本文档旨在整理工作学习中遇到的新的知识点和高频率遇到的旧的知识总结。

整理的过程并不是造轮子的过程，目的是归纳总结。

在遇到新问题时，应该从哪几个角度去分析处理问题。

以及如何快速掌握新的驱动。

归纳旧的问题的应用场景，整理出新知识的学习方法。

学习并不是漫步目的的发散学习，归纳出新知识点的掌握标准，并快速的理解应用。

**当你去学习和理解一个模块时，**

**需要看弄每一行代码吗？**

**需要理解结构体的每一个成员吗？**

**掌握模块的初始化和调用逻辑，以及使用场景**

**初阶掌握的标准**

模块的概念是什么

模块的调用流程是什么（什么时候open/close, resume/suspend，谁去调用的这些接口）

根据被调用的部分去关心对应的初始化流程

学习建议：

建议有余力去锻炼：

Document/ 目录下的文档

内核中函数头部的注释。

快速的方法：

搜索内核中相关的函数的使用，查看实例去学习

**加油**

[BSP driver 4](#_Toc97562122)

[1.1 Device Tree 5](#_Toc97562123)

[1.2 lcd 8](#_Toc97562124)

[lcd 8](#_Toc97562125)

[display 8](#_Toc97562126)

[rotation 8](#_Toc97562127)

[bootAnimation 8](#_Toc97562128)

[backlight 9](#_Toc97562129)

[brightness 9](#_Toc97562130)

[lcd notify 9](#_Toc97562131)

[Import 2.8” panel 9](#_Toc97562132)

[1.3 led 12](#_Toc97562133)

[led 12](#_Toc97562134)

[light 12](#_Toc97562135)

[LED out of control 12](#_Toc97562136)

[1.4 gpio 13](#_Toc97562137)

[gpio 13](#_Toc97562138)

[设备树如何管理gpio function 13](#_Toc97562139)

[GPIO driver 14](#_Toc97562140)

[gpio 推挽输出，开漏输出 15](#_Toc97562141)

[gpio driver strength 15](#_Toc97562142)

[gpio request 23](#_Toc97562143)

[pinctrl 23](#_Toc97562144)

[gpio irq 23](#_Toc97562145)

[1.5 i2c & spi 24](#_Toc97562146)

[I2C 24](#_Toc97562147)

[1.6 内核函数解析 25](#_Toc97562148)

[notifier chain 25](#_Toc97562149)

[list head 25](#_Toc97562150)

[work queue 25](#_Toc97562151)

[1.7 vibrator 26](#_Toc97562152)

[driver no work? 26](#_Toc97562153)

[1.8 video 27](#_Toc97562154)

[vidc fail to suspend 27](#_Toc97562155)

[1.9 usb 29](#_Toc97562156)

[usb数据包解析 29](#_Toc97562157)

[USB枚举过程 32](#_Toc97562158)

[USB Connectors and Cables 33](#_Toc97562159)

[usb2.0 OTG与connector block 34](#_Toc97562160)

[如何区分USB2.0和USB3.0 35](#_Toc97562161)

[USB 枚举的过程以及BC协议 38](#_Toc97562162)

[USB 眼图测试 39](#_Toc97562163)

[diag port enable 40](#_Toc97562164)

[usb host start流程 41](#_Toc97562165)

[usb extcon 42](#_Toc97562166)

[USB Ethernet test 43](#_Toc97562167)

[1.10 extcon driver 44](#_Toc97562168)

[1.11 PMU 45](#_Toc97562169)

[PMU typec-or-rid-detect-change 45](#_Toc97562170)

[PMU 交接 46](#_Toc97562171)

[1.12 battery 48](#_Toc97562172)

[根据一个节点去追函数 48](#_Toc97562173)

[1.13 pm\_runtime 49](#_Toc97562174)

[1.14 struct device/device tree 50](#_Toc97562175)

[devm架构 50](#_Toc97562176)

[of\_node/fwnode 50](#_Toc97562177)

[创建文件节点的几种方法 51](#_Toc97562178)

[1.15 buzzer & gp\_clk 52](#_Toc97562179)

[Analog PWM to driver buzzer 52](#_Toc97562180)

[GP\_CLK 52](#_Toc97562181)

[代码是如何被编译/执行的 54](#_Toc97562182)

[Makefile 54](#_Toc97562183)

[makefile函数 54](#_Toc97562184)

[Makefile, Kconfig, .config, obj的编译关系 54](#_Toc97562185)

[init进程 57](#_Toc97562186)

[property 57](#_Toc97562187)

[device目录 59](#_Toc97562188)

[overlay机制 59](#_Toc97562189)

[PRODUCT\_COPY\_FILE 59](#_Toc97562190)

[config.xml 59](#_Toc97562191)

[Android框架和java 语法 63](#_Toc97562192)

[Intent 63](#_Toc97562193)

[android 四大组件 65](#_Toc97562194)

[ ISSUE 66](#_Toc97562195)

[1.1 property get fail? 67](#_Toc97562196)

[1.2 如何确认一个文件是否被编译? 68](#_Toc97562197)

[1.3 如何检查一个driver的加载时间? 70](#_Toc97562198)

[节点权限问题总结 73](#_Toc97562199)

[NavigationBarPosition 75](#_Toc97562200)

[LCD driver ic(ILI9881C) 75](#_Toc97562201)

[auto brightness自动亮度调节 75](#_Toc97562202)

[GPIO 75](#_Toc97562203)

[list of URL 76](#_Toc97562204)

# BSP driver

[Linux内核专题 - 介绍\_beckdon的专栏-CSDN博客\_extcon](https://blog.csdn.net/beckdon/article/details/50424557)

<https://www.kernel.org/doc/html/latest/index.html>

## Device Tree

Device Tree是一个整体性的内容

建议仔细阅读

[Device Tree（二）：基本概念](http://www.wowotech.net/device_model/dt_basic_concept.html)

如何希望硬核了解device相关代码分析，以及与bootloader之间的交互,可以阅读:

[Device Tree（三）：代码分析](http://www.wowotech.net/linux_kenrel/dt-code-analysis.html)

对设备树常见问题的整理：

Property中对节点的引用建议去看parent node.

1. #cell的使用

#address-cells, #size-cells

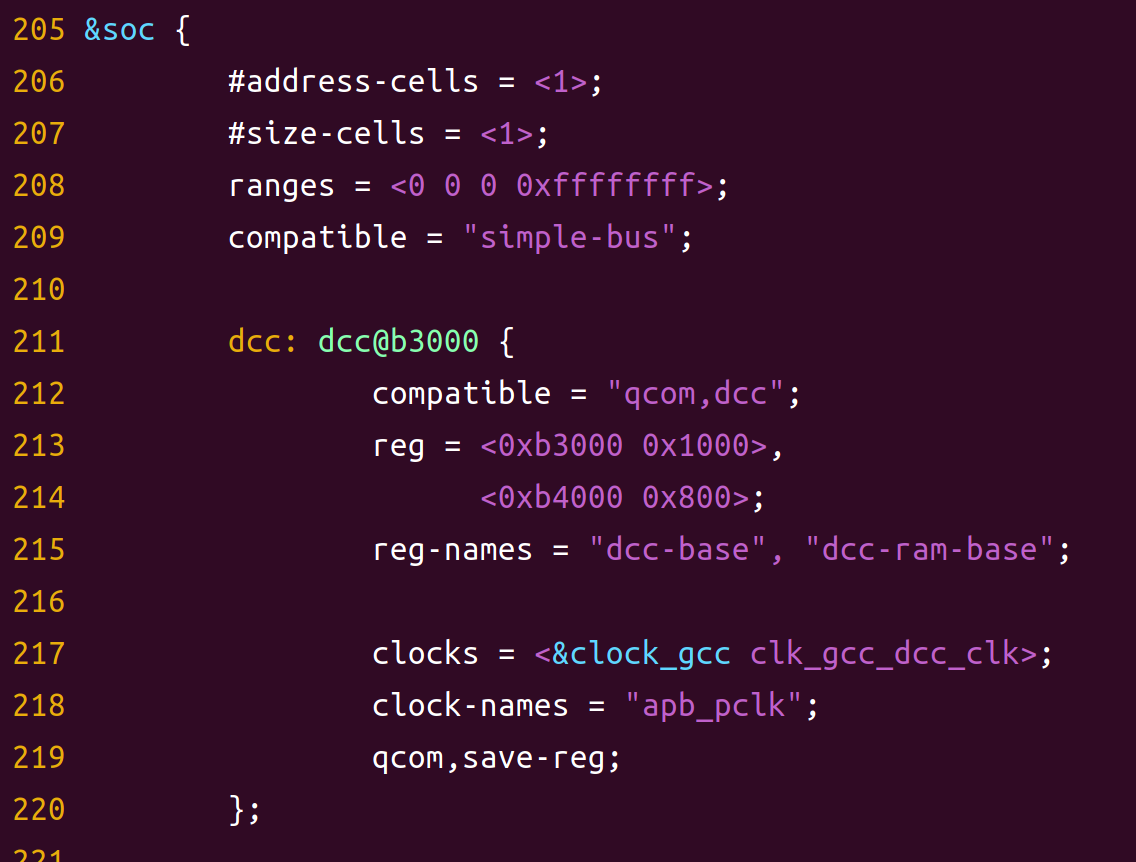
子节点(sub node)有寻址需求(需要定义reg property)，父节点需要定义这两个属性.

“#”是number的意思

#address-cells表示需要用多少个u32 cell来描述地址域

#size-cells表示…大小

eg:



疑问：

64位CPU为什么#address-cells 和 #size-cells不是2？< >表示数据类型最大是u32

对地址学习时可以再研究。

#interrupt-cells



标准结构

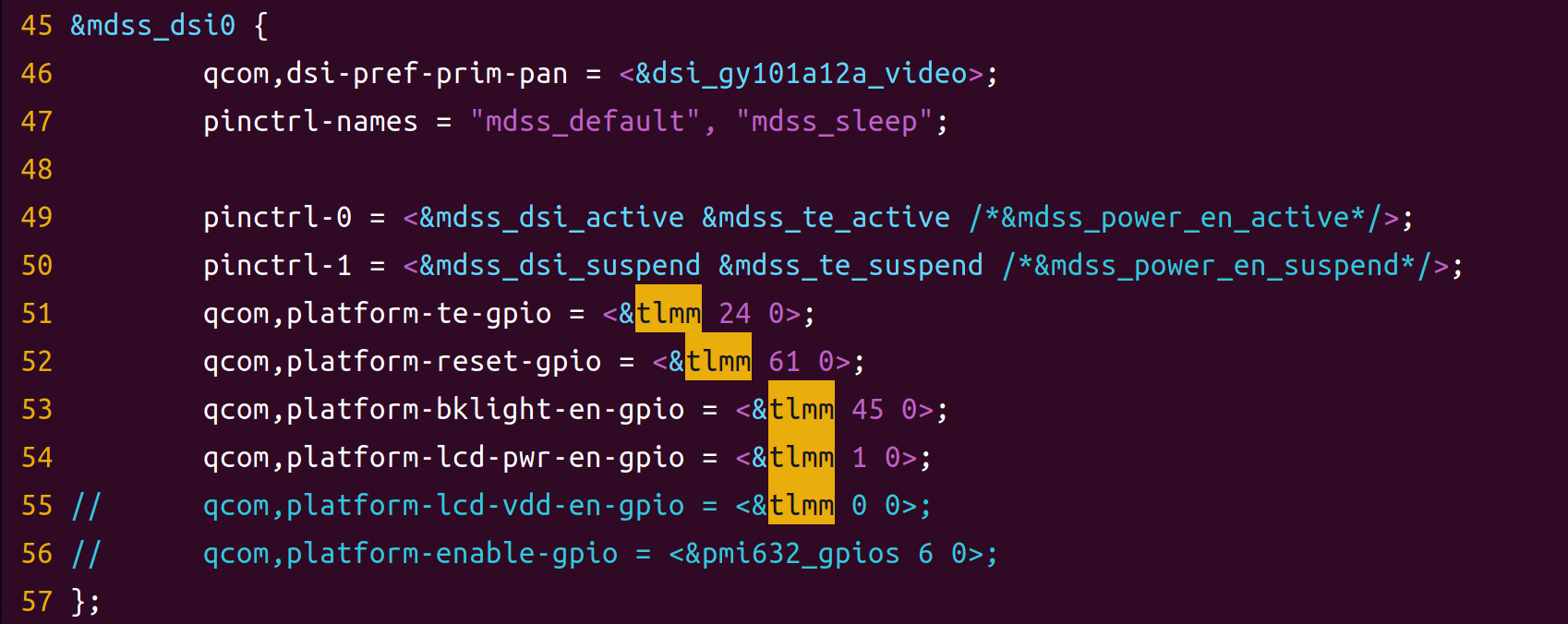
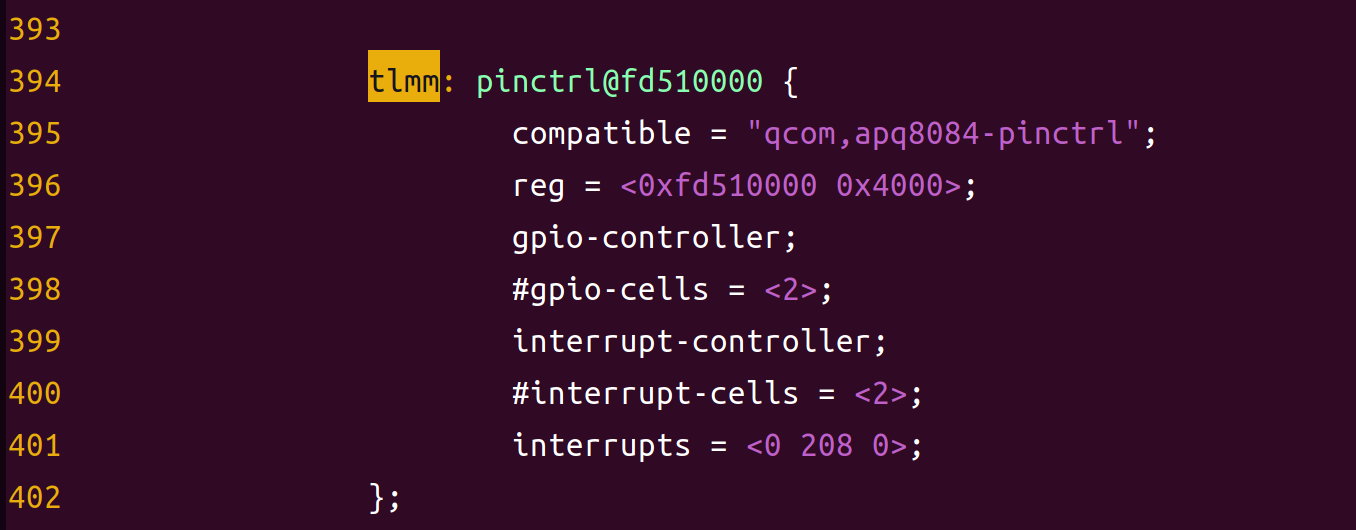
interrupr-parent //没有，属性跟随partent node的interrupt-parent

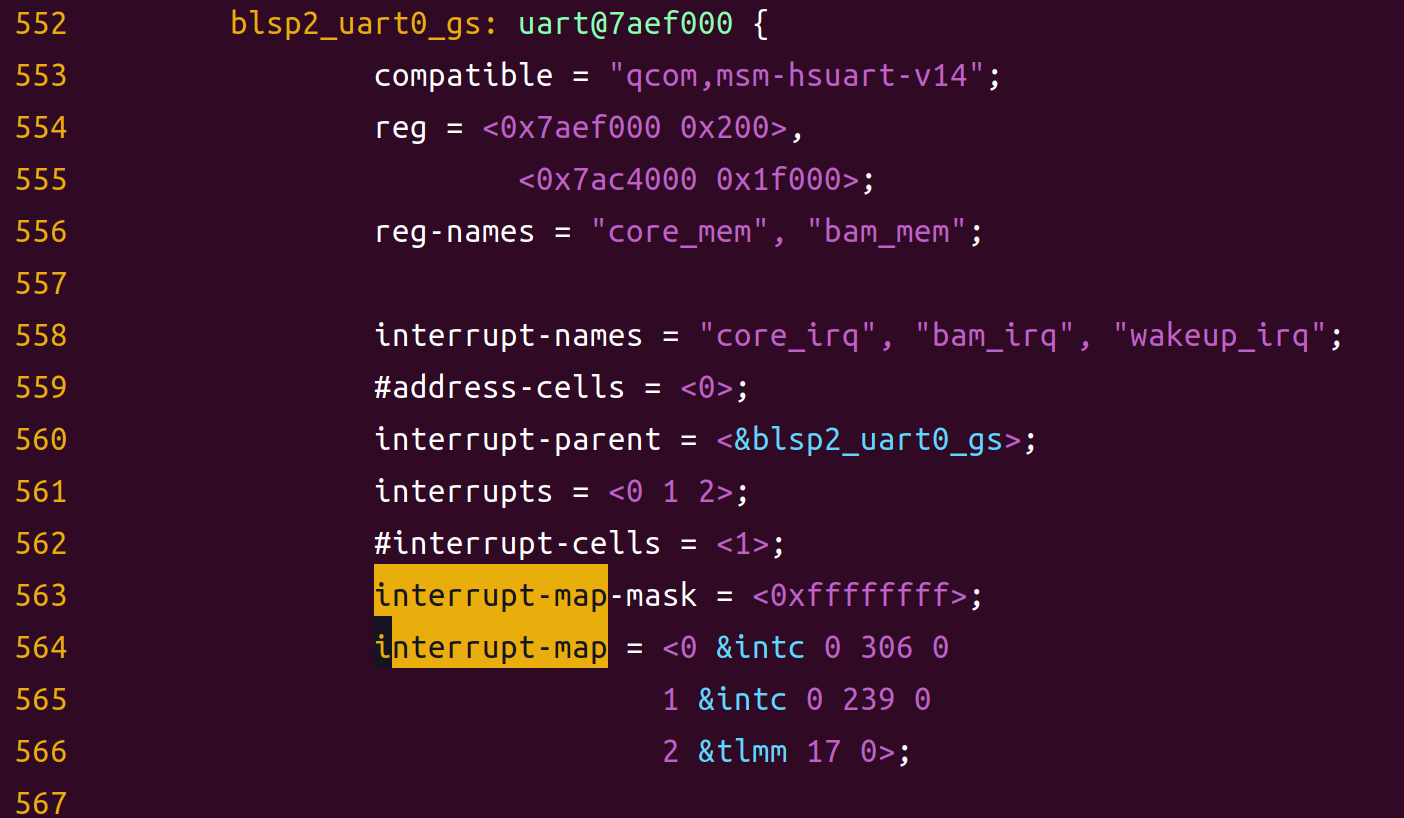
interrupts

interrupt-names

#gpio-cells

#cells的另一种引用方法





这里有一个问题：

gpio和interrupt-map都是同一种写法，怎么确定property依赖哪一个cells?

