Fosdem - ELC 2012

A new model for the system and devices latency

Jean Pihet <j-pihet@ti.com>

Introduction

Background

What is the 'latency' ?

There is some overhead when *a part of the system* goes to a low power mode in idle, both at suspend and resume times.
The allowed latency needs to be taken into account when deciding the next low power state.
'A part of the system' = SW, HW SoC, HW external.

How to specify the allowed latency ?

The PM QoS framework allows the kernel and user to specify the allowed latency.
The framework calculates the aggregated constraint value and calls the registered platform-specific handlers in order to apply the constraints at lower level.

Cf. Documentation/power/pm_qos_interface.txt for the available classes:

- PM QoS classes for cpu_dma_latency, network_latency, network_throughput.
- The per-device PM QoS framework provides the API to manage the per-device latency constraints.

Introduction

Background

What is the point of controlling the latency ?

The point is to dynamically optimize the power consumption of all system components.

Knowing the allowed latency (from the constraints) and the expected worst-case latency allows to choose the optimum power state.

Introduction

Terminology

- Latency : time to react to an external event, e.g. time spent to execute the handler code after an IRQ, time spent to execute driver code from an external wake-up event.
- HW latency : latency introduced by the HW to transition between power states.
- SW latency : time for the SW to execute low power transition code, e.g. IP block save & restore, caches flush/invalidate etc.
- System : 'everything needed to execute the kernel code', e.g. on OMAP3, system = CPU0 + CORE (main memory, caches, IRQ controller...).
- Per-device latency : latency of a device (or peripheral). The per-device PM QoS framework allows to control the devices states from the allowed devices latency.
- Cpuidle : framework that controls the CPUs low power states (=C-states), from the allowed system latency. Note : Is being abused to control the system state.
- PM runtime : framework that allows the dynamic switching of resources.

Introduction

OMAP SoC PM

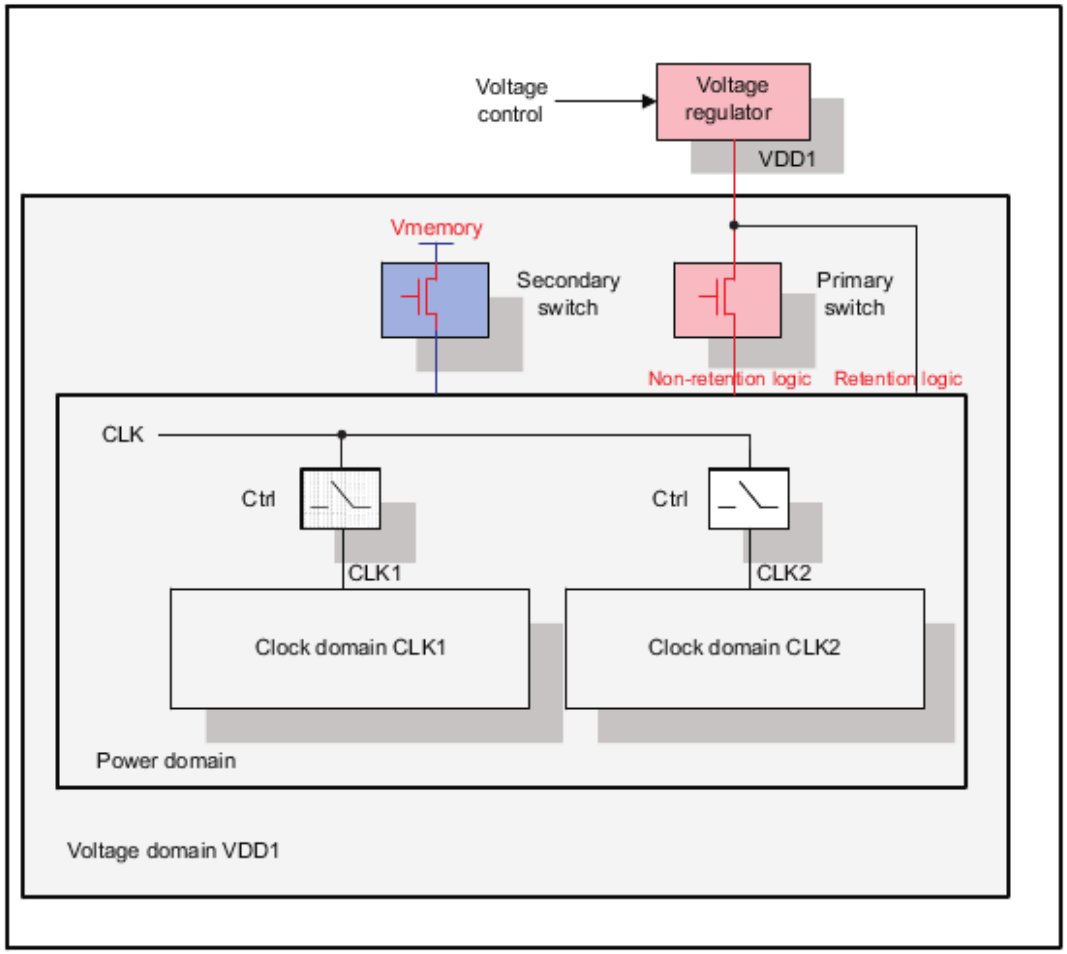
Dynamic and hierarchical PM

- Clock->Pwrdm->Voltdm->External Voltage Regulators
- Clock->DPLL->External Oscillators

