The OMAP PM interface

This document describes the temporary OMAP PM interface. Driver authors use these functions to communicate minimum latency or throughput constraints to the kernel power management code. Over time, the intention is to merge features from the OMAP PM interface into the Linux PM QoS code.

Drivers need to express PM parameters which:

- support the range of power management parameters present in the TI SRF;
- separate the drivers from the underlying PM parameter implementation, whether it is the TI SRF or Linux PM QoS or Linux latency framework or something else;
- specify PM parameters in terms of fundamental units, such as latency and throughput, rather than units which are specific to OMAP or to particular OMAP variants;
- allow drivers which are shared with other architectures (e.g., DaVinci) to add these constraints in a way which won't affect non-OMAP systems,
- can be implemented immediately with minimal disruption of other architectures.

This document proposes the OMAP PM interface, including the following five power management functions for driver code:

1. Set the maximum MPU wakeup latency:

   ```
   (*pdata->set_max_mpu_wakeup_lat)(struct device *dev, unsigned long t)
   ```

2. Set the maximum device wakeup latency:

   ```
   (*pdata->set_max_dev_wakeup_lat)(struct device *dev, unsigned long t)
   ```

3. Set the maximum system DMA transfer start latency (CORE pwrdm):

   ```
   (*pdata->set_max_sdma_lat)(struct device *dev, long t)
   ```

4. Set the minimum bus throughput needed by a device:

   ```
   (*pdata->set_min_bus_tput)(struct device *dev, u8 agent_id, unsigned long r)
   ```

5. Return the number of times the device has lost context:

   ```
   (*pdata->get_dev_context_loss_count)(struct device *dev)
   ```

Further documentation for all OMAP PM interface functions can be found in arch/arm/plat-omap/include/mach/omap-pm.h.

The OMAP PM layer is intended to be temporary

The intention is that eventually the Linux PM QoS layer should support the range of power management features present in OMAP3. As this happens, existing drivers using the OMAP PM interface can be modified to use the Linux PM QoS code; and the OMAP PM interface can disappear.

Driver usage of the OMAP PM functions

As the 'pdata' in the above examples indicates, these functions are exposed to drivers through function pointers in driver .platform_data structures. The function pointers are initialized by the *board-*.c* files to point to the corresponding OMAP PM functions:

- set_max_dev_wakeup_lat will point to omap_pm_set_max_dev_wakeup_lat(), etc. Other architectures which do not support these functions should leave these function pointers set to NULL. Drivers should use the following idiom:

  ```
  if (pdata->set_max_dev_wakeup_lat)
      (*pdata->set_max_dev_wakeup_lat)(dev, t);
  ```

The most common usage of these functions will probably be to specify the maximum time from when an interrupt occurs, to when the device becomes accessible. To accomplish this, driver writers should use the set_max_mpu_wakeup_lat() function to constrain the MPU wakeup latency, and the set_max_dev_wakeup_lat() function to constrain the device wakeup latency (from clk_enable() to accessibility). For example:

```
/* Limit MPU wakeup latency */
if (pdata->set_max_mpu_wakeup_lat)
    (*pdata->set_max_mpu_wakeup_lat)(dev, tc);

/* Limit device powerdomain wakeup latency */
if (pdata->set_max_dev_wakeup_lat)
    (*pdata->set_max_dev_wakeup_lat)(dev, td);

/* total wakeup latency in this example: (tc + td) */
```

The PM parameters can be overwritten by calling the function again with the new value. The settings can be removed by calling the function with a t argument of -1 (except in the case of set_max_bus_tput(), which should be called with an r argument of 0).

The fifth function above, omap_pm_get_dev_context_loss_count(), is intended as an optimization to allow drivers to determine whether the device has lost its internal context. If context has been lost, the driver must restore its internal context before proceeding.

Other specialized interface functions

The five functions listed above are intended to be usable by any device driver. DSPBridge and CPUFreq have a few special requirements. DSPBridge expresses target DSP performance levels in terms of OPP IDs. CPUFreq expresses target MPU performance levels in terms of MPU frequency. The OMAP PM interface contains functions for these specialized cases to convert that input information (OPPs/MPU frequency) into the form that the underlying power management implementation needs:

6. *(*pdata->dsp_get_opp_table)(void)*
7. *(*pdata->dsp_set_min_opp)(u8 opp_id)*
8. *(*pdata->dsp_get_opp)(void)*
9. *(*pdata->cpu_get_freq_table)(void)*
10. *(*pdata->cpu_set_freq)(unsigned long f)*
11. *(*pdata->cpu_get_freq)(void)*

Customizing OPP for platform

Defining CONFIG_PM should enable OPP layer for the silicon and the registration of OPP table should take place automatically. However, in special cases, the default OPP table may need to be tweaked, for e.g.:

- enable default OPPs which are disabled by default, but which could be enabled on a platform
- Disable an unsupported OPP on the platform
- Define and add a custom opp table entry in these cases, the board file needs to do additional steps as follows:

arch/arm/mach-omapx/board-xyz.c:

```
#include "pm.h"
....
static void __init omap_xyz_init_irq(void)
{
        ....
        /* Initialize the default table */
        omapx_opp_init();
        /* Do customization to the defaults */
        ....
}
```

NOTE:

omapx_opp_init will be omap3_opp_init or as required based on the omap family.