Automotive SW Architecture:
Engine Management Systems

ETR 13
École d'Été Temps Réel 2013
28/08/2013

Denis Claraz – Continental Automotive France
denis.claraz@continental-corporation.com
Introduction : Plan of the presentation

1. Introduction / Context

2. Static architecture

3. Dynamic architecture

4. Coding

5. Multi-Core

6. AUTOSAR

7. Conclusion
# Divisions and Business Units

## Continental Corporation

### Rubber Group

**Passenger and Light Truck Tires**
- Original Equipment
- Replacement Europe
- Replacement The Americas
- Replacement Asia
- Two-Wheel Tires

**Commercial Vehicle Tires**
- Truck Tires Europe
- Truck Tires The Americas
- Truck Tires Asia
- Industrial Tires

**ContiTech**
- Air Spring Systems
- Benecke-Kaliko
- Conveyor Belt
- Elastomer Coatings
- Fluid Technology
- Power Transmission
- Vibration Control
- Other Operations

## Continental Automotive

### Chassis & Safety
- Electronic Brake Systems
- Hydraulic Brake Systems
- Sensorics
- Passive Safety & ADAS
- Chassis Components

### Interior
- Body & Security
- Connectivity
- Commercial Vehicles & Aftermarket
- Instrumentation & Displays
- Interior Modules
- Multimedia

### Powertrain
- Transmission
- Hybrid Electric Vehicle
- Sensors & Actuators
- Engine Systems

### Employees Sales
- Continental Automotive: 148,000 employees, 26 bn € sales
- Automotive Systems Division Powertrain: 13,000 employees, 2.4 bn € sales
- Software (SW): 1,100 employees, 1.100 M € sales

---

*Status: Nov. 2011*
Market Driver: Emissions standards / CO2 reduction

Diesel Engines

1996: EURO 2
2000: EURO 3
2005: EURO 4
2009: EURO 5
2014: EURO 6

Sources: European Commission, EPA
Consequence : Evolution of complexity (Powertrain)

Example High End Project:
- OEM, 3rd party, competitors code
- 900 system functions
- 200 I/O
- 250,000 lines of code

High End:
- x 10 / 7 years

In average:
- x 10 / 10 years

- 32 bit Controller
- OSEK operating system
- SULEV Emissions
- Electronic throttle control
- ULEV Emissions
- Variable valve timing (Inlet and outlet)
- TLEV Emissions
- OBD-2 Diagnosis
- Sequential Injection
- Knock control
- 16 bit Controller
- C-Language

Average program size (kB)
Max program size (kB)
Average ECU price
ROM (kB)
**Consequence: High Reuse orientation**

- Reuse by Reference (“SW Factory”) since 90’s
  - Generic teams develop generic reusable (& configurable) components
  - Project teams integrate generic components and configure them
- Problem:
  - Compositionality & composability of Timing Constraints & Properties ?
  - How to ensure that a SW-C developed in a Generic Team works in a Specific Project?
- Solution: Platform approach
  - Reference Architecture
  - Control of diversity
  - Standardized process, method, tools
# Introduction: Plan of the presentation

<table>
<thead>
<tr>
<th>1. Introduction / Context</th>
<th>1. Functionnal partitionning</th>
</tr>
</thead>
<tbody>
<tr>
<td>2. Static architecture</td>
<td>2. Aggregate concept</td>
</tr>
<tr>
<td>3. Dynamic architecture</td>
<td>3. Variability</td>
</tr>
<tr>
<td>4. Coding</td>
<td></td>
</tr>
<tr>
<td>5. Multi-Core</td>
<td></td>
</tr>
<tr>
<td>6. AUTOSAR</td>
<td></td>
</tr>
<tr>
<td>7. Conclusion</td>
<td></td>
</tr>
</tbody>
</table>
Static Architecture : Functional Partitioning

Common Functional Architecture Aggregate Groups

Vehicle

Powertrain

Powertrain Management

Engine - Gasoline or Diesel

Engine Position & Speed

Air

Exhaust Gas

Electric Drive

Chassis

Torque

Ignition (Gasoline)

