DISSERTATION

A Transparent Online Test Approach for Time-Triggered Communication Protocols

ausgeführt zum Zwecke der Erlangung des akademischen Grades
eines Doktors der technischen Wissenschaften

unter der Leitung von

a.o.Univ.-Prof. Dipl.-Ing. Dr. A. Steininger
Inst.-Nr. E182/2
Institut für Technische Informatik
Embedded Computing Systems Group

eingereicht an der Technischen Universität Wien
Fakultät für Informatik

von

Dipl.-Ing. Eric Armengaud
Matr.-Nr. 0126956
Andritzer Reichsstraße 47 / 8
8045 Graz

Graz, im Juni 2008 ________________________

Abstract

Modern cars comprise different Electronic Computer Units that are interconnected by shared communication medias. This architecture enables a better apprehension of the environment and results in improved car behavior. At the same time, the resulting system complexity and further, the introduction of electronics for safety-critical applications raises dependability problems; human life and health are depending on the correct operation of the car. The time-triggered architecture [KB03] has been introduced in this context to cope with the growing complexity and to support highly dependable systems. While this architecture substantially eases system design and development, it does not explicitly support verification and maintenance operations.

This work is focused on the network, since this resource has been quickly recognized as playing a central role in maintaining the system in a safe state [NSSLW05]. Our intention is to provide an approach to test the *reliability* (continuity of correct services) of the communication system as well as the *availability* (readiness for correct services [ALRL04]) of the associated error detection mechanisms. We propose a new test approach that is *generic* (not rely on dedicated services), *non-intrusive* (no modification of the system), and *transparent* (to avoid deviation of normal service delivery). This last attribute enables test operation to be performed concurrently to normal operation, and thus decreases the test period, even if the system can not be halted for maintenance. This in turn minimizes the probability of system failure due to fault accumulation.

Our test approach is first based on the monitoring of the bus traffic. This is because the deterministic behavior of time-triggered communication systems provides a-priori known properties that can be efficiently checked for correctness. This pure monitoring approach is further complemented by a subtle stimulation of the clock synchronization mechanisms to test the error detection mechanisms within the nodes' receive path. This second approach exploits the tolerance boundaries of the clock correction and drives the system into a non-natural but still correct state. Since this state can only be reached and maintained with correct operation of the tester, the node reaction therefore provides information whether the tester frames have been correctly processed.

A main concern of this work is to prove the transparency of our approach. More especially, we introduce *deterministic replay operation* as a technique to remotely control the nodes' clock correction and prove that our method results in no threats for the system operation. Moreover, we show that the nodes' logical clocks always return within the accuracy of their underlaying oscillators. This property provides information about the current forces within the system and more especially on whether the additional test frames have been received or not. The theses presented in this work focus on the TTP/C and FlexRay communication protocols and the results are supported by simulations and experimental evaluations.

Kurzfassung

Moderne Autos umfassen eine Vielzahl an elektronischen Steuergeräten, die mittels verteilter Kommunikationssysteme verbunden sind. Diese Architektur ermöglicht eine bessere Wahrnehmung der Umgebung und daher eine bessere Reaktion des Autos. Gleichzeitig zieht jedoch die resultierende Komplexität des Systems und im weiteren die Einführung von sicherheitskritischen Anwendungen Verlässlichkeitsprobleme mit sich: Leben und Gesundheit von Menschen sind von der korrekten Funktion des Autos abhängig. In diesem Zusammenhang wurde die zeitgesteuerte Architektur [KB03] eingeführt, um einerseits die Komplexität des Systems besser bewältigbar zu machen und anderseits eine zuverlässige Architektur für sicherheitskritische Anwendungen anzubieten. Diese Architektur erleichtert zwar das Design und die Entwicklung des Systems, bietet jedoch keine direkte Unterstützung für Test und Wartung.

Diese Doktorarbeit ist auf das Netzwerk fokussiert, da diese Ressource eine zentrale Rolle für das Einhalten eines sicheren Systemzustandes darstellt [NSSLW05]. Es wird eine Methode vorgeschlagen, die sowohl die *Zuverlässigkeit* des Kommunikationssystems als auch die *Verfügbarkeit* [ALRL04] der darunterliegenden Fehlererkennungsmechanismen testet. Unsere Ansatz ist *generisch* (unabhängig von der Applikation), *nicht intrusiv* (keine Änderung des Systems) und *transparent* (keine Änderung der erbrachten Dienste). Die letztere Eigenschaft ermöglicht vor allem das Durchführen von Testoperationen gleichzeitig zur normalen Systemoperation. Infolgedessen kann die Testperiode minimiert werden, womit sich die Wahrscheinlichkeit eines Absturzes in Folge von Fehlerakkumulation verringert.

Unser Ansatz beruht zunächst auf der Überwachung des Kommunikationsmediums, denn das deterministische Verhalten des zeitgesteuerten Systems bietet voraussehbare Ereignisse an, die leicht zu verifizieren sind. Dieser Ansatz wird durch eine Stimulierung der Uhrensynchronisation weiter vervollständigt, um die Fehlererkennungsmechanismen innerhalb der Empfangspfade der Knoten zu testen. Dieser Ansatz macht sich die Toleranzgrenze der Uhrkorrektur zunutze und bewegt das System in einen nicht-natürlichen aber trotzdem gültigen Zustand. Da dieser Zustand nur aufgrund der Stimulation durch unseren Tester erreichbar ist, liefert die Reaktion der Knoten Informationen, ob die Tester Frames richtig verarbeitet worden sind oder nicht.

Ein wichtiger Beitrag in dieser Arbeit besteht darin, die Transparenz unseres Ansatzes zu beweisen. Im Besonderen führen wir die *deterministische Replay Operation* als eine Methode zur Fernsteuerung der Uhrensynchronisation des Knotens ein und beweisen weiter, dass dieser Ansatz keine Gefahr für das System darstellt. Wir zeigen außerdem, dass die Genauigkeit der logischen Uhrzeit eines Knotens immer innerhalb der Genauigkeit des Quarzes zurückkehrt. Diese Eigenschaft liefert Informationen über die aktuellen Kräften im System und insbesondere Informationen, ob die Tester-Frames richtig bearbeitet wurden. Die Ergebnisse dieser Arbeit sind auf TTP/C und FlexRay fokussiert und werden durch Simulationen und Experimente gestützt.

Résumé

Les voitures modernes sont composées de plusieurs dizaines d'unités de contrôle électronique interconnectés par des bus de terrain. Cette architecture améliore l'appréhension de l'environnement et permet ainsi une réaction de l'automobile plus adaptée. Cependant, la complexité résultante ainsi que l'introduction de composants électroniques pour des applications critiques amènent des problèmes de fiabilités : la vie et la santé d'êtres humains dépendent du bon fonctionement de leur voiture. L'architecture à temps partagé TTP [KB03] à été introduite dans le but de gérer la complexité et de supporter le développement de systèmes critiques. Bien que cette architecture supporte la conception et le développement de ce type de systèmes, elle n'offre que peu de support pour les opérations de vérification et de maintenance.

Ce travail est focalisé sur le réseau car cette ressource joue un rôle fondamental pour la stabilité du système [NSSLW05]. Notre intention est de proposer une approche pour tester la *fiabilité* (continuité du service) du réseau ainsi que la *disponibilité* [ALRL04] des méthodes de détection d'erreurs associées. Nous proposons dans ce sens une nouvelle méthode de test *générique* (pas de support de la part de l'application), *non-intrusive* (pas de modification du système) et *transparente* (évite les modifications dans la prestation de service du système). Cette dernière propriété rend possible l'exécution du test parallèlement au fonctionnement standard du système et donc réduit le temps moyen entre deux exécutions du test. Cette propriété réduit à son tour la probabilité d'accumulation de fautes et de défaillance du système.

Notre approche est basée premièrement sur la surveillance du traffic réseau. Effectivement, le comportement déterministique de l'architecture à temps partagé génère un comportement à-priori connu qui peut être utilisé de manière très efficace durant la phase de test. Cette première approche est de plus complémentée par une subtile stimulation du méchanisme de resynchronisation d'horloge afin de tester les méchanismes de détection d'erreurs au sein des nœuds. Cette deuxième approche utilise l'intervalle de tolérance pour la correction d'horloges et entraîne le système dans un état correct bien que non-naturel. Etant donné que le système ne peut atteindre – et rester dans – cet état que avec un fonctionnement correct de notre testeur, la réaction des nœuds donne des indications sur le status du traitement des messages de notre testeur.

Un point important de ce travail est de prouver la transparence de notre approche. Pour ce faire, nous introduisons la *reproduction déterministique de traffic réseau* comme méthode pour contrôler à distance la correction d'horloge des nœuds et prouvons que cette méthode ne représente pas de danger pour le système. De plus, nous montrons que l'exactitude des horloges est bornée par l'exactitude de leurs quartz. Cette propriété donne des indications sur les forces exercées sur le système et plus particulièrement si les messages de notre testeur ont été effectivement reçus. Les thèses présentées dans ce document sont focalisées sur les protocoles de communication TTP/C et FlexRay et sont illustrées par des simulations et expériences sur prototype.

Acknowledgments

This thesis was carried out during my employment as research assistant within the Embedded Computing Systems group at the Technical University of Vienna. The first round of thanks goes to my supervisor, Prof. Andreas Steininger, for introducing me to the area of testing and supporting this work with stimulating discussions. He gave me motivation when mine was down, and also helpful critiques when my enthusiasm lead me to forget some of the basics.

Further I would like to thank my colleagues at the department for the good atmosphere and the fruitful discussions that help me to improve the quality of this work. Hereby special thanks to Thomas Handl for his time, for listening and providing precious assistance in all those everyday annoying problems, and for the high-quality hints, reviews and comments.

This work wouldn't have obtained this quality without the help of my former colleagues from Elektrobit Vienna (formerly Decomsys). They introduced me to the fascinating world of FlexRay and supported me during my first industrial experiences with digital design. I would like to thank more especially the chip design team for teaching me the "Viennese way-of-life" and helping me to understand the subtleties of the Viennese dialect. Many thanks to Oliver Maischberger, Wolfgang Forster, Hannes Friedl and Florian Rothensteiner for their support during the organization and improvement of the prototype and for the interesting discussions.

Moving to the more private part of the acknowledgments, I would like to thank my parents for their support during my studies in France and for providing me the possibility to come abroad to Austria.

Special thanks goes to my daughter for coming to me each morning with her smile and saying "ayo papa!", thus giving me the energy required during the hard work of writing this document. Last but not least, my best thanks goes to my partner, Christina Kugi, for her precious support during everyday life, for the passionate discussions and for her love.

Merci!

Supported by the Austrian BM:vit FIT-IT project ExTraCT, project no. 810834.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Imagine if every Thursday your shoes exploded if you tied them the usual way. This happens to us all the time with computers, and nobody thinks of complaining.

JEF RASKIN (1943 – 2005)

Dozens of complaints like *"...during the drive the electronics gave out, the engine switched off for one second and also the servo-steering dropped to a crawl"* were listed by an Austrian automotive magazine [Zei06]. This survey describes the problem with the complexity of the electronic architecture that confronts the car industry nowadays, and with which every car maker is equally concerned. Electronic Component Units (ECUs) were introduced in the 80's to reduce manufacturing costs, improve functionality and even enable new, complex services. However, the part of electronics in cars is growing more and more and it is nowadays a science in itself to manage the resulting complexity. This trend is amplified, considering the use of ECUs for fundamental applications such as engine control.

Since the 90's, the car manufacturers are moving from centralized, wire connected architectures to distributed architectures using shared communication protocols. This architecture decreases the number of connections required, thus saving weight and reducing the number of potential error sources. Additionally, the information is globally available, which enables a better apprehension of the environment and therefore better suited reactions. Moreover, the distribution of the computation among different ECUs allows for the development of fault-tolerant architectures, architectures that can deliver correct services even if parts of the system have failed. This attribute is strongly required for safety-critical operation to avoid catastrophe with the possible loss of human life [Kop97].

The network has been soon recognized as playing a central role in maintaining the system in a safe state [NSSLW05]. Hence, this resource represents a fundamental building block of the system, and even sporadic faults might have severe consequences to the higher layers relying on it. Parallel to that, this resource is exposed to a hostile environment (electric and electromagnetic fields, large thermal gradients, splashes from oils, petrol, chemicals...) and particularly subject to fail.

The time-triggered paradigm [KB03] has been introduced in this context to provide a highly dependable architecture for safety-critical applications. One important feature is to provide a predictable behavior, such that the different subscribers know precisely whether the expected services are delivered or if the module has failed (error detection). Moreover, this timed behavior can be further used for fault-tolerant architectures where services are provided redundantly and the results merged to improve the system robustness (fault tolerance).

The limitation of this approach, however, is the definition of the fault hypothesis. Hence, the system is designed to support a given fault model – service deviation defined by its effect (value domain) and its occurrence (time domain) – and assumes this fault model to represent a worst-case scenario for the given environment. A problem can happen when the current fault is outside of the fault hypothesis. This can occur as a consequence of a previous ECU failure or more generally due to fault accumulation in the system [SS03]. Hence, the superposition of several faults can lead to a fault pattern or occurrence outside of the specified hypothesis and thus to a system failure. Consequently, periodic tests of the system as well as detailed diagnosis are required to complement the fault-tolerant architecture and test the system robustness as well as to perform recovery actions and restore the system fault tolerance.

1.1 Contributions and Objectives

The motivation for this work is to present a remote, transparent test approach for testing the reliability of time-triggered communication systems as well as the availability of the associated error detection mechanisms. We focus our work on the network, since the low-level mechanisms involved with the information exchange between different nodes delivers fundamental services to the system. Hence, without message exchange, the distributed application can not work. Moreover, we position our work to late product life-cycle and suppose a highly integrated system, during normal operation, and without explicit test support. This requires our test approach to be *generic* (not rely on dedicated services), *non-intrusive* (no modification of the system), and *transparent* (to avoid deviation of normal service delivery, both in the value and in the time domain). This last attribute enables test operation to be performed concurrently to normal operation, and thus decreases the mean time between successive test operations, even if the system can not be halted for maintenance. This in turn minimizes the probability of fault accumulation.

The contributions of this work are the following:

- **Proposal for a remote, transparent test approach for time-triggered communication systems**: During this work, we analyze which diagnostic information is globally available (observable at the communication medium) and which information can be additionally gathered using a subtle stimulation of the communication services and without disturbing normal system operation.

- **Detailed analysis of FlexRay and TTP/C clock synchronization algorithms**: Our test approach relies on specific attributes of convergence-average based clock synchronization algorithms used in time-triggered communication systems. We present in this work a detailed analysis of these distributed algorithms and explore their characteristics as well as their limitations.

- **Proof for the transparency behavior of the test approach**: Since this test operation is expected to be performed concurrently to normal operation, we have to guarantee that the system is not jeopardized and that the robustness is not decreased (i.e. the system presents the same tolerance to faults).

- **Architecture for a dedicated tester node**: The approaches proposed in this work require a dedicated tester node with tight connections between the monitoring path and a real-time computer unit. The architecture developed within this work is presented.

- **Dedicated test application**: An important part of this work was the validation of our approach. We have developed a dedicated test application with the main features (a) to calibrate the oscillators between the nodes and (b) to make local information globally available. Notice that these services are not required for our test approach and were only needed for the validation of our method.

1.2 Structure of the Thesis

The thesis is organized as follows: Chapter 2 presents the state of the art concerning testing automotive electronics. It starts with a description of the evolution of automotive electronics during the last fifty years in order to point out the system model and current problems. During a second part, the challenges of testing distributed systems are discussed. We review the different aims for testing, which mainly depend on the current life-cycle of the product, as well as the state of the art. After that, the time-triggered architecture is presented as a solution for supporting highly dependable systems. The last part of this chapter is dedicated to the STEACS project and summarizes the results achieved so far with regard to testing automotive communication systems.

The aim of Chapter 3 is to describe the problem statement. The motivation for this work is presented and the term *transparent* is defined. Further, a survey about the diagnosis information which can be gathered using pure monitoring is provided. While having a good coverage for transmitter modules, the resources allocated for message reception are not observable. To complement pure monitoring, a transparent test approach for stimulating the clock synchronization mechanism and thus gaining remote insights in the nodes' reception status is introduced. This chapter concludes with the identification of three main challenges for the validation of our test approach.

Chapter 4 is dedicated to the validation of our approach. We start with a formal description of quartz oscillators as well as clock synchronization algorithms to provide a formal background for the further validation. Then, we prove that our clock synchronization stimulation method (deterministic replay operation) poses no threat to the system since the *precision* property is not violated. In fact, we propose a method to compute the system precision according to the replay attributes (slew rate). After that, we focus on the observability of the system and show that the accuracy of the global time base is bounded by the accuracy of the physical clocks. This attribute is required during the test approach since the tester emulates a node with a bad quality (virtual) physical clock. The reaction of the standard nodes (clock synchronization behavior) provides information about their reception status with regard to the tester frames and thus about the test result. Moreover, this chapter provides remote evaluation methods for attributes of the clock synchronization and concludes with an illustrative example summarizing the results achieved so far.

We present in Chapter 5 our experimental setup. It consists on a four nodes FlexRay cluster and a dedicated tester node. The standard nodes have been modified to implement a voltage controlled oscillator. Their frequency can be configured by the distributed application, and different quartz drift scenarios can be thus emulated. Moreover, internal nodes' information such as offset and rate correction as well as current oscillator configuration is made globally available. The dedicated tester node provides a FPGA where a measurement controller as well as a microcontroller core have been integrated to provide the required functionality. The interface with the host as well as the results which are automatically generated are also described. During the last part of this chapter, we present a test campaign for the FlexRay clock synchronization algorithm. We illustrate that both bounded quartz accuracy and quartz stability are required for correct operation of this complex mechanism.

Finally, Chapter 6 concludes this work and provides an outlook on what can be expected from future research in this area.

Chapter 2

About Testing Automotive Electronics

Science... never solves a problem without creating ten more.

George Bernard Shaw (1856 – 1950)

Electronics are present everywhere in today's car. Electrical window control, radio and keyless entry form the visible part of the iceberg. Most of the control functions are realized or at least supported nowadays by electronic control units. The aim of this chapter is to explain the current situation, present the current challenges, and to review the current steps towards testing embedded automotive electronic components.

2.1 Electronics in Cars

2.1.1 History

The aggressive competition that occurs in the automotive domain leads to a high demand for improvements by the car manufacturers. New safety, convenience, security and environmental features are being implemented in cars, and existing systems are continuously optimized. This strong requirement for innovation has been largely supported by electronics for the last 30 years. The main reason for that is the low cost due to the mass market and the high development rate. Hence, Moore correctly predicted in 1965 that the number of transistors on a chip would double every 18 months [Moo65], and this trend is still true.

This fast development not only shrinks the price for the sake of functionality, but enables the integration of new (complex) functionalities, too. For example, the number of transistors for a microcontroller in charge of power train control was increased by a factor 300 (from 20,000 to over 6 million) between 1982 and 1999 [Ban99]. Concerning the memory requirements, less than 10kB were required in 1980 for the Electronic Control Units (ECU) against more than 500kB in 2000 [Run98]. Nowadays, experts estimate that for a luxury car 20% to 30% of the manufacturing costs comes from electronics [LH02, Run98], and that one third of the development costs are spent for electric/electronics [WW02]. Furthermore, it is agreed that more than 80% of all automotive innovations stem from electronics [FP05, LH02, Mar03].

This fast increase of electronic components lead to a parallel increase on the connection amount. In the middle of the 50's, 45 meters of wiring [LH02] were required to connect the different electrical components. Today, the VW Phaeton for example requires about 4 km of wiring resulting in an additional weight of 64 kg [Vas04]. Until the beginnings of the 90's, the communication between sensors, controllers and actuators were realized by point-to-point wiring. This communication scheme, however, presents the following main drawbacks [LHD99, NSSLW05], [Zur05]-7:

- Decreased reliability – each additional link reduces the mean time of failure
- Low maintainability – difficulty in undergoing modification and repair [ALRL04]
- Low flexibility and modularity for future enhancements
- High weight leading to increased fuel consumption
- High costs: the information is inefficiently distributed by the discrete channels
- High complexity during manufacturing and assembly operations
- Shrinking layout space from the cabin

The car industry recognized the limitation of point-to-point wiring and developed in the late 80's and 90's different proprietary solutions for centralized or distributed networks (see [LHD99] for an overview). Presently, more than 70 ECUs exchanging up to 2500 signals [Han05, FP05] (elementary information such as sensor output) can be found in middle-class cars, and automotive applications are evolving into complex distributed systems. The use of networks not only increases car reliability and performance, it enables further cost reduction and the development of centralized control strategies that are more optimized than decentralized ones [Sav06]. For example, car stability can be improved in combining steering and braking information.

2.1.2 A Hostile Environment

The electronic equipment in a car is subject to an extremely hostile environment [LHD99, Nob92, ZP93]:

- Mechanical vibrations up to 30 G
- High temperature range from -40°C to +150°C (exhaust temperature might increase up to 650°C)
- Large thermal gradients (40°C/min) leading to thermal fatigue phenomena
- Splashes from oil, petrol, water or ice
- High humidity (up to 99 percent), dust, sand storms, corrosive chemicals
- Conducted emissions (due to, e.g., commutation of electric motors) leading to positive over-voltages superimposed on the 12V power supply. Their amplitudes might be up to 120V and usually last 40ns to 400ms
- Radiated emissions leading to electromagnetic fields (electric power lines, broadcast radio or TV transmitters, radars, mobile transmitters – mobile radio, cellular phones) that can can exceed 200V/m; for comparison, domestic is about 3V/m and industrial 10V/m

This hostile environment is a source of transient faults and can prohibit the use of defined technologies (e.g. hard disks due to mechanical vibrations). Moreover, the fast increase of electronics in this difficult environment has led to a parallel increase of the fault rate due to electric and electronic components. Hence, about 20% of the faults were related to electric and electronic equipment in 1989 [ADA89], which represents a Mean Time To Failure (MTTF) of $5*10^6$ hours. This rate has evolved to 38% in 2006 [ADA06]. According to [SMM00] and [GS91], 30% (36%) of electrical failures are attributed to connection problems in the automotive domain (respectively aircraft domain). Concerning vehicle safety, the aim identified by the automotive industry is to have less than one incidence per year and per million cars. Considering that the car is operating 5% of the calendar time (i.e. 500h per year), this requirement equals a failure rate of 10^{-9}/hour [Kop99], which corresponds to a SIL4 conformance [IEC98].

2.1.3 Actual Situation and Challenges

Application domain

Kopetz [Kop99] defines an automobile as *"a complex mass product that is composed of a number of sophisticated subsystems"*. The application domain for electronics ranges from powertrain, chassis, security, telematics, driver interface, to body and comfort [FP05]. These different application domains have hugely varying constraints of bandwidth, cost and safety. More generally, it is usual to distinguish between system electronics and body electronics.

System Electronics, on one side, regroups powertrain and chassis and is concerned with the services related to the car movement. Such functions, while making use of development in the computer data network area, are control oriented and thus driven by control strategies. The control loop periods are typically in the range of some milliseconds [Kop99, NSSLW05]. Moreover, the latency jitter between sampling of the sensors and update of the actuators has to be tightly controlled. Such systems are typically safety-critical, since a failure might lead to a catastrophe with the possible loss of human life [Kop97]. For example, if steering is delayed because of a system failure, then car passengers are threatened. Consequently, the focus is set to the development of safe modules and mastering the interferences between functions.

Body electronics, on the other side, regroups the services not directly related to the movement of the car. It consists of security (against theft), telematics (for audio and video in cars), driver interface (information display), and comfort. An example of body electronics is the introduction of local area networks in the four doors of a BMW that reduced the weight by 15 kg while enhancing functionality [LH02]. While the emphasis for telematics services are set to bandwidth, (soft) real-time and confidentiality, the requirements for comforts electronics are much lower since the response time is determined by the reaction time of the human operator (range of 100msec to 200msec). Furthermore, body electronics are usually not safety-critical and a safe state is usually available (termination of current operation) [Kop99]. Indeed, the abnormal termination of an audio or video system due to a failure will (fortunately) not lead to a safety hazard.

The diversification of requirements combined with low-cost constraints have made today's vehicles to concurrently implement different types of networks, see Figure 2.1.

The need for standardization

As highlighted previously, electronics in cars are becoming more and more complex. In order to cope with this complexity and to create high quality products, the automotive industry is moving from proprietary solutions to standardized architectures. This presents, among other things, the following main advantages:

- lower costs for development and maintenance
- better quality, since a module has a much larger target audience and problems can be fixed earlier

- interchangeability between the suppliers and therefore more competition for better end products
- Concentration on the core business, i.e. application development, which is valuable to the customer

The standardization trend is taking place at different levels. At the communication level, the Society of Automotive Engineers[1] has classified networks into four types [SAE92], see Table 2.1. Alongside this, de-facto industry standards such as CAN [Law97], LIN [LIN03], MOST [MOS05] or FlexRay [MT06] communication protocols are being introduced or already prevail. A state of the art summary about industrial communication technology is provided in [Zur05].

network classification	speed	application
Class A	<10kbits/s low speed	convenience features
Class B	10-125 kbits/s medium speed	general information transfer
Class C	125 kbits/s - 1Mbits/s high speed	real-time control
Class D	>1Mbits/s	multimedia applications

Table 2.1: Classification of automotive networks

The Operating System (OS) is also being standardized. OSEK/VDX was introduced in 1998 [Joh98, ZP98] to provide a standardized operating system. Main features are the economic usage of memory resources and CPU performance while providing very fast response time and the capacity to manage task activation with short periods (>1KHz) for quick control loops. A state of the art summary concerning the implementation can be found in [BBFT06]. Due to the challenges facing the development of safe automotive software (reported in [KTWE03]), standardization moved one step forward and an industrial consortium[2] proposed the AUTomotive Open System ARchitecture (AUTOSAR) [HSF$^+$04, FBH$^+$06]. The experience of introducing reference architectures into the automotive development process is further described in [EOAG$^+$05], thus illustrating the importance of standardization at different levels.

2.1.4 System Model

As stated previously, car electronics are evolving into complex distributed systems. Hence, several networks can co-exist, implementing different communication protocols. Each single network is building a **cluster** as illustrated in Figure 2.1. A (computational) cluster is defined in [KN97] as *"a set of nodes that cooperate to perform*

[1]http://www.sae.org
[2]http://www.autosar.org

the intended service for the cluster environment". The different networks are interconnected using *gateway-nodes*, whose aim is to filter the incoming messages and transmit the selected ones to the other network. The clusters further consist of **nodes** – *"self-contained computer with their own hardware and software [KN97]"*– on which the distributed application is computed. The nodes typically present a standardized architecture comprising:

- the *physical layer*, which is in charge of physically accessing the communication medium and digitizing the data for further processing.
- the *communication controller*, which builds the interface between the high-level view required by the host and the serialized bit stream required by the physical layer. This part is moreover responsible for managing the access on the communication medium to avoid collision and guarantee correct message transmission.
- the *operating system*, whose aim is to provide standardized, high level functionality for the efficient application development.
- the *application*, which computes the local part of the distributed application. This application might be connected to sensors and/or actuators and build an interface to its environment.

The set of communication controllers plus the physical layer within a cluster are building the **communication system** [KN97].

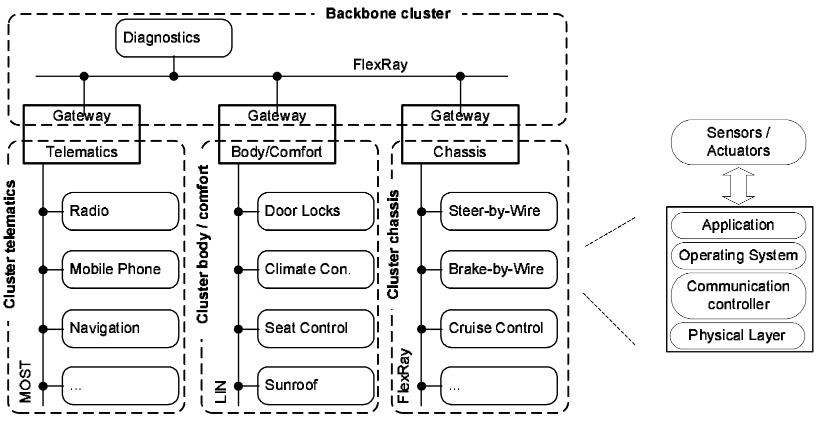


Figure 2.1: Example of an automotive network

2.2 Testing Distributed Systems

We have seen in Section 2.1 that automotive electronics are organized into distributed systems. We will now review the challenges and present a survey of the existing test approaches in order to improve the quality of the system.

2.2.1 Concepts and Definitions

Taxonomy

Avižienis et al [ALRL04] define correct service delivery as the correct implementation of the system function. A **failure**, on the contrary, occurs when the delivered service deviates from the correct service. A failure is caused by an **error**, which is a deviation of the system from the correct state. The root of an error is a **fault**. The fault is said to be *active* when it activates an error, otherwise it is *dormant*. The time interval between the fault occurrence and its activation is called *dormancy*. The time interval between error and failure is called *latency*. Figure 2.2 illustrates the dependencies.

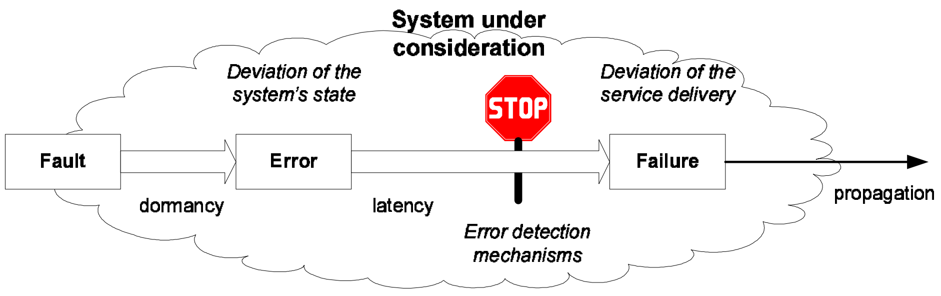

System under consideration
Deviation of the system's state
Deviation of the service delivery
STOP
Fault
Error
Failure
dormancy
latency
propagation
Error detection mechanisms

Figure 2.2: Fault, error, failure

The *total state* of a system is defined in [ALRL04] as the set of states comprising computation, communication, stored information, interconnection and physical condition. According to the previous definition, a fault is the root of a (microscopic or macroscopic) system deviation (error). However, this error does not necessarily lead to a system failure. Take the example of heavy ions hitting a chip. The heavy ions will change the electron distribution in part of the chip, thus changing the physical state of this chip. However, this new electron distribution might still be within the allowed bounds and thus does not lead to glitches (i.e. error at a higher abstraction level). If the heavy ions are leading to a glitch, there is still a non-zero probability that the resulting error (here a deviation of the computation state) is not latched into a memory element and disappears before appearing to the output pins. Following the same reasoning, even a latched glitch (stored information state) might not lead to a system failure if this bit is not used (or overruled) for the output computation.

To summarize, a fault *per definition* does lead to an error, but this error does not necessarily lead to a system failure. It is in fact the goal of fault tolerance mechanisms to act as a "shield" [ACL95] and prevent errors from leading to system failures. These mechanisms are designed according to a given fault hypothesis (description of the fault model in the value domain – e.g., number of ions hitting the chip or strength of the EMC field – and in the time domain – e.g., transient, burst, stuck-at). The coverage of this shield has a direct impact on the reliability of the system.

The operation of the error detection mechanisms is more complex because of two points. First, the ability for a fault to "hide" (dormant state) and activate at any arbitrary point in time can lead to *fault accumulation*. This effect might lead to fault patterns or occurrences outside of the fault hypothesis as defined during the design of the system, and can represent a threat for the system. Secondly, in the case of complex system, a failure in one module can *propagate* to another module and lead to a fault for this other module.

Improving the dependability

The concept of **dependability** is defined in [ALRL04] as *"the ability to deliver a service that can justifiably be trusted"*. It regroups the notions of:

- *availability*: readiness for correct service
- *reliability*: continuity of correct service
- *safety*: absence of catastrophic consequences on the user(s) and the environment
- *integrity*: absence of improper system alterations
- *maintainability*: ability to undergo modifications and repairs

There are different possibilities for improving the system's dependability. During system design, *fault tolerant* architectures based on error detection and system recovery can be proposed. During system development, *fault prevention* techniques (e.g. design rules) can be used. *Fault removal*, which consists of the verification, diagnosis and correction of the system, can be applied either during system development or during system operation. The verification can be static (e.g. data flow analysis, formal verification [SL95]) or dynamic (e.g. testing) when the system is exercised. Fault removal during system operation is defined as corrective or preventive maintenance. Finally, *fault forecasting* is performed during system evaluation to forecast (qualitatively or quantitatively) the scenarios that can lead to a system failure. More information can be found in [ALRL04]. In practice, testing is frequently used because of the representativeness of the results – the system is exercised – and because of its fast execution time.

2.2.2 Purposes of Testing

According to [Het88], *"testing is any activity aimed at evaluating an attribute or capability of a program or system and determining that it meets its required results"*. Depending on the current stage in the life cycle of the system, this common definition can be applied to testing a system with different goals in mind.

Verification is a common test procedure during system development that involves checking whether the system adheres to given properties [ALRL04]. It aims at revealing implementation faults, and the tests are focused on the investigation of a few mechanisms at a time. Typically, at this early point of development a complete target environment is often not available. Models or early prototypes of the system are used, and a suitable emulation of the target environment is required (sensor inputs, e.g.).

Conformance testing or **validation** [IEE90] represents the process of evaluating whether the parameters of an implemented system conform to a specification or a standard. At this point, correct system operation must be validated for the entire range of possible inputs and parameters. This test is usually executed with some kind of prototype system. In practice, however, these tests are restricted to a subset of relevant configurations, since an exhaustive test of all possible configurations is not feasible.

Robustness testing is defined in [MS97] as the characterization of the system behavior in the presence of erroneous and/or stressful input conditions. In the context of hardware, we have to extend this definition to stressful operating conditions. Prior to test execution, an appropriate fault hypothesis must be defined. Ideally, all possible inputs and operating conditions (including illegal ones) should be tested; again, however, only a subset will be feasible in practice. Therefore, fault-injection techniques are used either (a) to execute and check the implemented error detection and handling mechanisms, or (b) to derive a forecast for the systems robustness in field operation.

According to [MRW03], the purpose of **interoperability testing** is to prove end-to-end functionality between (at least) two fully assembled systems according to the standard(s) on which those systems are based. Typically, nodes from different vendors are integrated into a distributed system in order to test whether these nodes are able to communicate under a set of possible configurations.

Performance testing, or **system evaluation**, is used to assess the performance of a given system by means of metrics that can be used to compare the different implementations using a benchmark suite. Its main focus is on measuring the performance of a system for a well defined set of tasks [AKM$^+$01], and not to verify or validate a system. In contrast to verification, efforts are set to the standardization of the test procedure and the tests are typically run on an early product implementation.

Finally, the goal of **maintenance testing** is to detect and localize faults that emerge during the mission phase of a system. The scope is typically narrowed to (physical) faults that occurred since the last maintenance actions (e.g., due to ageing effects) [ALRL04].

Obviously these different testing aims span a large scope. During protocol and application development, experienced engineers use the test tool in a lab environment for debugging, and flexibility is the key issue. In context with maintenance testing, on the other hand, repair personnel with comparatively low protocol expertise are using the test tool in a rough environment with limited access, and for that reason standardization and ease of use are most crucial. Verification, conformance, interoperability, and maintenance tests are qualitative tests with the purpose of proving whether an assumption is correct or wrong, whereas robustness and performance tests are quantitative tests that aim at deriving a numerical characterization for a given attribute. This list does not claim to be exhaustive but aims at illustrating the diversity of testing with respect to the goal targeted, methods required and fault model applied.

2.2.3 The Challenges of Testing Safety-Critical Distributed Real-Time Systems

Distributed systems consist of a set of nodes that are running their own program in parallel (see Section 2.1.4). Testing distributed systems is therefore more complex than testing sequential ones because of the **state explosion** due to the concurrent process operations. In fact, the test scope is not merely limited to the different single processes but is enlarged to the (complex) process of interaction as well. For example, in a system that reacts to its environment, each new input combination might lead to a new program execution flow and a new order of task invocation.

Another problem is the **precise ordering of events** occurring in distinct, concurrently executing processes. Hence, because of the inherent geographical diversity of distributed system, means to precisely and globally time-stamp the events monitored or the stimulus to inject are required.

The test complexity is further increased by the **non-reproducible behavior** of distributed systems according to their inputs [MH89]. In fact, *races* are taking place between the different processes, and their outcome may depend on non-deterministic elements (e.g. CPU load, non-determinism in the communication protocol or non-deterministic statements in the application software [Sch94b]). Slight changes in one of these factors may change the outcome of such a race, which in turn leads to a totally different system behavior. The *probe effect* [Gai86], or *Heisenberg uncertainty [LMC87]*, which typically influences the existing races while improving system's observability, should be thus minimized. The sparse time base introduced by the time-triggered architecture solves this problem, see Section 2.3.2.

