A systematic approach for timing requirements

EMCC 2018, Munich

Version 1.1

Peter Gliwa
CEO GLIWA GmbH

Contents

GLIWA GmbH is an AUTOSAR development member

Introduction

Why care about timing?

- Proper timing of ECU software is essential for
  - Reliability / Availability
  - Safety (and often also Security)

- Timing problems are very often difficult
  - to identify as such, to debug, to solve

- ISO 26262 requires “freedom of interference”
  - Can only be guaranteed in the absence of timing problems

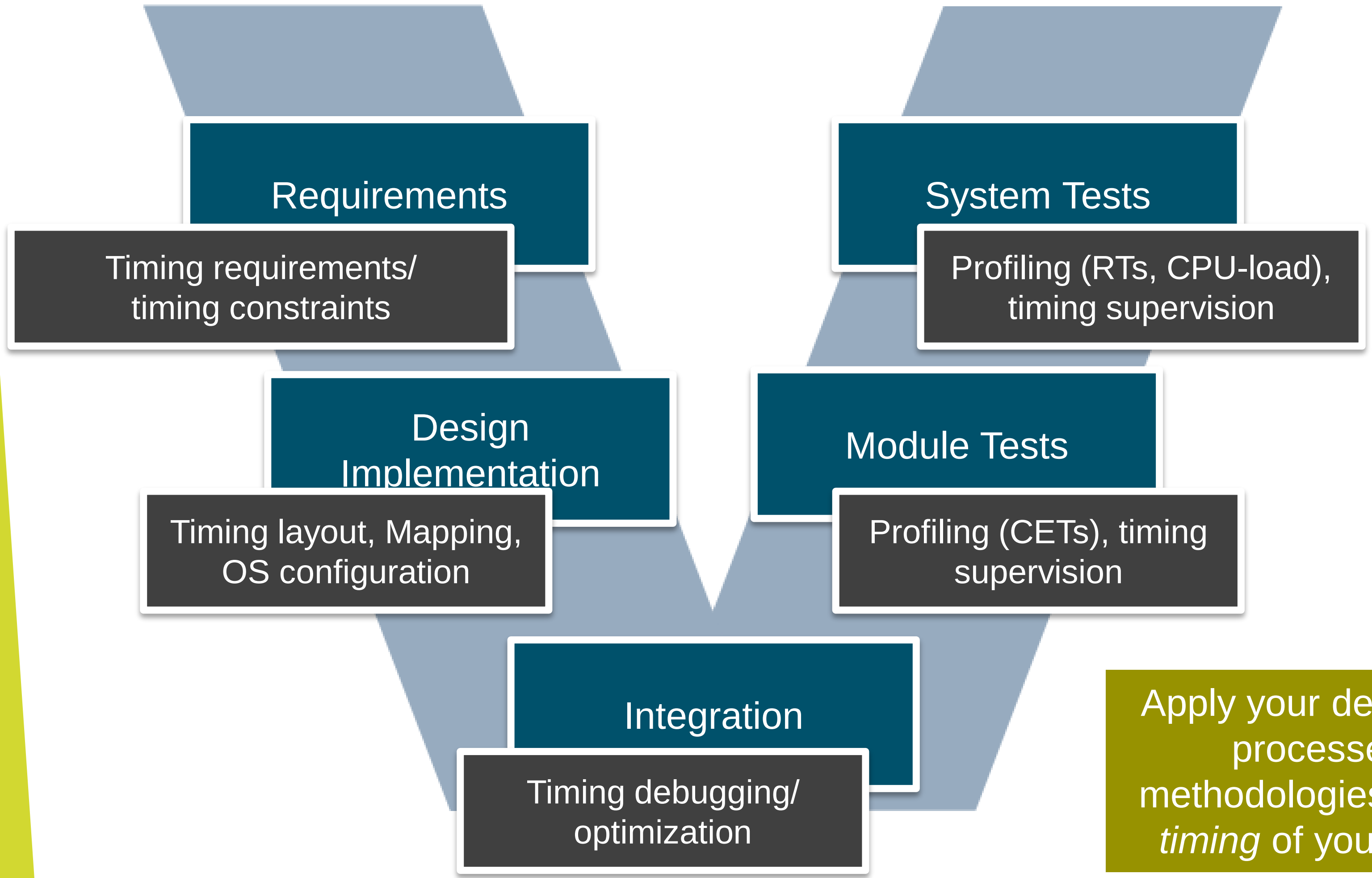
- Multicore
  - If your single-core timing is unstable already, multi-core will be a nightmare

Such indicator does not exist (unfortunately)

Single-core

Multi-core

Software Development Process


Requirements
Timing requirements/ timing constraints
Design Implementation
Timing layout, Mapping, OS configuration
Integration
Timing debugging/ optimization
Module Tests
Profiling (CETs), timing supervision
System Tests
Profiling (RTs, CPU-load), timing supervision

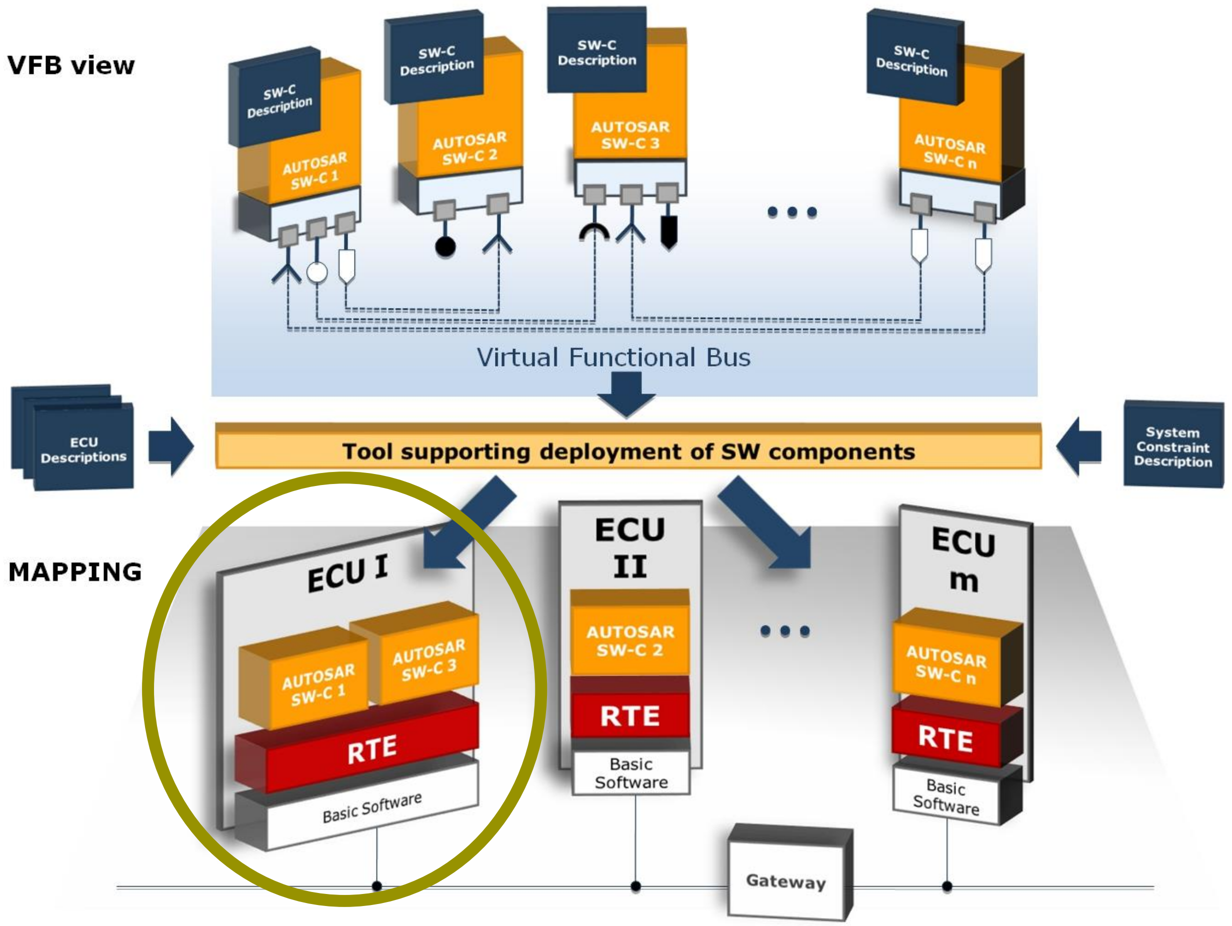
Apply your develeopment processes and methodologies also to the *timing* of your software!

Who is GLIWA embedded systems?

