How to meet automotive OEM’s timing requirements

7th Automotive SW Developer’s conference 2016
Version 1.0

Peter Gliwa
CEO GLIWA GmbH
• Introduction

• What are the requirements regarding timing of automotive OEMs?
  – Today
  – Tomorrow

• How can the requirements regarding timing be fulfilled?

• Conclusion
Introduction
Contents

• AUTOSAR & timing basics

• Why timing analysis?

• Multi-core timing analysis aspects

• Multi-core tracing

• Conclusion
Software Development Process

Design
- Timing requirements/constraints, Timing layout

Implementation
- Mapping, OS config

Integration
- Timing debugging/optimization

Module Tests
- Profiling (CETs), timing supervision

System Tests
- Profiling (RTs, CPU-load), timing supervision

Apply your development processes and methodologies also to the timing of your software!
Tracing: End-to-end model-check

- On its way from the **mind** to the **microcontroller**, an idea can suffer from **transition-errors**.
- Tracing allows an **end-to-end model-check**.
Multi-core: different kinds of parallelism

• Application parallelism
  – Each application runs on one core only.
  – Application ≠ thread

• Function parallelism
  – Executing portions of functional related code in parallel

• Instruction parallelism
  – Processor cores have pipelines which process instruction in parallel.
Applying timing know-how

- Last year we talked about timing basics and multi-core basics.

- So let us apply this knowledge today!
Timing requirements today and tomorrow!
A good ECU’s requirements specification is the foundation for sound and safe timing.

Requirements specifications should address two topics related to timing:

- A list of timing requirements (as far as they are known)
- Requirements regarding the environment, methodologies and tools
Timing requirements

- **Known timing requirements**, e.g.
  - start-up times (presence on bus)
  - sub event chains (as part of end-to-end constraint)
  - execution orders

- **Budgets**
  - overall CPU-load
  - CETs (also for BMW containers!)

- **Format**
  - formal if possible (e.g. TIMEX, Artime, tool-specific formats) but:
  - Doors is better than nothing
Methodology requirements

- Define a process/infrastructure for maintaining timing requirements → handle upcoming timing requirements
- Specify whether scheduling simulation/analysis shall be used. How is timing set-up/planned?
- Specify minimum requirements regarding timing requirements verification
  - Specify when/how often timing requirements get verified
  - Specify the environment (in-car, lab, simulation, static analysis, …)
  - Specify the conditions (scenarios, test coverage, …)
- Specify how timing verifications and timing properties are documented.
Typically, the OEM does not require the supplier to use a certain tool. However, in some cases this might make sense.

Specify who has the tools available. In-house solutions do not help much in a shared SW development.

Specify which exchange formats and export formats have to be supported.

Example 1: OEM wanted to measure, supplier provided in-house tool. Interfaces were often subject to changes, no manuals, no dedicated support → usage at OEM very difficult.

Example 2: OEM forgot to specify they want to able to measure. So they did not get access to the existing supplier internal tools.
In 2009, BMW started collecting generic text blocks for requirements specifications
  – Ensure that the timing topic is addressed properly in future projects.
  – Reuse approaches which were successful in the past.
  – Efficiency: avoid that authors of requirements specifications have to ‘research’ timing related aspects again and again.

The result: text templates for requirements specifications.
  – Timing requirements are mostly project-specific
  – Thus, templates focus on methodologies and tools.
Templates for requirements specifications

- Excel table with text templates
  - including recommendation according to ASIL level

- Word document with templates

- Some big OEMs follow this approach already. More and more follow…
Tracing/debugging in AUTOSAR in the future

- AUTOSAR currently lacks a standardized interface to get timing data efficiently out of an ECU
  - ORTI is outdated, does not support multi-core and is not AUTOSAR
  - PreTaskHook / PostTaskHook are very inefficient and not allowed “on the road”

- In April 2016, an AUTOSAR “Concept” was initiated:

  **OS aware tracing and debugging**

  Goal: ORTI successor within AUTOSAR *plus* efficient support of instrumentation based tracing

- The more projects will **require** the support of this upcoming standard, the sooner tools and RTOSs will **support** it.
Multi-core tracing
Tracing vs. timing measurement

- Timing measurement
  - produces timing parameters ("numbers") but no traces

- Tracing
  - produces traces which can be viewed and from which timing parameters can be derived
  - different kinds exist
    - Hardware-based tracing
    - Instrumentation based tracing
    - Hybrid approaches
Hardware based tracing – basic blocks

