

1. Overview

1.1 Overview

This manual explains power management for thermal system for R-Car H3/M3/M3N/E3/D3/V3U/V3H/V3M System Evaluation Board.

1.2 Function

Power management for thermal system supports the following functions:

Table 1-1 Power management functions

Function name	Overview of software specification
CPU Hotplug	CPU Hotplug is a function that user controls online and offline for CPU.
CPU Idle	CPU Idle is a function that Kernel dynamically enters CPU to Sleep mode or Core Standby mode based on the time until the next timer interrupt. If time until the next timer interrupt is longer, it enters to Core Standby mode with higher energy saving effect.
CPU Freq	CPU Freq is to control frequency and voltage for CPU.
Runtime PM	Runtime PM is a function to manage the clock and power domain of each device, and to control on or off for them according to the usage status of each device.
System Suspend to RAM	System Suspend to RAM is a function to halt all of the power supplies except for the backup power supply by saving the state of system to RAM for backup to reduce power consumption. Also, it's faster than cold boot because program load and initialization of device driver are skipped. In addition, the backup targets are devices of Salvator-X/XS, Ebisu standard.
IPA	IPA is a function to control DVFS for CPU when it's predicting excess than limit temperature.
EMS	EMS is a function to turn off CPU core when it's predicting excess than limit temperature.

Depending on each platform, the support of power management functions are different. Below table show more detail about support of power management functions on each R-Car Series, 3rd Generation platform:

Table 1-2 Detailed support of power management functions (R-Car H3/M3/M3N/E3/D3)

Platform Functions	R-Car H3	R-Car M3	R-Car M3N	R-Car E3	R-Car D3
CPUIdle	CA57 (0/1/2/3) CA53 (4/5/6/7)	CA57 (0/1) CA53 (2/3/4/5)	CA57 (0/1)	CA53 (0/1)	-
CPUFreq (DFS)	CA57 (0/1/2/3) CA53 (4/5/6/7)	CA57 (0/1) CA53 (2/3/4/5)	CA57 (0/1)	CA53 (0/1)	-
CPUFreq (ADVFS)	CA57 (0/1/2/3)	CA57 (0/1)	CA57 (0/1) [*]	-	-
CPUHotplug	CA57 (1/2/3) CA53 (4/5/6/7)	CA57 (1) CA53 (2/3/4/5)	CA57 (1)	CA53 (1)	-
IPA	CA57 (0/1/2/3) CA53 (4/5/6/7)	CA57 (0/1) CA53 (2/3/4/5)	CA57 (0/1)	CA53 (0/1)	-
EMS	CA57 (0/1/2/3) CA53 (4/5/6/7)	CA57 (0/1) CA53 (2/3/4/5)	CA57 (0/1)	CA53 (0/1)	-
System Suspend to RAM	Supported	Supported	Supported	Supported	-
Runtime PM	Supported	Supported	Supported	Supported	Supported

Note: [*] R-Car M3N just supports DVFS, not support AVS.

Table 1-3 Detailed support of power management functions (R-Car V3U/V3H/V3M)

Platform Functions	R-Car V3U	R-Car V3H	R-Car V3M
CPUIdle	-	-	-
CPUFreq (DFS)	-	-	-
CPUFreq (ADVFS)	-	-	-
CPUHotplug	-	-	-
IPA	-	-	-
EMS	-	-	-
System Suspend to RAM	-	-	-
Runtime PM	Supported	Supported	Supported

1.3 Reference

1.3.1 Standard

There is no reference document on standards.

1.3.2 Related documents

The following table shows the documents related to this power management for thermal system.

Table 1-4 Related document

Number	Issue	Title	Edition	Date
-	Renesas Electronics	R-Car Series, 3 rd Generation User's Manual: Hardware	Rev.2.20	Jun. 30, 2020
-	Renesas Electronics	R-CarH3-SiP System Evaluation Board Salvator-X Hardware Manual RTP0RC7795SIPB0011S	Rev.1.09	May. 11, 2017
-	Renesas Electronics	R-CarM3-SiP System Evaluation Board Salvator-X Hardware Manual RTP0RC7796SIPB0011S	Rev.0.04	Oct. 3, 2016
-	Renesas Electronics	R-CarH3-SiP/M3-SiP/M3N-SiP System Evaluation Board Salvator-XS Hardware Manual	Rev.2.04	Jul. 17, 2018
-	Renesas Electronics	R-CarE3 System Evaluation Board Ebisu Hardware Manual RTP0RC77990SEB0010S	Rev.0.03	Apr. 11, 2018
-	Renesas Electronics	R-CarE3 System Evaluation Board Ebisu-4D (E3 board 4xDRAM) Hardware Manual	Rev.1.01	Jul. 19, 2018
-	Renesas Electronics	R-CarD3 System Evaluation Board Hardware Manual RTP0RC77995SEB0010S	Rev.1.20	Jul. 25, 2017
-	Renesas Electronics	Security Board Support Package User's Manual	#0	-
-	Renesas Electronics	Initial Program Loader User's Manual: Software R-Car H3/M3/M3N/E3 Series	#0	-
-	Renesas Electronics	R-Car V3U series User's Manual	Rev.0.5	Jul. 31, 2020
-	Renesas Electronics	R-CarV3U System Evaluation Board Falcon Hardware Manual	Rev.0.01	Sep. 11, 2020
-	Renesas Electronics	R-Car V3H_2 Additional Document for User's Manual: Hardware	Rev.0.50	Jul. 31, 2020
-	Renesas Electronics	R-CarV3H System Evaluation Board Condor-I Hardware Manual	Rev.0.02	Nov. 11, 2019
-	Renesas Electronics	R-Car V3M R-Car Series, 3 rd Generation User's Manual: Hardware	Rev.3.00	Jun. 30, 2020
-	Renesas Electronics	R-CarV3M System Evaluation Board Eagle Hardware Manual	Rev.0.09	Nov. 20, 2017

#0: This manual refers to the latest edition.

1.4 Restrictions

- None

1.5 Notice

- None

2. Terminology

The following table shows the terminology related to this power management for thermal system.

Table 2-1 Terminology

Terms	Explanation
DVFS	<u>D</u> ynamic <u>V</u> oltage and <u>F</u> requency <u>S</u> caling
AVS	<u>A</u> daptive <u>V</u> oltage <u>S</u> caling
PMIC	<u>P</u> ower <u>M</u> anagement <u>I</u> ntegrated <u>C</u> ircuit
PSCI	<u>P</u> ower <u>S</u> tate <u>C</u> oordinate <u>I</u> nterface
CA57	<u>C</u> ortex <u>A57</u> for AP-System Core
CA53	<u>C</u> ortex <u>A53</u> for AP-System Core
APMU	<u>A</u> dvanced <u>P</u> ower <u>M</u> anagement <u>U</u> nit for AP-System Core
RST	<u>R</u> es <u>e</u> t
SYSC	<u>S</u> ystem <u>C</u> ontroller
GIC	<u>G</u> eneric <u>I</u> nterrupt <u>C</u> ontroller
CPG	<u>C</u> lock <u>P</u> ulse <u>G</u> enerator
IIC-DVFS	<u>I</u> I <u>C</u> bus interface for <u>DVFS</u>
ADVFS	<u>A</u> daptive <u>D</u> ynamic <u>V</u> oltage and <u>F</u> requency <u>S</u> caling
GPIO	<u>G</u> eneral <u>P</u> urpose <u>I</u> nput / <u>O</u> utput interface
SMC	<u>S</u> ecure <u>M</u> onitor <u>C</u> all
WFI	<u>W</u> ait <u>F</u> or <u>I</u> nterrupt
IPA	<u>I</u> ntelligent <u>P</u> ower <u>A</u> llocation
EMS	<u>E</u> mergency <u>S</u> hutdown

3. System Configuration

3.1 Hardware Environment

The following table shows the hardware needed to use this power management for thermal system.

Table 3-1 Hardware environment

Name	Version	Manufacture
R-CarH3-SiP System Evaluation Board Salvator-X	-	Renesas Electronics
R-CarM3-SiP System Evaluation Board Salvator-X	-	Renesas Electronics
R-CarH3-SiP/M3-SiP/M3N-SiP System Evaluation Board Salvator-XS	-	Renesas Electronics
R-CarE3 System Evaluation Board Ebisu R-CarE3 System Evaluation Board Ebisu-4D	-	Renesas Electronics
R-CarD3 System Evaluation Board Draak	-	Renesas Electronics
R-CarV3U System Evaluation Board Falcon	-	Renesas Electronics
R-CarV3H System Evaluation Board Condor-I	-	Renesas Electronics
R-CarV3M System Evaluation Board Eagle	-	Renesas Electronics
R-CarV3M Function Expansion Board	-	Renesas Electronics

3.2 Software Configuration

The following figure shows the software configuration of power management for thermal system.

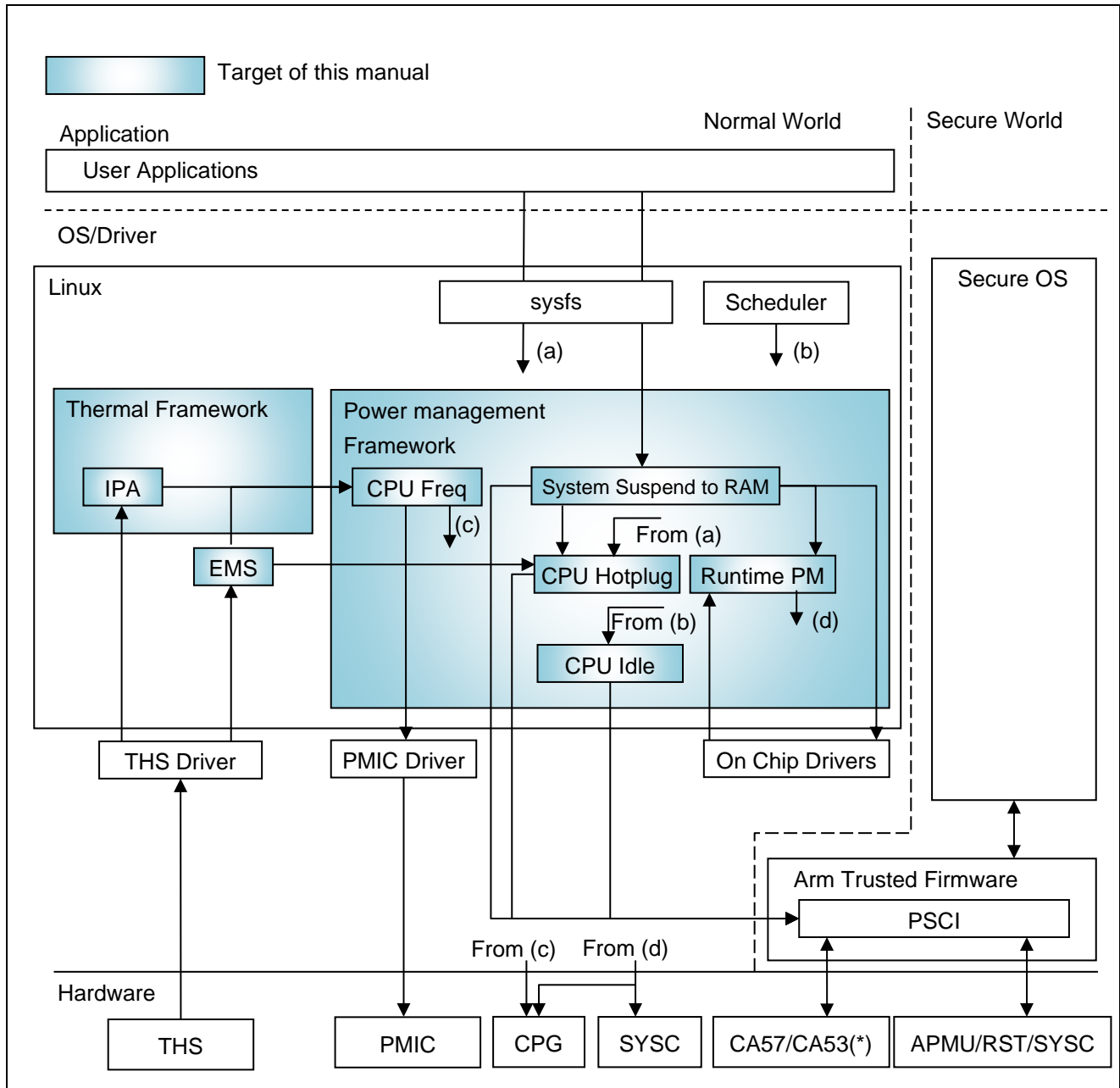


Figure 3-1 Block diagram of power management for thermal system (R-Car H3/M3/M3N/E3)

PSCI defines an Arm standard interface. It supports communication between Normal world and Secure world. Then, it allows Arm Trusted Firmware to arbitrate power management requests from secure software and Linux.

The following table shows the PSCI function that power management uses.

(*) Note: Support for CA57 only on R-Car H3/M3/M3N, support for CA53 only on R-Car H3/M3/E3.

Table 3-2 PSCI Function

Function	Description
CPU_OFF	Power down the calling core. This call is intended for use in hotplug. The core that is powered down by CPU_OFF can only be powered up again in response by CPU_ON.
CPU_ON	Power up a core. This call is intended for dynamic addition of cores, for use in secondary boot, hotplug, or big.LITTLE migration.
CPU_SUSPEND	Suspend execution on a core or higher level topology node. This call is intended for use in idle subsystems where the core is expected to return to execution through a wakeup event.
SYSTEM_SUSPEND	Enable System Suspend to RAM support. The semantics are equivalent to CPU_SUSPEND to the deepest low-power state.

 Target of this manual

Application

User Applications

OS/Driver

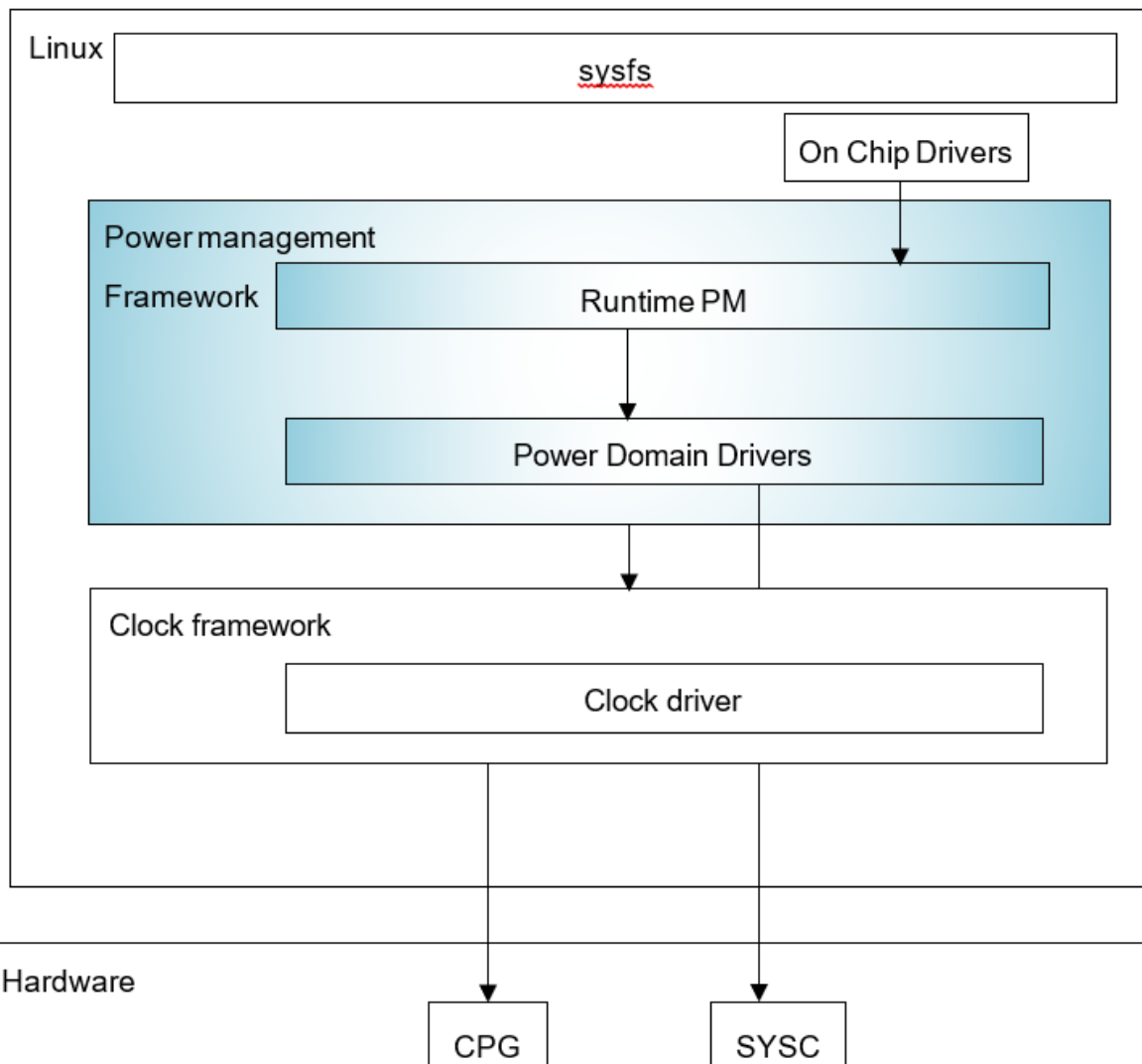


Figure 3-2 Block diagram of power management for thermal system (R-Car D3)

Note: R-Car D3 Draak board does not have PMIC.

