Linux Kernel Power Management (PM) Framework for ARM 64-bit Processors

L.Pieralisi

21/8/2014 - LinuxCon North America 2014

Outline

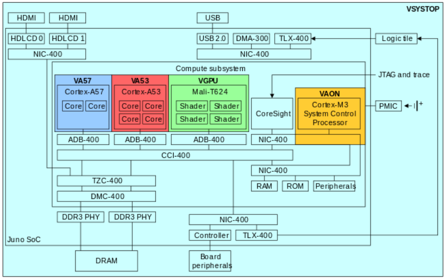
- ARM 32-bit Linux kernel power management support for huge legacy of ARM processors
  - from v4 Uniprocessor kernels to ARM v7 SMP multicluster systems
- Lack of established firmware interfaces is preventing merge of power management software in the mainline kernel
  - Lots of tricky platform specific code, maintained as out of tree BSP branches
  - Power management HW dependent SW layers incompatible with most of kernel SW interfaces
- ARM64 kernel port represents an opportunity to start afresh and learn lessons from the past

Legacy ARM 32-bit Linux Kernel Power Management

- Standards ? No, thanks
- ARM vendors power controllers
  - Differentiating through design quirks
- Standard power down procedures started appearing on ARM v7 TRMs
  - What works on ARM v7, might not on v6, and viceversa
  - ARM v7 power down procedure is standard, except for when it is not
  - ....and then came SMP systems....
- Kernel developers left to their own devices
  - Reverse engineering (lack of publicly available specs)
  - Power management interfaces designed for UP systems did not work on SMP
  - Secure/non-secure split overriden in most upstream design
  - Kernel subsystems (eg CPUidle) redesigned to support HW bugs (eg couple idle states)

ARM Chips Power Management Configurations

- Multiple power islands
  - Devices and CPUs power domains
- Core gating and Cluster gating
  - RAM retention states
  - Caches management
  - Coherency management
  - Locking
- Graphics power domain
  - Integration with CPU states
- System sleep states
- Always-on power domain



ARM Power Management HW/SW stack (1/2)

- Cross-divisional collaborative effort between ARM SW/HW engineers
- HW prototyping effort (TC2/Juno)
  - Power controller specifications
  - SW stack (Linux kernel/Trusted Firmware/power controller firmware)
- Tackle ARM vendors power control fragmentation
  - Power management HW still a prominent feature of vendors design
  - Design quirks are the rule, not the exception
- Ongoing effort at ARM to standardize main HW control feaures
  - HW SMP coherency management
  - Cache clean and invalidate (SW/HW)
  - Wake-up capabilities and GIC design

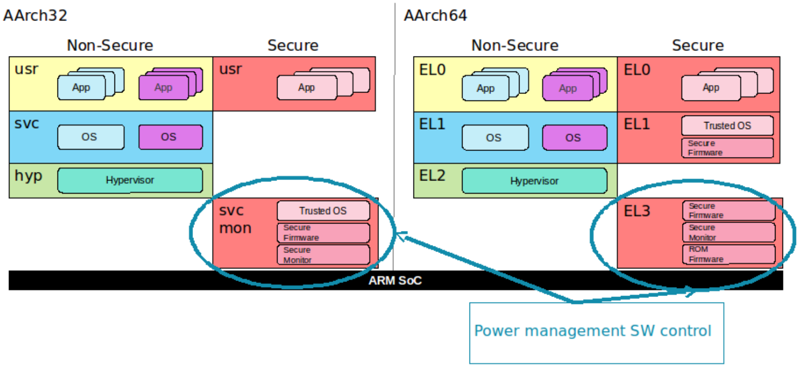
ARM Power Management HW/SW stack (2/2)

- Development and deployment of PSCI (Power State Coordination Interface) firmware interface
- Device tree and ACPI bindings standardization
  - Provide OS with standard interfaces
  - Configuration data for power management subsystems (CPU/device idle states)
  - Run-time services (CPU/devices power management)

Our Goal

Provide a HW/SW environment to foster power management standardization

ARM Power Management: Secure Operations



- Power management SW has to deal with secure operations
  - Coherency management, power control commands
  - Need for a standard API to communicate to secure world

ARM Power Management: Secure Operations

"... I guess what this ultimately comes down to is that we _did_ accept the work-arounds into the kernel source for secure parts - had we not done that and set a clear message like ARM64 does that work-arounds must be done prior to calling the kernel (irrespective of how that happens, iow, whether boot or resume) then we wouldn't have this problem right now. Such a statement would have raised lots of objections though, but with hindsight, it would have been the right thing to do to overrule those objections and just mandate it. It would've been a pain for some people, but we would not be in this situation now where there is no proper solution which works for everyone."

