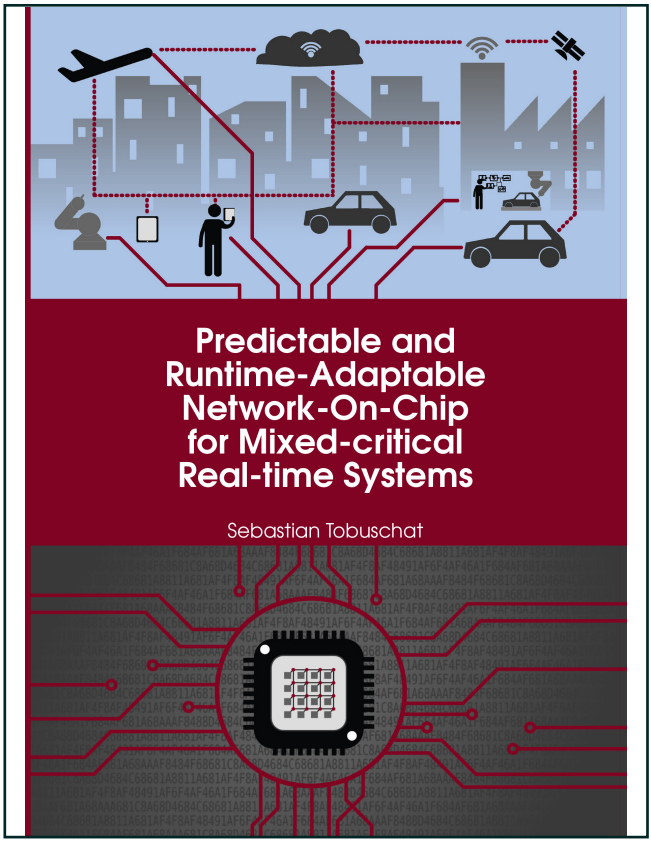
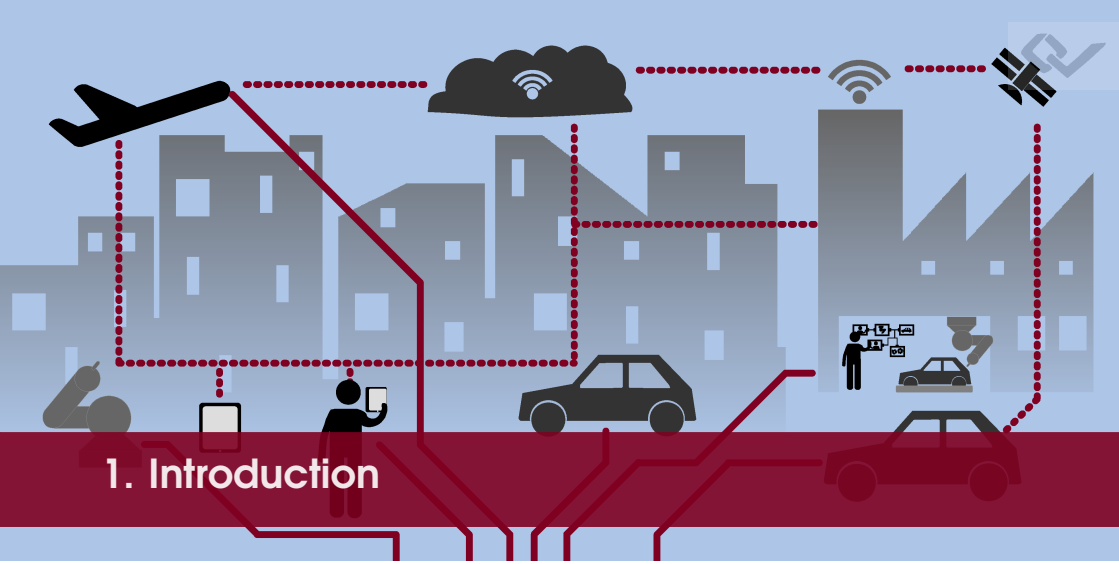
Sebastian Tobuschat (Autor)

