Linaro Connect
MADRID 2024 | MAY 14-17 2024

Lulling your Dragon to sleep

Low Power Modes on linux-arm-msm

$(whoami)

- Konrad Dybcio <konrad.dybcio@linaro.org> | <konradybcio@kernel.org>
- Over 1.5y @ Linaro
- 21 years old, from 🇵🇱 Poland
- linux-arm-msm co-maintainer
  - Taking care of incoming patches & actively poking at stuff
  - Long-time reviewer, contributor, watcher
  - Some userspace tools
- Putting Linux on phones (even iPhones 👀) since 2017

Intro

- Modern SoCs feature tens of meaningful cores (CPU and otherwise)
- The CPU subsystem (e.g. 12x Oryon + caches) is NOT the only place where heavy processing happens
  - CPUs often end up just relaying data between other IPs
- Every “number cruncher”[1] onboard needs electricity (often at the same time)

[1]DSPs, GPUs, CPUs, ASICs, MCUs, ISPs...

The Problem

- Most computers run on electricity
- Computing consumes power
- Joule-Lenz law: “*Consuming power generates heat*”
  - $Q = RI^2t$
  - Important observation: $I \searrow \Rightarrow Q \searrow\searrow$
  - Perhaps an even more important observation: $t \searrow \Rightarrow Q \searrow$
- Not computing.. **apparently also consumes power**!?

  (sidenote: it took humanity quite some time to realize that this is suboptimal)

The idea of not computing

- Old computers consumed roughly the same amount of energy, no matter the workload
  - Doesn't cut it anymore with 21st century 500W+ TDP packages
  - Old CPUs were just.. CPUs, no fancy DSPs etc.

- Nowadays things like cpuidle exist (helpful but still need more)
  - Turn off the CPU if it's not doing anything
  - Do it really often (as soon as nothing is scheduled)
  - Do it really fast (entry/exit idle in *nanoseconds*)
  - Wake up instantly when signaled (via WAKE IRQs)
  - **Historically, it's been really hard...**

Solution?

Add more cores, obviously!

Snapdragon - CPU subsystem

- A standard Arm recipe with a San Diego spice mix
  - Standard WFI/WFE + PSCI cpuidle
  - **CPU power sequencing happens on a MCU**

- The gory details are opaque to the OS
  - The OS requests something akin to ACPI PSTATEs
  - Voltage is silently set in lockstep with frequency
  - All CPU supplies are abstracted away

- All this is *relatively* new
  - 10y ago, all of this was handled through **10k+ LoC** in Linux
    - Almost every new chip generation required **10k more**
  - Nowadays things just work™

Snapdragon - off-CPU IP

- Non-CPU power governance happens through the **RPMh** core
  - It's a black box
  - Aggregated power requirements are sent there
  - RPMh lets us enter a **system-wide** sleep state ("**power collapse**")

- The actual mV/mA values are abstracted away for portability
  - e.g. assume max GPU frequency requires a "TURBO" state
  - "TURBO" may mean 1.2V on one SoC, 0.9V on another

- The code behind this **must be** robust, fail-safe and SoC-specific
  - Sounds intrinsically boring and difficult to both maintain and test
  - We developers hate this kind of responsibility
  - Perfect candidate to **NOT** be part of Linux

Snapdragon - RPMh

- Each **master** can place a **request** which will keep a **resource** online
  - “resource” is a vague term to represent some hardware (e.g. a clock)

- What’s a **master**? (generally speaking, may differ SoC-to-SoC)
  - CPUSS (where Linux/Windows/QNX runs)
  - All the DSPs: Audio, Compute, Modem, Sensor..
  - Always-On SubSystem (AOSS) - the island where RPMh lives
  - Camera island
  - PCIe island
  - The list grows every now and then

Snapdragon - RPMh (cont.)

- Only keep resources online when absolutely necessary and do so at the lowest possible power state
  - Minimizing I and t in the Joule-Lenz formula
- All such requirements must be described somewhere (DT/ACPI)
- After aggregating the data, the OS sends out requests
- **What happens if they're wrong / insufficient?**


guess I'll die

“But why?”

- Unclocked accesses are trapped and result in fatal violations
  - Hypervisorish to English: if you try accessing hardware that’s powered off, the platform is going to hard crash and reboot
- Undervoltage
  - You can run the GPU at max frequency and the lowest voltage available, just not for very long.. A few ns at best
- Missing links
  - Some resources physically connect parts of the SoC together - you may not be able to reach half of the peripherals hardware if one of them is down (see the unclocked access story again)

The most common pitfalls (in DTS & co.)

Missing:

- interconnect links
- clock references
- regulator consumers
- power domain references and state settings

All of these resources are SHUT DOWN by Linux if deemed unnecessary - often solely due to improper DT/ACPI description.

(see struct device_driver.sync_state and _ignore_unused cmdline params)

So, how to sleep properly?

- Clean up any lingering power hogs
  - Bootloaders are known for leaving some things in a messy state
- Shut down resources that you don’t need at runtime
  - Don’t go overboard and kill the DRAM interface though..
- Scale down the frequencies as you go
  - Don’t forget to crank down the power lines too
- The OS can only voice suggestions, RPMh is the ultimate decision maker

Dragons and cats are the same really...

Simply, don't bother the thing and it's gonna fall asleep in no time!

RPMh automatically disables resources when there are no "on" requests.

In power collapse, we can reach milliwatt-level power draw, as opposed to multiple watts at runtime.

Wonder what's the deep sleep power usage of this cat though.

Things to improve (Linux)

- Shave some yaks: go over all the platforms (and devices) and look which pitfalls we're still falling into
- Improve common code, ensure shutting down of unused resources of every type
- Don't exclusively chase closing the feature gap, mind the power gap too - it's going to majorly improve thermals and usability


LinaroConnect
MADRID 2024 | MAY 12-17 2024
Thank you
Patches welcome!
linux-arm-msm@vger.kernel.org

Bonus: A proper system sleep-wake cycle

1. The OS booted successfully and unused resources leftover from prior boot stages were cleaned up
2. <the user uses the computer for a bit>
3. The user clicks the “sleep” button
4. The display, sound, storage, etc. hardware is powered off
5. The OS propagates a magic SMC call to the TrustZone which powers off the entirety of CPUSS
6. <no other RPMh masters requested for any resources to stay up>
7. All collapsible power rails are shut down
8. The system clock sources (down to the crystal oscillator) are cut
9. The AOSS enters a retention state
10. <the computer happily sleeps, consuming mere milliwatts>
11. AOSS receives a wake-up interrupt
12. Steps 9-4 are undone, execution continues