RMK

http://lists.infradead.org/pipermail/linux-arm-kernel/2014-July/268718.html

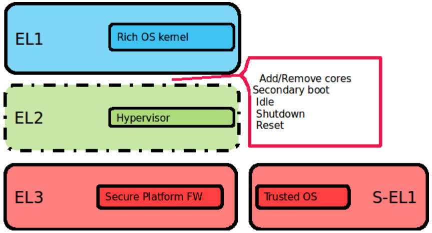
ARM Power Management: Early Devices

```
int __init ve_spc_init(void __iomem *baseaddr,
                       u32 a15_clusid, int irq)
{
        int ret;
        info = kzalloc(sizeof(*info), GFP_KERNEL);
        if (!info) {
                return -ENOMEM;
        }

        info->baseaddr = baseaddr;
        info->a15_clusid = a15_clusid;

        [...]

        sync_cache_w(info);
        sync_cache_w(&info);

        return 0;
}
```

```
void ve_spc_set_resume_addr(u32 cluster, u32 cpu,
                            u32 addr)
{
        void __iomem *baseaddr;

        if (cluster >= MAX_CLUSTERS)
                return;

        if (cluster_is_a15(cluster))
                baseaddr = info->baseaddr +
                           A15_BX_ADDR0 + (cpu << 2);
        else
                baseaddr = info->baseaddr +
                           A7_BX_ADDR0 + (cpu << 2);

        writel_relaxed(addr, baseaddr);
}
```

- Need to probe device drivers early (before `early_initcalls`)
  - Power controllers drivers
  - CPU memory mapped peripherals (no access through coprocessor - need DT probing)
- Resulting code incompatible with kernel driver model
  - Code exiled to arch/arm, because it does not belong anywhere else

Power State Coordination Interface



- Defines a standard interface for making power management requests across exception levels/operating systems
- Support virtualisation and a communication channel between normal and secure world
- Allow secure firmware to arbitrate power management requests from secure and non-secure software
- Default method for power control in Linux ARM64 kernel
  - CPU on/off (hotplug, cold-boot)
  - CPU suspend (suspend to RAM, CPUidle)
- Spec available at `http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.den0022b/index.html`

ARM

ARM64 Linux Power Management Requirements

- Reliance on PSCI as firmware interface
- Trusted firmware PSCI implementation publicly available for FVP models and Juno
  https://github.com/ARM-software/arm-trusted-firmware
- Device tree and ACPI bindings to standardize control data and improve code generality
  - Take part in ACPI standardization effort http://www.acpi.info
  - Actively push for Device Tree bindings for power management SW components
- Push back on design quirks and related SW hacks
- More collaboration between ARM and ARM partners on HW/SW interfaces
  - HW standardization effort must be deployed in ARM vendors designs
  - SW interfaces adopted by ARM partners Linux kernel developers

Linux Kernel Power Management Subsystems

- CPUidle: framework managing idle threads
  - CPUs enter idle states when idling, with power depth levels that depend on power management HW
- CPU hotplug: removing a CPU from a running system
  - Not designed for power management, abused as such
- Runtime PM
  - Devices power management
- Suspend to RAM: system wide (CPU and devices) sleep states
  - Triggered from userspace, also used as autosuspend when system is idle (eg on Android)
- Suspend to disk (aka Hibernation): same as Suspend-to-RAM, system image saved to disk instead of volatile memory

Linux Kernel PM example: CPUidle core

- CPU is running idle
  - Next event (in time) determines how long the CPU can sleep
  - CPU woken up by IRQs, behaves as WFI regardless of the idle state power depth
- Idle state entered through CPUidle driver enter() function
  - Prepare the CPU for power down
  - Execute the power down command

```
CPU idle state data

struct cpuidle_state {
        char name[CPUIDLE_NAME_LEN];
        char desc[CPUIDLE_DESC_LEN];
        unsigned int flags;
        unsigned int exit_latency; /* in US */
        int power_usage; /* in mW */
        unsigned int target_residency; /* in US */
        bool disabled; /* disabled on all CPUs */
        /* Idle state enter function */
        {int (*enter)(struct cpuidle_device *dev,
                      struct cpuidle_driver *drv,
                      int index);
        ...
};
```

