

# With Measurements to Real-Time Simulation

**Dipl.-Ing. Tapio Kramer**

INCHRON GmbH, Lichtenbergstr. 8, 85748 Garching b. M., Germany  
+49 89 5484-2960, [Kramer@INCHRON.com](mailto:Kramer@INCHRON.com)

**Dr. Ralf Münzenberger**

INCHRON GmbH, August-Bebel-Str. 88, 14482 Potsdam, Germany  
+49 331 97992-232, [Muenzenberger@INCHRON.com](mailto:Muenzenberger@INCHRON.com)



## Abstract

The components of (networked) embedded systems heavily influence each other's real-time behavior that therefore is hard to predict and test. Nonetheless it can cause critical errors which until now could only be found very late in the development process. The paper describes a straight forward approach to build a simulation model that enables system architects, developers and integrators to analyze the real-time behavior very efficiently.

A methodology is described how to use timing measurements from a prototype system to build a real-time simulation model. The model will be executed by the real-time simulator chronSim to provide an efficient platform for extensive real-time testing and debugging. Using the simulation models called 'Task-Models' to easily perform sensitivity analysis, they help to design robust and reliable embedded systems. The impact of planned changes can be evaluated upfront.

## 1. Motivation

Once an embedded system under development has real-time problems the motivation to use a fast and focused debugging method is obvious. The real-time simulation models provide the proper platform to fix these real-time problems efficiently. And besides this 'firefighting task' late in the development process, there is the motivation to design high quality products with proven robustness using real-time simulation during development.

The task is to analyze the real-time behavior of the system at a time, where prototypes are already available but changes still possible. From the functional description and its implementation the execution logic and dependencies of the system components are known. The measured trace data of the prototype provides the necessary components' execution times and activation statistics. Combining this information in a Task-Model and running it on a simulator allows exploring various system states to identify real-time critical situations.

The real-time requirements of a system strongly depend on the application and the developers focuses. Designing control algorithms the jitter of task-to-task activations might lower the control loop's quality. Concentrating on communication with other systems an end-to-end timing from sending to receiving application over a bus might be the key. In unstable systems the cause might be seldom extreme long task executions that do not complete before the same task was triggered once more (multi activation).

## 2. Real-Time Challenges

Solving real-time problems in embedded systems with growing functionality and interaction with other devices becomes more and more complex and expensive. Raising the product quality by making such problems rare requires a robust and reliable architecture. Thorough testing of the implemented systems in realistic situations by experienced personnel supports these quality efforts. But tests and measurements with prototypes can't do the job solely. To test the dynamic real-time behavior all components have to be available or simulated in real-time using expensive hardware. Most measurement methods require instrumentation of the unit under test with additional software, that itself changes the timing. Since embedded systems by design often have limited resources, the measurements can cover only short time slices in certain system states. And when one has found a problem all changes to the system take time and money. Performing what-if-analysis to evaluate solutions or extensions are very costly.

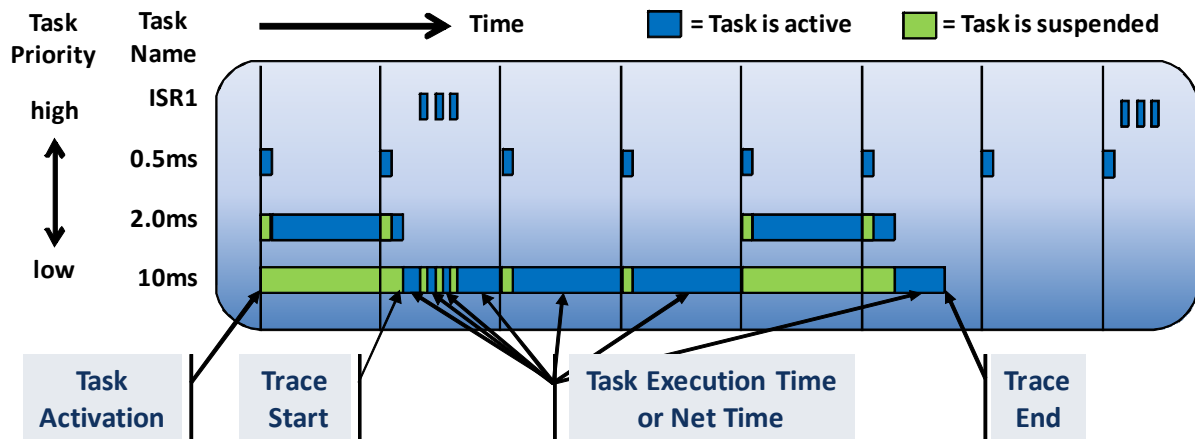
Modeling and simulation is with no doubt the best practice to master such complex systems' development projects. Simulating the real-time behavior allows a wide range of tests focused on the real-time aspects. The models can run over long time, cover multiple system states and will show the reaction to various stimuli that are hard to reproduce on a Hardware in the Loop System (HiL). Finding rare real-time errors, optimizing the performance and predicting the effect of planned changes to the system is easier with real-time simulation.

