Predictable hardware: The AURIX Microcontroller Family

Worst-Case Execution Time Analysis

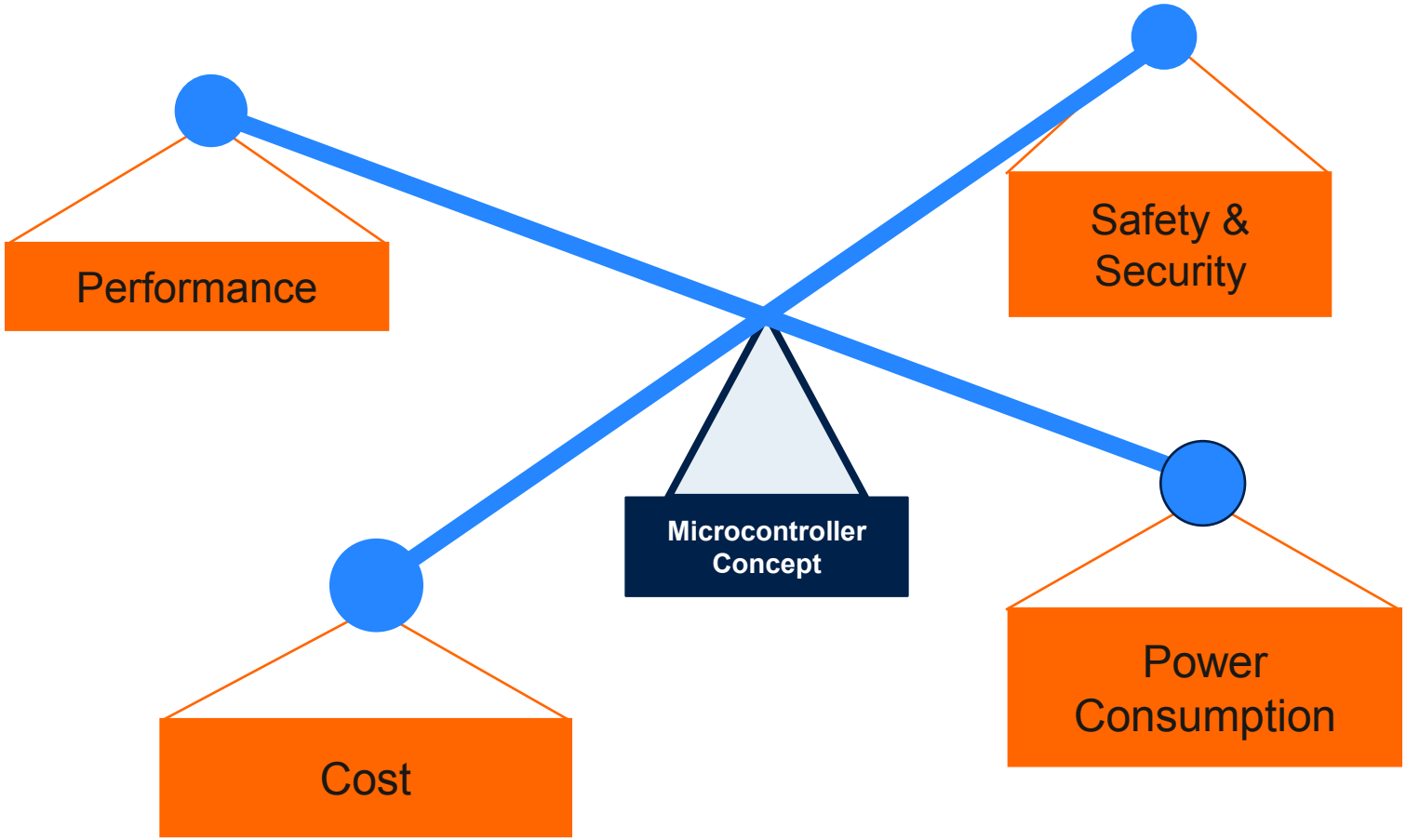
WCET 2013, July 9, 2013, Paris, France

Jens Harnisch (Jens.Harnisch@Infineon.com),
Infineon Technologies AG,
Automotive Microcontroller, Application and Concept Engineering

Outline

- Challenges for Automotive Microcontrollers
- Power train application characteristics
- AURIX: WCET related system features
- AURIX: WCET related core features
- Multi-Core Software Considerations
- Tracing Options