The presence of **timing constraints** makes the situation even worse. *Correctness*, in the context of real-time systems, not only depends on the logical results of the computations but also on the time when these results are produced [Kop97]. Deadlines can be *soft* if the result of computation is even of utility after the deadline has occurred (e.g. audio, video). Otherwise, a deadline is *firm* when the results can not be used after the deadline. In case of safety-critical systems, a deadline is *hard* if a catastrophe can occur when a deadline is missed. Consequently, the probe effect is much more stringent in real-time systems since the system's behavior in value *and* time domain should be kept unmodified.

Automotive electronics are further characterized by their **high degree of integration** due to their embedded nature, intensified by the aggressive costs and performance metrics that the automobile industry demands [Mar03]. The system's architecture is tailored to its requirements and consequently little or no test support or interface is available [Sch94b]. However, testing requires system changes and additional resources, which is incompatible with the automobile requirements and increases the unwanted probe effect.

Additionally, test operations are by nature intrusive, since test vectors are applied to the system. The stimulus application might not be compliant with normal system operation and **requires service interruption**. In case of safety-critical services, which can not be interrupted, this can constrain the test execution to time windows when the service is not required and thus decrease test coverage, and additionally reduce the environment's representativeness. A representative environment is all the more important to validate a system for a given context, since exhaustive tests (for every possible environment) are not feasible due to the system's complexity.

2.2.4 Monitoring Distributed Systems

The quality of a test result (in terms of test speed and coverage) attainable with given efforts strongly depends on the testability of the system under test. According to [BMS87] the two key aspects for testability are controllability and observability. Controllability is an indicator of how easy it is to bring the system or node under test to a given state. Observability is a measure of how easy it is to observe certain activities of the system or node under test. Both measures are largely determined by the accessibility of the system under test. In short, good accessibility leads to good testability.

Monitoring operation aims at improving the system's observability. It is defined in [Sch94a] as *"the process of extracting and gathering information regarding the behavior of a particular system"*. The definition given by Tsai et al [TBYS96] *"collect run-time information on the target system that can not be obtained by merely studying the program text"* insists on the online nature of monitoring operations. Application domains range from testing and debugging, dynamic safety checking to performance analysis and program optimization [Sch95, TBYS96]. Schmid [Sch94a] insists, moreover, on applying monitoring to improve the environmental model. In this case, monitoring provides a kind of loop-back during system development [PS99], which is used for tailoring a system to its environment. This loop-back can be implemented during field operation as well as input for recovery actions (*"Feedback Monitoring"*, [Sma04]).

There are two main approaches to implement monitoring services: software and hardware monitoring. First, during **software monitoring**, specific services are added in the software. Thane [Tha00] identifies four different approaches:

- *Probe-tasks*: Dedicated tasks are developed to monitor the status of the system. In [KHM05], the diagnosis tasks are automatically distributed over the nodes to maximize the system's reliability while minimizing resource requirement. As for the Inline probes, this method requires modifications within the node's software and / or the system architecture.
- *Kernel-probes*: the operating system provides means to monitor and log task activation, preemption, termination, interrupts and system calls. Examples for this approach are described in [DR92, Mah01]
- *Inline-probes*: the developer explicitly inserts a debug command (typically some kind of *"printf"* function) to return the actual state of the program. The approach proposed in [MSSP02] presents the advantage to split the program source from the debug commands and provides automated ways to generate the executable, and thus avoids the risks of screwing things up by accident. The use of a preprocessor switch is also a common method to optionally add debug comments.
- *Probe-nodes*: dedicated nodes are connected to the system to monitor the bus traffic. This approach can not be considered as a purely software-oriented method, since additional hardware (to get access on the communication medium) is required. This method will be discussed later.

The main problems of software monitoring are the probe effect – the system has to be modified – and the resource overhead. Notice one possibility for avoiding the probe effect is to leave the monitoring services in the system permanently. This implies that enough resources are available, which is not always the case in embedded systems.

The second main direction to improve the observability of the system is called **hardware monitoring**. This approach requires a (dedicated) hardware to collect the data [TBYS96], and thus presents the main advantage of minimizing the interference with the target system. Typically, the monitoring hardware is connected to the communication system from the node's CPU and monitors each memory transfer [BV89, FAF06a, TFCB90]. This approach uses architectural partitioning and concentrates on inter-unit communication. The main drawback is the low abstraction level (electrical signal) of the data being monitored.

The *probe-node* approach presented previously is an enhancement of hardware monitoring for distributed systems. Here, the communication system is external and usually a serial one, and the units are self-contained computer systems. Depending on the abstraction level under consideration, this approach might be classified either as a hardware- or software-monitoring method. In the case of a single node, it would represent a hardware monitoring method; The node is then considered as a black box, and the inputs / outputs are monitored. On the other hand, this approach can be classified as software monitoring for the whole cluster, since changes in the system architecture might be required. Additionally, the scope is then the distributed application and the additional node can then interpret the messages exchanged by the other nodes. Their main advantage is to enable remote monitoring: information about the node's input and outputs can be obtained without having to modify the node itself. However, as in a black box test, this method does not provide insights within the node. Examples for this approach are [DJM95, DGMK00, HP03, Sma04, ZS02]. Notice that according to Olsson et al [OBRB01], this method of monitoring at the system level (inter node communication) is more efficient and effective for testing distributed systems than monitoring at the node level (intra node communication).

Hybrid monitoring takes advantage of both hardware and software monitoring. In Hybrid monitoring, the triggering (more complex) is usually accomplished in software and the recording (easier to automate) in hardware [TBYS96]. Perturbation of the monitored system is greatly reduced, while high level data can be monitored. This approach is extremely welcome in highly integrated systems such as System on Chip. Examples for hybrid monitoring are [ESL01, KBG98, Tho05].

2.2.5 Improving the System's Controllability

Controllability is the complement of observability. While the main focus of observability is to provide information about the system's status, the aim of controllability is to move the system under test to a desired state. For that, a *workload generator* and *fault injection* methods are required to drive the system through its regular states and exercise the exceptional states [FAF06b]. The main challenges are (a) *reachability*, to reach each state of the system under test, and (b) *reproducibility*, to deterministically apply and reproduce a test scenario.

Different approaches can be found in the literature. Perhaps the most common one is to directly control the inputs of the system under test to generate the workload [Sch91, TH99] (*black-box* test [IEE90]). This requires an **environment simulator** [Sch92] when the test environment is expected to react according to the system's response. Otherwise, the test scenario can be generated off-line and statically applied to the system. One advantage of using a simulated environment rather than that of the real test environment is [Sch94b] *safety and/or cost*: Correct operation can not be guaranteed at this point, however, since confidence in the system being developed is still low or functionality might be lacking. Another advantage is *flexibility*: The tester generally has better control over a simulated environment than over a real one. It results in easier test execution, and enables system evaluation under invalid environment. However, the test *representativeness* (or *representativity* [Sch94b]) is usually a challenge. How can the tester be sure (a) that its test vectors are representative for the given environment, and (b) that every (realistic) scenario has been covered by its test campaign? This approach is non-intrusive for the system under test. Neither the nodes composing the system nor the system architecture has to be modified. However, it implies that at least a description of the system (for a simulator) or a prototype is available (unlike model-based testing that only requires an algorithm but no implementation of it, and therefore can be performed earlier).

A more intrusive approach for workload generation is the **replay of event histories** [MH89]. It uses the node's internal bus architecture to replay recorded interactions between the processing unit and the node's externals (memory, I/O...) [TFCB90, TH00]. One advantage of this approach is the reproducibility of the test execution. To compare, the reproduction of a human controlled test case can not be performed with the same exactitude in the time domain. The small time differences might influence the existing race condition (probe effect) and make the system react differently. Replay operation thus strongly supports cyclic debugging, when the test scenario is applied recurrently until the problem is corrected. Another advantage of this approach is to reduce interactions between the node under test and the external world to memory accesses. In such a context, the simulation of the environment is much easier, since it is limited to the interface of the processing unit. For example, in case of a temperature sensor, the room temperature does not require to be controlled anymore but only the sensor output.

This approach has been extended to the serial network of distributed embedded systems in [Arm06]. The benefit of this extension is to enlarge the test scope to several nodes at the same time. The replay of event histories is an off-line method, in the sense that the stimulus is applied statically and no interaction with the system under test is possible. In the case of distributed systems, cluster simulation [Gal99] can be used to interact with the system under test while still controlling the communication medium (the answer of the tester is computed online according to the reaction of the system). Notice that these approaches are more intrusive than the use of an environment simulator. While the nodes composing the system are left unchanged, the system architecture has to be modified. In return, the replay or simulation of a group of nodes enables the verification and validation of single nodes or another group of nodes before integration in a whole cluster.

Another method by which to improve controllability is the use of **fault injection** techniques. Contrary to the two previous approaches, this one is not directed toward the generation of a specific workload but rather in the activation of exceptional paths. While control of the system inputs aims at exploring the system states for correct behavior (testing for correctness, e.g. verification, validation), fault injections methods are more directed toward dependability evaluation. More especially, fault injection requires a workload and is performed on top with two main goals [ACL95]. First, *fault* *removal* aims at checking the error detection and fault tolerance mechanisms. Primary concerns are determinism and reproducibility to activate specific fault sets which are identified during design specification. The second goal, *fault forecasting*, aims at rating the efficiency of the test procedures or the attributes (e.g. coverage, error latency) of the error detection mechanisms. The focus is then set to the estimation of a representative statistical distribution of the fault space, and to a large number of experiments so as to be confident of the measure. Four main approaches are usually referenced in the literature [BP03]:

- *Hardware*-based methods use dedicated hardware to allow the injection of faults into the target. There exist two types of hardware-based fault injection methods: *Forcing techniques* and *Insertion techniques* [BP03]. Using forcing techniques, the fault injector forces a logical level at the selected points (e.g. MESSALINE [AAA+90], AFIT [BG03], using heavy-ions [KLD+94] or laser fault injection [SMF98]). On the contrary, insertion techniques use a special device to replace part of the circuit and inject the required faults (e.g. FIDYCO [RSH04]).

- *Software*-based fault injection (SWIFI) consists of changing the content of memory or registers based on specified fault models to emulate the consequences of hardware faults (e.g. [ACK+03], DOCTOR [HSR95], XCEPTION [CMS98], GOOFI [AVFK01]) or to inject software faults either in the operating system (e.g. MACH [DJM95], DOCTOR [HSR95], MAFALDA-RT [RAA02], [Ade03]) or in the distributed application (e.g. LOKI [CLJ+04]).

- *Simulation*-based methods are using a simulation model of the system (e.g. in Hardware Description Language) into which the faults are injected. The main advantages are high controllability and observability of the target. However, they are typically time (and resource) consuming, and depend on the accuracy of the simulation- and fault- model used. Example are MEPHISTO [JAR+94] and VERIFY [STB97].

- *Hybrid* approaches are combining two or more fault injection techniques to combine the advantages of the different methods (e.g. LIVE [AIMP97], NF-TAPE [SFB+00]).

A major challenge in testing a system is the creation of representative workload and test vectors. Hence, the use of inaccurate fault model not only increase validation effort (the system would be tolerant to faults not present during field operation) but might reduce the resulting dependability, too, since real fault models are not covered. **Online Test** aims at overcoming this problem and enables test operation during field operation. In the works described in [DJ94, DDS02, SFB+00], the test vectors are directly applied to the distributed system. To that aim, a framework to connect *lightweight fault injectors* is presented in [SFB$^+$00]. The focus is set to the interfacing of fault injection modules to enable the efficient enhancement of the test platform. In [DJ94], dedicated layers are locally inserted up to or below the layer under test. Higher test layers are focusing on the creation of new messages, while lower test layers aim at manipulating messages generated by other participants in the protocol. Both methods are highly intrusive since they require changes within the targeted node. The approach described in [DDS02] is more oriented toward the control of the environment for the validation of the system, and the changes are localized to the system's interface. Contrary to the previous works, the test scope in [SV02] is focused to the communication controller hardware instead of the whole system. During this work, local off-line and online [Nic96] BIST methods are investigated. In [ST99], the system produces its own stimulus through normal operation and the test coverage is analyzed. This online test approach is very interesting, since it is totally transparent and concurrent to normal system operation. However, the test coverage is not deterministic (it depends on system operation) and some paths (e.g. error detection mechanisms) have a low probability to be exercised.

Another important focus of online testing is the detection of dormant faults [SS03]. Since fault-tolerance is usually based on the single-fault assumption, the accumulation of faults might lead to a system failure. The periodic (online) test of the system reduces the error detection latency and decreases the probability of multiple faults within the system.

2.3 The Time-Triggered Architecture

Throughout the previous sections, we have seen (a) the organization of automotive electronics to be made of complex distributed real-time systems, and have discussed (b) the challenges of testing these systems and providing the required system quality. The time-triggered architecture [KB03] has been introduced in this context to provide a reliable architecture for safety-critical systems. We will review in this section the requirements and the concepts for this new architecture, as well as the main components such as the communication protocol and the middleware.

2.3.1 Requirements for Safety-Critical Applications

A safety-critical system is a system where a failure might lead to a catastrophe with the possible loss of human life [Kop97]. A typical example for that are "break-" or "steer-by-wire" systems, where the mechanic or hydraulic systems are replaced by electric ones. The following requirements have been listed in [WNSSL04, XbWP98]: (a) a system failure should not jeopardize human life, economics or environment. Moreover, (b) a single component failure should not lead to a failure of the whole system, and finally,

(c) the system is able to tolerate one major critical fault without losing functionality for a time long enough so that there is enough time to reach a safe parking area.

"X-by-wire" systems are real-time distributed systems implementing complex multi-variable control laws and delivering real-time service [WNSSL04]. We will now detail the requirements to support these kinds of safety-critical operations.

- **low and deterministic communication jitter** to enable the design of predictable control applications which have a fast response time for high speed control loops [KS97]. Additionally, the *recovery time* (outage time) must be kept low ($<$ 50ms according to [HT98]).
- **independent node development** to cope with the system complexity and enable the independent development of different nodes by different groups. This attribute is based on the precise specification of the node both in the time and value domain, and on an abstract model of the node services. Only then can the system designer know which function can be expected from the node, and which information is required for correct operation [KO02].
- **stability of prior services** means that *"the validated services of a node – both in the value domain and in the time domain – is not refuted by the integration of the node into an encompassing system-of-systems"* [KO02]. This requirement aims at enabling sequential node development and making the integration of different modules into a single computer unit easier. It reduces efforts significantly, since a module requires to be validated only once and not after every development stage.
- **constructive integration** to ensure that the integration of the *n+1* node will not disturb the operation of the *n* nodes already integrated [KB03]. While the *stability* requirement is focused to the design at the node level, this requirement has implication for the management of the network resources. Hence, the timing constraints (or timeliness) of the application must be satisfied even at the critical instant (i.e. when all nodes request the network resource at the same instant). It aims at saving time during system integration by avoiding sporadic failures due to the integration of an additional node in the system.
- **fault tolerance** and redundancy – the replication of a function – is strongly required in safety-critical systems in order to improve the system dependability and avoid that a single fault leads to a system failure [KS97]. At the physical layer a redundant communication channel might be used. From the communication services view, *babbling idiots* (one faulty node monopolizing the bus) must be avoided, for example by the use of bus guardians [Tem99]. From the application view, the realization of active replication demands mechanisms such as e.g. replica coordination, voting and internal state alignment [Bau01, KB03].

- the availability of a **global time base** with a precision under the microsecond range is required to synchronize the distributed control loops [KS97].

2.3.2 The Time-Triggered Computational Model

An **event-triggered** architecture is characterized by the fact that all system activities are initiated by an event [Kop97] and consequently *reacts* to its environment (an operation is started as soon as the event is received, regardless the current processing status). On the contrary, in the **time-triggered** architecture, every action is derived solely from the progression of real-time and thus *follows* the progression of its environment (an operation is started at a pre-defined starting point and processes the information that has occurred since the last computation; conflicts about processing resources are avoided per construction).

The time-triggered computational model is based on the representation of the controlled system as a **Real-time Entity** (RT entity), which represents the system as *"a subset of significant state variables"* [Kop98]. This RT entity can be observed at a particular point in time. The observation is then a **Real-Time Image** (RT image) – a current picture of a RT entity that is *"an accurate representation of the RT entity, both in the value and the time domains"* [Kop98]. Each node is provided with a RT image of the controlled system, which is processed locally according to a static, a-priori defined schedule. This deterministic progression of the system enables task multiplexing in the time domain in order to avoid conflicts and race conditions. This attribute, applied at the system level, then supports independent node development and the stability of prior service requirements, and when applied at the communication level, it helps support the requirement of constructive integration.

Furthermore, the static schedule is based on the concept of **sparse time base** [Kop92]. Because of the finite precision of synchronization algorithms and the effects of digitalization of time, it is usually impossible to consistently order events on the basis of their global time stamp. For example, an event occurring at time $t_{i+\epsilon}$ might be assigned different time stamps t_i and t_{i+1} by two clocks which are not perfectly synchronized. This problem is solved by the introduction of a sparse time base, which divides the time progression in an infinite sequence of activity and silence periods with a granularity larger than the clock synchronization precision, see Fig 2.3.

Using this model, the different clocks are re-synchronized during the *silence intervals* and agree on a global time base during the *activity intervals.* Events within the sphere of control of the system are allowed to occur only during an activity interval. External events have to be synchronized with the time base. Using this time model, the events are either simultaneous (e.g. Fig. 2.3, Events 2 and 3) or totally ordered within the system (e.g. events 1 and 2).

The synchronization of the system progression and the alignment of the communication with this sparse time base provides the following properties [Kop92]:

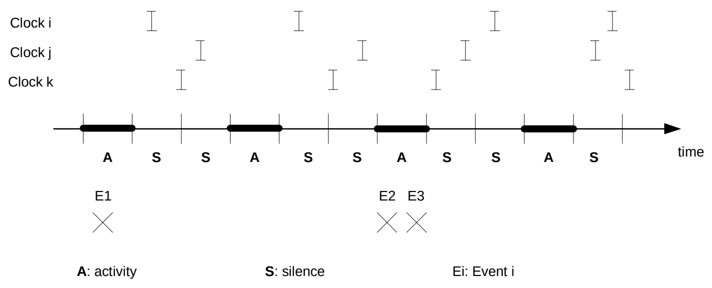


Figure 2.3: Sparse time model

- *consistent order property*, meaning that all nodes act in the same order on different observations and thus avoid state divergence. This can be guaranteed with this computational model, since all nodes receive all messages before the next one is sent.
- *simultaneity property*, the nodes act at about the same time on the same observation. This aims at keeping temporal coordination between the nodes and avoiding unsynchronized behavior of the system. Within the system, all nodes are acting on the messages at about the same time (defined by the precision of the clock synchronization).
- *temporal order property*, so that the nodes react on different observations in the temporal order of their occurrence. Notice that the temporal order property implies the consistent order property. Using this scheme, the messages are either simultaneous (within the same activity interval) or totally ordered.

Additionally, this concept improves the testability of the system [Sch91] since the size of the input space (in the time domain) is significantly reduced. Furthermore, the use of a sparse time base solves the problem of temporal order and simultaneity of actions without agreement protocols. It is argued in [Kop92] that the responsiveness of the system is actually improved, since the time units resulting from the granularity of the time base are shorter than the execution of an agreement protocol would take.

The sparse time base is also required for service replication to tolerate faults within the system. A **fault tolerant unit** [Kop97] consists of *"a set of replicated nodes that are intended to produce [...] the same results at approximately the same time"*. Their role is two-fold [WNSSL04]: they make the system resilient (a) to transmission errors (since the computation result is transmitted more than once) and (b) to measurement and computation errors (occurring before transmission). For that, they require [Kop97]:

- *Agreement on inputs*: all the nodes forming the fault tolerant unit must agree on the inputs in order to process the same data

- *Agreement on computation time*: The point in time when the results are available should be deterministic: the control structure should be static (to avoid dynamic task preemption and unpredictable task interference) and the message reception time should be known. The simultaneity property of the sparse time-base supports this requirement.
- *Deterministic algorithms*: Each algorithm should provide the same results.

2.3.3 Event- versus Time-Triggered Architecture

A lot of comparisons between event- and time-triggered architectures have already been published (e.g., [Alb04, GB06, SB06, APF02, NSL05] for some recent ones) without having clearly identified the "best" solution. In fact, the two architectures focus on different properties. Event-triggered architectures provide *flexibility* and try to improve the *overall performance* while the focus is set to *timeliness* and *worst-case execution time* for the time-triggered architecture [APF02, SB06].

Most of the communication protocols belong to the class of event-triggered architectures. These architectures, however, present some limitations with respect to the requirements presented in Section 2.3.1 for the development of safety-critical systems:

- the **system complexity** increases more than linear with the system size (number of elements and intensity of the interaction) [KBE+95]. Structuring – the description of a system at an abstract level – is required to cope with the complexity. Two methods are listed in [Kop98]: horizontal structuring (or *layering*) and vertical structuring (*partitioning*). Layering, on one hand, is related to the representation of the system at different abstraction levels and can be used both in event- and time-triggered systems. Partitioning, on the other hand, splits a system into a number of nearly independent subsystems with their own resources and *well-specified interfaces*, both in the temporal and value domain [Kop98]. This concept requires the system to be **composable** [KO02, Alb04] and adhere to the four principles previously explained:

  1. *independent development of nodes* at the architecture level.
  2. *stability of prior services* at the node level.
  3. *constructive integration* of the communication system.
  4. *replica determinism* is required for redundant nodes to provide the same external visible state at approximately the same time and to build a fault tolerant unit.

  Event-triggered architectures are not composable, since the temporal behavior of the communication system depends on the application software [KO02]. The performability of the communication system can be affected by adding nodes, since the critical delay (when all nodes require the network at the same instant) increases with the number of nodes. Moreover, the addition of nodes might cause queue overflow within the receivers, thus affecting the system and the stability of prior services.

- **robustness to their environment**: Time-triggered architectures are not driven by interrupts outside their sphere of control, but instead decide autonomously when to observe their environment. Consequently, and contrary to event-triggered architectures, there is no possibility for a malicious device to upset a time-triggered system [KBE$^+$95].
- **fault containment**: The time-triggered architecture provides an interface free of temporal control signals, (*temporal firewall*, see Section 2.3.4) thus providing error containment regions within the system. This attribute increases the overall system's dependability since errors are contained and do not lead to a system failure.
- time-triggered architectures are based on a static scheduling and on the off-line analysis of each module's execution time within the system. The static schedule simplifies the **inter-task synchronization** in resolving data dependencies of the temporal control structures, hence solving inter-task synchronization and avoiding race conditions. Additionally, timing analysis enables the **validation of time constraints** required for (hard) real-time systems. The work presented in [Ebn98] illustrates the difficulty to validate time constraints in event-triggered architectures and the improvements achieved with the time-triggered architecture.
- the periodic and a-priori defined task execution can be used for **fast fault detection** and the message transmission can be used as "heartbeat" to detect failed node [NSL05].
- **deterministic communication** with guaranteed worse-case transmission time and low jitters.

One important advantage of event-triggered architectures is that **fewer assumptions** are required to build a system [GB06]. Adding a node into a system does not require any change in the other nodes, but it can invalidate the temporal behavior [KO02]. This makes the event-triggered architecture more flexible and avoids a restrictive design process as required for the time-triggered architecture [Alb04]. Moreover, event-triggered systems make **better use of the bandwidth** due to better average transmission time (the messages are transmitted as soon as the communication medium is available). This leads also to a better average system reactivity.

To conclude, event-triggered systems are well suited for sporadic transmission, alarm, low-power sleep modes and best effort soft real-time systems. Time-triggered systems, on the other hand, trade the the flexibility for more predictability, determinism and guaranteed latencies.

2.3.4 Time-Triggered Communication

TDMA scheme

In a distributed system, the communication services are forming the foundation for the system since they enable the exchange of messages (actualization of RT-images in time-triggered architectures or events in event-triggered architectures) between the computing units. Kopetz [Kop98] compares the operation of time-triggered communication protocols to train systems between stations. There exists an a-priori known schedule when the train arrives or leaves a station, and the client has to adapt to the time base. Furthermore, the train system operates deterministically and independently from the activity at the station and is synchronized to a known time standard.

Examples of Time-Triggered communication protocols are TTP/C [EBK03] for aircrafts or FlexRay [MHB$^+$01] for cars. These protocols implement a Time Division Multiple Access (TDMA) scheme based on a-priori defined, time windows ("communication slots"), which are uniquely assigned to the nodes for message transmission within a periodic communication cycle, see Fig. 2.4. The messages are broadcasted above the communication medium and consequently are available for each node of the cluster. A fundamental principle is that the transmission depends only on the time progression and is not triggered by any external (not-deterministic) event.

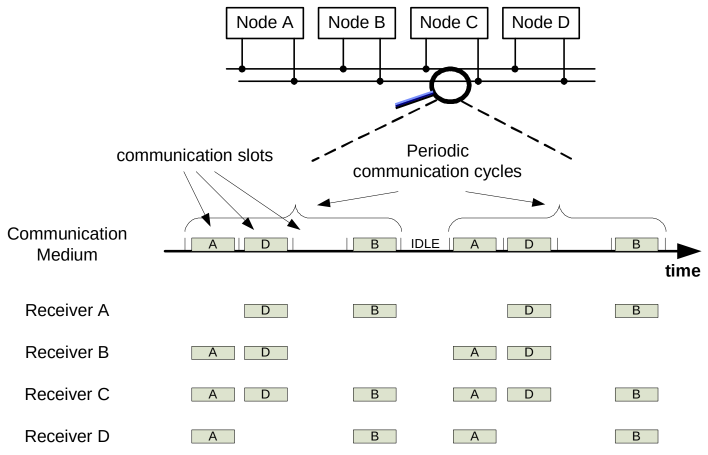


Figure 2.4: TDMA scheme

Time representation and clock synchronization

The FlexRay and TTP/C protocols defines the *microtick* as a node's internal time interval directly derived from the oscillator. The *macrotick* is the shortest time unit defined cluster wide and represents the granularity of the synchronized time base. The macrotick is defined as an integral number of microticks. A *slot* represents a time window eventually assigned to a node for the transmission of exactly one message. The FlexRay protocol further defines static and dynamic *segments* consisting of several slots and finally *communication cycles* as periodic communication elements. The equivalent to a communication cycle in TTP/C is the *TDMA round* . Figure 2.5 illustrates the time hierarchy for FlexRay.

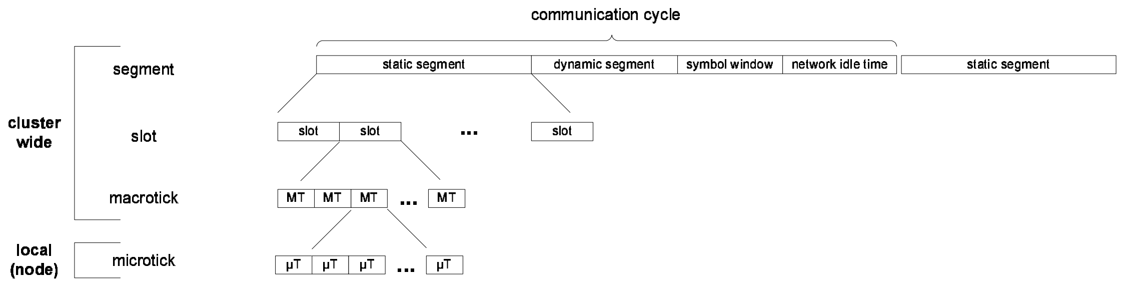


Figure 2.5: Time hierarchy for FlexRay

It seems evident that the establishment of a global notion of time among the node is a pre-requisite for the TDMA scheme for avoiding collision on the communication medium. Hence, periodic re-synchronization is required in order to correct the quartz drift and assure that the nodes' time bases are not drifting apart (the nodes within the system have approximately the same vision of the macrotick counter). For TTP/C and FlexRay, each node locally measures the time difference between the nodes' time bases and correct accordingly its own clock. A more formal description is provided in Section 4.1.3.

The central role of the clock synchronization mechanism for time-triggered communication protocols leads to an important behavior: the point in time when transmission occurs depends on the node's time base and thus on the node's clock correction mechanism. Notice also that the correction term depends on the messages received (in fact the reception time is used to measure the current time base deviation). This mechanism thus builds an implicit internal loop-back between the receive and the transmit path: A correctly received message has the potential to move the node's start of transmission. A detailed survey concerning clock synchronization algorithms is presented in Section 4.1.3.

Temporal firewall

Communication between two systems is commonly based on a master-slave control scheme. Data exchange can be initiated by the sender (*push* style) or by the receiver (*pull* style). In both cases the requester generates the control flow, and thus can start a transmission at any time. While this scheme is very comfortable for the master, the slave has to stay available at any time, which may result in high resource costs and difficult scheduling. Time-triggered communication protocols are using a combination of push and pull communication model [EBK03]. As stated before, the communication services are transporting a message from point A to point B according to a pre-defined schedule. The sender implements then a push style and transfers its data to a local memory, while the receiver implements a pull style to obtain the data (see Fig. 2.6). The communication services are in charge of transmitting the data from a memory to the other. This combination is ideal both for the sender and receiver since they can transmit and access the data whenever they want and do not need to be watchful for transmission request. This communication scheme is building a **temporal firewall** [KN97], a fully specified interface for the unidirectional exchange of data. Additionally to the resource saving, this interface is free of end to end control signals and thus avoids the possibility of control-error propagation.

2.3.5 Fault-Tolerant Middleware

The time-triggered communication system provides important attributes to support safety-critical operation (e.g., deterministic communication, channel redundancy). However, additional services are required to make the system tolerant to faults [TPDF99, NSSLW05]:

- **Membership** [Cri91, KGR89, RSV06] in order to consistently distribute the information concerning which nodes are currently operating correctly and which node has failed (agreement between the nodes)
- **Clique avoidance** to avoid the formation of cliques within the network
- **Message agreement** (replication of messages): the same information can be duplicated on the network to increase the probability of correct transmission, even in the presence of faults. The duplicated information is then filtered during the reception to present only one (fault-free) version of the message to the application.
- **Active replication** (replication of nodes): the same function is computed several times on different platforms. An agreement function is then required to extract one result out of the n computations (e.g., pick first, average). Note that the function also can be computed twice on the same platform (*double execution*).

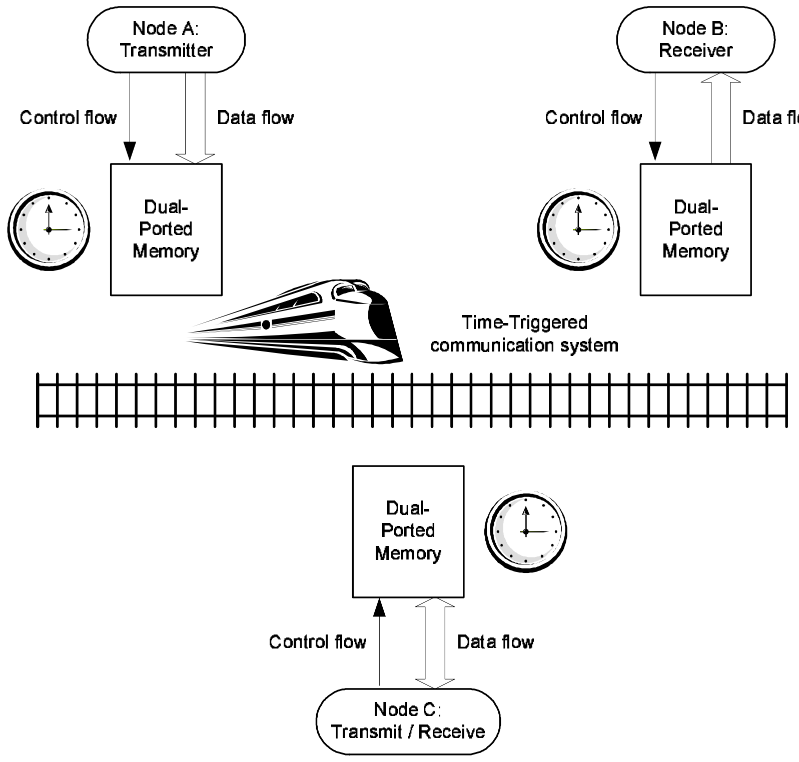


Figure 2.6: Push - pull combination

- **Recovery and re-integration** of nodes, after a fault has been detected
- **Assertion framework** to evaluate the correctness of the system during run-time. An assertion can either test the *pre-conditions* (respectively *post-conditions*) before (after) an operation or test the *invariants* of the system [TPDF99].

Other services are also usually provided by the operating system (OS) such as resources administration and validity checks, signature checks, the packing / unpacking of signals into messages and synchronization of the OS. These services are being standardized within the OSEK/VDX OS or the open architecture AUTOSAR (see Section 2.1.3).

2.3.6 Testing Time-Triggered Architectures

Time-triggered communication systems and fault-tolerant middleware are providing different services for fast fault detection (e.g. periodic message transmission working as a “heartbeat”) and informing about the current status of the system (e.g., which node has failed). Moreover, the concepts proposed by the time-triggered architecture support the system development by providing a work-around that eliminates the typical challenges of distributed systems (such as, e.g., race conditions). However, no explicit support is provided for testing or diagnosis, and the methods presented in the Sections 2.2.4 and 2.2.5 still apply.

Additionally, commercial tools are available for the analysis and diagnosis of the most popular automotive or real-time communication protocols. Examples are CANalyzer[3] for the CAN protocol, TTView[4] for the TTP protocol, or the BusDoctor[5] and the CANalyzer expansion for the FlexRay protocol. Implementation issues of these and similar tools along with some use-case scenarios can be found, e.g., in [POEL02, RV04]. All these solutions, however, only enable the monitoring of the bus traffic on top of the data link or higher layer employing COTS network controllers and device drivers in a promiscuous mode (if available) where even corrupt frames are forwarded to the processing CPU (plus some error flags). For the systematic, in-depth testing of a FlexRay based communication subsystem, this needs to be complemented by bus monitoring and dedicated measurements on lower abstraction levels, as well as an appropriate data analysis.

In addition to these monitoring approaches, a mechanism in the reverse direction that allows some kind of stimulus generation and injection or replay is required. In principle, any fault injection tool serves this purpose; see [HTI97] for a survey. Obviously, however, some protocol specific support is essential for performing efficient, well-aimed experiments that can be triggered by events on diverse protocol layers, e.g., and that also facilitates the generation of bus traffic with specific properties. The Disturbance Node[6] represents such a solution for a time-triggered protocol; with its particular strength for injecting physical faults like bus noise, short circuits, delays, etc., its focus, however, is on the physical layer itself rather than the communication services or the system. The BusDoctor[7], parallel to that, is focused on the data link layer and has the capability to send single frames on the network (and eventually simulate a collision, thus destroying a frame sent by another controller).

AUTOSAR (AUTomotive Open System ARchitecture [FBH+06]) defines an open and standardized software architecture for vehicle applications. This platform includes two main error handling mechanisms: The *Development Error Tracer* is targeted for integration support and reports node internal error messages. The *Diagnostic Event Manager* enables the tracing of status and errors during field operation. These mechanisms, however, are intrusive and dedicated to the application more than to the underlying architecture. Hence, they require explicit integration within the application and

[3]http://www.vector-informatik.com
[4]http://www.ttautomotive.com
[5]http://www.decomsys.com
[6]http://www.tttech-automotive.com/products/doc/TTTech_Automotive-TTX-Disturbance_Node-Flyer.pdf
[7]http://www.decomsys.com/flyer/Datasheet_BUSDOCTOR_2.pdf

are thus application specific. Furthermore, they require node internal resources (CPU time and memory), and finally the gathered data is focused to high level information, while detailed information about the communication status is difficult to obtain.

2.4 The STEACS Project

The acronym STEACS stands for *Systematic Test of Embedded Automotive Communication Systems*. In the automotive electronics, the network has been recognized as playing a central role in maintaining the system in a safe state, since the (critical) functions are now distributed and need to communicate [NSSLW05]. Moreover, the communication system is particularly exposed to EMI [NSL05], which usually increases its fault rate and decreases its quality of service (QoS). The aim of this research project was to address the arising test problems for the system integration of distributed embedded automotive electronics with a special focus on the communication subsystem. The main results are presented here.

2.4.1 Motivation and Requirements

To serve the various test purposes listed in Section 2.2.2, an ideal tester for a distributed embedded system based on FlexRay has the following properties regarding its practical employment:

Accessibility: For tests late in the product development cycle (conformance, maintenance), test points on individual nodes within the SUT are rarely accessible, since they are deeply embedded and sealed. This suggests a remote test approach. In principle, the communication network is an excellent access point for a tester due to its unique and central role for the communication among the distributed nodes. Still, however, all information that is local to a node, such as its status, is a priori not available at the communication network.

For tasks during early development phases, such as debugging during system design, the information thus gained will certainly not be sufficient. Fortunately, much better accessibility can be assumed during these phases, so the use of a remote test approach is not mandatory here.

