Early ethernet with Linux

Keerthy J
Tanmay patil
Siddharth Vadapalli

Texas Instruments

About us

- Keerthy - Software Applications Engineer at Texas Instruments – Mainly focused on boot time optimizations & board bring ups with Linux
- Tanmay - Software Applications Engineer at Texas Instruments – Focused on CPSW ethernet on Linux
- Siddharth – Linux open source engineer mainly contributing on ethernet, USB & PCIe.
- *Presentation is based on learnings we had while optimizing ethernet boot time on DRA821/J7200 platforms with Linux/RTOS.*

Texas Instruments

Agenda

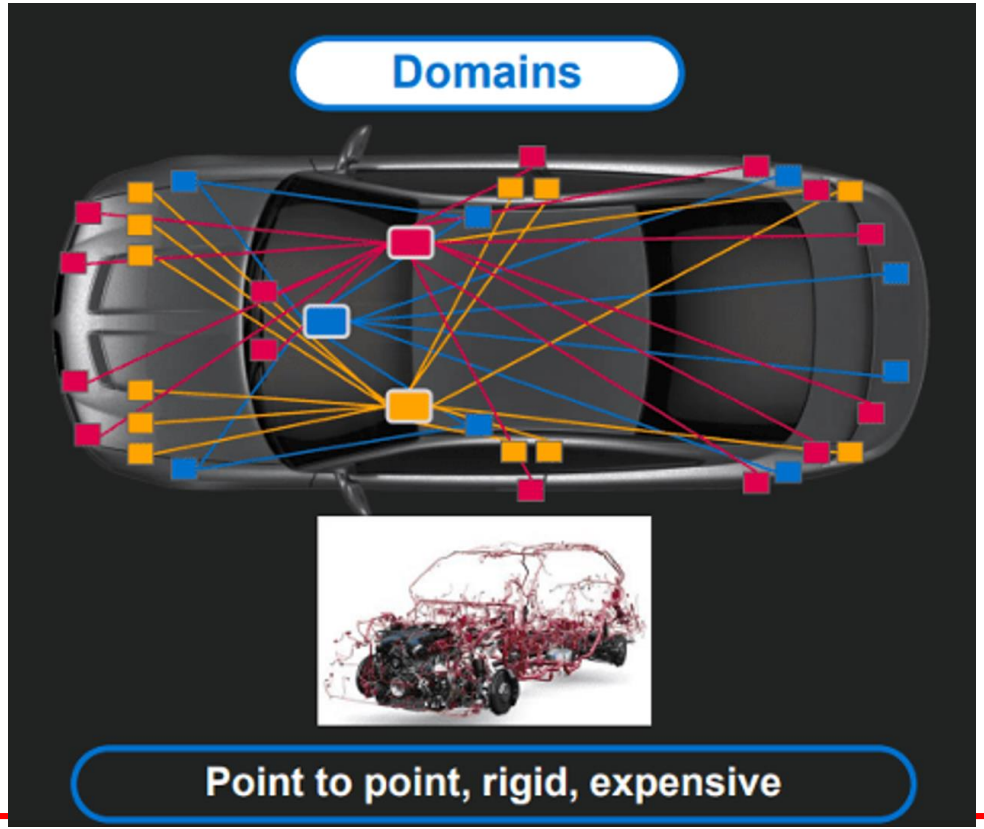
- Why do we need to start ethernet early?
- What?
  - Problem Statement
  - Gateway SOCs – An introduction to DRA821 SOC as example.
  - Default boot flow
- How?
  - Impact of boot media
  - Optimizations at various boot stages:
    - Boot loader
    - Linux
    - File system
  - Redesigning ethernet use case
    - Using Remote cores to initialize ethernet in parallel
    - Using proxy driver in Linux for ethernet
- Results
- Summary

Texas Instruments

Why do we need ethernet up early in automotive?

