Testing Platform for Hardware-in-the-Loop and In-Vehicle Testing Based on a Common Off-The-Shelf Non-Real-Time 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. The timing behavior of the Driver Assistance Functions is usually tested on real-time capable Hardware-in-the-Loop platforms as well as by in-vehicle tests, where the timing behavior is evaluated with the help of data loggers. Both, the data loggers and the Hardware-in-the-Loop platforms are mostly custom-made, proprietary and in consequence expensive. Moreover, many software developers usually have to share few instances. Existing inexpensive solutions show deficits in their real-time capabilities, which means for Hardware-in-the-Loop platforms that the real-time behavior cannot be guaranteed and for data loggers that they do not provide a common time base for relating data from different vehicles involved in a maneuver. 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 and an extended I/O interface, which can be used for in-vehicle tests as well. Thereby, the simulation software runs on the developer's desktop computer while the extended I/O interface provides a global time base and ensures the real-time communication with the System Under Test even for complex timing requirements. Two examples show how the introduced setup can be used to test Driver Assistance Functions on a Hardware-in-the-Loop platform and as a data logger for in-vehicle tests. Questions such as "How much time is needed by the Adaptive Cruise Control System to determine the relative speed of the preceding vehicle?" can be answered.**

Keywords-Hardware-in-the-Loop Testing; In-Vehicle Testing; Embedded Real-Time Systems; Temporal Behavior; Driver Assistance Systems.

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 testing of the timing behavior on the target hardware as presented in [1]. 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 these platforms very expensive. Often, these testing platforms are based on standard PC hardware in combination with a 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' desks, which inhibits the rapid prototyping development cycle.

The approach introduced in this paper uses a special, 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 developer's 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 interfaces. Even ECUs communicating via analog 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 suitable 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) [2] 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 more efficient way–enabling a higher test frequency.

Testing Driver Assistance Functions in vehicles adds another challenge to the test equipment. Having now the ECU in the vehicle means that a surrounding vehicle on which the Driver Assistance Function reacts has to be present. Furthermore, the surrounding vehicle has to be considered for evaluating the test result. The Driver Assistance Function Adaptive Cruise Control (ACC) [3], e.g., is a closed-loop control for the vehicle's speed considering the speed of the preceding vehicle. Common ACC systems have a radar sensor, which determines the relative speed and distance of the preceding vehicle. In some ACC systems, the driver can set the desired distance to the preceding vehicle and the ACC system tries to keep this distance by accelerating or decelerating the vehicle. If such a system is to be tested on the road it is necessary to be able to correlate the information about the preceding vehicle with the information of the vehicle with the ACC system. Issues to be tested might be:

- How much time elapsed between pressing the brake pedal in the preceding vehicle and a deceleration demand of the ACC system?
- How much time is needed by the radar device to determine the relative speed of the two vehicles?
- What is the difference between ACC algorithms on the behavior of the vehicle?
- How does an ACC algorithm perform compared to a human driver?

All questions have in common that having data from the vehicle with the ACC system is not enough. It is necessary to know the state of the preceding vehicle and to be able to correlate this information. Independent data loggers in the vehicles do not allow to record data on a common time base, which means that data cannot be correlated in time. It is therefore necessary to synchronize the data loggers in the different vehicles. The introduced RTA is able to synchronize its internal clock to the time provided by the Global Positioning System (GPS). Since the RTA is not only able to record data with a highly precise GPS-based time stamp but is also able to use the GPS-time to send data, it is possible for applications to replay data recorded in a vehicle with precisely the same timing behavior on a test platform in the laboratory.

Section II of this paper gives an insight into the testing of interconnected ECUs followed by a section that compares current HiL testing platforms relating to timing issues. In Section IV, the operating principles of the RTA as intelligent I/O device are introduced as well as an approach for a HiL testing platform based on the RTA. Finally, Section V shows examples for a HiL test setup and an in-vehicle test of an ACC system, which show the current usage of the RTA 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 software. Commonly, several ECUs and their respective software contribute to implement a vehicle function that can be experienced by the driver [4].

