Integrating Arm STL into microcontroller hypervisor on Armv8-R

110038

Alex Wilmot - Software Engineer, ETAS
Max Sinclair - Field Application Engineer, ETAS
George Bray - Product Manager, ETAS
Dr Andrew Coombes, Principal Automotive Software Product Manager, Arm
Bernhard Rill, Director Automotive Partnerships EMEA, Arm

This white paper describes strategies for integrating the Arm Self-Test Library (STL) into a software architecture which includes a microcontroller hypervisor.

Overview

The availability of microcontroller hypervisors based upon the virtualization capabilities of the Armv8-R architecture (including the 32-bit Cortex-R52+ and the 64-bit Cortex-R82 AE) provides a convenient approach to consolidating multiple applications into a single Electronic Control Unit (ECU). The hypervisor's ability to separate applications is important when it is necessary to ensure freedom from interference or to support a mixed-criticality system. This is achieved through the newly introduced Exception Level 2 (EL2) in the Armv8-R architecture which will host the hypervisor separation kernel.

Integrating the Arm Self-Test Library (STL) into a microcontroller hypervisor is a key step in meeting the safety objectives of applications with specific Automotive Safety Integrity Level (ASIL) requirements. This approach not only enhances fault detection and mitigation but also enables the development of flexible and scalable automotive systems that meet stringent safety standards.

This whitepaper describes strategies for integrating the Arm STL into a software architecture which includes a microcontroller hypervisor.

The paper will:

- Briefly introduce the virtualization capabilities of the Armv8-R and describe microcontroller hypervisors.
- Introduce the Arm STL.
- Describe how the "Out-of-Reset" use case can be addressed by the hypervisor.

- Describe options for integrating the STL in hypervisor to address the "Online" and "Online Event" use cases.
- Describe options and strategies for handling faults detected by the STL.

While the use of STL might be necessary to achieve specific ASIL targets for ISO26262, this white paper does not specifically address the topic of argumentation required to achieve functional safety certification. An overview on the topic of safety certification topic is outlined in the joint Exida/Arm paper "State of the Art Software Test Libraries (STL) and ASIL B: Truths, Myths, and Guidance"

Introduction to Armv8-R & microcontroller hypervisor

Cortex-R processors have been developed to enable applications where there are demands for real-time processing, requiring systems to respond in short and deterministic timeframes to meet system deadlines.

The Armv8-R architecture adds several features to support virtualization, of which two are the most visible:

- Three exception levels: EL0 for user code (Tasks), EL1 for operating system code, and one additional level, EL2, which is intended for hypervisor operations.
- A two-stage memory protection unit (MPU). This allows the OS to configure memory access at EL1 and the hypervisor to configure memory access at EL2.

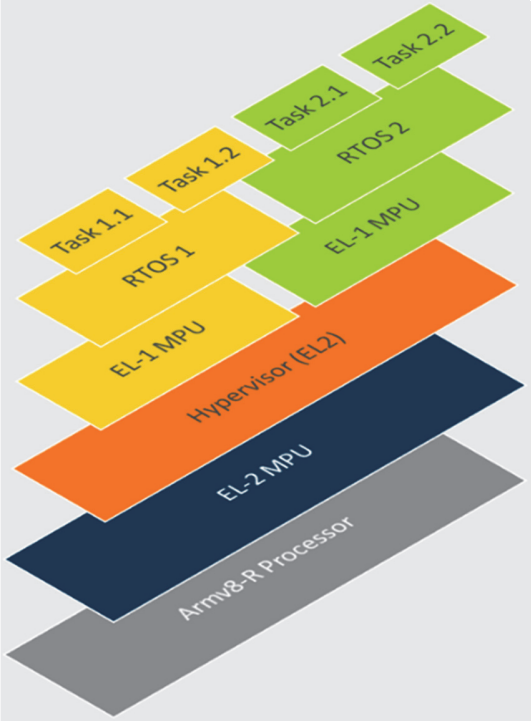

Figure 1: Armv8-R Cortex-R52+ exception level overview

Due to this, ECUs based on Armv8-R can support separated applications that run as independent instances – usually referred to as a *partition* or *virtual machine (VM)*.

The hypervisor creates the illusion of the software running inside a VM that is running on its own microcontroller. The hypervisor assigns VMs to physical cores, including multiple VMs running on a single core.

