# Integrating Safety and Multimedia Subsystems on a Time-Triggered System-on-a-Chip

Roman Obermaisser, Bernhard Frömel, Christian El Salloum, Bernhard Huber
{romano,froemel,salloum,huberb}@vmars.tuwien.ac.at
Vienna University of Technology, Austria

***Abstract*—The Time-Triggered System-on-a-Chip (TTSoC) architecture enables the realization of mixed-criticality systems using SoCs. The integration of subsystems with different criticality enables massive cost reduction by reducing the overall number of devices and networks (e.g., ECUs in car). To accomplish this goal, the TTSoC architecture offers inherent fault isolation mechanisms that prevent any unintended interference between application subsystems of different criticality. This paper demonstrates these capabilities using an exemplary automotive example with a safety-critical control subsystem and a multimedia subsystem. In the demo application, it is ensured by-construction that any design fault in the multimedia subsystem cannot have any adverse effect on the safety-critical control subsystem.**

## I. Introduction

During the past forty years, the semiconductor industry has made tremendous progress that has enabled the construction of chips approaching a billion of transistors on a single die. These spectacular improvements and the associated reduction of the cost per transistor have enabled new applications of embedded systems with ever increasing functionality. A representative example is the automotive industry, where a modern luxury car contains more than 70 Electronic Control Units (ECUs) and up to 10 millions lines of code [1]. In-vehicle electronics is already the strongest innovation driver in the automotive industry and accounts for up to 35% of the total value of a car. Considering future applications, like steer-by-wire, we can expect this trend to continue.

In this context, the Time-Triggered System-on-a-Chip (TTSoC) architecture provides an integrated execution environment for the component-based design of many different types of embedded applications (e.g., automotive, avionics, consumer electronics). By thorough decoupling of the computational components from the communication infrastructure, the design of a computational component can abstract from the implementation of the interconnect, which facilitates the rapid development of multi-core SoCs by using pre-verified functional cores.

For this purpose, the time-triggered SoC architecture provides an architectural framework that supports the side-effect-free composition of component services, based solely on the interface specifications, to form larger systems-of-systems.

Through inherent fault isolation mechanisms, the TTSoC supports the integration of components with different levels of criticality. In the scope of this paper, we demonstrate this capability by integrating a safety-critical automotive control subsystem with a non safety-critical multimedia subsystem.

The paper is structured as follows. Section 2 gives an overview of the TTSoC architecture. An actual target platform that conforms to the TTSoC architecture is the focus of Section 3. We outline the use of this platform and the integration of mixed-criticality subsystems by an exemplary application in Section 4. The paper concludes with a discussion in Section 5.

## II. The TT-SoC Architecture

The central element of the presented System-on-Chip (SoC) architecture is a time-triggered Network-on-Chip (NoC) that interconnects multiple, possibly heterogeneous IP blocks called micro components (see Figure 1). 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. For this reason, the trusted subsystem prevents a faulty micro component from sending messages during the sending slots of any other micro component.

Furthermore, the time-triggered SoC architecture supports integrated resource management. For this purpose, dedicated architectural elements called the Trusted Network Authority (TNA) and the Resource Management Authority (RMA) accept resource allocation requests from the micro components and reconfigure the SoC, e.g., by dynamically updating the time-triggered communication schedule of the NoC and switching between power modes.

### A. Micro Component

The introduced SoC can host multiple application subsystems (possibly of different criticality levels), each of which provides a part of the service of the overall system. An example of an application subsystem in the automotive domain would be a braking subsystem. A nearly autonomous and possibly heterogeneous Intellectual Property (IP)-block, which is used by a particular

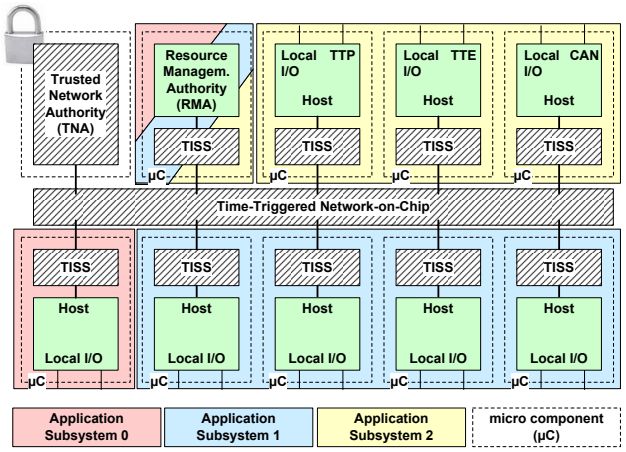


Fig. 1. Structure of Time-Triggered SoC Architecture: *trusted subsystem (shaded) and non-trusted subsystem (hosts of micro components)*

