Embedded Linux Conference
Europe

Optimizing and Developing Non-CPU Device Power Management by DEVFREQ

Chanwoo Choi
Samsung Electronics

#lfelc

THE LINUX FOUNDATION

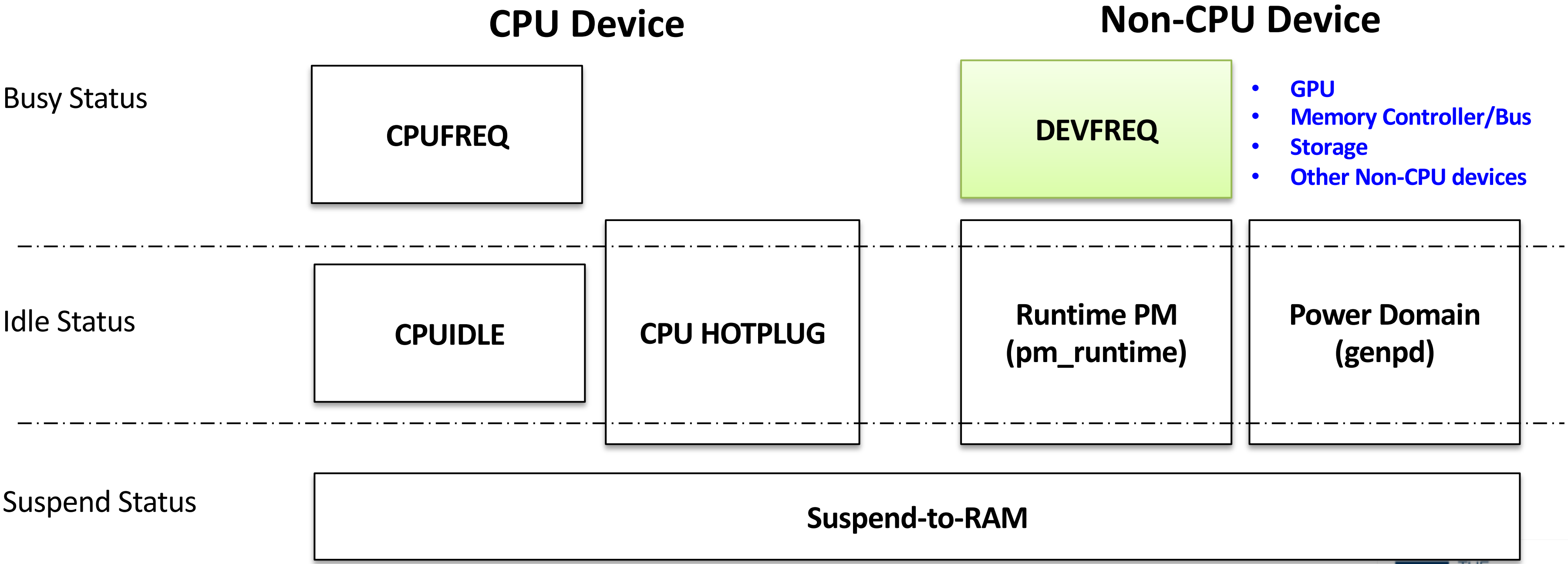
Contents

- DEVFREQ?
  - How To Add Devfreq Driver and Devfreq Governor
  - Sysfs Interface
- Collaboration with other Frameworks
- Simple Profiling and Performance Tuning Point
- Use-Case in Mainline Kernel
- Weakness of DEVFREQ and Further TODO

DEVFREQ?

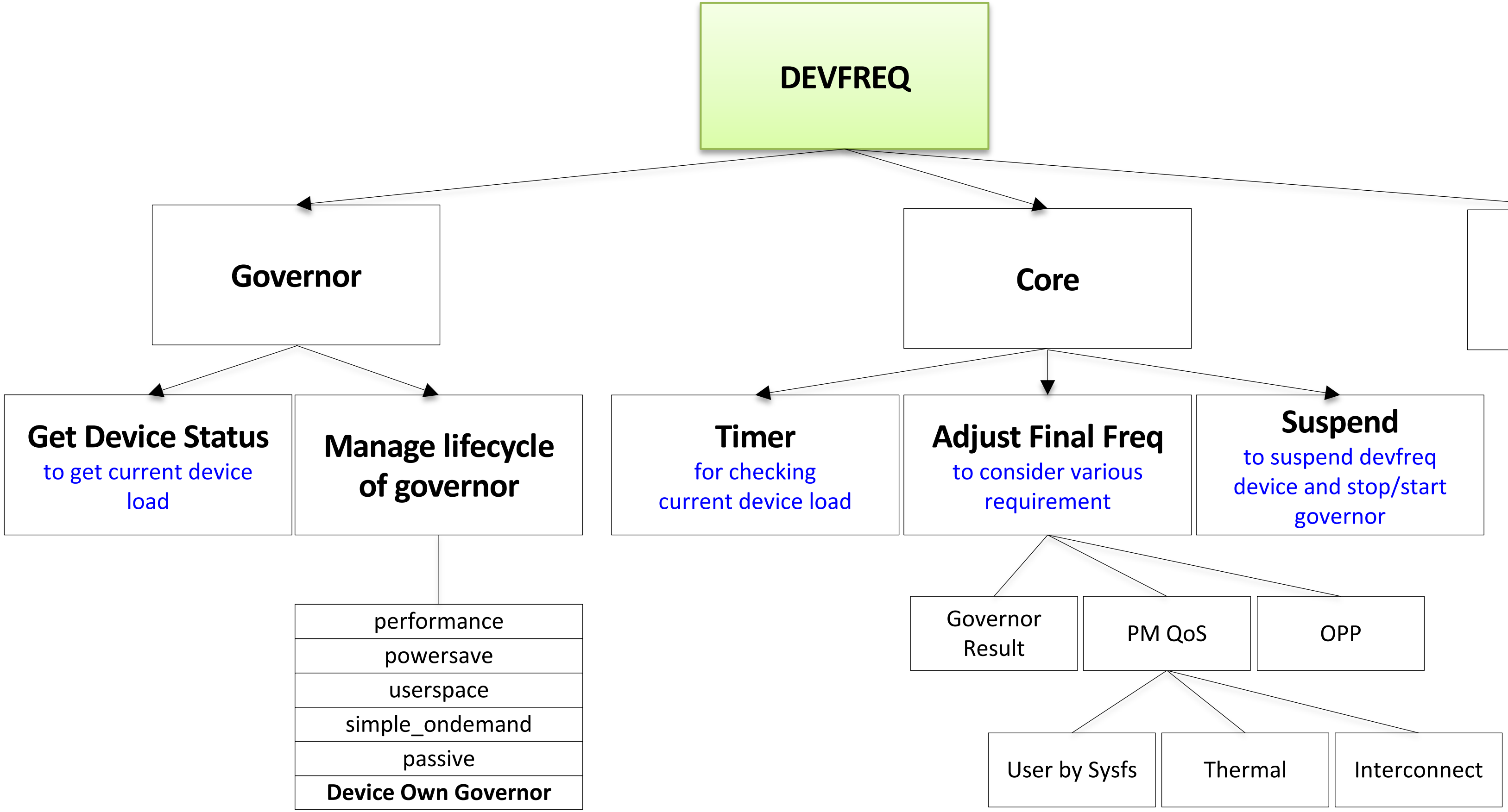
- **Performance** Demand
  - High-Quality image generated by GPU
  - Data Transfer within deadline via Memory Bus
  - Low Latency for accessing Storage
- **Power-Consumption** Requirement
  - Increase the battery capacity continuously
- **Need to support [1]DVFS for Non-CPU Power-management**
- Provide **power-management mechanism** for Non-CPU device to **keep balance** between Performance and Power

DEVFREQ with System Status



- GPU
- Memory Controller/Bus
- Storage
- Other Non-CPU devices

DEVFREQ Internal Module



DEVFREQ

Governor

Core

Sysfs
to show or store devfreq value

Get Device Status
to get current device load

Manage lifecycle of governor

performance
powersave
userspace
simple_ondemand
passive
Device Own Governor

Timer
for checking current device load

Adjust Final Freq
to consider various requirement

Suspend
to suspend devfreq device and stop/start governor

Governor Result

PM QoS

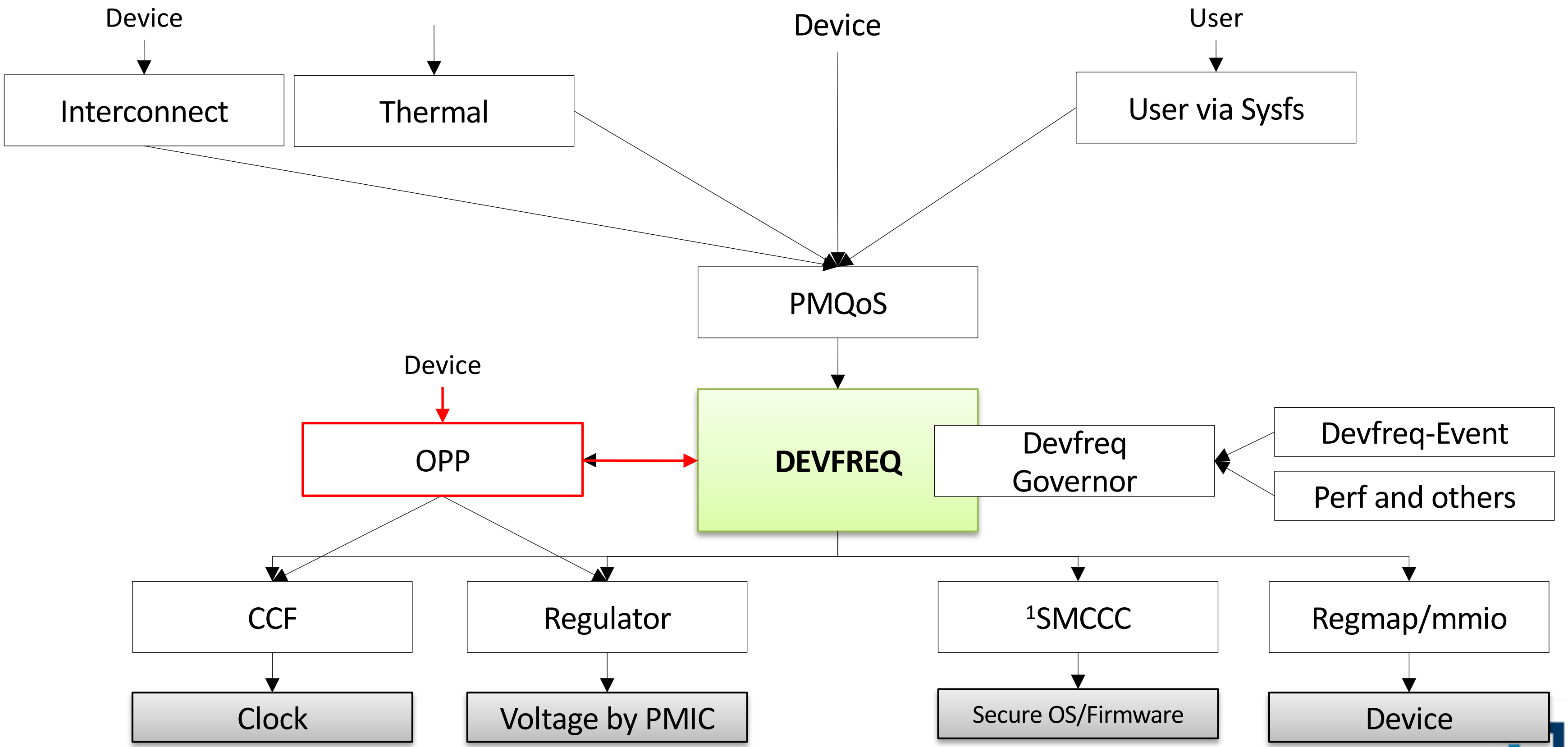
OPP

User by Sysfs

Thermal

Interconnect

DEVFREQ relation with External Framework



[1] SMCCC (Secure Monitor Call Calling Convention)