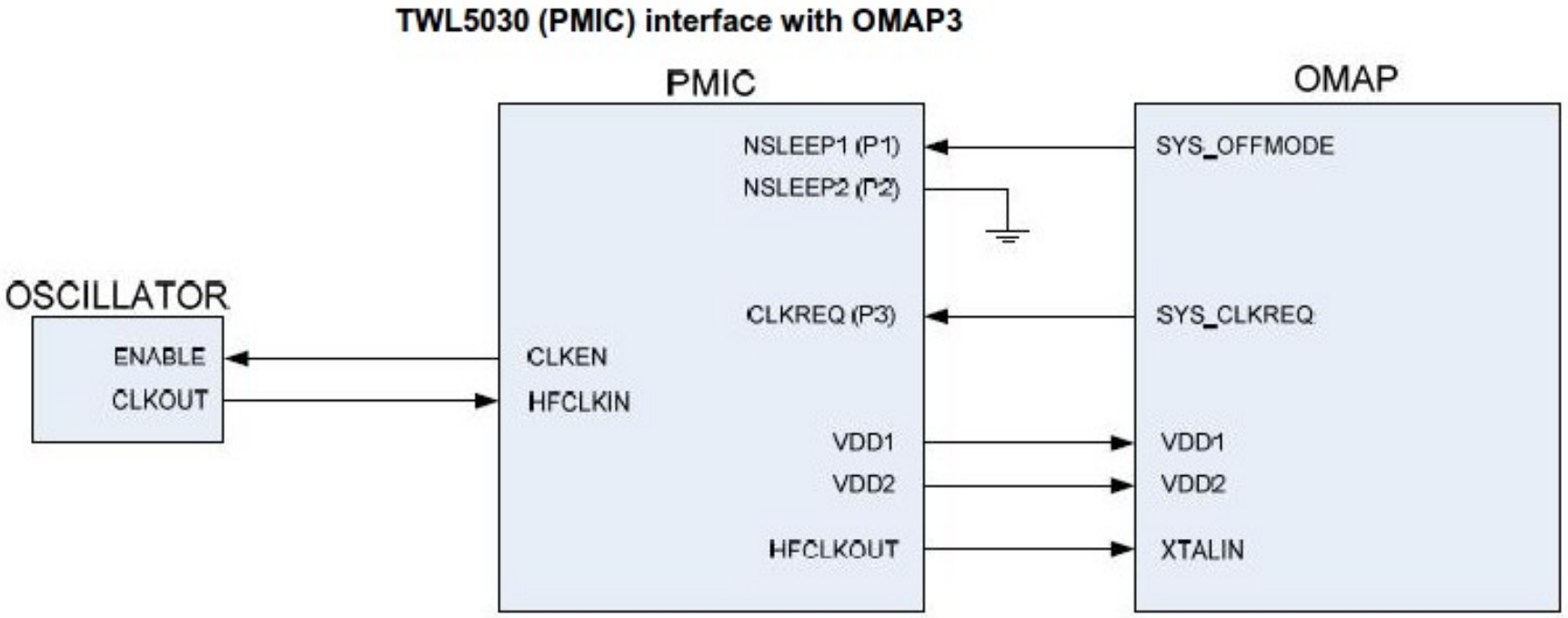
The HW latency depends on system settings

- The behavior of the voltage regulators and external oscillators depends on various system settings.
- The system settings can be dynamically controlled.
- E.g. OMAP <-> PMIC signals : SYS_CLKREQ, SYS_OFFMODE.

