Real6410 用户手册

（linux 部分）

目录

第一章 Real6410 开发板简介

1.1.硬件资源

Real6410是华天正科技推出的用于高端手持设备、微型智能控制设备的开发套件。采用韩国三星公司的ARM11内核的处理器S3C6410/S3C6410。该款套件核心板的尺寸仅相当于一个48mm*67mm 的方块的大小。Real6410套件由核心板和底板（外设板或称基本板）组成，核心板上集成三星 S3C6410处理器，128MB 的 DDR 内存以及 1GB 的 NANDFLASH，同时预留了 256K NORFLASH。为您的应用研发提供了充足的空间。底板上则提供以下外设接口：
1.两个四线 RS-232串口(COM0，COM1)
2.一个 USB HOST 接口
3.一个 10M/100M 自适应以太网接口，一个 TFT LCD 接口,一个触摸屏接口
4.一个 wm8987 sound 接口
5.一个 4x4 按键接口
6.一路视频输入（模拟 saa7113 或数字 ov9650，可以选择）
7.一路视频输出 TVOUT
8.一个 RTC 和 watchdog
9.一个 SDIO 接口 WIFI 模块
10.一个 SD 卡接口

核心板和底板配合即构成一个最小的完整应用系统。系统具有体积小、耗电低、处理能力强等特点，能够装载和运行嵌入式 Linux 操作系统。用户可以在这个系统平台上进行自主软件开发。Real6410套件中提供底板硬件电路图和硬件设计文档，极大的方便了用户进行硬件扩展开发。同时华天正科技提供完备的嵌入式 Linux 开发环境及丰富的开发调试工具软件。**S3C6410**微处理器的特性：

- ARM11 嵌入式处理器内核，主频可达 800MHz；
- 扩展总线最大频率 133MHz；
- 32 位数据总线和 32 位外部地址总线；
- 完全静态设计(0-667M)；
- 存储控制器(八个存储体)：
- 包含 SROM、 SRAM 控制器,NAND 控制器；
- 复位时引导芯片选择(8 比特、16 比特存储或 NAND 可供选择)；
- 五个三十二位定时器，（time0,time1 带有 PWM);
- 多达 64 个中断源的中断控制器；
- RTC；
- 四个 UART，Supports IrDA 1.0；
- 四个 DMA 控制器，每个 DMA 控制器有 8 个通道；（支持外设 DMA）
- 支持 STN 与 TFT LCD 控制器；
- 看门狗；
- IIS 音频接口；

- 两个 USB host 口,一个 USB device 口。
- IIC-Bus 接口；
- 两个串行外围接口电路（SPI）
- 三个 SD 卡接口（sopport 1/4/8 bit mode,rate up to 50MHz)；
- 自定义按键
- camera_if接口
- TV_out 接口
- MFC(多格式视频编解码)接口，支持 H263、H264、MPEG4 和 VC-1 硬件编解码。

Real6410 开发套件硬件主要结构：

Samsung S3C6410 处理器
1Gbytes 8 位 NAND FLASH
64Mbytes 32位 DDRRAM，共256MB
两个四线 RS-3 接口
一个 10M/100M 自适应以太网接口
camera 摄像头接口(可选 模拟TVP5150 或 ov9650)
两个 USB(一个 host,一个 device)接口
一个 SD 卡接口
自定义按键
AC97音频（使用 wm9713 芯片）
一个 F LCD 接口(可选两种不同接口的屏）
一个触摸屏接口。
MFC 接口
直流电源（需要客户自己购买）
复位建
运行状态指示 LED 灯
模块实现无线通讯功能

1.2.软件资源

对于 linux 部分提供以下的软件资源：

1. 引导程序版本
- s3c-u-boot-1.1.6

2. 内核版本
- s3c-Linux-2.6.28.4

3. 设备驱动
- 256M mDDR 支持
- LCD 驱动程序
- 三通道 MMC/SD 驱动
- 看门狗 watdog 驱动
- 实时时钟 RTC 驱动
- 2 通道 i2c 驱动
- 2 通道 spi 驱动
- 键盘接口 keybad 驱动
- GPIO 键盘驱动
- 触摸屏驱动
- 网卡芯片 DM9000AEP 驱动
- MFC 驱动
- TVENC、TVSCALER 驱动
- Rotator 驱动
- jpeg 驱动
- nand flash 驱动（2K page）
- onenand flash 驱动
- USB device 驱动、USB host 驱动、USB OTG 驱动
- fimc 驱动
- 2D、3D 加速器驱动
- ADC 驱动
- AES 驱动
- 电源管理驱动
- Camera 驱动
- SDIO WIFI 驱动
- 针对 nand 的 yaffs2、UBIFS 文件系统支持
- 声卡 WM9713 驱动（ALSA）

4. 文件系统
- ubifs/yaffs2/cramfs/fat32 文件系统

5. 图形界面
- qtopia-2.2.0
- QtE-4.5.2

6. 其他功能
- 提供支持 SD 卡启动的 u-boot，无需通过 JTAG 方式烧录 u-boot

- 通过 SD 方式升级系统，方便快捷
- 支持 USB 升级内核和文件系统

7. 编译器

- arm-none-linux-gnueabi-4.3.2

8. 调试工具

- DNW

第二章 Real6410 资源分配

2.1.物理地址分配

Address		Size（MB）	Description
0x0000_0000	0x07FF_FFFF	128	启动镜像区
0x0800_0000	0x0BFF_FFFF	64	内部 ROM
0x0C00_0000	0x0FFF_FFFF	64	Stepping Stone(8KB)
0x1000_0000	0x17FF_FFFF	128	
0x1800_0000	0x1FFF_FFFF	128	DM9000AEP
0x2000_0000	0x27F_FFFF	128	NAND(256M/1024M)
0x2800_0000	0x2FFF_FFFF	128	
0x3000_0000	0x37FF_FFFF	128	
0x3800_0000	0x3FFF_FFFF	128	
0x4000_0000	0x47FF_FFFF	128	
0x4800_0000	0x4FFF_FFFF	128	
0x5000_0000	0x5FFF_FFFF	256	DDR(128M/256M)
0x6000_0000	0x6FFF_FFFF	256	

2.2 Nand flash 地址分配

Address		Size	Description
0x0000_0000	0x0003_FFFF	256KB	Bootloader（u-boot）
0x0004_0000	0x003F_FFFF	3.75MB	Kernel（linux）
0x0040_0000	0x007F_FFFF	4MB	Cramfs file system
0x0080_0000	0x0FFF_FFFF	248MB	Real file system（ubifs）

2.3 启动方式的选择

Real6410 提供多种启动方式，可以根据拨码开关进行设置。目前只用到 NAND 和 SD 这 2 种，具体的设置方式如下：

启动方式	J1	J2	J3	J4	J5	J6	J7	J8
NAND	OFF	OFF	OFF	OFF	OFF	ON	ON	OFF
SD	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF

第三章 S3C6410 硬件结构简单分析

3.1 简介

S3C6410 的物理内存分成 Memory 和 Pheriperal 两部分，地址范围分别为 0x0~0x6fffffff 和 0x7fffffff。系统通过 SPINE 总线访问 Memory 空间，通过 PERI 总线访问 Pheriperal 空间。而为了适应不同外设的访问速度，又分别通过 AHB 总线访问 LCD、Camera、Accelerator 等高速外设，通过 APB 总线访问 iic、watchdog 等低速外设。

3.2 内存

Memory，又叫主内存，分为 4 大区域，分别是启动镜像区、内部内存区、静态内存区、动态内存区。
启动镜像区物理地址为 0x00000000~0x07ffffff，共 128MB。这个区域的作用正如它的名字所述，是用来启动系统的。但是这个范围内并没有实际的存储介质与之对应，只能在通过 OM[4:0]选择具体的启动介质后再把相应介质的物理地址映射到这个启动区，比如说选择了 IROM 启动方式后，就把 IROM 所占的地址空间映射为 0x00000000 开始的空间。

内部内存区物理地址为 0x08000000~0x0fffffff，共 128MB。这个区域对应着内部的内存地址，内部的 ROM 和 SRAM 都是分布在这个区间。其中，0x08000000~0x0bffffff 对应着内部 ROM，当然实际上内部的 ROM 也并没有 64MB 这么多，只有 32KB 是有实际存储介质的，这 32KB 是一个只读区，放的是 IROM 方式下的启动代码，选择 IROM 启动的时候首先运行的代码就是这一部分，称为 BL0，这部分代码由厂家固化。0x0c000000~0x0fffffff 对应内部 SRAM，实际可用的 SRAM 按照三星的手册是 4KB，其实这就是用于 nand flash 启动的 Steppingstone（但是这个 Steppingstone 是 8KB，这 2 者似乎有矛盾，不知道是不是我的理解不对）。

静态内存区物理地址为 0x10000000~0x3fffffff，共 6*128MB。这个区域用于访问挂在外部总线上的设备，比如说 SRAM、NOR flash、oneNand 等。这个区域被分割为 6 个 bank，每个 bank 为 128MB，数据宽度最大支持 16bit，每个 bank 通过 Xm0CS[5:0]来划定。和 S3C2410 不一样的是，bank2~bank5 能映射到 nand flash、CF 等非线性存储器，这并不是说可以通过 bank2~bank5 的地址段就能直接访问 nand flash、CF 内部的地址，相反，当映射到这些器件的时候这些 bank 的地址也不能再使用了，访问这些非线性存储器还是得通过 Pheriperal 空间的 AHB 总线进行，和 S3C2410 中的访问方式是一样的。不过有一个特例是，当 Xm0CS2 被映射到 nand flash 的时候，Steppingstone 的 4KB（or 8K？）SRAM 被映射到 bank2 开始的 4KB，而在以 nand flash 方式启动的时候 bank2 被映射到 0x00000000 开始的地方，实际上就是把 Steppingstone 映射到 0x0000000 了，这和 S3C2410 的情况还是相似的。

动态内存区物理地址为 0x40000000~0x6fffffff，共 3*256MB。其中第一个 256MB 为保留区，实际使用的动态内存区为 0x50000000~0x6fffffff，又分为 2 个区间，分别占 256MB，可以通过 DMC 的 Xm1CS[1:0]来进行着 2 个区间的选择。这个内存区主要是扩展 DRAM，最大可以扩展 512MB 的 DRAM。

Table 2-2. Device Specific Address Space

Address		Size(MB)	Description	Note
0x0000_0000	0x07FF_FFFF	128MB	Booting Device Region by XOM Setting	Mirrored Region
0x0800_0000	0x0BFF_FFFF	64MB	Internal ROM	
0x0C00_0000	0x0FFF_FFFF	64MB	Stepping Stone (Boot Loader)	
0x1000_0000	0x17FF_FFFF	128MB	SROMC Bank0	
0x1800_0000	0x1FFF_FFFF	128MB	SROMC Bank 1	
0x2000_0000	0x27FF_FFFF	128MB	SROMC Bank 2	
0x2800_0000	0x2FFF_FFFF	128MB	SROMC Bank 3	
0x3000_0000	0x37FF_FFFF	128MB	SROMC Bank 4	
0x3800_0000	0x3FFF_FFFF	128MB	SROMC Bank 5	
0x4000_0000	0x47FF_FFFF	128MB	Reserved	
0x4800_0000	0x4FFF_FFFF	128MB		
0x5000_0000	0x5FFF_FFFF	256MB	DRAM Controller of the Memory Port1	
0x6000_0000	0x6FFF_FFFF	256MB		

3.3 S3C6410 的时钟结构图：

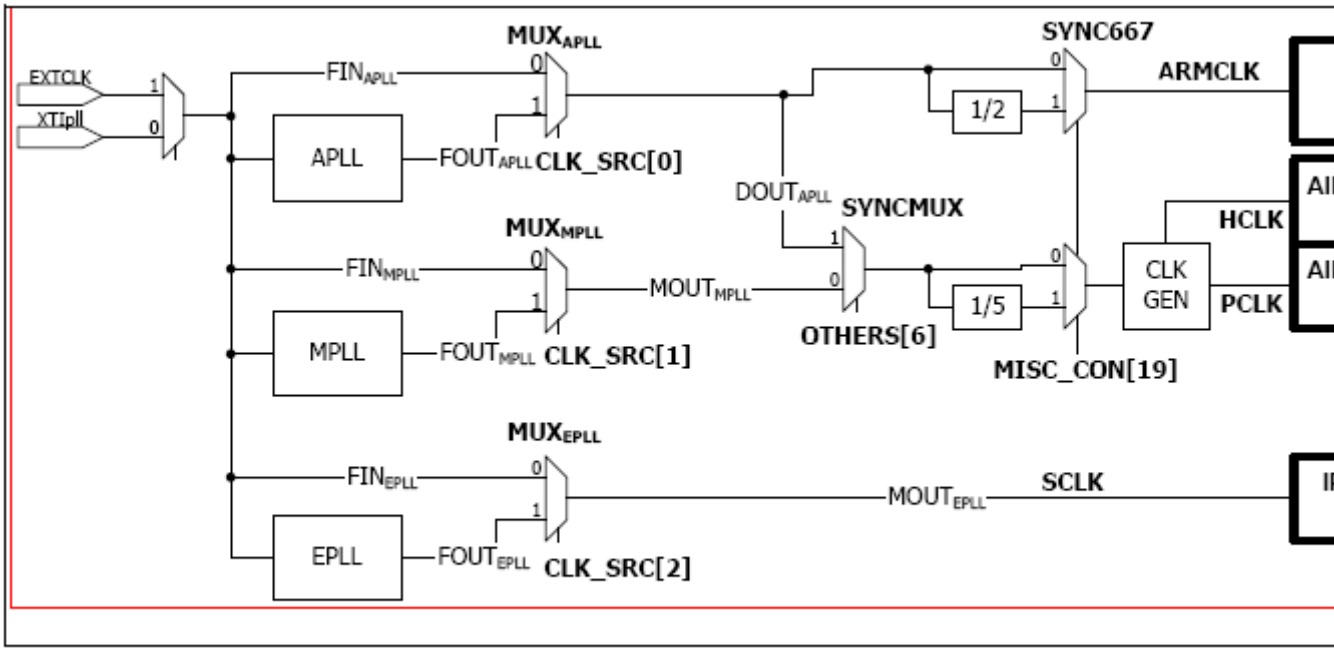


Figure 3-4. Clock generation from PLL outputs

由此可以看到 S3C6410 有 3 个 PLL，分别是 APLL、MPLL、EPLL，其中 APLL 为 ARM11 提供时钟，MPLL 为挂在 AXI、AHB、APB 上的设备提供时钟，EPLL 为需要特殊速率时钟的外设提供时钟信号。

3.4. S3C6410 的总线结构图：

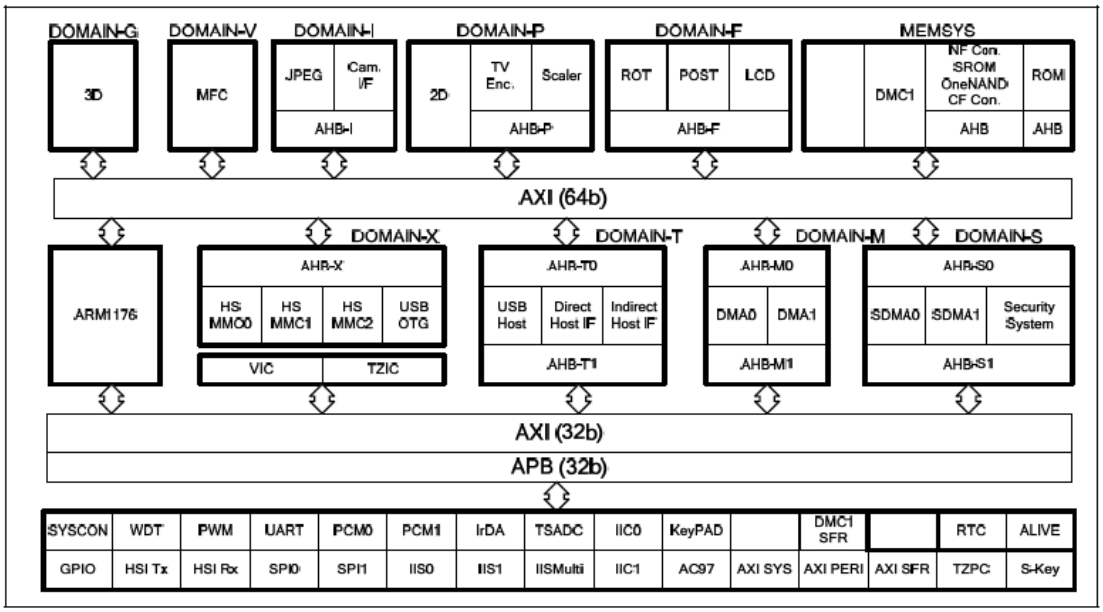


Figure 3-1. S3C6410X block diagram

以上这个图清楚地展示了芯片内部的各个设备和总线之间的互联关系。

3.5. S3C6410 启动方式。

这里主要是引用网友的一篇写得很好名为《S3C6410 启动模式介绍》文章上的内容：S3C6410 所支持的启动模式：

3.5.1 NORFlash启动

通过 Nor Flash 启动，此时 OM[4:1]为 0100 或 0101，对应 8bit 和 16bit。

3.5.2 NandFlash启动

虽然在 S3C6410 User Manual 中没有提到，但是也是支持的，从 S3C6400 User Manual 可以找到。OM[4:1]四个硬件管脚决定了 Nandflash 启动，以及支持的 Nandflash 的类型，包括大 Page 和小 Page，地址周期为 3，4，5。当然，XSELNAND 管脚也要为 1。

3.5.3 OneNAND启动

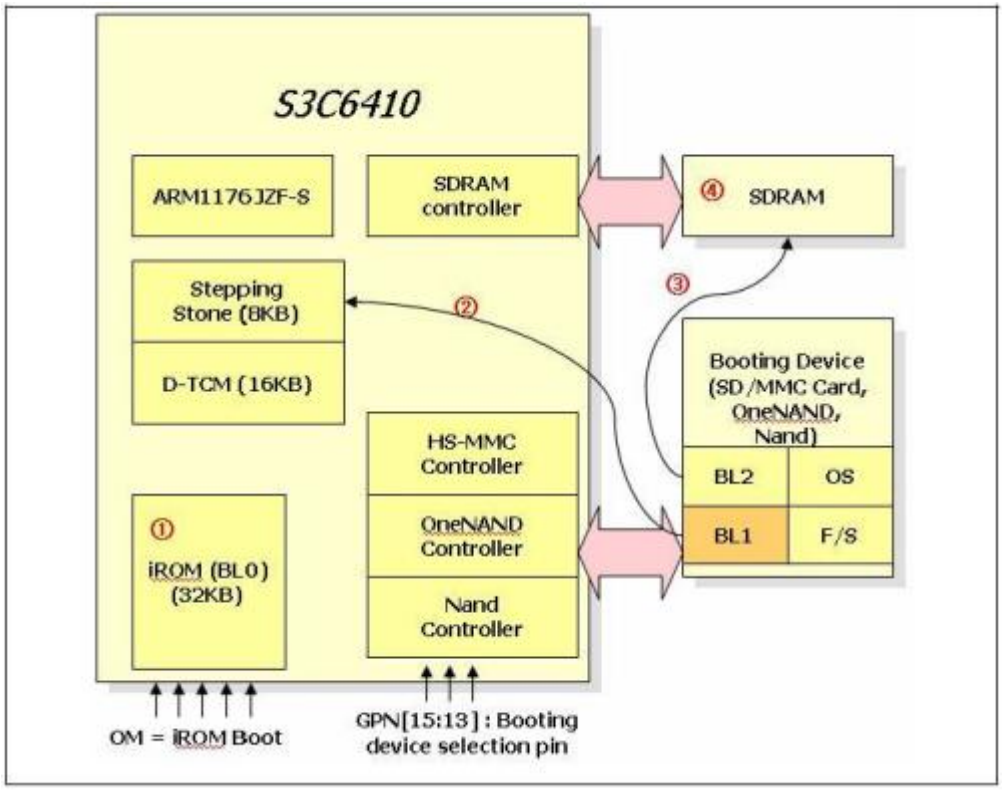
首先 XSELNAND 管脚为 0，其次 OM[4:1]为 0110，为 OneNand 启动模式。

3.5.4 MODEM启动

当 OM[4:1]为 0111 的时候，为 MODEM 启动。S3C6410 通过 MODEM 接口下载 boot 代码到内部 RAM 中，然后进行引导。

3.5.5 IROM启动

当 OM[4:1]为 1111 的时候，从 Internal ROM 中启动，此时 GPN[15:13]用于识别设备的类型。这种模式以前没见过，这里具体介绍一下。
IROM 模式可以支持 MoviNand，SD/MMC，iNand，OneNand 和 Nand 等。关于 IROM 的引导，具体过程如图：



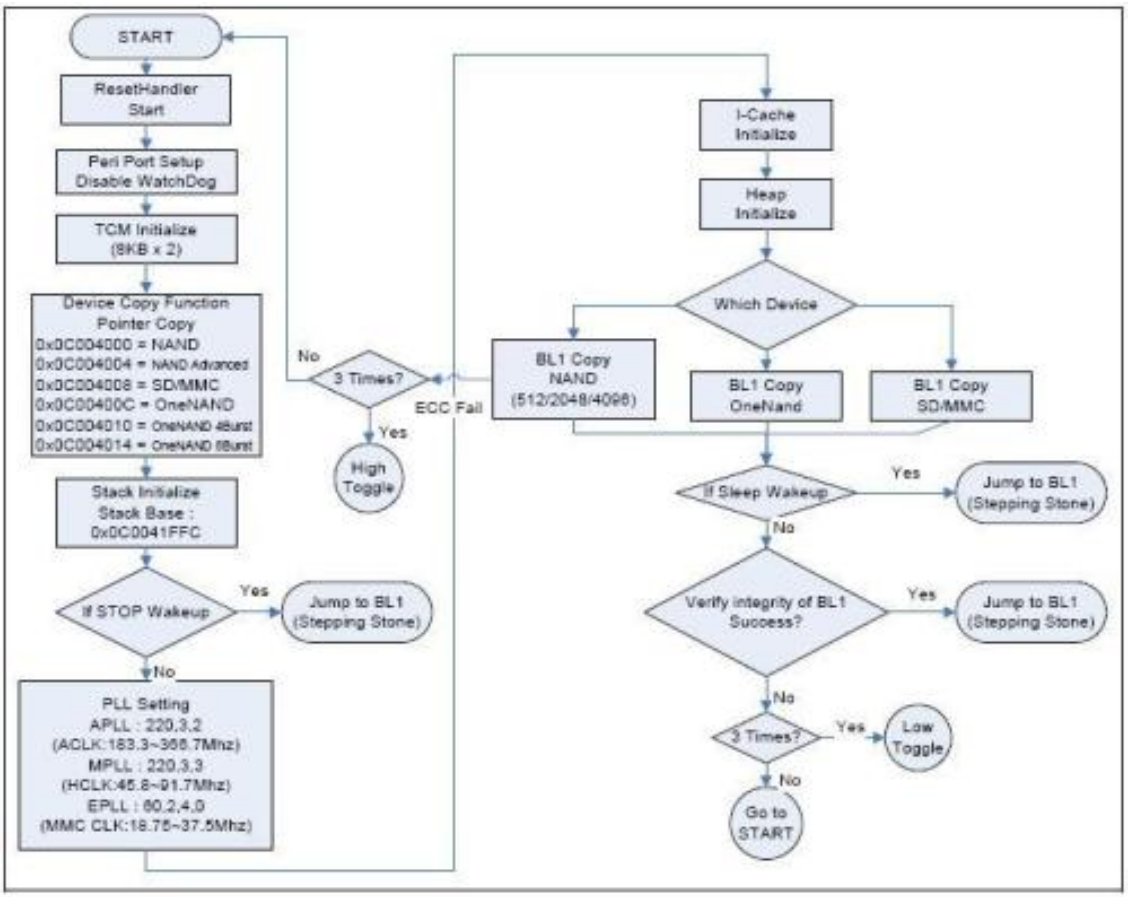
1. 处理器上电后，当 OM[4:1]=1111 时，运行 iROM 中的程序，这个程序被称为 Bootloader0(BL0)，它会做一些初始化的工作。
2. 然后根据 GPN[15:13]的管脚设置，选择从相应的设备(SD/MMC/OneNand/Nand)中的指定区域读取 4KB 的程序到 SteppingStone 中运行，这段代码被称为 Bootloader1(BL1)。
3. BL1 可以初始化系统时钟，UART，SDRAM 等设备，然后拷贝 Bootloader2(BL2)到 SDRAM 中。
4. 跳转到 SDRAM 中的 BL2，继续运行，BL2 可以支持更强大的功能，可以将 OS 加载到 SDRAM 中，然后运行 OS。

整个过程中，IROM 是最先被运行的，它会首先做一些初始化，具体 IROM 的流程如下：

1. 禁用 Watch-dog
2. 初始化 TCM

3. 初始化设备拷贝函数，用于拷贝 BL1 到 SteppingStone 中
4. 初始化栈区域
5. 初始化 PLL
6. 初始化指令 Cache
7. 初始化堆区域
8. 拷贝 BL1 到 SteppingStone 中
9. 验证 BL1
10. 跳转到 SteppingStone 中运行