application subsystem is denoted as a micro component. A micro component is a self-contained computing element, e.g., implemented as a general purpose processor or as special purpose hardware. An application subsystem can be realized on a single micro component or by using a group of possibly heterogeneous micro components (either on one or 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. 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 to other micro components of the application subsystem via the sent messages.

Due to encapsulation, the SoC architecture supports the detection and masking of such a 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 communication or 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* [2], is required for the seamless integration of independently developed application subsystems and micro components.

Also, encapsulation is of particular importance for the implementation of SoCs encompassing application subsystems of different criticality levels. 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.

For the purpose of encapsulation, a micro component comprises two parts: a *host* and a so-called *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 or a hardware fault restricted to the host of a micro component cannot affect the exchange of messages by other micro components.

### B. Requirements for the Time-Triggered Network-on-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.

*a)* ***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 [3].

*b)* ***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 for the *periodic transmission of messages* 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.

The allocation of sending slots to micro components occurs using a communication primitive called *pulsed data stream* [4]. A pulsed data stream is a time-triggered periodic unidirectional data stream that transports data in pulses with a defined length from *one* sender to *n* a priori identified receivers at a specified phase of every cycle of a periodic control system.

The pulsed behavior of the communication network enables the efficient transmission of large data in applications requiring a temporal alignment of sender and receiver. Temporal alignment of sender and receiver is required in applications where a short latency between sender and receiver is demanded. This is typical for many real-time

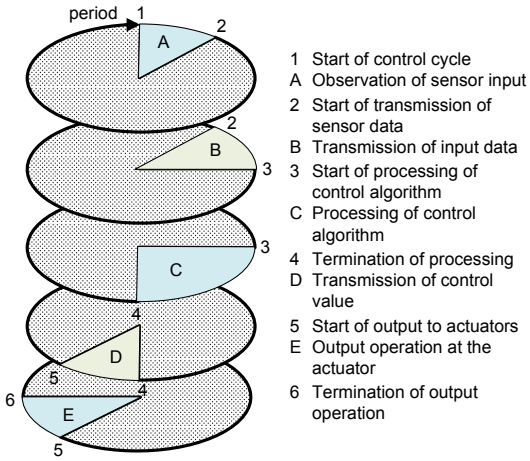


Fig. 2. Temporal Alignment in Control Loops. In this cyclic model of time, the perimeter represents the period of the control application.

systems. For example, consider a control loop realized by three micro components performing sensor data acquisition (A), processing of the control algorithm (C), and actuator operating (E) as it is schematically depicted in Figure 2. In this application, temporal alignment between sensor data transmission (B) and the start of the processing of the control algorithm (cf. instant 3 in Figure 2) as well as between the transmission of the control value (D) and the start of actuator output (cf. instant 5) is vital to reduce the end-to-end latency of the control loop, which is an important quality characteristic. By specifying two pulsed data streams corresponding to (B) and (D) in Figure 2, efficient, temporally aligned data transmission can be achieved.

Similarly, in a fault-tolerant system that masks failures by triple-modular redundancy, a high bandwidth communication service is required for short intervals to exchange and vote on the state data of the replicated channels. A real-time communication network should take consideration of these pulsed communication requirements and provide appropriate services.

### C. Architectural Elements for Resource Management

The purpose of the integrated resource management in the SoC architecture is to dynamically assign computational resources (i.e., micro components) to application subsystems and to grant communication resources and power to the individual micro components.

We distinguish two fundamentally different types of application subsystems: *Safety-critical applications subsystem* need to be certified to the highest criticality classes (e.g., class A according to DO-178B). *Non safety-critical* applications subsystems, on the other hand, do not require certification to the highest criticality classes. In general, these two types of applications subsystems will involve fundamentally different design paradigms. The focus of safety-critical applications lies on simplicity and determinism in order to facilitate thorough verification and validation. In contrast, non safety-critical applications can provide more complex application services (e.g., need to deal with insufficient a priori knowledge about the environment) and dynamism to handle the challenges of evolving application scenarios and changing environments.

