# AUTOSAR CP AUTOSAR AP POSIX

**Functional** 

Software

System

Architecture

Implementation,

Configuration

(E)

Enviro

Run-Time

Level

cheduling

PRIORITY

Task\_B\_5n

Task\_B\_10ms

PRIORIT

Task

ABR.

IPT

CET

DT

Task

cutio

Architecture

Functionality A

CP SW-C 1

AUTOSAR CP ECU1

Communication Bus, e. g. CAN

Each Cycle ends with a call to WaitEvent () and starts when

RTE handles reception of data

RTE handles transmission of data

PER

CET = CET1 + CET2 + CET3 NST = NST1 + NST2

**EXPLANATION** 

initial pending time

core execution time

gross execution time

response time

delta time

DT

TIME

OSEK/AUTOSAR CP AND AUTOSAR AP DETERMINISTIC CLIENT

ABR.

PER

DL

"regular" WaitEvent () call (no need for any

"scheduling" WaitEvent() in this approach)

his code replaces the code above avoiding different kinds

of WaitEvent (). RTE communication is not shown here

(and in fact it is often more efficient with this approach).

WaitEvent () returns. So this function is used for scheduling

Task B. User expectation: one CET reflects one cycle!

RTE schedules all Runnables with 5ms period

Virtual Functional Bus (VFB)

AUTOSAR

XML (ARXML)

description files

Task starts here (transition Ready → Running)

RTE schedules all Runnables with 10ms

"regular" WaitEvent() call

**Functionality B** 

SW-Cs are implemented as sets of Runnables

CP SW-C 3

CP SW-C 3

OS

Task as generated by

many RTE generators

commended set-up

AUTOSAR CP ECU2

CP Application

(void)WaitEvent( Rte\_Ev\_Cyclic2\_Task\_B\_0\_10ms | Rte\_Ev\_Cyclic2\_Task\_B\_0\_5ms );

(void)ClearEvent(ev & ( Rte\_Ev\_Cyclic2 Task B 0 10ms | Rte\_Ev\_Cyclic2 Task B 0 5ms ))

(void)Com\_ReceiveSignal(TempSens2\_Rx, (&TempSens2\_Tx\_local));
if ((ev & Rte\_Ev\_Cyclic2\_Task\_B\_0\_10ms) != (EventMaskType)0)

if ((ev & Rte\_Ev\_Cyclic2\_Task\_B\_0\_5ms) != (EventMaskType)0)

CanTp\_MainFunction(); // Runnable
CanXcp\_MainFunction(); // Runnable
(void)WaitEvent( Can Ev\_TriggerSM\_Task\_B );
(void)GetEvent(Task\_B, &ev);
(void)ClearEvent(ev & ( Can\_Ev\_TriggerSM\_Task\_B ));
my5ms\_worker\_runnable(); // Runnable

(void)Com\_SendSignal(Torque\_Tx, (&Torque\_Tx\_local));

EventMaskType ev;
CanTp MainFunction();
// the following WaitEvent call is a "regular" WaitEvent
.(void)GatEvent(Can\_Ev\_TriggerSM\_Task\_B);
(void)GetEvent(Task\_B, &ev);
(void)ClearEvent(ev & (Can\_Ev\_TriggerSM\_Task\_B));
CanXcp\_MainFunction();
TerminateTask().

a single Ready state only

(void)GetEvent(Task\_B, &ev);

TASK(Task\_B\_10ms) // BCC1

ASK(Task\_B\_5ms) // ECC

CP / AP

Here SOME/IP is used for

cross-platform communi-

cation. Other options are

using a gateway or a Signa

to Service translation

implemented as an AP

Application in the AP ECI

TRANSLATION

Mapping of Functionality to Software Components (SW-Cs

CP SW-C 2

CP SW-C 1

• Runnable my5ms\_worker\_runnable

OS

CORE 1

• Runnable my10ms\_worker\_runnable CP SW-C 2

Deployment: mapping of Runnables to tasks and tasks to cores

TASK(Task\_B)

## 01\_INTRODUCTION

For many years classical AUTOSAR operating systems, based on the OSEK/VDX standard, suited the automotive world quite well.