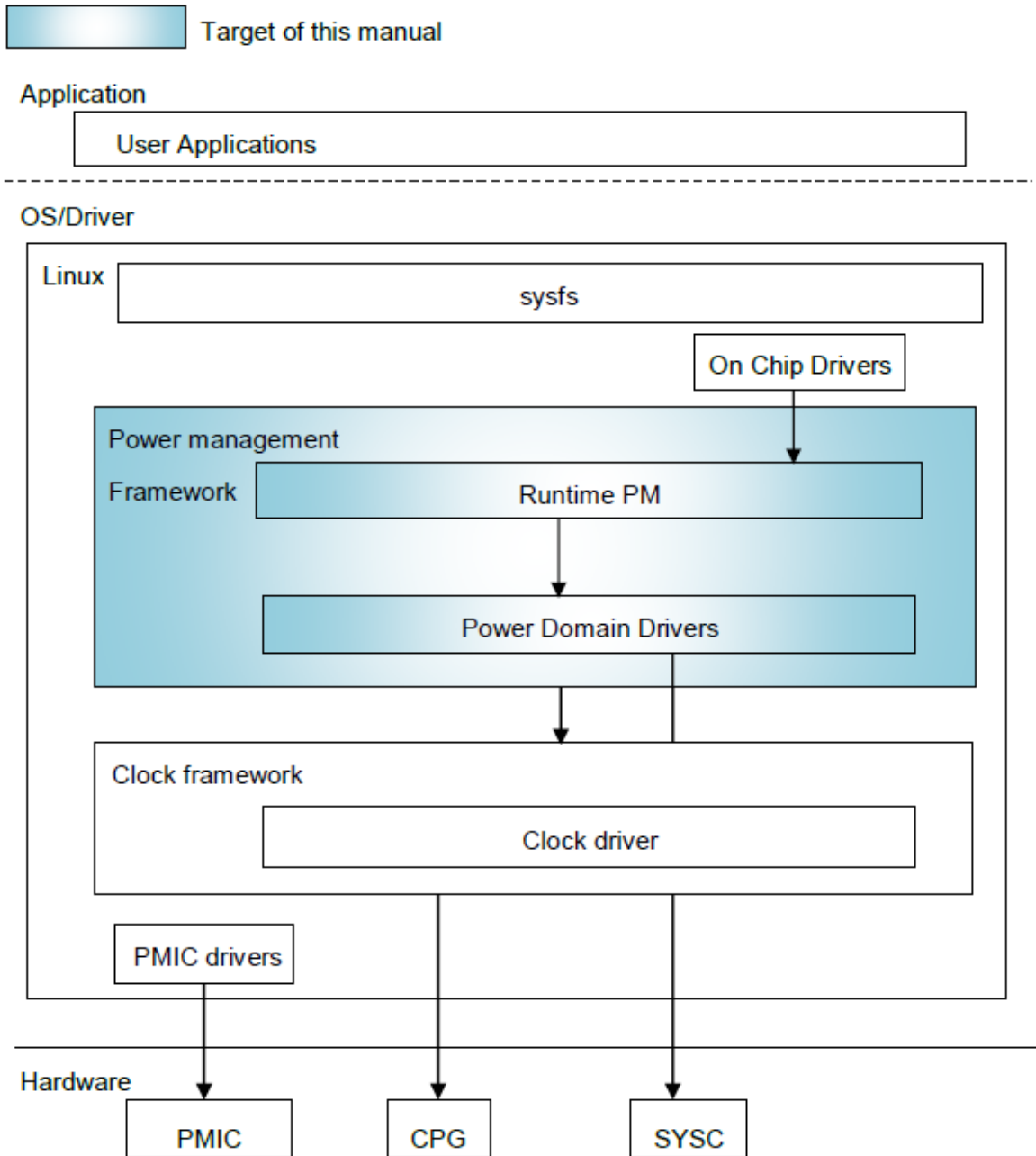


Figure 3-3 Block diagram of power management (R-Car V3U/V3H/V3M)

4. Function

4.1 CPU Hotplug

CPU Hotplug is Linux function to perform online and offline on multi-core system.

CPU Online: plug each CPU into the system

CPU Offline: unplug each CPU from the system

4.1.1 CPU Offline

The following figure shows the processing flow of CPU Offline.

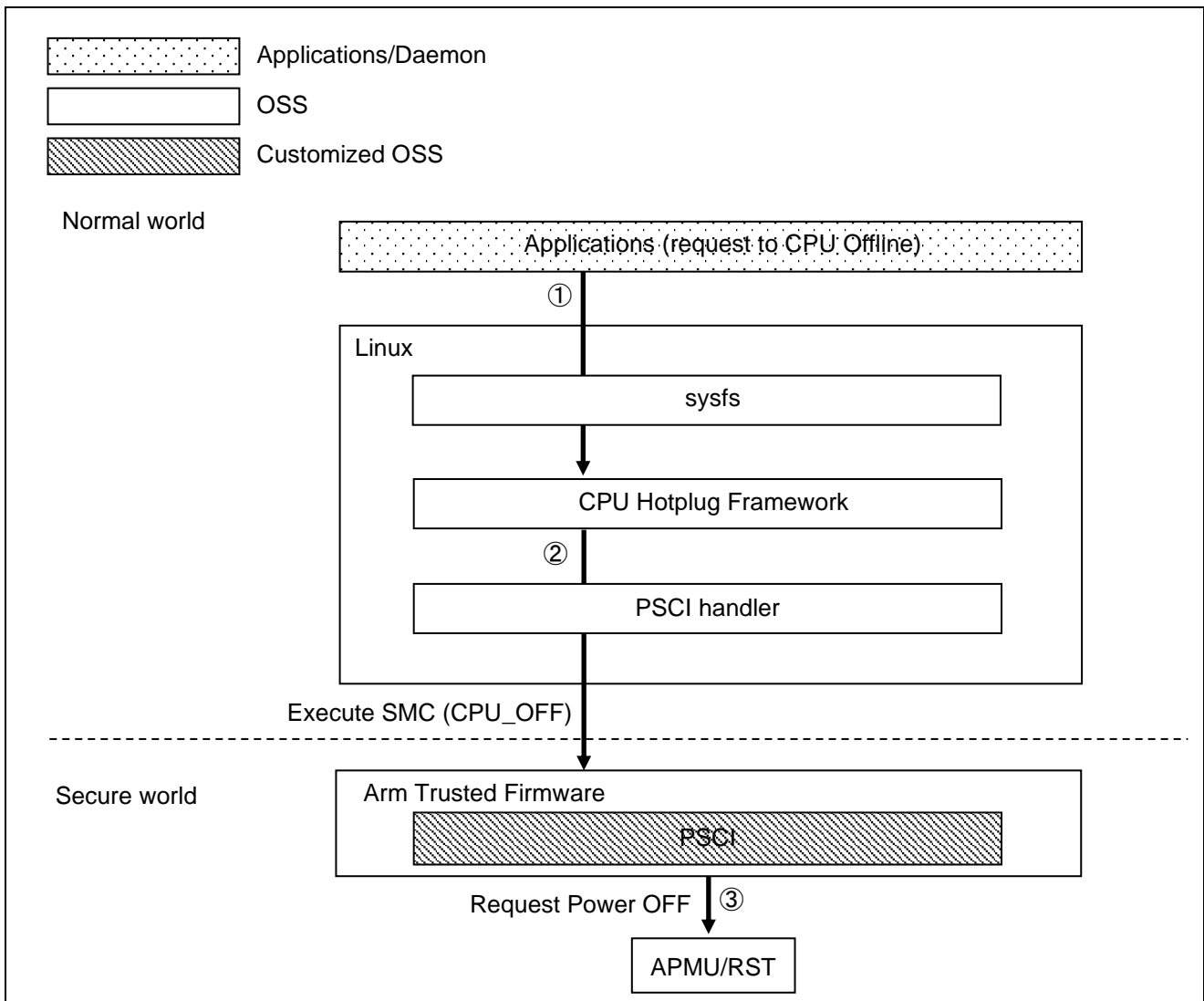


Figure 4-1 Processing flow of CPU Offline

① Applications request CPU Offline via sysfs.

echo 0 > /sys/devices/system/cpu/cpuY/online (R-Car H3: Y = 1/2/3/4/5/6/7; R-Car M3: Y=1/2/3/4/5; R-Car M3N: Y = 1; R-Car E3: Y=1)

② CPU Hotplug Framework requests to take CPU down to PSCI handler. Then, the PSCI handler issues CPU_OFF request to Arm Trusted Firmware via SMC instruction.

③ CPU which requested CPU_OFF is turned off in secure world.

4.1.2 CPU Online

The following figure shows the processing flow of CPU Online.

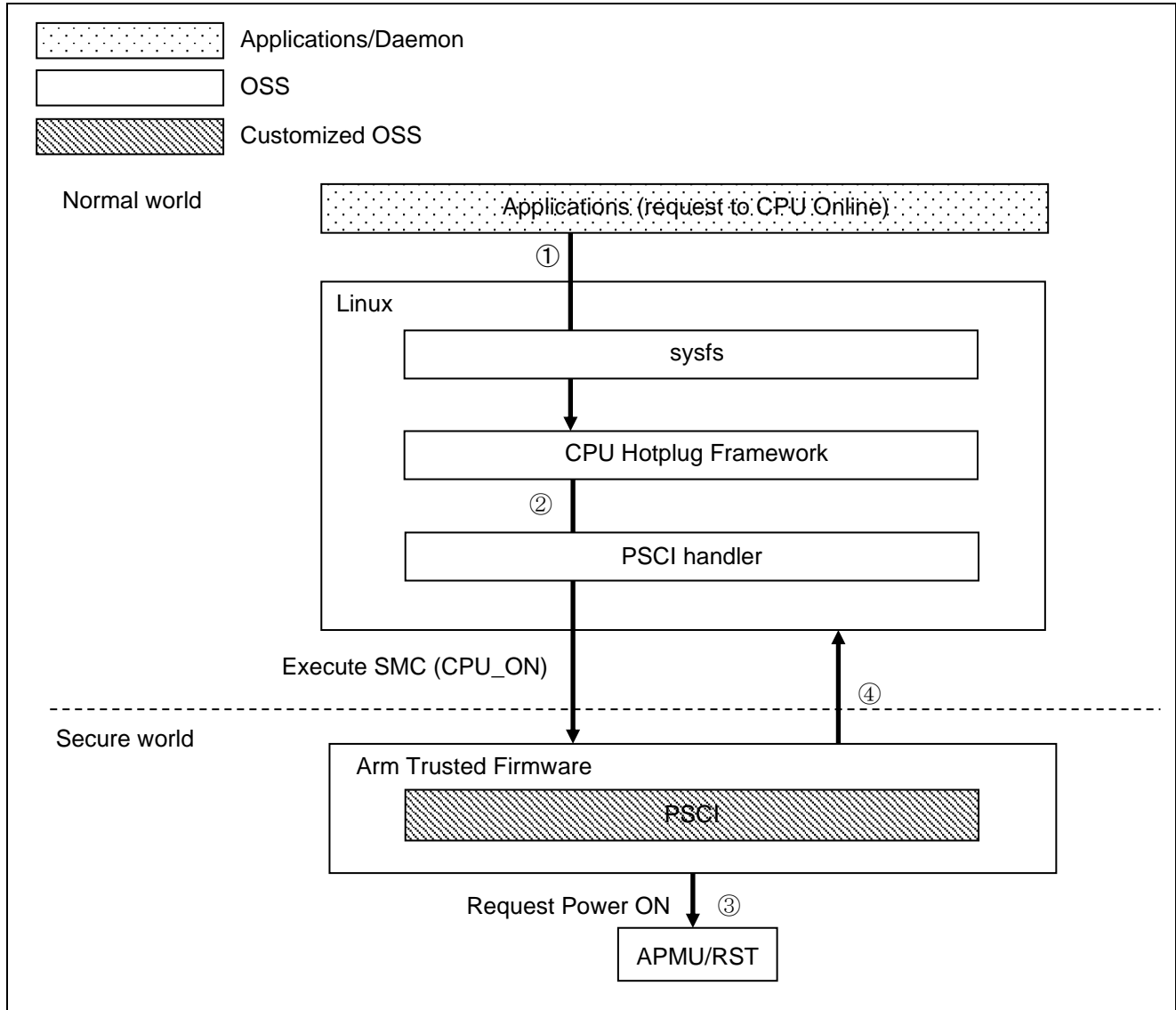


Figure 4-2 Processing flow of CPU Online

- ① Applications to work on Master CPU request CPU Online via sysfs.
echo 1 > /sys/devices/system/cpu/cpuY/online (R-Car H3: Y = 1/2/3/4/5/6/7; R-Car M3: Y = 1/2/3/4/5; R-Car M3N: Y = 1; R-Car E3: Y=1)
- ② CPU Hotplug Framework requests to take CPU up to PSCI handler. Then, the PSCI handler issues CPU_ON request to Arm Trusted Firmware via SMC instruction.
- ③ Slave CPU is turned on in secure world.
- ④ Slave CPU jumps to entry point and returns Linux.

4.2 CPU Idle

CPU Idle is Linux function to perform Core Standby in accordance with time until next process.

The following table shows CPU Idle configuration.

Table 4-1 CPU Idle configuration

CPU Idle configuration	CA57		CA53	
	Sleep mode	Core Standby mode	Sleep mode	Core Standby mode
Enable	Supported	Supported	Supported	Supported
Disable	Supported	Not supported	Supported	Not supported

The following table shows CPU Idle state.





Table 4-2 CPU Idle state

State	Description	State	
		Power domain	Clock
Sleep mode	CPU is put in sleep state.	ON	OFF
Core Standby mode	CPU is put in shutdown state.	OFF	OFF

In CPU Idle state, it has trade-off between Wake-up time and power consumption as showed it in Table 4-3.

For example, Core Standby state could reduce more power consumption, but it increases wake-up time

Table 4-3 Trade-off between Wake-up time and power consumption

	State	
	Sleep mode	Core Standby mode
Wake-up time	Low 	High 
Power consumption	High 	Low 

These states are chosen suitably according to current situation of System.

“Menu” governor is provided by Linux to choose the states. "Menu" takes predictive algorithm and calculates predicted latency time until next process. The suitable State is chosen according to calculated predicted idle time and the following parameters. “Menu” governor is default governor in current BSP, also BSP is only evaluated on this governor.

Table 4-4 Parameters to choose CPU Idle states

Parameters	Description
exit-latency	The latency time for moving from Idle state to Working state. Therefore, there is possibility to become less performance in case predicted idle time is less than exit-latency. Thus, the following formula must meet for Menu algorithm. $exit-latency \times multiplier \leq predicted\ idle\ time$
entry-latency	The latency required to enter the idle state. Preparation phase before committing the hardware to idle mode like cache flushing.
min-residency	The minimal residency time which should be stayed in Idle state. The process moving from Idle state to Working wastes a certain power consumption. If predicted idle time is less than min-residency, power consumption becomes bigger. Thus, the following formula must meet for Menu algorithm. $min-residency \leq predicted\ idle\ time$

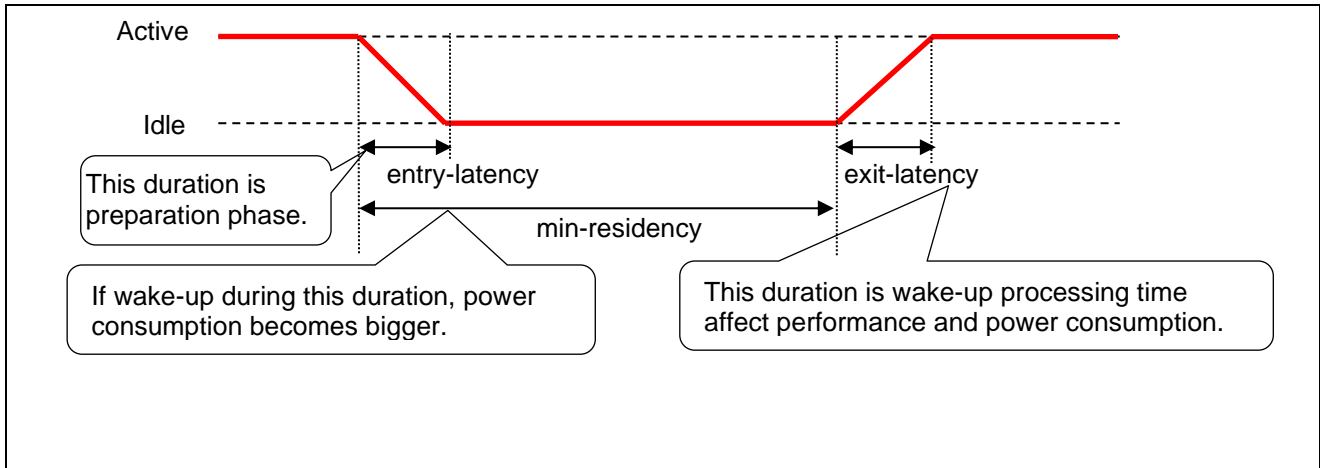


Figure 4-3 Parameters to choose CPU Idle states

The following figure shows the processing flow of CPU Idle.