The architectural elements for resource management follow this bivalent distinction of application subsystems and provide two distinct architectural elements for enabling integrated resource management, namely the Trusted Network Authority (TNA) and the Resource Management Authority (RMA). The RMA computes new resource allocations for the non safety-critical application subsystems, while the TNA ensures that the new resource allocations have no adverse effect on the behavior of the safety-critical application subsystems. As depicted in Figure 1 the TNA is part of the trusted subsystem of the SoC, whereas the RMA is not. By splitting the entire resource management into two separate parts, where only one is part of the trusted subsystem, the certification of the time-triggered SoC is significantly simplified, since the checking of the correctness of a resource allocation through the TNA is significantly simpler than its generation at the RMA.

### D. Gateways

The proposed SoC architecture supports gateways for accessing chip-external networks (e.g., TTP [5], Time-Triggered Ethernet (TTE) [6] and CAN [7] in Figure 1). The benefits of gateways include the interoperability with public networks, such as the Internet, and the ability to interconnect multiple SoCs to a distributed system. The realization of a distributed system enables applications based on the SoC architecture for ultra-dependable systems. In ultra-dependable systems, a maximum failure rate of $10^{-9}$ critical failures per hour is demanded [8]. Today's technology does not support the manufacturing of chips with failure rates low enough to meet these reliability requirements. Since components failure rates are usually in the order of $10^{-5}$ to $10^{-6}$ (e.g., [9] uses a large statistical basis and reports 100 to 500 failures out of 1 Million Electronic Control Units (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.

In case the chip-external network is also time-triggered (e.g., TTP [5], TTE [6]), the TDMA scheme of the NoC can be synchronized with the TDMA scheme of the chip-external network. Consequently, a message that is sent on the chip-external network is delivered to the micro components within a bounded delay with minimum jitter (only depending on the granularity of the global

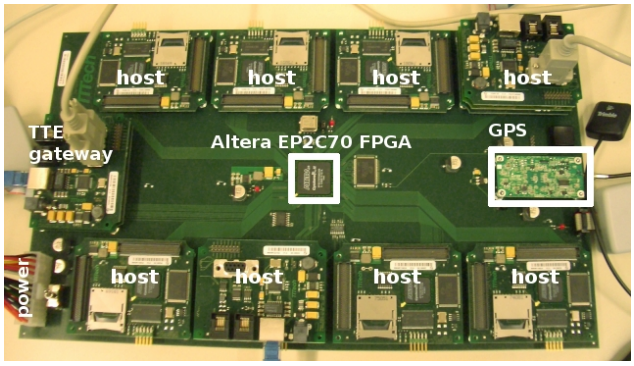


Fig. 3. Development Kit

time base). The alignment between pulsed data streams and messages on time-triggered networks ensures that replicated SoCs perceive a message at the same time, i.e., within the same inactivity interval of the global sparse time base [10]. This property is significant for achieving replica determinism [11] as required for active redundancy based on exact voting. Without synchronization between the NoC and the chip-external network, there could always occur a scenario in which one SoC forwards the message to the micro components in one period of the pulsed data stream, while another SoC would forward the message in the next period.

Furthermore, the introduced gateways provide the SoC with an externally synchronized time base. For example, the global time base of the SoC can be synchronized to GPS. Consequently, a timestamp assigned to an event is also meaningful outside the SoC. Furthermore, the global time base enables a global coordination of activities spanning multiple SoCs (e.g., output to actuators at same global point in time).

## III. Target Platform And Execution Environment

### A. MPSoC Development Kit

Our target platform is built on a custom-made, modular development kit that has been designed and manufactured during the FIT-IT TT-SoC project [12]. We use the kit to emulate a chip that is designed according to the TTSoC architecture and houses nine micro components. The kit consists of an FPGA motherboard with nine extension slots (see Figure 3). The central FPGA device resides on the motherboard: We use it for the NoC, all the TISSs and the TNA. The two other architectural components, the RMA and the gateway, are mapped to two of the nine available micro components, leaving seven free for user applications.

