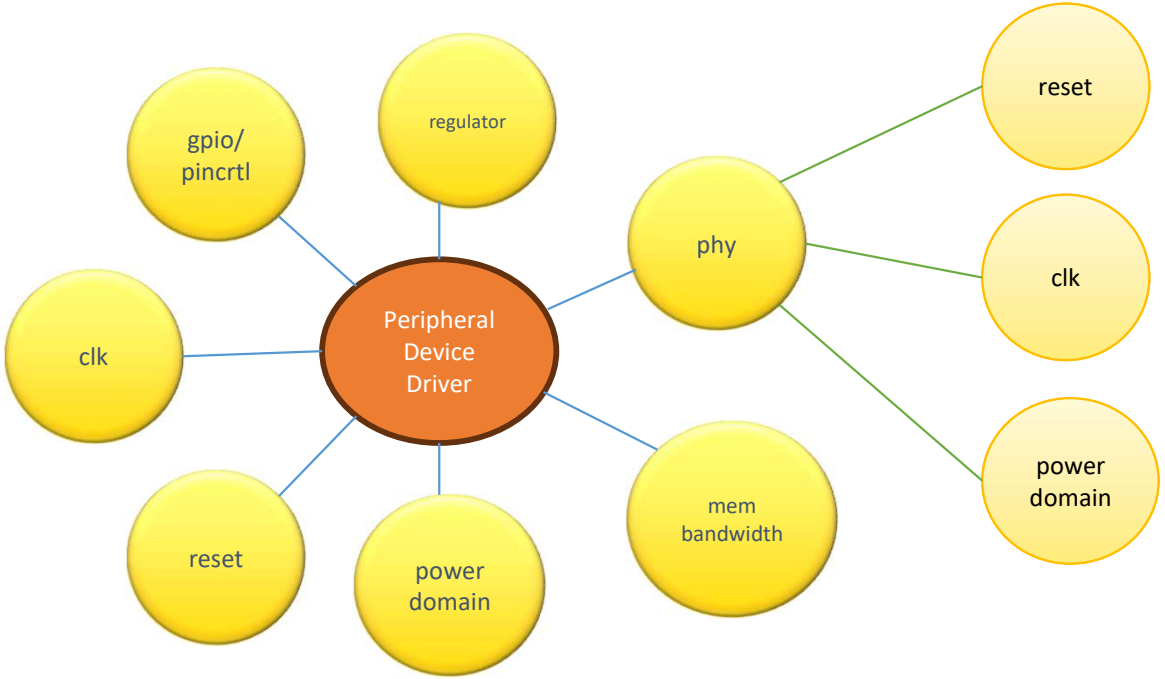
Abstracting platform resources using power & performance domains over SCMI

Nikunj Kela
Engineer, Senior Staff
Qualcomm Innovation Center, Inc.

Agenda

- Motivation
- Achieving abstraction using SCMI
- Our architecture overview
- Status and plan