- Timing analysis and embedded software expertise since 2003
  - Headquarters located in Weilheim near Munich; GLIWA Ltd. in York
  - 35 employees, many timing experts
  - Average annual growth over the past 7 years: **27,5%**
- Stack Analysis combining static and dynamic methods

AUTOSAR Timing Extensions (TIMEX)

AUTOSAR overview

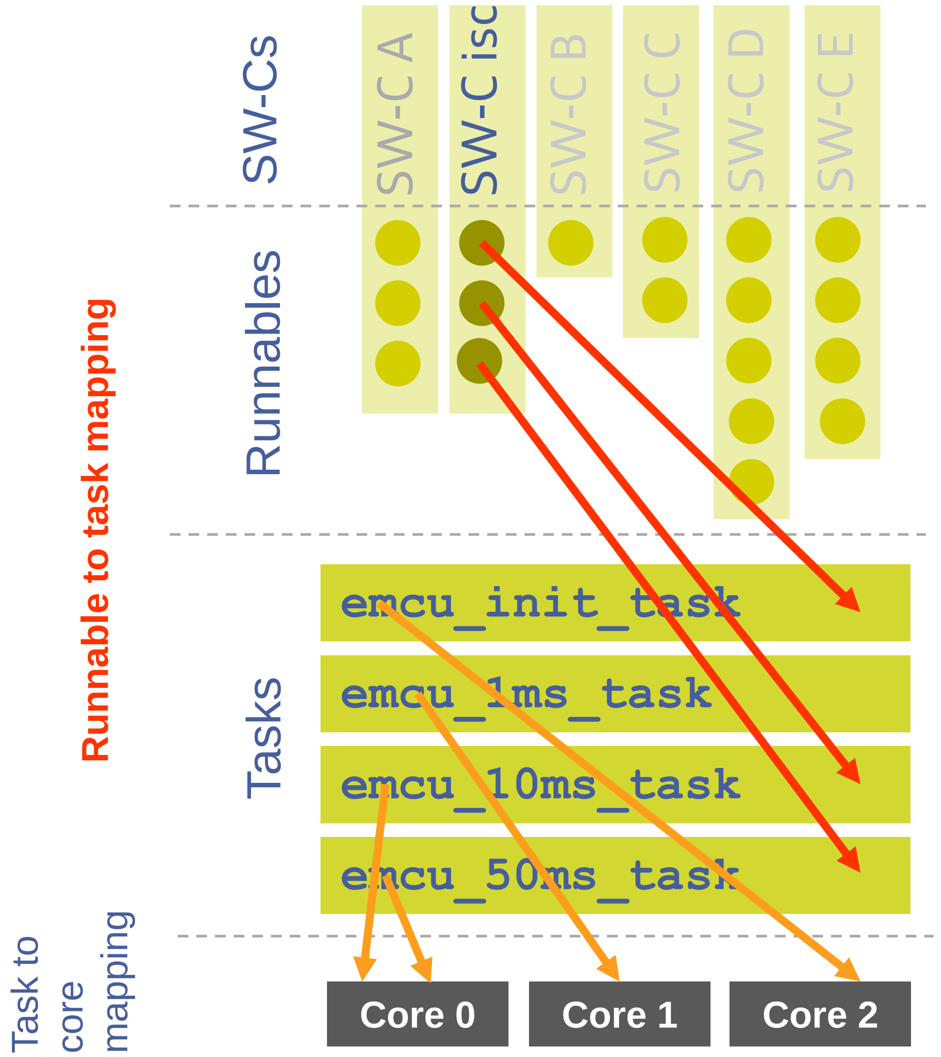


Let‘s focus on one ECU (“EcuTiming”)

Source: autosar.org

AUTOSAR: ECU example

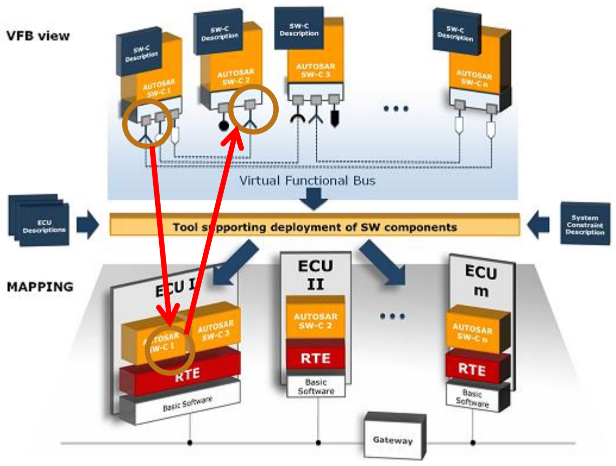
- The SW-C “idle speed control” of an engine management ECU is coded in three runnables:
  - IdleSpeedInit
  - IdleSpeed10ms
  - IdleSpeed50ms
- As part of the RTOS configuration, these get mapped to three different tasks which they share with many other runnables from other SW-Cs.
- For multi-core processors, tasks get also mapped to a certain core.

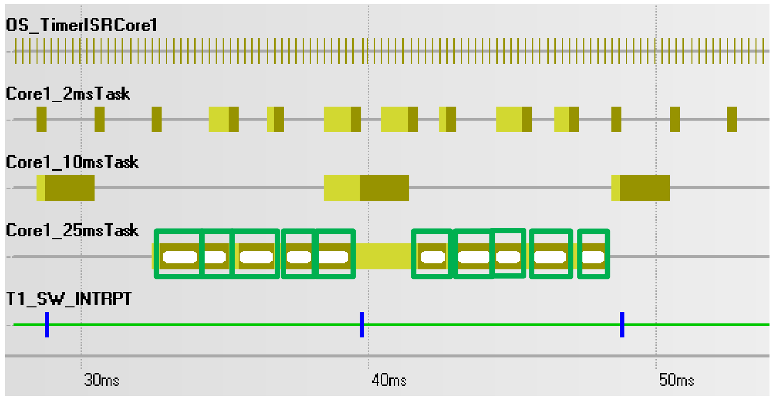


AUTOSAR Timing Extension (TIMEX)

- With AUTOSAR 4.0, the *Timing Extensions* were added allowing precise timing constraint specification.

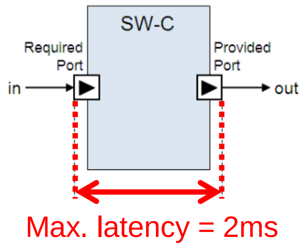
- Timing constraints can be applied to
  - Timing Description Events ○
  - Timing Description Event Chains →
  - ordered list of Executable Entities ▭

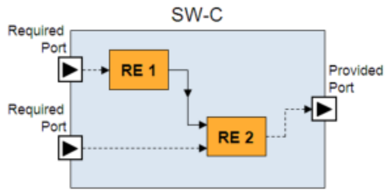


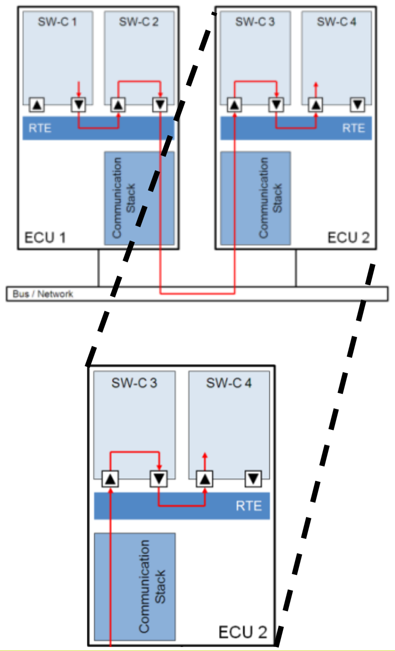


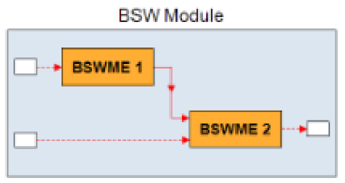
The views addressed by TIMEX

- **VfbTiming**
  timing related to interaction of SW-Cs at VFB level
- **SwcTiming**
  timing related to the internal behavior of atomic SW-Cs
- **SystemTiming**
  timing on system level incorporating readily configured ECUs, busses
- **BswModuleTiming**
  timing related to BSW module internal behavior
- **EcuTiming**
  timing related to everything inside one readily configured ECU









Source: Specification of Timing Extensions, autosar.org

TIMEX Constraint Types I

- **EventTriggeringConstraint**
  - Example use-case: supervise jitter
- **LatencyTimingConstraint**
  - Example use-case: avoid loss and duplication of data due to under- and oversampling and/or jitter
- **AgeConstraint**
  - Example use-case: make sure, data is not too old
- **SynchronizationTimingConstrain**
  - Example use-case: establish and maintain a consistent time base for the interaction between different subsystems