1. 反编译设备树

反编译andriod dtbo.img 为dtsi文件

mkdtimg dump dtbo.img -b <filename>

dtc -I dtb -O dts <dtb\_filename> -o <dts\_filename>

dtc在源码目录下scripts/dtc/dtc中

没有dtbo.img可以在项目目录中找dtb文件

1. 设备树在sys file 的位置

## lcd

### lcd

[msm8953平台LCD亮灭屏流程和LCD知识点总结\_移动开发\_jinron10的专栏-CSDN博客](https://blog.csdn.net/jinron10/article/details/103419337/?utm_medium=distribute.pc_relevant.none-task-blog-baidujs-2)

[高通MSM8916平台显示框架及lcd驱动概述\_lane](https://www.sohu.com/a/282532614_99956377)

[高通内核显示驱动MDSS解析(转) - Evsio0n](https://evsio0n.com/archives/286/)

[MIPI-DSI三种VideoMode理解\_Eliot\_shao的专栏-CSDN博客\_videoburstmode](https://blog.csdn.net/eliot_shao/article/details/52474348)

[驱动之LCD\_bigman\_123的专栏-CSDN博客\_](https://blog.csdn.net/bigman_123/article/details/19070579)

[android9.0上，实现双mipi屏\_xuhui\_7810的专栏-CSDN博客](https://blog.csdn.net/xuhui_7810/article/details/89213593)

[高通 android平台LCD驱动分析 - LoongEmbedded - 博客园](https://www.cnblogs.com/LoongEmbedded/p/5298269.html)

[高通平台 lcd driver 调试小结\_guoguo295的专栏-CSDN博客\_xdpi ydpi 高通](https://blog.csdn.net/guoguo295/article/details/38555551?utm_medium=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-9.add_param_isCf&depth_1-utm_source=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-9.add_param_isCf)

[MIPI\_DSI协议简要介绍\_caihaitao2000的博客-CSDN博客](https://blog.csdn.net/caihaitao2000/article/details/80224976)

[高通SDM450+android9.0读显示屏ID\_LoongEmbedded的专栏-CSDN博客](https://loongembedded.blog.csdn.net/article/details/114988994#comments_15518445)

### display

[Android系统启动流程分析\_麦田-CSDN博客\_android启动流程 sbin](https://blog.csdn.net/galensphang/article/details/13631929?utm_medium=distribute.pc_relevant.none-task-blog-baidujs-6#t18)

[android LCM启动流程——LK\_迟子涵925的博客-CSDN博客](https://blog.csdn.net/u010787514/article/details/82632684#t0)

[我所理解的高通平台Lcd驱动框架\_Croxd的博客-CSDN博客](https://blog.csdn.net/weixin_42432281/article/details/113246959)

[高通平台lcm 总结\_yctcgogo的专栏-CSDN博客](https://blog.csdn.net/yctcgogo/article/details/82655903)

### rotation

[Android 7.1 竖屏转横屏全过程实现-基于高通平台\_Eliot\_shao的专栏-CSDN博客](https://blog.csdn.net/eliot_shao/article/details/70766283?utm_medium=distribute.pc_relevant.none-task-blog-title-1&spm=1001.2101.3001.4242)

[Android9.0 MTK 平板横屏方案修改(强制app横屏 + 开机logo/动画+关机充电横屏 + RecoveryUI 横屏)\_cczheng-CSDN博客](https://blog.csdn.net/u012932409/article/details/102585974?utm_medium=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-2.add_param_isCf&depth_1-utm_source=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-2.add_param_isCf)

[【转】高通平台android9.0设置开机默认横屏显示 - 简书](https://www.jianshu.com/p/e44512de8c53)

[Android 7.1 竖屏转横屏全过程实现-基于高通平台\_Eliot\_shao的专栏-CSDN博客\_高通横屏竖用](https://blog.csdn.net/seek_0380/article/details/70766283?utm_source=blogxgwz8)

### bootAnimation

[android系统知识（8.0）---Android O 开关机动画流程\_zhangbijun1230的专栏-CSDN博客](https://blog.csdn.net/zhangbijun1230/article/details/79735530)

[MSM8909开机logo显示(1)---LCD背光的控制\_移动开发\_涛声依旧的专栏-CSDN博客](https://blog.csdn.net/armfpga123/article/details/79548827)

[Android开机动画总结\_卓越之路-CSDN博客](https://blog.csdn.net/mcsbary/article/details/89192508)

[android开机动画启动流程 - JavaShuo](http://www.javashuo.com/article/p-ajhxjlrl-hy.html)

[Android bootanim开机动画启动流程 - JavaShuo](http://www.javashuo.com/article/p-zhbzspum-co.html)

[Android开机动画的显示(一)\_SwallowJoe的博客-CSDN博客](https://blog.csdn.net/u014535072/article/details/82916548#t26)

[Android系统的开机画面显示过程分析\_thinkinwm的专栏-CSDN博客](https://blog.csdn.net/thinkinwm/article/details/17963781)

### backlight

[msm8953背光\_jinron10的专栏-CSDN博客\_msm8953的mpp1](https://blog.csdn.net/jinron10/article/details/104710753/?utm_medium=distribute.pc_relevant.none-task-blog-baidulandingword-6&spm=1001.2101.3001.4242)

[高通LCD的pwm背光驱动 - yooooooo - 博客园](https://www.cnblogs.com/linhaostudy/p/9173117.html#autoid-1-0-0)

[android p 解锁 忽然亮屏后 恢复亮度 - soul丶嘟 - 博客园](https://www.cnblogs.com/chjgongzuo/p/10728402.html)

[android 9.0关于屏幕亮度的整理\_yyqx1128\_wuting的博客-CSDN博客](https://blog.csdn.net/yyqx1128_wuting/article/details/107784760?utm_medium=distribute.pc_relevant_right.none-task-blog-BlogCommendFromMachineLearnPai2-6.nonecase&depth_1-utm_source=distribute.pc_relevant_right.none-task-blog-BlogCommendFromMachineLearnPai2-6.nonecase#t1)

### brightness

[Android frameworks去掉熄屏前先变暗的功能\_Just do IT.-CSDN博客](https://blog.csdn.net/zhoumushui/article/details/51443492?utm_medium=distribute.pc_relevant_right.none-task-blog-BlogCommendFromBaidu-6.nonecase&depth_1-utm_source=distribute.pc_relevant_right.none-task-blog-BlogCommendFromBaidu-6.nonecase)

[Android 9.0 (P版本) 亮度进度条变化等级更新\_苏法迪的专栏-CSDN博客](https://blog.csdn.net/su749520/article/details/84585672)

[Android 8.1 DisplayPowerController(四) 自动调节亮度(1)——流程\_sdkdlwk的博客-CSDN博客](https://blog.csdn.net/sdkdlwk/article/details/88368247?utm_medium=distribute.pc_relevant.none-task-blog-baidujs_title-2&spm=1001.2101.3001.4242)

[android 8.1自动背光随笔\_mafei19870124的专栏-CSDN博客](https://blog.csdn.net/mafei19870124/article/details/80435377?utm_medium=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-1.control&dist_request_id=6b0dd3ed-22ec-475c-9fdf-34bfd87a7435&depth_1-utm_source=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-1.control)

[自动背光调节\_lei7143的专栏-CSDN博客](https://blog.csdn.net/lei7143/article/details/73302287?utm_medium=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-1.control&dist_request_id=1328593.23209.16148289958593913&depth_1-utm_source=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-1.control)

### lcd notify

理解notifier 的机制，head初始化，client register/unregister, host call

查看drivers/video/fbdev/core/fb\_notify.c代码即可

small qusetion：其他driver如何引用host注册的head？

A: export

### Import 2.8” panel

11.23—11.27

导入新屏，需要对屏的结构进行分析。driver ic是ILI9327是一块RGB驱动芯片，所以外接了bridge ic icn6211。根据模组厂提供的init code，厘清屏的初始化流程。

对屏初始化需要使用spi对driver ic 进行初始化，然后使用iic对bridge ic进行初始化。最后mipi发送图片至bridge ic，转化成RGB信号给driver ic输出显示。

11.24

第一步初始化spi,根据qcom BLSP文档添加spi总线节点，补充status = “ok”;

但是根据移远提供的patch,目前只有一部分spi总线初始化成功。还有一些总线没有挂载成功，不清楚是引脚复用的问题还是TZ source code的影响。

根据ILITEAK厂商提供的code,使用GPIO模拟SPI的方式进行cmd的传输。

spi总结：

cs为片选引脚，cs为低表示有效。spi接多个设备需要接多个cs引脚。所以spi从设备没有地址。同时spi没有应答。所以spi纠错方式比较少，波形无法确认。

/\*---------------------------------------------------------------------------------------------------------------------------\*/

11.27

i2c，高通dts使用的是7位地址。8位地址是7位后补1/0表示读写位。

I2c写数据时序：

ST + 地址（8位）+ACK（0有效）+CMD（8位）+ACK（0）+DATA+ACK（0）+END

读取波形时，使用clk高电平位对sda读取状态。I2c发送一个数据对应是9个clk.

基本上能够读取到一个完整的写时序就代表i2c通信正常。如果i2c没有应答，第一，从波形图中读取i2c的地址，看地址是否对。第二，分析i2c芯片的电源是否有电，使能引脚是否使能。

/\*---------------------------------------------------------------------------------------------------------------------------\*/

11.30

Bridge ic ICN6211集创FAE来协助，量测6211的硬件信号分析软件上的错误，6211没有使能导致ic不工作。修改后量测结果，输入电压正常，输出频率符合数据，mipi输出的RGB波形与彩条模式下的波形一致。确认bridge IC工作正常。

/\*---------------------------------------------------------------------------------------------------------------------------\*/

12.1

Lcd driver ic奕力FAE来协助，量测IOVCC,RESET,CS,波形。量测引脚的电平状态有三段式变化。量测二段三段，二段正常为1.8V，三段异常为2.2V。三段异常。在不接屏的情况下，波形正常。结合bridge ic的调试文档，注意到bridge ic的输出数据电压准位为3.3V,diver ic的电压准位为1.8V。电压准位不匹配导致数据分析不准确。

/\*---------------------------------------------------------------------------------------------------------------------------\*/

12.2

跳线修改bridge ic的电压准位，彩条模式与mipi模式均显示正常。

总结：

驱动ic,学会用硬件的方法分析问题。多个ic共同工作的情况下，首先要确认各个硬件的单独工作状态是否正常。在单独工作正常的情况下，再确认连接部分是否有问题。

分析硬件的逻辑：

1. 量测power

量测ic的输入电压波形是否正常，使能引脚是否正常。工作升压power是否正常。

有关power的部分可以去参考ic的spec.

1. 量测信号输入pin

量测ICN6211的i2c输入信号，如果正常可以从i2c中解析到正确的数据

1. 量测信号输出pin

量测VSYNC，HSYNC，PCLK等pin的输出。

这三步只是一种思路，实际过程不一定按顺序排除。一般是检测是否有输入信号，然后根据情况去判断工作电压是否正常，或者输出信号是否正常。对于量测波形中有异常的问题，一定要深入分析具体原因。比如这次三段式电压，三段电压分别是多少，中间电压状态存在多久，最高电压是否是准位电压？不要忽略细节。

硬件分析很重要，后续的工作中需要去培养这方面的能力。软件分析是解决逻辑错误，硬件分析是定位错误。结合硬件可以更快的发现问题。实际工作中，问题的原因的多方多面的，不仅仅是代码的问题，需要协调各方人员一起推动去解决问题，需要锻炼自己沟通的能力。在一些新技术上，有薄弱点很正常，更重要的是如何解决。

## led

### led

[Android8.1系统Led的控制从底层到上层的实现\_只是一些暗恋而已的博客-CSDN博客](https://blog.csdn.net/qq_30624591/article/details/88216726?utm_medium=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-4.add_param_isCf&depth_1-utm_source=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-4.add_param_isCf)

[Android 7.1 从底层到上层分析 Led 例子\_qq\_35003588的博客-CSDN博客](https://blog.csdn.net/qq_35003588/article/details/99681852?utm_medium=distribute.pc_relevant.none-task-blog-title-4&spm=1001.2101.3001.4242)

[我的内核学习笔记11：linux leds-gpio驱动应用实例\_李迟的专栏-CSDN博客](https://blog.csdn.net/subfate/article/details/53494926?utm_source=blogxgwz6)

[Android硬件抽象层（HAL）概要介绍和学习计划\_老罗的Android之旅-CSDN博客](https://blog.csdn.net/luoshengyang/article/details/6567257)

### light

[Android Notification - 灯光提示\_chengchaooppo的专栏-CSDN博客](https://blog.csdn.net/chengchaooppo/article/details/105529783)

[2.5.8 Notification(状态栏通知)详解 | 菜鸟教程](https://www.runoob.com/w3cnote/android-tutorial-notification.html)

[傻瓜式Android APP开发入门教程 | 菜鸟教程](https://www.runoob.com/w3cnote/android-app-develop-learning.html)

### LED out of control

sprint新项目base gemini, led pin 有修改。

新项目的led需要修改的内容？led不受控制？

1. check led pin change list

from CPU gpio to I2C gpio, add I2C driver.

from I2C gpio to CPU gpio, whether disable I2C driver.

1. disable else gpio referenced and pinctrol

## gpio

### gpio

[高通平台中gpio简单操作和调试\_s\_jason的博客-CSDN博客\_android gpio调试](https://blog.csdn.net/s_jason/article/details/73864103)

[高通msm8909对gpio的使用（包括pinctrl的功能设置，gpio在节点中的引用，驱动程序获取gpio的方法）\_tasawen的博客-CSDN博客\_msm8909 gpio](https://blog.csdn.net/tasawen/article/details/89791344?utm_medium=distribute.pc_relevant_right.none-task-blog-BlogCommendFromMachineLearnPai2-1.nonecase&depth_1-utm_source=distribute.pc_relevant_right.none-task-blog-BlogCommendFromMachineLearnPai2-1.nonecase)

[致驱动工程师的一封信](http://www.wowotech.net/device_model/429.html)

[Linux设备树语法详解 - Abnor - 博客园](https://www.cnblogs.com/xiaojiang1025/p/6131381.html)

[设备树之pinctrl编写 - ZYNQ/FPGA - 米联客uisrc](http://www.uisrc.com/forum.php?mod=viewthread&tid=2644)

<https://www.kernel.org/doc/Documentation/gpio/consumer.txt>

[Linux内核中的GPIO系统之（3）：pin controller driver代码分析](http://www.wowotech.net/gpio_subsystem/pin-controller-driver.html)

[(1条消息) Linux下的gpio,gpiod\_嵌入式Linux-CSDN博客](https://blog.csdn.net/weiqifa0/article/details/105085365/#t7)

gpio dts格式

gpio解析函数

106 int of\_get\_named\_gpio\_flags(struct device\_node \*np, const char \*list\_name,

107 int index, enum of\_gpio\_flags \*flags)

怎么去读一个寄存器

GPIO 的输入输出模式硬件原理

三极管，MOS管的常见类型，及作用

### 设备树如何管理gpio function

12/24

drivers/pinctrl/qcom/pinctrl-msm8909.c中，

static const struct msm\_pingroup msm8909\_groups[]注册每个gpio的function table,

将每个功能列为enum,

enum msm8909\_functions {

...

msm\_mux\_gcc\_gp1\_clk\_b,

...

};

static const char \* const gcc\_gp1\_clk\_b\_groups[] = {

"gpio14",

};

static const struct msm\_function msm8909\_functions[] = {

…

FUNCTION(gcc\_gp1\_clk\_b),

…

};

### GPIO driver

最近有一些基础gpio问题发现自己并不是很清楚。所以打算从基础重新整理一遍。

Gpio是如何匹配的，最基础的配置是什么？

Gpio 引用和pinctrl

Gpio和gpiod的关系，以及使用的函数方法有什么区别？

Dts配置里每一个参数的作用，以及基础的设备树的配置。

Debug gpio的方法

Gpio中断

Gpio regitster有各种功能，driver是如何区分的？

Gpio如何用于i2c,spi,uart,irq等功能

Gpio传递给上层接口

Gpio suspend管理

总结：

gpio subsystem是一个很完整的功能，

像gpio的使用，gpio与pinctrl, gpiod, i2c, interrupt的关系需要积累。暂时无法描述。

看代码

看博客

看高通文档

80-P2472-1B\_E\_MSM8953\_GPIO\_Configuration\_Spreadsheet.xlsm

与drivers/pinctrl/qcom/pinctrl-msm8953.c

gpio func是对应的，所以kernel不需要去管理gpio register.

实际上有一些并不对应，但所有gpio都有Primary function: gpio.

