ERROR CONTAINMENT IN THE TIME-TRIGGERED SYSTEM-ON-A-CHIP ARCHITECTURE

R. Obermaisser, H. Kopetz, C. El Salloum, B. Huber
Vienna University of Technology, Austria

Abstract: The time-triggered System-on-a-Chip (SoC) architecture provides a generic multi-core system platform for a family of composable and dependable giga-scale SoCs. It supports the integration of multiple application subsystems of different criticality levels within a single hardware platform. A pivotal property of the architecture is the integrated error containment, which facilitates modular certification, robustness, and composability. By dividing the complete SoC into physically separated components that interact exclusively by the timely exchange of messages on a time-triggered Network-on-a-Chip (NoC), we achieve error containment for both computational and communication resources. The time-triggered design allows protecting the access to the NoC with guardians that are associated with each component. Based on the protection of the time-triggered NoC with inherent predictability and determinism, the architecture also enables error containment for faulty computational results. These value message failures can be masked using active redundancy (e.g., off-chip and on-chip Triple Modular Redundancy (TMR)) or detected using diagnostic assertions on messages. The design of the error containment mechanisms systematically follows a categorization of significant fault classes that an SoC is subject to (e.g., physical/design, transient/permanent). Evidence for the effectiveness of the error containment mechanisms is available through experimental data from a prototype implementation.

1. INTRODUCTION

Many large embedded control systems can be decomposed into a number of nearly independent *Distributed Application Subsystems (DASes).* In the automotive domain, the *power train control system*, the *airbag control system*, the *comfort electronics control system* and the *multimedia system* are

examples for DASes. A similar decomposition is performed in control system design aboard an airplane. Different DASes can be of *differing criticality level* and are often developed by *different organizations*. At a high level of abstraction—at the *Platform Independent Model (PIM)* level—a DAS can be described by a set of processing *Jobs* that exchange *messages* in order to achieve its stated objective. In order to eliminate any *error propagation path* from one DAS to another DAS and to reduce the overall system complexity, each DAS is often implemented on its own dedicated hardware base, i.e., a computer is assigned to each job of a DAS and a shared physical communication channel (in the automotive domain a *CAN* network (Bosch 1991)) is provided for the exchange of the messages within a DAS. In case of a failure of a DAS function it is then straightforward to identify the organization responsible for the malfunction, even if it is not clear whether the failure is caused by a transient hardware fault or a software error. We call such an architecture, where each DAS has its own dedicated hardware base, a *federated architecture.* In the automotive domain the massive deployment of federated architectures has led to a large number of Electronic Control Units (ECUs, i.e., nodes) and networks aboard a car. In a typical premium car more than fifty ECUs and five different networks can be found (Leohold 2005). This large number of ECUs and networks has some negative consequences: the high number of cabling contact points (which are a significant cause of failures) and the high costs. These negative consequences could be eliminated if one ECU could host more than one job of a DAS and thus the number of ECUs, networks and cables is significantly reduced. We call such an architecture, where a single integrated hardware base for the execution of different DASes is provided, an *integrated architecture*. Hammett R. describes aptly the technical challenge in the design of an integrated architecture: *The ideal future avionics systems would combine the complexity management advantages of the federated approach, but would also realize the functional integration and hardware efficiency benefits of an integrated system (Hammett 2003).*

In the recent past, a number of efforts have been made to develop an integrated architecture e.g., *Integrated Modular Avionics (IMA) (Wilkinson 2005)* in the aerospace domain, AUTOSAR (Heinecke et al. 2004) in the automotive domain, and DECOS (Obermaisser et al. 2006) as a cross-domain architecture. The key idea in these approaches is the provision of a partitioned operating system for a computer with a single powerful CPU. This operating system is intended to provide in each partition an encapsulated execution environment for a single job and eliminate any error propagation path from one job to another job. However, the required encapsulation, particularly w.r.t. to temporal properties and transient failures is difficult to achieve in such an architecture.

