# A Hardware-in-the-Loop Testing Platform Based on a Common Off-The-Shelf Non-Real-Time Simulation PC

Daniel Ulmer*, Steffen Wittel†, Karsten Hünlich† and Wolfgang Rosenstiel‡
*IT-Designers GmbH, Esslingen, Germany
Email: daniel.ulmer@it-designers.de
†Distributed Systems Engineering GmbH, Esslingen, Germany
Email: {steffen.wittel,karsten.huenlich}@distributed-systems.de
‡University of Tübingen, Department of Computer Engineering, Tübingen, Germany
Email: rosenstiel@informatik.uni-tuebingen.de

***Abstract*—The rapidly growing amount of software in embedded real-time applications such as driver assistance functions in cars leads to an increasing workload in the field of software testing. An important issue is thereby the timing behavior of the software running on the target hardware. For testing this issue, real-time capable Hardware-in-the-Loop platforms are needed. These testing platforms are mostly custom-made, proprietary and in consequence expensive. Moreover, many software developers usually have to share few instances. This paper shows an approach for a real-time capable Hardware-in-the-Loop platform based on a common off-the-shelf PC running a non-real-time operating system. Thereby, the simulation software runs on the developer's desktop computer while an extended I/O interface ensures the real-time communication with the System Under Test even for complex timing requirements as shown in an example.**

***Keywords*-Testing; Hardware-in-the-Loop; Embedded Real-Time Systems; Temporal Behavior;**

## I. Introduction

Software development for embedded real-time systems, in particular closed-loop control applications in the automotive industry running on Electronic Control Units (ECUs), requires a reliable testing of the timing behavior on the target hardware. Highly frequent hardware-software integration tests of the software module under development are required, especially if the software development is done in an agile or rapid prototyping manner. These tests are normally executed on a Hardware-in-the-Loop (HiL) testing platform.

The established HiL testing platforms are usually complex devices based on proprietary hardware and software, which makes the platforms very expensive. Often these testing platforms are based on standard PC hardware in combination with an Real-Time Operating System (RTOS) and therefore operated by separate tool chains. Since these testing platforms are very complex and hence expensive, they are usually shared by several developers and are located in separate laboratories instead of being close to the developers' desk, which inhibits the rapid prototyping development cycle.

The approach introduced in this paper uses an extended, real-time capable I/O interface denominated as Real Time Adapter (RTA) designed for the usage with a non-real-time desktop computer directly at the developers' desk. The PC is used to perform the simulation models and to define the expected timing behavior while the I/O interface is responsible for keeping and observing the timing towards the System Under Test (SUT). Unlike most commercial HiL testing platforms, this approach allows to specify an arbitrary timing behavior concerning the communication to the embedded SUT. Furthermore, the approach enables the engineer to use the same software tools for function development or unit testing as well as for testing on the target hardware.

ECUs for driver assistance functions are often connected via bus interfaces to their surrounding ECUs and can therefore be stimulated by supporting the corresponding bus interface. Even ECUs communicating via analogue or digital I/O ports with their environment are mostly capable of separating their application function from the I/O interfaces by stimulating the application functions via a common communication bus. Hence a HiL platform for functional testing on the target hardware is possible for this case.

Conducted experiments and the results obtained in an industrial setting addressing the tests of embedded systems connected via the industry standard Controller Area Network (CAN) [1] show that the combination of a real-time I/O interface and standard desktop hardware are as effective as established HiL testing platforms–but in a much more efficient way–enabling a much higher test frequency.

The following two sections of this paper give an insight into the testing of interconnected ECUs and the operating principles of the RTA as an intelligent I/O device. In Section 4, a comparison relating to timing issues is done between current HiL platforms and the introduced approach based on the RTA. Section 5 finally shows an example for a test setup used in the automotive industry.

## II. Test of interconnected ECUs

Significant parts of vehicle functions, especially modern driver assistance functions, are realized with the help of

 ISBN:978-1-61208-114-4 

software. Commonly, several ECUs and their respective software contribute to implement a vehicle function that can be experienced by the driver [2].

The distribution of software in different ECUs of the vehicle requires that the ECUs are able to communicate with each other. A common widely accepted approach for interconnecting the ECUs is by sending messages on a bus system such as CAN. In order to obtain a deterministic timing behavior the majority of the messages are sent in a cyclic manner with a pre-defined cycle time as shown in Figure 1. ECU1 periodically sends its calculation results to ECU2 and vice versa. Especially for closed-loop control vehicle functions–such as an Adaptive Cruise Control (ACC) [3]–it is important to meet the given timing requirements. The ECUs usually monitor the compliance with the pre-defined cycle strictly, because a violation can result in failure, which might be life-threatening to the passengers of the vehicle.