The distribution of software on 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 ACC–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 [5], 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 seen cycle time of this message on the bus will differ from the desired 20 ms. The ECUs will tolerate such an inaccuracy as long as the deviation is below a specified limit.

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. Usually, the ECU for Driver Assistance Functions monitors if the timing of the received messages is within the predefined limits. In some cases, the data content is additionally monitored on its in-time arrival as explained in Section V-B. Considered from a testing perspective, it is thus

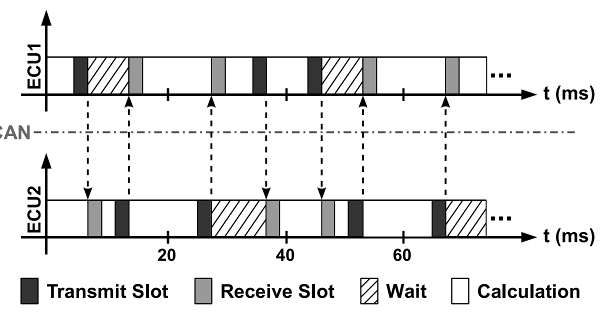

Figure 1. Cyclic communication of ECUs

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 ECU software on the target hardware. A common approach is to do this integration testing on HiL testing platforms.

Further on, it is useful having the HiL testing platform close to the software-developers' desks, 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 the ECU software, 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.

After having tested the combination of ECU software and hardware on a HiL platform, the ECU can be integrated into a vehicle to see the Driver Assistance Function work in its target environment. For Driver Assistance Functions such as an ACC, the target environment is not limited to the vehicle but the vehicle and its surrounding vehicles that the ACC reacts on have to be considered [6], too. The same also applies for the test of a Forward Collision Warning System (FCWS). The National Highway Traffic Safety Administration (NHTSA) requests for testing a FCWS in [7] that the instance in time when the driver has been warned can be correlated to the position of both vehicles involved in the test. This means that the measured data of both vehicles have to be correlated.

III. Current Testing Platforms

In the following, three different HiL approaches currently used in the automotive industry are discussed with a closer look on their timing behavior.

A. *HiL Platform Based on an RTOS*

Current HiL platforms, as they are introduced in [8], usually focus on ECU testing from both, a functional and a 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 [9], 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. To guarantee the real-time behavior of the CAN communication, an RTOS is used to implement the RBS for the CAN bus and the additional 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.

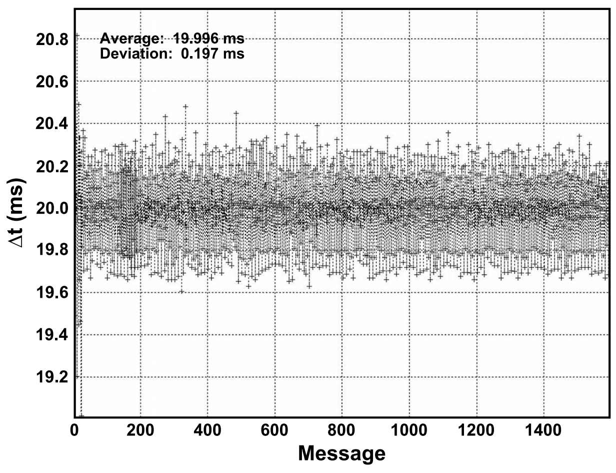

Figure 2. RTOS HiL – 20 ms cycle time message

Figure 2 shows the measured time between two CAN messages with the same message ID during a HiL test at a platform based on an RTOS [10][11][12]. According to the CAN schedule, the message is supposed to have a 20 ms cycle time. The plot displays that this implementation achieves an average cycle time of almost 20 ms with a standard deviation of about 197 µs. Single outliers are reaching up to a period of 20.826 ms between two consecutive messages. In this example, the measured timing still fulfills 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. In this case, the COTS computer can be 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