## 3. Data Required for Real-Time Simulation

To model an embedded system focusing the real-time behavior the hard- and software architecture has to be described. The net execution times of software components and the scheduling method of concurring processes are necessary. Stimuli to the system that trigger the dynamic reactions like I/O interfaces or bus communication may be provided in standard files or defined within the simulation tool. Depending on the chosen abstraction level the modeling of the functionality, and therefore the provided functional details, can be reduced to only those aspects that significantly affect the timing.

For the correct simulation the net execution times are required. By adding commands to the source code at critical points (e.g. start and end of software modules) the system behavior can be traced with timestamps to identify what system state occurred when. The drawback of this monitoring technique is twofold: – the trace commands add code to be executed and therefore change the real-time behavior of the system. – and since the memory is limited the trace will cover only short time periods showing only single system situations. Nevertheless will the trace provide realistic gross execution times of software modules in different system states. The net execution times can be derived later utilizing additional scheduling information.

Figure 1 depicts the task states over a span of time. All start and end points were derived from a system trace, where every state change was logged with its time stamp into a log file. Since higher priority tasks (upper rows) can suspend lower priority tasks (lower rows), the gross execution time (blue and green blocks) from trace start until trace end might be larger than the net execution time (blue blocks only). Depending on the occurred suspensions a low priority task might take very long to complete, still having had only short active execution times. In the shown graph the 10ms task needs nearly 3ms to complete having a net execution time of 1.5ms only.



**Figure 1: Gross and net execution times derived from trace data**

All software modules (from application and basic software) are scheduled by the OS using a predefined scheduling scheme bundling functionality into different tasks with different priorities. In this case the tasks are triggered periodically in different millisecond periods. Interrupts naturally can occur asynchronously and might even have application specific periods interfering with the OS schedule. In the example in Figure 1 the ISR1 could delay the 0.5ms task completion significantly if it occurred at the same point in time. This scheduling and software architecture is defined and provided to model the system with standard OS configuration files.

The hardware architecture including the different sources to stimulate the system can be added to the real-time model through a graphical configuration interface based on library components of peripheral and microcontroller components as part of the simulator tool.

#### 4. Modeling

With the data described in chapter 3 a real-time simulation model (Task-Model) can be generated with low effort. The real-time simulator chronSim then executes the Task-Model and simulates the scheduling of the tasks including their suspension and interrupts. The modeled system responds dynamically to the defined stimuli and generates similar (but more detailed) trace data that can be compared with traces from the prototypes.

The hardware architecture is entered into the simulator using graphical user interfaces. From a set of library components the microcontrollers, peripheral components and interfaces (I/O, memory, buses, etc) are selected and logically interconnected using memory addresses. Parts of the software architecture are modeled using the simulator tool. The application model is added as C code to enable reuse of existing target code (see Figure 2). The chosen scheduling method as well as the definition of the tasks and their priorities can be entered into the simulator environment using dialogs or by importing standard OS configuration files. In this example an OiL-File (OSEK Implementation Language) was used.

To model the application correctly the net execution time of tasks and functions has to be used since the gross time would include static suspension times that shall now be generated dynamically by the simulator instead. With the knowledge about the scheduling one could even manually analyze the recorded trace data to gather the net execution times of the different tasks. See e.g. in Figure 1: knowing that the 0.5ms task has a higher priority than the 2ms task and both being triggered simultaneously, it is clear, that the execution of the 0.5ms task will interrupt the 2ms task and has to be subtracted to get the net execution time of the 2ms task. But the suspension by ISR1 depends on the asynchronous stimulation by the interrupts.