Since the CAN bus itself is not deterministic [4], the ECU is responsible for the correct communication timing. Additionally, the priority of a CAN message is depending on its message ID. The precision of the bus timing of a certain message is hence depending on the precision of the ECUs' RTOS and on the predefined message ID. Both, the ECU and the CAN bus contribute to a deviation of the intended cycle time that can be measured on the bus. If a message is supposed to be sent with a cycle time of 20 ms, the cycle time on the bus will be not exactly 20 ms. The ECUs will tolerate such an inaccuracy as long the deviation is below a specified limit.

However, modern driver assistance functions narrow progressively the tolerance band of the allowed timing faults while the CAN bus is populated by more and more ECUs with increasing bandwidth requirements that exacerbate the situation. Seen from a testing perspective, it is thus essential that the reaction on corrupted bus timing is tested. This implies that the testing device itself is able to meet the timing requirements in the first instance and moreover to manipulate it arbitrarily.

The first integration step in the development cycle, where the timing behavior of an ECU's CAN interface can be tested, is the execution of the developed software on the target hardware. A common approach is to do this on HiL testing platforms.

Further on, it is useful having the HiL testing platform close to the software-developers' desk, especially if the ECU software is developed in an agile manner with frequent integration steps that require frequent testing on the target hardware. Commonly, different software parts for driver assistance functions are coded and tested by several developers in parallel. If these software parts are integrated into one ECU, the developers have to share the available HiL platforms. Instead of having a HiL testing platform waiting for the developer, the developer often needs to wait for the HiL platform.

## III. Real Time Adapter

The RTA [5] combines the functionality of a mobile data logger and an intelligent I/O device for CAN. Its core functions [6] are implemented in VHDL to increase the execution speed and run them as parallel as possible on the built-in FPGA. In the case of the mobile data logger the CAN messages from the SUT can be stored locally on the device, whereas in the case of the I/O device the messages are transferred to an external PC and vice versa via an Ethernet connection.

As illustrated in Figure 2, the PC can process the provided information and calculate the transmit time of the response based on high precision time stamps added by the RTA to each received CAN message. Hence, a variable processing time on the PC within the tolerance range does not matter. The RTA takes care about the correct sending points of the CAN messages as well as detects timing violations caused by messages with time stamps that cannot be transmitted in time. It decouples the non-real-time behavior of the PC from the precise real-time behavior towards the SUT, which even allows the performing of complex test cases with the exact timing at each test run. The timing of each message is thereby treated separately by the RTA and thus the transmit time can be simply manipulated during a test run. Especially, this characteristic is important for test cases that validate the correctness of the SUT communication timing.

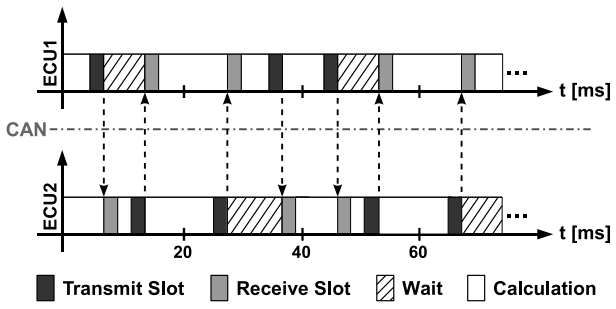


Figure 1. Cyclic communication of ECUs

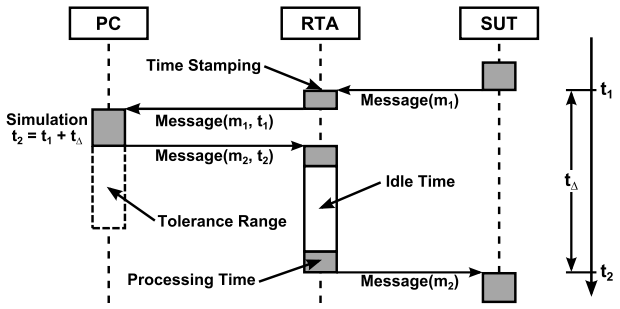


Figure 2. Sequence diagram of CAN RX/TX with an RTA

## IV. HiL Platforms

In the following current HiL approaches are discussed with a closer look on their timing behavior. Additionally, the new approach that addresses the requirements for an agile usage as well as for the timing issues is introduced.

