Texas Instruments
OMAP
System Engineering
Linux Development Center

LPC 2012

PM constraints μConf:

Discussion on the OMAP PM QoS constraints use cases and requirements

Jean Pihet <j-pihet@ti.com>

Introduction

PM QoS status

The current PM QoS currently has the following classes:

PM_QOS_CPU_DMA_LATENCY (type MIN),
PM_QOS_NETWORK_LATENCY (type MIN),
PM_QOS_NETWORK_THROUGHPUT (type MAX).

But ... PM QoS is not the right place for all kinds of constraints

PM QoS uses plist, which only retrieves the MIN or MAX from the constraints list,
Combination of constraints is not supported: ex. Multiple network cards in a firewall setup.

Specific frameworks shall implement the logic in /drivers.
Ex: per-device PM QoS constraints in drivers/base/power/qos.c
thermal in his own location
...

On-going discussions

New model for device and system latency
User space interface
Bus throughput constraints

On-going discussions 1

New model for devices and system latency [1]

The new model provides allows a flexible and dynamic way to calculate the expected system latency from the various contributors.

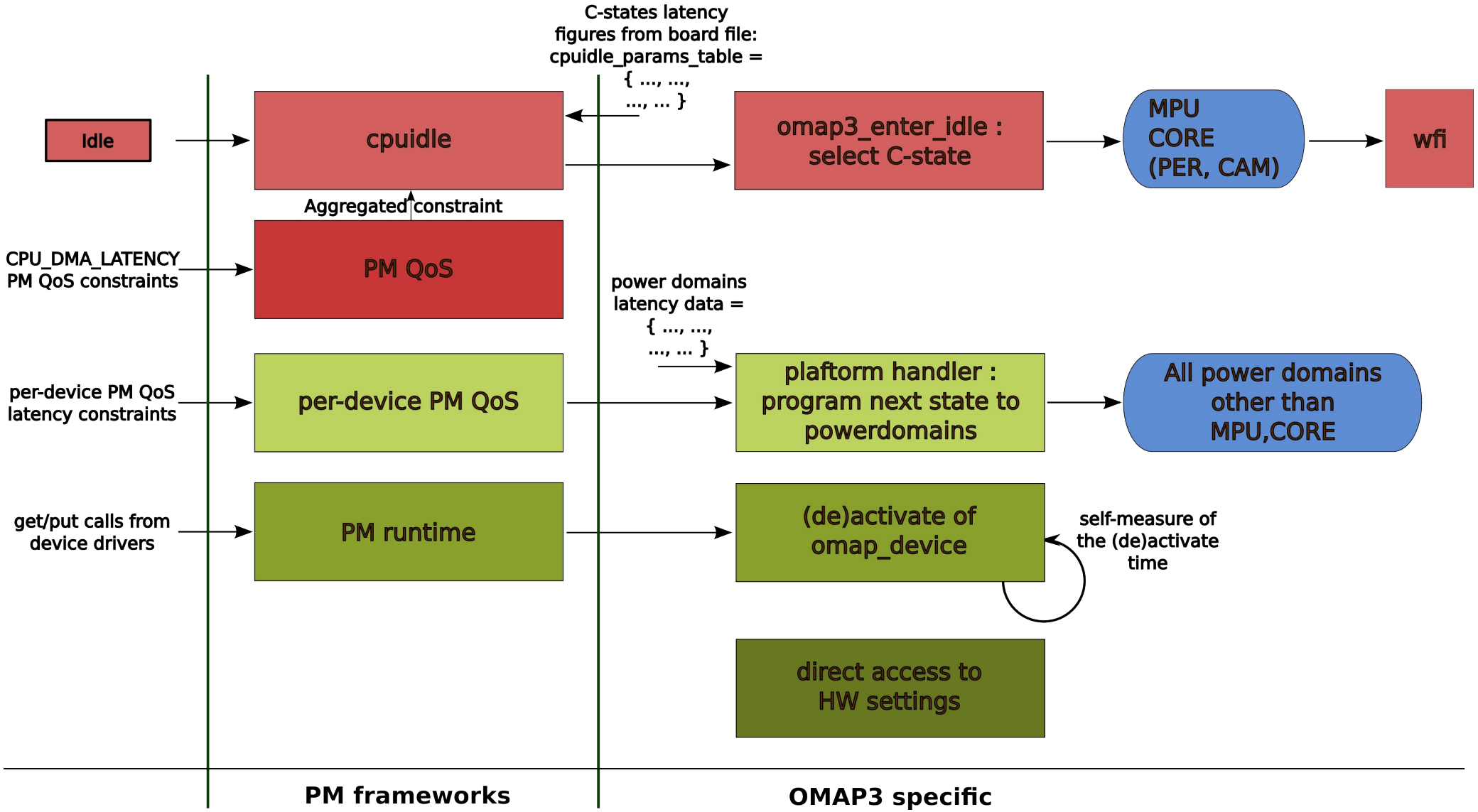
Implementing the new model for latency imposes some changes in the PM runtime framework for devices:

- Automatic measurement of device (de)activate latency,

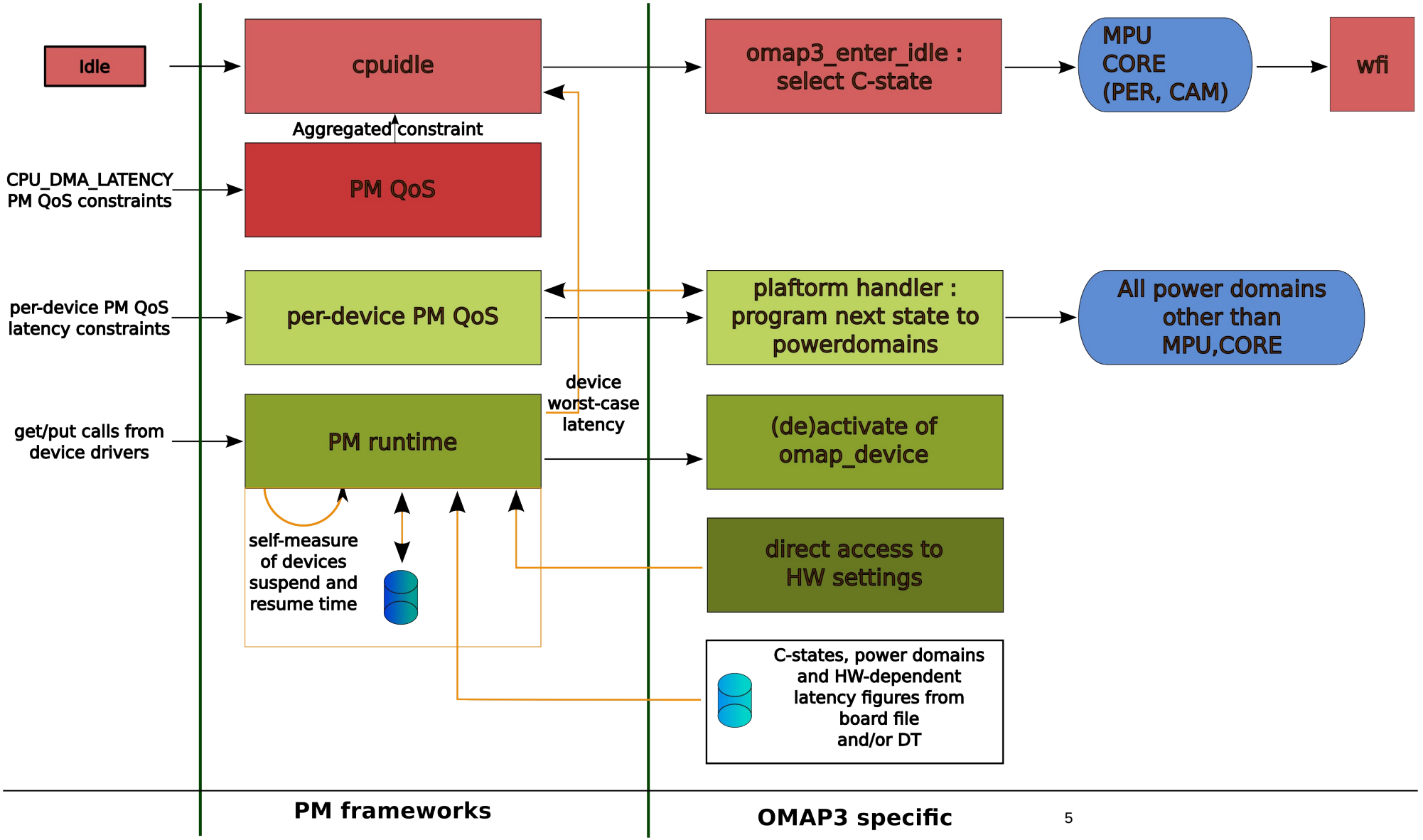
- Registration of latency contributors + re-calculation of overall latency upon (de)activation,