How To Add Devfreq Driver &
How To Add Devfreq Governor

DEVFREQ Device Driver

- Support DVFS by controlling clock and voltage according to device status

- The kind of DEVFREQ Driver
  - **GPU** like ARM Panfrost/Lima, MSM GPU
  - **Memory Bus** like AMBA AXI Bus
  - **Memory Controller** like DMC (Dynamic Memory Controller)
  - **Storage** like UFS (Universal Flash Storage)
  - **L2 Cache**
  - And so on

Step to Add DEVFREQ Driver

1. **Initialize 'struct devfreq_dev_profile' for device profile**
   - Initialize **'polling_ms'** of timer period and **'timer'** is either deferrable or delayed timer.
   - Implement **'target'** fucntion to set the next frequency/voltage
   - Implement **'get_dev_status'** function to get current device status
2. **Get clock/regulator and OPP table from DeviceTree**
   - devm_clk_get(), devm_regulator_get(), dev_pm_opp_of_add_table() or other OPP helper functions
3. **Choose governor**
   - simple_ondemand, userspace and others.
4. **Add devfreq device**
   - devm_devfreq_add_device(device, devfreq_dev_profile, governor, data)
5. **(optional) Register OPP notifier**
   - devm_devfreq_register_opp_notifier(device, devfreq)

How to Add DEVFREQ Driver - struct devfreq_dev_profile

Name	Descritpion	Mandatory or Optional
initial_freq	**Initial frequency.**	Optional
polling_ms	**Polling interval for timer.** If 0, disable polling. The unit is millisecond(ms).	Optional But, If simple_ondemand, Mandatory
timer	**Timer type is either deferrable or delayed timer. The default value is deferrable timer.** - DEVFREQ_TIMER_DEFERRABLE is for deferrable timer. with CONFIG_HIGH_RES_TIMERS, CONFIG_NO_HZ - DEVFREQ_TIMER_DELAYED is for delayed timer.	Optional
up_threshold	**If the load is over this value, the frequency jumps.** Valid value = 0 to 100. Default value is 90. downdifferential < upthreshold must hold.	Optional
down_differential	**If the load is under upthreshold - downdifferential, the frequency downs.** Valid value = 0 to 100. Default value is 5. downdifferential < upthreshold must hold.	Optional
(*target)	**Set operating frequency decided by devfreq core with both governo and PM QoS request**	**Mandatory**
(*get_dev_status)	**Return the current load of devfreq device. The result is used for deciding the next frequency by governor.**	Optional But, If simple_ondemand, Mandatory
(*get_cur_freq)	**Return the current correct frequency.**	Optional
(*exit)	**Exit the devfreq device.**	Optional

DEVFREQ Governor

- Decide proper frequency by governor algorithm
- User can add the own device governor

- Governor
  - **simple_ondemand**
  - performance
  - powersave
  - userspace
  - **passive**
- Device Own Governor
  - tegra_actmon

Step to Add DEVFREQ Governor

1. **Initialize ‘struct devfreq_governor’ for governor**
   - Initialize **‘name’** of governor name.
   - Implement **‘get_target_freq’** functioin to get next frequency decided by governor algorithm.
   - Implement **‘event_handler’** function to handle the governor event for governor lifecycle.

2. **Add devfreq governor**
   - devfreq_add_governor(devfreq_governor);

3. **The devfreq governor will be used by devfreq drivers.**

How to Add DEVFREQ Governor - struct devfreq_governor

Name	Descritpion	Mandatory or Optional
name	**Governor name like “simple_ondemand”, “performance”.**	**Mandatory**
attr	**Governor sysfs attribute flag.** Basically, common sysfs attributes are added to devfreq class and need to initialize the following flags for using non-general sysfs attributes . - DEVFREQ_GOV_ATTR_POLLING_INTERVAL : polling_interval - DEVFREQ_GOV_ATTR_TIMER : timer - DEVFREQ_GOV_ATTR_UP_THRESHOLD : up_threshold - DEVFREQ_GOV_ATTR_DOWN_DIFF : down_differential	Optional
flag	**Governor feature flag** - DEVFREQ_GOV_FLAG_IMMUTABLE : If set, this governor is never changeable to others. - DEVFREQ_GOV_FLAG_IRQ_DRIVEN : If set, this governor is working with irq instead of timer.	Optional
(*get_target_freq)	**Return the desired operating frequency for the device according to governor algorithm.**	**Mandatory**
(*event_handler)	**Callback for devfreq core to notify events to governors.** - DEVFREQ_GOV_START : When governor start - DEVFREQ_GOV_STOP : When governor stop - DEVFREQ_GOV_UPDATE_INTERVAL : When timer interval is updated via sysfs - DEVFREQ_GOV_SUSPEND : When governor suspend - DEVFREQ_GOV_RESUME : When governor resume	**Mandatory**

Example to Add Device Own Governor

- “tegra_actmon” governor

in drivers/devfreq/tegra30_devfreq.c

```
static struct devfreq_governor tegra_devfreq_governor = {
          .name = "tegra_actmon",
          .attr = DEVFREQ_GOV_ATTR_POLLING_INTERVAL,
          .flag = DEVFREQ_GOV_FLAG_IMMUTABLE
                    | DEVFREQ_GOV_FLAG_IRQ_DRIVEN,
          .get_target_freq = tegra_governor_get_target,
          .event_handler = tegra_governor_event_handler,
};
```

‘poling_interval’ sysfs is used for **Tegra ACTMON h/w period setting.**

‘_FLAG_IMMUTABLE’ means that if device used ‘tegra_actmon’ governor, it cannot change to other governors.

‘_IRQ_DRIVEN’ means that it is based on interrupt method instead of timer.

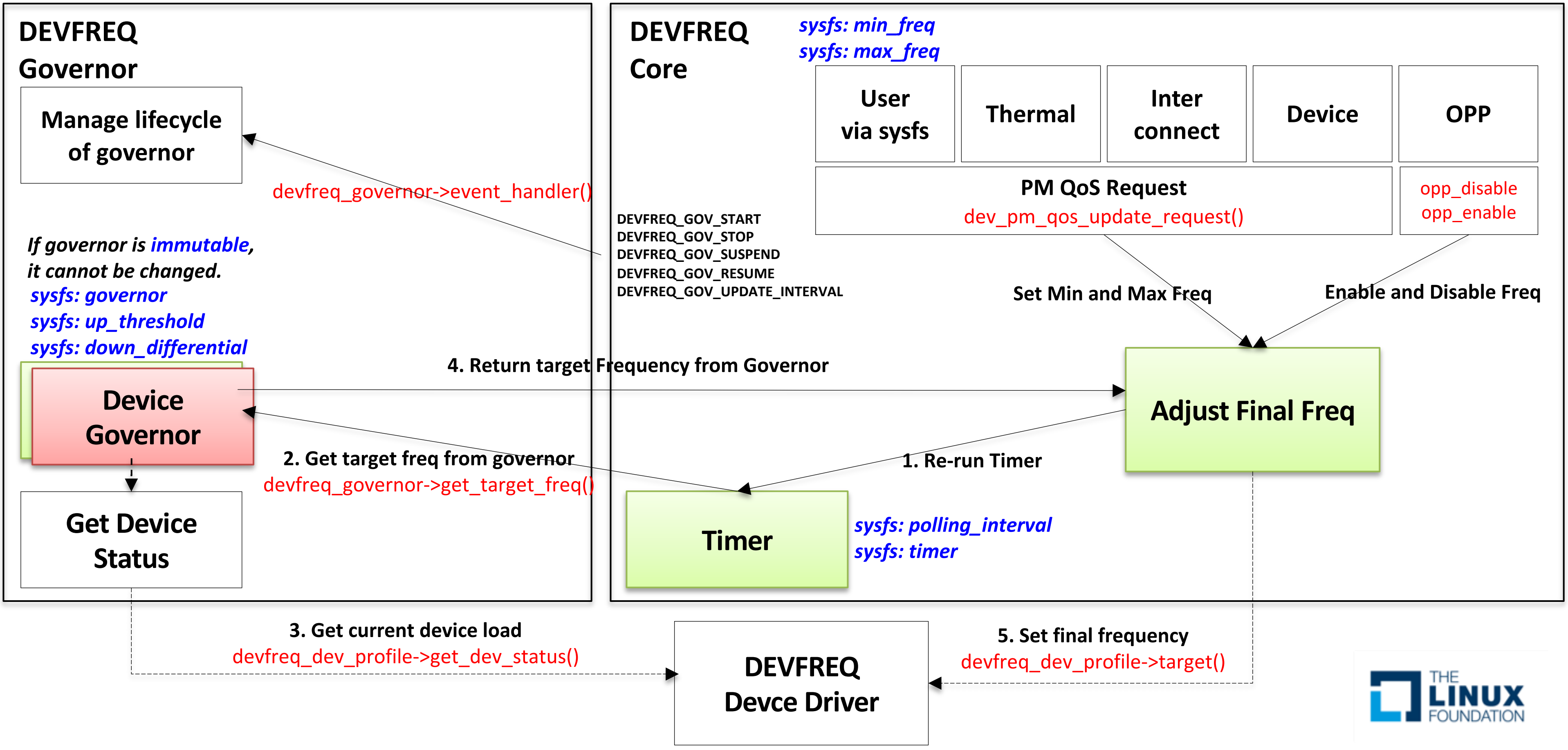
Return the desired operating frequency **according to Tegra ACTMON governor algorithm.**

Handle Tegra ACTMON governor **lifecycle in accordance with DEVFREQ_GOV_* event.**

In Summary,

- **Immutable** and **Interrupt method** for sampling
- Use **Tegra ACTMON Governor** instead of default devfreq governors