### A. HiL Platform Based on an RTOS

Current HiL platforms, as they are introduced in [7], usually focus on ECU testing from a functional and non-functional perspective. This means that the testing platform covers the testing of the reaction to electrical errors as well as the test of the functions required by a driver assistance function. The approach of testing the whole test plan at only one testing platform makes this platform very complex from the hardware as well as from the software point of view. Although current solutions, as proposed by ETAS [8], are based on off-the-shelf computer hardware, they have to be expanded by several special software and hardware components needed to achieve the required functionality. One important software component is the Residual Bus Simulation (RBS), which is responsible for imitating the environment around the ECU seen from a communication point of view. If the SUT is connected via CAN buses to its surrounding components, the RBS needs to ensure the same communication behavior as established by the real environment of the SUT. For guaranteeing the real-time behavior of the CAN communication, an RTOS is used to implement the RBS for the CAN bus and the additionally required software components such as environment models. If the schedule of the RTOS is set up correctly, a precise execution of the desired CAN schedule is guaranteed within the tolerance of the RTOS.

Figure 3 shows the measured time between two CAN messages with the same message ID during a HiL test at a platform based on an RTOS [9][10][11]. According to the CAN schedule, the message is supposed to have a 20 ms cycle. The plot displays that this implementation achieves an average cycle time of almost 20 ms with a standard deviation of about 200 µs. Single outliers are reaching up to a period of 20.8 ms between two consecutive messages. In this example the measured timing still fulfils the SUT requirements.

### B. HiL Platform for Functional Testing

For testing the functional behavior of different software modules running on the same ECU, it is helpful to have several testing platforms close to the developers' desks. Of course, for testing the ECUs reaction to electrical errors it is still necessary to use the complex platforms introduced before. For a quick test of a change in a hardware independent software module, HiL platforms based on a Common Off-The-Shelf (COTS) computer can be built that are connected via a CAN interface to the SUT. In this context using COTS components not only refers to hardware but additionally to software including a non-real-time Operating System (OS), typically Microsoft Windows. Additionally within the context of large companies, the IT support determines the use of virus scanners and other tools, if the PC interconnects with the corporate network. Using a standard computer means that it might also be a laptop. In this case it is easily possible to use the HiL setup within a test vehicle or while being at a field trial. Another advantage of using the standard desktop OS is that the already existing tool chain can be used to set up the HiL platform. Especially, the libraries of environment models for Model-in-the-Loop (MiL) and Software-in-the-Loop (SiL) simulations from earlier integration steps of the driver assistance function can be reused without the need of being ported to an RTOS environment.

Figure 4 shows the time between two consecutive CAN messages of the same message ID during a HiL test on such a platform without an RTOS. The plot displays that the implementation based on a COTS computer and a CAN interface achieves an average cycle time of 20 ms with

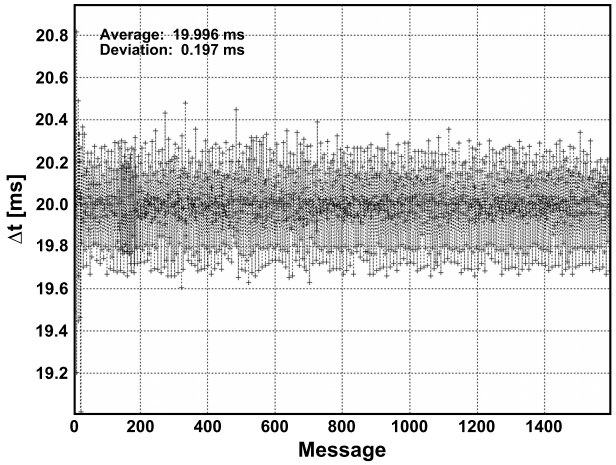


Figure 3. RTOS HiL – 20 ms cycle time message

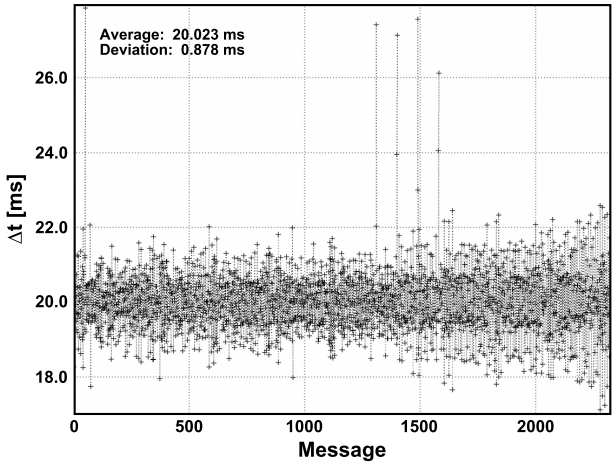


Figure 4. Non-real-time OS HiL – 20 ms cycle time message

a more than fourfold standard deviation of approximately 900 µs. Getting worse, in this case outliers of up to 8 ms can be seen. This approach only works, if the ECU tolerates such outliers.

