| | | | | | |
|---|---|---|---|---|---|
| | | 文档级别 | | 总页数 | |
| 文档名称 | | 编制人 | | 所在部门 | |
| | | 审核人 | | 编制日期 | |

# 基于 OMAP3530 的 android2.2 移植

## 目　　录

# 1. 概述

目前正在研究 OMAP3530，考虑到操作系统的移植，由于 android 是比较流行的开源系统，因此考虑将 android 移植到开发板上。TI 提供了最新的 android 开发 SDK，这里正好使用这个 SDK 对目前正在使用的开发板进行移植。

# 2. 开发板简要说明

目前使用的开发板是由国内 timll 公司生产的 SBC8100，其外设还算比较丰富。关于 SBC8100 这里不再说明。

# 3. 主机配置

由于编译 android 的工作量相对较大，这里不考虑使用虚拟机来进行，这里使用一台完全的 linux 主机来进行工作。主机采用 fedora13 操作系统，可以连接互联网，以下的操作都是以 root 进行的，虽然不安全，但比较方便。

# 4. 预备工作

## 4.1 下载 SDK 压缩包

从 TI 的网站上下载（不需要注册）最新的 android SDK 包，文件名为 TI_Android_FroYo_DevKit-V2.tar.gz，从 google 上搜索就可以找到，这是基于 android2.2 的，不过其内部不包含 android2.2 源代码。在下载页面将这一个下载就可，下面的文件都包含在这个压缩文件内。解压缩，生成 TI_Android_FroYo_DevKit-V2 目录。目录内有内核源代码、u-boot 源代码和 x-loader 源代码以及部分开发板的已编译好的文件系统等。

## 4.2 下载 android2.2 源代码

确认主机能够连接互联网，确认主机硬盘空间足够（这里的 android 源代码大约 2.4GB），打开一个终端。首先建立一个空文件夹，

```
mkdir myandroid
cd myandroid
```

然后将 repo 文件下载到/bin 目录内，改变其属性可执行，

```
curl http://android.git.kernel.org/repo>/bin/repo
chmod a+x /bin/repo
```

下载源代码，这个源代码是针对 TI 的，不是主版本的。

```
repo init -u git://gitorious.org/rowboat/manifest.git -m TI-Android-FroYo-DevKit-V2.xml
repo sync
```

下载过程比较漫长，如果主机 linux 版本较低，需要更新 python 到 2.5 或以上版本，否则在 repo init 时可能会报错。这里 git 在安装 fedora13 时已安装，如没有安装需要安装 git。

下载完毕，在 mydroid 文件夹内共有 16 个文件夹和 1 个 Makefile 文件，还有一个隐藏文件夹./repo，这个可能是给 repo 操作用到，在编译时没用。这个结构与普通的 android 的源代码结构似乎有一些差别。

下文就以/myandroid 作为源代码目录。这里的文件夹仅用于说明，在实际使用时应建立清晰的文件夹。

### 4.2.1 打包压缩

如果要提取打包源代码并不再更新，这里可以参考网上一篇关于 android 源代码打包的文章《Android 代码提取打包》。首先建立一个空文件夹，

```
cd /
mkdir source_android
cd myandroid
cp bionic cts device hardware bootable dalvik external kernel packages system build development frameworks prebuilt sdk ndk Makefile /source_android/ -ar
cd /source_android/
find . -name .git -type d | xargs rm –rf
cd /
7z a source_android.7z /source_android/
```

这里是用 7zip 压缩的，开始使用 tar 压缩，压缩时没有问题，但解压缩时报错，因此后改用 7zip。压缩文件大小不到 600MB。

## 4.3 下载编译相关工具

在编译 android 时需要很多相关的工具，在安装 fedora13 时不一定全部安装，因此可以参考网上的文章《Fedora13 下编译 Android2.2》先下载安装好这些工具。

```
yum install bison*
yum install gcc-c++
yum -y install flex
yum install ncurses-devel
yum install libX11-devel.i686
yum install readline-devel
yum install gperf
```