DEVFREQ Driver and Governor Behavior



DEVFREQ Governor

Manage lifecycle of governor

devfreq_governor->event_handler()

*If governor is **immutable**, it cannot be changed.*

sysfs: governor

sysfs: up_threshold

sysfs: down_differential

Device Governor

Get Device Status

DEVFREQ Core

sysfs: min_freq

sysfs: max_freq

User via sysfs

Thermal

Inter connect

Device

OPP

PM QoS Request

dev_pm_qos_update_request()

opp_disable

opp_enable

DEVFREQ_GOV_START

DEVFREQ_GOV_STOP

DEVFREQ_GOV_SUSPEND

DEVFREQ_GOV_RESUME

DEVFREQ_GOV_UPDATE_INTERVAL

Set Min and Max Freq

Enable and Disable Freq

4. Return target Frequency from Governor

Adjust Final Freq

1. Re-run Timer

2. Get target freq from governor

devfreq_governor->get_target_freq()

Timer

sysfs: polling_interval

sysfs: timer

3. Get current device load

devfreq_dev_profile->get_dev_status()

DEVFREQ Devce Driver

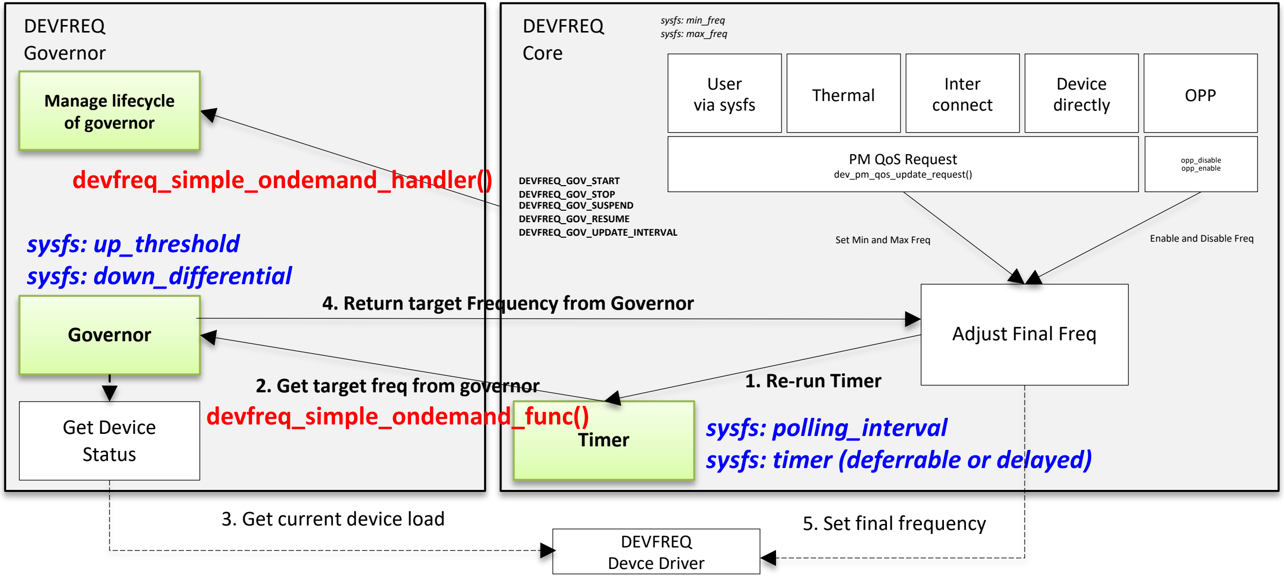
5. Set final frequency

devfreq_dev_profile->target()

Simple_ondmeand Governor Behavior

in drivers/devfreq/governor_simple_ondemand.c

```
static struct devfreq_governor devfreq_simple_ondemand = {
        .name = DEVFREQ_GOV_SIMPLE_ONDEMAND,
        .attr = DEVFREQ_GOV_ATTR_POLLING_INTERVAL
                | DEVFREQ_GOV_ATTR_TIMER
                | DEVFREQ_GOV_ATTR_UP_THRESHOLD
                | DEVFREQ_GOV_ATTR_DOWN_DIFF,
        .get_target_freq = devfreq_simple_ondemand_func,
        .event_handler = devfreq_simple_ondemand_handler,
};
```



DEVFREQ Governor

Manage lifecycle of governor

devfreq_simple_ondemand_handler()

sysfs: up_threshold
sysfs: down_differential

Governor

Get Device Status

DEVFREQ Core

sysfs: min_freq
sysfs: max_freq

User via sysfs

Thermal

Inter connect

Device directly

OPP

PM QoS Request
dev_pm_qos_update_request()

opp_disable
opp_enable

DEVFREQ_GOV_START
DEVFREQ_GOV_STOP
DEVFREQ_GOV_SUSPEND
DEVFREQ_GOV_RESUME
DEVFREQ_GOV_UPDATE_INTERVAL

Set Min and Max Freq

Enable and Disable Freq

Adjust Final Freq

4. Return target Frequency from Governor

1. Re-run Timer

2. Get target freq from governor

devfreq_simple_ondemand_func()

Timer

sysfs: polling_interval
sysfs: timer (deferrable or delayed)

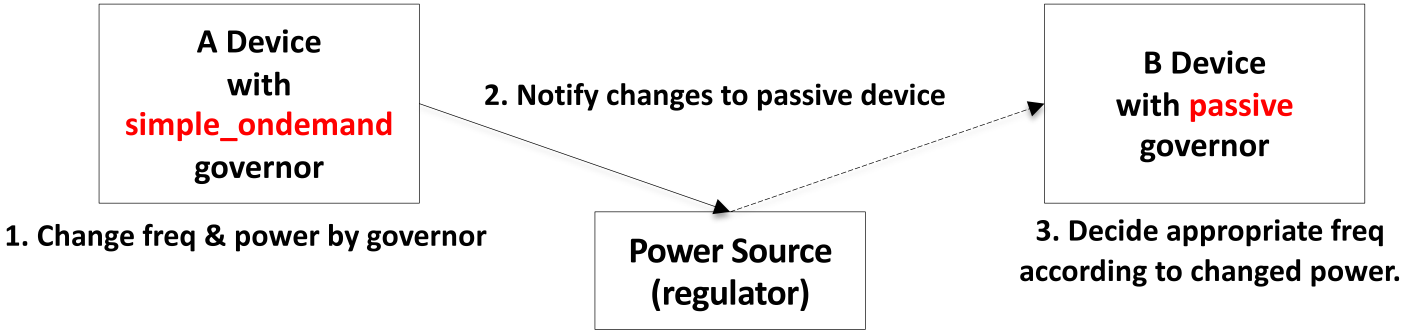
3. Get current device load

5. Set final frequency

DEVFREQ
Devce Driver

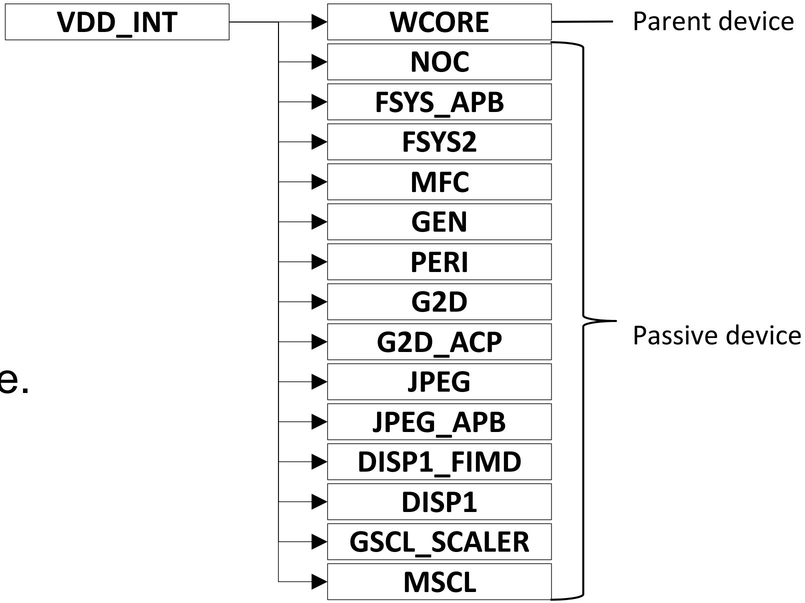
Passive Governor

- Depend on behavior of parent device such as other devfreq device or CPU.
- Pattern of using passive governor
  - Two devices share the same power source.


A Device with simple_ondemand governor
2. Notify changes to passive device
B Device with passive governor
1. Change freq & power by governor
Power Source (regulator)
3. Decide appropriate freq according to changed power.

Example of Passive Governor

- Memory Bus Device in Samsung Exynos5422 SoC
  - VDD_INT regulator provides power to 15 [1]AMBA AXI Bus device.
- Step to change freq/voltage
  - Decide next freq/voltage on WCORE device.
    - **User can change the governor of WCORE device via sysfs and then passive device freq will be changed.**
  - If next freq is higher than previous freq,
    - Change WCORE's freq & voltage
    - Change freq of 15 AMBA AXI Bus
  - If next freq is less than previous freq,
    - Change freq of 15 AMBA AXI bus
    - Change WCORE's freq & voltage



[1] AMBA (Advanced Microcontroller Bus Architecture) AXI (Advanced eXtensible Interface)

Sysfs Interface

Common Sysfs Interface for Devfreq Class

Name	Descritpion	RW
governor	Show and store the **current governor name**	RW
available_governor	Show the **available governor list**	RO
available_frequencies	Show the **available frequencies**	RW
target_freq	Show the **current frequency** of the one of OPP table	RO
cur_freq	Show the current frequency of hardware clock rate if get_cur_freq() is implemented by devfreq driver. If get_cur_freq() is not implemented, it is same with taget_freq.	RO
min_freq	Show and store the **minimum frequency**	RW
max_freq	Show and store the **maximum frequency**	RW
trans_stat	Show the **frequency transition statistics** and **time in state** To reset the statistics as following: echo 0 > /sys/class/devfreq/[dev name]/trans_stat	RW

Non-Common Sysfs Interface for Devfreq Governor

- Each governor is able to choose the following sysfs nodes if it is needed

Name	Descritpion	RW	Use-case
timer	Show and store the **timer type (deferrable or delayed)** - DEVFREQ_GOV_ATTR_TIMER	RW	simple_ondemand
polling_interval	Show and store the **polling interval** - DEVFREQ_GOV_ATTR_POLLING_INTERVAL	RW	simple_ondemand tegra_actmon
up_threshold	Show and store **up_threshold** tuning point - flag name : DEVFREQ_GOV_UP_THRESHOLD	RW	simple_ondemand
down_differential	Show and store **down_differential** tuning point - flag name : DEVFREQ_GOV_DOWN_DIFF	RW	simple_ondemand

