全志科技
Allwinner Technology

H3 项目

Android 定制化文档 V1.1

文档履历

版本号	日期	制/修订人	内容描述
V1.0	2015-1-13		正式版本
V1.1	2015-5-27		1.固件 pack 方式更新说明 2.增加切台保持最后一帧定制一节 3.自动保存 log 方式更新 4.文档中涉及 sdk 文件路径调整更新

目 录

1. 引言

1.1. 编写目的

本文档介绍 H3 方案中常见的定制问题，以帮助客户快速熟悉 sdk，加快产品上市。

1.2. 定义

homlet：面向客厅设备的产品线的名称。
dolphin：基于全志 H3 芯片的家庭娱乐产品代号。

2. SDK 概述

2.1. 如何建立开发环境

本节将介绍 H3 平台开发环境所需的软硬件资源及的搭建。

其中，开发所需要的软件环境和工具基本跟 android 原生的 ASOP 环境搭建是一样的，用户也可以同时参考 android 原生的 ASOP 开发搭建环境方法。

2.2. 硬件资源

- H3 主控 box 方案板 + 电源适配器；
- 串口线，hdmi/cvbs 线，以太网线，USB 线一条(根据具体的接口需求)等；
- PC：编译或者烧录开发用（Linux 系统），也可以安装虚拟机运行 XP 进行固件烧录(可选)；

2.2.1. 软件资源

Linux 主机（推荐使用 ubuntu12.04 64bit），硬盘空间至少 100G（可满足一次完全编译），一般来说 Linux 主机中需要：

-Python 的 2.7.3 版本；

-GNU Make 的 3.81-3.82 版本；

-JDK 6；

-git 的 1.7 或更高版本；

可选，在 ubuntu 安装虚拟机运行 xp，或者单独的 Windows XP 主机，作为固件烧写机器和本地调试环境，通常需要安装下面软件：

1 PhoenixSuit 一键烧写工具(分 linux 版本，windows 版本和苹果 mac 版本)；

2 USB 转串口驱动；

3 Android SDK；

下面以 ubuntu12.04 和 XP 为例，安装软件环境。

2.2.2. 安装 JDK（ubuntu12.04）

JDK 安装命令

```
$ sudo add-apt-repository "deb http://archive.canonical.com/ lucid partner"
$ sudo apt-get update
$ sudo apt-get install sun-java6-jdk
```

2.2.3. 安装平台支持软件（ubuntu12.04）

```
$ sudo apt-get install git gnupg flex bison gperf build-essential \
zip curl libc6-dev libncurses5-dev:i386 x11proto-core-dev \
    libx11-dev:i386 libreadline6-dev:i386 libgl1-mesa-glx:i386 \
```

```
libgl1-mesa-dev g++-multilib mingw32 tofrodos \
python-markdown libxml2-utils xsltproc zlib1g-dev:i386
$ sudo ln -s /usr/lib/i386-linux-gnu/mesa/libGL.so.1 /usr/lib/i386-linux-gnu/libGL.so
```

2.2.4. 安装编译工具链（ubuntu12.04）

编译工具链已经集成在 Android SDK 中，工具链位于 Android SDK 中的 lichee/brandy/gcc-linaro/中。

2.2.5. 安装 phoenixSuit（windows xp）

PhoenixSuit 包括三个版本，linux 版本、windosw 版本和苹果 mac 版本，位于 lichee/tools/tools_win 中，将 PhoenixSuitPacket.msi 复制到 XP 主机上，按照安装向导提示安装，即可完成 phoenixSuit 的安装。

2.2.6. 其他软件（windows xp）

建议在 windows 系统下安装 putty，并且网络映射到上述 Linux 编译服务器进行 SDK 源码的编译。在开发过程中缺少相关驱动也优先从 SDK 中查找。

2.3. 代码下载说明

请参考 SDK 发布文档的下载说明，须向全志申请下载 sdk 的权限和账号。

2.4. homlet 公版代码编译

代码编译、固件打包和烧写详细说明请参考文档《H3 lichee 使用手册》。

这里简要说明一下，

(1) 编译 lichee 代码

```
$cd lichee
$./build.sh lunch
All available lichee lunch:
    0. sun8iw6p1-android-eagle
    1. sun8iw6p1-android-secure
    2. sun8iw7p1-android-dolphin
    3. sun8iw7p1-android-secure
    4. sun8iw7p1-android-karaok
    5. sun8iw8p1-android
    6. sun9iw1p1-android-jaws
    7. sun9iw1p1-android-secure
    8. sun9iw1p1-android-optimus
Choice: 2 (选择需要编译的方案)
```

编译成功会打印

```
INFO: build kernel OK.
INFO: build rootfs ...
INFO: skip make rootfs for android
INFO: build rootfs OK.
----------------------------------------
build sun8iw7p1 android    lichee OK
----------------------------------------
```

(2) 编译 android 代码

```
$cd android
$ source ./build/envsetup.sh
$lunch    dolphin_fvd_p1-eng     #选择方案号
$extract-bsp                     #拷贝内核及驱动模块
$make –j8                        #后面的数值为同时编译的进程，依赖于主机的配置
$pack                            #打包生成固件
```

编译，打包成功会输出固件的地址，如：

```
normal
dragon image.cfg sys_partition.fex [OK]
----------image is at----------
/home/yourname/workspace/lichee/tools/pack/sun8iw7p1_android_dolphin-p1.img
pack finish
```

2.5. dolphin 新增方案定制须知

SDK 代码分为 android、lichee 两个目录，lichee 部分为 bootloader、内核、量产打包的代码。定制化及移植工作主要涉及到的目录：

```
lichee/brandy
lichee/linux-3.4
lichee/tools/pack/chips/sun8iw7p1/configs/
android/device/softwinner
```

2.5.1. lichee/linux-3.4

kernel 部分一般无需配置，但如果有需求要增加新的驱动、打开内核某些 Features 或者 更改预留内存时，可以对其进行更新，默认的 Android 内核配置文件是：

```
lichee/linux-3.4/arch/arm/configs/sun8iw7p1smp_android_dolphin_defconfig
```

注意：第一次编译内核时默认会复制 linux-3.4/arch/arm/configs/sun8iw7p1smp_android_dolphin_deconfig 到 linux-3.4/.config 作为编译的配置；如果 linux-3.4/.config 存在的话，并且编译同样的方案则不覆盖。

2.5.2. lichee/tools/pack/chips/sun8iw7p1/configs/

此目录下为各个方案的硬件板级配置文件及开机 logo，定制移植中，可复制公版（dolphin-p1）作为方案配置，根据方案的原理图和各个模块的实际使用情况进行修改。

2.5.3. android/device/softwinner

此目录下为各个方案的 Android 软件配置目录，定制移植中，也可复制一份公版配置作为方案配置，根据实际的定制化需求进行修改。

3. 添加定制的方案板配置

3.1. 添加定制的方案 lichee 配置

拷贝一份通用的配置，如 lichee/tools/pack/chips/sun8iw7p1/configs/dolphin-p1 为 lichee/tools/pack/chips/sun8iw7p1/configs/dolphin-xxx，然后按照实际的硬件电路进行配置修改，配置的方法见《H3_sys_config.fex 配置说明.pdf》和《H3_sys_partition.fex 分区表说明.pdf》。

通常对于盒子产品，定制化配置主要集中在“存储介质分区”、“遥控器地址键码”和“wifi 和蓝牙”等功能上。

以 dolphin-p1 为例，在 lichee\tools\pack\chips\sun8iw7p1\configs\dolphin-p1 中定义了各个方案的硬件参数配置，每个方案都由两个文件：sys_config.fex, sys_partition.fex, test_config.fex, env.cfg 和 bootlogo.bmp 来定义。其中 sys_config.fex, sys_partition.fex 为 android 固件使用，test_config.fex 为板卡测试工具 dragonboard 使用(复用 android 的 sys_config.fex). 对于每个模块中譬如 XXX_used 这个参数模块是表示该模块是否用到，当设置为 0(不可用)时其他参数可以不用配置，如模块[ps2_0_para]中的 ps2_used 设置为 0 时，ps2_scl 和 ps2_sda 可以不用配置。

注意：如果 env.cfg 未在方案目录下进行配置，将默认使用配置 lichee/tools/pack/chips/sun8iw7p1/default/env.cfg

3.1.1. 分区配置说明

以 dolphin-p1 为例，盒子系统中常用的分区大小和作用如下：

分区名	大小	用途
bootloader	16M	Bootloader 资源
env	16M	系统启动环境变量
boot	16M	Android Boot 分区，存放内核，根文件系统等
system	768M	Android System 分区，存放系统服务、应用等
recovery	32M	Android Recovery 分区，用于 Android Recovery 系统
misc	16M	Misc 分区，用于写入 BCB（Bootloader Cmd Block）进入 recovery
private	16M	存放厂商序列号等私有数据（私有分区）
sysrecovery	768M	固件备份分区，用于一键恢复功能，默认关闭。
cache	512M	Android Cache 分区，用于 Recovery 系统存放 OTA 固件等
UDISK	剩余大小	作为 Android 的 data 分区
Reserve0	16M	预留分区
Reserve1	32M	预留分区
Reserve2	16M	预留分区
klog	16M	内核 oops 时将 kernel 的 logbuf 打印到此分区

3.1.2. 添加新的分区

开发中添加的分区分为两种，一种为普通分区，另一种为私有分区。私有分区的数据在重新擦除量产升级后数据不会丢失，可以用于存放序列号等数据，公版默认有一个 private 分区，厂商可使用此分区来存放私有数据，如果实际使用不够，还可以继续添加。添加分区的方法参考**《H3 sys_partition.fex 分区表说明》。**

3.2. 添加定制的方案板 Android 配置

3.2.1. 修改方案资源

3.2.1.1. 修改 bootlogo

替换 lichee/tools/pack/chips/sun8iw7p1/configs/dolphin-p1/ bootlogo.bmp 文件。

图片要求：

(1) Bootlogo 格式必须是 32 位的 bmp 格式的图片。

(2) Bootlogo 图片的只支持分辨率为 1280*720 的 bmp 图片。

3.2.1.2. 修改存储自检 logo

存储自检 logo 是盒子启动过程中，如果内部存储出现故障，会在屏幕上显示的 logo 图片，可以提醒用户不要断电等待修复完成。

这个文件可以使用 bmp 图片用 lichee/tools/tools_win/LogoGen 工具生成。

bmp 图片要求：

(1) Bootlogo 格式必须是 32 位的 bmp 格式的图片。

(2) Bootlogo 图片支持 1280*720。

替换 android/device/softwinner/dolphin-xxx/ needfix.rle 文件，替换后如果需要测试效果，可在 adb shell 环境下使用如下命令：

```
#set_ext4_err_bit /dev/block/by-name/cache
#reboot
```

重启后可以看见自检界面。

3.2.1.3. 修改开机动画

修改开机动画可以替换掉 android/device/softwinner/dolphin-xxx/media/bootanimation.zip 文件，开机动画的制作方法如下：

1.准备 part0、part1 的逐帧图片资源（PNG），通常 part0 只播放一次，part1 循环播放至开机

2.准备 desc.txt，此文件的内容示例如下

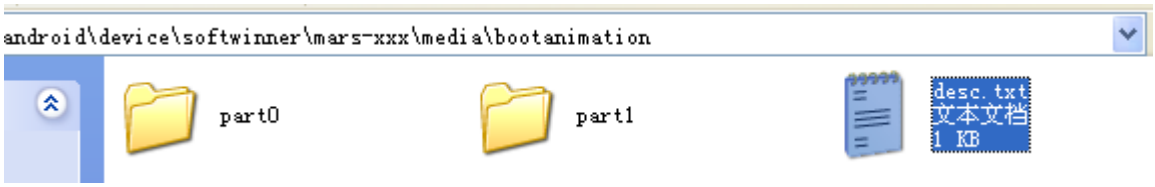
```
400 409 16
p 1 0 part0
p 0 0 part1
```

400 409 16 ---这里的 400 代表图片的像素（大小）宽度，409 代表图片的像素（大小）高度，16 代表帧数；

p 1 0 part0 ---这里的 p 代表标志符，1 代表循环次数为 1 次，0 代表阶段间隔时间为 0，part0 代表对应的文件夹名，为第一阶段动画图片目录；