对于 jdk 的安装，主机系统默认安装的 openjdk1.6，但 android 的编译说明中要求必须为 jdk1.5，并且是 update12 或以上。这里为了少走弯路，将系统默认安

装的 openjdk1.6 卸载了。卸载时先找出安装的名称，

```
yum list installed>tmpfile
```

打开 tmpfile 可以看到安装的程序名称，然后卸载 openjdk1.6，

```
yum erase jdk-xxxxxxxxxxxxxxxx
```
（这里记不清了）

这里在卸载 jdk 时又卸载了很多相关的工具，这里暂时先不管。然后似乎又卸载了一次 jdk 相关的工具（似乎是 doc 相关的）。

从网上下载了一个 jdk-1_5_0_18-linux-i586.rpm，安装。

```
rpm -ivh jdk-1_5_0_18-linux-i586.rpm
```

程序安装在/usr/java/jdk1.5.0_18 目录下，在/etc/profile.d 目录新建一个文件 java.sh，内容如下，

```
#!/bin/bash
JAVA_HOME=/usr/java/jdk1.5.0_18
CLASSPATH=$CLASSPATH:$JAVA_HOME:$JAVA_HOME/lib
PATH=$JAVA_HOME/bin:$PATH
export JAVA_HOME CLASSPATH PATH
```

保存，在命令行输入

```
source /etc/profile.d/java.sh
```

然后，

```
java -version
```

可以看到版本应为 1.5.0_18。

### 4.4 编译

下面可以尝试编译一下，网上的 OMAPpedia 里有一篇《Building Android》的文章，较为详细的说明了 android 的编译过程，可以作为参考。这里还参考了《TI-Android-FroYo-DevKit-V2 UserGuide》。

```
cd /myandroid
make TARGET_PRODUCT=omap3evm TARGET_BUILD_VARIANT=tests –j2
```

这里的-j2、-j4 是对应主机处理器的，大概可以加快编译速度，在《Building Android》里的说明是 make -j8 for 2 x Quad Core CPU, make -j4 for either a Quad Core CPU or a 2 x Dual Core, make -j2 for a Dual Core CPU and make for a Single Core CPU。本文的主机是双核处理器，因此为-j2。

经过较长时间，编译竟然通过，在/out 目录下有生成的各种文件。不过这些文件是针对 OMAP3530EVM 的，本文使用的 SBC8100 还需要再改写。

不过在 android 中似乎没有多少可以改动的，顶多是开机 logo 和 init.rc 等几个文件，也就是在/myandroid/device/ti/omap3evm 目录下的几个文件。这里还没有考虑到与界面、应用等有关的更深入的部分（因为本人暂时也不会）。这里可能需要修改开机 logo，可以参照网上的方法得到一个 android-robot-on-black-480x272.rle，放到/myandroid/device/rowboat/generic/initlogo 目录内，然后修改/myandroid/device/ti/omap3evm 目录内的 AndroidBoard.mk 文件，将其内 android-robot-on-black-480x640.rle 改为 android-robot-on-black-480x272.rle。

## 5. 运行 android

### 5.1 编译 x-loader、u-boot 和内核

这里将 SDK 包里的 x-loader、u-boot 和内核源代码都进行了少量的改动并移植到 SBC8100 板上。这里 x-loader 和 u-boot 可能不是必须的，这里只是考虑将 SDRAM 的频率改成 166MHz，并且使用较新的 x-loader 和 u-boot 可能对系统的初始化的优化好一些。

对于内核应使用 SDK 包内的源代码，如果已经将 TI 的 PSP-SDK-03.00.01.06 的源代码移植好，那么改动量很小，因为二者都是基于 2.6.32 版本的内核，只是 android 的 SDK 的内核源代码中增加了少量与 android 相关的代码。

这里只说明在修改 omap3_sbc8100_android_defconfig 并编译时出现的一个问题，其余不再详细说明。由于此前已经将 PSP-SDK-03.00.01.06 的内核源代码跑通，因此 omap3_sbc8100_android_defconfig 这个文件是在原来的 omap3_sbc8100_defconfig 上参照 omap3_evm_android_defconfig 修改得到的，其中有一项 CONFIG_USB_ANDROID 是比较重要的，需要选 y（否则在 android 启动时可能会出现大量类似 untracked pid xxx exited 的说明，android 也启动不起来），但是在执行

```
make CROSS_COMPILE=arm-eabi- omap3_sbc8100_android_defconfig
```