Resources, a typical peripheral driver interface with

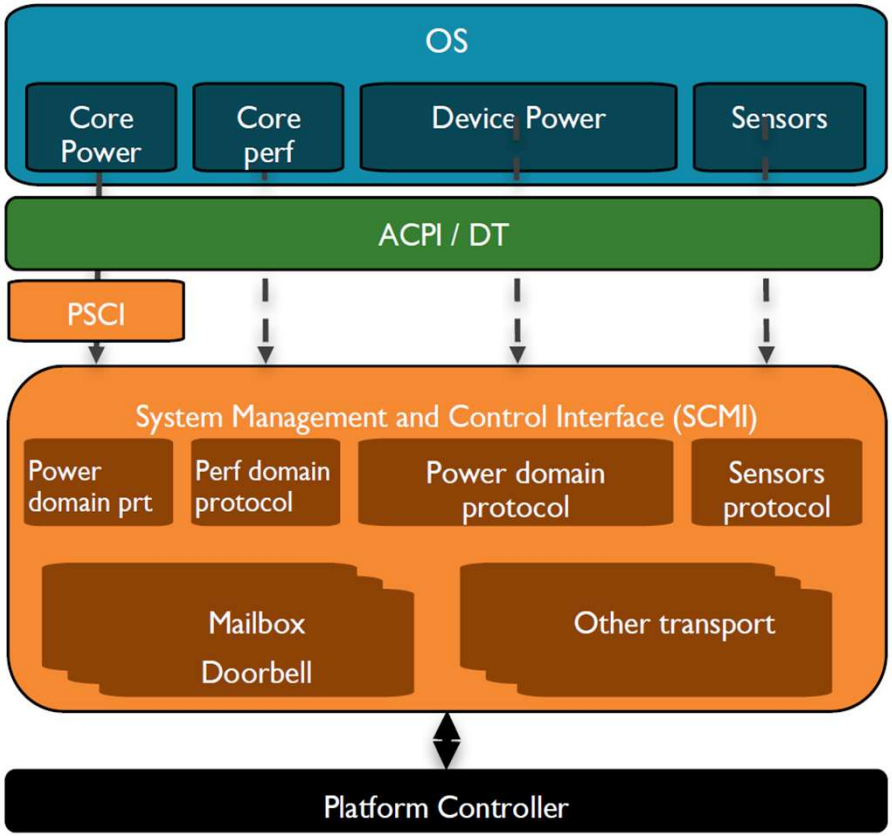


What are the problems?

- The Linux drivers become too complex
  - Need to interface with clock, regulator, interconnect, GPIO/Pin-control, PHY etc. frameworks
- Too many platform drivers needed for a single peripheral to work
  - Significant development efforts
  - Significant bring up efforts
  - Significant upstreaming efforts
  - Difficult to maintain so many drivers

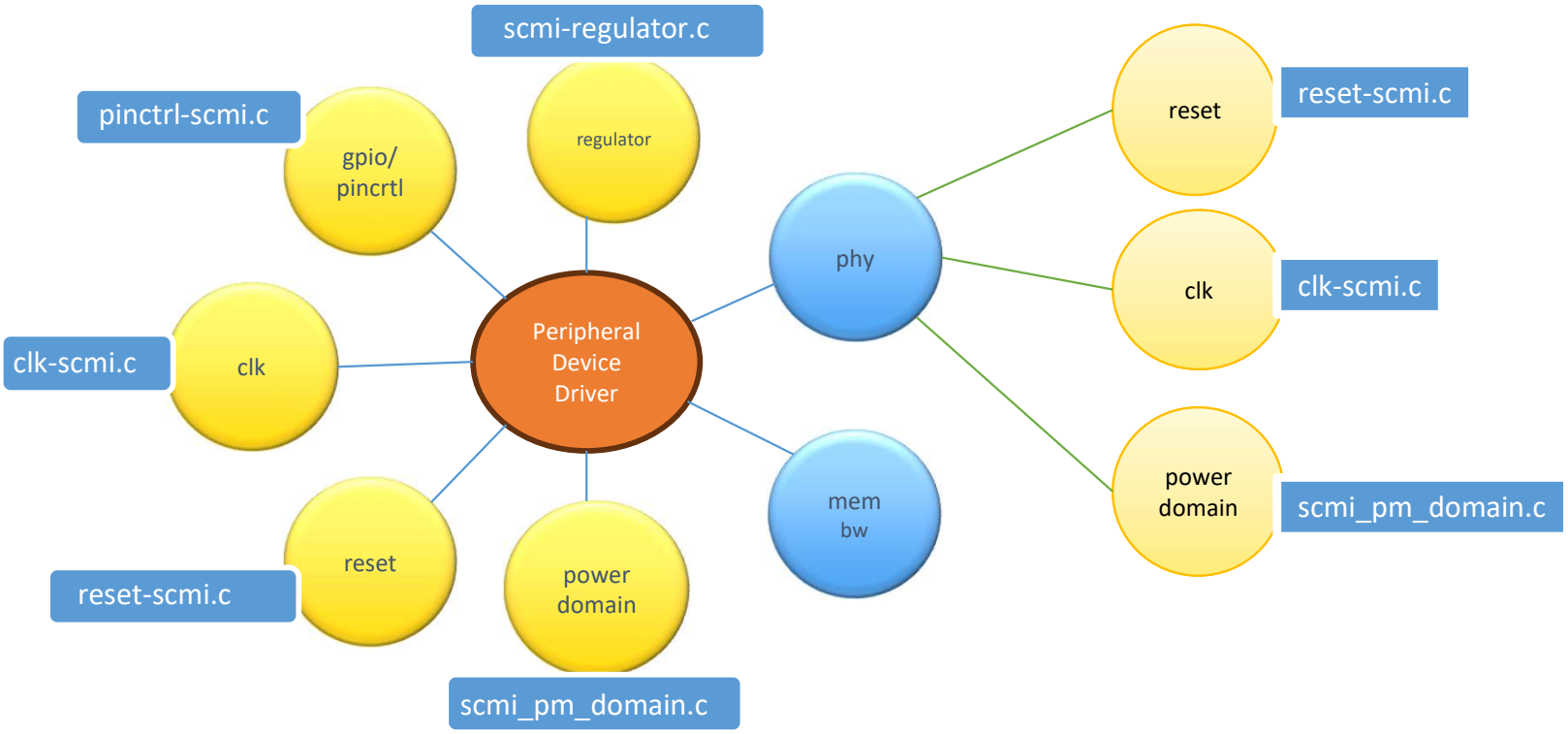
SCMI(System Control and Management Interface)

