全 志 科 技
Allwinner Technology

Mars4.4.2 定制化说明

V1.0

2014-04-28

Revision History

Version	Date	Changes compared to previous issue
v1.0	2014-04-28	建立初始版本

目 录

1. 代码说明

1.1. 代码下载

请参考 SDK 发布文档的下载说明

注意：Android4.2.2 开始 A31 和 A31s SDK 合并，请注意其中差异

1.2. 公版方案说明

方案名称	硬件参数	备注
mars-a31s512m	512M dram,8G 存储	A31s 512M 方案
mars-a31s	1G dram 8G 存储	A31s 1G 方案
mars_ml220	2G dram,8G 存储	A31 2G 方案

1.3. 公版代码编译

编译 lichee 代码

```
$cd lichee
$./build.sh -p sun6i_fiber_a31s
```

编译成功会打出

```
####################################
#              compile success      #
####################################
```

编译 android 代码

```
$cd android
$ source ./build/envsetup.sh
$lunch   mars_a31s-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：

```
dragon image.cfg sys_partition.fex [OK]
----------image is at----------

/home/yuguoxu/sugar/lichee/tools/pack/sun6i_android_mars-a31s512m.img

pack finish
```


2. 方案配置说明

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-3.3
lichee/tools/pack/chips/sun6i/configs/android
android/device/softwinner
```

2.1. lichee/linux-3.3

kernel 部分一般无需配置，但如果有需求要打开内核某些 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-3.3/arch/arm/configs/sun6ismp_fiber_android_defconfig
```

注意：sun6ismp_fiber_a31s_defconfig/sun6ismp_fiber_a31s512m_defconfig/sun6ismp_fiber_defconfig 三个配置已经合并为一个，即 sun6ismp_fiber_android_defconfig，在 lichee 下编译，比如：

```
./build.sh -p sun6i_fiber
./build.sh -p sun6i_fiber_a31s
./build.sh -p sun6i_fiber_a31s512m
```

默认都会复制 linux-3.3/arch/arm/configs/sun6ismp_fiber_android_deconfig 到 linux-3.3/.config 作为编译的配置；如果 linux-3.3/.config 存在的话，则不覆盖。

2.2. lichee/tools/pack/chips/sun6i/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（mars-a31s512m/ mars-a31s/ mars-ml220）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。

2.3. 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 lichee 配置

拷贝一份通用的配置，如 lichee/tools/pack/chips/sun6i/configs/android/mars-a31s512m 为 lichee/tools/pack/chips/sun6i/configs/android/mars-xxx，然后按照实际的硬件电路进行配置修改，配置的方法见《A31(s)_fiber_fex_guide_v2.0.pdf》及《A31-A31s sys_config.fex DDR 配置使用指南.pdf》

通常对于盒子产品，定制化配置主要在存储介质分区定制和遥控器键码定制上。

3.1. 分区配置说明

以 mars-a31s512m 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	Bootloader 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
data	1024M	Android Data 分区，用于安装应用、应用数据等
misc	16M	Misc 分区，用于写入 BCB（Bootloader Cmd Block）进入 recovery
recovery	32M	Android Recovery 分区，用于 Android Recovery 系统
sysrecovery	768M	固件备份分区，用于一键恢复功能
cache	512M	Android Cache 分区，用于 Recovery 系统存放 OTA 固件等
private	16M	存放厂商序列号等私有数据（私有分区）
databk	128M	Databk 分区
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

3.2. 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《sys_partition.fex 各配置属性的含义和使用.doc》

如需要增加一个名为 key 的私有分区，可在 sys_partition.fex 中 UDISK 分区前添加：

```
;------------------------->nandk, private partition
[partition]
    name         = key
    size         = 32768
    keydata      = 1
    user_type    = 0xC000
```

注意 keydata 必须保持为 1。

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/by-name/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。详见私有分区读写管理章节

4. 添加定制的方案版 android 配置

4.1. 添加方案

在 android/device/softwinner 目录下提供了三个方案供参考：

mars-a31s512m : A31s DRAM 512M nandflash/eMMC 8G 方案

mars-a31s : A31s DRAM 1G nandflash/eMMC 8G 方案

mars-ml220 : A31 DRAM 2G nandflash/eMMC 8G 方案

注：所有的方案都是存储介质 nandflash、eMMC 兼容的，即一个固件可烧写 NandFlash、eMMC 的机器

如添加一款 512M 的方案，可将 mars-a31s512m 拷贝一份为 mars-xxx，然后进入 mars-xxx 目录进行修改，具体的操作如下：

```
$ cd android/device/softwinner/
$ cp -r mars-a31s512m mars-xxx
$ cd mars-xxx/
$ rm -rf .git modules kernel                    #删除原有的 git modules 目录和内核
$ mv mars_a31s512m.mk    mars_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/A31S512M/XXX/g'   #替换掉 A31S512M 的字符串
$ find . -type f | xargs sed -i 's/a31s512m/xxx/g'     #替换掉 a31s512m 的字符串
$ git diff
$ git add –u
$ git commit -m "init first versioion"              #第一次提交
```

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 mars_xxx-eng
```

4.2. 修改方案资源

4.2.1. 修改 bootlogo

替换 lichee/tools/pack/chips/sun6i/configs/android/mars-xxx/ bootlogo.bmp 文件

4.2.2. 修改 initlogo

替换 android/device/softwinner/mars-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