DEVFREQ Governor of both Sysfs and Feature Flags

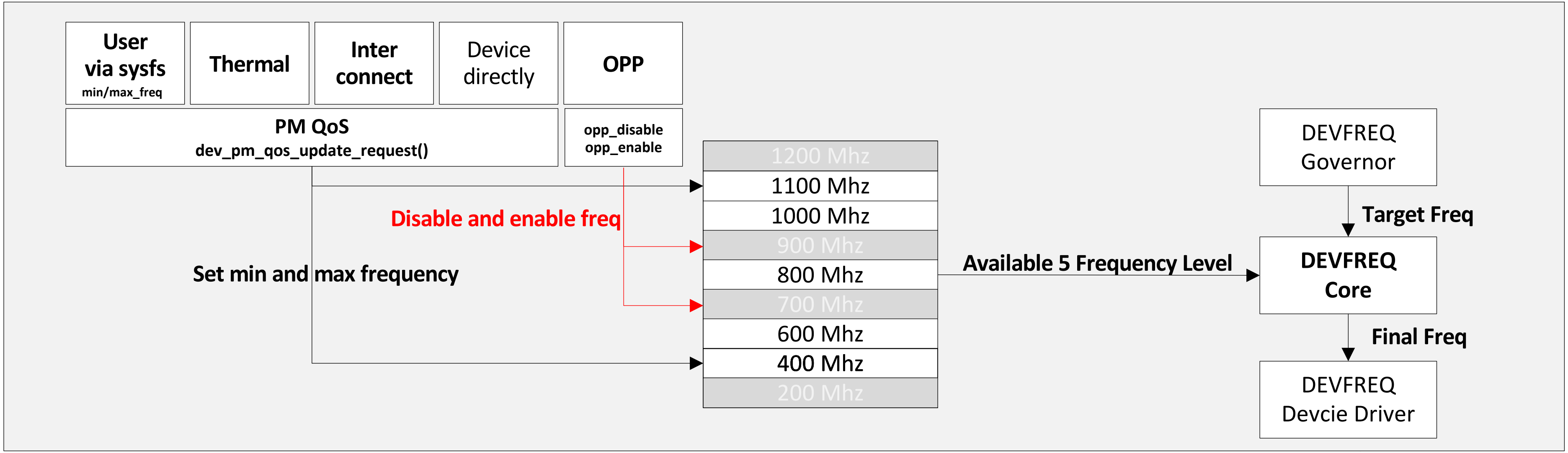
	governor sysfs node	simple _ondemand	peformance	powersave	userspace	passive	tegra30_actmon
Common Sysfs Interface for devfreq class	governor	O	O	O	O	O	O
	available_governors	O	O	O	O	O	O
	available_frequencies	O	O	O	O	O	O
	cur_freq	O	O	O	O	O	O
	target_freq	O	O	O	O	O	O
	min_freq	**O**	**O**	**O**	**O**	**O**	**O**
	max_freq	**O**	**O**	**O**	**O**	**O**	**O**
	trans_stat		O	O	O	O	O
Non-common Sysfs Interface for specific governor	**polling_interval**	**O**	**X**	**X**	**X**	**X**	**O**
	timer	**O**	**X**	**X**	**X**	**X**	**X**
	up_threshold	**O**	**X**	**X**	**X**	**X**	**X**
	down_differential	**O**	**X**	**X**	**X**	**X**	**X**
Governor feature for specific governor	immutable	X	X	X	X	O	O
	interrupt_driven	X (polling based on timer)	X	X	X	X	O (polling based on h/w irq)

Collaboration with other Frameworks

- OPP
- PM QoS
- Interconnect
- Thermal

What is Collaboration Purpose with other Framework?

- Boosting or Constrainting Frequency
  - Change the frequency range according to mode
    - ie. Performance, Optimized, Powersave and Ultra-Powersave mode
  - Boosting for preventing performance drop
  - Constrainting for preventing either high-temperature or misuse power



OPP (Operating Performance Points) with DEVFREQ

- OPP is mandatory to support DVFS with frequency and voltage.
- OPP provides helper function to get OPP info from devicetree
  - dev_pm_opp_of_add_table()
  - dev_pm_opp_of_remove_table()
- Handle clock and regulator by OPP helper function
  - dev_pm_opp_set_rate()
  - dev_pm_opp_set_regulators()
  - dev_pm_opp_put_regulators()

in arch/arm/boot/dts/exynos3250.c

```
bus_dmc_opp_table: opp_table1 {
        compatible = “opeating-points-v2”;

        opp-50000000 {
                opp-hz = /bits/ 64 <50000000>;
                opp-microvolt = <800000>;
        }
        opp-100000000 {
                opp-hz = /bits/ 64 <100000000>;
                opp-microvolt = <800000>;
        }
        opp-134000000 {
                opp-hz = /bits/ 64 <134000000>;
                opp-microvolt = <800000>;
        }
        opp-200000000 {
                opp-hz = /bits/ 64 <200000000>;
                opp-microvolt = <825000>;
        }
        opp-400000000 {
                opp-hz = /bits/ 64 <400000000>;
                opp-microvolt = <875000>;
        }
};

bus_dmc: bus_dmc {
        compatible = “Samsung,exynos-bus”;
        clocks = <&cmu_dmc CLK_DIV_DMC>;
        clock-names = “bus”;
        operating-points-v2 = <&bus_dmc_opp_table>;
}
```

OPP - Enable & Disable Each Frequency

- Enable and Disable OPP entries
  - dev_pm_opp_disable(dev, 200000000)
  - dev_pm_opp_disable(dev, 134000000)
  - dev_pm_opp_disable(dev, 100000000)

  - dev_pm_opp_enable(dev, 134000000)

```
in arch/arm/boot/dts/exynos3250.c

bus_dmc_opp_table: opp_table1 {
        compatible = “opeating-points-v2”;

        opp-50000000 {
                opp-hz = /bits/ 64 <50000000>;
                opp-microvolt = <800000>;
        }
        opp-100000000 {
                opp-hz = /bits/ 64 <100000000>;
                opp-microvolt = <800000>;
        }
        opp-134000000 {
                opp-hz = /bits/ 64 <134000000>;
                opp-microvolt = <800000>;
        }
        opp-200000000 {
                opp-hz = /bits/ 64 <200000000>;
                opp-microvolt = <825000>;
        }
        opp-400000000 {
                opp-hz = /bits/ 64 <400000000>;
                opp-microvolt = <875000>;
        }
};

bus_dmc: bus_dmc {
        compatible = “Samsung,exynos-bus”;
        clocks = <&cmu_dmc CLK_DIV_DMC>;
        clock-names = “bus”;
        operating-points-v2 = <&bus_dmc_opp_table>;
}
```

PMQoS used by DEVFREQ

- Set min and max frequency to guarantee device's demand
  - DEV_PM_QOS_MIN_FREQUENCY
  - DEV_PM_QOS_MAX_FREQUENCY

- Example,
  - PM QoS Request
    1. dev_pm_qos_add_request(device, qos_request, **DEV_PM_QOS_MIN_FREQUENCY**)
    2. dev_pm_qos_request_active(qos_request)
    3. dev_pm_qos_update_request(qos_request, **frequency**)
  - Read PM QoS Requests
    1. dev_pm_qos_read_value(dev, **DEV_PM_QOS_MIN_FREQUENCY**)
  - PM QoS Release
    1. dev_pm_qos_update_request(qos_request, **0**)
    2. dev_pm_qos_remove_request(qos_request)

Interconenct with DEVFREQ

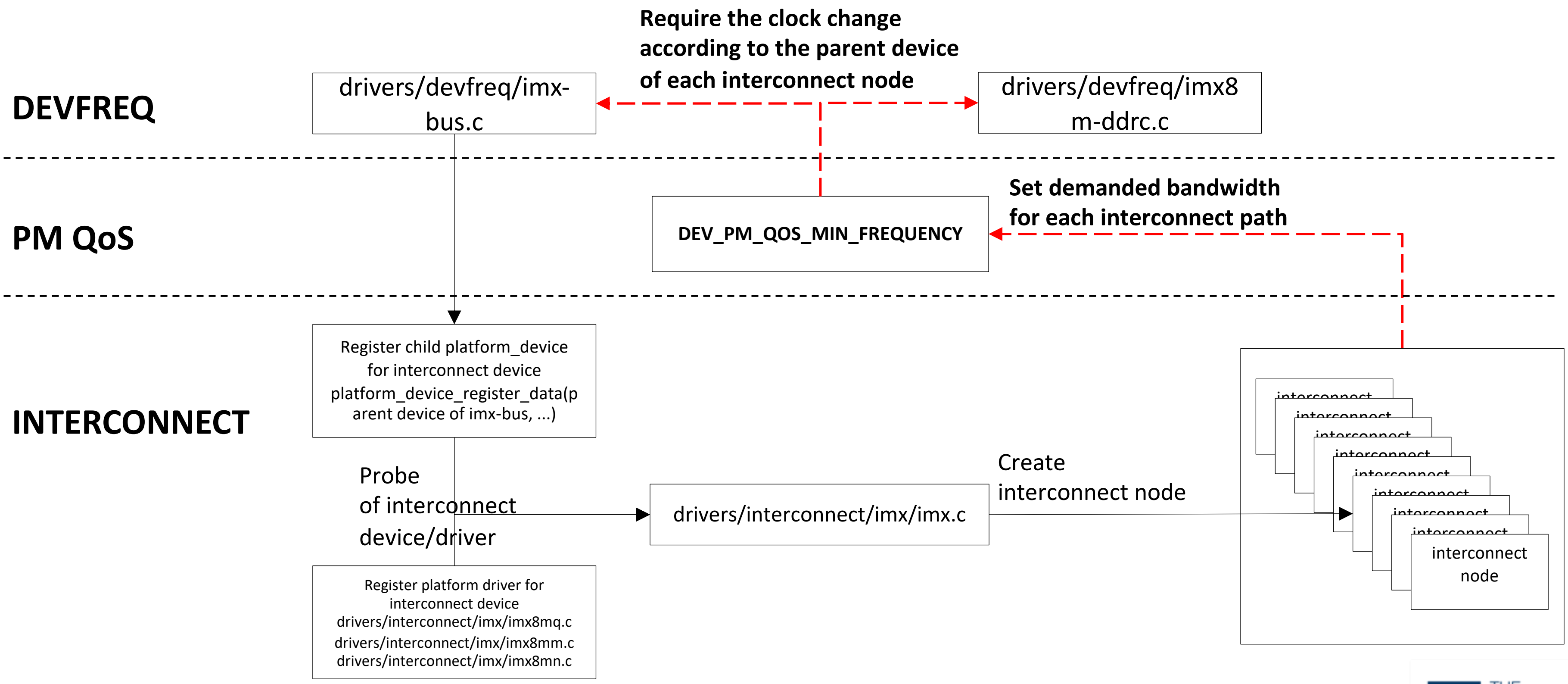
- Interconnect framework control the setting of the 'interconnects on an SoC' like memory controller and data bus.

