PSCI OS-initiated mode in TF-A

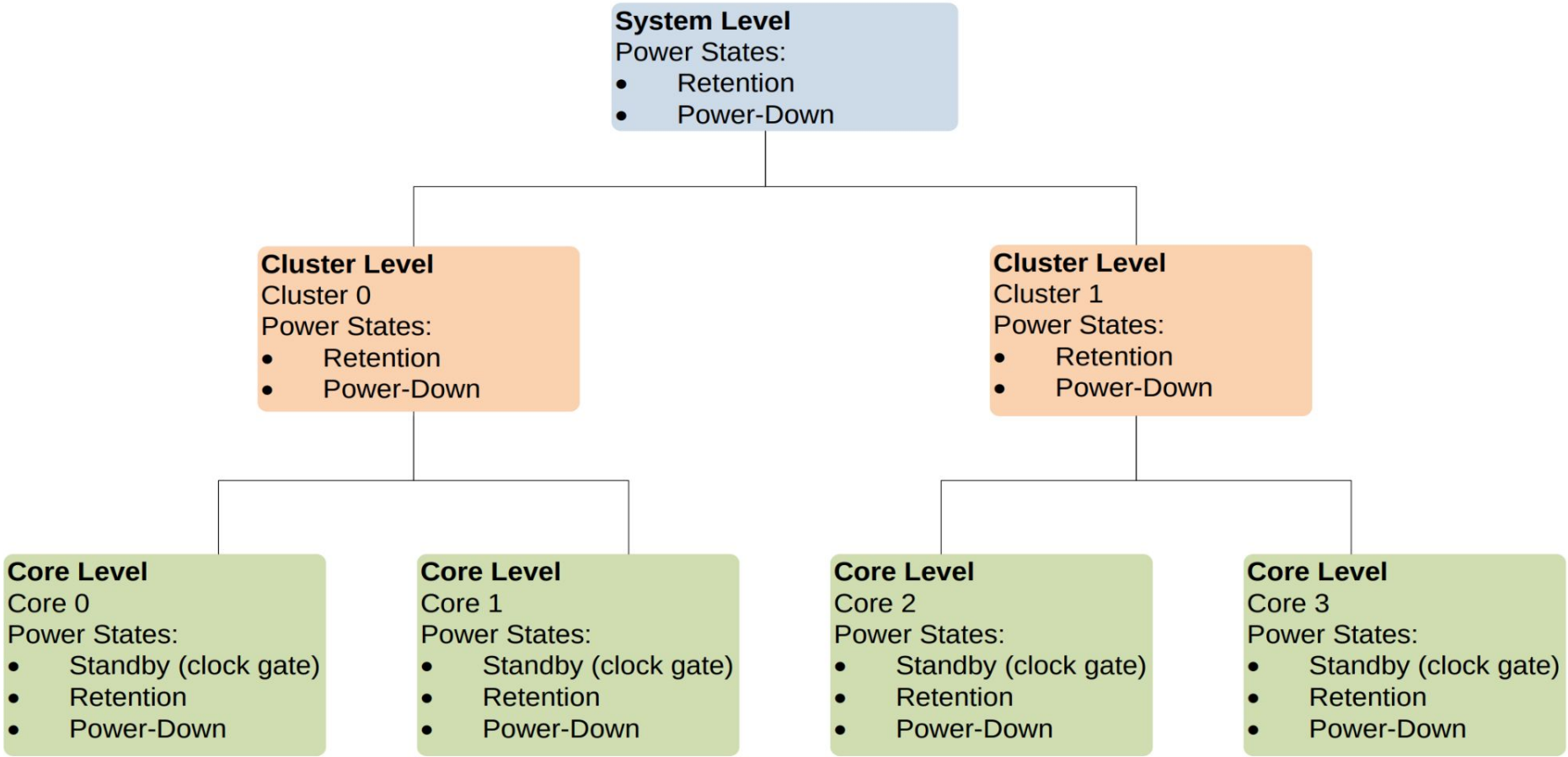
Maulik Shah (Qualcomm), Ulf Hansson (Linaro), Wing Li (Google)
23 Feb 2023

Agenda

- Intro to power state coordination
- Why OS-initiated mode?
- Current vendor implementations and workarounds
- Overview of proposed changes
- Testing on vendor platforms
- Requirements and implementation details

