W H I T E P A P E R

# Software Defined Vehicles and the need for standardisation

**By Andrea Gallo**

September 2021

## Abstract

This white paper will look at the implications coming with the disruption driven by the Software Defined Vehicle momentum. From the lack of standard interfaces and latency to the need for secure communication between virtual machines and the ability to maintain up-to-date, secure software - all the challenges identified have one thing in common. They all require collaborative software engineering to bring standardization and reference open source code bases to automotive. Without open standards the path to delivering the software defined vehicle will be a lot longer and more costly.

Linaro has a track record of bringing standardization to the Linux kernel and is now extending this to automotive.

## Contents

BACK TO THE TOP

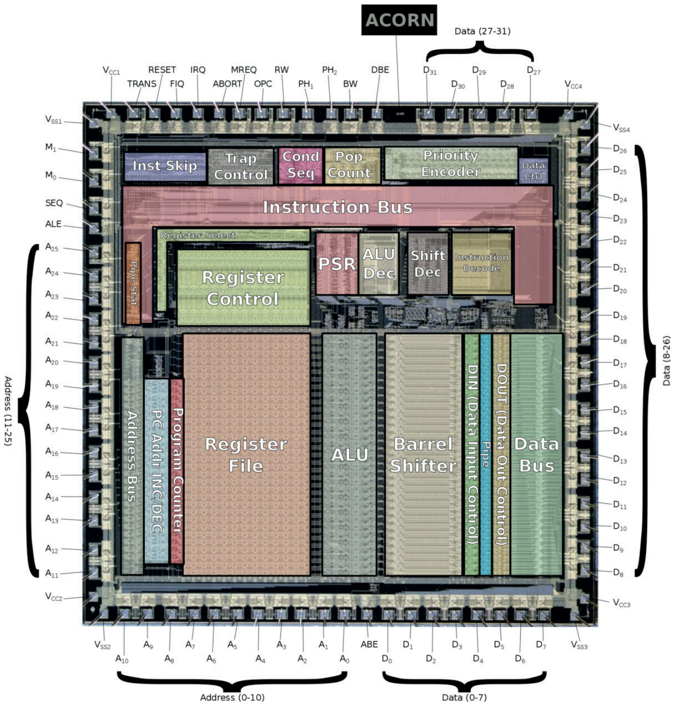
# The evolution of Software Defined Everything

Almost sixty years ago, in 1964 Piergiorgio Perotto invented and designed the Olivetti Programma P101, also known as the Perottina, the world-first electronic programmable desktop calculator in history. It also included a magnetic floppy card, the ancestor of the floppy disk. NASA purchased ten models to plan the Apollo 8 landing on the moon.

It featured registers, mathematical operations and even memory load/store and jump instructions -- all so similar to what we consider an assembly language nowadays!

The Perottina disrupted the existing market of electromechanical calculators, for it could be easily programmed to implement new functions without changing any mechanical parts. Not to mention that the electromechanical solutions had quite a low production yield and were quite expensive to manufacture. It was a revolution!

50 years ago in 1971 Federico Faggin designed the world's first microprocessor, the Intel 4004. It replaced the fixed function electronic designs with a programmable logic unit. It was a major revolution and the ancestor of the entire Intel x86 family of processors.



This year Arm Ltd celebrates their 30 year anniversary. It was actually 36 years ago that the first Arm1 chip was powered on at Acorn Computers.

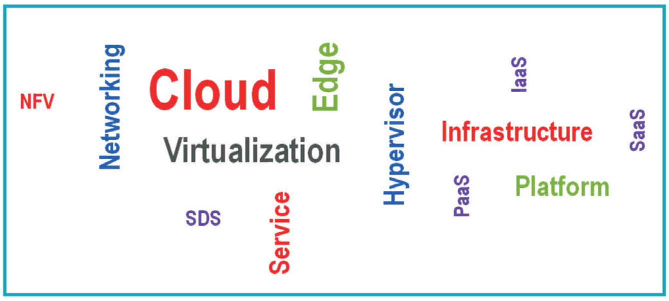
Twenty years ago the virtualization momentum started and eleven years ago in July 2010 NASA and Rackspace launched the open-source cloud-software OpenStack initiative.

Software has come to play a critical role in technology's evolution and has led us to where we are today - in the era of software defined everything.

Electromechanical calculators got replaced by programmable logic computers. Custom-designed telecommunication units have been replaced by Software Defined Networking (SDN) and Network Function Virtualization (NFV). Dedicated physically-located pools of hard drives, optical readers and magnetic tapes have been replaced by Software Defined Storage (SDS).

BACK TO THE TOP

Infrastructure as a Service (IaaS), Platform as a Service (PaaS), Software as a Service (SaaS) are nowadays common terms, as much as the new Car as a Service (CaaS) and Mobility as a Service (MaaS) terms are becoming familiar.



We live in a globalised world where consumers have become accustomed to always being online and connected. With IoT, VR, AI and edge computing, we are increasingly expecting our devices to intelligently communicate with one and another and be customizable. And while in many instances this is possible, there is one piece of technology which is critical to our lives which still has some way to go before it is truly part of our pool of connected devices - our cars. In order for our vehicles to become one of our connected devices we need to extend the software defined everything movement to automobiles.

The next technological disruption to address these needs is Software Defined Vehicles (SDV).

BACK TO THE TOP

# Automotive and software today

The concept of Software Defined Vehicles reflects the gradual transformation of  automobiles from highly electromechanical terminals to intelligent, expandable mobile electronic terminals that can be continuously upgraded.

