Simulation of Virtual ECUs in the context of ECU Consolidation

Nicolas Amringer, Synopsys GmbH, Germany
nicolas.amringer@synopsys.com

Peter Asemann, Elektrobit Automotive GmbH, Germany
peter.asemann@elektrobit.com

Abstract: Automotive electrical/electronic (E/E) architectures need to be transformed to meet the requirements of future vehicles, such as autonomy and connectivity. One concrete transformation step is to reduce the number of Electronic Control Units (ECUs) from up to 150 to a significantly smaller number by consolidating ECU Software (SW) stacks from various distributed onto a few centralized ECUs. Multicore and hypervisor technologies enable the consolidation of ECUs, on the other hand, software complexity is expected to increase sharply. For example, the number of lines of code per vehicle is expected to increase from 100 to 300 million by 2030. Therefore, ECU testing is becoming increasingly important. Virtual ECUs (vECUs) can be used to execute and test ECU SW stacks in PC-based simulations by replacing the hardware-dependent with simulator-dependent software. This work gives first an introduction to vECUs, especially Level 3 vECUs, and how they can be used in simulations. Next, ECU consolidation and its impact on ECU SW stacks and vECU simulations is examined. Two concrete use cases are described in this context. A summary and outlook follow at the end.

1 Introduction

Virtual ECUs

Virtual ECUs enable the execution of ECU SW stacks in PC-based simulations independent of the availability of real ECUs and prototypes. This brings, for example, the following advantages over testing with real hardware:

- Cheaper test setups
- Earlier testing / faster feedback loops
- Better scalability
- Support of code-level debugging

Depending on the amount of production ECU SW that is executed in PC-based simulations, [IVI20] defines 5 different vECU Levels – ranging from Level 0 to Level 4. A Level 0 vECU does not contain any production ECU SW (e.g. software interfaces) and consists only of a controller model or code generated from the controller model. Level 4 vECUs contain the complete production ECU SW stack as it can be flashed on a real ECU.

This includes, for example, the application and basic software that use the instruction set of the target microcontroller, as well as drivers with hardware-dependent accesses. The simulation of a Level 4 vECU is complex and requires an instruction set simulator (ISS) and appropriate hardware models. Tools like Synopsys Virtualizer support the simulation of Level 4 vECUs.

Level 3 Virtual ECU Simulation

This work focuses on Level 3 vECUs which contain only the hardware-independent parts of the production ECU SW stack. This includes the application and basic software, but not the drivers and the operating system. The instruction set of the target microcontroller does not matter for Level 3 vECUs, instead the ECU SW is compiled for the PC instruction set (e.g. x86). The simulation of a Level 3 vECU can be performed by replacing the hardware-dependent parts of the production ECU SW stack with simulator-dependent software. This is possible, for example, because most of today's ECU SW stacks follow a software architecture, where hardware accesses are typically made through dedicated (and standardized) interfaces and on hardware-related software layers. A widely used standard is for example AUTOSAR – AUTomotive Open System ARchitecture.

Figure 1 shows a Level 3 vECU containing ECU SW according to the AUTOSAR Classic standard. For the sake of clarity, the AUTOSAR runtime environment (RTE) is not shown. The application software is typically provided by an OEM or Tier 1, while tool vendors typically provide the basic software and semiconductors the drivers (see microcontroller abstraction layer). In this example, the Elektrobit (EB) basic software (EB tresos AutoCore) is used, but it could also be any other AUTOSAR stack. The AUTOSAR operating system and the microcontroller abstraction layer (MCAL) are provided by Synopsys Silver – a PC-based simulator for Level 1, 2 and 3 vECUs. In this case hardware-dependent software is completely replaced by simulator-dependent software. There might also be cases where it is possible to simulate the hardware-independent parts of the hardware-dependent, production AUTOSAR operating system and MCAL. However, this will not be discussed further here, as this work focuses on approaches based on standardized interfaces and software layers.

