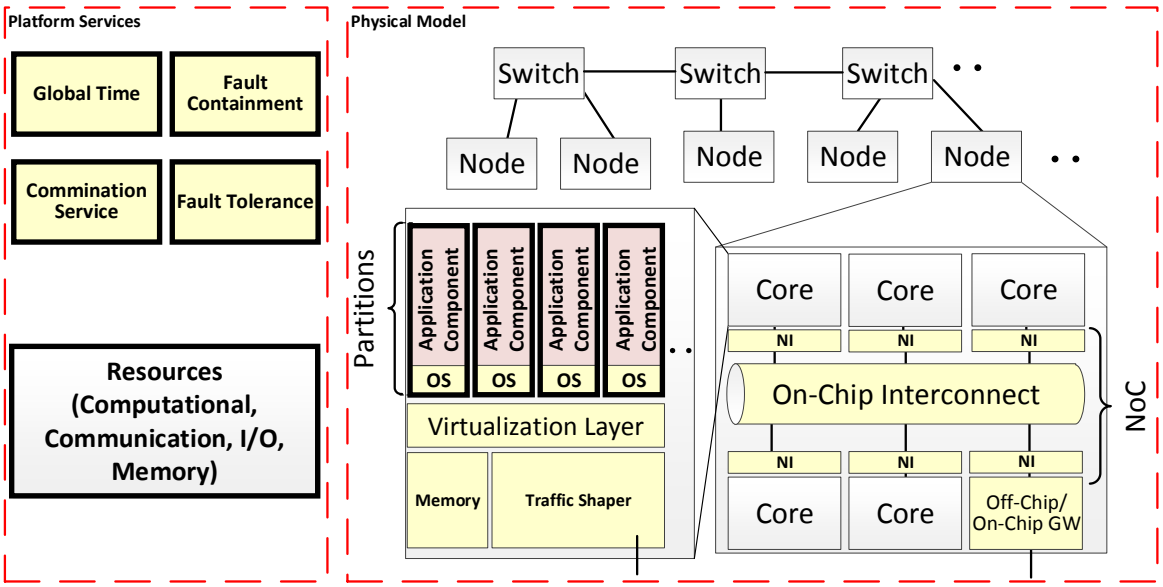


Figure 4.1 Physical and Logical System Model

Each node is a multi-core chip containing cores that are interconnected by a NoC. Each core has a Network Interface (NI) which is the connecting point between an IP core or several IP cores and the on-chip interconnect. According to the application and architecture requirements, the NoC has a corresponding topology for interconnecting the cores and the on-chip routers (e.g., mesh, torus, folded torus, hypercube, octagon).

The IP cores within a core can run a hypervisor that establishes partitions, each of which executes a corresponding software component (or component for short). The hypervisor establishes time and space partitioning, thereby ensuring that a software component cannot affect the availability of the computational resource in other partitions (e.g., time and duration of execution on IP core, integrity and timing of memory).

An off-chip/on-chip gateway is responsible for the redirection of messages between the NoC and the off-chip communication network. The off-chip communication network consists of a set of switches, that establish connections between the nodes and other switches.

From the logical point of view, the system model is structured into criticality levels. Each criticality level belongs to multiple application subsystems. Each of these applications is assigned to a safety integrity level such as Class A to E in avionics, ASIL A to D in automotive and SIL1-4 in multiple domains according to IEC-61508. Examples of application subsystems are steer-by-wire and brake-by-wire in the car that belong to the highest criticality level (ASIL D). An application subsystem can be further subdivided into components, which interact by the exchange of messages via ports. Each component provides services to its environment. The specification of a component's interface defines its services, which are the

intended behavior as perceived by the transmission of messages as a response to inputs, state and the progression of time. Three types of messages are distinguished based on their timing:

1. Periodic messages represent time-triggered communication. Their timing is defined by a period and phase with respect to a global time base.

2. Sporadic messages represent rate-constrained communication with minimum interarrival times between successive message instances.

3. Aperiodic messages have no timing constraints on successive message instances and no guarantees with respect to the delivery and the incurred delays.

The logical and physical system models for networked multi-core chips are defined with corresponding architectural services that are essential for the evolvability, scalability and complexity management of mixed-criticality systems. These services allow abstracting from the platform technology such as network protocols and types of processor cores. The communication services support the timing models (i.e., periodic, sporadic and aperiodic) as well as temporal/spatial partitioning based on the enforcement of time-triggered schedules, rate-constraints and permitted address information. The gateway services address the requirements for resolving the various communication services that are required for bridging between chip-level and cluster level communication for hierarchical system structures.

The global time services fulfill the demand for a consistent global time base in a system of networked multi-core chips with bounded precision and bounded accuracy. This global time is the foundation for the temporal coordination of activities and the establishment of a deterministic communication infrastructure.

The fault hypothesis is important to satisfy the requirements on fault detection, containment and masking. In addition, we introduce fault-tolerance services to mask component failures according to the fault hypothesis.

4.1.2 Platform Services

The proposed architecture supports different services such as global time, communication with heterogeneous traffic types, fault containment and fault tolerance. These services are the foundation for the development of mixed-criticality systems.

4.1.2.1 Global Time

In a distributed system where each node has its own local clocks, the clock synchronization problem appears. The aim of the clock synchronization service [1] is to establish a global time

[1] The overall clock synchronization idea has been discussed with the DREAMS partners in [Con14].

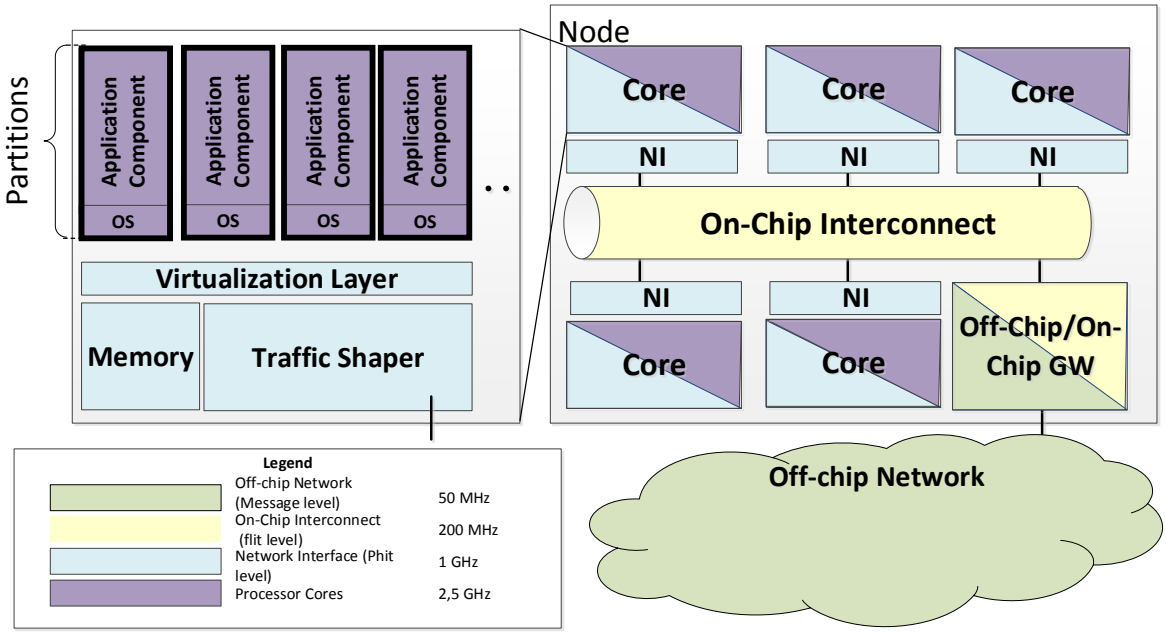


Figure 4.2 Example of Different Clock Speeds at Different Parts of the System

between the nodes and switches that have local clocks. The global time is an assumption that the clocks of all nodes and switches have “about the same value” at “about the same points in real-time”.

The concept of the global time gives the ability to establish the temporal coordination of activities, a deterministic communication infrastructure and a relationship between timestamps from different components.

The clock synchronization service is also responsible for establishing the global time in a system, which supports different clock domains. As shown in Figure 4.3, different parts of the system can operate at different clock speeds and components can include an arbitrary number of local clock domains, which are not visible outside of the core.

The goal of the clock synchronization service is to keep the time of the local clocks in close relation to each other or to a reference clock. In order to achieve this goal, the clock synchronization service performs two different modes of operation: a startup and keeping clocks synchronized during normal operation.

The startup mode establishes the initial synchronization between nodes and switches by negotiating and agreeing on the initial synchronization point [Obe11]. This initial synchronization is an essential prerequisite for the realization of a deterministic communication, which performs the temporal coordination of all communication activities using the global time. In fact, clock synchronization algorithms for most applied deterministic protocols

such as FlexRay and TTEthernet are based on the assumption of initially synchronized local clocks as established by a startup mode.

The startup mode does not depend on the individual power-on times of the participating nodes and switches. Therefore, most startup modes are generally divided in two phases: a "coldstart" and an "integration" phase.

In the coldstart phase, a new synchronized time-base is established by the nodes and switches rather than integrating to an existing one. The authors in [Obe11] discussed different coldstart algorithms based on different communication protocols.

The integration phase concerns the nodes and switches that are joining an already synchronized system. After the coldstart phase, the nodes and switches periodically exchange integration messages. Those messages are read by an integrating node and used to initialize their local clocks.

In the normal operation, each node and each switch perform the following three phases as stated in [Obe11]:

Phase 1: Collection of clock time values. In phase 1, a node or a switch receives the time values of other clocks actively participating in the synchronization where this time could be equal to the actual time or different.

Phase 2: Calculation of correction value. The purpose of this phase is bringing the collected clock time values from phase 1 closer together by computing a correction value of the collected clock time values.

Phase 3: Clock correction. In phase 3, a node or a switch uses the calculated correction value of phase 2 as a reference clock value.

At the chip level, a multi-core processor generally provides multiple clock domains to support clocking down of individual IP blocks as part of power management or the heterogeneous IP blocks require different speeds (e.g., high-clocked special purpose hardware and a slower general purpose CPU). The global time base at the chip-level embodies an independent clock domain, which typically is established through the low-frequency macro tick clock signal in the chip.

The gateway provides the synchronization service between the on-chip global time and the off-chip global time base using rate correction in combination with overflow time intervals. Since the on-chip global time base is typically faster than the off-chip global time base, it is supposed to be synchronized, if each N rising edge of the on-chip global time is associated with a rising edge of the off-chip global time base. However, if the on-chip global time base runs faster, after the N cycles, the next rising edge waits until the rising edge of the

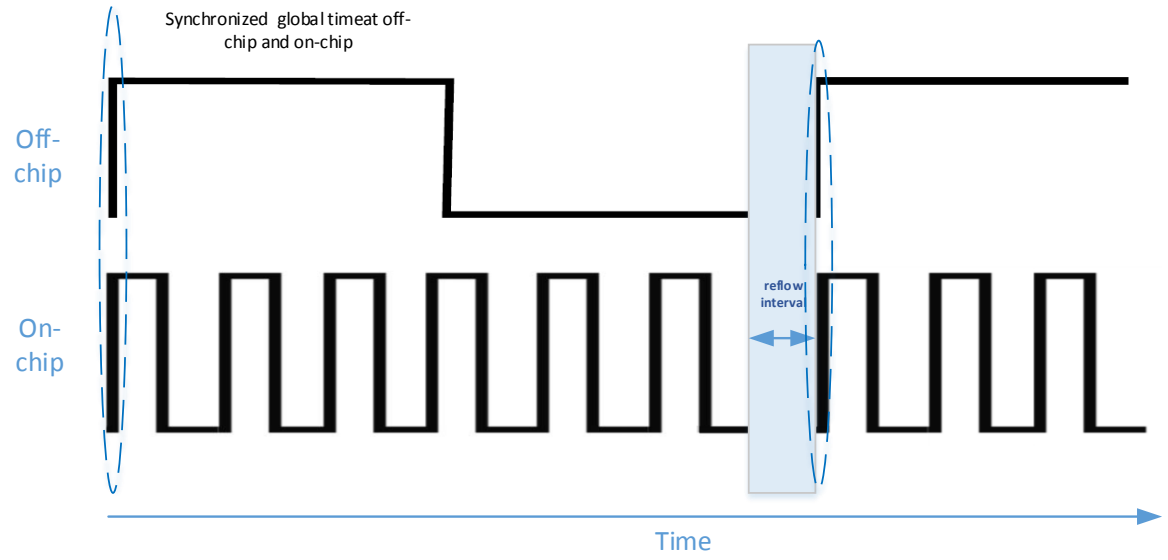


Figure 4.3 Example of Synchronized Global Time Based at On-chip and Off-chip

off-chip global time base. This interval is called reflow interval that determines the tolerable deviation between the rates of the off-chip and on-chip global time base. Figure 4.3 shows an example of this synchronization, where the on-chip global time base is six times faster than the off-chip global time base.

4.1.2.2 Communication with Heterogeneous Traffic Types

Mixed-criticality systems differ not only in the safety assurance levels but typically also exhibit varying temporal requirements for the underlying platform. Many functions of the highest criticality levels realize cyclic control services. For example, time-triggered communication protocols are well suited for safety-critical control functions [Obe11]. Functions of lower criticality often employ less restrictive timing models. Sporadic communication with rate-constraints offers higher potential for resource sharing and bounded latencies, but the higher variability of communication latencies. Aperiodic communication activities without temporal guarantees are suitable for non safety-relevant functions.

Therefore, the system model supports three types of criticality according to the traffic types:

- Time-triggered transmission of periodic messages: They are temporally defined by a period and phase with respect to a global time base and they can be assigned the highest priority to resolve conflicts with the other traffic types in order to minimize the latency jitter.

- Rate-constrained transmission of sporadic messages: Sporadic messages represent rate-constrained communication with minimum interarrival times between successive message instances.

- Best-effort transmission of aperiodic messages: They have no timing constraints on successive message instances and no guarantees with respect to the delivery and the incurred delays.

In the proposed system, the concept of Virtual Links (VLs) is used to realize bandwidth partitioning and hide the physical system structure of the platform from the components. The timing and reliability of the VL are determined by the properties of the underlying physical networks.

A VL is an end-to-end multicast channel between one sender component and multiple receiver components.

Time-triggered VLs serve for the time-triggered transmission of periodic messages at the specified period and phase with respect to a global time base. Rate-constrained VLs establish the transport of sporadic messages with minimum interarrival times. A rate-constrained VL also has a priority that determines how contention with other rate-constrained VL is resolved. Rate-constrained communication guarantees sufficient bandwidth allocation for each transmission with defined limits for delays and temporal deviations. Aperiodic messages do not require VLs, but are subject to a connectionless transfer.

In the proposed system model, we use the following mechanisms to resolve the contention between periodic, sporadic and aperiodic messages for VLs.

Timely block for periodic messages: Contention can be resolved by *timely block* of the physical links for the conflict-free traversal of periodic messages through the networks. Timely block ensures that a network is free when a periodic message arrives. In a time-triggered schedule, the periods and phases of other periodic messages can be aligned in such a way that there is no contention. For sporadic and aperiodic messages, a guarding window is required prior to the transmission of each periodic message. The guarding window prevents the start of a transmission of a sporadic or aperiodic message, if it would delay the subsequent periodic message.

For a VL, where the transmission of a periodic message occurs on multiple successive networks, the transmission of the periodic message and the associated guarding windows can be phase-aligned on these networks.

Each network has a time-triggered schedule that determines the time intervals of guardian windows and when a message received from on network is relayed to another network. Hence,

timely block requires an alignment between the time-triggered schedules of the on-chip and off-chip networks.

Shuffling for periodic messages: Shuffling is an alternative to timely block with a more efficient utilization of the network bandwidth. Shuffling avoids the overhead of the guarding windows where no messages can be transmitted.

In case of shuffling, periodic messages are assigned higher priority than sporadic and aperiodic messages. If contention occurs on a non-preemptive network, a periodic message is delayed by at most one sporadic or aperiodic message that is already in transmission. In the VL there can be an additional delay of another message with maximum size on every network of the end-to-end channel.

Interference between periodic messages can be avoided, when the time-triggered schedule separates any two periodic messages by a sufficiently large time interval. This timer interval must be at least as large as the transmission jitter, which is equal to the transmission duration of a maximum size message.

Shuffling for sporadic messages: Shuffling can be used to limit the effect imposed upon sporadic messages from lower priority sporadic and aperiodic messages. The maximum effect is the delay for transmitting one lower-priority message of maximum size. In combination with the rate-constraints for the higher priority messages, upper bounds for the transmission delays of sporadic messages can be specified.

4.1.2.3 Fault Containment

The fault containment between components is one of the key properties in mixed-criticality systems in order to improve robustness, attain clear integration responsibilities and enable modular certification. As we discussed in section 2.2, a Fault Containment Region (FCR) is defined as a set of subsystems that shares one or more common resources that can be affected by a single fault and is assumed to fail independently from other FCRs. The fault can be divided into two categories, namely design faults and physical faults. Based on these types, one can distinguish corresponding FCRs.

Since the system model supports safety-critical real-time systems, the system must have a failure rate with regard to critical failure modes that conforms to the ultra-high reliability requirement. In these systems with ultra-high reliability, a maximum rate of critical failures of 10^{-9} per hour is demanded. Therefore, design faults in hardware and software - that are introduced during the development of the platform and the application - are demanded to

Table 4.1 Fault Containment Regions for Design & Physical Faults

Fault Containment Region	**Containment Coverage (Correlated Failures per Hour)**
Partition	$< 10^{-9}$
Multiple partitions containing the same application component or the same guest OS	$< 10^{-9}$
Application cores with the hypervisor layer	$< 10^{-9}$
Cores	$< 10^{-9}$
Nodes	$< 10^{-9}$

lead to less than 10^{-9} correlated failures per hour for the corresponding FCRs (See Table 4.1)

A physical fault affects physical resources, such as mechanical or electronic parts. To form a fault containment boundary around a collection of hardware elements, one must provide independent power and clock sources and additionally electrical isolation and spatial separation. These requirements make it impractical to provide more than one FCR within a node at a safety-critical rigor (at a containment coverage with a probability of correlated failures of 10^{-9} failures per hour). We also regard each switch with the corresponding physical links to the nodes as a FCR. For example, a central guardian of a time-triggered network (e.g., TTEthernet switch) serves as a FCR [CMFC+98].

For physical faults, the hardware approach can provide a certain containment coverage by providing spatial separation of the cores, multiple clock domains and pin-out (e.g., grounding) on the chip layout (e.g., for SEEs [BPH98]). These on-chip FCRs for physical faults (i.e., cores) work only at single chip failure probabilities (e.g., around 10^{-5} to 10^{-6} correlated failures per hour [AC03]). Therefore, additional fault containment at off-chip level is required in ultra-dependable systems.

Physical fault containment and design fault containment are orthogonal properties. Physical fault containment does not assure design fault containment and vice-versa. For instance, one may use two separated chip processors (two FCRs for physical faults) to implement a function but both can fail simultaneously due to a single design fault of the software. In the same way, a NoC can assure design fault containment for two independent operating systems within the same chip and a single physical fault can make both fails.

4.1.2.4 Fault Tolerance

Fault tolerance is the property that enables a system to continue its operation normally or with a reduced quality, but without causing total breakdown of the whole system while it has

one or more faults within its components. A FCR is introduced in order to delimit the impact of a fault. As the fault-error-failure chain is introduced section 2.1, the fault that causes the failure can be categorized as (i) design fault of the hardware or software of the FCR, (ii) operational fault of the FCR and (iii) faults at the system-level. The latter type of fault is a failure of an FCR, which could cause a fault in another FCR via a sent message that deviates from the specification. If the transmission instant of the message is not in agreement with the specifications, we speak of a message timing failure. A message value failure means that the data structure contained in a message is incorrect.

Such a fault can be masked using fault tolerance mechanisms, and the most common mechanism is N-Modular Redundancy (NMR) [LA90]. *N* replicas receive the same requests and provide the same service. The output of all replicas is provided to a voting mechanism. The voting mechanism determines if the result is correct by selecting one of the results (e.g., based on the majority) or by transforming the results to a single one (average voter). The most frequently used *N*-modular configuration is Triple Modular Redundancy (TMR).

We denote the number of the replicated components and a voter as a Fault-Tolerant Unit (FTU). Replica determinism has to be supported by the mixed-criticality architecture to ensure that each FTU produces the correct outputs in defined time intervals. The deterministic communication (e.g. time-triggered communication) addresses key issues of replica determinism. Typically, the deterministic communication supports replica determinism by using the global time based in conjunction with predefined communication and computational schedules. The predefined schedules determine the points in time for triggered computational activities after the last message of a set of input messages has been received by all replicas of a FTU. The alignment of communication and computational activities on the global time base ensures temporal predictability and avoids race conditions.

4.1.3 Fault Hypothesis

The fault hypothesis specifies assumptions about the types of faults, the rate at which components fail and how components may fail [OP06]. The fault hypothesis is a central part in any safety-relevant system and provides the foundation for the design, implementation and test of the fault-tolerance mechanisms [Obe12].

The fault assumptions of the system model are based on IEC-61508-2, according to which transmission errors, deletion, corruption, delay, repetitions, masquerading and insertion need to be addressed [IEC10e].

Failure modes of FCRs are defined according to the effects appearing at the service interface of an FCR. The failure mode is independent of the actual cause or rate of the failure. The following failure modes are assumed:

- *Component crash:* The crash failure occurs when the node or the switch exhibits a permanent fault and produces no outputs.
- *Link failures:* The link failure occurs when the link exhibits a permanent or transient failure and fails to redirect a message. In combination with the component crash, this failure corresponds to the transmission error according to IEC-61508-2.
- *Omission:* An omission failure is a transmission failure where a sender is not able to generate a message and/or a receiver is not able to receive a message. This failure corresponds to the deletion according to IEC-61508-2.
- *Corruption:* This failure involves changes to the original data (e.g., due to EMI disturbances).
- *Delay:* Faulty nodes or switches can delay the transmission of messages.
- *Babbling idiot:* This failure occurs when a node or a switch starts sending untimely messages (e.g., insertions according to IEC-61508-2), possibly generating a high traffic load by generating more messages than specified.
- *Masquerading:* A masquerading failure is an erroneous node that assumes the identity of another node. In case of periodic time-triggered and sporadic rate-constrained communication, a faulty node sends messages with the incorrect virtual link identification. For best effort messages, the node will send messages with an incorrect MAC address.

Another important parameter is the maximum number of failures. This parameter of the fault hypothesis denotes the maximum number of FCR failures, which must be handled by the system. The maximum number of failures depends on the failure rate and the recovery interval of FCRs. We assume a single failure only, which is a prevalent assumption in many present day safety-critical systems (e.g., TTA [Kop04a]). A fault hypothesis with this assumption is also frequently denoted as a "single fault hypothesis" (from a system-level point of view).

4.2 Concrete Architecture Model

In this section, a detailed description of the architectural building blocks for the on-chip and off-chip networks is given. Furthermore, interfaces, data and control flows between the building blocks are discussed.

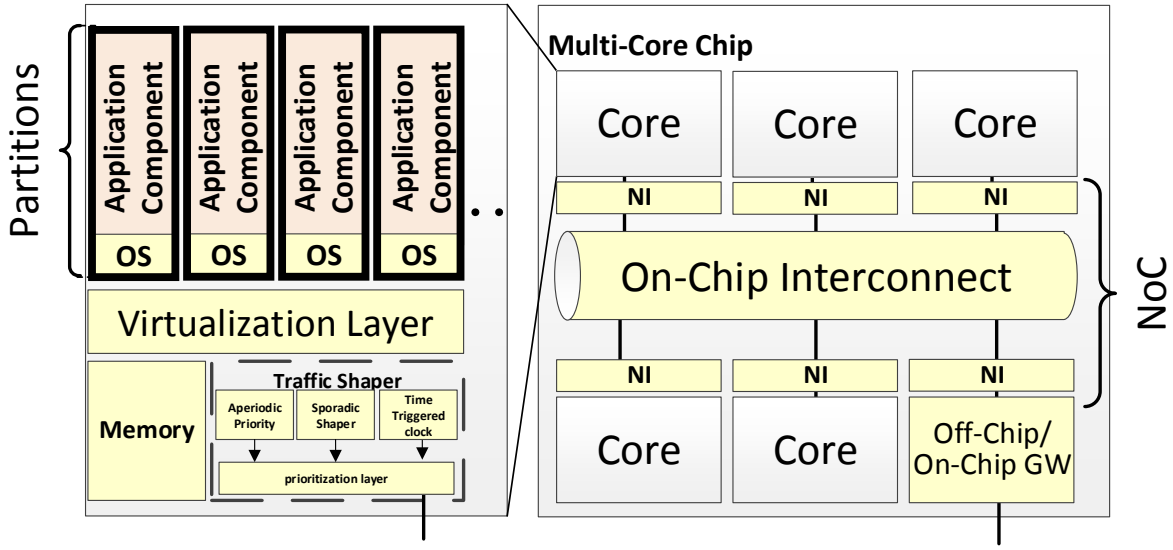


Figure 4.4 System Model of the Multi-core Chip

4.2.1 On-Chip Architectural Building Blocks

The overall architectural building blocks of a multi-core chip are depicted in figure 4.4. This section provides detailed information about these building blocks that consist of three groups: cores with application components, the on-chip network and the gateway.

4.2.1.1 Cores with Application Components

The architecture model supports different criticality levels for the application. Based on the criticality levels, we support different communication types to transfer the messages. The safety-critical applications require a deterministic behavior with a priori known temporal behavior. Therefore, predetermined time slots are used for the transfer of periodic messages. The non-safety applications use the event-triggered communication either via sporadic or aperiodic communication. Event-triggered messages are transmitted in those time intervals where the communication medium is not needed for the transmission of periodic messages. The separation of sporadic messages is enforced by the minimum interarrival times, while aperiodic messages are sent without temporal restrictions.

