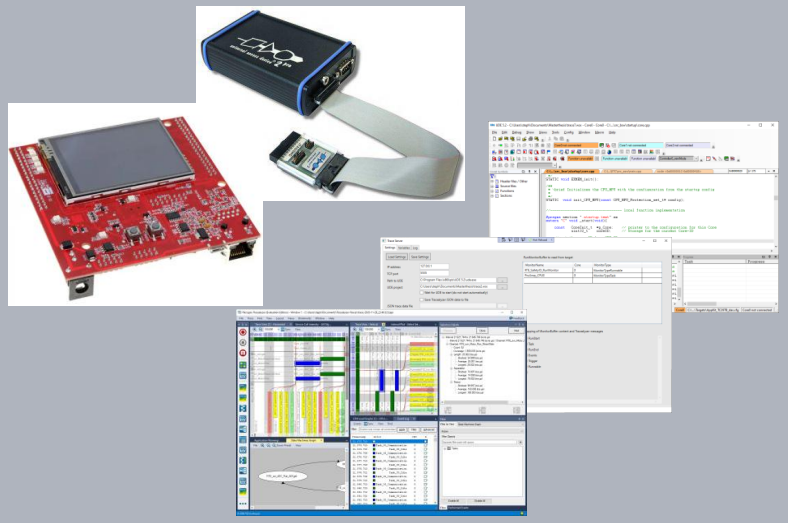
embeddedworld
Exhibition&Conference
... it´s a smarter world

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbeit
FACHBEREICH ELEKTROTECHNIK
UND INFORMATIONSTECHNIK

Control-Flow Evaluation on Multicore Controllers using Real-Time Trace Information

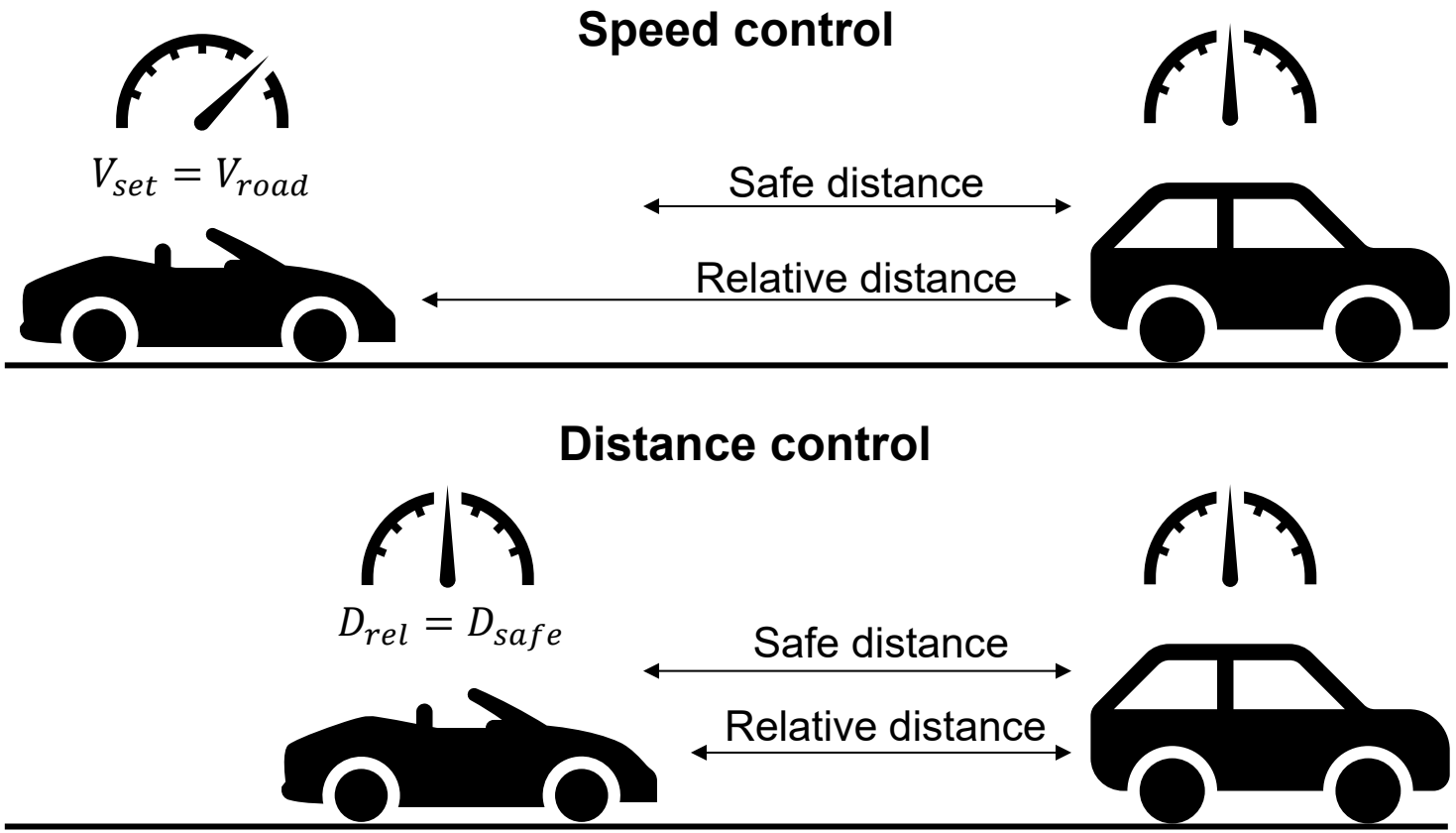


Stephan Radke
Thomas Barth
Prof. Dr-Ing. Peter Fromm

Department of Electrical Engineering
Hochschule Darmstadt – University of Applied Sciences

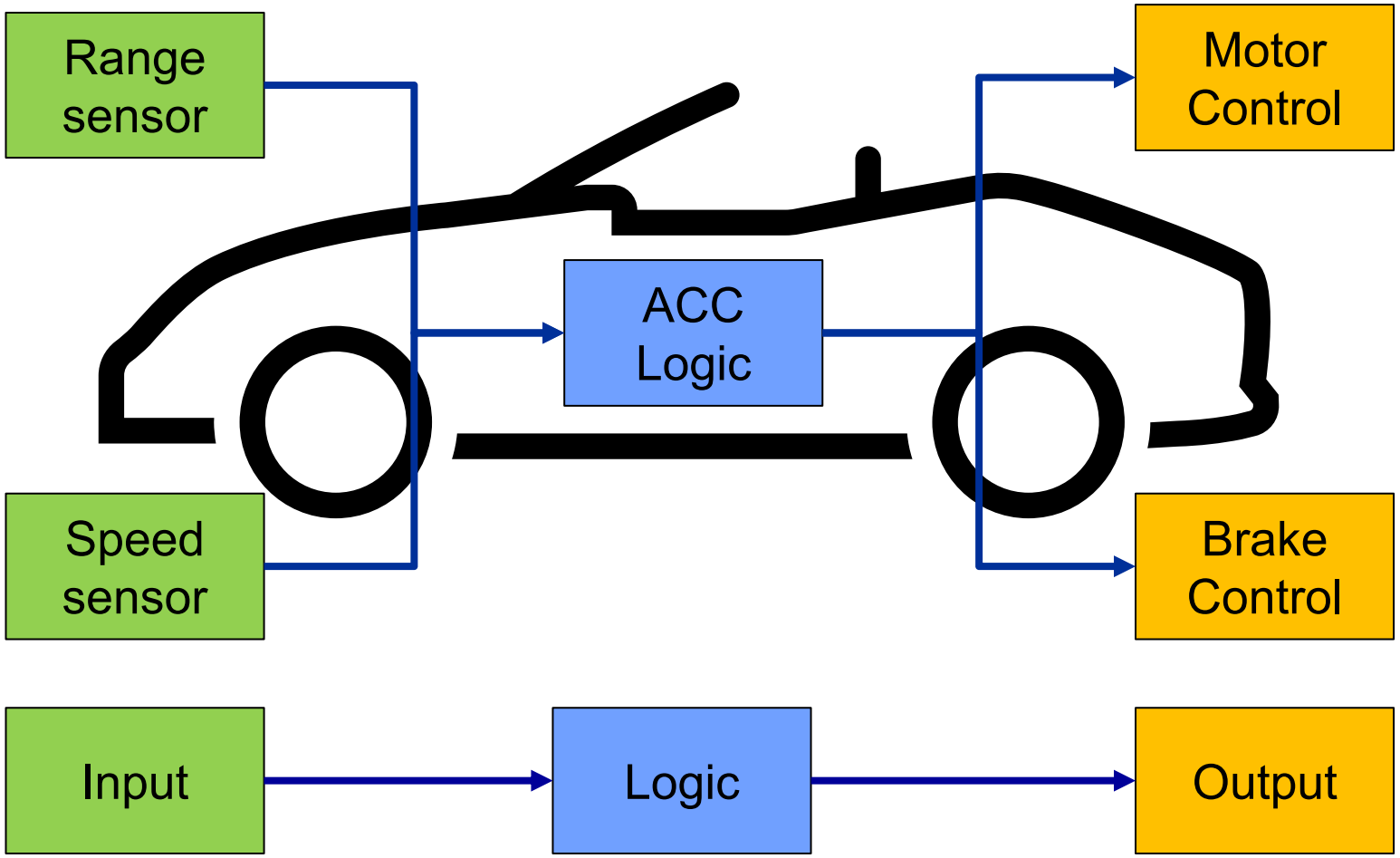
Timing matters

Adaptive Cruise Control (ACC) – Introductory Example

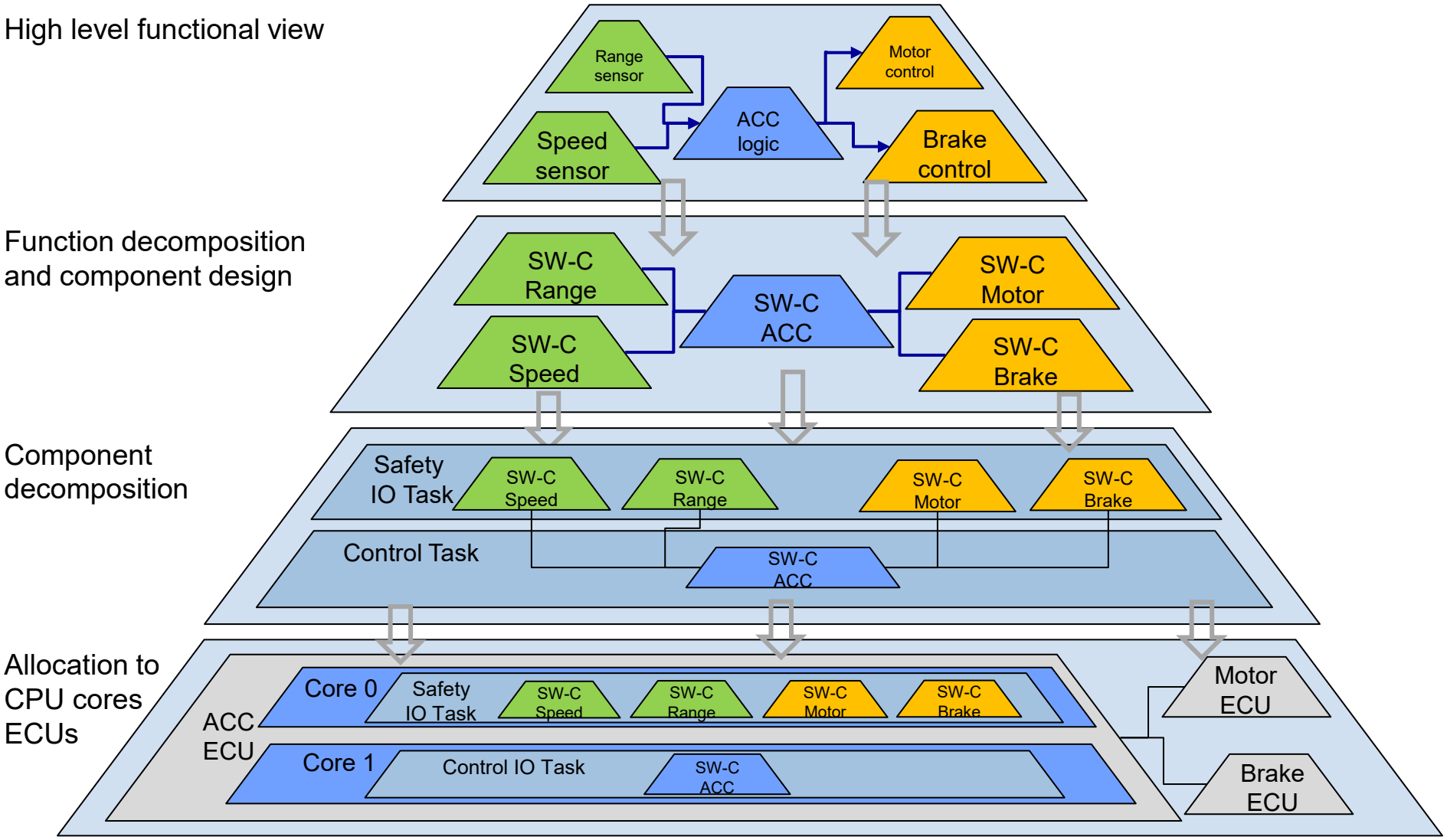


Timing matters (2)