This paper follows another route. The recent appearance *of multi-core Systems-on-a-Chip (SoCs)* (e.g., the *Cell multiprocessor (Kahle, Day et al. 2005)),* makes it possible to achieve the desired integration by assigning each job to a core of an SoC and by providing a time-triggered on-chip interconnect that supports composability and error containment between DASes. This paper focuses on a key property of this architecture, the *error containment between DASes*. The paper is structured as follows: In Section two we present an overview of the time-triggered SoC architecture. Section three is devoted to the issues of error containment with respect to design faults. Section four deals with error containment with respect to physical faults. Section five discusses implementation aspects and an experimental evaluation of the architecture. The paper terminates with a conclusion in Section six.

2. TIME-TRIGGERED SOC ARCHITECTURE

The central element of the presented SoC architecture is a *time-triggered NoC* that interconnects multiple, possibly heterogeneous IP blocks called *micro components* (see Figure 1), each one hosting a job of a DAS. The SoC introduces a *trusted subsystem*, which ensures that a fault (e.g., a software fault) within the host of a micro component cannot lead to a violation of the micro component's temporal interface specification in a way that the communication between other micro components would be disrupted. Therefore, the trusted subsystem prevents a faulty micro component from sending messages during the sending slots of any other micro component.

Another focus of the SoC architecture is integrated support for maintenance. The *diagnostic unit* is an architectural element that executes assertions on the messages sent by the micro components and stores diagnostic information in persistent storage for a later analysis.

Furthermore, the time-triggered SoC architecture supports dynamic integrated resource management. For this purpose, a dedicated architectural element called the *Trusted Network Authority (TNA)* accepts run-time resource allocation requests from the micro components and reconfigures the SoC, e.g., by dynamically updating the time-triggered communication schedule of the NoC and switching between power modes of micro components.

2.1 Micro Component

The introduced SoC can host jobs of multiple DASes (possibly of different criticality levels), each of which provides a part of the service of

