Unleashing the Power of Multi-Core MCUs by AUTOSAR Communication Stack Software Distribution

Thomas M. Galla
Elektrobit Austria GmbH
Vienna, Austria
Email: thomas.galla@elektrobit.com

I. Introduction

Until 2005 the software development in the area of mainstream computing enjoyed a free lunch profiting from the exponential growth of transistor densities and resulting clock speeds of Moore's Law [1]. Existing software benefitted from hardware changes without requiring modification. In 2005, however, mainstream computing hit a wall due to the limit in increasing the clock speed. As a result, chip manufacturers started to turn towards hyperthreading and homogeneous and later on heterogeneous multi-core architectures in order to achieve further performance gains. To benefit from these hardware changes, existing software had to be parallelized and modified to deal with the heterogeneity. The period of the uni-core free lunch was over.

The very same development that happened in mainstream computing is currently taking place in the automotive domain as well. Looking at modern micro controllers (MCUs) and systems-on-a-chip (SoCs) like Infineon's AURIX 2G [2] or Nvidia's Drive Xavier [3] we can clearly see the trend moving towards homogeneous and even heterogeneous multi-core hardware architectures. To benefit from this hardware development, changes in the automotive software including changes in standardized software architectures like AUTOSAR's layered software architecture [4] are required as well.

II. AUTOSAR Communication Stack Software Distribution

Starting with AUTOSAR 4.0.1 support for multi-core MCUs has been introduced into AUTOSAR by providing means to allocate application software components (SWCs) to dedicated cores and by facilitating the cross-core communication between those SWCs via the runtime environment (RTE) [5]. The whole AUTOSAR basic software (BSW), however, needed to be allocated to a single core. With AUTOSAR 4.2.1 the AUTOSAR basic software was divided into so-called functional clusters which could be allocated to different cores using the BSW schedule manager (SchM) for inter-core communication [6]. Since the communication stack as a whole is such a functional cluster, distribution of the communication stack over multiple cores was not supported. Although AUTOSAR 4.4 introduced the possibility to distribute the BSW modules of the lowest layer (the micro controller abstraction layer (MCAL)) the remainder of the AUTOSAR communication stack still had to be placed onto a single core. Having in mind that Amdahl's law [7] basically implies that the sequential part of the software imposes a theoretical limit on the speedup achievable by a multi-core MCU, it is obvious that a monolithic communication stack allocated to a single core will eventually become the performance bottleneck. Therefore, the distribution of the communication stack over the different cores is mandatory for harvesting the performance benefit of multiple cores.

When working on communication stack software distribution, the following things need to be considered in order to make efficient use of the multi-core resources. Inter-core communication and synchronization should be reduced as much as possible, since they typically involve inter-core interrupts which in turn lead to changes in the MCU's operation mode (transition from user to supervisory mode), pipeline stalls, and cache misses. In case inter-core calls are required and cannot be avoided, asynchronous calls should be favored over synchronous ones, since the latter block the caller until the callee is finished, thereby reducing the degree of parallelism and thus the potential speedup. Unfortunately this is not 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 re-design. In addition to that 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 additionally wastes CPU cycles on the blocked cores.

Another crucial thing to keep in mind is the fact that multi-core MCUs usually exhibit a non-uniform memory architecture (NUMA). Memory is divided into core-local memory (caches, flash, and RAM) dedicated to a single core, which can be accessed fast 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 access conflicts. In such a non-unifirm memory architecture the proper placement of code and data needs to be carefully considered. Frequently accessed code and data needs to be placed as close to the accessing core as possible. Using the static AUTOSAR memory mapping mechanisms [8] such placement should be performed based on access statistics derived under realistic load scenarios.

With those considerations in mind, we derive 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. This way we rule out any potentially concurrent access to the communication hardware peripherals (i.e., CAN, LIN, FlexRay, and Ethernet controllers) from different cores and thus allow for fully independent and parallel execution of the different sub-stacks 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 [9] and Com [10]) into different parts where each part is equipped with a dedicated processing function which takes care of processing the subset of the communication which is 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 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 the majority of the synchronous API calls of the AUTOSAR communication stack 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 multi-core capable PDU router (PduR) [11], which takes care of the required core transitions in those routing paths using the SchM's inter-core communication capabilities. By means of buffering or queuing within the PduR the use of asynchronous (instead of synchronous) inter-core calls is facilitated which 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 multi-core communication stack architecture depicted in Figure 1.