Combustion Process

Transmission

Body & interior

Engine States

Fuel

Engine Cooling & Lubrication

Electric Power

System Manager

Basic ECU Functions

Communication

Automotive Systems
Division Powertrain

9 / Denis Claraz / January, 28, 2013 © Continental Automotive SAS
Static Architecture: Functional Partitioning

Common Functional Architecture Aggregate Groups

Application SW (control)

Powertrain control block

Engine control block

Transmission control block

Vehicle control block

Transverse functions

Specialized driver block

Engine

Injection

Ignition

Knock Window

IO Platform driver block

IC Handler driver block

Infrastructure SW

CC block

OSEK OS

OSEK COM/NM

Proc. Mon L3

S & C Reprog

NVMY

KWP

UDS

LIN

CCP tun./flash

XCP

SA

Library

Continental Automotive SAS
Software overview

Engine position & speed:
150 SW-modules
10,000 eloc

80 Aggregates
2,000 ASW SW-C
**Introduction : Plan of the presentation**

<table>
<thead>
<tr>
<th>1. Introduction / Context</th>
<th>1. Functionnal partitionning</th>
</tr>
</thead>
<tbody>
<tr>
<td>2. Static architecture</td>
<td>2. Aggregate concept</td>
</tr>
<tr>
<td>3. Dynamic architecture</td>
<td>3. Variability</td>
</tr>
<tr>
<td>4. Coding</td>
<td></td>
</tr>
<tr>
<td>5. Multi-Core</td>
<td></td>
</tr>
<tr>
<td>6. AUTOSAR</td>
<td></td>
</tr>
<tr>
<td>7. Conclusion</td>
<td></td>
</tr>
</tbody>
</table>
The development of an Aggregate follows a defined & formalized process, with planning, milestones, reviews, delivery …
Static Architecture: Aggregate Process

All facets included in the Aggregate

- Function Description
  - Software Requirement Specification
  - Engine function expertise

- Supported System Configurations
  - Engine function expertise

- Simulation Models Control & Plant
  - Engine function expertise

- Calibration Hints Default Calibration
  - Engine tuning expertise

- Electronics Interface Specification
  - HW expertise

- Component Specification
  - Components expertise

- Software Design Specification, Code, Validation
  - SW expertise

- Validation Report Design Reviews
  - Engine function expertise

Aggregate Information managed as one Package
Introduction: Plan of the presentation

1. Introduction / Context
2. Static architecture
3. Dynamic architecture
4. Coding
5. Multi-Core
6. AUTOSAR
7. Conclusion

1. Functionnal partitionning
2. Aggregate concept
3. Variability
Introduction: Plan of the presentation

<table>
<thead>
<tr>
<th>1. Introduction / Context</th>
<th>1. Context / Link vs. Static</th>
</tr>
</thead>
<tbody>
<tr>
<td>2. Static architecture</td>
<td>2. Scheduling strategy</td>
</tr>
<tr>
<td>3. Dynamic architecture</td>
<td>3. Integration / Sequencing</td>
</tr>
<tr>
<td>4. Coding</td>
<td>4. Data consistency</td>
</tr>
<tr>
<td>5. Multi-Core</td>
<td></td>
</tr>
<tr>
<td>6. AUTOSAR</td>
<td></td>
</tr>
<tr>
<td>7. Conclusion</td>
<td></td>
</tr>
</tbody>
</table>
Context: Mixture of Time and Angle domains

Time based Events

Angular Events

TDC 2

TDC 3

60% 50% 40% 30% 20% 10% 0% of ROM size

Number of Cylinders

Recurrence of Top Dead Center

<table>
<thead>
<tr>
<th>Cylinders</th>
<th>@ 500 rpm</th>
<th>@ 6000 rpm</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>40 ms</td>
<td>3.3 ms</td>
</tr>
</tbody>
</table>

2 architectures in one CPU / one OS
Optimized design : Limited HW resources

\[ \text{var} = \text{interpolation (n, maf)} \times \text{interpolation (tco, t_ast)} \times \text{interpolation (tia)} \]

| tdc     | 100ms | 1 sec |

CPU Load at 6000 rpm:

Complete calculation at tdc: \( \text{cpu load} = 0.12\% \)

Calculation split between tdc, 100ms, 1s: \( \text{cpu load} = 0.06\% \)

Dynamic architecture impacted by Core Resources optimization

18 / Denis Claraz / January, 28, 2013 © Continental Automotive SAS
Optimized coding: Limited HW resources

"Bad" example

/* O2 sensor diagnosis conditions */
if (  
  lv_ls_up_diag  
  &&  LV_CDN_INH_DIAG_VLS_UP  
  &&  lv_end_ls_up_diag  
  &&  lv_thd_vls_ast  
  &&  lv_tscl_cor  
  &&  lv_tco_min_cat  
  &&  maf < c_maf_max_vls  
  &&  maf > c_maf_min_vls  
  &&  n_32 < c_n_max_vls  
  &&  n_32 > c_n_min_vls  
  &&  maf_kgh > c_maf_kgh_max_vls  
  &&  maf_kgh < c_maf_kgh_max_vls  
  &&  maf_kgh_mmv_dif < c_maf_max_dif_vls  
  &&  vs > c_vs_min_vls  
  &&  vs < c_vs_max_vls  
  &&  amp >= c_max_dep_vls  
  &&  cppwm_cps < c_cppwm_cps_max_ofs  
  &&  lv_up_lsh  
  &&  ...  )

"Good" example

/* O2 sensor diagnosis conditions */
if (  
  n_32 < c_n_max_vls  
  &&  maf_kgh < c_maf_kgh_max_vls  
  &&  vs < c_vs_max_vls  
  &&  maf < c_maf_max_vls  
  &&  lv_ls_up_diag  
  &&  LV_CDN_INH_DIAG_VLS_UP  
  &&  lv_end_ls_up_diag  
  &&  lv_thd_vls_ast  
  &&  lv_tscl_cor  
  &&  lv_tco_min_cat  
  &&  maf > c_maf_min_vls  
  &&  maf > c_maf_min_vls  
  &&  n_32 < c_n_max_vls  
  &&  n_32 > c_n_min_vls  
  &&  maf_kgh > c_maf_kgh_min_vls  
  &&  maf_kgh < c_maf_kgh_max_vls  
  &&  maf_kgh_mmv_dif < c_maf_max_dif_vls  
  &&  vs > c_vs_min_vls  
  &&  vs < c_vs_max_vls  
  &&  amp >= c_max_dep_vls  
  &&  cppwm_cps < c_cppwm_cps_max_ofs  
  &&  lv_up_lsh  
  &&  ...  )

Readability of the spec :  
the tests are grouped

Order of test may be different than spec, to realize directly the condition
## Introduction : Plan of the presentation

1. Introduction / Context
2. Static architecture
3. Dynamic architecture
4. Coding
5. Multi-Core
6. AUTOSAR
7. Conclusion

<table>
<thead>
<tr>
<th>1. Context / Link vs. Static</th>
</tr>
</thead>
<tbody>
<tr>
<td>2. Scheduling strategy</td>
</tr>
<tr>
<td>3. Integration / Sequencing</td>
</tr>
<tr>
<td>4. Data consistency</td>
</tr>
</tbody>
</table>
Scheduling strategy

**Preemptive**
High priority task interrupts current task at any time before end

- Large Resource Consumption

**Cooperative**
High priority task interrupts current task at pre-defined schedule points (every x μs)

- Controlled Response Time
- Minimized Resource Consumption
- Data Consistency for free
- Increased Maintenance Effort

**Non preemptive**
High priority task waits until end of current task

- Large Response Time

Automotive Systems Division Powertrain

21 / Denis Claraz / January, 28, 2013 © Continental Automotive SAS
Scheduling strategy

- Deadline defines the scheduling strategy
  - OSEK is a **fixed priority based scheduler**: If more than one task is ready to execute, then the task with the highest priority is chosen (+ FIFO)
  - Priorities **fixed** at design time according to **Deadline monotonic scheduling (DMS)**
    Tasks with shorter deadline have a higher priority
  - To save resources, tasks with similar deadline get same priority

