Electronic Design

DR. THOMAS GALLA, Chief Expert Automotive Networks,
Elektrobit, https://www.elektrobit.com/

Unleash Multicore-Processor Performance in Automotive Architectures

Software must be parallelized and modified to benefit from new approaches to enhance hardware performance in today's automotive designs. AUTOSAR's layered software architecture leverages MCUs to meet the latest demands.

For many decades, software developers benefitted from being able to use the same software code while working with increasingly powerful hardware. As hardware manufacturers regularly improved the performance of their semiconductors by enhancing transistor densities and clock speeds, software development enjoyed a "free ride"—they were able to readily develop on these new devices without having to change software architectures.

However, processing power has hit a wall due to the limit in increasing clock speeds. As a result, chip manufacturers have been turning to dramatically new approaches to achieve further performance gains. First it was hyper-threading and homogeneous architectures, and then heterogeneous multicore architectures. To benefit from these hardware changes, existing software had to be parallelized and modified to deal with the heterogeneity.

Modern microcontrollers (MCUs) and systems-on-a-chip for automotive, such as Infineon's AURIX 2G or NVIDIA's Drive Xavier, point to the trend toward homogeneous, and even heterogeneous, multicore hardware architectures. To benefit from these advances in hardware, changes in automotive software are required as well.

In this article, I will discuss how standardized software architectures, specifically AUTOSAR's layered software architecture, are being updated with today's powerful MCUs to enable dramatically improved performance.

Optimizing AUTOSAR for Multicore Architectures

A critical enhancement to the AUTOSAR software architecture has been the distribution of the AUTOSAR communication stack over different cores, which is mandatory for realizing the performance benefits of multicore architectures.

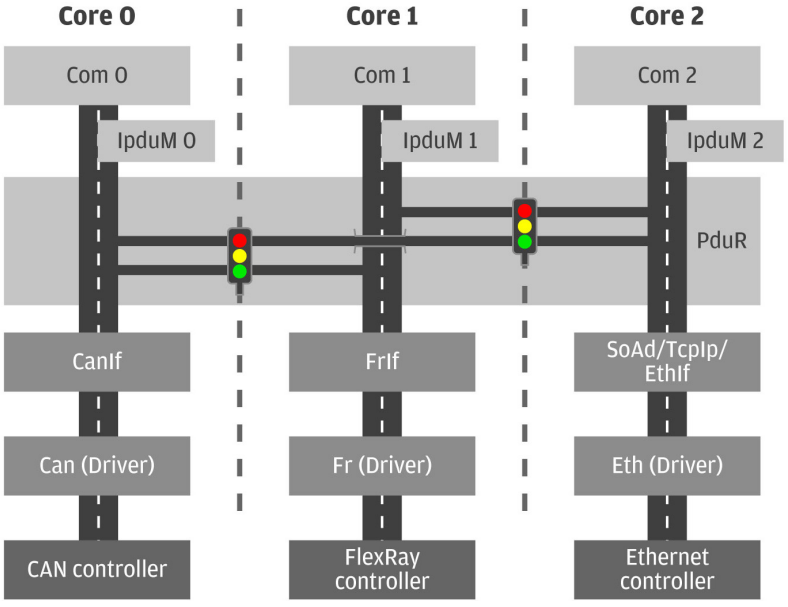
As background, in AUTOSAR 4.0.1, support for multicore MCUs was first introduced. In this update, AUTOSAR provided the means to allocate application software components (SWCs) to dedicated cores and facilitated the cross-core communication between those SWCs via the runtime environment (RTE). AUTOSAR basic software (BSW), however, was still allocated to a single core.

In AUTOSAR 4.2.1, the AUTOSAR basic software was divided into so-called functional clusters that could be allocated to different cores using the BSW schedule manager (SchM) for inter-core communication. Since the communication stack as a whole is such a functional cluster, distribution of the communication stack over multiple cores wasn't supported. And, although AUTOSAR 4.4 introduced the possibility to distribute the BSW modules of the lowest layer (the microcontroller abstraction layer), the remainder of the AUTOSAR communication stack still had to be placed onto a single core.

At this point in the evolution of AUTOSAR, it became obvious that a monolithic communication stack allocated to a single core would eventually become the performance bottleneck. That's because the sequential part of the software would continue to impose a theoretical limit on the speeds achieved by a multicore MCU. Hence, it led to the fresh approach of distributing the communication stack over the different cores, which is a necessity for reaping the performance benefit of multiple cores.

When working on communication stack software distribution, it's important to consider the following to make efficient use of the multicore resources:

- *Inter-core communication and synchronization should be reduced as much as possible*, since they typically involve inter-core interrupts that in turn lead to changes in the MCU's



This multicore communication stack architecture is by means of core allocation of the AUTOSAR stack.

operation mode (transition from user to supervisory mode), pipeline stalls, and cache misses.