Adaptive Cruise Control (ACC) – Introductory Example

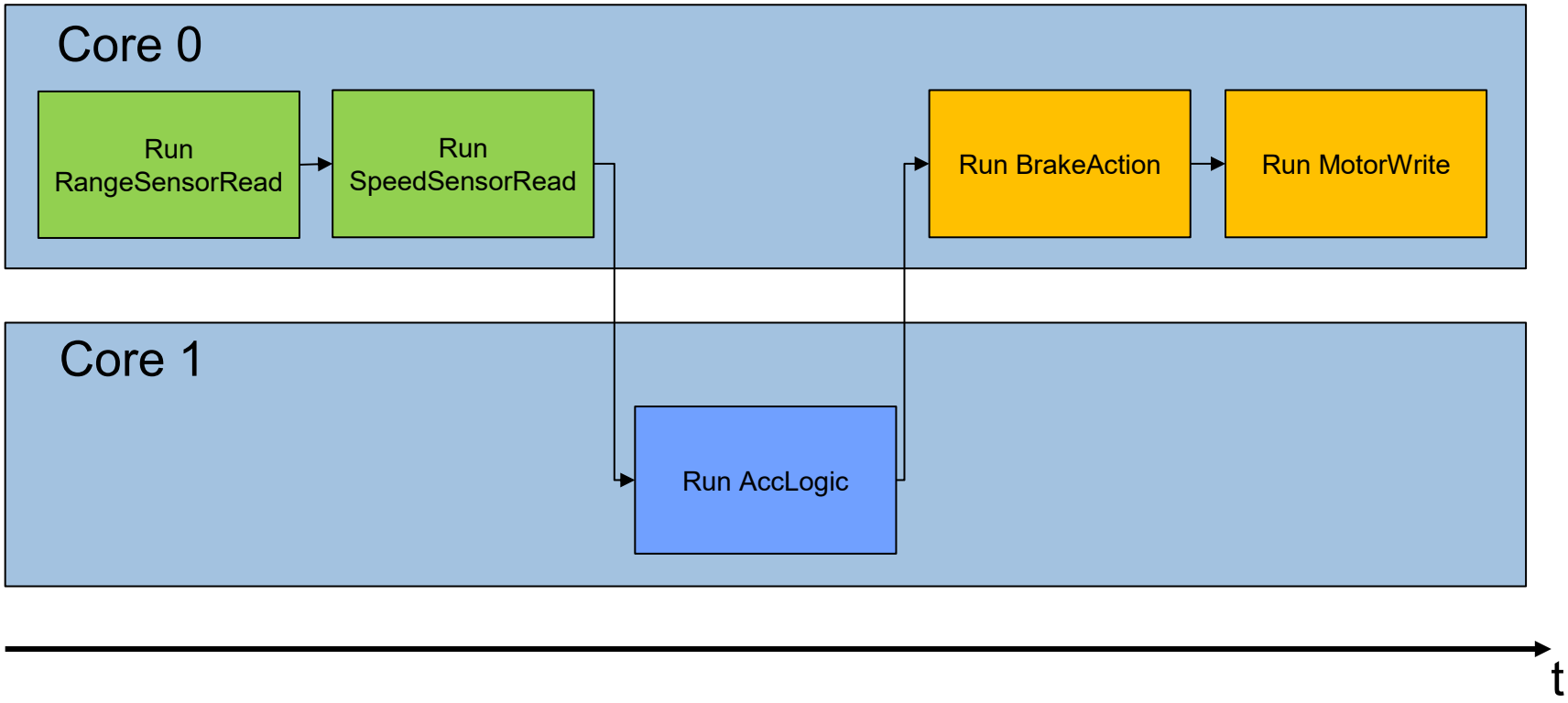


Timing matters (3)



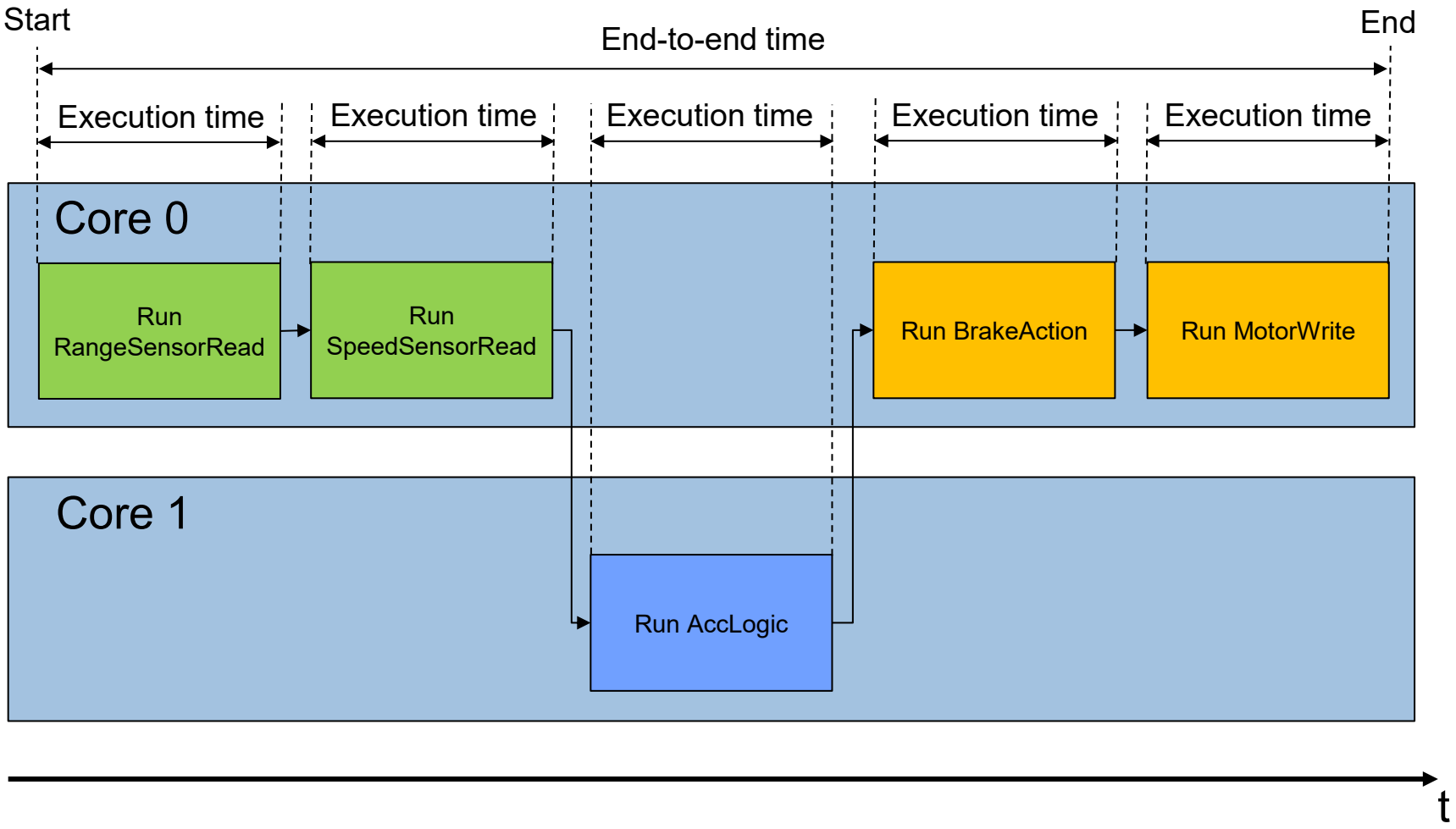
Timing issues and Control-Flows

Control-Flows describe the internal and external behavior of a system
They are implemented by sequences of runnables



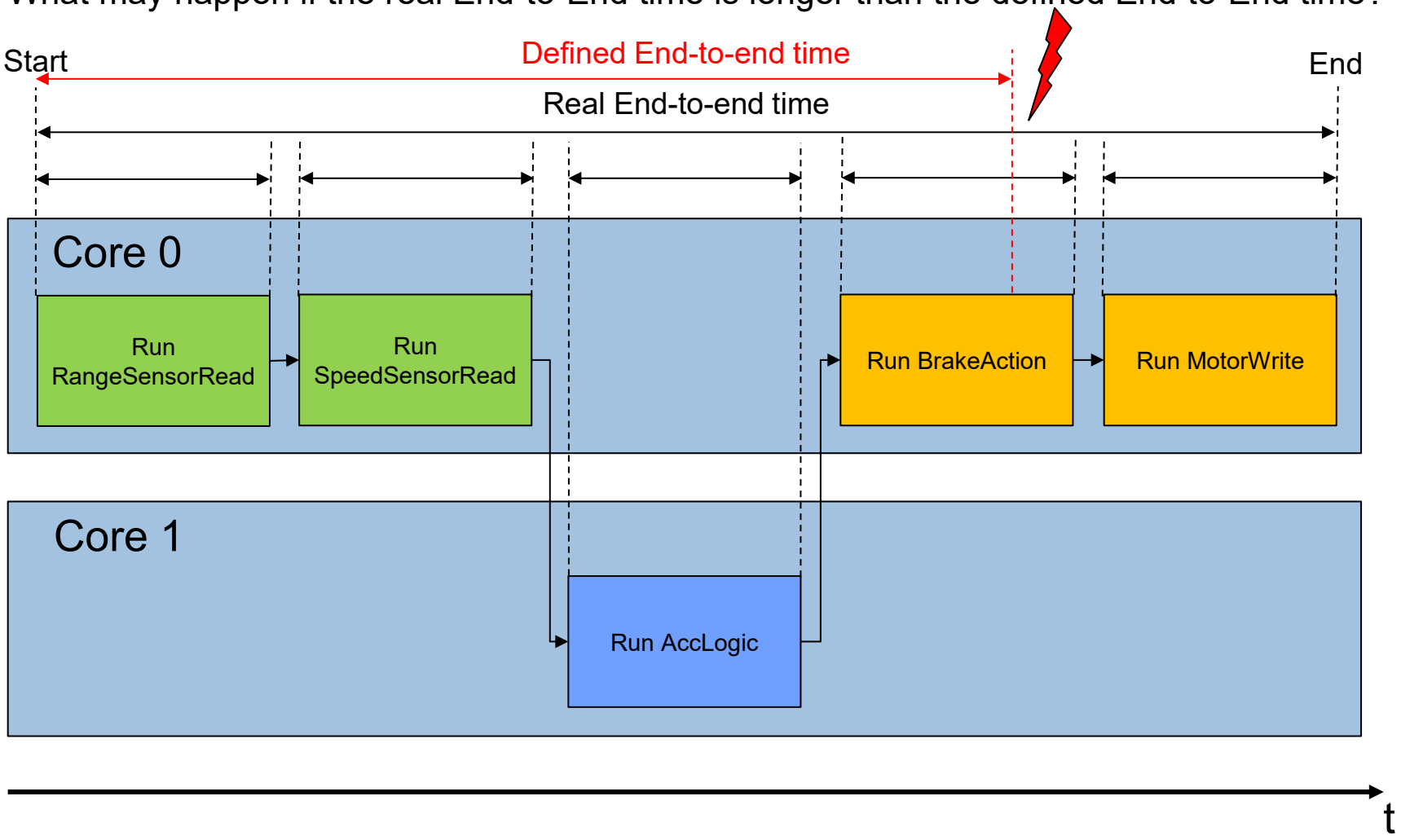
Timing issues and Control-Flows

Timing constraints apply to control flows as well as runnables



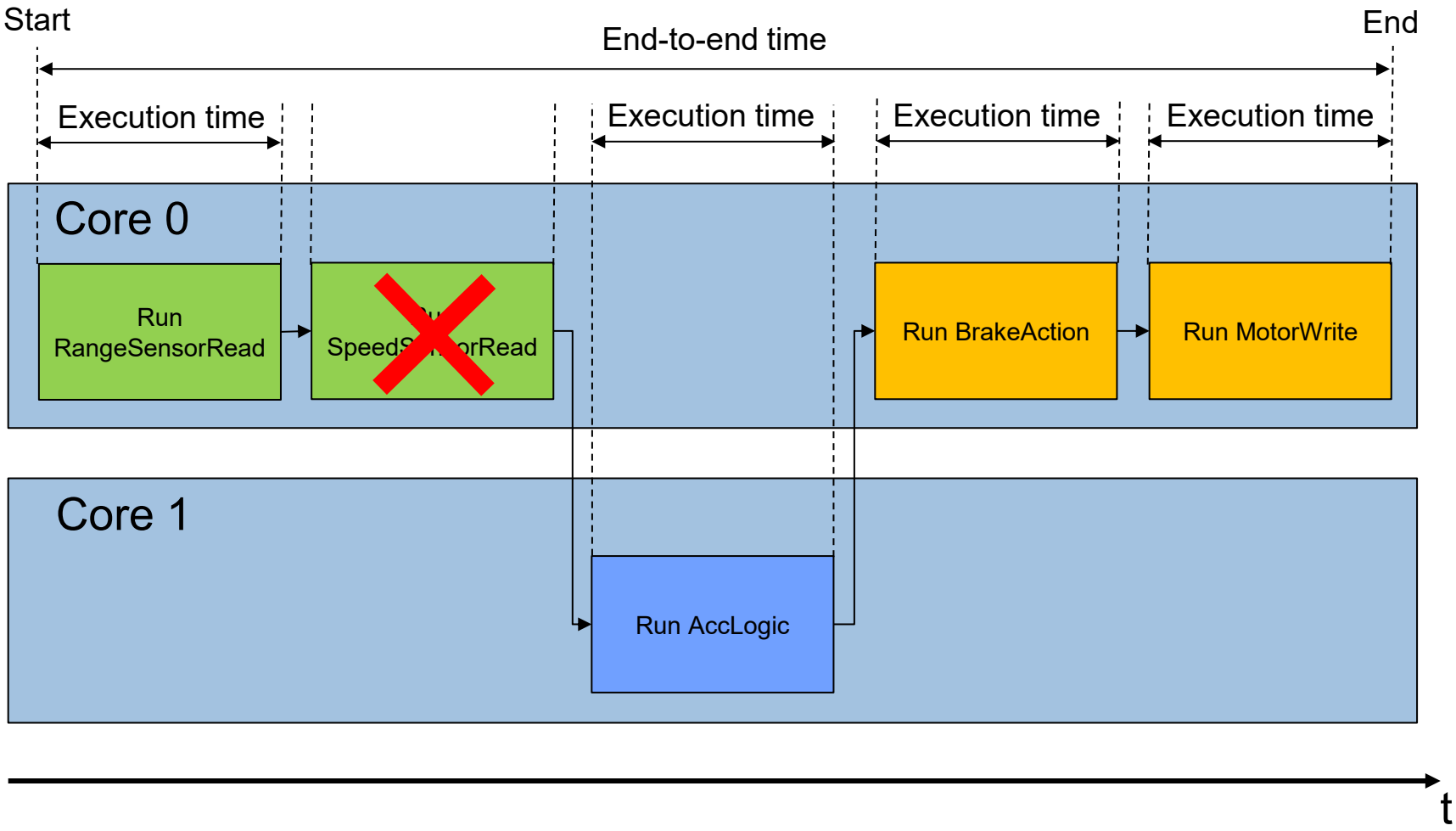
Timing issues and Control-Flows

What may happen if the real End-to-End time is longer than the defined End-to-End time?



