



Automotive multi-core trends in Germany

Embedded Technology 2016, Japan
Multi-/many-core summit

Peter Gliwa
CEO GLIWA GmbH

Version 2

Contents

- Introduction
- Multi-core in automotive projects
- Timing requirements in OEM's requirements specifications
- State-of-the-art multi-core timing analysis
- Upcoming Standard: ARTI (AUTOSAR Real-Time Interface)
- Summary



Introduction



Gliwa GmbH – company introduction



- Timing analysis and embedded software expertise since 2003
 - embedded timing secured in **hundreds of mass-production projects**
 - located near **Munich** in Weilheim i.OB., **Germany**
 - 26+ employees mostly **engineers**
- **Stack Analysis** combining static and dynamic methods
- Distributor for Japan:

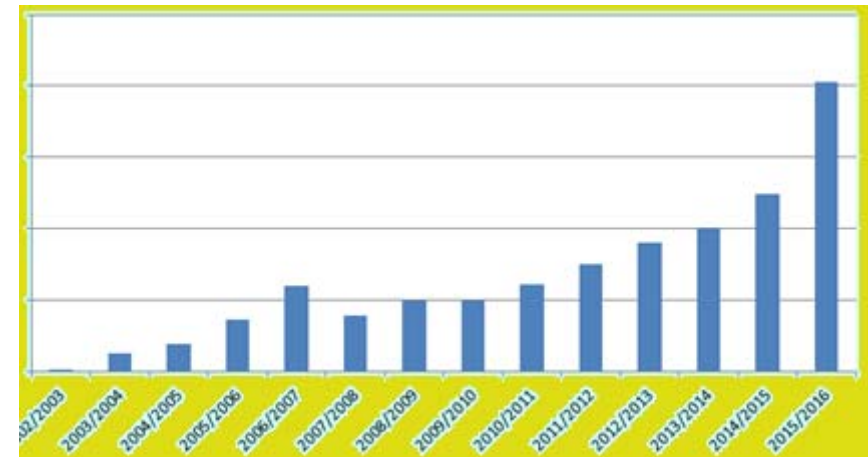


www.gliwa.com



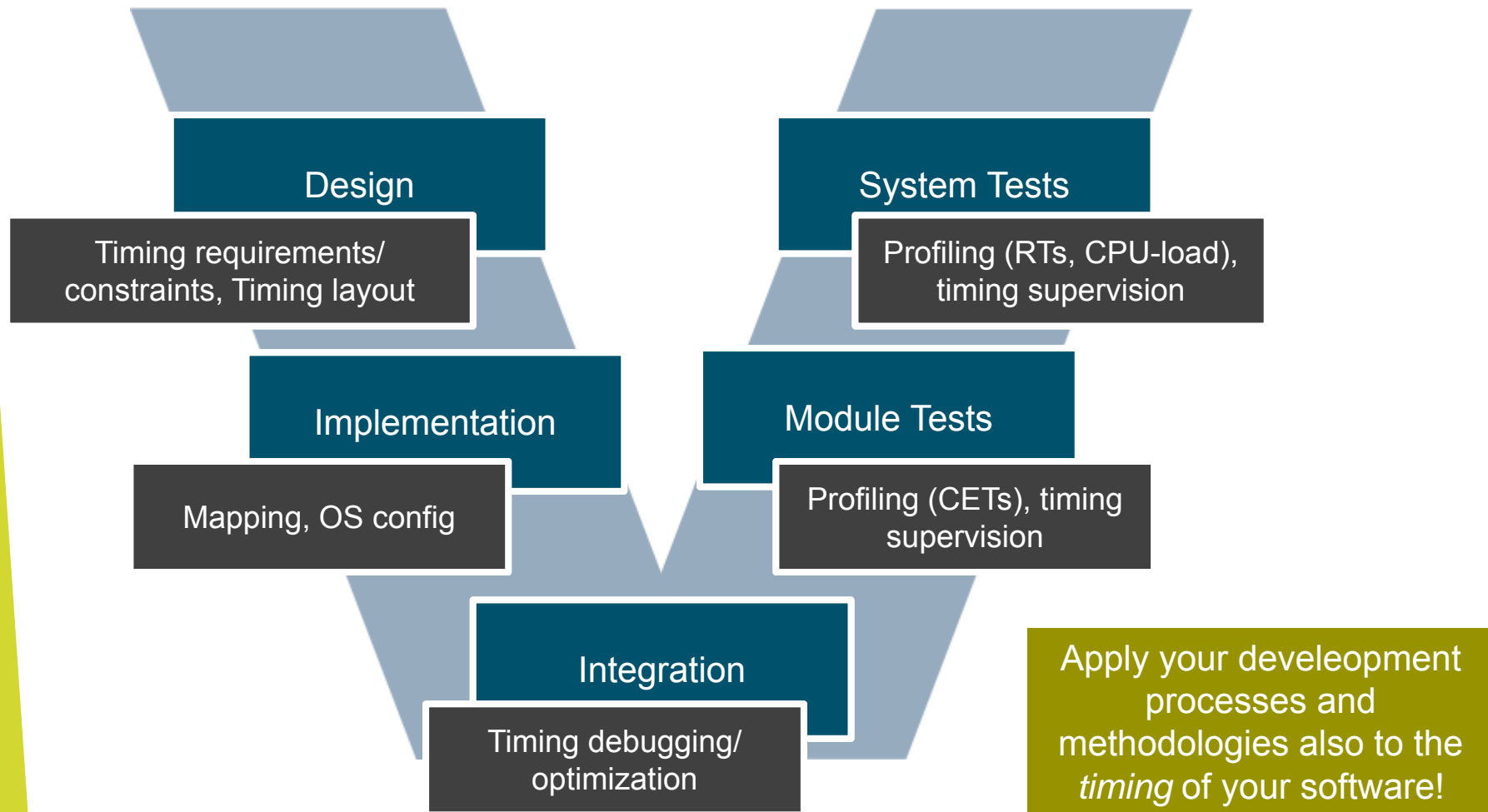
Latest figures/news

-  is the **de facto standard** for tracing in the German automotive market
-  is required by **several OEMs**
- Constant growth over the past years (mostly >20%pa)
- **GLIWA Ltd.** in York (UK) founded in 2015
- **GLIWA engineering** founded March, 2016