In addition, the architecture ensures time and space partitioning for the computational resources by using partitioning OSes and virtualization layers for the sharing of processor cores among mixed-criticality applications, including safety-critical ones.

In the architecture model, each application components executes application services that can send messages. The virtualization layer is also responsible for mapping the application messages to the correct communication type, i.e., either periodic, sporadic or aperiodic.

A traffic shaper is responsible for establishing temporal and spatial partitioning for the communication by enforcing the temporal parameters of the above-mentioned communication types. The traffic shaper is responsible for handling incoming messages from the application and forwarding them to the NoC, which assures the real-time guarantees for the safety critical messages and ensures that there is no collision between messages.

Based on the communication type, the traffic shaper performs different mechanisms for the transfer of messages to the NoC. Let us illustrate how these mechanisms work using the example in Figure 4.4 with three applications X, Y and Z for periodic, sporadic and aperiodic messages.

Transmission of Periodic Messages by Application Z

The core comprises several partitions, each of which can execute an application using an operating system or bare-metal. In mixed-criticality systems, the virtualization layer such as a hypervisor is widely used in order to provide subsystems of different criticality levels with isolated partitions.

Let us assume that application Z is a safety-critical application and sends periodic messages to the virtualization layer. The virtualization layer passes these outgoing messages to the time-triggered clock layer, since the configuration parameters in the virtualization indicate that application Z is periodic time-triggered. In the time-triggered clock layer, the incoming message is buffered in a corresponding virtual-link buffer. Finally, the message is sent to the prioritization layer at the time specified in the static communication schedule.

The prioritization layer contains three queues, namely one for each traffic type. The message that comes from the time-triggered clock layer is directly sent to the lower layer in case no message is being transmitted. Otherwise, the message is put into the queue for periodic messages.

The prioritization layer sends the messages to the NoC according to their priority. Periodic messages have the highest priority, whereas aperiodic messages are assigned the lowest priority.

Transmission of Sporadic Messages by Application Y

For sporadic communication, the configuration parameters in the virtualization layer identify the outgoing messages from application Y as sporadic, therefore the virtualization layer passes the messages to the sporadic shaper layer (cf. Figure 4.5). The sporadic shaper layer guarantees a Bandwidth Allocation Gap (BAG) interval between two consecutive instances of sporadic frames on the respective virtual link. The incoming sporadic messages are queued

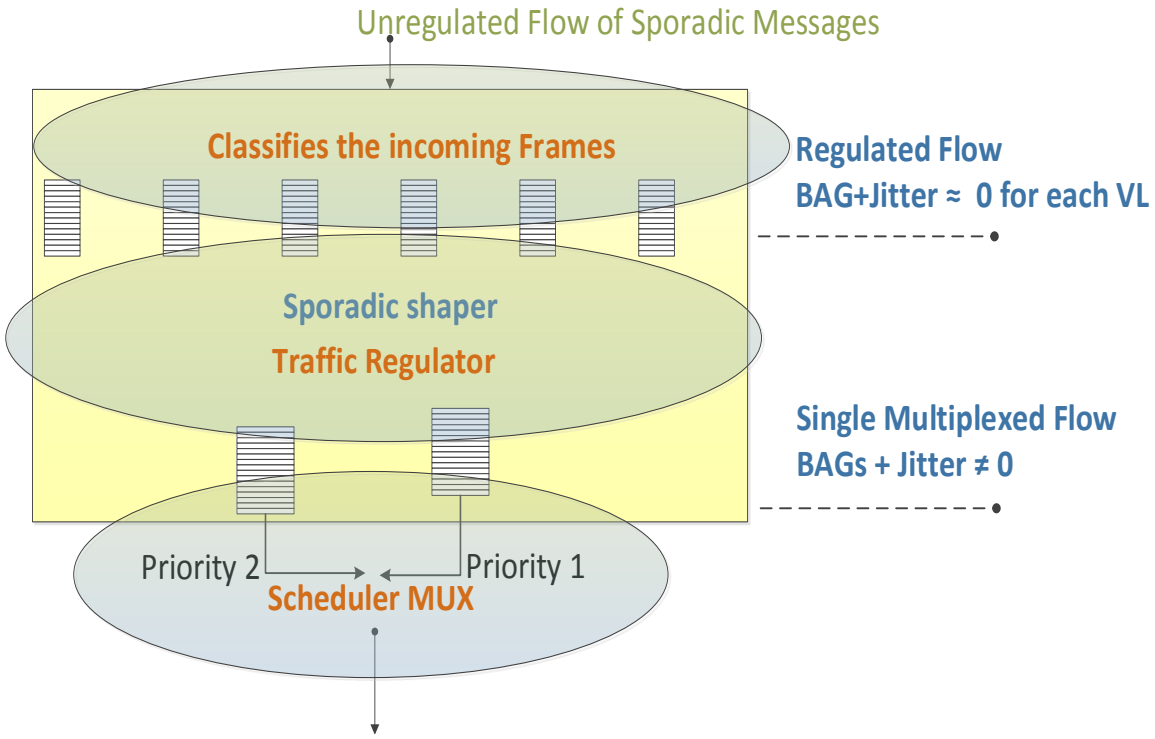


Figure 4.5 Modules of the Scheduled Flow Control Mechanism for Sporadic Messages

at the sporadic shaper layer using a dedicated queue for each virtual link. In the next step, the message is carried from its queue to one of two multiplexed queues controlled by the BAG. The multiplexed queues are used to multiplex the message flow that comes from the virtual links. Two multiplexed queues in the sporadic shaper layer serve for the distinction between two priority levels. Finally, the *controller layer* puts the messages from the sporadic shaper layer into the *sporadic queue* in the controller layer.

Transmission of Aperiodic Messages by Application X

Thirdly, an application X sends event messages without any restriction in the temporal domain. In our architectural model, two priority levels are distinguished for aperiodic messages. The virtualization layer passes the outgoing messages to the *aperiodic priority layer*, where the messages are queued in one of the aperiodic priority queues. Then, the message is sent to the lower layer according to the priority. In the controller layer, the incoming messages are queued in the aperiodic queue and sent to the NoC.

4.2.1.2 On-Chip Network

The multi-core chip contains cores that are interconnected by NoCs. In this dissertation, the architectural models of the core - that were presented in the previous section - build upon NoCs that contain two components: routers and network interfaces. We distinguish between two types of NoCs; priority based NoCs and NoCs with resource reservation.

In a priority based NoC such as the STNoC [DCC+11], the concept of virtual channels is used to realize priorities within the NoC. In order to support the mixed-criticality requirements, messages can be mapped to different priorities. For instance, the underlying NoC shall support at least two different priorities to bound or prevent effects of non safety-critical applications onto safety-critical ones. The highest priority is dedicated to the messages of the safety-critical applications.

The resource reservation NoC provides guaranteed services with predefined spatial/temporal allocations of messages to routers. An example of such a NoC is the TTNoC [OEHK08] which introduces a predictable on-chip interconnect with inherent fault isolation to facilitate the seamless integration of independently developed components, possibly with different criticality levels. However, while the TTNoC is suitable for the transmission of messages from the safety critical applications, it does not support the transmission of sporadic and aperiodic event-triggered messages.

Another resource-reservation is Æthereal [GDR05]. The Æthereal NoC introduces the concept of contention-free routing for guaranteed services with guaranteed throughput and bounded latency in addition to best-effort services. The contention-free routing guarantees a certain level of performance for a communication using resource reservation in the NoC [GDR05]. In contention-free routing or pipelined time-division multiplexed circuit switching, a connection is established by reserved wires and buffers for certain points in time. In order to map the presented architectural model to the Æthereal NoC the safety critical application would have to be mapped to guaranteed services, while the sporadic and aperiodic messages can use the best-effort services in the underlying NoC.

4.2.1.3 Gateway

Off-chip/on-chip gateways are used to establish the end-to-end communication over heterogeneous and mixed-criticality networks. The connection between off-chip and on-chip networks is established through gateways as illustrated in Figure 4.6. The gateway consists of the gateway core functionality, network interfacing and Media Access Controls (MACs).

The gateway core is responsible for processing incoming messages based on timely redirection, protocol conversion, monitoring and configuration services. The network interfacing

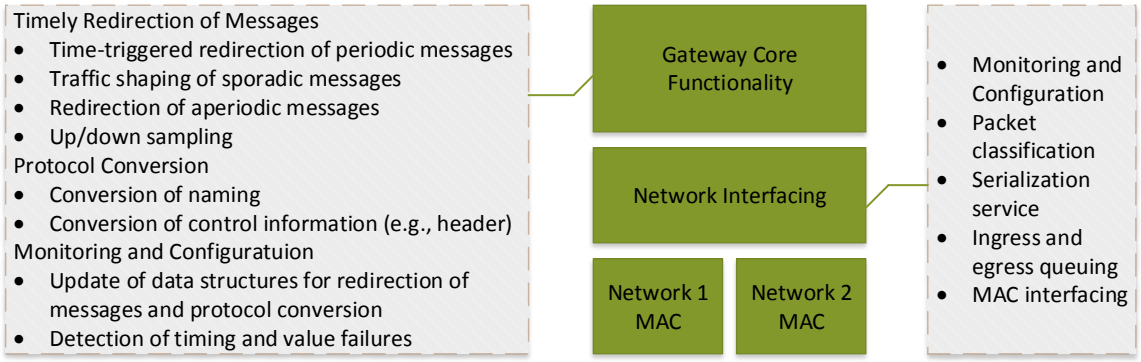


Figure 4.6 System Model of a Gateway

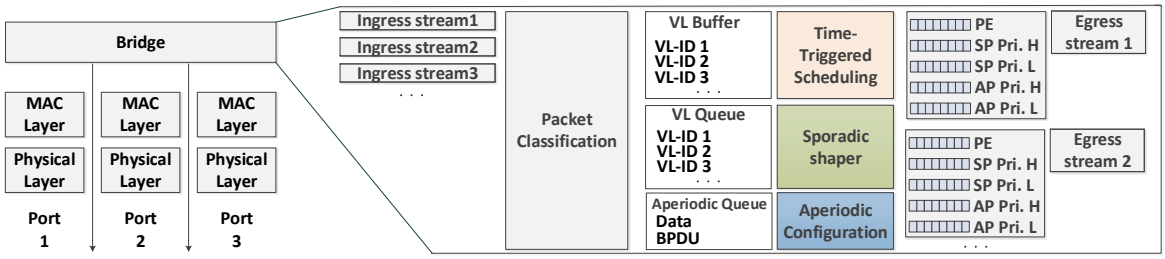


Figure 4.7 System Model of a Switch

provides the interface between the MAC and the gateway core. Furthermore, classification and serialization of the packets is performed in the network interfacing. In order to realize fault-tolerance, the gateway can include multiple network MACs. Each network MAC connects the gateway to either an off-chip network or an on-chip network. In case of network redundancy, multiple network MACs are required. Thus, the network interfacing is responsible for merging identical incoming messages and duplicating outgoing messages to be sent to different MACs. We present in Chapter 6 more details of the off-chip/on-chip gateway services and functionalities.

4.2.2 Off-Chip Architectural Building Blocks based on TTEthernet

This section provides detailed information on off-chip architectural building blocks, which are composed of two groups: switches and nodes.

4.2.2.1 Switch

Standard Ethernet does not guarantee either the real-time capability or bounded transmission delays. Therefore, it is not suitable for the real-time and safety aspects in mixed-criticality systems. In the architectural model of the proposed switch, we assume temporal and spatial

partitioning that guarantees predictable timing of periodic time-triggered messages and bounded latencies for sporadic rate-constrained messages.

The switch extends the standard Ethernet switch by adding the time deterministic message transfer capabilities, while retaining full compatibility with the requirements of IEEE 802.3.

Such a system simplifies the design of complex distributed systems and applications as well as processing of critical (periodic, sporadic) and non-critical Ethernet traffic (aperiodic). High-priority periodic messages are routed through the switch according to a predefined schedule with fixed latency and a transmission jitter in the sub-microsecond range. Sporadic messages are routed based on the AFDX protocol. All other messages are forwarded based on standard Ethernet when bandwidth is available.

As shown in Figure 4.7, the switch consists of multiple *ports* and a *bridge*. Each *port* contains a physical layer and a MAC layer. The physical layer is built according to IEEE 802.3. The MAC layer is based on IEEE 803.2 with the extensions explained in the following.

The MAC layer checks the validity of the destination address for an incoming message and distinguishes between traffic types based on connection-oriented and connectionless communication. The connection-oriented communication is used for the periodic and sporadic messages. Aperiodic messages use the connectionless communication. We regard a message as a tuple with the following elements:

- Message in connection-oriented communication: <type, VLID, data>
- Message in connectionless communication: <type, destination address, data>.

For example, these traffic types can be realized in TTEthernet as follows. The constant field is extracted from the destination address using the bit mask $0xffffffff0000$. In case the constant field has the predefined value of the Critical Traffic marker (CT marker), this message is either time-triggered or rate-constrained. Otherwise, the message will be regarded as aperiodic. The switch distinguishes between periodic and sporadic messages using the EtherType value. The IEEE Standardization Authority has assigned the values 0x88d7 [SAE11] and 0x0800 [ECR05] for the EtherType fields of time-triggered and rate-constrained messages.

Figure 4.7 shows the block diagram of the bridge. The task of the bridge is to handle and forward ingress messages to the egress ports depending on the traffic type. The bridge model contains five layers: the *bridge classification layer* that determines the traffic type of an ingress message, the *TT scheduling layer* that processes the periodic messages, the *Sporadic shaper layer* that handles the rate-constrained messages, the *Aperiodic configuration layer* that handles the aperiodic messages, and the *egress port layer* for realizing shuffling and timely block.

Processing of Periodic Messages

For each periodic message, a static communication schedule defines three parameters: period, phase and maximum frame size. Periodic messages are scheduled to be transmitted periodically, where the phase parameter defines the exact start time relative to the start time of the period. In each period multiple periodic messages with different phases are supported. In addition, the static communication schedule defines the ingress and egress ports of each time-triggered message, as well as the buffer size in the bridge.

When a periodic message arrives at the bridge from the MAC layer, the bridge classification layer checks the integrity and validity of the message. The integrity checking verifies that the message has the correct size and arrives from the correct ingress port as defined by the communication schedule for the virtual link of the message. Valid messages are put into the corresponding *virtual-link buffer*, which provides buffer space for exactly one message. In case this buffer is full and another message arrives with the same virtual-link identifier, the newer message replaces the old one.

The *TT scheduling layer* is responsible for relaying the periodic messages from the *virtual-link buffer* to the queue for periodic messages at the correct egress port according to the communication schedule. The communication schedule also determines the point in time when the periodic message is relayed, thereby ensuring the deterministic communication behavior.

For each egress port, the bridge has five egress queues with decreasing priorities: one queue for periodic messages, two queues for sporadic messages (each one for a different priority class) and two queues for aperiodic messages (also for two different priority classes).

The *egress port layer* forwards the messages from the egress queues to the MAC layer according to the priority. The highest priority is assigned to periodic messages, whereas aperiodic messages have the lowest priority.

Also, the shuffling and timely block mechanisms are realized in this layer. The timely block mechanism disables the sending of other Ethernet messages in the bridge during a guarding window prior to the transmission of a periodic message. For the shuffling mechanism, no guarding window is needed. In the worst-case, the bridge delays a periodic message for the duration of an Ethernet message of maximum size (i.e., 123 μs in case of 100 Mbps).

The virtual-link buffer and the static communication schedule ensure fault isolation for failures of nodes. For example, a babbling idiot failure of a node involving the transmission of untimely messages cannot affect the timing of messages from other nodes. In addition, masquerading failures are detected by the *bridge classification layer*. A masquerading failure occurs if an erroneous node assumes the identity of another node.

Processing of Sporadic Messages

Sporadic messages are encapsulated by a specific bandwidth allocation. The message flow of sporadic messages is associated with two main parameters for each virtual link: the Bandwidth Allocation Gap (BAG) and the jitter. The BAG value defines the minimum time between two Ethernet messages that are transmitted on the same virtual link. Jitter may be introduced by multiplexing all virtual links into shared egress queues.

In addition to these parameters, the bridge needs for every virtual link information about the ingress and egress ports, as well as the required queue size.

When a sporadic message arrives at the bridge, the message is checked in the filtering unit of the *bridge classification layer*. The size of the message must be below the maximum message size and the ingress port must comply with the configuration parameters of the virtual link. Valid messages are enqueued into the corresponding *virtual-link queue*.

The *RC shaper layer* realizes the traffic policing for the sporadic message by implementing an algorithm known as token bucket [ECR05]. This layer checks the time interval between consecutive messages on the same virtual link and moves sporadic messages from the *virtual-link queue* to one of the sporadic egress queues. We distinguish between two priority classes of sporadic messages with two corresponding sporadic egress queues. The *egress port layer* is responsible for forwarding the messages from the egress queue to the MAC layer.

Processing of Aperiodic Messages

The Spanning Tree Protocol (STP) as defined by IEEE 802.1D is used to establish a loop-free topology for communication of aperiodic messages [IEE04]. The supported periodic messages include Bridge Protocol Data Units (BPDU) and best-effort data messages.

BPDU messages are exchanged between switches to determine the network topology, e.g., after a topology change has been observed.

The switch needs two queues (called *aperiodic queues*) for these two aperiodic message types. The *bridge classification layer* puts incoming aperiodic messages into the BPDU queue or the aperiodic data queue.

The *Aperiodic configuration layer* consists of two processes: a process that handles aperiodic data messages and another one handling BPDU messages according to STP. Each process relays aperiodic messages from an *aperiodic queue* to one of the aperiodic egress queues. TTEthernet supports two classes of best-effort messages with each priority class having one aperiodic egress queue. The *egress port layer* forwards the messages from the egress queue to the MAC layer as explained previously for sporadic messages.

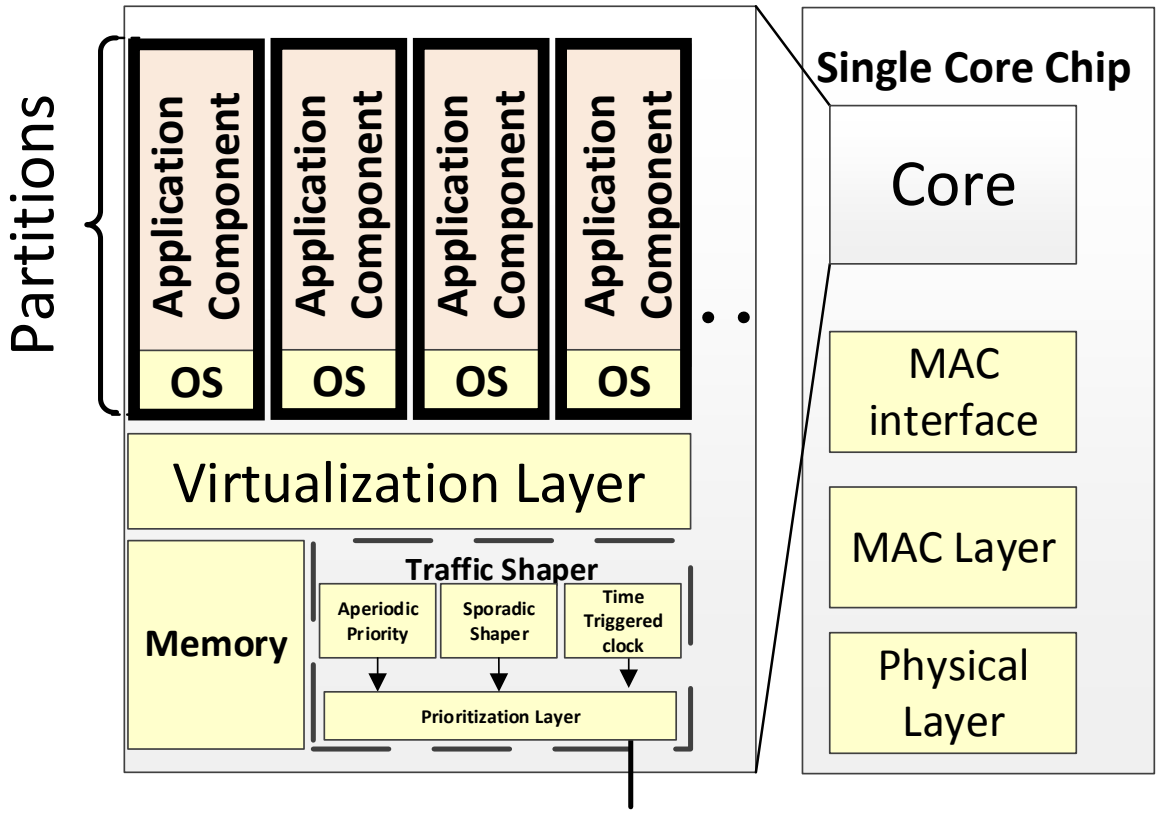


Figure 4.8 System Model of the Single Core Node

4.2.3 Node

In the proposed system, we distinguish between two types of nodes: multi-core nodes and single core nodes. The multi-core node is detailed explained in section 4.2.1. The system model of a single core node is shown in figure 4.8. To provide the temporal and spatial partitioning for such a system, the core architecture is built on top of the MAC layer. The MAC interface layer is responsible for providing services for the outgoing messages from the core and the incoming messages from the MAC layer. The messages that come from the MAC layer, are directly sent to the corresponding application.

Chapter 5

Redundancy for Mixed-Criticality Networks with Multiple Ring Topologies

The main requirement in the mixed-criticality systems is to achieve the high reliable system that can prevent a fault in a non safety-critical application from affecting safety-critical applications. This chapter introduces the redundancy and fault-tolerance solution for the off-chip networks based on TTEthernet as introduced in the system model in Chapter 4. We establish a balanced trade-off between cost and fault-tolerance for application subsystems of different criticality on shared networks. Safety-critical nodes are connected to two switches using redundant links, whereas non safety-critical nodes can use a single link to one switch.

The architecture supports periodic messages with low latency and jitter, sporadic messages with bounded latency and aperiodic communication. The proposed switches hide the duplication/deduplication of periodic and sporadic messages as well as the differences in the latencies incurred via different redundant paths. For periodic messages, a deterministic communication behavior with predefined latencies is ensured that does not change upon the failure of one of the redundant paths.

5.1 Mixed-Criticality Architecture based on a Ring Topology

Figure 5.1 illustrates the proposed architecture, which consists of four main parts: non-redundant nodes, redundant nodes with double channels, switches and Ethernet channels. Nodes are connected to switches in a star topology, while the topology for the interconnection of switches are rings or multiple rings. Figure 5.1 shows an example of a multi-ring topology,

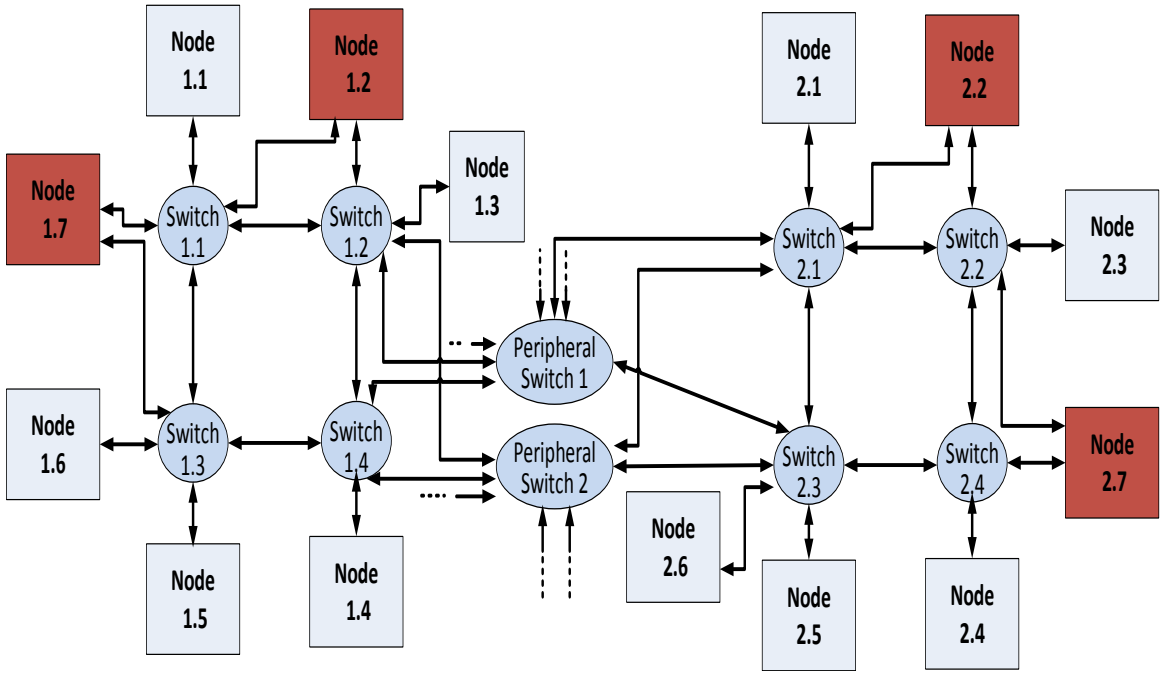


Figure 5.1 System Model of Mixed-Criticality System with Multiple Ring Topologies

where multiple rings are connected through peripheral switches. The peripheral switches connect to each ring using two Ethernet links to different switches.

Two categories of subsystems can be distinguished according to their criticality: safety-critical and non safety-critical nodes. Mixed-criticality for the nodes is supported based on the traffic type (i.e., periodic, sporadic, aperiodic).

Safety-critical nodes use double channels with periodic and sporadic traffic. The messages from safety-critical nodes are redundantly transferred on both channels, each of which is connected to a different switch. The aperiodic traffic can use both channels in order to increase the bandwidth.

