Review of content/outline for the early draft of HV whitepaper

A copy of the first draft text is included here, and a blue summary of most paragraphs.
The purpose of that is to discuss the general structure/content of the document on a higher level.

(and then to discuss the details of each paragraph)

Title: What HV technology can do for future automotive systems.

Motivation: Why to use HV:

Opinions on the high level purpose of the paper.

...We need to explain why virtualization is actually needed. (It is still not fully accepted as necessary by all)
Certain concrete security/safety issues that can be shown clearly and that HV can solve
System flexibility is another very important point.
*

Idea: There could of course be multiple Whitepapers, if we want to concentrate on a certain area, and avoid others.

Interaction between general-purpose and dedicated cores is poorly understood.

Today, we see the advent of multicore system-on-chip (SoC), originally design for the mass-market of consumer electronics, entering the critical infrastructure of cars and trucks. Buzzwords like cell-phone on wheels have been coined. However, this is only the beginning, in the near future we will see the advent of central compute platforms, i.e., massive multicore SoCs thing over not only fundamental functions in the vehicle but also its control. This is a game changer for the SW stacks we use in our cars, including the underlying operating systems.

SUMMARY: More cores in SoCs changes the SW stacks

Traditional automotive multicore SoCs put a clear focus on low-power, low temperature, deterministic timings and high reliability, to only mention the major characteristics. In contrast, the aforementioned consumer-electronic SoC designs are sacrifice reliability, low-power and low thermal dissipation for high compute bandwidth.

SUMMARY: Optimize for (average) performance, not automotive requirements

Feedback: There is more diversity than this chapter suggests. Some may choose a more consumer-like processors but there are also modern multi-core processors that are automotive grade some vendors create massive compute power but with high power needs (heat), others may be better at keeping low power.

(same, worded differently) Some cars choose more consumer-oriented hardware, but others choose are strict on using automotive grade hardware. Some vendors provide very high performance but with high power consumptions, others can create fairly good performance with still low power… Conclusion: We may need to describe this as more varied, but then to propose we need to solve as many as possible of these variations (i.e. also solve for the "worst-case")

Moreover, deterministic timing for guaranteeing low service latencies even in worst-case scenarios is traded with service strategies which optimize the average ant not the worst case. An example to this is as follows: in a manycore system with private L1 and a shared L2 cache, SW executing on different cores will mutually evict each other's cache entries.

SUMMARY: Optimizing for average performance, not real-time. Example: shared caches

This in turn will significantly add to their execution times, data and instructions must be re-fetched from the main memory, where modern on core pre-fetchers intend to lower the waiting times from core-perspective. Cache eviction is, however, not the only source of trouble. Each time when fetching an item from the main memory, the execution on a core is suspended until the actual memory fetch has been served. The resulting waiting time depends on the number of pending memory access requests from all the other cores and the complex memory access pattern implemented by modern DRAM controller. A common pattern prioritizes hits into the open row buffer, instead of serving memory read requests in a standard first-in-first-out manner. With a strategy for increasing the open row buffer hit rate, one allows memory read request to overtake each other inside the DRAM controller. Whilst this lowers the average service time as reads from an open memory bank row can be served faster, this may contribute to the waiting time of some other read.

SUMMARY: Details on the cache contention problem.

Feedback: This info is very good but too many details come too early. This is better moved to a later chapter

Obviously, there is no free lunch and the increase in compute bandwidth comes a long with a significant increase in the complexity of the behavior of SW executing on such modern multi-core SoCs, put together with higher energy consumptions and higher thermal dissipations as these SoCs run on much higher frequencies. This clearly points to the question, **what can we do with these processors as they are not only more powerful, but also much more costly?**

There are two main promises made:

1. The use of multicore and higher compute powers allows one to integrate multiple functions, resp. independent SW stacks into a single electronic control unit (ECUs) and thereby reducing the number of onboard ECUs and their cablings. Commonly, the different SW stacks come along with their different operating systems, ranging from tiny footprint operation systems like FreeRTOS up to monolithic giants like Android to be used in modern head units.
   SUMMARY: Increased consolidation

2. Deployment of new applications which are solely possible by having not only multiple compute cores available, but also so-called hardware accelerators. Examples to such accelerators are image processing units and graphic processing units, today, directly synthesized onto a SoC. Put this to an extreme, one sees chip designs, which are in fact dominated by the accelerator support, rather than the number of general-purpose cores put onto the chip. Commonly, HW vendor provide support by Linux drivers rather than making HW drivers available as part of an AUTOSAR MCAL.