Predictable and Runtime-Adaptable Network-On-Chip for Mixed-critical Real-time Systems



https://cuvillier.de/de/shop/publications/7995



1. Introduction

1.1 Motivation

Safety-critical and dependable embedded systems play an important role in our daily life. For example, modern vehicles integrate over 100 electronic control units (ECUs). Due to the ever-increasing demand for high performance together with low energy consumption and size, multicore systems, as known from general-purpose computing, are adopted by the safety-critical embedded market. The integration of multiple cores in a chip to a multiprocessor system on chip (MPSoC) offers the possibility to consolidate multiple functions or ECUs, which previously had been distributed and isolated by external buses. This consolidation of functions with different overall importance leads to mixed-criticality multicore systems [28]. In this context, a critical function is essential for the safety of the system. Therefore, this function is developed with high diligence and so the behaviour (e.g. timing) is well specified and tested. For non-critical functions the confidence in the characteristics is lower, e.g., the possibility that the function deviates from the specification is higher. Additionally, non-critical functions might be user provided and the risk of malicious functions trying to endanger system safety, e.g., through denial-of-service attacks, increases.

Figure 1.1 presents an example for typical features in a modern car. These include classical applications, such as engine control or entertainment functions, but also new complex functions for highly automated and autonomous driving, which all have different requirements. To provide all these features, a system must offer high performance and parallel processing, as well as

efficient communication and synchronization between different, possibly heterogeneous processing units. Figure 1.2 shows the functions of a vision based driver assistance system with the different processing needs of the functions. It consists of functions running well on classical CPUs, as e.g. the feedback loop or standard processing, which require complex computations but work on a small data set. But also of more advanced functions suited for processing on a DSP or GPU based system, as these need to process huge data sets but require less complex computations. Hence, heterogeneous and interconnected systems are needed to efficiently handle the workload.

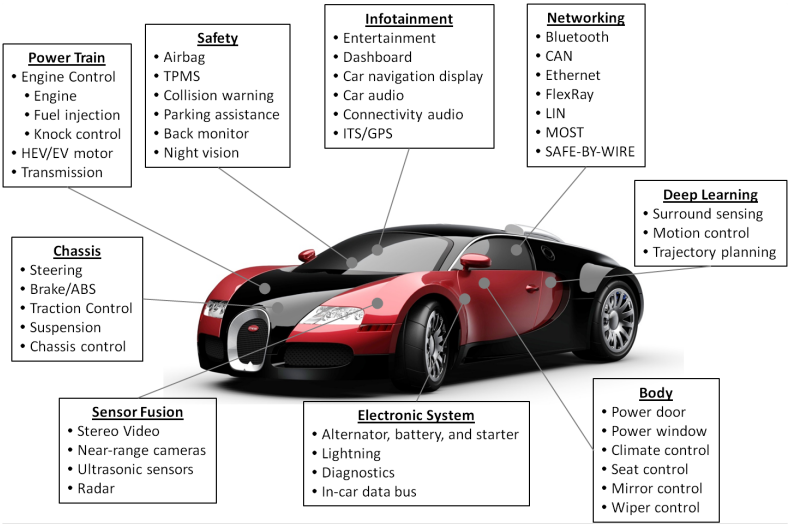


Figure 1.1: *Electronic features used pervasively in automobiles.*

To cope with the increasing complexity of interconnected functions and to reduce the cost and power consumption of a system, multicore systems are used to efficiently integrate different processing units in the same chip. This leads to a transition from many distributed (low performance) ECUs, which require massive wiring and have a high synchronization and communication overhead, over a domain centralized architecture to a software defined vehicle, as shown in Figure 1.3. In a domain centralized architecture or software defined vehicle, high-performance multicore ECUs are used to provide the functionalities, which were previously distributed. And while the domain centralized architecture tries to provide one ECU for each domain for domain-

