Measurement based timing failure detection in automotive embedded systems

Balázs Scherer, Gábor Horváth
Department of Measurement and Information Systems
Budapest University of Technology and Economics
H-1117 Budapest, Magyar Tudósok krt. 2. IE444
{scherer,horvath}@mit.bme.hu

Abstract

Statistics show that in safety-critical systems, projects should expect up to 80% of their resources to be spent on testing. Industrial experiences have shown that multitasking and real-time behavior based failures are among the hardest to identify. Our paper shortly describes the testing environment and traditional Grey box tests used for automotive ECU verification. We introduce the potential ways of complementing these traditional tests with measurements essential for timing failure detection. The problems of timing failure detection, like the uncertainty of statistical task execution time modeling and the difficulties of interrupt measuring and modeling are also presented. We introduce our measurement based low communication bandwidth solution for complementing traditional regression tests with timing failure detector capabilities. The state of this work and the merits and flaws of our solution are discussed. Our paper is closed with presenting additional possible usage of timing measurements and suggestions for applying modern hardware based tracing solutions in future tests.

1. Introduction

Automotive embedded software modules are typical examples of safety-critical systems. Developing software that can be marked reliable for this market requires much effort. Statistics show that in safety-critical systems, projects should expect up to 80% of their resources to be spend on testing [1].

The testing process of such embedded systems includes White box and Black or Grey box test types [2]. White box tests, like various static analyses and coverage tests are usually executed in the module level, and done by the software development team. Black or Grey box tests like limit value tests, functionality tests and regression tests are performed at the system or subsystem level, in the phase of module integrations. These tests are done by separate test teams that have no overview on the software source code.

In many cases the Black or Grey box testing process are done for a subsystem not for the whole functionality. For example the application level functionality of an ECU (Electronic Control Unit) is not present, because it is written parallel by a different manufacturer. In this case this function is substituted by a so called test library, to make the testing possible with some restrictions.

This paper describes the traditional way of Black or Grey box testing of an ECU. After presenting a current way of testing, and the failure modes covered by these traditional tests, suggestion for a measurement based timing failure detection is presented. This paper shows the software architecture and task model of a general purpose ECU, and also describes a way for measuring the run time of the selected tasks. These measured run time parameters are used for estimating the execution time of software tasks. The goal is to use these estimated task execution time parameters for various verifications. The current evaluation states of these verifications are discussed with their possible future usability.

2. Traditional ECU testing

Black box or Grey box tests examine the input and output variables of the systems, and perform limit checking or process identification methods to detect faults and errors. In the ECU testing process the regression tests are typical Grey box tests.

2.1. Regression tests

The regression tests are made after every software modifications. Regressions tests of an automotive ECU typically cover software modules [3] like input/output (sensor input, actuator output), communication (CAN, LIN, Flexray), on-board diagnostic (error filters, diagnostic trouble code storage), diagnostic communication (KWP2000, UDS, CCP, XCP), operation mode management (Initial, Software update, Ignition on, Limp-home …) and real-time operation system (task periodicity, task switching). These tests are done independently, focusing only on the given module and on its data and control connection to other modules.

2.2. Test environment

Regression tests are typically done in the environment shown on Figure 1.

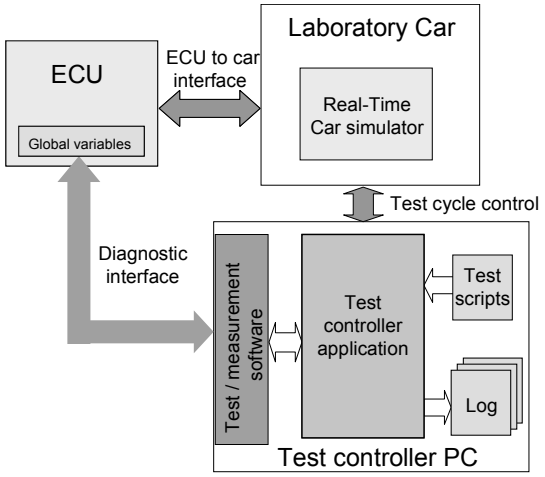


Figure 1. ECU testing environment

This environment contains the ECU, and a so called Laboratory car, which simulates the behavior of other ECUs, sensors and actuators of the car. The setup also contains a test controller PC that manages the behavior of the Laboratory car, executes the test scripts and runs the test measurement software. The test measurement software is able to monitor or change the internal global variables of the ECU. Most of the regression tests are done by checking, whether these internal variables contain the right values during the test cycle, and eventually modifying the proper internal variables to stimulate the system.

2.3. Test interface

The most critical part of this set-up is the test interface that provides access to the internal variables of the ECU. There are two traditional types of test interfaces used:

- Dual port RAM based test interface.
- Diagnostic communication based interface.

