Concepts and Tools for the Test of the Communication Sub-System of Time-Triggered Distributed Embedded Systems

Article · January 2007

DOI: 10.1115/DETC2007-34439

3 authors:

Martin Horauer
Fachhochschule Technikum Wien
55 PUBLICATIONS **375** CITATIONS
SEE PROFILE

Eric Armengaud
AVL LIST GMBH
82 PUBLICATIONS **309** CITATIONS
SEE PROFILE

Andreas Steininger
TU Wien
150 PUBLICATIONS **786** CITATIONS
SEE PROFILE

Proceedings of IDETC/CIE 2007
ASME 2007 International Design Engineering Technical Conferences &
Computers and Information in Engineering Conference
September 4-7, 2007, Las Vegas, Nevada, USA

DETC2007-34439

CONCEPTS AND TOOLS FOR THE TEST OF THE COMMUNICATION SUB-SYSTEM OF TIME-TRIGGERED DISTRIBUTED EMBEDDED SYSTEMS

Martin Horauer
Department of Embedded Systems
University of Applied Sciences Technikum Wien
Höchstädtplatz 5, A-1200 Vienna, Austria
Email: horauer@ieee.org

Eric Armengaud, Andreas Steininger
Department of Computer Engineering
Vienna University of Technology
Treitlstr. 3, A-1040 Vienna, Austria
Email: {armengaud, steininger}@ecs.tuwien.ac.at

ABSTRACT

With the adoption of FlexRay, the time triggered paradigm has been widely accepted by the automotive industry as a means to tackle the requirements of future automotive electronics. However, when compared with traditional event-triggered systems like CAN, the benefits of higher reliability come at the cost of increased complexity during system design. In fact, to support the development of these systems adequate tool-support will be mandatory. In this paper we discuss the requirements and concepts for and present an implementation of a test and diagnosis toolset for FlexRay-based automotive distributed networks. Next to data monitoring and recording, this toolset provides facilities for fault injection and replay. Hence, the presented implementation is tailored for an embedded test and fault diagnosis and will enable an assessment of the reliability and dependability of future automotive solutions.

INTRODUCTION

In recent years embedded electronic hard- and software has become the most important enabler for innovations in the automotive domain. In fact, modern cars employ up to 70 electronic control units that are interconnected using various different fieldbus systems like, e.g., the *Controller Area Network* (CAN), the *Local Interconnect Network* (LIN) or *ByteFlight*. The shared information provides an added value and allows the establishment of new applications, e.g. by combining speed with steering information or by relating sensor data from the wheels' rotation to improve the lane keeping stability of the car [1]. Hence, today's vehicle networks have transformed automotive control tasks, once the domain of mechanical or hydraulic components, into truly distributed electronic systems. Replacing rigid mechanical components with dynamically configurable electronic elements triggers an almost organic, system wide level of integration. As a result, the cost of advanced systems should plummet. However, car manufacturers have soon identified the network architecture as the crucial point in this context. Complex applications have shown that the above-mentioned fieldbus systems become a critical bottleneck. To that end an industrial consortium has established the communication protocol FlexRay to overcome these limitations, see [2].

Relying on both the time- and event-triggered paradigms, FlexRay promises reliability and fault-tolerance aspects with the bandwidth to serve the needs of a communication backbone and the flexibility for the coupling of sensor/actuator systems required for future automotive solutions. Since FlexRay will be introduced for safety-critical applications, e.g. X-by-wire systems, where a failure can lead to severe consequences, means for the evaluation of dependability properties are required. Clearly, testing is essential in order to evaluate whether the system is correctly implemented and configured and will react as expected in its future field environment, even in case of faults. While methods for testing of the computing nodes themselves on the one side and the bus on the other side do exist, a unified, accurate and systematic test approach on the system level is required that does not only consider the function of these singular components in isola-

tion. The test efforts for distributed embedded systems rise with system complexity due to the geographic diversity, the large system state space and the large fault range. Hence, in practice one is either faced with an excessive test duration or a very limited test coverage. Furthermore, experience shows that problems with interaction of "fault-free" components are becoming increasingly relevant, a problem that is further aggravated by the large number of new product variants. In this context, the contributions of this paper are (1) the concept for a systematic, structured test approach to handle the test complexity for FlexRay-based distributed embedded systems, and (2) the presentation of a system architecture for the implementation of a tester along with some use-cases and experimental results conducted with a prototype implementation.

