

# Revealing Runtime Conflicts

## AUTOSAR operating system directly measures task execution times

**Software design is easy when the processor can provide far more computing time than the application needs. In practice, however, just the opposite is often the case. Good real-time embedded programming requires a lot of time and effort by the software engineer to steer clear of the cliffs of insufficient computing time. Now an embedded operating system offers welcome assistance; not only does it determine runtimes of tasks and interrupts directly, the “TimingAnalyzer” analysis tool also reveals runtime conflicts.**

Modern embedded software is often complex and consists of multiple processes (tasks) running time-independently and interrupt service routines (ISRs). The use of a pre-emptive operating system like MICROSAR OS from Vector Informatik makes a significant contribution toward realizing a clear structure, shortened development times and better maintainability. The interactions of all implemented tasks must also be considered to obtain a stable system. Primary areas of focus are to achieve data consistency when accessing commonly used data areas and to optimize the runtime behavior of individual tasks.

Mechanisms provided by the operating system help to ensure data consistency. However, the developer is responsible for correct runtime behavior. The developer must ensure that all processes can be completed within their time limits.

Consider a case in which a user should be able to notice an effect within 10 ms after actuating a control; this is a typical example of a time limit for execution of a task. However, developers are usually confronted by periodic processes in which it is essential to

complete every operation before the beginning of the next cycle. Another aspect relevant to software design is that a running task may be interrupted at any time by an interrupt or a higher priority task.

Figure 1 illustrates what is required based on an application consisting of Task A and interrupts ISR 1 and ISR 2. The task is called periodically every 10 ms and requires an execution time of 5 ms; this results in 50% processor load. The execution time of the two interrupts is 3 ms each. Although interrupting the task with just one interrupt is noncritical (cycle 2), when both interrupts occur in the same cycle this leads to a computing time of 11 ms, which is unacceptable since it would extend beyond the time of the next starting point (cycle 5). Such time conflicts may lead to sporadically occurring and difficult to identify errors such as data losses, timeouts, perceptible delays, etc. Therefore, a thorough analysis is essential before delivery of the software.

