Real-time Systems Engineering @ Bosch
Dr. Arne Hamann, Robert Bosch GmbH
Real-time Systems Engineering @ Bosch

Outline

- Performance modeling & analysis of classical automotive systems
  - Motivation ... or the real complexity
  - Amalthea performance model
  - Current usage @ Bosch
  - Upcoming challenges

- Communication centric design in multi-core systems
  - Importance of cause-effect chains
  - Issues with concurrent execution in multi-core systems
  - Communication mechanisms as solution & impact on latencies
  - Experiments

- Timing-aware control design
  - Control and real-time systems engineering – two worlds collide
  - Co-engineering approach
  - Example
Outline

- Performance modeling & analysis of classical automotive systems
  - Motivation ... or the real complexity
  - Amalthea performance model
  - Current usage @ Bosch
  - Upcoming challenges

- Communication centric design in multi-core systems
  - Importance of cause-effect chains
  - Issues with concurrent execution in multi-core systems
  - Communication mechanisms as solution & impact on latencies
  - Experiments

- Timing-aware control design
  - Control and real-time systems engineering – two worlds collide
  - Co-engineering approach
  - Example
Performance modeling & analysis of classical automotive systems

Typical Classic Automotive Software Architecture Pattern

- **Functional view**
  - Runnables share variables and depend on each other
  - SWC_x = Component X
  - P_y = Runnable Y

- **Dynamic view**
  - 1ms
    - P_1, P_2, P_3, P_4, P_5, P_6
  - n-sync
    - P_1, P_2, P_3, P_4, P_5, P_6
  - 10ms
    - P_1, P_2, P_3, P_4, P_5, P_6
  - 20ms
    - P_1, P_2, P_3, P_4, P_5, P_6

- **Mapping**
  - Some variables are shared between components but in fact they are shared between runnables as well
  - Task
  - P_y = Runnable Y in a task
Performance modeling & analysis of classical automotive systems

Typical Distribution Pattern – Task-Level Parallelism

Task $P_y$ = Runnable $Y$ in a task

Runnables of an SWC or of a supplier reside on multiple cores
Performance modeling & analysis of classical automotive systems

The Real Complexity...

Functional view

SWC_1

P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow P_4 \rightarrow P_5 \rightarrow P_6

SWC_2

P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow P_4

P_5 \rightarrow P_6

SWC_3

P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow P_4

P_5 \rightarrow P_6

SWC_4

P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow P_4

P_5 \rightarrow P_6

SWC_X = Component X

P_y = Runnable Y

Fine-grain, legacy SW sharing between OEM and Tier1 with multiple dependencies
Performance modeling & analysis of classical automotive systems

Tasks to solve during migration to multi-core...

- Maintain single-core dependencies
- Ensure data consistency
- Balance core load
- Optimize memory placement of variables
- Bound latency of cause-effect chains
- ...

- Need a model capturing the system aspects required to solve those tasks
The Basic Idea

Constraints
Period $T_1 = 2\text{ms}$
Deadline $D_1 = 1.5\text{ms}$
Period $T_2 = 5\text{ms}$
Deadline $D_2 = 5\text{ms}$

Costs
$T_1$ takes $10\mu\text{s}$ on Core0, $20\mu\text{s}$ on Core3

Decisions
Run $T_1$ on Core0
Run $T_2$ on Core1
Offset of $T_2 = 1\text{ms}$

Simulation
System Performance Analysis
Optimization
Platform
Config Generation
Performance modeling & analysis of classical automotive systems

Suitable abstraction level needed
Real-time Systems Engineering @ Bosch

Outline

- Performance modeling & analysis of classical automotive systems
  - Motivation ... or the real complexity
  - Amalthea performance model
  - Current usage @ Bosch
  - Upcoming challenges

- Communication centric design in multi-core systems
  - Importance of cause-effect chains
  - Issues with concurrent execution in multi-core systems
  - Communication mechanisms as solution & impact on latencies
  - Experiments

- Timing-aware control design
  - Control and real-time systems engineering – two worlds collide
  - Co-engineering approach
  - Example
Performance modeling & analysis of classical automotive systems

AMALTHEA Model – Hardware

- Hardware elements
  - ECU
  - Microcontroller
  - Core
  - Memory
  - Network

- Latency Access Path

- Hardware Access Path

© Robert Bosch GmbH 2018. All rights reserved, also regarding any disposal, exploitation, reproduction, editing, distribution, as well as in the event of applications for industrial property rights.
Performance modeling & analysis of classical automotive systems