- Per-device constraints code should use the generic power domains code
(! OMAP has its own implementation of power domains code).
(! The generic power domain code has no provision for multiple power states, which OMAP is using (ON, INACTIVE, CSWR, OSWR, OFF), which prevents OMAP code for using it (for now)).

Current model for devices latency



New model for devices latency



On-going discussions 2

Bus throughput constraints

The throughput constraints shall be expressed in terms of a required throughput between the source ('from') and the target ('to') devices

Memory and other bus throughput: drivers shall translate the throughput requirements into frequencies requests,

Constraints for minimum and maximum bus throughput/frequency,
Ex. Some of the thermal management constraints can be translated in max throughput constraints

More flexible way of aggregating constraints: addition, max, bidirectional...

Questions:
How to combine multiple data flows: add, max... ?
Case of bidirectional bus?
User space?

On-going discussions 3

User space interface

User space is problematic:

- What to export to user space?

- User space should not override the in-kernel constraints. If in use an user space constraint is just one of the constraints in the list.
- Are multiple accesses needed?

- Debug entries are helpful for debugging

Proposals 1

New generic model for devices latency

The following needs to be implemented in the generic framework:

- Automatic measurement of device (de)activate latency,

- Registration of latency contributors + re-calculation of overall latency upon (de)activation,

- Augment the generic power domains code
  The generic power domain code has no provision for multiple power states, which OMAP is using (ON, INACTIVE, CSWR, OSWR, OFF), which prevents OMAP code for using it (for now)).

Proposals 2

Bus throughput

Devices oriented:

```
int request/remove_tput_constraint(device *from, device *to, int tput_kBs,
                                   int constraint_type);
```

constraint type gives information about the type of constraint (ex. min/max) and the way to aggregate them together (add, max etc).

Upon constraint update:
- the framework computes the aggregated value, then
- calls the low level driver code to apply the constraint,
- the driver code converts the throughput requirement into afrequency change,
- the driver calls the common clock framework to scale the frequency (and other HW dependant settings if needed -voltage etc.-)

Proposals 3

User space

- Export some of the constraints to user space (which ones?). If exported the user space is an additional constraints in the list (i.e. does not override the exisiting constraints).If multiple users are needed a user space daemon is the solution, possibly using generic lib code.