4.2.3. 修改开机动画

修改开机动画可以替换掉 android/device/softwinner/mars-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

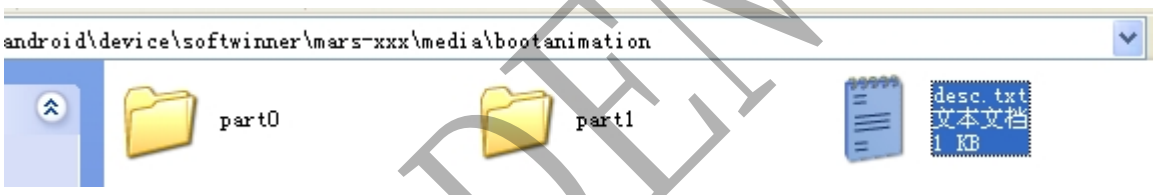
```
400 409 16
p 1 0 part0
p 0 0 part1
```

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；

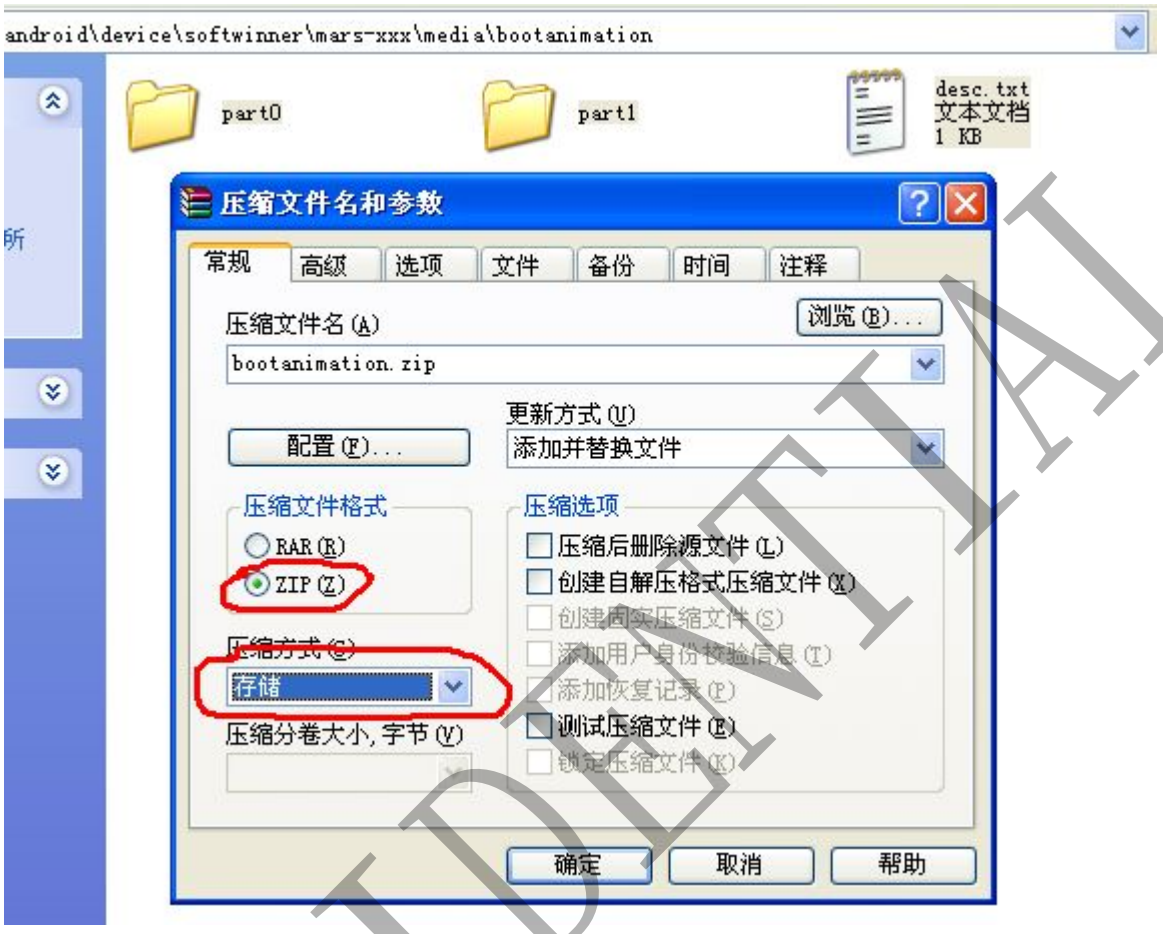
p 0 0 part1---这里的 p 代表标志符，0 代表本阶段无限循环，0 代表阶段间隔时间为 0，part1 代表对应的文件夹名，为第二阶段动画图片目录；

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

4.2.4. 修改开机音乐

替换掉 android/device/softwinner/mars-xxx/media/boot.wav 音乐资源

4.2.5. 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/mars-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

4.2.6. 替换默认壁纸

更换默认墙纸替换方案 overlay 文件：

android/device/softwinner/mars-xxx/overlay/frameworks/base/core/core/res/drawable-swxxxdp-nodpi/default_wallpaper.jpg

4.2.7. 默认字体大小设置

系统字体由 fontScale 来控制缩放，设置菜单中的小，普通，大，超大分别对应的 fontScale 为 0.85、1.0、1.15、1.3，修改默认字体大小可在 android/device/softwinner/fiber-xxx.mk 中设置 ro.font.scale 的值来设置默认的字体大小。如设置为大字体为：

```
PRODUCT_PROPERTY_OVERRIDES += \
       ro.font.scale=1.15 \
```

注：建议采用默认设置，即 ro.font.scale=1.0

4.3. 方案目录内文件说明

4.3.1.mars-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，需要集成到固件中的配置文件，产品名字，厂商名称等等，通常把这个文件名改为自己的方案名,如:mars-xxx.mk

mk 文件中常用的命令及意义如下：

4.3.1.1. 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，TVD 方案通过该命令打包进系统的应用有 TvdSettings,TvdLauncher,TvdVideo,TvdFileManager.

