



A20 Android 开发手册

V1.0

2013-03-15



Revision History

Version	Date	Section/ Page	Changes
1.0	2013-03-15		初始版本



目录

一、 A20 概述.....	4
1.1 A20 主控介绍.....	5
1.2 外围设备介绍.....	5
1.3 软件资源介绍.....	5
二、 建立开发环境.....	5
2.1 硬件资源.....	6
2.2 软件资源.....	6
2.2.1 安装 JDK (ubuntu12.04)	6
2.2.2 安装平台支持软件 (ubuntu12.04)	6
2.2.3 安装编译工具链 (ubuntu12.04)	6
2.2.4 安装 phoenixSuit (windows xp)	7
2.2.5 其他软件 (windows xp)	7
三、 源码下载.....	8
3.1 wing 源码下载.....	8
3.2 仓库的目录树.....	8
3.2.1 android 目录树.....	8
3.2.2 lichee 目录结构.....	9
3.2.2.1 buildroot 目录结构.....	9
3.2.2.2 linux-3.3 目录结构.....	10
3.2.2.3 u-boot 目录结构.....	11
3.2.2.4 tools 目录结构.....	12
3.2.2.5 boot 目录结构.....	12
四、 编译和打包.....	13
4.1 源码编译.....	13
4.1.1 lichee 源码编译.....	13
4.1.2 android 源码编译.....	13
4.2 打包固件.....	13
4.2.1 完全打包.....	13
4.2.2 局部打包.....	14
五、 固件烧写.....	14
5.1 使用 PhoenixSuit 烧写固件.....	14
5.2 使用 fastboot 更新系统.....	14
5.2.1 进入 fastboot 模式.....	14
5.2.2 fastboot 命令使用.....	15
六、 recovery 功能使用.....	15
6.1 键值的查看.....	15
6.2 按键选择.....	16
6.3 功能使用.....	16
七、 调试.....	17
7.1 调试 apk.....	17
7.2 调试 linux 内核.....	17









Confidential



一、A20 概述

A20 主控平台为珠海全志科技基于 ARM Cortex A7 开发的 Dual-Core 解决方案，GPU 采用 mali-400MP2，Memory 为 1G DDR3 (L) /LPDDR2，在无线方面支持 WIFI、BT、3G，该解决方案可以适用于 Tablet 和 Smart TV 等移动终端设备上。A20 与全志其他主控对比如下：

						
CPU	Quad-Core Cortex-A7	Dual-Core Cortex-A7	Single-Core Cortex-A8	Single-Core Cortex-A8	Single-Core Cortex-A8	Single-Core Cortex-A8
GPU	SGX544MP2	Mali400MP2	Mali400	Mali400	Mali400	Mali400
Video Decoder	4Kx2K	2160P	2160P	1080P	1080P	1080P
Video Encoder	H.264 1080P@60fps	H.264 1080P@30fps	H.264 1080P@30fps	H.264 1080P@30fps	H.264 1080P@30fps	H.264 1080P@30fps
Package	BGA609 18mmx18mm 0.65mm Pitch	BGA441 19mmx19mm 0.80mm Pitch	BGA441 19mmx19mm 0.80mm Pitch	BGA336 14mmx14mm 0.65mm Pitch	BGA336 14mmx14mm 0.65mm Pitch	eLQFP176 20mmx20mm
Application	Tablet Smartphone Smart TV	Tablet Smart TV	Tablet Smart TV	Smartphone	HDMI Dongle	Tablet E-reader

1.1 A20 主控介绍

A20 主控是采用双核 Cortex-A7 架构的 CPU，主频可达 1G (1008MHz)，功耗控制出色。图形方面，GPU 采用 Mali400MP2，兼容性更加出色。最高支持 2160P 的视频解码和 1080P@30fps 的编码，多媒体性能优异。A20 支持 1G 内存。另外 A20 还支持 1024x768 或 1024x600 等多种分辨率。

1.2 外围设备介绍

A20 主控平台支持丰富的 Camera 模块，WIFI 模块，蓝牙模块，3G 模块（电话系统），TF (SD) 卡扩展模块以及多种传感器。

1.3 软件资源介绍

A20 主控的系统 and 软件平台是建立在 Android 4.2 平台基础上，Linux 内核版本为 3.3。Android 生态系统支持影音，网络，娱乐，系统管理，个人助手等丰富的扩展。