Figure 4-7. Voltage, Power, and Clock Domain Hierarchical Architecture

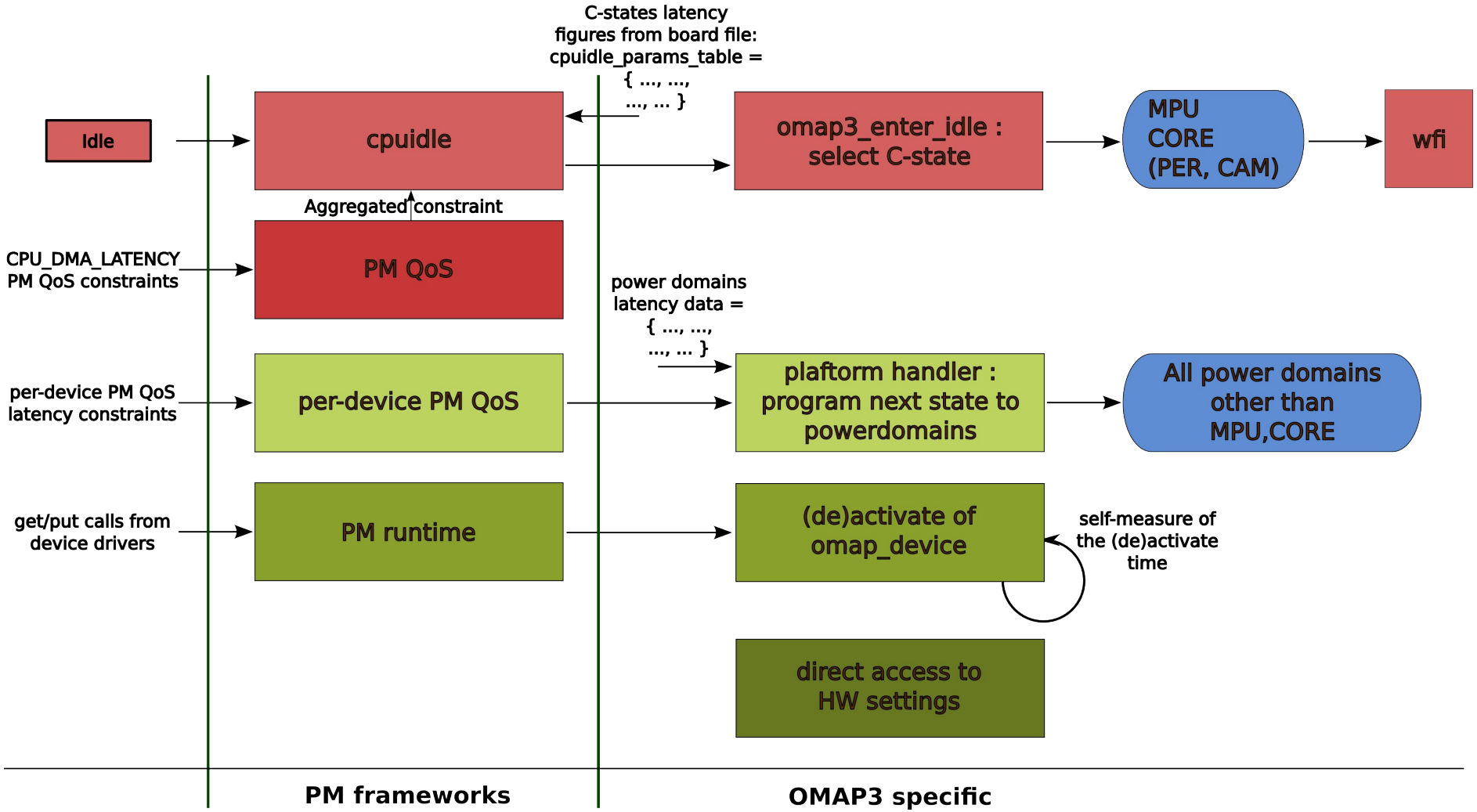


Source : OMAP3430 TRM, public version



Source : TWL4030 Power Scripts [2]