TIMEX Constraint Types II

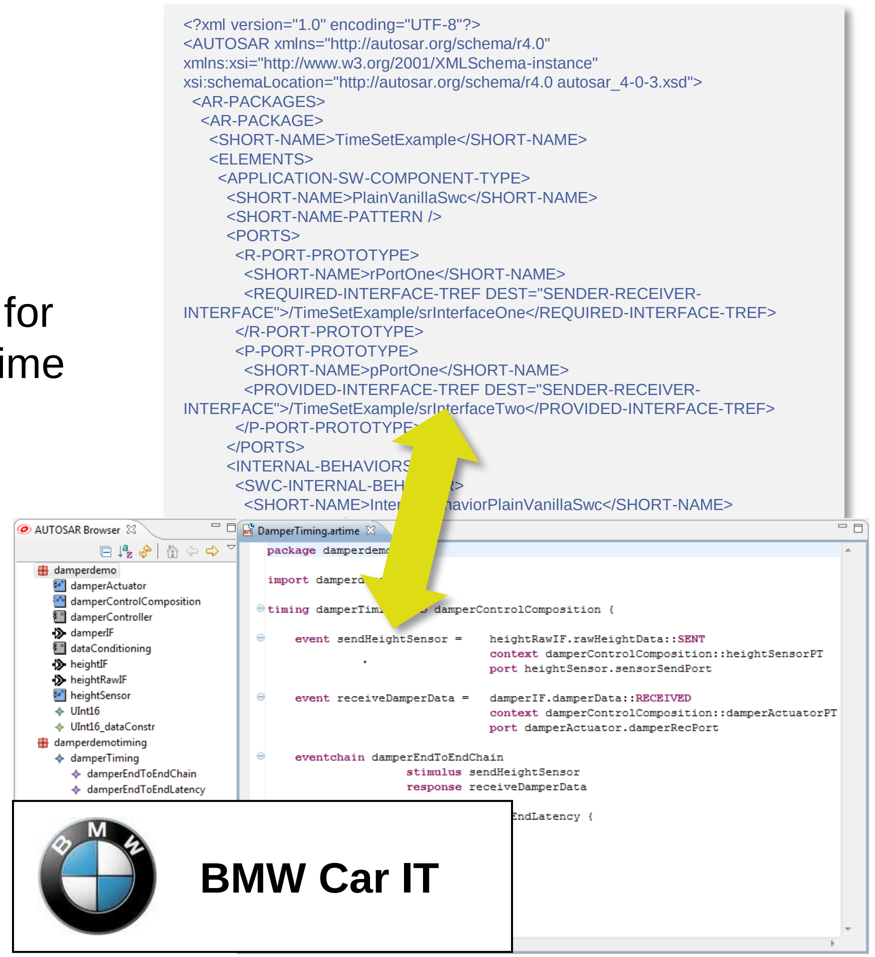
- **OffsetTimingConstraint**
  - Example use-case: bound the time offset between the occurrence of two arbitrary timing events

- **ExecutionOrderConstraint**
  - Example use-case: supervise the correct execution order of runnables

- **ExecutionTimeConstraint**
  - Example use-case: specify the maximum allowed run-time budget of a process

How to use TIMEX

- Option 1: write TIMEX directly i.e. ARXML
  - *Well, this certainly is no fun.*
- Option 2: use tool support
  - ArTime from BMW Car IT: language for specifying TIMEX requirements. ArTime is an Artop plug-in
  - In-house tools
  - Others (not widely used though)
  - *Well, there is much room for improvement, right?*
- Option 3: ask experts
  - *Isn't that frustrating?*
- Option 4: give up
  - *Yes, it is!*



```
<?xml version="1.0" encoding="UTF-8"?>
<AUTOSAR xmlns="http://autosar.org/schema/r4.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://autosar.org/schema/r4.0 autosar_4-0-3.xsd">
 <AR-PACKAGES>
  <AR-PACKAGE>
   <SHORT-NAME>TimeSetExample</SHORT-NAME>
   <ELEMENTS>
    <APPLICATION-SW-COMPONENT-TYPE>
     <SHORT-NAME>PlainVanillaSwc</SHORT-NAME>
     <SHORT-NAME-PATTERN />
     <PORTS>
      <R-PORT-PROTOTYPE>
       <SHORT-NAME>rPortOne</SHORT-NAME>
       <REQUIRED-INTERFACE-TREF DEST="SENDER-RECEIVER-
INTERFACE">/TimeSetExample/srInterfaceOne</REQUIRED-INTERFACE-TREF>
      </R-PORT-PROTOTYPE>
      <P-PORT-PROTOTYPE>
       <SHORT-NAME>pPortOne</SHORT-NAME>
       <PROVIDED-INTERFACE-TREF DEST="SENDER-RECEIVER-
INTERFACE">/TimeSetExample/srInterfaceTwo</PROVIDED-INTERFACE-TREF>
      </P-PORT-PROTOTYPE
     </PORTS>
     <INTERNAL-BEHAVIORS
      <SWC-INTERNAL-BEH        R>
       <SHORT-NAME>Inter       haviorPlainVanillaSwc</SHORT-NAME>
```

```
package damperdem
import damperd
timing damperTim       damperControlComposition {

    event sendHeightSensor =    heightRawIF.rawHeightData::SENT
                                context damperControlComposition::heightSensorPT
                                port heightSensor.sensorSendPort

    event receiveDamperData =   damperIF.damperData::RECEIVED
                                context damperControlComposition::damperActuatorPT
                                port damperActuator.damperRecPort

    eventchain damperEndToEndChain
                        stimulus sendHeightSensor
                        response receiveDamperData

                                EndLatency {
```

Timing requirements

Requirements specification documents

- A good ECU's requirements specification is the foundation for sound and safe timing.

- Two types of timing-related requirements should be addressed:
  - Dedicated timing requirements (as far as they are known)
  - Requirements regarding the environment, methodologies and tools

timing requirements

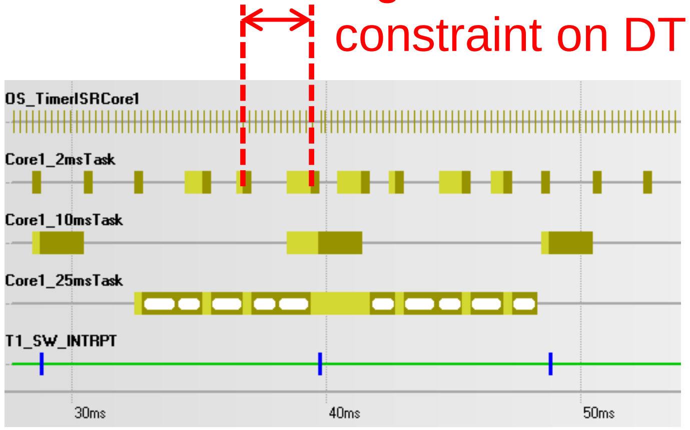
methodologies & tools requirements

Collection of typical timing requirements

- Max. allowed CPU-load
  - Present in most specs already
  - Specify *how* it is calculated (which observation frame T^O to be used)!
  - Remark: Cannot be specified using TIMEX
- Start-up time ("presence on the bus")
- End-to-end latency
- Data-age
- Max. CET for TASKs, ISRs, runnables
  - Think of *budgeting your timing*!
    → scheduling simulation
- Response-times (at least RT < period)
- Jitter (max. deviation from targeted period)

$$U^o = \frac{\sum_{n=1}^{N} CET_n^o}{T^o}$$

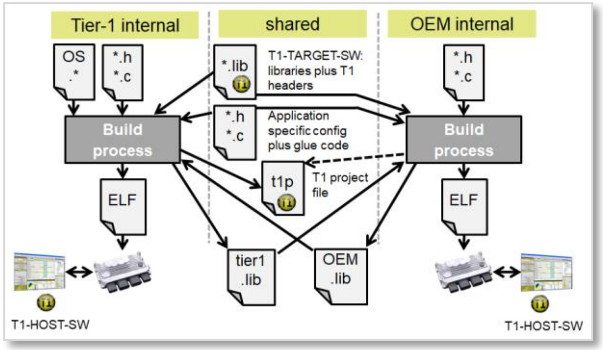
$$U^{max} = \max(U^o)$$



Fall-back when TIMEX not appropriate: constraints on timing parameters

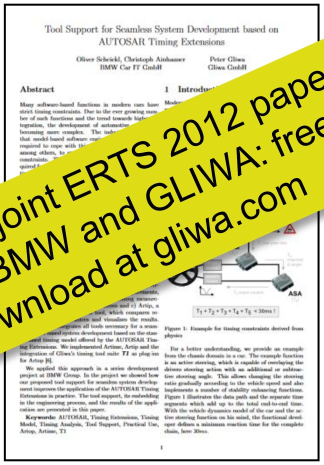
Collection of typical tool requirements

- Feature-set
  - Scheduling-simulation, -analysis
  - Profiling
  - Tracing
    - Synchronized traces from all cores
    - Runnables, functions, any code, data-flow, etc. without rebuilding the software
    - etc.
- Tool availability (OEM, tier-1, both, anybody, …)
  - Inhouse tools not appropriate
  - Make sure, all relevant data can be exchanged
- Scope
  - Scheduling-simulation to the detail-level of…
  - Timing verification
    - With automated HIL tests
    - In the car