Intrusiveness: The inter-operation of the nodes in a distributed system often suffers from very subtle effects, and this is where a probe effect is most troublesome. In this situation, the user might be willing to pay the price for keeping the test transparent for the ongoing system operation, which at the same time qualifies this strategy for online testing. In particular, this means that (i) the test must not change any application data, and (ii) in a real-time environment the test must not influence the timing behavior of the application (including the bus access pattern). Online monitoring fulfills this extremely difficult requirement.

For tests concentrating on a small subset of nodes it is often desirable to have some environment simulation provided by the tester. For this purpose, the tester cannot be "non-intrusive" – its visible bus activity is a desired property here.

Explicit test support: It would be quite unrealistic to assume that specific "test hooks" will be appended to an established protocol standard, or that series applications will take care for providing support for the tester. Therefore one requirement will be to come along without any supportive provisions or architectural changes in the SUT. Notice that this requirement prevents the explicit transmission of any node internal information (e.g. status) to the tester, which further aggravates the problem with the remote test mentioned above. Although it would well be possible to implement test support during early development phases, e.g., this option is not further pursued in order to keep the proposed approach generic.

At this point it becomes obvious that it will not be possible to achieve these contradicting goals with a single, monolithic tool. Rather, a bundle of complementary techniques is required to serve a given purpose.

Throughout this section the tester is assumed to be fault-free. Since it is a somewhat unique component, the test designer can allow higher cost as well as increased efforts for the tester, and an extensive self-test can be performed before putting it into operation. Additionally, a fault-tolerant architecture with redundant computing units can be proposed.

2.4.2 Problem Decomposition

It is well known that test complexity rises by far more than linearly – typically $O(n^2)$ – with the complexity of the SUT. The usual approach to combat this effect is "divide and conquer", i.e., partition the system into small portions that are relatively easy to test. In fact, the ability to easily decompose a system into parts with well defined behavior is one of the major merits of the time-triggered approach (see Section 2.3).

For the reasons explained above, however, it is necessary to perform a test of the assembled system even though all components may have already been tested in isolation. Here, of course, a decomposition of the system into single components is counterproductive. Performing an unstructured functional test of the configured system, on the other hand, results in excessive test duration and poor coverage (this is essentially the problem system integrators are going through today). Thus, some different kind of test structuring is needed.

The strategy proposed here is not to structure the *physical* system into components, but instead to identify and separate all *services* and break them into sub-services (which are called *mechanisms*) as far as possible. Note that these services and mechanisms are not necessarily bound to one physical node or component, but may rather be distributed. By virtue of this "vertical" decomposition into (ideally independent) mechanisms the problem is substantially simplified , while still retaining the system level view. For the implementation of this concept the use of a layer model for the system has been chosen, since the layer definition is generally based on the notion of services and hence, naturally suits this purpose. The vertical decomposition thus attained provides the following key advantages:

- It can be performed in a systematic manner, which aids in achieving a complete picture.
- It facilitates a systematic exploration of the fault space along orthogonal single mechanism failures ("basic faults"). Note that the granularity of the fault model chiefly depends on the decomposition granularity.
- Physical faults which are not directly reflected by one basic fault can be mapped to unique combinations thereof ("syndromes"), thus aiding in the generation of a fault dictionary.
- The global picture of involved individual mechanisms and the explicit visibility of their interrelations can substantially simplify diagnosis and an inspection of fault propagation, cf. [HJS01].
- If potential error signals issued by a mechanism are included in the model, a hierarchy of error signals can be constructed that further eases diagnosis.
- It aids in determining the abstraction level at which stimulation/monitoring should be performed to test a particular mechanism.

Let us illustrate this concept with the example of the network controller's receive service: Its task is to transform a serial signal received from the FlexRay bus via the physical line interface into the enclosed payload data that can finally be used within the application context. This involves many tasks in order to assemble, strip and check the received information (frame). Following this concept, this complex process is decomposed into a set of individual mechanisms M_k. Ideally, a mechanism has one single information input, one single information output and a status output. Its operation is entirely described by a simple model in a generic way, and one or several *attributes* can be used to characterize it (see Figure 2.7). Attributes can either be protocol configuration parameters (e.g., number of static slots) or protocol constants (e.g., maximum clock frequency deviation).

Based on this model, the system test can be projected to a check whether all attributes are in the allowed range. The test of an error detection mechanism, in turn, implies observing the reaction of the system to a message that has been generated with the associated attribute(s) being erroneous.

This approach is somewhat similar to the OSI layer model, however, is finer-grained. In [ASH$^+$04] the complete model for the FlexRay communication services has been

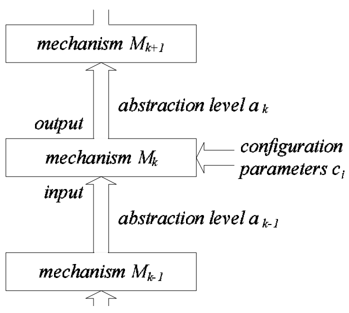


Figure 2.7: Mechanisms and Abstraction Levels

illustrated. The following key property of this model which will become important later on shall be highlighted: Mechanisms can be hierarchically ordered in levels, such that a high-level mechanism M_k builds upon the services provided by the lower-layer mechanisms $M_{k-1} \dots M_1$. Therefore the service of mechanism M_k does not suffer from an erroneous behavior or incorrect configuration of a higher-layer mechanism M_{k+i}, while it does suffer from an error of a lower layer mechanism M_{k-i}. This property allows us to check the sub-services one by one, starting with the lowest level mechanism and successively increasing the level. Thereby, every step builds upon the results of the previous one(s). In some rare cases, mechanisms comprise two attributes which are mutually dependent on each other, such that the test space becomes two-dimensional – still resulting in a significant saving.

It is easy to observe the activities of every node's transmit service on the FlexRay bus, which facilitates a completely transparent remote test approach: The tester is connected to the bus, passively observes the traffic and draws the appropriate conclusions. Due to the high regularity of the transmission service, all mechanisms are sufficiently exercised during normal operation, such that simple monitoring is indeed a sufficient test here [ST99]. The identification of a faulty node is easily possible due to the static TDMA schedule. Diagnosis with respect to the faulty mechanism can be based on the hierarchical structure of the mechanisms outlined above: Should, e.g., the sender's CRC generation mechanism be faulty (or incorrectly configured, in case of another mechanism), then all mechanisms below (encoding etc.) will still operate properly such that a message containing the incorrect CRC will finally be transmitted. The tester will receive this message, decode it, strip all of the framing information, etc., until it comes to CRC decoding. At this point, an error is signalled. Assuming a fault free tester (and physical line), this error can be directly projected to the sender's CRC encoding mechanism. This correspondence between the sender's and the tester's abstraction levels is illustrated in Figure 2.8.

Testing the receive service, however, requires additional provisions. The first prob-

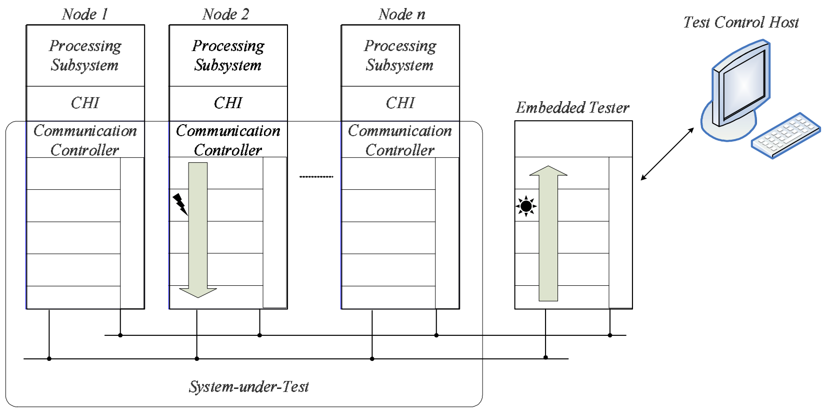


Figure 2.8: Remote testing

lem is that the operation of the receive service cannot be directly observed on the FlexRay bus. There are two options here: (a) Directly access diagnostic information on the nodes of the SUT. This is a very powerful approach but obviously inhibits a remote test. (b) Draw conclusions from the node's behavior that can be observed on the FlexRay bus. Here, the tester can exploit (with some limits) the fact that a node that does not receive messages on the bus will run out of sync, since clock synchronization is based on the reception of (valid) messages. The second problem is that the receive service includes many error detection mechanisms which are not regularly executed during normal operation. Therefore active stimulation will be required.

2.4.3 The Systematic Test Approach

With the layer approach the entire communication system can be decomposed into a collection of services and mechanisms, and the remaining task is to check the values of the attributes that characterize every mechanism of interest. For this purpose, access to its inputs and outputs are needed as well as to the configuration interface for monitoring and – in some cases – for control. Beyond just this mere physical access, an elaborate strategy of how to interpret the observed information and how to prepare a stimulation is mandatory. Here the concept relies on generic building blocks as illustrated in Figure 2.9 for one abstraction layer. The same approach can be reused for every other abstraction layer within the communication system, and, in principle, can be targeted effortlessly to other bus-protocols as well. For more details see [ASH05, ARS+05].

For either path reference, values are provided by the *correct behavior generation* module. This reference behavior is built from data obtained by the *service specification library* and by the *current configuration*. The specification of the system attributes (e.g.

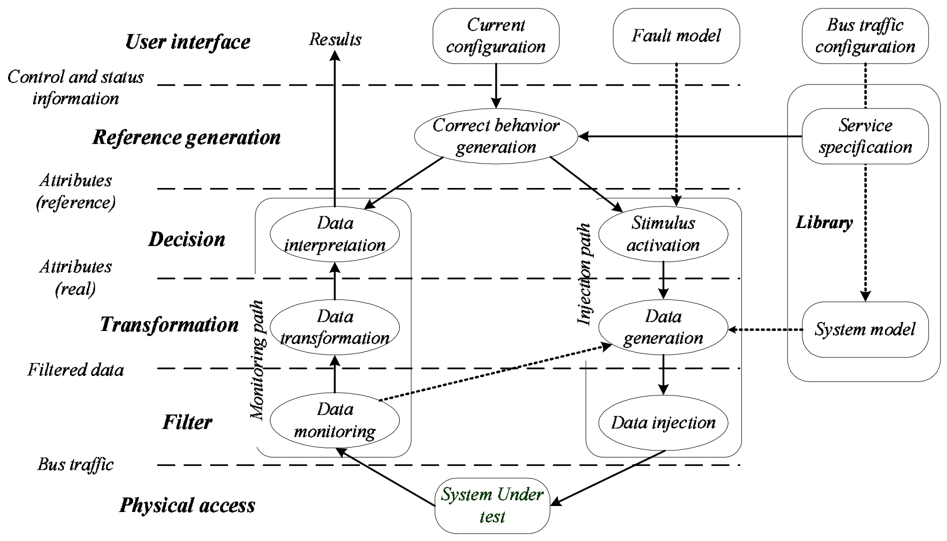


Figure 2.9: Test approach for one layer

min/max values for every attribute) is constant over time and, consequently, can be linked as a library to the test environment. The configuration, however, is specific for the actual system setup and must be provided by the test engineer (e.g. the actual duration of a static slot). Finally, an optional *system model* can be used to generate a bus traffic conforming to the specification and an optional *fault model* can be deployed to map a user defined fault model with the actual faults being injected.

In this model, the monitoring path aims at providing means for observing ongoing system operation, for (automatically) extracting the attributes of interest and evaluating their correctness (e.g. whether they are within the expected limits). The *data monitoring* module processes the received data up to the abstraction layer where the selected attributes can be best analyzed. Therefore, this module takes as input the bus traffic at the physical layer and de-encapsulates all data up to the particular layer by removing information used solely by the lower layers. The *data transformation* module takes the de-encapsulated information and reduces its dimensionality by extracting information relevant for processing the attribute values and for relating it to its source (node). The *data interpretation* module interprets the monitored attribute values and thus the corresponding system behavior by comparing it with reference values. The result may be either a match/no-match (useful for conformance tests) or the distance from the observed towards the expected reference values (useful for robustness tests). Furthermore, with this module one can easily record a result log for attributes over time, thus gaining insight to the evolution of attribute values. This feature is especially useful during system operation to obtain information for preventive maintenance, e.g. to identify the weakness of certain components. Another typical application of the monitoring path is automatic configuration parameter identification, see also [ASH06].

The injection path provides means to automatically select and modify one or a set of attribute(s) and emulate the resulting behavior in the communication. The *stimulus activation* module determines which and how attributes need to be modified. In particular, attributes are either selected based on information provided by a user-specified fault model (i.e. for robustness tests) or based on all the relevant attributes for the service being tested (i.e. for conformance tests). The *data generation* module modifies a given “normal” bus traffic in order to emulate the malfunction of one (or several) services by altering its/their corresponding attributes for a defined time interval. This data modification is performed at the location of the target service layer. This is accomplished by combining the information from the stimulus activation module either with recorded data obtained from the data monitoring module or with data derived from a system model. The *data injection* module takes as input the data file and performs all required processing down to the physical layer. THe last step is to push the data onto the wires with the appropriate timing.

Note that the underlying bus traffic for the *data generation* module might be either generated (from the *system model* module) or monitored and then modified (from the *data monitoring module*). Obtaining data from a running cluster presents the advantage of producing representative test vectors, but requires an operational system prototype and does not necessarily conform to the specification. This approach is well suited for robustness tests where good representativeness plays an important role and a running system is usually already present. In contrast, “perfect” bus traffic can be generated from a reference model. This approach does not require a running cluster and provides bus traffic which conforms to the specification, which is more appropriate, e.g., for conformance tests. While additional efforts are required to design the reference model, this second method typically allows more freedom in generating suitable bus traffic.

2.4.4 Summary and Outlook of the STEACS project

The main contribution of the STEACS project is the definition of a fined-grained layer model based on the concepts of attribute, mechanism and abstraction level. Using this decomposition, the distributed communication service can be divided into simple sub-services that can be easily tested and evaluated. Methods for getting access to the abstraction level of consideration have also been presented.

The broadcast nature of the communication network was used during the STEACS project to enable partial remote testing: an error within a node’s transmit path results in a corrupted transmission that can be monitored when accessing only to the network. This one-to-n communication can be exploited to obtain status information about the node’s transmission. However, the receive path is more difficult to monitor, since its status is only internally visible. During this project, dedicated test applications were required in order to monitor the system reaction while performing fault injection.

This limitation provided the context for our ExTraCT project: how can we obtain information of the node's receive path (more specifically, about the reception status) without influencing the application?

2.5 Chapter Summary

Electronics are currently present at every level of vehicle design. Both in the control domain or for the human-machine interface, electronic components are reducing the overall costs, while providing optimized or even new functions. Vehicles are now developing into complex distributed real-time systems, that almost always implement several clusters and different technologies. This current trend is confronted with two main challenges. First, cars are operating in a very harsh environment that might disturb operation and/or cause component failures. Second, electronic components have been introduced for safety-critical operations where a system failure might have catastrophic consequences. It seems evident that methods to test the system and provide guarantees of correct operation are mandatory for the sake of preventing such consequences.

This survey of test methods for distributed systems has highlighted different challenges coming from the state explosion, the race conditions and synchronization between the processes, the timing constraints and the high degree of integration. These challenges are additionally made more difficult to satisfy because of the large scope of testing, the different tools, knowledge and support required and the current constraints (e.g., possibility to insert debug code).

The time-triggered architecture has been introduced in this context to support the development of safety-critical systems. One of its most important attributes is enabling deterministic operation (both for communication and task execution) based on an a-priori known schedule. This planning is used to avoid per construction any conflicts or race conditions, to support constructive design (composability) and fault tolerance. While this architecture provides efficient support for fault-tolerant operations, the industry still requires means to test and diagnose the assembled system, and more specifically the communication subsystem (which plays a central role in this architecture and is especially exposed to EMI).

Toward that end, the STEACS project has made a first step. A fine-grained layer model was proposed to structure the communication service in mechanisms and abstraction levels. This decomposition was further used to achieve systematic testing of the communication functionalities. One primary requirement was to propose a remote solution in order to minimize the intrusiveness of the test approach. The remote tester which was proposed was indeed able to finely analyze the information available at the communication medium and was also able to inject patterns at the bus level. However, the node's internal information (e.g., reception status) was not available for the tester and modification of the system under test was required, thus rendering the approach intrusive.

Chapter 3

A New Approach for Transparent Testing

Everyone knows that debugging is twice as hard as writing a program in the first place. So if you're as clever as you can be when you write it, how will you ever debug it?

Brian Kernighan (1942 –)

Important improvements have been realized in the last few years to develop reliable architectures for automotive electronics. This helps improve the overall system quality and supports the introduction of safety-critical applications. The contribution of this work is the proposition of a transparent, online test approach focused on the communication services. First, we describe the problem statement define the term *transparent*. Next, we analyze the diagnostic information which is globally available at the network and then introduce our transparent test approach. We conclude by the identification of three challenges for the validation of our approach.

3.1 Problem Description

3.1.1 Motivation

We have seen in the previous chapter that the network plays a central role in maintaining the system in a safe state. At the same time, this service is the one which is most exposed with regard to external disturbances. The goal of this work is to provide an approach to test the *reliability* (continuity of correct services [ALRL04]) of the communication system and the *availability* (readiness for correct services [ALRL04]) of the associated error detection mechanisms. This second point is of utmost importance, since it represents the capacity of the system to correct its state before a failure occurs.

The time-triggered architecture (combination of communication services plus fault tolerant middleware) already provides different services for error detection and correction as well as fault tolerance. Our main concerns are **error diagnosis** and **online testing**. Error diagnosis aims at identifying the service that is currently deviating (which kind of deviation, which module has failed). This information is required for system recovery and during maintenance in order to decide the action to perform (e.g., new start of the system, replacement of one module).

Online testing focuses on error detection and fault-tolerant mechanisms, and aims at periodically exercising mechanisms which are infrequently used, in order to detect dormant faults and to avoid fault accumulation [SS03]. Hence, fault accumulation represents a real threat for the system since this effect can lead to a violation of the specified fault hypothesis and cause a failure of the system. The availability of the error detection mechanisms has a direct impact on the robustness of the system.

Our work applies to a late product life-cycle and assumes a highly integrated system, functioning during normal operation, and without explicit test support. Therefore, our test approach presents the following requirements (similar as in Section 2.4.1): test points on individual nodes are rarely accessible and we require a **remote test approach**. Additionally, we need a **non-intrusive approach**, since we can not afford any probe-effect that might be a threat for correct service delivery. Moreover, we need a **generic approach** which avoids any explicit support of the test operation, in order to save resources (CPU, bandwidth) and to avoid complex standardization processes.

Possible applications are online testing and diagnosis during field operation, as well as periodic maintenance in a garage. Hence, our approach enables the online checking of assertions and the access of various information, such as the current service quality or availability.

3.1.2 Problem Statement

According to [BMS87] the two key aspects for testability are controllability and observability. In a system with good controllability it is easy to apply test vectors and to propagate them to the suspected target location. Observability is a measure of how easy it is to observe certain activities of the SUT, particularly at the target location. Both measures are largely determined by the accessibility of the SUT. In short, good accessibility leads to good testability. The main challenge of our approach is the accessibility to the system. Communication controllers are building an interface between the application and the communication channel, and typically implement two independent processing paths to (a) send and (b) receive messages from/to the network. Figure 3.1-a pictures a typical setup to test the transmit path of a node. Since the inputs are provided by higher layers, intrusive by-passes are required from the tester in order to control the transmitter and apply the stimulus. In contrast, the outputs of the transmit path can be directly observed through the communication medium.

Testing the receive path (Figure 3.1-b) presents similar problems. Indeed, this module can be controlled by the physical medium, but an additional communication channel is required for observation. While a dedicated application could autonomously apply a-priori defined test vectors to the transmitter, the observation of the receive path does require an additional communication channel for test purposes. Such approaches are consequently not suited for validation and maintenance in the automotive context due to their high intrusiveness (the nodes have to be modified) and / or resource overhead.

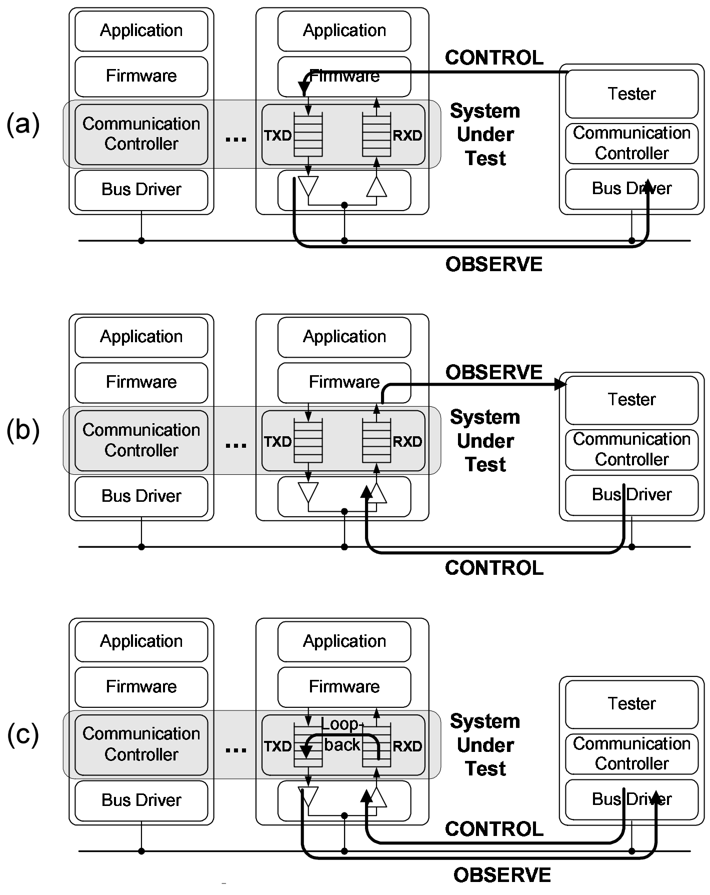


Figure 3.1: Observation and control path for testability

The periodic and a-priori known schedule implemented by time-triggered architectures is convenient for test operation, since a tester knows a-priori when and how a mechanism will perform. This periodic stimulation allows to remotely test the syntactic correctness of the messages generated and therefore also the correct operation of the mechanisms which are exercised within each node's transmit path. The receive path, on the other side, can not be easily observed. The use of an internal loop-back, as illustrated in Figure 3.1-c could be used to improve the observability. This approach presents the primary advantage of using the available communication channel to return the observation to the tester node, which saves resources and limits intrusiveness. While a traditional loop-back typically requires additional resources and dedicated services to confirm the information received, the key idea here is to use the implicit internal loop-back provided by the clock synchronization mechanism (see Section 2.3.4): A correctly received message has the potential to move the node's start of transmission, an effect which can be observed by our tester. It is the aim of this work to **present a novel framework for the transparent testing of time-triggered communication protocols** such as TTP/C and FlexRay, based on this very principle.

3.1.3 Classification of our Approach

We consider our approach to be basically a remote online monitoring method, with an extension towards transparent online testing, as presented in Section 3.2.2. To clarify this statement, we will briefly survey the related terminology in the following section.

Monitoring vs. testing

The normal test procedure is a stimulus/response measurement: Well-chosen test vectors are applied to the system under test and the response is observed and compared to a known good reference. This approach, by its very nature causes an interference with a potentially ongoing operation of the system under test, which contradicts with the requirements presented in Section 3.1.2.

However, under certain conditions the normal system operation can be considered a sufficient set of test vectors, and hence, the requirement to explicitly apply stimuli may be relaxed. It is sufficient rather to observe the ongoing system operation and check whether or not it meets the expectations. For this purpose some kind of rule checking is usually employed, e.g. [HP03]. The main question when using this monitoring approach is whether all relevant system functions are exercised during normal system operation within the observation time. Usually, without explicit stimulation some exceptional states (e.g., emergency handling) are very likely not to be entered and the associated resources hence not to be exercised and tested. This limited test coverage may cause problems in context with latent errors, as outlined in [SS03]. The major advantage of monitoring is that – since no active stimulation occurs – it is not necessarily intrusive, since it can be performed in a transparent way. In addition, the sequence of test vectors comes for free and is representative for the actual application. Our first strategy will therefore be to rely on monitoring the network traffic (see Section 3.2.1).

Online vs. off-line testing

In the classical sense, the test is a very specific mode of operation of the system under test: The system is taken off-line, a well chosen comprehensive set of test vectors is sequentially applied to the system under test in order to step through a sequence of states and thoroughly execute all relevant functions. The resulting system behavior is monitored and compared to a known good reference [Sch94b, TH99]. The theory for this off-line test approach is well developed and the test usually achieves good coverage, since having complete control over the system facilitates good testability. Online testing, in contrast, is performed while the system under test is still providing its service. The main advantages of online testing are the following:

- Online testing may be continuously ongoing, which allows very fast detection of errors – as compared with off-line tests that are usually performed in long intervals. Specifically, online testing is a means to prevent fault accumulation. While the test activities might jeopardize the reliability of the system, it has been shown in [SS03] that the negative impact of fault accumulation may be even worse.

- The continuous operation of the online test allows collecting a comprehensive and representative test record. This in turn facilitates statistical assessment and projections, e.g., the identification of "weak" components that are liable to fail soon ("preventing maintenance [ALRL04]").

Transparent testing

We define *transparent testing* as the capability to perform test operation without the system taking notice of it. As for online testing, it signifies that the system's status and service delivery is not modified in an observable way. As a consequence, it has no probe effect.

This means additionally that the test operation does not require extra processor time or chip area or does not compete with other system tasks for resources. This saving is especially important for the purpose of meeting the aggressive cost and performance goals required in the automotive industry. Moreover, it facilitates test execution, since no modification is required and standard hardware can be used for that purpose.

3.2 From Monitoring to Transparent Online Testing

The use of probe-nodes (see Section 2.2.4) is a common way of monitoring inter-node communication. In this section, we review which diagnosis information can be extracted only by monitoring the system and why the time-triggered scheme is especially beneficial for our purpose. We present further how the clock synchronization algorithm can be used for transparent test operation.

3.2.1 Remote Online Monitoring

Direct diagnosis

Being restricted to remote access, we have to carefully exploit every nuance of information which we are able to observe on the communication network. And indeed, the FlexRay specification facilitates a lot of checks to be performed on a message:

- **Time domain:** The static transmission schedule makes it possible to determine in advance, precisely when a node must transmit a message. A failure may be identified not merely because a message was sent at the wrong instant (incorrect temporal alignment) but also when it was omitted.
- **Value domain/syntax:** The syntactic correctness of a message can be checked with respect to coding, CRC, header integrity, etc.
- **Value domain/semantics:** Given that the tester is provided with application-specific knowledge, several assertions can be applied to the semantic correctness of a message, such as boundary checks, plausibility tests, model-based checks, etc.

  This last option, however, is out of the scope of a generic communication service test.

For our remote test setup this means that the tester can indeed passively listen to the traffic on the communication network and perform the above checks to judge the integrity of the messages. As soon as an abnormal behavior is detected, the tester can – thanks to the static schedule – easily project this error to the producing node (except for severe errors in the time domain). On the other hand, if the tester receives correct messages from a remote node, we may draen some important conclusions:

- After having received at least one correct message from each node that participates in the communication, we can conclude that there must be an intact link between the tester and any other node. For passive bus topologies this also means that an operational link between any two nodes exists as well.

- The basic services of the transmit path of that node must work properly, otherwise we would have encountered syntactic errors.
- The basic clock synchronization services must work properly, otherwise the frame would not be properly aligned.
- Depending on application and assertions we may be able to conclude that the application has produced correct output messages based on correctly received input messages.

This is already a very useful amount of diagnostic information, and in context with a clever interpretation the status of the communication subsystem can already be characterized quite well. Let us briefly summarize the coverage that we have attained so far:

Transmit path: We can detect permanent and transient errors in the transmit path safely, since the output of this path is directly observable. Moreover, due to the time-triggered nature of the communication, the services within the transmit path are periodically exercised. In the transmitter there are no exceptional paths and no mechanisms for error detection, whose status must be observed. The correspondence between the service layers of sender and receiver pointed out in Section 2.4.2 allows us to use the error symptoms at the tester's receive path for diagnosing the actual error in the sender's transmit path.

Receive path: The situation is much more difficult here: The receive path processes information from the network and provides it to the application, which means that we can observe its input but not its output. The receive path additionally maintains an error status which is in fact much more important for diagnosis than the actual message contents (these can be derived from the observed network traffic anyway). Obviously what we are lacking here is some mechanism to loop back this reception status information to the network where it can be observed. We may use assertions on the application level to serve this purpose. However, even if these are available, fault tolerance features of the application tend to mask the loss of single messages.

Clock synchronization service: The function of the clock synchronization service can, in principle, either be judged by the alignment of the transmitted frame, or by the absence of scheduled transmissions: Without a synchronization, the node will go into a silent mode. Identifying the failing node is possible, but further diagnostic information is not available.

In conclusion, we may be quite satisfied with the coverage of the transmit path. What we definitely lack is a way to observe the status of the receive path with our remote tester. Therefore we will investigate the clock synchronization service in order to be able to loop back this information to the communication network.

Exploiting the loop-back via the clock synchronization service

The basic service provided by the clock synchronization algorithm in time-triggered architectures is a system-wide, globally synchronized (sparse) local time base maintained by a so-called Macrotick count. Among other things this time base forms a fundamental prerequisite to maintain the TDMA scheme as described in Section 2.3.4. Based on the local clock and the static bus schedule every node determines the points in time within a cluster cycle when it is supposed to send. Should its local time (or its understanding of the global schedule) deviate, the node will incorrectly align its message with the global schedule. This in turn can be perceived by the tester either as a shifted message, as a truncated message or a message omission, when an optional bus guardian prevents the node from sending outside its pre-assigned transmission slot.

In short, this means that every sync message that is received correctly by a node N_x contributes to N_x's clock correction, which, in turn, is visible at the alignment of the messages sent by N_x. So apparently, the clock synchronization service forms the desired loop-back of the receive path error status to the communication network. This is illustrated in Figure 3.2.

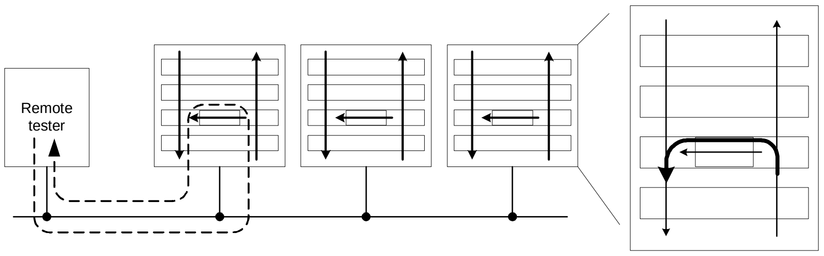


Figure 3.2: Loop-back via the clock synchronization service

In order to exploit this loop-back let us summarize several important properties of the clock synchronization service from the description above:

1. The algorithm only considers sync messages. All other messages (and their loss or failure of reception, respectively) have no influence on the local clock synchronization.

2. For a sync message to be considered by the algorithm it must have passed semantic checks in the value domain (like CRC check) and a frame alignment check

in the time domain, which are quite selective. With a faulty receiver, it is very unlikely for a message to pass these tests.

3. The algorithm can tolerate the loss of sync messages without problems, and only requires two sync frames per cluster cycle (for FlexRay) to maintain its synchronization.

4. If too few sync messages are received, then the node changes its mode to "listen only" for some cycles and switches off after a pre-configured time-out, unless it can successfully receive further sync frames.

From (2) we can conclude that permanent errors in the receive path will eventually lead to loss of synchrony and hence be detected by the remote tester – according to (4) as a message omission. Furthermore, it follows from (1) and (3) that the loss of single messages due to transient faults in the node under test's receive path is not detectable by the remote tester, even if a sync message is affected. According to (3) a transient fault must affect a sufficient number of sync messages during several succeeding cycles to become detectable. Hence, transient faults are very unlikely to lead to a loss of synchrony – which is good news from the point of view of robustness, but inhibits their remote detection.

From the proper function of the clock synchronization services, on the other hand, we can conclude that the clock synchronization algorithm must have received sync messages recently. Otherwise, the node would have eventually lost synchronization. This further implies with high probability that

- the basic services of the receive path must work at least up to the frame check level (this is where the decision is made whether or not to consider a sync message for clock correction) and that

- there must have been at least one successful reception during the previous communication cycle, otherwise the node would have gone into turned silent mode.

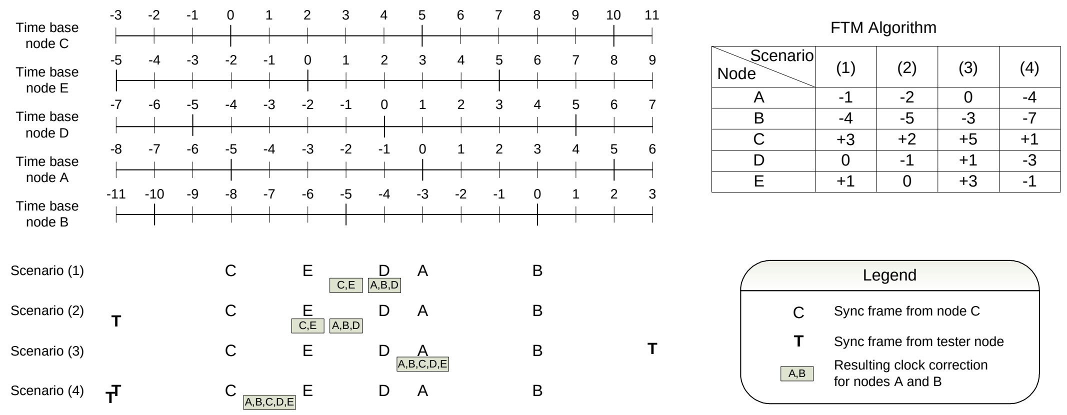
Beyond this "go/no-go" check as to whether or not the messages sent by the node are properly aligned within the schedule, it is conceivable to additionally perform a more subtle test on the position of the node's messages within the allowed time window. This position, however, is influenced by many effects and the variation resulting from the loss of one sync message is very small. Therefore, even if the tester can resolve the frame position precisely enough to recognize such variations, it is extremely difficult to distinguish the desired effect from variations caused by other sources.

3.2.2 Toward a Novel Approach for Transparent Online Testing

In the previous section we have identified the loop-back formed by the clock synchronization service and employed it for a rough go/no-go check of a node's receive path. Still, the diagnostic capabilities of this approach are far from satisfactory, and we still have no means to test the error detection capabilities in the receive path. At the same time, we already know that there is more information hidden in the frame position than we currently make use of. So let us go one step further.

We already know that it makes a difference for the result of the clock synchronization algorithm whether a sync message has been correctly received by a node or not, and that this difference becomes visible in the position of the frame transmitted by this node. Unfortunately – for our purpose – the individual local clocks are tightly synchronized, and potential outliers are suppressed by the FTM algorithm. So the loss of one sync message won‘t make much difference in practice. But what if our tester actively transmits messages that do have a measurable influence on the frame position if they are correctly received? In that care, the lack of such a reaction will in turn allow us to conclude that the test message has not been correctly received.

All the tester has to do for this purpose is to send one regular message ("stimulus message") within one communication round that (*i*) is a sync message, (*ii*) exhibits a sufficient deviation from the "normal" time base to cause a visible impact on the clock synchronization algorithm (i.e. represents a "stimulus"), but (*iii*) is still considered as valid. Condition (*i*) is easy to fulfil. Conditions (*ii*) and (*iii*) can, in principle, be met by positioning the stimulus message to an extreme location within the allowed time window. Figure 3.3 shows an illustrative numerical example:



Node \ Scenario	(1)	(2)	(3)	(4)
A	-1	-2	0	-4
B	-4	-5	-3	-7
C	+3	+2	+5	+1
D	0	-1	+1	-3
E	+1	0	+3	-1

Figure 3.3: Actively influencing the clock synchronization

The upper left part of the figure represents the mutual alignment of the internal time bases from the different nodes. The lower part shows how the individual messages transmissions fit into this time base. Notice that a message is always sent at time '0' according to the sender's time base. For example, the message transmitted by node D has an offset of -1 for node A and an offset of +4 for node C. Four scenarios are pictured here: First, as reference, without tester node (scenario 1), then with a single tester frame with negative offset (scenario 2) and positive offset (scenario 3) and finally with a tester sending two frames with an important negative offset (scenario 4). The results of the clock correction algorithm (FTM) are summarized in the top right of the figure for the different scenarios.

Let assume now that node D in the system under test perceives the following individual deviations from its local time base {-4, -2, 0, +1, +4} measured in "time units", which might be ticks of the local clock. If the tester did not interfere, the FTM algorithm would discard the highest (+4) and the lowest (-4) deviation and compute a (truncated) average over the remaining interval borders, which yields (0). By sending a stimulus message with a deviation of (-7) or (+7), respectively, the tester can cause node D to move the result of its FTM to offsets (-1) or (1), respectively, in this example.

