功耗分析和优化

发布版本:1.0

作者邮箱：cl@rock-chips.com

日期:2019.08.31

文件密级：公开资料

前言

概述

本文档主要介绍 RK 平台芯片功耗的一些基础概念和优化方法。

产品版本

产品名称	内核版本
所有芯片	所有内核版本

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

修订记录

日期	版本	作者	修改说明
2019.08.31	V1.0	陈亮	初始版本

---

1. 基础概念

1.1 频率（clk）和电压（voltage）

SoC 内部一般会包含很多模块，比如:ARM,GPU,DDR,I2C,SPI,USB 等，每个模块工作的时候，数字逻辑部分需要一个合适频率和对应的电压，模块频率越高，需要的电压也会越高，频率和电压是功耗的两个重要参数。

1.2 电压域 VD（voltage domain）和电源域 PD（power domain）

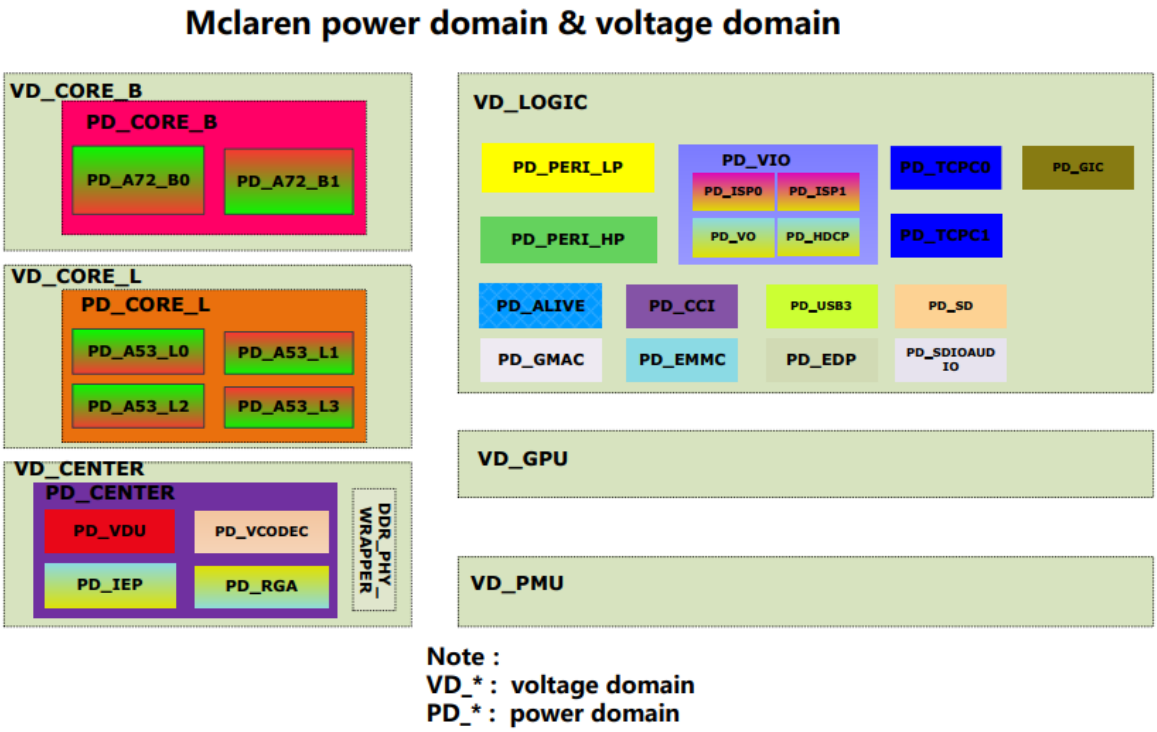
SoC 内部的各个模块，一般都有数字逻辑部分和 IO 部分，数字逻辑部分主要负责运算和状态控制，IO 部分主要负责接口信号传输（有些模块没有 IO，比如：ARM，GPU 等），数字逻辑和 IO 的供电一般都是分开的，IO 部分的功耗比较固定，而数字逻辑部分的功耗受频率和电压影响变化很大，为了优化功耗，把芯片内部数字逻辑部分按模块划分成电压域和电源域。

- 电压域表示芯片内部几个模块共同由一个外部电源供电的区域，可以独立调节或开关电压。一般运行电压相近，并且功耗不大的模块，可以放在同一个电压域中，但是如果功耗很大，最好独立一个电压域，方便做功耗管理，也避免峰值电流超过外部电源的限制。要保证所有的模块可以正常工作，需要把电压域的电压设置到对电压要求最高的模块所需要的电压（不包含关闭的模块）。
- 一个电压域里面可能包含很多个模块， 而这些模块一般不会同时工作，在有供电的情况下，不工作的模块存在漏电流（leakage），为了减小漏电流，通常会把一个电压域内部划分成几个区域，每个区域可以独立开关（switch off）电源，某个区域关闭电源后，就与其他模块隔离（isolation），可以大大减小漏电流，这样的区域就叫做电源域。

