深圳市九鼎创展技术有限公司技术文档

文档名称：x210v3s linux 平台手册

深圳市九鼎创展科技有限公司

地址：深圳市宝安区兴业路宝安互联网产业基地 B 区 3003B 室

网址：http://www.9tripod.com

论坛：http://www.xboot.org

版权声明

敬告：

在售开发板的手册会经常更新，请在 http://www.9tripod.com 网站下载最新手册，不再另行通知。

版本说明

版本号	日期	作者	描述
Rev.01	2012-4-25	lqm	原始版本
Rev.02	2012-7-30	lqm	增加 inand 烧写方式； 增加 qt4.5 移植文档； 增加 qtopia 移植文档；
Rev.03	2012-11-15	lqm	增加 android2.3 USB 蓝牙支持； 增加 android2.3 USB 鼠标支持； 增加 android2.3 USB 键盘支持； 增加多种 VGA 分辨率支持； 增加 4.3 寸液晶屏支持； 本文档和 i210 开发平台兼容使用。
Rev.04	2012-11-17	lqm	增加 QT4.8 文档说明
Rev.05	2012-11-22	lqm	增加 QT4.8 inand 平台文档说明
Rev.06	2013-07-18	lqm	分离出 linux 文档

技术支持

一：论坛

在 BBS 论坛上发贴询问，是取得技术支持的有效途径，我司技术人员将在 24 小时内回贴，并尽可能的在 48 小时内解决所提出的问题。如遇节假日将顺延。

论坛网址：www.xboot.org

技术支持时间：周一到周五，9：00 到 18：00

二：邮件

在通过 BBS 论坛仍然无法解决的情况下，可以采用电子邮件的形式咨询。

邮箱地址：supports@9tripod.com

三：QQ 群

九鼎创展所有开发平台都对应有 QQ 交流群，QQ 群为广大客户提供一个共同交流讨论的地方，并不代表能够得到及时支持，技术人员并不能保证每时每刻在群里面盯着问题，很多技术问题并不能马上就能得到解决，请尽量在论坛上发贴提问，留给技术人员测试的时间，感谢您的支持。

网　　址：　www.9tripod.com

联系电话：　0755-61952306

E-mail：　supports@9tripod.com

销售与服务网络

公司：深圳市九鼎创展科技有限公司
地址：深圳市宝安区兴业路宝安互联网产业基地 B 区 3003B 室
邮编：518101
电话：4000033436　0755-33121205　0755-33133436
网址：http://www.9tripod.com
论坛：http://www.xboot.org
淘宝：http://armeasy.taobao.com
1688：http://armeasy.1688.com

QQ 群：
x4412/ibox4412一群：　【16073601】
x4412/ibox4412 二群：　【211126231】
x4418/ibox4418 论坛：　【199358213】
x6818/ibox6818 论坛：　【189920370】
x210/i210 一群：　【23831259】
x210/i210 二群：　【211127570】
x3288 技术论坛：　【159144256】

热烈欢迎广大同仁扫描右侧九鼎创展官方公众微信号，关注有礼，您将优先得知九鼎创展最新动态！

目录

第1章 QT4.8 移植

1.1 交叉编译器的安装

将光盘中的交叉编译工具 arm-2009q3.tar.bz2 复制到 ubuntu 的任意目录并解压：

```
cp  $yourcrosscompiledir/arm-2009q3.tar.bz2  ~
tar xvf  arm-2009q3.tar.bz2  -C  /
```

这时，交叉编译工具会被安装到/usr/local/arm/目录。

交叉编译器环境变量的设置：

编辑/etc/profile，在最末尾处增加：

```
export PATH=/usr/local/arm/arm-2009q3/bin/:$PATH
```

执行如下指令让环境变量生效：

```
source  /etc/profile
```

1.2 安装 QT4.8 源码包

将光盘中的 QT4.8 源码包 qt_x210v3_130712.tar.bz2 拷贝到 ubuntu 的用户目录并解压：

```
cp  qt_x210v3_130712.tar.bz2  ~
tar  xvf  qt_x210v3_130712.tar.bz2
```

这时，源码包已经安装到用户目录，内容如下：

```
qt_x210v3/xboot/tools/qemu-system-arm/linux/realview-run.sh
qt_x210v3/xboot/tools/qemu-system-arm/linux/sdcard.zip
lqm@9tripod-server:~/x210$ ls
qt_x210v3  qt_x210v3_130712.tar.bz2  X210_A  X210_B  X210_B.iso  x210_ics_rtm_
lqm@9tripod-server:~/x210$ cd qt_x210v3/
lqm@9tripod-server:~/x210/qt_x210v3$ ls
buildroot  kernel  mk  uboot  version  xboot
lqm@9tripod-server:~/x210/qt_x210v3$
lqm@9tripod-server:~/x210/qt_x210v3$
lqm@9tripod-server:~/x210/qt_x210v3$ cd buildroot/
lqm@9tripod-server:~/x210/qt_x210v3/buildroot$ ls
arch  board  boot  CHANGES  Config.in  Config.in.legacy  configs  COPYING  dl
lqm@9tripod-server:~/x210/qt_x210v3/buildroot$ cd ..
lqm@9tripod-server:~/x210/qt_x210v3$ ls
buildroot  kernel  mk  uboot  version  xboot
lqm@9tripod-server:~/x210/qt_x210v3$
lqm@9tripod-server:~/x210/qt_x210v3$
lqm@9tripod-server:~/x210/qt_x210v3$
lqm@9tripod-server:~/x210/qt_x210v3$
lqm@9tripod-server:~/x210/qt_x210v3$
lqm@9tripod-server:~/x210/qt_x210v3$
lqm@9tripod-server:~/x210/qt_x210v3$ ls
buildroot  kernel  mk  uboot  version  xboot
lqm@9tripod-server:~/x210/qt_x210v3$
```

其中，mk 为编译脚本，uboot 和 xboot 为引导内核用的 bootloader，kernel 目录为 linux 内核目录，buildroot 目录为文件系统目录，version 给出了当前源码版本信息。

1.3 编译 uboot

注意，引导 qt 的 uboot 和引导 android 的 uboot 有一点差异，nand flash 分区做了调整，预了解详细差异，用户可对比源码。

注意：编译前，请务必确认手上的开发板是 nand flash 平台还是 inand 平台。

如果是 nand flash 平台，执行如下指令编译:

```
./mk  -un
```

这时，在 uboot 目录将会生成我们需要的映像 uboot_nand.bin，同时会拷贝到映像所释

放的目录，即 release 目录。

如果是 inand 平台，执行如下指令编译：

```
./mk   -ui
```

这时，在 uboot 目录将会生成我们需要的映像 uboot_inand.bin，同样也会将映像拷贝到 release 目录。

1.4 编译内核

执行如下指令编译内核：

```
./mk   -k
```

注意：无论是 inand 平台还是 nand flash 平台，都可以用同一个 zImage 文件，用同一个内核。

这时，在 kernel/arch/arm/boot 下会生成我们需要的映像 zImage，同时会将 QT 的内核映像 zImage-qt 拷贝到 release 目录。注意，这时 release 目录还会生成一个 zImage-initrd 的映像，它主要用于 xboot 升级映像，使用 uboot 时一定要烧写 zImage-qt 这个映像。

1.5 编译 xboot

我们精心编写了一套自主研发的 bootloader，无论在代码管理，还是固件升级，批量生产，以及代码调试方面，相对 uboot 都有很强的优势。xboot 最终生成的映像有十几 M，甚至更大，而 uboot 只有几百 K 大小，这是因为 xboot 它本身已经将前面的 zImage-qt 和 zImage-initrd 都打包到 xboot 中了，甚至还可以将 WINCE 的 eboot 也打包到 xboot 中。详细原理，可认真琢磨 xboot 源码。也正因为 xboot 的这种机制，因此务必先编译内核，再编译 xboot。而且，用户在更新内核驱动后，如果使用 xboot 机制，需将新的内核映像打包到 xboot，再更新 xboot 到开发板。

执行如下指令编译 xboot：

```
./mk   -x
```

编译完成后，xboot 会被拷贝到 release 目录。

1.6 编译文件系统

我们使用 buildroot 工具配置文件系统，在 QT 源码根目录下执行如下指令编译文件系统：

```
./mk   -r
```

编译完成后，我们需要的文件系统文件 rootfs.tar 存放在 buildroot/output/images 目录，编译脚本 mk 会自动将它复制到 release 目录中。有了 rootfs.tar，我们就能制作我们想要的文件系统映像了。

1.7 制作文件系统

1.7.1 制作基于 xboot 的文件系统

说明：xboot 只支持 inand 或 SD 卡存储，并不支持 nand flash 存储。

基于 xboot 的文件系统是通过 mkisofs 文件将 xboot 和 QT 文件系统打包成一个 bin 文件，这个 bin 文件最终可通过 xboot 使用 SD 卡更新到开发板的 inand 中。

执行如下指令制作基于 xboot 的文件系统：

```
sudo   ./mk   -U
```

注意，制作文件系统时请使用 root 权限。

制作完成后，在 release 目录下会生成 qt-update.bin 文件，即我们最终需要的更新文件。

1.7.2 制作基于 uboot 的文件系统

执行如下指令制作 inand 平台文件系统：

```
./mk   -re
```

这时，将会生成基于 inand 平台的文件系统映像 rootfs_qt4.ext3。

执行如下指令制作 nand flash 平台文件系统：

```
./mkfs   -rj
```

这时，将会生成基于 nand 平台的文件系统映像 rootfs_qt4.jffs2。

注意，如果没有 mkfs.jffs2 指令，可在 rehat 的 ftp 上下载：ftp://sources.redhat.com/pub/jffs2/mkfs.jffs2

下载后放到 ubuntu 文件系统的/sbin 目录就可以使用了。

1.8 烧写 uboot

注意：烧写 uboot 时，无论是 inand 平台还是 nand 平台，烧写方法都是相同的，请务必烧写对应的映像到开发板！如果出现 inand 平台烧 nand 映像或是 nand 平台烧 inand 映像，将无法再使用烧进去的 bootloader 更新，需在 WINDOWS 平台下，将 OM5 和 OM0 拨到高电平，再使用 DNW 工具先将正确的 uboot 映像下载到开发板的内存后再更新映像。

第一次烧写映像时，这时开发板上并没有 bootloader，有四种方法烧写 uboot。

1.8.1 将 uboot 烧写到 SD 卡上，再通过 fastboot 更新

准备一张 SD 卡，插到运行 ubuntu 系统的 PC 机上，使用命令终端进入 uboot 的 sd_fusing 目录，编译出基于 nand flash 的 bootloader 映像，再执行./nand_fusing.sh 脚本：

```
sudo ./nand_fusing.sh
```

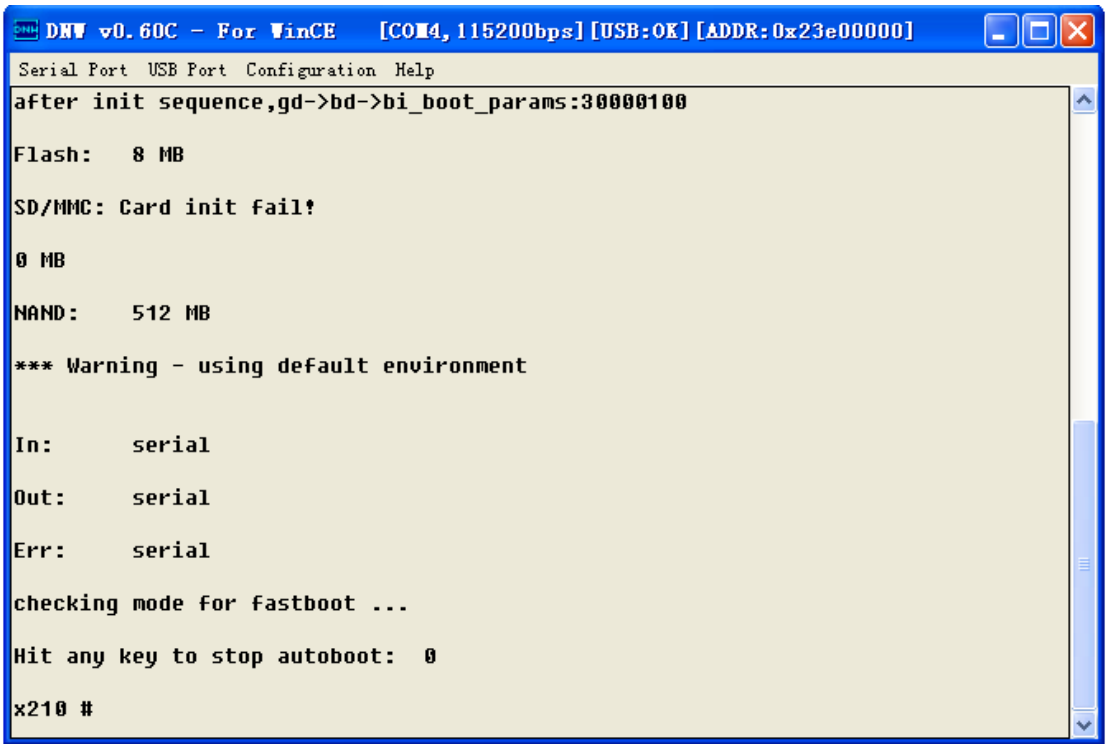
注意，如果是 inand 平台，则使用 sd_fusing.sh 脚本，同时需编译出基于 inand 的 bootloader 映像。

1.8.2 通过 DNW 将 uboot 烧写到 RAM 中，再通过 fastboot 更新

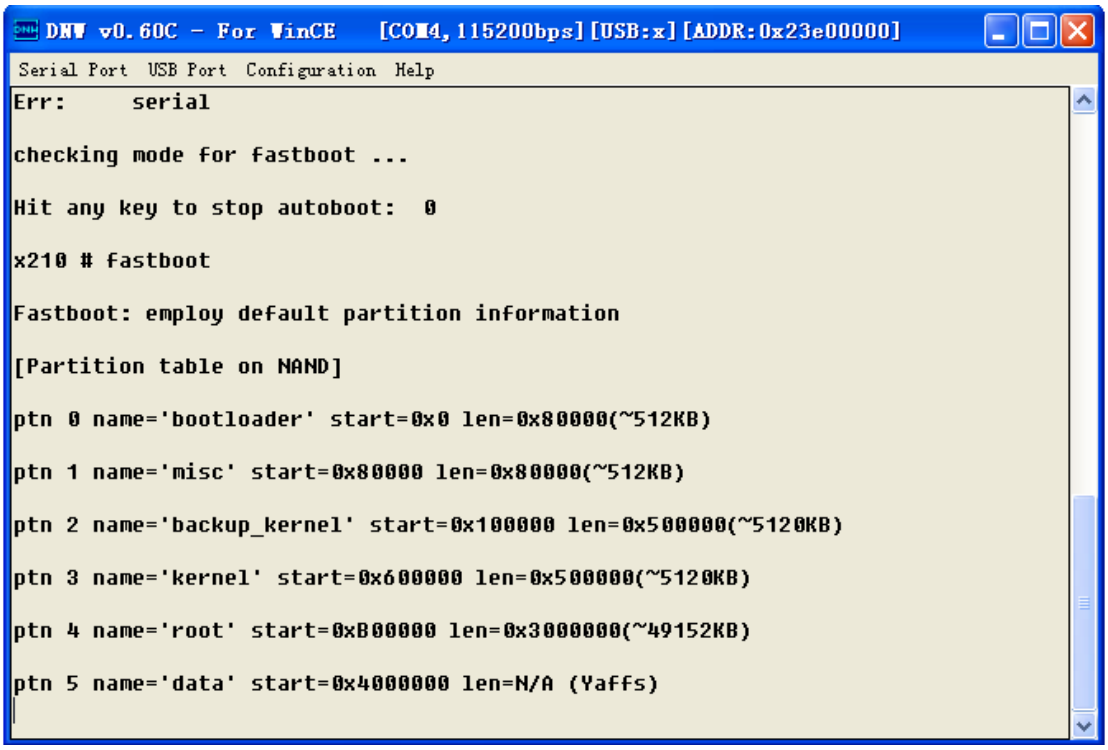
将开发板的拨码开关 OM5 和 OM0 拨到 1，连接 USB OTG 口到 PC 机，在 windows xp 系统下打开 DNW0.6C，连接串口，将 ADDR 设置为 0xd0020010，如下图所示：

打开开发板电源，按住 POWER 键(SW12)不放，这时，DNW 上的 USB 会显示 OK，再点击 USB Port->Transmit，找到 x210_usb.bin 文件，双击，这时，DNW 上的 USB 会显示 x 后再显示 OK，表明 210 芯片的寄存器已经初始化完毕。

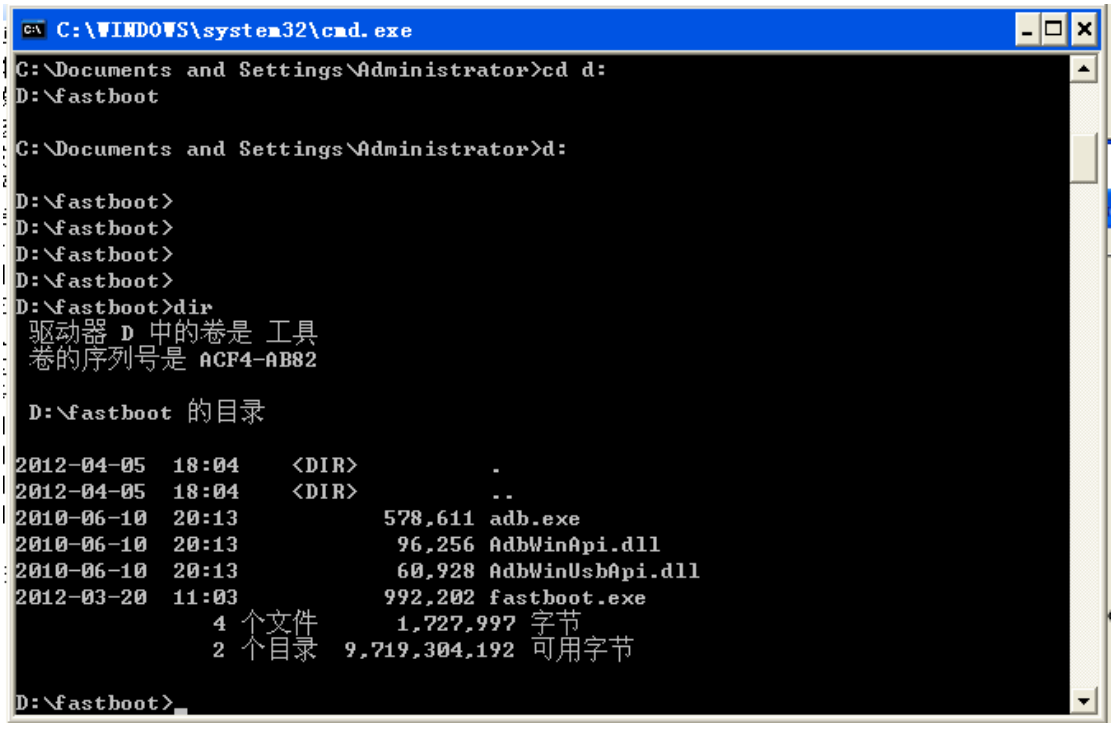
保持按住 POWER 键不放，再次将 DNW 的地址设置为 0x23e00000，点击 USB Port->Transmit，找到 QT4.8 目录下的 uboot.bin 文件，双击，这时 uboot 在 0x23e00000 处运行起来了：