Initiated in 2016 as an extension to the AUTOSAR CP ("Classic Platform"), the AUTOSAR AP ("Adaptive Platform") targets high-performance ECUs such as autonomous driving controllers [3](2.1). AUTOSAR AP comes with major impacts on the way software is developed – including all timing aspects.



This poster sheds some light on how timing is addressed in the existing AUTOSAR CP world, the new AUTOSAR AP world and beyond.

## 02 AUTOSAR CP

### 2.1 Architecture

When AUTOSAR was introduced in 2003, it defined the methodology for how to develop a system top-down. See the left side of the central diagram which deals with AUTOSAR CP.

## 2.2 Scheduling and Execution Management

In a nutshell: OSEK defines tasks as containers for code to be executed. Tasks have fixed priorities – leaving aside temporary priority modification through the priority ceiling protocol – and their attributes are defined at compile-time. Multicore Partitioned Scheduling is also statically configured off-board.

Typically, the RTE generator creates tasks like the nonterminating Extended Conformance Class (ECC) TASK\_B shown on the right. However, in many cases it would be better to create separate terminating Basic Conformance Class (BCC) tasks to increase efficiency and reduce complexity [8].

### 2.3 Timing Parameters

AUTOSAR CP release 4.1.3 introduced the definition of timing parameters. The figure on the right shows all timing parameters in red color.

## 03 GLOSSARY

## **AA** Adaptive Application

Abstract System Description Description of features on an abstract level without considering the actual mapping to hardware, software and physical communication interfaces **AMP** Asymmetric Multiprocessing implies that different cores are assigned different roles

AP Machine A (virtualized) ECU onto which software can be deployed

**ARXML** AUTOSAR XML, including system, software components and communication configuration **Common Software Architecture** overview of all software entities and their communication relation independent of the mapping to AUTOSAR CP and AUTOSAR AP

**ECU** Electronic Control Unit, an embedded system that controls motor vehicle subsystems MMU Memory Management Unit, a hardware component necessary to implement virtual memory

Multicore Partitioned Scheduling Scheduling algorithm in which tasks are statically partitioned between cores. Often chosen for AMP

Multicore Global Scheduling Scheduling algorithms in which cores are resources dynamically assigned to tasks. Often chosen for SMP

Multiprocessing (OS) OS support for multiple concurrent

**OSEK/VDX** A Standard Automotive Embedded OS reused for the AUTOSAR OS layer **PID** Process Identifier - a unique identifier for the process **POSIX** Portable Operating System Interface (Unix-like)

**Process** An address space realized with virtual memory, with

**RTE** AUTOSAR Run-Time Environment **SMP** In Symmetric Multiprocessing, all cores are considered

one or more threads executing within that address space

**SOME/IP** Scalable service-Oriented MiddlewarE over IP, an automotive middleware solution for control messages **SW-C** Software Component (CP)

**Task** A run-time software component as viewed by the OS

Task Scheduler **Thread** A single flow of control within a process. Process memory is accessible to all threads in the same process

**VFB** Virtual Function Bus (CP) Virtual Memory An abstraction layer which allows multiprocessing through multiple different views of memory

## 06 REFERENCES



Specification of Execution Management AUTOSAR AP Release 18-10 https://www.autosar.org/standards/adaptive-platform/

[2] AUTOSAR, Specification of Communication Management AUTOSAR AP Release 18-10

https://www.autosar.org/standards/adaptive-platform/

Parameters GET

Timing

https://www.autosar.org/standards/adaptive-platform/ Specification of Adaptive Platform Design AUTOSAR AP Release 18-10 [4] AUTOSAR, Methodology for Adaptive Platform AUTOSAR AP Release 18-10 https://www.autosar.org/standards/adaptive-platform/

Specification of Manifest AUTOSAR AP Release 18-10 https://www.autosar.org/standards/adaptive-platform.

EXPLANATION

deadline ("max. RT")

period

slack time

net slack time

Guidelines for the use of the C++14 language in critical and safety-related systems; AUTOSAR AP Release 18-10 https://www.autosar.org/standards/adaptive-platform/

Basic Conformance Class (BCC)

**Extended Conformance Class (ECC)** 

[7] IEEE, POSIX Certification http://get.posixcertified.ieee.org

preemption time (AUTOSAR CP only)

[8] Peter Gliwa, GLIWA, A systematic approach for timing requirements EMCC (Embedded Multi-Core Conference) 2018 in Munich https://gliwa.com/index.php?page=downloads&lang=eng/

ABR.