4.3.1.2. 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 mars-xxx 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,可以如下写：

```
PRODUCT_COPY_FILES += \

device/softwinner/mars-xxx/sun6i-ir.kl:system/usr/keylayout/sun6i-ir.kl \
```


4.3.1.3. PRODUCT_PROPERTY_OVERRIDES

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。

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
```

4.3.2.AndroidProducts.mk

这里只有一句话

```
PRODUCT_MAKEFILES := \
    $(LOCAL_DIR)/mars-xxx.mk
```

其中 mars-xxx.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.

4.3.3.BoardConfig.mk

这里一般定义了 Wifi 和其他一些的配置变量，Wifi 的配置变量值可参考该文档的“Wifi 配置”小节。

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.

4.3.4.init.sun6i.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.sun6i.rc 中加载

如果需要使用网络 adb 调试,可以在其中添加下面的语句

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

4.3.5.initlogo.rle

自定义开机第二个 logo 图片.可以使用 lichee\tools\tools_win\ LogoGen.zip 小工具生成

4.3.6. 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

4.3.7.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.

4.3.8.configs/sun6i-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

4.3.9.vendorsetup.sh

通常其内容为：

```
#!/bin/bash

add_lunch_combo mars_xxx-eng
```

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

```
#!/bin/bash

add_lunch_combo mars_xxx-user
```

编译时“lunch”会看到 mars_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，user 版本默认不打开 adb 并且不开放 root 权限，

4.3.10. package.sh

打包时会被调用的脚本,内容如下

```
cd $PACKAGE

DEBUG="uart0"

SIGMODE="none"
```



```
if [ "$1" = "-d" -o "$2" = "-d" ]; then
    echo "--------debug version, have uart printf-------------"
    DEBUG="card0";
else
    echo "--------release version, donnot have uart printf-------------"
fi
if [ "$1" = "-s" -o "$2" = "-s" ]; then
    echo "--------------------sig version--------------------"
    SIGMODE="sig";
fi
    ./pack -c sun6i -p android -b mars-xxx -d ${DEBUG} -s ${SIGMODE}


cd -
```

其中的 mars-xx 为自己方案的名称,对应 lichee/tools/pack/chips/sun6i/configs/android 下的方案板级配置目录

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。

5. 预留内存配置

5.1. 各方案的内存预留配置

<table>
<tr><td rowspan="2">A31(s) Homlet 方案
内存预留配置
(mars-44 android4.4.2)</td><td colspan="2">物理内存配置</td><td>虚拟内存配置</td></tr>
<tr><td>GPU 预留(显示)</td><td>DE/VE/CSI/MP 预留(多媒体)</td><td rowspan="2">vmalloc</td></tr>
<tr><td>CONFIG_ION_SUNXI_CARVEOUT_SIZE_512M</td><td>CONFIG_SUNXI_MEMORY_RESERVED_SIZE_512M</td></tr>
<tr><td rowspan="3">A31(s) 512M 方案</td><td rowspan="3">26M</td><td>64M(2D 1080P)</td><td rowspan="3">默认488M</td></tr>
<tr><td>72M(3D 1080P/2160P)</td></tr>
<tr><td>90M(蓝光 ISO，参考帧3) 默认</td></tr>
<tr><td rowspan="2">A31(s) 1G/2G 方案</td><td>CONFIG_ION_SUNXI_CARVEOUT_SIZE_1G</td><td>CONFIG_SUNXI_MEMORY_RESERVED_SIZE_1G</td><td></td></tr>
<tr><td>32M</td><td>122M(蓝光 ISO) 默认</td><td>384M</td></tr>
</table>

说明：

1. kernel 会根据系统实际物理内存的大小，自适应设置上述预留内存；
   在 linux-3.3/arch/arm/mach-sun6i/core.c 根据实际检测到的物理内存配置。

2. 1G/2G 方案共预留 154M 即可应付绝大多数蓝光视频播放和大型游戏内存需求，已做大量测试，不建议修改，如确实不够，则跟进实际情况增大 CONFIG_SUNXI_MEMORY_RESERVED_SIZE_1G 即可。

   修改方法:

   方法 1：直接修改 linux-3.3/arch/arm/configs/sun6ismp_fiber_android_defconfig 中的 CONFIG_SUNXI_MEMORY_RESERVED_SIZE_1G，然后删除 linux-3.3/.config 文件，重新编译内核。

   方法 2：在方案配置 mars-xxx/BoardConfig.mk 中加入，比如修改为 128Mbytes：

