

Increasing System Safety for by-wire Applications in Vehicles by using a Time Triggered Architecture

Th. Ringler^{*}, J. Steiner^{*}, R. Belschner[#] and B. Hedenetz[#]

^{*} University of Stuttgart, Institute for Industrial Automation and Software Engineering
Pfaffenwaldring 47
D-70569 Stuttgart, Germany,
Tel.: +49 - (0)711 - 685 - 7296, Fax: - 7302
email: {ringler, steiner}@ias.uni-stuttgart.de

[#] Daimler Benz AG
FT2/EA, HPC T721
D-70546 Stuttgart, Germany,
Tel.: +49 - (0)711 - 17 - 41930 / - 41341, Fax: - 47058
email: {belschner, hedenetz}@dbag.stg.daimlerbenz.com

Abstract. *By-wire* systems have been established for several years in the area of aircraft construction and there are now approaches to utilize this technology in vehicles. The required electronic systems must evidently be available and safe. In the same time the requirements of mass production have to be reached (long life time, long maintainability intervals, low costs, fulfillment of standards). This paper addresses a new automotive architecture approach - based on a time triggered architecture - and a framework for the application design of future by-wire systems in vehicles.

Introduction

The driving forces transforming our world towards the information society are also dramatically changing the mechanical engineering market domains. There is a clear trend to substitute mechanical, hydraulic, or pneumatic systems by computerized by-wire components in automotive systems. The aircraft domain has been the technology driver till today. A well-known example is the flight control system of the Airbus A320. However, it is estimated that the automotive industry with its huge market for components will take this role in future. It is estimated that in 2006, 25% of the total cost of a passenger car will be accounted for electronic parts. This also includes electronic brake and steering systems, which are supposed to offer a better functionality than conventional systems and therefore have the potential to increase overall traffic safety significantly. Further advantages but not less important are reduced costs for assembly during line production (reduction of packaging problems,

no brake fluid, no bleeding, simple maintenance) and easier adapting of assistance systems like anti braking system (ABS), electronic stability program (ESP), etc., since they could be realized only by software and sensors without additional mechanical or hydraulic components. To be affordable for mass production, the electronic systems have to offer the same availability and simultaneously safety of their mechanical/hydraulic counterparts, so that no mechanical/hydraulic backup solutions are needed. This means in the last consequence that the safety critical system must provide an electronic backup system which is realized by electronic redundancy.

The Time-Triggered Approach

A system architecture that suits best for the described application domain is based on the Time-Triggered (TT) approach. In a TT architecture all system activities are initiated by the progression of a globally synchronized time. It is assumed that all clocks are synchronized and every observation of the controlled object is timestamped with the synchronized time.

This paper addresses a new automotive architecture approach using the fault-tolerant Time-Triggered Protocol TTP [Kope97] that has been designed due to the class C SAE classification [SAE93a] for safety critical control applications, like brake-by-wire or steer-by-wire. Time-triggered systems based on TTP can provide the required high level of dependability, which has been clearly demonstrated in the ongoing EC-funded projects X-By-Wire [Dilg97a, Dilg97b] and TTA [HeTh98, Sche97]. The intention of the TTP architecture is to tolerate one arbitrary fault within electronic systems without any effects to the system behavior.

Time-Triggered Protocol (TTP)

For the realization of a distributed time-triggered architecture a communication network is necessary that provides the features mentioned above. None of the commonly used in-vehicles communication systems (CAN, A-BUS, VAN, J1850-DLC, J1850-HBCC[SAE93b]) meets the requirements for safety related by-wire systems since they were not designed for this case [KrSc97]. They are all lacking in being deterministic, in synchronization and fault tolerance characteristics. These missing properties are the motivation for developing new approaches for in-vehicle communication systems. As a new start we examine the Time-Triggered Protocol developed by the University of Vienna and Daimler-Benz Research. TTP is especially designed for safety related applications and fulfills these requirements. TTP is an integrated time-triggered protocol that provides:

- a membership service, i.e., every single node knows about the actual state of any other node of the distributed system

- a fault-tolerant clock synchronization service (global time-base),
- mode change support,
- error detection with short latency,
- distributed redundancy management.

