Safety Issues of Integrating IVI and ADAS functionality via running Linux and AUTOSAR in parallel on a Dual-Core-System[1]

Jörn Schneider, Tillmann Nett

Department of Computer Science
Trier University of Applied Sciences
Schneidershof
54293 Trier
J.Schneider@Hochschule-Trier.de
T.Nett@Hochschule-Trier.de

Abstract: The tight integration of In-Vehicle-Infotainment (IVI) applications and Advanced Driver Assistance Systems (ADAS) or even semi-automated driving functionality on the same hardware offers new chances for innovations in the automotive domain. One of the major challenges in this respect is to achieve a solution that satisfies the heterogeneous requirements from the involved application and operating system worlds and guarantees the necessary freedom from interference to attain the required safety according to ISO 26262 [ISO11]. This paper presents a prototypical solution and discusses the remaining challenges for the chosen virtualization-less approach.

The prototype was developed for the practical use in the research project econnect Germany and successfully used in electric vehicles of a field study in Trier. An open source variant of AUTOSAR OS and Linux run together on the same processor of the system. Each operating system uses it own processor core. The chosen solution allows for the interaction between the different applications of the two operating systems and requires no virtualization layers thus avoiding additional resource demand and communication latencies.

1 Introduction

A huge number of innovative functions in modern vehicles utilize the combination of formerly separated functional domains. Since the introduction of the CAN bus into vehicles ever more communication links have been established in order to enable interaction between previously isolated functionalities. A new area of functional interaction with promising possibilities are In-Vehicle-Infotainment (IVI) and Advanced

[1] This work is partly funded by the German Federal Ministry for Economic Affairs and Energy under grant number 01ME12043 within the econnect Germany project.

Driver Assistance Systems (ADAS) or other safety-related vehicle functions. Apart from this area there are further classes of future systems that could benefit from the integration of open source operating systems such as Linux or Android and AUTOSAR applications on the same hardware. Developing such systems confronts the industry with the challenge of integrating a general purpose operating system and its applications with a real-time operating system and its safety-related applications.

Within the econnect Germany project a prototype of a system running Linux and AUTOSAR on separate cores of the same processor was developed and used for a field study in five modified electric series cars. Although the system's intended functionality was different from an ADAS the basic challenges are quite similar. In this paper we investigate the suitability of the chosen approach for combining IVI and ADAS functionality or similar cases with safety-related real-time applications and general purpose functionality. This is done by summarizing the general challenges in Section 3, describing the chosen realization and identifying remaining issues in Section 4, and discussing the overall concept in comparison to other solution approaches in the final Section. Additionally, the initial usage background of the developed prototype system is given in Section 2 to allow for a better understanding of the chosen design.

2 Usage Background of the Prototype

The current system was implemented during the project econnect Germany for a field study in the City of Trier. The goal of the field study is to evaluate the user acceptance of Grid2Vehicle (G2V) and Vehicle2Grid (V2G) technologies for electric vehicles. Widespread deployment of electric vehicles poses great challenges for the energy sector, e.g., due to high and unbalanced loads on electric power networks. However, electric vehicles and renewable energies provide also great synergies [KT05]. With electric vehicles the fluctuating generation of renewable energies can be buffered. Charging of the vehicles may be adapted to current production. Using V2G technology, it will for instance be possible to provide energy from the vehicle batteries to the grid to compensate for low production.

Using V2G and G2V technologies necessitates a distributed IT architecture with an essential part in the vehicle itself. For instance the planning of the charge processes should be based on current and expected energy production, the conditions in the vehicle (e.g. state of charge) and the settings by the driver [NS14]. The latter requires an on-board computer with an interface through which the driver is able to input parameters such as needed range and time of departure. Furthermore, this on-board computer must plan and execute the charge processes. Both for planning as well as for execution of the charge processes a direct communication with other hardware within the vehicle is needed. For example to execute the charge process the system must communicate with the battery management system. Additional requirements for the on-board computer in the field study originate from the aim of evaluating end user acceptance. For instance, GPS coordinates of all trips were recorded.

Such a system includes parts with safety requirements. Although the functionality of the prototype itself is not safety-relevant, the communication on the main vehicle bus must not be adversely influenced as this might violate safety goals of other systems. Moreover, communication within the vehicle often poses hard real-time constraints. For these reasons, it was decided to use an AUTOSAR operating system for all parts with safety or real-time requirements.