Intro to power state coordination

- Power domain topology
  - Logical hierarchy of power domains in a system
  - Arises from the dependencies between power domains
- Local power states and composite power states
  - Local power states describe power states for an individual node
  - Composite power states describe the combined power states for an individual node and its parent node(s)
- Power state coordination
  - Entry into low-power states for a node above the core level requires coordinating its children nodes


System Level
Power States:
• Retention
• Power-Down
Cluster Level
Cluster 0
Power States:
• Retention
• Power-Down
Cluster Level
Cluster 1
Power States:
• Retention
• Power-Down
Core Level
Core 0
Power States:
• Standby (clock gate)
• Retention
• Power-Down
Core Level
Core 1
Power States:
• Standby (clock gate)
• Retention
• Power-Down
Core Level
Core 2
Power States:
• Standby (clock gate)
• Retention
• Power-Down
Core Level
Core 3
Power States:
• Standby (clock gate)
• Retention
• Power-Down

Intro to power state coordination

- Platform-coordinated mode
  - Default mode of coordination
  - The platform keeps track of the composite power states requested for each core and their parents, and chooses the deepest composite state that is allowed by the requests from the cores
- OS-initiated mode
  - Optional mode of coordination
  - The OS may request for a composite power state when the last core running in a power domain goes idle

Why OS-initiated mode?

- Scalability
  - In platform-coordinated mode, each core independently selects their own idle states
    - Doesn't account for cluster idle states shared between cores
  - In OS-initiated mode, the OS has knowledge of next wakeup for each core and can decide when cluster idle state is appropriate
    - Especially important for multi-cluster SMP systems and heterogeneous systems like big.LITTLE

Why OS-initiated mode?

- Simplicity
  - In platform-coordinated mode, the OS doesn't have visibility in the last core of a power domain going idle
    - Requires communication with an extra API side channel to maintain system and peripheral states
    - Design smell when a platform is using platform-coordinated but actually wants to use OS-initiated
  - In OS-initiated mode, the OS can perform last man activity (e.g. power off shared rail) when the last core goes idle
    - Eliminates the extra API side channel
    - Uses the well documented API between OS and platform

Current vendor implementations and workarounds

- STMicroelectronics
  - For ARM32 platforms, currently using OP-TEE to implement OS-initiated mode
  - For future ARM64 platforms, would use TF-A for OS-initiated mode
- Qualcomm
  - For mobile platforms, currently using custom secure monitor to implement OS-initiated mode
  - For Chrome OS platforms, currently using platform-coordinated mode in TF-A with custom driver logic for last man activity
    - Would switch to OS-initiated mode to simplify custom driver logic
- Google
  - Currently using platform-coordinated mode in TF-A with custom driver logic for last man activity
    - Would switch to OS-initiated mode to simplify custom driver logic

Overview of proposed changes

- https://review.trustedfirmware.org/q/topic:psci-osi

Gerrit CHANGES YOUR DOCUMENTATION BROWSE

	Subject	Status	Owner	Reviewers	Repo
☆	feat(plat/qti/sc7280): Add support for PSCI_OS_INIT_MODE	--	Maulik		TF-A/trusted-firmware-a
☆	feat(fvp): enable support for PSCI OS-initiated mode	--	Wing Li		TF-A/trusted-firmware-a
☆	build(psci): add build option for OS-initiated mode	--	Wing Li		TF-A/trusted-firmware-a
☆	docs(psci): add design proposal for OS-initiated mode	--	**Wing Li**		TF-A/trusted-firmware-a
☆	feat(psci): update PSCI_FEATURES	--	Wing Li		TF-A/trusted-firmware-a
☆	feat(psci): add support for OS-initiated mode	--	Wing Li		TF-A/trusted-firmware-a
☆	feat(psci): add support for PSCI_SET_SUSPEND_MODE	--	Wing Li		TF-A/trusted-firmware-a
☆	test(psci): add tests for OS-initiated mode	--	**Wing Li**		TF-A/tf-a-tests

Testing on FVP and Google platforms

- https://review.trustedfirmware.org/c/TF-A/tf-a-tests/+/17684
- Added a new PSCI CPU Suspend in OSI mode test suite to TF-A Tests
- Tested on FVP_Base_RevC-2xAEMvA and Google platforms
- Excluded from all other platforms using the build option `PLAT_TESTS_SKIP_LIST`