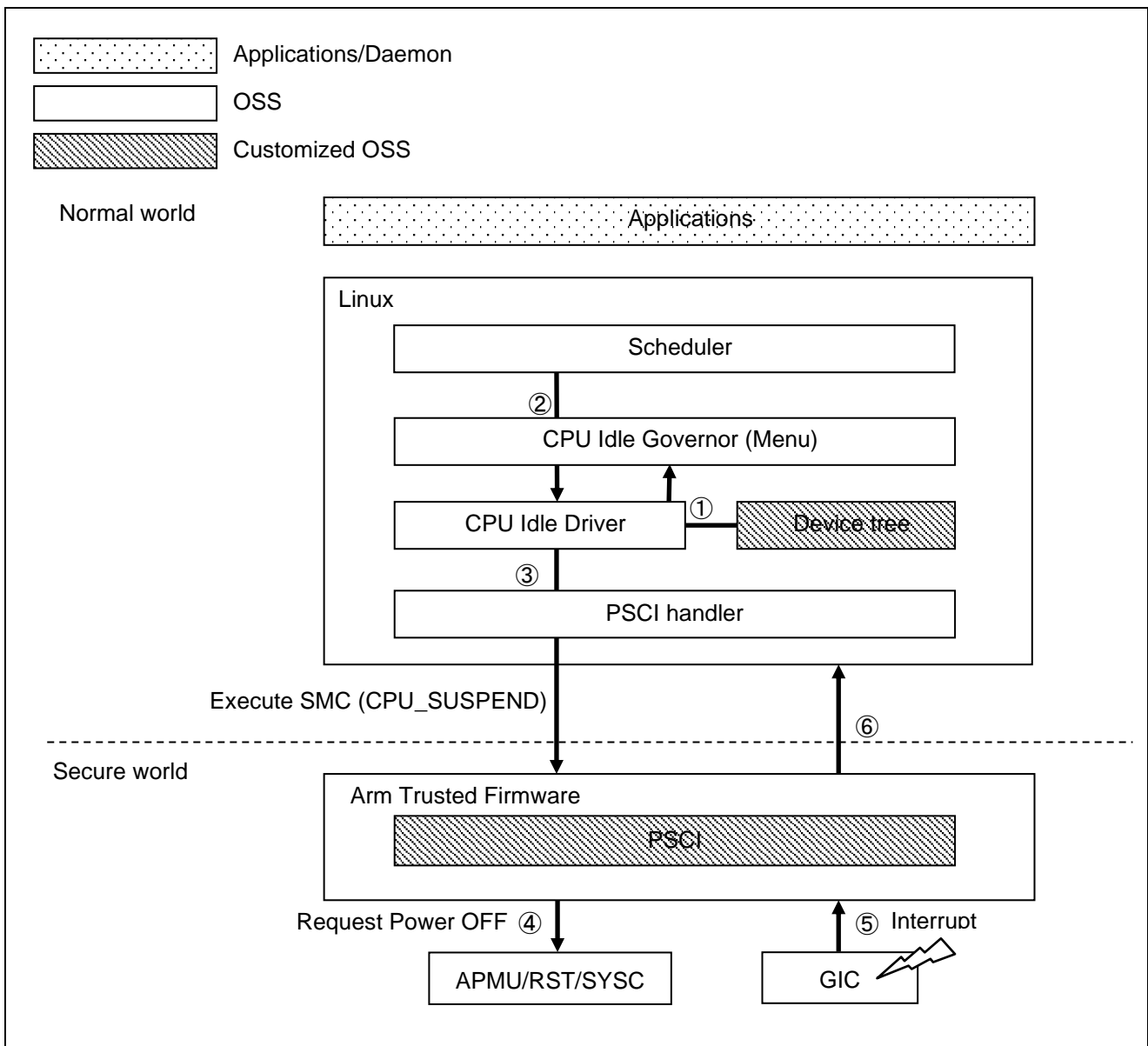


Figure 4-4 Processing flow of CPU Idle

- ① When Linux boots up (initialize), CPU Idle Driver gets parameters and CPU Idle states which defined in device tree. Also, it sends the parameters to CPU Idle Governor (Menu).
- ② CPU Idle Governor (Menu) receives request of CPU Idle according to current situation of System and chooses suitably CPU Idle state.
If Sleep mode state is selected, CPU Idle Driver issues WFI to CPU. If Core Standby state is selected, it advances to next step.
- ③ CPU Idle Driver requests to enter low power idle state. Then, the PSCI handler issues CPU_SUSPEND request to Arm Trusted Firmware.
- ④ CPU which requested CPU_SUSPEND is turned off in secure world.
- ⑤ When GIC receives the interrupt, Loader is booted up and kicks Arm Trusted Firmware. Then, Arm Trusted Firmware sets MMU/Cache.
- ⑥ CPU jumps to entry point and returns Linux.

4.3 CPU Freq

CPU Freq (DVFS) is Linux function to dynamically change voltage and frequency of CPU.

Also, it has AVS which adjusts voltage of CPU according to finished process of SoC.

About the change, CA57 supports change of voltage and frequency, CA53 supports change of frequency only.

4.3.1 DVFS

CPU Freq has 6 operating modes which are called CPU Freq Governor. The voltage and frequency of CPU are changed according to category of CPU Freq Governor.

The following table shows the CPU Freq Governor operating modes and default mode in current BSP.

Table 4-5 CPU Freq Governor operating mode

Governor	Description
Performance	The frequency is set max.
Powersave	The frequency is set min.
Ondemand	If CPU load is bigger than 95%, the frequency is set max. If CPU load is equal to or less than 95%, the frequency is set based on CPU load.
Conservative	If CPU load is bigger than 80%, the frequency is set one level higher than current frequency. If CPU load is equal to or less than 20%, the frequency is set one level lower than current frequency.
Userspace	It sets frequency which is defined by user.
Schedutil (Default)	Schedutil governor is driven by scheduler. It uses scheduler-provided CPU utilization information as input for making its decisions.

In R-Car Series, 3rd Generation BSP, from Linux kernel v5.4 onwards, “schedutil” is set as default governor. To make system works with the highest performance, the governor should be changed to “performance”, refer to section 5.3.2 CPU Freq operation for changing method. For Linux kernel v4.14.75 or older, “performance” is set as default governor.

The following figure shows the processing flow of DVFS as an example of Ondemand/Conservative governor.

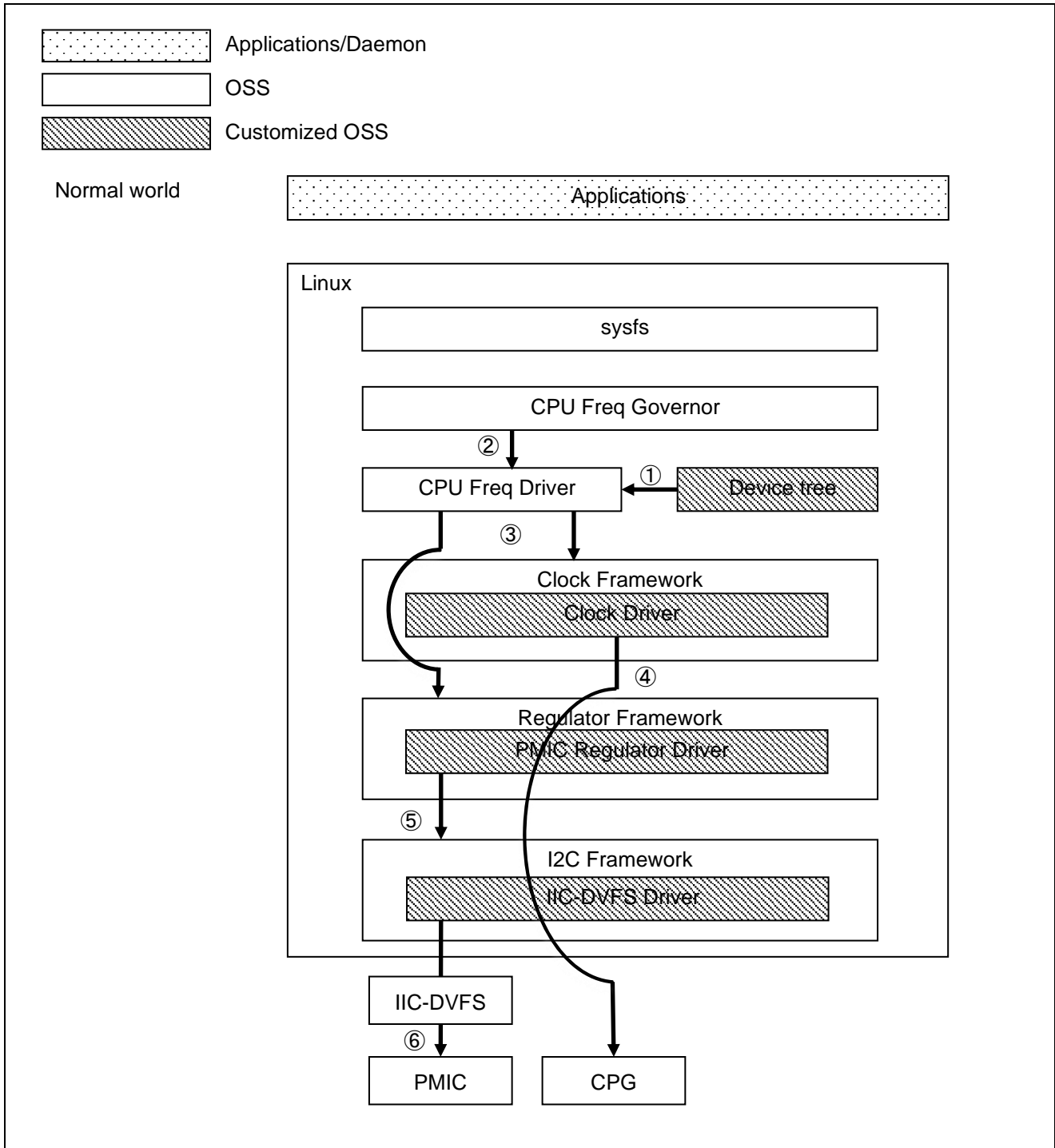


Figure 4-5 Processing flow of DVFS by CPU Freq

- ① When Linux boots up (initialize), CPU Freq Driver gets setting values of voltage and frequency which defined in device tree. Also, it sends them to CPU Freq Governor.
- ② CPU Freq Governor chooses the setting value of voltage and frequency of CPU according to Governor operating mode (*1). Also, it notifies the requested value of voltage and frequency to CPU Freq Driver.
(*1) Governor requests the setting value of frequency in accordance with the highest CPU load in CPU cluster.
- ③ CPU Freq Driver notifies the requested value of frequency to Clock Framework, notifies the requested voltage to Regulator Framework. Figure 4-6 shows overview flow which decides frequency. And, Table 4-6, Table 4-7, and Table 4-8 shows setting values of frequency and voltage which are defined by device tree.
- ④ Clock Driver changes the requested frequency by CPG control.
- ⑤ PMIC Regulator Driver notifies the requested voltage to I2C Framework (only for CA57 change.)

- ⑥ The requested voltage is changed by PMIC control via IIC-DVFS (only for CA57 change.)

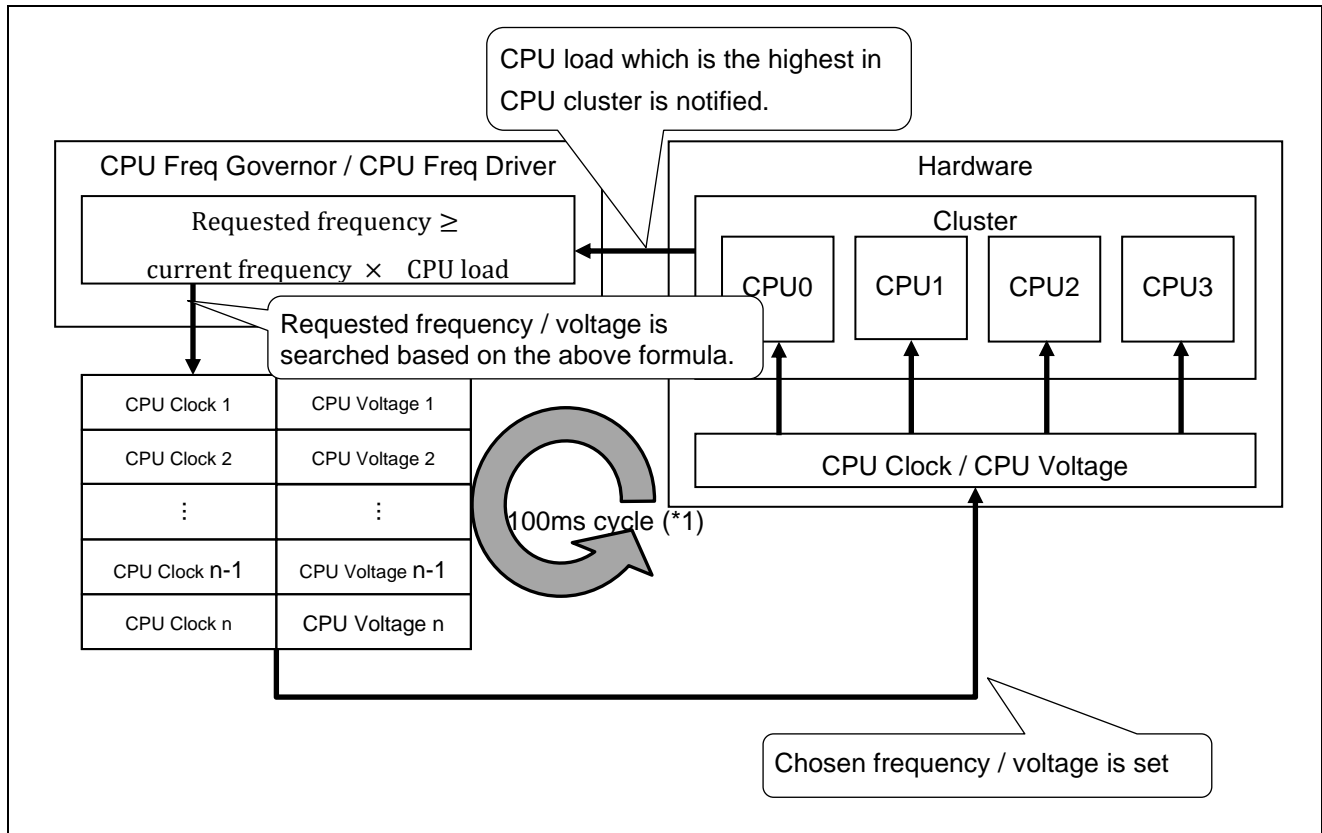


Figure 4-6 Overview flow which decides CPU frequency / voltage

(*1) 100ms is the sample value to execute governor work. It is applied to Conservative and Ondemand governors. The actual value can be adjusted based on target frequency (for Ondemand governor) or changing via cpufreq sysfs (for Conservative governor).

4.3.2 Frequency and AVS support

CA57 frequency of R-Car H3e-2G/M3e-2G supports 2.0GHz (16.6MHz x 1/1 x 120 x 32/32), but no supports AVS as R-Car Series, 3rd Generation Expansion H/W.

Table 4-6 and Table 4-7 show frequency and AVS patterns of CA57 that are applied for R-Car H3, R-Car M3; and Table 4-8 show frequency and voltage pattern of CA57 for R-Car M3N. AVS chooses voltage according to finished process of SoC. AVS is read when value is set to eFuse.

Table 4-6 CA57 frequency and AVS table of R-Car H3

Operating mode	CA57 Freq	AVS0	AVS1	AVS2	AVS3	AVS4	AVS5	AVS6	AVS7
Boost mode	1.7GHz (16.6MHz x 1/1 x 100 x 32/32)	0.96 V	0.95 V	0.93 V	0.91 V	0.89 V	0.88 V	0.87 V	0.86 V
	1.6GHz (16.6MHz x 1/1 x 96 x 32/32)	0.90 V	0.89 V	0.88 V	0.87 V	0.86 V	0.85 V	0.84 V	0.83 V
Normal mode	1.5GHz (16.6MHz x 1/1 x 90 x 32/32)	0.83 V	0.82 V	0.81 V	0.80 V	0.79 V	0.78 V	0.77 V	0.76 V
	1.0GHz (16.6MHz x 1/1 x 90 x 21/32)								
	0.5GHz (16.6MHz x 1/1 x 90 x 11/32)								

Table 4-7 CA57 frequency and AVS table of R-Car M3

Operating mode	CA57 Freq	AVS0	AVS1	AVS2	AVS3	AVS4	AVS5	AVS6	AVS7
Boost mode	1.8GHz (16.6MHz x 1/1 x 108 x 32/32)	0.96 V	0.95 V	0.93 V	0.91 V	0.89 V	0.88 V	0.87 V	0.86 V
	1.7GHz (16.6MHz x 1/1 x 100 x 32/32)	0.90 V	0.89 V	0.88 V	0.87 V	0.86 V	0.85 V	0.84 V	0.83 V
	1.6GHz (16.6MHz x 1/1 x 96 x 32/32)	0.90 V	0.89 V	0.88 V	0.87 V	0.86 V	0.85 V	0.84 V	0.83 V
Normal mode	1.5GHz (16.6MHz x 1/1 x 90 x 32/32)	0.83 V	0.82 V	0.81 V	0.80 V	0.79 V	0.78 V	0.77 V	0.76 V
	1.0GHz (16.6MHz x 1/1 x 90 x 21/32)								
	0.5GHz (16.6MHz x 1/1 x 90 x 11/32)								

Table 4-8 CA57 frequency and voltage table of R-Car M3N

Operating mode	CA57 Freq	Voltage
Boost mode	1.8GHz (16.6MHz x 1/1 x 108 x 32/32)	0.96 V
	1.7GHz (16.6MHz x 1/1 x 100 x 32/32)	0.90 V
	1.6GHz (16.6MHz x 1/1 x 96 x 32/32)	0.90 V
Normal mode	1.5GHz (16.6MHz x 1/1 x 90 x 32/32)	0.83 V
	1.0GHz (16.6MHz x 1/1 x 90 x 21/32)	
	0.5GHz (16.6MHz x 1/1 x 90 x 11/32)	

Table 4-9, Table 4-10 and Table 4-11 show the frequency pattern of CA53 that are applied for R-Car H3, R-Car M3, and R-Car E3.