Figure 2 shows an example source code on the left and the resulting C model on the right side. This C model is called in chronSim 'Task-Model' [1]. The net execution times are inserted as macros 'DELAY' into the Task-Model representing the CPU time consumed by the according functions, ISRs or tasks. To generate easy comparable trace entries in the simulation trace, the print commands in the source code can be replaced by 'EVENT' macros in the Task-model. The difference is, that these event macros do not distort the timing as the print commands do when running on the real prototype hardware.

The stimulation of the modeled ECU is also configured using the graphical user interface, added as C code to the simulation model or can be imported from standard files like csv. Additional stimulation from a bus communication may be added by using the virtual restbus simulation module by import of Fibex or CANdb files. The simulator generates the defined bus traffic and stimulates the model accordingly.

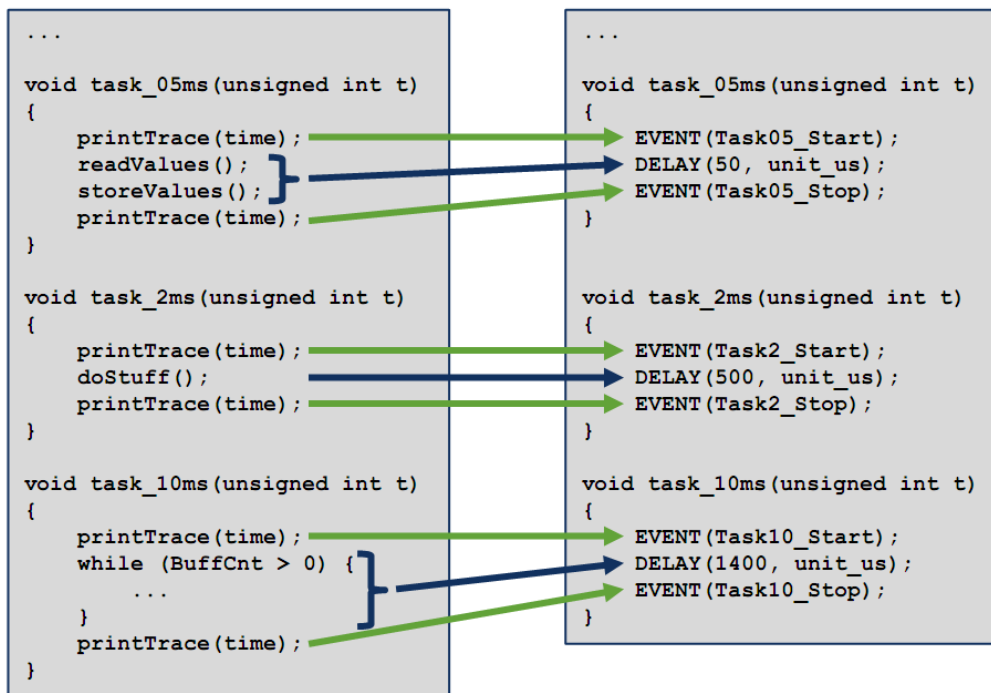


Figure 2: Source code conversion to Task-Models

## 5. Analysis

When analyzing the collected data with tools like Excel™ some simple results can be achieved. A profound information on the systems' robustness will not be gained. For example a static analysis of the systems real-time capabilities could be performed manually. All net execution times multiplied with their period and summed up certainly shall never exceed 100% CPU time. But how can the developer see what the system behavior is after running for a while, having drifting interrupt sources and maybe alternating execution times interfering with these interrupts in seldom phases? The real-time model allows monitoring the system states over a long time period and easily applying changes to do sensitivity analysis. With versatile graphical displays the user can monitor the system states and understand what the root cause of periodic CPU overloading is or where seemingly sporadic task delays are initiated.

Using the simulation the system's dynamics are easier to understand. The scheduling and the resulting system behavior become clear. The freely selectable abstraction level helps to concentrate on the timing aspects rather than getting lost in functional details.