Testing on STM32MP15

- Tested by Gabriel Fernandez (gabriel.fernandez@st.com)
- 1 cluster, 2 cores

```
cpus {
        #address-cells = <1>;
        #size-cells = <0>;

        cpu0: cpu@0 {
                compatible = "arm,cortex-a7";
                device_type = "cpu";
                reg = <0>;
                clocks = <&scmi_clk CK_SCMI_MPU>;
                clock-names = "cpu";
                #cooling-cells = <2>;
                enable-method = "psci";
                power-domains = <&CPU_PD0>;
                power-domain-names = "psci";
        };
        cpu1: cpu@1 {
                compatible = "arm,cortex-a7";
                device_type = "cpu";
                reg = <1>;
                clocks = <&scmi_clk CK_SCMI_MPU>;
                clock-names = "cpu";
                enable-method = "psci";
                power-domains = <&CPU_PD1>;
                power-domain-names = "psci";
        }

        idle-states {
                cpu_retention: cpu-retention {
                        compatible = "arm,idle-state";
                        local-timer-stop;
                        entry-latency-us = <130>;
                        exit-latency-us = <620>;
                        min-residency-us = <700>;
                        arm,psci-suspend-param = <0x00000001>;
                };
        };

        domain-idle-states {
                CLUSTER_STOP: core-power-domain {
                        compatible = "domain-idle-state";
                        arm,psci-suspend-param = <0x01000001>;
                        entry-latency-us = <230>;
                        exit-latency-us = <720>;
                        min-residency-us = <2000>;
                };
        };
};
```

```
psci {
        compatible = "arm,psci-1.0";
        method = "smc";

        CPU_PD0: power-domain-cpu0 {
                #power-domain-cells = <0>;
                domain-idle-states = <&cpu_retention>;
                power-domains = <&pd_core>;
        };

        CPU_PD1: power-domain-cpu1 {
                #power-domain-cells = <0>;
                domain-idle-states = <&cpu_retention>;
                power-domains = <&pd_core>;
        };

        pd_core: power-domain-cluster {
                #power-domain-cells = <0>;
                domain-idle-states = <&CLUSTER_STOP>;
        };
};
```

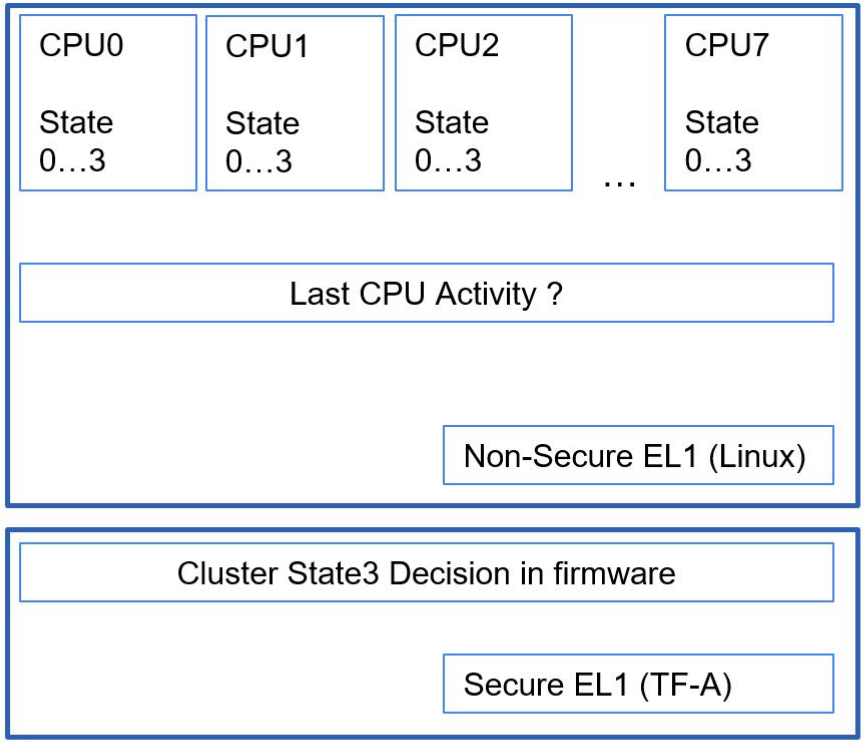
Testing on Qcom SC7280

- Platform: Qualcomm SC7280 – Available in upstream kernel and TF-A
  - Currently SC7280 uses PSCI platform-coordinated
- Tested by Maulik Shah (quic_mkshah@quicinc.com)
- Patches to use OS-Initiated on SC7280
  - Linux: https://patchwork.kernel.org/project/linux-arm-msm/list/?series=722018
  - Trusted Firmware: https://review.trustedfirmware.org/c/TF-A/trusted-firmware-a/+/19487