All these issues are supported implicitly by the protocol itself. A comprehensive description of the TTP protocol is given in [KoGr94, Kope97a, Kope97b]. The TTP protocol has been designed to tolerate any single physical fault in any one of its constituent parts (node, bus) without an impact on the operation of a properly configured cluster [KoGr94].

The overall TTP hardware architecture is characterized by both the TTP system architecture and the TTP node architecture as shown in Fig. 1. A TTP real-time system consists of a host subsystem, which executes the real-time application and the communication subsystem providing reliable real-time message transmission. The interface between these subsystems is realized by a dual ported RAM (DPRAM) called *Communication Network Interface (CNI)* [KrKo95, Krüg97]. The assembly of host and TTP-controller is called *Fail Silent Unit (FSU)*. Two FSUs form a single redundant *Fault-Tolerant Unit (FTU)*. The physical layer consists of two independent transmission channels.

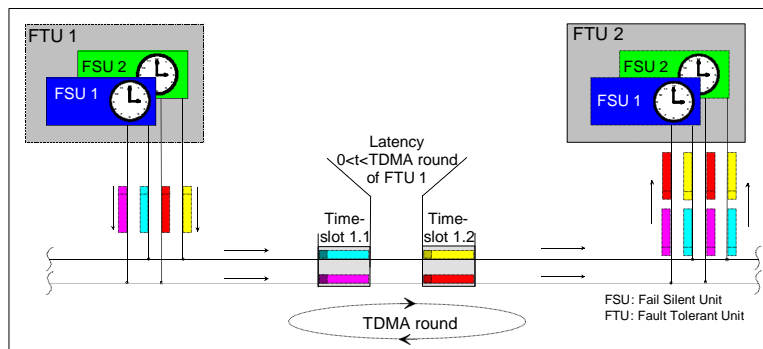


Fig. 1.: Architecture of a TTP-Based Fault-Tolerant Real-Time System

The communication network topology is a broadcast bus. Bus access is granted to the nodes under the control of a static TDMA scheme. In accordance to the time-triggered concept, the timeslots in which a node may send and receive frames on the bus are pre-allocated at the design time of the system. This TDMA message schedule is stored locally within each node. To achieve an orderly access to the bus, the nodes maintain a globally synchronized time base. This operation is an integral part of TTP. For fault-tolerance reasons a dual channel system has to be used.

Brake-by-Wire Case Study

As an example we are currently evaluating this approach within a brake-by-wire research car [HeBe98] (case study) without mechanical backup (see Fig. 2). The intention of this architecture is to tolerate one arbitrary fault without any effects of the system's performance. For this purpose, we use hot redundancy in hardware (electronic computing units - ECU) as well as redundancy in communication (TTP). Nevertheless the electric components like sensors, actuators and power supply are also designed redundantly.

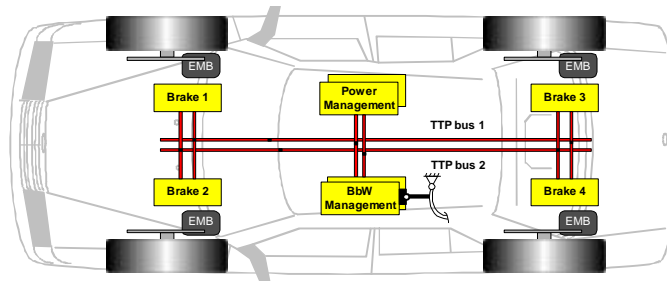


Fig. 2.: Brake-by-Wire Case Study

Our fault tolerant architecture consists of a set of two redundant ECU's (electronic control unit) for both the *Brake-by-Wire-Manager* (*BBW-Manager*, *BBWM*) and the *Power-Manager* (*PM*), and 4 single ECU's, one for each brake (see Fig. 2). The ECU's are connected by two replicated busses. In this case study the brake ECU's are not designed redundant, in order to reduce costs, since the failure of a single brake is not considered to be as severe as the failure of the BBW-Manager, or the Power-Manager.

