Multicore and Functional Safety; Solutions for new challenges

Dheeraj Sharma
June 2016
Agenda

• The Trend towards Multicore
• What is Functional Safety?
• Multicore in AUTOSAR
• Challenges presented by Multicore
• Summary
The Trend towards Multicore

Next generation of car infrastructure

- Domain Controllers will be huge multi-MCU and Multicore systems
  - ECU independent function
  - Connected to Actuator / Sensor ECUs
  - Reloadable functions
  - Connected via Automotive Ethernet

- Domain ECUs will be “small” Multicore or single core systems
  - Hard real time
  - I/O handling
  - Safety functions
The Trend towards Multicore

Evolution of Desktop-CPU Performance

Source: Data from c’t 7/2014, p. 127

© Elektrobit (EB), 2016
The Trend towards Multicore

Typical HW architecture (Infineon AURIX TC27x)

- **3 cores heterogeneous (2 lockstep)**
- **Cross-bar (i.e. bus per slave)**
  - Read-modify-write support
- **Split I/D scratchpad / cache (non-coherent) per core**
- **Shared ECC RAM (single-ported)**

Source: Datasheet AURIX TC27xT
Agenda

• The Trend towards Multicore
• What is Functional Safety?
• Multicore in AUTOSAR
• Challenges presented by Multicore
• Summary
The Importance of Functional Safety

From IEC 62304 Standard:

“There is no known method to guarantee 100 % safety for any kind of software. There are **three major principles** which promote safety for medical device software:

1. **Risk management**
2. **Quality management**
3. **Software engineering**”
The Importance of Functional Safety

Risk Control as part of risk management

• The **goal** of risk control is to reduce the risk to an “acceptable level”.

• **Methods** of risk control (in order of preference!):

  1. **Eliminate the risk (by design)**
     Easiest by omitting the functionality which creates the risk.

  2. **Implement protective measures**
     E.g., fault detection, prevention of fault propagation (**freedom from interference**), ...

  3. **Provide adequate information**
     E.g., warnings, guidance in the safety manual, ...
ISO 26262: Freedom from Interference

Memory
- Unintended writing to memory of another partition
- Register/Configuration corruption due to unintended writing to processor registers

CPU Time
- Blocking of partitions
- Wrong allocation of processor execution time

Communication
- Loss of communication
- Insertions of messages
Safety RTE for protected communication between Memory Partitions

Safety OS:
- Data Protection
- Stack Protection
- Context Protection
- OS Protection
- Hardware Error management

Safety E2E Protection for safe communication to other ECUs

Safety TimE Protection for alive supervision, deadline and control flow monitoring

Partitions
Agenda

• The Trend towards Multicore
• What is Functional Safety?
• Multicore in AUTOSAR
• Challenges presented by Multicore
• Summary
Multicore in AUTOSAR 4.0

CORE 0
- Application 0
  - System Services
  - Onboard Device Abstraction
  - Microcontroller Drivers
- Runtime Environment
- Communication Services
- Communication Hardware Abstraction
- Communication Drivers

CORE 1
- Application 1
  - System Services
  - Onboard Device Abstraction
  - Microcontroller Drivers
- Runtime Environment
- Communication Services
- Communication Hardware Abstraction
- Communication Drivers

Inter-core communication for each call
Concurrent access to comm. service
Potentially synchronous calls
Basic Software Distribution in AUTOSAR >4.0

AUTOSAR and Multicore

© Elektrobit (EB) 2016
Agenda

• The Trend towards Multicore
• What is Functional Safety?
• Multicore in AUTOSAR
• Challenges presented by Multicore
• Summary
Cohesive Architecture

• **Partitioning and core allocation**
  – Functional Grouping
  – Related functions on same core
  – Freedom from Interference

• **Safety Analysis**
  – Only feasible when working on small units
Architectural considerations