所以没有仔细看。

Define MACRO

### gpio 推挽输出，开漏输出

General-purpose input/output

<https://blog.csdn.net/yym6789/article/details/88674942>

这篇博客对GPIO的硬件原理做了非常好的解析，这里我只是简单的学习了推挽输出和开漏输出。

但是现在忘得也差不多了，不知道啥该学啥不用学。

…

### gpio driver strength

gpio driver strength如何生效，普通的gpio\_direction\_out是什么配置？

第一步vi –t gpio\_direction\_output

当然最好还是有实例来分析，这里是有原因的。

找一个实例分析：

1. drivers/video/fbdev/msm/mdss\_dsi\_panel.c
2. gpio\_direction\_output((ctrl\_pdata->lcd\_pwr\_en), enable);

tags追代码 <ctrl-]> gpio\_direction\_output

# pri kind tag                    文件

1 F   f    gpio\_direction\_output  arch/m68k/include/asm/mcfgpio.h

             typeref:**typename**:**int**

**static** **inline** **int** gpio\_direction\_output(unsigned gpio, **int** value)

2 F   f    gpio\_direction\_output  arch/mips/include/asm/mach-au1x00/gpio-au1300.h

             typeref:**typename**:**int**

**static** **inline** **int** gpio\_direction\_output(unsigned **int** gpio, **int** v)

3 F   f    gpio\_direction\_output  include/asm-generic/gpio.h

             typeref:**typename**:**int**

**static** **inline** **int** gpio\_direction\_output(unsigned gpio, **int** value)

4 F   f    gpio\_direction\_output  include/linux/gpio.h

             typeref:**typename**:**int**

**static** **inline** **int** gpio\_direction\_output(unsigned gpio, **int** value)

pe number and <Enter> (q or empty cancels):

这里有两种方法去追代码，

一是根据经验（猜），

首先排除看似不可能的头文件路径引用，其次对有可能的文件追进去看看，逻辑符合就向下追,不符合就换下一个有可能的头文件引用。

二是根据头文件的包含关系去追。

这种方式较为准确，这也是根据实例分析的好处。

drivers/video/fbdev/msm/mdss\_dsi\_panel.c

#include <linux/of.h>

#include <linux/of\_gpio.h>

#include <linux/gpio.h>

打开 gpio.h文件

1. include/linux/gpio.h
2. #ifndef \_\_LINUX\_GPIO\_H
3. #define \_\_LINUX\_GPIO\_H
4. #ifdef CONFIG\_GPIOLIB
5. #ifdef CONFIG\_ARCH\_HAVE\_CUSTOM\_GPIO\_H
6. #include <asm/gpio.h>
7. #else
8. #include <asm-generic/gpio.h>
9. **static** **inline** **void** gpio\_set\_value(unsigned **int** gpio, **int** value)
10. {
11. \_\_gpio\_set\_value(gpio, value);
12. }
13. #endif /\* ! CONFIG\_ARCH\_HAVE\_CUSTOM\_GPIO\_H \*/
14. #else /\* ! CONFIG\_GPIOLIB \*/
15. **static** **inline** **int** gpio\_direction\_output(unsigned gpio, **int** value)
16. {
17. **return** -ENOSYS;
18. }
19. #endif /\* ! CONFIG\_GPIOLIB \*/
20. #endif /\* \_\_LINUX\_GPIO\_H \*/

看到这里是有问题的，gpio\_dtrection\_out直接返回ENOSYS？？？

inline是内敛函数，功能是在函数调用的位置展开。并不是虚函数，

虚函数才是可以填充的啊！！！

细心一看，是config的问题，查看.config文件，CONFIG\_GPIOLIB是被配置的，

gpio\_direction\_output在#else 和 #endif中间，并没有被编译。

继续向下看，linux/gpio.h 在中CONFIG\_GPIOLIB又包含了asm-generic/gpio.h

CONFIG\_ARCH\_HAVE\_CUSTOM\_GPIO\_H没有被编译，所以不需要考虑asm/gpio.h

打开asm-generic/gpio.h文件

1. include/asm-generic/gpio.h
3. #ifdef CONFIG\_GPIOLIB
4. #include <linux/gpio/driver.h>
5. #include <linux/gpio/consumer.h>
6. **struct** device;
7. **struct** gpio;
8. **struct** seq\_file;
9. **struct** module;
10. **struct** device\_node;
11. **struct** gpio\_desc;
12. **static** **inline** **int** gpio\_direction\_output(unsigned gpio, **int** value)
13. {
14. **return** gpiod\_direction\_output\_raw(gpio\_to\_desc(gpio), value);
15. }

这里向下追就可以了，但是这里遇到了一个问题。

struct gpio\_desc;什么意思？

声明了一个空的结构体。可能是为了这里的函数可以引用这些结构体，但是在这里只是定义了一个壳，这里不对此类结构体解析，所以这里也不能引用结构体里的成员。

1. # pri kind tag                         文件
2. 1 F   f    gpiod\_direction\_output\_raw  drivers/gpio/gpiolib.c
3. typeref:**typename**:**int**
4. **int** gpiod\_direction\_output\_raw(**struct** gpio\_desc \*desc, **int** value)
5. 2 F   f    gpiod\_direction\_output\_raw  include/linux/gpio/consumer.h
6. typeref:**typename**:**int**
7. **static** **inline** **int** gpiod\_direction\_output\_raw(**struct** gpio\_desc \*desc, **int** value)
8. pe number and <Enter> (q or empty cancels):

按照头文件引用关系，我们先看gpio/comsumer.h

1. #ifdef CONFIG\_GPIOLIB
3. **int** gpiod\_direction\_output\_raw(**struct** gpio\_desc \*desc, **int** value);
5. #else /\* CONFIG\_GPIOLIB \*/
7. **static** **inline** **int** gpiod\_direction\_output\_raw(**struct** gpio\_desc \*desc, **int** value)
8. {
9. /\* GPIO can never have been requested \*/
10. WARN\_ON(1);
11. **return** -ENOSYS;
12. }
13. #endif /\* CONFIG\_GPIOLIB \*/

下面的不编译，看上面的。这里只是声明，定义还需要看那些文件包含了gpio/comsumer.h

现在看drivers/gpio/gpiolib.c

1. #include <linux/gpio.h>
2. #include "gpiolib.h"
4. /\*\*
5. \* gpiod\_direction\_output\_raw - set the GPIO direction to output
6. \* @desc:       GPIO to set to output
7. \* @value:      initial output value of the GPIO
8. \*
9. \* Set the direction of the passed GPIO to output, such as gpiod\_set\_value() can
10. \* be called safely on it. The initial value of the output must be specified
11. \* as raw value on the physical line without regard for the ACTIVE\_LOW status.
12. \*
13. \* Return 0 in case of success, else an error code.
14. \*/
15. **int** gpiod\_direction\_output\_raw(**struct** gpio\_desc \*desc, **int** value)
16. {
17. VALIDATE\_DESC(desc);
18. **return** \_gpiod\_direction\_output\_raw(desc, value);
19. }
20. EXPORT\_SYMBOL\_GPL(gpiod\_direction\_output\_raw);

头文件包含了linux/gpio.h,相当于包含了comsumer.h

VALIDATE\_DESC检验参数是否有效,向下追

1. **static** **int** \_gpiod\_direction\_output\_raw(**struct** gpio\_desc \*desc, **int** value)
2. {
3. **struct** gpio\_chip \*gc = desc->gdev->chip;
4. **int** ret;
6. /\* GPIOs used for IRQs shall not be set as output \*/
7. **if** (test\_bit(FLAG\_USED\_AS\_IRQ, &desc->flags)) {
8. gpiod\_err(desc,
9. "%s: tried to set a GPIO tied to an IRQ as output\n",
10. \_\_func\_\_);
11. //return -EIO;//rid: gt9xx need set the GPIOs as output when the GPIOs used for IRQs.
12. }
14. **if** (test\_bit(FLAG\_OPEN\_DRAIN, &desc->flags)) {
15. /\* First see if we can enable open drain in hardware \*/
16. **if** (gc->set\_single\_ended) {
17. ret = gc->set\_single\_ended(gc, gpio\_chip\_hwgpio(desc),
18. LINE\_MODE\_OPEN\_DRAIN);
19. **if** (!ret)
20. **goto** set\_output\_value;
21. }
22. /\* Emulate open drain by not actively driving the line high \*/
23. **if** (value)
24. **return** gpiod\_direction\_input(desc);
25. }
26. **else** **if** (test\_bit(FLAG\_OPEN\_SOURCE, &desc->flags)) {
27. **if** (gc->set\_single\_ended) {
28. ret = gc->set\_single\_ended(gc, gpio\_chip\_hwgpio(desc),
29. LINE\_MODE\_OPEN\_SOURCE);
30. **if** (!ret)
31. **goto** set\_output\_value;
32. }
33. /\* Emulate open source by not actively driving the line low \*/
34. **if** (!value)
35. **return** gpiod\_direction\_input(desc);
36. } **else** {
37. /\* Make sure to disable open drain/source hardware, if any \*/
38. **if** (gc->set\_single\_ended)
39. gc->set\_single\_ended(gc,
40. gpio\_chip\_hwgpio(desc),
41. LINE\_MODE\_PUSH\_PULL);
42. }
44. set\_output\_value:
45. **if** (!gc->set || !gc->direction\_output) {
46. gpiod\_warn(desc,
47. "%s: missing set() or direction\_output() operations\n",
48. \_\_func\_\_);
49. **return** -EIO;
50. }
52. ret = gc->direction\_output(gc, gpio\_chip\_hwgpio(desc), value);
53. **if** (!ret)
54. set\_bit(FLAG\_IS\_OUT, &desc->flags);
55. trace\_gpio\_value(desc\_to\_gpio(desc), 0, value);
56. trace\_gpio\_direction(desc\_to\_gpio(desc), 0, ret);
57. **return** ret;
58. }

这个函数大概的意思就是根据desc中的flags设置成开漏输出，或开集输出，或推挽输出。

我在mdss\_dsi\_panel.c中添加log去打印flags的log，结果编译报错说没有这个变量。

所以才会根据头文件引用来分析追溯。

一个是确认gpio实作的代码确实在这里，第二个是结构体成员打印不了可能是和定义的空结构体有关系。

在引用的位置添加代码不能打印关键信息，就只能在源代码位置打印。

打印log发现，一个都没有执行。

真实有效的代码段在52-56行，trace是对gpio的追踪，这里现在也不明白，先挖坑。

真正执行的是gc->direction\_output

gc 的结构体是**struct** gpio\_chip，所以就是gpio\_chip是通用模型，对应的platform注册struct gpio\_chip xxx = {.direction\_output=xxx;},进行差异化的细分。

grep一下

grep -nR "direction\_output" drivers/gpio/

grep -nR "direction\_output" drivers/pinctrl/

1. drivers/pinctrl/qcom/pinctrl-msm.c:411:**static** **int** msm\_gpio\_direction\_output(**struct** gpio\_chip \*chip, unsigned offset, **int** value)
2. drivers/pinctrl/qcom/pinctrl-msm.c:525: .direction\_output = msm\_gpio\_direction\_output,
3. **static** **int** msm\_gpio\_direction\_output(**struct** gpio\_chip \*chip, unsigned offset, **int** value)
4. {
5. **const** **struct** msm\_pingroup \*g;
6. **struct** msm\_pinctrl \*pctrl = gpiochip\_get\_data(chip);
7. unsigned **long** flags;
8. u32 val;
10. g = &pctrl->soc->groups[offset];
12. spin\_lock\_irqsave(&pctrl->lock, flags);
14. val = readl(pctrl->regs + g->io\_reg);
15. **if** (value)
16. val |= BIT(g->out\_bit);
17. **else**
18. val &= ~BIT(g->out\_bit);
19. writel(val, pctrl->regs + g->io\_reg);
21. val = readl(pctrl->regs + g->ctl\_reg);
22. val |= BIT(g->oe\_bit);
23. writel(val, pctrl->regs + g->ctl\_reg);
25. spin\_unlock\_irqrestore(&pctrl->lock, flags);
27. **return** 0;
28. }

**代码有三段：**

1.**const** **struct** msm\_pingroup \*g; g代表的时对应gpio的基地址。

这里需要看3个内容：

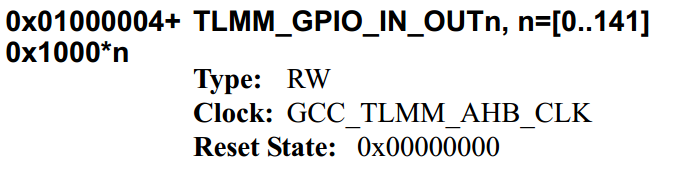
pinctrl\_msm8953.c中的static const struct msm\_pingroup数组

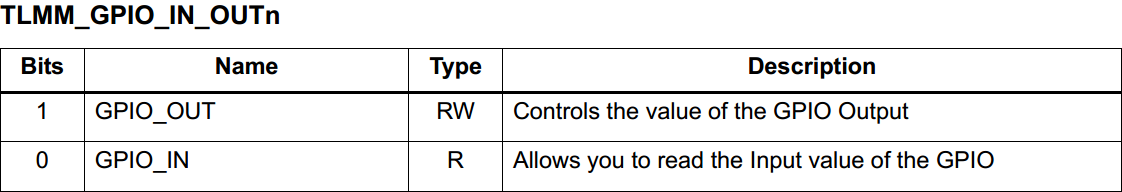
PINGROUP宏定义

msm8953的hardware register spec

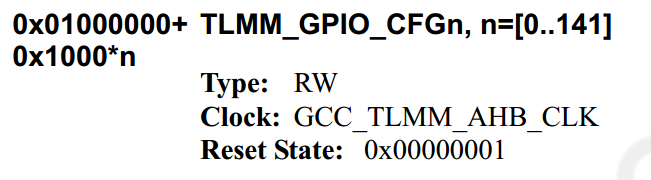
2.readl,val赋值,write标准的写寄存器操作。

out\_bit写1或置0到io\_reg.

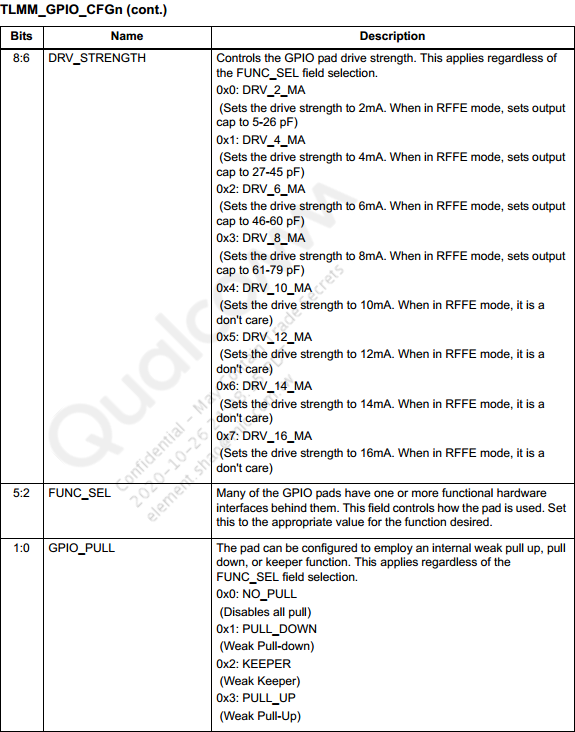




3.最后，oe\_bit置1写到ctl\_reg中。ob\_bit=9,看spec



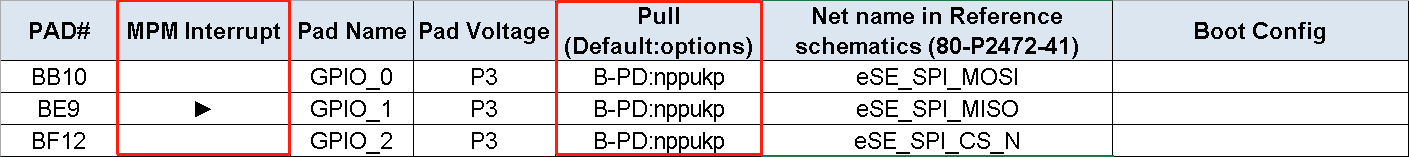


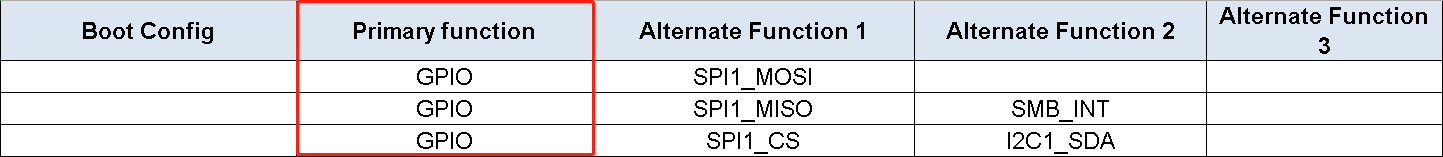


到这里就表示gpio\_direction\_output都没有配置gpio\_pull,func\_sel,DRV\_STRENGTH寄存器。

那么gpio的driver strength是多少呢？

1. 看高通的文档gpio table 和 hardware register





80-P2472-1B\_E\_MSM8953\_GPIO\_Configuration\_Spreadsheet

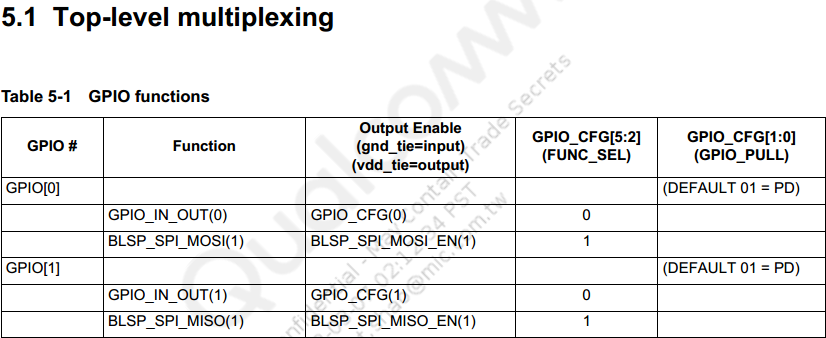
但这里的问题是，table只是说明了gpio是primary function.真实寄存器内的值是0x0吗？

Ps :MPM interrupt = Y，是指可配置可唤醒中断

PPs: ps这一条是在博客中看到的，对于每一个知识点，应当有自己的思考。即要学会怀疑它的合理性，也要学会证明他的真实性。

任何真实的东西都应当存在与代码中。

这里暂时不展开分析IRQ，大致的内容可以参考<http://www.wowotech.net/pm_subsystem/suspend-irq.html>



80-pc173-2x\_b\_sdm450\_hardware\_register\_description

不是很放心，因为这是SDM450的HW\_register.虽然SDM450=MSM8953,但是还是不放心。

1. 通过命令

1）可以cat d/gpio节点来查看当前gpio的配置。

1. triton:/ # cat d/gpio
2. gpiochip0: GPIOs 0-141, parent: platform/1000000.pinctrl, 1000000.pinctrl:
3. gpio0   : out 0 2mA pull down
4. gpio1   : out 0 2mA pull down

这里只能查看当前gpio的部分配置，一方面不是所有寄存器里的值，另一方面并不能代表gpio寄存器里默认的值。

简单的说，我们现在只看到gpio\_direction\_output修改了哪些寄存器，但是其它寄存器在这之前有没有被修改还没办法确认。比如gpio\_request, pinctrl, lk, SBL, ADSP等操作对gpio寄存器的影响？

gpio\_request应该没有去修改这些寄存器，因为request是属于sw管理的概念，但是后面还是会对gpio\_requeset的过程进行解析。包括i2c pin的这些寄存器是如何配置的。

2）sys/class/gpio/export

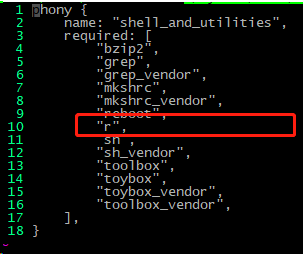
这个只能debug没有request的gpio，实用性不高

3）/system/bin/r

这个tool被包含在BSP源码中system/core/toolbox/r.c

但是现在SC600的平台没有编译这个tool.

在system/core/shell\_and\_utilities/Android.bp添加进编译



编译android后，/system/bin目录下就会有这个tool了.

