Resolve and standardize early access to hardware for automotive industry with Linux

Khasim Syed Mohammed

Engineering Lead – Texas Instruments

Who am I ?

- Linux device driver developer 2002-2012 with Texas Instruments, though most of kernel contributions went in 2.6 ☺

- Founder member for beagleboard.org with Jason Kridner

- Tech lead at Linaro (2012-2022)
  - 64bit Android on Arm software simulators
  - Project Ara – Modular phone project from Google
  - Android on Arm Servers with Docker containers
  - Arm’s N1SDP yoctification for UEFI and other kernel components.

- Back in Texas Instruments (2022) as Engineering Lead for Sitara MPU products.
  - Getting opportunity to work closely with automotive industry,
    - who are on proprietary operating systems for many use cases
    - relying on on-chip MCUs for safe and time critical applications.

Why am I here ?

Share and Learn to build the automotive use cases the “Linux” way with Linux subsystems (Kernel, U-Boot, distros) :

Safety Certification

- Enable fault less systems with proven safety qualified “open source” software.

Performance (Early boot)

- CAN response < than 100 msec.
- Wake up response on Ethernet < 150 msec.
- Audio tone on speakers < 500 msec.
- Camera stream to screen < 750 msec
- Display Animated graphics < 1 sec.
- and more ...

Power Management

Enable power efficient systems.

- Elaborate PM policies for remote cores.
- User space hooks to handle power modes.
- Suspend to RAM policies.

The solution industry has found is either with

- Heterogeneous processors (on chip MCUs) that is not scalable.
- With non- standard (no open standards followed) and not so Linux friendly approach.

What I want to walk away with ?

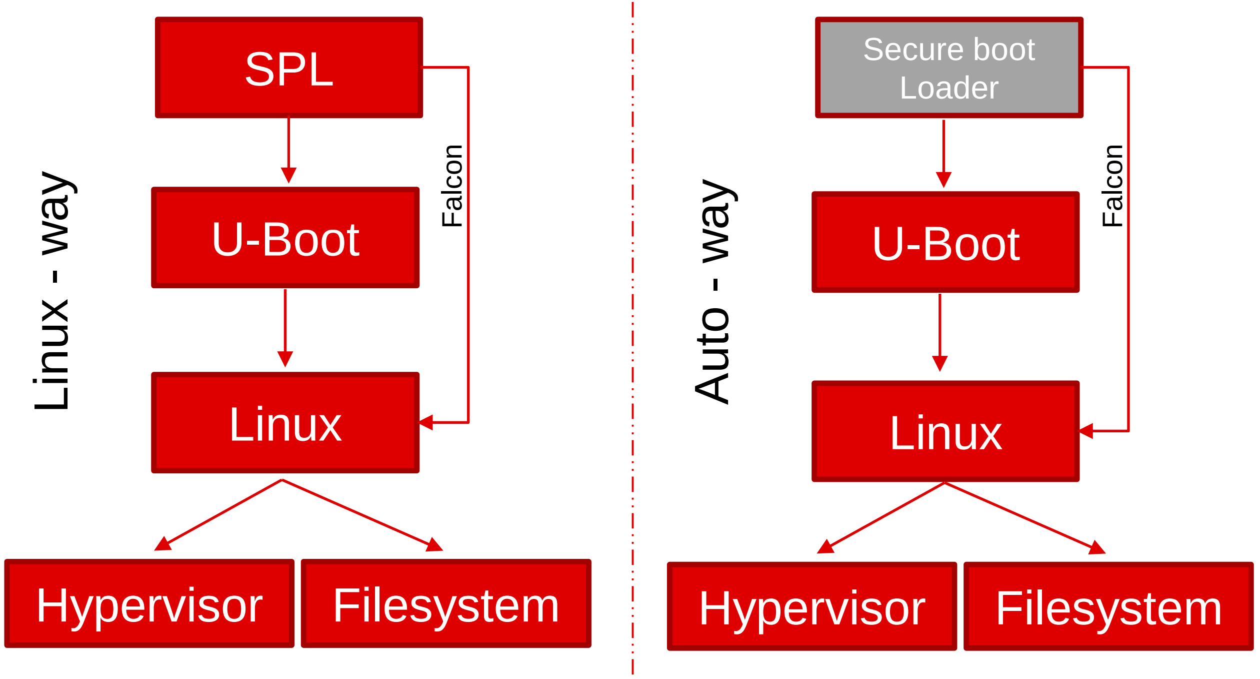
- Deep dive into exact problems and the current solutions and how we migrate the current RTOS based solutions to switch to Linux “only” based solutions.
- How we standardize the "Linux late attach" with heterogeneous SOC.
- If I could find representatives from the automotive OEM, SOC manufacturers and Linux kernel and user space maintainers to :
  - Collaborate and help in defining "Linux automotive" standards for the auto use case implementation
  - Harden & improvise the Linux kernel & drivers to meet the key performance requirements.
  - Learn from the subject matter experts here and incorporate the learnings in our solutions offered to customers / industry.