---

**Activation**

- **suspended**
- **ready**
- **running**
- **suspended**

**Priority**

- **Interrupts**
- **Preemption**
- **Cooperative Environment**
- **Background**

**Deadline**

- **Deadline < 100 µs**
- **Deadline > 1 s**

**Tasks**

- **Response time**
- **Delay**

---

**Automotive Systems**
**Division Powertrain**

© Continental Automotive SAS
Scheduling of EMS Applications on Multi-Cores
Practical example for Scheduling: Simulation of Task Sets

1. Differentiation of calculations to deadlines
2. Priorities corresponding to deadlines (DMS)
3. Schedule points for a defined blocking time
4. Preemption for tasks with short deadline

Response Time / Deadline

• response time min
• response time avg
• response time max

Example 0
## Task Details

<table>
<thead>
<tr>
<th><strong>Activation Pattern</strong></th>
<th>Describes the activation pattern (periodic/aperiodic/sporadic …)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Activation condition(s)</strong></td>
<td>Describes necessary conditions for task activation (e.g. engine must be running)</td>
</tr>
<tr>
<td><strong>Activated by</strong></td>
<td>The module / aggregate that activates the task (if there is one)</td>
</tr>
<tr>
<td><strong>Fastest Recurrence</strong></td>
<td>Fastest recurrence</td>
</tr>
<tr>
<td><strong>Phasing</strong></td>
<td>Phasing to other tasks (if any)</td>
</tr>
<tr>
<td><strong>Deadline</strong></td>
<td>Deadline of this task (if any)</td>
</tr>
<tr>
<td><strong>Impact of DL miss</strong></td>
<td>Impact if the deadline is missed (e.g. degradation of quality, fatal)</td>
</tr>
<tr>
<td><strong>Priority</strong></td>
<td>Recommendation for the priority</td>
</tr>
<tr>
<td><strong>Multi-Activation</strong></td>
<td>Recommended value for multi-activation</td>
</tr>
<tr>
<td><strong>Preempt/Cooperative</strong></td>
<td>Either P for preemptive or C for cooperative task</td>
</tr>
<tr>
<td><strong>Expected Runtime</strong></td>
<td>Runtime: expected / max allowed (if known)</td>
</tr>
<tr>
<td><strong>Data / Coupling</strong></td>
<td>Data exchange / coupling with other tasks (if known)</td>
</tr>
<tr>
<td><strong>File</strong></td>
<td>File that contains the task body</td>
</tr>
</tbody>
</table>
Offsets bewteen Tasks (load balancing)

In order to avoid load peaks, the time bases are not activated synchronously.
Verification of Scheduling: Missed deadlines

\[ w_{i,q}^{k+1} = (q + 1)C_i + B_i + \sum_{j \in \text{hp}(i)} \left\lceil \frac{w_{i,q}^k}{T_j} \right\rceil C_j \]

Schedulability analysis

In-situ measurements (instrumentation)

Simulation

Response Time/Deadline

Deadline = 20ms
Introduction: Plan of the presentation

1. Introduction / Context
2. Static architecture
3. Dynamic architecture
4. Coding
5. Multi-Core
6. AUTOSAR
7. Conclusion

1. Context / Link vs. Static
2. Scheduling strategy
3. Integration / Sequencing
4. Data consistency
Sequencing / Data life cycle: The real life

Control of Dataflow

Multi-project approach

High number of Runnables

Automotive Systems Division Powertrain

28 / Denis Claraz / January, 28, 2013 © Continental Automotive SAS
PowerSAR Dynamic Architecture Sequencing: The solution

Sort Functions by Alphabetical Order?

Or by Specification Chapters Order?

Or following Dynamic Architecture recommendation?

... The answer is ... the Phase concept:
PowerSAR Dynamic Architecture Sequencing: Phase Concept

Detection of System Transition
- System transitions detected as soon as possible so that nominal computations benefit from the initialization

Acquisitions & related Diagnoses
- Acquisitions (& diagnosis) done asap, to get the results for the complete Event.