The function of the BBW-Manager is to read the sensor values of the brake pedal, the revolution counters of the wheels, the yaw-sensor, the acceleration sensors, and to calculate from these signals the brake force set points for the four brake actuators. The BBW-Manager should also manage higher assistance functions like ABS, traction and driving dynamic control. The Power-Manager controls the charge of the batteries, proceeds active power management and monitors the status of the power supply. If the generator or one of the redundant power circuits fails or the charge of the batteries is going low the Power-Manager generates a warning signal. Each of the four brake actuators have one brake electronic to control the electric motor of the brake actuator. The brake electronics get the brake force set points from the BBW-Manager.

In contrast to event triggered systems a TTP system is built up by defining first the static message schedule. Fig. 3 depicts the result of this phase, namely the communication matrix of the above example on a brake-by-wire network. The Fig. 3

shows the static synchronous TDMA schedule and its constraint that each subsystem has to send exactly once in a TDMA cycle. The messages marked with 'I' are called I-Frames, used for reintegration of lost members and do not transmit information for the application layer.

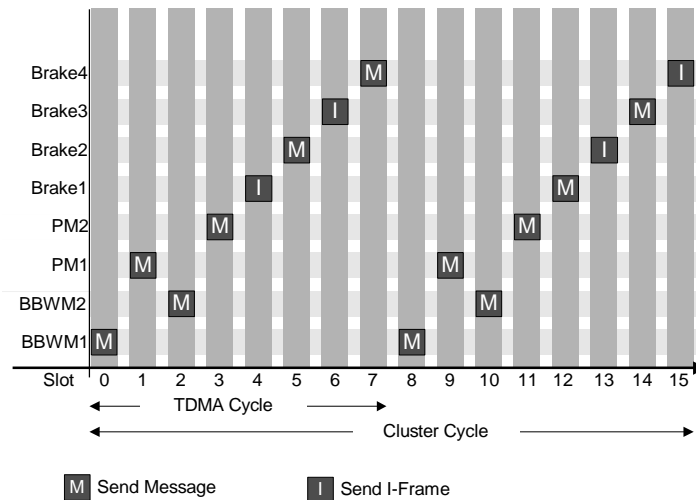


Fig. 3: Communication Matrix of the Brake-by-wire Case Study

The FSU's of the BBWM and the PM send their messages in vice versa time slots. In this way short burst errors can be recovered. A FSU slot has a length of about 1.2 msec. New brake force set points could be sent every 10 msec, which is needed for an ABS control loop. The brake control ECU's send their status and the current brake force. These messages are not as time critical as the transmission of the brake force set points. The brake ECU's send their messages only once in a cluster cycle, each 16 FSU slots. In the leaving FSU slots the brake ECU's send I-Frames for the network management.

Application Design Approach

In our brake-by-wire case study, a software fault, the exact same software running on redundant ECU, will lead to the same faulty reaction. This could be prevented by diverse software design, possibly developed by different teams. It would take more than double the effort and studies show that diverse software design does not automatically increase the reliability of the system [Bril90]. We think a more useful approach is to invest in the additional expenditure of reusable software components.

Software Components

Unlike the development of today's software, which is generally directly based on command lines, a comparable procedure in hardware development is inconceivable. In an analogous manner, no-one would think to achieve the functionality of an integrated circuit (IC) by using individual diodes, resistors and transistors. With the introduction of integrated hardware components the work of hardware engineers has changed from dealing with the detailed problems of circuit development to the integration of standard hardware components (e.g., IC). To achieve a corresponding method for software development, software components are needed in order to build an application from a design level. Therefore, the application designer does not need to deal with details at the code level.

The advantages for utilizing software components within the whole application system are simple maintainability and expandability, system quality improvement and increasing system safety with the use of sufficiently tested components. There are also predictable time and cost factors for project planning.

The development of software components requires an initial large investment, which is only worthwhile if the components are reusable to a high degree. The problem of specific software components is that their functionality is pre-defined and fixed. If the available components do not fulfill the desired requirements, a new component must be found or developed in order to fulfill these special requirements. This inflexibility reduces the reusability of software components. A promising solution for solving the problem of specific software components is to use *generic components*. We define a generic component as a reusable system element which has a closed functional unit that represents a specific problem area. However, unlike a specific software component, a generic component has flexible characteristics. By configuring certain system characteristics and setting certain parameters, a generic component can be adapted to the specific requirements of an application to be developed.