Notice that the stimulus message was in fact not considered in the midpoint, but rather discarded as an outlier. By occupying the outlier position, however, it caused the original outlier to be considered, which yielded a visible effect. In practice, however, true outliers will be rare – in the fault free case the clock synchronization will keep all local clocks within tight bounds. Therefore, pushing a value from the outlier position into the average will not necessarily cause a noticeable effect. Obviously the solution is to let the tester occupy all outlier positions plus one or more values with stimulus messages (Scenario 4 of Figure 3.3) to actively influence the result. This is relatively easy to achieve, since the FlexRay protocol [Fle05] defines a maximum of 15 synchronization frames per communication cycle and a maximum of four outliers (the two highest and the two lowest) to discard. Hence, the generation of 3 synchronization frames is enough to directly influence the result of the FTM. If more than the 15 allowed synchronization frames are sent within a cycle (e.g. when the application already sends 15 sync frames and the tester additionally three), then the first 15 are taken into account and an error is signaled to the hosts. Consequently, this method only requires three empty slots (out of the 2047 possible identifiers) prior to the 13th synchronization frame.

As compared with the mere monitoring approach discussed in Section 3.2.1 the proposed careful control of the clock synchronization services explicitly exercises the loop-back path and enables observability of the receiver status, which provides important advantages:

- For a node that adapts to the above changes properly we can conclude that it must have properly received the tester's stimulus message(s). Otherwise we can conclude that the receiver has encountered an error upon reception of the stimuli.

- By exercising the ability of the clock synchronization service to adapt to changes in the network timing, we attain a better coverage of the clock synchronization services.

It is important to notice here, that this approach is still completely transparent for the application. The tester's stimulus messages simply remain unused by the application, and even if they should fail to be received (e.g., due to an overly aggressive choice of their displacement) this makes no difference. Moreover, we still do not require any changes on the system under test, we only exploit the existence of the clock synchronization services in a transparent way. The only resources we consume is the bandwidth for three frames. One might argue here that the "unaligned" tester messages tend to unduely exhaust the fault-tolerance of the clock synchronization. In general, the negative impact of fault accumulation, which can be eliminated by the proposed online testing (as shown in [SS03]) constitutes a much more severe dependability threat. Further discussion can be found in Section 4.2.

The requirement on the tester for this type of test is the ability to send messages according to a very fine-grained time scale. In order to observe the reaction of the global time base, the tester must be able to sufficiently resolve the position of a message within the assigned time slot. In practice, it is necessary to influence several consecutive communication cycles.

3.2.3 Taking Control of the Clock Synchronization Service for the Loop-back

The next step in exploiting the clock synchronization service for testing is to have the tester generate sync frames in such a way that it essentially takes control of the remote nodes' clock correction mechanisms over time and forces them to follow a non conventional but still correct time pattern. This approach is known as *deterministic replay* [TH00] and is further discussed in Section 4.2.1. Sections 5.3.2, 5.3.3 and 5.3.4 describe test campaigns where deterministic replay is used to evaluate the limits of the FlexRay clock synchronization algorithm. In these experiments the tolerance of the clock synchronization service has been overstressed, and as a result, the nodes under test go into silent mode and communication breaks down. While such a test is still valuable for an off-line characterization of cluster parameters (e.g., conformance testing), it is not suitable any more for the transparent online testing we are aiming at.

However, we need not go that far. Depending on the perceived fault scenario a FlexRay node can exhibit different reactions: It may (1) discard the faulty message and continue providing its services, (2) turn into a listen only mode or (3) shut down after a timeout. In context with our transparent testing scenario (1) seems much more interesting: We can again use our tester to force the time synchronization to a "non-natural" state, however, we do not go beyond the capabilities of the clock synchronization services. This still allows us to distinguish whether or not the stimulus messages have been received. By modifying these stimulus messages (insertion of faults, e.g., for a possible strategy see [ASH05]) we can exercise exceptional paths and remotely trigger the node's error detection mechanisms. While this remains completely transparent for the application (the stimulus messages are not considered by the application), we can exploit the loop-back to observe the reaction of the receive path to the stimulus.

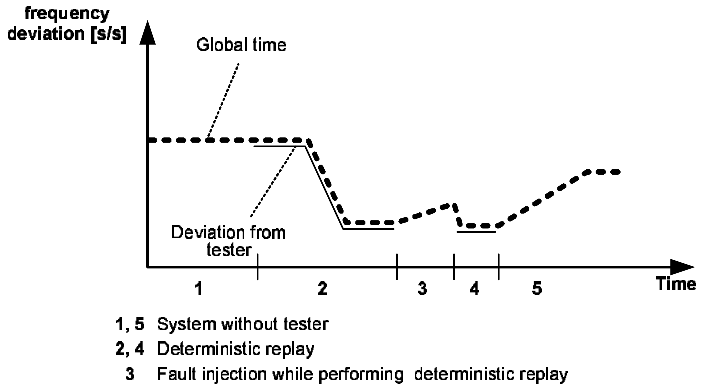

frequency deviation [s/s]
Global time
Deviation from tester
1
2
3
4
5
Time
1, 5 System without tester
2, 4 Deterministic replay
3 Fault injection while performing deterministic replay

Figure 3.4: Transparent test approach

Figure 3.4 illustrates the approach: During a first time, the tester (silently) synchronize to the system (phase 1). After that, our tester performs deterministic replay and drives the global time outside of some interval (phase 2). The objective is to move the system into a correct but non-natural state which the SUT will move from as soon as the tester stop sending messages. During the third phase, the tester frames are corrupted (e.g., CRC error), and consequently removed from the nodes' clock correction computation. The SUT starts moving back to a natural state, thus illustrating the correct rejection of the corrupted tester frames and thus the correct operation of the node's internal CRC decoder. A node that still follows the corrupted tester frames would indicate a corrupted error detection mechanism. During the (optional) fourth phase the tester performs correct deterministic replay again and finally the tester ends operation (phase 5).

The experiment pictured in Figure 3.5 illustrates this claim: We are considering a (simplified) cluster consisting of two nodes that transmit one frame each. They are configured for a communication cycle length of 3000μs and assumed to be fault free during the whole experiment. In addition a tester generates two stimulus messages with the aim of driving the communication cycle length down to 2998μs. All messages are fault free but for communication cycles 1930 to 1939 during which both stimulus messages exhibit header CRC errors (a total of 10 frames for each channel were modified).

We first started up the tester alone. After 1000 cycles we allowed the nodes under test to integrate into the ongoing network traffic. After further 1000 cycles the tester stopped generating messages and the nodes under test were free running. If our above hypotheses are true we can expect that:

- The SUT will be able to integrate on the network traffic generated by the tester, even if the cycle length slightly differs from the one configured
- The SUT will stay synchronized and thus provide its normal services
- The SUT will produce an observable but specification compliant reaction to the fault injected and still deliver its services without any alteration.

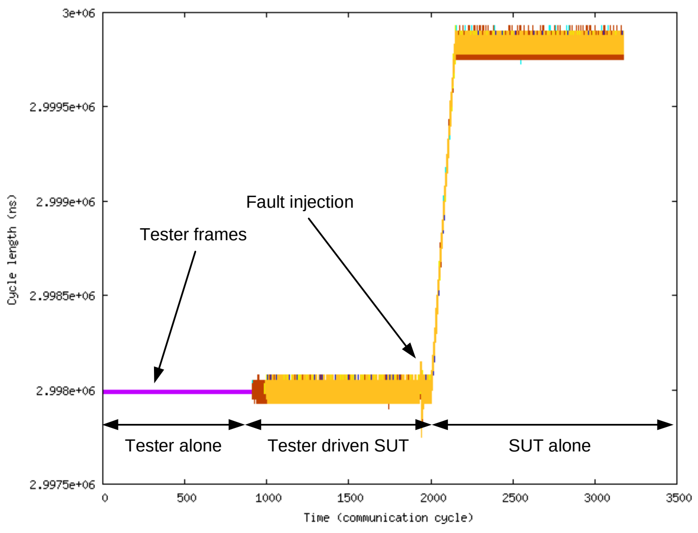


Figure 3.5: Fault injection experiment

Figure 3.5 pictures the cycle length of the four frames over time. As expected, the SUT integrated to the existing network traffic (shortly before communication cycle 1000), and stayed synchronous although the cycle length was shortened by 2μs. When the stimuli messages ceased, the transmission period of the cluster returned close to the configured value of 3000μs. Indeed the faulty frames sent between the cycles 1930 and 1940 lead to a reaction of the nodes under test but did not crash the communication services. The errors contained in these messages were correctly detected (thus providing evidence for the proper operation of the CRC checker) and discarded from the internal clock correction computation. During these 10 cycles, the cluster started to move back towards the configured 3000μs transmission period until the tester messages were fault-free again.

This simple experiment illustrated the transparency of the approach for the application. Indeed, the standard communication service was provided without noticeable modification (the two frames representing the application-related communication were transmitted without any disturbance), and the additional frames generated by the tester were ignored by the application. This behavior results from the constructive integration attribute of time-triggered systems: "*If* n *nodes are already integrated, the integration of* n + 1 *node must not disturb the system*" [KB03].

From the communication service point of view, the cycle length deviation (600 ppm) of the stimuli stayed within the range tolerated by the FlexRay protocol and hence did not drive the clock synchronization out of range, therefore they do not affect the application related network traffic. In practice, this means that this test of the error detection mechanisms can be performed in simultaneously for each node of the system while the system operation is ongoing. The only requirement is a number of time slots for the tester to send its stimuli through. Since this uses existing mechanisms, it requires no changes in the nodes or system architecture.

Table 3.2.3 illustrates the improvement in coverage between the monitoring method and the transparent approach presented here. In both cases, the transmit path is periodically exercised and our tester can observe its operation. The output of the clock synchronization service is also observable through the node's transmission time. However, it is likely that normal system operation will only partially exercise the mechanism. The active stimulation performed by our tester provides a better test coverage for this service. Notice that clock synchronization relies on the correct reception of frames. For a node which adapts to the above changes properly, we can conclude that it must have properly received the tester's stimulus message(s). Otherwise, we can conclude that the receiver has encountered an error upon reception of the stimuli. One further advantage of our approach is clearly the capacity to test the node's internal error detection mechanisms within the receive path.

Functional unit	**monitoring**	**deterministic replay**
Transmit path	✓	✓
Clock synchronization	✓	✓ (improved)
Receive path (data flow)	–	✓ (indirect)
Receive path (err. detection)	–	✓

Table 3.1: Test coverage comparison

3.3 Chapter Summary

We have presented in this chapter the rationales for this work. We described our main contribution as the proposition of a framework for the transparent test of time-triggered communication systems such as TTP/C and FlexRay. We have defined *transparent* as non-intrusive with respect to resources, since the node (both software and hardware) are left unmodified. Moreover, our approach is non-intrusive with respect to system operation and enables concurrent service delivery. This is a major requirement for online testing and qualifies our approach for use during field operation or during maintenance.

After that, we have seen that a pure monitoring approach already provides detailed information about nodes' transmit path and global information about the receive path and the clock synchronization mechanism. A further step for our approach was to send slightly shifted frames in order to remotely stimulate the clock synchronization of each node. Using the *composability* attribute of time-triggered communication systems and the fact that the tester frames are automatically filtered out by the application, we can argue that this test approach is transparent for the system.

The last stimulation step consists of taking control of the global time using deterministic replay and sending corrupted frames to test the fault detection mechanisms. However, this presents the following challenges:

- *controllability*: the proposed replay operation takes control of the global time. However, the test node is required to send several frames within a communication cycle and does not implement clock correction. We have to show that this approach is not a threat for the system and that the clock synchronization mechanism still provides its services with a comparable quality.
- *observability*: we argue that the system presents an observable reaction after the replay operation has finished (or while corrupted frames are sent). We have to show whether or not this behavior can be expected in any case and we must explore the bounds of this reaction (how much should the replay operation drift to drive the system out of this interval, how fast the system returns back to a stable state)
- *quality of service (QoS)*: metrics are strongly desirable in order to measure the quality of the services delivered during test operation. Nodes' local information such as offset correction or logical clock rate, and global information such as system precision are required to measure the effects of our approach and interpret the system reaction. The information should be gathered without any support from the nodes.

Chapter 4

Transparent Test Approach Validation

By three methods we may learn wisdom:
First, by reflection, which is the noblest;
second, by imitation, which is the easiest;
and third by experience, which is the bitterest

Confucius (around 551 – 479 B.C.)

This chapter is devoted to the analysis of our transparent test approach. Since our test approach relies on the distributed clock synchronization mechanism, we propose first a model for oscillators and review clock synchronization algorithms. Based on this information, we analyze the *controllability* of our approach and show that the proposed deterministic replay operation is not a threat for the clock synchronization algorithm or for the system in general. Another objective is to investigate the *observability* of our approach and to remotely detect whether the tester's frames have been taken into account for the clock synchronization or not. This information enables us to draw conclusions on each node's receive path and, more specifically, to test whether the tester frames have been processed correctly. Finally, we propose two methods to remotely measure the quality of service of the clock synchronization mechanism. These metrics are used to measure the influence of our test approach (and can be further used for evaluating the current system performance).

4.1 Definitions and System Model

The aim of this section is to introduce the background information required for this study. Our approach relies on clock synchronization and this algorithm aims at correcting drifting oscillators. Therefore, we start with proposing a model for quartz, then we review clock synchronization algorithms and finally we introduce the terms and concepts required further on in this work.

4.1.1 Quartz Modeling

We have seen in Section 2.1.4 that each node is a self-contained computer with its own (imperfect) quartz. However, quartz elements are very sensitive to their environment and there are different reasons why a quartz might drift from its nominal frequency. An interesting overview of phase noise in oscillators is provided in [Rub05]. We now propose a model based on the one presented in [GLS06], with f_p^0 as the nominal frequency of node p's quartz and $\rho_p(t)$ its *relative frequency deviation* such as $\rho_p(t) = \frac{f_p(t)}{f_p^0} - 1$.

$$f_p(t) = f_p^0 \cdot [1 + \rho_p^i + \rho_p^a(t) + \rho_p^n(t) + \rho_p^e(t)] \tag{4.1}$$

Here ρ_p^i is the initial frequency deviation at start-time, $\rho_p^a(t)$ considers the ageing effect, $\rho_p^n(t)$ the jitter due to short-term noise and finally $\rho_p^e(t)$ the jitter due to the environment. The short-term noise $\rho_p^n(t)$ has a typical magnitude of 10^{-8} to 10^{-12} [Com97, Sch95]. Parallel to this, the major influence of the environment jitter $\rho_e(t)$ is the temperature. Experiments show that in case of large temperature variation jitters in the order of 10^{-6} to 10^{-5} are possible [Com97, Sch95]. Additional influences are the stability of the power source, and mechanical effects such as shock and vibration (in the order of 10^{-9} per G [Com97]). The ageing effect $\rho_a(t)$ increases in the order of 10^{-7} per month [Com97]. Finally, the systematic deviation ρ_i represents the initial frequency when restarting an oscillator and can grow up to a magnitude of 10^{-5} [Sch95]. The relative frequency deviation $\rho_p(t)$ is then defined as the sum of the different effects.

We will make two assumptions in the following: First, for a given time window, the relative frequency deviation is bounded by a constant Π such that:

$$\forall t \ \ |\rho_p(t)| < \Pi \tag{4.2}$$

Secondly, we will assume for a small time window and for a nearly constant environment (minor changes in temperature and acceleration) that the initial offset ρ_p^i dominates the other effects (with these assumptions, the magnitude of ρ_p^i is at least a factor of 100 larger than the other effects). This means that both the stochastic jitter and the short-term variation are assumed to be negligible in comparison to the long-term variation. Similar assumptions have been made in [Sch98] (Assumption 4.1): *the* *stability condition bounds the variation of the instantaneous frequencies during* $\Delta t \geqslant 0$ *for any* $t \geqslant t_0$*, where* σ *is the oscillator stability*:

$$|\frac{f_p(t+\Delta t)}{f_p(t)} - 1| \leqslant \sigma \Delta t \tag{4.3}$$

Numerical examples illustrated for a stable environment (no more than 1° C temperature deviation within 10 sec.) that the short-term variations are negligible compared to the initial quartz deviation ρ_p^i [Sch98].

In the context of the digital systems considered in this work, the quartz oscillator delivers a pulse (*microtick*, see Section 2.3.4) which has been digitized in the value domain and alternatingly delivers a logical zero and a logical one. The length of the k^{th} microtick for node p is denoted $mic_p(k)$. With a reasonable stability σ we may assume the frequency to be constant during one microtick, such that $mic(k) = \frac{1}{f_p(k)}$ with $f_p(k)$ being the (constant or averaged) frequency during the k^{th} microtick. We can define the *quartz' relative period deviation* $\delta_p(k)$ such that $mic_p(k) = mic_p^0 \cdot [1 + \delta_p(k)]$, with mic_p^0 being the nominal period. We obtain by identification $\delta_p(k) = \frac{-\rho_p(k)}{1+\rho_p(k)}$. If we now assume $\rho_p(k) << 1$ we obtain $\delta_p(k) \simeq -\rho_p(k)$. Not surprisingly, the relative frequency- and period deviation have approximately the same amplitude but are of opposite sign. For the following, we will use the term of *oscillator deviation* for $\delta_p(k)$.

4.1.2 Definitions

Each node p implements a physical clock for measuring time [DHSS95, Kop97]. This physical clock consists of a counter $C_p(t)$ that is incremented by one every microtick, and of a physical oscillation mechanism (usually a quartz) that generates the microticks. This counter is furthermore periodically adjusted to build a logical clock $L_p(t)$ that is synchronized with the rest of the system (see Section 4.1.3 for a discussion about clock synchronization algorithms). Notice that the oscillator deviation $\delta_p(k)$ at node p is the cause of the *drift rate* ρ_k^p of the physical clock p at microtick k as defined in [Kop97] p.49. This is also the cause of the *rate of drift* of physical clocks as defined in [ST87]. The accuracy denotes *the maximum offset of a given clock from the external time reference during the time interval of interest* in [Kop97] p.50. It is defined as a *narrow envelope of real-time* for the clocks in [ST87]. In both definitions, accuracy represents a (bounded) distance between a clock under observation and a reference time source (real-time). The oscillator deviation definitely influences the physical clock, which in turn deviates from real-time. The accuracy of the physical- and logical clocks thus depends on the oscillator deviation.

The length (duration) of one communication cycle (re-synchronization period) for a node p is given by the number of microticks counted by the node's logical clock multiplied by the duration of each microtick. If we define $L_p(t_p^m)$ (resp. $L_p(t_p^{m+1})$) as the counter value of node p at the beginning of cycle m (resp. $m+1$), the length $cl_p(m)$ of communication cycle m is then:

$$cl_p(m) = t_p^{m+1} - t_p^m = \Sigma_{k=L_p(t_p^m)}^{L_p(t_p^{m+1})} mic(k) \tag{4.4}$$

The number of microticks to count for communication cycle m is given by the configured cycle length CL_0 adjusted by the correction term $CORR_p(m)$ that has been computed during the previous communication cycle by node p. If we further consider the quartz deviation to be nearly constant during two consecutive communication cycles, we finally obtain:

$$cl_p(m) = (CL_0 + CORR_p(m)) \cdot mic_0 \cdot (1 + \delta_p(m)) \tag{4.5}$$

Similar to Section 4.1.1, we define the *logical clock's deviation* $\delta_{L_p}(m)$ for a node p as the actual length of the communication cycle m divided by the theoretically ideal cycle length minus one (for the FlexRay protocol, whose correction is based on double cycles, the average of two consecutive cycles can be used). The logical clock's deviation is influenced both by the deviation of its oscillator and by the correction value.

$$\begin{aligned} \delta_{L_p}(m) &= \frac{cl_p(m)}{cl_0} - 1 = \frac{cl_p(m)}{CL_0 \cdot mic_0} - 1 \\ \delta_{L_p}(m) &= \delta_p(m) + \frac{(1 + \delta_p(m)) \cdot CORR_p(m)}{CL_0} \end{aligned} \tag{4.6}$$

Assuming the system to be synchronous, then all the (correct) logical clocks are at most π apart from each other. Using this property, we can state that the measurement of the same time interval (and more especially of one communication cycle) by any two nodes is at most 2π apart (start and end measurement both introduce an uncertainty of at most π). For the specific case of a communication cycle of length cl_0 our interval measurement therefore yields a *relative* precision of $\chi = \frac{2\pi}{cl_0}$.

We define the *global time* as the set of fault-free logical clocks within the system. Notice that this value is defined within a maximum jitter of π representing the precision between the nodes within the system. Its relative precision deviation is χ. Evaluating the global time with respect to real time is known as evaluating the accuracy of a clock synchronization algorithm. An important limitation is provided in [ST87]: The accuracy of a synchronized system is bounded by the accuracy of the underlying physical clocks.

We define further the *clock deviation interval* (CDI) as the interval between the current largest and the smallest oscillator deviations within the system ($\pm\chi$). Figure 4.1 illustrates our definition applied to an exemplary oscillator drift. During the first phases the oscillators are drifting with moderate increase or decrease of drift. Then, nodes 3 and 4 fail and the quartz deviation interval suddenly becomes narrower.

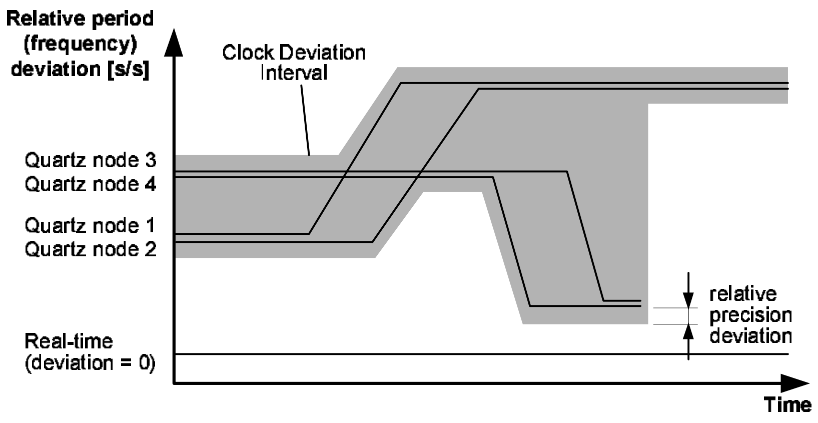

Relative period (frequency) deviation [s/s]
Clock Deviation Interval
Quartz node 3
Quartz node 4
Quartz node 1
Quartz node 2
relative precision deviation
Real-time (deviation = 0)
Time

Figure 4.1: Clock Deviation Interval (CDI)

4.1.3 Clock Synchronization

Overview

Clock synchronization algorithms are required to provide a commonly agreed time base within the system and correct the local oscillator drifts. Several important properties are listed in [LWL88, ST87]:

$P_{cs}1$**: Agreement** (or *precision enhancement* [Sch87]): Correct logical clocks within the system are at most π from each other, where π is the *precision* of the system.

$P_{cs}2$**: Bounded adjustment**: The correction term at re-synchronization is bounded.

$P_{cs}3$**: Accuracy**: The logical clock of a correct node is within a narrow envelope of real-time. This property is equivalent to the $(\alpha_1, \alpha_2, \alpha_3)$ validity from [LWL88].

We refer to an algorithm as an *internal clock synchronization* when its purpose is to maintain the maximal clock state deviation bounded by a precision π between any two nodes of the system. *External clock synchronization* aims at synchronizing a node with an external time standard (such as e.g. UTC or GPS). Note that an externally synchronized system is also implicitly internally synchronized. The contrary is not always true.

Different structures are available [Hor04]. *Asymmetric (master-slave) structures* are typically using one master node to provide the time reference to the other nodes (slaves). Their main advantage is the low cost in terms of algorithm complexity and number of messages exchanged. However the master represents a single point of failure, a fact that can not be tolerated when high reliability is required. In *symmetric structures*, each active node executes the same clock synchronization algorithm. This improves the system robustness at the cost of resources (computing and bandwidth). *Hierarchical structures* also exist, where the synchronization strategy is spread at different levels.

Three computation steps common to all synchronization algorithms have been highlighted in [AP98]: (a) *synchronization event detection*, (b) *remote clock estimation* and (c) *clock correction*.

The aim of the synchronization event detection is to determine a point in time for triggering a re-synchronization. This event should occur approximately simultaneously on all nodes. Two main approaches exist: First, the computation can be triggered when the internal clock reaches a given value (usually a multiple of the cycle length). This approach supposes an initially synchronized system. The second method is to trigger the re-synchronization by the reception of one (or more) message(s). With this second approach, the precision of algorithms depends on message transmission latencies and communication delay jitter.

The goal of remote clock estimation is to provide information about the clock values within the system. Two main approaches exist: The *Time Transmission* (TT) technique, where the node autonomously transmits (broadcast) its local clock value, and the *Remote Clock Reading* (RCR) technique, where a node p sends a request to another node q to read its clock. The main advantage of the RCR technique is the possibility of assessing information on the transmission delay and jitter. However, this method requires more bandwidth.

The last computation step is the clock correction itself. On one hand, *convergence-averaging* or *convergence function* based techniques use a set of remote clock estimates to compute a correction term [AP98]. *Convergence-nonaveraging* techniques, on the other hand, only use the fact that a given number of messages have been correctly received in order to perform the correction. Convergence functions might be deterministic or probabilistic. *Deterministic* functions give a guarantee on the system precision as long as their assumptions (e.g. wrt. quartz drift, number of faults within the system) are fulfilled. *Probabilistic* functions, however, provide a probabilistic guarantee that might fail to hold sometimes but with a known or bounded failure probability [Arv94].

Offset correction aims at correcting the accumulated clock state difference between the nodes within the system. This mechanism is thus required to reach an agreement on the global time base. The clock correction value can be applied *discretely* or *continuously* (amortized). While a discrete adjustment causes a discontinuity in the time base, a continuous adjustment modifies the rate of the logical clock during a given time interval. Notice that both methods are equivalent as long as the entire correction can be applied within a re-synchronization period [SC90].

The main goal of the *rate correction* is to artificially modify the frequency of the nodes logical clocks in order to minimize the future clock state differences. This mechanism, while not mandatory for synchronization, usually improves the effective precision of the system.

Time-Triggered Communication

Both TTP/C and FlexRay implement convergence-averaging clock correction and remote clock estimation based on the time transmission technique. Hence, for both protocols, a subset of nodes (e.g. with better oscillators) transmit *sync* frames (normal frames with the synchronization flag set), thus actively taking part in the clock correction process. The other nodes are passively synchronizing to the resulting global time. Re-synchronization is triggered when the node's local time base reaches a pre-configured value.

The TTP/C protocol foresees a four entry stack to store the remote clock measurements. Offset correction based on the fault tolerant average (FTA, see [KO87]) convergence function is computed with these four entries, where in practice the largest and smallest are discarded to average the remaining two. The correction value is applied continuously. The FlexRay protocol, on the other hand, implements both rate and offset correction based on the Fault Tolerant Midpoint (FTM, see [LWL88]) computation. This protocol foresees a sixteen entry stack, which is cleared at the beginning of each second cycle. Depending on the number of sync frames received, the k ($k \leq 2$) largest and k smallest measurements are removed. The remaining largest and smallest entries are finally averaged to compute a rate and an offset correction term.

The ability to persistently tune the frequency of the local time base to a value that is essentially different from the original oscillator frequency introduces the risk of a common mode drift of the global time base (precision is established when the nodes agree on *any* frequency). In order to avoid such a common mode drift, the Flexray protocol implements a *damping factor*, that locally tries to drive the rate correction back to zero. Consequently, each node tries to move its logical clock (and thus the approximately synchronized global time) to its own oscillator frequency. Figure 4.2 highlights the function of the damping factor in FlexRay for the node's rate calculation. The local rate correction (*vRateCorrection*) is first updated using the fault-tolerant midpoint of the measured rate differences. Then, depending on the result, the damping factor *pClusterDriftDamping* is applied to move the rate correction towards zero. The FlexRay specification [Fle05] recommends to configure the damping factor to approximately the same value for all nodes. Its range is limited from 0 to 20 microticks, and thus in every case smaller than the granularity of the global time base (macrotick, at least 1 microsecond or 40 microticks).

Impairing factors

Different effects are affecting the synchronization quality in a distributed system. Naturally, both the oscillator stability and the re-synchronization period have a direct impact on the system precision, but also the following factors are playing a role:

Clock reading error: In convergence-average clock synchronization, each node

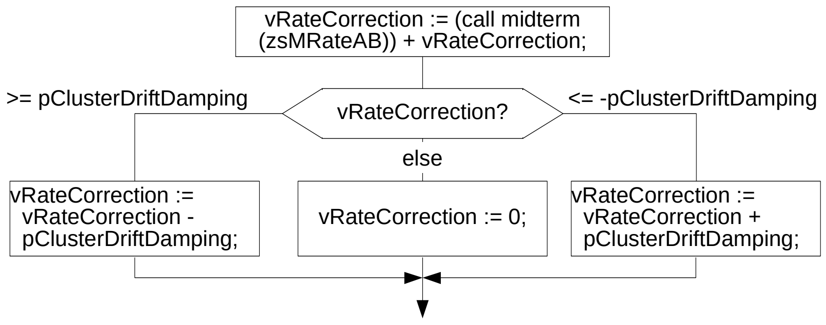

vRateCorrection := (call midterm (zsMRateAB)) + vRateCorrection;
>= pClusterDriftDamping
vRateCorrection?
<= -pClusterDriftDamping
else
vRateCorrection := vRateCorrection - pClusterDriftDamping;
vRateCorrection := 0;
vRateCorrection := vRateCorrection + pClusterDriftDamping;

Figure 4.2: Damping factor for the FlexRay protocol (from [Fle05] Fig. 8-15)

needs to read the local time of all other nodes, in order to compute its correction value and correct its time base. The *clock reading error* represents the difference between the measured and the actual clock state of a remote node at re-synchronization. There are two main reasons for this phenomenon. A first reason is the jitter during the communication (due to e.g. internal processing, quantization error, queuing or non-deterministic communication schemes). This can be minimized using deterministic communication schemes and time stamping the frames at a low level and with a high granularity (as in TTP/C and FlexRay). A second source of errors shows up when the clock readings are done long before the re-synchronization point. Hence, at re-synchronization, the actual clock state differences might be larger than the ones previously measured (the nodes have further drifted apart).

System history: State correction based clock synchronization algorithms compute their adjustment term according to the clock state differences measured during the previous cycle(s) only. Some algorithms, however, do implement rate correction and are additionally using a history of the system. One such example is FlexRay, where the last rate correction term is used for the computation of the new one. Consequently, the correction term (and thus the global time) not only depends on the current physical clock drifts but on the system history, too.

Faults within the system: For example, Byzantine clocks (presenting different values to the different nodes) might let two correct nodes drift apart. A faulty quartz (drifting apart its bounds) might let the logical time of a remote node drift beyond some bounds too, thus inhibiting the application to perform its tasks on time.

4.2 Safe Deterministic Replay Operation

The aim of this section is to study the controllability of our approach and to answer the following questions: Is it possible to remotely control the global time? Is this operation safe? We have proposed in [AFS08] a formal proof for the correctness of replay operation in TTP/C systems that is based on the Schneider's proof [Sch87]. Unfortunately this proof does not apply to FlexRay (in particular it does not allow for rate correction), and we are not aware of any other formal proof for the clock synchronization in FlexRay either. Therefore, we present here a more informal approach, in order to compute the precision of the system according to the attributes of the replay operation and to show that the agreement property $P_{cs}1$ of Section 4.1.3 holds (the system does not drift apart).

4.2.1 Deterministic Replay Operation

In a distributed system, the global time is a function of the nodes' oscillator deviations and of the system history. We introduce in this work the notion of *deterministic replay* as a method to drive the distributed clock synchronization algorithm into a desired state and to observe the system reaction. More precisely, we aim at influencing the clock synchronization mechanism and dictating a given behavior. For that we use a tester node with the capability to send a pre-defined bus traffic (containing more than one sync frame per communication cycle or round), regardless how the other nodes react.

Our tester sends N frames per cycle with $N > f$, where f represents the number of faults that can be tolerated by the algorithm. In general, the tester will influence the clock synchronization with a factor $\frac{N-f}{M}$ where N represents the number of frames sent by the tester, f the fault tolerance grade of the system (and on the same time the number of extremum rejected before the computation of the algorithm, usually the f slowest and the f fastest) and M the number of nodes averaged by the convergence function. Note that we focus here on FlexRay or TTP/C clock synchronization where the convergence function is averaging exactly two values ($M = 2$, see Section 4.1.3). In our case, $N = f + 1$ frames are sufficient to influence the clock synchronization with a factor of 1/2. For example, a system of four nodes can tolerate $f = 1$ fault and the tester then sends two sync frames per cycle. These frames are delayed and will be thus taken into account as one extremum of the convergence-average calculation. The second extremum is provided by the nodes themselves. Since we can assume that the nodes are tightly synchronized to another, they will correct half the clock state difference to the tester at each re-synchronization. The tester does not correct its time base, but the node does, and consequently the nodes will follow the tester's logical time.

Figure 4.3 illustrates deterministic replay operations. Six frames are transmitted in this example: four from standard nodes (N0, N1, N2, N3) and the two last (T1, T2) from a tester node. The standard nodes are tightly synchronized to each other, while the tester presents a larger deviation to the node. Notice that the tester frames are syntactically and timely correct (within the slot boundaries), otherwise the frames

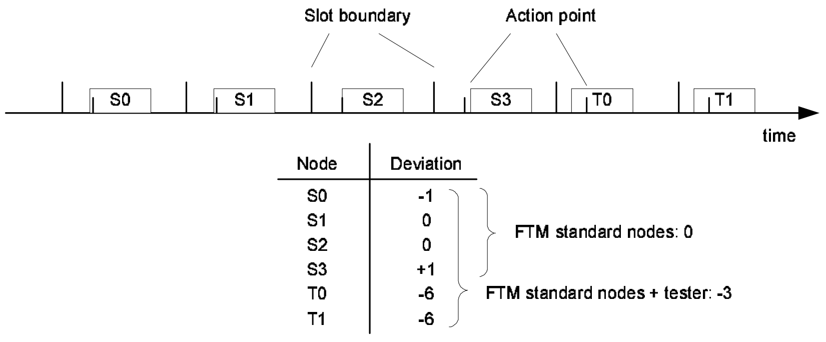


Figure 4.3: Deterministic replay

would have been removed from the clock's correction computation. The convergence function (here the Fault Tolerant Midpoint, FTM) applied to the four standard nodes returns a correction of zero. However, when the tester is taken into account, the correction result is shifted to -3, which represents half the clock state difference between the standard nodes and the tester.

An important question here is whether this deterministic replay operation can be a threat for the distributed clock synchronization algorithm. In fact, the tester sends $f+1$ frames per communication cycle (instead of one) and does not implement a clock correction mechanism as required. We will see in the following section under which conditions the tester operation can be qualified as safe and therefore whether or not it is a threat for the system. Some limitations are further experimentally evaluated in Section 5.3.

4.2.2 Computation of the System Precision

Our approach is based on the analytical computation of the system precision. We aim at finding some bounds to guarantee the agreement property of Section 4.1.3. Figure 4.4 illustrates replay operation. A tester node performs a deterministic replay (and thus does not execute clock correction) with an increasing deviation, while a standard node performs clock correction and tries to follow the tester.

Our analysis is based on the following assumptions:

(B1) *The tester is initially synchronized to the system under test.* This can be performed with every kind of synchronization algorithm before starting the replay.

(B2) *The oscillator deviation of a standard node is bounded and constant during the replay activity.* Here we refer to the quartz model from Section 4.1.1, and assume a very small σ in Equation (4.2) for the moment, which is not unrealistic in practice.

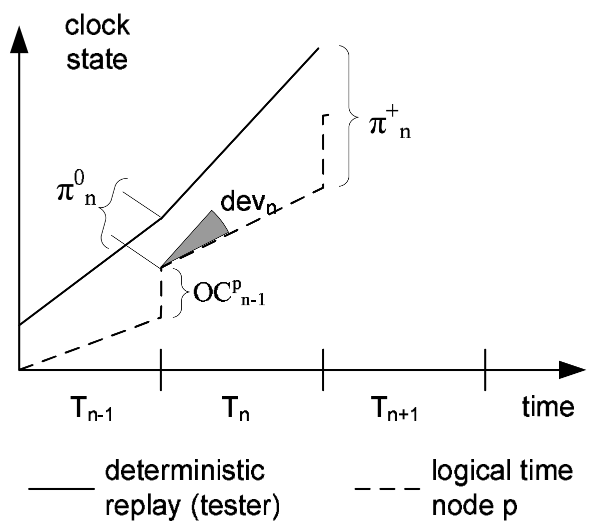


Figure 4.4: Precision computation

(B3) *Tester nodes do not fail.* This assumption simplifies the reasoning, but a fault tolerant tester architecture can be proposed as well.

(B4) *Typical replay operation*: The standard nodes are supposed to form a tightly synchronized clock group while the tester (i.e., the two virtual testers) is forming a second tightly synchronized clock group (see Section 4.2.1).

