Operating Performance Points (OPP) Library

C. 2009-2010 Nishanth Menon <nm@ti.com>, Texas Instruments Incorporated

1. Introduction

1.1 What is an Operating Performance Point (OPP)?

Complex SoCs of today consists of a multiple sub-modules working in conjunction. In an operational system executing varied use cases, not all modules in the SoC need to function at their highest performing frequency all the time. To facilitate this, sub-modules in a SoC are grouped into domains, allowing some domains to run at lower voltage and frequency while other domains run at voltage/frequency pairs that are higher.

The set of discrete tuples consisting of frequency and voltage pairs that the device will support per domain are called Operating Performance Points or OPPs.

As an example:

Let us consider an MPU device which supports the following: {300MHz at minimum voltage of 1V}, {800MHz at minimum voltage of 1.2V}, {1GHz at minimum voltage of 1.3V}

We can represent these as three OPPs as the following {Hz, uV} tuples:

- {300000000, 1000000}
- {800000000, 1200000}
- {1000000000, 1300000}

1.2 Operating Performance Points Library

OPP library provides a set of helper functions to organize and query the OPP information. The library is located in drivers/opp/ directory and the header is located in include/linux/pm_opp.h. OPP library can be enabled by enabling CONFIG_PM_OPP from power management menuconfig menu. Certain SoCs such as Texas Instrument's OMAP framework allows to optionally boot at a certain OPP without needing cpufreq.

Typical usage of the OPP library is as follows:

```
(users)        -> registers a set of default OPPs             -> (library)
SoC framework  -> modifies on required cases certain OPPs     -> OPP layer
               -> queries to search/retrieve information      ->
```

OPP layer expects each domain to be represented by a unique device pointer. SoC framework registers a set of initial OPPs per device with the OPP layer. This list is expected to be an optimally small number typically around 5 per device. This initial list contains a set of OPPs that the framework expects to be safely enabled by default in the system.

Note on OPP Availability

As the system proceeds to operate, SoC framework may choose to make certain OPPs available or not available on each device based on various external factors. Example usage: Thermal management or other exceptional situations where SoC framework might choose to disable a higher frequency OPP to safely continue operations until that OPP could be re-enabled if possible.

OPP library facilitates this concept in its implementation. The following operational functions operate only on available opps: dev_pm_opp_find_freq_{ceil, floor}, dev_pm_opp_get_voltage, dev_pm_opp_get_freq, dev_pm_opp_get_opp_count.

dev_pm_opp_find_freq_exact is meant to be used to find the opp pointer which can then be used for dev_pm_opp_enable/disable functions to make an opp available as required.

WARNING: Users of OPP library should refresh their availability count using get_opp_count if dev_pm_opp_enable/disable functions are invoked for a device, the exact mechanism to trigger these or the notification mechanism to other dependent subsystems such as cpufreq are left to the discretion of the SoC specific framework which uses the OPP library. Similar care needs to be taken care to refresh the cpufreq table in cases of these operations.

2. Initial OPP List Registration

The SoC implementation calls dev_pm_opp_add function iteratively to add OPPs per device. It is expected that the SoC framework will register the OPP entries optimally- typical numbers range to be less than 5. The list generated by registering the OPPs is maintained by OPP library throughout the device operation. The SoC framework can subsequently control the availability of the OPPs dynamically using the dev_pm_opp_enable / disable functions.

dev_pm_opp_add

> Add a new OPP for a specific domain represented by the device pointer. The OPP is defined using the frequency and voltage. Once added, the OPP is assumed to be available and control of its availability can be done with the

dev_pm_opp_enable/disable functions. OPP library internally stores and manages this information in the dev_pm_opp struct. This function may be used by SoC framework to define a optimal list as per the demands of SoC usage environment.

WARNING:

Do not use this function in interrupt context.

Example:

```
soc_pm_init()
{
        /* Do things */
        r = dev_pm_opp_add(mpu_dev, 1000000, 900000);
        if (!r) {
                pr_err("%s: unable to register mpu opp(%d)\n", r);
                goto no_cpufreq;
        }
        /* Do cpufreq things */
no_cpufreq:
        /* Do remaining things */
}
```

3. OPP Search Functions

High level framework such as cpufreq operates on frequencies. To map the frequency back to the corresponding OPP, OPP library provides handy functions to search the OPP list that OPP library internally manages. These search functions return the matching pointer representing the opp if a match is found, else returns error. These errors are expected to be handled by standard error checks such as IS_ERR() and appropriate actions taken by the caller.