执行 fastboot 指令，弹出界面如下：

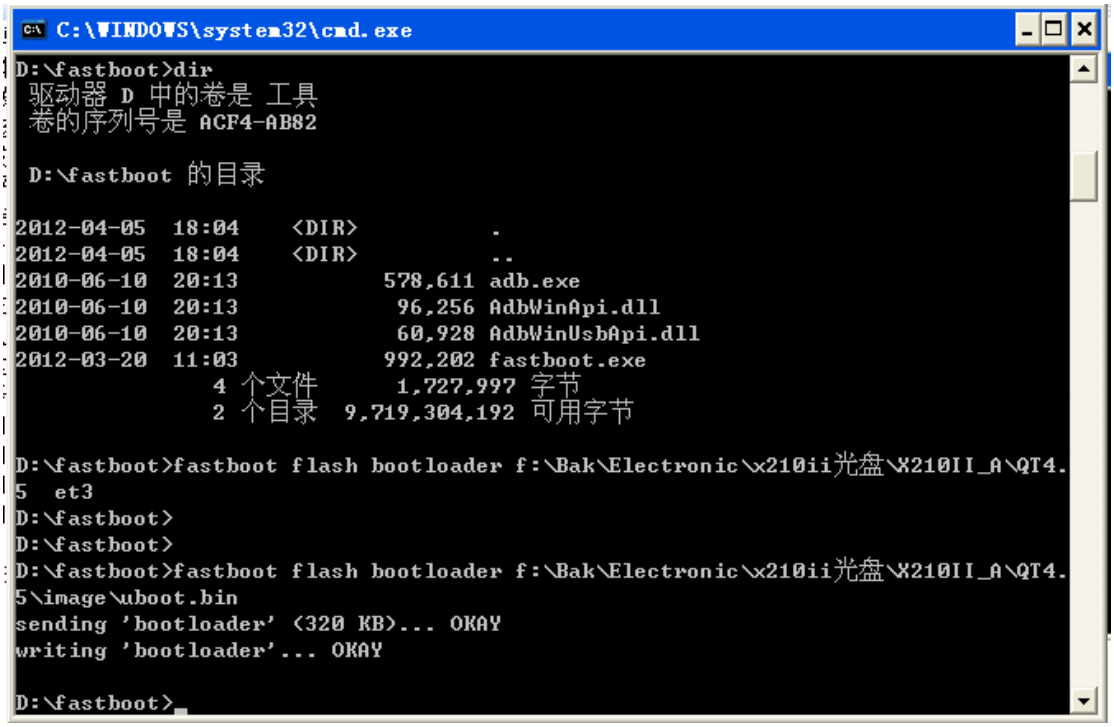


开启 windows 下的 DOS 界面，进入 fastboot 目录，如果硬盘上没有 fastboot 目录，从光盘中拷备过来：



执行如下指令下载 bootloader 到 nand flash:

```
fastboot flash bootloader uboot.bin
```



此时，uboot 已经烧录到 nand flash 中。将拨码开关拨到从 nand flash 启动，启动开发板，发现 uboot 已经运行起来了。

注意：当您的系统首次使用 DNW 和 fastboot 烧写映像时，需要安装 DNW 和 fastboot 驱动，两个驱动都要安装，而且这两个驱动并不相同！

1.8.3 将 uboot 烧写到 SD 卡上，再通过 uboot 指令更新

方法大同小异，请读者自行尝试。

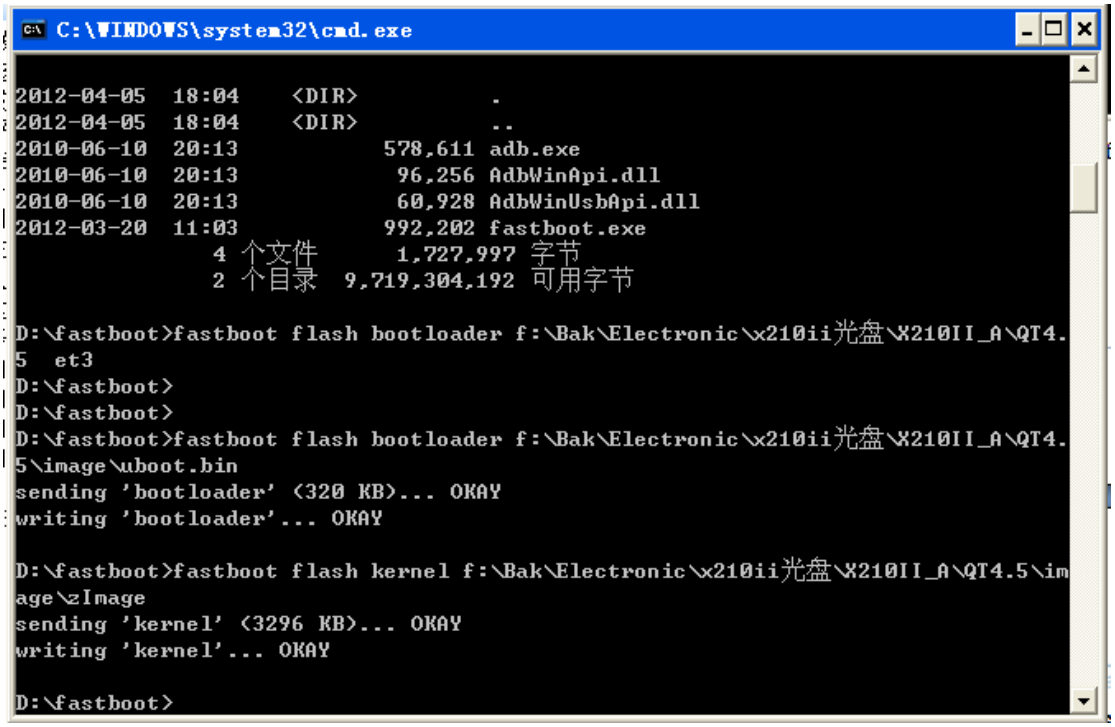
1.8.4 通过 DNW 将 uboot 烧写到 RAM 中，再通过 uboot 指令更新

方法大同小异，请读者自行尝试。

1.9 烧写内核

启动 uboot，在 3 秒倒计时期间按下空格键，进入 uboot 命令行，输入 fastboot 进入 fastboot 烧录界面。开启 windows 下的 DOS 界面，执行如下指令更新内核：

```
fastboot  flash  kernel  zImage-qt
```



也可以使用 uboot 的指令烧写内核，读者可自行尝试。

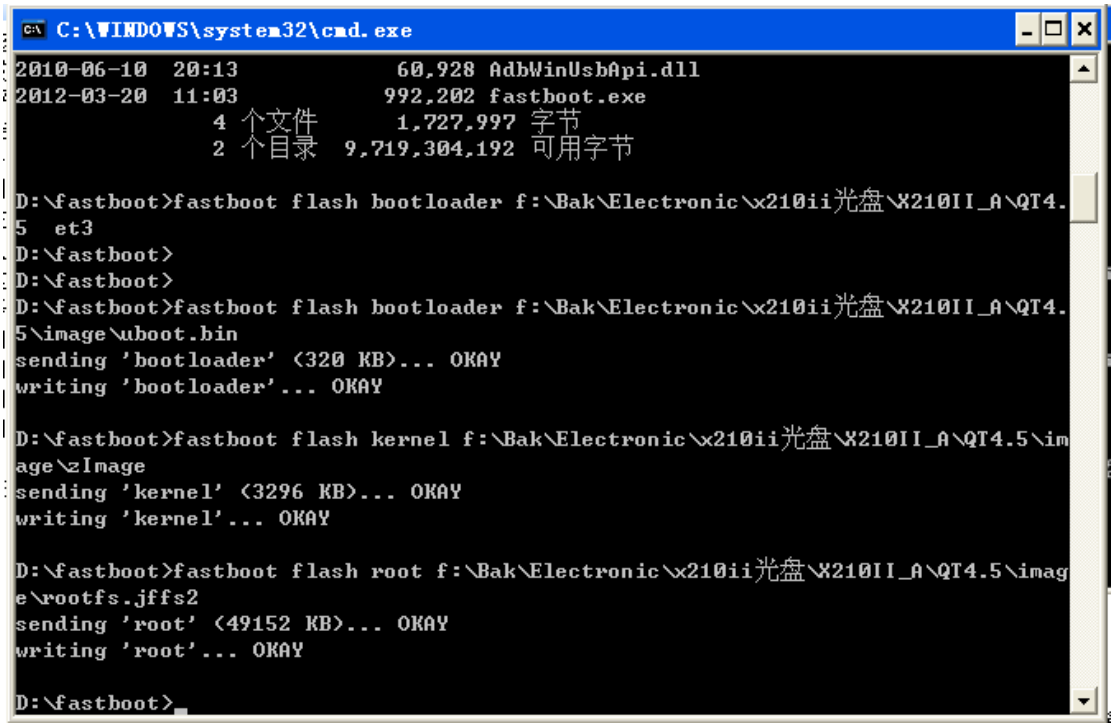
1.10 烧写文件系统

启动 uboot，在 3 秒倒计时期间按下空格键，进入 uboot 命令行，输入 fastboot 进入 fastboot 烧录界面。开启 windows 下的 DOS 界面，执行如下指令更新 nand 平台文件系统：

```
fastboot flash system rootfs_qt4.jffs2
```

执行如下指令更新 inand 平台文件系统：

```
fastboot flash system rootfs_qt4.ext3
```



1.11 使用 xboot 通过 SD 卡更新文件系统

1.11.1 制作量产卡

第一步：准备一张不小于 2G 的 SD 卡，接在 Linux 系统的 USB 端口；
第二步：在 Linux 的终端窗口，使用 fdisk /dev/sdb 命令删除原有的所有分区，sdb 为系统为 SD 卡分配的设备节点，具体由节点名称而定，有可能是 sdc,sde 等。使用如下指令查询设备节点：

```
cat /proc/partitions
```

示例如下：

```
[root@lqm mass-production]# cat /proc/partitions
major minor  #blocks  name

   8        0   36700160 sda
   8        1     512000 sda1
   8        2   36187136 sda2
 253        0   34144256 dm-0
 253        1    2031616 dm-1
   8       16    3879936 sdb
   8       17    3875840 sdb1
[root@lqm mass-production]#
[root@lqm mass-production]# fdisk /dev/sdb

WARNING: DOS-compatible mode is deprecated. It's strongly recommended to
         switch off the mode (command 'c') and change display units to
         sectors (command 'u').
```

```
Command (m for help): d
Selected partition 1

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.

WARNING: Re-reading the partition table failed with error 16:  设备或资源忙.
The kernel still uses the old table. The new table will be used at
the next reboot or after you run partprobe(8) or kpartx(8)
Syncing disks.
[root@lqm mass-production]#
```

输入 d，表示删除分区，输入 w 表示保存已经修改的分区信息。至此，原/dev/sdb1 被删除。拔掉 SD 卡，再插入 PC 机上，查询设备节点：

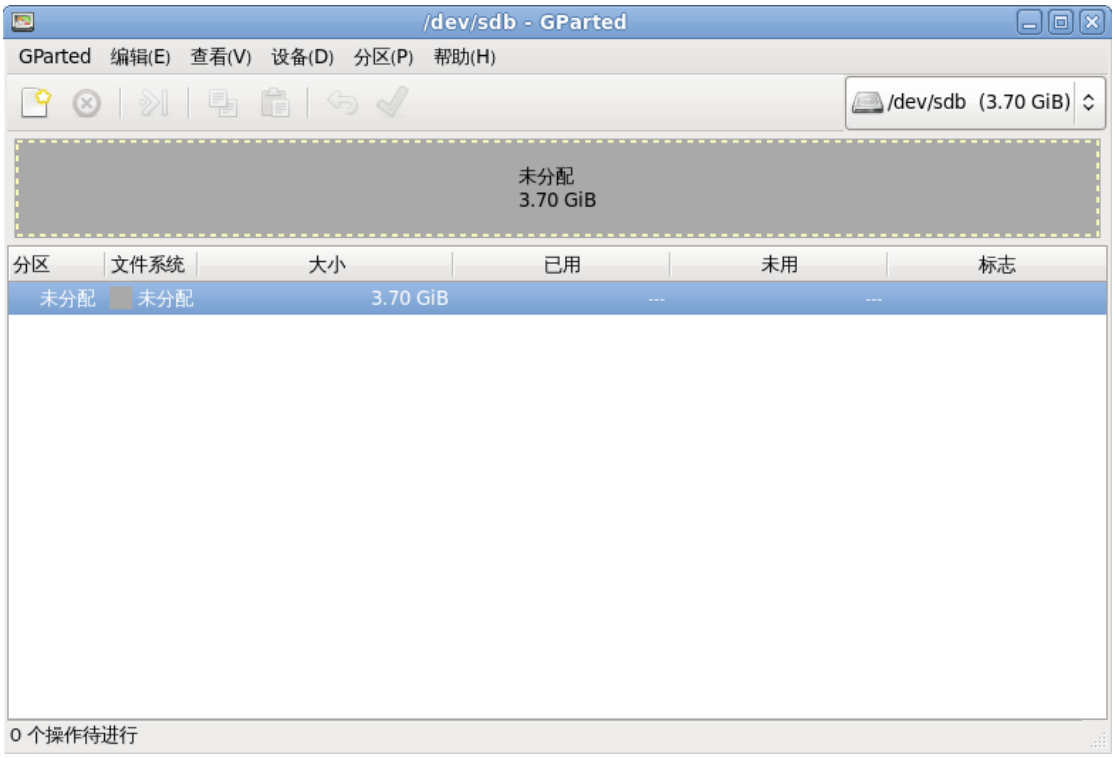
```
[root@lqm mass-production]# cat /proc/partitions
major minor  #blocks  name

   8        0   36700160 sda
   8        1     512000 sda1
   8        2   36187136 sda2
 253        0   34144256 dm-0
 253        1    2031616 dm-1
   8       16    3879936 sdb
[root@lqm mass-production]#
```

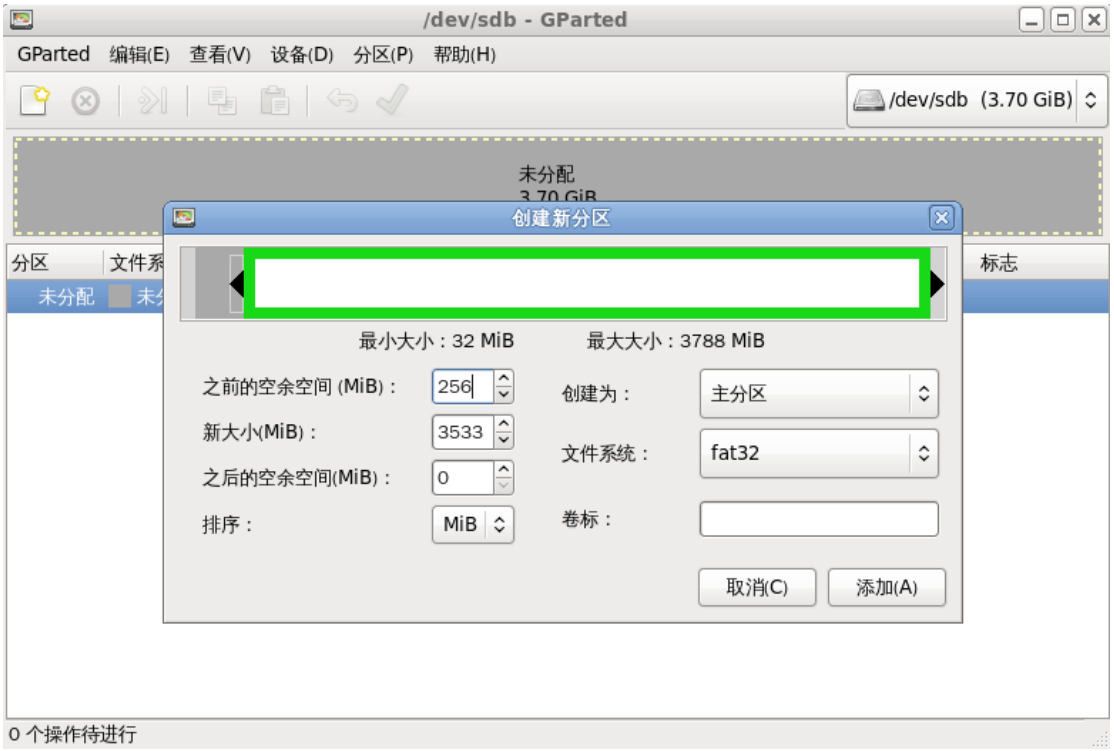
注意必须拔掉后再插入，否则仍然会提示存在/dev/sdb1 节点，会造成出错。

第四步：使用如下命令格盘：

```
gparted  /dev/sdb
```



选择分区->新建，预留 256M 空间给 xboot 和内核，剩下的分区使用 fat32 格式，如下图所示：



点击添加，选择菜单中的应用全部操作，完成 SD 卡的分区。

第五步：烧写 bootloader，运行命令 sudo ./x210-irom-sd.sh /dev/sdb xboot.bin，示例如下：

```
[root@lqm mass-production]# ./x210-irom-sd.sh /dev/sdb release/xboot.bin
记录了 24084+1 的读入
记录了 24085+0 的写出
```

```
12331520 字节(12 MB)已复制，12.0886 秒，1.0 MB/秒
^_^ The image is fused successfully
[root@lqm mass-production]#
```

第六步：将升级文件 qt-update.bin 拷贝到 fat32 分区，在 linux 下或 windows 下均可。然后在 SD 卡根目录下，创建一个名为 backdoor 的文件，内容为空即可。linux 下示例如下：

```
[root@lqm mass-production]# mount /dev/sdb1 /mnt
[root@lqm mass-production]# cp res_image/qt-update.bin /mnt
[root@lqm mass-production]# vim /mnt/backdoor
[root@lqm mass-production]# ls /mnt
backdoor    qt-update.bin
[root@lqm mass-production]#
```

至此，量产工具制作完成。

注意，上述操作在 ubuntu 或 fedora 下均可操作。

1.11.2 使用量产卡升级映像

第一步：将量产卡插入待烧写机器的外部 SD 卡插槽,并确认机器已插入待烧写系统卡；
第二步：按着 LEFT 键 + POWER 键开机，直到出现选择菜单界面；
第三步：通过上下键选择菜单，这里选择第二项，update system；
第四步：按确认键，等待即可；
第五步：更新完成后，系统会自动重新启动。

注意：qt-update.bin 文件已包括 xboot.bin,zImage 以及 rootfs_qt4.ext3 三个映像文件，执行上述操作，相当于更新了整个映像文件。量产卡制作一遍后，后续升级时只需将 qt-update.bin 拷贝到量产卡的根目录即可，当然还要确保根目录下存在 backdoor 文件。

1.11.3 更新 xboot 映像文件

将要更新的映像文件 xboot.bin 拷贝到脚本 x210-irom-sd.sh 所在目录，执行如下命令：
sudo ./x210-irom-sd.sh /dev/sdb xboot.bin

注意，/dev/sdb 为 SD 卡的盘符结点，结点名称因环境而异，有时为 sdc,sde 等，具体对应起来即可。在编译完整个 qt 源码后，编译脚本会在源码根目录创建 release 目录，同时会将映像文件 xboot.bin,qt-update.bin 等拷贝到该目录，因此推荐将烧写脚本 x210-irom-sd.sh 放在该目录，这时每次更新完相应的映像文件后，我们只需到该目录执行相关指令就可完成升级。

1.11.4 更新 kernel 映像文件

x210 映像包已将调试内核和 qt 内核打包到了 xboot 中，因此在修改内核后，在 release 下会生成 zImage-initrd 和 zImage-qt 两个内核映像包，需再编译 xboot，将两个内核打包到 xboot 映像中，再将 xboot.bin 烧到 SD 卡中启动，即完成内核映像的更新。具体步骤如下：
第一步：执行脚本编译内核：

```
./mk -k
```

将会在 out/release 目录下生成内核映像文件 zImage-initrd 和 zImage-android
第二步：执行脚本编译 xboot:

```
./mk -x
```

将会在 release 目录下生成 xboot 映像文件 xboot.bin，注意，这时 xboot.bin 已包含了第

一步编译出来的 zImage-initrd 和 zImage-qt 了。

第三步：将启动用的 SD 卡通过读卡器插到 linux 平台的 PC 机上，执行如下指令更新 xboot:

```
sudo ./x210-irom-sd.sh /dev/sdb xboot.bin
```

其中 sdb 需和 PC 机上生成的设备节点对应。如设备节点为 sdc，需更改为/dev/sdc.

说明：第一步和第二步可以直接执行指令./mk –x –k 替代。

如果使用 uboot 引导内核，只需要单独更新 zImage-qt 即可。

1.12 登录 QT4.8 文件系统控制台

进入文件系统后，访问控制台需要用户名和密码。

用户名：root

密码：123456

输入正确的用户名和密码即可访问。

1.13 使用电容触摸屏操作 UI 界面

电阻触摸屏和电容触摸屏有一些差异，相对映像，仅体现在文件系统上，与 bootloader 和内核映像无关。也就是说，同一个 bootloader 和内核，同时支持电阻触摸和电容触摸，所不同的是，我们需要在文件系统中来配置相关选项。默认配置情况下，支持电容触摸。即烧写出厂的映像文件后，电容触摸屏直接可用。而且我们已经将电容屏的校正文件打包在文件系统中，开机无须再校屏了。这里我们主要讲解电阻触摸屏如何操作 UI 界面。

1.14 使用电阻触摸屏操作 UI 界面

由于文件系统默认是电容触摸屏的配置，而且/etc 目录下的校屏文件是针对电容屏的，所以烧写默认映像后，电阻屏完全无法操作。为了方便，我们将电阻屏的校屏文件保存为/etc 目录下，只是将名称重命名为 pointercalres。这时，我们只需要重新配置一下环境变量就可以使用电阻屏操作 UI 界面了。

第一步：将/etc 目录下针对电阻触摸屏的脚本 qtopia 复制到/bin 目录：

cp /etc/S99qttest init.d

```
[root@x210ii ~]# cd /etc/
[root@x210ii etc]# ls
S99qttest*              network/                ssh_config
fstab                   nsswitch.conf           ssh_host_dsa_key
group                   os-release              ssh_host_dsa_key.pub
hostname                passwd                  ssh_host_ecdsa_key
hosts                   pointercal              ssh_host_ecdsa_key.pub
init.d/                 pointercalres           ssh_host_key
inittab                 profile                 ssh_host_key.pub
inputrc                 protocols               ssh_host_rsa_key
issue                   random-seed             ssh_host_rsa_key.pub
ld.so.cache             resolv.conf@            sshd_config
ld.so.conf              securetty               ssl/
ld.so.conf.d/           services                ts.conf
mdev.conf               setenv_cap*             wpa_supplicant.conf
moduli                  setenv_res*
mtab@                   shadow
[root@x210ii etc]#
[root@x210ii etc]#
[root@x210ii etc]# cp S99qttest  init.d/
[root@x210ii etc]#
```

第二步：重启开发板，即可正常使用电阻屏操作 UI 界面了。

1.15 使用电容触摸屏操作 tslib

在电容触摸屏能够正常操作 UI 的前提下，，执行如下指令：

```
ts_test
```

可以看到，有如下错误提示：

```
[root@x210ii etc]# ts_test
tslib: Selected device is not a touchscreen (must support ABS and KEY event types)

signal 2 caught
[root@x210ii etc]#
[root@x210ii etc]#
```

这是因为没有设置 TSLIB 的相关环境变量的缘故。执行如下指令声明环境变量：

```
cd    /etc
source    ./setenv_cap
```

再执行 ts_test 命令，即可正常操作了，示例如下：

```
[root@x210ii etc]# source ./setenv_cap
set tslib environment...
[root@x210ii etc]#
[root@x210ii etc]#
[root@x210ii etc]#
[root@x210ii etc]#
[root@x210ii etc]# ts_test
373.476175:    400    255      1
373.486196:    405    255      1
373.499988:    411    254      1
```

注意，我们将环境变量做成脚本，是为了同时兼容电阻屏和电容屏，如果您只使用一种触摸屏，可以将脚本内容写死在/etc/profile 文件中，这样机器重新启动后，就没有必要手动运行脚本了。

1.16 使用电阻触摸屏操作 tslib

在电阻触摸屏能够正常操作 UI 的前提下，执行如下指令：

```
ts_test
```

可以看到，有如下错误提示：

```
[root@x210ii ~]# ts_test
tslib: Selected device is not a touchscreen (must support ABS and KEY event types)
```