A non safety-critical node is connected to only one switch using a non redundant channel. Nevertheless, periodic and sporadic traffic can be replicated at the first switch in order to improve the reliability of the communication from non safety-critical nodes.

Moreover, the redundancy management hides the duplication of messages on the network as well as the differences with respect to latencies of the redundant paths. The redundancy management is located in the switches and the safety-critical nodes. The hiding of latency differences and redundancy occurs as following:

- **Periodic traffic:** Time-triggered communication is done according to a static communication schedule that is defined off-line. Due to the schedule, it is known in advance when each redundant message will arrive. The redundancy management exploits the

time-triggered schedule to identify corresponding redundant messages that arrive from different paths. The time-triggered schedule includes the point in time for deciding on the sending of one message out of the redundant periodic messages.

- **Sporadic traffic:** The redundancy management hides redundancy based on sequence numbers and a "First Valid Wins" policy [Com05]. All sporadic messages include a one byte sequence number field, which is handled separately for each virtual link. A virtual link is a unidirectional connection from one node to one or more destination nodes.

5.1.1 Conceptual Model of Extended Switch

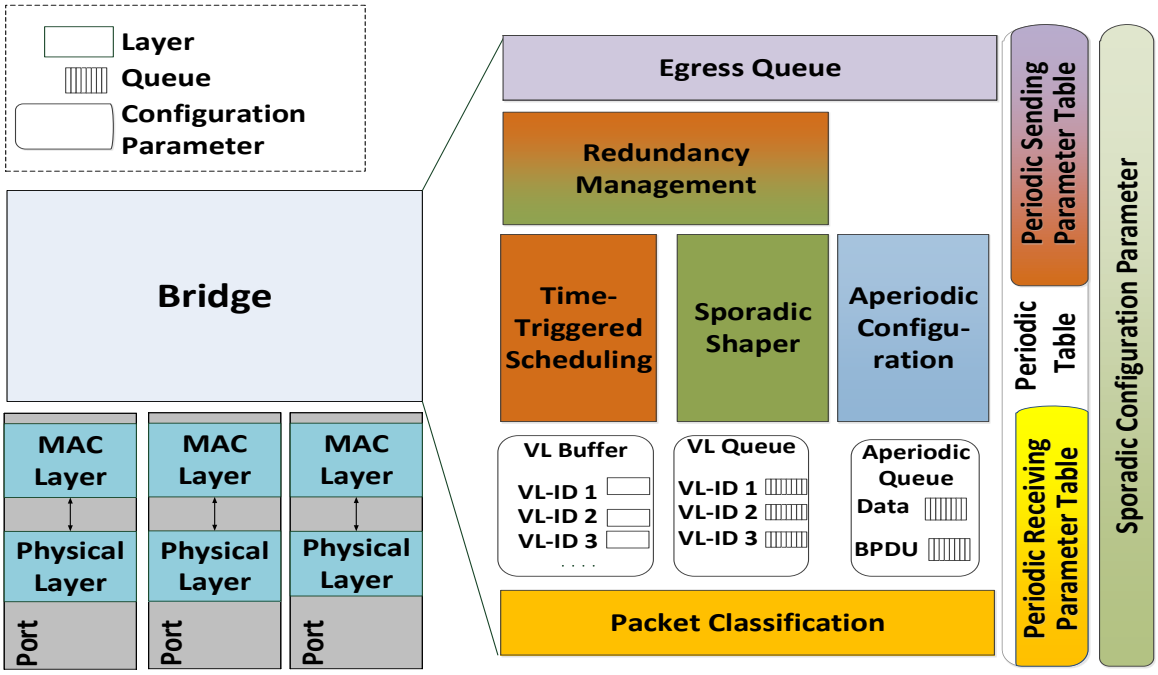


Figure 5.2 Extended Switch with Redundancy Management

The model of a switch with redundancy management for multi-ring topologies is illustrated in figure 5.2. The switch provides multiple *ports* and a *bridge*. Based on a priori knowledge on the permitted timing of messages, the switch supports error containment by ensuring that untimely message will not propagate between networks.

The time-triggered communication is based on a predefined schedule where there are two groups of parameters for each periodic message: a *periodic receiving parameter table* and a *periodic sending parameter table* providing the message period and phase with respect to a global time base (see figure 5.3).

The rate-constrained communication is based on configuration parameters that define a BAG and jitter for each virtual link. The BAG denotes the minimum time interval between two consecutive messages that are transmitted on the same virtual link. The jitter is the maximum timing variability that can be introduced by multiplexing the virtual links into shared egress queues. A message that arrives within the jitter is considered as timely, otherwise a new BAG interval is started. The structure of the configuration parameters is shown in figure 5.4.

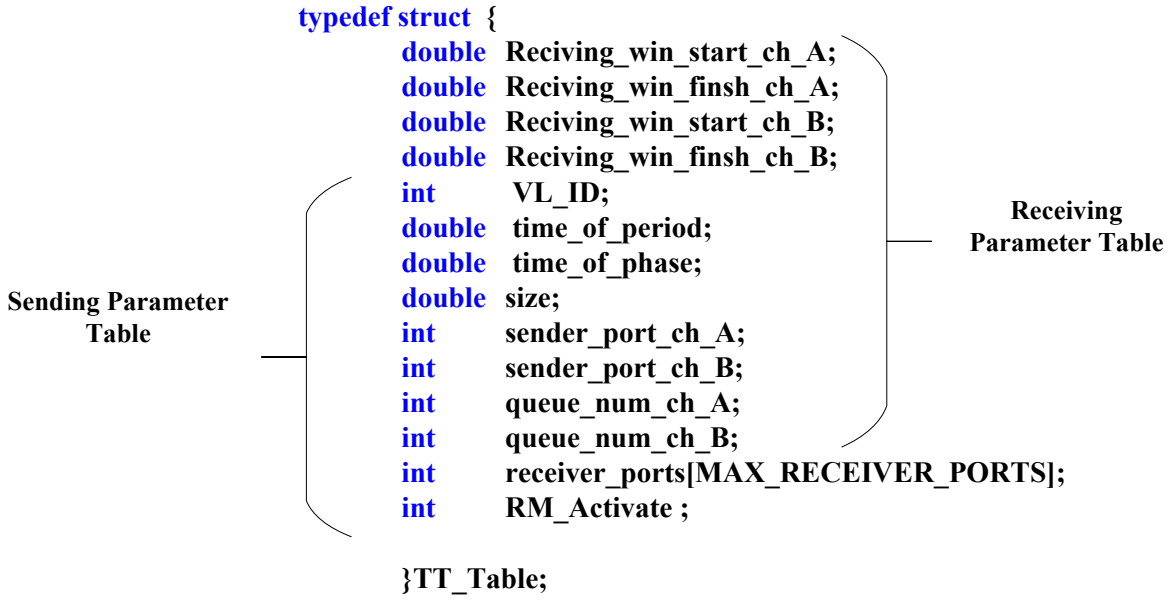


Figure 5.3 Time-Triggered Schedule

The switch architecture (cf. section 4.2.2.1) is extended by adding functionality in the existing layers and by adding a *redundancy management layer* to address error containment and redundancy management. The *redundancy management layer* hides the path and latency of the redundant messages.

5.1.2 Error Detection & Containment in the Switch

The proposed architecture provides several error detection and containment mechanisms, which are an essential part of error mitigation [Han06]. The switch performs error containment to avoid error propagation by the flow of erroneous messages.

The *MAC layer* detects errors such as noise spikes from power surges, crosstalk from wires being too close together and echoes due to faulty connections. Furthermore, the MAC layer uses Cyclic Redundancy Checks (CRCs) to detect transmission errors.

```
typedef struct {

                    double          BAG;
                    double          max_jitter; % jitter value
                    double          max_size;   % Maximum message size
                    int             sender_port_ch_A;
                    int             sender_port_ch_B;
                    int             receiver_ports[MAX_RECEIVER_PORTS];
                    int             RM_Activate ;
 % The active  can be:
•  No REDUDANCY: the switch sends the Packet according to the receiver
   ports.
•  FUSION: the switch applies the Redundancy Management before it sends
   the packet to the destination node.

                    int             priority;
                    int             queue_num_ch_A;
                    int             queue_num_ch_B;
                    int             VL-ID;
                    double          last_frame_time;
                    double          next_frame_time;
                    double          last_frame_time_B;
                    double          next_frame_time_B;
                    int             SN; % sequence number
                    Boolean         jitter;
} RC_msg;
```

Figure 5.4 Rate-Constrained Configuration Parameters

The *bridge classification layer* provides error containment based on the different traffic types as described in the following.

- **Periodic traffic:**

The *bridge classification layer* uses the *periodic receiving parameter table* to check the integrity and validity of the periodic messages. This includes the verification of the message size, checking whether messages arrive from the correct ingress port and whether they arrive within the specified receiving windows of the virtual link. Using theses predefined parameters, the switch protects receiving nodes and channels from several types of faulty message including untimely messages (e.g., babbling idiot failure), masquerading failures and syntactically invalid messages.

Periodic messages that arrive outside the receiving windows, message with a wrong sender port and messages with a wrong size are considered faulty.

Moreover, the *bridge classification layer* isolates messages from nodes of different criticality according to the VLID. The bridge classification layer assigns to each VLID a dedicated *virtual-link buffer*, which provides buffer space for exactly one message.

- **Sporadic traffic:**

When a sporadic message arrives at the *bridge classification layer*, the size of the message is compared against the maximum permitted message size. In addition, the ingress port

must comply with the *configuration parameters* of the VLID. The switch protects receiving nodes and channels from faulty messages such as a violation of the BAG. Valid messages are queued into the corresponding *virtual-link queue* where there is one such queue for each virtual link.

- **Aperiodic traffic:**

There is no guarantee and error containment for best-effort messages. However, any faulty message of the best-effort traffic type will not affect any other traffic types. This is due to the fact that the best-effort traffic has the lowest priority and the bridge classification layer handles best-effort messages using different queues, thereby isolating them from other traffic types.

The bridge classification layer interfaces the following three layers to handle the different traffic types:

1. Time-triggered scheduling layer

The *time-triggered scheduling layer* uses the information of each VLID that is listed in the *periodic sending parameter table* to pull the messages from the *virtual-link buffer* according to communication schedule and then forwards them to the correct egress ports. The communication schedule also ensures the deterministic communication behavior for the periodic messages. Furthermore, the *time-triggered scheduling layer* ensures the error containment for the failures of nodes. For example, babbling idiot failures of a node, which involve the transmission of untimely messages, cannot affect the timing of messages from other nodes.

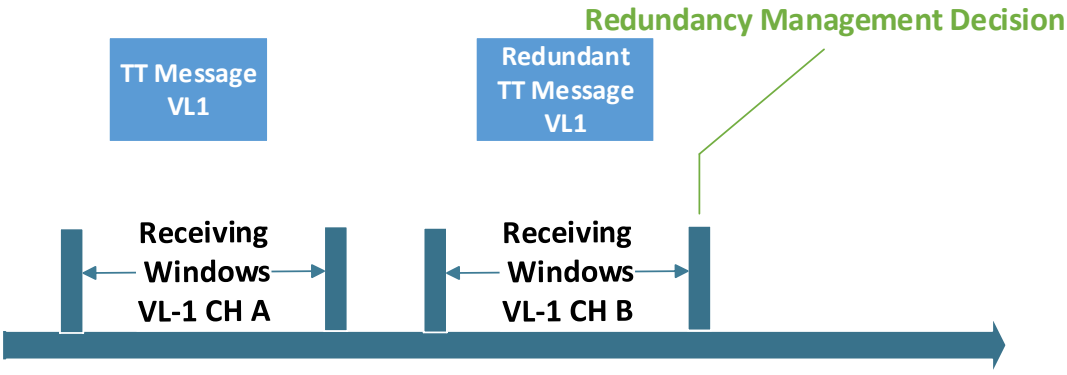


Figure 5.5 Time Line of a Redundant Periodic Message

2. Sporadic shaper layer

The *sporadic shaper layer* is extended by functionalities to protect receiving nodes and channels from faulty messages such as the babbling idiot failure. In case the rate-constrained message arrives with a shorter interarrival time than the specified BAG, the message is dropped. On the other hand, a correct message could be dropped in case the previous message is delayed.

3. Aperiodic configuration layer

The same Aperiodic configuration layer in Section 4.2.2.1 has been deployed in the extended architecture. Section 4.2.2.1 has a full description of this layer.

5.1.3 Mechanisms of Switch for Supporting Redundancy based on Multi-Ring Topologies

The *redundancy management layer* is responsible for eliminating the redundant copies of messages that arrive at the switch and to hide different paths of messages in the multi-ring topology for the critical traffic types (i.e., time-triggered and rate-constrained). Depending on the specification of the redundancy degree (cf. entry *RM_Activate* in *TT_Table* and *RC_Msg*) the redundancy management has different responsibilities.

Establishment of redundancy

The redundancy management layer creates copies of an incoming message belonging to the sporadic or periodic traffic and sends them using redundant paths of the ring topology. This mechanism is used on the first switch that meets the sporadic or periodic traffic.

Fusion of redundant messages

The fusion mechanism is deployed on the last switch of a message's path to a non safety-critical node. The fusion mechanism is also deployed in each safety-critical node to protect against a failure of one of the two attached switches.

In the fusion mechanism, the redundancy management handles the redundant messages differently according to the traffic type:

The redundancy management layer interfaces with the *time-triggered scheduling layer* to hide the redundant paths and to perform the deduplication of the periodic messages. As we mentioned in subsection 5.1.1, the *classification layer* assigns the redundant periodic message to separate *virtual-link buffers*. In order to hide the paths and different latencies of the different

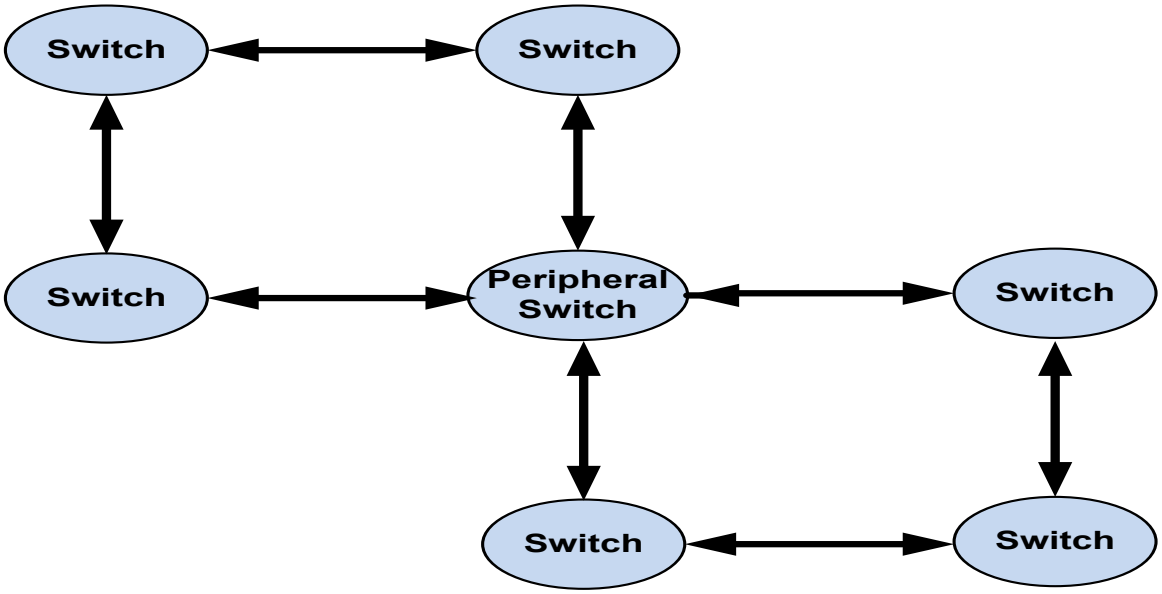


Figure 5.6 Peripheral Switch at the Interface between Rings

paths for the redundant periodic messages, the *redundancy management layer* requires a priori knowledge about the time-triggered schedule. This schedule includes information about the receiving time, the sending time and the corresponding buffer identification. The *redundancy management layer* checks the corresponding *virtual-link buffer* before the sending time and takes the decision to send one of the redundant periodic messages accordingly. The *redundancy management layer* not only hides the path of the redundant message, but also establishes deterministic arrival times of these messages. As shown in figure 5.5, the arrival times do not change in case of an omission failure affecting one of the redundant messages.

The *redundancy management layer* interfaces with the sporadic shaper layer to hide the redundancy of the sporadic paths and to perform the deduplication of sporadic messages. For each incoming sporadic message the *redundancy management layer* checks the sequence number and compares it with the sequence number that is listed in the configuration parameters. The "First Valid Wins" policy is used to take the decision on the forwarding of redundant messages. Upon the transmission of a message, the redundancy management layer updates the sequence number in the configuration parameters.

Fusion and deduplication in multi-ring topologies

In case of a multi-ring topology, the redundancy management layer performs both the fusion and the duplication at the interface between individual rings. The connection of multiple rings uses peripheral switches connected to different switches as shown in figure 5.1. Alternatively, a ring of ring topology can be used as shown in figure 5.6.

Non-redundant communication

In this mechanism, the redundancy management is not activated. Therefore, only the *time-triggered scheduling layer* and the *sporadic shaper layer* handle the periodic and sporadic messages.

5.1.4 Model of Extended Node

In order to support safety and to establish redundancy, we classify the nodes into two types: safety-critical nodes and non safety-critical nodes. The safety-critical nodes have the ability to communicate using two different channels while the non safety-critical nodes are connected to a single channel. Figure 5.7 shows the block diagram of a node. The safety-critical node is an extension of the node described in Section 4.2.3 with the redundancy management functionality on top of the MAC layers.

The redundancy management layer eliminates the redundant copies of the messages that already passed to the upper layers by deploying the fusion mechanism explained in Section 5.1.3. On the other hand, the non safety-critical node does not need any redundancy management since it enforces by the connected switch to forward only a single copy of each message as explained in Section 5.1.3. Section 4.2.3 has a full description of other node layers.

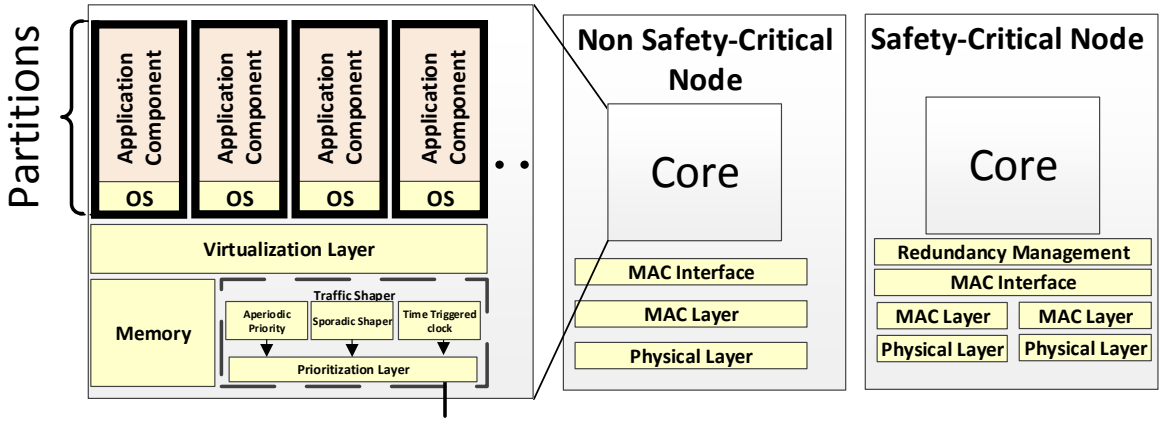


Figure 5.7 Block Diagram of (a) Non Safety-Critical Node and (b) Safety-Critical Node with Double Channels

Chapter 6

Off-chip/On-chip Gateways for Mixed-Criticality Systems

Multi-core processors introduce significant challenges for safety-critical systems and mixed-criticality systems. Shared resources of a multi-core processor such as caches, buses and inputs/outputs are a source of indeterminism in execution time analysis. As a consequence, the use of multi-core processors in safety-critical systems is a cause of concern to certification authorities. For example, avionic certification authorities point out that the features of multi-core processors could cause a loss of integrity, a loss of availability or non-deterministic behavior [Cer14].

In order to overcome these challenges and pave the way towards multi-core processors in safety-critical applications, deterministic multi-core platforms were introduced. Several multi-core architectures introduce message-based NoCs with TDMA, thereby avoiding the temporal unpredictability and the potential for fault propagation of architectures with shared memories and memory hierarchies. Examples for these architectures are the GENESYS MPSoC [SEH+12] and Æthereal [GH10].

However, a single multi-core chip is insufficient in many embedded applications. In many mixed-criticality systems, platforms encompassing networked multi-core chips are required. In addition to requirements exceeding the resources of a single chip, today's technology does not support the manufacturing of electronic devices with failure rates low enough to meet the reliability requirements of ultra-dependable systems. Since component failure rates per hour are usually in the order of 10^{-5} to 10^{-6}, ultra-dependable applications require the system as a whole to be more reliable than any one of its components. This can only be achieved by utilizing fault-tolerance strategies that enable the continued operation of the system in the presence of component failures.

Considering message-based NoCs and message-based off-chip networks, we obtain a hierarchical platform with on-chip and off-chip networks. Components on different multi-core chips need to interact through the communication over both on-chip and off-chip networks. This chapter introduces gateways between the chip-level and the cluster-level in order to establish this hierarchical platform. The gateways are specifically addressing mixed-criticality systems by combining heterogeneous timing models as well as temporal partitioning.

In what follows, the architecture of the proposed gateways including the services for selective redirection and fault isolation is presented in Section 6.1. The underlying algorithms are detailed in Section 6.2.

6.1 Architecture of Off-Chip/On-chip Gateway

An off-chip/on-chip gateway is responsible for the redirection of messages between the NoC and the off-chip communication network. Moreover, the off-chip/on-chip gateway supports different criticality levels. Three traffic classes are distinguished as the basis for different criticality levels as explained in Section 4.1. Examples of networks that support these three message types are TTEthernet [Com11] and FlexRay [Fle04] for the off-chip level. Æthereal NoC [GDR05] and the Time-Triggered NoC (TTNoC) [OESHK08] support these traffic types at the on-chip level.

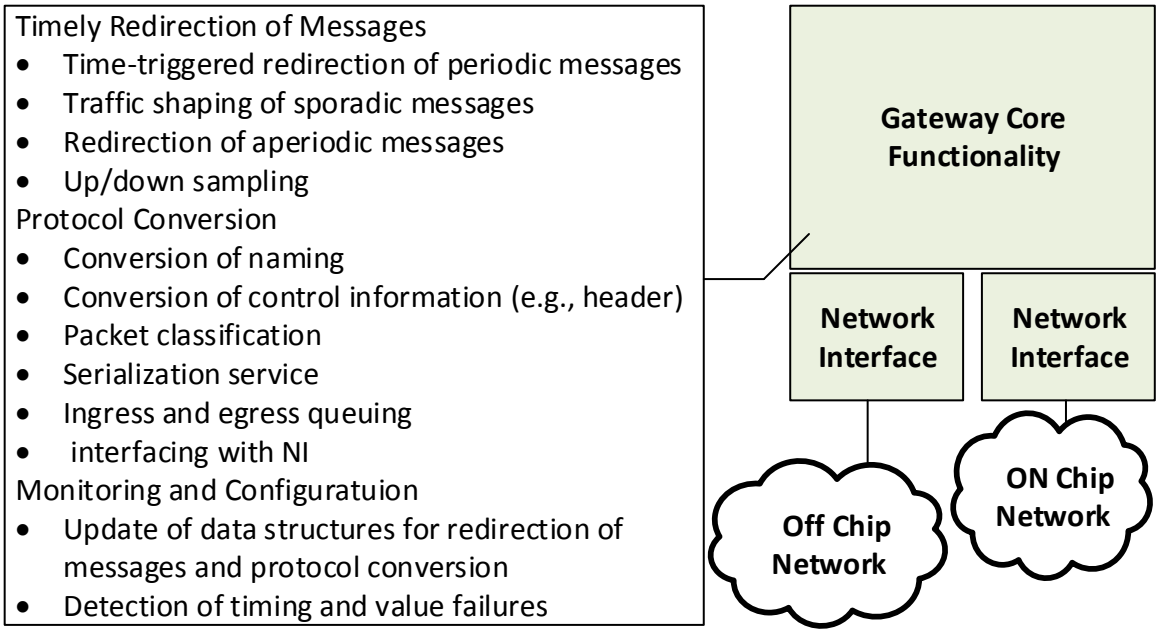


Figure 6.1 Off-Chip/On-chip Gateway

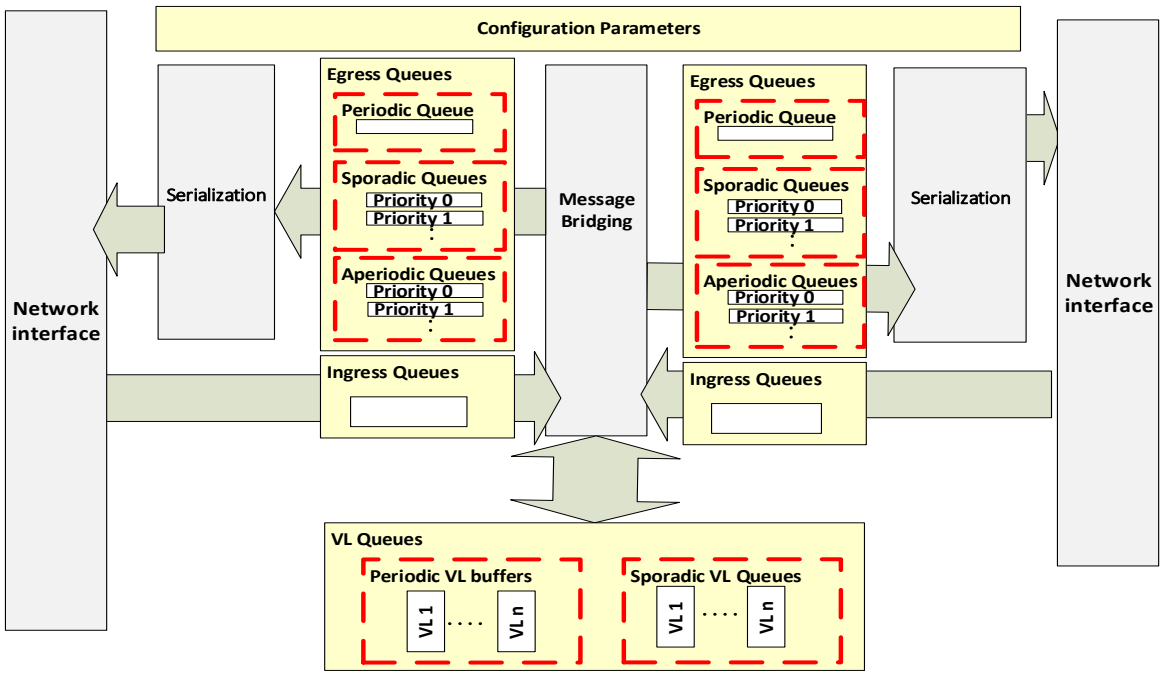


Figure 6.2 Architecture of the Off-Chip/On-chip Gateway

The off-chip/on-chip gateways are used in order to establish the end-to-end communication over heterogeneous and mixed-criticality networks. The connection between off-chip and on-chip networks is established through gateways as illustrated in Figure 6.1. The services of the gateway are explained in the following subsections.