- Provide debug entries for the constraints lists,

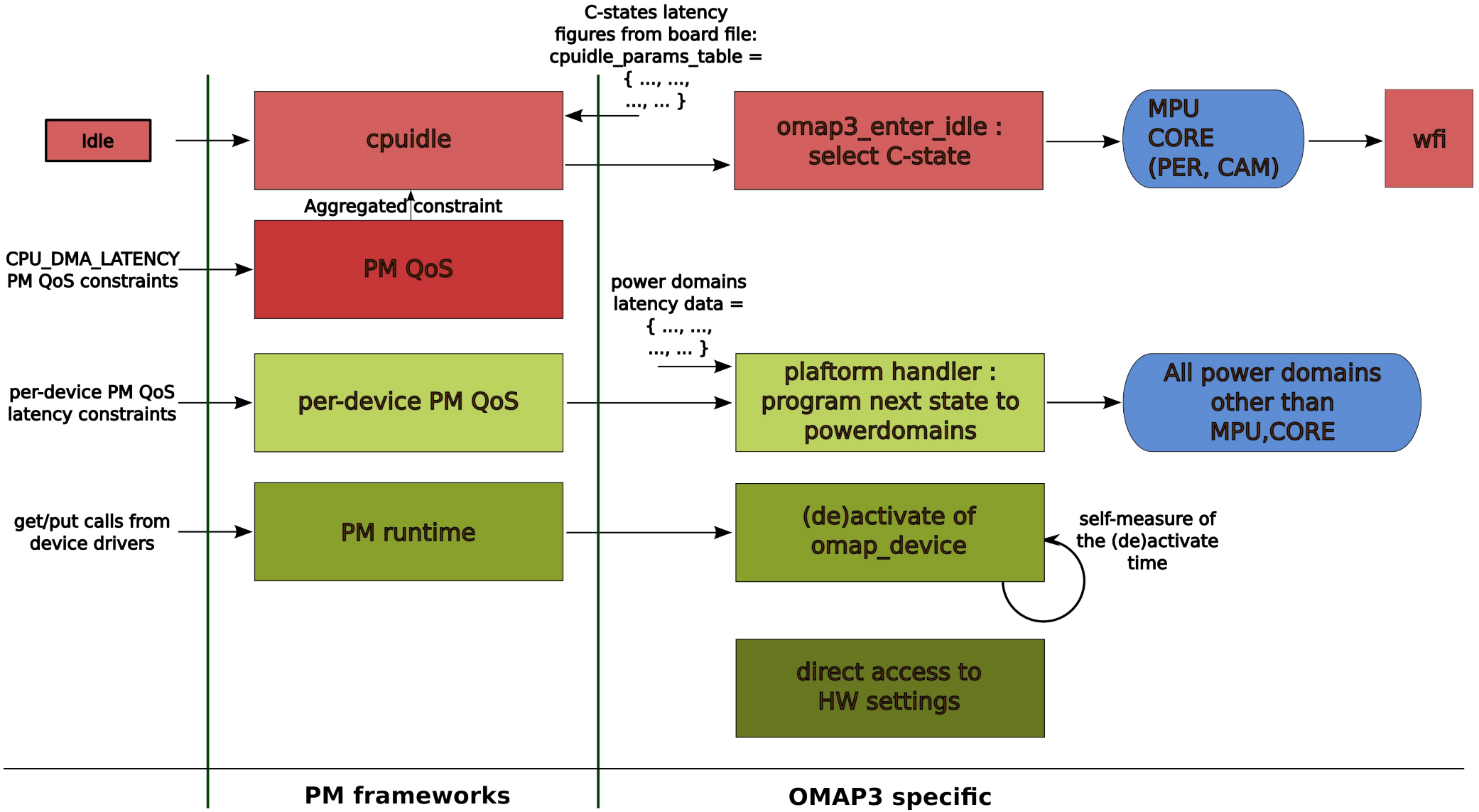
- Provide debug entries for the decisions taken by the framework.

Links

[1] A new model for the system and devices latency
http://www.omappedia.org/wiki/File:ELC-2012-jpihet-DeviceLatencyModel.pdf

Backup slides:
New model for devices and system latency

Current model



Problems

There is no concept of 'overall latency'.

No interdependency between PM frameworks

Ex. on OMAP3 : cpuidle manages only a subset of the power domains (MPU, CORE).
Ex. on OMAP3 per-device PM QoS manages the other power domains.
No relation between the frameworks, each framework has its own latency numbers.

