# Linux Thermal
# 开发指南

**版本号: 1.6**
**发布日期: 2021.08.05**

# 版本历史

| 版本号 | 日期 | 制/修订人 | 内容描述 |
|---|---|---|---|
| 1.0 | 2020.12.24 | AWA0863 | 1. 添加初始版本。 |
| 1.1 | 2021.02.02 | AWA0863 | 1. 补充常见问题章节。 |
| 1.2 | 2021.05.11 | AWA0863 | 1. 增加参数测量介绍章节。 |
| 1.3 | 2021.06.01 | AWA1442 | 1. 补充部分 thermal 参数的测试方法。<br>2. 补充常见问题章节。 |
| 1.4 | 2021.06.07 | AWA1442 | 1. 更新 thermal 参数的测试方法。<br>2. 补充常见问题章节。 |
| 1.5 | 2021.06.25 | AWA1442 | 1. 更正 gpu 参数计算方法。 |
| 1.6 | 2021.08.05 | AWA0863 | 1. 增加 Linux4.9 和 Linux5.4 的内核配置。 |

# 目　录

# 1 前言

## 1.1 文档简介

该使用文档介绍了 thermal 的温控策略配置方法，以及调试使用说明。

## 1.2 目标读者

thermal 模块开发、维护人员。

## 1.3 适用范围

表 1-1：适用产品列表

| 产品名称 | 内核版本 | 驱动文件 |
| --- | --- | --- |
| T509 | Linux-4.9 | drivers/thermal/* |
| MR813 | Linux-4.9 | drivers/thermal/* |
| R818 | Linux-4.9 | drivers/thermal/* |
| A133 | Linux-4.9/5.4 | drivers/thermal/* |
| R528 | Linux-5.4 | drivers/thermal/* |
| D1 | Linux-5.4 | drivers/thermal/* |

# 2 模块介绍

## 2.1 模块功能介绍

Thermal 俗称热控制系统，其功能是通过 temperature sensor 测量当前 CPU、GPU 等设备的温度值，然后根据此温度值，影响 CPU、GPU 等设备的调频策略，对 CPU、GPU 等设备的最大频率进行限制，最终实现对 CPU、GPU 等设备温度的闭环控制，避免 SOC 温度过高。

IPA(Intelligent Power allocator) 温控策略：引入 PID 控制，根据系统温度动态分配 power 给各个设备，并将 power 转化为频率限制。

## 2.2 相关术语介绍

表 2-1: 术语介绍

| 术语 | 说明 |
| --- | --- |
| Temperature sensor | 温度传感器。 |
| Thermal | CPU 温度控制系统。 |
| CPU | 中央处理器。 |
| GPU | 图像处理器。 |
| thermal zone | 将提供温度及 trip 点相关信息给 themal core 子系统。 |
| cooling device | themal core 子系统通过 cooling device 对 CPU、GPU 等设备最大频率进行限制。 |

## 2.3 模块配置介绍

### 2.3.1 Device Tree 配置说明

设备树中存在的是该类芯片所有平台的模块配置，设备树文件的路径为：kernel/linux-4.9/arch/arm64（32 位平台为 arm）/boot/dts/sunxi/CHIP.dtsi(CHIP 为研发代号，如 sun50iw10p1 等)。

参考文档：

linux-4.9/Documentation/thermal/power_allocator.txt
linux-4.9/Documentation/devicetree/bindings/thermal/thermal.txt
linux-4.9/Documentation/thermal/cpu-cooling-api.txt
或
linux-5.4/Documentation/driver-api/thermal/power_allocator.rst
linux-5.4/Documentation/devicetree/bindings/thermal/thermal.txt
linux-5.4/Documentation/driver-api/thermal/cpu-cooling-api.rst

- of-thermal

在 thermal 模块开发中，只需要将 thermal zone、thermal Sensor、trip point、cooling Device 的关系在 DTS 文件内按照规定的格式描述，of-thermal 模块就会根据 DTS 将描述的内容自动注册，逻辑关系由 of-thermal 模块维护，使驱动代码量大大减少。

```
thermal-zones{
    cpu_thermal_zone{
        polling-delay-passive = <500>;      //温度超过阈值，轮询温度周期(ms)
        polling-delay = <1000>;             //温度未超过阈值，轮询温度周期(ms)
        thermal-sensors = <&ths 2>;
        sustainable-power = <1000>;         //温度达到预设温度最大值，系统可分配的最大power
        k_po = <25>;                        //超过预设最高温度时pid的p参数
        k_pu = <50>;                        //未超过预设最高温度时pid的p参数
        k_i = <0>;                          //pid的i参数

        cpu_trips: trips{
            cpu_threshold: trip-point@0 {
                temperature = <70000>;      //代表系统温控在70度左右开启
                type = "passive";
                hysteresis = <0>;
            };
            cpu_target: trip-point@1 {
                temperature = <80000>;      //代表系统最高温度是80度左右
                type = "passive";
                hysteresis = <0>;
            };
            cpu_crit: cpu_crit@0 {
                temperature = <110000>;     //代表系统到达110度就会过温关机
                type = "critical";
                hysteresis = <0>;
            };
        };

        cooling-maps {
            map0 {
                trip = <&cpu_target>;
                cooling-device = <&cpu0
                THERMAL_NO_LIMIT
                THERMAL_NO_LIMIT>;
                contribution = <1024>;      //cpu分配功率权重，通过调整cooling device对应的
    contribution，可以调整降频顺序和降频尺度
            };
            map1{
                trip = <&cpu_target>;
                cooling-device = <&gpu
                THERMAL_NO_LIMIT
```

```
                THERMAL_NO_LIMIT>;
                contribution = <1024>;      //gpu分配功率权重，通过调整cooling device对应的
    contribution，可以调整降频顺序和降频尺度
            };
        };
    };

    gpu_thermal_zone{
        polling-delay-passive = <500>;
        polling-delay = <1000>;
        thermal-sensors = <&ths 0>;
    };

    ve_thermal_zone{
        polling-delay-passive = <0>;
        polling-delay = <0>;
        thermal-sensors = <&ths 1>;
    };

    ddr_thermal_zone{
        polling-delay-passive = <0>;
        polling-delay = <0>;
        thermal-sensors = <&ths 3>;
    };
};

cpu_target节点中的temperature:
可根据产品温控规格，适当调整该参数。提高该参数，会允许系统在高温情况下运行更快，性能更好。
当然，也会让产品的温度更高，所以需要注意，修改该参数后能否满足产品温度要求和高温测试等。
同理，降低该参数就会在一定程度上降低高温情况下的性能，可以让产品运行在较低的温度。
```