In the dual port RAM interface case the RAM of the ECU is replaced with a special hardware part that also makes the content of the RAM available for read and modification to the test tool. This solution is a nonintrusive one, with high data communication bandwidth. The main drawback of this dual port interface is that it is very costly, and from many points of view it is obsolete. Modern microcontrollers have enough internal SRAM for data storage, and their program memory Flash is accelerated enough for more than 100MHz CPU clock frequency with cache alike flash accelerator blocks. Therefore in many cases additional costly RAM chips are not used in these ECUs, and the dual port RAM approach cannot be applied. Certainly in modern microcontrollers there are ways to provide similar nonintrusive trace and modification interface, but these are chip vendor specific and currently not used commonly in testing processes. These technologies will be discussed in Section 9, in the future work part.

Another way for reaching the global variables of the ECU is the diagnostic communication based test interface. This solution is the most widespread used one, and test software systems like ETAS INCA [4], Vector CANape [5] support this solution. Diagnostic communications are often done on the same network interface as the normal communication. This technique is an intrusive one, however the load caused by diagnosis is treated as a normal load, and therefore if it is lower than a certain value, which is about up to 10% of the communication bandwidth, it should be handled by the ECU at any time.

In the automotive industry there are four diagnostic protocols that can be used for this kind of testing. These are the CCP (Can Calibration Protocol), XCP (Universal Measurement and Calibration Protocol Family), KWP2000 (Keyword Protocol 2000) and UDS (Unified Diagnostic Services).

KWP2000, and UDS mainly used for after production service purposes. Due to its small footprint and easiness CCP is the most commonly used protocol in the test and development processes.

CCP [6] is a simple master – slave protocol using CAN (Controller Area Network) as communication

interface. By using CCP the tester (master) can read or modify the content of the ECU's (slave) global memory. Tester can also program the Data Acquisition Processor in the slave to make it send measurement data periodically. The data to be sent periodically is specified by using the Object Descriptor Tables that describe the memory addresses and lengths of the data.

XCP is the upgraded version of CCP. While CCP is limited to the CAN bus, XCP can also use FlexRay, Ethernet and USB as test interface.

2.4. Global variables

To handle the global variables of the ECU, these diagnostic techniques are using the MAP or ELF files generated during the compile of the source code of ECU software. These files contain the name and address pairs of every global variables of the system, which will be the scope of all traditional regression tests. The information stored in the MAP or ELF files does not contain the linkage between the raw stored variables and their real physical meanings. Therefore to specify this transformation an ASAM-MCD2 MC [7] (ECU Measurement and Calibration Data Exchange Format) market name ASAP2 format file is generated from the names, memory address and sizes of the variables. This ASAP2 file complements the MAP/ELF file information with a transformation formula describing the way of converting a raw variable data to its real physical form. These ASAP2 files are used by the measurement and calibration tools like Vector CANape or ETAS INCA.

3. Timing failures

Traditional Regression tests are mainly focusing on one software module. Feedbacks and experiences have shown that these tests probably do not cover some module to module interaction or timing related problems. These problems are manifested in failure logs like unwanted resets, and strange system behavior in some situations.

Therefore there was a need for new tests that try to catch the cause of these problems.

3.1. Failure model

The failure model of such a real-time embedded system [8] can be divided into sequential and multitasking real-time behavior based failures shown on Figure 2.

The feedbacks and experiences have shown that the test coverage for sequential failures is high enough. Traditional white box and regression tests catch these types of failures, but the multitasking real-time failure detection is not perfect, so it should be improved.

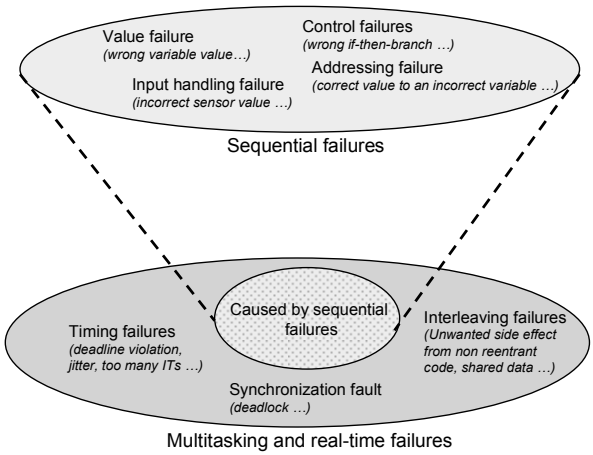


Figure 2. Failure model

There are three categories of such multitasking and real-time failures: the Timing failures, Synchronization failures and Interleaving failures.