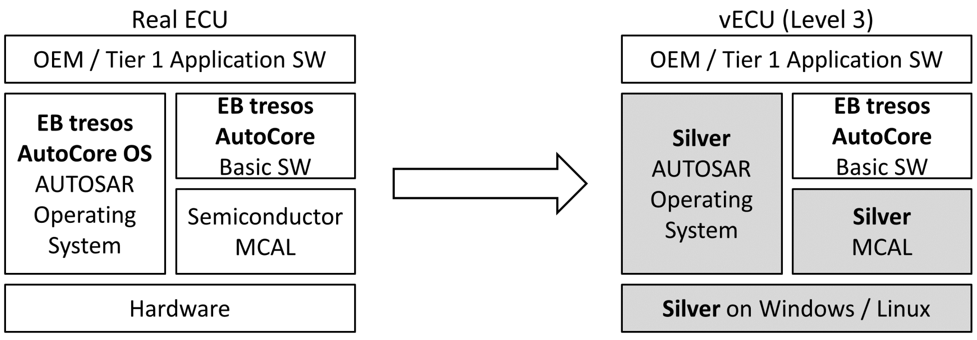
Real ECU
OEM / Tier 1 Application SW
EB tresos AutoCore OS AUTOSAR Operating System
EB tresos AutoCore Basic SW
Semiconductor MCAL
Hardware
vECU (Level 3)
OEM / Tier 1 Application SW
Silver AUTOSAR Operating System
EB tresos AutoCore Basic SW
Silver MCAL
Silver on Windows / Linux

Figure 1: Level 3 vECU: Simulation of application software and EB basic software (EB tresos AutoCore) with Synopsys Silver. Gray colored parts are replaced.

Another important aspect offered by vECUs is the possibility of real-time independent execution. In a discrete-event simulation (DES), a vECU can run on a virtual-time base, allowing it to run slower or faster than the real ECU (real-time). During the execution of events the virtual-time does not pass. For example, it is possible for two or more vECUs to communicate over a virtual bus without considering the bandwidth limitations or delays of the real vehicle bus – the bus communication is idealized because the virtual time does not increase until all bus data has been transmitted from the sending to the receiving vECU. Of course, it is also possible to perform a detailed simulation taking into account delays, bandwidth limitations or other details of the real vehicle communication.

Figure 2 shows two Level 3 vECUs using an EB tresos AutoCore basic software stack and exchanging bus data at the bus frame level (e.g. Ethernet frames). For this purpose, each vECU is connected via an AUTOSAR MCAL driver to a virtual device provided by Synopsys Silver. The communication between the AUTOSAR MCAL driver and the virtual device is based on a simulator-dependent interface. This interface is abstract in the sense that drivers do not access virtual device registers, but use an API to trigger the transmission and reception of bus frames, for example. On the other hand, virtual devices are also not realized as high-fidelity hardware models, but as abstract models which, for example, do not expose any registers to the drivers above. Bus frames provided by the drivers can be transferred from the virtual device over the virtual bus by providing the data at a shared memory location. In the next section, the impact of ECU consolidation on ECU SW stack is considered.

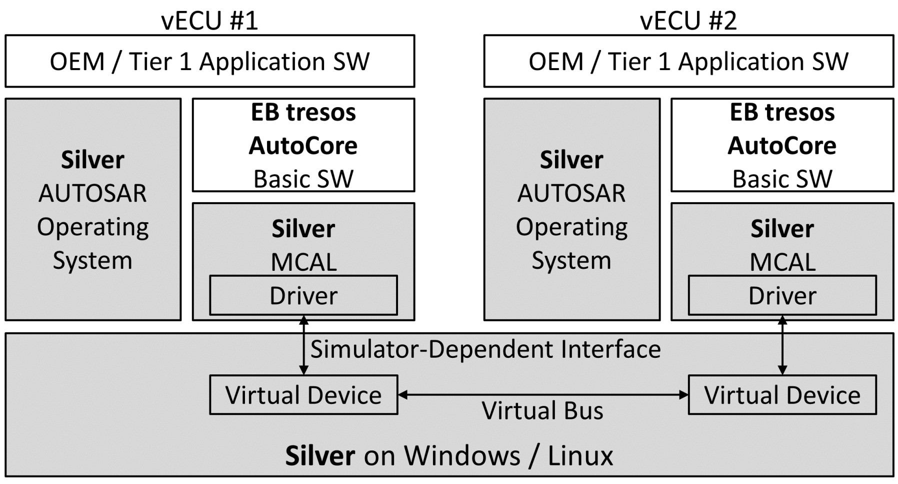
vECU #1
vECU #2
OEM / Tier 1 Application SW
OEM / Tier 1 Application SW
Silver AUTOSAR Operating System
EB tresos AutoCore Basic SW
Silver MCAL
Driver
Silver AUTOSAR Operating System
EB tresos AutoCore Basic SW
Silver MCAL
Driver
Simulator-Dependent Interface
Virtual Device
Virtual Bus
Virtual Device
Silver on Windows / Linux

Figure 2: Virtual bus communication between two Level 3 vECUs.