还是看一下流程图吧，理解起来会更直观一些，IROM 启动流程如图：



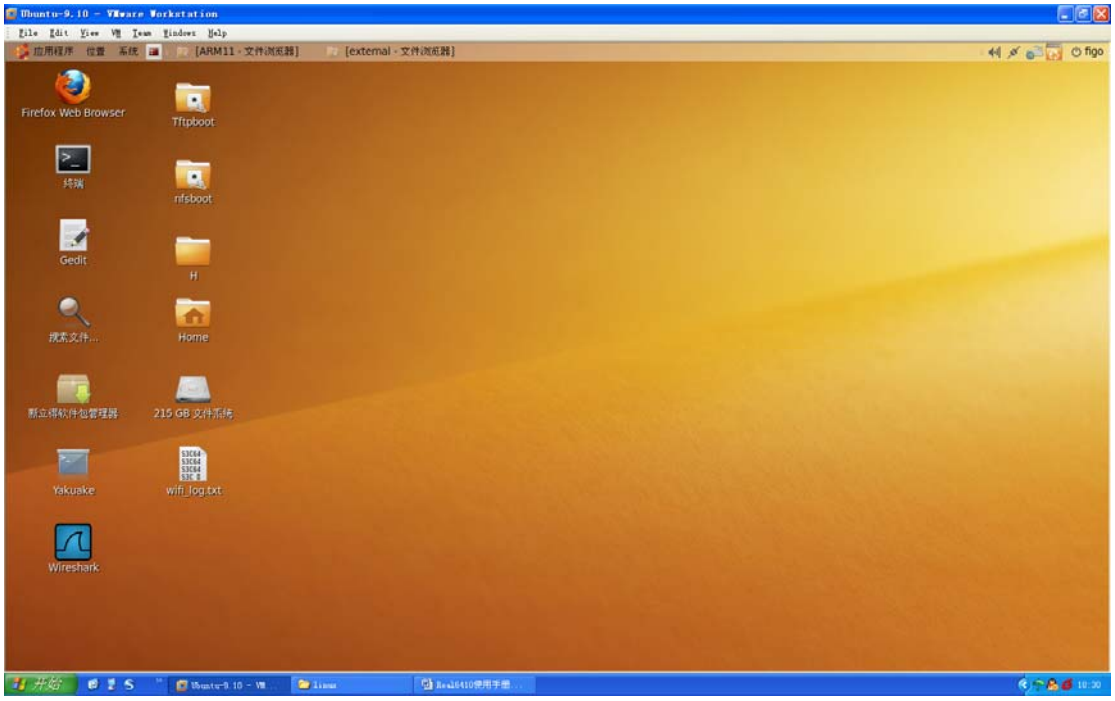
第四章 嵌入式 Linux 开发环境的建立

4.1 vmware+Ubuntu 的安装。

要进行嵌入式 linux 的开发首先要按照好一个主机开发环境，因为嵌入式 linux 下的大部分开发工作都是在 pc 中开发的。如果你的机器足够好，那么建议首先在 Windows 下安装一个虚拟机软件 vmware，毕竟 Windows 下可用的工具比较多，再在 vmware 基础上安装一个桌面版本的 Linux 系统。这里使用的是 vmware-7.0 和 Ubuntu-9.10，这 2 个软件都可以在网上下载到。安装过程比较冗长，这里不一一说明，可以参考《附录四 vmware 下安装 Ubuntu 教程》。

注意：有用户反映在 Ubuntu-9.10 下编译 qtopia-2.2.0 失败，这是由于 Ubuntu 没有预装相当多的库，因此要编译使用 qtopia-2.2.0 的用户可以考虑使用 Redhat AS4 的完全安装版，我们的 Qtopia-2.2.0 就是使用这个操作系统编译的。

安装结束后的界面如下：



不过在进行下面的开发的时候会发现默认下 Ubuntu 缺少很多需要用到的软件包，这要根据自己的实际情况使用 Ubuntu 的 apt 命令在线安装这些软件包。

4.2 建立 tftp 服务器。

在嵌入式 linux 开发过程中需要使用 tftp 方式从 Linux 主机下载文件到板子中，因此需要在主机 linux 系统中安装 tftp 服务器。

Ubuntu-9.10 中安装 tftp 服务器的方法如下：

4.2.1 安装程序

通过软件管理安装 tftp tftpd，前者是客户端，后者是服务程序。系统根据依赖会选上 openbsd-inetd。在 Ubuntu 的终端下输入命令如下：

sudo apt-get install tftp tftpd

4.2.2 在根目录/目录下建一个 tftpboot，把属性改成 777

cd /
sudo mkdir tftpboot
sudo chmod 777 tftpboot

4.2.3 修改存放目录

sudo vi /etc/inetd.conf
tftp dgram udp wait nobody /usr/sbin/tcpd /usr/sbin/in.tftpd /tftpboot

4.2.4 重新启动服务：

sudo /etc/init.d/openbsd-inetd restart
sudo in.tftpd -l /tftpboot

4.2.5 测试 tftp 服务器

在/tftpboot 文件夹下新建立一个文件
cd /tftpboot
touch test
进入另外一个文件夹
tftp 127.0.0.1
tftp> get test
是不是工作正常了呢？

现在把你编译好的内核文件拷贝到/tftpboot 下面，就可以使用 u-boot 的 tftp 命令加载内核到目标板内存了

4.3 建立 nfs 服务器

在嵌入式 linux 开发的时候，常常需要使用 nfs 以方便程序的调试。使用 nfs，用户可以将板子要用到的根文件系统放在主机目录下，开发板则通过以太网挂载到这个目录并将这个目录下的文件作为根文件系统的内容，这样用户的程序更新后不比重新烧写板子的根文件系统便能被重新使用，这点能够大大加快程序的调试。

Ubuntu 下安装 nfs 服务器的步骤如下：

4.3.1 进行 NFS 服务器端与客户端的安装：

sudo apt-get install nfs-kernel-server nfs-common portmap
安装客户端的作用是可以在本机进行 NFS 服务的测试。

4.3.2 配置 portmap

两种方法任选一种就可以：
(1):sudo emacs /etc/default/portmap
去掉 -i 127.0.0.1
(2)sudo dpkg-reconfigure portmap
运行后选择“否”
另外很重要的一点，要用 sysv-rc-conf (而不是 chkconfig)工具查看一下当前 nfs 和 portmap 的状态，若是 off，则用 sudo sysv-rc-conf portmap on 或 sudo sysv-rc-conf nfs-kernel-server on 打开

4.3.3 配置挂载目录和权限

emacs /etc/exports
我的配置如下：
/etc/exports: the access control list for filesystems which may be exported
to NFS clients. See exports(5).
#
Example for NFSv2 and NFSv3:
/srv/homes hostname1(rw,sync) hostname2(ro,sync)
#
Example for NFSv4:
/srv/nfs4 gss/krb5i(rw,sync,fsid=0,crossmnt)
/srv/nfs4/homes gss/krb5i(rw,sync)
#
/nfsboot *(rw,sync)
解释一下：
#后面的都是解释
/nfsboot 是 NFS 的共享目录，*表示任何 IP 都可以共享这个目录，你可以改为受限的 IP，rw 表示的是权限，sync 是默认的。

4.3.4 更新 exports 文件

只要你更改了/etc/exports, 你不可以通过 sudo exportfs -r 来更新 这个文件

4.3.5 重启 NFS 服务

sudo /etc/init.d/nfs-kernel-server restart 重启 nfs 服务

4.3.6 进行测试

尝试一下挂载本地磁盘(我的 linux 系统 IP 为 202.198.137.18，将/home/nfsboot 挂载到/mnt)
$ sudo mount 202.198.137.18:/nfsboot /mnt
运行 $ df 看看结果
$ sudo umount /mnt

4.4 交叉编译链的安装。

在主机上用来编译其他类型机器可执行代码的编译器就叫交叉编译器，我们进行嵌入式linux的开发的主机的处理器大部分都是x86，而我们的嵌入式系统的处理器有可能是arm、MIPS等非x86 处理器，这时候必须使用arm、MIPS等交叉编译器才能编译出这些处理器能够执行的代码。这里我们使用的是ARM公司提供的新一代的arm交叉编译器EABI编译器。

符合EABI标准交叉编译器： arm-none-linux-gnueabi-4.3.2 with EABI

EABI 说明：

交叉编译器在编译的时候，对于浮点运行会预设硬浮点运算 FPA(Float Point Architecture)，而没有 FPA 的 CPU，比如 SAMSUNG S3C2410/S3C2440，会使用 FPE(Float Point Emulation 即软浮点)，这样在速度上就会遇到极大的限制，使用 EABI(Embedded Application Binary Interface)则可以对此改善处理，ARM EABI 有许多革新之处,其中最突出的改进就是 Float Point Performance,它使用 Vector Float Point(矢量浮点)，因此可以极大提高涉及到浮点运算的程序性能。

优点：

更早的 arm 编译器版本，比如 arm-linux-gcc-3.4.1 不支持 armv6 构架，因此已经不能再使用了，但是之前的相当多的软件还是使用 arm-linux-gcc 编译器，如 qtopia-2.2.0 等。为此，我们提供了可以同时满足这 2 种需求的交叉编译器 arm-none-linux-gnueabi-4.3.2。该编译器使用了新的 glibc 库 2.8，并在编译器中预先安装好各种需要用到的库文件，如编译 qtopia 时需要用到的 jpeg、zlib、libts、libuuid 等等，在编译我们提供的软件系统时不需要额外安装任何第三方函数库，这点在编译 qtopia 的时候非常变得非常有意义。

使用这个编译器，你可以编译：

- linux 内核(linux-2.6.28.4)
- qtopia-2.2.0 图形系统
- busybox
- u-boot
- 其他很多 linux 应用程序(如 web server, boa, madplay 等程序)

首先这可以提高程序的浮点运算性能，其次你可以不必把时间花费在切换不同的编译器上。

对于 linux 所有源代码的编译都使用 ARM 公司推出的新一代 EABI 编译器。

应用光盘中提供了这 2 个编译器。将 linux-source/arm-none-linux-gnueabi-4.3.2.tar.bz2(或者名为 arm-2008q3.tar.bz2)这个文件复制到 linux 主机的/root 目录下，并在终端中分别执行下面的解压缩命令：

tar –xjvf arm-none-linux-gnueabi-4.3.2.tar.bz2

即可得到 arm-none-linux-gnueabi-4.3.2 目录。

为了使用方便，还可以编辑/etc/bash.bashrc 文件添加把编译器路径到环境变量 PATH 中，只要在这个文件中添加下面这 2 个语句即可：

PATH=/root/ arm-none-linux-gnueabi-4.1.0/bin:$PATH

Export PATH

编辑完毕后使用 source /etc/bash.bashrc 命令执行以下这个文件，让设置生效，之后再输入：

arm-none-linux-gnueabi-gcc –v

如果输出下面的信息则表面设置成功：

```
figo@figo-desktop:~$ arm-none-linux-gnueabi-gcc -v
Using built-in specs.
Target: arm-none-linux-gnueabi
Configured with: /scratch/julian/lite-respin/linux/src/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 --enable-languages=c,c++ --enable-shared --enable-symvers=gnu
--enable-__cxa_atexit --with-pkgversion='Sourcery G++ Lite 2008q3-72'
--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/julian/lite-respin/linux/install/arm-none-linux-gnueabi/libc
--with-gmp=/scratch/julian/lite-respin/linux/obj/host-libs-2008q3-72-arm-none-linux-gnueabi-i68
6-pc-linux-gnu/usr
--with-mpfr=/scratch/julian/lite-respin/linux/obj/host-libs-2008q3-72-arm-none-linux-gnueabi-i68
6-pc-linux-gnu/usr --disable-libgomp --enable-poison-system-directories
--with-build-time-tools=/scratch/julian/lite-respin/linux/install/arm-none-linux-gnueabi/bin
--with-build-time-tools=/scratch/julian/lite-respin/linux/install/arm-none-linux-gnueabi/bin
Thread model: posix
gcc version 4.3.2 (Sourcery G++ Lite 2008q3-72)
```

我们为了编译 qtopia 等软件的方便同时提供了 arm-linux-gcc 形式的调用命令，输入 arm-linux-gcc –v，如果没有意外将输出与上面同样的信息。

4.5 mkcramfs 的安装

有时候需要制作 cramfs 格式的文件系统，而这个工具对单个文件有 16MB 限制，我们提供了没有这个限制的 mkcramfs 工具，将这个文件复制到/usr/local/bin 目录下即可。之后在终端中输入 mkcramfs 测试，如果输出下面的内容便表面可以使用了：

```
figo@figo-desktop:~$ mkcramfs
usage: mkcramfs [-h] [-e edition] [-i file] [-n name] dirname outfile
```

```
-h          print this help
-E          make all warnings errors (non-zero exit status)
-e edition  set edition number (part of fsid)
-i file     insert a file image into the filesystem (requires >= 2.4.0)
-n name     set name of cramfs filesystem
-p          pad by 512 bytes for boot code
-s          sort directory entries (old option, ignored)
-v          be more verbose
-z          make explicit holes (requires >= 2.3.39)
dirname     root of the directory tree to be compressed
outfile     output file
```

注意：我们没有提供 mkyaffsimage、mkfs.ubifs 等可读写文件系统镜像制作工具，而是结合 mkcramfs 来使用 yaffs、ubifs 等格式。

第五章 快速上手

Uboot、kernel、rootfs 的镜像最后都要烧在 nandflash 下，因此先了解一下 nandflash 的分区情况以避免出现前后覆盖的情况：

分区名称	地址范围	分区描述
bootloader	0x00000000~0x0003FFFF	烧写 Uboot 的分区
kernel	0x00040000~0x003FFFFF	烧写 linux 内核的分区
cramfs	0x00400000~0x007FFFFF	烧写 cramfs 格式文件系统（用作备份）
ubifs	0x00800000~0x3FFFFFFF	烧写 ubifs 格式文件系统(真正的文件系统)

5.1 uboot 烧写

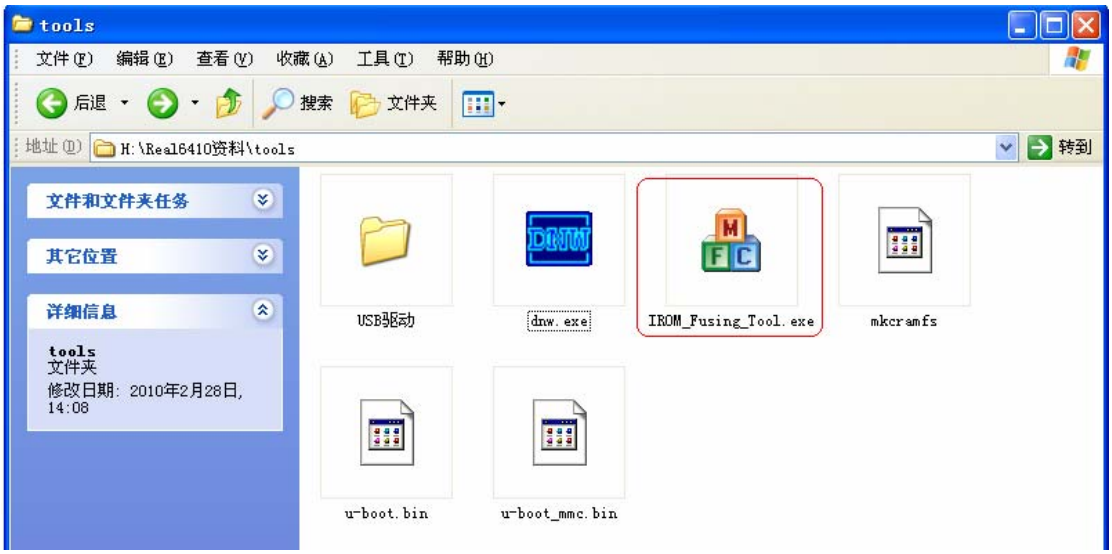
5.1.1.SD 启动的 uboot 烧写

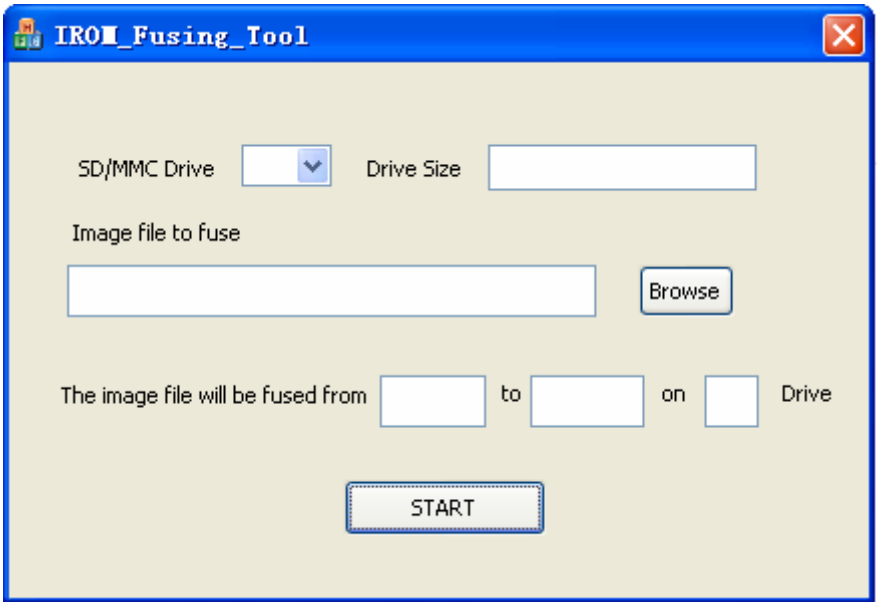
在没有仿真器等程序烧录工具的情况下，对于原本没有任何程序的板子，可以使用 S3C6410 的 MMC/SD 启动功能来烧写一个可用的 uboot。

注意：本文的实验都是基于 512MB 的 SD 做的，2G 的金士顿 SD 也试过没有问题，建议读者也使用这种容量的 SD，对于 2G 以上的 SD 的支持并没有经过试验。

