

Collaboration Support to Master Real-Time Challenges

Dr. Ralf Münzenberger

INCHRON GmbH, August-Bebel-Str. 88, 14482 Potsdam, Germany
+49 331 97992-232, Muenzenberger@INCHRON.com

Dipl.-Ing. Tapio Kramer

INCHRON GmbH, Lichtenbergstr. 8, 85748 Garching b. M., Germany
+49 89 5484-2960, Kramer@INCHRON.com



Abstract

Embedded systems often (e.g. in automobiles) are widely networked and run distributed applications on multiple microcontrollers. More and more the interconnected nodes are developed by different suppliers. Even for a single device multiple, separated development groups provide basic software, drivers, communication stacks or application. This common, but not simple development setup inevitably needs collaboration support.

The collaboration in such large development teams benefits fundamentally from the precise definition and communication of requirements. For functional aspects of embedded systems this is supported by profound methods and tools. When it comes to real-time performance and reliability aspects the tool support is very limited until now.

The paper describes a methodology to define and exchange the timing specification and requirements of embedded systems by using simulation models, so called 'Task-Models' – the collaboration becomes based on a real-time specification. The evaluation of real-time requirements is done with the real-time simulator chronSim. From early design to late implementation and test phases, the Task-Models describe the requirements and properties with adaptable abstraction levels. Using Task-Models each development team and project partner can communicate their view on the timing aspects with simulation models and their analysis' results. Without having to implement all SW on targets or disclose all details (IP protection), a common understanding of timing requirements and properties can be discussed and exchanged across team or even company borders.

1. Good specification is key

'Think First' is a well known design rule that always has its importance even in large, experienced development teams. One implementation of this rule is to begin every project with a thorough requirements gathering followed by putting down detailed specifications. This method is common knowledge and well supported by processes and tools in all abstraction levels – with one exception: real-time requirements.

The second as a fundamental constant might be the best defined SI unit. On the contrary timing requirements are often described using fuzzy phrases like 'fast', 'ASAP', 'after' or not at all. Timing has the nature to be hard to specify for humans having no adequate intuitive measuring method (unlike 'Foot'). Therefore the common way to describe a required functionality is to define the state before and after a required action. Sequences of events and actions are defined in chains and the progress depends on state changes – no time span or relation is mentioned. And when independent processes run in parallel it is therefore difficult to get them both related in one context, on one time scale.

Still, defining timing specifications in early phases is very important – even if the execution times of tasks or software modules are just a rule of thumb derived from earlier projects. Once the timing aspect is mentioned it will be part of the weighing of probably opposing design goals. If timing aspects would be involved as soon as they can be measured, the project will be so advanced that changes due to timing problems will be very costly. [2]

2. Real-Time specification in early phases

The real-time behavior of embedded systems is dependent on the chosen microcontrollers, peripheral components, the interaction with the outside world and significantly on the software. When one component interacts with another its reaction time has to be known upfront. This leads to the chicken and egg dilemma having to specify the timing of soft- and hardware before it was built to experience their timing.

Designing a system from the scratch the architect has to set certain cornerstones defining the general functionality and timing of the system. At this point he chooses what will be executed in parallel or in sequence. The reactions of the planned system to different stimuli scenarios is easy to define for functional aspects, where ‘walking’ through a static flow chart can be done without considering any timing aspects. But to design a robust and reliable system the timing aspects have to be taken into account.

To understand the dynamic reaction of the system a first simulation of the timing will help significantly. (see Figure 1) Parallel to the flow charts for functional description a simulation model of the real-time behavior is developed. For every function or task timing quotas are set. In the model and the simulation platform the scheduling of concurrent functions is defined.

