How to tackle modern massive-scale HPC on the vehicle

AGL All Member Meeting 2025-02 (Tokyo)

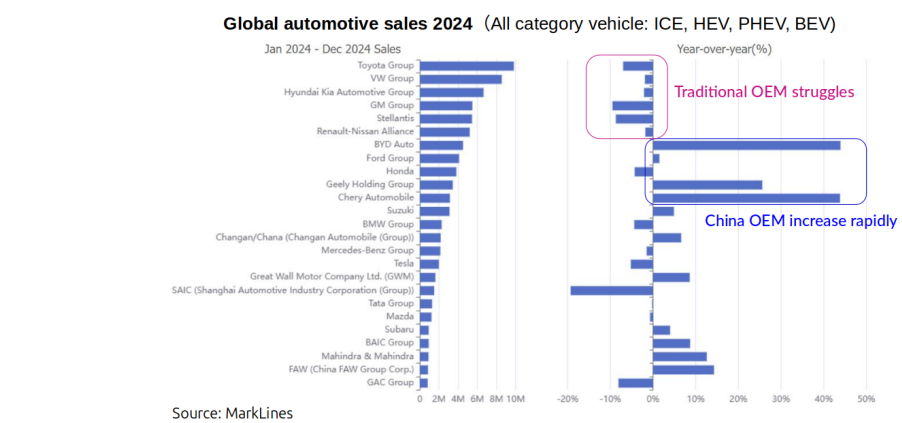
Hisao Munakata
hisao.munakata.vt(at)renesas.com

High Performance Computing Product Group
HPC SoC Business Division, SoC Software Enablement Department
Senior Director

2025-2-26

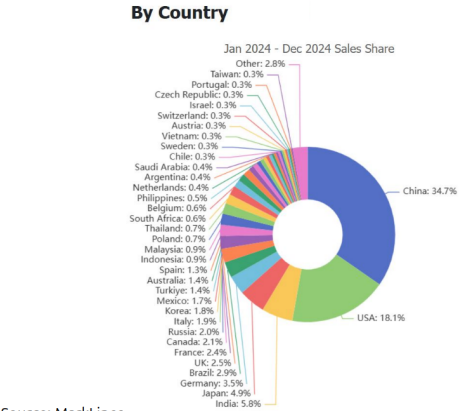
What is happening now

2024 Global automotive sales result and YoY growth state



https://www.marklines.com/en/vehicle_sales/dashboard

China is the biggest (34.7%) automotive market



Source: MarkLines

https://www.marklines.com/ja/vehicle_sales/dashboard

China market trends: People start buying Chinese cars, why?

	2020	(share)	2024	(share)	
China OEM	10,563,081	41.7%	17,578,193	55.9%	166%
USA OEM	3,365,175	13.3%	3,049,470	9.7%	91%
JP/KR OEM	5,149,487	20.3%	4,871,403	15.5%	95%
EU OEM	4,828,508	19.1%	4,013,172	12.8%	83%
Total	25,311,069		31,436,193		124%

https://www.marklines.com/ja/vehicle_sales/dashboard/nation?nationCode=CHN&fromRange=2020-01&toRange=2020-12&groupId=&

Car without ”upgradability” are no longer acceptable in China

SW: almost bi-weekly updates (OTA)

- Cockpit (40%)
  - Video and Audio apps.
  - Voice control, Sound effect
- AD/ADAS (45%)
  - NOA (Navigate On Autopilot)
  - LCC (Lane Change Control)
- Body (15%)
  - Driving Recorder
  - BT unlock
  - Lock optimization

HW: ECU replacement option

- Zeeker001 (2020)
  - Free SoC replacement offering
- NIO ES8/ES6/SC6 (2020)
  - Paid SoC replacement option
- In some cases, crowdfunding is used to determine which features to provides update option

Why foreign traditional OEMs still can hardly offer SW/HW upgrades?

Why traditional OEM struggles with SDV

Traditional car is considered "embedded" device

Non-embedded = General-purpose

- PCs, smartphones, etc
- Usage varies by user settings
- Install applications to customize
- Sufficient computing resources to accommodate various uses
- Memory and storage expansion
- Can expand HW via USB, etc.

