Consolidating Linux Power Management on ARM multiprocessor systems

L.Pieralisi

27/10/2011 - ELC Europe

ARM

Outline

Outline

Power Management Fundamentals

”If you’ve heard this story before, don’t stop me, because I’d like to hear it again”

G.Marx

Power Management Fundamentals

Total Energy Consumption

$$E = \int_0^t (CV_{dd}^2 f_c + V_{dd} I_{lkg})\, dt$$

Dynamic Energy Consumption + **Leakage Energy Consumption**

$$\int_0^t CV_{dd}^2 f_c \quad + \quad \int_0^t V_{dd} I_{lkg}$$

Power Management Fundamentals

Total Energy Consumption

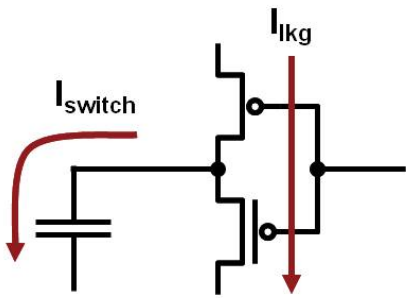
$$E = \int_0^t (CV_{dd}^2 f_c + V_{dd} I_{lkg})\, dt$$

To minimize I_{switch}

Reduce voltage
Reduce frequency
Clock gating
Reduce switched capacitance

To minimize I_{lkg}

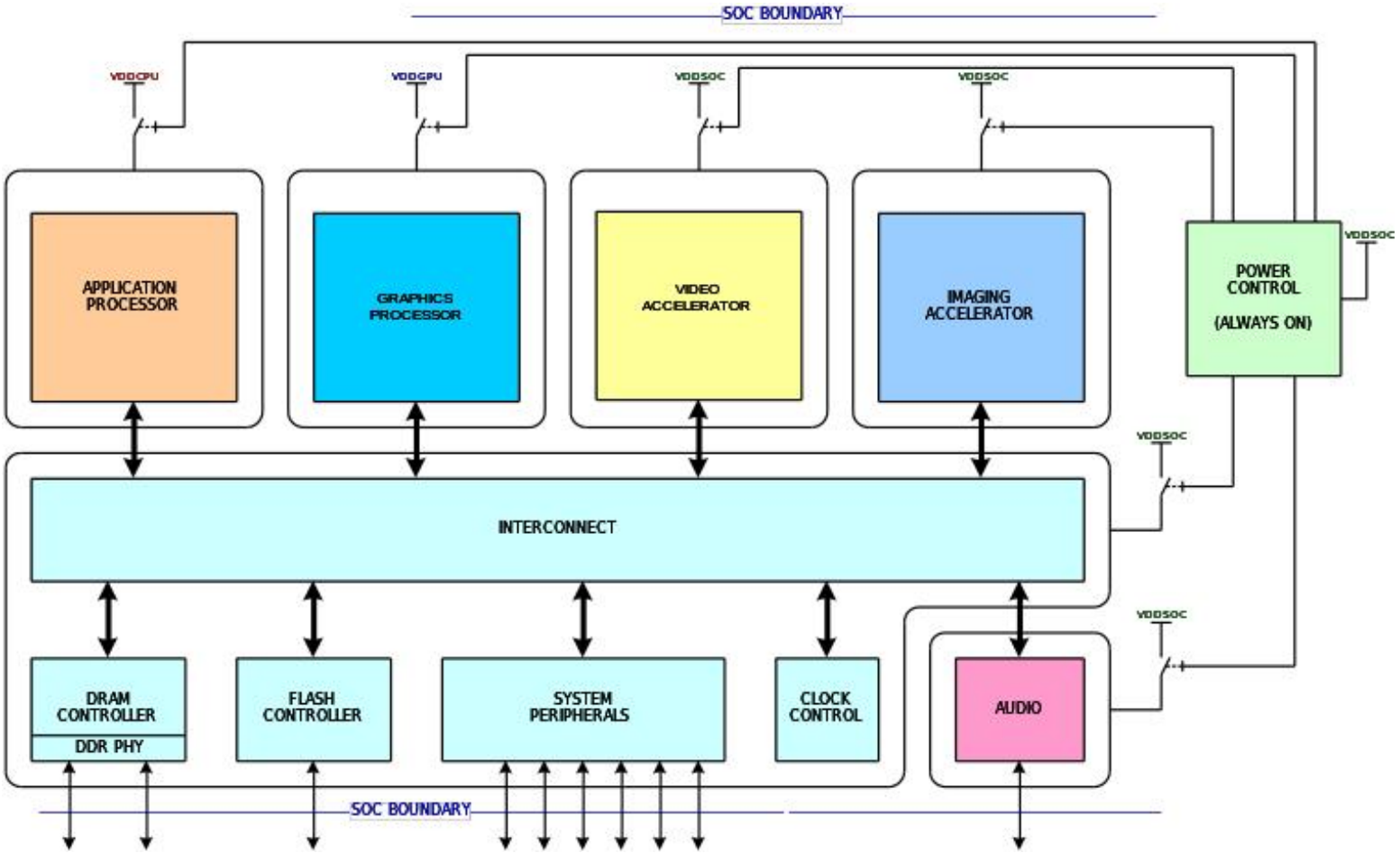
Reduce voltage
Less Leaky transistors
Power gating