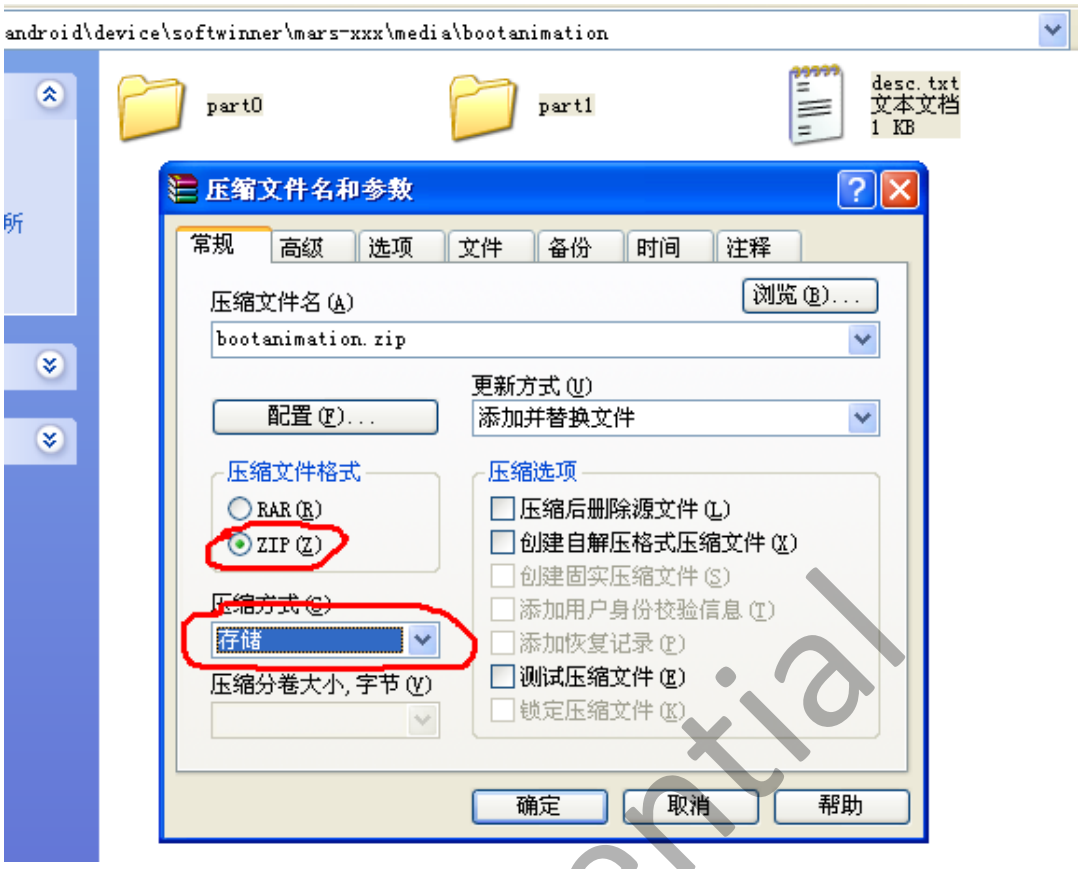
p 0 0 part1---这里的 p 代表标志符，0 代表本阶段无限循环，0 代表阶段间隔时间为 0，part1 代表对应的文件夹名，为第二阶段动画图片目录；

准备好的资源如下：



全部选中后使用 winrar 压缩

使用 zip 压缩，存储选项



点击确认生成 bootanimation.zip 动画。

3.2.2. 方案目录内文件说明

在 android\device\softwinner\目录下包含各个方案目录和提供给具体方案使用的公共配置目录，在这里可以添加自己的方案目录 xxx，软件上的配置全部放在这目录下，每个方案目录下的文件名字和结构基本相同，下面以 dolphin-fvd-p1 方案为例说明方案目录下重要文件的作用。

3.2.2.1. dolphin_fvd_p1.mk

文件内部定义了需要定制的信息，如需要预装的 apk，需要编译的 apk 源码、产品名字等等。应该把这个文件名改为自己的方案名，如：dolphin_fvd_p1.mk，文件中有几个比较重要变量的值可能要改的，意义如下：

(1) PRODUCT_PACKAGES

这里定义了需要添加的产品包或库，添加上去后，会编译该源码，打包之后固件里就会有该 apk 或库文件，需要添加生成的 apk 或.so 文件时，应该把它的.mk 文件中定义的 PACKAGENAME 的值加上去。

(2) PRODUCT_COPY_FILES

编译时把该环境变量中定的东西拷贝到指定的路径，如在 dolphin-fvd-p1 方案目录下把 init.rc 拷贝到根目录下：

```
PRODUCT_COPY_FILES += \
device/softwinner/dolphin-fvd-p1/init.rc:root/init.rc \
```

(3) PRODUCT_PROPERTY_OVERRIDES(定义 Property 环境参数)

此命令用于向 android 系统中添加系统属性，这些属性在编译时会被收集，最终放到系统的 system/build.prop 文件中，开机时被加载，可以被系统读取到并进行相应的设置。

比如下面属性定义了固件的默认时区、国家、语言。

```
PRODUCT_PROPERTY_OVERRIDES += \
        persist.sys.timezone=Asia/Shanghai \
        persist.sys.language=zh \
        persist.sys.country=CN
```

(4) PACK_BOARD

pack 命令是打包最后可用于 usb 烧录或者制作量产卡的固件，打包时会匹配板型，通过 dolphin-p1.mk 里面的编译变量 PACK_BOARD 来确定板型：

```
#pack parameter
PACK_BOARD := dolphin-p1
```

这个会和 lichee/tools/pack/chips/sun8iw7p1/configs/{$PACK_BOARD}/的配置项对应

pack： 打包产生固件

pack –d： 打包产生串口信息从 SD/TF 卡卡槽输出的固件。

3.2.2.2. AndroidProducts.mk

这里只有一句话：

```
PRODUCT_MAKEFILES := \
    $(LOCAL_DIR)/dolphin_fvd_p1.mk
```

其中 dolphin_fvd_p1.mk 就是上个小节中说的文件，所有也要改成自己方案的该文件名。

3.2.2.3. BoardConfig.mk

这里一般定义了 Wifi 和其他一些的配置变量，wifi 的配置可参考文档《H3 wifi 模块移植说明文档》。

同理，该文件下的和方案名有关的文字都要改成自己方案的。

3.2.2.4. init.rc

该脚本会在 android 系统启动时被调用，功能是做与方案有关的驱动加载及服务初始化，客户定制化的一些服务，也可以在 init.rc 中加载。

3.2.2.5. needfix.rle

开机文件系统自检修复界面的 logo 图片，可以使用 lichee\tools\tools_win\LogoGen.zip 小工具生成。

3.2.2.6. recovery.fstab

该文件中有定义 recovery 阶段各个分区或设备对应的挂载点，一般默认使用公版的分区形式。

3.2.2.7. vendorsetup.sh

其内容通常为：

```
add_lunch_combo dolphin_xxx-eng
```

在编译时“lunch”后就会看到 dolphin_xxx-eng 这个选项，如果将其修改为

```
add_lunch_combo dolphin_xxx-user
```

编译时“lunch”会看到 dolphin_xxx-user 这个选项，两者的区别在于 eng 版本用于工程开发，默认 adb 打开并开放 root 权限，而 user 版本默认不打开 adb 并且不开放 root 权限。

发布版本固件时通常以 user 模式编译，这样可以提高系统的安全性。

3.3. 如何添加新的 product

此处，我们将以添加一个新的 product（命名为 dolphin-xxx）为例来说明。

在 android/device/softwinner 目录下创建新的 procuct 目录：

1）clone 一个新的 dolphin-fvd-p1 仓库到本地。之所以需要一个新的仓库，是因为在编译过程中此目录下面会生成一些临时文件。

2）将此 dolphin-fvd-p1 文件夹整体复制，并重新命名为“dolphin-xxx”。

3）删除此目录下的“.git”文件夹。

4）利用某些工具（比如 UltraEdit 或者 shell 命令），将此目录下的所有文件内的“dolphin-fvd-p1”替换为“dolphin-xxx”,把所有文件内的“dolphin_fvd_p1”改为“dolphin_xxx”。

5）将“dolphin_fvd_p1.mk”重命名为“dolphin_xxxx.mk”

例如使用 shell 命令：

```
$ cd android/device/softwinner/
$ cp -r dolphin-fvd-p1 dolphin-xxx
$ cd dolphin-xxx/
$ rm -rf .git modules kernel                          #删除原有的 git modules 目录和内核
$ mv dolphin_fvd_p1.mk    dolphin-xxx.mk       #重命名 makefile 文件
$ find . -type f | xargs sed -i 's/dolphin_fvd_p1/dolphin_xxx/g'   #替换掉 dolphin_fvd_p1 的字符串
$ find . -type f | xargs sed -i 's/dolphin-fvd-p1/dolphin-xxx/g'   #替换掉 dolphin-fvd-p1 的字符串
$ git init                                                #重新初始化 git 管理
$ git add * -f
$ git commit -m "init first versioion for dolphin-xxx"                  #第一次提交
```

6) 修改 package.sh

把 board=dolphin-p1 修改为 2.3.2 创建的对应的方案配置目录名称。

7) 操作完成后，在 android 根目录下执行以下命令，便可加载方案

```
$source ./build/envsetup.sh
$lunch dolphin_xxx-eng
```

3.3.1. 创建此 product 对应的配置文件

1）在目录 lichee/tools/pack/chips/sun8iw7p1/configs/下，复制文件夹“dolphin-p1”。

2）将复制后的文件夹重命名为“dolphin-xxx”。

3）根据 product 的具体情况，修改“dolphin-xxx”目录下的 sys_config.fex 和 sys_partition.fex 文件内的配置信息 。

3.3.2. 验证新 product 的正确性

在验证之前，务必将 dolphin-xxx 目录整体备份一下，因为验证时会在此目录下产生一些临时文件。 验证方法就是整体编译，打固件烧写，看方案机器是否可以正常跑起来。

3.3.3. 创建此 product 对应的 git 仓库

如果验证无误，将备份的 dolphin-xxx 目录恢复出来。

1）本地创建 git 仓库。假设分支名为“dolphin-dev”，在 dolphin-xxx 目录下

```
$git init
$git add .
$git commit -m "create a new product ‘dolphin-xxx'."
$git branch dolphin-dev       ;// dolphin-dev 是主分支名称，请根据自己的实际情况决定
$cd ..
$git clone --bare dolphin-xxx dolphin-xxx.git
```

2）将生成的 dolphin-xxx.git 推送至服务器上；

3）删除本地的 dolphin-xxx 和 dolphin-xxx.git 文件夹。

3.3.4. 将此仓库添加至 repo 中

1）在文件 android\.repo\manifests\homlet-44-android.xml 内，根据创建的仓库名称添加如下信息：

<project path="device/softwinner/ dolphin-xxx" name="device/softwinner/dolphin-xxx" />

2) 将修改提交至服务器。

3.3.5. 将 tools 下针对此 product 新添加的配置文件提交至服务器

目录 lichee/tools/pack/chips/sun8iw7p1/configs/dolphin-xx/

4. OTA 升级说明

4.1. OTA 的升级范围

原生 Android 提供的 Recovery 升级程序只支持更新 system 分区、recovery 分区及 boot 分区。除此之外，我们根据产品特点，给 Recovery 扩展了一些专有功能，以满足 BSP 的更新需要。

分区类型	是否支持
Boot 分区更新	√
System 分区更新	√
Recovery 分区更新	√
Env 分区更新	√
Bootloader 分区更新	√
Nand 方案 Boot0/Uboot 升级	√
TSD/EMMC 方案 Boot0/Uboot 升级	√
sys_config.fex 更新	√
sys_partition.fex 更新	×

值得注意的是，BSP 中的大部分关于模块的配置都集中在 sys_config.fex 中，如果需要更新 sys_config.fex 的配置，就必须通过更新 uboot。