Each of the extension slots on the motherboard is occupied by an FPGA module which offers in return another extension slot. This enables stacking of modules. On top of the FPGA module, diverse application computers with I/O or multimedia add-on boards can be placed. In terms of the TTSoC architecture, the modules stacked on a single extension slot of the motherboard are considered as a host. The host represents, together with its TISS in the motherboard's FPGA device, a micro component. In our application, we employ two different kinds of hosts: one solely for computational needs and one other for processing and interfacing multimedia content. The first host consists of an Infineon Tricore TC1796 CPU module, while the latter one is realized differently: an FPGA module is used as an application computer and configured with the SPARC v7 Leon-III softcore CPU. On top of this FPGA application computer module, the multimedia add-on board provides a 320x240 pixel 18-bit colour LCD display and an AC97 audio codec with line-in and line-out jacks.

### B. Gateway between NoC and TT-Ethernet

One of the extension slots of the motherboard is equipped with a TTE [6] gateway, enabling off-chip communication. The gateway host consists of an FPGA module and an I/O add-on board. The FPGA is configured with the gateway logic and a TTE controller, while the I/O add-on board provides the physical TTE interface.

### C. Execution Environment

The custom execution environment running on each of the hosts is designed to initialize, boot and setup the host to run a job. In the TTSoC architecture, we consider a job as the temporally and spatially partitioned unit of work within a Distributed Application Subsystem (DAS) [13]. There are stable interfaces to the host's local hardware and to the platform's RT communication service which makes it viable to replace jobs during runtime. Further, the execution environment implements time-triggered scheduling to inform its job about message incoming and outgoing activity with lowest possible latency. The environment also takes care about reconfiguration events triggered by the RMA.

### D. Sensor

For user interaction we exploit a standard USB driving wheel with gas/brake pedals and connected it to a single-board-computer with three dedicated ethernet network interfaces. The single-board-computer is responsible of distributing the sensor input over ethernet to up to three different hosts simultaneously. That way we are able to simulate triple redundant sensors by duplicating the real sensor input 2-times.

## IV. Integrated Application

We want to show that a typical safety critical and a multimedia subsystem execute together (mixed) on our TTSoC architecture conforming target platform without interference. From a logical point of view, the application is split up into two separate DASs: an automotive control

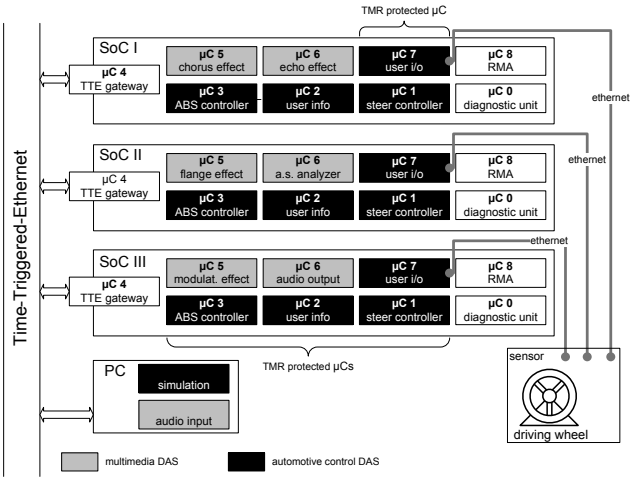


Fig. 4. Target Application Overview

Fig. 5. Racing Car Simulation

DAS and a multimedia DAS. Due to our (assumed) ultra-high dependability requirement, we want to implement the automotive control DAS in a TMR configuration, whereas we have no such requirements for the multimedia DAS. For the automotive control DAS, we need to acquire sensor data and provide actuator values from/to a force-feedback driving wheel and a car simulation running on a Personal Computer (PC). Multimedia content for the second DAS is also obtained from the PC. Physically our system is composed of three SoCs and a standard PC. All components are interconnected by TTE (see Figure 4).

### A. Automotive Control Subsystem

This DAS controls a vehicle in the open-source racing car simulator TORCS [14] by processing user input from an USB driving wheel with gas and brake pedals (sensor). Beside the adequate simulation model for testing brake-by-wire and steer-by-wire applications, the clean C++ class based software design features the development of car control modules (”robots”) that contain e.g. a keyboard, driving wheel or network I/O interface to drive a car within the simulator (see Figure 5).