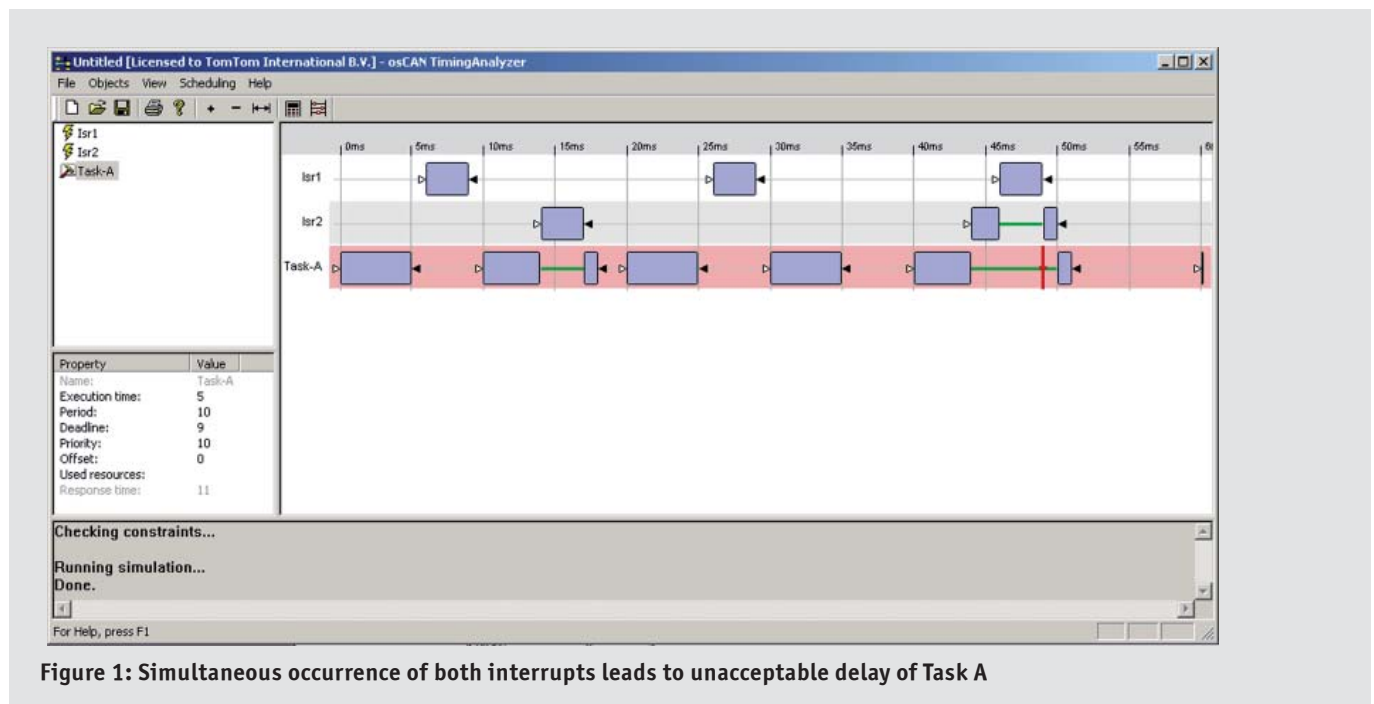


Figure 1: Simultaneous occurrence of both interrupts leads to unacceptable delay of Task A

### The right way to measure execution time in the operating system

Each analysis is based on precise knowledge of task and interrupt execution times, which can be determined by various methods. A simple solution is to insert special test routines for toggling IO ports at the beginning and end of the tasks. The IO ports are then observed, e.g. via an external oscilloscope or logic analyzer. Another option is to trigger special measurement routines that generate time stamps via their own timers and pass their data to a PC via a (serial) data interface.

The disadvantages of these methods are evident:

- > At many points, the code needs to be modified manually; this involves substantial effort and there is high risk of errors.
- > The measured time interval, besides covering the execution time of the routine being analyzed, also includes the execution times of all interrupting tasks and ISRs.

Effective immediately, MICROSAR OS now offers a function for measuring the execution and blocking times of selected tasks and interrupts directly in the operating system, where interruptions are considered in calculating execution time. This simplifies the test engineer's work significantly, because now it is no longer necessary to modify the application code. All that is needed is a test routine for reading out the measured data. In principle, any supported serial or parallel interface of the processor or ECU hardware or a memory map in the emulator can be used for this. The user can

then conveniently switch the measurement function on or off in the operating system configuration. Figure 2 describes these simplifications.

### Execution time analysis

After execution time measurements have been made in the operating system, the next step is for the software engineer to analyze the measured data and check whether each task can process its steps within the scheduled time slot. Vector Informatik supplies the "TimingAnalyzer" analysis tool with the operating system for this purpose.

The TimingAnalyzer applies the principle of Deadline Monotonic Scheduling. This approach is based on the assumption that for each task there is a time limit for processing its job. In the case of periodic tasks, this time limit must be less than the period. The system simulates aperiodic tasks as periodic tasks in which the period equals the minimum Inter-Arrival Time. The Inter-Arrival Time is the time between two task activations, e.g. two activations of the turn signal stalk.

The software engineer inputs the measured execution times of the tasks and ISRs into the TimingAnalyzer and supplements this data by indicating task priorities, deadlines to be met and typical cycle times. Based on these parameters, the TimingAnalyzer then checks to see whether all tasks always conform to their time limits, and it shows the computed time behavior graphically (Figures 1 and 3).