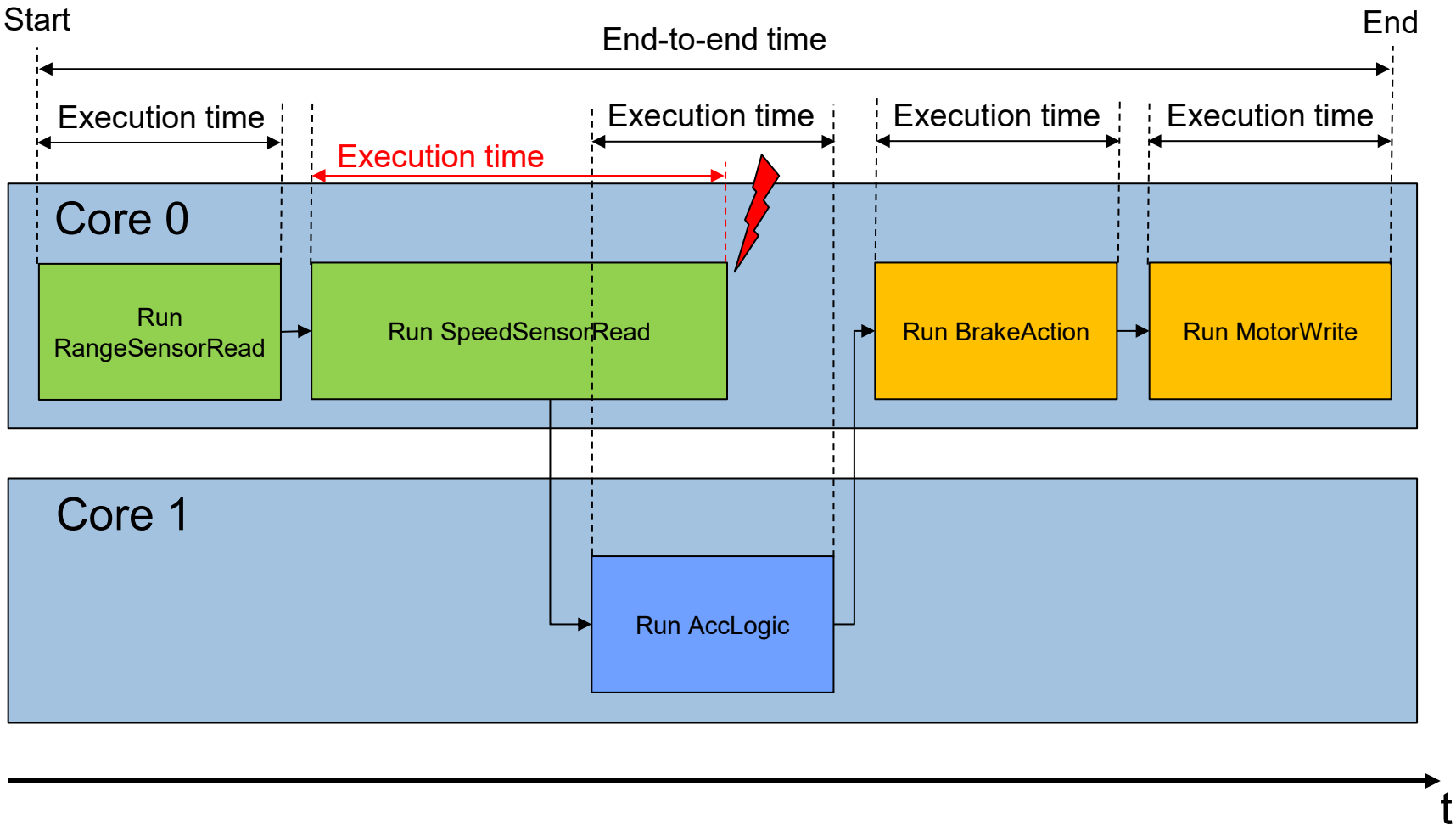
Timing issues and Control-Flows

What may happen if a runnable does not get executed due to implementation errors?



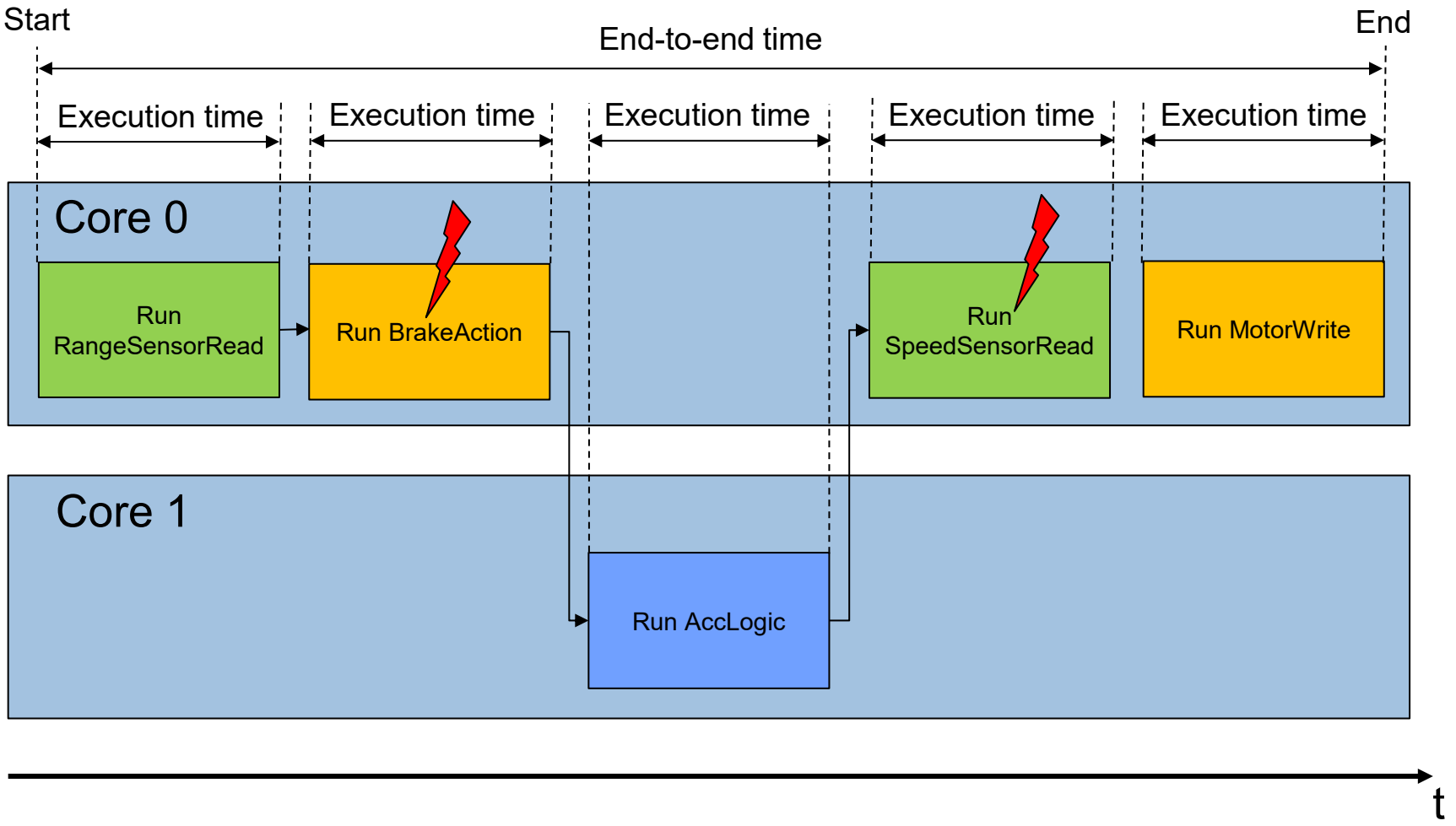
Timing issues and Control-Flows

What may happen if runnable needs far to long to execute?



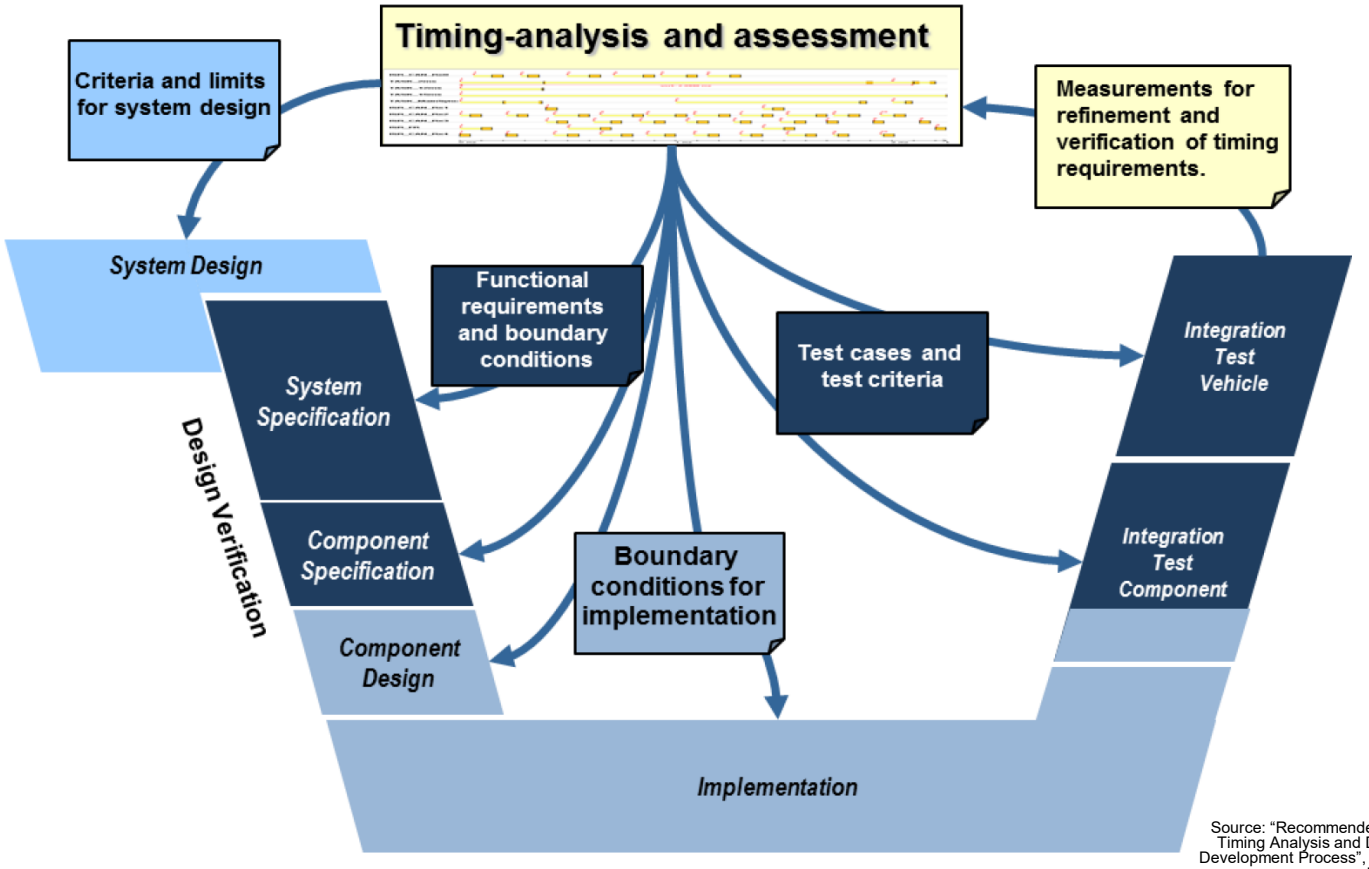
Timing issues and Control-Flows

What may happen if the execution order alternates?