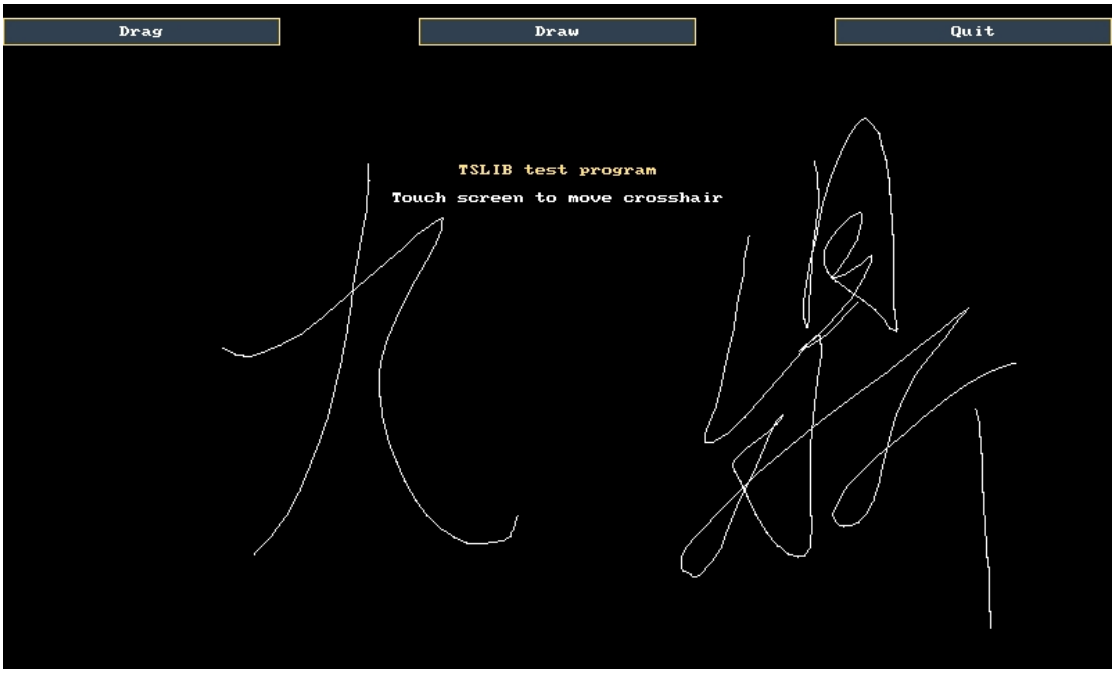
这是因为没有设置 TSLIB 的相关环境变量的缘故。执行如下指令声明环境变量：

```
cd    /etc
source    ./setenv_res
```

再执行 ts_test 命令，即可正常操作了，示例如下：

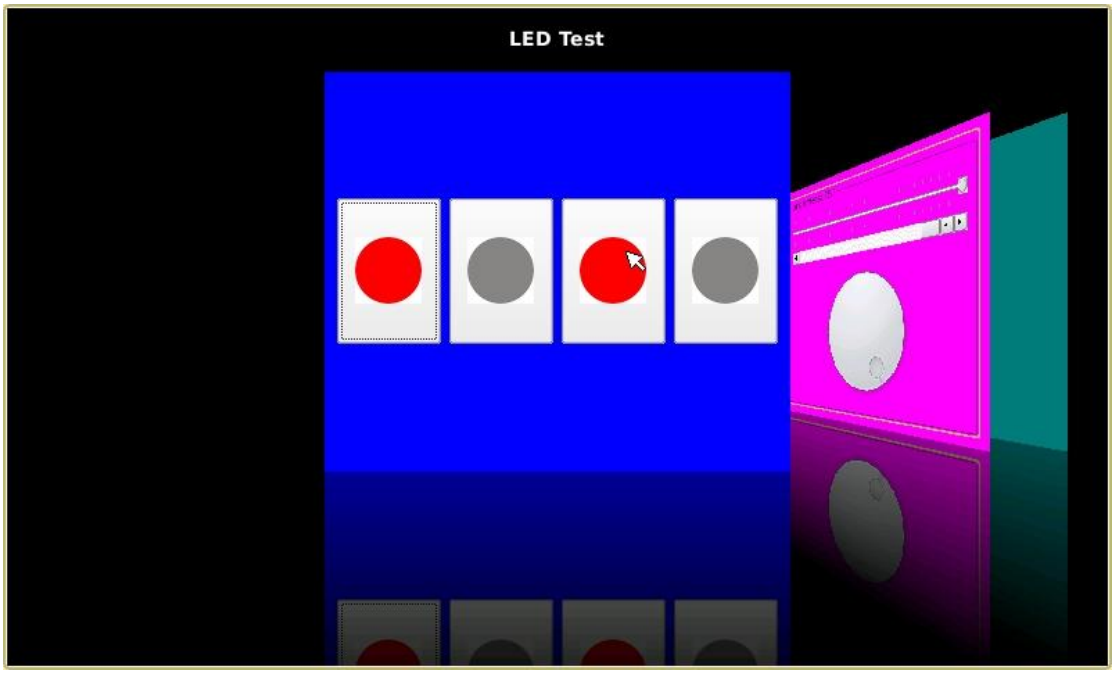
```
[root@x210ii etc]# source ./setenv_res
set tslib environment...
[root@x210ii etc]#
[root@x210ii etc]#
[root@x210ii etc]# ts_test
149.125126:    405    245      1
149.140718:    408    243      1
```

测试界面如下：



1.17 使用 QT_demo 测试 LED

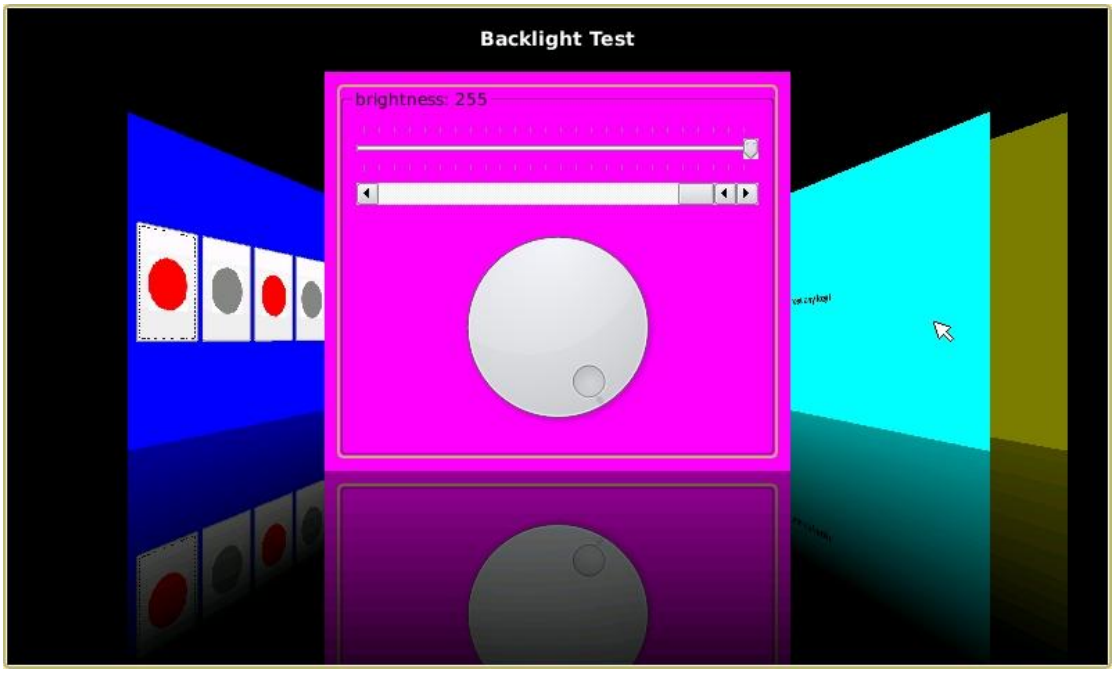
进入 QT4.8 系统后，默认会运行我们自主编写的测试 demo，进入 LED 测试界面，可以测试开发板的四盏 LED 灯。界面如下：