```
BOARD_KERNEL_CMDLINE += hw_reserve=128M
```

   其中，方法 2 不需要重新编译内核，但仍然需要重新编译 android，生成固件烧写后，可通过内核节点 cat /proc/cmdline 确认是否存在 hw_reserve 的关键字，否则不生效需要 make installclean 确保编译有效后再重新编译 android。

3. 512M 方案由于内存的限制，务必根据多媒体播放**规格**来调整预留内存的大小，不然造成浪费的同时影响运行体验，多媒体规格只需要调整 CONFIG_SUNXI_MEMORY_RESERVED_SIZE_512M 的默认值。

   3.1 如果只需要播放 2D 1080P 规格则修改为 64M，即可满足大部分网络视频的内存需求；

   3.2 如果只需要播放 3D 1080P 或者 2160P 则修改为 72M；

   3.3 如果需要支持蓝光 ISO 格式的视频，则需要修改为 90M。

修改方法:

方法 1：直接修改 linux-3.3/arch/arm/configs/sun6ismp_fiber_android_defconfig 中的 CONFIG_SUNXI_MEMORY_RESERVED_SIZE_512M，然后删除 linux-3.3/.config 文件，重新编译内核。

方法 2：在方案配置 mars-xxx/BoardConfig.mk 中加入，比如修改为 64Mbytes：

```
BOARD_KERNEL_CMDLINE += hw_reserve=64M
```

其中，方法 2 不需要重新编译内核，但仍然需要重新编译 android，生成固件烧写后，可通过内核节点 cat /proc/cmdline 确认是否存在 hw_reserve 的关键字，否则不生效需要 make cleaninstall 再重新编译 android。

4. 禁止修改 GPU 预留内存，即 linux-3.3/arch/arm/configs/sun6ismp_fiber_android_defconfig 中的 CONFIG_ION_SUNXI_CARVEOUT_SIZE_512M/CONFIG_ION_SUNXI_CARVEOUT_SIZE_1G 不作修改；实际测试已经满足绝大多数场景需求，即使 GPU 预留内存 ION 不足，系统也会从多媒体预留的内存申请，故无需修改 GPU 预留内存大小。

5.2. 512m 方案内存优化 Android 配置

参考方案配置 mars-a31s512m，即 device/softwinner/mars-a31s512m/文件夹。
涉及 mars_a31s512m.mk、fstab.sun6i 和 init.sun6i.rc 三个文件的配置。

5.2.1. mars_a31s512m.mk

1. Dalvik 虚拟机内存调整

默认使用$(call inherit-product, frameworks/native/build/tablet-dalvik-heap.mk)虚拟机配置，更好的兼容 apk，已经修改了下面两项配置：

```
dalvik.vm.heapgrowthlimit=72m
dalvik.vm.heapsize=256m
```

2. 新增下面三个配置

```
PRODUCT_PROPERTY_OVERRIDES += \
 ro.config.low_ram=true \
 ro.config.max_starting_bg=3 \
 ro.am.max_recent_tasks=4 \
```

其中， ro.config.low_ram=true 是 android4.4.2 引入的内存优化配置。
ro.config.max_starting_bg=3 是针对 512m 方案并行化应用启动。
ro.am.max_recent_tasks=4 是指最近后台最多的任务数。

5.2.2. fstab.sun6i

1. 加入 zram disk 大小配置，默认 180M

在 fstab.sun6i 最后一行加入

```
/dev/block/zram0 none swap defaults zramsize=188743680
```

5.2.3. init.sun6i.rc

加入下面红色字段：

```
# swapout to zram
    write /proc/sys/vm/page-cluster 0

on early-fs
    dispe2fsck
    mount_all /fstab.sun6i
    swapon_all /fstab.sun6i
    setprop ro.crypto.fuse_sdcard true
```


6. 配置方案遥控器

6.1. 修改红外遥控的地址码：

在文件 lichee\tools\pack\chips\sun6i\configs\android\mars-xxx 中，根据自己的遥控器的地址码修改如下红色字体配置：

