Managing Real-Time Risks in the Development of Safety-Critical Embedded Systems

2nd International CTI ISO 26262 Conference, 14-16 May, Troy, Michigan/USA

Dr. Ralf Münzenberger – Managing Director Professional Services
Mesut Özhan – Professional Services Engineer
Managing the real-time dynamics of safety-critical applications in today’s automotive industry is an increasingly challenging task for system engineers. With growing complexity and a high degree of integration along with the pressure to reduce time-to-market and unit costs, it becomes literally impossible to eliminate all possible failures before start of production just by testing. However, typical real-time failures, e.g. end-to-end response time violation or asynchronicity of data, can already be addressed in early design stages using model-based analysis and simulation techniques. As a provider of solutions for the development of embedded real-time systems and based on our substantial experiences from various customer projects in the automotive domain, we present in this talk, how state-of-the-art methods and tools can be applied in compliance with ISO 26262 requirements, in order to find and eliminate real-time issues and to evaluate and assess the real-time characteristics of a system design.
INCHRON GmbH

Company History
1996: Start of fundamental research
2003: INCHRON founded
2006: Investment from Hasso Plattner Ventures
2009: Premium Member AUTOSAR
2011: Global Reseller Agreement IBM Rational

Partners
IBM

Customers

Memberships
AUTOSAR
TIMM02use
A view on automotive E/E development

Time to Market

Cost of Quality

Market Requirements
Functions & Interconnections in Global Markets

Technology Trends
>70 ECU
Domain Controllers
Multi-Core

Standards

Obstacles
Geographic
Organization
Infrastructure

Research | Pre-Development | Tender | A | B | C | SOP
Specifications and complexity

Predominant Factors Causing Projects to Be Late

- Unrealistic schedules: 63.6%
- Changes in specifications: 58.8%
- Inadequate specifications: 41.7%
- Complexity of the application: 37.0%
- Too few developers/testing personnel: 36.1%
- Poor management oversight: 21.7%
- Poor development/testing tools: 13.5%
- Inefficient design/production: 11.5%
- Other: 3.5%

Addressed by INCHRON solutions

A V-model perspective on timing

Timing Requirement: $\Delta t \leq 300$ ms

Signal detection → Pre-processing → Object Verification → Tracking → Decision → Braking

Detection of timing errors

Active Limit

Implementation

Specification

Design

Integration & Test

Pre-processing → Tracking → Decision → Braking

CPU

FPGA

IO

© INCHRON GmbH 2013
Typical timing errors and their causes

A deadline violation occurs: T_10ms is activated before a previous activation has terminated.

<table>
<thead>
<tr>
<th>Processes</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Active at Boot</td>
<td>Priority</td>
<td>Preemptable</td>
</tr>
<tr>
<td>T_20ms</td>
<td>✔️</td>
<td>10</td>
<td>✔️</td>
</tr>
<tr>
<td>T_10ms</td>
<td>✔️</td>
<td>5</td>
<td>✔️</td>
</tr>
<tr>
<td>T_05ms</td>
<td>✔️</td>
<td>20</td>
<td>✔️</td>
</tr>
</tbody>
</table>

T_20ms has a higher priority than T_10ms.
Typical timing errors and their causes

A deadline violation occurs, this time for the task T_20ms.

This time T_20ms has the lowest priority, but T_10ms demands too much CPU time.
Another deadline violation for the task T_20ms.

This time the error is caused by a random interrupt burst.
In this data flow between two CPUs via a CAN bus, reuse and loss of data occurs.

The jitter of the task T_20ms on CPU1 that sends the CAN message and the fact, that sending is done at the end, seem to cause the data inconsistencies.
Typical timing errors and their causes

Data from different sources becomes out of sync

The time bases of the two sending controllers drift apart, e.g. due to unsynchronized clocks.
Didn’t expect this at all!!!

Deadlock

Image courtesy of [http://www.glommer.net/blogs/?p=189](http://www.glommer.net/blogs/?p=189)
Timing errors originating from ...