在制作更新包时，要想新的 sys_config.fex 生效，务必记得在执行 make 命令之前先执行 get_uboot 确保当前的 sys_config.fex 配置被打到 uboot 中。

4.2. 升级注意事项

4.2.1. OTA 不能改变分区数目及其大小

Recovery 只是一个运行在 Linux 上的一个普通应用程序，它并没有能力对现有分区表进行调整，所以第一次量产时就要将分区的数目和大小确定清楚，杜绝后续升级调整分区数目及其大小的想法，OTA 不能改变分区数目和分区的大小。

4.2.2. cache 分区的大小确定

原生 Recovery 机制中，因为 Recovery 内的分区挂载路径与 Android 的分区挂载路径并不完全相同，所以在 Android 上层传入更新包地址时，必须要保证这个包路径在 Recovery 和 Android 系统都是相同的。

能够读写的分区中只有 cache 分区和 data 分区会被 Recovery 和 Android 系统同时挂载，这意味着需要将包放这两个分区中，Recovery 才能识别。所以 Google 原生策略中，当在外部储存选择一个升级包时，都默认复制到 cache 分区中。所以在划分分区时需要注意要分配 cache 分区足够大的空间，否则可能出现无法容纳更新包而导致无法升级的问题。

4.2.3. misc 分区需要有足够的权限被读写

misc 分区 Recovery 与 Android 之间的桥梁，如果 misc 分区的读写权限过高，会导致上层应用无法对其写入数据，则会令 Recovery 功能异常。检验此功能存在问题时，请确保 misc 分区的设备节点/dev/block/xxx 和其软链接/dev/block/misc 有足够的权限被读写。比如，dolphin-fvd-p1 方案的 nandg 的 group 权限为 system。

```
root@android:/dev/block /by-name# ls -l
lrwxrwxrwx root          root      2000-01-02 07:16 misc -> /dev/block/mmcblk0p8(misc 分区软链接)
...
root@android:/dev/block # ls -l
brw-rw---- root          system        93,    48 2000-01-02 07:16 mmcblk0p8
.....
```

4.3. 制作 OTA 包

使用 OTA 区分三个包：

TargetFile: 包含制作时当前编译版本的 system 分区，boot 分区，recovery 分区等内容，可用于制作 OTA 完整包和差分包。

OTA 完整包：包含本次升级版本的所有内容，可以从之前各个版本直接升级到当前的版本。制作完整包需要当前版本的 TargetFile。

OTA 差分包：包含本次升级版本和之前特定一个版本的升级内容，只适用于之前特定一个版本升级到当前版本。制作差分包需要之前特定版本的 TargetFile 和当前版本的 TargetFile。

4.3.1. 制作不带签名的 OTA 包的步骤

1.制作不带签名的 TargetFile

在编译完 Android 后，输入以下命令：

```
$make_ota_target_file
```

或者

```
$get_uboot
$make target-files-package
```

make_ota_target_file 只是封装一些命令，具体可以查看 device/softwinner/common/vendorsetup.sh 里面关于 make_ota_target_file 的定义。

get_uboot 命令作用是从 lichee 目录中复制必要的更新文件到更新包中。不执行该命令会导致后面的 make 动作报错。

最后得到 TargetFile 的路径(device 表示编译的方案，time 表示当天的日期)：

out/target/product/[device1]/obj/PACKAGING/target_files_intermediates/[device2]-target_files-[time].zip

(其中 device1 = $tdevice, device2 = $TARGET_PRODUCT，可以 echo $tdevice 可以看 device1, echo $TARGET_PRODUCT 可看 device2)

2.制作不带签名的 OTA 完整包

```
$get_uboot
```

```
$make otapackage -j8
```

或者

```
$make_ota_package
```

make_ota_package 只是封装一些命令，具体可以查看 device/softwinner/common/vendorsetup.sh 里面关于 make_ota_package 的定义。

get_uboot 命令作用是从 lichee 目录中复制必要的更新文件到更新包中。不执行该命令会导致后面的 make 动作报错。

得到的 OTA 完整包的路径(device 表示编译的方案，time 表示当天的日期)：

out/target/product/[device1]/[device2]-ota-[time].zip

(其中 device1 = $tdevice, device2 = $TARGET_PRODUCT，可以 echo $tdevice 可以看 device1, echo $TARGET_PRODUCT 可看 device2)

3.制作不带签名的 OTA 差分包

要生成差分包，必须获得前一版本的 target-file 文件，具体参考前面如何制作 TargetFile。

将要升级版本(旧版)的 target-file 文件拷贝到 Android 的根目录下，并重命名为 old_target_files.zip。保证 Android 的根目录下只有一个*.zip 的文件。之后执行以下命名将可以生成差分包：

```
$make otapackage_inc
```

得到的 OTA 差分包的路径(device 表示编译的方案，time 表示当天的日期)

out/target/product/[device1]/[device2]-ota-[time]-inc.zip

(其中 device1 = $tdevice, device2 = $TARGET_PRODUCT，可以 echo $tdevice 可以看 device1, echo $TARGET_PRODUCT 可看 device2)

注意：

1. 该差分包仅对指定的前一版本固件有效。
2. 制作一个完整包，也会生成当前版本的一个 target-file 文件包。

4.3.2. 制作带签名的 OTA 升级包

签名可以让 Android 带有厂家私有的签名，在 OTA 升级的时候会预先校验签名，如果签名不匹配则无法进行 OTA 升级，签名校验保证了 OTA 包的来源合法性，OTA 包的完成性（没有被第三方修改过）。制作带签名的 OTA 升级包的流程如下：

1.生成没有签名的 TargetFile

参考制作没有待签名的 OTA 包生成 TargetFile 的方法。

2.TargetFile 签名

```
$./build/tools/releasetools/sign_target_files_apks -d [key_path] -o [unsigned_target_file.zip] [signed_target_file.zip]
```

[key_path]为存放 key 文件夹的路径，需要包括 4 个 key 分别是 media， platform，releasekey，shared，具体包含以下文件：media.pem，media.x509.pem，platform.pk8，releasekey.pem，releasekey.x509.pem，shared.pk8，media.pk8，platform.pem，platform.x509.pem，releasekey.pk8，shared.pem ，shared.x509.pem

-o 表示替换一个公用的 ota_key，这个 key 确保第三方的 ota 包没有办法升级

[unsigned_target_file.zip]表示上一步生成的没有签名的 TargetFile
[signed_target_file.zip]表示命令输出得到的经过签名的 TargetFile
签名的时候需要按照提示输入相应的证书的密码。

3.从签名过的 TargetFile 得到镜像（boot.img,system.img 和 recovery.img)

```
$ ./build/tools/releasetools/img_from_target_files [signed_target_file.zip]  [img.zip]
```

[signed_target_file.zip]表示经过签名的 TargetFile
[img.zip]表示命令输出得到的镜像压缩包

4.解压 img.zip,得到的 boot.img,system.img 和 recovery.img 复制到 target/product/[device1]/下面，重新 pack 得到可烧录的固件，是签名过的固件。

5.生成 ota 包完整包

```
$./build/tools/releasetools/ota_from_target_files -k [releaseKey] [signed_target_file.zip] [ota_full.zip]
```

[releaseKey] 和 sign_target_files_apks 用来签名的 release key 相对应
[signed_target_file.zip]表示经过签名的 TargetFile
[ota_full.zip]表示命令输出得到的 OTA 完整包

6.生成 ota 包完整包

```
$./build/tools/releasetools/ota_from_target_files -k [releaseKey] -i [signed_target_file_v1.zip] [signed_target_file_v2.zip] [ota_inc.zip]
```

[releaseKey] 和 sign_target_files_apks 用来签名的 release key 相对应
[signed_target_file_v1.zip]表示经过签名的版本 v1 的 TargetFile
[signed_target_file_v2.zip]表示经过签名的版本 v2 的 TargetFile
[ota_inc.zip]表示命令输出得到的 OTA 完整包

4.3.3. 制作 OTA 包常见问题和注意事项

1.执行签名命令的时候./build/tools/releasetools/sign_target_files_apks 出现类似的提示：
ERROR: no key specified for:
xxxx.apk
Use '-e <apkname>=' to specify a key (which may be an empty string to not sign this apk).
这种情况通常是定义这个 apk 的 Android.mk 文件编写不规范导致，解决方法修改这个 apk 的 Android.mk 文件，或者按照提示使用-e 参数，
例如-e xxx.apk= 表示不对这个 apk 重新签名
或者-e xxx.apk=[pathToKey]/[key]，[pathToKey]是存放 key 的文件夹，[key]根据实际情况选择 media，platform，releasekey 和 shared 其中一个

2.如何生成 key?
android/build/tools/mkkey.sh 脚本，打开脚本看到下面的一句定义:
AUTH='/C=CN/ST=GuangDong/L=ZhuHai/O=Allwinner/OU=Allwinner/CN=China/emailAddress=allwinnertech@com'
请按照实际需求修改，然后执行
./mkkey.sh media

按照提示输入这个 key 的密码
然后就会在此目录下生成:media.pem　media.pk8　media.x509.pem 三个文件。

3.TargetFile 和固件是否匹配的区分
在 Android 设备执行
adb pull /system/build.prop
会得到这个固件的 build.prop 文件
对于 TargetFile，解压出来查看 SYSTEM/build.prop，对比这两个 build.prop 如果一致，表示这个固件和 TargetFile 是匹配的.

4. 4. OTA 扩展功能

4.4.1. 支持外部储存读取更新包

Android 原生的 OTA 升级包是放在/cache 分区的，但是随着版本的迭代，有可能出现 cache 分区不足以容纳 OTA 升级包的状况。针对这种情况，新版本的 Recovery 支持软连接形式，从 U 盘、SD 卡直接读取更新包，不用再把更新包复制到 cache 分区中，从而减少升级时间。
该功能都封装在 adnroid/frameworks/base/swextend/os/java/softwinner/os/RecoverySystemEx.java 中，调用该类的静态方法 installPackageEx()方法，该方法需要传入更新包的路径和应用 Context 实例。

4.4.2. Usb-Recovery 功能

Usb-recovery 模式达到让用户即使不进入 Android 系统，也能够安装指定更新包的目的，让用户在系统异常无法进入系统的情况下，安装更新包恢复系统，给用户一条还原系统的通道。
Usb-recovery 模式是指将更新包改名为 update.zip，然后放到一个 u 盘的根目录上，插入 u 盘到小机中，按着小机的 Usb-recovery 按键进入 Usb-recovery。进行 Usb-recovery 模式之后，recovery 会自动搜索并安装 u 盘上的 update.zip 包。
Usb-recovery，即可通过 U 盘升级 OTA 包，有两种方法：
方法(1) 在“设置”-->“备份和重置”-->“系统恢复\升级”-->选择需要升级的 OTA 包；
方法(2) 按住机器"recovery 键"，上电进入 recovery 升级；
必须特别注意的是升级包必须命名为 update.zip，且只能放在该分区的根目录。
具体配置如下：
打开 Usb-Recovery 功能需要修改方案的 sys_config.fex 文件里相关配置，sys_config.fex 里面配置旁边也有相关说明，如下：

```
;------------------------------------------------------------------------------
;    used: 模块使能端        1：开启模块     0：关闭模块
;    mode: 模式选择          1：一键进入 OTA 升级        2：一键恢复（通过 sysrecovery 分区来恢复）    其他值：无效
;    recovery_key ：  按键配置    （例如：recovery_key= port:PH16<0><default>）
;------------------------------------------------------------------------------
[recovery_para]
used =    1
mode =    1
```

```
recovery_key = port:PH16<0><default><default><default>
```

5. 一键恢复功能

5.1. 配置说明

注意：支持“Usb-Recovery 功能”及“一键恢复”功能。在制定方案时只能二选一。

原理：Usb-recovery 走的是 OTA 流程，而一键恢复功能走的是类似量产的流程。

一键恢复：即在系统遭受破坏时，按住机器的"recovery 键"，上电进入系统恢复功能。此次恢复的系统为出厂时的系统，不包括后续用户自己安装的任何 apk。而且采用一键恢复功能，本地存储设备需要有一块专门分区存储，这样会减少用户可用空间。