- Two framework might be connected through PM QoS interface for guaranting performance.

DEVFREQ Device Driver	Bridge by PM QoS Interface	INTERCONNECT Device Driver
drivers/devfreq/imx-bus.c drivers/devfreq/imx8m-ddrc.c IMX SoC	PM QoS DEV_PM_QOS_MIN_FREQUENCY	drivers/interconnect/imx/imx.c drivers/interconnect/imx/imx8mq.c drivers/interconnect/imx/imx8mm.c drivers/interconnect/imx/imx8mn.c
drivers/devfreq/exynos-bus.c	PM QoS DEV_PM_QOS_MIN_FREQUENCY	drivers/interconnect/exynos/exynos.c *(But, not yet merged and under review)*

Example of connection between DEVFREQ and Interconnect Device Driver

Example of NXP I.MX SoC



DEVFREQ

drivers/devfreq/imx-bus.c

Require the clock change according to the parent device of each interconnect node

drivers/devfreq/imx8m-ddrc.c

PM QoS

DEV_PM_QOS_MIN_FREQUENCY

Set demanded bandwidth for each interconnect path

INTERCONNECT

Register child platform_device for interconnect device platform_device_register_data(parent device of imx-bus, ...)

Probe of interconnect device/driver

drivers/interconnect/imx/imx.c

Create interconnect node

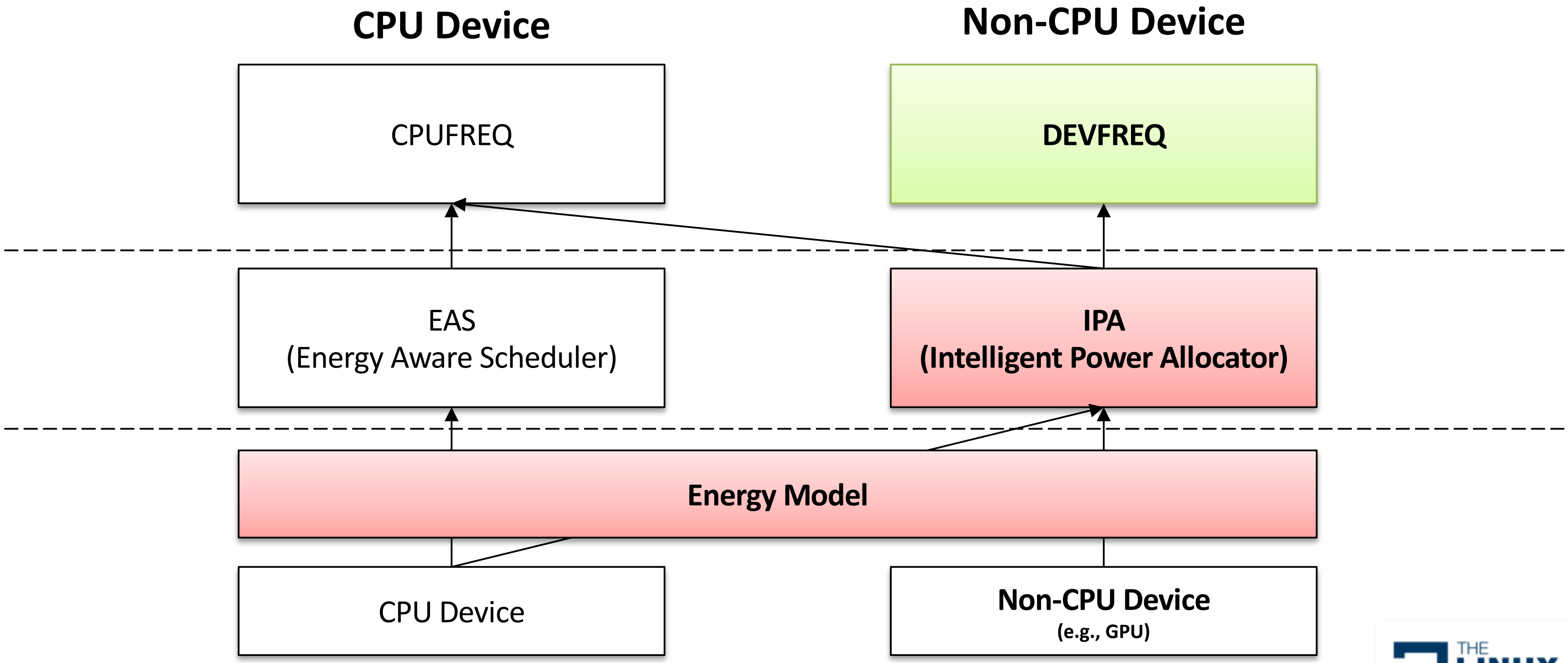
interconnect node

Register platform driver for interconnect device
drivers/interconnect/imx/imx8mq.c
drivers/interconnect/imx/imx8mm.c
drivers/interconnect/imx/imx8mn.c

Thermal with DEVFREQ

- Register devfreq device as a cooling device
  - drivers/thermal/devfreq_cooling.c
- Basically, adjust frequency by using fixed trip-points defined in devicetree
  - step_wise thermal governor
- More Advanced than fixed trip-points method, adjust frequency with IPA (Intelligent Power Allocator) governor using EM (Energy Model).

Thermal IPA and EM with DEVFREQ



Simple Profiling & Peformance Tuning Point

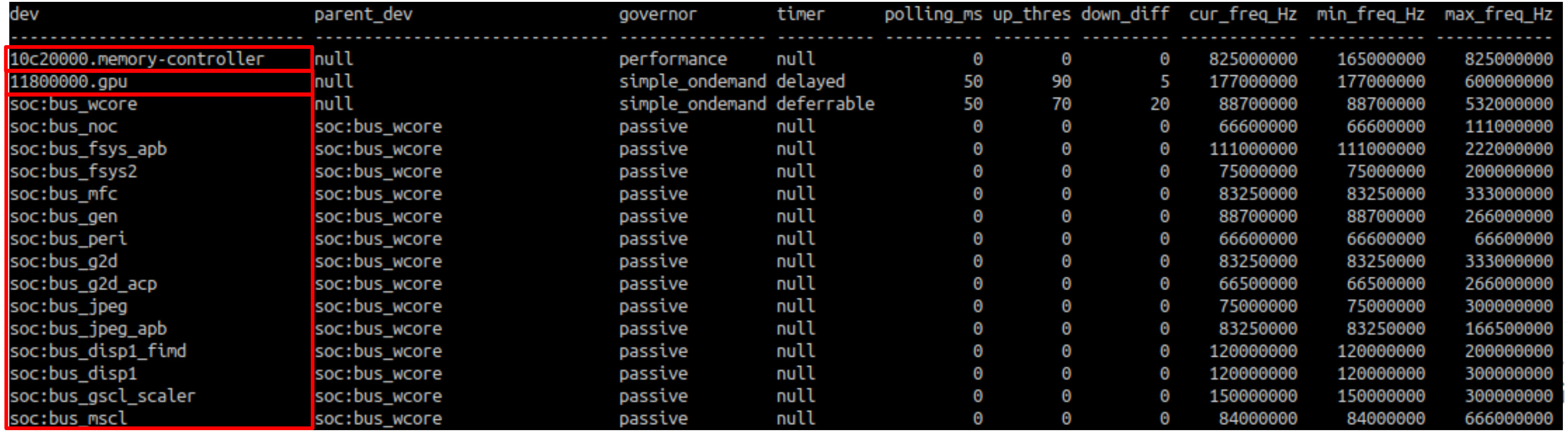
- sysfs
- debugfs
- tracepoint
- genpd, pm_runtime

Simple Profiling

- Debugfs
  - devfreq_summary
- Sysfs (Sampling)
  - min_freq, cur_freq, max_freq and trans_stat sysfs interface
- Tracepoint (Tracing)
  - devfreq, thermal, power
- Device power status (active or supended)
  - Generic Power Domain and Runtime PM

Simple Profiling - Debugfs Interface

- /sys/kernel/debug/devfreq/devfreq_summary
  - Test device : Odroid-XU3 (Samsung Exynos5422 SoC)
    - 17 Non-CPU devices
    - **1 GPU** (11800000.gpu) / **simple_ondemand / 177MHz ~ 600MHz**
    - **1 Memory Controller** (10c20000.memory-controller) / **performance / 165MHz~825Mz**
    - **1 AMBA AXI Bus** (soc:bus_wcore) / **simple_ondemand / 88.7MHz ~ 532MHz**
    - **14 AMBA AXI Bus** with ‘soc:bus_wcore’ parent device / **passive**