二、建立开发环境

本节将介绍，A20 平台开发环境所需的软硬件资源及的搭建。

2.1 硬件资源

- ✧ A20 主控开发板或主机一台
- ✧ 两台 PC：一台作为编译服务器（Linux 系统），令一台用于烧写固件（XP 系统）
- ✧ 串口线，12V 电源，小口 usb 各一个（条）

2.2 软件资源

Linux 主机（因为 A20 的软件系统方案选择的是 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

Windows XP 主机，作为固件烧写机器和本地调试环境，一般来说主机中需要：

- ✧ PhoenixSuit 一键烧写工具
- ✧ USB 转串口驱动
- ✧ Android SDK

下面以 ubuntu12.04 和 XP 为例，安装软件环境

2.2.1 安装 JDK（ubuntu12.04）

JDK 安装命令

```
$ sudo add-apt-repository "deb http://archive.canonical.com/ lucid partner"
```

```
$ sudo apt-get update
```

```
$ sudo apt-get install sun-java6-jdk
```

2.2.2 安装平台支持软件（ubuntu12.04）

```
$ sudo apt-get install git gnupg flex bison gperf build-essential \
```

```
zip curl libc6-dev libncurses5-dev:i386 x11proto-core-dev \
```

```
libx11-dev:i386 libreadline6-dev:i386 libgl1-mesa-glx:i386 \
```

```
libgl1-mesa-dev g++-multilib mingw32 tofrodos \
```

```
python-markdown libxml2-utils xsltproc zlib1g-dev:i386
```

```
$ sudo ln -s /usr/lib/i386-linux-gnu/mesa/libGL.so.1 /usr/lib/i386-linux-gnu/libGL.so
```



2.2.3 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/boot/config/gcc-linaro/` 中。

2.2.4 安装 phoenixSuit （windows xp）

phoenixSuit 位于 `lichee/tools/tools_win` 中，将 `PhoenixSuitPacket.msi` 复制到 XP 主机上，按照安装向导提示安装，即可完成 phoenixSuit 的安装。

2.2.5 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。



三、源码下载

本节将介绍 A20 开发平台下源码的下载及源码结构

3.1 wing 源码下载

参照《A20 仓库下载说明 V1.0》。

3.2 仓库的目录树

下面介绍仓库中的目录树结构，主要介绍 android 仓库的目录树结构，lichee 的目录树结构

3.2.1 android 目录树

android 目录下是 android 仓库中的 android 源码，A20 使用的 android 系统为 android4.2。查看 android 的目录树结构，在 android 的根目录下执行如下命令

\$ tree -L 1

```
.
├── abi
├── bionic
├── bootable
├── build
├── cts
├── dalvik
├── development
├── device
├── docs
├── external
├── frameworks
├── gdk
├── hardware
├── libcore
├── libnativehelper
├── Makefile
├── ndk
├── packages
└── pdk
```




```
├── prebuilts
├── sdk
├── system
└── tools
```

上面目录中，与 android 官方源码相同

3.2.2 lichee 目录结构

lichee 目录下包含 buildroot 工具、linux 内核、以及 uboot 源码。查看 lichee 的目录结构，在 lichee 的根目录下执行下面的命令

\$ tree -L 1

```
.
├── boot
├── buildroot
├── build.sh
├── linux-3.3
├── README
├── source
├── tools
└── u-boot
```

3.2.2.1 buildroot 目录结构

buildroot 的主要作用是

- ✧ 管理包之间的依赖关系
- ✧ 生成 ARM 交叉工具链
- ✧ 生成 U-Boot
- ✧ 制作与制定根文件系统
- ✧ 生成最终用于烧写的固件包

在 buildroot 的根目录下执行下列命令，查看 buildroot 目录结构如下

\$ tree -L 1

```
.
├── board
├── boot
├── build.sh
├── CHANGES
├── Config.in
├── configs
├── COPYING
├── dl
├── docs
├── external-packages
└── fs
```



```
├── linux
├── Makefile
├── output
├── package
├── README
├── scripts
├── target
└── toolchain
```