开启一键恢复功能需要做两个修改点：

(1) 在方案的 sys_config.fex 文件里，修改配置，如：

```
;-------------------------------------------------------------------------------
;    used: 模块使能端        1：开启模块     0：关闭模块
;    mode: 模式选择          1：一键进入 OTA 升级       2：一键恢复（通过 sysrecovery 分区来恢复）   其他值：无效
;    recovery_key ：  按键配置   （例如：recovery_key= port:PH16<0><default>）
;-------------------------------------------------------------------------------
[recovery_para]
used =   1
mode =   2
recovery_key = port:PH16<0><default><default><default>
```

(2) 然后在 lichee/tools/pack/chips/sun8iw7p1/configs/dolphin-xx/sys_partition.fex 文件里，添加 sysrecovery 分区

```
;------------------------------>nandk, system image backup—添加的分区
[partition]
    name            = sysrecovery
    size            = 1343488
    downloadfile = "sysrecovery.fex"
    verify          = 0
```

5.2. 注意事项

如果硬件上没有用于“一键恢复”的 GPIO，而在配置文件中配置了 system 下的 recovery_key 项，有可能会导致系统不断进入一键恢复。如果没有实际按键，把 sys_config.fex 的使能配置为 0

```
[recovery_para]
used =   0
mode =   2
recovery_key = port:PH16<0><default><default><default>
```

5.3. “一键恢复”失败常见原因

失败的常见原因：（不限于以下）

1) 硬件参数配置文件中的“recovery_key”参数是否配置正确？

2） 硬件问题：硬件上，按下按键时，GPIO 是否发生了对应的电平变化？

3） 生成的 img 文件过大，超出了 sysrecovery 分区的大小，导致烧录时就没将备份系统烧录进去。

6. Private 分区的配置与读写

通常 private 分区用在存储私有的信息，比如 mac 地址及唯一识别认证码等。
以 lichee/tools/pack/chips/sun8iw7p1/configs/dolphin-p1/sys_partition.fex 的 private 分区为例。

6.1. 配置 Private 分区

6.1.1. 分区表配置

修改 lichee/tools/pack/chips/sun8iw7p1/configs/dolphin-p1/sys_partition.fex 目录下的分区信息，如 dolphin-p1 的 private 分区的。

```
[partition]
    name        = private
    size        = 32768
    user_type   = 0x8000
    keydata     = 1
```

6.1.2. 软件层配置

修改 android\device\softwinner\dolphin-fvd-p1 下的 init.rc 文件中的如下脚本：

```
# try to mount /private
    export PRIVATE_STORAGE /mnt/private
    format_userdata /dev/block/by-name/private PRIVATE
    mkdir /mnt/private 0000 system system
    mount vfat /dev/block/by-name/private /mnt/private                    #挂载 private 分区
    exec /system/bin/busybox chmod 0777 /dev/block/by-name/rivate
```

6.2. Private 分区读写

homlet 提供一个扩展接口来读写 Private 分区，该接口位于：

android/framework/base/swextend/securefile/java/SecureFile.java

注: 构造 SecureFile 实例时,如果传入的是相对路径，则该文件位于定制的 private 分区内，如 SecureFile file = new SecureFile(“abc.rc”).

注意：对 mac 地址写入数据时，注意写入的数据大小不能超过 private 分区的大小。

SecureFile 对 private 分区中的文件操作和 File 对文件的操作类似,对文件读写的操作有如下四个接口：

```
/**把源文件内容写入本文件中
*@param srcFilePath   源文件路径,传入相对路径是表示该文件的根路径是private分区
*@param append    是否以添加的方式把数据写入文件末尾
*@return  返回真表示成功
*/
```

```
public boolean write(String srcFilePath, boolean append)

/**把源数据写入文件
*@param srcData    数据流,最大为1MB
*@param append    同上
*@return  同上
*/
public boolean write(byte[] srcData, boolean append)

/**把本文件的内容读到目标文件中,数据会覆盖掉目标文件,即append为false
*@param destFilePath    目标文件路径,可以为相对路径
*@return    同上
*/
public boolean read(String destFilePath)

/**把本文件内容读到目的数据流中
*@param destData    目的数据流,最大只能读入1MB
*@return    同上
*/
public boolean read(byte[] destData)
```

7. 系统预留内存配置

多媒体预留物理内存供 DE(Display Engine)/VE(Video Engine)/GPU 使用的大块连续物理内存的同时，在平时多媒体不使用的时候可以给系统使用。

预留物理内存的大小在**lichee/linux-3.4/arch/arm/configs/sun8iw7p1smp_android_defconfig**中进行配置，如下所示 H3 dolphin P1 机型对应的配置如下：

H3 512MB 方案默认 CMA 内存预留配置：160MB；

（512MB 方案推荐值，不要随意更改，碰到 H265 4K 播放不了的，最高配置为 170MB）

H3 1GB 方案默认 CMA 内存预留配置：256MB。

（1GB 方案推荐值，不要随意更改，碰到 H265 4K 播放不了的，根据参考帧数目适当增加，一个参考帧的大小为 12MB）

H3s 512MB 方案默认 CMA 内存预留配置为：130MB；

H3s 512MB 方案默认 CMA 内存预留配置为：200MB。

CONFIG_ION_SUNXI_RESERVE_LIST="**160M@0,256M@0,**130M@1,200M@1"

注意：@0 表示 H3，@1 表示 H3s

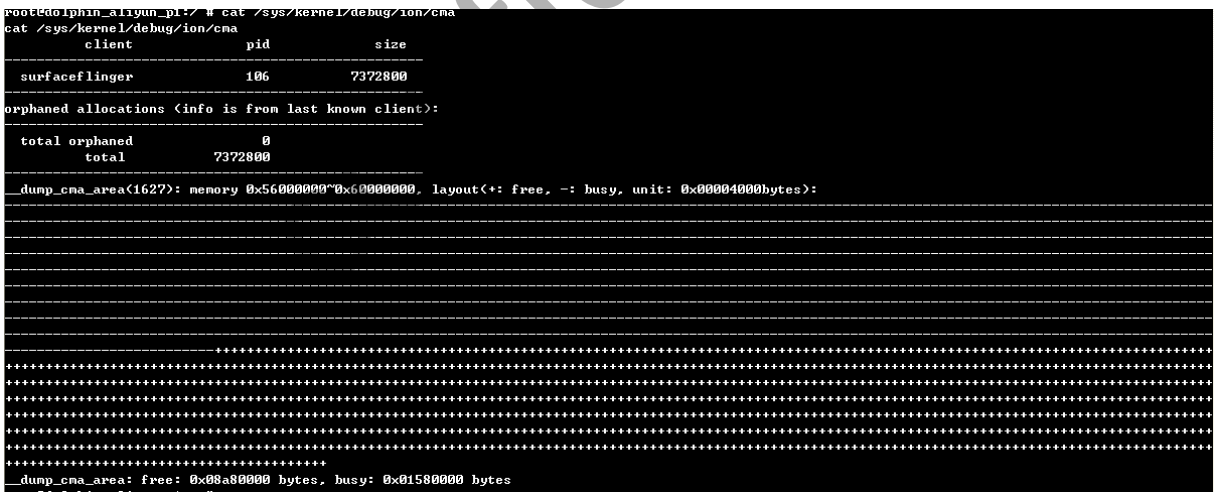
修改配置后，需要重新编译 lichee 内核，然后在 android 中通过 extract-bsp 更新内核相关文件，make -j16 编译完成后 pack 打包烧录固件即可。机器启动后，可以在串口去检查修改是否成功。

方法 1：通过内核启动打印[0.000000] cma: CMA: reserved 160 MiB at 56000000；

方法 2：可以通过命令：cat /sys/kernel/debug/cma 查看更新是否成功，如下所示：

根据截图中信息：__dump_cma_area(1627): memory 0x56000000~0x60000000

0x60000000 - 0x56000000 = 0xA000000 , 0xA000000/1024/1024 = 160MB



8. 通用定制化功能

8. 1. 遥控器配置

8.1.1. 遥控器地址码

默认情况下，红外驱动（drivers/input/keyboard/sunxi-ir-rx.c）是开启了地址码验证功能，只有通过地址校验的红外信号才会发送到应用层(android)，否则将丢弃该数据。地址码是通过 sys_config.fex 进行配置的，如：

```
[s_cir0]
ir_used             = 1
ir_rx               = port:PL11<2><1><default><default>
ir_power_key_code0   = 0x57
ir_addr_code0        = 0x9f00
```

其中 ir_addr_code0 配置的就是遥控器地址码，公版默认的地址是 0x9f00，请根据实际情况修改该值。

其他相关的配置项：
ir_used：是否启用红外遥控驱动，如果要使用红外遥控器，请设为 1，否则设为 0；
ir_rx：　红外信号输入 pin，默认不需要修改；

8.1.2. 获取遥控器的地址码、按键值

H3 平台的内核已经提供调试输出，供获取红外遥控器的原始键值。首先需要打开 ir 遥控器调试输出，命令如下：

echo 0xff　> /sys/module/sunxi_ir_rx/parameters/debug_mask

按下遥控器按键，串口输出如下：

```
    [  838.622097] IR RX IRQ Serve
    [  838.665233] IR RX IRQ Serve
    [  838.668390] dcnt = 70
    [  838.671046] 369 active_delay = 128
    [  838.674874] 384 len = 147
    [  838.675226] IR code = 0xb748fb04
    [  838.675226] IR RAW CODE : 72
    [  838.675226] IR CODE : 72
    [  838.675226] IR KEY VALE 72
    [  838.675226] ir_rx_irq_service: Rx Packet End, code=0xb748fb04,
ir_code=0xb748fb04, timer_used=1
    [  838.860144] IR KEY TIMER OUT UP
```

```
[  838.863656] ir_timer_handle: timeout
```

可以得到“IR code = 0xb748fb04”，其中后面 16bit 就是遥控器的地址码（即 0xfb04），“IR CODE : 72”，即当前按键值为 72（十进制）。

根据上述信息，即可配置红外遥控器的 kl 文件。

8.1.3. 多遥控器支持

1. 多遥控器实现简述

多遥控器的实现是通过在 android 启动的时候，动态扫描/system/usr/keylayout/目录下所有以 customer_ir_xxxx.kl 格式命名的 kl 文件，并为每个 customer_ir_xxxx.kl 文件创建映射数组 mapping；此映射数组 mapping 最终会设置到内核。

customer_ir_xxxx.kl 文件名中 xxxx 代表遥控器的地址码，假设遥控器地址码是 0x9f00，那么这个遥控器对应的 kl 文件命名应该是：customer_ir_9f00.kl。

2. 按键映射

按键映射表是供 android 层进行识别按键用的，映射表的位置在 android/device/softwinner/common/configs/keylayout 目录下的*.kl（盒子上的路径）

文件格式如下：

```
key 79   BACK                WAKE_DROPPED
key 22   MENU                WAKE_DROPPED
key 13   SEARCH              WAKE_DROPPED
key 2    DPAD_CENTER         WAKE_DROPPED
key 10   DPAD_DOWN           WAKE_DROPPED
key 67   DPAD_UP             WAKE_DROPPED
key 71   HOME                WAKE
```

对于每一个按键码（keycode）的描述的格式都固定：

key keycode FUNC WAKE/WAKE_DROPPED

其中 keycode 可以通过 getevent 获得；FUNC 为 android 定义好的的字符串，用于描述按键功能；WAKE/WAKE_DROPPED，选其一，其区别是：如果设备当前是休眠状态，如果此时接受到的按键是 WAKE 类型的，那么在唤醒机器之后，这个按键事件会同时通知到应用 app，如果是 WAKE_DROPPED 类型的按键，则仅仅是把机器唤醒，按键信息不会发送到 app。

3. 多遥控器配置

以遥控器地址 0x9f00 为例：

1. 根据遥控器地址码命名 kl 文件，如：customer_ir_9f00.kl；
2. 将 customer_ir_9f00.kl 放在 android/device/softwinner/common/configs//keylayout 目录下；
3. 修改 android/device/softwinner/common/common.mk 文件，增加需要 copy 的文件：

```
PRODUCT_COPY_FILES += \
device/softwinner/common/configs//keylayout/customer_ir_9f00.kl:system/usr/keylayout/customer_ir_9f00
.kl
```