Operating Systems

We are convinced that deterministic system behavior is basically needed for safety critical applications. Determinism can only be reached, if the application software is also time triggered. In the case of TTP, the application software system is strongly interconnected with the time division multiple access (TDMA) bus access scheme. In contrast to event-triggered networks, the global time triggered network rules over the application. The TTP communication subsystem has predefined, static binding points in time, when the host subsystem has to interact with the communication subsystem and, therefore, the operating system has to take care that the static time bounds are guaranteed. If the host subsystem does not interact as defined, the communication subsystem assumes a host error and becomes passive, due to the claimed fail-silent

behavior. This cannot be managed by tasks themselves, sometimes developed by different subcontractors. A fail tolerant operating system (FTOS) is necessary.

Prime-Level Scheduler (PLS)

As a prototype, an operating system called Prime-Level Scheduler (PLS) has been developed [Krug98]. The PLS is a time driven scheduler, which is based on an off-line task schedule. Tasks, which are scheduled by the PLS, are called fault-tolerant tasks (FT-tasks). The FT-tasks are only activated by the progression of time, they are non-preemptive and repeated periodically. The PLS is responsible that the time constraints of the FT-tasks are met. This concept was extended by features and services especially relating to the synchronization with TTP. In the following list the main characteristics of the PLS are summarized:

- The FT-tasks are executed cyclically by a given start time. The cycle time is based on the duration of one TTP-cluster-cycle. If the task exceeds the deadline, the task will be suspended (see Fig. 4).
- The time management is synchronized with the global time of TTP.
- The reintegration of a suspended task for the next cycle is optional.
- The PLS terminates itself in a predefined way if certain tasks are suspended.
- In the case of a failure of the communication network (reset, short-circuit, interruption), the PLS detects this and goes into an exception mode. In this mode, the FT-tasks are still executed in accordance with the tasks timetable.
- During run time, the PLS sends additional messages for diagnosis via CAN (Controller Area Network).

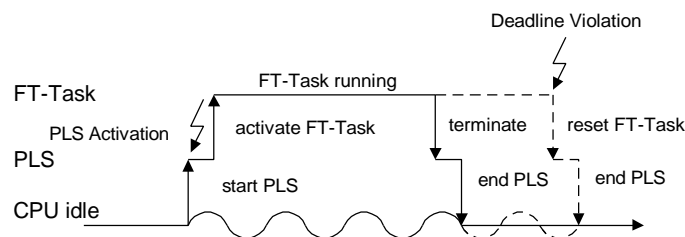


Fig. 4: PLS call (the dotted lines illustrate the faulty case)

For the deadline value of a task, its worst-case execution time has to be known. Since the execution time varies, the available CPU time isn't completely used. For that reason, a Multi-Level Scheduler has been developed.

Multi Level Scheduler (MLS)

In order to use the CPU time in a more efficient way, a second task level for non fault-tolerant tasks (NFT-tasks) was introduced with the MLS [Fisc98]. This results in a two level system consisting of a primary and secondary level. The primary level has the functionality of the described PLS. In contrast to the primary level, the secondary level is to perform non safety related tasks without time constraints. The

non safety related tasks are only executed, if the MLS detects enough free time at the primary level.

Examples of task classification:

Primary Level

- tasks dependent on the TTP communication
- tasks operating with quantitative variables (e.g., the loss of these variables would lead to a damage of vehicle and person)

Secondary Level

- tasks delivering diagnostic information (e.g., diagnostic information for automobile workshop)

More than two task levels are not practical, because the more levels exist, the more time has to be invested by the CPU for organizing the task levels.

Generally, to generate the required predefined schedule, the worst-case execution times of all tasks has to be known. Possible methods are described in the following chapter.

Worst-Case Execution Time (WCET) Analysis

The above mentioned PLS and MLS have to be supplied with the worst-case execution times which have to fulfill two requirements, they have to be safe, means they must not under-estimate the worst case and they have to be tight, to avoid the waste of resources.

Real-Time Software in Automotive Applications

The software structure of automotive applications is characterized by a cyclic execution: Reading in physical values, execution of algorithms and giving out the computed values. Assembler has been the standard programming language, which has been optimized manually due to the high pressure on costs of the used hardware. But there is a visible trend to use higher level languages especially C and code, generated by tools, which have been improved in respect to code efficiency.