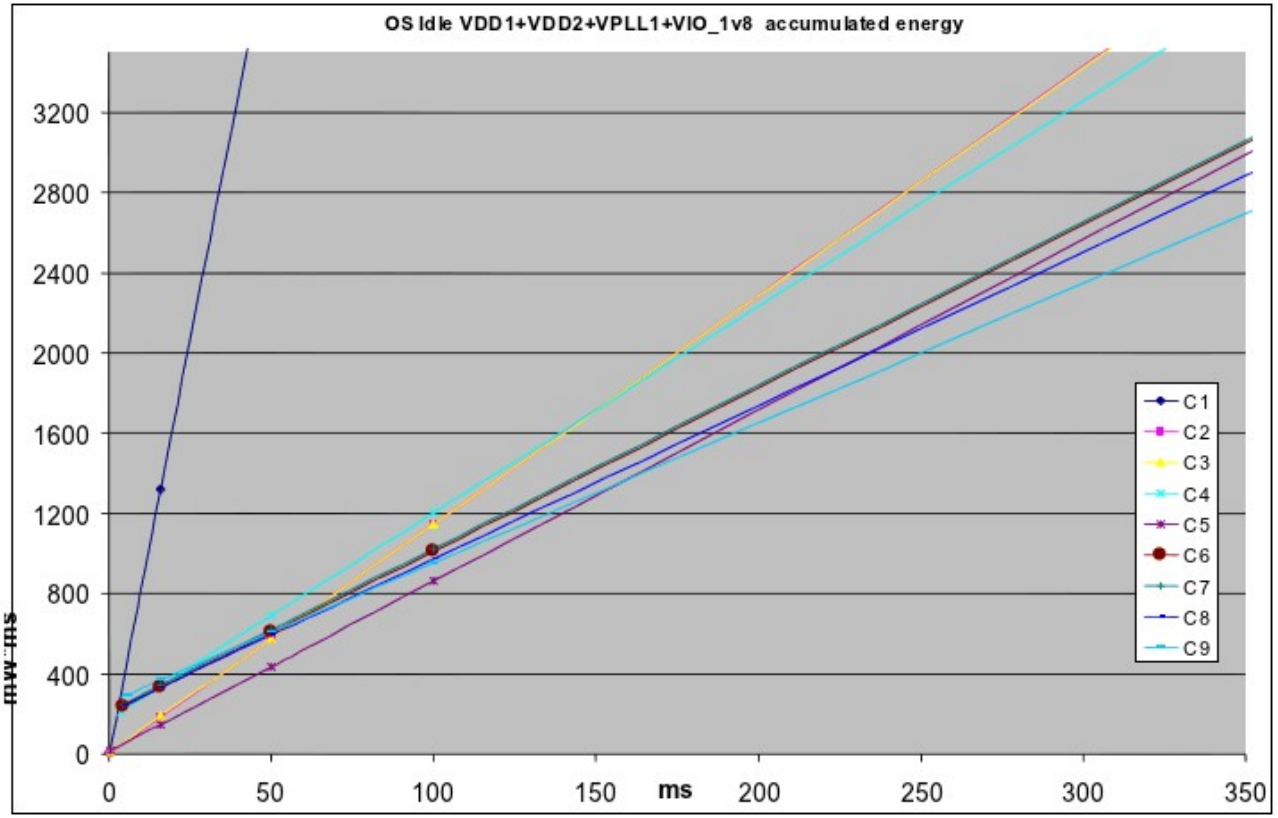
Current model



Current model : latency figures

cpuidle latency figures

From [1] : measuring the timing and the current consumption (thanks to the TI PSI team!) leads to the following graph of the energy spent vs time :



Current model : latency figures

cpuidle latency figures (cont'd)

Derive some usable figures from the measurements :

- Identify the energy-wise interesting C-states and threshold values (C1, C3, C5, C9)
- Aggregate the timings results. From the various sources of data the following figures are derived for all C-states (timings in us).

C-state	Sleep lat	Wake-up lat	Threshold
C1: MPU WFI/ON - CORE ON	73.6	78	151.6
C2: MPU WFI - CORE INA	165	88.16	345 (1)
C3: MPU CSWR - CORE INA	163	182	345
C4: MPU OFF - CORE INA	2852	605	150000 (2)
C5: MPU CSWR - CORE CSWR	800	366 (3)	2120
C6: MPU OFF - CORE CSWR	4080	801	215000 (1)
C7: MPU OFF - CORE OFF	4300	12933 (4)	215000 (5)

Notes: produce the actual figures (to be used in the code) involves a lot of operations : interpolation, intersection (linear algebra) etc.

Current model : latency figures

Inject the figures into the cpuidle framework :

```
/*
 * The MPU latency and threshold values for the C-states are the worst case
 * values from the HW and SW, as described in details at
 * http://www.omappedia.org/wiki/Power_Management_Device_Latencies_Measurement#cpuidle_results
 *
 * Measurements conditions and remarks:
 *   . the measurements have been performed at OPP50
 *   . the sys_offmode signal is not supported and so not used for the
 *     measurements. Instead the latency and threshold values for C9 are
 *     corrected with the value for Triton 2, which is 11.5ms
 *   . the sys_clkreq signal is not used and so a correction is needed - TBD
 *   . the sys_clkoff signal is supported, this value need to be corrected with
 *     the correct value of SYSCLK on/off timings (1ms for sysclk on, 2.5ms
 *     for sysclk off)
 *   . in order to force the cpuidle algorithm to chose the power efficient
 *     C-states (C1, C3, C5, C7) in preference, the other C-states have a
 *     threshold value equal to the next power efficient C-state
 *
 * The latency and threshold values can be overriden by data from the board
 * files, using omap3_pm_init_cpuidle.
 */
static struct cpuidle_params cpuidle_params_table[] = {
        /* C1 . MPU WFI + Core active */
        {73 + 78, 152, 1},
        /* C2 . MPU WFI + Core inactive */
        {165 + 88, 345, 1},
        /* C3 . MPU CSWR + Core inactive */
        {163 + 182, 345, 1},
        /* C4 . MPU OFF + Core inactive */
        {2852 + 605, 150000, 1},
        /* C5 . MPU RET + Core RET */
        {800 + 366, 2120, 1},
        /* C6 . MPU OFF + Core RET */
        {4080 + 801, 215000, 1},
        /* C7 . MPU OFF + Core OFF */
        {4300 + 13000, 215000, 1},
};
#define OMAP3_NUM_STATES ARRAY_SIZE(cpuidle_params_table)
```

Current model : latency figures

Power domains latency figures

From [1] :

Since cpuidle only manages the MPU and CORE the wake-up latency values for the other power domains must be measured separately, by adjusting the target states of the power domains (in /debug/pm_debug/xxxx_pwrdm/suspend).

