Making Full Use of Multi-Core ECUs with AUTOSAR Basic Software Distribution

Webinar
Agenda

- **Motivation for Multi-Core**
  - AUTOSAR Standard: SWC-Split
  - MICROSAR Extension: BSW-Split
  - BSW-Split: Technical Challenges
  - BSW-Split: Implementation
  - Summary
Motivation for Multi-Core

Motivation I

- There are many motivations for multi-core systems. E.g.
  
  - **Speedup**: A computational problem can be calculated in shorter time
    > Speedup is realized by parallelization of the problem
    > Some problems are easy to parallelize, some are not

- **Workload distribution**: Assignment of independent (sub-) problems to different cores
  > Each computational problem will require the same execution time
  > In total, several problems can be calculated in parallel
Motivation for Multi-Core

Motivation II

- There are many motivations for multi-core systems

- Improved system responsiveness: Distribution of time critical processes to different cores
  - Cores can take over special roles such as interrupt handler or background job worker
  - Low-priority housekeeping does not interfere with high priority job processing

- Others...

- Different motivations lead to different software architectures
- Motivations often overlap

- There is no “one fits all” architecture
Agenda

Motivation for Multi-Core

- **AUTOSAR Standard: SWC-Split**
- MICROSAR Extension: BSW-Split
- BSW-Split: Technical Challenges
- BSW-Split: Implementation
- Summary
AUTOSAR Multi-Core Paradigm

Enable the application SWC to realize project specific multi-core use cases

- The application SWC can use any core of an AUTOSAR multi-core system
  - Great degree of freedom for application deployment

- The AUTOSAR OS manages the cores and provides
  - Inter-core communication
  - Inter-core scheduling control
  - Data consistency mechanisms (Spinlocks, IOC)

- The AUTOSAR base software provides its services on all cores
  - Diagnostic
  - (Bus) Communication
  - Watchdog
  - ECU management
  - ...

![AUTOSAR Multi-Core Paradigm Diagram](image)
AUTOSAR BSW Stack

- Only limited degree of freedom for BSW deployment
- AUTOSAR describes three possibilities:
  - Mapping of:
    - I/O drivers
    - I/O hardware abstraction
    - Complex drivers (CDD)
  - Mapping of
    - Mode- and state management
  - Provision of communication service satellites
- BSW services which realize the master-satellite pattern
  - The master resides in the BSW partition
  - The satellites are mapped to different partitions
  - Functional scope of the satellites is implementation specific
Agenda

Motivation for Multi-Core
AUTOSAR Standard: SWC-Split

- **MICROSAR Extension: BSW-Split**
  - BSW-Split: Technical Challenges
  - BSW-Split: Implementation
  - Summary
In modern ECUs CPU load caused by the Base Software can be significant.

Typical drivers are:
- High communication load (e.g. Ethernet, CanFD)
- Advanced communication use cases (e.g. Service oriented, secured)
- BSW service utilization due to larger applications

AUTOSAR classic architecture is fine grained:
- Many different modules
- Relatively small modules

Many contributors to overall BSW CPU load.
Causes

- In BSW modules, there are up to three different causes for CPU load

- Cyclic main functions
  - Executed in task context
  - May call neighboring modules

- Interrupt service routines
  - Executed in interrupt context
  - Executed cyclic or sporadic
  - May call neighboring modules
In BSW modules, there are up to three different causes for CPU load:

- Interface implementation
  - Executed in task or interrupt context
  - May call neighboring modules

Relocation of a BSW module ideally addresses all of its entities:

- To relocate as much CPU load as possible
- To avoid concurrent access to the module’s data
- To avoid inter-module calls of second degree
MICROSAR Extension: BSW-Split

Consequences

- AUTOSAR SWC-Split is not always sufficient
- BSW-Split is sensible for specific BSW modules and specific use cases

- MICROSAR provides BSW-Split as standard feature and extension to AUTOSAR
- BSW-Split allows relocation of **non-multi-core** BSW modules
- Relocation is achieved by using tailored, multi-core capable BSW modules (Dem, EcuM, PduR...)
Agenda