6.1.1 Message-Classification Service

The message classification is based on the concept of VLs. The message-classification service is responsible for classifying the incoming messages from the network interface in order to decide on the corresponding buffer (i.e., VL queues and egress queues) according to the message type and the configuration parameters. Additionally, the message-classification service checks the message format and its control information, e.g. the Virtual Link IDentifier (VLID). In case the message has an invalid message format, it is discarded.

Moreover, the message-classification service uses the configuration parameters to check the integrity and validity of the periodic and sporadic messages. This includes the verification of the message size and checking whether messages arrive with correct VLID. In addition, the gateway checks whether the periodic messages arrive within the specified reception windows of the VL.

6.1.2 Message-Scheduling Service

This service guarantees the determinism of the redirection of periodic messages within the on-chip/off-chip gateway. Each periodic message has predefined timing parameters such as a period and a phase. According to the predefined configuration for the message scheduling, this service determines the points in time when the periodic messages are relayed.

6.1.3 Traffic-Shaping Service

This service is responsible for guaranteeing the minimum interarrival time between two consecutive sporadic messages on the respective VL. The minimum interarrival time is part of the configuration parameters for each VL.

6.1.4 Relaying of Aperiodic Messages

This service is responsible for relaying the aperiodic messages between ingress and egress queues based on the respective data direction and the destination addresses.

6.1.5 Down Sampling

This service provides the message exchange between networks with different periods of periodic messages or different rate-constraints of sporadic messages. Down sampling is also required to resolve the differences in the bandwidths of off-chip and on-chip networks. The gateway has to redirect a subset of the incoming messages to satisfy the timing requirements of the target network. In addition, the redirection needs to be synchronized between networks to ensure the forwarding of consistent data.

In the down sampling service, the gateway will send the most recent periodic message that arrived before the next transmission instant. In case of sporadic messages, the traffic shaper will drop all messages that arrive within the minimum interarrival time.

6.1.6 Protocol Conversion

The protocol conversion service is responsible for the encapsulation and decapsulation of incoming and outgoing messages. The gateway adapts the message format and the control information according to the used communication protocols (e.g., headers with addresses, flow-control information, CRC). In addition, the gateway needs to establish for each incoming messages a new address for the destination network. This computation is performed using

the address information of the incoming message and differs according to the traffic and network types.

In case of periodic and sporadic traffic, the new addressing information is either a VLID or a routing path towards the final destination or towards another gateway. The routing path is required for source-based routing, which is common in many NoCs. The VLID or the routing path can be acquired by a lookup of the incoming address information in the gateway's configuration.

In case of aperiodic traffic, the new addressing information is either a destination address or a dynamically computed routing path. The gateway can use the spanning tree protocol [stp04] to establish the destination address in a dynamic way.

6.1.7 Egress-Queuing Service

The egress queues consist of one periodic egress queue, multiple sporadic queues and one aperiodic egress queue. Each sporadic queue has its own priority level.

The deterministic behavior of the periodic messages is ensured by the message scheduling service (see Section 6.1.2) in combination with a higher priority than sporadic messages. The deterministic behavior guarantees that no conflict appears at the egress queue. Therefore one queue is sufficient, which needs to provide buffer capacity for a single periodic message of maximum size. To control the resolving of contention between the sporadic messages, we distinguish multiple queues according to their priorities. These queues are used to multiplex the frame flow that comes from the internal message queues. The queues provide guaranteed buffer capacities, which can also be realized by dynamic memory allocation. The guaranteed buffer capacities allow to prevent message loss due to the bounded accumulation of sporadic messages determined by the rate-constraints.

6.1.8 Ingress Queuing Service

The ingress queue consists of one FIFO queue for each network. The incoming messages from the network are queued into the ingress queue, then the ingress queuing service notifies the message classification service.

6.1.9 Virtual-Link Queuing Service

The VL queues belong to two groups: one for the periodic messages and the other one for the sporadic messages.

- **Periodic VL buffers**: Each periodic VL has one periodic VL buffer, which provides buffer space for exactly one message. In case this buffer is full and another message arrives with the same VLID, the newer message replaces the old one.

- **Sporadic VL queues**: Each sporadic VL has one queue. It is possible to store several messages of the respective VL in this queue.

6.1.10 Serialization Service

The serialization service forwards the messages from the egress queues to the network (off-chip or on-chip) according to the priority. The highest priority is assigned to periodic messages, whereas aperiodic messages have the lowest priority.

Also, the serialization service uses either shuffling or timely blocking to resolve contention between different traffic types. The timely block mechanism disables the sending of other messages in the egress queues during a guarding window prior to the transmission of a periodic message. For the shuffling mechanism, no guarding window is needed. In the worst-case, the gateway delays a periodic message for the duration of a sporadic or aperiodic message of maximum size.

6.1.11 Configuration Parameters

The configuration parameters of the gateway are as follows:

- **Guaranteed buffer capacity**: Each ingress queue, egress queue and VL queue is associated with a corresponding guaranteed minimum buffer capacity. The buffer capacity is determined by the maximum message size and the message timing. This buffer capacity can avoid message omissions of sporadic and periodic messages based on rate-constraints and message periods.

- **Address information of ports**: The VL associated with a port and the data direction (from the off-chip network or to the off-chip network) are defined.

- **Message type**: The message type is defined such as periodic, sporadic or aperiodic.

- **Timing parameters**: In case of periodic messages, the parameters include the period and phase. For sporadic messages, the interarrival time, the jitter and the priority are specified. In case of aperiodic messages, no timing parameters are required.

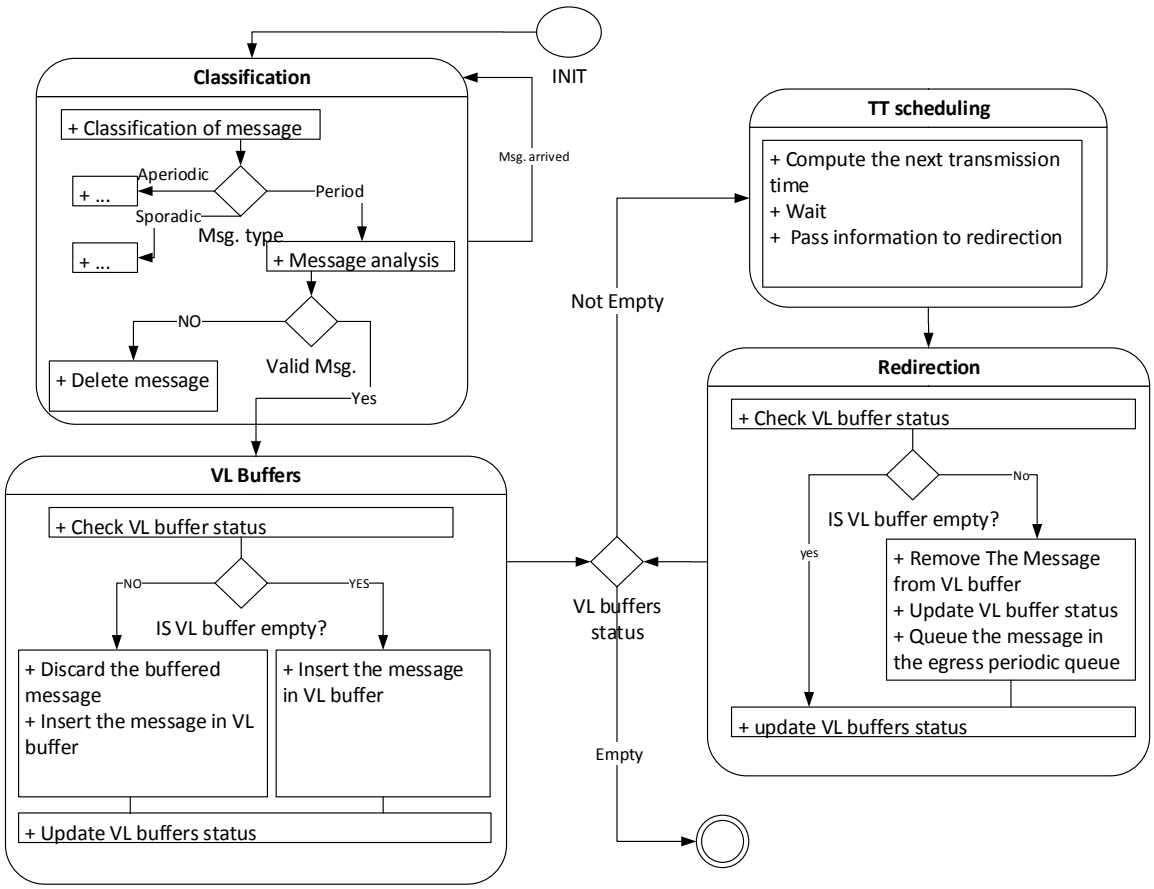


Figure 6.3 Flowchart for Periodic Messages

6.2 Processing of Different Traffic Types

In this section, we will describe the processing of the messages in the off-chip/on-chip gateway based on the introduced gateway services as described in Section 6.1.

Figure 6.2 illustrates the proposed off-chip/on-chip gateway architecture, which consists of the bridge, serialization, ingress, egress and VL-queue layers. The message bridging is responsible for handling incoming messages using timely redirection, protocol conversion, monitoring and configuration services. The network interface provides the interface between the network and the message bridging. Furthermore, the classification and serialization of the messages are performed. Each network interface connects the gateway to either an off-chip network (e.g., TTEthernet) or an on-chip network (e.g., STNoC).

The ingress layer is invoked by an incoming message from the on-chip network or the off-chip network. In the ingress layer, the incoming messages are relayed to the bridge layer.

The bridge layer classifies the incoming messages based on the message type (i.e., periodic, sporadic and aperiodic). We explain the processing for each message type in the following.

6.2.1 Processing of Periodic Messages

Figure 6.3 depicts the flowchart for periodic message transmissions. In the classification state, a *message analysis* function extracts from the periodic message the VLID. In case the incoming message does not have a defined VLID in the configuration parameters, the message is considered as invalid. Invalid messages are dropped in the classification state, while valid messages result in a transition to the VL buffer state.

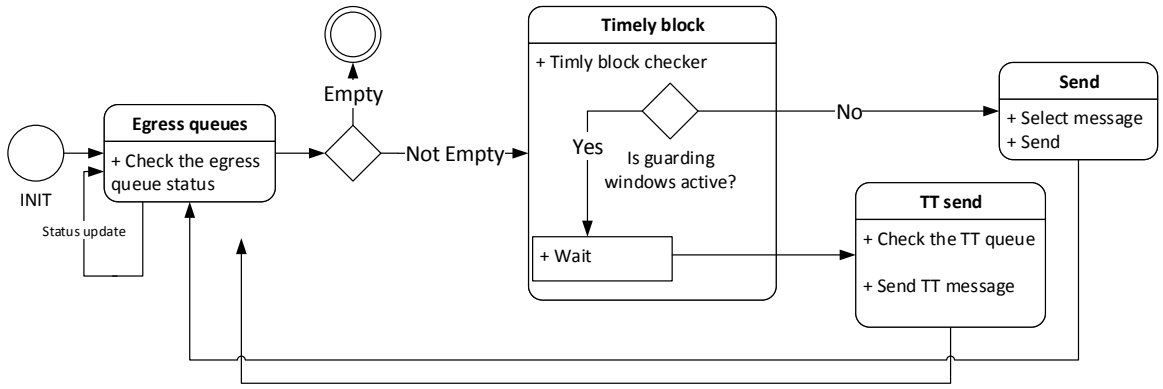


Figure 6.4 Flowchart for Timely Block Mechanism

Based on VLID, the *check VL buffer status* function retrieves the buffer identifier from the configuration parameters. Then, it puts the message into the VL buffer, which provides buffer space for exactly one message. In case this buffer is full and another message arrives with the same VLID, the newer message replaces the old one. When the message is buffered, the "VL buffer status" for the corresponding VLID is updated.

If the "VL buffer status" denotes that the buffer is not empty, the *next transmission time* function in the time-triggered scheduling state determines the point in time when the periodic message is relayed according to the communication schedule, thereby ensuring the deterministic communication behavior. At the next transmission time, the *pass information to redirection* function sends the information (i.e., VLID, buffer identifier and direction) to the redirection state. In the redirection state, the *check VL buffer status* function checks whether the corresponding VL buffer contains a message. This message is then sent to one of the egress objects according to the direction parameter, where the message is enqueued in a periodic egress queue. When the message is removed, the "VL buffer status" for the corresponding VLID is updated. These procedures are performed according to the communication schedule until the "VL buffer status" indicates that the buffer is empty.

The serialization is responsible for forwarding the message from the egress queues to the on-chip or off-chip network interface according to the priority. The highest priority is assigned to periodic messages, whereas aperiodic messages have the lowest priority. Using these priorities, the serialization supports two mechanisms to resolve collisions between the different types of messages:

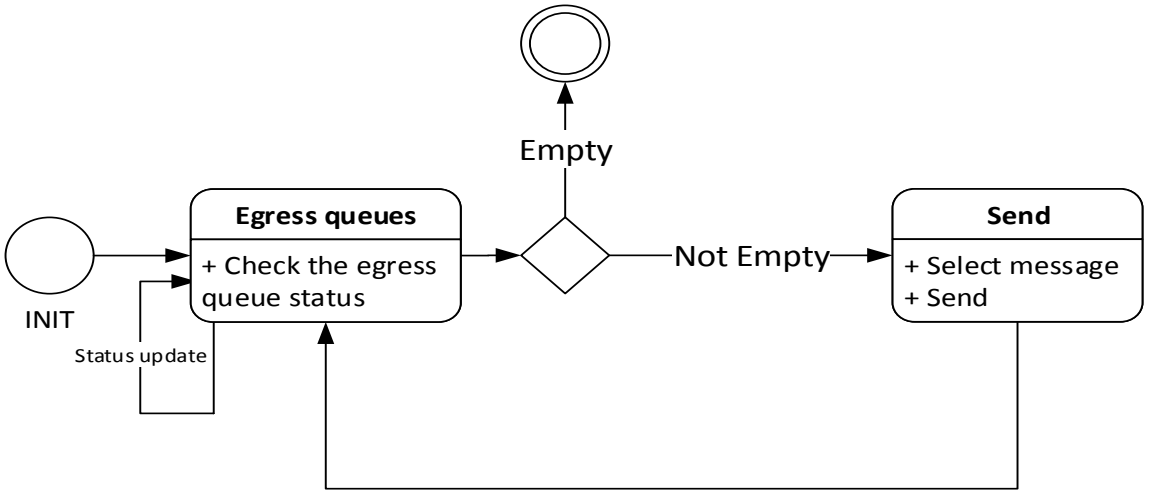


Figure 6.5 Flowchart for Shuffling Mechanism

- *Timely block:* According to the time-triggered schedule, the serialization knows in advance the transmission times of the periodic messages. Timely block means that the gateway reserves so-called guarding windows before every transmission time of a periodic message. The behavior of the timely block mechanism is illustrated in Figure 6.4. The egress queues have four egress queues with decreasing priorities: one queue for periodic messages, two queues for sporadic messages (each one for a different priority class) and one queue for aperiodic messages. The egress-queue status is updated when a message is enqueued in one of the egress queues. In case the status of the egress queue is “not empty”, the *timely block checker* function in the timely block state verifies that no guarding window is active.

  In case of guarding windows, the *wait* function imposes a delay until the next transmission time of the periodic message. If there is any periodic message, this message is sent. Otherwise, the process of the flowchart returns to the egress queue state.

  In case there are no guarding windows, the *select message* function in the send state selects one message out of the sporadic and aperiodic queues based on the priority and this message is sent.

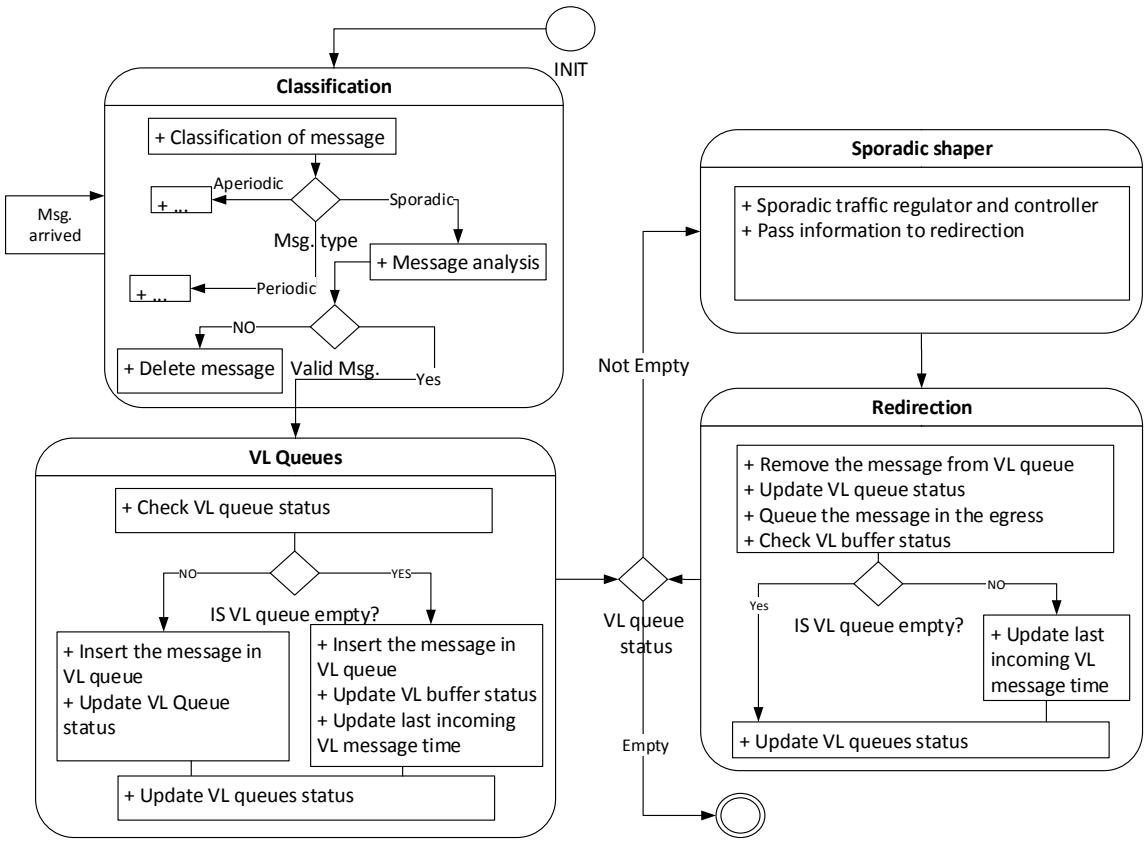


Figure 6.6 Flowchart for Sporadic Messages

If the status of the egress queues is still "not empty", the procedure is repeated until the egress queues are empty.

- *Shuffling:* If a low priority message is being transmitted while a high priority message arrives, the high priority message will wait until the low priority message is finished. Figure 6.5 shows the flowchart for the shuffling mechanism within the serialization object.

  The egress queue status is updated when a message is enqueued in one of the egress queues. In case the status of the egress queue is "not empty", the *select message* function removes one message from the egress queues based on the priority. The *send* function forwards the message to the network interface of the on-chip or off-chip network interface. If the status of the egress queue is still "not empty", the procedure is repeated until the egress queues are empty.

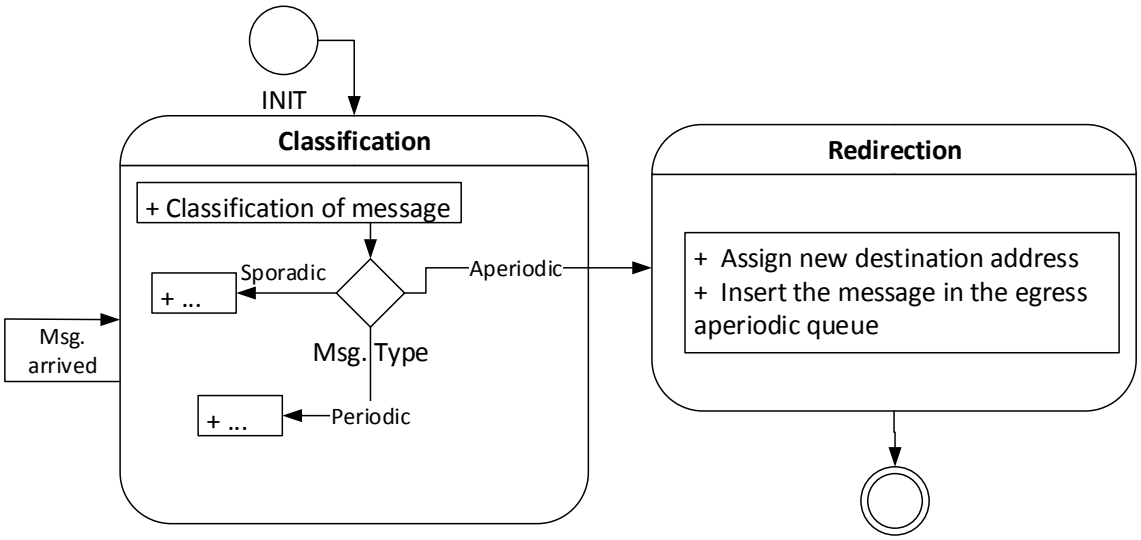


Figure 6.7 Flowchart for Aperiodic Messages

6.2.2 Processing of Sporadic Messages

Figure 6.6 depicts the flowchart for sporadic message transmissions. The *message analysis* function in the classification state checks incoming messages. The size of the message must be below the maximum message size according to the configuration parameters of the VL. A valid message is enqueued in the corresponding VL queue. When the message is enqueued, the "VL queue status" for the corresponding VLID is updated. In case the VL queue was empty, the *update time* function updates the reception time of the last incoming VL message. This timestamp is essential for traffic shaping and temporal partitioning.

In the sporadic shaper, the *sporadic traffic regulator and controller* function guarantees the minimum interarrival time between two consecutive instances of a sporadic message on the respective VL. The *sporadic traffic regulator and controller* function compute the necessary waiting time for each message based on the time of the latest incoming VL message. When the waiting time has expired, the *redirection* function passes the information (i.e., VLID, buffer identifier and direction) to the redirection state. In the redirection state, the *remove message from VL queue* function forwards the message from the VL queue to one of the sporadic egress queues according to the direction and priority parameters. In case the VL queue has another message, the time of the last incoming VL message is updated. This step allows the *sporadic traffic regulator and controller* function to send the next message after the minimum interarrival time. This procedure is repeated until the "VL queue status" is "empty".

Thereafter, the serialization is responsible for forwarding the message to the network interface of the off-chip or on-chip network as explained in the previous subsection.

6.2.3 Processing of Aperiodic Messages

Aperiodic messages have no timing constraints on successive message instances and no real-time guarantees. Therefore, the incoming messages are inserted into the corresponding aperiodic egress queue (see Figure 6.7). When the message is enqueued, the “egress queue status” is updated. Thereafter, the serialization is responsible for forwarding the message to the network interface of the off-chip or on-chip network.

Chapter 7

Scheduling of Sporadic and Periodic Traffic in Multi-Cluster Systems

Applications with different levels of criticality interact by the exchange of messages in a mixed-criticality system. One of the challenges of such a system is to provide a scheduling policy that ensures the timing guarantees for each level of criticality.

In this chapter, scheduling policies and resource allocation mechanisms of sporadic and periodic communication in multi-cluster systems are described.

7.1 Scheduling and Allocation Algorithm

In this section, the scheduling and allocation of the application to the platform are described based on the logical and physical models from Section 4.1.1. A major technical challenge is the mapping of the application subsystems with their computational components and communication activities to a multi-cluster platform, while satisfying real-time and reliability constraints. In particular, end-to-end communication channels with guaranteed timing are required to support the interaction of computational components located in different clusters along multiple network segments and switches. In multi-domain and mixed-criticality systems, typically different timing models need to coexist such as periodic time-triggered activities, event-triggered sporadic activities and aperiodic activities.