```
[ir_para]
ir_used                 = 1
ir_rx                   = port:PL04<2><1><default><default>
ir_power_key_code       = 0x57
ir_addr_code                    = 0x9f00
```

譬如说，如果地址码为 0x9f00，则修改成

ir_addr_code = 0x9f00

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 = 0x009f

6.2. 修改按键映射

在文件 android4.4\device\softwinner\mars-xxx\sun6i-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以下方式确定：

2) 在串口或 adb shell 中输入 getevent,然后点击按键；

3) 将打印出来的键值转换成十进制，此十进制的值即为按键对应的扫描码。

注意：扫描码不能重复，否则此文件将失效。

6.3. 修改“软鼠标模式”下使用的键值

7. 添加预编译 APK

7.1. 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 android4.1\device\softwinner\fiber-common\fiber-common.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.

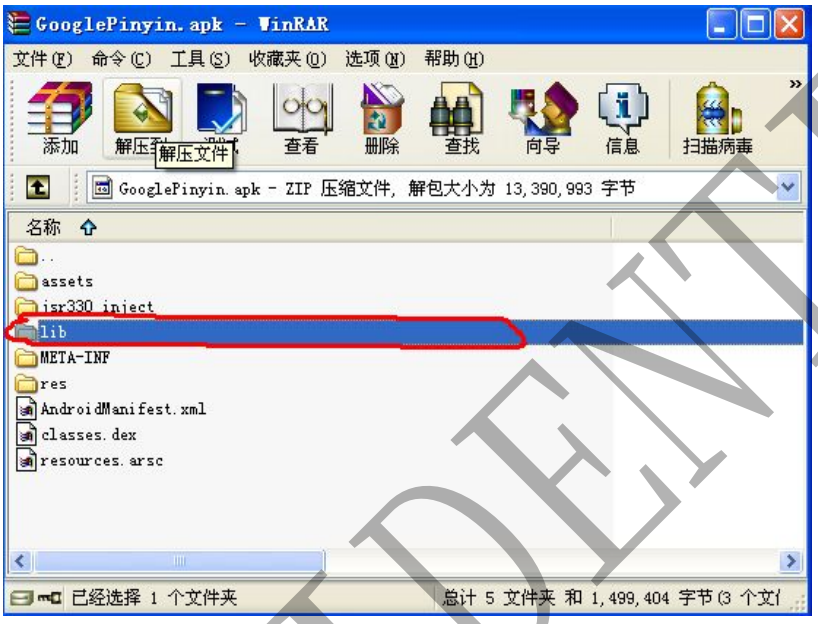
7.2. system 预装

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：

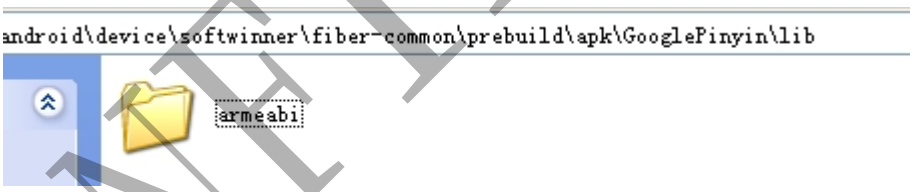
下面以 Google 拼音的预装为例：

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/fiber-common/prebuild/apk 目录

   解压出来的 APK JNI 库拷贝到 android/device/softwinner/fiber-common/prebuild/apklib 目录

   修改 android/device/softwinner/fiber-common/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/device/softwinner/fiber-common/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 fiber-common.mk 或 mars-xxx.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

7.3. 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

在 android/device/softwinner/fiber-common/prebuild/preinstallapk/Android.mk 中添加：

```
LOCAL_PATH := $(call my-dir)
#########################
include $(CLEAR_VARS)
LOCAL_MODULE := AllCast
LOCAL_MODULE_TAGS := optional
LOCAL_CERTIFICATE := PRESIGNED
LOCAL_MODULE_PATH := $(TARGET_OUT)/preinstall
LOCAL_MODULE_CLASS := APPS
LOCAL_SRC_FILES := $(LOCAL_MODULE).apk
```



```
include $(BUILD_PREBUILT)
```

2. 引用 APK 包

在 fiber-common.mk 或 mars-xxx.mk 中加入：

```
PRODUCT_PACKAGES += \
    AllCast
```

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. 添加 GMS 框架

系统默认不预装 GMS 框架，如果有需求，可以打开，修改如下位置代码：

android/device/softwinner/mars-xxx/ mars_xxx.mk

```
#include device/softwinner/fiber-common/prebuild/google/products/gms.mk
```

去掉前面的注释，重新编译，便可内置 GMS 服务

9. 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。

9.1. 配置默认 launcher

在文件 android/device/softwinner/mars-xxx/mars-xxx.mk 中，在变量“PRODUCT_PROPERTY_OVERRIDES”中增加两个配置项 ro.sw.defaultlauncherpackage 和 ro.sw.defaultlauncherclass。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.defaultlauncherclass=com.softwinner.launcher.Launcher
```

9.2. 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。

10. 全屏显示

10.1. 永久隐藏导航栏

可以通过配置系统属性来实现永久隐藏导航栏。如果系统属性“ro.statusbar.alwayshide”的值为“true”，那么导航栏总是隐藏。此属性值默认为“false”。

10.2. 应用主动隐藏导航栏

如果没有按照 1.6.1 的方法设置永久隐藏导航栏，应用也可以去主动的隐藏导航栏。

应用隐藏导航栏的方法可以使用 Android4.4 新引入的沉浸模式（IMMERSIVE MODE)。

代码例子：

```
private void hideSystemUI() {
    View mDecorView=getWindow().getDecorView();
    mDecorView.setSystemUiVisibility(
           View.SYSTEM_UI_FLAG_LAYOUT_STABLE
           | View.SYSTEM_UI_FLAG_LAYOUT_HIDE_NAVIGATION
           | View.SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN
           | View.SYSTEM_UI_FLAG_HIDE_NAVIGATION // hide nav bar
           | View.SYSTEM_UI_FLAG_FULLSCREEN // hide status bar
           | View.SYSTEM_UI_FLAG_IMMERSIVE_STICKY);
}
```

关于应用全屏、隐藏导航栏的详细信息，请参考官方文档 https://developer.android.com/index.html 中关于 Full-Screen，Immersive 和 View 类关于 SYSTEM_UI 相关 flag 的定义。

11. 配置关机模式

11.1. 关机时不再出现对话框

在文件 device\softwinner\xxx\xxx.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”。

11.2. 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\xxx\xxx.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”。

12. 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/mars-xxx/mars-xxx.mk 文件中，在变量“PRODUCT_PROPERTY_OVERRIDES”中增加一个新的配置项：

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

13. 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/mars-xxx/mars-xxx.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。

14. 音频动态管理

14.1. 音频动态管理概要

音频动态管理机制主要有三个特点：

1.音频支持多通道同时输出；

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。

14.2. 音频管理策略

目前公版对于音频动态管理机制的策略如下：

a.上电后，一开始默认 HDMI 输出；

b.在系统完成 BOOT 后注册监听 USB 音频设备和耳机的热插拔信息，代码位于 android\frameworks\base\services\java\com\android\server\AudioDeviceManagerObserver.java 和 android\frameworks\base\services\java\com\android\server\AudioManagerPolicy.java。

目前公版的做法是：

1)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；

开机之后如果 USB 音频输入设备插入,则自动切到该设备;

如果拔出 USB 音频输入设备,则自动切换到默认的 codec

2)对于输出：

有两种策略:

1.自动切换(默认):

(1)如果有 USB 耳机

开机后,如果是插着 USB 耳机,自动切换到 USB 耳机输出;否则,根据当前使用的显示设备决定使用对应的音频输出.