Evaluation of timings and control flows

Evaluation of timing constraints is part of the overall development process

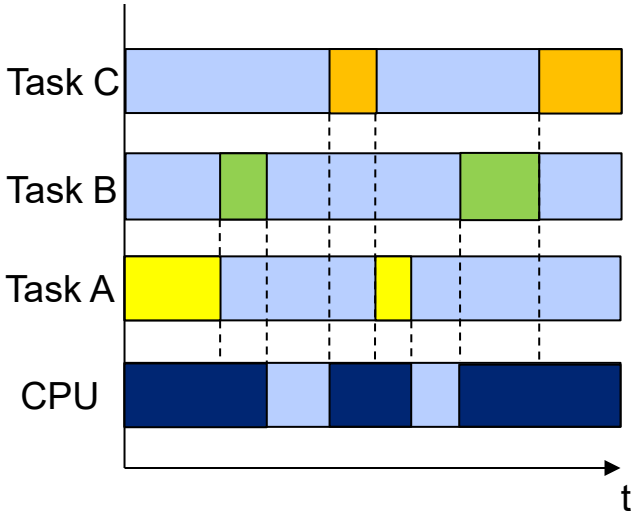


Source: "Recommended Methods and Practices for Timing Analysis and Design within the AUTOSAR Development Process", Document 645, AUTOSAR CP R19-11

Timing constraints and timing issues

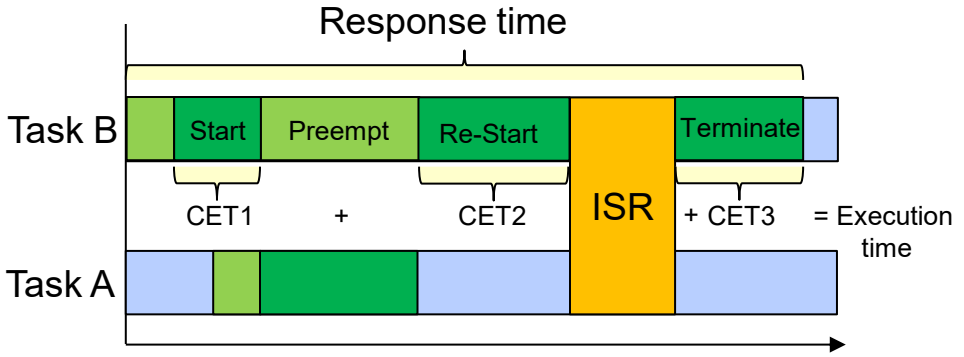
System Utilization

- Sum of work handled by a system within a given time frame
- Amount of time the system is occupied by one or more operations
- If occupation is higher than provided capacity -> overload



Timing behavior

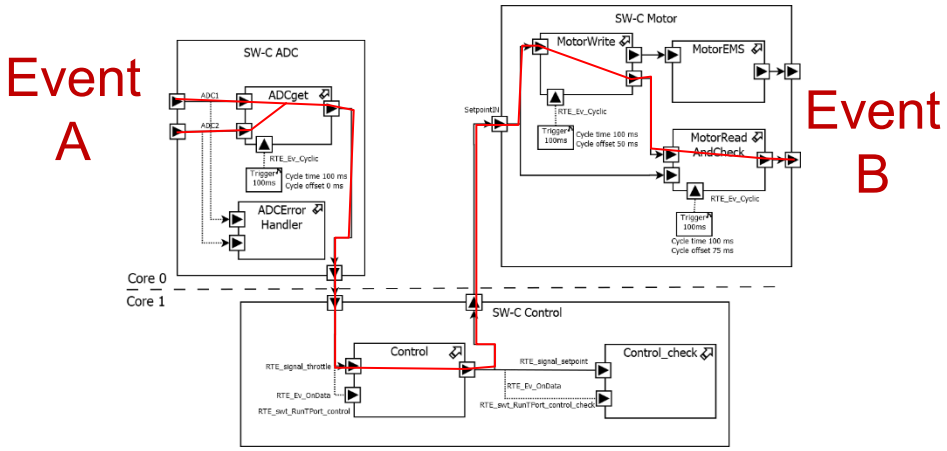
- Execution time
  - Time an operation needs to be executed
  - Depends on the capabilities of the hardware
- Response time
  - Time between start and end of an operation or process
  - Involves interruptions, delays and blockings



Timing constraints and timing issues

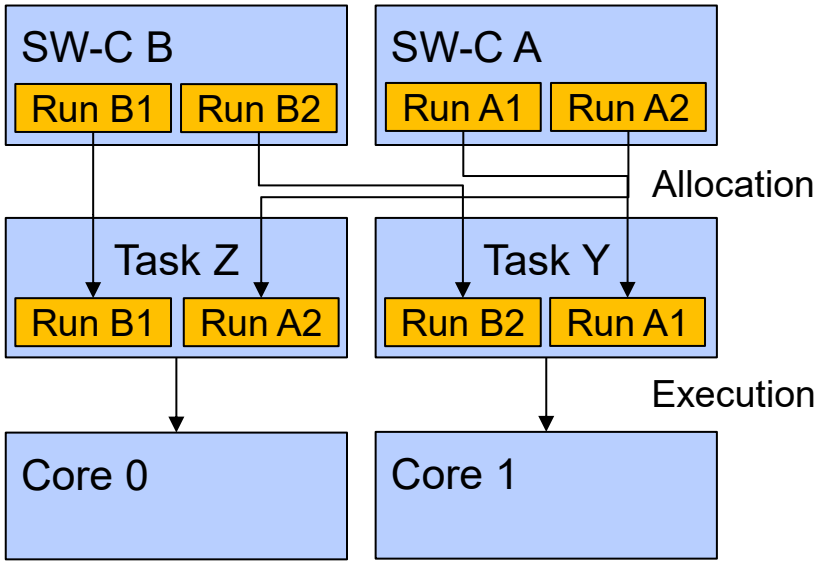
Sequences

- Control flows that describe the behavior of the system
- Control flows are incorrect if
  - Instructions are processed in a wrong sequence or
  - are not processed at all
- Event chains describe the sequences of actions constituting the control flow of the system

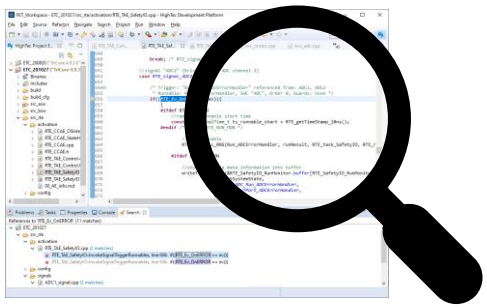


Execution context

- Assignment of operations to OS tasks and CPU cores
- Allocation is part of the design process and needs to be validated



Methods for timing analysis



Simulation

- Code simulation
- Scheduling simulation
- Network simulation
- Processor- and Hardware-In-The-Loop Simulation
- Discrete event simulation

Analytical

- Static code analysis
- Scheduling analysis
- Network analysis
- Compositional analysis

HIL-Simulation (Hardware-in-the-Loop) - MATLAB & Simulink https://de.mathworks.com/discovery/hardware-in-the-loop-hil.html

Measuring

- Time Measurement on ECU or Network level
- Tracing

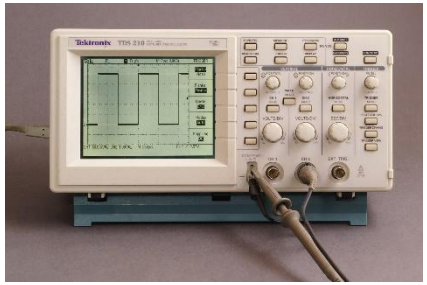
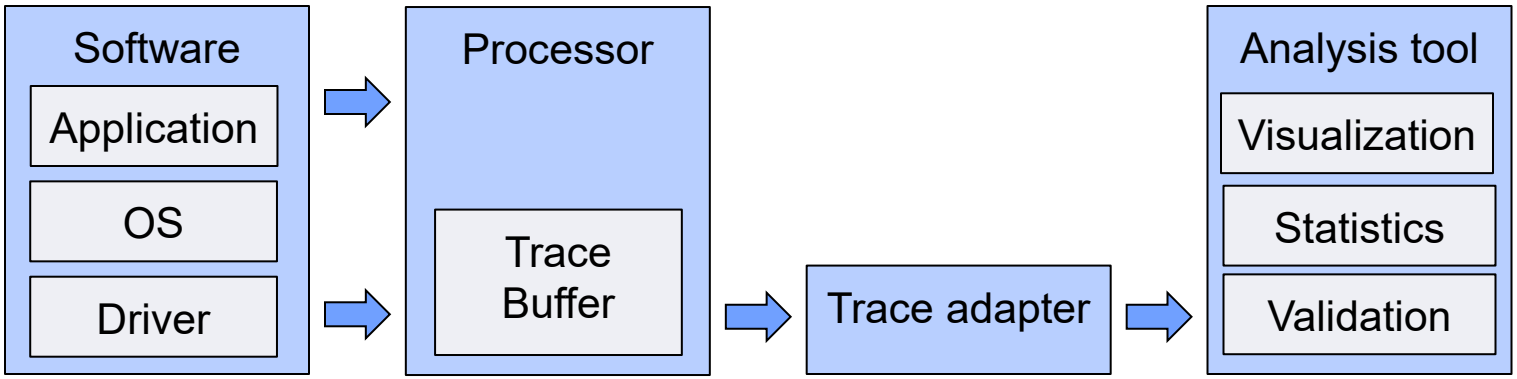


Photo: Rainer Knäpper, Free Art License (http://artlibre.org/licence/lal/en/), https://commons.wikimedia.org/wiki/File:Digitaloszilloskop_IMGP1971_WP.jpg

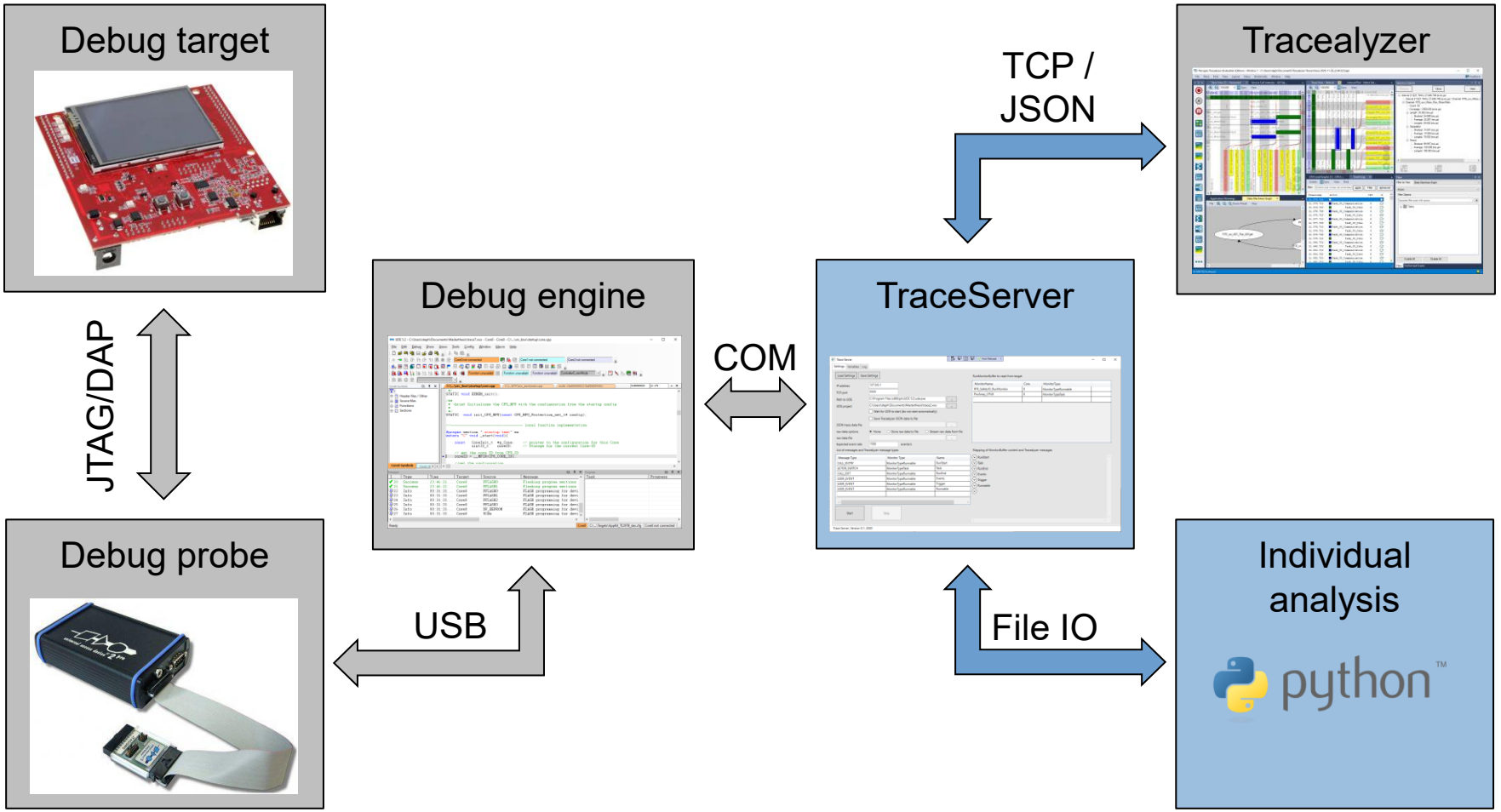
Tracing

- Observation of the real system
- Continuously buffering of measurement data streams
  - OS events
  - Calls
  - ➢ In combination with time stamps and event information
- Visualization of traced data streams for debug and analytical purposes
- Usable to reconstruct the observed scenario and extracting different kinds of timing information