In the following, the logical model and the physical model are formally described. Then, the scheduling algorithm is explained for mapping both models considering timeliness, performance, safety and availability.

7.1.1 Logical Scheduling Model

The logical scheduling model encompasses information about the applications and the dependencies between the applications. The dependencies between applications for each subsystem are captured using a Directed Acyclic Graph (DAG). Each application sends a message after completing its execution to the target applications.

A target application can be activated only after all its inputs have arrived. For example, in figure 7.1 application $AP6$ can be activated after it received the messages sent by $AP2, AP3$ and $AP4$.

As shown in figure 7.1, each application has the following parameters:

- Service id
- Worst-Case Execution Time (WCET)
- Size of produced messages

In addition to these parameters, each subsystem has a criticality level, deadline, traffic type and timing parameters (i.e., periodic or BAG). The timing of periodic messages is characterized by a period and phase with respect to a global time base. Sporadic messages have a minimum inter-arrival time.

7.1.2 Physical Model

The physical model includes information about the nodes, network channels and network switches. Nodes can host applications from the logical scheduling model. Nodes are connected to network switches in a star topology, while the topology for the interconnection of switches is a tree.

The structure of the platform is modeled as an undirected graph $\mathscr{G}_{PHY}(\mathscr{V}_{PHY}, \mathscr{E}_{PHY})$, where $\mathscr{V}_{PHY}$ represents the set of nodes, gateways and network switches. $\mathscr{E}_{PHY}$ represents the set of network channels, where each network channel $e_i = (v_m, v_k) \in \mathscr{E}$ is an undirected communication connection between v_m and v_k.

We denote the set of VLs with $\mathscr{V}L$. A VL $vl_i \in \mathscr{V}L$ is an end-to-end multicast channel, which can include several network channels between one sender and multiple receivers. A vl_i can be denoted as a sequence $[e_1, e_2, \ldots]$ of network channels.

Each node executes a set of applications, where each application communicates with others applications using a set of messages $\mathscr{M}$.

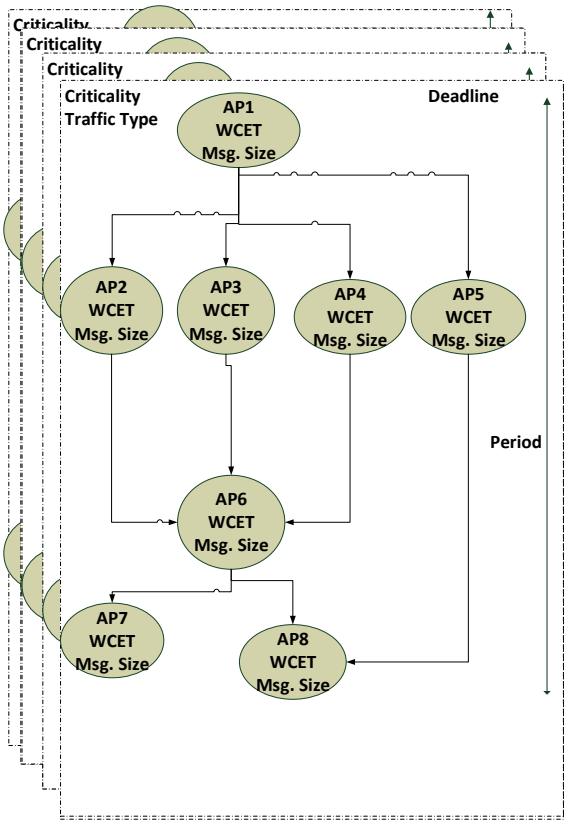


Figure 7.1 Directed Acyclic Graphs of Logical Model

7.1.3 Scheduling Model

The scheduling model serves for the optimization of the overall execution and communication time of the system. We minimize the maximum completion time of an application including the necessary communication and execution times of other required applications (i.e., critical path of the DAG).

Each application is executed in a node, it then produces a message for one or more target applications that can be located in other nodes. The schedule defines the timing parameters for the message transmissions by nodes and off-chip networks (e.g., network switches).

A short summary of the variables and constants of the algorithm formulation is given first. N_i represents the number of applications in the subsystem with id i. M is the number of available nodes and L is the number of available network switches.

$$\text{set of applications } \mathscr{AP}_i = \left\{AP_{i_0}, \ldots, AP_{i_{N-1}}\right\}, |\mathscr{AP}_i| = N_i$$
$$\text{set of nodes } \mathscr{NE} = \{NE_0, \ldots, NE_{M-1}\}, |\mathscr{NE}| = M$$
$$\text{set of network switches } \mathscr{SW} = \{sw_0, \ldots, sw_{L-1}\}, |\mathscr{SW}| = L$$

The logical model of a subsystem can be represented mathematically as follows:

$$\mathscr{G}_i = \begin{cases} period \in \mathbb{N} \\ deadline \in \mathbb{N} \\ criticality \in \text{CRITICALITIES (e.g., ASIL A to ASIL D according to ISO26262)} \\ traffic\_type \in \{Periodic, Sporadic\} \\ D = \begin{matrix} AP_0 & \dots & AP_{N-1} \\ [d_0 & \dots & d_{N-1}] \end{matrix}, \quad d_i \in \mathbb{N} \\ DSM = \begin{matrix} & AP_0 & & AP_{N-1} \\ AP_0 & & & \\ \vdots & & & \\ AP_{N-1} & & & \end{matrix} \begin{bmatrix} dsm_{00} & \dots & dsm_{0N-1} \\ \vdots & \ddots & \vdots \\ dsm_{N-10} & \dots & dsm_{N-1N-1} \end{bmatrix}, dsm_{ij} \in \{0,1\} \end{cases} \tag{7.1}$$

where D is a vector with the WCET of each application. DSM denotes the application dependencies where the element of the matrix dsm_{ij} equals 1 if s_j depends on s_i and otherwise it is equal to zero.

The physical model is depicted as a matrix in equation 7.2. The matrix element to_{ij} in equation 7.2 is equal to one if there is a connection between the two elements, otherwise it is zero. This connection is bidirectional, i.e., the matrix is symmetric.

$$TO = \begin{matrix} & NE_0 & \dots & NE_{M-1} & sw_0 & \dots & sw_{L-1} \\ NE_0 & & & & & & \\ \vdots & & & & & & \\ NE_{M-1} & & & & & & \\ sw_0 & & & & & & \\ \vdots & & & & & & \\ sw_{L-1} & & & & & & \end{matrix} \begin{bmatrix} to_{00} & \dots & to_{0\,M-1} & to_{0\,M} & \dots & to_{0\,M+L-2} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ to_{M+L-2\,0} & \dots & to_{M+L-2\,M-1} & to_{M+L-2\,M} & \dots & to_{M+L-2\,M+L-2} \end{bmatrix} \tag{7.2}$$

We use a set of tuples for each node and each network channel in order to express the resource allocations. Such a tuple is defined as follows:

$$< StartTime, EndTime, ID > \tag{7.3}$$

A node executes one application at a time which blocks the node for a certain interval (from *StartTime* to *EndTime*). Likewise, a message blocks the network channel for its transmission time.

The applications of a given graph are allocated to the nodes and represented in the following matrix:

$$A = \begin{array}{c} \\ NE_0 \\ \vdots \\ NE_{M-1} \end{array} \begin{array}{c} \begin{array}{ccc} AP_0 & \dots & AP_{N-1} \end{array} \\ \begin{bmatrix} A_{00} & \dots & A_{0N-1} \\ \vdots & \ddots & \vdots \\ A_{M-10} & \dots & A_{M-1N-1} \end{bmatrix} \end{array} ,A_{ij} \in \{0,1\} \tag{7.4}$$

Each application is allocated to a single node:

$$\sum_{i=0}^{M-1} A_{ij} = 1 \tag{7.5}$$

Based on A and TO the scheduling model defines the set of $\mathscr{V}L$ where each $\mathscr{V}L$ is a subgraph of the platform graph. A $\mathscr{V}\mathscr{L}$ for an application of AP_i where $(AP_i \in \{AP_{i_0}, AP_{i_1}, \dots\})$ comprises a set of hops from the sender to the receivers as follows:

$$\mathscr{V}\mathscr{L}_{AP_{i_n}} = \{h0, h1, \dots\} \qquad for \quad AP_{i_n} \tag{7.6}$$

Each hop is a network channel between nodes or switches.

7.2 Scheduling Algorithm

This section presents a scheduling algorithm based on neighbourhood search, which also serves as an example for the evaluation framework. The inputs for the scheduling algorithm are generated from a scenario definition and the scheduling results are an input for the simulation environment.

The scheduling algorithm is outlined in Figure 7.2. It supports the following scheduling constraints:

- *No collisions in execution:* The scheduling of applications on nodes must avoid collisions. Thus, the time intervals of applications scheduled on the same node must not overlap.

- *No collisions in communication:* The scheduling of message transmissions must avoid collisions. Thus, the time intervals for message transmissions must not overlap at each network channel.
- *Dependencies:* An application cannot start execution in a node before all required inputs have arrived.

```
1: Load 𝒱L & 𝒢_i & A
2: set the 𝒯_EPS for all root applications to zero
3: set the status of the root applications to active
4: for each active application do
5:     reserve earliest possible time in the allocated node
6:     if traffic type is Periodic then
7:         for each hop do
8:             reserve transmission time in network channel
9:         end for
10:    else
11:        for each hop do
12:            reserve bandwidth for transmission in network channel
13:            compute WCL
14:        end for
15:    end if
16:    if target application received all messages from relied upon applications then
17:        mark target application as active
18:        earliest possible start time of target application = max. reception time of incoming
   messages
19:    end if
20:    mark the application as finished
21: end for
22: calculate the graph cost
```

Figure 7.2 Scheduling Algorithm

In the first line of the scheduling algorithm, we load the inputs (i.e., constant $\mathscr{G}_i$) and we initialize the decision variables (i.e., $\mathscr{V}\mathscr{L}, A$). The status of each application is one of the following:

- *Inactive:* The application depends on input from others applications and one or more of these inputs are still unscheduled.
- *Active:* All required inputs of an application are already scheduled.
- *Finished:* The application was already scheduled.

Initially, all applications in $\mathscr{G}_i$ have the status inactive. We denote the earliest possible start time as $\mathscr{T}_{EPS}$. For the root applications in $\mathscr{G}_i$, $\mathscr{T}_{EPS}$ is set to zero and the application status is set to active. The root applications do not depend on any other applications.

The next step is to schedule the active applications. First, we reserve a possible time interval for the application execution in the node, where the application is located. The start of this interval must be greater than $\mathscr{T}_{EPS}$ of the application and the interval will end after the WCET of the application (line 5). In case a target application is located in another node, we reserve for each hop a message transmission time in case of periodic messages (line 7 to 9).

For sporadic messages we reserve the bandwidth based on the message size and the minimum interarrival time of the message. A Worst-Case Latency (WCL) (line 11 to 14) is computed based on the formula in the following subsection to determine the contribution to the critical path.

In case all inputs for the target application have been scheduled, the target application will change its status to active and set $\mathscr{T}_{EPS}$ equal to the message reception time. This process is repeated for all active applications.

To find near optimal solutions for the system schedule, a heuristic neighbourhood is used as outlined in figure 7.3. We change the location for one application, which is randomly chosen and reschedule the system again. By comparing the critical path latency between the new and the previous solution, we can select the minimum critical path latency and repeat these steps to optimize the solution.

```
1:  for j=0; j<iterations, j++ do
2:      change the location of one application
3:      compute the VL according to equation 7.6
4:      Update VL & A
5:      run scheduling algorithm
6:      if new cost < old cost then
7:          keep new location
8:      else
9:          backtrack to old location
10:     end if
11: end for
```

Figure 7.3 Heuristic Neighbourhood Algorithm

In case of periodic messages with different periods, we use the hyper-period of all periodic messages. To find the scheduling solution within the hyper-period, we use the algorithm as shown in figure 7.4.

1: **for** k=1; k<($\mathscr{G}_i$.period/hyperperiod) **do**
2: mark all applications as inactive
3: set earliest start time of root applications to $\mathscr{G}_i$.period $\cdot\, k$
4: run the scheduling algorithm from line 3
5: **end for**

Figure 7.4 Scheduling of Hyper-period

7.3 Worst-Case Latency

The communication bandwidth B_p is reserved for the periodic communication, while the bandwidth B_s is dedicated to the sporadic communication.

The access delay for the periodic messages using *timely block* depends on the precision Π of the global time base, which is typically in the microsecond range. In case of *shuffling*, the access delay is equal to the transmission delay of the largest message (denoted as $d_{\max}$).

The sporadic messages are described as a set M_{sp}:

$$M_{sp} = \{< id, mint, size, priority >\} \tag{7.7}$$

where *mint* is the minimum interarrival time. A given message with a priority p can be delayed by other messages with higher or equal priority:

$$M(p) = \{m \in M_{sp} | m.priority \geq p\} \tag{7.8}$$

The access delay of the sporadic messages depends on the competing messages that need to be sent before the message. We can establish the worst-case delay incurred by the sporadic messages with shuffling in $M(p)$ at the cluster level:

$$d^1_{sp,cl}(p) = d_{max} + \sum_{m \in M(p)} \frac{m.size}{B_s} \tag{7.9}$$

During the delay $d^1_{sp,cl}(p)$, additional instances of sporadic messages might request the transmission in case the minimum interarrival time is smaller than $d^1_{sp,cl}(p)$. The incoming sporadic messages within the access delay $d^1_{sp,cl}(p)$ that have higher priority will further increases the access delay. Thereby, we get a second-order delay $d^2_{sp,cl}(p)$.

By iteratively computing the increasing access delays due to the incorporation of additional instances of sporadic messages, we can derive the following access delay of order i for

a sporadic message at cluster level:

$$d^{i}_{sp,cl}(p) = d_{max} + \sum_{m \in M(p)} \frac{m.size}{B_s} \left(\left\lfloor \frac{d^{i-1}_{sp,cl}(p)}{m.mint} + 1 \right\rfloor \right) \tag{7.10}$$

If the following condition is satisfied, the increase of the access delays is bounded

$$\sum_{m \in M(p)} \frac{m.size}{m.mint} \leq B_s \rightarrow \exists j \in \mathbb{N}, \forall n > j: \\ \begin{aligned} d^{j}_{sp,cl}(p) &= d^{j+n}_{sp,cl}(p) \\ d_{sp,cl}(p) &= d^{j}_{sp,cl}(p) \end{aligned}$$

In order to compute the cluster delay d_{cluster}, we further need to consider the physical link delay which depends only on the size of the message and the link bandwidth B_l.

$$d_{link} = m.size / B_l$$

The cluster delay for the different traffic types is thus as follows:

$$d_{cluster}(m) = \begin{cases} d_{link} + \Pi & \text{timely block} \\ d_{link} + \max\limits_{m_{sp} \in M_{sp}} \frac{m_{sp}.size}{B_l} + \Pi & \text{periodic \& shuffling} \\ d_{link} + d_{sp,cl}(m.priority) + \Pi & \text{sporadic \& shuffling} \end{cases}$$

Chapter 8

Implementation and Evaluation

The purpose of this chapter is to demonstrate the feasibility of the proposed architecture using a simulation environment. The simulation environment executes mixed-criticality applications realized with either data-driven or time-driven models of computation at off-chip and on-chip level.

We evaluate empirically how the architectures achieve the dissertation objectives: (i) real-time capability (ii) determinism, (iii) fault isolation, and (iv) support for different timing models.

8.1 Implementation

The simulation environment is implemented using the GEM5 simulation tools [MSB+05] for on-chip communication and the OPNET simulation tools [OPN15] for the off-chip communication.

8.1.1 Off-chip Communication

The simulation environment for the off-chip communication implements the proposed architecture of the nodes (cf. Section 4.2.3) and switches (cf. Section 4.2.2.1) using the OPNET simulation tools. The simulation allows different connections of nodes using one or more switches in a given topology (e.g., star, ring). The main simulation building blocks at the cluster level are generic building blocks of the infrastructure elements of a system, which can be configured and extended to create an application-specific simulation model:

- *Generic model of a switch.* Switches are central building blocks of an off-chip communication system. We have developed a generic simulation model of a switch supporting

periodic, sporadic and aperiodic communication. In order to construct the overall simulation model, the user can perform multiple instantiations of the generic switch, establish connections to nodes and other switches, and assign to each switch instantiation a corresponding configuration. The switch configuration defines the message timing including a periodic communication plan.

- *Generic model of a node.* The user can perform instantiates of the generic node and connect each instantiation to switches. Nodes can be configured to produce messages according to application-specific parameters (e.g., interarrival time distributions of sporadic messages, periods of periodic messages). In addition, nodes can be extended with the application behavior (e.g., C++ application code).

The off-chip simulation environment realizes the proposed architecture models of the switch (cf. Section 4.2.2.1), node (cf. Section 4.2.3), extended switch with the redundancy management (cf. Section 5.1.1) and safety-critical node with double channels (cf. Section 5.1.4) for the ring topology. To evaluate the reliability and safety of the architecture, fault injection simulation building blocks are implemented, which inject faults according to the fault assumptions (cf. Section 4.1.3).

8.1.2 On-chip Communication

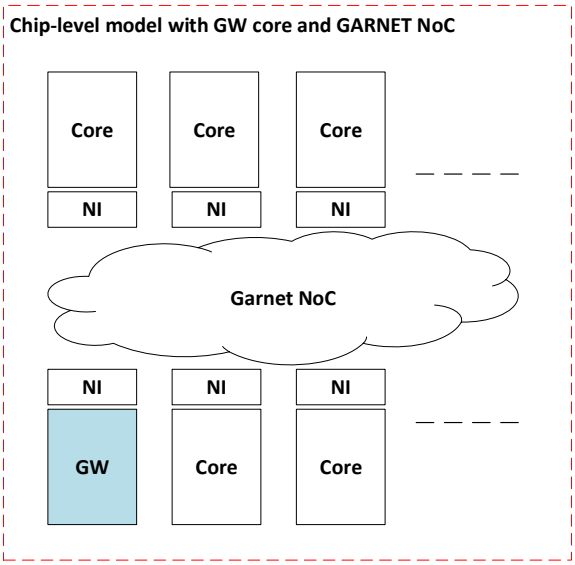


Figure 8.1 Overview of Simulation Building Blocks for On-chip System

The simulation model of the multi-core node with an off-chip/on-chip gateway was implemented using the GEM5 simulation tools [MSB+05]. The gateway was realized as a

core connected through the NoC to the others cores. The NoC simulation model uses the fixed pipeline GARNET interconnection network [AKPJ09].

Figure 8.1 shows the overall simulation building blocks. The gateway and the cores are connected through the GARNET NoC [AKPJ09] in a configurable network topology. GEM5 also allows configuring the cores (e.g., CPU, memory controller, L2 cache controller). The proposed gateway architecture from chapter 6 has been realized using GEM5 and integrated into the GARNET interconnection network. The class diagram for the gateway is illustrated in Figure 8.2. The constituting classes of the gateway are the bridge, serialization, on-chip network interface, off-chip network interface as well as ingress, egress and VL queues.

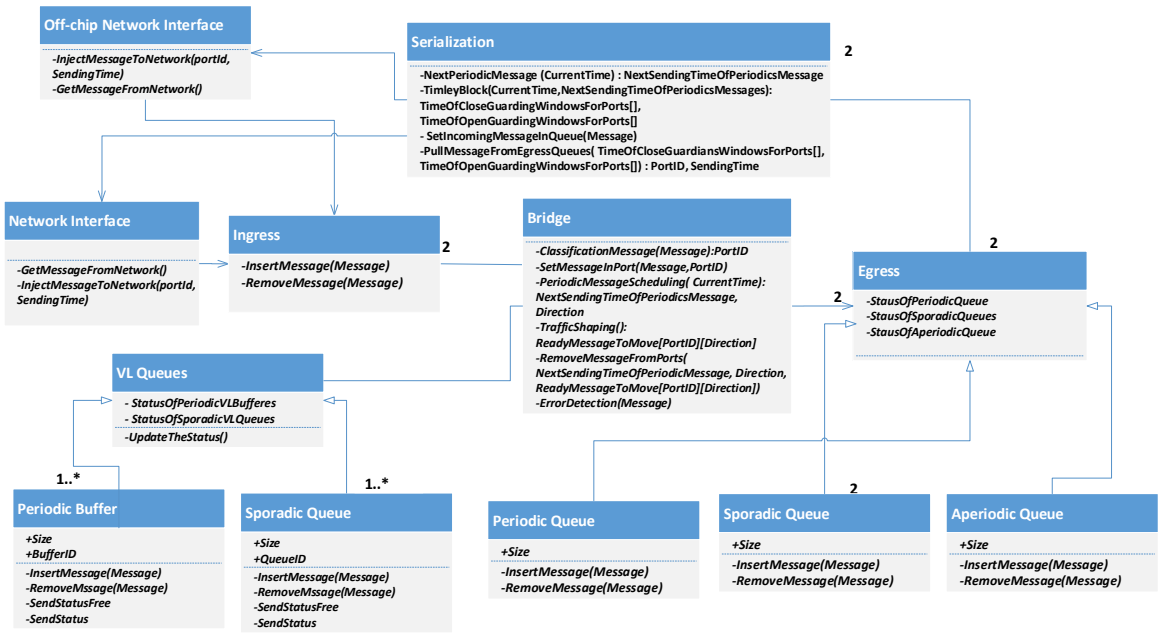


Figure 8.2 Gateway Class Diagram

In the GARNET NoC, the cache coherence protocols are used to establish the communication between the cores and the interconnecting networks. The cache coherence protocol "Network test" [1] is used to test a network configuration with a specific protocol and observe the statistics of the simulation with the messages sent from the core to the other cores. This protocol was extended to allow the gateway to interact with the GARNET interconnection network. Two virtual networks are used, namely one for the periodic messages and the other one for the sporadic and aperiodic messages. This division gives priority to the periodic messages within the interconnection network. In addition, the message generation in the cores was realized to produce the different traffic types.

[1] http://www.m5sim.org/Network_test

8.1.3 Framework for Evaluation of Scheduling Algorithms

A simulation framework has been established to evaluate and verify scheduling algorithms using automatically generated scenarios. Figure 8.3 depicts the validation framework, which consists of five building blocks: input parameters, a logical model of the application, a physical model of the platform, a scheduling model and a simulation environment. The input parameters include general information about the system (e.g., number of switches, number of applications). These parameters are used to generate the application model (cf. subsection 7.1.1) and the platform model (cf. subsection 7.1.2). The application and platform models are used as input for the schedule module, which computes the communication and execution schedule. The simulation environment evaluates the behavior under the computed schedule. The simulation framework provides feedback to the designer concerning the impact of the schedules on the timing and reliability of the system.

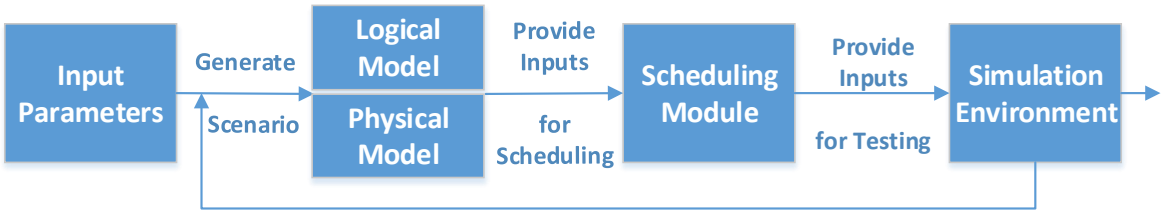


Figure 8.3 Validation Framework

The implementation of the simulation and scheduling framework uses the OPNET modeler in combination with a custom tool-chain as we explained above. The tool-chain consists of four building blocks: a random generator for the physical model of the platform, a random generator for the logical model of the application, a scheduling algorithm, and a simulation environment for off-chip communication [AO13]. The tool-chain uses input parameters for the generation of the simulation models as summarized in table 8.1.

The following subsections explain in detail the implementation of the building blocks.

8.1.3.1 Random Generator for Physical Model of Platform

The random generator for the platform is implemented using the OPNET modeler. Nodes are connected to switches in a star topology, while the topology for the interconnection of switches is trees organized in successive levels.

The generator starts by connecting the switches randomly in a tree topology. The switches are partitioned into a number of horizontal layers. The model generator picks one switch (called root switch) and sets the level number to zero. Then, the model generator picks a number of switches and connects them to the root switches. These switches are children

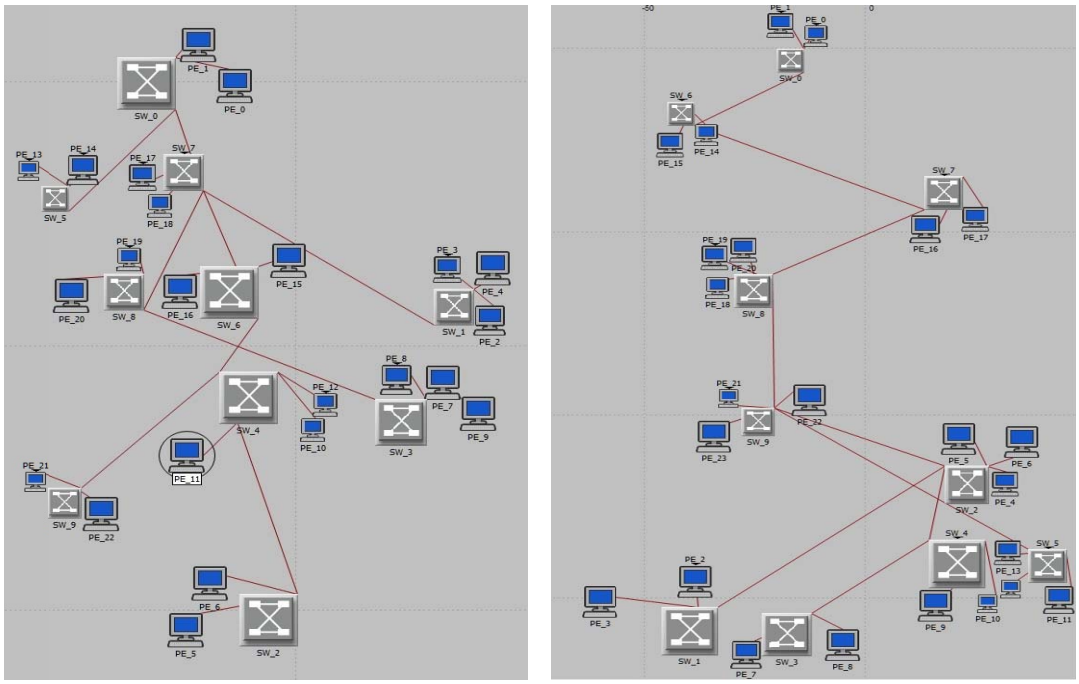


Figure 8.4 Example of Platform with 10 Switches, 23 nodes, 30% PCSW and 50% PCL

of the parent switch at the level of the *parent* + 1. The number of switches at each level is selected randomly from a minimum of one switch to a maximum number of children switches. This maximum is determined by the percentage of children switches out of the number of remaining switches that still do not have a parent switch. In addition, one or more of the children switches can skip a level and directly proceed to *"parent level"* + 2 based on the targeted percentage of children switches PCL (e.g., "SW_5" in Figure 8.4(b)). Finally, the nodes will be distributed randomly around the switches.