In the following we will treat time and microticks in the same way, i.e., we will assume for the sake of simplicity that logical clocks have unit $[ns]$. We define π^0_n as the system precision, i.e., the maximum difference of any two correct nodes' logical clocks, at the beginning of cycle n (just after re-synchronization) and π^+_n as the system precision at the end of cycle n (before the $(n+1)^{th}$ re-synchronization). In case of convergence-average correction, for the purpose of re-synchronizing, every node p subtracts an offset correction term from its current logical clock value at the end of each round. In the following, node p's offset correction term at cycle n will be denoted by OC^p_n. Note, that each node's correction term OC^p_n can be expressed as a fraction γ_p of the system's clock state difference, i.e., its precision, at cycle n (with $|\gamma_p| \leq 1$)

$$OC^p_n = \gamma_p \cdot \pi^+_n \tag{4.7}$$

For each cycle there exists a node M (respectively a node m) which maximizes (respectively minimizes) the offset correction. Thus we have:

$$OC^M_n = \alpha \cdot \pi^+_n \tag{4.8}$$
$$OC^m_n = \beta \cdot \pi^+_n \tag{4.9}$$

We further define T as the nominal re-synchronization period (in $[ns]$) and dev_n as the maximum rate deviation between any two correct nodes' logical clocks, i.e., $T \cdot dev_n$ gives the time interval any two correct processes may diverge in a single round. Due to assumption (B4), during the process of testing one extreme is always formed by a tester while the other extreme is formed by a standard node i, yielding $dev_n = \rho_{L_i}(n) - \rho_{L_{tester}}(n)$, with $\rho_{L_i}(n)$ and $\rho_{L_{tester}}(n)$ being the logical clock deviation of standard node i and of the tester at cycle n, respectively. With this in mind we obtain:

$$\pi_n^+ = \pi_n^0 + T \cdot dev_n \tag{4.10}$$
$$\pi_n^0 = \pi_{n-1}^+ - (OC_{n-1}^M - OC_{n-1}^m) + \epsilon \tag{4.11}$$

where ε is the error made by offset correction (including quantization and the error made by measuring the other node's clocks). Equations (4.10) and (4.11) describe the system precision behavior. The precision π_n^+ at the end of a cycle is given by the precision at the beginning of this cycle plus the deviation of the logical clocks during the period T. Concerning the term π_n^0, the precision after re-synchronization is provided by the precision before re-synchronization minus the difference in correction performed by the outliers M and m. The precision can be re-written as the following series:

$$\pi_n^+ = (1 - (\alpha - \beta)) \cdot \pi_{n-1}^+ + T \cdot dev_n + \epsilon \tag{4.12}$$

In the following we compare two scenarios: **(i)** *standard operation*, where all nodes are standard nodes and thus both M and m apply offset correction and **(ii)** *replay operation* where the tester (either M or m) does not apply offset correction:
ad (i) In this case M and m correct part of the precision, s.t., $OC_{n-1}^M - OC_{n-1}^m$ in Equation (4.11) yields π_{n-1}^+, i.e., $\alpha - \beta = 1$. Typically both m and M correct half the precision, i.e., $\alpha = -\beta = 1/2$. Equation (4.12) thus becomes:

$$\pi_n^+ = T \cdot dev_n + \epsilon \tag{4.13}$$

ad (ii) Wlog. assume that m is the tester node (which does not apply offset correction). Thus $\beta = 0$, yielding:

$$\pi_n^+ = (1-\alpha)^n \cdot \pi_0 + T \sum_{i=0}^{n-1} (1-\alpha)^i dev_{n-i} + \epsilon \sum_{i=0}^{n-1} (1-\alpha)^i \tag{4.14}$$

Remember that $lim_{n \to \infty} \sum_{i=0}^{n} \phi^i = \frac{1}{1-\phi}$ for $|\phi| < 1$. Thus for growing n, we finally obtain

$$\pi_n^+ = \frac{1}{\alpha} \cdot \epsilon + T \sum_{i=0}^{n-1} (1-\alpha)^i dev_{n-i} \tag{4.15}$$

Notice, that in case of a constant logical clock deviation ($dev_{n-i} = k$) precision π_n^+ converges to $\frac{1}{\alpha} \cdot Tk + \frac{1}{\alpha} \cdot \epsilon$, i.e., is worsened by the factor $\frac{1}{\alpha}$ in comparison to the precision of a system containing only standard nodes (Equation 4.13). Hence, instead of having all the nodes correcting the entire clock state difference (as in (i)), in (ii) some nodes are correcting α (with $|\alpha| < 1$) of the current clock state difference. However, the offset correction term is expected to exactly correct the clock state difference accumulated during a cycle for both (i) standard and (ii) replay operation; otherwise the system would drift apart. In both scenarios we have the same logical clock deviation, the same convergence function and the same final cumulated offset correction. However, the asymmetry of system (ii) leads to a larger clock state difference and thus to a worsened precision. Though worsened, the system precision can still be computed (and bounded), if the logical clock deviation dev_i, which itself depends on the tester's replay attributes, is known. Consequently, we can argue that deterministic replay is not a threat for the system if and only if this operation is precisely controlled. The next section presents numerical examples for different kinds of replay operations.

4.2.3 Numerical Examples with Different Replay Attributes

Definitely dev_n differs whether a rate correction mechanism is instantiated or not, as a rate correction leads to an adaption of ρ_{L_i} towards $\rho_{L_{tester}}$ and thus to a decreasing dev_n. In the following we explore two test scenarios: (i) The tester makes a step in its deviation, i.e., $\rho_{L_{tester}}(0) = \rho_{L_m}(0)$ and $\rho_{L_{tester}}(n > 0) = \rho_{L_m}(0) + \Delta$. (ii) The tester increases its deviation in a linear manner, i.e., $\rho_{L_{tester}}(0) = \rho_{L_m}(0)$ and $\rho_{L_{tester}}(n) = \rho_{L_m}(0) + n\theta$. We will study the effect of both (i) and (ii) on systems which perform (a) offset correction only (e.g. TTP/C) and those which perform (b) offset plus rate correction (e.g. FlexRay).

We assume moreover that the standard node contributes for half of the offset correction ($\beta = -\frac{1}{2}$) and define the *virtual oscillator drift* ρ_{vir} as the oscillator drift required by a standard node to reach the same logical clock deviation as a tester, which does not apply any correction. The motivation is to find an equivalence between a standard node whose oscillator is drifting (with ρ_{vir}) and our tester. We will see that the deviation of the logical clocks are the same, but the system precision is worsened by a factor $\frac{1}{\beta}$. The virtual deviation of round n is computed as the sum of the tester's deviation in round n plus the deviation that emerges from the missing tester's correction (offset & rate).

(i.a) Step with offset correction: The deviation of the standard nodes' logical clocks equals their physical deviation ($\rho_{L_i}(n) = \rho_i$), which is constant by assumption (B2). Therefore, we obtain $dev_n = \rho_i - \rho_{L_{tester}} = \Delta$. The correction term then converges to $T\Delta$ and using Equation (4.15) system precision converges to $2T\Delta + 2\epsilon$. The virtual oscillator drift is $\rho_{vir}(n) = \rho_{L_{tester}}(0) + 2\Delta$.

(i.b) Step with offset & rate correction: Contrary to the first case, the standard nodes' logical clock rates are continuously corrected. We have $dev_0 = 0$, $dev_1 = \Delta$ and $dev_{n>1} = \frac{1}{2}dev_{n-1}$. We can easily see that the standard node's logical clock rate

converges to the tester's logical clock rate. We obtain $dev_n = \kappa \ll \rho_i - \rho_{L_{tester}}$. The system precision then converges to $2T\kappa + 2\epsilon$ (logical clock deviation applied to the re-synchronization cycle). Note, that (i.b) definitely yields a better precision as $0 \approx \kappa \ll \Delta$. Virtual oscillator drift is $\rho_{vir}(n) = \rho_{L_{tester}}(0) + 2\Delta + 2\kappa$.
(ii.a) Ramp with offset correction: Using the fact that $dev_{n-i} = (n-i)\theta$ and that $\sum_{i=0}^{n-1}(n-i)(\frac{1}{2})^i \approx 2n-2$ for large n, we can see that the precision converges to $\pi = T\theta(2n-2) + 2\epsilon$. Note, that π is bounded iff n and θ are bounded. This confirms the intuitive result that the ramp must be bounded both in time and value (slew rate). The virtual deviation is then $\rho_{vir}(n) = \rho_{L_{tester}}(0) + 2\theta(n-1)$.
(ii.b) Ramp with offset & rate correction: In contrast to (ii.a), the standard node's logical clock rate is following the tester's logical clock rate. More specifically, if we assume that the standard node's rate correction mechanism corrects half the rate difference we get $dev_{n>0} = \frac{1}{2}dev_{n-1} + \theta$. The logical clock deviation between tester and standard nodes finally converges to $dev_n = 2\theta$. The correction term at cycle n thus converges to $2T\theta$ and the system precision to $\pi = 4T\theta + 2\epsilon$. For virtual oscillator deviation we obtain $\rho_{vir}(n) = \rho_{L_{tester}}(0) + \sum_{i=0}^{n} dev_i = \rho_{L_{tester}}(0) + 2\theta n$. We will see later that the FlexRay clock synchronization algorithm implements a *damping factor* that slightly shifts the result of the rate correction towards the nominal frequency of the underlying oscillator. This does not invalidate the agreement attribute but slightly worsens the precision.

Table 4.1 summarizes the results achieved. As expected, the precision of the system is bounded for a "bounded" replay operation (bounded deviation). Furthermore, it makes a difference for the precision whether rate correction is used or not. Interestingly, the virtual oscillator drift is approximately the same with and without rate correction.

correction	**step**	
	π	ρ_{vir}
offset	$2T\Delta + 2\epsilon$	$\rho_{L_{tester}}(0) + 2\Delta$
offs & rate	$2T\kappa + 2\epsilon$	$\rho_{L_{tester}}(0) + 2\Delta + 2\kappa$

correction	**ramp**	
	π	ρ_{vir}
offset	$2T\theta(n-1) + 2\epsilon$	$\rho_{L_{tester}}(0) + 2\theta(n-1)$
offs & rate	$4T\theta + 2\epsilon$	$\rho_{L_{tester}}(0) + 2\theta n$

Table 4.1: Convergence results

4.2.4 Experimental Validation

In the previous sections we have proposed a strategy for taking control of the global time via a tester node. We have further proven that this does not jeopardize normal system operation, as long as it is performed with care, i.e. certain conditions are met.

In the following discussion, we are going to give experimental validations for these claims, thus substantiating that our approach indeed works in practice.

The first experiment presents a simulation of a 5 nodes TTP/C network performed with the SIDERA simulation tool [Han06]. In our experiment the round length was set to 1000 μs (five slots of 200 μs each). Nodes S1, S2, T0 and T1 transmit sync frames and thus influence the clock synchronization, while S0 only passively synchronizes to the global time. The oscillator deviations of S0, S1 and S2 were set to constant, while those of T0 and T1 are both increased by $50e^{-6}$ each cycle. Note that T0 and T1 are standard nodes and thus are playing the "virtual nodes" as described previously. According to Table 4.1, we expect then the global time deviation to be approximately half the deviation of T0 and T1.

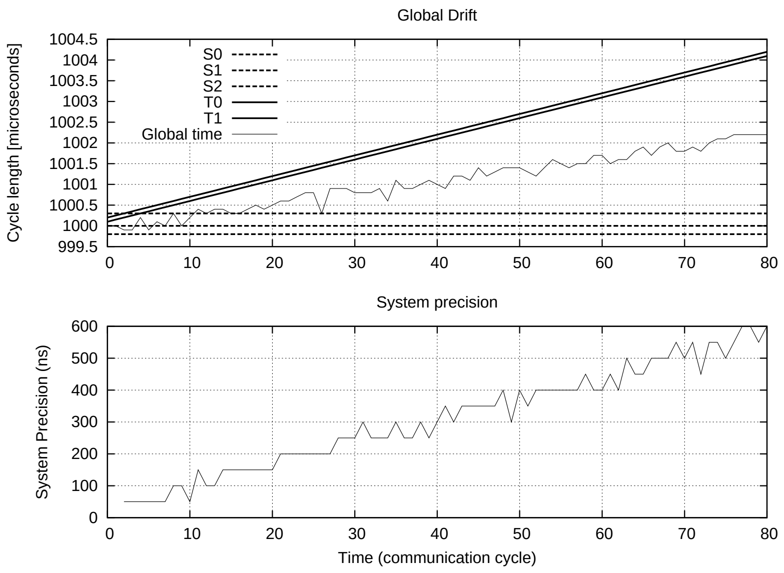


Figure 4.5: Drift and precision with TTP/C

Figure 4.5 illustrates the results. The upper diagram shows as expected that the global time approximately follows half way the drift imposed by the nodes T0 and T1. We are in the typical case where the groups (TO, T1) and (S1, S2) both influence for half the correction term. The lower diagram shows the linear dependence of the system precision with respect to the deviation, as predicted by Table 4.1. Therefore, as long as we keep the tester's clock deviation within bounds (i.e. stop increasing the ramp before Δ's borderline is crossed), the precision will remain within bounds as well.

Figure 4.6 (left part) illustrates an experiment performed with a four nodes FlexRay network. Nodes 2 and 3 have a constant oscillator deviation (respectively $-150e^{-6}$ and $-250e^{-6}$) during the experiments. The oscillator deviations of nodes 0 and 1 are initialized with $-200e^{-6}$ and – starting with cycle 420 – present a linear varying oscillator deviation of $5e^{-6}$ every 3 communication cycles. Figure 4.6 (right part) illustrates the equivalent experiment performed with our tester node [FA07]. The reference nodes (2 and 3) present the same oscillator deviation as in the previous experiment.

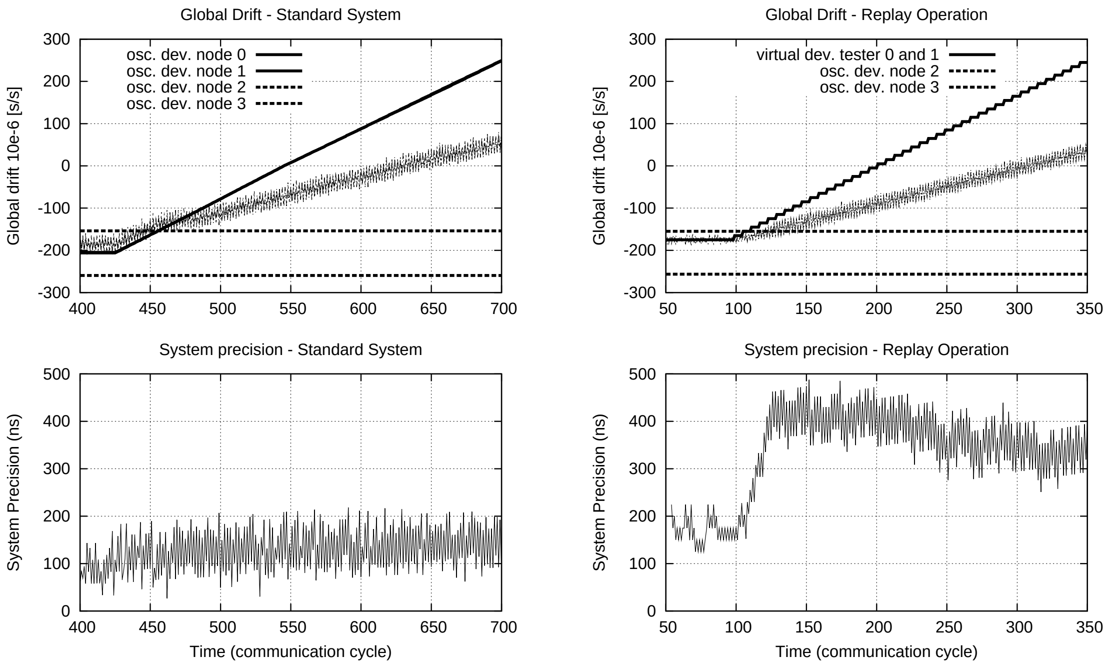


Figure 4.6: Real nodes with oscillator drift

Regarding the experiment with a standard system, several important observations can be made. First, before cycle 450, the oscillator deviations of nodes 0 and 1 are within the interval spanned by the oscillator deviations of node 2 and 3. During this interval, nodes 2 and 3 are thus rejected as outliers and the logical time exactly follows nodes 0 and 1 (these two nodes are influencing the clock correction with a factor 1). During a second time (after cycle 450), the oscillator deviations of nodes 0 and 1 are higher than the ones of node 2 and 3. Consequently, either node 0 or node 1 is now rejected as outlier while the other one influences the correction with a factor 1/2. As expected, we can observe that the global time deviation grows with half the amplitude of the oscillator deviation from nodes 0 and 1. Furthermore, the precision of the system is bounded and does not depend on the current clock deviation interval.

Regarding the replay operation, as expected, the virtual oscillator deviations computed by our tester are similar to the real oscillator drift of nodes 0 and 1. The steps observed arise from the timestamping granularity of our tester node (25ns). It is important to notice that the precision of the system is bounded for this experiment (agreement property of Section 4.1.3). This experimental measurement provides an additional check during deterministic replay operation. As expected, the precision here is approximately twice the precision of the previous experiment. Notice additionally that the maximal clock state deviation between the standard nodes (system precision when the tester is not taken into account) is below 100ns.

Moreover, the results for TTP/C and FlexRay confirm the results presented in Table 4.1. While the same oscillator deviation is required both for TTP/C (offset correction) and FlexRay (offset + rate correction) to achieve the same effect, the system precision with rate correction is bounded and does not depend on the current difference between the physical clock deviations.

4.2.5 Online Test of the Oscillator Stability

The results presented before are based on the assumption that the nodes under test present a constant oscillator deviation during the experiment time. We will now describe an algorithm that checks online whether the oscillator deviation of the nodes within the system has changed, which would invalidate our initial assumption, or not. This computation is important, in order to allow interrupting test operations before the system precision exceeds the specified bounds (and thus introduces a fault into the system). Note that this extension is required for FlexRay since the actual oscillator frequency of the nodes is masked by the rate correction mechanism. For TTP/C, the actual oscillator frequencies are directly visible through the offset correction value and thus can be easily tested online (see [AS07]).

Characterization of the offset correction

We have seen that during deterministic replay operation, the offset correction is an illustration of the instantaneous frequency difference between a node's and the tester's logical clocks. In the case of FlexRay, the nodes are implementing a rate correction and thus are correcting the rate of their logical clock. However, the deterministic replay presented here (ramp) continuously modifies the speed (frequency) of its logical time base and the nodes under test are following the tester. Their offset correction depends then (a) on the slew rate of the tester's global time and (b) on frequency variations of their respective own oscillators. Since we can control the slew rate during replay operation, the offset correction is thus a good indicator for the stability of the node's internal oscillator.

For the case of FlexRay, whose re-synchronization period is based on a double cycle basis (see Section 4.1.3), the rate deviation dev_{2n} at cycle $2n$ is given by the rate deviation dev_{2n-2} from cycle $2n-2$ plus the additional deviation of the tester $\rho_{L_{tester}}$ minus the rate correction $RateCorr_{2n}$.

$$dev_{2n} = dev_{2n-2} + 2\theta - RateCorr_{2n} \tag{4.16}$$

If we now use the fact that the rate correction at cycle $2n$ is half the rate deviation of cycle $2n-2$ minus the damping factor (see Section 4.1.3) $DampFact$, we obtain for Equation (4.16):

$$dev_{2n} = \frac{1}{2} \cdot dev_{2n-2} + 2\theta + \frac{DampFact}{T} \tag{4.17}$$

Similarly to Section 4.2.3, we obtain for the precision and for the offset correction (remember that the re-synchronization period is $2 \cdot T$ and that the offset correction is half the system precision):

$$OC_n = 4 \cdot (T \cdot 2\theta + DampFact) + \epsilon \tag{4.18}$$

Equation (4.18) illustrates the linear dependency between the logical clocks difference (θ) and the node's offset correction OC. Assuming the tester's ramp linear, a modification of the offset correction indicates a modification of the instantaneous node's logical clock difference, which in turns indicates that the node's oscillator drift has changed (remember the effect of the oscillator deviation to the node's logical clock, see Equation (4.5)).

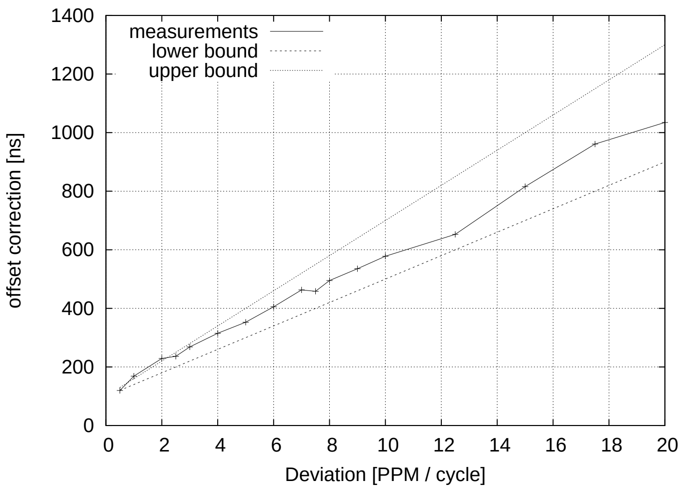


Figure 4.7: Characterization of the offset correction for different clock deviations

We have experimentally measured the correction term for different frequency differences. In total, 17 frequency differences have been measured during 200 communication cycles and each experiment has been repeated 5 times. The results are pictured in Figure 4.7. The graph compares the measured averaged offset correction (given in nanoseconds) with a lower and upper bound for different slew rate of the logical time during replay (given in PPM – or $10e^{-6}$ s/s – per communication cycle). The lower bound as been defined as $\epsilon = 0$ and the upper bound as $\epsilon = 3\theta \cdot T$. This arbitrary value represents a measurement error of θ during the rate correction (influencing the term OC_n for $2\theta \cdot T$) and of $\theta \cdot T$ due to the offset correction.

We can notice that the measurement curve is not straight-line. This is due to the limited granularity of our tester. Hence, in our configuration a slew rate of 2 PPM per cycle translates to a speed-up of the tester's logical time by two microticks every 5 cycles. This leads to a punctual step of the global time every five cycles and slightly increases the average offset correction. The next measurement (2,5 PPM per cycle), in comparison, represents a jump of one microtick every two cycles and the measurement error is minimized.

Experimental validation

During the following, we illustrate how the above approach can be used to detect non-regularity of the node's internal oscillator during deterministic replay operation. For both experiments, the tester is sending two frames (replay operation) and the system under test consists of four nodes. The two following figures illustrate the oscillator drift of the four nodes under test (part A) and a filtered (averaged) view of the node's offset correction (part B).

During the first experiment, two nodes present a constant oscillator deviation (-50 PPM) while the other two ones perform periodic jumps of their oscillator deviation (from -50 PPM to -100 PPM for node 0 and from -50 PPM to 0 PPM for node 1). As expected, it can be observed in Figure 4.8 that the (filtered) offset corrections present local jumps just after a oscillator deviation modification.

For the second experiment, three of the four nodes present a constant oscillator deviation of -50 PPM. The fourth node exhibits a linearly varying oscillator deviation between -60 PPM and -160 PPM. As expected, the offset correction amplitude of this node is increased by 3 microticks or 75ns. According to our characterization (see Figure 4.7 and Equation 4.18), this represents an additional frequency change of approximately 2 PPM per cycle.

4.3 Accuracy of the Logical Clocks

The objective of this section is to study the observability of our test approach. We want to show that the global time (set of logical clocks within the system) always converges to the *clock deviation interval* (CDI) as defined in Section 4.1.2. This attribute is

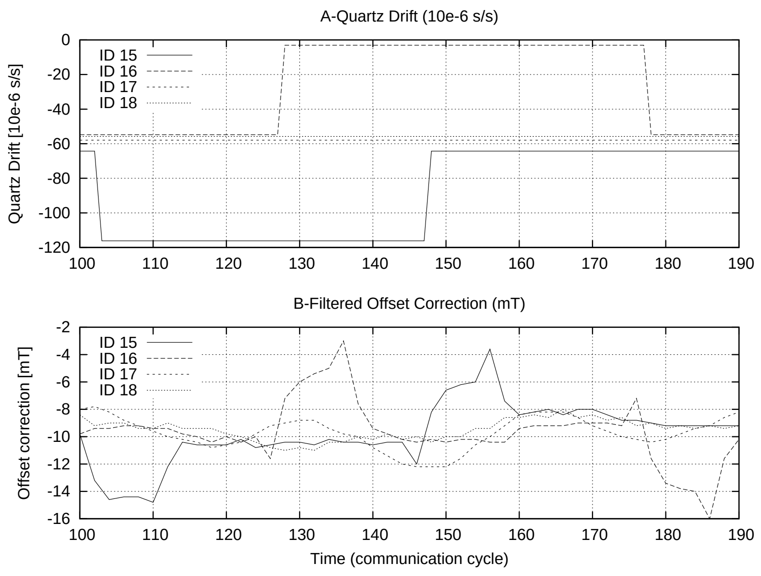


Figure 4.8: Deviation with sudden jump

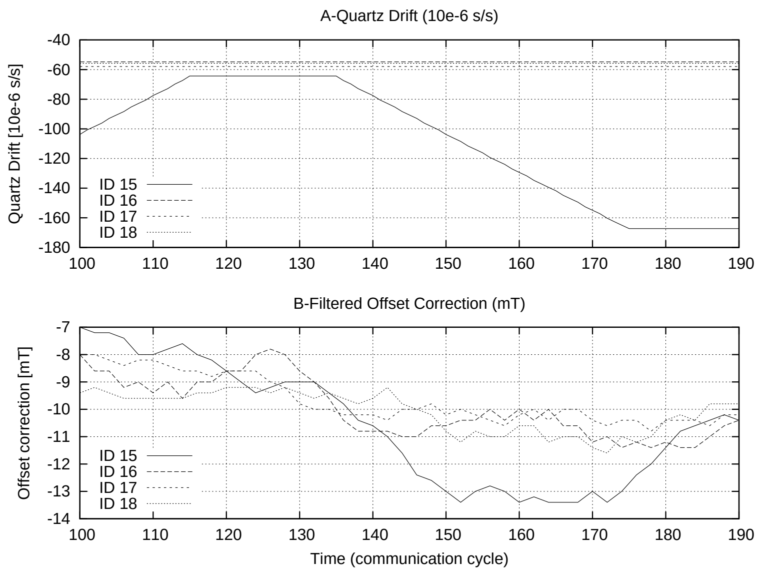


Figure 4.9: Deviation with fast drift

required for our test approach, in order to detect whether the system under test has accepted or rejected the test pattern. More specifically, we want to prove that:

P1a: if the global time deviation is initially within the CDI, it will stay within this interval as long as the CDI does not change faster than the dynamics of the synchronization algorithm allow a tracking, and that

P1b: if the global time deviation is outside of the CDI, it will return to within this interval as fast as the dynamics of its clock synchronization algorithm allows.

4.3.1 Analysis for TTP/C

This communication protocol implements state correction and the adjustment is computed as the Fault Tolerant Average (FTA) of the logical clock differences [TTT05]. The state correction is not applied instantaneously but amortized during the beginning of the communication cycle. We claim that for a fault-free system and for stable oscillator drift, the global time deviation will stay within the CDI. This means there exist at least two nodes a and b, such that for each node p, the logical clock deviation δ_{L_p} is comprised between these two oscillator deviations: ($\exists a, b \in N_{ok}$ $\forall p \in N_{ok}, \delta_a(m) - \chi \leqslant \delta_{L_p}(m) \leqslant \delta_b(m) + \chi$). To prove this claim, let's suppose at cycle m the logical clock deviation of every node p be smaller than its underlying oscillator deviation:

$$\forall p \in N_{ok} : \delta_{L_p}(m) < \delta_p(m) \tag{4.19}$$

This condition implies that the correction term $CORR_p(m)$ is negative for each node (see Equation 4.6):

$$Eq\ 4.19 \Leftrightarrow \forall p \in N_{ok} : CORR_p(m) < 0 \tag{4.20}$$

However, this correction term is computed as a fault tolerant average of the logical clock differences. This means that a node subset ($N_{FTA} \subset N_{ok}$) is selected from which the clock difference average is computed. Therefore, assuming that each correction is negative is equivalent to assuming that each single logical clock runs faster than an average of a subset of these clocks:

$$\begin{aligned} Eq\ 4.20 \Leftrightarrow \forall p \in N_{ok} : \textstyle\sum_{i \in N_{FTA}} \frac{L_i(m-1) - L_p(m-1)}{|N_{FTA}|} < 0 \\ \Leftrightarrow \forall p \in N_{ok} : \textstyle\sum_{i \in N_{FTA}} \frac{L_i(m-1)}{|N_{FTA}|} < L_p(m-1) \end{aligned} \tag{4.21}$$

This last hypothesis is mathematically false, since an average of a subset cannot be strictly less than each element of this subset. We can therefore conclude that Equation 4.19 was false as well. We would demonstrate similarly that the logical clock of each node cannot be slower than its underlying oscillator ($\forall p \in N_{ok}, \delta_{L_p}(m) > \delta_p(m)$ is falsified). Consequently, there exists at least one node whose logical clock is faster and one node whose logical clock is slower than its underlying oscillator:

$$\exists a, b \in N_{ok}: \quad \begin{aligned} \delta_{L_a}(m) &\geqslant \delta_a(m) \\ \delta_{L_b}(m) &\leqslant \delta_b(m) \end{aligned} \tag{4.22}$$

Using the agreement property ($P_{cs}1$) we have $\delta_{L_a} \simeq \delta_{L_b}$, and we can finally conclude that the global time is within the CDI as defined in Section 4.1.2. Note that this proof is not specific for TTP/C but can be generalized to every clock synchronization algorithm that uses convergence-average based state correction. More specifically, this also applies to the state correction part of the FlexRay protocol.

4.3.2 Analysis for FlexRay

This communication protocol implements both a state and a rate correction [Fle05]. While the state correction directly processes the last clock measurements, the rate correction uses an internal history for computation. The result of the rate correction is further influence by the damping factor (see Section 4.1.3), and consequently, each node tries to move its logical clock (and thus the approximately synchronized global time) to its own oscillator frequency. We will show that important properties arise from this attribute:

- When the global time is outside of the clock deviation interval (e.g. after test operation, or when some outliers have failed), the global time will move back towards the nearest oscillator deviation bound.
- The damping factor is a constant and is not dependent on the difference between the global time and the oscillator frequency. Consequently, the global time does not converge to some kind of oscillator frequency average – its actual position within the clock deviation interval depends on the system history.

Let's explore more formally our two previous statements through the analysis of three scenarios:

Scenario 1: $C_1 : \forall i \in N_{ok}\ \delta_{L_i}(m) < \delta_i(m)$. For cycle m, the oscillator deviation of each node is larger than its corresponding logical clock deviation (and thus the global time deviation is outside of the CDI). Therefore, the correction value is strictly negative (see Equation 4.6). We have seen previously that the state correction alone can not be the reason for this behavior. This implies that the rate correction of each node at communication cycle m is strictly negative ($C_1 \Rightarrow \forall i \in N_{ok}\ vRateCorrection_i(m) < 0$). If this overrun is larger than the damping factor, then each node will reduce its local rate correction, thus increasing its logical clock's period. Otherwise, the rate correction will be set to zero (see Figure 4.2). In both cases we have $\forall i \in N_{ok}\ vRateCorrection_i(m + 1) > vRateCorrection_i(m)$ and thus the relative period deviation of the logical clock (global time) is increased until scenario C_1 is left (the logical clock has moved back within the CDI).

Notice that condition C_1 is more general than $C_{1a} : \forall i, j \in N_{ok}\ \delta_{L_i}(m) < \delta_j(m)$ (the logical clock period deviation of each node is smaller than every underlying oscillator period deviation). We have $C_{1a} \subset C_1$. We are interested in C_{1a}, and since the argumentation is valid for C_1 it will hold for C_{1a} too.

Scenario 2: $C_2 : \forall i \in N_{ok}\ \delta_{L_i}(m) > \delta_i(m)$. For cycle m, the logical clock deviation of each node is greater than its underlying oscillator deviation (and thus the global time deviation is outside of the CDI). Using the previous argumentation the logical clocks will move back toward the CDI until C_2 is not valid anymore.

Scenario 3: $C_3 : \exists i, j \in N_{ok}$ *such that* $\delta_{L_i}(m) \geqslant \delta_i(m)$ *and* $\delta_{L_j}(m) \leqslant \delta_j(m)$. For cycle m, there exists at least one logical clock deviation that is faster (or equal) and one logical clock deviation that is slower (or equal) than its underlying oscillator deviation ($C_3 = \overline{C_1 \cup C_2}$; the global time deviation is within the CDI). Consequently, their respective rate correction term will have a different sign ($C_3 \Rightarrow \exists i, j \in N_{ok}$ $vRateCorrection_i(m) \leqslant 0 \leqslant vRateCorrection_j(m)$). Similarly to the state correction, the rate correction is updated as the average of a subset of logical clocks' rate differences. Consequently, it is mathematically impossible that the next rate correction terms for every node are all strictly positive or strictly negative (see Section 4.3.1). Moreover, the damping factor can not modify the rate correction computation, such that Scenario 3 is left, since this mechanism always tries to move locally the rate correction term toward zero (see Figure 4.2). Scenario 3 is then a stable state (as long as the CDI is not changing faster than the dynamics of the synchronization algorithm) and we have $\exists i, j \in N_{ok}\ vRateCorrection_i(m+1) \leqslant 0 \leqslant vRateCorrection_j(m+1)$.

4.3.3 Experimental Validation – Static behavior

The aim of this Section is to highlight the behavior of clock synchronization algorithms and validate the formal analysis presented in Sections 4.3.1 and 4.3.2. Two approaches have been selected: First, the SIDERA tool [Han06] has been used to simulate the behavior of the TTP/C protocol. Second, Decomsys' NODE<ARM>[1] hardware equipped with FlexIM[2] boards has been used to implement a FlexRay cluster and experimentally study its behavior (see Section 5 for further information on the experimental setup).

For both environments, the system under test consists of four nodes, whose oscillators can be controlled. This function is available in the user interface of the SIDERA simulator, while dedicated boards with digital to analog converters and voltage controlled oscillators have been developed for the FlexRay nodes. The actual cycle length

[1]http://www.decomsys.com/flyer/NODE_ARM.pdf
[2]http://www.decomsys.com/flyer/FLEXIM.pdf

for each node has been measured by our tester node as the distance between two consecutive frames with the same identifier. The cycle (respectively round) length has been configured to 5 ms for the FlexRay cluster and 0,8 ms for the TTP/C cluster. In the following tables, $N1...N4$ stands for the relative oscillator period deviation of node 1 to node 4, the cycle length (cl) is given in ns and GT stands for the resulting global time deviation. Each drift is given in 10^{-6}.

For the first campaign, we define a set of oscillator deviations $\{-400 \cdot 10^{-6}, -300 \cdot 10^{-6}, -200 \cdot 10^{-6}, -100 \cdot 10^{-6}\}$ that we assign to every node in a cyclic manner during the four test cases. The cold starter node (i.e. the one which performs the start-up and thus initializes the global time) is fixed and set to node 1 for this experiment. Since every oscillator deviation is sequentially assigned to node 1, the initialization of the global time is set to four different values. Table 4.2 summarizes the drift of the global time with respect to the cold start node (for this experiment always node 1). As expected, for the TTP/C protocol the cycle length remains nearly constant for the chosen oscillator deviation set (between 800.170ns and 800.190ns – the minimal differences that have been observed are due to the clock reading errors and are within the precision of the system). Concerning the FlexRay protocol, we can clearly identify two groups. The results of the first two experiments are nearly identical (cycle length of 5.001.050ns and 5.001.100ns – within the precision window of 200ns) but considerably differ from the results of the last two experiments (5.001.400ns and 5.001.420ns). We can thus confirm that, for FlexRay systems, not only the set of oscillator deviation values matters but also the previous correction value – the global time thus depends on the system history.

Exp.	**N1**	N2	N3	N4	cl	GT
TTP/C – Fault Tolerant Average						
1	-100	-400	-300	-200	800.180	-225
2	-200	-100	-400	-300	800.190	-235
3	-300	-200	-100	-400	800.170	-210
4	-400	-300	-200	-100	800.170	-210
FlexRay – Fault Tolerant Midpoint						
1	-100	-400	-300	-200	5.001.100	-220
2	-200	-100	-400	-300	5.001.050	-210
3	-300	-200	-100	-400	5.001.400	-280
4	-400	-300	-200	-100	5.001.420	-285

Table 4.2: Static convergence point depends on cold starter

The aim of this second campaign is to check if the deviation of the global time stays within the *clock deviation interval* (CDI). For that purpose, we have used sixteen different oscillator deviation sets (four groups that we have permutated between the nodes) and measured the resulting cycle lengths. This experiment validates our claim that the global time is bounded by the clock deviation interval in a stable environment.

Notice that for a very narrow CDI, the global time deviation was slightly slower than the slowest oscillator. This interval, however, represents a few nanoseconds and is much smaller than the relative precision χ. Consequently we can claim that the global time is within the clock deviation interval as defined in Section 4.1.2.