Callers of these functions shall call dev_pm_opp_put() after they have used the OPP. Otherwise the memory for the OPP will never get freed and result in memleak.

dev_pm_opp_find_freq_exact

Search for an OPP based on an *exact* frequency and availability. This function is especially useful to enable an OPP which is not available by default. Example: In a case when SoC framework detects a situation where a higher frequency could be made available, it can use this function to find the OPP prior to call the dev_pm_opp_enable to actually make it available:

```
opp = dev_pm_opp_find_freq_exact(dev, 1000000000, false);
dev_pm_opp_put(opp);
/* dont operate on the pointer.. just do a sanity check.. */
if (IS_ERR(opp)) {
        pr_err("frequency not disabled!\n");
        /* trigger appropriate actions.. */
} else {
        dev_pm_opp_enable(dev,1000000000);
}
```

NOTE:

This is the only search function that operates on OPPs which are not available.

dev_pm_opp_find_freq_floor

Search for an available OPP which is *at most* the provided frequency. This function is useful while searching for a lesser match OR operating on OPP information in the order of decreasing frequency. Example: To find the highest opp for a device:

```
freq = ULONG_MAX;
opp = dev_pm_opp_find_freq_floor(dev, &freq);
dev_pm_opp_put(opp);
```

dev_pm_opp_find_freq_ceil

Search for an available OPP which is *at least* the provided frequency. This function is useful while searching for a higher match OR operating on OPP information in the order of increasing frequency. Example 1: To find the lowest opp for a device:

```
freq = 0;
opp = dev_pm_opp_find_freq_ceil(dev, &freq);
dev_pm_opp_put(opp);
```

Example 2: A simplified implementation of a SoC cpufreq_driver->target:

```
soc_cpufreq_target(..)
{
        /* Do stuff like policy checks etc. */
        /* Find the best frequency match for the req */
        opp = dev_pm_opp_find_freq_ceil(dev, &freq);
        dev_pm_opp_put(opp);
        if (!IS_ERR(opp))
```

```
                soc_switch_to_freq_voltage(freq);
        else
                /* do something when we can't satisfy the req */
        /* do other stuff */
}
```

4. OPP Availability Control Functions

A default OPP list registered with the OPP library may not cater to all possible situation. The OPP library provides a set of functions to modify the availability of a OPP within the OPP list. This allows SoC frameworks to have fine grained dynamic control of which sets of OPPs are operationally available. These functions are intended to *temporarily* remove an OPP in conditions such as thermal considerations (e.g. don't use OPPx until the temperature drops).

WARNING:

Do not use these functions in interrupt context.

dev_pm_opp_enable

Make a OPP available for operation. Example: Lets say that 1GHz OPP is to be made available only if the SoC temperature is lower than a certain threshold. The SoC framework implementation might choose to do something as follows:

```
if (cur_temp < temp_low_thresh) {
        /* Enable 1GHz if it was disabled */
        opp = dev_pm_opp_find_freq_exact(dev, 1000000000, false);
        dev_pm_opp_put(opp);
        /* just error check */
        if (!IS_ERR(opp))
                ret = dev_pm_opp_enable(dev, 1000000000);
        else
                goto try_something_else;
}
```

dev_pm_opp_disable

Make an OPP to be not available for operation Example: Lets say that 1GHz OPP is to be disabled if the temperature exceeds a threshold value. The SoC framework implementation might choose to do something as follows:

```
if (cur_temp > temp_high_thresh) {
        /* Disable 1GHz if it was enabled */
        opp = dev_pm_opp_find_freq_exact(dev, 1000000000, true);
        dev_pm_opp_put(opp);
        /* just error check */
        if (!IS_ERR(opp))
                ret = dev_pm_opp_disable(dev, 1000000000);
        else
                goto try_something_else;
}
```

5. OPP Data Retrieval Functions

Since OPP library abstracts away the OPP information, a set of functions to pull information from the dev_pm_opp structure is necessary. Once an OPP pointer is retrieved using the search functions, the following functions can be used by SoC framework to retrieve the information represented inside the OPP layer.

dev_pm_opp_get_voltage

Retrieve the voltage represented by the opp pointer. Example: At a cpufreq transition to a different frequency, SoC framework requires to set the voltage represented by the OPP using the regulator framework to the Power Management chip providing the voltage:

```
soc_switch_to_freq_voltage(freq)
{
        /* do things */
        opp = dev_pm_opp_find_freq_ceil(dev, &freq);
        v = dev_pm_opp_get_voltage(opp);
        dev_pm_opp_put(opp);
        if (v)
                regulator_set_voltage(.., v);
        /* do other things */
}
```