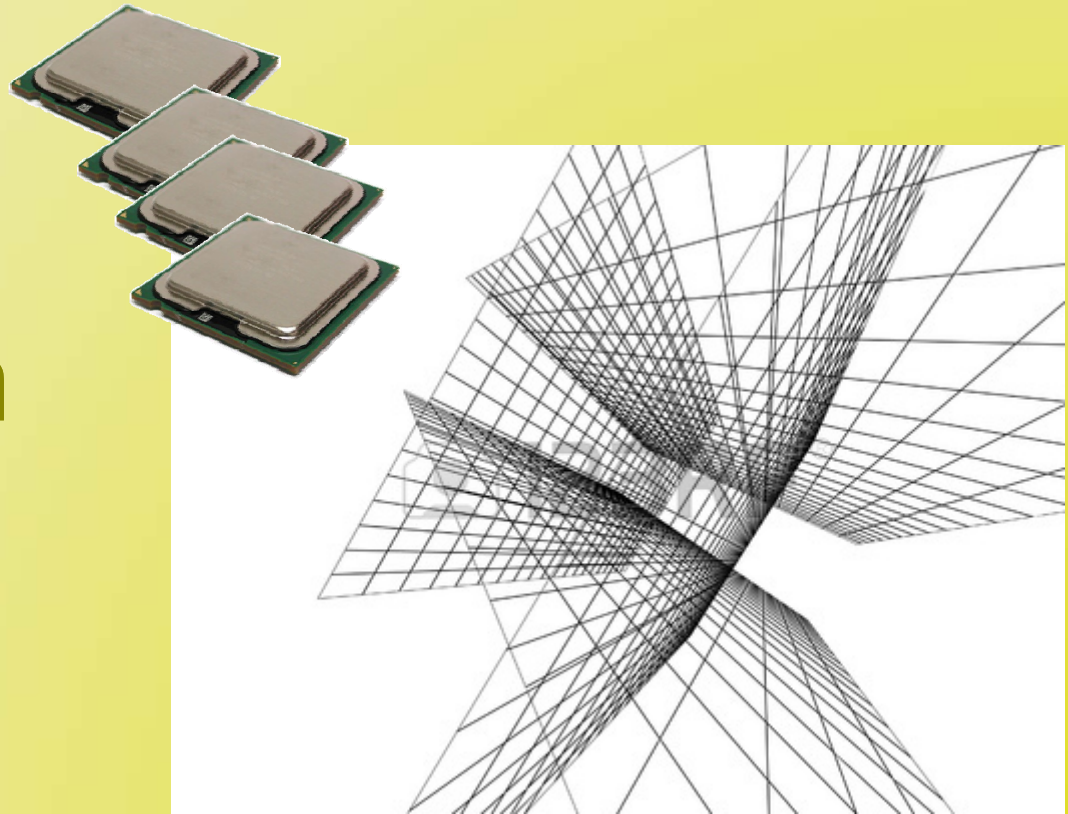


Yearly turnover

Timing: a new species?

