全 志 科 技
Allwinner Technology

Sugar 定制化说明文档

V1.2

2014-06-20

Revision History

Version	Date	Section	Changes compared to previous issue
V0.6	2013-03-30		创建文档
V1.0	2013-07-23		更新文档，添加 chip id 获取；u 盘升级及一键恢复；usb 唤醒；mcu 配置
V1.1	2014-01-20	2,23	更新红外配置、AXP152 驱动配置
V1.2	2014-06-20	all	重新编排目录结构和相关章节调整，适配 a20 sugar sdkv2.1。

全 志 科 技
Allwinner Technology

目录

全 志 科 技
Allwinner Technology

引言

◆ 编写目的

本文档介绍 sugar 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。

◆ 定义

homlet：面向客厅设备的产品线的名称，目前包括 a10s，a20，a31(s)。
sugar：基于全志 A20 芯片的家庭娱乐产品 sdk 标号，已发布 sdkv1.2，sdkv2.0；最新的为 sdkv2.1。

◆ 注意事项

文档名词说明：

1. pack_xxx 表示 pack_brandy 和 pack_v1.0；
2. sugar-xxx 表示各个具体的方案，比如 sugar-evb，sugar-ref001，sugar-standard 等。

1 SDK 概述

1.1如何建立开发环境

本节将介绍 Sugar 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。

1.1.1 硬件资源

- A20 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)；

1.1.2 软件资源

Linux 主机（因为 Sugar v2.1 的软件系统方案为 android4.2，所以只能使用 64bit 系统，推荐使用 ubuntu12.04），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 2.6-2.7 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。

1.1.3 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
```

1.1.4 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
```

1.1.5 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。

1.1.6 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。

1.1.7 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。

1.2 代码下载说明

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

1.3 Sugar 公版参考方案简介

方案名称	AXP 型号	是否带 MCU	DRAM 大小	NAND/EMMC 大小	wifi 模组型号
sugar-stander	AXP152	N	1GB	TSD 4G	AP6210
sugar-ref001	AXP209	Y	1GB	NAND 8GB/TSD 8G	rtl8188eu
sugar-evb	AXP209	N	1GB	EMMC 8GB	--

1.4 Sugar 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《**A20_SugarV2.1_固件编译打包及烧录 v1.0**》。这里简要说明一下，

(1) 编译 lichee 代码

```
$cd lichee
$./build.sh -p sun7i_android
```

编译成功会打印

```
INFO: build u-boot OK.
INFO: build rootfs ...
INFO: skip make rootfs for android
INFO: build rootfs OK.
INFO: build lichee OK.
```

(2) 编译 android 代码

```
$cd android
$ source ./build/envsetup.sh
$lunch    sugar-ref001-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/sugar42/lichee/tools/pack_brandy/sun7i_android_sugar-ref001.img
pack finish
```

1.5Sugar 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/linux-3.4
lichee/tools/pack_brandy/chips/sun7i/configs/android
android/device/softwinner
```

1.5.1 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/sun7ismp_android_defconfig
```

注意：第一次编译内核时默认会复制 linux-3.4/arch/arm/configs/sun7ismp_android_deconfig 到 linux-3.4/.config 作为编译的配置；如果 linux-3.4/.config 存在的话，则不覆盖。

1.5.2 lichee/tools/pack_brandy/chips/sun7i/configs/android

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（sugar-ref001/ sugar-standard）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。

1.5.3 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。

2 添加定制的方案板配置

2.1添加定制的方案 lichee 配置

拷贝一份通用的配置，如 lichee/tools/pack_brandy/chips/sun7i/configs/android/sugar-ref001 为 lichee/tools/pack/chips/sun7i/configs/android/sugar-xxx，然后按照实际的硬件电路进行配置修改，配置的方法见《A20_sys_config.fex 配置说明 v2.1_20140616.pdf》和《A20_sys_partition.fex 分区表说明 v1.3_20140612.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 sugar-ref001 为例，在 lichee\tools\pack_brandy\chips\sun7i\configs\android\sugar-ref001 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 和 sys_partition.fex 来定义。建议使用 axp 209 的板子在 sugar-ref001 基础上修改，使用 axp152 的板子在 sugar-standard 基础上修改，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。

2.1.1 分区配置说明

以 sugar-ref001 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	512M	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
databk	256M	Databk 分区
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	656M	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
data	1024M	Android Data 分区，用于安装应用、应用数据等
UDISK	剩余大小	系统内部存储分区，可被挂载至 PC 作为大容量存储盘

2.1.2 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《A20_sys_partition.fex 分区表说明 v1.3_20140612.pdf》。

如需要增加一个名为 key 的私有分区：

```
;----------------------------->nandk, user_part partition
[partition]
    name            = key
    size            = 32768
    user_type       = 0x8000
    keydata         = 0x8000
```

系统运行起来后，在/dev/block/目录下，nandk 的节点即为 key 分区的块设备，/dev/block/key 为块设备的软连接，可以对此节点进行格式化，以文件系统的方式访问，也可以以 raw 设备的方式读写分区数据。详见“Private 分区的配置和读写”章节。

此外，从 sugar sdk v2.1 开始，挂载内部 sdcard 做了自适应处理，会挂载 sys_partition.fex 里面定义 UDISK（最后一个分区）作为内部 sdcard，所以不能修改最后一个分区的名字。对于 sugar v1.2 和 sugar v2.0 来说，新增分区后需要对 vold.fstab 做对应的修改，修改方法参见“8.13 增加分区后内部的 UDISK 盘挂不上的问题”章节。

2.2添加定制的方案板 Android 配置

2.2.1 修改方案资源

2.2.1.1 修改 bootlogo

替换 lichee/tools/pack_brandy/chips/sun7i/configs/android/sugar-xxx/ bootlogo.bmp 文件。

2.2.1.2 修改 initlogo

替换 android/device/softwinner/sugar-xxx/ initlogo.rle 文件，
这个文件可以使用一般的 bmp 图片用 lichee/tools/tools_win/LogoGen 工具生成。

2.2.1.3 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，其大小与 initlogo 一样，格式也一样。

替换 android/device/softwinner/sugar-xxx/ initlogo.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。

2.2.1.4 方案目录内文件说明

在 android\device\softwinner\目录下添加自己的方案目录 xxx（如 sugar-ref001），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 sugar-ref001 为例说明。

