Deeply Embedded Real-Time Hypervisors for the Automotive Domain
Dr. Gary Morgan, ETAS/ESC
Overview of presentation

This presentation is concerned with what drives the use of hypervisors in the automotive domain, and what an automotive-class hypervisor might look like.

The main overall driver is convergence:

- Hardware is becoming less diverse.
  - Fewer processor types.
- Software is converging on a smaller number of operating systems.
  - AUTOSAR (international de facto automotive software standard)
  - Linux
  - Applications are converging on a smaller number of ECUs

- Coupled with a much wider range of application requirements

- Increased safety and security
Car electrical/electronic (E/E) systems are getting more complex.

Software trends
- More complex
- More diverse
- Dynamic
- Highly connected

Hardware trends
- More processors (multicore, manycore)
- More memory
- Faster, more diverse networks
- COTS and CE convergence

Internet in the car
- Car = Wi-Fi hotspot for mobile networked devices
- Car = extended display ("terminal mode") for CE devices

Reality: Car = node in the internet
- Functions migrate into the cloud

Scenarios
- Car in the internet
  - Car = part of highly networked ecosystem
  - Via networking, car offers a tangible value proposition
E/E system complexity needs to be reduced

E/E Complexity
- Too many ECUs
- Merge to Domain Control Units
- High speed inter-DCU links
- Legacy for control/sense

Key questions
- What’s in a DCU?
- How are “virtual ECUs” isolated?
- How is the system coordinated?
- How is legacy supported?

Fast backbone – gigabit range
Hypervisor introduction

The purpose of the hypervisor is to allow multiple ECU “images” to run on the same ECU, with appropriate levels of separation and scheduling guarantees
- Allows different parts of the E/E system to be combined on a smaller number of ECUs.

The hypervisor is a thin layer that sits between the hardware and the “images”. It provides a virtual machine (VM) in which the “image” runs
- This is also called a type-1 hypervisor.
- Type-2 hypervisors run on top of an existing OS or RTOS

In addition to separation, it also provides abstractions to allow access to hardware devices and inter-VM communications.
Deeply Embedded Automotive Hypervisors
Hypervisors as part of the solution

Abstract architecture of ETAS hypervisor system

Virtual Machine

Virtual Machine

Hypervisor

MPU abstraction

Exceptions

Services

VM-VM Comms VDE

Shared IO VDE

Safe, secure inter-VM comms

For protected systems

ECU “image”.

Virtual Device Emulator arbitrates shared hardware

IO via HV. Slow.

Direct IO. Fast.

1:1 CPU to VM mapping. Lower latency. Easier scheduling

Own IO

Own IO

CPU, MPU

Shared IO

CPU, MPU

Own IO

Own IO

Implementation on Aurix and ARM processors
Hypervisor introduction

− Reduce integration costs
  − Less need to port old applications to new operating systems.
  − AUTOSAR facilitates semi-portable application code. Hypervisor solution competes with AUTOSAR for application integration.
    − If you have 2 or more systems, which is better?
      − Integrate using AUTOSAR
      − Integrate using a hypervisor

− So you need to compare systems integrated with AUTOSAR with those integrated with a hypervisor
Integrating Systems

Original systems

Integrate with AUTOSAR

Integrate with Hypervisor

<- Applications
<- Communications
<- OS and drivers
Hypervisor introduction

- Integration with AUTOSAR
  - May be hard to handle different versions of the same OS
  - Different OS’s cannot be handled (e.g. AUTOSAR and Linux)
  - What is the cost of re-configuring the RTE and BSW?
  - How is mixed criticality handled?
- Integration with hypervisor
  - Better spatial separation
  - Better temporal separation
  - Better API separation
  - VM restart compared to complete ECU restart
  - Better security
  - Faster boot time
  - Some costs rise
    - Common code is duplicated in each VM rather than shared

These lead to much better support for safety systems and mixed criticality systems.
Deeply Embedded Automotive Hypervisors
Overview of real-time problems

- Hardest requirements are typically in engine control
- A key performance indicator is interrupt latency
- Typical “high end” controller
  - Around 40,000 interrupts per second
    - AUTOSAR tends to concentrate interrupt onto one core
    - Overheads too high if CAT2 ISRs are used
  - Around 120 interrupt sources
  - Around <5us interrupt latency, fastest case
  - Clock speed around 300 MHz
  - 2MB FLASH
  - <512MB RAM
  - 2 or 3 cores
  - Shared RAM, FLASH and peripherals

- CPU loads are often in excess of 90%

- How do you build a hypervisor in such a small system and how do you guarantee its real-time properties?
Deeply Embedded Automotive Hypervisors
Current VM scheduling approaches

- Simplest approach: 1 VM per core.
  - Approach adopted by ETAS’ hypervisor
  - No VM scheduling to worry about
  - Scheduling of HV internal operations at highest interrupt priority
  - Low interrupt latency

- Cyclic (Arinc 653)
  - Predictable
  - Fair (avoids starvation)
  - Long interrupt latency

- Hypervisor implements VM processes
  - All processes in all VMs scheduled by hypervisor
  - Bad for portability and legacy compatibility
    - Hypervisors’ scheduler may not support AUTOSAR, legacy and Linux scheduling policies.
Multiple VMs per core are required

A scheduling model is needed which is
- Fair to the VMs (no starvation)
- Enforces temporal separation
- Has low interrupt latency
- Can be modelled and reasoned about
- Has low run-time overheads

Partitions have a Synchronous Mode that runs when budgets permit
Interrupts take time from the Asynchronous Mode budget until their budget is used up
The hypervisor always has the highest priority

Proposed priority space assignment
Burns, Evripidou, Morgan
Automotive has many domain-specific requirements

Small systems:
- Low interrupt latency
- Small footprint
- Static configuration
- Hard real time

Large systems:
- Comprehensive peripheral support
- Dynamic configuration
- Feature download
- Soft real time

All systems:
- Certification
- Boot loaders
- Diagnostics
- Security
- Configuration
- Portability
- Safety
- Calibration
- Multicore
- Shared-device abstractions

Some requirements are contradictory and need configuration to resolve them. For example, static vs. dynamic configuration should be statically configurable.

Configuration allows some requirements to be removed. E.g., Diagnostics might be configurable.

How many of these requirements are supported by current commercial hypervisors? Very few.
An automotive hypervisor for 2018 onwards?

- Support all relevant use cases and requirements

- More than one VM per core, with predictable timing.
  - more VMs than cores

- More than one core per VM, with predictable timing.
  - support for multi-core VMs.

- VM migration
  - Migrate VMs to avoid overloaded or faulty hardware

- Timing model
  - System becomes a composition of VMs, with known timing behaviour

- Communications (RTE++)
  - System-wide inter-VM communications mechanism
Thank you for your attention

Dr Gary Morgan
ETAS/ESC

gary.morgan@etas.com
+44 (0)1904 562584