In our work we will focus on the Timing failures, because the static analysis in the white box test phase leaves these one uncovered the most.

3.2. Possible solutions

The most foundational thing of Timing failure detection is the static prediction or runtime measurement of software task WCETs (Worst Case Execution Time). These execution time predictions or measurements can be used to calculate the worst case response time for each task, and therefore analyze whether the system will be able to behave in real-time in every situation by keeping the schedule of the tasks. Real-time systems out of their schedule can show symptoms like unwanted resets and strange behavior for a short time.

Static calculation of execution times and WCET is not a trivial task and many articles discuss its problems [9]. A recent survey is a good introduction to these methods [10]. The two main sources of WCET deviation is from low-level hardware optimization features, and high level path dependencies. Low level features are for example the effects of caches and pipelines. These dependencies have an impact on close neighboring blocks and in the general case the dependency decreases for more distant blocks. High

level features are for instance the data dependent paths and mutually exclusive paths.

There are many tools regarding this topic, and many of them are commercially available. Some tools use pure static analysis of the program, while other tools combine static analysis with dynamic measurements of the execution times of program parts. Unlike most applications of static program analysis, WCET tools must analyze the machine code, not (only) the source code. This means that the analysis depends on the target processor, so WCET tools typically come in several versions, one for each supported target processor or even for each target system with a particular set of caches and memory interfaces. Some parts of the machine code analysis may also depend on the compiler that generates the machine code. A typical example for such compiler dependency is provided by Hitex: four compilers (GNU GCC, and three commercial compilers) were compared, with the same processor (LPC2294 an ARM7TDMI core processor) at the same speed, with the same benchmark and even the commercial compilers were differing more then 30% in their execution time results [11].

Another way of execution time verification is a purely measurement based solution, when the timing parameters are acquired during run time testing. However this is theoretically simpler, but requires code instrumentation and its coverage is limited by the run time test (only execution paths affected by the run time test can be inspected).

3.3. Restrictions

To make our work the easiest to apply in the future, in accordance with our industrial partner we decided to complement the existing grey box tests with a timing failure detector property. Therefore we used a measurement based approach that can be integrated into the test environment described in section 2.

Our main goal is to perform this new test property as “background” verification during each traditional regression test. This way we can increase the test coverage without causing additional costs by lengthening the test process. The term “background” means the following: during regression tests, this new test simply measures some important features, by using the traditional diagnostic channels (this way do not cause significant additional load to the system under test), and at the end of the traditional test, a statement is given highlighting the noticed timing failures.

4. Measurement

The first phase of planning the timing failure detector is the selection of features and parameters to be measured during the regression tests. A detailed knowledge about the software architecture of an automotive ECU is needed for this decision. The architecture of the system under test can be used also to create a model or simulation of it. This is important, because in many cases the complexity of the ECU software is too big for trying out theories and the cost and time schedule of the industrial test stands could also be a bottleneck of experimentation.

4.1. System model

An ECU or TCU can be treated as a general purpose real-time embedded system with the software modules shown on Figure 3 (this figure is based on OSEK COM specification, with some modification).

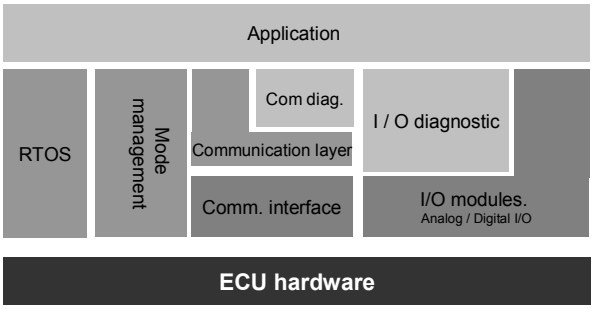


Figure 3. ECU software architecture

From the measurement and instrumentation point of view the RTOS model of the system is the most important. Specifications for automotive RTOS support are described first in the OSEK/VDX OS [12] standard, which is used as a base of AUTOSAR specifications for operating systems. Most of the operating systems used in ECUs are conform to this standard.

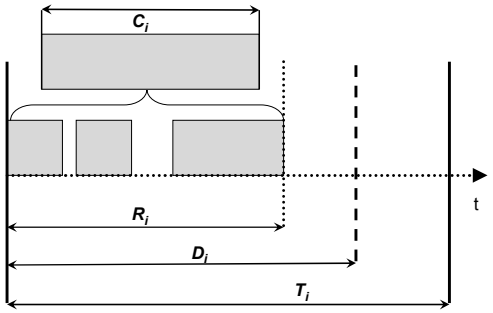
The RTOS model of a usual ECU can be described as:

- Fixed priority, preemptive scheduling.
- Fixed task periods with 2.5ms, 5ms, 10ms, 20ms, 30ms, 50ms and 100ms tasks.
- Tasks with lower period have the higher priority.
- Interrupts have short and predictable execution.