Smartphones and computers transitioned from hardware upgrades to software development when they had reached the physical limits of what was possible. The automotive industry is in a similar transitional phase where in order to evolve our cars into connected devices, we need to rethink how we build cars and move towards software development.

Over the years, the amount of electronics in automobiles has steadily increased.

So far more and more functionalities have been added via fixed-function embedded electronics, also known as control units or engine control units (ECU). Each ECU is connected to a few sensors and actuators and manages a specific fixed function in the car. According to the IEEE, the number of ECUs in a premium car has increased by 50% in the last ten years, reaching a massive number of 150 ECUs in one single car! Consumers expect their cars to function similarly to their smartphone, as such the management of functionalities needs to become more centralised and software based - not only to leverage IoT but also to reduce manufacturing and maintenance costs. The current set up in many cars with more than 100 ECUs does not lend itself to the future connected car.

So what steps are currently being taken towards realizing the Software Defined Vehicle?

## Simplifying the management of ECUs - A single unified bus

Work is underway to simplify the management of ECUs. Traditionally, multiple ECUs have been connected together and have been communicating over a Controller Area Network bus (CANbus). The addition of distributed audio and video functionalities has increased the number of wires in a car with a direct impact on the overall car Bill of Materials (BoM). The effort to simplify the complexity and amount of wiring in a car and reduce its manufacturing cost has led to the definition and adoption of a single unified bus in the car, also known as Audio Video Bridging (AVB), Time Sensitive Networking (TSN) or Automotive Ethernet.

It is a single Ethernet-based bus shared by all ECUs in the car and it features real-time capabilities and pre-allocated guaranteed time slots, so that critical sensors or isochronous media pipes can timely transmit and receive their data as needed.

## Central automotive servers and multiple zonal gateways

The increased computing capabilities in recent microprocessors is driving the next change: multiple embedded fixed-function ECUs can be combined into one or two central automotive servers and multiple zonal gateways. Car makers can add new functionalities or improve existing features at will over the life of a car, as long as there are enough spare computing resources and the required hardware support is there. Combining multiple ECUs into one or two server CPUs provides

BACK TO THE TOP

economic savings coming from multiple angles: BoM, multiplicity, size and complexity of the circuit boards, test plans and test harness, etc.

## The head unit becomes customisable

The same trend applies to the head unit. Multiple independent instrumentation clusters, displays and gauges, are being merged into one single large central display, which provides virtually unlimited indicators. Multiple buttons, knobs and levels are being merged and transformed into icons and virtual buttons on a touchscreen display. The user interface of a car could be reconfigured and customised at will by the users.

This is the start of the Software Defined Vehicle (SDV) momentum!

# The transition to Software Defined Vehicles and the challenges of implementation

While there is work currently happening which signals the beginning of the Software Defined Vehicle transition, there are several problems that need to be solved and implications to be considered in order to realise this major shift.

## Vendor lock-in and the lack of standard interfaces

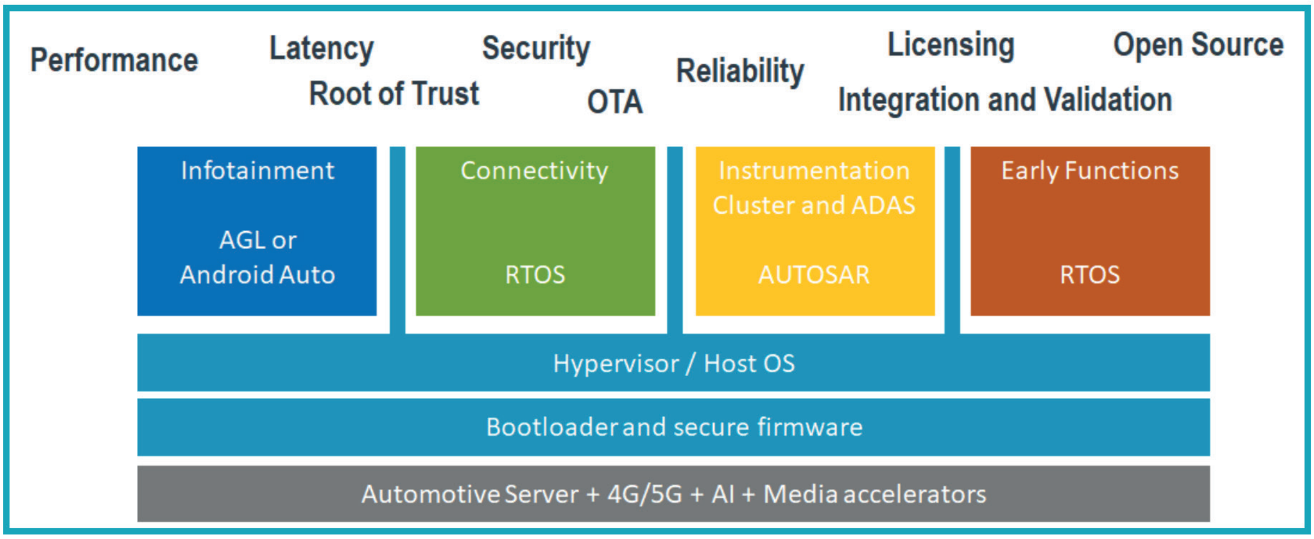
There is an immediate consequence of the trend to merge multiple ECUs into one or two server chips: the individual software code bases running on each ECU need to be ported to the new CPU and run altogether one next to the other. There are non negligible implications:

- Compatibility of different software code bases and runtimes
- Cross-component interference
- Disruption of safety critical functionalities
- Different security levels provided by different functions
- Overall cost of integration and maintenance

The de-facto industry solution is to adopt virtualization technologies. A hypervisor can provide well isolated virtualized environments, one for each ECU. Assuming that the hypervisor technology can emulate the legacy ECU as close as possible, each ECU code base can run almost unmodified in its own virtual machine and the resulting server platform may run a mix of:

- Safety certified OS and bare metal applications for safety critical functions - ADAS, engine control, instrumentation cluster, connectivity
- Real Time OS for early functions - remote cameras, audio subsystem
- High Level Rich OS (Linux, Android) for the main dashboard and user services - audio/video infotainment, navigation, etc.

