Timing in AUTOSAR CP, AUTOSAR AP and beyond

Peter Gliwa, EMCC 2019
Contents

- The big picture of Timing in AUTOSAR
- Timing top down in AUTOSAR CP
- Timing top down in AUTOSAR AP → including some suggestions
- Status of ARTI (AUTOSAR/ASAM Run-Time Interface)
- Status of TIMEX for AUTOSAR AP
AUTOSAR performance vs. dancing

- AUTOSAR CP, Single-core
  - Cf. one single guy doing break-dance
  - High performance core

- AUTOSAR CP, Multi-core
  - Cf. dancing chorus
  - Communication between (often very similar) cores

- AUTOSAR AP, Any-core
  - Cf. dancing crowd
  - Rather non-deterministic behavior
  - Difficult to control
The big picture
The big picture: general structure

1. shared

2. CP

3. AP

4. shared

Today’s tour

Layers (of abstraction)
Step 1: Functional Architecture

Layers (of abstraction)

1. shared

2. CP

3. AP

4. shared

Today’s tour
Functional architecture

Functional Architecture

Functionality A

Functionality B

Functionality C
Step 2: AUTOSAR CP top-down

1. shared

2. CP

3. AP

4. shared

Today's tour

Layers (of abstraction)
In AUTOSAR CP, functionality gets mapped onto “Software Components” (SW-C).
Implementation, System Configuration

**Software Architecture**

- **Functional Architecture**
  - Functionality A
  - Functionality B
  - Mapping of Functionality to Software Components (SW-Cs)

- **Virtual Functional Bus (VFB)**

- **Software Components (SW-Cs)**
  - CP SW-C 1
  - CP SW-C 2
  - CP SW-C 3
  - SW-Cs are implemented as sets of Runnables

**Implementation, System Configuration**

- AUTOSAR XML (ARXML) description files

- **Runnables**
  - my10ms_worker_runnable
  - my5ms_worker_runnable
  - (...)

- Timing Parameters
  - [Charts and Diagrams]
Deployment

Functional Architecture

- Functionality A
- Functionality B

Mapping of Functionality to Software Components (SW-Cs)

Software Architecture

- CP SW-C 1
- CP SW-C 2
- CP SW-C 3

Virtual Functional Bus (VFB)

Implementation, System Configuration

- AUTOSAR XML (ARXML) description files

Deployment

AUTOSAR CP ECU1
- CP Application
- RTE
- BSW
- OS

AUTOSAR CP ECU2
- CP Application
- RTE
- BSW
- OS

SW-Cs are implemented as sets of Runnables.

Deployment, mapping of Runnables to tasks and tasks to cores

Communication Bus, e.g. CAN

Runnable my10ms_worker_runnable
Runnable my5ms_worker_runnable
(...)

11
Scheduling: what does the OS do?

Tasks in OSEK: container for code, e.g. several runnables
The RTE adds another layer of scheduling on top of the OS.

Scheduling: what does the RTE do?

```c
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();
        }
    }
}
```
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();
        }
    }
}
Step 3: AUTOSAR AP top-down

Layers (of abstraction)

1. shared

2. CP

3. AP

4. shared

Today’s tour
In AUTOSAR AP, functionality gets mapped onto “Adaptive Applications” (AA).
What is an **Adaptive Application**?

• Think of it as a program as written for a PC.
  – Plus a description of its services, the *Service Instance Manifest*
  – Plus a description of its execution properties, the *Execution Manifest*
  – It comes with its own main function.

• In contrast to CP, the AP software of an ECU has several main functions, one for each AA. → Just like on your PC.
Adaptive Application: example

```c
int main(int argc, char *argv[]) {
    int retval;
    // initialize App data here

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

    // save persistent App data and free all resources here

    return retval; // terminate with success
}
```

Ups, one important thing missing for AP…
Adaptive Application: example

```c
int main(int argc, char *argv[]) {
    int retval;
    // initialize App data here
    ExecutionClient.ReportProcessState(kRunning);
    // call App code here (which may or may not return), e.g.:
    // retval = AppCode();
    ExecutionClient.ReportProcessState(kTerminating);
    // save persistent App data and free all resources here
    return retval; // terminate with success
}
```

The Application must report its state to the Execution Manager.
Implementation, System Configuration

Software Architecture

Implementation, System Configuration

Functionality C

Mapping of Functionality to Adaptive Applications (AA)

Application 1

Application 2

AP Applications are implemented as POSIX applications.

SOFTWARE PACKAGE

Application 1
executable and data

Execution Manifest

Service Instance Manifest

APPLICATION 2
executable and data

Execution Manifest

Service Instance Manifest

Machine Manifest

SOFTWARE PACKAGE
Deployment can take at runtime (“adaptive”).
For those familiar with Linux:
The Execution Manager is similar to systemd, each AA resembles a systemd service.
Deterministic Client

• **Definition** *Deterministic Client [1]*

Adaptive Application interface to *Execution Management* to support control of the **process-internal cycle**, a deterministic worker pool, activation time stamps and random numbers.

• Using the Deterministic Client is **optional**.

• In the following we will concentrate on the “**process-internal cycle**” aspect only.
Deterministic Client: example

```c
int AppCode(void)
{
    ActivationReturnType dccType;  // Deterministic Client
    // Cycle (DCC) type

    while (1) {  // endless loop
        dccType = DeterministicClient.WaitForNextActivation();  // each execution of the code below is one "Cycle"
        switch (dccType) {
            case kRegisterServices:
                // call handler registering services here
                break;
            case kServiceDiscovery:
                // call service discovery handler here
                break;
            case kInit:
                // call init handler here
                break;
            case kRun:
                // call cyclic App handler here
                break;
            case kTerminate:
                return 0;  // terminate with success
            default:  // invalid return value
                return 1;  // terminate with error
        }
    }
}
```

If the process uses the Deterministic Client, the App code called from the main function shown earlier could look like this.

