Linux Thermal 框架 &电源管理基础

Duzi Huang

2020.7.14.

北京·小米科技园

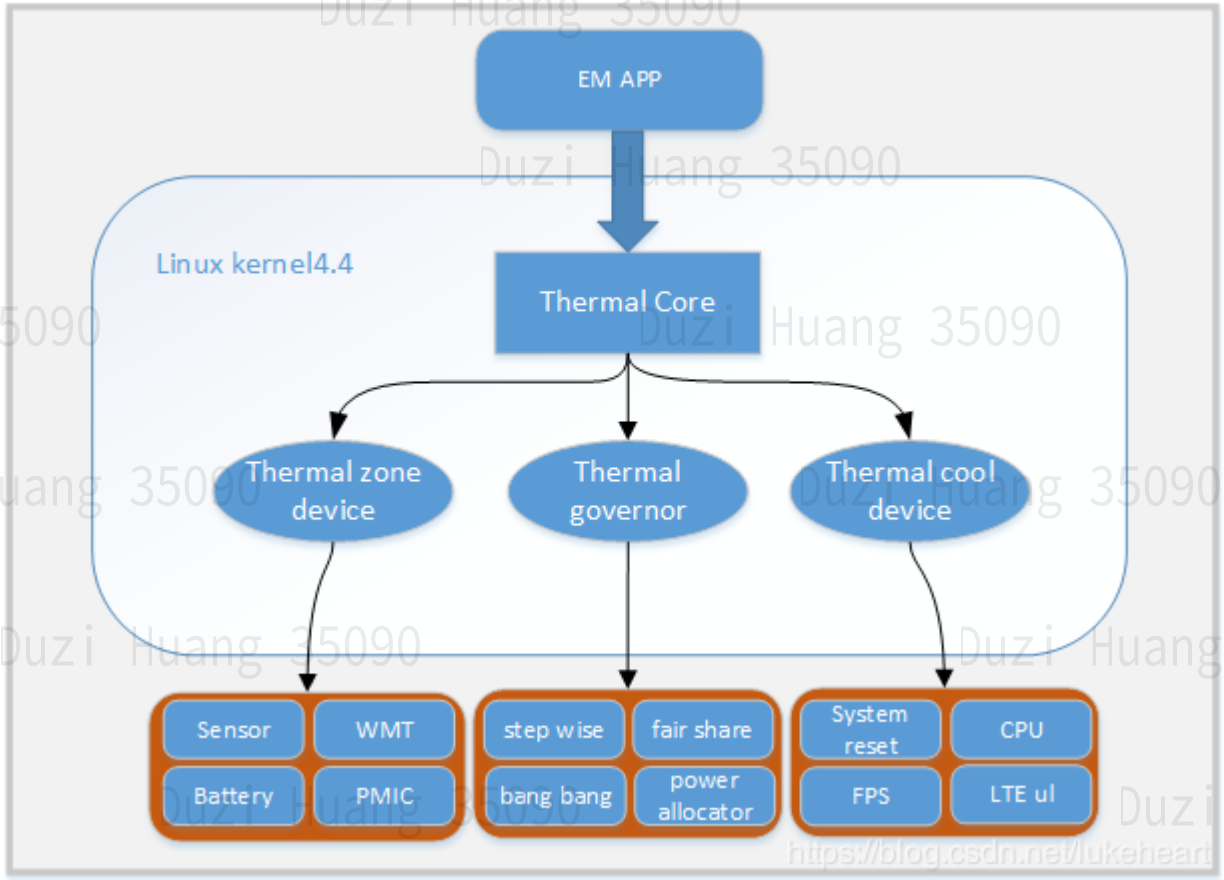
前言

- 1970~1972年，Ken Thompson & Dennis Ritchie为了玩游戏Space Travel，完成了第一版UNIX系统，并在此基础上改进了计算机语言，诞生了C语言
- 1991年21岁的Linus Torvalds，在赫尔辛基大学上学的时候出于个人爱好，编写了LINUX的第一版内核
- 如今基本所有的人类的智能设备和服务器都在运行LINUX系统，基本所有的硬件开发和驱动都是使用C语言，并且现今互联网底层至少60%的通讯协议是使用C语言完成的
- 单纯&无功利心的兴趣和爱好能引导我们造福人类
- 牛马驴无脑奋斗，人类更需要思考
- 下文仅仅提供给热设计工程师简单培训，了解相关领域的基础知识，作者水平有限，难免谬误。指穷于为薪，火传也

目录

Thermal框架图

- Linux thermal框架主要包括
  - 感知层：thermal zone device
  - 控制策略层：thermal governor
  - 执行层：thermal cooling device
  - 中枢：thermal Core
- Thermal zone device: 获取温度设备的抽象
- Thermal cooling device: 降低温度措施的抽象
- Thermal governor: 温控策略，step wise, bang bang, user space, power allocator， fair share
- Thermal core: 作为user space和kernel的接口，同时也是Thermal框架的中枢



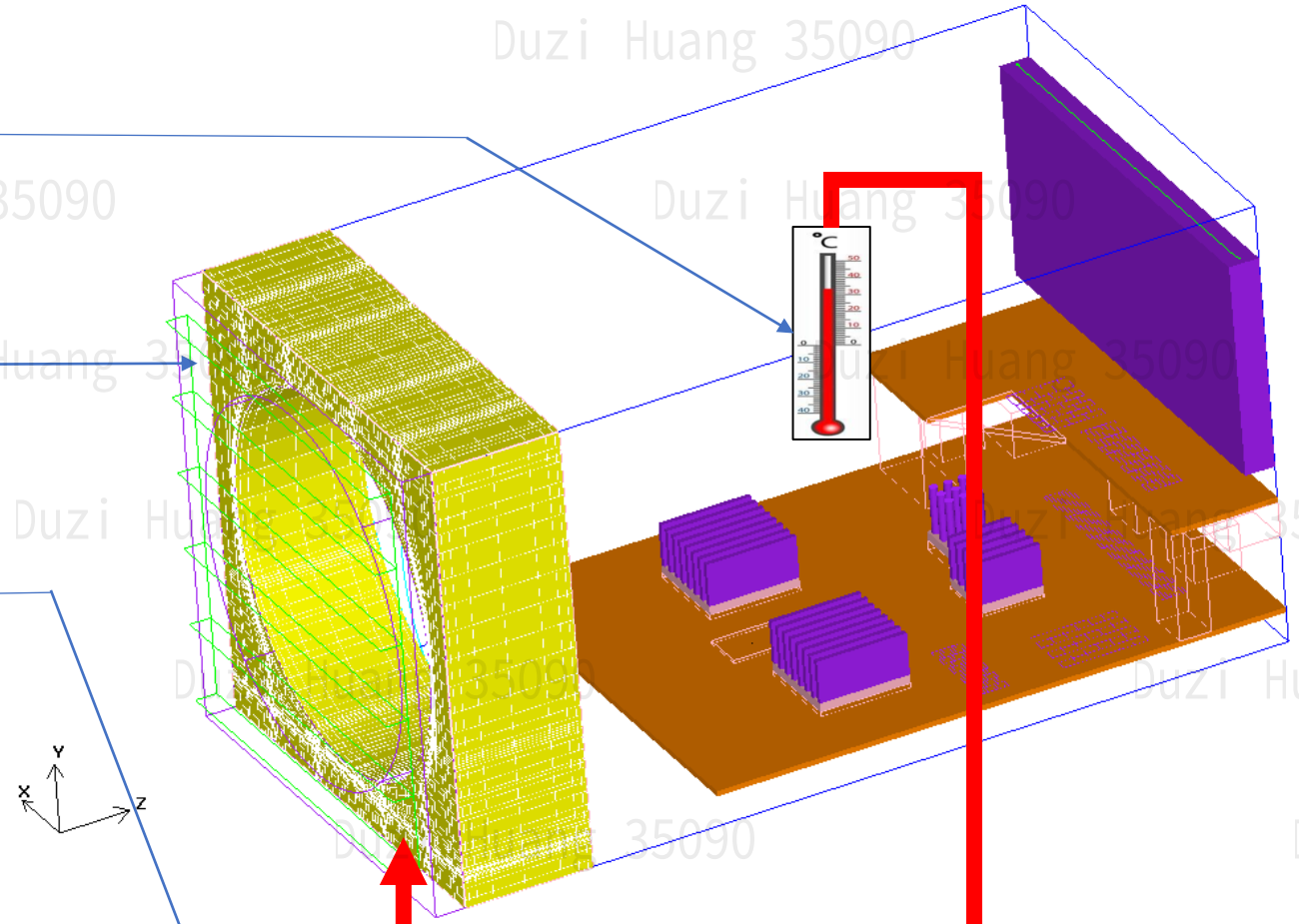
Thermal是Linux内核中的温度控制模块，主要用于控制在系统运行过程中芯片产生的热量，通过传感器设备实时监控设备温度，在温度发生变化时候通过一定的cooling机制促使芯片和外设工作环境温度能够稳定在安全的范围内。因此，不管是在Linux或者Android系统中，Thermal都将对产品的热设计、功耗及CPU频率进行动态平衡性管理。

Thermal Zone Device

Thermal Cooling Device

Thermal Governor

Thermal Core



风扇调速策略（线性/PID）

温度风扇控制模块

Thermal Zone Device

- 一个能提供温度的设备操作函数主要有 : 绑定函数、获取温度函数、获取触发点温度函数
- 绑定函数 : Thermal core 用来绑定用的，绑定函数主要在LTF（Linux Thermal Framework）中实现，zone_device和cooling_device会分别向LTF注册，并在LTF中实现绑定功能
- 温度获取函数：获取设备温度用的，这个也好理解，一般 SOC 内部会有温度传感器提供温度，有些热敏电阻通过 ADC 也算出温度，这个函数就是取这些温度值
- 触发点获取函数：描述什么时候控制的东西就是触发点，每个 thermal zone device 会定义很多触发点，那么每个触发点的温度就是通过该函数获得

Thermal Cooling Device

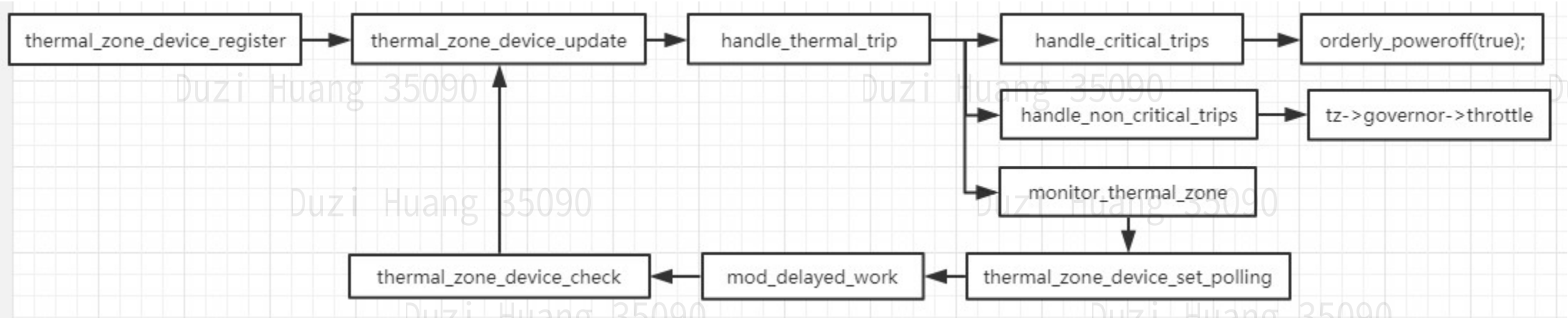
- cooling device即控制温度的设备
- 降温可以从两方面来理解，一个是加快散热，另外一个就是降低产热量。风扇，散热片这些是用来加快散热，CPU,GPU 这些 Cooling devices 是通过降低产热来降温
- Thermal Cooling device 抽象的方式是，认为所有的能降温的设备有很多可以单独控制的状态。例如，风扇有不同的风速状态，CPU/GPU Cooling device 有不同最大运行频率状态，这样当温度高了之后通过调整这些状态来降低温度
- cooling device 维护一个cooling等级，即state，一般state越高即系统的冷却需求越高
- cooling device只根据state进行冷却操作，是实施者，而state的计算由thermal governor完成

Thermal Governor

- Thermal Governor 是降温策略的一个抽象，主要是根据温度来选择 thermal cooling devices 等级的方法，举个简单的例子，当前的温度升高速很快，选择风扇 3 档风，温度升高不快，选择 1 档风。这就是一个 Governor
- 所有的策略都通过 throttle 这个函数实现，内核已经实现了一些策略，step_wise, user_space, power_allocator, bang_bang 等
- step_wise：开环控制，基于温度阈值和趋势，逐步浏览每个cooling设备的cooling等级
- fair_share：基于权重，根据分配的权重划分确定cooling设备等级
- Bang Bang :使用迟滞来突然打开或关闭冷却装置，它的目的是控制风扇,不能节流,但只是打开或关闭。
- user_space：将热区的控制权移交给用户空间，用户可以定制化相关的热功耗参数
- power_allocator：闭环控制，功率基于每个相关设备的功率，温度和当前功耗

Thermal Core

- 有了获取温度的设备，有了温控控制的设备，有了控制方法，Thermal Core 就负责把这些整合在一起
- 注册函数 ,thermal Core 通过对外提供注册的接口，让 thermal zone device、thermal cooling device、thermal governor 注册进来
- Thermal zone/cooling device 注册的过程中 thermal core 会调用绑定函数，绑定的过程最主要是一个 cooling device 绑定到一个 thermal_zone 的触发点上
- Thermal core 使能 delayed_work 循环处理 , 使整个 thermal 控制流程运转起来
- 当温度升高超过温度触发点的话，就会使能对应的 cooling device 进行降温处理



手机端配置举例(MTK：MEIZU15 PLUS)

```
15Plus:/sys/class/thermal $ ls
cooling_device0 cooling_device2 cooling_device4 cooling_device6 thermal_zone1 thermal_zone3
cooling_device1 cooling_device3 cooling_device5 thermal_zone0   thermal_zone2 thermal_zone4
```

