# 重要文件

## 1、.config

路径：alps/out/target/product/k37tv1\_bsp/obj/KERNEL\_OBJ/

作用：里面存放了整个工程的编译选项，由Kconfig和 xx\_defconfig 一起生成 .config

## 2、xx\_defconfig

路径：alps\kernel-4.4\arch\arm\configs\

作用：里面存放一些注定一些编译选项是否需要开启，最终生成 .config

## 3、ProjectConfig.mk

路径：alps\device\mediateksample\k39tv1\_bsp\_1g

作用：这个是原先MTK提供给我们的配置文件，我们的Config.mk只是从中拉出一些我们的外设出来进行配置：其他一些共用的东西就使用ProjectConfig.mk中的。

## 4、zyt.h

路径：kernel-4.4\drivers\misc\mediatek\include\mt-plat\mt6739\include\mach\zyt.h

作用：开一些宏定义

使用：使用这个文件里面的宏的时候需要加上头文件 #include “mach\zyt.h”

# 按键

## 1、按key power键灭屏从log中怎么找出相应的位置

从kernel\_log…中找到关键字：

灭屏：

<6>[ 588.485133]

<6>[ 588.604037] (0)[1031:system\_server]PM: suspend entry 2010-01-01 00:18:13.542257806 UTC

<6>[ 588.604086] (3)[1086:AlarmManager]PM: Syncing filesystems ...

# 屏的相关知识

**路径：**

**（1）、内核：alps\kernel-4.4\drivers\misc\mediatek\lcm**

**（2）、lk：alps\vendor\mediatek\proprietary\bootable\bootloader\lk\dev\lcm**

## 调试过程中发现uboot没有logo或者显示花屏kernel正常

### 平台：MT6580

分析：可能是由于log的分辨率和屏的分辨率对应不上，目前有这几种分辨率wvga: 480X800; fwvga: 480x854; qhd 540x960 ; hd 720x1280；所以应该在 Config.mk中配置当前分辨率的log如 ZMK\_BOOT\_LOGO\_FOLDER=wvga ；然后编译source zmkpl 与source zmklk （执行source zmklk的时候会打包 log为log.bin）。

Log所在的路径： alps/vendor/mediatek/proprietary/bootable/bootloader/lk/dev/logo （这里面的log是一整套，系统会根据你当前的状态显示不同的log。）

## 2、显示状态下按power key正常关机 没有调用lcm\_suspend函数

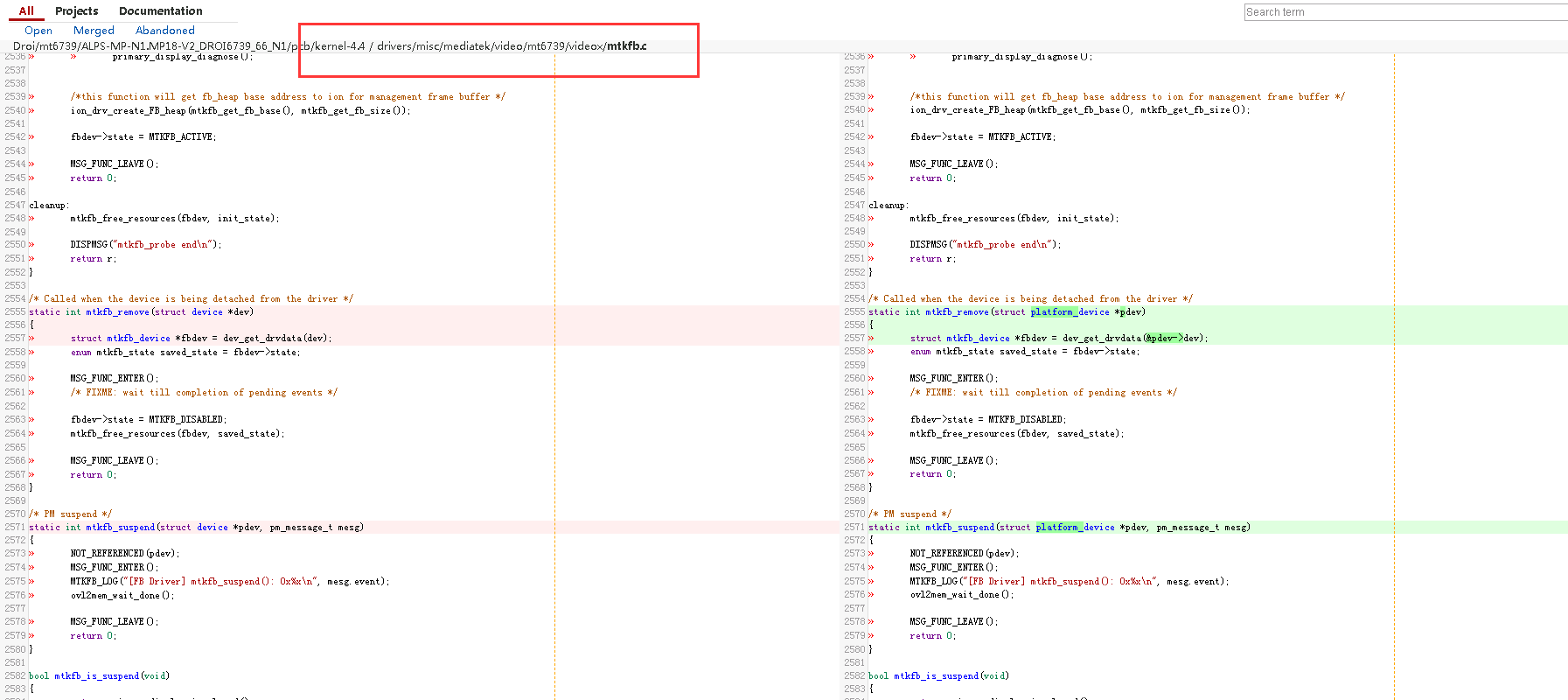
### 平台：MT6739

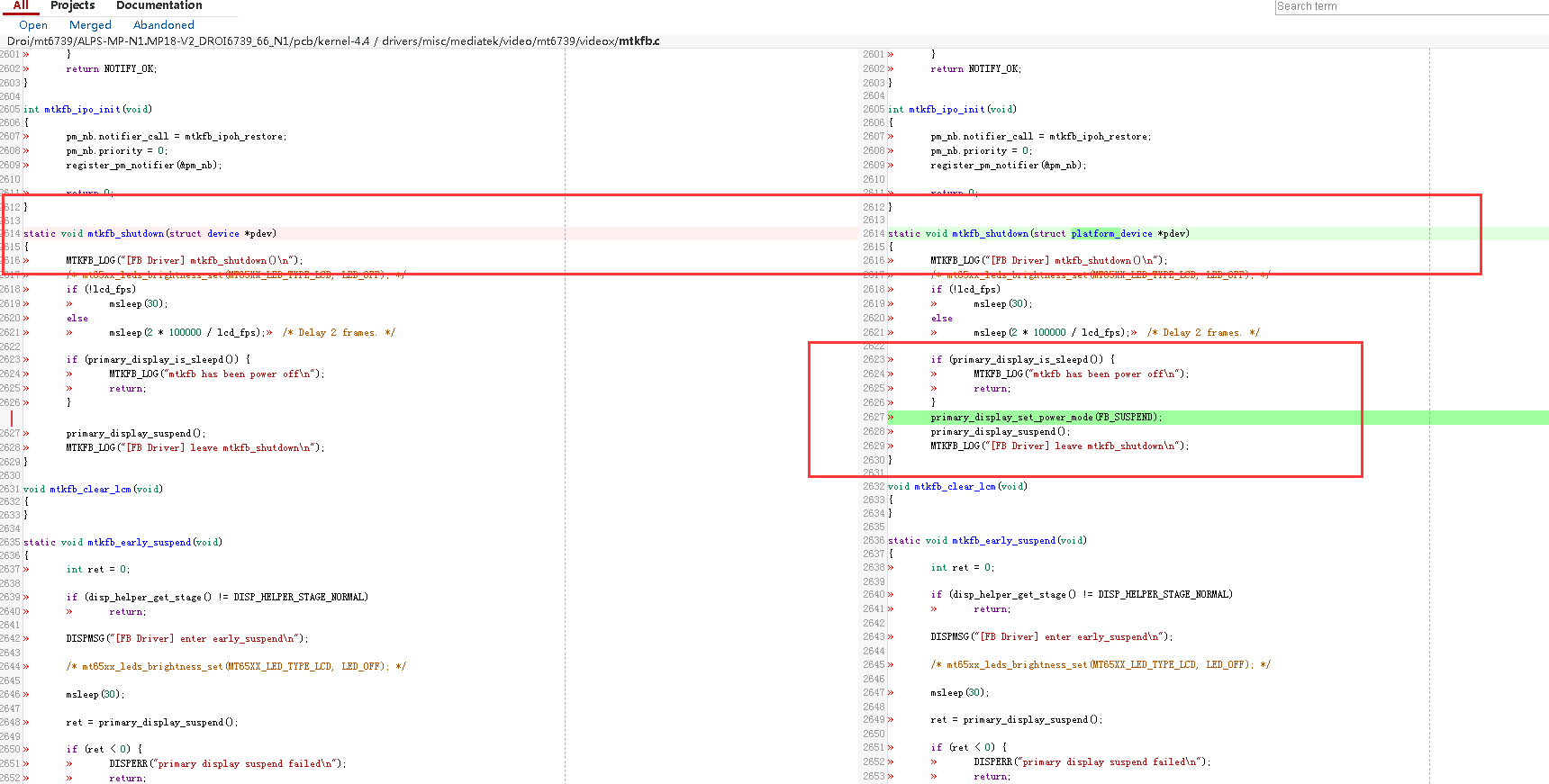
现象：

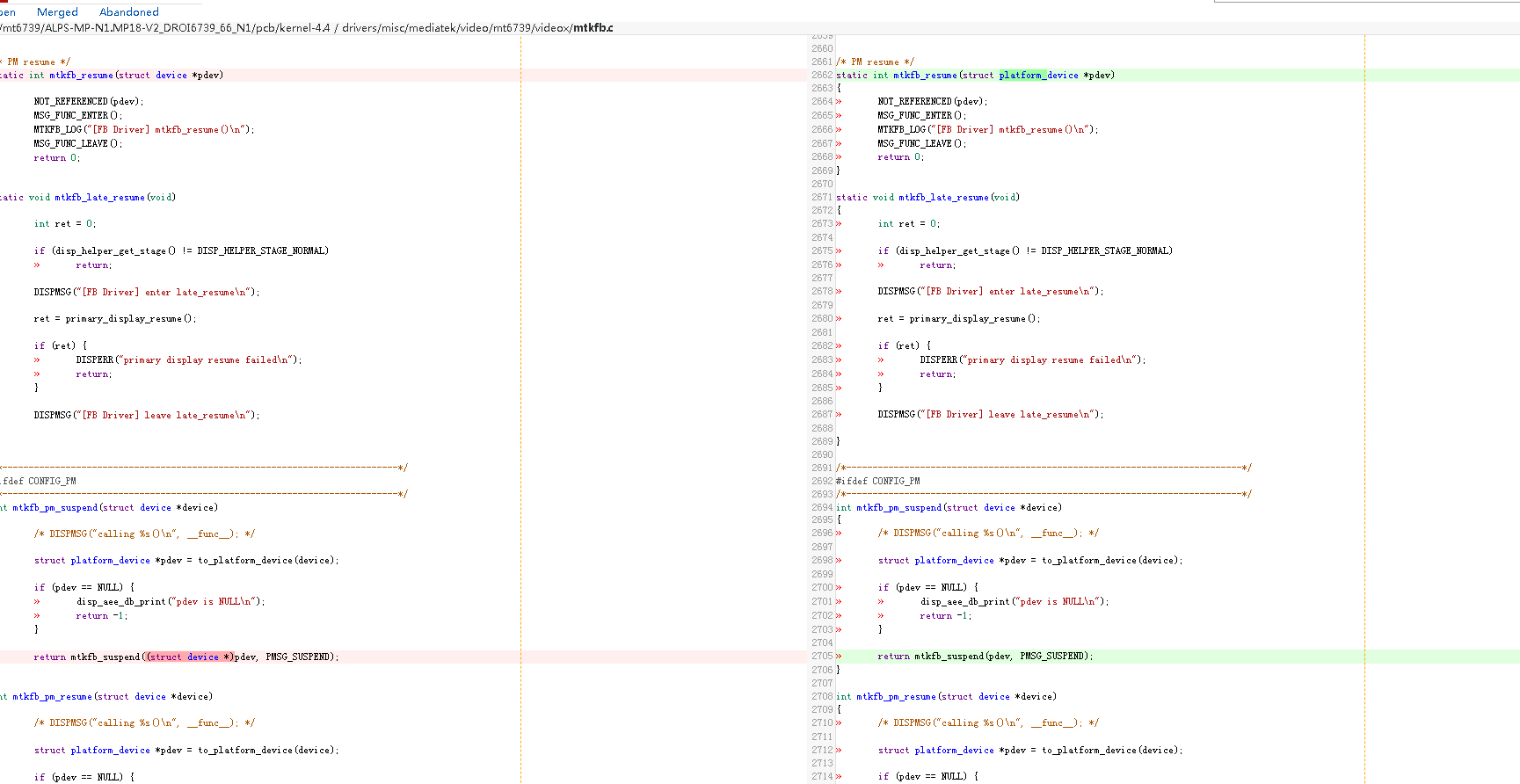
1 显示状态下按power key正常关机 没有调用lcm\_suspend函数，现在整机正常关机是做重启的准备-------必需要调用28 10（sleep\_in）

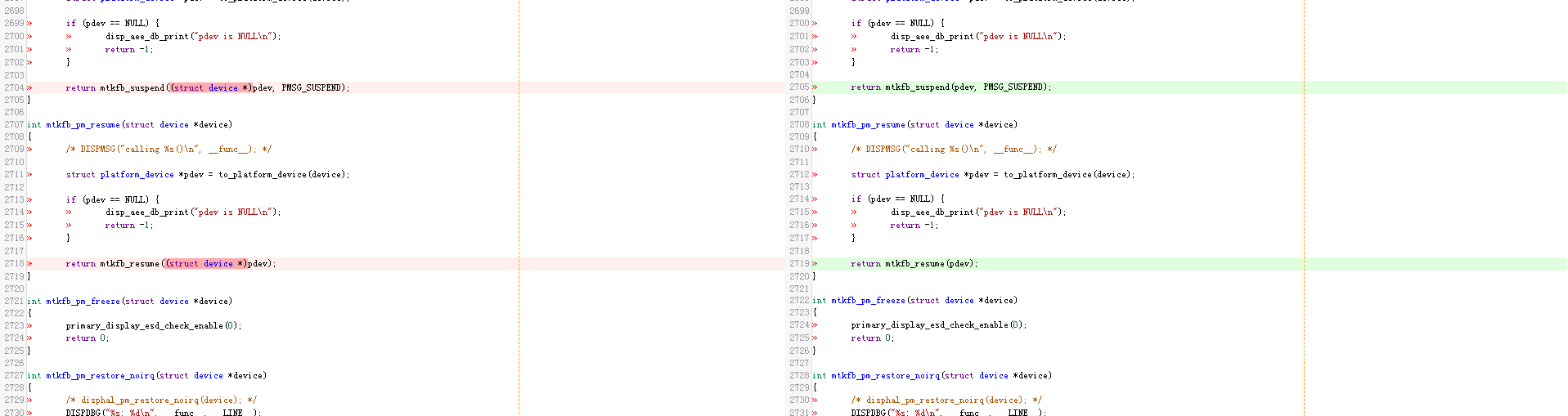
2 关机状态插USB显示充电画面后拔USB ，没有调用28 10（sleep\_in） .-------必需要调用28 10（sleep\_in）

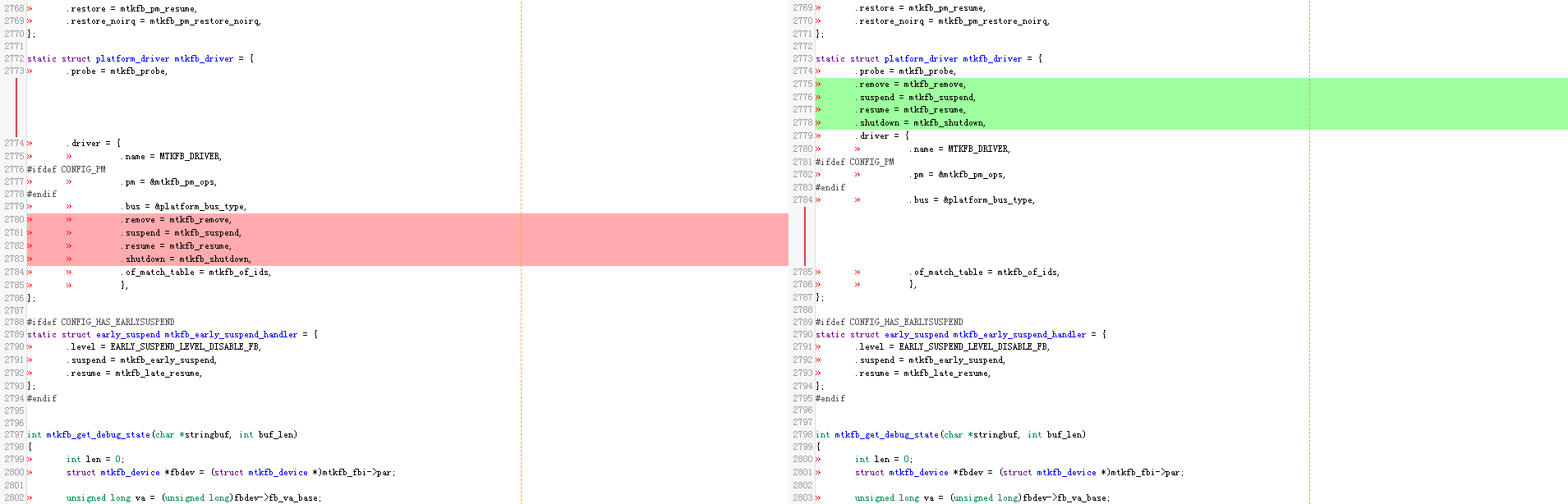
解决方法：











## 3、计算DSI数据速率的方式，以及如何配置时钟clk的方式

### 平台：MT6739 MT6580 MT6737

1、DSI vdo mode下的数据速率data\_rate的大致计算公式为：

**Data rate= (Height+VSA+VBP+VFP)\*(Width+HSA+HBP+HFP)\* total\_bit\_per\_pixel\*frame\_per\_second/total\_lane\_num\2上下边缘**