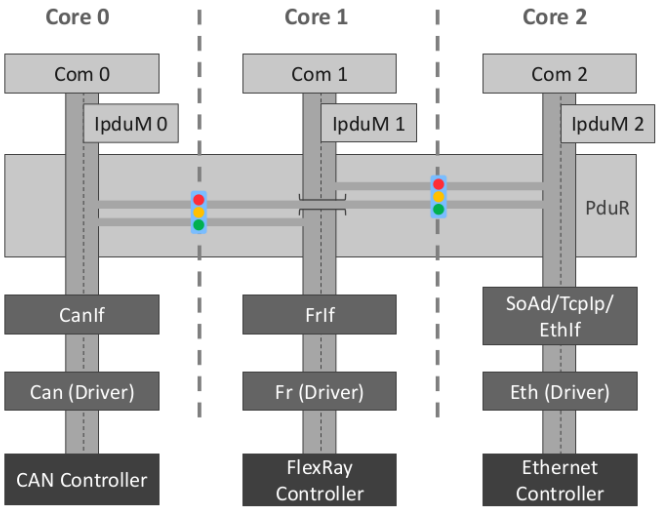

Fig. 1. Multi-core Communication Stack Architecture

The presented approach 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 ECU of a premium vehicle requiring a vast amount of data to be routed between different networks and exhibiting very complex routing paths. In this setup an STMicroelectronics Chorus 6M MCU [12] has been 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 ehibiting time-critical event chains involving multiple ECUs and requiring a strictly deterministic timing on several CAN networks. In this setup an Infineon AURIX 2G MCU has been used where the CAN and LIN sub-stacks were allocated on one core whereas the FlexRay and Ethernet sub-stacks have been 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 in this setup. 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. This optimized memory mapping resulted in an overall reduction of 15% CPU load compared to a naïve unoptimized memory mapping.

III. Summary & Conclusion

In this paper, we argued that the efficient use of multi-core MCUs requires distribution of the AUTOSAR basic software in general and especially 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 to locate code and data within memory with a strong affinity to the respective core using AUTOSAR's static memory mapping functionality to properly make use of fast core-local memory and to 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 with almost no overhead for inter-core communication and synchronization.

References

[1] G. E. Moore, "Cramming More Components Onto Integrated Circuits," *Proceedings of the IEEE*, vol. 86, no. 1, pp. 82–85, Jan 1998.

[2] Infineon Technologies AG, "AURIX TC39xXX/TC39xXP – High-Performance Chassis, Powertrain, Connectivity and Autonomous Driving Microcontroller," Infineon Technologies AG, Tech. Rep., Oct 2017.

[3] M. Demler, "Xavier Simplifies Self-Driving Cars – Nvidia's SoC is First Single-Chip Autonomous-Driving System," *Microprocessor Report*, vol. 2017, no. 6, Jun 2017.

[4] AUTOSAR Consortium, "AUTOSAR – Layered Software Architecture," AUTOSAR Consortium, Tech. Rep. Release 4.4, Rev 0, Oct 2018.

[5] ——, "AUTOSAR – Specification of RTE Software," AUTOSAR Consortium, Tech. Rep. Release 4.4, Rev 0, Oct 2018.

[6] ——, "AUTOSAR – Guide to BSW Distribution," AUTOSAR Consortium, Tech. Rep. Release 4.4, Rev 0, Oct 2018.

[7] G. M. Amdahl, "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities," in *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, ser. AFIPS '67 (Spring). New York, NY, USA: ACM, 1967, pp. 483–485. [Online]. Available: http://doi.acm.org/10.1145/1465482.1465560

[8] AUTOSAR Consortium, "AUTOSAR – Specification of Memory Mapping," AUTOSAR Consortium, Tech. Rep. Release 4.4, Rev 0, Oct 2018.

[9] ——, "AUTOSAR – Specification of I-PDU Multiplexer," AUTOSAR Consortium, Tech. Rep. Release 4.4, Rev 0, Oct 2018.

[10] ——, "AUTOSAR – Specification of Communication," AUTOSAR Consortium, Tech. Rep. Release 4.4, Rev 0, Oct 2018.

[11] ——, "AUTOSAR – Specification of PDU Router," AUTOSAR Consortium, Tech. Rep. Release 4.4, Rev 0, Oct 2018.

[12] STMicroelectronics, "SPC584Gx, SPC58EGx, SPC58NGx Datasheet – 32-bit Power Architecture Microcontroller for Automotive ASIL-D Applications," STMicroelectronics, Tech. Rep., Feb 2018.