The following section presents some common aspects that need to be considered for testing FlexRay-based systems. Afterwards we present some related work and tools before we turn to the concepts and architecture of our test-suite. The subsequent description of use-cases will then show how the tool-set can be used in practice. Finally, some test experiments and results acquired with our prototype implementation are outlined before we conclude the paper.

TESTING DISTRIBUTED EMBEDDED SYSTEMS

A test concept for distributed embedded systems must address the following aspects:

Controllability defines (1) whether the *system-under-test* (SUT) can be put into a test mode and (2) how the input test sequences can be applied to the SUT. In the context of automotive electronics, the first item (*a.k.a. offline test*) is by its nature easier to achieve during application development in the lab, before system operation or during maintenance. It is, however, much more problematic for an *online test*.
The second item is often hard to achieve due to accessability restrictions of the distributed embedded system. Hence, some kind of *remote-test* that provides the test-stimuli via the fieldbus system is desirable here. Note that remote testing is applicable for both offline and online modes.

Observability defines where and how the outputs of the system and potential internal states of the SUT can be observed without influencing the system itself (*a.k.a. probe-effect*). Furthermore, it addresses how the output information of the test can be collected for a succinct evaluation and interpretation. Again the employment of the fieldbus is desirable for this purpose. For a remote-test some kind of information flow from the SUT to the tester must occur over the field bus in order to transfer the desired status information.

Prior to the test of a distributed embedded system, the definition of a *fault-hypothesis* is mandatory. It makes some assumptions about the types and numbers of faults that are to be expected; hence, it partitions the fault-space into covered and uncovered faults. In particular, the fault hypothesis addresses the error containment regions (or fault containment regions when one can assume fail-silent behavior), the failure modes, the failure frequency and the error detection and recovery mechanisms, see [3] for elaborate definitions.

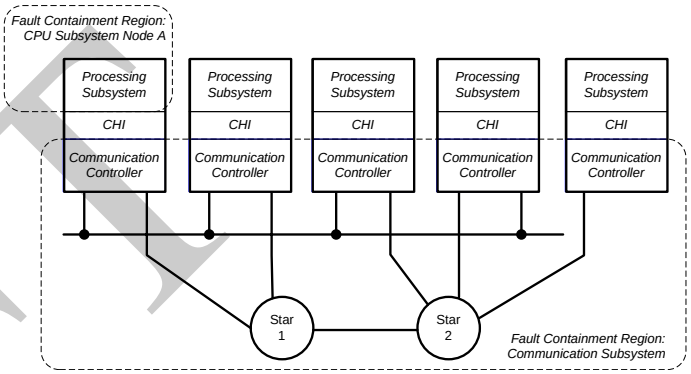

Figure 1. A FlexRay system showing some fault containment regions

For distributed embedded systems – like FlexRay – that follow the time-triggered paradigm the fault containment regions are defined by so called temporal firewalls [4]. They block control information and let pass data solely. An example for such a temporal firewall is the controller-host-interface (CHI) at every node, cf. Fig. 1. This interface is a kind of dual-ported memory residing between the network controller and the node's processing unit. Access from either side is only allowed at predefined instants in time defined by the time-triggered schedule. Thus, in principle, every fault-containment region can be tested independently in isolation from each other, e.g., using node tests and a test of the communication subsystem, respectively. Furthermore, a system-test of the entire system may be required for the case when an actual implementation deviates slightly from the specification or when configuration faults shall be detected.
Tools required for such tests should provide capabilities for monitoring, data generation and injection, and replay as well as pattern-generation, and data-analysis.

RELATED WORK

Tools to *monitor* bus-traffic are in widespread use for testing, debugging and optimization of various communication services and protocols, see e.g. [5, 6]. Several different implementations and approaches exist for LANs ranging from pure software monitors (e.g. tcpdump[1], ethereal[2], wireshark[3], libnet [7]) at different levels of abstraction up to sophisticated approaches employing some kind of dedicated COTS network interface hard- and software. Examples for COTS tools for network analysis

[1] http://www.tcpdump.org
[2] http://www.ethereal.com
[3] http://www.wireshark.org

and monitoring tools tailored to fieldbus systems like CAN, LIN, MOST, FlexRay or TTP/C are CANalyzer[4] or TTView[5]. Implementation issues of these and similar tools along with some use-case scenarios can be found e.g. in [8, 9]. The majority of these monitoring and network analysis tools operate at or above the medium access layer employing COTS network controllers and device drivers in a promiscuous mode where all (even corrupt) frames are forwarded to the processing CPU.

