全 志 科 技
Allwinner Technology

A20_SugarV2.1 固件编译打包及烧录

V1.0

(Only for A20 Sugar sdkV2.1)

2014-06-18

Revision History

Version	Date	Changes compared to previous issue
V1.0	2014-06-18	初建版本

目录

1. 概述

1.1. 编写目的

该文档的目的在于，简要介绍 A20 sugar　v2.1 sdk 中 Boot1.0 和 Boot2.0 共存时的编译方法。同时支持老的 boot1.0 和新的 boot2.0 的目的在于方便使用 boot1.0 的老旧方案更新 sdk；新的方案务必使用 Boot2.0。

1.2. 适用范围

A20 sugar sdkv2.1。

1.3. 相关人员

预期读者为客户案支持人员和客户。

2. 前言

为了实现 boot1.0、nand1.0 到 boot2.0、nand2.0 的过渡，对 boot1.0、nand1.0 代码和 boot2.0、 nand2.0 代码进行统一管理，采用不同的编译方式实现对 boot1.0、boot2.0 的使用(boot1.0 绑定 nand1.0, boot2.0 绑定 nand2.0)。

对于使用 boot2.0 的用户无须关注 boot1.0 的相关信息。

编译方法说明：
1. Android 部分的编译跟 A20 sugar v2.0 方法一样，没有做任何的改动；
2. Lichee 部分的编译默认为 boot2.0(方法跟 A20 sugar v2.0 一样)，如果想使用 boot1.0，则需增加-v boot1.0 参数。

3. lichee 目录结构

A20 sugar sdkv2.1 开始，lichee 的目录结构发生了变化，但 android 是没有变化的。

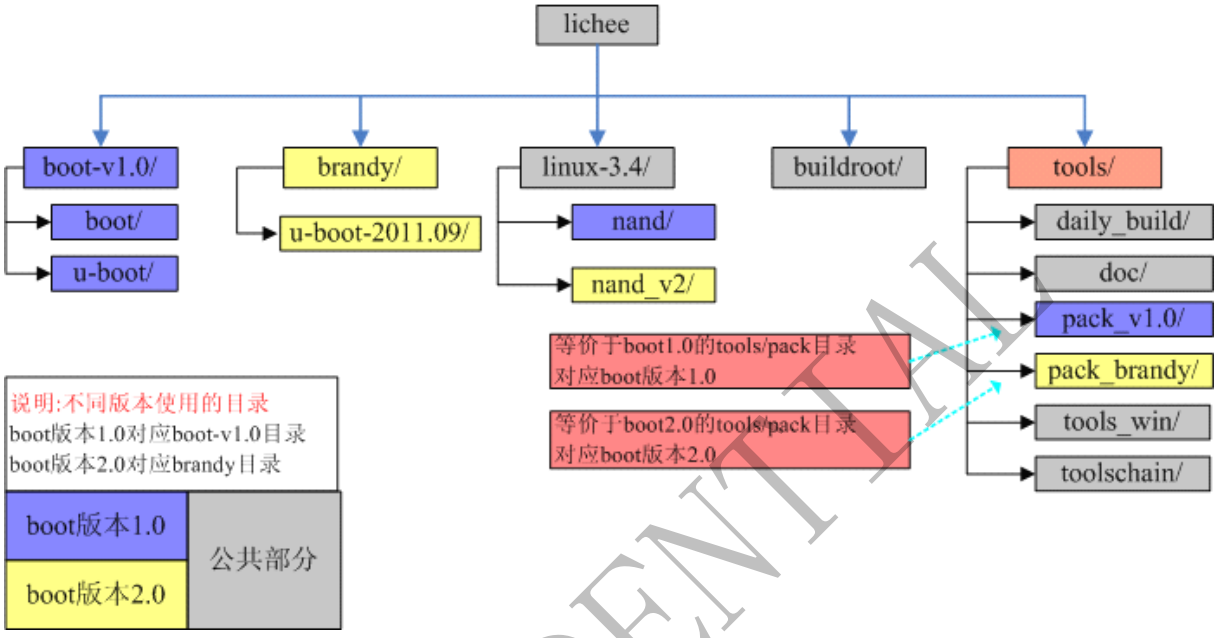


图 1 A20 Sugar v2.1 合并后的 lichee 目录结构

1、boot1.0 对应目录 boot-v1.0；
2、boot2.0 对应目录 brandy；
3、boot1.0 使用 tools/pack_v1.0 目录，相当于原 tools/pack 目录；
4、boot2.0 使用 tools/pack_brandy 目录，相当于原 tools/pack 目录；