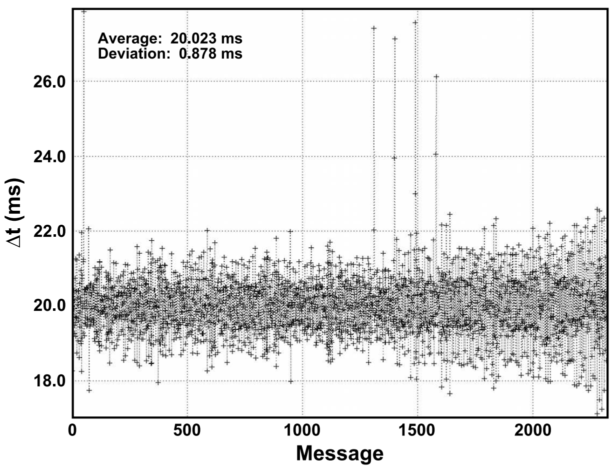

Figure 3. Non-real-time OS HiL – 20 ms cycle time message

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 3 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 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 development tool chain is necessary to implement the RBS on the CAN interface. This leads to a fixed 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. Data Logger for In-Vehicle Testing

Testing a Driver Assistance Function in the vehicle is usually done with the help of a data logger. The data logger collects and stores the data, which is transferred between ECUs within the vehicle. The test result is based on the collected data. Common data loggers like [13] are able to record several different buses within one vehicle on a common time base. A time synchronization for data loggers of several vehicles is mostly custom made and thus expensive. For evaluating Driver Assistance Functions such as an ACC, this feature is important. Since the interaction between vehicles raise questions such as "How much time is needed by the Adaptive Cruise Control System to determine the relative speed of the preceding vehicle?".

There are several ways like RFC 958 [14] or IEEE 1588 [15] to synchronize independent clocks. As mentioned by Luther [16] IEEE 1588, also referred to as Precision Time Protocol, reaches a temporal synchronicity among vehicles with less than five milliseconds deviation. This turned out to be sufficient for the verification of vehicle functions with a velocity of up to 20 m/s. For synchronization at least at the beginning of each measurement, a connection between the devices is required. A single adjustment of the clocks may lead to inaccuracies of the time stamps at longer test runs, whereas a continuous adjustment with IEEE1588 can be difficult to be implemented especially for moving vehicles without the existence of a direct wired connection. Thereby, the clocks are usually synchronized amongst themselves and not to a global time like provided by the Global Positioning System (GPS) [17].

IV. Testing Platform optimized for Driver Assistance Systems

In this section, the RTA and the approach for a HiL testing platform based on the RTA device that addresses the requirements for an agile usage as well as for the timing issues are introduced.

A. The Real Time Adapter

The RTA [18] combines the functionality of a mobile data logger and an intelligent I/O device for CAN. Its core functions [19] are implemented in VHDL to increase the execution speed and run them as parallel as possible on the built-in FPGA. Additionally the RTA's time base, which is implemented in VHDL as well, can be synchronized to the time provided by a GPS receiver with a deviation of less than 10 µs [18]. 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. In both cases, incoming as well as outgoing CAN messages can be processed with respect to the GPS-time.

As illustrated in Figure 4, 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 it detects timing violations caused by messages with time stamps that cannot be transmitted in time. For the timing violation detection, the RTA compares the intended sending point of each message with the actual transmission time taking account of an adjustable tolerance as described in [20]. The RTA decouples the non-real-time behavior of the PC from the precise real-time behavior towards the SUT, which allows the execution of complex test cases with a repeatable precise timing behavior 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 at its limits.

The use of RTA devices in test environments allows the recording and the replay of CAN messages with high temporal precision. But without time synchronization between RTA devices the time stamps may not be generally comparable, because they refer to built-in oscillators with individual drifts. For the verification of Driver Assistance Functions during road tests, e.g., of an ACC as illustrated in Figure 5, it is necessary to record CAN messages in different vehicles with synchronized time stamps. This allows a comparison of the time stamps within the limits of the clock synchronization accuracy. Having these synchronized CAN traces enable us, e.g., to answer questions such as ”How much time elapsed from pressing the brake pedal in the vehicle in front until the ACC decelerates the following vehicle?”. Another use case with a detailed example is given in Section V.