   SUMMARY: Hardware enables new compute-intensive functions

With both cases, one can spot the following communalities:

1. SW is executing in parallel and may try to access obviously shared resources like accelerators and network interfaces at the same time.
2. Contention on the use of a processor's infrastructure like memory busses and memory mapped registers can easily result in unwanted side-effects, e.g., process data arrives too late for the aforementioned reasons or data even becomes inconsistent due to read/writer races.
3. New and legacy applications to be put together on the same chip come along with their own operating systems, let the latter be highly specialized or of a general purpose nature.

To successfully deploy different SW stacks and their operating systems on a single SoC either calls for the use of hand-crafted mechanisms and protocols for sharing resources among the different operating systems or that one puts a supervisory SW-layer in place. It is up to this extra SW layer to orchestrate the use of the processor like any other operating system. Simply putting everything on top of a single operating system is infeasible, as it would require a porting of all of the legacy applications to the target operating system as far as possible or requires to re-implement hardware drivers for the target platform and the chosen operating system.

SUMMARY: The challenges with these new systems include shared use of singe-use features, and bus contention, etc.

"as it would require a porting" section - We could expand this a bit more. There is a list of different reasons for virtualization – (the ability to reuse existing software with minimal changes - (instead of porting as mentioned here) and another one is to reuse popular OS/environments that would be a very large job to recreate from scratch, e.g. Android)

Still, there is a difference between standard operating systems and such a supervisory SW-layer. The supervisory SW layer, also commonly denoted as hypervisor requires execution rights at a higher level of privileges as the operating systems running on top of it. This is as with any operating system, the latter executes at a higher level of privileges as its userspace applications. This is to execute privileged instructions, i.e, instructions which change the state of the processor or to restore the context of a shared resource whenever the latter is handed over to a different user, resp. application.

SUMMARY: A hypervisor is required as the solution (?)

1. Use of legacy systems with minor modifications,
   a. address what kind of modifications we expect,

1. What does the HW(vendor) to support platform virtualization

Also address problem of open source firmware and driver (MCAL) qualification when running virtualized drivers (see also section 3). HV helps with this by

- isolating critical devices from non-critical ones, allows one to build systems with mixed-criticality w.r.t. safety-relevance.
- qualified driver needs to come with the driver host or even the HV provider

SUMMARY: Hardware support for virtualization is included in modern processors

Kai, Adam & Bernhard

This is directed towards the HV vendor to avoid the problems we have seen in the past.

Feedback: Is the above a comment, or part of the whitepaper text? If it is the rationale for why the chapter exists, then let's rewrite and expand it.

Content: All modern processors, including Arm Cortex-A's and Intel's x86, support the virtualization of operating systems by means of providing adequate functionality for providing a virtual view of the system and having system software, the hypervisor, have full control of guest operating systems. A microkernel can offer support for this functionality. As already described, the microkernel will only offer the necessary functionality and all other support for running VMs shall be implemented in user-level functionality. For supporting virtualization extensions of the CPU, the microkernel provides the functionality to create VM containers and context switch those between other VMs and normal programs on the microkernel. The virtual platform, that is required to run a guest operating system, is provided by user-level virtual machine monitor (VMM). A common design pattern is to use one VMM per VM, using the isolation features of the microkernel to protect VMs among each other.

SUMMARY: This paragraph speaks a lot about microkernel / HV / software layer also, and only a small part about actual Hardware features?

Dmitry mentions i.mx 8 has special features that simplify device sharing/assignment to VMs, e.g. USB that could be interesting case-study information. Details pending (make sure to check what is public information first).

Feedback: "in user-level functionality" is this emulation code in the HV or what is meant byzk user level? Can we clarify?

1. ~~Surveillance,~~ Monitoring, Isolation (Timing and Spatial) and all that

To establish well-defined behavior of SW at platform-level several design paradigms can be followed, where each prioritize different aspects, e.g., fault-detection versus information hiding, high-performance vs. good worst-case timing behavior. At the bottom-line it appears that one of the fundamental principles of establishing safe and secure execution environments is about isolation and surveillance.

Sharing of HW in the presence of parallel system executions

Isolation properties in the presence of parallel systems executions:

Spatial isolation, hiding of secrets

Temporal isolation, implicit and explicit shared resources…

Also include use of special purpose Guest/OS for isolating a specific functionality, i.e, building safety and security island

Kai & Adam

1. Inter-core communication

Matti, Dimitri

Ideas:

Split into two main tracks. There are cores with direct communication and those that have inter-core links.
...comes down to what can be communicated in an atomic manner.
The size of the mailbox is one atomic unit. Other links are serial, so the size of atomic unit is essentially one bit only.
Reference MCU-style hypervisors.