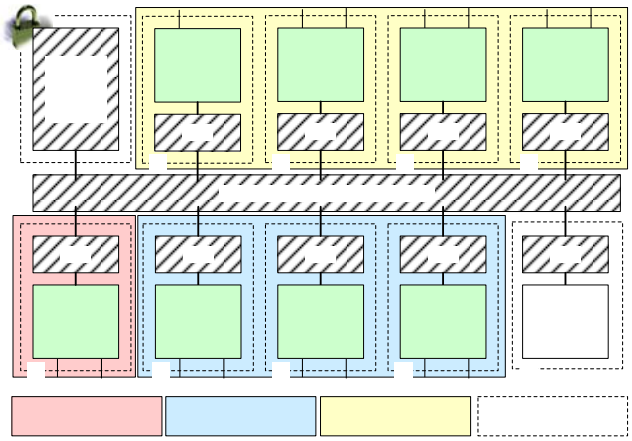


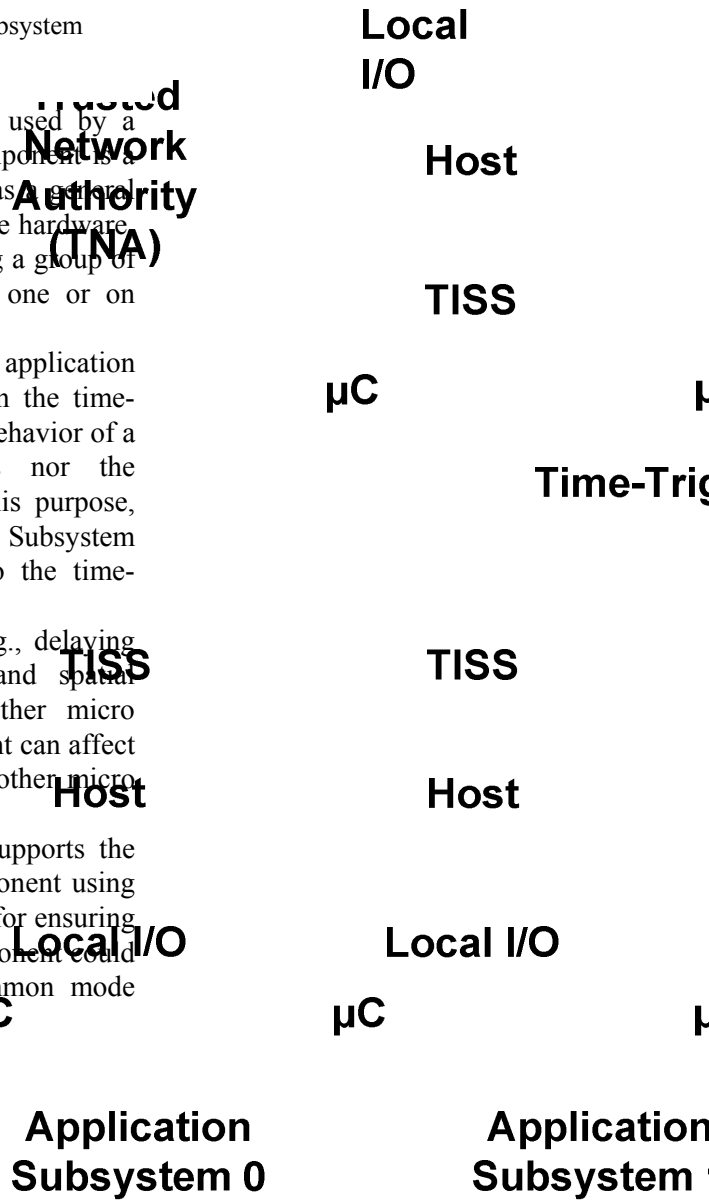
Figure 1: Structure of Time-Triggered SoC Architecture: trusted subsystem (shaded) and non trusted subsystem (hosts of micro components)

the overall system. A nearly autonomous IP-block, which is used by a particular DAS is denoted as a micro component. A micro component is a self-contained computing element, e.g., it can be implemented as a general purpose processor with software, and FPGA or as special purpose hardware. A DAS can be realized on a single micro component or by using a group of possibly heterogeneous micro components (either located on one or on multiple interconnected SoCs).

The interaction between the micro components of an application subsystem occurs solely through the exchange of messages on the time-triggered NoC. Each micro component is encapsulated, i.e., the behavior of a micro component can neither disrupt the computations nor the communication performed by other micro components. For this purpose, each micro component contains a so-called Trusted Interface Subsystem (TISS), which guards the access of the micro component to the time-triggered NoC (see also Section 3).

Encapsulation prevents *by design* temporal interference (e.g., delaying messages or computations in another micro component) and spatial interference (e.g., overwriting a message produced by another micro component). The only manner, in which a faulty micro component can affect other micro components, is by providing faulty input value to other micro components of the application subsystem via the sent messages.

Due to the provided encapsulation, the SoC architecture supports the detection and masking of such a value failure of a micro component using Triple Modular Redundancy (TMR). Encapsulation is necessary for ensuring the independence of the replicas. Otherwise, a faulty micro component could disrupt the communication of the replicas, thus causing common mode failures.



Encapsulation is also a key mechanism for the *correctness-by-construction* of application subsystems on an SoC. The SoC architecture ensures that upon the incremental integration of micro components, the prior services of the already existing micro components are not invalidated by the new micro components. This property, which is denoted as *composability* (Kopetz and Obermaisser 2002), is required for the seamless integration of independently developed DASes and micro components.

Also, encapsulation is of particular importance for the implementation of SoCs encompassing DASes of different criticality levels. Consider for example a future automotive system, which will incorporate DASes ranging from a safety-critical drive-by-wire DAS to a non safety-critical comfort DAS. In such a mixed criticality system, a failure of micro components of a non safety-critical application subsystem must not cause the failure of application subsystems of higher criticality.

