



Amlogic Advanced SoC Solution for Tablet / PMP Products

- January 2011

目录:


- Chapter One: Amlogic AML8726-M brief
- Chapter Two: Openlinux应用
- Chapter Three: 版本发布ppt
- Chapter Four: u-boot培训文档
- Chapter Five: LCD_Tuning_on_M1
- Chapter Six: 软件相关硬件培训
- Chapter Seven: 触摸屏
- Chapter Eight: 指南一 按键、电源部分介绍
- Chapter Nine: 指南二 添加自己的项目相关内容
- Chapter Ten: 指南三

-
- Chapter Eleven: 指南四 扩展外设的应用及实践
 - Chapter Twelve: 指南五 应用开发和实践
 - Chapter Thirteen: Andriod 四大组件

Distribute to Skynoon!

**It can not be distributed without authorization
of Amlogic**

Distribute to Skelnoon!



Chapter One:

Amlogic AML8726-M brief

(Arvin Zuo)

Distribute to Skynoon!



ARM Cortex-A9

1 目前最先进的推测型八级流水线，该流水线具有高效、动态长度、多发超标量及无序完成特征，这款处理器的性能、功效和功能均达到了前所未有的水平，完全能够满足消费、网络、企业和移动应用等领域尖端产品的要求。

2 ARM v7架构

3 可扩展的Cortex-A9 MPCore™多核处理器和较为传统的Cortex-A9单核处理器

Cortex-A9处理器最主要的流水线性能:

- 4 先进的取指及分支预测处理，可避免因访问指令的延时而影响跳转指令的执行
- 5 最多支持四条指令**Cache Line**预取挂起，这可进一步减少内存延时的影响，从而促进指令的顺利传输
- 6 每个周期内可连续将两至四条指令发送到指令解码，确保充分利用超标量流水线性能。**Fast-loop**模式：执行小循环时提供低功耗运行
- 7 超标量解码器可在每个周期内完成两条完全指令的解码

Cortex-A9处理器最主要的流水线性能：

- 8 支持指令预测执行：通过将物理寄存器动态地重新命名至虚拟寄存器池来实现
- 9 提升了流水线的利用效率，消除了相邻指令之间的数据依赖性，减少了中断延时
- 10 支持寄存器的虚拟重命名：以一种有效的、基于硬件的循环展开方法，提高了代码执行效率，而不会增加代码大小和功耗水平
- 11 四个后续流水线中的任何一个均可从发射队列中选择执行指令—提供了无序分配，进一步提高了流水线利用效率，无需借助于开发者或编译器指令调度。确保专为上一代处理器进行优化的代码能够发挥最大性能，也维护了现有软件投资

Cortex-A9处理器最主要的流水线性能:

12 每周期支持两个算术流水线、加载-存储(load-store)或计算引擎以及分支跳转的并行执行

13 可将有相关性load-store指令提前传送至内存系统进行快速处理, 进一步减少了流水线暂停, 大幅提高了涉及存取复杂数据结构或C++函数的高级代码的执行效率

14 支持四个数据Cache Line的填充请求: 而且还能通过自动或用户控制预取操作, 保证了关键数据的可用性, 从而进一步减少了内存延时导致的暂停现象

15 支持无序指令完成回写: 允许释放流水线资源, 无需受限于系统提供所需数据的顺序

Amlogic 8726-M: Most Powerful MID Solution



Cortex **mali**

Cortex A9 CPU

1GHz CPU, 3-10X faster than ARM11/9

Android 2.2 / 3.0

**Most Advanced Android 2.2 Froyo
Android 3.0 Ready**

1080P HD Video

**1080P H.264, VC-1, WMV, MKV, RMVB
1080P HDMI TV Output**

Powerful
3D Engine

**Mali 400 @300Mhz
2 X Mali 200, 30 X Mali 55**



Superior Cortex-A9 and Mali 400 Performances

	Cortex-A9	ARM11	ARM9
DMIPS / MHz	2.5	1.2	1.04
Context Switch Time	-	2X	10-16X
Web Page Load Time	-	2.5X	5X

- More Performance
- Faster User Experience

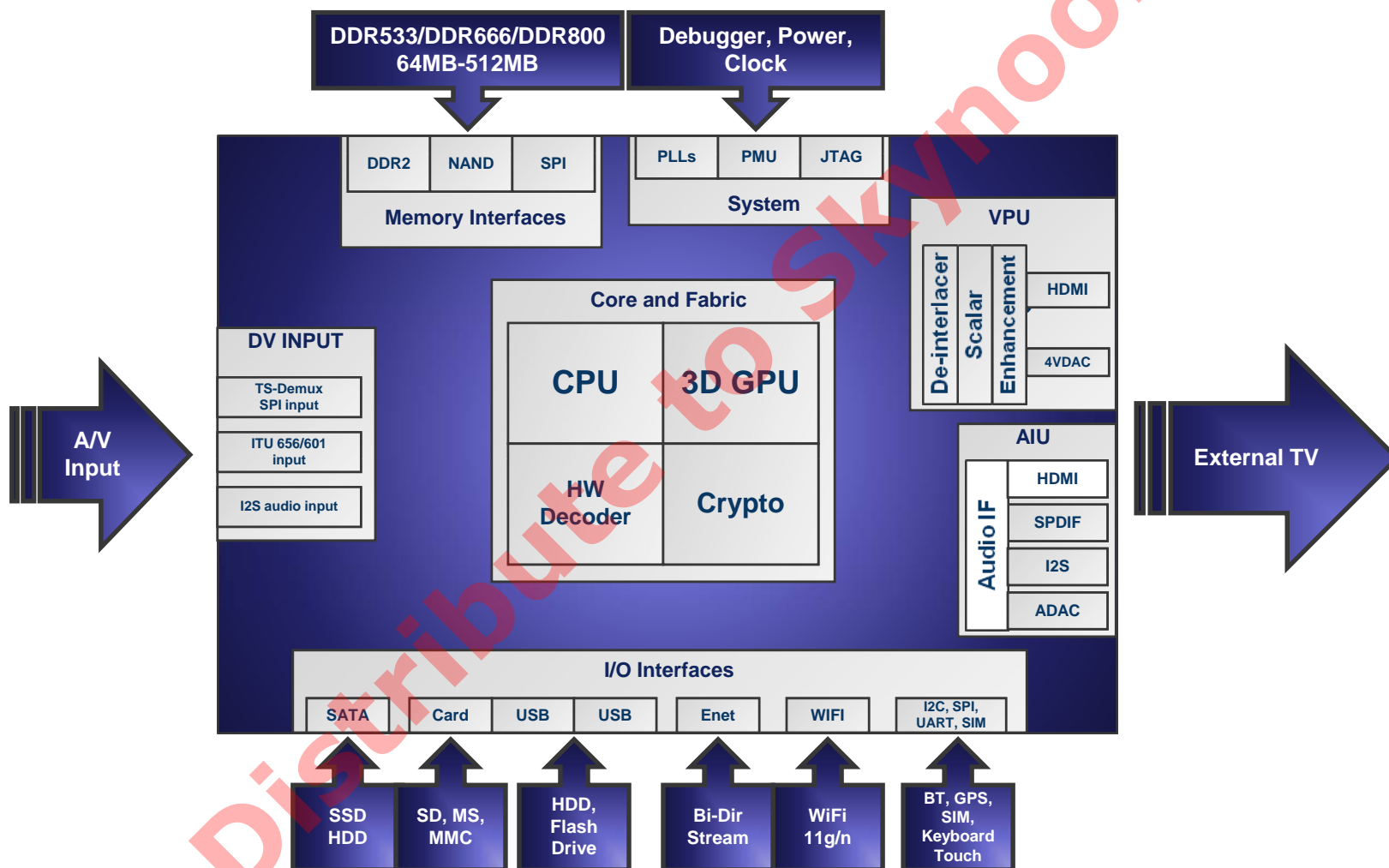
	Mali 400	Mali 200	Mali 55
Performance	275MHz	275Mhz	200Mhz
Fill Rate / Sec	275Mpix	275Mpix	100Mpix
Triangle / Sec	~30M	16M	1M
Power @ 30 FPS	43mW	77mW	~60mW

- More Powerful 3D
- Lower Power

• ARM Internal Benchmark Data



AML 8726-M Block Diagram



AML8726-M System Features

■ System Core

- ARM-Cortex A9
- Media CPU
- OpenGL ES 2.0 GPU
- Dual-core Video DSP
- Crypto engine supporting DES, 3DES, AES, DVB-CS
- Multi-channels DMA engine
- Timers and NMI
- Multiple Power Saving Modes

■ Memory Sub-System

- 8-bits SLC/MLC NAND interface
- 1-4 bits SPI NOR
- 16/32 bit DDR2 memory interface
- Up to 800MHz, 512MB DDR2 memory
- QoS-based internal fabric

■ Data I/O ports

- SATA port (PHY/controller)
- 2x USB ports (PHY/controller)
- Ethernet port (controller-only, RMII)
- 3 SD/SDIO port (PHY/controller)
- TS-input parallel SPI port

■ A/V ports

- HDMI 1.3 with HDCP and CEC
- 4x Video DAC (CVBS,S-Video,YPbPr,VGA)
- 2x Audio DAC
- SPDIF digital output
- I²S digital audio input and output
- ITU 601/656 digital video / camera input YUV 4:2:2, RGB 565,10bits bayer, 2bytes bayer

■ Misc

- SPI, i2c, UART, PWM interfaces
- Integrated RTC
- Spread spectrum clock
- SAR ADC for misc physical inputs
- BGA Package

AML8726-M Full HD Multimedia Support

■ Video Formats

- H.264 (1080p@30fps) HP/MP/BP L1.4 Support - *.mkv, *.ts, *.m2ts, *.mov, *.mp4
- VC-1 / WMV (1080p @30fps) Advanced Profile Level 3
- RMVB (720p@30fps) - *.rm, *.rmvb
- MPEG1/2/4 (1080p@30fps) - *.avi, *.mpg, *.mpeg, *.dat, *.vob, *.ts, *.mov, *.mp4
- MJPEG (D1@30fps) - *.avi, *.mov
- H.263 (D1@30fps) - *.avi, *.flv

■ Audio Formats:

- *.mp3 (MP3), *.wma (WMA), *.wav (PCM), *.ogg (Ogg Vorbis), *.m4a (AAC), *.mp4 (AAC), *.aac (AAC), *.ape (APE), *.FLAC (FLAC), *.ALAC (ALAC)

■ Advanced Video Enhancement Features

- ADV processing: Advanced De-blocking for digitally compressed Video source
- 3D noise reduction
- Pixel based motion adaptive edge enhanced de-interlacing
- Non-linear panorama scalar
- Color enhancement and color management

■ Subtitle: SMI, SRT, SUB, SSA, ASS, SUB+IDX, ISO Navigation

■ Playlist: PLS, M3U, WPL

■ File System: FAT32, exFAT, NTFS, HFS+, Samba, ext2/3

Chapter Two : Openliunx

(Philip)

Openlinux相关介绍

- 1 关于openlinux的几点说明
 - openlinux是我司code和文档对外发布的唯一窗口。
 - Openlinux上有多平台的信息，不仅仅是mid
 - 网址：<http://openlinux.amlogic.com> ->arm->android

Openlinux相关介绍2

- 2 openlinux获取code前的准备工作
 - 搭建基本编译环境（虚拟机或linux服务器）
 - 安装基本插件（git, jdk5.0, repo, flex, bison, gperf, libSDL-dev, libesd0-dev, libwxgtk2.6-dev (optional), build-essential, zip, curl.）
 - 注意：要装jdk5.0, jdk6不支持，因为不兼容
 - 生成key文件交由我司添加权限
 - 获取key的方法：
 - 运行\$ssh-keygen -t rsa, 生成id_rsa.pub,生成的目录为~/.ssh/id_rsa.pub,将此文件发送给我司相关sales, 由其申请添加权限。当我司审核授权后, 即可下载code。

Openlinux相关介绍3

■ 3 从openlinux上获取code

— 如果是从ubuntu进行开发，需要安装一些插件

• 运行如下命令：

• `$sudo apt-get install git-core gnupg flex bison gperf libsdl-dev libesd0-dev libwxgtk2.6-dev build-essential zip curl libncurses5-dev zlib1g-dev`

• 安装repo，在home目录下创建bin目录，然后获取repo并授予它执行权限：

• `$cd~`

• `mkdir bin`

• `curl http://android.git.kernel.org/repo >~/bin/repo`

• `chmod a+x ~/bin/repo`

• 把bin目录加入\$PATH：

• `$export PATH=$HOME/bin:$PATH`

Openlinux相关介绍3 (2)

- 在home目录创建mydroid目录用来存放android源码
 - `mkdir mydroid`
 - `cd mydroid`
- 下载我司MID的android源码:
 - `$repo init -u git@openlinux.amlogic.com:platform/manifest.git -b froyo-amlogic-mid`
 - `repo sync`

编译code

- Code编译主要分为两部分，一部分是linux kernel，一部分是android rootfs。因此需要分开编译，编译过程如下：
 - 编译rootfs:
 - `$source build /envsetup.sh`
 - `$lunch m1ref-eng`
 - `make`
 - 编译kernel:
 - 编译kernel因为涉及很多外设驱动，因此需要配置不同的config文件；
 - 目前，我司提供了3套开发板的参考设计，对应的分别为A板，B板，C板。其中a，b板为MID参考开发板，c板属于不带屏的产品，如媒体盒之类的。
 - 用户在kernel、arch/arm/configs下面可以看到各种config文件，其中以meson开头的为我司添加的config文件。目前统一了命名规则，如：refa00，其中a是指a版，00指公板，以此类推

编译code2

- 编译kernel前，需要将对应的config文件拷贝到kernel根目录，并替换原有的.config;
- 如果有些客户不是完全按照我们参考设计所支持的外设，二是自己更换了一些外设，那么就需要客户自己完善相关驱动，更改config文件对应部分。
- 编译kernel的命令如下：
 - make ulmage

启动调试开发板

- 目前启动调试开发板，有两种方式：通过tf卡启动和烧录到nand启动
 - 通过tf卡启动，需要将tf卡分为2个分区，并预留一部分放置uboot；一个分区为fat32分区，用来放置kernel的镜像uImage和一些apk等。另一个分区为ext3格式，用于放置rootfs；
 - 对应的uImage镜像的编译路径/kernel/arch/arm/boot/uImage
 - 对应的rootfs的编译路径为/out/target/product/mlref/root,
 - 注：需提前将/mlref目录下的system文件夹复制到root目录下对应文件夹中
 - 通过nand启动，可以直接烧录到nand对应地址中，但是我们不提倡用户这样做，因为nand分区的地址有可能我司会调整，所以提议用户采取我司提供的一键升级模式进行烧录

一键升级烧录升级

- 一键烧录升级，就是在上电的时候按着menu键，这样就会自动进入recovery模式进行自动烧录和升级。需要先将编译好的几个文件放到tf卡中，具体内容和操作如下：
 - checkout latest code
 - make android
 - cd android_root_dir
 - . build/envsetup.sh
 - lunch m1ref-eng
 - make
 - build ulmage
 - enable nand support in kernel
 - enable ramfs

一键升级烧录升级2

- enable General setup->Initial RAM filesystem and RAM disk
- set "Initramfs source file" to root dir(out/target/product/m1ref/root)
 - 注：此目录为相对目录，对应指定到rootfs对应的目录

- Make ulmage
- copy ulmage to out/target/product/m1ref

— build ulmage_recovery

- set General setup->"Initramfs source file" to recovery root dir(out/target/product/m1ref/recovery/root)
- make ulmage
- rename ulmage into ulmage_recovery
- copy ulmage_recovery to out/target/product/m1ref

一键升级烧录升级3

- build update.zip
 - cd android_root_dir
 - make otapackage
- flash image into nand
 - cp system.img userdata.img update.zip ulimage ulimage_recovery into sdcard

一键升级烧录升级3 (2)

■ 一键烧录和升级

- 保证机器是关机状态，插入tf卡，按住menu键，插入电源
- 系统上电后，可松开menu键
- 系统启动到recovery模式后，按音量键进行菜单选择，选中update.zip文件
- 按返回键进入升级
- 完成后，系统会回到recovery模式，选择 `reboot system now`，按返回键确定即可
- 注意：我们有时会更新uboot，为保证code的升级和uboot匹配，最好采用同一时间发布的uboot和code
- 另外我们如果tf卡中没有uboot，需要往nand里面烧录uboot，方法如下：
 - `Fatload mmc 0 82000000 u-boot.bin`
 - `nand rom_write 82000000 0 60000`

通过tf卡方式启动

- Tf卡的分区

- 我们需要分两个分区，同时预留一部分放置uboot，可以理解为分3个分区
- 我们需要进入linux系统，用fdisk工具进行操作
 - 先用fdisk工具降卡里原有的分区删除
 - 用fdisk工具对卡重新分区，注意第一个分区最好不要从第一个sector开始，预留一部分空间用来放置uboot

开机动画的修改（一）

开机动画文件是bootanimation.zip，默认存放在system/media/下，

bootanimation.zip文件是zip压缩文件，压缩方式要求是存储压缩，包含一个文件和两个目录：

1. 动画属性描述文件：desc.txt
2. 第一阶段动画图片目录：part0
3. 第二阶段动画图片目录：part1

desc.txt文件内容：

480 427 30

p 1 0 part0

p 0 10 part1

开机动画的修改（二）

desc. txt文件分析:

480 427 30

宽 高 帧数

p 1 0 part0

标志符 循环次数 阶段切换间隔时间 对应目录名

p 0 10 part1

标志符 循环次数 阶段切换间隔时间 对应目录名

=====

开机动画的修改（三）

标志符：必须是：p

循环次数： 0 ：表示本阶段无限循环

阶段切换间隔时间：

单位是一个帧的持续时间，比如帧数是30，那么帧的持续时间就是 $1\text{秒}/30 = 33.3$ 毫秒。

阶段切换间隔时间期间开机动画进程进入休眠，把CPU时间让给初始化系统使用。
也就是间隔长启动会快，但会影响动画效果。

part0和part1目录内包含的是两个动画的系列图片，图片为PNG格式。

系列图片文件的加载刷新按文件名的名称排序。

开机动画的修改（四）

制作bootanimation.zip：首先从里面的图片说起 图片一定要转换成PNG格式，建议图片要和mid对应的分辨率一样，如果不是也可以的 有可能变形 注意图片的大小要统一（如果不会做连接的图片组 直接找个适合自己分辨率的动态图片分解出PNG格式的静态图片，用ImageReady可以批量导出）

建立part1 part2或android文件夹 放进一组连接的图片组开始压缩成ZIP格式 名字一定要bootanimation.zip 压缩方式一定是存储 开始写desc.txt

desc.txt 格式(解释看上面)

有2个文件夹 part1、part2的话，比如分辨率240*320，那么如下：

240 320 15

p 1 1 part1

p 0 0 part2

保存后拖入bootanimation.zip里面OK，压缩方式要求是存储压缩。

Chapter Three : Code Release

(Kelvin Wei)

Release code notice

- 1 发布代码基本步骤
- 2 简单的错误排查
- 3 版本的控制和代码返回

Release code notice

1 同步最新代码

```
rootfs#repo sync
```

```
error: packages/amlogic/TSCalibration/:  
contains uncommitted changes
```

```
rootfs#cd packages/amlogic/TSCalibration
```

```
TSCalibration#git pull
```

```
m1ref#cd -
```

Release code notice

1. 同步最新代码

rootfs#repo sync

error: Cannot remove project
"device/amlogic/m1ref": uncommitted
changes are present
commit changes, then run sync again
rootfs#cd device/amlogic/m1ref
m1ref#rm 红色字部分
m1ref#cd -

@mlogic

Release code notice

1. 同步最新代码

```
rootfs#cd packages/amlogic/FileBrowser/
```

```
FileBrowser#git pull
```

```
FileBrowser#cd -
```

```
rootfs#repo sync
```

Release code notice

1 同步最新代码

```
kernel# git pull
```

异常

```
kernel# git status
```

Release code notice

2 执行编译初始化脚本

```
rootfs#. build/envsetup.sh
```

```
rootfs#lunch m1ref-eng
```

```
rootfs#lunch m1ref-user
```

Release code notice

3. 全部清除上次编译结果

rootfs#make clean

异常情况出现

kernel#make clean

Release code notice

4. 注意修改版本号

定义于\build\core\version_defaults.mk 的
最后一个变量

```
ifeq "" "$(CUSTOM_VERSION)"  
    # This is the custom version  
    CUSTOM_VERSION := V0.01  
endif
```

每次发code前请手动改成你要发的版本号

Release code notice

5. 注意修改机器型号

关于型号，我们已经有现成的接口，请修改
device\amlogic\mlref.mk里面的变量

```
PRODUCT_MODEL := Amlogic M1 reference  
board
```

修改为你的机器型号

Release code notice

6 开始编译rootfs

rootfs#make

异常情况出现

rootfs#cd error

error#git status

Release code notice

7 Kernel的配置和编译

kernel#make menuconfig
enable ramfs,

enable General setup->Initial RAM
filesystem and RAM disk

set "Initramfs source file" to root
dir(out/target/product/m1ref/root)

kernel#make ulmage

copy ulmage to out/target/product/m1ref

Release code notice

8 Kernel_recovery的配置和编译

```
kernel#vi .config
```

```
modify (out/target/product/m1ref/root) to  
(out/target/product/m1ref/recovery/root)
```

```
Kernel#make ulmage
```

```
Kernel#cd arch\arm\boot
```

```
Boot#mv ulmage ulmage_recovery
```

```
Boot#cp ulmage_recovery
```

```
android_root_dir/out/target/product/m1ref
```

Release code notice

9 打包rootfs

rootfs#make otapackage

忘记修改版本号和型号的最后补救

Rootfs#cd

out\target\product\m1ref\system

修改build.prop中的

ro.build.display.id

ro.product.model

Release code notice

10 发布前的测试

cp update.zip ulmage ulmage_recovery
into sdcard for update

应该重点测试最新更新的部分以及测试其他部分没有大的BUG.