系统中位置
一个ThermalZone可以对应到多个cooling_device: 如CPU，GPU都可以通过xo NTC读数触发

```
15Plus:/sys/class/thermal/cooling_device1 $ ls
cur_state max_state power subsystem type uevent
for i in $(seq 0 6)
do
    printf "cooling_device$i\t"
    type=$(cat cooling_device$i/type)
    printf "$type\t"
    cur_state=$(cat cooling_device$i/cur_state)
    printf "$cur_state\t"
    max_state=$(cat cooling_device$i/max_state)
    printf "$max_state\n"
done
cooling_device0         battery_control02       0       1
cooling_device1         battery_control01       0       1
cooling_device2         battery_control00       0       1
cooling_device3         thermal-cpufreq-0       0       13
cooling_device4         thermal-cpufreq-1       0       8
cooling_device5         thermal-gpufreq-0       0       4
cooling_device6         thermal-isp-0           0       2
```

```
for i in $(seq 0 5)
do
    printf "thermal_zone$i\t"
    type=$(cat thermal_zone$i/type)
    printf "$type\t"
    policy=$(cat thermal_zone$i/policy)
    printf "$policy\n"
done

thermal_zone0   mngs-thermal    power_allocator         cdev【0-1】 -> ../cooling_device3
thermal_zone1   APOLLO          step_wise               cdev【0-4】 -> ../cooling_device4
thermal_zone2   GPU             power_allocator         cdev【0-1】 -> ../cooling_device5
thermal_zone3   ISP             step_wise               cdev【0-4】 -> ../cooling_device6
thermal_zone4   battery         step_wise
thermal_zone5   meizu_ntc       step_wise               cdev【0-2】 -> ../cooling_device【0-2】
```

手机端配置举例(Hi Silicon：Honor 9)

cooling_device

```
cooling_device0 thermal-devfreq-0       05
cooling_device1 thermal-cpufreq-0       04
cooling_device2 thermal-cpufreq-1       04
```

thermal_zone

```
thermal_zone0   soc_thermal     power_allocator  cdev【0-2】 -> ../cooling_device[0-2]
thermal_zone1   Battery         user_space
thermal_zone2   cluster0        user_space
thermal_zone3   cluster1        user_space
thermal_zone4   gpu                user_space
thermal_zone5   modem           user_space
thermal_zone6   ddr                user_space
thermal_zone7   system_h        user_space
thermal_zone8   flash_led       user_space
thermal_zone9   charger         user_space
thermal_zone10  pa_0            user_space
thermal_zone11  dcxo0           user_space
thermal_zone12  hisi_shell      user_space
thermal_zone13  hisi_ambient    user_space
```

手机端配置举例(Qualcomm：Pixel)

cooling_device

	type	cur_state	max_state
cooling_device0	thermal-cpufreq-0	1900800	21
cooling_device1	thermal-cpufreq-1	2457600	30

thermal_zone

thermal_zone0	mnh_ipu1	step_wise
thermal_zone1	mnh_ipu2	step_wise
thermal_zone2	mnh_cpu	step_wise
thermal_zone3	mnh_lpddr	step_wise
thermal_zone4	usb	step_wise
thermal_zone5	battery	step_wise
thermal_zone6	usb_port_temp	step_wise
thermal_zone7	pm8998_tz	step_wise
thermal_zone8	pmi8998_tz	step_wise
thermal_zone9	pm8005_tz	step_wise
thermal_zone10	msm_therm	step_wise
thermal_zone11	quiet_therm	step_wise
thermal_zone12	xo_therm	step_wise
thermal_zone13	fpc_therm	step_wise
thermal_zone14	back_therm	step_wise
thermal_zone15	pa_therm	step_wise
thermal_zone16	tsens_tz_sensor0	step_wise
thermal_zone17	tsens_tz_sensor1	step_wise
thermal_zone18	tsens_tz_sensor2	step_wise
thermal_zone19	tsens_tz_sensor3	step_wise
thermal_zone20	tsens_tz_sensor4	step_wise
thermal_zone21	tsens_tz_sensor7	step_wise
thermal_zone22	tsens_tz_sensor8	step_wise
thermal_zone23	tsens_tz_sensor9	step_wise
thermal_zone24	tsens_tz_sensor10	step_wise

注意：

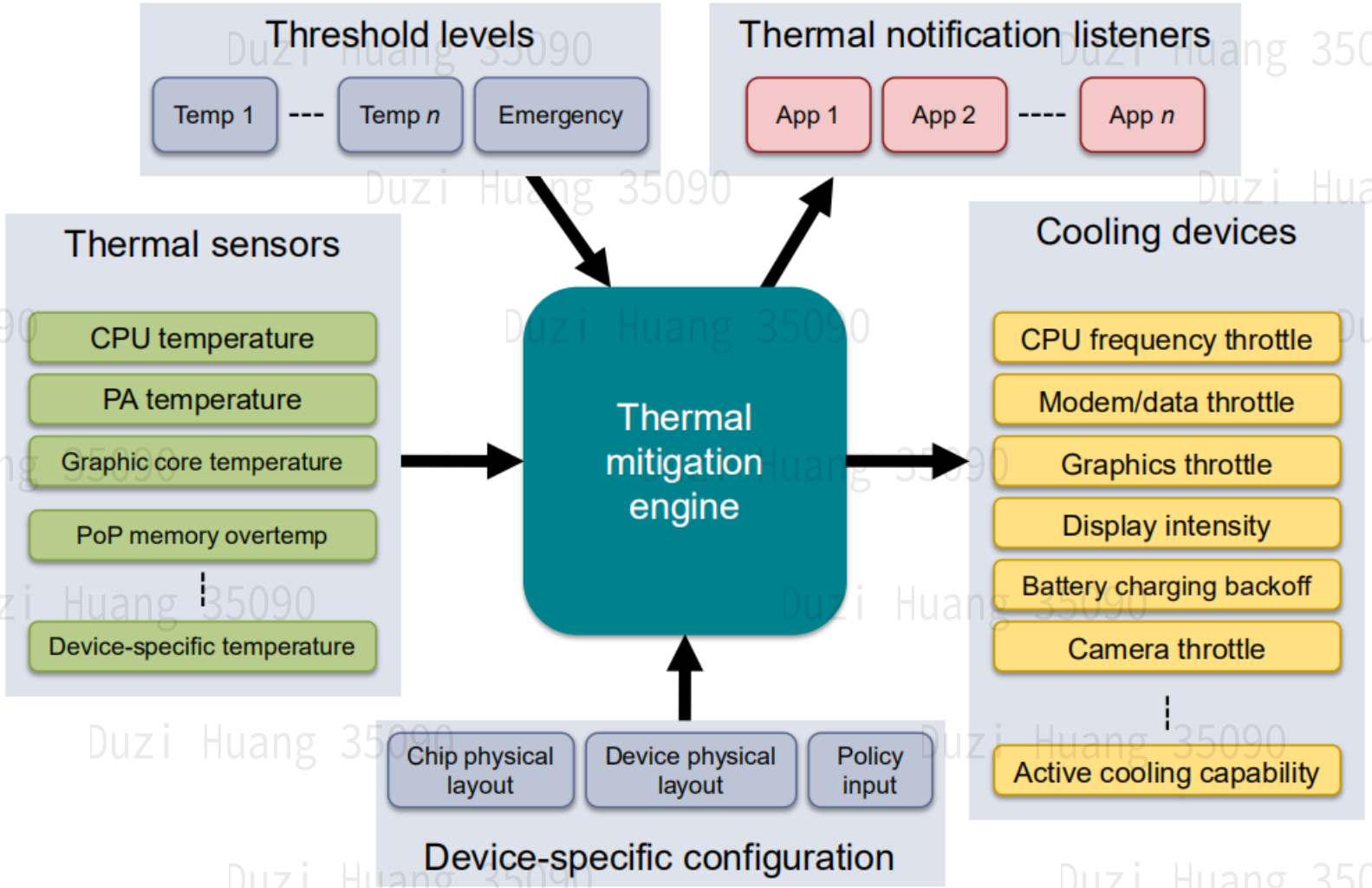
- 以上介绍的是Liunx内核4.4的Thermal框架
- 由于不同的平台在HAL层或者Framework层做了不同的定制化，所以需要根据不同的平台来设计相应的热功耗需求
- linux内核层的thermal框架一般而言是不需要修改的，也是为了保证下层的统一性
- 定制化的需求在上层实现，如：高通采用的thermal-engine，首先上层设置trip point，包括恢复温度与临界温度，thermal engine主要有algo_monitor运作sensor monitor起辅助作用够成

目录

热管理功能概述

功能	变化	描述
Thermal Engine	加入了5G通讯的管控动作	• Thermal Engine扩展到覆盖所有可能的管控动作，所有Thermal Core中注册的Cooling Device能在Thermal Engine中使用 • Thermal Engine可以用来添加任何OEM定制规则 • Thermal Engine和Thermal Core在功能上完全等同了
Thermal Core	为遵从google的GKI(Generic Kernel Image)需求，高通对Linux Thermal Core的定制移到了QGKI中	• 增加了新的thermal sysfs节点来读写现存的Thermal Core规则 • Thermal Core仍然是Thermal Engine和温度传感器&设备驱动的主要接口，Thermal Engine在这个关系中以然作为客户端存在
Battery Current Limit(BCL) Driver	Thermal Software增加了新的传感器socd	• 由于上述变化，Thermal Core不再支持下降的阈值设定 • 新的传感器socd被添加，并将读取电池充电耗尽百分比，也就是说，当soc为60%时，这种新的传感器将电荷耗尽百分比读取为40% • 这使得能够为该传感器设置正温度阈值，并在内核中定义管控策略
CPU Voltage Mitigation	追加了该Cooling Device	• 利用两个核心的电压计划，Cooling Device驱动将在引导过程中建立一个新的频率映射 • 新的频率映射定义了此Cooling Device支持的分级数目
Thermal HAL 2.0	追加了Thermal HAL 2.0	• Thermal HAL 2.0为游戏等客户端提供温度信息，以便应用程序本身能够调整性能，以提供持续使用更好的用户体验。

热管控概念体系架构


Threshold levels
Temp 1

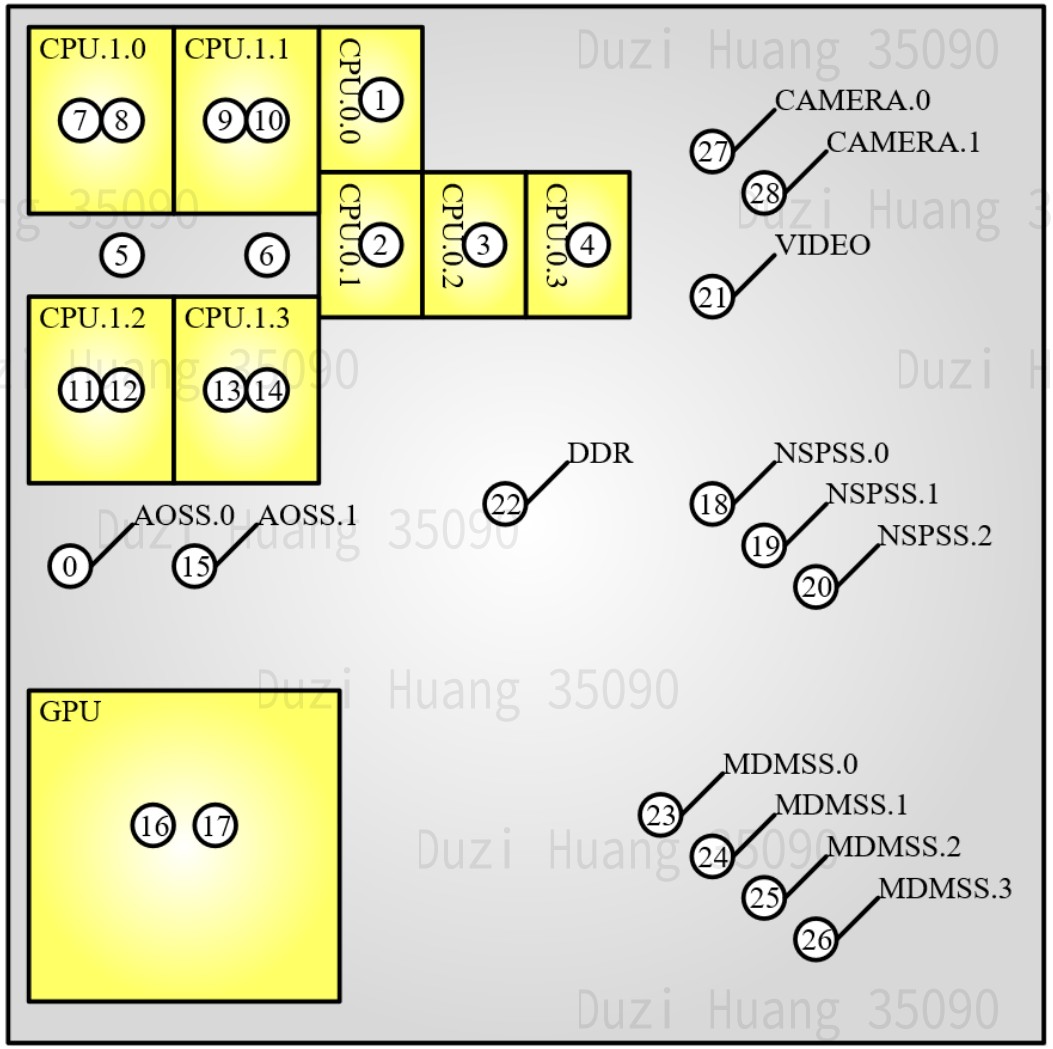
Temp n
Emergency
Thermal notification listeners
App 1
App 2

App n
Thermal sensors
CPU temperature
PA temperature
Graphic core temperature
PoP memory overtemp
Device-specific temperature
Thermal mitigation engine
Cooling devices
CPU frequency throttle
Modem/data throttle
Graphics throttle
Display intensity
Battery charging backoff
Camera throttle
Active cooling capability
Chip physical layout
Device physical layout
Policy input
Device-specific configuration