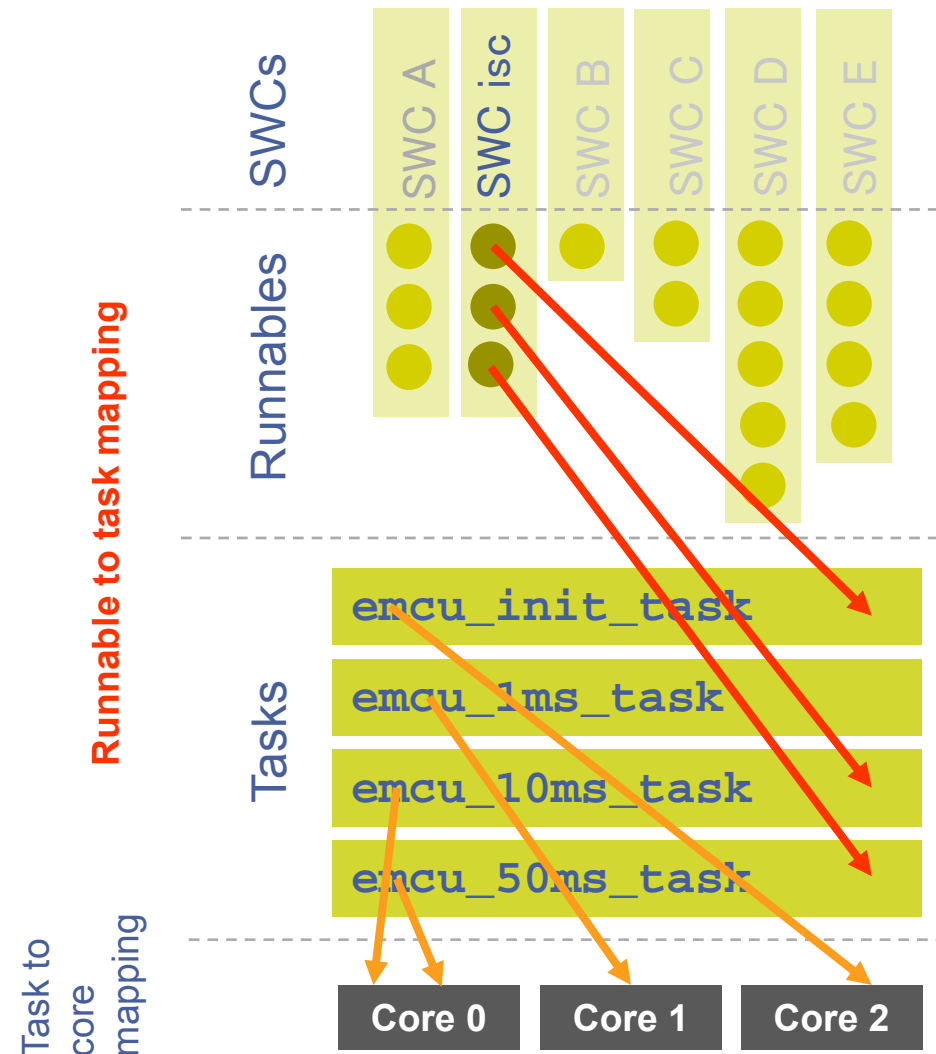


Multi-core in automovie projects

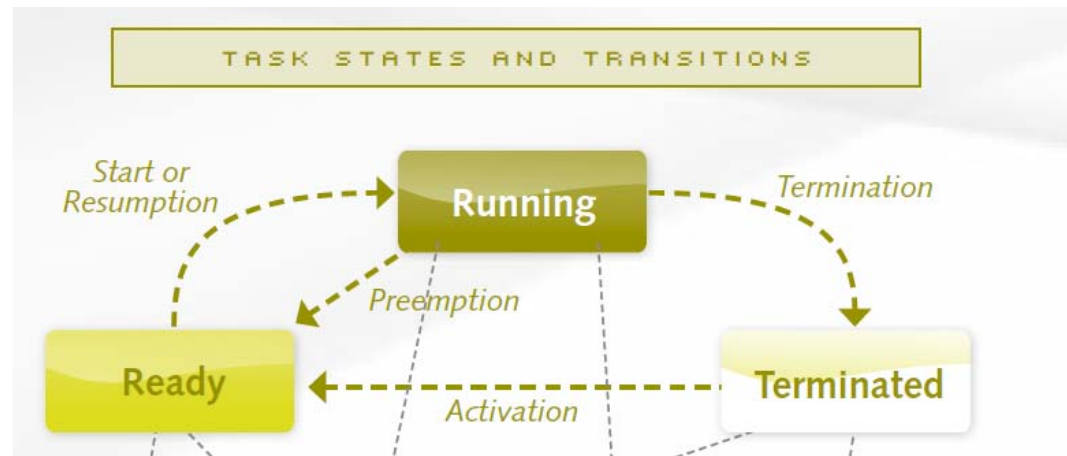


AUTOSAR scheduling: definition of terms, example

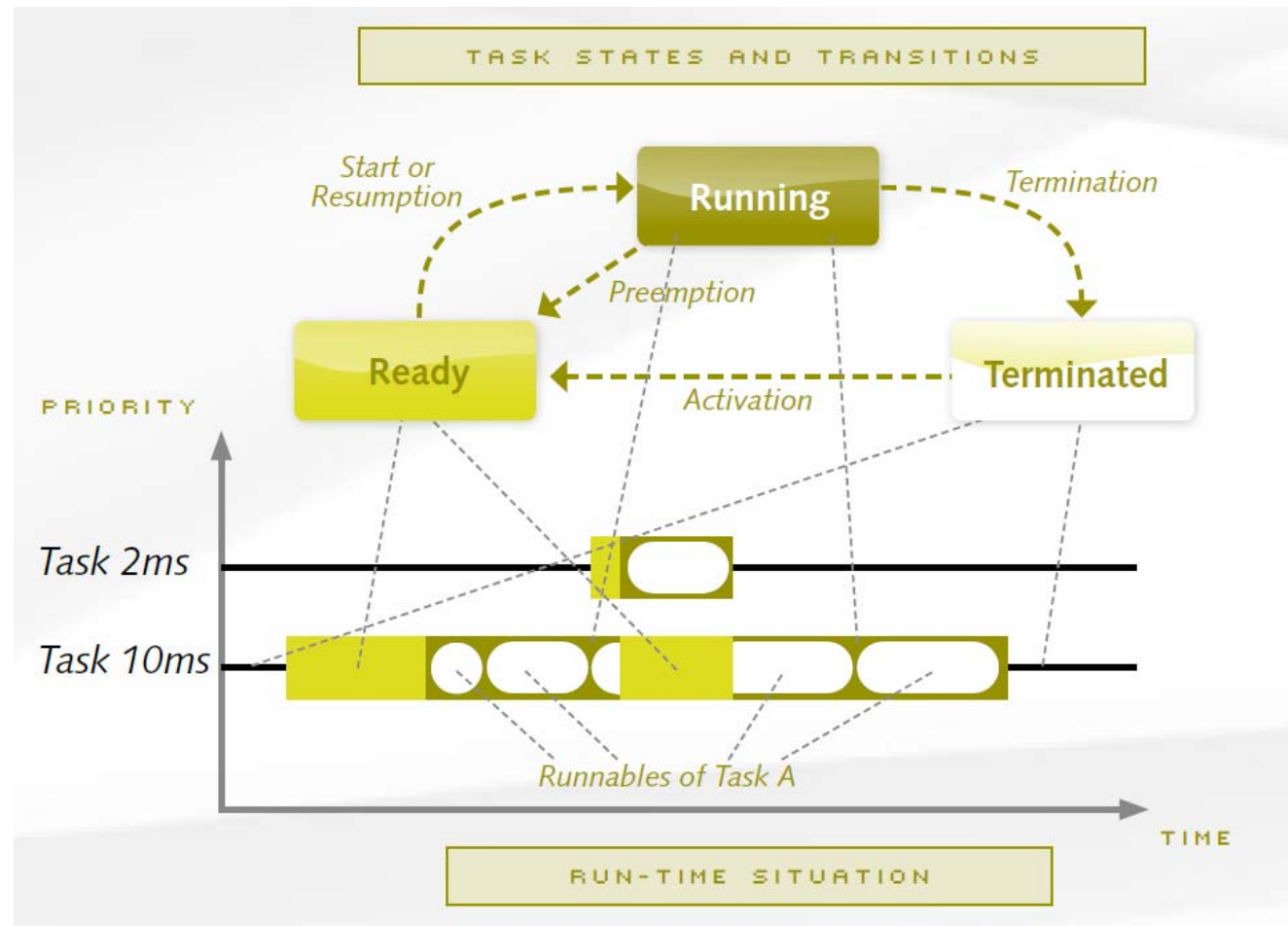
- A **software component** (SWC) “idle speed control” of an engine management ECU is coded in three **runnables**:
 - IdleSpeedInit
 - IdleSpeed10ms
 - IdleSpeed50ms
- As part of the RTOS configuration, these get mapped to three different **tasks** which they share with many other runnables from other SWCs.
- For multi-core processors, tasks get also mapped to a certain **core**.