It is important to note that adopting a hypervisor is not a cost free move, for when it comes to access to peripherals via device drivers in the VM, there is a tight link between the virtual device drivers and the hypervisor of choice. This is true for both type 1 and type 2 hypervisors.



BACK TO THE TOP

The embedded software running on an ECU requires some rework to be ported on top of the virtualized device drivers. The implication is that there may be a natural lock-in with the hypervisor technology of choice, if the hypervisor support is baked directly into the virtualized device drivers. It may become very expensive to switch hypervisors.

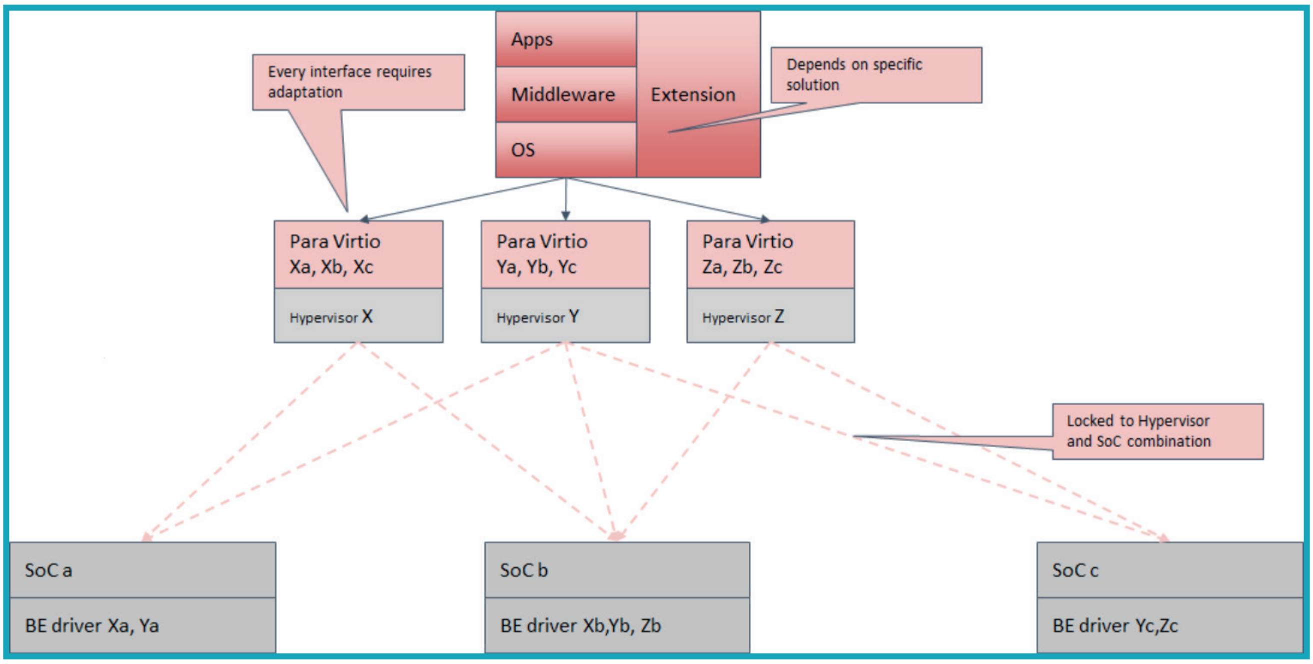
By having a dependency between the virtualized device drivers and the hypervisor, it forces a very early commitment to a SoC and Hypervisor choice in what is likely a multi year design and development effort. A given selected solution may be old technology by the time a vehicle gets close to the go-to market phase but there may not be an opportunity to migrate to a newer, better or more secure technology at that point.