1. sugar-xxx.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：sugar-ref001.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，比如 TVD 盒子使用的应用程序包有针对于 Homlet 的 TvdSettings，TvdLauncher，TvdVideo，TvdFileManager。

(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 sugar-ref001 方案目录下定制了一个用于红外遥控的按键映射文件，想在编译时把它拷贝到 system/usr/keylayout 目录,则可以这么写：

```
PRODUCT_COPY_FILES += \

device/softwinner/sugar-ref001/sun7i-ir.kl:system/usr/keylayout/sun7i-ir.kl \
```

(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
```

2. AndroidProducts.mk

这里只有一句话：

```
PRODUCT_MAKEFILES := \
```

```
$(LOCAL_DIR)/sugar-ref001.mk
```

其中 sugar-ref001.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. 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《A20_Android4.2 wifi+bt 配置说明 v0.3_20140609.pdf》。

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。

4. init.sun7i.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.sun7i.rc 中加载。

5. initlogo.rle

自定义开机 log 图片，目前默认的是 720p 的图片，图片可以使用 lichee\tools\tools_win\LogoGen.zip 小工具生成。

6. needfix.rle

开机文件系统自检修复界面的 logo 图片，与 initlogo 一样大，同样可以使用 lichee\tools\tools_win\LogoGen.zip 小工具生成。

7. 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。

8. sun7i-ir.kl

针对遥控器按键的映射值配置，自定义遥控器按键的配置可参考“配置自己的遥控器”小节。

9. vendorsetup.sh

其内容通常为：

```
add_lunch_combo sugar_xxx-eng
```

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

```
add_lunch_combo sugar_xxx-user
```

编译时“lunch”会看到 sugar_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。

10. vold.fstab

这里是关于存储设备(如 SD 卡,nandflash,usb,sata)的定义，在刚创建定制的方案目录时可以不做改动，等到需要定制板子的存储设备的挂载时再修改，参考“8.11 USB 外设挂载配置”章节。

11. package.sh

打包时会被调用的脚本，以 sugar-ref001 为例内容如下：

```
#!/bin/bash
DEBUG="uart0"
SIGMODE="none"

if [ "$1" = "-d" -o "$2" = "-d" ]; then
        echo "----------debug version, have card0 printf----------"
        DEBUG="card0";
else
        echo "----------realse version, have uart0 printf----------"
fi

cd $PACKAGE
   ./pack -c sun7i -p android -b sugar-ref001    -d ${DEBUG} -s ${SIGMODE}
cd -
```

移植时需要将 sugar-ref001 改为自己方案的名称，而且名字

跟 lichee/tools/pack_brandy/chips/sun7i/configs/android 下的方案配置目录名称要一致，否则打包 pack 的时候会报错，目前该脚本支持三个参数，常用两个：

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。

2.3如何添加新的 product

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

在 android/device/softwinner 目录下，

2.3.1 创建 procuct 目录

1）clone 一个新的 sugar-ref001 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 sugar-ref001 文件夹整体复制，并重新命名为“sugar-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“sugar-ref001”替换为“sugar-xxx”。

5）将“sugar-ref001.mk”重命名为“sugar-ref002.mk”

例子，使用 shell 命令：

```
$ cd android/device/softwinner/
```

```
$ cp -r sugar-ref001 sugar-xxx
$ cd sugar-xxx/
$ rm -rf .git modules kernel                    #删除原有的 git modules 目录和内核
$ mv sugar-ref001.mk    sugar-xxx.mk       #重命名 makefile 文件
$ git init                                      #重新初始化 git 管理
$ git add * -f
$ find . -type f | xargs sed -i 's/sugar-ref001/sugar-xxx/g'   #替换掉 sugar-ref001 的字符串
$ git diff
$ git add –u
$ git commit -m "init first versioion for sugar-xxx"                    #第一次提交
```

6) 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 sugar_xxx-eng
```

2.3.2 创建此 product 对应的配置文件

1）在目录 lichee\tools\pack_brandy\chips\sun7i\configs\android\下，复制文件夹“sugar-ref001”。

2）将复制后的文件夹重命名为“sugar-xxx”。

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

2.3.3 验证新 product 的正确性

在验证之前，务必将 sugar-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。

2.3.4 创建此 product 对应的 git 仓库

如果验证无误，将备份的 sugar-xxx 目录恢复出来。

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

```
$git init

$git add .

$git commit -m "create a new product ‘sugar-xxx'."

$git branch sugar-dev       ;// sugar-dev 是主分支名称，请根据自己的实际情况决定

$cd ..
```

```
$git clone --bare sugar-xxx sugar-xxx.git
```

2）将生成的 sugar-xxx.git 推送至服务器上；

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

2.3.5 将此仓库添加至 repo 中

1）在文件 android4.2\.repo\manifests\sugar-android.xml 内，根据创建的仓库名称添加如下信息：

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

2) 将修改提交至服务器。

2.3.6 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_brandy/chips/sun7i/configs/android/sugar-xx/

3 OTA 升级说明

3.1 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 sugar 平台的产品特点，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/Boot1(Uboot)升级	√(v2.0 以上)
TSD/EMMC 方案 Boot0/Boot1(Uboot)升级	√(v2.0 以上)
sys_config.fex 更新	√(v2.0 以上)
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。

3.2 升级注意事项

3.2.1 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。分区配置参考《A20 sys_partition.fex 分区表说明》。

3.2.2 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。

举个简单的例子：假如当前在 Android 系统上选择了内部 SD 卡的一个包，它传入的包路径为”/mnt/sdcard/update.zip”。当重启进入 Recovery 时 ，因为 Recovery 环境下并不会挂载/mnt/sdcard，所以 Recovery 没有办法从这个路径找到对应的升级包。

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。

3.2.3 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，sugar-ref001 方案的 nandg 的 group 权限为 system。

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

3.3 制作更新包

3.3.1 制作完整更新包

在 Android 源代码根目录中，输入以下命令：

```
get_uboot
make otapackage -j16
```

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。

命令执行完成后，会在 cout 目录：

(out/target/product/sugar-xxx/obj/PACKAGING/target_files_intermediates/) 中 生 成 sugar_xxx-ota-xxx.zip，该文件就是完整更新包。

3.3.2 制作差分升级包

差异升级包，简称差分包，它仅使用于两个版本之间，升级必要的内容而非全部内容，体积小是它最大的优点。要生成差分包，必须获得前一版本的 target-file 文件，也将是比较文件。当我们生成完一个版本的固件之后，在 android 根目录执行以下命令：