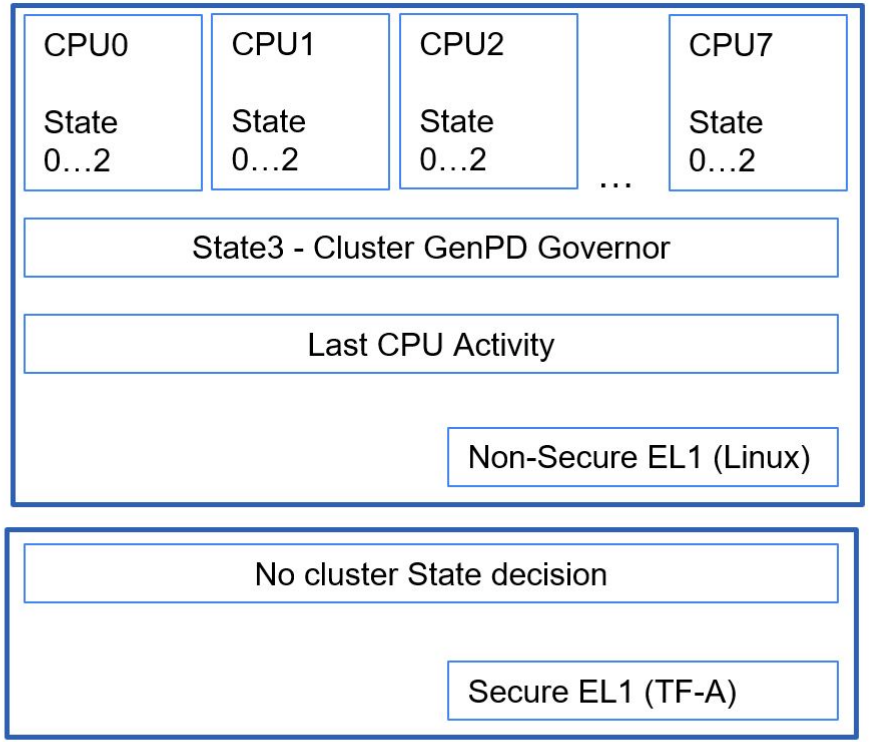
SC7280 - CPUIdle states

- 8 CPUs, 1 L3 cache
- CPU Idle States – Platform-coordinated
  - State0 – WFI
  - State1 – Core Collapse
  - State2 – Rail Collapse
  - State3 – L3 off + System resources voted off
- CPU Idle States – OS-Initiated
  - State0 – WFI
  - State1 – Core Collapse
  - State2 – Rail Collapse
- Cluster domain idle state
  - State3 - L3 off + System resources voted off

PSCI: Flattened v/s Hierarchical view of idle states



Platform-coordinated



OS-Initiated

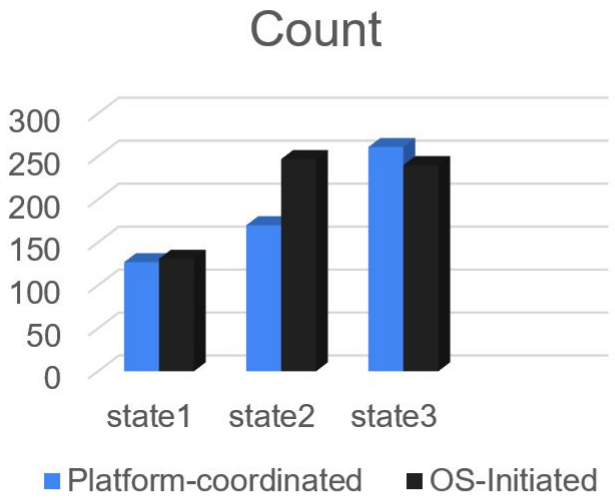
PSCI - OS-Initiated vs Platform-coordinated comparison