BMW: TIMEX in mass-production

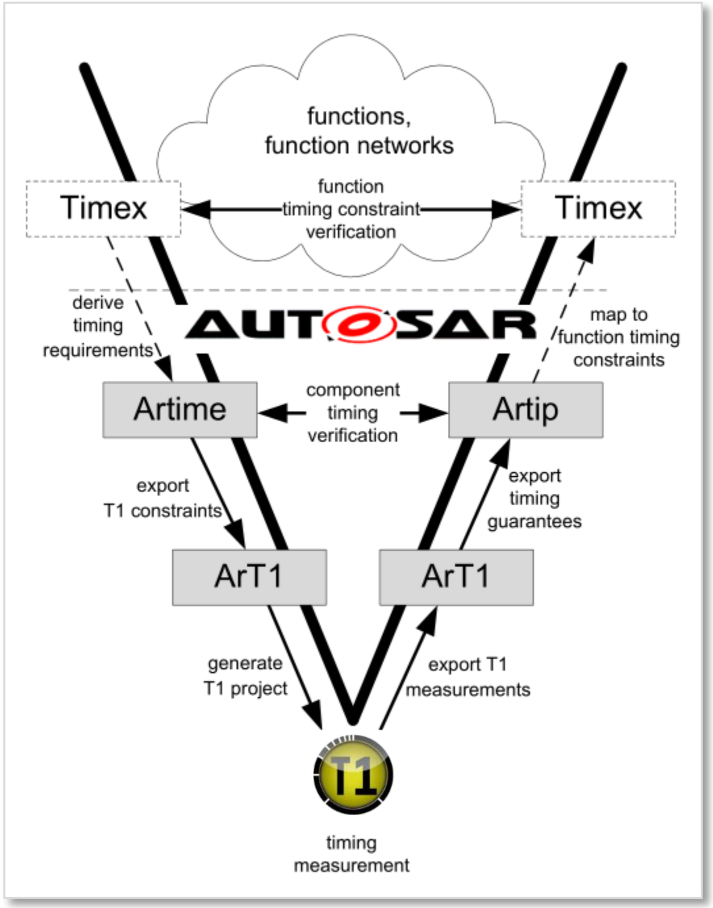
- In 2009, BMW used TIMEX/ArTime for a chassis ECU
  - Formal specification of timing requirements
  - Seamless interface specification ⬄ verification:
    timing requirements were imported and then verified by T1
  - Textual specification of requirements regarding methodologies & tools



Tool Support for Seamless System Development based on AUTOSAR Timing Extensions

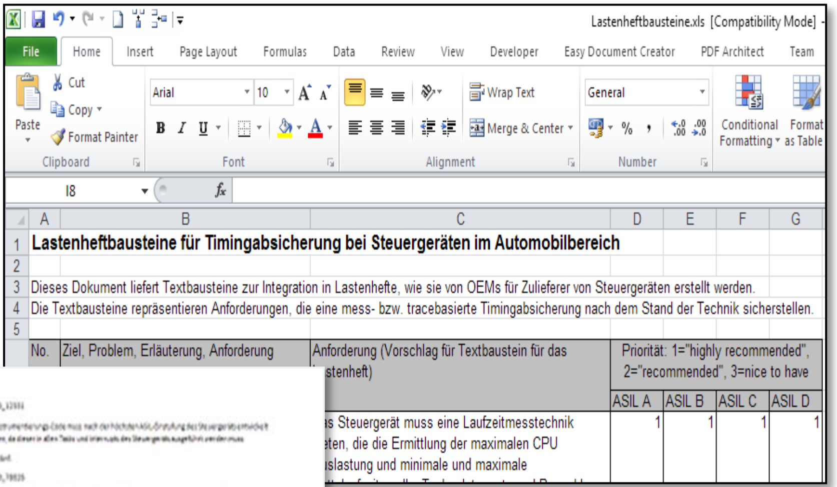
Oliver Scheickl, Christoph Ainhauser, BMW Car IT GmbH
Peter Gliwa, Gliwa GmbH

See joint ERTS 2012 paper by BMW and GLIWA: free download at gliwa.com



Templates for requirements specifications

- Top-up result: text templates for requirements specifications
  - Reuse generic requirements in future projects
- 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…



	A	B	C	D	E	F	G
1	**Lastenheftbausteine für Timingabsicherung bei Steuergeräten im Automobilbereich**						
2							
3	Dieses Dokument liefert Textbausteine zur Integration in Lastenhefte, wie sie von OEMs für Zulieferer von Steuergeräten erstellt werden.						
4	Die Textbausteine repräsentieren Anforderungen, die eine mess- bzw. tracebasierte Timingabsicherung nach dem Stand der Technik sicherstellen.						
5							
	No.	Ziel, Problem, Erläuterung, Anforderung	Anforderung (Vorschlag für Textbaustein für das …stenheft)	Priorität: 1="highly recommended", 2="recommended", 3=nice to have			
				ASIL A	ASIL B	ASIL C	ASIL D
			…s Steuergerät muss eine Laufzeitmesstechnik …ten, die die Ermittlung der maximalen CPU …slastung und minimale und maximale	1	1	1	1

[illegible]

Template

Actual requirements specification

Tracing:
Timing verification

CanInterrupt
dr_fastest
Mo_Alarm
CAN_TX_mode_10ms
CAN_TX_mode_2ms
dr_VCU_10ms
MoCMEM_Cyclic
dr_10ms
dr_EMU_10ms
dr_VCU_20ms
48ms 000us
50ms
52ms
54ms
56ms

(Scheduling-) tracing vs. timing measurement

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

“ ”

- Scheduling 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
    - On-chip scheduling tracing (future)

“ ”

- Profiling
  - The process of collecting timing parameters

ARTI

- 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:

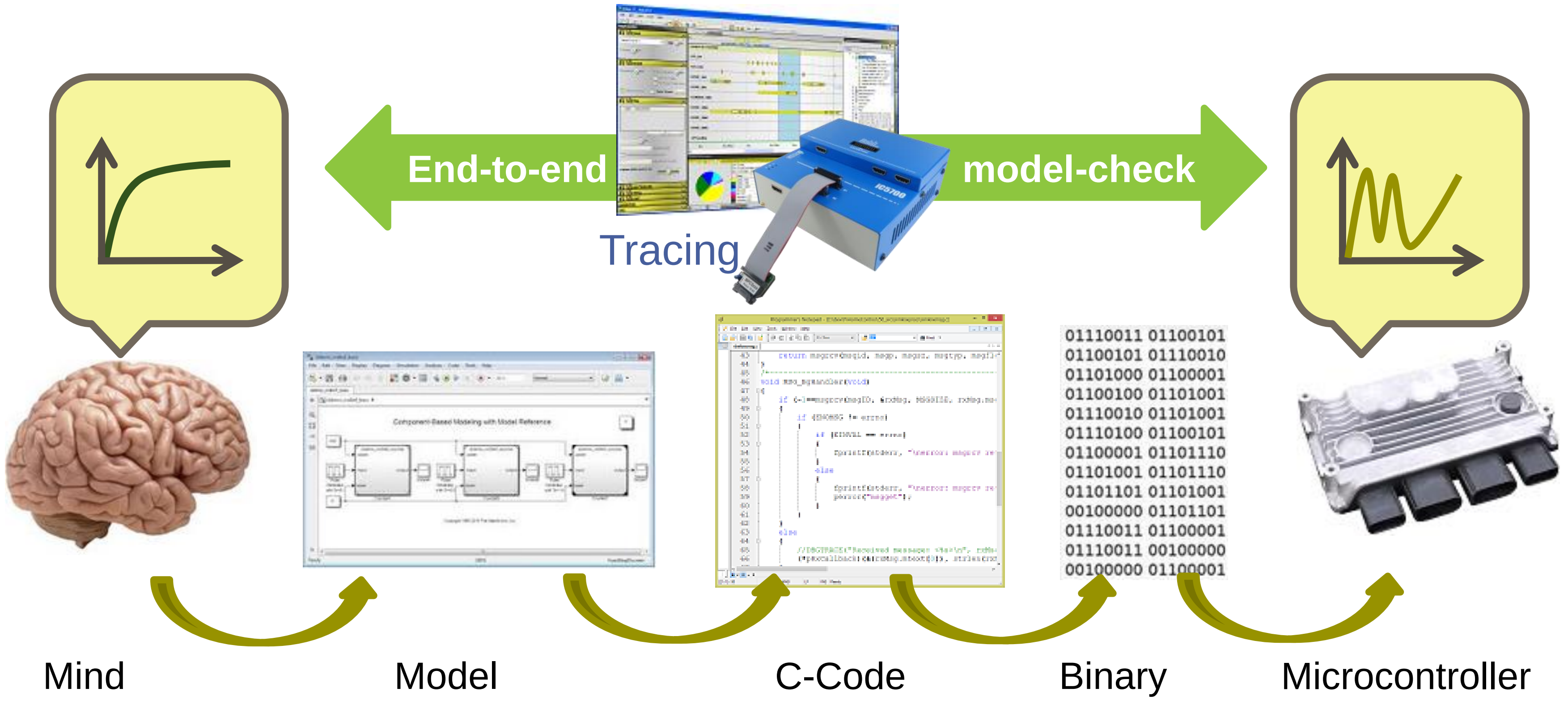
  ARTI (“AUTOSAR Run-time interface”)

  Goal: ORTI successor within AUTOSAR *plus* efficient support of

  - instrumentation based tracing
  - non-intrusive on-chip scheduling tracing
  - multi-core
  - runnables
  - AUTOSAR AP