AUTOSAR scheduling: task states

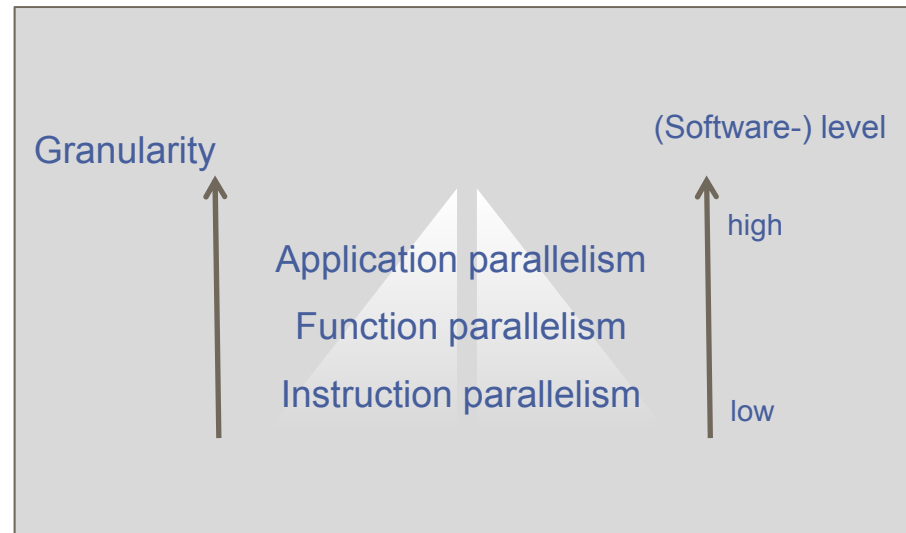


AUTOSAR scheduling: run-time situation



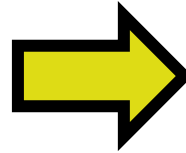
Multi-core: different kinds of parallelism

- Application parallelism
 - Each application runs on one core only.
 - Think of independent threads under a desktop OS
- Function parallelism
 - Executing portions of functional closely related code in parallel
 - Think of a sort algorithm
- Instruction parallelism
 - Processor cores have pipelines which process instructions in parallel.



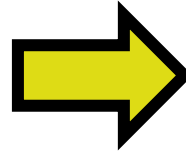
Multi-core: different kinds of parallelism

- Application parallelism
 - Each application runs on one core only.
 - Application \neq thread



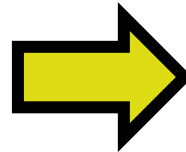
- **Easy:** take two single-core ECUs and assign each SW to the core of a multi-core ECU

- Function parallelism
 - Executing portions of functional related code in parallel



- **Extremely difficult:** automotive SW is typically not designed to be parallelized

- Instruction parallelism
 - Processor cores have pipelines which process instruction in parallel.



- **Mostly easy:** follow some design rules to exploit your pipeline

The Multi-core Poster – Multi-core on one sheet of paper

AN INTRODUCTION TO AUTOMOTIVE Multi-core

1. INTRODUCTION

A table of parallel processing:

- Imagine you want to have a kitchen built in one day (→ 8 hours)
- You ask a craftsman to do it but he says: "It will take me 16 hours."
- You might have a second one in order to get the job done in time.
- But while one craftsman connects the electric, the other one takes the time to build the kitchen, the other one can't use his power tools and is blocked.
- They also spend a lot of the time talking to each other.
- They finish after 11 hours (completely demand and you give to get the job done).

This poster sheds a light on automotive multi-core embedded software through aspects. Proper multi-core know-how helps to avoid software projects running into situations as described above.

2. WHY MULTI-CORE?

Multi-core processors have been used for decades in domains other than automotive. Every PC and every smartphone comes with at least a dual-core processor. The main reason for using more than one core within the processor is the ever increasing need for more computation power.

Moore's Law – dated 1965 – says: "The number of transistors in a dense integrated circuit doubles approximately every two years."

Good reasons for more computing power include:

- More and more advanced vehicle features (lane-keeping, autonomous driving, car-to-x communication, etc.)
- Stricter safety requirements (diagnostic, memory protection, on-target supervision, etc.)
- Increasing use of standards and generated code leads the scope of optimization.
- Building faster higher clock speed high-core processors becomes less expensive at some point due to the following reasons:
- Power consumption (P = P (heating) core)
- Die (Dielectric) conductivity problems
- Power dissipation (→ "Silicon Drought")

3. MULTI-CORE THEORY

Andrius's law: "The speed of a program using multiple processors is limited by the time needed for the sequential fraction of the program." (1)

→ In other words:

- It takes a certain time to write a code to solve a problem. After execution is not going to do it in a month.

Andrius's law applies when there is a significant portion of code which cannot be parallelized.

Goldbach's law: "Programmers tend to use the size of the problem to use the available equipment to solve a problem. Therefore, if faster (more parallel) equipment is available, larger problems can be solved in the same time." (2)

Goldbach's law applies when a given problem can be replaced by a larger problem solving the old problem plus other problems.

4. MULTI-CORE HARDWARE ARCHITECTURES

Multi-core multi-core processors have different cores of different types.

Examples:

- Infineon TC1797 (TC1.1 and PC0)
- Infineon BPC100 with TPU
- Infineon S32A with M4/M5
- Infineon TC277 (several different cores, see next section)

Multi-core multi-core processors have a number of cores of the same type.

Examples:

- Infineon BPC100
- Infineon TC277 (see TC1.1 core, see next section)

Multi-core multi-core processors combine the same single-core software on two separate cores at the same time, for safety reasons. The result of the two cores get continuously compared by the hardware. When a mismatch (error) occurs, the processor will switch to a safe state.