For testing purposes a mechanism in the reverse direction that allows some kind of *data generation* and *injection* or *replay* is required as well. Whereas the first two mechanisms typically allow arbitrary forms of data to be generated and injected to the communication subsystem, the latter provides some facilities to re-enact previously recorded scenarios. As with the monitoring solutions data generators are either implemented as software using COTS networking hardware (although some kind of special device drivers may be required here), or at or next to the physical layer employing some kind of arbitrary waveform generators and/or programmable pattern generators. The generated and injected data can be both – correct or faulty ones. Hence, a subset of the existing fault injection tools and techniques fall into this category as well, see [10] for a survey.

Replay of recorded data can be done either in causal order of the recorded events, or by obeying both the causal and temporal order. For event triggered communication systems a clock synchronization of the distributed system and timestamping of outgoing packets must be enforced; especially when intermediate active network devices are present, since these devices influence both the causal and the temporal order. This requirement, however, can be relaxed for time-triggered communication since clock synchronization is an integral communication service and the instants in time, when messages are transmitted are known a priori. Implementations in this regard are provided, e.g., by the CANalyzer or by [11, 12] for distributed applications in different domains.

THE TEST CONCEPT

To tackle the test complexity we first introduced a fine-structured layer-model of FlexRay, see [13]. In particular, we structured the layers of the communication subsystem – in analogy to the ISO/OSI model – into a set of fine granular functional entities: *Abstraction levels* a_i and *mechanisms* m_i. Herein, an abstraction level represents a detailed view of the system whereas a mechanism provides simple — ideally atomic — services that are controlled by several inputs a_{i-1} and optional configuration parameters c_i and produce one or more outputs a_i, cf. Fig. 2. An example for such a mechanism m_i is a decoder that may be configured and parameterized via suitable decoder parameters c_i;

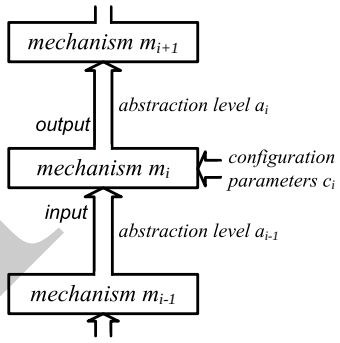

Figure 2. Mechanisms and abstraction levels

input to this mechanism is the encoded bit stream at the abstraction level a_{i-1} and output is the raw bit stream at the abstraction level a_i.

To reduce the test complexity we monitor bus traffic – using an assumed fault free embedded tester node – at a number of discrete abstraction levels. Our test proceeds from the lowest abstraction level towards the highest. Herein, an error detected at an abstraction level a_i may stem either from an erroneous input a_{i-1}, erroneous configuration c_i or a defective mechanism m_i. Assuming that all mechanisms below have passed their test properly, one can reason that the fault originates from the corresponding mechanism or one above in the sending path of the respective remote node, cf. Fig. 3. In this way the sending path of a remote node can be tested and diagnosed quite well. The test the receive path, however, requires additional provisions. In particular the remote node's reaction to a test stimulus applied on the fieldbus must be observed. This requires (i) direct access to that node for read-out of the respective mechanisms' outputs, or (ii) some kind of loop-back of this information (i.e. the node under test replies to the test stimulus by sending its internal status over the fieldbus), or (iii) some kind of regular status-output that provides the relevant information. When (iii) is not available, option