其中，boot 目录里存放 Boot 代码，config 目录里存放预定义好的配置文件，比如我们的 sun7i_defconfig，dl 目录里存放已经下载好的软件包，scripts 目录里存放 buildroot 运作的代码，target 目录里存放用于生成根文件系统的一些规则文件。对于我们来说最为重要的是 package 目录，里面存放了将近 3000 个软件包的生成规则，我们可以在里面添加我们自己的软件包或者是中间件。更多关于 buildroot 的介绍，可以到 [buildroot](http://buildroot.org) 的官方网站获取

3.2.2.2 linux-3.3 目录结构

Linux-3.3 目录结构包含了 linux kernel 3.3 的源码，在 Linux-3.3 的根目录使用如下命令查看，该目录的结构

\$ tree -L 1

```
.
├── arch
├── bImage
├── block
├── build.sh
├── COPYING
├── CREDITS
├── crypto
├── Documentation
├── drivers
├── firmware
├── fs
├── include
├── init
├── ipc
├── Kbuild
├── Kconfig
├── kernel
├── lib
├── MAINTAINERS
├── Makefile
├── mm
├── modules
└── net
```



```
|— output
|— README
|— REPORTING-BUGS
|— rootfs
|— samples
|— scripts
|— security
|— sound
|— tools
|— usr
└─ virt
```

以上目录结构跟标准的 Linux 内核是一致的，除了多一个 `modules` 目录。`modules` 目录是我们扩展用来存放没有跟内核的 `menuconfig` 集成的外部模块的地方。我们目前放了 `example`, `nand`, `mali` 和 `wifi` 这 4 个外部模块，其中 `example` 是示例用的，`mali` 是我们的 GPU 驱动，`test` 是模块测试用例，目前只存放了 `nand` 的测试用例。

在 `modules` 的根目录下执行如下命令，结果如下

\$ tree -L 1

```
.
|— example
|— mali
|— nand
└─ wifi
```

3.2.2.3 u-boot 目录结构

u-boot 目录中存放的是 A20 主控平台 Linux 内核引导代码，在 u-boot 的根目录下执行下列命令，结果如下

\$ tree -L 1

```
.
|— api
|— arch
|— board
|— boards.cfg
|— build.sh
|— common
|— config.mk
|— COPYING
|— CREDITS
|— disk
|— doc
|— drivers
|— examples
└─ fs
```



```
├── include
├── lib
├── MAINTAINERS
├── MAKEALL
├── Makefile
├── mkconfig
├── mmc_spl
├── nand_spl
├── nand_sunxi
├── net
├── onenand_ipl
├── post
├── README
├── rules.mk
├── snapshot.commit
├── spl
└── tools
```

除了添加我们自己的 sunxi 平台设置，目录结构与官方网站上的没有区别，有关 u-boot 的详情请参阅 [u-boot](#) 的官方文档。

3.2.2.4 tools 目录结构

tools 目录为打包工具目录，与打包相关的脚本和工具都放在该目录中。它的结构如下：

\$ tree -L 1

```
.
├── daily_build
├── pack
└── tools_win
```

3.2.2.5 boot 目录结构

boot 目录中存放的是 A20 平台的 bootloader，该目录为 A20 启动代码，该目录的结构如下：

\$ tree -L 1

```
.
├── boot0
├── boot1
├── config
├── Makefile
├── pack
└── workspace
```



四、编译和打包

本节讲解源码编译和打包的方法

4.1 源码编译

下面将分别介绍 lichee 和 android 的从源码进行编译的方式方法

4.1.1 lichee 源码编译

使当前目录为 lichee 的根目录。然后执行下面的命令：

./build.sh -p sun7i_android

或者：

\$. buildroot/scripts/mksetup.sh #导入环境变量，根据提示选择对应选项

\$ mklichee

即完成了一次 lichee 的编译（根据服务器配置，耗时至少 10 分钟），编译成功，屏幕上会出现

INFO:build u-boot OK.

...

INFO:build rootfs OK.

INFO:build lichee OK.