Release code notice

11 为工程师做rootfs. tgz

```
Rootfs#cd out/target/product/m1ref
```

```
M1ref#mkdir android.rootfs
```

```
M1ref#cp -r root/* android.rootfs/
```

```
M1ref#cp -r system/* android.rootfs/system/
```

Release code notice

11 为工程师做rootfs. tgz

```
M1ref#cd android.rootfs  
android.rootfs #vi init.rc
```

```
#mount yaffs2 mtd@system /system  
#mount yaffs2 mtd@userdata /data nosuid  
nodev
```

```
tar cvzf ../android.rootfs.tgz .
```


Release code notice

12 rootfs.tgz的使用

put to SD card:

```
sudo tar zxvf android.rootfs.tgz --numeric-owner  
-C /mnt
```

Release code notice

13 rootfs的TAG添加

1)本地添加tag

```
# repo --trace forall -c 'git tag amlogic-  
android.mid.01.05.2011'
```

2)增加一个远程服务器svn-sc 以获取写权限

```
# repo --trace forall -c 'git remote add svn-sc  
gituser@192.168.1.1:$REPO_PROJECT'
```

Release code notice

13 rootfs的TAG添加

3) 将tag更新到远程服务器上去

```
# repo --trace forall -c 'git push --tag  
svn-sc amlogic-  
android.mid.01.05.2011:refs/tags/amlogic-  
android.mid.01.05.2011 '
```

远程删除服务器上tag的方法

```
#repo forall -c 'git push svn-sc :tagname'
```

Release code notice

14 kernel的TAG添加

增加本地tag: `#git tag tagName`

删除 tag: `#git tag -d tagName`

删除远程 tag: `#git push
origin :refs/tags/tagName`

Release code notice

15. 代码返回到某一时间点

Rootfs:

```
rootfs#repo forall -c git checkout tagName
```

Kernel:

```
kernel#git checkout tagName
```

Release code notice

16 Kernel状态查看

查看已有tag以及简述

```
kernel#git log
```

```
commit
```

```
f2d9dacf0578f729be460b22eaf5672bd4022dd6
```

```
Merge: 1a5cb2c a81919d
```

```
Author: tiejun.peng
```

```
<tiejun.peng@amlogic.com>
```

```
Date: Fri Jan 7 17:33:58 2011 +0800
```

Release code notice

16 Kernel状态查看

Merge branch 'froyo-amlogic' of
git://xxx.xxx.xxx.xxx/m1-kernel-android into
froyo-amlogic

kernel#git show
f2d9dacf0578f729be460b22eaf5672bd4022dd6
可以看到这个TAG大致修改情况

Chapter4: U-boot应用与开发

(Elvis)

1. 什么是u-boot?

U-Boot, 全称 Universal Boot Loader, 是遵循 GPL 条款的开放源码项目。其源码目录、编译形式与 Linux 内核很相似, 事实上, 不少 U-Boot 源码就是相应的 Linux 内核源程序的简化, 尤其是一些设备的驱动程序, 这从 U-Boot 源码的注释中能体现这一点。

http://openlinux.amlogic.com/wiki/index.php/Arm/Boot_Loader

<ftp://ftp.denx.de/pub/u-boot/>

2. u-boot用来做什么？

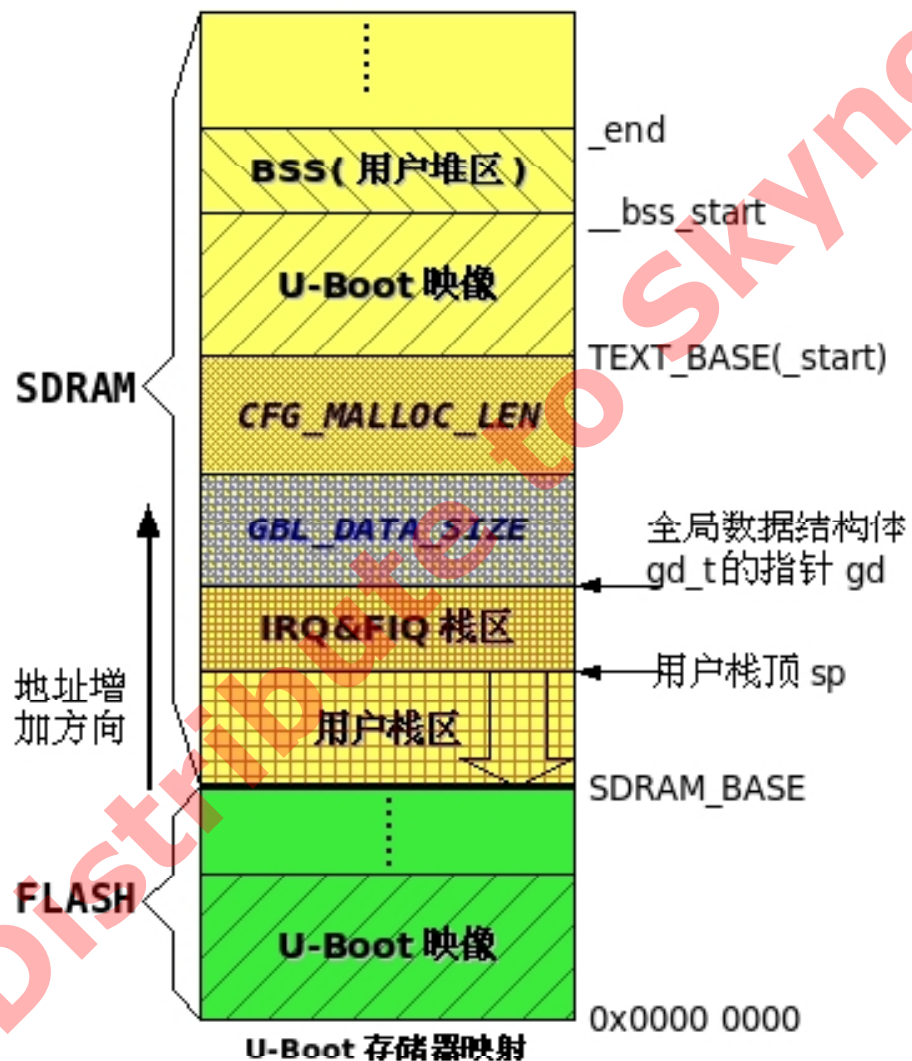
1. 引导OS。
2. 硬件调试。

Distribute to Skynoon!

3. u-boot怎么用？

- U-boot基本结构——A,B,C
- U-boot基本使用——C,D,E,F
- U-boot定制化设计——G,H,I

A. u-boot结构



B. u-boot 目录结构

Board: 目标板相关文件，主要包含SDRAM、FLASH驱动

Common: 独立于处理器体系结构的通用代码，多是对下一层驱动程序的进一步封装，还有一些基本命令。

Arch: 与处理器相关的文件。

Driver: 通用设备驱动。

Doc: u-boot的说明文档。

Examples: 一些测试程序，可以使用u-boot下载后运行。

B. u-boot 目录结构 (2)

Include: 头文件和目标板配置文件。尤其configs子目录下与目标板相关的配置头文件是移植过程中经常要修改的文件。

Lib: 通用函数库。

Net: 与网络功能相关的文件目录。

Disk: 通用的硬盘接口程序。

Fs: 文件系统。

Post: 上电自检程序。

Tools: 通用工具, 比如mkimage。

C. u-boot的编译

Make help

make <board_name>_config

make

编译完成后会在build/目录下生成：u-boot-aml.bin

D. u-boot基本命令

1 Nand命令: nand scrub, nand rom_write

2 Spi命令: sf probe 2

3 下载命令: loadb, tftp, tftpboot, nfs

4 内存操作命令: md, mm, mw, cp, cmp

5 FS操作命令: fatload, extload

6 help 或 ?

7 Env命令: printenv, setenv, saveenv, defenv

8 存储设备操作: mmcinfo, usb start

E. 修改配置文件

以m1_8726m_mid.h举例：

- 1.基本配置
- 2.命令支持

Distribute to Skynoon!

F. 如何增加和修改命令

```
include/Command.h
```

```
#define
```

```
U_BOOT_CMD(name, maxargs, rep, cmd, usage, help) \
```

```
cmd_tbl_t __u_boot_cmd_##name Struct_Section =  
    {#name, maxargs, rep, cmd, usage, help}
```

G. 定制化设计

以mid_aml8726m.c举例。

start_armboot

main_loop

switch_boot_mode

upgrade

H. 几个重要文件

Makefile

Readme.mk

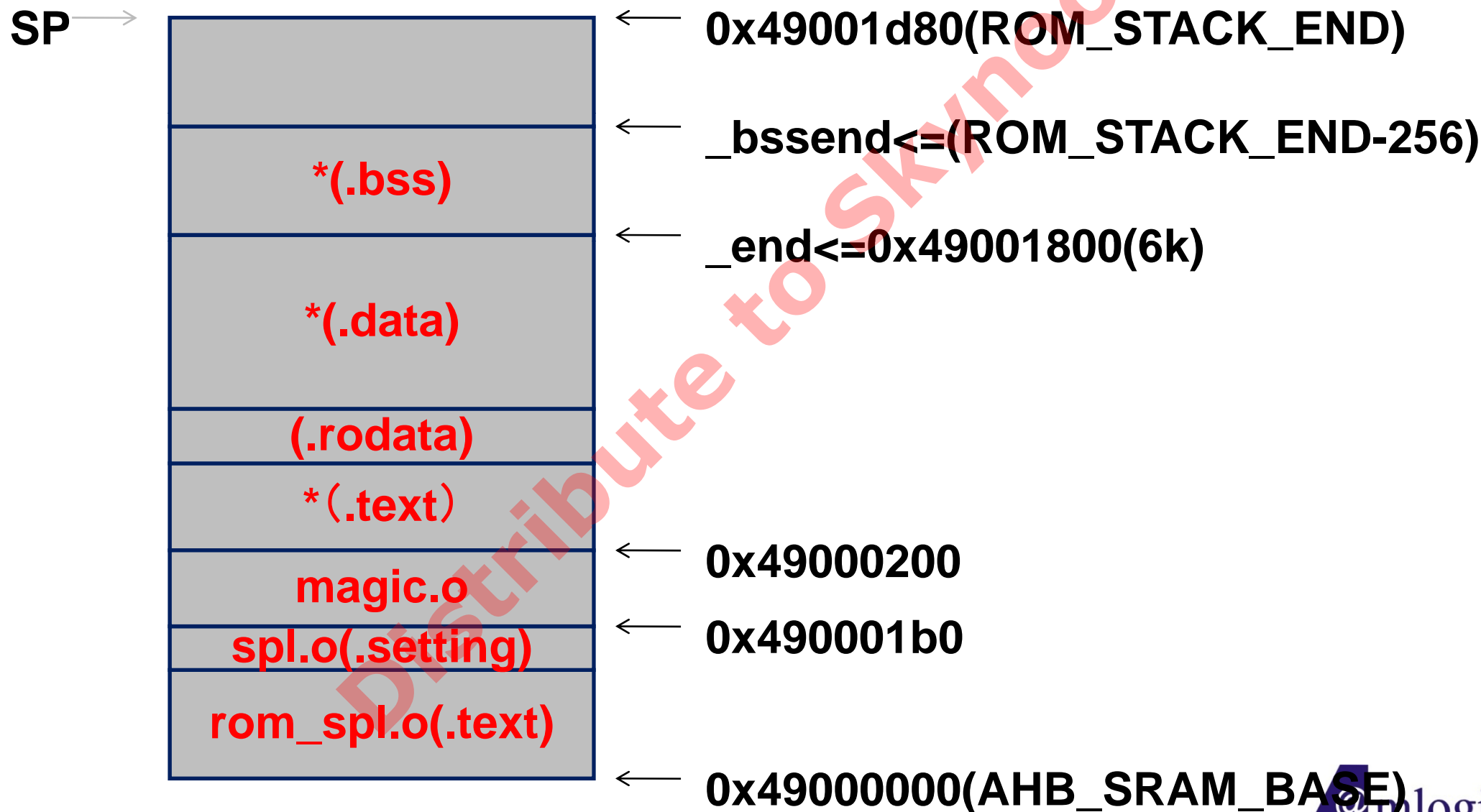
mkconfig

config.mk

u-boot.lds

Distribute to Skynoon!

I. Rom_spl结构



Chapter Six :软件相关硬件培训 (Zhong Feng)

Distribute to SSVN only!

基本系统

- 1 2pcs 128Mx16 DDR2, 容量512M bytes。Memory的极限容量为512M Byte (32bit)。可以使用一组16bit DDR2工作, memory带宽下降, 系统性能也会有所下降。
- 2 1pcs TSSOP MLC nand, 容量1G bytes。注意, nand CE2和SPI_CS_A复用, 如果要使用2CE或4CE的nand flash, 就不能搭配使用SPI norflash, 需要直接从nand boot。确实需要保留SPI boot的话, 可以添加SPI_CS/nand_CE2的切换电路, reset后默认连接到SPI_CS, 在SPI boot结束后, 软件控制切换连接到Nand_CE2。

操作调试接口

- 1 Console UART接口 : UART_TX输出电平为5V/0V, 个别电脑连接可能会失败, 请尝试使用USB->串口转换dongle
- 2 JTAG接口: 请配合使用ARM cortex a9 JTAG debug工具(debug ARM CPU), 或者Amlogic JTAG dongle (Debug ARC CPU).

电源部分工作原理

- 涉及软件部分：
 - 1. 开机步骤：power key硬开机，输出维持hold；开机下检测到RTC_GPO或PWR_KEY_DET变低时关机。
 - RTC_GPO是内建RTC的中断输出，定时到达后会输出低电平，可做定时开关机，不过第一次是随机的状态（工厂作业不方便）。这个pin也可以用作输入。
 - 2. 开机后启动设置：boot里面要先侦测ADAPTOR有没有插入，有插入直接开机，若没有则先check 电池电量，电量偏低时设置成终止启动。
 - 3. Power saving：最小系统工作，关闭所有受控外围电路。
- 此时1.2V降低电压可节省功耗，1.2V调压原理：通过LED_CS0恒流源输出调节，共15阶输出，没阶14ua。

电源部分工作原理(2)

- 4. AAT3620电池充电部分控制，读取电池充电状态，控制充电电流大小。
- 5. 背光调光：PWM调光（LED_BL_PWM）和恒流源调光（VGHL_CS0）---13UA/15阶结合。
- 反馈信号连接到LCD_CS0，可以通过LCD_CS0的15阶电流源（电流以13uA为步进递增）控制模拟调光。需要根据背光IC的反馈电压选择合适的亮度调节串阻。EN信号连接到PWM pin，可以通过PWM信号实现数字调光，增加调光的级数。
- 6. 电池电压侦测。10 bit/3.75V（1.75V对应ADC是543）
- 7. 上电顺序：原则上逐次打开，各部分供电不要一下子打开。如可先打开LCD3.3，延迟一段时间后（如设成500ms）再打开背光电源（背光渐进打开，刚打开时PWM占空比设到最低），延迟1s后再打开wifi（如果需要），500ms后再打开VCCX2（如果需要）

扩展IO IC应用:

- SN7320 地址可设置, 注意看配置, 初始化前先做复位。中断不用。

flash配置/power on config

- Poweron config说明:
- 1. romcode首先尝试从选定的INTL设备(SPI/nand/small-nand 3选1) boot, 从选定的INTL设备(3选1)boot失败后, 自动尝试从card启动 (A/B/C)。
- 2. 使用小容量nand flash时, 如果page size是512 byte, NAND_RE NAND_WE boot配置要修改为00
- 3. romcode Card boot支持SD/SDHC/MMC(romboot不支持MMC4.3 / eMMC / iNand / movinand, SPI+card无此限制)
- 4. romcode Card boot支持SD/SDHC/MMC (romboot不支持MMC4.3 / eMMC / iNand / movinand, SPI+card无此限制)
- 5. 作为调试的技巧, 可以将INTL设置为不存在的设备, 来达到skip INTL设备中的bootloader, 直接从card启动的目的。通产可以把INTL设备选定为reserve选项, INTL boot失败, romcode直接转到card启动。
- 6. 选择非ROMBOOT的SPI BOOT时, 系统只从SPI_A启动, 不会跳转到card boot。

卡/LCD RGB调换/I2S/GP_INPUT[1..3]相关

- card接口支持：SD/MMC/MS 3IN1。
- 使用6bit LCD接口时，最好高位对齐连接，空出的低位数据线可以用作GPIO。
- GP_INPUT[1..3]只能做输入，不能做输出
- GP_INPUT不是中断口，有些有特殊设备的中断信号是脉冲波，注意不能用到这里。
- VGHL_CS0, LED_CS0, LED_CS1用作输入信号时，注意这里是一个比较器，参考电压0.48V，请保证输入低电平<0.2V，输入高电平>1V

内部时钟应用

- ITU601 MCLK和RMII CLK内部时钟冲突，电路上可以同时支持，但是不能同时使用。如果需要同时使用RMII和ITU601(例如网络视频通话)，ITU601要使用外部晶振, 或使用LCD_VGHL_PWM输出24MHz。
- LED_PWM or VGHL PWM 可从demod_400MHz_clock分频产生50MHZ用于以太网。利用CPU自己产生的50M频率时，注意其时序较RX信号偏早，故其他接到CPU ethernet模块时需要利用阻容电路延迟。
- LCD_STH可产生32.768K.
- VGHL_PWM(PWMB)/LED_BL_PWM(PWMA)可做普通PWM应用。

其他:

- M1增加控制关闭内部上下拉电流的能力。软件应注意关闭内部上下拉电阻，以降低工作和待机功耗。
- AVDD33_HDMI没供电，panel CLK没输出。
- USB: 1 Mini OTG + 1 HOST。
- SDIO:A/B
- UART:A/B, 各有三个bongding口，应用是注意查看对应pin脚。
- I2C:硬件有2组A和B，常用B组。
- ADC按键：通常一个ADC最多设计8个按键。需要较少的按键时，可以合并阻值，请务必保持按键电压为标准值，避免修改driver。
- 摄像头：CLK注意有两个，一个摄像头系统CLK，一个pixel信号同步CLK。
- 模拟输出：4个通道，可输出YPbPr信号，也可以在Y信号上配置成输出CVBS信号。理论上可互相交换设置，但最好统一，便于管理。
- 注意WIFI/DEMODULE/ADC模块3.3V供电CPU内部是相连的。

常见问题及解决办法:

1. 插上DC电源，发现没有供电。
 - (1) 可能是前端保护电路起作用，即可能电阻、稳压管参数不对，造成电路被关断
 - (2) A版本R52，B版本R101，没有焊接，即接DC不直接供电，可以通过按Power键是否有供电来判断。
2. 电压供电正常、复位正常，但在插入载有u-boot的TF卡时串口没有打印信息。
 - (1) 确认u-boot是否正确（256M还是512M的），且有没有正确通过WINHEX做到TF卡里，可以找其他板子尝试。
 - (2) u-boot频率是否太高，板子性能不好，u-boot降频进行尝试
 - (3) TF卡供电是否正常，CLK、CMD、D0是否有信号。（可以通过稳压电源观察，功耗是否比没有插入TF卡时增加约0.5W.