To learn more, see Best Practices for Armv8-R Cortex-R52+ Software Consolidation.

Introduction to Arm STL

Arm STLs consist of a set of tests which verify the proper functionality of the individual CPU building blocks. The STLs are one mechanism which enables an application to achieve the ASIL-B rating without dual lock step (DCLS) cores. They provide diagnostic coverage to prevent faults from leading to single-point failures or prevent them becoming latent as the result of multiple-point faults. For this the STLs need to be executed in the following scenarios:

- Out-of-Reset (OOR) – On startup of the application, the full test suite is executed to ensure the correctness of the underlying hardware before the application is run.
- Online (OL)– A periodic execution of the STLs where all tests are executed at a frequency determined by the application's safety requirements.
- Online Event (OLE) – The same as OL mode, but after each test part, the STLs check for a pending interrupt, allowing control to exit if an interrupt is present.

At their core, the STLs require two interactions:

- Initialization – The STLs must be initialized exactly once before out-of-reset scheduling begins and initialized at least once before online mode scheduling begins.
- Scheduling – The STLs require the application to call the scheduler to trigger test execution.

In the case of Armv8-R, both interactions must be executed at EL2 to ensure that you can achieve the highest Register Transfer Level (RTL) netlist coverage.

When executing the STLs (either as OOR, OL or OLE), interrupts are disabled to prevent interference with the tests. The time when the STLs are executing is therefore described as an interrupt blocking window - which prevents interrupts related to the application from being handled. Because OLE allows for interrupts to be handled between test parts, it has a shorter interrupt blocking window than OL.

The STL execution scenarios come directly from the respective STL reference manuals. It is up to the integrator to choose how they are used. To ensure full coverage Out-of-Reset must be used when the system is initialized, with either a choice of Online or Online Event used during the application execution.

Introduction to ETAS's Hypervisor for Embedded Targets (RTA-HVR)

RTA-HVR is ETAS's embedded hypervisor offering for real-time systems. Looking ahead to future industry needs, there is a growing demand for the consolidation of ECUs. RTA-HVR allows integrators to run multiple software stacks on a single ECU, assuring spatial and temporal separation. It supports running stacks from different vendors and in mixed criticalities.

RTA-HVR is a type 1 "Bare-metal" hypervisor, which runs directly on the platform's hardware. To be able to fully utilize the capabilities of RTA-HVR, the hardware platform must provide virtualization support such as that found in the Armv8-R series. By doing so RTA-HVR can partition system resources such as memory, peripherals and processor time, the partition (commonly referred to as a Virtual Machine) looks and behaves like its own system. ETAS also has a second hypervisor offering (RTA-LWHVR) for hardware without virtualization support.

STL integration test setup

The STLs were specifically tested on a developer board with both an M and R core. They were integrated into an internal ETAS RTA-OS testing application which is part of its test suite. Output traces were recorded into a RAM buffer with a synchronized clock, this allowed for accurate recording of the number of CPU cycles taken.

STL Out of Reset execution

Execution Options

Within ETAS RTA-HVR, there are a multitude of approaches one can take to run the STLs at boot-time, depending on what image the STL library was built into.

Master Software

Upon booting, before the HVR is started, the only software running out of reset will be the master software (which could also be referred to as the boot firmware). This software is not part of RTA-HVR and is to be provided by the integrator and/or silicon supplier. The purpose of this is to get the board running, configure any hardware as required, and to start the hypervisor. All the master software is executed at EL2, meaning the STLs can be executed directly with no exception level switch. Following on from running the STLs, the hypervisor can then be started and execution of the VMs can begin.