ARM 32-bit CPUidle consolidation issues

- Current ARM CPUidle drivers in the kernel cannot be easily cleaned-up/consolidated
- ARM 32-bit SMP CPUidle code was merged separately through different trees, with no consolidation effort
- `cpu_suspend()` kernel interface was defined, but there are still open issues that prevent consolidation
  - Lack of standard power down procedures
  - Lack of standard idle state entry interface
- Multi-Cluster Power Management (MCPM) introduction helped define a common entry point for idle
  - It requires the kernel to run in secure world unless security is overriden in firmware
  - It relies on early devices initialization in the kernel

ARM 32-bit CPUidle consolidation issues

- Current ARM CPUidle drivers in the kernel cannot be easily cleaned-up/consolidated
- ARM 32-bit SMP CPUidle code was merged separately through different trees, with no consolidation effort
- `cpu_suspend()` kernel interface was defined, but there are still open issues that prevent consolidation
  - Lack of standard power down procedures
  - Lack of standard idle state entry interface
- Multi-Cluster Power Management (MCPM) introduction helped define a common entry point for idle
  - It requires the kernel to run in secure world unless security is overriden in firmware
  - It relies on early devices initialization in the kernel

ARM 32-bit v7 shutdown procedure (1/3)

```
ENTRY(tegra_disable_clean_inv_dcache)
        stmfd   sp!, {r0, r4-r5, r7, r9-r11, lr}
        dmb                                     @ ensure ordering

        /* Disable the D-cache */
        mrc     p15, 0, r2, c1, c0, 0
        bic     r2, r2, #CR_C
        mcr     p15, 0, r2, c1, c0, 0
        isb

        /* Flush the D-cache */
        cmp     r0, #TEGRA_FLUSH_CACHE_ALL
        blne    v7_flush_dcache_louis
        bleq    v7_flush_dcache_all

        /* Turn off coherency */
        exit_smp r4, r5

        ldmfd   sp!, {r0, r4-r5, r7, r9-r11, pc}
ENDPROC(tegra_disable_clean_inv_dcache)
```

https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/arch/arm/mach-tegra/sleep.S?id=refs/tags/v3.16

ARM 32-bit v7 shutdown procedure (2/3)

```
ENTRY(omap4_finish_suspend)
        stmfd   sp!, {r4-r12, lr}
        cmp     r0, #0x0
        beq     do_WFI                          @ No lowpower state, jump to WFI
        [...]
skip_secure_l1_clean:
        bl      v7_flush_dcache_all

        mrc     p15, 0, r0, c1, c0, 0
        bic     r0, r0, #(1 << 2)               @Disable the C bit
        mcr     p15, 0, r0, c1, c0, 0
        isb

        bl      v7_flush_dcache_all

        bl      omap4_get_sar_ram_base
        mov     r8, r0
        ldr     r9, [r8, #OMAP_TYPE_OFFSET]
        cmp     r9, #0x1                        @ Check for HS device
        bne     scu_gp_set
        mrc     p15, 0, r0, c0, c0, 5           @ Read MPIDR
        ands    r0, r0, #0x0f
        ldreq   r0, [r8, #SCU_OFFSET0]
        ldrne   r0, [r8, #SCU_OFFSET1]
        mov     r1, #0x00
        stmfd   r13!, {r4-r12, r14}
        ldr     r12, =OMAP4_MON_SCU_PWR_INDEX
        DO_SMC
        [...]
ENDPROC(omap4_finish_suspend)
```

https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/arch/arm/mach-omap2/sleep44xx.S?id=refs/tags/v3.16