The significative power domains latencies are derived from the measurements as follows:

Power Domain	RET latency	OFF latency	Table location
MPU	121	1830	(5), (6)
NEON	0	0	Included in MPU transitions?
CORE	153	3082	(3), (4)
PER	0	671	(1), (2)

Notes:
sys_clkreq and sys_offmode are not supported

Current model : latency figures

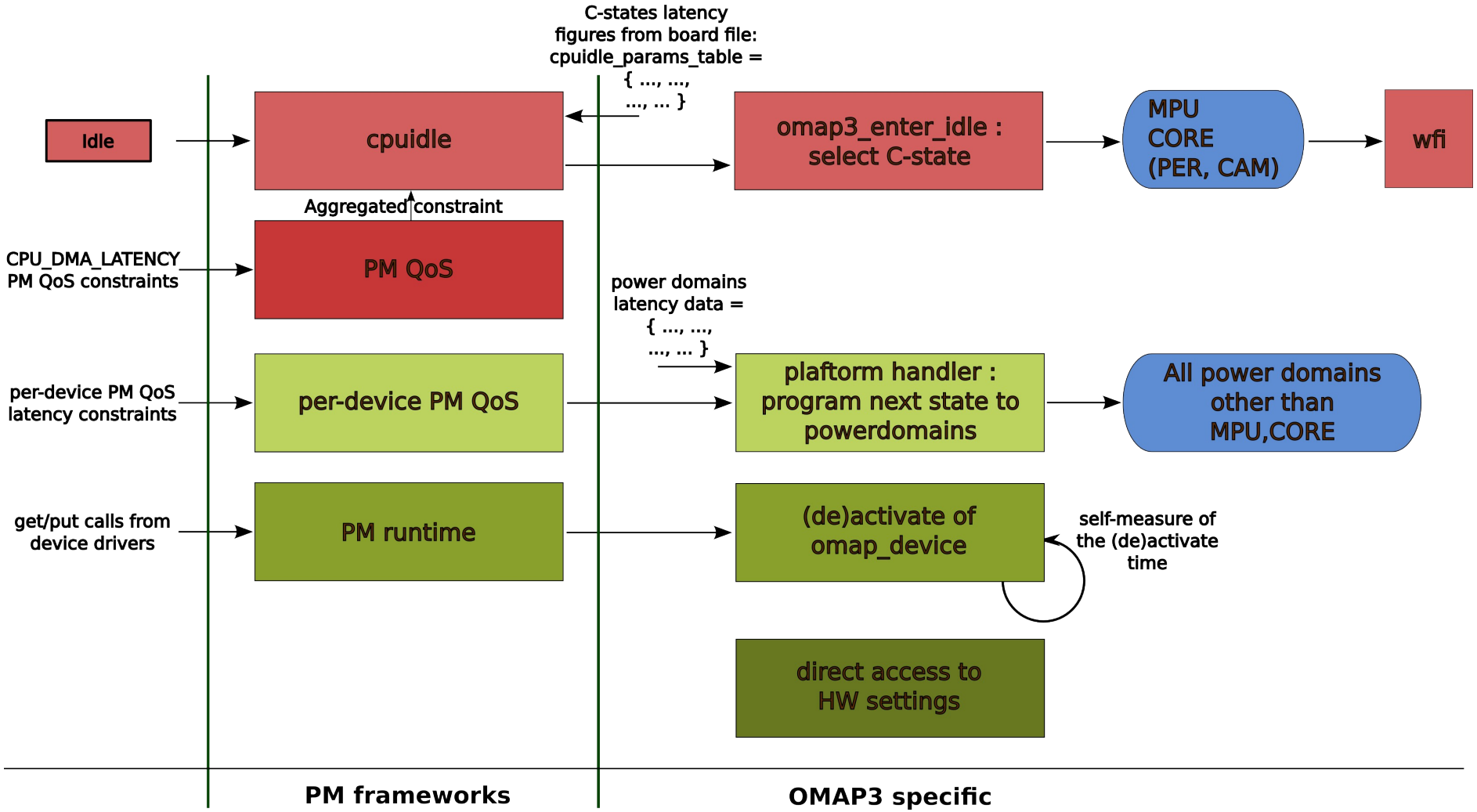
Inject the figures into the powercpuidle framework :

```
/*
 * Powerdomains
 *
 * The wakeup_lat values are derived from HW and SW measurements on
 * the actual target. For more details cf.
 * http://www.omappedia.org/wiki/Power_Management_Device_Latencies_Measurement#Results_for_individual_power_domains
 *
 * Note: the latency figures for MPU, PER, CORE, NEON have been obtained
 * from actual measurements.
 * The latency figures for the other power domains are preliminary and
 * shall be added.
 *
 * Note: only the SW restore timing values are taken into account.
 * The HW impact of the sys_clkreq and sys_offmode signals is not taken
 * into account - TDB
 */

static struct powerdomain mpu_3xxx_pwrdm = {
        .name             = "mpu_pwrdm",
        ...
        .wakeup_lat = {
                [PWRDM_FUNC_PWRST_OFF] = 1830,
                [PWRDM_FUNC_PWRST_OSWR] = UNSUP_STATE,
                [PWRDM_FUNC_PWRST_CSWR] = 121,
                [PWRDM_FUNC_PWRST_INACTIVE] = UNSUP_STATE,
                [PWRDM_FUNC_PWRST_ON] = 0,
        },
        ...
};

static struct powerdomain core_3xxx_es3_1_pwrdm = {
        .name             = "core_pwrdm",
        ...
        .wakeup_lat = {
                [PWRDM_FUNC_PWRST_OFF] = 3082,
                [PWRDM_FUNC_PWRST_OSWR] = UNSUP_STATE,
                [PWRDM_FUNC_PWRST_CSWR] = 153,
                [PWRDM_FUNC_PWRST_INACTIVE] = UNSUP_STATE,
                [PWRDM_FUNC_PWRST_ON] = 0,
        },
        ...
};
```

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.