温度传感器布局

- 靠近die的热点区域布置传感器：SM8350布置了29个TSENS传感器，比SM8250多4个
- PCB上布置更多的NTC，如靠近PA的区域等
- On-die的传感器精度和采样率：SM8350的精度<1.5℃，采样率<1ms
- 当温度超过XBL中预定义的临界高/低阈值时,就会发生硬连接tsens_reset

On-die温度传感器



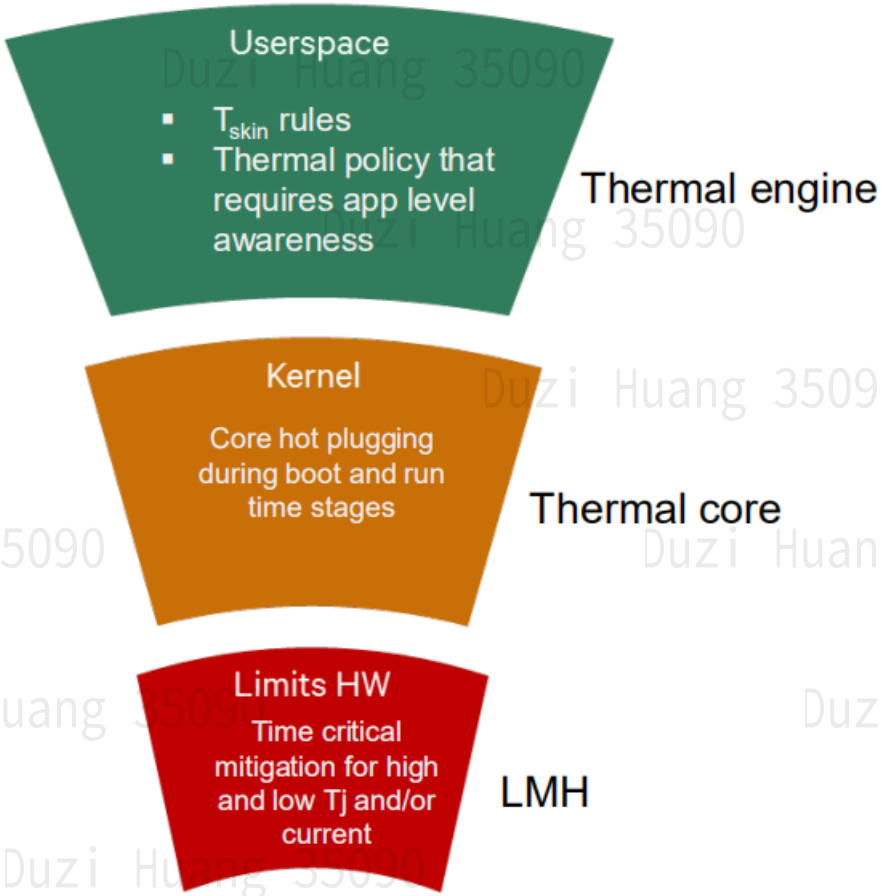
Software Channel	Controller	Port	Name
0	0	0	AOSS.0
1	0	1	CPU.0.0
2	0	2	CPU.0.1
3	0	3	CPU.0.2
4	0	4	CPU.0.3
5	0	5	CPUSS.0
6	0	6	CPUSS.1
7	0	7	CPU.1.0
8	0	8	CPU.1.1
9	0	9	CPU.1.2
10	0	10	CPU.1.3
11	0	11	CPU.1.4
12	0	12	CPU.1.5
13	0	13	CPU.1.6
14	0	14	CPU.1.7
15	1	0	AOSS.1
16	1	1	GPUSS.0
17	1	2	GPUSS.1
18	1	3	NSPSS.0
19	1	4	NSPSS.1
20	1	5	NSPSS.2
21	1	6	VIDEO
22	1	7	DDR
23	1	8	MDMSS.0
24	1	9	MDMSS.1
25	1	10	MDMSS.2
26	1	11	MDMSS.3
27	1	12	CAMERA.0
28	1	13	CAMERA.1

热敏电阻-硬件到软件映射

PIN/GPIO	Source PMIC	Schematic net name	HLOS thermal zone	Location
AMUX_1 (104)	PM8350	SM8350_SKIN_THERM	Skin-msm-therm-usr	Placed near the MSM
AMUX_2 (103)	PM8350	CAM_FLASH_THERM	camera-therm-usr	Placed near the dual rear flash
AMUX_3 (70)	PM8350	HOT_POCKET_THERM	hot-pocket-therm-usr	Placed in the hot pocket
AMUX_4 (82)	PM8350	REAR_CAM_THERM	rear-cam-therm-usr	Placed equidistant from both of the SDR733
GPIO_01 (142)	PM8350	TOF_THERM	tof-therm-usr	Placed near the RFC TOF
GPIO_03 (117)	PM8350	SDR_MMW_THERM	sdr-mmw-therm-usr	Place near the SDR mmW
GPIO_01	PMR735A	BEAMER_THERM1	mmw-pa1-usr	Placed close to north SMR525
GPIO_02	PMR735A	BEAMER_THERM2	mmw-pa2-usr	Placed close to the east SMR525
GPIO_03	PMR735A	BEAMER_THERM3	mmw-pa3-usr	Placed close to the west SMR525
GPIO_01	PMR735B	PA_THERM1	pa_therm1-usr	–
GPIO_02	PMR735B	PA_THERM2	pa_therm2-usr	–
GPIO_02 (124)	PM8350B	WLC_THERM	wlc-therm-usr	Placed near the IDT wireless charger
USB_THERM (92)	PM8350B	USB_CONN_THERM	conn-therm-usr	Placed near the USB-C
XO_THERM (32)	PMK8350	XO_THERM	xo-therm-usr	–

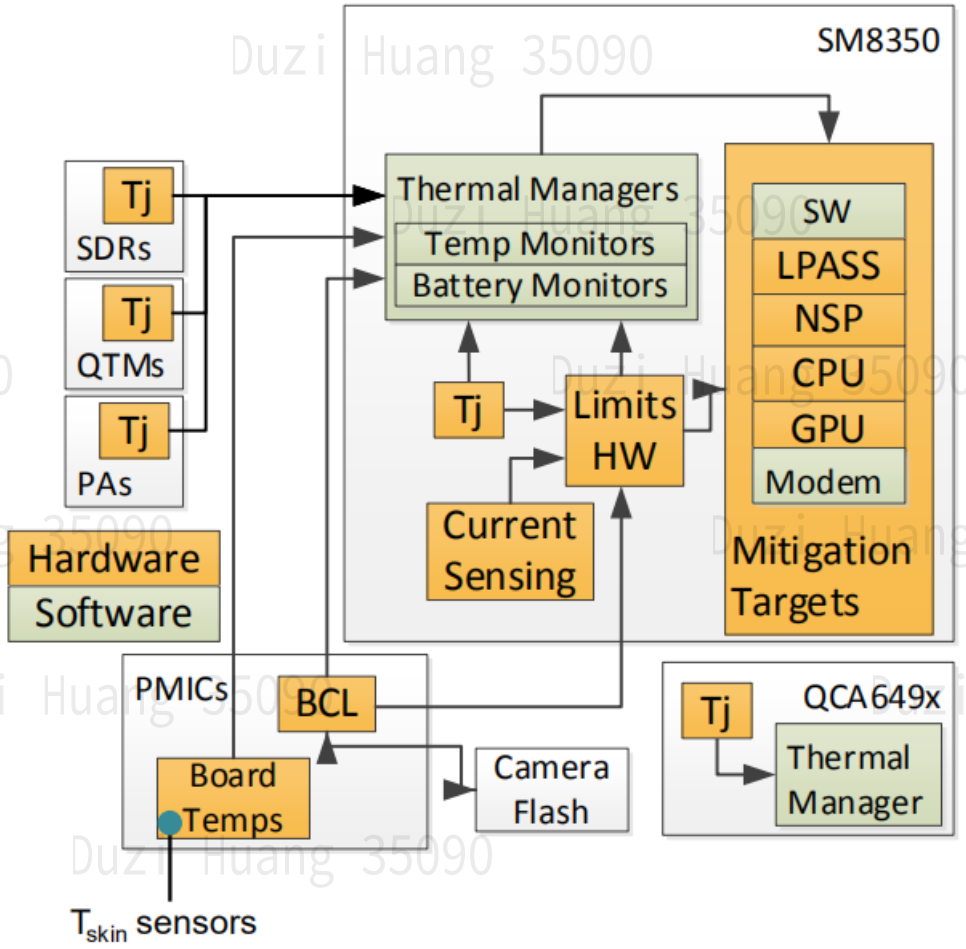
热管理软件架构

- 热管理目的
  - 管理芯片温度（典型值：85℃ to 95℃）
  - 管理外部设备的壳温
    - 金属40℃，塑料45℃
- 传感器
  - 芯片结温传感器
  - 板载热敏电阻(NTC): PA, XO, PMIC等
- 管理设备
  - 降低性能的被动冷却
  - OEM可以配置设备选择和阈值,通过配置文件调整id的可变性
- 架构
  - User Space-自定义T_{skin}规则，应用层感知
  - Kernel-core隔离
  - Limits Hardware-高、低结温或电流的响应时间敏感型管控



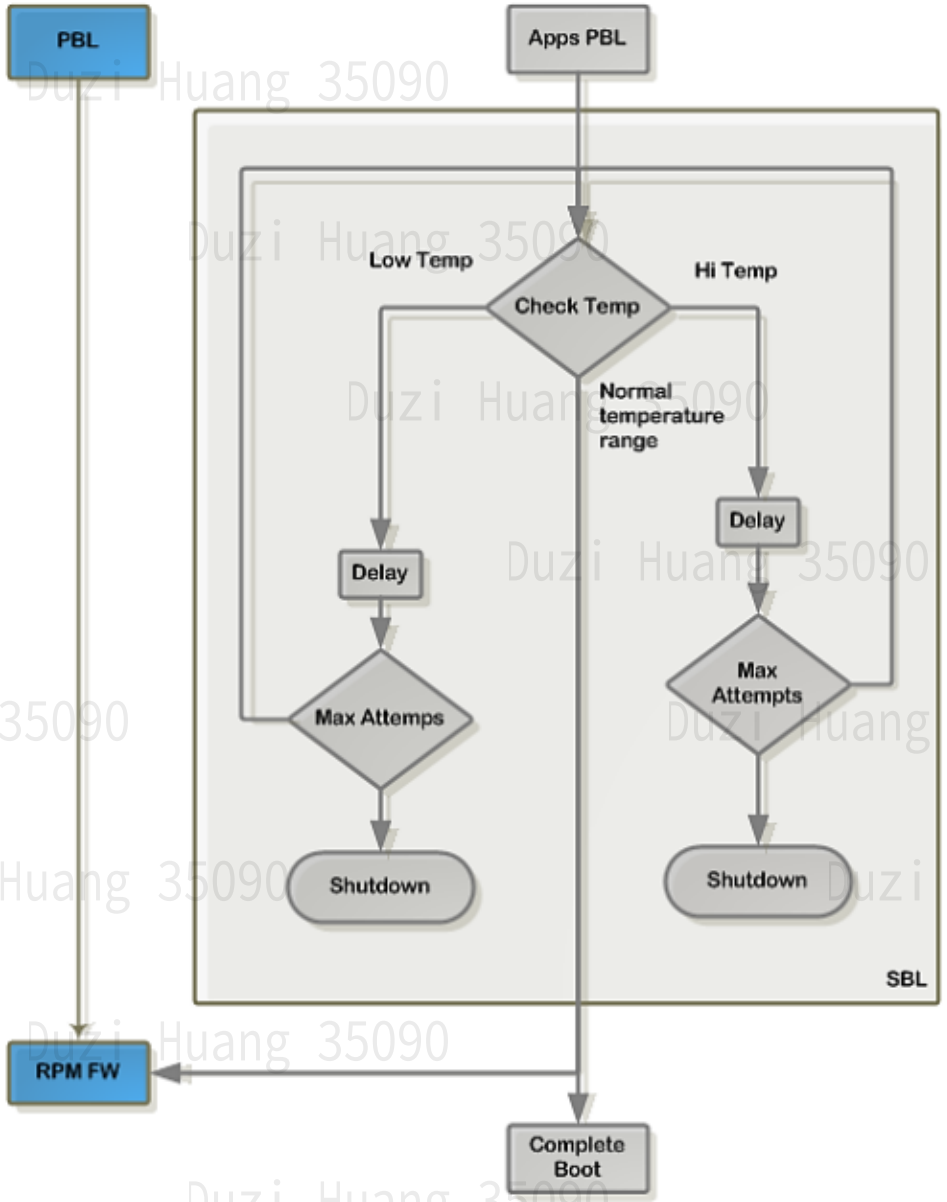
热管理的角色

- 从这个芯片组开始，Thermal Engine能够作为主要的热软件框架来添加定制的OEM热规则
- 背景：
  - Google已经主动从核心内核映像(gki)中删除供应商的特定代码，这样他们就可以在不破坏qti代码的情况下进行更改
  - 因而，很多依赖于Thermal Core特性的QTI热管理不得不删除，如：
    - Low_Limits_Floor和Low_Limits_Cap调节器
    - Cooling Device上下限在Thermal Zone sysfs的入口、轮询延迟和被动轮询延迟的入口
    - Cooling Device支持的最小底部管控
- Thermal Engine被扩展到包括5G的热管控
- Thermal Core仍将遵循以下功能：
  - core隔离
  - GPU Tj 管理
  - Tskin管理（Thermal Engine的替代方式）