Exp.	N1	N2	N3	N4	GT Min	GT Max	GT Variation
TTP/C – Fault Tolerant Average							
1 – 4	-400	-300	-200	-100	-240	-210	30
5 – 8	400	300	200	100	260	300	40
9 – 12	400	400	400	350	398	399	1
13 – 16	-400	-400	-400	-350	-402	-401	1
FlexRay – Fault Tolerant Midpoint							
1 – 4	-400	-300	-200	-100	-285	-210	75
5 – 8	400	300	200	100	200	280	80
9 – 12	400	400	400	350	395	396	1
13 – 16	-400	-400	-400	-350	-403	-400	3

Table 4.3: Deviation of global time within the clock deviation interval

These two campaigns have highlighted three important properties of convergence-averaging based clock synchronization algorithms. First, the global time for FlexRay does not only depend on the oscillator drifts but also on the history (here, which node has performed the start-up). This dependency is not true for the TTP/C protocol, and the global time only depends on the oscillator drift set within the system. Second, the accuracy of the global time is bounded by the clock deviation interval. Third, the global time deviation stays approximately constant for a constant oscillator drift.

4.3.4 Experimental Validation – Dynamic behavior

The aim of this campaign is to illustrate the behavior of the global time with respect to a changing CDI. In particular, we focus on uncorrelated and ”slow” drifts (slower than the dynamics of the clock synchronization algorithm), and on node crashes. Figure 4.10 pictures the experiment for a FlexRay system: Both the oscillator- and the logical time deviations of the four nodes are illustrated. During a first part of the experiment (until communication cycle 300), the CDI stays constant. Then, between communication cycles 300 and 750, the CDI is slowly moved. Finally, a node crash is simulated for node 3 and node 4 at cycle 780 while the deviations of node 1 and node 2 stay constant. As a result, the CDI is suddenly narrowed.

Two important behaviors can be observed here. First, the global time deviation “follows” the CDI, even when the latter is slowly drifting. This is an important behavior, since it illustrates property P1a: In a fault-free system, if the global time deviation is initially within the CDI, it will stay within this interval. Second, when the CDI is abruptly modified (e.g. due to node crash, communication cycle 780), the global

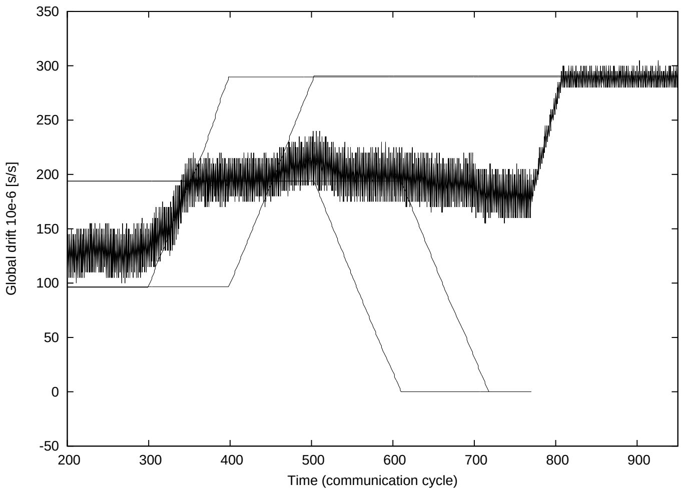


Figure 4.10: Global time behavior with respect to dynamic oscillator deviation for FlexRay systems

time deviation returns back toward the new CDI with a speed defined by the size of the damping factor (which determines in fact the dynamics of the FlexRay clock synchronization). This represents Scenario 1 of Section 4.3.2. The observed behavior illustrates property P1b: Even if the global time deviation is outside of the CDI, it will converge back to this interval. Notice that during a small amount of time (communication cycles 780 to 810) the global time deviation is outside the CDI. This state is however bounded in time and only consecutive to an abrupt modification of the CDI.

Figure 4.11 illustrates a similar CDI behavior for a TTP/C system. Two main differences can be noticed: First, the global time deviation is always approximately the fault tolerant average of the oscillator deviations within the system. Second, the global time deviation returns instantaneously (within the next cycle) to the CDI. These two behaviors are due to a correction algorithm solely based on clock measurements (there is no rate correction using the system history).

These two experiments illustrate the principles of our test approach. In a first time, a tester node (emulating several nodes) enlarges the CDI and shifts the global time. At the end of the experiment, the tester node stops sending messages (fail-silent behavior). This fact can be detected using the fast ramp followed by the global time while it returns to the new CDI.

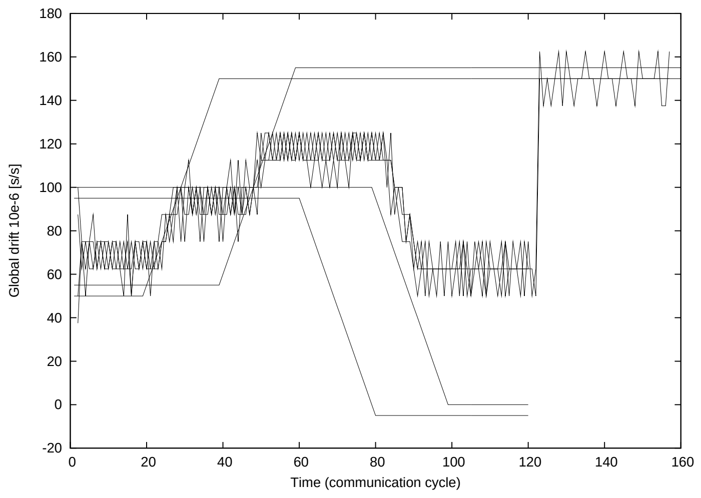


Figure 4.11: Global time behavior with respect to dynamic oscillator deviation for TTP/C systems

4.4 Remote Evaluation of the QoS

We present now two methods for the remote Quality of Service evaluation with respect to clock synchronization. First, we propose a method to *remotely assess the nodes' offset correction value* both for convergence functions implementing offset correction (e.g. TTP/C) and offset plus rate correction (e.g. FlexRay). Second, a remote method for *precision approximation* is presented. The assessment of the nodes' offset correction term and of the system precision are totally transparent for the system under test.

4.4.1 Remote Offset Correction Monitoring

This method requires the oscillator deviation to be stable (approximately constant) during consecutive re-synchronization periods, which can be usually assumed with our quartz model (see Section 4.1.1). In the general case, a tester node with a time base synchronized to the network is required. The transmission times of the different nodes during a communication cycle are recorded (remote clock estimation process, see Section 4.1.3). In a fault-free case, each node will correct its own time base such that it converges approximately to the same value (agreement property of the clock synchronization, see Section 4.1.3). Therefore, the local nodes' offset correction values can be computed as the difference between the remote clock estimation and the tester's offset correction (in the case of a perfectly synchronized tester with a correction of 0, the node's offset correction represents exactly the remote clock estimation). The remote clock estimation delivers an information about the current state of the node's time base while the next tester's offset correction provides an (agreed) result after re-synchronization (see Figure 4.12). The precision of this measurement is bounded by the system precision. This method can be applied both for convergence functions using offset only or offset plus rate correction.

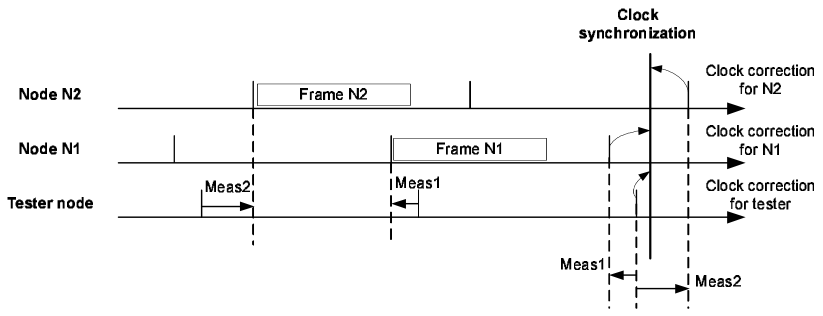


Figure 4.12: Remote nodes' offset correction measurement

In case of FlexRay, the fact that the offset is corrected only every two communication cycles (during the odd cycle) can be used to lower the requirements on the tester. Hence, a direct comparison between two successive cycle lengths enables the assessment of the offset correction (see Figure 4.13). Note that we do not know precisely the cycle start of each node. Instead of this, we measure the cycle length for a node k as the distance between two successive frames with the same identifier sent by this node. The even cycle length is the effect of the node's oscillator deviation plus rate correction while the previous cycle length had the same rate correction and same oscillator deviation with the additional offset correction. Consequently, a direct comparison between two successive cycles suffices to get a node's local offset correction. With this method, the tester does not require anymore to synchronize to the network or to perform clock correction. A simple time counter triggered by each frame reception suffices.

This measurement assumes physical clock deviation and rate correction changes between consecutive cycles to be negligible. With respect to the clock deviation, we refer to the stability condition in equation 4.3. The rate correction remains the same for an even and the subsequent odd cycle, and would therefore not require consideration in the ideal case. In practice, however, the cycle length is measured from a frame to another and not from one re-synchronization point to the other. Consequently, during the measurement of the odd cycle, the new, potentially modified rate correction value is already applied. Therefore we must require the rate correction changes to be negligible as well. In a standard system, the rate is varying very slowly (slower than one clock tick per cycle, see Section 4.1.1) anyway. In our specific case with deterministic replay, we do change the rate but have full control over it. It would therefore be possible to account for the error thus introduced, but experiments have shown that it can still be

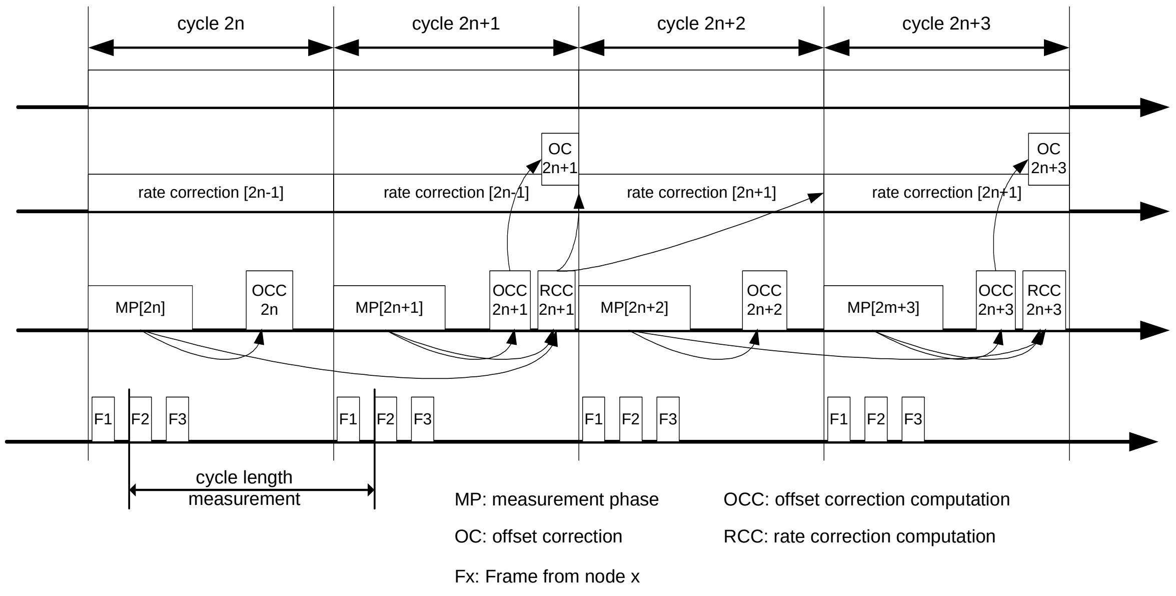


Figure 4.13: Offset and rate correction computation measurement for FlexRay [Fle05]

neglected.

4.4.2 Remote Precision Approximation

The *precision* of a distributed system is defined as the maximal distance between any two logical clocks (see Section 4.1.3). This metric is difficult to measure, since the logical clocks are not visible outside of the nodes (and therefore not globally visible at the network). What we need here is (a) a snapshot of each node time base in isolation and (b) a common reference to compare the time bases and actually compute the precision.

The a-priori-known schedule from time-triggered communication system can be used: the transmission time from node k represents node k's schedule interpretation according to its local time base. Moreover, the periodic transmission delivers periodic and successive snapshots of each node's time base. The next problem is the serial behavior of the communication. Hence, transmission is multiplexed in time to avoid collision on the bus. The logical clocks snapshots are then not simultaneous and can not be directly used for precision measurement. We have two solutions to solve this problem:

- a tester node implementing clock synchronization can be used as reference. Then the precision is computed as the largest interval between any two time measurements within one communication cycle (see Figure 4.14, precision A). This approach presents high requirements on the tester hardware since a clock synchronization algorithm has to be available as well as a processing engine to compute online the precision and/or to forward the information.

- the instantaneous rate of the nodes' logical clocks can be used to interpolate the measured snapshot to a common point in time and perform the precision approximation (see Figure 4.14, precision B). This naturally requires the capability to remotely measure the rate of each node's logical clock. In the case of TTP/C, the averaged offset correction terms during a given time window provide information about the average node's clock rate in comparison with the other nodes within the system. Notice that the logical clock and the physical clock are running at the same rate when the correction term has been processed. We require a reference node (whose nominal frequency is known) in this case and this method works only under the assumption that the oscillator deviation stays approximately constant during the measurement phase. In case of FlexRay, the double-cycle re-synchronization scheme can be advantageously used here again to perform this measurement (see previous section). Note that due to rate correction the rate of the physical clock and of the logical clock of a given node usually differ.

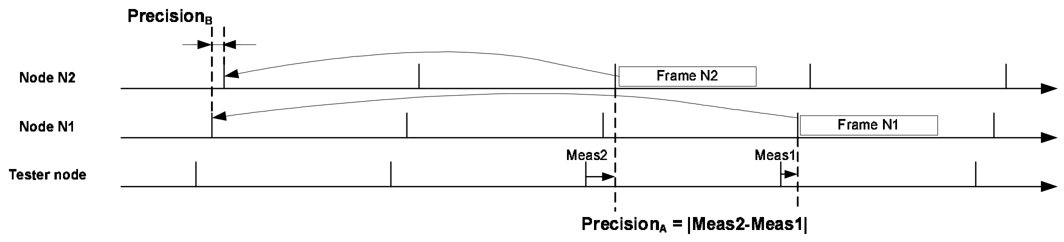


Figure 4.14: Remote precision measurement

4.5 Chapter Summary

4.5.1 Illustrative Fault Injection Experiment

Figure 4.15 presents an exemplary fault injection experiment for a FlexRay network. This test case regroups the findings made in this chapter and illustrates how they can be combined to perform a transparent online test of the error detection mechanisms. The "tester's cycle length" graph (part A) describes the tester operation. During the

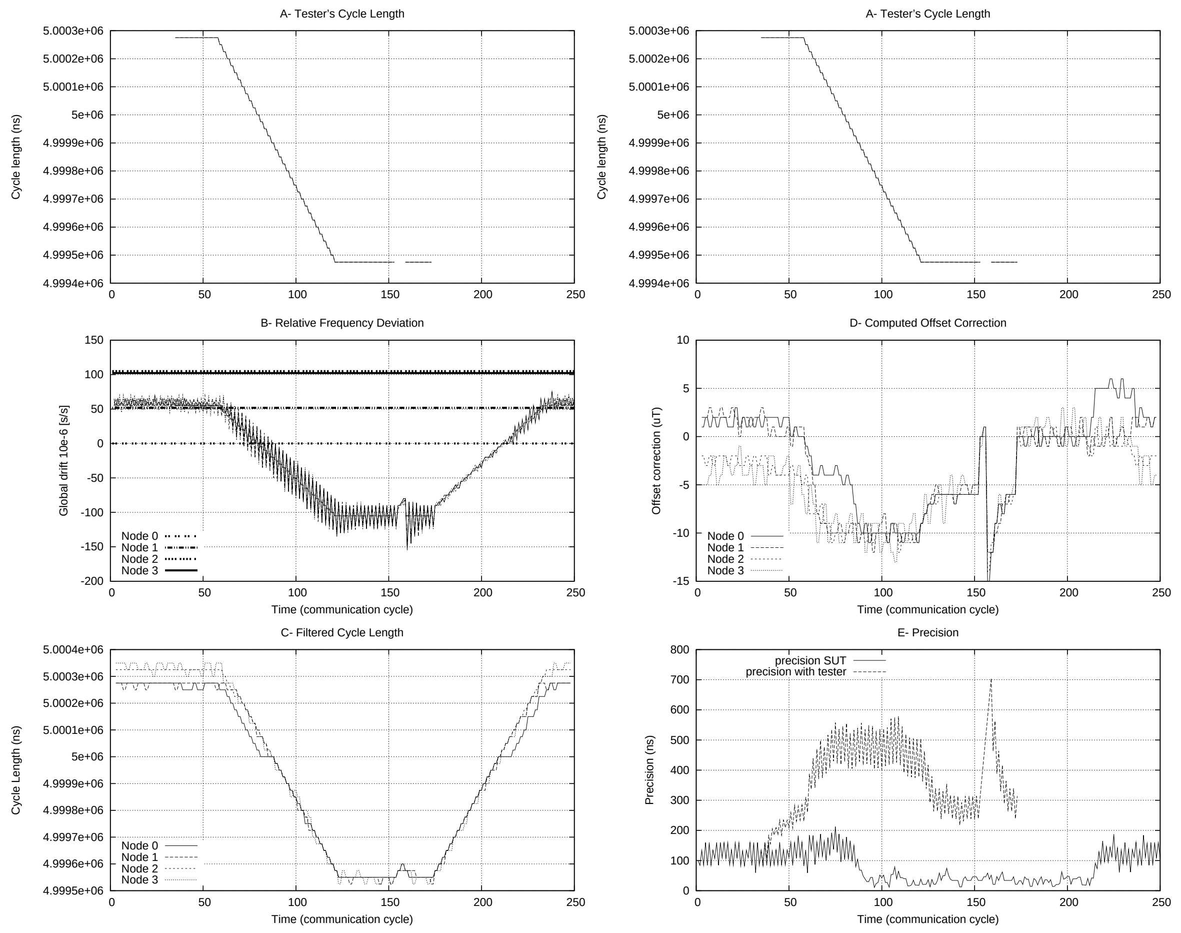


Figure 4.15: Remote test of the fault-detection mechanisms

first 30 cycles the tester listens to the application and initializes its cycle length to the current system value (start-up operation). After that, it performs a replay operation with the constant deviation of $50e^{-6}$ (or 250ns more than the $5000\mu s$ configured, which corresponds to the current global time deviation of the system) during further 25 cycles. Then, our tester replays a ramp in order to drive the global time outside of the former CDI, which is, in our case, located between 0 and $100e^{-6}$ (see Figure 4.15-B). Finally, the tester performs a replay with a constant deviation of $-100e^{-6}$ and corrupts its frames between cycles 151 and 155 (fault injection) before stopping operation at cycle 175. Note that a second tester has validated the presence of corrupted frames during fault injection.

The graph B ("relative frequency deviation") plots the oscillator deviation (CDI) as well as the global time deviation (response of the system). The graphs C, D and E present different measurements performed online by our tester. Figure 4.15-C presents the rate of the nodes' logical clock (without the offset correction term). This value illustrates the nodes physical clock plus their rate correction. The nodes' local offset correction term, computed as described in Section 4.4.1, is illustrated in Figure 4.15-D. Finally, Figure 4.15-E presents the precision approximation as described in Section 4.4.2. Different important behaviors can be observed here:

The precision of the system under test (without considering the tester) is always under 200ns during the whole experiment. The total precision (when taking the tester into account) is bounded, as well, and does not depend on the current tester's deviation. This result is consistent with the analysis presented in Section 4.2 and more especially in Table 4.1.

The tester sends corrupted frames between the cycles 151 and 155. The correct rejection of these frames by the different nodes is particularly visible with the offset correction (graph D), and also with the precision graph. Notice that during this interval, the filtered cycle length of the four nodes moves back toward the CDI as analyzed in Section 4.3. The clock state deviation between the nodes and the tester increases (since this last does not correct its clock) until the tester sends correct frames again (cycle 156), which represents an extremum for the precision graph (and for the nodes' offset correction). Finally, the nodes' logical clocks return to within the original CDI after the tester has stopped transmission.

At the beginning of the ramp (cycles 60 to 80), the filtered cycle length of node 0 does not exactly coincide with the three other nodes. This corresponds to the interval where the tester is faster than the physical clocks of node 1 to 3 but slower than the physical clock of node 0 (see graph B). After cycle 80 and until cycle 120 the tester is faster than all four physical clocks and the filtered cycle length of all four nodes perfectly coincides. This behavior is due to the *damping factor* which slightly modifies the rate correction [Fle05] toward the own physical clock deviation, and can be used to remotely measure the oscillator drifts within a distributed system (see [AS07] for a detailed analysis). This effect can be further used by our tester to detect whether the global time is still within the CDI or not, and thus optimize the test operation.

4.5.2 Summary

We have provided background knowledge required for this study. We have identified two attributes regarding the frequency deviation of quartz oscillators: *bounded deviation* and *stability.* Moreover, we have identified time-triggered communication protocols to implement convergence-average clock correction functions and we have defined terms such as global time, logical clock (deviation) and clock deviation interval (CDI).

Further, we presented three important analyses for our transparent test approach in this chapter. The first one concerns the controllability: We have shown that it is indeed possible to perform deterministic replay operation and to take control of the clock synchronization of an operating network without posing a threat to the system. We have seen that the resulting precision is bounded for a carefully chosen replay operation. With this method, we are then in measure to modify the clock deviation interval during the test operation and to safely drive the global time outside of its former CDI.

The second analysis has shown that the global time always converges to within the accuracy of its underlying physical clocks (CDI) independently of the system history. This behavior provides an indication on the current “forces” that are driving the logical clocks. More especially, when replay operation is driving the global time out of the clock deviation interval, the reception of corrupted tester frames can be remotely identified. Hence, the nodes’ logical clocks will start to move back to the CDI, thus providing information about the correct processing of the tester frames.

During the last part of this chapter we discussed different remote methods for the evaluation of the clock synchronization mechanism. We have seen how the local nodes’ offset correction as well as the system precision can be assessed. These two measurements can be further used independently from our test approach to evaluate the quality of the service provided.

Chapter 5

Experimental Setup

Always listen to experts. They'll tell you what can't be done and why. Then do it.

Robert Heinlein (1907 – 1988)

The goal of this chapter is to describe the experimental setup used to perform the different evaluations presented during this work. This setup principally consists of a FlexRay cluster of four standard nodes running a dedicated test application, and an additional tester node with the capability to monitor the network and perform replay operation. Further, we present a study on the FlexRay clock synchronization, in order to provide additional insights and explore the limits of this distributed mechanism.

5.1 System Under Test

5.1.1 Hardware

Our test application consists of four Decomsys FLEXIM boards[1] each one hosted by a NODE<ARM>[2]. The FLEXIM board is an Industry Pack module and offers a FlexRay controller as well as two FlexRay physical layers. The NODE<ARM>, on the other side, is a stand-alone automotive prototyping platform providing an ARM CPU, a FPGA, and two Industry Pack interfaces to connect up to two Industry Pack modules. The FLEXIM boards have been modified such that the onboard oscillator has been disconnected. Instead, an additional board providing a voltage controlled

[1]http://www.decomsys.com/flyer/FLEXIM.pdf
[2]http://www.decomsys.com/flyer/NODE_ARM.pdf

oscillator generates the required pulses. The quartz frequency is controlled by the application using the analog output of the node's platform. This setup allows the calibration of the nodes' oscillators to minimize the initial frequency difference as well as the emulation of different oscillator deviation scenarios during the test campaigns.

5.1.2 Software

FlexRay configuration

The test application is based on the Decomsys' skDemo3 application (additional information available in [Dec06]). This configuration defines a cycle length of $5000\mu s$ with a static segment containing 52 static slots of $87\mu s$ each and a payload length for the static frames of 64 bytes (32 words). Further, a dynamic segment containing 58 minislots of $6\mu s$ are defined as well as a NIT segment of 125 μs. The slots 15 to 18 are assigned for transmission to the nodes 0 to 3, respectively.

In case an error in the clock correction mechanism has been detected, the node turns into a silent mode (either *normal passive* or *halt* mode [Fle05]). Depending on the configuration, the communication controller hardware might be able to re-synchronize automatically to the bus traffic, or an action from the application might be explicitly required. In our case, neither the hardware nor the software can restart a node after it has failed. For a normal application, this would decrease the system robustness; for our experiments, however, it permits to detect more precisely when a node has first failed.

Concerning the application, each node implements two tasks that are synchronized with the FlexRay communication cycle. The system task, on one side, is started $4800\mu s$ after the cycle start (i.e. 200 μs before the end of a cycle) and aims at re-synchronizing the operating system to the FlexRay global time and testing the current controller status. The application task, on the other side, is started $200\mu s$ after the cycle start and delivers the services for our test application.

The test application

Our dedicated test task delivers different services common to the four nodes. First, it reads the internal offset and rate correction values as well as the current quartz configuration and makes this information globally available (the information is packed into a frame and transmitted on the FlexRay network). The information, which is concatenated with the node status as well as the reception statistics, is printed out via a serial interface. A terminal can be connected to each single node to see its diagnosis information. Finally, each node implements a state machine to control its own oscillator drift in accordance with the other nodes. The rough synchronization of the four state machines is performed using message exchange; a master node (usually node 0) triggers the start of operation and further transitions at cycle level. The fine synchronization between the nodes relies on the FlexRay global time (precision of few hundreds of nanoseconds), and its local synchronization to the operating system (*simultaneity* property of the time-triggered architecture, see Section 2.3.2). Notice that these three services are very useful for the investigations performed in this work. They are, however, definitely not required for the application of our approach in practice.

5.1.3 System Initialization

Nodes' oscillators calibration

We have seen in Section 4.1.1 that oscillators are deviating from their nominal frequency. We describe here the calibration method we have applied to minimize the initial drift between the standard nodes. This operation was required to minimize the side effects during the experiments.

Our calibration consists of two steps. First, the quartz of node 0 is manually tuned to obtain the desired frequency. After that, we developed a master-slave application to automatically calibrate the oscillator of the three other nodes to the first one. This application basically tunes the oscillators of these nodes to minimize the offset and rate correction difference with the reference node (remember that this information is globally available with our test application). This new kind of synchronization algorithm makes the oscillator configuration converge, such that the resulting frequencies are close to the reference one. The output is available at the serial interface and can be used to update the corresponding header files. Note that this file further defines for each single quartz the configuration corresponding to the deviations from -500e^{-6} to 500e^{-6} with a granularity of 50e^{-6}. These definitions are relative and only a zero reference needs to be corrected after a calibration.

The tester node does not provide a voltage-controlled oscillator, but its initial cycle length can be adjusted to hit the nominal cycle length of the monitoring node as close as possible. The granularity for the calibration of the replay node is 25ns while the granularity for the standard node calibration is better than 10ns. The attainable accuracy of the calibration is presented in the following.

Experimental assessment of the quartz accuracy within the prototype

We have furthermore measured the frequency stability of our system. Figure 5.1 illustrates the oscillator drift rate for the replay node (with identifier 14) during 18,000 communication cycles (90 seconds) and for the standard nodes (identifiers 15 to 18) during 60,000 communication cycles (300 seconds). The initial rate offset is located between -8 and 8ns per communication cycle. It corresponds to a drift below 2e^{-6}s/s. The drift rate variation is below 2e^{-11}s/s^2 for the standard nodes and below 1e^{-12}s/s^2

for the replay node.

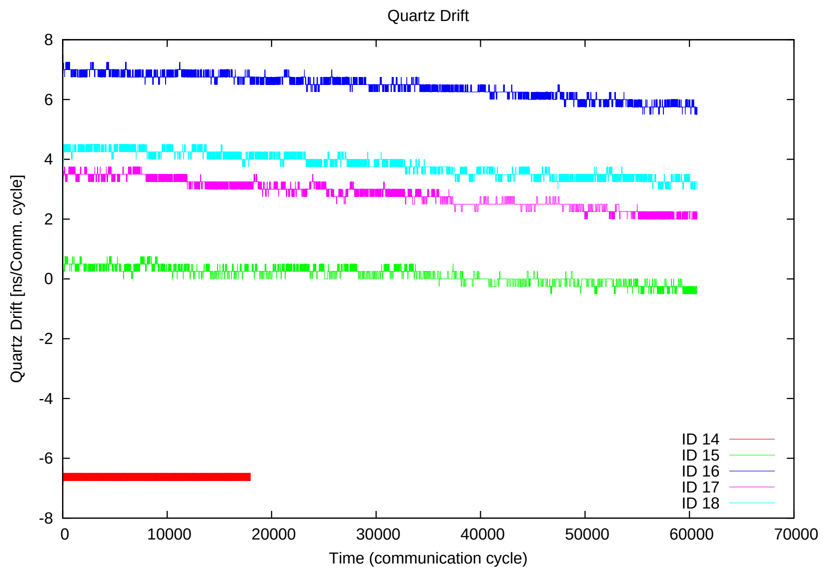


Figure 5.1: Quartz drift measurement

During a second experiment, we have periodically measured the drift rate variation of the different nodes for a duration of two hours. Table 5.1 shows the results (maximal and minimal quartz drift rate) for the different nodes. We can observe, as expected, that the short term rate variation is very low (below than $6e^{-11}$s/s^2) and that for an experiment time of a few seconds this effect can be neglected in comparison to the stimulus applied. Additionally, the quartz stability increases with the time after power-up. Furthermore, the replay node presents a better quartz stability than the standard nodes. Recall that the standard nodes implement an oscillator control mechanism that introduces some noise. This oscillator characterization illustrates that a calibration before starting the experiments is required and that further drifts are negligible. These results are coherent with the oscillator model presented in Section 4.1.1.

Node	minimal drift [s/s^2]	maximal drift [s/s^2]
Replay node	$< 1e^{-12}$	$< 1e^{-12}$
Standard Nodes	$1e^{-11}$	$6e^{-11}$

Table 5.1: Long term quartz drift

5.2 Tester Node Architecture

The core of our test concept relies on a dedicated tester node with the ability to monitor the FlexRay bus traffic and perform replay operation. We present here the internal architecture as well as the support required by a host PC to control the test execution and interpret the data gathered.

5.2.1 Overview

The hardware platform for our tester node is the Decomsys' NODE<MPC5200>[3]. It consists of a Motorola PowerPC MPC5200[4] connected to a 32MB SDRAM block and to an Altera's Cyclone II FPGA (EP2C70[5]) providing different interfaces (FlexRay, CAN, LIN, Ethernet, USB, Analog and Digital inputs / outputs). The FPGA design comprises a dedicated measurement controller [ASH$^+$04] as well as a SPEAR processor [Fle07] for accessing and processing online the data coming from a FlexRay network. The MPC5200 is running RedBoot[6] and the interface to a host computer is performed using a telnet connection. This interface affords simple instructions such as memory read / write operations. Note that the FPGA interface is mapped in memory and therefore its registers are available for direct host access. At the host side, a Perl[7] script is used to configure the hardware, control the operation and post-process the data. Figure 5.2 illustrates our tester's architecture.

5.2.2 The Dedicated FPGA Design

Architecture Overview

As pictured in Figure 5.3, our tester node basically consists of different *monitoring units* delivering data from the different sources (e.g. CAN, LIN, digital or analog input). These data are ordered in the *data interface (monitoring)* and further sent to the host through the *wrapper*. An opposite architecture (*data interface (replay)*) is provided for the transmission of data. The interface to the FlexRay bus uses a *FlexRay protocol engine*. More information concerning our prototype is available in [ASH$^+$04].

The tester node enhancement is achieved by the integration of the SPEAR microcontroller for the processing of complex stimuli. The motivation for this IP is first to be platform independent (in contrary to Altera's NIOS[8] or Xilinx's Microblaze[9]).

[3]http://www.decomsys.com/flyer/Datasheet_NODE_MPC5200.pdf
[4]http://www.freescale.com/files/32bit/doc/data_sheet/MPC5200BDS.pdf
[5]http://www.altera.com/literature/hb/cyc2/cyc2_cii5v1.pdf
[6]http://www.sourceware.org/redboot/
[7]http://www.perl.org/
[8]http://www.altera.com/
[9]http://www.xilinx.com

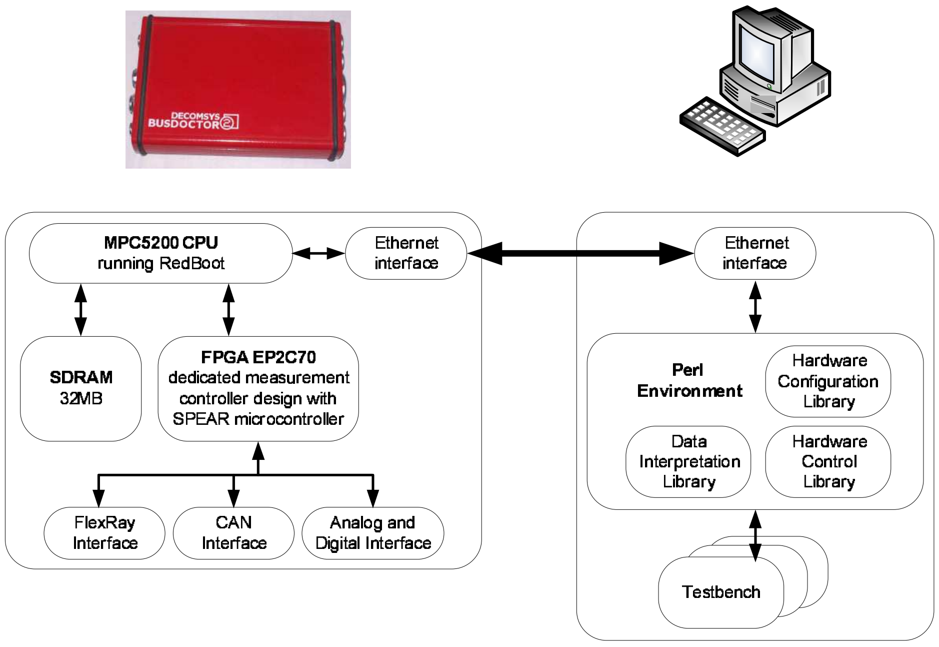

DECOMSYS BUSDOCTOR
MPC5200 CPU running RedBoot
Ethernet interface
SDRAM 32MB
FPGA EP2C70 dedicated measurement controller design with SPEAR microcontroller
FlexRay Interface
CAN Interface
Analog and Digital Interface
Ethernet interface
Perl Environment
Hardware Configuration Library
Data Interpretation Library
Hardware Control Library
Testbench

Figure 5.2: Overview of the tester's architecture

Moreover, we need real-time properties to guarantee the computation of the next stimulus iteration before the next communication cycle starts. This is provided by using an instruction set of constant execution time. Finally, we are in a prototyping phase and require a flexible, extendable computing unit. One main concept of SPEAR is the support of *Hardware Extension Modules* [HD02], which provides an environment for easy addition of new modules to this core. The integration is supported in hardware by well defined interfaces and in software by simple module access using memory mapping.

The SPEAR core requires four dedicated connections to our bus analyzer (in gray in Figure 5.3). The first one (*programming connector*) goes from the CPU to the parallel programmer module so as to load the program to execute. The second one, the *replay connector*, transfers the generated packets together with a 25ns accurate timestamp to the FlexRay transmitter. This timestamp precisely defines the transmission time of the frame for deterministic replay. The third dedicated interface, the *receiver connector*, is concerned with gathering the time differences between the tester node's time base and the received frames. This input is required to further compute the stimulus for the next communication cycle. Finally, the *monitoring connector* is used to send SPEAR status information to the user.