Chip designers spend a lot of effort to build common code blocks: right execution delay between the cores, separate clock lines, isolated and fixed 2nd CPU, potential parallelizing around each CPU, etc. (3)

Examples: Texas Instruments TMS320C6x, Infineon Aurix™ (TC1.1 core with dual-core core, see next section)

5. EXAMPLE INFINEON TC27X "AURIX™" (3)

Each TC27X has local program memory and local data memory that it can access with no delay. With significant delay for the CPU and local memory, each TC27X can also access data/program memory of other cores, see also section "On-Chip" (4).

Access to peripherals "out" up to 4 or 7 CPU stall cycles depending on the peripheral bus configuration.

The shared program flash and the shared data flash cause a maximum of 5 or 6 stall cycles. These numbers show that location of data and code has a significant impact on the timing.

There are several other Infineon TC27X (5).

Each TC27X has local program memory and local data memory that it can access with no delay. With significant delay for the CPU and local memory, each TC27X can also access data/program memory of other cores, see also section "On-Chip" (4).

Access to peripherals "out" up to 4 or 7 CPU stall cycles depending on the peripheral bus configuration.

The shared program flash and the shared data flash cause a maximum of 5 or 6 stall cycles. These numbers show that location of data and code has a significant impact on the timing.

There are several other Infineon TC27X (5).

6. AUTOSAR AND MULTI-CORE

AUTOSAR originally was designed for single-core processors but has been extended with a number of multi-core features:

- Starting and shutting down other cores
- Core lock activation and lock checking (however, lock migration is not supported and also not expected)
- Spinlocks ("on-core synchronization", explained later)
- ICC (Inter-Core Application Communication)

AUTOSAR does not (yet) support:

- AUTOSAR RTE optimization across cores (transmission resource locks can be optimized away on a single-core system but not on a multi-core system)
- Inter-core data passing by reference (copying data is mandatory which becomes an issue when dealing with large data)

7. DATA-CONSISTENCY, SPINLOCKS

While a single-core application can use internal locking to ensure data consistency, this is not sufficient for multi-core systems sharing data between cores. A command "disable all interrupts" only affects the core executing the command. AUTOSAR introduces spinlocks for synchronization in multi-core systems.

Example: assume an application has two frequent interrupts and it needs to know the total number of occurrences of both interrupts.

at Both interrupts get executed on a single core:

at Both interrupts get executed on different cores:

The spinlock-related AUTOSAR version are:

- Release/spinlock releases a spinlock. Critical sections must be released in the correct order. The last obtained spinlock must be released first.
- Get/spinlock returns a spinlock when no other spinlock is using it. If another spinlock is using it then Get/spinlock spins (loops) until the spinlock can be correctly obtained.
- TryGet/spinlock is a non-blocking version of Get/spinlock. It always returns immediately with no spinning.

The straight-forward implementation that is rarely suitable for real applications and/or unattended delays when one core acquires then handles one or more interrupts. A lock is shown below and can be used as spinlock usage.

8. GOLDEN RULES

The golden rules for creating simple, easy-to-develop and efficient multi-core software are:

- Statically allocate code and data to cores so that you can analyze and optimize that allocation
- Localize code and data on one core to minimize cross-core accesses
- Duplicate data and code where appropriate to achieve the goal
- Decouple code on different cores
- Remember that statically allocated data typically up to 64 bits can be accessed by one core.

Ask for your own free copy at the GAIO booth: D-34

Size requires extra know-how and brings some discomfort. Before long, we can expect significant multi-core support, including the AUTOSAR standards and code generators. Complex, single-core projects will be the exception and will be regarded kindly as artifacts.

9. GLOSSARY

TERM	DEFINITION
AAE (Automotive Architecture Engineering)	A multi-core system with a separate operating system per core.
Asynchronous	Addressing of multi-core processors based on up to three (three) CPU cores.
Caching	Cache is a temporary data, code and control flow between different software components.
Coherency	Coherency is a property of a system that ensures that all data is consistent and up-to-date.
ICC	Inter-Core Application Communication. Part of the AUTOSAR OS responsible for managing communication between OS-Applications in parallel, and to implement, from one core to another.
IK	Internal Kernel. A kernel is a software component that manages resources to handle a multi-core system.
Multi-core	Having more than one core in a processor. It is a multi-core system that implements heterogeneous multi-core.
Non-blocking	An implementation of a kernel that is guaranteed not to block.
OS	Operating System. An operating system is a software component that manages resources to handle a multi-core system.
OS-Application	AUTOSAR base for collection of application software. Above that one OS-Application can run on one core but an OS-Application can run on more than one core.
Parallel	Set of processing steps for handling a sequence of data items. As soon as the first data item is processed from the first data to process the second item, the second item can be processed.
Spinlock	A lock mechanism for achieving mutual exclusion. It is a multi-core system that implements heterogeneous multi-core.
System	System is a multi-core system that implements heterogeneous multi-core.
Task	Collection of software that executes sequentially and often, but not necessarily, periodically.

Timing requirements today and tomorrow



Requirements specification documents

- 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



methodologies &
tools requirements

Timing requirements

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 containers holding OEM code!)
- **ISO 26262**
 - requires “freedom of interference”
 - in other words: **correct and secured timing**

Methodology requirements


methodologies &
tools requirements


- Define a process/infrastructure for **maintaining timing requirements** → handle upcoming timing requirements
- Select appropriate timing analysis techniques, e.g.
 - scheduling simulation for the early project phase and
 - traing/measurement for the late phase
- 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**.

Tool requirements

- 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.

methodologies &
tools requirements

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 be able to measure. So they did not get access to the existing supplier internal tools. 