8.1.4. 修改“软鼠标模式”下使用的键值

软鼠标模式是指使用遥控器的按键去模拟鼠标的动作，需要通过配置属性来关联遥控器按键和鼠标动作，如下所示，在方案配置文件中添加以下属性

```
#define virtual mouse key
  PRODUCT_PROPERTY_OVERRIDES += \
            ro.softmouse.left.code=21 \
            ro.softmouse.right.code=22 \
            ro.softmouse.top.code=19 \
            ro.softmouse.bottom.code=20 \
            ro.softmouse.leftbtn.code=23 \
            ro.softmouse.midbtn.code=-1 \
            ro.softmouse.rightbtn.code=-1
```

8.2. 开关机及待机

8.2.1. 配置短按和长按遥控器 power 键行为

通过修改方案目录下的 overlay/frameworks/base/core/res/res/value/config.xml 配置短按和长按遥控器的 power 键的行为

```
<!-- longPress behavior
   0 :  什么都不做
   1:    显示 GLOBAL_ACTIONS 对话框
   2:    弹出关机对话框
   3:    直接关机   -->
<integer name="config_longPressOnPowerBehavior">2</integer>
<!-- shortPress behavior
   0 :  休眠
   1:    弹出关机对话框
   2:    直接关机  -->
<integer name="config_shortPressOnPowerBehavior">0</integer>
```

8.2.2. 待机(休眠唤醒)

目前 H3 sdk 是默认支持 super standby 休眠模式，如下表，

standy_mode	CPU 是否断电	sys_config.fex
super	断电	standby_mode = 1

super standby：

1. 对于 super standby，休眠时 CPU 电源是关掉的，CPU 不耗电；

8.3. 配置出厂时的默认 launcher

在原生的 android 系统中，如果系统中存在多个 launcher，系统初次启动时将会弹出一个对话框，上面列出了当前系统中所有的 launcher。然后用户从列表中选择一个作为当前使用的 launcher。

很多用户并不懂这个列表是什么意思，从而就产生了困扰。对于许多厂家来讲，他们也希望用户第一眼看到的就是他们定制化的 launcher。

基于这种实际需求，我们新增加了一种机制，允许方案客户针对不同的方案配置出厂时的默认 launcher。

在文件 device/softwinner/dolphin-fvd-p1/dolphin_fvd_p1.mk 中，在变量“PRODUCT_PROPERTY_OVERRIDES”中增加两个配置项 ro.sw.defaultlauncherpackage 和 ro.sw.defaultlaunchercla ss。 这两个配置项分别对应所选 launcher 的 package name 和 class name。

譬如，如果想把 TvdLauncher 作为出厂时的默认 launcher，可以如下添加

```
ro.sw.defaultlauncherpackage=com.softwinner.launcher \
ro.sw.defaultlaunchercla ss=com.softwinner.launcher.Launcher
```

当然，某些方案可能仍然希望保留原生做法，那么只要不添加上述两个字段即可。

8.4. 替换鼠标图标

替换 3 个图片文件：

android\frameworks\base\core\res\res\drawable-mdpi\pointer_arrow.png

android\frameworks\base\core\res\res\drawable-hdpi\pointer_arrow.png

android\frameworks\base\core\res\res\drawable-xhdpi\pointer_arrow.png

8.5. 隐藏软键盘

关于软键盘和物理键盘的共生关系，android 原生的策略是这样的：

在需要调用输入法时，如果没有物理键盘插入，则弹出软键盘供用户输入。如果系统检测到有物理键盘输入，则隐藏软键盘，希望用户直接通过物理键盘输入。此时，状态栏上会出现一个键盘图标，点击此图标，弹出输入法列表界面，在此界面的最上部有一个开关项：“使用物理键盘”，并且此开关的状态为“开”。如果将此开关的状态切换为“关”，则软键盘将会重新弹出。

但是某些用户希望：在插入物理键盘后，软键盘仍然存在。为了满足这一要求，系统做了修改，默认情况下，如果插入物理键盘，软件盘仍然存在。

如果希望系统在这方面仍然保持 android 原生的做法，可在文件 device/softwinner/dolphin-fvd-p1/dolphin_fvd_p1.mk 文件中，

在变量“PRODUCT_PROPERTY_OVERRIDES”中增加一个新的配置项：

```
ro.sw.hidesoftkbwhenhardkbin=1
```

8.6. 添加预编译 APK

8.6.1. 源码预装

官方应用程序源码在 android/package/apps 目录下，如果需要在生成固件时包含某个 apk(如音乐播放器,超清播放器)，需要把其包名添加到需要编译的列表中。

比如要添加超清播放器，则在 android\device\softwinner\dolphin-xxx\dolphin_xxx.mk 文件内，给变量 PRODUCT_PACKAGES 添加一个新的值“Gallery3D”（注意，可能需要添加“ \”做分隔）。Gallery3D 就是该应用的包名(PACKAGE_NAME)，包名可以在每个应用程序源码的 Android.mk 文件中找到“LOCAL_PACKAGE_NAME”的值。

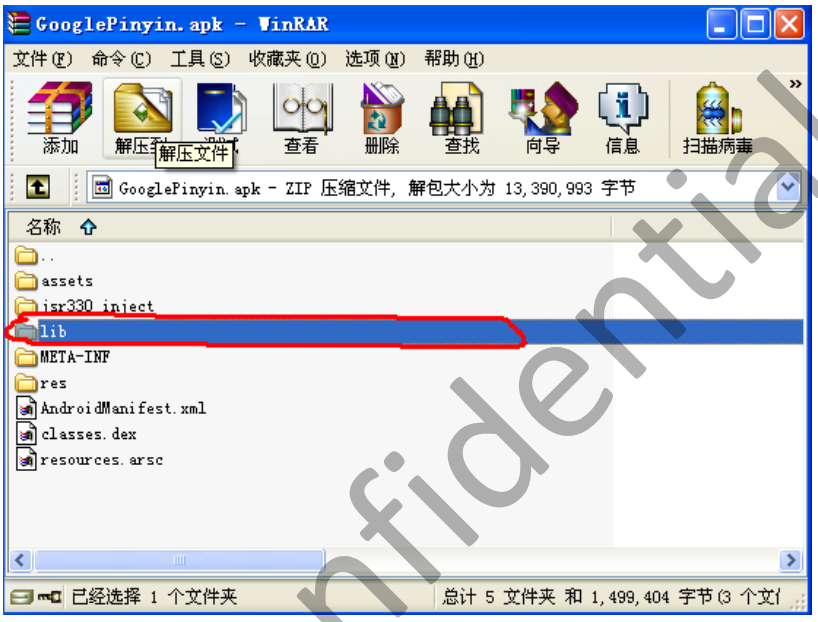
8.6.2. system 预装

对于第三方没有源码的 APK 预装，如果希望预装到 system 分区（用户不可卸载），可以按以下步骤操作：

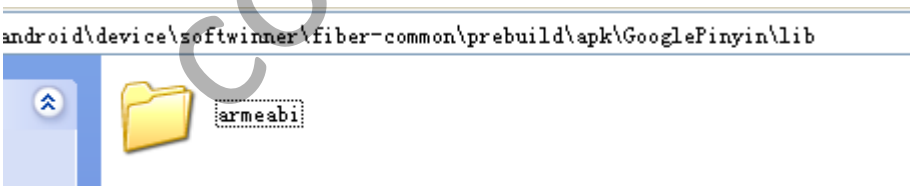
下面以 Google 拼音的预装为例：

1. 查看 APK 有没有 JNI 库：

   使用 winrar 等软件打开 apk，检查 apk 是否包含 lib 库



   如果包含 lib 库，解压其中的 lib 库，注意必须解压 armeabi 文件夹下的 mips/x86 下的无用。



2. APK/JNI 库集成到方案目录

   将完成的 APK 拷贝到 android/vendor/tvd/prebuild/apk 目录解压出来的 APK JNI 库拷贝到 android/vendor/tvd/prebuild/prebuild/apklib 目录修改 android/vendor/tvd/prebuild/prebuild/apk/Android.mk，添加如下

```
#####################################
include $(CLEAR_VARS)
LOCAL_MODULE := GooglePinyin
LOCAL_MODULE_TAGS := optional
LOCAL_SRC_FILES := $(LOCAL_MODULE).apk
```

```
LOCAL_MODULE_CLASS := APPS
LOCAL_OVERRIDES_PACKAGES := PinyinIME #需要覆盖系统默认 APK，不需要去掉此行
LOCAL_MODULE_SUFFIX := $(COMMON_ANDROID_PACKAGE_SUFFIX)
LOCAL_CERTIFICATE := PRESIGNED
include $(BUILD_PREBUILT)
###################################
```

修改 android/vendor/tvd/prebuild/prebuild/apklib/Android.mk

```
########################################
# GooglePinyin
########################################
include $(CLEAR_VARS)
LOCAL_PREBUILT_LIBS := libjni_hmm_shared_engine.so \
    libjni_googlepinyinime_latinime_5.so \
    libjni_googlepinyinime_5.so \
    libjni_delight.so \
    libgnustl_shared.so
LOCAL_MODULE_TAGS := optional
include $(BUILD_MULTI_PREBUILT)
```

3. 引用 APK 包

在 dolphin_fvd_p1.mk 中加入：

```
PRODUCT_PACKAGES += \
    libjni_hmm_shared_engine \
    libjni_googlepinyinime_latinime_5 \
    libjni_googlepinyinime_5 \
    libjni_delight \
    libgnustl_shared
PRODUCT_PACKAGES += \
  GooglePinyin
```

注意 system 预装的 APK 是不可卸载的

8.6.3. Preinstall 预装

preinstall 方式预装 apk 常用于 JNI 库较多、用户可卸载的场景，可以按照以下步骤操作：

1. APK 集成到方案目录

将 Abc.apk 拷贝至 vendor/tvd/prebuild/preinstallapk/

编写 Android.mk，参考：

```
include $(CLEAR_VARS)
LOCAL_MODULE := Abc
LOCAL_MODULE_TAGS := optional
LOCAL_CERTIFICATE := PRESIGNED
LOCAL_MODULE_PATH := $(TARGET_OUT)/preinstall
LOCAL_MODULE_CLASS := APPS
```

```
LOCAL_MODULE_SUFFIX := $(COMMON_ANDROID_PACKAGE_SUFFIX)
LOCAL_SRC_FILES := $(LOCAL_MODULE).apk
include $(BUILD_PREBUILT)
```

在方案的 mk 文件添加 PRODUCT_PACKAGES 添加 Abc

2. 重新编译，生成固件

此方法预装的 apk 不需要额外预装 jni 库，同时预装的 apk 是预装到 data 分区，可卸载。

8.7. 如何控制 GPIO

在 sys_config.fex 文件中，添加类似如下的配置信息：

```
;------------------------------------------------------------------------------

;gpio configuration
;------------------------------------------------------------------------------
[gpio_para]
gpio_used                       = 1
gpio_num                         = 2
gpio_pin_1                      = port:PH20<1><default><default><1>
gpio_pin_2                      = port:PB03<1><default><default><0>
```

在这个范例中，变量 gpio_used 置为“1”表示此配置将起作用，其他的就是各个 GPIO 的配置信息。这些 GPIO 的编码必须从“1”开始依次递增。

通常盒子会有两个 gpio 控制两个颜色的灯，为了向 Android 框架提供统一路径控制 Led 灯，所以提供一个指定控制命名 Led 的 GPIO 的配置，包括 normal 和 standby，通过 sys_config.fex 中的[led_assign]进行指定：

范例：

```
[led_assign]
normal_led    = “gpio_pin_2”
standby_led    = “gpio_pin_1”
```

其中：normal_led 指定控制系统正常运行状态下的 led 指示灯对应的 gpio，standby_led 指定控制系统在 standby 状态下的 led 控制 gpio，其取值为[gpio_para]指定的名字。