系统起来后,拔出 USB 耳机,自动切换到与当前显示设备对应的音频输出设备.

插入 USB 耳机,自动切换到该 USB 耳机输出.

(2)如果没有 USB 耳机

开机后,切换到与当前显示设备对应的音频输出设备,否则默认以 codec 输出.

(3)HDMI 线和 av 线插拔时,音频设备输出模式变为对应的设备.

2.用户选择优先:

(1)如果有 USB 耳机

开机后,如果是插着 USB 耳机,自动切换到 USB 耳机输出;否则,根据设置里上次用户选择的音频设备输出(第一次刷机时有默认值);

系统起来后,拔出 USB 耳机,自动切换到设置里保存的输出设备;

插入 USB 耳机,自动切换到该 USB 耳机输出.

(2)如果没有 USB 耳机

开机后,依据设置里上次用户选择的音频设备输出(第一次刷机时有默认值);

(3)HDMI 线和 av 线插拔时,音频设备输出模式不改变.

3.这两种策略可以通过修改数据库的值来设置,默认策略可修改文件:

android\device\softwinner\mars-ml220\overlay\frameworks\base\packages\SettingsProvider\res\values\defaults.xml 中的 def_audio_manage_policy 的值,其中可设置为 0 或 1,分别表示如下:

0：自动切换模式

1：用户选择优先模式每次有新的 USB 输入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。

c.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)，代码位于 android\device\softwinner\fiber-common\prebuild\packages\TvdSettings\src\com\android\settings\SoundSettings.java 和 AudioChannelsSelect.java。

14.3. 上层提供的接口

如需自定义音频策略，可使用 frameworks\base\media\java\android\media\AudioManager.java 提供的接口。

```
/* 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",... */
/* define audio device name */
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);
```

14.4. 音频接口使用例子

可以参考 android\frameworks\base\services\java\com\android\server\AudioManagerPolicy.java 中的代码。

14.4.1. 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);
mCtx.registerReceiver(this, filter);

//在AudioManagerPolicy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);
final String extra = bundle.getString(AudioDeviceManagerObserver.EXTRA_MNG);
//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；
//extra是设备的类型（如耳机是“AUDIO_H2W”）；
}
```


14.4.2. 音频输出/输入模式

```
//一、设置音频输出设备

mAudioManager=(AudioManager)context.getSystemService(Context.AUDIO_SERVICE);

ArrayList<String> lst = new ArrayList<String>();

//1. 设置HDMI音频输出

lst.add(AudioManager.AUDIO_NAME_HDMI);

mAudioManager.setAudioDeviceActive(lst, AudioManager.AUDIO_OUTPUT_ACTIVE);

//2. 设置SPDIF音频输出

lst.clear();

lst.add(AudioManager.AUDIO_NAME_SPDIF);

mAudioManager.setAudioDeviceActive(lst, AudioManager.AUDIO_OUTPUT_ACTIVE);

//3. 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);

//4. 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


//二、设置音频输入设备

//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);


//三、获取当前可用的输入设备列表

lst = mAudioManager.getAudioDevices(AudioManager.AUDIO_INPUT_TYPE);


//四、获取当前可用的输出设备列表

lst = mAudioManager.getAudioDevices(AudioManager.AUDIO_OUTPUT_TYPE);
```


15. 支持 spdif 接口

15.1. 使能 spdif 模块

在配置文件 lichee\tools\pack\chips\sun6i\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。

15.2. 安装 spdif 驱动

修改 android4.4\device\softwinner\<方案>\init.sun46.rc 文件,在其中加入加载 spdif 驱动的脚本.

