Introduction to Kernel Power Management

Kevin Hilman, Linaro
khilman@kernel.org

ELC 2015, San Jose CA
Complexity is growing...
- more CPUs
- more integrated devices
- more power domains
- micro controllers
- firmware, etc.

Kernel is evolving....
Introduction to Kernel Power Management

Kevin Hilman, Linaro

khilman@kernel.org

ELC 2015, San Jose CA

When wakeup occurs (e.g. in ISR):

Subsystem / Driver control:

```
struct dev_pm_ops {
    Exists in struct device_driver,
    device_init_wakeup(dev, bool)
    ...
    int (*resume_early)(struct device *, bool);
    int (*suspend_late)(struct device *, bool);
    int (*resume)(struct device *);
    void (*complete)(struct device *);
    int (*prepare)(struct device *);
    ...
};
```

Wakeup from Suspend

-> finish()

-> enter()

```
struct bus_type, ...
struct dev_pm_ops
```

Per-device:

```
-> resume()
-> resume_noirq()
-> suspend_late()
-> suspend()
```

Clocks

Voltage scaling:

- regulator framework

Frequency scaling:

- clock framework

```
regulator_get_voltage()
clk_set_rate()
clk_get_rate()
```

CPUfreq

```
drivers/cpufreq/cpufreq-dt.c
```

- what about device DVFS?

- pluggable governors for selecting "best" OPP

```
...
```

```
1008000 1375000
800000 600000 300000
```

Operating-points = <

- described in DT
- tuple of frequency, minimum voltage

OPPs

Operating Performance Points

```
CPU
L2$
```

Complexity is growing...

- more CPU's
- more integrated devices
- more power domains
- more controllers
- firmware, etc.

Kernel is evolving...

```
```

PM domains

Dynamic PM

- save power when doing "something"

- save power when doing "nothing"

Save power when doing "nothing"