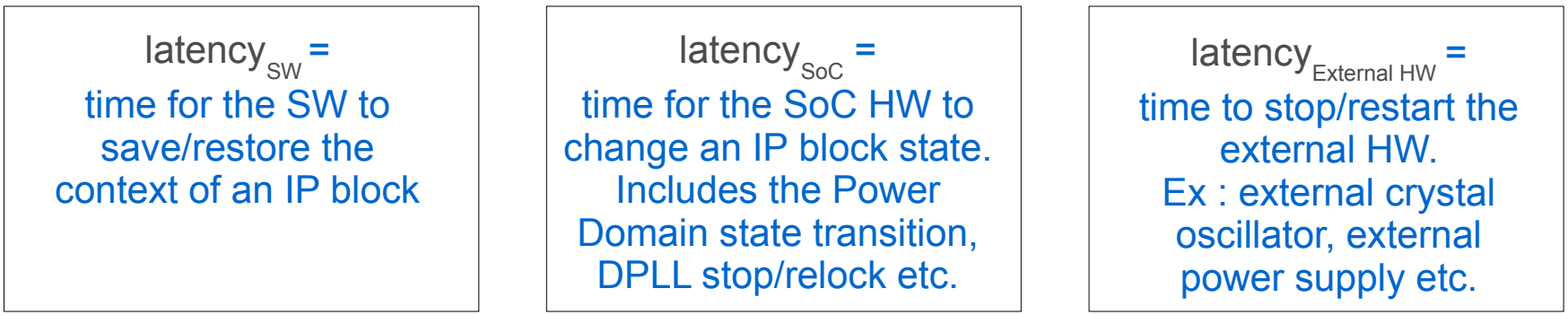
Measuring the latency of all power domains is difficult : take measurements, derive energy graphs, calculate intersections, adapt to missing key parameters etc.

Solution proposal

Overall latency calculation

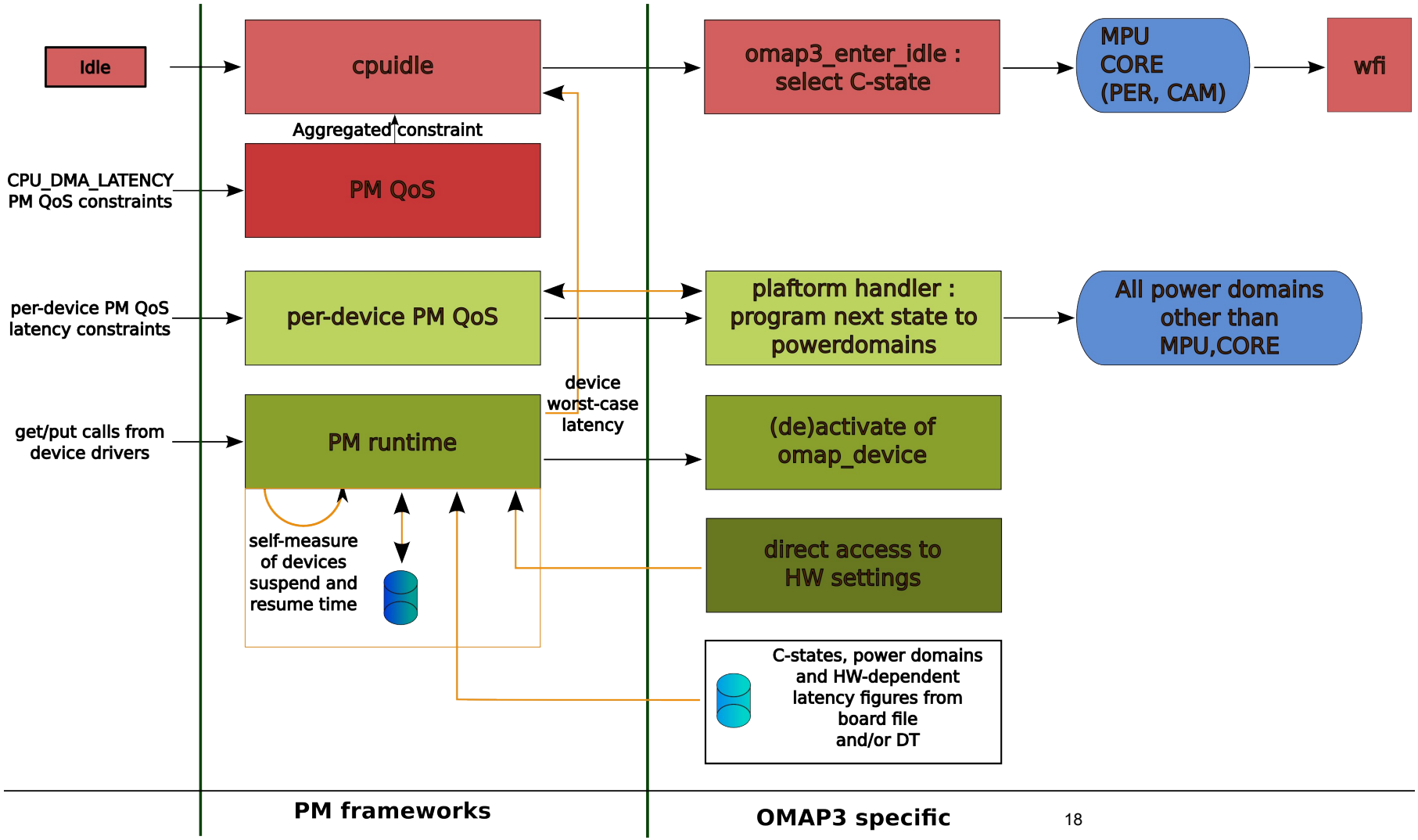
We need a model which breaks down the overall latency into the latencies from every contributor :

