SFO15-502: Generic cpuidle driver

Daniel Lezcano
Generic ARM cpuidle driver

This presentation will describe the how to write a new cpuidle driver to fit into the ARM generic cpuidle framework
Introduction: Legacy drivers

- Duplicated code
- Per platform directory
  - Hard to track and maintain
- Low level PM code inside the driver itself
  - Dependencies with the `<mach/*>` and `<asm/*>`
- Static idle states table
Changes with the cpuidle drivers

- Huge changes around the drivers
- Encapsulation of the PM low level code
  - PM callbacks through platform_data driver
- Moved the drivers to drivers/cpuidle
  - Linux-3.0 => 0 drivers
  - Linux-4.0 => 14 drivers
- Factored out common code inside the generic cpuidle framework
Switch to another driver

- platform_data callbacks no longer accepted
  - Introduction of the ARM64 generic cpuidle driver thanks to PSCI
Generic ARM cpuidle driver

Device tree idle state definitions → Generic ARM cpuidle driver → Low level PM ops

Cpuidle framework
Device tree idle state definitions

- The idle state are defined in the device tree

```c
idle-states {
    CPU_SPC: spc {
        compatible = "qcom.idle-state-spc",
        "arm.idle-state";
        entry-lateness-us = <150>;
        exit-lateness-us = <200>;
        min-residency-us = <2000>;
    };
};
```
Device tree idle state definitions

- The cpu definition refers to the idle state

```c
cpu@3 {
  compatible = "qcom,krait";
  enable-method = "qcom,kpss-acc-v2";
  device_type = "cpu";
  reg = <3>;
  next-level-cache = <&L2>;
  qcom,acc = <&acc3>;
  qcom,saw = <&saw3>;
  cpu-idle-states = <&CPU_SPC>;
};
```
Low level PM ops

- On ARM: struct cpuidle_ops
  - Init and suspend

- On ARM64: struct cpu_operations
  - Contains cpu initialization, cpu hotplug and cpu idle
  - Not so much to say, everything is PSCI
Low level PM ops

- The initialization is handled by the PM low level code via a macro ...

```c
static struct cpuidle_ops qcom_cpuidle_ops __initdata = {
    .suspend = qcom_idle_enter,
    .init = qcom_cpuidle_init,
};

CPUIDLE_METHOD_OF_DECLARE(qcom_idle_v1,
    "qcom,kpss-acc-v1", &qcom_cpuidle_ops);
```
Low level PM ops

- … and with a matching table.
- The init function parses the DT and initializes its internals with the PM callback

```c
static const struct of_device_id qcom_idle_state_match[] __initconst = {
    { .compatible = "qcom,idle-state-spc", .data = qcom_cpu_spn },
};
```

idle state name  PM callback
Generic ARM cpuidle driver

- Has a default WFI idle state
- Parses the DT and fill the driver with the different idle states
- Initialize the ops (init, suspend)
Generic ARM cpuidle driver : init

- Lookup for the ‘enable-method’

```c
cpu@0 {
    compatible = "qcom,krait";
    enable-method = "qcom,kpss-acc-v2";
    device_type = "cpu";
    reg = <0>;
    next-level-cache = <&L2>;
    qcom.acc = <&acc0>;
    qcom.saw = <&saw0>;
    cpu-idle-states = <&CPU_SPC>;
};
```

```c
CPUIDLE_METHOD_OF_DECLARE(qcom_idle_v1,
    "qcom,kpss-acc-v2", &qcom_cpuidle_ops);
```
Generic ARM cpuidle driver : init

- Assigns cpuidle_ops with the platform ops
- Calls cpuidle_ops.init()
Generic ARM cpuidle driver : suspend

- One function: arm_enter_idle_state
- If index is 0 then WFI
- Otherwise calls arm_cpuidle_suspend which in turns calls cpuidle_ops.suspend(index)
- It is up to the low level PM code to switch to the right sleep state based on the index passed as parameter
Conclusion

- **Self-contained**: backend driver is in the low level PM code
- **Flexible**: Idle states are described in the device tree
- **Simple**: You just have to care about writing the PM callbacks
Next steps

- Multiple drivers support via DT definition

- Idle state timer shutdown via DT with the power domains (WIP)
  - Currently a simple property ‘local-timer-stop’
Credits

- Lorenzo Pieralisi for the ARM64 generic cpuidle driver
- Lina Iyer for writing the first cpuidle driver based on the new ARM generic cpuidle driver (QCOM)