Embedded (= for specific purposes)

- Products for specific applications
- Because the functionality is the same, cost competition is intense
- Minimum computing resources
- Cannot update SW after shipment
- No free HW expandability
- Examples: HA, industrial, game, feature-phone, automobile

"Embedded" is a battle against constraints due to cost competition

Embedded software developer's behavior discipline

Embedded developer need to tackle with various constraints

- Limited CPU processing capability (time constraints)
  - Response time for interrupt events must be defined (real-time requirements)
  - Startup time constraints (lightweight startup processing)
  - Power constraints (thermal management, ensuring battery life, etc.)
- Memory size cannot be freely expanded (capacity constraints)
  - RAM size = Working memory, frame-buffer (screen size)
  - ROM size = ROM size = Program size limitations.
- Zero-defect requirement (No bugs allowed)
  - Cannot update SW after shipment (extremely high reliability required)
  - Secure test coverage by use-case based system validation

Embedded SW requires "precise control to overcome constraints"

Embedded SW developer aims for ”Complete Predictability”

Functions required to achieve ”complete predictability (determinacy)”

- Task execution order management→ Controlled execution sequence based on task priority.
- Guaranteed interrupt response time → By the setting of interrupt-disabled regions.
- Memory management → Static memory allocation.
- Hardware resource management → Static functional assignment
- Failure analysis and recovery → Implement custom log message.

- In embedded systems, software designers bear full responsibility for ensuring that the system does not break down under any foreseeable condition by designing software that achieves ”complete predictability.”

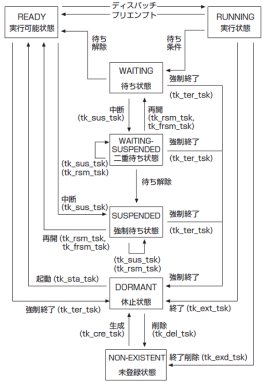
If HW/SW is fixed, ”complete predictability” might be possible, but...

RTOS is the foundation of ”Complete Predictability”

Programmer retain full control of task execution order

- Preemption
  - Higher-priority task can interrupt running task
  - It can wake-up task inside the program, allowing deterministic execution as expected.
  - It can wake-up task from an interrupt handler, making it possible to control the time from interrupt occurrence to execution.
- Programmers take full responsibility for state transitions between tasks (= predictability)
- If task added later, predictability collapses

図 1. タスク状態遷移図



T-kernel のタスクの状態遷移

https://www.tron.org/wp-content/themes/dp-magjam/pdf/t-kernel_7.0/html_ja/task_states_and_scheduling_rules.html

”RTOS” and ”Linux” fundamentally differs

RTOS = create complete predictability

- Applicable domains
  - HA, feature-phone, industrial
  - Automotive ECU control
- For uP (typically single-core)
- Ensures real-time performance
- Functional safety
- Examples of RTOS:
  - Open-source: FreeRTOS, Zephyr
  - Commercial: SafeRTOS, QNX

Linux = OS arbitrate execution order

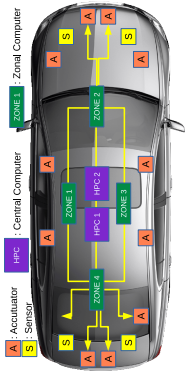
- Applicable domains (broad)
  - Super-computer (science,..)
  - Servers (finance, securities)
  - Auto(IVI), Android-phones
  - IoT, routers
- 1,000 cores+ and 5,000 threads+
- Multi-user, multi-task capable
- Memory protection, virtualization
- OSS → Allows behavior analysis

Task execution order management in RTOS and Linux largely differs

”HPC”, that can run Linux OS, realizes SDV concept

Partial software updates require OS-managed arbitration

- Infotainment (HMI), Gateway (Network Router)
- Architectural Revolution of Vehicle Computers (HPC)
- HPC is not a dedicated embedded system
- Can handle various workloads
  - Integrates existing ECU’s functions
  - Requires real-time guarantees for certain use cases
  - Needs to comply with functional safety requirements
  - Must consider cloud connectivity and cybersecurity