Texas Instruments

Domain Based architecture

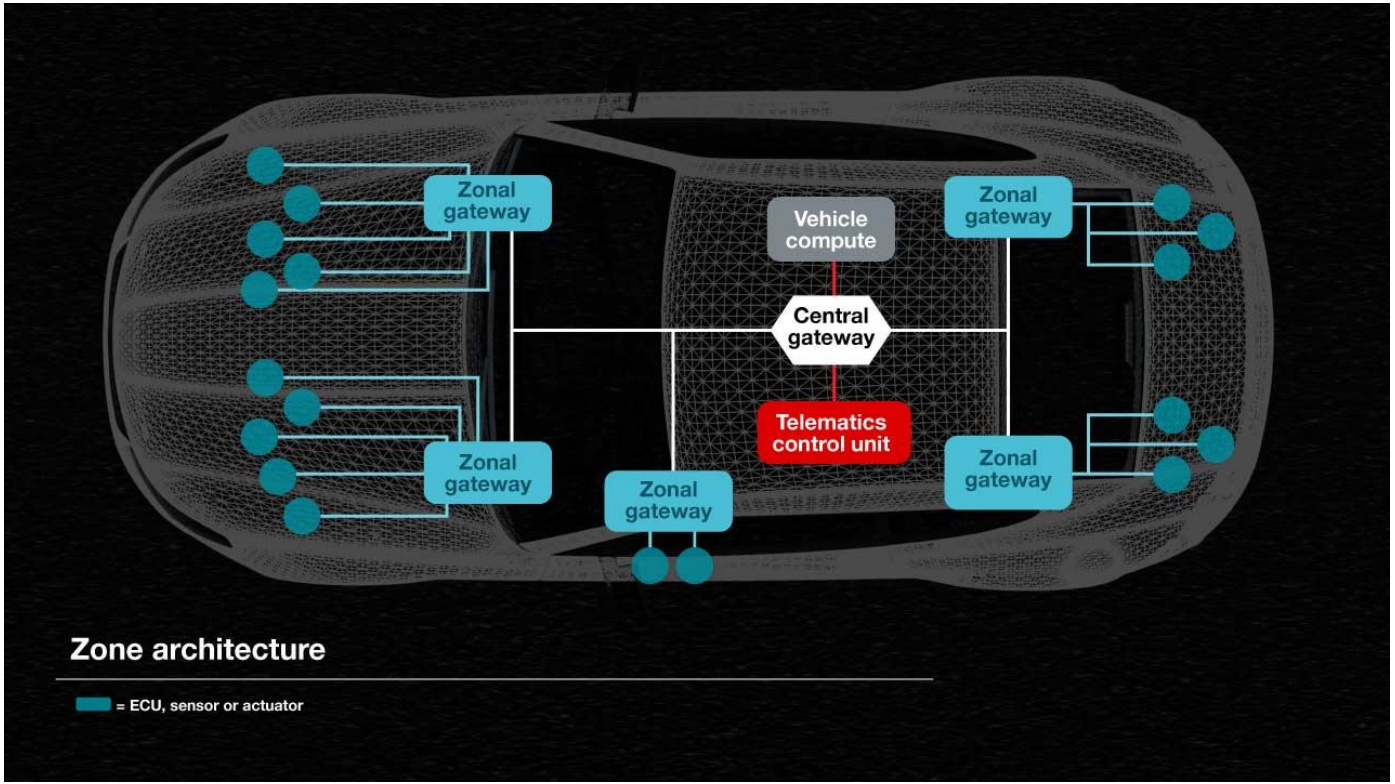


Texas Instruments

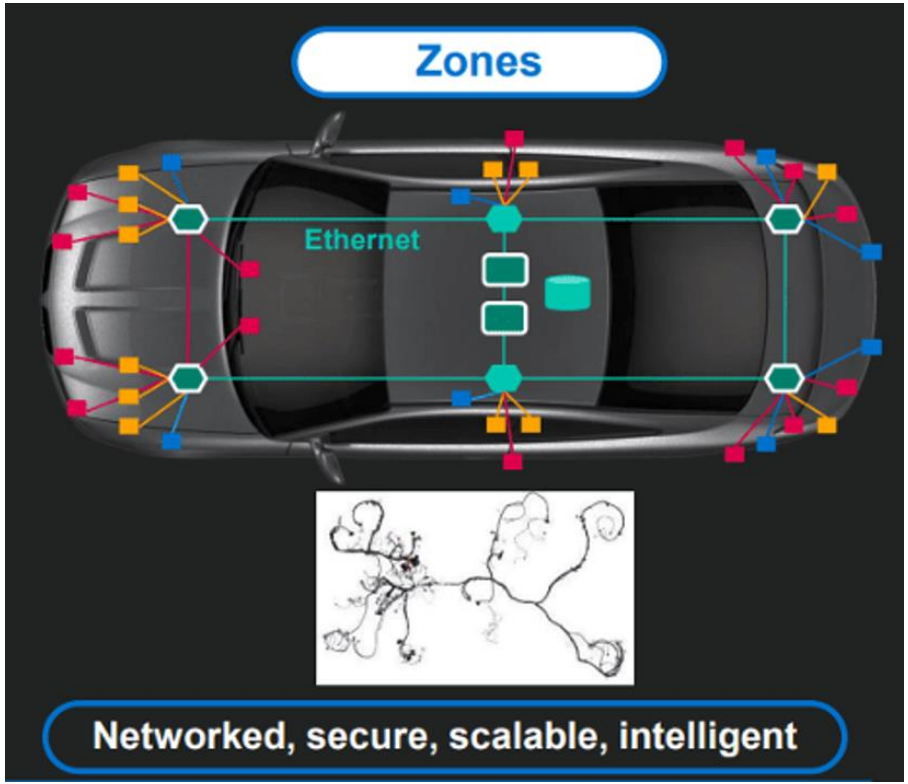
Domain based architecture in the automotive industry

- Domain based architecture, ECUs are categorized into domains based on their function.
- The vehicle network has evolved into a domain architecture where the ECUs are grouped based on a related function that each ECU is performing.
- This has increased network complexity
- Automotive vehicle designers are now considering the use of a zone-based architecture, which offers the advantage of having a vehicle compute module to control vehicle functions.

Zonal Architecture & Gateway SOCs



Zone architecture



What do we need?