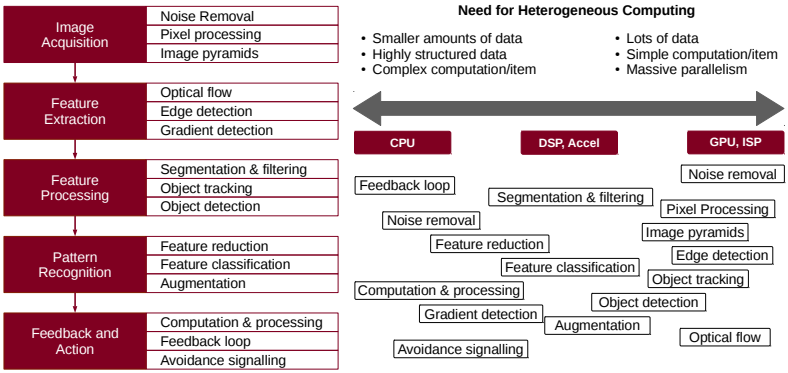


Figure 1.2: *Vehicle computing evolution (based on [1])*

common processing and domain isolation, the software defined approach processes the workload of different domains on the same ECU. Such an approach improves the synchronization and communication between the processing units and hence the performance. At the same time it reduces the isolation properties as functions of different domains with diverse safety requirements are now using the same MPSoCs and network connections, leading to mixed-criticality systems.

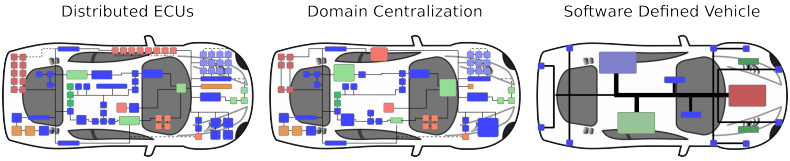


Figure 1.3: *Vehicle computing evolution.*

Especially with the upcoming autonomous driving, the correct functioning of the system must be guaranteed. With the transition of the responsibility from the human to the machine, there will be no driver supervising the decisions and actions of the system, cf. Figure 1.4. Hence, as sketched in Figure 1.5, there will be no driver overtaking in case of errors (as e.g. induced by interferences in mixed criticality systems) and the system must provide a technical fallback. Such technical fallback requires to prove the correct functioning under all cases.

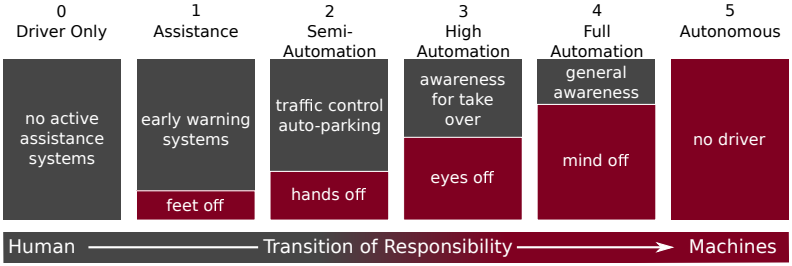


Figure 1.4: *Transition of responsibility (based on [1]).*

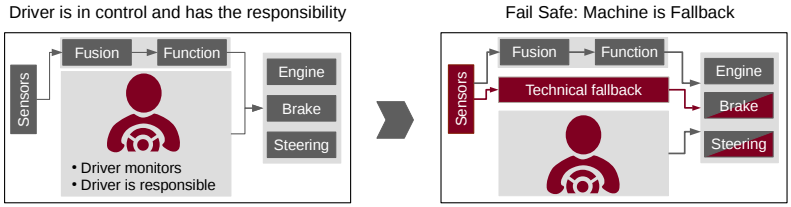


Figure 1.5: *Machine as fallback (based on [1]).*

Safety standards explicitly mention this problem in context of mixing different criticalities and, for example, require *sufficient independence* (IEC 61508-2, 2010 [15]):

> **7.4.2.3**: Where an E/E/PE safety-related system is to implement both safety and non-safety functions, then all the hardware and software shall be treated as safety-related unless it can be shown that the implementation of the safety and non-safety functions is sufficiently independent (i.e. that the failure of any non-safety-related functions does not cause a dangerous failure of the safety-related functions).