同时底层需要配置CONFIG\_DEVMEM=y，重新编译，否则会报cannot open /dev/mem

使用语法./r register\_address

读寄存器r使用总结：

line 1: syntax error: unexpected ")"

说明编译器版本不兼容，需要使用源码的编译环境编译

### gpio request

### pinctrl

学习pinctrl，首先要了解pinctrl的基本用法，APIs和dts的格式。

然后解析pinctrl API是如何修改gpio寄存器的？以及pinctrl和gpio是如何交互的？

第三点，额外获得了一个咨询，driver probe之前，会对device的pinctrl进行默认操作。

#### pinctrl的基本用法

如何学习pinctrl的用法呢？还在一样我们找实例分析

按照一样的思路，看初始化API和调用API。

1. drivers/video/fbdev/msm/mdss\_dsi.c
2. **static** **int** mdss\_dsi\_pinctrl\_init(**struct** platform\_device \*pdev)
3. {
4. **struct** mdss\_dsi\_ctrl\_pdata \*ctrl\_pdata;
6. ctrl\_pdata = platform\_get\_drvdata(pdev);
7. ctrl\_pdata->pin\_res.pinctrl = devm\_pinctrl\_get(&pdev->dev);
8. **if** (IS\_ERR\_OR\_NULL(ctrl\_pdata->pin\_res.pinctrl)) {
9. pr\_err("%s: failed to get pinctrl\n", \_\_func\_\_);
10. **return** PTR\_ERR(ctrl\_pdata->pin\_res.pinctrl);
11. }
13. ctrl\_pdata->pin\_res.gpio\_state\_active
14. = pinctrl\_lookup\_state(ctrl\_pdata->pin\_res.pinctrl,
15. MDSS\_PINCTRL\_STATE\_DEFAULT);
16. **if** (IS\_ERR\_OR\_NULL(ctrl\_pdata->pin\_res.gpio\_state\_active))
17. pr\_warn("%s: can not get default pinstate\n", \_\_func\_\_);
19. ctrl\_pdata->pin\_res.gpio\_state\_suspend
20. = pinctrl\_lookup\_state(ctrl\_pdata->pin\_res.pinctrl,
21. MDSS\_PINCTRL\_STATE\_SLEEP);
22. **if** (IS\_ERR\_OR\_NULL(ctrl\_pdata->pin\_res.gpio\_state\_suspend))
23. pr\_warn("%s: can not get sleep pinstate\n", \_\_func\_\_);
25. **return** 0;
26. }
27. **static** **int** mdss\_dsi\_pinctrl\_set\_state(
28. **struct** mdss\_dsi\_ctrl\_pdata \*ctrl\_pdata,
29. **bool** active)
30. {
31. **struct** pinctrl\_state \*pin\_state;
32. **struct** mdss\_panel\_info \*pinfo = NULL;
33. **int** rc = -EFAULT;
35. **if** (IS\_ERR\_OR\_NULL(ctrl\_pdata->pin\_res.pinctrl))
36. **return** PTR\_ERR(ctrl\_pdata->pin\_res.pinctrl);
38. pin\_state = active ? ctrl\_pdata->pin\_res.gpio\_state\_active
39. : ctrl\_pdata->pin\_res.gpio\_state\_suspend;
40. **if** (!IS\_ERR\_OR\_NULL(pin\_state)) {
41. rc = pinctrl\_select\_state(ctrl\_pdata->pin\_res.pinctrl,
42. pin\_state);
43. **if** (rc)
44. pr\_err("%s: can not set %s pins\n", \_\_func\_\_,
45. active ? MDSS\_PINCTRL\_STATE\_DEFAULT
46. : MDSS\_PINCTRL\_STATE\_SLEEP);
47. } **else** {
48. pr\_err("%s: invalid '%s' pinstate\n", \_\_func\_\_,
49. active ? MDSS\_PINCTRL\_STATE\_DEFAULT
50. : MDSS\_PINCTRL\_STATE\_SLEEP);
51. }
52. **return** rc;
53. }

总结一下pinctrl的使用，三个结构体，三个API.

1. **struct** pinctrl \*pinctrl;
2. **struct** pinctrl\_state \*gpio\_state\_active;
3. **struct** pinctrl\_state \*gpio\_state\_suspend;
4. //init:先获取pinctrl结构体，再获取active/suspend state
5. **struct** pinctrl \*devm\_pinctrl\_get(**struct** device \*dev)
6. **struct** pinctrl\_state \*pinctrl\_lookup\_state(**struct** pinctrl \*p, **const** **char** \*name)
7. **//set:在调用的位置，设置相应的state**
8. **int** pinctrl\_select\_state(**struct** pinctrl \*p, **struct** pinctrl\_state \*state)

接下来看一下pincrtl dts的配置，pinctrl\_select\_state的具体逻辑我们后面再看。

pinctrl\_lookup\_state(ctrl\_pdata->pin\_res.pinctrl, MDSS\_PINCTRL\_STATE\_DEFAULT);

vi –t MDSS\_PINCTRL\_STATE\_DEFAULT

#define MDSS\_PINCTRL\_STATE\_DEFAULT "mdss\_default"

grep –nR "MDSS\_PINCTRL\_STATE\_DEFAULT"

1. &mdss\_dsi0 {
2. qcom,dsi-pref-prim-pan = <&dsi\_kd070d57\_video>;
3. pinctrl-names = "mdss\_default", "mdss\_sleep";
4. pinctrl-0 = <&mdss\_dsi\_active &mdss\_te\_active &mdss\_power\_en\_active>;
5. pinctrl-1 = <&mdss\_dsi\_suspend &mdss\_te\_suspend &mdss\_power\_en\_suspend>;
6. qcom,platform-te-gpio = <&tlmm 24 0>;
7. qcom,platform-reset-gpio = <&tlmm 61 0>;
8. qcom,platform-bklight-en-gpio = <&tlmm 45 0>;
9. qcom,platform-lcd-pwr-en-gpio = <&tlmm 1 0>;
10. qcom,platform-lcd-vdd-en-gpio = <&tlmm 0 0>;
11. };
12. &tlmm {
13. pmx\_mdss {
14. mdss\_dsi\_active: mdss\_dsi\_active {
15. mux {
16. pins = "gpio61";
17. };
18. config {
19. pins = "gpio61";
20. };
21. };
22. mdss\_dsi\_suspend: mdss\_dsi\_suspend {
23. mux {
24. pins = "gpio61";
25. };
26. config {
27. pins = "gpio61";
28. };
29. };
30. };
31. };
32. pmx\_mdss\_te {
33. mdss\_te\_active: mdss\_te\_active {
34. mux {
35. pins = "gpio24";
36. function = "mdp\_vsync";
37. };
38. config {
39. pins = "gpio24";
40. drive-strength = <2>; /\* 8 mA \*/
41. bias-pull-down; /\* pull down\*/
42. };
43. };
45. mdss\_te\_suspend: mdss\_te\_suspend {
46. mux {
47. pins = "gpio24";
48. function = "mdp\_vsync";
49. };
50. config {
51. pins = "gpio24";
52. drive-strength = <2>; /\* 2 mA \*/
53. bias-pull-down; /\* pull down \*/
54. };
55. };
56. };
57. pmx\_mdss\_power\_en:pmx\_mdss\_power\_en{
58. mdss\_power\_en\_active:mdss\_power\_en\_active{
59. mux{
60. pins = "gpio0","gpio1";
61. function = "gpio";
62. };
63. config {
64. pins = "gpio0","gpio1";
65. drive-strength = <8>;
66. bias-disable = <0>;
67. output-high;
68. };
69. };
70. mdss\_power\_en\_suspend:mdss\_power\_en\_suspend{
71. mux{
72. pins = "gpio0","gpio1";
73. function = "gpio";
74. };
75. config{
76. pins = "gpio0","gpio1";
77. drive-strength = <2>;
78. bias-pull-down;
79. output-low;
80. };
81. };
82. };

总结有两点:一个是设备节点引用pinctrl node，第二个是pinctrl node的语法

1. &device node{
2. pinctl-names="xxx\_default", "xxx\_sleep";
3. pinctrl-0 = <&pinctrl\_node\_active>;
4. pinctrl-1 = <&pinctrl\_node\_suspend>;
5. }
6. pinctrl names = "";可以有多个property,对应有多少个pinctrl-num;一般就两个;
7. 每个pinctrl-num = <>;可以有多个节点，用空格隔开
8. &tlmm{//根节点是gpio controller
9. pinctrl\_node{
10. pinctrl\_node\_active{
11. mux{            };  //设置gpio，func
12. config{              };  //设置很多config，可以去看document
13. }
14. pinctrl\_node\_suspend{
15. mux{              };
16. config{              };
17. }
18. }
19. }
20. 主要是pinctrl\_node的配置语法以及参数，多看一下device tree中其它节点的配置参数，以及下面的文档，就算是基本掌握了pinctrl的使用方法了。
21. 对pinctrl node property的解释可以参考Documentation/devicetree/bindings/pinctrl/pinctrl-bindings.txt

#### 解析

解析pinctrl是如何修改gpio寄存器的，同时这里也有几个问题。

在设备节点中，可以看到pinctrl\_node和gpio\_node事实上是有重复的。pinctrl API和GPIO API对同一个gpio寄存器的影响？

pinctrl-names和pinctrl-num的匹配过程？

解析devm\_pinctrl\_get

先看一下devm\_pinctrl\_get的返回值struct pinctrl

1. /\*\*
2. \* struct pinctrl - per-device pin control state holder
3. \* @node: global list node
4. \* @dev: the device using this pin control handle
5. \* @states: a list of states for this device
6. \* @state: the current state
7. \* @dt\_maps: the mapping table chunks dynamically parsed from device tree for
8. \*      this device, if any
9. \* @users: reference count
10. \*/
11. **struct** pinctrl {
12. **struct** list\_head node;
13. **struct** device \*dev;
14. **struct** list\_head states;
15. **struct** pinctrl\_state \*state;
16. **struct** list\_head dt\_maps;
17. **struct** kref users;
18. };

简单的解析，pinctrl结构体当中有device，pinctrl\_state.也就是说只要过去pinctrl结构体就可以获取pinctrl的所有state的操作方法。

深度解析就是pinctrl通过list\_head node添加到一个链表中，可以通过一个list遍历所有的pinctrl结构体。

pinctrl\_state，和list\_head state也是同样的道理。

接下来看代码

在pinctrl\_core.c中可以看到pinctrl被封装成list，pinctrl中包含device和pinctrl\_state.所以解析的具体内容需要看pinctrl driver是如何添加list的。获取的这两个API都没有实际操作，暂时不看。

1. **struct** pinctrl \*devm\_pinctrl\_get(**struct** device \*dev)
2. p = pinctrl\_get(dev);
3. p = find\_pinctrl(dev);
4. list\_for\_each\_entry(p, &pinctrl\_list, node)
5. **struct** pinctrl \*devm\_pinctrl\_get(**struct** device \*dev)
6. {
7. **struct** pinctrl \*\*ptr, \*p;
9. ptr = devres\_alloc(devm\_pinctrl\_release, **sizeof**(\*ptr), GFP\_KERNEL);
10. **if** (!ptr)
11. **return** ERR\_PTR(-ENOMEM);
13. p = pinctrl\_get(dev);
14. **if** (!IS\_ERR(p)) {
15. \*ptr = p;
16. devres\_add(dev, ptr);
17. } **else** {
18. devres\_free(ptr);
19. }
21. **return** p;
22. }
23. **struct** pinctrl \*pinctrl\_get(**struct** device \*dev)
24. {
25. **struct** pinctrl \*p;
27. **if** (WARN\_ON(!dev))
28. **return** ERR\_PTR(-EINVAL);
30. /\*
31. \* See if somebody else (such as the device core) has already
32. \* obtained a handle to the pinctrl for this device. In that case,
33. \* return another pointer to it.
34. \*/
35. p = find\_pinctrl(dev);
36. **if** (p != NULL) {
37. dev\_dbg(dev, "obtain a copy of previously claimed pinctrl\n");
38. kref\_get(&p->users);
39. **return** p;
40. }
42. **return** create\_pinctrl(dev);
43. }
44. **static** **struct** pinctrl \*find\_pinctrl(**struct** device \*dev)
45. {
46. **struct** pinctrl \*p;
48. mutex\_lock(&pinctrl\_list\_mutex);
49. list\_for\_each\_entry(p, &pinctrl\_list, node)
50. **if** (p->dev == dev) {
51. mutex\_unlock(&pinctrl\_list\_mutex);
52. **return** p;
53. }
55. mutex\_unlock(&pinctrl\_list\_mutex);
56. **return** NULL;
57. }

list\_for\_each\_entry表示pictrl已经被存储到了pinctrl\_list当中.剩下的内容就是比较pinctrl中的device与预期的device是否符合。详细解析请看[list head](#_list_head).

现在的问题就是pinctrl是如何被添加到pinctrl\_list的。

这是一次向上追代码的例子：

首先list\_for\_each\_entry(p, &pinctrl\_list, node)

pinctrl\_list是所有pinctrl结构体的链表头，

查看定义static LIST\_HEAD(pinctrl\_list);

静态变量，对pinctrl\_list的操作都应该在本文件当中。

搜索pinctrl\_list,找到list\_add\_tail(&p->node, &pinctrl\_list);

这里是对pinctrl\_list添加成员的地方，一路向上搜索。

1. list\_add\_tail
2. create\_pinctrl
3. pinctrl\_get//获取的时候是这个函数，添加的时候怎么也是这个函数？
4. //看代码逻辑，两个阶段的代码逻辑不同
5. devm\_pinctrl\_get
6. grep -nR "devm\_pinctrl\_register" drivers/pinctrl/
7. 我们的项目是高通的，所以选择qcom目录下的搜索结果
8. drivers/pinctrl/qcom/pinctrl-msm.c
9. pctrl->pctrl = devm\_pinctrl\_register(&pdev->dev, &msm\_pinctrl\_desc,
10. pctrl);

解析pinctrl\_lookup\_state

解析pinctrl\_select\_state

#### probe之前是否会执行pinctrl

1. **static** **int** really\_probe(**struct** device \*dev, **struct** device\_driver \*drv)
2. {
3. ...
4. /\* If using pinctrl, bind pins now before probing \*/
5. ret = pinctrl\_bind\_pins(dev);
6. ...
7. **if** (dev->bus->probe) {
8. ret = dev->bus->probe(dev);
9. **if** (ret)
10. **goto** probe\_failed;
11. }
12. ...
13. }

pinctrl\_bind\_pins函数放进来太长，代码段太长不便阅读，所以只保留了核心部分。

1. /\*\*
2. \* pinctrl\_bind\_pins() - called by the device core before probe
3. \* @dev: the device that is just about to probe
4. \*/
5. **int** pinctrl\_bind\_pins(**struct** device \*dev)
6. {
7. **int** ret;
9. dev->pins = devm\_kzalloc(dev, **sizeof**(\*(dev->pins)), GFP\_KERNEL);
10. dev->pins->p = devm\_pinctrl\_get(dev);
12. dev->pins->default\_state = pinctrl\_lookup\_state(dev->pins->p,
13. PINCTRL\_STATE\_DEFAULT);
14. dev->pins->init\_state = pinctrl\_lookup\_state(dev->pins->p,
15. PINCTRL\_STATE\_INIT);
16. **if** (IS\_ERR(dev->pins->init\_state)) {
17. /\* Not supplying this state is perfectly legal \*/
18. ret = pinctrl\_select\_state(dev->pins->p,
19. dev->pins->default\_state);
20. } **else** {
21. ret = pinctrl\_select\_state(dev->pins->p,
22. dev->pins->init\_state);
23. }

26. #ifdef CONFIG\_PM
27. /\*
28. \* If power management is enabled, we also look for the optional
29. \* sleep and idle pin states, with semantics as defined in
30. \* <linux/pinctrl/pinctrl-state.h>
31. \*/
32. dev->pins->sleep\_state = pinctrl\_lookup\_state(dev->pins->p,
33. PINCTRL\_STATE\_SLEEP);
34. dev->pins->idle\_state = pinctrl\_lookup\_state(dev->pins->p,
35. PINCTRL\_STATE\_IDLE);
36. #endif
38. **return** 0;
40. }

看一下结构体成员，device中包括dev\_pin\_info结构体，结构体包含了pinctrl结构体和四个pinctrl\_state: "default","init","idle","sleep". 再看上面的代码，也是通过同样的函数devm\_pinctrl\_get获取pinctrl,再获取pinctrl\_state. 设置init state,如果没有init state,就设置为default state.

值得注意的是，这里获取的pinctrl\_state的name是"default"，但是很多device的pinctrl-names是"xxx\_default"，name不匹配，这一段代码就不会生效。

1. /\*\*
2. \* struct device - The basic device structure
3. \* @pins:       For device pin management.
4. \*              See Documentation/pinctrl.txt for details.
5. \*/
7. **struct** device {
8. #ifdef CONFIG\_PINCTRL
9. **struct** dev\_pin\_info     \*pins;
10. #endif
11. }
13. /\*\*
14. \* struct dev\_pin\_info - pin state container for devices
15. \* @p: pinctrl handle for the containing device
16. \* @default\_state: the default state for the handle, if found
17. \* @init\_state: the state at probe time, if found
18. \* @sleep\_state: the state at suspend time, if found
19. \* @idle\_state: the state at idle (runtime suspend) time, if found
20. \*/
21. **struct** dev\_pin\_info {
22. **struct** pinctrl \*p;
23. **struct** pinctrl\_state \*default\_state;
24. **struct** pinctrl\_state \*init\_state;
25. #ifdef CONFIG\_PM
26. **struct** pinctrl\_state \*sleep\_state;
27. **struct** pinctrl\_state \*idle\_state;
28. #endif
29. };
31. /\*\*
32. \* @PINCTRL\_STATE\_DEFAULT: the state the pinctrl handle shall be put
33. \*      into as default, usually this means the pins are up and ready to
34. \*      be used by the device driver. This state is commonly used by
35. \*      hogs to configure muxing and pins at boot, and also as a state
36. \*      to go into when returning from sleep and idle in
37. \*      .pm\_runtime\_resume() or ordinary .resume() for example.
38. \* @PINCTRL\_STATE\_INIT: normally the pinctrl will be set to "default"
39. \*      before the driver's probe() function is called.  There are some
40. \*      drivers where that is not appropriate becausing doing so would
41. \*      glitch the pins.  In those cases you can add an "init" pinctrl
42. \*      which is the state of the pins before drive probe.  After probe
43. \*      if the pins are still in "init" state they'll be moved to
44. \*      "default".
45. \* @PINCTRL\_STATE\_IDLE: the state the pinctrl handle shall be put into
46. \*      when the pins are idle. This is a state where the system is relaxed
47. \*      but not fully sleeping - some power may be on but clocks gated for
48. \*      example. Could typically be set from a pm\_runtime\_suspend() or
49. \*      pm\_runtime\_idle() operation.
50. \* @PINCTRL\_STATE\_SLEEP: the state the pinctrl handle shall be put into
51. \*      when the pins are sleeping. This is a state where the system is in
52. \*      its lowest sleep state. Could typically be set from an
53. \*      ordinary .suspend() function.
54. \*/
55. #define PINCTRL\_STATE\_DEFAULT "default"
56. #define PINCTRL\_STATE\_INIT "init"
57. #define PINCTRL\_STATE\_IDLE "idle"
58. #define PINCTRL\_STATE\_SLEEP "sleep"

### gpio irq

## i2c & spi

<https://blog.csdn.net/qq_33790215/article/details/97614552#t3>

### I2C

--------------------------------------------------------------------------------------------------------------------------------

使用tool读取I2C

i2cget tool

adb root

adb push i2cget /data

adb shell

cd data

./i2cget -y -f 03 0x2c 0x20 w

bus addr reg

./i2cget –help

--------------------------------------------------------------------------------------------------------------------------------

#### Add I2C-to-gpio chip driver

在新项目上添加LED的功能。

首先需要添加LED的dts node.

LED是挂载在i2c-to-gpio tca6408 chip上，所以需要先加tca6408 driver.

tca6408是挂载在I2C BUS 1下，所以需要检查I2C1的dts.