Problem Statement
(ask)

Current Solution
(lacks community collaboration)

Long term solution
(Standardized, Linux based public and collaborative)

1 : “Safe & Secure” Boot Loaders



Why Special Boot loaders ?

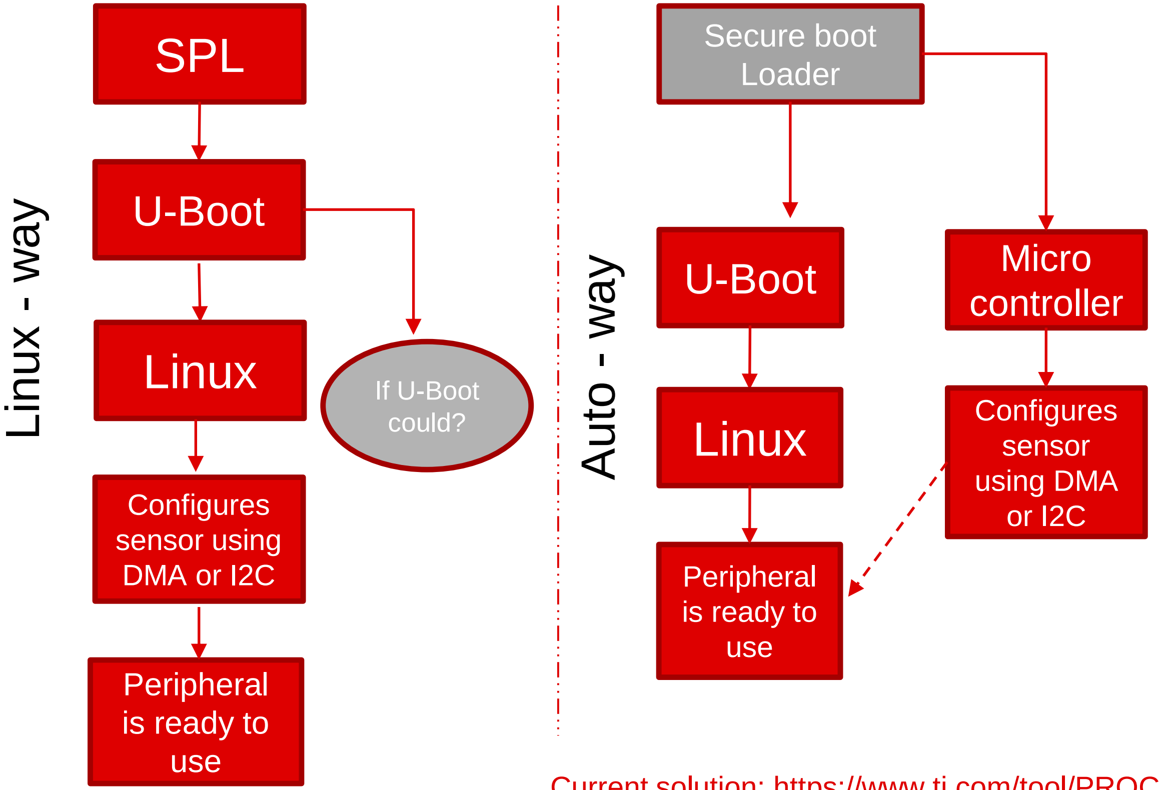
- SPL is not tuned to required performance (boot < 10 msec)
- SPL is not easily structured to boot remote heterogenous cores (at least for TI)
- SPL doesn’t meet safety compliance (TUV certified) with MISRA C and LDRA tool compliance.
- SPL is more prone to security vulnerabilities (as per industry stalwarts)
- SPL to Linux handoff need tweaks (peripherals already configured by SPL before Linux)