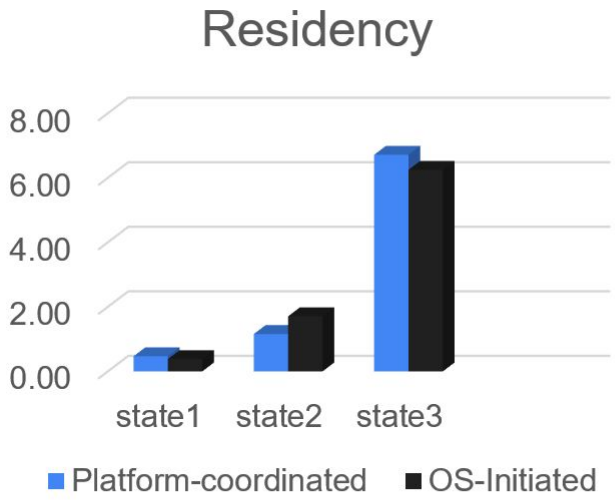
- Use case: display on, wifi, modem off – 10 seconds

- Numbers
  - Residency: Time in seconds cpu / cluster stays in power collapse
  - Count: No. of times cpu / cluster entered in power collapse
  - Average of 3 iterations for actual count and residency
  - Numbers captured with fixed CPU frequencies

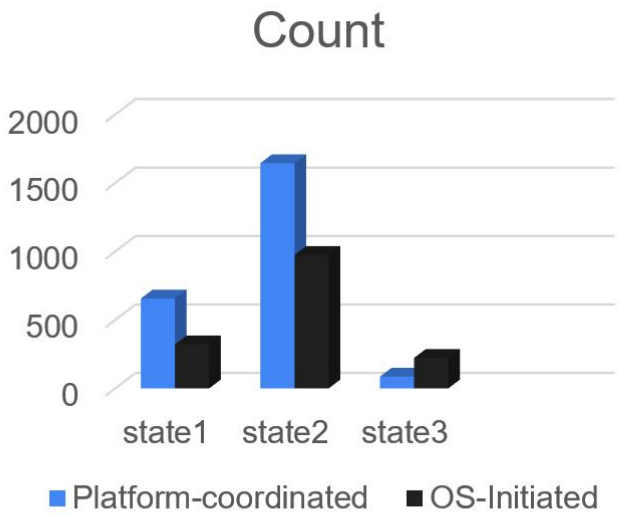
- Results for
  - Only single cpu online
  - Multiple cpus (8) online