Other requirements could not be immediately addressed with AUTOSAR, for instance the need to provide a graphical user interface with touch control and a TCP/IP based communication link via UMTS to the field study server and the gateway to the energy control center of Stadtwerke Trier and the charging infrastructure. Therefore, Linux was chosen as base for these functions.

Regarding the interaction between the two worlds of applications (AUTOSAR and Linux) the requirement was established to have a most flexible interface allowing for data communication with a high bandwidth and the invocation of services across OS boundaries. Last but not least, the chosen approach should be scalable to other system classes such as the combination of ADAS and IVI functionality as considered in this paper.

According to these requirements the prototype was realized as a hybrid system with AUTOSAR and Linux running on separate cores of the same microprocessor. The interaction between the system parts uses shared memory, thus allowing for the required flexible, high bandwidth interface.

3 Safety Challenges

The integration of IVI and ADAS functionality comes with challenges along more than one dimension. The focus of this paper is on the safety issues, therefore other aspects are largely ignored unless a clear relation to functional safety exists.

One problem of integrating safety-related functionality on IVI hardware is that hardware of this class might not be adequately qualified for safety-relevant issues. The automotive industry needs to monitor this issue and find suitable solutions. It is clearly beyond the scope of this paper to discuss this aspect further. However, the interested reader might have a look into a recent position paper related to this issue, that is part of an ongoing discussion on the usage of consumer electronics components in automotive applications [ZVEI14].

Another issue with such hardware is that it is typically designed to allow for a good average case performance and neglects the fact that worst case performance is of utmost importance for real-time systems. Typically this makes the precise prediction of worst case execution and reaction times of software on these systems extremely hard [Cul10]. Although several special aspects regarding real-time behavior are discussed throughout this paper, in-depth considerations of WCET analysis or schedulability analysis are out of the scope.

3.1 Observations on the ISO 26262

Part 9 of the ISO 26262 norm [ISO11] discusses „criteria for the co-existence of elements“. The considered cases are „coexistence within the same element of safety-related sub-elements with elements that have no ASIL assigned; and safety-related sub-elements that have different ASILs assigned“. Clearly this includes the case of IVI and ADAS software running on the same hardware. The norm prescribes that all sub-elements need to be developed to the highest ASIL level unless freedom from interference can be shown. As no Linux implementations are available that can be considered to satisfy ASIL A or higher, it is mandatory to show freedom from interference whenever ISO 26262 applies to such a combination.

The key term “freedom from interference” is often used in the context of combining safety-related and non-safety-related functionalities (or regarding combination of functionalities with different requirement classes on safety, e.g. ASIL). There are sometimes misconceptions about this term. First of all „freedom from interference“ does not mean that there is (or has to be) no influence between different classes of functionality. A sensible understanding could be that no undesired influence is allowed to be established. At least one would expect that harmful influence cannot occur. Surprisingly the definition used in the ISO 26262 norm is weaker than that.

Part 1 of the norm [ISO11] defines „freedom from interference“ as: „absence of cascading failures between two or more elements that could lead to the violation of a safety requirement“. Cascading failures are defined as follows according to the ISO 26262: „failure of an element of an item causing another element or elements of the same item to fail“ (cf. [ISO11], part 1). An example of a cascading failure would be, if one application crashes and thereby corrupts data needed by another application in a way that the latter also crashes. A difficulty arising from this definition is that it only captures situations where first one element fails and then one or more further elements also fail due to this first failure (failure: „termination of the ability of an element to perform a function as required“, cf. [ISO11], part 1).[2]

Consider the following example, Linux (or any other software element) starts and performs all functions as required. Additionally it writes into the memory area used by AUTOSAR (or of any other safety-related software element). In a strict sense this would still be understood as „freedom from interference“ according to the definition cited above. Certainly, it cannot be meant that way, because the affected element (e.g. AUTOSAR) could fail in an arbitrary way due to a fault in the causing element, but without the causing element failing by itself. We suggest that the definition of freedom from interference should also include this case of error propagation and only be used in the then stronger way of understanding. In the remainder of this paper we use freedom from interference in the stronger sense, i.e. including freedom from error propagation with subsequent failure. Please note, that unless expressed otherwise the terminology established in [Avi04] is used throughout this paper.

