

# Rockchip

## CPU-Freq 开发指南

发布版本:1.01

日期:2017.02

# 前言

## 概述

本章主要描述 CPUFreq 的相关的重要概念、配置方法和调试接口。

## 产品版本

与本文档相对应的产品版本如下。

产品名称	内核版本
RK3399	Linux4.4

## 读者对象

本文档（本指南）主要适用于以下工程师：

技术支持工程师

软件开发工程师

## 作者信息

章节号	章节名称	作者信息
全文	全文	XF

## 修订记录

修订记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

修订日期	版本	修订说明
2016.07.01	1.0.0	第一次临时版本发布
2017.02.13	1.0.1	修改了 dts 的路径和节点说明

# 目录

1	重要概念 .....	1-1
2	配置方法 .....	2-1
2.1	DTS 配置 .....	2-1
2.1.1	时钟的配置 .....	2-1
2.1.2	电压域配置 .....	2-1
2.1.3	频率电压表配置 .....	2-2
2.1.4	EAS 配置 .....	2-6
2.2	Menuconfig 配置 .....	2-8
3	调试接口 .....	3-1
3.1	变频策略切换 .....	3-1

# 1 重要概念

CPUFreq 是内核开发者定义的一套支持动态调整 CPU 频率和电压的的框架模型。它能有效的降低 CPU 的功耗，同时兼顾 CPU 的性能。

CPUFreq 通过不同的变频策略，选择一个合适的频率供 CPU 使用，目前的内核版本提供了以下几种策略：

- sched: EAS 使用的调频策略。
- interactive: 根据 CPU 负载动态调频调压；
- conservative: 保守策略，逐级调整频率和电压；
- ondemand: 根据 CPU 负载动态调频调压，比 interactive 策略反应慢；
- userspace: 用户自己设置电压和频率，系统不会自动调整；
- powersave: 功耗优先，始终将频率设置在最低值；
- performance: 性能优先，始终将频率设置为最高值。

## Energy Aware Scheduling (EAS)

EAS 是新一代的任务调度策略，它结合了 CPUFreq 和 CPUIdle 的策略，在为某个任务选择运行 CPU 时，同时考虑了性能和功耗，保证了系统能耗最低，并且不会对性能造成影响。CPUFreq sched 的调度策略就是专门给 EAS 使用的 CPU 调频策略。

# 2 配置方法

## 2.1 DTS 配置

### 2.1.1 时钟的配置

CPU 的时钟已经在 DTSI 中配置好，每个 CPU 节点下都有个 Clocks 属性，具体的名字由 CRU 模块决定，不同的平台有不同的名字。以 RK3399 为例，在头文件 dt-bindings/clock/rk3399-cru.h 中有定义大核 A72 为 ARMCLKB，小核 A53 为 ARMCLKL。所以 rk3399.dtsi 中可以看到如下配置

```
cpu_l0: cpu@0 {
    device_type = "cpu";
    compatible = "arm,cortex-a53", "arm,armv8";
    reg = <0x0 0x0>;
    enable-method = "psci";
    #cooling-cells = <2>; /* min followed by max */
    dynamic-power-coefficient = <100>;
    clocks = <&cru ARMCLKL>;
    cpu-idle-states = <&CPU_SLEEP &CLUSTER_SLEEP>;
};

cpu_b0: cpu@100 {
    device_type = "cpu";
    compatible = "arm,cortex-a72", "arm,armv8";
    reg = <0x0 0x100>;
    enable-method = "psci";
    #cooling-cells = <2>; /* min followed by max */
    dynamic-power-coefficient = <436>;
    clocks = <&cru ARMCLKB>;
    cpu-idle-states = <&CPU_SLEEP &CLUSTER_SLEEP>;
};
```