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

在对应 out 目录下 out/target/product/sugar-xxx/obj/PACKAGING/target_files_intermediates/，生成命名为 sugar_xx-target_files-xxx.zip 的 target-file 文件，将其保存起来。

将具有版本特征的 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
```

将在 out/target/product/sugar-xxx/目录下，生成 xxx_xxx-ota-xxx-inc.zip 差分包。

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。

3.4 Recovery 界面支持遥控器

原生的 Recovery 并不支持红外遥控器操作。考虑到可能有部分用户有这方面的需求，最新版本的 Recovery 已经添加遥控器支持。该功能可以在编译时定制，若关闭遥控器支持，则进入到 Recovery 无命令时，过 5 秒自动重启，不显示 Recovery 选项界面；若打开遥控器支持，进入 Recovery 时显示 Recovery 选项界面供用户控制。

3.4.1 编译开关

在 android/device/softwinner/sugar-xxx/BoardConfig.mk 文件中添加变量。

```
# recovery IR controll
SW_BOARD_IR_RECOVERY := true
```

该标志位默认为开启状态，若需将此功能关闭，添加变量并设成 false 即可。推荐打开。

3.4.2 按键配置

目前支持三个按键，分别是“向上”、“向下”、“确定”。

如果需要修改则在 android//bootable/recovery/ui.cpp 修改以下宏定义：

```
4 #define IR_KEY_UP 67

5 #define IR_KEY_DOWN 10

6 #define IR_KEY_ENTER 2
```

修改的键码为对应 android/device/softwinner/suger-xxx/sun7i-ir.kl 的红外键码即可。

3.5 OTA 扩展功能

3.5.1 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。

3.5.2 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：

在 lichee\tools\pack_brandy\chips\sun7i\configs\android\sugar-xxx\sys_config.fex 文件里，修改相关配置，如：

```
[system]

anrecovery_key = port:PH16<0><default>

注意按键名是 anrecovery_key，如果存在一键恢复 recovery_key 的配置，只能二选一。
```

除了物理按键能进入 Usb-recovery 功能，通过红外遥控器也可以进入 Usb-recovery，由于红外遥控检测需要延时解码，且操作不方便，不建议使用，详细方法参见“遥控器进入 Usb-Recovery 和一键恢复功能”章节。

4 一键恢复功能

4.1配置说明

注意：Sugar 支持通过“Usb-Recovery 功能”及“一键恢复”功能。但两个功能由于使用同一个物理按键，因此在制定方案时只能二选一。

原理：Usb-recovery 走的是标准 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 lichee\tools\pack_brandy\chips\sun7i\configs\android\sugar-xxx\sys_config.fex 文件里，修改按键配置，如：

```
[system]
recovery_key    =    port:PH16<0><default>
```

注意按键名是 recovery_key，如果存在一键进入 Usb-Recovery 升级则配置名为 anrecovery_key，两者只能二选一。

(2) 然后在 lichee\tools\pack_brandy\chips\sun7i\configs\android\sugar-xxx\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
;------------------------------>nandl, UDISK
[partition]
    name             = UDISK
```

```
        downloadfile = "diskfs.fex"

        verify          = 0
```

Sugar v2.1 在新增新的分区后不用做任何的修改。

但在 Sugar v1.2 和 v2.0 中，由于用户 UDISK 盘由原来的 nandk 变成 nandl，因此需要修改 android 里面对本地磁盘的分区设置，具体需要修改文件 android\device\softwinner\sugar-ref001\vold.fstab。如下修改红色的 nandk 改为 nandl：

```
    dev_mount     sdcard     /mnt/sdcard     auto /devices/virtual/block/nandl
```

4.2 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。

4.3 “一键恢复”失败分析

失败的可能原因：（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。

4.4遥控器进入 Usb-Recovery 和一键恢复功能

从 SugarV2.1 开始支持通过遥控器进入 Usb-Recovey，可以通过配置遥控器的来进入恢复模式。

使用说明：

盒子上电后，马上连续短按遥控器配置的 revoery 按键，直到进入恢复模式为才停止短按按键（一键恢复是查看 LED 是否闪烁或者电视是否有进度条；一键 U 盘升级是查看电视是否进入 OTA 升级），如果没有进入恢复模式，请重复操作。

配置如下：

在 lichee\tools\pack_brandy\chips\sun7i\configs\android\sugar-xxx\sys_config.fex 文件里，修改 ir_boot_para，如：

```
    [ir_boot_para]

    ir_used                 = 0
```

```
ir_mode             = 1
ir_rx               = port:PB04<2><default><default><default>
ir_recovery_key     = 0x57
ir_addr_code        = 0x9f00
```

配置说明：

ir_used：模块使能，默认为 0 即关闭，1:；

ir_mode：1: 一键进入 U 盘升级；　2：一键恢复；　其他值：无效

ir_rx：引脚配置

ir_recovery_key：recovery 按键码，默认是 power 键值

ir_addr_code：遥控器地址码

5 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_brandy\chips\sun7i\configs\android\sugar-ref001\sys_partition.fex 的 private 分区为例。

5.1 配置 Private 分区

5.1.1 分区表配置

修改 lichee\tools\pack_brandy\chips\sun7i\configs\android\sugar-ref001 目录下的分区信息，如 sugar-ref001 的 nandh 是作为 private 分区的。

```
;------------------------------>nandh
[partition]
    name            = private
    size            = 32768
    user_type       = 0x8000
    keydata         = 1
```

5.1.2 软件层配置

修改 android\device\softwinner\sugar-ref001 下的 init.sun7i.rc 文件中的如下脚本：

```
# try to mount /private
    export PRIVATE_STORAGE /mnt/private
    format_userdata /dev/block/private PRIVATE
    mkdir /mnt/private 0000 system system
    mount vfat /dev/block/private /mnt/private                  #挂载 private 分区
    exec /system/bin/busybox chmod 0777 /dev/block/private    #修改 private 块设备权限，供 Dragon apk 访问
```

5.2 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)
```

6 系统预留内存配置

多媒体预留物理内存是仅供 DE(Display Engine)/VE(Video Engine)/MP 使用的连续物理内存，这段内存其他模块时无法使用的。

预留物理内存的大小在 **lichee/linux-3.4/arch/arm/mach-sun7i/include/mach/memory.h** 中进行配置，如下所示：