Automotive Microcontroller – The Challenge



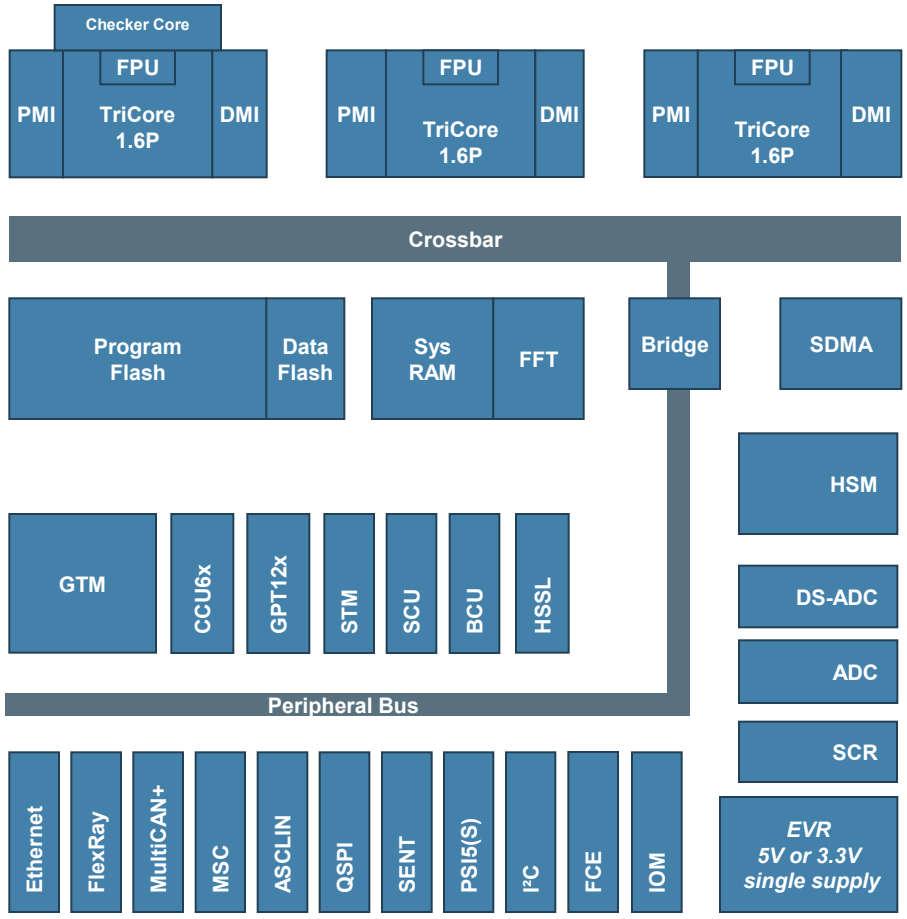
Keep Constraints for Hard Real Time Support:

- **Predictability**
- **Keep latency constraints (e.g. for interrupts)**
- **Non intrusive trace support**

Characteristics of typical applications running on AURIX (power train)

- Static assignment of tasks to cores, separated schedulers (AMP)
- Time and event driven components (event driven components handled via pre-emption)
- Roughly every 7th instruction is a branch
- Floating point instructions: varying, depends on code generation
- Data locality: Data (except for LUTs) fits usually into DSPR, having 0 clocks latency
- Look up tables are frequently used
- Instruction level parallelism: 0.8 is often a good result; with high optimization more than 1 instruction per cycle is feasible (TriCore TC 1.6P has 3 pipelines)

AURIX Multi-Core Controller: WCET related system features



- Highly predictable architecture with duplicated resources (local memories, crossbar) to avoid resource conflicts
- Priority driven and round robin arbitration for masters attached to crossbar
- Starvation protection in crossbar
- No cache coherence
- Dedicated and scalable communication instructions

TriCore 1.6E and TriCore 1.6P: WCET related features

TriCore 1.6**E**fficiency	TriCore 1.6**P**erformance
	

Common TriCore 1.6 Instruction Set

- **One pipeline only for high power efficiency**
- **4 pipeline stages** for up to 200MHz
- Power 0.2mW/MHz
- **1.2 – 1.4 DMIPS/MHz**

- Superscalar: integer, load store and loop pipeline
- **6 pipeline stages** for up to 300MHz
- Power 0.3mW/MHz
- **1.6 – 2.3 DMIPS/MHz**