### 2.1.2 电压域配置

由于不同的 SDK 或样机，会有不同的电源方案，所以 CPU 的电压域配置一般在板级的 DTSI 或 DTS 中，且需要对每个 CPU 节点进行配置，即每个 CPU 节点增加 CPU-supply 属性。以 RK3399 为例，在 arch/arm64/boot/dts/rockchip/Rk3399-evb-rev3.dtsi 中有如下配置

```
vdd_cpu_b: syr827@40 {
    compatible = "silergy,syr827";
    reg = <0x40>;
    vin-supply = <&vcc5v0_sys>;
    regulator-compatible = "fan53555-reg";
    pinctrl-0 = <&vsel1_gpio>;
    vsel-gpios = <&gpio1 17 GPIO_ACTIVE_HIGH>;
    regulator-name = "vdd_cpu_b";
    regulator-min-microvolt = <712500>;
};
```

```

        regulator-max-microvolt = <1500000>;
        regulator-ramp-delay = <1000>;
        fcs,suspend-voltage-selector = <1>;
        regulator-always-on;
        regulator-boot-on;
        regulator-initial-state = <3>;
        regulator-state-mem {
            regulator-off-in-suspend;
        };
    };

    vdd_cpu_l: DCDC_REG2 {
        regulator-always-on;
        regulator-boot-on;
        regulator-min-microvolt = <750000>;
        regulator-max-microvolt = <1350000>;
        regulator-ramp-delay = <6001>;
        regulator-name = "vdd_cpu_l";
        regulator-state-mem {
            regulator-off-in-suspend;
        };
    };
};

```

在 arch/arm64/boot/dts/rockchip/Rk3399-evb.dtsi 中有如下配置

```

&cpu_l0 {
    cpu-supply = <&vdd_cpu_l>;
};
&cpu_l1 {
    cpu-supply = <&vdd_cpu_l>;
};
&cpu_l2 {
    cpu-supply = <&vdd_cpu_l>;
};
&cpu_l3 {
    cpu-supply = <&vdd_cpu_l>;
};
&cpu_b0 {
    cpu-supply = <&vdd_cpu_b>;
};
&cpu_b1 {
    cpu-supply = <&vdd_cpu_b>;
};

```

## 2.1.3 频率电压表配置

### 2.1.3.1 默认配置

频率电压表在 DTSI 中会有默认的配置，在每个 CPU 节点下有 operating-Points-V2 属性，并且有个 opp Table 的节点和它配合使用。以 RK3399 为例，在 RK3399-opp.dtsi 中有如下配置

```

cluster0_opp: opp-table0 {

```

```
compatible = "operating-points-v2";
opp-shared;

opp@408000000 {
    opp-hz = /bits/ 64 <408000000>;
    opp-microvolt = <800000>;
    clock-latency-ns = <40000>;
};
opp@600000000 {
    opp-hz = /bits/ 64 <600000000>;
    opp-microvolt = <800000>;
    clock-latency-ns = <40000>;
};
opp@816000000 {
    opp-hz = /bits/ 64 <816000000>;
    opp-microvolt = <850000>;
    clock-latency-ns = <40000>;
    opp-suspend;
};
opp@1008000000 {
    opp-hz = /bits/ 64 <1008000000>;
    opp-microvolt = <925000>;
    clock-latency-ns = <40000>;
};
opp@1200000000 {
    opp-hz = /bits/ 64 <1200000000>;
    opp-microvolt = <1000000>;
    clock-latency-ns = <40000>;
};
opp@1416000000 {
    opp-hz = /bits/ 64 <1416000000>;
    opp-microvolt = <1125000>;
    clock-latency-ns = <40000>;
};
};

cluster1_opp: opp-table1 {
    compatible = "operating-points-v2";
    opp-shared;

    opp@408000000 {
        opp-hz = /bits/ 64 <408000000>;
        opp-microvolt = <800000>;
        clock-latency-ns = <40000>;
    };
    opp@600000000 {
```

```

        opp-hz = /bits/ 64 <600000000>;
        opp-microvolt = <800000>;
        clock-latency-ns = <40000>;
    };
    opp@816000000 {
        opp-hz = /bits/ 64 <816000000>;
        opp-microvolt = <825000>;
        clock-latency-ns = <40000>;
        opp-suspend;
    };
    opp@1008000000 {
        opp-hz = /bits/ 64 <1008000000>;
        opp-microvolt = <875000>;
        clock-latency-ns = <40000>;
    };
    opp@1200000000 {
        opp-hz = /bits/ 64 <1200000000>;
        opp-microvolt = <950000>;
        clock-latency-ns = <40000>;
    };
    opp@1416000000 {
        opp-hz = /bits/ 64 <1416000000>;
        opp-microvolt = <1025000>;
        clock-latency-ns = <40000>;
    };
    opp@1608000000 {
        opp-hz = /bits/ 64 <1608000000>;
        opp-microvolt = <1100000>;
        clock-latency-ns = <40000>;
    };
    opp@1800000000 {
        opp-hz = /bits/ 64 <1800000000>;
        opp-microvolt = <1200000>;
        clock-latency-ns = <40000>;
    };
};

&cpu_l0 {
    operating-points-v2 = <&cluster0_opp>;
    sched-energy-costs = <&RK3399_CPU_COST_0
&RK3399_CLUSTER_COST_0>;
};

&cpu_l1 {
    operating-points-v2 = <&cluster0_opp>;
    sched-energy-costs = <&RK3399_CPU_COST_0
&RK3399_CLUSTER_COST_0>;
};