Phases = “Dynamic partitioning”:
- Function development: Definition of the Phase of Runnables
- Integration: Runnables plugged into the defined Phase
- Phases order fixed, standard across SystemEvents & Projects

Execution sequence

**System Variables computation**
- System variables are based on ECU inputs, and are used in a high number of functions

**Calculation of Basic Setpoints**
- Basic setpoints are based on system variables and requests

**Realisation Of Setpoints**
- BSW is informed about new ASW data. Basically, piloting of the HW is done here.

**Data Processing for Next cycle**
- Data needed for next occurrence, or for other Events have no « internal deadline » and are located here. So, they will be displayed with one occurrence delay.

Proprietary and confidential. Distribution only by express authority of Continental AG or its subsidiaries.

30 EMS2-MCR / D.Claraz / May 9th, 2013 © Continental Automotive SAS
Introduction : Plan of the presentation

1. Introduction / Context
2. Static architecture
3. Dynamic architecture
4. Coding
5. Multi-Core
6. AUTOSAR
7. Conclusion

1. Context / Link vs. Static
2. Scheduling strategy
3. Integration / Sequencing
4. Data consistency
Context: Coupling

SW Components control
System Components
Example 1: Counter increment in 2 tasks

Low priority task T1

```
counter++;  
load register from @counter  
inc register  
store register to @counter
```

High priority preemptive task T2

```
counter++;  
load register from @counter  
inc register  
store register to @counter
```

---

<table>
<thead>
<tr>
<th>Active</th>
<th>Step</th>
<th>Reg.</th>
<th>Mem</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>T1</td>
<td>Load</td>
<td>x→5</td>
<td>5</td>
</tr>
<tr>
<td>T1</td>
<td>Inc</td>
<td>5→6</td>
<td>5</td>
</tr>
<tr>
<td>T1</td>
<td>Store</td>
<td>6</td>
<td>5→6</td>
</tr>
</tbody>
</table>

Final Result: 7

<table>
<thead>
<tr>
<th>Active</th>
<th>Step</th>
<th>Reg.</th>
<th>Mem</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>T2</td>
<td>Load</td>
<td>x→5</td>
<td>5</td>
</tr>
<tr>
<td>T2</td>
<td>Inc</td>
<td>5→6</td>
<td>5</td>
</tr>
<tr>
<td>T2</td>
<td>Store</td>
<td>6</td>
<td>5→6</td>
</tr>
<tr>
<td>T1</td>
<td>Load</td>
<td>x→6</td>
<td>6</td>
</tr>
<tr>
<td>T1</td>
<td>Inc</td>
<td>6→7</td>
<td>6</td>
</tr>
<tr>
<td>T1</td>
<td>Store</td>
<td>7</td>
<td>6→7</td>
</tr>
</tbody>
</table>

Final Result: 7

---

<table>
<thead>
<tr>
<th>Active</th>
<th>Step</th>
<th>Reg.</th>
<th>Mem</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>T2</td>
<td>Load</td>
<td>x→5</td>
<td>5</td>
</tr>
<tr>
<td>T2</td>
<td>Inc</td>
<td>5→6</td>
<td>5</td>
</tr>
<tr>
<td>T2</td>
<td>Store</td>
<td>6</td>
<td>5→6</td>
</tr>
<tr>
<td>T1</td>
<td>Inc</td>
<td>5→6</td>
<td>6</td>
</tr>
<tr>
<td>T1</td>
<td>Store</td>
<td>6</td>
<td>6→6</td>
</tr>
</tbody>
</table>