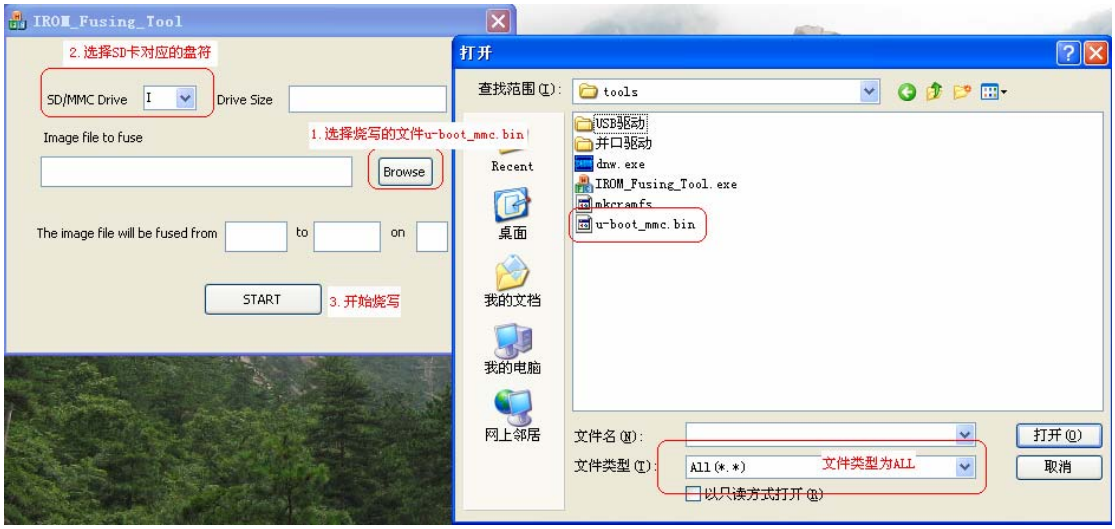
烧写过程如下：

在 Windows 下，通过光盘中的 tools/IROM_Fusing_Tool 工具将 MMC/SD 启动的 uboot 烧写到 SD 中。

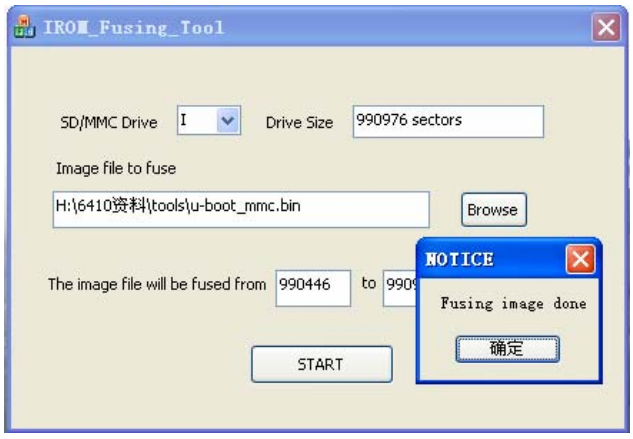




把格式为 FAT32 的 SD 卡通过 SD 读卡器插进 PC，并如下图步骤所示进行操作：



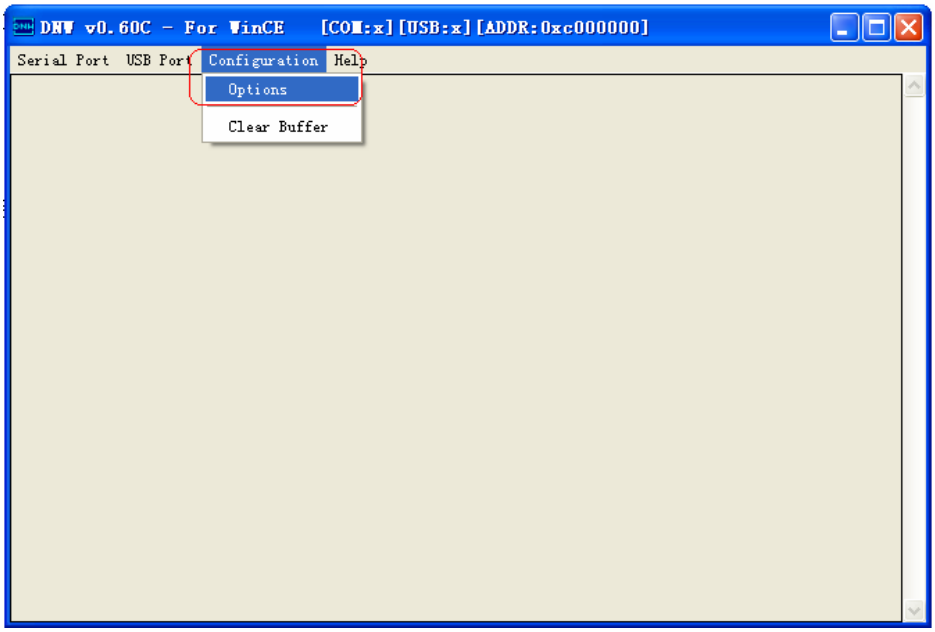
注意这里要烧写 SD 启动的 u-boot_mmc.bin。成功后如下图所示：

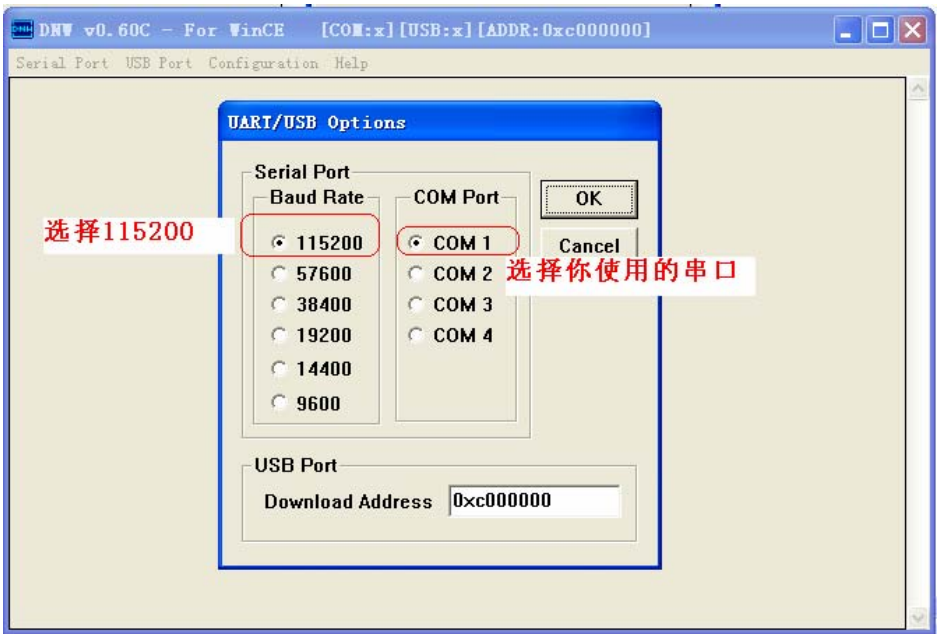


把 SD 取出并插入到板子的 SD 插槽中，通过设置板子的拨码开关使板子启动方式为 SD 启动，设置方式在 2.3 节中已经做了说明。

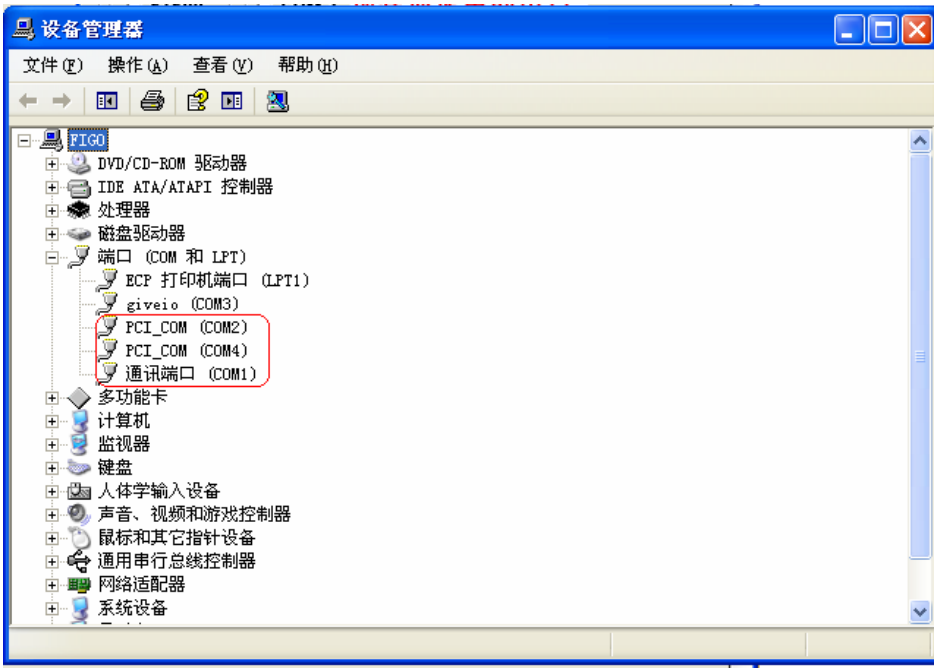
打开光盘中的 tools/dnw 工具并对这个工具进行设置：

第一步设置所使用的串口端口：



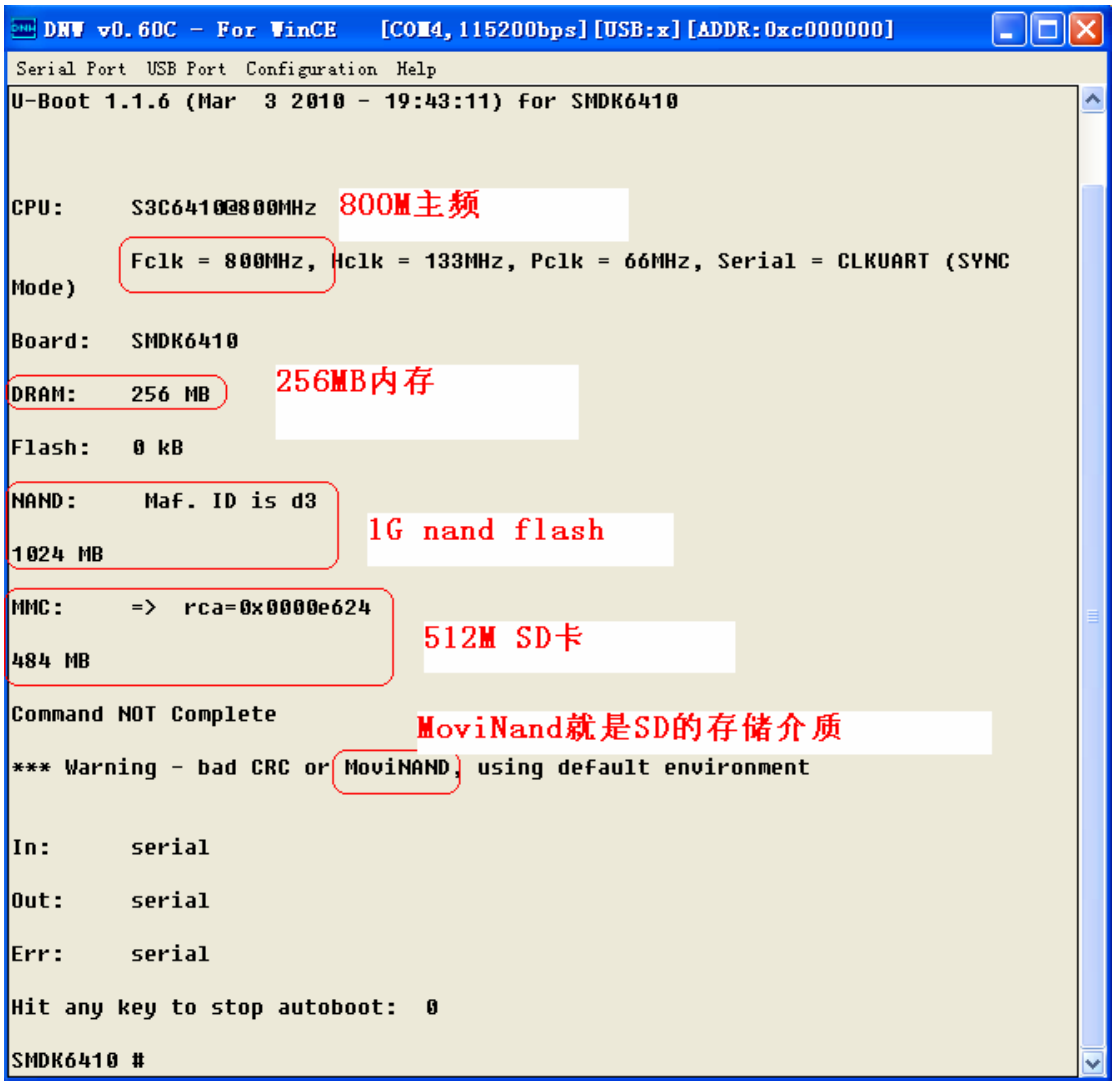


串口的端口号情况可以查看 Windows 的设备管理器：



第二步是连接这个串口设备：

Dnw 工具可用后便可以板子上电，如果没有意外，dnw 中将打印出 uboot 的启动信息：



5.1.2 NandFlash 启动的 uboot 烧写

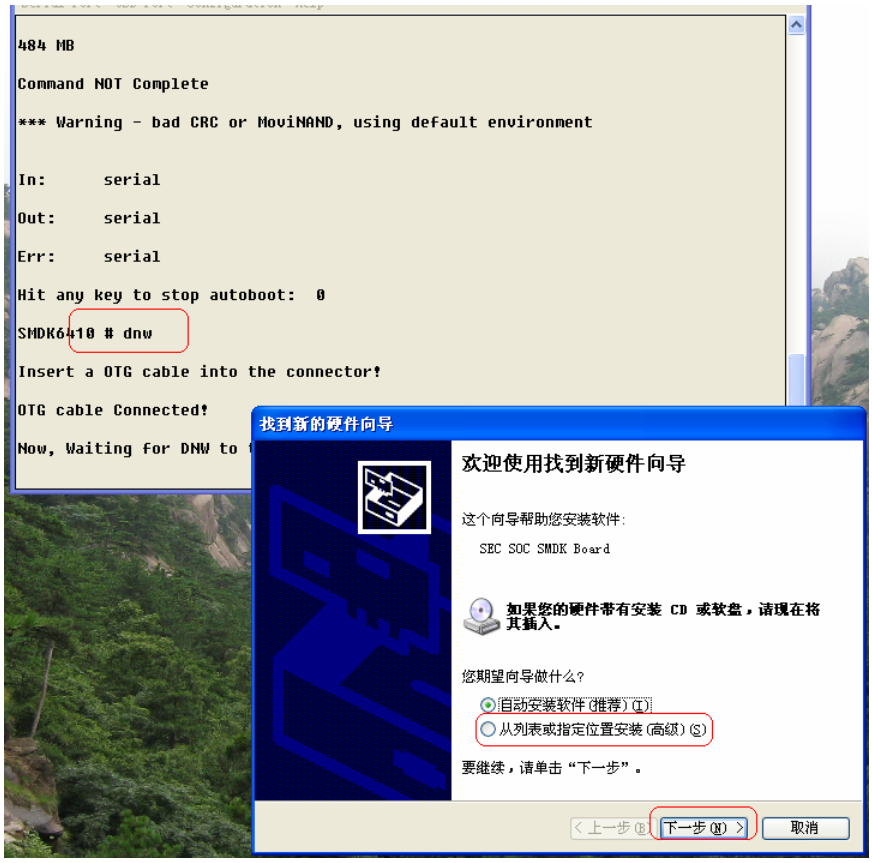
在有 SD 启动的 uboot 后，便可以利用这个 uboot 来将其他代码烧写到板子的 Nandflash 中了。这里演示如何将 Nandflash 启动的 uboot 烧写到 Nand 中。步骤如下：

如果到目前为止还不能使用 Ubuntu 中的 tftp 网络下载，那么可以在 Windows 下使用 USB 下载的方式将 uboot 下载到板子中，这里先以这种方式进行演示。

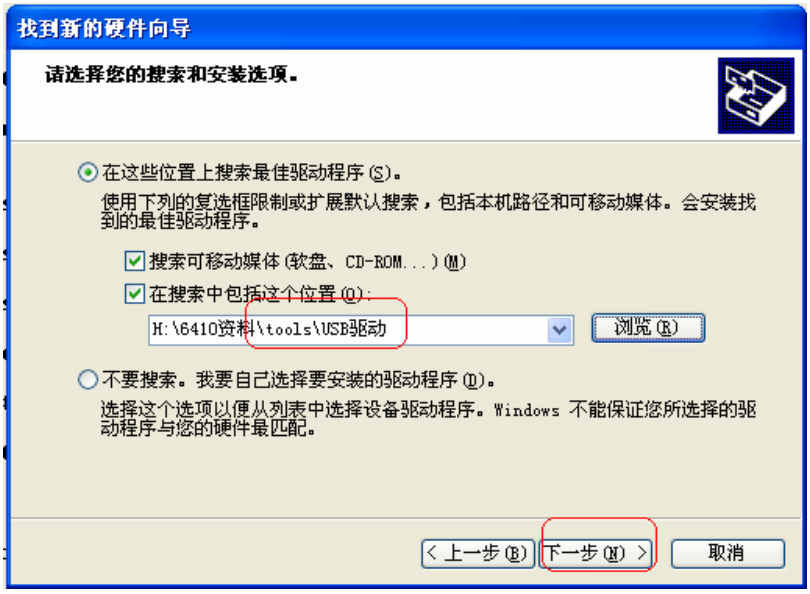
第一步，通过 USB 将 uboot 下载到内存中

输入 USB 下载命令 dnw：

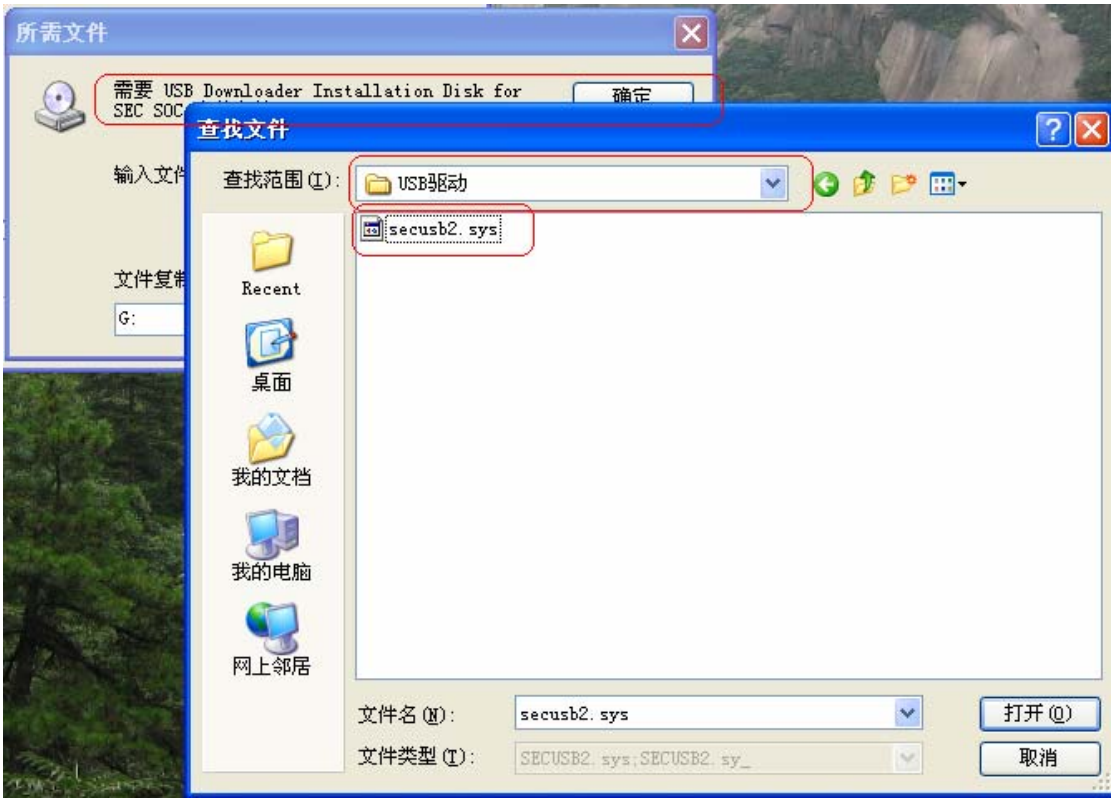
如果你是第一次使用，会提示安装板子的 USB 驱动程序，如果驱动已经安装则可以跳过这几步。



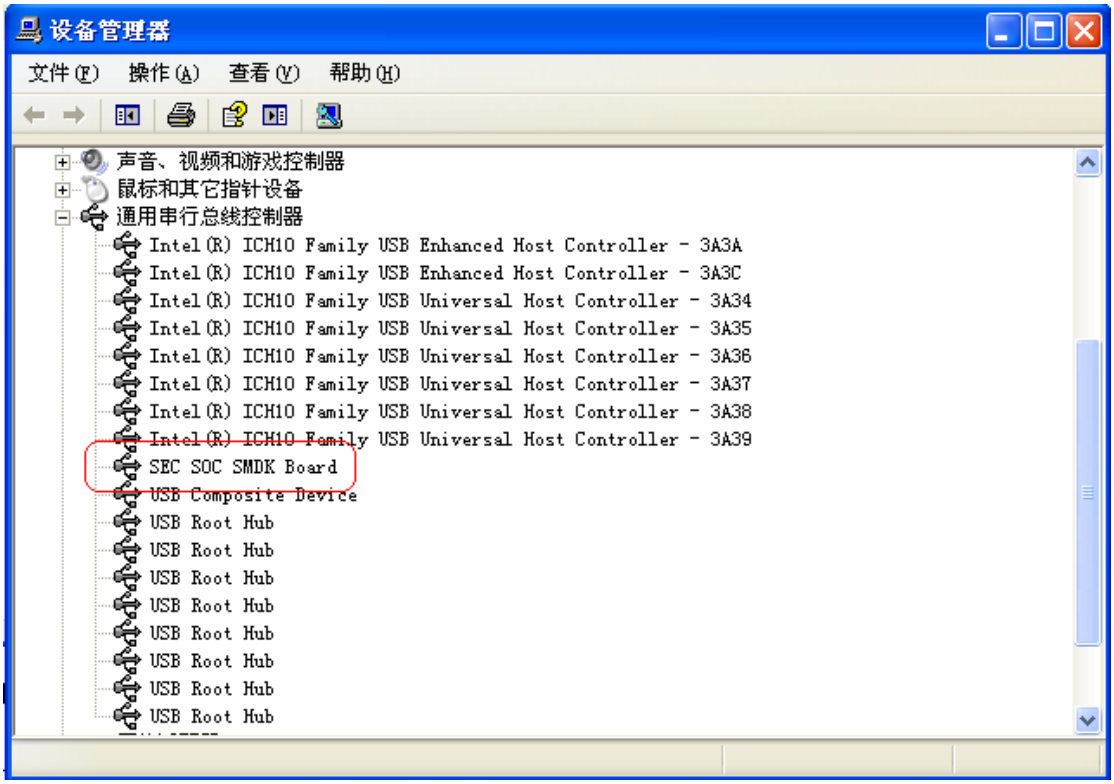
找到光盘中 tools/USB 驱动中提供的 USB 驱动：



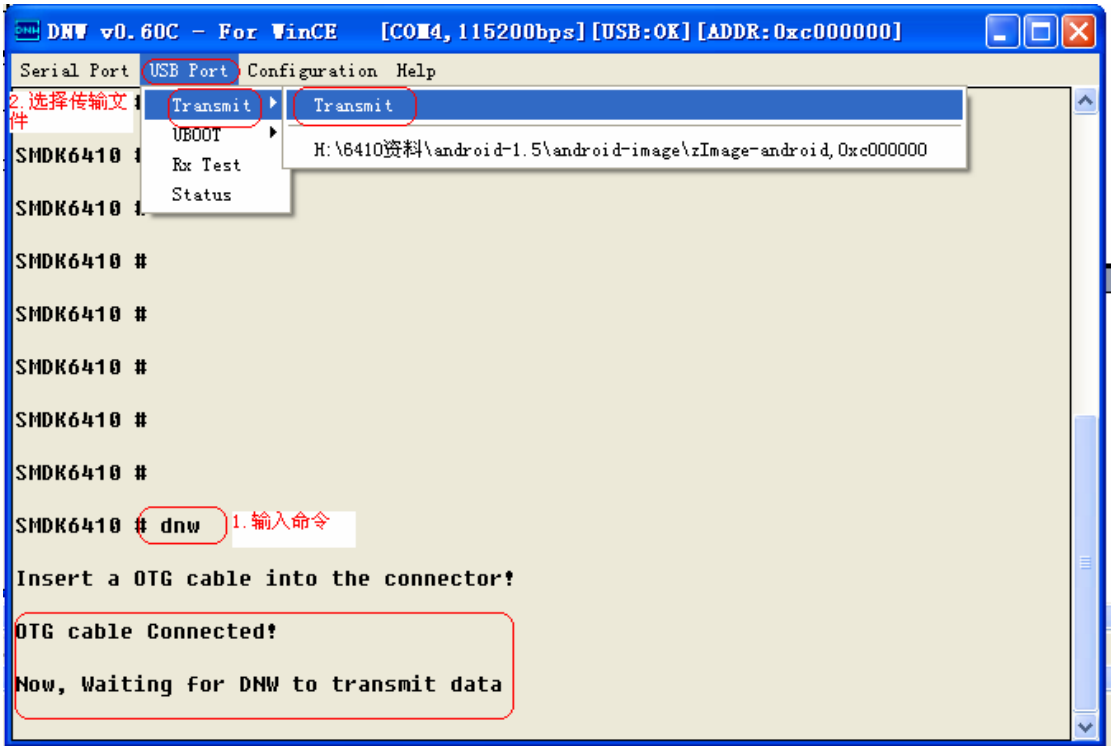
根据提示找到 sys 文件：



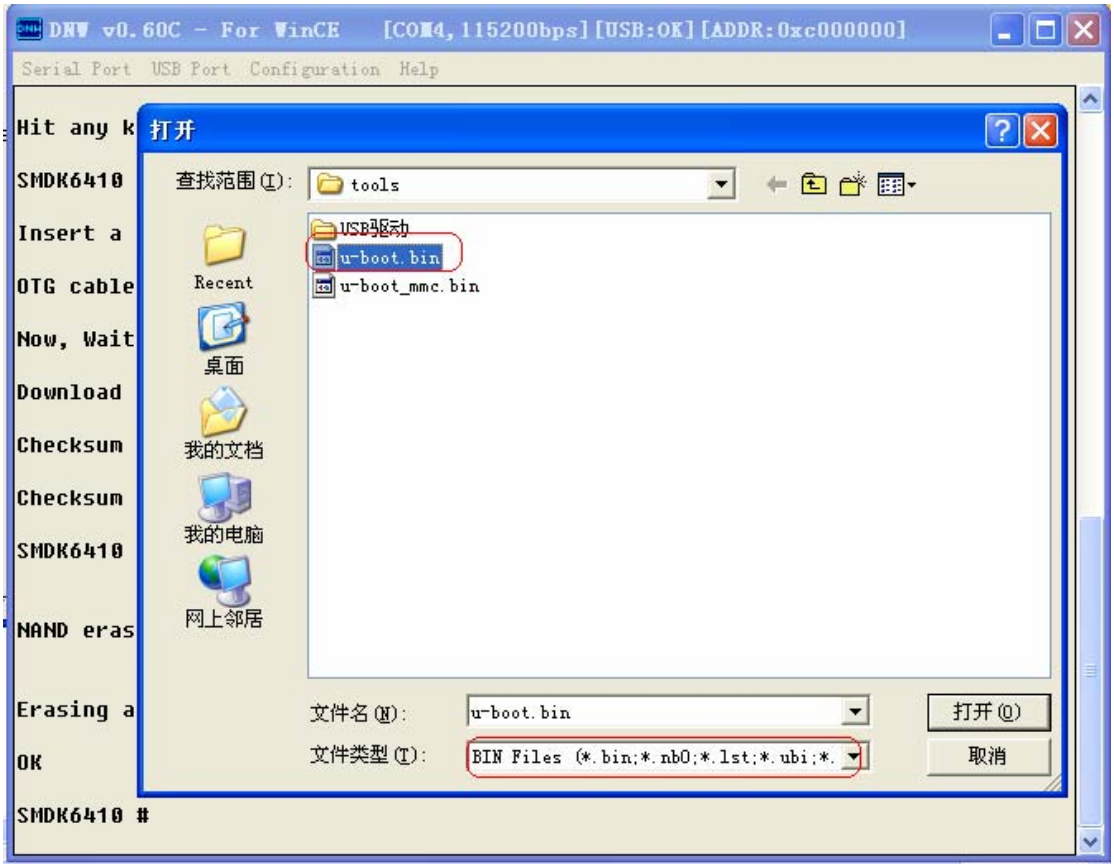
完成安装后在硬件设备管理器中看到已经安装的 USB 设备驱动：

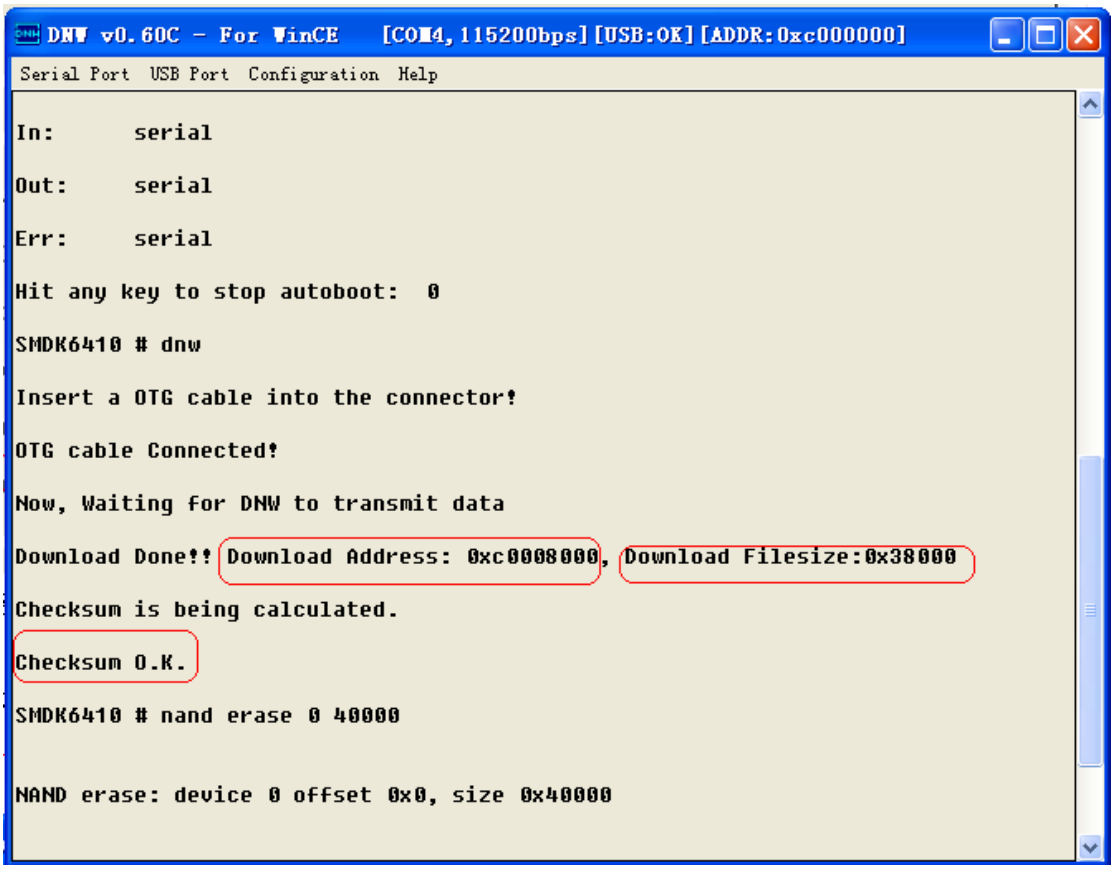


到此，驱动安装成功，这时候使用 usb 线连接开发的小 USB 口到主机，并打开 dnw 下载 uboot 的可执行文件：



选择光盘中 tools 目录下的 u-boot.bin。注意，这个才是 nand 启动的 uboot，不要选择 SD 启动的 u-boot_mmc.bin。



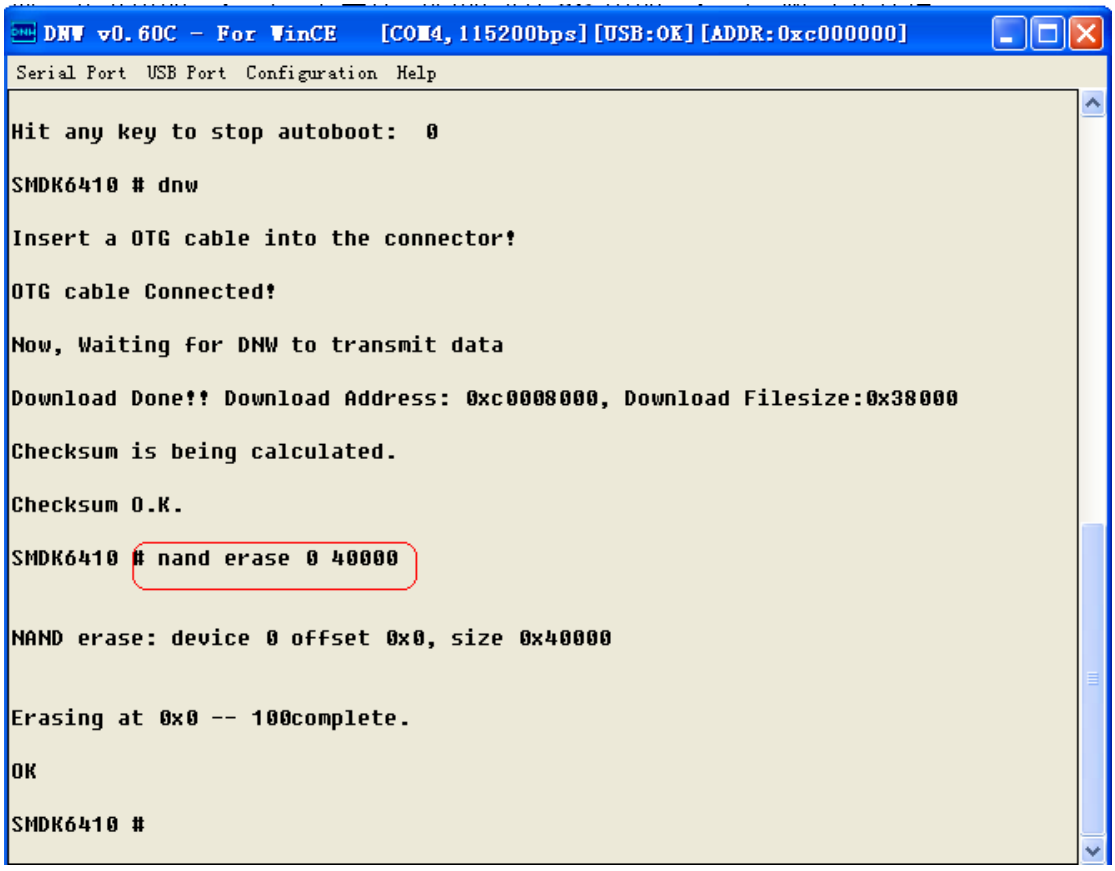


到这里 uboot 已经被下载到板子内存 0xC0008000 的位置中。

第二步是将已经下载到内存中的 uboot 固化到 nandflash

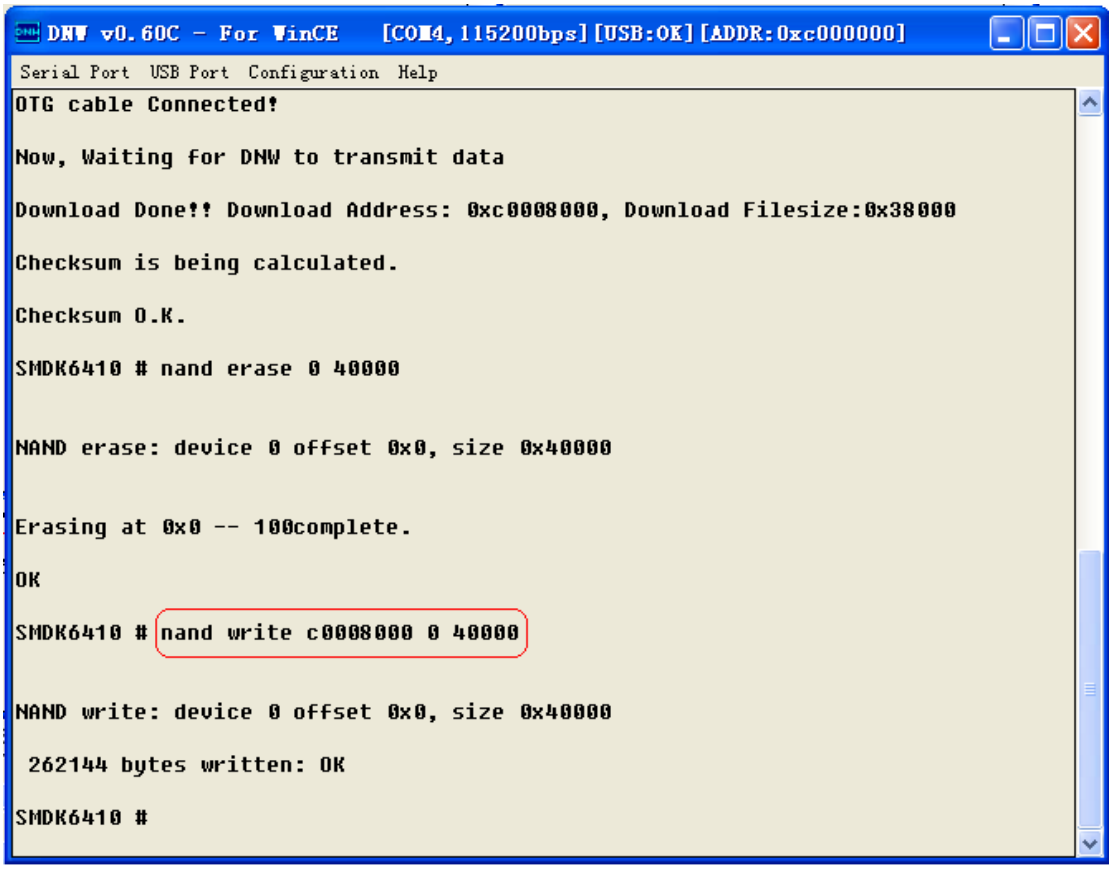
Uboot 要烧写到 nand 中 0 开始的 256KB 中，在烧写之前需要先把这块区域擦除：

```
nand erase 0 40000
```