[2] The norm clearly distinguishes between “fault” and “failure” as in [Avi04]. Cascading failures according to the norm require one component to fail, it is not enough for the component to cause a fault.

In other parts the norm refers to another type of failures with importance for the considered case of integrating IVI and ADAS functionality: common cause failures. A common cause failure has a single root cause and affects two or more parts of a system, e.g. the AUTOSAR and the Linux part. The co-location of AUTOSAR and Linux on the same hardware clearly increases the potential for common cause failures. Potential root causes can be faults of the shared parts of the hardware, i.e. almost everything except for the processor cores. Moreover, the software responsible for booting the system could become a common cause for a failing start-up of both AUTOSAR and Linux in the current system design. The section on implementation gives details regarding the currently used boot sequence.

3.2 Relation between Safety and Security

For the considered combination of IVI and ADAS functionality on the same hardware it is not sufficient to focus on non-malicious faults, i.e. ignoring the potential actions of humans with the aim of causing harm. This includes the case of intruders on the Linux part of the system. Many solutions that are sufficient without malicious actions are rendered ineffective when human intruders need to be considered. Please note, that the current system relies on standard solutions to achieve security under Linux. This is not considered to be sufficient by the authors for the combination of IVI and ADAS systems. However, it is beyond the scope of the paper to discuss potential improvements in detail.

4 Current Implementation

The current prototype was previously presented in [NS13]. This section will now present a more detailed overview of how separation was achieved and which issues still remain open. The prototype was implemented on a Pandaboard ES (Revision B). The Pandaboard ES is an embedded development platform using the OMAP4460 SOC [TI14]. The OMAP4460 is based on the ARM architecture and comprises two Cortex-A9 cores, two Cortex-M3 cores as well as some other specialized cores for image processing, face detection etc. For the prototype only the two Cortex-A9 cores were used.

In addition to the Pandaboard, several peripheral hardware was added. To communicate with other ECUs, an MCP2515 CAN controller was connected via SPI. To track the location of the vehicle a GPS module was added, which is connected via UART. Both the GPS as well as the CAN module are controlled by AUTOSAR. For the user interface a display was connected over HDMI. This display is combined with a touch module, which is connected over USB. Finally, for communication a USB UMTS stick was used.

4.1 Boot Sequence

During start-up of a OMAP4460 the Cortex-A9 Core0 is initialized first by the ROM code. The ROM code fetches the boot code from non-volatile storage and starts executing it. The system uses Das U-Boot [De12] as a boot-loader to load an AUTOSAR

conforming operating system from non-volatile storage. This AUTOSAR image currently also includes a complete Linux kernel statically linked in a separate section of the executable. The additional space used up by the Linux kernel within the AUTOSAR image increases the load time from the non-volatile storage medium. We added additional start-up code which configures and starts the other core. On the second core a short start-up routine is used to load the Linux image contained in the AUTOSAR image and transfer control to the Linux image. All start-up parameters needed by Linux are provided by the core1 start-up routine and are written to the correct locations in memory. These parameters also include the `maxCPUs=1` option, which instructs the Linux kernel to run in single core mode.

There are some possibilities to further reduce the start-up times of AUTOSAR that were not currently taken. First, the load time of the AUTOSAR image with the added Linux image could be improved by adding a driver for the non-volatile storage and loading Linux on the second core. In this case, the load time for AUTOSAR would only be increased because of the added size due to the driver. Furthermore, Das U-Boot initialized several hardware modules that are not used by AUTOSAR, such as for example USB. Some of these hardware modules are later used without further re-initialization from within Linux, so that the initialization routines within Das U-Boot could not be removed. However, these initialization routines could be moved to the second core, so that the start-up of AUTOSAR is not delayed further by these hardware initializations. This additional initialization code could also be read from non-volatile storage on the second core, as not to increase the size of the AUTOSAR image.

4.2 Hardware Assignment

To reduce the possibility for interference between safety levels the hardware was assigned statically to one of the two cores, if possible. For shared hardware, it was necessary to protect all configurations made by AUTOSAR from changes through Linux and vice versa. Therefore, shared hardware was either initialized by the boot loader prior to starting AUTOSAR (e.g. main memory, interconnects) or the configuration routines in Linux were disabled or protected. Protection was implemented using the hardware locking mechanism provided by the OMAP4460 architecture [TI14]. The hardware assignment is shown in Table 1.