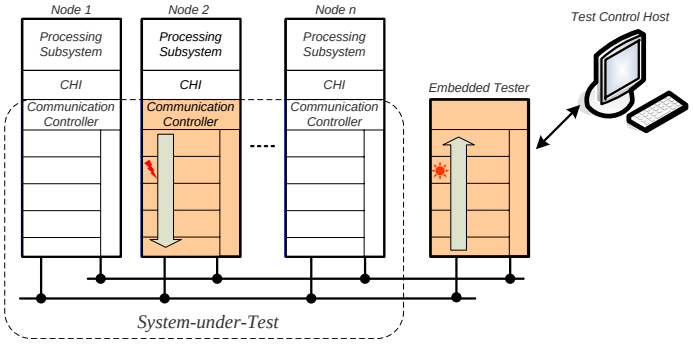

Figure 3. Remote Testing

[4] http://www.vector-informatik.com
[5] http://www.tttech.com

(ii) seems to be the desired solution here, given the limited accessibility of automotive controllers that rules out (i). Loop-back can be provided as a service of the node's CPU relaying information from the receive to the transmit path. A less intrusive loop-back can eventually be exploited via the clock synchronization services [14, 15]; this approach is currently under investigation.

The advantages of this approach are (i) the fault space can be decomposed in an intuitive way into orthogonal basic faults, (ii) a systematic exploration of the fault space is feasible as long as the mechanisms are independent from each other, and (iii) physical faults can be mapped to unique "syndromes", i.e. combinations of basic faults. As a limitation of this approach, however, simple common mode faults, e.g. power supply faults or outages of the clocking source, are not directly included, but can be represented by their respective syndromes.

IMPLEMENTATION OF THE TEST-TOOL

Our prototype implementation consists of an embedded part and a host part. The embedded part consists of an FPGA that provides FlexRay receivers, transmitters, a protocol engine, a timer, and monitoring and replay units that allow data acquisition/injection at different levels of abstraction, see Fig. 4. Furthermore, we developed the firmware for a RTAI Linux running on top of an embedded ARM CPU core. Data can be stored/retrieved either to/from a flash disk for field operation or via a Fast Ethernet connection to/from a remote control host, see [16–18] for further implementation details.

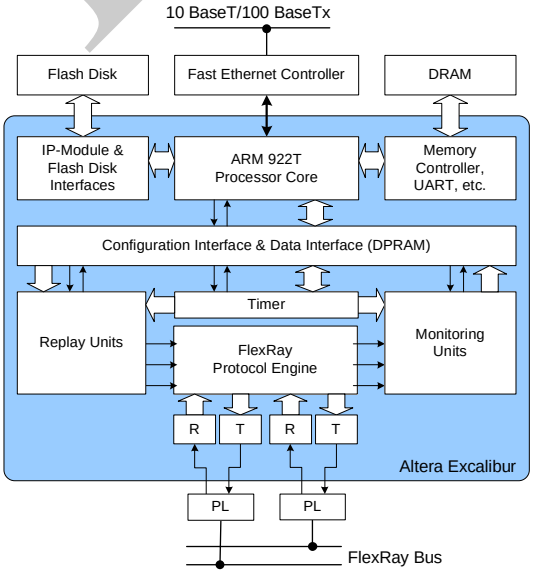

Figure 4. Architecture of the embedded tester hardware

Figure 5. Screenshot of the User Interface

The host part consists of a library to manage the flow of data and control information, a graphical user interface to present the acquired data to the user in various different ways, e.g. raw data, scope, gauges, or in form of a dial widget (see Fig. 5 for a screenshot). In addition, the user interface is used to control the embedded tester, in particular it allows for the definition of complex triggers, the configuration of filters, the selection of abstraction levels and it controls the flow of data. To support replay, monitored data can be either modified and/or generated; the latter is per default made conform to the FlexRay specification [2]. The entire monitoring/replay process can be scripted and exported thus enabling automatic test scenarios.

USE CASES

The above described test approach for FlexRay-based systems will be usable in various stages in the life-cycle of a product, e.g., during development or later-on whilst maintenance. Depending on the actual stage different tools and test approaches for the various use-cases will be required. The following description lists some use-cases for the test concepts and tools as outlined in the previous sections:

For an early application development usually only the required nodes and infrastructure for the tasks at hand are available. Hereby, the developer is confronted with the task of *verification* that aims at revealing implementation faults. Monitoring is used to check the application's output on the bus, and replay using data generation may be used to emulate the corresponding environment and provide a working bus infrastructure. E.g., one can emulate with our tester multiple sensor nodes relating the information of the wheels' states for the development of an ESP system. Hence, an emphasis for the test environment at this stage is set on flexibility.