- Thermal driver

```
ths: thermal_sensor{
    compatible = "arm,sun50iw9p1";
    reg = <0x0 0x05070400 0x0 0x400>;
    clocks = <&clk_ths>;
    clock-names = "bus";
    nvmem-cells = <&ths_calib>;
    nvmem-cell-names = "calibration";
    #thermal-sensor-cells = <1>;
};
```

- Cooling device

cpu device

```
cpu0: cpu@0 {
    device_type = "cpu";
    compatible = "arm,cortex-a53","arm,armv8";
    reg = <0x0 0x0>;
    enable-method = "psci";
    clocks = <&clk_pll_cpu>;
```

```
    operating-points-v2 = <&cpu_opp_l_table>;
    cpu-idle-states = <&CPU_SLEEP_0>;
    dynamic-power-coefficient = <100>;
    #cooling-cells = <2>;
};

dynamic-power-coefficient: cpu动态功耗系数，由P = c * v * v * f / 1000000得来（参数c就是动态功耗系
    数）。
```

## cpu opp table

```
cpu_opp_l_table: opp_l_table {
    compatible = "operating-points-v2";
    opp-shared;

    opp@408000000 {
        opp-hz = /bits/ 64 <408000000>;
        opp-microvolt = <820000>;
        clock-latency-ns = <244144>; /* 8 32k periods */
    };
    opp@816000000 {
        opp-hz = /bits/ 64 <816000000>;
        opp-microvolt = <880000>;
        clock-latency-ns = <244144>; /* 8 32k periods */
    };
    opp@1008000000 {
        opp-hz = /bits/ 64 <1008000000>;
        opp-microvolt = <940000>;
        clock-latency-ns = <244144>; /* 8 32k periods */
    };
    opp@1200000000 {
        opp-hz = /bits/ 64 <1200000000>;
        opp-microvolt = <1020000>;
        clock-latency-ns = <244144>; /* 8 32k periods */
    };
    opp@1416000000 {
        opp-hz = /bits/ 64 <1416000000>;
        opp-microvolt = <1100000>;
        clock-latency-ns = <244144>; /* 8 32k periods */
    };
};
opp-hz: 频率
opp-microvolt: 频率对应的电压
```

## gpu device

```
gpu: gpu@0x01800000 {
    device_type = "gpu";
    compatible = "arm,mali-midgard";
    reg = <0x0 0x01800000 0x0 0x10000>;
    interrupts = <GIC_SPI 95 IRQ_TYPE_LEVEL_HIGH>,
             <GIC_SPI 96 IRQ_TYPE_LEVEL_HIGH>,
             <GIC_SPI 97 IRQ_TYPE_LEVEL_HIGH>;
    interrupt-names = "JOB", "MMU", "GPU";
    clocks = <&clk_pll_gpu>, <&clk_gpu0>, <&clk_gpu1>;
    clock-names = "clk_parent", "clk_mali", "clk_bak";
```

```
    #cooling-cells = <2>;
    ipa_dvfs:ipa_dvfs {
        compatible = "arm,mali-simple-power-model";
        static-coefficient = <17000>;
        dynamic-coefficient = <750>;
        ts = <254682 9576 0xffffff98 4>;
        thermal-zone = "gpu_thermal_zone";
        ss-coefficient = <36>;
        ff-coefficient = <291>;
    };
};
static-coefficient:  gpu静态功耗计算系数
dynamic-coefficient:  gpu动态功耗计算系数
```

gpu dvfs

```
gpu dvfs表:
gpu: gpu@0x01800000 {
        gpu_idle = <1>;
        dvfs_status = <1>;
        operating-points = <
             /* KHz   uV */
            600000 950000
            576000 950000
            540000 950000
            504000 950000
            456000 950000
            420000 950000
            384000 950000
            360000 950000
            336000 950000
            306000 950000
         >;
};
```

## 2.3.2 board.dts 配置说明

board.dts 用于保存每一个板级平台的设备信息（如 demo 板，perf1 板等），里面的配置信息会覆盖上面的 Device Tree 默认配置信息。thermal 模块在 board.dts 中无用户可用配置。

## 2.3.3 sysconfig 配置说明

thermal 模块在 sysconfig 中无用户可用配置。

## 2.3.4 kernel menuconfig 配置说明

对于 Linux4.9，进入内核根目录，执行 make ARCH=arm menuconfig（64 位平台为 make ARCH=arm64 menuconfig）；对于 Linux5.4，进入 longan 根目录，执行./build.sh menu-

config。

首先，进入到 Device Drivers ->Generic Thermal sysfs driver，如下所示：

对于 Linux4.9，menuconfig 界面配置如下：

```
Device Drivers  --->
    [*] Generic Thermal sysfs driver  --->
        --- Generic Thermal sysfs driver
        [*]   Expose thermal sensors as hwmon device
        [*]   APIs to parse thermal data out of device tree
        [ ]   Enable writable trip points
              Default Thermal governor (user_space)  --->
        [ ]   Fair-share thermal governor
        [ ]   Step_wise thermal governor
        [ ]   Bang Bang thermal governor
        -*-   User_space thermal governor
        [*]   Power allocator thermal governor
        [*]   generic cpu cooling support
        [ ]   Generic clock cooling support
        [*]   Generic device cooling support
        [*]   Thermal emulation mode support
        < >   QorIQ Thermal Monitoring Unit
        <*>   Allwinner sunxi next genaration thermal driver
```

