CPUidle: One Driver to Rule Them All

D.Lezcano, L.Pieralisi

LCA 2013

CPUidle Common Driver: Goal

- The number of SoCs is increasing, so is their complexity
- Vendors started to implement fine grained power gating capabilities (per core)
- CPUidle framework is gaining momentum
- Unfortunately, each vendor is focused on their own solution without taking care of:
  - Maintainability
  - Code reuse
  - Future improvements
  - Consistency across the cpuidle drivers

CPUidle: Current issues

- copy/paste is commonplace to implement a driver from scratch
- Result is a lot of duplicate code (which may duplicate bugs) eg. ux500 vs imx6 cpuidle driver (mostly the same)

CPUidle: UP platforms

WFI = Wait for interrupt (Clock gating)

OMAP3	7 idle states very complex driver using an extra array with data
AT91	2 idle states WFI WFI + DDR self refresh
Kirkwood	2 idle states WFI WFI + DDR self refresh
SH mobile	1 state WFI defines a custom function but actually it reimplements cpu_do_idle()

- For UP systems, apart from OMAP3, all other drivers are very similar
- A default idle driver
  - state1 => WFI
  - state2 => DDR self refresh ops

CPU idle: SMP platforms (1/2)

WFI = Wait for interrupt (Clock gating)

Calxeda	2 idle states WFI power gating (PSCI based)
Tegra2	1 idle state WFI
Tegra3	2 idle states WFI Power gating
IMX5	1 state WFI custom but described as "CPU state retained/ Powered off"
IMX6	2 states Last man standing copy'n'pasted from u8500 WFI Clock Off

CPUidle: SMP platforms (2/2)

WFI = Wait for interrupt (Clock gating)

Exynos	2 states (CPU hotplug) WFI CPU power down
u8500	2 states WFI CPU state retained
OMAP4	3 states WFI custom CPU off/MPU on CPU off/MPU off using coupled C-states using an extra array

CPUidle status: maintainership

- drivers are under the SoC vendor maintainer umbrella
  - not a single entry point
  - reviews carried out by different people
  - different origin from the merge tree

CPUidle status: internal

- The cpuidle drivers tend to manage the peripherals power
  - Checks if the specific power domain could be powered down
  - Override the governor's decision
  - Adds more complexity
  - Introduces more dependencies

CPUidle Common Driver: guidelines

- pm_runtime / generic power domain
- cpuidle attach to genpd
- device tree representation

CPUidle Common Driver: pm_runtime / genpd

- use pm_runtime for the peripherals
- use the generic power domain for the peripherals
- if something is missing in pm_runtime or genpd : improve them, do not hack the driver

```
static struct rmobile_pm_domain sh7372_pm_domains[] = {
    ...
     {
            .genpd.name = "A4S",
    ...
};
```

CPUidle Common Driver: cpuidle attach

- Attach the generic power domain to a cpuidle state

```
int pm_genpd_attach_cpuidle(struct generic_pm_domain *genpd, int state);
int pm_genpd_name_attach_cpuidle(const char *name, int state);
```

Listing 1: example shmobile

```
pm_genpd_name_attach_cpuidle("A4S", 4);
```

CPUidle Common Driver: cpuidle attach

- Domain poweron/poweroff will disable the cpuidle state

```
int pm_genpd_poweron(struct generic_pm_domain *genpd)
{
    ...
    genpd->cpu_data->idle_state->disabled = true;
    ...
}

int pm_genpd_poweroff(struct generic_pm_domain *genpd)
{
    ...
    genpd->cpu_data->idle_state->disabled = false;
    ...
}
```

CPUidle Common Driver: cpuidle attach

- The cpuidle governor won't choose the disabled state

```
int menu_select(struct cpuidle_driver *drv, struct cpuidle_device *dev)
{
   ...
   for (i = CPUIDLE_DRIVER_STATE_START; i < drv->state_count; i++) {
      struct cpuidle_state *s = &drv->states[i];
      struct cpuidle_state_usage *su = &dev->states_usage[i];

      if (s->disabled || su->disable)
            continue;

      ...
}
```