4.1.2 android 源码编译

在确保 lichee 已经编译，并且使当前目录为 android 的根目录。然后执行下面的命令

\$. build/envsetup.sh #导入环境变量

\$ lunch #根据自己的开发平台，选择方案

\$ extract-bsp #拷贝内核和模块到 android 中

\$ make -j8 #-j 开启多核编译，服务器开发一般为服务器 cpu 数量的一半

编译成功，会在 out/target/product/wing-xxx/ 目录下面会生成 boot.img, recovery.img, system.img 3个包。

4.2 打包固件

本节涉及两种打包方式，一种是完全打包，一种是局部打包

4.2.1 完全打包

在保证 lichee 和 android 都编译完成的基础上，相关环境变量已经导入，只需要在 android



的根目录下执行下列命令即可

\$ pack

打包成功后，将会在 lichee/tools/pack 目录下生成 sun7i_android_xxx.img 文件，即生成我们所需的固件

4.2.2 局部打包

boot.img 镜像中包含 linux kernel 和内存盘 ramdisk，如果内核有修改，就需要重新编译内核，然后再 android 目录下执行下列命令，即可打包生成 boot.img

\$. build/envsetup.sh

\$ lunch #选择 wing-xxx 产品

\$ extract-bsp

\$ make bootimage

这样就生成了 boot.img，类似的方法就可以重新打包生成 system.img

\$. build/envsetup.sh

\$ lunch #选择 wing-xxx 产品

\$ make systemimage-nodeps

重新生成的镜像在 android 目录下的 out/target/product/wing-xxx/目录下

五、固件烧写

本小节介绍 A20 下的固件的烧写

5.1 使用 PhoenixSuit 烧写固件

安装好 PhoenixSuit 后，双击运行，单机一键刷机，选择我们生成的固件，如果我们的 A20 主动设备没有烧如任何系统，可以按住设备上的 recovery 键后插入 usb 然后按几下 power 键系统就会进入自动升级状态。如果已经烧入 android 系统，开机后与 windows 主机连接好，单机立即升级即可自动烧入新的固件。

5.2 使用 fastboot 更新系统

fastboot 是一种线刷，就是使用 USB 数据线连接手机的一种刷机模式，在 A20 主控中，可以使用 fastboot 的功能来实现局部系统的更新。

5.2.1 进入 fastboot 模式

- ◇ 启动开发板，在串口界面敲任意按键，可以进入 u-boot；如果进不了 fastboot，将 lichee/tools/pack/chips/sun7i/configs/android/default/env.cfg 中的 bootdelay=0 改成 bootdelay=2 重新打包固件即可（需要安装 google-usb_driver 驱动）。



- ✧ 在串口命令行输入 fastboot 命令，进入 fastboot 模式；
- ✧ 通过 pc 端的 fastboot 工具烧录各个固件包（fastboot 是 windows 下的一个工具（android sdk 中有），上网自己下载一个，解压到本地，然后将 fastboot.exe 添加到 windows 环境变量）
- ✧ 进入在 windows 命令行：cmd 进行命令行模式，于是可以在命令行执行 fastboot 指令
- ✧ 退出 fastboot 模式：ctl+c

5.2.2 fastboot 命令使用

在 windows 命令行使用 fastboot 命令。

擦除分区命令：

\$ fastboot erase boot #擦除 boot 分区

\$ fastboot erase system #擦除 system 分区

\$ fastboot erase data #擦除 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

六、recovery 功能使用

Recovery 是 Android 的专用升级模式，用于对 android 自身进行更新；进入 recovery 模式的方法是，在 android 系统开机时，按住一个特定按键，则会自动进入 android 的 recovery 模式。

6.1 键值的查看

按键是通过 AD 转换的原理制成。当用户按下某个按键的时候，会得到这个按键对应的 AD 转换的值。同时，所有的按键的键值都不相同，并且，键值之间都有一定的间隔，没有相邻。比如，键值可能是 5,10,15,20，但是不可能是 5,11,12,13。

为了方便用户查看不同按键的键值，这种方法要求连接上串口使用，因此适合于开发阶段使用。具体步骤是：

- ✧ 把小机和 PC 通过串口线连接起来，设置屏幕焦点在串口调试软件上；
- ✧ 用户开机之前，按住 PC 键盘上的数字键“3”；
- ✧ 开机，等待，1 秒后可以松开电脑键盘；