Is public open source SBL an option ?

- TI has public SBL, open for community collaboration & we can commit to safety qualification of software.

TI SBL Public Sources : https://github.com/TexasInstruments/mcupsdk-core/tree/next/examples/drivers/boot

2 : Devices (display, camera, sensors) in Action instantly


Linux - way
SPL
U-Boot
Linux
If U-Boot could?
Configures sensor using DMA or I2C
Peripheral is ready to use
Auto - way
Secure boot Loader
U-Boot
Micro controller
Linux
Configures sensor using DMA or I2C
Peripheral is ready to use

Why Configure early ? And why DMA / I2C ?

- Few sensors have more than 4K registers.
- Registers are configured over I2C (non contiguous) or DMA.
- Device should be ready before Linux drivers and apps are up. Can't spend time after boot.
- Current solution uses MCU - brings in safety compliance but increases the cost of SOCs.
- Linux late attach : while MCUs have performed all the initializations and Linux takes over, the handoff isn't clear for every driver (example: simple framebuffer)

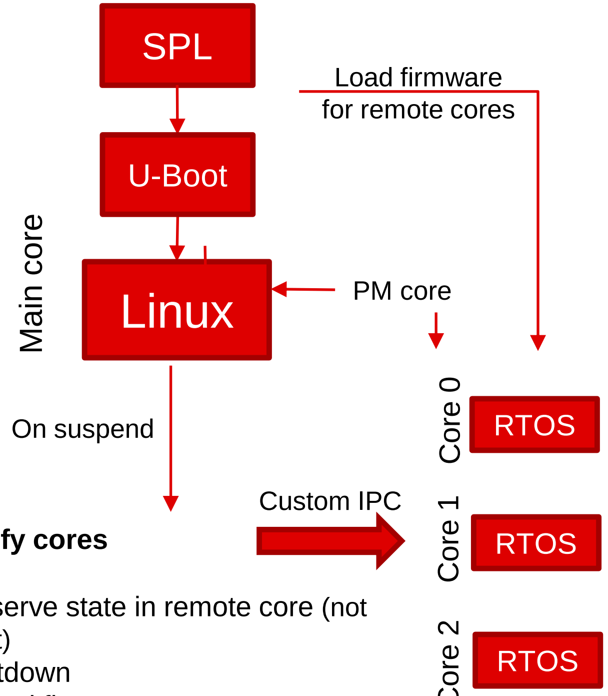
What's the long term solution ?

- Make U-Boot / SPL multi-threaded ?
- If DMA/I2C triggered from U-Boot, we need a standard method to release, reallocate the channels, memory region.
- How to utilize the multiple "A"-cores

Current solution: https://www.ti.com/tool/PROCESSOR-SDK-J721E

3 : Power management with remote cores

Linux - way



- **Notify cores**

- Preserve state in remote core (not strict)
- Shutdown
- Reload firmware on resume

Auto - way

- Mimics the Linux way
- Isn't sufficient with Industry ask especially with EV picking up in every segment.
- Every SOC company will define their own framework, tools and publish "incomparable" results

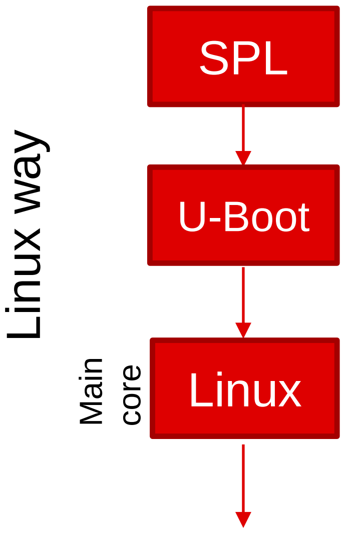
Power Management and handle remote cores.