- **Caches: 2 way LRU**
- **Easy to describe RAW and structural dependencies**
- **No support of simultaneous multithreading**
- **4 stage data store buffer**

Support for hard real time systems

- Support for high average case performance usually contradicts high predictability
- Static timing analysis: precision of results and efficiency of analysis strongly depend on hardware architecture features
- Three classes of hardware architectures [1]:
  - Fully timing compositional architectures: no timing anomalies
  - Compositional architectures with constant-bounded effects: timing anomalies without domino effects -> TriCore is assumed to belong to this class
  - Non-compositional architectures

[1] Predictability Considerations in the Design of Multi-Core Embedded Systems. C. Cullmann, C. Ferdinand, G. Gebhard, D. Grund, C. Maiza, J. Reineke, B. Triquet, Wilhelm. Ingénieurs de l'Automobile, 807, 2010.

Design Guidelines for predictable multicore architectures

- Established by C. Cullmann, C. Ferdinand, G. Gebhard, D. Grund, C. Maiza, J. Reineke, B. Triquet, S. Wegener and R. Wilhelm [1]

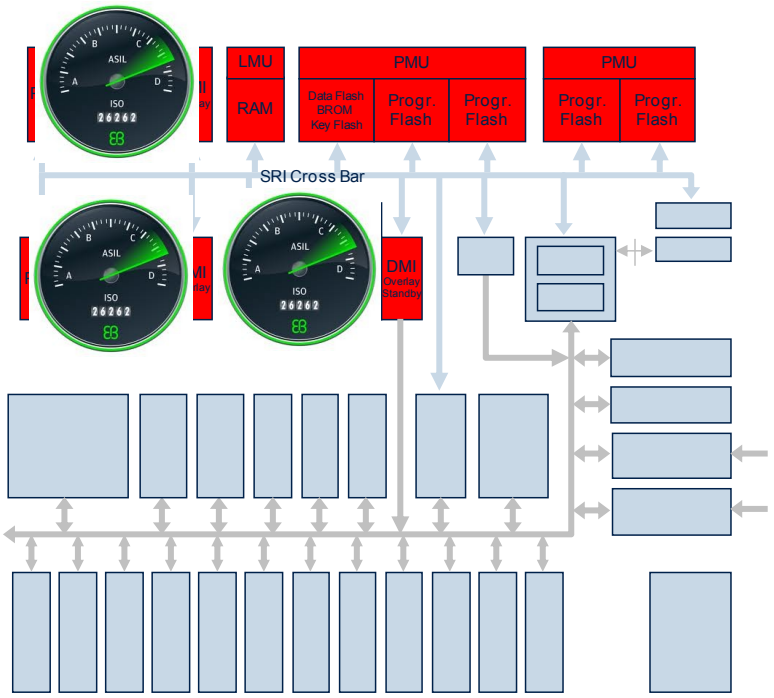
AURIX 

1. Fully timing compositional architecture ✓ (with bounds)
2. Disjoint instruction and data caches ✓
3. Caches with LRU replacement policy ✓
4. A shared bus protocol with bounded access delay ✓
5. Private caches ✓
6. Private memories, or, only share lonely resources ✓

[1] Predictability Considerations in the Design of Multi-Core Embedded Systems. C. Cullmann, C. Ferdinand, G. Gebhard, D. Grund, C. Maiza, J. Reineke, B. Triquet, S. Wegener, and R. Wilhelm. Ingénieurs de l'Automobile, 807, 2010.