如何去让一个driver运行起来？

1. 添加dts

添加tca6408的dts node, 添加led的dts node.

1. 根据config文件

根据dts的compatible找到对应的drive.c

根据Makefile文件确认编译driver.c需要的config

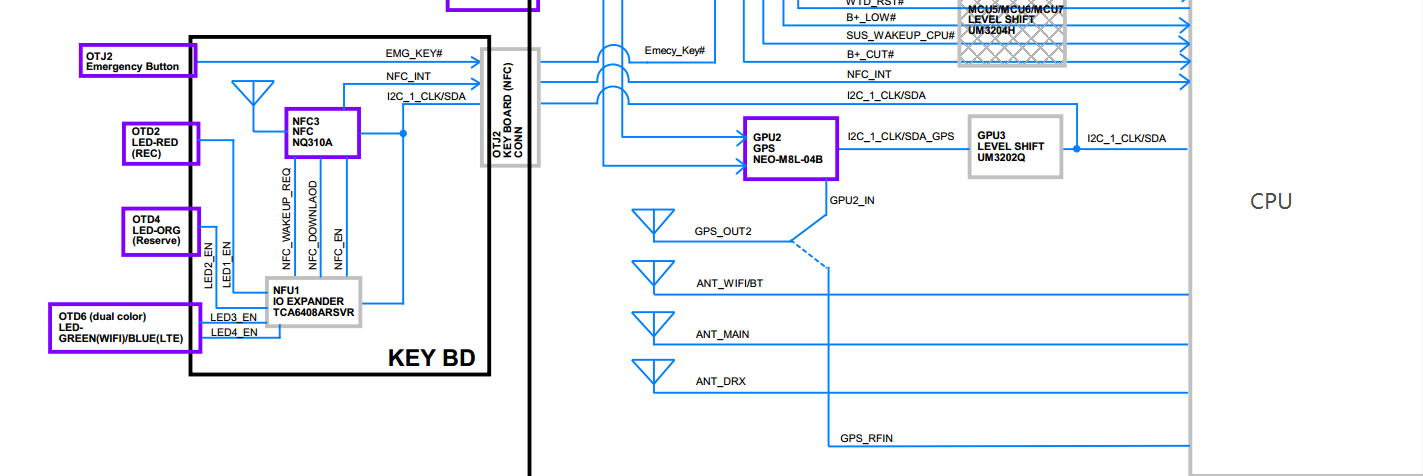
在defconfig文件添加对应的config

1. 修改代码

改代码需要改什么？代码包含了设备运行的全部逻辑，面向用户的open/read/write/close.面向内核的file system.面向自身的init/exit.

但是这些都不需要我们去改，driver的框架基本都是包含了这些内容。

我们需要保证的是driver的probe有被执行，代表driver的代码有正常的加载，然后再去看driver的工作状态。



那么，这边如愿以偿的遇到了tca6408不能正常工作。

表现状况为led匹配不到gpio.

方法一：修改代码，比较和旧项目代码的不同，将可能有关系的地方移植移植过来。

这样做确认能解决很多问题，但是如果对其逻辑不清晰的话，也就只是碰碰运气。

方法二：看log，通过软件的状态来判断出错的地方。

方法三：硬件分析。

修改代码，添加了CONFIG\_GPIOLIB（GPIOLIB的作用是什么？）, I2C BUS config，I2C gpio pinctrl等

tca6408的driver文件在pca953x.c中，将新文件与旧文件所有的不同直接copy.

tca6408的dts node保持一致。

1. tca6408: tca6408@20 {
2. compatible = "ti,tca6408";
3. reg = <0x20>;
4. status = "ok";
5. qcom,nq-vpad = <&tlmm 130 0x00>;
6. pinctrl-names = "nfc\_int","nfc\_pwr";
7. pinctrl-0 = <&nfc\_int\_active &nfc\_pwr\_active>;
8. pinctrl-1 = <&nfc\_int\_suspend &nfc\_pwr\_suspend>;
9. gpio-controller;
10. #gpio-cells = <2>;
11. };
12. @@ -759,6 +769,66 @@ **static** **int** pca953x\_probe(**struct** i2c\_client \*client,
14. chip->client = client;
16. +#ifdef CONFIG\_ARCH\_MITAC\_LYTX
17. +       **if**(!strcmp(client->name,"tca6408"))
18. +       {
19. +               nfc\_pinctrl = devm\_pinctrl\_get(&client->dev);
20. +               **if** (IS\_ERR(nfc\_pinctrl)) {
21. +                       **if** (of\_property\_read\_bool(client->dev.of\_node, "pinctrl-names")) {
22. +                               dev\_err(&client->dev, "Error encountered while getting pinctrl");
23. +                               rc = PTR\_ERR(nfc\_pinctrl);
24. +                       }
25. +                       dev\_dbg(&client->dev, "Target does not use pinctrl\n");
26. +                       nfc\_pinctrl = NULL;
27. +               }
28. +
29. +               **if**(nfc\_pinctrl != NULL) {
30. +                       nfc\_default\_state = pinctrl\_lookup\_state(nfc\_pinctrl, "nfc\_int");
31. +
32. +                       **if** (IS\_ERR\_OR\_NULL(nfc\_default\_state))
33. +                       {
34. +                               rc = PTR\_ERR(nfc\_default\_state);
35. +                               dev\_err(&client->dev, "Can not lookup %s pinstate rc=%d\n", "default", rc);
36. +                       }
37. +                       **else**
38. +                       {
39. +                               pinctrl\_select\_state(nfc\_pinctrl, nfc\_default\_state);
40. +                       }
41. +               }
42. +
43. +               /\*\*R01MB have no vbat\_en pin for NFC board\*\*/
44. +               vbat\_gpio = of\_get\_named\_gpio(np, "qcom,nq-vbat", 0);
45. +               **if** (gpio\_is\_valid(vbat\_gpio))
46. +               {
47. +                       rc = gpio\_request(vbat\_gpio, "nfc\_vbat\_gpio");
48. +                       **if** (rc) {
49. +                               dev\_err(&client->dev, "unable to request vbat gpio\n");
50. +                       }
51. +
52. +                       dev\_err(&client->dev,"%s: vbat gpio before test value : %d \n", \_\_func\_\_,gpio\_get\_value(vbat\_gpio));
53. +
54. +                       rc = gpio\_direction\_output(vbat\_gpio, 1);
55. +                       **if** (rc) {
56. +                               dev\_err(&client->dev, "unable to set vbat gpio\n");
57. +                       }
58. +                       dev\_err(&client->dev,"%s: vbat gpio after test value : %d \n", \_\_func\_\_,gpio\_get\_value(vbat\_gpio));
59. +                       usleep\_range(10000, 11000);
60. +               }
61. +               vpad\_gpio = of\_get\_named\_gpio(np, "qcom,nq-vpad", 0);
62. +               **if** ((!gpio\_is\_valid(vpad\_gpio)))
63. +                       **return** -EINVAL;
64. +               rc = gpio\_request(vpad\_gpio, "nfc\_vpad\_gpio");
65. +               **if** (rc) {
66. +                       dev\_err(&client->dev, "unable to request vpad gpio\n");
67. +               }
68. +               dev\_err(&client->dev,"%s: vpad gpio before test value : %d \n", \_\_func\_\_,gpio\_get\_value(vpad\_gpio));
69. +               rc = gpio\_direction\_output(vpad\_gpio, 1);
70. +               **if** (rc) {
71. +                       dev\_err(&client->dev, "unable to set vpad gpio\n");
72. +               }
73. +               dev\_err(&client->dev,"%s: vpad gpio after test value : %d \n", \_\_func\_\_,gpio\_get\_value(vpad\_gpio));
74. +       }
75. +#endif
76. reg = devm\_regulator\_get(&client->dev, "vcc");
77. **if** (IS\_ERR(reg)) {
78. ret = PTR\_ERR(reg);
79. @@ -837,15 +907,18 @@ **static** **int** pca953x\_probe(**struct** i2c\_client \*client,
80. **else**
81. ret = device\_pca957x\_init(chip, invert);
82. **if** (ret)
83. -               **goto** err\_exit;
84. +               dev\_err(&client->dev, "pca953x init err: %d\n", ret);
85. +               //goto err\_exit;
87. ret = devm\_gpiochip\_add\_data(&client->dev, &chip->gpio\_chip, chip);
88. **if** (ret)
89. -               **goto** err\_exit;
90. +               dev\_err(&client->dev, "pca953x add gpiochip err: %d\n", ret);
91. +               //goto err\_exit;
93. ret = pca953x\_irq\_setup(chip, irq\_base);
94. **if** (ret)
95. -               **goto** err\_exit;
96. +               dev\_err(&client->dev, "pca953x irq setup err: %d\n", ret);
97. +               //goto err\_exit;

修改之后表现为kernel加载过程中，led亮起2-3s后灭掉,kernel加载完后检查led节点已被创建，但是不能通过节点控制led，I2C通信失败。

第16-75行代码，看起来是nfc\_pinctrl，但实际上也是tca6408的power enable pin。当时没有仔细研究这里的逻辑，导致后面多花了很多时间。

第83-85行代码，看起来是tca6408刚上电，第一次通信会失败，所以只是报错。加载的log如下。

1. [    1.206207] i2c-msm-v2 78b5000.i2c: probing driver i2c-msm-v2
2. [    1.206368] AXI: msm\_bus\_scale\_register\_client(): msm\_bus\_scale\_register\_client: Bus driver not ready.
3. [    1.206384] i2c-msm-v2 78b5000.i2c: msm\_bus\_scale\_register\_client(mstr-id:86):0 (not a problem)
4. [    1.206987] AXI: msm\_bus\_scale\_register\_client(): msm\_bus\_scale\_register\_client: Bus driver not ready.
5. [    1.207715] pca953x 1-0020: pca953x\_probe: vpad gpio before test value : 0
6. [    1.207739] pca953x 1-0020: pca953x\_probe: vpad gpio after test value : 1
7. [    1.207824] 1-0020 supply vcc not found, **using** dummy regulator
8. [    1.208192] i2c-msm-v2 78b5000.i2c: NACK: slave not responding, ensure its powered: msgs(n:2 cur:0 tx) bc(rx:1 tx:1) mode:FIFO slv\_addr:0x20 MSTR\_STS:0x011363c8 OPER:0x00000010
9. [    1.208239] pca953x 1-0020: failed reading **register**
10. [    1.208254] pca953x 1-0020: pca953x init err: -107
11. [    1.208796] pca953x probe finish

这两段代码实际可以使tca6408正常工作，但是后续不能正常工作，需要接着分析log.

log：BUS ERROR.

如果slave device没有连接，i2c会报NOACK。但是这边报的是I2C BUS error.

所以很自然认为BUS config有问题，所以去dts中比对bus node config。修改之后仍然有问题.

又去检查i2c bus gpio config.

cat /sys/kernel/debug/gpio

1. /sys/kernel/debug # cat gpio
2. gpiochip0: GPIOs 0-141, parent: platform/1000000.pinctrl, 1000000.pinctrl:
3. gpio0   : out 0 2mA pull down
4. gpio1   : in  0 2mA pull up
5. gpio2   : in  2 2mA no pull
6. gpio3   : out  2 2mA no pull
7. ……
8. gpiochip1: GPIOs 1016-1023, parent: i2c/1-0020, tca6408, can sleep:
9. gpio-1019 (                    |?                   ) out lo
10. gpio-1020 (                    |?                   ) out lo
11. gpio-1021 (                    |?                   ) out hi
12. gpio-1022 (                    |?                   ) out hi

I2C1是gpio2,gpio3.和旧项目对比旧项目都是out pin.

所以认为是gpio的配置被修改了。但是实际上这里的in/out只是表示GPIO direction register的值，如果function register不是gpio，dir-reg的值就不影响实际功能。这里的差异可能是因为lk的I2C1没有配置，所以和旧项目产生了差异。详细内容可以看：lk对I2C的配置流程。

这边认为是gpio2, gpio3的配置被修改了，所以在dts中找是否有其他的节点对gpio2/3的配置。

1. grep -nR "gpio2" arch/arm64/boot/dts/qcom/mitac/lytx
2. grep -nR "tlmm 2" arch/arm64/boot/dts/qcom/mitac/lytx
3. grep -nR "gpio3" arch/arm64/boot/dts/qcom/mitac/lytx
4. grep -nR "tlmm 3" arch/arm64/boot/dts/qcom/mitac/lytx

gpio3确认被占用了，修改之后仍然有问题.

因为led亮起2-3s左右就灭下去，所以认为是kernel加载阶段，gpio2,gpio3被占用了。kerne加载阶段的过程中，也会启动加载ADSP（ADSP的加载时间也需要研究一下）.所以认为可能是ADSP占用了gpio2,gpio3.检查ADSP gpio config,排除疑虑。

1. sensors/dsps/reg\_defaults/sensor\_def\_qcomdevl.conf
2. # SPI GPIO
3. 6340       1 0x00010002 # version
4. 6341  0xFFFF 0x00010003 # gpio num
5. 6342       1 0x00010002 # active func sel
6. 6343       1 0x00010002 # active dir
7. 6344       0 0x00010002 # active pull
8. 6345       3 0x00010002 # active drive
9. 6346       1 0x00010002 # inactive func sel
10. 6347       1 0x00010002 # inactive dir
11. 6348       2 0x00010002 # inactive pull
12. 6349       3 0x00010002 # inactive drive

因为一直认为是gpio2,gpio3被占用。但是有没有找到被占用的源。所以应该去读一下gpio2, gpio3的寄存器，看一下寄存器的实际状态。但是因为现在的上层代码编译不过，且编译方式和旧项目不同。

旧项目的r tool也不能使用。所以没去去读。

因为是I2C BUS error, 所以使用i2c tool debug的方法也不能用。

最后只能量测硬件信号。

量硬件，不一定需要量测示波器波形，而是从硬件的角度去思考IC的工作状态。

对于一颗IC，硬件需要量测的就是Power、输入、输出。如果power和输入都是正常的，那么IC就应该输出正常的信号。

注意：输入有时并不是单一的输入源，可能有多个输入共同作用产生一个输出源。

这次量测硬件的重点放在了I2C1 BUS上，忽略了TCA6408 IC. 实际上是TCA6408的power enable pin被占用导致TCA6408不能正常工作。

最后只能通过读寄存器的方式去看gpio被修改为什么功能。希望通过创建线程去读寄存器，但是失败了，可以再研究。

请求场外援助，编译r tool去读寄存器，确认gpio2, gpio3的function仍然是i2c。所以问题不在I2C BUS上。

这个时候才想到去确认TCA6408 IC的power.找到占用的地方，移除恢复正常。

LK 添加I2C

GPIOLIB

## 内核函数解析

### notifier chain

1. 什么是notifier chain

一般来说，kernel module 是相互独立的，notifier chain用于kernel module 间通信.

具体查看kernel/notifier.c include/linux/notifier.h

1. 有几种类型

atomic blocking raw srcu

原子(spinlock) 阻塞(rw\_sem) 原始(锁provided by caller) 阻塞变式，

具体查看include/linux/notifier.h

blocking raw比较常用

1. 初始化，调用，以及实例

<https://www.cnblogs.com/3me-linux/p/6122444.html>

<https://blog.csdn.net/u014134180/article/details/86563754>

实例查看drivers/video/fbdev/core/fb\_notify.c

1. 自己的理解

register notifier,是将notifier handler注册在notify head的链表下.

当调用notify call chain(nh,val,void \*),会逐个调用nh链表下的handler.

handler根据val 和 void \*选择不执行或者执行相应的func(switch val).

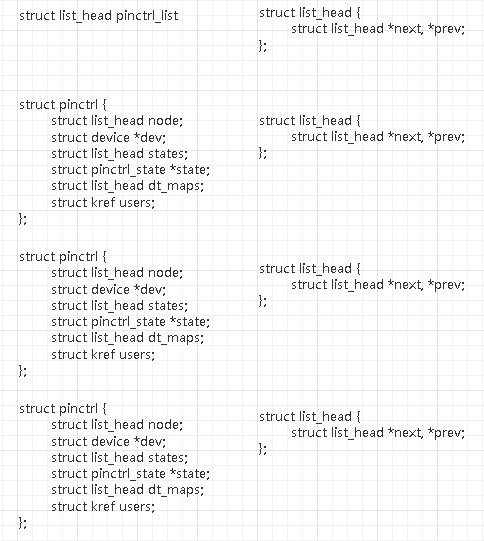
关键是通知者和被通知者需要获取同一个notifier head.

### list head

在对内核代码的学习中，经常对看到和list相关的宏，这对list进行拆解和学习。

以pinctrl为例。

1. **static** **struct** pinctrl \*find\_pinctrl(**struct** device \*dev)
2. {
3. **struct** pinctrl \*p;
5. mutex\_lock(&pinctrl\_list\_mutex);//对list加锁，这是很重要的
6. list\_for\_each\_entry(p, &pinctrl\_list, node)//解析
7. **if** (p->dev == dev) {
8. mutex\_unlock(&pinctrl\_list\_mutex);
9. **return** p;
10. }
12. mutex\_unlock(&pinctrl\_list\_mutex);
13. **return** NULL;
14. }
15. list\_for\_each\_entry(p, &pinctrl\_list, node)
16. 有很多独立的pinctrl结构体p，p通过list\_head结构体node,添加到pinctrl\_list链表中。所以可以根据pinctrl\_list遍历所有的pinctrl结构体。
17. **struct** pinctrl {
18. **struct** list\_head node;
19. **struct** device \*dev;
20. **struct** list\_head states;
21. **struct** pinctrl\_state \*state;
22. **struct** list\_head dt\_maps;
23. **struct** kref users;
24. };
25. **struct** list\_head {
26. **struct** list\_head \*next, \*prev;
27. };



那么添加list的函数呢

搜索pinctrl\_list

list\_add\_tail(&p->node, &pinctrl\_list);

### work queue

## vibrator

### driver no work?

c25 base line vibrator not worl.

1. check kernel node function.
2. check hardware lib 🡪 fail.
3. check service
4. check settings

## video

### vidc fail to suspend

Sprint suspend时会有概率遇到vidc suspend fail的问题。

vidc是什么？如何了解vidc?

1. 百度
2. 根据log定位.c文件查看相应注释//啥也没有
3. 查看.h文件，查看大致接口框架
4. 根据dts property查看Document目录下的文档
5. 查看高通相关文档
6. 问别人

最后得出vidc是一颗编解码芯片driver.