The user input together with feedback from the simulation should be processed according to *(i)* an Anti-Lock Braking System (ABS) algorithm and *(ii)* a configurable driving-wheel to steering angle translation. Purpose of an ABS is to prevent wheel-locking while (emergency) braking to increase safety during driving. This is achieved by constantly monitoring the speed of each wheel and if one or more wheels rotate considerably slower than others, locking is prevented by reducing the braking force on those wheels that start to lock.

Our automotive control DAS consists of the following classes of jobs: the ABS controller job, the steering controller job, the user info job (informs the user about e.g. settings, events, ...), braking actuator jobs, wheel speed sensor jobs, a driving wheel job and braking/gas pedal jobs. For practical reasons, we merge the braking actuator and wheel speed sensor jobs with the racing car simulation that runs on the PC. To further simplify the design, the driving wheel and pedal jobs are reduced to a single ”user I/O job”.

Figure 4 shows our job deployment. To ensure ultra-high dependability, we implement TMR by instantiating each job 3-times and deploy the single instances on three physically different components. We execute the simulation on a standard PC that runs Linux with RTAI [15]. We extended the racing car simulator with our own car control module which provides the required sensor input (i.e. individual wheel speeds, forces) from the simulation to the rest of the DAS and allows to set car actuators (i.e. individual wheel brakes, gas pedal, steer) within the simulation. Due to TMR, the simulation’s outputs are replicated 3-times, while the arriving input of other jobs (e.g. actuator set values from the ABS controller job) is voted on.

We have the following port-layout:

- **simulation:** in: gas pedal actuator set value, brake actuator set value (4x), steer actuator set value
  out: wheel speed sensor values (4x)
- **user I/O:** out: driving wheel sensor value, gas pedal sensor value, brake pedal sensor value
- **user info:** in: wheel speed sensor values (4x), brake pedal sensor value, ABS status value
- **ABS controller:** in: wheel speed sensor values (4x), brake pedal sensor value
  out: brake actuator set values (4x), ABS status value
- **steer controller:** in: driving wheel sensor values
  out: steer actuator set value

### B. Multimedia Subsystem

The multimedia DAS processes audio input through various effect processors. In our design we have the following four effects: chorus, echo, modulation and flange [16]. We also permit to dynamically define which set of effects is applied on the audio input.

The DAS is designed to operate on stereo audio input with a sampling rate of 44.1 kHz and a depth of 16-bit (CD-quality, 1.35Mbit/sec). Audio input is obtained from the audio input job that runs on the PC. All jobs operate with the same audio format and thus have the same attributes with respect to period and message size. Each effect is implemented as a single job on one of the micro components. Additionally, an audio spectrum analyzer job is executed on a micro component which is equipped with a multimedia add-on board. This job displays the processed audio visually on an LCD screen by making use of the Fast-Fourier Transformation (FFT) algorithm. Another micro component, also equipped with a multimedia add-on board, is used for the audio output job, that offers the processed audio on the AC97 codec's line-out. We have a total of seven jobs in the multimedia DAS, whereas six of them are distributed equally to unused micro components on our three SoCs (see Figure 4).

Our DAS supports a configurable set of effects to be applied on arbitrary audio input. To increase resource efficiency, we employ the TTSoC architecture's feature to dynamically reconfigure the communication channels. We define several different *DAS modes*: a mode that applies no effects (= the startup mode), one that applies all effects, four different modes that apply a single effect and the reduced mode, where all effects are applied, but only half the bandwidth is used (downsampling to 22.5kHz necessary at the audio input job). Mode switches are triggered by the audio input job that conveniently allows user interaction, as it is mapped to the PC. The RMA which handles reconfiguration requests and knows about the defined DAS operational modes, creates a valid NoC access schedule for each TISS. The TNA guarantees that the reconfiguration activities have no impact on the static resource allocation of the safety critical control DAS.