Shared memory is also a communication method...
– Discuss (lack of) atomicity, buffer sealing etc.
– Discuss cache coherency and other complications...
Atomic operations e.g. ARM Read-and-exchange, Check-and-set etc. Those are only guaranteed on some memory types. Undefined behavior on some areas which include caches. Caches need to be coherent (have the same value).
Can be undefined or even undocumented.

**Inter-processor-interrupt IPI used to trigger the reading of a data item.*

Hardware mailboxes are usually a word or a few words. Once written and interrupt triggered, the value is locked down and not changeable (by the original writer) – avoid "time of check, time of use" type bugs and vulnerabilities.

How do different processor architectures provide features for this ? ARM, X86... MIPS...?

New interconnects may guarantee some of the coherency requirements CCIX, OpenCAPI...is dead.... NVLINK, also some things in PCI Express 4.0? CXL (Intel, based on PCI)

Open Asymmetric Multiprocessing - OpenAMP - messaging standards built on top of this... Often the implementation uses the hardware capabilities for mailboxes/links etc.

Cache locking?

New ARM designs/solutions need to be considered - ARM engineers can help.

Sharing Devices (and VIRTIO)

Methods and implications

Artem wants to cover Performance issues on device sharing. General, not VIRTIO specific.

Maybe first chapter is rather named paravirtualization techniques?

and then VIRTIO is a sub chapter. Or its own chapter.

Kai & Gunnar

Content:

As already outlined, the VMM component in the system needs to provide a virtual platform to the guest operating system. This includes a set of common devices that a VM typically needs, including console, network and block devices. When an OS runs bare-metal on the hardware, a regular driver is used to drive, for example, the network interface controller (NIC). The interface between the NIC and the OS is based on memory-mapped IO which is the optimal way of communicating with a physical device. When providing a VM a virtual device, the memory-mapped approach is not the most efficient one. First, because this requires the so-called trap-and-emulate technique where each access to the MMIO-region is trapped into the VMM and second because the NIC needs to be emulated by the VMM which is actually more complicated than required for this use-case. It is much easier in terms of required software as well as offers more performance when using a device for a VM that is particularly made for being used in Vms.

SUMMARY: Need for virtual platform. Shortly compare full hardware virtualization/emulation with… not doing that.

Discussion (Adam): Agree, VIRTIO kind of assumes paravirt. for devices. Memory/CPU side is similar - either hardware has support or not.
That should be described separately from device handling, because paravirt means something different there.

Fortunately, there is a standard for this, VirtIO, which all current operating systems, including Linux, offer support for.

With development going back over 10 years the VirtIO device model was first introduced for the educational "lguest" hypervisor and became the default I/O virtualization method used with QEMU/KVM. The VirtIO devices have been partially standardized by the OASIS standardization body in 2015 with the VIRTIO 1.0 specification which describes the transport layer and a limited set of device models. The currently standardized device models are: network, block, console, entropy, memory ballooning and SCSI devices. In addition, several devices exist as "implemented" devices, e.g., GPU, input, 9pfs, vsock and crypto.

SUMMARY: The current contents of VIRTIO spec

VirtIO relies on a DMA-like memory model meaning that all allocations are handled by the "driver" part of the device and the "device" implementation can directly access the allocated buffers for reading or writing. This enables a resource saving and fast operating mode inside the guest-OS. Metadata is transported using so called virt-queues that resemble ring-buffers. Depending on the architecture used, different transport and device discovery modes are supported: PCI for x86 and MMIO (memory mapped input/output) for ARM.

Having a standardized device model allows for multiple compatible implementations of both driver and device parts of the system. Thereby allowing customers to migrate their software stacks from one hypervisor to the other. The availability of VirtIO in many major operating systems keeps driver implementation and maintenance effort to a minimum and shall work in the favor of porting software stacks from one platform to the other. Due to the split into a standardized frontend in the VM and a customized driver running outside the guest-OS, resource usage and security properties can be implemented without impeding the standardized model and stay inter-operable.

Many vendors of automotive-grade hypervisors have already adopted VirtIO-based devices into their system offerings due to the reasons mentioned above. This will improve the availability of commodity devices like network and block storage inside cars.