2.2 Time-Triggered Network-on-a-Chip

The time-triggered NoC interconnects the micro components of an SoC. The purposes of the time-triggered NoC encompass clock synchronization for the establishment of a global time base, as well as the predictable transport of periodic and sporadic messages.

Clock Synchronization: The time-triggered NoC performs clock synchronization in order to provide a global time base for all micro components despite the existence of multiple clock domains. The time-triggered NoC is based on a uniform time format for all configurations, which has been standardized by the OMG in the smart transducer interface standard (OMG 2002).

Predictable Transport of Messages: Using Time-Division Multiple Access (TDMA), the available bandwidth of the NoC is divided into periodic conflict-free sending slots. We distinguish between two utilizations of a periodic time-triggered sending slot by a micro component. A sending slot can be used either for the periodic or the sporadic transmission of messages. In the latter case, a message is only sent if the sender must transmit a new event to the receiver. If no event occurs at the sender, no message is sent and thus no energy is consumed.

2.3 Diagnostic Unit

The diagnostic unit helps maintenance engineers in choosing appropriate maintenance actions. A maintenance action is either an update of the software in the host of a micro component to eliminate a design fault, or the replacement of a SoC component that is subject permanent physical faults.

For this purpose, the diagnostic unit collects error indications from micro components and also performs error detection itself. For each detected error, an entry is inserted into an error log for a later analysis.

Attribute	***Interface Feature Check***
Valid	A message is valid if it meets CRC, range and logical checks
Checked	A message is checked if it passes the output assertion
Permitted	A message is permitted w.r.t. a receiver, if it passes the input assertion of that receiver
Timely	A message is timely if it is in agreement with its temporal specifications
Correct	A message is correct if its value and temporal specifications are met
Insidious	A message is insidious if it is permitted but incorrect (requires a global judgment)

Table 1: DSoS Message Classification

The error detection is based on the message classification defined in the Dependable Systems-of-Systems (DSoS) conceptual model (Jones, Kopetz et al. 2001). In the DSoS model (see *Table 1*), a message is classified as *checked,* if it passes the *output assertion*. In addition, each message has to pass an *input assertion* in order to be processed by the receiving micro component. Once a message passes the input check it is called *permitted*. The input and output assertions consist of syntactic, temporal, and semantic checks. A message is called *valid,* if it passes the syntactic check. The CRC ensures that the content is in agreement with the checksum. We term a message *timely* if it is in agreement with the temporal specification. The *value-correctness* of a message can only be fully judged by an omniscient observer. However, application-specific plausibility checks can be applied. Note, that this implies the possibility that a message is judged as being *permitted* and therefore passing the input assertion but classified as incorrect by the omniscient observer. Such a message is called *insidious*.

In order to perform this message classification, the following three types of checks are executed at the SoC:

1. ***Output assertions***: The output assertions are computed at the diagnostic unit. The diagnostic unit observes all messages that are exchanged on the time-triggered NoC and executes predicates encoding a priori knowledge w.r.t. to the message syntax and semantics.
2. ***Temporal checks***: For sporadic communication, the TISS detects overflows of message queues. These checks apply both to queues with received messages (i.e., input ports), as well as to queues with messages that shall be sent (i.e., output ports).

3. ***Input assertions***: Input assertions are computed by micro components that receive messages from other micro components. Input assertions test the message syntax and the message semantics.

Errors detected via these three mechanisms are stored in the error log. While the results of (1) are already available at the diagnostic unit, the checks in (2) and (3) employ error indication messages on a diagnostic communication channel. The diagnostic communication channel of a micro component is realized as a statically reserved slot of the time-triggered NoC.

3. ERROR CONTAINMENT FOR DESIGN FAULTS

In his classical paper (Gray 1986), Jim Gray proposed to distinguish between two types of software design errors, Bohrbugs and Heisenbugs. Bohrbugs are design errors in the software that cause reproducible failures. Heisenbugs are design errors in the software that seem to generate quasi-random failures. From a phenomenological point of view, a transient failure that is caused by a Heisenbug cannot be distinguished from a failure caused by transient hardware malfunction.