The Deterministic Client comes with different cycle types. See switch-case values.

Loop body: each execution = one cycle
Deterministic Client Cycle (DCC)
POSIX processes start with a single (main) thread but can spawn new threads.

POSIX Scheduling

- **Ready**: Wake up, Switch, Preempt, Admit
- **Running**: Scheduling Latency, Wait (also “Block” or “Sleep”), Terminate
- **Waiting**: WAITT: accumulated time spent in “Waiting”
- **New**, **Non-existing**, **Done**: Create, Free

(*) When calculating RUNT, the time “stolen” (interrupts etc.) is deducted
Timing of Threads; definition of CET

\[ \text{CET} = \text{RUNT1} + \text{RUNT2} + \text{RUNT3} + \text{RUNT4} + \text{RUNT5} + \text{RUNT6} \]
Proposed timing parameter mapping

The CET for an App Process accumulates the run-time of all its threads. As a consequence, the CET might be greater than the GET.

App Process

- thread 1
  - RUNT1
  - RUNT2

- thread 2
  - RUNT3

- thread 3
  - RUNT4
  - RUNT5
  - RUNT6

\[\text{CET} = \text{RUNT1} + \text{RUNT2} + \text{RUNT3} + \text{RUNT4} + \text{RUNT5} + \text{RUNT6}\]
Step 2: AUTOSAR CP top-down
## Timing parameters

### OSEK/AUTOSAR CP and AUTOSAR AP Deterministic Client

<table>
<thead>
<tr>
<th>ABR.</th>
<th>Explanation</th>
<th>ABR.</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>IPT</td>
<td>initial pending time</td>
<td>PER</td>
<td>period</td>
</tr>
<tr>
<td>CET</td>
<td>core execution time</td>
<td>ST</td>
<td>slack time</td>
</tr>
<tr>
<td>GET</td>
<td>gross execution time</td>
<td>PRE</td>
<td>Preemption time (AUTOSAR CP only)</td>
</tr>
<tr>
<td>RT</td>
<td>response time</td>
<td>DL</td>
<td>Deadline (&quot;max. RT&quot;)</td>
</tr>
<tr>
<td>DT</td>
<td>delta time</td>
<td>NST</td>
<td>Net slack time</td>
</tr>
</tbody>
</table>

### POSIX OS Scheduling

<table>
<thead>
<tr>
<th>ABR.</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL</td>
<td>Scheduling Latency</td>
</tr>
<tr>
<td>WAITT</td>
<td>accumulated time spent in &quot;Waiting&quot;</td>
</tr>
<tr>
<td>RUNT</td>
<td>accumulated time spent in &quot;Running&quot;</td>
</tr>
</tbody>
</table>

**CPU-utilization (also referred to as CPU-load):** the sum $U$ of all CETs within a defined time-frame $t_O$ (the "observation frame") in relation to $t_O$ times the number of relevant cores ($n_C$).

Such CETs can relate to a specific code-fragment, a runnable, a task, a thread, a process, a core or the complete system.

$$U = \sum_{n=1}^{N} \frac{CET(n)}{n_C \cdot t_O}$$
Posters by GLIWA

Download PDF from gliwa.com or request a free hard-copy

Timing in AUTOSAR CP/AP

ISO 26262

Multi-core

Timing Analysis
Status of ARTI

- User selects events to trace
- AS components provide hooks
- Trace-tools provide trace-code

**Data exchange format specification for**
- Model ("system configuration")
- Traces
- Timing parameters

**ARTI (AUTOSAR / ASAM Run-Time Interface)**
- AUTOSAR draft release in October 2018
- ASAM project started in 2019
- Left side of V-model: AUTOSAR, right side: ASAM (cf. A2L)
Status of TIMEX for AUTOSAR AP

• Expert discussions ongoing
  – General discussion on how to address timing in AP
  – Definition of Events in AP
    → TIMEX always needs items (in the AUTOSAR meta-model) which can be referenced

• Target: first version of with R19-11 (AUTOSAR release in November this year)
Summary

• I hope you enjoyed today’s tour through CP/AP Timing!

• AP brings many completely new aspects (compared to CP)
  – However, as AP is POSIX-based, we can apply some of our Linux, QNX, etc. experience.
  – With the right mapping / definition of timing-parameters we can reuse some of the CP (timing) ideas.
    → possibly standardize the mapping?

• It is the (trace) tool-vendors task to build bridges from CP to AP.
Contents

- Basics (Compilers, RTOSs, processors)
- Timing theory
- Timing analysis techniques
- Examples from automotive projects
- Timing optimization
- Multi-core, many-core
- AUTOSAR
- Safety, ISO 26262
Thank you
References

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

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

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

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

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

and safety-related systems
AUTOSAR AP Release 18-10
https://www.autosar.org/standards/adaptive-platform/

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

[8] Peter Gliwa, GLIWA, A systematic approach for timing requirements
EMCC (Embedded Multi-Core Conference) 2018 in Munich
https://gliwa.com/downloads/

[9] The Open GROUP, POSIX Standard
https://publications.opengroup.org/standards/unix

[10] WIKIPEDIA, Hypervisor
https://en.wikipedia.org/wiki/Hypervisor

https://en.wikipedia.org/wiki/POSIX

[12] Red Hat, Topics
https://www.redhat.com/en/topics/containers

Communication, Concurrency, and Threads
Prentice Hall, 2003

[14] SOA Manifesto Authors, SOA Manifesto
http://www.soa-manifesto.org

and Design within the AUTOSAR Development Process
https://www.autosar.org/standards/classic-platform/