以 RK3399 为例，有 6 个 VD：

- VD_CORE_B: 包含两个大核 Contex-A72，功耗比较大，所以独立一个电压域。
- VD_CORE_L: 包含四个小核 Contex-A53，功耗比较大，所以独立一个电压域。
- VD_LOGIC: 包含一些外设的控制器和系统总线，比如：USB，EMMC，GMAC，SPI，I2C，EDP，VOP，AXI，AHB，APB 等。
- VD_CENTER: 包含 vdpu,vepu,iep,rga 和 DDR 控制器。
- VD_GPU: 包含 GPU，功耗比较大，所以独立一个电压域。
- VD_PMU: 包含 PMU,SRAM,GPIO,PVTM 等涉及待机唤醒流程的模块。

框图如下：



1.3 DCDC(Direct Current)和 LDO(low dropout regulator)

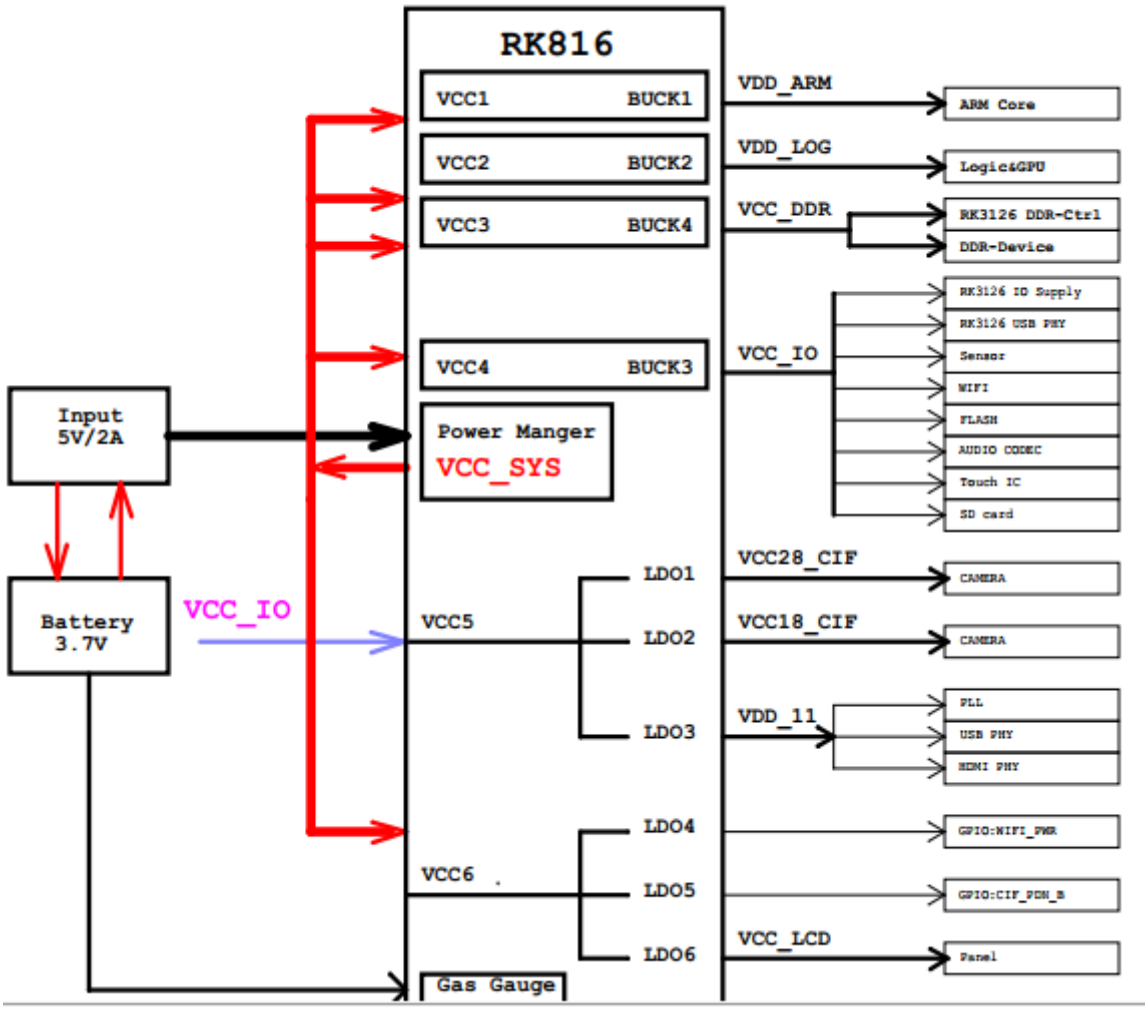
SoC 的外部供电主要分为 DCDC 和 LDO 两种：

- DCDC 一般是指开关电源，转换效率高，效率能达到~80%~90%，电流较大时，需要用 DCDC 提高电源效率；
- LDO 主要特点是输入电流等于输出电流，所以电源效率就等于输出电压/输入电压，假设输入 3.8V，输出 1.0V，电源效率就是 1V/3.8V=26.3%, 电源效率低；

以 RK3126+RK816 的供电方案为例：

- RK816 的 4 路 BUCK 分别给 RK3126 的 ARM、LOG、DDR、IO 供电，因为这几路电流都比较大(BUCK 是一种降压型 DCDC)；
- RK816 的 6 路 LDO 分别给 RK3126 的 PLL，PHY 和一些外设供电，这些模块的电流一般都比较小；

供电框图如下：



1.4 静态功耗和动态功耗

- 静态功耗是指 SoC 内部模块没有工作时，晶体管的泄露电流所消耗的功耗，静态功耗大小随温度和电压的升高而增大。
- 动态功耗是指 SoC 内部模块工作时，内部电路翻转所消耗的功耗，动态功耗大小随频率和电压的升高而增大。