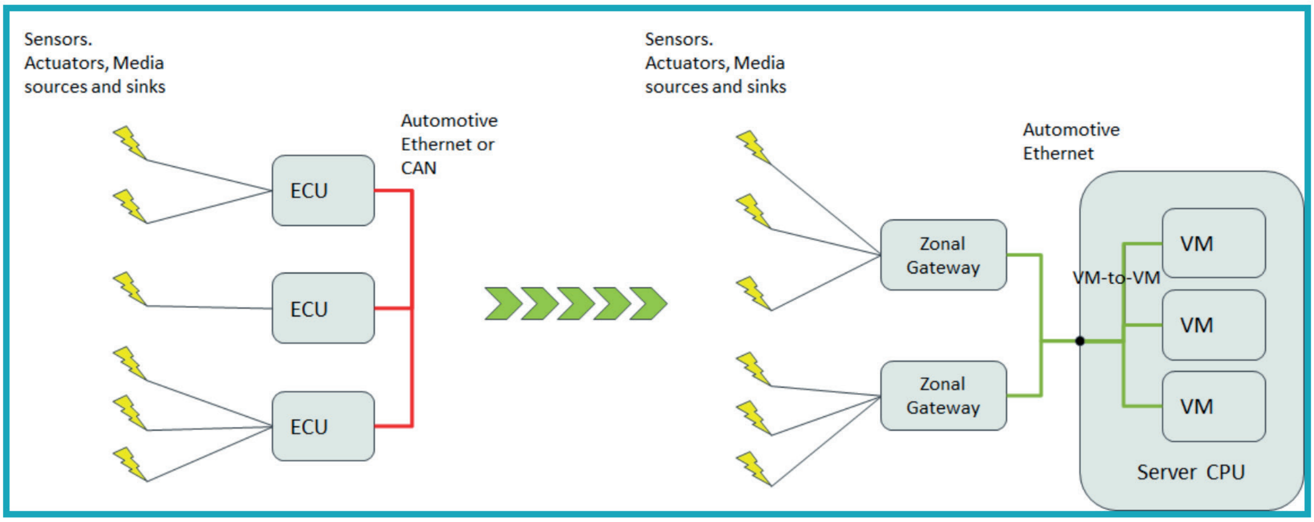
There is a need to define standard interfaces which would work with every hypervisor and which could be adopted to modify the ECU device drivers.

## Secure and Reliable VM-to-VM communication

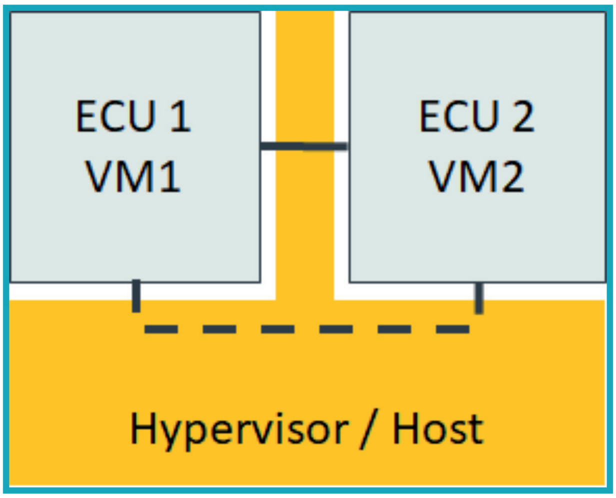
The second challenge is to adapt the CANbus or the Automotive Ethernet bus used by fixed-function embedded ECUs to communicate between them. The problem is to identify its corresponding software-based communication infrastructure across all VMs.

The natural choice is to use the virtualized LAN drivers in the VMs, so that the software originally




Sensors. Actuators, Media sources and sinks
ECU
ECU
ECU
Automotive Ethernet or CAN
Sensors. Actuators, Media sources and sinks
Automotive Ethernet
Zonal Gateway
Zonal Gateway
VM-to-VM
VM
VM
VM
Server CPU

developed for the ECUs would still communicate over an ethernet-compatible bus. The next step is to look at the underlying infrastructure at the host system level, which would be used to implement the backend for the virtual ethernet devices. This really depends on the host system. Is it a type 1 hypervisor or a type 2? Is it running Linux or an RTOS or something else? Is it the default "local loop" or something custom?



It is quite difficult if not risky to accept different implementations on different host platforms. There is a need to define one standard host VM-to-VM solution, which shall be reliable and secure.

In addition, use cases like bus recorder for the insurance companies should be considered, these may require a secure storage service and may dictate a specific implementation.

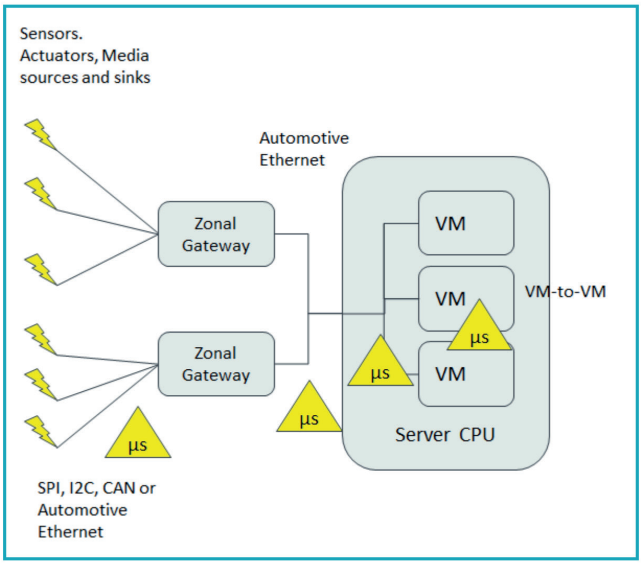
## Latency and the need for a defined fast path

In the traditional set up, fixed function ECUs are connected directly to the sensors, relays and actuators in their scope of operation. This ensures timely data transfer, be it for early functions or audio/video subsystems.

Once the ECUs are transferred into Virtual Machines, the same real time requirements need to be met in the new set up. The ethernet link is still used to connect the sensors and relays/actuators to the main server processor via the zonal gateways. Time Sensitive Networking and time slicing, bandwidth management, etc. are all still present but guaranteed only up to the physical connection to the server processor.

Once packets are delivered to the hardware buffers, the ultimate latency until the application software functions are executed depends on multiple software layers: the host OS and type 2 hypervisor or the bare metal hypervisor, the OS running in the VM, the application itself. As a result, the latency is out of one's control.

Building on top of the standard interfaces for virtualized devices, there is a need to define a fast path between the application running in the VM and the hardware FIFOs, bypassing the VM OS and the hypervisor.