点击图中任意指示灯，暗色对应开发板上 LED 灯灭，红色对应开发板上 LED 灯亮。

1.18 使用 QT_demo 调节背光

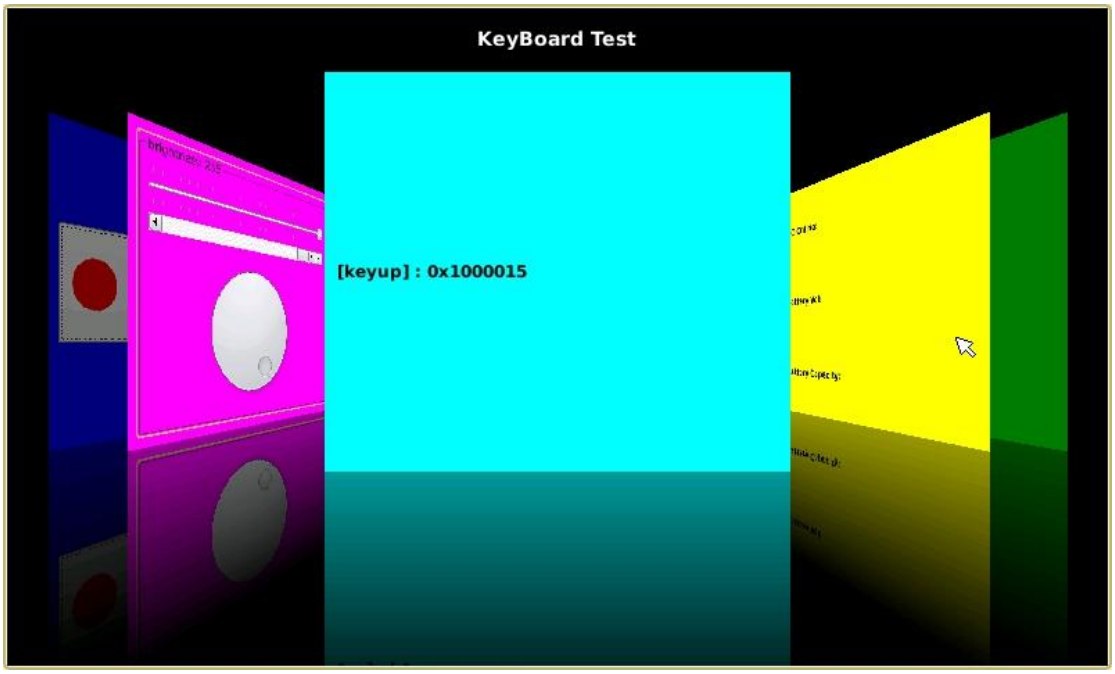
测试界面如下：



滑动圆形滑轮，可对开发板背光进行亮暗调节。

1.19 使用 QT_demo 测试按键

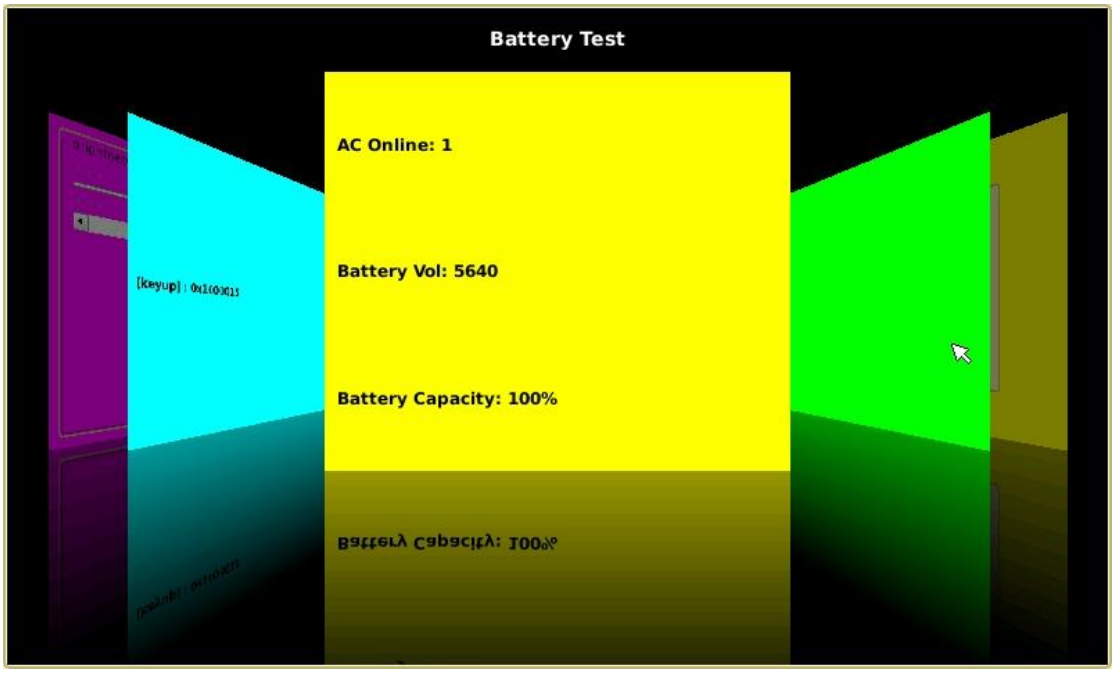
测试界面如下：



按下开发板任一独立按键，图中界面即会显示相应键值，同时，按下时提示[keydown]，抬起时提示[keyup]。

1.20 使用 QT_demo 测试 ADC 电压

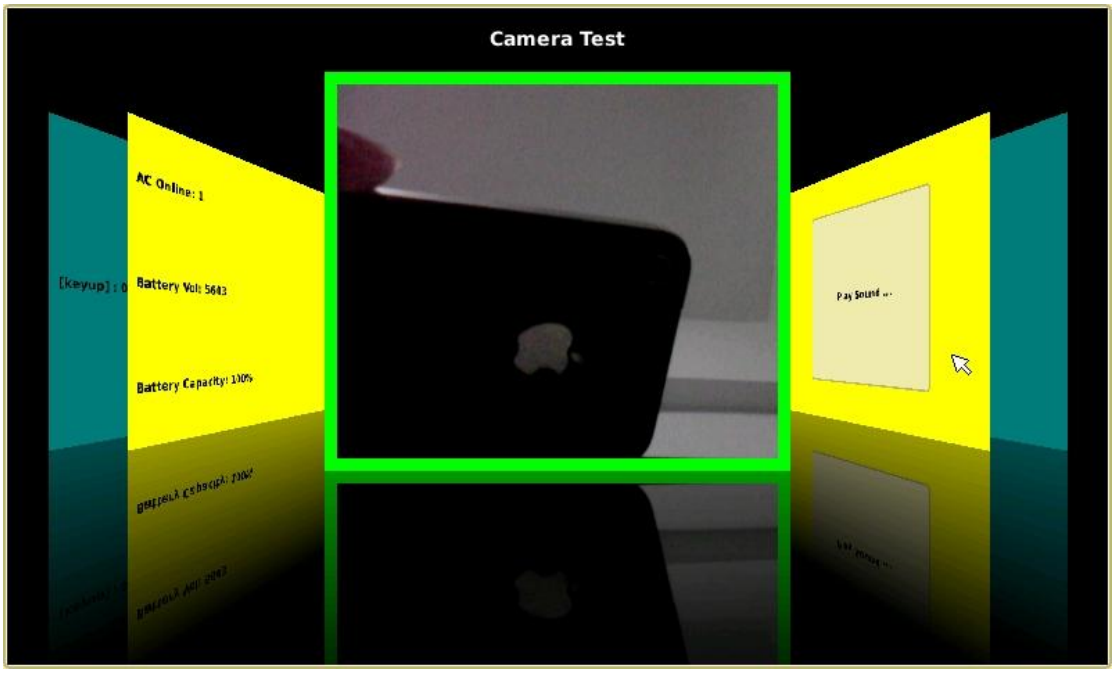
测试界面如下：



使用一字螺丝刀旋转精密电位器上面的旋转按钮，可以看到界面上的 Battery Vol 的值会相应变化，说明这里对电位器上 ADC 电压采样有效。

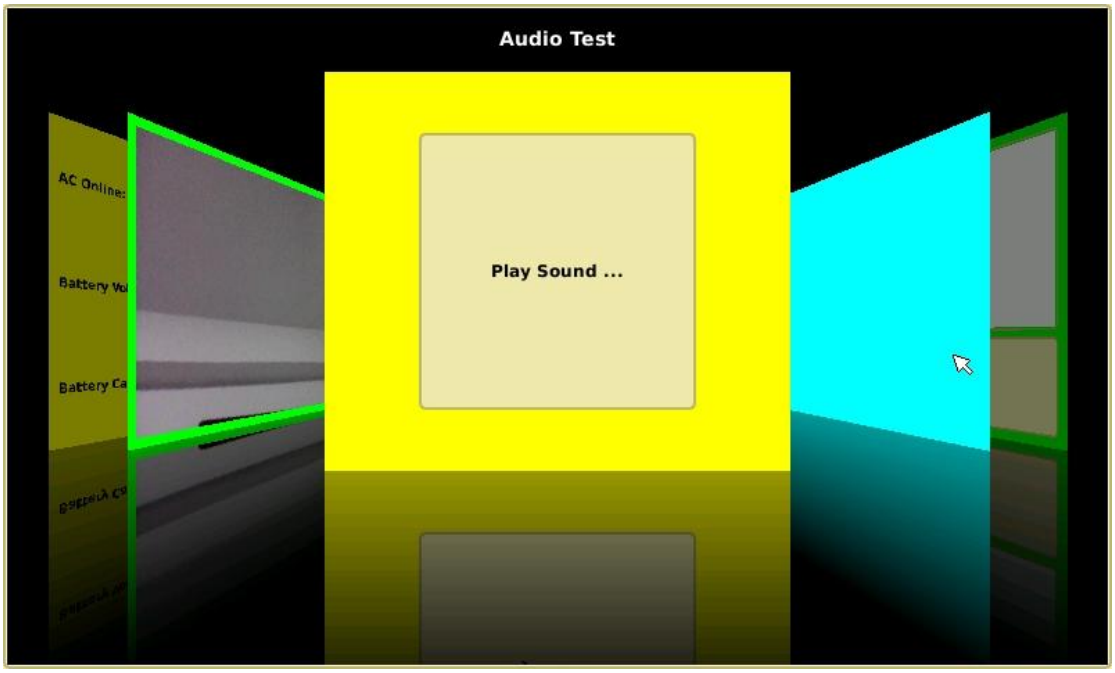
1.21 使用 QT_demo 测试摄相头

将 OV2655 摄相头模组可靠接到开发板上，进入摄相头测试界面，会显示摄相头采集的视频数据，如下图所示：



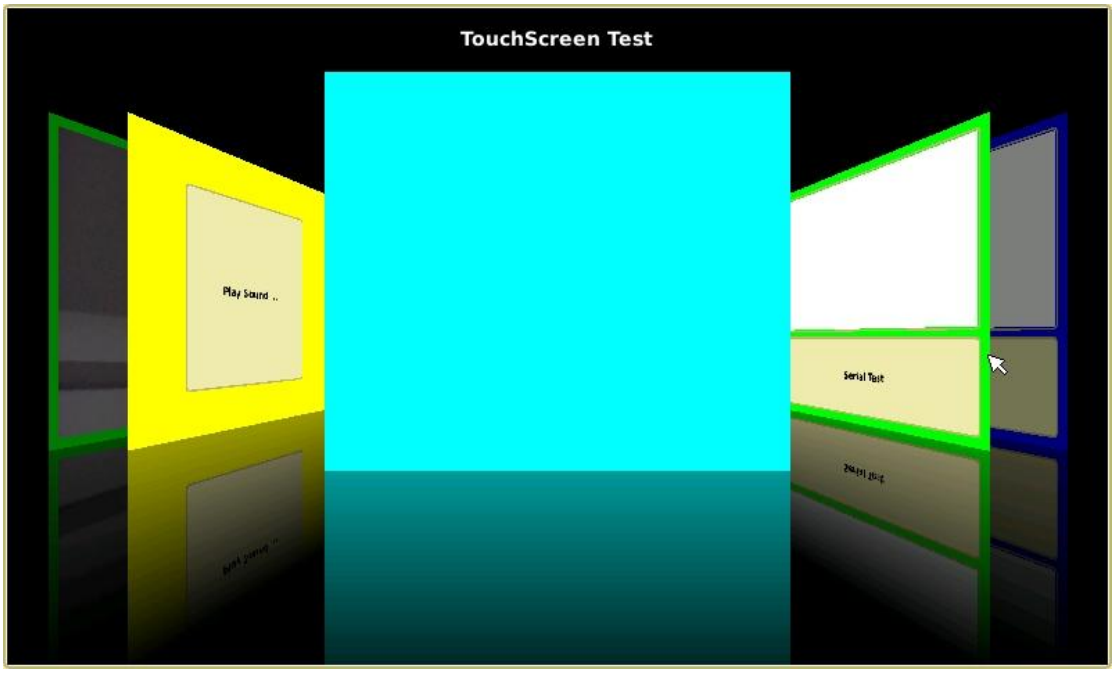
1.22 使用 QT_demo 测试音频

将喇叭或耳机接到开发板的对应接口，点击下图中的 Play Sound 按钮，会播放测试歌曲：

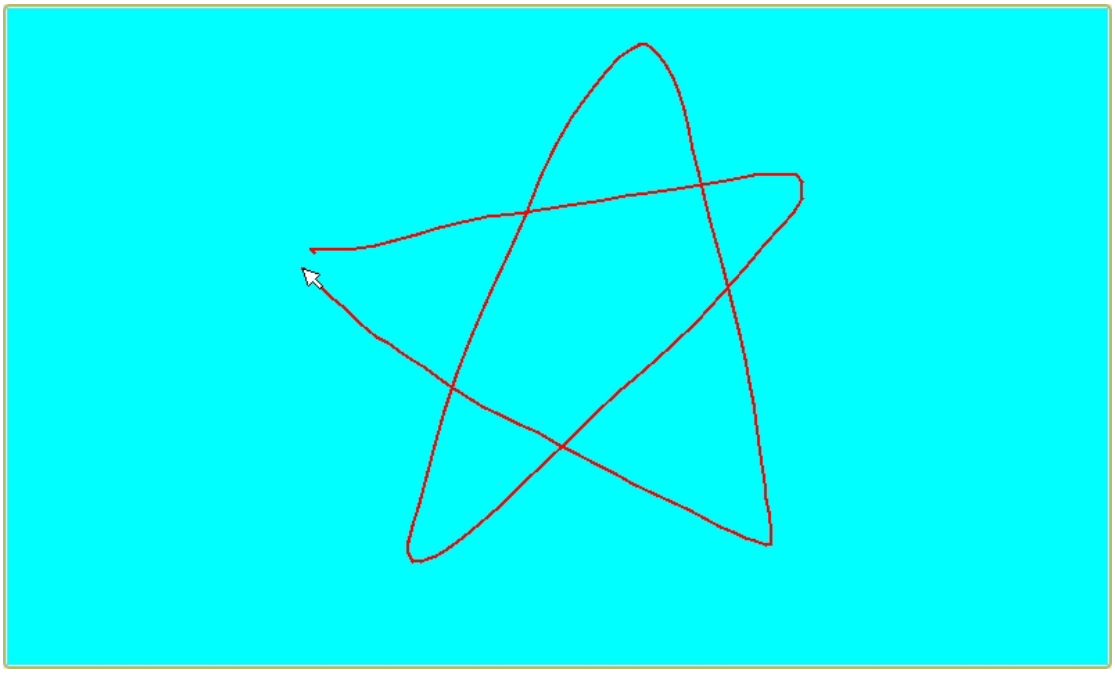


1.23 使用 QT_demo 测试触摸屏

进入如下界面：

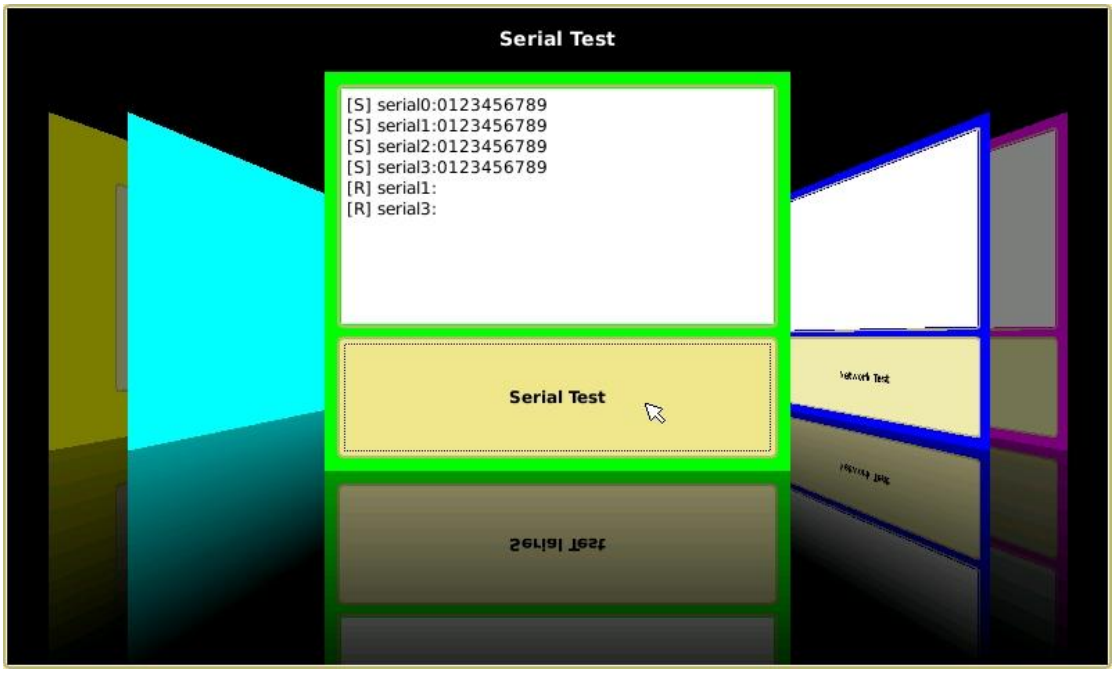


单击绿色矩形框，界面会进入全屏模式，这时我们可以任意书写来测试触摸屏了，测试示例图片如下：



1.24 使用 QT_demo 测试串口

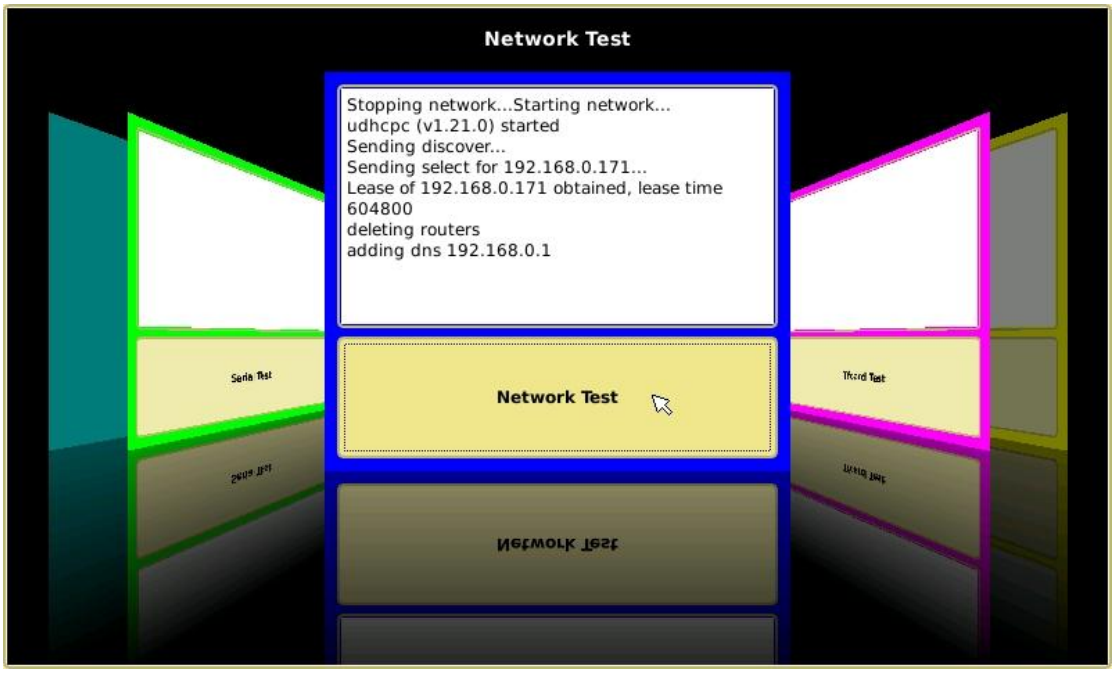
测试界面如下：



S5PV210 共有四组串口，将需要测试的串口的 RXD 和 TXD 短路，再点击图中的 Serial Test 按钮，上图数据框中会进行自发自收测试并提示相应结果。

1.25 使用 QT_demo 测试网络

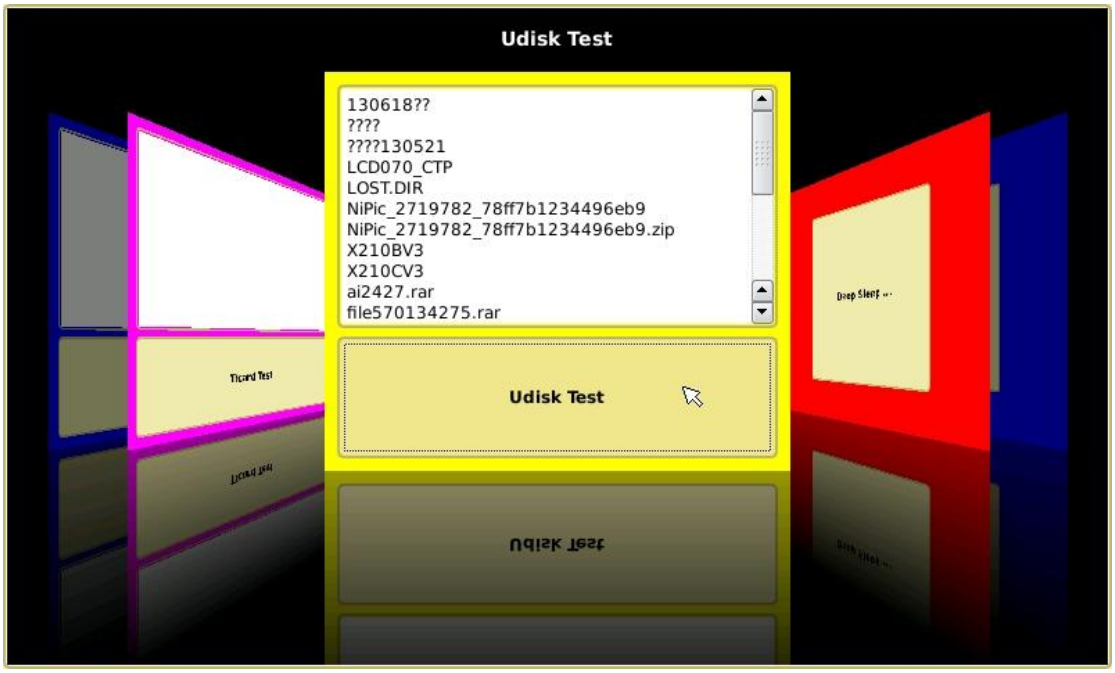
将网线连接开发板的有线以太网接口，点击界面中的 Network Test 按钮，如果网络已经连通，则会添加 DNS，如下图所示：



如果没有连通，则会提示相应错误。

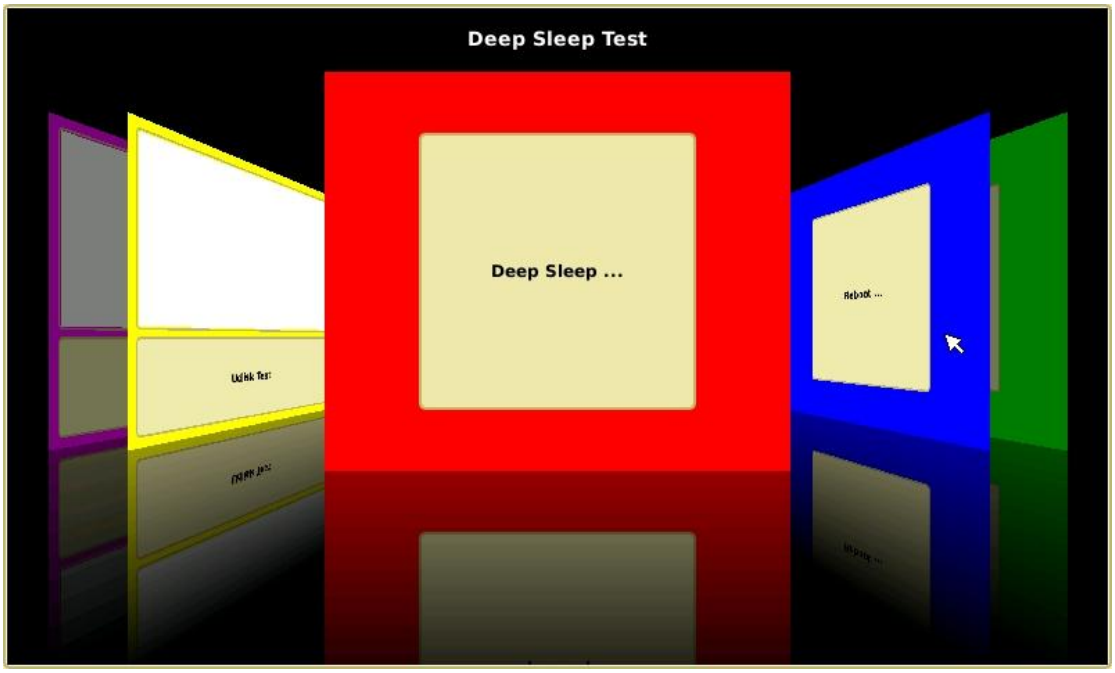
1.26 使用 QT_demo 测试 U 盘

将 U 盘接到开发板的任何一个 USB HOST 接口，点击 Udisk Test，数据框中会列出 U 盘中的数据，如果找不到，则会提示无法 mount U 盘，如图：



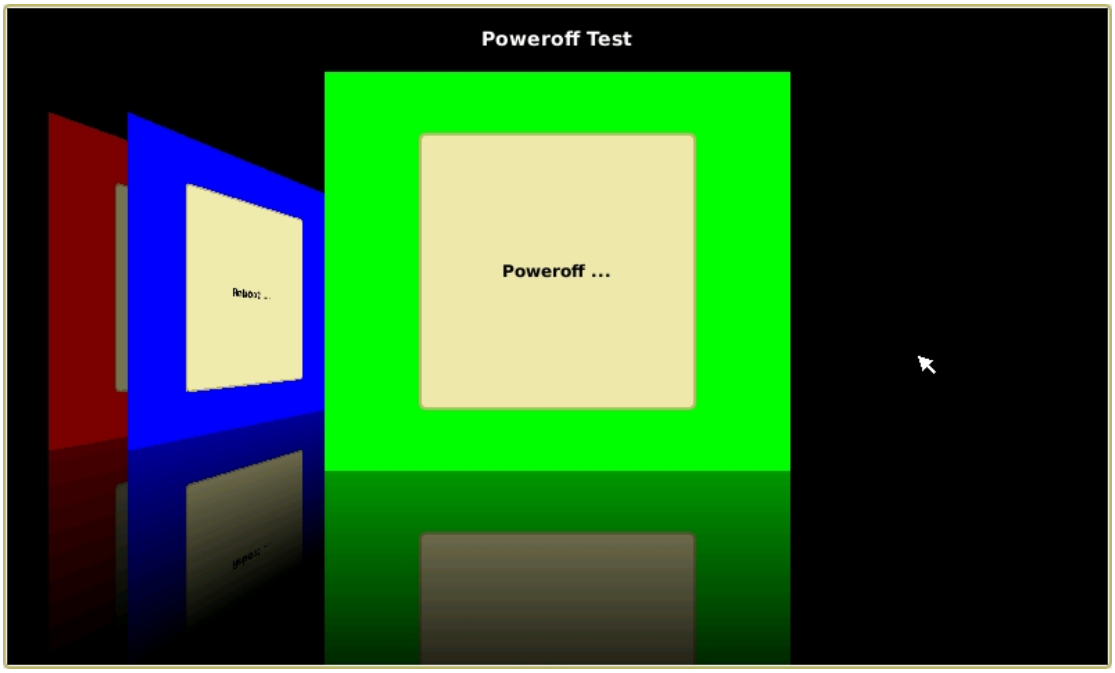
1.27 使用 QT_demo 测试休眠唤醒

点击 Deep Sleep，开发板会进入深度睡眠状态，这时屏幕会全黑，串口终端也将没有任何信息提示，只有按下 POWER 键，方可唤醒开发板。



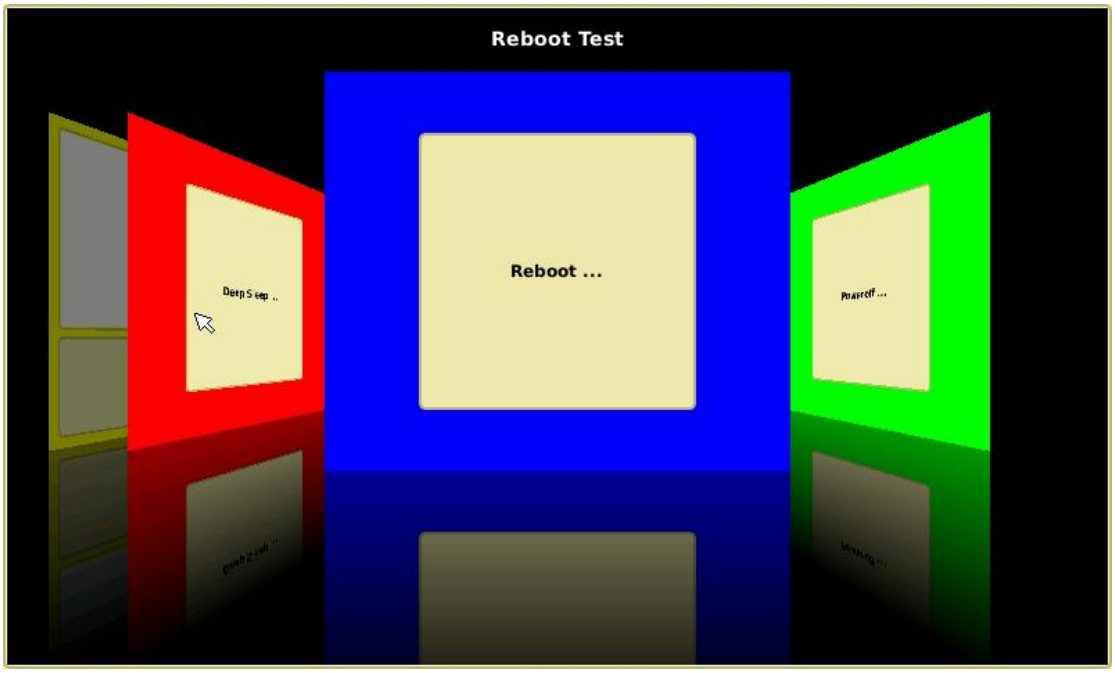
1.28 使用 QT_demo 测试关机

点击 Poweroff 按钮，开发板将会关机。



1.29 使用 QT_demo 测试重启

点击 Reboot 按钮，开发板将重启。



1.30 Qt Creator 的安装

下载地址：

http://qt-project.org/downloads#qt-creator

选择版本：

Qt Creator 2.5.2 for Linux/X11 64-bit (81 MB) **//说明：具体根据自己的机器而定**

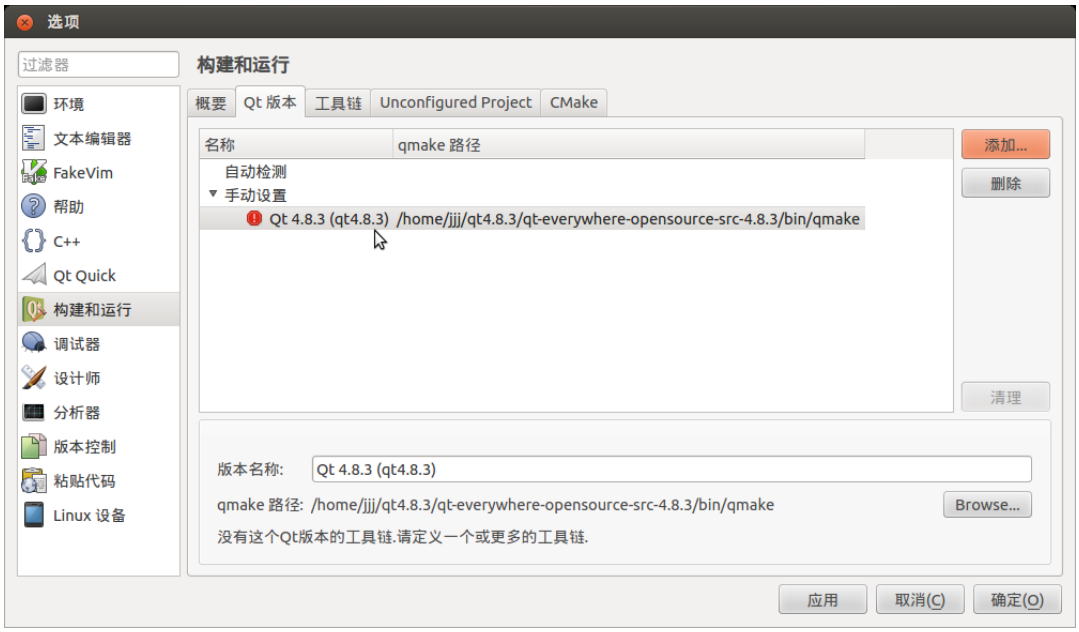
修改文件属性：由于下载下来的文件一般可能默认不可执行，故修改为：

```
chmod 777 qt-creator-linux-x86_64-opensource-2.5.2.bin
```

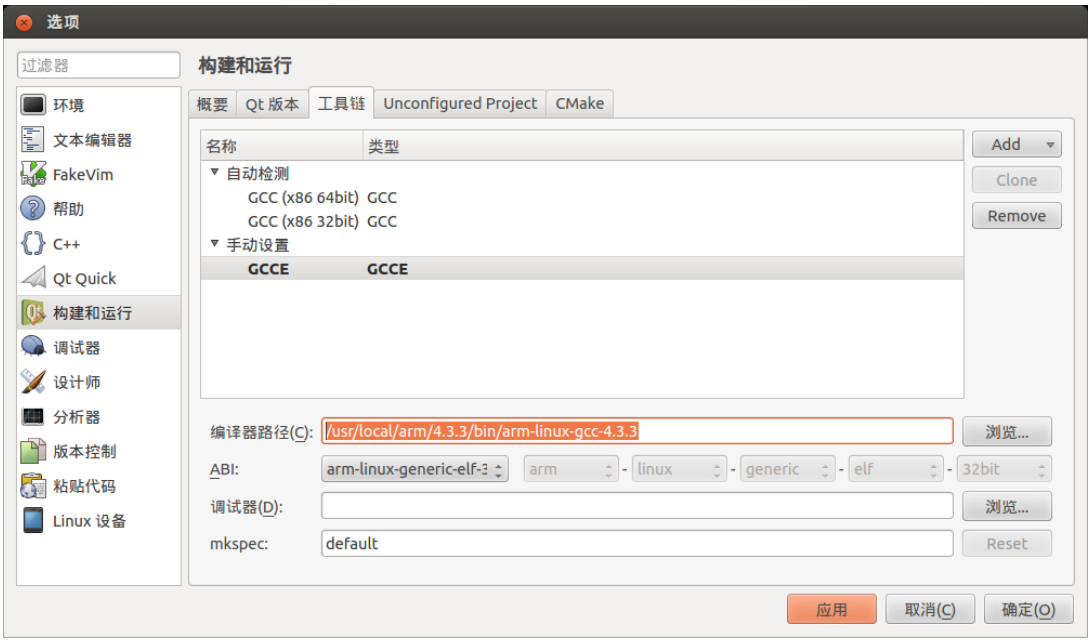
执行./qt-creator-linux-x86_64-opensource-2.5.2.bin,开始安装，会出现图形界面，只需下一步即可。