The worst case response time of the tasks can be calculated accordingly to the Deadline Monotonic

Analysis [13] as it is can be done in many automotive OSEK/VDX OS based system [14].

Deadline monotonic analysis (DMA) is a technique to calculate the worst-case response time of tasks. It can be used to ensure that all tasks will meet their deadlines, or in other words, that the system is schedulable. Standard notations of DMA are shown on Figure 4.



T_i is the period of task i
D_i is the deadline of task i
C_i is the worst-case execution time of task i
R_i is the worst-case response time to task i

Figure 4. DMA notations

T_i and D_i can be derived from requirements. While the value of C_i is need to be measured.

4.2. RTOS instrumentation and measurements

In some RTOS the built in instrumentation can provide the value of C_i, however it is not a trivial task, because it requires low kernel level instrumentation. Our industrial partner's RTOS hasn't got such instrumentation, and they refuse to make any such deep kernel level modification on their RTOS, because of its high testing cost. The built in instrumentation of that RTOS can provide the value of R_i, the number of interrupts in the last 2.5ms time period and the processor load in the last 100ms period, so we should use these values.

Unfortunately the value of R_i alone is useless, because a low R_i value can cover a high C_i value and vice versa as shown on a measurement results of Figure 5. But, complementing the value of R_i with the task end timestamp the scheduling can be restored, and an estimation for C_i can be given.

The measurement of task end timestamp is very simple, it does not require deep kernel level instrumentation, and our partner accepted to do this implementation.

The measured parameters have the resolution of 1μs. This resolution is enough for timing failure detection, and ensures that the amount of data to be transferred do not overload the communication bus.

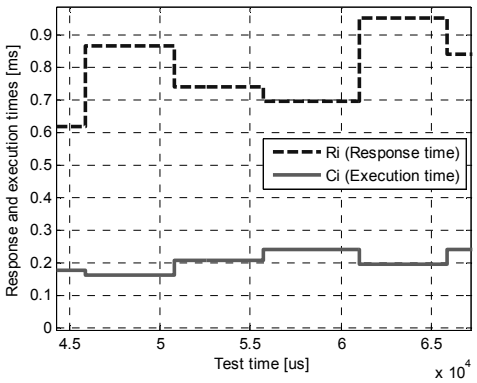


Figure 5. R_i and C_i relationship

4.3. Test system

The development of the Timing failure detector is done in three stages. The first stage is to try out the methods in a Matlab based scheduler simulator. The second state is to verify the theories in an ECU simulator, and the third last stage is to apply it on a real ECU.

The ECU simulator is based on the STM32F107 ARM Cortex M3 based MCU, with 72MHz of CPU clock speed, 256kbytes of Flash, and 64kbytes of RAM, and 2 CAN channels. The properties of this ECU simulator is about the same as a mid or low range automotive ECU. We use the Vector CCP stack on this card as diagnostic communication. Vector CANape is used as measurement software, and we substitute the Laboratory car with a National Instruments cRIO modular hardware. Our test system just likes the one shown on Figure 1, but much simpler. Our ECU simulator internally provides the skeleton of the software layers shown on Figure 3. We can use three types of RTOS make our tests RTOS independent. These three RTOS are FreeRTOS, μC-OS and eCOS. The instrumentation described in 4.2 is implemented for all of these RTOS, but to verify our theories for some RTOS we added more sophisticated instrumentation too.

5. Worst case task response time calculation

Timing failure detection is based on the calculation of tasks WCETs. Measurement based techniques for

task WCET estimation were evaluated in our Matlab based simulator, then in our test system. The first step of task WCET estimation is to estimate the execution time (C_i) for every task in every period. This is done by using R_i and other measured parameters. To estimate Ci from R_i a reconstruction of scheduling is needed.

5.1. Reconstruction of scheduling

From the known priorities and periodicity of tasks, the measured response time and from the end of task time stamps the scheduling can be restored. This statement is true with 2 restrictions.

The first is that we do not take into the account the effect caused by priority inversion, where the higher priority tasks are blocked by a lower priority one. This restriction can be done in our cases, because in such systems it is usually prohibited to use blocking task synchronization methods that can lead to priority inversion.