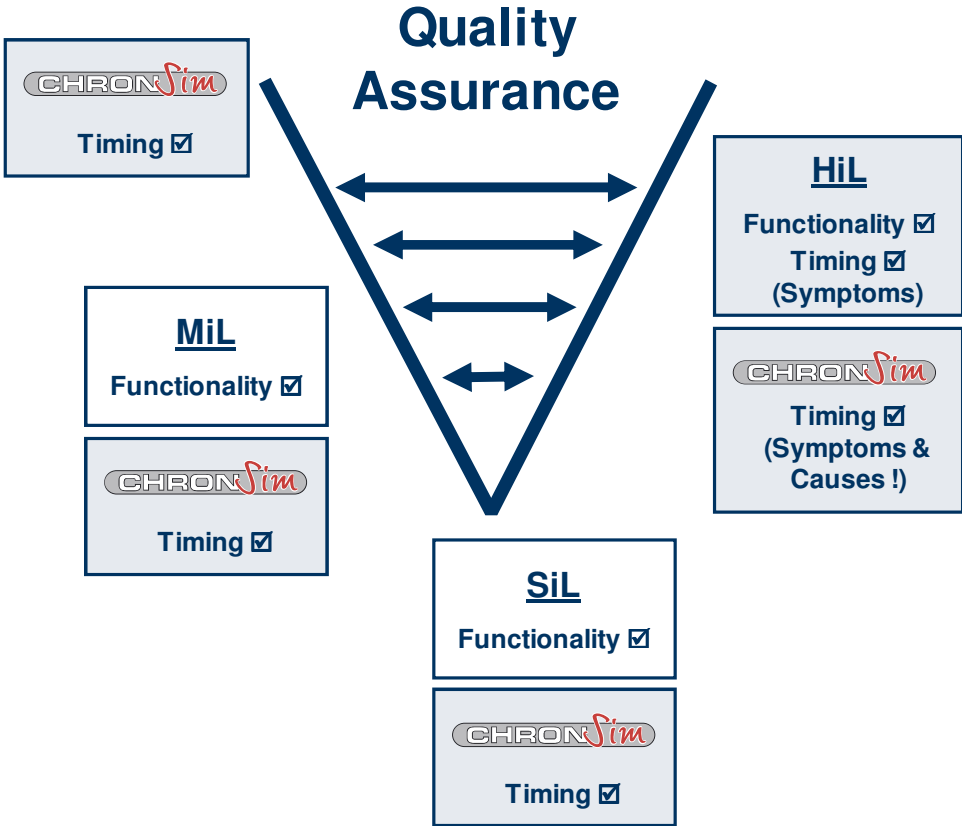


Figure 1: Quality assurance during development processes

Figure 1 shows the commonly used V-model where timing aspects are considered very late in the development process. Model and Software in the Loop (MiL and SiL) allow testing the system focusing functional requirements only, since a hardware related execution is not included yet. Therefore real-time behavior becomes available late, when an evaluation or prototype hardware is used to execute the implemented software on a Hardware in the Loop (HiL) system. Expensive iterations within parts of the V or even over the complete V-model are needed to improve the timing aspects, if no real-time simulation is available. By modeling and simulating the real-time behavior, best if done at the project start, it is possible to shorten or even avoid costly iterations.

3. Designing distributed, networked systems requires good collaboration

Considering the timing aspects already in early phases is important for individual embedded system development projects. When the desired functionality will be performed by a distributed, networked system it becomes unavoidable. In modern automobiles the embedded systems are networked over deterministic bus systems.

The functionality is defined by the OEM as well as the suppliers. Often the software is developed by multiple parties including the OEM. Depending on their individual focus the OEM, supplier and 'sub-supplier's' have different views on the systems' timing requirements. Nevertheless each party's timing component has an impact on the other party's development. When e.g. the OEM chooses a time triggered bus for communication, the supplier has to architect the embedded system to be synchronized to the bus. And the function developer has to consider the effects of an external timing to his e.g. control loop with an own cycle time, that might then show interferences.