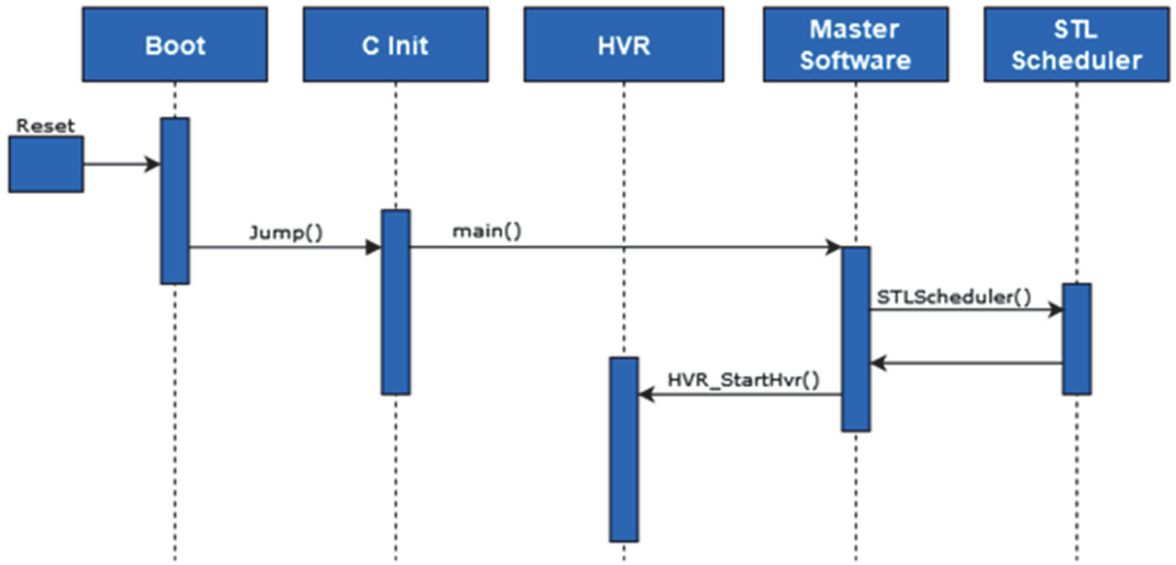

Figure 2: Master software sequence for OOR execution of STLs

Virtual Device Extension (VDE)

ETAS RTA-HVR defines a Virtual Device Extension (VDE) as a ‘plug-in’ that can be used to provide extra functionality that cannot be implemented in a VM, such as inter-VM communication, virtualized device drivers and more. Upon starting, RTA-HVR will initialize each VDE instances on the relevant core’s dependent on the hypervisor configuration. Since VDEs run at the same exception level as the hypervisor (EL2), the VDE can directly call the STL scheduler during this initialization callout to run the STLs during boot time