Texas Instruments

Problem statement: Early ethernet availability

Requirements:

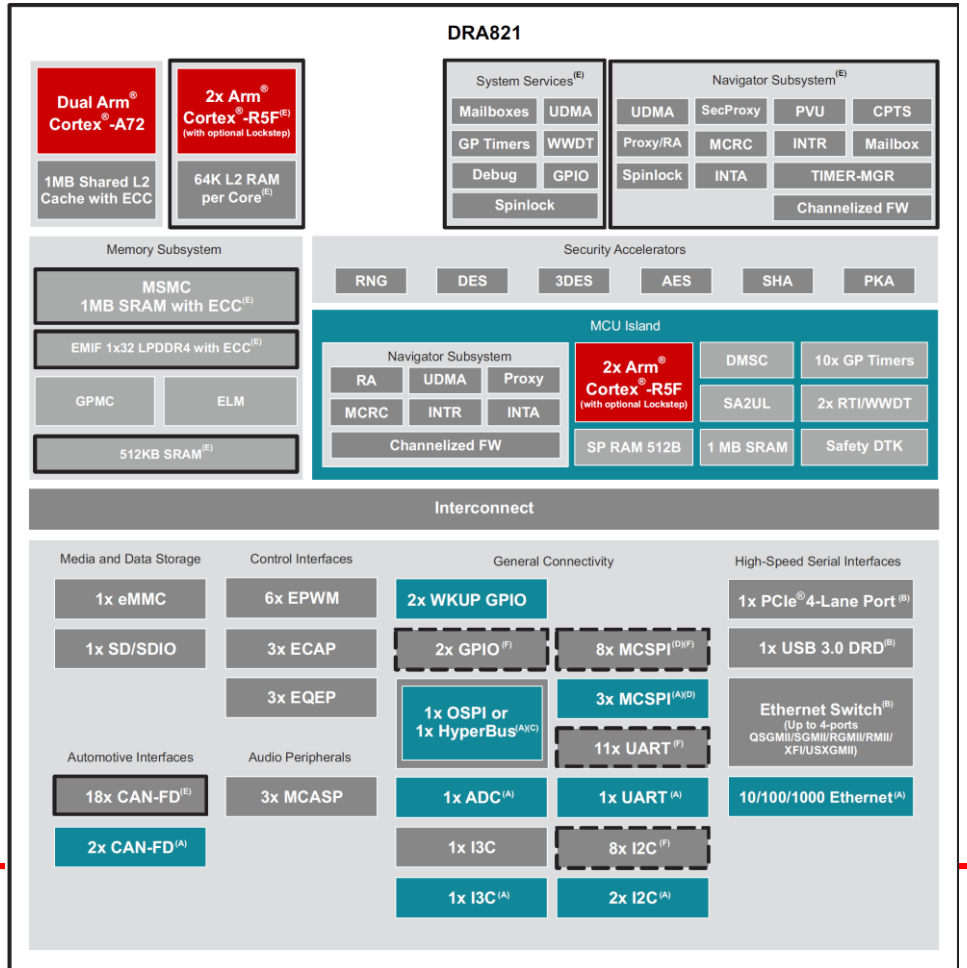
- Operating system: Linux running on the Application cores.
- Optimized boot flow
- Heterogeneous cores executing in parallel

Ethernet needs to be available in 2-3 Seconds for timely communication. To realize some of the common use cases below.

