LCU14-210: Qualcomm® Snapdragon™ Processor Power Management - Unique challenges for power frameworks

Lina Iyer, LCU14

Staff Engineer, Qualcomm Innovation Center Inc.

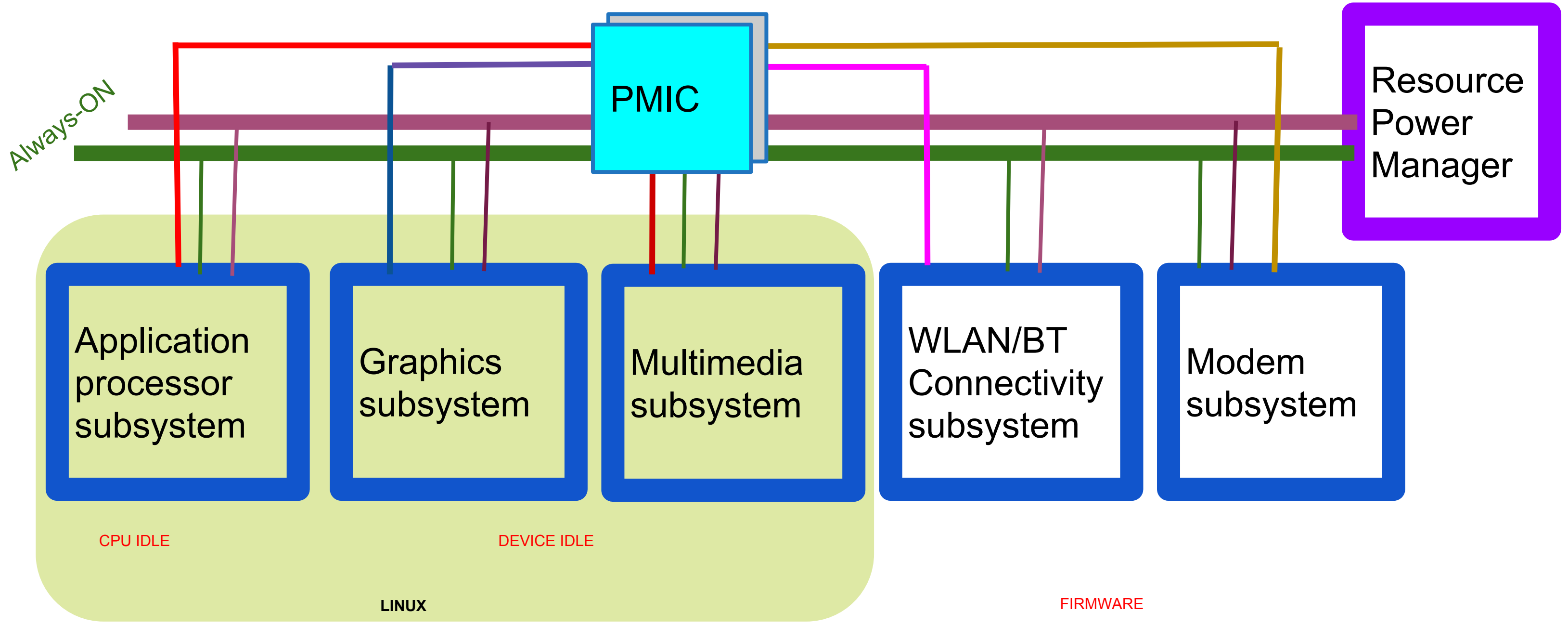
Overview

- Snapdragon SoCs and the Qualcomm® Krait™ CPUs
- Power Delivery Network (PDN)
- Resource Power Manager Processor (RPM)
- Idle States
- Subsystem Power Manager (SPM)
- Challenges and Solutions

Snapdragon SoCs And The Krait CPUs

- Snapdragon SoCs
  - Some have Krait 200/300 (8060, 8064)
  - Some have Krait 400/450 (8074, 8084)
  - Some have standard ARM cores (8x26, 8x10)
- SoCs with Krait cores have
  - Individual clock control
  - Individual power control for power control
- Hierarchical power states leading to SoC wide low power
  - CPUs idle => L2 idle => Cluster idle => SoC idle
  - Combination of idle states, yields idle states at the next level and so on..