对于 Linux5.4，menuconfig 界面配置如下：

```
Device Drivers  --->
    [*] Generic Thermal sysfs driver  --->
        --- Generic Thermal sysfs driver
        [*]   Thermal state transition statistics
        (0)   Emergency poweroff delay in milli-seconds
        [*]   APIs to parse thermal data out of device tree
        [*]   Enable writable trip points
              Default Thermal governor (step_wise)  --->
        -*-   Step_wise thermal governor
        [*]   User_space thermal governor
        [*]   Power allocator thermal governor
        [*]   Generic cpu cooling support
        [*]   Generic device cooling support
        [*]   Thermal emulation mode support
        <*>   Allwinner sunxi thermal driver
```

配置项说明：

```
Default Thermal governor 选择，默认为power_allocator
Thermal emulation mode support:支持thermal模拟温度功能
generic cpu cooling support :打开通用的cpu cooling
generic device cooling support:打开通用的device cooling
Allwinner sunxi next generation thermal driver:sunxi thermal驱动
```

# 2.4 参数测量介绍

温控参数汇总如下。

表 2-2: cpu cooling device 参数

| 参数 | 说明 |
| --- | --- |
| contribution | cpu 分配功率权重，可以不定义使用预设值，也可以根据方案实测通过 dts 或文件系统节点配置 |
| dynamic-power-coefficient | cpu 动态功耗系数，根据芯片实测通过 dts 配置 |
| static | cpu 静态功耗系数 |

表 2-3: gpu cooling device 参数

| 参数 | 说明 |
| --- | --- |
| contribution | gpu 分配功率权重，可以不定义使用预设值，也可以根据方案实测通过 dts 或文件系统节点配置 |
| dynamic-coefficient | gpu 动态功耗系数，根据芯片实测通过 dts 配置 |
| static-coefficient | gpu 静态功耗系数，根据芯片实测通过 dts 配置 |
| ts | gpu 静态功耗系数，根据芯片实测通过 dts 配置 |
| ss-coefficient | |
| ff-coefficient | |

表 2-4: thermal zone 参数

| 参数 | 说明 |
| --- | --- |
| temperature (threshold) | ipa 温控策略介入温度，根据方案规格要求通过 dts 或文件系统节点配置 |
| temperature (target) | ipa 温控策略目标温度，根据方案规格要求通过 dts 或文件系统节点配置 |
| temperature (crit) | 过温关机温度，根据方案规格要求通过 dts 或文件系统节点配置 |
| sustainable-power | 温度达到目标温度时，系统可分配的最大功率，根据方案规格要求通过 dts 或文件系统节点配置 |
| k_po | pid 的 p 参数，可以不定义使用预设值，也可以根据方案实测通过文件系统节点配置 |
| k_pu | pid 的 p 参数，可以不定义使用预设值，也可以根据方案实测通过文件系统节点配置 |
| k_i | pid 的 i 参数，可以不定义使用预设值，也可以根据方案实测通过文件系统节点配置 |

| 参数 | 说明 |
| --- | --- |
| k_d | pid 的 d 参数，可以不定义使用预设值，也可以根据方案实测通过文件系统节点配置 |

## 2.4.1 cpu cooling device 参数

### 2.4.1.1 contribution

cpu 分配功率权重，如果方案规格没有特别要求，可以设置为 <1024>。

通过调整 cooling device 对应的 contribution，可以调整降频顺序和降频尺度。假如在高温下，cpu 和 gpu 都满负载运行，发现 cpu 更容易被降频，如果想让 gpu 优先降频，可以增大 cpu 的 contribution，比如修改为 <4096>。

### 2.4.1.2 dynamic-power-coefficient

**cpu 动态功耗系数，只与芯片平台相关，方案不需要修改。**

计算方法有如下。

1. cpu 硬件负责人提供 cpu 最大功率下的功耗数据。

| A013 (VDD = 0.9V) | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- |
| CPU_Freq(MHz) | R_V (mV) | CPU_I(mA) | Power (mW) | V*V*Freq(V^2*MHz) | P-L(mW) | △P/△F (mW/MHz) | Temp(° C) |
| 888 | 103.96 | 1039.60 | 935.64 | 719.28 | 73.00 | | 49 |
| 936 | 109.82 | 1098.20 | 988.38 | 758.16 | 73.00 | 1.10 | 50 |
| 984 | 115.64 | 1156.40 | 1040.76 | 797.04 | 73.00 | 1.09 | 50 |
| 1032 | 121.26 | 1212.60 | 1091.34 | 835.92 | 73.00 | 1.05 | 52 |
| 1080 | 126.95 | 1269.50 | 1142.55 | 874.80 | 73.00 | 1.07 | 52 |
| 1128 | 132.65 | 1326.50 | 1193.85 | 913.68 | 73.00 | 1.07 | 53 |
| 1176 | 138.38 | 1383.80 | 1245.42 | 952.56 | 73.00 | 1.07 | 54 |
| 1224 | 144.08 | 1440.80 | 1296.72 | 991.44 | 73.00 | 1.07 | 54 |
| 1272 | 149.77 | 1497.70 | 1347.93 | 1030.32 | 73.00 | 1.07 | 55 |
| 1320 | 155.56 | 1555.60 | 1400.04 | 1069.20 | 73.00 | 1.09 | 56 |
| 1368 | 161.27 | 1612.70 | 1451.43 | 1108.08 | 73.00 | 1.07 | 57 |
| 1416 | 167.02 | 1670.20 | 1503.18 | 1146.96 | 73.00 | 1.08 | 57 |
| 1464 | 172.87 | 1728.70 | 1555.83 | 1185.84 | 73.00 | 1.10 | 59 |
| 1512 | 178.71 | 1787.10 | 1608.39 | 1224.72 | 73.00 | 1.10 | 60 |
| 1560 | 184.52 | 1845.20 | 1660.68 | 1263.60 | 73.00 | 1.09 | 60 |
| | | | | | | | |
| | | | | | | | |