Save power when doing "something"

```
```

```
Idle
CPU idle
```

```
CPU idle:
```

```
Performance Impact:
predictable events (e.g. timers)
```

```
Don't wake up...
only to press snooze and go back to sleep
```

```
Idle CPUs
Idle devices
```

```
Active
Static
Dynamic
Work in progress
```

```
SusPEND
Resume
Wakeups
```

```
Clocks
Regulators
CPUfreq
```

```
Opps
```

```
NOHZ_IDLE
Runtime PM
```

```
PM domains
PM QoS
```

```
Idle CPUs
Idle devices
```

```
Static PM, System PM
```

```
Dynamic PM
```

```
Quality:
```

```
CPU
L2$
```

```
DSP
```

```
GPU
```

```
GIC,
PMUs,
VFP,
CoreSight,
etc.
```

```
```

```
```

```
```

```
PM QOS
```

```
Wakeup when doing "nothing"

Wakeup when doing "something"
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```
Static PM, System PM
- traditional suspend/resume
  CONFIG_PM_SLEEP=y
- system wide, all devices
- initiated by userspace
- any device can prevent suspend

- userspace is "frozen"
  (c.f. Documentation/freezing-of-tasks.txt)

MUST Read: Documentation/power/devices.txt
Driver model: key concept

\textbf{struct dev_pm_ops}

Exists in struct device_driver, struct bus_type, ...

\begin{verbatim}
struct dev_pm_ops {
    int (*prepare)(struct device *dev);
    void (*complete)(struct device *dev);
    int (*suspend)(struct device *dev);
    int (*resume)(struct device *dev);
    ...
    int (*suspend_late)(struct device *dev);
    int (*resume_early)(struct device *dev);
    ...
};
\end{verbatim}
echo mem > /sys/power/state

Platform specific:
struct platform_suspend_ops
 ->begin()
 ->prepare()
 ->enter()
 ->wake()
 ->finish()
 ->end()

Per-device:
struct dev_pm_ops
 ->prepare()
 ->suspend()
 ->suspend_late()
 ->suspend_noirq()
 ->resume_noirq()
 ->resume_early()
 ->resume()
 ->complete()
Wakeup from Suspend

Subsystem / Driver control:
- `device_init_wakeup(dev, bool)`
- `enable_irq_wake()`
- `disable_irq_wake()`

When wakeup occurs (e.g. in ISR):
- `pm_wakeup_event()`

Enable / disable from user space:
- `/sys/devices/.../power/wakeup`
Introduction to Kernel Power Management

Kevin Hilman, Linaro
khilman@kernel.org
ELC 2015, San Jose CA

Enable/disable from user space:
- When wakeup occurs (e.g. in ISR):
  - Subsystem/Driver control:
    - pm_wakeup_event()
    - disable_irq_wake()
    - enable_irq_wake()
    - device_init_wakeup(dev, bool)
    - resume_early(dev);
    - resume(dev);
    - suspend(dev);

Platform specific:
- end()
- finish()
- wake()
- complete()
- resume_noirq()
- suspend()

Clocks
Resume
-
- Example...
  Documentation/power/regulator/consumer.txt
  clk_set_rate()
  clk_get_rate()

CPUfreq
...
what about device DVFS?
...
c.f. Documentation/power/opp.txt
800000
600000
300000
/* kHz - tuple of frequency, minimum voltage
OPPs
devfreq}
1200000
1025000
uV*/

- micro controllers
- more power domains
- more CPUs
- userspace is "frozen"
- initiated by userspace
- system wide, all devices
- traditional suspend/resume
  (c.f. Documentation/power/devices.txt)

Complexity is growing...
- more CPUs
- more integrated devices
- more power domains
- more controllers
- firmware, etc.

Kernel is evolving...
Dynamic PM

- based on "activity"
- Active PM
  Save power when doing "something"

- Idle PM
  Save power when doing "nothing"
Active PM: Underlying Frameworks

Frequency scaling: clock framework
- clk_get_rate()
- clk_set_rate()

Voltage scaling: regulator framework
- regulator_get_voltage()
- regulator_set_voltage()
- Documentation/power/regulator/consumer.txt

Example... drivers/cpufreq/cpufreq-dt.c

CPU DVFS using CPUFreq
- Select "best" OPP based on requirements
- pluggable governors for selecting "best" OPP
- performance, powersave, ...
- ondemand, onlinemode based on load, tunable
- interactive: ondemand++
- Documentation/cpu-freq/core.txt
... what about device DVFS? ... devfreq

Operating Performance Points OPPs
- tuple of frequency, minimum voltage
- Described in DT
  cpufreq: cpu0 {
    operating-points = <
      /* kHz mV */
      300000 1025000
      600000 1200000
      800000 1313000
      1008000 1375000
    >}
  }
c.f. Documentation/power/opp.txt

Clocks

Regulators
Operating Performance Points

- tuple of frequency, minimum voltage
- Described in DT