注意：目前编译 lichee 都会编译对应的 uboot，
使用 boot2.0 是自动编译 lichee/brandy 目录，
使用 boot1.0 时自动编译 lichee/boot-v1.0/u-boot 目录。

FAQ:

1. 怎样区分之前的 sdk 使用的是 boot1.0 还是 boot2.0 ？

答：(A) a20 sugar v1.2 sdk 使用的是 boot1.0；
(B) a20 sugar v2.0 sdk 使用的是 boot2.0；

2. 怎样将之前 homlet A20 v1.2 或者 v2.0 的 sys_config.fex 或者 uboot 的修改同步到 v2.1？

答：如上图 1 所示，
(A) boot1.0 对应的目录是 lichee/boot-v1.0/u-boot/, lichee/tools/pack_v1.0；
(B) boot2.0 对应的目录是 lichee/brandy/u-boot-2011.09/, lichee/tools/pack_brandy；
其中，sdkv1.2 和 v2.0 的 pack 目录分别 tools/pack_v1.0 和 lichee/tools/pack_brandy 目录。

3. 客户同步到最新代码后如何将开发分支切换到 sugar v2.1？

由于从 sugar v2.0 开始，linux-3.3 已不再更新了，同样 sugar v2.1 也没有对 linux-3.3 做任何更新，但为了保持跟 sugar v1.2 一致，仍然保留 linux-3.3 的对外下载信息，但 linux-3.3 的修改截止于 sugar v1.2，而在 sugar v2.0 时只对 linux-3.3 打了 tag 没有做任何更新。

所以下载代码后，当切换到 v2.1 开发时，注意下面的报错信息，是由于 linux-3.3 没有更新任何 tag 导致的，可以忽略。

```
lichee$ repo forall -c git checkout -b v2.1 homlet-sugar42-v2.1
Switched to a new branch 'v2.1'
Switched to a new branch 'v2.1'
Switched to a new branch 'v2.1'
Switched to a new branch 'v2.1'
fatal: git checkout: updating paths is incompatible with switching branches.
Did you intend to checkout 'homlet-sugar42-v2.1' which can not be resolved as commit?
Switched to a new branch 'v2.1'
Switched to a new branch 'v2.1'
```

上面错误由于 linux-3.3 的 tag 没有 homlet-sugar42-v2.1，忽略即可。

```
@Exdroid24:~/workspace/sugarv2.1/lichee$ repo branch
*  v2.1                      | not in linux-3.3
```

开发建议：如果基于 sugar v2.0 开发的，为了不产生误解，建议把 linux-3.3 的整个目录删除。

4. 基于 tag sugar v2.1 切换到 tag v2.0 时，编译出错怎样处理？

A) 由于 android 目录结构没有变化，所以在 android 根目录下切换成功，命令如下：

```
repo forall -c git checkout -b sugar_v2.0 homlet-sugar-android-v2.0
```

B) 在 lichee 下切换时也不会报错

```
repo forall -c git checkout -b sugar_v2.0 homlet-sugar-lichee-v2.0
```

C）但在 lichee 下编译时时报错，信息如下：

```
lichee$ ./build.sh -p sun7i_android
ERROR: You are not at the top directory of lichee.
ERROR: Please changes to that directory.
```

上面的报错的原因是由于 sugar v2.1 调整了目录结构，把原来的

```
lichee\boot    ---> lichee\boot-v1.0\boot
lichee\u-boot ---> lichee\boot-v1.0\u-boot
```