Memory Controller

GPU

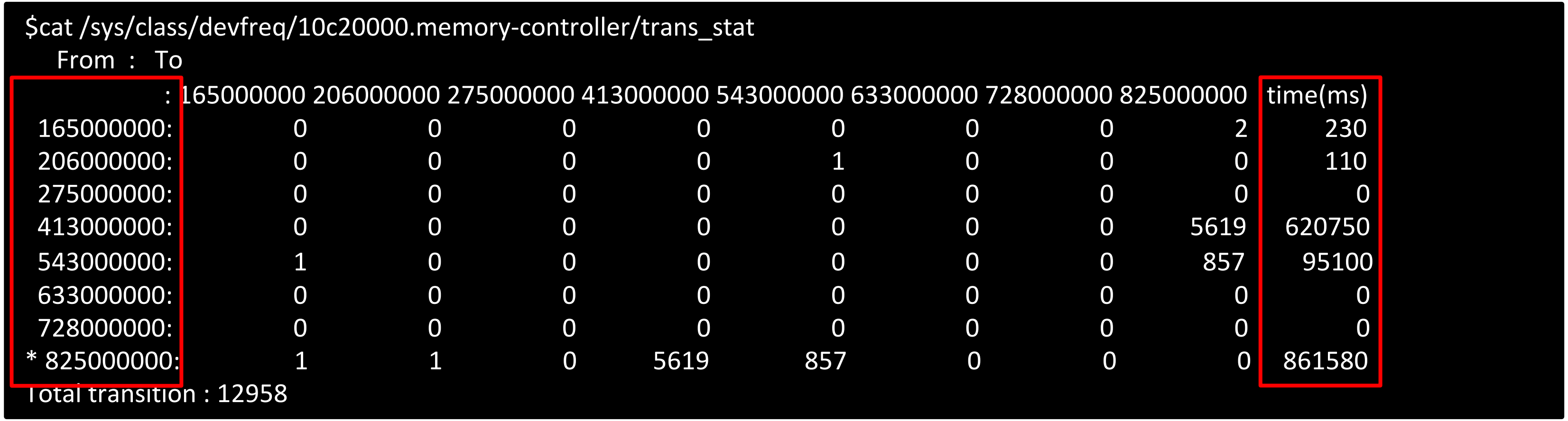
Memory Data Bus

```
dev                            parent_dev                     governor        timer      polling_ms up_thres down_diff cur_freq_Hz  min_freq_Hz  max_freq_Hz
------------------------------ ------------------------------ --------------- ---------- ---------- -------- --------- ------------ ------------ ------------
10c20000.memory-controller     null                           performance     null                0        0         0    825000000    165000000    825000000
11800000.gpu                   null                           simple_ondemand delayed            50       90         5    177000000    177000000    600000000
soc:bus_wcore                  null                           simple_ondemand deferrable         50       70        20     88700000     88700000    532000000
soc:bus_noc                    soc:bus_wcore                  passive         null                0        0         0     66600000     66600000    111000000
soc:bus_fsys_apb               soc:bus_wcore                  passive         null                0        0         0    111000000    111000000    222000000
soc:bus_fsys2                  soc:bus_wcore                  passive         null                0        0         0     75000000     75000000    200000000
soc:bus_mfc                    soc:bus_wcore                  passive         null                0        0         0     83250000     83250000    333000000
soc:bus_gen                    soc:bus_wcore                  passive         null                0        0         0     88700000     88700000    266000000
soc:bus_peri                   soc:bus_wcore                  passive         null                0        0         0     66600000     66600000     66600000
soc:bus_g2d                    soc:bus_wcore                  passive         null                0        0         0     83250000     83250000    333000000
soc:bus_g2d_acp                soc:bus_wcore                  passive         null                0        0         0     66500000     66500000    266000000
soc:bus_jpeg                   soc:bus_wcore                  passive         null                0        0         0     75000000     75000000    300000000
soc:bus_jpeg_apb               soc:bus_wcore                  passive         null                0        0         0     83250000     83250000    166500000
soc:bus_disp1_fimd             soc:bus_wcore                  passive         null                0        0         0    120000000    120000000    200000000
soc:bus_disp1                  soc:bus_wcore                  passive         null                0        0         0    120000000    120000000    300000000
soc:bus_gscl_scaler            soc:bus_wcore                  passive         null                0        0         0    150000000    150000000    300000000
soc:bus_mscl                   soc:bus_wcore                  passive         null                0        0         0     84000000     84000000    666000000
```

Simple Profiling - Sysfs Interface

- min_freq, max_freq and cur_freq
  - How to control them
    - echo [available frequency] > /sys/class/devfreq/[dev name]/min_freq and read it
    - echo [available frequency] > /sys/class/devfreq/[dev name]/max_freq and read it
    - cat > /sys/class/devfreq/[dev name]/cur_freq
  - How to use them for profiling
    - Make simple shell script to print the frequency periodically.
- trans_stat
  - Show transition statistics and time in each frequency
  - How to control them
    - echo 0 > /sys/class/devfreq/[dev name]/trans_stat and read it

Simple Profiling - Sysfs 'trans_stat'



```
$cat /sys/class/devfreq/10c20000.memory-controller/trans_stat
    From  :   To
           : 165000000 206000000 275000000 413000000 543000000 633000000 728000000 825000000  time(ms)
  165000000:         0         0         0         0         0         0         0         2       230
  206000000:         0         0         0         0         1         0         0         0       110
  275000000:         0         0         0         0         0         0         0         0         0
  413000000:         0         0         0         0         0         0         0      5619    620750
  543000000:         1         0         0         0         0         0         0       857     95100
  633000000:         0         0         0         0         0         0         0         0         0
  728000000:         0         0         0         0         0         0         0         0         0
* 825000000:         1         1         0      5619       857         0         0         0    861580
Total transition : 12958
```

- How to use it for performance profiling
  1. Try to tune and optimize the your code
  2. **Reset trans_stat** by 'echo 0 > /sys/class/devfreq/[dev name]/trans_stat'
  3. **Read 'time_in_state'** by 'cat /sys/class/devfreq/[dev name]/trans_stat'
  4. **Execute benchmark tool**
  5. **Read 'time_in_state'**
  6. **Calculate diff 'time_in_state'** between before and after benchmark tool

Simple Profiling - Tracepoint

- DEVFREQ
  - Track devfreq behavior when frequency change and monitoring
    - /sys/kernel/debug/tracing/event/**devfreq/devfreq_frequency**
    - /sys/kernel/debug/tracing/event/**devfreq/devfreq_monitor**
- Thermal
  - Track throttling or un-throttling point due to high temperature
    - /sys/kernel/debug/tracing/event/**thermal/thermal_temperature**
    - /sys/kernel/debug/tracing/event/**thermal/thermal_zone_trip**
- PM QoS
  - Track what request the pm qos of both minimum and maximum freq
    - /sys/kernel/debug/tracing/event/**power/dev_pm_qos_update_rquest**

Simple Profiling - Tracepoint Event

- devfreq_monitor
  - Show when device monitoring is executed by timer.
  - It is used to check how often it has been monitored.
- devfreq_frequency
  - Show frequency change point.
  - This is useful for determining whether or not the frequency has changed at the appropriate timing and checking the history of frequency change.
- thermal_temperature
  - Show temperature of thermal device like CPU, GPU
- thermal_zone_trip
  - Show trip point when arrived at the specific temperature (throttling or un-throttling)
- dev_pm_qos_update_request
  - Show qos request point with request value

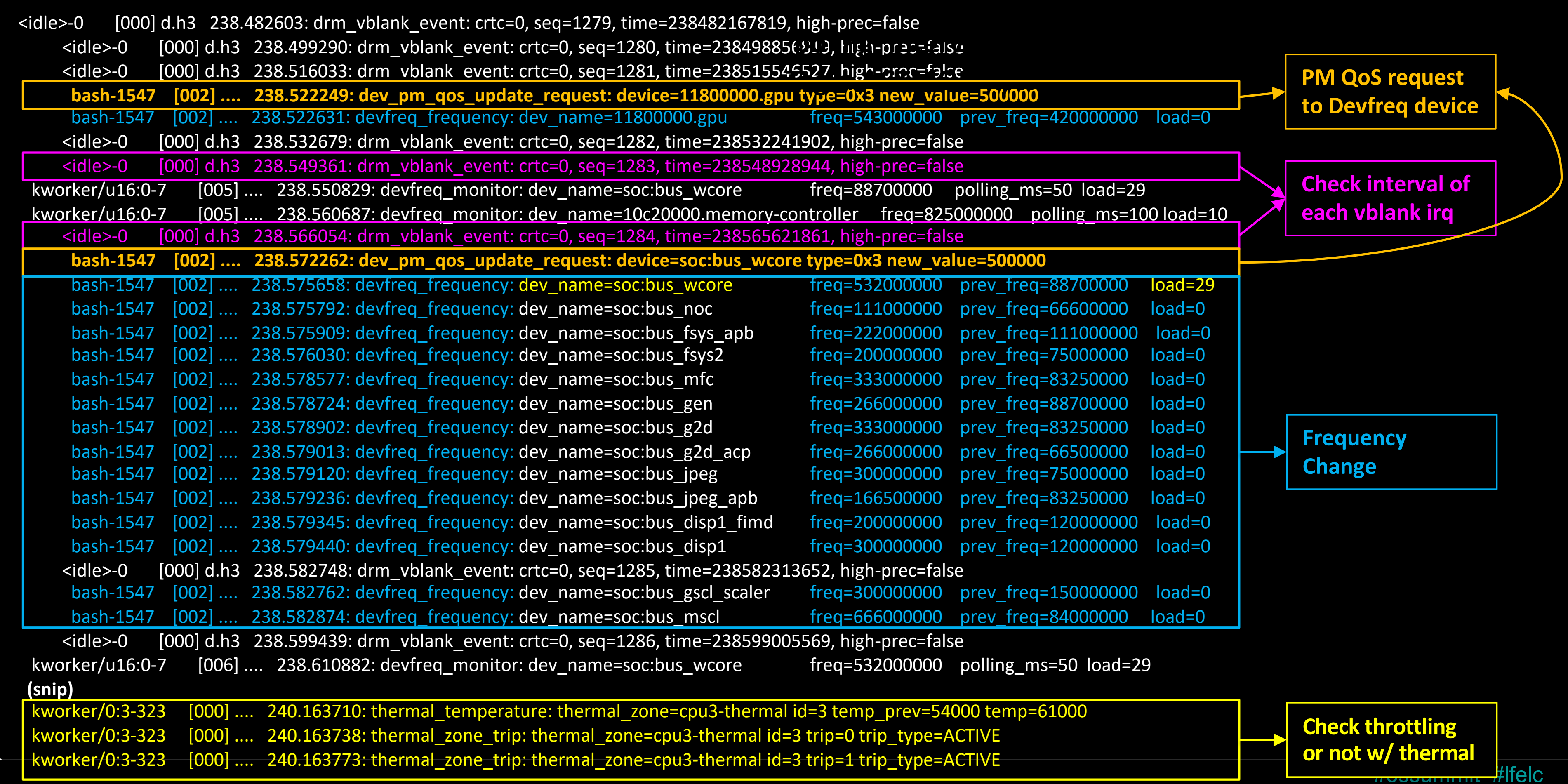
Simple Profiling - Tracepoint

- How to use it for performance profiling

  1. Enable tracepoint of devfreq, PM QoS and thermal

  2. **Enable tracepoint of performance-sensitive devices**
     - DRM for display controller
       - if 60 fps is required, each vblank interrupt must happen within approximate 16 ms.
     - V4L2 for video playback
     - Storage access latency