动态功耗公式：

```
    /* C是常量， V是电压， F是频率 */
    P(d)= C * V^2 * F
```

1.5 DVFS(Dynamic Voltage and Frequency Scaling)、CPUFREQ 和 DEVFREQ

模块工作频率越高，电压越高，则功耗也越大，所以需要通过动态调整频率电压优化功耗，当系统空闲时，降低频率和电压，系统繁忙时，提高频率和电压。

- DVFS 是指动态调频调压的技术，作为 CPUFREQ 和 DEVFREQ 底层技术的实现；
- CPUFREQ 是指动态调节 CPU 频率的软件框架，包含几种不同的调频策略，细节请查看文档《Rockchip-Developer-Guide-Linux4.4-CPUFreq-CN》；
- DEVFREQ 是指动态调节外设(不包含 CPU)频率的软件框架，包含几种不同的调频策略，细节请查看《Rockchip-Developer-Guide-Linux4.4-Devfreq》；

2. 功耗测量

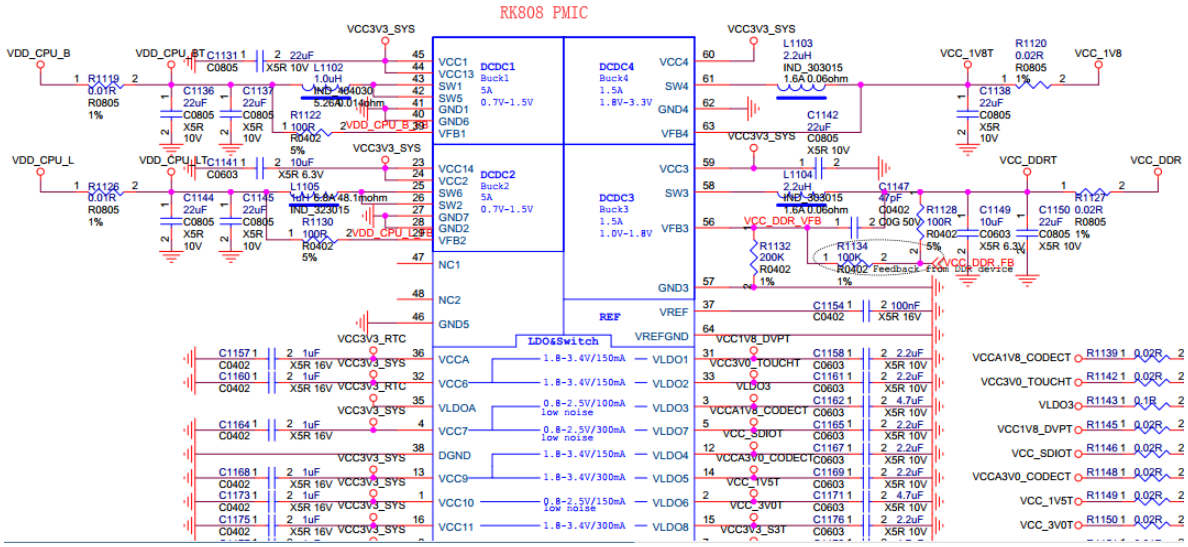
优化功耗之前，需要把各路电源的电压和电流都测量出来，分析数据，然后再做对应的优化。

注：温度是影响功耗的一个重要参数，所以测量功耗的同时需要实时记录温度，获取温度的命令如下：

```
cat /sys/class/thermal/thermal_zone0/temp
```

2.1 测量方法

在电路上串联一个电阻 R，测量电阻两端的电压差 U， 则电流 I=U/R，一般电阻用 0.01 欧姆，需要根据电流大小调整电阻大小。
以 RK3399 EVB 板为例，采用电压测量法，在 VDD_CPU_B,VDD_CPU_L,VCC_1V8,VCC_DDR 输出端都串联了 0.01 欧姆电阻，如下图：