Power Distribution



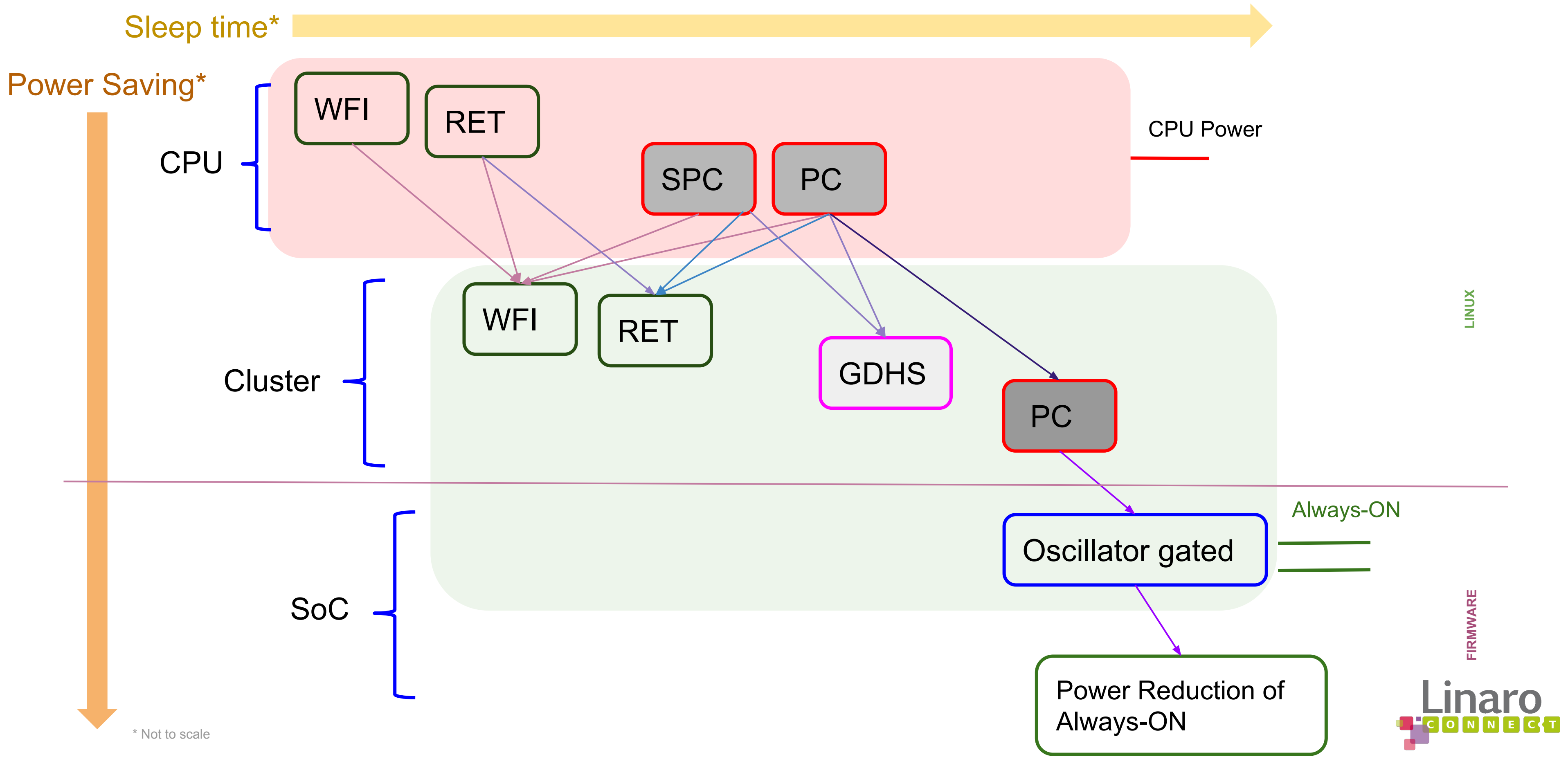
APSS Power Delivery Network (PDN)

- Application processor subsystem (APSS) controls its own power
  - APSS is the cores along with other peripheral components that support the cores
- Voltage on the rail is a combination of requirements for each core
- Caches are generally driven by SoC backbone rails
  - Enters lower voltage idle states when the SoC enters low power states
- Other system rails
  - Subsystem rails controlled by subsystem
  - Shared rails controlled by the RPM processor