WAITT

RUNT

SL

[9] The Open GROUP, POSIX Standard

https://publications.opengroup.org/standards/unix [10] WIKIPEDIA, Hypervisor https://en.wikipedia.org/wiki/Hypervisor [11] WIKIPEDIA, POSIX

https://en.wikipedia.org/wiki/POSIX [12] Red Hat, Topics, What are Linux containers? https://www.redhat.com/en/topics/containers [13] Kay A. Robbins, Steven Robbins, UNIX Systems Programming: Communication

CPU-utilization (also referred to as CPU-load): the

(the "observation frame") in relation to  $t_{\Omega}$  times the

Such CETs can relate to a specific code-fragment, a

runnable, a task, a thread, a process, a core or the

sum U of all CETs within a defined time-frame  $t_{\rm O}$ 

number of relevant cores  $(n_{C})$ .

\_\_\_

complete system

CET = RUNT1 + RUNT2 + RUNT3

RUNT4 + RUNT5 + RUNT6

TIME

Concurrency, and Threads, Prentice Hall, 2003 [14] SOA Manifesto Authors, SOA Manifesto http://www.soa-manifesto.org

[15] AUTOSAR. Recommended Methods and Practices for Timing Analysis and Design within the AUTOSAR Development Process https://www.autosar.org/standards/classic-platform/

 $\sum_{n=1}^{N} CET(n)$ 

## 04 AUTOSAR AP

### 4.1 Architecture

**Software** 

System

Architecture

Implementation,

Configuration

Client

ecution

Client

Deterministi

Let's take a look at how timing is or can be

Service

Instance

Manifest

int main(int argc, char \*argv[])

For deterministic execution requirements.

an optional Deterministic Client API is available.

int retval;
// initialize App data here

return retval; // terminate with success

/ call App code here (which may or may not return), e.g.: / retval = AppCode():

\* if the process uses the DeterministicClient, the App code alled from the main function above could look like this: \*/

ActivationReturnType dccType; // Deterministic Clien

Wait (also "Block"

POSIX [9] is a set of standards for operating systems, which

allows applications to be easily moved from OS to OS without

Its scope is mostly the portable interface presented to the

out of scope. The POSIX standards include the C language, processes, threads, signals, timers, file system, I/O, IPC, real

time extensions and more. POSIX fundamentally requires virtual memory in order to support multiprocessing.

However, many OSes do not go through the process, for

practical reasons. A prominent example of such a POSIX-like

operating systems is GNU / Linux, although specific variants

application and leaves most of the internal OS implementation

POSIX certifications for operating systems are available [7]

RUNT: accumula

time spent in "Runni

Non-existing

05 POSIX

requiring changes.

are certified.

(\*) When calculating RUNT, the time

"stolen" (interrupts etc.) is deducted

Applications are deployed as Software Package

Machine

Manifest

SW Update &

Configuration

Management

Authentication, SW Package

Deployment, SW Package

pdate, Reconfiguration

Functionality C

Mapping of Functionality to Adaptive Applications (A

AP Applications are implemented as POSIX application

AUTOSAR AP ECU (3)

Sinit, configuration, shutdow

OS (POSIX)

**3** "

'call WaitForNextActivation()'

"return from
WaitForNextActivation()

ne Cyclo

The CET for an App Process

accumulates the run-time of

all its threads. As a conse-

quence, the CET might be

greater than the GET.

(Virtual) Machine / Container / Hardware

pplication 2

and data

Service

**Instance** 

Manifest

SOFTWARE PACKAGE

Execution

Manifest

Application 1

Process states (as seen from the Execution Manager's perspective)

Communication

DCC\_Suspended

NST = ST-GET of kServiceDiscovery in between TIME

RUNT2

**RUNT5 RUNT6** 

thread 2

The Application processes and threads are

scheduled by the underlying POSIX OS.

ACTIVATION signal

Management

Initializing Running Terminating

Adaptive Application Lifecycle

Idle Starting Running Terminating Terminated

Service Registration

**DCC** = Deterministic Client Cycle

CYCLES

All timing parameters

RUNT1

are related to kRun.