Motivation for Multi-Core
AUTOSAR Standard: SWC-Split
MICROSAR Extension: BSW-Split

- BSW-Split: Technical Challenges
  BSW-Split: Implementation
  Summary
Naïve Approach

- **Goal**: Distribution of BSW load
- **Consequence**: Relocation of modules with significant contribution to the CPU load (utilization)
- CPU utilization ($u$) is calculated as the sum of execution times ($\Delta t_{mod}$) of a module during a specific timespan ($\Delta t$)

\[
u = \frac{\sum_{k=0}^{n} \Delta t_{mod_k}}{\Delta t}
\]

- Influencing factors are
  > Individual execution times
  > Invocation frequency ($\frac{n}{\Delta t}$)
- **Deduction**: Relocate modules which have a
  - high execution time or
  - rather small execution time but run frequently

- That simple?
Challenge 1: Context Change

- **Starting situation:** Relocation of complete modules to maximize effect on CPU load
  - Interfaces
  - Main functions
  - Service Routines
  - Data

- **Challenge:** Context change to different core on invocation of a API
Challenge 1: Context Change - Realizations

**Solution:** There are two fundamentally different approaches

- **Event driven invocation**
  - The caller passes an event to the remote core which activates e.g. a task
  - The task executes a worker, which calls the interface function

- **Data driven invocation**
  - The caller serializes the interface invocation request in a queue
  - A worker on the remote core reads from the queue, deserializes the request and calls the interface
Challenge 1: Context Change - Consequences

- Both variants add execution time overhead
  - On the caller-core (notification / serialization)
  - On the called core (worker execution, task scheduling, deserialization)

- **Deduction (revisited):** Relocate modules which have
  - high execution time or
  - rather small execution time but run frequently **and**
  - a positive net effect of relocated execution time vs. overhead

- In other words: There is not much sense in relocating a small execution time which causes overhead that is larger
Challenge 2: Data Lifetime

- **Starting situation:** Relocated BSW module is executed in a different core context

- **Challenge:** Data lifetime
  - Some data is only available during specific contexts. E.g. Buffers in µC peripherals
  - Example CAN reception:
    1. Data resides in mailbox
    2. Pointer is passed to the interface
    3. Interface (its proxy) triggers upper layer on a different core
    4. The pointer is passed to the UL on the different core
    5. Interface returns
    6. Driver releases buffer!
    7. UL is being executed

- **Solution:** Data must be buffered by proxy
  - Consumes memory
  - Consumes execution time
  - Requires efficient implementation
BSW-Split: Technical Challenges

Challenge 3: Synchronous vs. Asynchronous

- **Starting situation**: Context changes are asynchronous to achieve best performance
  - Realization of a Fire-and-Forget behavior

- **Challenges**
  - Return values/out parameters are required immediately
  - Sequences might get broken due to parallel processing

- **Solutions**
  - Provision of core-local multi-core capable BSW-modules which allow invocation without context switch
  - Implementation of cross-core context-switch with synchronous behavior
  - Implementation of “special solutions” (E.g. Assumptions on return values, limitations...)
Motivation for Multi-Core
AUTOSAR Standard: SWC-Split
MICROSAR Extension: BSW-Split
BSW-Split: Technical Challenges

**BSW-Split: Implementation**

Summary
BSW-Split: Implementation

Intro

- BSW-Split requires
  - Thorough analysis of the considered modules
  - Deep knowledge of the BSW stack and its internal interactions and sequences
  - Proxy / Worker functionality for context switch

- MICROSAR BSW-Split provides
  - A predefined set of modules for relocation
  - Additional functionality to support the relocation performance efficiently
  - Supporting documentation for the distribution

- MICROSAR BSW-Split achieves
  - CPU load distribution
  - Improved responsiveness by relocating long-running interrupt service routines
Overview

- Multi-core BSW-Split allows relocation of communication bus cluster
  - Relocation of individual channels is not supported
  - Relocation affects all bus relevant BSW modules

- Currently supported
  - Ethernet

- Rationale
  - Execution times of whole modules comparably high
    - Potential to distribute considerable load
    - Good ratio between overhead and distributed load
  - Favorable interfaces to other BSW modules
  - Favorable communication behavior of Ethernet
BSW-Split: Implementation

Architecture

- Relocation of complete bus clusters
- Example for Ethernet:
  - Driver, Interface
  - Socket Adaptor (SoAd)
  - Service Discovery (Sd)
  - Network and state management (UdpNM, EthSM)
  - Switch driver
  - ...