SoC Technology and Power Consumption

- Dynamic power, frequency scaling
- Static (leakage) power, G (Generic) process, LP (Low Power) process
- Temperature variations
- RAM retention
- CPU vs. IO devices
- Need for more agressive and holistic power management

Power Managed SoC Example



Kernel Power Management Mechanics

System suspend	User space forces system to sleep
CPU idle	Idle threads trigger sleep states
CPU freq	CPU Frequency scaling
Runtime PM	Devices Power Management
CPU hotplug	Remove a CPU from the running system

Focus on CPU Power Management (PM)

Get code in the kernel to enable efficient and stable core support for hotplug, cpu suspend and cpu idle

Outline

The Case for Generalization

- SMP power down procedure is common and complex
- Code can be shared across different platforms
- Written, debugged, tested once for all

Our Goal

Merge in the kernel common code reusable by all partners

ARM A9 SMP CPU Shutdown Procedure

1. save per CPU peripherals (IC, VFP, PMU)
2. save CPU registers
3. clean L1 D$
4. clean state from L2
5. disable L1 D$ allocation
6. clean L1 D$
7. exit coherency
8. programme SCU CPU Power CTRL register
9. call wfi (wait for interrupt)

ARM A9 SMP CPU Shutdown Procedure

1. save per CPU peripherals (IC, VFP, PMU)
2. save CPU registers
3. clean L1 D$
4. clean state from L2
5. disable L1 D$ allocation
6. clean L1 D$
7. exit coherency
8. programme SCU CPU Power CTRL register
9. call wfi (wait for interrupt)

ARM A9 SMP CPU Shutdown Procedure

1. save per CPU peripherals (IC, VFP, PMU)
2. save CPU registers
3. clean L1 D$
4. clean state from L2
5. disable L1 D$ allocation
6. clean L1 D$
7. exit coherency
8. programme SCU CPU Power CTRL register
9. call wfi (wait for interrupt)

ARM A9 SMP CPU Shutdown Procedure

1. save per CPU peripherals (IC, VFP, PMU)
2. save CPU registers
3. clean L1 D$
4. clean state from L2
5. disable L1 D$ allocation
6. clean L1 D$
7. exit coherency
8. programme SCU CPU Power CTRL register
9. call wfi (wait for interrupt)

ARM A9 SMP CPU Shutdown Procedure

1. save per CPU peripherals (IC, VFP, PMU)
2. save CPU registers
3. clean L1 D$
4. clean state from L2
5. disable L1 D$ allocation
6. clean L1 D$
7. exit coherency
8. programme SCU CPU Power CTRL register
9. call wfi (wait for interrupt)

ARM A9 SMP CPU Shutdown Procedure

1. save per CPU peripherals (IC, VFP, PMU)
2. save CPU registers
3. clean L1 D$
4. clean state from L2
5. disable L1 D$ allocation
6. clean L1 D$
7. exit coherency
8. programme SCU CPU Power CTRL register
9. call wfi (wait for interrupt)