- OTA
- Inter ECU communication
- Telematics data relaying

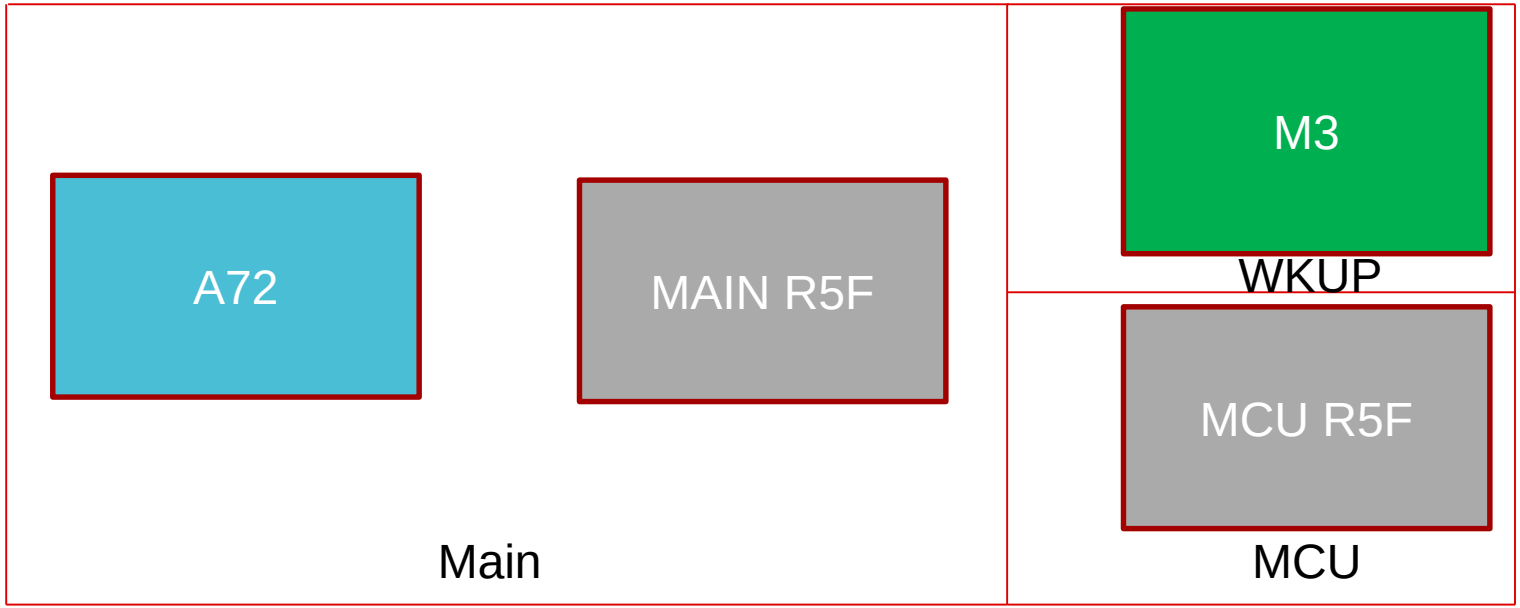
Texas Instruments

Gateway SOC – An introduction to DRA821 SOC



Texas Instruments

Cores Vs OS

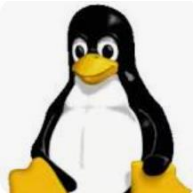


FreeRTOS (Ethernet Firmware aka EthFw)

Linux

TI Foundational Security

Linux on A72: Primary responsibilities



- Gateway Applications on A72/Linux are very data centric
- Storage: The SD/eMMC driver manages the secondary storage.

- Networking: The CPSW IP provides the network interface & Linux has native driver to initialize & control the network layer.
- Remoteproc drivers to accomplish IPC between R5F/M3 for security algorithms.
- Console – All the Application logs from all the remote cores come on the console owned by Linux.

Texas Instruments

Cortex M3: TI foundational security

The Security controller is the M3 subsystem in the TDA4 family of devices is responsible to run the software stack for the device foundational security management. This software entity is responsible for:

- Foundational device security
- Secure boot with secure keys/root-of-trust, Security configuration and Debug unlock
- RSA or ECC Root Keys, AES symmetric key
- Utilizes PKA, SHA2 and AES Crypto accelerators
- Basic security functionality to extend root of trust (optional)
- Authenticated Key ring to extend root-of-trust keys
- SYSFW and Bootloader Rollback protection via eFuse
- Device Unique Key / Key Derivation to support 3P stacks

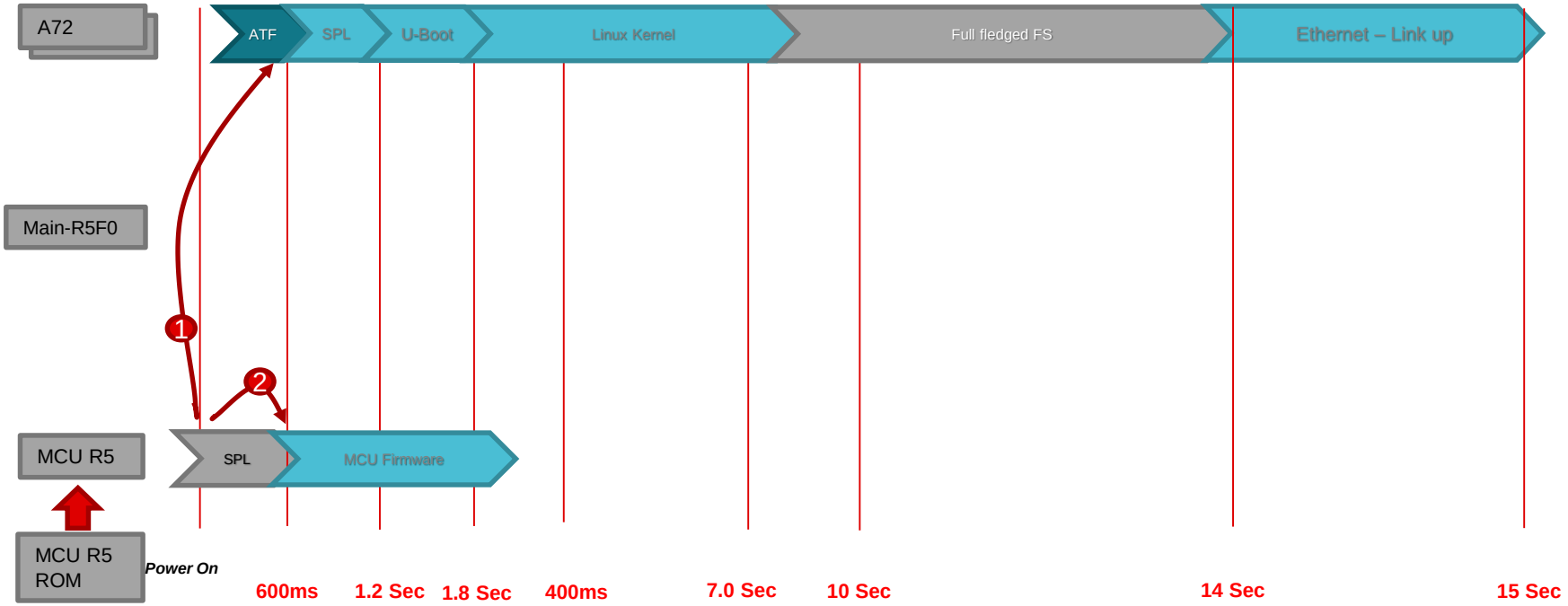
Texas Instruments

MCU R5F: Device Manager

- This is an RTOS task
- Centralized Power & Resource manager.
- All the Clock/module requests will go through MCU R5F.
- Book keeping is done by the device manager running on MCU R5F.