Final Result: 6
Data consistency

Example 2: Copying 64 bit data on a 32 bit controller

Low priority task T1

```c
u64 a;
...
a = AAAAAAAABBBBBBBBU;
...
```

High priority preemptive task T2

```c
a = 1111111122222222U;
```

<table>
<thead>
<tr>
<th>Active</th>
<th>Step</th>
<th>Register</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>T1</td>
<td>Load upper</td>
<td>AAAAAAAA</td>
<td>xxxxxxxxxxxxxxxxxx</td>
</tr>
<tr>
<td>T1</td>
<td>Store upper</td>
<td>AAAAAAAA</td>
<td>AAAAAAAAxxxxxxx</td>
</tr>
<tr>
<td>T1</td>
<td>Load lower</td>
<td>BB BBBB</td>
<td>AAAAAAAAxxxxxxx</td>
</tr>
<tr>
<td>T1</td>
<td>Store lower</td>
<td>BB BBBB</td>
<td>AAAAAAABB BBBB</td>
</tr>
<tr>
<td>T2</td>
<td>Load lower</td>
<td>2222222</td>
<td>11111111xxxxxxx</td>
</tr>
<tr>
<td>T2</td>
<td>Store lower</td>
<td>2222222</td>
<td>111111112222222</td>
</tr>
<tr>
<td>T1</td>
<td>Load lower</td>
<td>BB BBBB</td>
<td>111111112222222</td>
</tr>
<tr>
<td>T1</td>
<td>Store lower</td>
<td>BB BBBB</td>
<td>11111111BB BBBB</td>
</tr>
</tbody>
</table>
Data consistency

Example 3: Calculation of average acquisition with reset

Low priority Task T1

```c
/* Calculate the average */
if (Counter != 0) {
    average = Sum/Counter;
}
/* Reset Sum and Counter */
Sum = 0;
Counter = 0;
```

High priority preemptive Task T2

```c
Sum += new_acquisition;
Counter++;```

1. Wrong average: Sum new, Counter old (Counter loaded once in register, reused twice)

2. Average ok, but one acquisition of Sum and Counter is lost

3. Wrong next average: Sum incremented, but one Counter is missing
Introduction: Plan of the presentation

1. Introduction / Context
2. Static architecture
3. Dynamic architecture
4. Coding
5. Multi-Core
6. AUTOSAR
7. Conclusion
Why C Coding rules (1)

/* write the first 10 Fibonacci numbers into x[] */

```c
for(u8_least i=0;i<10;i++)
    x[i]=(i==0?0:i<=2?1:((i-1)%2?-1:1)+x[i-1]*x[i-1])/x[i-2];
```

Need for correct, readable, and understandable code

Embedded Systems:

=> The generated ASM code matters !!
Why C Coding rules (2)

near u16 *ptr1;
far u16 *ptr2;
...

if (lv_a == true) {
    lv_b = (c != 5);
    lv_c = false;
}
...

ISO-C Compatibility and platform independance to be ensured

near/far not defined by ISO

"true" not defined by ISO
Why C Coding rules (3)

\[
\begin{align*}
s16 \ x &= -32000; \\
u16 \ y &= 2; \\
if \ ((x \ / \ y) < 0) & \ldots
\end{align*}
\]

\[x \ / \ y = 16768\]
on 16 bit systems

\[-32000 \equiv 1000 \ 0011 \ 0000 \ 0000\]
x is promoted to u16
\[\rightarrow 1000 \ 0011 \ 0000 \ 0000 \equiv 33536\]
\[\rightarrow 33536 \ / \ 2 = 16768\]

\[
\begin{align*}
s32 \ x &= -32768; \\
s32 \ y &= \text{0x8000}; \\
if \ (x \ == \ y) & \ldots
\end{align*}
\]

\[
\begin{align*}
s32 \ x &= -32768; \\
s32 \ y &= \text{0x8000}; \\
if \ (x \ == \ y) & \ldots
\end{align*}
\]

\[
\begin{align*}
\text{correct} \\
s32 \ y &= -(s32) \text{0x8000U};
\end{align*}
\]

\[
\begin{align*}
\text{if} \ (x \ == \ y) & \ldots
\end{align*}
\]

\[
\begin{align*}
\text{else} & \ldots
\end{align*}
\]

\[
\begin{align*}
\text{if} \ (-x \ == \ y) & \ldots
\end{align*}
\]

\[
\begin{align*}
\text{else} & \ldots
\end{align*}
\]

\[
\begin{align*}
\text{if} \ (-x \ == \ y) & \ldots
\end{align*}
\]

\[
\begin{align*}
\text{else} & \ldots
\end{align*}
\]

\[
\begin{align*}
\text{if} \ (-x \ == \ y) & \ldots
\end{align*}
\]

\[
\begin{align*}
\text{else} & \ldots
\end{align*}
\]

\[
\begin{align*}
a &= 1, \text{if 32 Bit platform} \\
a &= 2, \text{if 16 Bit platform}
\end{align*}
\]

Portability between different targets to be ensured

What is the value of 'a'

\[a = 1, \text{if 32 Bit platform}
\]

\[a = 2, \text{if 16 Bit platform}\]
Why C Coding rules (4)