Tracing: End-to-end model-check

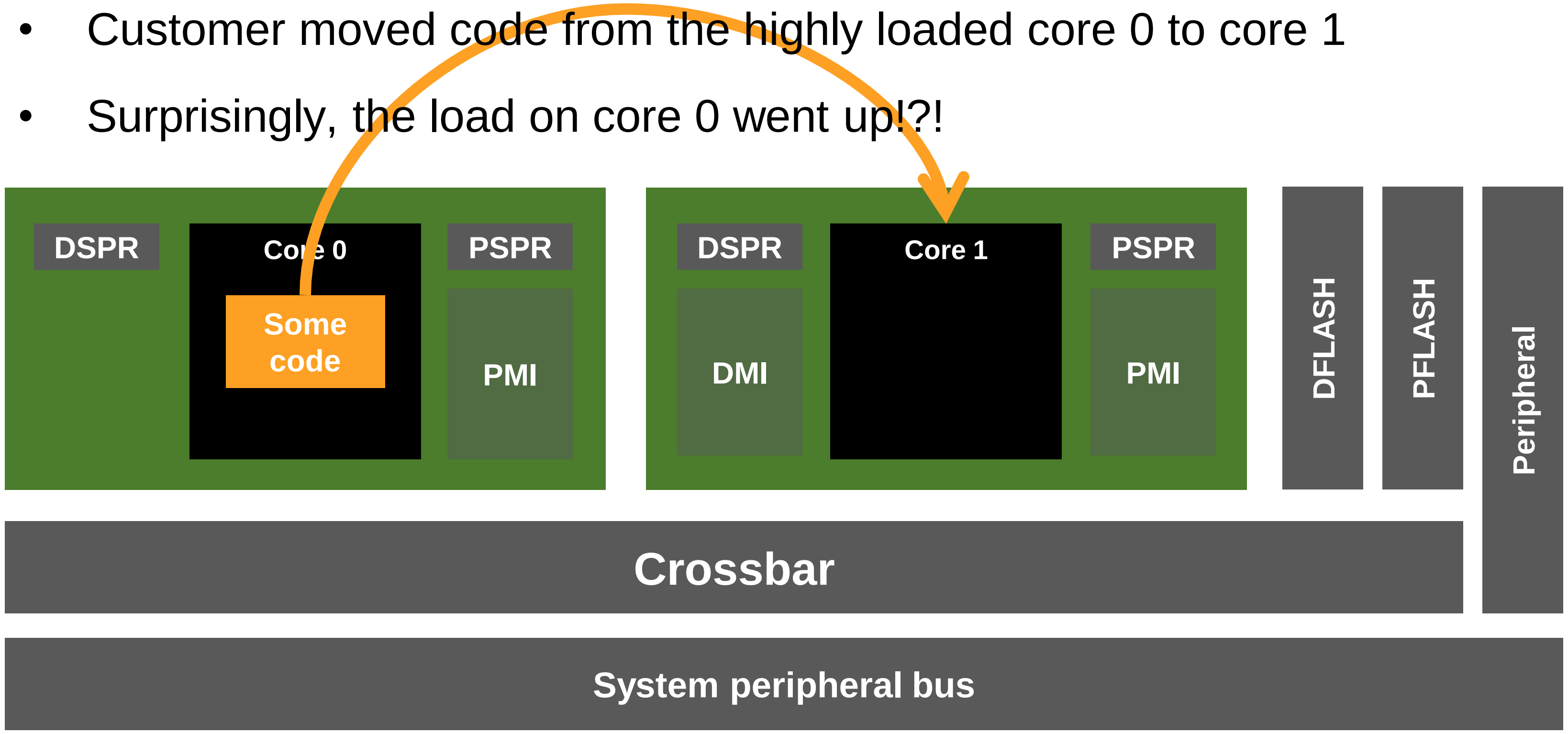
- On its way from the **mind** to the **microcontroller**, an **idea** can suffer from **transition-errors.**
- Tracing allows an **end-to-end model-check**.



```
01110011 01100101
01100101 01110010
01101000 01100001
01100100 01101001
01110010 01101001
01110100 01100101
01100001 01101110
01101001 01101110
01101101 01101001
00100000 01101101
01110011 01100001
01110011 00100000
00100000 01100001
```

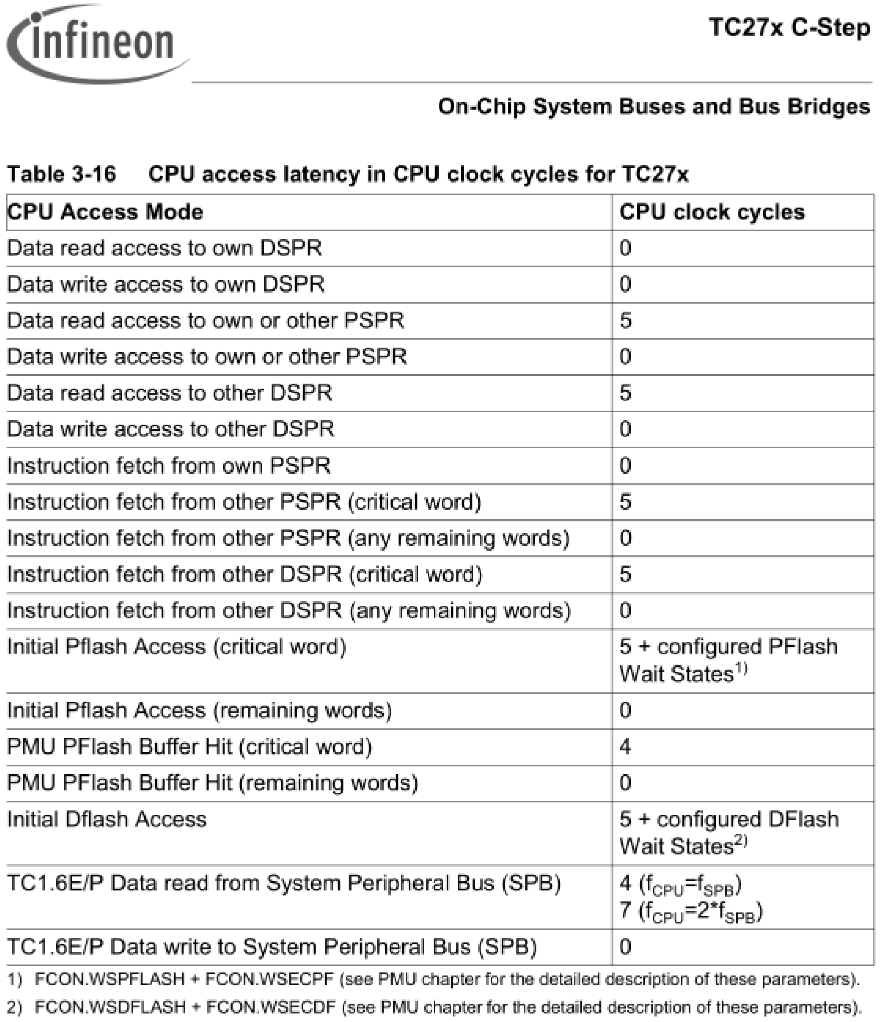
Multi-core example of missing verification

- Customer moved code from the highly loaded core 0 to core 1
- Surprisingly, the load on core 0 went up!?!



DSPR = data scratch pad RAM	DMI = data memory interface
PSPR = program scratch pad RAM	PMI = programmemory interface

Memory read access times: AURIX™ manual



TC27x C-Step

On-Chip System Buses and Bus Bridges

Table 3-16 CPU access latency in CPU clock cycles for TC27x

CPU Access Mode	CPU clock cycles
Data read access to own DSPR	0
Data write access to own DSPR	0
Data read access to own or other PSPR	5
Data write access to own or other PSPR	0
Data read access to other DSPR	5
Data write access to other DSPR	0
Instruction fetch from own PSPR	0
Instruction fetch from other PSPR (critical word)	5
Instruction fetch from other PSPR (any remaining words)	0
Instruction fetch from other DSPR (critical word)	5
Instruction fetch from other DSPR (any remaining words)	0
Initial Pflash Access (critical word)	5 + configured PFlash Wait States[1)]
Initial Pflash Access (remaining words)	0
PMU PFlash Buffer Hit (critical word)	4
PMU PFlash Buffer Hit (remaining words)	0
Initial Dflash Access	5 + configured DFlash Wait States[2)]
TC1.6E/P Data read from System Peripheral Bus (SPB)	4 ($f_{CPU}=f_{SPB}$) 7 ($f_{CPU}=2*f_{SPB}$)
TC1.6E/P Data write to System Peripheral Bus (SPB)	0