A major drawback of this approach is that the environment models and the RBS have to be either implemented on the OS of the COTS computer or at least the RBS has to be shifted to the CAN interface. In the first case, the timing behavior of the RBS is depending on the timing behavior of the non-real-time OS. In the latter case, a separate tool chain is necessary to implement the RBS on the CAN interface. This leads to a fix communication schedule, which can be only manipulated at runtime if a complex handshake between the PC and the RBS is set up. If the implementation of the timing supervising software within the ECU is not too strict, the first approach works in practical use.

### C. HiL Based on a COTS Computer and an RTA

Since the timing requirements of the ECUs tighten and the implemented driver assistance functions require more and more precise data at an accurate point in time, the timing behavior shown in Figure 4 is not acceptable anymore. Additionally, if the implemented function, e.g., for interacting vehicles [12], is not only depending on the data value but also on its arrival time, the targeted testing of the reaction on certain bus timing becomes necessary. A HiL platform, which solves the timing issues while leaving the RBS on the COTS computer (PC), is introduced in [5] and [6]. The approach leaves it up to the PC to define the intended sending time of a message. This time stamp is then handed over together with the payload to the RTA. While the computer is responsible for calculating timing and content, the RTA precisely plans, executes and supervises the desired timing. If for any reason the desired timing cannot be kept within a certain tolerance, the RTA informs the simulation software on the computer. It is then up to the simulation application to repeat the test case. Thereby, an upper limit prevents the HiL testing platform from repeating the same test case too often.

Timing violations usually originates from the non-real-time OS on the PC in combination with the time consuming or concurrent execution of programs during a test run, e.g., anti-virus scanners, mail software or automatic update clients. The test implementation itself and the resources consumption associated with it also affect the timing. Test cases, which need more processing time on the PC for one simulation step as the expected cycle time of the SUT, are not suitable to be executed on this platform.

Figure 5 shows the result of the introduced solution for a current ECU with driver assistance functions. The intended cycle time of 20 ms is kept with a standard deviation of 5 µs. Even the outliers, which occur in this case due to the occupied bus, are less than 40 µs.

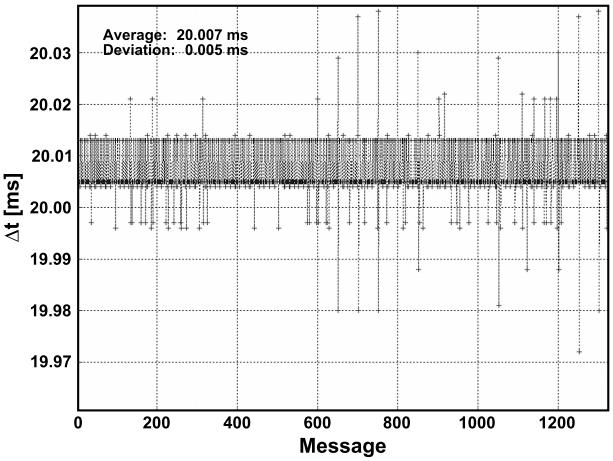


Figure 5. RTA HiL – 20 ms cycle message

The performance of the HiL testing platform primarily depends on the COTS hardware used to set up the platform, which determines the test case limitations in terms of timing. Practical experiences with a prototypical implementation show that approximately one of 1000 test cases has to be repeated. Moreover, measurements during the evaluation revealed an average pass through time of about 4 ms to receive a CAN message from the SUT and send the response back. The time also includes the calculation of a common test step within the simulation on the PC. In the example this means that the HiL testing platform has roughly 16 ms at a cycle time of 20 ms to compensate outliers occurred during the performing of a test case. Based on these obtained results the outliers are not an exclusion criterion for the use in a production environment, because they are detected and reported by the RTA.