```
#define HW_RESERVED_MEM_SIZE_1G          SZ_128M
#define HW_RESERVED_MEM_SIZE_512M        (SZ_64M+SZ_32M+SZ_8M)
```

内核启动会根据 dram 的大小来自适应设置上面的预留内存，

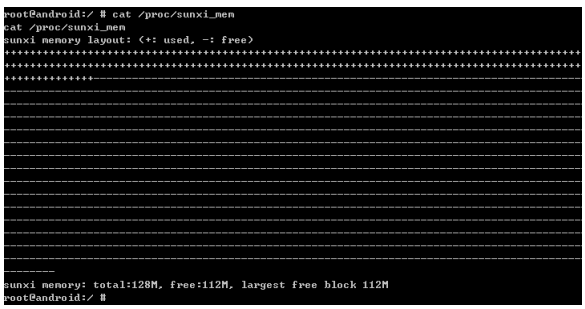
(1) 1G 方案默认配置 128M，该配置基本可以覆盖所有蓝光视频对内存的需求；

(2) 512M 方案默认配置 104M 内存，该配置可以播放基本的蓝光视频（1080P，3 参考帧）。

由于 A20 平台 512M 方案的内存资源比较紧张且珍贵，如有需求更多内存用于其他方面或者不需要支持蓝光视频的播放（如仅需要支持 1080P 3D 视频播放或者 1080P 2D 视频的播放），可以参考下表自行修改上面的配置：

A20 预留内存配置	物理内存配置	
	DE/VE/CSI/MP 预留	系统可使用内存（cat /proc/meminfo）
A20 512M 方案	**80M（2D 1080P）**	**383M**
	88M（3D 1080P / 2160P）	**391M**
	104M（蓝光 ISO 参考帧 3）	**407M**
A20 1G/2G 方案	**128M（蓝光 ISO）**	**895M**

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 shell 下通过 cat /proc/sunxi_mem 查看更新是否成功。如下所示，1G 方案默认配置为 128M 预留内存，



7 AXP152 电源驱动配置

SDK 默认的电源配置是 AXP209；如果使用 AXP152，则需要修改 kernel 的配置文件；如果硬件与配置不相符，会导致机器不能启动。具体配置方法呢如下：

1.1 配置 AXP 类型

$ make ARCH=arm menuconfig

1.选择 axp 类型（axp15），如下图：

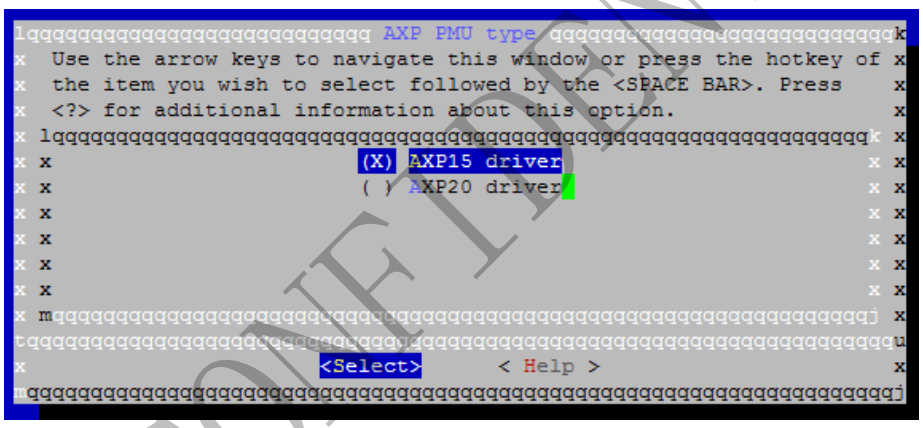
--> Device Drivers

--> Power supply class support (POWER_SUPPLY [=y])

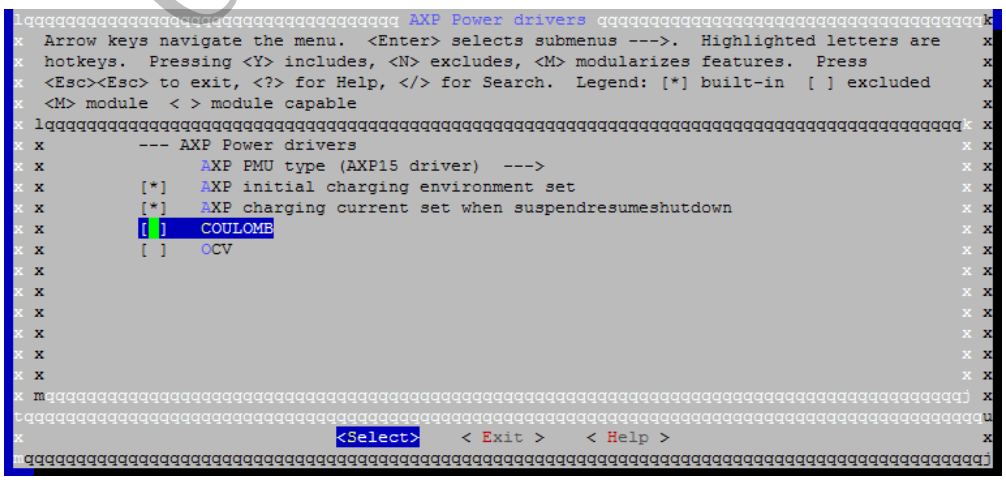
--> AXP Power drivers (AW_AXP [=y])

--> AXP PMU type (<choice> [=y])

--> AXP15 driver



2.返回上一级，**将 AXP Power drivers 下的 COULUMB 项去掉，否则编译失败：**



1.2 配置 sys_config.fex

AXP152的 i2c 地址与AXP209不一样，为正常使用AXP152驱动，需要在方案配置文件中配置AXP152地址；同时，还需要配置 AXP152各路初始化时候的输出电压，保证系统核心模块正常供电。具体配置如下：

1. 初始化电压设置：

```
[axp15_para]
dcdc1_vol       = 3000
dcdc2_vol       = 1250
dcdc3_vol       = 1500
dcdc4_vol       = 1250
aldo1_vol       = 3000
aldo2_vol       = 3000
dldo1_vol       = 3300
dldo2_vol       = 3300
```

2. 配置 I2C 地址：