2.2 测量工具

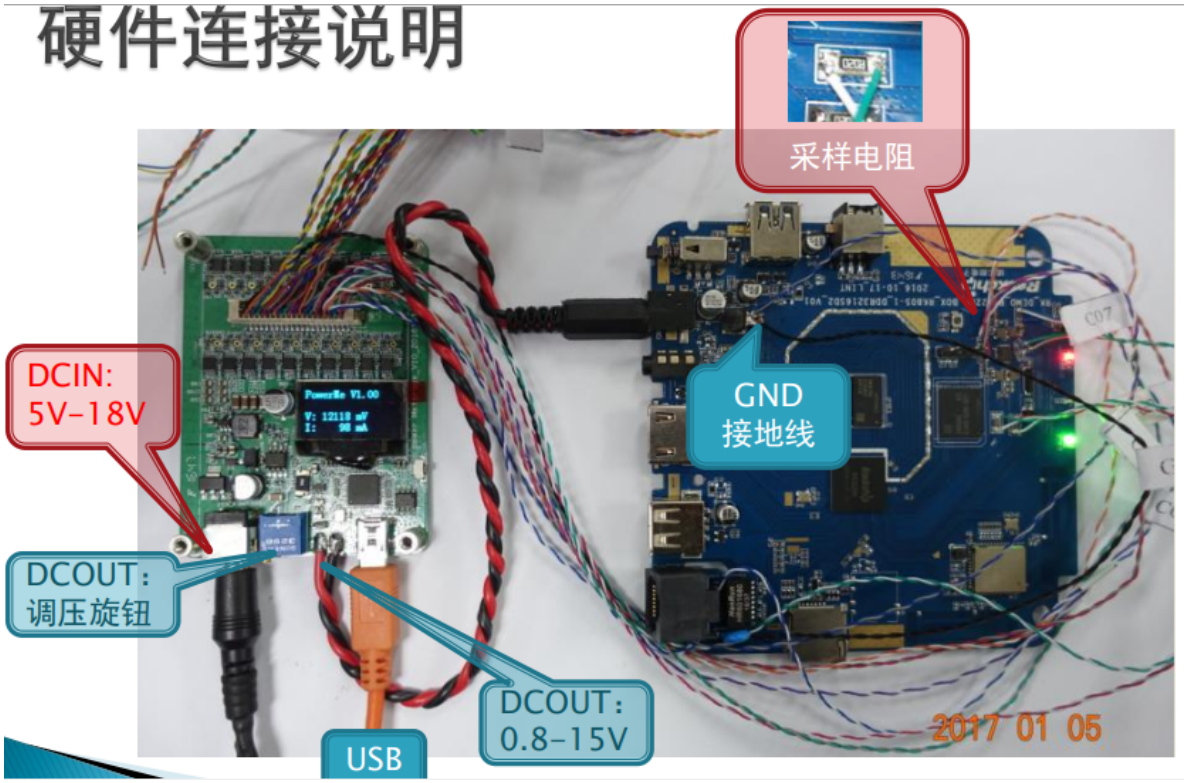
由于需要测量的电源有很多路，使用多路电压电流采集器可以有效地提高测试效率。PowerMeterage 是 RockChip 开发的电压电流采集工具，可以同时测量 20 路功耗数据，界面如下：

RK3399 (Sample:13094 1212 Sample/S 00:00:10)

Name	12V		VSYS		VCC3V3_SYS		VDD_CPU_B		VDD_LOG		VDD_CPU_L		VDD_CENTER		VCC_DDR		VCC_1V8		VDD_GPU	
Avg	11.891V	258.4mA	5.046V	565.3mA	3.431V	91.1mA	0.812V	39.9mA	0.94V	292.9mA	0.821V	72.3mA	0.909V	433.9mA	1.254V	140.6mA	1.816V	43.8mA	0.798V	199.8mA
Rms	11.891V	260.8mA	5.046V	570.6mA	3.431V	91.1mA	0.814V	130.4mA	0.94V	293mA	0.821V	79.9mA	0.909V	438.8mA	1.254V	147.2mA	1.816V	43.9mA	0.798V	283.5mA
Max	11.92V	523.3mA	5.075V	1156.9mA	3.435V	104.4mA	1.145V	1609.1mA	0.941V	319.6mA	1.074V	425.6mA	0.912V	627.9mA	1.259V	319.4mA	1.819V	70mA	0.802V	559.8mA
Now	11.894V	249.5mA	5.047V	548.9mA	3.431V	91.1mA	0.8V	9.2mA	0.94V	292.2mA	0.817V	67.1mA	0.909V	424.7mA	1.255V	134.5mA	1.816V	43.6mA	0.798V	172.1mA
CH/mR	CH:0	50mR	CH:1	50mR	CH:2	50mR	CH:3	50mR	CH:4	50mR	CH:5	50mR	CH:6	50mR	CH:7	50mR	CH:8	50mR	CH:9	50mR

Name	IR_LED		MIPI_BL		CAMERA_HOST2		VDD_LCD	
Avg	4.963V	0.1mA	18.345V	4.5mA	5.024V	146.9mA	2.787V	20.8mA
Rms	4.963V	0.1mA	18.345V	4.5mA	5.024V	147.3mA	2.787V	20.8mA
Max	4.985V	0.2mA	18.36V	4.7mA	5.051V	160.5mA	2.79V	21.9mA
Now	4.962V	0.1mA	18.344V	4.5mA	5.024V	146.6mA	2.788V	20.8mA
CH/mR	CH:10	50mR	CH:11	100mR	CH:12	100mR	CH:13	100mR

PowerMeterage 硬件连接如下：



3. 功耗数据分析

3.1 计算理论功耗

使用 PowerMeterage 工具，分解各路功耗，DCDC 按~80%-90%的效率折算到电池端，LDO 输出电流等于输入电流，把 DCDC，LDO 和其他电源都折算到电池端，加起来估算总功耗，如果和实测电池端的功耗相差太大，则可能存在漏电，需要进一步分析。

以 RK3326 EVB 板为例，静态桌面功耗数据如下：

注：各路测试结果需要折算到电池端(3.8V)的功耗，所以对比电池端的实测电流和理论电流会更方便。

类型	各路输出	电压(V)	实测电流(mA)	3.8V 端理论电流（mA）	备注
DC/DC	VDD_ARM	0.96	10.20	3.23	按 80%效率, 换算公式： V * I / 效率 / 电池电压
DC/DC	VDD_LOG	0.96	89.30	28.20	例如： LOG 3.8V 端理论电流= 0.96 * 89.3 / 0.8 / 3.8 = 28.2
DC/DC	VCC_DDR	1.26	38.50	15.91	
DC/DC	VCC_IO	2.99	4.50	4.43	
LDO	VCC_1V8	1.81	28.80	28.80	LDO 输出电流等输入电流
LDO	VDD_1V0	1.00	10.90	10.90	