AMALTHEA Model – Software

- **Software behavior**
  - Tasks, runnables, schedulers, …
  - Description on different levels of abstraction
  - Runnables characterized by
    - Execution time (distribution)
    - Variable access (distribution)
  - Detailed (probabilistic) call sequences possible

- **Operating System behavior**

```plaintext
Overhead

<table>
<thead>
<tr>
<th>OS Overhead Api</th>
</tr>
</thead>
<tbody>
<tr>
<td>Api Terminate Task</td>
</tr>
<tr>
<td>Api Schedule</td>
</tr>
<tr>
<td>Api Request Resource</td>
</tr>
<tr>
<td>Api Release Resource</td>
</tr>
<tr>
<td>Api Set Event</td>
</tr>
<tr>
<td>Api Wait Event</td>
</tr>
<tr>
<td>Api Clear Event</td>
</tr>
<tr>
<td>Api Inter Process Activation</td>
</tr>
<tr>
<td>Api Enforced Migration</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OS Overhead Isr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Isr Category1</td>
</tr>
<tr>
<td>Pre Execution Cycles</td>
</tr>
<tr>
<td>Post Execution Cycles</td>
</tr>
<tr>
<td>Isr Category2</td>
</tr>
<tr>
<td>Pre Execution Cycles</td>
</tr>
<tr>
<td>Post Execution Cycles</td>
</tr>
</tbody>
</table>
```

T_1 takes 10µs on Core0, 20µs on Core3
AMALTHEA Model – Constraints

- Runnable Sequencing Constraints
- Timing Constraints
  - Order Constraint
  - Synchronization Constraint
  - Repetition Constraint
  - Delay Constraint
- Age Constraint
- Reaction Constraint
- Data Age Constraints
- Arrival Curves
- Mapping Constraints
  - Pairing Constraints
  - Separation Constraints
- Property Constraints
Real-time Systems Engineering @ Bosch

Outline

- Performance modeling & analysis of classical automotive systems
  - Motivation … or the real complexity
  - Amalthea performance model
  - Current usage @ Bosch
  - Upcoming challenges

- Communication centric design in multi-core systems
  - Importance of cause-effect chains
  - Issues with concurrent execution in multi-core systems
  - Communication mechanisms as solution & impact on latencies
  - Experiments

- Timing-aware control design
  - Control and real-time systems engineering – two worlds collide
  - Co-engineering approach
  - Example
Performance modeling & analysis of classical automotive systems

PLAT4MC - Multicore Tool Platform @ Bosch Series Production

- Simulation
  - Timing Architect
  - INCHORN
  - SyntaVision
- APP4MC tools
  - Partitioning
  - ... Editor
- Architecture Exporter/Importer
  - EA2AMALTSEA
  - Rhapsody 2AMALTSEA
- Optimization
  - Validations
  - Data Consistency check
  - Locomo
- Supporting tools
  - Merger
  - Visualization
- PL specific tools
  - Multicore Checks
  - Flux2AMALTSEA
- SW sharing
  - DAIMLER
  - BMW

AMALTSEA Model

Trace2AMALTSEA

SCA2AMALTSEA

ELF2AMALTSEA

AR2AMALTSEA

XML2AMALTSEA

ECU Software Development Tools

Trace Tooling

SCA

LLVM

eclipse Platform
Performance modeling & analysis of classical automotive systems

PLAT4MC - Automated AMALTHEA Model Generation

- AUTOSAR Specification
- Architecture Tools
- Trace / Timing
- .c/.h source files
- HW Model/OS Model
- AR2AMALTHEA
- Architecture 2AMALTHEA
- Trace 2AMALTHEA
- SCA2AMALTHEA
- AMALTHEA Model Validations
- AMALTHEA Model MERGER
- 3rd Party Simulation e.g. Timing Architects
- Optimization Tooling (e.g. Partitioning)
- Data Consistency Checks
Performance modeling & analysis of classical automotive systems

Status Quo of Performance Simulation

Discuss, change, refine, optimize

System model

Software

• Application structure, execution cycles, data sizes, communication

Hardware

• Components, frequency, performance factor, access latencies

Design decisions

• Mappings
• Activations

Performance simulation

HW/SW System

• Mapping of scheduler to execution components, task activations

Assessment

• Timing
• Utilization
• Bottlenecks

Event based simulation of dynamic effects, using net execution times of functions + data stalls

Data Stalls

Net execution time

Core 0

Core 1

Current modelling approach suited for (homogeneous) many-core architectures
Optimize placement of variables and code (in system and core local memories) in order to improve execution time load of cores (for a fixed task to core mapping).