```
[pmu_para]
pmu_used                    = 1
pmu_twi_addr                = 0x30
```

8 通用定制化功能

8.1遥控器配置

此处只做简单介绍，更详细的遥控器配置介绍在《A20_红外遥控与休眠唤醒配置 v1.0》。

8.1.1 修改红外遥控的地址码

地址码在文件 lichee/tools/pack_brandy/chips/sun7i/configs/android/sugar-ref001/sys_config.fex 中，根据自己的遥控器的地址配置[ir_para]的参数：

```
ir_addr_code = 0x9f00
```

譬如说，如果地址码为 0x7f80，则修改成：

```
ir_addr_code = 0x7f80
```

如果发现无效，则将两个字节的值交换一下位置，修改成：

```
ir_addr_code = 0x807f
```

获取遥控器地址码的具体方法可以参考：《A20_红外遥控与休眠唤醒配置 v1.0》的 2.2.1 获取遥控器地址的说明。

8.1.2 修改按键映射

在文件 android/device/softwinner/sugar-ref001/sun7i-ir.kl 中，重新建立按键扫描码与系统中定义的按键名称的映射关系。

按键扫描码可以通过在串口中输入 getevent，然后点击按键时看打印出来的键值来确定。

注意：扫描码不能重复，否则此文件将失效；kl 文件的具体格式可参考《A20_红外遥控与休眠唤醒配置 v1.0》文档中的 2.2.2 节。

8.1.3 配置无 MCU 的红外唤醒功能

在文件 lichee/tools/pack_brandy/chips/sun7i/configs/android/sugar-ref-001/sys_config.fex 中，跟无 MCU 的红外唤醒功能相关的配置在[ir_para]中；如果需要打开该功能，将 ir_wakeup 的值设为 1，有 MCU 的板子请将该值设为 0。

一个完整的配置如下：

```
[ir_para]
ir_used                 = 1
ir_rx                   = port:PB04<2><default><default><default>
;不带 mcu 的需要设置为 1，带 mcu 的设置为 0
ir_wakeup = 1
;遥控器的电源键键值
power_key = 0x57
;遥控器的地址码
ir_addr_code = 0x9f00
```

注意：遥控器的电源键键值和遥控器的地址码必须配置正确，无 MCU 红外唤醒功能才能正常使用，遥控器的电源键键值必须与 sun7i-ir.kl 的一致。

8.1.4 修改“软鼠标模式”下使用的键值

(待续)

8.2开关机及待机

8.2.1 关机配置

8.2.1.1 关机时不再出现对话框

在文件 device\softwinner\sugar-ref001\sugar-ref001.mk 中 PRODUCT_PROPERTY_OVERRIDES 字段后面追加一个属性值，如下所示：

```
ro.statusbar.directlypoweroff = true;
```

就可以在关机时不再出现对话框，而是直接进入关机流程，界面上仅仅出现一个提示。

如果希望保留原生做法，那么可以将其值设为“false”或者删除此字段。

此值默认为“false”。

8.2.1.2 短按 power 键关机

Android 原生的做法是：长按 power 键关机，短按 power 键进入 standby。

某些方案可能希望短按遥控器的 power 键进入关机，为此，在文件

device\softwinner\sugar-ref001\sugar-ref001.mk 中 PRODUCT_PROPERTY_OVERRIDES 字段后面追加一个属性值，如下所示：

```
ro.sw.shortpressleadshut = true;
```

此值默认为“false”。

8.2.2 开机配置

因为盒子产品多为遥控器操作，此处针对遥控器开机进行简要说明。遥控器开机可分为两种类型：带 MCU 方案和不带 MCU 方案。

8.2.2.1 带 MCU 方案

关机状态下，CPU 是断电的，为实现遥控器开机功能，方案中增加了 14 pin 的 MCU；MCU 在关机状态下是不掉电的，能够对红外遥控器的信号进行解码。如果 MCU 解析到遥控器的 power 键被按下，将唤醒 AXP 输出电源，CPU 上电开机。

注意：MCU 的固件在解析红外信号的时候首先是匹配红外遥控器地址码，进而匹配 power 按键键值；因此 MCU 是与遥控器是配套使用的，如果遥控器地址或者 power 按键值不匹配，不能通过遥控器开机。

8.2.2.2 不带 MCU 方案

如果方案中不带 MCU，关机状态下（CPU 断电）是不能够通过遥控器开机的；这种方案通常设计为按遥控器 CPU 进入待机，待机状态下按下遥控器，系统恢复之前的运行状态。

注意：方案设计的时候，需要明确是否需要遥控开机的功能，合理选择是否配置 MCU；或者在 PCB layout 的时候预留 MCU 的位置，作为备用。

8.2.3 待机(休眠唤醒)

目前 A20 sdk 是支持 super standby 和 normal standby 两种休眠模式，两种休眠模式对比如下表，

super standby/normal standby 对比

standy_mode	CPU 是否断电	sys_config.fex	PMU 要求	MCU 要求
super	断电	standby_mode = 1	axp209	必须
normal	不断电	standby_mode = 0	axp152/axp209	可选

super standby 与 normal standby 最明显的区别是 CPU 是否断电；

1. 对于 super standby，休眠时 CPU 电源是关掉的，CPU 不耗电；
2. 对于 normal standby，休眠时 CPU 电源并没有关闭，CPU 进入 WFI（wait for interrupt）模式。

需要注意的是，super standby 是需要硬件支持的：

第一：PMU 芯片必须使用 axp209，axp152 不支持 super standby；

第二：硬件上必须配置 MCU，如果是无 MCU 方案，不能使用 super standby。

注意：具体的休眠唤醒模式请根据实际需求进行配置，详细配置方法见《A20_红外遥控与休眠唤醒配置 v1.0》。

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。

8.3.1 配置默认 launcher

在文件 android4.2\device\softwinner\sugar-ref001\sugar-ref001.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
```

8.3.2 保留原生做法

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：

android4.2\frameworks\base\core\res\res\drawable-mdpi\pointer_arrow.png

android4.2\frameworks\base\core\res\res\drawable-hdpi\pointer_arrow.png

android4.2\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/sugar-ref001/sugar-ref001.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。