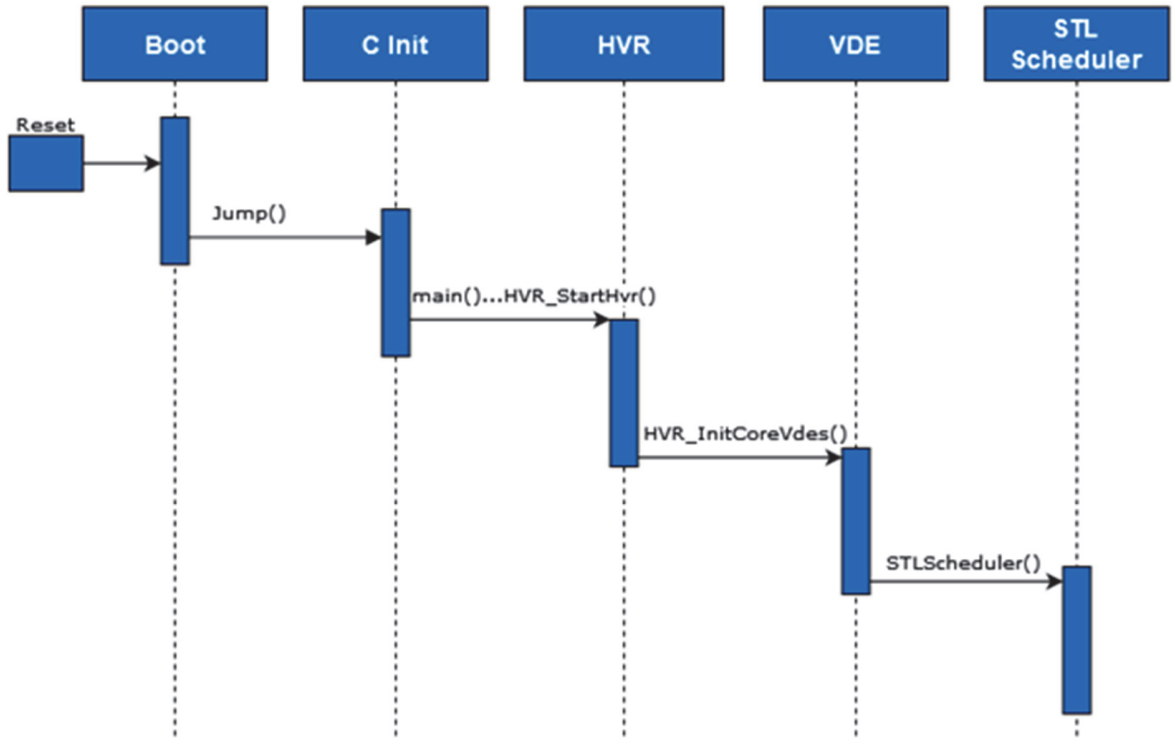

Figure 3: VDE sequence for OOR execution of STLs

Virtual Machine (VM) Startup

Finally, the last mechanism the STLs could be executed during boot-time, is during the 'boot' of the VM itself using mechanisms required in classic AUTOSAR described in Online (OL) and Online Event (OLE) Execution. Since the VMs will be running at EL1, this would mean they would need to make a hyper-call to a VDE instance, for the VDE to do the actual execution of the STLs at EL2. This means this method is essentially the same as running it directly from the VDE initialization, but with more steps and cycles needed. Although it would allow an existing AUTOSAR application with the STLs embedded to run them within the hypervisor.

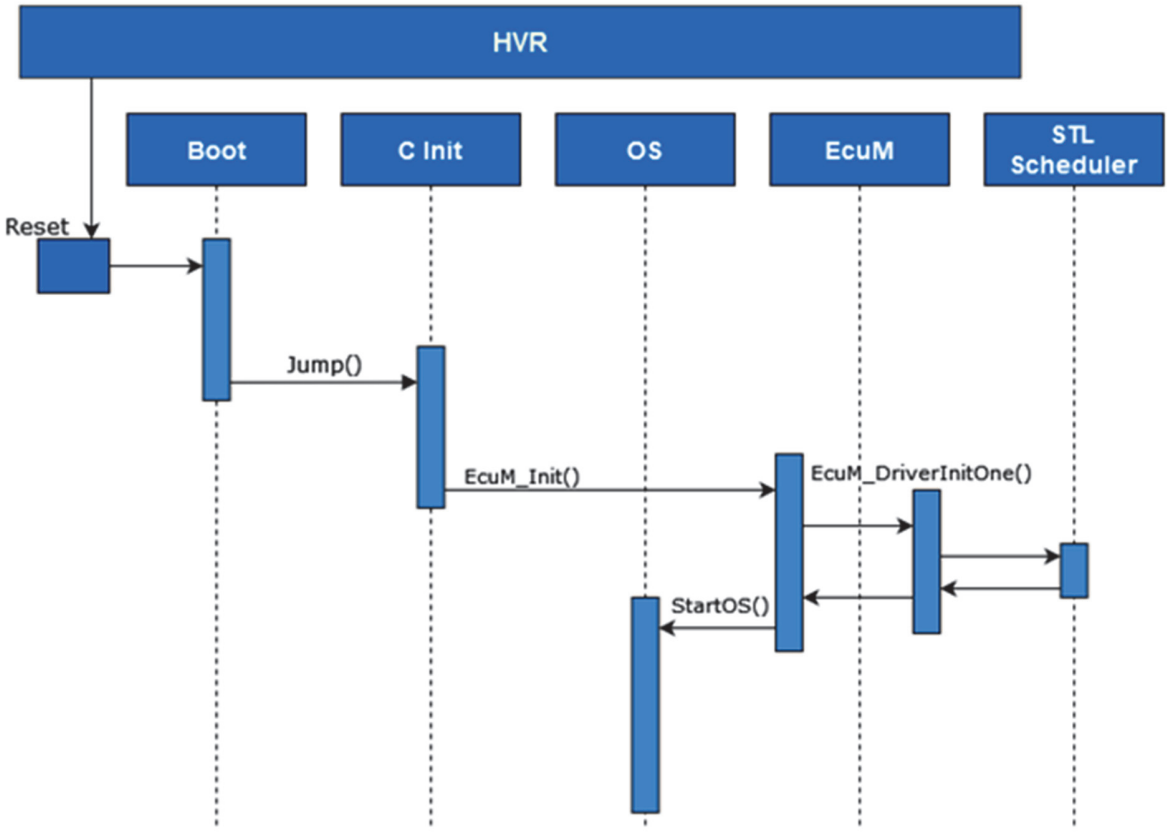

Figure 4: VM sequence for OOR execution of STLs

Comparison

The approaches mentioned above only dictate the phase during startup in which the STLs run in OOR mode. Clearly the methodology does not influence the execution time of the STL OOR test suite itself. The STL OOR test suite was measured to take approximately 420,000 CPU cycles in

our test environment, which would equal an execution time for 1.4ms at a CPU clock speed of 300MHz (this includes the time to initialize the STLs and restore the context after the STL execution). In case further details on the measurement are of interest feel free to contact your ETAS and/or Arm representative.