Sugar v2.0 的代码在开始编译的时候脚本 lichee\buildroot\scripts\mkcmd.sh 会检查 boot 和 u-boot 的位置，如下图所示，只要将 mkcmd.sh 的指定更改一下或者把 boot 和 u-boot 从 boot-v1.0 目录复制到 lichee 根目录也可以解决编译错误的问题。


```
diff --git a/scripts/mkcmd.sh b/scripts/mkcmd.sh
index 3ed8719..0441bef 100755
--- a/scripts/mkcmd.sh
+++ b/scripts/mkcmd.sh
@@ -29,11 +29,11 @@ function mk_info()

 # define importance variable
 LICHEE_TOP_DIR=`pwd`
-LICHEE_BOOT_DIR=${LICHEE_TOP_DIR}/boot
+LICHEE_BOOT_DIR=${LICHEE_TOP_DIR}/boot-v1.0/boot
 LICHEE_BR_DIR=${LICHEE_TOP_DIR}/buildroot
 LICHEE_KERN_DIR=${LICHEE_TOP_DIR}/linux-3.4
 LICHEE_TOOLS_DIR=${LICHEE_TOP_DIR}/tools
-LICHEE_UBOOT_DIR=${LICHEE_TOP_DIR}/u-boot
+LICHEE_UBOOT_DIR=${LICHEE_TOP_DIR}/boot-v1.0/u-boot
 LICHEE_OUT_DIR=${LICHEE_TOP_DIR}/out

 # make surce at the top directory of lichee
```

切换到 v2.0 编译报错的处理方法

4. 编译打包

4.1. 编译内核和打包 android 固件

由于 boot 版本 1.0 与 nand1.0, boot 版本 2.0 与 nand2.0 绑定，所以选择编译 nand2.0 即选择了使用 boot 版本 2.0，同样意味着使用的 tools 目录下 pack_brandy 目录。(默认情况下是编译 nand2.0)

4.1.1. 选择使用 boot2.0

4.1.1.1. 编译 linux kernel 和 uboot

a）进入 lichee 目录

```
$cd lichee
```

b) 编译内核

方法 1，不指定-v 参数，默认使用 boot2.0：

```
Lichee$ ./build.sh -p sun7i_android
```

方法 2，明确指定-v boot2.0 即使用 boot2.0：

```
Lichee$ ./build.sh -p sun7i_android [-v boot_v2.0]
```

以上两种方法均可以编译 boot2.0。

注意：选择了 boot2.0, 同时使用 lichee/tools/pack_brandy 目录下的 boot 相关二进制文件和 sys_config.fex 等平台配置文件，编译完内核后会自动编译 boot2.0 的 uboot，即编译 lichee/brandy/u-boot-2011.09。

注释：-p 参数指定平台，脚本默认会从 linux-3.4/arch/arm/configs/sun7ismp_android_defconfig 复制到 linux-3.4/.config 作为内核编译配置，如果 linux-3.4/.config 存在则不覆盖。

4.1.1.2. 编译 android

a）初始 android 编译环境

```
android$ source build/envsetup.sh
```

b） 选择编译方案

```
android$ lunch
```

lunch 后提示：Which would you like? [full-eng]

输入 16 即选择参考方案 sugar_ref001-eng

c）拷贝内核和内核驱动

```
android$   extract-bsp
```

d）编译 android

```
android$ make -j8
```

-j 表示编译时使用的线程数，越多越快，但需要考虑编译主机的 cpu 数目。

e）打包 android 固件

```
android$ pack
```

输出固件地址，boot2.0 即 pack_brandy 目录。

```
/lichee/tools/pack_brandy/sun7i_android_sugar-ref001.img
```

4.1.2. 选择使用 boot1.0

注意：使用 boot2.0 方案即 brandy 的请忽略本章节。

4.1.2.1. 编译 linux kernel 和 uboot

a）进入 lichee 目录

```
$cd lichee
```

b) 编译内核

必须指定-v boot_v1.0 参数：

```
Lichee$ ./build.sh -p sun7i_android -v boot_v1.0
```

注意：选择了 boot1.0,同时使用 lichee/tools/pack_v1.0 目录下的 boot 相关二进制文件和 sys_config.fex 等平台配置文件，编译完内核后会自动编译 boot1.0 的 uboot，即编译 lichee/boot-v1.0/u-boot。

注释：-p 参数指定平台，脚本默认会从 linux-3.4/arch/arm/configs/sun7ismp_android_defconfig 复制到 linux-3.4/.config 作为内核编译配置，如果 linux-3.4/.config 存在则不覆盖。

4.1.2.2. 编译 android

a）初始 android 编译环境

```
android$ source build/envsetup.sh
```

b） 选择编译方案

```
android$ lunch
```

lunch 后提示：Which would you like? [full-eng]