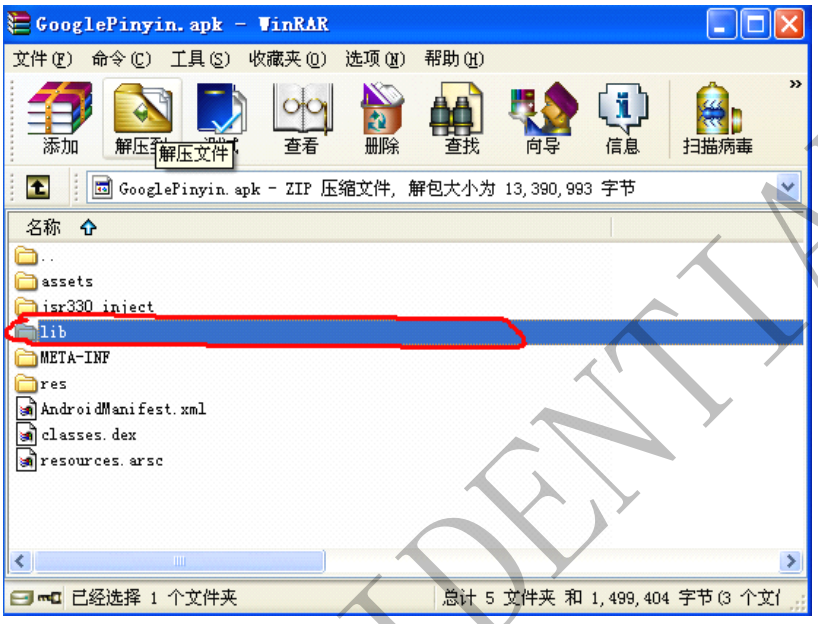
比如要添加超清播放器，则在 android\device\softwinner\wing-common\ProductCommon.mk 文件内，给变量 PRODUCT_PACKAGES 添加一个新的值“Gallery3D”（注意，可能需要添加“ \”做分隔）。Gallery3D 就是该应用的包名(PACKAGE_NAME)，包名可以在每个应用程序源码的 Android.mk 文件中找到“LOCAL_PACKAGE_NAME”的值。

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/device/softwinner/wing-common/prebuild/apk 目录
   解压出来的 APK JNI 库拷贝到 android/device/softwinner/wing-common/prebuild/apklib 目录
3. 重新编译，生成固件

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 APK 拷贝至 android/device/softwinner/wing-common/preinstallapk
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去除 GMS 框架

Sugar-ref001 默认预装 GMS 框架，如果无需求，可以关闭，这样可以节省内存，修改如下位置

代码：andrioid/device/softwinner/wing-common/ProductCommon.mk

```
#$(call find-copy-subdir-files,*,$(LOCAL_PATH)/googleservice/gapps-jb-20130301-signed/system,system) \
```

加上前面的注释，重新编译，便可去除内置 GMS 服务。

8.8多屏互动设备名称

在多屏互动场景中，为了统一多个服务向客户端展示的设备名称，现在系统中增加了一个属性“persist.sys.device_name”，在文件 android/device/softwinner/sugar-ref001/sugar-ref001.mk 文件中定义，默认值为“MiniMax”。

方案商可以在出厂前修改此属性的值，也可以在 Settings 应用程序中添加一个设置项，使得消费者可以自定义设备名称。

8.9如何控制 GPIO

8.9.1 定义需要控制的 GPIO

参考 lichee\tools\pack_brandy\chips\sun7i\configs\android\sugar-standard\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><1>
```

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。

8.9.2 配置 boot 阶段初始化的 gpio

功能

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 lichee\tools\pack\chips\sun7i\configs\android\sugar-xxx\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]
use             = 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 use 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_brandy\chips\sun7i\configs\android\sugar-ref001\sys_config.fex 文件选项[boot_init_gpio]参数，表示在 boot 阶段，设置 ph20 输出高电平。

```
;------------------------------------------------------------------------
;     boot 阶段上电初始化 GPIO
;          use       :模块使能       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]
   use              =    1
   gpio0       = port:PH20<1><default><default><1>
```

8.9.3 控制 GPIO 的接口

8.9.3.1 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

8.9.3.2 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_level：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。

8.10SystemMix 可扩展接口说明

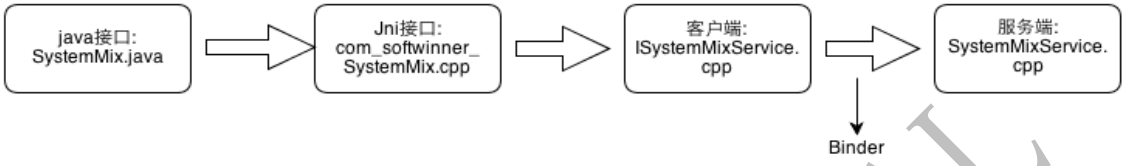
在 android/frameworks/base/swextend/systemmix 目录,我们提供了一套用于访问底层高权限信息的接口，客户可参照里面的做法来扩展该 SystemMix 类的功能。

8.10.1 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。

8.10.2 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：



8.10.3 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.

8.11 USB 外设挂载配置

8.11.1 USB 配置

修改lichee\tools\pack_brandy\chips\sun7i\configs\android\sugar-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。 具体请参考 gpio 配置说明。 《配置与 GPIO 管理.doc》
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。

8.11.2 vold.fstab 配置

A. 修改 android/device/softwinner/<方案>/vold.fstab 文件。

该文件定义了每个存储设备的挂载点，其中每一行代表一个存储设备,它的格式为：

dev_mount <设备标签> <挂载点> <分区个数(一般设为auto)> <存储设备在文件系统上的路径> 中间用 tab 制符号隔开

如要定义 SATA 设备，它挂载到/mnt/sata 目录下，设备路径为/devices/platform/sw_ahci.0，那就添加这么一句

```
dev_mount          sata       /mnt/sata          auto       /devices/platform/sw_ahci.0
```

假设我现在要在 vold.fstab 中添加一个 USB1 的设备的定义，就先打开板子的打印，输入 logcat,然后在该 USB1 的插口插入一个 U 盘，这时会看到一个类似于如下的打印：

```
I/USB3G    (    90): event { 'add', '/devices/platform/sw-ehci.1/usb1/1-1/1-1.3', 'usb', '', 189, 3 }

I/USB3G    (    90): path : '/sys/devices/platform/sw-ehci.1/usb1/1-1/1-1.3'