This is the standard procedure that must be adopted by all platforms, for cpu hotplug (cache cleaning and wfi), suspend and idle

ARM A9 SMP CPU Shutdown Procedure

1. save per CPU peripherals (IC, VFP, PMU)
2. save CPU registers
3. clean L1 D$
4. clean state from L2
5. disable L1 D$ allocation
6. clean L1 D$
7. exit coherency
8. programme SCU CPU Power CTRL register
9. call wfi (wait for interrupt)

This is the standard procedure that must be adopted by all platforms, for cpu hotplug (cache cleaning and wfi), suspend and idle

idle
notifiers

Outline

ARM Common PM Code Components

- CPU PM notifiers
- cpu suspend/resume
- L2 suspend/resume
- CPUs coordination, hotplug

CPU PM notifiers (1/3)

- Introduced by C.Cross to overcome code duplication in idle and suspend code path
- CPU events and CLUSTER events
- GIC, VFP, PMU

CPU PM notifiers (2/3)

```
static int cpu_pm_notify(enum cpu_pm_event event, int nr_to_call, int *nr_calls)
{
        int ret;

        ret = __raw_notifier_call_chain(&cpu_pm_notifier_chain, event, NULL,
                                nr_to_call, nr_calls);

        return notifier_to_errno(ret);
}

int cpu_pm_enter(void)
{
        [...]

        ret = cpu_pm_notify(CPU_PM_ENTER, -1, &nr_calls);
        if (ret)
                cpu_pm_notify(CPU_PM_ENTER_FAILED, nr_calls - 1, NULL);

        [...]

        return ret;
}

//CPU shutdown
cpu_pm_{enter,exit}();
//Cluster shutdown
cpu_cluster_pm_{enter,exit}();
```

CPU PM notifiers (3/3)

```
static int gic_notifier(struct notifier_block *self, unsigned long cmd, void *v)
{
        int i;

        [...]
        switch (cmd) {
        case CPU_PM_ENTER:
                gic_cpu_save(i);
                break;
        case CPU_PM_ENTER_FAILED:
        case CPU_PM_EXIT:
                gic_cpu_restore(i);
                break;
        case CPU_CLUSTER_PM_ENTER:
                gic_dist_save(i);
                break;
        case CPU_CLUSTER_PM_ENTER_FAILED:
        case CPU_CLUSTER_PM_EXIT:
                gic_dist_restore(i);
                break;
        }

        return NOTIFY_OK;
}

static struct notifier_block gic_notifier_block = {
        .notifier_call = gic_notifier,
};
```

▸ A9 shutdown

CPU suspend (1/3)

- Introduced by R.King to consolidate existing (and duplicated) code across diffent ARM platforms
- save/restore core registers, clean L1 and some bits of L2
- L2 RAM retention handling poses further challenges

CPU suspend (2/3)

- 1:1 mapping page tables cloned from $\mathrm{init\_mm}$
- C API, generic for all ARM architectures

```
int cpu_suspend(unsigned long arg, int (*fn)(unsigned long))
{
        struct mm_struct *mm = current->active_mm;
        int ret;

        if (!suspend_pgd)
                return -EINVAL;

        [...]

        ret = __cpu_suspend(arg, fn);
        if (ret == 0) {
                cpu_switch_mm(mm->pgd, mm);
                local_flush_tlb_all();
        }

        return ret;
}
```

CPU suspend (3/3)

- registers saved on the stack

```
void __cpu_suspend_save(u32 *ptr, u32 ptrsz, u32 sp, u32 *save_ptr)
{
        *save_ptr = virt_to_phys(ptr);

        /* This must correspond to the LDM in cpu_resume() assembly */
        *ptr++ = virt_to_phys(suspend_pgd);
        *ptr++ = sp;
        *ptr++ = virt_to_phys(cpu_do_resume);

        cpu_do_suspend(ptr);
}
```

CPU suspend (3/3)