- Considers allocation constraints (White List, Black List) as well as call/access statistics.

**Aurix2G 6 cores**
Basis for WATERS Industrial Challenges 2016/17

https://waters2017.inria.fr/challenge/
Real-time Systems Engineering @ Bosch

Outline

- Performance modeling & analysis of classical automotive systems
  - Motivation ... or the real complexity
  - Amalthea performance model
  - Current usage @ Bosch
  - **Upcoming challenges**

- Communication centric design in multi-core systems
  - Importance of cause-effect chains
  - Issues with concurrent execution in multi-core systems
  - Communication mechanisms as solution & impact on latencies
  - Experiments

- Timing-aware control design
  - Control and real-time systems engineering – two worlds collide
  - Co-engineering approach
  - Example
Trends in automotive E/E systems

Large-scale integration of heterogeneous applications on (Cross)-Domain & Vehicle Centralized E/E Architectures

Heterogeneous HW platforms to satisfy tremendous need for computing power

Computing Power Demand
Serial computing in embedded systems is hitting the technological limits

Factor 2 - 20

Source: Bosch

Source: ARM

Performance modeling & analysis of classical automotive systems
Performance modeling & analysis of classical automotive systems

Extending the AMALTHEA Hardware Model

- Performance factor \( (PF) \) is not enough for heterogeneous hardware
- Non linear performance effects due to specific acceleration
- \( PF \) is not transparent regarding heterogeneous effects
- Infeasible to compare different ISAs

Goal: Enable our models & simulation for heterogeneous software and hardware

* E.g., IPC (Instructions Per Cycle)
Performance modeling & analysis of classical automotive systems

Basic Flow – Tackle the Gap

Model HW and SW with the AMALTHEA mechanisms including the new HW extension for more flexibility and heterogeneous architectures

Transform performance characteristics into execution times

Simulation tools e.g.

- SYMTA VISION
- INCHRON
- TA Timing Architects

Simulate core execution time and memory accesses (supported by SOTA simulation tools)
Performance modeling & analysis of classical automotive systems

Predictability on High-Performance Platforms

- Shared memory is a big bottleneck in high-end µP based real-time platforms
- Interference effects are more severe by orders of magnitude compared to µC platforms

- Support systems engineering with performance analysis for high-performance platforms

- Goal: predictable real-time behavior

Avg. memory access latencies per word

Source: Roberto Cavicchioli, Nicola Capodieci, Marko Bertogna, Memory interference characterization between CPU cores and integrated GPUs in mixed-criticality platforms. ETFA 2017
Real-time Systems Engineering @ Bosch

Outline

- Performance modeling & analysis of classical automotive systems
  - Motivation ... or the real complexity
  - Amalthea performance model
  - Current usage @ Bosch
  - Upcoming challenges

- Communication centric design in multi-core systems
  - Importance of cause-effect chains
  - Issues with concurrent execution in multi-core systems
  - Communication mechanisms as solution & impact on latencies
  - Experiments

- Timing-aware control design
  - Control and real-time systems engineering – two worlds collide
  - Co-engineering approach
  - Example
Communication Centric Design

Introduction 1/2: Complexity of communication dependencies
Communication Centric Design
Introduction 2/2: Importance of cause-effect chains

Very simple SW structure of an engine control system
Outline

- Performance modeling & analysis of classical automotive systems
  - Motivation ... or the real complexity
  - Amalthea performance model
  - Current usage @ Bosch
  - Upcoming challenges

- Communication centric design in multi-core systems
  - Importance of cause-effect chains
  - Issues with concurrent execution in multi-core systems
  - Communication mechanisms as solution & impact on latencies
  - Experiments

- Timing-aware control design
  - Control and real-time systems engineering – two worlds collide
  - Co-engineering approach
  - Example
Data inconsistency problem

Single Core – Task priorities

- Single core: Legacy code contains implicit assumptions about priorities and thus execution sequences

```
Task B reads x two times (prio > p)
```

```
If (x>0) {
  sqrt(x);
}
```

```
Task A writes x (prio p)
delayed
```

```
x = -1
```

- Multi-core: These assumptions often break the functionalities and require lots of debugging of race conditions

```
Task B reads x two times (prio > p)
```

```
If (x>0) {
  sqrt(x);
}
```

```
Task A writes x (prio p)
```

```
x = -1
```