Table 4-9 CA53 frequency table of R-Car H3

Operating mode	CA53 Freq	Voltage
Normal mode	1.2GHz (16.6MHz x 1/1 x 72 x 32/32)	Fixed value
	1.0GHz (16.6MHz x 1/1 x 72 x 27/32)	
	0.8GHz (16.6MHz x 1/1 x 72 x 21/32)	

Table 4-10 CA53 frequency table of R-Car M3

Operating mode	CA53 Freq	Voltage
Boost mode	1.3GHz (16.6MHz x 1/1 x 78 x 32/32)	Fixed value
Normal mode	1.2GHz (16.6MHz x 1/1 x 72 x 32/32)	
	1.0GHz (16.6MHz x 1/1 x 72 x 27/32)	
	0.8GHz (16.6MHz x 1/1 x 72 x 21/32)	

Table 4-11 CA53 frequency table of R-Car E3

Operating mode	CA53 Freq	Voltage
Normal mode	1.2GHz (48.0MHz x 1/2 x 1/2 x 32/32)	Fixed value
	1.0GHz (48.0MHz x 1/2 x 1/2 x 27/32)	
	0.8GHz (48.0MHz x 1/2 x 1/2 x 21/32)	

The following figure shows the flow of AVS.

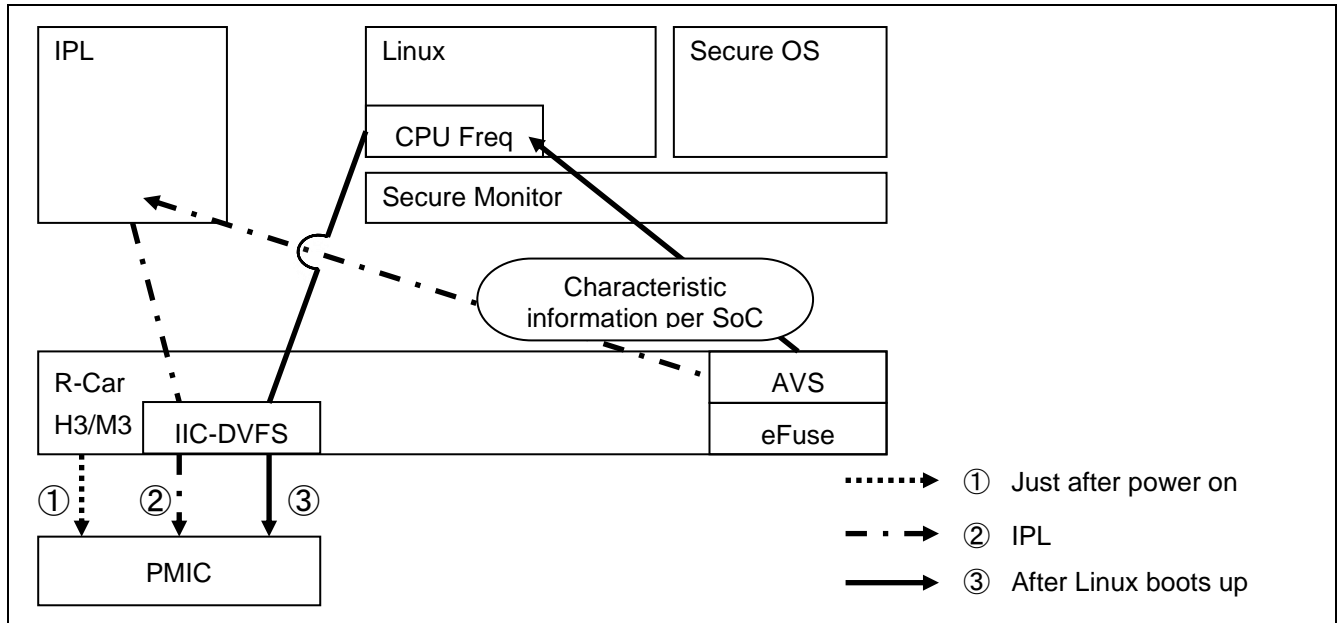


Figure 4-7 Processing flow of AVS

- ① eFuse contains the value of AVS table for each specific SoC, and that value is read via AVS module. Base on that value, target AVS table as described in DT will be selected. Each AVS table contains the frequency-voltage pairs for each supported CPU frequency (from 500MHz to 1.7GHz on R-Car H3 or to 1.8GHz on R-Car M3).
- ② Just after power on, frequency depends on MD pins setting and voltage is initial value of PMIC.
Frequency of CA57=1.5GHz, Frequency GPU=600MHz, Voltage =AVS DVFS VID_x (x=AVS1/AVS2 terminal output value)
- ③ IPL sets one frequency-voltage pair of target AVS pattern according to characteristic information per SoC.
Frequency of CA57=1.5GHz, Frequency of GPU=600MHz, Voltage=voltage value in pair with 1.5GHz.
- ④ After Linux boots up, AVS pattern is chosen according to characteristic information per SoC. Then, CPU Freq dynamically changes operating mode (frequency and voltage) from AVS pattern chosen in accordance with CPU load.
Frequency of CA57=CPU Freq control, Frequency of GPU=600MHz, Voltage=CPU Freq control

Table 4-12 Frequency/Voltage setting in each software

		CA57 Frequency	GPU Frequency	Voltage
①	Just after power on	1.5GHz	600MHz	AVS DVFS VID _x (x=AVS1/AVS2terminal output value)
②	IPL	1.5GHz	600MHz	voltage value in pair with 1.5GHz
③	After Linux boots up	DVFS	600MHz	DVFS

4.4 Runtime PM

Runtime PM is Linux function to control clock supply and power domain for IP modules. Also, it is controlled in consideration of the configuration of the power domain.

The following figure shows the processing flow of Runtime PM.

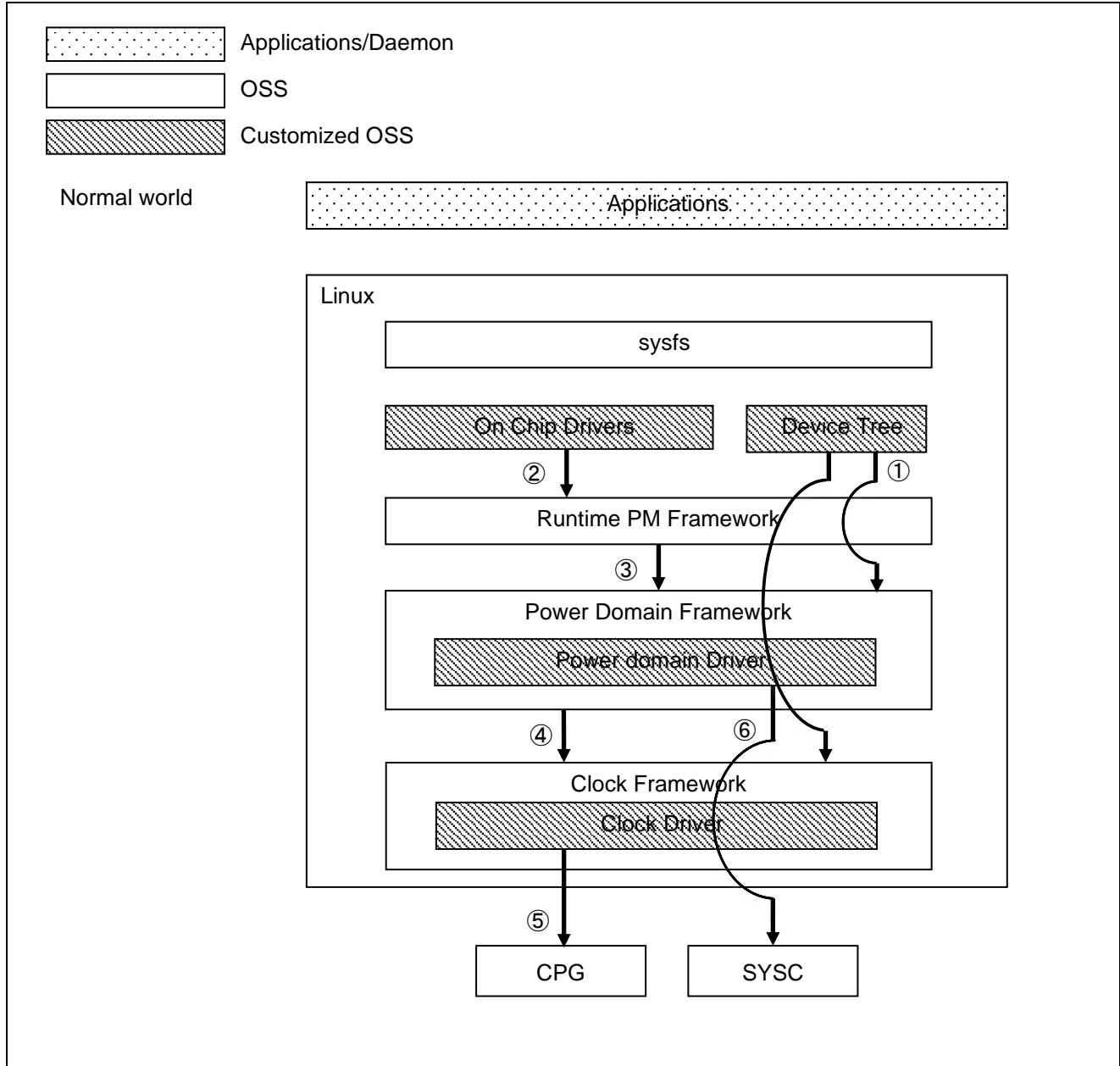


Figure 4-8 Processing flow of Runtime PM

- ① When Linux boots up (initialize), Clock Framework gets settings of clock which defined in device tree. Also, Power Domain Framework gets settings of power domain which are defined in device tree.
- ② On Chip Drivers notify status of module which controls its own to Runtime PM Framework via Runtime PM API.
- ③ Runtime PM Framework notifies status of module to Power Domain Framework.
- ④ Power domain Framework requests the clock supply or stop of the module to the Clock Framework.
- ⑤ Clock Framework enables or stops the clock of module by CPG control via Clock Driver.
- ⑥ Power domain Framework turns on or off the power domain by SYSC control via Power Domain

The following figure shows the processing of Runtime PM as example.

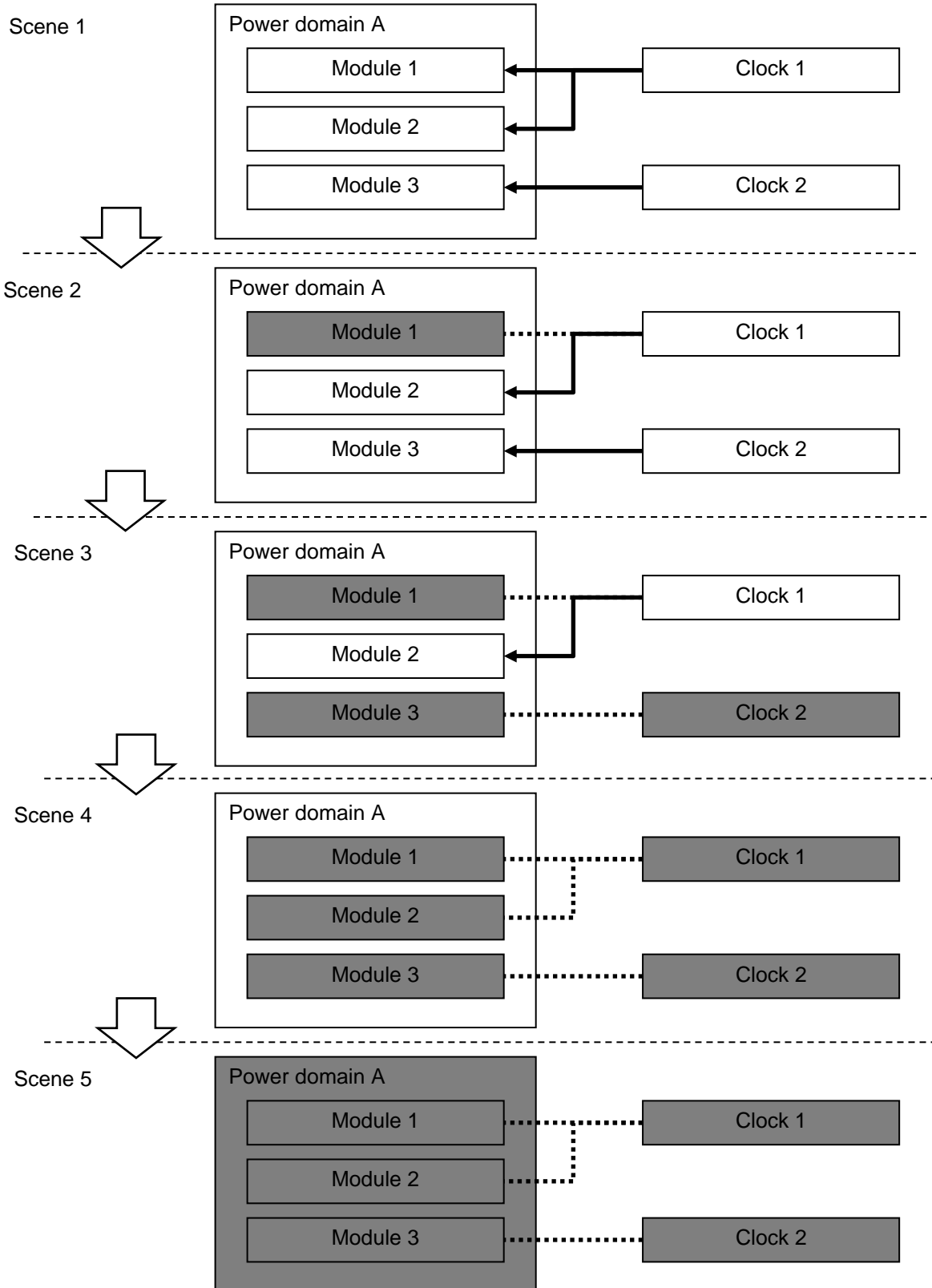


Figure 4-9 Processing of Runtime PM as example

- ① Module 1, Module 2 and Module 3 are belonged in Power domain A. And Clock 1 is supplied to Module 1 and Module 2. (Scene 1)

- ② If Module 1 does not need clock, Clock Framework turns off clock to Module 1. However, root of Clock 1 is not turned off, because Clock 1 is supplied to Module2. (Scene 2)
- ③ If Module 3 does not need clock, Clock Framework turns off clock to Module 3. Then, root of Clock 2 is turned off because all modules which use Clock 2 are turned off. (Scene 3)
- ④ If Module 2 does not need clock, Clock Framework turns off clock to Module 2. Then, root of Clock 1 is turned off because all modules which use Clock 1 are turned off. (Scene 4)
- ⑤ Power domain A is turned off, because all clocks of modules which belong to Power domain A are turned off. (Scene 5)

Then, the following figure shows the processing of considering the configuration of power domain as example.

