DT power domain bindings - idle states

Lorenzo Pieralisi <lorenzo.pieralisi@arm.com>
Sudeep Holla <sudeep.holla@arm.com>

20/8/15 – Energy Aware Scheduling II LPC microconference
Characterize idle states through affected power domains on idle state entry
Define what devices (inclusive of CPU components like PMU) are affected by idle state entry
Improve CPU PM notifiers mechanism
  - Current implementation does not allow fine grain notification control
  - Simple notification chain, notifier behaviour independent of idle state
  - Diversify retention versus shutdown states
Idle States Device Tree Bindings

```plaintext
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>;
};
```


- Configuration data provided by DT through well established bindings
Idle States: Generic Power Domains (1/4)

```
static inline int of_genpd_add_provider_simple(struct device_node *np, 
    struct generic_pm_domain *genpd)

static inline int of_genpd_add_provider_onecell(struct device_node *np, 
    struct genpd_onecell_data *data)

/**
 * genpd_dev_pm_attach - Attach a device to its PM domain using DT.
 * @dev: Device to attach.
 *
 * Returns 0 on successfully attached PM domain or negative error code.
 */
int genpd_dev_pm_attach(struct device *dev)
{
    [...]  
    ret = of_parse_phandle_with_args(dev->of_node, "power-domains", 
        
    
    "#power-domain-cells", 0, &pd_args); 
}
```

power/power_domain.txt?id=refs/tags/v4.2-rc6
IDLE STATES: GENERIC POWER DOMAINS (2/4)

```
pd_cores: power-domain-cores@80000000 {
    #address-cells = <2>;
    #size-cells = <2>;
    compatible = "arm,power-controller";
    reg = <0x0 0x80000000 0x0 0x1000>;
    #power-domain-cells = <1>;
};

idle-states {
    [...]
    CPU_SLEEP_0: cpu-sleep-0 {
        compatible = "arm,idle-state";
        arm,psci-suspend-param = <0x1010000>;
        [...]  
        power-domains = <&pd_cores 0>;
    };  
};

CPU0: cpu@0 {
    device_type = "cpu";
    reg = <0x0 0x0>;
    enable-method = "psci";
    next-level-cache = <&L1_0>;
    cpu-idle-states = <&CPU_SLEEP_0>;
    L1_0: l1-cache {
        compatible = "arm,arch-cache";
        power-domains = <&pd_cores 0>;
    };  
};
```
Power domains can be attached to specific idle states
Devices (inclusive of caches and CPUs) linked to power domains

Can be generalized to CPU devices (timers, PMUs,...)
  - Timers, caches and interrupt controllers are not backed by a struct device though....

May be used to manage the infamous CPUIDLE_FLAG_TIMER_STOP
  - Use idle state power domain information to detect local timer shutdown

Coresight™ TC2 example on the LAKML
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
```c
int cpu_pm_enter(void)
{
    int nr_calls;
    int ret = 0;

    read_lock(&cpu_pm_notifier_lock);
    ret = cpu_pm_notify(CPU_PM_ENTER, -1, &nr_calls);
    if (ret)
    {
        /*
         * Inform listeners (nr_calls - 1) about failure of CPU PM
         * PM entry who are notified earlier to prepare for it.
         */
        cpu_pm_notify(CPU_PM_ENTER_FAILED, nr_calls - 1, NULL);
        read_unlock(&cpu_pm_notifier_lock);
    }
    return ret;
}

int cpu_cluster_pm_enter(void)
{
    int nr_calls;
    int ret = 0;

    read_lock(&cpu_pm_notifier_lock);
    ret = cpu_pm_notify(CPU_CLUSTER_PM_ENTER, -1, &nr_calls);
    if (ret)
    {
        /*
         * Inform listeners (nr_calls - 1) about failure of CPU cluster
         * PM entry who are notified earlier to prepare for it.
         */
        cpu_pm_notify(CPU_CLUSTER_PM_ENTER_FAILED, nr_calls - 1, NULL);
        read_unlock(&cpu_pm_notifier_lock);
    }
    return ret;
}
```
static int gic_notifier(struct notifier_block *self, unsigned long cmd, void *v) {
    int i;

    for (i = 0; i < MAX_GIC_NR; 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,
};

static void __init gic_pm_init(struct gic_chip_data *gic) {
    [...]if (gic == &gic_data[0])
        cpu_pm_register_notifier(&gic_notifier_block);
}
CPU PM notifiers: runtime PM to the rescue

- ARM PM domains for CPUs/Clusters
- Create the kernel infrastructure for seamless CPU and devices power management in the kernel
- Eventually get rid of CPU PM notifiers
Conclusion

- Define DT bindings to link power domains to idle states
  - Determine affected "devices" on idle state entry
  - Implement CPU PM notifiers through genPD
  - Enforce idle state power domain dependencies

- Improve "all or nothing" CPU PM notifiers save/restore behaviour

- Define CPUs power domain topologies through power domain linkage to CPU nodes
Thank You

The trademarks featured in this presentation are registered and/or unregistered trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved. All other marks featured may be trademarks of their respective owners.