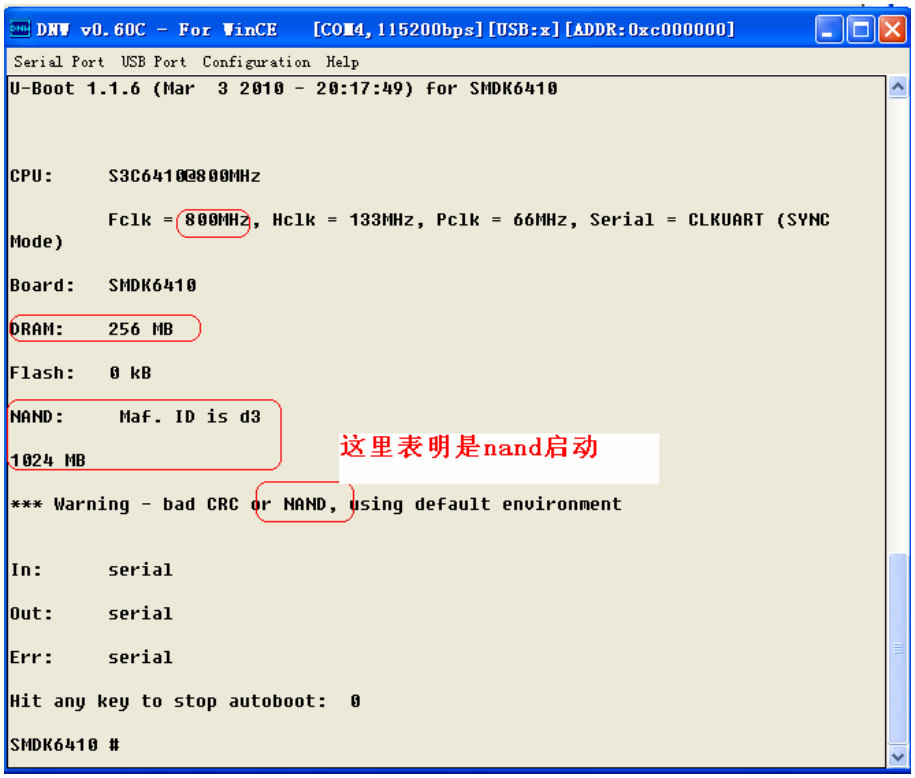
接着将 0xc0008000 中的 uboot 烧写到 nand 中 0 开始的 256K：

```
nand write c0008000 0 40000
```



到这里已经把 uboot 固化到 nand 中，重新设置板子的拨码开关，使启动方式修改为 nand 启动，设置方式参考前面内容。

设置完后重启板子，如果没有意外，将输出以下信息：

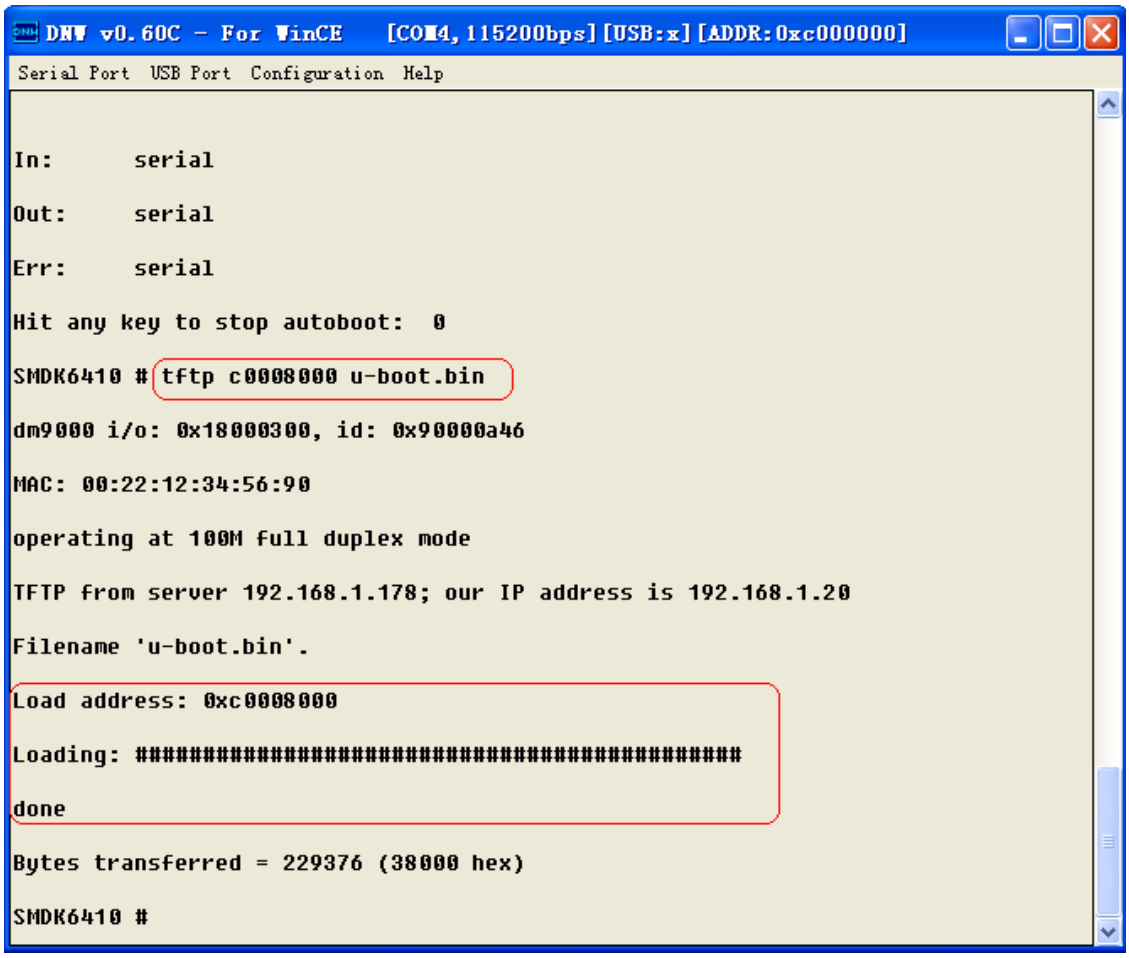


上面的过程是使用 USB 将 uboot 下载到板子内存的方式，这里再演示一下 SD 启动的 uboot 通过 tftp 方式将 nand 启动的 uboot 可执行文件下载到板子内存的方式。

第一步要将 nand 启动的 u-boot.bin 放到 Ubuntu 下的 tftp 共享目录/tftpboot/下。

第二步在 dnw 使用 tftp 下载命令将 u-boot.bin 下载到内存中地址为 0xc0008000 的地方：

```
tftp c0008000 u-boot.bin
```



第三步和 USB 下载方式一样将要放 u-boot.bin 的 nand 前 256K 擦除：

```
nand erase 0 40000
```

第四步将 0xc0008000 中的 uboot 烧写到 nand 中 0 开始的 256K：

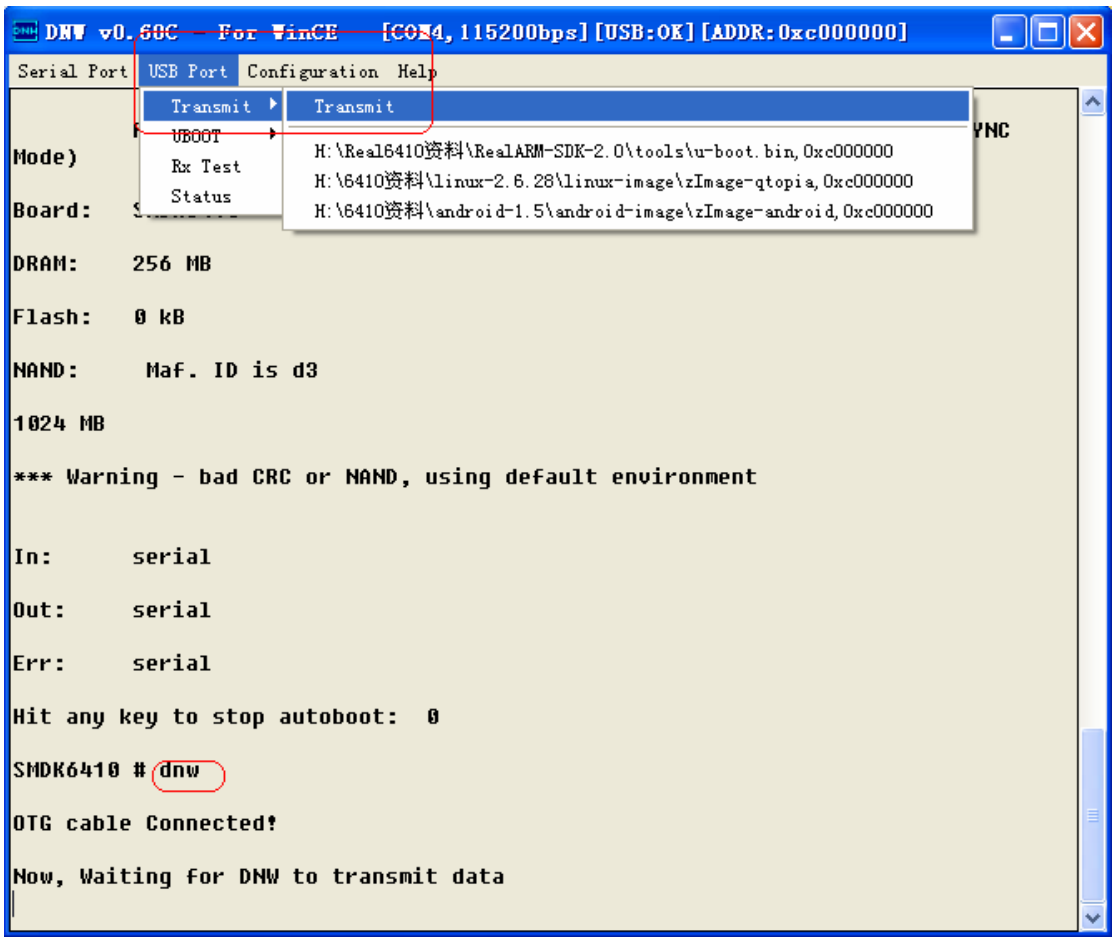
```
nand write c0008000 0 40000
```

这样就完成了 tftp 下载方式的 uboot 烧写。USB 下载和网络下载各有各的优点，USB 的特点是速度快，不需要复杂的设置，但是有时候会导致系统的崩溃（机率很小）。Tftp 下载速度也很快，但是设置比较复杂，初学者容易被搞晕，但是 tftp 工作却十分稳定。用户可以根据自己的习惯和实际情况选择。

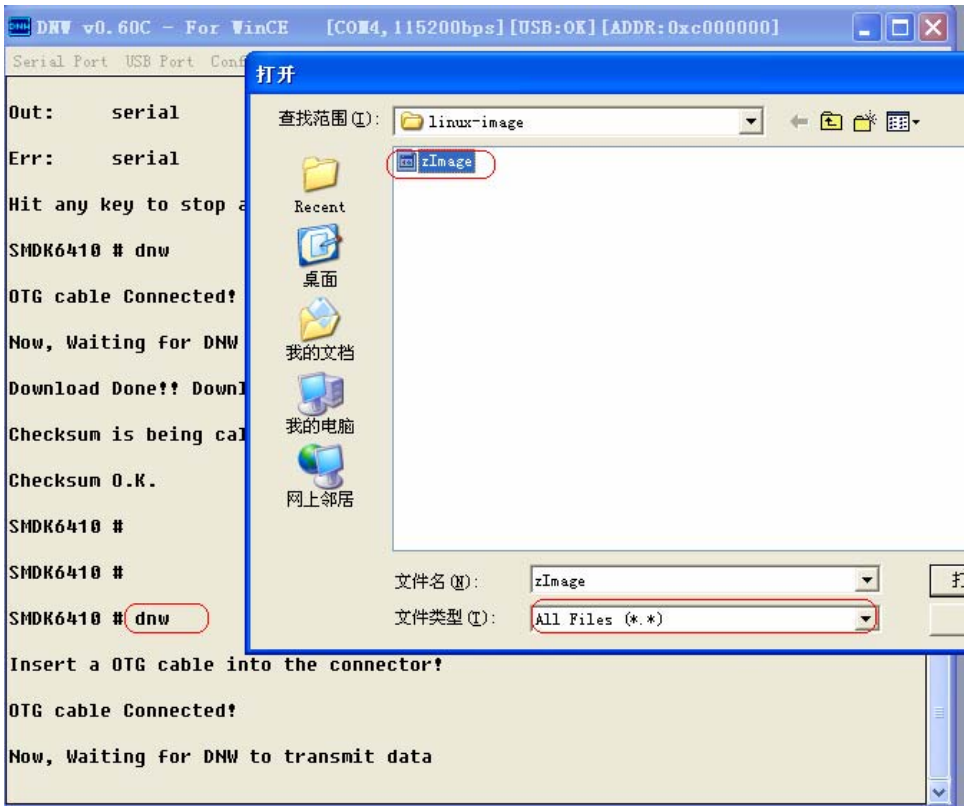
5.2 Linux 内核的烧写

第一步 通过 USB 将内核文件下载到板子内存中：

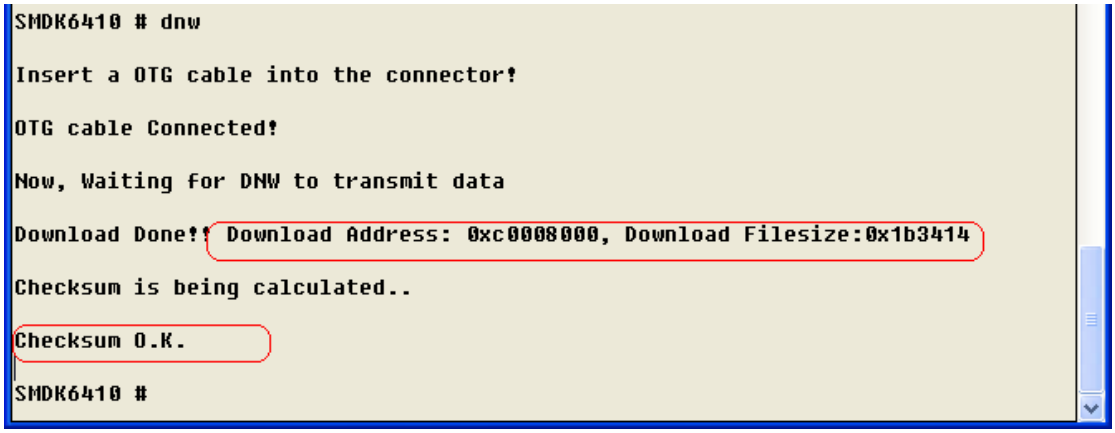
```
dnw
```



打开光盘中 linux\linux-image 目录下的 zImage，将内核镜像下载到板子：

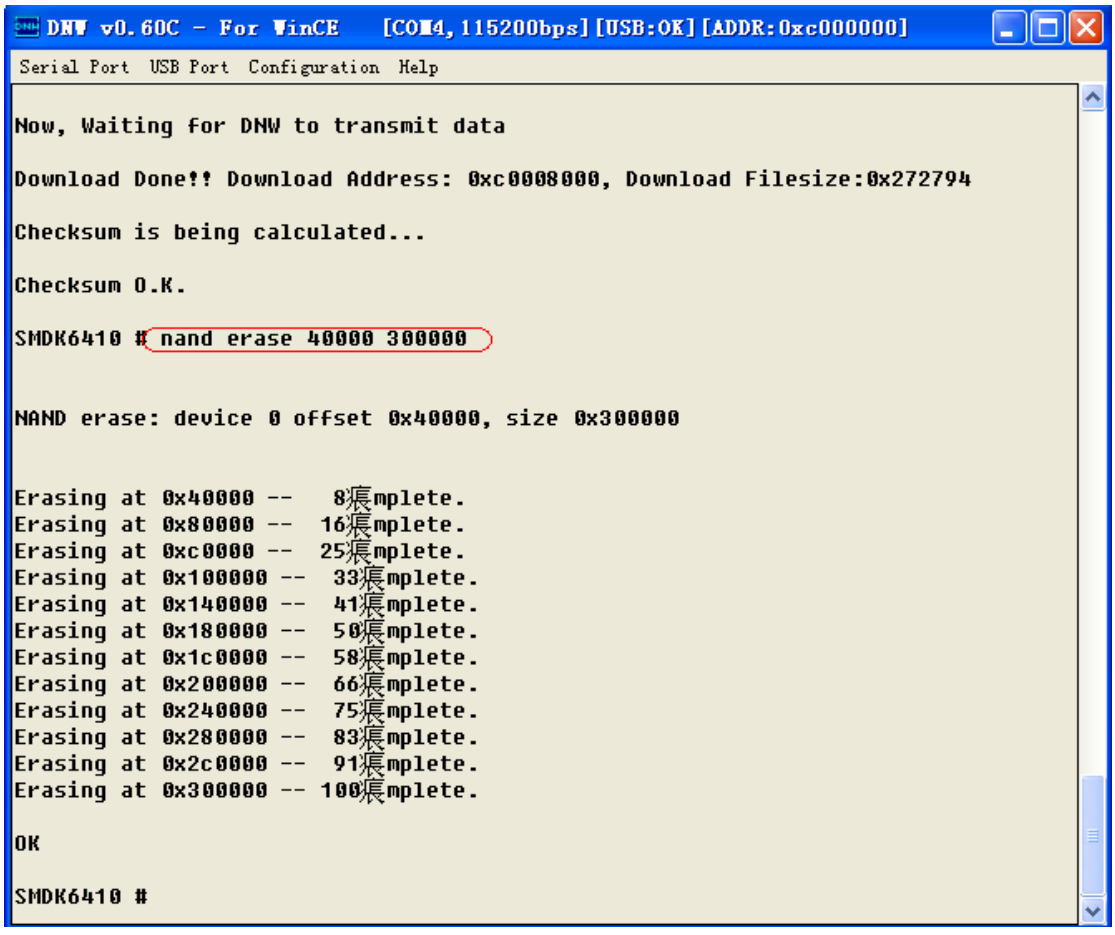


下载结束后：



第二步 擦除 nandflash 中存放内核的那部分空间：

```
nand erase 40000 300000
```



第三步 将内存中的内核固化到 Nandflash

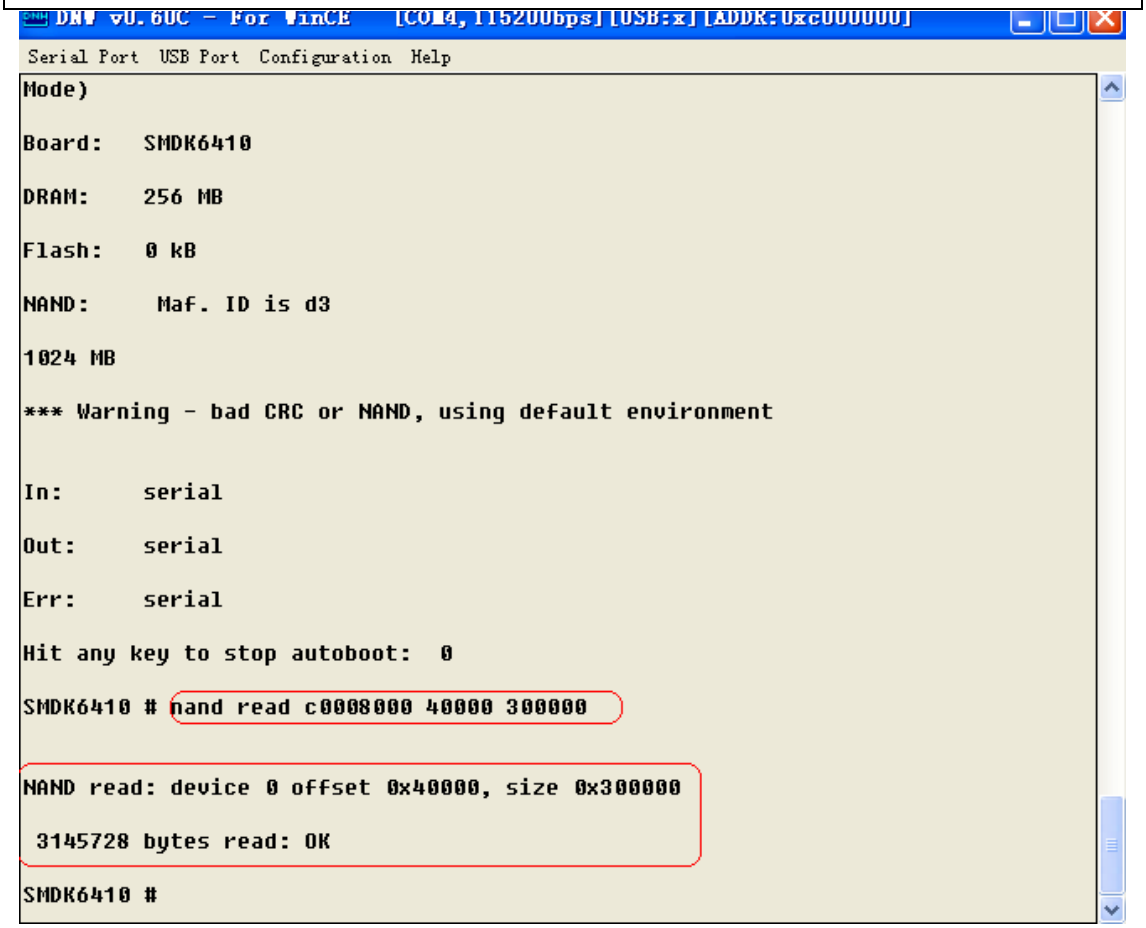
```
nand write c0008000 40000 300000
```

```
OK

SMDK6410 # nand write c0008000 40000 300000

NAND write: device 0 offset 0x40000, size 0x300000
 3145728 bytes written: OK

SMDK6410 #
```

第四步 测试内核是否能启动

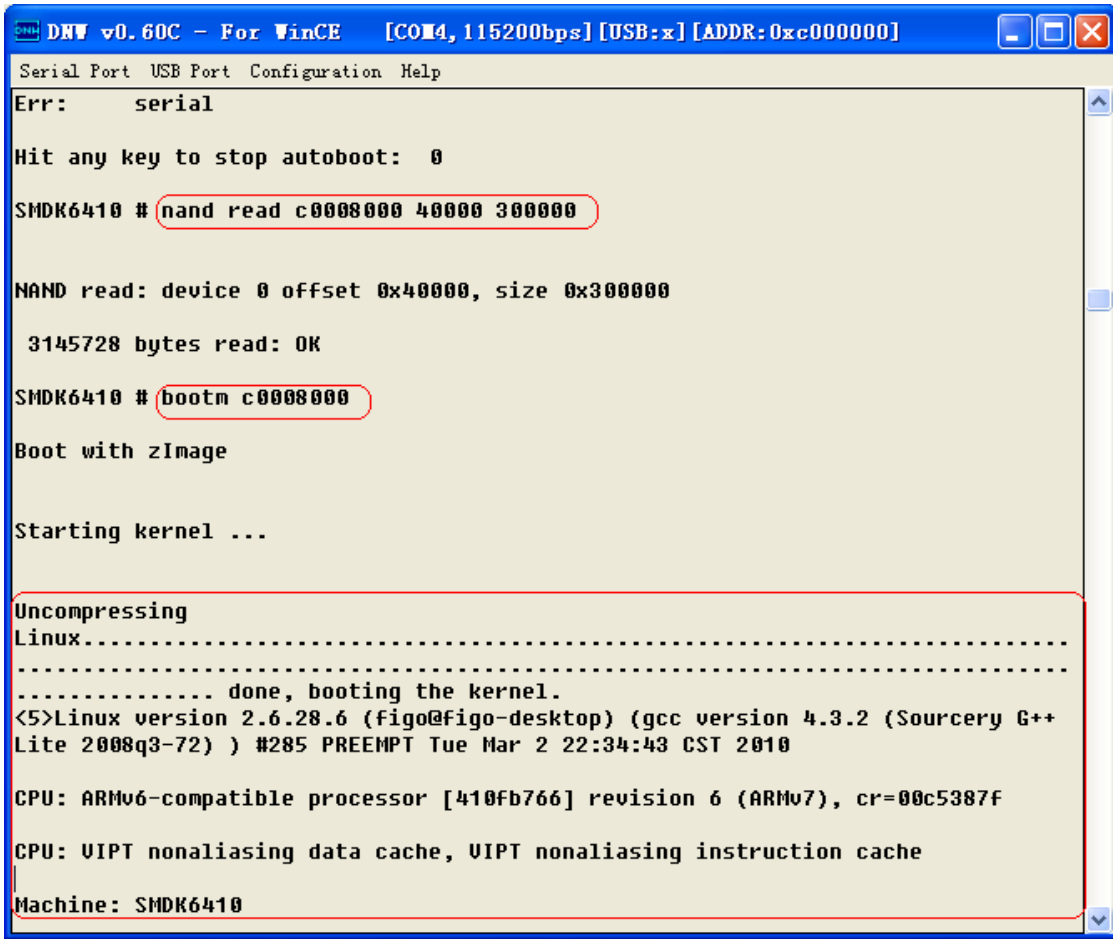
复位板子或者断电重启，然后将内核从 nand 中复制到内存 0xC0008000 的地方：

```
nand read c0008000 40000 300000
```