输入 16 即选择参考方案 sugar_ref001-eng

c）拷贝内核和内核驱动

```
android$   extract-bsp
```

d）编译 android

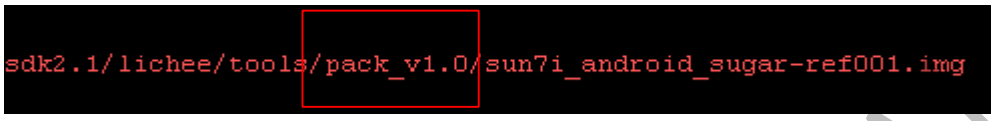
```
android$ make -j8
```

-j 表示编译时使用的线程数，越多越快，但需要考虑编译主机的 cpu 数目。

e）打包 android 固件

```
android$ pack
```

输出固件地址，boot1.0 即 pack_v1.0 目录。



4.2. 编译生成 dragonboard 固件

4.2.1. 使用 boot1.0 生成 dragonboard 固件

a)、lichee 目录下执行(以 wing-evb-v10 为例)

```
lihcee$ ./build.sh -p sun7i_dragonboard   -v boot_v1.0
```

b)、生成 dragonboard 固件

```
lihcee$ ./build.sh   pack
```

c)、选择项目

All valid chips:

0. Sun7i

Please select a chip: **0**

d)、选择项目平台

All valid platforms:

0. android

1. dragonboard

2. linux

Please select a platform: **1**

e)、选择平台

All valid boards:

0. wing-evb-v10

1.wing-k70

2. wing-s738

Please select a board: **0**

f)、生成固件到 tools/pack_v1.0 目录下

```
sdk2.1/lichee/tools/pack_v1.0/sun7i_dragonboard_wing-k70.img
```

4.2.2. 使用 boot2.0 生成 dragonboard 固件

a)、lichee 目录下执行(以 wing-evb-v10 为例)(默认使用 boot2.0)

```
lihcee$ ./build.sh -p sun7i_dragonboard
或者
lihcee$ ./build.sh -p sun7i_dragonboard   [-v boot_v2.0]
```


b)、生成 dragonboard 固件

```
lihcee$ ./build.sh  pack
```

c)、选择项目

All valid chips:

0. sun7i

Please select a chip: **0**

d)、选择项目平台

All valid platforms:

0. android

1. dragonboard

2. linux

Please select a platform: **1**

e)、选择平台

All valid boards:

0. wing-evb-v10

1.wing-k70

2. wing-s738

Please select a board: **0**

f)、生成 dragonboard 固件到 tools/pack_brandy 目录下

```
'a20/sdk2.1/lichee/tools/pack_brandy/sun7i_dragonboard_wing-evb-v10.img
```


4.3. boot 编译(可忽略此章节)

（注意：默认编译内核的同时会编译 u-boot，所以开发者可直接忽略此章节）

boot 版本 1.0 和 boot 版本 2.0 将 uboot 都会随编译内核一起编译(亦可以单独编译)，不管是 boot 版本 1.0 还是 boot 版本 2.0 编译后，生成的文件都将自动分别放到 tools 目录下对应 pack_v1.0 和 pack_brandy 目录下，其中 boot 版本 1.0 生成的文件对应放到 pack_v1.0 目录下，boot 版本 2.0 生成的文件将自动放到 pack_brandy 目录下。

4.3.1. 编译 boot 版本 1.0(boot-v1.0)

Boot 版本 1.0 编译分为三个部分：boot0、boot1、u-boot。

注意：使用 boot2.0 方案即 brandy 的请忽略本章节。

A）编译 boot0

a) 该部分编译使用 ARMCC 编译器，服务器上是编译不了的，由 boot 相关人员专门编译提供，该部分编译结果随 tools 目录分支更新，用户无须关注 boot0。

B）编译 boot1

a) 进入 lichee/boot-v1.0/boot 目录

```
cd lichee/boot-v1.0/boot
```

b) 编译 boot1

```
lichee/boot-v1.0/boot$ make
```

c) 编译结果将自动 copy 到 tools/pack_v1.0 目录下

C）编译 u-boot

a) 进入 lichee/boot-v1.0/u-boot 目录

```
cd lichee/boot-v1.0/u-boot
```

b) 编译 u-boot

```
lichee/boot-v1.0/u-boot$ make distclean        //清除前一次配置
lichee/boot-v1.0/u-boot$ make sun7i_config    //设置编译配置
lichee/boot-v1.0/u-boot$ make -j8      //编译 uboot
```

c) 编译结果同样是自动 copy 到 tools/pack_v1.0 目录下\

4.3.2. 编译 boot 版本 2.0(brandy)