```



```

};

&cpu_l2 {
    operating-points-v2 = <&cluster0_opp>;
    sched-energy-costs = <&RK3399_CPU_COST_0
&RK3399_CLUSTER_COST_0>;
};

&cpu_l3 {
    operating-points-v2 = <&cluster0_opp>;
    sched-energy-costs = <&RK3399_CPU_COST_0
&RK3399_CLUSTER_COST_0>;
};

&cpu_b0 {
    operating-points-v2 = <&cluster1_opp>;
    sched-energy-costs = <&RK3399_CPU_COST_1
&RK3399_CLUSTER_COST_1>;
};

&cpu_b1 {
    operating-points-v2 = <&cluster1_opp>;
    sched-energy-costs = <&RK3399_CPU_COST_1
&RK3399_CLUSTER_COST_1>;
};

```

### 2.1.3.2 板级配置

不同的产品对频率电压的需求可能不同，可以在板级 DTSI 或者 DTS 中增加新的 OPP TABLE 覆盖 DTSI 中的。

特别注意，对于 DTSI 中已经有的频率，而板级不想使用该频点，板级 DTS 中还是要引用过来，并且加上 `status = "disabeld";` 属性，比如现在板级不想要 1416000000，则应该做如下配置

```

&cluster0_opp {
    opp@408000000 {
        opp-hz = /bits/ 64 <408000000>;
        opp-microvolt = <800000>;
        clock-latency-ns = <40000>;
    };
    opp@600000000 {
        opp-hz = /bits/ 64 <600000000>;
        opp-microvolt = <800000>;
        clock-latency-ns = <40000>;
    };
    opp@816000000 {
        opp-hz = /bits/ 64 <816000000>;
        opp-microvolt = <850000>;
        clock-latency-ns = <40000>;
    };
}

```

```

        opp-suspend;
    };
    opp@1008000000 {
        opp-hz = /bits/ 64 <1008000000>;
        opp-microvolt = <925000>;
        clock-latency-ns = <40000>;
    };
    opp@1200000000 {
        opp-hz = /bits/ 64 <1200000000>;
        opp-microvolt = <1000000>;
        clock-latency-ns = <40000>;
    };
    opp@1416000000 {
        opp-hz = /bits/ 64 <1416000000>;
        opp-microvolt = <1125000>;
        clock-latency-ns = <40000>;
        status = "disabled";
    };
};
}

```

## 2.1.4 EAS 配置

### 2.1.4.1 默认配置

EAS 在选择一个 CPU 时，需要考虑这个 CPU 每个频率的运算能力和对应的功耗，以及这个 CPU 所在的 cluster 的功耗，所以对应每个 opp 项，需要填写每个频率对应的 CPU 运算能力和功耗情况。默认配置在 arch/arm64/boot/dts/rockchip/rk3399-sched-energy.dtsi 中，cpu 节点中的 sched-energy-costs 属性指定每个 CPU 对应的单核数据（&CPU\_COST\_A53）和 CLUSTER 数据（&CLUSTER\_COST\_A53）。

```

cpu_l0: cpu@0 {
    device_type = "cpu";
    compatible = "arm,cortex-a53", "arm,armv8";
    reg = <0x0 0x0>;
    enable-method = "psci";
    #cooling-cells = <2>; /* min followed by max */
    dynamic-power-coefficient = <121>;
    clocks = <&cru ARMCLKL>;
    cpu-idle-states = <&cpu_sleep>;
    operating-points-v2 = <&cluster0_opp>;
    sched-energy-costs = <&CPU_COST_A53 &CLUSTER_COST_A53>;
};