Existing ECU functions are integrated into the HPC for updatability

R-Car X5H is far more powerful than Intel's Core-i9 14900K

R-Car X5H SoC		Intel Core i9-14900K (Raptor Lake Refresh)
Arm CA9 x 32 Arm CR52 x 6 (dual LS x 3)	CPU core number	24
1,000k (CA) + 60k (CR)	CPU (DMIPS)	450k ～ 900k *1 (PassMark 45,000)
2,000 (support 8k x 10 input)	GPU (GFLOPS)	793
400	NPU (TOPS)	Not included
2.7GHz	CPU clock	3.2 GHz
3 nm	process node	10 nm
2,916	pin count	1,700

1: We used **1 PassMark ≈ 10-20 DMIPS** as a general approximation performance conversion

Way to break the current blockers

Mind set of ”traditional embedded SW developer”

Chasing a perfectly pre-coordinated world

- Hardware utilization first
- No hesitation for proprietary APIs
- Developed from scratch (no reuse)
- Each chip requires a dedicated development environment
  - IP-specific software libraries
  - BSP (Board Support Package) tailored for specific boards
- Aims complete predictability

Embedded system SW developer feels a sense of craftsman’s proud

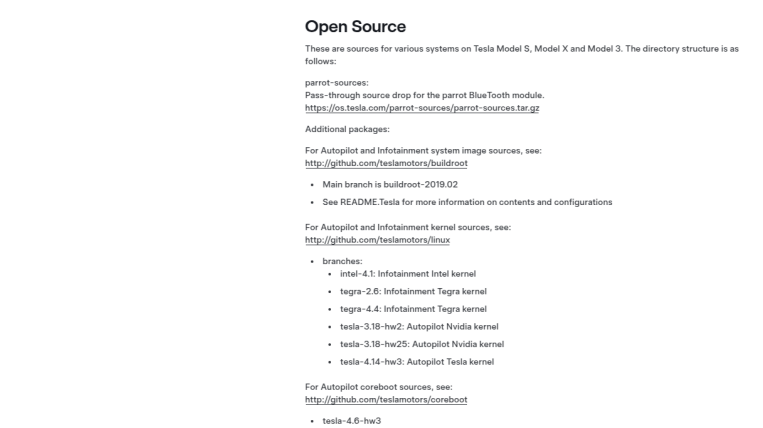
Mind set of ”Enterprise/IT system SW developer”

Aims flexible, transformable, commonality

- SW is not tied to specific hardware
- SW can run on a wide range of hardware
  - Processor performance range
  - Memory and storage size
  - Screen size (landscape/portrait)
- Concurrent application development
- HW/SW evolve asynchronously
- Relies on OS’s arbitration for a task execution
  - OS includes smart scheduling mechanism

As SW and HW are fully decoupled, they can migrate respectively

FYI: Tesla is using Linux for Autopilot control



Open Source

These are sources for various systems on Tesla Model S, Model X and Model 3. The directory structure is as follows:

parrot-sources:
Pass-through source drop for the parrot BlueTooth module.
https://os.tesla.com/parrot-sources/parrot-sources.tar.gz

Additional packages:

For Autopilot and Infotainment system image sources, see:
http://github.com/teslamotors/buildroot

- Main branch is buildroot-2019.02
- See README.Tesla for more information on contents and configurations

For Autopilot and Infotainment kernel sources, see:
http://github.com/teslamotors/linux

- branches:
  - intel-4.1: Infotainment Intel kernel
  - tegra-2.6: Infotainment Tegra kernel
  - tegra-4.4: Infotainment Tegra kernel
  - tesla-3.18-hw2: Autopilot Nvidia kernel
  - tesla-3.18-hw25: Autopilot Nvidia kernel
  - tesla-4.14-hw3: Autopilot Tesla kernel

For Autopilot coreboot sources, see:
http://github.com/teslamotors/coreboot

- tesla-4.6-hw3

https://www.tesla.com/legal/additional-resources#open-source

Avoid bringing company internal politics into ”SW partitioning”

Old-guard crony (veteran embedded)

- Avoid touching existing code
  - Prioritize real-time performance
- Refuse to engage with general-purpose OS features
  - Speculative scheduling
  - Dynamic memory management
- Add Linux tasks on RTOS
  - Only provide POSIX-compliant APIs
  - Do not utilize OS arbitration