- registers saved on the stack
- L1 complete cleaning

```
void __cpu_suspend_save(u32 *ptr, u32 ptrsz, u32 sp, u32 *save_ptr)
{
        *save_ptr = virt_to_phys(ptr);

        /* This must correspond to the LDM in cpu_resume() assembly */
        *ptr++ = virt_to_phys(suspend_pgd);
        *ptr++ = sp;
        *ptr++ = virt_to_phys(cpu_do_resume);

        cpu_do_suspend(ptr);

        flush_cache_all();
}
```

CPU suspend (3/3)

- registers saved on the stack
- L1 complete cleaning
- L2 partial cleaning

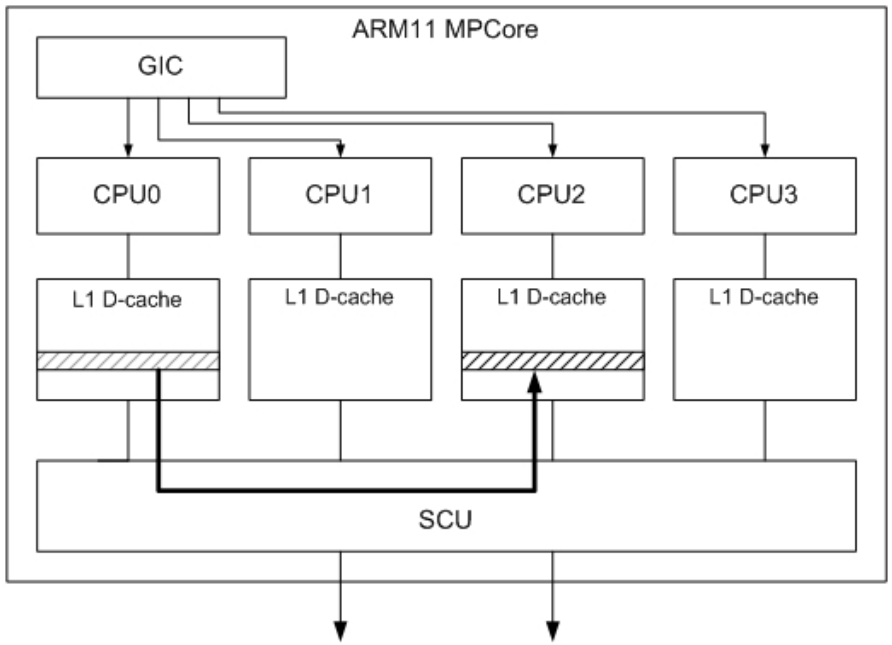
```
void __cpu_suspend_save(u32 *ptr, u32 ptrsz, u32 sp, u32 *save_ptr)
{
        *save_ptr = virt_to_phys(ptr);

        /* This must correspond to the LDM in cpu_resume() assembly */
        *ptr++ = virt_to_phys(suspend_pgd);
        *ptr++ = sp;
        *ptr++ = virt_to_phys(cpu_do_resume);

        cpu_do_suspend(ptr);

        flush_cache_all();
        outer_clean_range(*save_ptr, *save_ptr + ptrsz);
        outer_clean_range(virt_to_phys(save_ptr),
                          virt_to_phys(save_ptr) + sizeof(*save_ptr));
}
```

We Are Not Done, Yet: Cache-to-Cache migration

”Once you eliminate your number one problem, number two gets a promotion.”

G.Weinger

We Are Not Done, Yet: Cache-to-Cache migration



- SCU keeps a copy of D$ cache TAG RAMs
- To avoid data traffic A9 MPCore moves dirty lines across cores
- Lower L1 bus traffic
- Dirty data might be fetched from another core during power-down sequence

We Are Not Done, Yet: Cache-to-Cache migration

- When the suspend finisher is called L1 is still allocating
- accessing current implies accessing the sp
- Snooping Direct Data Intervention (DDI), CPU might pull dirty line in

```
ENTRY(disable_clean_inv_dcache_v7_all)
        stmfd   sp!, {r4-r5, r7, r9-r11, lr}
        mrc     p15, 0, r3, c1, c0, 0
        bic     r3, #4                          @ clear C bit
        mcr     p15, 0, r3, c1, c0, 0
        isb

        bl      v7_flush_dcache_all
        mrc     p15, 0, r0, c1, c0, 1
        bic     r0, r0, #0x40                   @ exit SMP
        mcr     p15, 0, r0, c1, c0, 1
        ldmfd   sp!, {r4-r5, r7, r9-r11, pc}
ENDPROC(disable_clean_inv_dcache_v7_all)
```