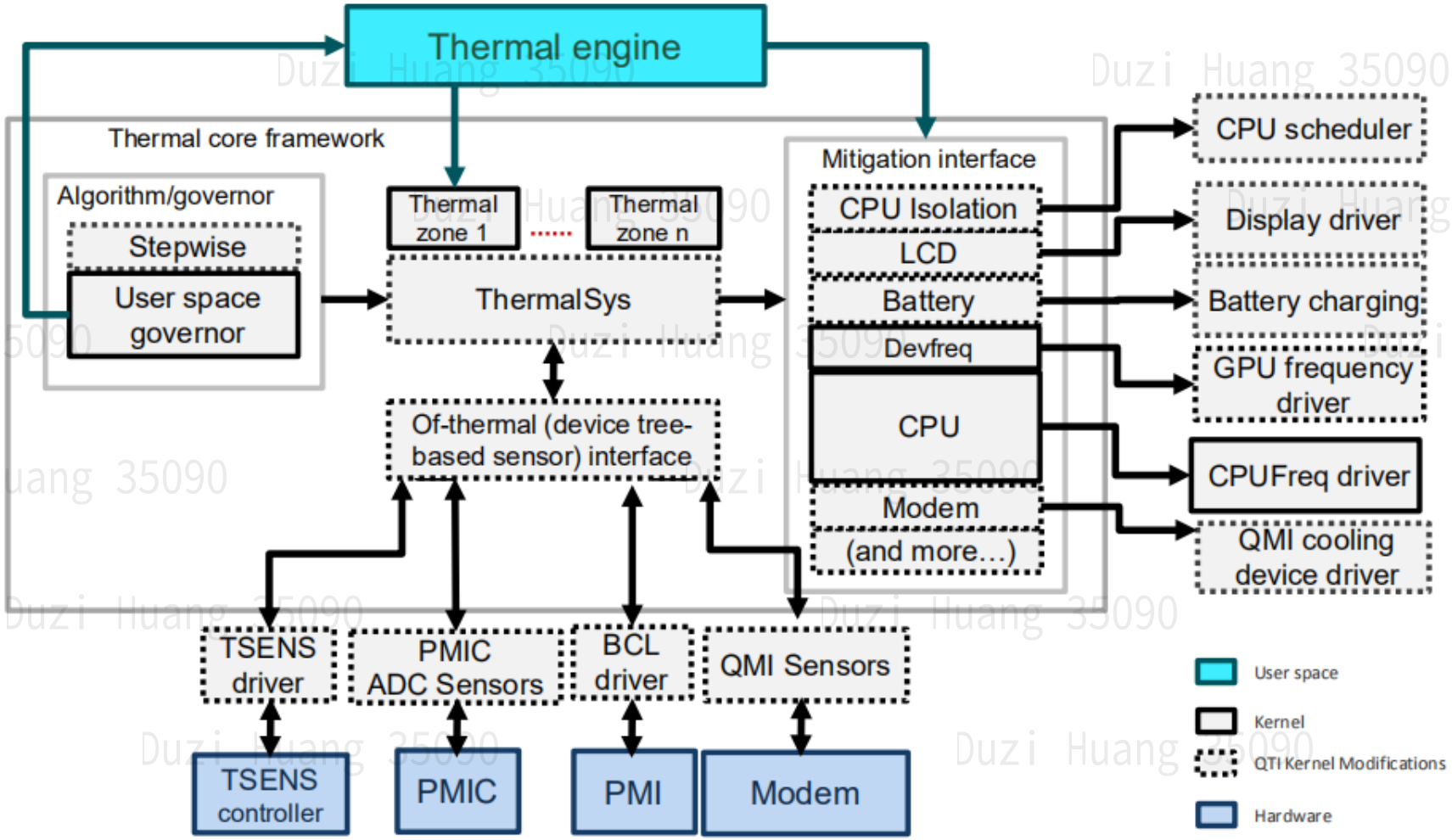


BTM-开机热管理算法

- 开机热管理（Boot Thermal Management）
  - 设备开机前，确保整机温度处于一个合法的区间内
  - 在引导加载程序构建中配置温度阈值、延迟和最大尝试
- BTM默认禁用：
  - nLowerThresholdDegC设置为-150℃
  - nUpperThresholdDegC设置为150℃
- 如果OEM指定了正确的温度阈值，则可以推迟启动，如果当前温度于或低于相应的阈值
- 如果超过预定义的重试次数，引导可能会失败
- 在引导加载程序库中实现：
  - Qcompkg / library / boottempchecklib / boottempcheck.c



Thermal Core框架-架构



Thermal Core框架-特色亮点

- Thermal Core在内核中取代了内核热监控（KTM）
- 当User Space调控器在Thermal Zone中使用时，当传感器阈值被触发，将向Thermal Engine发送通知
- 管控动作通过Cooling Devices聚合
- 热接口解析设备树以及聚合阈值并发送给传感器驱动程序

Thermal Core框架-设备树入口

- 芯片组dtsi文件中的带注释的Thermal Zone示例

Thermal Zone Name	pop-mem-step {
Polling rate after 95C interrupt received	polling-delay-passive = <10>;
Polling rate for non-interrupt driven sensors	polling-delay = <0>;
Thermal sensor being monitored (tsens 2 and controller 1)	thermal-sensors = <&tsens1 2>;
Thermal algorithm that will be used to throttle cooling dev	thermal-governor = "step_wise";
Trip point(s) definitions	trips {
Name of trip point	pop_trip: pop-trip {
Trip point value	temperature = <95000>;
Clear point relative to temperature above	hysteresis = <0>;
Passive cooling device (e.g. CPU, GPU)	type = "passive";
	};
	};
Cooling action definitions (i.e. action to take based off trip)	cooling-maps {
Name of cooling map	pop_cdev4 {
Reference to trip point that will activate cooling map	trip = <&pop_trip>;
Action taken based off of trip. Throttle CPU4, no lower limit bound, Max mitigation is last level - 1	cooling-device =<&CPU4 THERMAL_NO_LIMIT (THERMAL_MAX_LIMIT-1)>;
	};

读取sysfs中的管控等级

- 管控等级可以通过sysfs中Cooling Device的节点读取：
- sys/class/thermal/cooling_deviceX/cur_state
- cur_state 代表管控指数，而不是实际数值
- 如果没有管控，cur_state数值是0
- 数值大于0代表逐步加深的管控等级

- Cooling device mitigation level

Mitigation level	Performance level
0	Full performance
1	Full perf - 1
2	Full perf - 2
3	Full perf - 3
. . .	. . .
n	Minimum performance

Thermal Core更新

- Thermal Zone文件夹中多了名为“config”的新的sysfs节点，这个节点列出了当前Thermal Zone的设置，格式和Thermal Engine的config文件类似

```
sensor pm8350c-bcl-lvl0
policy step_wise
polling_delay 0
passive_polling_delay 100
trip 0
trip_threshold 1
trip_threshold_clr 1
device  thermal-devfreq-0+cpu-isolate5+cpu-isolate4
upper_limit  2+1+1
lower_limit  2+1+1
```

- 添加了一个新的debug sysfs条目， 用于User Space,以修改现有的Thermal Zone配置
  - /sys/kernel/debug/thermal/config
  - 对于写入上述节点的单个写入,可以更改单个Thermal Zone配置
  - 只能更改现有的Cooling Device和触发配置
  - 多个热区配置更改必须一个接一个地写入

echo “sensor pm8350c-bcl-lvl0\n policy step_wise\n polling_delay 0\n passive_polling_delay 100\n trip 0\n trip_threshold 1\n trip_threshold_clr 1\n device thermal-devfreq-0+cpu-isolate5+cpu isolate4\n upper_limit 2+1+1\n lower_limit 2+1+1\n” > /sys/kernel/debug/thermal/config

User Space Thermal Engine

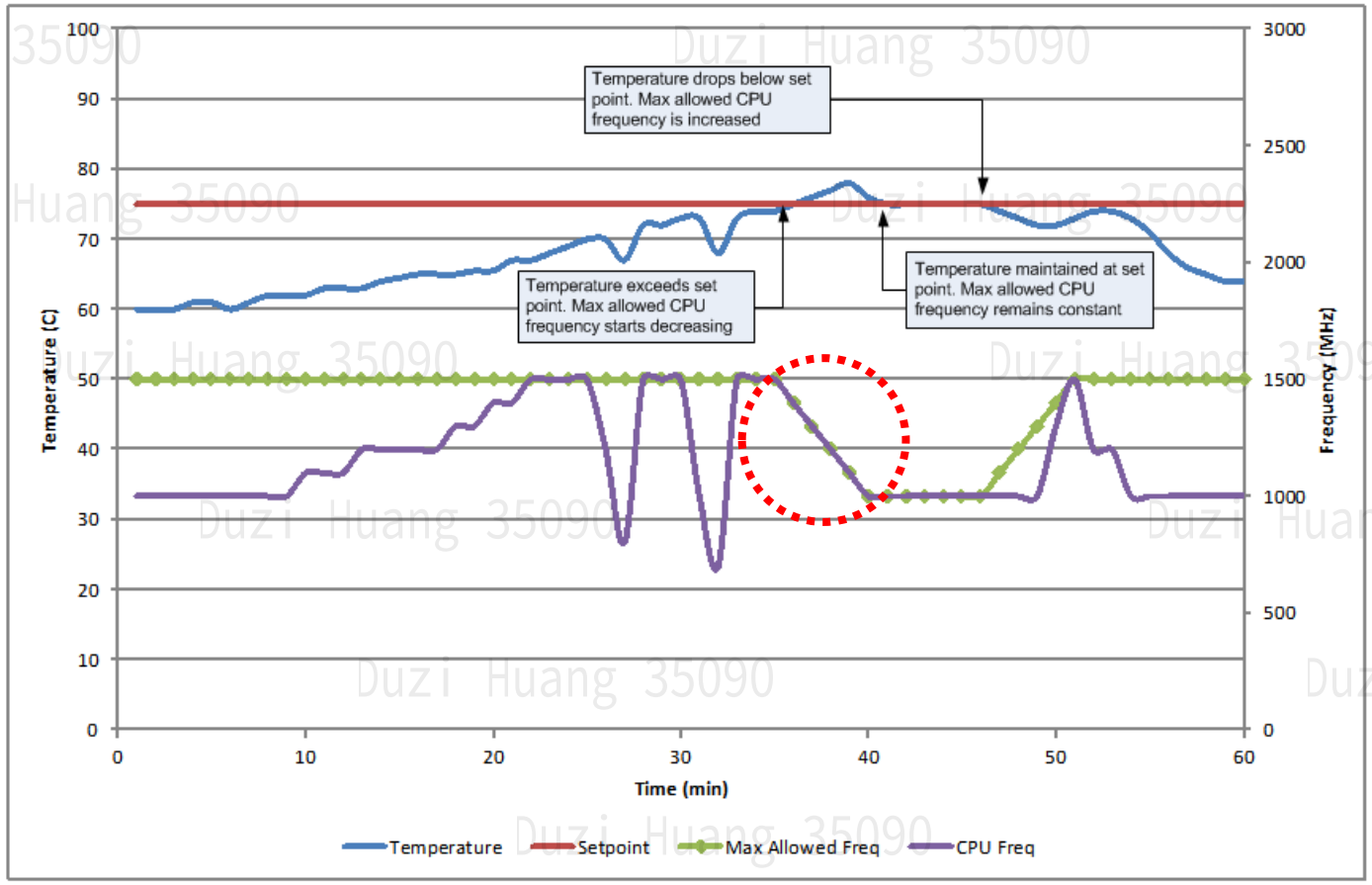
- Thermal Daemon进程最初在msm8660芯片组商用
- Thermal Engine已经重新设计了Thermal Daemon进程，以支持与传感器管理器的集成，并允许多种算法
  - Thermal Engine-支持并行运行的监视(monitor)和动态热管理(DTM)算法
  - Dynamic Algorithm-展示了较少的调优工作量和改进的平均DMIPS (Dhrystone Million Instructions executed Per Second)
  - Virtual Sensor-两个以上的传感器读数和相关的权重的组合，以准确地关联壳温
  - Dynamic Parameter Update-重要的Thermal Engine配置参数集可以在运行时更新,以OEM更好地定制DTM

热控算法

- 在设置文件中可以使用多种类型的算法，主要包括：
  - DTM-algo_type ss
  - Monitor-algo_type monitor
- DTM
  - 设置：algo_type ss
  - OEM设定一个温度阈值，算法相应地动态地限制OEM指定的操作(如：CPU频率)
  - 用于壳温和结温控制
    - 相对于monitor算法，仅需叫少的调制工作
    - 温度限制动作能得到强化
  - 仅用于CPU&GPU管控(如：降低运行的最大主频)
  - 可以在thermal-engine.conf文件中修改set_point和set_point_clr，来改变触发和清除温度
- Monitor (for: LCD, Modem, Battery)
  - 设置：algo_type monitor
  - OEM制定一系列触发阈值，并针对每个阈值设定一个精确的响应动作
  - Monitor算法适用于管控LCD, MODEM, BATTERY，不适用于CPU, GPU，它需要对每个触发点进行广泛的调制工作
  - 可以在thermal-engine.conf文件中修改thresholds和threshold_clr，来改变触发和清除温度
- 建议使用DTM作为CPU和GPU的管控算法，因为它可以极大地减轻调优工作，提高性能，并更严格地将温度保持到OEM设置点

