Evaluation of architecture variants for hard real-time systems

Timing as part of system architecture
- **Hard real time** requirements
  
  The system response time to a certain event **always** has to be **within a certain limit**.

- **Soft real time** requirements
  
  The system response time to a certain event should usually be **within a certain limit**. It is not problematic if this limit is violated occasionally.

A timing fault in a real-time system occurs when a task or interrupt misses its deadline.
Where do temporal requirements come from?

• Physics of the system (braking distance, ...)

• Resulting form system architecture (redundancies, partitioning, selection of technologies, ...)

• Resulting from hardware architecture (selection of technologies, ...)

• Resulting from software architecture (number of tasks, ...)

• Resulting from mechanical architecture (geometry, ...)

The vehicle drives along the road with increased speed. Suddenly a pedestrian steps onto the road. A collision is inevitable. The FABSY-Unit detects the pedestrian, realizes that a collision is inevitable and activates the airbag, which preserves the pedestrian from severe damage.
How to meet timing requirements

Strategies
- Scheduling (preemptive, non-preemptive)
- OS (priority based, preemptive)
- Watchdog based-mechanisms (e.g. Control flow analysis)
- Scheduling Bus-Protocol level
Bottom-up approach:

- Software units are assembled to construct software components
- Components realize tasks (i.e., work units) in applications
- Tasks are scheduled (i.e., planned)
**Bottom-up approach** is pursued in lots of projects, but problematic:

Scheduling is **not considered in system design** and is the **final step** during system integration.

The adherence to timing constraints is strongly dependent on provided components:

- Units/components: contain implementation details influencing worst-case execution times
- Application: Mapping of components to tasks and jobs (e.g., runnables) to OS threads restrict scheduling possibilities

Distributed development and buying software components aggravate the problems imposed by bottom-up approach.
• Subsequent changes in software units, components and applications are very expensive
• Correction influences execution-time behavior
  • Components' worst-case execution times change
  • Changes in thread mapping aggravate the problem
• Rework may be necessary if a component needs too much CPU time and scheduling fails
  • Inefficient coding
  • Inapt application structure
Top-Down Specification

- OEM has knowledge about the entire system
- Applications are provided with execution budgets
- Units and components have to use the budgets wisely
- Framework of temporal constraints defines scope of actions

Q: What is better, bottom-up or top-down?
A: Both!
Architecture and Real-Time Systems

- Functional architecture is developed with the requirements
- Enables to evaluate the schedulability
- Shows timing requirements and infrastructure
- It identifies and explores alternative implementation strategies consistent with the requirements and risks.
A failure causes the service to deviate from its specified behavior (e.g., faulty output values). The failure can be caused by an error, that is a discrepancy in the system’s internal state and an error (e.g., a deadline is not met) is caused by a faulty assumption (e.g., tasks are not terminated optimally).

System can fail

- System architecture is designed faulty
- Timing requirements are not derived properly

Hardware can fail

- Caused by random faults (see HW-Metrics)
- Caused by systematic faults: bugs
- Caused by system specification/design faults
- etc.

Software can fail

- Caused by systematic failures in software
- Caused by hardware failures
- Caused by system specification/architecture faults
- etc.
FABSY – Preliminary architectural assumption
FABSY – Functional Architecture

ASIL B: Airbag does not inflate in case of crash with pedestrian

ASIL D: Undesired activation of Airbag
Fighting timing faults

Finding **systematic timing faults** at its root and not just dealing with the effects.

Timing faults origin in:
- System design
- SW-Architecture
- HW-Architecture
The functionality has to be allocated to control units

Transmission paths depending on technology (LIN, CAN, Flexray, …) and distance cause further delay and have to be considered → architecture of the onboard electrical system (Bordnetzarchitektur)
FABSY – Technical Architecture
FABSY – Evaluation of Architecture
- Ereignisgesteuerte Simulation mit Zeitbasis
- Scheduler simuliert das ausführen von Tasks, Runables, …
- Tasks werden aktiviert, unterbrochen, wiederaufgenommen
- Anzahl von Cores, Taktzeit, busspezifischen Übertragungszeiten können berücksichtigt werden
- Asynchronität von zb.: Flexray und μC kann miteinbezogen werden

- Datenfluss in Wirkketten wird grafisch dargestellt
The result

- Timing budgets are defined and assigned
- Architecture is designed
- Architecture is evaluated based on assigned budgets

- Established methods still need to be carried out (code checking, watchdogs, integration tests, ..) but will cause less effort
In case of questions

Dr.-Ing. Isabella Stilkerich  
Software-Engineering Specialist  
Schaeffler Technologies AG & Co. KG

isabella.stilkerich@schaefler.com

Frank Sigiliano Pinecker  
Process Consultant  
Methodpark Consulting GmbH

frank.pinecker@methodpark.com
OUR PHILOSOPHY IS SIMPE AND SHORT

ENABLER FOR INNOVATIVE SOFTWARE & SYSTEMS ENGINEERING.