```c
cpu0: cpu@0 {
  operating-points = <
  /* kHz uV */
  300000 1025000
  600000 1200000
  800000 1313000
  1008000 1375000
}>;
}
```

c.f. Documentation/power/opp.txt

OPPs
CPU DVFS using CPUFreq

- Select "best" OPP based on requirements
- pluggable governors for selecting "best" OPP
- performance, powersave, ...
- ondemand: heursitics based on load, tunable
- interactive: ondemand++

Documentation/cpu-freq/core.txt

... what about device DVFS? .... devfreq
Active PM: Underlying Frameworks

Frequency scaling: clock framework
- `clk_get_rate()`
- `clk_set_rate()`

Voltage scaling: regulator framework
- `regulator_get_voltage()`
- `regulator_set_voltage()`
- `Documentation/power/regulator/consumer.txt`

Example... `drivers/cpufreq/cpufreq-dt.c`
Introduction to Kernel Power Management

Kevin Hilman, Linaro
khilman@kernel.org

ELC 2015, San Jose CA

CP...m...s

1. Structure

- Suspend
- Resume

2. Wakeups

- NOHZ_IDLE

3. PM domains

- CPUfreq

4. OPPs

- CPUidle

5. Runtime PM

- Active

- Idle

- Static

- Dynamic

- Work in progress

- Next steps

- Clocks

- Regulators

- CPUfreq

- OPPs

- Idle CPUs

- Idle devices

- Energy-aware scheduling: EAS

- Next Steps

- System-wide:
  - for use by genpd governors

- Devices are often grouped into domains

- Tiny implementation of PM domains

- Domain callbacks instead of type/class/bus

- Energy-saving clock management

- Frequency scaling

- Power gating

- Power management framework

- Platform-specific hooks

- State entry

- Legacy:
  - platform-specific driver

- State definitions in DT

- CPU idle states have "depth"

- Longer wakeup latency

- Limitations:
  - Compares against min residency
  - Looks at

- 1) Break even point (based on enter/exit times)

- Save power when doing "something"

- Complexity is growing:
  - More CPUs
  - More integrated devices
  - More power domains
  - More controllers
  - Firmware, etc.

- Kernel is evolving...
Idle PM: tickless idle

```
CONFIG_NOHZ_IDLE=y
```

- stop periodic tick when idle
- only makes sense for next "visor" or incomplete

NOHZ_IDLE

CPU Idle: How deep is sleep?

- arm.h idle()
- basement

CPU idle states have "sleepy" energy state:
- platform specific drive
- scheduler tracked
- State Definitions in DT
- legacy platform specific drive
- State entry:
- platform setup hooks
- based on sleep in sleeping

Idle for CPUs

- CPU Idle states have "sleepy" energy state:
- legacy platform specific
- scheduler tracked
- State Definitions in DT
- legacy platform specific drive
- State entry:
- platform setup hooks
- based on sleep in sleeping

Idle CPUs

Idle for devices: Runtime PM

- per device idle
- state tracked by driver, based on sleeping
- devices are deferred
- one driver, common resources
- idle manager config
- class NOHZ all instance space

Idle devices

Runtime PM callbacks

- Use count: 1
- prepare for next "event"
- sleep period
- idle state

Runtime PM

- Use count: 0
- save context
- ensure wakeups enabled
- prepare for low-power state

Use count: 1

- pm_runtime_resume()

- pm_runtime_idle()

Generic PM Domains (gmpd)

- In DT:
- pm.ko
- pm_powercontroller

Example power

```
power <device> in idle state
```

CPU freq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power

CPUfreq governors

- CPUfreq:
- scheduled tasks
- consumes context:
- consumes power

CPU idle governors

- CPU idle:
- scheduled tasks
- consumes context:
- consumes power
Idle for CPUs

CPU idle states have "depth"
- more power savings
- longer wakeup latency

State Definitions in DT
- legacy: platform-specific driver

State entry
- platform-specific hooks
- based on compatible string

idle-states {
  CPU_STBY: standby {
    compatible = "qcom,idle-state-stby",
    "arm,idle-state";
    entry-latency-us = <1>;
    exit-latency-us = <1>;
    min-residency-us = <2>;
  };
}

CPU_SPC: spc {
  compatible = "qcom,idle-state-spc",
  "arm,idle-state";
  entry-latency-us = <150>;
  exit-latency-us = <200>;
  min-residency-us = <2000>;
};

cpu@0 {
  [...]
  cpu-idle-states = &CPU_STBY &CPU_SPC;
};

Documentation/devicetree/bindings/arm/idle-states.txt
CPUidle: How deep to sleep?

drivers/cpuidle/governors/menu.c

1) Break even point (based on enter/exit times)
- looks at predictable events (e.g. timers)
- compares against min residency

2) Latency tolerance
- checks QoS (PM_QOS_CPU_DMA_LATENCY)
- compares against min residency

3) Performance Impact:
- black magic "multiplier" based on load
- favor shallower states under heavy load

Limitations:
- not very SMP or multi-cluster aware
Idle PM: tickless idle

CONFIG_NOHZ_IDLE=y

- stop periodic tick when idle
- only wakes for next "event" or interrupt

Don't wake up...
only to press snooze and go back to sleep
Idle for devices: Runtime PM

- per-device idle
- single device at a time
- idleness controlled by driver, based on activity

- devices are independent
- one device cannot prevent others from runtime suspending

- does NOT affect user space