图 2-1: cpu 最大功率下的功耗

2. 通过 Excel 工具，将 V^2*F 和 cpu 动态功耗 power 拟合，得到如下的线性折线图。

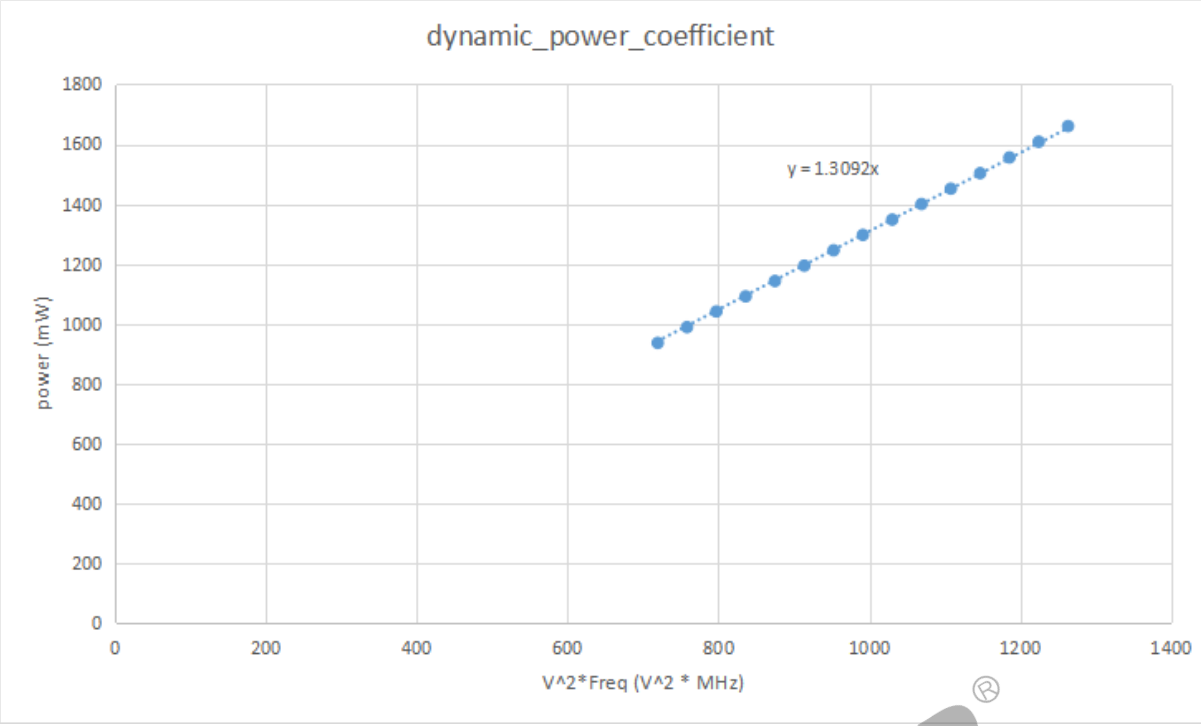


图 2-2: dynamic_power_coefficient 曲线

3. 假如 cpu 功耗数据是四核下的功耗数据，而我们实际需要的是单核的动态功耗系数。所以动态功耗系数 dynamic_power_coefficient = 1.3092/4 = 0.327。

动态功耗系数 dynamic_power_coefficient 需要配置在 Device Tree 的 cpu 节点中，为了计算方便，参数需要是整数。

```
cpu0: cpu@0 {
    device_type = "cpu";
    ......
    dynamic-power-coefficient = <327>;
    ......
};
```

所以，根据 cpu 动态功耗系数，也可以计算出 cpu 动态功耗。

```
cpu动态功耗计算公式: Pdyn = Capacitance * Voltage^2 * Frequency * Utilisation (linux-4.9/
    Documentation/thermal/cpu-cooling-api.txt)，其中，
Capacitance是动态功耗系数dynamic-power-coefficient;
Voltage为供电电压;
Frequency为运行频率;
Utilisation为利用率，一般为1;
```

#### 2.4.1.3 static

cpu 静态功耗系数，Linux4.16 及后续版本内核不再使用，详见如下提交：

```
commit 84fe2cab48590e4373978e4ef2031c977de98995
Author: Viresh Kumar <viresh.kumar@linaro.org>
Date:   Tue Dec 5 11:02:46 2017 +0530

    cpu_cooling: Drop static-power related stuff

    No one has used it for the last two and half years (since it was
    introduced by commit c36cf0717631 (thermal: cpu_cooling: implement the
    power cooling device API), get rid of it.

    Acked-by: Eduardo Valentin <edubezval@gmail.com>
    Signed-off-by: Viresh Kumar <viresh.kumar@linaro.org>
    Signed-off-by: Rafael J. Wysocki <rafael.j.wysocki@intel.com>
```

### 2.4.2 gpu cooling device 参数

#### 2.4.2.1 contribution

gpu 分配功率权重，如果方案规格没有特别要求，可以设置为 <1024>。

#### 2.4.2.2 gpu 其他参数

**gpu 的其他相关参数，只与芯片平台相关，方案不需要修改。**

gpu 的其他相关参数、动态功耗计算公式、静态功耗计算公式等，由 gpu 模块负责人直接提供，如有疑问，可以找相关负责人咨询。

### 2.4.3 thermal zone 参数

#### 2.4.3.1 temperature (target)

首先根据方案需求，确定温控策略的目标温度，一般考虑“性能”、“温升”等因素，分为以下几种情况：

1. 方案规格有明确的温度限制要求。如“播放 3 小时本地全景视频，产品表面温升 ≤15 度”。

这种情况下，目标温度可以设置为方案规格的最高温度。

2. 方案规格有明确的高温性能限制要求。如 “高温下 cpu 频率不能低于 1.4G”、“高温下 cpu 不能降频”、“高温下 cpu 保持最高性能 80%” 等。