- A set of operating system-independent software interface that are used in system management
- SCMI is split into two layers
  - Protocols
    - The services you want to provide
    - e.g., clock, voltage, power-domain, reset, performance etc.
  - Transports
    - The mechanism by which the parameters/results are communicated between the caller of the interface(an Agent) and the implementer (the platform)
    - e.g. mailbox, smc/hvc call, virtio etc.

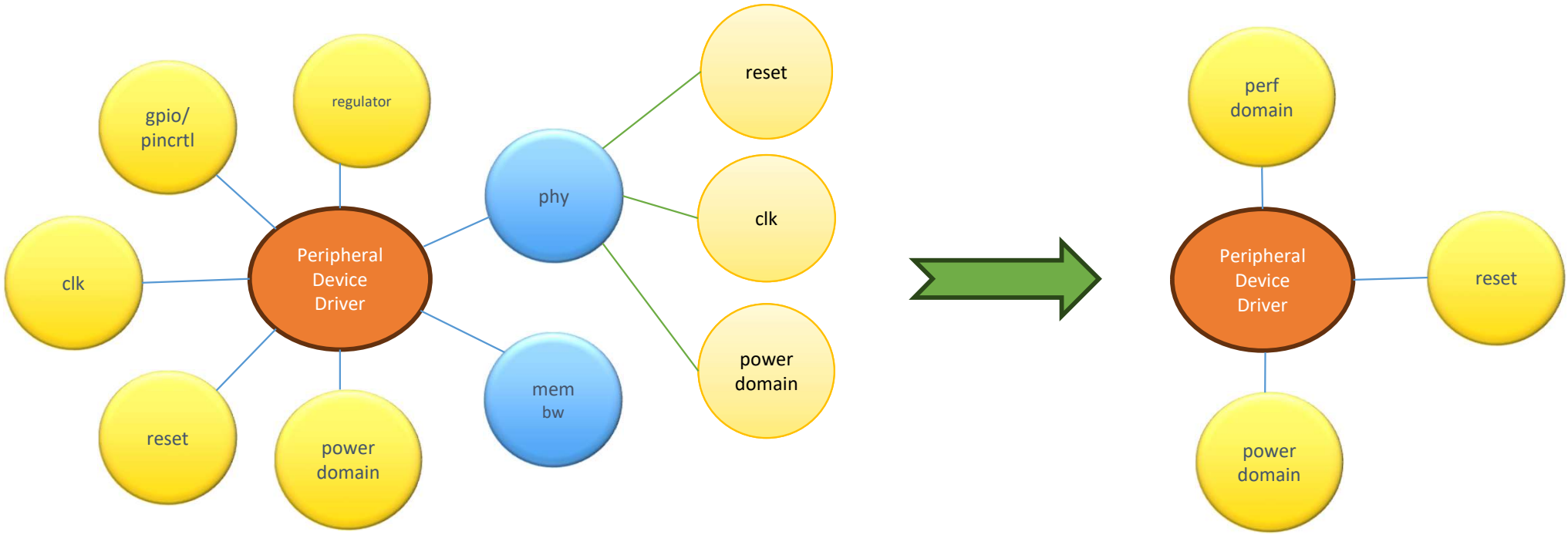


source: LinaroConnect 2016

SCMI to achieve abstraction