对于这一类没有时间（必要）去学习的driver，理解大致框架/功能，各阶段的初始化，主要是在什么时间会被调用。随什么操作而关闭即可。

<https://blog.csdn.net/xuecz1230/article/details/105814026>

<https://ciellee.blog.csdn.net/article/details/109627437>

看到log可以看到大量的

msm\_vidc: info: Opening video instance: 0000000000000000, 1

msm\_vidc: info: Closed video instance: 0000000000000000

去看代码，可以看到从msm\_vidc.c对应到msm\_v4l2\_vidc.c的open/close接口

所以vidc suspend应该是和close有关系。

在一次fail状况下，看到close video instance在fail to suspend后。

但是close 3s 后device才可以进入suspend。

所以推测vidc close 3s后可以正常suspend.对比正常的log, vidc close 远早于suspend 3s。所以猜测暂时成立。在msm\_v4l2\_vidc调用close接口后，在执行suspend的动作需要一点时间看起来也是比较合理的。

那么问题是vidc open/close是谁调用的呢？

观看sprint log, vidc open在panel on之后。

但是panel off后并没有close video.再断掉ACC，拔除USB，device进入suspend才会调用vidc close video.（是否是上层AP设置？是否是camera录影的设定?）

如果是这个逻辑，相同平台的其他设备应该也会有这个问题。

使用chiron pro测试，

在panel off的状态下，拔除USB，device调用suspend应该会出现这个问题。

但实际情况，拔除USB，device唤醒panel. 为什么呢？chiron pro的USB接口，同时具有供电的功能，所以拔插USB都会唤醒device。

Sprint 供电和USB接口是分开的，USB接口拔除不会唤醒设备。

Surfing pro也是power和USB分开的设计，所以插拔surfing pro的USB并不会唤醒设备。但是并没有复现bug, 同时surfing pro没有msm\_vidc的相关log。

现在的问题是

(1)vidc 和 panel没有关系，看起来和USB有关系?USB plug out, vidc close. Sprint panel on, vidc open.

(2)surfing pro usb in/out, panel still off.

(3)sprint usb in, panel on. panel off & usb out, panel still off.

(4)sprint usb in, panel on reason?

最后得出panel on reason is Vysor.

Vysor connect open vidc, Vysor disconnect close vidc.

使用surfing pro 连接Vysor去测试，成功复现，符合逻辑。

## usb

usb 基础概念

usb state

usb spec 9.1.1

### usb数据包解析

1. 编码方式

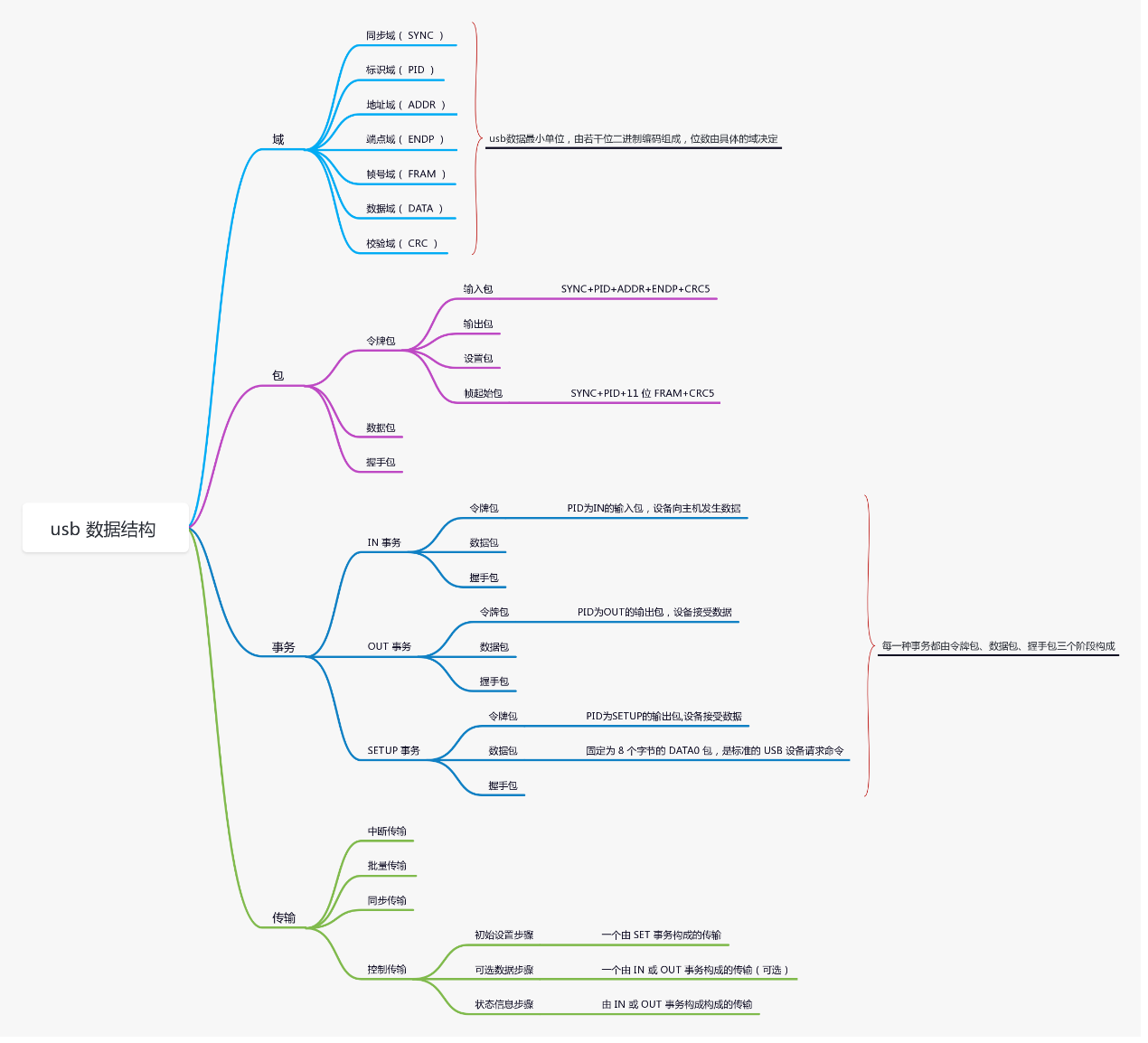
USB 采用不归零取反来传输数据，当传输线上的差分数据输入 0 时就取反，输入 1 时就保持原值。

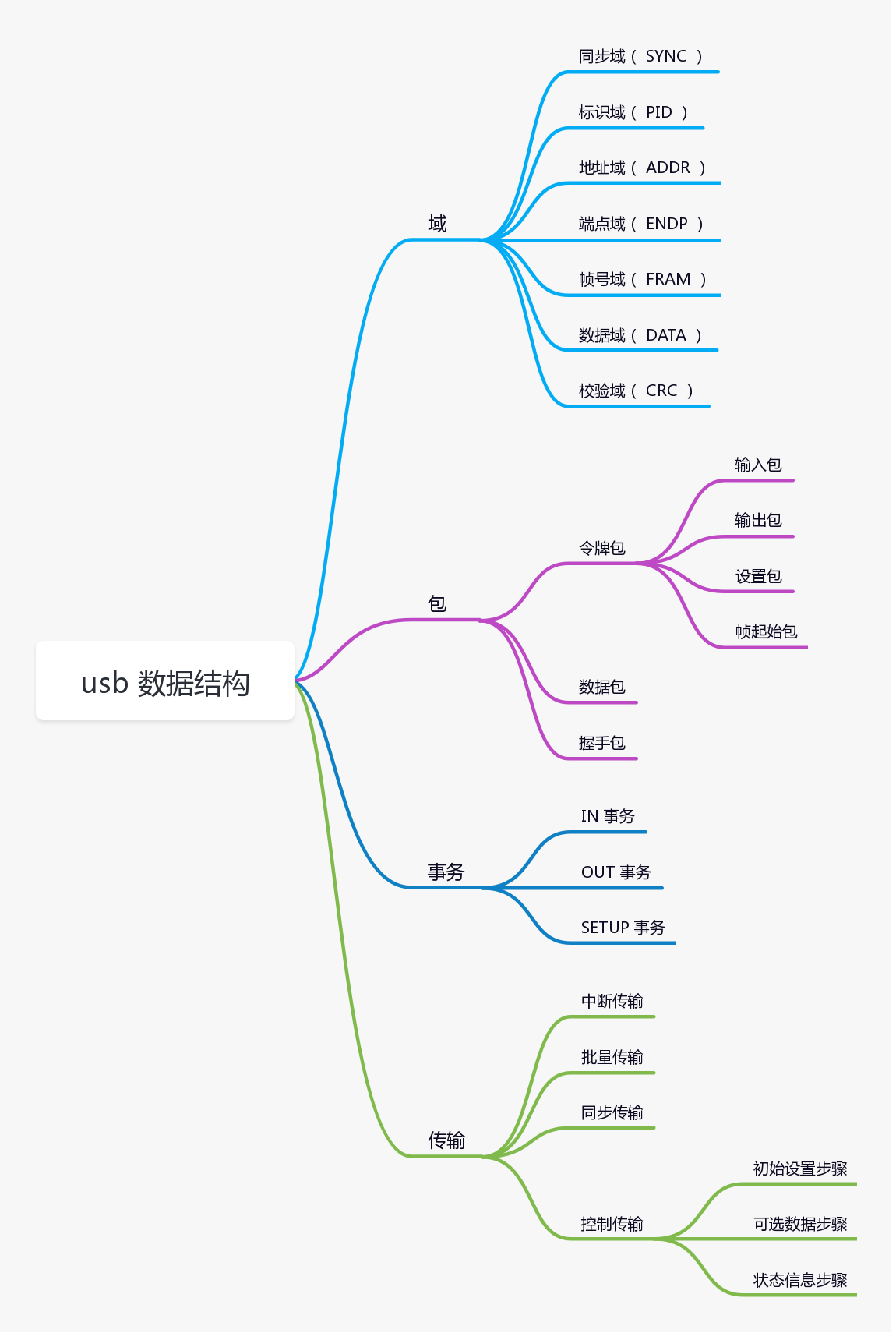
1. USB数据结构

USB 数据是由二进制数字串构成的，首先数字串构成域（有七种），域再构成包，包再构成事务（ IN 、 OUT 、 SETUP ），事务最后构成传输（中断传输、并行传输、批量传输和控制传输）。

<https://blog.csdn.net/yuanyou/article/details/5957609>

Chapter 8 Protocol Layer



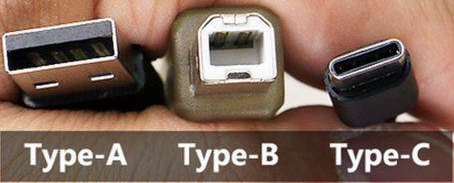


### USB枚举过程

### USB Connectors and Cables

Connectors：Plug Receptacle

Cable

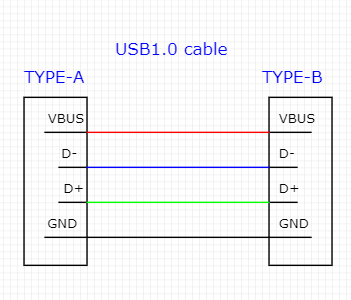


type-A, type-B, type-C

micro usb是mini usb的USB 2.0标准的下一代规范标准化。



micro USB(左) , mini USB(右)



我认为的USB 1.0 cable示意图

### usb2.0 OTG与connector block

OTG

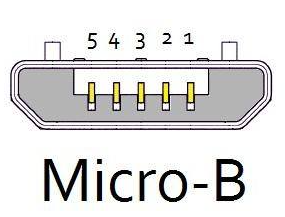
在usb2.0的基础上提出OTG概念.

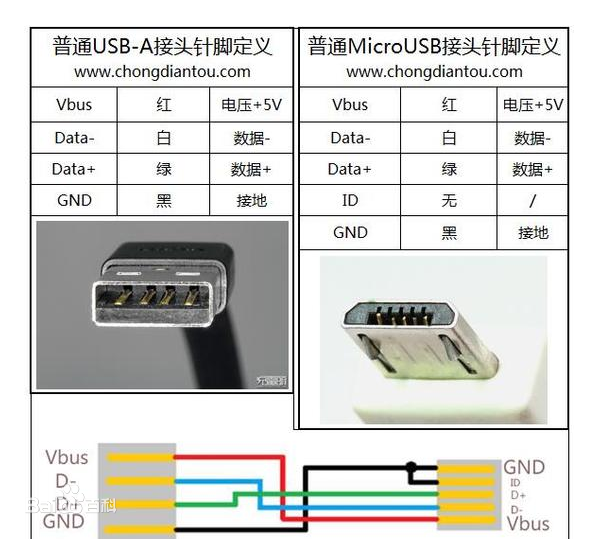
让一些设备既可以当host,也可以当client.

在connector上添加一根ID pin,来切换设备状态。

ID float,为client.

ID接地，为host.





### 如何区分USB2.0和USB3.0

1. 根据颜色区分

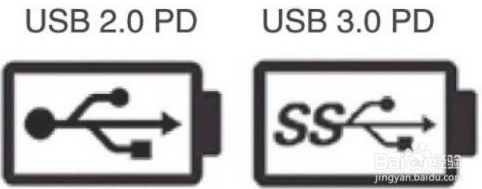
PC type A端口或cable线type A端口颜色

黑色或白色 USB2.0

蓝色 USB3.0

1. 根据标识区分

PC type A端口或cable线type A端口



1. 根据pin脚区分

USB 2.0 4 pin

USB 3.0 9 pin

type-A公口



1. 理论速度

USB 2.0 480Mbps 60MB/s

USB 3.0 5Gbps 625MB/s

1. 查看节点

cat sys/devices/platform/soc/7000000.ssusb/7000000.dwc3/udc/7000000.dwc3/current\_speed

1. 查看log

USB 2.0 high speed

USB 3.0 super speed

Ubuntu PC或Android device

usb3.0 RX/TX的作用

usb3.0增加了5pin 触点，在cable中增加了4条传输线。

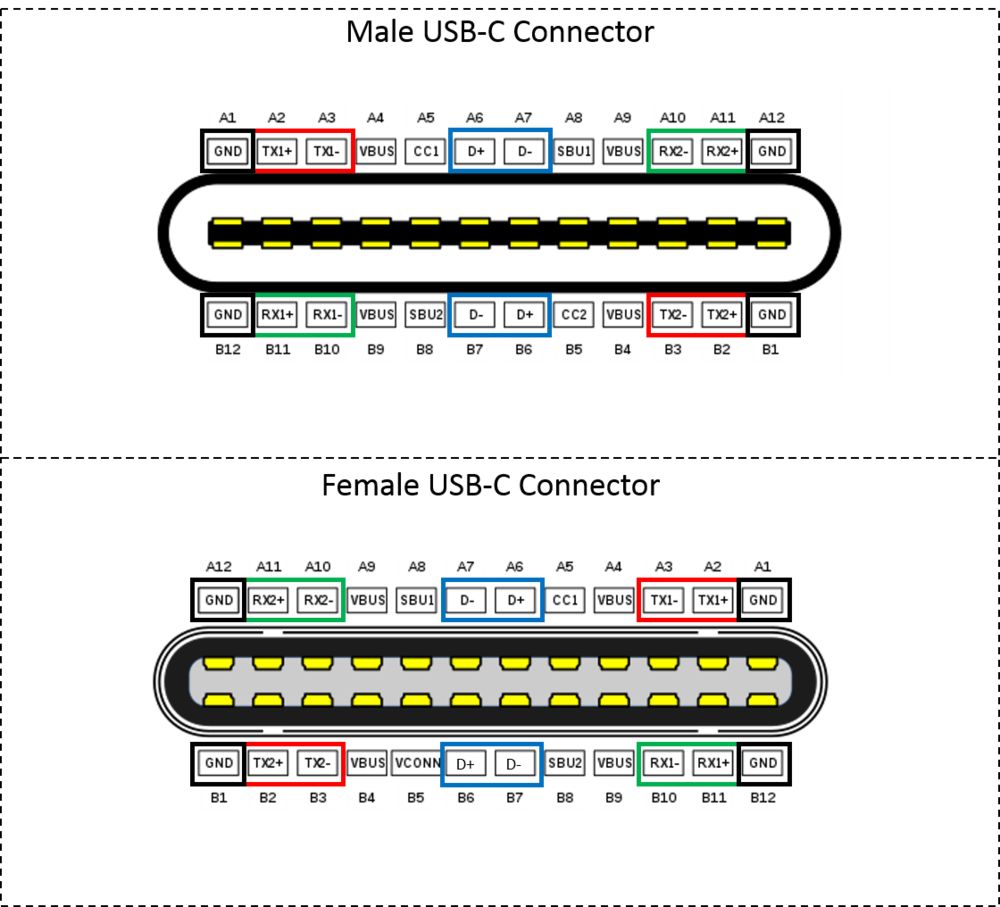
额外增加的4条（2对）线路提供了“SuperSpeed USB”所需带宽的支持，得以实现“超速”。

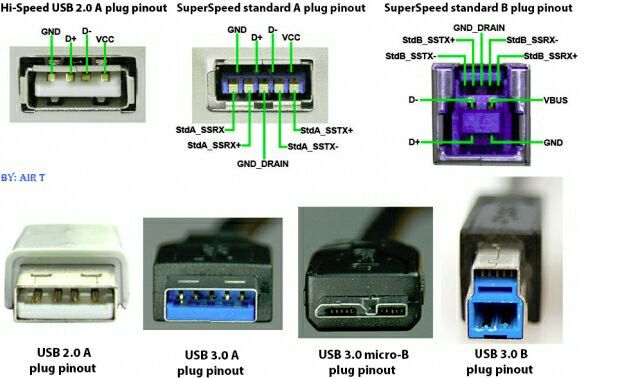
带宽和速度之间的关系:

usb3.0双向数据传输，2.0半双工。简化数据时间等待。

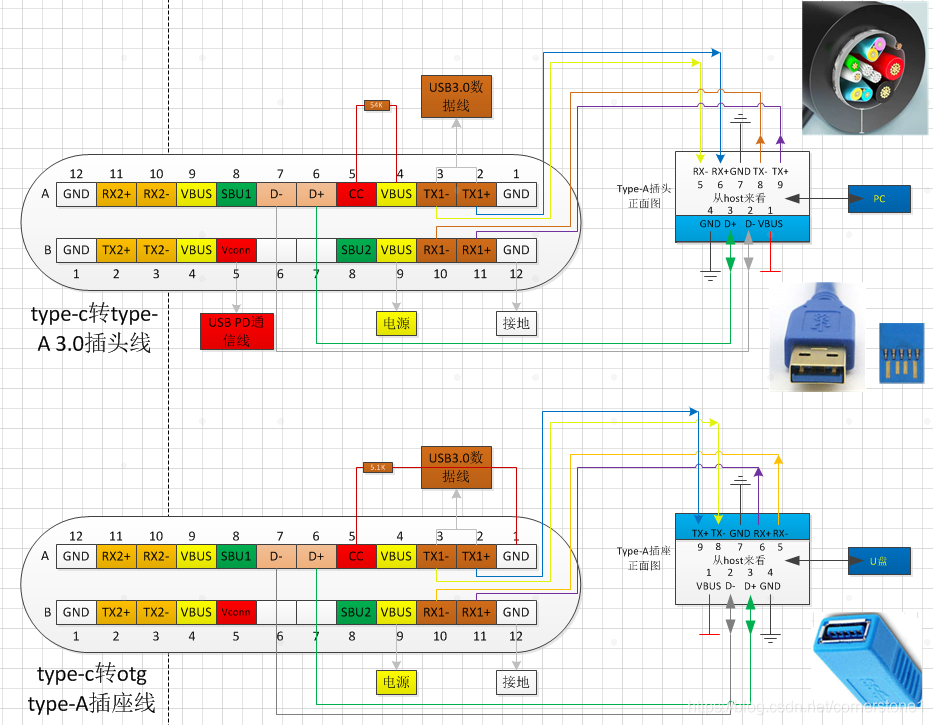
USB 3.0 block

type C 公母口





USB 3.0 type A



type-c cable block

### USB 枚举的过程以及BC协议

usb host 需要区分usb device的类型，如USB 2.0 device or USB 3.0 device.

usb device需要区分 host 端是PC or AC.