常见问题及解决办法：（2）

3. 启动kernel过程中，初始化的时候重启。

(1) 用示波器测量VCCx的电压，是否在被拉低。很有可能由于VCCx2打开，AVDD、VCC5V升压导致VCCx电压跌落。

(2) VCCx前端电容值加大，VCCx2端电容值减少，且优化AVDD、VCC5V的上电时序。

4. 原本好的机器，跑不起来，串口也没有打印信息。

(1) 检查串口连接是否正确。

(2) 先复位机器，并用稳压电源供电，观察并比较其电流。若电流较小（如公板没有u-boot的时候电流约100mA@5V），则因为没有u-boot；若电流稍大（如公板中少了u-boot后，电流约200mA@5V），则是u-boot坏掉，需要重新烧u-boot。

常见问题及解决办法：（3）

5. u-boot烧在NAND Flash里面，但需要从TF卡启动u-boot。

(1) 通过4.7K电阻把NAND_WE(或NAND_RE)下拉到地，并用启动卡启动便可。

6. 机器只有在连接串口板时，才能正常启动。

(1) 由于串口的RX软件没有设置上拉，通过上拉4.7K或10K的电阻可以解决。(最终需要软件设置内部上拉来实现这个功能)

7. 系统能够正常跑起来，但LCD的CLK没有输出。

(1) 由于AVDD33_HDMI(pin_J4\K4\L4)没有供电引起。

8. LCD屏能正常显示，但在显示黑色图片或UI时会出现白色细条纹。

(1) 检查LCD的3.3V的电压是否稳定(很有可能P-MOS没有导通—即没有控制)

常见问题及解决办法：（4）

9. u-boot可以跑起来，也可以通过按menu键进入升级的界面，但选择升级rootfs时，显示找不到TF卡，同时查看打印信息时，确实没有显示初始化TF卡。

(1) 确认原理图是参考REF-A还是REF-B，同时确认code是用REF-A还是REF-B，因为两者CARD_DET/EN的管脚不一致，导致检测不到卡的插入。也可以测量CARD_DET/EN管脚或者TF卡的电源，看起是否正常。

10. 在进入机器人图标时死掉，并一直重复打印：

```
ReadSpare2112_1 blk 800 page 0 ret -2
```

```
GET blk 800 page sttua in formart
```

```
nand dio read ret -2
```

常见问题及解决办法：(5)

且重新对NAND进行擦写、升级也会出现同样的现象。

(1) 由于原来有code的两片NAND调换了位置，及原来A片NAND存放code，但在code没有擦除的情况下焊接到了B片nand的位置。可以通过把NAND调换回去。

(2) 通过命令擦掉两个NAND，然后重新升级。

11. 启动kernel到一半，然后有重新打印u-boot的信息，一直反复。

(1) 确认复位芯片是否焊接好。

(2) 用示波器测量VCCx的电源是否有波动，确定是否因为点屏造成VCCx电压的下降，而导致硬件电路的复位。

12. WIFI没法连接。

(1) 查看打印信息，确认WIFI初始化是否通过。

(2) 确认模块的PIN36管脚是否有32.768KHz的时钟输入。

常见问题及解决办法：(6)

- (3) 确认模块的PIN31 (WLBT_REGON) 管脚在工作时是否为高。
- (4) 确认WLBT_REGON是否先拉高, 然后通过PIN7(WL_RST_N)对模块进行复位。
- (5) 确认SDIO的D0-D3、CMD是否有上拉。
- (6) 确认天线是否焊接好。

13. Audio 插入耳机没有声音或不能正常侦测。

- (1) 注意查看耳机插入的侦测逻辑, WA 7D/10D目前的原理图都是插入低, 不插入高; 7D/10D现在的图有个缺陷, 插入耳麦一体的插头时不能正常侦测, 下版改善

14. flash由一颗更换成两颗后不能工作。

- (1) 修改flash的配置。

Chapter7: 触摸屏

(Sunny)

触摸屏简介

目前主要有以下几类触摸屏，：

电阻式

电容式

红外式

表面声波式

*以电阻式和电容式最为常用

电阻式触摸屏原理

- 四线制:

两层导电层都是ITO,互为电压检测层

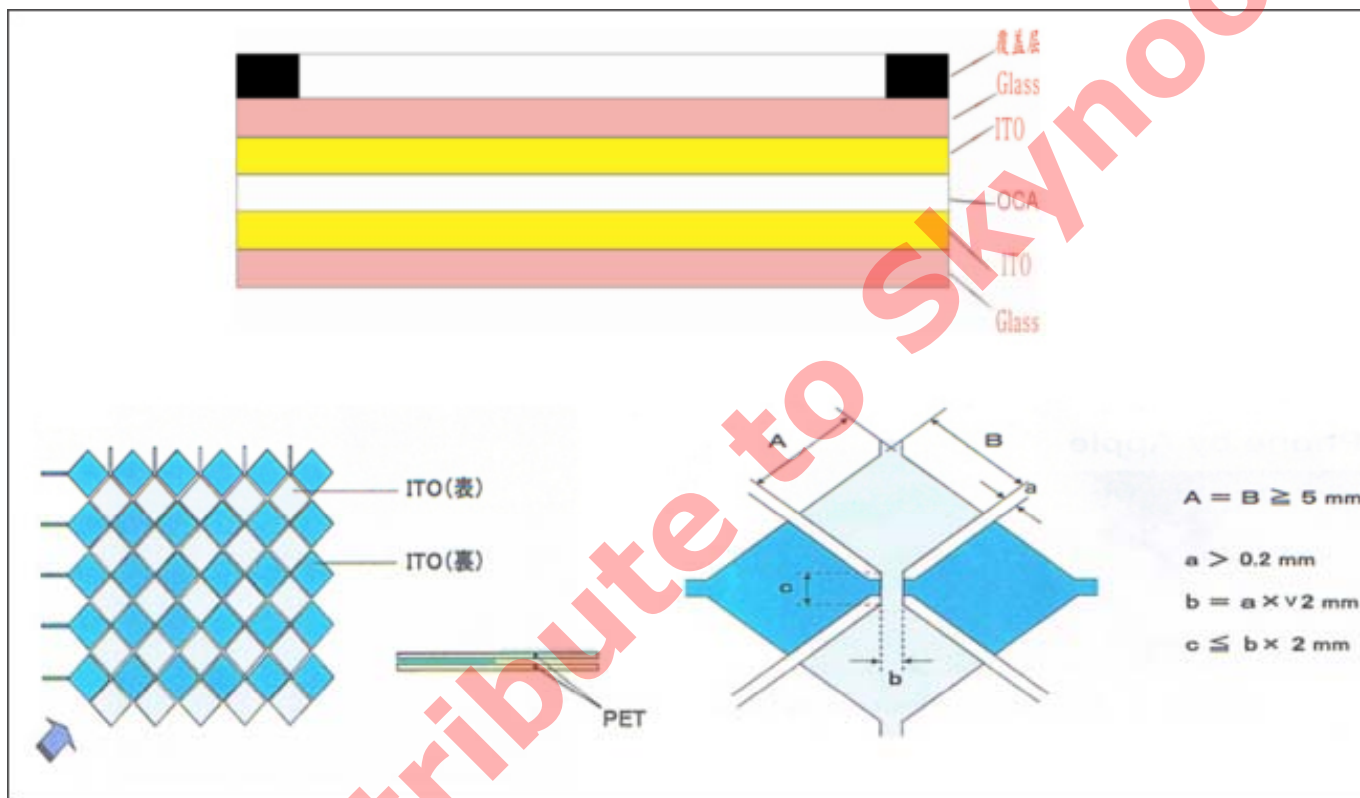
- 五线制:

外层导电层使用的是延展性好的镍金涂层,内层用ITO

- 优点:

1. 屏及控制系统便宜;
2. 精确度高;

电容式触摸屏原理（1）



绿色和蓝色的ITO模块位于两层ITO涂层上，可以把它们看作是X和Y方向的连续变化的滑条，对X和Y方向上不同的ITO模块分别扫描以获得触摸点的位置

电容式触摸屏原理（2）

优点

1. 触摸效果好;
- 2. 支持多点;
- 3. 透光性好;
- 4. 耐磨损, 寿命长;
- 5. 只需要一次或者完全不需要校正;

常用驱动IC

- 电阻式

I2C接口: TSC2003、TSC2007、ZT2083

SPI接口: ADS7846、TSC2046、XPT2046

内部ADC:

电容式

I2C接口: sis92xx、ilitek、raydium、himax, goodix

Tsc2007配置及调试（1）

```
static int tsc2007_init_platform_hw(void)
{
    /* set input mode */
    gpio_direction_input(GPIO_TSC2007_PENIRQ);
    /* set gpio interrupt #0 source=GIOD_24, and triggered by falling edge(=1) */
    gpio_enable_edge_int(GPIO_TSC2007_PENIRQ_IDX, 1, 0);
    return 0;
}

static int tsc2007_get_pendown_state(void)
{
    return !gpio_get_value(GPIO_TSC2007_PENIRQ);
}
```

Tsc2007配置及调试（2）

- static int tsc2007_get_pendown_state(void)
- {
- return !gpio_get_value(GPIO_TSC2007_PENIRQ);
- }

- #define XLCD 800
- #define YLCD 600
- #define SWAP_XY 0
- #define XPOL 0
- #define YPOL 1
- #define XMIN 130
- #define XMAX 3970
- #define YMIN 250
- #define YMAX 4000

Tsc2007配置及调试（3）

```
int tsc2007_convert(int x, int y)
{
    #if (SWAP_XY == 1)
        swap(x, y);
    #endif
    if (x < XMIN) x = XMIN;
    if (x > XMAX) x = XMAX;
    if (y < YMIN) y = YMIN;
    if (y > YMAX) y = YMAX;
    #if (XPOL == 1)
        x = XMAX + XMIN - x;
    #endif
    #if (YPOL == 1)
        y = YMAX + YMIN - y;
    #endif
    x = (x - XMIN) * XLCD / (XMAX - XMIN);
    y = (y - YMIN) * YLCD / (YMAX - YMIN);
    y = y - 32 * YLCD / (y / 2 + YLCD);
    return (x << 16) | y;
}
```

Tsc2007配置及调试（4）

```
static struct tsc2007_platform_data tsc2007_pdata = {  
    .model = 2007,  
    .x_plate_ohms = 400,  
    .get_pendown_state = tsc2007_get_pendown_state,  
    .clear_penirq = NULL,  
    .init_platform_hw = tsc2007_init_platform_hw,  
    .exit_platform_hw = NULL,  
    .poll_delay = 40,  
    .poll_period = 10,  
    .abs_xmin = 0,  
    .abs_xmax = XLCD,  
    .abs_ymin = 0,  
    .abs_ymax = YLCD,  
    .convert = tsc2007_convert,  
};  
#endif
```

RAYDIUM配置及调试（1）

- `#ifdef CONFIG_RAYDIUM_CAPACITIVE_TOUCHSCREEN`
- `#include <linux/capts.h>`
- `#define TS_IRQ_GPIO ((GPIOD_bank_bit2_24(24)<<16) | GPIOD_bit_bit2_24(24))`
- `#define TS_IRQ_IDX (GPIOD_IDX + 24)`
- `#define TS_RESET_GPIO ((GPIOD_bank_bit2_24(23)<<16) | GPIOD_bit_bit2_24(23))`
- `static int ts_init_irq(void);`
- `static int ts_get_irq_level(void);`
- `static struct ts_platform_data ts_pdata = {`
- `.mode = TS_MODE_INT_LOW,`
- `.irq = INT_GPIO_0,`
- `.init_irq = ts_init_irq,`
- `.get_irq_level = ts_get_irq_level,`
- `.info = {`
- `}`

RAYDIUM配置及调试（2）

- .xmin = 0,
- .xmax = 800,
- .ymin = 0,
- .ymax = 600,
- .zmin = 0,
- .zmax = 1,
- .wmin = 0,
- .wmax = 1,
- .swap_xy = 0,
- .x_pol = 0,
- .y_pol = 0
- },
- .data = (void *) (TS_RESET_GPIO+1),
- };

RAYDIUM配置及调试（3）

```
static int ts_init_irq(void)
{
    int group = ts_pdata.irq - INT_GPIO_0;
    int mode = ts_pdata.mode;

    if (mode < TS_MODE_TIMER_READ) {
        gpio_direction_input(TS_IRQ_GPIO);
        if (mode == TS_MODE_INT_FALLING) {
            gpio_enable_edge_int(TS_IRQ_IDX, 1, group);
        }
        else if (mode == TS_MODE_INT_RISING) {
            gpio_enable_edge_int(TS_IRQ_IDX, 0, group);
        }
    }
}
```

RAYDIUM配置及调试（4）

```
    else if (mode == TS_MODE_INT_LOW) {  
        gpio_enable_level_int(TS_IRQ_IDX, 1, group);  
    }  
    else if (mode == TS_MODE_INT_HIGH) {  
        gpio_enable_level_int(TS_IRQ_IDX, 0, group);  
    }  
    }  
    return 0;  
}  
  
static int ts_get_irq_level(void)  
{  
    return gpio_get_value(TS_IRQ_GPIO);  
}  
#endif
```

Touch调试

- 1. 电路检查;

包括电源, 电压, 排线连接, 上拉电阻等

- 2. 检查软件配置;

包括I2C器件地址, IRQ/RESET IO配置, 中断是否打开

- 3. 示波器查看IRQ波形;

- 4. 示波器查看I2C波形;

- 5. 数据分析;

- 6. 联系原厂;

Chapter8:

Amlogic mid 基本指南(一)

(Spike)

1. kernel代码结构

- Bsp位于arch\arm\mach-meson
- Config文件位于arch\arm\configs
- Amlogic相关的driver位于drivers\amlogic
- Audio dac driver位于sound\soc\aml
- Sensor driver位于drivers\i2c\chips

2. Kernel config

- 从我们openlinux上获取代码之后，首先找一个和你们板子原理图相近的ref config
- Kernel目录执行make help可以看到我们所有的config

3. GPIO控制

- `int set_gpio_mode(gpio_bank_t bank,int bit,gpio_mode_t mode)`
- `gpio_mode_t get_gpio_mode(gpio_bank_t bank,int bit)`
- `int set_gpio_val(gpio_bank_t bank,int bit,unsigned long val)`
- `unsigned long get_gpio_val(gpio_bank_t bank,int bit)`
- `int clear_mio_mux(unsigned mux_index, unsigned mux_mask)`
- `int set_mio_mux(unsigned mux_index, unsigned mux_mask)`
- 例子:
`set_gpio_val(GPIOA_bank_bit(8), GPIOA_bit_bit0_14(8), 1);`
`set_gpio_mode(GPIOA_bank_bit(8), GPIOA_bit_bit0_14(8), GPIO_OUTPUT_MODE);`

4. Battery driver

- 默认的battery driver位于drivers\amlogic\power目录
- Device Drivers->Amlogic Device Drivers->Amlogic Power Management Support ->Amlogic Power support
- 硬件上通过8726m的adc读取电压来判断battery level
- Bsp里面的结构体定义了一些battery相关的函数和变量
- ```
static struct aml_power_pdata power_pdata = {
 .is_ac_online = is_ac_connected,
 .set_charge = set_charge,
 .get_bat_vol = get_bat_vol,
 .get_charge_status = get_charge_status,
 .set_bat_off = set_bat_off,
 .bat_value_table = bat_value_table,
 .bat_charge_value_table = bat_charge_value_table,
 .bat_level_table = bat_level_table,
 .bat_table_len = 37,
};
```

## 5. Adc keypad

- Adc keypad, driver目录位于  
drivers\amlogic\input\adc\_keypad
- Device Drivers->Amlogic Device Drivers->Amlogic Input  
device support->Amlogic ADC keypad support
- 按键和adc值的映射表格位于bsp里面的数组
- ```
static struct adc_key adc_kp_key[] = {  
    {KEY_PAGEDOWN, "vol-",  CHAN_4,      0, 60},  
    {KEY_PAGEUP,   "vol+",  CHAN_4,     306, 60},  
    {KEY_TAB,       "exit",   CHAN_4,     602, 60},  
    {KEY_LEFTMETA, "menu",  CHAN_4,     760, 60},  
    {KEY_HOME,      "home",  CHAN_4,     854, 60},  
};
```

6. GPIO keypad

- Driver目录位于amlogic\input\key_input
- Device Drivers->Amlogic Device Drivers->Amlogic Input device support
->Amlogic Key Input(custom) support
- 按键相关的参数和函数位于bsp
- `int _key_code_list[] = {KEY_POWER};`
- `static int key_input_init_func(void){`
- `}`
- `static int key_scan(int *key_state_list){`
- `key_state_list[0] = (READ_CBUS_REG(ASSIST_HW_REV)&(1<<10))? 0:1;`
- `}`

6.GPIO keypad (2)

- static struct key_input_platform_data key_input_pdata = {
- .scan_period = 20,
- .fuzz_time = 60,
- .key_code_list = &_key_code_list[0],
- .key_num = ARRAY_SIZE(_key_code_list),
- .scan_func = key_scan,
- .init_func = key_input_init_func,
- .config = 0, // 0 int mode, 1 polling mode
- };

7. brightness control

- Driver位于drivers\amlogic\display\backlight
- Device Drivers->Amlogic Device Drivers->Amlogic Display Driver->Amlogic Backlight Support ->Amlogic backlight support
- 亮度设置函数位于bsp.c
- static void aml_8726m_set_bl_level(unsigned level)
- 上层传下来的level范围是0 - 255

8. 开机logo

- Driver位于drivers\amlogic\display\aml_logo
- Device Drivers->Amlogic Device Drivers->Amlogic Display Driver->setup logo
- 需要在启动参数里面添加相应的参数
- setenv bootcmd 'mmcinfo 0;fatload mmc 0 84100000 logo.bmp;fatload mmc 0 82000000 ulmage 400000;bootm'
- setenv bootargs root=/dev/cardblksd2 rw rootfstype=ext3 init=/init console=ttyS0,115200 logo=osd1,0x84100000,lcd,full
- http://openlinux.amlogic.com/wiki/index.php/Arc/Kernel_Info/How_to_build_startup_logo

Chapter9:

Amlogic mid 基本指南(二)

(Heming)

1. Kernel添加自己的config

- 在make menuconfig修改kernel配置之后，配置会保存到kernel/.config 文件，如果想要添加一个新的config可以把kernel/.config 复制到kernel/arch/arm/config，改成meson_ref***_defconfig模式的文件名（必须全小写），就可以在make help时看到提示

2. Kernel添加新的驱动

- 要向kernel里面添加新的驱动，需要修改Kconfig、和相应的makefile。例如：添加一个atheros的wifi驱动
- 1. 找到kernel/drivers/amlogic/wifi 添加相应的文件夹
- 2. 修改kernel/drivers/amlogic/wifi 的Kconfig，添加新的config名称和depend关系及简介
- 3. 修改kernel/drivers/amlogic/wifi 的makefile，包含驱动文件夹的makefile
- 这样在make menuconfig的时候就可以看到添加的驱动了，可以选择编译

Rootfs添加新的工程配置

- 编译rootfs的时候运行lunch可以看到可供选择的工程（eng和user2种模式），如果想自己添加一个新的工程来编译需要进行以下修改：
 1. 修改android/device/amlogic/vendorsetup.sh，添加新工程的eng和user模式
 2. 新建一个***ref的文件夹，可以拷贝原来的项目然后修改
 3. AndroidProducts.mk里面要指定匹配的***ref.mk，***ref.mk里面PRODUCT_NAME和PRODUCT_DEVICE也要匹配***ref
 4. BoardConfig.mk里面TARGET_TOUCH_CALIBRATION_METHOD和TARGET_SUPPORT_MULTI_TOUCH

Rootfs添加新的工程配置

- 根据实际情况配置
- 5. `set_display_mode.sh` 里的`panel`根据实际屏幕大小配置
- 6. `system.prop` 里的`ro.sf.gsensorposition`根据实际gsensor相对主板位置填写 (0~7)
- 7. `AndroidBoard.mk`里的`init.rc`改成相应的`init.***ref.rc`, `initlogo.rle.bak`改成实际适合屏幕大小的logo
- 最后重新运行`build/envsetup.sh`就可以lunch自己添加的工程编译了

Chapter10:

Amlogic mid 基本指南(三)

(Sandy)

1如何在menuconfig添加自己的BSP和屏的dirver.