2、DSI cmd mode下的数据速率data\_rate的大致计算公式为：

Data rate= width\*height\*1.2\* total\_bit\_per\_pixel\*frame\_per\_second/total\_lane\_num

参数注释：

data\_rate ： 表示的是数据速率

width，height ：屏幕分辨率

VSA VBP VFP ：DSI vdo mode的vertical porch配置参数

HSA HBP HFP ：DSI vdo mode的horizontal porch配置参数

total\_bit\_per\_pixel ：表示的是一个pixel需要用几个bit来表示，比如RGB565的话就是16个bit

frame\_per\_second ：就是我们通常看到的fps，叫做帧率，表示每秒发送多少个帧，一般是60帧每秒

total\_lane\_num ：表示的是data lane的对数。

3、DSI采用的是双边采样，则clk等于数据速率的一半，因此： clk=data\_rate/2

有两种配置clk的方式，第一种方式配置四个参数得到，第二种配置方式直接配置频率，建议采用第二种。

第一种方式，通过div分频倍频实现，各个平台略有差异，但是原理基本一致，请参考porting guide，如下举例89平台:

params->dsi.pll\_div1 = ; //配置范围为0,1,2,3的时候，对应的div1\_real等于1，2，4,4

params->dsi.pll\_div2 = ; //配置范围为0,1,2,3的时候，对应的div2\_real等于1，2，4,4

params->dsi.fbk\_div = ; //范围 0..63

params->dsi.fbk\_sel = ; //配置范围为0,1,2,3的时候，对应的fbk\_sel\_real等于1，2，4,4

输出频率=26MHz\*（fbk\_div+1）\*(2\*fbk\_sel\_real)/(div1\_real\*div2\_real)

第二种方式，直接配置clk大小：

params->dsi.PLL\_CLOCK = LCM\_DSI\_6589\_PLL\_CLOCK\_234;//这里举例89平台，使用一个宏，表示配置的clk等于234MHz。但是在89之后的平台，使用直接配置一个频率数字的方式，比如params->dsi.PLL\_CLOCK = 234，表示234MHZ）

4、在lcm porting过程中，这些参数都定义在lcm\_drv.h文件中的LCM\_DSI\_PARAMS结构体中，随着平台的发展，或许有所不同，但是基本原理都是一致的，如何配置clk的大小，请先根据自己的帧率、像素格式、porch值、屏的分辨率、data lane对数等计算出data\_rate，然后计算出clk。

## 4、烧入正确的初始化代码还是白屏

### 平台：MT6739 MT6580 MT6737

原因：

1、经过查看硬件电路图发现支持4通道。

2、所以当时软件就配置为4通道（平台的4通道）。

3、实际客户给的初始化代码是两通道的，这导致平台和屏的对应不上来，进而也就无法使用了。

## 5、录制屏幕 （screenrecord）

（1）正常开机显示正常画面后输入：adb shell "screenrecord /sdcard/video.mp4

（2）多次按power键亮灭屏复现问题。

（3）按“ctrl + c”键停止录制

（4）adb pull sdcard/video.mp4 D:\

# TP的相关知识

## 1、出现分辨率不对的现象

（1）、固件分辨率不对（APK升级 开机升级）

（2）、配置的X Y 不对（Config.mk配置的宽高要确保有效果）

## 2、两个重要的宏（目前还没有细看）

现象：TP可以动，但会出现一些奇怪的现象比如不能用了或者分辨率不对

CONFIG\_ENABLE\_TYPE\_B\_PROTOCOL

CONFIG\_ENABLE\_REGULATOR\_POWER\_ON

## 3、修改底层打开不了上层创建的system/data文件的权限：

路径：alps\device\mediatek\sepolicy\basic\non\_plat\hal\_bootctl\_default.te

修改：allow hal\_graphics\_composer\_default system\_data\_file:file { read open };

## 4、TP手势开关

文件路径：/data/system/gesture\_switch

背景：TP手势开关和上层的通讯是通过判断一个文件是否存在来判断手势开关是打开还是关闭的。

现象：当上层（java）创建一个文件的时候，底层（kernel）去访问这个文件的时候没有权限导致读失败。

解决方法：

1. 通过按power键让程序跑读取文件的代码，同时抓取上层log 使用命令 adb shll logcat > 1.txt
2. 从log中搜索关键字 “denied”
3. 很容易找到手势相关的log:

01-01 12:19:23.270 255 255 W composer@2.1-se: type=1400 audit(0.0:1383): avc: denied { read } for name="gesture\_switch" dev="dm-0" ino=6010 scontext=u:r:hal\_graphics\_composer\_default:s0 tcontext=u:object\_r:system\_data\_file:s0 tclass=file permissive=0

分析：

（1）、denied { read } for name="gesture\_switch" 读文件 gesture\_switch 没有权限。

（2）、scontext=u:r:hal\_graphics\_composer\_default:s0 tcontext=u:object\_r:system\_data\_file:s0 tclass=file permissive=0 需要在 hal\_graphics\_composer\_default.\*（这个文件的后缀暂时还不知道怎么来的这里是.te）文件里面添加 allow hal\_graphics\_composer\_default system\_data\_file:file { read open };

4、按第三步添加相应的权限既可以正常访问。

# 相机的相关知识

## 如何移植一个新的摄像头代码

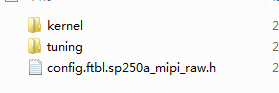
### 平台：MT6580 MT6739 MT6737

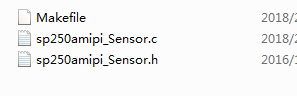
注意：摄像头的id匹配情况的log可以通过 kernel\_log.boot 文件看出（关键字：get\_imgsensor\_id）。

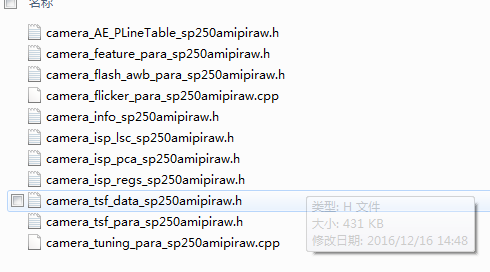
1）、厂家一般会提供一份代码。

2、需要修改的地方：

a、在zlib/zcamera 的路径下面新建一个文件夹，命名如 sp250a\_mipi\_raw\_zh286\_jm 里面的目录结构如下：



在kernel 目录下有文件

在tuning目录下有文件

b、config.ftbl.sp250a\_mipi\_raw.h文件是一个差值文件，有时候厂家没有提供，只要从相同像素的其他项目考过来就好。

c、需要修改的文件

**Makefile（右边是正确）：**



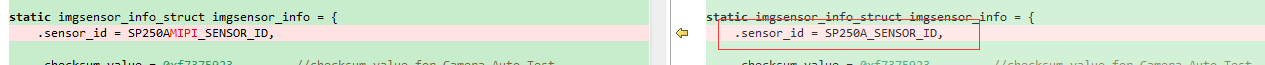
**sp250amipi\_Sensor.c**

关闭我们没有的头文件：





修改标准的命名接口：





camera\_flicker\_para\_sp250amipiraw.cpp

关闭我们没有的头文件：



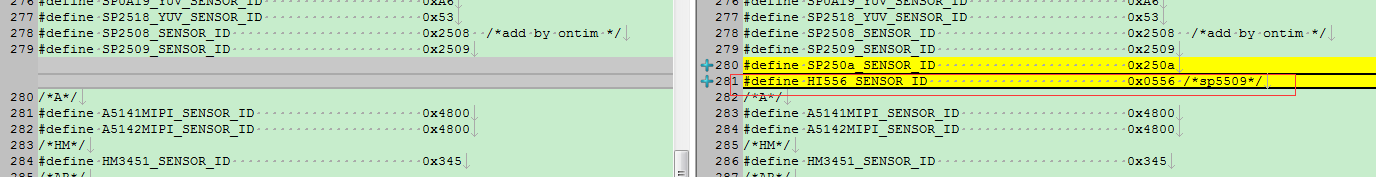


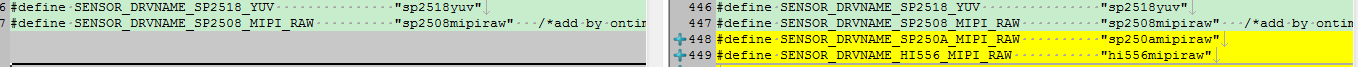
修改标准的命名接口：camera\_info\_ov8858mipiraw.h



d、需要增加相应的ID

路径：alps\kernel-3.18\drivers\misc\mediatek\imgsensor\inc\ kd\_imgsensor.h





e、编译 source zmk 下载 system.img

f：需要配置上电时序，文件路径在：alps\kernel-4.4\drivers\misc\mediatek\imgsensor\src\mt6739\camera\_hw\imgsensor\_cfg\_table.c 例如如下配置：

#if defined(S5K4H8\_MIPI\_RAW) //这个宏在project.mk中

{

SENSOR\_DRVNAME\_S5K4H8\_MIPI\_RAW,

{

{SensorMCLK, Vol\_High, 0},

{PDN, Vol\_Low, 0},

{RST, Vol\_Low, 0},

{DOVDD, Vol\_1800, 0},

{AVDD, Vol\_2800, 0},

{DVDD, Vol\_1200, 0},

{AFVDD, Vol\_2800, 1},

{PDN, Vol\_High, 0},

{RST, Vol\_High, 0}

},

},

#endif

g: 文件：xxx\_senso\_raw.c修改方向：IMAGE\_NORMAL 修改：颜色SENSOR\_OUTPUT\_FORMAT\_RAW\_Gb

## 2、闪光灯的具体流程

### 平台：MT6580

2.1、程序路径

前闪驱动路径：alps\kernel-3.18\drivers\misc\mediatek\flashlight\src\mt6580\sub\_strobe.c

后闪驱动路径：alps\kernel-3.18\drivers\misc\mediatek\flashlight\src\mt6580\constant\_flashlight\ leds\_strobe.c

2.2、功能配置

前闪：

（1）、平台默认是没有支持前闪的，如果项目需要支持前闪应该在Config.mk中配置：ZMK\_SUB\_CAMERA\_FLASH\_SUPPORT=yes;

（2）、前闪没有真闪的功能，也就是通过pmic来控制的；如果要配置不同的电流应该在zyt.h中配置：

1）

ZCFG\_H\_SUB\_CAMERA\_FLASH\_USE\_ISNK0 副摄闪光灯使用ISINK0，默认40mA. 都未定义时，默认ISNK 0-1

ZCFG\_H\_SUB\_CAMERA\_FLASH\_USE\_ISNK1 副摄闪光灯使用ISINK1，默认40mA

2）

ZCFG\_H\_SUB\_CAMERA\_ISINK\_REGISTER\_STEP 副摄闪光灯使用ISINK[0-3]四路double使能，需定义值[0-5] 对应电流值[4-8-12-16-20-24]\*4\*2 mA

注意：1）和2）是没有办法同时打开的。同时打开的时候1）跑不到。

后闪：

（1）、平台默认是支持后闪的，如果项目不需要支持后闪的时候可以在Config.mk里面配置ZMK\_MAIN\_CAMERA\_FLASH\_SUPPORT = no,这个时候系统会自动跑到一个空的程序 dummy\_flashlight 里面，上层调用接口的时候才不会报错，而且又不会操作到错误的IO口。

（2）、后闪系统默认是真闪，如果想要配置不同电流的假闪应该在zyt.h中配置相应的宏：

ZCFG\_H\_MAIN\_CAMERA\_FLASH\_USE\_ISNK0 主摄闪光灯使用ISINK0，默认40mA

ZCFG\_H\_MAIN\_CAMERA\_FLASH\_USE\_ISNK1 主摄闪光灯使用ISINK1，默认40mA

ZCFG\_H\_MAIN\_CAMERA\_FLASH\_USE\_ISNK2 主摄闪光灯使用ISINK2，默认40mA

ZCFG\_H\_MAIN\_CAMERA\_FLASH\_USE\_ISNK3 主摄闪光灯使用ISINK3，默认40mA

## 3、相机打开慢放功能【SlowMotion】

[FAQ19355] [VR]如何开启SlowMotion功能

[SOLUTION]

Q:平台是否支持SlowMotion功能？

A:登陆MOL搜索FAQ01338，或者直接在DCC系统里搜索Videospecification，下载VideoSpec文档查看

(2) MTK Video Spec\_SP-> Feature List->VR Feature->SlowMotion Record(SMVR)+VP Speed Adjustment (Note-33)

PS：所有平台共用一份VideoSpec文档，所以可以下载任一平台下的Spec

开启SlowMotion功能只需以下3个条件：

以MT6797平台为例：

1、打开Feature Option

找到$project目录下的ProjectConfig.mk文件

将其中的MTK\_SLOW\_MOTION\_VIDEO\_SUPPORT置为yes

MTK\_SLOW\_MOTION\_VIDEO\_SUPPORT = yes

2、在feature table中设定类似120fps的capability

feature table的配置请参考：FAQ14389和FAQ05993

代码示例如下：

/vendor/mediatek/proprietary/custom/mt6797/hal/sendepfeature/imx230\_mipi\_raw/config.ftbl.imx230\_mipi\_raw.h

#if 1

// Slow Motion

FTABLE\_CONFIG\_AS\_TYPE\_OF\_DEFAULT\_VALUES(

KEY\_AS\_(MtkCameraParameters::KEY\_HSVR\_SIZE\_FPS),

SCENE\_AS\_DEFAULT\_SCENE(

ITEM\_AS\_DEFAULT\_("1280x720x120"),//该值请根据VideoSpec进行设定

ITEM\_AS\_VALUES\_(

"1280x720x120",

"1280x736x120"

)

),

)

#endif

N之前的做法：

3、MediaProfiles.cpp中有设定支持120fps的profile

该步的设定需要与第2步中的设定对应

MediaProfiles::createMTKSlowMotionCamcorderProfiles(MediaProfiles \*profiles)

case VAL\_CHIP\_NAME\_MT6797:

{

//CAMCORDER\_QUALITY\_MTK\_VGA\_120

MediaProfiles::CamcorderProfile \*VGA120Profile =

createMTKCamcorderProfile(CAMCORDER\_QUALITY\_MTK\_VGA\_120, CAMCORDER\_DAY\_MODE, BACK\_CAMERA);

profiles->mCamcorderProfiles.add(VGA120Profile);

//CAMCORDER\_QUALITY\_MTK\_720P\_120 //与第2步中设定的"1280x720x120"对应

MediaProfiles::CamcorderProfile \*SM720P120Profile =

createMTKCamcorderProfile(CAMCORDER\_QUALITY\_MTK\_720P\_120, CAMCORDER\_DAY\_MODE, BACK\_CAMERA);

profiles->mCamcorderProfiles.add(SM720P120Profile);

//CAMCORDER\_QUALITY\_MTK\_720P\_180

MediaProfiles::CamcorderProfile \*SM720P180Profile =

createMTKCamcorderProfile(CAMCORDER\_QUALITY\_MTK\_720P\_180, CAMCORDER\_DAY\_MODE, BACK\_CAMERA);

profiles->mCamcorderProfiles.add(SM720P180Profile);

//CAMCORDER\_QUALITY\_MTK\_1080P\_120

MediaProfiles::CamcorderProfile \*SM1080P120Profile =

createMTKCamcorderProfile(CAMCORDER\_QUALITY\_MTK\_1080P\_120, CAMCORDER\_DAY\_MODE, BACK\_CAMERA);

profiles->mCamcorderProfiles.add(SM1080P120Profile);

}

break;

N之后的做法：

3、以6739的480P为例：

请check 文件mediaprovider.java中的setCaptureRate是否有按如下方式定义：

frameworks/base/media/java/android/media/MediaRecorder.java

public void setCaptureRate(double fps) {

// Make sure that time lapse is enabled when this method is called.

setParameter("time-lapse-enable=1");

setParameter("time-lapse-fps=" + fps);