```
#spdif

    insmod /system/vendor/modules/sun6i_spdif.ko

    insmod /system/vendor/modules/sun6i_spdma.ko

    insmod /system/vendor/modules/sndspdif.ko

    insmod /system/vendor/modules/sun6i_sndspdif.ko
```

15.3. 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

16. 配置流媒体缓冲策略

Android/frameworks/base/media/CedarX-Projects/CedarXAndroid/IceCreamSandwich/CedarXPlayer.cpp 文件中，提供以下四个参数供调整流媒体播放缓冲策略使用：

- SMOOTH_PLAY_TIME
- DEFAULT_CACHE_HIGH_THRESHOLD
- DEFAULT_CACHE_LOW_THRESHOLD
- ALWAYS_USE_DEFAULT_CACHE_THRESHOLD_VALUE

定义以上四个宏为不同值，CedarX 播放器将采用不同的数据缓冲策略。

参数“SMOOTH_PLAY_TIME”表示播放过程中，缓冲一次可平滑播放的时间，单位为秒；例如当前网速为 BW，当前视频码率为 BR，当数据量小于 L 时会决定暂停缓冲，则视频暂停缓冲时需下载的数据量 H = (BR-BW)*SMOOTH_PLAY_TIME + L;

参数“DEFAULT_CACHE_HIGH_THRESHOLD”表示启动播放时（第一次播放时，网速和码率未估计出来），需要缓冲的数据量，该参数影响进入播放后视频画面显示出来的速度；该参数单位为字节。

当播放过程中，缓冲数据量小于参数“DEFAULT_CACHE_LOW_THRESHOLD”时，播放器会决定暂停以缓冲数据。该参数单位为字节；

当参数“ALWAYS_USE_DEFAULT_CACHE_THRESHOLD_VALUE”被设为 1，播放过程中决定暂停或者启动播放的数据阈值始终采用“DEFAULT_CACHE_LOW_THRESHOLD”和“DEFAULT_CACHE_HIGH_THRESHOLD”，否则，决定启动播放的数据阈值根据网速、视频码率和“SMOOTH_PLAY_TIME”计算得出。

1.1 推荐设置

1. DEFAULT_CACHE_LOW_THRESHOLD 一定不能小于 48KB，推荐设置为 64KB

2. DEFAULT_CACHE_HIGH_THRESHOLD 设置太高会使视频播放启动速度变慢，切台时间长；设置太低会使视频刚播放出来就进入暂停，等待重新缓冲数据，实际使用中，根据具体的网络带宽设置为 192~512KB 之间；

3. SMOOTH_PLAY_TIME 会影响数据缓冲量，尤其当网速和视频码率相差较大时；为防止缓冲数据量太大，播放器内部已限制最大缓冲数据量为 4MB；

17. SystemMix 可扩展接口说明

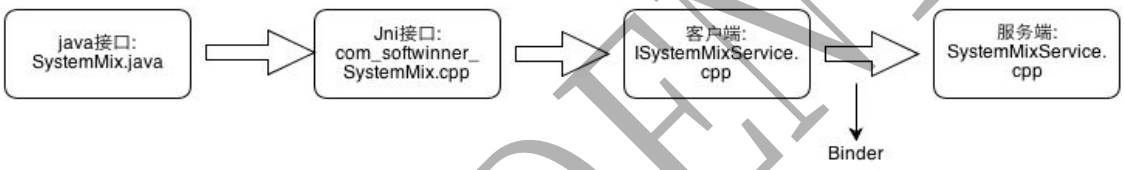
在 android/frameworks/base/swextend/systemmix 目录,我们提供了一套用于访问底层高权限信息的接口,客户可参照里面的做法来扩展该 SystemMix 类的功能

1.2 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.systemmix 机制给 apk 以 root 权限去访问这些信息.

1.3 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:



1.4 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.

18. 如何控制 GPIO

18.1. 配置 GPIO

在 lichee\tools\pack\chips\sun6i\configs\fiber\xxx\sys_config1.fex 文件中，添加类似如下的配置信息：

```
;gpio configuration
;------------------------------------------------------------
 [gpio_para]
gpio_used               = 1
gpio_pin_1              = port:PH10<1><default><default><0>
gpio_pin_2              = port:PH20<1><default><default><1>
gpio_pin_3              = port:PB03<0><default>
```

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。

18.2. 配置 boot 阶段初始化的 gpio

在配置文件 sys_config.fex 中配置[gpio_init]参数。

范例：

```
[gpio_init]
pin_1                   = port:PH10<1><default><default><0>
pin_2                   = port:PH20<1><default><default><1>
```

以上配置表示：在 boot 阶段，设置 ph10 输出低电平，设置 ph20 输出高电平。

18.3. 控制 GPIO 的接口

18.3.1. java 层的接口

java 控制 GPIO 的接口定义在文件 Gpio.java 中，其路径为 android4.1\frameworks\base\swextend\gpio\java\Gpio.java。

18.3.2. 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

如图：

```
# pwd
/sys/class/gpio_sw
# ls
PH1    PH11   PH2
```

```
# pwd
/sys/class/gpio_sw/PH1
# ls
data       drv_level  power      subsystem  uevent
device     mul_sel    pull       trigger
```

在 GPIO 节点的子目录中可以看到 data ,drv_level, mul_sel,pull 等我们可以对 data ,drv_level, mul_sel,pull 这 4 个文件进行读写操作来控制 GPIO 口。

mul_sel： 值代表端口目前的功能

pull： 值代表端口目前的电阻状态

drv_level： 值代表端口目前的驱动等级

data： 值代表端口目前的电平状态

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 android4.1\frameworks\base\swextend\gpio\libgpio\GpioService.cpp 中的代码。

19. OTA 升级说明

19.1. 升级注意事项

19.1.1. 分区大小

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。

19.1.2. Cache 分区大小

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。

19.1.3. Misc 分区有足够权限被读写

Misc 分区 Recovery 与 Android 之间的桥梁，如果 Misc 分区的权限过高，导致上层应用无法对其写入数据，则会令 Recovery 功能异常。请确保 misc 分区的设备节点/dev/block/xxxxx 和其软连接/dev/block/misc 有足够的权限被读写。

19.2. 制作更新包

19.2.1. 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/mars-xxx/obj/PACKAGING/target_files_intermediates/)中生成 mars_xxx-ota-xxx.zip，该文件就是完整更新包。

19.2.2. 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 根目录执行以下命名：

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

在对应 out 目录下 out/target/product/mars-xxx/obj/PACKAGING/target_files_intermediates/，生成命名为 mars_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。之后执行以下命名将可以生成差分包：