```c
u8 div(u8 val, u8 idx) {
    return val / a[idx];
}
```

c = 5;
...
if (c = 4)
...

Safe and robust code to be ensured

Good:
c = 5;
...
if (c == 4)
...

Better:
c = 5;
...
if (4 == c)
...
**C Coding Rules Basis**

The C Coding Rules are based on

- **ANSI Standard**
- **ISO/IEC 9899:1990 Standard**
  - exceptions are *inline* functions (based on ISO/IEC 9899:1999 Standard) and inline "asm"
- **MISRA-C:2004**
  (Motor Industry Software Reliability Association)
- **HIS Subset of MISRA** (based on MISRA-C 1.0)
- **AUTOSAR C Implementation Rules**
Floating Point: IEEE

The IEEE has standardized the computer representation for binary floating-point numbers in IEEE 754.

This standard provides two basic formats –

<table>
<thead>
<tr>
<th>Single Precision</th>
<th>Double Precision</th>
</tr>
</thead>
<tbody>
<tr>
<td>C language : 'float'.</td>
<td>C language : 'double'.</td>
</tr>
<tr>
<td>Size : 32 bits.</td>
<td>Size : 64 bits.</td>
</tr>
<tr>
<td>Significand (fraction / mantissa) precision of 24 bits (i.e. about 7.22 decimal precision).</td>
<td>Significand precision of 53 bits (i.e. about 15.99 decimal precision).</td>
</tr>
</tbody>
</table>
Floating Point: IEEE 754 Format

Single Precision Data (32 bit) Format:

Bit representation of floating point constant in IEEE 754 format

Ex: +10.0 (dec) => +1010.0 (bin) => +1.01 * 2^3

(-1)^{sign} \cdot 2^{(exponent - bias)} \cdot (1 + mantissa \cdot 2^{-23})

(-1)^0 \cdot 2^{(130 - 127)} \cdot (1 + 2^{21} \cdot 2^{-23}) = 2^3 \cdot (1 + 2^{-2}) = 8.0 \cdot 1.25 = 10.0f
Floating Point: Rounding errors (1)

Rounding Errors in Basic Operations

\[ E = (A + B) \times (C + D) \]
\[ F = (A \times C) + (A \times D) + (B \times C) + (B \times D) \]

Where \( A = 1.1, B = 2.2, C = 3.3, D = 4.4 \) and

Expected Result is 25.41

IF \((E == F)\) THEN
\begin{align*}
E & \text{ equals } F \\
\text{ELSE} & \\
E & \text{ not equal to } F \\
\text{ENDIF}
\end{align*}

Obtained Result: \(25.4100018 \neq 25.4099998\)

- Rounding errors during the calculation of the two values being compared.
- if \(|E-F| < \text{min}\) better than if \((E == F)\)
- Order of evaluation can affect the result.
Adding / Subtracting Values with very Different Magnitudes

\[
\left[ \frac{a + \frac{1}{a}}{a} \right] + \left[ \frac{a - \frac{1}{a}}{a} \right] 
\times \left[ \frac{a + \frac{1}{a}}{a} \right] - \left[ \frac{a - \frac{1}{a}}{a} \right]
\]

= \(2a \times \frac{2}{a}\)

= 4
Introduction : Plan of the presentation

1. Introduction / Context
2. Static architecture
3. Dynamic architecture
4. Coding
5. Multi-Core
6. AUTOSAR
7. Conclusion

1. Motivation / Constraints
2. Elements of solution
Multi Core SW Architecture MCSA

Why Multi Core

Helps to Resolve Contradicting Requirements

Increasing performance requirements
\[ \Rightarrow \text{classical approach: operating frequency increase} \]

Reduction of power dissipation \( P_d \)
\[ \Rightarrow \text{classical approach: operating frequency reduction} \]

MultiCore is the new Market Standard
Scheduling of EMS Applications on Multi-Cores

Scalability of Cores*

Different performance needs for different application classes require a flexible approach in the controller architecture

*Only cores relevant for running EMS software independently are shown
Multi Core SW Architecture MCSA
Technical challenges for Multi Core Software

- Function calls from one core to another
  - Synchronous calls
  - Asynchronous calls

- Runnables which are today called in a sequence may run in future in parallel
  - Low priority tasks can “overtake” high priority on the other core
  - Data consistency issues, update of global data by different producers
  - Concurrent access Spin lock, Wait states, Synchronizations

- How to allocate runnables to cores in an efficient way
  - According to the static software architecture (by SWCs)
  - According to the dynamic software architecture (by Tasks/Processes/Runnables)

- What’s the right approach to distribute the runnables
  - Statically at software compile time
  - Dynamically at software execution time

- How to prepare existing legacy software (EMS2) to be MultiCore Ready
Introduction : Plan of the presentation

1. Introduction / Context
2. Static architecture
3. Dynamic architecture
4. Coding
5. Multi-Core
6. AUTOSAR
7. Conclusion

1. Motivation / Constraints
2. Elements of solution
Scheduling of EMS Applications on Multi-Cores ASW Partitioning

Static Architecture is based on functional grouping based on diversity management based on data flow encapsulation designed on specification level starting point for reuse and maintenance.

Dynamic Architecture is based on sequential and priority grouping based on efficiency based on robustness packaged into OS tasks.
Scheduling of EMS Applications on Multi-Cores
ASW Partitioning

Runnable assignment according to Static Architecture

Core 1
- Engine Speed
- Ignition
- Intake Model
- ENSD SEG
- IGRE SEG
- INSY SEG
- IGRE 100ms
- INSY 10ms
- INSY 100ms

Core 2
- Fuel Mass Set point
- Injection
- FMSP SEG
- INJR SEG
- FMSP 10ms
- INJR 10ms
- INJR 100ms

Calculation sequences will be broken

Increased communication overhead and possibilities for spin locks are added
Scheduling of EMS Applications on Multi-Cores
ASW Partitioning

Runnable assignment according to Dynamic Architecture

Core 1
Calculation sequences mostly kept

Core 2
Communication during parallel execution?

Communication mostly necessary at task end
Data Consistency: Single Core Design Patterns not applicable !!

> Exemple Copy-Until-Consistent (CuC)

