TF-A Tech Forum

S2idle-driven Low power Mode selection using PSCI in AM62L

Date: 2026-04-02

Scaria Kochidanadu

Who Am I?

Scaria Kochidanadu

Embedded Software Developer at Texas Instruments

Power Management Expert on TI Sitara SoCs

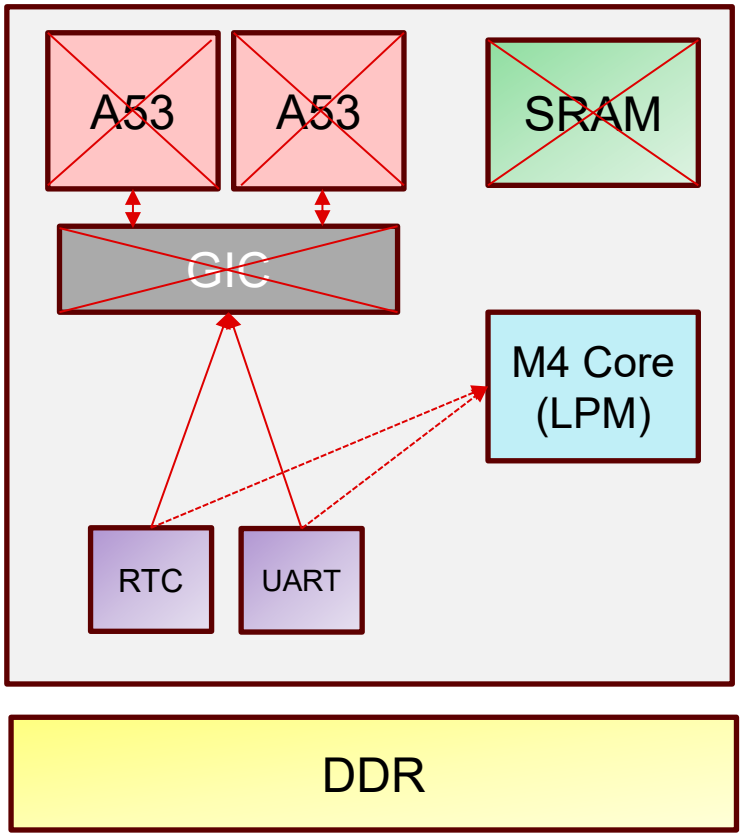
Contributed to Uboot, TFA, Linux

Overview

- AM62L Architecture
- Devicetree: Idle-states and Power domain Mapping
- S2idle Call flow
- PSCI-Implementation
  - Validate power state
  - Core coordination in Suspend sequence
  - Resume sequence
- Challenges faced
  - Re-entry of into idle-state
  - System state context LOST
  - S2idle re-entry on resume
  - Handling interrupt affinity when a core is turned off
- Standby idle-state
  - Validation Logic Problem
  - Our workaround

AM62L Architecture

Deep Sleep



Available Low power modes

System Retention Modes

Standby – Low Latency

Cores: Both ON (WFI)
Few PLLs downclocks, non critical LPSCs turned off
DDR: Auto Self Refresh
Wakeup sources: All interrupts
Fast, low power saving

Standby – High Latency

Cores: One core OFF
DDR: Self Refresh
Multiple PLL and devices off
Wakeup sources: All interrupts

System Power-Down Modes

DeepSleep

Cores: Both OFF
DDR: Self Refresh
Wakeup sources: RTC, USB, UART, IO-Daisy Chain

RTC + DDR

Cores: Both OFF
DDR: Self Refresh
Wakeup sources: RTC
Maximum power saving

Device tree Configuration

- CPU idle states

```
    cpu_sleep_0: stby {
        compatible = "arm,idle-state";
        idle-state-name = "Standby";
        arm,psci-suspend-param = <0x00000001>;
        entry-latency-us = <25>;
        exit-latency-us = <100>;
        min-residency-us = <1000>;
    };

    cpu_sleep_1: powerdown {
        compatible = "arm,idle-state";
        idle-state-name = "PowerDown";
        arm,psci-suspend-param = <0x012233>;
        entry-latency-us = <10000>;
        exit-latency-us = <10000>;
        min-residency-us = <1000000>;
        local-timer-stop;
    };
};
```