Principle of generating and evaluating trace data

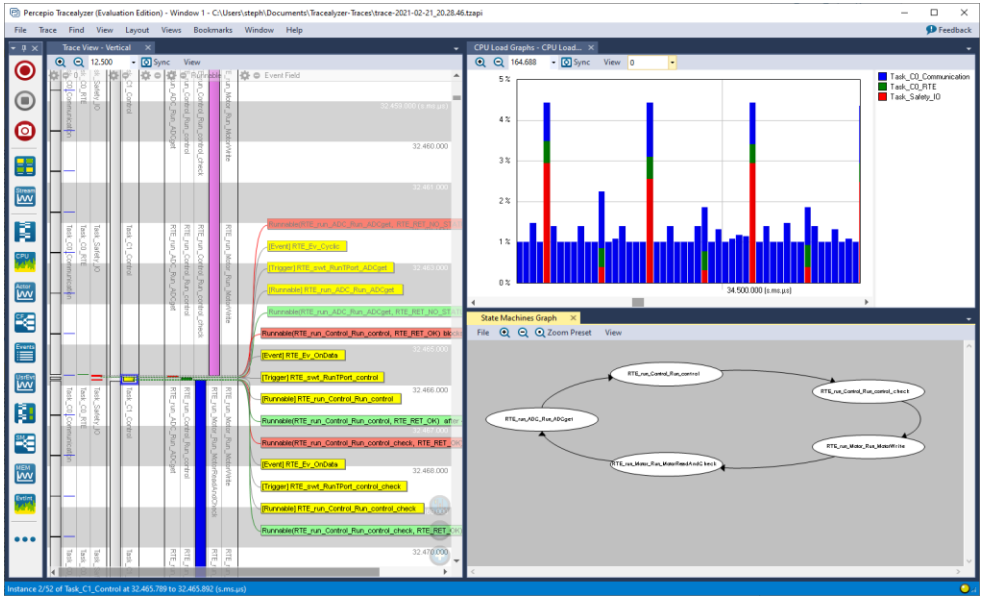
Setup



Percepio Tracealyzer

Tool for tracing and visualization of embedded and IoT software at runtime

- Different views provide insights into the real-time behavior of the traced system
- Capable to load trace data from the target using
- Additionally, a TraceAPI exists that allows to send trace information over software-based interface

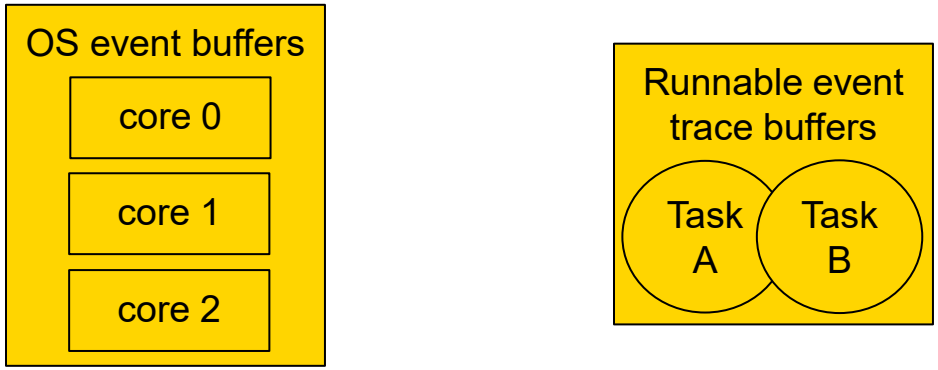


Trace data sources

For generating a holistic view of the timing behavior of the system different kinds of events need to be traced.

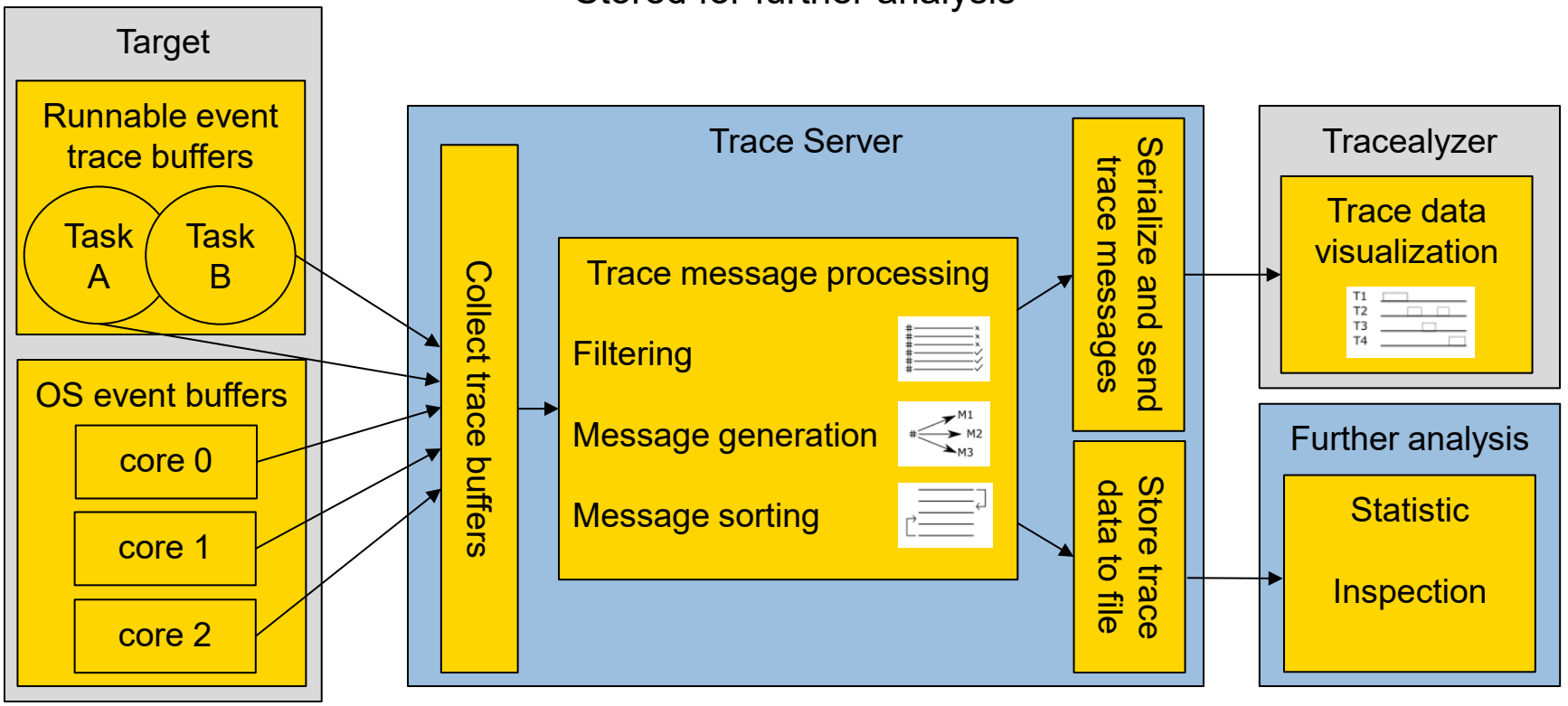
1. OS scheduling events of each CPU core to see task switches
2. Applicational events, like runnable calls and terminations, because these describe the behavior of the system to the outside world

On the trace target an instrumentation is needed to collect both types of events into two different types of trace buffers.



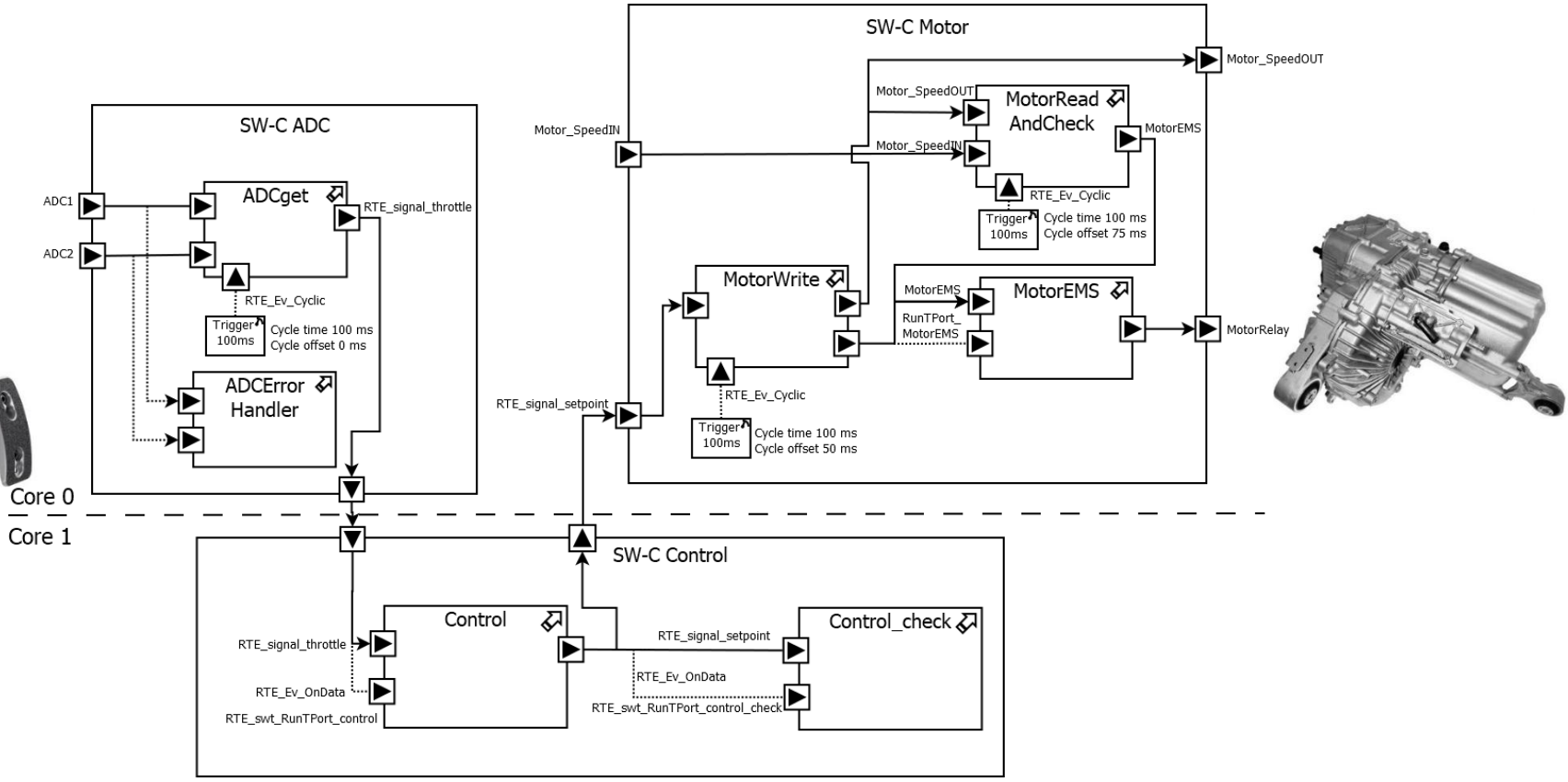
Tracing process

- Trace buffer entries are captured from multiple trace buffers
  - OS events from each CPU core
  - Runnable event data from each applicational RTE task
    - After processing data is
      - send to a trace visualization tool or
      - Stored for further analysis