接着跳启动内存中的内核：

```
bootm c0008000
```



看到这些信息表明内核烧写成功了。

5.3 cramfs 根文件系统的烧写

Android 或者带有带有 Qtopia 的根文件系统可以使用 yaffs 或者 ubifs 等可读写的文件系统，不过 yaffs 在我们的试验过程中并不十分稳定，即使使用了 Google 修补过以后的 yaffs 源码仍然会偶尔导致系统崩溃，因此我们采用了性能更好的 ubifs，该文件系统由 IBM 等公司联合开发，用于嵌入式系统中，并已经被 2.6.27 以后的 linux 内核所采用。

但是目前版本的 uboot 的 nand 驱动对可读写的 ubifs/yaffs2 支持不完善，可以利用 cramfs 文件系统来间接实现 ubifs/yaffs2 的制作，所以这里首先介绍如何烧写一个 cramfs 到板子中。这个文件系统建议在系统调试完成之前一直保留而不要删除，这样可以在需要更新 ubifs/yaffs2 文件系统的时候随时能使用。

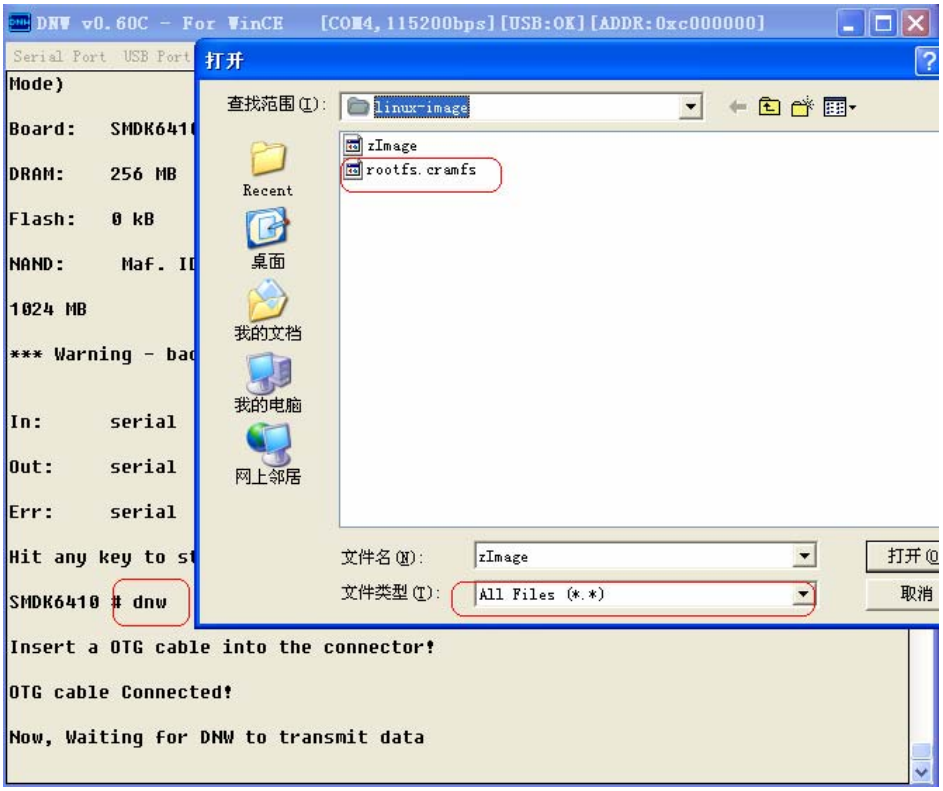
为了避免读者产生混乱这里再次强调，真正用到的文件系统都是使用 ubifs 格式的根文件系统，cramfs 文件系统只是为了烧写而制作的辅助文件系统。在后续我们将对 uboot 进行修正，让 uboot 能够直接支持 ubifs。

注意：如果在 uboot 下直接烧写 ubifs/yaffs2 的镜像文件，可能造成大量 nand 坏块，因此请谨慎做这样的操作。

光盘中的 linux\linux-image\rootfs.cramfs 是我们为了实现 ubifs/yaffs2 格式烧写而制作的最小文件系统，将这个文件系统下载到板子，并烧写到板子 Nandflash 的 cramfs 分区。烧写步骤如下：

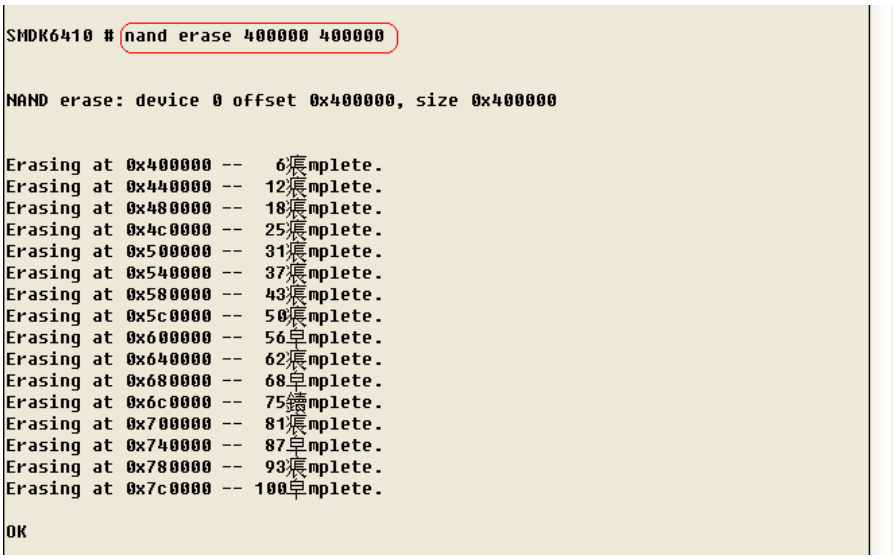
第一步 使用 USB 将 cramfs 下载到板子内存中：

```
dnw
```



第二步 删除 nand 中的 cramfs 分区：

```
nand erase 400000 400000
```



第三步 把内存中的根文件系统烧写到 Nandflash 中的 cramfs 分区：

```
nand write c0008000 400000 400000
```

```
SMDK6410 # nand write c0008000 400000 400000

NAND write: device 0 offset 0x400000, size 0x400000
 4194304 bytes written: OK
SMDK6410 #
```

第四步 测试 cramfs 是否烧写成功：

在启动内核之前先要设置启动参数以便让内核去挂载 cramfs：

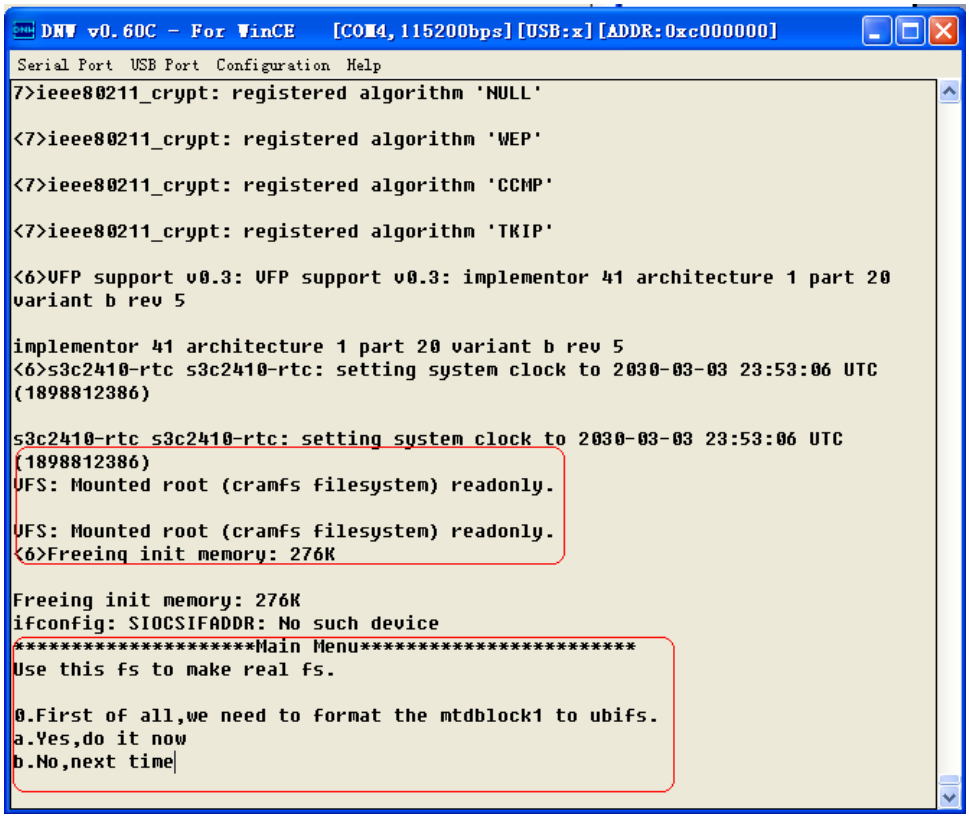
setenv bootargs noinitrd root=/dev/mtdblock0 console=ttySAC0 init=/linuxrc

saveenv（保存参数）

```
SMDK6410 # setenv bootargs noinitrd root=/dev/mtdblock0 console=ttySAC0
init=/linuxrc

SMDK6410 # saveenv

Saving Environment to NAND...

Erasing Nand...Writing to Nand... done

SMDK6410 #
```

如果内核已经按照上一步来烧写，那么重启板子后就应该能够内核启动成功，并且成功挂载到 cramfs：

```
DNW v0.60C - For WinCE    [COM4,115200bps][USB:x][ADDR:0xc000000]
Serial Port  USB Port  Configuration  Help
Flash:   0 kB

NAND:     Maf. ID is d3

1024 MB

In:      serial

Out:     serial

Err:     serial

Hit any key to stop autoboot:  0

NAND read: device 0 offset 0x40000, size 0x300000
 3145728 bytes read: OK

Boot with zImage


Starting kernel ...


Uncompressing
Linux.....................................................................
.....................................................................
.............. done, booting the kernel.
<5>Linux version 2.6.28.6 (figo@figo-desktop) (gcc version 4.3.2 (Sourcery G++
Lite 2008q3-72) ) #285 PREEMPT Tue Mar 2 22:34:43 CST 2010
```



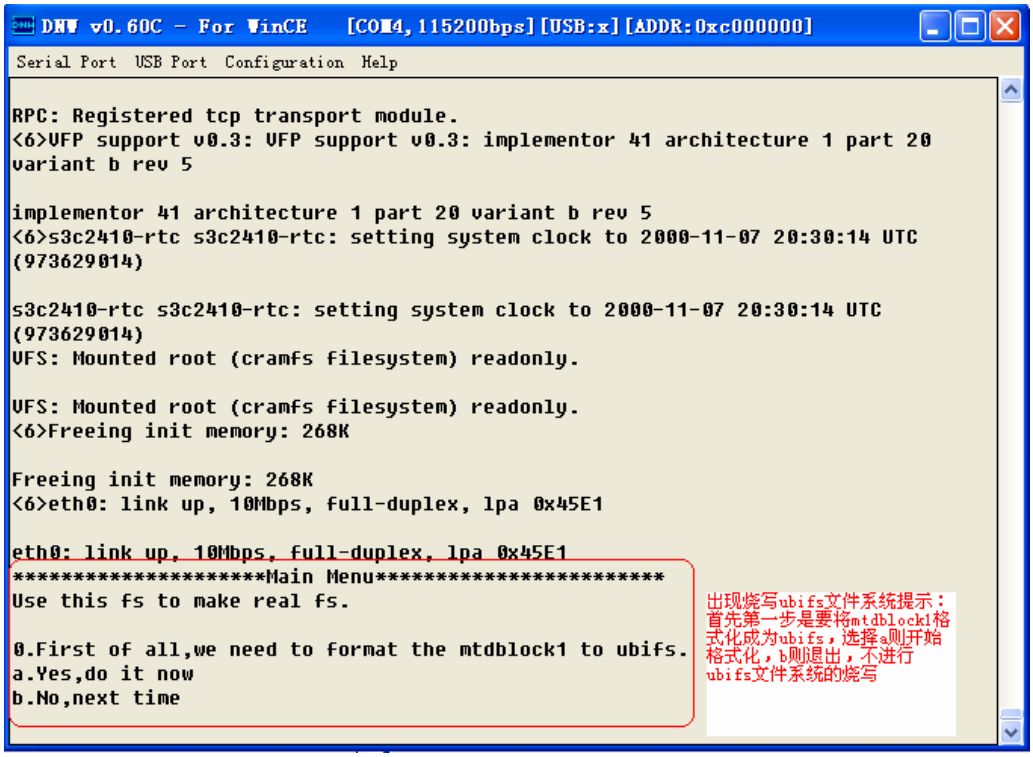
到这里，内核已经成功启动，并已经挂载到 cramfs。

5.4 ubifs 根文件系统的烧写

上一步烧写的 cramfs 只是为了进行 ubifs 烧写而制作的，在修正了 uboot 对 ubifs 的支持后将会不再采用这种方式。下面开始介绍基于 cramfs 的 ubifs 根文件系统烧写方式。

按照之前介绍的步骤烧写完 cramfs 之后便可以开始 ubifs 根文件系统的烧写了。

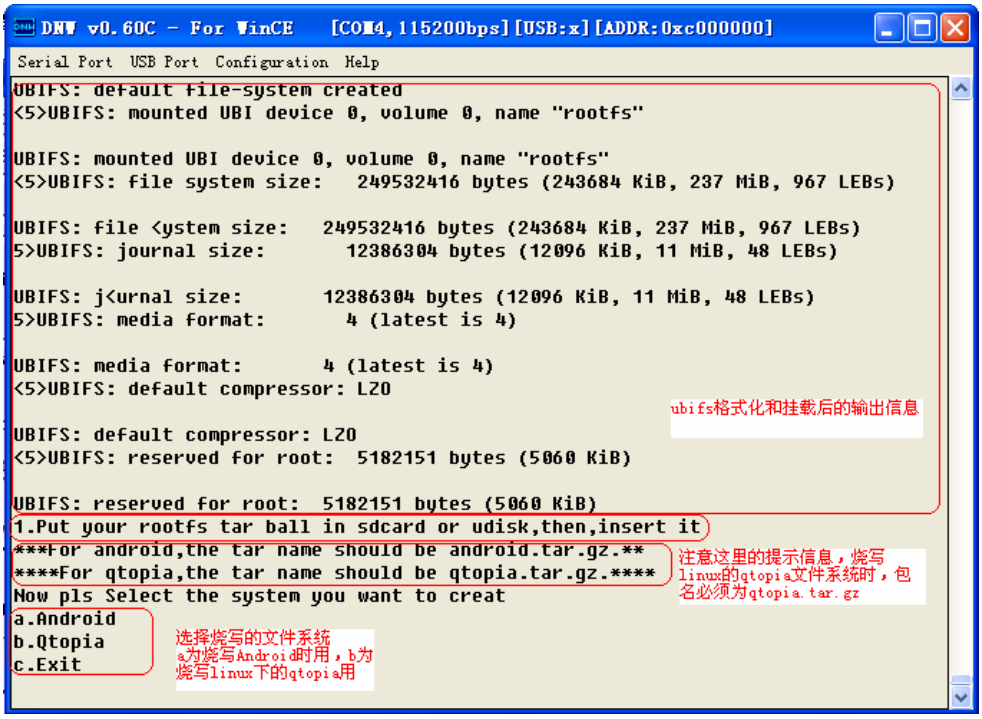
第一步是将已经制作好的根文件系统 linux\linux-image\qtopia.tar.gz 复制到 U 盘或者 SD 卡中，接着把 U 盘或者 SD 卡插到板子上，给板子上电，启动内核并挂载 cranfs 后会出现下面的选择提示：



这里首先提示用户输入命令，是否现在就格式化 Nandflash 的第二个分区。内核中对 Nandflash 设置了 2 个分区，一个是 cramfs 格式，大小为 4MB，就是 mtdblock0。另一个是 ubifs 格式，大小为（1G-8M）B，就是上图中说的 mtdblock1。

注意：bootloader、kernel 烧写在 Nandflash 中的那部分空间并没有在内核中划作相应的分区，这是为了保护这 2 个分区不被误操作。

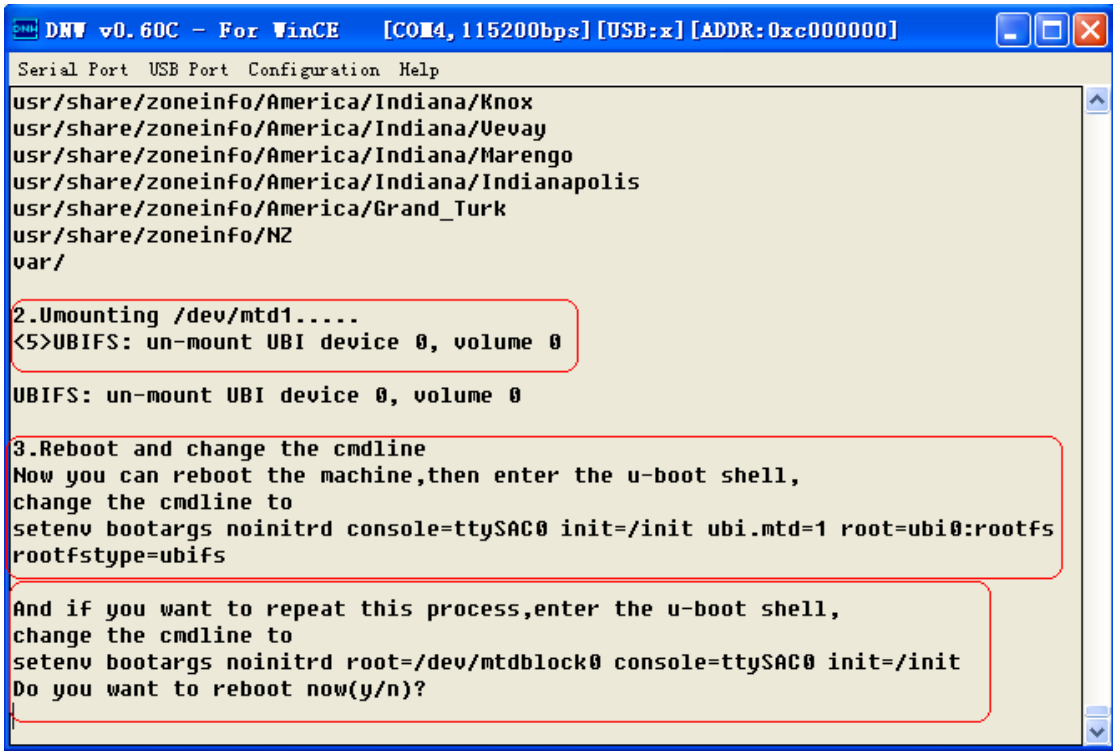
在往 mtdblock1 烧写 ubifs 之前先要对这个分区进行格式化为 ubifs。所以如果现在就想开始烧写 ubifs，那么这里输入 a 就开始对 ubifs 分区进行格式化了：



这时候进入第二步选择，这里可以选择是烧入含有 qtopia 的根文件系统还是烧写

Android 的根文件系统，从这里也可以得知，烧写 Android 根文件系统只是这一步的选择不一样而已。

这里选择烧写 Qtopia，选择 b，结果如下：

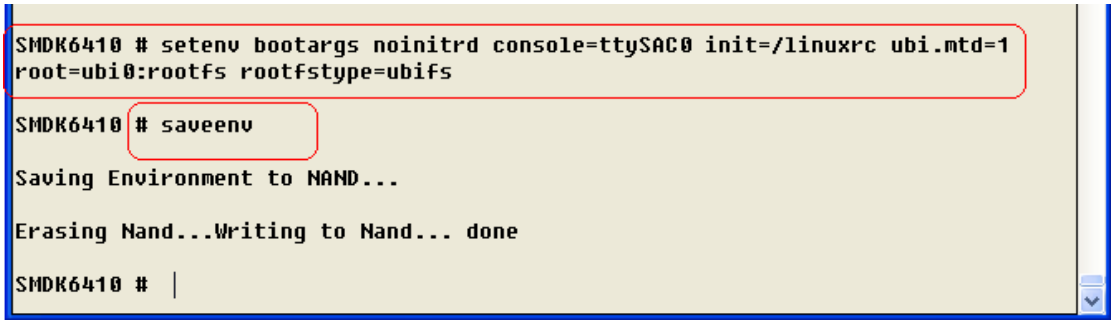


到这里 ubifs 烧写完成，接下来测试一下这个文件系统是否可用。

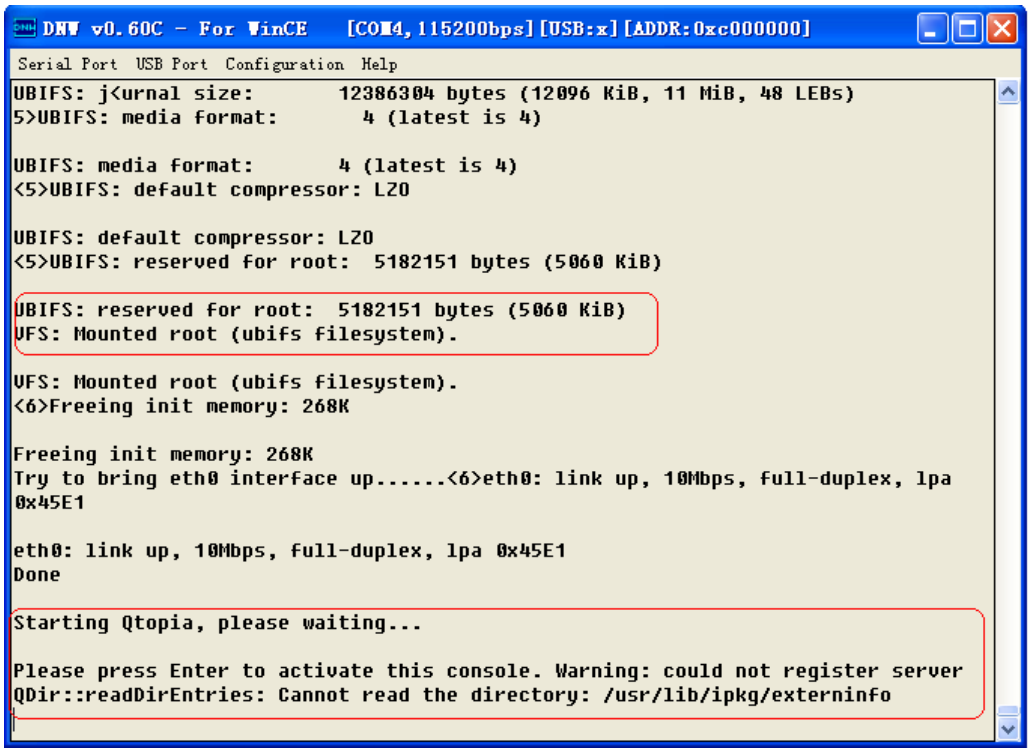
首先复位系统重新进入 uboot 的命令行界面，修改一下内核的启动参数，让内核不再挂载 cramfs 格式的 mtdblock0，转而挂载 ubifs 格式的 mtdblock1：

```
setenv bootargs noinitrd console=ttySAC0 init=/linuxrc ubi.mtd=1 root=ubi0:rootfs rootfstype=ubifs
```

然后使用 saveenv 命令保存一下这些参数，



重启板子让内核启动并挂载 ubifs 文件系统，如果没有问题那么将在 dnw 看到下面的输出，并在 LCD 中看到 qtopia 的启动界面：



这时系统便成功启动了，点击触摸屏即可操作 GUI 界面。

注意：如果触摸屏不准可以在进入 linux 后使用下面的 2 条命令进行校准：

rm /etc/pointercal
reboot

这样等待校准提示的出现后点击 5 点即可。

第六章 Uboot 的使用教程

6.1 uboot 配置与编译

6.1.1 uboot 的配置

将光盘中的 linux\linux-source\s3c-u-boot-1.1.6-Real6410.tar.bz2 复制到 Ubuntu 的主目录下，打开一个终端输入下面命令进行解压：

```
tar –jxvf s3c-u-boot-1.1.6-Real6410.tar.bz2
```

这样便得到了 s3c-u-boot-1.1.6-Real6410 源码目录。在进行新的配置和编译之前可以通过下面命令进行清除：

```
make distclean
```

接着使用下面命令进行配置：

```
make smdk6410_config
```

6.1.2 uboot 的编译

配置结束后就可以准备编译了。不过 s3c-u-boot-1.1.6-Real6410 支持 SD 启动和 Nandflash 启动这两种方式，针对不同的启动方式，为了简化用户的工作，可以直接运行我们提供的 2 个脚本：

```
make_nand_image   → 编译 nand 启动 uboot 脚本
make_mmc_image    → 编译 SD 启动 uboot 脚本
```

这 2 个脚本就放在 s3c-u-boot-1.1.6-Real6410 目录下，直接运行即可。

如果要编译 Nandflash 启动的 uboot，在终端中进入 uboot 源码目录后输入下面指令：

```
./make_nand_image
```

如果没有意外，将在 s3c-u-boot-1.1.6-Real6410 目录下产生 u-boot.bin，把这个 u-boot.bin 烧写到 Nandflash 中即可。

如果要编译 SD 启动的 uboot，在终端中进入 uboot 源码目录后输入下面指令：

```
./make_mmc_image
```

如果没有意外，将在 s3c-u-boot-1.1.6-Real6410 目录下产生 u-boot_mmc.bin，把这个 u-boot_mmc.bin 烧写到 SD 中并插入板子即可启。