Texas Instruments

Default boot flow(MMC-SD as boot media)



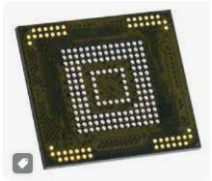
How? do we optimize

Texas Instruments

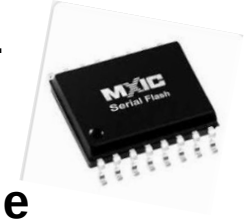
Impact of boot media

The choice of boot/filesystem media is critical.

- MMC-SD: This is cost effective and size will be big but the slowest. Not recommended for early use cases.



- eMMC: This is a fast boot media and typically larger in size and can do both boot loader hosting as well as the filesystem hosting.



- OSPI/xSPI: This is the fastest amongst the mentioned boot media. This is much smaller in size compared to SD/eMMC.

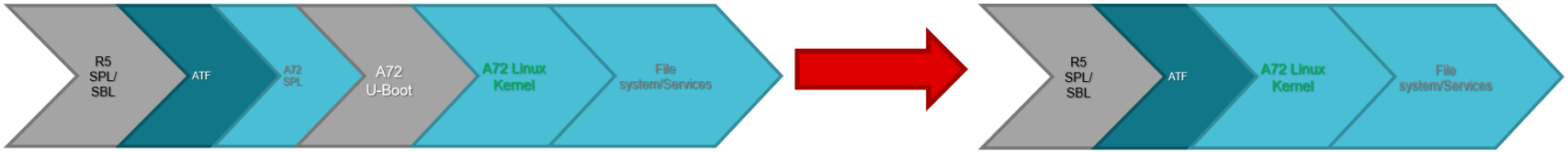
Typically OSPI as boot media and eMMC hosting the file system is a preferred choice for gateway use cases.

Generic Linux Bootflow optimizations

Texas Instruments

Tweaking the boot flow

- ROM code is the primary boot loader which loads secondary boot loader.
- Secondary boot loader could be the U-Boot based SPL(Secondary program loader) or the custom SBL(Secondary boot loader).
- Instead of going all the way till U-Boot(tertiary boot loader) to load all the remaining cores, we load the other cores at the SBL/SPL stage itself.



Texas Instruments

Optimizations - Bootloader

- Secondary boot loader optimizations:
  - Disable all the console prints.
  - Start remote core firmware loading as soon as the DDR is initialized.
  - Firmware loading starts as early as 100 milli Seconds
  - OSPI boot will further speed up the booting against MMC-SD

  The above optimizations saved about 1.5 Seconds

Texas Instruments

Optimizations – Linux/DTB

- Disable drivers that are not used to make the kernel image lean & mean. Reduced kernel loading time.
- Add loglevel=0 parameter to bootargs to suppress all the kernel prints.
- Switch the file system from MMC-SD to eMMC.
- Disable all the device tree nodes that are not needed for the use case
  - USB, PCIe, UARTs, SPI, Serdes to name a few that are not needed.
  - MMC-SD probing adds additional 1S so disable that as the last step once the file system is switched to eMMC
  - Switch the file system from MMC-SD to eMMC.

- **The above optimizations saved 6-7 Seconds to get to file system.**

Texas Instruments

Optimizations – Filesystem

- Switch to tiny rootfs from full fledged file system.
  - Full file system is of the order of GBs due to which loading file system takes a long time.
  - We can save more time by avoiding all the non-essential file system services initialization
  - Tiny rootfs is a bare minimal busybox based file system, does not contain multiple libraries that are essential for the vision_apps framework & graphics modules to be functional.
  - Copy /lib & /usr/lib contents from full fledged FS to the corresponding folders in tiny rootfs
  - Copy the above prepared SD card rootfs to eMMC.

Texas Instruments

Ethernet specific Optimizations

Texas Instruments

Auto-neg what/why is it?

- Ethernet devices can link up with different combinations of link rates and duplexity. But the wire encoding for all the different combinations differs and hence devices cannot communicate with the lwo link-partners in differing configuration
- Auto-negotiation, defined in IEEE 802.3 (clause 28), specifies a signaling mechanism for ethernet devices to advertise its capabilities to the link partner
- Auto-negotiation helps standardizing the connection mechanism between any two ethernet devices
- In Automotive setup we don’t really need Auto-neg as the devices are fixed and do not change run-time.