Hardware	Assigned to
Screen	Linux
Touch-pad	Linux
UMTS module	Linux
USB subsystem	Linux
Non-volatile memory (SD-Card)	Linux
Main Memory (DRAM)	Linux/AUTOSAR (configured by u-boot)
L1 Caches	One per Core (only activated in Linux)
L2 Cache	Linux

L3 OCM RAM (SRAM)	AUTOSAR
CAN module	AUTOSAR
SPI interface	AUTOSAR
UART interface	AUTOSAR
L3 and L4 interconnects	Linux/AUTOSAR (configured by u-boot)
Interrupt Distributor	Linux/AUTOAR (protected using HW spinlocks)
Interrupt CPU Interfaces	One per OS
Hardware spinlocks	Linux/AUTOSAR (configured by AUTOSAR)
Clock Tree	Linux/AUTOSAR (configured by u-boot and managed by Linux)

Table 1: Hardware Assignment (taken with additions from [NS13]).

4.3 Memory

The used platform provides several types of memory, which were used: Non-volatile memory (SD-Card), 1GB of DRAM, 1Mb L2 cache shared between cores, 32kB L1 data cache per core, 32kB L1 instruction cache per core and 56kB SRAM. The non-volatile RAM was fully assigned to and managed by Linux.

Within DRAM a special area was reserved for AUTOSAR. The physical address range used for this area was reserved within Linux using the `mem=` kernel command line parameter. Using this parameter the reserved range is by default excluded from any mappings in the MMU. However, additional mappings to this memory region can be added manually within the kernel. Furthermore, this memory may be accessed using the `/dev/mem` device node with root privileges. Therefore, the AUTOSAR memory region was protected from unintended accesses using the MMU. Security flaws within Linux that provide elevated privileges or access to kernel space may still be abused to tamper with the AUTOSAR memory region. The reserved memory region is also used for communication facilities. For this a part of the memory region is mapped from a kernel module as IO memory. This shared memory can be used to implement the data structures necessary for communication. For AUTOSAR the reserved section of the DRAM was used for code and global data.

To ensure predictability the MMU was deactivated by AUTOSAR for its own core. By deactivating the MMU, unpredictable delays from unexpected TLB refills were avoided [Sch12b]. Also, for OMAP4460 SOCs disabling the MMU also disables all caches for memory used by that core. This way interferences between Linux and AUTOSAR from sharing the L2 cache and cache coherency protocols are avoided. However, disabling the caches also greatly reduces the performance of the system. As a first step to mitigate this problem, the SRAM section was completely assigned to AUTOSAR and disabled in Linux. In our final implementation, all stacks were moved to the SRAM to increase

speed[3]. While this method provides some speed improvements, future modifications can further improve the usage of SRAM for speedup. Also instead of statically assigning parts of the system to SRAM it may be possible to use the SRAM as a scratchpad memory managed by the OS or the compiler [ABS01]. Furthermore, freedom from interference only requires disabling the shared L2 cache. Depending on the architecture, it may be possible to only enable the unshared L1 cache [ARM11a]. Assigning the SRAM section to AUTOSAR, as in the current implementation, requires that the SRAM is not used by Linux. Currently SRAM may be accessed within Linux using a kernel module. To test if the memory could be safely assigned, this module was instrumented with log outputs. These outputs showed that SRAM was not used by the Linux portion.

The current solution leaves two main issues. First, using the `mem=` parameter the address space of AUTOSAR is excluded from the memory mapping used by Linux. However, there is no restriction within Linux preventing it from changing this memory mapping. Erroneous code within the kernel or with root privileges could in principle still map any part of the AUTOSAR address space, hence impairing safety requirements. For this reason all parts of the user interface were run without root privileges if possible. Secondly, disabling the caches within AUTOSAR will greatly decrease the speed. While assigning the SRAM to AUTOSAR may mitigate this problem somewhat, this only partially solves the issues. A better solution would be to enable L1 caches globally and also analyze for which portions of the code or data the L2 caches could be activated as well.