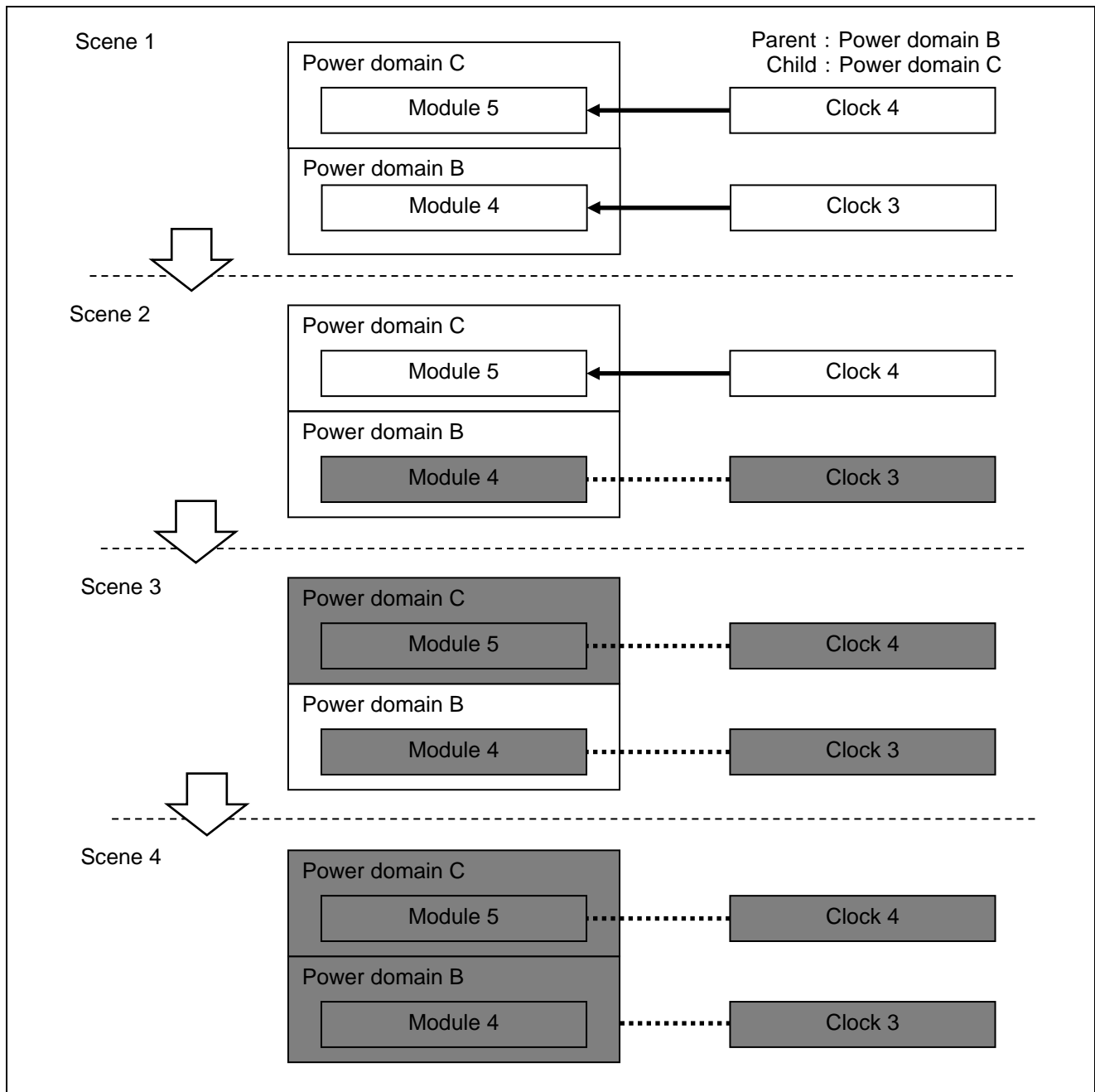


Figure 4-10 The processing of considering the configuration of power domain as example

- ① Module 4 belongs in Power domain B, Module 5 belongs Power domain C. Also, the Power domain has a Parent-Child relationship, Power domain B is Parent, Power domain C is Child. (Scene 1)

- ② If Module 4 does not need clock, Clock Framework turns off clock to Module 4. Power domain B is not turned off, because Power domain C is ON. (Scene 2)
- ③ If Module 5 does not need clock, Clock Framework turns off clock to Module 5. And Power domain C is turned off, because all clocks of modules which belong to Power domain C are turned off. (Scene 3)
- ④ Power domain B is also turned off, because Power domain C is OFF. (Scene 4)

4.5 System Suspend to RAM

System Suspend to RAM is a function to halt all of the power supplies except for the backup power supply by saving the state of system to RAM for backup to reduce power consumption. Also, it's faster than cold boot because program load and initialization of device driver are skipped. It supports only self-refresh of whole SDRAM in current BSP.

In addition, the backup targets are devices of Salvator-X/XS and Ebisu standard.

The following figure shows HW configuration of System Suspend to RAM.

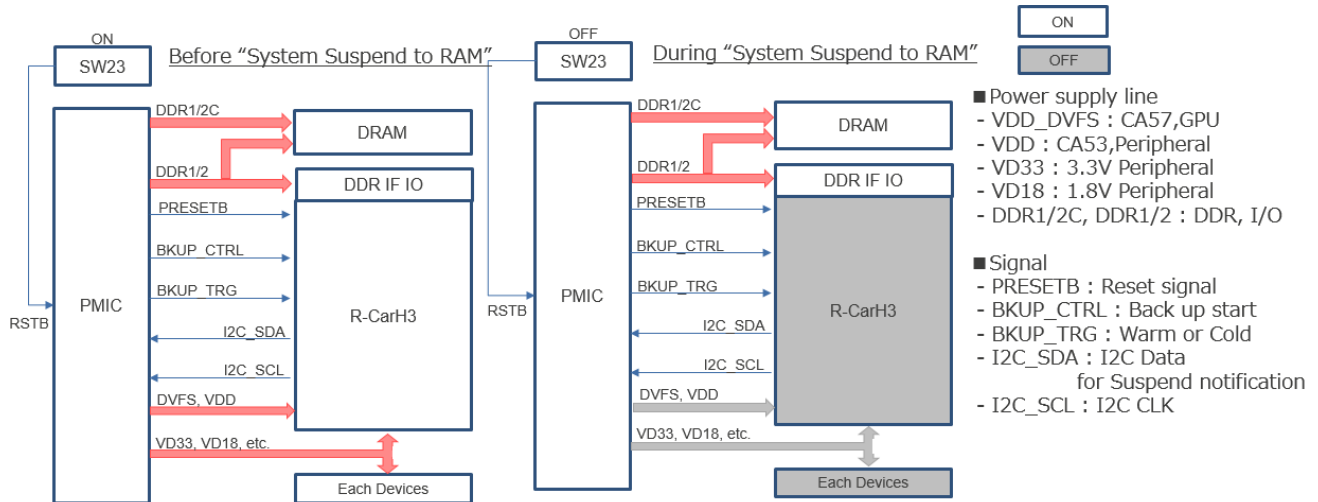


Figure 4-11 HW configuration of System Suspend to RAM

The following figure shows the processing flow of System Suspend to RAM.

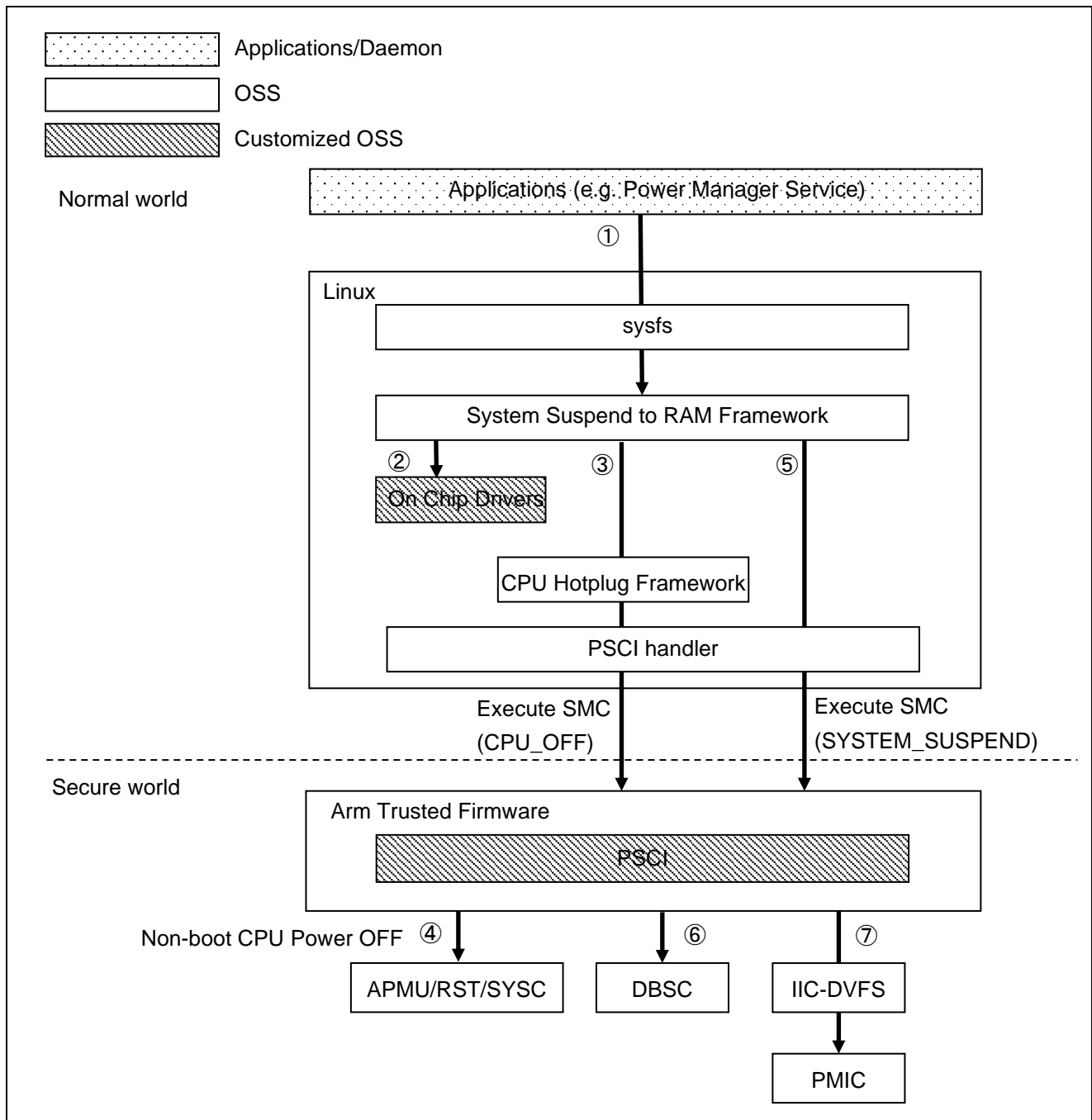


Figure 4-12 Processing flow of System Suspend to RAM

- ① Application requests System Suspend to RAM using i2c-tools and via sysfs. The i2c-tools command is needed to enable DDR backup mode in PMIC.
i2cset -f -y 7 0x30 0x20 0x0F
echo deep > /sys/power/mem_sleep
echo mem > /sys/power/state
- ② System Suspend to RAM Framework notifies device driver (On Chip Drivers) of moving to System Suspend to RAM. Then, device drivers back up the needed data to SDRAM.
- ③ System Suspend to RAM Framework requests to take Non-boot CPU down to PSCI handler via CPU Hotplug Framework. Then, the PSCI handler issues CPU_OFF request to Arm Trusted Firmware via SMC instruction.
- ④ Non-boot CPU is turned off in secure world. It's repeated only the number of Non-boot CPU.
- ⑤ When Non-boot CPU is all OFF, System Suspend to RAM Framework requests SYSTEM_SUSPEND to Arm Trusted Firmware.
- ⑥ Arm Trusted Firmware sets self-refresh of SDRAM by DBSC control.
- ⑦ Boot-CPU notifies via IIC-DVFS to PMIC just prior to OFF. PMIC halts all of the power supply except for the backup power supply.

Note: For more detail and notice about System Suspend to RAM function, please refer to section 7.1 Design Note for System Suspend to RAM support.

Note: In case of system resumes and CPU Freq is being run with “userspace” governor, please set the frequency after resume same as frequency before suspend. (This is based on current implementation of upstream kernel.)

4.6 Thermal Management

Thermal management is to control SoC temperature using power management function. Thermal management has the following three functions.

Table 4-13 Thermal management functions

Function	Description
IPA	IPA is a Linux function to control DVFS/DFS for CPU(CA57/CA53) based on temperature(T_j) which is read from thermal channel. <u>Note:</u> Temperature is read from THS3 on R-Car H3/M3/M3N; and from THS1 on R-Car E3. ⁽¹⁾
EMS	EMS is a function to scale CPU frequency down to minimum value and turn off secondary core for CPU in condition of high temperature(T_j) which is read from THS (THS1/2/3 on R-Car H3/M3/M3N and THS1 on R-Car E3). ⁽¹⁾
System shutdown	System shutdown is a function to turn off system before SoC temperature reaches a limit.

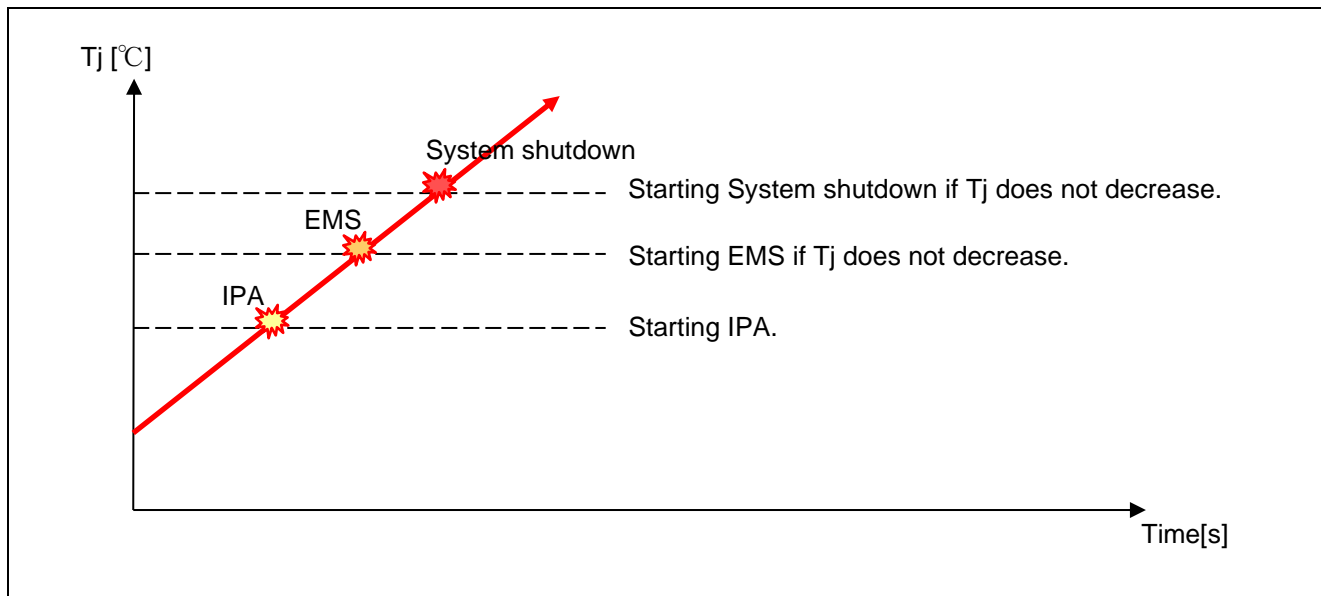


Figure 4-13 Summary of thermal management starting

The following figure shows the software configuration of thermal management system.

¹ - On R-Car H3/M3/M3N, thermal module has three channels (THS1/THS2/THS3). On R-Car E3, thermal module has only one channel (THS1).

- For temperature to control IPA, it is got from THS3 channel on R-Car H3/M3/M3N; and is got from THS1 on R-Car E3.

- On R-Car E3, IPA controls only DFS (so, it does not control PMIC). And for EMS and System Shutdown, they read temperature from THS1 only.

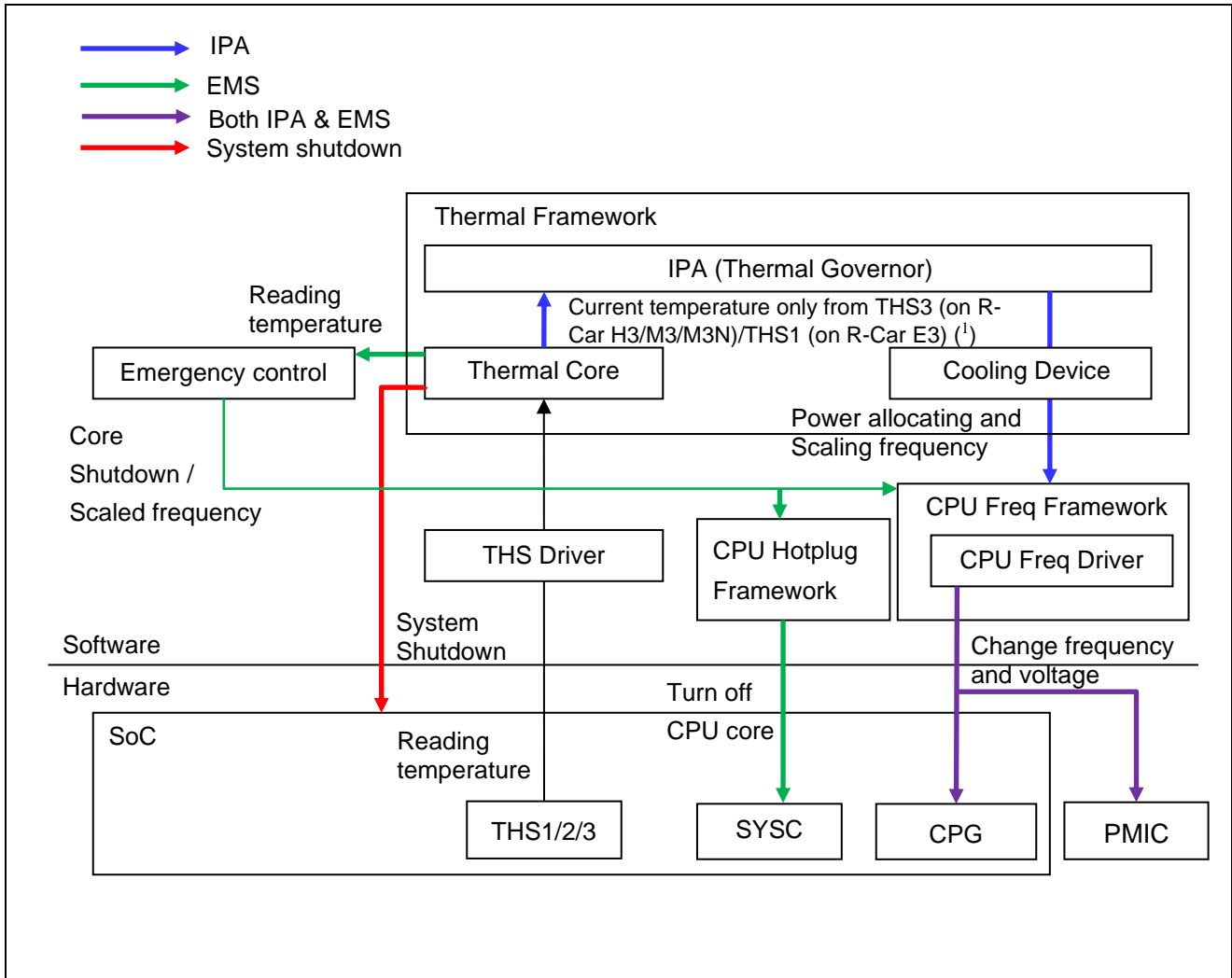


Figure 4-14 Block diagram of thermal management

4.6.1 IPA

IPA is a Linux function to control DVFS ⁽¹⁾ for CPU based on temperature(Tj) which is read from THS3 on R-Car H3/M3/M3N or THS1 on R-Car E3. The following figure shows the processing flow of IPA.