更多详细信息参考《H3_sys_config.fex 配置说明.pdf》中的[gpio_para]和[led_assign]的配置说明。

8.7.1. 定义需要控制的 GPIO

系统上电的时候，能快速的初始化用户自定义的 GPIO 口，这里包括：上电亮灯等。

配置在 sys_config.fex 文件中，根据自己方案中要上电初始化 GPIO 来添加类似如下的配置信息：

范例：

```
[boot_init_gpio]
used                    = 1
gpio_pin_1      = port:PH13<1><default><default><0>
gpio_pin_2      = port:PH14<1><default><default><1>
```

以上配置表示：在 boot 阶段，设置 PH13 输出低电平，PH14 输出高电平。

注意事项

1) 主键 [boot_init_gpio]下的子键 used 如果等于 0，那整一组模块的 GPIO 口在上电时候不会被初始化。
2) 子键的GPIO的名字是可以自定义的，例如上述gpio_pin_1、gpio_pin_2都是根据需求来自定义名称的。

比如，方案配置 lichee\tools\pack\chips\sun8iw7p1\configs\dolphin-p1\sys_config.fex 文件选项[boot_init_gpio]参数，表示在 boot 阶段，设置 power0 输出高电平。

```
;----------------------------------------------------------------------------
;    boot 阶段上电初始化 GPIO
;          used      :模块使能端       置 1：开启模块    置 0：关闭模块
;          gpiox ：上电初始化 gpio （名称自定，但不能重复，并且 GPIO 允许可以多个）
;----------------------------------------------------------------------------
  [boot_init_gpio]
  used              =    1
  pio0    =   port:power0<1><default><default><1>
```

更多详细信息参考《H3_sys_config.fex 配置说明.pdf》中的[boot_init_gpio]的配置说明。

8.7.2. 配置 boot 阶段初始化的 gpio 功能

8.7.3. 控制 GPIO 的接口

8.7.3.1. java 层的接口

java 控制 GPIO 的接口定义在文件 Gpio.java 中，其路径为：
android\frameworks\base\swextend\gpio\java\Gpio.java
在正确配置了 sys_config 的[led_assign]后，能使用 Gpio.java 的 setNormalLedOn(bool)和 setStandbyLedOn(bool)接口方便的操作 Led 的亮灭。

8.7.3.2. c++层的接口

系统启动后，在/sys/class/gpio_sw/目录下看到各个 GPIO 节点的子目录，如下图；因为只配置了一个 gpio pin，所以只看到 PH20：

```
root@android:/sys/class/gpio_sw # ls -a
PH20
```

进入在 GPIO 节点的子目录中可以看到 data、drv、cfg、pull 等，这 4 个文件文件节点就是内核 export 到用户空间的 gpio 操作接口，通过对文件节点的读写，可以灵活配置 GPIO。

```
root@android:/sys/class/gpio_sw/PH20 # ls -a
cfg
data
device
drv
power
pull
subsystem
uevent
```

- cfg：设置/读取 gpio 的功能
  0x00：input
  0x01：output
- pull：设置/读取 gpio 电阻上拉或者下拉
  0x00：关闭上拉/下拉
  0x01：上拉
  0x02：下拉
  0x03：保留
- drv：设置/读取 gpio 的驱动等级
  0x00：level 0
  0x01：level 1
  0x02：level 2
  0x03：level 3
- data：设置/读取 gpio 的电平状态
  0x00：低电平
  0x01：高电平

在 C 语言中可以用 read 和 write 函数直接操作这 4 个文件。具体的范例可参考文件 android\frameworks\base\swextend\gpio\libgpio\GpioService.cpp 中的代码。

8.8. SystemMix 可扩展接口说明

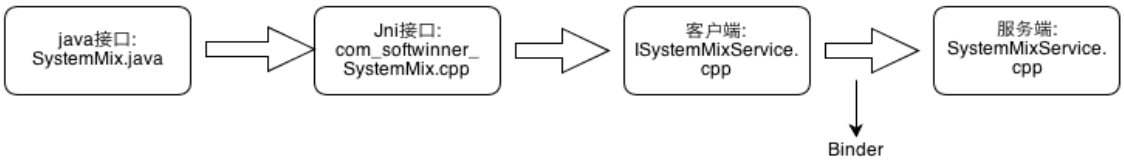
在 android/frameworks/base/swextend/systemmix 目录,我们提供了一套用于访问底层高权限信息的接口，客户可参照里面的做法来扩展该 SystemMix 类的功能。

8.8.1. 提供该接口的目的

目前有些信息，如 mac 地址、序列号等，是保存在底层文件中，一般为 root/system 等这些高权限，因此客户自己开发的 apk 没权限去访问该文件的信息，所以使用 systemmix 机制给 apk 以 root 权限去访问这些信息。

8.8.2. 原理

该机制使用了 android 上使用广泛的客户端<--->服务端机制去实现，调度流程为：



8.8.3. 接口使用

该 java 类目前提供了三个接口：

```
/** 获取某个property属性,传入属性的key值,返回其对应的value,如果key值不存
* 在,返回null
*/
public static String getProperty(String key);

/** 设置某个property属性的值,传入属性的key值和value值,如果该属性key不
*存在,则新建该属性并赋值为value
*/
public static String setProperty(String key, String value);

/** 获取系统启动参数(即系统启动后的/proc/cmdline文件中的参数)
*/
public static String getCmdPara(String name)
如获取MAC地址:String mac = SystemMix.getCmdPara(“mac_addr”);

客户在扩展该SystemMix的接口时,可参考getCmdPara()方法的调度流程,对java端,jni端,客户端,服务端都要做相应的接口扩展.
```

8.9. 开机 logo 与开机动画设置接口

由于开机 logo 和开机动画是固定的，现在 sdk 中提供了一个 AdManager 类用于设置开机 logo 和开机动画的接口：

```
/*
   描述：设置开机动画，传入开机动画文件路径，系统会拷贝一份作为开机动画
   输入：path代表开机动画压缩文件的路径
   输出：返回0代表成功，1代表失败
*/
public static int setBootAnimation(String path)

/*
   描述：设置开机logo，传入开机logo文件路径，系统会拷贝一份作为开机logo
   输入：path代表开机logo文件的路径
   输出：返回0代表成功，1代表失败
*/
public static int setBootLogo(String path)
```

设置开机 logo 还需要 sys_config.fex 设置 advert_disp 为 1，如下：

```
[boot_disp]
advert_disp          = 1
auto_hpd             = 1
output_type          = 2
hdmi_channel         = 0
```

开机 logo 的要求和开机动画的要求请看 3.2.1.1 节和 3.2.1.4 节，如果传入开机 logo 不是按照规定要求的图片，可能会显示默认开机 logo 或者显示花屏，如果传入的开机动画不符合要求也有可能显示默认开机动画或者显示异常。

8.10. USB 外设挂载配置

8.10.1. USB 配置

修改/lichee/tools/pack/chips/sun8iw7p1/configs/dolphin-xxx目录下的sys_config.fex文件的[usbc0][usbc1][usbc2]三组模块的参数，USB控制标志配置项几个名称的定义为:

配置项	配置项含义
usb_used =xx	USB 使能标志(xx=1 or 0)。 置 1，表示系统中 USB 模块可用， 置 0，则表示系统 USB 禁用。 此标志只对具体的 USB 控制器模块有效。
usb_port_type =xx	USB 端口的使用情况。(xx=0/1/2) 0：device only 1：host only 2：OTG
usb_detect_type=xx	USB 端口的检查方式。 0：无检查方式 1：vbus/id 检查
usb_id_gpio=xx	USB ID pin 脚配置。
usb_det_vbus_gpio＝xx	USB DET_VBUS pin 脚配置。
usb_drv_vbus_gpio=xx	USB DRY_VBUS pin 脚配置。
usb_host_init_state=xx	host only 模式下，Host 端口初始化状态。 0：初始化后 USB 不工作； 1：初始化后 USB 工作。

8.10.2. Android 方案 fstab 和 storage_list.xml 配置

Android 上面外部存储的挂载主要由方案目录下的 fstab.sun8i 和 overlay/frameworks/base/core/res/res/xml/storage_list.xml 共同配置。

Fstab.sun8i 里面包含两种分区的定义，第一种是 Android 启动的时候就自动挂载好，如 system 分区。

```
/dev/block/by-name/system      /system      ext4      ro      wait
```

第二种是由 Vold 管理的外部存储，主要区别在于最后的字段包含 voldmanaged=xxxx

```
/devices/platform/sunxi-ehci.1    auto    vfat    defaults        wait,check,voldmanaged=usbhost:auto
/devices/platform/sunxi-mmc.0/mmc_host    auto    vfat    defaults    wait,check,voldmanaged=extsd:auto
```

其中一个部分/devices/platform/sunxi-ehci.1 表示相应节点路径（不需要完整路径，只需要到能唯一匹配 uevent 传上来的节点路径的开头部分即可），由于是由 Vold 管理，挂载点为 auto，实际挂载点是 voldmanaged=usbhost，指向 mnt/usbhost 下面。

与 fstab 相匹配的是 overlay/frameworks/base/core/res/res/xml/storage_list.xml，由于实现 usb-hub 支持，storage_list.xml 不再需要配置 usb 存储对应的 storage，指保留内部虚拟 sdcard 和外部 sdcard 的配置即可。

```
    <storage android:mountPoint="/mnt/sdcard"
                android:storageDescription="@string/storage_internal"
                android:primary="true"
                android:removable="false"
                android:emulated="true"
                android:mtpReserve="100"
                android:allowMassStorage="false"
                android:maxFileSize="0"/>

    <storage android:mountPoint="/mnt/extsd"
                android:storageDescription="@string/storage_sd_card"
                android:primary="false"
                android:removable="true"
                android:emulated="false"
                android:mtpReserve="0"
                android:allowMassStorage="true"
                android:maxFileSize="0"/>
```


8.11. 获取 Chip ID

目前 H3 芯片每颗芯片都有自己的 chipid，可用于用户生成特定产品序列号及其他。该接口的使用请参考 android\frameworks\base\swextend\os\java\softwinner\os\ChipInfo.java。

8.12. 支持开机视频

将视频命名为 boot.mp4，放到/system/media/下或者/data/local/下。

系统启动优先从/data/local/检测视频文件，如果没有则从/system/media/下获取，如果两个路径件都没有，则默认使用启动动画。

8.13. 支持外置 USB 蓝牙 dongle

具体配置方法和已验证的支持列表请参见《H3 USB 蓝牙配置使用说明书》和《Allwinner H3 WIFI_BLUETOOTH Support list》。

9. 显示设置

9. 1. 修改显示输出设置

9.1.1. 修改遥控器快捷操作中的显示列表

在文件 Display_configs.xml(路径 android/device/softwinner/dolphin-common/overlay/frameworks/base/core/res/res/values) 中，有一个列表变量 default_display_format 和与之相对应的 default_display_format_names，根据自己的需要来增、删其中表项。

9.1.2. 修改 Settings 里面具体的显示列表

在文件 Display_configs.xml（路径 device/softwinner/dolphin-common/overlay/frameworks/base/core/ res/res/values）中，

有一个列表变量 support_tv_format_values 和与之相对应的 support_tv_format_entries，根据自己的需要来增、删其中的表项。

9.1.3. 修改显示策略

当前的策略是：

1）在 boot 阶段，按照优先级排列，决定输出显示模式的因素是：

(5) 系统检测设备当前各个显示设备端口上电缆的连接情况，此处决定输出显示类型，如 HDMI 或 CVBS；

(6) 根据输出显示类型来使用上次保存的相应的输出模式；

(7) 如果获取上次保存的相应的输出模式失败，则使用 sys_config.fex 配置的默认值；

(8) 最后，如果输出类型是 HDMI，则会在 boot 阶段根据 EDID 解析的情况来判断电视是否支持该模式，不支持则使用默认模式 720P50Hz。（此功能可配置，详见下面特别说明）

相关代码在文件 lichee/brandy/u-boot-2011.09/board/sunxi/de.c 中的函数 board_display_device_open()内。