LDO	VCC3V0_PMU	3.01	1.20	1.20	
类型	各路输出	电压(V)	实测电流(mA)	3.8V 端理论电流(mA)	备注
电池端	VBAT	3.81	94.60	92.67	理论值和实测值相近

3.2 对比 EVB 数据

分解的各路功耗数据，在相同场景下和 EVB 上的数据做对比，确认是否存在异常，比如下面是 RK3326 EVB 板和客户样机在静态桌面下的功耗对比，可以看出来客户板子的 ARM、LOG 两路功耗存在异常，需要进一步分析。

类型	各路输出	EVB		客户样机	
		电压(V)	电流(mA)	电压(V)	电流(mA)
DC/DC	VDD_ARM	0.96	10.20	1.10	212.50
DC/DC	VDD_LOG	0.96	89.30	1.00	151.30
DC/DC	VCC_DDR	1.26	38.50	1.27	40.50
DC/DC	VCC_IO	2.99	4.50	2.99	4.80
LDO	VCC_1V8	1.81	28.80	1.81	29.80
LDO	VDD_1V0	1.00	10.90	1.00	10.20
LDO	VCC3V0_PMU	3.01	1.20	3.01	1.40
电源端	VBAT	3.81	94.60	3.81	191.6

3.3 各路数据分析

3.3.1 VDD_CORE/VDD_CPU/VDD_ARM

这三个名字是指同一个电源，即：ARM 核心的电源，主要从以下几个方面分析功耗：

- 确认频率电压表(opp-table)是否正常，实测电压和设置电压是否一致。

相关的命令如下：

```
/* 获取频率电压表, target这一列表示某个频率需求的电压 */
cat /sys/kernel/debug/opp/opp_summary
 device                rate(Hz)    target(uV)    min(uV)    max(uV)
-----------------------------------------------------------------------
...
 cpu0
                     408000000        950000      950000     1350000
                     600000000        950000      950000     1350000
                     816000000       1000000     1000000     1350000
                    1008000000       1125000     1125000     1350000
                    1200000000       1275000     1275000     1350000
                    1248000000       1300000     1300000     1350000
                    1296000000       1350000     1350000     1350000

/* 查看cpufreq当前使用的变频策略，默认的interactive策略cpu是变频的 */
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
interactive
```

```
/* 设置userspace策略定频cpu，然后设置不同频率，比较设置电压和实测电压 */
echo userspace > /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_governor
userspace

/* 查看cpufreq支持的频率点 */
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_available_frequencies
408000 600000 816000 1008000 1200000 1248000 1296000

/* 设置定频的频率 */
echo 408000 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_setspeed

/* 确认当前频率 */
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq
408000

/* 确认当前电压，并实测比较，vdd_arm表示regulator的名字，不同项目会有变化 */
cat /sys/kernel/debug/regulator/vdd_arm/voltage                                <
950000

/* 获取所有regulator当前的电压 */
cat /sys/kernel/debug/regulator/regulator_summary                               <
 regulator                      use open bypass voltage current     min     max
---------------------------------------------------------------------------------
...
 vcc3v8_sys                       0   12      0  3800mV     0mA  3800mV  3800mV
    deviceless                                                      0mV     0mV
    vdd_logic                     0    4      0   950mV     0mA   950mV  1350mV
       dmc                                                        950mV  1350mV
       ff400000.gpu                                               950mV  1350mV
       bus-apll                                                   950mV  1350mV
       deviceless                                                   0mV     0mV
    vdd_arm                       0    2      0   950mV     0mA   950mV  1350mV
       cpu0                                                       950mV  1350mV
       deviceless                                                   0mV     0mV
...
```