The outputs of the generator are CSV files which include the information of the distributed system in a matrix form (cf. matrix in equation 7.2) and a simulation project including the OPNET components of the distributed system and their interconnection. Figure 8.4 shows an example of two output networks for the same input parameters.

8.1.3.2 Random Generator for Logical Model of Application

The random generator for the application is implemented in C++ and generates random DAGs with applications, dependencies between applications, and further information as explained in equation 7.1. The input variables for the generator are the number of the graphs, the number of applications and the percentage of children applications in the next level (PCS).

Input Parameters	
NOTATION	**DEFINITION**
SW_num	number of network switches
NE_num	number of nodes
CHA_num	number of network channels
num_Periodic_graph	number of periodic logical graphs
num_Sporadic_graph	number of sporadic logical graphs
num_Periodic_applications	number of periodic applications
num_Sporadic_applications	number of sporadic applications
PCSW	percentage of children switches
PCL	percentage of children switches that exceed one level
link_bandwidth	used bandwidth of Ethernet links
MAX_SERVICE_EXEC_TIME	maximum execution time of applications
MIN_SERVICE_EXEC_TIME	minimum execution time of applications
PCS	percentage of children applications
P	period

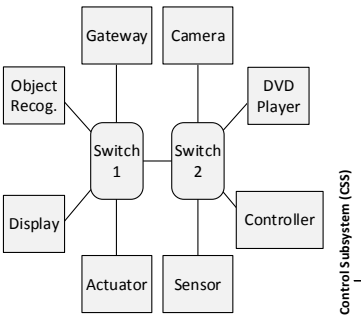
Table 8.1 Definition of Input Parameters

The application graph is structured as a DAG with a tree structure and multiple roots. The application graph is partitioned into a number of horizontal levels. The generator picks a random number of applications and assigns the level number as zero. An application at level zero is called root application. Each root application will be connected to a random number of children applications at the next level. Then each children application has a random number of children applications at the next *level+1*. Each level has its number of children applications based on a random selection of the product of the PCS with the number of applications that still do not have a parent application. Additionally, the model generator assigns a random value for the WCET of each application and also assigns a deadline to the entire graph.

The output of the generator is also a CSV file, which includes information about the logical system graphs. This information is explained in equation 7.1 in section 7.1.3.

8.1.3.3 Scheduling Algorithm

The scheduling algorithm is implemented in C++ to find a feasible solution. This algorithm uses CSV files from the random generators for the platform and application to initialize the



					Tim-Triggered CSS			Rate-Constrained CSS			Rate-Constrained CSS with Babl. Idiot		
Source	Destination	Bandwidth (Mbps)	Period (ms)	Playload size (Bytes)	Max (µs)	Min (µs)	ε (µs)	Max (µs)	Min (µs)	ε (µs)	Max (µs)	Min (µs)	ε (µs)
Camera	Object Rec.	[2.7, 10.8]	0.8	[300, 900]	2385	79	2306	2385	79	2306	2397	79	2318
DVD	Display	9.6	1	1200	367	295	72	364	295	69	Fault Injection		
FlexRay	DVD	2,048	1	256	130	113	16	114	68	45	114	68	45
FlexRay	Camera	2,048	1	256	130	113	16	114	68	45	114	68	45
CAN	DVD	0.256	1	32	136	120	16	137	75	62	137	75	62
CAN	Camera	0.256	1	32	136	120	16	137	75	62	137	75	62
FlexRay	Controller	2,048	1	256	130	113	16	114	68	45	114	68	45
FlexRay	Sensor	2,048	1	256	130	113	16	114	68	45	114	68	45
CAN	Controller	0.256	1	32	136	120	16	137	75	62	137	75	62
CAN	Sensor	0.256	1	32	136	120	16	137	75	62	137	75	62
Sensor	Controller	5.12	10	64	20	19	1	124	17	107	124	17	107
Controller	Actouter	5.12	10	64	43	42	1	143	26	116	233	26	207
Actouter	Sensor	5.12	10	64	75	74	1	243	26	217	243	26	217

Figure 8.5 Example Scenario for Automotive Use-case

input parameters. The algorithm as described in section 7.1.3 is used as an example in the validation framework.

The outputs of this scheduler are CSV files that include configuration parameters of the simulation building blocks of the platform.

8.1.3.4 Verification Using Simulation Environment for Off-chip Communication

The simulation environment for off-chip communication (cf. Section 8.1.1) is used to verify the scheduling algorithm. The output of the generator is a simulation project which is configured using the parameter files. We extended existing switches (cf. Section 8.1.1) for reading the configuration parameters from CSV file. Additionally, we implemented model parsers to read CSV file and to set the configuration parameters of node.

The output of the simulation is information about the timing of the applications (i.e., latency and jitter).

8.2 Evaluation

Several tests have been performed to evaluate the proposed mixed-criticality architecture with the extended nodes and switches. The transmission delay and jitter were observed for the safety-critical and the non safety-critical applications. Moreover, fault injection was used to test the fault-tolerance mechanisms and to evaluate the reliability of the system.

8.2.1 Automotive Evaluation Use-case

This section shows an automotive example scenario for the simulation environment. The main objective of this scenario is to study the application behavior under different traffic types (e.g., periodic and sporadic) and to evaluate the handling of faults.

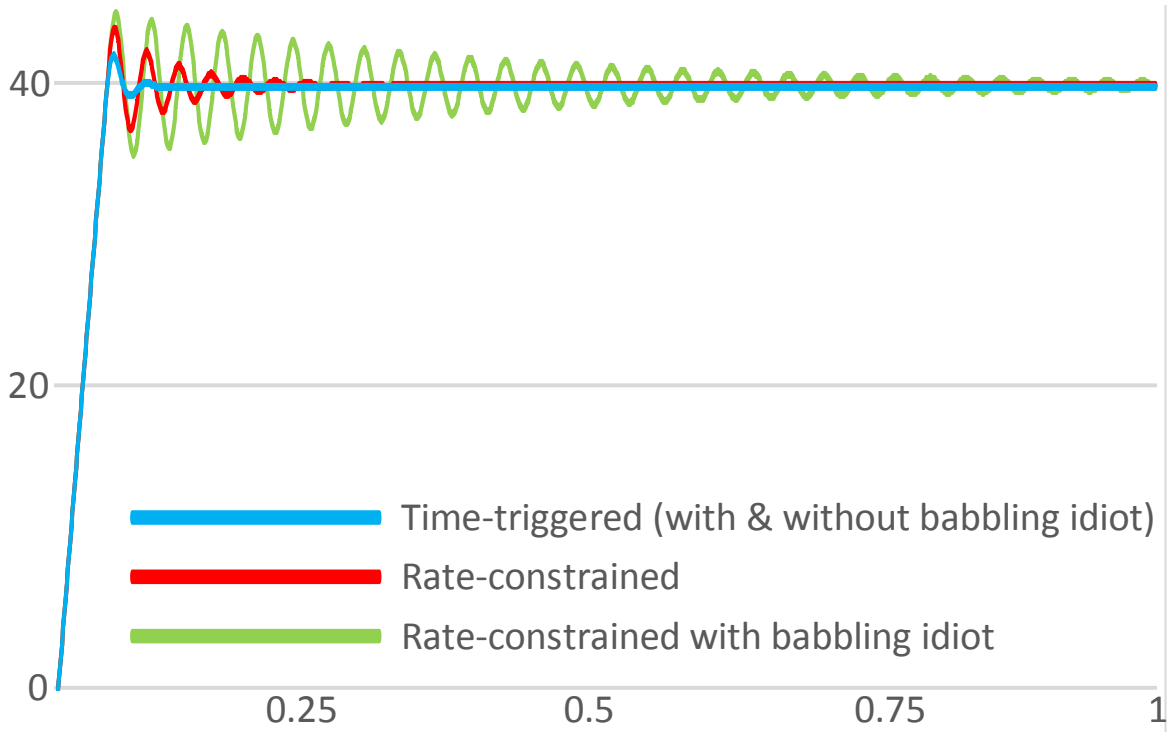


Figure 8.6 Simulation Results for Automotive Use-case

Figure 8.5 illustrates the network topology and the traffic generated by the nodes in this example. The topology consists of several nodes that communicate over an off-chip communication network with two switches. The sensor, the controller and the actuator nodes realize a cruise controller to maintain the speed of a car at a desired value.

This example scenario including the switches and nodes was realized using the presented simulation components. The cruise-control subsystem was implemented using an environmental simulation in the sensor node and a PID controller in the controller node. Sporadic communication was used between the camera, DVD player and the gateway. The control service was realized both with sporadic and periodic frames. In addition, faults were injected by simulating a babbling idiot failure of the DVD node.

Figure 8.6 shows the observed control behavior. The periodic messages result in the best controller behavior without any effect from the babbling idiot. The sporadic communication exhibits a transient oscillation for 0.3sec. The babbling idiot failure increases the latency and latency jitter, thereby increasing the duration of the transient oscillation to more than 1sec.

The table in Figure 8.5 contains an overview of the observed minimum and maximum latencies, as well as the latency jitter for the periodic (red rows) and sporadic (black rows) communication.

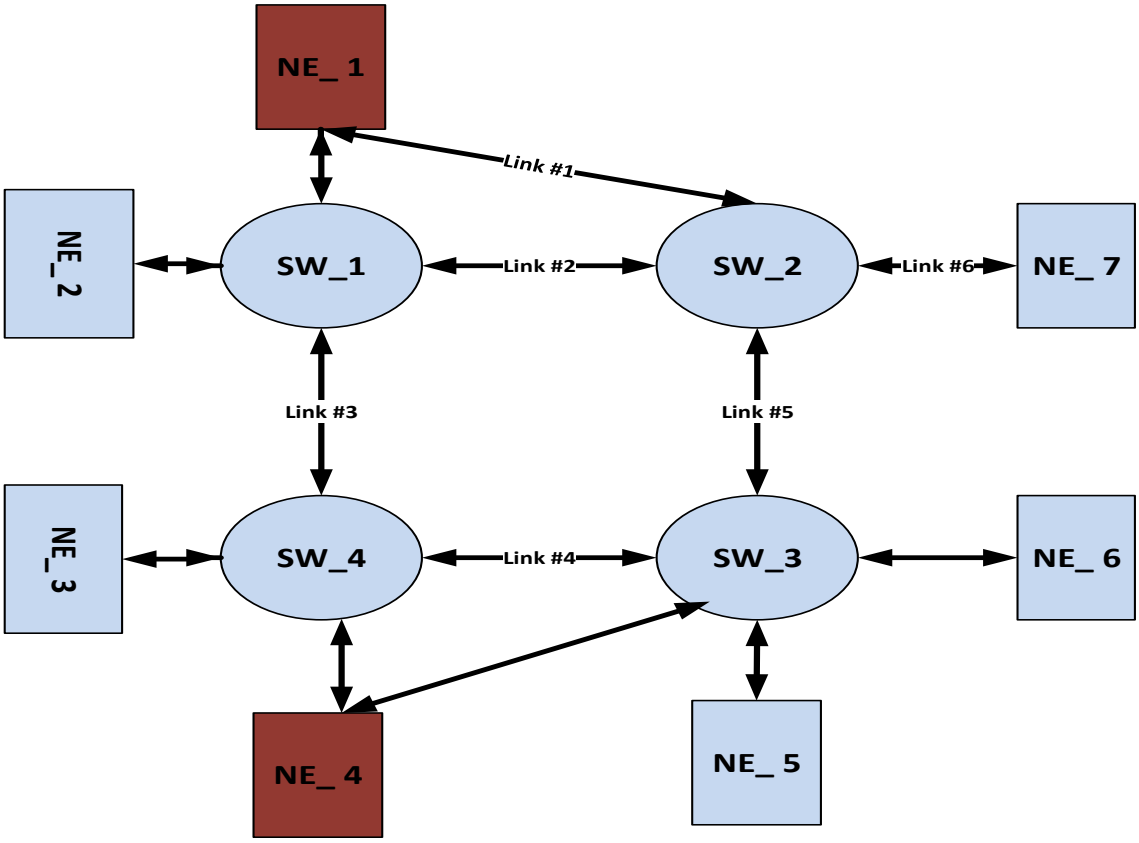


Figure 8.7 Evaluation Scenario Based on Ring Topology

8.2.2 Evaluation Use-case Based on Ring Topology

Figure 8.7 illustrates the ring topology scenario, which consists of five non safety-critical nodes, two safety-critical nodes and four switches that are connected through off-chip networks. Each node contains one or more of the applications that are listed in table 8.2. As shown in table 8.3 the data exchange of the applications is performed using periodic messages and sporadic communication.

For the evaluation of the proposed architecture, we compared the fault-free case with fault scenarios such as babbling idiot failures, delay failures, omission failures, link failures and crash failures. Table 8.3 illustrates the traffic generated by the nodes for these scenarios and the observed timing of the communication system.

Table 8.4 lists the simulation results for the fault-free case and the fault scenarios of the evaluation. We observed significant discrepancies of the end-to-end jitter, i.e., the difference between the maximum and minimum end-to-end latency between applications. The jitter of the periodic messages is *zero* (when abstracting from the limited precision of the global time base), while the jitter of the sporadic messages depends on the priority of the virtual link.

	App. 1	App. 2	App. 3	App. 4	App. 5	App. 6	App. 7
Service Rate (ms)	0.5	8	5	1	0.05-0.1	0.01	0.25-0.04
Size (bytes)	512-786	1450	200	1024	400	64	800-1200

Table 8.2 Application Timing Behavior of Evaluation Scenario Based on Ring Topology

ID	Application types	Sender	Traffic type	Receivers
1	App. 1	Node 1	Periodic	Node 5 Node 6
2	App. 4	Node 1	Periodic	Node2
3	App. 7	Node 2	sporadic	Node 2 Node 3 Node 4 Node 5 Node 6 Node 7
4	App. 2	Node 3	sporadic	Node 2 Node 5 Node 7
5	App. 7	Node 3	sporadic	Node 2 Node 3 Node 4 Node 5 Node 6 Node 7
6	App. 1	Node 4	Periodic	Node 5 Node 6
7	App. 2	Node 4	sporadic	Node 2 Node 5 Node 7
8	App. 4	Node 4	Periodic	Node 2
9	App. 5	Node 4	sporadic	Node 1 Node 3 Node 6 Node 7
10	App. 3	Node 5	Periodic	Node 1
11	App. 5	Node 5	sporadic	Node 1 Node 3 Node 6 Node 7
12	App. 2	Node 6	sporadic	Node 2 Node 5 Node 7
13	App. 6	Node 6	sporadic	Node 1 Node 4
14	App. 3	Node 7	Periodic	Node 3
15	App. 6	Node 7	sporadic	Node 1 Node 4

Table 8.3 Messages Exchange in the Evaluation Scenario Based on Ring Topology

In the babbling idiot failure scenario, node 3 floods the network with untimely aperiodic messages during a fault interval of 500 ms in the overall simulation time interval. The periodic messages are unaffected because of the timely block mechanism used at the egress ports as listed in Table 8.4. The latency and jitter of the sporadic messages increases (e.g., ID 3, ID 7), which results from the competing messages at the egress port where the shuffling mechanism is used for resolving the event-triggered contention.

In the link failure scenario, link 4 fails for 500 ms during the simulation time interval. Also in the omission failure scenario, all message transmissions via link 2 are dropped during the fault interval of 500 ms during the simulation time. The latency and jitter of the periodic messages in both scenarios are unchanged because the worst-case path (out of the redundant paths) is considered at the design phase. The jitter of the sporadic communication is effected

when a message takes a longer redundant path and the shortest path becomes unavailable during the fault interval.

In the omission failure scenario, switch 3 emulates the behavior of the transient fault. Switch 3 does not relay the messages and it does not receive messages in two fault intervals of 500 ms during the simulation. The safety-critical node 4 does not loose any messages, while the non safety-critical nodes 5 and 6 are effected by the crash failure of switch 3. In addition, the latency and jitter of the sporadic messages are effected, which results from the sporadic messages taking a longer path to arrive at the destination address and the shuffling mechanism at the egress port.

In the delay failure scenario, the transmission of messages from node 5 is delayed by 100 ms during a fault time interval of 500 ms. The delayed periodic messages are dropped by the first switch, while the delayed sporadic messages exhibit an increased latency.

ID	Application Type	Sender	Fault Free Case (NE3)		Babbling Idiot Failure (SW2)		Omission Failure (SW3)		Omission Failure (L2)		Link Failure (L3)		Delay Failure (NE5)	
			Latency	Jitter	Latency	Jitter	Latency	Jitter	Latency	Jitter	Latency	Jitter	Latency	Jitter
1	App.1	NE 1	0.10062	0	0.10062	0	0.10062	0	0.1006	0	0.1006	0	0.1006	0
2	App.4	NE 1	1.019175	0	1.01917	0	1.01917	0	1.0192	0	1.0192	0	1.0192	0
3	App.7	NE 2	93.96654	93.946	97.9142	97.8935	122.017	122	119.91	119.89	90.01	89.99	93.967	93.946
4	App.2	NE 3	134.0502	126	Fault Injection		146.067	146.02	110.3	110.23	76.054	75.987	132.05	124
5	App.7	NE 3	155.1907	155.14	Fault Injection		177.245	177.22	151.39	151.37	151.16	151.14	155.19	155.14
6	App.1	NE 4	0.09216	0	0.09216	0	0.09216	0	0.0922	0	0.0922	0	0.0922	0
7	App.2	NE 4	138.7938	138.76	158.05	158.018	139.749	139.05	138.79	138.76	142.19	141.85	166.07	166.03
8	App.4	NE 4	2.032455	0	2.03245	0	2.03245	0	2.0325	0	2.0325	0	2.0325	0
9	App.5	NE 4	70.08917	68.273	70.0892	68.2726	97.6626	97.644	70.089	68.273	71.054	69.041	70.079	68.262
10	App.3	NE 5	5.06685	0	5.06685	0	5.06685	0	5.0668	0	5.0668	0	5.0668	0
11	App.5	NE 5	61.64938	61.636	61.6812	61.6681	88.2622	88.174	57.89	57.877	65.681	65.668	161.65	161.64
12	App.2	NE 6	132.0975	132.05	138.811	138.761	100.163	100.13	134.05	126.04	137.2	157.18	132.1	132.05
13	App.6	NE 6	92.003	91.972	100.058	98.8602	102.105	102.09	92.003	91.972	96.002	95.981	100.05	100.02
14	App.3	NE 7	0.008089	0	0.00809	0	0.00809	0	0.0081	0	0.0081	0	0.0081	0
15	App.6	NE 7	98.06489	78.733	98.143	98.1304	112.26	112.25	97.091	88.377	78.095	78.03	98.065	78.733

Table 8.4 Simulation Results of Evaluation Scenario Based on Ring Topology
(The listed jitter is the average observed jitter in ms of all destination nodes and the listed latency is the average observed latency.)

8.2.3 Evaluation Use-case Based on Gateway

This section presents an evaluation based on a use-case with the proposed off-chip/on-chip gateway architecture. The main objective of the evaluation is to study the real-time behavior of an off-chip/on-chip gateway with networked multi-core chips under different traffic types (i.e. periodic, sporadic and aperiodic). A simulation environment for an off-chip communication system [AO13] was used for the off-chip network. The off-chip/on-chip gateway was implemented in GEM5 and combined with the GARNET NoC to emulate the chip-level.

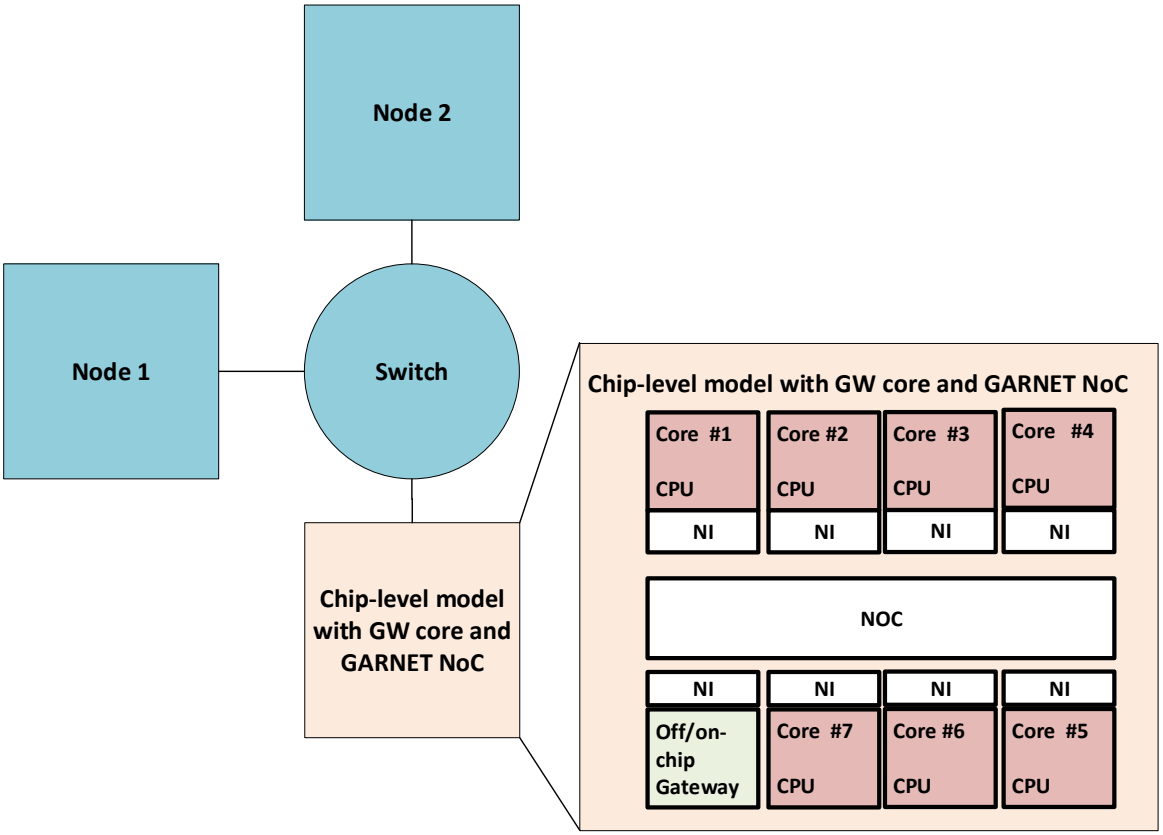


Figure 8.8 Evaluation Use-case Based on Gateway

Figure 8.8 illustrates the use-case topology which consists of two nodes and a multi-core chip that communicate over the off-chip network. The multi-core chip includes seven cores and an off-chip/on-chip gateway interconnected by the GARNET NoC.

Table 8.5 summarizes the traffic generated by the nodes and the cores as well as the observed timing of the communication system. The table contains information of the communicated data (e.g., sender and destination) and the traffic type. The traffic type can be periodic, where the application sends the messages periodically based on the period and

phase, sporadic with minimum and maximum interarrival times, or aperiodic with randomly sent messages.

There are two evaluation scenarios. In the gateway scenario, the proposed off-chip/on-chip gateway is used for the interaction between the off-chip and on-chip networks. In the second scenario, the incoming messages for the off-chip/on-chip gateway are transferred based on first-in/first-out.