6.2 uboot 命令使用

6.2.1 网络命令的使用

在程序调试过程中，常常要在 uboot 下把主机中的某个文件下载到板子内存中，最常用的下载方式就是使用 uboot 的网络下载命令了。下面介绍用的最多的几条下载命令。

- 设置板子 ip

如果板子预设的 ip 地址（192.168.1.20）不满足你的要求，可以通过下面指令配置板子 ip 地址：

```
setenv ipaddr 192.168.1.20
```

- 设置服务器 ip

在使用 tftp 下载的时候，默认是从服务器中下载的，这个默认服务器 ip 地址是（192.168.1.178），如果你的服务器地址不是这个，可以通过下面指令进行修改：

```
setenv serverip 192.168.1.178
```

要想让这个环境变量的设置在板子重启后仍然生效，需要使用到环境变量保存命令：

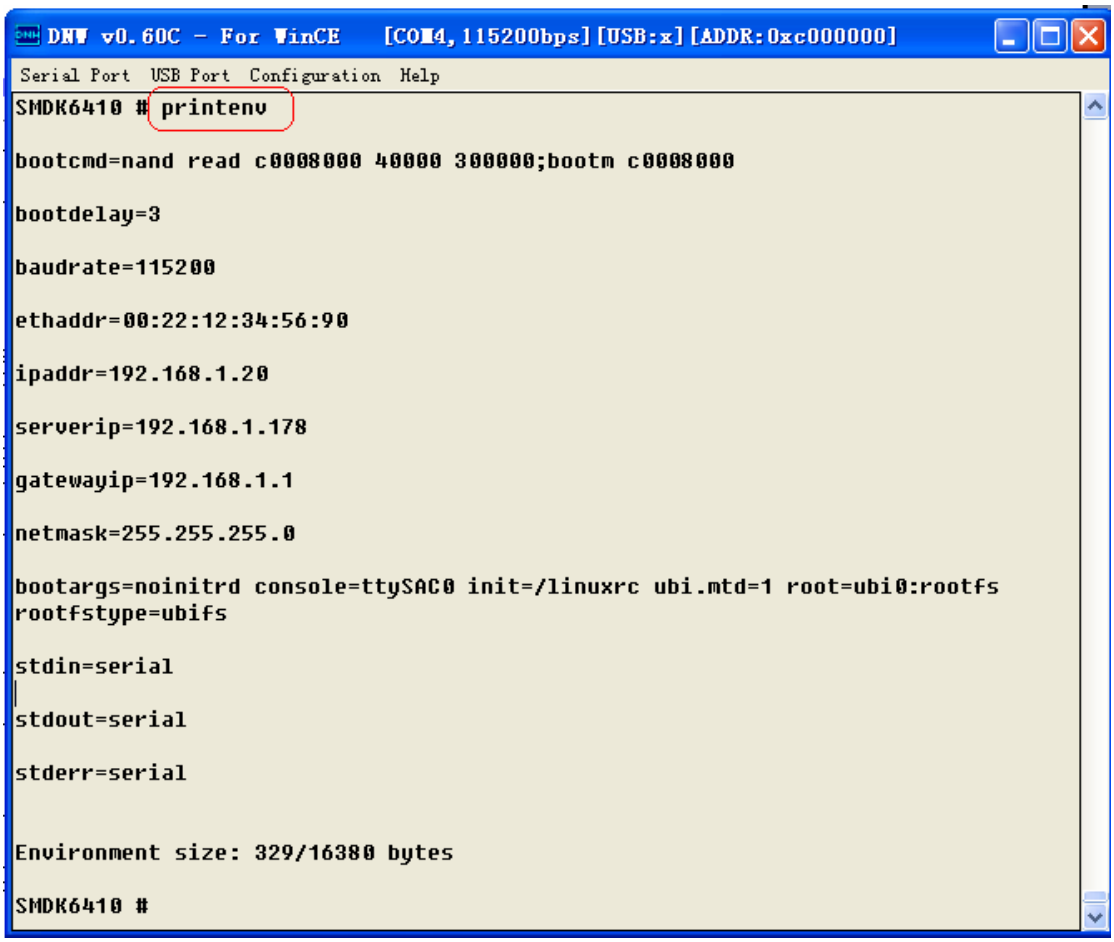
```
saveenv
```

- 打印环境变量

显示环境变量命令：

```
printenv
```

这样便显示出 uboot 中的环境变量设置情况：



- Tftp 命令

  使用 Tftp 下载命令可以从主机的 tftp 共享目录下载特定文件到板子内存：

```
tftp c0008000 zImage
```

其中 c0008000 是下载到内存中的地址，zImage 是下载的文件名，默认的 tftp 服务器地址为服务器 ip。

- Ping 命令

  在使用网络的时候如果网络不通往往要使用 ping 命令测试网络的连通性：

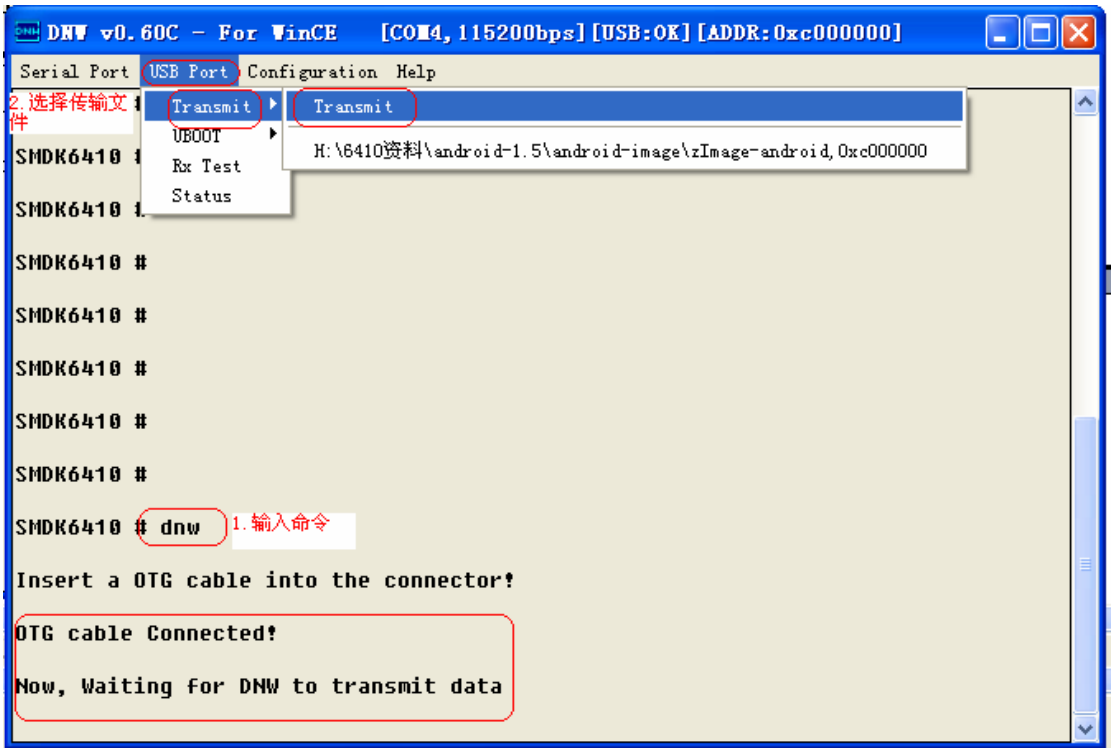
```
ping 192.168.1.1
```

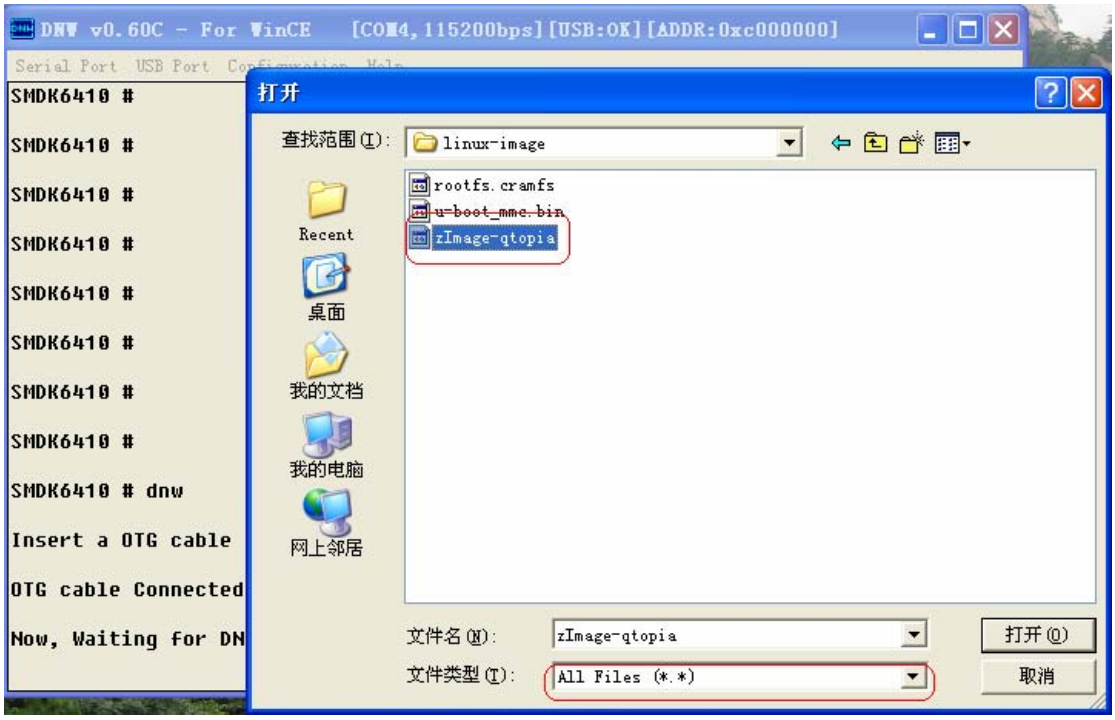
6.2.2 USB 命令的使用

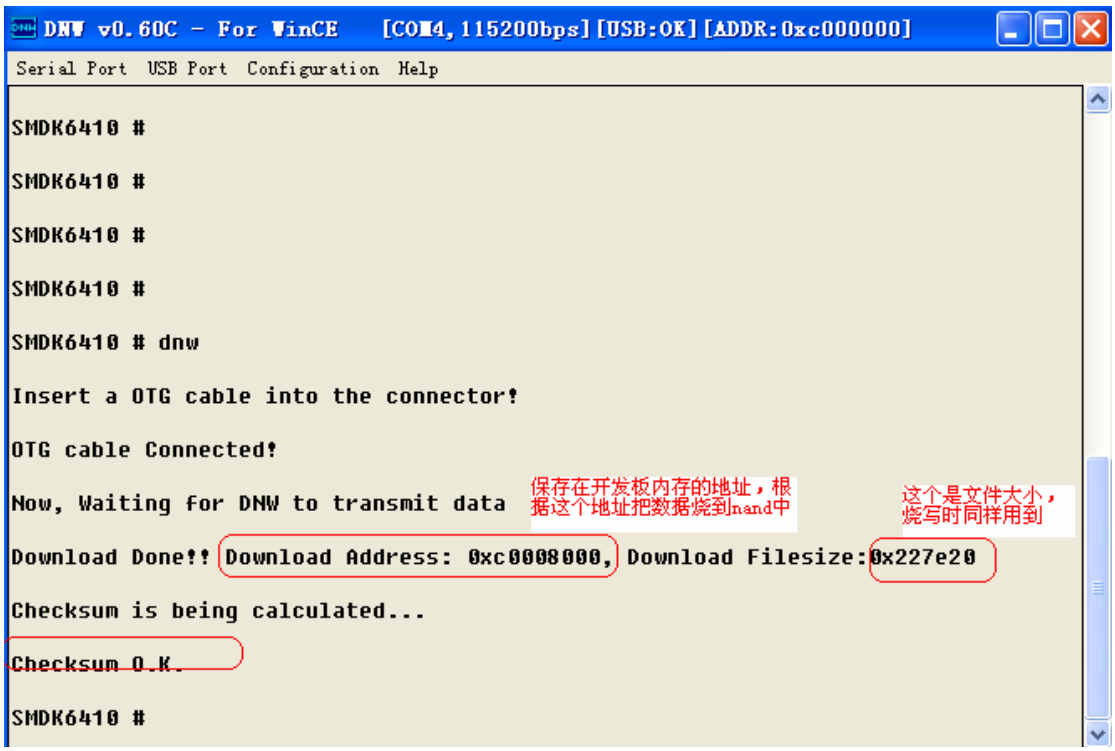
Uboot 中提供了使用 USB 线从主机下载文件的命令，用法非常简单：

```
dnw
```

这时用 USB 线将开发板的小 USB 口连接到到主机，打开 dnw 下载文件：


DNW v0.60C - For WinCE [COM4,115200bps][USB:OK][ADDR:0xc000000]
Serial Port
USB Port
Configuration
Help
2.选择传输文件
Transmit
UBOOT
Rx Test
Status
Transmit
H:\6410资料\android-1.5\android-image\zImage-android,0xc000000
SMDK6410 #
SMDK6410 # dnw
1.输入命令
Insert a OTG cable into the connector!
OTG cable Connected!
Now, Waiting for DNW to transmit data


DNW v0.60C - For WinCE [COM4,115200bps][USB:OK][ADDR:0xc000000]
SMDK6410 # dnw
Insert a OTG cable
OTG cable Connected
Now, Waiting for DN
打开
查找范围(I):
linux-image
rootfs.cramfs
u-boot_mmc.bin
zImage-qtopia
Recent
桌面
我的文档
我的电脑
网上邻居
文件名(N):
zImage-qtopia
文件类型(T):
All Files (*.*)
打开(O)
取消



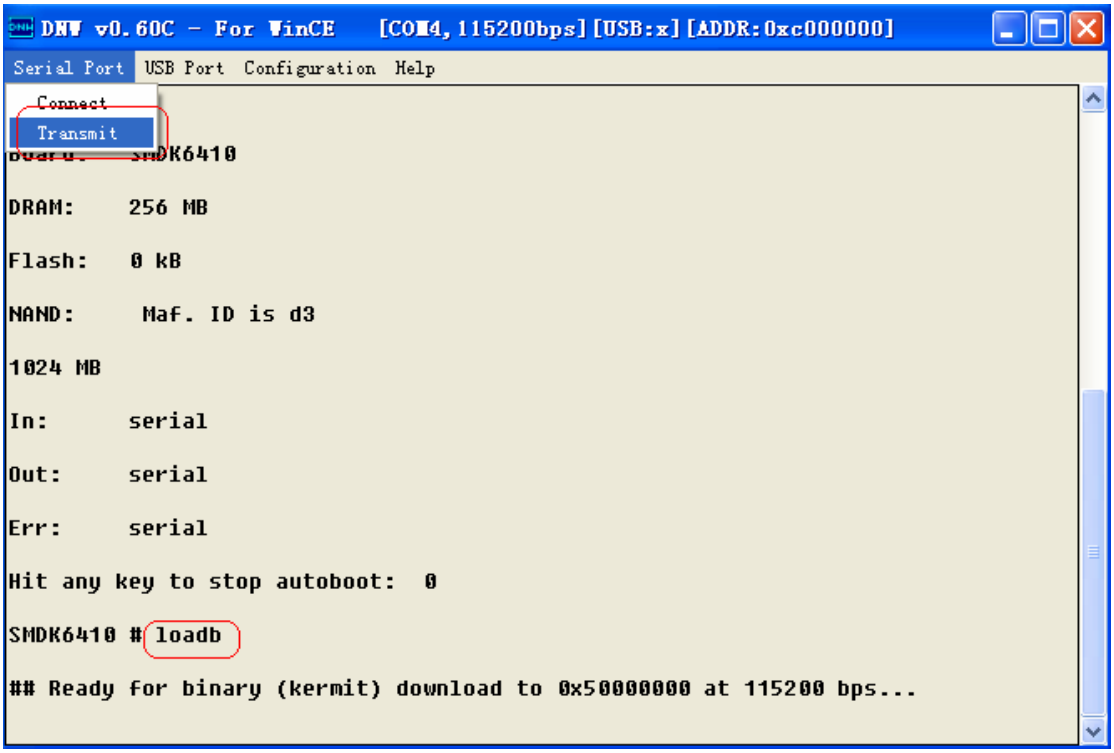
默认情况下文件数据保存在内存地址为 0xc0008000 的地方。结合 nand flash 命令操作就能过实现通过 USB 烧写代码。

6.2.3 串口下载命令

在网络驱动和 usb 驱动不可用的情况下，可以使用串口来从主机下载文件，命令：

```
loadb
```

接着从串口终端通过串口的发送菜单进行文件的传输，如下图所示：



6.2.4 Nandflash 操作命令

通过 nand flash 操作命令能够更新板子中 Nandflash 的内容。

- Nand 擦除命令

Nand 在写入任何数据之前需要擦除，命令为：

```
nand erase 0 40000
```

表示从 nandflash 的 0 地址开始擦除，共擦除 0x40000 个字节。

- Nand 的写命令：

```
nand write c0008000 0 40000
```

表示将内存地址 0xc0008000 开始的数据写到 nand 中 0 开始的地址中，共写 0x40000 个字节。

- Nand 的读命令:

```
nand read c0008000 0 40000
```

表示将 nand 中 0 开始的数据复制到内存地址为 0xc0008000 开始的地方，共复制 0x40000 个字节。

6.2.5.启动 linux 内核相关命令

- 内核启动参数设置命令

在启动内核之前需要设置启动参数，可以使用 setenv bootargs 命令完成。比如下面的启动参数是使用 nfs 方式挂载文件系统的内核启动参数：

```
setenv bootargs noinitrd root=/dev/nfs console=ttySAC0 init=/linuxrc
nfsroot=192.168.1.178:/nfsboot
ip=192.168.1.20:192.168.1.178:192.168.1.1:255.255.255.0::eth0:on
```

这是以 nfs 作为根文件系统的启动参数，其中 192.168.1.20 是板子 ip，192.168.1.178 是服务器 ip，192.168.1.1 是网关 ip，255.255.255.0 是掩码。可以根据自己网络的实际情况更换合适的 ip 和 nfs 目录。

使用 nfs 作为根文件系统对于程序的调试是非常有意义的，这样板子的根文件系统可以放在开发主机上（Ubuntu），不必每次文件系统更新后都要重新烧写 flash。

如果程序调试已经结束，需要把文件系统烧写到板子的 Nandflash 中，并让内核到 Nandflash 中挂载文件系统，那么需要使用下面的启动参数了：

```
setenv bootargs noinitrd root=/dev/mtdblock0 console=ttySAC0 init=/linuxrc
```

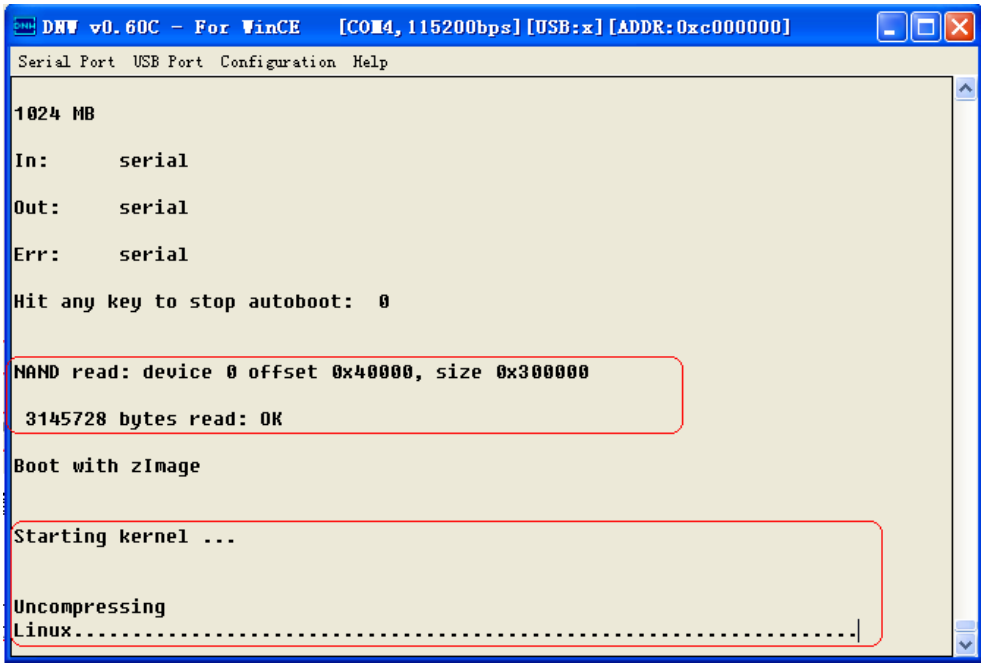
这是以 nand 中的 mtdblock0 分区作为根文件系统的启动参数，这里默认情况下放的是 cramfs 格式的文件系统。

如果要使用板子中的 ubifs 分区，也就是 mtdblock1 分区作为根文件系统，那么需要使用下面的启动参数：

```
setenv bootargs noinitrd console=ttySAC0 init=/linuxrc ubi.mtd=1 root=ubi0:rootfs
rootfstype=ubifs
```

- Uboot 自动运行命令

Uboot 在板子复位后，如果用户没有在命令界面中按下某个按键，那么 uboot 会自动运行某些命令。比如下图中，如果用户在倒计时结束之前没有按下按键那么 uboot 自动从 Nandflash 中复制数据到内存，并跳转到内存中启动内核：



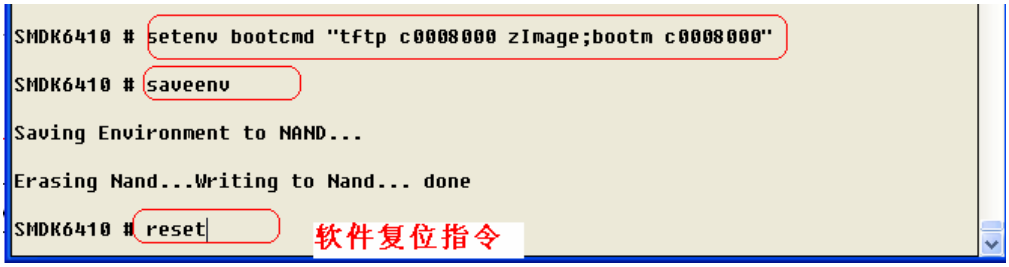
对于这个复位后自动运行的动作实际上是可以自定义的。比如说，要实现这样的动作：

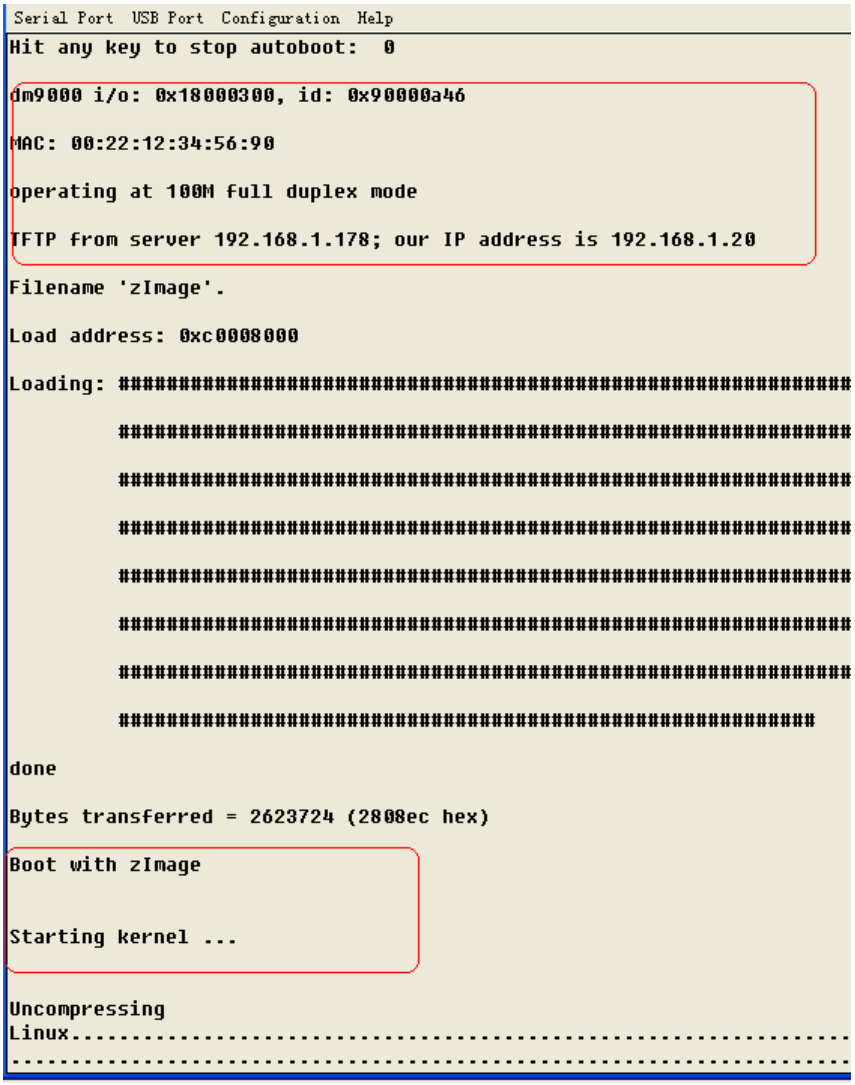
让 uboot 在复位后自动通过 tftp 从主机下载内核文件 zImage 到内存中并启动内核的，那么可以这样设置：

```
setenv bootcmd "tftp c0008000 zImage;bootm c0008000"
```

个人比较喜欢设置为这个，因为这样的话在内核改动后不需要重新烧写到 Nandflash 中，直接放到 Ubuntu 的 tftp 共享目录下，由 uboot 下载到内存中直接启动，不需要经过 nand 烧写。

设置完毕后输入 saveenv 保存一下，重启板子后运行效果如下：





当内核调试完毕，内核需要烧写到 Nandflash 中，这时 uboot 的自动运行动作就变成了从 Nandflash 中读取内核到内存，并跳转到内存启动内核。这样就需要如下设置：

```
setenv bootcmd "nand read c0008000 40000 300000;bootm c0008000"
```

uboot 中还有很多其他有用的命令，这里无法一一列举，只说明了几个常用命令的使用方法，更多命令的使用请参考《附录五 uboot 命令教程》

第七章 Linux 内核使用教程

光盘中提供的 linux 内核源码在 linux\linux-source\s3c-linux-2.6.28-Real6410.tar.bz2，该版本内核是目前功能最完善的。把该压缩包复制到 Ubuntu 主目录下，通过下面命令进行解压：

```
tar –jxvf s3c-linux-2.6.28-Real6410.tar.bz2
```