}

4、

在/device/mediatek/mt6739/media\_profiles.xml中，将相关item的frameRate从30改成120

<EncoderProfile quality="480p" fileFormat="3gp" duration="30">

<Video codec="h264"

bitRate="4500000"

width="640"

height="480"

frameRate = "30" />

改为

<EncoderProfile quality="480p" fileFormat="3gp" duration="30">

< Video codec="h264"

bitRate="4500000"

width="640"

height="480"

frameRate = "120" />

## 4、兼容前后相同的摄像头

### 平台：6739

以下以兼容前后摄像头都是5025为例

（0）复制原有的5025相关的kernel和hal的代码命名为5025A，把原来命名为5025的文件改为5025A，具体如下：

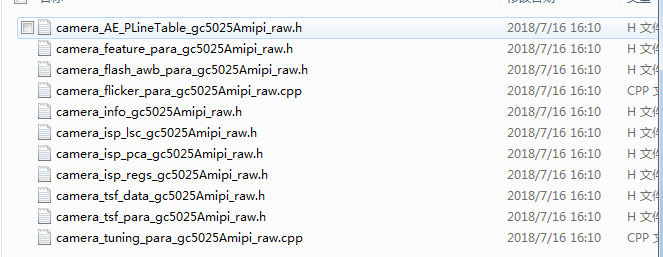
路径：kernel-4.4\drivers\misc\mediatek\imgsensor\src\common\v1\gc5025\_mipi\_raw\



路径：alps\vendor\mediatek\proprietary\custom\mt6739\hal\sendepfeature\ gc5025\_mipi\_raw\

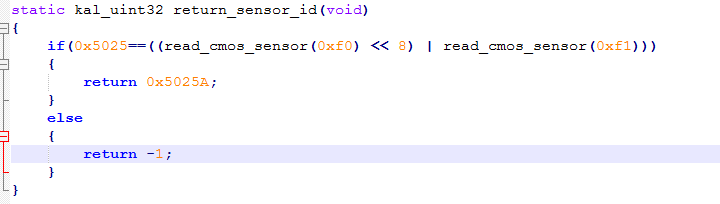


路径：vendor\mediatek\proprietary\custom\mt6739\hal\imgsenso \gc5025\_mipi\_raw\



文件里面的头文件命名也记得改过来。

（1）、修改kernel里面的alps\kernel-4.4\drivers\misc\mediatek\imgsensor\src\common\v1\gc5025\_mipi\_raw\gc5025mipi\_Sensor.c文件里面的读取ID的地方，把读到5025的ID强制转换为5025A，这是为了让系统识别出和另一个5025的区别具体如下。



## 5、摄像头常见问题

平台：6739

### （1）、前后摄像头出现连接不上CCT，或者调试过程中断开

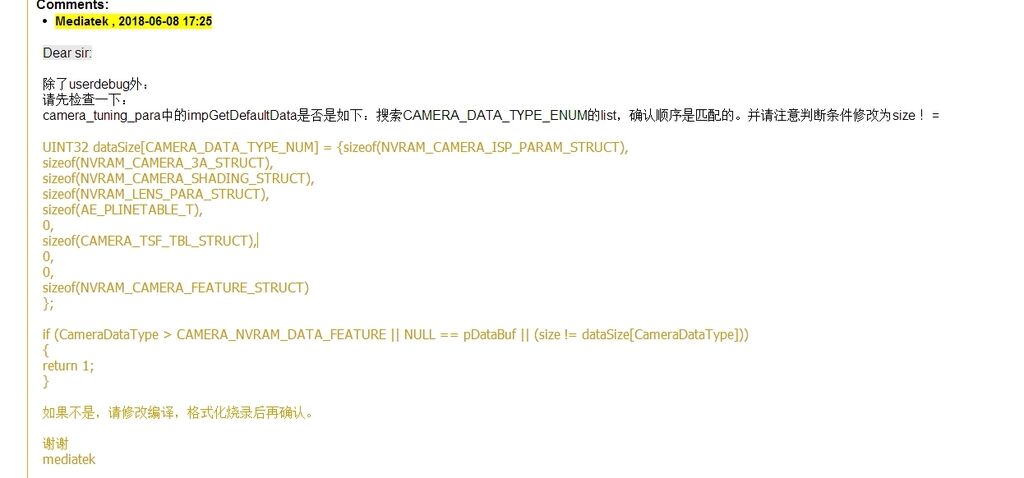
有以下两种改法可供参考：

1）

文件路径：alps\vendor\mediatek\proprietary\custom\mt6739\cgen\cfgfileinc\CFG\_Camera\_File\_Max\_Size.h

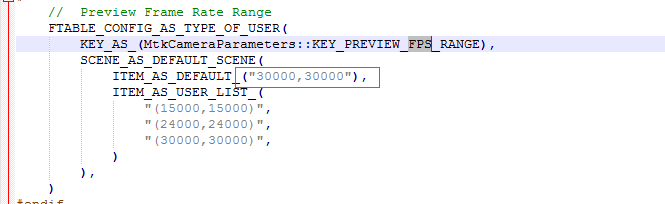
相应的宏修改为： #define MAXIMUM\_NVRAM\_CAMERA\_FEATURE\_FILE\_SIZE (5344)

2）、

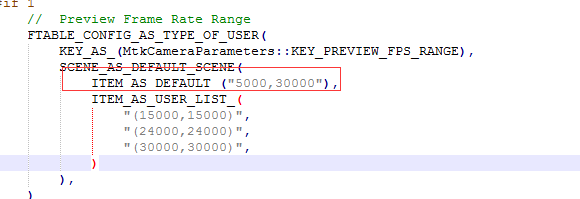


### （2）、预览或者录像被设置为固定帧率30帧

修改config文件（如）中



修改为：



其中5000是最低帧率5帧的意思。

### （3）、录像模式如果打开eis模式界面会变暗

修改下面两个文件：

文件路径：

alps\vendor\mediatek\proprietary\hardware\mtkcam\legacy\platform\mt6739\v1\adapter\MtkDefault\MtkDefaultCamParameter.cpp

修改方法：

把

//if(mpParamsMgr->getRecordingHint() && mpParamsMgr->getVideoStabilization())

改为

if(mpParamsMgr->getRecordingHint() && mpParamsMgr->getVideoStabilization() && (!isEisWithDfr))

文件路径：

alps\vendor\mediatek\proprietary\custom\mt6739\hal\camera\camera\_custom\_eis.cpp

修改方法：

把

bool GetEisLinkWithDfr()

{

return true; // true : dynamic frame rate off when eis on; false : dynamic frame rate on when eis on

}

改为

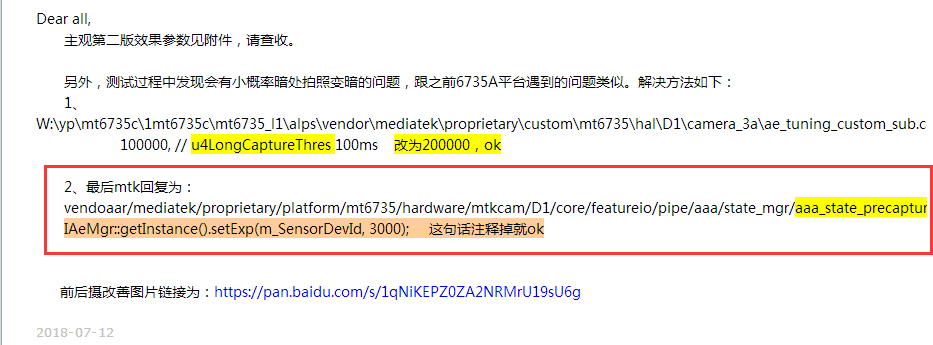
bool GetEisLinkWithDfr()

{

return false; // true : dynamic frame rate off when eis on; false : dynamic frame rate on when eis on

}

### （4）、概率性拍照的时候预览变暗



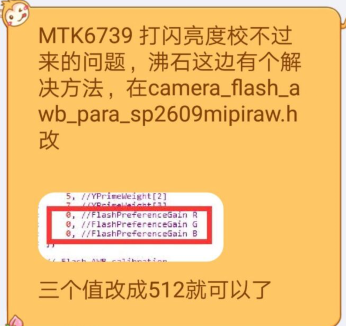
alps\vendor\mediatek\proprietary\hardware\mtkcam\legacy\platform\mt6739\core\featureio\pipe\aaa\state\_mgr\aaa\_state\_precapture.cpp

IAeMgr::getInstance().setExp(m\_SensorDevId, 3000);

目前实践证明：第一种改法有效，第二种改法无效。

### （5）、开启闪光灯拍照拍出来的照片是黑色的

修改的文件在turning文件夹里面



### （6）、调整ISP参数后须要恢复出厂设置之后才可以连接上工具

修改路径：

vendor/mediatek/proprietary/hardware/mtkcam/legacy/platform/mt6739/core/featureio/drv/nvram/nvram\_drv.cpp

修改方法：

if( a\_eNvramDataType==CAMERA\_DATA\_PDC\_TABLE)

{

err = readDefaultData(a\_eSensorType, u4SensorID, a\_eNvramDataType, a\_pNvramData);

if (err != NVRAM\_NO\_ERROR)

logE("read default PDC Table error!");

return err;

}

下面加一段：

if( a\_eNvramDataType==CAMERA\_NVRAM\_DATA\_FEATURE)

{

err = readDefaultData(a\_eSensorType, u4SensorID, a\_eNvramDataType, a\_pNvramData);

if (err != NVRAM\_NO\_ERROR)

logE("read default Feature Table error!");

return err;

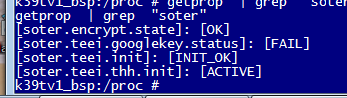
}

## （7）、设置为自动打闪，闪光灯无论在什么场景都会打闪

修改文件camera\_AE\_PLineTable\_gc8024mipi\_raw.h 里面的 -10, //i4StrobeTrigerBV 为 -20, //i4StrobeTrigerBV

# 指纹

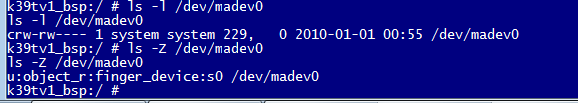
## 1、如果是使用豆荚的TEE，如果要使用指纹需要做加密的动作

（0）、首先通过指令判断机子里面的加密机制有没有被插除掉；输入指令 getprop | grep "soter" 如果出现那就是没有加密，如果是那就是加密过的，指纹可以使用。

（1）、解密狗插入电脑

（2）、打开工具

## 查看指纹驱动节点的权限



## 3、指纹调试步骤（这里以迈瑞威为例子）：

**（1）、调试kernel的代码：**

**1）mt6739.dts（添加属于自己指纹模块的compatible）：**

hct\_finger: hct\_finger {

compatible = "mediatek,hct\_finger"; //这个需要和驱动里面的匹配

};

**2）k39tv1\_bsp.dts**

添加指纹用到的spi的compatible

&spi1 {

status = "okay";

#address-cells = <1>;

#size-cells = <0>;

spi1\_fingerprint@0 {

compatible = "mediatek,fingerprint"; //这个需要和驱动里面的匹配

spi-max-frequency = <6000000>;

reg = <0>;

fingerprint,touch-int-gpio = <&pio 36 0>;

};

};

添加相应指纹的一些节点配置

/\* hct\_finger ctrl start \*/

&hct\_finger {

compatible = "mediatek,hct\_finger"; //这个貌似可以不需要

interrupt-parent = <&pio>;

interrupts = <36 2>;

debounce = <36 0>;

pinctrl-names="default","finger\_power\_en1", "finger\_power\_en0","finger\_power\_18v\_en1","finger\_power\_18v\_en0","finger\_spi0\_mi\_as\_spi0\_mi","finger\_spi0\_mi\_as\_gpio","finger\_spi0\_mo\_as\_spi0\_mo","finger\_spi0\_mo\_as\_gpio","finger\_spi0\_clk\_as\_spi0\_clk","finger\_spi0\_clk\_as\_gpio","finger\_spi0\_cs\_as\_spi0\_cs","finger\_spi0\_cs\_as\_gpio", "finger\_int\_as\_int";

pinctrl-0 = <&finger\_pin\_default>;

pinctrl-1 = <&finger\_power\_en1>;

pinctrl-2 = <&finger\_power\_en0>;

pinctrl-3 = <&finger\_power\_18v\_en1>;

pinctrl-4 = <&finger\_power\_18v\_en0>;

pinctrl-5 = <&finger\_spi0\_mi\_as\_spi0\_mi>;

pinctrl-6 = <&finger\_spi0\_mi\_as\_gpio>;

pinctrl-7 = <&finger\_spi0\_mo\_as\_spi0\_mo>;

pinctrl-8 = <&finger\_spi0\_mo\_as\_gpio>;

pinctrl-9 = <&finger\_spi0\_clk\_as\_spi0\_clk>;

pinctrl-10 = <&finger\_spi0\_clk\_as\_gpio>;

pinctrl-11 = <&finger\_spi0\_cs\_as\_spi0\_cs>;

pinctrl-12 = <&finger\_spi0\_cs\_as\_gpio>;

pinctrl-13 = <&finger\_int\_as\_int>;

status = "okay";

};

&pio {

finger\_pin\_default: finger\_pin\_default {

};

finger\_power\_en1: finger\_power\_en1 {

pins\_cmd\_dat {

pins = <PINMUX\_GPIO13\_\_FUNC\_GPIO13>;

slew-rate = <1>;

output-high;

};

};

finger\_power\_en0: finger\_power\_en0 {

pins\_cmd\_dat {

pins = <PINMUX\_GPIO13\_\_FUNC\_GPIO13>;

slew-rate = <1>;

output-low;

};

};

finger\_power\_18v\_en1: finger\_power\_18v\_en1 {

pins\_cmd\_dat {

pins = <PINMUX\_GPIO4\_\_FUNC\_GPIO4>;

slew-rate = <1>;

output-high;

};

};

finger\_power\_18v\_en0: finger\_power\_18v\_en0 {

pins\_cmd\_dat {

pins = <PINMUX\_GPIO4\_\_FUNC\_GPIO4>;

slew-rate = <1>;

output-low;

};

};

finger\_spi0\_mi\_as\_spi0\_mi: finger\_spi0\_mi\_as\_spi0\_mi {

pins\_cmd\_dat {

pins = <PINMUX\_GPIO33\_\_FUNC\_SPI0\_A\_MI>;

slew-rate = <0>;

bias-disable;

output-low;

input-schmit-enable = <0>;

};

};

finger\_spi0\_mi\_as\_gpio: finger\_spi0\_mi\_as\_gpio {

pins\_cmd\_dat {

pins = <PINMUX\_GPIO33\_\_FUNC\_GPIO33>;

slew-rate = <0>;

output-low;

input-schmit-enable = <0>;

};

};

finger\_spi0\_mo\_as\_spi0\_mo: finger\_spi0\_mo\_as\_spi0\_mo {

pins\_cmd\_dat {

pins = <PINMUX\_GPIO34\_\_FUNC\_SPI0\_A\_MO>;

slew-rate = <1>;

bias-disable;

output-low;

input-schmit-enable = <0>;

};

};

finger\_spi0\_mo\_as\_gpio: finger\_spi0\_mo\_as\_gpio {

pins\_cmd\_dat {

pins = <PINMUX\_GPIO34\_\_FUNC\_GPIO34>;

slew-rate = <1>;

output-low;

input-schmit-enable = <0>;

};

};

finger\_spi0\_clk\_as\_spi0\_clk: finger\_spi0\_clk\_as\_spi0\_clk {

pins\_cmd\_dat {

pins = <PINMUX\_GPIO32\_\_FUNC\_SPI0\_A\_CLK>;

slew-rate = <1>;

bias-disable;

output-low;

input-schmit-enable = <0>;

};

};

finger\_spi0\_clk\_as\_gpio: finger\_spi0\_clk\_as\_gpio {

pins\_cmd\_dat {

pins = <PINMUX\_GPIO32\_\_FUNC\_GPIO32>;

slew-rate = <1>;

output-low;

input-schmit-enable = <0>;

};

};

finger\_spi0\_cs\_as\_spi0\_cs: finger\_spi0\_cs\_as\_spi0\_cs {

pins\_cmd\_dat {

pins = <PINMUX\_GPIO31\_\_FUNC\_SPI0\_A\_CSB>;

slew-rate = <1>;

bias-disable;

output-low;

input-schmit-enable = <0>;

};

};

finger\_spi0\_cs\_as\_gpio: finger\_spi0\_cs\_as\_gpio {

pins\_cmd\_dat {

pins = <PINMUX\_GPIO31\_\_FUNC\_GPIO31>;

slew-rate = <1>;

output-low;

input-schmit-enable = <0>;

};

};

finger\_int\_as\_int: finger\_int\_as\_int {

pins\_cmd\_dat {

pins = <PINMUX\_GPIO36\_\_FUNC\_GPIO36>;

slew-rate = <0>;

bias-pull-up = <00>;

};

};

};

**3）、驱动（alps\kernel-4.4\drivers\misc\mediatek\fingerprint）**

保证驱动的probe可以跑完、而且

**（2）、配置相应的服务**

1.1修改alps\device\mediatek\common\device.mk文件

Android O原生的配置不会编译指纹相关的原生bin和库文件，需要在device.mk中添加打包的配置，然后整编。具体目录按照每家客户修改(device/xxx(厂商)/xxx(具体项目型号))