R Sufficient independence of implementation in a mixed-criticality system is established by proving that timing interference or the probability of a dependent failure between the non-safety and safety-related parts is sufficiently low in comparison with the highest safety integrity level associated with the safety functions [15]. While a failure can result from, for example, a fault, wilful timing attack, or wilful memory manipulation and influence timing, data consistency, or other parameters of the system.

Networks-on-chip (NoCs), as a scalable and modular interconnect, are used as a promising solution for MPSoCs, due to their performance, power, and size benefits [31]. In a NoC resources, such as the output ports of the routers, are shared among the different functions and safety classes [126; 216]. Hence, applications of different safety levels will inevitably compete with each other in a NoC for resources (cf. Section 4.1). This resource sharing couples the execution behaviour across cores and, thus, impacts non-functional properties like timing, which are of particular interest in safety-critical environments (as discussed above).

One approach to solve the problem of mixed-criticality is to develop all functions to the highest relevant safety level (cf. IEC 61508-2, 2010 [15]). This leads to higher development costs and lower system utilization. Another approach is to provide *sufficient independence* through quality-of-service (QoS) mechanisms. The challenging part of the latter approach is to efficiently utilize system resources while providing a bounded and feasible interference. This typically leads to a trade-off between providing real-time guarantees for certain applications and performance for the others, as well as the introduced overhead by the quality-of-service mechanisms.

1.2 Standards for Safety

The safety of the public is a major driver of the automotive, railway, industry automation and aviation industry. Based on Part 4 of the IEC 61508, safety can be defined as "freedom from unacceptable risk of physical injury or of damage to the health of people, either directly, or indirectly as a result of damage to property or to the environment" [15]. To ensure the safety of a system there are industrial and research efforts towards standardization of the safety life cycle for electronic products. To reach a safety level and certify or qualify a system, several standards and guidelines must be followed depending on the field of application. There are many national and international organizations, which publish design guidelines and regulations for different domains. Some of these are the International Standards Organization (ISO), the International Electrotechnical Commission (IEC), the Radio Technical Commission for Aeronautics (RTCA) and the Society of Automotive Engineers (SAE). Furthermore, there exist national restrictions by law. Several of the domain specific safety standards are based on the IEC 61508 as sketched in Figure 1.6. This chapter gives a summary of some safety standards relevant for the on-chip network architecture.

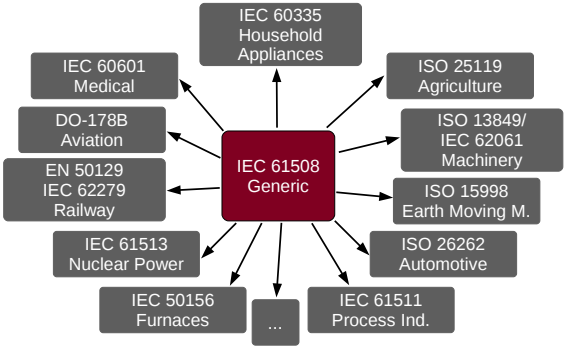


Figure 1.6: *IEC 61508 as the root of several safety standards.*

The IEC 61508 defines design and verification requirements to establish safety in systems that incorporate electronic/electrical components and their communication [15].

For the avionics, the DO-178B, DO-254, and DO-297 handle respectively the development and validation of software, hardware, and integrated modular avionics [2; 5; 6]. Together these include design considerations on system, hardware, and software level. The ARP-4754 and ARP-4761 describe methods and considerations to get through the certification process of a complex highly-integrated avionics system [13; 14].

Similarly, standards and approaches exist for the development of heavy machinery (IEC 62061), systems for process industries (IEC 61511), railway (IEC 62279), and power plants (IEC 61513).