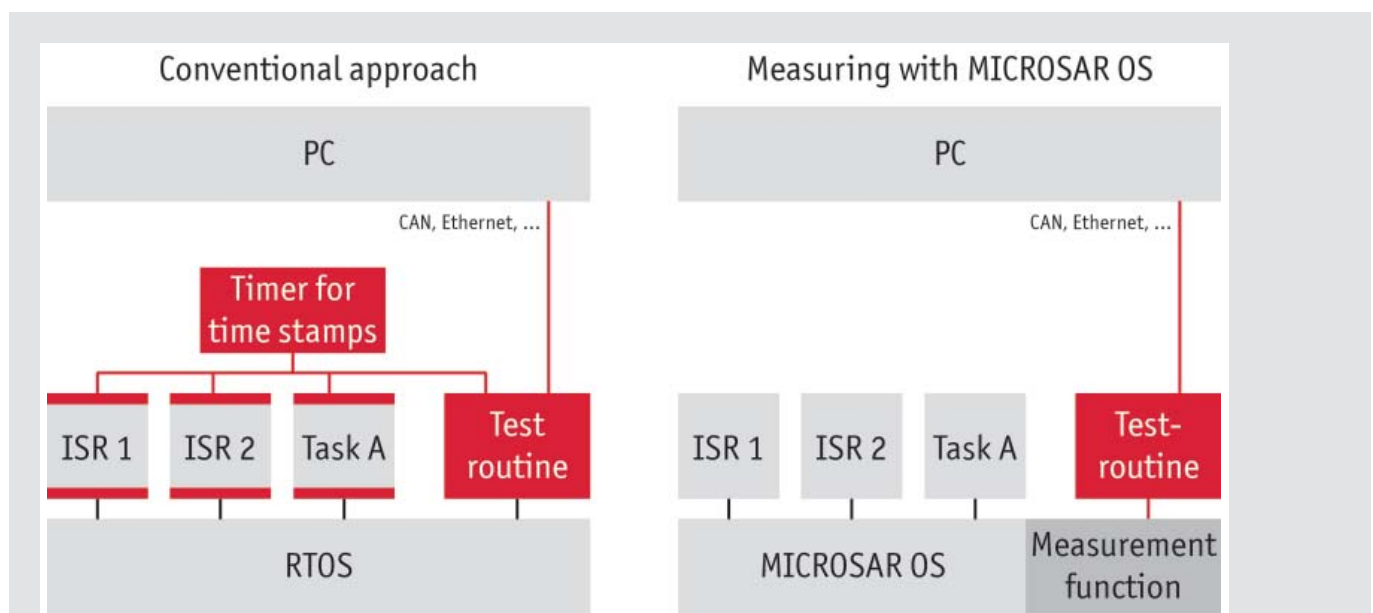


Figure 2: Methods of runtime measurement: The areas shown in red are software modifications to be made by the user

The analysis algorithm also takes into account the special dependencies of tasks and interrupts that share hardware or software resources.

Figure 3 shows a solution found with the help of the Timing-Analyzer for the example of Figure 1. ISR 2 is subdivided into a core function with an execution time of 1 ms on the interrupt level and a low-priority post-processing with an execution time of 4 ms on the task level. Such a scenario can be set up quickly in the Timing-Analyzer, and results are available within just a few seconds. This gives the developer a quick and easy way to check the effects of large changes made to the source code or new configurations.

### Outlook

The extended capabilities of MICROSAR OS together with the Timing-Analyzer create an efficient duo for embedded software development. While the operating system simplifies the process of measuring execution times of tasks and interrupt service routines, the analysis tool lets users graphically display results and check for execution time conflicts. Currently, measured data are acquired by dedicated measurement routines and are transferred to the analysis tool manually. Future plans are automatically uploading the data to the PC and TimingAnalyzer using standardized protocols such as XCP.

### Translation of a German publication in Automobil Elektronik, 1/2010

#### Literature:

- [1] [www.asam.net](http://www.asam.net)
- [2] [www.autosar.org](http://www.autosar.org)

#### Links:

- Homepage Vector: [www.vector.com](http://www.vector.com)
- Product Information MICROSAR OS: [www.vector.com/autosar](http://www.vector.com/autosar)

#### All Figures:

Vector Informatik GmbH



#### Dr. Helmut Brock

is product manager for the real time operating systems osCAN and MICROSAR OS in the productline "Embedded Software"

Vector Informatik GmbH  
 70499 Stuttgart  
 Germany  
 Tel. +49 (0)711/80670-385,  
 Fax +49 (0)711/80670-425,  
 E-mail: [helmut.brock@vector.com](mailto:helmut.brock@vector.com)