#add by microarray #(这个固定加上去即可)

PRODUCT\_PACKAGES += android.hardware.biometrics.fingerprint@2.1-etservice

PRODUCT\_PACKAGES += android.hardware.biometrics.fingerprint@2.1

PRODUCT\_PACKAGES += android.hardware.biometrics.fingerprint@2.1-service

PRODUCT\_COPY\_FILES += frameworks/native/data/etc/android.hardware.fingerprint.xml:system/etc/permissions/android.hardware.fingerprint.xml

说明：

　　　　android.hardware.biometrics.fingerprint@2.1-service在打包编译bin文件的时候，会启动该服务的android.hardware.biometrics.fingerprint@2.1-service.rc文件打包的init.rc中，该模块需要整编，不然该bin文件起不来，导致fingerprint.default.so打不开。该bin文件存放在手机的/vendor/bin/hw/目录下

　　　　android.hardware.biometrics.fingerprint@2.1-service.so:实现fingerprintservice和android.hardware.biometrics.fingerprint@2.1-service通讯的binder化的库文件，在手机的system/lib(lib64)/目录下

1.2 指纹原生相关库和bin文件 //这一点我们不需要操作

　　　Android O去掉了fingerprintd，并引入了android.hardware.biometrics.fingerprint@2.1-service来启动fingerprint.default.so，用HIDL来实现Framework和android.hardware.biometrics.fingerprint@2.1-service的通讯，这部分源码在hardware/interfaces/biometrics/fingerprint/2.1/default/目录下。

　　　android.hardware.biometrics.fingerprint@2.1.so，这个HIDL化的binder，源码在:hardware/interfaces/biometrics/fingerprint/2.1/目录下。

1.3 修改alps\device\mediatek\mt6739\manifest.xml文件

该文件是注册指纹的hidl，这样android系统能够通过该注册的方式找到想应的binder来和android.hardware.biometrics.fingerprint@2.1-service进行通讯，[从而使得fingerprintservice能够通过binder调用到android.hardware.biometrics.fingerprint@2.1-service](mailto:从而使得fingerprintservice能够通过binder调用到android.hardware.biometrics.fingerprint@2.1-service)接口，并进行指纹的相关操作。

该文件在device/xxx(厂商)/xxx(具体项目型号)/目录下。具体配置如下：

<hal format="hidl">

<name>android.hardware.biometrics.fingerprint</name>

<transport>hwbinder</transport>

<version>2.1</version>

<impl level="generic"></impl>

<interface>

<name>IBiometricsFingerprint</name>

<instance>default</instance>

</interface>

</hal>

**如果没有配置会有如下log打印：**

01-11 04:28:51.647 596 596 W /system/bin/hwservicemanager: getTransportFromManifest: Cannot find entry android.hardware.biometrics.fingerprint@2.1::IBiometricsFingerprint in either framework or device manifest, using default transport.

01-11 04:28:51.648 1457 1490 E system\_server: service android.hardware.biometrics.fingerprint@2.1::IBiometricsFingerprint declares transport method EMPTY but framework expects hwbinder.

01-11 04:28:51.649 1457 1490 E FingerprintService: Failed to get biometric interface

01-11 04:28:51.649 1457 1490 E FingerprintService: android.os.RemoteException: HwBinder Error: (-2147483648)

01-11 04:28:51.649 1457 1490 E FingerprintService: at android.os.HwBinder.getService(Native Method)

01-11 04:28:51.649 1457 1490 E FingerprintService: at android.hardware.biometrics.fingerprint.V2\_1.IBiometricsFingerprint.getService(IBiometricsFingerprint.java:44)

01-11 04:28:51.649 1457 1490 E FingerprintService: at com.android.server.fingerprint.FingerprintService.getFingerprintDaemon(FingerprintService.java:238)

01-11 04:28:51.649 1457 1490 E FingerprintService: at com.android.server.fingerprint.FingerprintService$FingerprintServiceWrapper.isHardwareDetected(FingerprintService.java:1190)

01-11 04:28:51.649 1457 1490 E FingerprintService: at android.hardware.fingerprint.IFingerprintService$Stub.onTransact(IFingerprintService.java:156)

01-11 04:28:51.649 1457 1490 E FingerprintService: at android.os.Binder.execTransact(Binder.java:674)

01-11 04:28:51.649 1457 1490 W FingerprintService: fingerprint HIDL not available

**（3）、权限配置**

alps\device\mediatek\mt6739\sepolicy\bsp \device.te

#add for microarray fingerprint begin

type finger\_device, dev\_type;

#add for microarray fingerprint end

alps\device\mediatek\mt6739\sepolicy\bsp \file\_contexts

#add for microarray fingerprint begin

/dev/madev0 u:object\_r:finger\_device:s0

/data/system/ma\_fingerprint/\* u:object\_r:ma\_data\_file:s0

alps\device\mediatek\mt6739\init.mt6739.rc

#add for microarray fingerprint

mkdir /data/system/ma\_fingerprint 0770 system system

#add for microarray fingerprint

chmod 0660 /dev/madev0

chown system system /dev/madev0

alps\device\mediatek\mt6739\ manifest.xml

<hal format="hidl">

<name>android.hardware.biometrics.fingerprint</name>

<transport>hwbinder</transport>

<version>2.1</version>

<impl level="generic"></impl>

<interface>

<name>IBiometricsFingerprint</name>

<instance>default</instance>

</interface>

</hal>

alps\device\mediatek\sepolicy\bsp\non\_plat\init.te

allow init ma\_data\_file:dir {create\_file\_perms};

allow init ma\_data\_file:file {create\_file\_perms};

alps\system\core\rootdir\ ueventd.rc 这个文件暂时不知道干啥。

alps\device\mediatek\mt6739\sepolicy\bsp\hal\_fingerprint\_default.te

#add for microarary fingerprint begin

allow hal\_fingerprint\_default finger\_device:chr\_file {open write read ioctl};

allow ma\_data\_file labeledfs:filesystem associate;

allow hal\_fingerprint\_default self:capability {dac\_override dac\_read\_search};

file\_type\_auto\_trans(hal\_fingerprint\_default, system\_data\_file, ma\_data\_file);

allow hal\_fingerprint\_default ma\_data\_file:file {create\_file\_perms};

allow hal\_fingerprint\_default ma\_data\_file:dir {open read write search};

allow hal\_fingerprint\_default teei\_fp\_device:chr\_file { read write ioctl open }; #打开spi的操作权限

#add for microarray fingerprint end

**（4）、增加一些上层操作库文件(**这些文件放置的路径和device.mk里面配置有关**)**

alps\device\mediatek\common\device.mk 配置：

PRODUCT\_COPY\_FILES += \

frameworks/native/data/etc/android.hardware.fingerprint.xml:system/etc/permissions/android.hardware.fingerprint.xml

PRODUCT\_COPY\_FILES += \

vendor/transsion/device/common/micro\_tee/system/lib/hw/fingerprint.default.so:system/lib/hw/fingerprint.default.so \

vendor/transsion/device/common/micro\_tee/system/lib/libfprint-x32.so:system/lib/libfprint-x32.so \

vendor/transsion/device/common/micro\_tee/system/vendor/thh/fp\_server:/vendor/thh/fp\_server

例如：

alps\vendor\transsion\device\common\micro\_tee\system\lib\libfprint-x32.so

alps\vendor\transsion\device\common\micro\_tee\system\lib\hw\fingerprint.default.so

alps\vendor\transsion\device\common\micro\_tee\system\lib\hw\microarray.default.so

alps\vendor\transsion\device\common\micro\_tee\system\vendor\thh\fp\_server

# 电池曲线相关知识

## 一些宏的含义

（1）、

#define TEMPERATURE\_T0 25

#define TEMPERATURE\_T1 10

#define TEMPERATURE\_T2 0

#define TEMPERATURE\_T3 -10

#define TEMPERATURE\_T4 -20

BATTERYPSEUDO100 //100%藏帕

## 2、关于节点 /sys/class/power\_supply/battery/

节点的作用：上报一些关于电池的数据：充电状态、充电类型、电池的健康状态、电量水平、温度、电压、电流、等等。

文件：alps\kernel-4.4\drivers\power\ power\_supply\_sysfs.c

mtk\_battery.c

关键字：battery\_main

## 3、开机后的电池电量显示高于关机前电量显示

该现象属正常，手机剩余电量是通过监测电池电压值得到的。

手机处于稳定的开机状态时，电流较为稳定，锂离子在电解液中的分布有较为稳定的浓度梯度。

关机后就没有电流了，锂离子的浓度梯度也会逐渐减低并最终变为零，这时电压就会上升。

再开机，由于形成稳定的浓度梯度需要一个短暂的时间，因此电池电压会暂时比关机前更高，但手机电池的实际电量并没有增加。

## 4、关于GM3.0充电问题的ONLINE

### [FAQ20713] GM3.0电量问题处理流程

一、客户自查

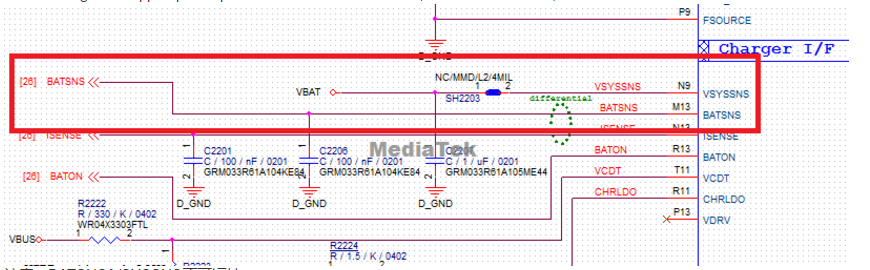
电量相关问题，请先排查如下几点：

1、check hardware schematic

硬件设计对电量计算有极大影响，所以务必确认如下两项：

（1）ISENSE/BATSNS硬件连接是否和对应平台参考设计相同

使用的charger IC support power path则使用ISENSE测量VBAT，反之使用BATSNS；



注意：BATSNS/VSYSSNS不可短接

（2）CP\_S/CP\_N 差分包地走线

PCB走线注意细则参考文档：《Fuel Gauge Application Notes\_Vx.pptx》

2、check ZCV table

（1）是否有按规范打表

（2）是否有按规范合入dtsi

操作指导参考文档：  
《GM3.0 Customized Setting Flow\_Vx.pptx》

3、check car\_tune\_value

确认是否有按规范校准car\_tune\_value

car\_tune\_value校准方法参考文档：  
《GM3.0 Customized Setting Flow\_Vx.pptx》

4、check customization

确认是否做过客制化修改，如有请提供修改内容；

mtk\_battery\_property.h

mtk\_battery\_table.h

若支持5个温度，则提供：

xxx\_battery\_prop\_ext.dtsi

xxx\_battery\_table\_ext.dtsi

若支持4个温度则提供：

xxx\_battery\_prop.dtsi

xxx\_battery\_table.dtsi

注：如果贵司做过客制化，从头文件解析zcv，请说明；

客制化参数含义参考文档： 《 GM3.0\_Customization parameter design guide\_Vx.pptx 》

二、提CR注意事项

如若上述自查均PASS，电量计算仍存在问题，劳烦开启daemon log，提供问题复现步骤和log：

1、 fuelgauge daemon log开启方法：

（1）setprop（注意需要root权限）

adb shell setprop persist.mediatek.fg.log.enable 1

（2）改code，重编bootimage：

/kernel/drivers/power/mediatek/battery/mtk\_battery.c文件中：

//static signed int gFG\_daemon\_log\_level = BM\_DAEMON\_DEFAULT\_LOG\_LEVEL;

修改为：

static signed int gFG\_daemon\_log\_level = 8;

static int Enable\_BATDRV\_LOG = 3;

修改为：

static int Enable\_BATDRV\_LOG = 8;

（3） adb 命令写节点，但重启会消失（注意需要root权限）：

adb shell

echo 8 > /sys/devices/platform/battery/FG\_daemon\_log\_level

三、学习文档

针对电量计初学者，务必先到敝司DCC下载相关文档，了解电量计的测试调试方法、参数意义；

必读文档列举如下：

Fuel Gauge Application Notes\_Vx.pptx

Gauge Master 3 0 Introduce\_Vx.pptx；

GM3.0 Customized Setting Flow\_Vx.pptx；

GM3.0\_Customization parameter design guide\_Vx.pptx；

四、更换Charger IC注意事项

若贵司使用友商提供的charger，则porting时需要注意如下修改点：

1、使用友商Charger IC + 敝司fuelgauge 3.0方案：

① lk中务必porting charger ic driver,确保可以正常停充、disable power\_path;

停充和disable power\_path 的位置：

若定义MTK\_CHARGER\_NEW\_ARCH（自o1.mp1引入）：

/vendor/mediatek/proprietary/bootable/bootloader/lk/platform/common/power/mtk\_battery.c文件中check\_sw\_ocv()；

未定义MTK\_CHARGER\_NEW\_ARCH：

/vendor/mediatek/proprietary/bootable/bootloader/lk/platform/mt6771/mt\_battery.c文件中 mt65xx\_bat\_init()；

②kernel中charger IC driver porting

确保driver API符合/kernel-4.4/drivers/misc/mediatek/include/mt-plat/charger\_class.h文件中struct charger\_ops规范；

需要注意的是，若对应charger IC支持HWOCV测量功能，请添加如下callback:

int (\*get\_zcv)(struct charger\_device \*, u32 \*uV);

2. 使用友商Charger IC + 外挂gauge，则disable gm3.0对应修改：

（1） kernel部分

① 定义宏：CONFIG\_MTK\_DISABLE\_GAUGE= yes；

② Android O版本还需在battery\_probe()中作如下修改，disable recovery mode调用：

/\*\*\*\*\*\*

if (is\_recovery\_mode())

battery\_recovery\_init();

\*/

（2）system部分：

disable所有fuelgauge service启动code；

①init.mt67xx.rc

on charger

#start fuelgauged

service fuelgauged /vendor/bin/fuelgauged

# class main

user root

group root

disabled

#注意，Android O无此项

service fuelgauged\_nvram /vendor/bin/fuelgauged\_nvram

# class main

user root

group root

oneshot

disabled

②meta\_init.rc、recovery/etc/init.rc等做同步修改

（3）change fgadc reset source

不使用GM3.0的话，需要将寄存器 PMIC\_RG\_FGADC\_RST\_SEL\_ADDR设为0，code修改位置：

①自alps-mp-o1.mp1 branch MT6771平台后引入MTK\_CHARGER\_NEW\_ARCH，则在lk中作如下修改：

\*\*/lk/platform/common/power/mt\_battery.c文件中，修改使is\_disable\_bat()函数返回true

②之前平台或未定义MTK\_CHARGER\_NEW\_ARCH，则直接在Preloader中修改：

/vendor/mediatek/proprietary/bootable/bootloader/preloader/platform/mt67xx/src/drivers/battery.c :

void fuel\_gauge\_init(void)

{

.......

//only for GM 3.0 : set FGADC reset source selection = 1 ,GM3.0 reset

ret = pmic\_config\_interface((U32)(PMIC\_RG\_FGADC\_RST\_SRC\_SEL\_ADDR), (U32)(0),

(U32)(PMIC\_RG\_FGADC\_RST\_SRC\_SEL\_MASK),

(U32)(PMIC\_RG\_FGADC\_RST\_SRC\_SEL\_SHIFT));

.....