For an in depth robustness analysis it is necessary to find the weak spots, where the system is sensitive to changes. Using Task-Models the real-time relevant factors can be easily varied to experience the limitations of the system's real-time capabilities. – Changing the execution times by adding a global factor to the DELAY commands. – Adding more stimuli by increasing the interrupt rates and messages on the busses. – Applying drift and jitter to the clocks and time triggered stimuli. Performing these tests on real hard- and software would cost immense amounts of time and money.

It is not sufficient just to use worst-case timing analysis to find possible system failures. As soon as parallel, unsynchronized processes run or multiple processors are involved worst-case analysis will miss critical system situations. E.g. having one process being executed in best case time and the other in worst case time a race condition can occur, that cannot be found using only best or worst-case timing. Sometimes even alternating execution times (due to functions called every second time the task is executed) can make a system toggle between two extreme execution paths causing hard to find behavior.

In addition the real-time simulation allows an optimization on various aspects like memory need, CPU power or expandability. If the gathered timing data is detailed down to single functions within the tasks it is even possible to evaluate how remapping of functions will affect the system.

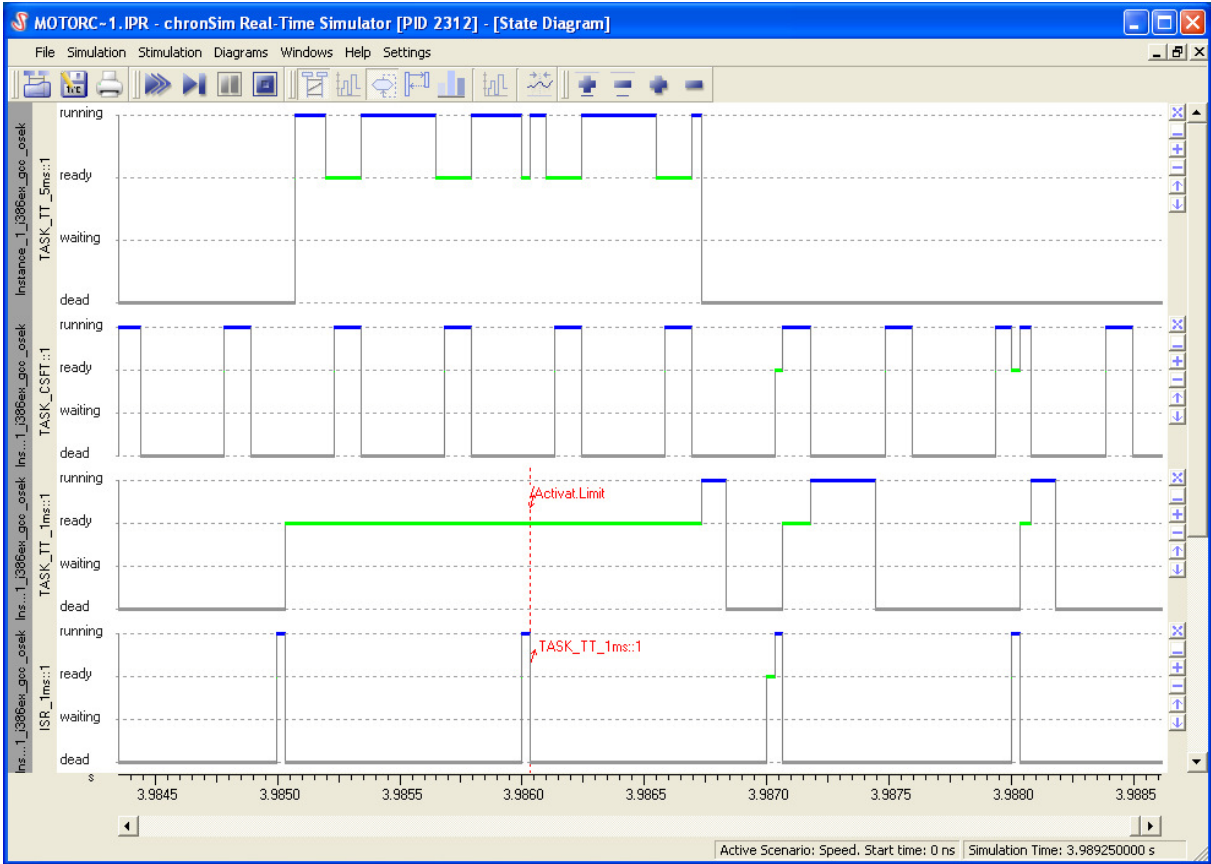


Figure 3: Task state diagram with multiple activation problem

Figure 3 shows how task states and errors are highlighted in the simulator. The task states of three tasks and one ISR over a time span of about 4ms are plotted here. Blue lines represent