In the SoC architecture, an error that is caused by a design fault in the sending micro component could only propagate to another micro component via a message failure, i.e., a sent message that deviates from the specification. In general, a message failure can be a *message value failure* or a *message timing failure (Cristian and Aghili 1985)*. A message value failure implies that a message is either invalid or that the data structure contained in a valid message is incorrect. A *message timing failure* implies that the *message send instant* or the *message receive instant* are not in agreement with the specification.

The propagation of an error due to a message timing failure is prevented by design in the SoC architecture. The TISS acts as a guardian for a micro component and prevents that a micro component can violate its temporal specification. In contrast, the value failure detection is in the responsibility of the receiving micro components and the diagnostic unit of the SoC architecture.

In order to avoid the propagation of an error caused by a message failure we need error detection mechanisms that are in different Fault Containment Regions (FCRs) than the message sender. Otherwise, the error detection mechanism may be impacted by the same fault that caused the message failure.

3.1 Fault Containment Regions for Design Faults

The FCRs of the SoC architecture with respect to design faults are as follows:

Trusted Subsystem: We assume that the trusted subsystem is free of design faults. In order to justify this strong assumption, the design of the trusted subsystem must be made as small and simple as possible in order to facilitate formal analysis. The time-triggered design of the trusted subsystem reduces the probability of Heisenbugs (e.g., no race conditions, clear separation of logical and temporal control). The control signals are derived from the progression of the sparse global time base (Kopetz 1992), which guarantees that all TISSes will visit the same state within a silence interval of the sparse time base.

Micro Components: The non safety-critical software of a micro component is encapsulated by the trusted subsystem such that even a malicious fault in the non safety-critical software of the host will not affect the correct function of the safety-critical software in the hosts of other micro components.

3.2 Timing Failure Containment

For the purpose of error containment w.r.t. message timing failures, each micro component comprises two parts: a host and a Trusted Interface Subsystem (TISS). The host implements the application services. Using the TISS, the time-triggered SoC architecture provides a dedicated architectural element that protects the access to the time-triggered NoC. Each TISS contains a table which stores a priori knowledge concerning the global points in time of all message receptions and transmissions of the respective micro component. Since the table cannot be modified by the host, a design fault restricted to the host of a micro component cannot affect the exchange of messages by other micro components.

3.3 Value Failure Containment

Since value failures are highly application-specific, their detection in the SoC architecture relies on the host. An important baseline for these detection mechanisms is the timing failure containment of the time-triggered NoC. The SoC architecture ensures that a received message truly stems from the associated sender (i.e. masquerading protection) and that the message is temporally correct.

It is the responsibility of each micro component, to decide whether the actual message should be accepted or not. Therefore, it can run tests

exploiting application-specific knowledge concerning the values of the signals (variables) that are transported within the message. Such tests are usually called executable assertions (Hecht 1976; Saib 1987). Executable assertions are statements, which can be made about the signals within a message. They are executed whenever a message is received in order to see if they hold. If not, the message is regarded as incorrect and will be discarded.

In the SoC architecture we support various types of assertions. For example executable assertions with signal range checks and slew rate checks can be applied to check whether a value is within its physical limits (e.g., the speed of a car cannot change within one second from zero to 200 kilometers per hour). The combination of signal range checks and slew rate checks into a single assertion offers the possibility to define bounds for extreme values in dependency on the actual rate of change.

A more elaborated technique is "model-based fault detection" (Isermann, Schwarz et al. 2002), where a process model of the controlled system is used to detect errors by relating two or more system variables to each other.

4. ERROR CONTAINMENT FOR PHYSICAL FAULTS

Depending on the persistence of a fault, one can classify physical faults into permanent faults and transient faults (Avizienis, Laprie et al. 2001). A permanent fault occurs, if the hardware of an FCR brakes down permanently. An example for a permanent fault is a stuck-at-zero fault of a memory cell. A transient fault is caused by some random event. An example for a transient fault is a Single Event Upset (SEU) caused by a radioactive particle (Normand 1996).