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



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
omap3_enter_idle : select C-state
MPU CORE (PER, CAM)
wfi
Aggregated constraint
CPU_DMA_LATENCY PM QoS constraints
PM QoS
per-device PM QoS latency constraints
per-device PM QoS
plaftorm handler : program next state to powerdomains
All power domains other than MPU,CORE
device worst-case latency
get/put calls from device drivers
PM runtime
(de)activate of omap_device
self-measure of devices suspend and resume time
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.

Next steps

Start the discussions with the maintainers (here and on MLs)

- lkml, linux-pm
- linux-arm(-kernel), linux-omap

- Points to discuss :

. Generic clock/power/voltage domains implementation vs OMAP specific code

. Proper use of the *use count* field to identify the device-only latency from the propagated latency

. Identify the impact on the PM runtime latency measurement code

. (OMAP) What are the independent factors ? What are the settings which have an impact on the latency ?

. (OMAP) How to pass the SoC and board specific data (board file, DT) ?

Write and submit code (!)

Links

Omappedia wiki

PM debug & profiling
http://www.omappedia.org/wiki/Power_Management_Debug_and_Profiling

[1] PM devices latency measurements
http://www.omappedia.org/wiki/Power_Management_Device_Latencies_Measurement

[2] TWL4030 Power Scripts
http://omappedia.org/wiki/TWL4030_power_scripts

Submitted patches and discussions on MLs

OMAP specific patches for per-device latency
http://www.spinics.net/lists/linux-omap/msg61692.html

The slides for this presentation

are posted at [1]

Thank you !

Questions ?