- 查看 cpu 负载，分析是否有异常的任务或中断。

```
/* 使用top命令查看任务负载，不同版本的top输出会有差异，此版本top支持查看线程和线程运行的cpu
*/
top -m 5 -t
User 51%, System 2%, IOW 0%, IRQ 0%
User 712 + Nice 0 + Sys 33 + Idle 634 + IOW 0 + IRQ 0 + SIRQ 0 = 1379

/* PR这一列表示线程当前运行的cpu，所有cpu的负载百分比加起来等于100%，所以每个cpu的最高的负
载百分比为100%/NR_CPU，4核的SoC单个CPU最高负载百分比为25% */
  PID   TID PR CPU% S     VSS     RSS PCY UID      Thread          Proc
 2631  2631  3  25% R   3104K    552K  fg root     busybox         busybox
 2632  2632  2  25% R   3104K    552K  fg root     busybox         busybox
 2633  2633  1   3% R    740K    400K  fg root     top             /data/top
  255   476  0   0% S  15492K   4988K  fg system   HwBinder:255_1
 /vendor/bin/hw/android.hardware.sensors@1.0-service
  419   478  1   0% S 3770752K 256884K  fg system   SensorService
system_server

/* 使用cpustats观察cpu频率变化 */
```

```
cpustats
Total: User 600 + Nice 0 + Sys 3 + Idle 591 + IOW 0 + IRQ 0 + SIRQ 0 = 1194
  408000kHz 0 +
  600000kHz 0 +
  816000kHz 0 +
  1008000kHz 0 +
  1200000kHz 0 +
  1248000kHz 0 +
  1296000kHz 0 +
  1416000kHz 0 +
  1512000kHz 1200 = 1200 /* 统计时间内，总共有1200个系统时间片（jiffies），1512M运行了
1200个时间片 */
/* 下面可以看到每个cpu的负载情况，包括用户态，内核态，中断和空闲时间 */
cpu0: User 0 + Nice 0 + Sys 1 + Idle 294 + IOW 0 + IRQ 0 + SIRQ 0 = 295
cpu1: User 299 + Nice 0 + Sys 1 + Idle 0 + IOW 0 + IRQ 0 + SIRQ 0 = 300
cpu2: User 1 + Nice 0 + Sys 1 + Idle 296 + IOW 0 + IRQ 0 + SIRQ 0 = 298
cpu3: User 300 + Nice 0 + Sys 1 + Idle 0 + IOW 0 + IRQ 0 + SIRQ 0 = 301

/* 通过cpufreq节点查看各个频率运行时间的占比，时间单位：jiffies */
cat /sys/devices/system/cpu/cpu0/cpufreq/stats/time_in_state
408000 718186
600000 548
816000 368
1008000 1578
1200000 1104
1248000 84
1296000 101
1416000 678
1512000 47495

/* 查看所有外设的中断数量 */
cat /proc/interrupts
           CPU0       CPU1       CPU2       CPU3
  1:          0          0          0          0     GICv2  29 Edge
 arch_timer
  2:     181898     165057     636772     839244     GICv2  30 Edge
 arch_timer
  5:     180743      39000      28905      65189     GICv2  62 Level
rk_timer
 13:     260634          0          0          0     GICv2  39 Level
ff180000.i2c
 14:     354805          0          0          0     GICv2  40 Level
ff190000.i2c
 15:          0          0          0          0     GICv2  41 Level
ff1a0000.i2c
...
```

3.3.2 VDD_GPU

VDD_GPU 的功耗主要是确认频率电压表是否正常，实测电压和设置电压是否一致，用的是 devfreq 节点。

注：有些芯片 GPU 模块没有独立的 VD，会把 GPU 放在 VDD_LOGIC 里面，这时候需要确认的则是 VDD_LOGIC 的电压是否正常。

```
/* 获取频率电压表 */
cat /sys/kernel/debug/opp/opp_summary
```

```
 device                rate(Hz)    target(uV)    min(uV)    max(uV)
----------------------------------------------------------------------
...
 platform-ff400000.gpu
                      200000000        950000      950000      950000
                      300000000        950000      950000      950000
                      400000000       1025000     1025000     1025000
                      480000000       1100000     1100000     1100000
                      520000000       1150000     1150000     1150000
...

/* 查看gpu devfreq当前使用的变频策略，默认的simple_ondemand策略gpu是变频的 */
cat /sys/class/devfreq/ff400000.gpu/governor
simple_ondemand
注：ff400000.gpu其中ff400000是gpu的寄存器地址，所以不同芯片这个name也会不同。

/* 设置userspace策略定频gpu，然后设置不同频率，比较设置电压和实测电压 */
echo userspace > /sys/class/devfreq/ff400000.gpu/governor
cat /sys/class/devfreq/ff400000.gpu/governor
userspace

/* 查看gpu devfreq支持的频率点 */
cat /sys/class/devfreq/ff400000.gpu/available_frequencies
520000000 480000000 400000000 300000000 200000000

/* 设置定频的频率 */
echo 200000000 > /sys/class/devfreq/ff400000.gpu/userspace/set_freq

/* 确认当前频率 */
cat /sys/class/devfreq/ff400000.gpu/cur_freq
200000000

/* 确认当前电压，并实测比较 */
cat /sys/kernel/debug/regulator/vdd_gpu/voltage
950000

/* 查看gpu的负载 */
cat /sys/class/devfreq/ff400000.gpu/load
0@200000000Hz
```

3.3.3 VDD_LOGIC

VDD_LOGIC 里面一般会包含很多模块，为了方便功耗管理，内部又会划分成很多个 PD，主要从以下几个方面分析功耗：

- 确认各个模块的运行频率和开关状态。

```
cat /sys/kernel/debug/clk/clk_summary
   clock                         enable_cnt  prepare_cnt        rate   accuracy
  phase
------------------------------------------------------------------------------
--------
 xin24m                                   9           10    24000000          0
0
...
    pll_gpll                              1            1  1200000000          0
0
```

```
       gpll                              9            20  1200000000          0
0
          clk_sdio_div50                 1             1   100000000          0
0
             clk_sdio                    1             5   100000000          0
0
                sdio_sample              0             1    50000000          0
0
                sdio_drv                 0             1    50000000          0
180
          clk_emmc_div50                 1             1   300000000          0
0
             clk_emmc                    1             5   300000000          0
0
                emmc_sample              0             1   150000000          0
42
                emmc_drv                 0             1   150000000          0
180
...
```