We have the following port-layout:

- **effect (4x):** in: audio input value
  out: audio output value
- **audio spectrum analyzer:** in: audio input value
- **audio input:** out: audio output, RMA reconfiguration request
- **audio output:** in: audio input

## V. Conclusion

Many embedded applications encompass safety-critical and non safety-critical subsystems (e.g., infotainment and control services in the automotive or avionic domain). As shown in this paper, the TTSoC architecture enables the integration of subsystems with different criticality by providing encapsulation mechanisms that prevent any unintended interference between these subsystems. The example application described in this paper consists of two fundamentally different subsystems. The control subsystem is safety-critical and is thus replicated on three redundant SoCs by using Triple Modular Redundancy (TMR). Two of the SoCs are shared with the multimedia subsystem, which is not safety-critical and therefore does not employ active redundancy. Thereby, a massive cost benefit arises because the multimedia subsystem does not induce any additional chips or networks in the demo application. At the same time, the dependability of the safety-critical subsystem is not affected by the integration. It is guaranteed by construction that any fault in the multimedia subsystem of the demo application cannot have any adverse effect on the control subsystem.

## VI. Acknowledgments

This work has been supported in part by the European IST project ARTIST2 under project No. IST-004527, the European IST project DECOS under project No. IST-511764, and the national FIT-IT project TT-SoC under grant number 813299/7852.

## References

[1] K.D. Muller-Glaser, C. Reichmann, P. Graf, M. Kuhl, and K. Ritter. Heterogeneous modeling for automotive electronic control units using a case-tool integration platform. *Computer Aided Control Systems Design, 2004 IEEE International Symposium on*, pages 83–88, 4-4 Sept. 2004.

[2] J. Sifakis. A framework for component-based construction. In *Proc. of 3rd IEEE Int. Conference on Software Engineering and Formal Methods (SEFM05)*, pages 293–300, September 2005.

[3] OMG. Smart Transducers Interface. Specification ptc/2002-05-01, Object Management Group, May 2002. Available at http://www.omg.org/.

[4] H.Kopetz. Pulsed data streams. In *IFIP TC 10 Working Conference on Distributed and Parallel Embedded Systems (DIPES 2006)*, pages 105–124, Braga, Portugal, October 2006. Springer.

[5] H. Kopetz and G. Grünsteidl. TTP – a protocol for fault-tolerant real-time systems. *Computer*, 27(1):14–23, January 1994. Vienna University of Technology, Real-Time Systems Group.

[6] H. Kopetz, A. Ademaj, P. Grillinger, and K. Steinhammer. The Time-Triggered Ethernet (TTE) design. *Proc. of 8th IEEE Int. Symposium on Object-oriented Real-time distributed Computing (ISORC)*, May 2005.

[7] Robert Bosch Gmbh, Stuttgart, Germany. *CAN Specification, Version 2.0*, 1991.

[8] N. Suri, C.J. Walter, and M.M. Hugue. *Advances In Ultra-Dependable Distributed Systems*, chapter 1. IEEE Computer Society Press, 10662 Los Vaqueros Circle, P.O. Box 3014, Los Alamitos, CA 90720-1264, 1995.

[9] B. Pauli, A. Meyna, and P. Heitmann. Reliability of electronic components and control units in motor vehicle applications. In *VDI Berichte 1415, Electronic Systems for Vehicles*, pages 1009–1024. Verein Deutscher Ingenieure, 1998.

[10] H. Kopetz. Sparse time versus dense time in distributed real-time systems. In *Proc. of 12th Int. Conference on Distributed Computing Systems*, Japan, June 1992.

[11] S. Poledna. Replica determinism in distributed real-time systems: A brief survey. *Real-Time Systems*, 6:289–316, 1994.

[12] TT-SoC. Report on board manufacture. FIT-IT TTSoC project deliverable D3.2, November 2007.

[13] Roman Obermaisser, Hermann Kopetz, Christian El Salloum, and Bernhard Huber. Error containment in the time-triggered system-on-a-chip architecture. In *Proceedings of the International Embedded Systems Symposium (IESS'07)*, Irvine, CA, USA, May 2007.

[14] Christophe Guionneau Eric Espi. Torcs, the open racing car simulator, 1997. At http://torcs.sf.net.

[15] D. Beal, E. Bianchi, L. Dozio, S. Hughes, P. Mantegazza, and S. Papacharalambous. RTAI: Real-Time Application Interface. *Linux Journal*, April 2000.

[16] Udo Zoelzer, editor. *Dafx: Digital Audio Effects*. John Wiley & Sons, Inc., New York, NY, USA, 2002.