Online and Online Event Execution

Scheduling Options

There is much more flexibility to the integrator for running the STLs in OL or OLE mode on the hypervisor. The integrator can use various methods for running STLs from client software/VMs (which is out of scope for this document). Alternatively with the hypervisor running itself and VDEs in EL2, there are much more suitable methods using these constructs

Virtual Device Extension (VDE)

After startup of the HVR, VDEs can provide APIs to VMs and handle interrupts directly at EL2. This means the STLs can be run in two different ways:

- Periodic scheduling can be managed within the VM, which makes hyper-call(s) to the VDE to trigger the STL scheduler.
- The VDE can directly trigger and manage a periodic interrupt to run the STLs directly without interaction with any VM.

The first method is more desirable if the STLs need to be controlled or triggered via the VM and would take approximately a similar number of cycles as a classic AUTOSAR approach. The second method would involve fewer cycles, since the scheduling and STL execution is all managed directly in the VDE. This means that no extra cycles are required for handling VM requests to the hypervisor directed to the VDE instance, or an exception level switch.

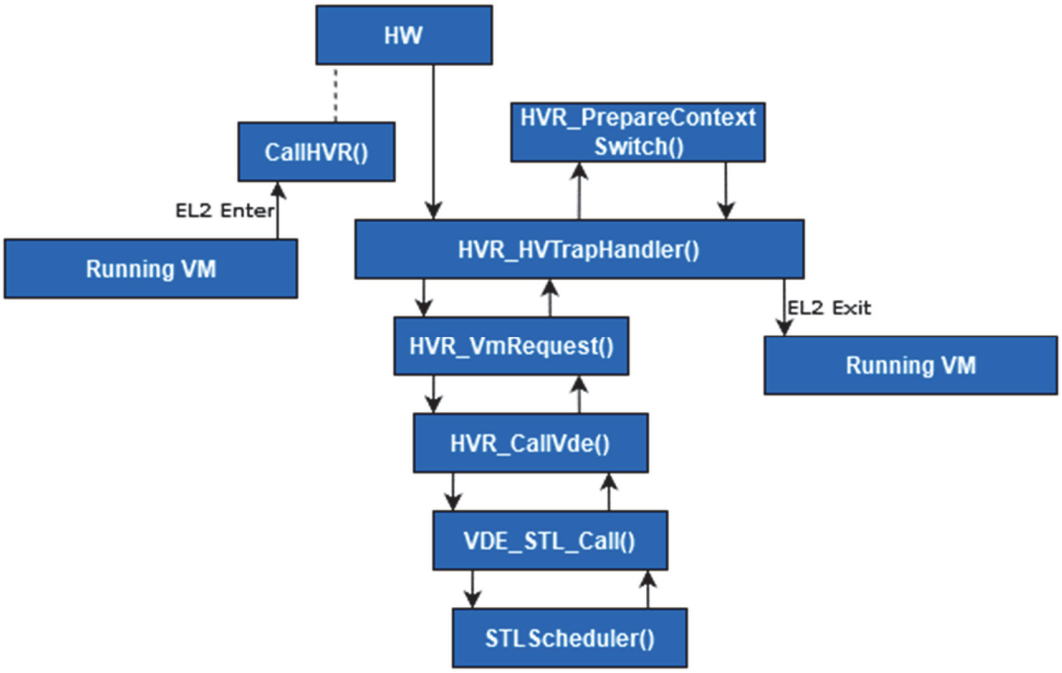

Figure 5: VDE sequence of OL execution of STLs

EL2 Virtual Machine (VM)

Normally, when the hypervisor schedules a VM and loads its context, it will switch to EL1 at the point of returning to the VM context. However, for the purpose of executing the STLs, RTA-HVR can be trivially modified so that it allows 'special-purpose' VMs that instead run at EL2. The context switching time to and from this VM is slightly faster, however the memory footprint of the hypervisor itself is also slightly larger due to the extra configuration of an additional VM for STL execution.