2 ECU Consolidation

The simulation of vECUs will become increasingly important in the future as the proportion of software in vehicles will increase sharply over the next years. On the other hand, adding more ECUs to vehicles is not possible for several reasons: weight, costs, and power distribution.

To cope with this, E/E architectures need to be transformed into centralized architectures, where various decentralized ECU SW stacks are consolidated on single high-performance ECUs. A promising approach that enables the consolidation of ECUs in real vehicles is virtualization. Modern microcontroller even offer hardware-assisted virtualization support for such use cases. Multiple ECU SW stacks can be executed on a single ECU by using hypervisor technology that allows not only hardware sharing but also isolation between ECU SW stacks with different safety requirements (freedom from interference). This separation can also enable the use of ECU SW stacks from different vendors, that can be independently developed, tested, updated and certified.

Figure 3 shows distributed ECUs communicating via a real bus that have been consolidated on a single ECU. Each of the ECU SW stacks (distributed ECUs) resides in a virtual machine (VM) that is executed on top of a hypervisor which runs bare-metal on the underlying hardware. ECU SW stacks use drivers to access virtual devices provided by the hypervisor – virtual devices may have real and even shared counterparts (see dashed line). Communication between VMs (inter-VM) can be realized by using a virtual bus, which may have been a real bus in the distributed E/E architecture.

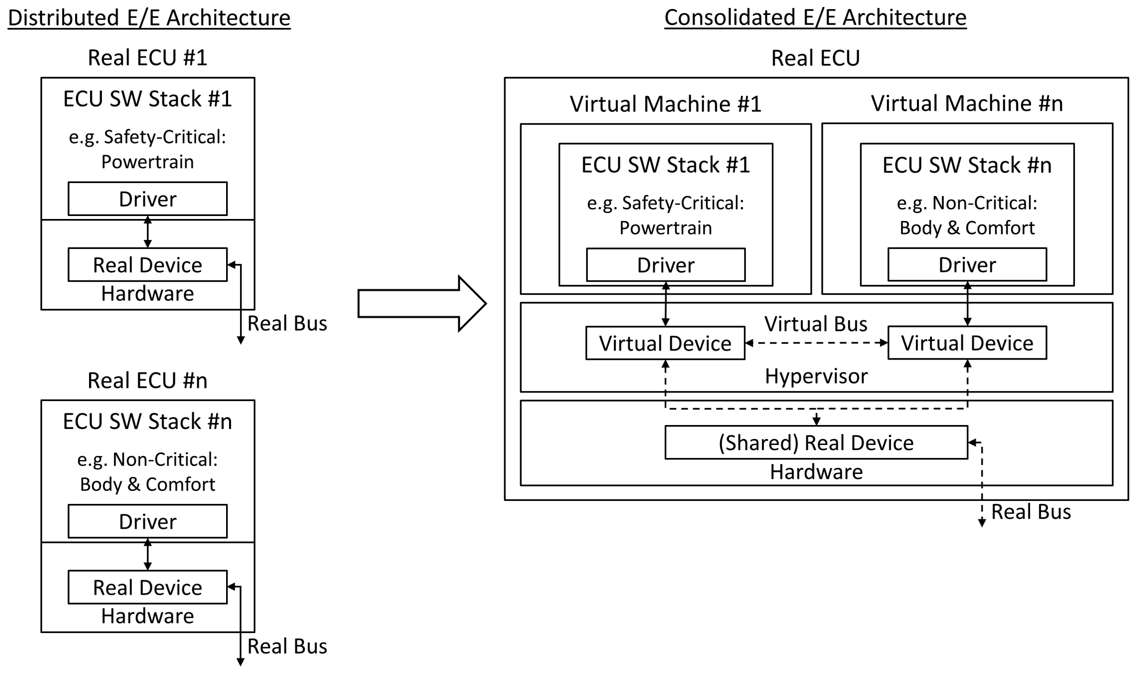

Figure 3: ECU consolidation: Execution of multiple ECU SW stacks on a single ECU, enabled by a hypervisor.

Virtual I/O Device (VIRTIO) is a promising standard that allows access to virtual devices via standardized interfaces. VIRTIO postulates paravirtualization, which comes with the requirement that the guest (see virtual machines in Figure 3) software must be prepared for virtualization and is aware of being virtualized. On the other hand, paravirtualization is becoming increasingly popular. One of the reasons for this is the higher performance compared to classic virtualization approaches, such as those described by [PG74]. VIRTIO was initially driven by Rusty Russell [Rus08] and is standardized by the Organization for the Advancement of Structured Information Standards (OASIS).