- 1) drivers\amlogic\display\vout修改Kconfig和Makefile;
- 2) arch\arm\mach-meson修改Kconfig和Makefile;

如何修改ADC面板按键？

- kernel\include\linux\input.h有相关键值的宏，然后和device\amlogic\mlref\aml-usbkbd.kl里实际意义对应，添第1个值，如KEY_TAB。“exit”是注释，CHAN_4是ADC通道。如果只有6个键，3.3V电压分成5份，我们的ADC精度是10 bit, 即1023. 如下方面添ADC值。

- MENU 0 0
- VOL+ 0.58 $0.58 \times 1023 / 3.3 = 179$
- VOL- 0.92 $0.92 \times 1023 / 3.3 = 285$
- SEARCH 1.29 $1.29 \times 1023 / 3.3 = 400$
- EXIT 1.67 $1.67 \times 1023 / 3.3 = 518$
- HOME 2.09 $2.09 \times 1023 / 3.3 = 648$

如何修改开机的绿色机器人logo?

1. 通过Ubuntu的命令 `sudo apt-get install imagemagick` 安装 `imagemagick` 工具。
2. 做好一张与屏的分辨率一样大小的开机 `logo.png`。
3. 在虚拟机中用命令 `convert -depth 8 logo.png rgb:logo.raw` 把 `logo.png` 转化成 `logo.raw`。
4. 把生成的 `logo.raw` 放到 `rootfs` 的根目录下。
5. 在 `rootfs` 下运行命令 `out/host/linux-x86/bin/rgb2565 -rle < logo.raw > initlogo.rle.bak`, 即可生成文件 `initlogo.rle.bak`。
6. 把生成的该文件放到 `\aml_8726_rootfs\out\target\product\mlref\root` 下面即可。

如何在set_display_mode.sh里修改屏的分辨率
，例如现在屏是1024*600的分辨率？

panel)

```
/system/bin/busybox fbset -fb  
/dev/graphics/fb0 -g 1024 600 1024 1200 16 -rgba  
5/11,6/5,5/0,0/0  
echo $1 > /sys/class/display/mode  
echo 0 0 1024 600 0 0 18 18 >  
/sys/class/display/axis
```


在什么文件如何修改GSENSOR方向？

system.prop里修改ro.sf.gsensorposition的值，
如果GSENSOR IC是正面紧靠LCD，用0, 1, 2, 3值；
如果GSENSOR IC是反面紧靠LCD，用4, 5, 6, 7值；

如果是电容屏，如何去掉开机后的5点触屏校准？

BoardConfig.mk里

TARGET_TOUCH_CALIBRATION_METHOD := none

Distribute to SkyMoon

如果是电容屏，如何enable浏览器里两点touch放大？

BoardConfig.mk里

TARGET_SUPPORT_MULTI_TOUCH := true

在那里修改touch 屏的I2C的地址？

- Bsp里static struct i2c_board_info __initdata
aml_i2c_bus_info[] = {}根据touch 屏spec填写地址，例如
- #ifdef CONFIG_ITK_CAPACITIVE_TOUCHSCREEN
- {
- I2C_BOARD_INFO("itk", 0x41),
- .irq = INT_GPIO_0,
- .platform_data = (void *)&itk_pdata,
- },
- #endif

Chapter11

扩展外设的应用及实践

(Frank)

涉及内容

- 1 Android三种架构
- 2 HAL
- 3 实例

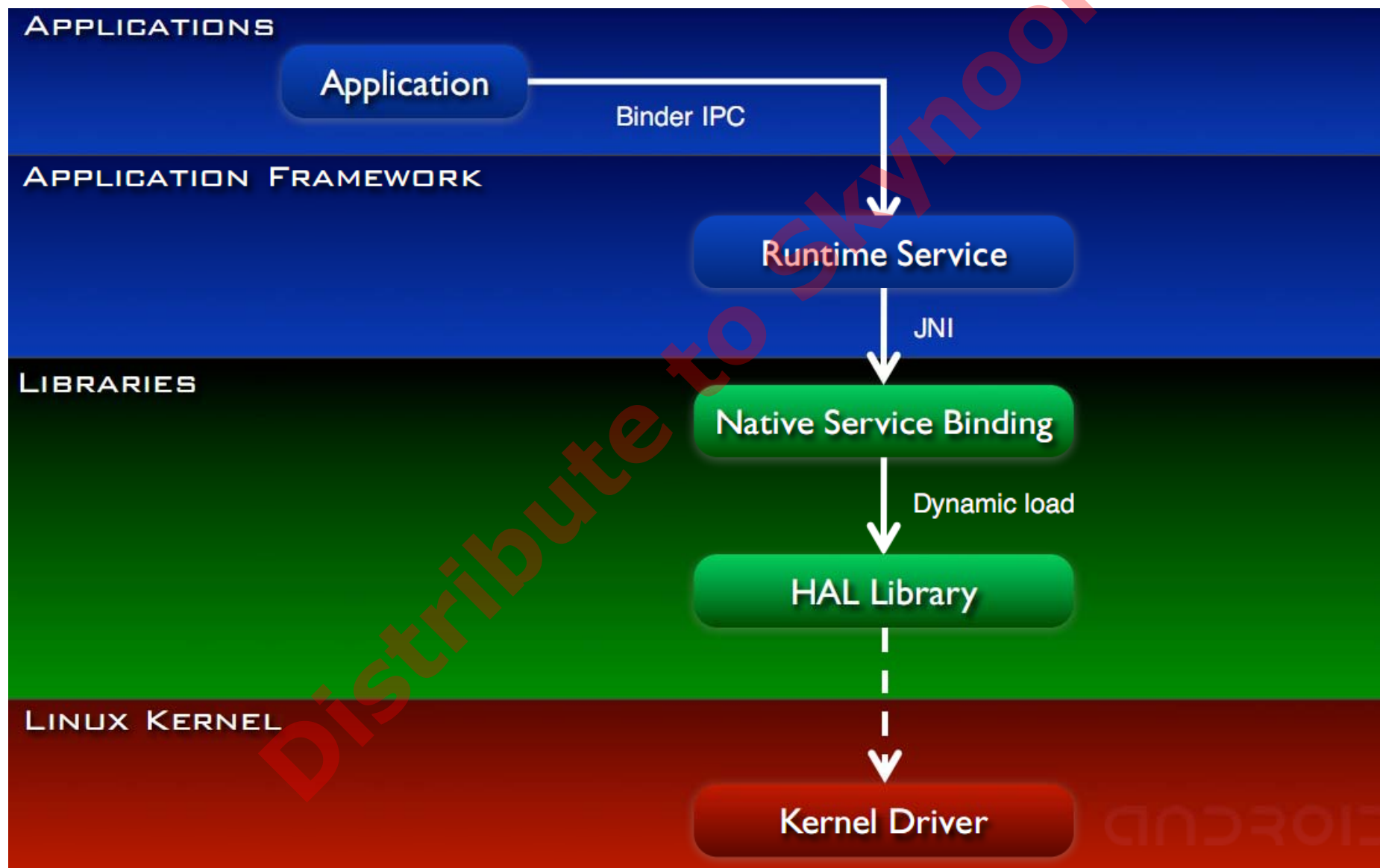
Distribute to Skynoon!

Android的三种架构

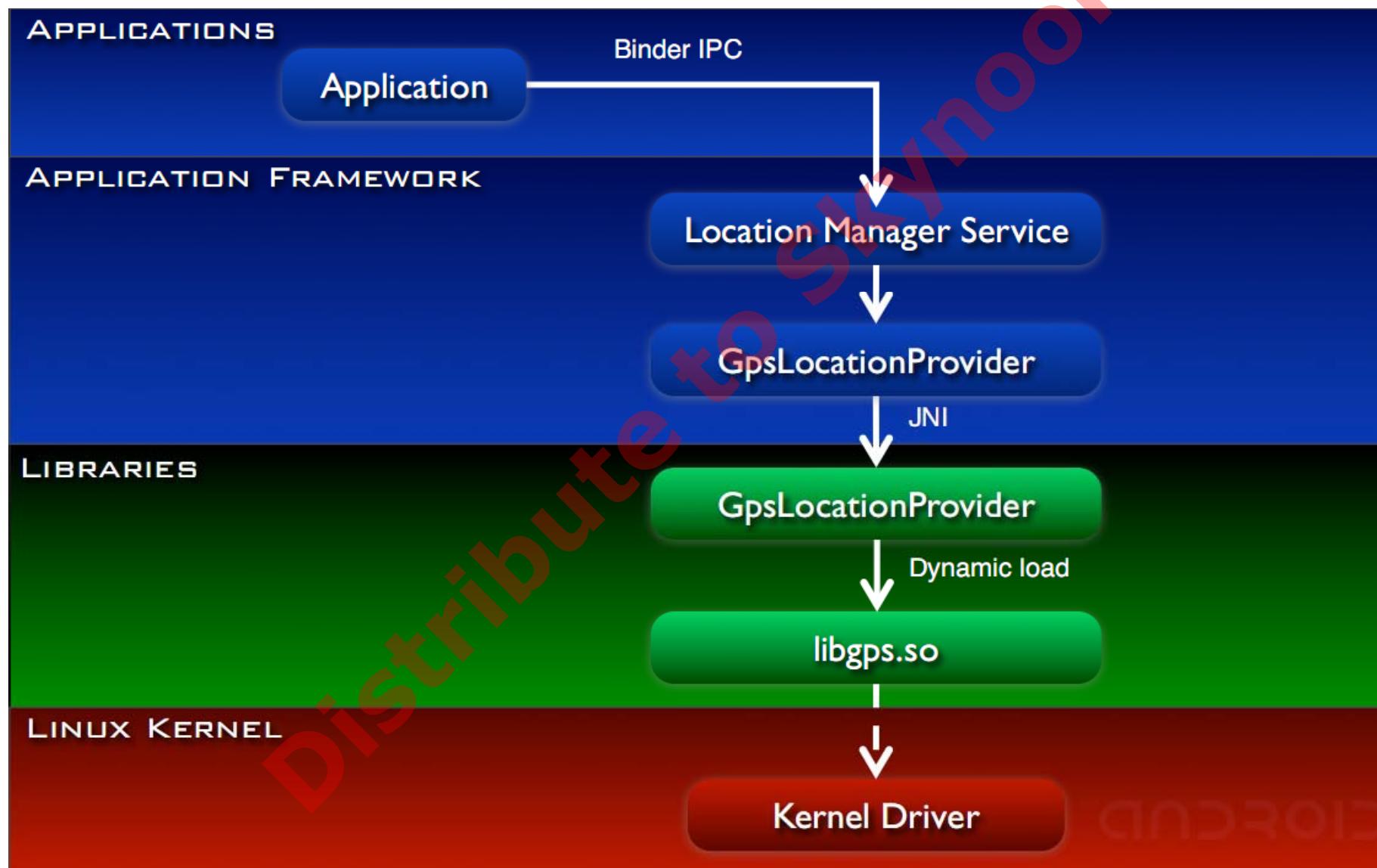
- App->Runtime Service in Java→Lib
- App->Runtime Service in Java->Native Service→lib
- App->Runtime Service in Java->Native Deamen->lib

Distribute to SKYNOON!

Android的三种架构(2)

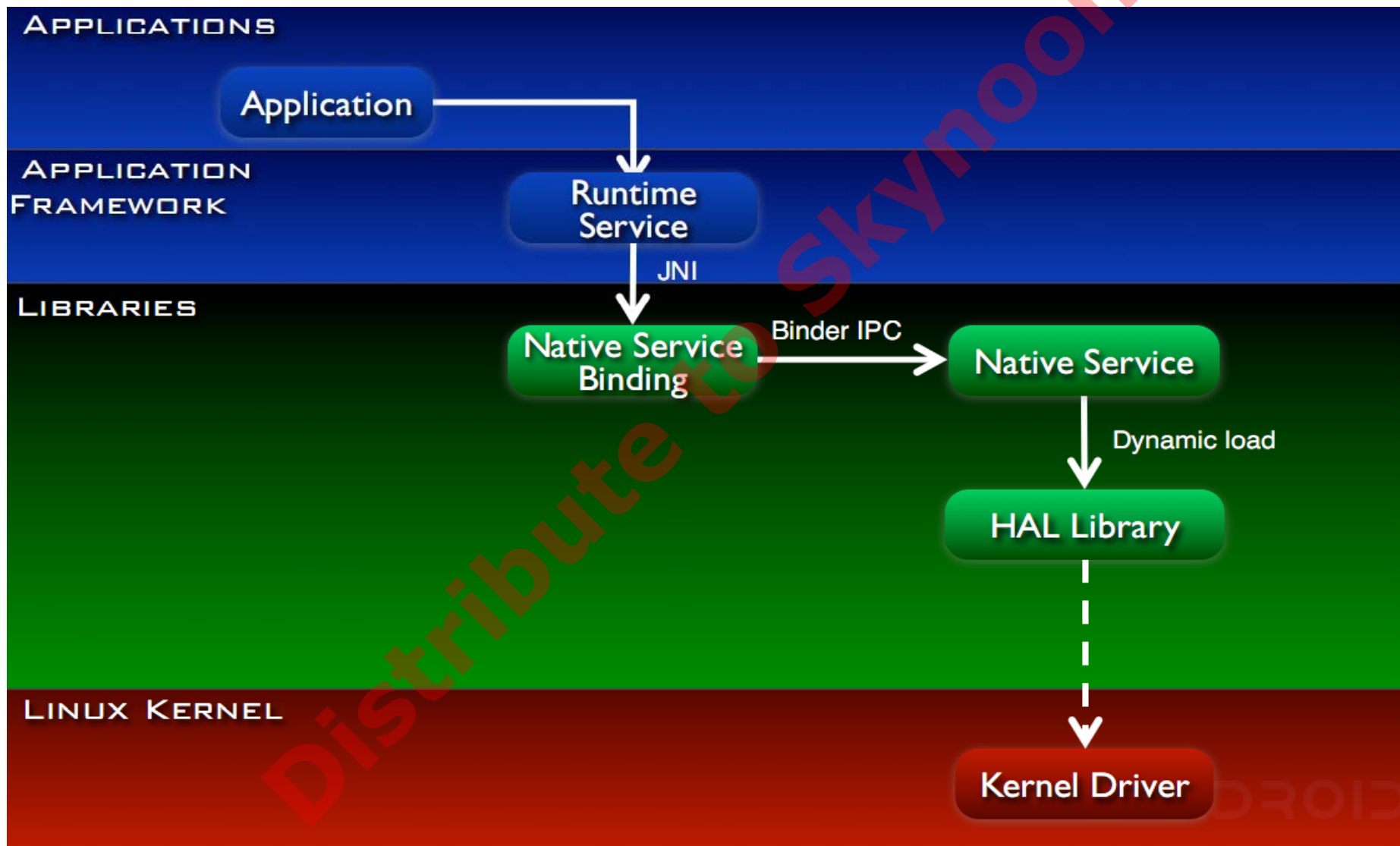


Android的三种架构(3)