Sensors. Actuators, Media sources and sinks
Automotive Ethernet
Zonal Gateway
Zonal Gateway
VM
VM
VM
VM-to-VM
µs
µs
µs
µs
Server CPU
SPI, I2C, CAN or Automotive Ethernet

## The challenges of continually updating software

The more software there is in a product and the more complex that software becomes, the more essential it becomes to define a software update strategy. The way software is developed is changing and it is edging closer and closer to continuous release and deployment mechanisms with new jobs like devops becoming familiar in the automotive industry.

### Over-the-Air Updates

Over-the-Air updates (OTA) can generally be split into two phases: the first phase takes care of the handshake protocol with the software provider and downloads the new software image, the second phase takes care of the flashing procedure. The former takes place as a runtime service while the latter usually requires a reboot of the platform. There are multiple implications to be considered.

First of all, there must be a fallback mechanism in case the new image is corrupted, not complete or the update process fails. The vehicle cannot just become unusable. It is not acceptable for smartphones or computers, it cannot even be conceived for a vehicle.

### Security and Trust

Then there is the security aspect. The whole process to identify, download and flash the update must be trusted. The vehicle must not be compromised. It is mandatory to have all signature verification procedures and best practices. Not to mention the requirement to ensure that the entire software is secure and trusted while the vehicle is powered on. Security, root of trust, applications for trust, non-repudiable logs, secure storage and system health monitoring.

A software defined vehicle can integrate components from many different suppliers, at a recent conference one car maker hinted at modules provided by as many as twenty different vendors! Integrating components from multiple suppliers introduces the potential for disrupting a component from another vendor. There is also the case of dependent components, a process also known as a transactional update. This is when one component is updated if, and only if, a given related component also gets updated successfully at the same time. There needs to be a rollback mechanism in case chained updates do not all complete as expected.

### Continuous Integration and Testing

Lastly continuous integration and testing needs to be considered. There are so many moving parts in the system, so many patches being released and tested in parallel - both during development as well as during maintenance. There is a need for an automatic system - and its infrastructure - to continuously build and test the entire system or each subsystem - depending on the software architecture - for every patch pull request.

# Achieving the Software Defined Vehicle through open source and common standards

The Arm ecosystem is well known for its very high pace of innovation, thanks to the business model created in 1992 and pursued since then by Arm Ltd. Multiple semiconductor manufacturers license the Arm microprocessor IP and its fundamental peripherals, they then add their own differentiating blocks and design their unique system-on-chip. The outcome is a significant large number of vendors competing in the same market segments.

While on one side this fosters innovation at an incredible speed, on the other side the implications are that software integrators shall adapt and port their software stacks to each new SoC. Often the base software development kits delivered by the semiconductor companies will be based on different firmware releases, different operating system releases, different toolchain versions, etc. The consequences are that applications or software stacks from software vendors and system integrators may not work the same on different platforms or may have different security levels. Maintenance over time, rebasing to new releases, porting fixes are all expensive and difficult to consistently apply to all supplier platforms.

Overall the total cost of ownership can be pretty significant if competing semiconductor companies design their SoCs without being aligned to the same set of well defined standards.

This is why Linaro was formed in 2010 - to bring companies together in the Arm ecosystem to work on common foundational software. Bringing engineering resources together to collaborate on common software projects reduces overall fragmentation, allowing Linaro's member companies to reduce their costs for development and validation of Arm-based software.

Linaro and its member companies are driving multiple projects that help realize the Software Defined Vehicle revolution through reference open source software.

## Reducing the hypervisor lock-in through Project Stratos

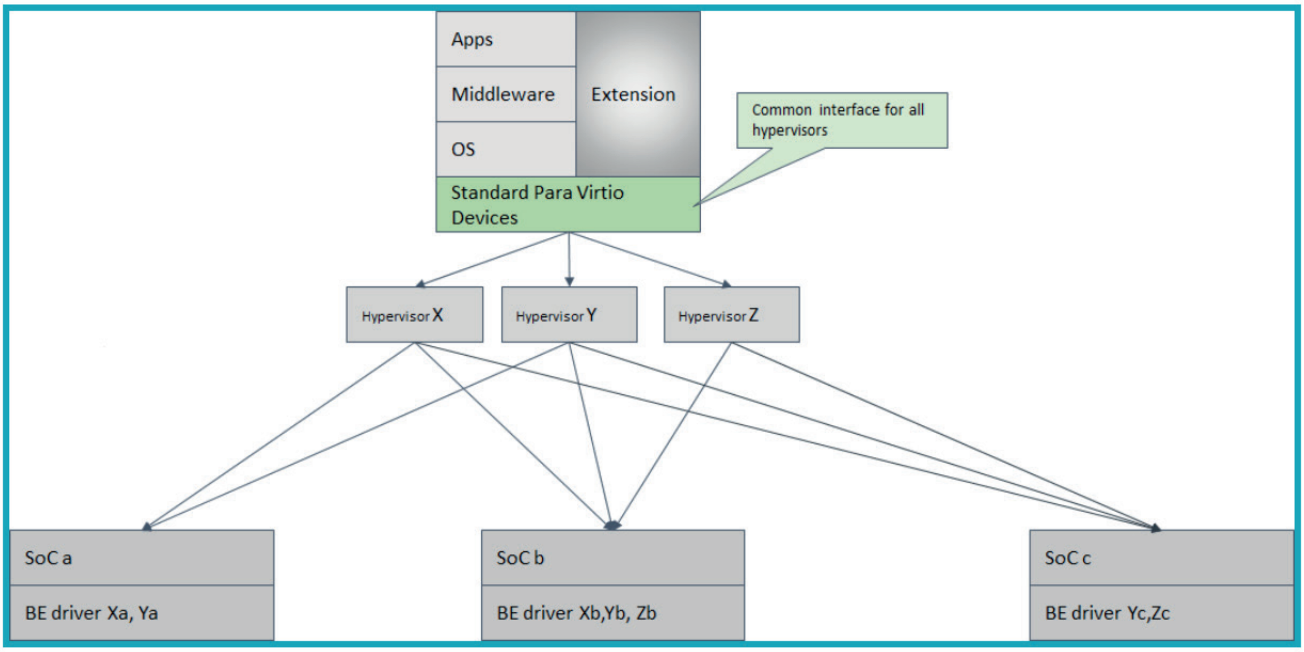
Project Stratos addresses the need for reducing the application lock-in inherent in the current mix of Hypervisor and SoC proprietary solutions. Its aim is to establish virtio interfaces complying with the OASIS Virtual I/O Device (VIRTIO) specification. Project Stratos also generates open-source implementations of the front and backends for that specification.