}

五、常见问题

1、电量显示-1

表示fuelgauge deamon无法正常运行，导致的原因多为fuelgauge deamon和kernel不匹配；

劳烦先自行排查libfgauge\_gm30.so和kernel是否属同一基线版本；

也可提供 mtk\_battery\_internal.h / mtk\_battery.h 以及客戶libfgauge\_gm30.so，由敝司帮忙排查；

2、电量固定为25%、50%、75%

需确认：

是否为factory mode 或 meta mode？

是否disable gauge？

是否使用假电池？

是否EVB？

是否客制化电量上报？

3、1%持续时间长

（1）重载放电

[ Method]减小pseudo1参数值Q\_MAX\_SYS\_VOLTAGE； [Risk]更重载情况下会有drop现象；

（2）轻载放电

[Method1]减小gauge0% 参数值PMIC\_MIN\_VOL/POWERON\_SYSTEM\_IBOOT； [Risk]轻载放至低电量切换到重载会drop；

[Method2]调高0%关机电压SHUTDOWN\_GAUGE0\_VOLTAGE； [Risk]电量无法耗尽，轻载关机电池电压高；

[Method3]降低UI 1%持续时间SHUTDOWN\_1\_TIME； [Risk]电量无法耗尽，轻载关机电池电压高；

[Method4]降低 DIFFERENCE\_FGC\_FGV\_TH; [Risk]曲线不线性；

（3）重载转轻载

属正常现象，电池本身特性所致；

4、100%持续时间长

（1）放电场景

[Method1]增大pseudo100； [Risk]无，呈现充电本身特性；

[Method2]disable PSEUDO100\_EN\_DIS； [Risk]无，使充电藏帕放电不藏帕

[Method3]减小KEEP\_100值； [Risk]无；

（2）充电场景

[Method]增大pseudo100； [Risk]无，呈现充电本身特性；

5、放电Drop

表现：VBAT达到3.4V关机时，uisoc还比较高；

（1）重载场景

[Method]增大pseudo1；

（2）轻载场景

[Method]增大gauge 0% ；

（3）轻载至低电量切换至重载场景

[Method]正常现象，电池本身特性导致；

6、放电关机电压未抵到3.4V

表现：UI已经到0%关机，但电池电压仍高于3.4V；

（1）中重载场景

[Method1]降低gauge 0%； [Risk]轻载至低电量切换到重载会有drop；低电量没插充电器开不了机；

[Method2]降低SHUTDOWN\_GAUGE0\_VOLTAGE； [Risk]可能导致电池过放；

（2）轻载场景

[Method1]增加UI 1% timeout； [Risk]UI 1%显示时间较长；

[Method2]降低SHUTDOWN\_GAUGE0\_VOLTAGE； [Risk]可能导致电池过放；

（3）轻载至低电量切换至重载场景

[Method]正常现象，电池本身特性导致；

### [FAQ17516] GM2.0 电池未饱电量却显示100%

现象：

1.电池未饱开机后电量却显示100%，例如：电池电压为4.27V对应ZCV Table mapping出来的电量为96%，开机后却显示电量为100%；

2.放电时100%维持时间很久；

3.GM2.0打开后充电到100%明显变快了。

[SOLUTION]

首先，这是正常现象，可以通过修改客制化宏值BATTERYPSEUDO100调整，这部分出于用户感受度调整的算法；这样处理的目的是，由于手机给电池充电CV阶段末期充电电流很小，可能给客户的体验是9x%～100%或者100%充电很久的感受，我们通过这种方法将这部分充电时间藏起来不让用户看到；另外一个目的是，可能有些用户反馈100%电量用的时间很短，我们通过这种方法让100%显示的时间比较久，让用户觉得电池比较耐用。

其次，GM2.0算法将电量分为SOC/UI\_SOC/UI\_SOC2三层来处理：

SOC:实际反映电池当前的容量，通过老化、温度补偿、负载补偿和库仑器等实时计算的结果；

UI\_SOC:在SOC的基础上做一些单调性和平滑处理后的电量值，因为SOC会根据实际电池情况变化有可能会跳变；

UI\_SOC2:实际SHOW给用户看到的电量，根据UI\_SOC和两个客制化宏值BATTERYPSEUDO100/BATTERYPSEUDO0线性mapping出来；

这部分介绍可以参考文档《Gauge Master 2 1 Introduce\_V1.1.pptx》；

最后，适当修改BATTERYPSEUDO100就可以改善这几个现象，具体改善效果如下两个表格：

BATTERYPSEUDO100 = 95时的情况：

ui\_soc pseudo100 ui\_soc2

94 95 99

95 95 100

96 95 100

97 95 100

98 95 100

99 95 100

100 95 100

BATTERYPSEUDO100 = 98时的情况：

ui\_soc pseudo100 ui\_soc2

97 98 99

98 98 100

99 98 100

100 98 100

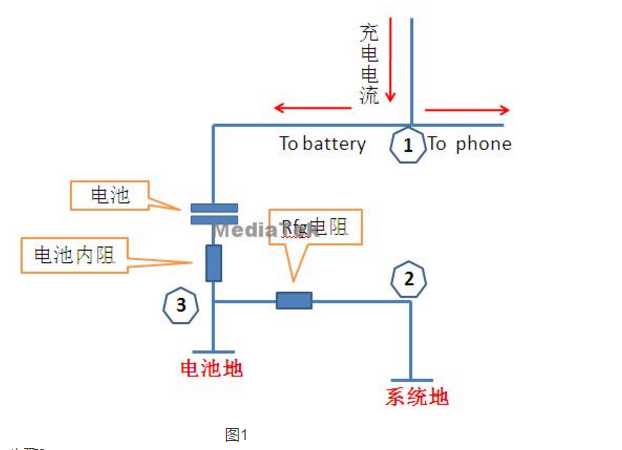
### [FAQ05421] [BMT]怎样去校准CAR\_TUNE\_VALUE?

步骤1

如图1，充电的大概框架，充电从1点分两路，To battery一路与To phone.

To battery 流过电池+电池内阻+rfg电阻.

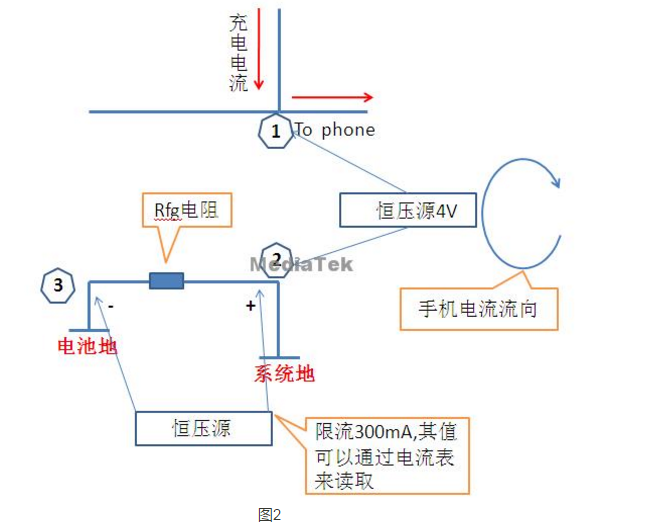
3点为电池地。2点为系统地。



步骤2

如图2，拔掉电池，在1点，与2点，加个恒压源，定为4V。为手机供电。并让手机开机进入工厂模式。

并在rfg加入恒压源，具体做法，如图2在2点与3点加个恒压源，2点为+，3点为-，并限流300mA，具体请通过精度比较高的电流表来读数值。确保为300mA（大概300mA）.



步骤3.

如图2，有两条电流回路，互不影响。

步骤4.

如步骤2设定流过rfg的电流为300mA，通过工厂模式，读取电流值为A，假设从手机上面读出来

是330mA

测试得到Car\_tune\_value = 300/330 = 0.909 = 91

步骤5:

将CAR\_TUNE\_VALUE = 91的值写入软体，并编译。

重新进入工厂模式

重新设定恒压源的电流值为

300mA 读取工厂模式下电流值 A

400mA 读取工厂模式下电流值 B

500mA   读取工厂模式下电流值  C

600mA   读取工厂模式下电流值  D

如果300mA与A 400mA与B 500mA与C  600mA与D

相差在3%的范围内，说明校准了。

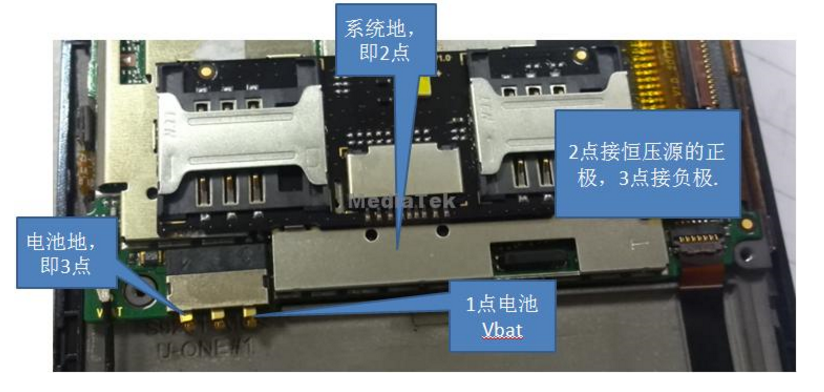
步骤6，

重复2--5，多测几台手机，取个平均数。

将这个平均数写入CAR\_TUNE\_VALUE.

如图3显示，1点，2点与3点，对应的手机的具体位置.

请注意。1点与3点的位置请根据电池的放置的位置来决定。



## 5、充电logo

Kernel充电logo排版：alps\vendor\mediatek\proprietary\external\libshowlogo\cust\_display.h

编译：需要全编烧写system.img logo.bin 才有效果。

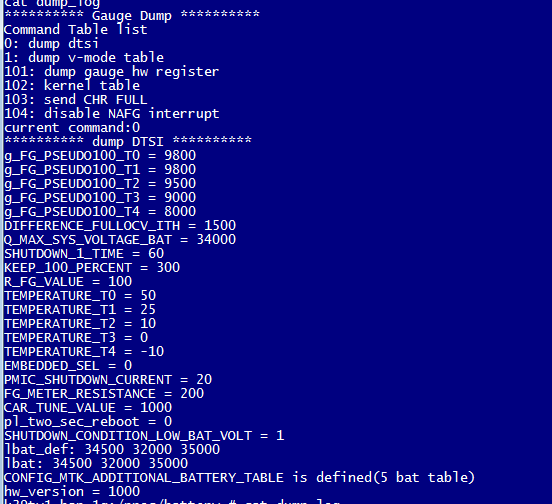
## 6、分析充电

**（1）通过查看抓到的MTKlog**

软件搜索 vbat= 充电的容量 CAR=

**（2）、通过adb 读取一些配置信息**

/proc/battery/dump\_log :



实现的函数（alps\kernel-4.4\drivers\power\mediatek\battery\mtk\_battery.c）：

void battery\_debug\_init(void)

## 7、计算电池电池中的锂含量

公式：电池容量（安时）\* 30% = 电池中的锂含量（克）

## 8、开关机电压设置

开机电压（lk）：

路径：/vendor/mediatek/proprietary/bootable/bootloader/lk/platform/mt6739/mt\_battery.c

关键字：BATTERY\_LOWVOL\_THRESOLD

关机电压：

路径：alps\kernel-4.4\drivers\power\mediatek\battery\mtk\_power\_misc.c

函数：static int shutdown\_event\_handler(struct shutdown\_controller \*sdd)

宏：

Q\_MAX\_SYS\_VOLTAGE\_BAT0

SHUTDOWN\_GAUGE0\_VOLTAGE //关机电压条件

# GPS相关知识

## 1、GPS 的一些分析手段

摘至：[FAQ07950] 如何分析GPSLog

对于GPSLog有两种类型的文件，主要是依据其文件内容划分。文件内部存储的只是$GP开口的log，那么这种log称为NMEA log；文件内部除了$GP的log，还有其他的一些log，例如$PMTK的log等等，这种log称为GPS DebugLog。

其实一般情况下，可以通过NMEALog就可以看出来很多的信息。看NMEALog主要是对NMEA协议里的集中常用NMEA语句的各个字段熟悉即可。

1. 基本检查

GPGGA中可以得到UTC时间、经纬度、当前定位状态、使用的卫星数目、HDOP等等；

GPGSA中可以得到定位类型、所使用到的卫星ID；

GPGSV中可以得到可视卫星的数目，然后会一次列举出每科可视卫星的ID、方位角、仰角和信号强度；

GPRMC中可以得到UTC时间、定位状态、经纬度、速度、UTC日期；

GPACCURACY是我司加入的获取定位精度的语句。

2. 检查SNR

根据SNR值是很好的判断卫星信号强弱的方法，强信号是指CNR大于40db，中讯号是指 CNR大于30db，弱讯号是指CNR大于20db，20db以下视为极弱讯号。我司根据的spec:6颗卫星SNR>40db，可以达到CEP:3m。

在测试传导性能/整机性能的时候，打入-130dbm的信号，如果可以看到收到的卫星SNR>40.5db,这表示贵司的硬件GPS天线做的不错;如果是在35db以下，可能就是表示硬件GPS做的很差喽。

3. 缺少哪些辅助资讯

如果产看是否触发AGPS，需要从GPS debug Log中分析，搜索PMTK730这个语句，因为辅助数据主要是看时间、位置、星历这三个。$PMTK730,0,0,0,1,0,1,1,0：从这个语句中的一个1表示缺少星历、第二个1表示缺少位置、第三个1表示缺少时间。PMTK710表示获取星历辅助数据，PMTK712表示获取时间辅助资讯，PMTK713表示缺少位置辅助资讯。

当所有的辅助资讯都有拿到之后，显示出来的PMTK730后面的都会是0.

4. 查看卫星分布如何

通过看GPGSV里的卫星的方位角和仰角可以判断卫星的位置，通过判断GPGSA里的卫星可以知道当前定位使用了哪些卫星，这样就可以知道当前使用定位计算的这些卫星的卫星分布如何、信号强度如何。

通过GPGGA里的HDOP来判断当前卫星的分布如何，这个值越小越好。卫星分布越好，越有可能产生的定位精度越高。所以HDOP是定位精度的充分但非必要条件。

5. 当前有哪些星历被保存

通过搜索GPS Debug Log文件中PMTKEPH，该关键词会告诉你当前有哪些卫星是已经解析下来辅助资讯的。例如PMTKEPH,3,12,15,18表示有3颗卫星已经解析下来了星历数据，他们的卫星ID 是12、15、18.

6. 如何判断GPS已经正常工作

GPS工作起来首先会在GPS debug Log中打印出来PMTK010的字样，所以通过判断这个可以知道GPS是否有正常工作起来。

7. 星历数据是来自EPO、HotStill还是实时解算

如果知道当前使用到的卫星星历数据是来自实时接收解算下来的星历、还是EPP,或者是HotStill，同样也是通过判断PMTKEPH。如果该关键字后面跟着的卫星ID是正整数，那么是实时接收解算下来的星历；

如果是浮点数，而且是负数，但是小数点后都是0，表示的是EPO；

如果是浮点数，而且是负数，但是小数点后的是非0，表示的是Hotstill；

8. 判断AGPS/EPO/HotStill是否工作

从GPS Debug log中搜索到wk，epo表示EPO有工作；

从GPS debug log中搜索到wkbee表示Hotstill有工作；

从GPS Debug log中搜索到wkssi表示AGPS有工作。

9. H/W/C/F GPS Start

通过从打开GPS的第一条GPGGA语句来确认是什么启动方式，如果该语句中的时间是235944，那么表示没有时间辅助资讯，这是完全冷启动；如果有时间信息，但是位置是8960.0000，N，0000.0000，E，这表示是冷启动；如果既有时间也有位置，可以看第一条PMTKEPH中是几颗有卫星的星历，如果一颗没有，那么这是暖启动；如果发现也有位置、时间、而且至少4颗以上的星历，那么这是热启动。

10. 定位

看定位精度，请以3D定位为准。因为2D定位下定位精度会很大。

11. 如何判断是否是coclock 1、000045.784,HBD,IF1,BOT,ClkType,254这里的254表示是coclock，255表示是TCXO。**如果是coclock，那么可以通过如下log判断是否有校准成功。**

000069 $PMTK013,32,ClkType,254,C0,1.373238,C1,-0.196573,k,1\*24 2、还可以通过kernel log来判断是否校验成功 ：首先：Co\_clock\_flag=1表示gps co clock；0表示TCXO。其次

12, 如何判断GPS power 成功。请通过GPS\_INIT\_DONE来确认。

13，如何判断之前有没有定位成功过。

这在AGPS 认证测试中经常会用来判断测试前是否有3D fix过。

请通过PMTKDBG的倒数第五个数据是否非0，如下红色部分。

000103 $PMTKDBG,12,0,0,0,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0,0,0,0,0,

0093,0001,0002,0000,0000,0022,0001,0000,0.0000,0.0224,0.0000,0.0000,0.0000,0.0000,\*4C

14, 如何确认MNL Version。

通过000045.807,HBD,IF1,BOT,MNLVer,AXN\_3.60,1312,MNL\_VER\_14011701ALPS05\_3.60\_06,22,40

15,如何确认acqusition、bit sync、subframe sync的状态。

请过如下log，下面log中表示捕获到9颗卫星，有9颗卫星已经bit sync，6颗卫星子帧同步。

$PMTK015,532335,6,438756,NACQ,9,NBS,9,NSFS,6,NEXS,6\*6C

16、如何检查clock的稳定性？

Clock的稳定对gps的performance有很大影响，所以在设计的时候，要努力做到有稳定的clock。

目前MTK制定的clock指标数据，可以参考如下：

没有热干扰的情况下，建议clock drift 小于2.5ppb/s。

在有热干扰的情况下，建议clock drift小于10ppb/s。

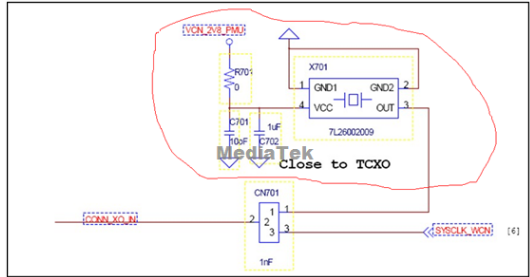
检查产品的clock，需要抓取一份保持定位10min中的gps debug log，将该log导入到gps doctor工具中，就可以看到clock drift的数据。

## 2、GPS的Coclock方案和TCXO方案的区别与联系

### 平台： MT6737 MT6739 MT6580

（1）、什么是Coclock方案？

Coclock是会节省一颗TCXO物料，即下图中的红色框中的物料。所以clock source的来源就会变成从另外的地方取得



（2）、Coclock为何需要校准？

目前GPS/WIFI/FM/BT都是需要26M的clock 的，尤其是GPS，对clock的要求会比较高。如果没有经过校准，GPS会遇到各种不预期的行为。校准主要是为了校准出来一条频率对温度的曲线，这样软件就可以使用该曲线学习到clock的准确值。目前产线经过校准，校准的温度区间比较有限。其余的温度范围内的曲线是推算出来的。