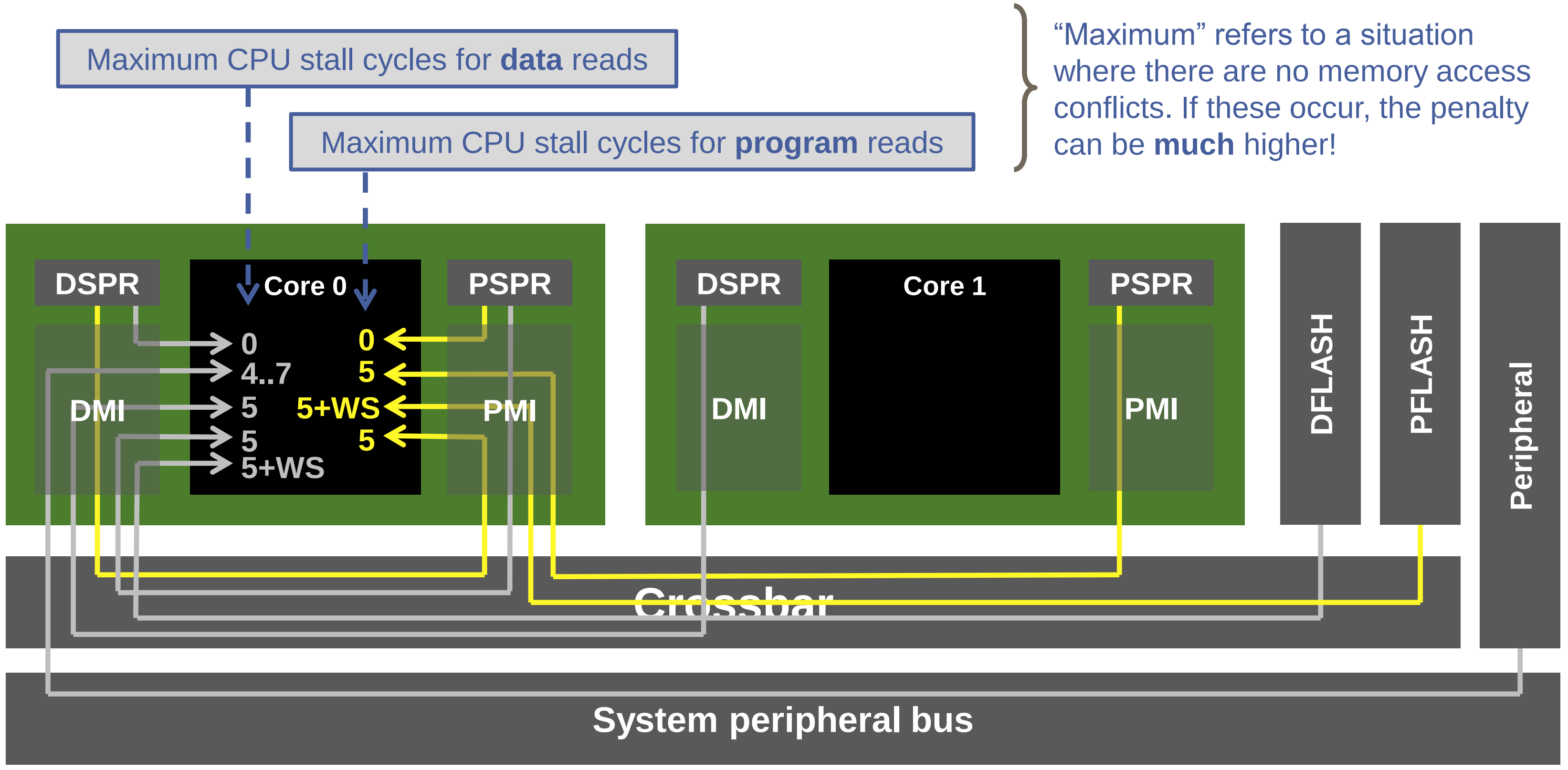
1) FCON.WSPFLASH + FCON.WSECPF (see PMU chapter for the detailed description of these parameters).

2) FCON.WSDFLASH + FCON.WSECDF (see PMU chapter for the detailed description of these parameters).

On Chip Bus Access Times

The table describes the CPU access times in CPU clock cycles for the TC27x. The access times are described as maximum CPU stall cycles where e.g. a data access to the local DSPR results in zero stall cycles. Pls. note that the CPU does not always immediately stall after the start of a data read from another SPR due to instruction pipelining effects. This means that the average number will be below the here shown numbers.

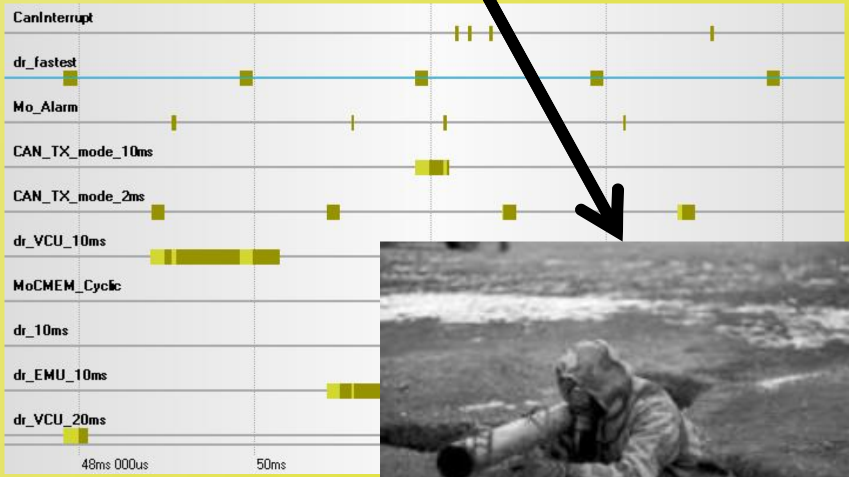
Multi-core example of missing verification



"Maximum" refers to a situation where there are no memory access conflicts. If these occur, the penalty can be **much** higher!

⟶	data read access	DSPR = data scratch pad RAM	DMI = data memory interface
⟶	program read access	PSPR = program scratch pad RAM	PMI = programmemory interface

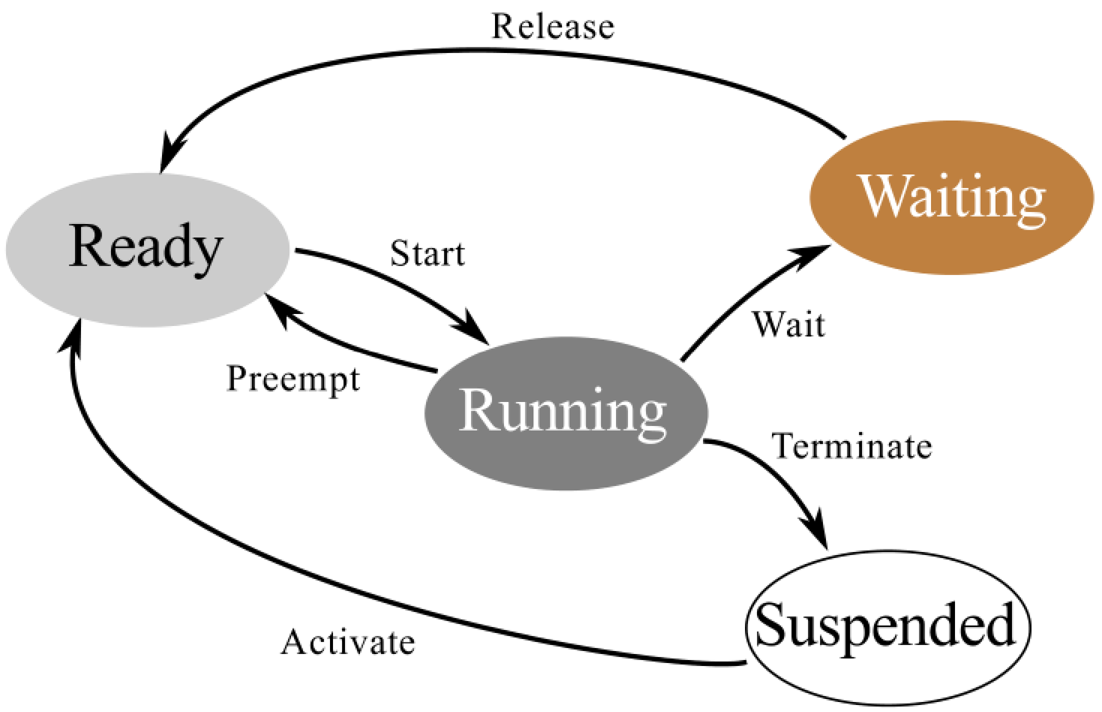
Mass-production projects



A report from the frontline

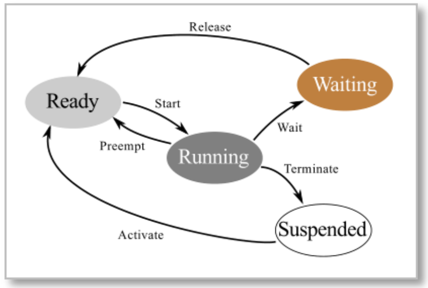
(plus recommendations)