ARM 32-bit v7 shutdown procedure (3/3)

```
#define v7_exit_coherency_flush(level) \
        asm volatile( \
        "stmfd       sp!, {fp, ip} \n\t" \
        "mrc        p15, 0, r0, c1, c0, 0        @ get SCTLR \n\t" \
        "bic        r0, r0, #"__stringify(CR_C)" \n\t" \
        "mcr        p15, 0, r0, c1, c0, 0        @ set SCTLR \n\t" \
        "isb        \n\t" \
        "bl        v7_flush_dcache_"__stringify(level)" \n\t" \
        "clrex       \n\t" \
        "mrc        p15, 0, r0, c1, c0, 1        @ get ACTLR \n\t" \
        "bic        r0, r0, #(1 << 6)        @ disable local coherency \n\t" \
        "mcr        p15, 0, r0, c1, c0, 1        @ set ACTLR \n\t" \
        "isb        \n\t" \
        "dsb        \n\t" \
        "ldmfd       sp!, {fp, ip}" \
        : : : "r0","r1","r2","r3","r4","r5","r6","r7", \
              "r9","r10","lr","memory" )
```

https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/arch/arm/include/asm/cacheflush.h?id=refs/tags/v3.16

ARM 32-bit CPUidle consolidation issues

- Current ARM CPUidle drivers in the kernel cannot be easily cleaned-up/consolidated
- ARM SMP CPUidle code merged separately through different trees, with no consolidation effort
- `cpu_suspend()` kernel interface was defined, but there are still open issues that prevent consolidation
    - Lack of standard power down procedures
    - Lack of standard idle state entry interface
- Multi-Cluster Power Management (MCPM) introduction helped define a common entry point for idle
    - It requires the kernel to run in secure world unless security is overriden in firmware
    - It relies on early devices initialization in the kernel

ARM 32-bit CPUidle enter examples (1/3)

```
static int armada_370_xp_enter_idle(struct cpuidle_device *dev,
                                    struct cpuidle_driver *drv,
                                    int index)
{
        int ret;
        bool deepidle = false;
        cpu_pm_enter();

        if (drv->states[index].flags & ARMADA_370_XP_FLAG_DEEP_IDLE)
                deepidle = true;

        ret = armada_370_xp_cpu_suspend(deepidle);
        if (ret)
                return ret;

        cpu_pm_exit();

        return index;
}
```

https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/drivers/cpuidle/cpuidle-armada-370-xp.c?id=refs/tags/v3.16

ARM 32-bit CPUidle enter examples (2/3)

```
static int idle_finisher(unsigned long flags)
{
        exynos_enter_aftr();
        cpu_do_idle();

        return 1;
}

static int exynos_enter_core0_aftr(struct cpuidle_device *dev,
                                   struct cpuidle_driver *drv,
                                   int index)
{
        cpu_pm_enter();
        cpu_suspend(0, idle_finisher);
        cpu_pm_exit();

        return index;
}

static int exynos_enter_lowpower(struct cpuidle_device *dev,
                                 struct cpuidle_driver *drv,
                                 int index)
{
        int new_index = index;

        if (num_online_cpus() > 1 || dev->cpu != 0)
                new_index = drv->safe_state_index;

        if (new_index == 0)
                return arm_cpuidle_simple_enter(dev, drv, new_index);
        else
                return exynos_enter_core0_aftr(dev, drv, new_index);
}
```

https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/drivers/cpuidle/cpuidle-exynos.c?id=refs/tags/v3.16

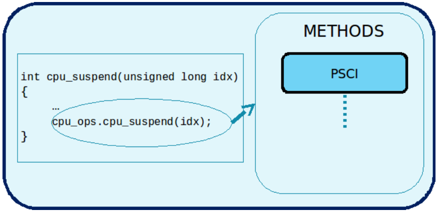
ARM 32-bit CPUidle enter examples (3/3)

```
static int notrace bl_powerdown_finisher(unsigned long arg)
{
        /* MCPM works with HW CPU identifiers */
        unsigned int mpidr = read_cpuid_mpidr();
        unsigned int cluster = MPIDR_AFFINITY_LEVEL(mpidr, 1);
        unsigned int cpu = MPIDR_AFFINITY_LEVEL(mpidr, 0);

        mcpm_set_entry_vector(cpu, cluster, cpu_resume);
        [...]

        mcpm_cpu_suspend(0);

        /* return value != 0 means failure */
        return 1;
}

static int bl_enter_powerdown(struct cpuidle_device *dev,
                              struct cpuidle_driver *drv, int idx)
{
        cpu_pm_enter();

        cpu_suspend(0, bl_powerdown_finisher);

        mcpm_cpu_powered_up();

        cpu_pm_exit();

        return idx;
}
```

https://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/drivers/cpuidle/cpuidle-big_little.c?id=refs/tags/v3.16