4.1 Fault-Containment Region for Physical Faults

In non safety-critical systems, we distinguish the same types of FCRs on the SoC architecture for physical faults as for design faults, namely the trusted subsystem and individual micro components. For a physical fault that affects a micro component, the SoC is capable of recovering from its induced errors. The performed recovery strategy depends on the persistence of the fault.

A permanent fault affecting a micro component requires reallocating the functionality of the affected micro component to a correct one. If the SoC contains multiple micro components of identical hardware, where the functional differentiation is provided by application software, the allocation of application subsystems to micro components can be dynamically

reconfigured by the TNA. Relaxing the requirement for 100% correctness for devices and interconnections may dramatically reduce the cost of manufacturing, verification and test of SoCs. However, a permanent fault affecting the trusted subsystem is not tolerated by the SoC architecture. For transient faults, on the other hand, a restart of the micro component can be triggered.

Contrary to non safety-critical systems, for safety-critical systems we have to consider the entire SoC as a single FCR. Due to the shared power supply, physical components (e.g. housing), and the proximity we cannot assume with a probability demanded for, e.g., ultra dependable systems that a physical fault like a SEU affects only a single micro component while sparing the trusted subsystem of the SoC. Therefore, masking of physical faults for safety-critical systems has to be performed at cluster level, where multiple SoC are interconnected to a distributed system.

4.2 Non Safety-Critical Systems

The masking of timing failures occurs in the same manner as for design faults. In case the host of a micro component is affected by a physical fault, the trusted subsystem ensures that there is no interference with the timely exchange of messages by other micro components.

Value failure detection is not in the responsibility of the SoC architecture, but in the responsibility of the micro components. For example, in non safety-critical systems value failure detection and correction can be performed in a single step by on-chip triple modular redundancy (TMR). In contrast to design faults, the masking of value failures resulting from physical faults can be performed in a systematic application-independent manner through active redundancy such as TMR. In this case, three replicated deterministic micro components perform the same operations in their host. They produce – in the fault-free case – correct messages with the same content that are sent to three replicated receivers that perform a majority vote on these three messages.

Value failure detection and timing failure detection are not independent. In order to implement a TMR structure for value failure masking at the application level in the micro components, the integrity of the timing of the architecture must be assumed. An intact sparse global time-base is a prerequisite for the system-wide definition of the distributed state, which again is a prerequisite for masking value failures by voting.

4.3 Safety-Critical Systems

In ultra-dependable systems, a maximum failure rate of 10^{-9} critical failures per hour is demanded (Suri, Walter et al. 1995). Today's technology does not support the manufacturing of chips with failure rates low enough to meet these reliability requirements. Since component failure rates are usually in the order of 10^{-5} to 10^{-6} (e.g., (Pauli et al. 1998) uses a large statistical basis and reports 100 to 500 failures out of 1 Million ECUs in 10 years), 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-tolerant strategies that enable the continued operation of the system in the presence of component failures. Therefore, it is demanded to realize a distributed system based on the SoC architecture. For this purpose, the SoC architecture supports gateways for accessing chip-external networks (e.g., TTP (Kopetz and Grünsteidl 1994), TTE (Kopetz, Ademaj et al. 2005)).

By the use of a time-triggered protocol (e.g. TTP or TTE) for the chip external network, timing failure containment can be realized by a guardian, which judges a message as untimely based on a priori knowledge of the send and receive instants of all messages of a single SoC.

As for non-safety-critical systems, value failure containment can be implemented by the use of TMR. However, for safety-critical systems the three replicated micro components have to reside on different micro components.

5. IMPLEMENTATION

Using a distributed system, the prototype implementation emulates an SoC with four micro components, as well as the TNA. A Time-Triggered Ethernet (TTE) network emulates the NoC and supports the time-triggered exchange of periodic and sporadic messages.