In a later stage during the development cycle the applications executing on various electronic control units from differ-

ent vendors will be coupled with each other. Hereby, a focus of the test will be set on the *inter-operability* and protocol *conformance* of the different nodes and applications running on top of them. The first one tries to prove the end-to-end functionality between (at least) two communication systems according to the standard(s) on which those systems are based. In contrast, conformance testing evaluates whether the parameters of an implemented system conform to the protocol specification – in our case FlexRay. Herein, correct system operation has to be validated for the entire range of possible inputs and parameters. For both use-cases parameter identification and validation is useful. This can be achieved by the combination of monitoring, measurements and replay followed by a statistical analysis and interpretation, see [19].

After system integration and verification the test focus will undoubtedly be set on *robustness testing*. Hereby, one scrutinizes the system behavior in the presence of erroneous and/or stressful input conditions. Prior to test execution an appropriate fault hypothesis must be defined. With that in mind one can succinctly generate and/or manipulate the bus traffic and inject faults at the corresponding instants and locations. With the help of the monitoring tools the output of these experiments is recorded and analyzed in order to check whether the respective fault tolerance mechanisms triggered.

As an optimization step *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. Here the main focus is on measuring the performance of a system for a well defined set of tasks, and not to verify or validate a system. Monitoring alone will usually serve for this purpose.

During the operational phase of a car *maintenance testing* is used to detect and localize faults. The scope is typically narrowed to (physical) faults that occurred since the last maintenance actions (e.g., due to ageing effects). In practice, e.g., some kind of tester node is coupled to the system-under-test as illustrated in Fig. 3 and a set of tests are executed in order to unveil defective components. Therefore, usually status information after a set of pre-defined tests is read-out.

Verification, conformance, inter-operability, and maintenance tests are qualitative tests with the purpose to prove 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. The user interface has to present the information in a way appropriate for the particular case.

EXPERIMENTAL EVALUATION

The aim of the following experiments is to illustrate the adequacy of our test environment for some of the above use-cases.

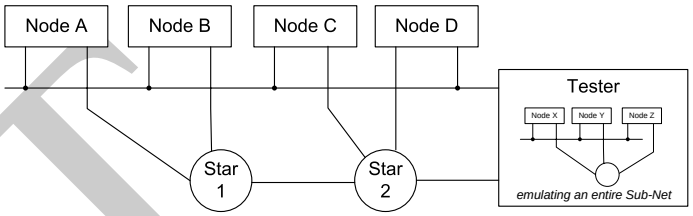

Figure 6. Experimental Setup

In a first experimental series we investigated whether the clock synchronization mechanism as implemented by a set of COTS electronic control units (ECUs) from our industrial partner Decomsys GmbH[6] conforms to the FlexRay standard. Therefore we coupled our tester to a SUT cluster, cf. Fig. 6.

Herein, the tester emulated several nodes of its own generating SYNC messages. In this way we were able to dictate the clock synchronization thus forcing all other nodes to synchronize towards the tester. By applying a deterministic replay (forcing the point in time the tester's frames are sent without taking into account the SUT [20]) and monitoring the bus traffic at the same time we could investigate whether the remaining nodes could follow suite. In particular we tuned the maximum rate and step correction values until the nodes in the SUT switched to silent mode. Fig. 7 illustrates the results of the clock step experiment and Tab. 1 summarizes our findings. Herein the values given in the "Specification" column reflect the theoretical values from the FlexRay specification for the configuration of *pRateCorrectionOut* with 180μT and *pOffsetCorrectionOut* with 20μT (μT stands for microtick). As expected, the nodes of the SUT turned silent when the shift for the rate correction exceeded 4.5μs.

Table 1. Results of the Clock Synchronization Experiments

Parameter	Results	Specification
max. rate correction	4.5μs	4.5μs
max. step correction	450 ns	500 ns

For the step correction experiment our tester generated two messages (ID 13 and ID 14) with the SYNC bit set, thus, dictating the clock synchronization in our setting. After about 300 communication cycles we modified the step correction value of the tester by increasing it by $\delta_s = 25$ns. Then we left this value constant for 40 communication cycles and set it back to the original value for the following 40 communication cycles. Afterwards we applied a step correction value increased by additional 25ns (i.e. $\delta_s = 50$ns) for the following 40 communication cycles

[6] http://www.decomsys.com

before we set this value back once more. This procedure was applied periodically resulting in a stimulus as depicted in Fig. 7 (upper plot). At the same time we monitored the cycle length (the number of microticks after clock correction applied to the local node quartz) of the SUT nodes. These values followed the stimulus – as can be seen in Fig. 7 (lower plot) – up to a logical clock state deviation of $\delta_s = 450$ns. Afterwards, they were no longer able to follow suite; the nodes turned silent.