- If inter-core calls are required and can't be avoided, *asynchronous calls should be favored over synchronous ones*. The latter block the caller until the callee is finished, thereby reducing the degree of parallelism and thus the potential speedup. Unfortunately, this isn't always possible since, for legacy reasons, AUTOSAR's communication stack makes heavy use of synchronous APIs and changing that would be a major backwards-incompatible redesign.
- In addition, *inter-core mutual exclusion by means of locks should be avoided* if possible, because this blocks all other involved cores while one core resides in the exclusive area. Since typical inter-core mutual exclusion primitives like spinlocks involve busy waiting, this also wastes CPU cycles on the blocked cores.
- *Another crucial consideration is the proper placement of code and data required by the non-uniform memory architecture used by most multicore MCUs.* Memory is divided into core-local memory (caches, flash, and RAM) dedicated to a single core, which can be accessed quickly and conflict-free by that core, and global memory (flash and RAM), which is shared among the different cores and where access to this memory is substantially slower and subject to conflicts. In such a non-uniform memory architecture, proper placement of code and data is critical. Frequently accessed code and data needs to be placed as close to the accessing core as possible. Using the static AUTOSAR memory-mapping mechanisms, such placement should be performed based on access statistics derived under realistic load scenarios.

Stack Distribution Strategy

With these considerations in mind, we can develop the general distribution strategy for the AUTOSAR communication stack. We split the communication stack into sub-stacks based on the particular network type (i.e., CAN, LIN, FlexRay, and Ethernet) and allow each of these sub-stacks to be allocated to a dedicated core. Thus, any potentially concurrent access to the communication hardware peripherals (i.e., CAN, LIN, FlexRay, and Ethernet controllers) from different cores can be ruled out. In addition, fully independent and parallel execution of the different sub-stacks is possible without interaction among them.

To drive this separation and independence even further, we split the general network-type independent BSW modules of the communication stack (i.e., IpduM and Com) into different parts. Each part is equipped with a dedicated processing function that takes care of processing the subset of the communication originating from, or targeting to, a particular network type. Those dedicated processing functions are then allocated on the dedicated core for the respective network type.

By doing this, we effectively keep all of the communication of a particular network type local to a single core and rule out interference with the communication of any other network type. Thus, we avoid inter-core communication and synchronization, maximize the independent execution of the different communication sub-stacks, and are able to keep most of the AUTOSAR communication stack's synchronous API calls local to the respective core.

The communication paths originating on one network type and targeting some other network type (i.e., gateway routing paths), and communication paths targeting multiple network types (i.e., multicast routing paths), are handled by a multicore-capable PDU router (PduR). The PduR takes care of the required core transitions in those routing paths using the SchM's intercore communication capabilities. Buffering or queuing within the PduR facilitates the use of asynchronous (instead of synchronous) inter-core calls. This results in a decoupling of caller and callee, and thus keeps the execution of the sub-stacks for the different network types independent even for these kinds of communication paths.

This kind of core allocation of the AUTOSAR communication stack results in the multicore communication stack architecture *(see figure)*.

Successful OEM Implementations

The approach described here has been successfully

implemented and deployed in two real-life automotive series projects for a major German car manufacturer. The first project dealt with the central gateway electronic control unit (ECU) of a premium vehicle that required a vast amount of data to be routed between different networks and exhibited very complex routing paths. In this setup, an STMicroelectronics Chorus 6M MCU was used, where the CAN, FlexRay, and Ethernet sub-stacks were each allocated to dedicated cores.

The second project dealt with a powertrain domain master ECU exhibiting time-critical event chains involving multiple ECUs and requiring strictly deterministic timing on several CAN networks. In this setup, an Infineon AURIX 2G MCU was used, where the CAN and LIN sub-stacks were allocated on one core and the FlexRay and Ethernet sub-stacks were allocated on another core.

Due to the reduced number of communication paths crossing core boundaries in this project, almost no overhead for inter-core communication and synchronization (less than 1% additional CPU load) was measurable. As far as memory mapping is concerned, we gathered access statistics under realistic load scenarios and optimized the memory mapping for frequently accessed code and data. The optimized memory mapping reduced CPU load by 15% compared to an unoptimized memory mapping.

Summary

The efficient use of multicore MCUs requires distribution of the AUTOSAR basic software in general, and particularly of the communication stack. We proposed to split the communication stack according to the different network types to prevent concurrent access to the communication hardware peripherals, to allow for fully independent and parallel execution of the different sub-stacks, and to reduce the need for inter-core communication and synchronization.

We recommend locating code and data within memory with a strong affinity to the respective core using AUTOSAR's static memory-mapping functionality to properly use fast core-local memory as well as prevent/reduce conflicts upon access to slower global memory. Implementing this approach and deploying it in two series projects for a major German OEM showed that by means of distributing the communication stack and doing a proper allocation of the application software components, an efficient use of the multiple cores of an AURIX 2G MCU can be achieved. And there's almost no overhead for inter-core communication and synchronization.

Dr. Thomas Galla is chief expert of automotive networks at Elektrobit. In this role, he's focused on AUTOSAR-based ECU development, multicore software, and communication stacks. Thomas has over 15 years of experience in his field. He earned his Master's in computer science from Vienna University of Technology in 1995 and received his Ph.D in 2000.