安装完成后，在桌面上会生成 Qt Creator 的快捷方式，双击运行，配置开始设置 qmake 和交叉编译工具。

点击工具一>选项，出现选项设置界面，选择构建和运行，再选择 Qt 版本，添加我们用来编译应用程序用的 Qt 版本，也就是我们之前移植 Qt4.8.3 编译的 Qt 代码，点击添加，选择目录，如下图：



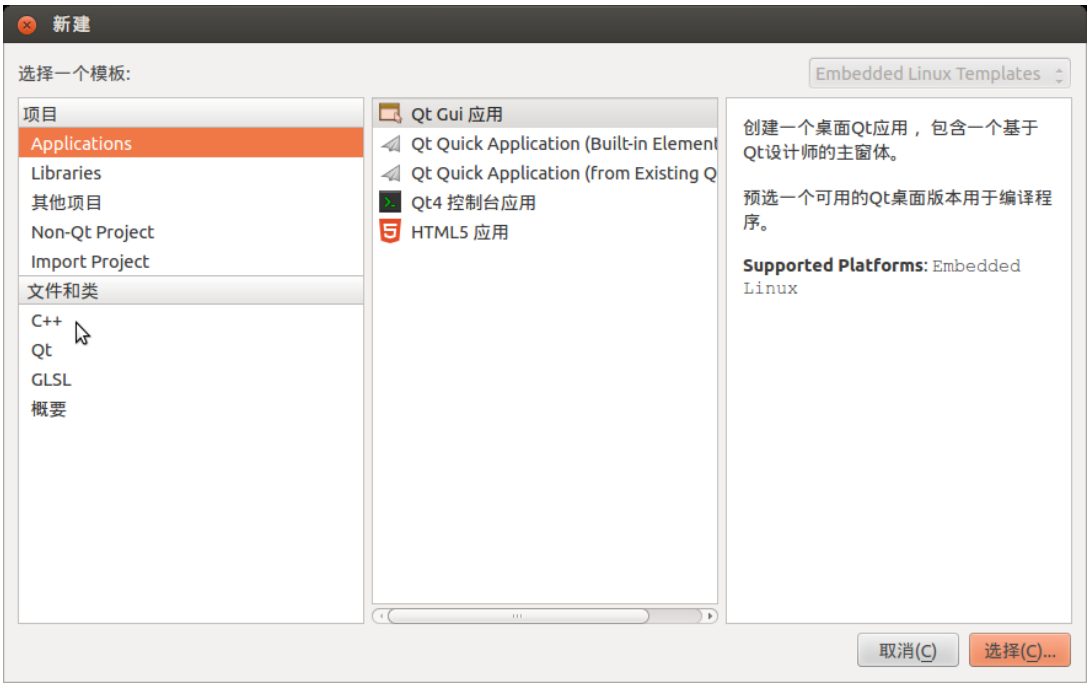
设置好 qmake，再添加工具键，点击工具链，点击添加，选择 GCCE，选择交叉编译工具的路径，如下图：



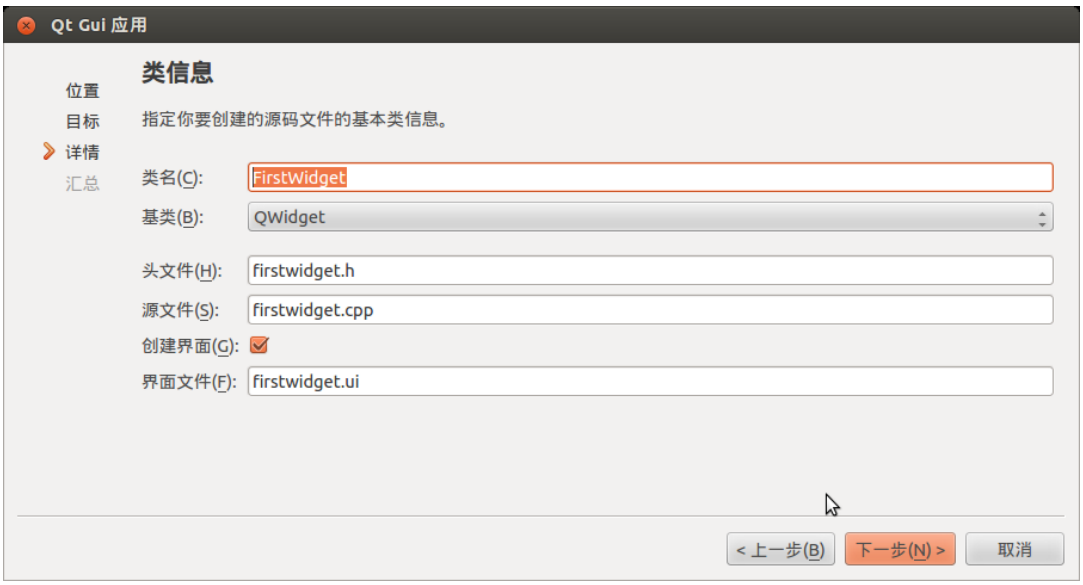
设置好 IDE，就可以开始编写 Qt 应用程序了。

1.31 建立第一个 QT 应用程序

点击 Qt Creator 文件菜单，选择新建文件或工程，依次选择 Application->Qt Gui 应用，选择



设置工程名称及保存目录，比如我们设置为 FirstApp,目录为/home/***/qtcode，并且选中设为默认的工程路径。点击下一步，进入目标设置，再次下一步，进入类信息设置界面，类名可以自定义，基类的选择，根据自己的需要，选择 QWidget,QMainWindow,QDialog,三者的区别这里不多做说明，可以自己查看帮助文档，或者搜索，这里我们选择 QWidget，类名为 FirstWidget，可以选择是否创建 UI 文件，Qt 的 UI 文件，可用于可视化的编辑 UI，也就是拖控件，使用比较方便，也可以不用 UI 文件，用纯代码的方式，编写 UI，看个人习惯。这里我们先用 UI 文件创建，然后再删除 UI 文件用纯代码编写，可以了解两者的区别。



点击下一步，再点完成，到这里，我们的第一个 Qt 应用程序工程就已经建好了，可以根据我们需要，来编辑。

在界面文件中，有一个 firstwidget.ui 文件，就是我们程序默认的显示的界面，双击这个文件，就会启动 Qt 设计师，可以直接在设计师中编辑，修改控件或者布局。在左边的控件列表中选择 Display widgets,拖动一个简单的 QLabel 到窗口上，拖动边框调整大小，在右边的属性窗口中，可以看到这个 QLabel 控件的属性。修改 text 属性，改为 hello qt!

保存文件，关闭设计师，回到代码界面。点击 IDE 上的构建，构建项目“FirstApp”,编译当前工程。完成后，在工程目录下，默认会生成一个新的文件目录，里面有中间文件以及生成的目标程序，当前的目标文件为 qtcode/FirstApp-build-embedded-Qt_4_8_3__qt4_8_3____/FirstApp。

将目标文件复制到 U 盘，拷贝到开发板上。启动开发板，挂载 U 盘，运行程序

```
mount /dev/sda1 /mnt
cd mnt
./FirstApp
```

程序运行，就可以看到我们编写的界面了。背景是另一个程序，我们关注左上角的 FirstWidget 就可以了。

下面我们试着用另一种界面编写方式，不使用 UI 文件，用纯代码的方式来编写界面。在工程中选中 UI 文件，删除，可以连文件一起删除，清理后再编辑工程，会报错，在 firstwidget.h 和 firstwidget.cpp 中删除所有和 ui 相关的代码，即删除所有编译不通过的地方。然后在 firstwidget.cpp 中包含头文件

```
#include <Qlabel>
```

在构造函数中:

```
QLabel * pLbl = new QLabel("hello qt!", this);
```

```
pLbl->setGeometry(100, 50, 100, 30);
```

编译后将生成的新的目标文件放到开发板上运行。效果和使用 UI 文件基本一致。

在 x210ii 上，我们实现了一些基本的功能测试，如 LED，按键，PWM 等，源码在 qtcode.tgz 中，用户可以自行分析。

第2章 x210v3 qtopia 系统移植

由于整个移植过程比较繁琐，为便于初学者学习，我们制作了一些编译脚本，用户只需执行相应的脚本即可编译整个 QT2.2.0 平台以及相关的小程序。由于编译 QT 并不像 android 那样耗用很大的资源，我们可以在虚拟机下编译。以下所有移植过程全部基于 Fedora13 完成，当然用户也可以在 ubuntu 下编译，具体选择什么平台并不重要。值得注意的是，在编译时很可能会弹出一些错误，一般出错都是缺少一些安装库文件或者工具造成，按照错误提示信息安装好相应工具即可。

2.1 安装交叉编译工具

移植 QT4.8 和 QTOPIA 文件系统时，需使用我们指定的交叉编译工具，不保证其他编译工具能够正常使用。

在光盘中找到安装文件 x210-arm-linux-gcc-4.3.3.tgz，拷贝到 linux 系统的 PC 机下并解压：

```
sudo tar xvf x210-arm-linux-gcc-4.3.3.tgz -C /
```

添加环境变量：

```
sudo vim ~/.bash_profile
```

在最末一行添加如下语句：

```
PATH=$PATH:/usr/local/arm/4.3.3/bin/
```

执行如下命令，让新声明的环境变量生效：

```
source ~/.bash_profile
```

在当前命令终端执行如下指令测试交叉编译工具是否安装成功：

```
arm-linux-gcc –v
```

成功打印信息如下：

```
[lqm@lqm share]$ arm-linux-gcc -v
Using built-in specs.
Target: arm-none-linux-gnueabi
Configured with: /scratch/maxim/arm-lite/src-4.3-arm-none-linux-gnueabi-lite/gcc-4.3/configure
--build=i686-pc-linux-gnu --host=i686-pc-linux-gnu --target=arm-none-linux-gnueabi
--enable-threads --disable-libmudflap --disable-libssp --disable-libstdcxx-pch --with-gnu-as
--with-gnu-ld
--with-specs='%{funwind-tables|fno-unwind-tables|mabi=*|ffreestanding|nostdlib:;:-funwind-table
s}' --enable-languages=c,c++ --enable-shared --enable-symvers=gnu --enable-__cxa_atexit
--with-pkgversion='Sourcery G++ Lite 2009q1-176'
--with-bugurl=https://support.codesourcery.com/GNUToolchain/ --disable-nls
--prefix=/opt/codesourcery --with-sysroot=/opt/codesourcery/arm-none-linux-gnueabi/libc
--with-build-sysroot=/scratch/maxim/arm-lite/install-4.3-arm-none-linux-gnueabi-lite/arm-none-li
nux-gnueabi/libc
--with-gmp=/scratch/maxim/arm-lite/obj-4.3-arm-none-linux-gnueabi-lite/host-libs-2009q1-176-ar
m-none-linux-gnueabi-i686-pc-linux-gnu/usr
--with-mpfr=/scratch/maxim/arm-lite/obj-4.3-arm-none-linux-gnueabi-lite/host-libs-2009q1-176-a
```

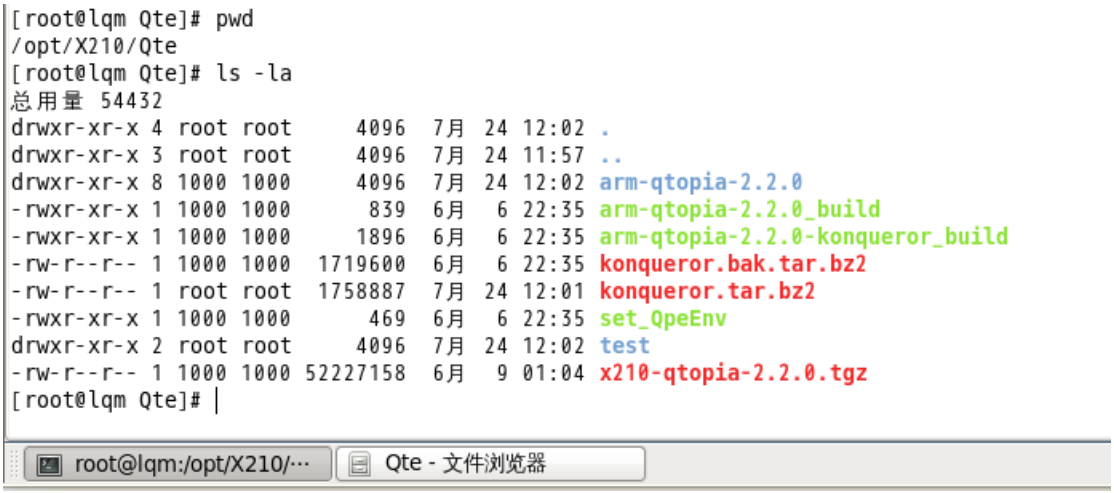
```
rm-none-linux-gnueabi-i686-pc-linux-gnu/usr                    --disable-libgomp
--enable-poison-system-directories
--with-build-time-tools=/scratch/maxim/arm-lite/install-4.3-arm-none-linux-gnueabi-lite/arm-none
-linux-gnueabi/bin
--with-build-time-tools=/scratch/maxim/arm-lite/install-4.3-arm-none-linux-gnueabi-lite/arm-none
-linux-gnueabi/bin
Thread model: posix
gcc version 4.3.3 (Sourcery G++ Lite 2009q1-176)
[lqm@lqm share]$
```

2.2 安装 Qtopia 源码

将光盘中的源码包 x210-Qtopia-sdk.tgz 拷贝到 linux 系统的 PC 机并解压：

```
sudo  tar  xvf  x210-Qtopia-sdk.tgz  -C  /
```

解压完毕，源码会安装在/opt/X210 目录，如下图：



各文件说明如下：

set_QpeEnv：环境变量设置脚本，编译 Qtopia 程序时需用到。

x210-qtopia-2.2.0.tgz：qtopia 源码。

arm-qtopia-2.2.0_build：qtopia 编译脚本

konqueror.tar.bz2 浏览器 konqueror 源代码

arm-qtopia-2.2.0-konqueror_build：编译 konqueror 的脚本

2.3 编译 Qtopia 源码

在命令行终端进入/opt/X210/Qte 目录，执行如下指令编译源码：

```
./arm-qtopia-2.2.0_build
```

注意，如果提示找不到 arm-linux-g++之类的错误，通过如下指令进入超级权限模式：

```
sudo su
```

再执行上面指令编译即可。

这里编译会花很长一段时间，请耐心等待。很可能会遇到一些编译错误，基本上都是一些依赖的库文件或工具没有安装，按照错误提示安装即可。

执行如下指令编译浏览器源码：

```
./arm-qtopia-2.2.0-konqueror_build
```

注意，浏览器源码并非必须编译，如果不需要，可以不用编译。

编译完成后，生成的映像文件会存放到/opt/X210/Qte/arm-qtopia-2.2.0/qtopia/image/opt 目录，如下图所示。

```
[root@lqm image]# cd opt/
[root@lqm opt]# ls
kde  Qtopia
[root@lqm opt]# pwd
/opt/X210/Qte/arm-qtopia-2.2.0/qtopia/image/opt
[root@lqm opt]#
```

2.4 制作 Qtopia 的 bootloader

同 qt4.8 的 uboot。

2.5 制作 Qtopia 的 kernel

同 qt4.8 的 kernel。

2.6 制作 Qtopia 文件系统

为了让读者能够更快的体验 Qtopia 文件系统，我们这里做了一个 demo 系统，供读者参考。

将光盘中的 rootfs-qtopia_130718.tgz 拷贝到 linux 系统的 PC 机上并解压：

```
tar  xvf  rootfs-qtopia_130718.tgz  .
```

会新生成一个名叫 rootfs-qtopia 的目录，将光盘中 X210II_B\linux\qtopia2.2.0\shell 目录的 mkfs 脚本拷贝到当前目录，即和 rootfs-qtopia 目录在同一路径下，

执行如下命令生成基于 inand 平台的文件系统映像：

```
./mkfs  ext3
```

执行如下命令生成基于 nand 平台的文件系统映像：

```
./mkfs  jffs2
```

将会在当前目录生成我们要制作的 roofs 文件系统。

用户可以替换成自己编译出来的 QT 文件系统，只需将/opt/X210/Qte/arm-qtopia-2.2.0/qtopia/image/opt/目录下的 Qtopia 目录和 kde（kde 目录在编译浏览器 konqueror 后才有）拷贝到文件系统的“opt/”目录下后，再重新执行上面的脚本命令打包文件系统即可。

第3章 Qtopia 文件系统的烧写

3.1 在裸板上烧写 bootloader

同 QT4.8 的相关章节。

3.2 烧写内核

同 QT4.8 的相关章节。

3.3 烧写文件系统

启动 uboot，在 3 秒倒计时期间按下空格键，进入 uboot 命令行，输入 fastboot 进入 fastboot 烧录界面。开启 windows 下的 DOS 界面，执行如下指令更新 nand 平台文件系统：

```
fastboot  flash  system  rootfs-qtopia.jffs2
```

执行如下指令更新 inand 平台文件系统：

```
fastboot  flash  system  rootfs-qtopia.ext3
```

3.4 使用电容触摸屏操作 UI 界面

电阻触摸屏和电容触摸屏有一些差异，相对映像，仅体现在文件系统上，与 bootloader 和内核映像无关。也就是说，同一个 bootloader 和内核，同时支持电阻触摸和电容触摸，所不同的是，我们需要在文件系统中来配置相关选项。默认配置情况下，支持电容触摸。即烧写出厂的映像文件后，电容触摸屏直接可用。而且我们已经将电容屏的校正文件打包在文件系统中，开机无须再校屏了。这里我们主要讲解电阻触摸屏如何操作 UI 界面。

3.5 使用电阻触摸屏操作 UI 界面

由于文件系统默认是电容触摸屏的配置，而且/etc 目录下的校屏文件是针对电容屏的，所以烧写默认映像后，电阻屏完全无法操作。

第一步：将/etc 目录下针对电阻触摸屏的脚本 qtopia 复制到/bin 目录：

```
cp  /etc/qtopia  /bin
```

第二步：删除/etc 目录下的校屏文件，重启开发板；

第三步：机器重启后，会自动执行校屏程序，如下图：

依次点击十字架进行校屏。校正完成后，会在/etc 目录下生成最新的校屏文件 pointercal。这时，电阻触摸屏就能正常操作 UI 界面了。

3.6 使用电容触摸屏操作 tslib

在电容触摸屏能够正常操作 UI 的前提下，进入/usr/local/bin 目录，执行如下指令：

```
./ts_test
```

可以看到，有如下错误提示：

```
[root@x210 /]# cd /usr/local/bin/
[root@x210 bin]# ./ts_test
/dev/touchscreen/ucb1x00: No such file or directory
[root@x210 bin]#
```

这是因为没有设置 TSLIB 的相关环境变量的缘故。执行如下指令声明环境变量：

```
cd  /etc
source  setenv_cap
```

再执行 ts_test 命令，即可正常操作了，示例如下：

```
[root@x210 /]# cd /etc/
[root@x210 /etc]# source setenv_cap
set tslib environment...
[root@x210 /etc]# cd /usr/local/bin/
[root@x210 bin]# ./ts_test
```

注意，我们将环境变量做成脚本，是为了同时兼容电阻屏和电容屏，如果您只使用一种触摸屏，可以将脚本内容写死在/etc/profile 文件中，这样机器重新启动后，就没有必要手动运行脚本了。

3.7 使用电阻触摸屏操作 tslib

在电阻触摸屏能够正常操作 UI 的前提下，进入/usr/local/bin 目录，执行如下指令：

```
./ts_test
```

可以看到，有如下错误提示：

```
[root@x210 /]# cd /usr/local/bin/
[root@x210 bin]# ./ts_test
/dev/touchscreen/ucb1x00: No such file or directory
[root@x210 bin]#
```

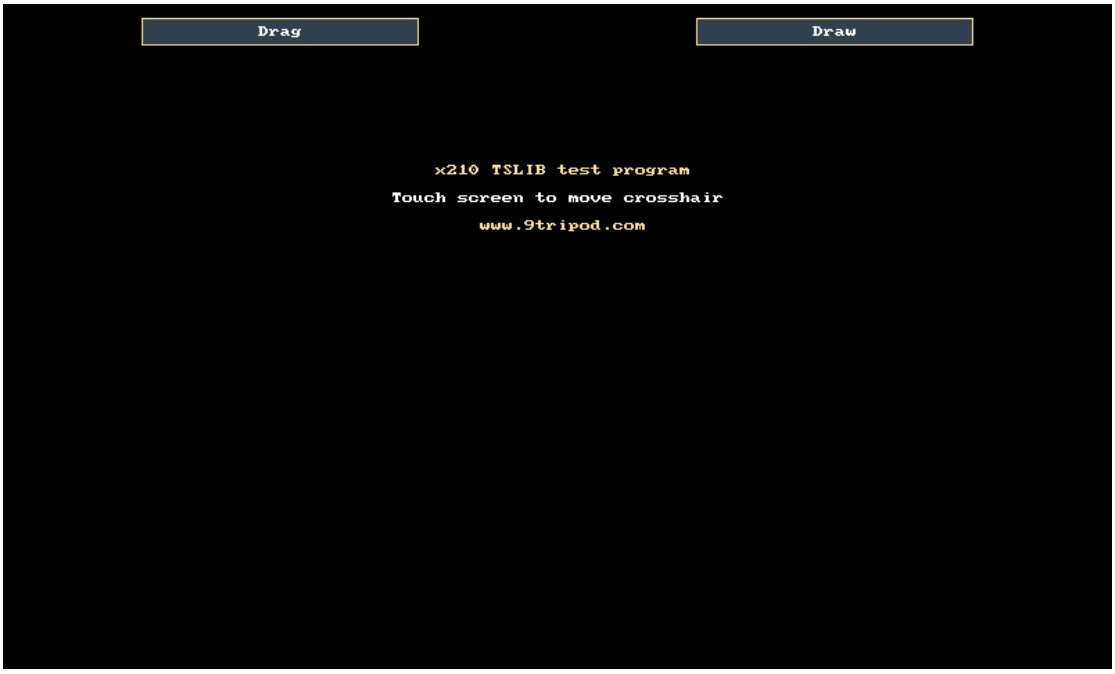
这是因为没有设置 TSLIB 的相关环境变量的缘故。执行如下指令声明环境变量：

```
cd    /etc
source    setenv_res
```