The first problem that needs to be addressed is the need for common frontend interfaces via the virtio standard. The diagram shows how having a common virtio interface allows the hypervisor to be transparent to the application. A previous generation hardware can now be used to start development on the next generation without as much cost to port to the new platform when it arrives. The selection of a solution can be opened to a wider market of SoCs and the decision made later in the vehicle's development. This ensures you end up with more up to date and secure software in your vehicles.

The second problem being actively addressed is the need to reduce the duplication of effort in implementing the backends for the standard interfaces. The majority of the functionality is the same and only a shim is required per hypervisor. There are obvious benefits to this common implementation of the backend.

First of all this enables the rapid proliferation of any new standard to all hypervisors that share the common backend.  Secondly, security is enhanced when there are more eyes and testing of the core functionality due to

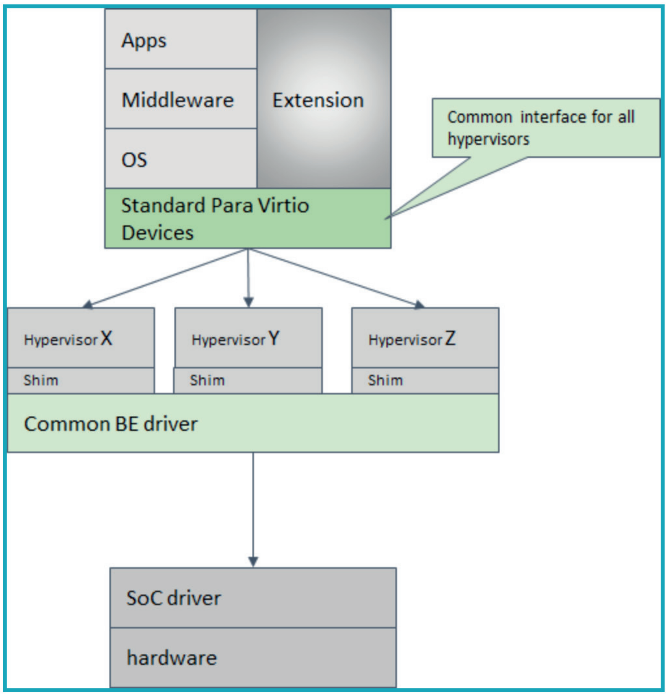
## Virtio as a common framework



it being developed once in collaboration. And finally, in this case the implementation of the common backend via the RustVMM brings its own assurances of memory management security and stability issues.

The diagram below highlights the simplification achieved by having the common backend driver and the reduced impact to the code.

## Common backend for a device



Current interfaces under development to be established as standards with front end and common backends implemented with the RustVMM are: - virtio-I2c, virtio-rpmb, virtio-gpio and virtio-scmi. Work is progressing to establish secure vm to vm multimedia via virtio-video and a "fat virtqueue" to address the issue when using virtio between guests and the global memory model breaks down.

### Automotive Grade Linux (AGL) Demo

As part of the bootstrap of the Stratos efforts, we worked with the AGL virtualization expert group (EG-VIRT) to demonstrate how virtio enabled the goal of generating hypervisor-independent interfaces.

The demo was a very simple PoC with two targets as a collaboration with the AGL image. In parallel Linaro member companies worked to replicate the environment on their physical hardware and uncover issues.

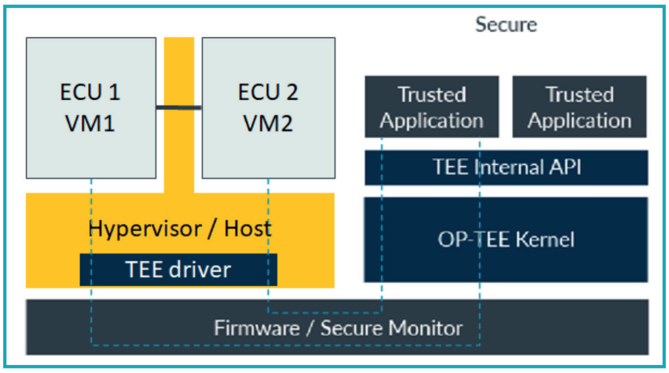
This work generated an expansion of QEMUs ability to support a guest loader in Xen, which supports collaboration where there is no common hardware platform for development.

## Secure and reliable communication through OP-TEE and Functional Safety

As noted in the previous section, each hypervisor can provide its own proprietary implementation of the VM-to-VM communication infrastructure.