For the automotive domain so far no necessity for certification exists. However, qualification approaches are used to ensure the correct functioning of the system. The ISO 26262, for example, states that "*If the embedded software has to implement software components of different ASILs, or safety-related and non-safety-related software components, then all of the embedded software shall be treated in accordance with the highest ASIL, unless the software components meet the criteria for coexistence in accordance with ISO 26262-9:2011, Clause 6.*", where Clause 6 proposes "*In the case of the coexistence of sub-elements that have different ASILs assigned or the coexistence of sub-elements that have no ASIL assigned with safety-related ones, it can be beneficial to avoid raising the ASIL for some of them to the ASIL of the element. When determining the ASIL of sub-elements of*

an element, the rationale for freedom from interference is supported by analyses of dependent failures focused on cascading failures". The freedom of interference is later defined as "*absence of cascading failures between two or more elements that could lead to the violation of a safety requirement*".

Besides the ISO 26262, other standards exists, which can influence the design of an automotive system. The ISO 15005 describes constraints to ensure the safe operation of a road vehicle while it is in motion. This concerns especially the interaction of the user and the vehicle's information and control system [3]. The ISO 16951 handles the prioritized presentation of messages and windows to the user by the vehicle's information and control system [4]. Hence, if such applications use components of an interconnected system, they can influence the design of the network architecture.

By applying these rules to a system-on-chip, the parts of the hardware and runtime environment (RTE), which are always used, must be certified to the highest relevant safety level. For all other components "sufficient independence" must be implemented. Therefore, NoCs, whenever used for communication between safety critical components such as automotive functions, are or will be, depending on the safety-critical domain, the subject of regulation through standards and certification procedures to ensure their correct functioning. In this context, not only the possibly high average performance and low costs play a critical role but also the ability to prove adherence to the safety requirements. This adds another complexity layer to the design process and requires traceability with respect to real-time properties, e.g., application of formal analysis methods such as Real-Time Calculus [213], Network Calculus [134], or Compositional Performance Analysis [99].

1.3 Real-Time Traffic Properties

Modern safety- or mixed-critical embedded systems host heterogeneous applications, with different requirements and behaviour (cf. Section 1.1). This includes applications with different safety-criticality as well as different real-time requirements. An example for the automotive domain is the integration of pedestrian detection in advanced driver assistance systems and entertainment applications. In this sense, criticality can be broken down into at least two orthogonal aspects as shown in Figure 1.7: *safety criticality* and *time criticality*.

For real-time (time-critical) applications the correctness of the system function depends not only on functional but also on temporal aspects. That

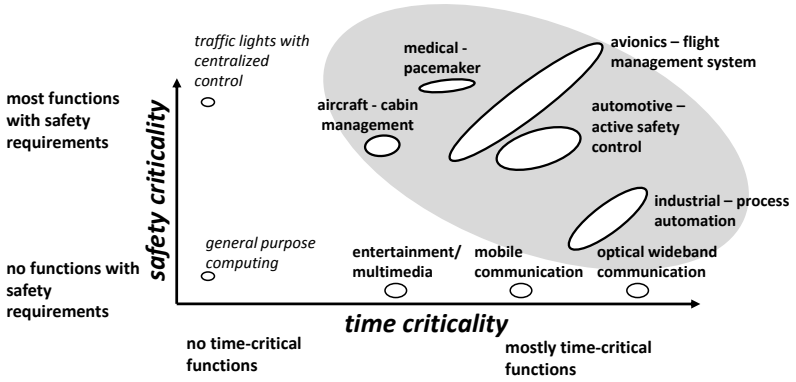


Figure 1.7: *Two dimensions of mixed-criticality [24].*

is, these applications should always finish computations before a given time or receive a certain minimum throughput to ensure correctness or safety. The time by which a specific result must be produced is called *deadline*. Typical examples for such applications without safety requirements are embedded mobile communication (e.g. UMTS or LTE) or entertainment applications.