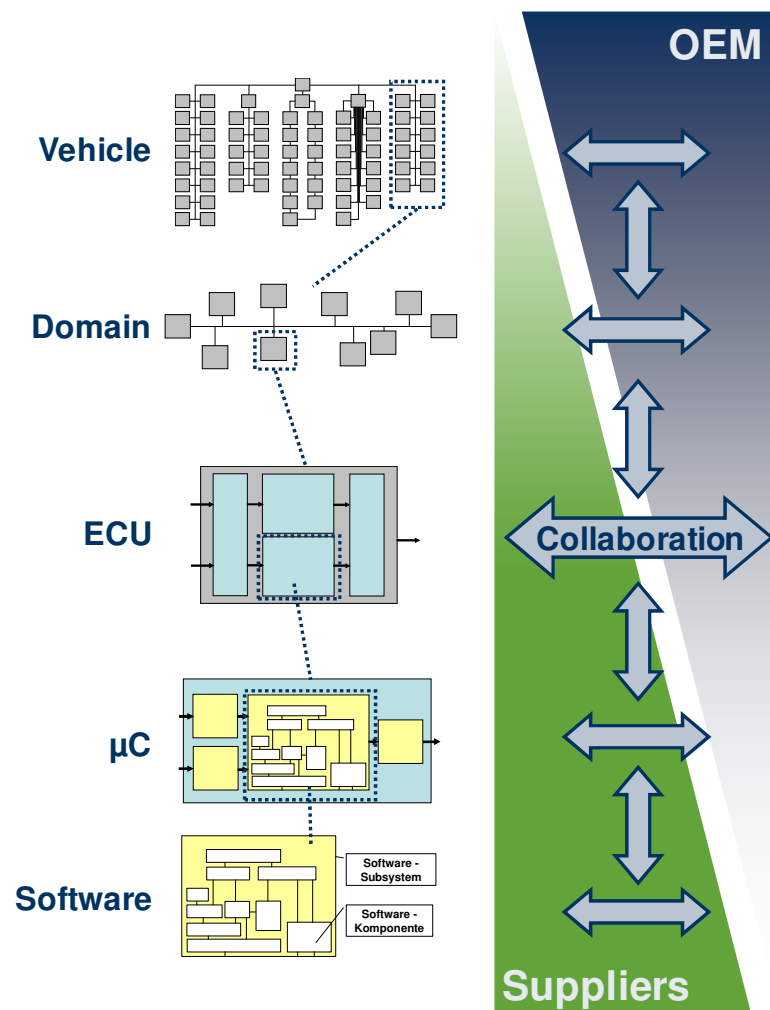


Figure 2: Collaboration across system levels and development partners

Figure 2 depicts the various levels of electronic equipment in modern cars. The OEM is integrating all components from his suppliers, therefore has a focus on the higher, networked levels. He defines the communication matrix and partly how the functions are distributed to the embedded systems. The tier 1 suppliers deliver complete sub-systems sometimes including multiple microcontrollers. Their focus might be the optimal balance between cost and performance using different microcontrollers. Reuse of existing developments and the distribution of function to the computing nodes are also in their focus. Finally the software is written by the supplier, sub-suppliers and, with growing proportion, the OEM.

A good collaboration between OEM and suppliers, between architects, integrators and testers as well as between all system levels is obviously the key factor for a well integrated, optimized networked embedded system.

4. Real-Time simulation models

There is no doubt, that good collaboration requires exchange of information and a common platform to describe the desired design goals. Requirement management tools provide the databases as an exchange platform, but no data contents about the project. Text documents, flow charts, UML- and SimulinkTM-models describe the functional aspects but not the timing behavior considering the soft- and hardware architecture.

Real-Time requirements covering all system levels have so far not been standardized or made widely available. For a complete system description and base for collaboration there was no versatile tool or data format in use until now. Task-Models can close that gap and enable embedded system developers to cover also the timing aspects when working together in a design project.