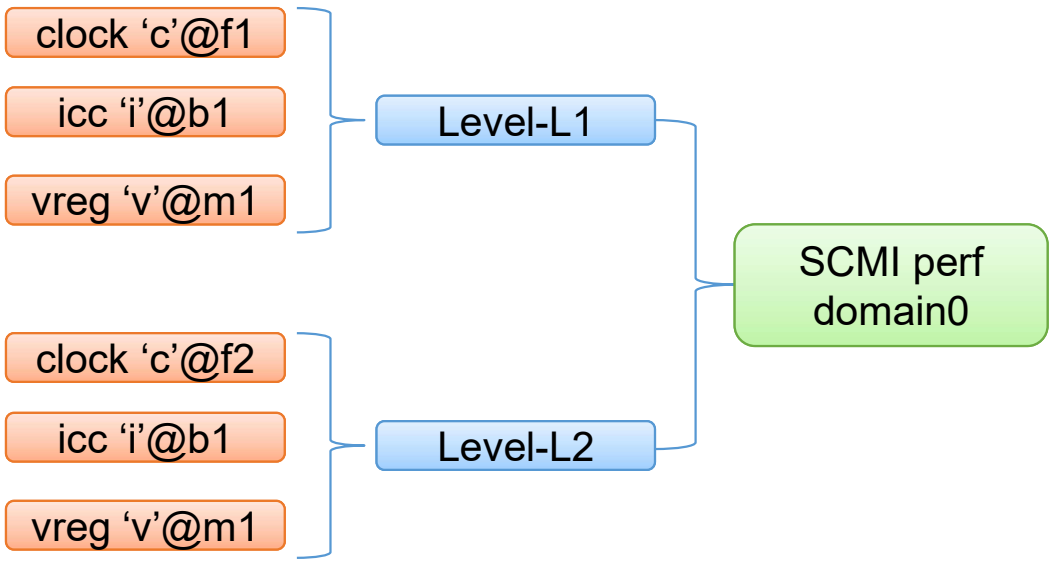


Can we do better?



Our approach

- Define logical/modelled power and performance domains
- Peripheral drivers can now request Firmware to configure device Performance Level thus avoiding multiple requests for individual resources
- Use SCMI power and performance protocols to achieve
  - Better abstraction (Interconnect Bandwidth and PHY can also be abstracted)
  - Reduce # of requests to Firmware

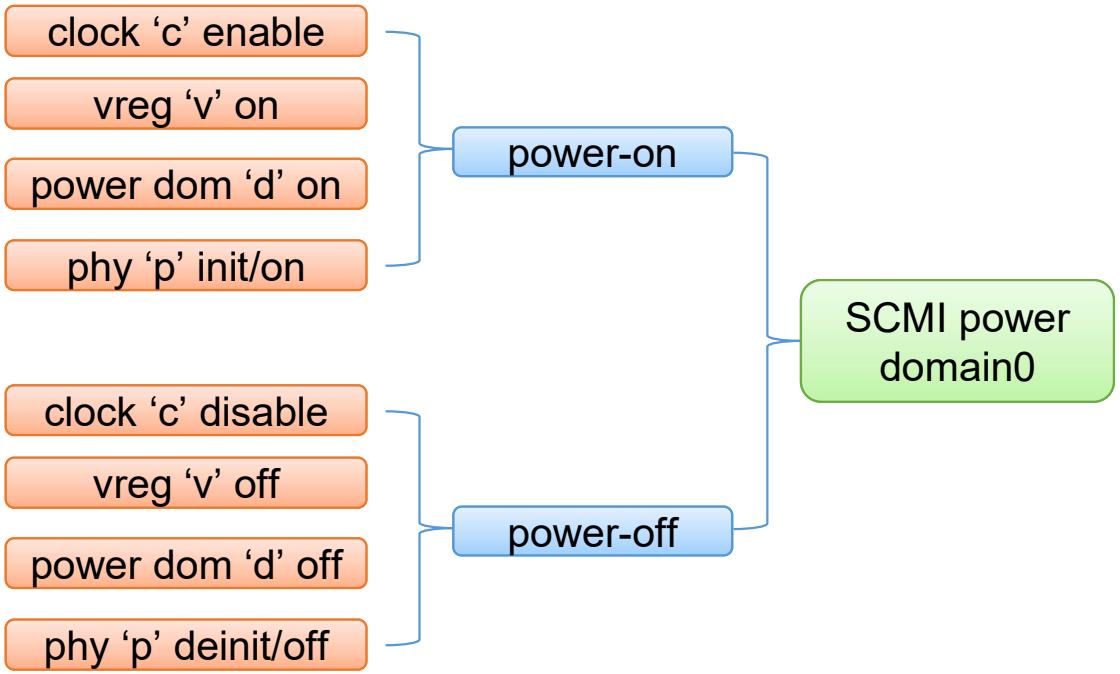


Continue…(Modelled performance domains)