For purely safety critical systems the integrity of computation needs to be preserved. An example are traffic lights that are controlled by a centralized controller. A failure might lead to catastrophic consequences, e.g., pedestrians or car passengers can be injured or killed. However, it is not important if the (correct) computation result is achieved within milliseconds or seconds. A non-switching traffic light is more acceptable as a wrong state.

Domains with both safety and real-time requirements are of special interest, as they cover many of the important future scenarios like advanced driver assistance system (ADAS) or autonomous driving (cf. Figure 1.7). In these domains, the failure of the function (e.g. violation of a real-time requirement) can have catastrophic effects. While this domain is already challenging, as techniques from real-time and dependability (e.g. predictability) need to be combined, the demands for lower energy consumption and higher efficiency of systems lead to the integration of different domains on the same MPSoC (cf. Section 1.1). In such mixed-critical systems, applications with safety and real-time requirements (e.g. engine control, ADAS) are running together with applications with purely real-time requirements (e.g. entertainment) or

no strict requirements at all. Hence, the system must efficiently combine *real-time*, *dependability*, and *high performance* mechanisms.

The timing properties can be further divided into *best-effort*, *throughput bound*, *soft real-time*, and *hard real-time* [71; 114; 142]. Figure 1.8 shows an example for possible timing requirements for some automotive applications.

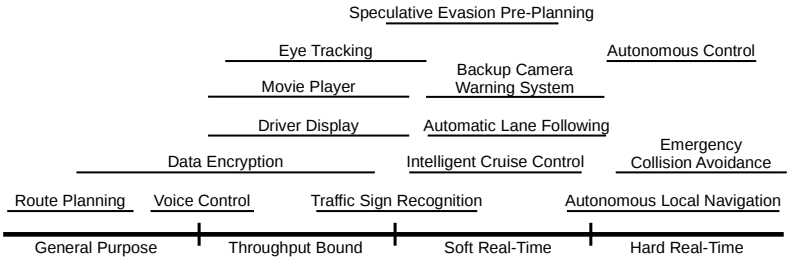


Figure 1.8: *Exemplary timing requirements for some automotive applications [71; 114].*

Hard real-time applications have firm deadlines, i.e., the utility of the produced result is zero or even renders the whole system as unfeasible when the deadline is crossed. For safety-critical real-time applications, any delay in their execution beyond their deadline, including high latencies of on-chip transmissions, may have severe consequences for the whole system, i.e., fatal failures and prohibitive degradation of service. Hence, these applications are usually not allowed to miss any deadline. That is, for example, the worst-case latency must stay below an assumed upper limit, which is derived from the deadline. For purely hard real-time applications, the functionality does not directly affect user safety but is, for example, important for the user experience. Hence, a deadline violation can cause client loss or substantial financial penalty. To achieve the needed performance for hard real-time applications, the system is typically dimensioned for the worst-case behaviour. Additionally, for safety-critical systems, the correctness of the behaviour, even in the worst-case, must be verified according to safety standards. Due to these requirements the characteristics of safety-critical hard real-time senders and their network traffic are usually well specified and tested and hence known at design time.

Soft real-time applications, on the other hand, may tolerate occasional deadline misses. The main difference, when compared to hard real-time senders, is that these applications are rarely required to rigorously meet all

their deadlines, i.e., the produced results have some utility after the deadline or are not affecting the safety (e.g. can safely be discarded) [142; 201]. An example are control algorithms based on a feedback loop or video analysis for night vision in a car. The algorithms in such systems may tolerate a limited number of cases when instead of new sampling data old values are used. Thus, it can compensate occasional deadline misses without any severe consequences for the system [193; 218; 239].

Similar to soft real-time, there are throughput bound applications, which must comply to overall real-time performance objectives in terms of a minimum achieved throughput over a given period. Again, these applications are rarely required to rigorously meet all their deadlines, i.e., the short term throughput can underrun the required throughput as long as the long-term throughput is acceptable. For instance, video streaming done as a part of an infotainment function in a car does not influence vehicle safety, but video frames must still arrive with a certain latency to prevent quality drops and glitches. Still, for a producer of an infotainment system the quality of user experience may play a critical role in the market success of a product. Consequently, a producer may accept sporadic drop of the video quality but may lose clients whenever it happens too often.