We Are Not Done, Yet: Cache-to-Cache migration

- When the suspend finisher is called L1 is still allocating
- accessing current implies accessing the sp
- Snooping Direct Data Intervention (DDI), CPU might pull dirty line in

```
ENTRY(disable_clean_inv_dcache_v7_all)
        stmfd   sp!, {r4-r5, r7, r9-r11, lr}
        mrc     p15, 0, r3, c1, c0, 0
        bic     r3, #4                          @ clear C bit
        mcr     p15, 0, r3, c1, c0, 0
        isb

        bl      v7_flush_dcache_all
        mrc     p15, 0, r0, c1, c0, 1
        bic     r0, r0, #0x40                   @ exit SMP
        mcr     p15, 0, r0, c1, c0, 1
        ldmfd   sp!, {r4-r5, r7, r9-r11, pc}
ENDPROC(disable_clean_inv_dcache_v7_all)
```

We Are Not Done, Yet: Cache-to-Cache migration

- When the suspend finisher is called L1 is still allocating
- accessing current implies accessing the sp
- Snooping Direct Data Intervention (DDI), CPU might pull dirty line in

```
ENTRY(disable_clean_inv_dcache_v7_all)
        stmfd   sp!, {r4-r5, r7, r9-r11, lr}
        mrc     p15, 0, r3, c1, c0, 0
        bic     r3, #4                          @ clear C bit
        mcr     p15, 0, r3, c1, c0, 0
        isb

        bl      v7_flush_dcache_all
        mrc     p15, 0, r0, c1, c0, 1
        bic     r0, r0, #0x40                   @ exit SMP
        mcr     p15, 0, r0, c1, c0, 1
        ldmfd   sp!, {r4-r5, r7, r9-r11, pc}
ENDPROC(disable_clean_inv_dcache_v7_all)
```

L2 Management: The Odd One Out (1/2)

- L310 memory mapped device (aka outer cache)
- Clearing C bit does NOT prevent allocation
- L2 RAM retention, data sitting in L2, not accessible if MMU is off
- If not invalidated, L2 might contain stale data if resume code runs with L2 off before enabling it
- We could clean some specific bits: which ones ?
- If retained, L2 must be resumed before turning MMU on

L2 Management: The Odd One Out (1/2)

- L310 memory mapped device (aka outer cache)
- Clearing C bit does NOT prevent allocation
- L2 RAM retention, data sitting in L2, not accessible if MMU is off
- If not invalidated, L2 might contain stale data if resume code runs with L2 off before enabling it
- We could clean some specific bits: which ones ?
- If retained, L2 must be resumed before turning MMU on

L2 Management: The Odd One Out (2/2)

- if L2 content is lost, it must be cleaned on shutdown but can be resumed in C
- if L2 is retained, it must be resumed in assembly before calling cpu_resume

```
static void __init pl310_save(void)
{
        u32 l2x0_revision = readl_relaxed(l2x0_base + L2X0_CACHE_ID) &
                L2X0_CACHE_ID_RTL_MASK;

        l2x0_saved_regs.tag_latency = readl_relaxed(l2x0_base +
                L2X0_TAG_LATENCY_CTRL);
        l2x0_saved_regs.data_latency = readl_relaxed(l2x0_base +
                L2X0_DATA_LATENCY_CTRL);
        [...]
}

//asm-offsets.c
DEFINE(L2X0_R_PHY_BASE, offsetof(struct l2x0_regs, phy_base));
DEFINE(L2X0_R_AUX_CTRL, offsetof(struct l2x0_regs, aux_ctrl));
[...]
```

The Missing Bit: Security Management

- L2 management in non-secure mode is fragmented
- No standard, no way to have a unified solution
- Something we should focus on
- Implications (limitations) on hotplug and CPU logical numbering

Putting Everything Together (1/2)