这种情况下，让样机运行在方案规格指定的高温性能，使用 stability 在满负载场景下测试 5-10 分钟，得出 cpu 稳定温度。

- 若 cpu 稳定温度低于 90 摄氏度，目标温度可以设置为 90 摄氏度。
- 若 cpu 稳定温度高于 90 摄氏度，低于 110 摄氏度，目标温度可以设置为 cpu 稳定温度。
- 若 cpu 稳定温度高于 110 摄氏度，需要综合考虑规格合理性、芯片寿命和稳定性、硬件散热等情况，目标温度由方案硬件负责人确定。

3. 方案规格有明确的温度和高温性能限制要求。如 “cpu 最高温度不能超过 90 度，且 cpu 频率不能低于 1.4G”。

这种情况下，样机运行在方案规格指定的高温性能，使用 stability 在满负载场景下测试 5-10 分钟，得出 cpu 稳定温度。

- 若 cpu 稳定温度低于方案规格的最高温度，目标温度可以设置为方案规格的最高温度。
- 若 cpu 稳定温度高于方案规格的最高温度，需要综合考虑规格合理性、硬件散热等情况，目标温度由方案硬件负责人确定。

#### 2.4.3.2 temperature (threshold)

如果方案规格没有特别要求，温控策略介入温度可以设置为比温控策略目标温度低 10-15 摄氏度。

#### 2.4.3.3 temperature (crit)

方案规格有明确的过温关机要求，如 “支持 IC 过温保护：115 度, 断电关机功能”，过温关机温度可以设置为方案规格的指定值。
需要注意，过温关机温度需要比温控策略目标温度高 5-10 摄氏度，否则高温下很容易触发过温关机，温控策略也将失去意义。

#### 2.4.3.4 sustainable-power

sustainable-power 是指，温度达到目标温度时，系统可分配的最大功率。

IPA 策略最后在计算 CPU 可分配的 power 时会受 sustaibable_power 和 CPU 负载的影响。在 CPU 负载一定的情况下，同个温度点下 CPU 可分配的 power 与 sustainable-power 成正

比关系，即 sustainable-power 越大 CPU 可分配的 power 越大；在 sustainable-power 一定的情况下，同个温度点下 CPU 可分配的 power 与 CPU 负载成反比关系，即 CPU 负载越小，CPU 分配的 power 越大，即无法使用一个 sustainable-power 同时满足不同负载下的温控行为，需要根据实际情况选择不同的负载系数计算得出。
由此可以看出，在利用 CPU 作为 cooling-device 的时候，CPU 的负载情况会对 IPA 策略有较大影响，表现出来的可能是相同温度下 CPU 满载时会降频，而空载时迟迟不降频。因此，**在利用 CPU 作为 cooling-device 时，不建议搭配 performance 等不会根据负载调频的 cpufreq governor 使用，建议使用 ondemand 或 conservative 等会根据负载自动调频的 governor，利用 cpufreq governor 弥补 IPA 策略在 CPU 轻载时的控制不足。**

IPA 策略最终也是通过限制 cooling-deivce 的频率的来达到降温控温的效果，所以需要确保所有的 cooling-device 工作在最低频点下时系统的温度不会超过目标温度。如果因为环境温度过高或 cooling-device 本身发热过大，导致各 cooling-device 工作在最低频点时系统的温度依然超过目标温度，IPA 策略也无法起到降温控温的作用。

在基于上述认知的情况下，sustainable-power 的计算方法有如下两种。

- 方法一

简单来说，比如温控策略的目标温度是 90 摄氏度，如果 cpu 长期运行在 1.4G 频率下温度可以稳定在 90 摄氏度，那么 sustainable-power 就是 cpu 运行在 1.4G 频率下的功率。
当考虑有 gpu 等其他 cooling device 时，情况也是类似的，比如温控策略的目标温度是 90 摄氏度，如果 “cpu 运行在 1.4G，gpu 运行在 600M……” 温度可以稳定在 90 摄氏度，那么 sustainable-power 就是 “cpu 运行在 1.4G，gpu 运行在 600M……” 下的功率。

以仅有 cpu cooling device 为例，说明 sustainable-power 的计算方法。

```
首先确保各cooling-device均工作在最低频率点时设备的温度不会超过目标温度90摄氏度，然后再需要找到一个可以让设
    备稳定运行在目标温度为90摄氏度度的频率点。假如cpu运行在1608M频点，使用stability在满负载场景下测试5
    -10分钟，cpu温度基本在90度左右。所以sustainable-power采用了cpu工作在1608M时的功率。

cpu动态功耗计算：
根据公式Pdyn = Capacitance * Voltage^2 * Frequency * Utilisation (linux-4.9/Documentation/
    thermal/cpu-cooling-api.txt)，
假如Capacitance=0.225（dynamic-power-coefficient配置为225），cpu电压Voltage=1.04V，cpu频率
    Frequency=1608M，四核，cpu动态功耗为：
sustainable-power = K * cpu_dynamic_power    （0.5 * core_num  ≤ K ≤ core_num）
（K取CPU核数的一半时即根据CPU半载情况计算sustainable-power，取CPU核数时即根据CPU满载情况计算
    sustainable-power。系数K可根据实际情况进行微调，默认建议使用核数的一半进行计算。）

取CPU核数的一半进行计算，上述公式计算的结果为
sustainable-power = K * cpu_dynamic_power = 2 * 0.225 * 1608 * 1.04^2 = 782mW
```

若使用 sustainable-power=782 进行 stablity 测试，发现存在温度达到 92 度的情况，为了防止出现此类温度，后续将 sustainable-power 调整为 778，调整后基本消除了 cpu 达到 92 度的情况。