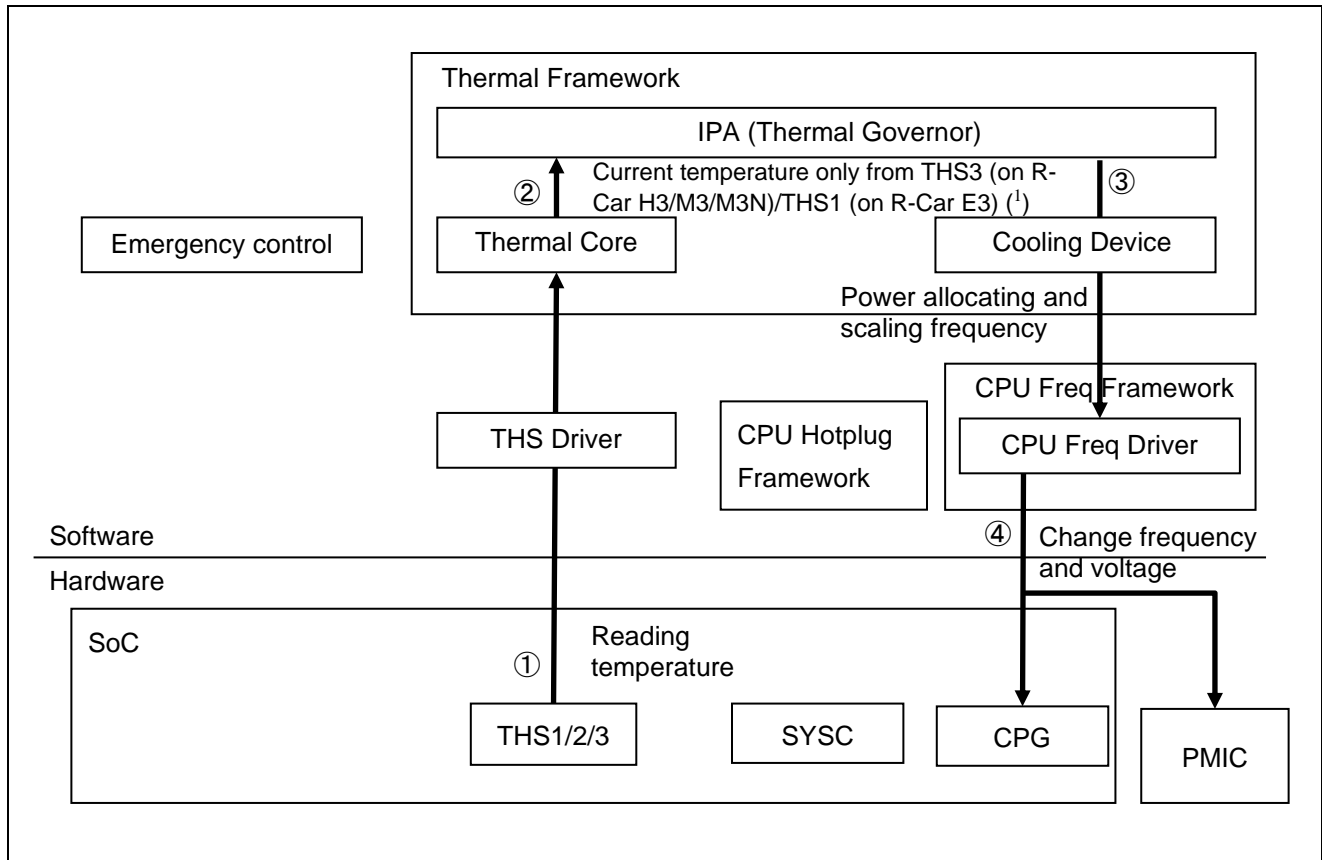


Figure 4-15 Processing flow of IPA

- ① THS Driver reads Tj from THS1/THS2/THS3. Then, they are input to Thermal Core.
- ② Tj from THS3 (on R-Car H3/M3/M3N) or THS1 (on R-Car E3) is sent to IPA. ⁽¹⁾
- ③ If Tj at step 2 is over 90 degree Celsius, IPA calculates power which can be allocated based on Tj. Then, IPA scales available frequency of CPU.
- ④ CPU Freq Driver sets the available frequency which is calculated from IPA.

4.6.2 EMS

EMS is a function to turn off core for CPU based on temperature(Tj) which is read from THS(THS1/2/3) (¹).

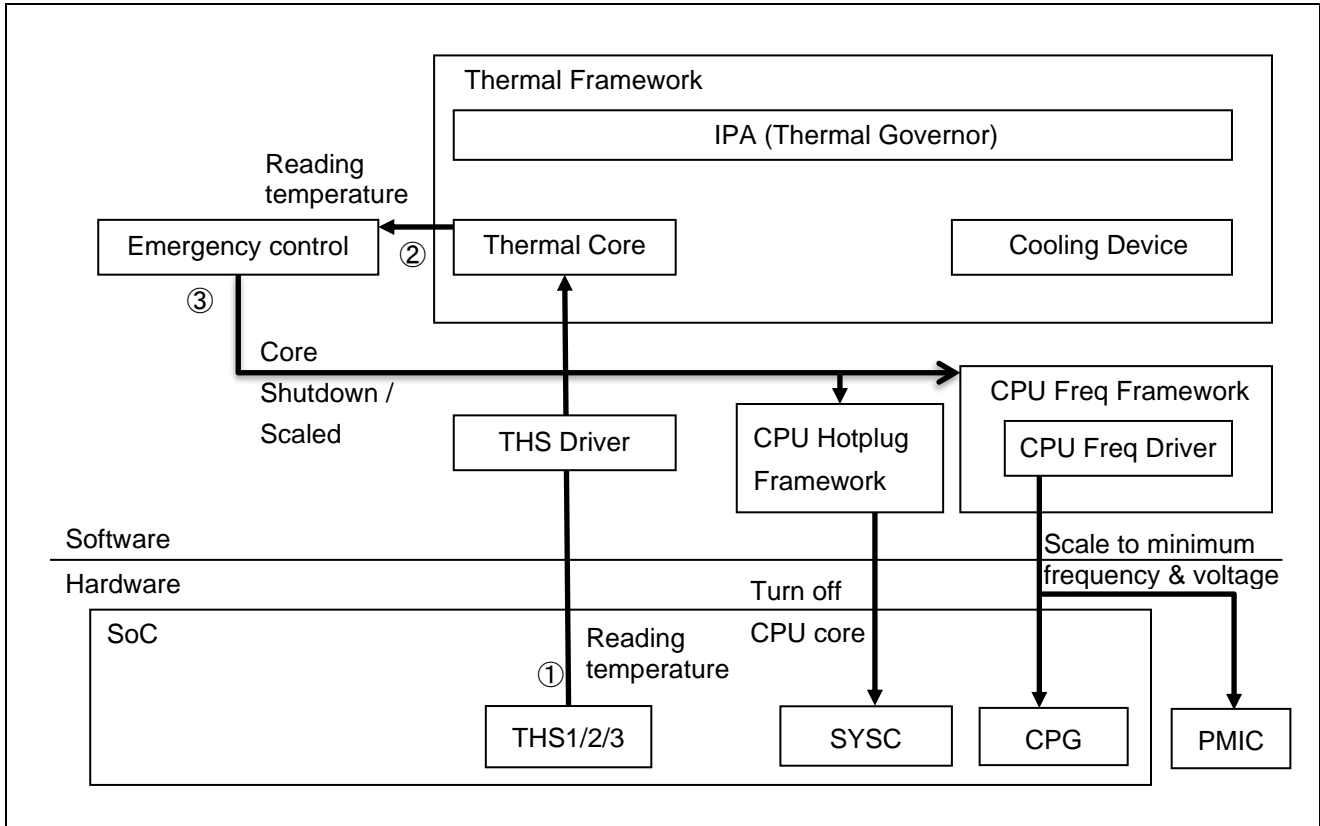


Figure 4-16 Processing flow of EMS

- ① THS Driver reads Tj from THS1/THS2/THS3. Then, they are input to Thermal Core.
- ② Emergency Control gets Tj from Thermal Core.
- ③ If any Tj from THS1/THS2/THS3 is over 110 degree Celsius, Emergency Control scales CPU frequency down to minimum frequency value and turns off CPU core.

4.6.3 System Shutdown

System Shutdown is function to turn off system before SoC temperature reaches a limit.

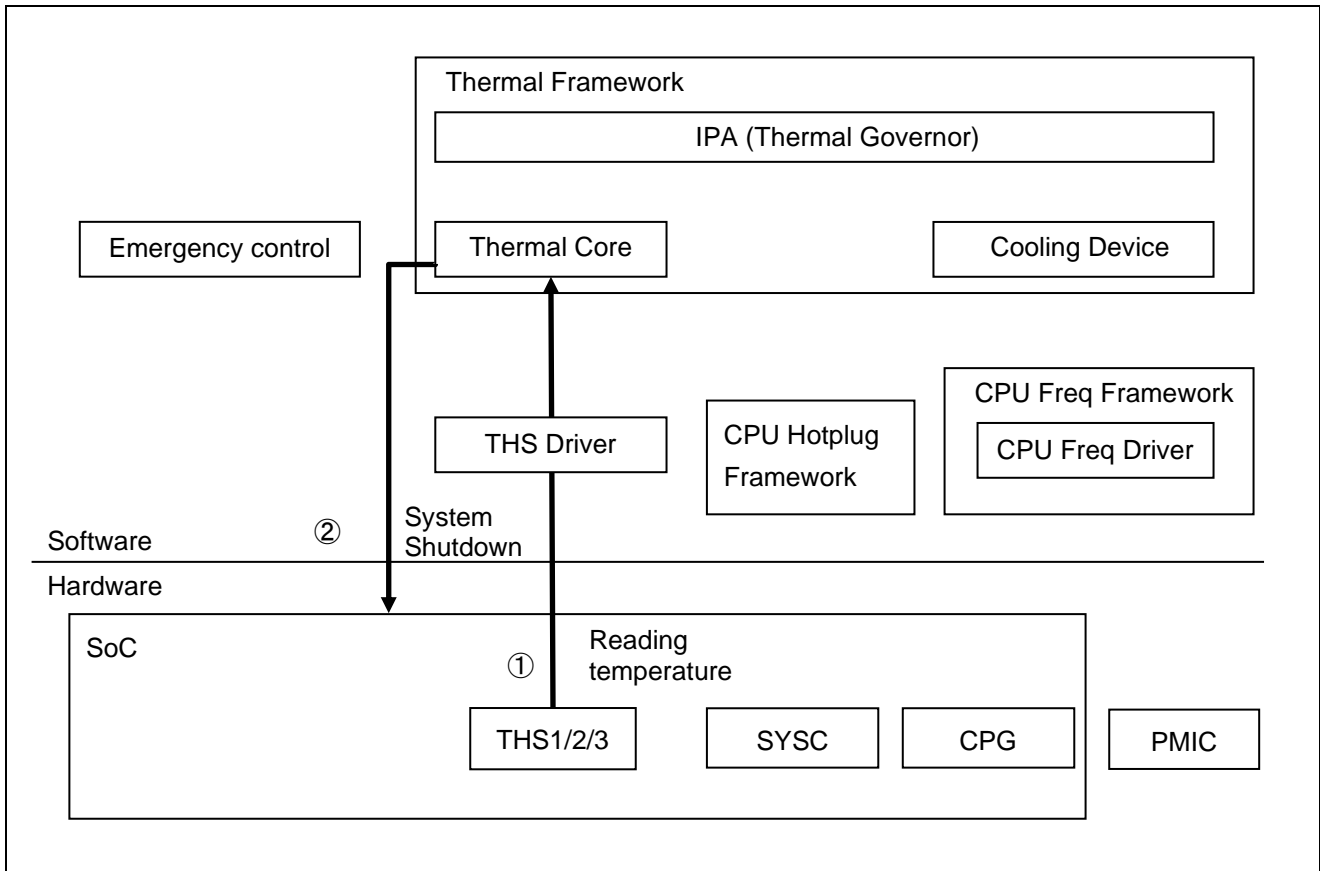


Figure 4-17 Processing flow of System shutdown

- ① THS Driver reads Tj from THS1/THS2/THS3 (1). Then, they are input to Thermal Core.
- ② If any Tj from THS1/THS2/THS3 is over 120 degree Celsius, Thermal Core shuts down the system.

5. External Interface

The following table shows system information of power management.

Table 5-1 System information of power management

Function	Function Information
CPU Hotplug	/sys/devices/system/cpu/online
	/sys/devices/system/cpu/offline
	/sys/devices/system/cpu/cpuY/online
CPU Idle	/sys/devices/system/cpu/cpuX/cpuidle/*
CPU Freq	/sys/devices/system/cpu/cpufreq/boost
	/sys/devices/system/cpu/cpuX/cpufreq/*
System Suspend to RAM	/sys/power/state

(R-Car H3: X = 0/1/2/3/4/5/6/7; R-Car M3: X = 0/1/2/3/4/5; R-Car M3N: X = 0/1; R-Car E3: X = 0/1)

(R-Car H3: Y = 1/2/3/4/5/6/7; R-Car M3: Y = 1/2/3/4/5; R-Car M3N: Y = 1; R-Car E3: X = 1)

Detailed explanation of system information is described in Linux documentation. Also, the following table shows Linux document of power management.

Table 5-2 Linux document of power management

Function	Document Information
CPU Hotplug	Documentation/cputopology.txt
	Documentation/cpu-hotplug.txt
CPU Idle	Documentation/cpuidle/sysfs.txt
CPU Freq	Documentation/cpu-freq/user-guide.txt
System Suspend to RAM	Documentation/power/states.txt

5.1 CPU Hotplug

5.1.1 CPU Hotplug definition

CPU Hotplug doesn't need a special definition.

5.1.2 CPU Hotplug operation

CPU Hotplug can be operated via sysfs (/sys/devices/system/cpu/cpuY/online). (R-Car H3: Y = 1/2/3/4/5/6/7; R-Car M3: Y = 1/2/3/4/5; R-Car M3N: Y = 1; R-Car E3: Y = 1)

The following figure shows example of operation for CPU Hotplug.

```
/* cpu1 to offline */  
$ echo 0 > /sys/devices/system/cpu/cpu1/online  
  
/* cpu1 to online */  
$ echo 1 > /sys/devices/system/cpu/cpu1/online
```

Figure 5-1 Example of operation for CPU Hotplug

5.2 CPU Idle

5.2.1 CPU Idle definition

CPU Idle function can be used by defining idle-states node in the device tree, and registering the CPU_SLEEP_0/1 to each CPU core.

The following figure shows example of definition for idle-state node

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

    a57_0: cpu@0 {
        compatible = "arm,cortex-a57", "arm,armv8";
        reg = <0x0>;
        device_type = "cpu";
        enable-method = "psci";
        cpu-idle-states = <&CPU_SLEEP_0>;
        ...
    };
    ...
    a53_0: cpu@100 {
        compatible = "arm,cortex-a53", "arm,armv8";
        reg = <0x100>;
        device_type = "cpu";
        enable-method = "psci";
        cpu-idle-states = <&CPU_SLEEP_1>;
        ...
    };
    ...
    idle-states {
        entry-method = "psci";

        CPU_SLEEP_0: cpu-sleep-0 {
            compatible = "arm,idle-state";
            arm,psci-suspend-param = <0x0010000>;
            local-timer-stop;
            entry-latency-us = <400>;
            exit-latency-us = <500>;
            min-residency-us = <4000>;
            status = "okay";
        };

        CPU_SLEEP_1: cpu-sleep-1 {
            compatible = "arm,idle-state";
            arm,psci-suspend-param = <0x0010000>;
            local-timer-stop;
            entry-latency-us = <700>;
            exit-latency-us = <700>;
            min-residency-us = <5000>;
            status = "okay";
        };
    };
};
```

Figure 5-2 Example of definition on device tree for idle-states node

In R-Car Series, 3rd Generation BSP kernel, these reference parameters (entry-latency-us, exit-latency-us and min-residency-us) are set for Salvator-X/XS and Ebisu board. It is necessary that these parameters are tuned in for other boards of customer environment.

5.2.2 CPU Idle operation

The operation of CPU Idle can be confirmed via sysfs (/sys/devices/system/cpu/cpuX/cpuidle/*).

In this chapter value of X and Y as follow:

X = 0/1/2/3/4/5/6/7 (on R-Car H3); X = 0/1/2/3/4/5 (on R-Car M3); X = 0/1 (on R-Car M3N and R-Car E3)

Y = 0 (for Sleep mode); 1 (for Core Standby mode)

Confirm number of times each state was entered :

The following figure shows example about confirming number of times each state was entered.