再执行 ts_test 命令，即可正常操作了，示例如下：

```
[root@x210 /root]# cd /etc/
[root@x210 /etc]# source setenv_res
set tslib environment...
[root@x210 /etc]# ./ts_te
-/bin/sh: ./ts_te: not found
[root@x210 /etc]# cd /usr/local/bin/
[root@x210 bin]# ./ts_test
```

测试界面如下：




Drag
Draw
x210 TSLIB test program
Touch screen to move crosshair
www.9tripod.com
12:17

第4章 Linux 开发指南

4.1.1 触摸屏校正

当第一次烧写文件系统后，启动开发板时，会出现触摸屏校正界面，按照提示依次点击出现的十字架即可完成校屏。进入 QT 界面后，也可以通过点击 Settings->Recalibrate 进行校屏。如果出现校屏误操作，使得无法正常点击屏幕，也可以在进入控制台后，手动删除/etc 下的 pointercal 文件，再重启开发板，即可进入校屏界面，如下图。

注意，如果您使用的电容触摸屏，则无需校正触摸屏。

4.1.2 播放 mp3

有两种方法可以播放 mp3，第一种是直接使用 QT 上的 UI 界面。点击子界面 Applications 下的 Music，将会出现播放界面。在歌曲列表中选择要播放的歌曲，点击上面的播放按钮，开始播放歌曲。也可以在 Documents 一栏直接双击歌曲播放。

第二种方法需要制作 mplayer，通过命令行敲命令播放。此种方法虽然麻烦，但是在平时的驱动调试中，会更加的方便。在进行底层驱动程序开发时，很多情况下是没有 UI 界面的。就算有 UI 界面，有时候也会存在没有触摸屏，触摸屏没有调好等因素。因此有必要掌握第二种方法。

将 mplayer 和要播放的音视频文件拷贝到 SD 卡，然后将 SD 卡插到右侧卡槽，使用如下命令挂载 SD 卡：

```
cd  /
mkdir  sdcard
mount  /dev/mmcblk0p1  /sdcard
cd  sdcard
```

使用如下命令播放：

```
./mplayer  *.avi
```

```
./mplayer    *.mp3
```

连接串口后，可以通过 PC 键盘的 0 或 9 调节音量。

4.1.3 在后台运行程序

在上一节中给出了播放音乐的示例，但是这时候 mplayer 已经占据了终端控制台，在音乐播放完之前，我们无法再使用终端控制台了。又比如我们开发一款产品时，就需要在启动文件系统后运行一个应用程序，如果运行了一个程序，终端控制台就被占用了，那将极大的限制我们的功能。为止，我们可以将程序放在后台运行。使用方法很简单，我们只需在执行的指令后面添加一个”&”即可。如播放音乐时使用如下命令：

```
./mplayer    *.mp3    &
```

4.1.4 中止程序的运行

中止程序的运行有多种方式，最直接的方式就是直接按 ctrl+c。如前面我们正在播放一段音频文件，我们可以按 ctrl+c 退出程序。但是如果程序在后台运行，那么我们按 ctrl+c 就不管用了。这时我们可以使用 kill 命令。

```
kill+PID
kill+文件名
```

4.1.5 屏幕抓图

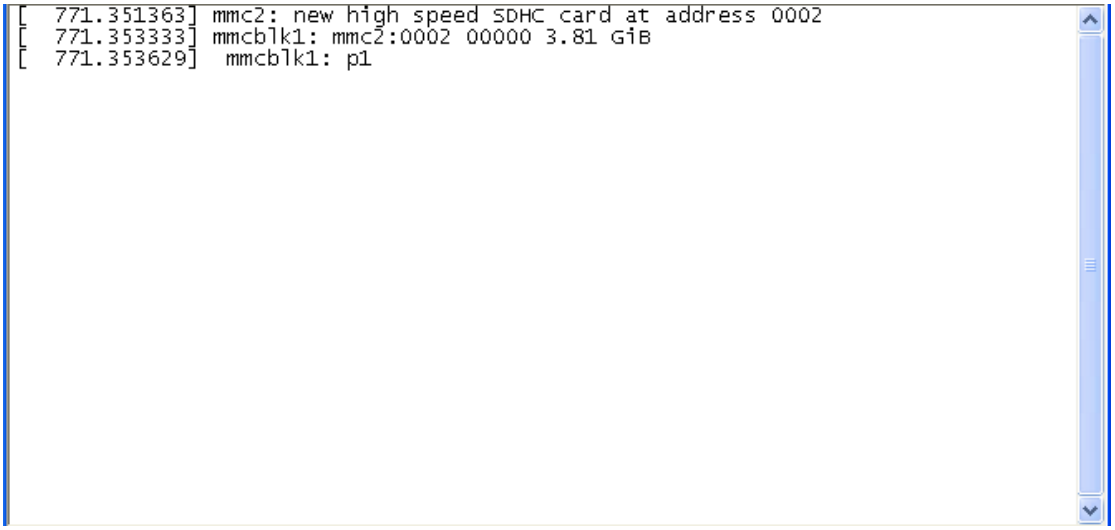
本文档中的各个图片，都是采用 gsnap 这个工具进行抓图的。进入 QT 图形界面后，我们能在 LCD 上看到丰富多彩的人机交互界面。通过 gsnap 可以抓取到图形界面精彩的瞬间。在控制台终端输入如下命令：

```
gsnap    test_pic.jpg    /dev/fb0
```

这时在当前目录将会保存 test_pic.jpg 图像文件。详细的 gsnap 移植步骤在后面会有详细描述。

4.1.6 挂载 SD 卡

进入 QT 图形界面后，在命令终端会有控制台出现，这时可以通过控制台查看文件系统的内容。将 SD 卡插到开发板的右侧卡槽，串口终端会有如下提示：

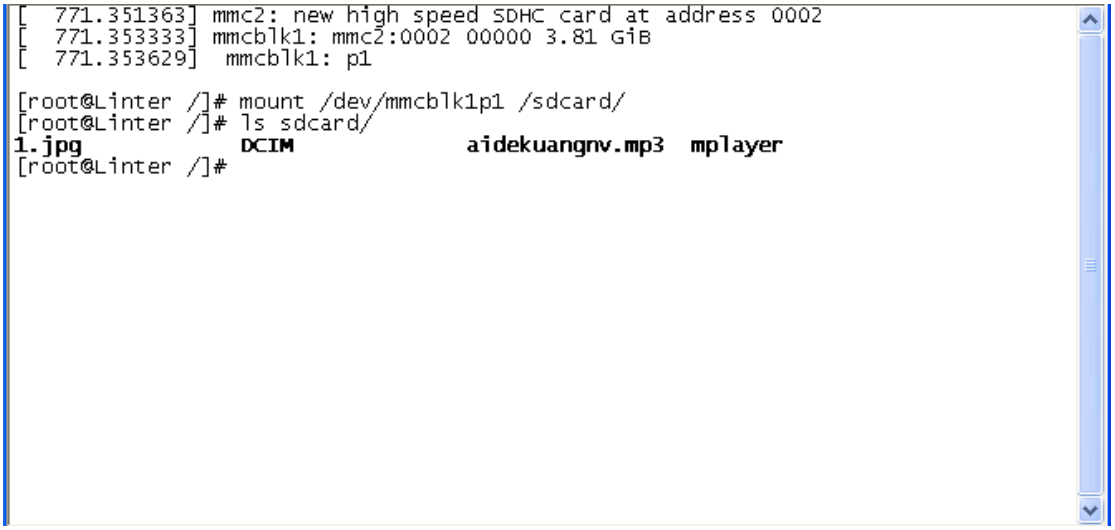


这时在文件系统的/dev 目录将会自动生成一个名叫 mmcblk0p1 的块设备文件。它就是

对应的 SD 卡的设备文件，使用如下命令挂载 SD 卡到/sdcard 目录：

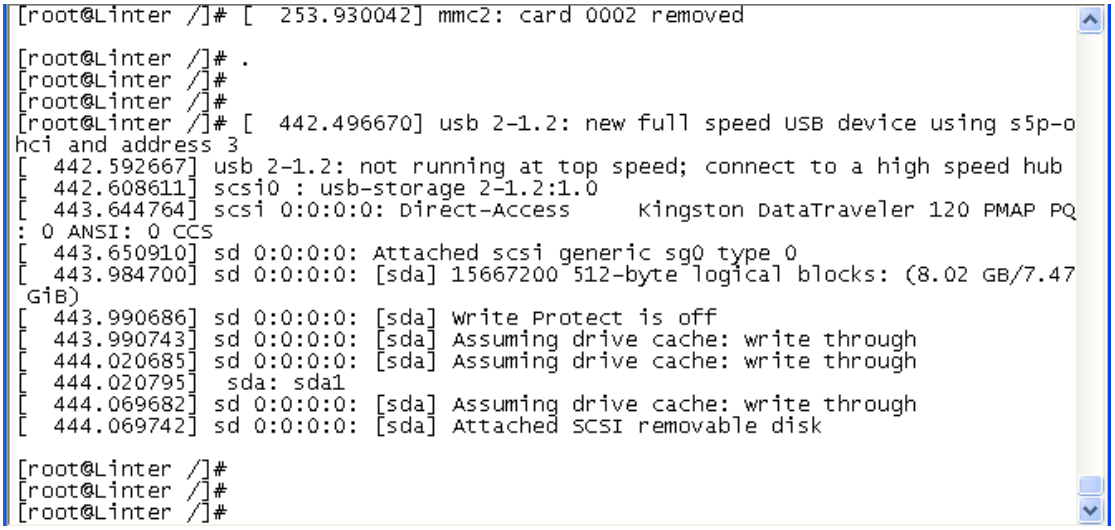
```
mkdir   /sdcard
mount   /dev/mmcblk0p1   /sdcard
```

查看/scard 目录下的内容，即是我们 SD 卡中的内容，如下图所示：



4.1.7 挂载 U 盘

进入 QT 图形界面后，在命令终端会有控制台出现，这时可以通过控制台查看文件系统的内容。插入 U 盘后，串口终端会有如下提示：



这时在文件系统的/dev 目录将会自动生成一个名叫 sda1 的块设备文件。它就是对应的 U 盘设备文件，使用如下命令挂载 U 盘到/udisk 目录：

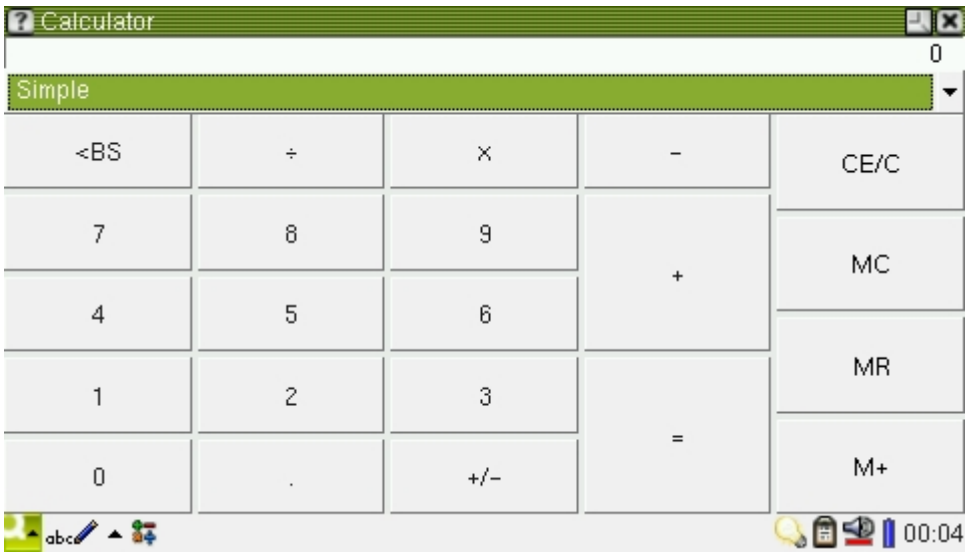
```
mkdir   /udisk
mount   /dev/sda1   /udisk
```

查看/udisk 目录下的内容，即是我们 U 盘中的内容，如下图所示：

```
ptypc               random              ttype               zero
ptypd               rfkill              ttypf
ptype               root                ttyq0
[root@Linter /]#
[root@Linter /]#
[root@Linter /]#
[root@Linter /]# ls
bin      dev      home     lib      mnt      proc     sbin     sys      usr
data     etc      init     linuxrc  opt      root     sdcard   tmp      var
[root@Linter /]# mkdir udisk
[root@Linter /]# mount /dev/sda1 udisk
[root@Linter /]# ls udisk/
200???                          [????????].??__?8?__480P??.xv.flv
2012-07-23-TP-Driver.7z         [????????].??__?9?__480P??.xv.flv
210?PS?                         convertor.bin.tar.gz
???                             es_file_explorer_v1_6_1_3.apk
???? - ????.mp3                 i210
???? - ????.mp3                 pic
AdobephotoshopCS5               pic.zip
PS??                            update.bin
USB_WIFI_NW336                  xboot.tgz
Word                            zImage
[????????].??__?10?__480P??.xv.flv  zImage-android
[root@Linter /]#
```

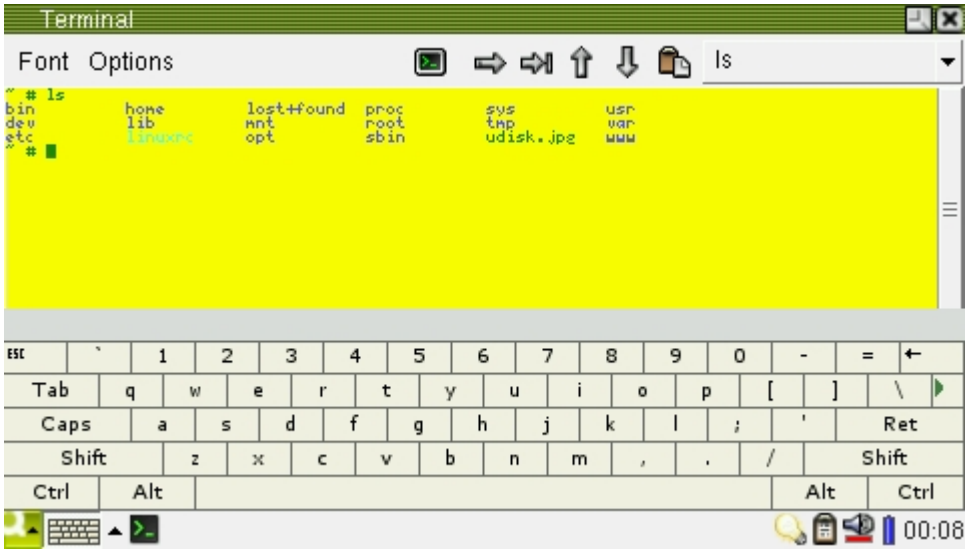
4.1.8 计算器

在 Applications 中单击 Calculator，将会弹出计算器界面，如下图：



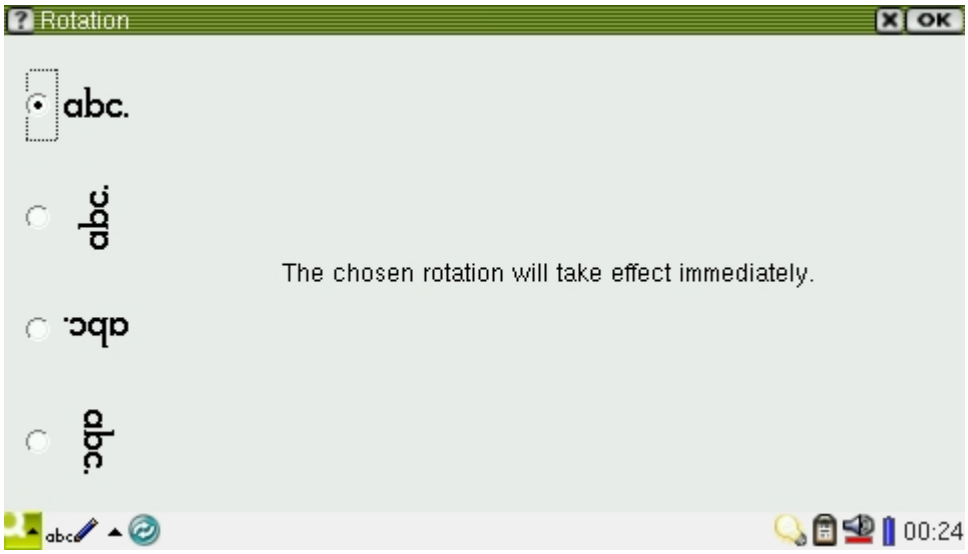
4.1.9 命令终端

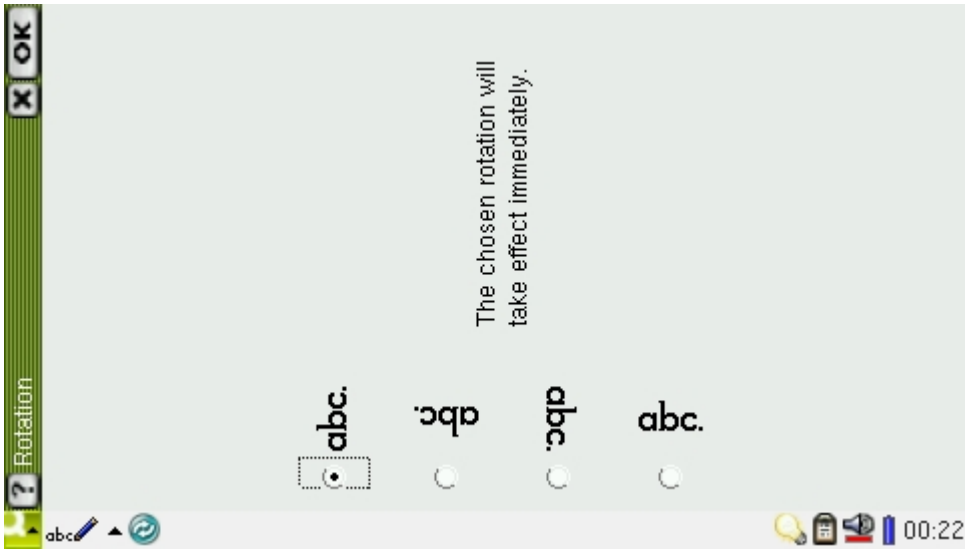
在 Applications 中单击 Terminal，将会在 LCD 上显示终端界面，用户可以通过左下脚的软键盘或是外接 USB 键盘实现终端控制台的操作，所不同的是显示区域从传统的串口终端变到了 LCD 上。这样我们的开发板就成了一套完整的输入输出系统了。用户也可以通过 Options 设置显示的相关属性，如下图：



4.1.10 屏幕旋转

在 Setting 中单击 Rotation，出现屏幕旋转的界面，选择需要的方向，按 OK 返回，即可看到想要的结果。需要注意的是，有时需要重启才能看到想要的结果，这是因为 QT 本身的特性。旋转前和旋转后的效果图如下：





4.1.11 时间设置

因为开发板上存放了一块后备电池，因此设置时间后，掉电能保存，具体通过 RTC 驱动程序实现。设置时间步骤如下：

点击右下脚的时间显示区域，将会弹出一个菜单，点击”Set time…”，打开时间设置界面，即可设置时区，时间日期等，如下图:

4.1.12 通过串口与 PC 交互数据[待续]

4.1.13 保存系统时钟

Linux 可以使用 date 指令更改时间日期。例如：

date -s 201104162350 #设置为 2011 年 4 月 16 日 23：50 分

hwclock -w #把刚设置的时间存入 RTC 寄存器

hwclock -s #恢复 linux 系统时钟为 RTC 寄存器值，一般将该指令放在 rcS 中开机自动执行。

4.1.14 掉电保存数据到 flash

由于本系统采用了 yaffs2 文件系统，因此可以很方便的保存数据，确保掉电后数据不丢失。如我们从 U 盘中拷备一首歌曲到/home/lqm 目录：

```
cp /mnt/muyangqu.mp3 /home/lqm
```

重启开发板，我们发现在/home/lqm 目录仍然存在刚才拷备的这首歌曲，说明掉电后数据并没有丢失。

4.1.15 设置开机自动运行程序

借助启动脚本可以设置各种程序开机后自动运行，这点很类似于 WINDOWS 的 Autobat 自动批处理文件。启动脚本位于/etc/init.d/rcS 中，我们可以将自己想要开机运行的程序或是开机执行的指令放在 rcS 里面。比如我们想制作一个简单的开机音乐，我们就完全可以在 rcS 中添加如下语句：

```
./mplayer  start.mp3  &
```

这时，开机后就会播放名叫 start.mp3 的音乐了。注意 start.mp3 需要在当前执行指令所在目录。

4.1.16 查看开发板内存信息

x210 开发板默认配置 512M DDR2 SDRAM，在 uboot 启动时，打印信息上会给出 RAM 大小信息：

```
U-Boot 1.3.4 (May 27 2012 - 23:06:48) for SMDKV210


CPU:  S5PV210@1000MHz(OK)
        APLL = 1000MHz, HclkMsys = 200MHz, PclkMsys = 100MHz
        MPLL = 667MHz, EPLL = 96MHz
                       HclkDsys = 166MHz, PclkDsys = 83MHz
                       HclkPsys = 133MHz, PclkPsys = 66MHz
                       SCLKA2M  = 200MHz
Serial = CLKUART
Board:   X210
DRAM:    512 MB
after init sequence,gd->bd->bi_boot_params:30000100
Flash:   8 MB
SD/MMC: Card init fail!
0 MB
NAND:    512 MB
*** Warning - using default environment

In:      serial
Out:     serial
Err:     serial
checking mode for fastboot ...
Hit any key to stop autoboot:  0
x210 #
```