（3）、Coclock 方案和TCXO方案，在end user使用上有什么不同？

TCXO是会提供比较精准的clock。而Cocock提供的时钟相对没有那么精准，是需要软件的补偿，并且要经过长时间的学习，才可以达到和TCXO相当的效果。Coclock情况下，每当在不同的温度下进行定位过程后，都会有学习到频率温度曲线。当频率对温度曲线，温度范围越广，表示学习的越完整，那么end user就会体验越好。

## 3、下载程序后或者恢复出厂设置后的第一次GPS可能可以搜得到之后就不行了

### 平台： MT6737 MT6739

原因：MTK的GPS的 Coclock 方案需要先校准GPS才可以定位。

如何从log中看出有没有校准信息：

首先通过log关键字ClkType,254 ；这里的254就是表示coclock，255表示是TCXO。

然后如果是 coclock 那么可以通过如下log判断是否有校准成功。

000069 $PMTK013,32,ClkType,254,C0,1.373238,C1,-0.196573,k,1\*24。

# 光感和接近

## 1、[FAQ06568] [sensor]如何解决光感不灵敏的问题

由于结构限制或者传感器本身原因，工厂模式下检测到ic出来的值变化不够明显，可检测的范围小等。导致自动背光效果不明显。

[SOLUTION]

硬件上没有方法改动的话，建议合理的修改，cust\_alsps.c中的.level和.value这2个数组的值，driver里处理数据的原理：**【其中6739这两个表格已经被拉到dts中】**

从driver里传感器IC寄存器出来的数据是level值，根据此level值参照上述2个数组的对应关系，由level数组的下标取到.value数组的下标对应的值，然后上报value值给工厂模式和上层看到。

所以需要您调cust\_alsps.c里的那2个数组，如果是被遮挡而引起变化范围变小的话，应该是寄存器出来的level变小了，所以建议把cust-als.c里的level也相应的减小，value不变，则应该有所改善。多试几组减小.level的情况。

另外如果需要更细腻的背光等级，可以适当的把这2个数组加大，取值更连续一些。（这个映射还与android的value(LUX)--->255级背光的映射有关，也不是连续的）

请参考alps\device\mediatek\common\overlay\sensor\frameworks\base\core\res\res\values\config.xml

# FLASH的相关知识

1、如何兼容新的FLASH

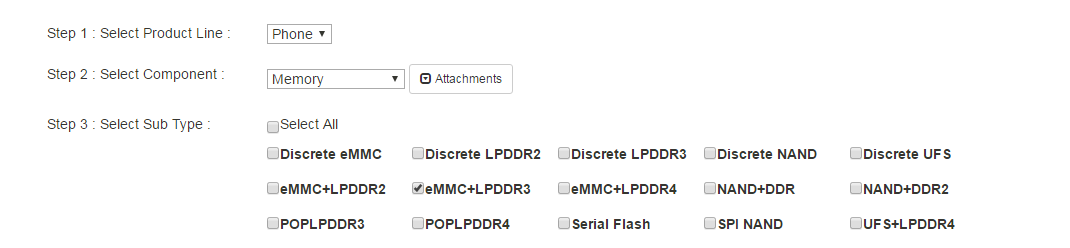
(1)、从MTK的oline里面下载FLASH的配置参数。

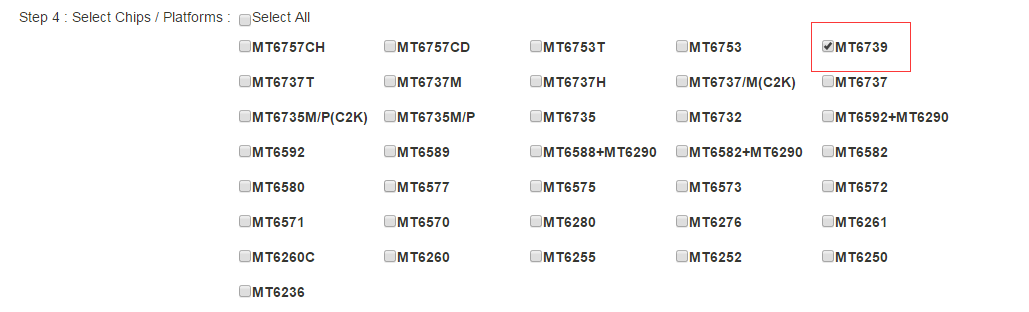
例如配置：

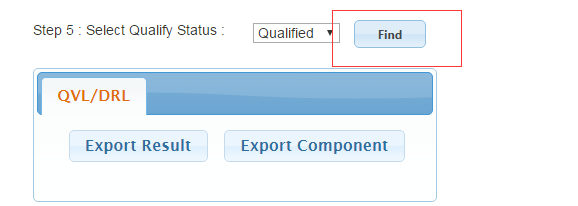
C:\Users\Administrator\AppData\Roaming\feiq\RichOle\37841979.bmp

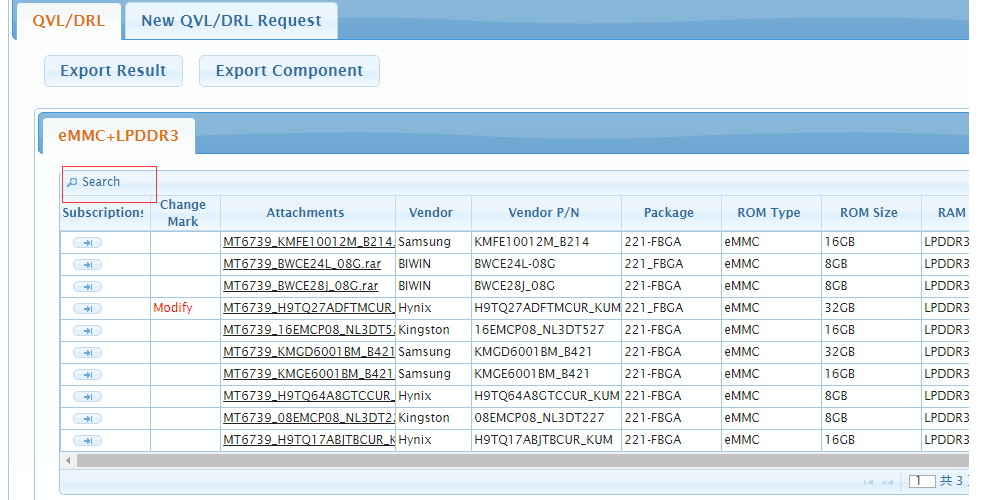
步骤如下：

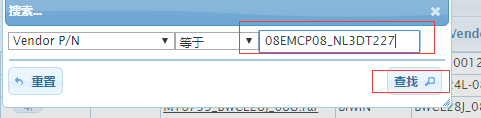


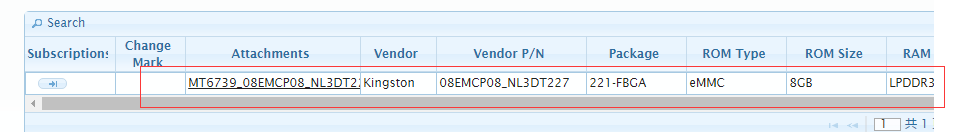


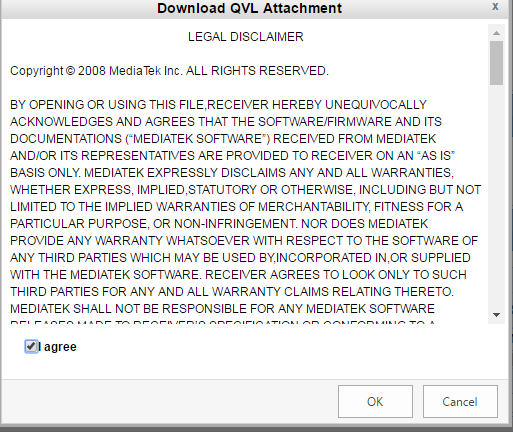


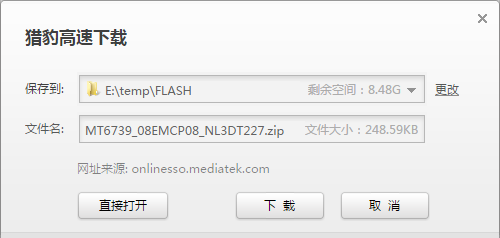






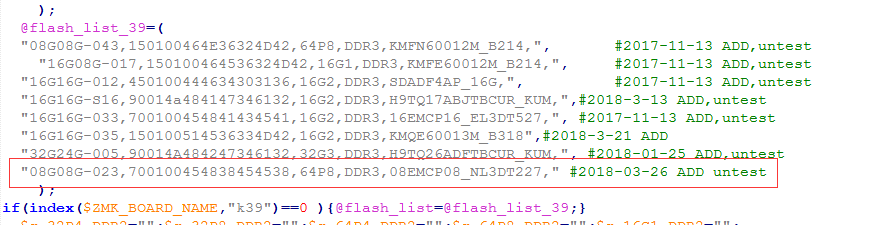






下载下来的文件拷贝到 zcommon\emi 中相应的文件中，其实最终改到的文件是在 Z:\MTK6737\_AND\_6739\_TWO\MT6737\_alps-release-o1.mp1-pre11\alps\vendor\mediatek\proprietary\bootable\bootloader\preloader\tools\emigen\MT6739 目录下。

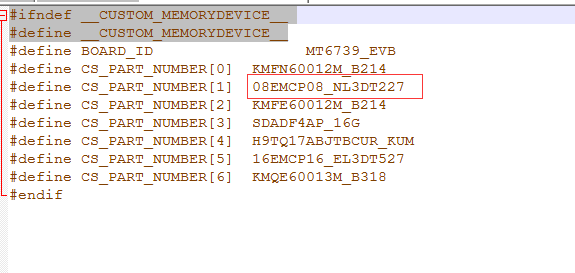
(2)、修改配置脚本 zgencode.pl



（3）、脚本修改的头文件路径：

Z:\MTK6737\_AND\_6739\_TWO\MT6737\_alps-release-o1.mp1-pre11\alps\vendor\mediatek\proprietary\bootable\bootloader\preloader\custom\k39tv1\_bsp\inc\ custom\_MemoryDevice.h

添加里面的宏 ：

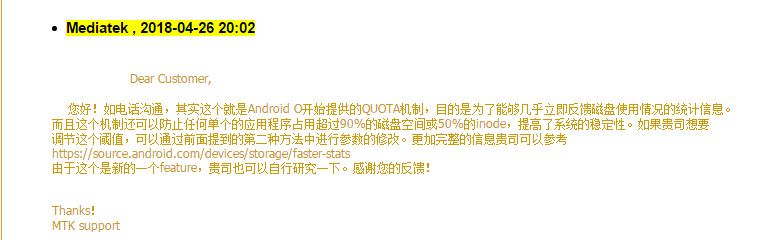


## 2、当存储空间使用到剩余2G的时候出现拷贝不了和相机报内存不足的现象。

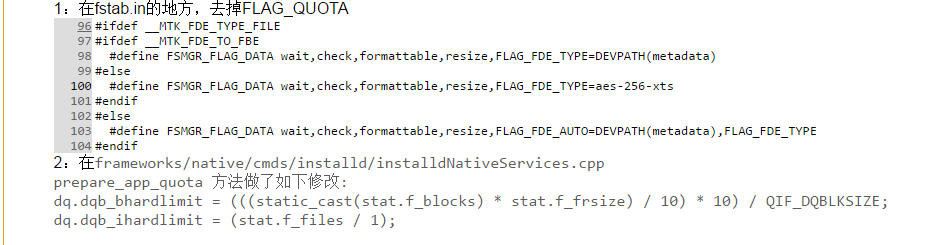
**原因：**由于Android O开始提供的QUOTA机制，会预留一定的空间来保持系统的稳定性。

**如何和客户解释：**

经过验证测试确认这个问题属于QUOTA机制的正常现象。QUOTA机制的作用是：防止任何单个的应用程序使用了全部的资源，导致系统不稳定；QUOTA机制会留有一定的余量来保证系统的稳定运行。 Android O开始引入了QUOTA机制，目前6739的 QUOTA机制 默认保留了 90% 的 blocks 和 50% 的 inodes。以下是MTK的回复：

.

**MTK6739中关于quota：**



**有关好文：**

**linux如何通过quota配置用户的磁盘配额（https://yq.aliyun.com/articles/24214）：**

在生产或工作环境中，你可能会遇到这样的情况：在多人多任务的环境下，有多人共用一个磁盘空间，而有部分人可能向这个共同的空间上放很多或很大的文件，而导致磁盘空间极速减少。这样必然导致其他人的空间会迅速变小。为了让大家有个平等的使用磁盘空间的权限，我们就需要用到磁盘配额。我来限制用户对空间的使用情况。在Linux系统下，用quota来实现此功能。

除了上面的情况外，一般还用在下面的一些场景中，在一些提供web服务的空间中，每个人的网页空间的容量都是受限制的。在邮件服务中，每个人的邮件空间也是受限制。在公司的文件共享服务中，每个人使用的硬盘空间也是需要限制。而针对Linux的主机，如果是多个用户的话，也是需要考虑到用户和用户组的限制。

注意事项：

1、磁盘配额加载于内核中

2、只适用于ETX2/EXT3/EXT4文件系统，对目录是不起作用的哦，这一点要特别注意，虽然挂载点是一个目录，但他实际上是文件系统设备。

3、只针对于户或用户组，但root除外

4、针对用户组的限额，是指某个用户组中的所有成员一起使用的限制而不是每个人所使用的限制

在开始操作前，我们来看看和quota配置相关的内容

inode：限制用户可以建立的文件数量

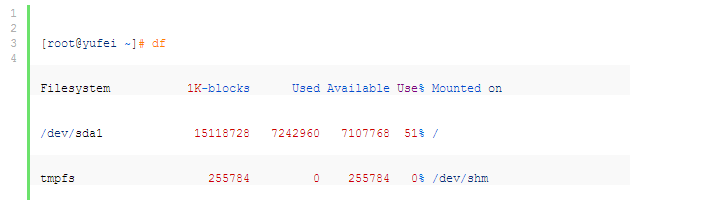
block：限制用户磁盘的容量，以KB为单位

soft：软限制。就是在超过这个值后，会提醒用户，但用户的权限还没有被限制。

hard：硬限制。这是最高的限制，用户或组绝对不会超过这个限制值。如果超过了，就限制死了用户的使用权限了。

grace time：宽限时间。这个就是当达到soft限制时，还没有到达hard限制的时候，用来提醒用户的。如果在警告的天数之内，用户都不进行任何磁盘管理操作，那么soft限制值会即刻取代hard限值来作为quota的限制，就算你没有达到hard限制，也用不了了。

下面开始我们的演示操作：



我的RHEL6系统，因为是一个虚拟机，所以没有太多设置，只有一个根分区，没得选择，我们就对根分区来进行磁盘配额。还有要提醒一点，你的系统中要有至少两个用户，因为磁盘配额对ROOT用户不起作用。

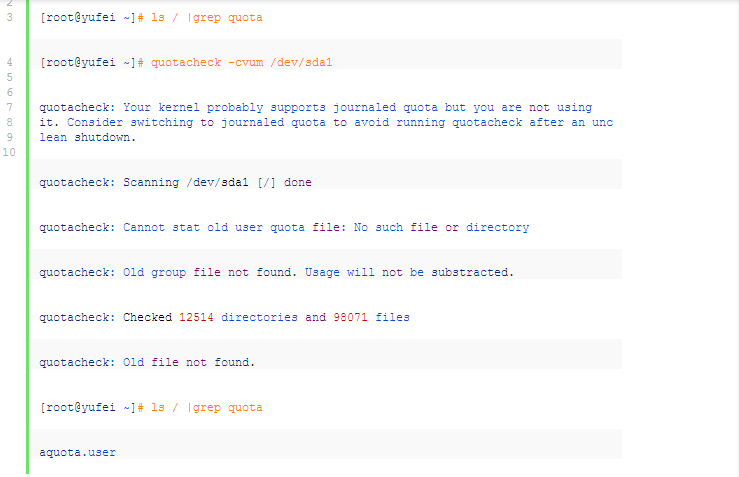
**第一步：重新挂载文件系统或者是修改/etc/fstab文件**

因为重新挂载只针对当前有效果，如果想下次开机也起作用，就需要把配额的参数写入fstab文件中。



现在有了，启用了对用户的配额限制。

**第二步：创建磁盘配额数据库**



看到aquota.user这个用户的配置文件后，那说明我们创建成功了

这里介绍一个几个参数的作用

-c:创建磁盘配额数据库文件

-v:查看创建的过程

-u:创建用户的磁盘配额数据库文件

-g:创建用户组的磁盘配额数据库文件

-a:创建所有磁盘的配额数据库文件，使用此参数的时候，后面就不用接设备了

-m:把以前的磁盘配额信息清除，在对/分区创建的时候，必需用此参数

**第三步：启动磁盘配额**



**第四步：编辑磁盘配额文件**



这主文件中的内容后面的限制作用是什么，我们在上面已经说了，这里就不啰嗦了，这里面已经有的blocks和inodes是指这个用户现在在已经存在的相关文件与大小，以K为单位哦！

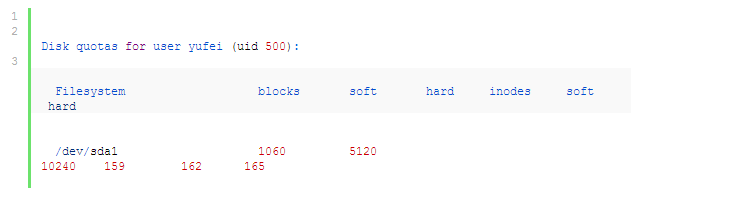
上面显示的内容大概意思就是

用户yufei(uid 500)