Domain idle states

```
main_sleep_deep: main-sleep-deep {
    compatible = "domain-idle-state";
    arm,psci-suspend-param = <0x2012235>;
    entry-latency-us = <250000>;
    exit-latency-us = <100000>;
    min-residency-us = <500000>;
    local-timer-stop;
};

main_sleep_rtcddr: main-sleep-rtcddr {
    compatible = "domain-idle-state";
    arm,psci-suspend-param = <0x2012234>;
    local-timer-stop;
    entry-latency-us = <300000>;
    exit-latency-us = <600000>;
    min-residency-us = <1000000>;
};
```

Power domain Map

```
&psci {
    CPU_PD: power-controller-cpu {
        #power-domain-cells = <0>;
        power-domains = <&CLUSTER_PD>;
        domain-idle-states = <&cpu_sleep_0>, <&cpu_sleep_1>;
    };

    CLUSTER_PD: power-controller-cluster {
        #power-domain-cells = <0>;
        domain-idle-states = <&cluster_sleep_0>;
        power-domains = <&MAIN_PD>;
    };

    MAIN_PD: power-controller-main {
        #power-domain-cells = <0>;
        domain-idle-states = <&main_sleep_deep>, <&main_sleep_rtcddr>;
    };
};
```

```
&scmi_pds {
    power-domain-map = <3 &CLUSTER_PD>, /* CPSW3G */
                       <39 &CLUSTER_PD>, /* DSS0 */
                       <38 &CLUSTER_PD>, /* DSS_DSI0 */
                       <15 &MAIN_PD>, /* TIMER0 */
                       <16 &MAIN_PD>, /* TIMER1 */
                       <17 &MAIN_PD>, /* TIMER2 */
                       <18 &MAIN_PD>, /* TIMER3 */
```

S2idle flow and OSI mode

s2idle Call Flow:

```
Linux Kernel                    PSCI Firmware (TF-A)
============                    ====================

1. Freeze tasks
     |
     v
2. Suspend devices
     |
     v
3. cpuidle framework ----------> CPU_SUSPEND
   (per CPU)                          |
     |                                v
     |                          Coordinate power
     |                          state requests
     |                                |
     |                                v
     |                          CPU enters low-power
     |                          hardware state
     |                                |
     |                                V
     |                          Wakeup event (eg. RTC)
     |<--------- Resume ---------
     |
     v
4. Resume devices
     |
     v
5. Thaw tasks
```

```
                    OSI "Last Man Standing" Flow

Cluster with 2 Cores          OS Action                  PSCI Request
====================          =========                  =============

1. Core 0,1: ACTIVE
         |
         | Core 0 becomes idle
         v
2. Core 0: IDLE           --> OS requests local       --> CPU_SUSPEND
   Core 1: ACTIVE             Core Power Down             (Core PD only)
                              Cluster stays ON
         |
         | Core 1 (LAST) becomes idle
         v
3. Core 0,1: IDLE         --> OS recognizes           --> CPU_SUSPEND
                              "Last Man" scenario         (Composite State)
                              Requests Composite:
                              - Core 1: PD                Core:    PD
                              - Cluster: PD               Cluster: PD
                              - System: PD                System:  PD
         |
         v
4. Firmware Verification  --> PSCI firmware checks
   & System Power Down        all cores/clusters idle
                              If verified: Power down
                              entire system
                              If not: Deny request
                              (race condition)
```

AM62L PSCI Implementation and design

Validate power state

CPUidle-driver

Power state parameter
Example :
0x01000021 - Standby
0x02012235 - DeepSleep

Bit Fields
Bit 0-15 : State ID – Implementation Defined

Bit 16 : Power state 0 = Retention
1 = Power down
Bit 24-25 : Power level 0 = CPU
1 = Cluster
2 = System

Role of the function

state_info population

- state_info[0] = CPU state
- state_info[1] = Cluster state
- state_info[2] = System state

For Retention States :

0x01000021 → validate_power_state() →
state_info[0] = 1 (Core-idle)
state_info[1] = 2 (Standby-1)
state_info[2] = 0 (RUN)