active and green lines represent suspended tasks. The ISR\_1ms triggers TASK\_TT\_1ms every millisecond. But since TASK\_TT\_5ms and TASK\_CSFT have a higher priority, the TASK\_TT\_1ms has not completed before it is triggered again at 3.9860s. It had an activation limit of one activation at a time configured in the OIL-File and the simulator highlights this OS error state.

## 6. Simulation of Networked Systems

Automotive ECUs are more and more bearing multiple functions and applications being executed in a networked, interdependent environment. The ECU in this case was communicating with other nodes over the widely used CAN (Controller Area Network). Here the communication is more or less 'on demand' and event driven. Every node has it's own communication schedule and can access the bus at every time. This results in a bus load that varies significantly and may even exceed 100% for long periods e.g. when the attached nodes all start sending at the same time at startup. So the communication stack and software architecture of the ECU has to cope with high and unpredictable bus loads but can run asynchronously.

In a later model the ECU shall be changed using the bus system FlexRay [2]. Here all communication is organized in fixed time slots using a global time base to synchronize all participants. This modification of the ECU leads to the significant change to a time triggered communication method and requires a review of the complete software architecture. Since the application itself will not change it is possible to reuse the gathered execution times and easily adapt the model to simulate the planned synchronous communication over FlexRay. With slight changes and even before any hard- and software is designed it is possible to predict the impact of the new bus schedule on the ECU. An architect can evaluate different software architectures to optimize the system for FlexRay in a very early phase [3]. He can avoid the chicken and egg dilemma having to wait for test results from prototypes running on test benches to define the software architecture best suited for FlexRay.



Figure 4: Task states of multiple ECUs combined with CAN and FlexRay communication

Figure 4 shows some task states (upper four rows) of different ECUs (automotive Electronic Control Units) together with the CAN- (5<sup>th</sup> and 6<sup>th</sup> row) and FlexRay-communication (lower two rows) on one time scale. Using these diagrams the timely relation of state changes on different ECUs (e.g. of tasks sending and receiving messages on the bus) can be related to communication on the bus. An end-to-end timing can be observed including delays due to drift and jitter of the involved, asynchronous timing sources.

## **Conclusion**

Modeling the real-time behavior of embedded systems with Task-Models allows efficient analysis and tests. The robustness of the system will become clear having performed sensitivity analysis covering a broad range of system states. Once problems are found, what-if-analysis methods using the real-time simulation offer an efficient, fast way to solve them.

Real-Time simulation using chronSim Task-Models can be done even in late development phases with very low efforts. Even though the trace data may only cover certain situations, a sensitivity analysis will point to those corner cases, where an additional testing with prototypes is indicated.

The earlier timing requirements are defined and tested, the fewer real-time problems will have to be solved in late, expensive development phases. Task-Models can already be used by system architects to design robust and reliable embedded systems.

## **References**

- [1] T. Komarek, M. Dörfel, R. Münzenberger; Developing Real-Time Constrained Embedded Software Using Task-Models; In proceedings of the Advanced Automotiv Electronics aae2007 Gaydon; [www.aae-show.co.uk](http://www.aae-show.co.uk)
- [2] T. Kramer; Wechsel von CAN zu FlexRay; AUTOMOBIL-ELEKTRONIK, electronica – Sonderausgabe 2008
- [3] R. Münzenberger, M. Dörfel, C. Diederichs, U. Margull, G. Wirrer; Entwurf echtzeitfähiger Steuergerätesoftware in FlexRay-Netzwerken; In proceedings of the KfZ-Entwicklerforum 2007, Ludwigsburg