Today, VIRTIO can be considered the de facto standard for paravirtualization, which is also becoming increasingly important in the automotive industry (see e.g. SOAFEE[1] [Spe21]). Linux, which is also attracting more and more attention in the automotive industry, but also infotainment operating systems, such as Android Automotive OS [OS22], and real-time operating systems (RTOS) already support VIRTIO.

Figure 4 show the high-level architecture of VIRTIO which consists of a frontend (FE) and backend (BE) driver as well as a device. The communication between the VIRTIO FE and BE driver is based on so called virtqueues – a virtqueue is basically a queue of buffers [Rus08]. For example, a VIRTIO network driver (VIRTIO-net driver) can use one virtqueue for sending and one for receiving Ethernet frames.

VIRTIO is not only relevant for ECU consolidation uses cases where a hypervisor is used, but also in hypervisor-less use cases. In the next section, the impact of both uses cases on vECU simulations is considered.

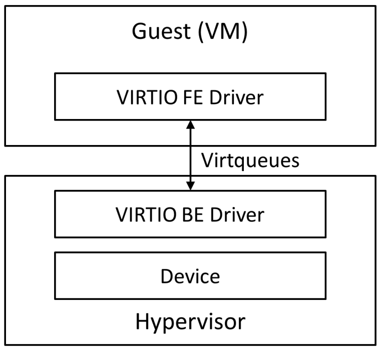

Figure 4: VIRTIO high-level architecture: Frontend (FE) and backend (BE) driver communicate via virtqueues.

3 ECU Consolidation and Virtual ECU Simulation

Hypervisor Use Case

Figure 5 shows the simulation of an EB AUTOSAR Classic and Adaptive ECU SW stack (EB corbos Adaptive Core and EB corbos Linux) with Synopsys Silver. As shown in Figure 3, it is assumed that both software stacks have been consolidated on a single ECU to be executed on a hypervisor in the real vehicle and that the real bus communication (here Ethernet) of the distributed setup is replaced by virtual bus communication provided by the hypervisor. By using a hypervisor, the ECU SW stacks can be decoupled from the underlying hardware. On the other hand, VIRTIO enables the decoupling between ECU SW stacks and the underlying hypervisor. In this concrete use case, the VIRTIO-net FE driver is provided by EB and part of the ECU SW stacks that can be executed in the real vehicle. For simulation purposes, Synopsys Silver provides the virtual Ethernet devices as well as the VIRTIO-net BE driver. Here, a hypervisor (on top of Windows® / Linux®) is used to simulate EB corbos Adaptive Core and EB corbos Linux with Synopsys Silver.

[1]EB and Synopsys are members of the Scalable Open Architecture for Embedded Edge (SOAFEE) project.

In comparison to VIRTIO, the AUTOSAR MCAL is typically provided by the semiconductor vendor of the microcontroller and hardware-dependent (see section 1). In situations where an AUTOSAR Classic stack is executed on top of a hypervisor, either the hypervisor has to provide a hypervisor-dependent interface (hypercall interface) that can be used in the AUTOSAR MCAL of the corresponding ECU SW stack, or the AUTOSAR MCAL has to be based on VIRTIO. EB, for example, provides a VIRTIO driver for AUTOSAR Classic that can be used in this use case.

