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.
  - 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):

- CAN response < 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 ...

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.

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 ...

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

Power Management
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.
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
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

Power management with remote cores

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

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, stdby, 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

**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.

<table>
<thead>
<tr>
<th>Component</th>
<th>MAC open in networkd</th>
<th>MAC open in driver probe</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel MAC port open</td>
<td>~ 920 ms</td>
<td>~10 ms</td>
</tr>
<tr>
<td>Phy link up</td>
<td>3100 ms</td>
<td>~ 1300</td>
</tr>
<tr>
<td><strong>Total Boot time</strong></td>
<td><strong>2.5 to 3 seconds can be saved.</strong></td>
<td></td>
</tr>
</tbody>
</table>

**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.

---

**Linux way**

- SPL
  - U-Boot
  - Secure boot Loader
  - Linux
    - CPSW Probe
    - System-networkd
    - Phy Link Up

**Auto - way**

- SPL
  - U-Boot
  - Secure boot Loader
  - Linux
    - CPSW Probe
    - MAC open
    - System-networkd
    - Phy Link Up
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

www.ti.com/sitara