Impact of Ethernet Phy Configuration

To have a faster ethernet link-up, disable/skip auto-negotiation

- Auto-negotiation takes considerable time (2s normal for DP83869) to settle down on a specific link configuration.
- Hence, wherever possible, we can disable autonegotiation and configure both link-partners in a known configuration or use a fixed-link
- This will reduce the link-up time for phy from RESET to ~200ms (based on DP83869)

Texas Instruments

Optimizations – CPSW and phy Driver

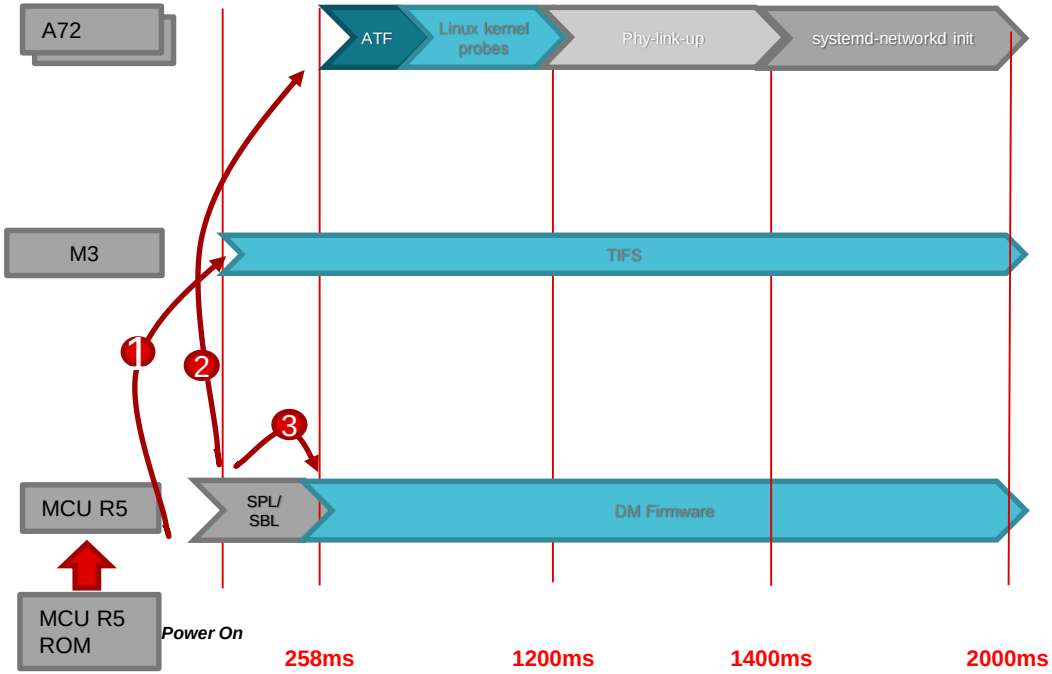
- CPSW driver waits for the filesystem to be loaded to start configuring ethernet MAC and phy.
- Move the dev_open call to CPSW probe to get the MAC and phy configuration started. This will move-up the link-up negotiations with link partner by ~2s
- This will allow the external devices(As the switch is functional at this point) to communicate via the Switch.
- For the initial phy link-up, change the frequency of phy state machine to run at 100ms instead of 1s to detect the link earlier.

Optimised Results with Native Linux Driver

Boot stage	Time (milli Seconds)	Comments
SBL(R5)	258	
ATF + Kernel probes (A72)	936	MAC port configuration is complete
Wait for phy link up (A72) (Automotive phy)	200	Forwarding through switch possible at this point
Initialisation using systemd-networkd (A72)	~600	Switch configuration is complete. Ping with Linux working
Total	**~2000**	