The synchronization of the RTA devices uses a GPS based approach as shown in Figure 6 that ensures a comparable global time base for the time stamps as well as a continuous adjustment of the internal clocks during the test runs. In addition to the position and time information, GPS receivers provide a high-precision output so-called Pulse Per Second (PPS), which signalizes the start of a second. The time information initially preloads the clock of the RTA, whereas the PPS and the built-in oscillator increment the clock. Assuming that the drift of the RTA’s oscillator does not abruptly change under ordinary circumstances but rather is slowly influenced by, e.g., temperature and / or aging. These assumptions did hold throughout all our experiments in the last four years. The RTA counts the number of oscillator ticks per GPS-second and thus obtains the time step per tick to be used by the clock for the next second. This procedure ensures a highly precise time base. Combined with the RTA’s Time Stamping Unit it is possible to attach a time stamp based on the GPS-time to each CAN message.

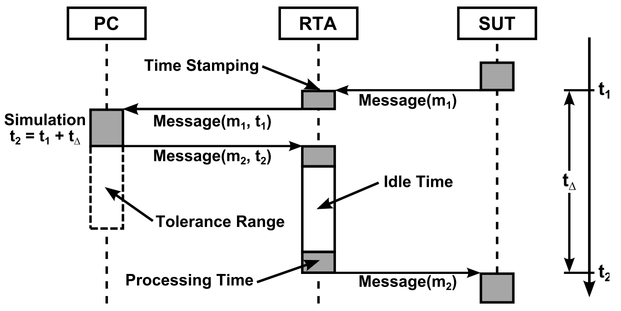

Figure 4. Sequence diagram of CAN RX/TX with an RTA

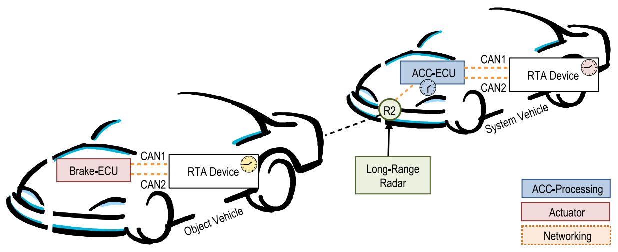

Figure 5. Schematic representation of an ACC test case

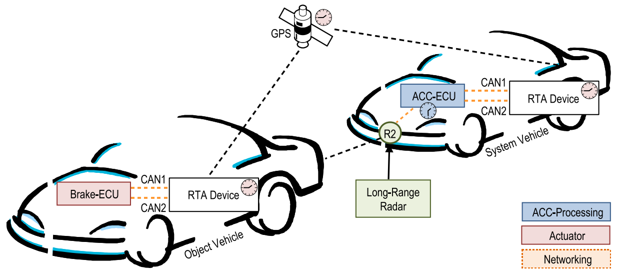

Figure 6. GPS synchronized RTA devices

B. HiL Testing Platform Based on an RTA and a COTS Computer

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 3 is not acceptable anymore. Additionally, if the implemented function, e.g., for interacting vehicles [21], 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 [18] and [19]. 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

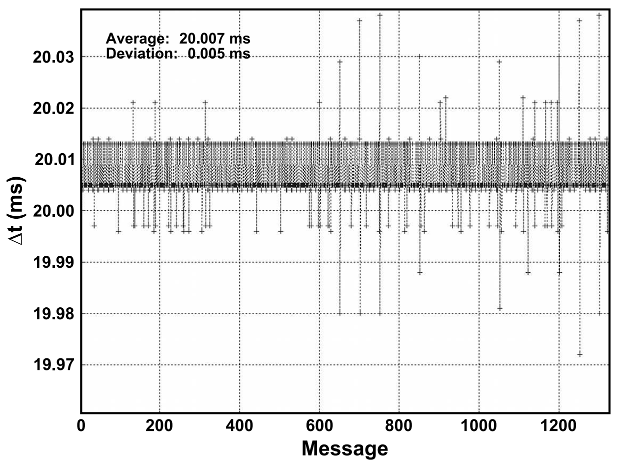

Figure 7. RTA HiL – 20 ms cycle message

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 time consuming or concurrent executions during a test run, e.g., anti-virus scanners, mail software, automatic update clients or mouse movements. 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 7 shows the result of the introduced solution for a current ECU with Driver Assistance Functions. The intended cycle time of 20 ms is kept by the RTA HiL 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.