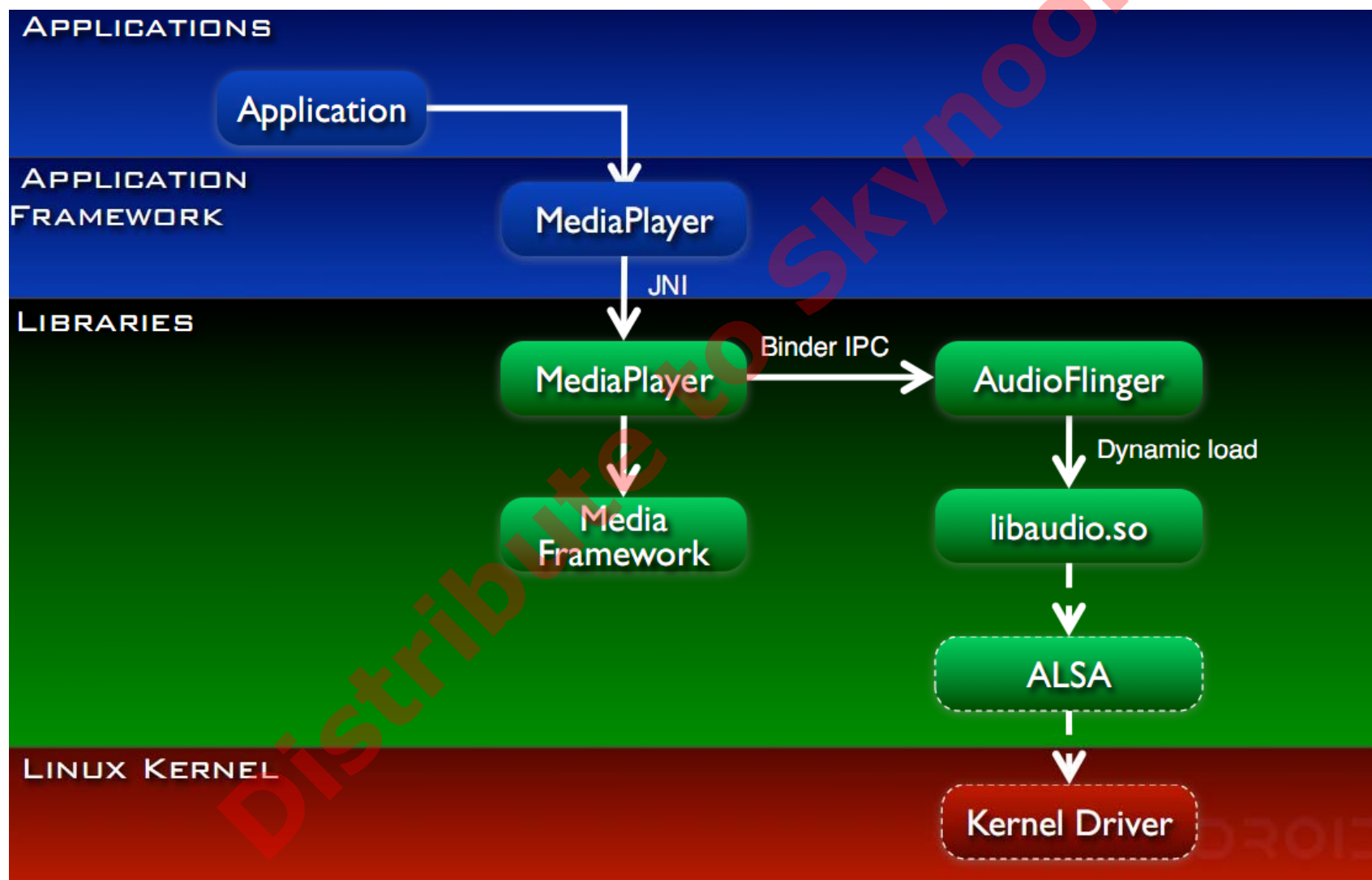


@mlogic

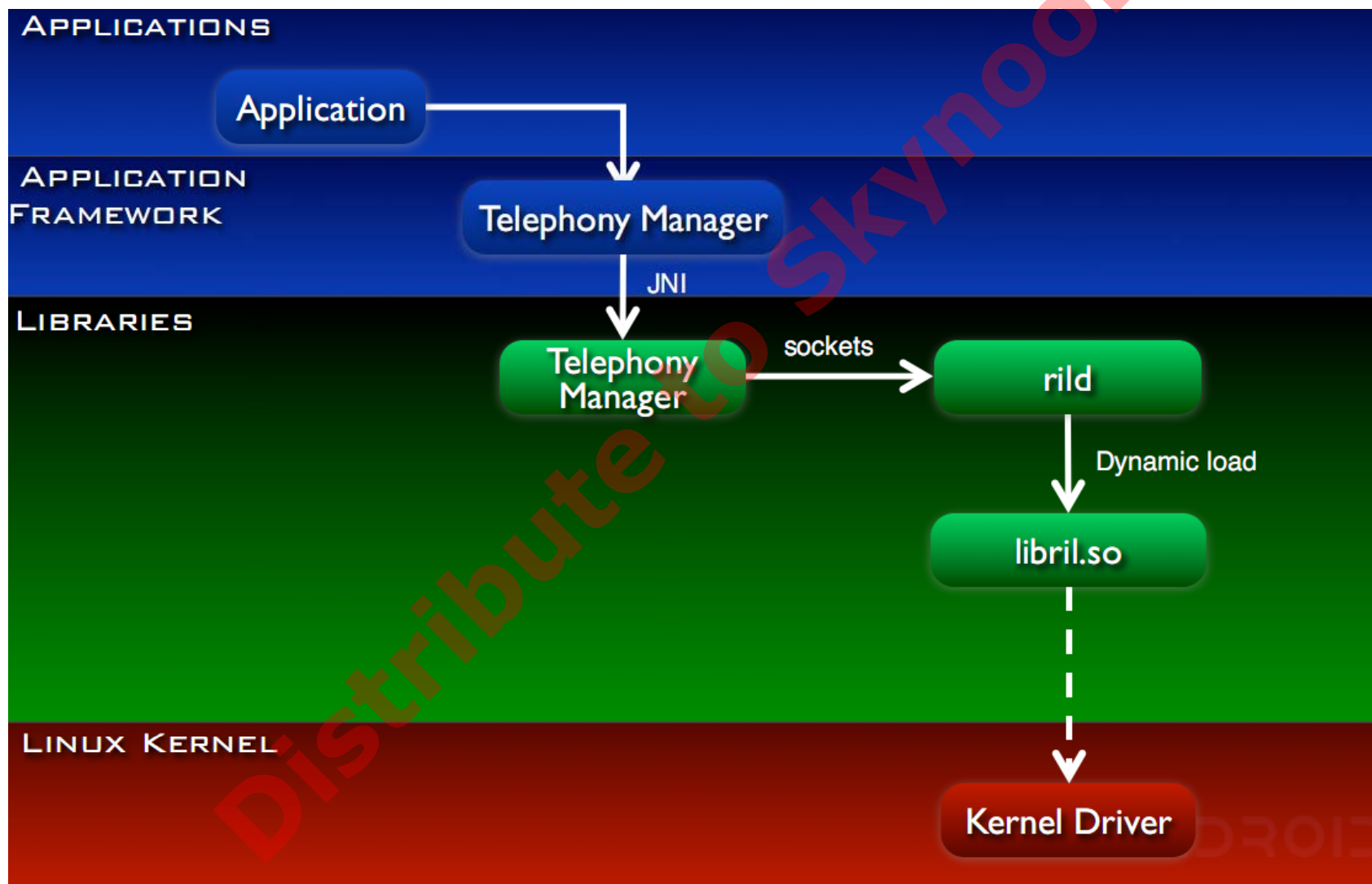
Android的三种架构(4)



Android的三种架构 (5)



Android的三种架构(6)



认识HAL

- 代码位置: hardware/libhardware
- `int hw_get_module(const char *id, const struct hw_module_t **module);`
- `typedef struct hw_module_t {`
 - ...
- `struct hw_module_methods_t* methods;`
- ...
- `} hw_module_t;`

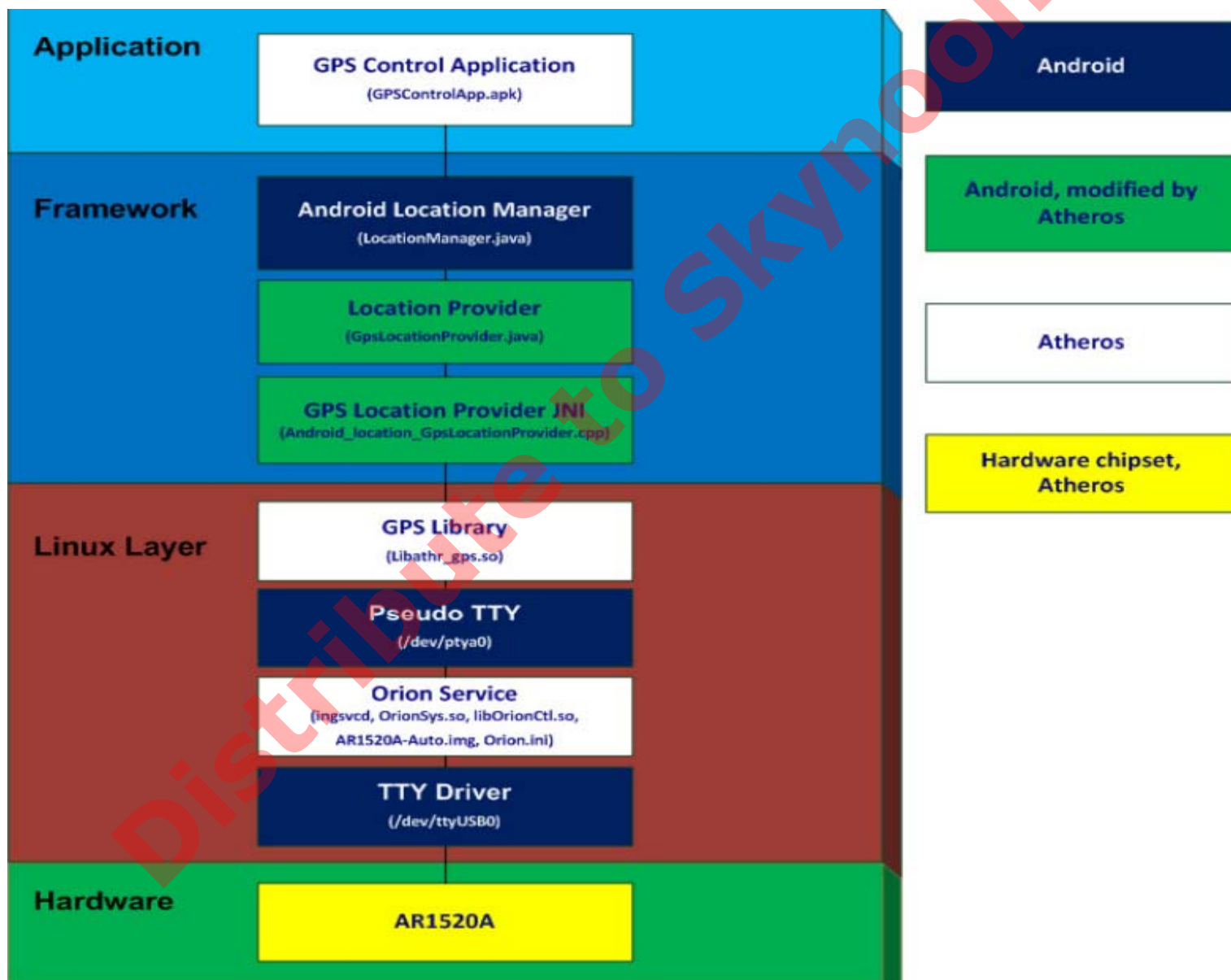
认识HAL (2)

- `typedef struct hw_module_methods_t {`
- `int (*open)(const struct hw_module_t* module, const char* id,`
- `struct hw_device_t** device);`
- `} hw_module_methods_t;`

例子-lights

- 代码
- Framework/base/services/jni/com_android_server_LightsService.cpp
- Hardware/hardware/libhardware/include/hardware/lights.h
- Hardware/amlogic/lights

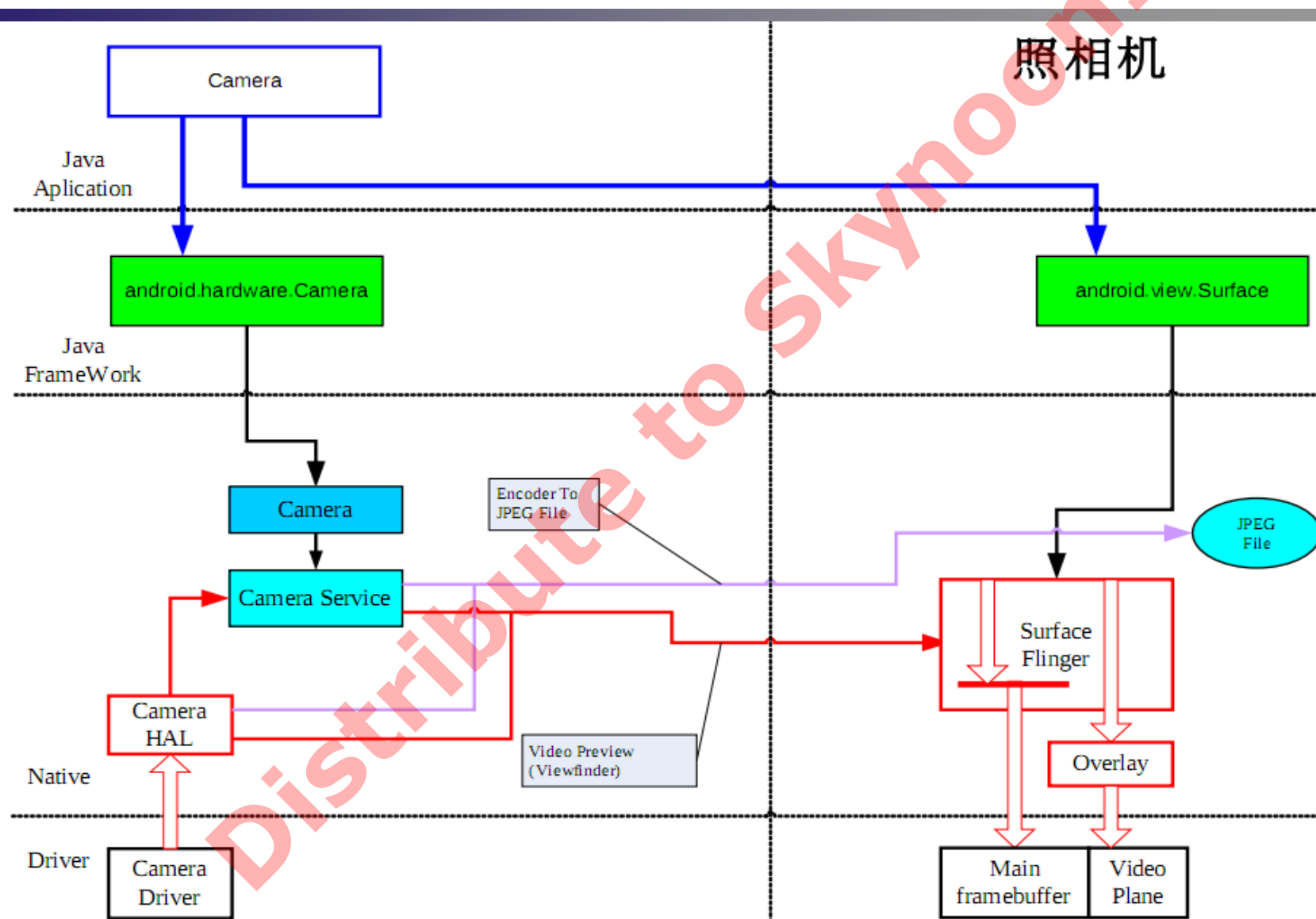
例子GPS



例子GPS

- 代码位置
- frameworks/base/location/java/com/android/internal/location/GpsLocationProvider.java
- frameworks/base/core/jni/android_location_GpsLocationProvider.cpp
- Hardware/libhardware_legacy/gps
- Hardware/amlogic/gps

例子Camera



例子 camera

- 代码位置
- frameworks/base/camera
- frameworks/base/libs/camera
- Frameworks/base/ include/camera

Chapter12:

应用开发与实践

(Jay)

eclipse下导入android源代码

1. 把eclipse工程配置文件复制到android源码根目录下

```
cp development/ide/eclipse/.classpath ./
```

2. 修改eclipse程序的配置

1)、修改eclipse缓存设置

把eclipse.ini（在eclipse软件的安装目录下）的3个值改为下面的值：

```
-Xms128m
```

```
-Xmx512m
```

```
-XX:MaxPermSize=256m
```

2)、把android-formatting.xml和android.importorder导入eclipse

android-formatting.xml和android.importorder都放在development/ide/eclipse/下

android-formatting.xml用来配置eclipse编辑器的代码风格；android.importorder用来配置eclipse的import的顺序和结构。

在window->preferences->java->Code style->Formatter中导入android-formatting.xml

在window->preferences->java->Code style->Organize Imports中导入

android.importorder

3. 把android源码作为一个工程导入eclipse

新建Java Project（不是android project），选择从已存在的工程导入，工程名任意，完成。

导入时，eclipse要build工程，比较慢。

Android的内部资源以及重编命令

1. Framework 代码中 以`com.android.internal.R`开头的内部资源都在 `frameworks\base\core\res\res`下面。
2. 改动这部分的代码，用命令：`make framework-res.apk` 来重新编译。
3. 改动`frameworks\policies`下面的代码，用命令：`make android.policy` 来重新编译。
4. 改动`frameworks\base\services`下面的代码，用命令：`make services` 来重新编译。
5. 或者改动的目录下 使用`mm`命令

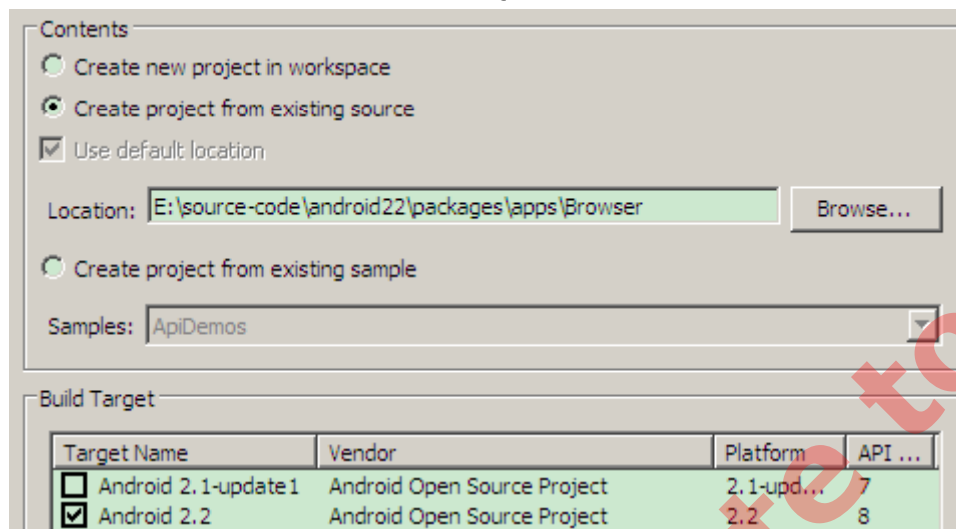
Package下面app改动

1. AndroidManifest.xml 改动不多。一般都是activity的属性
2. res\layout 或res\xml 目录下的布局文件。
3. Src 目录下的java源代码文件。
4. 在代码中通过一个参数来控制某个功能是否启用或在列表中去掉某一项。

```
SystemProperties.getBoolean("ro.monkey", false);  
(device\amlogic\mlref\system.prop)
```

Eclipse导入单个项目（1）

1. New→Android Project



Eclipse导入单个项目（2）

2. 由于在Android源码中，很多方法、成员、类、包都被打上@hide标签，这些成员在SDK中没有公开，以至于在编译Launcher源码时最常遇到的类android.view.View的成员mScrollX无法访问。

下面说说如何解决这个问题。

1, 准备好编译后的Android源码。

2, 在该源码的out目录下寻找包含你所用隐藏类的jar文件，通常文件名为classes.jar。例如framework的jar文件为 out\target\common\obj\JAVA_LIBRARIES\framework_intermediates\classes.jar。

3, 在eclipse的Android项目中，选择项目属性->Java Build Path->Libraries->Add Library->User Library->Next-> User Libraries进入到User Libraries管理界面，点击New新建一个User Library，比如android_framework，点击Add Jars把Jar包加入到建立的User Library中，最后点击OK就可以了。

注意：为了访问因此成员，需要改变类搜索顺序，选择项目属性->Java Build Path->Order and Export，把所建立的User Libraries移到Android SDK的上面。

这个时候你的eclipse中的错误应该已经减少，甚至没有了。

要想在模拟器上马上看效果的话，按照以下方式进行修改：

改掉原始包的名字，切记使用eclipse的重命名机制（在包名上按F2可修改），不仅是类的引用，还有很多xml文件内部的引用（如import com.android.launcher3.R;），只要重命名不错，这些都可以一次性搞定的。最后在AndroidManifest.xml文件里面，找到这句话删除掉（android:sharedUserId="android.uid.shared"）。到现在为止，你就拥有了自己的 Launcher了！

Chapter13:

Android 四大组件

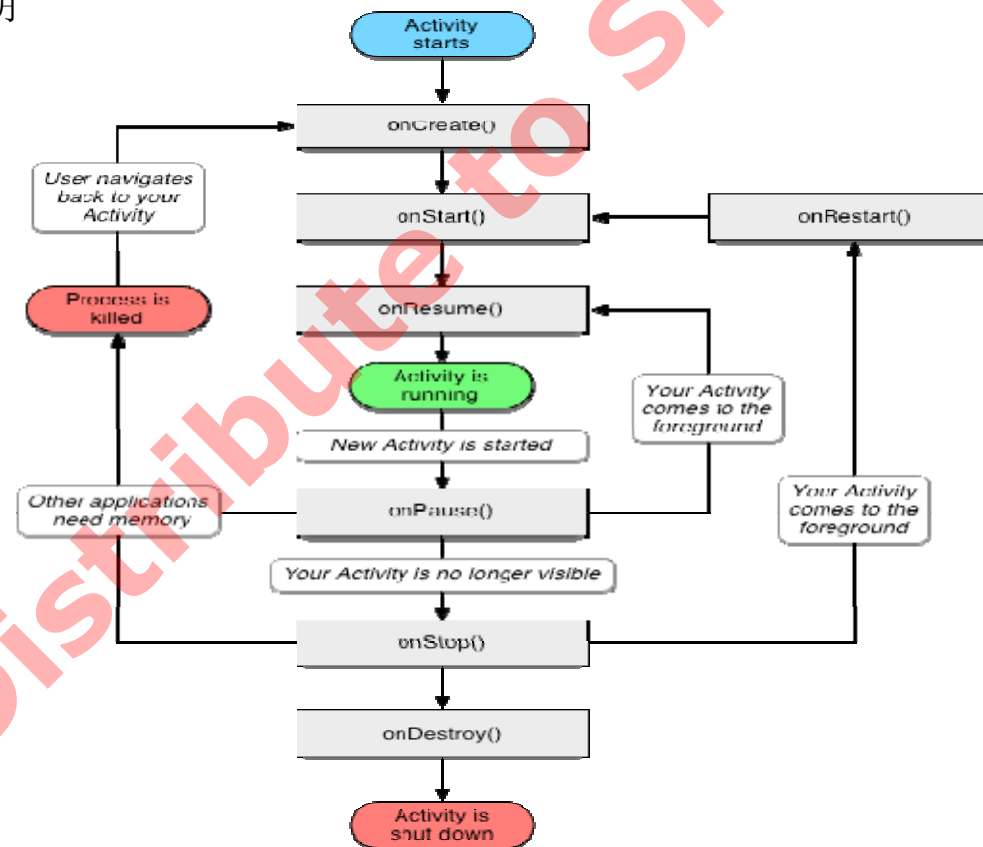
(Jay)

Android 之 Intent

- Android基本的设计理念是鼓励减少组件间的耦合，因此Android提供了Intent（意图），Intent提供了一种通用的消息系统，它允许在你的应用程序与其它的应用程序间传递Intent来执行动作和产生事件。使用Intent可以激活Android应用的三个核心组件：活动、服务和广播接收器。
- Intent可以划分成显式意图和隐式意图。
- 显式意图：调用Intent.setComponent()或Intent.setClass()方法指定了组件名或类对象的Intent为显式意图，显式意图明确指定了Intent应该传递给哪个组件。
- 隐式意图：没有调用Intent.setComponent()或Intent.setClass()方法指定组件名或类对象的Intent为隐式意图。Android系统会根据隐式意图中设置的**动作(action)、类别(category)、数据（URI和数据类型）**找到最合适的组件来处理这个意图。那么Android是怎样寻找到这个最合适的组件呢？记的前面我们在定义活动时，指定了一个intent-filter，Intent Filter（过滤器）其实就是用来匹配隐式Intent的，如果Intent Filter定义的动作、类别、数据（URI和数据类型）与Intent匹配，就会使用Intent Filter所在的组件来处理该Intent。想要接收使用startActivity()方法传递的隐式意图的活动必须在它们的意图过滤器中包含“android.intent.category.DEFAULT”

Android 四大组件之 Activity

1. 一个Activity通常展现为一个可视化的用户界面。
2. 展示activity窗口的可视化内容区域是一些具有层次关系（很像数据结构中的树）的视图，而视图则是由类View的子类表示的。
3. Activity生命周期



Android 四大组件之 Activity

```
public class Activity extends ApplicationContext {  
    protected void onCreate(Bundle savedInstanceState);  
    protected void onStart();  
    protected void onRestart();  
    protected void onResume();  
    protected void onPause();  
    protected void onStop();  
    protected void onDestroy();  
}
```

- 打开应用时先后执行了onCreate()->onStart()->onResume三个方法
- 按BACK键时，我们这个应用程序将结束，这时候我们将先后调用onPause()->onStop()->onDestroy()三个方法
- 按HOME的时候，Activity先后执行了onPause()->onStop()这两个方法，这时候应用程序并没有销毁。当我们再次启动ActivityDemo应用程序时，则先后分别执行了onRestart()->onStart()->onResume()三个方法

Android 四大组件之 Activity

- 在一个Activity中可以使用系统提供的startActivity(Intent intent)方法打开新的Activity, 在打开新的Activity前, 你可以决定是否为新的Activity传递参数:

第一种: 打开新的Activity, 不传递参数

```
public class MainActivity extends Activity {  
    @Override protected void onCreate(Bundle savedInstanceState) {  
        .....  
        Button button =(Button) this.findViewById(R.id.button);  
        button.setOnClickListener(new View.OnClickListener() {  
            // 打开一个新的Activity  
            public void onClick(View v) {  
                // 打开一个新的Activity类  
                //新建一个显式意图, 第一个参数为当前Activity类对象, 第二个参数为你要  
                startActivity(new Intent(MainActivity.this, NewActivity.class));  
            }  
        });  
    }  
}
```

Android 四大组件之 Activity

第二种：打开新的Activity，并传递若干个参数给它：

```
public class MainActivity extends Activity {  
    @Override protected void onCreate(Bundle savedInstanceState) {  
        .....  
        button.setOnClickListener(new View.OnClickListener() { //点击该按钮会打开一个新的Activity  
            public void onClick(View v) {  
                Intent intent = new Intent(MainActivity.this, NewActivity.class)  
  
                Bundle bundle = new Bundle(); //该类用作携带数据  
                bundle.putString("name", "andorid");  
                bundle.putInt("age", 4);  
                intent.putExtras(bundle); //附上额外的数据  
                startActivity(intent);  
            }  
        });  
    }  
}
```

在新的Activity中接收前面Activity传递过来的参数：

```
public class NewActivity extends Activity {  
    @Override protected void onCreate(Bundle savedInstanceState) {  
        .....  
        Bundle bundle = this getIntent().getExtras();  
        String name = bundle.getString("name");  
        int age = bundle.getInt("age");  
    }  
}
```


Android 四大组件之 Activity

- 如果你想在Activity中得到新打开Activity 关闭后返回的数据，你需要使用系统提供的startActivityForResult(Intent intent, int requestCode)方法打开新的Activity，新的Activity 关闭后会向前面的Activity 传回数据，为了得到传回的数据，你必须在前面的Activity中重写onActivityResult(int requestCode, int resultCode, Intent data)方法：

```
public class MainActivity extends Activity {  
    @Override protected void onCreate(Bundle savedInstanceState) {  
        .....  
        Button button =(Button) this.findViewById(R.id.button);  
        button.setOnClickListener(new View.OnClickListener() {  
            public void onClick(View v) {  
                //第二个参数为请求码，可以根据业务需求自己编号  
                startActivityForResult (new Intent(MainActivity.this, NewActivity.class), 1);  
            }  
        });  
        //第一个参数为请求码，即调用startActivityForResult()传递过去的值  
        //第二个参数为结果码，结果码用于标识返回数据来自哪个新Activity  
        @Override protected void onActivityResult(int requestCode, int resultCode, Intent data) {  
            String result = data.getExtras().getString("result");//得到新Activity 关闭后返回的数据  
        }  
    }  
}
```

当新Activity关闭后，新Activity返回的数据通过Intent进行传递，android平台会调用前面Activity 的onActivityResult()方法，把存放了返回数据的Intent作为第三个输入参数传入，在onActivityResult()方法中使用第三个输入参数可以取出新Activity返回的数据。

Android 四大组件之 Activity

- 使用startActivityForResult(Intent intent, int requestCode)方法打开新的Activity, 新Activity关闭前需要向前面的Activity返回数据需要使用系统提供的setResult(int resultCode, Intent data)方法实现:

```
public class NewActivity extends Activity {  
    @Override protected void onCreate(Bundle savedInstanceState) {  
        .....  
        button.setOnClickListener(new View.OnClickListener() {  
            public void onClick(View v) {  
                Intent intent = new Intent();//数据是使用Intent返回  
                intent.putExtra("result", "传智播客的学生很可爱");//把返回数据存入Intent  
                NewActivity.this.setResult(RESULT_OK, intent);//设置返回数据  
                NewActivity.this.finish();//关闭Activity  
            }  
        });  
    }  
}
```

setResult()方法的第一个参数值可以根据业务需要自己定义, 上面代码中使用到的RESULT_OK是系统Activity类定义的一个常量, 值为-1, 代码片断如下:

```
public class android.app.Activity extends ..... {  
    public static final int RESULT_CANCELED = 0;  
    public static final int RESULT_OK = -1;  
    public static final int RESULT_FIRST_USER = 1;  
}
```

Android 四大组件之 Service

- Android中的服务和windows中的服务是类似的东西，服务一般没有用户操作界面，它运行于系统中不容易被用户发觉，可以使用它开发如监控之类的程序。服务的开发比较简单，如下：
- 第一步：继承Service类
- ```
public class SMSService extends Service { }
```
- 第二步：在AndroidManifest.xml文件中的<application>节点里对服务进行配置：
- ```
<service android:name=".SMSService" />
```
- 服务不能自己运行**，需要通过调用Context.startService()或Context.bindService()方法启动服务。这两个方法都可以启动Service，但是它们的使用场合有所不同。使用startService()方法启用服务，调用者与Service之间没有关联，即使调用者退出了，服务仍然运行。使用bindService()方法启用服务，调用者与Service绑定在了一起，调用者一旦退出，服务也就终止，大有“不求同时生，必须同时死”的特点。
- Android Service的生命周期并不像Activity那么复杂，它只继承了onCreate(), onStart(), onDestroy()三个方法。
- 如果打算采用Context.startService()方法启动服务，在服务未被创建时，系统会先调用服务的onCreate()方法，接着调用onStart()方法。如果调用startService()方法前服务已经被创建，**多次调用startService()方法并不会导致多次创建服务，但会导致多次调用onStart()方法**。采用startService()方法启动的服务，只能调用Context.stopService()方法结束服务，服务结束时会调用onDestroy()方法。
- 如果打算采用Context.bindService()方法启动服务，在服务未被创建时，系统会先调用服务的onCreate()方法，接着调用onBind()方法。这个时候调用者和Service绑定在一起，调用者退出了，系统就会先调用服务的onUnbind()方法，接着调用onDestroy()方法。如果调用bindService()方法前服务已经被绑定，多次调用bindService()方法并不会导致多次创建服务及绑定(也就是说onCreate()和onBind()方法并不会被多次调用)。如果调用者希望与正在绑定的Service解除绑定，可以调用unbindService()方法，调用该方法也会导致系统调用服务的onUnbind()-->onDestroy()方法。

Android 四大组件之 Service

- 服务常用生命周期回调方法如下：
- `onCreate()` 该方法在服务被创建时调用，该方法只会被调用一次，无论调用多少次 `startService()` 或 `bindService()` 方法，服务也只被创建一次。
- `onDestroy()` 该方法在服务被终止时调用。
- 与采用 `Context.startService()` 方法启动服务有关的生命周期方法
- `onStart()` 只有采用 `Context.startService()` 方法启动服务时才会回调该方法。该方法在服务开始运行时被调用。多次调用 `startService()` 方法尽管不会多次创建服务，但 `onStart()` 方法会被多次调用。
- 与采用 `Context.bindService()` 方法启动服务有关的生命周期方法
- `onBind()` 只有采用 `Context.bindService()` 方法启动服务时才会回调该方法。该方法在调用者与服务绑定时被调用，当调用者与服务已经绑定，多次调用 `Context.bindService()` 方法并不会导致该方法被多次调用。
- `onUnbind()` 只有采用 `Context.bindService()` 方法启动服务时才会回调该方法。该方法在调用者与服务解除绑定时被调用。

Android 四大组件之 Service

- 采用startService() 启动服务
- 采用Context.startService() 方法启动服务的代码如下：
- ```
public class HelloActivity extends Activity {
 @Override
 public void onCreate(Bundle savedInstanceState) {

 Button button =(Button) this.findViewById(R.id.button);
 button.setOnClickListener(new View.OnClickListener() {
 public void onClick(View v) {
 Intent intent = new Intent(HelloActivity.this, SMSService.class);
 startService(intent);
 }
 });
 }
}
```

# Android 四大组件之 Service

- 采用bindService() 启动服务
- 采用Context.startService() 方法启动服务的代码如下:
- ```
public class HelloActivity extends Activity {  
    ServiceConnection conn = new ServiceConnection() {  
        public void onServiceConnected(ComponentName name, IBinder service) {  
        }  
        public void onServiceDisconnected(ComponentName name) {  
        }  
    };  
    @Override public void onCreate(Bundle savedInstanceState) {  
        Button button =(Button) this.findViewById(R.id.button);  
        button.setOnClickListener(new View.OnClickListener() {  
            public void onClick(View v) {  
                Intent intent = new Intent(HelloActivity.this, SMSService.class);  
                bindService(intent, conn, Context.BIND_AUTO_CREATE);  
                //unbindService(conn); //解除绑定  
            }  
        });  
    }  
}
```

Android 四大组件之 BroadcastReceiver

- 广播接收者（BroadcastReceiver）用于异步接收广播Intent，广播Intent的发送是通过调用Context.sendBroadcast()、Context.sendOrderedBroadcast()或者Context.sendStickyBroadcast()来实现的。通常一个广播Intent可以被订阅了此Intent的多个广播接收者所接收，广播接收者和JMS中的Topic消息接收者很相似。要实现一个广播接收者方法如下：

- 第一步：继承BroadcastReceiver，并重写onReceive()方法。

- ```
public class IncomingSMSReceiver extends BroadcastReceiver {
```

- ```
    @Override public void onReceive(Context context, Intent intent) {
```

- ```
 }
```

- ```
}
```

- 第二步：订阅感兴趣的广播Intent，订阅方法有两种：

- 第一种：使用代码进行订阅

- ```
IntentFilter filter = new IntentFilter("android.provider.Telephony.SMS_RECEIVED");
```

- ```
IncomingSMSReceiver receiver = new IncomingSMSReceiver();
```

- ```
registerReceiver(receiver, filter);
```

- 第二种：在AndroidManifest.xml文件中的<application>节点里进行订阅：

- ```
<receiver android:name=".IncomingSMSReceiver">
```

- ```
 <intent-filter>
```

- ```
        <action android:name="android.provider.Telephony.SMS_RECEIVED"/>
```

- ```
 </intent-filter>
```

- ```
</receiver>
```

Android 四大组件之 BroadcastReceiver

- Intent, Android还有很多广播Intent, 如: 开机启动、电池电量变化、时间已经改变等广播Intent。
- 通常一个BroadcastReceiver对象的生命周期不超过5秒, 所以在BroadcastReceiver里不能做一些比较耗时的操作, 如果需要完成一项比较耗时的工作, 可以通过发送Intent给Activity或服务, 由Activity或服务来完成。
- ```
public class IncomingSMSReceiver extends BroadcastReceiver {
```
- ```
    @Override public void onReceive(Context context, Intent intent) {
```

 - ```
 //发送Intent启动服务, 由服务来完成比较耗时的操作
```
  - ```
        Intent service = new Intent(context, XxxService.class);
```
 - ```
 context.startService(service);
```
  - ```
        //发送Intent启动Activity, 由Activity来完成比较耗时的操作
```
 - ```
 Intent newIntent = new Intent(context, XxxActivity.class);
```
  - ```
        context.startActivity(newIntent);
```
- ```
 }
```
- ```
}
```


Android 四大组件之 ContentProvider

- 当应用继承ContentProvider类，并重写该类用于提供数据和存储数据的方法，就可以向其他应用共享其数据。虽然使用其他方法也可以对外共享数据，但数据访问方式会因数据存储的方式而不同，如：采用文件方式对外共享数据，需要进行文件操作读写数据；采用sharedpreferences共享数据，需要使用sharedpreferences API读写数据。而使用ContentProvider共享数据的好处是统一了数据访问方式。
- 当应用需要通过ContentProvider对外共享数据时，第一步需要继承ContentProvider并重写下面方法：
- ```
public class PersonContentProvider extends ContentProvider{
```
- ```
    public boolean onCreate()
```
- ```
 public Uri insert(Uri uri, ContentValues values)
```
- ```
    public int delete(Uri uri, String selection, String[] selectionArgs)
```
- ```
 public int update(Uri uri, ContentValues values, String selection, String[] selectionArgs)
```
- ```
    public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)
```
- ```
 public String getType(Uri uri)}
```
- 第二步需要在AndroidManifest.xml使用<provider>对该ContentProvider进行配置，为了能让其他应用找到该ContentProvider，ContentProvider 采用了authorities（主机名/域名）对它进行唯一标识，你可以把 ContentProvider看作是一个网站（想想，网站也是提供数据者），authorities 就是他的域名：
- ```
<manifest .... >
```
- ```
 <application android:icon="@drawable/icon" android:label="@string/app_name">
```
- ```
        <provider android:name=".PersonContentProvider" android:authorities="cn.itcast.provider.personprovider"/>
```
- ```
 </application>
```
- ```
</manifest>
```
- 注意：一旦应用继承了ContentProvider类，后面我们就会把这个应用称为ContentProvider（内容提供者）。

Android 四大组件之 ContentProvider

- ContentProvider类主要方法的作用：
- `public boolean onCreate()`
 - 该方法在ContentProvider创建后就会被调用， Android在系统启动时就会创建ContentProvider 。
- `public Uri insert(Uri uri, ContentValues values)`
 - 该方法用于供外部应用往ContentProvider添加数据。
- `public int delete(Uri uri, String selection, String[] selectionArgs)`
 - 该方法用于供外部应用从ContentProvider删除数据。
- `public int update(Uri uri, ContentValues values, String selection, String[] selectionArgs)`
 - 该方法用于供外部应用更新ContentProvider中的数据。
- `public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)`
 - 该方法用于供外部应用从ContentProvider中获取数据。
- `public String getType(Uri uri)`
 - 该方法用于返回当前Uri所代表数据的MIME类型。如果操作的数据属于集合类型，那么MIME类型字符串应该以 `vnd.android.cursor.dir/` 开头，例如：要得到所有person记录的Uri为 `content://cn.itcast.provider.personprovider/person`，那么返回的MIME类型字符串应该为：“`vnd.android.cursor.dir/person`”。如果要操作的数据属于单一数据，那么MIME类型字符串应该以 `vnd.android.cursor.item/` 开头，例如：得到id为10的person记录，Uri为 `content://cn.itcast.provider.personprovider/person/10`，那么返回的MIME类型字符串应该为：“`vnd.android.cursor.item/person`”。

Android 四大组件之 ContentProvider

- 使用ContentResolver操作ContentProvider中的数据
- 当外部应用需要对ContentProvider中的数据进行添加、删除、修改和查询操作时，可以使用ContentResolver 类来完成，要获取ContentResolver 对象，可以使用Activity提供的getContentResolver() 方法。 ContentResolver 类提供了与ContentProvider类相同签名的四个方法：
 - `public Uri insert(Uri uri, ContentValues values)`
 - 该方法用于往ContentProvider添加数据。
 - `public int delete(Uri uri, String selection, String[] selectionArgs)`
 - 该方法用于从ContentProvider删除数据。
 - `public int update(Uri uri, ContentValues values, String selection, String[] selectionArgs)`
 - 该方法用于更新ContentProvider中的数据。
 - `public Cursor query(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)`
 - 该方法用于从ContentProvider中获取数据。
- 这些方法的第一个参数为Uri，代表要操作的是哪个ContentProvider和对其中的什么数据进行操作，假设给定的是：`Uri.parse("content://cn.itcast.provider.personprovider/person/10")`，那么将会对主机名为`cn.itcast.provider.personprovider`的ContentProvider进行操作，操作的数据为person表中id为10的记录。

Android 四大组件之 ContentProvider

- 使用ContentResolver对ContentProvider中的数据进行添加、删除、修改和查询操作：

```
ContentResolver resolver = getContentResolver();

Uri uri = Uri.parse("content://cn.itcast.provider.personprovider/person");

//添加一条记录
ContentValues values = new ContentValues();
values.put("name", "itcast");
values.put("age", 25);
resolver.insert(uri, values);

//获取person表中所有记录
Cursor cursor = resolver.query(uri, null, null, null, "personid desc");
while(cursor.moveToNext()){
    Log.i("ContentTest", "personid="+ cursor.getInt(0)+ ",name="+ cursor.getString(1));
}

//把id为1的记录的name字段值更改新为liming
ContentValues updateValues = new ContentValues();
updateValues.put("name", "liming");
Uri updateIdUri = ContentUris.withAppendedId(uri, 2);
resolver.update(updateIdUri, updateValues, null, null);

//删除id为2的记录
Uri deleteIdUri = ContentUris.withAppendedId(uri, 2);
resolver.delete(deleteIdUri, null, null);
```