The last category of general purpose or best-effort (BE) does not have strict temporal requirements, e.g, a deadline miss will not endanger system safety. However, this typically also means that such application are less tested with respect to temporal properties and only designed targeting average performance metrics. For example, the frequency and size of accesses to the (on chip) network are not known. Still, the system should provide sufficient resources to process such applications (e.g. be work conserving), as they can, for example, be used to increase the long-term efficiency or user experience (e.g. diagnosis functions or route planning). Hence, achieving high performance and low latency is a common design goal, as long as all guarantees for safety- or time-critical applications can be delivered.

1.4 Requirements of Safety-critical Embedded Systems

As shown in the sections before, the communication of safety-related data must be protected at run-time against effects of faults, which may lead to failures of the system. These faults include transient faults, physical damage as well as lack of sufficient independence between tasks (e.g. timing interference or data corruption). For example, the ISO 26262 provides a list of faults, presented in Table 1.1 regarding the exchange of information, which

must be considered in case of an interconnect for certification purposes. An end-to-end protection defines a set of mechanisms, which avoid these faults or allow a reliable detection and appropriate countermeasures.

Some of these faults directly relate to real-time metrics for the on-chip interconnect, e.g., a *delay of information* or *blocking access to the communication channel*. Others relate to consistency and the protection of packets done directly in the interconnect. For example, without a proper flow control in the network, packets might be dropped or overwritten leading to corruption of information. Other faults, although not directly related to the temporal metrics, can influence the predictability indirectly. Transient errors, malfunctioning, or malicious senders, for example, can introduce uncertainty and dynamics to the system, e.g., sporadic overloads due to re-transmissions or babbling idiots. Such dynamics hinder predictability or even render it impossible.

These faults can be detected and partly avoided in the software layer of a system. For example, AUTOSAR provides several mechanisms to cope with these faults (cf. Table 1.2), as e.g., *CRC*, *Data ID*, *Counter*, *Regular transmission + timeout monitoring* [10; 77]. The *Data ID* is a unique identifier to verify the identity of each transmitted (safety-related) data element. The *Counter* is a simple counter that is incremented on every send request. It can be used to implement an *alive counter* and a *sequence counter*. For the sequence counter, the value is checked at receiver side for correct incrementation, while for the *alive counter* it is only checked whether is changes at all. Based on these mechanisms receiver communication and sender acknowledgement timeouts can be implemented. For this, a receiver is executed independently of the data transmission (e.g. periodic activation and checking for new data) and checks the validity of the received data (based on CRC, counter, and Data ID). With this, a wrong counter detects a duplication of previous data, loss of communication, or timeouts. Table 1.2 shows the fault coverage of different mechanisms. These mechanisms can be realized in software (cf. AUTOSAR E2E Protocol Specification [10]), hardware, or a hybrid solution. To increase the efficiency of a system, the hardware can provide support to detect and avoid such faults, e.g., hardware CRC checking or quality of service (QoS) mechanisms to limit interference. The latter are of special interest, as such mechanisms change the timing and behaviour of the system. Additionally, they can drastically degrade the performance, utilization, or adaptability of a system. For the other mechanisms (and faults), the influence on the system design and behaviour can be straightforwardly derived.

Table 1.1: *Summary of communication faults in the design of the automotive systems according to ISO 26262 [7; 10].*

Fault Type	**Description**
Repetition of information	A type of communication fault, where information is received more than once.
Loss of information	A type of communication fault, where information or parts of information are removed from a stream of transmitted information.
Delay of information	A type of communication fault, where information is received later than expected.
Insertion of information	A type of communication fault, where additional information is inserted into a stream of transmitted information.
Masquerading	A type of communication fault, where non-authentic information is accepted as authentic information by a receiver.
Incorrect addressing	A type of communication fault, where information is accepted from an incorrect sender or by an incorrect receiver.
Incorrect sequence of information	A type of communication fault, where information is accepted from an incorrect sender or by an incorrect receiver.
Corruption of information	A type of communication fault, which changes information.
Asymmetric information from sender to multiple receivers	A type of communication fault, where receivers do receive different information from the same sender.
Information from a sender received by only a subset of the receivers	A type of communication fault, where some receivers do not receive the information.
Blocking access to a communication channel	A type of communication fault, where the access to a communication channel is blocked.