- 确认各个 PD 的开关状态。

```
cat /sys/kernel/debug/pm_genpd/pm_genpd_summary                                <
domain                          status          slaves
    /device                                             runtime status
----------------------------------------------------------------------
pd_gpu                          off
    /devices/platform/ff400000.gpu                      suspended
pd_vi                           off
    /devices/platform/ff4a8000.iommu                    suspended
pd_vo                           on
    /devices/platform/ff460f00.iommu                    active
    /devices/platform/ff470f00.iommu                    suspended
    /devices/platform/ff2e0000.video-phy                suspended
    /devices/platform/ff450000.dsi                      active
    /devices/platform/ff460000.vop                      active
    /devices/platform/ff470000.vop                      suspended
    /devices/platform/ff480000.rk_rga                   suspended
pd_vpu                          off
    /devices/platform/ff440440.iommu                    suspended
    /devices/platform/ff442800.iommu                    suspended
    /devices/platform/vpu_combo                         suspended
pd_mmc_nand                     on
    /devices/platform/ff380000.dwmmc                    unsupported
    /devices/platform/ff390000.dwmmc                    unsupported
    /devices/platform/ff3b0000.nandc                    active
pd_gmac                         off
pd_sdcard                       off
pd_usb                          on
    /devices/platform/ff300000.usb                      active
```

- DDR 模块一般会放在 VDD_LOGIC 里面，并且 DDR 模块功耗较大，和 GPU 一样使用 devfreq 策略优化功耗，所以需要确认频率电压表和实测电压。DDR 还有一些低功耗的配置，比如，pd_idle,sr_idle,odt 开关和其他一些 timing 配置，调试流程比较复杂，需要参考详细的 DDR 文档。

```
cat /sys/kernel/debug/opp/opp_summary
 device                rate(Hz)    target(uV)    min(uV)    max(uV)
-------------------------------------------------------------------
 platform-dmc
                      194000000        950000     950000      950000
                      328000000        950000     950000      950000
                      450000000        950000     950000      950000
                      528000000        975000     975000      975000
                      666000000       1000000    1000000     1000000
...

/* ddr默认使用dmc_ondemand变频策略 */
cat /sys/class/devfreq/dmc/governor
dmc_ondemand
```

其他设置频率和电压的命令与GPU devfreq一样。

3.3.4 VCC_DDR

VCC_DDR 主要是给 DDR 颗粒和 SoC 上 DDR-IO 部分供电，影响 VCC_DDR 功耗的参数有：DDR 频率，DDR 的负载，DDR 低功耗配置，DDR 颗粒类型等。相同条件下，不同厂商的 DDR 颗粒，功耗可能也会有很大的差异。

3.3.5 VCC_IO

VCC_IO 主要是给 SoC 上的 IO Pad 和一些外设供电，主要从以下几个方面分析功耗：

- 外设模块的工作状态，是否存在漏电。
- SoC 上的 IO 管脚状态是否与外设匹配，比如，IO 输出高，而接的外设管脚是低电平。

3.4 常见场景分析

3.4.1 静态桌面

主要是显示模块在工作，CPU，GPU，DDR 应该降到最低频率，并且进入低功耗状态，VDD_CPU,VDD_GPU,VDD_LOGIC 都调到 opp-table 的最低电压，确认 clk_summary，pm_genpd_summary 的状态，确认外设模块（WIFI、BT 等）都处于关闭状态。静态桌面一般作为其他场景功耗的基础，所以需要优先把功耗调到最优状态。

3.4.2 视频播放

主要是视频解码器(VPU/RKVDEC)在工作，GPU 一般处于关闭状态，重点确认 DDR 的运行频率和 VDD_LOGIC 的电压是否正常。

3.4.3 游戏

主要是 CPU 和 GPU 在工作，重点分析 CPU 和 GPU 的负载，频率变化，VDD_CPU 和 VDD_GPU 的电压是否正常。

3.4.4 二级待机

一般情况下，VDD_CPU 和 VDD_GPU 会关闭电源，VDD_LOG 只保留部分唤醒模块供电，所以重点分析 IO，DDR 颗粒和一些外设的功耗。

4. 功耗优化策略

4.1 CPU 优化

- 调整 cpufreq 参数。

```
/* 默认使用的是interactive变频策略，相关参数如下： */
ls -l /sys/devices/system/cpu/cpu0/cpufreq/interactive
go_hispeed_load   /* 负载大于go_hispeed_load且频率小于hispeed_freq时，直接跳到
hispeed_freq */
hispeed_freq         /* 从低频跳转到高频时，需要先跳到hispeed_freq过渡 */
above_hispeed_delay /* 频率大于hispeed_freq时，每次提频前需要维持的时间 */
min_sample_time      /* 每次提频后，如果下一个频率是降频，降频前需要维持的时间 */
target_loads         /* 变频的目标负载 */
timer_rate           /* 负载采样时间，单位:us */
timer_slack          /* cpu进入idle后的负载采样时间 */
boost                /* 频率小于hispeed_freq时，持续boost到hispeed_freq */
boostpulse           /* 频率小于hispeed_freq时，boost到hispeed_freq，维持一段时间 */
boostpulse_duration /* boostpulse维持的时间，单位:us */
io_is_busy           /* io等待是否计算到cpu负载 */

我们主要调整hispeed_freq，target_loads，timer_rate这三个参数：

hispeed_freq:选择一个合适的过渡频率，让cpu稳定在中间频率，功耗最优，这个值太大或太小都会造
成cpu容易跳到高频，增加功耗。
target_loads:加大后更容易跑低频，功耗下降，性能也会下降。
timer_rate:加大后更容易跑低频，功耗下降，性能也会下降。
```