Chapter14:

MBOX 的Android 平台开发 介绍

(TieJun)

准备工作

1. 下载uboot，rootfs和kernel 代码。

mbox的代码分支为：froyo-amlogic

repo init -u

git://XXX.XXX.XXX.XXX/platform/manif est.git -b

froyo-amlogic

2. 下载编译工具。

3. 了解手头开发板的版本。

目前主要有MBOX_8626M1_V1.0 2010_07_19,

AML_8726-M_DEV_BOARD 2010-10-11_v1.0,

Meson board _AML_8726-M 2010-11-18_v1.1。

编译kernel

1. Make help : 查看我们的得到的和开发板匹配或最相似config。

MACH_MESON_8726M_REFC01:

Meson 板 V1.0,V1.1。它们的主要差别在：
nandflash, wifi控制的GPIO。

对应文件board-8726m-refc01.c

MACH_MESON_8726M_REFC02:

MBOX_8626M1_V1.0 2010_07_19

2. Make menuconfig

注意配置Uart。

3. Make ulmage (-jX)

编译rootfs

1.配置环境变量，选择编译配置

. Build/envsetup.sh

lunch 3或4

注意：如果3和4切换，需先手工把out目录删除。

2.Make

3.制作yaffs格式文件。

mkyaffsimage。

编译uboot

1.选择配置

```
make m1_8726m_ref_config  
uboot\board\amlogic\m1_8726m_ref  
m1_8726m_ref.h中新特征  
#define CONFIG_SWITCH_BOOT_MODE  
#define CONFIG_EFUSE  
#define CONFIG_SARADC 1  
#define CONFIG_UPGRADE 1
```

2.make

烧录code

1.烧录uboot

A.烧录到SPI flash:sf

B.烧录到nandflash:nand rom_write

2.烧录kernel。

3.烧录rootfs。

4.配置启动参数。

```
set bootargs 'a9_clock=800M root=/dev/mtdblock2 rw  
init=/init console=ttyS0'
```

```
set bootcmd 'nand read 0x82000000 0x00800000  
0x00400000;bootm 82000000'
```

启动信息分析

1. Starting kernel ...
2. Creating 5 MTD partitions on "Samsung":
3. ***Entered
sound/soc/aml/aml_m1_armdev_wm8900.c
4. Mali<2>: MMU session begin
Mali<2>: Core: session_begin: for Mali-400
PP
5. Mali<2>: Session has ended

Chapter15:

AML8726-M调试

(Zong Feng)

MID调试——硬件部分

要保证一个系统可正常的工作，首先要确保硬件是正确无误的。

五个重要的环节：

- 原理图设计.
- Layout.
- 元器件不良.
- 制程作业不良.
- 来自外部的ESD/EOS损坏.

硬件几乎所有的问题我们都可以追溯到这五个环节。

调试流程:

1. review原理图/layout, 查看所设计项目功能, 性能, 电源, 连接, 参数选择等方面是否满足设计。这个环节往往是最重要的, 如果考虑周全便可省去后面调试很多的麻烦, 但往往需要在调试中发现问题完善设计。
2. 在拿到平台时, 先不要急于上电, 首先观察板子的焊接状况如何, 有没有短路断路, 器件损坏的问题。利用万用表查看板子有没有短路现象。
3. 在第二步确认无误后可上电查看, 首先也确保最小系统的电源正常, 查看3.3V/1.2V/1.8V, 测试电压精度及电源noise(纹波)。有条件可利用有电流显示的变压箱观察电流状况。确认复位信号正常, power on config正确。
4. 拿到正确的code, 烧录程序。现在我们MID有多种不同的设计, 不同的DDR大小, 不同的flash型号, 不同的IO口配置, 对应的boot也是不一样的, 所以在升级code或从TF卡启动code时首先确保你所拿到的程序是对的。

调试流程:

如果拿到错误的code, 你可能会遇到连接串口从卡启动没有打印消息(与DDR大小不符), 或打印“u boot, jump u”的字符(flash型号尚不支持), 或启动到一个阶段后打印消息停止(IO口的分配与code不一致)。

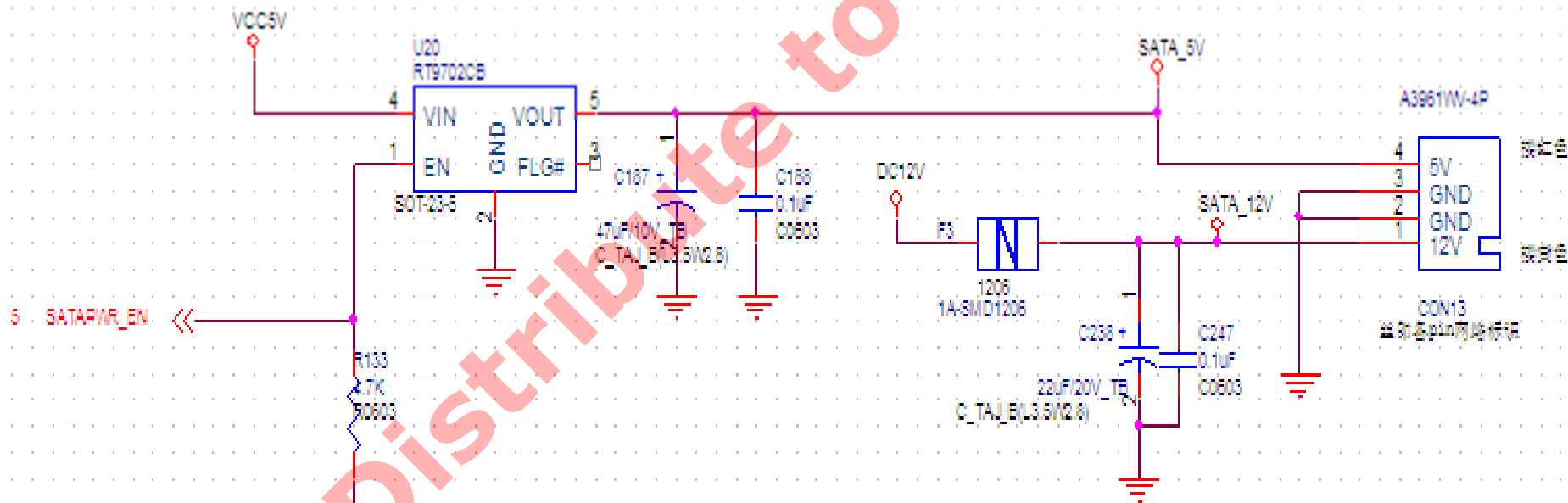
5.有了正确的code后, 升级code, 查看功能。借助串口打印消息, 可查看是否有异常发生。如你可能会发现I2C不断报错, 根据对应I2C地址查看相应设备。

务必check开机过程中电源情况, 尤其是电池供电, 电量偏低情况下, 利用四个探头同时接上电池VBAT, 1.2V/3.3V/1.8V, 观察整个启机过程中, 各电源是否有大的波动, 有些开机时不断重启或死机的现象可能就是因为电源波动所致。

各环节出现的实际问题：

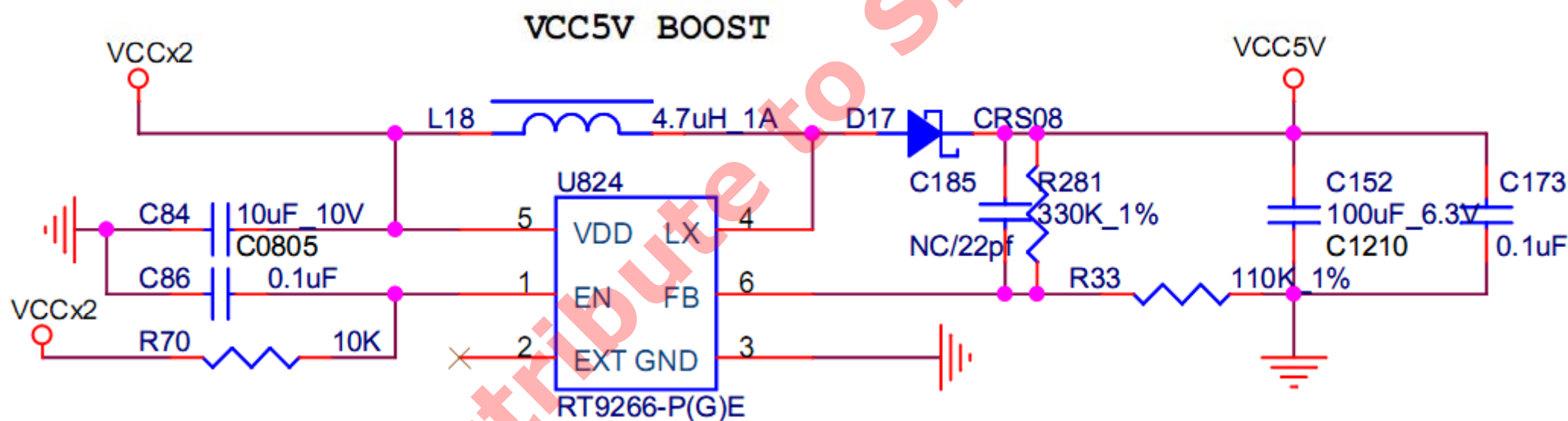
一.原理图设计：

1.原件选型不合适，如下图，U20用作移动硬盘的供电的限流保护，选用了RT9702，这颗料的保护电流偏低(500ma)，插上硬盘时就会发热保护同时引起VCC5V的波动。需改用RT9701(1.1A)。



各环节出现的实际问题:

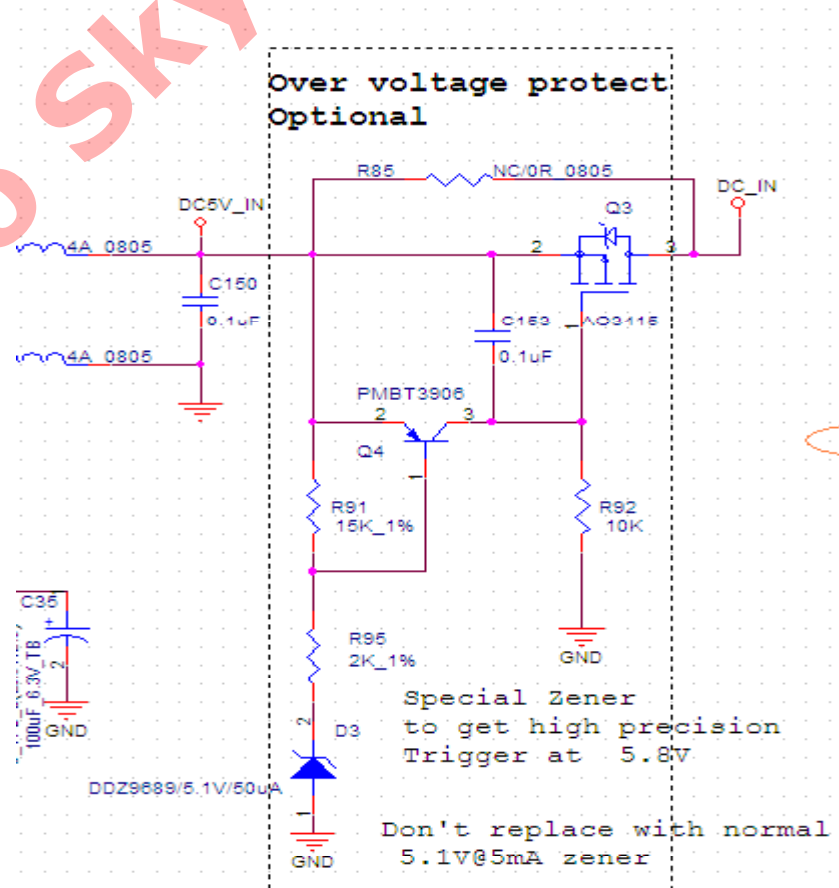
2. 参数选择不合适, 如下图, C152原来为10uF, 此部分给U盘供电, 当插上U盘时, 引起VCC5V的波动, 进而造成VCCX2波动, 最终导致3.3V跌落, 系统复位, 改成100uF, 稳定电源。



各环节出现的实际问题:

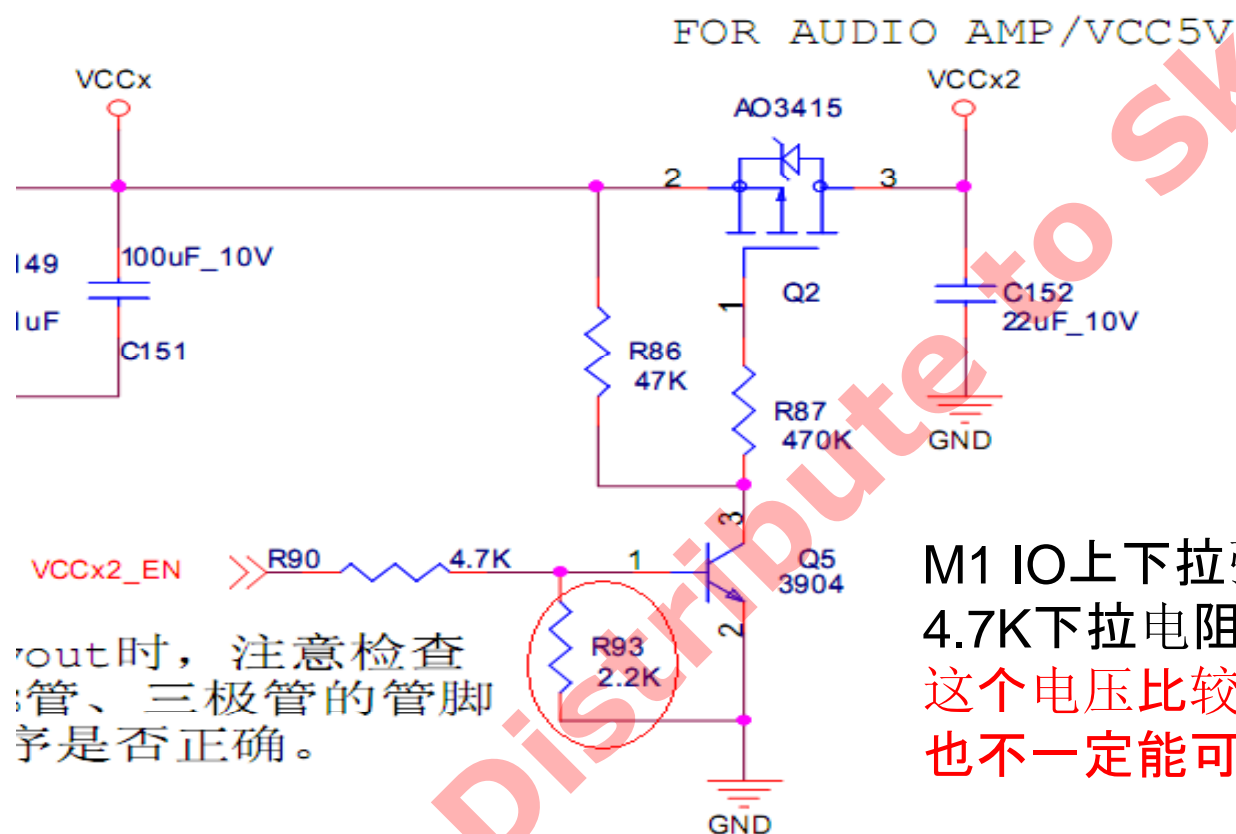
另外如DCDC Fb反馈电阻阻值不合适, 造成电源输出电压不对。Panel VGH/VGL/AVDD部分反馈及稳压管选择不合适。VCOM电压不对, 显示异常。背光电流反馈串阻阻值不合适, 背光偏暗。

如右图，过压保护电路，稳压管选择不合适，导致正常电压就会保护



各环节出现的实际问题:

如下图, R93为什么要用2.2K?



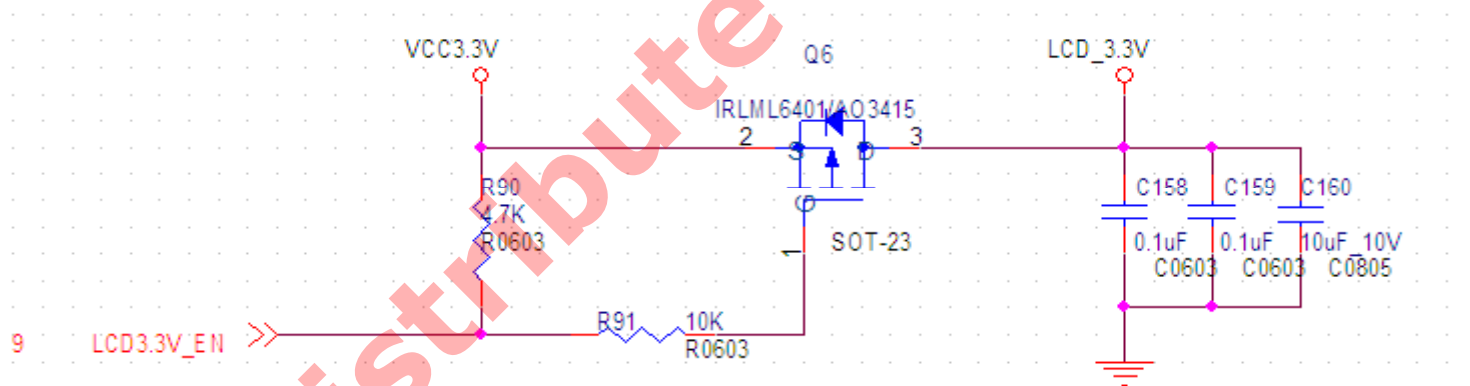
out时, 注意检查
管、三极管的管脚
是否正确。

M1 IO上下拉强度较高, 约82uA,
4.7K下拉电阻会被驱动到0.4V,
这个电压比较高, 不能关闭三极管,
也不一定能可靠的关闭一些设备的EN pin。

各环节出现的实际问题:

3. 电源波动。

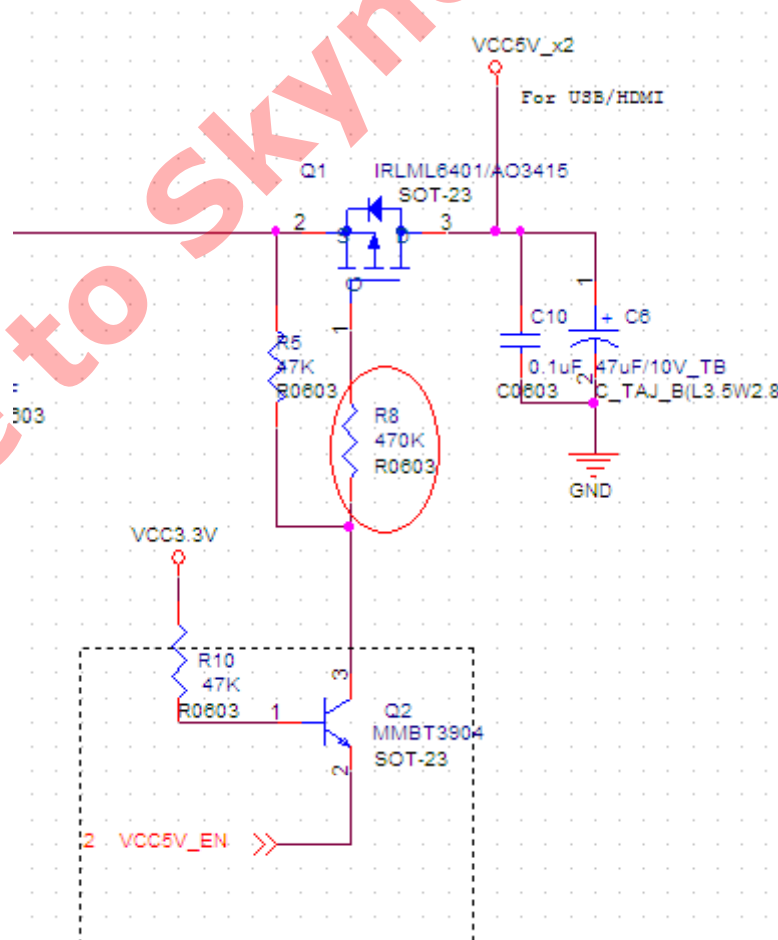
下面这个电路是LCD3.3V的开关控制电路，如果是给大尺寸的屏供电时，要注意查看该处的电流大小，有一款10寸 HSD屏，LCD_3.3V的正常工作电流有200多mA，瞬间电流spec描述有1A，这个屏在LCD_3.3打开的瞬间，3.3V会发生0.7V的跌落，导致电池电压偏低时系统复位。遇到这种问题可通过延缓Q6打开速度的方式减少屏对3.3V电源的冲击。



各环节出现的实际问题:

下面电路的设计,R8用470K这么大的电阻目的是什么?