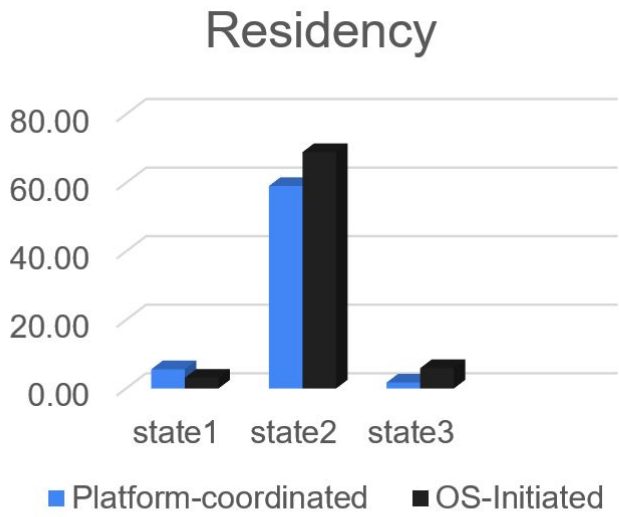
Stats Comparison – Single CPU





Stats Comparison – Multiple CPUs





Requirements

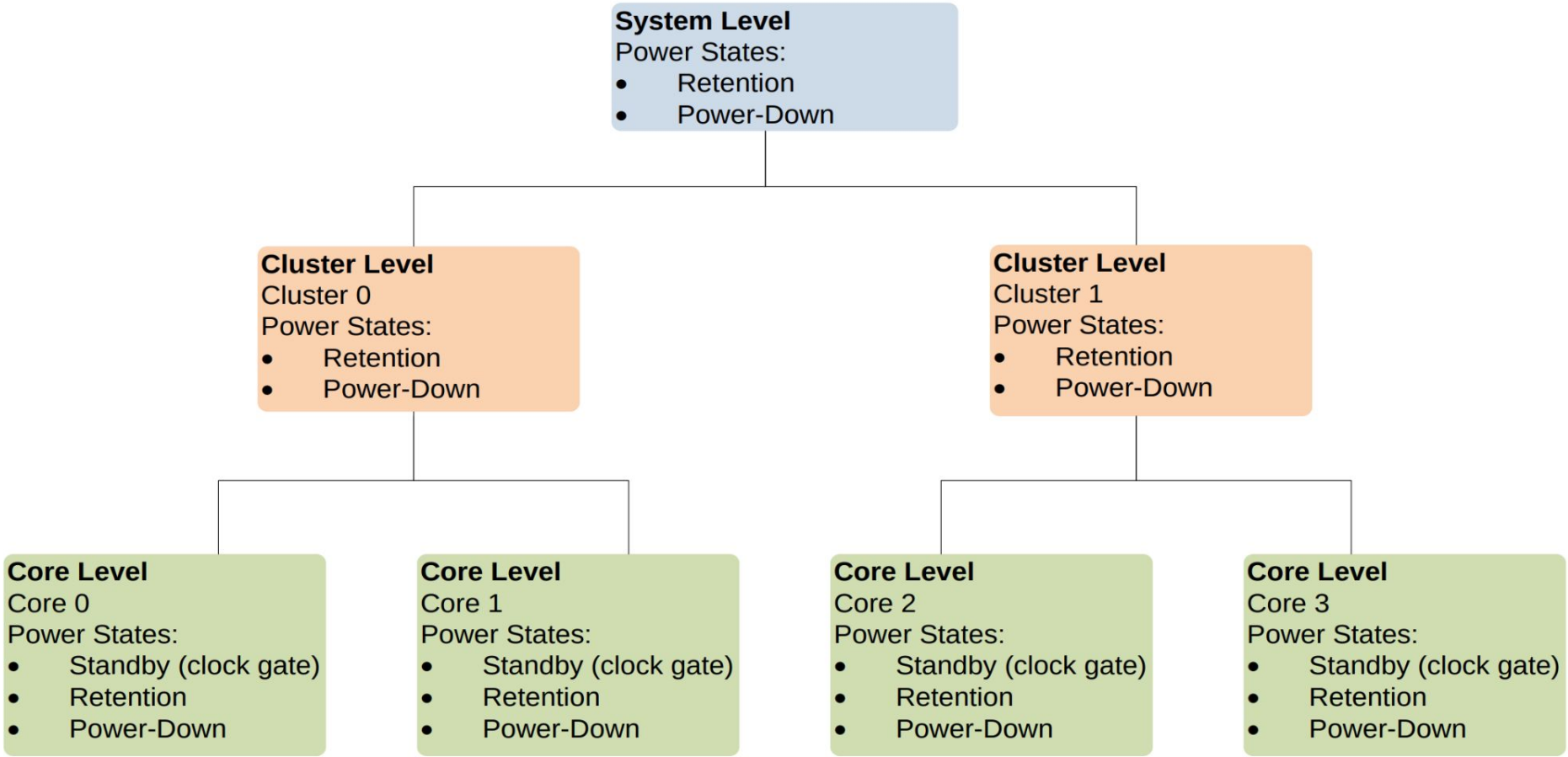
- `PSCI_FEATURES`
  - Function to detect whether a PSCI function is implemented and its properties
- `PSCI_SET_SUSPEND_MODE`
  - Function to switch between the two different modes of power state coordination
    - The default mode of coordination is platform-coordinated mode
- `CPU_SUSPEND`
  - Function to move a topology node into a low-power state
  - In platform-coordinated mode, the PSCI implementation coordinates requests from all cores to determine the deepest state to enter
  - In OS-initiated mode, the PSCI implementation must comply with the request
    - If a request is inconsistent with the implementation's view of the system's state, the request must be rejected

Requirements

- Race conditions in OS-initiated mode
  - The OS might request an idle state for a node from one core, while at the same time, the PSCI implementation observes that another core in the node is powering up
  - If the PSCI implementation doesn't process requests in the order the OS intended, then the implementation can end up in an incorrect state

Requirements

- Addressing race conditions
  - The OS must specify the deepest power level for which it sees the calling core as the last running core (last man)
    - Even if the OS doesn't want a node at a certain power level to go idle, it must still indicate if the core is the last core to go idle at that power level
  - The PSCI implementation must check whether the calling core is the last core to go idle in the requested power level, or otherwise reject the request