Message Exchange and Message Generated							Result using proposed GW		Result using FIFO	
Message Type	App. ID	VLID	Sender (Core #)	Destination (Core #)	Payload size	Message Generator Behaviour	Max. Latency	JiNode r	Max. Latency	JiNode r
Period	1	1	Node 2	NoC (4)	64	Period = 2 ms ; Phase=0	27.3 µs	9 ns	27.3 µs	9 ns
	2	2	Node 1	NoC(2)	80	Period = 2 ms ; Phase=15µs	34.7 µs	12 ns	34.7 µs	12 ns
	3	3	NoC(1)	Node 2	70	Period = 2 ms ; Phase=75 µs	24.3 µs	0	423.3 µs	212.8 µs
	4	4	Node 1	Node 2	64	Period = 2 ms ; Phase= 18 µs	20.8 µs	0	20.8 µs	0
	5	5	Node 2	NoC (3)	100	Period = 2 ms ; Phase=111µs	70.2 µs	12 ns	70.2 µs	12 ns
	6	6	NoC (2)	Node 2	64	Period = 2 ms ; Phase=172µs	37.1 µs	0	201 µs	140 µs
	7	7	Node 1	NoC (1)	100	Period = 2 ms ; Phase=80µs	40.8 µs	0	40.8 µs	0
	8	8	NoC (4)	Node 1	64	Period = 2 ms ; Phase=183µs	35.5	0	114.8 µs	93.3 µs
	9	9	Node 2	NoC (2)	100	Period = 2 ms ; Phase=207µs	39.2 µs	30 ns	39.2 µs	30 ns
	10	10	NoC (3)	Node 2	80	Period = 2 ms ; Phase=298µs	38.4 µs	0	597.3 µs	573 µs
Sporadic	11	11	Node 2	NoC (1)	80	U(1.9ms ,2.1 ms)	328.2 µs	101.1 µs	341.7 µs	310.3 µs
	12	12	NoC (4)	Node 2	70	U(1.9ms ,2.1 ms)	573.8 µs	85.4 µs	685 µs	645 µs
	13	13	Node 1	Node 2	100	U(1.9ms ,2.1 ms)	433.7 µs	95.8 µs	427.3 µs	92.3 µs
	14	14	Node 2	NoC (2)	64	U(1.9ms ,2.1 ms)	879.2 µs	113.5 µs	863.6 µs	798.5 µs
	15	15	NoC (1)	Node 1	200	U(1.9ms ,2.1 ms)	174.7 µs	120.3 µs	609.3 µs	534.8 µs
	16	16	Node 1	NoC (3)	100	U(1.9ms ,2.1 ms)	582.4 µs	143.9 µs	546.2 µs	470.6 µs
	17	17	NoC (3)	Node 1	80	U(1.9ms ,2.1 ms)	586 µs	189µs	945.7 µs	835.7 µs
	18	18	Node 2	NoC (4)	70	U(1.9ms ,2.1 ms)	773.3 µs	124.7 µs	753.2 µs	119.3 µs
	19	19	NoC (2)	Node 1	64	U(1.9ms ,2.1 ms)	425 µs	187 µs	704.5 µs	270.7 µs
	20	20	Node 1	Node 2	100	U(1.9ms ,2.1 ms)	453.2 µs	103.4 µs	479.4 µs	121.7 µs
Aperiodic	21	-	NoC (1)	Node 2	80	1000 messages each 1 s	1.2 ms	1.11 ms	971.2 µs	959.4 µs
	22	-	NoC (4)	Node 1	100	1000 messages each 1 s	813 µs	779.7 µs	1.081 ms	1.062 ms
	23	-	NoC (2)	Node 2	64	1000 messages each 1 s	1.4 ms	1.355 ms	1.028 ms	1.002 ms
	24	-	NoC (3)	Node 2	70	1000 messages each 1 s	1.488 ms	1.408 ms	1.097 ms	1.061 ms
	25	-	NoC (2)	Node 1	100	1000 messages each 1 s	1.492 ms	1.432 ms	1.028 ms	993 µs
	26	-	NoC (3)	Node 1	64	1000 messages each 1 s	1.435 ms	1.377 ms	1.032 ms	999 µs
	27	-	Node 2	NoC (1)	64	1000 messages each 1 s	2.06 ms	1.483 ms	2.034 ms	1.347 ms
	28	-	Node 2	NoC (2)	100	1000 messages each 1 s	2.859 ms	2.125 ms	2.92 ms	2.359 ms
	29	-	Node 1	NoC (3)	80	1000 messages each 1 s	2.344 ms	1.814 ms	2.434 ms	1.853 ms
	30	-	Node 1	NoC (4)	80	1000 messages each 1 s	2.396 ms	1.789 ms	2.378 ms	1.821 ms

Table 8.5 Message Exchange in the Evaluation Use-case Based on Gateway and Simulation Results

Table 8.5 lists the simulation results for the two scenarios of the evaluation use-case. We observed significant discrepancies of the end-to-end jitter, i.e., the difference between the maximum and minimum end-to-end latency between applications.

In the gateway scenario, the jitter of the periodic messages that are sent from the gateway to the off-chip network is *zero* (when abstracting from the limited precision of the global

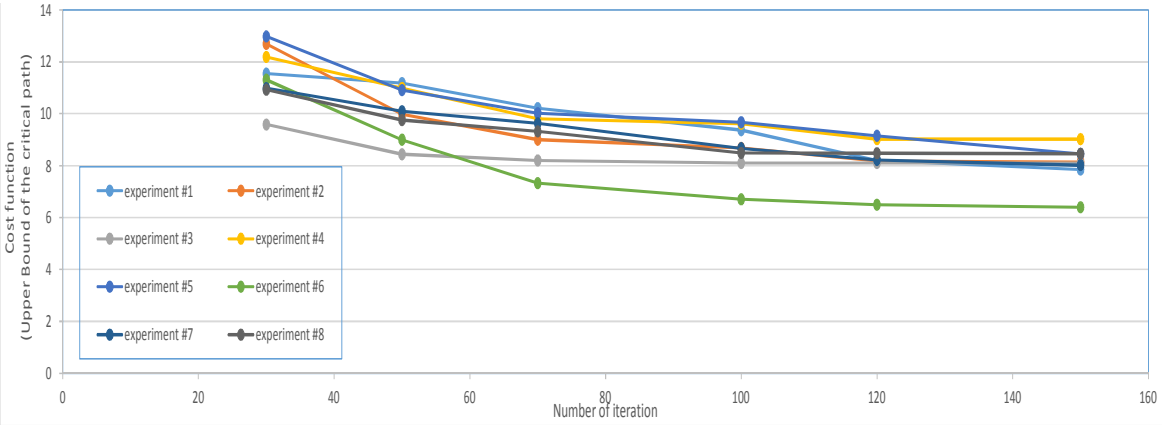


Figure 8.9 Upper bound for critical path delay after different iterations

time base). The jitter of the periodic messages from the off-chip network to the on-chip network is in the range of nano seconds. This jitter results from the sending of messages through the interconnection network without guarantees for periodic messages. The jitter of the sporadic message is in the range of micro seconds, whereas the aperiodic messages have no real-time guarantees. The latency and jitter of the aperiodic messages are higher than for the other message types.

In the second evaluation scenario where the getaway is not used, the latency and jitter of the periodic messages from the gateway to the off-chip network increases (e.g., App.ID 3, App.ID 8). Here, a non real-time message delays the sending of the periodic messages. The latency and the jitter of the periodic messages from the off-chip network to the on-chip network are unaffected because of the guarantees provided by the off-chip switch.

The latency and jitter of the sporadic messages from the off-chip network to the on-chip network are bounded, because the sending of sporadic messages is constrained by the minimum and maximum interarrival times. However, the latency and the jitter of the sporadic messages from the gateway to the off-chip network increase significantly (e.g., App.ID 12, App.ID 14). The latency and the jitter of the aperiodic messages is slightly better than in the first scenario.

8.2.4 Evaluation of Scheduling Algorithm

In this section, the scheduling algorithm is evaluated using the validation framework that presented in Section 8.1.3. The input parameters of this framework comprise 10 switches, 40 nodes, 89 network channels, and a PCL of 30%. The nodes execute different numbers of periodic and sporadic services with execution times between 10μs and 150μs. The dependencies of these services are generated randomly in four graphs and the PCS is 30%.

The message size is selected randomly from values between 64 bytes and 1500 bytes. The periodic services are executed periodically, which means that every service is executed once in the period. The period of each graph is selected randomly from one of the following periods $\{1,2,4,8,16,32,64,128ms\}$. The sporadic services are executed sporadically based on the minimum and maximum interarrival time. The next point in time for executing a sporadic service is calculated based on the uniform distribution between the minimum and maximum interarrival time.

In our use-case, this uniform distribution is selected randomly from one of the following: U(0.9ms,1.1ms), U(1.9ms,2.1ms), U(3.9ms,4.1ms), U(7.9ms,8.1ms), U(15.9ms,16.1ms), U(31.9ms,32.1ms), U(63.9ms,64.1ms), U(127.9ms,128.1ms).

We generated eight different platforms and eight different applications. Each logical model is represented using two graphs: a graph with 50 periodic services and another one with 50 sporadic services.

For each of the eight models, we tested the scheduling algorithm with different numbers of iterations using the input parameters that were explained in the above use-case. The value of the cost function (i.e., upper bound of delays on the critical path) is shown for different numbers of iterations in Figure 8.9. Each color represents one of the eight input models to the scheduling algorithm. Figure 8.9 provides insight into the effect of the number of iterations on critical path delay.

The number of iterations required for a "near optimal" solution depends on the total number of services used in the simulation.

Table 8.6 shows the results from the simulation framework. Table 8.6 contains an overview of the observed mean delays, the maximum delay and jitter of the experiments for the periodic and sporadic communication.

Number of Experiments	Max. Latency of Periodic Msgs.	Periodic Jitter	Max. Latency of Sporadic Msgs.	Sporadic Jitter
50	18.51 ms	0.013 ms	31.93 ms	16.82 ms

Table 8.6 Use-case Result from Simulation Environment

8.3 Discussion and Interpretation of Results

In this dissertation, we have proposed an architecture for mixed-criticality systems based on networked multi-core chips. Mixed-criticality systems integrate on the same platform multiple functions with different importance and certification assurance levels. Moreover, we

provide separation mechanisms such that the functions of different criticality levels are isolated, so they cannot influence each other. The foundation for the separation are mechanisms for temporal and spatial partitioning [Rus99b], which establish fault containment and the absence of unintended side-effects between functions. Partitions encapsulate resources temporally (e.g., latency, jitter, duration of availability during a scheduled access) and spatially (e.g., prevent functions from altering code or private data of other partitions).

In order to improve the reliability of the communication protocols, we provide fault tolerance mechanisms for distributed real-time communication networks based on redundant transmissions. In redundant transmissions, data packets are duplicated and sent in an intelligent way to reduce the probability of losing both packets. Furthermore, this dissertation introduced a novel architecture for selective fault-tolerance based on a ring topology with support for real-time requirements, mixed-criticality integration, different traffic types and error containment.

The increase in complexity imposed by these mechanisms is minimized by building upon the existing services of real-time Ethernet and by exploiting the three traffic types of periodic, sporadic, and aperiodic communication.

The scheduling problem for these systems requires to take decisions on the duplication and fusion of messages, the deployment of safety-critical and non safety-critical nodes as well as the topology (e.g., avoidance of loops).

Selective Fault-Tolerance

The proposed ring architecture supports selective fault-tolerance, which permits a balanced trade off between cost and fault-tolerance for each subsystem of a mixed-criticality system. Fault-tolerance can be adjusted at the level of nodes and individual messages:

1. *Redundant nodes with duplicate messages:* Redundant nodes have connections to two switches and two duplicates of each message are sent to switches. Consequently, a failure of a switch or the physical link to the switch can be tolerated.

2. *Redundant nodes with single messages:* In case of less important messages, a single copy can be sent to one of the switches in order to reduce the load of the communication system.

3. *Non redundant nodes with duplicate messages:* Non redundant nodes have only a single connection to one switch. This switch, which is immediately connected to the node, can perform a duplication of the messages and send two redundant copies to two neighbors. Consequently, the failure of any switch except for the one that is

immediately connected to the non redundant node can be tolerated. Also, the switch immediately connected to a node can fuse the redundant messages, thereby hiding the replication from the node.

4. *Non redundant nodes with single messages:* Communication load can be reduced by sending only a single message without duplication of the switch, thereby losing however the ability to tolerate switch failures.

Error containment has been establishment in the switch for failures of nodes affecting periodic and sporadic messages. In addition, we introduce redundant channels using heterogeneous paths for periodic and sporadic messages to protect against the failure of any network component such as a link or a switch.

Evidence for the fault-tolerance mechanisms is provided by the simulation environment for the proposed architecture, which has been developed using OPNET. In the evaluation different fault scenarios are compared with fault-free scenarios based on the failure modes of IEC-61508-2.

The simulation results demonstrate that the new architecture supports mixed-criticality systems and guarantees real-time communication with bounded delays and minimal jitter. The periodic messages were not affected in the different fault scenarios, while the sporadic messages show bounded fault-effects.

Support for Different Traffic Types

The fault-tolerant architecture supports the communication needs of mixed-criticality based on the three traffic types of real-time Ethernet. Mixed-criticality subsystems typically differ not only with respect to their safety and reliability requirements, but also in the underlying timing models. Periodic messages serve for safety-relevant control applications with high temporal regularity. Such applications require temporal predictability with minimum delays and jitter to ensure a high quality of control. We guarantee real-time support with bounded end-to-end latency and small jitter of the time-triggered messages that are sent according to a predefined scheduling table.

Sporadic messages support a better bandwidth utilization at the cost of weaker temporal guarantees. Sporadic messages provide bounded delays, but higher jitter than periodic messages. Aperiodic communication aims at non safety-relevant communication without temporal guarantees.

Support for Ring Topology

In the state-of-the-art, many network topologies can be found, which differ w.r.t. performance metrics [DT03, De03] such as throughput, maximum channel load, latency and fault-tolerance. On the other hand, the topology involves cost through the number of links and complexity for routing and scheduling.

A backbone is an example of a linear bus, where each node is attached to a linear trunk line. However, a bus topology is not scalable because the bus becomes the bottleneck when more nodes are added. A further problem is the susceptibility to common failure modes.

Indirect or switch-based networks are another class of interconnection networks, which can exhibit predefined patterns or irregular topologies. An important parameter determining the latency is the diameter of the topology. The bisection bandwidth, the node and edge connectivity are important parameters for the network's fault-tolerance. The cost is significantly effected by the degree of nodes and the number of physical links.

A complete graph would be ideal w.r.t. fault-tolerance and latency. However, it would involve prohibitive cost and limited scalability, because the number of physical connections of a switch is limited by hardware constraints.

A ring with n switches offers a diameter of $\lfloor \frac{n}{2} \rfloor$, an edge connectivity of 2 and a bisection bandwidth of 2. It is thus ideal for a single fault hypothesis (cf. fault assumption in Section 4.1.3).

An alternative would be a d-dimensional mesh. A degree of $2d$ results in a diameter of $d(\sqrt[d]{n}-1)$, an edge connectivity of d and a bisection bandwidth of $n^{\frac{d-1}{d}}$ [RR13]. However, the significant additional cost is not required based on the single fault hypothesis.

Support for Scheduling of Heterogeneous Timing Models

The scheduling of applications with inter-job dependencies and different traffic classes such as periodic time-triggered and sporadic rate-constrained traffic is important for the use of real-time Ethernet in mixed-criticality systems. The presented scheduling and validation framework enables the systematic evaluation of scheduling algorithms for large numbers of test cases based on a generic definition of test scenarios and the automatic generation of application and platform models. In our tool-chain, the application and platform models are directly used as input to a real-time Ethernet simulation environment to provide feedback on the temporal behavior.

Support for Hierarchical Architecture

Many upcoming mixed-criticality systems use networked multi-core platforms due to resource requirements exceeding a single chip. Since several multi-core chips for mixed-criticality systems use on-chip networks, gateways between off-chip and on-chip networks are required.

This dissertation has introduced an off-chip/on-chip gateway architecture that supports different timing models including periodic time-triggered, sporadic rate-constrained and aperiodic communication. Different timing models are significant in many mixed-criticality systems, because subsystems with different safety assurance levels are often based on different models of computation (e.g., safety-critical time-triggered control application vs. non safety-relevant event-triggered comfort systems).

Support for Temporal and Spatial Partitioning

The proposed architecture is based on synchronous global time, which is globally synchronized within the system of hierarchical networked multi-core chips. The deterministic communication and the temporal activities are established with respect to this global time. The proposed architecture supports the determinism and real-time requirements using a scheduler that enforces temporal constraints. The scheduler ensures deterministic behavior for the real-time messages by establishing temporal segregation and ensuring isolation of the synchronous real-time messages from other asynchronous messages.

Furthermore, the proposed architecture provides the spatial partitioning using protected time slots to guarantee that real-time messages are not influenced by other non-critical messages in the system.

Evaluation Use-cases

For the evaluation of the proposed architecture, we provide four use-cases:

- The automotive example demonstrates how the simulation framework can be used to gain insight into the timing and reliability of a distributed control application based on different configurations of the real-time Ethernet communication system. Sporadic rate-constrained communication has resulted in higher variability of communication latencies with a negative impact on the quality of control in the cruise control function. A babbling idiot has a considerable effect on the application behavior in case of sporadic

rate-constrained communication, whereas periodic time-triggered traffic establishes fault isolation based on the static communication schedule.

- The ring topology example exhibits how the proposed fault tolerance mechanism for the system can be used to increase the safety and reliability of the system. This evaluation example shows that the periodic time triggered messages serve for safety-relevant control applications with high temporal regularity. The proposed architecture can guarantee real-time support with bounded end-to-end latency and small jitter of the periodic time-triggered messages. The sporadic messages can serve in the safety-relevant applications that can tolerate higher jitter, where sporadic messages offer a better bandwidth utilization at the cost of weaker temporal guarantees. Sporadic messages provide bounded delays, but higher jitter than periodic messages. Aperiodic communication aims at non safety-relevant communication without temporal guarantees.

- In the off-chip/on-chip gateway use case, the proposed architecture was compared to the behavior of a normal gateway (first in first out) with aperiodic communication. The simulation results demonstrate that the new architecture supports spatial and temporal guarantees with bounded delays and minimal jitter.ate the proposed scheduling. This validation framework provides feedback on the runtime of the scheduler and the temporal behavior of the ensuing system.

Chapter 9

Conclusion

The importance of large-scale mixed-criticality systems comprised of networked multi-core chips is increasing in many application domains. This dissertation has introduced an architecture for mixed-criticality systems based on networked multi-core chips with techniques for real-time support, mixed-criticality integration, different traffic types, fault tolerance and error containment. The architecture models encompass both the on-chip level and the off-chip level.

At the on-chip level, the architecture is built on top of the existing on-chip interconnects (e.g., Æthereal, STNoC), which they employ either a TDMA scheme or priority-based virtual networks to establish resource guarantees with respect to bandwidth and latency. To bridge between the on-chip level and the off-chip level, gateways are introduced for different integration levels while providing real-times guarantees, fault isolation and mixed-criticality capability.

At the off-chip level, the architecture is compatible with the Ethernet since this protocol has become attractive for many industrial domains such as automotive, avionics and railway. The architecture gives solutions to enable mixed-criticality communication allowing hard real-time and non safety-critical traffic such as aperiodic best effort messages to coexist simultaneously within one physical system. Moreover, the temporal and spatial partitioning, time guarantees as well as fault isolation are supported in the architecture.

Furthermore, this dissertation has introduced techniques for selective fault-tolerance using a ring topology with support for real-time requirements, mixed-criticality integration, different traffic types and error containment. The real-time support is guaranteed by the bounded end-to-end latency and small jitter of the periodic time-triggered messages that are sent according to a predefined scheduling table. Mixed-criticality is supported by safety and non safety-critical nodes. The safety-critical nodes are connected to two switches using redundant links, whereas non safety-critical nodes use a single link to one switch only.

Error containment has been establishment in the switch for failures of nodes affecting periodic time-triggered and sporadic rate-constrained messages. In addition, we introduce redundant channels using heterogeneous paths for periodic time-triggered and sporadic rate-constrained messages to protect against the failure of any network component such as a link or a switch.

In such large-scale mixed-criticality systems, techniques for scheduled end-to-end communication with resource reservations and temporal guarantees are required. In this dissertation, heuristic scheduling techniques for periodic time-triggered and sporadic rate-constrained messages in a multi-hop network were introduced. The presented scheduling and validation framework enables the systematic evaluation of scheduling algorithms for large numbers of test cases based on a generic definition of test scenarios and the automatic generation of application and platform models.

Different traffic types including periodic time-triggered, sporadic rate-constrained and aperiodic best-effort are supported in the simulation environment. Generic building blocks of the simulation environment comprise nodes (i.e. multi-core node and single core node) and switches, which can be instantiated and configured based on application-specific communication schedules (e.g., period and phase of periodic time-triggered messages) and communication parameters (e.g., minimum interarrival time of sporadic rate-constrained messages). The simulation environment enables system developers to investigate the implications of the choice of different traffic types and communication schedules on the timing and reliability of the system. To evaluate the architecture models, different use case are used. The simulation results provide experimental evidence for the real-time support and fault isolation.

Bibliography

[80211] IEEE Standard 802.1Q-2011. *Media Access Control (MAC) Bridges and Virtual Bridge Local Area Networks*, 2011.

[AB06] César Rego A and Renato Duarte B. A filter and fan approach to the job shop scheduling problem, 2006.

[AC03] J Arlat and Y Crouzet. Comparison of physical and software-implemented fault injection techniques. *IEEE Transactions on Computers*, 52(9):1115–1133, 2003.

[AER04] H. Ammari and H. El-Rewini. Integration of mobile ad hoc networks and the internet using mobile gateways. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, pages 218–, April 2004.

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

[ALRL04] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *Dependable and Secure Computing, IEEE Transactions on*, 1(1):11–33, Jan 2004.

[AO13] M. Abuteir and R. Obermaisser. Simulation environment for time-triggered ethernet. In *2013 11th IEEE International Conference on Industrial Informatics (INDIN)*, pages 642–648, July 2013.

[AO14] M. Abuteir and R. Obermaisser. Mixed-Criticality Systems Based on Time-Triggered Ethernet with Multiple Ring Topologies. In *IEEE International Symposium on Industrial Embedded Systems (SIES), 2014 9th*, 2014.

[AO15] M. Abuteir and R. Obermaisser. Scheduling of rate-constrained and time-triggered traffic in multi-cluster ttethernet systems. In *Industrial Informatics (INDIN), 2015 13th IEEE International Conference on*, July 2015.

[AOA16] H. Ahmadian, R. Obermaisser, and M. Abuteir. Time-triggered and rate-constrained on-chip communication in mixed-criticality systems. In *2016 IEEE 10th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSOC)*, pages 117–124, Sept 2016.

[AOOM15] M. Abuteir, R. Obermaisser, Z. Owda, and T. Moudouthe. Off-chip/on-chip gateway architecture for mixed-criticality systems based on networked multi-core chips. In *2015 IEEE 18th International Conference on Computational Science and Engineering*, Oct 2015.

[ari05] Aircraft data network part 7 avionics full duplex switched ethernet (AFDX) network, June 27 2005.

[ASe03] A. Ademaj, H. Sivencrona, and et al. Evaluation of fault handling of the time-triggered architecture with bus and star topology. In *Proc. of the Int. Conference on Dependable Systems and Networks*, 2003.

[ASS+08] E. Althammer, E. Schoitsch, G. Sonneck, H. Eriksson, and J. Vinter. Modular certification support the decos concept of generic safety cases. In *Industrial Informatics, 2008. INDIN 2008. 6th IEEE International Conference on*, pages 258–263, July 2008.

[AVB16] Iso/iec/ieee international standard - information technology – telecommunications and information exchange between systems – local and metropolitan area networks – specific requirements – part 1ba: Audio video bridging (avb) systems. *ISO/IEC/IEEE 8802-1BA First edition 2016-10-15*, pages 1–52, Oct 2016.

[BAP05] M. Barranco, L. Almeida, and J. Proenza. ReCANcentrate: A replicated star topology for CAN networks. In *Proceedings of the 10th IEEE Conference on Emerging Technologies and Factory Automation (ETFA 2005)*, volume 2, pages 8 pp. –476, sept. 2005.

[BBB+09] James Barhorst, Todd Belote, Pam Binns, Jon Hoffman, James Paunicka, Prakash Sarathy, John Scoredos, Peter Stanfill, Douglas Stuart, and Russel Urzi. A research agenda for Mixed-Criticality systems. In *Cyber-Physical Systems Week*, 4 2009.

[BCSM08] B.D. Bui, M. Caccamo, Lui Sha, and J. Martinez. Impact of cache partitioning on multi-tasking real time embedded systems. In *Embedded and Real-Time Computing Systems and Applications, 2008. RTCSA '08. 14th IEEE International Conference on*, pages 101–110, Aug 2008.

[BDM06] Luca Benini and Giovanni De Michelli. *Networks on chips : technology and tools*. The Morgan Kaufmann series in systems on silicon. Elsevier Morgan Kaufmann Publishers, Amsterdam, Boston, Paris, 2006.

[BE83] E.A. Benhamou and J. Estrin. Multilevel internetworking gateways: Architecture and applications. *Computer*, 16(9):27–34, Sept 1983.

[Ben96] A. Bender. Design of an optimal loosely coupled heterogeneous multiprocessor system. In *European Design and Test Conference, 1996. ED TC 96. Proceedings*, pages 275–281, Mar 1996.

[BK10] Diane Barrett and Gregory Kipper. *Virtualization and Forensics*. Syngress, Boston, 2010.

[BPH98] A. Meyna B. Pauli and P. Heitmann. liability of electroniccomponents and control units in motor vehicle applications. *In VDI Berichte 1415, Electronic Systems for Vehicles*, page 1009–1024, 1998.

[BS05] T. Bjerregaard and J. Sparso. A router architecture for connection-oriented service guarantees in the mango clockless network-on-chip. In *Design, Automation and Test in Europe, 2005. Proceedings*, pages 1226–1231 Vol. 2, March 2005.

[25] Certification Authorities Software Team (CAST). Position paper on multi-core processors - cast-32. 2014.

[Car16] Thomas Carlsson. Hms - industrial network market shares 2016 according to hms. *HMS - Industrial network market shares 2016 according to HMS*, Feb 2016.

[Cer14] Certification Authorities Software Team (CAST). Position paper cast-32 multi-core processors. Technical report, 2014.

[CGL+08] Marcello Coppola, Miltos D. Grammatikakis, Riccardo Locatelli, Giuseppe Maruccia, and Lorenzo Pieralisi. *Design of Cost-Efficient Interconnect Processing Units: Spidergon STNoC*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 2008.