Simple Profiling – Tracepoint Example



```
<idle>-0     [000] d.h3  238.482603: drm_vblank_event: crtc=0, seq=1279, time=238482167819, high-prec=false
     <idle>-0     [000] d.h3  238.499290: drm_vblank_event: crtc=0, seq=1280, time=238498856819, high-prec=false
     <idle>-0     [000] d.h3  238.516033: drm_vblank_event: crtc=0, seq=1281, time=238515546527, high-prec=false
      bash-1547   [002] ....  238.522249: dev_pm_qos_update_request: device=11800000.gpu type=0x3 new_value=500000
      bash-1547   [002] ....  238.522631: devfreq_frequency: dev_name=11800000.gpu        freq=543000000   prev_freq=420000000   load=0
     <idle>-0     [000] d.h3  238.532679: drm_vblank_event: crtc=0, seq=1282, time=238532241902, high-prec=false
     <idle>-0     [000] d.h3  238.549361: drm_vblank_event: crtc=0, seq=1283, time=238548928944, high-prec=false
  kworker/u16:0-7      [005] ....  238.550829: devfreq_monitor: dev_name=soc:bus_wcore        freq=88700000   polling_ms=50  load=29
  kworker/u16:0-7      [005] ....  238.560687: devfreq_monitor: dev_name=10c20000.memory-controller   freq=825000000   polling_ms=100 load=10
     <idle>-0     [000] d.h3  238.566054: drm_vblank_event: crtc=0, seq=1284, time=238565621861, high-prec=false
      bash-1547   [002] ....  238.572262: dev_pm_qos_update_request: device=soc:bus_wcore type=0x3 new_value=500000
      bash-1547   [002] ....  238.575658: devfreq_frequency: dev_name=soc:bus_wcore        freq=532000000   prev_freq=88700000   load=29
      bash-1547   [002] ....  238.575792: devfreq_frequency: dev_name=soc:bus_noc          freq=111000000   prev_freq=66600000   load=0
      bash-1547   [002] ....  238.575909: devfreq_frequency: dev_name=soc:bus_fsys_apb     freq=222000000   prev_freq=111000000   load=0
      bash-1547   [002] ....  238.576030: devfreq_frequency: dev_name=soc:bus_fsys2        freq=200000000   prev_freq=75000000   load=0
      bash-1547   [002] ....  238.578577: devfreq_frequency: dev_name=soc:bus_mfc          freq=333000000   prev_freq=83250000   load=0
      bash-1547   [002] ....  238.578724: devfreq_frequency: dev_name=soc:bus_gen          freq=266000000   prev_freq=88700000   load=0
      bash-1547   [002] ....  238.578902: devfreq_frequency: dev_name=soc:bus_g2d          freq=333000000   prev_freq=83250000   load=0
      bash-1547   [002] ....  238.579013: devfreq_frequency: dev_name=soc:bus_g2d_acp      freq=266000000   prev_freq=66500000   load=0
      bash-1547   [002] ....  238.579120: devfreq_frequency: dev_name=soc:bus_jpeg         freq=300000000   prev_freq=75000000   load=0
      bash-1547   [002] ....  238.579236: devfreq_frequency: dev_name=soc:bus_jpeg_apb     freq=166500000   prev_freq=83250000   load=0
      bash-1547   [002] ....  238.579345: devfreq_frequency: dev_name=soc:bus_disp1_fimd   freq=200000000   prev_freq=120000000   load=0
      bash-1547   [002] ....  238.579440: devfreq_frequency: dev_name=soc:bus_disp1        freq=300000000   prev_freq=120000000   load=0
     <idle>-0     [000] d.h3  238.582748: drm_vblank_event: crtc=0, seq=1285, time=238582313652, high-prec=false
      bash-1547   [002] ....  238.582762: devfreq_frequency: dev_name=soc:bus_gscl_scaler  freq=300000000   prev_freq=150000000   load=0
      bash-1547   [002] ....  238.582874: devfreq_frequency: dev_name=soc:bus_mscl         freq=666000000   prev_freq=84000000   load=0
     <idle>-0     [000] d.h3  238.599439: drm_vblank_event: crtc=0, seq=1286, time=238599005569, high-prec=false
  kworker/u16:0-7      [006] ....  238.610882: devfreq_monitor: dev_name=soc:bus_wcore        freq=532000000   polling_ms=50  load=29
 (snip)
  kworker/0:3-323     [000] ....  240.163710: thermal_temperature: thermal_zone=cpu3-thermal id=3 temp_prev=54000 temp=61000
  kworker/0:3-323     [000] ....  240.163738: thermal_zone_trip: thermal_zone=cpu3-thermal id=3 trip=0 trip_type=ACTIVE
  kworker/0:3-323     [000] ....  240.163773: thermal_zone_trip: thermal_zone=cpu3-thermal id=3 trip=1 trip_type=ACTIVE
```

PM QoS request to Devfreq device

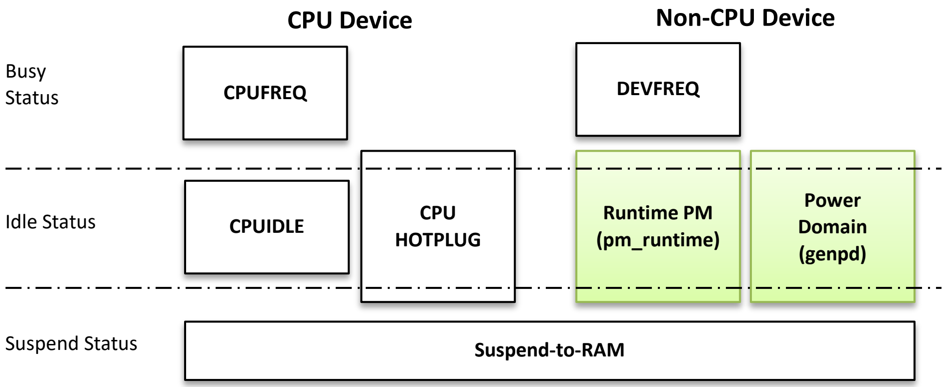
Check interval of each vblank irq

Frequency Change

Check throttling or not w/ thermal

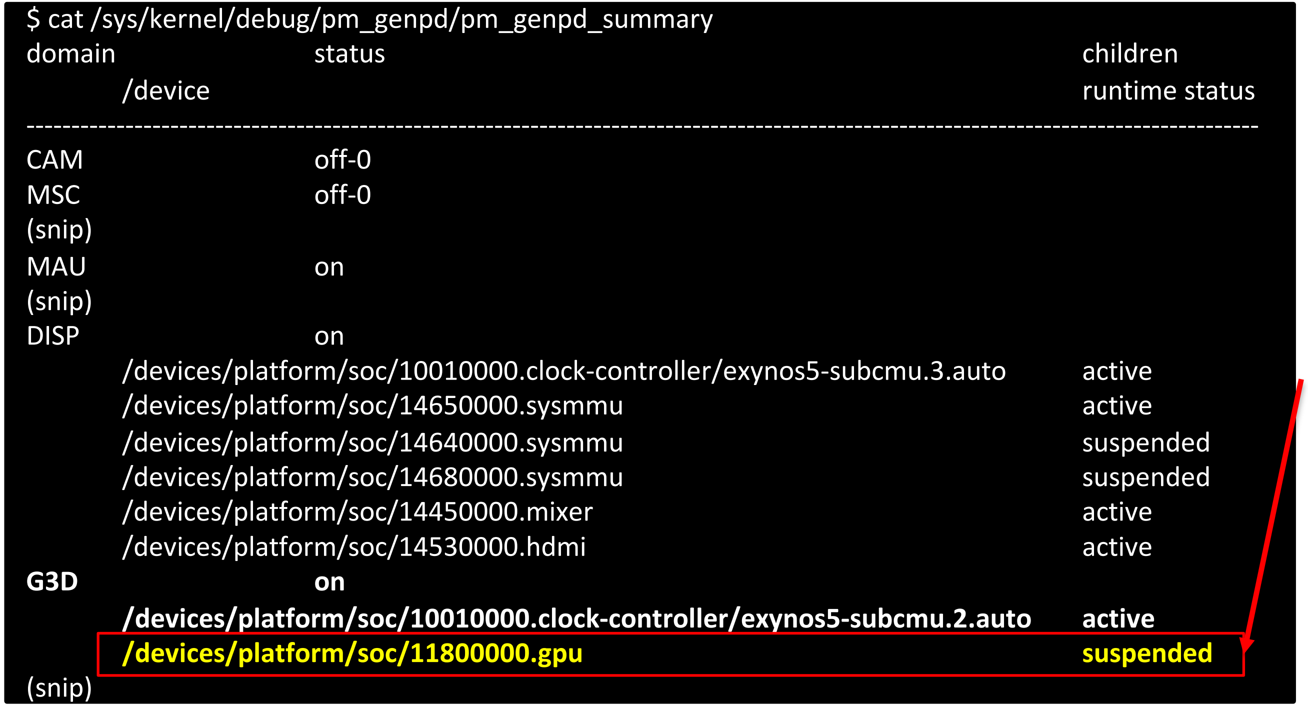
Simple Profiling - Device Status

- Runtime PM
  - /sys/devices/platform/soc/[device name]/power/runtime_status
- Generic Power Domain
  - /sys/kernel/debug/pm_genpd/pm_genpd_summary



Simple Profiling - Check device status via Debugfs

- Can check the device status as following:



```
$ cat /sys/kernel/debug/pm_genpd/pm_genpd_summary
domain                    status                                                          children
        /device                                                                           runtime status
------------------------------------------------------------------------------------------------------------
CAM                       off-0
MSC                       off-0
(snip)
MAU                       on
(snip)
DISP                      on
        /devices/platform/soc/10010000.clock-controller/exynos5-subcmu.3.auto             active
        /devices/platform/soc/14650000.sysmmu                                             active
        /devices/platform/soc/14640000.sysmmu                                             suspended
        /devices/platform/soc/14680000.sysmmu                                             suspended
        /devices/platform/soc/14450000.mixer                                              active
        /devices/platform/soc/14530000.hdmi                                               active
G3D                       on
        /devices/platform/soc/10010000.clock-controller/exynos5-subcmu.2.auto             active
        /devices/platform/soc/11800000.gpu                                                suspended
(snip)
```

Test device : Odroid-XU3 (Samsung Exynos5422 SoC)

The devfreq_summary shows ‘GPU’ devfreq device.

But, when try to print tracepoint for GPU, doesn’t work, even If GPU device uses simple_ondemand with timer.

Because GPU device status is suspended by runtime PM.