5.1 Implementation of Micro Components

Each micro component consists of two single board computers, namely one implementing the host and the second one implementing the TISS. The single board computer of the host is a Soekris Engineering net4521, which incorporates a 133Mhz 468 class ElanSC520 processor from AMD, 64 MBytes of RAM, and two 100 Mbit Ethernet ports. The implementation of the TISS is based on a single board compact computer of type Soekris Engineering net4801, which incorporates a 266 MHz 586 class NSC SC1100

processor. As an operating system, both single board computers use the realtime Linux variant Real-Time Application Interface (RTAI) (Beal, E.Bianchi et al. 2000).

In our prototype implementation we emulate the time-triggered NoC with TTE (Kopetz, Ademaj et al. 2005). For this purpose, a hardware implementation of the TTE controller is employed which is based on a PCMCIA FPGA card.

Dynamic reconfiguration of the NoC occurs via configuration messages sent by the TNA to the TISSes. The reconfiguration is triggered by the TTE controller via a dedicated reconfiguration interrupt at a predefined periodic instant with reference to the global time. Thus the reconfiguration is consistently performed at all TISSs at the same tick of the global time. At the reconfiguration instant, the TTE controller is switched to inactive and the MEDL is updated. The controller is reactivated after the update has finished.

The prototype implementation implements the memory interface between host and TISS via a standard Ethernet connection. On top of an optimized version of the standard real-time driver as distributed by the RTnet open-source project (Kiszka, Wagner et al. 2005) we have implemented a request/reply protocol through which the host can access the CNI and the local configuration parameters of the TISS.

5.2 Experimental Evaluation of Encapsulation

As part of the prototype implementation, we have also performed an early experimental evaluation for validating the encapsulation mechanisms provided by the trusted subsystem. The goal of the experiments is to test the spatial and temporal partitioning in the presence of an arbitrary behavior of faulty hosts. Therefore, the prototype setup contains a micro component with a host performing fault injection, as well as a monitoring device realized by a standard PC. As part of the experiments, the faulty host has systematically iterated through all possible message header values of a request message destined to the TISS.

The monitoring device has been connected to the TTE switch and has used its Ethernet interface in promiscuous mode. The tool WireShark has been used to observe and log the traffic of the entire TTE network. The analysis of the collected logs has yielded the following results:

- ***No discontinuities.*** In the experiments, all sent messages have included sequence numbers. The logs have indicated that no messages have been lost due to the behavior of the faulty lost.
- ***No additional messages.*** Other than the messages configured by the TNA, no messages have been observed on the TTE network.

• ***No effect on temporal properties.*** The temporal properties (i.e., bandwidth, latency, message order) of the TTE network with fault injection by a host have been identical to the temporal properties without fault injection.

In extension to these encouraging results, a more comprehensive validation using fault injection experiments is required as part of future work, e.g., focusing on physical fault injection with radioactive particles.

6. CONCLUSION

The advances of the semiconductor industry are leading to the appearance of billion transistors SoCs, where multiple computers – called *cores* – can be implemented in a single die (Kahle, Day et al. 2005). These developments are opening new alternatives for the design of an integrated architecture for embedded control systems. By assigning each job of a distributed application subsystem (DAS) to a core of a multi-core SoC, the *direct interference* between jobs of different DASes is eliminated without the introduction of a complex partitioned operating system. The *indirect interference* among DASes is eliminated by the provision of a dynamic time-triggered NoC that is controlled by a trusted network authority. This paper has focussed on the error containment properties of the integrated TT-SoC architecture, which in our opinion is superior to the error containment that can be achieved in federated architectures where ECUs are interconnected by a CAN network. The paper contains experimental data on a prototype implementation of the architecture that support our claims.

ACKNOWLEDGEMENTS

This work has been supported in part by the European IST project ARTIST2 under project No. IST-004527 and the European IST project DECOS under project No. IST-511764.