```c
struct dev_pm_ops {
    ...
    int (*runtime_suspend)(struct device *dev);
    int (*runtime_resume)(struct device *dev);
    int (*runtime_idle)(struct device *dev);
};
```
Tell PM core whether device is in use
"I'm about to use it"
- `pm_runtime_get()`, `__sync()`
- Increment use count, `pm_runtime_resume()`

"I'm done... for now"
- `pm_runtime_put()`, `__sync()`
- Decrement use count, `pm_runtime_idle()`

Similar to legacy clock framework usage for clock gating
- `clk_enable()`, `clk_disable()`

Excellent: [Documentation/power/pm_runtime.txt](#)
Runtime PM: callbacks

Use count: 1 --> 0
- ->runtime_suspend()
- prepare for low-power state
- ensure wakeups enabled
- save context

Use count: 0 --> 1
- ->runtime_resume()
- restore context
- etc.

Autosuspend
- pm_runtime_set_autosuspend_delay()
- pm_runtime_mark_last_busy()
- pm_runtime_put_autosuspend()
Idle CPUs

Idle devices

Runtime PM

PM QoS

Generic PM Domains (genpd)

Energy-aware scheduling: EAS

Idle for CPUs: Runtime PM

- pm idle states have 'lights' to
  express power states
- longer run-up latency

State Definitions in DT
- legacy platform-specific state

State entry
- platform-specific looks
  - based on transition

Idle for devices: Runtime PM

- per device idle
- sleep times are
  - platform-specific
  - tweaking needed

-dominance to
- class NOHZ idle space

Runtime PM callbacks

Use count 1 --> 0
- entries when suspended
- exits when resumed

Use count 0 --> 1
- resuming
- no conflict

pm_runtime_get()
pm_runtime_set_autosuspend_delay()
pm_runtime_resume()

I'm about to use it

pm_runtime_put()

I'm done...

pm_runtime_mark_last_busy()

Incuse use count, pm_runtime_resume()

pm_runtime_get(), _sync()

Tell PM core whether device is in use

Runtime PM API

TeAPMcore whether device is in use

- "I'm about to use it" (device resumed)
- "I'm done...

Increment use count, pm_runtime_resume()

Use count: 0 --> 1
- save context
- ensure wakeups enabled
- prepare for low-power state

- pm_runtime_put_autosuspend()
- pm_runtime_mark_last_busy()
- pm_runtime_set_autosuspend_delay()

PM QoS

- pm_qos_cpu_dma_latency
- sched_policy change dpm idle state

Per device
- per device governors with specific device
- device governors
- genpd:
  - prevent PM domain power off
  - use runtime PM for CPU-connected
    "extras"
  - for use by genpd "governors"
  - for genetic algo governors

Quality: PM QoS

Generic implementation of PM domains

-Based on number PM
- When all devices do PM domain are runtime suspended...
- apm --power_pause (CPU idle)
- When first device is domain is resumed...
- apm --power_resume (CPU idle)

genpd governors
- allow a domain transition before power gating e.g., per-device Governor callback

Example genpd:

```c
reg = <0x12340000 0x1000>
compatible = "foo,power-controller",

reg = <0x12350000 0x1000>
compatible = "foo,i-leak-current",
```

Example use by device:

```c
genpd->power_on()
genpd->power_off()
reg = <0x12350000 0x1000>
compatible = "foo,i-leak-current",
```

Sample use in DT:

```c
e.g. genpd:

reg = <0x12340000 0x1000>
compatible = "foo,power-controller",

reg = <0x12350000 0x1000>
compatible = "foo,i-leak-current",
```

CPU genpd can hit low-power state (off)

L2$ plus shared L2$

These are to be replaced by scheduler-driven data

PM domains

Grouping

PM QoS

Idle for devices: Grouping

- group of devices can be a group
- can be nested
- power gating has lockdown implications
- external registry ramp up, etc.

Idle for devices:

- devices diplayed into domains
- power gated as a group
- can be nested
- power gating has lockdown implications
- external registry ramp up, etc.

Documentation/device-tree/bindings/arm/idle-states.txt

```c
cpu@0 {
  idle-states {
    cpu-idle-states = <&CPU_STBY &CPU_SPC>;
    ...
  }
}
```

```c
compatible = "qcom,idle-state-spc",
```

```c
CPU_SPC:
  standby {
    ... };
```

```c
min-residency-us = <2>;
```

```c
"arm,idle-state";
```

```c
compatible = "qcom,idle-state-stby",
```

```c
CPU_STBY:
  standby {
    ... }
```

```c
exit-latency-us = <200>;
```

```c
entry-latency-us = <150>;
```

```c
"arm,idle-state";
```
Idle for devices: Grouping

Devices are often grouped into domains
- power gated as a group
- can be nested
- power gating has latency implications
- external regulator ramp up, etc.

Linux: PM domains
- override ops for a group of devices
- if PM domain present, PM core uses domain callbacks instead of type/class/bus

```c
struct dev_pm_domain {
    struct dev_pm_ops ops;
    ...
};
```
Generic PM Domains (genpd)

Generic implementation of PM domains
- Based on runtime PM
- When all devices in domain are runtime suspended...
  \[ \text{genpd} \rightarrow \text{power\_off}() \]
- When first device in domain is runtime resumed...
  \[ \text{genpd} \rightarrow \text{power\_on}() \]

genpd governors
- allow custom decision making before power gating
- e.g. per-device QoS constraints
genpd in DT
Example genpd:

```plaintext
power: power-controller@12340000 {
  compatible = "foo,power-controller";
  reg = <0x12340000 0x1000>;
  #power-domain-cells = <1>;
};
```

Example use by device:

```plaintext
leaky-device@12350000 {
  compatible = "foo,i-leak-current";
  reg = <0x12350000 0x1000>;
  power-domains = <&power 0>;
};
```

From: Documentation/devicetree/bindings/power/power-domain.txt
Quality: PM QoS

System-wide: e.g. `PM_QOS_CPU_DMA_LATENCY`
- Used by CPUidle to determine depth of idle state

Per-device
- attach QoS constraints with specific devices
- genpd: prevent PM domain power off
  - `PM_QOS_FLAG_NO_POWER_OFF`
- e.g. genpd: per-device wakeup latency
  - `DEV_PM_QOS_RESUME_LATENCY`
  - for use by genpd "governors"

Documentation/power/pm_qos_interface.txt
Introduction to Kernel Power Management

Kevin Hilman, Linaro
khilman@kernel.org
ELC 2015, San Jose CA

NOTE: For the full experience, use the original SVG at:
http://people.linaro.org/~kevin.hilman/conf/elc2015/Intro_Kernel_PM.svg

When wakeup occurs (e.g. in ISR):