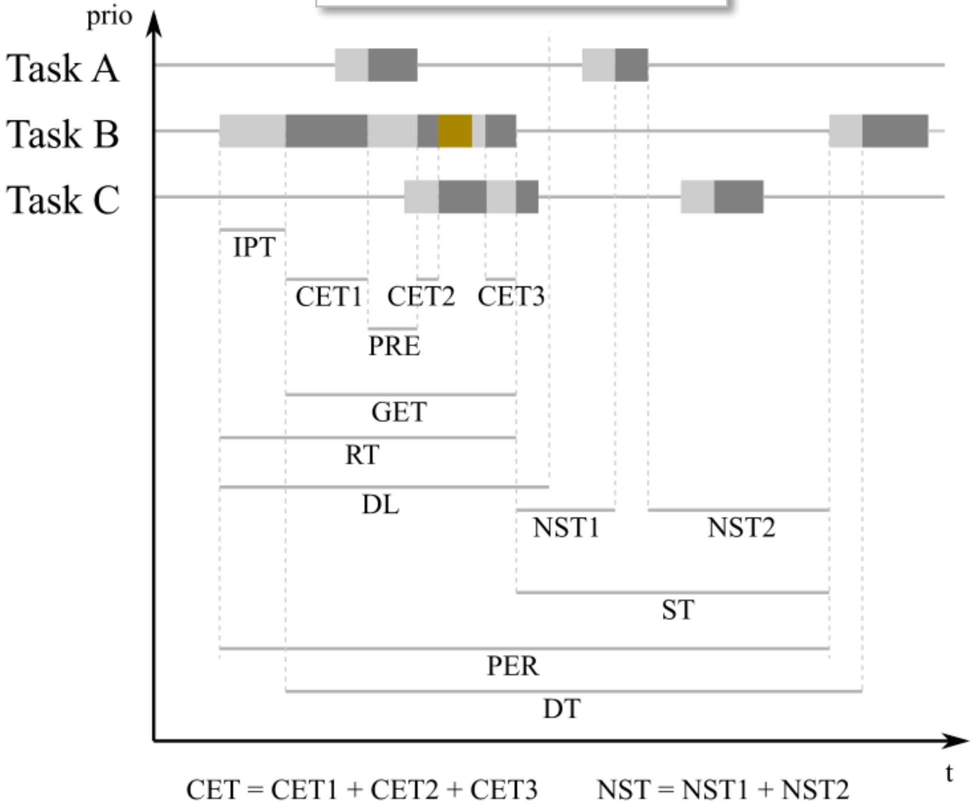
OSEK / AUTOSAR task states



ECC = **E**xtended **C**onformance **C**lass
TASK = container for code, e.g. runnables

Timing parameters





Abr.	Explanation (EN)
IPT	initial pending time
CET	core execution time
GET	gross execution time
RT	response time
DT	delta time
PER	period
ST	slack time
PRE	preemption
JIT	jitter
CPU	cpu load
DL	Deadline
NST	Net slack time

Typical ECC usage by the RTE

```
TASK(Task_B)
{
  EventMaskType ev;
  for(;;)
  {
    (void)WaitEvent(    Rte_Ev_Cyclic2_Task_B_0_10ms |
                        Rte_Ev_Cyclic2_Task_B_0_5ms );

    (void)GetEvent(Task_B, &ev);

    (void)ClearEvent(ev & ( Rte_Ev_Cyclic2_Task_B_0_10ms |
                            Rte_Ev_Cyclic2_Task_B_0_5ms ));

    if ((ev & Rte_Ev_Cyclic2_Task_B_0_10ms) != (EventMaskType)0)
    {
      CanNm_MainFunction();
      CanSM_MainFunction();
    }

    if ((ev & Rte_Ev_Cyclic2_Task_B_0_5ms) != (EventMaskType)0)
    {
      CanTp_MainFunction();
      CanXcp_MainFunction();
    }
}
```

Typical ECC usage by the RTE

Users want to see the body of the loop as one occurrence of Task_B

```
TASK(Task_B)
{
  EventMaskType ev;
  for(;;)
  {
    (void)WaitEvent(    Rte_Ev_Cyclic2_Task_B_0_10ms |
                        Rte_Ev_Cyclic2_Task_B_0_5ms );

    (void)GetEvent(Task_B, &ev);

    (void)ClearEvent(ev & ( Rte_Ev_Cyclic2_Task_B_0_10ms |
                            Rte_Ev_Cyclic2_Task_B_0_5ms ));

    if ((ev & Rte_Ev_Cyclic2_Task_B_0_10ms) != (EventMaskType)0)
    {
      CanNm_MainFunction();
      CanSM_MainFunction();
    }

    if ((ev & Rte_Ev_Cyclic2_Task_B_0_5ms) != (EventMaskType)0)
    {
      CanTp_MainFunction();
      CanXcp_MainFunction();
    }
  }
}
```

```
(void)WaitEvent( Can_Ev_TriggerSM_Task_B );
(void)GetEvent(Task_B, &ev);
(void)ClearEvent(ev & ( Can_Ev_TriggerSM_Task_B ));
```

Task starts here

?? CET ??

Non terminating ECC task

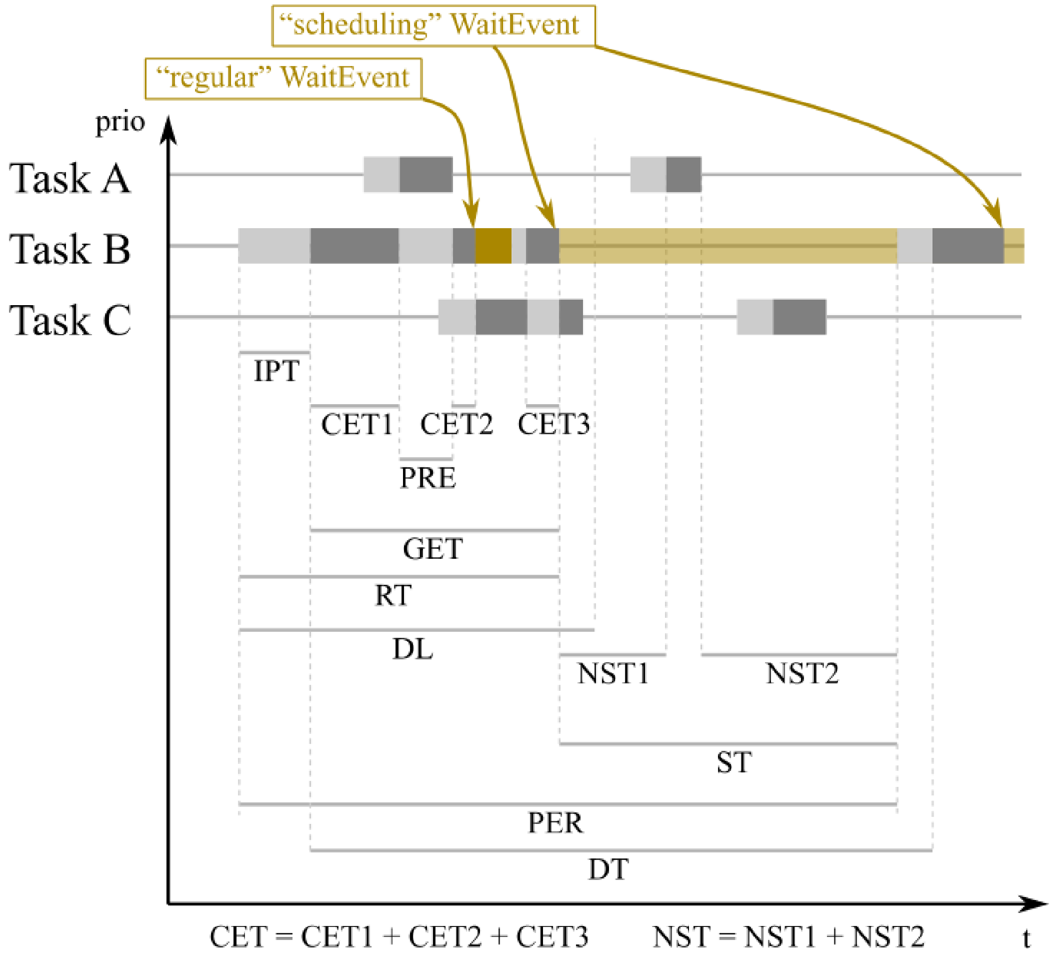
Task “ends” here (in fact it switches to state *waiting*)

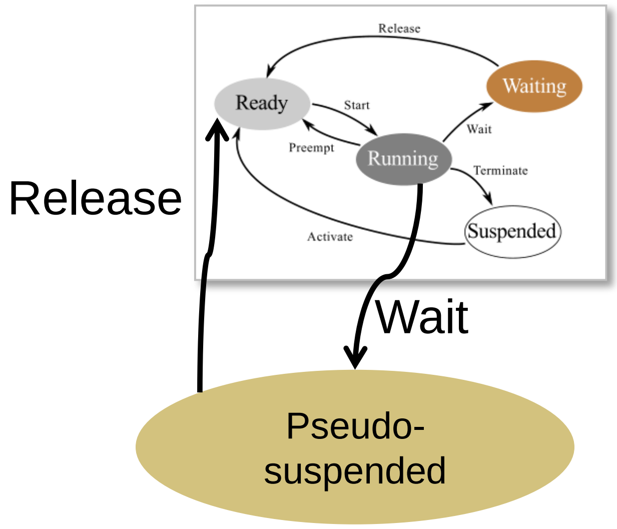
Task “restarts” here (in fact it switched from *waiting* via *ready* to *running*)