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. Application of the Introduced Platform in Real World Examples

In the following, an in-vehicle verification of a Driver Assistance System as well as a test of a monitoring algorithm are presented. These examples describe the current usage of the RTA in the automotive industry.

A. In-Vehicle Verification of a Driver Assistance System

In Figure 6, the test setup for an in-vehicle verification of an ACC system is shown. Both the preceding and the following vehicle are equipped with an RTA device. The vehicles drive one behind the other as illustrated in the figure. The goal of this in-vehicle test is to verify the information about the relative speed between the preceding vehicle and the following vehicle delivered by the long-range radar.

Definition (System Vehicle)**.** *The system vehicle is the vehicle equipped with the Driver Assistance System that is to be tested.*

Definition (Object Vehicle)**.** *The object vehicle is the preceding vehicle in the test case that is tracked by the long-range radar of the following vehicle.*

In this test, the system vehicle is the following vehicle that tracks the preceding vehicle with its long-range radar. One RTA records thereby the velocity of the object vehicle, whereas the other RTA records the velocity of the system vehicle as well as the relative velocity between both vehicles determined by the long-range radar. Overall, the following four road tests were performed in which the object vehicle decelerates to a standstill and the system vehicle is stopped to avoid a rear-end collision:

- Manual stop of the system vehicle carried out by a defensive human driver.
- Manual stop of the system vehicle carried out by an aggressive human driver.
- Automatic stop of the system vehicle carried out by the ACC with the desired distance set to the maximum possible value.
- Automatic stop of the system vehicle carried out by the ACC with the desired distance set to the minimum possible value.

After the test runs the velocity information of the object vehicle (v_{obj}) and the system vehicle (v_{sys}) recorded by the two GPS synchronized RTA devices is merged and visualized on a common time axis. Figure 8 shows the velocity curve of the object vehicle and of the system vehicle as well as the difference of the velocities ($v_{rel,calc}$). Furthermore, the relative velocity provided by the long-range radar ($v_{rel,radar}$) is shown as a dotted line of the provided data points. The steep curve of v_{sys} relative to v_{obj} in Figure 8 shows that the defensive driver of the system vehicle responds very quickly and with strong braking to the

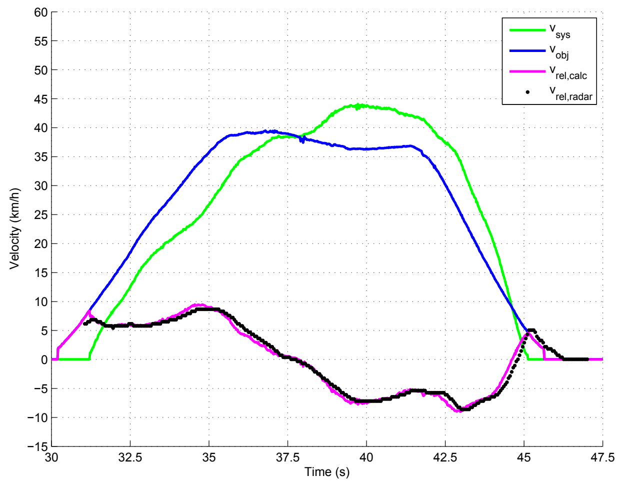

Figure 8. Driver with a defensive braking response

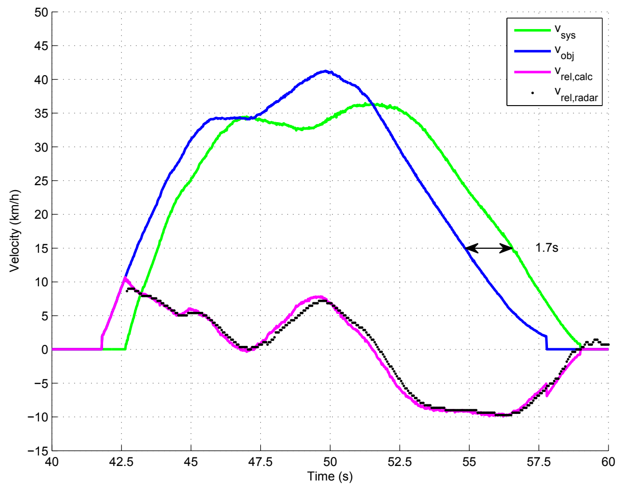

Figure 10. ACC with the desired distance set to the "MAXIMUM"

deceleration of the object vehicle. Thus, the system vehicle stops at 45.13 s while the object vehicle comes to a standstill at 45.65 s.