Templates for requirements specifications

- 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**.

timing
requirements

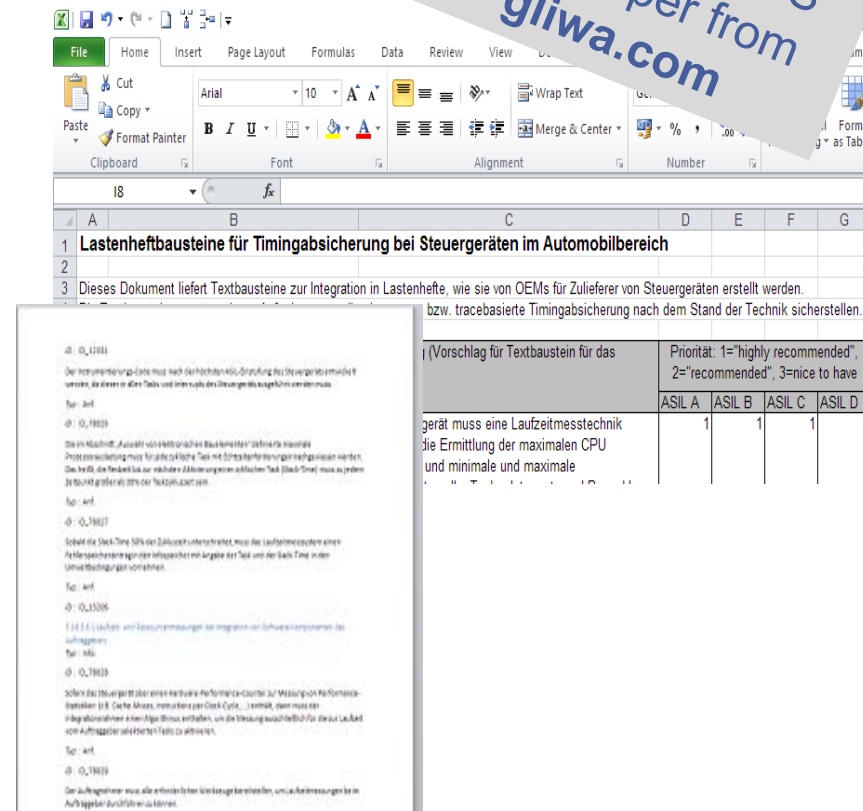


methodologies &
tools requirements

Templates for requirements specifications

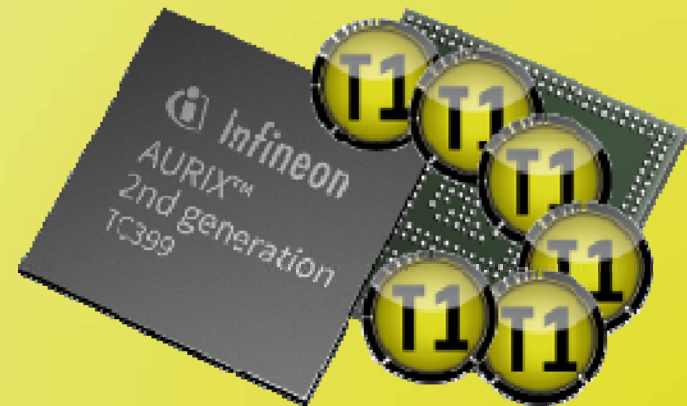
- Excel table with text templates
 - including recommendation according to ASIL level
- Word document with templates
- Some big OEMs follow this approach already. More and more follow...

Download ERTS
2012 paper from
gliwa.com

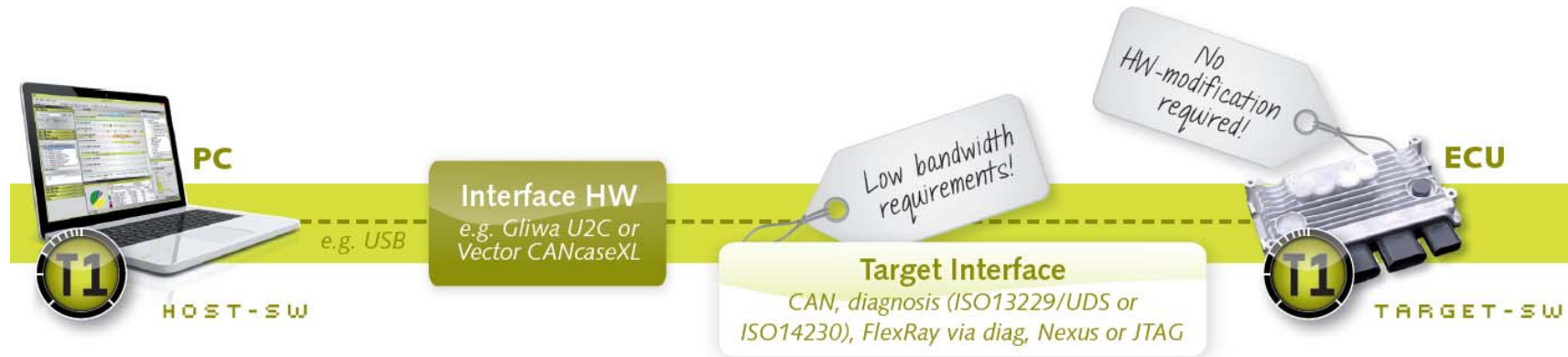


	ASIL A	ASIL B	ASIL C	ASIL D
gerät muss eine Laufzeitmesstechnik die Ermittlung der maximalen CPU und minimale und maximale	1	1	1	1