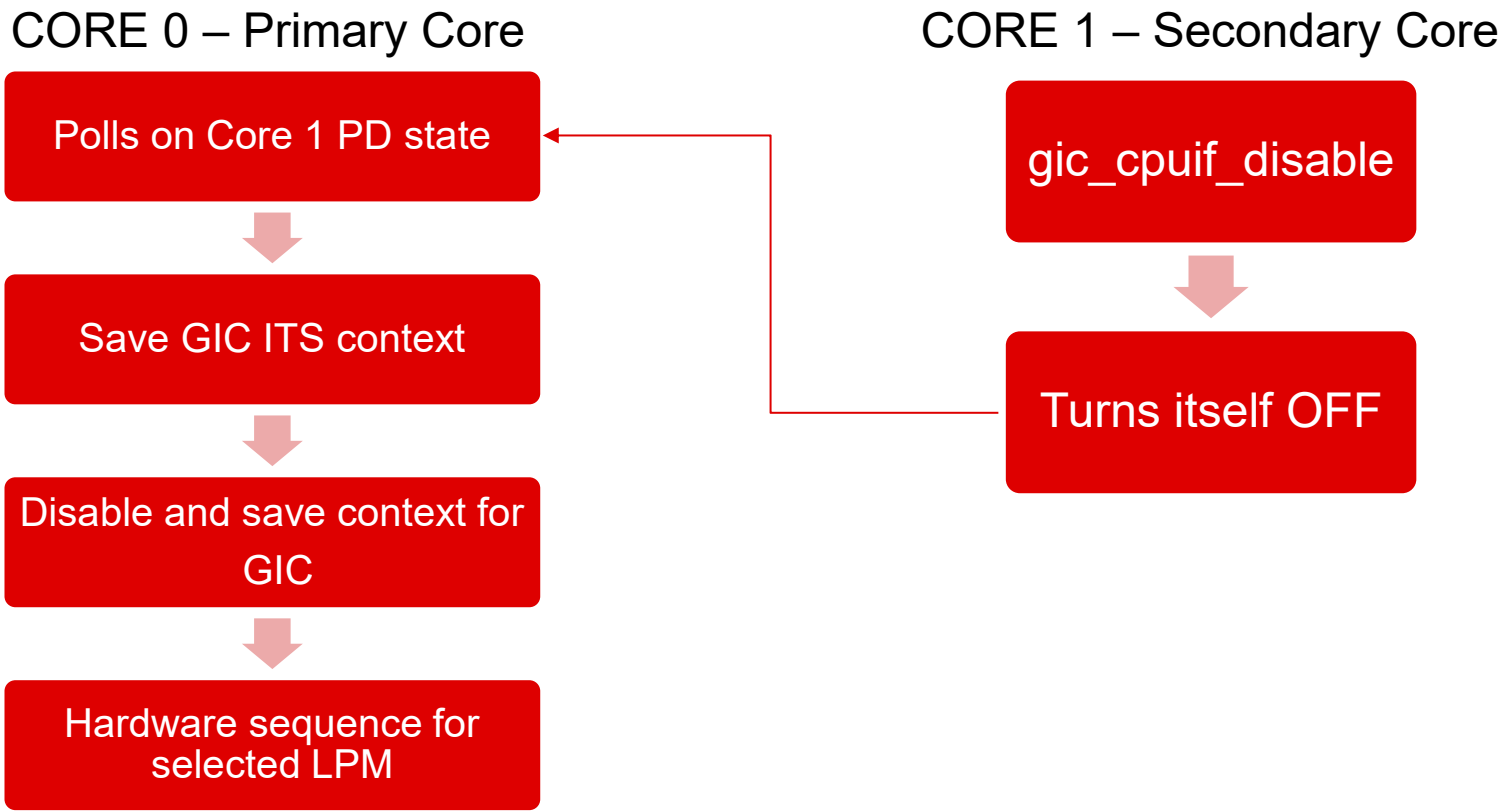
For Power down States :

0x02012234 → validate_power_state() →
state_info[0] = 4
state_info[1] = 4
state_info[2] = 4