There have only used processor with simple architecture and flat memory in automobiles. But the increasing functionality, which will be realized in future by software, leads to usage of hardware platforms with higher performance. The decision, if platforms with caches and pipelines will be used in time triggered architectures depends on the powerfulness of WCET-analysis and the possibilities in making them deterministic enough by deterministic cacheing strategies for example.

Known Approaches

The estimation of the worst-case execution time of a real-time software is a complex task. The influences of all participating parts - application software, compiler and used hardware platforms have to be taken into consideration. There are described

two general approaches for estimating the worst-case execution time in literature [Kope97c]: The pragmatic approach by on-line measurement and the analytical approach by modeling all participating parts. Their characteristics are discussed below:

Analytical Approach - Modeling

An analytical model has to be built for each part (software, compiler and hardware platform). Because of the strong influence of the underlying hardware and used the compiler, the analysis has to be done at source code and machine code level. By this model-based method, it is generally possible to compute safe worst-case times, and there could be reached successes for simple hardware architectures [PuSc97], but it is still a big challenge for modern hardware architectures with parallel units and hierarchical memory organization. There are known many approaches focusing sub-problems, but there is still a lack of a general method. And because of the complexity there is no professional tool available [Pusch97].

Pragmatic Approach - Measurement

This approach is based on measuring the execution times on-line during runtime without costly modeling, only with few additional lines of code and external equipment like logic analyzer or oscilloscope. But there is no guarantee to find the path through the code with the maximum execution time.

We are convinced, that for practical usage of the WCET-analysis a mixture between the presented approaches is required, which allows fast configuration for other hardware platforms and gives a maximum of safe bounds of predicted WCET (but no guarantee). So we have focused our researches to a method, based on intelligent measurement including model-based components: The source-code is included with additional functions, which are recording timing information during runtime, in an early state of system evaluation. After runtime, timing information is analyzed by a timing tool, computing the worst-case path.

Developing Tools

Every part of the developing process should be supported by tools in order to handle the complexity of the system, accelerate the design process and increase the reliability. A tool environment for realizing time triggered architectures has to support the design of the task and message schedule, the realization of the application, the system integration and the test of the availability and safety requirements. We need several kinds of tools:

- For definition of the static message and task schedule a tool has to be available. This tool has to allow a graphical definition of the time and message constraints and generates a valid message and task schedule. A first prototype realization was developed by the University of Vienna [Noss97].

- For realization of applications in the automotive environment typically tools like Matlab/Simulink and StateMate are used, to develop control loops and simulate the system behavior. These tools have to be adapted to the constraints of time triggered architectures. Also the possibilities to integrate software components and worst-case execution times have to be supported.
- For system integration a tool for monitoring the messages on the communication channels and for simulation of non-available nodes is required. During development the complete system is usually not available or subsystems are developed by other departments or outsource partners. For testing and evaluating issues the simulation of the non-available nodes are required [Flei97].
- For testing the availability and safety requirements we need the possibilities for fault injection in software and hardware and reliability modeling. For fault injection on the communication bus a fault injection device called TTP-stress is available [HeSc98].

Conclusion

In this paper we present a new architecture approach for future automotive by-wire systems without mechanical or hydraulic backup based on a time triggered architecture. In the first part of the paper we introduce the time triggered architecture and the time triggered protocol. We showed that time triggered architectures are well suited for safety critical by-wire applications. In the second part we describe demands for the application design. This part contains demands for reusable software components, for the operating systems and tools for the developing process.