<table>
<thead>
<tr>
<th>Phase / Scope</th>
<th>System</th>
<th>Environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specification</td>
<td>Faulty requirements</td>
<td>Faulty assumptions</td>
</tr>
<tr>
<td>Design / Implementation</td>
<td>Faulty system (model)</td>
<td>Faulty environment (model)</td>
</tr>
<tr>
<td>Runtime</td>
<td>Random malfunctions</td>
<td>Unexpected environment conditions</td>
</tr>
</tbody>
</table>

... is this about safety?
Timing requirement perspectives

Timing requirements determined by safety analysis

- Fault (latent Error)
- Error (detected)
- (System) Failure
- Fault Reaction Time
- Fault Tolerant Time Interval

Propagation into other Systems

Safe State

Hazard

Timing requirements determined by functional analysis

Timing Requirement: $\Delta t \leq 300$ ms
Integrating safety mechanisms into the system

**Monitoring**
- Data flow
- Control flow
- Execution / Response time

**Error Detection**
- Plausibility
- Indicate error state

**Error Handling**
- Switch operation mode
- Shutdown system
- Ignore?!

Timing Requirement: $\Delta t \leq 300$ ms
Integrated perspective on timing and safety

Timing Requirement: $\Delta t \leq 300$ ms

Specification

Detection of timing errors

Design

Integration & Test

Implementation

© INCHRON GmbH 2013
Integrated perspective on timing and safety

Model-based analysis and simulation

Timing Requirement: $\Delta t \leq 300$ ms

Specification

Detection of timing errors

Design

Brainstorming

Implementation

Integration & Test

© INCHRON GmbH 2013

Timing Requirement: $\Delta t \leq 300$ ms
Virtual integration
Real-time criteria

- Specification
  - Bus / CPU load
  - Response times
  - Activation limits
  - Start-to-start jitter
  - Event chains
    - End-to-end latency
    - Data age
    - Data loss
    - Reuse of data
  - Execution order
  - Loss / Blocking of IRQs
  - Data consistency

- Virtual Integration
WCRT analysis and RT simulation

Lowest response time in simulation

Largest response time in simulation

Best-case response time from analysis

Worst-case response time from analysis
ISO 26262 compliant processes

<table>
<thead>
<tr>
<th>3. Concept phase</th>
<th>4. Product development at the system level</th>
<th>5. Product development at the hardware level</th>
<th>7. Production and operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-5 Item definition</td>
<td>4-5 Initiation of product development at the system level</td>
<td>5-5 Initiation of product development at the hardware level</td>
<td>7-5 Production</td>
</tr>
<tr>
<td>3-6 Initiation of the safety lifecycle</td>
<td>4-6 Specification of the technical safety requirements</td>
<td>5-6 Specification of hardware safety requirements</td>
<td>7-6 Operation, service (maintenance and repair), and decommissioning</td>
</tr>
<tr>
<td>3-7 Hazard analysis and risk assessment</td>
<td>4-7 System design</td>
<td>5-7 Hardware design</td>
<td></td>
</tr>
<tr>
<td>3-8 Functional safety concept</td>
<td>4-8 Item integration and testing</td>
<td>5-8 Evaluation of the hardware architectural metrics</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>5-9 Evaluation of the safety goal violations due to random hardware failures</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>5-10 Hardware integration and testing</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>6-5 Initiation of product development at the software level</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>6-6 Software architectural design</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>6-7 Software unit design and implementation</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>6-8 Software unit testing</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>6-9 Software unit testing</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>6-10 Software integration and testing</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>6-11 Verification of software safety requirements</td>
<td></td>
</tr>
</tbody>
</table>
# ISO 26262 part 4 recommends simulation

## Table 3 — System design verification

<table>
<thead>
<tr>
<th>Methods</th>
<th>ASIL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td>1a System design inspection&lt;sup&gt;a&lt;/sup&gt;</td>
<td>++</td>
</tr>
<tr>
<td>1b System design walkthrough&lt;sup&gt;a&lt;/sup&gt;</td>
<td>++</td>
</tr>
<tr>
<td>2a Simulation&lt;sup&gt;b&lt;/sup&gt;</td>
<td>+</td>
</tr>
<tr>
<td>2b System prototyping and vehicle tests&lt;sup&gt;b&lt;/sup&gt;</td>
<td>+</td>
</tr>
<tr>
<td>3 System design analyses&lt;sup&gt;c&lt;/sup&gt;</td>
<td></td>
</tr>
</tbody>
</table>

A: see Table 1

---

<sup>a</sup> Methods 1a and 1b serve as a check of complete and correct implementation of the technical safety requirements.

<sup>b</sup> Methods 2a and 2b can be used advantageously as a fault injection technique.