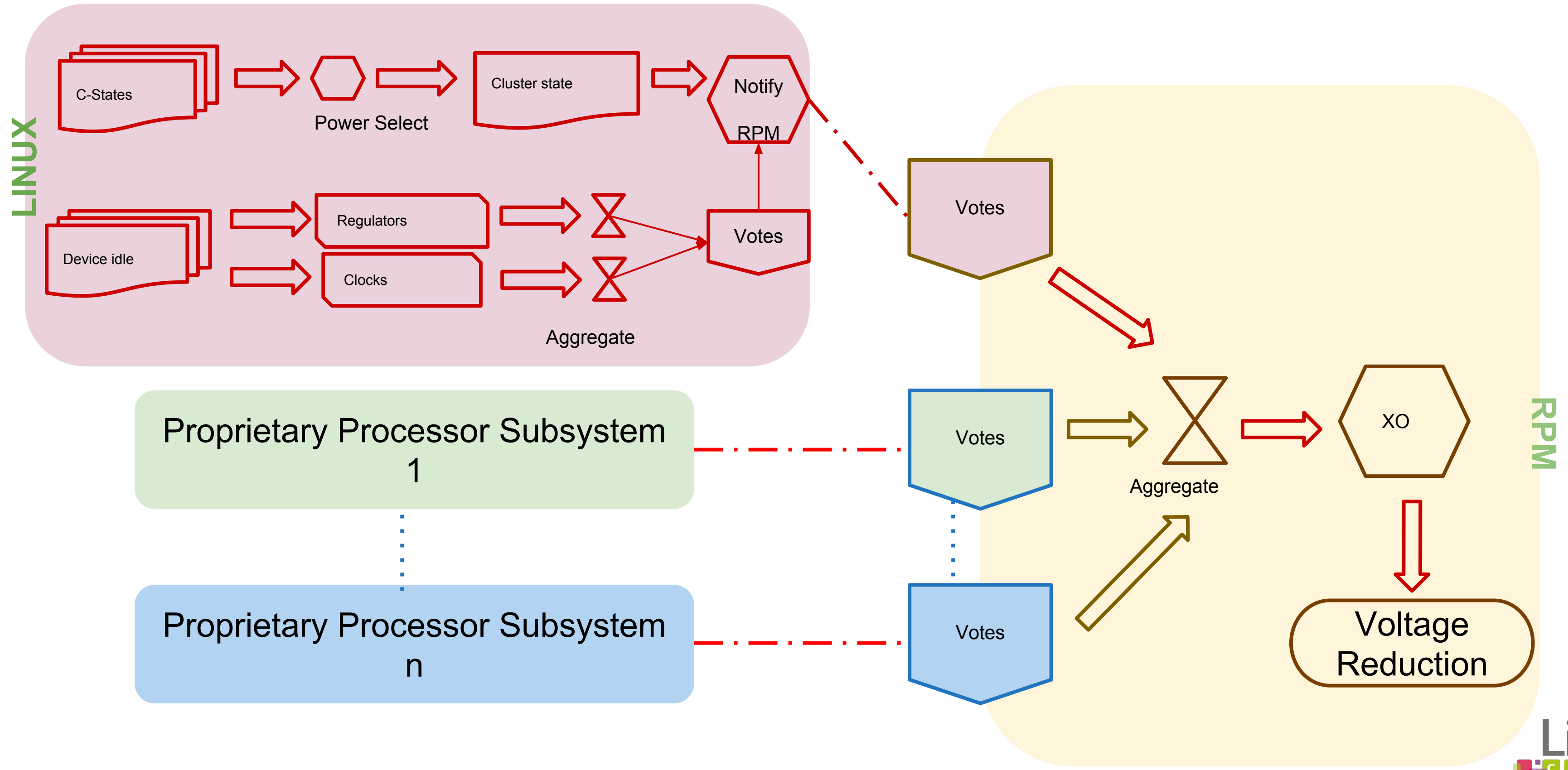
CPU & Cluster Idle State

- CPUs generally support 4 idle state (not all SoCs)
  - C1: Clock gating
  - C2: Retention - Clock gating at lower voltage
  - C3: Standalone Power Collapse - CPU power down
  - C4: Power Collapse - Allowing for deeper SoC idle or power rail off
- L2 States (cluster states):
  - Clock gating
  - Retention
  - GDHS (Globally Distributed Head Switch off - Memory retained, Logic off)
  - Power Collapse - Memory flushed and powered off, Logic off
- When all CPUs enter low power mode, L2 enters low power mode
  - L2 needs to be ready before the CPU can resume execution
- L2 enters an idle state matching the shallowest state entered by a CPU
  - Even if one CPU enters clock gating, L2 can enter an idle state

Idle States And Power States



SoC Idle.. How The Dominoes Fall?