```

### 2.1.4.2 板级配置

RK3399 evb rev2 的板子对应的配置如下  
(arch/arm64/boot/dts/rockchip/rk3399-evb-rev2.dtsi):

```

&CPU_COST_A72 {
    busy-cost-data = <
        232 349 /* 408MHz */

```

```

        341    547 /* 600MHz */
        464    794 /* 816MHz */
        573   1141/* 1008MHz */
        683   1850/* 1200MHz */
        805   2499/* 1416MHz */
        915   2922/* 1608MHz */
// 1024 3416 /* 1800MHz */
>;
idle-cost-data = <
    15
    0
>;
};

&CPU_COST_A53 {
    busy-cost-data = <
        121    40 /* 408M */
        179    62 /* 600M */
        243    90 /* 816M */
        300   126/* 1008M */
        357   196/* 1200M */
        421   246/* 1416M */
        449   263/* 1512M */
    >;
    idle-cost-data = <
        6
        0
    >;
};

&CLUSTER_COST_A72 {
    busy-cost-data = <
        232   349 /* 408MHz */
        341   547 /* 600MHz */
        464   794 /* 816MHz */
        573   1141/* 1008MHz */
        683   1850/* 1200MHz */
        805   2499/* 1416MHz */
        915   2922/* 1608MHz */
// 1024 3416 /* 1800MHz */
    >;
    idle-cost-data = <
        65
        65
    >;
};

```

```

&CLUSTER_COST_A53 {
    busy-cost-data = <
        121    40 /* 408M */
        179    62 /* 600M */
        243    90 /* 816M */
        300    126 /* 1008M */
        357    196 /* 1200M */
        421    246 /* 1416M */
        449    263 /* 1512M */
    >;
    idle-cost-data = <
        56
        56
    >;
};

```

**busy-cost-data** 表示 CPU\CLUSTER 在不同频率全速运行时对应的运算能力和功耗数据。这个表格一定要和 OPP 表项按顺序一一对应，如果 OPP Table 里面把某个 OPP 项 Disabled 了，那么这个表格里面对应的数据也要注释掉，否则会影响 EAS 的调度策略。

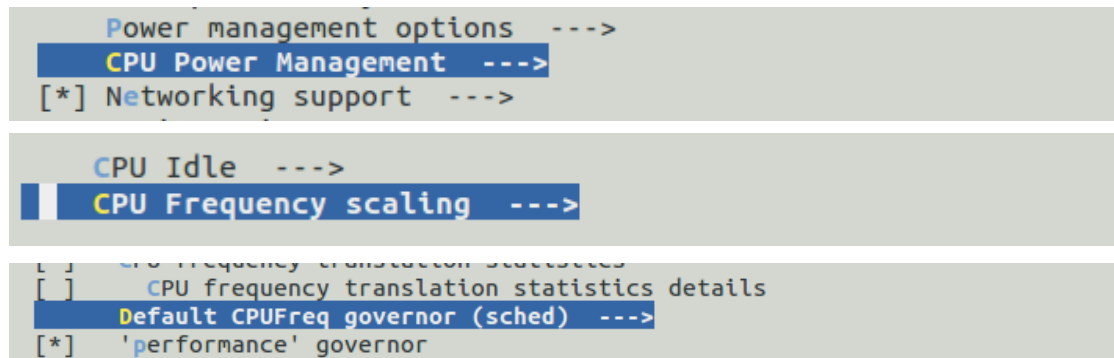
**Idle-cost-data** 表示 CPU\CLUSTER 在不同的 idle 状态下的功耗数据，EAS 用这个值来考虑唤醒一个 CPU 或 CLUSTER 对功耗影响有多大。