协议的流程先后顺序是什么。

### USB 眼图测试

项目初期需要过USB 眼图测试。

测试分为MB端信号测量，以及HUB端信号测量。HUB为EE指定HUB。

测试需要修改两版image。

测试的逻辑代码已经写好，需要根据测试端口的差异，修改判断条件。

if( hdev->maxchild > 1 ){//root hub下已挂载usb device的最大值

if(port1 == 1 ){

1. 在需要测试的port口连接client device.
2. 通过log确定port value.
3. 根据port value修改代码的判断逻辑
4. test image验证是否会打印test log.

USB port 序列号：

n-port:config.interface

：前表示usb device基于root hub的连接位置

：后表示usb device的配置接口

n表示USB总线，n>0

port表示USB总线的port口，port=0,表示总线自身的配置

port.port2表示hub的port2端口

config

interface

举例：

1-0:1.0 root hub1本身，config1,interface0

1-1:1-0 root hub1 port1接device,

1-1.4:1-0 root hub1 port1 -> port4接device(说明1-1是hub)

### d**iag p**ort enable

什么是diag port

bt和wifi的一些测试需要diagnostics port .

和usb 有什么关系

diag属于usb config的一种，一般charging mode是可以enable diag pory.

具体内容可以查看device/qcom/common/rootdir/etc/init.msm.usb.configfs.rc

对于android device的usb config

每一种config（MTP, PTP, chargind mode）都有自己的idVendor, idProduct

sys.usb.config也有不同的值

### usb host start流程

### usb extcon

extcon是一个driver 但本质上是具有group管理的notifiy.

如USB driver 有 EXTCON\_USB\_HOST和EXTCON\_USB

USB 注册调用链extcon\_register\_notifier

EXTCON 注册通知链头，INIT\_NOTIFIER\_HEAD

PMU 调用通知 extcon\_set\_state\_sync(…id…)

extcon\_sync --> raw\_nitifier\_call\_chain

### USB Ethernet test

tulkas2 的cradle上有一颗ethernet IC，现在需要将厂商提供的source code 编译成KO档，source code中另外添加了一个ioctl tool，用于修改IC MAC地址。

编译ioctl tool的流程：

1. 进入docker 环境,源码目录下。

执行make 命令报错，查看makefile 文件。单独执行

gcc ioctl.c –o ioctl

将编译生成的文件Push 到device中执行。

报错64-bit ELF file

1. docker环境下使用arm-gcc编译

编译报错，缺少32库文件

1. 给docker 环境添加lib32库文件

没做到

1. 在本机环境下编译

安装编译器：arm-linux-gnueabihf-gcc: hf对浮点的处理逻辑不一样，其他一样

安装库文件：lib32z1

本机编译：arm-xxx ioctl.c –o ioctl

将编译成功的库文件push到device中执行

报错No such file or directory

1. 和device中的可执行文件/system/bin比较
2. file ioctl
3. readelf –l ioctl

device端32-bit Elf文件为EXEC，本机编译生成文件类型为DYN

百度，静态编译的问题

arm-linux-gnueabihf-gcc --**static** ioctl.c -o ioctl

push测试，成功

## extcon driver

1. 什么是extcon driver

external connector class driver负责管理 外部连接器的检测。

通过注册相应的GPIO IRQ，去检测相应设备的拔插动作，并通知给相应的驱动。

<https://blog.csdn.net/qq_24622489/article/details/120436289?spm=1001.2014.3001.5501>

1. USB ID/VBUS IRQ是如何通知到对应的driver.

notifier head 是如何获取的.

1. 设备树，结构体，初始化，调用，以及例程
2. extcon-usb-gpio.c

extcon-usb-gpio.c是如何通知到usb otg driver?

extcon.c中注册notifier head，nh按照index区分不同的外部类型，如EXTCON\_NONE, EXTCON\_USB, EXTCON\_USB\_HOST

extcon-usb-gpio.c注册irq,irq触发notifier.

dwc3-msm.c 注册extcon\_register\_notifier.

## PMU

### PMU typec-or-rid-detect-change

USB 的detect 属于PMU管理，USB ID状态变化触发irq，PMU通知USB OTG driver.

### PMU 交接

<aside> 📌 錄影檔與相關文件下載點 <https://drive.google.com/drive/folders/1R1GrLAk6Fe74Y79iCHZR4r-bvNHsGBrj?usp=sharing>

</aside>

<aside> 📌 2.看睡睡醒醒的case 打印出哪個irq triggerd 以及增加APSS 沒睡的Clock狀態 解Vdd\_low

找範例

<http://10.88.25.195:18080/c/aiotbc/git/bsp/sc600_10/aosp/kernel/msm-4.9/+/1967>

</aside>

<aside> 📌 3. 軟件到底有沒有正確進入suspend(Vmin累加) <http://10.88.25.195:18080/c/aiotbc/git/bsp/sc600_10/aosp/kernel/msm-4.9/+/2036> <http://10.88.25.195:18080/c/aiotbc/git/bsp/sc600_10/aosp/kernel/msm-4.9/+/2086>

</aside>

<aside> 📌 1.解決wakelock自動打印問題 <http://10.88.25.195:18080/c/aiotbc/git/bsp/sc600_10/aosp/kernel/msm-4.9/+/2144> <http://10.88.25.195:18080/c/aiotbc/git/bsp/sc600_10/mitac/device_mitac_hera_pro/+/2150> <http://10.88.25.195:18080/c/aiotbc/git/bsp/sc600_10/aosp/platform/frameworks/base/+/2146> <http://10.88.25.195:18080/c/aiotbc/git/bsp/sc600_10/aosp/device/qcom/sepolicy/+/2147>

<http://10.88.25.195:18080/c/aiotbc/git/bsp/sc600_10/aosp/platform/frameworks/base/+/2918>

userspace <http://10.88.25.195:18080/c/aiotbc/git/bsp/sc600_10/aosp/kernel/msm-4.9/+/3349> <http://10.88.25.195:18080/c/aiotbc/git/bsp/sc600_10/aosp/platform/frameworks/base/+/3350>

</aside>

<aside> 📌 Power profile <http://10.88.25.195:18080/c/aiotbc/git/bsp/sc600_10/mitac/device_mitac_hera_pro/+/2761> <https://source.android.com/devices/tech/power>

</aside>

## battery

### 根据一个节点去追函数

recovery 在OTA升级时会报battery capacity 0,导致升级失败

从recovery代码分析，最后确定节点是/sys/class/power/supply/battery/capacity

1. grep –nR “capacity” driver/power/supply

grep –nR “capacity” driver/power/supply/qcom

capacity换成battery

根据关键字去筛选

1. ls –lh /sys/class/power/supply/battery

device -> ../../../\*\*\*qcom,qpnp-smb5

定位到drivers/power/supply/qcom/qpnp-smb5.c

查找/capacity

1. battery\_capacity\_show //查看代码逻辑排除
2. smblib\_get\_prop\_batt\_capacity //加log逐级确认,也不是这里

smblib\_set\_prop\_batt\_capacity

在qpnp-smb5.c中查找capacity, battery, batt, get/set, show/store之类的关键字

找到smb5\_batt\_get\_prop { case POWER\_SUPPLY\_PROP\_CAPACITY: ….. }

判断代码逻辑以及添加log确认smb5\_batt\_get\_prop，但这并不是完全的入口。

/smb5\_batt\_get\_prop

得到struct power\_supply\_desc batt\_psy\_desc = {

.get\_property = smb5\_battert\_get\_prop,

}

/batt\_psy\_desc

得到devm\_power\_supply\_register(…, &batt\_psy\_desc, …);

追上面函数到drivers/power/supply/power\_supply\_core.c

/get\_property

得到power\_supply\_get\_property

代码逻辑加log确认符合

1. power\_supply\_get\_property & POWER\_SUPPLY\_PROP\_CAPACITY

grep 信息太多不好继续在向上追

power\_supply\_get\_property添加WARN\_ON(1)

log太多，系统加载不起来

smb5….case CAPACITY:添加WARN\_ON(1)

得到smb5\_batt\_get\_prop

power\_supply\_get\_property

power\_supply\_show\_property

dev\_attr\_show

1. power\_supply\_show\_property

在drivers/power/supply/power\_supply\_sysfs.c

内部逻辑创建了一系列的节点，统一规范…

不再深追逻辑

## pm\_runtime

<https://www.cnblogs.com/hellokitty2/p/9974273.html>

怎样动态地打开关闭设备的电源 ？ 最简单的方法：在驱动程序中，open时打开电源，在close时关闭电源。但是有一个缺点，当多个App使用该设备时可能造成干扰。  
解决方法：给驱动添加计数值，当该值大于0时打开电源，等于0时关闭电源。

runtime PM只是提供辅助函数，比如：  
(1).增加计数/减少计数  
(2).使能runtime pm

最好的资料是runtime\_pm.txt  TODO：翻译它

例子：\drivers\input\misc\bma150.c

pm\_runtime\_enable //bma150\_probe

pm\_runtime\_disable //bma150\_remove

pm\_runtime\_get\_sync //bma150\_open

pm\_runtime\_put\_sync //bma150\_close

## struct device/device tree

* 1. node与property之间的关系
  2. node与函数之间的关系
  3. node转换成结构体的流程

kernel中相同的内容重复出现，

有整个文件替代，后续补充重复不覆盖，后续补充重复覆盖，

DT属于覆盖，但是哪些文件加载哪些文件不加载需要厘清

### devm架构

12/30

有一些以devm开头的申请资源的函数

devm\_kzalloc(dev,size);

alloc\_dr(release,size,gfp,node)

devres\_add

devm\_get\_gpiod\_from\_child(dev,con\_id,child);

1.devres\_alloc(release,size,flag)

2.devres\_add(dev,devres)

1.将申请的资源与device绑定，驱动注册失败时调用release函数

驱动注册失败会进入bus\_remove\_driver…

2.将资源添加到dev->devres链表中，释放时根据链表释放。

devm相关代码在各级目录的devres.c中，了解相应的部分最好是去追一遍代码，si有些函数追不了，用ctags.

### of\_node/fwnode

12/31

源码注释:

\* @of\_node: Associated device tree node.

\* @fwnode: Associated device node supplied by platform firmware.

struct device\_node \*of\_node; /\* associated device tree node \*/

struct fwnode\_handle \*fwnode; /\* firmware device node \*/

这个注释太抽象了，设备树的相关节点。

但是struct devices\_node 中 有 struct fwnode\_handle

这一部分没有验证，打印两个地址就可以确认了。

但是从代码的角度确认是父子关系。

所以使用of函数搜node,或者是fwnode函数搜fw\_node都是一样的。

看个人喜好。

fwnode\_get\_named\_gpiod(fwnode,propname)

of\_get\_named\_gpiod\_flags(to\_of\_node(fwnode), propname, 0, &flags);

父节点子节点嵌套在of\_node结构体中

### 创建文件节点的几种方法

12/22

1. device\_creat
2. DEVICE\_ATTR

ssize\_t show(); 🡪cat

ssize\_t store(); 🡪echo

static DEVICE\_ATTR(name,0644,show,store); //权限不对,会编译错误

在probe中添加device\_create\_file(dev,dev\_attr\_name);

创建的文件节点在sys/devices/soc/soc:dev\_name/name

如何把文件创建在syc/class中？

## buzzer & gp\_clk

### Analog PWM to driver buzzer

12/17

在pwm资源不够用时，需要使用其他方法去模拟一段有频率有占空比的波形。

1. 首先确认cpu端口是否有其他资源引脚可以作为pwm引脚。  
   高通的平台一般都是PMU输出pwm.也需要仔细查找。
2. 使用gpio模拟pwm,最简单可行的方法。但是频率不可控，占用cpu等。

使用线程控制gpio输出pwm信号。

<https://blog.csdn.net/u010632165/article/details/89722539#_11>

1. 使用GP\_CLK时钟信号模拟。

<https://blog.csdn.net/u010787514/article/details/82463356>

<https://blog.csdn.net/qq_35141454/article/details/101545510>

1. 使用hrtimer高精度定时器模拟。

<https://blog.csdn.net/feixiangtiakongn/article/details/103909288?utm_medium=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-1.control&depth_1-utm_source=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-1.control>

<https://blog.csdn.net/qq_35141454/article/details/101428415>

什么样的IO可以代替PWM，第一可以输出频率，是否可以修改duty；第二，输出电流是多少，是否符合要求。Cyclo的buzzer使用了一个放大电路，应该注意输入是否符合电路要求。

### GP\_CLK

1/15/2021

1. dts 添加pinctrl,
2. dts 添加设备节点引用pinctrl

-clocks = <&clock\_gcc clk\_gp3\_clk\_src>;

+clcoks = <&clock\_gcc clk\_gcc\_gp3\_clk>;

1. clock-gcc-target.c配置clk

F(freq,s,div,M,N);

F(5000, xo, 16, 1, 240)

N的参数是高通使用tool计算出来的，上面的引用节点也是高通给的。

debug clk

dir

/sys/kernel/debug/clk/gcc\_gp2\_clk

enable

/sys/kernel/debug/clk/gcc\_gp2\_clk/enable

rate

/sys/kernel/debug/clk/gcc\_gp2\_clk/rate

对于自己不懂得东西还是多问问比较好，对于clock部分还可以追指针函数的源码

# 代码是如何被编译/执行的

## Makefile

:= = +=

立即赋值 mk结束后赋值 add结束后赋值

一次赋多值 用空格间隔开，常加\表示换行

makefile函数

foreach

$(foreach <temp\_val>,<list>,<text>)

eg：

names := a b c d

files := $(foreach n,$(names),$(n).o)

files的值”a.o b.o c.o d.o”

filter

$(filter pattern…,list)

保留符合pattern的list.

sources := foo.c bar.c baz.s ugh.h   
files:=  $(filter %.c %.s,$(sources))

files的值” foo.c bar.c baz.s”

eval

$(eval text)

是 text 的内容将作为makefile的一部分而被make解析和执行。

call

if and or

subst

dir notdir

basename

Makefile, Kconfig, .config, obj的编译关系

一般关系

Kconfig

depends on

A depends on B表依赖。CONFIG\_A 的前提是CONFIG\_B

select

A select C 表选择。CONFIG\_A就会CONFIG\_C.

makefile 会说明编译关系

Kconfig会说明依赖关系

编译生成的.config文件在 obj/kernel/msm-4.9/.config

--------------------------------------------------------------------------------------------------------------------------------

启动解析

cmdline、init进程

验证一个driver是否加载，以及加载时间。

1.查看out.../obj/KERNEL\_OBJ/drivers目录下相应的\*.o文件

2.查看对应的defconfig文件

以chiron\_pro\_a10为例

arch/arm64/configs/chiron\_pro\_a10-perf\_defconfig //for user image build config -b user

arch/arm64/configs/chiron\_pro\_a10\_defconfig //for user-debug image

arch/arm64/configs/msm8953-perf\_defconfig //no use

arch/arm64/configs/msm8953\_defconfig //no use

arch/arm/configs/msm8953-perf\_defconfig //no use

arch/arm/configs/msm8953\_defconfig //no use

选用对应的defconfig可以查看AndroidBoard.mk

device/mitac/chiron\_pro/AndroidBoard.mk

ifeq ($(TARGET\_BUILD\_VARIANT),user)

KERNEL\_DEFCONFIG := chiron\_pro\_a10-perf\_defconfig

else

KERNEL\_DEFCONFIG := chiron\_pro\_a10\_defconfig

defconfig没有合并覆盖的逻辑，只有一个文件决定driver的配置。

:=和=

=会在makefile文件展开后决定变量的值，即最后决定：a=1,b=$(a),a=2 ==> b=2;

:=使用当前的值。b=1;

3.查看System.map符号表

4.driver加载的顺序

1).与defconfig无关

2).与Makefile的顺序有关

3).与driver初始化段有关 module\_init call6.