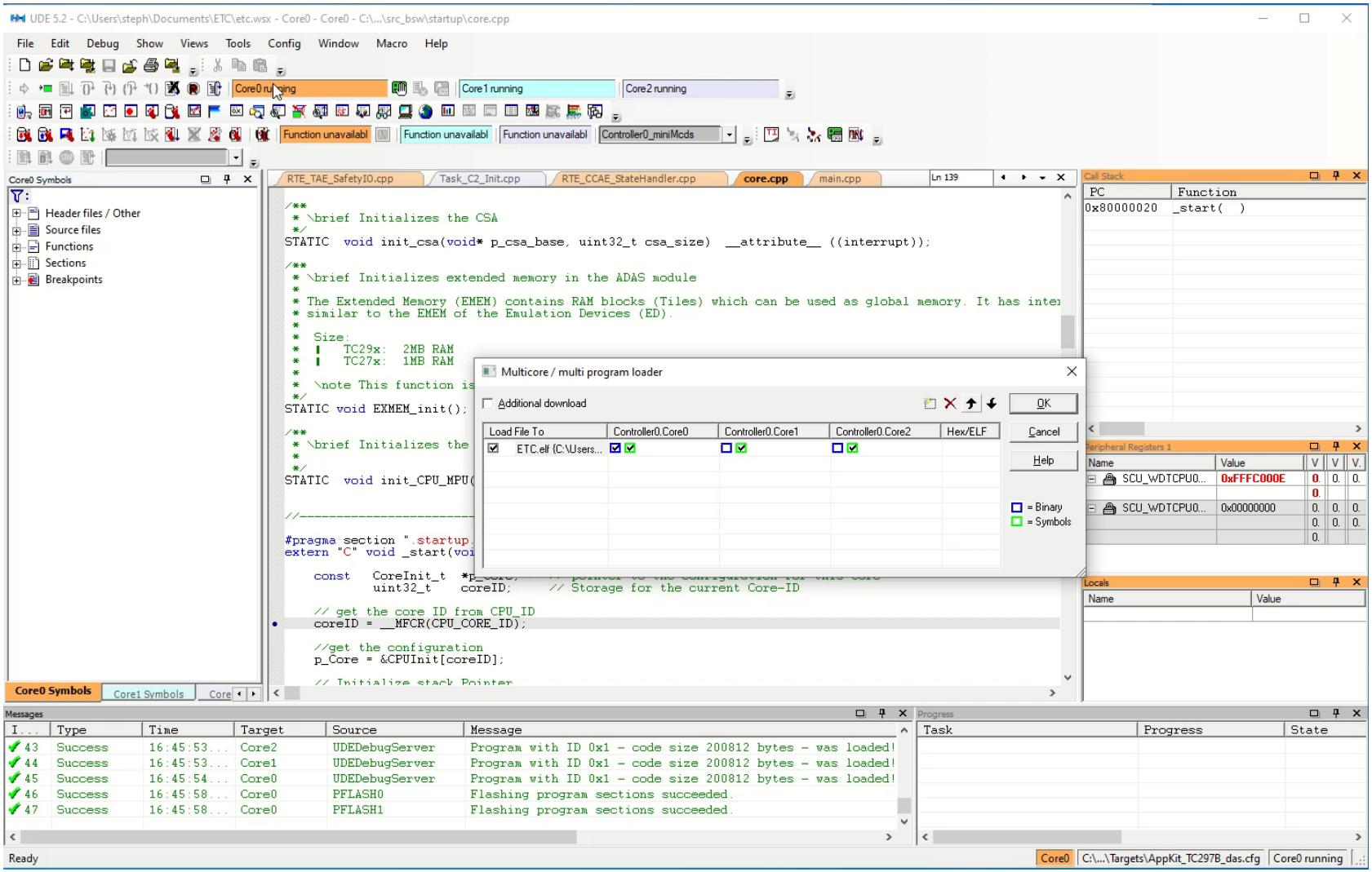


System under investigation

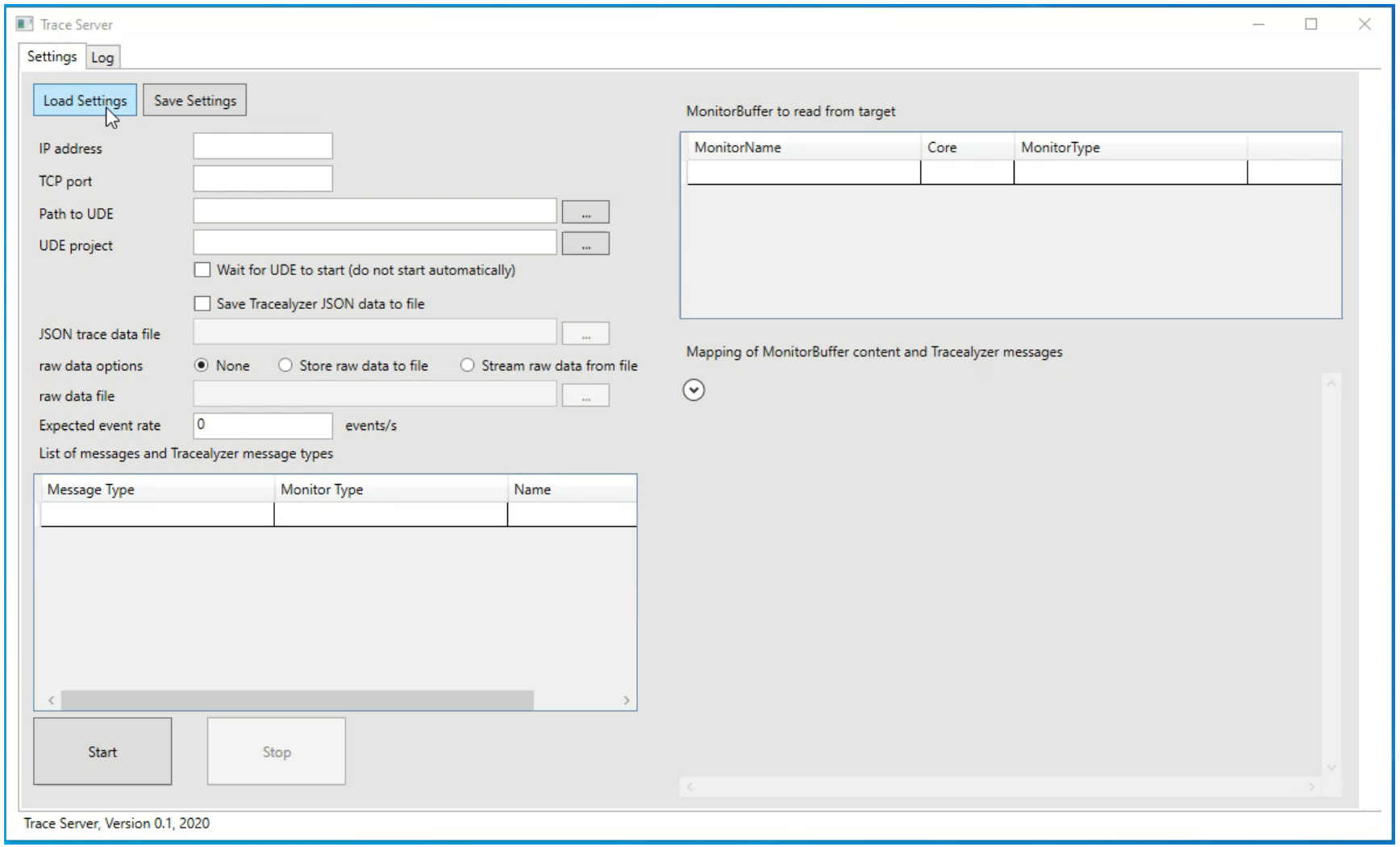
Electronic Throttle Control (ETC) project



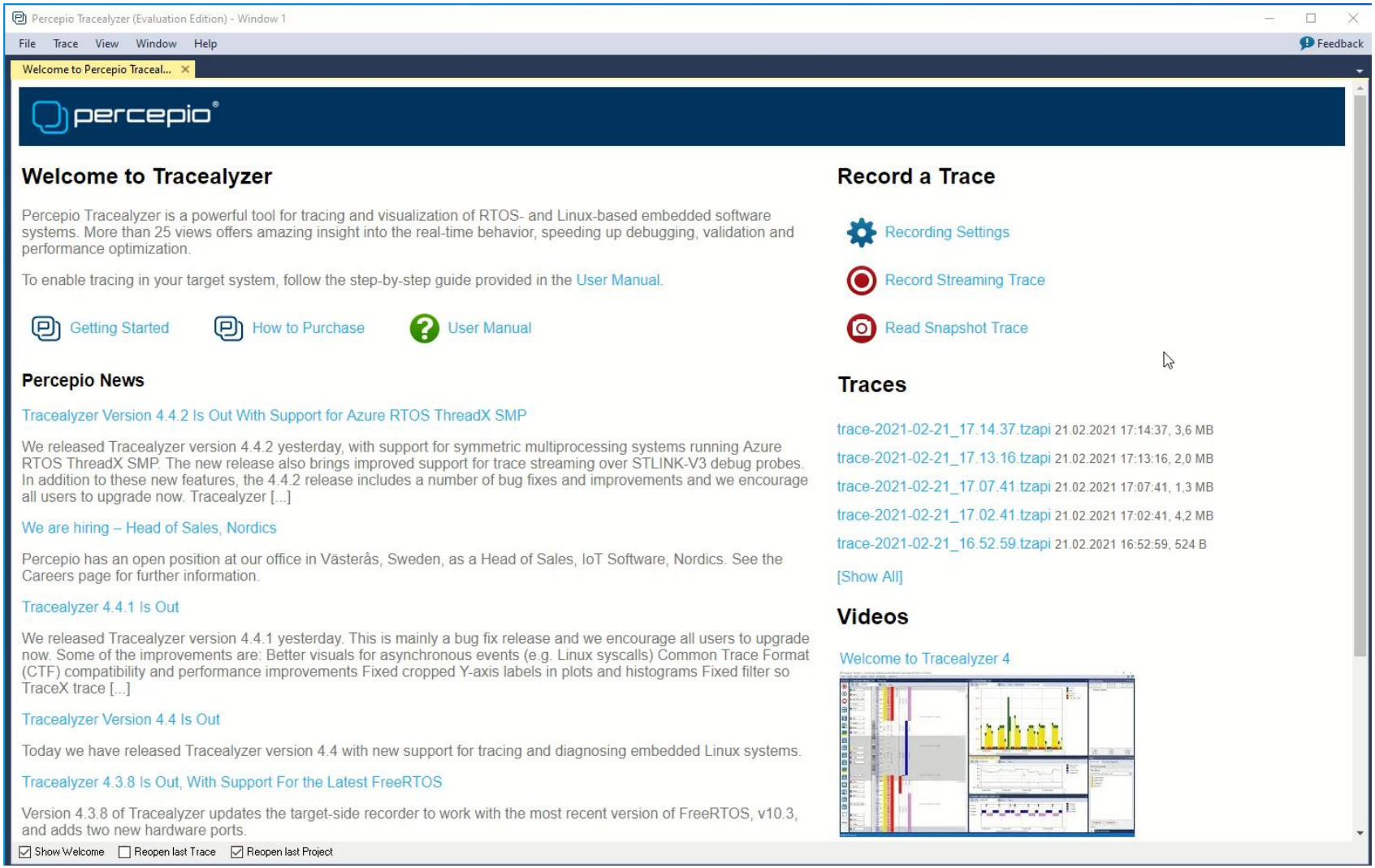
Demo


UDE 5.2 - C:\Users\steph\Documents\ETC\etc.wsx - Core0 - Core0 - C:\...\src_bsw\startup\core.cpp
File Edit Debug Show Views Tools Config Window Macro Help
Core0 running
Core1 running
Core2 running
Function unavailabl
Function unavailabl
Function unavailabl
Controller0_miniMcds
Core0 Symbols
Header files / Other
Source files
Functions
Sections
Breakpoints
RTE_TAE_SafetyIO.cpp
Task_C2_Init.cpp
RTE_CCAE_StateHandler.cpp
core.cpp
main.cpp
Ln 139
/**
* \brief Initializes the CSA
*/
STATIC void init_csa(void* p_csa_base, uint32_t csa_size) __attribute__ ((interrupt));
/**
* \brief Initializes extended memory in the ADAS module
*
* The Extended Memory (EMEM) contains RAM blocks (Tiles) which can be used as global memory. It has inte
* similar to the EMEM of the Emulation Devices (ED).
*
* Size:
* TC29x: 2MB RAM
* TC27x: 1MB RAM
*
* \note This function is
*/
STATIC void EXMEM_init();
/**
* \brief Initializes the
*
*/
STATIC void init_CPU_MPU(
//-----------------------
#pragma section ".startup.
extern "C" void _start(voi
const CoreInit_t *p_
uint32_t coreID; // Storage for the current Core-ID
// get the core ID from CPU_ID
coreID = __MFCR(CPU_CORE_ID);
//get the configuration
p_Core = &CPUInit[coreID];
// Initialize stack Pointer
Multicore / multi program loader
Additional download
OK
Cancel
Help
Load File To
Controller0.Core0
Controller0.Core1
Controller0.Core2
Hex/ELF
ETC.elf (C:\Users...
= Binary
= Symbols
Call Stack
PC
Function
0x80000020 _start( )
Peripheral Registers 1
Name
Value
SCU_WDTCPU0...
0xFFFC000E
SCU_WDTCPU0...
0x00000000
Locals
Name
Value
Core0 Symbols
Core1 Symbols
Messages
I...
Type
Time
Target
Source
Message
43 Success 16:45:53... Core2 UDEDebugServer Program with ID 0x1 - code size 200812 bytes - was loaded!
44 Success 16:45:53... Core1 UDEDebugServer Program with ID 0x1 - code size 200812 bytes - was loaded!
45 Success 16:45:54... Core0 UDEDebugServer Program with ID 0x1 - code size 200812 bytes - was loaded!
46 Success 16:45:58... Core0 PFLASH0 Flashing program sections succeeded.
47 Success 16:45:58... Core0 PFLASH1 Flashing program sections succeeded.
Progress
Task
Progress
State
Ready
Core0
C:\...\Targets\AppKit_TC297B_das.cfg
Core0 running