```
/* Check usage of CPU0 in state 1(for Core Standby mode) */
$ cat /sys/devices/system/cpu/cpu0/cpuidle/state1/usage

/* Check usage of CPU0 in state 0(for Sleep mode) */
$ cat /sys/devices/system/cpu/cpu0/cpuidle/state0/usage
```

Figure 5-3 Example about confirming CPU Idle parameters

Enable/disable CPU Idle at boot time by default:

Currently, the CPU Idle is enabled at boot time by default.

The following figure shows example for enabling/disabling CPU Idle function.

```
/* Enable CPU Idle configure in arch/arm64/configs/defconfig */
CONFIG_ARM_CPUIDLE=y

/* Disable CPU Idle configure in arch/arm64/configs/defconfig */
# CONFIG_ARM_CPUIDLE is not set
```

Figure 5-4 Example about enabling/disabling CPU Idle at boot time by default

Enable/disable CPU Idle at Runtime:

After CPU Idle is enabled at boot time, it can be enabled/disabled at runtime.

The following figure shows example about enabling/disabling CPU Idle status at runtime.

```
/* Enable CPU Idle at runtime */
$ echo 0 > /sys/devices/system/cpu/cpu0/cpuidle/state1/disable

/* Disable CPU Idle at runtime */
$ echo 1 > /sys/devices/system/cpu/cpu0/cpuidle/state1/disable
```

Figure 5-5 Example about enabling/disabling CPU Idle at runtime

Note: The CPU Idle status is shared between all CPUs on same cluster (CA57 or CA53) in current BSP. Therefore, if CPU Idle status in CPU0 is disabled, CPU Idle status of other CPUs on same cluster (CA57) are also disabled.

5.3 CPU Freq

5.3.1 CPU Freq definition

CPU Freq function can be used by defining operating-points in the device tree. The following figure shows example of definition for operating-points.

The following figure shows example of definition for CPU Freq of R-Car H3.

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

    a57_0: cpu@0 {
        compatible = "arm,cortex-a57", "arm,armv8";
        reg = <0x0>;
        device_type = "cpu";
        enable-method = "psci";
        clocks = <&cpg CPG_CORE R8A7795_CLK_Z>;
        cpu-idle-states = <&CPU_SLEEP_0>;
        operating-points-v2 = <&cluster0_tb0>,
        <&cluster0_tb1>, <&cluster0_tb2>,
        <&cluster0_tb3>, <&cluster0_tb4>,
        <&cluster0_tb5>, <&cluster0_tb6>,
        <&cluster0_tb7>;
        cpu-supply = <&vdd_dvfs>;
    };
    ...
    a53_0: cpu@100 {
        compatible = "arm,cortex-a53", "arm,armv8";
        reg = <0x100>;
        device_type = "cpu";
        enable-method = "psci";
        clocks = <&cpg CPG_CORE R8A7795_CLK_Z2>;
        cpu-idle-states = <&CPU_SLEEP_1>;
        operating-points-v2 = <&cluster1_opp>;
    };
    ...
};

cluster0_tb0: opp_table0 {
    compatible = "operating-points-v2";
    opp-shared;
```

```
        opp@1500000000 {
            opp-hz = /bits/ 64 <1500000000>;
            opp-microvolt = <830000>;
            clock-latency-ns = <300000>;
        };
        opp@1600000000 {
            opp-hz = /bits/ 64 <1600000000>;
            opp-microvolt = <900000>;
            clock-latency-ns = <300000>;
            turbo-mode;
        };
        opp@1700000000 {
            opp-hz = /bits/ 64 <1700000000>;
            opp-microvolt = <960000>;
            clock-latency-ns = <300000>;
            turbo-mode;
        };
    };
cluster1_opp: opp_table10 {
    compatible = "operating-points-v2";
    opp-shared;
    opp@800000000 {
        opp-hz = /bits/ 64 <800000000>;
        opp-microvolt = <820000>;
        clock-latency-ns = <300000>;
    };
    opp@1000000000 {
        opp-hz = /bits/ 64 <1000000000>;
        opp-microvolt = <820000>;
        clock-latency-ns = <300000>;
    };
    opp@1200000000 {
        opp-hz = /bits/ 64 <1200000000>;
        opp-microvolt = <820000>;
        clock-latency-ns = <300000>;
    };
};
```

Figure 5-6 Example of definition on device tree for operating-points

5.3.2 CPU Freq operation

The confirmation of parameters and operation for CPU Freq can be done via sysfs (/sys/devices/system/cpu/cpuX/cpufreq/*). (R-Car H3: X = 0/1/2/3/4/5/6/7; R-Car M3: X = 0/1/2/3/4/5; R-Car M3N and R-Car E3: X = 0/1)

The following figure shows example of operation for CPU Freq.

```
/* Checking Governor which is currently used */
$ cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor

/* Switching to Performance Governor */
$ echo performance > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor

/* Switching to Ondemand Governor */
$ echo ondemand > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor

/* Checking current frequency of CPU */
$ cat /sys/devices/system/cpu/cpu0/cpufreq/cpuinfo_cur_freq

/* Checking the available frequency */
$ cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_frequencies /* Normal mode */
$ cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_boost_frequencies /* when Boost mode is enabled*/

/* Change frequency of CPU to new frequency */
$ echo userspace > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
$ echo new_frequency > /sys/devices/system/cpu/cpu0/cpufreq/scaling_setspeed
/* with new_frequency is available_frequencies */
```

Figure 5-7 Example of operation for CPU Freq

Note: The CPU Freq configuration is shared between the same kind of CPUs in current BSP. Therefore, if the frequency in CPU0 is changed, the frequency of other CPUs on same cluster (CA57) are also changed as same frequency.

The following figure shows example of operation for Boost.

```
/* Boost disabled */
$ echo 0 > /sys/devices/system/cpu/cpufreq/boost

/* Boost enabled */
$ echo 1 > /sys/devices/system/cpu/cpufreq/boost
```

Figure 5-8 Example of operation for Boost

The following figure shows example of enabling/disabling AVS function:

```
/* Default AVS function in Linux kernel is enabled. For disabling AVS, please change in defconfig as below */
CONFIG_RCAR_POWER_AVS=n /* Disable AVS function */
```

Figure 5-9 Example of disabling AVS function

Note: For disabling AVS function in Initial Program Loader, please refer to Initial Program Loader User's Manual (as in Table 1-4), Chapter 5.3 Option setting, RCAR_AVS_SETTING_ENABLE part.

5.4 System Suspend to RAM

5.4.1 System Suspend to RAM definition

System Suspend to RAM doesn't need a special definition.

5.4.2 System Suspend to RAM operation

System Suspend to RAM can be operated via sysfs (/sys/power/state).

The following figure shows example of operation for System Suspend to RAM.

```
/* Required PMIC and board setting on Salvator-X/XS for System Suspend to RAM */  
1. Change switches setting:  
   SW6: 1 pin side  
   SW7: 1 pin side  
   SW8-1 : OFF  
   SW8-2 : OFF  
   SW8-3 : OFF  
   SW8-4 : OFF  
2. Set to PMIC to backup mode via i2c-tools command:  
   $ i2cset -f -y 7 0x30 0x20 0x0F  
3. Change SW23 to OFF  
  
/* Starting System Suspend to RAM */  
4. Request System Suspend to RAM  
   $ echo deep > /sys/power/mem_sleep           # this is default (*)  
   $ echo mem > /sys/power/state  
  
/* Trigger Resume state */  
5. Change SW23 to ON
```

Figure 5-10 Example of operation for System Suspend to RAM

The parameters of sysfs (/sys/power/state) excluding “mem” is not supported in current BSP.

(*) Note: The initial value of /sys/power/mem_sleep is "deep", so in default environment this step can be skipped.

5.5 Runtime PM

5.5.1 Runtime PM Definition

Runtime PM APIs can be used by defining the following items in device node of consumer driver.

The purpose of the definition is for power consumer device to probe directly with compatible driver and to bind to clock and power domain providers that it uses.

Table 5-3 Definition of clock and power domain in device node

Definition	Explanation
<code>compatible = "<vendor>, <platform-module name>";</code>	This property ensures for device can be probed directly with compatible driver.
<code>clocks = <&cpg CPG_MOD #>;</code>	The 'clocks' property contains module stop clock that device uses for its operation.
<code>power-domains = <&sysc pd_id>;</code>	<p>This property contains the sysc handler and id of power domain provider for this device.</p> <p><i>Note: 'pd_id' is identified number of power domain.</i> <i>(e.g. in R-Car H3/M3/M3N/E3/D3/V3U/V3H/V3M, power domain ids are: R8A7795_PD_ALWAYS_ON, R8A7795_PD_3DG_A,...) They are defined in header files:</i> <i>include/dt-bindings/power/r8a7795-sysc.h (for R-Car H3)</i> <i>include/dt-bindings/power/r8a7796-sysc.h (for R-Car M3)</i> <i>include/dt-bindings/power/r8a77965-sysc.h (for R-Car M3N)</i> <i>include/dt-bindings/power/r8a77990-sysc.h (for R-Car E3)</i> <i>include/dt-bindings/power/r8a77995-sysc.h (for R-Car D3)</i> <i>include/dt-bindings/power/r8a779a0-sysc.h (for R-Car V3U)</i> <i>include/dt-bindings/power/r8a77980-sysc.h (for R-Car V3H)</i> <i>include/dt-bindings/power/r8a77970-sysc.h (for R-Car V3M)</i></p>

The following figure shows example of clock and power domain control properties of a device node in the device tree of R-Car H3. It is similar for R-Car M3/M3N/E3/D3/V3U/V3H/V3M.

<pre> i2c0: i2c@e6500000 { compatible = "renesas,i2c-r8a7795"; clocks = <&cpg CPG_MOD 931>; power-domains = <&sysc R8A7795_PD_ALWAYS_ON>; }; </pre>

Figure 5-11 Device node with clock and power domain example

5.5.2 Runtime PM operation

Using Runtime PM APIs in device driver code

After device is successfully probed with driver, the Runtime PM state of the device is disabled. The clock and power domain of the device are also turned off. The following basic steps show how to add Runtime APIs to the driver source code:

Step 1: Enable Runtime PM for the device

Before using other Runtime PM APIs, it must be ensured that the device has been enabled.

The `pm_runtime_enable()` is usually called in *probe()* or *init()* function of the driver.

Step 2: Resume the device

Before using hardware (e.g. initialize setting, transfer data ...), the device must be resumed.

The `pm_runtime_get_sync()/pm_runtime_get()` (in synchronous/asynchronous context respectively) is usually called before hardware is set, to turn on the power domain and module clock of the device.

Step 3: Suspend the device

After the device has been completed operating, the device should be suspended to save power.

The `pm_runtime_put_sync()/pm_runtime_put()` (in synchronous/asynchronous context respectively) is usually called after the device completes its operation to turn off module clock and power domain to save power.

Step 4: Disable the device before removing it from system

Before removing the device from system, it must be disabled to prevent subsystem-level Runtime PM callbacks from being run for the device.

The `pm_runtime_disable()` must be called before device removal or error occurring case (symmetric with `pm_runtime_enable()`)

The following figure shows basic example of using Runtime PM APIs, the foo driver will resume its device each time start transferring data and suspend device when transferring complete.

```
static int foo_probe(struct platform_device *pdev)
{ ...
    pm_runtime_enable(dev);
    ...
}

static int start_transfer (struct foo_dev *dev, int *buf, int length)
{ ...
    pm_runtime_get_sync(dev);
    //init channel and set data to transfer to register
    ...
}

static int end_transfer(struct foo_dev *dev)
{ ...
    //finish transfer, setting register before stopping channel
    pm_runtime_put_sync(dev);
    ...
}

static int foo_remove(struct platform_device *pdev)
{ ...
    pm_runtime_disable(dev);
}
```

Figure 5-12 Runtime PM APIs basic usage flow

Usage notes and recommendation

- ① For using Runtime PM APIs, the device must be successfully probed with driver. Each device must be set as Table 5-3. In case of one driver controlling several modules in different power domains, it should consider to break out them so that each driver controls only separating device.
- ② Because Runtime PM APIs can control to turn on/off module clock, it should remove direct Clock framework APIs to simplify driver source code and avoid redundant code if the device driver does not use special clocks (e.g. external clocks ...).
- ③ Differences between synchronous and asynchronous Runtime PM APIs

When resuming device using **synchronous** APIs, it **ensures** that clock and power domains have been turned on right before `pm_runtime_get_sync()` finished successfully.

When resuming device using **asynchronous** APIs, it **does not** ensure that clock and power domains have been turned on right before `pm_runtime_get()` finished successfully. It just adds a request to system to turn on the clock and power domain for device, and clock and power domain will be turned on by system after certain duration (depending on workload of system.). The advantage of asynchronous function is less processing time. It is suitable in the case that needs quick processing time.

It is similar for `pm_runtime_put_sync()/pm_runtime_put()`.

- ④ With devices lay on always-ON power domain, in fact, their driver may initialize the hardware one time (in *probe()* or *init()* function). Then in processing, they just set hardware for their operation (e.g. transfer data...) and do not initialize hardware more. In this case, the driver can operate well because the power is always on and module registers are retained after `pm_runtime_put_sync()/pm_runtime_put()`.

But with devices lay on other power domains, after `pm_runtime_put_sync()/pm_runtime_put()` the module may be turned off and registers are not retained, so it does recommend using flow as **Figure 5-12**, mean that initializing all necessary registers after each `pm_runtime_get_sync()/pm_runtime_get()` and before using device.

- ⑤ In special cases, device just uses Runtime PM APIs to control clock only. In the device tree, it should be set as *power-domains= <&sysc_R8A7795_PD_ALWAYS_ON>*. In case of GFX, for example, its module hardware supports control power by itself. It just needs Runtime PM APIs to control clock. So it should be set as *power-domains= <&sysc_R8A7795_PD_ALWAYS_ON>* although GFX lays on other power domain.

5.6 Thermal management (IPA/EMS)

5.6.1 Thermal management (IPA/EMS) definition

IPA and EMS function can be used by defining CPU and THS in the device tree. The following example shows definition of EMS on R-Car H3 for three sensors (THS1, THS2 and THS3) and IPA for THS3. ⁽¹⁾

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

    a57_0: cpu@0 {
        compatible = "arm,cortex-a57", "arm,armv8";
        reg = <0x0>;
        device_type = "cpu";
        power-domains = <&sysc R8A7795_PD_CA57_CPU0>;
        next-level-cache = <&L2_CA57>;
        enable-method = "psci";
        cpu-idle-states = <&CPU_SLEEP_0>;
        #cooling-cells = <2>;
        dynamic-power-coefficient = <854>;                /* (1) */
        cooling-min-level = <0>;
        cooling-max-level = <2>;
        clocks = <&cpg CPG_CORE R8A7795_CLK_Z>;
        operating-points-v2 = <&cluster0_tb0>,
            <&cluster0_tb1>, <&cluster0_tb2>,
            <&cluster0_tb3>, <&cluster0_tb4>,
            <&cluster0_tb5>, <&cluster0_tb6>,
            <&cluster0_tb7>;
    };

    ... ..
    a53_0: cpu@100 {
        compatible = "arm,cortex-a53", "arm,armv8";
        reg = <0x100>;
        device_type = "cpu";
        power-domains = <&sysc R8A7795_PD_CA53_CPU0>;
        next-level-cache = <&L2_CA53>;
        enable-method = "psci";
        cpu-idle-states = <&CPU_SLEEP_1>;
        #cooling-cells = <2>;
        dynamic-power-coefficient = <277>;
        cooling-min-level = <0>;
        cooling-max-level = <2>;
        clocks = <&cpg CPG_CORE R8A7795_CLK_Z2>;
        operating-points-v2 = <&cluster1_opp>;
        capacity-dmips-mhz = <540>;
    };

    ... ..
}
```

Figure 5-13 Example of definition on device tree of CPU for IPA/EMS