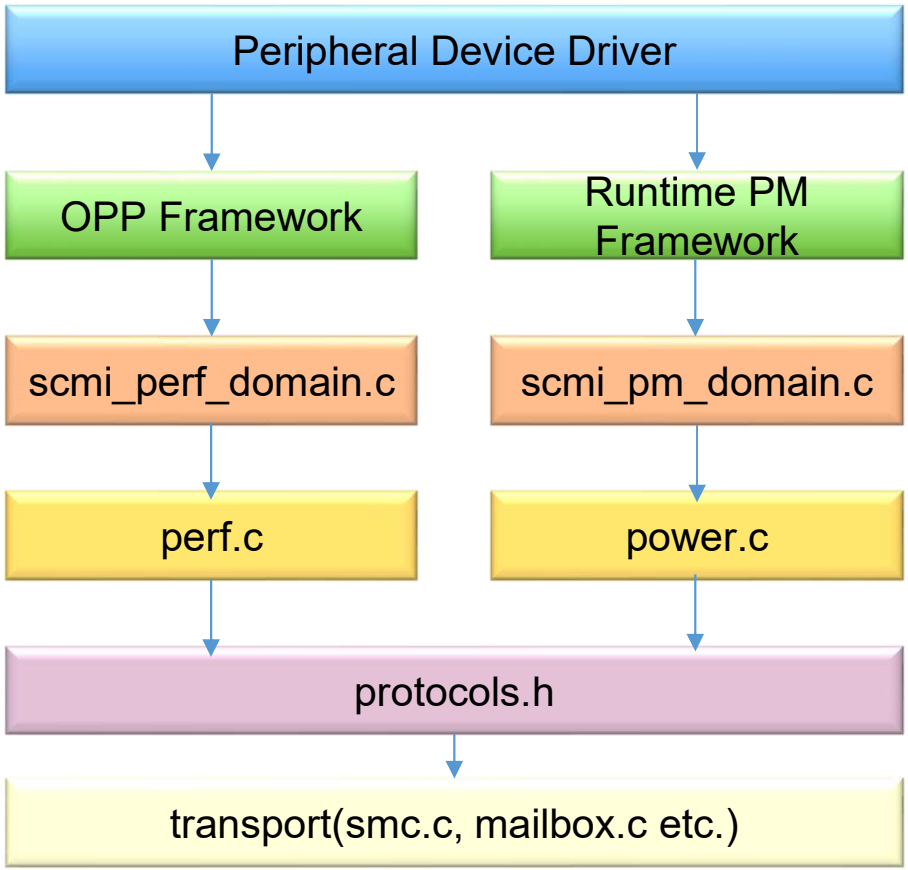
- In Linux, the performance domains have been used for compute(CPU) devices only so far
- We extend its usage to peripheral devices
- Define performance domains to perform
  - Dynamic clocks voting i.e. set_rate operations
  - Dynamic interconnect bandwidth voting
  - Dynamic changes to PHY modes
  - Dynamic changes to baud rates in case of UART
- Define performance levels for the peripherals
  - For baud-rates in UART, perf levels can be defined as
    - LEVEL_9600, LEVEL_115200, etc.
  - What these levels translate to is hidden in the Firmware

source: wikimedia commons
license: CC BY-SA 2.0 Deed | Attribution-ShareAlike 2.0 Generic | Creative Commons
no changes were made

Continue…(Modelled power domains)