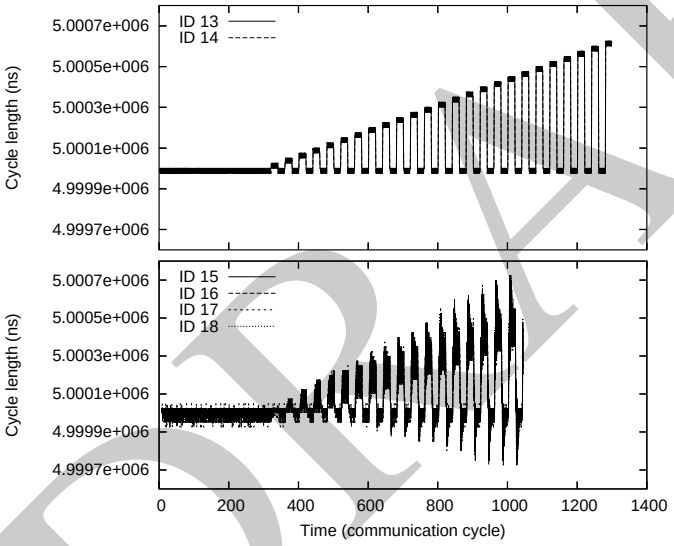

Figure 7. Step Correction Experiment: Stimulus (upper plot) and Response (lower plot)

Fig. 8 provides a detail view of the cycle length one step before the nodes of the SUT turned silent (stimulus + response) along with the SUT nodes' internal rate and offset correction values. It can be observed that the SUT nodes follow the dominating replay node with a delay of two communication cycles. This is due to the observation time required for the nodes to detect a change of the logical time and on the re-synchronization period from the FlexRay algorithm (2 cycles).

Moreover, it seems that the nodes' cycle length might be up to 250ns longer than the replay node's cycle length. This behavior can be explained by the concurrent action of the rate and offset mechanisms. Due to the reaction delay after the step, the SUT nodes' clock state are too late in comparison to the replay node. Additionally the frequency has changed, too. This leads to a punctual double correction, on one side to correct the accumulated delay (due to the frequency step in the past) and on the other side to adjust the rate (for the future).

The amplitude of the logical clock step for Fig. 8 (lower plot) is 425ns. It can be observed, that the nodes' offset correction values grow until 20 microticks (500ns) at communication cycle 1006, which is the maximal offset correction allowed for this configuration.

The values for the offset and rate correction are computed according to the Fault Tolerant Midpoint (FTM) algorithm from the nodes' logical time differences. In our case, the tester node builds a group with its two SYNC frames and the SUT builds another group (tightly synchronized) with four SYNC frames. Since the tester node presents a large shift (time difference) after a step, it will be taken into account for the SUT nodes' clock synchronization. However, due to the FTM computation, the nodes will correct only half the time difference between the tester node and their own clock (the tester node provides the minimal value and the SUT the maximal value – approx. 0 – for the time difference measurement). As a result, the clock state can not catch up the delay accumulated just after the step (cycle 1004) and accumulate even more delay that is corrected afterwards (cycle 1006). This effect leads to an offset correction larger than the step amplitude. It can be then easily interpolated that the nodes will turn into silent with a logical clock step below the configured 500ns (in our case 450ns).