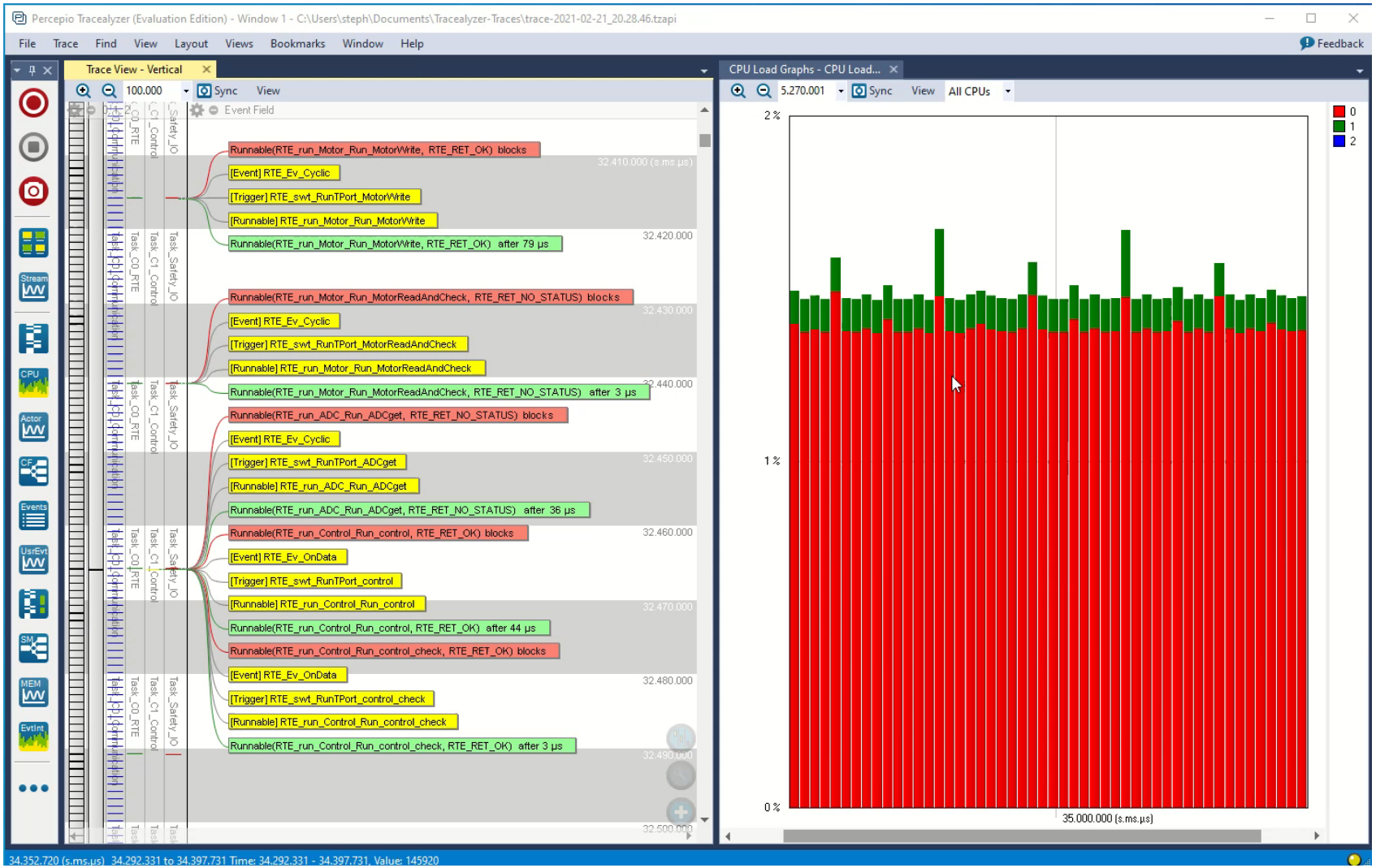
Demo


Trace Server
Settings
Log
Load Settings
Save Settings
IP address
TCP port
Path to UDE
UDE project
Wait for UDE to start (do not start automatically)
Save Tracealyzer JSON data to file
JSON trace data file
raw data options
None
Store raw data to file
Stream raw data from file
raw data file
Expected event rate
0
events/s
List of messages and Tracealyzer message types
Message Type
Monitor Type
Name
Start
Stop
MonitorBuffer to read from target
MonitorName
Core
MonitorType
Mapping of MonitorBuffer content and Tracealyzer messages
Trace Server, Version 0.1, 2020

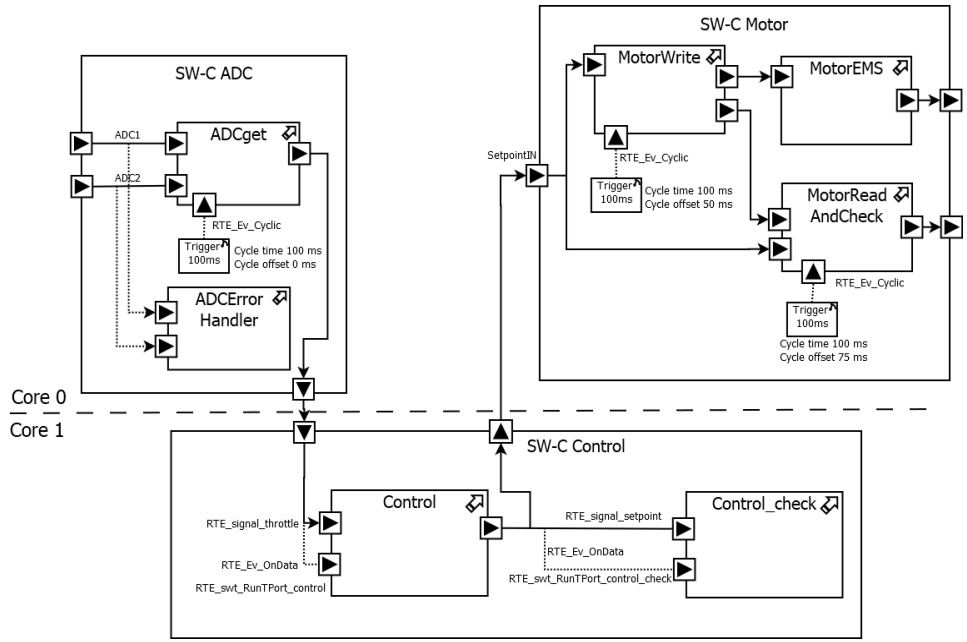
Demo

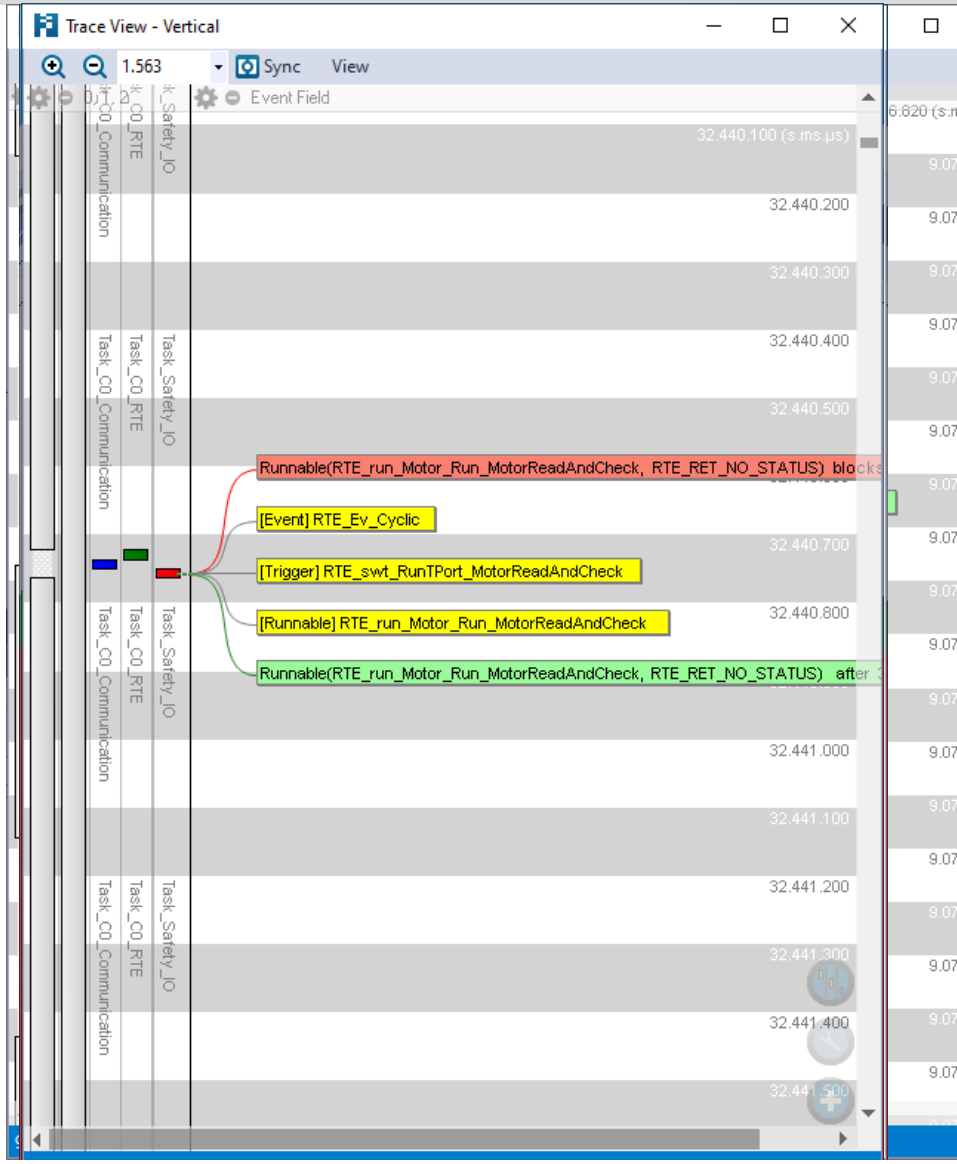

Percepio Tracealyzer (Evaluation Edition) - Window 1
File Trace View Window Help
Feedback
Welcome to Percepio Traceal...
percepio
Welcome to Tracealyzer
Percepio Tracealyzer is a powerful tool for tracing and visualization of RTOS- and Linux-based embedded software systems. More than 25 views offers amazing insight into the real-time behavior, speeding up debugging, validation and performance optimization.
To enable tracing in your target system, follow the step-by-step guide provided in the User Manual.
Getting Started
How to Purchase
User Manual
Percepio News
Tracealyzer Version 4.4.2 Is Out With Support for Azure RTOS ThreadX SMP
We released Tracealyzer version 4.4.2 yesterday, with support for symmetric multiprocessing systems running Azure RTOS ThreadX SMP. The new release also brings improved support for trace streaming over STLINK-V3 debug probes. In addition to these new features, the 4.4.2 release includes a number of bug fixes and improvements and we encourage all users to upgrade now. Tracealyzer [...]
We are hiring – Head of Sales, Nordics
Percepio has an open position at our office in Västerås, Sweden, as a Head of Sales, IoT Software, Nordics. See the Careers page for further information.
Tracealyzer 4.4.1 Is Out
We released Tracealyzer version 4.4.1 yesterday. This is mainly a bug fix release and we encourage all users to upgrade now. Some of the improvements are: Better visuals for asynchronous events (e.g. Linux syscalls) Common Trace Format (CTF) compatibility and performance improvements Fixed cropped Y-axis labels in plots and histograms Fixed filter so TraceX trace [...]
Tracealyzer Version 4.4 Is Out
Today we have released Tracealyzer version 4.4 with new support for tracing and diagnosing embedded Linux systems.
Tracealyzer 4.3.8 Is Out, With Support For the Latest FreeRTOS
Version 4.3.8 of Tracealyzer updates the target-side recorder to work with the most recent version of FreeRTOS, v10.3, and adds two new hardware ports.
Record a Trace
Recording Settings
Record Streaming Trace
Read Snapshot Trace
Traces
trace-2021-02-21_17.14.37.tzapi 21.02.2021 17:14:37, 3,6 MB
trace-2021-02-21_17.13.16.tzapi 21.02.2021 17:13:16, 2,0 MB
trace-2021-02-21_17.07.41.tzapi 21.02.2021 17:07:41, 1,3 MB
trace-2021-02-21_17.02.41.tzapi 21.02.2021 17:02:41, 4,2 MB
trace-2021-02-21_16.52.59.tzapi 21.02.2021 16:52:59, 524 B
[Show All]
Videos
Welcome to Tracealyzer 4
Show Welcome
Reopen last Trace
Reopen last Project

Demo


Percepio Tracealyzer (Evaluation Edition) - Window 1 - C:\Users\steph\Documents\Tracealyzer-Traces\trace-2021-02-21_20.28.46.tzapi
File Trace Find View Layout Views Bookmarks Window Help
Feedback
Trace View - Vertical
100.000
Sync View
Event Field
Runnable(RTE_run_Motor_Run_MotorWrite, RTE_RET_OK) blocks
[Event] RTE_Ev_Cyclic
[Trigger] RTE_swt_RunTPort_MotorWrite
[Runnable] RTE_run_Motor_Run_MotorWrite
Runnable(RTE_run_Motor_Run_MotorWrite, RTE_RET_OK) after 79 µs
Runnable(RTE_run_Motor_Run_MotorReadAndCheck, RTE_RET_NO_STATUS) blocks
[Event] RTE_Ev_Cyclic
[Trigger] RTE_swt_RunTPort_MotorReadAndCheck
[Runnable] RTE_run_Motor_Run_MotorReadAndCheck
Runnable(RTE_run_Motor_Run_MotorReadAndCheck, RTE_RET_NO_STATUS) after 3 µs
Runnable(RTE_run_ADC_Run_ADCget, RTE_RET_NO_STATUS) blocks
[Event] RTE_Ev_Cyclic
[Trigger] RTE_swt_RunTPort_ADCget
[Runnable] RTE_run_ADC_Run_ADCget
Runnable(RTE_run_ADC_Run_ADCget, RTE_RET_NO_STATUS) after 36 µs
Runnable(RTE_run_Control_Run_control, RTE_RET_OK) blocks
[Event] RTE_Ev_OnData
[Trigger] RTE_swt_RunTPort_control
[Runnable] RTE_run_Control_Run_control
Runnable(RTE_run_Control_Run_control, RTE_RET_OK) after 44 µs
Runnable(RTE_run_Control_Run_control_check, RTE_RET_OK) blocks
[Event] RTE_Ev_OnData
[Trigger] RTE_swt_RunTPort_control_check
[Runnable] RTE_run_Control_Run_control_check
Runnable(RTE_run_Control_Run_control_check, RTE_RET_OK) after 3 µs
32.410.000 (s.ms.µs)
32.420.000
32.430.000
32.440.000
32.450.000
32.460.000
32.470.000
32.480.000
32.490.000
32.500.000
CPU Load Graphs - CPU Load...
5.270.001
Sync View All CPUs
2 %
1 %
0 %
0
1
2
35.000.000 (s.ms.µs)
34.352.720 (s.ms.µs) 34.292.331 to 34.397.731 Time: 34.292.331 - 34.397.731, Value: 145920