Figure 9 shows a stopping process of the system vehicle carried out by an aggressive driver. Thereby, the system vehicle stops in the test run just behind the object vehicle and nearly at the same point in time. This is shown in the graph by the fact that the velocity curves of both vehicles reach 0 km/h almost simultaneously. Moreover, it is seen that there are points in time where the long-range radar does not provide information about the relative velocity between the object vehicle and the system vehicle. In the time range between 54.91 s and 56.81 s, the object vehicle could not be correctly recognized.

Figure 10 shows a stopping process of the system vehicle carried out by the ACC with the desired distance set to the maximum possible value. Thereby, the Driver Assistance Function decelerates the system vehicle depending on the object vehicle. The ACC is trying to keep the specified distance. It is apparent that the system vehicle responds in the road test with a delay of about 1.7 s. The nearly parallel course of v_{sys} and v_{obj} indicates that the ACC keeps the driver's distance setting most of the time. In contrast, the defensive driver in Figure 8 has decelerated the system vehicle more than the driver of the object vehicle did, which can be seen by the intersection of the velocity curves at the end of the measurement.

Figure 11 shows a stopping process of the system vehicle carried out by the ACC with the desired distance set to the minimum possible value. It is apparent that the curves of v_{sys} and v_{obj} run parallel to each other with a larger temporal distance of approximately one second compared to the test in Figure 10. Thereby, the larger temporal distance results in a standstill of the system vehicle at a later time.

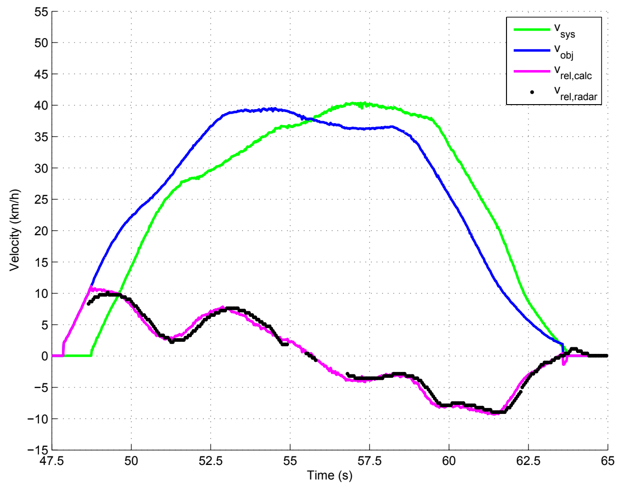

Figure 9. Driver with an aggressive braking response

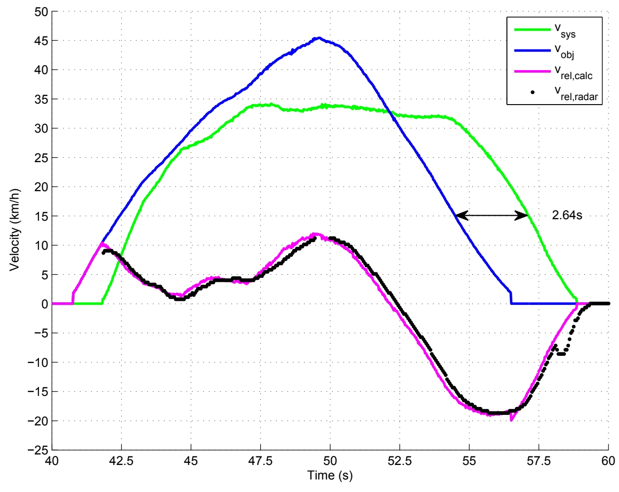

Figure 11. ACC with the desired distance set to the "MINIMUM"