State-of-the-art multi-core timing analysis



Instrumentation-based tracing: T1



T1-HOST-SW

PC based SW tool for visualization, analysis and configuration

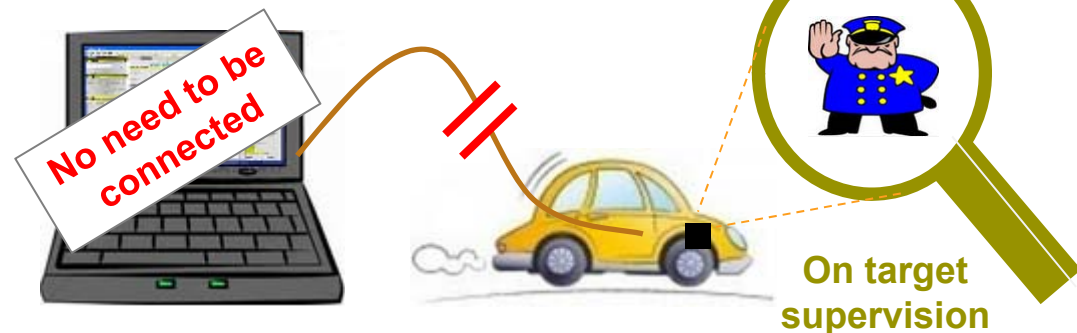


T1-TARGET-SW

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

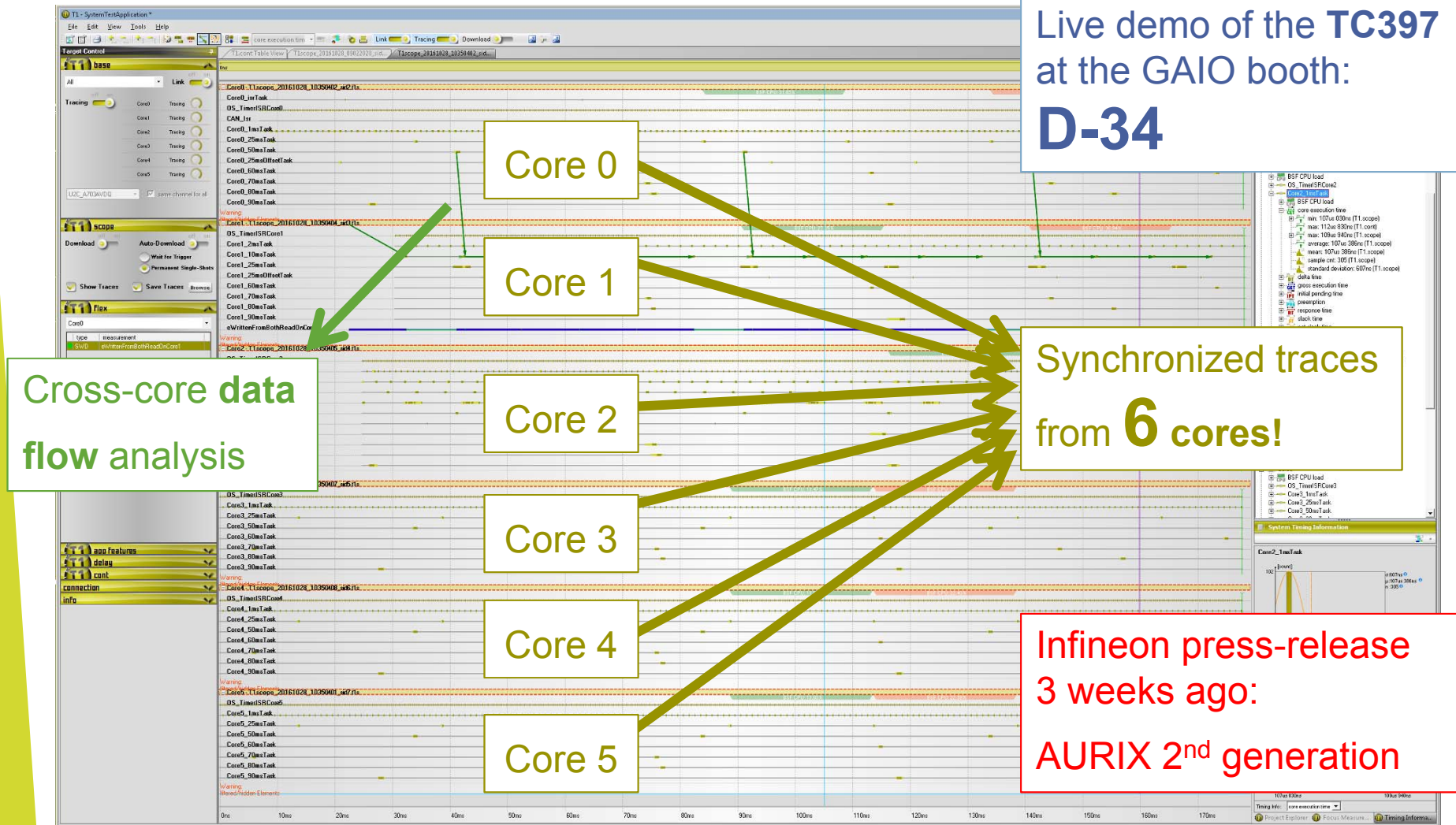
Instrumentation based tracing – “online” and “offline”

- 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



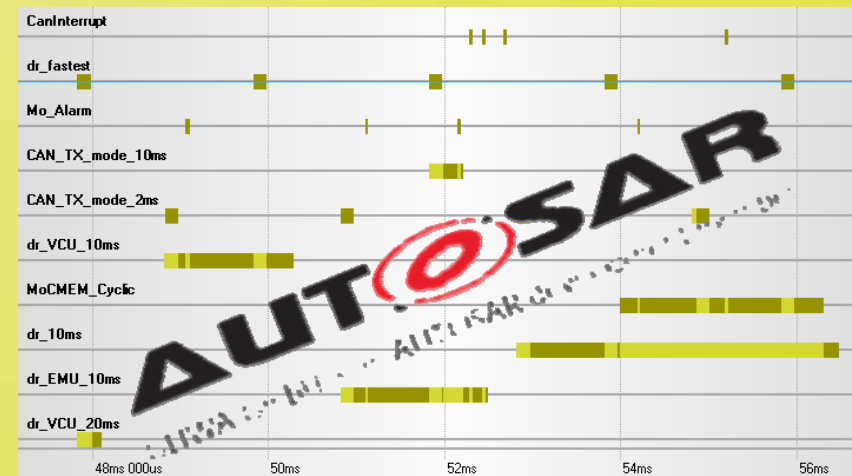
Latest news: synced traces fro 6 cores

Live demo of the **TC397**
at the GAIO booth:
D-34



ARTI

AUTOSAR Real-Time Interface



AUTOSAR and timing

- **Timing Extensions**
"TIMEX"; since AUTOSAR 4.0
→ Allow specification of timing requirements
- **Timing Analysis**
First released with 4.1.3
→ Use-cases based guide to timing
- **AUTOSAR OS**
Contains timing protection mechanisms
→ As of 4.2.2: Execution-, Locking- and Inter-Arrival Time Protection
- **ARTI** (AUTOSAR Run-Time Interface)
not a standard yet (probably in 2017)
→ more details later