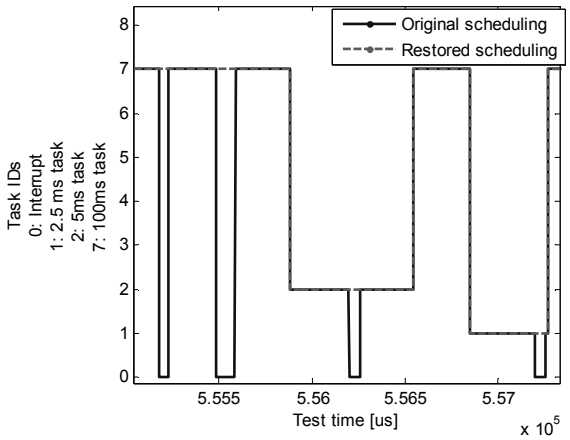


Figure 6. Restored scheduling vs. original

The second is to neglect the effect of the interrupts, which is a much harder problem to solve. Figure 6 shows an example for scheduling restoration, without compensating the effects of interrupts.

5.2. Effects of interrupts

In most of the companies developing embedded software products for safety critical systems, there are guides for handling and using interrupts. These guides according to safety standards [15] usually make more and more restrictions for the use of interrupts as the safety level of the product is become higher. For example, for a SIL (Safety Integrity Leve) 1 or 2 product it is only a recommendation to use less interrupt then a normal one, for a SIL 3 product it is a hard rule to use as small amount of ITs as possible, and handle events with periodic polling instead. This means that in safety critical systems we can assume only a few interrupt sources.

Interrupt modeling has two main parts to discuss. The first is the execution time prediction of an interrupt. The second is the distribution of interrupt occurrences in time.

The execution time prediction of interrupt can be based on statistical WCET calculation results. Interrupts in safety critical systems generally has only a few high level execution time dependences, and mainly the low level ones like IT jitter, pipeline flush etc. dominates. Our survey and measurements made on different interrupt sources in the test system shows that we can suggest an extreme value probability distribution for interrupt execution time modeling.

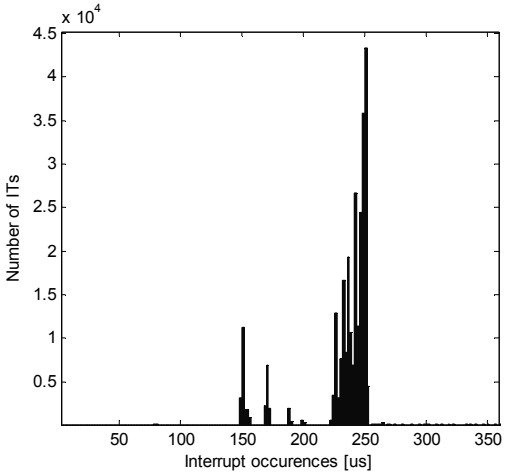


Figure 7. CAN powertrain IT distribution

The second problem is to specify the distribution of interrupt occurrences in time. Some of the few IT sources will be periodic for a sure, like RTOS heart beat timer, but most of them will be unpredictable. However the events like message receiving interrupts of communication interfaces seems to be unpredictable and random, but that is not absolutely true. In automotive systems most of these events tend to be periodic, even when using an event base communication like CAN. For example in normal communication mode every CAN message is transmitted periodically, but in non TTCAN networks these are not synchronized.

Figure 7 shows an example for interrupt occurrence distribution in a 500kbit/sec powertrain CAN network. It is impossible to create an unequivocal probability distribution for every interrupt sources. Each source should be modeled separately.

5.4. Statistical worst case R_i calculation

Usually Deadline Monotonic Analysis (DMA) is used to calculate the worst-case response time of tasks. DMA is using the following iterative formula:

$$R_i^0 = C_i$$

$$R_i^{n+1} = C_i + \sum_{\forall k \in hp(i)} \left\lceil \frac{R_i}{T_k} \right\rceil C_k$$

where

$$\sum_{\forall k \in hp(i)} \left\lceil \frac{R_i}{T_k} \right\rceil C_k$$

is the total interference from all higher-priority tasks, and ***hp(i)*** is the set of tasks with priority higher than ***i***.

In this formula the worst case task execution times should be used as C_i and C_k parameters. In our example it only can be done by using probability distributions in the following way:

1. Compensate the IT effects for every task ***i*** from the restored scheduling to get the probability distribution of C_i.
2. Estimate worst case C_i distribution for every task.
3. Make the iterative computation with the probability distributions of tasks worst case C_i and with the probability distribution of interrupt effects.

There are suggestions for statistical worst case response time calculations in the literature [16],[9] but generally response time calculation is a very difficult NP-hard problem [17]. To make that problem harder in this situation, we should use uncertain arguments, because as presented in section 5.2 the effects of interrupts generally cannot be modeled correctly by probability distributions, and in the procedure above we should use the IT models twice. As a conclusion, the results of the statistical worst case response time calculation can be given numerically, but these results could lead to a very long tail and imprecise probability distributions at the end. As we studied it, these computation doubtfully could give distributions, where the probability for timing failure is lower than 10^{-8} or 10^{-9} (general requirement for such systems) for any test case. Therefore these calculations would indicate a huge value of false positive detections, and therefore the test would be useless.