→ Need for data consistency
Distribution and load dependent end-to-end latencies

- End-to-end behaviour along cause effect chains is non deterministic
- Heavily depends on distribution & scheduling
- 188 possible chains
- Prohibitive for “large scale engineering” where we need to handle thousands of variants
- It’s not about optimization!
- **Determinism needed**: distribution and load independent timing behaviour
Real-time Systems Engineering @ Bosch

Outline

▶ Performance modeling & analysis of classical automotive systems
  ▶ Motivation ... or the real complexity
  ▶ Amalthea performance model
  ▶ Current usage @ Bosch
  ▶ Upcoming challenges

▶ Communication centric design in multi-core systems
  ▶ Importance of cause-effect chains
  ▶ Issues with concurrent execution in multi-core systems
  ▶ Communication mechanisms as solution & impact on latencies
  ▶ Experiments

▶ Timing-aware control design
  ▶ Control and real-time systems engineering – two worlds collide
  ▶ Co-engineering approach
  ▶ Example
Implicit communication to achieve data consistency

- Automotive embedded systems are organized in tasks containing functions that communicate over shared memory (using labels)

- **Explicit communication**
  - No regulations in place, each function directly reads and writes labels
  - Possible races are handled using locks by the developers

- **Implicit communication**
  - Local copies are created for each read label at the beginning of the task
  - All computations work on the local copies
  - The local copies are written back to the shared memory at the end of the task
  - Result: data consistency on task level: all functions operate on the same data set
Logical Execution time (LET) communication

- Mechanism to ensure determinism and data consistency
- Data is communicated at the beginning and end of the period (activation interval)
- Deterministic availability of data irrespective of where the task executes
- Decouples communication and execution
  - Also independent of where data is mapped
- Incurs longer latency
- Simplified event chain timing analysis for complex event chains with multi-rate tasks
Cause-effect chain revisited using LET
Communication Centric Design
Analysis of end-to-end latencies

► For real-world systems **implicit & LET communication** need to be taken into account

► End-to-end latency analyses & simulation approaches are available for **direct communication**
  ► MAST, SymTA/S, pyCPA, Prelude, Timing Architects, Real-time Calculus, ... (name it)

► However, these tools generally ignore communication semantics or focus on schedulability
  analysis considering task deadlines only

► Idea: **transform the performance model** to take into account the different communication
  semantics
Communication Centric Design
Transformation for implicit communication

- Goal: data consistency on task level
  - Different tasks might work on different values at the same time instant
- Trivial transformation: For each Task T
  - Adding one copy-in runnable $C_{pi}$: Create a local copy for all data that is read or modified
  - Adding one copy-out runnable $C_{po}$: Write back local copies
  - Add these copy runnables $C_{pi}$ and $C_{po}$ to the cause-effect chain
Many different possibilities to implement LET communication

Need to perform copy operation between each pair of communicating tasks

Here: copy operations done by high priority copy interrupts

Leads to jitter
Real-time Systems Engineering @ Bosch

Outline

- Performance modeling & analysis of classical automotive systems
  - Motivation ... or the real complexity
  - Amalthea performance model
  - Current usage @ Bosch
  - Upcoming challenges

- Communication centric design in multi-core systems
  - Importance of cause-effect chains
  - Issues with concurrent execution in multi-core systems
  - Communication mechanisms as solution & impact on latencies
  - Experiments

- Timing-aware control design
  - Control and real-time systems engineering – two worlds collide
  - Co-engineering approach
  - Example
Communication Centric Design

HW Model

- Simplified AURIX Architecture

- Memory Access Time
Communication Centric Design

SW Model

- Key data of the model
  - **1250 Runnables** mapped to
  - **21 Tasks & Interrupts** accessing
  - **10,000 Labels** (shared data)
  - Event chains