- **What is a basic block?**
  - A basic block is a set of machine instructions, which always get executed sequentially.
  - The first instruction is the only entry point
  - The last instruction is the only exit point

- **What is a branch?** Strictly speaking: any modification of the IP (Instruction Pointer) other than IP++. Most importantly:
  - **calls** (and **returns**) and
  - **jumps**

### Basic block

```
mov.aa    a2,a15
ld.w     d15,[a15]
st.w     [+a2]4,d15
mov        d15,#1
j        .L8

.L5:
jeq       d15,#1,.L9
jeq       d15,#2,.L10
jeq       d15,#3,.L11
j        .L12

.L9:
mov.aa    a2,a15
ld.a        a4,[+a2]4
ld.w        d0,[a15]
mov        d15,#2
st.b       [a4],d0
ld.a        a2,[a2]
ld.hu       d0,[a2]
j        .L13

.L10:
mov.aa    a2,a15
ld.a        a4,[+a2]4
ld.w        d0,[a15]
mov        d15,#2
st.h       [a4],d0
ld.a        a2,[a2]
ld.hu       d0,[a2]0
call      CAN_PreWrite
```
Hardware based tracing – detailed description

• An on-chip trace unit reports each taken branch to externally connected tracing hardware.

• The external hardware stores the branch target address together with a time stamp in the trace-buffer.

• Using a disassembler and a “best guess”, the execution times of the remaining instructions are calculated. Now the instruction flow is complete.

• This is the principle. Reality is slightly more complicated.
Instrumentation-based tracing: T1

T1-HOST-SW
PC based SW tool for visualization, analysis and configuration

T1-TARGET-SW
Embedded software component which traces, analyses and supervises at run-time

**Interface HW**
e.g. Gliwa U2C or Vector CANcaseXL

**Target Interface**
CAN, diagnosis (ISO13229/UDS or ISO14230), FlexRay via diag, Nexus or JTAG

**Low bandwidth requirements!**
No HW-modification required!
Instrumentation based tracing – “online” and “offline”

- The trace data can be analyzed
  - Off-target (including visualization)
  - On-target (continuous “on-the-fly analysis”)

- On-target analysis allows
  - not only measurements but supervision: compare measured parameters with pre-defined limits
    - Continuously
    - In-car without PC connected
    - Callbacks allow e.g. error buffer entries
  - storing results in non-volatile memory → collect results from billions of executions
Instrumentation based vs. hardware tracing

<table>
<thead>
<tr>
<th>Feature / property</th>
<th>Instrumentation based tracing</th>
<th>Hardware based tracing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scope</td>
<td>Scheduling, TASKs, ISRs, runnables, functions, selected code-fragments</td>
<td>Opcode with link to HLL* code (if available), scheduling optionally</td>
</tr>
<tr>
<td>Focus</td>
<td>Scheduling, timing (debugging, visualization, supervision, verification)</td>
<td>Code debugging, detailed profiling</td>
</tr>
<tr>
<td>Interface</td>
<td>Any (CAN, Flexray, JTAG, Nexus, UART)</td>
<td>Dedicated HW-trace interface</td>
</tr>
<tr>
<td>Bandwidth requirements</td>
<td>low</td>
<td>High</td>
</tr>
<tr>
<td>In car-use</td>
<td>Yes (even without PC connected)</td>
<td>No</td>
</tr>
<tr>
<td>Code modification</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Max # of cores</td>
<td>unlimited</td>
<td>2 (on AURIX)</td>
</tr>
</tbody>
</table>

* HLL = high-level-language, e.g. C, Ada, C++
ATdemo: automotive tools demo

- Demo platform for automotive multi-core SW development

- Infineon AURIX TC275

- Multi-core demo software incl. development environment: build, flash and analyze your own application in minutes!

- With multi-core OS and T1 integrated ready for use.
Brief **T1** multi-core tool demo

Cross-core task activations

Cross-core data flow (including DFA = data-flow analysis)
Conclusion

A CONCLUSION IS THE PLACE...

WHERE YOU GOT TIRED OF THINKING.
Conclusion

- The awareness of the importance of timing is constantly growing.

- More detailed timing requirements (beyond “CPU-load < 80%”) are necessary for building safe and reliable systems.

- Well engineered requirements regarding methodologies & tools are necessary for efficient SW development.

- Tracing gets your timing on the safe side.

Thank you