```c
struct dev_pm_ops {
    Exists in struct device_driver,
    enable_irq_wake()
    device_init_wakeup(dev, bool)
    int (*suspend_late)(struct device* dev);
    int (*suspend)(struct device* dev);
    int (*prepare)(struct device* dev);
    Platform specific:
    -Wakeup from Suspend
    struct bus_type,
    ...
    struct dev_pm_ops
    ->resume_early()
    ->suspend_noirq()
    ->suspend_late()
    ->suspend()
    ->prepare()
}
```

Clocks
Regulators
CPUfreq
OPPs

- Interactive:
  - ondemand++
  - ondemand: heuristics based on load, tunable

CPU idle states have "depth"

- deeper states
- longer wakeup latency

CPU idle:

- stop periodic tick when idle

CPU idle states have "depth"

- longer wakeup latency

CPU idle:

- stop periodic tick when idle

Idle devices

- per-device idle
- single device at a time

Idle for devices:

- Grouping

PM domains

- use runtime PM for CPU-connected "extras"

- Integrate CPUidle and CPUfreq with the scheduler
- take advantage of energy-saving hardware
- Teach the scheduler new heuristics for task placement to

Energy-aware scheduling:

- EAS

CPU idle

Example...

Documentation/power/regulator/consumer.txt

regulator_setVoltage()
clk_set_rate()

- Interactive:
  - ondemand++
  - ondemand: heuristics based on load, tunable

CPU DVFS using CPUFreq

c.f. Documentation/power/opp.txt

1008000 1375000
800000
600000
operating-points = <
1200000
uV /*

Clocked clocks

PM domains

- model clusters as genpd made up of "CPU genpd"s
- combine into a "CPU genpd"

Scheduler-driven data

- leaky-device@12350000

Example use by device:

Example genpd:

genpd in DT

#power-domain-cells = <1>
reg = <0x12340000 0x1000>
compatible = "foo,power-controller"

Generic PM Domains (genpd)

- use runtime PM for CPU-connected "extras"

- for use by genpd "governors"

PM QoS

- use runtime PM for CPU-connected "extras"

- for use by genpd "governors"

PM_QOS_FLAG_NO_POWER_OFF

Quality:

- PM QoS
  - not very SMP or multi-cluster aware

Complexity is growing...
- more CPUs
- more integrated devices
- more power domains
- more controllers
- firmware, etc.

Work in progress

Next steps

Questions?

Any
Runtime PM, genpd ideas

Unify idle for CPUs and devices
- use runtime PM for CPUs
- use runtime PM for CPU-connected "extras"
  (e.g. GIC, PMUs, VFP, CoreSight, etc.)
- combine into a "CPU genpd"

Extend to CPU Clusters
- model clusters as genpd made up of "CPU genpd"s
  plus shared L2$ 
- when CPUs in cluster are idle (runtime suspended) 
  cluster genpd can hit low-power state (off)

Energy-aware scheduling: EAS

An on-going effort to improve energy efficiency of the scheduler.

- Teach the scheduler new heuristics for task placement to
  take advantage of energy-saving hardware

- Integrate CPUidle and CPUfreq with the scheduler
  - scheduler tracks load statistics for its own decision making
    (e.g. task placement, load balancing, etc.)

- CPUidle / CPUfreq governor decisions are based on their own load-based calculations, heuristics (and some black magic).

These are to be replaced by scheduler-driven data
Runtime PM, genpd ideas

Unify idle for CPUs and devices
- use runtime PM for CPUs
- use runtime PM for CPU-connected "extras"
  (e.g. GIC, PMUs, VFP, CoreSight, etc.)
- combine into a "CPU genpd"

Extend to CPU Clusters
- model clusters as genpd made up of "CPU genpd"s
  plus shared L2$
- when CPUs in cluster are idle (runtime suspended)
  cluster genpd can hit low-power state (off)

Next Steps
Energy-aware scheduling: EAS
- Teach the scheduler new heuristics for task placement to take advantage of energy-saving hardware
- Integrate CPUidle and CPUfreq with the scheduler