Innovators (IT techno-hazard)

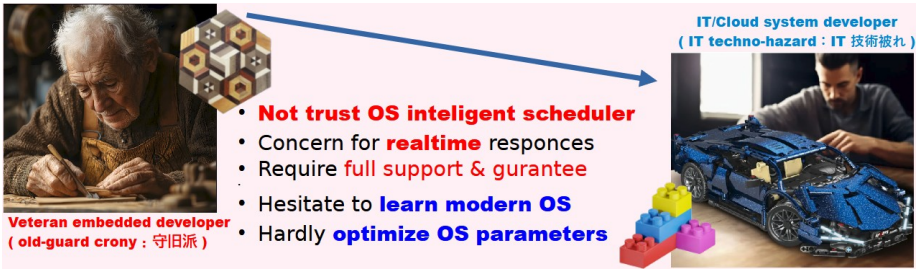
- Follow the ”smartphone approach”
  - Lead to code bloat
  - Cause longer system boot times
  - Insufficient system validation
- Lack understanding of automotive-specific requirements
  - Ample computing resources
  - No care for cost constraints
  - Cannot integrate legacy codes

The automotive industry is still dominated by ”embedded” thinking, but failing to embrace new trends will leave companies behind.

Need to convince veteran developer with tangible example

Typical old-guard crony's complains to modern OS adopter

- Cannot rely on OS's automated task execution order control
- Cannot trust worst-case behavior, like real-time deadline miss



Software Migration Strategy to Leverage HPC

To leverage multi-threading, focus on parallelism instead of processing density

- Refine RTOS processing contents
  - Carefully examine deadline requirements (do not simply migrate as-is).
  - Identify processes that can release the CPU during execution
- Minimize synchronization points by adopting stateless structure
- Utilize OS automatic arbitration mechanisms effectively
  - Minimize the use of real-time tasks
  - Optimize scheduling parameters
  - Offload deadline-sensitive processes to co-processors
  - Allocate sufficient memory resources
- Leverage Hypervisor for hardware separation (guest OS, audit functions)
- Containers for application encapsulation

Renesas plans to release new R-Car reference platform

R-Car V4H SBC (Single Board Computer)

- CA76 quad core + CR52 realtime core
- Multiple camera input + embedded NPU
- Developer friendly
  - Easy to buy (from web)
  - Affordable price
  - Small form factor + common peripheral interface
  - Web based support (no paper contract required)
- OSS community friendly
  - Plan to support in Linux upstream kernel
  - Plan to support in Xen upstream kernel
  - Plan to support AGL environment

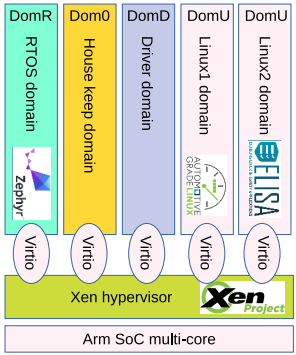
Consider "AGL SDV reference PF" to convince "embedded" folks

Integrate SDV core SW on top of AGL

- Xen hypervisor (now chasing FUSA)
- Zephyr RTOS (now chasing FUSA)
- ELISA Linux kernel (now chasing FUSA)

- OASIS Vritio (common VM interface)
- COVESA VSS (common dataset)

- AGL reference HW
- R-Car V4HSBC (new)



We can proof the validity of truly OSS-based modern SW platform

Conclusion: AGL has potential to truly open the door for SDV

- Traditional OEMs (US/EU/JP) struggle to realize vehicle upgrade experience (SDV), which results in a share loss in the biggest automotive market.
- One major blocker to realizing the SDV concept is the ”embedded” way of thinking that chases ”Complete Predictability” of SW behavior.
- It is hard to convince veteran embedded SW developers to work with modern general-purpose OS like Linux. We need to show tangible performance and capability.
- As AGL maintains code, we can build an SDV-ready SW platform combining existing LF’s SW assets like Xen, Zephyr, and ELISA. Also, we hope to contribute to ongoing Xen and Zephyr FUSA compliance.