在进入文件系统后，可以通过 cat 命令查询 Linux 系统分配到的 SDRAM 大小。执行如下命令：

```
cat  /proc/meminfo
```

```
[root@Linter /]# cat /proc/meminfo
MemTotal:         330112 kB
MemFree:          301148 kB
Buffers:               0 kB
Cached:            14384 kB
SwapCached:            0 kB
Active:             5056 kB
Inactive:          12600 kB
Active(anon):       3272 kB
Inactive(anon):      488 kB
Active(file):       1784 kB
Inactive(file):    12112 kB
Unevictable:           0 kB
Mlocked:               0 kB
SwapTotal:             0 kB
SwapFree:              0 kB
Dirty:                 0 kB
Writeback:             0 kB
AnonPages:          3300 kB
Mapped:             7712 kB
Shmem:               488 kB
Slab:               8180 kB
SReclaimable:       4304 kB
SUnreclaim:         3876 kB
KernelStack:         336 kB
PageTables:          164 kB
NFS_Unstable:          0 kB
Bounce:                0 kB
WritebackTmp:          0 kB
CommitLimit:      165056 kB
Committed_AS:      10344 kB
VmallocTotal:     450560 kB
VmallocUsed:       22484 kB
VmallocChunk:     410620 kB
[root@Linter /]#
```

我们看到内存总量只有 70M 左右，这是因为各种多媒体处理，如 MFC，JPEG 等驱动分配了一些内存，我们可以通过 dmesg 命令查询开机信息，上会给出了具体的内存使用情况：

```
[    0.000000] sclk_mdnie_pwm: source is ext_xtal (0), rate is 24000000
[    0.000000] s5pv210: 37748736 bytes system memory reserved for mfc at 0x30b97
000
[    0.000000] s5pv210: 37748736 bytes system memory reserved for mfc at 0x40000
000
[    0.000000] s5pv210: 25165824 bytes system memory reserved for fimc0 at 0x424
00000
[    0.000000] s5pv210: 10137600 bytes system memory reserved for fimc1 at 0x43c
00000
[    0.000000] s5pv210: 25165824 bytes system memory reserved for fimc2 at 0x445
ab000
[    0.000000] s5pv210: 25165824 bytes system memory reserved for jpeg at 0x45da
b000
[    0.000000] s5pv210: 3072000 bytes system memory reserved for fimd at 0x475ab
000
[    0.000000] s5pv210: 10485760 bytes system memory reserved for texstream at 0
x47899000
[    0.000000] s5pv210: 3379200 bytes system memory reserved for pmem_gpu1 at 0x
48299000
[    0.000000] s5pv210: 8388608 bytes system memory reserved for g2d at 0x485d20
00
[    0.000000] Built 1 zonelists in Zone order, mobility grouping on.  Total pag
es: 130048
[    0.000000] Kernel command line: console=ttySAC0,115200 root=/dev/mtdblock4 r
```

第5章 嵌入式 Linux 开发环境的搭建

5.1.1 x210 分区表

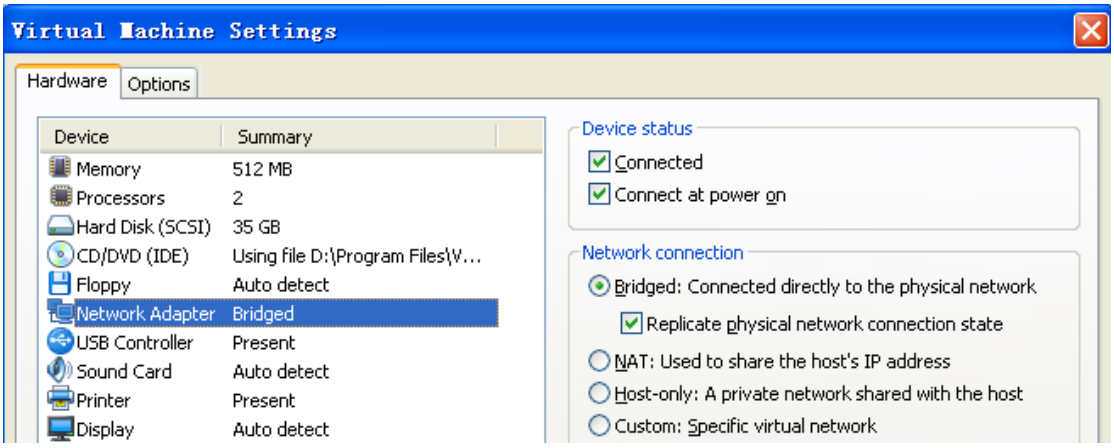
x210 nand 平台分区表如下：

映像	起始地址	文件尺寸
uboot	0x00000000	0x00080000
kernel	0x00600000	0x00500000
rootfs	0x00b00000	0x03000000

5.1.2 使用 TFTP 烧写 uboot

注意，使用 TFTP 烧写映像时，务必先将 TFTP 环境搭建好。

在虚拟机的 VM->Setting 中可以看到设置界面如下:

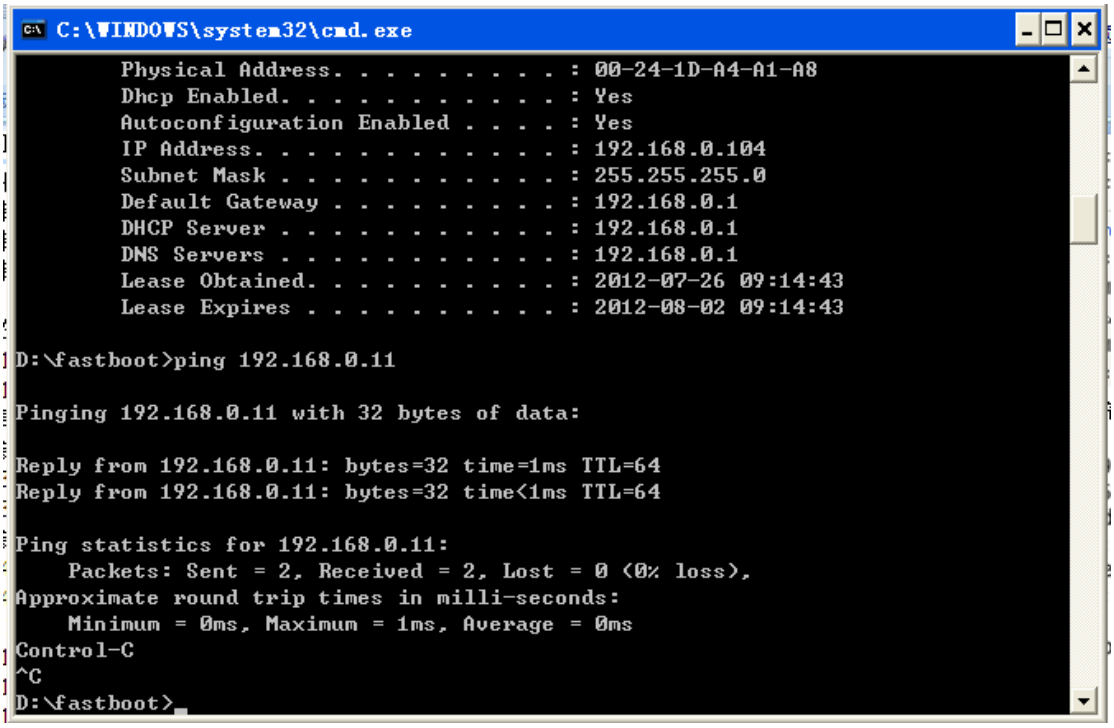


在 Fedora 的终端输入 ifconfig 指令,查看 Fedora 系统的 IP 地址,如下图:

```
[lqm@lqm share]$ ifconfig
eth1      Link encap:Ethernet  HWaddr 00:0C:29:7C:6D:5D
          inet addr:192.168.0.11  Bcast:192.168.255.255  Mask:255.255.0.0
          inet6 addr: fe80::20c:29ff:fe7c:6d5d/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1760845 errors:169 dropped:148 overruns:0 frame:0
          TX packets:1945407 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:586362024 (559.1 MiB)  TX bytes:888277347 (847.1 MiB)
          Interrupt:19 Base address:0x2000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:1232 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1232 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:127086 (124.1 KiB)  TX bytes:127086 (124.1 KiB)
```

在 windows 桌面的命令行 DOS 界面(开始->运行->cmd, 然后回车),输入 ipconfig /all 查询桌面系统的 IP 地址,如下图:



可以看到,桌面系统的 IP 地址为 192.168.0.104,Fedora 系统的 IP 地址为 192.168.0.11.在桌面系统的 DOS 界面 PING Fedora 系统,上图已经显示，可以 PING 通。

在 Fedora 中 PING 桌面系统,如下图:

```
[lqm@lqm share]$ ping 192.168.0.104
PING 192.168.0.104 (192.168.0.104) 56(84) bytes of data.
64 bytes from 192.168.0.104: icmp_seq=1 ttl=64 time=2.42 ms
64 bytes from 192.168.0.104: icmp_seq=1 ttl=64 time=2.49 ms (DUP!)
64 bytes from 192.168.0.104: icmp_seq=2 ttl=64 time=0.659 ms
64 bytes from 192.168.0.104: icmp_seq=2 ttl=64 time=0.885 ms (DUP!)
64 bytes from 192.168.0.104: icmp_seq=3 ttl=64 time=1.12 ms
64 bytes from 192.168.0.104: icmp_seq=3 ttl=64 time=1.16 ms (DUP!)
^C
--- 192.168.0.104 ping statistics ---
3 packets transmitted, 3 received, +3 duplicates, 0% packet loss, time 2253ms
rtt min/avg/max/mdev = 0.659/1.460/2.496/0.728 ms
[lqm@lqm share]$
```

这时,桌面系统和 Fedora 系统相互能够 PING 通,说明网络设置正常.

在 Fedora 的命令终端使用 route 命令查询子网掩码,如下图:

```
[lqm@lqm share]$ route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.0.0     *               255.255.0.0     U     1      0        0 eth1
default         192.168.1.1     0.0.0.0         UG    0      0        0 eth1
[lqm@lqm share]$
```

可以发现子网掩码为 255.255.0.0,如果不是请设置为 255.255.0.0.

启动开发板的 uboot,进入配置菜单,按 q 退出交互菜单,进入命令行控制台,输入 printenv 查看 uboot 设置参数,如下图:

```
DRAM:    512 MB
after init sequence,gd->bd->bi_boot_params:30000100
Flash:   8 MB
SD/MMC: Card init fail!
0 MB
NAND:    512 MB
In:      serial
Out:     serial
Err:     serial
checking mode for fastboot ...
Hit any key to stop autoboot:  0
x210 # pri
bootargs=console=ttySAC0,115200 root=/dev/mtdblock4 rw init=/linuxrc rootfstype=
jffs2
bootcmd=nand read C0008000 600000 500000; bootm C0008000
mtdpart=80000 400000 3000000
bootdelay=3
baudrate=115200
ethaddr=00:40:5c:26:0a:5b
ipaddr=192.168.0.103
gatewayip=192.168.0.1
netmask=255.255.0.0
serverip=192.168.0.11

Environment size: 311/16380 bytes
x210 #
```

上面 serverip 为服务器 IP,由于我们需要从 Fedora 系统下载映像文件,因此应该将它设置为 Fedora 的 IP 地址.ipaddr 为开发板的 IP 地址,前面我们设置子网掩码为 255.255.0.0,因此这里为保证和服务器 IP 在同一网段,必须设置为 192.168.*.*.设置完成后,在开发板的 uboot 命令控制台 PING 服务器 IP,如下图:

```
In:      serial
Out:     serial
Err:     serial
checking mode for fastboot ...
Hit any key to stop autoboot:  0
x210 # pri
bootargs=console=ttySAC0,115200 root=/dev/mtdblock4 rw init=/linuxrc rootfstype=
jffs2
bootcmd=nand read C0008000 600000 500000; bootm C0008000
mtdpart=80000 400000 3000000
bootdelay=3
baudrate=115200
ethaddr=00:40:5c:26:0a:5b
ipaddr=192.168.0.103
gatewayip=192.168.0.1
netmask=255.255.0.0
serverip=192.168.0.11

Environment size: 311/16380 bytes
x210 # ping 192.168.0.11
dm9000 i/o: 0x88000300, id: 0x90000a46
DM9000: running in 16 bit mode
MAC: 00:40:5c:26:0a:5b
operating at 100Mbps FULL duplex mode
host 192.168.0.11 is alive
x210 #
```

当提示 host *.*.*.* ls alive 时,表示开发板已经和服务器 PING 通了,这时就可以从服务器的 tftp 工作目录下载映像文件了。

注意：以上设置，已搭建好 TFTP 以及网络调试环境，后续章节不再重复。

为了便于调试，我们可以修改编译脚本，使得每次编译 uboot 时都会将生成的映像文件拷贝到 tftp 工作目录，例如编译从 nand 启动的 u+boot 时，修改脚本如下：

```
#!/bin/sh
# create by liuqiming
# date: 2011-11-24

MODE=$1
```

```
CPU_NUM=$(cat /proc/cpuinfo |grep processor|wc -l)
CPU_NUM=$((CPU_NUM+1))

# 如果没有传入参数，则提示错误
if [ -z $MODE ]; then
        echo "\nERROR: Must set the compile mode"
        echo "Example: $0 nand or $0 sd\n"
        exit
fi

# 如果传入参数大于 1，则提示错误
if [ $# -gt 1 ]; then
        echo -e "\033[40;32m"
        echo "Usage: ./mk [mode]"
        echo "        mode:       nand, sd"
        echo "compile: ./mk nand or ./mk sd"
        echo -e "\033[40;37m"
fi

if [ $MODE = "nand" ]; then
        make distclean
        make x210_nand_config
        make -j${CPU_NUM}
        mv u-boot.bin uboot_nand.bin
        if [ -f uboot_nand.bin ]; then
                #cp uboot_nand.bin ../out/release/uboot.bin
                cp uboot_nand.bin /home/lqm/tftpboot/uboot.bin
                echo "^_^ uboot_nand.bin is finished successful!"
                exit
        else
                echo "make error,cann't compile u-boot.bin!"
                exit
        fi
elif [ $MODE = "sd" ]; then
        make distclean
        make x210_sd_config
        make -j${CPU_NUM}
        mv u-boot.bin uboot_sd.bin
        if [ -f uboot_sd.bin ]; then
                #cp uboot_sd.bin ../out/release/uboot.bin
                cp uboot_sd.bin /home/lqm/tftpboot/uboot.bin
                echo "^_^ uboot_sd.bin is finished successful!"
                exit
```

```
    else
        echo "make error,cann't compile u-boot.bin!"
        exit
    fi
else
    echo -e "\033[40;32m"
    echo "Usage: ./mk [mode]"
    echo "       mode:    nand, sd"
    echo "compile: ./mk nand or ./mk sd"
    echo -e "\033[40;37m"
    exit
fi
```

这样在编译完后，将会把生成的映像文件 uboot_nand.bin 复制到 tftp 工作目录 /home/lqm/tftpboot，同时命名为 uboot.bin。在 uboot 的控制台界面通过下面的指令下载映像到 RAM：

```
tftp   c0008000   uboot.bin
```

正常情况下会有如下提示：

```
baudrate=115200
ethaddr=00:40:5c:26:0a:5b
ipaddr=192.168.0.103
gatewayip=192.168.0.1
netmask=255.255.0.0
serverip=192.168.0.11

Environment size: 311/16380 bytes
x210 # ping 192.168.0.11
dm9000 i/o: 0x88000300, id: 0x90000a46
DM9000: running in 16 bit mode
MAC: 00:40:5c:26:0a:5b
operating at 100Mbps FULL duplex mode
host 192.168.0.11 is alive
x210 # tftp c0008000 uboot.bin
dm9000 i/o: 0x88000300, id: 0x90000a46
DM9000: running in 16 bit mode
MAC: 00:40:5c:26:0a:5b
operating at 100Mbps FULL duplex mode
TFTP from server 192.168.0.11; our IP address is 192.168.0.103
Filename 'uboot.bin'.
Load address: 0xc0008000
Loading: ##########################
done
Bytes transferred = 327680 (0x50000)
x210 #
```

再使用下面命令擦除 uboot 区域，同时烧写 uboot.bin 到 nand flash：

```
nand   erase   0   80000
nand   write   c0008000   0   80000
```

如下图所示：

```
DM9000: running in 16 bit mode
MAC: 00:40:5c:26:0a:5b
operating at 100Mbps FULL duplex mode
host 192.168.0.11 is alive
x210 # tftp c0008000 uboot.bin
dm9000 i/o: 0x88000300, id: 0x90000a46
DM9000: running in 16 bit mode
MAC: 00:40:5c:26:0a:5b
operating at 100Mbps FULL duplex mode
TFTP from server 192.168.0.11; our IP address is 192.168.0.103
Filename 'uboot.bin'.
Load address: 0xc0008000
Loading: #######################
done
Bytes transferred = 327680 (0x50000)
x210 # nand erase 0 80000

NAND erase: device 0 offset 0x0, size 0x80000
Erasing at 0x60000 -- 100% complete.
OK
x210 # nand write c0008000 0 80000

NAND write: device 0 offset 0x0, size 0x80000
Main area write (4 blocks):
 524288 bytes written: OK
x210 #
```

这时，整个 tftp 烧写 uboot 的过程全部完成。以上步骤看起来比较繁琐，但是只要操作熟练，用起来就非常方便了。

5.1.3 使用 TFTP 烧写 kernel

第一步：进入 Linux 内核目录，修改编译脚本：

```
cd   kernel
vim   mk
```

修改后内容如下：

```
#!/bin/sh
#
# Description       : Build Android Script.
# Authors       : http://www.9tripod.com
# Version       : 0.01
# Notes             : None
#
QT_KERNEL_CONFIG=x210_jffs_defconfig

CPU_NUM=$(cat /proc/cpuinfo |grep processor|wc -l)
CPU_NUM=$((CPU_NUM+1))

# make distclean
make ${QT_KERNEL_CONFIG} || return 1
make -j${CPU_NUM} || return 1

cp arch/arm/boot/zImage ~lqm/tftpboot

echo "" >&2
echo "^_^ qt kernel path: arch/arm/boot/zImage" >&2
```

更改 mk 的可执行权限：

```
chmod   777   mk
```

执行脚本编译内核，编译完后脚本将会自动将生成的内核映像文件 zImage 拷贝到 tftp 工作目录：

```
    ./mk
```

第二步：开机进入 bootloader，进入 uboot 控制台，使用 TFTP 下载内核到内存：

```
tftp   c0008000   zImage
```

第三步：擦除 nandflash 中内核所在区域：

```
nand   erase   600000   500000
```

第四步：将下载到内存中的内核映像文件写到 nandflash:

```
nand   write   c0008000   600000   500000
```

这时已更新内核映像，重启开发板查看效果。

5.1.4 使用 TFTP 烧写文件系统

与烧写 kernel 类似，首先通过 tftp 将文件系统映像下载到内存，然后通过下面的指令烧写映像到 nandflash:

```
nand   erase   b00000   3000000
nand   write   c0008000   b00000   3000000
```

由于开发板出厂提供的文件系统包含了 QT 界面，比较大，因此不建议使用 tftp 下载，用户在项目开发过程中，可以去掉 QT 所占有的文件，文件系统长度将会大幅降低，这时使用上面的方式就比较方便了。

5.1.5 使用 tftp 启动 kernel

在 uboot 的交互界面按空格进入控制台，执行下面的命令：

```
setenv   bootcmd   'tftp c0008000 zImage;bootm c0008000'
save
```

重新启动，可以看到以下启动信息：

```
Out:      serial
Err:      serial
checking mode for fastboot ...
Hit any key to stop autoboot:  0
dm9000 i/o: 0x88000300, id: 0x90000a46
DM9000: running in 16 bit mode
MAC: 00:40:5c:26:0a:5b
operating at 100Mbps FULL duplex mode
TFTP from server 192.168.0.11; our IP address is 192.168.0.103
Filename 'zImage'.
Load address: 0xc0008000
Loading: ##################################################################
         ##################################################################
         ##################################################################
         ################################
done
Bytes transferred = 3375344 (0x3380f0)
Boot with zImage
In bootm linux!!!
cmdline:console=ttySAC0,115200 root=/dev/mtdblock4 rw init=/linuxrc rootfstype=j
ffs2

Starting kernel ...

Uncompressing Linux... done, booting the kernel.
[    0.000000] Initializing cgroup subsys cpu
```

上图表明在 3 秒倒计时后，程序自动通过 TFTP 将 zImage 下载到 0xc0008000 的地址，最后跳到该地址执行内核程序。这时我们并没有将内核下载到 nand flash 中，而是直接将电

脑上生成的 zImage 映像下载到内存运行了，这也是我们最常用，也最方便的一种调试内核的手段之一。

5.1.6 使用 nand 启动 kernel

在 uboot 的交互界面按 q 进入控制台，执行下面的命令：

```
setenv    bootcmd    'nand read c0008000 600000 500000;bootm c0008000'
save
```

重新启动，可以看到以下启动信息：

```
SD/MMC: Card init fail!
0 MB
NAND:    512 MB
*** Warning - using default environment

In:      serial
Out:     serial
Err:     serial
checking mode for fastboot ...
Hit any key to stop autoboot:  0

NAND read: device 0 offset 0x600000, size 0x500000
Main area read (40 blocks):
 5242880 bytes read: OK
Boot with zImage
In bootm linux!!!
cmdline:console=ttySAC0,115200 root=/dev/mtdblock4 rw init=/linuxrc rootfstype=j
ffs2

Starting kernel ...

Uncompressing Linux... done, booting the kernel.
[    0.000000] Initializing cgroup subsys cpu
[    0.000000] Linux version 2.6.35.7 (jjj@ubuntu-server) (gcc version 4.5.1 (So
urcery G++ Lite 2010.09-50) ) #1 PREEMPT Tue Jul 24 17:04:49 CST 2012
[    0.000000] CPU: ARMv7 Processor [412fc082] revision 2 (ARMv7), cr=10c53c7f
```

上图表示在 3 秒倒计时后，程序自动从 nand 的 0x80000 开始的地址读取 5M 的数据到 0xc0008000,然后 PC 指针跳转到 0xc0008000 开始执行内核程序。

第6章 嵌入式 Linux 应用程序移植示例

本手册给出的所有应用程序全部在九鼎创展 x210/x210ii/i210 开发板上运行，这里仅给出了一些比较基础，常用的应用程序，旨在为用户打开 Linux 世界奇妙的大门，用户定能举一反三，编写出属于自己的更加丰富完美的程序。