Linux Kernel ARM64 CPU operations

- Generic kernel interface for CPU operations
  - boot
  - hotplug
  - idle
- CPU operations represent the interface through which the kernel carries out the required actions on CPUs
  - Interface hides the actual method implementation
  - Hooks initialized at boot through DT or ACPI



```
int cpu_suspend(unsigned long idx)
{
    …
    cpu_ops.cpu_suspend(idx);
}
```

```
struct cpu_operations {
        const char *name;
        int (*cpu_init)(struct device_node *,
                        unsigned int);
        int (*cpu_init_idle)(struct device_node *,
                             unsigned int);
        int (*cpu_prepare)(unsigned int);
        int (*cpu_boot)(unsigned int);
        void (*cpu_postboot)(void);
        int (*cpu_disable)(unsigned int cpu);
        void (*cpu_die)(unsigned int cpu);
        int (*cpu_suspend)(unsigned long);
};
```

ARM64 CPUidle enter

```
static int arm_enter_idle_state(struct cpuidle_device *dev,
                                struct cpuidle_driver *drv, int idx)
{
        int ret;

        if (!idx) {
                cpu_do_idle();
                return idx;
        }

        ret = cpu_pm_enter();
        if (!ret) {
                /*
                 * Pass idle state index to cpu_suspend which in turn will
                 * call the CPU ops suspend protocol with idle index as a
                 * parameter.
                 */
                ret = cpu_suspend(idx);

                cpu_pm_exit();
        }

        return ret ? -1 : idx;
}
```

http://lists.infradead.org/pipermail/linux-arm-kernel/2014-July/274249.html

ARM64 CPU Operations: PSCI suspend

```
static int __maybe_unused cpu_psci_cpu_suspend(unsigned long index)
{
        struct psci_power_state *state = __get_cpu_var(psci_power_state);
        /*
         * idle state index 0 corresponds to wfi, should never be called
         * from the cpu_suspend operations
         */
        if (WARN_ON_ONCE(!index))
                return -EINVAL;

        return psci_ops.cpu_suspend(state[index - 1],
                                    virt_to_phys(cpu_resume));
}

```

http://lists.infradead.org/pipermail/linux-arm-kernel/2014-July/274245.html

- Suspend operations become a stub that calls into firmware to carry out power down
  - Equivalent of `mwait` in x86 world

Idle States Device Tree Bindings

```
idle-states {
        entry-method = "arm,psci";
        CPU_SLEEP_0: cpu-sleep-0 {
                compatible = "arm,idle-state";
                arm,psci-suspend-param = <0x0010000>;
                entry-latency-us = <40>;
                exit-latency-us = <100>;
                min-residency-us = <150>;
        };
        CLUSTER_SLEEP_0: cluster-sleep-0 {
                compatible = "arm,idle-state";
                arm,psci-suspend-param = <0x1010000>;
                entry-latency-us = <500>;
                exit-latency-us = <1000>;
                min-residency-us = <2500>;
        };
};
```

http://lists.infradead.org/pipermail/linux-arm-kernel/2014-July/274251.html

- Configuration data provided by firmware through well established bindings

ARM64 Kernel PM: What's to come

- Augment DT idle states bindings with power domains information
  - Link CPU components (inclusive of caches) and devices to their respective power domains
  - Add power domains information to DT idle states
- Implement ACPI ARM PM drivers for ACPI 5.1
- Spec out ARM system sleep states
  - PSCI system sleep states management
  - ACPI and DT system sleep states bindings

Conclusion

- ARM 32-bit Linux kernel huge legacy code
    - HW power control design quirks and related SW workarounds
    - Lack of power management standards in both HW and SW
    - Lots of out-of-tree power management ARM vendors code
- ARM 64-bit power management
    - No legacy
    - ARM 32-bit SW design experience
    - Upstream momentum for standard interfaces and systems
- Standardization required in both HW and SW to prevent same ARM 32-bit mistakes
    - HW power management recommendation to be shared with ARM vendors
    - Foster PSCI standardization deployment
    - Actively contribute to ACPI specifications and device tree bindings

THANKS !!!