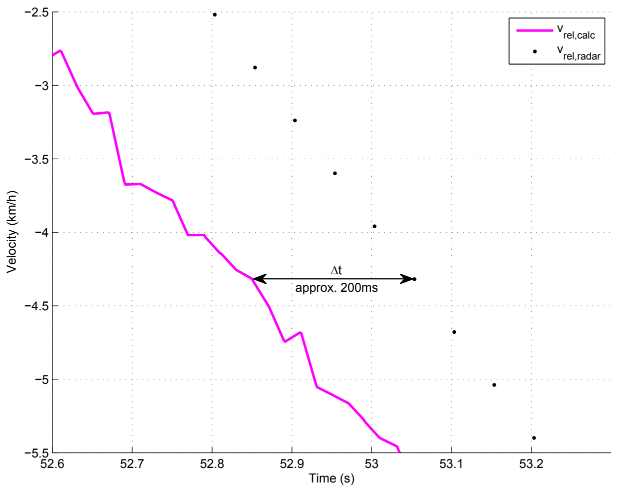

Figure 12. Processing time of the radar ECU

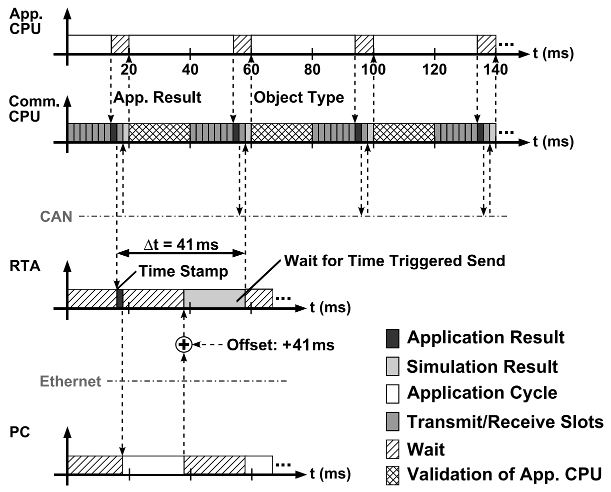

Figure 14. Synchronization of the testing platform with the SUT

Just before standstill $v_{rel,radar}$ is inaccurate. Beginning from 58.10 s, $v_{rel,radar}$ diverges from $v_{rel,calc}$.

In accordance to [22], the radar ECU should provide the information with a maximum delay of 250 ms. To verify this precept, an enlargement of Figure 11 is shown in Figure 12, in which the delay (Δt) between $v_{rel,radar}$ and $v_{rel,calc}$ is marked. Thereby, Δt is calculated according to

$$\Delta t = t_{v_{rel,radar}}(v) - t_{v_{rel,calc}}(v)$$

with $v = -4.32\,km/h$. In this case, Δt is about 200 ms.

B. *Test of a Monitoring Algorithm*

An example for a HiL test setup used in the automotive industry is displayed in Figure 13, 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 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

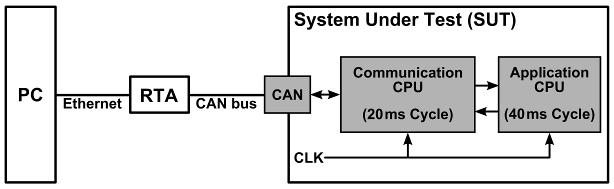

Figure 13. Example for a test setup

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 14. 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 illustrates 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.

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.