In order to make this standard across all hypervisors, a common virtio device shall be specified and adopted. If needed, security can be added by leveraging the Trusted Execution Environment by providing some secure handshake between the VMs or even by providing a secure trusted application.

It is an area to be explored from multiple perspectives: functionality, requirements, performance and safety.



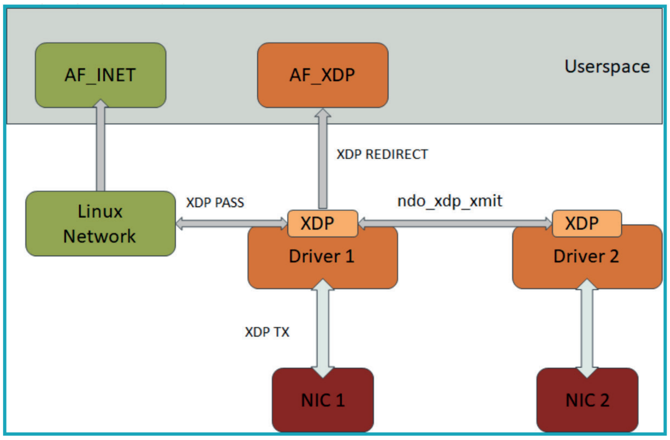
OP-TEE contains a full implementation to make a complete Trusted Execution Environment. It has been used in production on all sorts of devices like mobile phones, tablets, laptops and surveillance cameras for many years. All of this was made possible by Linaro in 2014 when OP-TEE became an open source project.

The last year or two car manufacturers, Tier 1 suppliers and even OEMs have started questioning whether OP-TEE fulfills any safety requirements, like ISO-26262. The answer to that currently is no. This has led to an investigation where the goal has been to understand what it would take to get OP-TEE safety ready (ASIL-B). In Linaro we've spent a couple of months looking at MISRA C, trying to understand use cases from the automotive industry by talking to car manufacturers one to one as well as hosting workshops. We now have a picture of what it would take to get OP-TEE safety ready. If the automotive industry is ready to collaborate, we believe we can repeat the success we've seen with OP-TEE in the areas mentioned above.

## Reducing latency through Time Sensitive Networking and Time Sensitive Applications

In an automotive car environment, the timely and reliable delivery of packets is of the utmost importance. However the best effort nature of the Ethernet protocol is not a good match. Moreover the traditional protection mechanisms for packet delivery (e.g TCP/IP) fail to meet the strict requirements of the automotive industry.

With the intention of building on time sensitive networking (TSN) and improving its upstream support, Linaro has engaged on the kernel related activities almost three years ago. We have contributed the initial TSN drivers as well as the general architecture in the kernel for configuring and managing TSN switches.



Although TSN can offer deterministic latency, there might be use cases where it needs to be less than 1µs. Measuring the kernel network stack yielded numbers way above that. Since the kernel has to go through a number of layers to deliver a packet to a userspace application, this slowly adds up to ~70µs. We adopted the AF_XDP in-kernel fast path, which can deliver packets to user space directly by-passing the slower kernel layers. Although we didn't meet the strict 1us

BACK TO THE TOP

timing, we demonstrated we could deliver packets up to the OPC-UA software layers in user space within ~4µs.

The next step is to deliver packets to the same software layers but running inside a VM. This implies measuring and coping for the added latency of the hypervisor and the guest operating system in addition to the host OS. The ongoing work in Project Stratos will lead the analysis and definition of the right virtio interfaces to enable the same fast path performances across the VM boundaries.

## Delivering the latest, most secure and validated software through Linaro's Trusted Substrate and Linaro's Linux Kernel Functional Test projects

Trusted Substrate is a collaborative project for the integrated, tested and packaged foundation of open source secure boot and trusted execution environments. The project brings standards based secure booting and over-the-air (OTA) updates to the most trust demanding embedded computing projects such as automotive and robotics.

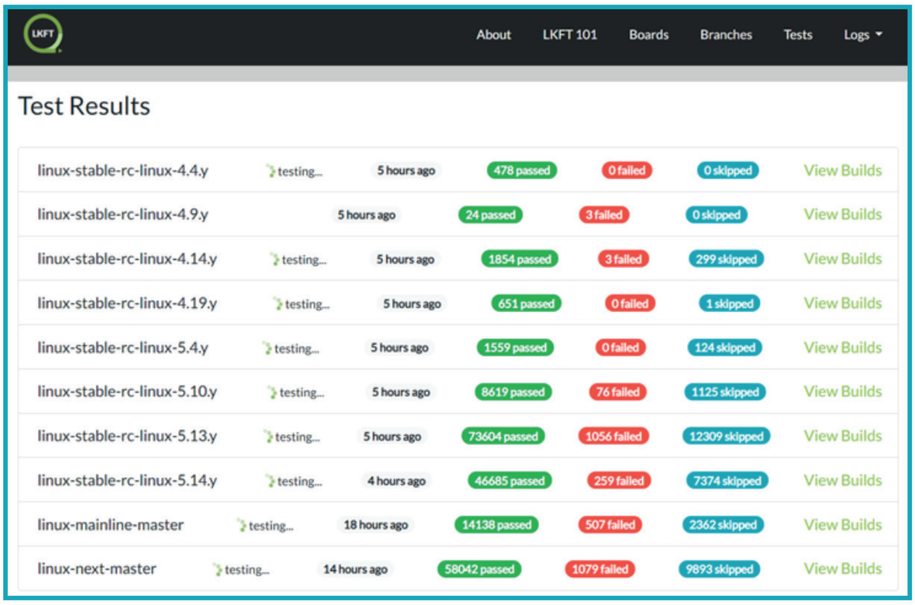
Trusted Substrate is aligned with Arm standardisation and certification programs - specifically Platform Security Architecture (PSA) and System Ready.