In order to minimize the programming time of the SPEAR core, the programming connector is implemented as a dual clock FIFO. The replay connector uses the asynchronous concept developed in [FA07] with a 16 bits to 16 bits interface. Since the

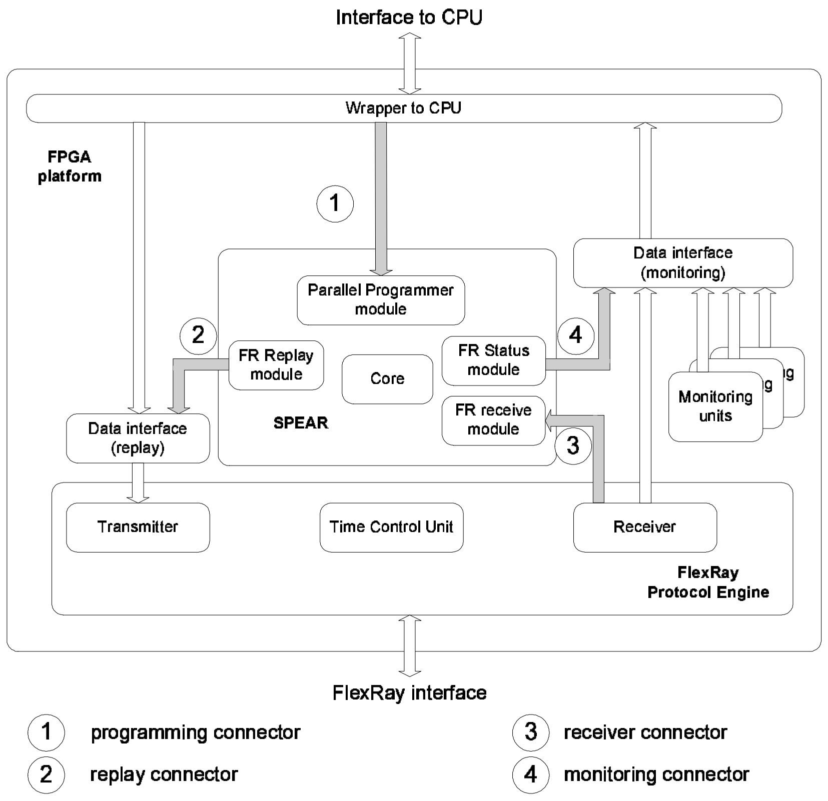


Figure 5.3: Tester's FPGA architecture

packets to transmit are larger than 16 bits (between 128 bits and 2176 bits), several successive accesses are required. The receiver and monitoring connectors are implemented as 32 bits to 16 bits (respectively 16 bits to 16 bits) asynchronous connectors. The main advantages of our asynchronous, delay-insensitive connector in comparison to standard synchronous serial or parallel is the data transfer efficiency pro interconnect wire as well as the flexibility during design and robustness during operation. More information are available in [FA07].

The main advantages of having the SPEAR microcontroller implemented within the measurement controller are (i) the high interaction between these two modules and (ii) the fast response time. Hence, each internal status data can be directly mapped to SPEAR's extension units to provide more accurate information (both in the time and value domain). The firmware running on the SPEAR microcontroller has moreover a better control to the replay operation. The minimization of data exchange latencies is important to maximize the CPU time available for processing and avoid missing deadlines.

The SPEAR Extension

Our SPEAR implementation provides a 4kB internal data memory as well as a 8kB program memory for loading and executing programs. The *parallel programmer module* replaces the original UART module and presents the same interface to the SPEAR core while using a different programming environment. The *FlexRay (FR) replay module* prepares and send the data for deterministic replay operation to the measurement controller. The *FR receive module* is in charge of buffering the frames information received on the FlexRay bus and presenting a compacted form to the SPEAR. Finally, the *FR status module* stores the (diagnosis) information the SPEAR wants to send to the host and generates a packet to the measurement controller when triggered.

The interface from the parallel programmer module is the same as the one of the miniUART [Sei07], in order to ensure compatibility and save development time. The three other modules have been regrouped into a *bdConnector* module, whose programming interface is described in Table 5.2. The *status* register delivers information to the SPEAR core about the completion of data transmission (SPEAR status and replay data) as well as data availability from the FlexRay protocol engine. This last flag is set once per cycle and is used to trigger a new computation step. The *configuration* register provides control about monitoring operation and informs the measurement controller that a new data packet from SPEAR is available. The *type* and *version* registers are providing version and type number for tracking. Finally, the *FlexRay receive Data*, *Replay Data* and *SPEAR Status Data* registers are building the actual data interface.

Offset [bytes]	**Register name**	**Operation**
BASE + 0	Status	read
BASE + 2	Configuration	write
BASE + 4	Type	read
BASE + 6	Version	read
BASE + 8	FlexRay receive Data	read
BASE + 10	Replay Data	write
BASE + 12	SPEAR Status Data	write

Table 5.2: bdConnector register interface

5.2.3 Firmware

In this section we focus on the dedicated firmware running on the SPEAR microcontroller. Concerning the MPC5200 CPU, a standard RedBoot is managing the interface to the host computer and thus will not be considered any further. The firmware running on the SPEAR is executed once per cycle and triggered by the *TCU_available* flag that signals the end of a static segment and thus the availability of new frames to process. Note that the computation has to be finished before the end of the current cycle in order to timely generate the frames to transmit for the next cycle.

The firmware basically consists of a state machine to control the system operation, dedicated drivers to control the bdConnector extension, and a diagnosis function to compute the current nodes' offset correction and logical clock rate as well as the system precision. The state machine provides the following basic states:

- *Initialization*: system initialization
- *Measurement*: the software is listening silently to the bus traffic and averages the current cycle length over time
- *InitCycleLength*: the tester's cycle length is initialized (according to the measurement phase) and replay operation is started. The logical clock deviation between the tester and the standard nodes is approximately zero for stable quartz (see section 4.1.1)
- *ReplayStatic*: the tester operates a deterministic replay without changing the cycle length (its logical clock deviation is constant)
- *ReplayRamp*: the tester operates a deterministic replay and linearly changes its cycle length. The slew rate of the ramp can be configured
- *End*: current operation (replay, monitoring) are stopped

The development of a state machine for a given experiment is made very flexible. Hence, the designer is free to define both the sequences of states to perform as well as the transition conditions. Moreover, both absolute transition conditions (e.g., 50 cycles after start) and relative transition condition (when the nodes' offset correction is negative) can be used, thus increasing further the test flexibility.

The firmware provides additionally drivers to control the *bdConnector* module. More especially, four functions are available:

- *SendMonFrame*: to send diagnosis data to the host (using the FR status module)
- *SendReplayFrame*: to send a frame to replay including timestamp (using the FR replay module)
- *ReadTCUData*: to read the frame information monitored and store it into a pre-defined structure (FR receive module)
- *HCRC*: to compute a FlexRay header CRC for frame generation

Parallel to the replay operation, our tester is able to monitor the FlexRay bus traffic (including its own frames). Through the *SendMonFrame* function, the firmware can insert diagnosis information for the end user within the data flow. The messages contain the following fields:

- *absolute cycle counter*: counter incremented for each cycle since the beginning of the experiment.
- *current state*: current state of the state machine.
- *current cycle length*: current cycle length applied for deterministic replay. This information is only relevant in the ReplayStatic and ReplayRamp states.
- *offset correction*: results of the offset correction for each node (as described in Section 4.4.1)
- *filtered cycle length*: nodes' cycle length after their respective offset correction has been removed. This information represents the current speed of the node's logical clock (speed of the physical clock corrected by the rate correction term)
- *virtual quartz deviation*: quartz deviation a standard node should have to present the same logical clock deviation as the tester

Additionally, some fields are left blank for debugging purposes or for further developments. Note that for most of the experiment presented, a second (silent) tester was used to test the correct operation of the first tester.

5.2.4 Host PC

Different important tasks are also required from the host PC to support test execution. Hence, the host is in charge of configuring the hardware prior the test campaign, of controlling the data exchange during the experiment and finally of processing the data received for interpretation.

Hardware configuration

The configuration part, before the experiment can be started, consists of three main points. After a power-up of the platform, the MPC5200 CPU executes the boot loader and stays in a default state where FPGA accesses are not possible. The first step is then to configure the interface such that FPGA registers can be mapped in memory.

The second configuration point concerns the FlexRay protocol engine and the FPGA in general. Hence, the embedded FlexRay core is expected to synchronize to the running cluster and thus requires to be accordingly configured and started.

The third and last point is to load the compiled SPEAR program into the FPGA. The SPEAR's parallel programmer module (see Figure 5.3) provides an internal FIFO for loading the firmware. The FIFO size is large enough to enable write burst and thus optimize the memory access but unfortunately not large enough to save the entire program. Therefore, the host is expected to split the compiled program into blocks and poll the programmer module for availability before transmitting a new block. As soon as a data is available in the programmer module, the internal SPEAR's bootloader moves it into its internal program memory. The end of a program is signalized using a special character. After reception of this character, the SPEAR performs an internal reset and starts operation according to the program previously loaded (see [Fle07]).

Data interface

Contrary to the configuration process, the data exchange between host and FPGA presents real-time constraints. Hence, during monitoring operation, the data are stored into local, small sized memories (128kB) and need to be periodically read out before an overflow occurs. Moreover, our tester presents the capability to perform deterministic replay with off-line generated data, which have to be transferred to the FPGA platform while ensuring that a memory underflow does not occur (otherwise a transmit deadline could be missed).

For both data transfer we use the 32MB SDRAM block available to the MPC5200 CPU as intermediate memory (see Figure 5.3). During monitoring operation, the monitored data are periodically saved into this larger memory area and transmitted afterward to the host, thus releasing the real-time constraints. The internal memory operations are triggered in real-time by the host and performed locally by the RedBoot (thus decreasing the execution time in comparison with memory operations to the host). Concerning replay, the entire bus traffic logfile is transferred to the SDRAM area prior test execution. During the experiment, the FPGA FIFO is filled (in real-time) using internal memory operations controlled by the host. This solution still presents some limitations with regard to the data volume. For our configuration, however, 32MB represents several minutes of logging and is enough for our purpose.

Data post-processing

After the completion of the experiment, the data are stored to the host as a memory dump. The first part of the operation is to extract the firmware diagnosis information from the monitored data. The two resulting files are then parsed and the following graphs are automatically generated:

- *cycle length*: The evolution of the cycle length both for the tester and the system (standard nodes plus tester) are illustrated. These two graphs pictures the stimulus and system response.

- *nodes' offset correction*: Two different illustrations of the local nodes' offset correction are available. On the one hand, this information is made globally available by our dedicated test application (see Section 5.1.2). This graph exactly reflects the reality, but data gathering is intrusive (requires bandwidth and dedicated node's local services). On the other hand, the nodes' offset correction is remotely extracted by our tester as explained in Section 4.4.1. The resulting plot includes measurement errors, but the information is gathered transparently. An additional advantage of these two concurrent plots concerns the evaluation of the remote approach.

- *nodes' rate correction*: The node's rate correction term is also made available by our test application (see Section 5.1.2) and then can also be plotted. The filtered cycle length (see Section 5.2.3) is also pictured.

- *physical and logical clock deviation*: This graph concerns the deviation of the physical and logical clocks. The current physical clock deviation information is provided by our test application. Each node periodically transmits its current quartz configuration, information that can be used to compute the node's physical clock deviation. The logical clock deviation information is obtained from the cycle length information.

- *system precision*: Finally, the precision of the system is approximated as described in Section 4.4.2 and made graphically available. Notice that our approach enables us to graph both the precision of the SUT (maximal clock state difference within the four standard nodes constituting the SUT) as well as the precision of the entire system (including the tester).

5.3 Evaluation of the FlexRay Clock Synchronization

The aim of this section is to present different experiments performed with the intention of evaluating the FlexRay clock synchronization mechanism. During this campaign, we perform deterministic replay and emulate oscillators that deviate within and beyond their tolerance interval. In particular, we will show how (a) frequency steps, (b) slow and (c) fast frequency drifts are handled by the FlexRay clock synchronization algorithm and might pose a threat to the system.

5.3.1 Overview

Each test campaign is described in five graphs (see the previous Section). The two firsts are representing the cycle length (respectively for the tester – graph A – and for the SUT – graph B). The third and fourth figures illustrate the internal nodes' *rate* (graph C) and *offset correction* (graph D) while the last figure represents the *system precision* (graph E).

We provide in the following two distinct precision measurements: The first one only concerns the four standard nodes (SUT) while the second one additionally includes the replay node. From the point of view of the application, only the first precision graph is interesting, since the replay node transmits frames without any content (the frames are discarded by the application). We will see that the precision of the SUT is not significantly worsened while applying the different stimuli. This is explained by the fact that the standard nodes are all submitted to the same stimuli and are expected to react identically (in fault-free case). However, this is not the case for the replay node, which does not implement a standard clock synchronization mechanism.

During the following experiments, the maximal offset correction is configured to 20 microticks (clock pulse; 25ns period) or 500ns while the maximal rate correction is bounded at 180 microticks (4500ns). Note that these boundaries are set arbitrary. Each node has been calibrated to the monitoring node at the beginning of the experiment as described in Section 5.1.3.

5.3.2 Testing the Offset Correction

The aim of this experiment is to test the response of the SUT to frequency jumps (stimulating the offset correction). For that, the replay node abruptly modifies its cycle length. More especially, we applied 25 steps of different amplitudes (from 25ns up to 625ns, with a granularity of 25ns). Figure 5.4 represents the cycle length of the replay node plus nodes under test, the offset and rate correction of each standard node, and the system precision. It can be observed that the precision of the SUT is always much better than 100ns, which can be expected, since the quartz have been calibrated and the nodes are computing the same convergence function with the same inputs. Furthermore, if the replay node is taken into account, the precision is punctually worsened just after changes of the cycle length; otherwise it stays under 200ns. As expected, the standard nodes turn into silent as soon as the precision relative to the replay node is larger than 1000ns, which represents in fact the configured maximal offset correction of 500ns or 20 microticks (remember that during a replay operation, the nodes are correcting half the clock state difference).

Different important behaviors can be observed here. First, the offset is corrected substantially only once after the cycle length has been changed. The rate correction, on the contrary, follows the cycle length changes. For the FlexRay protocol, the offset correction mechanism only corrects instantaneous clock state differences, while long term frequency differences are corrected by the rate correction mechanism. Second, the cycle length of the SUT presents an overrun just after a (positive or negative) step. This effect is due to the cumulative action of both correction mechanisms, on

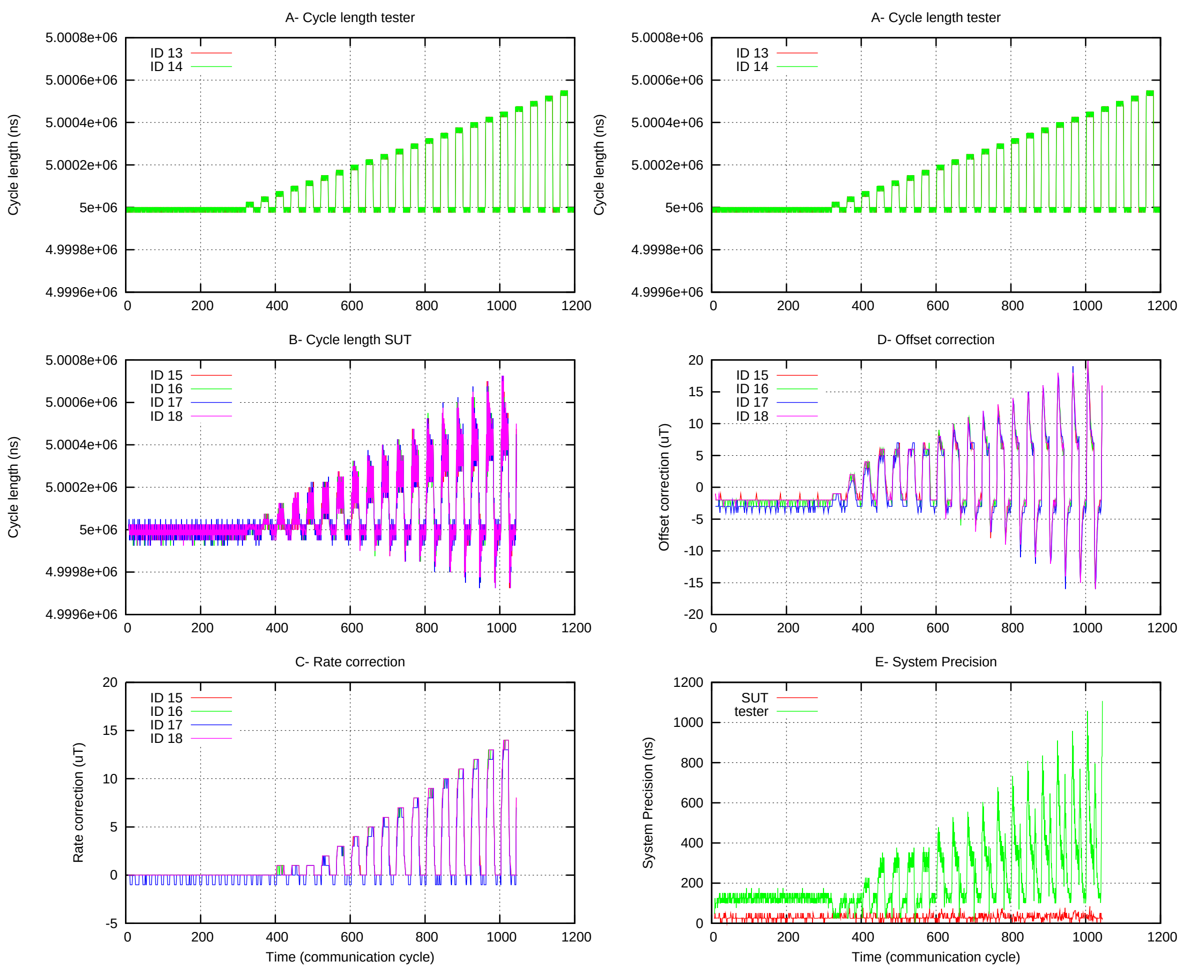


Figure 5.4: Maximal offset correction

one side to correct the delay accumulated (due to the frequency step in the past) and on the other side to adjust the rate of the logical clock (for the future). Third, the SUT nodes turned already to silent for a logical clock state deviation of 450ns (instead of the 500ns or 20 microticks configured). This last effect is a limitation of our test approach. Hence, the tester is performing deterministic replay and does not correct its time base (and more specifically the rate of its logical clock). The nodes, which have already accumulated a delay, only correct half of their logical clock rate and accumulate additional delay at the next computation step. This accumulation leads to the overrun of the offset correction for steps lower than the configured 500ns.

According to [Kop97] (p.52) the precision of the system has to be better than the granularity of the logical time base. Considering two nodes that are drifting apart and a time base granularity of one microsecond, we obtain a maximal offset correction of 500ns per node (the two nodes' time base are at most 1 microsecond or one time base unit apart). For this configuration, it corresponds to an instantaneous deviation (frequency step) of $100e^{-6}$ s/s, which is far below the $\pm 1500e^{-6}$ s/s defined in the FlexRay specifications. The situation is even worse for longer cycle length.

5.3.3 Testing the Rate Correction

During this experiment, the replay node generates a bus traffic whose cycle length is continuously slowed down from the configured 5000μs to 5006μs. This behavior emulates a logical clock drifting from an ideal value of 0 s/s down to $-1200e^{-6}$ s/s. The standard nodes stay synchronized as long as their internal rate correction values are lower than the configured 180 microticks (representing 4,5μs or a drift of $-900e^{-6}$ s/s). Figure 5.5 presents the results.

As expected, the nodes under test detect an error and turn into silent when the cycle length is longer than 5004,5μs (representing the configured 5000μs plus the 4,5μs maximal rate correction). This point in time (short after cycle 1200) corresponds to a required rate correction larger than the configured boundary. The offset correction, on the other side, does not go beyond its boundaries, since the instantaneous cycle length deviation (logical clock's state difference during two consecutive communication cycles) is quite small. Figure 5.5-E pictures the precision of the nodes under test and nodes under test plus replay node. It can be observed that the precision of the nodes under test stays under 100ns, while the clock state difference with the replay node can grow up to 500ns.

5.3.4 Emulating Fast Quartz Rate Changes

This last experiment illustrates the behavior of the FlexRay clock synchronization mechanism for quartz whose drift rate is changing fast. This change (quartz acceleration) represents the derivative of the quartz drift rate, similarly to the drift rate being the derivative of the quartz drift. The results are presented in Figure 5.6.

The FlexRay protocol does not implement a clock acceleration correction, and consequently the rate and offset correction have to be continuously corrected. We can observe that the rate correction stays within its configured bounds during the whole experiment. However, the offset correction does increase and lets the standard nodes turn to silent after communication cycle 2100. This behavior can be locally explained: The cycle length of the replay node is continuously stretched, thus (a) modifying the rate of the logical clocks and (b) making the replay node drift away from the standard nodes. The amplitude of this drift depends of the slew rate of the replay node's cycle length, which in turns influences the instantaneous drift between the tester and the

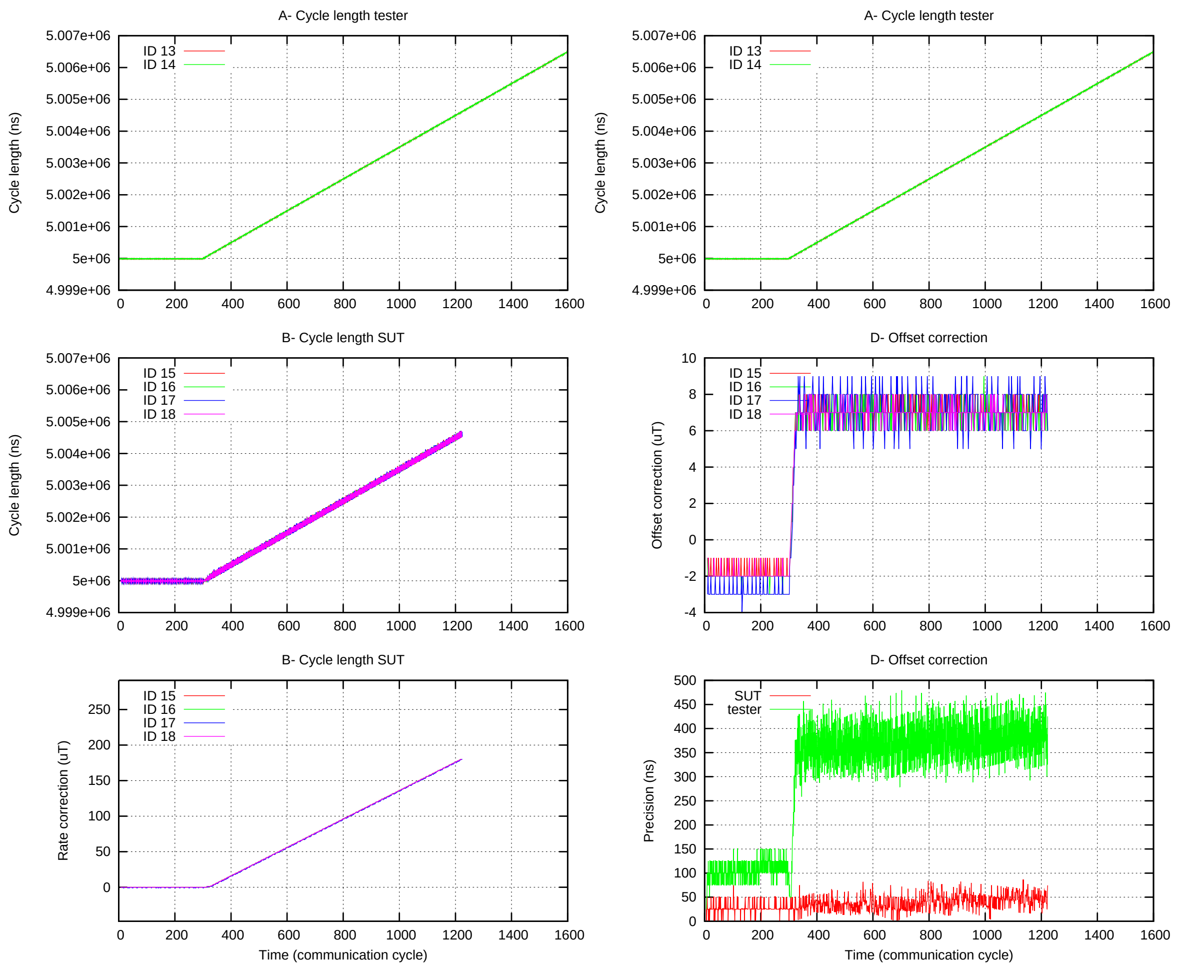


Figure 5.5: Maximal rate correction

nodes' logical clocks. For a large slew rate, the standard nodes have to apply a large offset correction that might be beyond their configured boundaries, thus generating an error. Notice furthermore that the absolute value of the quartz deviation is quite low at error time (approximately $200e^{-6}$ s/s). Important here is not the absolute quartz drift value but the relative quartz drift variation from a node to another.

An important behavior can be observed in Figure 5.6. Depending on whether the replay node is drifting toward or apart from the standard node's nominal quartz frequency, the precision (respectively the offset correction) presents a different amplitude. This effect results from the damping factor, which slightly modifies the rate correction toward the underlaying quartz. This effect is advantageous when the replay node is drifting toward to the quartz frequency, since the standard node's logical clock is (correctly) overcorrected. On the other hand, this leads to an undercorrection (thus worsening the system's precision) when the replay node is drifting apart from the quartz nominal frequency. This behavior has been successfully used in [AS07] to propose a method for non-intrusive remote quartz measurement.

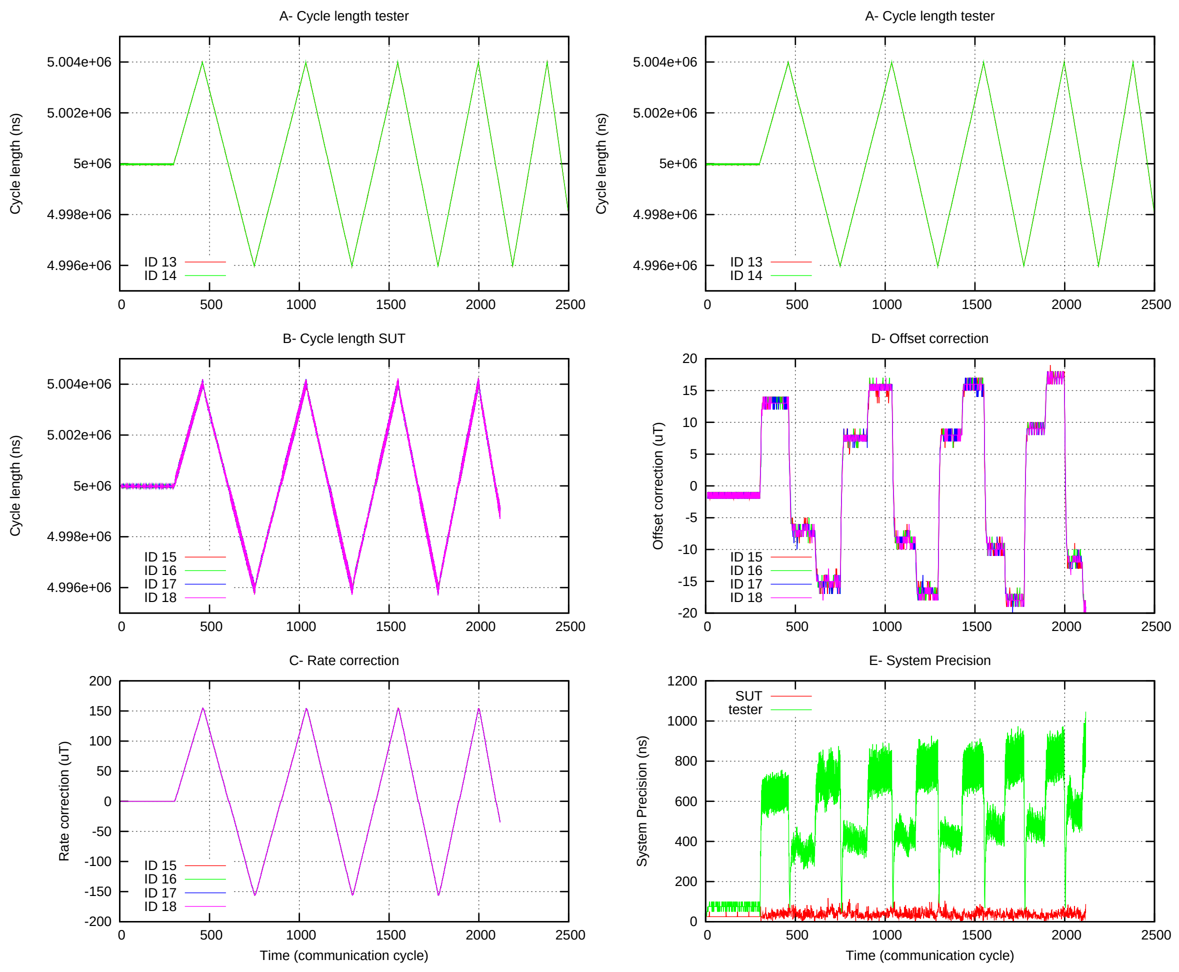


Figure 5.6: Fast quartz rate changes

Figure 5.6-E illustrates the precision of the system. We can observe here that the precision within the standard nodes is always better than 100ns, and that the nodes turn to silent when the clock difference to the replay node is larger than 1000ns. This behavior is coherent with the maximal offset correction term which is set to 20 microticks (see Section 5.3.2).

5.4 Chapter Summary

We presented in this chapter the experimental setup we used for the different evaluations during this work. Our setup principally consists of a four nodes system with a dedicated tester. The four nodes are providing two main services for improving the system controllability and observability. First, their oscillator can be individually controlled and thus different kinds of deviations can be emulated. Second, node's internal information such as current oscillator deviation or offset and rate correction are made globally available.

We presented additionally the internal architecture of our tester node. It basically consists of a dedicated measurement controller with an additional microcontroller which enables the stimulus computation on the fly. This capability is especially interesting when performing online tests because it can interrupt test execution when a condition is violated (see Section 4.2.5). We have described further a modular approach for the design of the firmware. The decomposition in basic blocks largely supports the fast and efficient generation of new test programs. Moreover, we have listed all information available during test execution as well as the standard outputs of our test environment.

The last part of this chapter presented a test campaign regarding the clock synchronization mechanism. During this study, the following stability behavior has been experimentally confirmed: as long as (1) the logical clock modification does not introduce an instantaneous state difference larger than the maximal offset correction, and (2) the logical clock rate does not differ from the node's quartz rate by more than the maximal rate correction, then the clock synchronization's error detection mechanisms will not let the nodes enter a silent mode. While condition (2) leads to the well-known requirement of bounded quartz drifts, condition (1) requires the quartz to present a stable frequency behavior. Specifically, the maximal quartz drift within a re-synchronization period must be bounded by half the granularity of the logical time base. This is usually the case due to the naturally stable behavior of oscillators. However, this has to be taken into account for our test approach or when the environment can not be assumed to be constant anymore (e.g., temperature step due to engine start).

Chapter 6

Conclusion

This is not the end. It is not even the beginning of the end.
But it is, perhaps, the end of the beginning.

Winston Churchill (1874 – 1965)

6.1 Summary

The main contribution of this thesis is the proposal for a remote, transparent test approach for time-triggered communication systems such as FlexRay and TTP/C. The benefit is the ability to test the availability of the error detection mechanisms as well as the reliability of the communication services concurrently to normal system operation, and thus to minimize the time interval between consecutive checks. This, in turns, helps minimize the probability of fault accumulation and of system failure. Moreover, the diagnosis information can be logged for further maintenance or directly used for system recovery.

The proposed test approach is divided into two main parts. On the one hand, passive monitoring from the bus traffic is sufficient to get diagnosis information from the node's transmission services. The cyclic transmission scheme provides a periodic stimulation of these mechanisms. On the other hand, active stimulation is required to gather status information from the nodes' internal clock synchronization mechanism as well as from the receive services. We use the fact that (i) correct reception is required to influence the clock synchronization and (ii) a tolerance region exists for the clock correction. Our stimulation strategy is based on the replay of frames which are syntactically correct but slightly shifted in the time domain. These frames, when correctly received, influence the nodes' internal clock synchronization. Finally, this can be observed according to the nodes' transmission point during the next cycles.

Since this test operation is intended to be performed concurrently to normal system operation, we have paid special attention to the transparency of our approach. We have shown that normal service delivery is not disturbed by our test approach due to the temporal composability attribute of time-triggered communication. The approach does not require any resource at the node level; neither their hardware nor their software are modified. At the architecture level, at most three communication slots must be available. Moreover, we have proved that deterministic replay is not a threat for the system (the system presents the same robustness).

The thesis presented in this work are supported by simulations and experimental evaluations. For this purpose, we have developed a test application that can modify the oscillator drifts online and that makes node internal status information globally available. Moreover, we have enhanced a tester node with a microcontroller core to enable the online computation of complex replay scenarios. The test setup was further used for evaluating the FlexRay clock synchronization with regard to atypical oscillator drifts. We have shown that not only bounded oscillator accuracy but also stable oscillator (whose frequency is not changing too fast with respect to the re-synchronization period) are required to avoid failure of the FlexRay clock synchronization algorithm.

6.2 Outlook

This thesis has led to complex analysis and different findings concerning the behavior of convergence-average based clock synchronization. Nevertheless, some questions are still open:

- **Formal proof of the FlexRay clock synchronization algorithm**: This communication protocol is also expected to be used for safety-critical applications. Although many studies have been published and much experience has been gained with this new protocol, to the best of our knowledge, no formal proof for this complex algorithm exists at this point in time. The proof presented in [LWL88] focuses on fault tolerant midpoint offset correction. While an additional rate correction mechanism is expected to improve the system precision (we have experimentally confirmed this behavior for fault-free systems), further studies are still required to ensure correct operation also in presence of faults.

- **Deterministic replay operation for external clock synchronization**: We have shown that deterministic replay operation can be used to safely control the global time. This specific test operation introduces an additional hierarchy between the nodes: the tester nodes synchronize internally while standard nodes are synchronizing both to the tester nodes and the other standard nodes and thus follow the tester nodes. For external synchronization, the tester nodes could be connected to an external time base which the cluster should synchronize to.

- **Efficient integration of this module test to a global test approach**: The proposed method is focused on the communication services. It would be interesting to investigate transparent test approaches for other modules and to combine them to cover the whole system. A challenging point will be to find a trade-off between the modules that can be efficiently tested using a generic approach and the modules that require a dedicated test approach (typically application tasks that are not standardized).

- **Fault-tolerant tester node**: The tester node proposed in this work has the ability of sending more than one “sync” frames within a communication cycle. From the network point of view, this node is emulating several nodes at once and a single failure of our tester might overrule the fault detection mechanisms and hazard the whole system. A fault-tolerant architecture for our tester would make our approach more robust (no single point of failure anymore).

Bibliography

[AAA+90] J. Arlat, M. Aguera, L. Amat, Y. Crouzet, J.-C. Fabre, J.-C. Laprie, E. Martins, and D. Powell. Fault Injection for Dependability Validation: a Methodology and some Applications. *IEEE Transactions on Software Engineering*, 16(2):166–182, Feb. 1990.

[ACK+03] J. Arlat, Y. Crouzet, J. Karlsson, P. Folkesson, E. Fuchs, and G.H. Leber. Comparison of Physical and Software-Implemented Fault Injection Techniques. *IEEE Transactions on Computers*, 52(9):1115–1133, Sept. 2003.

[ACL95] J. Arlat, Y. Crouzet, and J. Laprie. Fault Injection for Dependability Validation of Fault-Tolerant Computing Systems. In *Twenty-Fifth International Symposium on Fault-Tolerant Computing 'Highlights from Twenty-Five Years'*, pages 400–407, June 1995.

[ADA89] Motorwelt ADAC. Die ADAC Pannenstatistik 1989, April 1989.

[ADA06] Motorwelt ADAC. Die ADAC Pannenstatistik 2006, April 2006.

[Ade03] Astrit Ademaj. *Assessment of Error Detection Mechanisms of the Time-Triggered Architecture Using Fault Injection.* PhD thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 3/3/182-1, 1040 Vienna, Austria, 2003.

[AFS08] Eric Armengaud, Matthias Fuegger, and Andreas Steininger. Safe Deterministic Replay for Stimulating the Clock Synchronization Algorithm in Time-Triggered Systems (to appear). In 7^{th} *IEEE International Workshop on Factory Communication Systems (WFCS'08)*, page 10 p., May 2008.

[AIMP97] A.M. Amendola, L. Impagliazzo, P. Marmo, and F. Poli. Experimental Evaluation of Computer-Based Railway Control Systems. In *Twenty-Seventh Annual International Symposium on Fault-Tolerant Computing*, pages 380–384, June 1997.

[AKM$^+$01] J. Arlat, K Kanoun, H. Madeira, J.V. Busquests, T. Jarboui, A. Johansson, and R Linström. State of the Art. DBench project deliverables, Aug. 2001.

[Alb04] Amos Alber. Comparison of Event-Triggered and Time-Triggered Concepts with Regard to Distributed Control Systems. In *Embedded World*, pages 235–252, 2004.

[ALRL04] A. Avižienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, Jan. 2004.

[AP98] Emmanuelle Anceaume and Isabelle Puaut. Performance Evaluation of Clock Synchronization Algorithms. Technical Report RR-3526, INRIA - Institut National de Recherche en Informatique et en Automatique, 1998.

[APF02] Luis Almeida, Paulo Pedreiras, and Jose Alberto G. Fonseca. The FTT-CAN Protocol: Why and How. *IEEE Trans. on Industrial Electronics (TIE)*, 49(6):1189–1201, December 2002.

[Arm06] Eric Armengaud. Low Level Bus Traffic Replay for the Test and Debugging of Time-Triggered Communication Systems. In *IEEE Workshop on Design & Diagnostics of Electronic Circuits & Systems (DDECS'06)*, pages 155–156, Apr. 2006.

[ARS$^+$05] Eric Armengaud, Florian Rothensteiner, Andreas Steininger, Roman Pallierer, Martin Horauer, and Martin Zauner. A Structured Approach for the Systematic Test of Embedded Automotive Communication Systems. In *IEEE International Test Conference (ITC'05)*, pages 1–8, Nov. 2005.