Figure 15 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 developer's 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 slice based RTOS to the SUT would have meant to change the complete simulation kernel and therefore several days of implementation work.

```
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

```
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

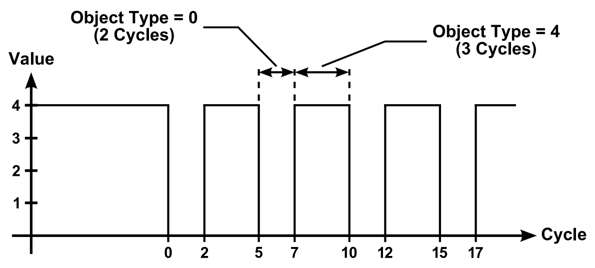

Figure 15. CAN trace for cyclic stimulation

VI. Conclusion and Future Work

The measurements demonstrated 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. For each variant, the tests can be executed at maximum in real-time. 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. An additional benefit of the introduced RTA is that the same hardware supports in-vehicle testing of Driver Assistance Functions. It has been shown that the precise time stamping synchronous to the global GPS-time can be used to correlate the information of several vehicles in one diagram on a common time axis. The reaction of the system vehicle to an action executed by an object vehicle can be evaluated quantitatively. It has been shown that the information about the object vehicle provided by the long-range radar can be verified independently by the information of the two RTAs.

Combining the synchronization to GPS with the time triggered sending mechanism of the RTA enables the highly precise replay of data that is recorded during an in-vehicle test. It is left for future work to evaluate how the RTA can support ECU testing with recorded data. For recorded data as well as for in-the-loop simulations, the GPS synchronization combined with the time triggered sending ensures that independently of time, location and RTA device a test case can be repeatedly executed with precisely the same timing.

REFERENCES

[1] Daniel Ulmer, Steffen Wittel, Karsten Hünlich and Wolfgang Rosenstiel, "A Hardware-in-the-Loop Testing Platform Based on a Common Off-The-Shelf Non-Real-Time Simulation PC," in *ICONS 2011, The Sixth International Conference on Systems*, 2011.

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

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

[4] Christoph Marscholik and Peter Subke, *Road vehicles - Diagnostic communication: Technology and Applications*. Hüthig, 2008.

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

[6] Bart Broekman and Edwin Notenboom, *Testing Embedded Software*. Addison-Wesley, 2002.

[7] National Highway Traffic Safety Administration, "Forward Collision Warning System Confirmation Test," 2008.

[8] Christoph Marscholik and Peter Subke, *Datenkommunikation im Automobil*. Hüthig, 2007.

[9] ETAS GmbH, "LABCAR System Components," Access Date: December 28, 2011. [Online]. Available: http://www.etas.com/en/products/labcar_system_components-details.php

[10] ETAS GmbH, "LABCAR-RTPC - Real-Time Simulation Target for HiL Testing," Access Date: December 28, 2011. [Online]. Available: http://www.etas.com/en/products/labcar_rtpc.php

[11] Gerd Wittler and Jürgen Crepin, "Real-time and Performance Aspects of Hardware-in-the-Loop (HiL) Testing Systems," *ATZonline*, 2007.

[12] Jan Kiszka, "Xenomai: The RTOS Chameleon for Linux," Real-Time Systems Group, Leibniz Universität Hannover, Tech. Rep., 2007.

[13] G.i.N. Gesellschaft für industrielle Netzwerke GmbH, "MultiLog," Access Date: December 30, 2011. [Online]. Available: http://gin.de/index.php?device=1002&lang=de

[14] David Mills, "Network time protocol," RFC 958, Internet Engineering Task Force, 1985.

[15] Kang Lee and John Eidson, "IEEE-1588 Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems," in *In 34th Annual Precise Time and Time Interval (PTTI) Meeting*, 2002.

[16] Jürgen Luther and Hans-Werner Schaal, "Luftbrücke für Busdaten," *Elektronik automotive*, 2010.

[17] Guochang Xu, *GPS - Theory, Algorithms and Applications*. Springer, 2007.

[18] IT-Designers GmbH, "Real Time Adapter Datasheet (RTA-C4ENa)," 2010.

[19] Daniel Ulmer, Andreas Theissler and Karsten 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.

[20] Daniel Ulmer and Steffen Wittel, "Approach for a Real-Time Hardware-in-the-Loop System Based on a Variable Step-Size Simulation," in *Proceedings of the 22nd IFIP International Conference on Testing Software and Systems: Short Papers*, 2010.

[21] Daniel Ulmer and Andreas 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.

[22] Hermann Winner, Stephan Hakuli and Gabriele Wolf, *Handbuch Fahrerassistenzsysteme*. Vieweg+Teubner Verlag, 2009.