后得到的.config 文件中这一项却始终没有选中，原因不明，也没有找到相关的关联选项，后来又拷贝出一份 omap3_evm_android_defconfig 并直接改名为 omap3_sbc8100_android_defconfig，然后再修改后编译得到的.config 中这一项就选中了。

## 5.2 配置文件系统

为了避免频繁擦写 NAND FLASH 或 MMC 卡，这里考虑先使用 NFS 文件系统。首先建立 NFS 文件系统目录，这里考虑名称为/nfs，在命令行中键入

```
mkdir nfs
cd nfs
```

将/myandroid/out/target/product/omap3evm/root/目录内的内容都拷贝到/nfs 目录内，

```
cp -Rfp /myandroid/out/target/product/omap3evm/root/* .
```

然后将/myandroid/out/target/product/omap3evm/system 目录内的内容拷贝到/nfs/system 目录内

```
cp -Rfp /myandroid/out/target/product/omap3evm/system/ .
```

还可以将/data 目录内的内容拷贝到/nfs/data 目录内，

```
cp -Rfp /myandroid/out/target/product/omap3evm/data/ .
```

不过如果将/data 目录内的内容拷贝到文件系统内，android 第一次启动的时间（即 android 字样的闪烁时间）将非常漫长，因为增加了很多程序。

```
cd dev
mknod -m 660 console c 5 1
mknod -m 660 null c 1 3
cd..
cd..
chmod 777 –R nfs(这后两句可能是不需要的)
```

然后对主机的 NFS 配置项中增加/nfs 目录为 NFS 服务器目录，在防火墙配置中对 NFS 服务放行。NFS 文件系统配置完毕。

## 5.3 运行

给目标板加电，进入 u-boot，这里的内核是通过 tftp 下载的，

```
tftpboot 0x80000000 uImage
setenv bootargs init=/init console=ttyS2,115200n8 mem=128M vram=12M omapfb.vram=0:12M noinitrd ip=192.168.0.27:192.168.0.6:192.168.0.254:255.255.255.0:omap3evm:eth0:off root=/dev/nfs rw nfsroot=192.168.0.6:/nfs,nolock omap_vout.vid1_static_vrfb_alloc=y androidboot.console=ttyS2
bootm 0x80000000
```

内核启动完毕后，在显示屏上会出现两个小机器人（比较奇怪），如果没有改动开机 logo，使用的 omap3evm 的开机 logo，可能机器人只能显示出一部分。然后出现 android 字样的闪烁，等待较长时间后进入界面，可能是因为显示屏分辨率较低或横屏显示（也可能其它原因），画面不是很精致。此时电流大约 590mA

（电压 5.4V）。

这里曾尝试将显示屏旋转过来，即在 u-boot 配置中增加 omapfb.rotate=1 omapfb.vrfb=y，但 android 显示不正常。

### 5.3.1 触摸屏驱动改进

这里使用触摸屏时效果不好，并且定位不准确，很难使用，这可能是 android 主要支持电容屏，对电阻屏的支持不太好（也可能其它原因）。经查找需要修改内核源代码中触摸屏的驱动 ads7846.c，这里参照 omap3evm 的修改方法。首先需要得到触摸屏的原始边界值，也就是 raw values，得到的方法需要使用 tslib，运行 tslib 的 ts_print_raw，用笔在屏上划，从串口可以得到类似如下的数据，共 5 列（.的前后各一列）

xxxxx.xxxxx: xvalue yvalue xx

其中找出第 3 列（X 方向）和第 4 列（Y 方向）的最大值和最小值，然后就可以参照 omap3evm 的修改方法通过 CONFIG_MACH_OMAP3_SBC8100 宏修改代码。修改这部分代码后重新编译生成内核，使用新的内核后触摸屏的效果有较大改善（但还不是特别好用）。

基于 OMAP3530 的 android2.2 的初步移植工作完成。