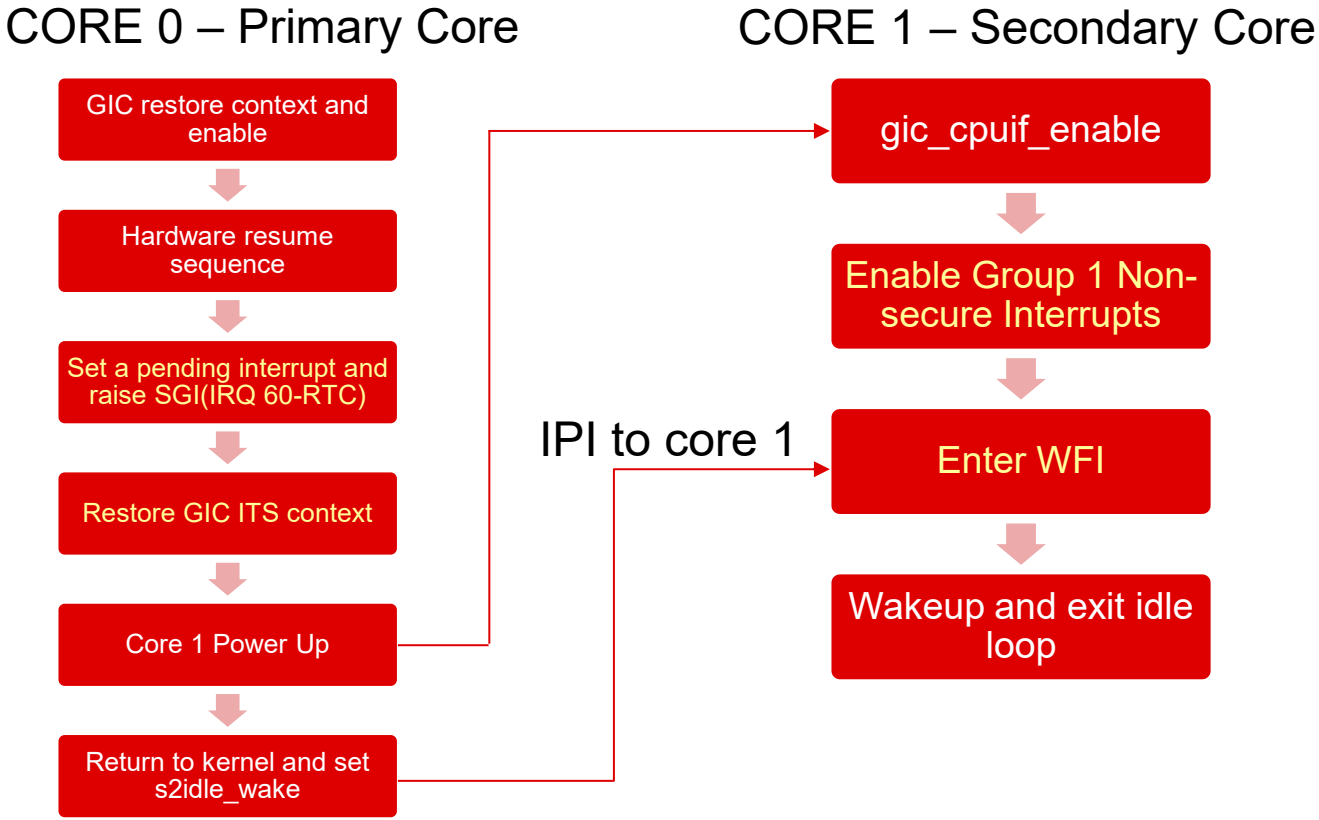
4 – MAX_OFF_STATE

→ Mode selection Logic comes here:
LPM - Deepsleep for parameter 0x02012235
LPM - RTC+DDR for parameter 0x02012234

Suspend Sequence – pwr_domain_suspend()



Resume Sequence – pwr_domain_suspend_finish()