```
$make otapackage_inc
```

将在 out/target/product/mars-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。

19.3. 专有功能

19.3.1. Usb-Recovery

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。

相关配置如下：

打开配置文件 lichee/tools/pack/chips/sun6i/configs/android/mars-xxx/sys_config.fex：

```
[system]
;recovery_key = port:PH9<0><default><default><default>
anrecovery_key = port:PH9<0><default><default><default>
```

将 anrecovery_key 键配置成 usb-recovery 键对应的的 gpio 口即可。

19.3.2. 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 android/frameworks/base/swextend/os/java/com/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。

19.4. 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。

19.4.1. 编译开关

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

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

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。

19.4.2. 按键配置

目前支持四个按键，分别是“向上”、“向下”、“确定”和“HOME”键。“HOME”键可以隐藏/显示 Recovery 控制界面。

在 android/device/softwinner/mars-xxx/recovery/ir_keycode.cpp 修改以下宏定义：

```
3 #define IR_KEYCODE_HOME 71
4 #define IR_KEYCODE_UP 67
5 #define IR_KEYCODE_DOWN 10
6 #define IR_KEYCODE_ENTER 2
```

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

19.5. 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 mars 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。

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	√(v1.0 中支持 Boot0 升级，但没有开启此功能,v1.2 中已开启)
TSD/EMMC 方案 Boot0/Boot1(Uboot)升级	√(v1.2 以上)
Uboot 升级	√
sys_config.fex 更新	√
sys_partition.fex 更新	×

值得注意的是，BSP 中大部分关于电路的配置都集中在 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。

20. 一键恢复功能配置

20.1. 原理

在量产时，会将整个固件包文件（*.img）烧录到一个单独的分区（sysrecovery） 。在启动阶段通过特定的 GPIO 来触发系统进入恢复流程，将整个系统重新烧录一遍。

20.2. 目的

在系统无法正常启动或出现其他异常时，可以通过一种比较简单的方法来挽救系统。

20.3. 操作步骤

1）系统断电
2）按住“recovery”键
3）上电，3 秒钟左右，会有 led 闪烁，此时释放“recovery”键。

20.4. 配置

1） 按键配置

在 lichee\tools\pack\chips\sun6i\configs\android\xxx\sys_config.fex 文件中，根据自己方案中“recovery”键使用的 GPIO 来添加类似如下的配置信息：

```
[system]
recovery_key = port:PH16<0><1>
```

2） 配置分区

在 lichee\tools\pack_v1\chips\sun6i\configs\android\xxx\sys_partiton.fex 文件添加分区，分区大小按需分配：

```
[partition]
    name         = sysrecovery
    size         = 1572864
    downloadfile = "sysrecovery.fex"
    user_type    = 0xC000
    verify       = 0
```

20.5. 注意事项

1. 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。

20.6. “一键恢复”失败的可能原因

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

21. IR 唤醒系统功能配置

21.1. 开启 IR 功能

A31 android4.2 v0.9 版本之后增加了”通过 IR power 键唤醒系统的功能”,可根据自己方案配置是否需要使用该功能.

配置如下:

(1)android/devices/softwinner/mars-xxx/configs/sun6i-ir.kl 文件中定义 IR 的按键映射关系,必须确保 POWER 键有对应的映射关系。

(2)lichee/tools/pack/chips/sun6i/configs/android/mars-xxx/sys_config.fex 文件中,

[wakeup_src_para]

wakeup_event = 0x200

--> 该值为 0x200 时配置 IR 为唤醒源,如果不填或其他值,则没有该 IR 唤醒系统的功能.

[ir_para]

ir_power_key_code = 0x57

--> 该值为 IR 的唤醒键,一般配置为 power 键的按键码,在上面"wakeup_event = 0x200"时,按下该按键可以唤醒系统.如果没配置则该 IR 唤醒系统功能不可用.

22. 私有分区的读写管理

22.1. 软件层配置

修改 android4.4\device\softwinner\mars-xxx 下的 init.sun6i.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/private /mnt/private   #挂载private分区

    chmod 777 /dev/block/by-name/private #修改private块设备权限
```

22.2. 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”).

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)
```


23. 开发中常用的调试方法

23.1. 提高内核打印等级

修改 android/device/softwinner/fiber-common/ BoardConfigCommon.mk

BOARD_KERNEL_CMDLINE := console=ttyS0,115200 rw init=/init loglevel=8

但注意对外发布的固件一定要把 loglevel 改回 4，否则会影响系统速度！

23.2. 记录 logcat 到文件系统

如果为了调试方便，可以在开发中将系统的 logcat 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/mars-xxx/init.sun6i.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存储空间！

23.3. fastboot 烧写

修改内核文件、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，常用命令如下：

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


24. Declaration

This document is the original work and copyrighted property of Allwinner Technology ("Allwinner"). Reproduction in whole or in part must obtain the written approval of Allwinner and give clear acknowledgement to the copyright owner.

The information furnished by Allwinner is believed to be accurate and reliable. Allwinner reserves the right to make changes in circuit design and/or specifications at any time without notice. Allwinner does not assume any responsibility and liability for its use. Nor for any infringements of patents or other rights of the third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of Allwinner. This datasheet neither states nor implies warranty of any kind, including fitness for any particular application.