Using this, a single-purpose VM can be created that simply calls the STL scheduler in a while loop. The integrator then can use the RTA-HVR configuration to dictate the scheduling of this VM to run at the desired rate. This would mean that every time the VM is scheduled, it would execute the scheduler before the next VM is scheduled.

Careful consideration is required due to the interrupt blocking window of the STLs. For safety cases on a hypervisor, spatial and temporal separation must be ensured between VMs in their own sandbox(s). If this special purpose VM blocked interrupts for a period when the hypervisor clock tick interrupt fires, this would mean for the duration of the test set, it would be blocking a

context switch to the next VM. This would not be allowing the hypervisor to keep true to the defined scheduling policy for the VMs.

Below shows an example of the sequence of events occurring starting from a VM A, to running the STLs inside VM B. The diagram matches a usual context switch for the VM – however since the VM is designed to run at EL2, there is no switch performed by the VM.

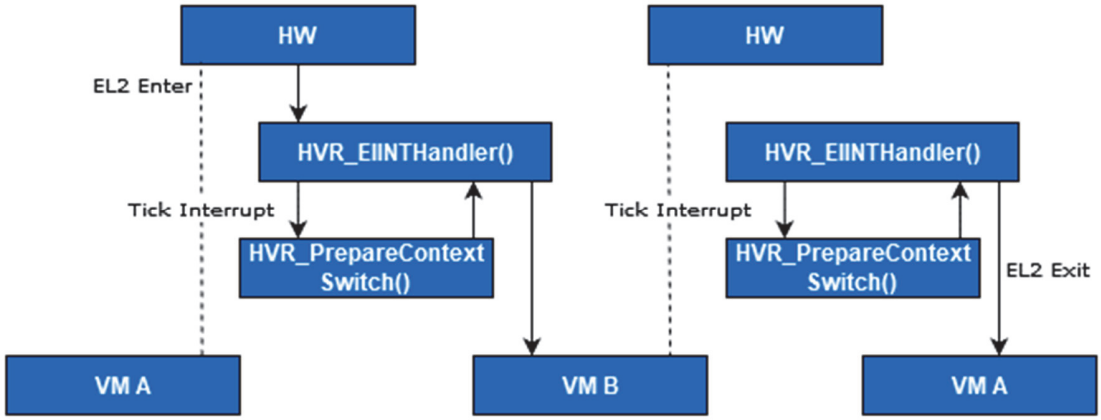

Figure 6: Master software sequence for OOR execution of STLs

EL1 Virtual Machine (VM)

Finally, the STLs can be integrated into the VM itself in the same approach as in Classic AUTOSAR or other RTOS client software. This would be desirable if an application already had STLs integrated, as the modifications required to run the application on the hypervisor would be minimal, whilst yielding similar execution times of the STLs as previously shown on a non-hypervisor-based system. This would essentially be the same process as having a STL VDE, as the EL2 switch described in Classic AUTOSAR Application. However, this is the same procedure used to make a VDE call from a VM in RTA-HVR, and hence will yield the same or similar duration to call and return from the STLs.