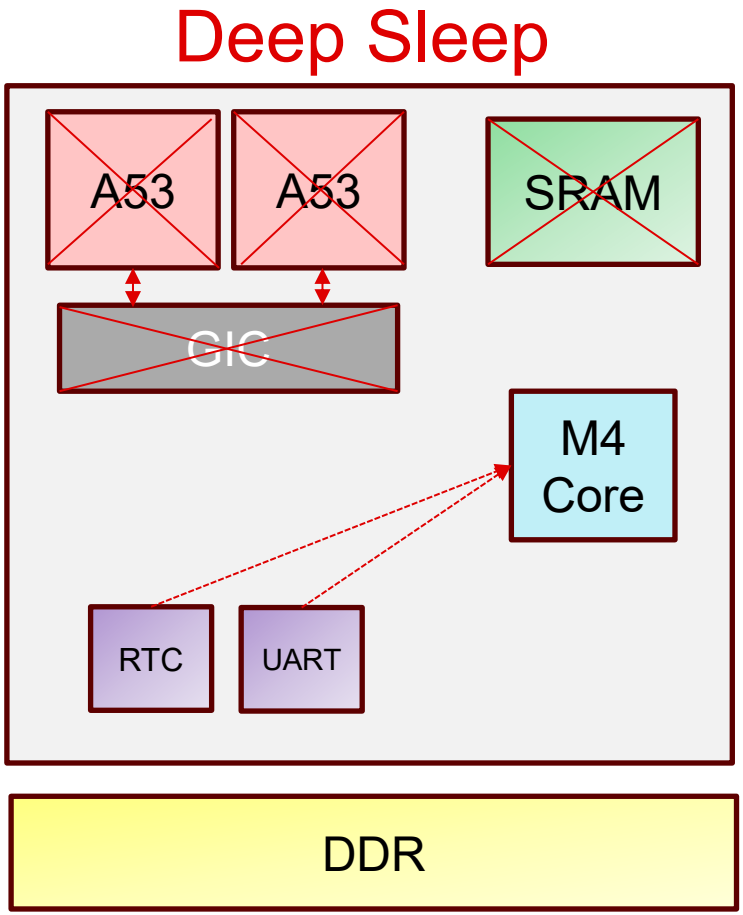
Challenges faced

Re-entry into Idle state

- GIC is in a turned OFF state and does not register any interrupt
- Wakeup IRQs go to the M4 co-processor which wakes up the system
- On resume, since there are no pending interrupts from GIC's perspective, we re-enter the idle states
- To prevent this we need to setup a pending interrupt

```
/* 60 irqn = RTC */
gicv3_set_spi_routing(60, GICV3_IRM_ANY, 0);
gicv3_enable_interrupt(60, 0);
gicv3_set_interrupt_pending(60, 0);
plat_ic_raise_ns_sgi(60, 0);
```

in pwr_domain_suspend_finish()



System State is LOST

Step	s2idle (`PM_SUSPEND_TO_IDLE`)	Deep Sleep (`PM_SUSPEND_MEM`)
Process freezer	✓ Same	✓ Same
PM notifiers (PM_SUSPEND_PREPARE)	✓ Same	✓ Same
`dpm_suspend_start()` — normal suspend	✓ Same	✓ Same
`ti_sci_suspend()` (dev PM ops)	✓ Runs, but **no PREPARE_SLEEP sent**	✓ Sends PREPARE_SLEEP/DM_MANAGED
`dpm_suspend_late()`	✓ Same	✓ Same
`dpm_suspend_noirq()`	✓ Same	✓ Same
`ti_sci_suspend_noirq()` — IO isolation	✗ No-op (state != MEM)	✓ IO pads isolated
Platform prepare (suspend_ops->prepare)	✗ Skipped for s2idle	✓ Runs if present
Secondary CPU offline	✗ All CPUs stay online	✓ Hot-unplug all non-boot CPUs
Global IRQ disable	✗ IRQs remain active	✓ `local_irq_disable()`
`syscore_suspend()`	✗ **SKIPPED ENTIRELY**	✓ **Runs all syscore callbacks**
PSCI call type	CPU_SUSPEND (per-CPU cpuidle)	SYSTEM_SUSPEND (system-wide)
Firmware notification	✗ None (no PREPARE_SLEEP)	✓ TIFS receives PREPARE_SLEEP
DDR self-refresh management	By PSCI/TF-A if domain sleeps	✓ Explicit via TIFS