特别说明：上述④所描述的功能可以使用配置来关闭。具体 sys_config.fex 配置和使用场景如下：

场景一：

[boot_disp]

hdmi_mode_check = 1

这配置会打开此功能：根据 EDID 解析的情况来判断电视是否支持该模式，不支持则使用默认模式 720P50Hz。

这功能主要解决的问题场景是：当前使用的电视支持 HDMI 显示模式 A，但是连接到另一台电视，上电开机时就会检查新电视是否支持显示模式 A，如不支持，则使用默认模式。这能够避免新电视不支持显示模式 A 从而出现一直黑屏无显示的状况。

场景二：

[boot_disp]

hdmi_mode_check = 0

这配置关闭 HDMI 显示模式检测功能。

关闭此功能后，（插着 HDMI 的情况下）每次开机都使用上次系统关机前使用的显示模式。因为 HDMI 显示模式检测功能依赖于电视的 EDID。如果 EDID 有问题，此功能会造成每次重启都恢复默认值，使用户体验变差，但可以通过红外遥控配置来切换到支持的模式。此时可以设置 hdmi_mode_check = 0 来关闭此功能。这就要求厂商的遥控器要定义一个专属的切换 hdmi 分辨率的按键。

2) 进入系统后,DisplaymanagerPolicy2 类对显示设备热插拔的检测,在接收到某种显示设备插拔信息时,会重新切换显示输出模式。

相关代码在文件

android/frameworks/base/core/java/android/hardware/display 中。

3) 对于遥控器操作,Homlet 定义了一个快捷键操作,点击遥控器上该快捷键时,会按根据当前显示类型切换到默认显示模式。

相关代码在文件

android/frameworks/base/policy/src/com/android/internal/policy/impl/PhoneWindowManager.java 中实现。

4）recovery 阶段，系统也会检测设备当前各个显示端口上电缆的连接情况，根据连接情况决定输出模式。

相关代码在文件 android/bootable/recovery/minui/graphics.c 中的函数 gr_init()内。

9.1.4. 优化开机显示过程(无黑屏功能)

开机过程中，显示设备只在 boot 阶段打开一次，保证从启动到进入系统的全过程无黑屏现象，开机 logo 平滑过渡。

相关代码在文件 lichee/brandy/u-boot-2011.09/board/sunxi/de.c 。

注意：此功能对 bootlogo 图片有一定的要求，具体如下，

(1) Bootlogo 格式必须是 32 位的 bmp 格式的图片。

(2) Bootlogo 图片的分辨率 1280*720。

(3)Bootlogo 图片必须以 bootlogo.bmp 为名字命名，存放位置为：lichee/tools/pack/chips/sun8iw7p1/boot-resource/boot-resource。

(4) sys_config.fex 里必须配置：

[boot_disp]

output_full = 1

9.1.5. 丽色系统

H3 平台上支持丽色系统，目前丽色系统包含三种模式：标准模式（NORMAL)，增强模式（ENHANCE）和柔和模式（SMOOTH)。

目前使用了 KeyEvent.KEYCODE_PROG_RED 按键来切换模式，如果需要修改这个遥控器按键映射，需要修改/system/usr/keylayout/对应的遥控器的 kl 文件中的 PROG_RED 按键对应的键值即可。

```
key   72    PROG_RED              WAKE
```

关于遥控器映射配置相关内容请参考 8.1 遥控器配置。

9.1.6. 显示模块相关配置

显示模块相关配置（包括显示模式、显示方式、双屏同显、横屏竖显配置等）请参考文档《H3 显示模块说明书》中“3.1.3 模块配置介绍”一节。

10.修改屏保界面

允许系统在待机时间进入屏保界面，在屏保界面下可显示时钟或厂商自己的广告页面，能够更好地提高用户体验，修改使能屏保界面的方法如下：

10.1. 设置默认屏保应用

android/device/softwinner/dolphin-fvd-p1/overlay/frameworks/base/core/res/res/values/config.xml

```
<!-- Is the dreams feature supported? -->
<bool name="config_dreamsSupported">true</bool>
<!-- If supported, are dreams enabled? (by default) -->
<bool name="config_dreamsEnabledByDefault">true</bool>
<!-- If supported and enabled, are dreams activated when docked? (by default) -->
<bool name="config_dreamsActivatedOnDockByDefault">true</bool>
<!-- If supported and enabled, are dreams activated when asleep and charging? (by default) -->
<bool name="config_dreamsActivatedOnSleepByDefault">true</bool>
<!-- ComponentName of the default dream (Settings.Secure.SCREENSAVER_COMPONENT) -->
<string name="config_dreamsDefaultComponent">com.android.dreams.web/com.android.dreams.web.Screensaver</string>
```

其中 config_dreamsDefaultComponent 为默认的屏保应用的包名，这个应用位于 android/vendor/fvd/packages/WebScreensaver 目录下

android/device/softwinner/dolphin-fvd-p1/overlay/frameworks/base/packages/SettingsProvider/res/values/defaults.xml

```
<!-- If this is true, the screen will come on when you unplug usb/power/whatever. -->
<bool name="config_unplugTurnsOnScreen">true</bool>
```

10.2. 修改广告界面

WebScreensaver 应用能检测当前是否有网络连接，无网络连接的情况下，将使用应用 asset 文件夹下自带的 HTML5 屏保界面，显示为一数字时钟。有网络连接的情况下，可获取网页显示出来：

默认的 URL 在 android/vendor/fvd/packages/WebScreensaver/src/com/android/

dreams/web/Screensaver.java 中：

```
inal SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(this);
                String    url;
                if(isNetConntected())
                        url = prefs.getString("url", "http://www.allwinnertech.com/#zh");
                else
            url = prefs.getString("url", "file:///android_asset/index.html");
          final boolean interactive = prefs.getBoolean("interactive", false);
```

第一个 URL 为在线的 URL 地址，第二个为本地 HTML5 应用的 URL，在线网页支持使用 JavaScript 动态播放网页。

10.3. 设置默认的待机时间

android/device/softwinner/dolphin-fvd-p1/overlay/frameworks/base/packages/SettingsProvider/res/values/defaults.xml

```
<integer name="def_screen_off_timeout">1800000</integer>
```

def_screen_off_timeout 即为默认的待机时间，以毫秒为单位

11.多媒体

11. 1. 音频动态管理

11.1.1. 音频管理策略

公版的默认音频切换的详细策略请见《**H3 音频模块说明书.pdf**》中的“音频输入输出切换策略”。

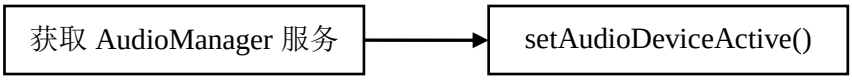
11.1.2. 音频通道输出设置接口

在 AudioManager 中我们提供以下接口来获取或设置输出的通道。
frameworks\base\media\java\android\media\AudioManager.java

```
 /* 定义三种默认音频设备的名称,如果是USB音频设备,则名字的组成为AUDIO_USB后面加上USB声卡对应卡号的id，id路径为/sys/class/sound/cardX/id，假如id为AW，则USB音频设备的名字为AUDIO_USB_AW*/
    public static final String AUDIO_NAME_CODEC       = "AUDIO_CODEC";
    public static final String AUDIO_NAME_HDMI        = "AUDIO_HDMI";
    public static final String AUDIO_NAME_SPDIF       = "AUDIO_SPDIF";
    /* define type of device */
    public static final String AUDIO_INPUT_TYPE       = "audio_devices_in";
    public static final String AUDIO_OUTPUT_TYPE  = "audio_devices_out";
    public static final String AUDIO_INPUT_ACTIVE = "audio_devices_in_active";
    public static final String AUDIO_OUTPUT_ACTIVE    = "audio_devices_out_active";
/** 获取当前可用的音频设备
* @param devType   值为AudioManager.AUDIO_INPUT_TYPE是,返回当前可用的的音频输入设备列表;或者为AudioManager.AUDIO_OUTPUT_TYPE时返回当前可用的音频输出设备列表,参数为其他值时返回null
*/
    public ArrayList<String> getAudioDevices(String devType)
/** 获取当前被使用的音频设备 */
* @param devType   值为AudioManager.AUDIO_INPUT_ACTIVE是返回当前被使用的音频输入设备列表;或者为AudioManager.AUDIO_OUTPUT_ACTIVE时返回当前被使用的音频输出设备列表,参数为其他值时返回null
*/
    public ArrayList<String> getActiveAudioDevices(String devType)
/** 把传进来的音频设备设置为当前被使用的音频输出/输入设备,每次调用该方法会更新当前被使用的设备列表
* @param devices音频设备列表,必须是getAudioDevices()得到的设备列表中的某些设备
*@param state   状态,值只能为AUDIO_INPUT_ACTIVE和AUDIO_OUTPUT_ACTIVE
*  调用该方法,会把传入的设备列表中的设备全部设置为被使用状态,这会覆盖之前的被使用设备列表,*/
    public void setAudioDeviceActive(ArrayList<String> devices, String state)
```

11.1.3. 音频相关接口使用例子

简单的用法：



可设置输入和输出的通道。通过 setAudioDeviceActive()的参数区分。

设置声音从 SPDIF 输出，代码示范：

```
mAudioManager = (AudioManager)mContext.getSystemService(Context.AUDIO_SERVICE);
mAudioManager.setAudioDeviceActive(AudioManager.AUDIO_NAME_SPDIF,AudioManager.AUDIO_
OUTPUT_ACTIVE);
```

具体的使用实例，可以参考

android/frameworks/base/services/java/com/android/serve/ AudioManagerPolicy.java

11.1.3.1. 监听 USB 音频设备热插拔

监听 USB 音频设备热插拔广播的例子：

```
//注册接收USB音频设备热插拔的信息
IntentFilter filter = new IntentFilter();
filter.addAction(Intent.ACTION_AUDIO_PLUG_IN_OUT);
mContext.registerReceiver(mBroadcastReceiver, filter);

//在mBroadcastReceiver的onReceive()方法中,关于设备热插拔信息如下
public void onReceive(Context context, Intent intent) {
Bundle bundle = intent.getExtras();
final int state = bundle.getInt(AudioDeviceManagerObserver.AUDIO_STATE);
final String name = bundle.getString(AudioDeviceManagerObserver.AUDIO_NAME);
final int type = bundle.getInt(AudioDeviceManagerObserver.AUDIO_TYPE);
//state有两个值:AudioDeviceManagerObserver的PLUG_IN和PLUG_OUT分别表示设备插入或移
除;name是该USB音频设备名;type有两个值AudioDeviceManagerObserver的AUDIO_INPUT_TYPE和
AUDIO_OUTPUT_TYPE分别表示该设备是音频输入设备还是输出设备
}
```

11.1.3.2. 音频输出/输入模式

音频输入输出设备的设置举例：

```
mAudioManager=(AudioManager)context.getSystemService(Context.AUDIO_SERVICE);
ArrayList<String> lst = new ArrayList<String>();
//设置HDMI音频输出
lst.add(AudioManager.AUDIO_NAME_HDMI);
mAudioManager.setAudioDeviceActive(lst, AudioManager.AUDIO_OUTPUT_ACTIVE);
//设置SPDIF音频输出
lst.clear();
lst.add(AudioManager.AUDIO_NAME_SPDIF);
mAudioManager.setAudioDeviceActive(lst, AudioManager.AUDIO_OUTPUT_ACTIVE);
//设置CODEC音频输出(注:如果切换显示设备到CVBS输出时,音频应该同时切换到CODEC输出)
```

```
lst.clear();
lst.add(AudioManager.AUDIO_NAME_CODEC)
mAudioManager.setAudioDeviceActive(lst, AudioManager.AUDIO_OUTPUT_ACTIVE);
// 设置某个USB音频输出,比如"AUDIO_NAME_USB0",如果有usb音频设备插入时,通过getAudioDevices(...)接口得到的可用音频设备列表中就可以看到该USB音频设备的名字
lst.clear();
lst.add("AUDIO_NAME_USB0");
mAudioManager.setAudioDeviceActive(lst, AudioManager.AUDIO_OUTPUT_ACTIVE);
以上关于设置音频设备生效的操作中,lst可以为任意个设备的组合,以实现多设备同时输出

//设置音频输入设备生效,目前只支持使用单设备做输入,因此lst中的元素只有一个,比如设置使用CODEC输入,则:
lst.clear();
lst.add(AudioManager.AUDIO_NAME_CODEC);
mAudioManager.setAudioDeviceActive(lst, AudioManager.AUDIO_INPUT_ACTIVE);

//获取当前可用的输入设备列表
lst = mAudioManager.getAudioDevices(AudioManager.AUDIO_INPUT_TYPE);
//获取当前可用的输出设备列表
lst = mAudioManager.getAudioDevices(AudioManager.AUDIO_OUTPUT_TYPE);
```