Texas Instruments

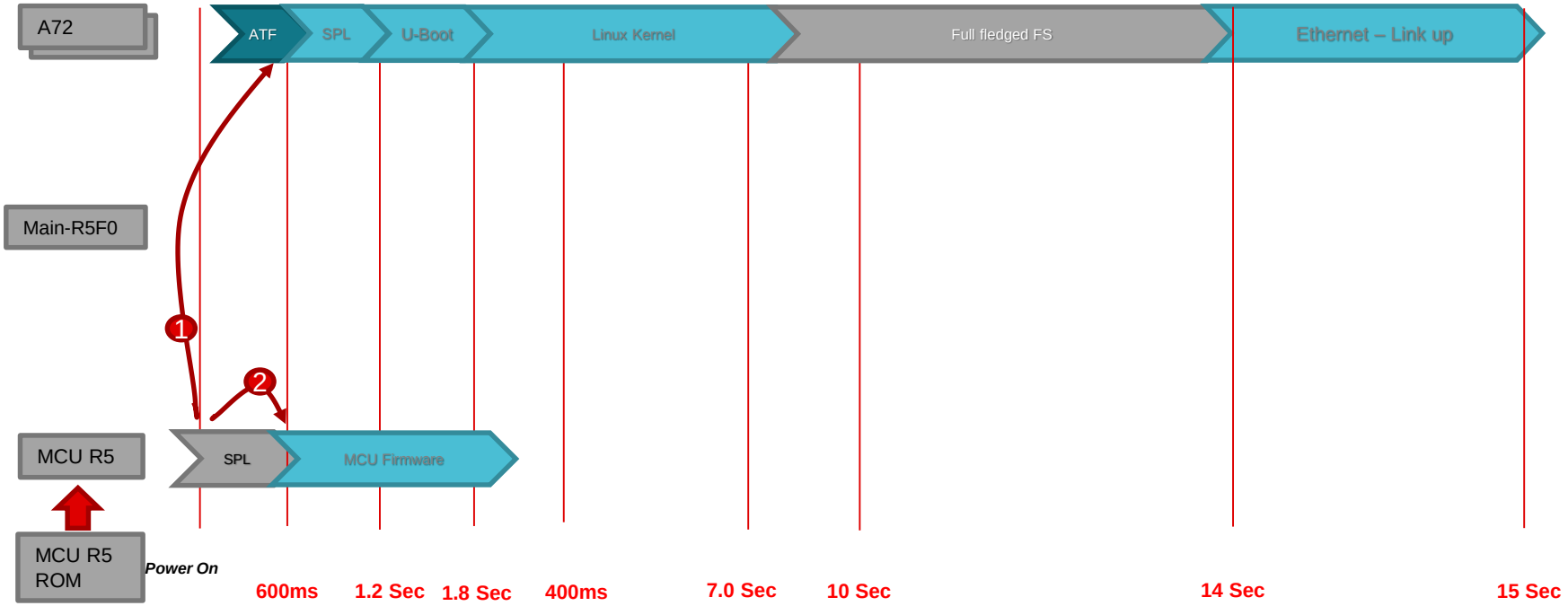
Timing Diagram for Native Linux Driver



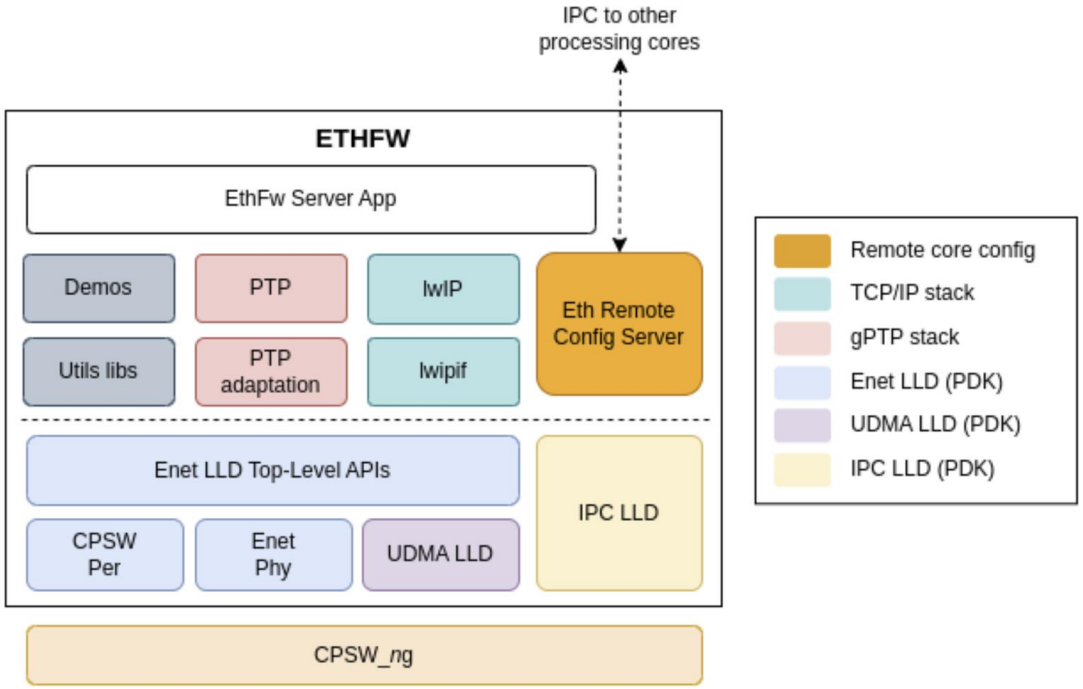
How? To optimize even further

Texas Instruments

Default boot flow(MMC-SD as boot media)



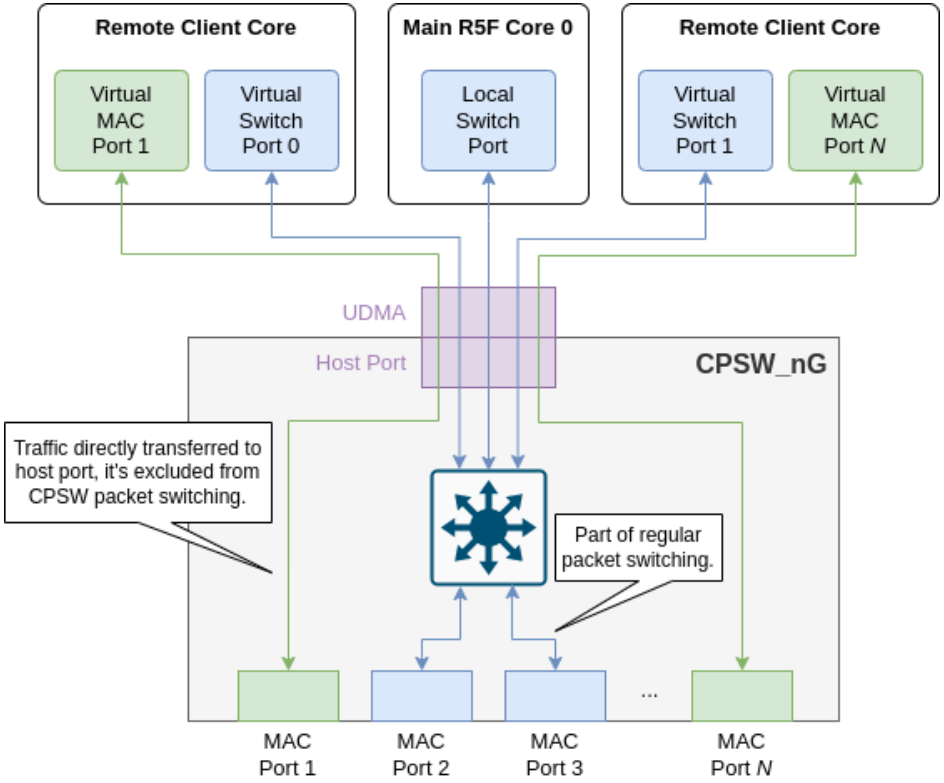
EthFw (Ethernet Fimrware): Block Diagram