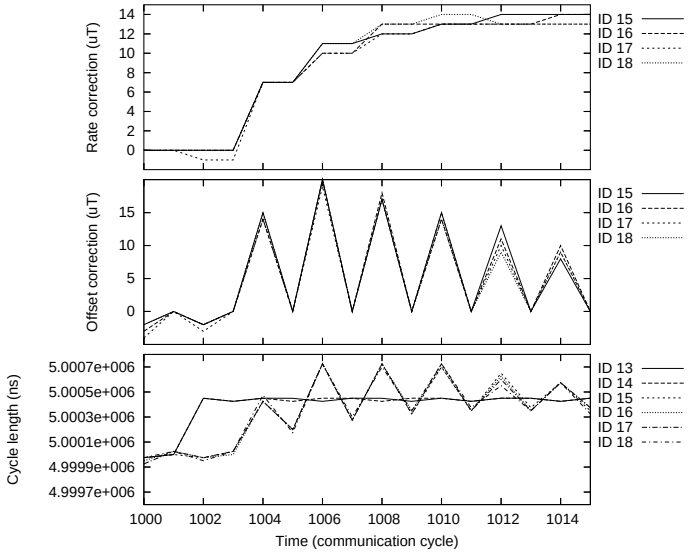

Figure 8. Details of the Step Correction Experiment: Rate Correction (upper plot), Offset Correction (mid plot) and Cycle Length (lower plot)

In a second experiment series, we used the replay mode of the tester to shift the "action point" of a frame – i.e., the point in time when transmission of this frame is started – in a way so that the frame moves close to/beyond its associated slot boundary. The aim was to create a slightly-off-specification[7] failure [21]

[7] An example for a slightly-off-specification failure is when a node sends on

and observe the rate of faulty frames at different points of the communication medium and with different hardware implementations (physical layer chips from different vendors). The acquired results revealed a tolerance in the receiving window so that frames were accepted "as correct" by some nodes and rejected by other ones, see [22] for details. This observation allows an important conclusion: There is definitely a non-zero probability that slightly-off-specification failures may cause Byzantine effects in the network.

In a third test-campaign we injected bit-flips into an ongoing traffic of a COTS cluster. In particular we conducted 15 experiments where each one consisted of more than 3,000 communication cycles and, hence, represented 9 to 15 seconds of bus traffic. For every experiment we injected something between 46 and 1,092 deviations and monitored the traffic with our tester. Whereas a standard COTS controller could classify the injected errors only either as a syntax, content or bit-violation error, our tester revealed detailed information of the cause for the recorded errors (e.g. Transmit Start Sequence error, Byte Start Sequence error, Symbol Window Violation, Cycle counter error,Static Payload Length error, etc.) depending on the actual location and time where the fault was injected.

CONCLUSION

With the adoption of FlexRay for series production, test solutions for the complete life cycle of automotive communication subsystems will gain significance. In particular, tests during various stages of the development process will be required to aid in the implementation and verification process since ever more applications with enhanced functionalities will be introduced. To that end, this paper introduced a systematic approach for structuring the system under test and reducing the test complexity. Next to an implementation some use-cases were presented and underpinned with experimental results that show the applicability of our test-tool.

The presented approach can be re-used for many different purposes, e.g. for debugging and testing during system verification, for inter-operability or robustness tests as well as even for maintenance tests. Future research will be directed towards enhancing the combination of monitoring, replay, and fault-injection with a fault dictionary that will enable a concise fault diagnosis and location.

ACKNOWLEDGMENT

The presented concepts and tools have been developed in the context of the FIT-IT research projects STEACS (807146), ExTracT, and the FHplus project DECS (811414) all managed by the Austrian Research Agency FFG.

its slot boundaries in such a manner that it is accepted as temporal accurate by a subset of nodes whereas it is not accepted by a different subset of nodes, hence, causing asymmetry in the system.

REFERENCES

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

[2] "–", *FlexRay Communications Systems - Protocol Specification Version 2.1 Rev. A.*, FlexRay Consortium, 2005, http://www.flexray.com.

[3] H. Kopetz, *On the Fault Hypothesis for a Safety-Critical Real-Time System*, Workshop on Future Generation Software Architectures in the Automotive Domain, pp.14–23, San Diego, USA, Jan. 2004.

[4] H. Kopetz and R. Nossal, *Temporal Firewalls in Large Distributed Real-Time Systems*, 6^{th} IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems, pp. 310–315, Tunis, Tunisia, Oct. 1997

[5] P.D. Baker and K. Neal, *System and method for general purpose network analysis*, United States Patent No. 6.000.041, Dec. 1999.

[6] K. Bisson and T. Troshynski, *Switched Ethernet testing for avionics applications*, IEEE Aerospace and Electronic Systems Magazine, Vol. 19, Issue 5, pp. 31–35, 2004.