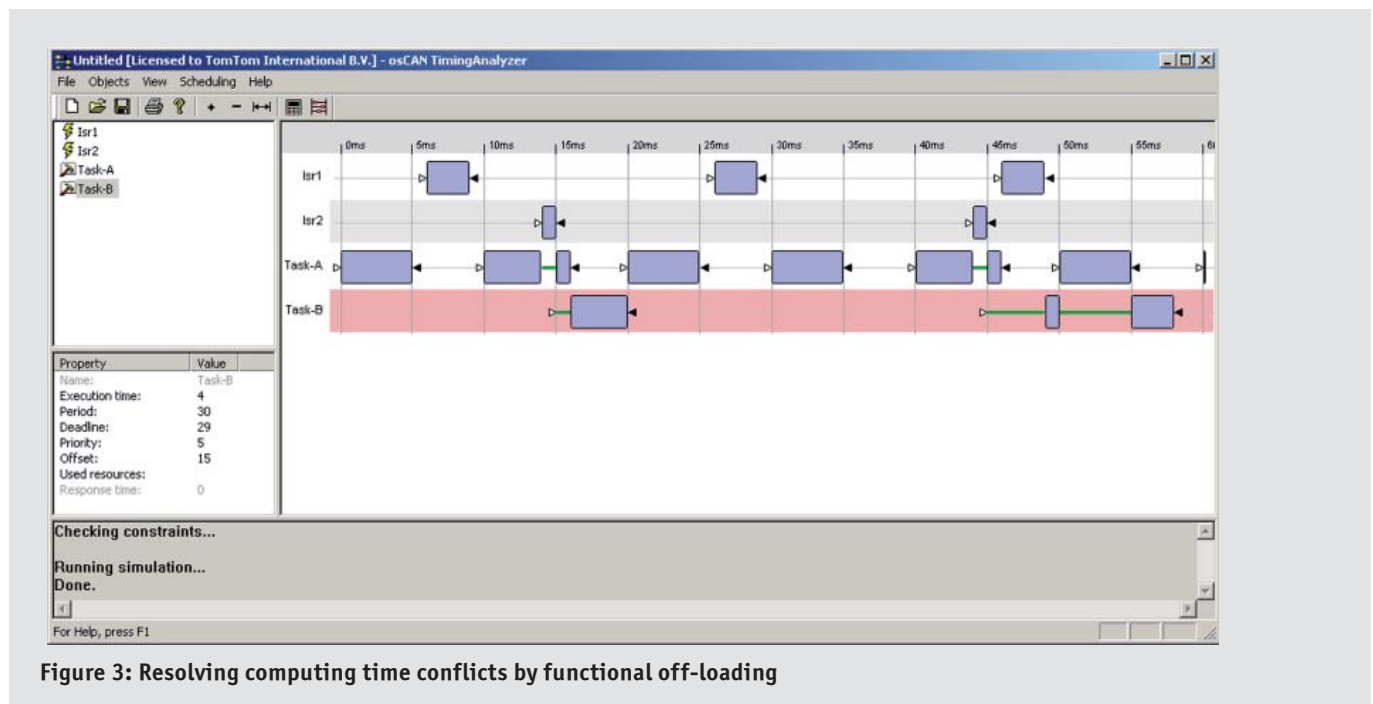


Figure 3: Resolving computing time conflicts by functional off-loading

**>> Your Contact:**

**Germany and all countries, not named below**

Vector Informatik GmbH, Stuttgart, Germany, [www.vector.com](http://www.vector.com)

**France, Belgium, Luxembourg**

Vector France, Paris, France, [www.vector-france.com](http://www.vector-france.com)

**Sweden, Denmark, Norway, Finland, Iceland**

VecScan AB, Göteborg, Sweden, [www.vector-scandinavia.com](http://www.vector-scandinavia.com)

**Great Britain**

Vector GB Ltd., Birmingham, United Kingdom, [www.vector-gb.co.uk](http://www.vector-gb.co.uk)

**USA, Canada, Mexico**

Vector CANtech, Inc., Detroit, USA, [www.vector-cantech.com](http://www.vector-cantech.com)

**Japan**

Vector Japan Co., Ltd., Tokyo, Japan, [www.vector-japan.co.jp](http://www.vector-japan.co.jp)

**Korea**

Vector Korea IT Inc., Seoul, Republic of Korea, [www.vector.kr](http://www.vector.kr)

**India**

Vector Informatik India Prv. Ltd., Pune, India, [www.vector.in](http://www.vector.in)

**E-Mail Contact**

[info@vector.com](mailto:info@vector.com)