DTM算法示例

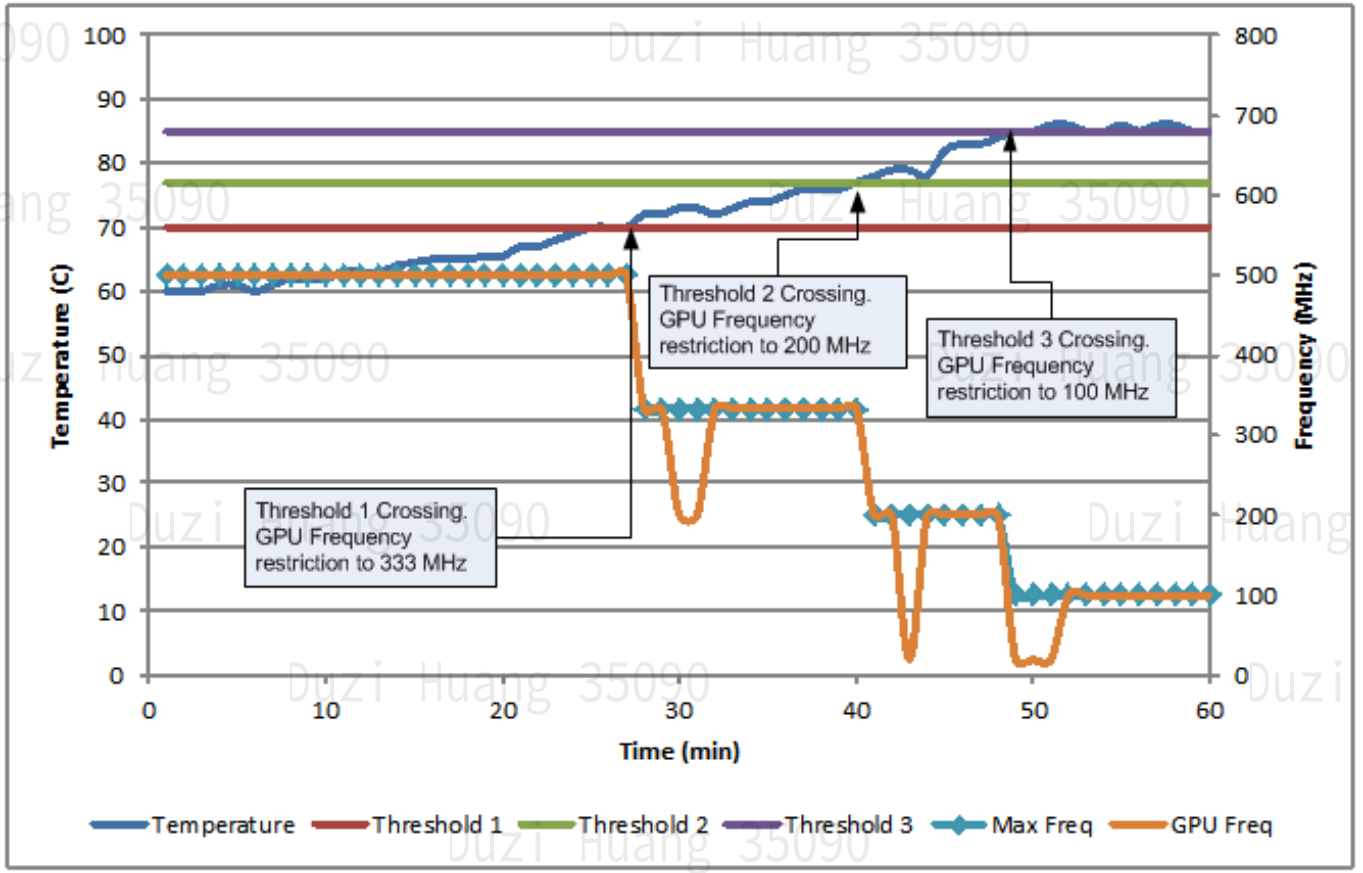
- 当温度超过设定点(75 ℃)时,降低性能,直到温度稳定。
- 根据规则上定义的sampling参数启用轮询模式。
- 当温度超过设定的阈值时降低性能,并随着温度低于设定点(75 ℃),最大允许频率能够爬升。（此处注意!动态的,不是一步切到指定频率）
- 如果温度进一步下降到设定点(50 ℃)以下,中断模式将重新启用。
- 仅用于CPU / GPU控制。



Threshold算法示例

- 测量的温度跨越预定义的阈值，然后设置预定义的缓解级别。
- 当传感器温度达到70 ℃时，GPU频率由500 MHz降低到333 MHz。
- 最终传感器温度维持在85 ℃。

Parameter	Level 1	Level 2	Level 3
Threshold set	70°C	77°C	85°C
Threshold clear	65°C	70°C	77°C
GPU frequency	333 MHz	200 MHz	100 MHz



Thermal Engine配置文件I

- 基本格式

{debug}　　//可选，若有此选项表示使能调试息

sampling　　// 默认采样率，以ms为单位

[<算法实例标签>]　　// 算法实例标签

algo_type　　// 算法类型，必须是算法实例的第一个字段

disable　　//可选， 用于默认情况下禁止该算法实例起作用

……

Debug mode and default sampling

```
debug
sampling    1000
```

Monitor algorithm instance example

```
[BATTERY-CHARGING-EXAMPLE-RULE]
algo_type       monitor
Sensor          quiet_therm
Sampling        2000
Thresholds      43000 45000
thresholds_clr  41000 43000
Actions         battery
action_info     2 3
```

#Dynamic algorithm instance example

```
[SS-CPU0]
algo_type ss
sampling 10
sensor cpu0-silver-usr
device cpu0
set_point 85000
set_point_clr 65000
time_constant 0
```

Thermal Engine配置文件II

monitor算法配置格式

```
[<算法实例标签>]
algo_type       monitor
sensor          <传感器名称>
sampling        <采样率(ms)>
descending      //    可选，默认门限是升序，有此字段后门限顺序为降序
thresholds    <门限值(mC/mA)>
thresholds_clr  <清除门限值>
actions       <达到门限时的动作，多个动作时用'+'连接>
action_info   <动作额外信息，多个额外信息间用'+'连接>
```

ss算法配置格式

```
[<算法实例标签>]
algo_type       ss
sampling        <采样率(ms)>
sensor          <传感器名称>
device          <被 ss 算法调整的设备>
set_point       < ss 算法调整的目标值(mC/mA)>
set_point_clr   < ss 算法停止调整的值>
time_constant  <当当前和上一个错误采样值相等时的延迟调整系数>
device_max_limit    <设备最大值(mC/mA)>        // 可选
```

pid算法配置格式

```
[<算法实例标签>]
algo_type       pid
sampling        <采样率(ms)>
sensor          <传感器名称>
device          <被PID算法调制的设备>
set_point       <PID算法调整的目标值(mC/mA)>
set_point_clr   <PID算法停止调整的值>
p_const         <PID算法中的P常量>
i_const         <PID算法中的I常量>
d_const         <PID算法中的D常量>
i_samples       <积分组件报错时的积分样本数>
dev_units_per_calc  <积分算法输出调整单元值(每次调整的值)>
```

virtual算法配置格式

该算法实际作用是用已存在的传感器模拟出一个新的传感器。

```
[<虚拟传感器名字>]
algo_type       virtual
trip_sensor     <传感器名称>
set_point       <当高于此温度时，虚拟传感器开始polling模式>
set_point_clr   <当高于此温度时，虚拟传感器停止polling模式>
sensors         <用于计算温度总和的传感器数组>
weights         <权重值数组>
sampling        <默认采样率>
```

- 算法类型：算法类型共有4种： monitor, pid, ss, virtual。
- SS既DTM策略，即Dynamic Thermal Management，动态控制算法有两个模式，DTM(用于频率控制)和PID(用于温度保持)。DTM是基于温度setpoint，提升或这降低CPU的最大允许频率。每一个最大允许频率的调解是基于DCVS frequency table中的可用频率steps。温度高于setpoint时，最大允许频率会按step逐渐下降，温度低于setpoint_clr时，最大允许频率会按step逐渐上升。该策略可以用于控制GPU频率

Thermal Engine配置文件III

- 配置文件字段解释
- thresholds取值
  - 'thresholds'/'thresholds_clr'/'actions'/'action_info'接受最多8级以空格分开的门限
- action取值
  - 'none':
    - actions: 无动作
    - action_info: 忽略
  - 'report':
    - actions: 报告"穿过门限消息"给UI
    - action_info: 忽略
  - 'cpu':
    - actions: 调整cpu频率
    - action_info: cpu最大允许频率，单位：KHz
  - 'cpuN':
    - actions: 调整第N个cpu频率
    - action_info: cpuN最大允许频率，单位：KHz
  - 'hotplug_N':
    - actions: 拔掉第N个cpu(cpu_down)
    - action_info: 0: 插上cpu(cpu_up)，1:拔掉cpu(cpu_down)
  - 'lcd':
    - actions: 限制lcd亮度
    - action_info: lcd最大亮度，取值范围：0-255
  - 'modem':
    - actions: 请求限制modem功能
    - action_info: modem的限制级别，取值范围:0-3
  - 'fusion':
    - actions: 请求限制 fusion modem功能
    - action_info: fusion modem的限制级别，取值范围:0-3
  - 'battery':
    - actions: 限制电池充电电流
    - action_info: 电池充电电流限制级别，取值范围:0-3
  - 'gpu':
    - actions: 调整gpu频率
    - action_info: gpu最大允许频率，单位：KHz
  - 'wlan':
    - actions: wlan(无线局域网)限制
    - action_info: WLAN限制级别，取值范围0-4
  - 'shutdown':
    - actions: 关机
    - action_info: 关机延迟时间,单位:ms

Thermal Engine配置文件示例I

```
sampling        1000

[PMIC_THERM_MON]

algo_type       monitor
sensor          PMIC_THERM
sampling        5000
thresholds      40200       45000       50000
thresholds_clr  38000       43000       48000
actions         cpu+report  cpu         cpu
action_info     1188000+0   368640      245760
```

描述：

- 默认采样率为1s；
- PMIC_THERM传感器采样率为5s；
- 当温度升到到40.2度以上时，触发门限1，调节cpu最大允许频率为1188000KHz，在本例中，由于此频率是最大值，因此实际无动作；同时上报此消息，action_info值0被忽略；
- 当温度降低到38度以下时，清除门限1，并上报此消息；
- 当温度升高到45度以上时，触发门限2，调节cpu最大允许频率为368640KHz；
- 当温度降低到43度以下时，清除门限2，调节cpu最大允许频率为1188000KHz。

Thermal Engine配置文件示例II

```
sampling        2000
[bcl_monitor]
algo_type       monitor
descending
sensor          bcl
sampling        1000
thresholds      100     0
thresholds_clr  500     100
actions         report  report
action_info     0       0
```

描述：

- 使能调试信息输出；
- 默认采样率为2s；
- 对于电池电流限制(battery current limit)传感器，采样率为1s；
- 当电池电流升到imax - 100mA时，触发门限1，并上报此消息；
- 当电池电流降到imax - 500mA时，清除门限1，并上报此消息；
- 当电池电流升到imax时，触发门限2，并上报此消息；
- 当电池电流降到imax -100mA时，清除门限2，并上报此消息。

Thermal Engine配置文件示例III

```
debug
[virtual_sensor0]
algo_type          virtual
sensor             tsens_tz_sensor8
set_point          35000
set_point_clr      30000
sensors            tsens_tz_sensor1  tsens_tz_sensor5
weights            40  60
sampling           250

[Test-PID]
algo_type  pid
sensor             virtual-sensor-0
device             cpu1
sampling           250
set_point          55000
set_point_clr      50000
p_const            1.25
i_const            0.8
d_const            0.5
i_samples          10
dev_units_per_calc      5000
```

描述：

- PID实例Test-PID基于virtual-sensor-0的结果；
- virtual-sensor-0需要用户手动定义；
- trip_sensor 用来指示虚拟传感器何时开始进入polling模式(轮询)，trip_sensor必须为常规传感器，不能为另外一个虚拟传感器；
- set_point是trip sensor的门限值，当高于此门限值时，trip sensor将从中断模式进入轮询模式，轮询频率由虚拟传感器的sampling字段设置；
- set_point_clr是trip sensor的门限值，当低于此门限值时，trip sensor将停止轮询模式然后等待下一个门限事件；
- weights给定了传感器数组的权重值；
- 虚拟传感器的set_point必须小于pid算法的set_point，以便当达到set_point时pid能收到通知，另外，如果虚拟传感器未进入轮询模式，pid将不能获取到它的温度；
- 如果tirp sensor不支持从中断模式到查询模式的改变，此时虚拟传感器的采样率将应该和pid的采样率一致。

添加定制配置到Thermal Config

- 无需重新编译源代码，可以添加定制规则到config中
- 通过将新规则放在名为thermal-engine.conf的文件中,并使用ADB推送到设备(/system/etc/thermal-engine.cong),添加一个规则:

```
adb push <location_of_thermal-engine.conf> /vendor/etc/thermal-engine.conf
```

- 如：添加一个对GPU的管控规则，把如下代码放入thermal-engine.conf:

```
[SS-GPU]
algo_type ss
sampling 65
sensor gpu
device gpu
set_point 60000
set_point_clr 57000
time_constant 0
```

- 为使得以上规则生效需要重新启动一下thermal engine:

```
stop thermal-engine
start thermal-engine
```

Thermal Engine的管控设备

管控设备	描述
cpuX	Adjustment of maximum allowed operating frequency per cluster For silver cluster mitigation, use cpu0 For gold cluster mitigation, use cpu4 For gold prime core, use cpu7
thermal-cluster-7-4	This is the preferred method for mitigating gold and gold prime cores based on power and performance efficiency.
gpu	Adjustment of maximum allowed operating frequency
cpu-isolateX	Takes specific core offline
shutdown	Safety mechanism to ensure that the CPU cores shut off before junction temperature limits are exceeded
lcd	Adjustment of maximum backlight intensity
Battery	Adjustment of maximum allowable charge rate
nsp	Adjustment of maximum allowed operating frequency