4.4 Interrupts

The OMAP4460 SOC uses an ARM generic interrupt controller (GIC) [ARM11b] to configure and deliver interrupts to the cores. The GIC is composed of a single shared interrupt distributor and one CPU interface per core. The interrupt distributor receives interrupt signals and distributes them to the CPU interfaces for handling by the core. Interrupt signals can be distributed to one or more cores [ARM11b].

In the current state of the system, interrupts are statically assigned to one of the two cores. During start-up AUTOSAR configures all interrupt signals required by itself to be targeted to the core on which it is running. The initialization routine for the GIC within Linux has been changed not to reassign any interrupts targeted for the AUTOSAR core. Additional care has been taken to avoid race conditions. For the ARM GIC used by the OMAP4460 architecture interrupt targets are configured by using memory mapped registers. For these registers one processor word is used to manage the targets for four different interrupts. In Linux the mapped words are written using a read-modify-write pattern. To avoid races with AUTOSAR these registers are protected using a HW lock[4].

[3] For the version of the system presented in [NS13] all data portions were put into SRAM. Later changes to the system however caused the bss section to grow beyond 56kB so that this section was removed from SRAM.
[4] The ARM GIC specification [ARM11b] indicates that these registers are also byte accessible. However the Linux kernel currently does not use this method to set the targets, so that races are possible if both AUTOSAR and Linux try to modify the same register and no additional locking is used.

If it can be guaranteed that all interrupts are completely initialized within AUTOSAR prior to starting Linux these locks are not needed. In that case, it can also be guaranteed that the shared hardware does not introduce any additional delays. Similarly, interrupt priorities are managed by using one word to store the priorities of four different interrupts. Again, these registers must be protected using HW locks shared between AUTOSAR and Linux or it must be ensured that all priorities are set by AUTOSAR prior to loading Linux and are not changed while the system is running.

Masking interrupts can be done in multiple ways using the ARM GIC. First, all interrupts can be disabled for a single core using the `cpsid` instruction. Second, some portion of the interrupts can be masked for a single core using priority masking in the CPU interface for that core. Third, single interrupts can be globally masked using the distributor. Currently within the used AUTOSAR implementation interrupts are only disabled via the `cpsid` instruction. No masking of single interrupts or groups of interrupts is done. However, these functionalities could easily be used. Masking based on interrupt priorities can be done using the CPU interface, so that no shared hardware is needed. Also, the registers of the distributor for masking/unmasking of single interrupts are implemented race-free by using separate set and clear registers. This means no additional locks are needed for performing masking of interrupts independently on both systems.

Using the current solution there is no guarantee that Linux will not deactivate any interrupts targeted towards AUTOSAR either by masking them or by setting another target core. However like any hardware, the GIC registers are protected within Linux by the MMU. The code which maps these registers to configure the GIC is very small and could be easily verified. If this code is instrumented so that it will not change any interrupts targeted towards AUTOSAR it would be possible to ensure that this portion of the Linux kernel does not cause any failures of safety-related functions. This, however, still leaves the possibilities of other parts of the Linux kernel mapping the GIC registers and changing the configuration of the interrupts. This could either be done by an attacker or because of severe bugs in the Kernel. One possibility to mitigate this issue would be to secure any access to the MMU, so that no additional mappings to the GIC registers can be introduced.

4.5 Interconnects

The OMAP 4460 SOC uses several types of interconnects for communication between hardware components. Both Cortex-A9 cores are connected to most other SOC hardware by a local interconnect. This local interconnect further connects to the L3 interconnect and the L4_ABE interconnect. The L4_ABE interconnect serves hardware, which is part of the audio backend. Other hardware, which is used (UART, SPI, some timers) is served by the L4_PER interconnect, which in turn is connected to the L3 interconnect [TI14].

Since the system only provides a single set of interconnects for both cores, these interconnects have to be shared between Linux and AUTOSAR. This also means, that

requests to configure or service hardware from Linux may delay requests from AUTOSAR. The L3 interconnect uses a leaky bucket algorithm [Tu86] for bandwidth regulation [TI14]. Using this algorithm the bandwidth for each master on the interconnect is limited and data can be processed at a higher priority as long as the maximum configured bandwidth is not exceeded. However, since both cores in the Cortex-A9 dual processor are connected to the L3 interconnect using the same local processor interconnect, the bandwidth for both cores can only be set together. Still, this method may be used to protect accesses to safety relevant hardware by the processor, to guarantee freedom from interference from other hardware.