以下所有程序均使用 arm-none-linux-gnueabi V4.3.3 版本交叉编译工具，如果使用其他版本编译器将无法在我们的开发板上运行。检查交叉编译环境命令如下：

```
arm-none-linux-gnueabi-gcc –v
```

```
[lqm@lqm qtopia2.2.0]$ arm-none-linux-gnueabi-gcc -v
Using built-in specs.
Target: arm-none-linux-gnueabi
Configured with: /scratch/maxim/arm-lite/src-4.3-arm-none-linux-gnueabi-lite/gcc-4.3/configure --build=i686-pc-linux-gnu --host=i686-pc-linux-gnu --target=arm-none-linux-gnueabi --enable-threads --disable-libmudflap --disable-libssp --disable-libstdcxx-pch --with-gnu-as --with-gnu-ld --with-specs='%{funwind-tables|fno-unwind-tables|mabi=*|ffreestanding|nostdlib:;:-funwind-tables}' --enable-languages=c,c++ --enable-shared --enable-symvers=gnu --enable-__cxa_atexit --with-pkgversion='Sourcery G++ Lite 2009q1-176' --with-bugurl=https://support.codesourcery.com/GNUToolchain/ --disable-nls --prefix=/opt/codesourcery --with-sysroot=/opt/codesourcery/arm-none-linux-gnueabi/libc --with-build-sysroot=/scratch/maxim/arm-lite/install-4.3-arm-none-linux-gnueabi-lite/arm-none-linux-gnueabi/libc --with-gmp=/scratch/maxim/arm-lite/obj-4.3-arm-none-linux-gnueabi-lite/host-libs-2009q1-176-arm-none-linux-gnueabi-i686-pc-linux-gnu/usr --with-mpfr=/scratch/maxim/arm-lite/obj-4.3-arm-none-linux-gnueabi-lite/host-libs-2009q1-176-arm-none-linux-gnueabi-i686-pc-linux-gnu/usr --disable-libgomp --enable-poison-system-directories --with-build-time-tools=/scratch/maxim/arm-lite/install-4.3-arm-none-linux-gnueabi-lite/arm-none-linux-gnueabi/bin --with-build-time-tools=/scratch/maxim/arm-lite/install-4.3-arm-none-linux-gnueabi-lite/arm-none-linux-gnueabi/bin
Thread model: posix
gcc version 4.3.3 (Sourcery G++ Lite 2009q1-176)
[lqm@lqm qtopia2.2.0]$
```

声明：以下所有应用程序全部为九鼎创展科技有限公司原创作品，所有内容全经我们严格测试，建议用户按照下面步骤动手编译一遍，以增强自己的理解，不推荐直接使用我们提供好的文件。另外，敬请商业人士勿侵犯版权。

6.1.1 Hello World

第一步：生成可执行文件

在 Fedora 或 ubuntu 系统的/home/lqm/share/x210 目录建立下面的目录：

```
cd  ~/share
mkdir -p x210/app-ex/hello
```

新建 hello.c 文件并编辑如下内容：

```
vim  hello.c
```

```c
#include <stdio.h>

int main(void)
{
        printf("hello,embeded world!\n");
}
```

这是一个最基础的应用程序，我们直接敲入命令进行编译：

```
arm-none-linux-gnueabi-gcc -o hello hello.c
```

编译完成后，在当前目录会生成 hello 可执行文件，我们可以使用 file 命令查询执行文件是否为 ARM 体系文件：

```
[root@lqm hello]# file hello
hello: ELF 32-bit LSB executable, ARM, version 1 (SYSV), dynamically linked (uses shared libs), for GNU/Linux 2.6.16, not stripped
[root@lqm hello]#
```

第二步：将可执行文件下载到开发板运行

比较常用的方式有以下四种：

1） 通过串口和 sz/rz 工具

2） 复制到存储媒介，如 SD 卡，U 盘等

3） 通过 NFS 挂载文件系统，这时不用将可执行文件拷备到开发板了，推荐调试使用这种方式！

4） 通过 ftp 传输

这里介绍第二种方法，以 U 盘为例，其他方法请读者自行尝试。

将生成的 hello 文件拷备到 U 盘，再将 U 盘插入开发板的 USB 口，将 U 盘 mount 到 udisk 目录：

```
cd  /
mkdir  udisk  [如果存在 udisk 目录，则不用执行该指令]
mount  /dev/sda1  /udisk
```

进入 udisk 目录，可以看到刚才拷贝的 hello 文件了：

```
[root@Linter /]#
[root@Linter /]#
[root@Linter /]#
[root@Linter /]# cd /
[root@Linter /]# mkdir udisk
mkdir: can't create directory 'udisk': File exists
[root@Linter /]# mount /dev/sda1 udisk/
[root@Linter /]# cd udisk/
[root@Linter /udisk]# ls
MPR3202                         jdk-6u27-linux-i586.bin
README.diskdefines              ldlinux.sys
Remove_LiLi.bat                 lili.ico
SmartClean.ini                  md5sum.txt
android_gingerbread_m12.rar     pics
autorun.bak                     pool
autorun.inf                     preseed
boot                            syslinux
casper                          ubuntu
ch7026                          usb-creator.exe
dists                           wubi.exe
fastboot.rar                    x210_android_driver.rar
g210                            x210_usb.bin
hello                           x210ii
install                         x210ii android??????.pdf
[root@Linter /udisk]#
```

运行 hello：

```
./hello
```

打印信息如下：

```
[root@Linter /]#
[root@Linter /]#
[root@Linter /]# cd /
[root@Linter /]# mkdir udisk
mkdir: can't create directory 'udisk': File exists
[root@Linter /]# mount /dev/sda1 udisk/
[root@Linter /]# cd udisk/
[root@Linter /udisk]# ls
MPR3202                         jdk-6u27-linux-i586.bin
README.diskdefines              ldlinux.sys
Remove_LiLi.bat                 lili.ico
SmartClean.ini                  md5sum.txt
android_gingerbread_m12.rar     pics
autorun.bak                     pool
autorun.inf                     preseed
boot                            syslinux
casper                          ubuntu
ch7026                          usb-creator.exe
dists                           wubi.exe
fastboot.rar                    x210_android_driver.rar
g210                            x210_usb.bin
hello                           x210ii
install                         x210ii android??????.pdf
[root@Linter /udisk]# ./hello
hello,x210!
[root@Linter /udisk]#
```

表明，程序已经成功运行。

6.1.2 LED 测试程序

同样我们建立 LED 测试程序的编译目录：

```
mkdir  led
```

```
cd    led
vim    led.c
```

编辑如下内容：

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ioctl.h>

#define DEVICE_NAME      "/dev/vib"
#define LED_ON 0x11
#define LED_OFF 0x22

int main(int argc,char **argv)
{
      int on;
      int led_no;
      int fd;

      fd = open(DEVICE_NAME,0);//打开设备
      if(fd == -1)
      {
            printf("open device %s error \n",DEVICE_NAME);
            return 0;
      }
      else
      {
            printf("open device %s ok! \n",DEVICE_NAME);
      }

      while(1)
      {
            ioctl(fd,LED_ON);
            sleep(1);
            ioctl(fd,LED_OFF);
            sleep(1);
      }
      close(fd);
      return 0;
}
```

使用以下命令编译：

```
arm-none-linux-gnueabi    -o    led    led.c
```

将生成的可执行文件下载到开发板运行，仔细观察 LED 的效果。

6.1.3 mplayer 移植

在 windows 桌面，我们有强大的暴风影音，千千静听等来播放音视频，那么在 Linux 系统下，是否也有类似的软件呢？答案是肯定的，mplayer 不仅能播放音视频，还遵循 GNU 协议，开放源码。事实上，很多优秀的音视频播放器，都采用了 mplayer 核，所不同的只不过在这个基础上做了个外壳而已。下面我们从网上下载源码编译可执行文件，一起来体验 linux 下多媒体的乐趣！

第一步：从网上下载最新的 mplayer 源代码 MPlayer-1.0rc2.tar.bz2 以及 libmad-0.15.1b.tar.gz 并解压；

第二步：进入 libmad-0.15.1b 目录，配置编译环境：

```
./configure --enable-fpm=arm --host=arm-linux --disable-shared --disable-debugging --prefix=/usr/local/arm/4.3.3/lib CC=/usr/local/arm/4.3.3/bin/arm-none-linux-gnueabi-gcc
```

编译：

```
make
```

安装：

```
make  install
```

值得注意的是，由于这里使用的交叉编译工具为 4.3.3 版本，在编译时会弹出如下错误：

```
cc1: error: unrecognized command line option "-fforce-mem"
```

这是因为 gcc3.4 或更高版本，已经将 fforce-mem 去除了，我们只需要在 makefile 中找到该字符串，将其删除就可以了。

第三步：进入 MPlayer-1.0rc2 目录，配置编译环境：

```
./configure --cc=/usr/local/arm/4.3.3/bin/arm-none-linux-gnueabi-gcc --target=arm-linux --enable-static --prefix=/tmp/mplayer-rc2 --disable-win32dll --disable-dvdread --enable-fbdev --disable-mencoder --disable-live --disable-mp3lib --enable-mad --enable-libavcodec_a --language=zh_CN --disable-armv5te --disable-armv6 --with-extraincdir=/usr/local/arm/4.3.3/lib/include --with-extralibdir=/usr/local/arm/4.3.3/lib/lib --host-cc=gcc --enable-ossaudio --disable-ivtv
```

编译：

```
make
```

如果一切正常，在当前目录下将会生成 mplayer 文件。

第四步：将生成的 mplayer 拷到 U 盘中，同时拷备视频文件如*.avi，音频文件如*.mp3，将 U 盘插入开发板 U 盘接口，正常挂载 U 盘后，使用如下命令播放音视频：

```
./mplayer  *.avi
./mplayer  *.mp3
```

6.1.4 TSLIB 移植

待续

6.1.5 屏幕抓图工具 gsnap 移植

屏幕抓图的方法有多种，有不少爱好者自己动手写抓图小程序。这里我们使用 JPEG 库来处理。具体用到了 jpegsrc.v6b.tar.gz 和 gsnap.tar.gz 两个源码包。

一：安装 libjpeg

解压 jpeg 库源码包，进入根目录：

```
tar   zxf   jpegsrc.v6b.tar.gz
cd   jpeg-6b
```

二：配置编译环境：

```
./configure --prefix=/usr/local/arm/4.3.3/arm-none-linux-gnueabi
--exec-prefix=/usr/local/arm/4.3.3/arm-none-linux-gnueabi --enable-shared --enable-static
```

三：修改 makefile

CC = gcc 修改为 **CC = arm-none-linux-gnueabi-gcc**

AR = ar ac 修改为 **AR = arm-none-linux-gnueabi-ar ac**

AR2=ranlib 修改为 **AR2= arm-none-linux-gnueabi-ranlib**

确保—exec-prefix 已经设置为/usr/local/arm/4.3.3/arm-none-linux-gnueabi，如果没有，手动设置。

四：在/usr/local/arm/4.3.3/arm-none-linux-gnueabi 下建立 man/man1 目录：

```
cd   /usr/local/arm/4.3.3/arm-none-linux-gnueabi
mkdir   -p   man/man1
```

五：编译，安装

```
make
make   install
```

这时，在/usr/local/arm/4.3.3/arm-none-linux-gnueabi/man/man1 目录下将会生成以下文件：

```
cjpeg.1   djpeg.1   jpegtran.1   rdjpgcom.1   wrjpgcom.1
```

在/usr/local/arm/4.3.3/arm-none-linux-gnueabi/lib 目录下生成以下文件：

```
libjpeg.a   libjpeg.la   libjpeg.so   libjpeg.so.62   libjpeg.so.62.0.0
```

六：解压 gsnap

```
tar   zxf   gsnap.tar.gz
cd   gsnap
```

七：修改 makefile

```
all:
    arm-none-linux-gnueabi-gcc -g gsnap.c -ljpeg -o gsnap
clean:
    rm -f gsnap
```

八：编译，得到可执行文件 gsnap

```
make
```

九：将 jpeg 库文件复制到文件系统的 lib 目录，注意保持文件的链接属性

```
cp   -a   libjpeg.s*   “文件系统路径”/lib
```

十：将可执行文件 gsnap 复制到文件系统的 sbin 目录

```
cp   gsnap   “文件系统路径”/sbin
```

十一：重新制作文件系统，下载到开发板上，使用如下命令即可截获图形界面：

```
gsnap   1.jpg   /dev/fb0
```

同样可以将图片保持为 bmp,png 等其他格式。

6.1.6 数学函数库调用

建立程序编译路径：

```
mkdir    math
cd    math
vim    math.c
```

编辑如下内容：

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main(void)
{
    double a=9.0;
    printf("sqrt(%f)=%f\n",a,sqrt(a));

    return 0;
}
```

使用如下指令编译：

```
arm-none-linux-gnueabi-gcc -o math math.c –lm
```

将生成的可执行文件下载到开发板上运行，如下图所示：

```
[root@Linter /]#
[root@Linter /]#
[root@Linter /]# mount /dev/sdb1 udisk/
[root@Linter /]# cd udisk/
[root@Linter /udisk]# ls
MPR3202                         kernel_V20110512.tgz
README.diskdefines              ldlinux.sys
Remove_LiLi.bat                 led
SmartClean.ini                  lili.ico
android_gingerbread_m12.rar     math
app-ex_V20110511.tgz            md5sum.txt
autorun.bak                     pics
autorun.inf                     pool
boot                            preseed
casper                          syslinux
ch7026                          ubuntu
dists                           usb-creator.exe
fastboot.rar                    wubi.exe
g210                            x210_android_driver.rar
hello                           x210_usb.bin
install                         x210ii
jdk-6u27-linux-i586.bin         x210ii android??????.pdf
kernel120621.tgz
[root@Linter /udisk]# ./math
sqrt(9.000000)=3.000000
[root@Linter /udisk]#
```

6.1.7 多进程编程示例

在 Linux 下通用调用 fork 函数创建新的进程。调用 fork 时，系统将产生一个与当前进程相同的进程。它与原有的进程具有相同的数据，连接关系和在程序同一处执行时的连续性。通常将原有的进程叫父进程，新创建的进程叫子进程。

fork 调用将分两次返回，从父子进程返回。进程创建语法如下：

```
#include <unistd.h>
pid_t    pid;
pid = fork();
```

如果 pid 返回 0，表示说明从子进程返回，否则从父进程返回，此时返回的是进程的 ID 号。我们可以通过 getpid()函数来获得进程的 ID 号。

首先建立程序编译目录：

```
mkdir    process
cd    process
vim    process.c
```

编辑如下内容：

```
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>

main()
{
    pid_t pid;
    pid=fork();
    if (pid<0)
    {
        printf("fork is error!\n");
        return 1;
    }
    else if (pid == 0)
    {
        while (1)
        {
            printf("the child process is running now.pid=%d\n",getpid());
            sleep(1);//linux 延时函数,延时 1 秒
        }
    }
    else
    {
        while (1)
        {
            printf("the perent process is running now.pid=%d\n",getpid());
            sleep(1);
        }
    }
    return 0;
}
```

使用如下指令编译：

```
arm-none-linux-gnueabi-gcc -o process process.c
```

将生成的可执行文件下载到开发板运行，仔细观察串口监控信息：

```
autorun.inf                    pool
boot                           preseed
casper                         process
ch7026                         syslinux
dists                          ubuntu
fastboot.rar                   usb-creator.exe
g210                           wubi.exe
hello                          x210_android_driver.rar
install                        x210_usb.bin
jdk-6u27-linux-i586.bin        x210ii
kernel120621.tgz               x210ii android??????.pdf
[root@Linter /udisk]# ./process
the perent process is running now.pid=370
the child process is running now.pid=371
the perent process is running now.pid=370
the child process is running now.pid=371
the perent process is running now.pid=370
the child process is running now.pid=371
the perent process is running now.pid=370
the child process is running now.pid=371
the perent process is running now.pid=370
the child process is running now.pid=371
the perent process is running now.pid=370
the child process is running now.pid=371
^C
[root@Linter /udisk]#
```

6.1.8 makefile 编程示例

在上面的很多测试程序实例中，我们都使用的一条指令进行编译。在实际开发过程中，很多情况下都是会存在有很多文件，而且文件之间还会有错综复杂的关系。这时我们再靠敲指令来编译，就显得太繁琐，这时 makefile 就开始大显身手了。

makefile 就好比批处理文件，里面写了一系列集合，当运行 make 编译时，便会按 makefile 提供的命令及顺序完成编译。

这里我们给出三个文件：main.c，func.c，func.h。主程序在 main.c 中，在 main.c 中程序会调用 func.c 中的函数，func.c 中的函数又会用到 func.h 中定义的变量。

main.c 文件内容如下：

```
#include "func.h"

extern int fd;

int main(int argc,char **argv)
{
    fd = open(DEVICE_NAME,0);//打开设备
    if(fd == -1)
    {
        printf("open device %s error \n",DEVICE_NAME);
        return 0;
    }
    else
    {
        printf("open device %s ok! \n",DEVICE_NAME);
    }

    while(1)
    {
```

```
            glint_led();
        }
        close(fd);
        return 0;
}
```

该文件会调用 glint_led ()函数，这个函数在 func.c 中。func.c 的内容如下：

```
#include "func.h"

void glint_led(void)
{
        ioctl(fd,LED_ON);
        sleep(1);
        ioctl(fd,LED_OFF);
        sleep(1);
}
```

这里仅仅是一个读取按键的函数，供 main 函数调用。该函数需要用到了一些变量，另外还需要一些头文件支持，这些都存放在 func.h 中，其内容如下：

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ioctl.h>

#define DEVICE_NAME      "/dev/vib"
#define LED_ON 0x11
#define LED_OFF 0x22

int fd;
```

很明显，这是基于前面的 LED 测试程序，人为的分成的三个文件。我们的目的不在于分离代码，而在于学习 makefile 的编写方法。

当不使用 makefile 时，我们使用如下指令编译：

```
arm-none-linux-gnueabi-gcc -o mkfile main.c func.c
```

编译完成后，将会生成可执行文件 mkfile。将它下载到开发板上运行，和前面的按键测试完全相同。现在我们尝试编写第一个属于自己的 makefile:

```
mkfile:main.o func.o
        arm-none-linux-gnueabi-gcc -o mkfile main.o func.o
main.o:main.c
        arm-none-linux-gnueabi-gcc -c main.c -o main.o
func.o:func.c func.h
        arm-none-linux-gnueabi-gcc -c func.c -o func.o

clean:
        rm -f mkfile *.o
```

执行 make 后，编译器会依次编译 main.c 和 func.c 文件，生成 main.o 和 func.o 文件，最后将这两个.o 文件打包到可执行文件 mkfile 中。这时将 mkfile 文件下载到开发板运行，效果和前面的是一样的。我们可以执行 make clean 指令清除生成的.o 文件和可执行文件。

makefile 具有很强大的推理功能，我们完全可以简化上面的代码。优化后的代码如下：

```
OBJS=main.o func.o
CC=arm-none-linux-gnueabi-gcc

mkfile:${OBJS}
    ${CC} -o $@ $^
main.o:
func.o:func.h

clean:
    rm -f mkfile *.o
```

可见，这次比上面的完整版要简化多了。前面通过变量 OBJS 定义了要编译的源文件，变量 CC 给出了交叉编译工具。$@ 表示目标文件的全称，即 mkfile，$^表示所有被依赖的文件，并以空格分开，即 main.o func.o。后面的 clean 为清除指令，执行 make clean 后会执行 clean 后面的指令。需要注意的是，rm 指令后面千万不要使用$@符号来表征我们要删除的目标文件，因为这时候$@已经不再表示 mkfile 了，而表示 clean。同样，使用 make 指令编译，一样能够生成我们需要的目录文件 mkfile。

上面的 makefile 使用了变量以及预定义变量。第一句即定义了变量 OBJS，将它赋值为 main.o func.o，第二句定义了变量 CC，将它赋值为一个交叉编译工具定义。引用变量时，通过${*}表示，这里*表示前面定义的变量。

上面使用了$@和$^两个预定义变量，GNU make 主要有以下七种预定义变量：

$*	不包含扩展名的目标文件名称
$+	所有的依赖文件，以空格分开，以出现的先后为序，可能包含重复的依赖文件
$<	第一个依赖文件的名称
$?	所有的依赖文件，以空格分开，这些依赖文件的修改日期比目标的创建日期晚
$@	目标的完整名称
$^	所有的依赖文件，以空格分开，不包含重复的依赖文件
$%	如果目标是归档成员，则该变量表示目标的归档成员名称。例如，如果目标名称为 mytarget.so(image.o)，则$@ 为 mytarget.so，而$% 为 image.o。

对比以上几种编译方式，我们不然发现，其实最简的还是第一种，因为它就一句话就搞定了。那么在 makefile 中，我们是否也可以精简到只有一句话呢？答案是肯定的。我们继续利用 makefile 强大的推理功能进行简化，得到如下 makefile 代码：

```
OBJS=main.o func.o
CC=arm-none-linux-gnueabi-gcc

mkfile:${OBJS}
    ${CC} -o $@ $^

clean:
```

```
rm -f mkfile *.o
```

这次，makefile 真正编译的代码，就只有上面红色部分一条指令了。和前面比较，不难发现，单独对 main.c 和 func.c 两个文件编译的指令已经去掉了。前面我们提到，makefile 具有强大的推理功能，我们在生成目标文件 mkfile 时，makefile 会推理出它需要 main.c 和 func.c 两个文件，因此它首先就会去编译这两个文件，最后再执行目标文件的生成。因此我们完全可以将它们省去。

这里只是 makefile 的一点基础，读者可以借助于其他书籍对 makefile 作更深一层的了解。

第7章 其他产品介绍

7.1 核心板系列

X6410CV10
X210CV3
X210CV4
G210CV10
I210CV20
X4412CV2
X4418CV2
X6818CV3
X3288CV3

7.2 开发板系列

x6410 开发板
x210 开发板
g210 开发板
i210 开发板
x4412 开发板
x4418 开发板
X6818 开发板
X3288 开发板
ibox4412 卡片电脑
ibox4418 卡片电脑
ibox6818 卡片电脑

说明：产品详细规格，以及更多其他产品请关注九鼎创展官方网站和论坛。