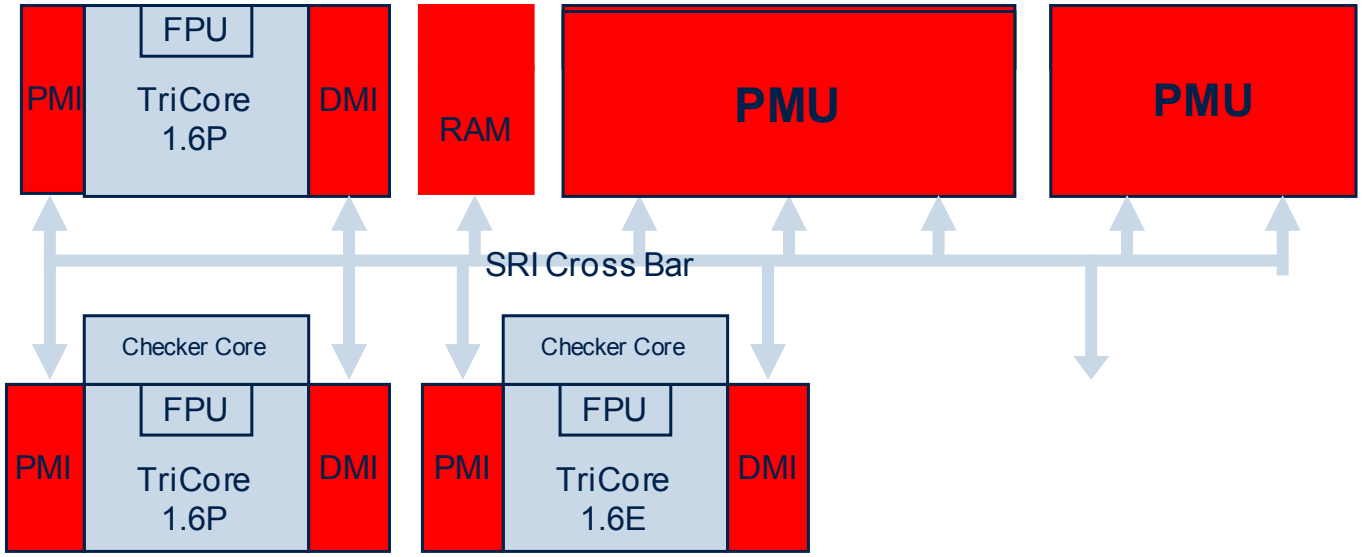
Some software mapping examples

- Interrupt driven tasks on one core, time driven tasks on other(s)
- Split all tasks across cores to balance performance
- With and without OS (computationally intensive functionality)
- Split by ASIL level
- Split by tier 1 / OEM
- Important: Differentiate between your(!) optimization objectives (metrics) and constraints
- Examples for both optimization objectives and constraints :
  - Core load, memory balance, specific task latencies, end-to-end latencies

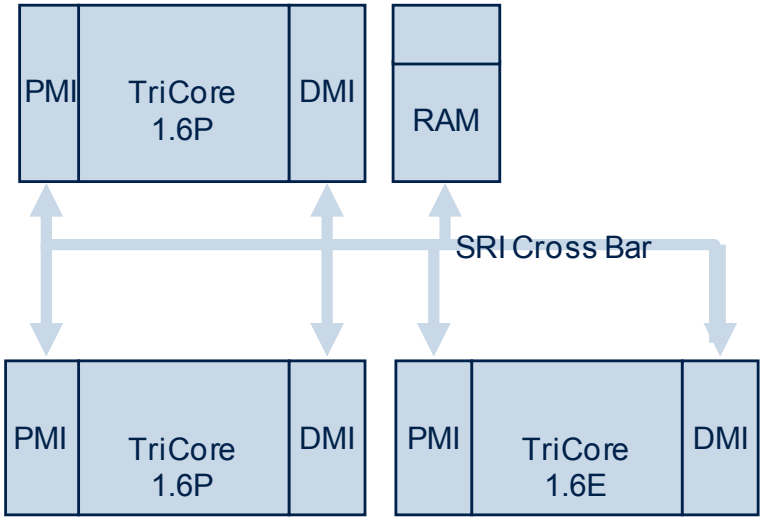


What does mapping of software mean?

- To map: **functionality to cores, code and data to memories**!
  To use: scheduling and communication (AUTOSAR mechanisms and/or direct mechanisms).
- Major mapping criteria: Core performance, support for lock step, memory access latencies and sizes, access conflicts
- Summary for the software developer:
  - Mapping of functionality often constrained by performance of cores, lock step and hence is not so complex
  - Mapping of code and data: Up to **9(!)** different memory locations to consider: possibly complex



Real life example: Race condition and deadlock after few micro seconds!