```c
ctr++;  // increment data counter */
/* Get values from ADC */
val1 += GetValueADC(CHANNEL5);
val2 += GetValueADC(CHANNEL7);
val3 += (val1 / val2);

/* copy data until it is consistent */
{
    tmp_ctr = MAKE_VOLATILE(u8, ctr);
    tmp_val1 = MAKE_VOLATILE(u16, val1);
    tmp_val2 = MAKE_VOLATILE(u16, val2);
    tmp_val3 = MAKE_VOLATILE(u16, val3);
    while (tmp_ctr != MAKE_VOLATILE(u8, ctr));
}
```
Introduction : Plan of the presentation

1. Introduction / Context
2. Static architecture
3. Dynamic architecture
4. Coding
5. Multi-Core
6. AUTOSAR

7. Conclusion
Conclusion (1/2)

- High coupling: ES functions control the same physical process
- High reuse orientation @ ES: Maybe sometimes too far…
- Component Based Development: Projects integrate configurable solutions

- Cooperative Scheduling: Trade-off Response Time/Resource consumption/Consistency
- Sequence and Consistency key issues: Dependence of Runnables, Independence of Tasks

- Architecture standardization: Functions designed to fit into Platform Tasks
- Use of DMS: Difficulty to evaluate deadlines (cultural problem, robustness margin, …)
- Verification of Architecture by static or in-situ measurements, simulation
Conclusion (2/2)

Future challenges:

- More openness of Platform (box business)
- Multicore
- AUTOSAR compatibility (efficiency, support of basic concepts, independence)
- Process efficient development & integration
Thank you for your Attention