“scheduling” WaitEvent

“regular” WaitEvent

Previous run-time situation plus “scheduling WaitEvent”





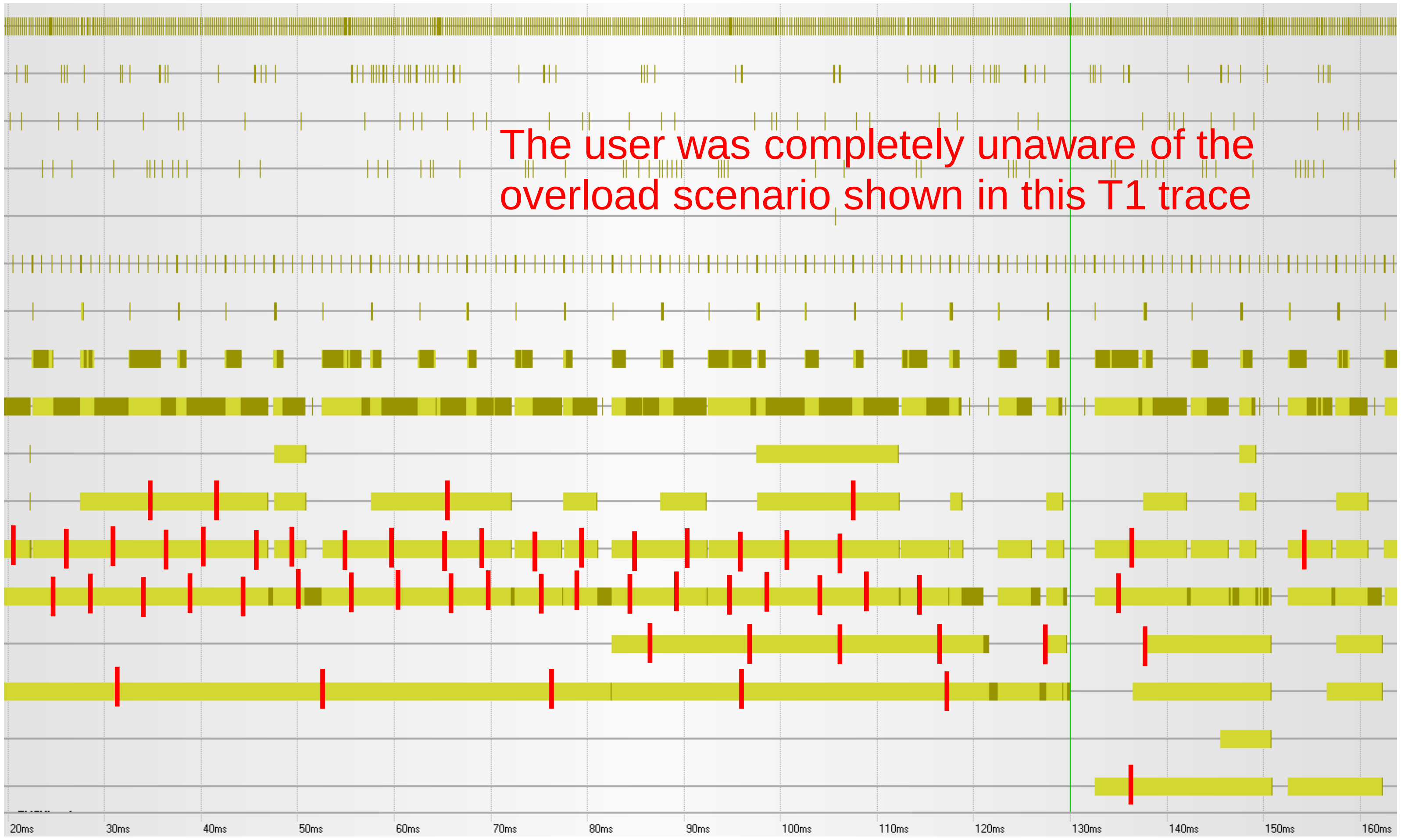
Houston, we have a problem

- Since 15 years, GLIWA serves as a team of firefighters for timing-related problems

- More and more often we see the same problematic OS configuration:
  - non-terminating ECC task
  - a second layer of scheduling

  → unnecessary complexity violating the "keep it simple" rule

Recent example (there are many alike)



ErrorHook useless for this setup

```
void ErrorHook(StatusType status)
{
  switch(status) {
    case E_OS_LIMIT:
      /* failed task activation
       * as a result of an overload
       * situation  */
      SystemReset();
      break;
    default:
      break;
  }
}
```

ErrorHook: called by the OS, implemented by the user (of the OS)

User’s intention: Reset when system is overloaded

BUT: ErrorHook does not get called when an event is re-triggered

Summary of disadvantages (of the ECC setup shown)

The setup with non-terminating ECC tasks adds a second layer of scheduling on top of the OS.

- No supervision through ErrorHook “E_OS_LIMIT”
- ECC in general requires more resources
  - more RAM
  - more Stack (a separate stack at least per prio, typically per task)
  - more run-time
- Difficult to analyze: ARTI will disallow a mixture of the two kinds of WaitEvent
- Increased complexity without any benefit → error-prone

Typical ECC usage by the RTE

Users want to see the body of the loop as one occurrence of Task_B

```
TASK(Task_B)
{
  EventMaskType ev;
  for(;;)
  {
    (void)WaitEvent(   Rte_Ev_Cyclic2_Task_B_0_10ms |
                       Rte_Ev_Cyclic2_Task_B_0_5ms

    (void)GetEvent(Task_B, &ev);

    (void)ClearEvent(ev & ( Rte_Ev_
                            Rt

    if ((ev & Rte_Ev_C                              e)0)
    {
      CanNm_MainFun
      CanSM_MainFunc
    }

    if ((ev & Rte_Ev_Cyc          _B_0_5ms) != (EventMaskType)0)
    {
      CanTp_MainFunction();
      CanXcp_MainFunction();
    }
}
```

Task starts here

?? CET ??

N rminating ECC task

ds” here (in fact it o state *waiting*)

starts” here (in it switched from *waiting* via *ready* to *running*)

WaitEvent

“regular” WaitEvent

```
(void)WaitEvent( Can_Ev_TriggerSM_Task_B );
(void)GetEvent(Task_B, &ev);
(void)ClearEvent(ev & ( Can_Ev_TriggerSM_Task_B ));
```

What a mess!

Recommended configuration

```
TASK(Task_B_10ms) // BCC1
{
  CanNm_MainFunction();
  CanSM_MainFunction();
  TerminateTask();
}

TASK(Task_B_5ms) // ECC
{
  EventMaskType ev;
  CanTp_MainFunction();
  // the following WaitEvent call is a "regular" WaitEvent
  (void)WaitEvent( Can_Ev_TriggerSM_Task_B );
  (void)GetEvent(Task_B, &ev);
  (void)ClearEvent(ev & ( Can_Ev_TriggerSM_Task_B ));
  CanXcp_MainFunction();
  TerminateTask();
}
```

Use BCC1 whenever possible

Conclusion

A CONCLUSION IS THE PLACE...
WHERE YOU GOT TIRED OF THINKING.

Conclusion

- AUTOSAR TIMEX is powerful, flexible and complicated
  - Tool-support for good usability yet to come

- Rather than *not* specifying any timing requirements
  - Define min./max. constraints on timing parameters
  - Use semantics provided by tools (e.g. TA Tool Suite)
  - Use informal description (text is better than nothing)

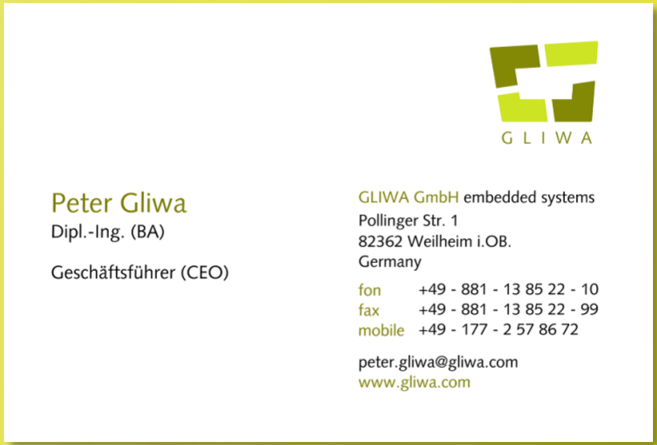
- Reuse your requirements: build a pool of requirement templates

- Verify your timing (requirements) with the real system!

- Get the single-core timing right before addressing multi-core.

- **Keep it simple!** Complexity increases the probability of things going wrong.

Thank you



GLIWA

Peter Gliwa
Dipl.-Ing. (BA)

Geschäftsführer (CEO)

GLIWA GmbH embedded systems
Pollinger Str. 1
82362 Weilheim i.OB.
Germany

fon	+49 - 881 - 13 85 22 - 10
fax	+49 - 881 - 13 85 22 - 99
mobile	+49 - 177 - 2 57 86 72

peter.gliwa@gliwa.com
www.gliwa.com