Investigation of trace data

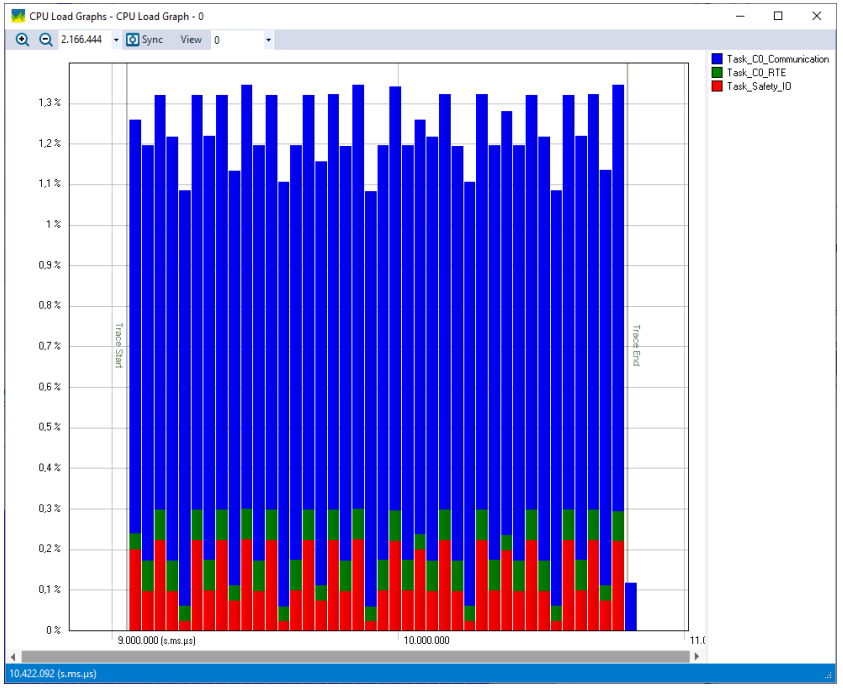




Investigation of trace data

Utilization

- Shows the occupation of the CPU cores
- Zoom can be used to adjust the time frame the utilization is measured



Statistics

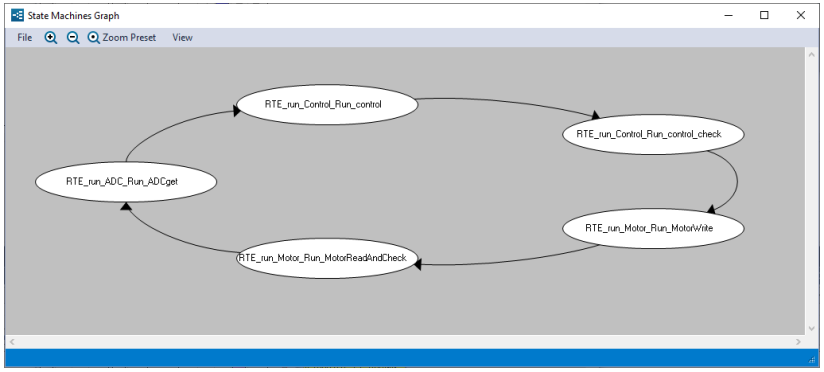
- All execution times of actors to provide an overview of worst- and best-case execution and response times

Actor	Count	CPU Usage	Execution Time			Response Time		
			Min	Avg	Max	Min	Avg	Max
Task_C0_Idle	1	98.764	1.717.496	1.717.496	1.717.496	1.738.990	1.738.990	1.738.990
Task_C1_Idle	1	98.804	1.698.354	1.698.354	1.698.354	1.700.000	1.700.000	1.700.000
Task_C0_Communication	1738	1.030	10	10	12	10	10	12
Task_C0_RTE	69	0.065	15	16	17	15	16	17
Task_C1_Control	18	0.100	97	97	97	97	97	97
Task_Safety_IO	87	0.141	10	28	50	10	28	50

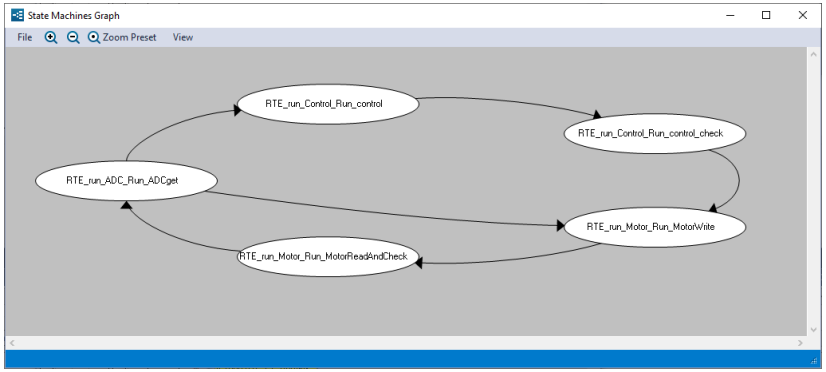
Investigation of trace data

State machine graph

- Life trace data can be used to calculate intervals and brings them into sequences
- State machine graphs show the runnable execution order



- Unexpected transitions indicate that runnables are not called properly
- This information is used for further investigation

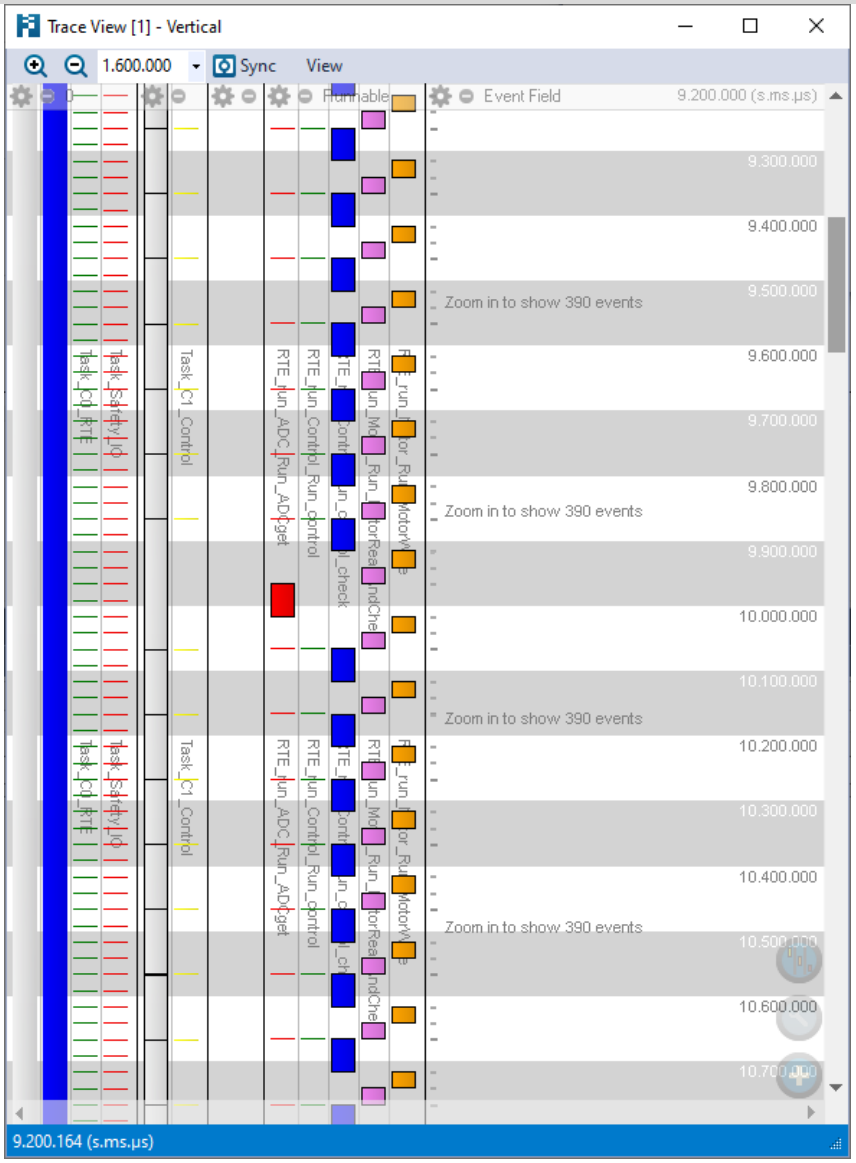


- After Run_ADCgets sometimes Run_MotorWrite has been called directly but Run_control needs to be called first

Debug support

Further investigation of trace view shows position where unexpected transition occurred

- After runnable Run_ADCget the control task does not become active
- Obviously OnData event does not arrive the task Control



Conclusion and Outlook

- Prototype solution shows that using existing tools can help to trace the timely behavior of multicore systems
  - By fusing OS and application events and
  - Flexibly map them to trace message objects of a visual trace diagnostics tool
- Tracing both types of events is extremely helpful to analyze complex timing behavior of safety application on multicore systems
- TraceServer and Tracealyzer do not know the application and the model behind
  - No automatic validation of control flows
- Outlook:
  - Implementing statistical evaluation of End-to-End times
  - Combining trace data with RTE model

Contact

Stephan Radke
Thomas Barth, M.Sc.
Prof. Dr.-Ing. Peter Fromm

Hochschule Darmstadt – University of Applied Sciences
FB Elektrotechnik und Informationstechnik
Birkenweg 8
64295 Darmstadt

Email: Stephan.Radke@stud.h-da.de, Thomas.Barth@h-da.de, Peter.Fromm@h-da.de
Web: www.eit.h-da.de

Backup

Principle of generating and evaluating trace data

Setup: description of components

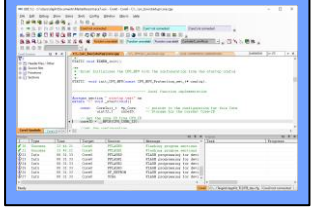
Debug target

TriCore AURIX TC297
Application board

Debug probe

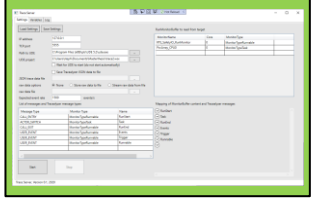
pls UAD2
Universal Access Device
Target access via JTAG

Debug engine



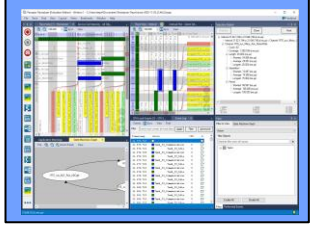
pls UDE
Universal Debug Engine
Microcontroller Debugger for AURIX and many others

Trace Server



Trace Server
Self-developed tool for accessing the Debugger and capture and converting trace information form the target

Tracealyzer



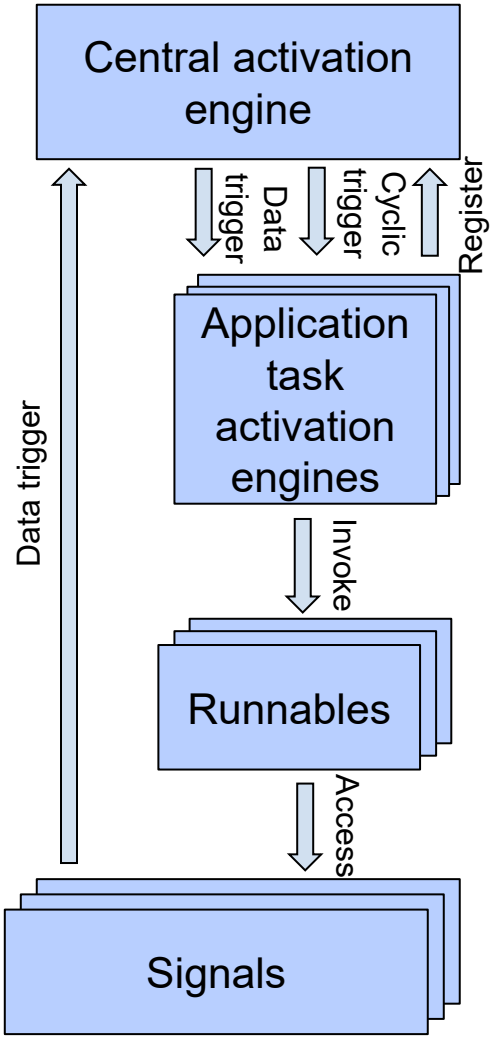
Percepio Tracealyzer
Visual trace diagnostics solution
Provides a TraceAPI interface for receiving external trace data

System under investigation

Modelled and generated runtime environment (RTE)

- Comparable to AUTOSAR CP architecture
- Activation engine
  - Controls the runtime behavior of the system
  - Invokes runnables according to
    - cyclic triggers (periodically)
    - data triggers (events)
- Signal layer
  - Realizes the data flow between Runnables
  - Stores data and handles accesses

- Runs on PXROS-HR real time operating system



Principle of generating and evaluating trace data

Features

Trace data buffer entry types can be mapped individually to Tracealyzer message object types

- Actor switch
- Service call entry / exit
- User event