System Level
Power States:
• Retention
• Power-Down
Cluster Level
Cluster 0
Power States:
• Retention
• Power-Down
Cluster Level
Cluster 1
Power States:
• Retention
• Power-Down
Core Level
Core 0
Power States:
• Standby (clock gate)
• Retention
• Power-Down
Core Level
Core 1
Power States:
• Standby (clock gate)
• Retention
• Power-Down
Core Level
Core 2
Power States:
• Standby (clock gate)
• Retention
• Power-Down
Core Level
Core 3
Power States:
• Standby (clock gate)
• Retention
• Power-Down

Step		OS View			PSCI View		
		Core0	Core1	Cluster0	Core0	Core1	Cluster0
0: Initial state. Core0 is in powerdown, and Core1 is running.	before	PD	R	R	PD	R	R
	after	PD	R	R	PD	R	R
1: Core1 goes idle. The OSPM requests that Core1 enter powerdown and Cluster0 enter Retention state.	before	PD	R	R	PD	R	R
	after	PD	PD	Ret	PD	R	R
2: Core0 receives an interrupt and wakes up into the PSCI implementation in the secure platform firmware.	before	PD	PD	Ret	PD	R	R
	after	PD	PD	Ret	R	R	R
3: Core0 moves into OSPM and starts processing interrupts.	before	PD	PD	Ret	R	R	R
	after	R	PD	R	R	R	R
4: Core0 goes idle and OSPM requests Core0 powerdown, Cluster0 powerdown.	before	R	PD	R	R	R	R
	after	PD	PD	PD	R	R	R
5: Core0's idle request overtakes Core1's request. The implementation puts Core0 into powerdown but ignores the cluster request since it sees Core1 is still running.	before	PD	PD	PD	R	R	R
	after	PD	PD	PD	PD	R	R
6: The implementation observes Core1's request and puts Core1 to powerdown and Cluster0 to retention.	before	PD	PD	PD	PD	R	R
	after	PD	PD	PD	PD	PD	Ret

Step		OS View			PSCI View		
		Core0	Core1	Cluster0	Core0	Core1	Cluster0
Resuming from Step 4 above	before	R	PD	R	R	R	R
	after	PD	PD	PD	R	R	R
5: Core0's idle request overtakes Core1's request. **The implementation rejects Core0's request since it includes Cluster0 and Core1 is still awake.**	before	PD	PD	PD	R	R	R
	after	PD	PD	PD	R	R	R
6: The implementation observes Core1's request and rejects it **since it includes Cluster0 but Core0 is still awake.**	before	PD	PD	PD	R	R	R
	after	PD	PD	PD	R	R	R
7: The OS resumes on Core 0.	before	PD	PD	PD	R	R	R
	after	R	PD	R	R	R	R
8: The OS resumes on Core 1.	before	R	PD	R	R	R	R
	after	R	R	R	R	R	R

Step		OS View			PSCI View		
		Core0	Core1	Cluster0	Core0	Core1	Cluster0
0: Initial state Core0 in powerdown, and Core1 is running	before	PD	R	R	PD	R	R
	after	PD	R	R	PD	R	R
1: Core1 goes idle – the OSPM requests Core1 powerdown and Cluster0 retention	before	PD	R	R	PD	R	R
	after	PD	PD	Ret	PD	R	R
2: Core0 receives an interrupt and wakes up into the implementation	before	PD	PD	Ret	PD	R	R
	after	PD	PD	Ret	R	R	R
3: Core0 moves into OSPM and starts processing interrupts	before	PD	PD	Ret	R	R	R
	after	R	PD	R	R	R	R
4: Core0 goes idle and OSPM requests Core0 powerdown, Cluster0 to stay Running	before	R	PD	R	R	R	R
	after	PD	PD	R	R	R	R
5: Core0's idle request overtakes Core1's request. The implementation puts Core0 to powerdown.	before	PD	PD	R	R	R	R
Even though the OS made a request for the cluster to run, the implementation does not know to reject Core0's request, since it doesn't include a Cluster state	after	PD	PD	R	PD	R	R
6: Core1's request is observed by the implementation. The implementation puts Core1 to Power Down and Cluster0 to retention	before	PD	PD	R	PD	R	R
	after	PD	PD	R	PD	PD	Ret