- Huge amount of data dependencies
  - challenge exact analysis methods

<table>
<thead>
<tr>
<th>Period</th>
<th>1 ms</th>
<th>2 ms</th>
<th>5 ms</th>
<th>10 ms</th>
<th>20 ms</th>
<th>50 ms</th>
<th>100 ms</th>
<th>200 ms</th>
<th>1000 ms</th>
<th>sync</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 ms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>I</td>
<td>I</td>
<td>I</td>
<td></td>
<td></td>
<td>I</td>
</tr>
<tr>
<td>2 ms</td>
<td></td>
<td></td>
<td>I</td>
<td>I</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>I</td>
</tr>
<tr>
<td>5 ms</td>
<td></td>
<td></td>
<td>I</td>
<td>IV</td>
<td>IV</td>
<td>IV</td>
<td>II</td>
<td>II</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>10 ms</td>
<td>II</td>
<td>II</td>
<td>II</td>
<td>VI</td>
<td>IV</td>
<td>IV</td>
<td>II</td>
<td>II</td>
<td>I</td>
<td></td>
</tr>
<tr>
<td>20 ms</td>
<td></td>
<td>I</td>
<td>I</td>
<td>IV</td>
<td>VI</td>
<td>II</td>
<td>IV</td>
<td>I</td>
<td>II</td>
<td>I</td>
</tr>
<tr>
<td>50 ms</td>
<td></td>
<td></td>
<td>II</td>
<td>II</td>
<td>III</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100 ms</td>
<td>I</td>
<td>I</td>
<td>V</td>
<td>IV</td>
<td>II</td>
<td>VI</td>
<td>II</td>
<td>III</td>
<td>IV</td>
<td></td>
</tr>
<tr>
<td>200 ms</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000 ms</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td></td>
<td>I</td>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Angle-sync</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>IV</td>
<td>IV</td>
<td>I</td>
<td>III</td>
<td>I</td>
<td>I</td>
<td>V</td>
</tr>
</tbody>
</table>

© Robert Bosch GmbH 2018. All rights reserved, also regarding any disposal, exploitation, reproduction, editing, distribution, as well as in the event of applications for industrial property rights.
Communication Centric Design

Experiment setup

- Analysis of 2 cause-effect chains
- Calculation of end-to-end latency distribution
  - Direct communication
  - Implicit communication
  - LET communication
- Comparison of overhead for copy operations
- Use of scheduling simulation engine of SymTA/S
  - Worst-case end-to-end latency of limited interest
Communication Centric Design

End-to-end latency EC1

- Reaction semantic: 10ms → 10ms → 10ms

---

Direct communication

Implicit communication

LET communication

Latency ~ x 2

~ same Latency
Communication Centric Design
End-to-end latency EC2

- Reaction semantic: 100ms → 10ms → 2ms

Direct communication
Implicit communication
LET communication

Latency ~ x5
Latency ~ x3
Observation 1: implicit communication reduces data access costs

Observation 2: LET communication further reduces data access costs since less copy operations are performed

Room for optimizing the data placement to reduce data access costs
Conclusion

- Large scale engineering requires mechanisms that simplify timing analysis
  - Simplicity, maintainability, composability key principles of robust design

- Benefits offered by Implicit and LET communication in terms of determinism and data consistency outweigh the increase in latency

- Communication semantics need to be accounted for in the timing analysis
  - Impact each stage: Task Formation, Task Mapping, End-to-end Latencies

- Existing academic approaches handling co-scheduling of computation/communication should be extended towards meeting the goals of determinism and data consistency
Real-time Systems Engineering @ Bosch

Outline

- Performance modeling & analysis of classical automotive systems
  - Motivation ... or the real complexity
  - Amalthea performance model
  - Current usage @ Bosch
  - Upcoming challenges

- Communication centric design in multi-core systems
  - Importance of cause-effect chains
  - Issues with concurrent execution in multi-core systems
  - Communication mechanisms as solution & impact on latencies
  - Experiments

- Timing-aware control design
  - Control and real-time systems engineering – two worlds collide
  - Co-engineering approach
  - Example
Timing-aware Control Design
Two Disciplines – Two Worlds

Control Engineer

Software Engineer

Too little communication and too little understanding
**Timing-aware Control Design**

System as seen by the software engineer

ECU

- **Tasks**
- **Scheduling**
- **Cores, Memories**

Deadline = Period WCET, WCRT

$$R_i = C_i + \sum_{j \in D_p(i)} C_j \left[ \frac{R_i}{T_j} \right] \leq D_i = T_i$$

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n \cdot \left( \sqrt{n} - 1 \right) \ln 2 \approx 69.3\%$$

**"Real-time performance"**

- **High**
  - P1 | P2
  - P1 | P2
  - P1 | P2

- **Middle**
  - P3
  - P3

- **Low**
  - P4
  - P5
  - P6

- **P5 Timing Profile**

  - Sampling Jitter
  - Response Time Jitter
Timing-aware Control Design
System as seen by the control engineer