Some system settings are not included in the model

Mainly because of the (lack of) SW support at the time of the measurement session.
Ex. On OMAP3 : voltage scaling in low power modes, sys_clkreq, sys_offmode and the interaction with the PowerIC.

Dynamic nature of the system settings

The measured numbers are for a fixed setup, with predefined system settings.
The measured numbers are constant.

Problems (more of them!)

Self-measuremente of OMAP devices (de)activate : Great idea, but ...

- The code is not generic enough, only the omap_device code has the feature implemented.
- The self-measurement results are not used at all (excepted to issue a 'New worst case (de)activate latency' debug message).

Measuring the various latencies is difficult

- The measurement procedure needs to be re-run for every different HW (or possibly SW) setup.
- Measuring the latency of all power domains is difficult : take measurements, derive energy graphs, calculate intersections, adapt to missing key parameters etc.

Solution proposal

Overall latency calculation

We need a model which breaks down the overall latency into the latencies from every contributor :

$$latency = latency_{SW} + latency_{HW}$$
$$latency = latency_{SW} + latency_{SoC} + latency_{External\ HW}$$

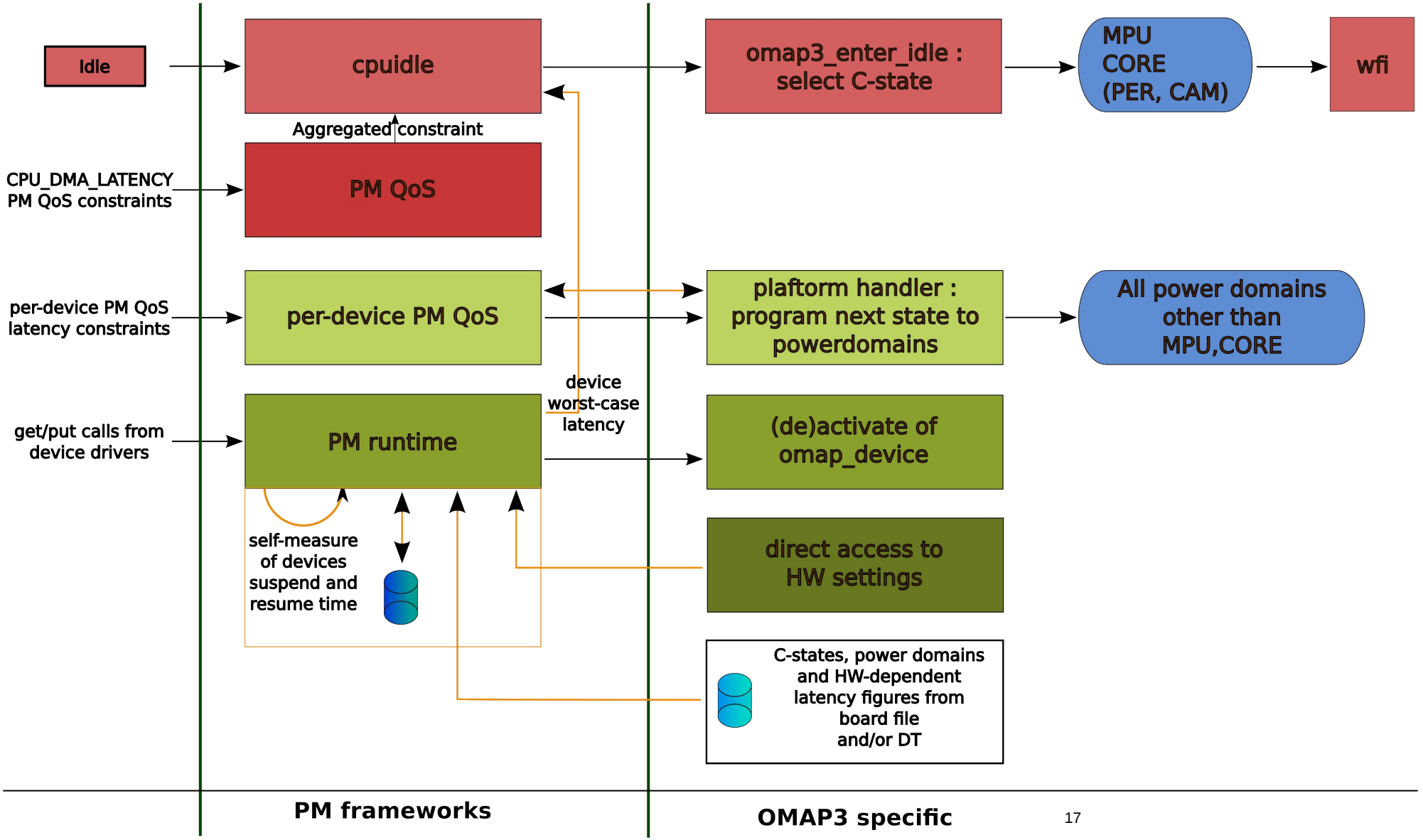
$latency_{SW}$ =
time for the SW to save/restore the context of an IP block