Step		OS View			PSCI View		
		Core0	Core1	Cluster0	Core0	Core1	Cluster0
0: Initial state Core0 in powerdown, and Core1 is running	before	PD	R	R	PD	R	R
	after	PD	R	R	PD	R	R
1: Core1 goes idle – the OSPM requests Core1 powerdown and Cluster0 retention and **identifies itself as last down in Cluster0**	before	PD	R	R	PD	R	R
	after	PD	PD	Ret	PD	R	R
2: Core0 receives an interrupt and wakes up into the implementation	before	PD	PD	Ret	PD	R	R
	after	PD	PD	Ret	R	R	R
3: Core0 moves into OSPM and starts processing interrupt	before	PD	PD	Ret	R	R	R
	after	R	PD	R	R	R	R
4: Core0 goes idle and OSPM requests Core0 powerdown, Cluster0 to stay Running, and **identifies itself as last down in Cluster0**	before	R	PD	R	R	R	R
	after	PD	PD	R	R	R	R
5: Core0's idle request overtakes Core1's request. The implementation rejects it, as it is a cluster request and Core 1 is still running	before	PD	PD	R	R	R	R
	after	PD	PD	R	R	R	R
6: Core1's request is observed by the implementation. The implementation rejects it, as it is a cluster request and Core 0 is still running.	before	PD	PD	R	R	R	R
	after	PD	PD	R	R	R	R
7: OS resumes on Core 0	before	PD	PD	R	R	R	R
	after	R	PD	R	R	R	R
8: OS resumes on Core 1	before	R	PD	R	R	R	R
	after	R	R	R	R	R	R

Current implementation of platform-coordinated mode

- The functions of interest in the `CPU_SUSPEND` call stack
  - `psci_validate_power_state`
    - Calls a platform specific `validate_power_state` handler, which updates the `state_info` object with the requested states for each power level
  - `psci_find_target_suspend_lvl`
    - Returns the deepest power level that was requested to enter a low-power state
  - `psci_do_state_coordination`
    - Takes the target power level and the `state_info` object, and updates the `state_info` object with the coordinated target power state for each level
  - `pwr_domain_suspend`
    - This platform specific function performs the necessary actions to suspend the calling core and its parents based on the `state_info` object

Proposed implementation of OS-initiated mode

- Add a boolean build option `PSCI_OS_INIT_MODE` for a platform to enable optional support for PSCI OS-initiated mode
  - Defaults to 0
- If `PSCI_OS_INIT_MODE=0`, the changes on the following slides will not be compiled into the build

Proposed implementation of OS-initiated mode

- `PSCI_FEATURES`
  - Update `psci_features` to return 1 in bit[0] to indicate support for OS-initiated mode for CPU_SUSPEND
- `PSCI_SET_SUSPEND_MODE`
  - Define a `suspend_mode` enum: `PLAT_COORD` and `OS_INIT`
  - Define a `psci_suspend_mode` global variable with a default value of `PLAT_COORD`
  - Implement a new function handler `psci_set_suspend_mode`

Proposed implementation of OS-initiated mode

- `CPU_SUSPEND`
  - Update the platform specific `validate_power_state` handler to populate the `state_info` object based on the state ID from the power state parameter
  - `psci_find_target_suspend_lvl` remains unchanged
  - Implement a new function `psci_validate_state_coordination` that ensures the request satisfies the following conditions
    - The requested power states for each power level are consistent with the system's state
    - The calling core is the last core running at the requested power level
  - Note: this differs from `psci_do_state_coordination` in that
    - The `psci_req_local_pwr_states` map must not be modified if the request were rejected
    - The `state_info` argument must not be modified since it contains the requested power states from the OS

Proposed implementation of OS-initiated mode

- `CPU_SUSPEND`
  - Update `psci_cpu_suspend_start` to do the following
    - If `PSCI_SUSPEND_MODE` is `PLAT_COORD`, call `psci_do_state_coordination`
    - If `PSCI_SUSPEND_MODE` is `OS_INIT`, call `psci_validate_state_coordination`
      - If validation fails, propagate the error up the call stack
  - Update the return type of the platform specific `pwr_domain_suspend` handler from `void` to `int`
    - Allows the platform to perform validations based on HW states

Questions?