[Arv94] K. Arvind. Probabilistic Clock Synchronization in Distributed Systems. *IEEE Trans. Parallel Distrib. Syst.*, 5(5):474–487, 1994.

[AS07] Eric Armengaud and Andreas Steininger. Non-Intrusive Remote Oscillator Drift Monitoring for Preventive Maintenance of Time-Triggered Systems. Technical Report RR-62/2007, Vienna University of Technology, Institute of Computer Engineering, Treitlstr. 3/3/182-1, 1040 Vienna, Austria (http://vmars.tuwien.ac.at/frame-papers.html), 2007.

[ASH$^+$04] Eric Armengaud, Andreas Steininger, Martin Horauer, Roman Pallierer, and Hannes Friedl. A Monitoring Concept for an Automotive Distributed Network - The FlexRay Example. In 7th *IEEE Workshop on Design & Diagnostics of Electronic Circuits & Systems (DDECS'04)*, pages 173–178, Strana Lesna, Slovakia, April 2004.

[ASH05] Eric Armengaud, Andreas Steininger, and Martin Horauer. Efficient Stimulus Generation for Remote Testing of Distributed Systems – The FlexRay Example. In Locia Lo Bello and Thilo Sauter, editors, *10th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'05)*, volume 1, pages 763–770, Catania, Italy, September 2005.

[ASH06] Eric Armengaud, Andreas Steininger, and Martin Horauer. Automatic Parameter Identification in FlexRay based Automotive Communication Networks. In 11th *IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'06)*, pages 897–904, Sept. 2006.

[AVFK01] J. Aidemark, J. Vinter, P. Folkesson, and J. Karlsson. GOOFI: Generic Object-Oriented Fault Injection Tool. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 83–88, July 2001.

[Ban99] R. Bannatyne. Semiconductor Developments for Automotive Systems. In *IEEE 49th Vehicular Technology Conference*, volume 2, pages 1392–1396, Jul. 1999.

[Bau01] Günther Bauer. *Transparent Fault Tolerance in a Time-Triggered Architecture.* PhD thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 3/3/182-1, 1040 Vienna, Austria, 2001.

[BBFT06] Jean-Luc Bechennec, Mikael Briday, Sebastien Faucou, and Yvon Trinquet. Trampoline, An OpenSource Implementation of the OSEK/VDX RTOS Specification. In 11th *IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'06)*, pages 62–69, Sep. 2006.

[BG03] S. Blanc and P.J. Gil. Improving the Multiple Errors Detection Coverage in Distributed Embedded Systems. In *Proceedings of the 22nd International Symposium on Reliable Distributed Systems*, pages 303–312, Oct. 2003.

[BMS87] Paul H. Bardell, William H. McAnney, and Jacob Savir. *Built-in test for VLSI: pseudorandom techniques.* Wiley-Interscience, New York, NY, USA, 1987.

[BP03] Alfredo Benso and Paolo Prinetto. *Fault Injection Techniques and Tools for Embedded Systems Reliability Evaluation.* Kluwer Academic Publishers, Norwell, MA, USA, 2003.

[BV89] V.P. Banda and R.A. Volz. Architectural Support for Debugging and Monitoring Real-Time Software. In *Euromicro Workshop on Real Time*, pages 200–210, June 1989.

[CLJ$^+$04] Ramesh Chandra, R.M. Lefever, K.R. Joshi, M. Cukier, and W.H. Sanders. A Global–State-Triggered Fault Injector for Distributed System Evaluation. *IEEE Transactions on Parallel and Distributed Systems*, 15(7):593–605, July 2004.

[CMS98] J. Carreira, H. Madeira, and J.G. Silva. XCEPTION: a Technique for the Experimental Evaluation of Dependability in Modern Computers. *IEEE Transactions on Software Engineering*, 24(2):125–136, Feb. 1998.

[Com97] Hewlett-Packard Company. Fundamentals of Quartz Oscillators. In *HP Application Note 200-2*, 1997.

[Cri91] F. Cristian. Reaching Agreement on Processor-Group Membership in Synchronous Distributed Systems. *Distributed Computing*, 4(4):175–188, 1991.

[DDS02] P.H. Deussen, G. Din, and I. Schieferdecker. An On-Line Test Platform for Component-Based Systems. In *Proceedings of the 27th Annual NASA Goddard/IEEE Software Engineering Workshop*, pages 96–103, Dec 2002.

[Dec06] Decomsys. Getting Started Guide, v1.3a. Technical report, Decomsys GmbH, Stumpergasse 48/28, 1060 Vienna, Austria, 2006.

[DGMK00] S. Deb, S. Ghoshal, V.N. Malepati, and D.L. Kleinman. Tele-Diagnosis: Remote Monitoring of Large-Scale Systems. *IEEE Aerospace Conference Proceedings*, 6:31–42, March 2000.

[DHSS95] Danny Dolev, Joseph Y. Halpern, Barbara Simons, and Ray Strong. Dynamic Fault-Tolerant Clock Synchronization. *Journal of the ACM*, 42(1):143–185, 1995.

[DJ94] Scott Dawson and Farnam Jahanian. Deterministic Fault Injection of Distributed Systems. In *Dagstuhl Seminar on Distributed Systems*, pages 178–196, 1994.

[DJM95] S. Dawson, F. Jahanian, and T. Mitton. A Software Fault Injection Tool on Real-Time Mach. In *Proceedings of the 16th IEEE Real-Time Systems Symposium*, pages 130–140, Dec. 1995.

[DR92] Paul S. Dodd and Chinya V. Ravishankar. Monitoring and Debugging Distributed Real-Time Programs. *Software - Practice and Experience*, 22(10):863–877, 1992.

[EBK03] Wilfried Elmenreich, Gnther Bauer, and Hermann Kopetz. The Time-Triggered Paradigm. In *Proceedings of the Workshop on Time-Triggered and Real-Time Communication*, Manno, Switzerland, Dec. 2003.

[Ebn98] C. Ebner. Efficiency Evaluation of a Time-Triggered Architecture for Vehicle Body-Electronics. In *Proceedings of the 10th Euromicro Workshop on Real-Time Systems*, pages 62–67, June 1998.

[EOAG+05] Ulrik Eklund, Örjan Askerdal, Johan Granholm, Anders Alminger, and Jakob Axelsson. Experience of Introducing Rference Architectures in the Development of Automotive Electronic Systems. In *SEAS '05: Proceedings of the second international workshop on Software engineering for automotive systems*, pages 1–6, New York, NY, USA, 2005. ACM Press.

[ESL01] M. El Shobaki and L. Lindh. A Hardware and Software Monitor for High-Level System-on-Chip Verification. In *International Symposium on Quality Electronic Design*, pages 56–61, March 2001.

[FA07] Wolfgang Forster and Eric Armengaud. A Novel Interconnection Approach for Globally Asynchronous Locally Synchronous Circuits. In 15th *Austrian Workshop on Microelectronics (Austrochip'07)*, pages 107–114, Oct. 2007.

[FAF06a] André V. Fidalgo, Gustavo R. Alves, and José M. Ferreira. A Modified Debugging Infrastructure to Assist Real-Time Fault Injection Campaigns. In *IEEE Workshop on Design & Diagnostics of Electronic Circuits & Systems (DDECS'06)*, pages 174–179, Apr. 2006.

[FAF06b] André V. Fidalgo, Gustavo R. Alves, and José M. Ferreira. A Modified Debugging Infrastructure to Assist Real Time Fault Injection Campaigns. In *IEEE Workshop on Design & Diagnostics of Electronic Circuits & Systems (DDECS'06)*, pages 174–179, Apr. 2006.

[FBH+06] Helmut Fennel, Stefan Bunzel, Harald Heinecke, Juergen Bielefeld, Simon Fuerst, Klaus-Peter Schnelle, Walter Grote, Nico Maldener, Thomas Weber, Florian Wohlgemuth, Jens Ruh, Lennart Lundh, Thomas Sanden, Peter Heitkaemper, Robert Rimkus, Jean Leflour, Alain Gilberg, Ulrich Virnich, Stefan Voget, Kenji Nishikawa, Kazuhiro Kajio, Klaus Lange, Thomas Scharnhorst, and Bernd Kunkel. Achievements and Exploitation of the AUTOSAR Development Partnership. In *Convergence 2006*, page 10, October 2006.

[Fle05] FlexRay. Flexray Communications Systems – Protocol Specification Version 2.1 (available at http://www.flexray.com). FlexRay Consortium, 2005.

[Fle07] Martin Fletzer. SPEAR2 Handbuch. Technical report, Vienna University of Technology, Institute of Computer Engineering, Treitlstr. 3/3/182-1, 1040 Vienna, Austria, 2007.

[FP05] Oliver Falkner and Christiane Picard. Quo vadis Kfz-Elektronik? *Elektronik Automotive*, 1:82–85, Jan. 2005.

[Gai86] Jason Gait. A Probe Effect in Concurrent Programs. *Source SoftwarePractice & Experience*, 16(3):225–233, Mar. 1986.

[Gal99] Thomas Galla. *Cluster Simulation in Time-Triggered Real-Time Systems.* PhD thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 3/3/182-1, 1040 Vienna, Austria, 1999.

[GB06] D.A. Gwaltney and J.M. Briscoe. Comparison of Communication Architectures for Spacecraft Modular Avionics Systems. NASA/TM2006214431, 2006.

[GLS06] Georg Gaderer, Patrick Loschmidt, and Thilo Sauter. Quality monitoring in clock synchronized distributed systems. In 6^{th} *IEEE International Workshop on Factory Communication Systems (WFCS'06)*, pages 13–22, Juni 2006.

[GS91] D. Galler and G. Slenski. Causes of Aircraft Electrical Failures. *Aerospace and Electronic System Magazine*, 6(8):3–8, Aug. 1991.

[Han05] P. Hansen. New S-Class Mercedes: Pioneering Electronics. *The Hansen Report on Automotive Electronics*, 18(8):1–2, October 2005.

[Han06] Alexander Hanzlik. SIDERA - a Simulation Model for Time-Triggered Distributed Real-Time Systems. *International Review on Computers and Software (IRECOS)*, 1(3):181–193, Nov. 2006.

[HD02] Wolfgang Huber and Martin Delvai. Peripherieanbindung an SPEAR Extension Modules. Technical report, Vienna University of Technology, Institute of Computer Engineering, Treitlstr. 3/3/182-1, 1040 Vienna, Austria, 2002.

[Het88] William Hetzel. *The Complete Guide to Software Testing, Second Edition.* Wiley, 1988.

[HJS01] Martin Hiller, Arshad Jhumka, and Neeraj Suri. An approach for analysing the propagation of data errors in software. In *DSN '01: Proceedings of the 2001 International Conference on Dependable Systems and Networks (formerly: FTCS)*, pages 161–172, Washington, DC, USA, 2001. IEEE Computer Society.

[Hor04] Martin Horauer. *Clock Synchronization in Distributed Systems.* PhD thesis, Technische Universität Wien, Faculty of Technical and Natural Sciences, Dept. of Automation, Treitlstr. 3/3/182-1, 1040 Vienna, Austria, 2004.

[HP03] R. Husemann and C.E. Pereira. BR-Tool: a Real-Time Bus Monitoring and Validation System for Fieldbus-Based Industrial Automation Applications. In *IEEE Conference on Emerging Technologies and Factory Automation*, pages 145–152, September 2003.

[HSF$^+$04] Harald Heinecke, Klaus-Peter Schnelle, Helmut Fennel, Jürgen Bortolazzi, Lennart Lundh, Jean Leflour, Jean-Luc Maté, Kenji Nishikawa, and Thomas Scharnhorst. AUTomotive Open System ARchitecture - An Industry-Wide Initiative to Manage the Complexity of Emerging Automotive E/E architectures. In *Proceedings of Convergence, SAE-2004-21-0042*, March 2004.

[HSR95] Seungjae Han, K.G. Shin, and H.A. Rosenberg. DOCTOR: an Integrated Software Fault Injection Environment for Distributed Real-Time Systems. In *Proceedings of the International Computer Performance and Dependability Symposium*, pages 204–213, Apr. 1995.

[HT98] G. Heiner and T. Thurner. Time-Triggered Architecture for Safety-Related Distributed Real-Time Systems in Transportation Systems. In *Twenty-Eighth Annual International Symposium on Fault-Tolerant Computing*, pages 402–407, June 1998.

[HTI97] Mei-Chen Hsueh, Timothy K. Tsai, and Ravishankar K. Iyer. Fault Injection Techniques and Tools. *Computer*, 30(4):75–82, April 1997.

[IEC98] IEC/SC65A. Iec61508-1 Functional Safety of Electrical Electronic Programmable Electronic Safety-Related Systems - Part 1 : General Requirements, 1998.

[IEE90] IEEE. IEEE Standard Glossary of Software Engineering Terminology. IEEE Std 610.12-1990, 1990.

[JAR$^+$94] E. Jenn, J. Arlat, M. Rimen, J. Ohlsson, and J. Karlsson. Fault Injection into VHDL Models: the MEFISTO Tool. In *Twenty-Fourth International Symposium on Fault-Tolerant Computing*, pages 66–75, June 1994.

[Joh98] D. John. OSEK/VDX History and Structure. In *OSEK/VDX Open Systems in Automotive Networks (Ref. No. 1998/523), IEE Seminar*, pages 2/1–214, Nov. 1998.

[KB03] Hermann Kopetz and Günther Bauer. The Time-Triggered Architecture. *Proceedings of the IEEE*, 91(1):112 – 126, January 2003.

[KBE$^+$95] H. Kopetz, M. Braun, C. Ebner, A. Kruger, D. Millinger, R. Nossal, and A. Schedl. The Design of Large Real-Time Systems: the Time-Triggered Approach. In *Proceedings of the 16th IEEE Real-Time Systems Symposium*, pages 182–187, Dec. 1995.

[KBG98] A. Kirschbaum, J. Becker, and M. Glesner. Run-Time Monitoring of Communication Activities in a Rapid Prototyping Environment. In *Ninth International Workshop on Rapid System Prototyping*, pages 52–57, June 1998.

[KGR89] Hermann Kopetz, G. Grunsteidl, and Johannes Reisinger. Fault-Tolerant Membership Service in a Distributed Real-Time System. In *IFIP WG10.4 Int'l Working Conference on Dependable Computing for Critical Applications*, pages 167–174, Aug. 1989.

[KHM05] N. Kandasamy, J.P. Hayes, and B.T. Murray. Time-Constrained Failure Diagnosis in Distributed Embedded Systems: Application to Actuator Diagnosis. *IEEE Transactions on Parallel and Distributed Systems*, 16(3):258–270, March 2005.

[KLD$^+$94] J. Karlsson, P. Liden, P. Dahlgren, R. Johansson, and U. Gunneflo. Using Heavy-Ion Radiation to Validate Fault-Handling Mechanisms. *IEEE Micro*, 14(1):8–23, Feb. 1994.

[KN97] H. Kopetz and R. Nossal. Temporal Firewalls in Large Distributed Real-Time Systems. In *Proceedings of the Sixth IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems*, pages 310–315, Oct. 1997.

[KO87] H. Kopetz and W. Ochsenreiter. Clock Synchronization in Distributed Real-Time Systems. *IEEE Transactions on Computers*, 36(8):933–940, 1987.

[KO02] H. Kopetz and R. Obermaisser. Temporal Composability. *Computing & Control Engineering Journal*, 13(4):156–162, Aug. 2002.

[Kop92] H. Kopetz. Sparse Time versus Dense Time in Distributed Real-Time Systems. In *Proceedings of the 12th International Conference on Distributed Computing Systems*, pages 460–467, June 1992.

[Kop97] Hermann Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications.* Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[Kop98] H. Kopetz. The Time-Triggered Model of Computation. In *Proceedings of the 19th IEEE Real-Time Systems Symposium*, pages 168–177, Dec. 1998.

[Kop99] H. Kopetz. Automotive Electronics. In *Proceedings of the 11th Euromicro Conference on Real-Time Systems*, pages 132–140, 1999.

[KS97] M. Krug and A.V. Schedl. New Demands for Invehicle Networks. In *Proceedings of the 23rd EUROMICRO Conference*, pages 601–605, Sep. 1997.

[KTWE03] Hermann Kopetz, Ken Tindell, Fabian Wolf, and Rolf Ernst. Safe Automotive Software Development. In *Design, Automation and Test in Europe Conference and Exhibition*, pages 616–621, Mar. 2003.

[Law97] Wolfhard Lawrenz. *CAN System Engineering. From Theory to Practical Applications.* Springer Verlag, New York, NY, USA, 1997.

[LH02] G. Leen and D. Hefferman. In-Vehicle Networks, Expanding Automotive Electronic Systems. In *IEEE Transaction on Computers*, pages 88–93, January 2002.

[LHD99] G. Leen, D. Heffernan, and A. Dunne. Digital Networks in the Automotive Vehicle. *Computing & Control Engineering Journal*, 10(6):257–266, 1999.

[LIN03] LIN. LIN Specification Package - Revision 2.0 (available at http://www.lin-subbus.org/). LIN Consortium, 2003.

[LMC87] T. J. LeBlanc and J. M. Mellor-Crummey. Debugging Parallel Programs with Instant Replay. *IEEE Transactions on Computers*, 36(4):471–487, Apr. 1987.

[LWL88] J. Lundelius-Welch and N. Lynch. A New Fault-Tolerant Algorithm for Clock Synchronization. *Information and Computation*, 77(1):1–36, 1988.

[Mah01] D. Mahrenholz. Minimal Invasive Monitoring. In *Proceedings of the Fourth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, pages 251–258, May 2001.

[Mar03] D. Marsh. Network Protocols Compete for Highway Supremacy. In *EDN Europe*, pages 26–38, June 2003.

[MH89] Charles E. McDowell and David P. Helmbold. Debugging Concurrent Programs. *ACM Comput. Surv.*, 21(4):593–622, 1989.

[MHB$^+$01] R. Mores, G. Hay, R. Belschner, J. Berwanger, C. Ebner, S. Fluhrer, E. Fuchs, B. Hedenetz, W. Kuffner, A. Krüger, P. Lohrmann, D. Millinger, M. Peller, J. Ruh, A. Schedl, and M. Sprachmann. FlexRay - The Communication System for Advanced Automotive Control Systems. In *Society of Automotive Engineers (SAE) 2001 World Congres*, March 2001.

[Moo65] Gordon E. Moore. Cramming More Components Onto Integrated Circuits. *Electronics*, 38:114–117, April 1965.

[MOS05] MOST. MOST Specification, Revision 2.4 (available at http://mostnet.de). MOST Cooperation, 2005.

[MRW03] Scott Mosely, Steve Randall, and Anthony Wiles. Experience within ETSI of the Combined Roles of Conformance Testing and Interoperability Testing. *The 3rd Conference on Standardization and Innovation in Information Technology*, pages 177–189, Oct 2003.

[MS97] Arup Mukherjee and Daniel P. Siewiorek. Measuring software dependability by robustness benchmarking. *IEEE Transactions on Software Engineering*, 23(6):366–378, June 1997.

[MSSP02] D. Mahrenholz, O. Spinczyk, and W. Schroder-Preikschat. Program Instrumentation for Debugging and Monitoring with AspectC++. In *Proceedings of the Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, pages 249–256, Apr. 2002.

[MT06] Rainer Makowitz and Christopher Temple. FlexRay – A Communication Network for Automotive Control Systems. In 6th *IEEE International Workshop on Factory Communication Systems (WFCS'06)*, pages 207–212, Juni 2006.

[Nic96] M. Nicolaidis. Theory of Transparent BIST for RAMs. *IEEE Transactions on Computers*, 45(10):1141–1156, Oct. 1996.

[Nob92] I Noble. EMC and the Automotive Industry. *Electronics & Communication Engineering Journal*, pages 263–271, October 1992.

[NSL05] Nicolas Navet and Franoise Simonot-Lion. *Fault Tolerant Services for Safe In-Car Embedded Systems*. CRC Press / Taylor & Francis, 2005.

[NSSLW05] Nicolas Navet, Ye-Qiong Song, Francoise Simonot-Lion, and Cedric Wilwert. Trends in Automotive Communication Systems. *Proceedings of the IEEE*, 93(6):1204–1223, June 2005.

[OBRB01] T. Olsson, N. Bauer, P. Runeson, and L. Bratthall. An Experiment on Lead-Time Impact in Testing of Distributed Real-Time Systems. In *Proceedings of the Seventh International Software Metrics Symposium*, pages 259–168, April 2001.

[POEL02] Philipp Peti, Roman Obermaisser, Wilfried Elmenreich, and Thomas Losert. An Architecture supporting Monitoring and Configuration in Real-Time Smart Transducer Networks. In *Proceedings of the IEEE Sensors 2002*, volume 2, pages 1479–1484, June 2002.

[PS99] B. Plale and K. Schwan. Run-Time Detection in Parallel and Distributed Systems: Application to Safety-Critical Systems. In *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems*, pages 163–170, June 1999.

[RAA02] M. Rodriguez, A. Albinet, and J. Arlat. MAFALDA-RT: a Tool for Dependability Assessment of Real-Time Systems. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 267–272, June 2002.

[RSH04] B. Rahbaran, A. Steininger, and T. Handl. Built-in Fault Injection in Hardware – The FIDYCO Example. In *Second IEEE International Workshop on Electronic Design, Test and Applications*, pages 327–332, Jan 2004.

[RSV06] Valrio Rosset, Pedro F. Souto, and Fransisco Vasques. A Group Membership Protocol for Communication Systems with both Static and Dynamic Scheduling. In 6^{th} *IEEE International Workshop on Factory Communication Systems (WFCS'06)*, pages 207–212, Juni 2006.

[Rub05] Enrico Rubiola. The Leeson Effect – Phase Noise in Quasilinear Oscillators (available at http://www.rubiola.org), Feb. 2005.

[Run98] W. Runge. Development Tendencies in Automotive Electronics. In *Twenty-Second IEEE/CPMT International Electronics Manufacturing Technology Symposium*, pages 5–9, Apr. 1998.

[RV04] M. Sonza Reorda and M. Violante. On-Line Analysis and Perturbation of CAN Networks. In *DFT '04: Proceedings of the Defect and Fault Tolerance in VLSI Systems, 19th IEEE International Symposium on (DFT'04)*, pages 424–432, Washington, DC, USA, 2004. IEEE Computer Society.

[SAE92] SAE. SAE Handbook. SAE International, 1992.

[Sav06] Sergio Matteo Savaresi. The Role of Real-Time Communication for Distributed or Centralized Architectures in Vehicle Dynamics Control Systems. In 6th *IEEE International Workshop on Factory Communication Systems (WFCS'06)*, pages 67–72, Juni 2006.

[SB06] Jason J. Scarlett and Robert W. Brennan. Re-evaluating Event-Triggered and Time-Triggered Systems. In 11th *IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'06)*, pages 655–661, Sept. 2006.

[SC90] Frank Schmuck and Flaviu Cristian. Continuous Clock Amortization Need not Affect the Precision of a Clock Synchronization Algorithm. In *PODC '90: Proceedings of the ninth annual ACM symposium on Principles of distributed computing*, pages 133–143, New York, NY, USA, 1990. ACM Press.

[Sch87] Fred B. Schneider. Understanding Protocols for Byzantine Clock Synchronization. Technical Report TR 87–859, Cornell University, Ithaca, NY, USA, Dept. of Computer Science, Upson Hall, Ithaca, NY 14853, 1987.

[Sch91] W. Schütz. On the Testability of Distributed Real-Time Systems. In *Proceedings of the Tenth Symposium on Reliable Distributed Systems*, pages 52–61, Oct. 1991.

[Sch92] W. Schütz. *The Testability of Distributed Real-Time Systems.* PhD thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 3/3/182-1, 1040 Vienna, Austria, 1992.

[Sch94a] Ulrich Schmid. Monitoring Distributed Real-Time Systems. *Real Time System Journal*, 7(1):33–56, 1994.

[Sch94b] W. Schütz. Fundamental Issues in Testing Distributed Real-Time Systems. *Real Time System Journal*, 7(2):129–157, 1994.

[Sch95] B.A. Schroeder. On-Line Monitoring: a Tutorial. *Computer*, 28(6):72–78, June 1995.

[Sch98] Klaus Schossmaier. *Interval-based Clock State and Rate Synchronization.* PhD thesis, Technische Universität Wien, Faculty of Technical and Natural Sciences, Dept. of Automation, Treitlstr. 3/3/182-1, 1040 Vienna, Austria, 1998.

[Sei07] Roman Seiger. miniUART Dokumentation. Technical report, Vienna University of Technology, Institute of Computer Engineering, Treitlstr. 3/3/182-1, 1040 Vienna, Austria, 2007.

[SFB$^+$00] D.T. Stott, B. Floering, D. Burke, Z. Kalbarczpk, and R.K. Iyer. NF-TAPE: a Framework for Assessing Dependability in Distributed Systems with Lightweight Fault Injectors. In *Proceedings of the Computer Performance and Dependability Symposium IPDS*, pages 91–100, March 2000.

[SL95] D.P. Sidhu and T.-K. Leung. Formal Methods for Protocol Testing: a Detailed Study. *IEEE Transactions on Software Engineering*, 15(4):413–426, Apr. 1995.

[Sma04] Idriz Smaili. *Real-Time Monitoring for the Time-Triggered Architecture.* PhD thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 3/3/182-1, 1040 Vienna, Austria, 2004.

[SMF98] Jr. Samson, J.R., W. Moreno, and F. Falquez. A Technique for Automated Validation of Fault Tolerant Designs Using Laser Fault Injection (LFI). In *Twenty-Eighth Annual International Symposium on Fault-Tolerant Computing*, pages 162–167, June 1998.

[SMM00] J. Swingler, J. Mcbride, and C. Maul. Degradation of Road Tested Automotive Connectors. *IEEE Transactions on Components and Packaging Technologies*, 23(1):157–164, Mar. 2000.

[SS03] C. Scherrer and A. Steininger. Dealing with Dormant Faults in an Embedded Fault-Tolerant Computer System. *IEEE Transactions on Reliability*, 52(4):512–522, Dec. 2003.

[ST87] T. K. Srikanth and Sam Toueg. Optimal Clock Synchronization. *Journal of the ACM*, 34(3):626–645, 1987.

[ST99] A. Steininger and C. Temple. Economic Online Self-Test in the Time-Triggered Architecture. In *Design and Test of Computers, IEEE*, volume 16, pages 81–89, July–Sept 1999.

[STB97] V. Sieh, O. Tschache, and F. Balbach. VERIFY: Evaluation of Reliability using VHDL-Models with Embedded Fault Descriptions. In *Twenty-Seventh Annual International Symposium on Fault-Tolerant Computing*, pages 32–36, June 1997.

[SV02] A. Steininger and J. Vilanek. Using Offline and Online BIST to Improve System Dependability - the TTPC-C Example. In *Proceedings of the 2002 IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages 277–280, Sept 2002.

[TBYS96] J.P. Tsai, Y.-D. Bi, S. Yang, and R. Smith. *Distributed Real Time System: Monitoring, Visualization, Debugging and Analysis.* Wiley-Interscience, New York, NY, USA, 1996.

[Tem99] Christopher Temple. *Enforcing Error Containment in Distributed Time-Triggered Systems: The Bus Guardian Approach.* PhD thesis, Technische Universität Wien, Institut für Technische Informatik, Treitlstr. 3/3/182-1, 1040 Vienna, Austria, 1999.

[TFCB90] J.J.P. Tsai, K.-Y. Fang, H.-Y. Chen, and Y.-D. Bi. A Noninterference Monitoring and Replay Mechanism for Real-Time Software Testing and Debugging. *IEEE Transactions on Software Engineering*, 16(8):897–916, August 1990.

[TH99] H. Thane and H. Hansson. Towards Systematic Testing of Distributed Real-Time Systems. In *Proceedings of the 20th IEEE Real-Time Systems Symposium*, pages 360–369, December 1999.

[TH00] H. Thane and H. Hansson. Using Deterministic Replay for Debugging of Distributed Real-Time Systems. In *Proceedings of the 12th Euromicro Conference on Real-Time Systems*, pages 265–272, June 2000.

[Tha00] Henrik Thane. *Monitoring, Testing and Debugging of Distributed Real-Time Systems.* PhD thesis, Mlardalen Real-Time Research Centre (MRTC), Department of Computer Engineering, Mlardalen University (MDH), 2000.

[Tho05] M. Thoss. Automated High-Accuracy Hybrid Measurement for Distributed Embedded Systems. In *Third International Workshop on Intelligent Solutions in Embedded Systems*, page May, 39–48 2005.

[TPDF99] C. Tanzer, S. Poledna, E. Dilger, and T. Fuhrer. A fault-tolearnce layer for distributed fault-tolerant hard real-time systems. In *Proceedings of* *the Annual IEEE Workshop on Fault-Tolerant Parallel and Distributed Systems*, page 21p, 1999.

[TTT05] TTA Group TTTech. Time-Triggered Protocol TTP/C High Level Specification Document Protocol Version 1.1, available at http://www.tttech.com/technology/specification.htm, 2005.

[Vas04] Thomas Vasek. Rechner auf Rädern. *Technology Review*, 7:80–41, July 2004.

[WNSSL04] Cedric Wilwert, Nicolas Navet, Ye-Qiong Song, and Francoise Simonot-Lion. *Design of automotive X-by-wire systems.* CRC Press, 2004.

[WW02] M. Weber and J. Weisbrod. Requirements Engineering in Automotive Development-Experiences and Challenges. In *IEEE Joint International Conference on Requirements Engineering*, pages 331–340, 2002.

[XbWP98] Brite-EuRaum 111 Program X-by Wire Project. X-by-Wire - Safety Related Fault Tolerant Systems in Vehicales, Final Report, 1998.

[Zei06] Kurt Zeillinger. Fehler im System. *auto touring, das ÖAMTC Magazin (in German)*, 6:6–10, 2006.

[ZP93] Enrico Zanoni and Paolo Pavan. Improving the reliability and safety of automotive electronics. *IEEE Micro*, 13(1):30–48, 1993.

[ZP98] A. Zahir and P. Palmieri. OSEK/VDX-Operating Systems for Automotive Applications. In *OSEK/VDX Open Systems in Automotive Networks (Ref. No. 1998/523), IEE Seminar*, pages 4/1–418, Nov. 1998.

[ZS02] M. Zulkernine and R.E Seviora. A Compositional Approach to Monitoring Distributed Systems. In *International Conference on Dependable Systems and Networks*, pages 763–772, June 2002.

[Zur05] R. Zurawski, editor. *The Industrial Communication Technology Handbook.* CRC Press, Feb. 2005.

Index

List of Publications

[1] Eric Armengaud, Matthias Fuegger, and Andreas Steininger. Safe Deterministic Replay for Stimulating the Clock Synchronization Algorithm in Time-Triggered Systems (to appear). In 7th *IEEE International Workshop on Factory Communication Systems (WFCS'08)*, page 10 p., May 2008.

[2] Eric Armengaud. Experimental Evaluation of the FlexRay Clock Synchronization Service. In *20. ITG/GI/GMM Workshop, Testmethoden und Zuverlässigkeit von Schaltungen und Systemen*, pages 85–90, Feb. 2008.

[3] Wolfgang Forster and Eric Armengaud. A Novel Interconnection Approach for Globally Asynchronous Locally Synchronous Circuits. In 15th *Austrian Workshop on Microelectronics (Austrochip'07)*, pages 107–114, Oct. 2007.

[4] Eric Armengaud, Andreas Steininger, and Alexander Hanzlik. The Effect of Quartz Drift on Convergence-Average based Clock Synchronization. In 12th *IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'07)*, pages 1123–1130, Sept. 2007.

[5] Martin Horauer, Eric Armengaud, and Andreas Steininger. Concepts and Tools for the Test of the Communication Sub-System of Time-Triggered Distributed Embedded Systems. In *International Conference on Design Engineering Technical Conferences & Computers and Information in Engineering (ASME 2007)*, page 8, Sept. 2007.

[6] Eric Armengaud, Andreas Steininger, and Martin Horauer. Automatic Parameter Identification in FlexRay based Automotive Communication Networks. In 11th *IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'06)*, pages 897–904, Sept. 2006.

[7] Eric Armengaud and Andreas Steininger. Pushing the Limits of Remote Online Diagnosis in FlexRay based Networks. In 6th *IEEE International Workshop on Factory Communication Systems (WFCS'06)*, pages 45–54, June 2006.

[8] Eric Armengaud and Andreas Steininger. A Remote and Transparent Approach for the Test and Diagnosis of Automotive Networks. In *Junior Scientist Conference (JSC'06)*, pages 11–12, Apr. 2006.

[9] Eric Armengaud. Low Level Bus Traffic Replay for the Test and Debugging of Time-Triggered Communication Systems. In *IEEE Workshop on Design & Diagnostics of Electronic Circuits & Systems (DDECS'06)*, pages 155–156, Apr. 2006.

[10] Eric Armengaud. ExTraCT: A New Approach for the Transparent Test of Time-Triggered Communication Services. In *18. ITG/GI/GMM Workshop, Testmethoden und Zuverlässigkeit von Schaltungen und Systemen*, pages 63–67, March 2006.

[11] Eric Armengaud, Florian Rothensteiner, Andreas Steininger, Roman Pallierer, Martin Horauer, and Martin Zauner. A Structured Approach for the Systematic Test of Embedded Automotive Communication Systems. In *IEEE International Test Conference (ITC'05)*, pages 1–8, Nov. 2005.

[12] Eric Armengaud, Andreas Steininger, and Martin Horauer. Efficient Stimulus Generation for Remote Testing of Distributed Systems – The FlexRay Example. In Locia Lo Bello and Thilo Sauter, editors, *10th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'05)*, volume 1, pages 763–770, Catania, Italy, September 2005.

[13] Eric Armengaud, Andreas Steininger, and Martin Horauer. An Efficient Test and Diagnosis Environment for Communication Controllers. In *Proceedings of Austrochip 2005*, pages 67–70, October 2005.

[14] Eric Armengaud, Florian Rothensteiner, Andreas Steininger, and Martin Horauer. A Flexible Hardware Architecture for Fast Access on Large Non-Volatile Memories. *IEEE Workshop on Design & Diagnostics of Electronic Circuits & Systems (DDECS'05)*, Apr. 2005.

[15] Eric Armengaud, Andreas Steininger, and Martin Horauer. A Method for Bit Level Test and Diagnosis of Communication Services. In 8^{th} *IEEE Workshop on Design & Diagnostics of Electronic Circuits & Systems (DDECS'05)*, pages 69–74, Hungary, April 2005.

[16] Roman Pallierer, Martin Horauer, Martin Zauner, Andreas Steininger, Eric Armengaud, and Florian Rothensteiner. A Generic Tool for Systematic Tests in Embedded Automotive Communication Systems. In *Embedded World 2005 Conference*, pages 42–49, Germany, Feb. 2005.

[17] Martin Horauer, Florian Rothensteiner, Martin Zauner, Eric Armengaud, Andreas Steininger, Hannes Friedl, and Roman Pallierer. An FPGA based SoC Design for Testing Embedded Automotive Communication Systems employing the FlexRay Protocol. In *Proceedings of the Austrochip 2004 Conference*, pages 119–125, September 2004.

[18] Eric Armengaud, Andreas Steininger, Martin Horauer, and Roman Pallierer. A Layer Model for the Systematic Test of Time-Triggered Automotive Communication Systems. In 5th *IEEE International Workshop on Factory Communication Systems (WFCS'04)*, pages 275–283, Austria, September 2004.

[19] Eric Armengaud, Andreas Steininger, Martin Horauer, and Roman Pallierer. Design Trade-offs for Systematic Tests of Embedded Communication Systems. In *Supplemental Volume of the International Conference on Dependable Systems and Networks (DSN'04)*, pages 118–119, Florence, Italy, June 2004.

[20] Eric Armengaud, Andreas Steininger, Martin Horauer, Roman Pallierer, and Hannes Friedl. A Monitoring Concept for an Automotive Distributed Network - The FlexRay Example. In 7th *IEEE Workshop on Design* & *Diagnostics of Electronic Circuits* & *Systems (DDECS'04)*, pages 173–178, Strana Lesna, Slovakia, April 2004.

[21] Eric Armengaud. Commercial Break Detection Device. US. Patent Application: US 2001-0055463, 2001.

Curriculum Vitae

Eric Armengaud

Oct. 6th1978	Born in Paris, France
Sept. 1984 – Jul. 1989	Elementary school Nogent sur Marne, France
Sept. 1989 – Jul. 1996	High School, Edouard Branly Nogent sur Marne, France
Jul. 1996	Graduation from High School with distinction
Sept. 1996 – Jul. 2002	Studies of Computer Sciences ESIEE Paris
Sept. 1999 – Aug. 2000	Working student at Philips, Vienna development of μC-based modules
May 2001 – Jul. 2001	Working student at Siemens, Vienna development of DSP-based modules
Jul. 2002	Master's Degree in Computer Sciences
Aug. 2002 – Aug. 2007	Digital Designer Decomsys, Vienna
Since Sept. 2003	PhD Studies and research assistant Vienna University of Technology