[7] M. Schiffman, *Building Open Source Network Security Tools*, Wiley & Sons, 2002. ISBN-10: 0471205443

[8] P. Peti, R. Obermaisser, W. Elmenreich, and T. Losert, *An architecture supporting monitoring and configuration in real-time smart transducer networks*, Proc. of IEEE Sensors, pp. 1479–1484, 2002.

[9] M.S. Reorda and M. Violante, *On-line analysis and perturbation of CAN networks*, 19^{th} IEEE Int. Symposium on Defect and Fault Tolerance in VLSI Systems, pp. 424–432, Cannes, France, Oct. 2004.

[10] M.C. Hsueh, T.K. Tsai, and R.K. Iyer, *Fault Injection Techniques and Tools*, IEEE Transactions on Computer, Vol. 30, No. 4, pp. 75-82, 1997.

[11] D. Geels, G. Altekar, S. Shenker, and I. Sotica, *Replay Debugging for Distributed Applications*, Proc. of the USENIX Annual Technical Conference, pp. 289–300, 2006.

[12] D. Sundmark, H. Thane, J. Huselius, A. Pettersson, R. Mellander, I Reiyer, and M, Kallvi, *Replay Debugging of Complex Real-Time Systems: Experiences from Two Industrial Case Studies*, Proc. of the 5^{th} International Workshop on Automated Debugging, Ghent, Belgium, Sep. 2003.

[13] E. Armengaud, A. Steininger, M. Horauer, and R. Pallierer, *A Layer Model for the Systematic Test of Time-Triggered Automotive Communication Systems*, 5^{th} International Workshop on Factory Communication Systems, pp. 275-283, Vienna - Austria, Sept. 22 - 24, 2004.

[14] E. Armengaud and A. Steininger, *Pushing the Limits of Re-*

mote Online Diagnosis in FlexRay Networks, 6^{th} International Workshop on Factory Communication Systems, pp. 45–54, Torino, Italy, June 28-30, 2006.

[15] E. Armengaud, *A Remote and Transparent Approach for the Test and Diagnosis of Automotive Networks*, Junior Scientist Conference, pp. 11-12, Vienna - Austria, April 19-21, 2006.

[16] E. Armengaud, A. Steininger and M. Horauer, *A Method for Bit Level Test and Diagnosis of Communication Services*, 8^{th} Int. Workshop on Design and Diagnostics of Electronic Circuits and Systems, Sopron - Hungary, pp. 69-74, April 2005.

[17] E. Armengaud, A. Steininger and M. Horauer, *A Flexible Hardware Architecture for Fast Access on Large Non-Volatile Memories*, 8^{th} Int. Workshop on Design and Diagnostics of Electronic Circuits and Systems, Sopron - Hungary, pp. 113-120, April 2005.

[18] M. Horauer, F. Rothensteiner, M. Zauner, E.Armengaud, A. Steininger, H. Friedl and R. Pallierer, *An FPGA based SoC Design for Testing Embedded Automotive Communication Systems employing the FlexRay Protocol*, Austrochip 2004, pp. 119-123, Villach - Austria, October 2004. ISBN: 3-200-00211-5

[19] E. Armengaud, A. Steininger and M. Horauer, *Automatic Parameter Identfication in FlexRay based Automotive Communication Networks*, 11^{th} IEEE International Conference on Emerging Technologies and Factory Automation, Prague - Czech Republic, pp. 897-904, Sept. 2006.

[20] E. Armengaud, *Low Level Bus Traffic Replay for the Test of Time-Triggered Communication Systems*, 9^{th} IEEE Workshop on Design & Diagnostics of Electronic Circuits & Systems (DDECS'06), pp. 55–156, Czech Republic, Apr. 2006.

[21] A. Ademaj. *Slightly-Off-Specification Failures in the Time-Triggered Architecture*, 7^{th} Annual IEEE International Workshop on High Level Design Validation and Test, pp. 7–12, Cannes - France, Oct. 2002.

[22] E. Armengaud, A. Steininger and M. Horauer, *Efficient Stimulus Generation for Remote Testing of Distributed Systems - The FlexRay Example*, 10^{th} IEEE International Conference on Emerging Technologies and Factory Automation, Catania - Italy, pp. 763-770, Sept. 2005, ISBN: 0-7803-9402-X.