Table 1.2: *Fault detection coverage for different mechanisms.*

	Counter	Data ID	CRC	Transmission on regular bases and timeout monitoring	QoS
Repetition of information	x	—	—	—	—
Loss of information	x	—	—	—	—
Delay of information	x	—	—	x	x
Insertion of information	x	x	x	—	—
Masquerading	—	x	x	—	—
Incorrect addressing	—	x	—	—	—
Incorrect sequence of information	x	—	—	—	—
Corruption of information	—	—	x	—	—
Asymmetric information from sender to multiple receivers	—	—	x	—	—
Information from a sender received by only a subset of the receivers	x	—	—	—	—
Blocking access to a communication channel	x	—	—	x	x

As discussed in Section 1.1, NoCs are foreseen as a communication backbone for large systems-on-chip (SoCs) integrating different ECUs. As a result of such integration, it can happen that different traffic classes, i.e., hard real-time tasks, soft-real time, throughput bound, and best-effort (BE), share the SoC resources. This causes co-dependencies between applications running on different cores, what may endanger safety. Unpredictable and bursty accesses from BE senders may lead to contention in network buffers. In on-chip interconnects without appropriate QoS mechanisms transmissions are scheduled as soon as they arrive, and all traffic receive equal treatment. This leads to the possibility of an unbounded timing interference, which may lead to missed deadlines for real-time traffic. Hence, NoCs for future safety-critical systems need to provide QoS mechanisms.

NoC architectures are judged by *performance* (e.g. latency, throughput, utilization), *cost* (e.g. design effort, HW overhead), *predictability* (e.g. formal analysis; guarantees on performance metrics), and *flexibility/adaptability* (e.g. adapt to system internal and external changing conditions; re-use for different use cases) [24; 54; 69]. The challenge in providing mechanisms for these is that they can be contradictory. For example, an arbitration based on a static time-division multiplexing (TDM) achieves a high predictability.

Under certain conditions, it can even achieve a fair performance, e.g., when the TDM slots can be fully utilized. However, the design is usually developed according to the worst-case behaviour, leading to unused slots during normal operation of the system. Hence, the design is not work-conserving leading to a low utilization and degradation of performance. Additionally, a TDM approach can typically not easily adapt to changing conditions. Summarizing, the most important requirements of a NoC architecture are:

- efficient support of different traffic types
  - sufficient independence (limited interference between applications)
  - guaranteed worst-case performance for real-time applications (predictability)
  - as good as possible actual-case performance for non-real-time applications (e.g. high utilization, work conserving, no "second class citizens")
- low cost
  - low design effort
  - low hardware overhead
  - reusability: allow the same architecture to be used for different usecases/domains
  - allow efficient verification (bounds on interference)
- flexibility
  - allow to adapt to system internal and external changing conditions
  - reusability: allow the same architecture to be used for different usecases/domains
  - safety and real-time "as a feature"

1.5 Research Objective and Contribution

For safety-critical systems, a major goal is the avoidance of hazards. For this, safety-critical systems are qualified or even certified to prove the correct functionality under all possible cases. A predictable behaviour can help to ease the qualification process (e.g. analysis) of the system. Thus, achieving a predictable behaviour is an important goal for these systems. For the interconnect (e.g. the NoC) design, this means that providing predictable resource sharing between concurrent transmissions is a major driver. However, the support for temporal properties should not cancel out benefits resulting from the application of NoCs, e.g., high efficiency, scalability, flexibility, and low production costs. Hence, these other design goals shall also be reached, to reduce cost, increase efficiency, and market competition.