5G Thermal Engine的管控设备

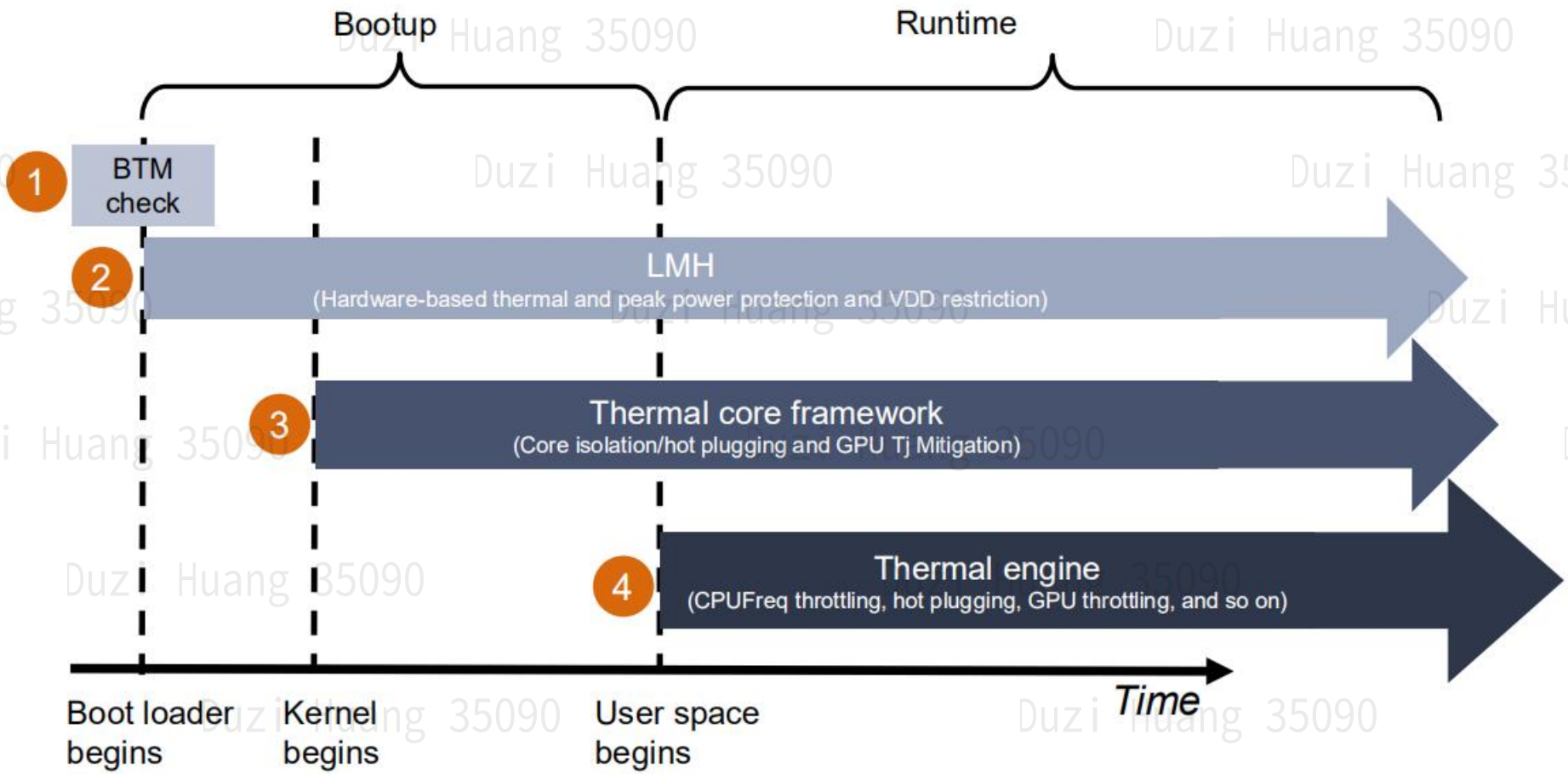
管控设备	描述	相关sensors
modem_mmw_skin0-3	毫米波天线模组壳温管控，减少天线元素	热敏电阻布置在天线模组附件
modem_skin	Modem壳温管控，CC减少，LTE回退	与modem活动相关的热区布置的热敏电阻
modem_pa	4G PA管控	PA的热敏电阻
modem_pa_fr1	Sub6 PA管控	Sub6 PA热敏电阻

LMH

- LMH(limits management hardening)是一种基于硬件的保护电路
- LMH通过以下方式增加了设备的可靠性和健壮性:
  - 在PMIC电源轨道的指定能力范围内管理CPU子系统消耗的峰值电流
  - 如果任何CPU过热,则提供快速的热管理响应;这适用于电源和性能集群
- LMH DCVSh使CPU供电在硬件级别上根据CPU时钟频率降低
- 从SM8250芯片组开始, LMH DCVSh内核驱动程序已被弃用，这意味着HLOS (Thermal Core Framework)不再表决LMH的频率变化,而是直接与cpufreq驱动程序对话。
- LMH的特性总结，如下表:

Feature	CPU	GPU	NSP
Peak current management	Yes Applies only to Gold cluster	No	Yes Controls all of CX rail
Thermal management	Yes • Junction temperature management using hardware • Applies to both clusters	No Thermal management is still handled by the thermal core software	Yes Junction temperature management using hardware
BCL	Yes	Yes	Yes
VDD restriction	Yes	Yes	Yes

热管理开机时序


Bootup
Runtime
1
BTM check
2
LMH
(Hardware-based thermal and peak power protection and VDD restriction)
3
Thermal core framework
(Core isolation/hot plugging and GPU Tj Mitigation)
4
Thermal engine
(CPUFreq throttling, hot plugging, GPU throttling, and so on)
Time
Boot loader begins
Kernel begins
User space begins

5G热管控策略

Thermal level	LTE MDM mitigation	LTE PA mitigation	5G NR Sub6 MDM mitigation	5G NR Sub6 PA mitigation	5G NR mmW MDM mitigation	5G NR mmW PA mitigation
Level 1	NA	UL Throttling	• 2Rx Fallback • Drop SCells (all SCells dropped)	UL Throttling	Drop SCells (All SCells Dropped)	QTM Element Reduction UL Throttling
Level 2	▪ 2Rx Fallback ▪ Drop SCells (one by one)	MTPL Backoff Duty Cycling	LTE Fallback	• MTPL Backoff Duty Cycling • LTE Fallback	Drop PSCell	QTM Module Switching
Level 3	Limited Service	Limited Service	Limited Service	Limited Service	Limited Service	• NSA → LTE Fallback • SA → Limited Service

Mitigation details are subject to change

- 根据热需要和仿真,分别为每个传感器(TMD)设置温度阈值。
- 表中的每一行都对应一个温度级别。
- 如果UE在NSA模式下运行,如果Modem管控正在发生, 5G管控首先发生,然后是LTE管控。
- PA管控独立发生在LTE和NR上。
- modem_skin操作与MDM Tj相同,但温度水平可以由OEM设置。

目录

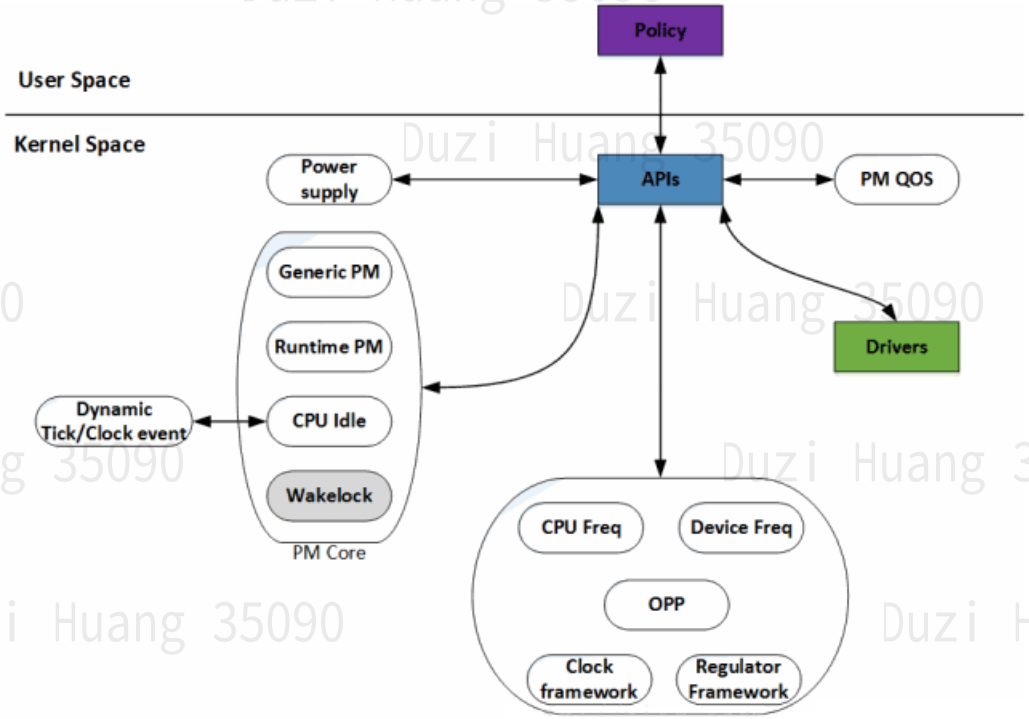
- Linux Thermal 框架
- 高通8350平台热管理概论
- Linux电源管理 ←
- Android电源管理基础

电源管理系统架构

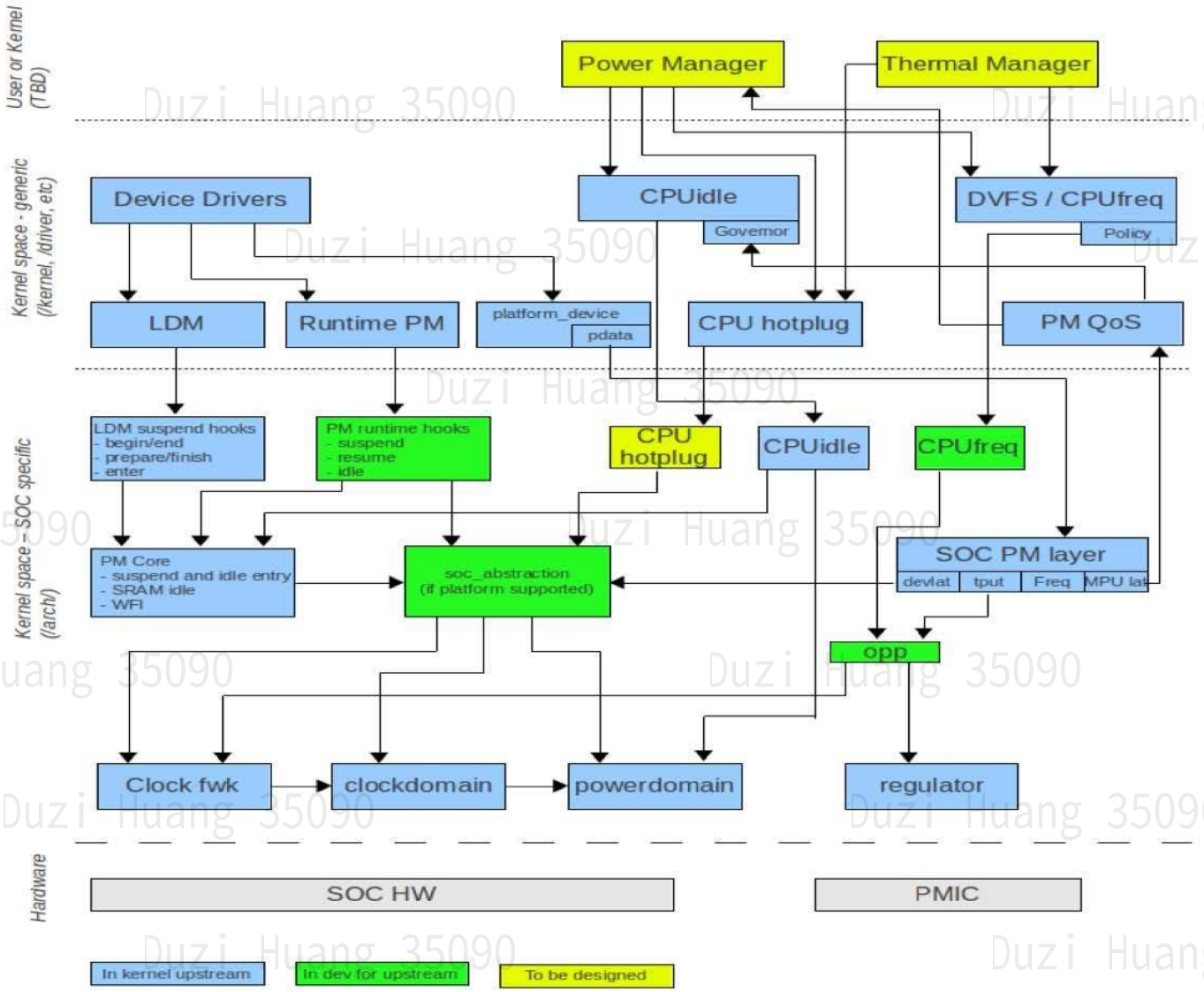
概述

Linux 电源管理非常复杂，牵扯到系统级的待机、频率电压变换、系统空闲时的处理以及每个设备驱动对于系统待机的支持和每个设备的运行时电源管理，可以说和系统中的每个设备驱动都息息相关。对于消费电子产品来说，电源管理相当重要。因此，这部分工作往往在开发周期中占据相当大的比重，下图大体可以归纳为如下几类：

- 1. CPU 在运行时根据系统负载进行动态电压和频率变换的CPUFreq
- 2. CPU 在系统空闲时根据空闲的情况进行低功耗模式的CPUIdle
- 3. 多核系统下CPU 的热插拔支持
- 4. 系统和设备对于延迟的特别需求而提出申请的PMQoS，它会作用于CPUIdle 的具体策略
- 5. 设备驱动针对系统Suspend to RAM/Disk 的一系列入口函数
- 6. SoC 进入suspend 状态、SDRAM 自刷新的入口
- 7. 设备的runtime（运行时）动态电源管理，根据使用情况动态开关设备
- 8. 底层的时钟、稳压器、频率/电压表（OPP 模块完成）支撑，各驱动子系统都可能用到



系统架构


User or Kernel (TBD)
Kernel space - generic (/kernel, /driver, etc)
Kernel space - SOC specific (/arch/)
Hardware
Power Manager
Thermal Manager
Device Drivers
CPUidle
Governor
DVFS / CPUfreq
Policy
LDM
Runtime PM
platform_device
pdata
CPU hotplug
PM QoS
LDM suspend hooks
- begin/end
- prepare/finish
- enter
PM runtime hooks
- suspend
- resume
- idle
CPU hotplug
CPUidle
CPUfreq
PM Core
- suspend and idle entry
- SRAM idle
- WFI
soc_abstraction
(if platform supported)
SOC PM layer
devlat
tput
Freq
MPU lat
opp
Clock fwk
clockdomain
powerdomain
regulator
SOC HW
PMIC
In kernel upstream
In dev for upstream
To be designed