REFERENCES

Avizienis, A., J. Laprie, et al. (2001). Fundamental concepts of dependability.

Beal, D., E.Bianchi, et al. (2000). "RTAI: Real-Time Application Interface." Linux Journal.

Bosch (1991). CAN Specification, Version 2.0. Stuttgart, Germany, Robert Bosch Gmbh.

Cristian, F. and H. Aghili (1985). Atomic Broadcast: From simple message diffusion to Byzantine agreement. Proc. 15th IEEE Int. Symp. on Fault-Tolerant Computing (FTCS-15).

Gray, J. (1986). Why do Computers Stop and What can be done about it? Proc. of the 5th Symp. on Reliability in Distributed Software and Database Systems. Los Angeles, CA, USA.

Hammett, R. (2003). "Flight-critical distributed systems: design considerations [avionics]." IEEEAerospace and Electronic Systems Magazine **18**(6): 30–36.

Hecht, H. (1976). "Fault-Tolerant Software." ACM Computing Survey **8**(4): 391-407.

Heinecke, H., et al. (2004). AUTomotive Open System ARchitecture - An Industry-Wide Initiative to Manage the Complexity of Emerging Automotive E/E-Architectures. Proc. of the Convergence Int. Congress & Exposition On Transportation Electronics.

Isermann, R., R. Schwarz, et al. (2002). "Fault-Tolerant Drive-by-Wire Systems." Control Systems Magazine **22**: 64-81.

Jones, C. H., H. Kopetz, et al. (2001). Revised Conceptual Model of DSOS, University of Newcastle upon Tyne, Computer Science Department.

Kahle, J. A., M. N. Day, et al. (2005). "Introduction to the Cell multiprocessor." IBM Journal of Research and Development **49**(4/5): 589–604.

Kiszka, J., B. Wagner, et al. (2005). RTnet – a flexible hard real-time networking framework. Proc. of 10th IEEE Int. Conference on Emerging Technologies and Factory Automation.

Kopetz, H. (1992). Sparse time versus dense time in distributed real-time systems. Proc. of 12th Int. Conference on Distributed Computing Systems. Japan.

Kopetz, H., A. Ademaj, et al. (2005). The Time-Triggered Ethernet (TTE) design. Proc. of 8th IEEE Int. Symposium on Object-oriented Real-time distributed Computing (ISORC).

Kopetz, H. and G. Grünsteidl (1994). "TTP– a protocol for fault-tolerant real-timesystems." Computer **27**(1): 14–23.

Kopetz, H. and R. Obermaisser (2002). "Temporal composability." Computing & Control Engineering Journal **13**: 156–162.

Lamport, L. and R. Shostak (1982). "The Byzantine Generals Problem." ACM Trans. on Programming Languages and Systems **4**(3): 382-401.

Leohold, J. (2005). Automotive Systems Architecture. Architectural Paradigms for Dependable Embedded Systems. Vienna, Austria**:** 545-592.

Normand, E. (1996). "Single Event Upset at Ground Level." IEEE Trans. on Nucl. Science **43**: 2742.

Obermaisser, R., et al. (2006). "DECOS: An Integrated Time-Triggered Architecture." journal of the Austrian professional institution for electrical and information engineering **3**: 83-95.

OMG (2002). Smart Transducers Interface Specification, Object Management Group.

Pauli, B., et al. (1998). "Reliability of electronic components and control units in motor vehicle applications." VDI Berichte 1415, Electronic Systems for Vehicles: 1009–1024.

Saib, S. H. (1987). Executable Assertions - An Aid to Reliable Software. Proc. of Asilomar Conference on Circuits Systems and Computers.

Suri, N., C. J. Walter, et al. (1995). Advances in Ultra-Dependable Distributed Systems, Chapter 1, IEEE Computer Soc. Press.

Wilkinson, C. (2005). "IMA aircraft improvements." Aerospace and Electronic Systems Magazine, IEEE **20**(9): 11-17.