- **No standards** defined for notification
  - ex: how long to wait, min/max expectations from remote core after notify, etc..
- Every **reload of firmware costs extra cycles** for authentication of firmware - impacts resume latency numbers.
- Cores are turned on/off abruptly, the states aren't preserved before suspend**, left for RTOS world to decide**.
- **Scaling frequency up/down dynamically** need further notification mechanism which isn't available for remote cores.
- There are multiple different modes (other than just deepsleep, stndby, etc) where **Linux user space hooks are missing**.

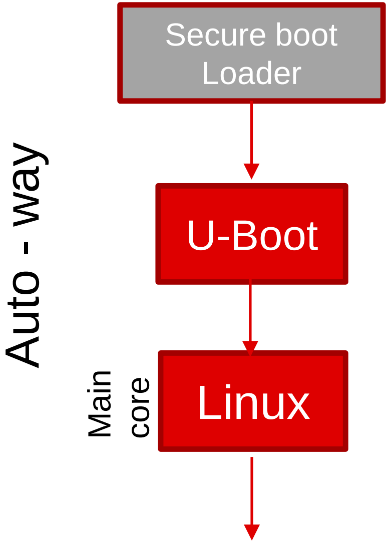
What's the long term solution ?

- Need an Industry standard – RTOS and Linux community should collaborate and engage in defining this standard.
- Benchmarking tools should be made available.

4 : Early Ethernet / Connectivity Notifications



- CPSW Probe
- System-networkd
    - **MAC open**
- Phy Link Up



- CPSW Probe
    - **MAC open**
- System-networkd
- Phy Link Up

Improvise Phy Link up time :

- Improvement because the MAC port open function call is pushed into probe from system-networkd
- Phy link up time depends on the phy and its configuration used. It varies from boot to boot. This was the best time.

Component	**MAC open in networkd**	**MAC open in driver probe**
Kernel MAC port open	~ 920 ms	~10 ms
Phy link up	3100 ms	~ 1300
Total Boot time *	**2.5 to 3 seconds can be saved.**	

Long terms solutions

- What's alternative to MAC open in probe ?
- CAN has been left to the mercy of AutoSAR – No Linux/SPL possibilities for early CAN response < 50ms
- Ethernet stack require tweaks for network boot, packet handling by firmware on MCUs – need an upstream path.

Let’s not Conclude – let’s discuss

- Other questions :
  - Has Android automotive OS solved the issues being discussed here – No. Are they applicable there as well ? Yes.
  - Key question that pops up: What happens when Linux kernel crashes ? Why is this still a doubt ? How to harden Linux enough, what other constraints to impose on application/user space to gain the confidence.
  - Is ELISA the forum for any standardized mechanisms to implement these hacks and fixes in a standardized way ?
- How we get safety certification out of the way for SPL, U-Boot, ATF and Linux subsystems.
- Are there more such fixes required at product level that needs to be further discussed.
- Looking for a forum where we discuss this beyond respective kernel mailing list.

If interested to collaborate and work with us on these initiatives : khasim@ti.com

Thank you.

Contact Information:

- khasim@ti.com
- nsekhar@ti.com
- j-keerthy@ti.com
- vigneshr@ti.com
- srk@ti.com

Collaborate with us @

- https://www.ti.com/linux
- http://opensource.ti.com/
- https://www.ti.com/processors
- https://www.ti.com/edgeai
- https://github.com/TexasInstruments

Thanks to open source solutions and partners

Baylibre Konsulko Group beagleboard.org THE LINUX FOUNDATION Linaro

www.ti.com/sitara