A part of the Task-Model consists of model code in standard C. Here the execution times of tasks, functions and software modules are entered. It can be derived by abstracting the target C code and annotating it with the target execution time on the level of ISRs, tasks, functions or other software modules. Or it can be the starting point of a project having a skeleton code describing the software structure. Throughout the development it will become more detailed and finally may even be replaced by the target code itself. [1]

Figure 3 lists on the left side a simple Task-Model for an OSEK OS. The macro 'DELAY' replaces the function code with its time budget. So the ISR will run for 30 microseconds and then use an OS function call to have the scheduler activate the task. This will run for 150 microseconds as long as it is not suspended by a higher priority task. If the timing behavior of the modeled system has to be defined more in detail, there are all possibilities, that C offers available. The right example lists several methods to have the execution time vary depending on system states or other stimuli. This enables dynamic reactions of the simulation model showing the same timing as the real system.

```

ISR(ISR_Rotation)
{
    DELAY(30, unit_us);
    ActivateTask(Task_Rot);
}

TASK(TASK_Rot)
{
    DELAY(150, unit_us);
    TerminateTask();
}

```

Example of the most simple implementation of a Task Model for OSEK.

The ISR_Rotation is executed in 30 us and Task_Rot in 150 us.

```

TASK(TASK_TT_5ms) {
    DELAY(300, unit_us); ①
    Schedule(); ②
    while(!finished()) {
        DELAY(100, unit_us); } ③
    exectime = 10*data_size; } ④
    DELAY(exectime, unit_us);
    DELAY(gaussian(500, 10), unit_us); ⑤
    TerminateTask();
}

```

Example of a more complex Task Model

- ① : constant execution time
- ② : scheduling point
- ③ : control flow dependent execution time
- ④ : data dependent execution time
- ⑤ : probability distribution of execution time

Figure 3: Simple and versatile Task-Models

To complete the system description the hardware, peripherals and stimuli are described using graphical user interfaces and libraries provided by the simulator tool. The scheduling method and software architecture with the definition of processes, priorities and e.g. stack sizes can be entered as well or be imported from standard OS configuration files like Oil (OSEK Implementation Language).

Task-Models can be adapted to the abstraction level, the development phase, the focused timing detail of all involved development partners. According to the system levels in Figure 2 a network designer can model the communication across multiple nodes without having to have detailed models of the nodes' internals. Accordingly the developer of an application part within one node, can model his parts very detailed. But he reduces the communication details and peripheral timing behavior models to simple stimuli sources as far as they influence his applications timing.

The real-time simulator chronSim executes the Task-Model and schedules the tasks and functions according to the configured system architecture. Low priority processes become suspended by higher priority processes. The system reacts dynamically to stimuli, interrupts and bus communication. Through execution of the timing model the system states can be visualized and timing aspects become clear.

5. Collaboration with IP protection

Disclosing all information to the partners in a joint development might be ideal to support a good collaboration. But this is not possible in the real world. Partners in one project are competitors in others. Security algorithm details cannot be divulged to 3rd parties. Having methods to conceal individual know how is therefore a must. With Task-Models IP protection can perfectly be done because details of the algorithm can be disclosed on the needed level of detail.

And not only the IP protection is a good reason to reduce the detail level provided to the partners. Having a method to simplify and abstract the data to focus the required aspects avoids unneeded, time consuming complexity. Why should a real-time model of a communication bus simulate every application function, when the dynamic bus load is currently analyzed?