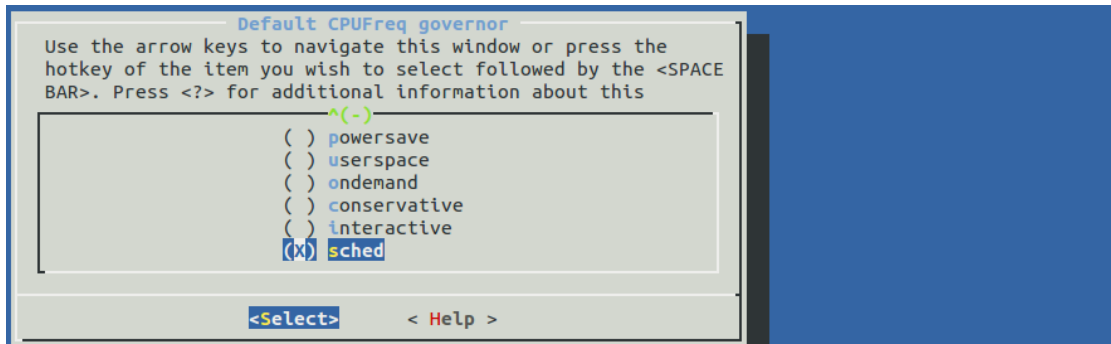
CPU\_COST\_A72\A53 分别表示 A72\A53 单核的性能和功耗数据。

CLUSTER\_COST\_A72\A53 分别表示 A72\A53 对应的 CLUSTER 开启和关闭的功耗数据，只有 CLUSTER 里面的 CPU 全部关闭后，才能关闭对应的 CLUSTER。这个节点里面 busy-cost-data 表格只用到了功耗数据，性能数据没有用到。

## 2.2 Menuconfig 配置

make ARCH=arm64 menuconfig





这边可以选择变频策略，如没有特殊需求，请保持默认值 sched。

# 3 调试接口

## 3.1 变频策略切换

对于一个只有一个 Cluster 的平台，在串口中输入如下命令(xxx 为策略名称: sched、interactive、conservative、ondemand、userspace、powersave、performance)

```
echo xxx > /sys/devices/system/cpu/cpufreq/policy0/scaling_governor
```

对于两个 cluster 的平台，在串口中输入如下命令，分别改变不同 cluster 的变频策略：

```
echo xxx > /sys/devices/system/cpu/cpufreq/policy0/scaling_governor
```

```
echo xxx > /sys/devices/system/cpu/cpufreq/policy4/scaling_governor
```

### CPU 定频

CPU 定频，需要先将变频策略改为 userspace，再设置频率。

方法一：

对某个 cpu 定频，如 cpu0 定频 216MHz，在串口中输入如下命令：

```
echo userspace > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
```

```
echo 216000 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_setspeed
```

设置完后，查看当前频率：

```
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq
```

方法二：

目前我们的芯片，有的是一个 cluster 如 RK3288，有的是两个 cluster 如 RK3399，一个 cluster 下的所有 cpu 共用一个时钟，所以设定某个 cpu 的频率时，其实会对相同 cluster 下的所有 cpu 都定频。所以我們也可以通过第二种办法对整个 cluster 的 cpu 定频。

```
echo userspace > /sys/devices/system/cpu/cpufreq/policy0/scaling_governor
```

```
echo userspace > /sys/devices/system/cpu/cpufreq/policy4/scaling_governor
```

```
echo 216000 > /sys/devices/system/cpu/cpufreq/policy0/scaling_setspeed
```

```
echo 216000 > /sys/devices/system/cpu/cpufreq/policy4/scaling_setspeed
```

设置完后，查看当前频率：

```
cat /sys/devices/system/cpu/cpufreq/policy0/scaling_cur_freq
```

```
cat /sys/devices/system/cpu/cpufreq/policy4/scaling_cur_freq
```

一个 policy 对应一个 cluster，每个 cluster 下对应哪些 CPU 可以通过以下命令获取

```
cat /sys/devices/system/cpu/cpufreq/policy0/related_cpus
```

```
cat /sys/devices/system/cpu/cpufreq/policy4/related_cpus
```