若各 cooling-device 长期运行在最高频点下跑 stability 依然无法使温度达到 target 温度，则选

取最高频点进行 sustainable-power 的计算。如：一个方案的目标温度是 90 摄氏度，该方案只有 cpu 作为 cooling-device，该 cpu 的最高频点是 1560MHz。将 cpu 频率设置为 1560MHz 后使用 stability 进行测试后发现 SOC 在该条件下最终温度稳定在 60 摄氏度，那依然取 cpu 在 1560MHz 下的数据计算 sustanable_power。

- 方法二

由于 IPA 策略使用的是 PID 控制算法，其对当前可分配能量的计算公式为：Power = sustainable-power + P + I + D，其中“I”作为累计误差只考虑了温度超过目标温度时候取值为负数的误差，且 I 值只有在温度低于介入温度（switch_on 温度）时才会被清零 reset。因此当温度如果比较长时间超过目标温度时，会导致“I”一直进行进行累积，当“I”累积值大于 sustainable-power + P 时，可分配的能量将一直为 0，cooling-device 将一直处于最低频率状态，直到温度降回介入温度。
因此，我们希望温度不要到了目标温度再进行降频，而是在介入温度和目标温度之间的某一个我们预期的温度点就进行降频降温，让系统的温度不会轻易达到目标温度，尽量不要让“I”参数的误差累积过大。

例如，我们设置的温控策略介入温度设置为 70 摄氏度，温控策略目标温度设置为 85 摄氏度，这表示系统在 70 摄氏度的时候会提供一个比较大的功率值，随着温度的升高，提供的功率逐渐减小，减到一定程度后开始降频，如果温度继续升高，功率继续降低，cpu 等 cooling device 频率也继续降低。所以超过 70 摄氏度的时候，只是获取温度的时间间隔缩短了，并不一定会降频，具体什么时候降频可以通过修改 sustainable-power 的值进行调整。

以某个平台为例，我们设置的温控策略介入温度设置为 70 摄氏度，温控策略目标温度设置为 85 摄氏度，温控策略限制频率的温度为 75 摄氏度，CPU（四核）和 GPU 都作为 cooling device。当温度超过 70 摄氏度，温控策略开始工作，即缩短获取温度的时间间隔，75 摄氏度的时候开始限制频率（比介入温度高 5 摄氏度，这样设置可以减小温控刚开始时频率波动的幅度），最终最高温度不超过 85 摄氏度。那么 75 摄氏度时的功率值就是所有 cooling device 的最大功率之和。
首先，要确保各 cooling-device 均工作在最低频率点时设备的温度不会超过目标温度，然后再根据我们的设定的目标温度进行下面的计算。

它们的最大功率之和计算如下：

```
cpu动态功耗计算：
根据公式Pdyn = Capacitance * Voltage^2 * Frequency * Utilisation (linux-4.9/Documentation/
    thermal/cpu-cooling-api.txt)，
假如Capacitance=0.100（dynamic-power-coefficient配置为100），cpu最大频点电压Voltage=1.125V，cpu最
    大频点频率Frequency=1416M，四核，cpu动态功耗为：
cpu_dynamic_power = 0.100 * 1416 * 1.125^2 = 179mW

gpu动态功耗计算：
根据公式根据公式Pdyn = Capacitance * Voltage^2 * Frequency（仅作举例，实际以gpu模块负责人的说法为
    准），
假如Capacitance=0.733（dynamic-power-coefficient配置为733），gpu最大频点电压Voltage=1.10V，gpu最
    大频点频率Frequency=800M，gpu动态功耗为：
gpu_dynamic_power = 0.733 * 800 * 1.10^2 = 709mW
```

```
gpu静态功耗计算：
根据以下公式（仅作举例，实际以gpu模块负责人的说法为准），
t_scale = (a * T^3) + (b * T^2) + (c * T) + d
v_scale = V^3
P(s)= ( C / 1000 ) * ( T_scale / 1000000 ) * V_scale
假如static-coefficient设置为411000，ts设置为32000 4700 -80 2，则C = 411000，a = 2，b = -80，c =
    4700，d = 32000，开始降频的温度值T = 75C，V = 1.1V，那么
t_scale = ( 2 * 75 * 75 * 75 ) + ( -80 * 75 * 75 ) + ( 4700 * 75 ) + 32000 = 778250
v_scale = 1.1 * 1.1 * 1.1 = 1.331
gpu_static_power = ( 411000 / 1000) * ( 778250 / 1000000 ) * 1.331 = 425mW

最大功率之和：P_max = K * cpu_dynamic_power + gpu_dynamic_power + gpu_static_power
对于CPU的power，我们按照半载进行计算，即K取CPU核数的一半为2，带入公式计算得到：
P_max = 2 * cpu_dynamic_power + gpu_dynamic_power + gpu_static_power = 1492
```

sustainable-power 的计算如下：

```
整个系统可分配的power可以通过以下公式计算得到：

Power = sustainable-power + P + I + D                    （linux-5.4/drivers/thermal/
    power_allocator.c）

按照上面我们方案的设定，75摄氏度时系统开始限制频率，即整个系统的在75摄氏度时的可分配power为所有的cooling-
    device的power之和，此时PID参数中的I和D均为0：

P_75 = P_max = Power = sustainable-power + P = sustainable-power + 2 * sustainable-power /
    (target- threshold) * (target- 75)

sustainable-power + 2 * sustainable-power / (85 - 70) * (85 - 75) = 1492mW

sustainable-power = 639mW
```

使用 sustainable-power=639 进行不同场景的测试，比如 Antutu、Geekbench 等，抓 trace 数据，分析频率和温度的变化情况，或者通过 lisa 工具绘图分析，看看是否符合预期，如果不符合预期就修改 CPU 最大功耗的 K 值进行计算，反复调试，直到符合预期。