Memory is connected directly to the cache and memory controllers, which are part of the dual core Cortex-A9 subsytem [TI14]. Thus, memory read or write accesses from the core that is running AUTOSAR only need to be arbitrated with other memory accesses. No arbitration is needed for accesses to memory and accesses to other hardware from Linux.

Currently AUTOSAR also uses two timers, which are connected using the L3 and L4_PER interconnects. These timers are used for system ticks and timeouts of the UART module. In case of heavy congestion on one of the two interconnects, configuration updates to these timers could be delayed. As a possible solution timers which are part of the ABE module could also be used. These timers are fully accessible from the Cortex-A9 cores using the L4_ABE interconnect only. If these timers are used and the ABE is disabled within Linux, no congestion can appear on the L4_ABE interconnect.

As discussed congestion may be an issue, which can influence real-time behavior. However, most important hardware, e.g. memory, is connected separately to the cores thus limiting the potential effects. Furthermore, depending on the hardware it may be possible to use reserved interconnects to avoid any congestion. For example, in case of the OMAP4460 SOC timers within the ABE backend could be used [TI14]. While this may require deactivating some part of the hardware, this may be easily replaceable for example using a soundcard connected via USB.

4.6 Clock-Tree and Power Management

Currently clock-tree and power management are performed by Linux. The complete clock-tree is initialized during start-up by U-boot and later managed by Linux. Experiments on the hardware platform showed a risk of overheating during full load, if the speed is not throttled. If code on the Linux side can cause the SOC to fail due to overheating, this could also severely impact safety-related functions. Hence, a thermal management is needed for reliability. This thermal management is currently provided by the Linux kernel. Unfortunately, the CPU speed can only be set for both cores together, so that Linux will also control the speed of AUTOSAR. To ensure that all deadlines are met, timing analysis on AUTOSAR was done at the lowest possible rate. However, the uncontrolled change of clock speed can cause serious problems in real-time software, even if scheduling and timing anomalies could be avoided. Therefore, the control of the clock speed should be given to the safety-related real-time part, i.e. AUTOSAR in the case of combining IVI and ADAS functionality. Moreover the current implementation

limits the processing capacity that can be used for safety-related functions to the capacity provided by the chip at the lowest frequency. A closer analysis which frequencies do not cause thermal issues could also allow running the system at a higher speed. In that case all lower frequencies should be disabled within Linux.

It would be possible to implement the thermal management within AUTOSAR. With such an implementation, either power management could be done by AUTOSAR independently without regards for Linux. If the Linux programs can run at arbitrary speed, this is the simplest solution. On the other hand, it would still be possible to use the normal power management infrastructure within Linux and forward the decision of the installed CPUfreq governor to AUTOSAR. AUTOSAR could then use this additional information to provide the needed CPU speed to Linux.

Power management is a severe issue for any system using multiple operating systems. Throttling the CPU is only possible in cooperation with both systems. Instead of throttling the usual solution is to temporarily completely halt cores when possible [Chi08]. This method is also employed by the current prototype. Both Linux and AUTOSAR perform a `wfi` instruction in their idle tasks, which reduces the power consumption of the current core. Still thermal issues may require more sophisticated solutions to the problem of power management. Without any thermal management depending on the hardware an additional point of failure or attack would be opened. For example, an attacker could just run a program which burns CPU cycles, thus increasing the heat, which may lead to a failure of both cores. Such an attack might also be run without any elevated privileges. Similarly, faulty code could also burn CPU cycles and increase the heat on the die. This means that any solution in which safety-related functions are combined with non-safety-related ones on the same hardware requires a thermal management which can be verified not to lead to any failures. This also applies to solutions in which there is a partitioning hypervisor.