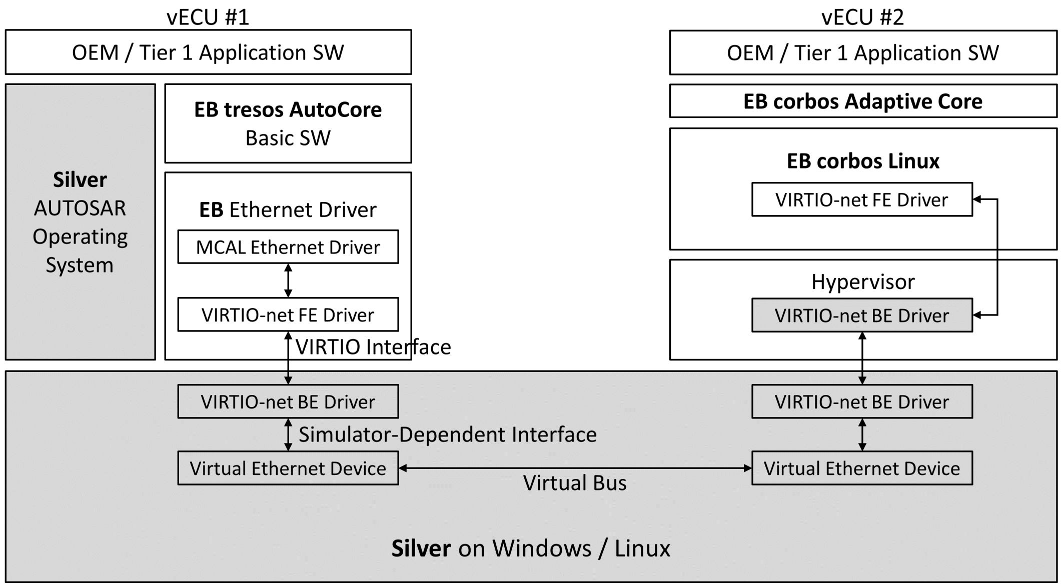

Figure 5: ECU consolidation: Simulation of an EB AUTOSAR Classic and Adaptive ECU SW stack with Synopsys Silver.

Hypervisor-Less Use Case

Open Asymmetric Multi-Processing (OpenAMP) [Ope22] and RPMsg-Lite [RPM21] are two frameworks that can be used for shared memory communication on multicore ECUs. Even though the shared memory communication can be done via a hypervisor, it is not required. OpenAMP and RPMsg-Lite are both using VIRTIO as an interface to hardware- or hypervisor-dependent software. For example, Figure 6 shows that both technologies use the Remote Processor Messaging (RPMsg) framework on top of a VIRTIO FE driver. RPMsg [RpM22] provides an interface that can be used by applications to send and receive messages between different ECU SW stacks via communication channels realized in shared memory. AUTOSAR Classic does not standardize shared memory communication, but offers the concept of complex device drivers (CDDs). For example, a CDD makes it possible to integrate special software that is not specified in AUTOSAR – OpenAMP and RPMsg-Lite are just two examples here.

Figure 7 shows how two ECU SW stacks that share memory in the real vehicle can be simulated with Synopsys Silver. Again, virtual devices are provided by Synopsys Silver to enable the communication between the ECU SW stacks – in this case the shared memory communication of the real ECUs is realized by the virtual bus provided by Synopsys Silver. Also in this use case, a hypervisor is used to simulate the Linux ECU SW stack with Synopsys Silver.

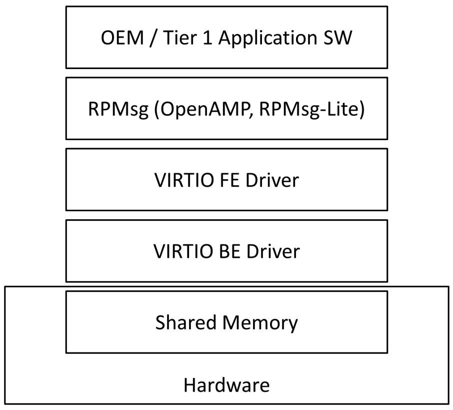

Figure 6: OpenAMP / RPMsgLite high-level architecture: Hardware-independent shared memory communication enabled by VIRTIO (FE driver).