Code Flow on Linux



drivers/pmdomain/arm/

drivers/firmware/arm_scmi

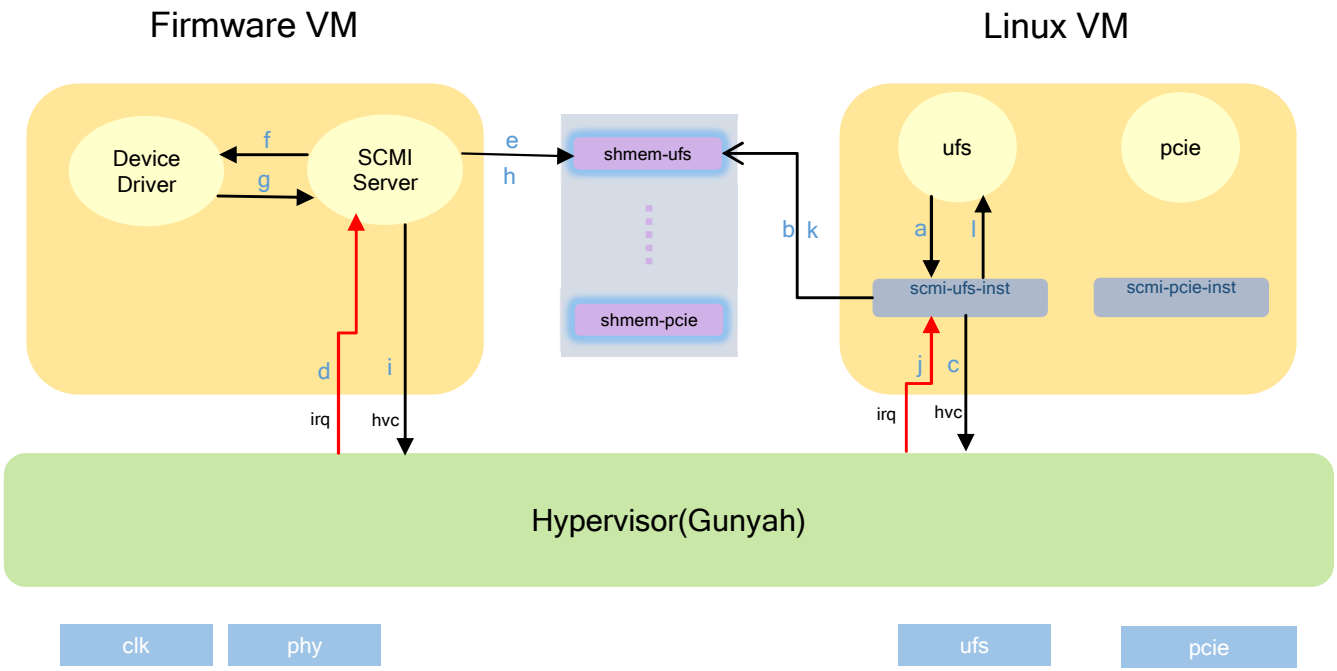
Changes pushed in Linux Kernel

- SCMI performance driver is available to work with peripheral devices by extending genpd framework [1]
- OPP framework is extended to work with SCMI performance driver [2]
- Helper APIs are now available so the consumer drivers can avoid the manual attach/detach in case of multiple power-domains [3]

[1]: https://lore.kernel.org/all/20230825112633.236607-1-ulf.hansson@linaro.org/
[2]: https://lore.kernel.org/all/20230925131715.138411-1-ulf.hansson@linaro.org/
[3]: https://lore.kernel.org/all/20240130123951.236243-1-ulf.hansson@linaro.org/

Our Architecture – Firmware Managed Resources

- Dedicated SCMI channels (shmem + doorbells) for each peripheral device
- FW VM has multiple vCPUs; the requests can be served in parallel



SA8775p SoC

How to figure fw-managed setup in drivers

- Since we are using power and perf protocol(and not clocks, voltage etc.), peripheral drivers need to know if running in a fw-managed setup or locally managed setup

1. Check on the presence/absence of the clocks(and/or other resources) in DT
    - **Pros**: don't need anything additional
    - **Cons**: may work for some driver; clock maybe optional on some platforms; may not scale; insufficient condition to ascertain fw-managed setup
2. Use a new compatible for each device(e.g. "qcom,ufshc-fw-managed")
    - **Pros**: cleaner and explicit way to determine fw-managed setup
    - **Cons**: HW IP is not changed so may not be warranted;
3. Use vendor DT property for devices (e.g. "qcom,firmware-managed-resources")
    - **Pros**: cleaner and explicit way to determine fw-managed setup
    - **Cons**: repetition of the same property in device tree nodes; vendor specific so 3rd party drivers may not accept it

Continue…

4. Use a vendor agnostic DT property for devices (e.g. “firmware-managed-resources”)
    - **Pros**: cleaner and explicit way to determine fw-managed setup
    - **Cons**: repetition of the same property in device tree nodes; different vendors may abstract differently
5. Use a board(platform)level compatible
    - **Pros**: common way to all devices
    - **Cons**: each driver will need to match for board(platform) level DT; might not scale

- Upstream discussion(not concluded) on this can be found [4]
- We will conform to upstream suggestions once the discussion concludes

[4]: https://lore.kernel.org/all/be31801e-bb21-426b-f7aa-2b52727de646@quicinc.com/

Status and plan

- Firmware VM side development is 90% complete
- Most of the Linux peripheral device driver changes are under internal review
- SCMI transport driver changes for Qualcomm hypervisor is available in upstream [5]
- We plan to send peripheral driver changes in upstream soon

[5]: https://lore.kernel.org/all/20231009191437.27926-1-quic_nkela@quicinc.com/


LinaroConnect
MADRID 2024 | MAY 12-17 2024
Thank you
MAD24-310