[cit09] ISO/DIS 26262-1 - Road vehicles -Functional safety- Part 1: Vocabulary. Technical report, Geneva, Switzerland, July 2009.

[CMFC+98] A. B. Campbell, O. Musseau, V. Ferlet-Cavrois, W. J. Stapor, and P. T. McDonald. Analysis of single event effects at grazing angle [cmos srams]. *IEEE Transactions on Nuclear Science*, 45(3):1603–1611, Jun 1998.

[Com05] Airlines Electronic Engineering Committee. Aircraft Data Network Part 7 Avionics Full Duplex Switched Ethernet AFDX network, 2005.

[Com11] Avionics Interface Technologies Company. White paper: SAE AS6802 deterministic ethernet network solution. Technical report, March 2011.

[Con14] DREAMS Consortium. D1.2.1 distributed real-time architecture for mixed criticality systems. 2014.

[CRM10] A. Crespo, I. Ripoll, and M. Masmano. Partitioned embedded architecture based on hypervisor: The xtratum approach. In *Dependable Computing Conference (EDCC), 2010 European*, pages 67–72, April 2010.

[DBN14] S.K. Datta, C. Bonnet, and N. Nikaein. An iot gateway centric architecture to provide novel m2m services. In *Internet of Things (WF-IoT), 2014 IEEE World Forum on*, pages 514–519, March 2014.

[DC97] J. Desbonnet and P.M. Corcoran. System architecture and implementation of a cebus/internet gateway. *Consumer Electronics, IEEE Transactions on*, 43(4):1057–1062, Nov 1997.

[DCC+11] F. Dubois, J. Cano, M. Coppola, J. Flich, and F. Petrot. Spidergon STNoC design flow. In *Proc. of 5th IEEE/ACM Int. Symposium on Networks on Chip*, pages 267–268, May 2011.

[De03] J. Duato and et al. *Interconnection Networks – An Engineering Approach*. Elsevier, 2003.

[DI11] DDC-I. *DEOS – A Time & Space Partitioned DO-178 Level A Certifiable Family of RTOS Products*, 2011.

[DT03] William Dally and Brian Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[ECR05] Airlines Electronic, Engineering Committee, and Aeronautical Radio. *Aircraft Data Network Part 7 Avionics Full Duplex Switched Ethernet (Afdx) Network Arinc Specification 664P7*. 2005.

[EDPP00] P. Eles, A. Doboli, P. Pop, and Zebo Peng. Scheduling with bus access optimization for distributed embedded systems. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 8(5):472–491, Oct 2000.

[EKP+98] P. Eles, K. Kuchcinski, Zebo Peng, A. Doboli, and P. Pop. Process scheduling for performance estimation and synthesis of hardware/software systems. In *Euromicro Conference, 1998. Proceedings. 24th*, volume 1, pages 168–175 vol.1, Aug 1998.

[Esw09] A. Eswaramurthy, V.; Tamilarasi. Hybridizing tabu search with ant colony optimization for solving job shop scheduling problems. *International Journal of Advanced Manufacturing Technology*, 40 Issue 9/10:1004, April 2009.

[F. 03] F. Madren. Redundancy with standards in industrial Ethernet LANs. In *A White Paper for Network Engineers in Factories, Transportation Systems, Utilities, and Other Heavy Duty Networking Applications*. 2003.

[FFR+11] P. Ferrari, A. Flammini, S. Rinaldi, G. Prytz, and P.C. Juel. Architecture of an embedded time gateway between ptp and sntp. In *Industrial Embedded Systems (SIES), 2011 6th IEEE International Symposium on*, pages 71–74, June 2011.

[Fle04] 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 2.0*, July 2004.

[For07] B.A. Forouzan. *Data Communications and Networks*. Tata McGraw-Hill, 2007.

[GDR05] K. Goossens, J. Dielissen, and A. Radulescu. Aethereal network on chip: concepts, architectures, and implementations. *Design Test of Computers, IEEE*, 22(5):414–421, Sept 2005.

[GH10] Kees Goossens and Andreas Hansson. The aethereal network on chip after ten years: Goals, evolution, lessons, and future. In *Proceedings of the 47th Design Automation Conference*, DAC '10, pages 306–311, New York, NY, USA, 2010. ACM.

[Han06] R.S. Hanmer. Error containment. In *Proc. of the 2006 Conference on Pattern Languages of Programs*, PLoP '06, pages 18:1–18:11, New York, NY, USA, 2006. ACM.

[HDPDB05] B. Hall, K. Driscoll, M. Paulitsch, and S. Dajani-Brown. Ringing out fault tolerance. a new ring network for superior low-cost dependability. In *Dependable Systems and Networks, 2005. DSN 2005. Proceedings. International Conference on*, pages 298–307, June 2005.

[IEC10a] Functional safety of electrical/electronic/programmable electronic safety-related systems – part 1: General requirements. IEC 61508-1, 2010.

[IEC10b] Functional safety of electrical/electronic/programmable electronic safety-related systems – part 2: Requirements for electrical / electronic / programmable electronic safety-related system. IEC 61508-2, 2010.

[IEC10c] Functional safety of electrical/electronic/programmable electronic safety-related systems – part 3: Software requirements iec 61508. EC 61508-3, 2010.

[IEC10d] Industrial communication networks - profiles - part 3-13: Functional safety fieldbuses - additional specifications for cpf 13. IEC 61784-3, 2010.

[IEC10e] IEC. 61508 functional safety of electrical/electronic/programmable electronic safety-related systems. *International electrotechnical commission*, 2010.

[IEE04] IEEE. IEEE Standard for Local and metropolitan area networks: Media Access Control (MAC) Bridges. *IEEE Std 802.1D-2004 (Revision of IEEE Std 802.1D-1998)*, pages 1–277, 2004.

[IEE13] IEEE 802.1Qbv IEEE. *Enhancements for Scheduled Traffic, Draft 0.2*, 2013.

[Int14] Intel. Intel® virtualization technology for directed i/o architecture specification. October 2014.

[ISO93] ISO-11898. *Road vehicles – Interchange of Digital Information – Controller Area Network (CAN) for High-Speed Communication*. Int. Standardization Organisation, ISO 11898, 1993.

[JLLK07] Hyo-Moon Jeong, Myung-Jin Lee, Dong-Kyu Lee, and Soon-Ju Kang. Design of home network gateway for real-time a/v streaming between ieee1394 and ethernet. *Consumer Electronics, IEEE Transactions on*, 53(2):390–396, May 2007.

[KA96] Yu-Kwong Kwok and I. Ahmad. Dynamic critical-path scheduling: an effective technique for allocating task graphs to multiprocessors. *Parallel and Distributed Systems, IEEE Transactions on*, 7(5):506–521, May 1996.

[KG14] B. Kisacanin and M. Gelautz. *Advances in Embedded Computer Vision*. Springer, 2014.

[KL99] J. Kaiser and M.A. Livani. Achieving fault-tolerant ordered broadcasts in CAN. In *Proc. of European Dependable Computing Conference*, pages 351–363, 1999.

[KOAAT14] A. Khalifeh, R. Obermaisser, M. Abuteir, and D. Abou-Tair. Systems-of-systems framework for providing real-time patient monitoring and care. In *Proceedings of the 8th International Conference on Pervasive Computing Technologies for Healthcare*, ICST, Brussels, Belgium, Belgium, 2014.

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

[Kop04a] H. Kopetz. The fault hypothesis for the time-triggered architecture. In *Proc. of the IFIP World Computer Congress*, 2004.

[Kop04b] Hermann Kopetz. An integrated architecture for dependable embedded systems. In *Reliable Distributed Systems, 2004. Proceedings of the 23rd IEEE International Symposium on*, pages 160–161, Oct 2004.

[Kop11] Hermann Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications*. 2011.

[KS90] D.B. Kirk and J.K. Strosnider. Smart (strategic memory allocation for real-time) cache design using the mips r3000. In *Real-Time Systems Symposium, 1990. Proceedings., 11th*, pages 322–330, Dec 1990.

[KsH10] H. Kimm and Ho sang Ham. Integrated fault tolerant system for automotive bus networks. In *Proc. of 2010 Int. Conference on Computer Engineering and Applications*, volume 1, pages 486–490, 2010.

[LA90] P. A. Lee and T. Anderson. *Fault Tolerance: Principles and Practice*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2nd edition, 1990.

[LAK92] J.C. C. Laprie, A. Avizienis, and H. Kopetz, editors. *Dependability: Basic Concepts and Terminology*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1992.

[Liv99] M.A. Livani. SHARE: A transparent approach to fault-tolerant broadcast in CAN. In *Proc. of 6th Int. CAN Conference (ICC6)*, November 1999.

[LJ07] Zhonghai Lu and Axel Jantsch. Slot allocation using logical networks for tdm virtual-circuit configuration for network-on-chip. In *Proceedings of the 2007 IEEE/ACM International Conference on Computer-aided Design*, ICCAD '07, pages 18–25, Piscataway, NJ, USA, 2007. IEEE Press.

[LTMJ05] Zhonghai Lu, Rikard Thid, Mikael Millberg, and Axel Jantsch. Nnse: Nostrum network-on-chip simulation environment. In *In Proc. of SSoCC*, 2005.

[Lyn02] Lynuxworks. Lynxos user's guide, release 4.0. 2002.

[MMT10] J. Marcello, C.E. Moron, and L.C. Trevelin. A gateway architecture for qos management considering time constraint application. In *Systems Man and Cybernetics (SMC), 2010 IEEE International Conference on*, pages 1300–1305, Oct 2010.

[MSB+05] Milo M. K. Martin, Daniel J. Sorin, Bradford M. Beckmann, Michael R. Marty, Min Xu, Alaa R. Alameldeen, Kevin E. Moore, Mark D. Hill, and David A. Wood. Multifacet's general execution-driven multiprocessor simulator (gems) toolset. *SIGARCH Comput. Archit. News*, 33(4):92–99, November 2005.

[MT89] Mihajlo D. Mesarovic and Yasuhiko Takahara, editors. *Abstract Systems Theory*, volume 116 of *Lecture Notes in Control and Information Sciences*. Springer-Verlag, Berlin/Heidelberg, 1989.

[MTCM05] A. Mello, L. Tedesco, N. Calazans, and F. Moraes. Virtual channels in networks on chip: Implementation and evaluation on hermes noc. In *Integrated Circuits and Systems Design, 18th Symposium on*, pages 178–183, Sept 2005.

[MZZM13] M. Mechtri, D. Zeghlache, E. Zekri, and I.J. Marshall. Inter-cloud networking gateway architecture. In *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, volume 2, pages 188–194, Dec 2013.

[NYC06] Rabia Nessah, Farouk Yalaoui, and Chengbin Chu. A branch and bound algorithm to minimize total weighted completion time on identical parallel machines with job release dates. In *Service Systems and Service Management, 2006 International Conference on*, volume 2, pages 1192–1198, Oct 2006.

[OAKAT15] R. Obermaisser, M. Abuteir, A. Khalifeh, and D. Abou-Tair. Systems-of-systems framework for providing real-time patient monitoring and care: Challenges and solutions. In *Communications in Computer and Information Science*, volume 515. 2015.

[OAO15] Z. Owda, M. Abuteir, and R. Obermaisser. Co-simulation framework for networked multi-core chips with interleaving discrete event simulation tools. In *2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA)*, Sept 2015.

[OAOD14] Z. Owda, M. Abuteir, R. Obermaisser, and H. Dakheel. Predictable and reliable time triggered platform for ambient assisted living. In *2014 8th International Symposium on Medical Information and Communication Technology (ISMICT)*, April 2014.

[OAOK14] K. Okano, Y. Aoki, T. Ohta, and Y. Kakuda. An autonomous clustering-based inter-domain routing protocol for heterogeneous mobile ad hoc networks. In *Mobile Ad-hoc and Sensor Networks (MSN), 2014 10th International Conference on*, pages 144–150, Dec 2014.

[Obe05] Roman Obermaisser. *Event-Triggered and Time-Triggered Control Paradigms*, volume 22. 2005.

[Obe11] R. Obermaisser. *Time-triggered communication*. Taylor & Francis, 2011.

[Obe12] Roman Obermaisser. *Time-triggered communication*. Embedded systems. Taylor & Francis, Boca Raton, 2012.

[OEHK08] Roman Obermaisser, C. El Salloum, B. Huber, and Hermann Kopetz. The time-triggered system-on-a-chip architecture. In *Industrial Electronics, 2008. ISIE 2008. IEEE International Symposium on*, pages 1941–1947, 2008.

[OESHK08] R. Obermaisser, C. El Salloum, B. Huber, and Hermann Kopetz. The time-triggered system-on-a-chip architecture. In *Industrial Electronics, 2008. ISIE 2008. IEEE International Symposium on*, pages 1941–1947, June 2008.

[OESHK09] R. Obermaisser, C. El Salloum, B. Huber, and Hermann Kopetz. From a federated to an integrated automotive architecture. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 28(7):956–965, July 2009.

[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 *Dependable Computing Conference, 2008. EDCC 2008. Seventh European*, pages 123–134, May 2008.

[OOA+14] R. Obermaisser, Z. Owda, M. Abuteir, H. Ahmadian, and D. Weber. End-to-end real-time communication in mixed-criticality systems based on networked multicore chips. In *Digital System Design (DSD), 2014 17th Euromicro Conference on*, Aug 2014.

[OP06] R. Obermaisser and P. Peti. A fault hypothesis for integrated architectures. In *Intelligent Solutions in Embedded Systems, 2006 International Workshop on*, pages 1–18, 2006.

[OPN15] OPNET Technologies. *OPNET Modeler 17.1 Documentation*, 2015.

[oR11] Special C. of RTCA. DO-178C, software considerations in airborne systems and equipment certification, 2011.

[OUOA16] Z. Owda, M. Urbina, R. Obermaisser, and M. Abuteir. Hierarchical transactional memory protocol for distributed mixed-criticality embedded systems. In *2016 IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing*, pages 334–343, Aug 2016.

[PAG+16] P. Petrakis, M. Abuteir, M. D. Grammatikakis, K. Papadimitriou, R. Obermaisser, Z. Owda, A. Papagrigoriou, M. Soulie, and M. Coppola. On-chip networks for mixed-criticality systems. In *2016 IEEE 27th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 164–169, July 2016.

[Pau08] Christian Peter Paukovits. *The Time-Triggered System-on-Chip Architecture: Wien, Techn. Univ., Diss., 2009*. 2008.

[PBe12] J. Proenza, M. Barranco, and et al. The design of the CANbids architecture. In *Proc. of the IEEE Conf. on Emerging Technologies and Factory Automation*, 2012.

[PK07] Paul Parkinson and Larry Kinnan. Safety-critical software development for integrated modular avionics. 2007.

[Pow92] D. Powell. Failure mode assumptions and assumption coverage. In *Fault-Tolerant Computing, 1992. FTCS-22. Digest of Papers., Twenty-Second International Symposium on*, pages 386–395, July 1992.

[PP92] Shiv Prakash and Alice C. Parker. Sos: Synthesis of application-specific heterogeneous multiprocessor systems. *Journal of Parallel and Distributed Computing*, 16(4):338 – 351, 1992.

[Pri92] P.J. Prisaznuk. Integrated modular avionics. In *Aerospace and Electronics Conference, 1992. NAECON 1992., Proceedings of the IEEE 1992 National*, pages 39–45 vol.1, May 1992.

[RG05] M. Rosenblum and T. Garfinkel. Virtual machine monitors: current technology and future trends. *Computer*, 38(5):39–47, May 2005.

[RK03] P. Ratanchandani and R. Kravets. A hybrid approach to internet connectivity for mobile ad hoc networks. In *Wireless Communications and Networking, 2003. WCNC 2003. 2003 IEEE*, volume 3, pages 1522–1527 vol.3, March 2003.

[RR13] T. Rauber and G. Rünger. *Parallel Programming for Multicore and Cluster Systems*. Springer, 2nd edition, 2013.

[Ruf97] J. Rufino. Dual-media redundancy mechanisms for CAN. Technical Report CSTC RT-97-01, Centro de Sistemas Telemáticos e Computacionais do Instituto Superior Técnico, Lisboa, Portugal, January 1997.

[Rus99a] J. Rushby. Partitioning for avionics architectures: Requirements, mechanisms, and assurance. NASA Contractor Report CR-1999-209347, NASA Langley Research Center, June 1999.

[Rus99b] John Rushby. Partitioning for safety and security: Requirements, mechanisms, and assurance. NASA Contractor Report CR-1999-209347, NASA Langley Research Center, June 1999. Also to be issued by the FAA.

[Rus01] J. Rushby. Modular certification. Technical report, Computer Science Laboratory SRI International, 333 Ravenswood Avenue, Menlo Park, CA 94025, USA, September 2001.

[Rus07] J. Rushby. Just-in-time certification. In *Engineering Complex Computer Systems, 2007. 12th IEEE International Conference on*, pages 15–24, July 2007.

[RVA+98] J. Rufino, P. Veríssimo, G. Arroz, C. Almeida, and L. Rodrigues. Fault-tolerant broadcasts in CAN. In *Proc. of the 28th Int. Symposium on Fault-Tolerant Computing Systems*, pages 150–159, June 1998.

[s0911] AS-6802 – Time-Triggered Ethernet, 11 2011.

[SAE11] SAE - AS-2D Time Triggered Systems and Architecture Committee. AS-6802 – Time-Triggered Ethernet, 11 2011.

[SBct] W. Steiner and G. Bauer. Mixed-criticality networks for adaptive systems. In *Digital Avionics Systems Conference (DASC), 2010 IEEE/AIAA 29th*, pages 5.A.3–1–5.A.3–10, Oct.

[Se09] W. Steiner and et al. TTEthernet dataflow concept. In *Proceedings of the 2009 Eighth IEEE International Symposium on Network Computing and Applications*, NCA '09, pages 319–322, Washington, DC, USA, 2009. IEEE Computer Society.

[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 *Digital System Design (DSD), 2012 15th Euromicro Conference on*, pages 105–113, Sept 2012.

[SML10] J. Sahoo, S. Mohapatra, and R. Lath. Virtualization: A survey on concepts, taxonomy and associated security issues. In *Computer and Network Technology (ICCNT), 2010 Second International Conference on*, pages 222–226, April 2010.

[SP07] M. Short and M.J. Pont. Fault-tolerant time-triggered communication using can. *Industrial Informatics, IEEE Transactions on*, 3(2):131–142, May 2007.

[SS01] O. Sinnen and L. Sousa. Comparison of contention aware list scheduling heuristics for cluster computing. In *Parallel Processing Workshops, 2001. International Conference on*, pages 382–387, 2001.

[Ste06] K. Steinhammer. *Design of an FPGA-Based Time-Triggered Ethernet System*. PhD thesis, Technische Universität Wien, Austria, 2006.

[Ste10] W. Steiner. An evaluation of smt-based schedule synthesis for time-triggered multi-hop networks. In *Real-Time Systems Symposium (RTSS), 2010 IEEE 31st*, pages 375–384, Nov 2010.

[Ste11] W. Steiner. Synthesis of static communication schedules for mixed-criticality systems. In *Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW), 2011 14th IEEE International Symposium on*, pages 11–18, March 2011.

[stp04] IEEE standard for local and metropolitan area networks: Media Access Control (MAC) bridges. *IEEE Std 802.1D-2004 (Revision of IEEE Std 802.1D-1998)*, pages 1 –277, 9 2004.

[Sys10] Sysgo. *PikeOS Safe and Secure Virtualization*, 2010.

[SZZS08] Guangming Song, Yaoxin Zhou, Weijuan Zhang, and Aiguo Song. A multi-interface gateway architecture for home automation networks. *Consumer Electronics, IEEE Transactions on*, 54(3):1110–1113, August 2008.

[TSPS12] Domitian Tamas-Selicean, Paul Pop, and Wilfried Steiner. Synthesis of communication schedules for ttethernet-based mixed-criticality systems. In *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/-Software Codesign and System Synthesis*, CODES+ISSS '12, pages 473–482, New York, NY, USA, 2012. ACM.

[TSX00] Yajie Tian, N. Sannomiya, and Yuedong Xu. A tabu search with a new neighborhood search technique applied to flow shop scheduling problems. In *Decision and Control, 2000. Proceedings of the 39th IEEE Conference on*, volume 5, pages 4606–4611 vol.5, 2000.

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

[UNR+05] R. Uhlig, G. Neiger, D. Rodgers, A.L. Santoni, F.C.M. Martins, A.V. Anderson, S.M. Bennett, A. Kagi, F.H. Leung, and L. Smith. Intel virtualization technology. *Computer*, 38(5):48–56, May 2005.

[VD10] J.-P. Vasseur and A. Dunkels. *Interconnecting Smart Objects with IP: The Next Internet*. Morgan Kaufmann, 2010.

[WEK10] A. Wasicek, C. El Salloum, and H. Kopetz. A system-on-a-chip platform for mixed-criticality applications. In *Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 2010 13th IEEE International Symposium on*, pages 210–216, 2010.

[YKK+06] Geon Yoon, Dae Hyun Kwon, Soon Chang Kwon, Yong Oon Park, and Young Joon Lee. Ring topology-based redundancy ethernet for industrial network. In *SICE-ICASE, 2006. International Joint Conference*, pages 1404–1407, Oct 2006.

Acronym

TDMA	Time Division Multiple Access
PE	Processing Element
VL	Virtual Link
ECU	Electronic Control Unit
TTEthernet	Time-Triggered Ethernet
NoC	Network-on-a-Chip
VLID	Virtual Link IDentifier
BAG	Bandwidth Allocation Gap
OSI	Open Systems Interconnection
MINT	Minimum Inter-Arrival Times
VLAN	Virtual Local Area Network
AVB	Audio/Video Bridging
AUTOSAR	AUTomotive Open System ARchitecture
QoS	Quality of Service
AFDX	Avionics Full-Duplex Switched Ethernet
CAN	Controller Area Network
CIS	CAN Interface Subsystem
CRC	Cyclic Redundancy Check

CSMA/CA Carrier Sense Multiple Access Collision Avoidance

ADN Aircraft Data Network

MPSoC Multi-Processor-System-on-a-Chip

NMR N-Modular Redundancy

TMR Triple Modular Redundancy

TTNoC Time-Triggered Network-on-a-Chip

TTSoC Time-Triggered System-on-a-Chip

COTS Commercial-Off-The-Shelf

SAE Society of Automotive Engineers

CPS Cyber-Physical System

CT marker Critical Traffic marker

NI Network Interface

FCR Fault Containment Region

MU Management Unit

WCL Worst-Case Latency

FTU Fault-Tolerant Unit

TDM Time-Division Multiplexing

MILP Mixed Integer Linear Programming

DAG Directed Acyclic Graph

WCET Worst-Case Execution Time

SIL Safety Integrity Level

MAC Media Access Control

Selected Publications

1. M. Abuteir and R. Obermaisser. Simulation environment for time-triggered ethernet. In *2013 11th IEEE International Conference on Industrial Informatics (INDIN)*, pages 642–648, July 2013

2. Z. Owda, M. Abuteir, R. Obermaisser, and H. Dakheel. Predictable and reliable time triggered platform for ambient assisted living. In *2014 8th International Symposium on Medical Information and Communication Technology (ISMICT)*, April 2014

3. A. Khalifeh, R. Obermaisser, M. Abuteir, and D. Abou-Tair. Systems-of-systems framework for providing real-time patient monitoring and care. In *Proceedings of the 8th International Conference on Pervasive Computing Technologies for Healthcare*, ICST, Brussels, Belgium, Belgium, 2014

4. M. Abuteir and R. Obermaisser. Mixed-Criticality Systems Based on Time-Triggered Ethernet with Multiple Ring Topologies. In *IEEE International Symposium on Industrial Embedded Systems (SIES), 2014 9th*, 2014

5. R. Obermaisser, Z. Owda, M. Abuteir, H. Ahmadian, and D. Weber. End-to-end real-time communication in mixed-criticality systems based on networked multicore chips. In *Digital System Design (DSD), 2014 17th Euromicro Conference on*, Aug 2014

6. R. Obermaisser, M. Abuteir, A. Khalifeh, and D. Abou-Tair. Systems-of-systems framework for providing real-time patient monitoring and care: Challenges and solutions. In *Communications in Computer and Information Science*, volume 515. 2015

7. M. Abuteir and R. Obermaisser. Scheduling of rate-constrained and time-triggered traffic in multi-cluster ttethernet systems. In *Industrial Informatics (INDIN), 2015 13th IEEE International Conference on*, July 2015

8. Z. Owda, M. Abuteir, and R. Obermaisser. Co-simulation framework for networked multi-core chips with interleaving discrete event simulation tools. In *2015 IEEE 20th Conference on Emerging Technologies Factory Automation (ETFA)*, Sept 2015

9. M. Abuteir, R. Obermaisser, Z. Owda, and T. Moudouthe. Off-chip/on-chip gateway architecture for mixed-criticality systems based on networked multi-core chips. In *2015 IEEE 18th International Conference on Computational Science and Engineering*, Oct 2015

10. P. Petrakis, M. Abuteir, M. D. Grammatikakis, K. Papadimitriou, R. Obermaisser, Z. Owda, A. Papagrigoriou, M. Soulie, and M. Coppola. On-chip networks for mixed-criticality systems. In *2016 IEEE 27th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, pages 164–169, July 2016

11. Z. Owda, M. Urbina, R. Obermaisser, and M. Abuteir. Hierarchical transactional memory protocol for distributed mixed-criticality embedded systems. In *2016 IEEE 14th Intl Conf on Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing*, pages 334–343, Aug 2016

12. H. Ahmadian, R. Obermaisser, and M. Abuteir. Time-triggered and rate-constrained on-chip communication in mixed-criticality systems. In *2016 IEEE 10th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSOC)*, pages 117–124, Sept 2016