4.7 Communication

To share data between AUTOSAR and Linux real-time communication facilities were added. For stream transmissions a non-blocking ring buffer implementation was used. Multiple ring buffers are provided for multiple data streams. Using these ring buffers AUTOSAR tasks can send data packets to Linux, which are then received in a kernel thread (bouncer thread). This kernel thread takes the packet and copies it to a lookaside cache, which is created using the `kmem_cache_create()` method [CKR05]. Lookaside caches are frequently used within Linux to flush buffers used by hardware, e.g. network cards, to internal kernel buffers. The `kmem_cache_create()` provides a pool allocator for blocks of fixed size. Blocks which are freed are not directly returned to the free memory managed by the kernel, but rather to the pool for quick re-use. Pools for blocks of equal size can be shared. The bouncer thread is scheduled every 100ms and removes all packets from the ring buffer to the lookaside cache so that the ring buffer itself remains usable from within AUTOSAR, even if the packets are not picked up by a user-space process. All packets are stored in a linked list, which can be retrieved using a device node. In AUTOSAR multiple tasks sending data simultaneously through the same

ring buffer must be synchronized with each other. For this the real-time resource sharing mechanisms within AUTOSAR are used, so that real time capabilities are not impaired. The structure of the communication facilities is presented in Figure 1.

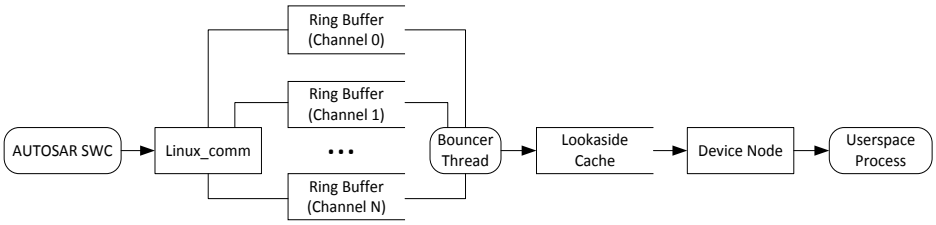

Figure 1: The structure of the communication facilities from AUTOSAR to Linux

In case the lookaside cache grows beyond a configurable limit, no further packets are retrieved from the ring buffers by the bouncer thread. In this case the ring buffers may overflow if further packets are written on the AUTOSAR side. Similarly, if the kernel bouncer thread is not regularly scheduled, this may also cause congestion in one of the ring buffers. If a ring buffer overflows the packet is dropped within AUTOSAR to enforce the non-blocking behavior. This may result in loss of data on the side of Linux, but not in any missed deadlines on the AUTOSAR side. As Linux is not supposed to perform any safety relevant functions, the missed data does not impair the overall safety of the system.

To ensure no delays during communication from caches or cache coherency protocols, caching for the shared memory areas is disabled in Linux. Caches are deactivated by mapping the memory area as IO memory within the kernel. No caches are used within AUTOSAR. To keep the pointers indicating begin and end of the ring buffer consistent, data is first written to the buffer within AUTOSAR, then a memory fence is issued and after the fence the pointer is updated. Similarly in Linux first the data is read and then after a memory fence the pointer is updated.

Although this ring buffer mechanism is only used to send data from AUTOSAR to Linux, the opposite direction could be supported as easily. As a secondary communication mechanism for state data (as opposed to stream data), atomic writes to reserved memory words are used. This communication mechanism is currently only used to communicate data from Linux to AUTOSAR.

While the current communication structure could still be improved, for example to decrease the risk of lost data, it does not pose any risks to safety-related functions. If any of the buffers overflow data may be lost. However, since this data is accepted by Linux, this transmission must not be part of any safety-related functions. Similarly, the data that is received from Linux must be treated as unreliable by all safety-related functions. Hence, even if the communication mechanism misbehaves in any way, the safety-related functions cannot be impaired in any way.

5 Competing Solutions

The usual solution to ensure freedom from interference is to use separate hardware for safety-related and non-safety-related functions. However, separating the hardware increases the cost of the system. Also, separating the hardware requires additional communication facilities such as dedicated busses. Moreover, the required flexible and high bandwidth communication channel is not available in such a setting. This impedes innovations that utilize a close coupling of IVI and ADAS functionality.

Another possibility is to develop the complete system according to the highest ASIL. However, this requires extensive verification and testing of non-safety-related functions, which may not be feasible. This would also have to encompass the complete Linux kernel, if a Linux portion is used. While some soft real-time adaptations of the Linux kernel exists [RH07] these have not been developed according to ISO 26262 and hence cannot be employed for safety-related functions. Using different operating systems than Linux typically prevents short development cycles and increases development costs.