**App Process** 

— thread 3

Core view

POSIX OS SCHEDULING

EXPLANATION

accumulated time spent in "Waiting"

accumulated time spent in "Running"

scheduling latency

Core 0

plication

and data

Process /

Application 2

SOFTWARE PACKAGE

Application 2

Other services

e.g. State Management, Time

Management, Diagnostics, etc

Execution

Manifest

addressed in AUTOSAR CP and AUTOSAR AF

AUTOSAR AP differs from AUTOSAR CP in that it defines a SOA (Service-Oriented Architecture [14]) for the Adaptive Applications (AA).

The AUTOSAR Runtime for Adaptive Applications (ARA) constitutes the environment in which the applications can run and access a wide variety of high level services, plus low level interfaces to the OS [3](3.1). AAs are developed in C++ [6]. The OS of AUTOSAR AP is **POSIX** as opposed to OSEK/VDX of AUTOSAR CP.

The ARA runs on an AUTOSAR AP machine, which can

- a physical machine including an MMU in order to
- support virtual memory for multiprocessing [3](4.4) • a virtual machine – supported by any kind of
- hypervisor [10]
- a container a standardized encapsulated software environment [12]

## The most important components of the ARA

- n this context are:
- Execution Management [1]
- Communication Management [2]
- POSIX Operating System [9]

### 4.2 Organizing Execution

The execution of an AA is organized into multiple layers of abstraction on top of the OS scheduling level. Each layer can be modeled with a dedicated state machine as shown in the AP part of the picture on the left.

The Execution Manager and AA need to synchronize their view of the AA state. The Execution Manager creates the AA process ("Idle" to "Starting"). AA's execution state will start at "Initializing". After initialization, the AA reports its "Running" execution state.

In the case of self-termination, the AA reports its execution state as "Terminating". The AA will then clean up, commit to storage, and exit normally from the process. The Execution Manager will wait on the process exit status ("Terminating" to "Terminated"). If the execution management initiates the termination, it sends a POSIX signal to the AA, which will react by running the same procedure.

### 4.3 Deterministic Client

Beside other features, the Deterministic Client API [1](7.6.2) offers event-driven and time-driven cyclic execution. The picture on the left shows a code example for such a cycle, its state machine and the execution over time.

The timing parameters we propose (CET, DT, ... in red) relate to a specific cycle type, in the picture "kRun". For each cycle type a separate set of timing parameters is calculated.

## 4.4 OS Scheduling

AA tasks are processes and threads, scheduled to run on the available cores according to the POSIX scheduling policies, including "FIFO", "RR", and "OTHER" policies [9]. In contrast to the Multicore Partitioned Scheduling used in OSEK, POSIX OS implementations generally use Multicore Global Scheduling, dynamically assigning cores to threads at run-time. A useful general model for process and thread states is shown in the scheduling state diagram [13] (3.2, Fig 3.1) at the bottom right in the picture.

When a process is created ("non-existing" to "new"), usually by the "parent", the OS assigns it a Process Identifier (PID), an address space and the first thread of execution, also called the "main thread", for which AP standardizes the configuration [1](7.7.3.3) (7.6.3.1).

The new thread is admitted ("new" to "ready"), and can then be picked up for execution via a context-switch ("ready" to "running"). A thread can release the core voluntarily ("running" to "waiting" / blocked / sleeping), or involuntarily (preemption: "running" to "ready"). The OS wakes up and switches-in waiting threads and

preempted threads respectively at the appropriate time ("waiting" to "ready", "ready" to "running"). When a thread terminates ("running" to "done" / dead / terminated), the OS will keep the exit status information.

When the exit status has been collected, it will be freed and the thread will cease to exist ("done" to "non-existing").

A "done" process whose parent process does not collect its exit status is called a "Zombie" process, whose long lasting presence should be avoided, as it uses up valuable PIDs.

## 4.5 Timing Parameters

GLIWA introduced the timing parameters shown in the CP section of the picture on the left several years ago into AUTOSAR CP [15]. With this poster we propose a mapping of the timing parameters to the cycles of the deterministic client as outlined in the picture.



www.gliwa.com

GLIWA GmbH embedded | Pollinger Str. 1 | D-82362 Weilh