I/USB3G    (    90): VID :size 5,vid_path '/sys/devices/platform/sw-ehci.1/usb1/1-1/1-1.3/idVendor',VID    '17ef
```

其中第一行的"/devices/platform/sw-ehci.1/usb1/1-1/1-1.3"就是该接口设备在文件系统上的路径，这样就可以在 vold.fstab 里添加该设备的定义，如下：

```
dev_mount          usbhost1           /mnt/usbhost1      auto       /devices/platform/sw-ehci.1/usb1/1-1/1-1.3
```

这样在 USB1 设备插入时,会把它挂载到/mnt/usbhost1 路径下。

B. 修改 android4.2\device\softwinner\< 方 案 >\overlay\frameworks\base\core\res\res\xml 中 的 storage_list.xml 文件，在这里添加自己的在 vold.fstab 中定义的 USB 设备,可以仿照其他的来写，其中每个参数的含义在该文件头有说明.该文件定义了上层应用读取的设备列表。

C.修改 android\device\softwinner\<方案>\目录下的 init.sun7i.rc 文件，在

"on early-init"的地方添加创建这些挂载目录的文件，如：

```
        mkdir /mnt/usbhost0 0000 system system

        mkdir /mnt/usbhost1 0000 system system
```

8.12获取 Chip ID

目前 A20 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。

8.13增加分区后内部的 UDISK 盘挂不上的问题

对于 sugar v1.2 或者 v2.0，如果添加了新的分区后，还需要对 vold.fstab 做修改，否则会发现增加分区后内部的 UDISK 盘挂不上的问题，比如在 sys_partition.fex 文件中定义了 n 个分区，那剩下的第 n+1 个就是作为本地盘/mnt/sdcard，假设目前定义了 8 个分区，分别是从 nanda~nandh，那剩下的 nandi 就是本地盘。

方法：修改 android\device\softwinner\<方案名>\目录下的 vold.fstab 文件，把

```
dev_mount	sdcard	/mnt/sdcard	auto	/devices/virtual/block/nandi
```

这句话的/devices/virtual/block/nandi 改为自己定制的本地盘的名字，该文件是在管理 android 存储设备挂载的 vold 启动后被读取。比如新增了 1 个，此时 nandi 改为 nandj。

9 显示设置

9.1全屏显示

在原生的 android4.2 系统中，界面底部的状态栏始终存在。但对于在客厅使用的设备，隐藏状态栏是有必要的。为此，homlet SDK 中提供了两种机制用来隐藏状态栏。

9.1.1 总是隐藏状态栏

系统新添加了一个属性“ro.statusbar.alwayshide”，如果此属性的值为“true”，那么状态栏总是隐藏。

此属性值默认为“false”。

9.1.2 使能 windowFullscreen

上诉机制一刀切，要么始终有状态栏，要么始终没有。很多客户希望由应用程序自身确定是否全屏，也就说希望原先在 2.3 之前的 android 系统上能够全屏显示的 activity 在 421 的系统上仍然能够全屏显示。

为此，在 homlet 4.2 的 SDK 中，我们必须做改动，使得原先在 2.3 及之前版本的 android 上用于全屏显示的标签或代码在 4.2 上仍然起作用。譬如如下代码：

```
<activity android:name=".ActivityDemoActivity"
          android:label="@string/app_name"
android:theme="@android:style/Theme.NoTitleBar.Fullscreen"
>
```

以上代码在 2.3 及之前的系统中，能够使 activity 全屏显示，但在 4.2 上平板模式下就不起作用了。在我们改造后，这段代码仍然起作用。

目前系统中，默认使能这一机制。如果希望遵循原生的系统，可在文件 device\softwinner\sugar-ref001\sugar-ref001.mk 中 PRODUCT_PROPERTY_OVERRIDES 字段后面追加一个属性值，如下所示：

```
ro.statusbar.inheritfullscn = false;
```

注意：“ro.statusbar.inheritfullscn”的配置只有在“ro.statusbar.alwayshide”为“false”的情况下才起作用。

9.2修改显示输出设置

9.2.1 修改遥控器快捷操作中的显示列表

在文件 DispList.java（路径 android\frameworks\base\core\java\android\view）中，有一个列表变量 mShortCutArray，根据自己的需要来增、删其中表项。

9.2.2 修改 Settings 里面具体的显示列表

在文件 DispList.java（路径 android\frameworks\base\core\java\android\view）中，有一个列表变量 mItemArray，根据自己的需要来增、删其中的表项。

9.2.3 修改显示策略

当前的策略是：

1）在 boot 阶段，系统检测设备当前各个显示端口上电缆的连接情况，根据电缆连接情况和上次保存的输出模式这 2 个因素，决定输出模式。

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。

2）在系统进入 launcher 之前，系统将根据当前输出模式、当前电缆连接情况和上次保存的输出模式这 3 个因素，重新决定输出模式。

相关代码在文件 android4.2\frameworks\base\services\java\com\android\server\SystemServer.java 中函数 run()内的最后 100 多行。

3) 系统进入 launcher 的时候,会在系统 APP—SystemUi 中去注册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\packages\SystemUI\src\com\android\systemui\statusbar\policy\DisplayController.java 中。

4) 对于遥控器操作,Homlet 定义了一个叫”Tv System”的快捷键操作,点击遥控器上该快捷键时,会按顺序切换当前的显示模式。

相关代码在文件

android\frameworks\base\policy\src\com\android\internal\policy\impl\PhoneWindowManager.java 中_interceptKeyBeforeDispatching()方法的 keyCode == KeyEvent.KEYCODE_TV_SYSTEM 时的处理中

和子类 MyDispList 的 onHide()方法的实现中。

5）recovery 阶段也牵扯到显示问题，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。

显示的输出策略参见《A20_显示配置和接口说明文档 v1.0_20140605.pdf》中的“4.1 热插拔消息处理”。

9.2.4 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 图片的最大支持分辨率为 1920*1080，建议使用不超过分辨率 1280*720 的 bmp 图片。

(3) Bootlogo 图片的存放位置为：lichee\tools\pack_brandy\chips\sun7i\boot-resource\boot-resource，必须以 bootlogo.bmp 为名字命名。

详见《A20_显示配置和接口说明文档 v1.0_20140605.pdf》里面的“2.3.Bootlogo 的规格要求”一节。

10 多媒体

10.1配置流媒体缓冲策略

参考 android/frameworks/av/media/CedarX-Projects/CedarXAndroid/IceCreamSandwich$/CedarXPlayer.h 文件中声明了接口