Resource Power Manager (RPM)

- Low power core to manage system shared resources
- RPM ‘shared resources’ are finite and fixed
  - Pre-defined resources, shared between OSes like Linux, Windows, etc and other non-application processors and peripherals in the SoC
- Subsystems communicate with RPM through proprietary protocols
  - Registers writes as in 8064
  - Ring buffer based IPC as in 8074
- Controls the idle entry and exit of the SoC
  - Each subsystem votes for the idle state values
  - RPM aggregates and configures the best possible SoC state
- Subsystems indicate their low power entry/exit through interrupts
  - Enter: SPM sends an interrupt to RPM indicate core power down
  - Exit: SPM sends an interrupt to RPM, to ensure dependencies are ready, before the core starts executing

Subsystem Power Manager (SPM)

- SPM is a hardware Finite State Machine
- SPM aides idle entry and exit for cores - CPU or L2
  - In most SoCs, CPU and L2 both have SPM to enter low power states
- Multiple idle states can be configured in an SPM
  - An idle-state is represented as a set of predefined bytes: SPM sequence
  - SPM sequences are unique for an SoC, based on the hardware configuration
- Core - SPM handshake starts with the core executing WFI
  - The Core clock-gates and then triggers the SPM state machine
  - SPM waits for interrupt from GIC, to finish executing the state machine
  - Upon completion of the state machine, the SPM returns control to the core
- SPMs are in an always-on power domain
  - They remain powered on, even when they power off their core
- SPM can communicate with the PMIC to regulate voltage
- CPUs warm boot into secure mode if they were powered down

Challenges

- Represent hierarchical idle states
- Idle states can be enabled/disabled at runtime
- Direct PM QoS (Quality of Service) requirements only to certain CPUs
- Interrupt sensitivity to exit latencies

Hierarchical Idle States

- Why?
    - Snapdragon SoCs differ from one another in a very small, but important ways
    - Multiple possibilities of CPU, Cluster and SoC idle states based on hardware support
    - Allows the same driver to support a multitude of SoCs

- How its done today
    - Set up each idle state (power & latency numbers) in the DT (boards file in older SoC)
    - Similar to ARM common idle-state bindings
        - But have Overhead time, Overhead energy and steady state power in DT
    - Hierarchy represented as DT nodes and subnodes
        - Cluster idle state define the minimum required C-State
    - Allows for statistics to be provided for each level for each CPU or cache
    - Allows idle-states to be enabled/disabled from sysfs

https://www.codeaurora.org/cgit/quic/la/kernel/msm-3.10/tree/arch/arm/boot/dts/qcom/msm8974-v2-pm.dtsi?h=msm-3.10

https://www.codeaurora.org/cgit/quic/la/kernel/msm-3.10/tree/arch/arm/boot/dts/qcom/msm8994-pm.dtsi?h=msm-3.10

Idle States Enable/Disabled At Runtime

- Why?
  - Runtime conditions may disallow some states
    - Eg., Processor running at low voltage cannot allow retention modes
    - Use case based power manager

- How its done today
  - Each state has a sysfs node to allow/disallow the low power mode
  - A core already in the low power state would be brought out to active and the mode disabled
  - Idle states in DT, define the latency and power needed to enter/exit the low power mode and the power drained while in the low power state
    - Total energy in state = Overhead energy + Energy in Steady State
  - Power calculations instead of residency helps choose the best state
    - Does not utilize the menu governor heuristics

https://www.codeaurora.org/cgit/quic/la/kernel/msm-3.10/tree/drivers/cpuidle/lpm-levels.c?h=msm-3.10

Direct PM QoS Only To Certain CPUs

- Why?
  - PM QoS use of CPU_DMA_LATENCY in idle governor applies to all CPUs
    - Need to guarantee performance required by the drivers
  - But performance may only be needed only for some CPUs
    - CPUs determined by organization. Eg, performance cpus only.
    - CPUs that would handle an IRQ and need performance for the device IRQ

- How its done today
  - Introduce per-CPU PM QoS feature to QoS framework
  - Idle power-select calculations use QoS required for the CPUs in the cluster
    - PM QoS based on CPUmask used to determine cluster's QoS requirement
  - Only affected CPUs are notified of PM QoS state changes
  - Patches in review in the mailing lists

Interrupt Sensitivity To Exit Latencies

- Why?
  - Some device IRQ are sensitive to exit latencies
  - Performance drops, when CPU has high latency to wake up and handle the IRQ
  - But we have sufficient time between interrupts to enter deeper sleep states
  - Drivers know when the interrupt would fire in the future (predictable time)
  - PM QoS if used, prevents deeper sleep states completely
  - Need to balance power-performance equation for the use case
- How its done today
  - Drivers notify Event Timer framework when they know they are expecting an interrupt
  - Idle driver calculates the first wake up from the ET framework and the timer wheel
  - Sets up a timer to wake the CPU up before the ‘Event’
  - Best possible low power mode chosen for the new sleep time
  - Timer wakes up the CPU and the CPU is ready to handle the interrupt (Event)
  - Directed only to the CPU that needs to handle the interrupt

https://www.codeaurora.org/cgit/quic/la/kernel/msm-3.10/tree/drivers/soc/qcom/event_timer.c?h=msm-3.10

More about Linaro Connect: connect.linaro.org
Linaro members: www.linaro.org/members
More about Linaro: www.linaro.org/about/