Common Idle Entry

```
void enter_idle(unsigned cstate, unsigned rstate, unsigned flags)
{
        [...]
        __cpu_set(cpu_index, cpuidle_mask);

        if (cpumask_weight(cpuidle_mask) == num_online_cpus())
                cluster->power_state = rstate;

        cpu_pm_enter();

        if (cluster->power_state >= SHUTDOWN)
                 cpu_cluster_pm_enter();

        cpu_suspend(0, suspend_finisher);

        cpu_pm_exit();

        if (cluster->power_state >= SHUTDOWN)
                cpu_cluster_pm_exit();

        __cpu_clear(cpu_index, cpu_idle_mask);

        return 0;
}
```

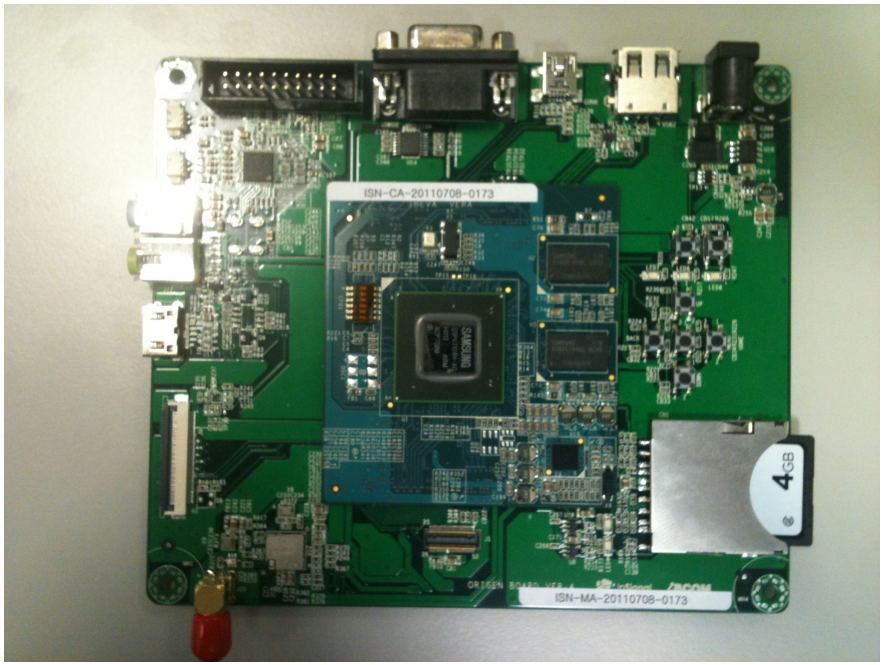
▸ A9 shutdown ▸ next

Putting Everything Together (2/2)

```
void cpu_hotplug(void)
{
        [...]
        disable_clean_inv_dcache_all();

        if (cpu->power_state >= SHUTDOWN)
                        scu_power_mode(cpu_index, SCU_PM_POWEROFF);

        cpu_do_idle();
}

int suspend_finisher(unsigned int long)
{
        [...]
        if (cluster->cluster_down && cluster->power_state == SHUTDOWN)
                outer_flush_all();

        cpu_hotplug();

        return 1;
}
```

Outline

Origen Test Case



- Standard dual-Core A9 system
- Deep idle C-states hit if one cpu is hotplugged
- Lots of duplicated code (L2, GIC)
- Lots of cruft removed @ Linaro Connect, took us two hours

Outline

CPU idle and CPU hotplug

- ARM CPUs on an SMP cluster can be powered down independently
- Most platforms require hotplug for logical cpu id !=0 before enabling deep C-states
- Mechanism created for a purpose and used for the wrong one (high latencies)

CPU idle + sched_mc

- let the scheduler migrate threads in a power efficient manner
- ... and idle does the rest

Conclusion

- Single CPU uncoordinated shutdown becoming more and more important as the number of cores grows
- idle + sched_mc long term solution to manage core idleness
- PM notifiers and CPU/L2 suspend/resume code provides the basis for common PM for all ARM platforms
- Lots of platform code getting cleaned up and consolidated

- Outlook
  - Cluster support
  - Security management

THANKS !!!