主要技术

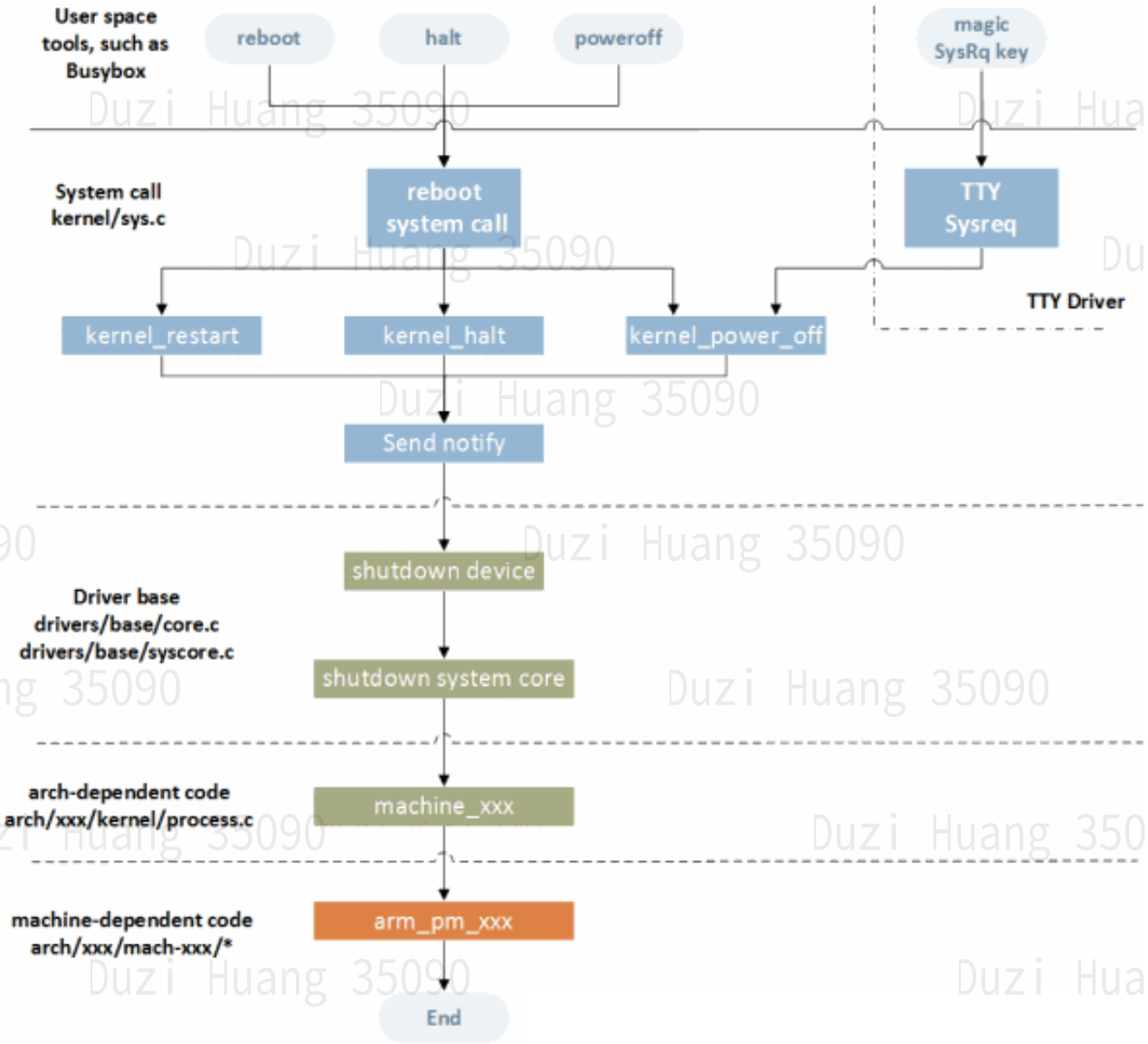
Linux电源管理主要使用的技术包括:

- CPUFreq：也叫DVFS（Dynamic voltage and frequency scaling），即动态电压频率调整。在系统运行时根据系统负载动态调节。
- DEVFreq: CPUFreq只针对CPU做动态电压频率调节。devfreq可以对设备，如DRAM,GPU等做动态电压频率调节。
- CPUIdle: CPU在系统空闲时根据空闲的情况进行低功耗模式。
- CPUHotplug: 多核系统下CPU的热插拔支持。
- PM QOS: 系统和设备对于延迟的特别需求而提出申请的PM QoS，它会作用于CPUIdle的具体策略.
- SUSPEND: 设备驱动针对系统Suspend to RAM/Disk的一系列入口函数。
- RUNTIME PM: 设备的runtime（运行时）动态电源管理，根据使用情况动态开关设备。
- Regulator：用于调节CPU等模块的电压和电流值。
- OPP: 可以使SOCs或者Devices正常工作的电压和频率组合。内核提供这一个Layer，是为了在众多的电压和频率组合中，筛选出一些相对固定的组合，从而使事情变得更为简单一些。
- Thermal: 温控管理。

电源管理相关源码主要分布于:
kernel/power/ *
drivers/power/
drivers/base/power/*
drivers/cpuidle/*
drivers/cpufreq/*
drivers/devfreq/*
include/linux/power_supply.h
include/linux/cpuidle.h
include/linux/cpufreq.h
include/linux/cpu_pm.h
include/linux/device.h
include/linux/pm.h
include/linux/pm domain.h
include/linux/pm runtime.h
include/linux/pm wakeup.h
include/linux/suspend.h
Documentation/power/*.txt

系统reboot/shut down过程

reboot/shut down流程



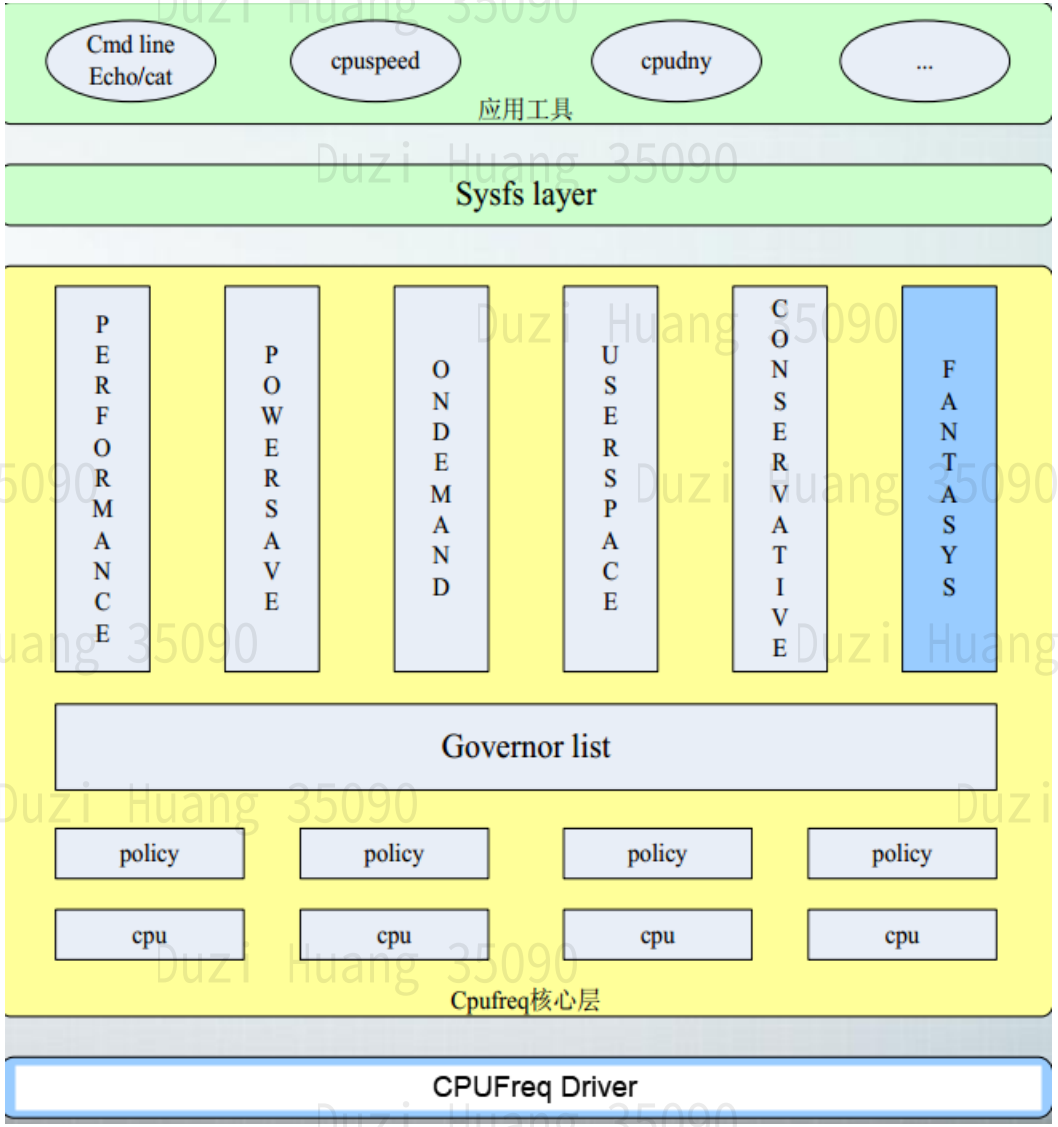
CPUFreq

简介

- CPUFreq是一种实时的电压和频率调节技术，也叫DVFS（Dynamic Voltage and Frequency Scaling）动态电压频率调节。
- 为何需要DVFS
  - 随着技术的发展，CPU的频率越来越高，性能越来越好，芯片制造工艺也越来越先进。但高性能的同时也带来高发热。其实移动嵌入式设备并不需要时刻保持高性能。因此，需要一种机制，实现动态地调节频率和电压，以实现性能和功耗的平衡。

CPUFreq软件框架

- 和一般的linux子系统类似，CPUFreq采用了机制与策略分离的设计架构。分为三个模块：
  - cpufreq core： 对cpufreq governors和cpufreq drivers进行了封装和抽象并定义了清晰的接口，从而在设计上完成了对机制和策略的分离。
  - cpufreq drivers：位于cpucore的底层，用于设置具体cpu硬件的频率。通过cpufreq driver可以使cpu频率得到调整。cpufreq driver借助Linux Cpufreq标准子系统中的cpufreq_driver结构体，完成cpu调频驱动的注册及实现。
  - cpufreq governor：位于cpucore的上层，用于CPU升降频检测，根据系统和负载，决定cpu频率要调节到多少。cpufreq governor借助于linux cpufreq子系统中cpufreq_governor结构体，完成了cpu调频策略的注册和实现。



实现原理&接口

- linux cpufreq通过向系统注册实现cpufreq driver和cpufreq governor。cpu governor实现调频的策略，cpu driver实现调频的实际操作，从而完成动态调节频率和电压。一般情况下，优先调节频率，频率无法满足，再调节电压以实现调频。
- cpufreq相关的节点位于/sys/devices/system/cpu/cpu0/cpufreq目录下

```
shell@tiny4412:/sys/devices/system/cpu/cpu0/cpufreq # ls
affected_cpus
cpuinfo_cur_freq
cpuinfo_max_freq
cpuinfo_min_freq
cpuinfo_transition_latency
related_cpus
scaling_available_governors
scaling_cur_freq
scaling_driver
scaling_governor
scaling_max_freq
scaling_min_freq
scaling_setspeed
stats
```

节点	cat	echo
affected_cpus	需要软件协调频率的cpu列表	×
related_cpus	需要软件或者硬件来协调频率的cpu列表	×
cpuinfo_max_freq	cpu能够运行的最高频率（硬件读取，KHz）	×
cpuinfo_min_freq	cpu能够运行的最低频率（硬件读取，KHz）	×
cpuinfo_cur_freq	cpu当前频率（硬件读取，KHz）	×
cpuinfo_transition_latency	cpu在不同频率之间切换所需要的时间（ns）	×
scaling_available_governors	当前内核中支持的所有cpufreq governor	×
scaling_driver	cpufreq控制的硬件驱动	×
scaling_cur_freq	cpu工作频率（软件读取，内核认定，KHz）	×
scaling_max_freq	当前policy的上限（KHz）	改变上限
scaling_min_freq	当前policy的下限（KHz）	改变下限
scaling_governor	cpu调频策略的名称	改变策略
scaling_setspeed	cpu运行频率（仅userspace策略有效）	设置频率

目录

ACPI简介

- ACPI（高级配置与电源接口），ACPI是一种规范（包含软件与硬件），用来供操作系统应用程序管理所有电源接口。
  - ACPI将计算机系统的状态划分为四个全局状态（G0-G3），共7个状态，其中G0对应S0，G1将低功耗状态细分为四个状态，对应S1-S4，G2、G3代表关机状态分别对应S5、S6。

ACPI State	Description
S0	正常工作状态
S1	CPU与RAM供电正常，但CPU不执行指令
S2	比S1更深的一个睡眠层次，这种模式通常不采用
S3	挂起到内存
S4	挂起到硬盘
S5	Soft Off，CPU、外设等断电，但电源依旧会为部分极低耗设备供电
S6	Mechanical Off，全部断电

Linux系统电源状态

在Linux操作系统中，将电源划分为如下几个状态：

ACPI State	Linux State	Description
S0	On(on)	Working
S1	Standby(standby)	CPU and RAM are powered but not executed
S2	------	------
S3	Suspend to RAM (mem)	CPU is Off,RAM is powered and the running content is saved to RAM
S4	Suspend to Disk(disk)	All content is saved to Disk and power down
S5	Shutdown	Shutdown the system

STR(Suspend to RAM)