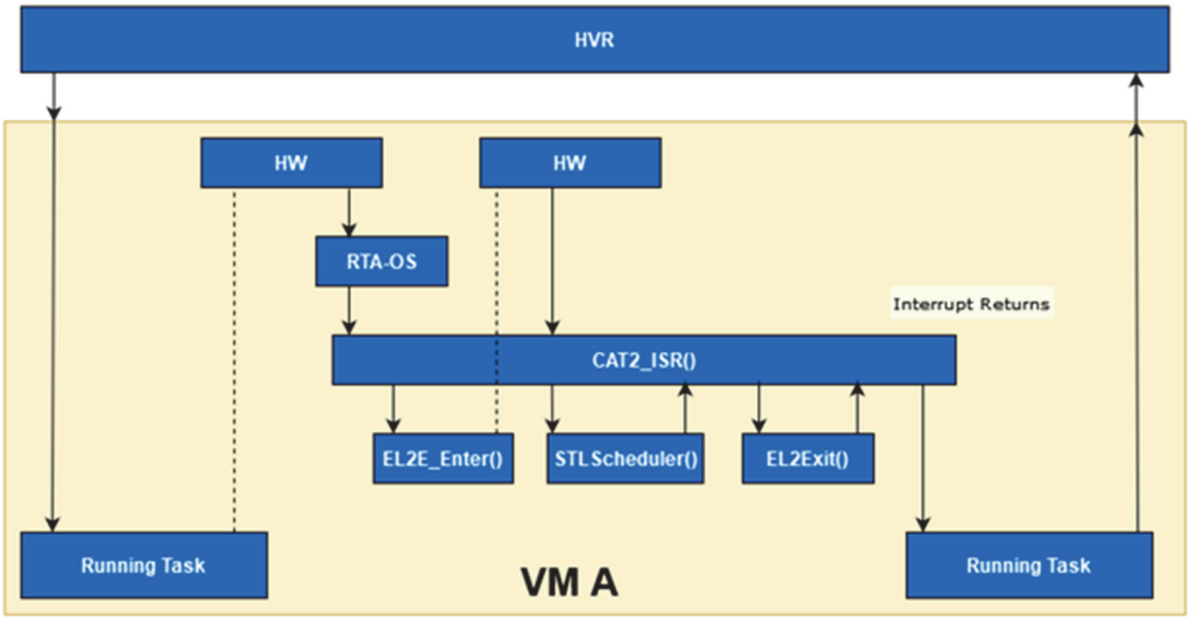

Figure 7: EL1 VM sequence for OL execution of STLs

Comparison

The following table shows the number of cycles required before the STL Scheduler can begin executing the STLs, with the start point being another VM running requiring a context switch first.

Method	CPU Cycles
STL VDE	2636
EL2 VM	2143
VM Integrated STLs	2292

Table 1: Comparison of setup time of OL execution methods in a hypervisor application

As mentioned in the test setup we had a CPU clock speed of 300 MHz, thus depending on the software architecture selected to initialize the STLs the time for the setup varies between ~7.15us and 8.79us.

In the context of a hypervisor, the OLE mode of execution for the STLs could be most plausible in some cases. For example, in the case of core sharing, a single physical core can contain multiple executing VMs. In this case, a clock tick interrupt is required for the hypervisor to schedule both VMs. In this scenario, the OLE mode would be preferable, as if the clock tick interrupt occurs during execution of multiple STL test parts, it would allow the hypervisor to switch to the other VM in a smaller time window, thus not blocking execution of another VM

Fault Handling

Within ETAS RTA-HVR, fault handling is done via a set of callbacks that the hypervisor makes to the master software code, allowing the integrator freedom in how the system shall react to different events. The hypervisor allows the possibility of restarting VM's in the case that faults have occurred within them at runtime.

However, in the case of running STLs if a test failure occurs, this signals a fault with the core that is being tested, meaning that restarting a VM will cause it to then run with possibly faulty hardware. Instead, the most appropriate reaction would be to reset cores individually depending on where the fault occurred. This would however introduce new considerations for the integrator to make, such as implications on GIC configuration when resetting a single core in a cluster, and overall the option to reset a CPU has high dependencies on the overall CPU integration by the silicon partner.

When discussing fault handling in a classic AUTOSAR system, both a watchdog, and secondary core were discussed to monitor the status registers of the SBIST controller. In the case of the hypervisor, it would be most likely that only a watchdog would be suitable. This is because most likely other cores will be running separate virtual machines.

Summary and Conclusion

The integration of the Arm Self-Test Library (STL) into Armv8-R based microcontroller hypervisors offers a robust solution for consolidating multiple applications into a single ECU whilst having a running mechanism (through the STLs) to validate the integrity of the CPU(s). This approach is particularly beneficial for ensuring freedom from interference and supporting mixed-criticality systems, where applications of varying safety levels can coexist without compromising overall system integrity.

All measurements shown in this white paper are derived from commercially available development boards. If further information is required on the measurements or other topics, contact your representatives at Arm or ETAS.

Glossary

Abbreviation	Explanation
ASIL	Automotive Safety Integrity Level
ASW	Application Software
AUTOSAR	Automotive Open System Architecture
BSW	Basic Software Stack
CDD	Complex Device Driver
ECU	Electronic Control Unit
EL2	Execution level 2 (hypervisor exception level)
OL	Online (method of executing STL)
OLE	Online Event (method of executing STL)
OOR	OOR – Out of reset (method of executing STL)
STL	Software Test Libraries
VDE	Virtual Device Extension (ETAS RTA-HVR “plug-in”)
VM	Virtual Machine