#### 2.4.3.5 k_po

超过目标温度时 pid 的 p 参数，如果方案规格没有特别要求，可以不定义使用预设值。

```
k_po计算：
根据公式k_po = sustainable-power / (desired_temperature - switch_on_temp)(linux-4.9/
    Documentation/thermal/cpu-cooling-api.txt)，
假如sustainable-power=793，desired_temperature=85，switch_on_temp=70，四核，cpu动态功耗为：
k_po = 793 / (85 - 70) = 53
```

若使用 stablity 进行测试，出现 cpu 温度过多的大幅低于目标温度值的情况，即为 k_po 调节过量，可以适当将 k_po 改小。

#### 2.4.3.6 k_pu

未超过目标温度时 pid 的 p 参数，如果方案规格没有特别要求，可以不定义使用预设值。

```
k_pu计算:
根据公式k_pu = 2 * sustainable-power / (desired_temperature - switch_on_temp)(linux-4.9/
    Documentation/thermal/cpu-cooling-api.txt),
假如sustainable-power=793，desired_temperature=85，switch_on_temp=70，四核，cpu动态功耗为:
k_pu = 2 * 793 / (85 - 70) = 106
```

若使用 stablity 进行测试，出现 cpu 温度过多的大幅度超过目标温度值的情况，即为 k_pu 调节过量，可以适当将 k_pu 改小。

#### 2.4.3.7 k_i

pid 的 i 参数，如果方案规格没有特别要求，可以不定义使用预设值。

#### 2.4.3.8 k_d

pid 的 d 参数，如果方案规格没有特别要求，可以不定义使用预设值。

## 2.5 源码结构介绍

```
kernel/
|-- drivers/thermal/sunxi_thermal-ng.c //thermal sensor驱动代码
|-- drivers/thermal/cpu_cooling.c //thermal cpu cooling代码
|-- drivers/thermal/devfreq_cooling.c //thermal devfreq cooling代码
```

## 2.6 驱动框架介绍

无。

# 3 模块使用范例

无。

# 4 FAQ

## 4.1 调试方法

### 4.1.1 调试工具

无。

### 4.1.2 调试节点

无。

## 4.2 常见问题

### 4.2.1 查看 sensor 温度

不同平台温度 sensor 的个数及温度监控区域 thermal_zone 是不一样的。多个温度监控区域在/sys/class/thermal 目录下就会有多个 thermal_zone。查看 thermal_zone 的温度，下面以 thermal_zone0 为例：

```
查看thermal_zone的类型

#cat sys/class/thermal/thermal_zone0/type
cpu_thermal_zone
```

```
查看thermal_zone温度

#cat sys/class/thermal/thermal_zone0/temp
36000
温度单位为mC,也就是36摄氏度
```

### 4.2.2 模拟温度

thermal 有温度模拟功能，可以通过模拟温度校验温度策略是否符合预期。

```
设置thermal_zone0的模拟温度

#echo 80000 > /sys/class/thermal/thermal_zone0/emul_temp
```

```
关闭thermal_zone0的模拟温度功能

#echo 0 > /sys/class/thermal/thermal_zone0/emul_temp
```

### 4.2.3 关闭温控

以关闭 thermal_zone0 温控为例。

```
关闭温控策略

#echo disabled > /sys/class/thermal/thermal_zone0/mode

解除所有cooling device的限制
#echo 0 > /sys/class/thermal/thermal_zone0/cdev*/cur_state
```

### 4.2.4 不同温控策略下芯片性能和温度的关系

传统的 stepwise 温控策略，通过 dts 配置芯片在不同温度下 cpu 运行频率、打开核数等性能限制。所以对于 stepwise 策略，芯片在特定温度下性能是确定的。

与 stepwise 温控策略不同，对于 IPA 温控策略，芯片在特定温度下性能是不确定的。若需要了解，可以通过实际测试得出。

### 4.2.5 如何设定过温不关机

修改 cpu_crit@0 节点的 temperature 为很大的值，就不会触发过温关机。

```
        cpu_trips: trips{

            ......

            cpu_crit: cpu_crit@0 {
                temperature = <110000>;     //代表系统到达110度就会过温关机
                type = "critical";
                hysteresis = <0>;
            };
        };
```

同时，在使用 PMIC 的方案上，可能也需要关闭 PMIC 的过温保护功能。详见《Linux_PMIC_开发指南》。

```
pmu0: pmu@0{

    ......

    overtemp_shutdown = <1>;     //过温保护，0为关闭
    overtemp_value = <145>;      //过温保护温度

    ......
};
```

## 4.2.6 如何修改温控策略的目标温度

可以根据方案需求，修改 trip-point@1 节点的 temperature 为温度策略的目标温度。
如若实测温度高于目标温度，可以适当改小 sustainable-power 和 trip-point@0 节点的 temperature；若修改后仍不起效，可以考虑瓶颈是否在硬件。

```
    cpu_thermal_zone{

        ......

        sustainable-power = <1000>;          //温度达到预设温度最大值，系统可分配的最大power

        ......

        cpu_trips: trips{
            cpu_threshold: trip-point@0 {
                temperature = <70000>;       //代表系统温控在70度左右开启
                type = "passive";
                hysteresis = <0>;
            };
            cpu_target: trip-point@1 {
                temperature = <80000>;       //代表系统最高温度是80度左右
                type = "passive";
                hysteresis = <0>;
            };

            ......

        };

    ......

    };
```

## 4.2.7 高温场景下出现卡顿、掉帧等性能不足问题

这种高温下由于温控导致的性能问题原因可以有多种，可以针对不同的场景、不同的原因采用不同的调试解决方法。但是不管是哪种原因造成的，其根本原因都是因为在 IPA 策略下系统可以分配的 power 变小了。所以在调试这类问题之前，要对 IPA 控温的策略有一个基本的了解：
IPA 控温策略是通过对比当前温度与设定的目标温度之间的误差做 PID 闭环控制，从而决定当前温度下每个性能设备（如：CPU、GPU 等）可以分配到的 power。一旦当前温度超过目标温度，且超过的温度值越来越大时，整个系统每个性能设备可以分配到的功耗都会越来越小，因此温度越高系统的性能下降越厉害。在 IPA 控温策略下，系统的温度主要由两部分决定：发热 + 散热。发热主要由两部分组成：环境温度 + 各性能设备发热的温度之和。