CPUidle Common Driver: dts

- An array of callbacks for all the arch specific idle functions
- A cpuidle driver structure initialized with the dts information

CPUidle Common Driver: dts

```
cpuidle {
        idlestates {
                idlestate0: idlestate@0 {
                        enter = "simple";
                        exit_latency = <1>;
                        target_residency = <1>;
                        power_usage = <0xffffffff>;
                        flags = <0x1>;
                        state_name = "WFI";
                        desc = "ARM WFI";
                };
        };
};
```

Conclusion

- CPUidle drivers need consolidation
- Use pm_runtime for the peripherals
- Use the generic power domain with cpuidle
- Common code must be factorized
- DT to the rescue for idle states data

- Outlook
  - DT states bindings
  - Starting by creating a single driver for existing drivers
  - Integrate ARM PSCI proposal

THANKS !!!

ARM Common PM Code Components

- CPU PM notifiers
- Local timers save/restore
- CPU suspend/resume

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,
};
```

Local timers save/restore

```
void enter_idle(...)
{
        [...]
        clockevents_notify(CLOCK_EVT_NOTIFY_BROADCAST_ENTER, &cpu);
        [...]
        cpu_do_idle();
        [...]
        clockevents_notify(CLOCK_EVT_NOTIFY_BROADCAST_EXIT, &cpu);
        [...]
}

void enter_idle(...)
{
        struct tick_device *tdev = tick_get_device(cpu);
        [...]
        cpu_do_idle();
        [...]
        /* Restore the per-cpu timer event */
        clockevents_program_event(tdev->evtdev, tdev->evtdev->next_event, 1);
}
```

- Enter broadcast mode if a global timer is available
- Rely on always-on firmware timer and restore timer through clock events programming API

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)
{
        u32 *ctx = 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)
{
        u32 *ctx = 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_louis();
}
```

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)
{
        u32 *ctx = 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_louis();

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

Security Management

- Most of the operations should be carried out in secure world
- non-secure cache-line clean/invalidate can be deferred
- Policy decisions made in Linux
- ARM PSCI implementation in the making

Putting Everything Together (1/2)

CPU idle skeleton state enter function

```
struct pm_pms {
        unsigned int cluster_state;
        unsigned int cpu_state;
};

void enter_idle(unsigned int cluster_state, unsigned int cpu_state)
{
        int cpu = smp_processor_id();
        struct pm_pms pms;
        struct cpumask tmp;
        [...]

        cpu_set(cpu, cpuidle_mask);
        cpumask_and(&tmp, &cpuidle_mask, topology_core_cpumask(cpu));

        pms.cluster_state = cluster_state;
        pms.cpu_state = cpu_state;

        if (!cpu_weight(&tmp) == cpu_weight(topology_core_cpumask(cpu)))
                pms.cluster_state = 0;

        cpu_pm_enter();
        if (pms.cluster_state >= SHUTDOWN)
                cpu_cluster_pm_enter();

        cpu_suspend(&pms, suspend_finisher);

        cpu_pm_exit();

        if (pms.power_state >= SHUTDOWN)
                cpu_cluster_pm_exit();
        cpu_clear(cpu, cpu_idle_mask);
        return 0;
}
```

v7 shutdown next

Putting Everything Together (2/2)

```
int suspend_finisher(unsigned long arg)
{
        struct pm_pms *pp = (struct pm_pms *) arg;
        [...]
        smc_down(...);
        return 1;
}

smc_down:
        ldr r0, =#SMC_NUMBER
        smc #0
        /*
         * Pseudo code describing what secure world
         * should do
         */
        {
        disable_clean_inv_dcache_louis();
        if (cluster->cluster_down && cluster->power_state == SHUTDOWN) {
                flush_cache_all();
                outer_flush_all();
        }
        normal_memory_lock();
        disable_cci_snoops();
        normal_memory_unlock();
        power_down_command();
        cpu_do_idle();
        }
```