在/dev/sda1上所占用的空间是1060K，没有任何的限制，文件数为159,也没有任何的限制。

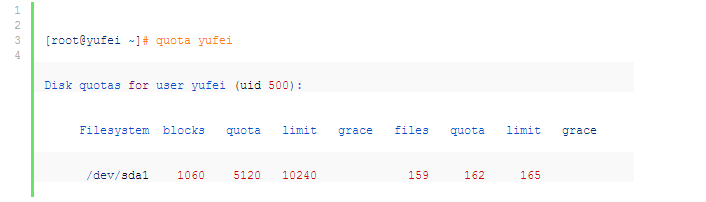
其实上面的编辑方式是调用vi的，所以你可以自由编辑！

我们把限制全部加上。



然后保存退出

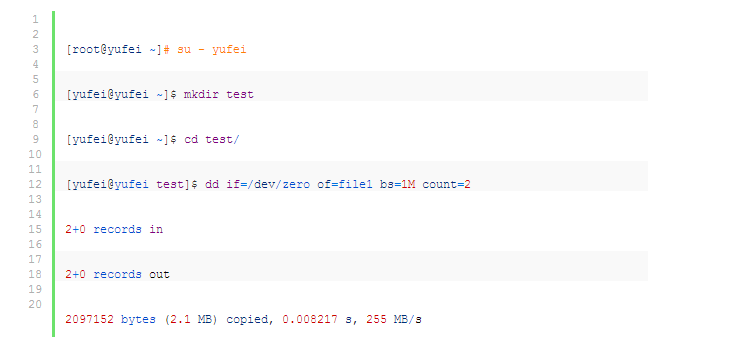
**第五步：查询用户的磁盘配额情况**

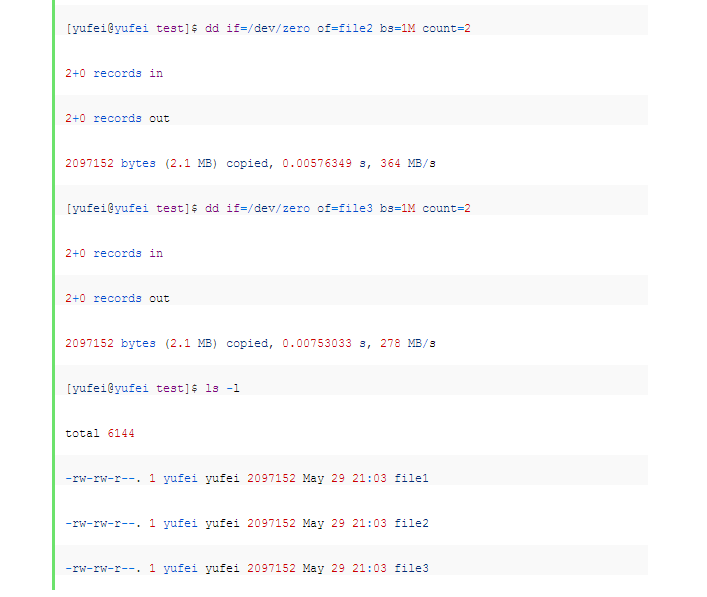


从查询的结果来看，yufei用户有5M的空间，但最多不超过10M。文件数量为162个，但最多不超过165个。

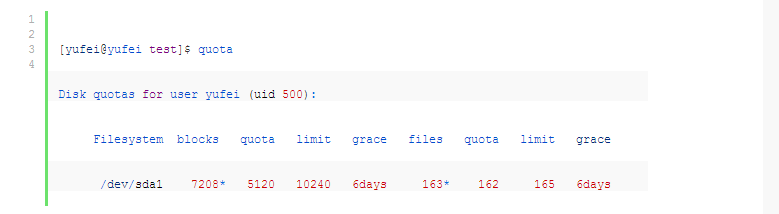
**第六步：测试**

这一步在实际工作没有啦，我们只是为了演示，才做的这一步。当然，如果你想在实际工作中测试的话，也是可以的。



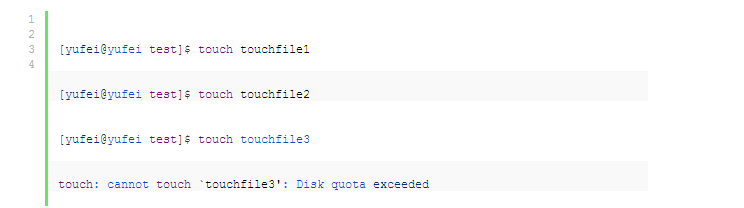


创建了3个2M的文件，外加一个test目录，这时候我们再来看看用户的配额情况



这时候我们看到的信息就和前面的不一样了，超过软限制的，就带上了\*号了。而且宽限时间为6天。在目前的情况下，我最多还能创建2个文件，最多还能有2M多的空间。

下面我们继续来增加文件。



看到没有，当我们再创建第三个文件的时候，提示，超过磁盘配额了。

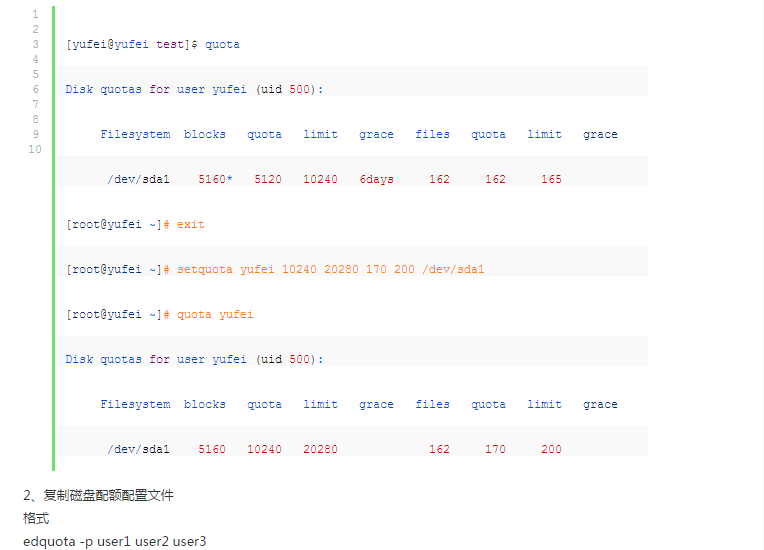


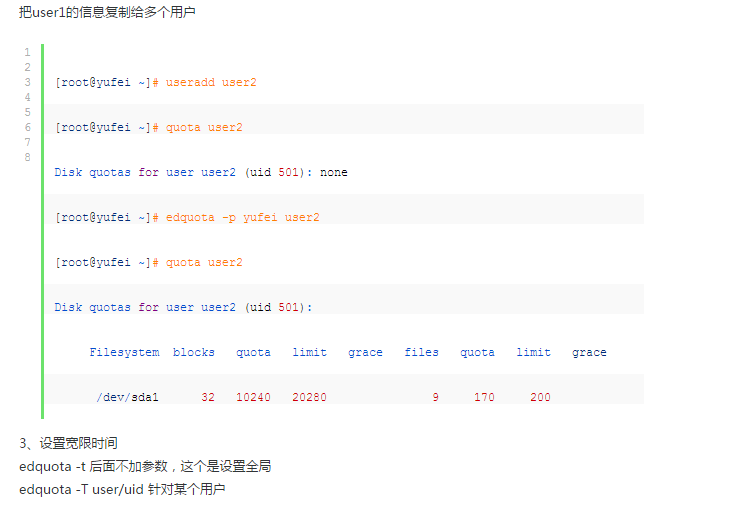


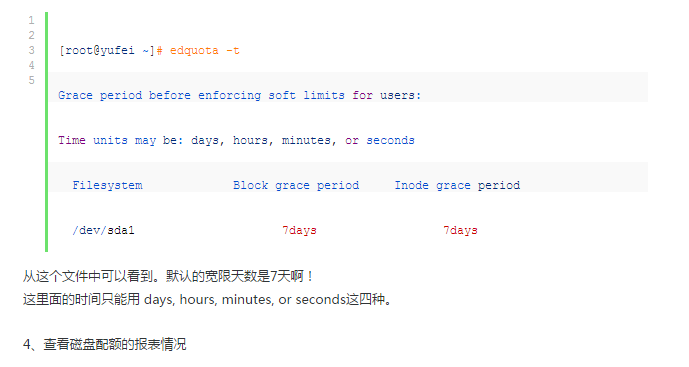




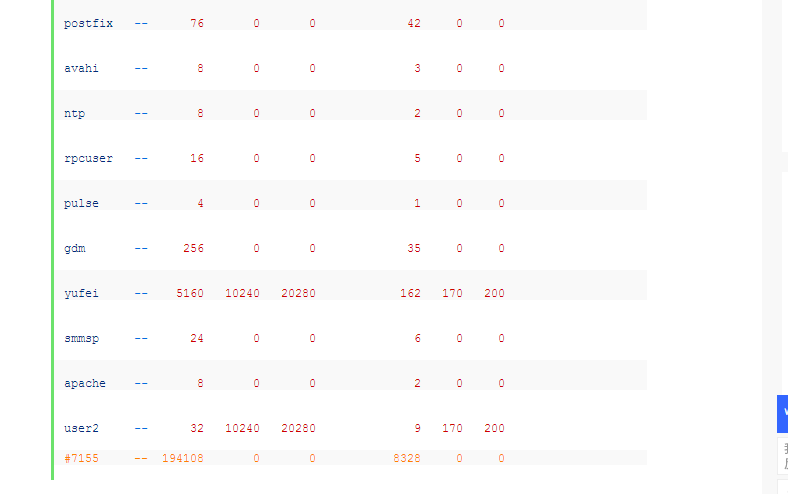
















## 3、MTK6580分离式FLASH和一体式FLASH

区别：分离式flash和二合一flash配置方法是不一样的，分离式的用DLPDDR3\_8G, 二合一的用64P8\_DDR3之类的 **分离式的flash都用DLP开头**

# 振荡器（vibrator）

## 1、工作原理

有两根线（正负极），系统只要给他输出一定的电压就可以振荡起来。

## 2、原理图

## 3、底层实现代码

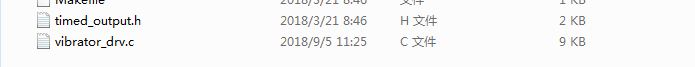
平台：MTK6739

路径：

alps\kernel-4.4\drivers\misc\mediatek\vibrator\mt6739

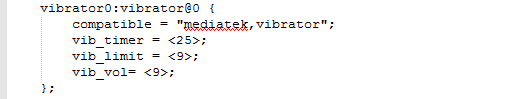


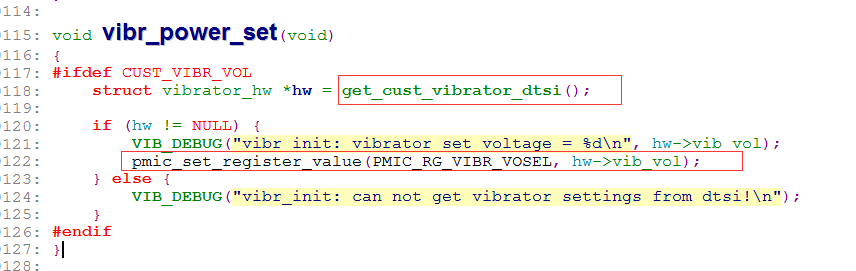
alps\kernel-4.4\drivers\misc\mediatek\vibrato



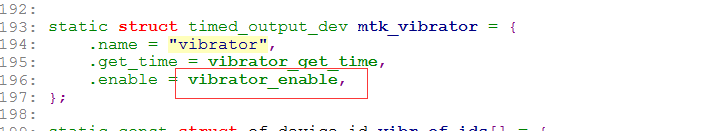
工作大致流程：

1、开机获取DTS中配置的电压，并设置相应的电压



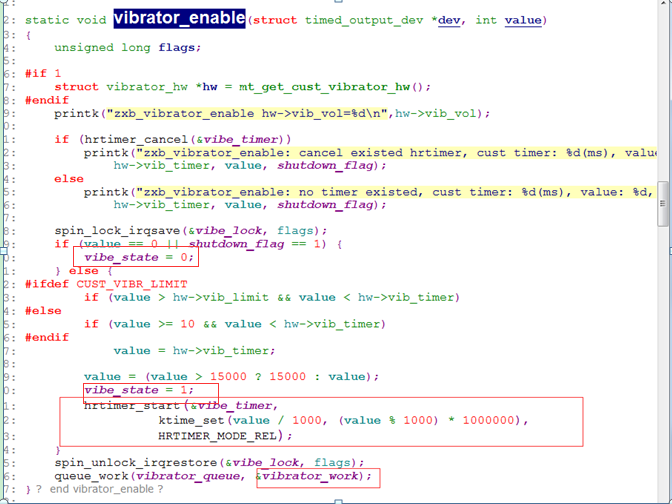


2、通过上层往下调

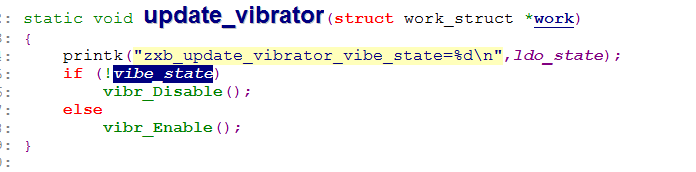


在vibrator\_enable函数中设置时间（由上层传下来）同时设置vibe\_state的值，然后启动工作队列

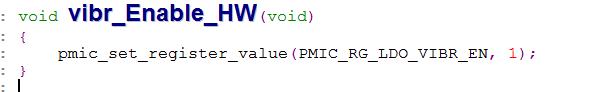




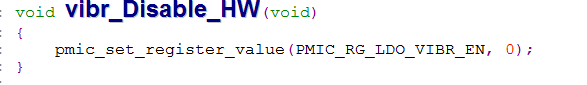
在 update\_vibrator函数中通过判断vibe\_state的值（这个是上一步上层传下来的）调用相应的函数



最终掉到 打开马达



关闭马达



# 温升

## 1、查看温升log的关键字

主板温度：T\_AP

# 功耗问题

## 1、FAQ好文

[FAQ11904] [LTE功耗]MT6290 LTE 待机功耗问题汇总

# 其他

## 1、开关机动画的制作

（1）、首先把jpg图片按数字顺序命名如（zxb001 zxb002… 或者 001 002 003 或者 00001 0002 00003..）

（2）把所有图片放到一个命名为part0的文件夹中。

（3）把最后一张图片放到命名为part1的文件夹中。

（4）创建一个desc.txt文件内容如下：

720 1440 10

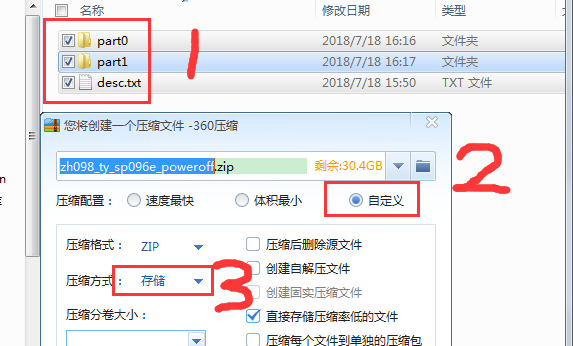
p 1 0 part0

p 0 0 part1

注释：第一行 前两个表示分辨率，第三个表示播放帧率，一般是动画张数除以5。

第二行 和 第三行 一般是固定不需要修改。

（5）选择三个文件及文件夹右击压缩（不能创建一个文件夹吧三个文件放里面然后压缩这个文件夹），选择压缩方式为存储，具体如下：



（6）把压缩好的文件放到zlib/poon的文件中，然后在Config.mk中配置：



（7）测试的时候可以先用adb push到 /system/media文件下面，记得开机动画命名改为：bootanimation.zip ,关机动画改为：shutanimation.zip；然后重启验证（注意关机动画需要按power键关机才可以看到效果，如果用adb reboot是看不到效果的）。

# Config.mk的配置

1、去掉 开机磁盘碎片整理垃圾回收功能

# disable trim

PRODUCT\_PROPERTY\_OVERRIDES += ro.trim.config=true

# 测试模式

## 1、自动化的工厂测试(\*#37\*#)

路径： \MT6737\_alps-release-o1.mp1-pre11\alps\vendor\mediatek\proprietary\factory

**（1）、摄像头工厂模式（factory Mode）测试失败或者预览画面不吐出动态图像**

1，进入factory mode之后，自动测试结果camera 部分fail；

2，进入factory mode之后，进入单项测试，预览画面图像固定，抓main log，发现CRC校验fail；

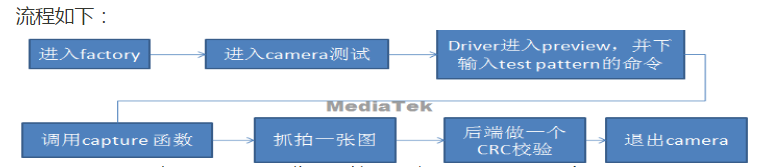
3，进入factory mode之后，进入单项测试，动态预览画面，抓main log，发现CRC校验fail；