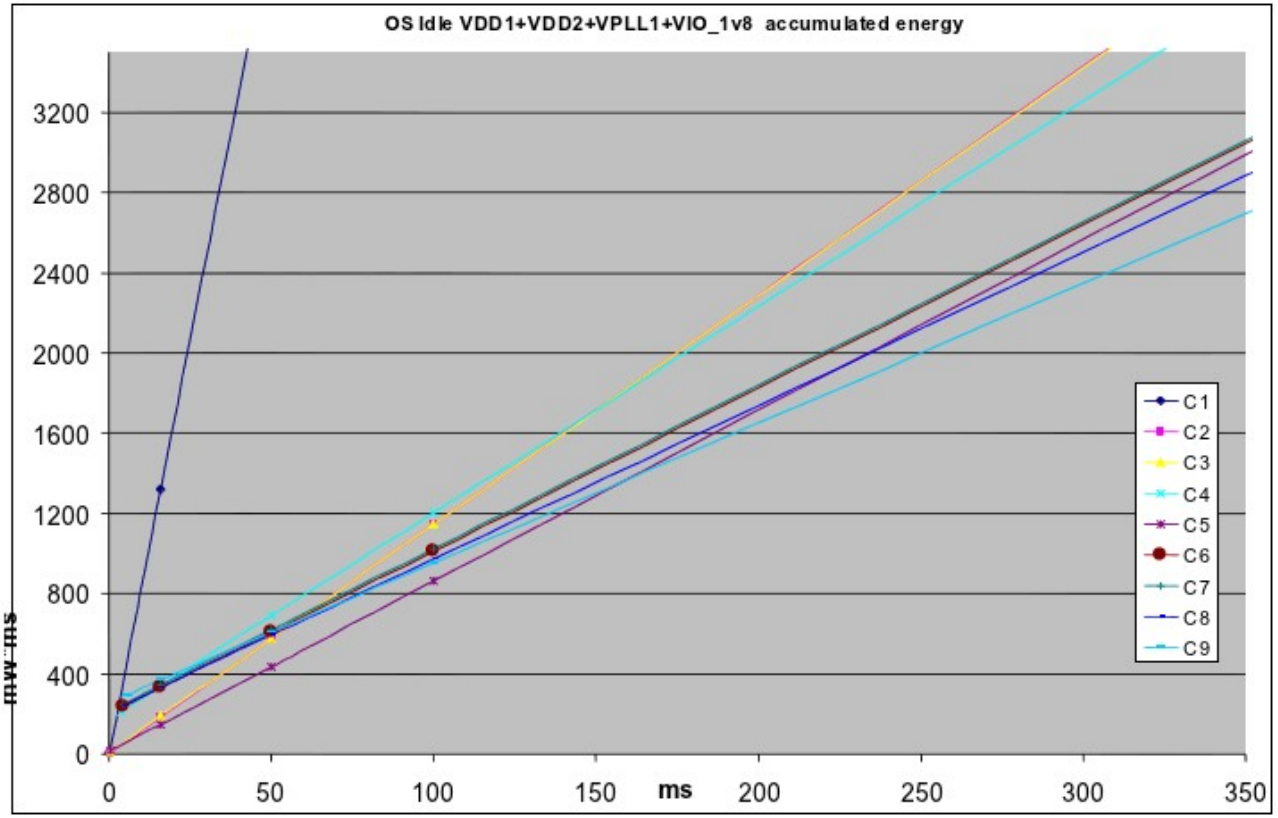

Embedded Linux Conference
February 15 - 17, 2012 ▪ Redwood Shores, CA

Back-up slides

Current model : latency figures

cpuidle latency figures

From [1] : measuring the timing and the current consumption (thanks to the TI PSI team!) leads to the following graph of the energy spent vs time :



Current model : latency figures

cpuidle latency figures (cont'd)

Taking the minimum energy from the graph allows to identify the 4 energy-wise interesting C-states: C1, C3, C5, C9 and the threshold time for those C-states to be efficient.

Aggregated timings results

From the various sources of data the following figures are derived for all C-states (timings in us).

C-state	Sleep lat	Wake-up lat	Threshold
C1: MPU WFI/ON - CORE ON	73.6	78	151.6
C2: MPU WFI - CORE INA	165	88.16	345 (1)
C3: MPU CSWR - CORE INA	163	182	345
C4: MPU OFF - CORE INA	2852	605	150000 (2)
C5: MPU CSWR - CORE CSWR	800	366 (3)	2120
C6: MPU OFF - CORE CSWR	4080	801	215000 (1)
C7: MPU OFF - CORE OFF	4300	12933 (4)	215000 (5)

Notes:
The power efficient C-states are identifed as C1, C3, C5, C7
(1) When not measured, the threshold value equals to the next power efficient C-state
(2) The threshold value is derived using the *intersection of* C3 and C4 in the graph
(3) No *sys_clkoff* is supported, this value need to be corrected
(4) Addition of HW and SW parts, using [2]
(5) The threshold value calculation is the intersection of the lines in the graph, using linear algebra

Current model : latency figures

Inject the figures into the cpuidle framework :

```
/*
 * The MPU latency and threshold values for the C-states are the worst case
 * values from the HW and SW, as described in details at
 * http://www.omappedia.org/wiki/Power_Management_Device_Latencies_Measurement#cpuidle_results
 *
 * Measurements conditions and remarks:
 *   . the measurements have been performed at OPP50
 *   . the sys_offmode signal is not supported and so not used for the
 *     measurements. Instead the latency and threshold values for C9 are
 *     corrected with the value for Triton 2, which is 11.5ms
 *   . the sys_clkreq signal is not used and so a correction is needed - TBD
 *   . the sys_clkoff signal is supported, this value need to be corrected with
 *     the correct value of SYSCLK on/off timings (1ms for sysclk on, 2.5ms
 *     for sysclk off)
 *   . in order to force the cpuidle algorithm to chose the power efficient
 *     C-states (C1, C3, C5, C7) in preference, the other C-states have a
 *     threshold value equal to the next power efficient C-state
 *
 * The latency and threshold values can be overriden by data from the board
 * files, using omap3_pm_init_cpuidle.
 */
static struct cpuidle_params cpuidle_params_table[] = {
        /* C1 . MPU WFI + Core active */
        {73 + 78, 152, 1},
        /* C2 . MPU WFI + Core inactive */
        {165 + 88, 345, 1},
        /* C3 . MPU CSWR + Core inactive */
        {163 + 182, 345, 1},
        /* C4 . MPU OFF + Core inactive */
        {2852 + 605, 150000, 1},
        /* C5 . MPU RET + Core RET */
        {800 + 366, 2120, 1},
        /* C6 . MPU OFF + Core RET */
        {4080 + 801, 215000, 1},
        /* C7 . MPU OFF + Core OFF */
        {4300 + 13000, 215000, 1},
};
#define OMAP3_NUM_STATES ARRAY_SIZE(cpuidle_params_table)
```