这样便能在当前目录下得到 s3c-linux-2.6.28-Real6410，为了方便调试，我们还提供了 2 个位于该目录的可执行脚本：

build　　－＞　不执行 make xconfig,直接运行 make 来编译内核，并且在编译结束后把 zImage 复制到/tftpboot 下面供 uboot 下载，用户可以根据需要修改这个脚本的内容

xbuild　　－＞　执行 make xconfig 对内核进行配置后再使用 make 编译内核，最后把编译出来的 zImage 复制到/tftpboot 下

开启一个终端并分别输入下面 2 个命令即可运行这 2 个脚本：

```
/build
/xbuild
```

当然，用户也可以直接运行内核原有的

```
make xconfig
或者
make menuconfig
```

命令打开内核的配置界面，并运行

```
make
```

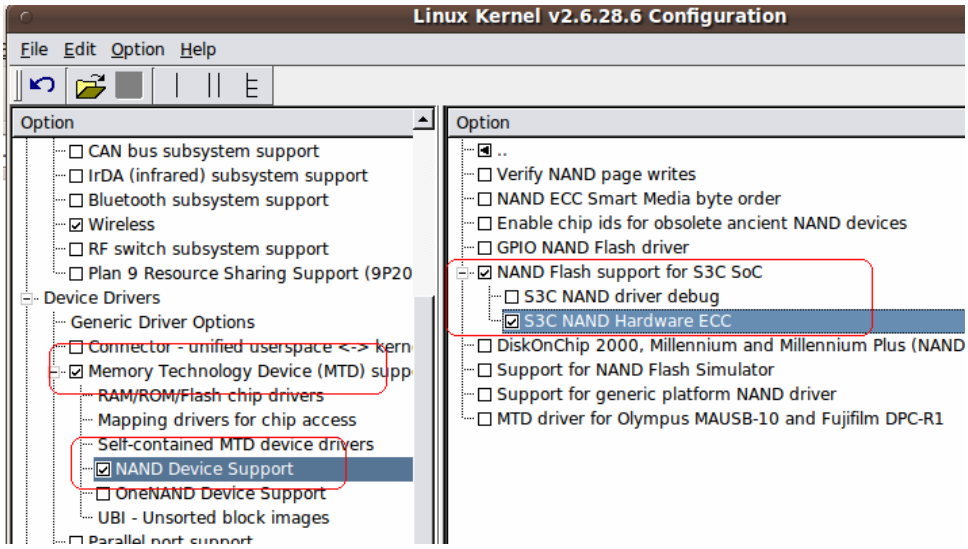
进行内核的编译。

下面开始介绍内核中各个功能模块的使用方式，主要是各个驱动模块的配置与测试。默认是使用 make xconfig 的配置界面。

注意：下面的大部分测试除了USB 部分只要使用编译好的内核镜像和文件系统便可进行，大部分的测试程序已经预装。

7.1.Nand flash 驱动

7.1.1 配置



Nandflash 驱动配置位于配置界面的 Device Drivers->Memory Technology Device 中，按照上图进行选择，其中要注意的是要选择 ECC 方式为 Hardware ECC。

对于 Nandflash 驱动还有一个重要的方面是 Nandflash 的分区，这个分区表位于内核源码目录的 arch/arm/plat-s3c/include/plat/partition.h 文件中：

```
    struct mtd_partition s3c_partition_info[] = {
#if 0
        {
                .name           = "Bootloader",
                .offset         = 0,
                .size       = (256*SZ_1K),
                .mask_flags     = MTD_CAP_NANDFLASH,
        },
        {
                .name           = "Kernel",
                .offset         = (256*SZ_1K),
                .size       = (4*SZ_1M) - (256*SZ_1K),
                .mask_flags     = MTD_CAP_NANDFLASH,
        },
#endif
        {
                .name           = "cramfs",
                .offset         = (4*SZ_1M),
                .size       = (4*SZ_1M),
        },
          {
```

```
                .name           = "ubifs",
                .offset         = MTDPART_OFS_APPEND,
                .size       = MTDPART_SIZ_FULL,
        }
};
```

这里我们只划分了 2 个文件系统分区。

7.1.2 测试

保存配置并编译内核，下载到板子中启动后将在启动信息中看到下面的输出：

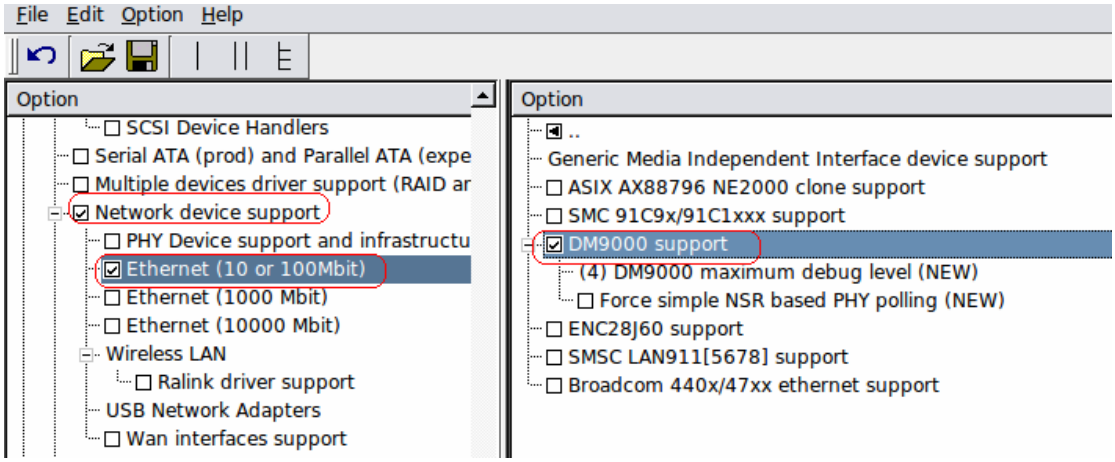
```
Driver 'sr' needs updating - please use bus_type methods
S3C NAND Driver, (c) 2008 Samsung Electronics

S3C NAND Driver, (c) 2008 Samsung Electronics
S3C NAND Driver is using hardware ECC.

S3C NAND Driver is using hardware ECC.
<6>NAND device: Manufacturer ID: 0xec, Chip ID: 0xd3 (Samsung NAND 1GiB 3,3V 8
-bit)

NAND device: Manufacturer ID: 0xec, Chip ID: 0xd3 (Samsung NAND 1GiB 3,3V 8-
bit)
<5>Creating 2 MTD partitions on "NAND 1GiB 3,3V 8-bit":

Creating 2 MTD partitions on "NAND 1GiB 3,3V 8-bit":
<5>0x00400000-0x00800000 : "cramfs"

0x00400000-0x00800000 : "cramfs"
<5>0x00800000-0x40000000 : "ubifs"
```

这表明 Nandflash 驱动已经成功加载，并对 Nandflash 划分了 2 个区。

7.2 DM9000AE 驱动

我们使用的 10/100M 以太网控制器型号为 DM9000AE。

7.2.1 配置



7.2.2 测试

保存配置编译内核并下载到板子中启动内核，挂载 cramfs 文件系统作为测试。进入文件系统后输入如下指令进行以太网的测试：

ifconfig eth0 up	使能网络接口
ifconfig eth0 192.168.1.20	配置板子 ip 地址
ping 192.168.1.178	ping 局域网中的其他机器 ip

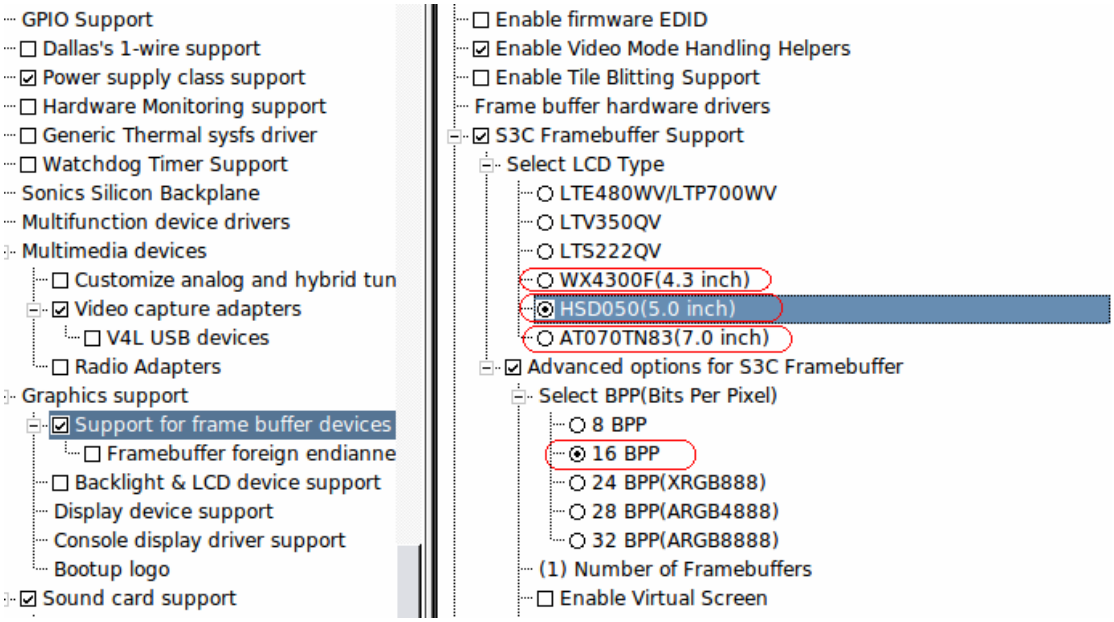
如果结果如下图所示那么表明网卡已经能够成功使用了。

```
[root@Real6410 /]# ifconfig eth0 up
[root@Real6410 /]# ifconfig eth0 192.168.1.20
[root@Real6410 /]# ping 192.168.1.178
PING 192.168.1.178 (192.168.1.178): 56 data bytes
64 bytes from 192.168.1.178: seq=0 ttl=64 time=31.313 ms
64 bytes from 192.168.1.178: seq=1 ttl=64 time=0.517 ms
64 bytes from 192.168.1.178: seq=2 ttl=64 time=0.828 ms
64 bytes from 192.168.1.178: seq=3 ttl=64 time=0.495 ms
64 bytes from 192.168.1.178: seq=4 ttl=64 time=0.496 ms
64 bytes from 192.168.1.178: seq=5 ttl=64 time=0.488 ms
64 bytes from 192.168.1.178: seq=6 ttl=64 time=0.501 ms
64 bytes from 192.168.1.178: seq=7 ttl=64 time=0.492 ms
64 bytes from 192.168.1.178: seq=8 ttl=64 time=0.511 ms
64 bytes from 192.168.1.178: seq=9 ttl=64 time=0.482 ms
64 bytes from 192.168.1.178: seq=10 ttl=64 time=0.513 ms
64 bytes from 192.168.1.178: seq=11 ttl=64 time=0.471 ms
64 bytes from 192.168.1.178: seq=12 ttl=64 time=0.504 ms
64 bytes from 192.168.1.178: seq=13 ttl=64 time=0.502 ms
64 bytes from 192.168.1.178: seq=14 ttl=64 time=0.524 ms
64 bytes from 192.168.1.178: seq=15 ttl=64 time=0.479 ms
64 bytes from 192.168.1.178: seq=16 ttl=64 time=0.530 ms
64 bytes from 192.168.1.178: seq=17 ttl=64 time=0.477 ms
64 bytes from 192.168.1.178: seq=18 ttl=64 time=0.512 ms
64 bytes from 192.168.1.178: seq=19 ttl=64 time=0.584 ms
64 bytes from 192.168.1.178: seq=20 ttl=64 time=0.518 ms
64 bytes from 192.168.1.178: seq=21 ttl=64 time=0.507 ms
```

7.3 LCD 驱动

我们提供了多种尺寸的 LCD 显示器，包括 4.3``、5``、7``等，下面针对不同尺寸进行配置。

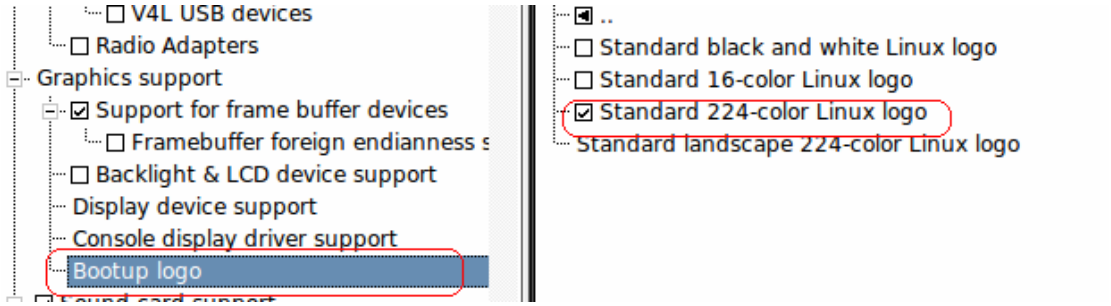
7.3.1 配置



注意要根据自己的 LCD 尺寸选择不同的 LCD 型号，不然显示将有问题。红色部分是我们提供的针对不同 LCD 的配置选项。

7.3.2 测试

LCD 测试最简单的办法就是使用内核自带的开机界面的显示图案，如下图所示的配置方法选择一个启动画面：



Linnux 内核默认是一个在 LCD 左上方的小企鹅，不过我们修改为我们自己制作的一个小企鹅加 Google logo 的图片，表意为提供的 2 个操作操作系统（Linux+Android）。对于开机界面的制作可以参考《附录一 linux 启动画面的制作教程》。

保存配置编译内核，并下载到板子中启动，如果 LCD 驱动没有问题将在 LCD 上看到一个漂亮的开机画面☺

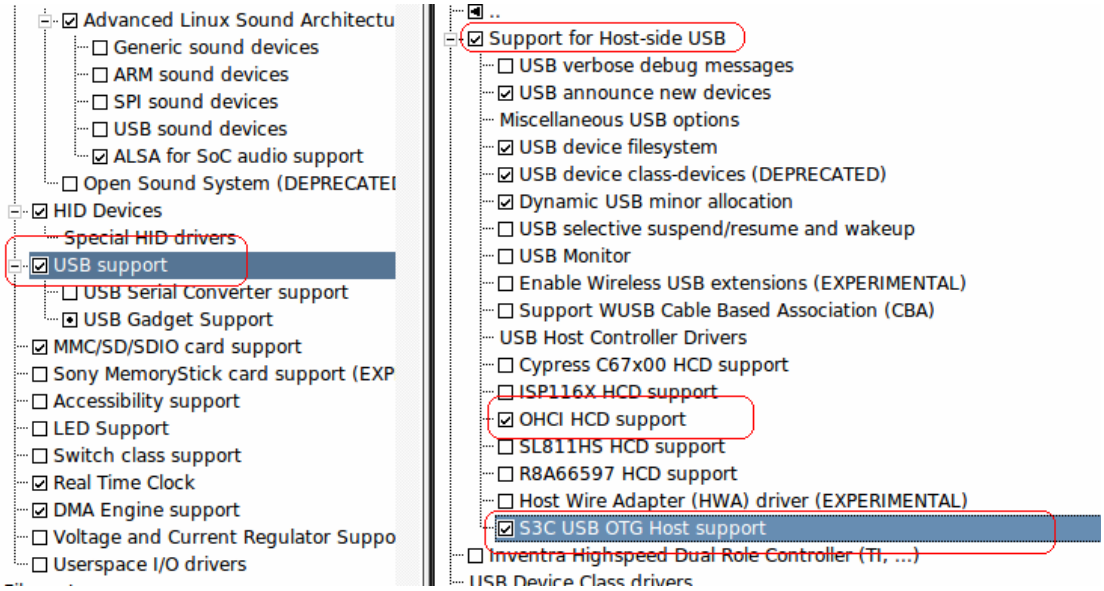
7.4 USB OTG 驱动

S3C6410 中包含了内核中包含了一个 usb-2.0 标准的 USB OTG 接口，OTG 接口能够作为 host 也能作为 device，但是目前还不能同时把 host 和 device 同时编译在一起，否则会出现冲突。这个问题可以通过修改驱动解决，这个问题将在后面的更新中解决。

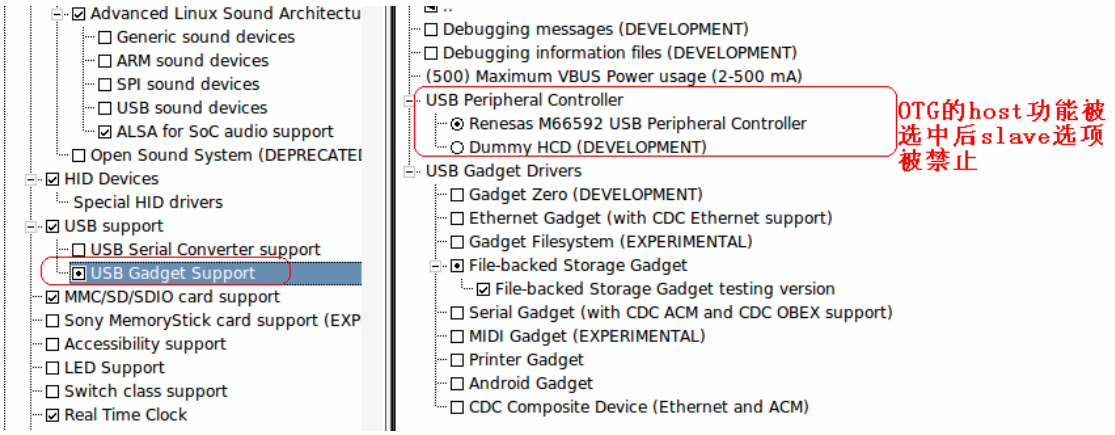
在我们的板子中，小口 USB 就是 OTG 接口，大口 USB 为下一节介绍的 USB Host 接口。

7.4.1 OTG 的 Host 功能配置

这里首先是配置 OTG 的 host 功能：

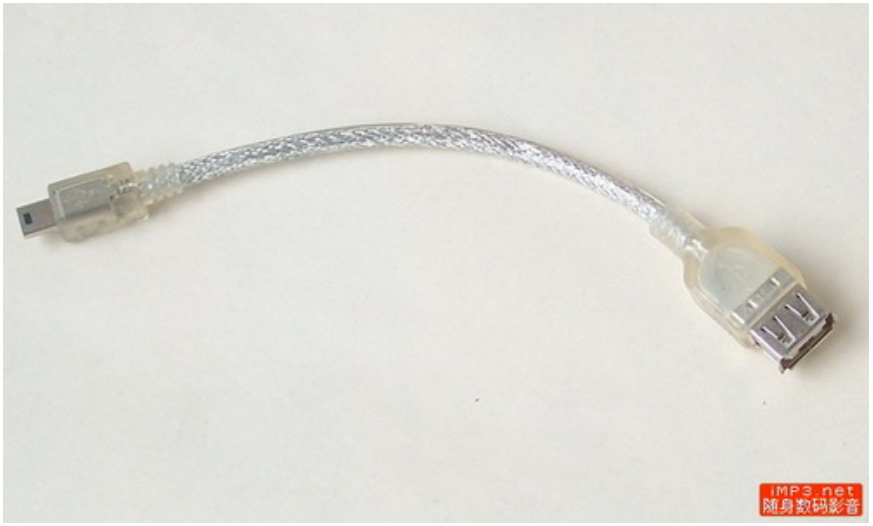


选中 host 功能后，OTG 的 slave 功能将会被禁止：



7.4.2 OTG 的 Host 功能测试

要测试 OTG 首先要准备一个 OTG 线，这种线的形状如下：



这种线可以在淘宝或者数码市场买到，也可以参考《附录二 DIY 一根 OTG 线》，使用普通的 USB 线制作一根可用的 OTG 线。不过我的 OTG 线不好用，更没时间自己做一根，而是直接在电路板上将 OTG 的 ID 引脚拉到地来做一下试验。

配置结束后保持并重新编译内核下载到板子中，如果没有意外，启动过程中将看到下面信息：

```
hub 2-0:1.0: USB hub found
<6>hub 2-0:1.0: 1 port detected
hub 2-0:1.0: 1 port detected
<6>usb usb2: New USB device found, idVendor=1d6b, idProduct=0002
usb usb2: New USB device found, idVendor=1d6b, idProduct=0002
<6>usb usb2: New USB device strings: Mfr=3, Product=2, SerialNumber=1
usb usb2: New USB device strings: Mfr=3, Product=2, SerialNumber=1
<6>usb usb2: Product: S3C OTGHCD
usb usb2: Product: S3C OTGHCD
<6>usb usb2: Manufacturer: Linux 2.6.28.6 EMSP_OTGHCD
usb usb2: Manufacturer: Linux 2.6.28.6 EMSP_OTGHCD
<6>usb usb2: SerialNumber: s3c_otghcd
usb usb2: SerialNumber: s3c_otghcd
<6>Initializing USB Mass Storage driver...
Initializing USB Mass Storage driver...
<6>usbcore: registered new interface driver usb-storage
usbcore: registered new interface driver usb-storage
<6>USB Mass Storage support registered.
USB Mass Storage support registered.
```

启动后使用 OTG 线连接 U 盘和板子 OTG 接口，将输出下面信息：

```
hub 1-0:1.0: unable to enumerate USB device on port 2
<6>usb 2-1: new high speed USB device using s3c_otghcd and address 3
usb 2-1: new high speed USB device using s3c_otghcd and address 3
<6>usb 2-1: configuration #1 chosen from 1 choice
usb 2-1: configuration #1 chosen from 1 choice
<6> uba: uba: uba1 uba1

<6>usb 2-1: New USB device found, idVendor=0204, idProduct=6025
usb 2-1: New USB device found, idVendor=0204, idProduct=6025
<6>usb 2-1: New USB device strings: Mfr=1, Product=2, SerialNumber=3
usb 2-1: New USB device strings: Mfr=1, Product=2, SerialNumber=3
<6>usb 2-1: Product: Flash Disk
usb 2-1: Product: Flash Disk
<6>usb 2-1: Manufacturer:
usb 2-1: Manufacturer:
<6>usb 2-1: SerialNumber: CCCBB999988887777
usb 2-1: SerialNumber: CCCBB999988887777
```

板子中的 mdev 会自动挂载这个 U 盘到/mnt/udisk 下，使用 ls 看一下即可看到 U 盘内容：

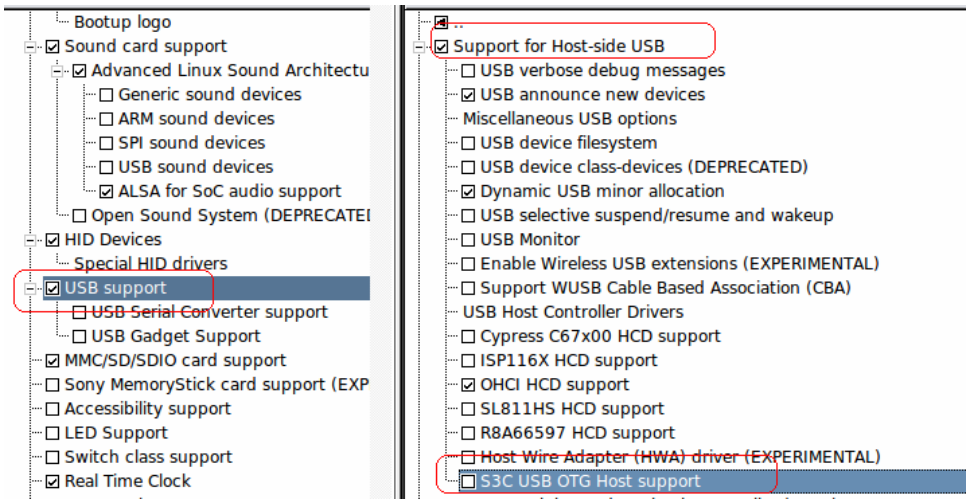
```
<6>usb 2-1: New USB device found, idVendor=0204, idProduct=6025
usb 2-1: New USB device found, idVendor=0204, idProduct=6025
<6>usb 2-1: New USB device strings: Mfr=1, Product=2, SerialNumber=3
usb 2-1: New USB device strings: Mfr=1, Product=2, SerialNumber=3
<6>usb 2-1: Product: Flash Disk
usb 2-1: Product: Flash Disk
<6>usb 2-1: Manufacturer:
usb 2-1: Manufacturer:
<6>usb 2-1: SerialNumber: CCCBB999988887777
usb 2-1: SerialNumber: CCCBB999988887777

[root@Real6410 /]# ls /mnt/u
/mnt/ubi/    /mnt/udisk/
[root@Real6410 /]# ls /mnt/udisk/
??linux?3G???????.doc              REAL6410????V1.0.pdf
Android????.doc                    porting qte-4.5.2 to S3C6410.doc
[root@Real6410 /]#
```

如果没有自动挂载，可以使用手工挂载：

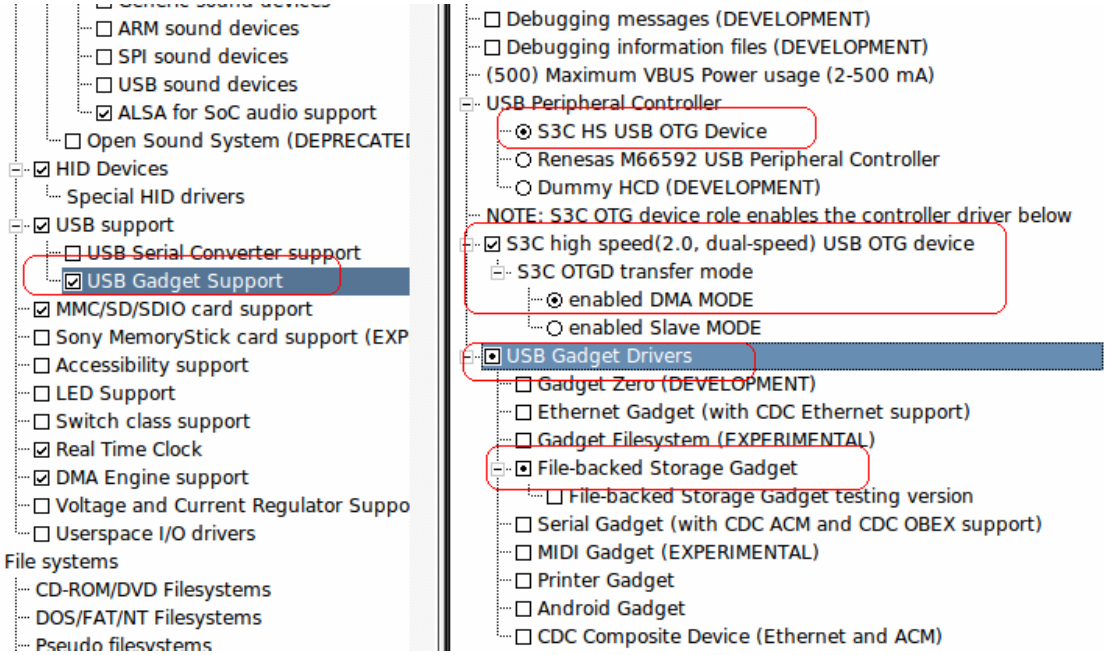
mount –t vfat /dev/uba1 /mnt/udisk

7.4.3 OTG 的 slave (device)功能配置

首先要取消 OTG 的 host 选中：



接着选中 OTG 的 slave 功能：



要注意，USB Gadget Drivers 是以驱动模块的形式编译的，而不是编译到内核中。

7.4.4 OTG 的 slave (device)功能测试

保存上面的配置后编译，将内核 zImage 下载到板子中，并把中的 drivers/usb/gadget/g_file_storage.ko 模块复制到板子根文件系统的/root 目录中，重启系统，进入文件系统后在板子命令中作以下操作：

```
dd if=/dev/zero of=10m bs=1M count=10
insmod /root/g_file_storage.ko file=10m stall=1 removable=1
```