  the scheduler tracks load statistics for its own decision making (e.g. task placement, load balancing, etc.)
- CPUidle / CPUfreq governor decisions are based on their own load-based calculations, the heuristics (and some black magic).
  These are to be replaced by scheduler-driven data
Energy-aware scheduling: EAS

An on-going effort to improve energy efficiency of the scheduler.

- Teach the scheduler new heuristics for task placement to take advantage of energy-saving hardware

- Integrate CPUIdle and CPUfreq with the scheduler
  - scheduler tracks load statistics for its own decision making (e.g. task placement, load balancing, etc.)

- CPUIdle / CPUfreq governor decisions are based on their own load-based calculations, heuristics (and some black magic).

These are to be replaced by scheduler-driven data
Introduction to Kernel Power Management

Kevin Hilman, Linaro
khilman@kernel.org

ELC 2015, San Jose CA

When wakeup occurs (e.g. in ISR):
- disable_irq_wake()
- enable_irq_wake()
- device_init_wakeup(dev, bool)

Resume:
- (*resume_early)(struct device* dev);
- (*resume)(struct device* dev);
- (*suspend)(struct device* dev);
- (*complete)(struct device* dev);

Suspend:
- (*prepare)(struct device* dev);
- (*suspend_late)(struct device* dev);
- (*suspend_noirq)(struct device* dev);
- (*resume_noirq)(struct device* dev);
- (*resume_early)(struct device* dev);

Driver model:
- key concept

NOTE:
- For the full experience, use the original SVG at:
  http://people.linaro.org/~kevin.hilman/conf/elc2015/Intro_Kernel_PM.svg

PM Domains
- CPUFreq
- CPUIdle
- Idle CPUs
- Idle devices
- OPPs
- Clocks
- Regulators
- Active
- Static
- Dynamic
- Work in progress
- Next steps
- Suspending
- Resume
- Wakeups

Suspend:
- pm_runtime_mark_last_busy()
- pm_runtime_set_autosuspend_delay()
-> runtime_suspend()

Resume:
- pm_runtime_resume()
- pm_runtime_get()
- pm_runtime_idle()
- pm_runtime_put()
- pm_runtime_set_autosuspend_delay()
- pm_runtime_mark_last_busy()

CPUfreq:
- CPUfreq:
  - Occupied
  - Available
  - Use
  - What about device DVFS?

PM QoS
- Limitations:
  - Break even point (based on enter/exit times)
- Userspace is "frozen"
- Any device can prevent suspend initiated by userspace system wide, all devices

PM Domains
- CPUfreq:
  - Frequency scaling:
    - Clock framework
  - CPUfreq:
    - Performance, powersave, ...

CPUidle:
- CPUidle:
  - Only wakes for next "event"
or interrupt

Quality:
- PM QoS

Power gating has latency implications
- Power gated as a group
- Devices are often grouped into domains

Idle for devices:
- Grouping
- Platformspecific hooks
- State entry
- Idle for CPUs
- Unify idle for CPUs and devices

BM domains
- Auto-suspend
- Ensure wakeups enabled
- Prepare for low-power state

Use count:
- Increment use count, pm_runtime_resume()
- Decrement use count, pm_runtime_idle()

Auto-suspend
- pm_runtime_get(), _sync()
- pm_runtime_put(), _sync()

Power:
- power: power-controller@12340000
- reg = <0x12340000 0x1000>
- genpd->power_on()

Regulator:
- PM domains
- PM QoS
- CPUs
- OPPs
- MUXes
- Drivers
Phew... that was all just a bad dream

Any Questions?

Energy-aware scheduling: EAS
- An ongoing effort to improve energy efficiency of the scheduler.
  - Teach the scheduler new heuristics for task placement to take advantage of energy-saving hardware.
  - Integrate CPU idle and CPU freq with the scheduler.
  - The scheduler tracks load statistics for its own decision making (e.g., task placement, load balancing, etc.).
  - CPU idle/CPU freq governor decisions are based on their own load-based calculations, heuristics (and some black magic).
  - These are to be replaced by scheduler-driven data.