Ethernet Firmware software architecture

Texas Instruments

Overview of ethernet with EthFW



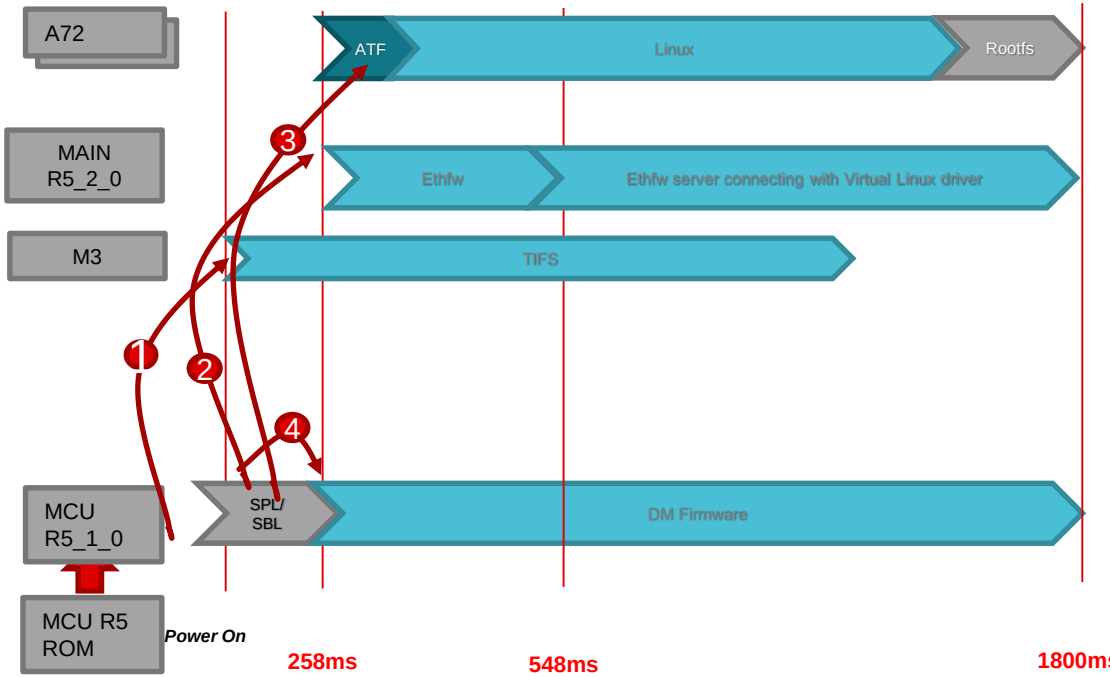
Texas Instruments

Optimised Results with Ethfw

Boot stage		Time (milli Seconds)	Comments
SBL(R5)		258	
Wait for phy link up (R5) (Automotive phy)	Initialisation using systemd-networkd (A72 running virt eth client driver)	200	Added Manual delay of 200ms to account for phy link up. No processor runs during this delay
Configuring the switch (R5)		**80**	**Switch configuration is complete**
		1600	Switch configuration is complete. Ping with Linux working
Total		**~1800**	

Texas Instruments

Timing diagram with Ethernet firmware and virtual Linux driver



Summary

- Optimizations possible at each phase of boot.
  - Recommendation to have the HW designed to take advantage of fast booting devices like OSPI, xSPI, eMMC.
  - Use multi OS strategy to run all cores in parallel
  - Optimize boot loader to load the critical cores first that are needed for the Use case.
  - Use EthFw in parallel to get maximum parallel processing from the cores in the SOC.
  - Prioritize ethernet initialization – Let the control core initializing switch first at boot.
  - Prioritize the systemd services to get the ethernet functionality first
  - Design your use case in such a way that cores always have something to run rather than wait for another core.

Texas Instruments

References

- EthFw user guide: https://software-dl.ti.com/jacinto7/esd/processor-sdk-rtos-j784s4/latest/exports/docs/ethfw/docs/user_guide/ethfw_c_ug_top.html
- DRA821/J7200 technical reference manual: https://www.ti.com/lit/pdf/spruiu1

Texas Instruments

Credits and Acknowledgement

- Texas Instruments Inc.
- The Linux Foundation.

- **E-mail:**
  - Keerthy J <j-keerthy@ti.com>
  - Tanmay Patil <t-patil@ti.com>
  - Siddharth Vadapalli <s-vadapalli@ti.com>