The results of the probability based worst case R_i calculation could be improved greatly, by applying detailed IT measurements. This detailed IT measurement should provide the type of the IT, the start time and the end time of it. Therefore it would be possible to restore the scheduling with the effect of interrupts, including IT nestings too. From this restored scheduling the WCET of tasks can be predicted in a more exact way, and the worst case response time calculation could use a better IT model derived from the measurements.

This approach would be the best way for timing failure detection, but the measurement communication bandwidth of this solution is very high. In the most optimistic calculations for data transfer, we can assume 2 bytes pro ITs to log (low, 10µs resolution start and stop timestamp in a 2.5ms period), and 2 ITs pro 1ms (a normal automotive system has more than 2 ITs pro 1ms). This very optimistically calculated data transfer indicates more then 15% load, without any other measurements to a 500kbit/sec CAN network. In section 2.3 we specified that up to 10% of diagnostic communication load is allowed for our test in a traditional environment. So as a conclusion this method cannot be applied to complement traditional Grey box tests.

6. Simplified, low data rate timing failure detection

Section 5 has shown that statistical worst case response time calculation cannot be used in the Grey box test environment introduced in section 2.2. The question is, whether there is any way to use the measurements shown in section 4.2 to make a statement about the timing healthiness of the system.

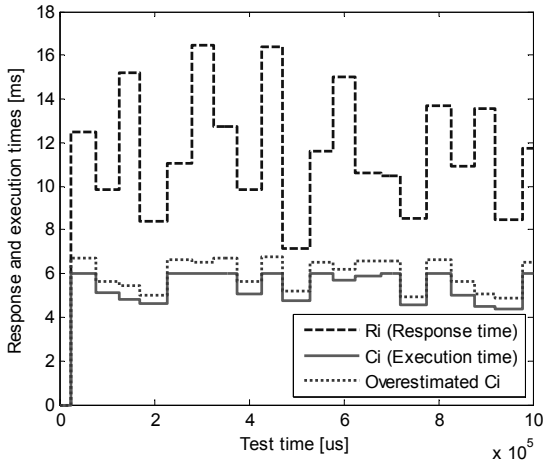


Figure 8. Overestimated task execution times

Our suggestion is to use the reconstructed scheduling shown in section 5.1, but without trying to remove the effects of interrupts from it. The result is a set of overestimated executing times for every tasks. Figure 8 shows a sample relation between this overestimated execution time, the task response time and real execution times.

There is a guarantee that the maximum of these overestimated execution times will be higher than the worst case execution time in the test cycle. These overestimated worst case execution times could be used for the DMA algorithm, without taking into the account the effect of the interrupts, because the worst case overestimated execution times already contains the effects of the ITs. But there is no guarantee that the result of this calculation will be the realistic worst case situation, because the maxima of the overestimated execution times not definitely contain the maxima of interrupt interferences. Therefore the possible differences between the interrupt effects embedded into the overestimated execution time maxima, and the worst case interrupt load should be compensated.

Our suggestion is not to make this compensation for task to task, because that procedure would not fit to the mainly periodic nature of interrupt occurrences. So instead of compensating the possible interrupt differences to task to task we suggest to take this compensation at the system schedule level. To do this we need a prediction for the schedule ability of the tasks, without calculating the worst case response times with DMA.

The so called Liu and Layland bound can be used for this purpose [18]. This bound specifies that a system consists of ***n*** tasks is schedulable if the sum of the maximum utilizations of the tasks, denoted as ***U***, satisfies the inequality:

$$U \le n(2^{\frac{1}{n}} - 1)$$

The Liu and Layland bound gives a sufficient and hence conservative condition. A system may be schedulable though its maximum utilization exceeds this bound.

Our suggestion is to use this conservative condition and complement it with the worst case IT difference:

$$U \le n(2^{\frac{1}{n}} - 1) - Id$$

Id is the difference between the maximum IT load measured during the test, and the minimum of IT loads of the periods, which are used for the calculation of ***U***. A period is used for the calculation of ***U*** if an overestimated task execution time is selected from it. The IT load is calculated by subtracting the sum of overestimated task execution times plus the CPU load from the period of time the CPU load was measured (usually it is synchronized with a low period task like the 100ms task). This IT load is therefore the time when the CPU was handling interrupts during the execution of the idle task.

Our experiences have shown that this is a really conservative higher bound, therefore if the calculated utility of the system pass this, then there is a very low probability to any timing failure remains in the measured test case.