References

- [Bri90] S. Brilliant, J. C. Knight, N. Leveson: „Analysis of faults in an N-version software experiment,“ IEEE Transactions on Software Engineering, SE-16(2), February 1990.
- [Dilg97a] E. Dilger et al.: „Towards an Architecture for Safety Related Fault Tolerant Systems in Vehicles“, Proceedings of the ESREL '97 International Conference, June 1997.
- [Dilg97b] E. Dilger, T. Führer, B. Müller, S. Poledna, T. Thurner: „X-By-Wire: Design von verteilten, fehlertoleranten und sicherheitskritischen Anwendungen in modernen Kraftfahrzeugen“, Tagungsband der 17. VDI/VW-Gemeinschaftstagung Systemengineering in der Kfz-Entwicklung, Wolfsburg, 3.12. - 5.12.97
- [Fisc98] M. Fischer, „Entwicklung eines fehlertoleranten TTP-basierten Multi-Level-Schedulers für den 80C167“, master thesis, Institute for Industrial Automation and Software Engineering, University of Stuttgart, Stuttgart, June 1998
- [Flei97] W. Fleisch, Th. Ringler, R. Belschner: „Simulation of application software for a TTP real-time subsystem“, European Simulation Multi Conference, May 1997.
- [HeBe98] B. Hedenetz, R. Belschner, „Brake-by-wire without Mechanical Backup by Using a TTP-Communication Network“, SAE International Congress 1998, SAE 981109.

- [HeSc98] B. Hedenetz, A. V. Schedl: „Fault Injection and Fault Modeling for a Safety Critical Automotive Communication System“, ESREL 98, Trondheim, Norway, June 1998.
- [HeTh98] G. Heiner, T. Thurner, „Time-Triggered Architecture for Safety-Related Distributed Real-Time Systems in Transportation Systems“, FTCS-28, June 1998.
- [Karl95] J. Karlsson, P. Folkesson, J. Arlat, Y. Crouzet, G. Leber, „Integration and Comparison of Three Physical Fault Injection Techniques“, Predictably Dependable Computing Systems, Springer Verlag 309-329, 1995.
- [KrKo95] A. Krüger, H. Kopetz, „A Network Controller Interface for a Time-Triggered Protocol“, SAE Symposium on Future Transportation Electronics: Multiplexing and In-Vehicle Networking, SAE, 1995.
- [Kope94] H. Kopetz, G. Grünsteidl, „TTP - A Protocol for Fault-Tolerant Real-Time Systems“, IEEE Computer, pages 14-23, January 1994.
- [Kope97a] Kopetz et al.: „A Prototype Implementation of a TTP/C Controller. Proc. SAE Congress '97“, Detroit, Michigan, 1997.
- [Kope97b] H. Kopetz, „Real-Time Systems - Design Principles for Distributed Real-Time Systems“, Kluwers Academic Publishers, 1997.
- [Kope97c] H. Kopetz, „The Systematic Design of Embedded Real-Time Systems“, Three day intensive Seminar, Munich, 1996
- [KrSc97] Markus Krug and Anton V. Schedl: „New Demands for Invehicle Networks“, Proceedings of the 23rd Euromicro Conference", pp. 601-606, Budapest, Hungary, September 1997.
- [Krüg97] A. Krüger, „Interface design for Time-Triggered Real-Time System Architectures“, doctor thesis, Institut für Technische Informatik, Vienna University of Technology, 1997.
- [Krug98] M. Krug: „Concept and Implementation of a Dependable Automotive Operating System“, doctor thesis, Institut für Technische Informatik, Universität Tübingen, 1998.
- [Krüg97] A. Krüger, „Interface design for Time-Triggered Real-Time System Architectures“, doctor thesis, Institut für Technische Informatik, Vienna University of Technology, 1997.
- [Noss97] Roman Nossal, „An Application-Oriented Methodology for the Development of Real-Time Applications“, doctor thesis, Institut für Technische Informatik, Vienna University of Technology, 1997.
- [PuSc97] P. Puschner and A. Schedl. „Computing maximum task execution times - a graph-based approach“, Real-time Systems, 13(1):67-91, July 1997
- [PuVr97] P. Puschner and Alexander Vrhoticky „Problems in Static Worst-Case Execution Time Analysis“, Research Report No. 6/96, Institut für Technische Informatik, Vienna University of Technology, 1996.
- [SAE93a] SAE, „Class C Application Requirement Considerations“, SAE Recommended Practice J2056/1, SAE, June 1993.
- [SAE93b] SAE, „Survey of Known Protocols“, SAE Information Report J2056/2, SAE, April 1993.
- [Sche98] Scheidler, G. Heiner, R. Sasse, E. Fuchs, H. Kopetz, C. Temple, „Time-Triggered Architecture - (TTA)“, Advances in Information Technologies: The Business Challenge, J.-Y. Roger et al. (Eds.), IOS Press, 1997, pages 758-765.