$latency_{SoC}$ =
time for the SoC HW to change an IP block state. Includes the Power Domain state transition, DPLL stop/relock etc.

$latency_{External\ HW}$ =
time to stop/restart the external HW. Ex : external crystal oscillator, external power supply etc.

Note : every latency factor might be divided into smaller factors. E.g. : On OMAP a DPLL can feed multiple power domains.

New model


Idle
cpuidle
Aggregated constraint
CPU_DMA_LATENCY
PM QoS constraints
PM QoS
per-device PM QoS
latency constraints
per-device PM QoS
get/put calls from
device drivers
PM runtime
device
worst-case
latency
self-measure
of devices
suspend and
resume time
omap3_enter_idle :
select C-state
MPU
CORE
(PER, CAM)
wfi
plaftorm handler :
program next state to
powerdomains
All power domains
other than
MPU,CORE
(de)activate of
omap_device
direct access to
HW settings
C-states, power domains
and HW-dependent
latency figures from
board file
and/or DT
PM frameworks
OMAP3 specific

Impact on the current code

Reduce the measurement results into factors

From the model, derive the independent factors for the overall latency.
Differentiate the fixed factors from the variable ones (i.e. At HW level a power domain transition worst-case latency is fixed).

Pass the latency data along with board-specific data

From the board files.
From (DT) Device Tree data.

Note : Which data to pass from board files or DT ? Cf. Discussions on l-a-k & l-o MLs.

Introduce functions to calculate the devices and power domains worst case latency

Clean-up of the code that directly touches the HW settings which have an impact on the overall latency.
When a HW setting is touched, re-calculate the overall worst case latency.

Impact on the current code

Self-measuremente of devices (de)activate worst case latency

Ideally:

Implement the self-measurement *in a generic way* in devices runtime PM : in generic power domain code or in devices get/put functions.

Real world:

1. OMAP has its own implementation of clock/power/voltage domains

2. The generic power domain code has no provision for multiple power states, which OMAP is using (ON, INACTIVE, CSWR, OSWR, OFF), which prevents OMAP code for using it (for now).

Question: How to integrate the full solution ?

Impact on the current code

Self-measuremente of devices (de)activate worst case latency

Question: How to integrate the full solution ?

=> Implement the features in logical steps:

1. provide a reference implementation using the OMAP code,

2. bring the concept of multiple power domains states in the generic framework,

3. change OMAP code to use the generic power domains,

4. repeat 2-3 for clocks (hint : common clock framework) and voltages,

5. port the self-measurement feature in generic code (runtime PM)

Impact on the current code

Self-measuremente of devices (de)activate worst case latency

Notes:

1. (De)activate a device can cover the overall latency by propagation through the clock/power/voltage domains.
   So use the clock, power and voltage domains and DPLLs ***use count*** field to differentiate the measurement of the device-only latency from the other factors,

2. The ***use count*** field needs to be accessible to runtime PM from the generic clock/power/voltage domains frameworks.