When using hypervisor technology data handling needs to adhere to high-level security and safety requirements such as isolation and access restrictions. VirtIO and its layer provide the infrastructure for sharing devices. However, the handling of the data has to be implemented somewhere. Here, the use of a dedicated hypervisor service to do so appears promising and well aligns with the microkernel, capability-based design. An illustration to such a setup is provided in the figure above where we sketch the setup of a shared flash device. Once again, the movement of data around the memory prior to sending it to the device might be intended or can be avoided by using references in buffer descriptors inside the hypervisor service instead.

SUMMARY: Example (flash device) to show how security and safety is achieved, using the architecture suggested by VIRTIO

The hypervisor service executes the handling of data, e.g., send and receive from and to the device. It implicitly guarantees that data is shielded from other partitions. Moreover, the use of a dedicated hypervisor service allows one to impose usage quotas over the timeline and thereby eliminate DoS attacks and improve the predictability in the timing behavior as interferences of co-running subsystems are strictly bounded. Examples to such budgeting schemes are periodic servers which either give fixed time slices to each VM or bound the maximum amount of data to be written or read during a periodically repeating interval. In the context of networking, traffic shaping algorithms are well known and have been studied in the past. Leaky-bucket based Quality-of-Service networking is an example to this, where efficient implementations based on dynamic counters have been proposed.

SUMMARY: Example of HV added features, e.g. monitoring and enforced quota on resources.

Hypervisor-rooted device sharing also allows the provision of dedicated mechanisms, e.g., when copying a network packet from the receive queue to the receive buffer inside a VM, the hypervisor may ensure end-to-end protection. When using VirtIO-based device sharing, such mechanism can be implemented close to the driver, rather than using a coleoptera of methods inside the subsystems and re-implementing the Virtio-drivers for the hosted guest-OS. Therefore, it can be believed, that VirtIO does not only work for easy system integration, but also to improve security, timing predictability and safety within the resulting systems of systems.

SUMMARY: More unique features possible with an HV

Security implications

Artem has some objections to VIRTIO from security point of view. It could be included inside this text or as a sub-chapter of this chapter.

ALL SUMMARIES TOGETHER (AS OUTLINE)

This replaces each paragraph/chapter with its summary to get an overview.
We can then see if the content follows the desired structure, we can evaluate if some things need to move to a different place, and maybe also see also what content is missing.

TITLE: What HV technology can do for future automotive systems.

Motivation: Why to use HV:

SUMMARY: More cores in SoCs changes the SW stacks

SUMMARY: Optimize for (average) performance, not automotive requirements

SUMMARY: Optimizing for average performance, not real-time. Example: shared caches

SUMMARY: Details on the cache (contention) problem.

SUMMARY: What can we do with these new processors?

SUMMARY: Increased consolidation

SUMMARY: Hardware enables new compute-intensive functions

SUMMARY: The challenges with these new systems include shared use of singe-use features, and bus contention, etc.

SUMMARY: A hypervisor is required as the solution

SUMMARY: A HV must execute in higher privilege mode

= PURPOSE 1 of the paper.

Introduction of concepts

...

= PURPOSE 2 of the paper.

Surveillance, Isolation (Timing and Spatial) and all that

= PURPOSE 2 of the paper.

SUMMARY: Explain isolation, timing, spatial

Inter-core communication

Matti, Dimitri to write (Dmitry suggested that Matti does it better)

= PURPOSE 2 / 3 of the paper.

What does the HW(vendor) to support platform virtualization

(NOTE: Reordered chapters)

SUMMARY: Hardware support for virtualization *is* included in modern processors

= PURPOSE 3 of the paper.

Sharing Devices -- Virtio

SUMMARY: The need for virtual platform. Shortly compare full hardware virtualization/emulation with… not doing that.

SUMMARY: The current contents of VIRTIO spec

SUMMARY: More why standard platform is needed, and why VIRTIO model (i.e. para virtualized model?) is more performant

SUMMARY: Example (flash device) to show how security and safety is achieved, using the architecture suggested by VIRTIO

SUMMARY: Example of HV added features, e.g. monitoring and enforced quota on resources.

SUMMARY: More unique features possible with an HV

Microkernel, Monolithic

= PURPOSE 2 / 3 of the paper.

Additional work

(How to improve the future of virtualization usage)

Conclusions and wrapup

https://docs.google.com/document/d/18mTJw1DRqKmjIGoWcxFHZ8eodkWGGIEo/edit