## BSP的初始配置

kernel加载有两个关键点：

1. 编译时如何选择config文件

2. 加载时如何选择对应的dts

添加config文件

<http://10.88.26.86:18080/c/aiotbc/git/bsp/sc606t/linux/poky/meta-qti-bsp/+/73>  
<http://10.88.26.86:18080/c/aiotbc/git/bsp/sc606t/linux/kernel/msm-4.9/+/72>

添加dts

<http://10.88.26.86:18080/c/aiotbc/git/bsp/sc606t/linux/kernel/msm-4.9/+/71>

ADSP

<http://10.88.26.86:18080/c/aiotbc/git/bsp/sc606t/linux/sensors/+/106>

## kernel是如何加载各个driver

## init进程

### property

android property机制

https://blog.csdn.net/stoic163/article/details/88815430

2.3 HAL

hardware 功能检测

## device目录

### overlay机制

1.overlays机制

1)

PRODUCT\_PACKAGE\_OVERLAYS

DEVICES\_PACKAGE\_OVERLAYS

优先级PRODUCT > DEVICE

2)

OVERLAYS := file1 file2

优先级file1 > file2

<https://blog.csdn.net/the_Sunshine_of_King/article/details/56488563?utm_medium=distribute.pc_relevant.none-task-blog-2%7Edefault%7EBlogCommendFromMachineLearnPai2%7Edefault-1.control&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2%7Edefault%7EBlogCommendFromMachineLearnPai2%7Edefault-1.control>

AAPT android资源打包机制

aidl

### PRODUCT\_COPY\_FILE

解析过程build/core/Makefile

### config.xml

1/12/2021

config.xml的加载顺序，以autoBrightnessLevels为例

首先注册符号

frameworks/base/core/res/res/values/symbols.xml:1828: <java-symbol type="array" name="config\_autoBrightnessLevels" />

1.framworks层

core/res/res/values/config.xml

2.device/common层device/qcom/common/device/overlay/frameworks/base/core/res/res/values/config.xml

3.device/company

device/mitac/chiron\_pro/overlay/frameworks/base/core/res/res/values/config.xml:105: <integer-array name="config\_autoBrightnessLcdBacklightValues">

3>2>1

加载顺序应该是类似设备树，覆盖补充加载。

还有两个目录也会有可能出现，应该都是common属性，客制化属性时不考虑

device/qcom/common/automotive/device/overlay/frameworks/base/core/res/res/values/config.xml

vendor/qcom/proprietary/resource-overlay/common/Frameworks/res/values/config.xml

和import android.os.Systemproperties有关系吗？

2.10 OUT

--------------------------------------------------------------------------------------------------------------------------------

CTS测试

1. 安装java环境
2. 安装Android Studio
3. 安装SDK

https://blog.csdn.net/u012764358/article/details/107049333/

https://blog.csdn.net/weixin\_39529302/article/details/114606435

https://blog.csdn.net/qq\_22948593/article/details/109957099

# Android框架和java 语法

<https://developer.android.google.cn/guide>

## Intent

intent是一个对象，用于传递消息的对象。intent可以传递消息给谁，可以传递什么消息？

在追溯代码的过程中，又如何定位到传递的对象？

Intent的基本用途主要包括：

启动 Activity、启动服务、传递广播。

Intent对象可以封装传递下面6种信息：

1. 组件名称（ComponentName）
2. 动作（Action）
3. 种类（Category）
4. 数据（Data）
5. 附件信息（Extra）
6. 标志（Flag）

123用于区分Activity, Service, BroadcastRecvier

456是用于传输数据

关于组件，通俗的解释可能就是对象。这个还是需要通过实例去理解此概念。

如何定位传递的对象？

实例1:

1. **private** **void** doMasterClear() {
2. Intent intent = **new** Intent(Intent.ACTION\_FACTORY\_RESET);
3. intent.setPackage("android");
4. intent.addFlags(Intent.FLAG\_RECEIVER\_FOREGROUND);
5. intent.putExtra(Intent.EXTRA\_REASON, "MasterClearConfirm");
6. intent.putExtra(Intent.EXTRA\_WIPE\_EXTERNAL\_STORAGE, mEraseSdCard);
7. intent.putExtra(Intent.EXTRA\_WIPE\_ESIMS, mEraseEsims);
8. getActivity().sendBroadcast(intent);
9. // Intent handling is asynchronous -- assume it will happen soon.
10. }

首先是追Intent.ACTION\_FACTORY\_RESET的变量类型和值。

找到**new** Intent执行的对应函数。

ACTION\_FACTORY\_RESET的类型是string, 值是"android.intent.action.FACTORY\_RESET";

new Intent相当于setAction()，最终执行action.intern(). action的值是string.

action 是一个动作参数，在AndroidManifest.xml声明组件时，会设置action参数。

AndroidManifest.xml全路径：frameworks/base/core/res/AndroidManifest.xml

搜索得到

1. **<receiver** android:name="com.android.server.MasterClearReceiver"
2. **<action** android:name="android.intent.action.FACTORY\_RESET" **/>**

com.android.server.MasterClearReceiver

frameworks/base/services/core/java/com/android/server/MasterClearReceiver.java

1. **public** **void** onReceive(**final** Context context, **final** Intent intent) {

setAction对应getAction

putExtra对应getBooleanExtra，getStringExtra

setPackage, addFlags在此例中没有解析，暂时不分析。

getActivity().sendBroadcast(intent); 理解为发送internt的意思，功力不够，暂时不深入分析

## android 四大组件

活动（activity），用于表现功能；

服务（service），后台运行服务，不提供界面呈现；

广播接受者（Broadcast Receive），用于接收广播；

内容提供者（Content Provider），支持多个应用中存储和读取数据，相当于数据库。

# ISSUE

* 1. property get fail?

persist.sys.display.default.rotation

Bootanimation.cpp property\_get

PhoneWindowManage.java SystemProperties.getInt

init.target.rc setprop 0

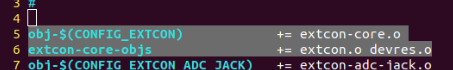
setprop 3

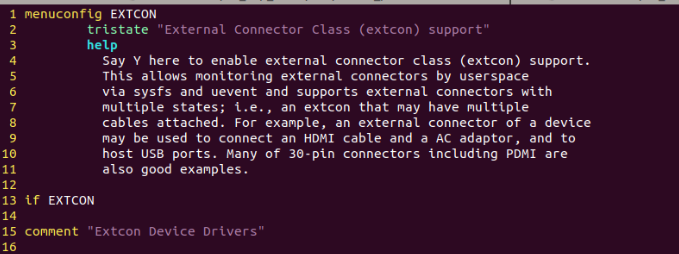
BootAnim get 0, JAVA get 3

* 1. 如何确认一个文件是否被编译?

以extcon driver为例.

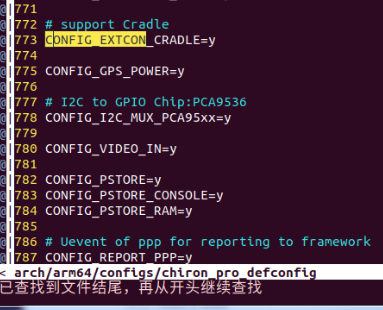
1. 查找Makefile以及Kconfig文件，确认编译文件对应的config项.





1. 确认编译对应的defconfig文件

arch/arm64/config/chiron\_pro\_defconfig



一般情况是可以找到CONFIG\_EXTCON=y,但是这里并没有

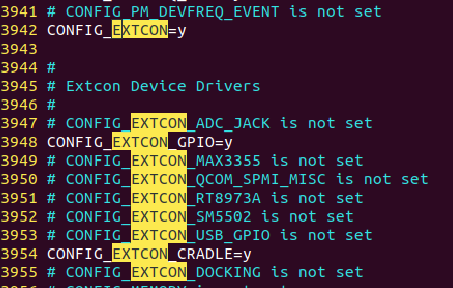
可以去确认其他driver Kconfig文件是否有select EXTCON.

grep -nR “select EXTCON” --include=Kconfig

1. 确认.config文件

obj/kernel/msm-4.9/.config

(obj/KERNEL\_OBJ link to obj/kernel/msm-4.9)



类似的文件

obj/kernel/msm-4.9/ include/config/auto.conf

obj/kernel/msm-4.9/ include/config/ tristate.conf

config转化成C语言可识别的宏:

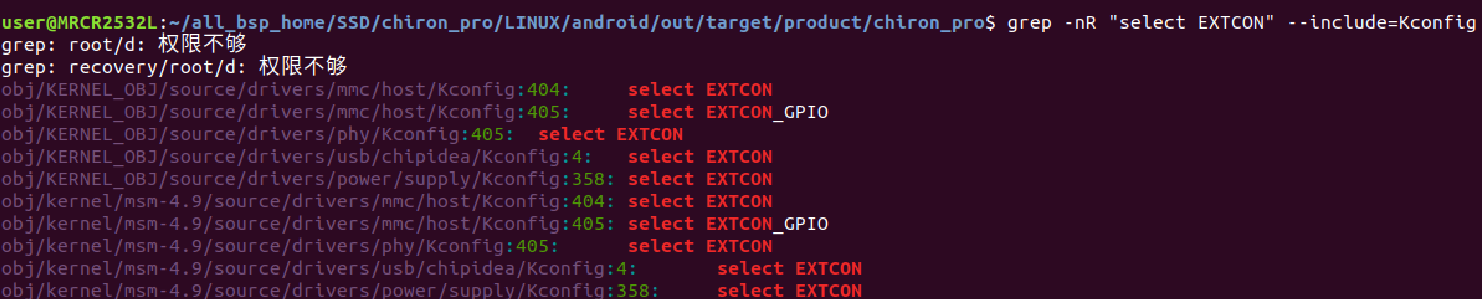
obj/kernel/msm-4.9/include/generated/autoconf.h

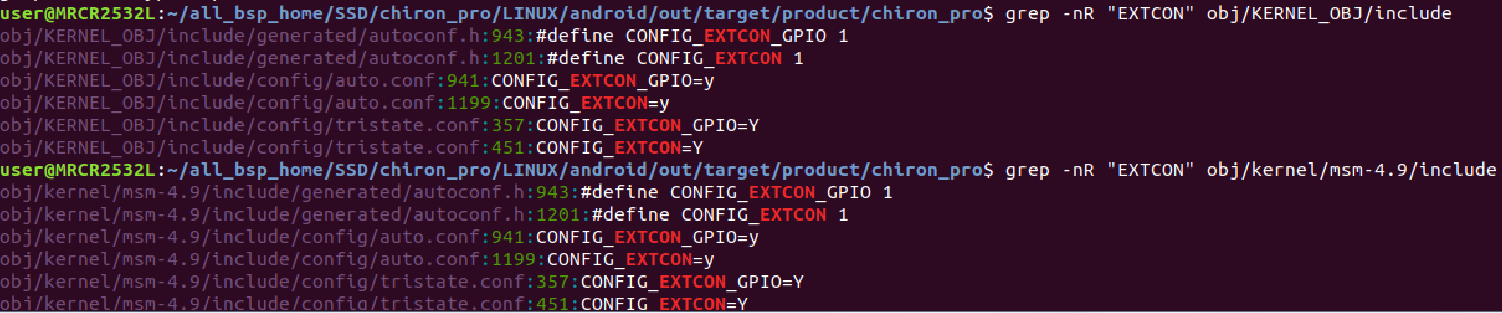
也可以对比旧的config文件看看是否有变化

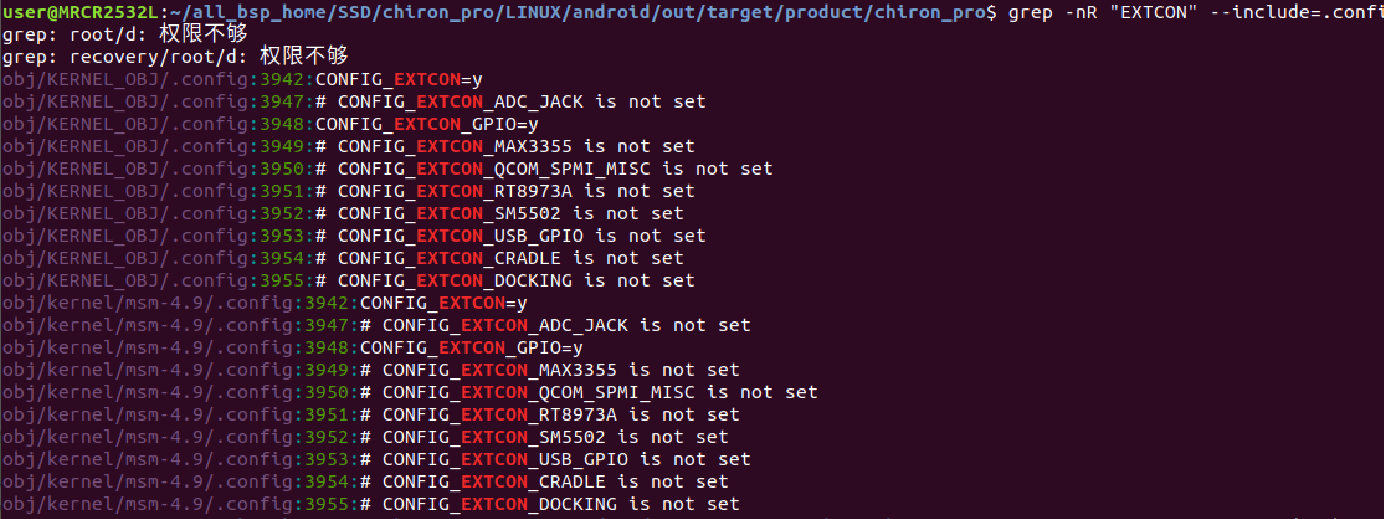
obj/kernel/msm-4.9/.config.old

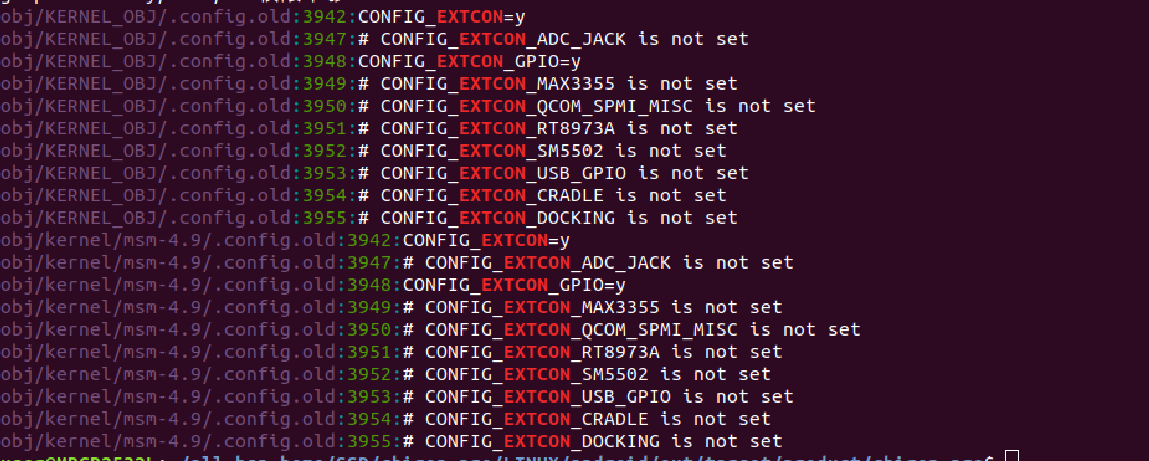
1. 确认编译生成相应的.o文件

ls obj/kernel/msm-4.9/driver/extcon/









* 1. 如何检查一个driver的加载时间?

节点权限问题总结

1/19/2021

1. kernel层创建节点

static DEVICE\_ATTR(node\_name,0664,show,store);

show函数对应cat,read函数,

得到buf的内容

store函数对应echo,write函数,

buf写入的参数，count是buf的长度。

返回值如果小于count,show函数会重新读count-ret字节大小的剩下buf.

1. device层节点加权限

在init.target.rc中添加相应的权限，一般都是

on boot

chown system system /sys/devices/soc/soc:buzzer/buzzer

chmod 0660 /sys/devices/soc/soc:buzzer/buzzer

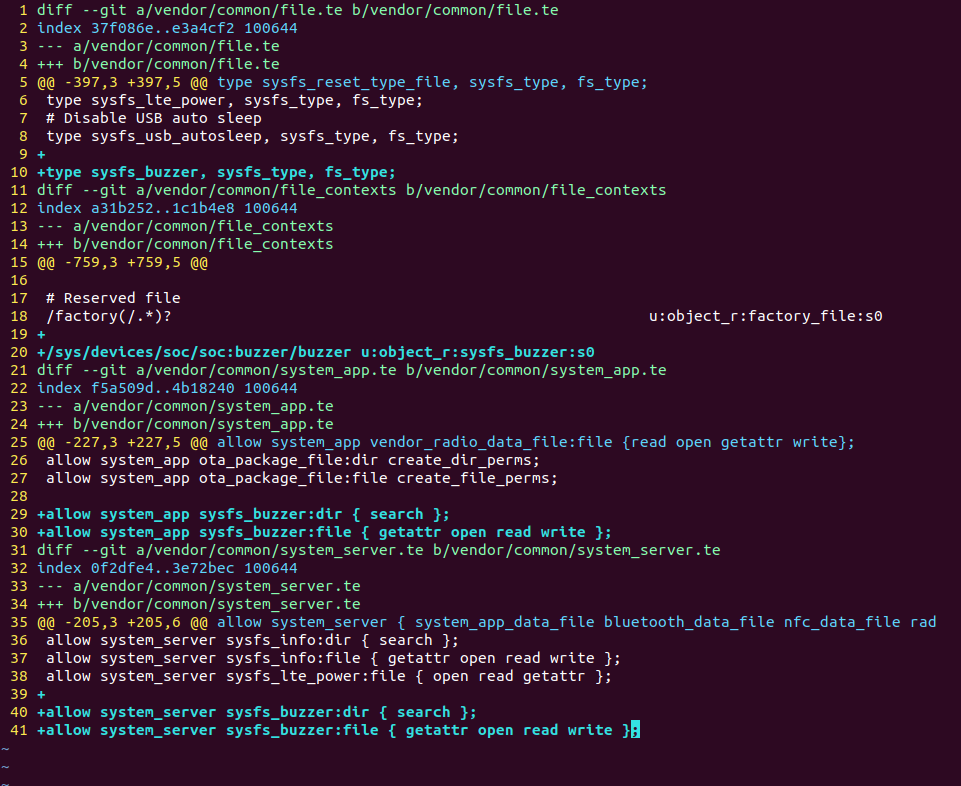
init.target.rc一般修改/device/company\_name/project\_name/init.target.rc

init.tarfet.rc区分project和soc.

on boot是init.target.rc的执行条件

1. sepolicy添加system\_app权限

sepolicy可以添加很多权限，但是可能会引起GMS认证，never\_allow等错误

一般是给节点加system\_server,system\_app权限，再给app添加系统权限。

1. android stuio给apk添加系统权限

platform.pk8和platform.x509.pem生成keystore系统签名文件，编译时自动签名。

1. 下载keytool-importkeypair工具

[https://github.com/getfatday/keytool-importkeypair](https://link.jianshu.com/?t=https%3A%2F%2Fgithub.com%2Fgetfatday%2Fkeytool-importkeypair)

1. ./keytool-importkeypair -k ./release.keystore -p android -pk8 platform.pk8 -cert platform.x509.pem -alias platform

-k 表示要生成的 keystore 文件的名字，这里命名为 release.keystore

-p 表示要生成的 keystore 的密码，一般就用android

-pk8 表示要导入的 platform.pk8 文件

-cert 表示要导入的platform.x509.pem

-alias 表示给生成的 release.keystore 取一个别名，这是命名为 platform

将项目中的platform.pk8和platform.x509.pem copy到keytool目录下，

文件路径在bsp/LINUX/android/build/target/product/security/

<https://www.mobibrw.com/2015/2826>

<https://blog.csdn.net/weixin_33804990/article/details/89617190>

1. 在android studio中使用签名，

在项目目录右键open module setting ->signing configs导入keystore

也可以在Gradle Scripts->build.gradle(app)添加代码

<https://blog.csdn.net/u011904605/article/details/53148683>

NavigationBarPosition

issue:triton + region image navigation在display size为large时，

getResources().getDisplayMetrics()判断navgiation在右侧

nvgationbarPosition

services/core/java/com/android/server/policy/PhoneWindowManager.java

navigationBarPosition函数中判断position位置是mNavigationBarCanMove，

CanMove也会决定getDisplayMetrics获取到屏幕尺寸。

mNavigationBarCanMove = width != height && shortSizeDp < 600;

还是需要多看源码，多动手。

getResources().getDisplayMetrics()

getRealMetrics获取屏幕的分辨率

getMetrics获取屏幕分辨率-navigationbarPosition

可以在android.util.DisplayMetrics中查看相关源码，如果去追这一部分代码，可以看到最后只是返回了R.height,R.width。具体这个类是怎么填充的看不到。

canMove决定Metrics应该还是从PhoneWindowManage处去追。

LCD driver ic(ILI9881C)

command

写command需要page + address

auto brightness自动亮度调节

1. 分析流程

settings代码目录 package/apps

<https://blog.csdn.net/lilidejing/article/details/45022697>

<https://blog.csdn.net/decisiveness/article/details/51899173>

GPIO

KBA-190803031133

performance

list of URL

<https://www.kernel.org/>

<https://www.kernel.org/doc/Documentation/>

<http://ilinuxkernel.com/?cat=3>