syscore_ops

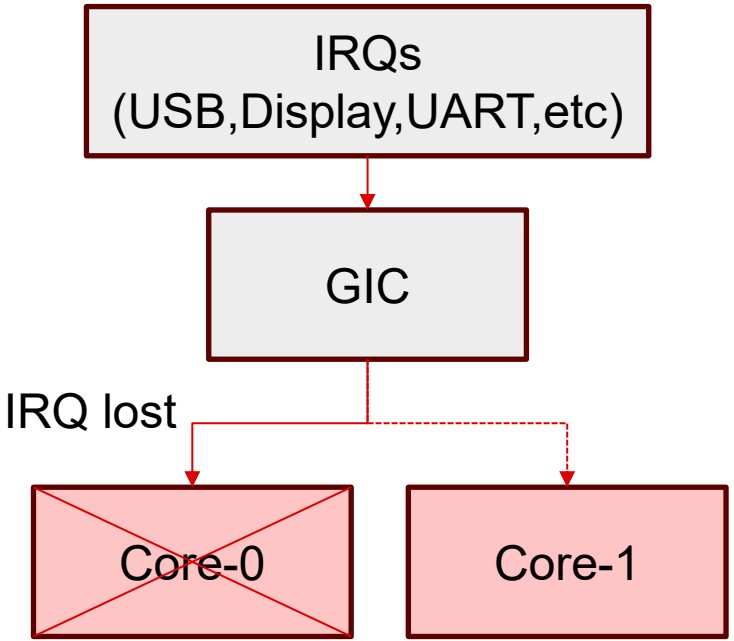
```
1. psci_idle_syscore_suspend()     – suspend CPU PM domains via genpd
2. its_save_disable()              – quiesce and save GIC ITS state
3. irq_gc_suspend()                – save generic IRQ chip state
4. cpu_pm_suspend()                – fire CPU_CLUSTER_PM_ENTER notifiers
5. timekeeping_suspend()           – snapshot wall clock, suspend clocksource
6. sched_clock_suspend()           – snapshot sched_clock epoch
[irq_pm has no .suspend – fires only on resume]
```

```
1. sched_clock_resume()            – restore sched_clock epoch
2. timekeeping_resume()            – resume clocksource, correct wall clock
3. cpu_pm_resume()                 – fire CPU_CLUSTER_PM_EXIT notifiers
4. irq_gc_resume()                 – restore generic IRQ chip state
5. irq_pm_syscore_resume()         – re-enable IRQF_EARLY_RESUME IRQs
6. its_restore_enable()            – reprogram and re-enable GIC ITS
7. psci_idle_syscore_resume()      – resume CPU PM domains via genpd
```

Interrupt Affinity Handling

- Most IRQs routed to Core-0
- In Standby High-Latency mode : Turn OFF one core dynamically (Eg. Core-0)
  - IRQ may target OFF core → Missed Wakeup
- Solution:
  - Re-route IRQS → Active Core
  - Restore affinity on resume

Is re-routing of interrupt affinity in firmware a Valid approach



Standby Mode Implementation

The two paths taken by PSCI

- After we have validated the power state parameter, we check pstate(16th bit of the power state parameter) and target_power_lvl
- Depending on the value we go to one of the two paths

pstate=0 & target_pwr_lvl=0
CPU fast-path

All other cases – Ex.
Standby mode : 0x01000021

```
static void am62l_cpu_standby(plat_local_state_t cpu_state)
{
    u_register_t scr;
    scr = read_scr_el3();
    /* Enable the Non secure interrupt to wake the CPU */
    write_scr_el3(scr | SCR_IRQ_BIT | SCR_FIQ_BIT);
    isb();
    /* dsb is good practice before using wfi to enter low power states */
    dsb();
    /* Enter standby state */
    wfi();
    /* Restore SCR */
    write_scr_el3(scr);
}
```

```
rc = psci_cpu_suspend_start(cpu_idx,
                            &ep,
                            target_pwrlvl,
                            &state_info,
                            is_power_down_state);
```

