What VirtIO can do for multi-OS integration in the vehicle

Coupling HVs and Classic AUTOSAR

Kai Lampka
Agenda

I. Introduction
II. Device sharing across partitions, here ETH AVB
III. Conclusion
On the road today

Past

- 70-150 distributed **single core** ECUs
- Each ECU features a small set of separated functions, e.g. window lifter...
- Classic ProOSEK, AUTOSAR OS
- Multiple deterministic network stacks in use (CAN, FlexRay, LIN, etc.)

Today

- More single and few multi-core ECUs running multiple functions
- High degree of predictability / hardware reliability
- Multicore version of **Classic** OSEK / AUTOSAR OS
- **No hardware virtualization support**

Upcoming

- **Performance-centric multi-core SoC** (x86, ARM, and GPU-architectures) as known from consumer electronics
- Networking via real-time Ethernet
Towards highly integrated architectures

Platform spectrum

Predictability- and reliability-oriented designs

Applications with highest timing & availability demands

Model-based control
- Few sensors and few actors
- Well understood control systems

Performance-oriented designs

Applications with highest computing demands

Complex decision models
- Many sensors & actors
- Not well understood algorithms
Prerequisites for tomorrow's in-car (composed) SW stacks

- Spatial isolation for correct behavior
- Temporal isolation for predictable behavior
- Minimal attack and failure surface
Conflict: SoC vs. goals

1) Unsecured software and unsecured devices
   • Flash devices are external devices, i.e. accessible with a proper set of tools
   • Execute-in-place (XiP) on Flash not necessarily possible, i.e. images need to be copied into SDRAM
   • Parallel executing software (when booting a software stack) on different cores (who is doing what and when?)

2) Resources which are obviously shared
   • SDRAM, cores, and GPUs
   • Ethernet controller, CAN controller
   • QSPI-Flash/eMMC-Flash/Hyperflash
   • Arm Trustzone/OpTEE (single threaded)

3) Resources which are implicitly shared among computing elements
   • Common SoC Infrastructure, e.g. AXI-Bus, Caches

4) Running third-party software potentially at privilege level of the OS (e.g. device drivers from third parties)
   • Different OS in a single SoC, e.g. Linux, Android, Classic AUTOSAR, QNX, etc. (who is the master, who is the servant?)
I. Introduction

II. Device sharing across partitions, here ETH AVB

III. Conclusion
### SW stack integration with (heterogeneous) multi-cores

#### AUTOSAR Adaptive foundation, services (examples), and applications

<table>
<thead>
<tr>
<th>Service</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ara::com</td>
<td>Communication management</td>
</tr>
<tr>
<td>ara::dm</td>
<td>Diagnostic management</td>
</tr>
<tr>
<td>ara::dt</td>
<td>Log &amp; trace</td>
</tr>
<tr>
<td>ara::em</td>
<td>Execution management</td>
</tr>
<tr>
<td>ara::hm</td>
<td>Platform health management</td>
</tr>
<tr>
<td>ara::pm</td>
<td>Persistency management</td>
</tr>
<tr>
<td>ara::sm</td>
<td>State management</td>
</tr>
</tbody>
</table>

#### Classic AUTOSAR services and applications

- **Platform health mmgt**
- **Power state management**
- **Com service**
- **Runtime environment (RTE)**
- **Fee**
- **AUTOSAR OS**

#### Physical devices

- Ethernet switch
- DMA controllers
- GPU
- SDRAM
- Flash devices

- **Hypervisor X**
- **Linux**
- **QNX**

- **Microcontroller abstraction layer**
Means of interaction in a nutshell

Several mechanisms

• Inside OS:
  – Inter process communication (IPC),
  – shared memory + notification
  – IRQs

• Node ↔ node: shared memory, notifications

Generic approach for VM interaction: VirtIO

(already discussed in the morning session)

• Official standard
• Wide availability of guest drivers (Linux, *BSD, Windows, ...)
• Used to share: Block devices, Network, GPUs .......
  – .....
What VirtIO can do for multi-OS integration in the vehicle

VirtIO—in action: Virtual networking

Default approach: VirtIO-net

- Common OSs have VirtIO-net drivers
- Host needs to provide VirtIO-net component

Optimized solution

- Driver included
Inside an example

What VirtIO can do for multi-OS integration in the vehicle

<table>
<thead>
<tr>
<th>Hypervisor kernel</th>
<th>Classic AUTOSAR application</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Boot manager</td>
</tr>
<tr>
<td></td>
<td>Boot loader</td>
</tr>
<tr>
<td></td>
<td>SafetyOS</td>
</tr>
<tr>
<td></td>
<td>(hosting ETH controller)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Userland VM1</th>
<th>Userland VM 2</th>
<th>Hypervisor applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel A</td>
<td>Kernel B</td>
<td></td>
</tr>
</tbody>
</table>

Access?
Hosting the device inside the HV partition

What VirtIO can do for multi-OS integration in the vehicle
What VirtIO can do for multi-OS integration in the vehicle

Hosting the device outside
What VirtIO can do for multi-OS integration in the vehicle

Agenda

I. Introduction
II. Device sharing across partitions, here ETH AVB
III. Conclusion
Conclusion

Benefits

• VirtIO device support is available in Linux, Android, and many other operating systems
• Builds upon the kernel user space interface of Linux and allows large flexibility, because the devices themselves make no assumption about the hardware
• Implement VirtIO-based devices that follow either existing standards or specify new ones

Most prominent use cases

• VirtIO Net
• VirtIO Block device
• VirtIO Console (character device)

VirtIO suitable for cross-partition device sharing.
Thank you.
Get in touch!