A third possibility is to use a hypervisor to virtualize the two systems [ISM09]. However, the additional software layer will have an impact on performance [Bru10,Sch12a] that may not be acceptable in all cases. Also, to use a hypervisor for partitioning of safety-related and non-safety-related functions, it is necessary to verify the hypervisor at the level required by the safety-related functions. Furthermore, it must be possible to guarantee that the scheduling of the hypervisor can ensure real-time performance for the safety-related system parts. This may for example require additional idle time allocated to the real time portions. This further reduces the possible performance.

References

[ABS01] Avissar, O.; Barua, R.; Stewart, D.: Heterogeneous memory management for embedded systems. In Proceedings of the 2001 international conference on Compilers, architecture, and synthesis for embedded systems. ACM, 2001.

[ARM11a] Cortex-A Series - Programer's Guide. Version 2. ARM, 2011.

[ARM11b] ARM Generic Interrupt Controller – Architecture Specification. Architecture Version 2. ARM, 2011.

[Avi04] Avizienis, A.; Laprie, J. C.; Randell, B.; Landwehr, C.: Basic concepts and taxonomy of dependable and secure computing. In IEEE Transactions on Dependable and Secure Computing, 2004, 1, 11-33.

[Bru10] Bruns, F.; Traboulsi, S.; Szczesny, D.; Gonzalez, E.; Xu, Y.; Bilgic, A.: An evaluation of microkernel-based virtualization for embedded real-time systems. In 22nd Euromicro Conference on Real-Time Systems (ECRTS), 2010.

[Chi08] Chisnall, D.: The Definitive Guide to the XEN Hypervisor. Prentice Hall, 2008.

[CKR05] Corbet, J.; Kroah-Hartman, G.; Rubini, A.: Linux Device Drivers, 3rd Edition. O'Reilly, 2005.

[Cul10] Cullmann, C.; Ferdinand, C.; Gebhard, G.; Grund, D.; Maiza, C.; Reineke, J.; Triquet, B.; Wilhelm, R: Predictability considerations in the design of multi-core embedded systems. In Proceedings of Embedded Real Time Software and Systems (2010): 36-42.

[De12] Denk, W: Das U-Boot. 2012. url: http://git.denx.de/cgi- bin/gitweb.cgi?p=u-boot.git;a=summary.

[ISM09] Iqbal, A.; Sadeque, N.; Mutia, R. I.: An overview of microkernel, hypervisor and microvisor virtualization approaches for embedded systems. Report, Department of Electrical and Information Technology, Lund University, Sweden 2110 (2009).

[ISO11] Road vehicles – Functional safety. ISO 26262, First Edition. International Organization for Standardization, 2011.

[KT05] Kempton, W.; Tomić, J.: Vehicle-to-grid power implementation: From stabilizing the grid to supporting large-scale renewable energy. In Journal of Power Sources, vol. 144, no. 1, pp. 280–294, 2005.

[NS13] Nett, T.; Schneider, J.: Running Linux and AUTOSAR side by side. In 7th Junior Researcher Workshop on Real-Time Computing, pp. 29-32, Sophia Antipolis, France, 2013.

[NS14] Nett, T.; Schneider, J.: Automated Planning of Charge Processes for Privately Owned Electric Vehicles. In 3rd International Conference on Connected Vehicles & Expo (ICCVE), 2014.

[RH07] Rostedt, S.; v. Hart, D.: Internals of the RT Patch. In Proceedings of the Ottawa Linux Symposium. Vol. Two. Ottawa, Canada, 2007, pp. 161–171.

[Sch12a] Schneider, J.: Overcoming the Interoperability Barrier in Mixed-Criticality Systems. In Proceedings of the 19th ISPE International Conference on Concurrent Engineering – CE, 2012

[Sch12b] Schneider, J.: Why current Memory Management Units are not suited for Automotive ECUs.In Proceedings of the Automotive Safety & Security, 2012.

[TI14] OMAP4460 Multimedia Device Silicon Revision 1.x. Technical Reference Manual. Version AB. Texas Instruments, 2011 (revised 2014).

[Tu86] Turner, J.: New directions in communications (or which way to the information age?). In IEEE Communications Magazine, Volume 24, Issue 10, 1986

[ZVEI14] Position Paper: Consumer Components in Safe Automotive Applications. German Electrical and Electronic Manufacturers' Association (ZVEI e.V.), July 2014, http://www.zvei.org/Publikationen/Position-paper-Consumer-Semi-Automotive.pdf.