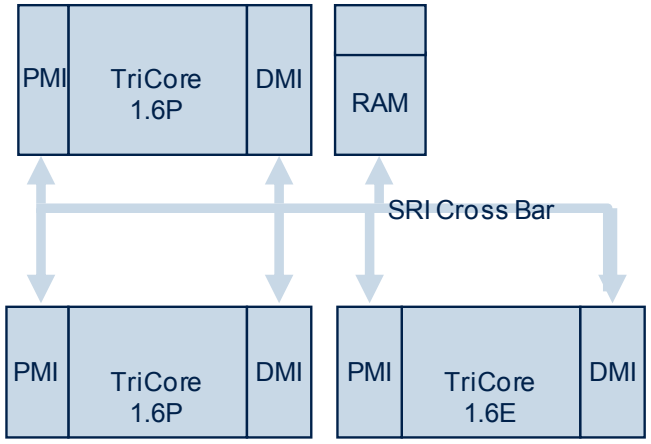


core 0 ready	core 1 ready	core 2 ready	not used

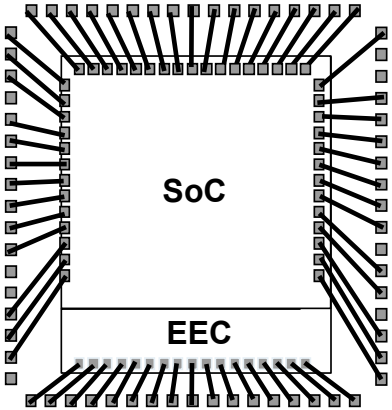
- 3 bits of a 32 bit variable used to synchronize all three cores, located in RAM and cached in all 3 DMIs

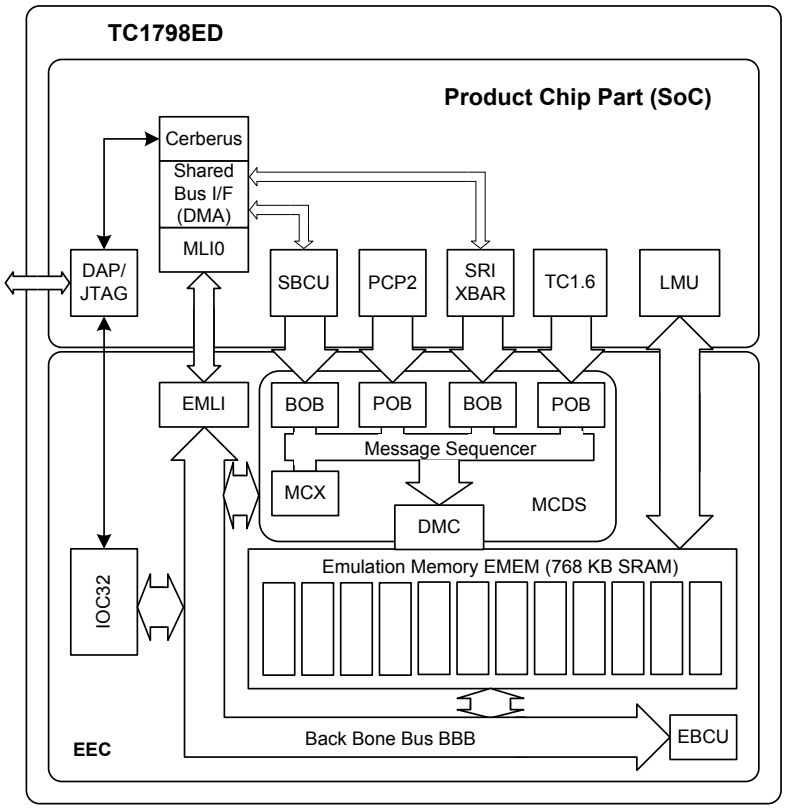
Real life example: Details

- 3 bits of a 32 bit variable used in RAM to synchronize all 3 cores, after synchronization
  cores should proceed independently.
- Atomic operation applied to 32 bit variable to modify ready bit for each core.
- Because the synchronization may often be necessary, the developer decided to cache the variable.
- However, atomicity refers only to physical memory location (in this case the cache, and not LMU RAM)!
- Atomicity was wanted, but not really guaranteed, hence a race condition was provoked!
- Incorrect status of variable (due to race condition) led to deadlock.
- Incorrect solution of the developer: change timing via cross bar priorities -> software was executing without deadlock, but was still incorrect!
- Correct solution: Do not cache globally used variables!