Boot 版本 2.0 编译分为两个部分：boot0、u-boot。

A) 编译 boot0

a) 该部分编译使用 ARMCC 编译器，服务器上是编译不了的，由 boot 相关人员专门编译提供，该部分编译结果随 tools 目录分支更新，用户无须关注 boot0。

B) 编译 u-boot

a) 进入 lichee/brandy/u-boot-2011.09 目录

```
cd lichee/brandy/u-boot-2011.09
```

b) 编译 u-boot

```
lichee/brandy/u-boot-2011.09$ make distclean        //清除前一次配置
lichee/brandy/u-boot-2011.09$ make sun7i_config    //设置编译配置
```



```
lichee/brandy/u-boot-2011.09$ make -j8        //编译 uboot
```

c) 编译结果同样是自动 copy 到 tools/pack_brandy 目录下\

5. 烧录固件的方法

pack 生成 android 固件或者 dragonboard 固件之后，提供两种烧录固件的方法：usb 方式和卡方式。

5.1. usb 方式

Usb 烧写方式即通过 usb 线缆将 PC 与板子 usb0 连接，利用 PC 上的软件实现烧写。
PC 可用于烧写的软件有 LiveSuit、PhoenixSuit、PhoenixUSBPro。
烧写软件位于 SDK 的 lichee\tools\tools_win\USB_update_and_produce 目录。
其中 LiveSuit 分别提供 mac 版和 linux 版，PhoenixSuit 和 PhoenixUSBPro 只提供 windowes 版。

各个软件所在的压缩包内有软件使用方法，此处只介绍 windows 下如何进入 usb 烧写模式。
1、上电时进入 usb 烧写模式
连接板子串口，上电时在 SecureCRT 终端中一直按 2，PhoenixSuit 或 PhoenixUSBPro 会自动检测到板子，并且进入烧写过程。
2、按 uboot 键进入 usb 烧写模式
按住 uboot 键上电，板子会进入烧写模式，PhoenixSuit 或 PhoenixUSBPro 会自动检测到板子，并且进入烧写过程。
3、reboot efex 进入烧写模式
板子内已经存在 android 固件，并且可以启动，系统运行时在 adb shell 或者串口终端中输入 reboot efex 重启进入烧写模式。

5.2. 卡量产方式

卡量产即通过 SD 卡进行刷机。需要使用 PhoenixCard 制作量产卡，
PhoenixCard 所在路径为：lichee\tools\tools_win。

卡量产方式按以下步骤实现：
1、使用 PhoenixCard 软件在 PC 上制作量产卡；
2、将制作好的量产卡插入板子上，然后上电，此时 HDMI 输出量产进度，板子电源灯闪烁。
3、当电源灯停止闪烁时，量产完毕，拔出 SD 卡。

6. 优缺点说明

6.1. 优点

1.1、兼容了 boot1.0 和 boot2.0 的代码，方便客户的选择开发；
1.2、代码的统一管理，方便选择使用；

6.2. 缺点

2.1、tools 目录结构的更改，容易造成用户使用的困扰；
2.2、tools 目录的结构更改，与其他平台（A23\A80）差异，给开发人员带来维护上的难度，维护难度加大。
2.3、对于盒子产品，boot1.0 不具备掉电保护功能(对于 nand 机器)，所以不建议使用 boot1.0。

7. 注意事项

1、由使用 boot1.0 转到使用 boot2.0 或者是由使用 boot2.0 转到使用 boot1.0 必须对内核进行重新编译。

2、boot1.0 打包使用的配置文件是 tools/pack_v1.0 目录下的配置，而 boot2.0 使用的配置文件是 tools/pack_brandy 目录下的配置。

3、boot1.0 和 boot2.0 的 u-boot 都是自动编译，如果单独更改 uboot，可单独编译 uboot，然后重新打包即可。

4、boot1.0 的 boot1 如果更改必须单独编译，编译方式见【boot 编译】。

8. Declaration