```

thermal-zones {
    emergency {                                /* → node for EMS */
        polling-delay = <1000>;
        on-temperature = <110000>;             /* (2) */
        off-temperature = <95000>;             /* (3) */
        target_cpus = <&a57_1>, <&a57_2>, <&a57_3>, <&a53_0>, <&a53_1>, <&a53_2>, <&a53_3>;
        status = "disabled";
    };
    sensor_thermal3: sensor-thermal3 {
        polling-delay-passive = <250>;
        polling-delay = <0>;
        /* sensor ID */
        thermal-sensors = <&tsc 2>;
        sustainable-power = <6313>;            /* (4) */
        trips {
            threshold: trip-point0 { /* milliCelsius */
                temperature = <90000>;          /* (5) */
                hysteresis = <2000>;
                type = "passive";
            };
            target: trip-point1 { /* milliCelsius */
                temperature = <100000>;         /* (6) */
                hysteresis = <2000>;
                type = "passive";
            };
            sensor3_crit: sensor3-crit {
                temperature = <120000>;         /* (7) */
                hysteresis = <2000>;
                type = "critical";
            };
        };
    };
    cooling-maps {                               /* → node for IPA */
        map0 {
            trip = <&target>;
            cooling-device = <&a57_0 0 2>;
            contribution = <1024>;
        };
        map1 {
            trip = <&target>;
            cooling-device = <&a53_0 0 2>;
            contribution = <1024>;
        };
        .....
    };
};

```

Figure 5-14 Example of definition on device tree of THS for IPA/EMS

The following parameters in device tree depend on SoC and Board. They are tuned for Salvator-X/XS board.

Table 5-4 The tuned parameters of IPA/EMS in device node

No	Parameters	Value	Depend	Description
(1)	dynamic-power-coefficient	854 (CA57) 277 (CA53)	SoC	This value is based on power measurement/estimation and common with R-Car Series, 3 rd Generation series.
(2)	on-temperature	110000	SoC/Board	This value indicates the emergency temperature and invokes emergency shutdown functionality when exceeding this temperature. It's set lower than temperature (7).
(3)	off-temperature	95000	SoC/Board	This value indicates the temperature to disable emergency shutdown. It's set lower than on-temperature (2).
(4)	sustainable-power	6313 (R-Car H3) 3874 (R-Car M3) 2439 (R-Car M3N) 717 (R-Car E3)	SoC	This value is based on power measurement/estimation and different by each SoC.
(5)	temperature	90000	SoC/Board	This value indicates the temperature to enable IPA. It's set higher than the temperature of idle state.
(6)	temperature	100000	SoC/Board	This value indicates the target temperature for IPA. It's set in the temperature of max performance.
(7)	temperature	120000	SoC/Board	This value indicates the temperature to execute the System shutdown. It's set lower than limit temperature of Board.
(7)	contribution	1024 (CA57, CA53)	SoC	This value indicates weight of power allocation for each cooling device in system.

About System shutdown, please refer to RENESAS_RCH3M3M3NE3_ThermalSensor_UME.pdf.

5.6.2 Thermal management (IPA/EMS) operation

After thermal sensor is successfully initialized and registered to thermal core, the core will continuously read temperature from the sensor. When read temperature exceeds IPA starting trip point, IPA will start to throttle CPU frequency.

If the temperature continues to raise and exceed EMS trip point, then CPU core will be turned off.

The system will be shut down completely if System Shutdown trip point is reached.

About detail processing of EMS, when the temperature exceeds the EMS trip point, EMS just turns off target CPU core(s) that is current online in the system. So, the target CPU core(s) that are plugged-out by user are not controlled by EMS operation. In example in Table 5-5, the cpu7 is not controlled by EMS.

Table 5-5 Example about details EMS operation (on R-Car H3)

Steps	System status, user action and EMS operation	Online CPU in system
1	System boot with 8 cores and EMS target cpu in device tree is 8 cores ([cpu0, cpu1, ..., cpu7]): target_cpus = <57_1>, <57_2>, <57_3>, <53_0>, <53_1>, <53_2>, <53_3>;	cpu0, cpu1, cpu2, cpu3, cpu4, cpu5, cpu6, cpu7
2	User (or other factors) plugs-out one cpu (e.g. cpu7)	cpu0, cpu1, cpu2, cpu3, cpu4, cpu5, cpu6
3	The temperature exceeds the EMS trip point. And EMS starts and plugs-out all current online target CPU core(s).	cpu0
4	The temperature reduces and is under the EMS trip point. And EMS stops and plugs-in only offline CPU core(s) that is plugged-out at step 3.	cpu0, cpu1, cpu2, cpu3, cpu4, cpu5, cpu6

5.6.3 Disabling thermal management (IPA/EMS) operation

In some cases, the CPU cooling (scale down CPU frequency and shutdown CPU core) is not necessary, so thermal management can be disabled. Disabling thermal management (IPA/EMS) can be done by changing configures in file arch/arm64/configs/defconfig as below figure:

```
[ The settings of enabling IPA/EMS ]
/* IPA and EMS configures in arch/arm64/configs/defconfig are respective as below */
CONFIG_CPU_THERMAL=y                # for IPA.
CONFIG_RCAR_THERMAL_EMS=y           # for EMS.

[ The settings of disabling IPA/EMS ]
/* Disable IPA and EMS configures in arch/arm64/configs/defconfig */
# CONFIG_CPU_THERMAL is not set      # for IPA.
# CONFIG_RCAR_THERMAL_EMS is not set # for EMS.
```

Figure 5-15 Example of disabling thermal management (IPA/EMS)

6. Integration

6.1 Directory Configuration

The power management directory configuration is shown below.

```

arch/arm64/boot/dts/renesas/
├── r8a77950.dtsi
├── r8a77950-salvator-x.dts
├── r8a77951.dtsi
├── r8a77951-salvator-x.dts
├── r8a77951-salvator-xs.dts
├── r8a779m1.dtsi
├── r8a779m1-salvator-x.dts
├── r8a779m1-salvator-xs.dts
├── r8a77960.dtsi
├── r8a77960-salvator-x.dts
├── r8a77960-salvator-xs.dts
├── r8a77961.dtsi
├── r8a77961-salvator-xs.dts
├── r8a779m3.dtsi
├── r8a779m3-salvator-xs.dts
├── r8a77965.dtsi
├── r8a77965-salvator-x.dts
├── r8a77965-salvator-xs.dts
├── r8a779m5.dtsi
├── r8a779m5-salvator-x.dts
├── r8a779m5-salvator-xs.dts
├── r8a77970.dtsi
├── r8a77970-eagle.dts
├── r8a77970-eagle-function.dts
├── r8a77970-es1.dtsi
├── r8a77970-es1-eagle.dts
├── r8a77970-es1-eagle-function.dts
├── r8a77980.dtsi
├── r8a77980-es2.dtsi
├── r8a77980-condor.dts
├── r8a77980-es2-condor.dts
├── r8a77990.dtsi
├── r8a77990-ebisu.dts
├── r8a77990-ebisu-4d.dts
├── r8a77990-es10-ebisu.dts
├── r8a77990-es10-ebisu-4d.dts
├── r8a77995.dtsi
├── r8a77995-draak.dts
├── r8a779a0.dtsi
├── r8a779a0-falcon.dts
├── salvator-common.dtsi
├── salvator-x.dtsi
├── salvator-xs.dtsi
drivers/cpufreq/
├── cpufreq-dt-platdev.c

```

```
drivers/soc/renesas/  
├── r8a7795-sysc.c  
├── r8a7796-sysc.c  
├── r8a77965-sysc.c  
├── r8a77970-sysc.c  
├── r8a77980-sysc.c  
├── r8a77990-sysc.c  
├── r8a77995-sysc.c  
├── r8a779a0-sysc.c  
├── rcar-avs.c  
├── rcar_ems_ctrl.c  
├── rcar-sysc.c  
└── rcar-sysc.h  
drivers/thermal/  
└── rcar_gen3_thermal.c  
include/linux/soc/renesas/  
└── rcar_ems_ctrl.c  
include/dt-bindings/power/  
├── r8a7795-sysc.h  
├── r8a7796-sysc.h  
├── r8a77961-sysc.h  
├── r8a77965-sysc.h  
├── r8a77970-sysc.h  
├── r8a77980-sysc.h  
├── r8a77990-sysc.h  
├── r8a77995-sysc.h  
└── r8a779a0-sysc.h
```

Figure 6-1 Directory configuration

7. Reference

7.1 Design Note for System Suspend to RAM support

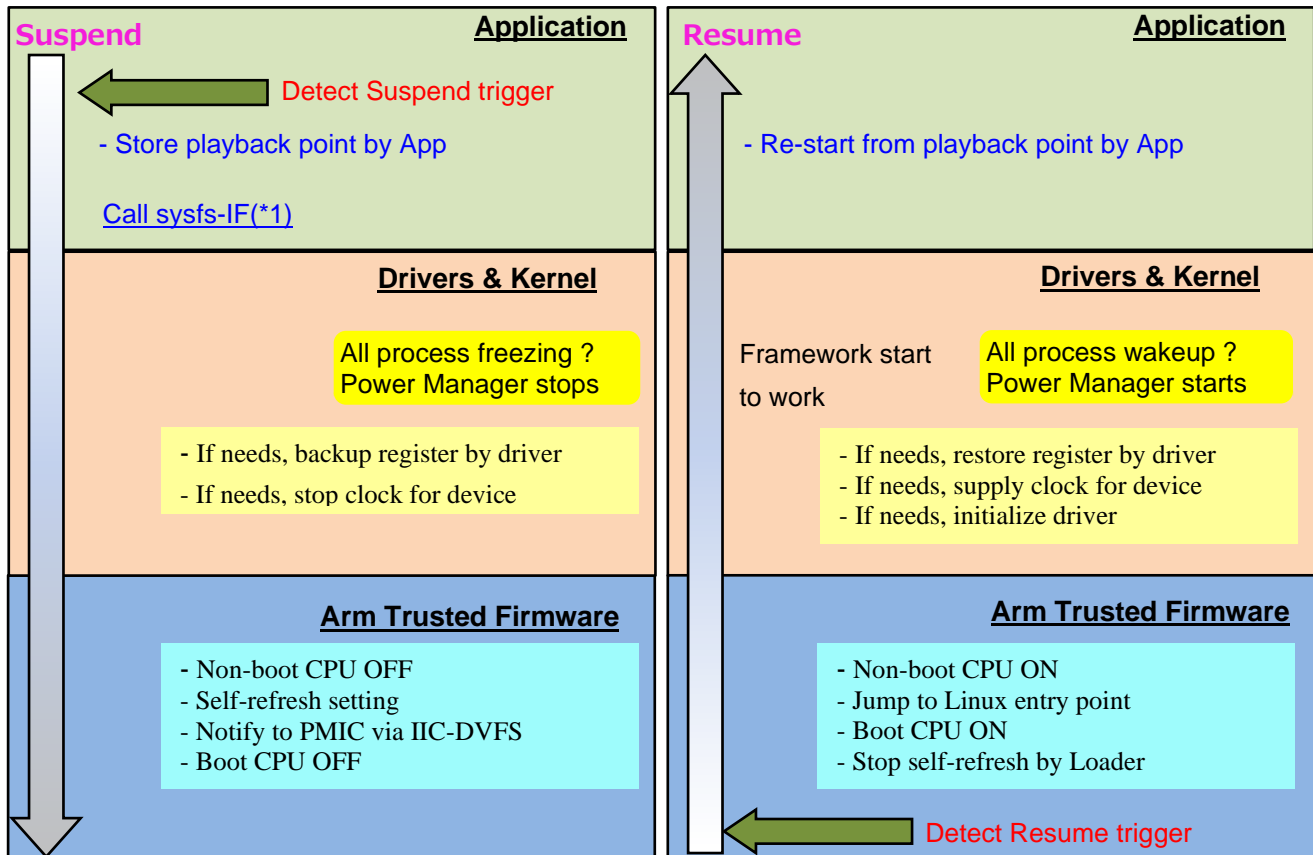
Depending on the use-case that System Suspend to RAM is supported, the necessary implementation of modules (from application layer to driver/kernel layer) are performed differently. The following shows an example of preparation App layer for System Suspend to RAM in R-Car Series, 3rd Generation.

Table 7-1 An example of preparation App layer for System Suspend to RAM

Use-cases with Suspend to RAM		Preparation App layer	Application side (Customer side)
Network communication	UDP	Not required	Suspend : Not need Resume : Not need
	TCP	Required	Suspend : Not need Resume : Need to re-connect
Video playback		Required	Suspend : Need to record pause position Resume : Need to re-start from recoded pause position
Audio playback		Not required	Suspend : Not need Resume : Not need
3D graphics		Not required	Suspend : Not need Resume : Not need
SD data read/write		Required	Suspend : Need to unmount Resume : Need to re-mount
USB memory data read/write		Required	Suspend : Need to unmount Resume : Need to re-mount
eMMC data read/write		Not required	Suspend : Not need Resume : Not need
HDMI input/output		Not required	Suspend : Not need Resume : Not need

In “Not required” use cases, since Kernel and driver layer recovers to the state before suspend, application layer does not need any special change.

Below figure is whole software sequence System Suspend to RAM in R-Car Series, 3rd Generation as an example for video playback.



(*1) "System Suspend to RAM" put it into the state that can change to Suspend before issuing the demand.

Note: Blue text must be done by Application layer.
Application code must ensure this when applying System Suspend To RAM.

Figure 7-1 System Suspend to RAM whole software sequence

As above sequence, the supported System Suspend to RAM in R-Car Series, 3rd Generation platform is allocated in all layers from Application to Arm Trusted Firmware. Beside Application layer, System Suspend to RAM has been implemented by kernel developers. For Application layer, application developers must ensure System Suspend to RAM implementation in application source code.

Below items are some implementation notes for application:

- In case of video playback, before suspend, the application must: pause the player, back up the current position of data stream. When resume, the application must: restore the current position of data stream, resume the player.

Below figure is sequence of System Suspend to RAM.

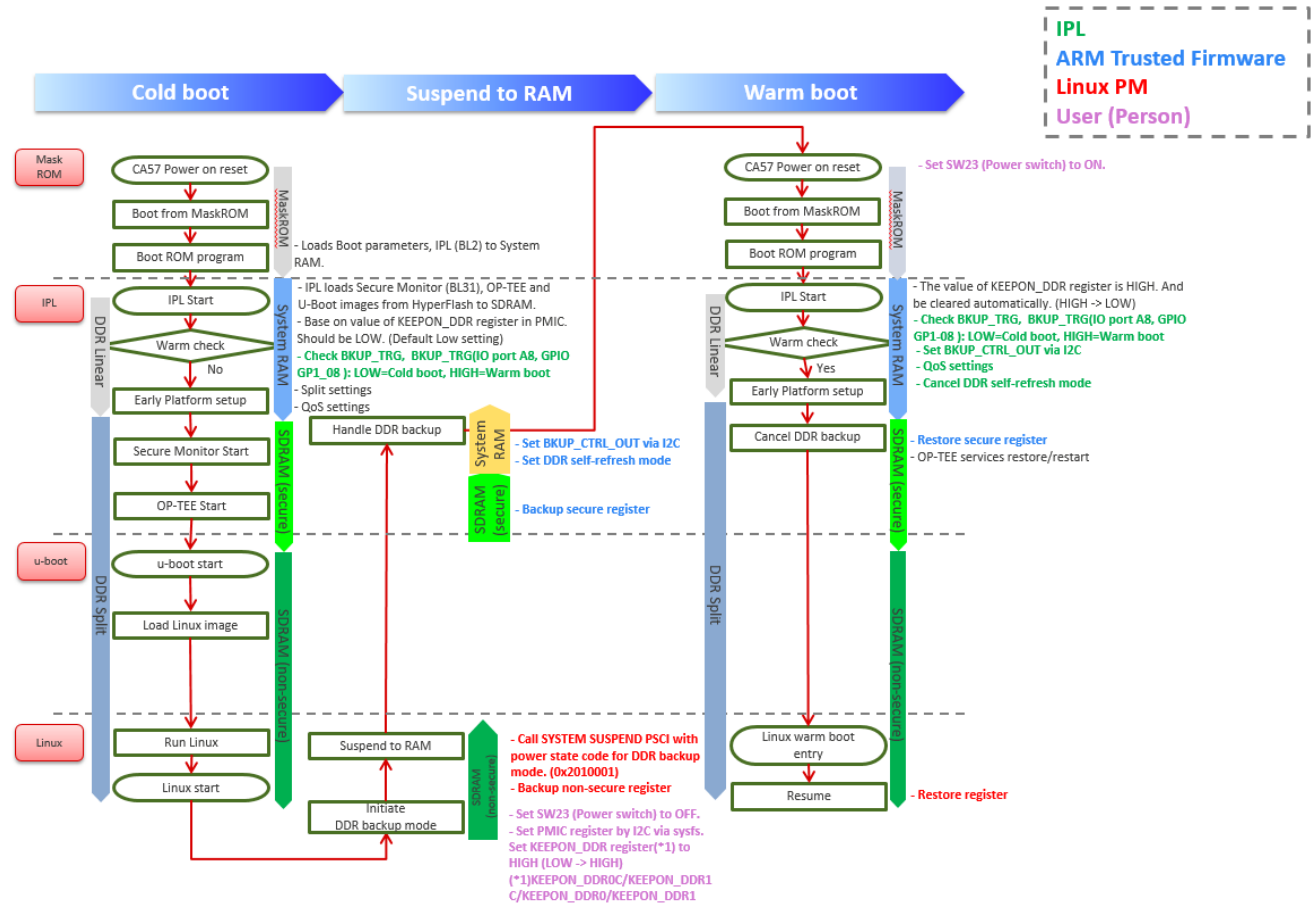


Figure 7-2 System Suspend to RAM sequence

7.2 Power management function depend on PMIC

PMIC is a separated chip with R-Car Series, 3rd Generation SoC. And in fact, R-Car Series, 3rd Generation platform can be deployed with several kinds of PMIC.

CPU Freq, IPA, and System Suspend to RAM functions are implementation dependent on the PMIC mounted on the R-Car H3-SiP/M3-SiP System Evaluation Board Salvator-X/XS (on R-Car H3/M3/M3N); System Suspend to RAM functions are implementation dependent on the PMIC mounted on the R-Car E3 System Evaluation Board Ebisu (on R-Car E3). When using the other board (not the R-Car H3-SiP/M3-SiP System Evaluation Board Salvator-X/XS and R-Car E3 System Evaluation Board Ebisu as described in 3.1), it is needed to disable function that depend on the PMIC.

The following table shows that the PMIC dependence of power management functions and the way to disable it.

Table 7-2 Power management functions depend on PMIC

Functions	PMIC dependence	How to disable
CPU Hotplug	No	
CPU Idle	No	
CPU Freq	Yes	CONFIG_CPU_FREQ is not set.
Runtime PM	No	
System Suspend to RAM	Yes	System Suspend to RAM cannot be disabled on Linux. Therefore, it is disabled by Arm Trusted Firmware (BL31). Refer to chapter 4.3.1 in “Security Board Support Package User’s Manual”.
IPA	Yes	CONFIG_CPU_THERMAL is not set.
EMS	No	