The diagnostic communication load of this solution is not more then 8% with using CCP messages in a 500kbit/sec CAN network. Therefore it can be applied in traditional Grey box tests.

7. Using timing measurements in regression tests

Methods presented in section 5 and 6 do not take into account the type of the tests. These sections only deal with the measured values and their applicability; however the term regression test could involve features that can be beneficial for failure detection.

Many times regression tests are made after a modification of a software system that previously had been marked as reliable [2]. This modification can be a change of a parameter or adding a new function to the system, but the important thing is that there was a previous version marked as reliable. Therefore, if a timing data set of a good version or versions can be collected and learned by the checker, then many potential errors can be uncovered by checking. The question is whether there are any differences between the timing characteristic of the new software version and the previous ones.

7.1. WCET deviations between tests

Collected information form the previous measurements can be used to check the differences between the new and the stored overestimated worst case execution times. It is very important to match comparable test cases. Thus the tester should contain a large set of data for every regression test cycle with every hardware version and every major software version.

The collection of such data set is a very time demanding one, but it can be done during the normal

tests, and this comparison can be an efficient way to catch problems. For example in a situation where the part of a software system is tested, there are probably no deadline violations, but a significant increase in an execution time compared to a previous version could indicate some programming faults.

7.2. Executing time deviations in different operating modes

Automotive ECUs have many operating modes. Some of them are listed in section 2.1. Task execution times statistics highly depend on the operation modes. Execution times much smaller then the global worst case execution time could indicate serious errors in some operating modes. A real-life precedent for this is an incorrect behavior in power-safe modes, which lead to the discharge of the battery. Therefore it is important to go beyond the global worst case execution time test. The checker should perform an execution time check for every operating modes of the system separately.

7.3. Executing time clustering

Not only the worst case execution times can contain information, but the change of the execution time parameters during a test flow also can be interesting. Therefore it is beneficial to try to make clusters form execution time measurements. These clusters represent the major execution paths of the system software, however an exact relationship cannot be given between them (many paths can have similar execution times). These clusters can be used to check, whether the system during a test flow tends to be executing the same functionalities as it was done in the previously good releases. Significant differences between the execution paths can also signal software faults.

8. Conclusions

Our paper shortly described the testing environment and traditional Grey box tests used for automotive ECU verification. We introduced the potential ways of complementing these traditional tests with measurements essential for timing failure detection. In section 5, we have presented the problems of restoring the scheduling of the system that essential for acquiring the execution times of tasks. The difficulties of interrupt modeling are also discussed. Section 5.4 has shown a probability based approach for compensating the effects of interrupts, but our studies has shown that the amount of measured information with traditional diagnostic communication based solutions is not enough to give a usable probability distribution based solution for the timing failure detection problem. Section 6 has shown a highly simplified low data rate solution for timing failure detection. This method can be used in the traditional test environments and diagnostic communication. It can give a high probability for signaling situations from the measured data that can lead to timing failure. Finally section 7 has introduced the possible further applications of timing measurements in regression test. These applications require a huge timing data set from reliable applications, and based on this data set differences can be detected in new software versions.

The results and theories presented in this work have been evaluated in the test system presented in section 4.3, and preliminarily studies have been done for some of the methods by using data set from real automotive regression tests. The actuality of this problem is shown by a recent publication of Vector Informatics showing their solutions for timing analysis [19].

9. Future work

The theories presented in this paper are already evaluated in the test system, but many more measurements, and failure detection case studies are needed to presents their applicability in real situations. Adopting the test from the test system to a real environment also provides challenges.

9.1. Modern tracing techniques

Section 2.3 has presented the test interfaces used in Grey box tests. Dual memory interfaces are getting more and more obsolete for such tests, but most of the modern microcontrollers provide ways for high data rate nonintrusive data and instruction tracing.

A typical example for such embedded trace support is the CoreSight Debug support and Embedded Trace Macrocell of ARM [20]. These hardware blocks provide nonintrusive debug and trace interface even in the lowest end 32bit microcontrollers. On-the-fly debugging, integrated data trace and optional instruction trace are the main features of these blocks. Data trace provides features like program counter sampling, event counters, and interrupt execution tracing with timing statistics. Instruction Trace enables analysis of execution history, and can be used for code coverage, and performance analysis. These hardware blocks also enable the interfacing to the internal memory and peripherals in a nonintrusive way (AHB access port for ARM Cortex cores), therefore variables specified in ASAP2 files can be acquired and modified

in a same way as it its done in a dual port memory test interface case.