## V. Example

An example for a test setup used in the automotive industry is displayed in Figure 6, which comprises of a standard PC with an RTA as well as of the SUT itself consisting of two CPUs that are connected to the same clock oscillator. Thereby, the PC and the RTA are used to simulate the ECU's environment. For safety critical reasons, some applications within the ECU are tested on module level embedded into the final hardware. The *Communication CPU* has two tasks with 20 ms cycle time. On the one hand, it

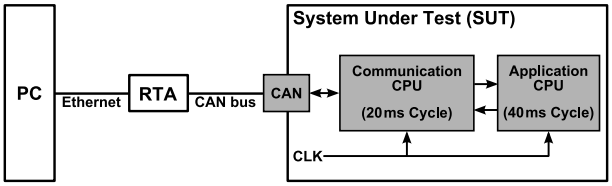


Figure 6. Example for a test setup

implements the bus communication that consists of receiving and transmitting messages and updating the internal signal database. On the other hand, this CPU is used to validate the results of critical functions running on the *Application CPU*. The *Application CPU* runs at 40 ms cycle time and is responsible for processing the implemented driver assistance functions.

One software module running on the *Application CPU* implements a safety critical requirement. In this example we assume that a sensor sends a signal denominated *Object Type*. This signal is specified to be zero for two CAN cycles and four for the following three CAN cycles, if the sensor is faulty. The safety critical requirement of the software module is to detect this situation and to prevent a driver assistance function from interfering. The correct implementation of the software module is to be tested on the target hardware and hence at a HiL platform. Since the clocks of the SUT and the HiL platform are independent, it cannot be guaranteed that the sequence is received correctly at the SUT's internal data interface. To achieve reproducible test results, it is necessary to synchronize the testing platform with the SUT. The synchronization mechanism is shown in Figure 7. Some *Application Results* are handed over from the *Application CPU* to the *Communication CPU*, which transmits the corresponding CAN message on the CAN bus. The RTA delivers this message together with a receive time stamp to the PC running the environment simulation. After calculating the simulation environment model, the result is handed over to the RTA for being sent 41 ms ahead in time. This ensures that the result is available for the *Application CPU* right before a new application cycle begins.

Listing 1 shows a pseudo code sample for an implementation on a PC based platform for functional testing. Since the sending time of the message is in this case depending on the scheduling of the *TransmitThread* of the non-real-time OS, it cannot be guaranteed that the sequence is sent as specified.

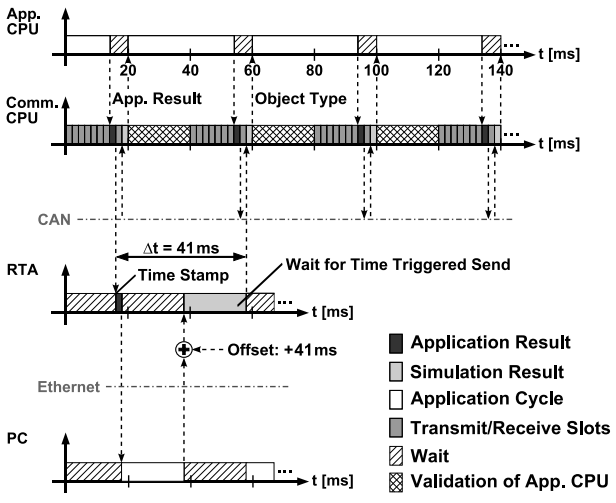


Figure 7. Synchronization of the testing platform with the SUT

```
WHILE(NOT quit)
BEGIN

  // Receive CAN Message
  Receive(in_message)

  // Calculate Environment Simulation Model
  out_message = CalcEnvModel(in_message)

  // Calculate Output Message Time Stamp
  time_stamp = in_message.time_stamp + 41

  // Transmit CAN Message using the Windows
  // Event Timer in a separate Thread
  TransmitThread(out_message, time_stamp)

  // Wait until next Cycle
  WaitForNextWindowsTimeEvent()

END
```

Listing 1. Standard PC synchronization mechanism

Listing 2 illustrates that in case of an RTA based HiL testing platform the precise sending of the message is done by the RTA and therefore independently of the OS timing deviations. In the worst case, a message is sent too late to the RTA and the test case is then being declared invalid and repeated.

```
WHILE(NOT quit)
BEGIN

  // Receive CAN Message
  RTA_ReceiveMessage(in_message)

  // Calculate Environment Simulation Model
  out_message = CalcEnvModel(in_message)

  // Calculate Output Message Time Stamp
  time_stamp = in_message.time_stamp + 41

  // Transmit CAN Message to RTA
  RTA_TTS_TransmitMessage(out_message, time_stamp)

  // Wait until next Cycle
  RTA_WaitForNextCycle()

END
```