dev_pm_opp_get_freq

Retrieve the freq represented by the opp pointer. Example: Lets say the SoC framework uses a couple of helper functions we could pass opp pointers instead of doing additional parameters to handle quiet a bit of data parameters:

```
soc_cpufreq_target(..)
{
```

```
                /* do things.. */
                 max_freq = ULONG_MAX;
                 max_opp = dev_pm_opp_find_freq_floor(dev,&max_freq);
                 requested_opp = dev_pm_opp_find_freq_ceil(dev,&freq);
                 if (!IS_ERR(max_opp) && !IS_ERR(requested_opp))
                        r = soc_test_validity(max_opp, requested_opp);
                 dev_pm_opp_put(max_opp);
                 dev_pm_opp_put(requested_opp);
                /* do other things */
        }
        soc_test_validity(..)
        {
                 if(dev_pm_opp_get_voltage(max_opp) < dev_pm_opp_get_voltage(requested_opp))
                          return -EINVAL;
                 if(dev_pm_opp_get_freq(max_opp) < dev_pm_opp_get_freq(requested_opp))
                          return -EINVAL;
                /* do things.. */
        }
```

dev_pm_opp_get_opp_count

Retrieve the number of available opps for a device Example: Lets say a co-processor in the SoC needs to know the available frequencies in a table, the main processor can notify as following:

```
soc_notify_coproc_available_frequencies()
{
       /* Do things */
       num_available = dev_pm_opp_get_opp_count(dev);
       speeds = kzalloc(sizeof(u32) * num_available, GFP_KERNEL);
       /* populate the table in increasing order */
       freq = 0;
       while (!IS_ERR(opp = dev_pm_opp_find_freq_ceil(dev, &freq))) {
               speeds[i] = freq;
               freq++;
               i++;
               dev_pm_opp_put(opp);
       }

       soc_notify_coproc(AVAILABLE_FREQs, speeds, num_available);
       /* Do other things */
}
```

6. Data Structures

Typically an SoC contains multiple voltage domains which are variable. Each domain is represented by a device pointer. The relationship to OPP can be represented as follows:

```
SoC
 |- device 1
 |    |- opp 1 (availability, freq, voltage)
 |    |- opp 2 ..
 ...  ...
 |    `- opp n ..
 |- device 2
 ...
 `- device m
```

OPP library maintains a internal list that the SoC framework populates and accessed by various functions as described above. However, the structures representing the actual OPPs and domains are internal to the OPP library itself to allow for suitable abstraction reusable across systems.

struct dev_pm_opp

The internal data structure of OPP library which is used to represent an OPP. In addition to the freq, voltage, availability information, it also contains internal book keeping information required for the OPP library to operate on. Pointer to this structure is provided back to the users such as SoC framework to be used as a identifier for OPP in the interactions with OPP layer.

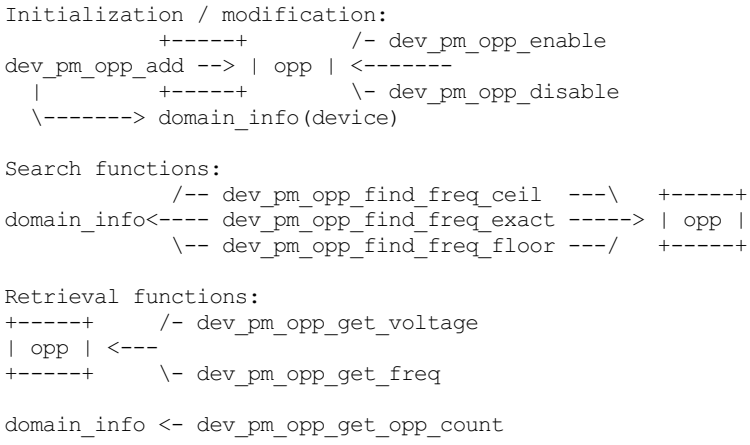
WARNING:

The struct dev_pm_opp pointer should not be parsed or modified by the users. The defaults of for an instance is populated by dev_pm_opp_add, but the availability of the OPP can be modified by dev_pm_opp_enable/disable functions.

struct device

This is used to identify a domain to the OPP layer. The nature of the device and its implementation is left to the user of OPP library such as the SoC framework.

Overall, in a simplistic view, the data structure operations is represented as following:


Initialization / modification:
+-----+ /- dev_pm_opp_enable
dev_pm_opp_add --> | opp | <-------
| +-----+ \- dev_pm_opp_disable
\-------> domain_info(device)
Search functions:
/-- dev_pm_opp_find_freq_ceil ---\ +-----+
domain_info<---- dev_pm_opp_find_freq_exact -----> | opp |
\-- dev_pm_opp_find_freq_floor ---/ +-----+
Retrieval functions:
+-----+ /- dev_pm_opp_get_voltage
| opp | <---
+-----+ \- dev_pm_opp_get_freq
domain_info <- dev_pm_opp_get_opp_count