- PduR as router between cores for
  - Interface PDUs
  - Transport Protocol PDUs

- Interfaces to neighboring BSW modules either
  - Unchanged for multi-core capable BSW modules
  - Via proxy / worker pattern
Architecture Benefits

- Efficient handling of PDU interfaces between SoAd and PduR
  - PDU interfaces are data-oriented
  - Data can be transferred efficiently

- Minimized impact for non-PDU interfaces
  - Direct API invocation of neighboring modules supported by multi-core BSW for
    > Dem, Det, BswM (Upcoming), ComM (Upcoming), XCP (Upcoming)
  - Usage of synchronous proxy / worker pattern as user implemented wrappers for
    > NvM, User callouts

- No impact on relocated modules
  - Tightly coupled modules remain on one core
Every core handles one or more communication interfaces

PduR was enhanced to route the following objects between cores
- Transmit-requests (PDUs)
- Rx-Indications (PDUs)
- Tx-Confirmations

Objects are only exchanged between cores if target bus is mapped to different core
- Minimizes performance penalty for core-local communication

All inter core communication is handled deferred in main function context
- PduR provides cyclic main functions on cores with bus clusters
Implementation - PduR Data Exchange

- All elements to be exchanged between cores are buffered in PduR
  - Rationale: Data availability in upper/lower layer module is not guaranteed during deferred processing
  - No additional buffer required if routing path already requires buffering (e.g. buffered gateway routings)

- PduR distinguishes between two fundamental types of communication
  - Type I:
    > Unbuffered interface PDU API forwarding
    > Unbuffered interface PDU API gateway routing
  - Type II:
    > TP PDUs
    > Buffered PDUs
    > Trigger transmit PDUs
BSW-Split: Implementation

Implementation – PduR Communication Types

- **Type I**
  - Usage of unidirectional FiFio queues in PduR
  - Queue-size configurable
  - One queue per core-to-core communication path and direction
  - Spinlock-free implementation

- **Type II**
  - Usage of the standard PduR buffers
  - Data consistency ensured with Spinlocks
    - Custom locks for minimized API execution time
    - Individual locks per routing path to avoid spinning
Configuration I

- Tool-assisted configuration
  - Manual configuration for
    > General settings
    > Architecture defining parameters
  - Automated configuration for
    > Routing path configuration

- New configuration options in PduR (manually set):
  - Deferred Reception/Transmission
    > Enables main function based forwarding and routing
  - Main Function Period
    > Influences delay of cross-core routing paths
  - Multicore Queue Size
    > Size of the queues for Type I communication
  - Support Multicore
    > Enables/disables multi-core support in general
Configuration II

- New configuration options in PduR:
  - BSW module Application assignment
    - Determines between which modules the PduR must realize cross-core communication
  - Lock Ref (set automatically)
    - Exclusive Area implementation of the routing path
  - Multicore Routing Path (set automatically)
    - Determines if the routing path is cross-core
Agenda

Motivation for Multi-Core
AUTOSAR Standard: SWC-Split
MICROSAR Extension: BSW-Split
BSW-Split: Technical Challenges
BSW-Split: Implementation

Summary
MICROSAR BSW-Split provides preselected distribution of BSW modules
- No replacement for application SWC distribution
- Effect on CPU-load highly project dependent
  - Type of communication (Type I, Type II)
  - Amount of communication (Resulting CPU load in communication stack)

Motivation
- Workload distribution
- Improved system responsiveness
- Speedup

Provides efficient implementation for the performance critical interfaces

Chance for “communication-heavy” ECUs which cannot fit BSW onto one core
- Especially when Ethernet has a major contribution to the BSW load
For more information about Vector and our products please visit

www.vector.com

Author:
Zeeb, Alexander
Vector Germany