AUTOSAR

Timing Analysis
V0.0.3
R4.1 Rev 3

Document Title Timing Analysis

Document Owner AUTOSAR

Document Responsibility AUTOSAR

Document Identification No 645

Document Classification Auxiliary

Document Version 0.0.3

Document Status Draft

Part of Release 4.1

Revision 3

Document Change History

Date	Version	Changed by	Description
2013-10-11	0.0.1	Team	Prepared document based on AUTOSAR LaTeX template MS2 Review comments taken into account; added references to main AUTOSAR requirements for traceability (see Section 1.9); removed empty chapters and unified structure of chapter 3 and chapter 4; completed LaTeX migration
2013-12-18	0.0.2	Peter Gliwa & Team	

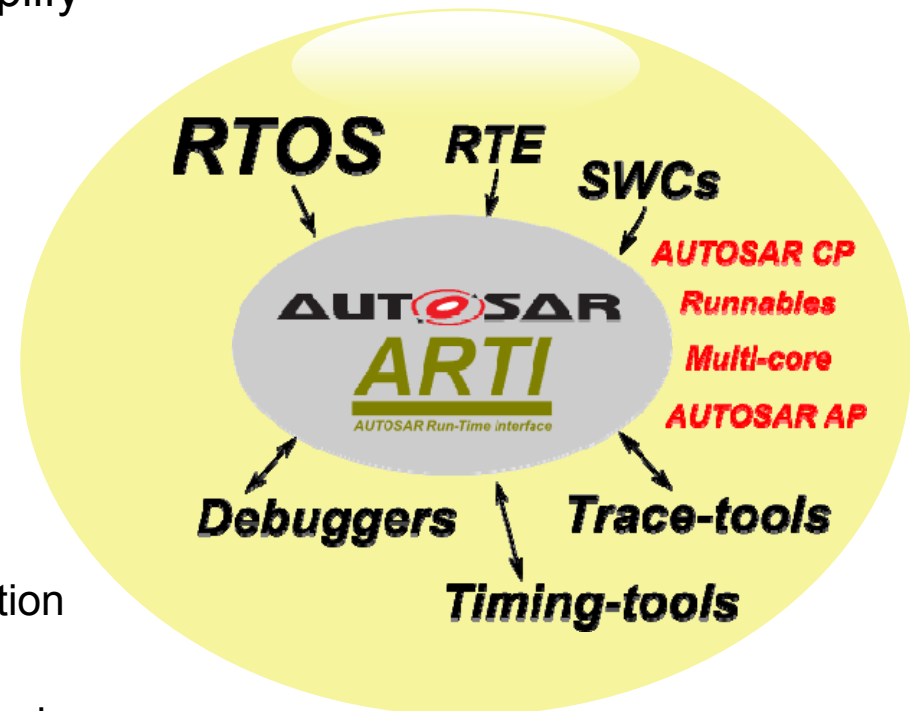
1 of 91

Document ID 645: AUTOSAR_TPL_TimingAnalysis
— AUTOSAR CONFIDENTIAL —

Timing Analysis document

ARTI goals

- Generally, ARTI shall support and simplify the following:
 - Debugging
 - Tracing
 - Timing Measurement / Profiling
- ARTI shall support
 - Multi-core
 - Runnables
 - Instrumentation-based tracing and measurement solutions
 - The actual AUTOSAR-OS implementation
 - TIMEX
 - debugging and tracing beyond ECU level, e.g. end-to-end timing taking several ECUs and buses into account
 - AP (adaptive platform)



ARTI: who is behind it?

AUTOSAR/ OS vendors



Timing Tool vendors



Debug/Trace Tool vendors

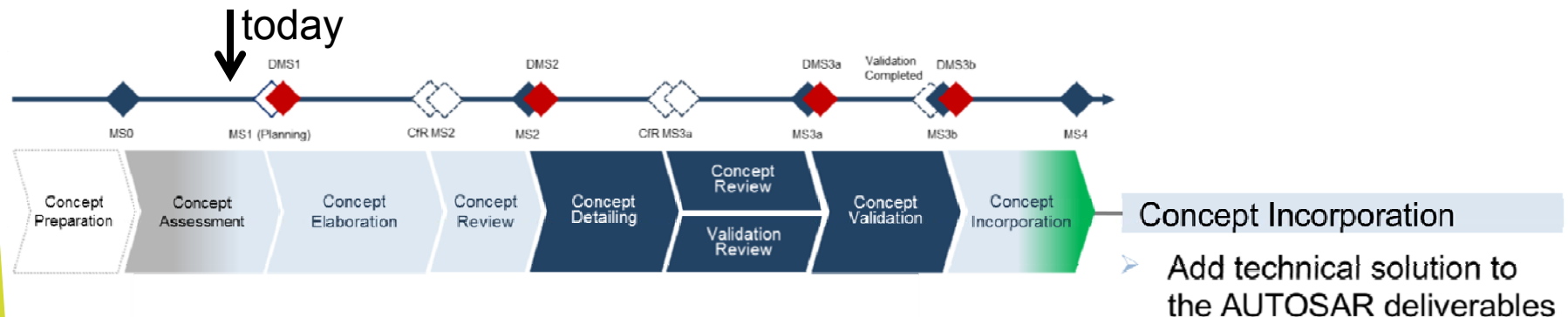


Users, AUTOSAR experts



ARTI: current status/schedule

- Concept owner: Peter Gliwa <peter.gliwa@gliwa.com>
- MS0 passed 2016-09-07
- Further planning:
 - MS1 (WP Assessments completed): 2016-11-25
 - MS2 (review readiness): 2017-02-15
 - MS3a (Call for review): 2017-09-15
 - **MS3b (Validation completed): 2017-11-01**



Conclusion



Conclusion

- More **detailed timing requirements** 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.
- Developing multi-core ECU SW **without timing** tools is far **too costly** and **error prone**.



peter.gliwa@gliwa.com

Thank you