Trace, Measure and Calibrate in Target System with Emulation Devices (EDs)

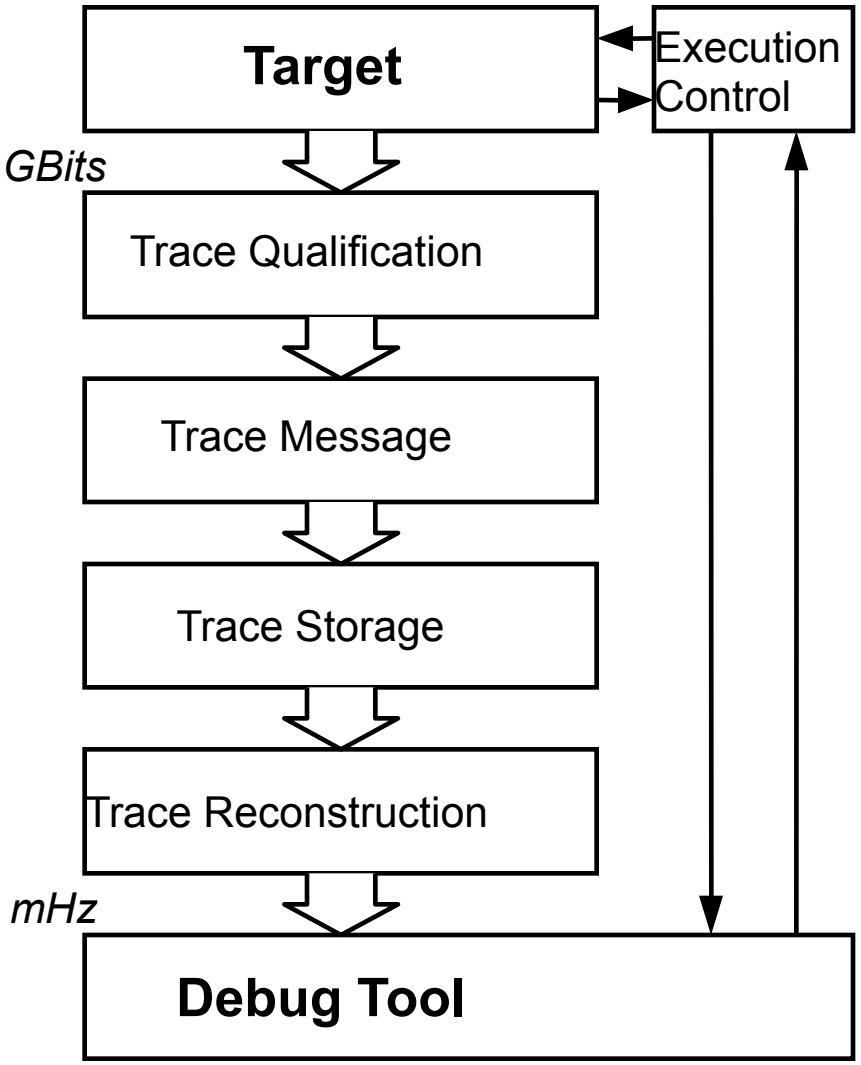




ED_block_diagram.vsd

- Tracing as non-intrusive technology (with regards to timing) gains importance for multi-core software development
- EDs: Same package, no additional pins needed
- Calibration memory with same access speed as flash

Flow for using MCDS



- Establish test case
- Configure “trigger, event, action”
- Generate only relevant information
- Save messages chronologically
- Interpolate history
- Present in human readable form

Summary

- Reaching higher performance with more cores rather than with core and memory hierarchy optimizations might simplify WCET, if there is little interference between the cores
- Powerful protection and tracing mechanisms will be needed to assure non-interference or at least assess the degree of interference
- AURIX has a focus on protection mechanisms to avoid interference (where not needed) and powerful tracing options to assess interference (where unavoidable)

Timing and Performance Analysis Partners, Acknowledgements

- Academia Partners
  - Prof. Dr. Bernhard Bauer, Augsburg University
  - Prof. Dr. Heiko Falk, Ulm University
- Industry Partners
  - AbsInt Angewandte Informatik GmbH, Gliwa GmbH, Symtavison GmbH, Timing Architects GmbH
  - Debugger Vendors with Trace support: iSYSTEM AG für Informatiksysteme, Lauterbach GmbH, PLS Programmierbare Logik & Systeme GmbH
- Acknowledgements
  - Prof. Dr. Claire Maiza (Grenoble INP/ Ensimag)
  - Reinhard Deml, Frank Hellwig, Pawel Jewstafjew, Dr. Albrecht Mayer (Infineon)

Thank you for the attention