- 挂起到内存，俗称待机、睡眠（Sleep），进入该状态，系统的主要工作如下：
  - 1、将系统当前的运行状态等数据保存在内存中，此时仍需要向RAM供电，以保证后续快速恢复至工作状态
  - 2、冻结用户态的进程和内核态的任务（进入内核态的进程或内核自己的task）
  - 3、关闭外围设备，如显示屏、鼠标等,中断唤醒外设不会关闭，如电源键
  - 4、CPU停止工作
- Standby也属于睡眠的一种方式，属于浅睡眠。该模式下CPU并未断电，依旧可以接收处理某些特定事件，视具体设备而定，恢复至正常工作状态的速度也比STR更快，但也更为耗电。举个例子来说，以该方式进入睡眠时，后续通过点击键盘也能将系统唤醒。而以mem进入的睡眠为深度睡眠，只能通过中断唤醒设备唤醒系统，如电源键(此时按电源键，不会经过正常的开机流程的BIOS、BOOTLOAD等)，此时按键盘是无法唤醒系统的。

STD(Suspend to Disk)

- 挂起到硬盘，俗称休眠（Hibernation）将系统当前的运行状态等数据保存到硬盘上，并自动关机。下次开机时便从硬盘上读取之前保存的数据，恢复到休眠关机之前的状态。譬如在休眠关机时，桌面打开了一个应用，那么下一次开机启动时，该应用也处于打开状态。而正常的关机-开机流程，该应用是不会打开的。
- Linux内核代码声明如右,位于kernel/power/suspend.c
- 在新版内核中，进程freeze的功能被单独抽离出来作为一个电源状态，该状态仅仅是冻结进程，并不会使系统进入低功耗状态（如切断CPU时钟源、关闭外设供电等）。
  相关宏定义位于：linux/include/linux/suspend.h
- 其中状态4就是STD，所谓的休眠状态（Hibernation）

```
35  const char * const pm_labels[] = {
36          [PM_SUSPEND_FREEZE] = "freeze",
37          [PM_SUSPEND_STANDBY] = "standby",
38          [PM_SUSPEND_MEM] = "mem",
39  };
40  const char *pm_states[PM_SUSPEND_MAX];
41  static const char * const mem_sleep_labels[] = {
42          [PM_SUSPEND_FREEZE] = "s2idle",
43          [PM_SUSPEND_STANDBY] = "shallow",
44          [PM_SUSPEND_MEM] = "deep",
45  };
```

```
35  #define PM_SUSPEND_ON           ((__force suspend_state_t) 0)
36  #define PM_SUSPEND_FREEZE       ((__force suspend_state_t) 1)
37  #define PM_SUSPEND_STANDBY      ((__force suspend_state_t) 2)
38  #define PM_SUSPEND_MEM          ((__force suspend_state_t) 3)
39  #define PM_SUSPEND_MIN          PM_SUSPEND_FREEZE
40  #define PM_SUSPEND_MAX          ((__force suspend_state_t) 4)
```

查看Android支持的电源模式

```
graymonkey@graymonkey-RESCUER-R720-15IKBN:~$ adb shell
root@generic_x86_64:/ # su
root@generic_x86_64:/ # cat /sys/power/state
freeze mem
root@generic_x86_64:/ #
```

- Android手机是不支持休眠模式的，休眠模式需要一块与RAM大小一致存储空间。

Idle State: CPU Idle

- Linux系统运行的基础是基于进程调度，实际上内核调度的线程（task），内核并不会区分线程与进程，都将他们当做一个线程（task）来处理；当所有的进程都没事儿干的时候，系统就会启用idle进程，使系统进入低功耗状态（如关闭一些服务、模块功能，降低CPU工作频率等），即idle状态，以达到省电的目的。idle状态又可以划分为不同的层级，以MTK的芯片为例，通常划分为以下几个状态:

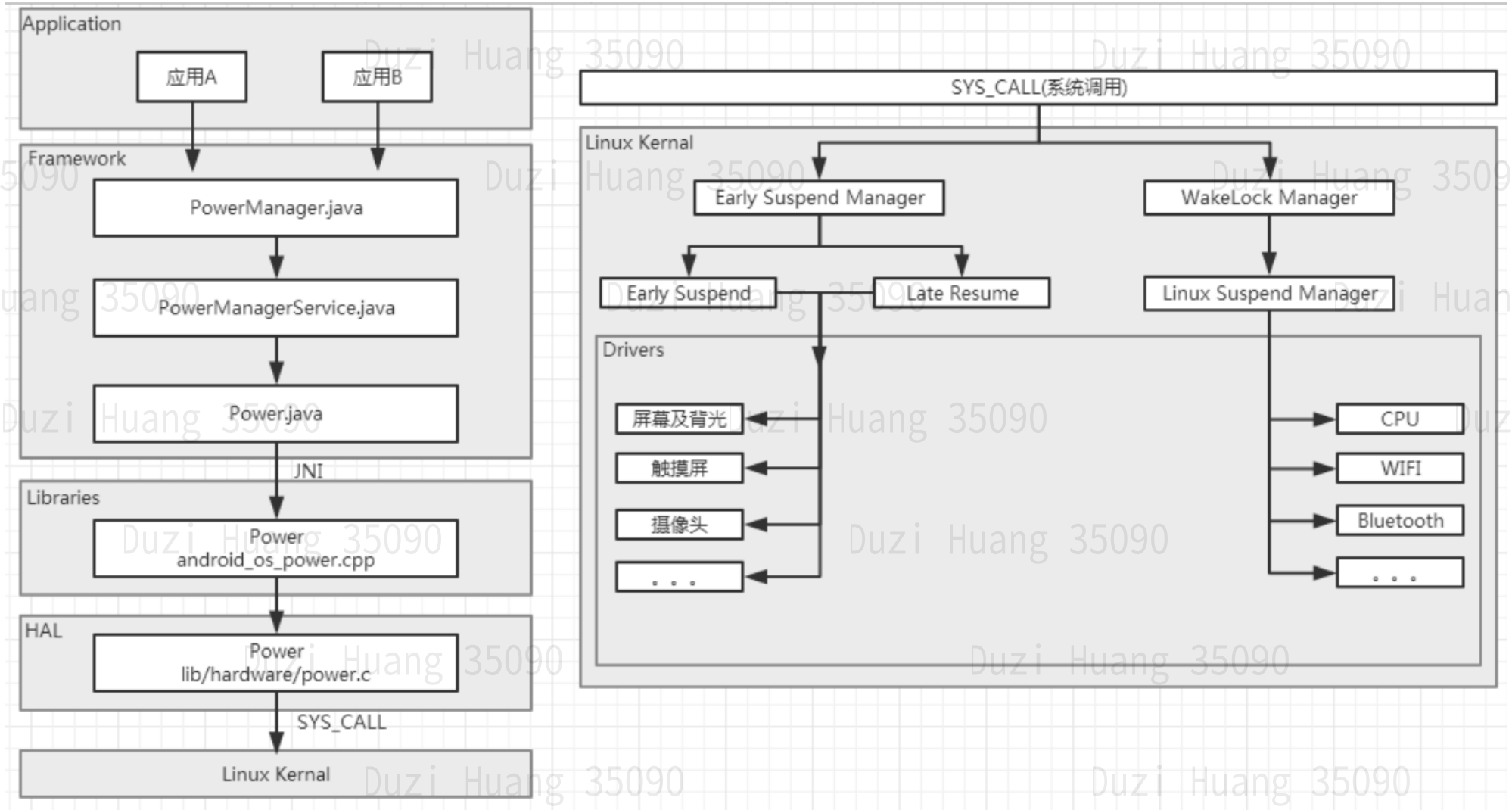
状态	描述
soidle（screen on idle)	亮屏 Idle 模式，该模式下与正常工作状态差别不大，唯一的区别就cpu处于空闲状态
rgidle	浅度 Idle 模式，cpu处于 WFI（wait for interrupt），屏幕熄灭，同时关闭一些不需要的服务及模块，注意此状态cpu的时钟源与RTC模块是工作正常的，此时是可以通过TimerTask的定时触发激活系统的，TimerTask依赖于CPU的RTC模块，而Alarm则依赖于PMIC的RTC模块
dpidle(deep idle)	深度idle模式，该模式下cpu的时钟源和hrtimer（高精度定时器模块（RTC））被关闭，所有进程（包括系统进程）被冻结，即进入上文所述的睡眠状态

Idle State: Device Idle

- Device Idle属于Doze模式中概念，即指当手机屏幕熄屏、不充电、静置不动。
- Doze模式中的idle状态在概念属于浅idle状态，只是关闭了一些特定服务和模块，并非立即进入睡眠，当然这个过程当中依旧有可能满足睡眠条件而进入睡眠状态，至于如何进入请参考下文【睡眠触发入口】一节。

Doze模式的限制
网络接入被暂停
系统忽略wake locks
标准的AlarmManager alarms（包括setExact()和setWindow()）被延缓到下一个maintenance window
如果你需要在Doze状态下启动设置的alarms，使用setAndAllowWhileIdle()或者setExactAndAllowWhileIdle()。当有setAlarmClock()的alarms启动时，系统会短暂退出Doze模式
系统不会扫描Wi-Fi
系统不允许sync adapters运行
系统不允许JobScheduler运行

Android电源管理框架



WakeLock

- 唤醒锁，一种锁机制，用于阻止系统进入睡眠状态，只要有应用获取到该锁，那么系统就无法进入睡眠状态。该机制起初是早期Android为Linux内核打得一个补丁，并想合入到linux内核，但被Linux社区拒绝，后续Linux内核引入自己的Wakelock机制,Android系统也使用的是linux的Wakelock机制，所以该机制并非Android特有的机制。Android系统提供了两种类型的锁，每一个类型又可分为超时锁与普通锁，超时锁，超时会自动释放，而普通锁则必需要手动释放。

类型	描述
WAKE_LOCK_SUSPEND	阻止系统进入睡眠状态（STR）
WAKE_LOCK_IDLE	阻止系统从idle进程进入那些具有较大中断时延、禁用了较多中断源的低功耗状态（睡眠除外），持有该类型的锁，不影响系统进入睡眠状态。自Android API-17（对应android linux内核版本3.4）移除了该类型的唤醒锁。

应用层提供的锁类型如下，这些锁都需要手动释放：

FLAG	CPU	屏幕	键盘
PARTIAL_WAKE_LOCK	开启	关闭	关闭
SCREEN_DIM_WAKE_LOCK	开启	变暗	关闭
SCREEN_BRIGHT_WAKE_LOCK	开启	变亮	关闭
FULL_WAKE_LOCK	开启	变亮	变亮

锁的释放

- Linux3.4内核中摒弃了之前的wakelock机制，引入wakeup source机制来进行睡眠管理，为了保证上层接口不变，Android的Linux内核便将wakeup source包装成wakelock。
- 当我们应用层释放锁之后，它并不会马上消失。wakelock分为激活和非激活状态，非激活状态300S之内，无人在申请wakelock，那么它将从红黑二叉树，LRU链表当中删除，如此便可复用锁，节省系统开销。

预挂起&迟唤醒

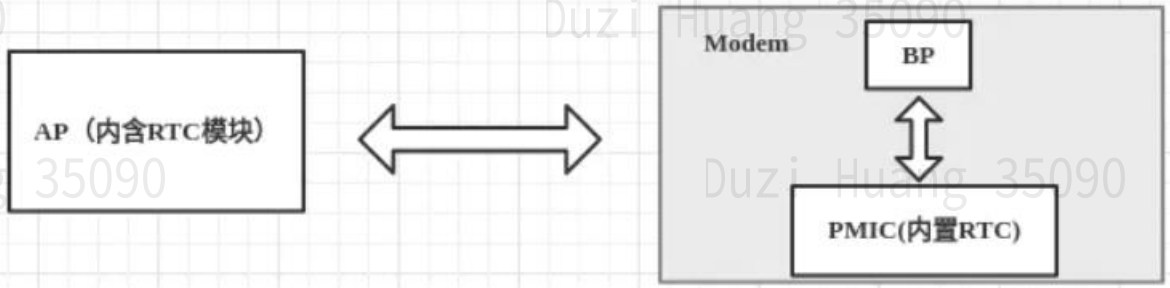
- Early Suspend
  - 预挂起机制是Android特有的挂起机制，这个机制作用是关闭一些与显示相关的外设，比如LCD背光、重力感应器、触摸屏，但是其他外设如WIFI、蓝牙等模块等并未关闭。此时，系统依旧可以处理事件，如音乐播放软件，息屏后依旧能播放音乐。
  - 需要注意的是Early Suspend机制与WakeLock机制相互独立，就算有应用持有wakelock锁，系统依旧可以通过Early Suspend机制关闭与显示相关的外设。
  - 注意：Android 4.4起，也就是引入ART的版本，摒弃了early suspend机制，改用了fb event通知机制，即后续版本只有suspend、resume以及runtime suspend、runtime resume。
- Late Resume迟唤醒机制，用于唤醒预挂起的设备。

睡眠触发入口

- 在wakelock中，有3个地方可以让系统从early_suspend进入suspend状态。
  - wake_unlock，系统每释放一个锁，就会检查是否还存其他激活的wakelock，若不存在则执行Linux的标准suspend流程进入睡眠状态
  - 在超时锁的超时回调函数，判断是否存在其他激活的wakelock，若不存在，则进入睡眠状态
  - autosleep机制，android 4.1引入该机制，亮屏时会向autosleep节点写入off，熄屏则会写入mem。Android一灭屏，就会尝试进入睡眠，失败之后系统处于idle进程超过一定时间，则又尝试进入睡眠，判断标准同上，若存在wakelock则进入失败

手机来电与Alarm为何能唤醒系统

- Android在硬件架构上将处理器分为二类：Application Processor（AP）和Baseband Processor（BP），AP是ARM架构的处理器，用于运行Linux+Android系统，耗电量高；BP用于运行实时操作系统（RTOS），用于处理手机通信，耗电量低。
- 当AP进入睡眠，有来电时，Modem（调制解调器）将唤醒AP；而我们平时所用的Alarm在硬件上则是依赖PMIC（电源管理芯片）中的RTC模块，所以即使AP断电进入睡眠，我们定的闹钟依旧会生效。



THANKS