MID的设计中电源是比较重要的一环。选择DCDC电源时要慎重(1.2V需要选择1.2A规格),根据系统设计功耗选择合适功率的型号,务必实际测试电源瞬间响应,纹波及电压调整率等状况。

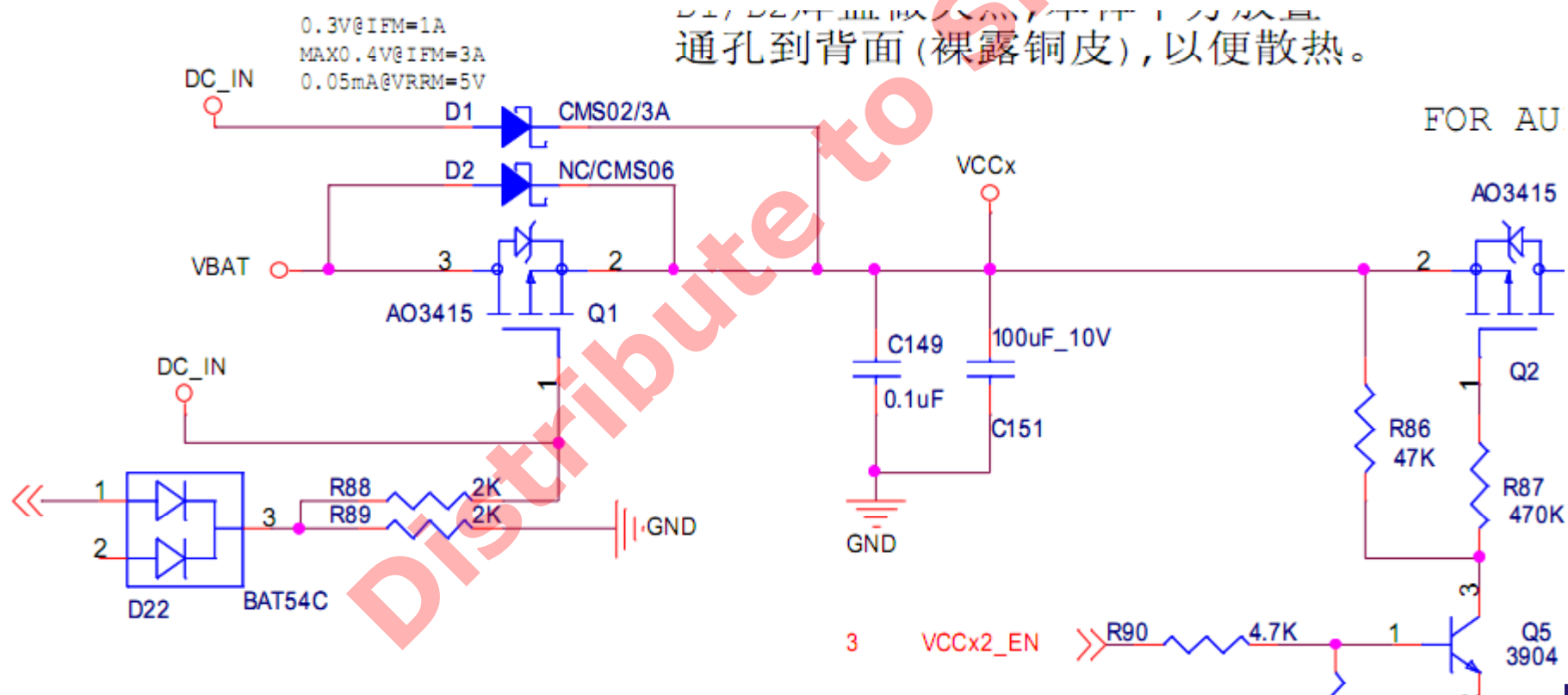


各环节出现的实际问题:

如下图，大尺寸、大功耗的平台上，D2要装上，避免拔掉外部电源时，电池供电下，VCCX掉电。详见插件分析。

Microsoft Word
97-2003 文档

D1的选择要注意什么？



各环节出现的实际问题：

如果系统功耗很大，电池供电时，因为电压偏低，所以电流会更大，此时上面的图可改用双开关管，Q1上再并接一个开关管。

Distribute to Skynoon!

4.加上了多余的器件。



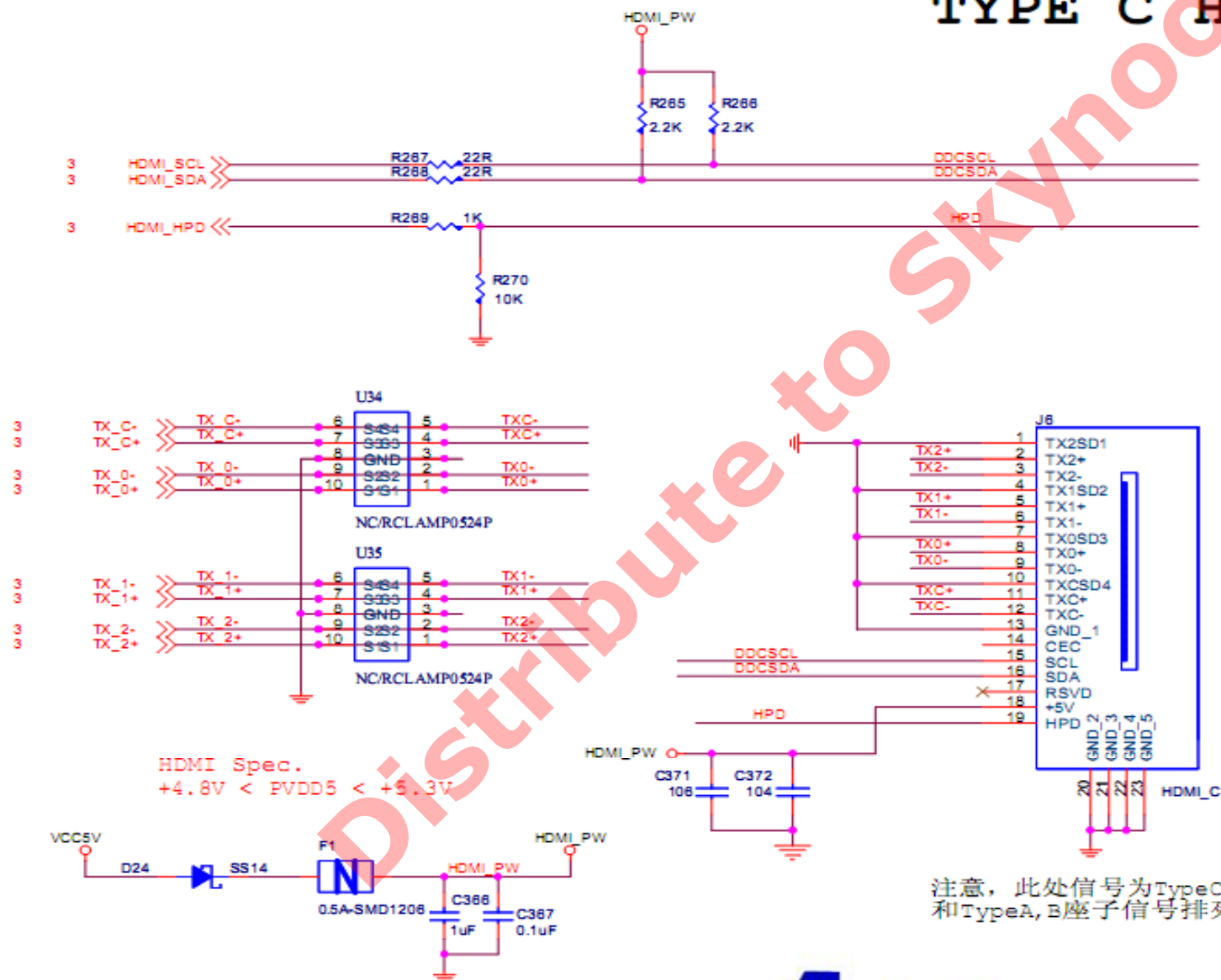
各环节出现的实际问题:

5.MID平台关机下通过HDMI接到电视上,电视有打开,此时再打开MID, G-SENSOR异常(VCC3.3V有2.15V)或个别U盘不读(用万用表测量 HDMI_VCC5V,+5V上有0.9V电压)。

Distribute to Skynote!

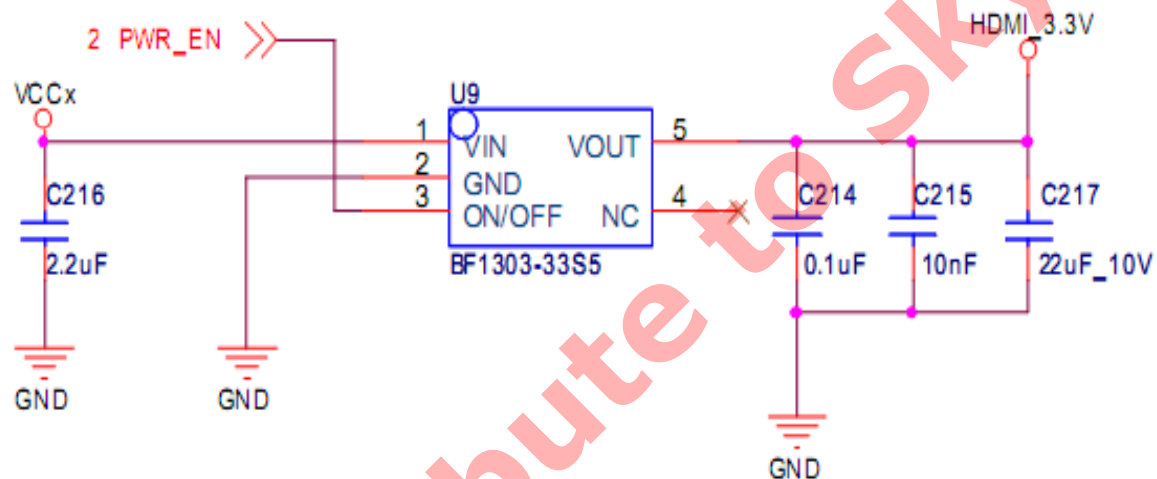
各环节出现的实际问题:

TYPE C HDMI



注意, 此处信号为TypeC排列,
和TypeA, B座子信号排列不同

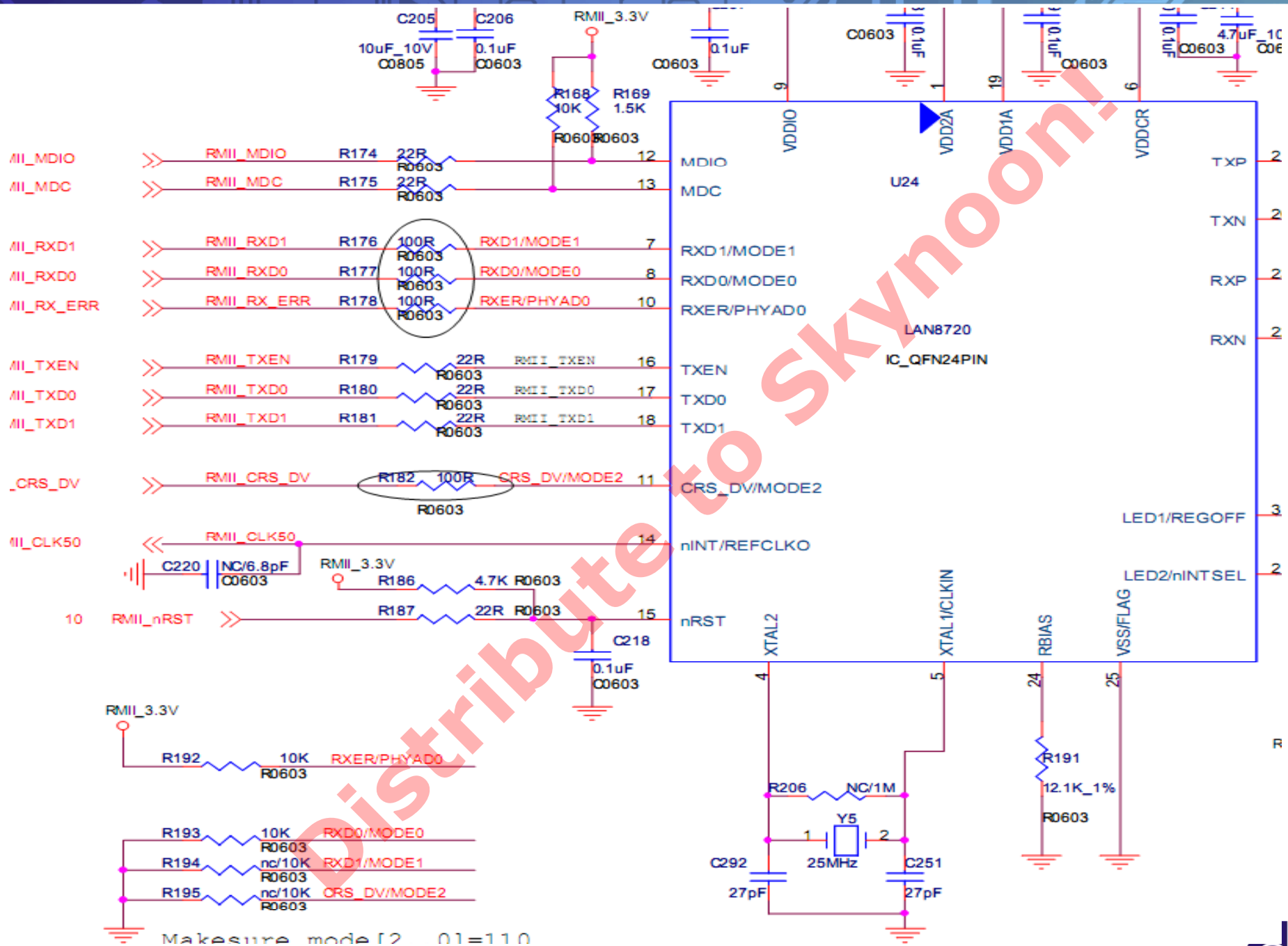
各环节出现的实际问题:



各环节出现的实际问题:

6.Ethernet 工作不稳定,尤其是在1.2V电压偏低的情况下,如下图原因是8720输出的50M CLK较RX信号相位超前,导致不能正常同步信号。RX的串阻改到100R可OK.

Distribute to Skymoon!



各环节出现的实际问题:

二.layout.

1.DDR部分, 主要是四层板。

icrosoft Word
97-2003 文档

2.主芯片电源走线:VBAT/VCCX/1.2V/3.3V TRACE走线不能太窄, why?

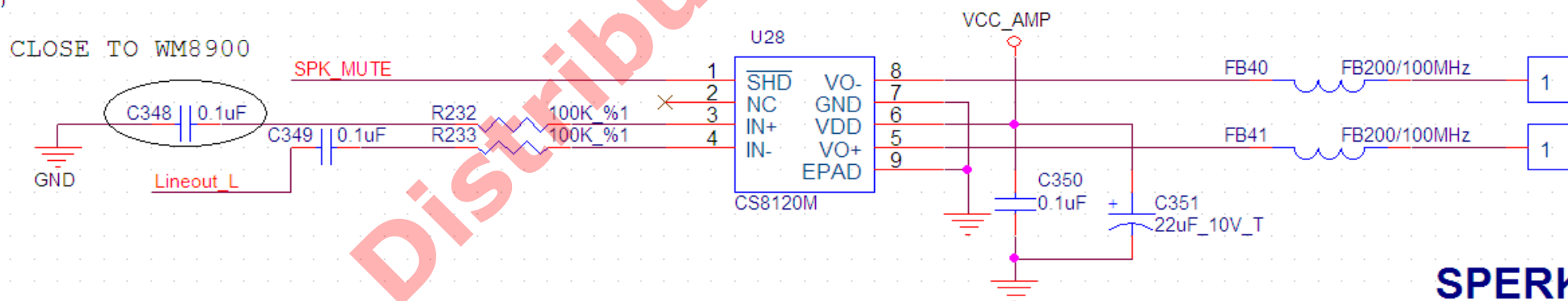
1mm宽10cm长的线, 在1A电流下的压降约67mV, MID上, VBAT, VCCx, 1.2V, 1.8V都是关键信号, 上面的瞬间电流可能达到1A(会在走线上瞬间产生大的跌落), 线宽长度换算后, 要保证每个环节压降<10mV, 并且注意多打过孔连接(电池接口和DC_IN接口的GND也要多打过孔连接)。

如电池走线宽度太窄的话, 其上面的压降偏大, 到后端电压更低, 容易引起复位之类异常。所以至少保证VBAT/VCCX的trace有1.5mm以上宽度, 长度不能太长(若长度偏长, 需要加大宽度)。其他3.3V/1.2/1.8V也一样, 视长度而异, 至少有1.2mm的宽度。

各环节出现的实际问题:

3. 电源DCDC FB走线: 调试时发现有的项目DCDC降压IC的FB走线较长, 引起输出的电压纹波偏大。

4. 模拟地与数字地隔离, 避免被干扰, 如audio部分, 有可能耳机座子或功放因结构的原因距离audio DAC太远, 导致模拟信号的回流路径跨越数字信号区域, 当某功能的数字部分工作时, 引入地噪声, 造成声音输出伴随噪音。



各环节出现的实际问题:

三.器件不良:

- 1.电感不良。现象：BUCK DCDC 电感发热，电源纹波偏大，负载较重时，电源输出产生DROP。经查是DCDC 降压所用电感不良，实测感值不够。
- 2.钽电容不良。现象：上电前测试所有电源输出均没有短路，但是接入电源后，用有电流显示的调压箱观察，发现有900多mA的电流，此系有一路DCDC后面的滤波钽电容不良，耐压值不够，更换后OK.

各环节出现的实际问题:

四.制程作业方面:

1.缺件:

复位IC 供电磁珠没有装上。现象:电源都正常,但系统就是不能工作,装上此bead后即OK.

Wifi MODULE 4329 SDIO信号上拉电阻没有装,导致不能正常工作。

主芯片test 引脚需要加上拉,漏装的话只能从SD卡启动,不能从nand flash正常启动。

2.装错件:

充电 IC AAT3620充电电流设置pin的电阻阻值装错,原本应该1A大小的充电电流变成了1.8A.

1.2V输出反馈的分压阻值装错,导致输出电压不准。

3.短路、虚焊。

4.给panel AVDD供电网络上的磁珠错装成1K电阻。VGH/VGL稳压管装上错误稳压值的型号。

5.电阻装成电容或电容装成电阻。

6.电阻、电容, DDR等被打碎。

谢谢!

Q



A

Distribute to Skynoon!