**Challenges**

- Partitions are mapped to cores
  - Groups of tasks etc. are distributed
- Cross-core communication takes time
  - Beware: Transparent communication in AUTOSAR
- Locks are expensive
  - Locking only possible via OS
  - Spinlocks can block entire CPU
- True parallel execution
  - Race conditions
  - Task priorities work only on one core

**Countermeasures**

- Keep partitions cohesive
- Locate related and communication-intensive applications on same core or partition
- Where possible, use lock-free algorithms
- Reduce cross-core communication to a minimum
Single-core scheduling with shared resources

- AUTOSAR supports preemptive and non-preemptive tasks
- **Shared resources to protect access to global data**
- AUTOSAR uses Priority Ceiling Protocol (PCP)
- Upon occupying a resource, the task receives the priority of the highest task that can access the resource

![Diagram showing scheduling with shared resources]

Task 1 receives priority of Task 3
Shared resources on Multicore

- **Priority Ceiling Protocol** (PCP) does **not work** for Multicore
  - Separate schedulers
  - No common notion of priority

- AUTOSAR specifies **spinlocks for Multicore synchronization**
  - Busy waiting on memory location via atomic instruction (e.g. TAS, CAS, LL/SC)

- **Multicore Priority Ceiling Protocol** (MPCP) assigns priority higher than highest core-local priority to each inter-core resource
  - Can be implemented in AUTOSAR (spinlock and interrupt lock)
  - Used to prevent deadlock
    - *e.g. when task holding a spinlock is preempted by task that also tries to obtain the spinlock*
Multicore scheduling with shared resources

- Spinlocks waste processing time
- Spinlocks create high traffic on the processors interconnect
- Inter-core locking creates scheduling dependencies

Task 2 could have completed while Task 3 was waiting for the spinlock
Multicore scheduling in AUTOSAR

Global Scheduling

Semi-partitioned Scheduling

Partitioned Scheduling

AUTOSAR
Multicore scheduling in AUTOSAR

• Logical separate **OS instance per core**
  – OS resources are effective only per core
  – Priorities defined per core
  – (Memory protection configured per core)

• **Static mapping** of tasks to cores (i.e. partitioned scheduling)
  – No fancy features like dynamic load-balancing

• **Cross-core interrupts** possible
• **Cross-core events** possible
• **Inter-core locking** via spinlocks; but spinlocks can be expensive.
Partitioned Multicore scheduling

- Cross-core task activations/interrupts/callbacks/sync. C/S calls create scheduling dependencies (remember independent schedulers/timebases)
- Cross-core scheduling dependencies can increase context switches
- Analysis is complex

Core 2
Task 4
Task 3

Core 0
Task 2
Task 1

If Task 1 executed a bit longer, no additional context switches would occur
Full-blown scheduling complexity

• What are the scheduling dependencies here?

“In my thesis I have shown how incredibly complex the analysis is and quite frankly you simply should not build systems like this.”

Dr. Mircea Negrean

(Title of thesis: “Performance Analysis for Multi-Mode Multicore Systems with Shared Resources – Principles and Application to AUTOSAR”)

© Elektrobit (EB) 2016
Agenda

• The Trend towards Multicore
• What is Functional Safety?
• Multicore in AUTOSAR
• Challenges presented by Multicore
• **Summary**
## Corollaries

- **Reducing inter-core locks**
  - Increases potential parallelism
  - Decreases complexity

- **Reducing core-scope of global data**
  - Reduces need for inter-core locks
  - Increases potential use of caches

- **Separating global data**
  - Reduces need for inter-core locks
  - Helps establishing memory protection

- **Reducing consistency**
  - Increases potential parallelism

## Questions to ask

- Where is data accessed?
- What kind of lock is required?
- Is atomic access possible?

- Where is data accessed?
- Are separate per-core instances possible?

- Do members of a struct/elements of an array have different access requirements?

- Is the order of calls relevant?
- May data be inconsistent in intermediate states?
Contact us!

www.elektrobit.com
dheeraj.sharma@elektrobit.com