以上三种情况都属于不正确的部分，正常的应该是进入自动测试不会failed，进入单项测试，画面为固定的图像（test pattern），1~2s之后退出，抓main log，CRC校验pass

[SOLUTION]

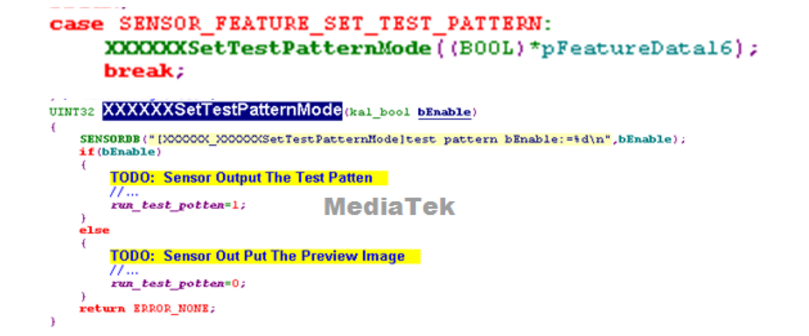
•进入工厂模式测试camera，发现出现的是彩色条纹，纯色图片或者其他的test pattern，然后退出preview。 这个是**正常**的，进入工厂模式后，driver会向sensor下输出test pattern的命令，然后下capture命令调用capture setting，抓拍一张图后退出camera。我们会将抓到的图片（test pattern是固定输出）做校验，来判断硬件是否是ok的

流程如下：



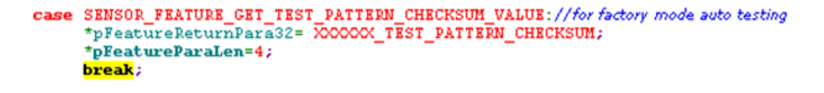
如果sensor吐出来的不是固定图像或者校验失败，需要做如下几个部分的check：

1、在camera 驱动中增加函数：feature\_control case： SENSOR\_FEATURE\_SET\_TEST\_PATTERN：代码如下：

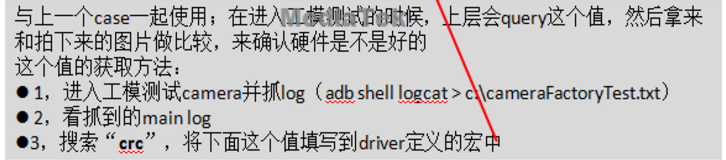


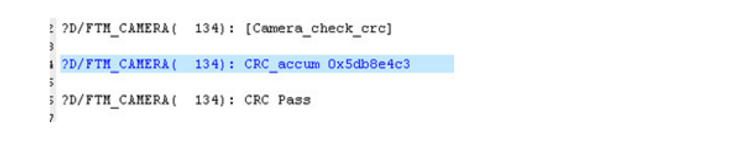


2、在camera 驱动中增加 函数：feature\_control case：SENSOR\_FEATURE\_GET\_TEST\_PATTERN\_CHECKSUM\_VALUE









**注意：**

1，获取CHECKSUM\_VALUE这个值的时候，最好在工模下多试几次，因为有些sensor吐出来的固定图像的数据实际上是变化的，这个时候，每次输出的CRC\_Accum值就会不一样，此时要找FAE确认。

2，如果之前已经确认是ok的，但是又修改了驱动中的HV Mirror或者GrabWindow，Startx，starty等参数，会导致CheckSum这个值又和之前的不一致，请注意重新按照上面的方法k这个参数

3，有时候出现每次checksum值都不一致，是因为掉线引起的，这个时候需要修改驱动那边的驱动能力，PCLK的相位等来解决掉线问题之后再来测试

PS：掉线是指当我们去抓data的时候，假设是在PCLK的上升沿去抓取data，这个时候由于PCLk的驱动能力不足或者太大，导致去抓data的时候，数据还没有稳定或者已经在变化了，此时抓到的data就不正确，通常表现为抓到的都是0，呈现出来的是绿条纹，也有可能是PCLK和data之间的相位没有匹配，导致波形是ok的，可是抓取data的时机也不对，导致的掉线问题。解决方法一般有：改善MCLK的驱动能力，改善PCLK的驱动能力，调整PCLk的相位，降低PCLK的帧率等~

4，如果发现不同的模组出的CRC\_Accum的值不一样，还需要check sensor driver的otp\_en的寄存器位，并将其关掉。

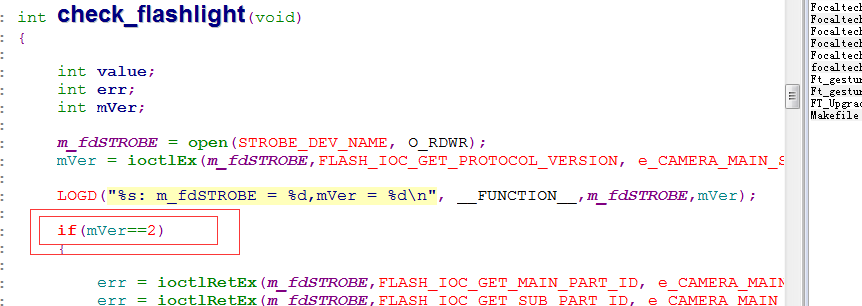
**（2）、LED测试项改为闪光灯测试**

更改过的例子：

MT6739 : 

注意点：

1、



这里面的这个2可能不同的大版本不一样。

2、

#define STROBE\_DEV\_NAME "/dev/flashlight" //这个闪光灯的路径根据具体的大版本来确定。

## 2、关机工厂测试(关机状态按音量键下键加开机键)

进入方法：按音量键的下键启动

### （1）、TP测试里面没有CTP测试

原因：由于系统误以为你这个TP是电阻屏

修改：每个TP驱动在运行的时候必须上报自己是电阻TP还是电容TP：

加在TP的init中即可：

static int tpd\_local\_init(void)

{

。。。

。。。

tpd\_type\_cap = 1;

return TPD\_OK;

}

### （2）、添加工厂测试模式的测试项（以下已添加指纹工厂测试为列）

主要修改的文件有：

1、

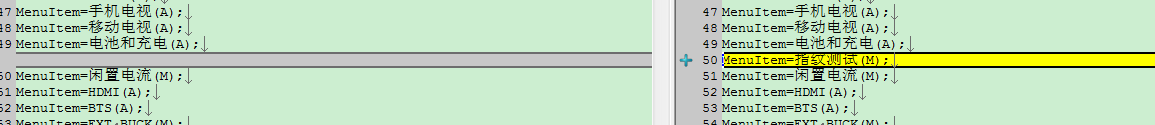


——》这两个文件主要的作用是在指纹驱动中做一些相关指纹状态的判断，然后把判断的结果返回给上层显示。（关机工厂测试里面有一些测试项是这样实现，但并不是一定的。）

2、



——》这两个文文件里面主要是在测试界面添加测试项，只有这里的名字和测试项里面的名字匹配，才会显示测试项，所以这里很关键。具体修改如下：（里面的A：表示把自测项加在自动测试的菜单里面；M:代表把测试项加到手动测试的菜单里面。）



这里的前后顺序和显示的顺序是一致的。

3、



——》这个文件里面主要开启测试项的宏，脚本开关宏也是这里改的。具体如下：



4、



——》添加编译脚本，具体如下：

TEST\_SRC\_FILES := \



5、



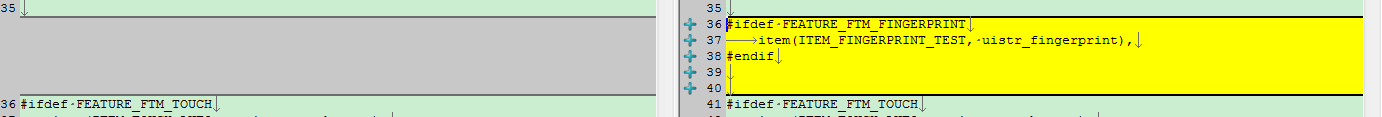
——》这两个是添加测试项翻译，具体如下：

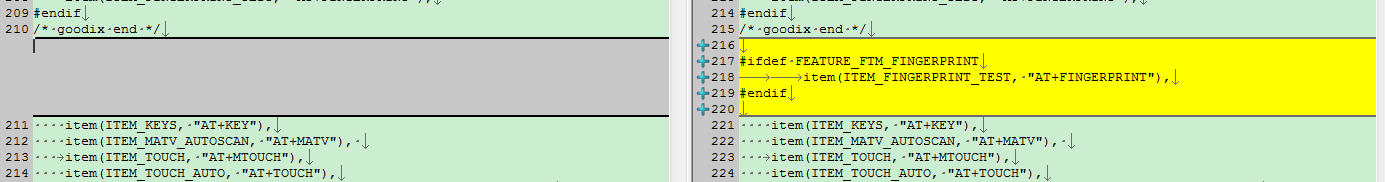


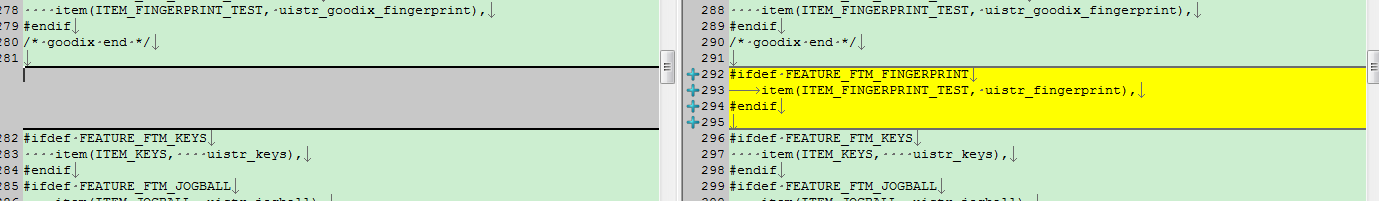
6、



——》添加item目前还没有细看，具体如下（3处）：







7、

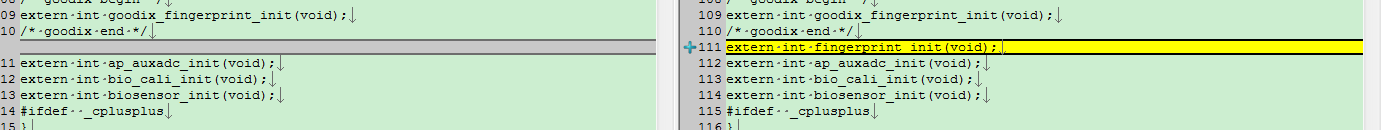


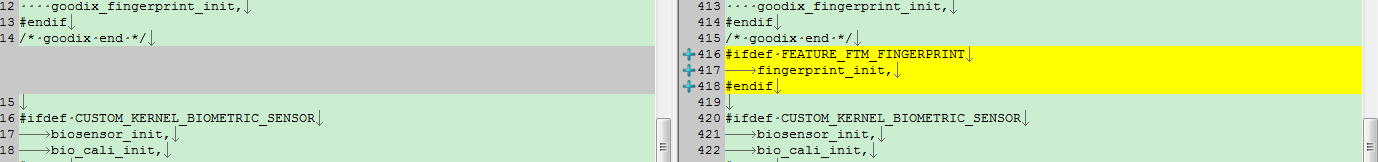
——》关于指纹的关键测试文件，具体的测试逻辑都是在这里面，每个测试项都有自己的一个文件，具体太多就不展示。

8、



——》第7步初始化的入口在这里添加，具体如下：

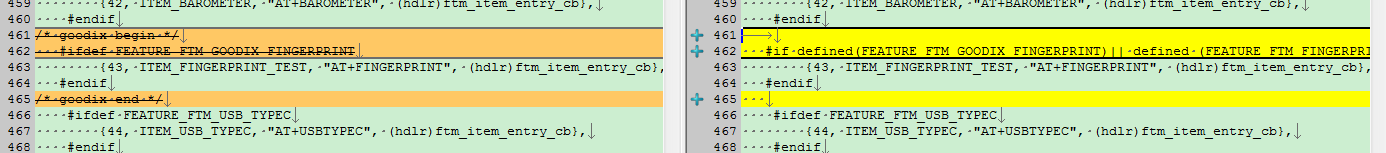




9、



——》添加测试ID，具体如下：



10、指纹的全部入口。

\alps\vendor\mediatek\proprietary\factory\src\ factory.cpp

## 3、工程测试模式（\*#9981# \*#\*#7548135#\*#\*）

## 4、老化测试（\*#\*#1234#\*#\*）

# 管脚配置

## 1、DCT配置

### 平台：MT6580 MT6737 MT6739

工具路径：alps\vendor\mediatek\proprietary\scripts\dct //其中6737和3739已经把工具拉到了zprj中。

.dws文件路径：{F\_KERNEL}/drivers/misc/mediatek/dws/${gPLATFORM\_NAME}/${ZMK\_BOARD\_NAME}.dws //其已经被拉到 BOARD\_CFG 中。

例如：alps\kernel-4.4\drivers\misc\mediatek\dws\mt6739/ k39tv1\_bsp\_1g.dws

## 2、DTS配置

### 平台：MT6580

# 调试相关知识

## 1、串口tx不在屏蔽盖里的项目

### 平台：MT6580

zh286 zh096

## 2、adb打印log

### 平台：MT6580、MT6737、MT6739

执行命令：adb pull storage/emulated/0/mtklog log/aaa

## 3、adb开启开机后不吐的串口log

### 平台：MT6580、MT6737、MT6739

MTK：开启log ：adb shell setprop persist.uartconsole.enable 1

## 4、user和userdebug版本开log

如果是MTK AndroidM以上的版本的话，因为userdebug和user版本默认开启了system verified boot，所以需要执行

1 adb root

2 adb disable-verity

3 adb reboot

4 adb remount

# 编译下载相关知识

## 1、全编：

### 平台：MT6580

source zmake 项目目录名 后面可以跟 user 或者 eng

## 2、单步编译

### 平台：MT6580

1）、拷贝项目到大版本中：

./zgencode.pl zprj/ZH286/JM/DRVONLY/ debug flase r //其中的flase表示编译出来是否有fota

./zgencode.pl zprj/ZH286/JM/DRVONLY/ user flase r

2）、编译pl

source zmkpl

3）、编译lk

Source zmklk

4）、编译boot

source zmkbootimage

5）、编译system

source zmk

## 3、开机向导wifi界面无法跳过

### 平台：MT6580

现象：开机向导到wifi的时候没有skip按钮，导致一定要连接wifi。

原因：当前版本开启了FRP(Factory Reset Protection)功能，download only方式下载，被Google FRP判定为untrusted reset，这样导致的结果就是在恢复出厂设置开机选择WIFI的界面，不可以skip，必须选择并连接WIFI，连接网络之后，会跳出Verify your account的界面，必须输入之前登录这支手机的Google账号密码。

账号密码验证通过，只是证明机主身份，接着还要重新输入新的Google账号密码（可以和原来的一样），进而进入手机launcher。

解决：下载的时候使用 Format ALL + Download 即可。

## 4、下载报错汇总

### 平台：MT6580 MT6739 MT6737

(1)



原因 ：软件没有兼容相应的FLASH 导致读FLASH校验信息出错。

（2）



原因：对应的选项中选择的文件是不对的，后者文件的命名不对。

（4）刷机过了红条，到了紫色条卡住。（错误代码4008）

这种情况出现的话，大家可以把电池拿下来，然后重新安装上，进入REC后选择关机。然后重新刷。

（5）驱动安装好了，一点刷机就弹出错误提示。

这种情况一般是USB口的供电问题，使用PC后面的USB插口，或者拔掉几个用电量大的设备。

（6）正常操/作情况下，和步骤2出现的问题提示不一样的。（典型的3013问题，出现率最高的）

这个问题就是大家经常问的，出现这个问题基本上都是因为大家升级到了4.1.2然后想刷回4.0.4的情况，大家使用4.1.2刷机时候用的SP\_Flash\_Tool重新刷4.0.4即可，或者用甲鱼官方4.1.2的SP\_Flash\_Tool。

# ADB使用

## 1、userdebug版本执行adb remount失败

[DESCRIPTION]

android 6.0 以后版本默认会打开system verified boot，即在userdebug和user版本会把system映射到dm-0设备，然后再挂载。挂载前会检查system分区数据完整性，如果system分区被恶意修改了则不允许挂载system。

[SOLUTION]

userdebug版本如果需要remount system分区来push文件debug，不需要重新编译版本disable dm-verity，只需要执行以下adb命令即可。

adb root

adb disable-verity

adb reboot

重新启动后再执行:

adb remount即可把system分区remount成rw。

贵司向system分区push文件后，请不要再adb enable-verity，否则就会无法开机，因此push文件后，system分区数据就发生了变化。

adb disable-verity/enable-verity 命令只能在userdebug模式下使用。user版本不支持关闭dm-verity。

如果您的adb不支持adb disable-verity命令，请更新android sdk platform-tools到最新版本。或直接到以下的link下载最新版的独立adb tool。

http://forum.xda-developers.com/showthread.php?t=2317790

其他相关信息请参考https://source.android.com/security/verifiedboot/index.html

## 2、查看GPIO口状态

### 平台：6739

cat /sys/devices/platform/1000b000.pinctrl/mt\_gpio