<sup>c</sup> For conducting safety analyses, see ISO 26262-9:2011, Clause 8.
ISO 26262 part 6 recommends simulation

Table 6 — Methods for the verification of the software architectural design

<table>
<thead>
<tr>
<th>Methods</th>
<th>ASIL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td>1a Walk-through of the design&lt;sup&gt;a&lt;/sup&gt;</td>
<td>++</td>
</tr>
<tr>
<td>1b Inspection of the design&lt;sup&gt;a&lt;/sup&gt;</td>
<td>+</td>
</tr>
<tr>
<td>1c Simulation of dynamic parts of the design&lt;sup&gt;b&lt;/sup&gt;</td>
<td>+</td>
</tr>
<tr>
<td>1d Prototype generation</td>
<td>0</td>
</tr>
<tr>
<td>1e Formal verification</td>
<td>0</td>
</tr>
<tr>
<td>1f Control flow analysis&lt;sup&gt;c&lt;/sup&gt;</td>
<td>+</td>
</tr>
<tr>
<td>1g Data flow analysis&lt;sup&gt;c&lt;/sup&gt;</td>
<td>+</td>
</tr>
</tbody>
</table>

<sup>a</sup> In the case of model-based development these methods can be applied to the model.

<sup>b</sup> Method 1c requires the usage of executable models for the dynamic parts of the software architecture.

<sup>c</sup> Control and data flow analysis may be limited to safety-related components and their interfaces.
What are appropriate scheduling properties?

Table 3 — Principles for software architectural design

<table>
<thead>
<tr>
<th>Methods</th>
<th>ASIL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td>1a Hierarchical structure of software components</td>
<td>++</td>
</tr>
<tr>
<td>1b Restricted size of software components</td>
<td>++</td>
</tr>
<tr>
<td>1c Restricted size of interfaces</td>
<td>+</td>
</tr>
<tr>
<td>1d High cohesion within each software component</td>
<td>+</td>
</tr>
<tr>
<td>1e Restricted coupling between software components</td>
<td>+</td>
</tr>
<tr>
<td>1f Appropriate scheduling properties</td>
<td>++</td>
</tr>
<tr>
<td>1g Restricted use of interrupts</td>
<td>+</td>
</tr>
</tbody>
</table>

D.2.2 Timing and execution

With respect to timing constraints, the effects of faults such as those listed below can be considered for the software elements executed in each software partition:

— blocking of execution;
— deadlocks;
— livelocks;
— incorrect allocation of execution time;
— incorrect synchronization between software elements.

EXAMPLE Mechanisms such as cyclic execution scheduling, fixed priority based scheduling, time triggered scheduling, monitoring of processor execution time, program sequence monitoring and arrival rate monitoring can be considered.
Table 3 — Principles for software architectural design

<table>
<thead>
<tr>
<th>Methods</th>
<th>ASIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1a Hierarchical structure of software components</td>
<td>++</td>
</tr>
<tr>
<td>1b Restricted size of software components(^a)</td>
<td>++</td>
</tr>
<tr>
<td>1c Restricted size of interfaces(^a)</td>
<td>+</td>
</tr>
<tr>
<td>1d High cohesion within each software component(^b)</td>
<td>+</td>
</tr>
<tr>
<td>1e Restricted coupling between software components(^a, b, c)</td>
<td>+</td>
</tr>
<tr>
<td>1f Appropriate scheduling properties</td>
<td>++</td>
</tr>
<tr>
<td>1g Restricted use of interrupts(^a, d)</td>
<td>+</td>
</tr>
</tbody>
</table>

\(^a\) In methods 1b, 1c, 1e and 1g "restricted" means to minimize in balance with other design considerations.

\(^b\) Methods 1d and 1e can, for example, be achieved by separation of concerns which refers to the ability to identify, encapsulate, and manipulate those parts of software that are relevant to a particular concept, goal, task, or purpose.

\(^c\) Method 1e addresses the limitation of the external coupling of software components.

\(^d\) Any interrupts used have to be priority-based.
Managing real-time risks in all development phases

- IBM Rational
- Autosar
- ARXML, OIL, FIBEX, DBC, C-Code …
- Trace Analysis, Requirements and Reporting
- WCRT and Schedulability Analysis
- Statistical Analysis
- Residual Bus Simulation
- C-Code Simulation
- Execution Time Estimation
Managing real-time risks in all development phases

- No entry hurdles
  - Import of OIL files, modeling in C
  - First usable results produced after only two days
- Very short turnaround times
- Total time effort for this project approx. 2 weeks

"The feasibility of such change requests can now be analyzed in 1/3 of the usual time. This saves time and money, allows fast feedback to the customer and gives more confidence in the modified system."

- chronSIM is a valuable tool. Without, several problems fixed would still be present in our system today.
- It took us only 10 days of training and occasional consulting to get to a very high level of expertise
- Support from INCHRON has always been excellent.
- The tool was worth the investment.
THANK YOU!