Sophisticated Control Algorithm

Periodic Sampling, No Jitter

No/Constant Control Delay Calculation takes 0/constant time

No Output Jitter, “Freshest” value always available

A/D Converter

D/A Converter

Plant
\dot{x} = Ax + Bu
y = C^T x

Read Data

Write Data

u_k

y_k

y_k

u_k

u_k

y(t)

u(t)

"Control performance"
Timing-aware Control Design

Consequences

- “That guy has unclear, not implementable requirements
  -> I’ll optimize resources”
- Guaranteed period, task-wide data consistency,
  last-is-best (LIB) communication
- Load-dependent behavior & jitter

- “My algorithm performs always better in simulation than in the prototype vehicle →
  I’ll test my algorithm only in the vehicle and make it more robust”
- Long design iterations, late rework, ...
  (adds burden to heavy calibration tasks)
- Wasted HW resources
Real-time Systems Engineering @ Bosch

Outline

- Performance modeling & analysis of classical automotive systems
  - Motivation ... or the real complexity
  - Amalthea performance model
  - Current usage @ Bosch
  - Upcoming challenges

- Communication centric design in multi-core systems
  - Importance of cause-effect chains
  - Issues with concurrent execution in multi-core systems
  - Communication mechanisms as solution & impact on latencies
  - Experiments

- Timing-aware control design
  - Control and real-time systems engineering – two worlds collide
  - Co-engineering approach
  - Example
Timing-aware Control Design

Solution: Early Simulation Feedback

Control Engineer (Simulink)

Initial Control Design

Virtual Integration with SW Platform

Virtual Integration with SW Platform

Software Engineering (TraceAnalyzer, SymTA/S)

Runnables, periods, constraints

Schedule Definition & Timing Model

Schedule Definition & Timing Model

If needed: iterate long before code is actually implemented

Annotation of timing to simulation model

Timing Analysis

Evaluation of Control under actual resource constraints

Timing profile for runnables

© Robert Bosch GmbH 2018. All rights reserved, also regarding any disposal, exploitation, reproduction, editing, distribution, as well as in the event of applications for industrial property rights.
Timing-aware Control Design
Solution Details: Simulink Toolbox

Generic instrumentation of model with timing blocks

Blocks are configured with timing profiles (i.e. lists of time stamps)
Timing-aware Control Design
Solution Details: Timing Profile Generation

Generates timing traces & time stamps for instrumented model

Real-time Systems Engineering @ Bosch

Outline

- Performance modeling & analysis of classical automotive systems
  - Motivation ... or the real complexity
  - Amalthea performance model
  - Current usage @ Bosch
  - Upcoming challenges

- Communication centric design in multi-core systems
  - Importance of cause-effect chains
  - Issues with concurrent execution in multi-core systems
  - Communication mechanisms as solution & impact on latencies
  - Experiments

- Timing-aware control design
  - Control and real-time systems engineering – two worlds collide
  - Co-engineering approach
  - Example
Timing-aware Control Design

Proof of Concept: Car Road Damping (Active Suspension)

- Is tool and concept applicable for complex systems?
- Case study: Damp car body acceleration with body control
- Complex system: 12 runnables, 7 accelerometers, 4 force actuators
- Exercise basic workflow with tool
- Tested different platform timing configurations
- Published SAE 2015 World Congress

Road unevenness in m

Source of Model:
Timing-aware Control Design

Simulink model structure

- **Disturbance (Street)**
- **Diff. Eq.: Moving parts of car + actuators**
- **Set point generation**
- **Local Controllers**
- **Observation of system's quantities**
- **Sensors**

Sensors

Differential Equations

Diff. Eq.: Moving parts of car + actuators

Set point generation

Local Controllers

Observation of system's quantities

Disturbance (Street)
Timing-aware Control Design

Different Solutions Alternatives

Road unevenness in m

Results for one set of control parameters (Shown: z-Acceleration in m/s²):

- Ideal timing
- Timing Single Core
- Timing Distributed ECUs

• Approach is able to handle complex systems
• Results are plausible and show expected differences in response
Timing-aware Control Design
Co-Engineering can start ...

▶ I can see the effects of real-world timing on my control performance already in functional simulation
▶ But now I want to redistribute tasks to balance load in multi-core...
THANK YOU

...AND HARALD MACKAMUL, JÖRG TESSMER, FALK WURST, TOBIAS BEICHTER, SYED AOUN RAZA, DIRK ZIEGENBEIN, JENS GLADIGAU, DAKSHINA DASARI, MICHAEL PRESSLER