Cluster idle-state validation flow

Data structure used to store the requested power states

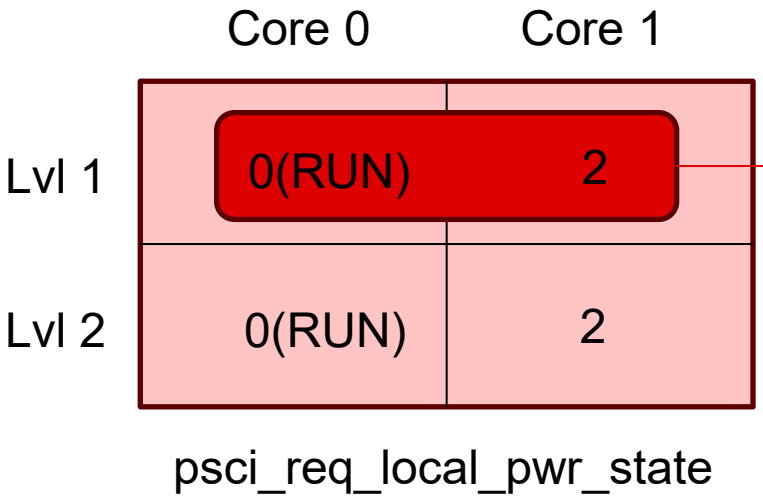
```
static plat_local_state_t
    psci_req_local_pwr_states[PLAT_MAX_PWR_LVL][PLATFORM_CORE_COUNT];
```

```
psci_update_req_local_pwr_states(end_pwrlvl, cpu_idx, state_info, prev);

for (lvl = PSCI_CPU_PWR_LVL + 1U; lvl <= end_pwrlvl; lvl++) {
    /* Get the requested power states for this power level */
    start_idx = psci_non_cpu_pd_nodes[parent_idx].cpu_start_idx;
    req_states = psci_get_req_local_pwr_states(lvl, start_idx);
    ncpus = psci_non_cpu_pd_nodes[parent_idx].ncpus;
    target_state = plat_get_target_pwr_state(lvl,
                                             req_states,
                                             ncpus);
    /*
     * Verify that the requested power state matches the target
     * local power state.
     */
    if (state_info->pwr_domain_state[lvl] != target_state) {
        if (target_state == PSCI_LOCAL_STATE_RUN) {
            rc = PSCI_E_DENIED;
        } else {
            rc = PSCI_E_INVALID_PARAMS;
        }
        goto exit;
    }
```

```
plat_local_state_t plat_get_target_pwr_state(unsigned int lvl,
                                             const plat_local_state_t *states,
                                             unsigned int ncpu)
{
    plat_local_state_t target = 6, temp;
    const plat_local_state_t *st = states;
    unsigned int n = ncpu;

    assert(ncpu > 0U);

    do {
        temp = *st;
        st++;
        if (temp < target)
            target = temp;
        n--;
    } while (n > 0U);

    return target;
}
```

Validation Problem

- In the standby mode flow, the psci_req_local_pwr_state is populated by **only one core** :




```
plat_local_state_t plat_get_target_pwr_state(unsigned int lvl,
                                             const plat_local_state_t *states,
                                             unsigned int ncpu)
{
    plat_local_state_t target = 6, temp;
    const plat_local_state_t *st = states;
    unsigned int n = ncpu;

    assert(ncpu > 0U);

    do {
        temp = *st;
        st++;
        if (temp < target)
            target = temp;
        n--;
    } while (n > 0U);

    return target;
}
```

Workaround

```
plat_local_state_t plat_get_target_pwr_state(unsigned int lvl,
                                             const plat_local_state_t *states,
                                             unsigned int ncpu)
{
    plat_local_state_t target = PLAT_MAX_OFF_STATE, temp;
    const plat_local_state_t *st = states;
    unsigned int n = ncpu;

    assert(ncpu > 0U);

    do {
        temp = *st;
        st++;
        /*  The power state of the CPU STANDBY called by fast path in psci_cpu_suspend()
         *  is RUN and the power states are in an increasing order of power saved.
         *  Thus the target power state for the cluster is the minimum of the power states
         *  requested by all the cores that is not RUN.
         */
        if ((temp < target) && ((temp != PSCI_LOCAL_STATE_RUN) || (lvl != MPIDR_AFFLVL1)))
            target = temp;
        n--;
    } while (n > 0U);

    return target;
}
```

Ignore RUN state for lvl=1(Cluster)

Open to Discussion

Thank You