Listing 2. RTA synchronization mechanism

Figure 8 shows the results achieved on the CAN bus with a bus load of 60 % and a cycle time of 20 ms for the CAN messages. The sequence of two cycles zero and three cycles four is precisely executed. In the project context we have implemented this testing challenge on the RTA based HiL platform since this platform is available at every developers' desk and the modification of the existing simulation code has been limited to adding a constant offset to the time stamp of an incoming message. We have decided against an implementation on an RTOS HiL since there is only one instance available, which can either be used for implementing new features or for running tests. Synchronizing the time

 ISBN:978-1-61208-114-4 

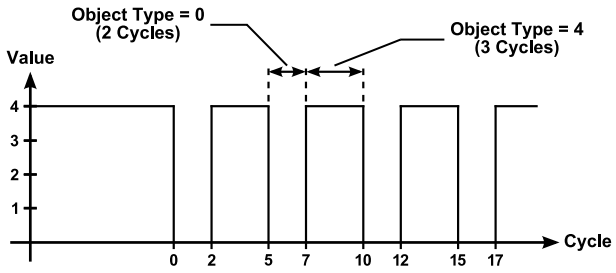


Figure 8. CAN trace for cyclic stimulation

slice based RTOS to the SUT would have meant to change the complete simulation kernel and therefore several days of implementation work.

## VI. Conclusion and Future Work

The measurements demonstrate that it is possible to implement a HiL testing platform fulfilling the timing requirements of modern driver assistance functions and the requirements of an agile or rapid prototyping development process within the automotive industry. It has also been shown that current testing platforms address one of these aspects while the RTA approach addresses both. It has also been argued that the achieved timing on the CAN bus of the RTA based HiL platform is more precise than the timing of the RTOS HiL. It is left for future work to study the advantages of the RTA approach in terms of the definition and flexible manipulation of the timing behavior, e.g., for deterministic robustness tests of the function software. One aspect might be the modeling of a statistic temporal distribution where the parameters can be influenced by random testing or by evolutionary testing. Additionally, the RTA approach might be used as a cost efficient HiL setup for a continuous integration tool chain for embedded software development. Due to the usually large number of variants on the level of hardware-software integration, a high test volume must be considered here. However, each test can be executed at maximum in real-time for each variant. This means that for quick results many parallel HiL platforms are necessary. The price efficient HiL testing platform based on the RTA is a necessary step to implement this idea.

## References

[1] ISO, *ISO 11898-1:2003: Road vehicles — Controller area network (CAN) — Part 1: Data link layer and physical signalling*. International Organization for Standardization, 1993.

[2] C. Marscholik and P. Subke, *Road vehicles - Diagnostic communication: Technology and Applications*. Hüthig, 2008.

[3] Daimler AG, “The challenge of accident prevention”, *Milestones in Vehicle Safety. The Vision of Accident-free Driving*, 2009.

[4] K. Etschberger, *Controller Area Network. Basics, Protocols, Chips and Applications.* IXXAT Automation, 2001.

[5] IT-Designers GmbH, “RZA”, Access Date: November 10, 2010. [Online]. Available: http://www.it-designers.de/RTA

[6] D. Ulmer, A. Theissler, and K. Hünlich, “PC-Based Measuring and Test System for High-Precision Recording and In-The-Loop-Simulation of Driver Assistance Functions”, in *Proceedings of the Embedded World Conference*, 2010.

[7] C. Marscholik and P. Subke, *Datenkommunikation im Automobil*. Hüthig, 2007.

[8] ETAS GmbH, “LABCAR System Components - ETAS Products”, Access Date: November 10, 2010. [Online]. Available: http://www.etas.com/en/products/labcar_system_components.php

[9] ETAS GmbH, “LABCAR-RTPC Real-Time Simulation Target for HiL Testing”, Access Date: November 10, 2010. [Online]. Available: http://www.etas.com/en/products/labcar_rtpc.php

[10] G. Wittler and J. Crepin, “Real-time and Performance Aspects of Hardware-in-the-Loop (HiL) Testing Systems”, *ATZonline*, 2007.

[11] J. Kiszka, “Xenomai: The RTOS Chameleon for Linux”, Real-Time Systems Group, Leibniz Universität Hannover, Tech. Rep., 2007.

[12] D. Ulmer and A. Theissler, “Application of the V-Model for the development of interacting vehicles and resulting requirements for an adequate testing platform”, in *Proceedings of the Software and Systems Quality Conferences*, 2009.

 ISBN:978-1-61208-114-4 