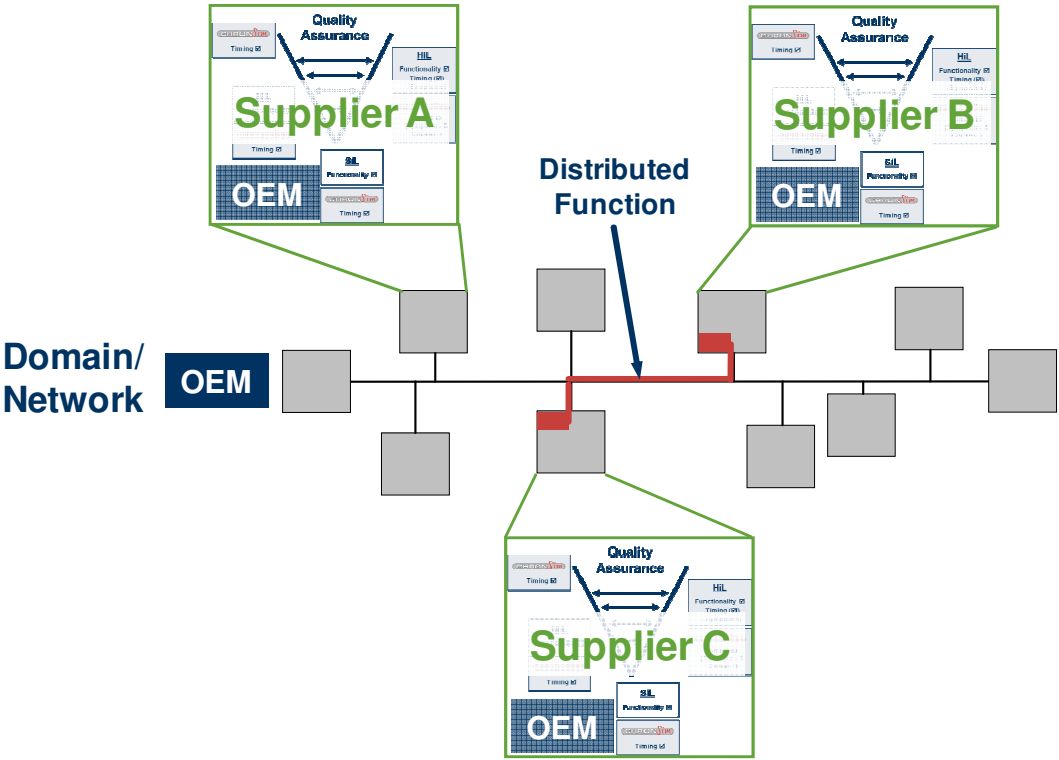


Figure 4: Distributed function on a networked platform

Describing the timing of a system using Task-Models allows to share an executable specification of the dynamic system without having to provide the source code. The DELAY macros replace the actual function code and do represent its correct timing behavior. Depending on the chosen detail level a DELAY can substitute a simple function call or a large application with complex internal structure (see Figure 3). And if the architect has to design a system where some components haven't been programmed yet, the DELAY represents the timing budget being assigned for that code.

Figure 4 highlights how the parties are involved when developing distributed functions. The OEM has requested a function (red boxes and lines) with code running on two suppliers' systems (grey boxes). To design and test the real-time capabilities every of the three parties has to have a good knowledge of the other partners' parts. When supplier B and C provide Task-Models to describe their embedded systems, the OEM can use these timing models together with his bus communication definition to integrate and test the complete function in the network – even before the suppliers have delivered hard- and software prototypes. Each supplier can use the provided Task-Models from the OEM and other partners to test his system. Internally he might use a very detailed model of his system supporting a detailed debugging and system knowledge. The model provided to the partners can be more abstract to protect the IP.

6. Conclusion

Good collaboration is a key factor for success when designing complex distributed embedded systems with multiple development parties. And a key component of this collaboration is the exchange of detailed timing requirements and specifications.

The real-time simulator chronSim provides with its Task-Models a platform to describe and analyze the timing of embedded systems with simulation models. The abstraction level can be chosen according to the designers focus or to protect each party's IP.

Using Task-Models the embedded system's timing can be analyzed and exchanged across all development groups. Distributed functions running on multiple networked nodes can be simulated within one environment, designed and tested to gain a high quality robust system.

References

- [1] T. Komarek, M. Dörfel, R. Münzenberger; Developing Real-Time Constrained Embedded Software Using Task Models; aae2007 Gaydon; www.aae-show.co.uk
- [2] T. Kramer; Die Echtzeit rechtzeitig beherrschen; Hanser Automotive 11 2008