Support runtime PM for device to reduce power waste

- DEVFREQ provides the following governor helper function to control governor according to device power status

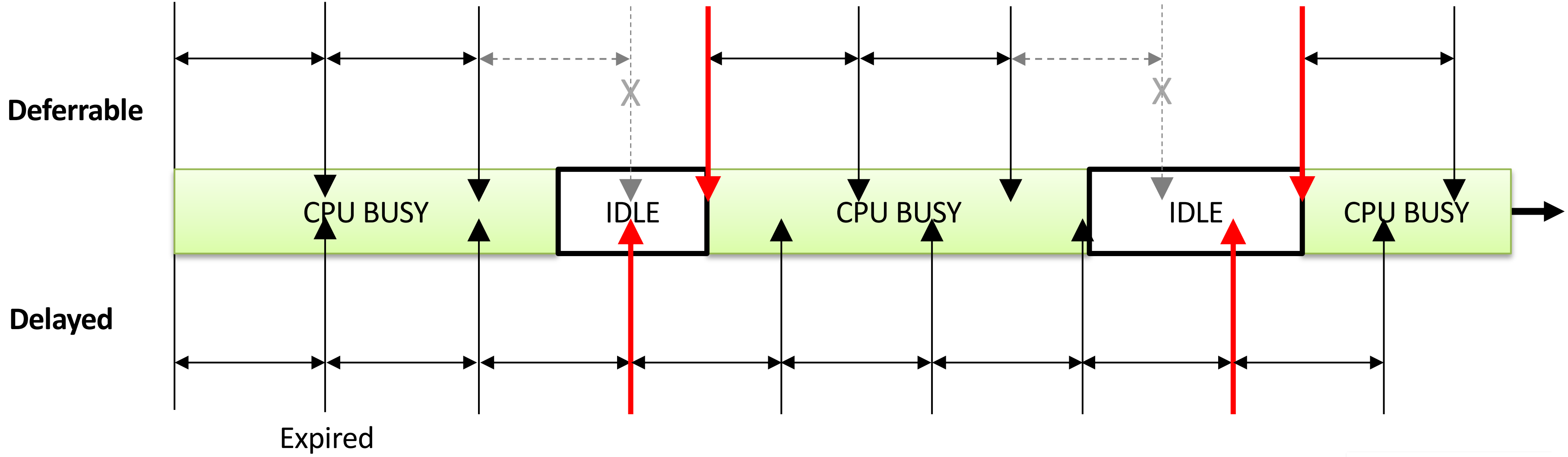
Governor Helper Function	Governor Status	Description
devfreq_monitor_start(struct devfreq)	DEVFREQ_GOV_START	Start governor for devfreq device
devfreq_monitor_stop(struct devfreq)	DEVFREQ_GOV_STOP	Stop governor for devfreq device
devfreq_monitor_suspend(struct devfreq)	DEVFREQ_GOV_SUSPEND	Suspend governor for devfreq device If this has 'suspend-opp' property in devicetree, set suspend frequency indicatd by 'suspend-opp.
devfreq_monitor_resume(struct devfreq)	DEVFREQ_GOV_RESUME	Resume governor for devfreq device If this has 'suspend-opp' property in devicetree, recover the last frequency
devfreq_update_interval(struct devfreq)	DEVFREQ_GOV_UPDATE_INTERVAL	Update polling interval by devfreq device driver instead of sysfs interface

Performance Tuning Point - Sysfs

- timer and polling_interval
  - echo (deferrable|delayed) > /sys/class/devfreq/[dev name]/timer
    - **deferrable** timer is not expired if CPU idle.
    - **delayed** timer doesn’t care CPU status.
  - echo [positive integer] > /sys/class/devfreq/[dev name]/polling_interval
- up_threshold and down_differential
  - echo [0-100] > /sys/class/devfreq/[dev name]/up_threshold
  - echo [0-100] > /sys/class/devfreq/[dev name]/down_differential

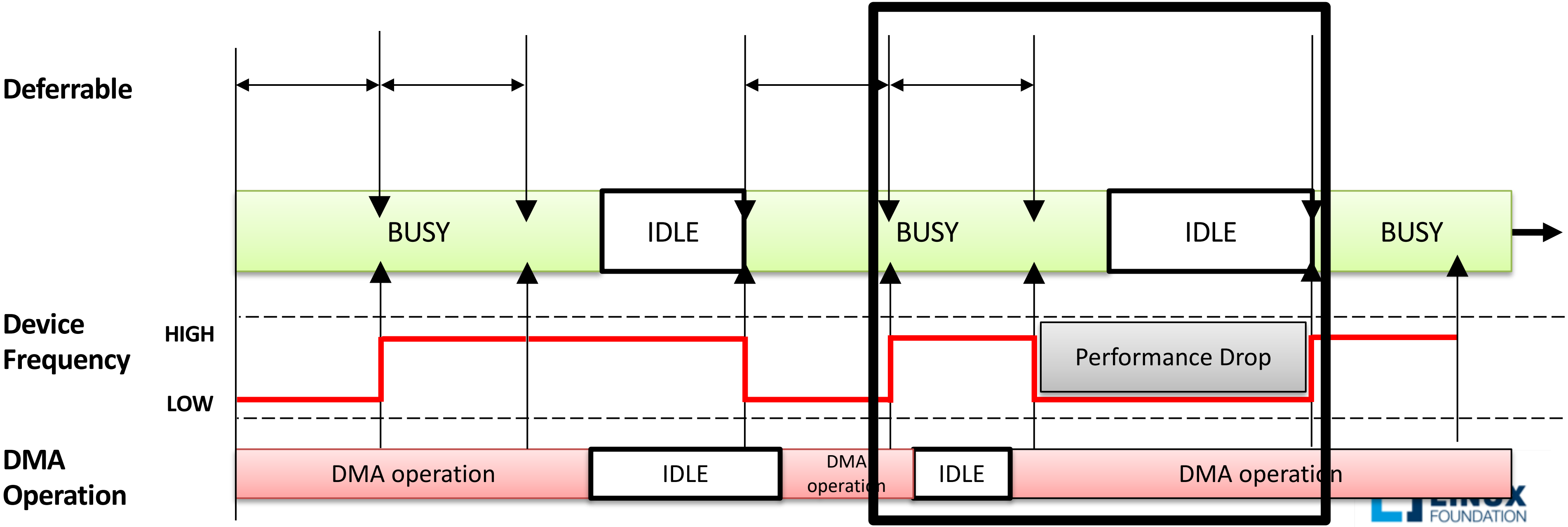
Performance Tuning Point - Deferrable vs. Delayed

- Difference between deferrable vs. delayed timer
  - Need CONFIG_HIGH_RES_TIMERS and CONFIG_NO_HZ for derrable timer



Performance Tuning Point - Deferrable timer’s bad case

- In case of DMA without CPU operation,
  DMA operation transfer data between memory and device.



Performance Tuning Point - Sysfs

- timer and polling_interval

timer	polling_interval	Recommendation use-case But, it is not always true.
deferrable	long	- The non-cpu device is related to CPU status - Don't want to wakeup CPU due to dev monitoring
deferrable	short	- Don't want to wakeup CPU due to dev monitoring **-** Need to react fastly on CPU busy
delayed	long	- The non-cpu device is less or not related to CPU status like DMA.
delayed	**short**	- The non-cpu device is less or not related to CPU status like DMA. - Need to react fastly always

- up_threshod and down_differential

Freq Up Speed	Freq Down Speed	up_threshold	down_differential	Performance vs. Low Power
Fastly	Fastly	low	high	-
Fastly	***Slowly***	***low***	***low***	***Highest Performance***
Slowly	Fastly	high	high	-
Slowly	***Slowly***	***high***	***low***	***Lowest Power***

Performance Tuning Point - Summary

timer	polling _interval	up _threshold	down _differential	min_freq	max_freq	PM QoS Request	Recommendation use-case But, it is not always true.
How often monitoring dev in accordance with CPU busy or idle	**How often monitoring dev on CPU busy for reactivly**	**Frequency Up Speed for reactivity**	**Frequency Down Speed for reactivity**	**Frequency Boosting for high-performance**	**Resource Limiting for low-power or high-temperature**		
-	-	-	-	-	-	-	Powersave Govenor
deferrable	long	high	low	default	Lower max_freq close to min_freq	-	Lowest Power
deferrable	short	-	-	Non-Aggressive Use			- The non-cpu device is related to CPU status and also don't want to wakeup CPU due to dev monitoring. - polling_interval is short, it will be reacted as soon as possible.
deferrable	long	-	-	Aggressive Use			- Keep the lower power on almost case and want to support high-performance on fixed scenario use-case.
delayed	short	low	low	Aggressive Use			- The non-cpu device is less or not related to CPU status like DMA. - Fastly frequency up and slowsy frequency down. - Never permit the performance drop for specific scenario.
delayed	short	low	low	Higher min_freq close to max_freq	default	Aggressive use	Highest Performance
-	-	-	-	-	-		Performance Governor

Use-Case in Mainline Kernel

DEVFREQ Driver in Mainline Kernel (1/2)

- GPU
- ARM AMBA Bus
- DMC (Dynamic Memory Controller)
- UFS (Universal Flash Storage) Storage
- L2 Cache
  - Recently, mainline posted for Qualcomm Krait L2 Cache and under review.

DEVFREQ Driver in Mainline Kernel (2/2)

Device Type	Driver Path	SoC Vendor	Used Governor	Description
GPU	drivesr/gpu/drm/panfrost/panfrost_devfreq.c	ARM	simple_ondemand	Almost driver have been using **simple_ondemand** governor. It means that must **need to improve simple_ondemand** governor or **suggest new innovative governor** like cpufreq schedutil governor.
	drivesr/gpu/drm/lima/lima_devfreq.c	ARM	simple_ondemand	
	drivesr/gpu/drm/msm/msm_gpu.c	Qualcomm	simple_ondemand	
Memory Controller	drivers/memory/samsung/exynos5422-dmc.c	Samsung	simple_ondemand	
	driver/devfreq/imx8m-ddrc.c	NXP	simple_ondemand	
	drivers/devfreq/rk3399_dmc.c	Rockchip	simple_ondemand	
Memory Data Bus	drivers/devfreq/exynos-bus.c	Samsung	simple_ondemand *passive*	
	drivers/devfreq/imx-bus.c	NXP	simple_ondemand	
Storage (UFS)	drivers/scsi/ufs/ufshcd.c	Generic device	simple_ondemand	
Specific SoC Device	drivers/devfreq/tegra30_devfreq.c	Nvidia	*tegra_actmon*	
	drivers/devfreq/tegra20_devfreq.c	Nvidia	*tegra_actmon*	

Weakness of DEVFREQ & Further TODO

What are Weakness of DEVFREQ?

- **Too old governor** based on timer-based sampling method. Need new governor or method to monitor device for immediate response.
  - Schedutil governor of CPUFREQ framework

- **Too simply checking the device status at that time** without considering history and don’t expect future device status to prevent performance drop.
  - PELT (Per-Entity Load Tracking) of CPU scheduler
  - Ladder governor of CPUIDLE framework

Further ToDo

- Support ‘**required-opp**’ property of OPP to specify the correct pair between parent and passive device.
- Expand ‘**passive**’ **governor depend on CPU Frequency**
  - In the mainline, there are many requirement about this. But, it has not yet completed.
- For more immediate response, support **kthread-based timer**
  - DEVFREQ_TIMER_WQ_DEFERRABLE : Deferrrable work
  - DEVFREQ_TIMER_WQ_DELAYED : Delayed work
  - DEVFREQ_TIMER_KTHREAD : Kthread with SCHED_NORMAL
  - DEVFREQ_TIMER_KTHREAD_RT : Ktherad with SCHED_FIFO
- Need **kselfset** for DEVFREQ device

Appendix

- ‘devfreq_frequency’ tracepoint patch (merged to devfreq.git)
  - https://git.kernel.org/pub/scm/linux/kernel/git/chanwoo/linux.git/log/?h=devfreq-next
- [v4,0/2] PM / devfreq: Add governor feature and attribute flag
  - https://patchwork.kernel.org/project/linux-pm/cover/20201020030407.21047-1-cw00.choi@samsung.com/
- up_threshold and down_differential patch (not yet posted, but can refer to it)
  - https://git.kernel.org/pub/scm/linux/kernel/git/chanwoo/linux.git/log/?h=devfreq-testing

Embedded Linux Conference

Europe