经过这样的步骤，用户会看到屏幕上有如下的打印信息出现：

welcome to key value test

press any key, and the value would be printed

press power key to exit

这表示系统已经进入了按键的键值测试模式，这种模式下将一直等待用户按下按键，并



在串口屏幕上把按键的键值打印出。这样，用户可以很方便地查看不同按键的键值。比如，当按下某一个按键，用户可以看到如下的打印信息。

```
key value = 8
key value = 8
key value = 8
key value = 63
```

由于 AD 采用的速度非常快，所以同一个按键按下，屏幕上会出现多个值。用户可以看出，这个按键的键值是 8。最后出现的 63 是松开按键的时候的采用，是需要去掉的干扰数据。因此，用户查看按键键值的时候只要关注前面打印出的数值，后面出现的应该忽略不计。

6.2 按键选择

通常情况下，一块方案板上的按键个数不同，或者排列不同，这都导致了方案商在选择作为开机阶段 recovery 功能的按键有所不同。因此，系统中提供了一种方法用于选择进入 recovery 模式的按键：

在 efex/sys_config.fex 配置脚本中，提供了一项配置，用于选择按键的键值，如下所示：

[recovery_key]	
key_max	= 4
key_min	= 6

它表示，所选择用于作为 recovery 功能的按键的键值范围落在 key_min 到 key_max 之间，即 4 到 6 之间。由于所有按键的选择都可以通过前面介绍的方法查看，因此，假设用户要选的按键是 a，用户这里选择配置的方法是：

- ◇ 按照前面介绍的方法，读出所有按键的键值；
- ◇ 读出 a 的键值 a1，同时取出两个相邻于 a 的键值，记为 b1 和 c1，b1 大于 c1；
- ◇ 计算出 $(a1 + b1)/2$ ， $(a1 + c1)/2$ ，分别填写到 key_max 和 key_min 处；
- ◇ 如果 a1 刚好是所有按键的最小值，则取 key_min 为 0；如果 a1 刚好是所有按键的最大值，则取 key_max 为 63；

经过以上的步骤，就可以选择一个特定的按键进入 recovery 模式。取了一个平均值的原因是考虑到长时间的使用，电阻的阻值可能会略有变化导致键值变化，取范围值就可以兼容这种阻值变化带来的键值变化。

6.3 功能使用

在 android 编译完毕之后，使用如下命令

```
$ get_uboot
```

```
$ make otapackage
```

就可以在 android/out/target/product/wing-xxx/目录下生成一个 wing-xxx-ota-buildtime.zip 文件。

在系统启动时，按住设定的特定按键进入 recovery 模式，进入该模式后，可以选择升级文件升级。



七、调试

本节介绍如何生成调试应用和固件的方式方法

7.1 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 wing-evb 产品

\$ 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-evb-v10/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 push Gallery2Tests.apk /system/app/

7.2 调试 linux 内核

在更改了内核相关文件后，在 lichee 目录下执行以下命令编译内核

\$./build.sh -p sun7i_android

在 android 目录下执行以下命令，打包生成 boot.img

\$. build/envsetup.sh

\$ lunch #选择 wing-xxx 产品

\$ extract-bsp

\$ make bootimage

生成了 boot.img 之后，需要通过 fastboot 工具刷到小机：

- ✧ 重新启动开发板，在串口界面敲任意按键，可以进入 u-boot；
- ✧ 在串口命令行输入 fastboot 命令，进入 fastboot 模式；
- ✧ 进入 windows 命令行：cmd 进行命令行模式，在命令行执行 fastboot(前提是已经安装了 fastboot 工具)，将 boot.img 拷贝到小机上即可。

\$ fastboot erase boot

\$ fastboot flash boot boot.img

7.3 调试 android 系统

在 android 目录下执行以下指令，在 out/target/product/*wing-xxx* 目录下生成 android system.img。

\$. build/envsetup.sh



\$ lunch #选择 wing-xxx 产品

\$ make systemimage-nodeps

通过 fastboot 工具刷到小机:

- ✧ 重新启动开发板, 在串口界面敲任意按键, 可以进入 u-boot;
- ✧ 在串口命令行输入 fastboot 命令, 进入 fastboot 模式;
- ✧ 进入在 windows 命令行: cmd 进行命令行模式, 在命令行执行 fastboot 指令(前提是已经安装了 fastboot 工具), 将 system.img 拷贝到小机上即可。

\$ fastboot erase system

\$ fastboot flash system system.img