```
bool setCacheParams(int nMaxCacheSize, int nStartPlaySize, int nMinCacheSize, int nCacheTime, int bUseDefaultCachePolicy);
```

用于设置网络流媒体播放的缓冲策略，该接口各参数作用如下表所示：

参数	说明
nMaxCacheSize	缓冲区最大值，当下载的数据量超过 nMaxCacheSize 字节时，缓冲线程停止继续下载数据；
nStartPlaySize	启动播放的数据量大小，当播放器处于自动缓冲状态，播放器检测到数据量大于 nStartPlaySize 时，从缓冲状态恢复到播放状态；
nMinCacheSize	最小缓冲数据量，当播放器处于播放状态，播放器检测到数据量小于 nMinCacheSize 字节时，从播放状态切换到缓冲状态；
nCacheTime	播放器根据 nCacheTime 计算 nStartPlaySize 的大小，nCacheTime 单位为秒，nStartPlaySize = nCacheTime * Bitrate，Bitrate 为当前比特率，该计算的意义为缓冲大约 nCacheTime 秒的数据量再开始播放；
bUseDefaultCachePolicy	是否使用播放器默认策略，1：使用，0：不使用 默认播放策略数值： nMaxCacheSize：20M nStartPlaySize：计算所得 nMinCacheSize：64K nCacheTime：30

该函数需要在播放器的 Prepare()函数被调用后才能被调用，播放过程中随时可设。

10.2音频动态管理

10.2.1 概要

音频动态管理机制主要有三个特点：

1.非透传模式下，支持多通道同时输出，透传模式下支持 HDMI，SPDIF 单路输出；

2.支持单通道输入；

3.支持 USB 音频设备热插拔检测；

这三个功能都有相应的接口。

10.2.2 音频管理策略

Sugar v2.1 公版的默认音频切换的详细策略请见《**A20_音频切换策略_V1.2_20140616.pdf**》。

目前公版对于音频动态管理机制的策略如下：

a.上电时，假如插入 HDMI，就从 HDMI 输出；假如只插入 AV 那么就从 AV 输出。加入什么都不错直接启动，那么默认是从 AV 端输出。

b.然后注册检测显示设备的热插拔广播信息,显示输出设备热插拔,音频输出做相应改变,代码 f 在 rameworks\base\packages\SystemUI\src\com\android\systemui\statusbar\policy\DisplayController.java

c.在系统 APK:SystemUI 起来后注册监听 USB 音频设备和耳机的热插拔信息.目前公版的做法是:

对于输入:启动时,先检查是否有 USB 音频输入设备接入,有则切换至该输入设备

对于输出:启动时,先检查耳机是否已插入,有则切换至该输出设备,否则再检查 USB 设备,有多个输出设备接入时,只选择第一个.

然后开始监听热插拔广播.

每次有新的 USB 设备插入则切换至该输入通道,拔出时切换至默认的 codec 音频输入通道.

每次有新的 USB 输出设备插入,或耳机插入,则在状态栏中弹出"音频输出设备插入"的通知,用户点击该通知会弹出关于"选择音频输出模式"的设置项,可多选.代码位于 frameworks\base\packages\SystemUI\src\com\android\systemui\statusbar\policy\SoundController.java

d.针对遥控器操作,由于定义了一个快捷键用于点击时切换显示输出模式,所以音频输出部分也要做相应的切换.代码位于

frameworks\base\policy\src\com\android\internal\policy\impl\PhoneWindowManager.java 的截获到 key 时对于 keyCode == KeyEvent.KEYCODE_TV_SYSTEM 情况的处理中.

由于设置中有手动切换显示输出模式的设置项,因此音频在显示模式切换时也要做相对应的切换,代码位于

android4.2\device\softwinner\fiber-common\prebuild\packages\TvdSettings\src\com\android\settings\DisplaySetting.java 的最后十几行

e.用户可自己设置选择哪些音频输出模式(设置->声音->选择音频输出模式),该代码位android4.2\device\softwinner\fiber-common\prebuild\packages\TvdSettings\src\com\android\settings\SoundSetting.java 和 AudioChannelsSelect.java

10.2.3 上层提供的接口如下

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,则名字叫做"AUDIO_USB0","AUDIO_USB1",... */
    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)
```

10.2.4 音频相关接口使用例子

可以参考

frameworks\base\packages\SystemUI\src\com\android\systemui\statusbar\policy\SoundController.java中的代码

10.2.4.1监听 USB 音频设备热插拔

```
//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
```

```
}
```

10.2.4.2音频输出/输入模式

(1)操作

```
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);
```

10.3支持 spdif 功能

10.3.1 使能 spdif 模块

在配置文件 lichee\tools\pack\chips\sun7i\configs\android\<方案>\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。

10.3.2 安装 spdif 驱动

修改 android\device\softwinner\<方案>\init.sun7i.rc 文件,在其中加入加载 spdif 驱动的脚本。

```
#spdif

    insmod /system/vendor/modules/sun7i_spdif.ko

    insmod /system/vendor/modules/sun7i_spdma.ko

    insmod /system/vendor/modules/sndspdif.ko

    insmod /system/vendor/modules/sun7i_sndspdif.ko
```

10.3.3 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。

10.4支持开机视频

将媒体文件放在/system/media/目录下，并且命名为 boot.mp4 即可。

11 常用的调试方法

11.1 提高内核打印等级

修改 lichee/tools/pack_brandy/chips/sun7i/configs/android/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！

11.2 将 logcat 和 dmesg 信息保存到文件系统

为了调试方便，可以在开发调试阶段将系统的 logcat 和内核 dmesg 自动打印到 data 分区文件系统上保存，这种方法可以方便调试偶发问题，可以在 android/device/softwinner/sugar-xxx/init.sun7i.rc 中使能下面语句：

```
service logger_kernel /system/bin/logger.sh kernel
        class main
        user root

service logger_android /system/bin/logger.sh android
        class main
        user root
```

但注意对外发布的 release 固件一定要注释掉这些打印，因为后台的打印信息会占用 data 区存储空间！

11.3 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
```

11.4使用网络 adb 调试

如果需要使用网络 adb 调试，可以在 android/devices/softwinner/suga-ref001/init.sun7i.rc 添加下面的语句：

```
on boot

setprop service.adb.tcp.port 5555

stop adbd

start adbd
```

11.5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 sugar-ref001

$ 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/wing-xxx/data/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/

12 Declaration