11.2. 支持 spdif 功能

11.2.1. 使能 spdif 模块

在配置文件 lichee/tools/pack/chips/sun8iw7p1/configs/dolphin-p1/sys_config.fex 中，将参数 spdif_used 配置为 1：

```
spdif_used = 1
```

同时，根据方案的原理图配置 spdif 所需的 pin 脚参数“spdif_dout”。注意，需要在此文件内排查此 pin 脚是否存在使用冲突问题。

11.2.2. 安装 spdif 驱动

修改 android\device\softwinner\dolphin-xxx\init.rc 文件,在其中 on fs 阶段加入加载 spdif 驱动的脚本。

```
on fs
#spdif
        insmod /system/vendor/modules/sunxi_spdif.ko
        insmod /system/vendor/modules/sunxi_spdma.ko
        insmod /system/vendor/modules/sndspdif.ko
        insmod /system/vendor/modules/sunxi_sndspdif.ko
```

11.2.3. 切换声道至 spdif

详情请见"音频动态管理"章节中,"音频输出模式"小节的代码例子。

11.3. 支持 USB 输入输出音频设备

11.3.1. 安装 usb audio 驱动

android\device\softwinner\dolphin-common\init.sun8i.rc 文件,在启动完成后加载 usb audio 驱动。

```
#insmod usb audio card ko
    insmod /system/vendor/modules/snd-hwdep.ko
    insmod /system/vendor/modules/snd-usbmidi-lib.ko
    insmod /system/vendor/modules/snd-usb-audio.ko
```

11.3.2. 切换声道至 USB 音频设备

详情请见"音频动态管理"章节中,"音频输出模式"小节的代码例子。

11.4. 蓝光挂载

MediaPlayer 添加提供蓝光文件(以.iso 后缀)挂载服务。

11.4.1. 蓝光使用范例

1.需要挂载蓝光文件时，调用如下接口
MediaPlayer.setDataSource(Context context, Uri uri, Map<String, String> headers)
2.使用完之后，MediaPlayer.release()会释放挂载实例。

11.5 缓冲策略定制

11.5.1 缓冲策略结构体说明

```
typedef struct CACHE_PARAM_CONFIGURATION
    {
        const char*   strApkName;
        int           eCachePolicy;
        int           nStartPlaySize;
        int           nStartPlayTimeMs;
        int           nCacheBufferSize;
```

```
}CacheParamConfig;
```

结构体重成员意义分别介绍如下：

strApkName

表示需要定制缓冲策略的 apk 名字

eCachePolicy

表示缓冲策略，不同的值代表不同策略

0:CACHE_POLICY_ADAPTIVE,

1:CACHE_POLICY_QUICK,

2:CACHE_POLICY_SMOOTH,

3:CACHE_POLICY_USER_SPECIFIC_PARAMS.

通常，adaptive mode 使用于 vod ，quick mode 最好用在直播流或者网速较快网络，smooth mode 最好在网速受限播放高码率视频时使用，例如 DLNA 播放由手机录制的视频（码率在 20MBps）。

nStartPlaySize

表示缓冲到 nStartPlaySize 大小后，即开始播放。

nStartPlayTimeMs

表示缓冲到 nStartPlayTimeMs 时间后，即开始播放，通常在点播中使用。

nCacheBufferSize

表示最大缓冲大小。

11.5.2 缓冲策略例子

例如某应用 com.xxx.livevideo 需要定制缓冲策略，即可通过如下方式来实现：

```
typedef struct CACHE_PARAM_CONFIGURATION
static CacheParamConfig CacheParamForSpecificApk[] =
{
    {"com.xxx.livevideo", 3/*cache policy*/, 0/*stat play size*/, 2000/*start play time*/,
    20*1024*1024/*buffer size*/}
};
```

上述例子中，缓冲策略为 3，表示为用户定制策略，0 表示立即开始播放，2000 表示在非直播情况下，缓冲 2s 后开始播放，20*1024*1024 表示最大缓冲大小为 20M，通过上述的设置，即可实现用户缓冲策略的定制。

11.6 切台保持最后一帧定制

修改 android/frameworks/av/media/liballwinner/LIBRARY/pri_config.h 文件。

```
#define KEEP_LAST_FRAME_FLAG 0
```

保留最后一帧通过宏“KEEP_LAST_FRAME_FLAG”定义，默认为“0”表示切台黑屏；“1”表示切台保留最后一帧。

12.减少系统启动时间

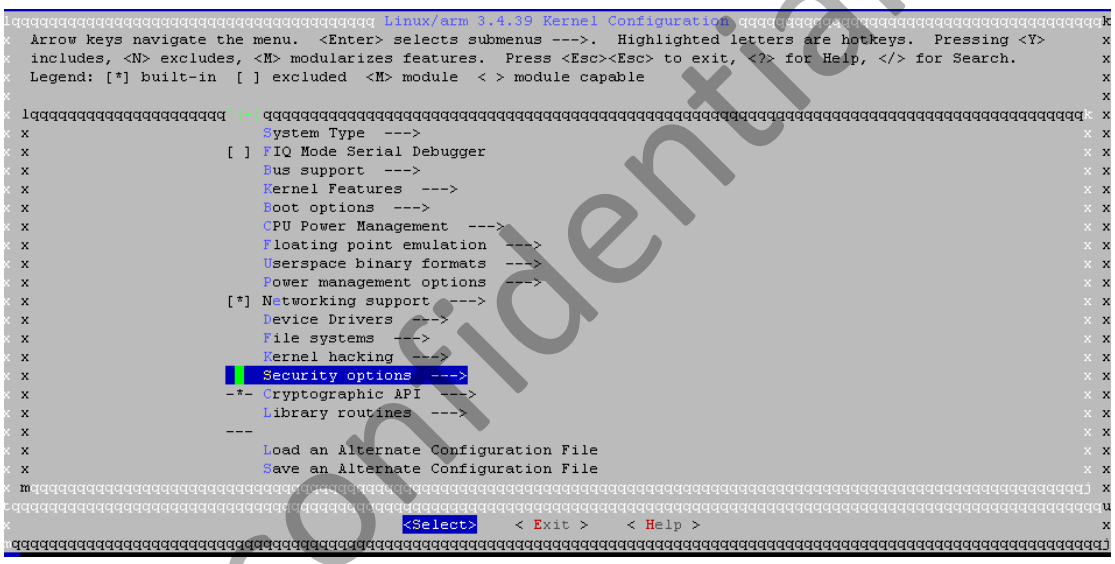
12.1. 降低内核打印等级

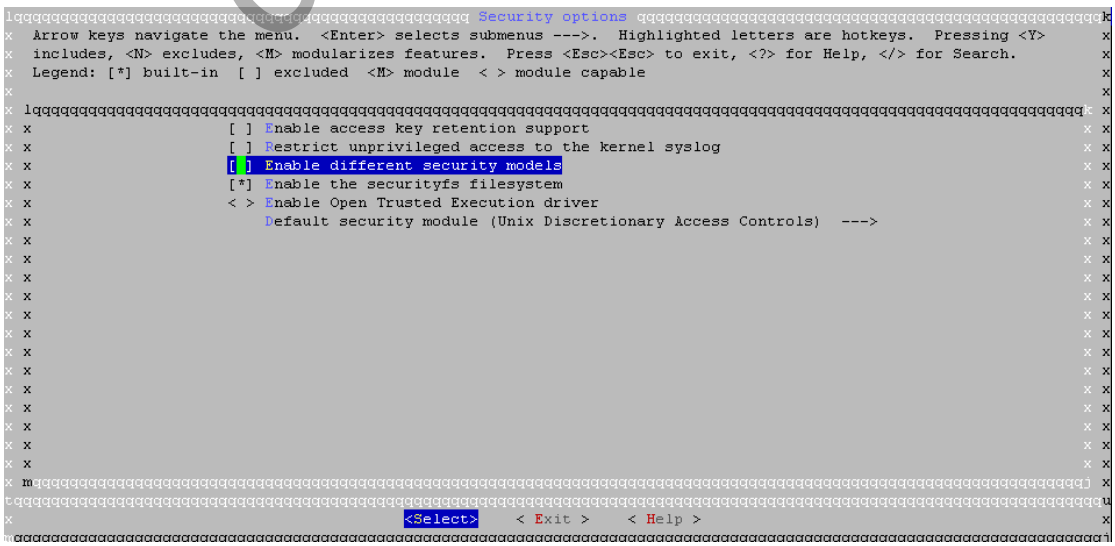
将内核打印等级调整为 4，修改 H3/lichee/tools/pack/chips/sun8iw7p1/configs/default/env.cfg

-loglevel=8 改为+loglevel=4

12.2. 去掉 SELinux（sdk 默认已经关闭）

在 lichee/linux-3.4 执行
make ARCH=arm menuconfig
选择"Security options" --> 反选"Enable different security models"，重新编译内核。





13.常用的调试方法

13.1. 提高内核打印等级

修改 lichee/tools/pack/chips/sun8iw7p1/configs/default/env.cfg 中的 loglevel 等级即可，比如：

```
loglevel=4  改为 8
```

但注意对外发布的固件一定要把 loglevel 改回 4，否则会影响系统启动速度或者系统性能！

13.2. 将 logcat 和 dmesg 信息保存到文件系统

为了调试方便，默认编译 eng 固件的时候系统会把的 logcat 和内核 dmesg 自动保存保存到/sdcard/下面。这个是在 android\device\softwinner\common\common.mk 中定义的

```
ifeq ($(TARGET_BUILD_VARIANT),eng)
        PRODUCT_COPY_FILES += \
            device/softwinner/common/init.debug.rc:root/init.debug.rc
endif
```

Init.debug.rc 是会启动 log_bg 服务来记录 log，编译 user 固件是不会保存 log 到/sdcard 目录。

13.3. 使用 fastboot 烧写

使用 fastboot 的功能来实现局部系统的更新。比如，修改内核 buildin 文件或者 init.rc 文件后，使用 make bootimage 命令可以生成 boot.img，使用 fastboot 可以将 boot.img 烧写到机器中 boot 分区，而不用烧写整个固件，从而大大缩短开发时间，命令如下：

```
# adb reboot-bootloader        #进入 fastboot 模式
# fastboot flash boot boot.img #只烧写 boot 分区
# fastboot reboot              #重启
```

使用 fastboot 烧写其他分区请参考 android fastboot 命令：

擦除分区命令：

```
# fastboot erase boot      #檫除 boot 分区
# fastboot erase system    #檫除 system 分区
# fastboot erase data      #檫除 data 分区
```

烧写分区命令：

```
# fastboot flash boot boot.img          #把 boot.img 烧写到 boot 分区
# fastboot flash system system.img      #把 system.img 烧写到 system 分区
# fastboot flash data userdata.img      #把 userdata.img 烧写到 data 分区
```

13.4. 使用网络 adb 调试

如果需要使用网络 adb 调试，可以在 android/device/softwinner/dolphin-common/init.sun8i.rc 添加下面的语句：

```
on boot
setprop service.adb.tcp.port 5555
stop adbd
start adbd
```

13.5. 调试 apk

修改应用程序 Gallery2，编译修改推送到小机

$. build/envsetup.sh

$ lunch #选择方案

$ cd packages/apps/Gallery2

$ mm

执行“mm”命令局部编译 Gallery2 应用程序，生成 Gallery2Tests.apk。如下所示。

Install: out/target/product/dolphin-xxx/system/app/Gallery2Tests.apk

然后在 windows 命令行下将生成的 Gallery2Tests.apk 推送到小机的相应目录 system/app 下即可（注：需要预先安装 adb）。如下所示：

在 windows 命令行：cmd 进入命令行模式。

> adb remount

> adb push Gallery2Tests.apk /system/app/

14. Declaration