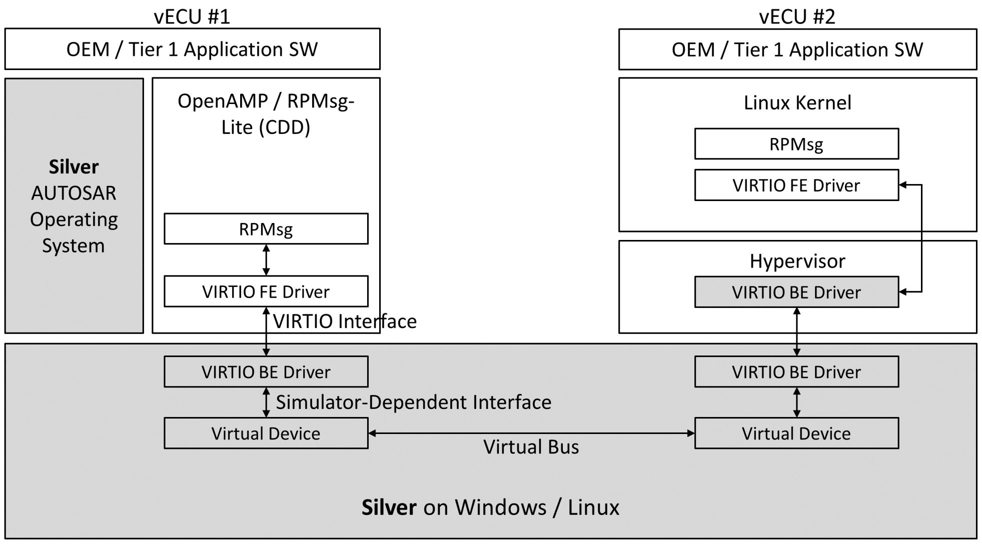

Figure 7: ECU consolidation: Simulation of RPMsg communication between a RTOS (here AUTOSAR) and Linux with Synopsys Silver.

4 Summary and Outlook

This work shows that VIRTIO is an important standard for future E/E architectures. When ECU consolidation is performed in a hypervisor use case, VIRTIO enables the decoupling of the ECU SW stack from the underlying hypervisor. However, VIRTIO is also used in use cases where ECU SW stacks are consolidated without using a hypervisor. In both cases, VIRTIO provides a standardized interface and thus creates the basis for the simulation of vECUs. VIRTIO can offer the possibility to run all software layers of the production ECU SW stack in simulations without using a simulator-dependent interface (see e.g. Figure 5). Depending on the concrete use case, Synopsys Silver provides for the simulation of consolidated ECU SW stacks virtual devices / buses (e.g. Ethernet), AUTOSAR MCAL drivers as well as VIRTIO BE drivers. EB provides AUTOSAR Classic and Adaptive ECU SW stacks including VIRTIO FE drivers.

In the future, VIRTIO will continue to gain relevance for the automotive industry and will be further developed (see e.g. [Spe21]). At the time of writing (June 2022), the specification for VIRTIO-can (Controller Area Network) was under review [MH21] and a CAN device ID was already reserved in the draft VIRTIO 1.2 committee specification [OAS22].

References

[IVI20] Prostep IVIP. Smart Systems Engineering – Requirements for the Standardization of Virtual Electronic Control Units (V-ECUs). Prostep IVIP White Paper 2020-3 / V 1.0, 2020. Accessed 31-May-2022: `https://www.prostep.org/fileadmin/downloads/WhitePaper_V-ECU_2020_05_04-EN.pdf`.

[MH21] Harald Mommer and Stefan Hajnoczi. RFC: virtio-can: Add the device specification, 2021. Accessed 27-Jun-2022: `https://markmail.org/thread/hdxj35fsthypllkt`.

[OAS22] OASIS Open. Virtual I/O Device (VIRTIO) Version 1.2. Oasis committee specification draft 01, 09 May 2022. Edited by Michael S. Tsirkin and Cornelia Huck. `https://docs.oasis-open.org/virtio/virtio/v1.2/csd01/virtio-v1.2-csd01.html`. Latest stage: `https://docs.oasis-open.org/virtio/virtio/v1.2/virtio-v1.2.html`.

[Ope22] OpenAMP (Open Asymmetric Multi-Processing), 2022. Accessed 27-Jun-2022: `https://www.openampproject.org`.

[OS22] Android Automotive OS. Virtualization Overview, 2022. Accessed 27-Jun-2022: `https://source.android.com/devices/automotive/virtualization`.

[PG74] Gerald J. Popek and Robert P. Goldberg. Formal Requirements for Virtualizable Third Generation Architectures. *Commun. ACM*, 17(7):412–421, 1974.

[RPM21] RPMsg-Lite User's Guide Rev. 3.2.0 / 11.0, 2021. Accessed 27-Jun-2022: `https://nxpmicro.github.io/rpmsg-lite`.

[RpM22] Remote Processor Messaging (rpmsg) Framework, 2022. Accessed 27-Jun-2022: `https://docs.kernel.org/staging/rpmsg.html`.

[Rus08] Rusty Russell. Virtio: Towards a de-Facto Standard for Virtual I/O Devices. *SIGOPS Oper. Syst. Rev.*, 42(5):95–103, 2008.

[Spe21] Matt Spencer. How the SOAFEE Architecture Brings A Cloud-Native Approach To Mixed Critical Automotive Systems. Arm white paper, September 2021. Accessed 27-June-2022: `https://armkeil.blob.core.windows.net/developer/Files/pdf/white-paper/arm-scalable-open-architecture-for-embedded-edge-soafee.pdf`.