- 关闭部分 cpu，限制 cpu 最高频率。

```
/* 关闭cpu2，cpu3 */
echo 0 > /sys/devices/system/cpu/cpu2/online
echo 0 > /sys/devices/system/cpu/cpu3/online

/* 限制cpu0最高跑1200MHz */
echo 1200000 > /sys/devices/system/cpu/cpu0/cpufreq/scaling_max_freq
```

- ARM Big-Little 架构的 SoC，可以通过 CPUSET，把高负载的任务绑定在小核上，小核能效比更高。

注：对于 SMP 架构的 SoC，也可以把任务集中绑定在某些 cpu 上，让其他 cpu 可以进入低功耗状态，但是可能会导致 cpu 容易跑高频，功耗反而上升。

```
/* 创建小核分组 */
mkdir /dev/cpuset/little

/* 设置小核分组允许使用的cpu */
echo 0-3 > /dev/cpuset/little/cpus

/* 把pid=1111的任务加入到小核分组 */
echo 1111 > /dev/cpuset/little/tasks

/* Android系统，默认会创建几个分组，框架层把任务分类到不同的分组，可以调整每个分组的cpus，分
析功耗 */
ls /dev/cpuset
background          /* 后台任务 */
foreground          /* 前景任务 */
system-background /* 系统级后台任务 */
```

```
top-app           /* 前景的焦点任务 */
```

- 通过 CPUCTL 限制高负载任务使用 cpu 的带宽(需要打开 CONFIG_CFS_BANDWIDTH 宏)。

```
/* 创建带宽限制分组 */
mkdir /dev/cpuctl/mygroup

/* 设置带宽限制的周期为10ms */
echo 10000 > /dev/cpuctl/mygroup/cpu.cfs_quota_us

/* 每个周期内，组内任务运行总时间不能超过5ms，这个值可以大于cfs_quota_us，因为是多个cpu运行时间总和 */
echo 5000 > /dev/cpuctl/mygroup/cpu.cfs_period_us

/* 把相关任务加入分组 */
echo 1111 > /dev/cpuctl/mygroup/tasks
echo 1112 > /dev/cpuctl/mygroup/tasks

/* cpu.shares表示通过权重限制任务的带宽，用于性能优化，不会影响功耗 */
/dev/cpuctl/mygroup/cpu.shares
```

4.2 DDR 优化

- 场景变频：不同场景配置不同 DDR 频率，4K 视频，录像，双屏显示等。

```
/* 场景定义 */
include/dt-bindings/clock/rk_system_status.h
#define SYS_STATUS_NORMAL       (1<<0)
#define SYS_STATUS_SUSPEND      (1<<1)
#define SYS_STATUS_IDLE         (1<<2)
#define SYS_STATUS_REBOOT       (1<<3)
#define SYS_STATUS_VIDEO_4K     (1<<4)
#define SYS_STATUS_VIDEO_1080P  (1<<5)
...

/* dts配置不同场景的频率 */
arch/arm64/boot/dts/rockchip/px30.dtsi
dmc: dmc {
    compatible = "rockchip,px30-dmc";
    ...
    system-status-freq = <
        /*system status         freq(KHz)*/
        SYS_STATUS_NORMAL       528000
        SYS_STATUS_REBOOT       450000
        SYS_STATUS_SUSPEND      194000
        SYS_STATUS_VIDEO_1080P  450000
        SYS_STATUS_BOOST        528000
        SYS_STATUS_ISP          666000
        SYS_STATUS_PERFORMANCE  666000
    >;
    ...

/* 获取当前系统处于哪个场景 */
cat /sys/class/devfreq/dmc/system_status
0x401
```

- 负载变频：监控负载，自动调整 DDR 频率，负载变频可能会导致性能下降，可以结合场景变频，在某些场景下，固定 DDR 频率。

```
/* dts配置负载变频参数，需要打开dfi节点监控DDR利用率 */
dmc: dmc {
    compatible = "rockchip,px30-dmc";
    ...
    /* 通过dfi监控DDR的利用率 */
    devfreq-events = <&dfi>;
    /*
     * 调频阀值：
     * 当利用率超过40%时，调到最高频，
     * 当负载小于40%且大于40%-20%是维持当前频率
     * 当负载小于40%-20%，会调到一个频率，使得负载差不多为40%-2%/2。
     */
    upthreshold = <40>;
    downdifferential = <20>;

/* 查看当前系统DDR的负载 */
cat /sys/class/devfreq/dmc/load                                    <
33@528000000Hz
```

- DDR DEVFREQ 更详细的配置和优化请查看文档《Rockchip-Developer-Guide-Linux4.4-Devfreq》。

4.3 温控优化

当温度升高到一定程度时，功耗会急剧上升，并且在高压的情况下，表现更明显。

- 改善硬件散热条件。
- 优化软件温控策略，避免温度波动太大。
- 通过软件限制，避免高温时，同时出现高压。

```
&cpu0_opp_table {
    /* 温度超过85度的时候，限制cpu最高电压为1.1V */
    rockchip,high-temp = <85000>;
    rockchip,high-temp-max-volt = <1100000>;

    /* 或者直接通过限制最高主频来避免高压 */
    rockchip,high-temp-max-freq = <1008000>;
};
```

4.4 电源优化

- 降压电路，压降和电流较大时，建议使用 DCDC，提高效率，减小功耗。

例如：
输入 3.3V，输出 1.0V-50mA

电源类型	输入电流	功耗
LDO	50mA	165mW
DCDC(按 80%效率)	18.9mA	62.4mW