基于上述对 IPA 策略的认知，可以对高温下性能不足的问题进行以下分类：

- **测试环境温度被人为提升（如：温箱），机器在这种环境下性能下降：**
  这种场景通常出现在对机器进行高温性能实验的时候，环境温度的升高导致系统无需跑在重负载就能接近或超过目标温度点。CPU 等 cooling-device 随着温度升高会逐渐降低自身的工作频率，但是由于环境温度本身就很高导致最终系统的温度依然很接近或超过目标温度。这种情况下的温度来源是：环境温度占大头，性能设备发热温度占小头，当总的发热量一定时，环境温度越高，性能设备的发热就会被压缩减小，且当长时间超过目标温度时，PID 算法中的误差累积越来越大，最终会导致所有的 cooling-device 都工作在最低频点。
  设想一下，一旦环境温度等于或者超过目标温度，系统的性能设备逐渐被压缩到最低频率，但是即使如此，温度也不可能降到目标温度以下。
  因此，在此类场景下，要解决性能下降的方式要么是降低环境温度，以确保设备不会那么快达到目标温度；要么是提高目标温度，给设备的性能发热提供更多热量裕量。如果产品规格对某个环境温度场景下的性能有明确要求，不能修改测试的环境温度及目标温度，那 IPA 温控策略无法满足性能需求，需协同产品经理、方案软件负责人、方案硬件负责人一起商讨方案的散热处理。
- **有多个 cooling-device 情况下，高温出现性能下降：**
  如果是在常温的环境下，由于系统跑大负载导致机器出现过温，性能下降，那此时的温度来源主要大头是性能设备的发热。如果此时 thermal zone 使用了多种 cooling-device（如：CPU、GPU、VE 等），可以先对现象进行归类（卡顿、显示异常、掉帧等），进而分析造成这种现象的主要原因是哪个性能设备性能下降引起（CPU、GPU、VE），然后适当提高这个性能设备在进行 power 分配时的比重，让这个性能设备能分配到更多的 power，运行在更高的频点。具体调试方法如下：

```
获取thermal_zone0下的cooling-device0的比重

#cat sys/class/thermal/thermal_zone0/cdev0_weight
1024

提高thermal_zone0下的cooling-device0的比重

#echo 4096 > sys/class/thermal/thermal_zone0/cdev0_weight
```

注意：这里设置的比重需为1024的倍数

如果上述方法依然不能改善性能下降问题，可参考上一小章的方法，提高温控目标温度并重新测定参数，或方案增加散热措施进行散热。

- **只有一个 cooling-device 情况下，高温出现性能下降：**
  如果是在常温的环境下，由于系统跑大负载导致机器出现过温，性能下降，那此时的温度来源主要大头是性能设备的发热。如果此时 thermal zone 只使用了一种 cooling-device（如：CPU），想改善性能下降问题只能通过提高温控目标温度并重新测定参数，或方案增加散热措施进行散热来解决。

## 4.2.8 为什么温度超过 switch_on 温度而没有超过目标温度，cpu 频率会被限制到最低频

IPA 策略在计算 cpu 需要的 power 的时候还会乘上当前 cpu 的负载情况（power * load），当负载为 0% 时计算的 power 结果为 0，即当前 cpu 所需的 granted_power 为 0，频率会降至最低频。
在一些只有 cpu 作为 cooling-device，cpufreq-governor 为 performance 的应用场景中，还会看到超过 switch_on 温度后 cpu 的频率一直在最低频和最高频之间来回跳动。这是因为 cpu 的负载情况时而为 0% 空载，时而为大于 0% 的轻载，当计算得到的 power 为 0 时分配最低功耗，cpu 最大工作频率被限制为最低频；当计算得到的 power 即使很小但是不为 0 时，由于只有一个 cooling-device，依然会为这个 cooling-device 分配全部的 power，所以 cpu 最大工作频率设置为最高频。
这类问题大多出现在通过外部方式加热机器，且 cpufreq-governor 使用 performance 的情况下。这种情况下 cpu 本身没有跑多大的负载，但却依然跑在最高频，导致偶尔会出现负载为 0 的情况。这类问题的解决办法要么是更换 cpufreq-governor 为 ondemand 或 conservative，要么适当让 CPU 跑上一定负载。

## 4.2.9 为什么使用 IPA 温度从高于目标温度降回低于目标温度一段时间后，系统性能还是被限制

由于 IPA 策略使用的是 PID 控制算法，其对当前可分配能量的计算公式为：Power = sustainable-power + P + I + D，其中 “I” 作为累计误差只考虑了温度超过目标温度时候取值为负数的误差，且 I 值只有在温度低于 switch_on 温度时才会被清零 reset。因此当温度如果比较长时间超过目标温度时，会导致 “I” 一直进行进行累积，当 “I” 累积值大于 sustainable-power + P 时，可分配的能量将一直为 0，cooling-device 将一直处于最低频率状态，直到温度低于 switch_on 温度将 PID 参数复位，被限制的现象才会消失。这类问题的解决办法是重新调整 IPA 参数或降低实验环境的温度，不要让系统温度长时间超过目标温度。

## 4.2.10 其他温控方法

- 使用更低功耗的 cpu 调频策略，如 ondemand、conservative、powersave 等。详见《Linux_CPUFREQ_ 开发指南》。
- cpufreq 删除高频点、增加低频点。详细咨询方案硬件开发人员。

**著作权声明**



**商标声明**



**免责声明**