Trusted Substrate provides reference open source implementations for Dependable Boot, Other-the-Air (OTA) updates with anti-bricking and anti-roll back protections, Trust Services.

In order to find out more about Linaro Trusted Substrate, please visit https://www.linaro.org/trusted-substrate

Linaro's Linux Kernel Quality program covers both Linux kernel testing and testing of the LTS-derived Android-Common Kernel. Linaro's Linux Kernel Functional Testing (LKFT) framework (LKFT) is the most reliable Linux long-term-stable functional test framework in the industry. On a weekly basis, across the latest 6 Linux LTS releases, the linux-next branch and linux-mainline branches, Linaro build-tests and reports on over 350 release+architecture+target combinations on every git-branch push. We run functional-testing on nearly 40 of these combinations on real and emulated hardware and report back consistently with results in under 48 hours. We have run over 156 Million Test runs of the Linux LTS trees to date against a variety of embedded, emulated, and server platforms. We work weekly with LTS maintainers to execute testing and report regressions on the latest release-candidates before the releases are made.

We also build and functional test (Android CTS & VTS) Android Common Kernels weekly and report regressions in the Linux kernel and AOSP directly to Linux upstream maintainers and Google respectively. To date we've run over 530 Million Test runs against a variety of mobile chipsets, preventing regressions before they ever hit production mobile devices. Explore Linaro's Linux kernel functional test project at https://lkft.linaro.org



Test Results

| | | | | | | |
|---|---|---|---|---|---|---|
| linux-stable-rc-linux-4.4.y | testing... | 5 hours ago | 478 passed | 0 failed | 0 skipped | View Builds |
| linux-stable-rc-linux-4.9.y | | 5 hours ago | 24 passed | 3 failed | 0 skipped | View Builds |
| linux-stable-rc-linux-4.14.y | testing... | 5 hours ago | 1854 passed | 3 failed | 299 skipped | View Builds |
| linux-stable-rc-linux-4.19.y | testing... | 5 hours ago | 651 passed | 0 failed | 1 skipped | View Builds |
| linux-stable-rc-linux-5.4.y | testing... | 5 hours ago | 1559 passed | 0 failed | 124 skipped | View Builds |
| linux-stable-rc-linux-5.10.y | testing... | 5 hours ago | 8619 passed | 76 failed | 1125 skipped | View Builds |
| linux-stable-rc-linux-5.13.y | testing... | 5 hours ago | 73604 passed | 1056 failed | 12309 skipped | View Builds |
| linux-stable-rc-linux-5.14.y | testing... | 4 hours ago | 46685 passed | 259 failed | 7374 skipped | View Builds |
| linux-mainline-master | testing... | 18 hours ago | 14138 passed | 507 failed | 2362 skipped | View Builds |
| linux-next-master | testing... | 14 hours ago | 58042 passed | 1079 failed | 9893 skipped | View Builds |

# Accelerating Cloud Native In Automotive Through Arm SOAFEE Initiative

Another key technology component to consider in software defined vehicles is the utility of  the vast ecosystem of cloud native development which has matured over decades in the enterprise domain.

The cloud native design paradigm has been used in the enterprise domain to manage software complexity and deliver incremental functionality deployment. Introducing cloud native devops to automotive makes sense for the very same reasons and is likely to result in the automotive industry opening up to a wide ecosystem of software companies. This will help pave the way for the same type of revolution that we have seen in the mobile space.

Although promising, there are many challenges to consider when bringing cloud native into the automotive space. First and foremost, it is critical to extend the existing cloud native infrastructure for automotive workload development, which is realtime and safety relevant. An example of such an infrastructure element could be the usage of a Kubernetes orchestrator and the need to extend it to orchestrate mixed critical workloads.

These are complex issues which need to be solved by an ecosystem of companies across the automotive value chain. This is why, in collaboration with ecosystem leaders like Linaro and others across the automotive supply chain, Arm has recently announced the Scalable Open Architecture for Embedded Edge (SOAFEE) initiative and two new Arm-based centralized compute development platforms. The aim is to accelerate the adoption of the cloud native design paradigm in the software-defined future of automotive and "datacenter-on-wheels"architecture.

As a key ecosystem partner of Arm, Linaro intends to work closely on Arm's SOAFEE initiative and be a partner to help achieve scale in addition to contributing all the key technology components described in this document.

## How do I get involved?

Together with our member companies, Linaro has been working on all the key technologies we consider essential to enable the Software Defined Vehicle revolution through open source software. These span from the secure boot, Trusted Execution Environment and OTA, security and trust at system level, virtio and hypervisors, Time Sensitive Networking (TSN) and Automotive Ethernet, Linux and Android functional testing.

To speak to Linaro about how we can collaborate, contact contactus@linaro.org.

### About Linaro

Linaro leads collaboration in the Arm ecosystem and helps companies work with the latest open-source technology. The company has over 250 engineers working on more than 70 open-source projects, developing and optimizing software and tools, ensuring smooth product roll outs, and reducing maintenance costs. Work happens across a wide range of technologies including artificial intelligence, automotive, datacenter & cloud, edge & fog computing, high performance computing, IoT & embedded and mobile.

BACK TO THE TOP