The CoreSight Debug support and Embedded Trace Macrocell are ARM specific solutions but the IEEE-ISTO 5001-2003 Nexus port also specifies similar goals, so in the future most of the microcontroller will have similar functionalities.

Another good example for the need of such data trace support is the TMS570 of Texas Instruments. The TMS570 is a state of the art dual core ARM Cortex-R4F based floating point MCUs that meets IEC61508/SIL3 safety standards and released in 2009. These series of microcontrollers is primary designed for safety critical applications, with redundant dual core CPU and memories with error correcting capabilities. The TMS570 is enabled with a CoreSight debug support, but beside this support it provides a RAM trace port (RTP) module, which allows to do data trace of CPU or other master accesses to the internal RAM and peripherals with up to 133MBytes/s transfer rate.

9.2. Applying tracing techniques

Currently the diagnostic communication based testing is the most widespread used. Therefore it is important to give solutions for that low data speed interface, however unquestionably the trace hardware interfaces introduced above will be used in future testing environments. From our point of view these interfaces enables more precise measurements, thus the probability based timing failure detection approach presented in section 5.4 can be applied. These tracing methods also could have great use in the checks presented in section 7. For example, by using precise interrupt measurement a knowledge base for every interrupt type can be built and therefore the worst case interrupt situations could be predicted much more realistically, then general probability distributions.

We are about to improve our test system with such tracing tools, and adopt our methods for this technology.

10. References

[1] Brooks, F.P. Jr., "The Mythical Man-Month: Essays on Software Engineering", *Addison-Wesley Pub. Co.*, 1995. ISBN-10: 0201835959.

[2] Thomas Müller and others, "Certified Tester Foundation Level Syllabus", *International Software Testing Qualifications Board*, Version 2010. www.istqb.org.

[3] Joerg Schaeuffele, Thomas Zurawka, "Automotive Software Engineering: Principles, Processes, Methods, and Tools*" SAE International*, June 1 2005, ISBN-10: 0768014905.

[4] ETAS Group, INCA homepage: http://www.etas.com/en/products/inca.php

[5] Vector Informatik GmbH. CANape homepage: http://www.vector.com/vi_canape_en.html

[6] ASAM MCD-1.CCP: "CAN Calibration Protocol", *ASAP Standard*, Version 2.1, February 1999.

[7] ASAM MCD-2 MC: "ECU Measurement and Calibration Data Exchange Format" market name ASAP2. version 1.6.1

[8] H. Thane. "Monitoring, testing and debugging of distributed real-time systems" *In Doctoral Thesis, Royal Institute of Technology*, KTH, S100 44 Stockholm, Sweden, May 2000. Mechatronic Laboratory, Department ofMachine Design.

[9] Insup Lee, Joseph Y-T. Leung, Sang H. Son, editors, "Handbook of Real-Time and Embedded Systems", *Chapman & Hall/CRC*, 200y, ISBN-10: 1-58488-678-1.

[10] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, P. Stenström, "The worst-case execution time problem - overview of methods and survey of tools" *ACM Transactions on Embedded Computing Systems*, Volume 7, Issue 3, April 2008.

[11] Trevor Martin "The Insider's Guide to the Philips ARM7-based Microcontrollers. An Engineer's introduction to the LPC2100 series", *Hitex Ltd,* 2005 ISBN: 0-9549988 1

[12] OSEK/VDX "Operating System Specification 2.2.3" February 17th, 2005. http://portal.osek-vdx.org/files/pdf/specs/os223.pdf

[13] Ken Tindell "Deadline Monotonic Analysis", *Embedded System Progrming magazine*, June 2000.

[14] Andrew Coombes, "Deadline Timing and OSEK", *Embedded Systems Programming magazine,* December 2002.

[15] IEC standard 61508, „Functional safety of electrical/electronic/programmable electronic safety-related systems", http://www.iec.ch

[16] Mark K. Gardner, Jane W. S. Liu "Analyzing Stochastic Fixed-Priority Real-Time Systems", *Tools and Algorithms for the Construction and Analysis of Systems. Lecture Notes in Computer Science,* 1999, Volume 1579/1999, 44-58, DOI: 10.1007/3-540-49059-0_4

[17] Friedrich Eisenbrand Thomas Rothvoß "Static-priority Real-time Scheduling: Response Time Computation is NP-hard" August 21, 2008.

[18] C. L. Liu and J. W. Layland."Scheduling algorithms for multiprogramming in a hard-real-time environment", *Journal of the Association for Computing Machinery*, 20(1):46-61, January 1973.

[19] Helmut Brock, Vector Informatik, "AUTOSAR OS measures task execution times", *EETimes Design*. September 2010.

[20] Alex Growcoot „Cortex-M3 Debug & Optimization" *ARM Technical Symposia*, November 2009.