第一个指令是创建一个虚拟盘，大小为 10MB，第二个指令挂载 g_file_storage.ko 模块，并把刚刚创建的虚拟盘作为储存介质。

执行结果如下：

```
~ # dd if=/dev/zero of=10m bs=1M count=10
e_storage.ko file=10m stall=1 removable=110+0 records in
10+0 records out
~ # insmod /root/g_file_storage.ko file=10m stall=1 removable=1
g_file_storage gadget: File-backed Storage Gadget, version: 7 August 2007
g_file_storage gadget: Number of LUNs=1
g_file_storage gadget-lun0: ro=0, file: /10m
Registered gadget driver 'g_file_storage'
~ #
```

这时通过 usb 线连接 pc 和板子上的 Mini USB 即可在 pc 上看到一个 10MB 的 U 盘。也可以使用 SD 作为 g_file_storage.ko 的储存介质，命令如下：

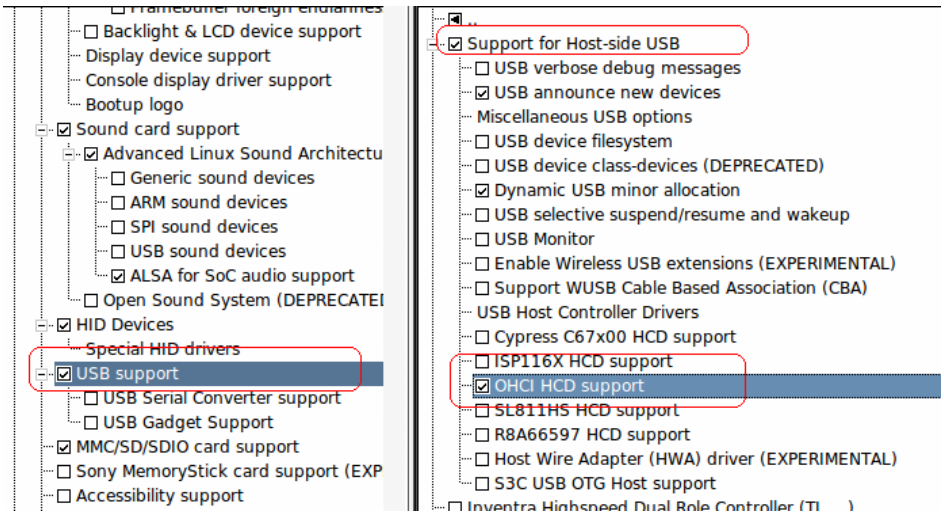
```
insmod /root/g_file_storage.ko file=/dev/mmcblk0p1 stall=0 removable=1
```

其中/dev/mmcblk0p1 是 SD 的设备节点，这时候板子就是一个 SD/MMC 读卡器了。

7.5 USB host 驱动

板子中的大口 USB 就是符合 USB-1.1 的 host 接口。

7.5.1 配置



这样只要选中 OHCI HCD support 就是选择了 S3C6410 的 host。

7.5.2 测试

保存、编译、下载内核，进入文件系统后，插入 U 盘到 USB host 接口，将输出下面信息：

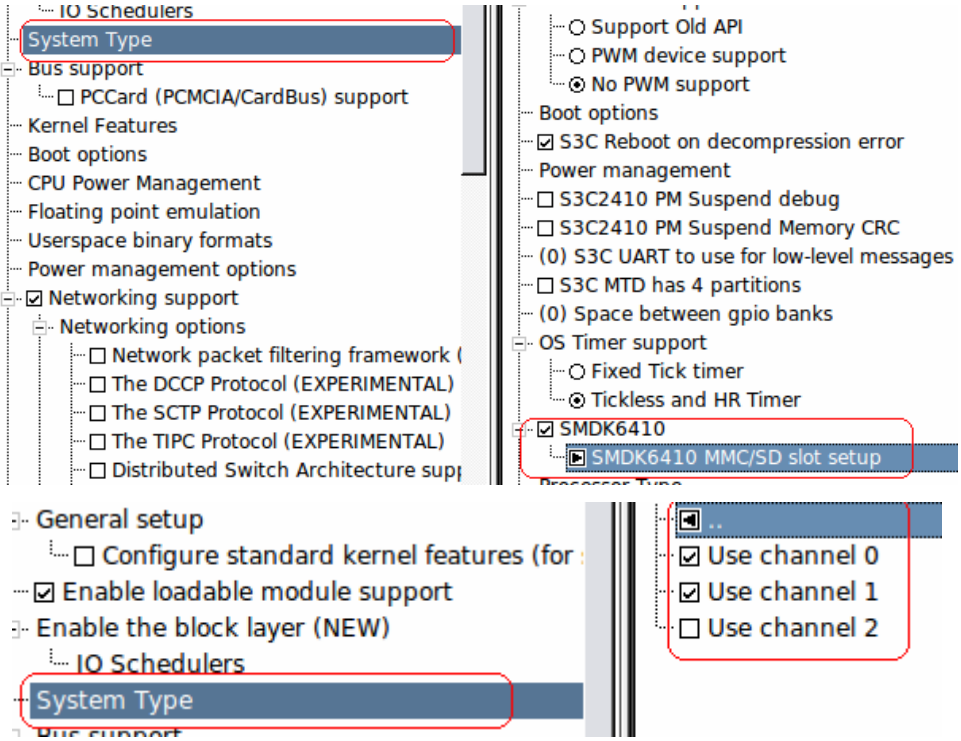
```
[root@Real6410 /]# <6>usb 1-1: new full speed USB device using s3c2410-ohci and address 2
usb 1-1: new full speed USB device using s3c2410-ohci and address 2
<6>usb 1-1: configuration #1 chosen from 1 choice
usb 1-1: configuration #1 chosen from 1 choice
<6> uba: uba: uba1 uba1

<6>usb 1-1: New USB device found, idVendor=0204, idProduct=6025
usb 1-1: New USB device found, idVendor=0204, idProduct=6025
<6>usb 1-1: New USB device strings: Mfr=1, Product=2, SerialNumber=3
usb 1-1: New USB device strings: Mfr=1, Product=2, SerialNumber=3
<6>usb 1-1: Product: Flash Disk
usb 1-1: Product: Flash Disk
<6>usb 1-1: Manufacturer:
usb 1-1: Manufacturer:
<6>usb 1-1: SerialNumber: CCCBB999988887777
usb 1-1: SerialNumber: CCCBB999988887777
```

这时候 U 盘应该已经被自动挂载到/mnt/udisk 中，使用

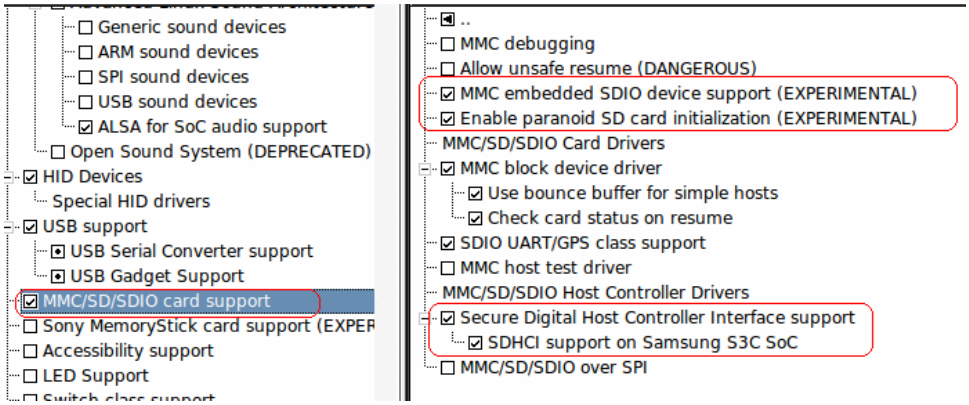
7.6 MMC/SD 驱动

7.6.1 配置

首先要在 System Type->SMDK6410->SMDK6410 MMC/SD slot Setup 中选择所需要的 MMC/SD 通道：



这里选择的是通道 0 和同道 1，其中通道 0 连接着 WIFI 模块，同道 1 连接着 SD 卡。下面继续配置 MMC/SD 驱动：



7.6.2 测试

保存配置并重新编译内核下载到板子中，插入 SD 卡后给板子上电启动，如果没有意外将在启动信息中看到下面的输出：

```
<6>asoc: AC97 HiFi <-> s3c64xx-ac97 mapping ok
asoc: AC97 HiFi <-> s3c64xx-ac97 mapping ok
<6>mmc1: new high speed SD card at address e712
mmc1: new high speed SD card at address e712
<6>mmcblk0: mmc1:e712 SD02G 1.83 GiB
mmcblk0: mmc1:e712 SD02G 1.83 GiB
<6> mmcblk0: mmcblk0: p1 p1

<6>ALSA device list:
ALSA device list:
```

接着在进入根文件系统后，运行下面的命令查看 SD 是否已经被识别：

ls /dev/mmcblk0

如果这个节点存在，那么表明已经被识别了。而我们已经使用 busybox 中的 mdev 自动挂载这个 SD 到文件系统中的/mnt/sdcard 目录下，在这个目录下 ls 一下便能看到 SD 中的文件。如果没有自动挂载，可以使用下面的命令进行挂载：

mount –t vfat /dev/mmcblk0 /mnt/sdcard

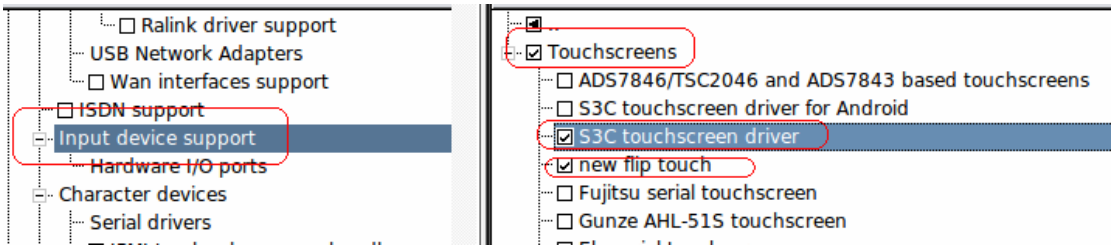
如果没有输出什么错误提示那么应该已经挂载成功，运行下面指令查看 SD 中的文件内容：

ls /mnt/sdcard

如果没有意外将显示出 SD 中的文件。

7.7 Touchscreen 驱动

7.7.1 配置



这里需要注意，我们提供了分别用于 Android 和 Qtopia 下的触摸屏驱动，这两者主要区别就是校准方式的不一样，在下文会分别介绍这两种情况下的校准。这里选择 Qtopa 下的触摸屏选择 S3C touchscreen driver。

7.7.2 测试

板子中提供了一个 ts_calibrate 的校准程序，这个程序会打印出每次按下的触摸屏坐标，使用这个作为测试程序即可：

运行/usr/local/bin/ts_calibrate，这时候 LCD 出现校准界面，按照提示点击屏幕便能在命

令行中看到下面的输出信息：

```
[root@Real6410 /]# /usr/local/bin/ts_calibrate
xres = 480, yres = 272
Took 41 samples...
Top left : X = 2842 Y = 1609
Took 30 samples...
Top right : X = 1063 Y = 1617
Took 26 samples...
Bot right : X = 1056 Y = 2552
Took 31 samples...
Bot left : X = 2861 Y = 2534
Took 29 samples...
Center : X = 1953 Y = 2083
651.721680 -0.212035 0.001351
-251.100830 0.001346 0.184931
Calibration constants: 42711232 -13895 88 -16456144 88 12119 65536
[root@Real6410 /]#
```

7.7.3 触摸屏的校准

Qtopia 和 Android 中使用触摸屏都需要校准，这里分别说明一下这 2 个系统下触摸屏的校准方式。

对于 Qtopia 系统下如果发现触摸屏不准，或者触摸屏驱动已经加载但是点击触摸屏没有反应，都可以先进入系统后运行下面命令删除之前的校准文件并重启系统校准:

```
rm /etc/pointercal
reboot
```

这里要注意，reboot 命令已经是重启系统，切不要在这时候断电重启或者 reset 重启，等待下次启动后便能进入校准。

对于 Android 的校准则要复杂一些。目前使用的策略是直接在驱动层进行校准，让驱动层输出的触摸屏数据就是已经校准后的数据。方法如下：

- 让板子进入 qtopia 或者 nfs 等可读写的根文件系统，cramfs 因为是不可写的因此不能用来做这个工作。
- 进入 qtopia 或者 nfs 的 shell 界面后，运行/usr/local/bin/ts_calibrate，在输出下图信息后，读取红色圈住的部分的 7 个数据。

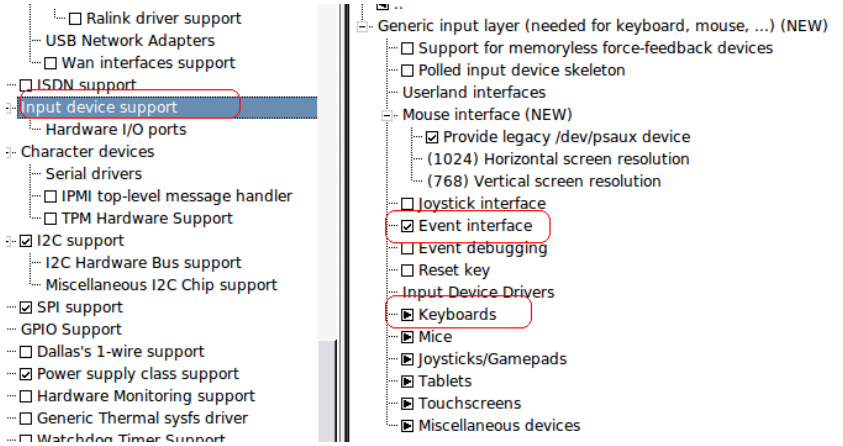
```
[root@Real6410 /]# /usr/local/bin/ts_calibrate
xres = 480, yres = 272
Took 41 samples...
Top left : X = 2842 Y = 1609
Took 30 samples...
Top right : X = 1063 Y = 1617
Took 26 samples...
Bot right : X = 1056 Y = 2552
Took 31 samples...
Bot left : X = 2861 Y = 2534
Took 29 samples...
Center : X = 1953 Y = 2083
651.721680 -0.212035 0.001351
-251.100830 0.001346 0.184931
Calibration constants: 42711232 -13895 88 -16456144 88 12119 65536
[root@Real6410 /]#
```

- 将上面这 7 个数据用来代替
  drivers/input/touchscreen/s3c-ts_android.c
  中的 pointercal 数组的 7 个数据。
- 重新编译内核便能实现 Android 下使用的触摸屏驱动的校准。

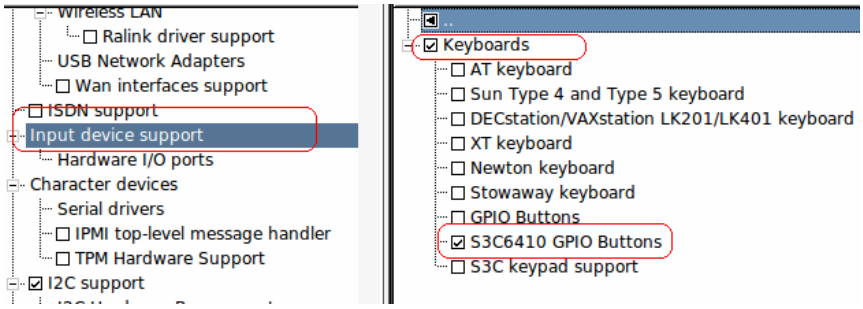
另外，对于 Android 来说，不同尺寸 LCD 需要不同的校准系数，不过这里并不需要做什么，因为只要选中合适尺寸的 LCD 尺寸，s3c-ts_android.c 就会根据这个尺寸选择合适的系数。

7.8 Keyboard 驱动

7.8.1 配置



接着进入 Keyboards 中进一步配置：



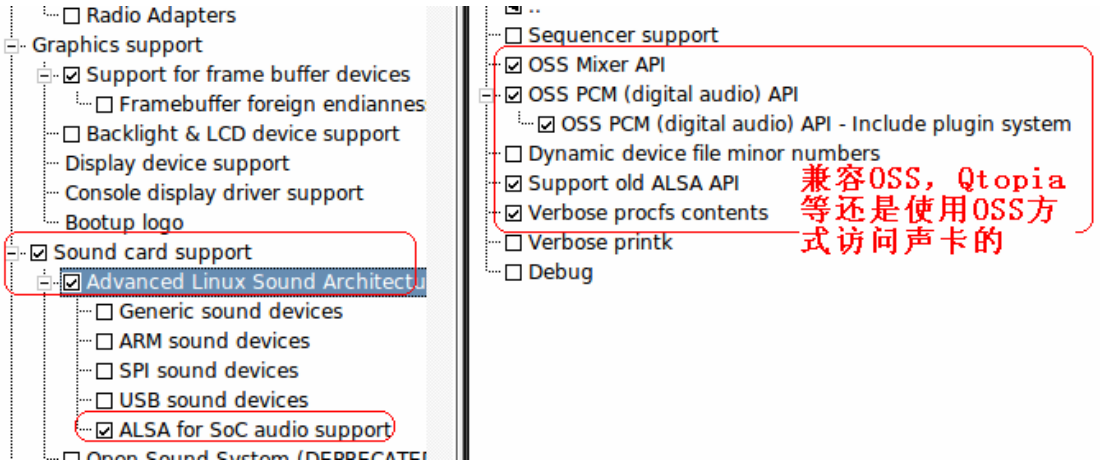
7.8.2 测试

键盘驱动可以在进入 Qtopia 或者 Android 系统后按下各个按键进行测试，不过这个键盘对 Qtopia 的支持似乎不是很好。

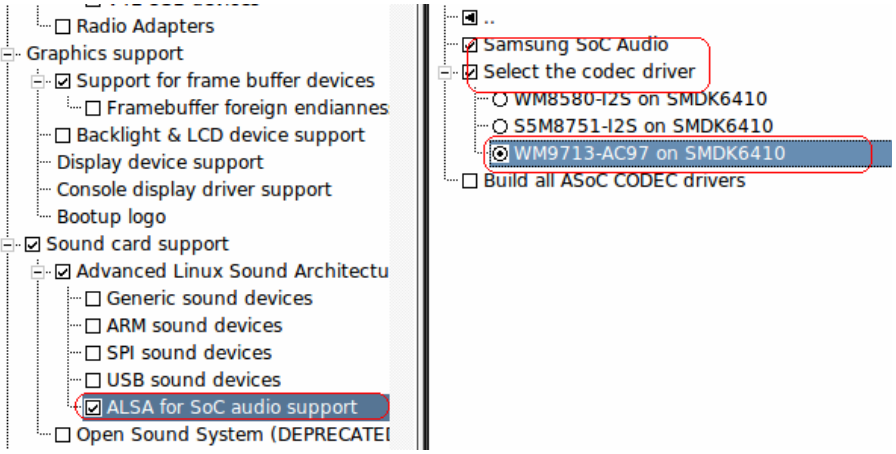
另外光盘中的 linux\Applications\keyboard 也可以用来测试。

7.9 Audio 驱动

7.9.1 配置



点击进入 ALSA for SoC audio support 进一步配置：



7.9.2 测试

进入 Qtopia 后运行其中的 Mp3 播放器，插入耳机到耳机插孔中便能听到音乐。也可以在命令行中运行下面命令测试：

```
madplay /root/Documents/Innocent.mp3
```

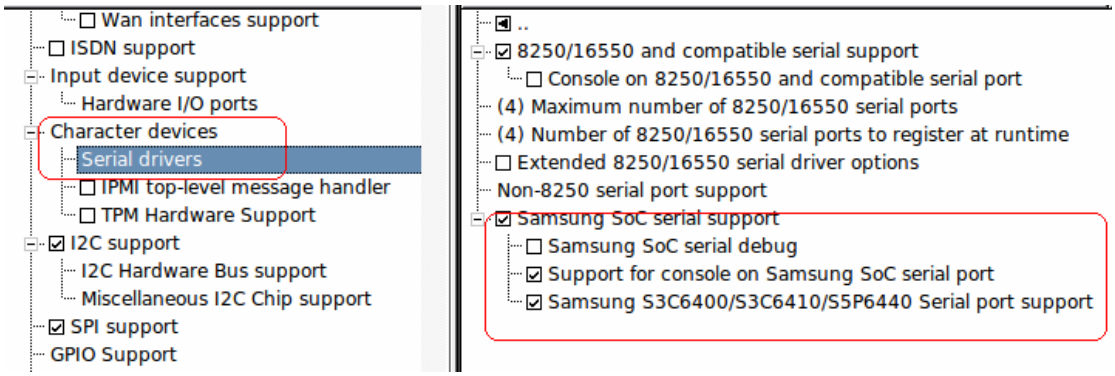
```
[root@Real6410 /]# madplay /root/Documents/Innocent.mp3
MPEG Audio Decoder 0.15.2 (beta) - Copyright (C) 2000-2004 Robert Leslie et al.
         Title: Innocent
        Artist: Joe Hisaishi
         Album: piano stories
         Track: 9
          Year: 2000
         Genre: New Age
       Comment: www.VeryCD.com
```

不过不够完美的地方是还有一点噪音，后续会做进一步修正这个 bug。

7.10 GPRS 驱动

7.10.1 配置

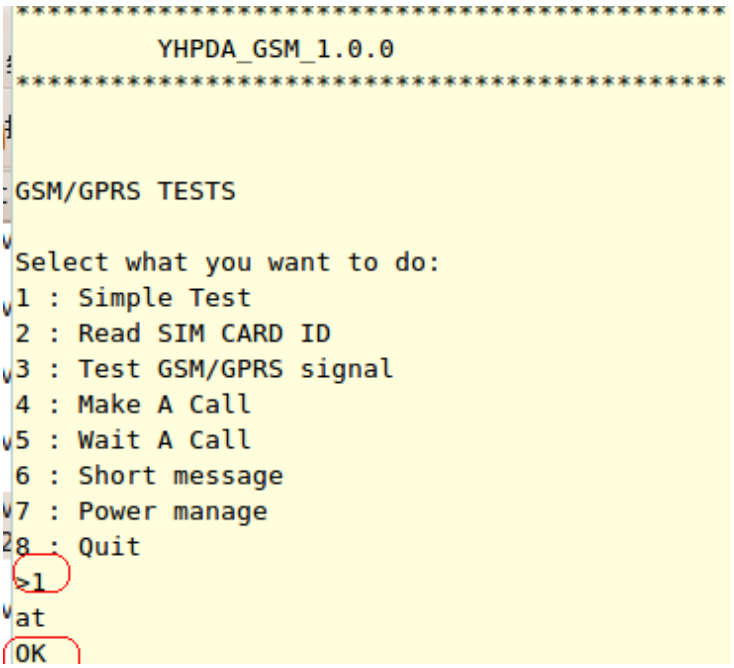
GPRS（SIM300）模块是通过 Uart1 连接到处理器的，因此 GPRS 驱动从功能上看实际上就是一个串口驱动。如下图配置好串口驱动即可：



目前这个驱动支持了 S3C6410 的 4 个串口。

7.10.2 测试

在板子中运行 linux\Applications\gprs\gprs_s3c2410_linux 程序，结果如下：



这里可以分别选择 1~8 测试不同功能。上图就是测试了最简单的 AT 指令。

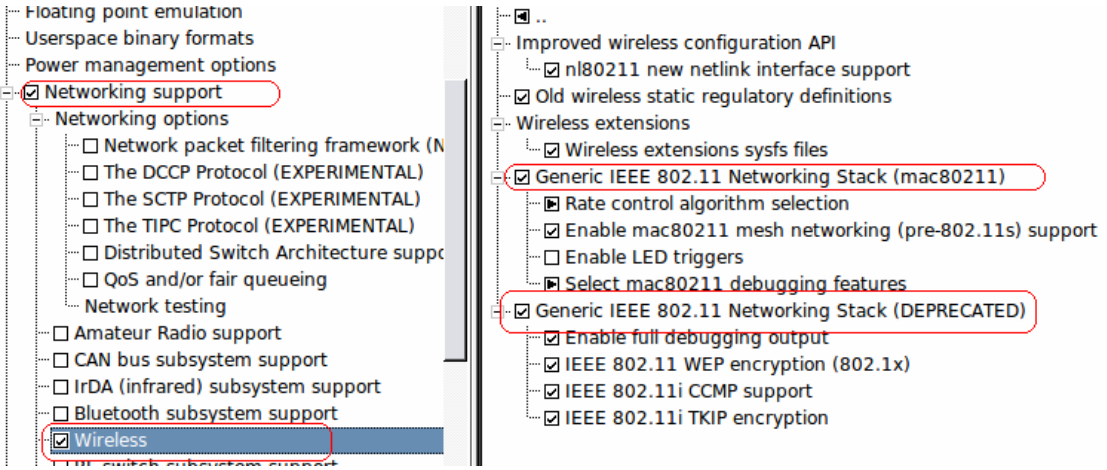
如果上面的测试不成功，需要检查硬件的连接情况。GPRS 的 LED 也可以辅助调试这个模块，如果这个 LED 一直没亮那么很可能就是硬件有问题了。如果 LED 在闪但是测试仍

然不通过的话，断电重启。

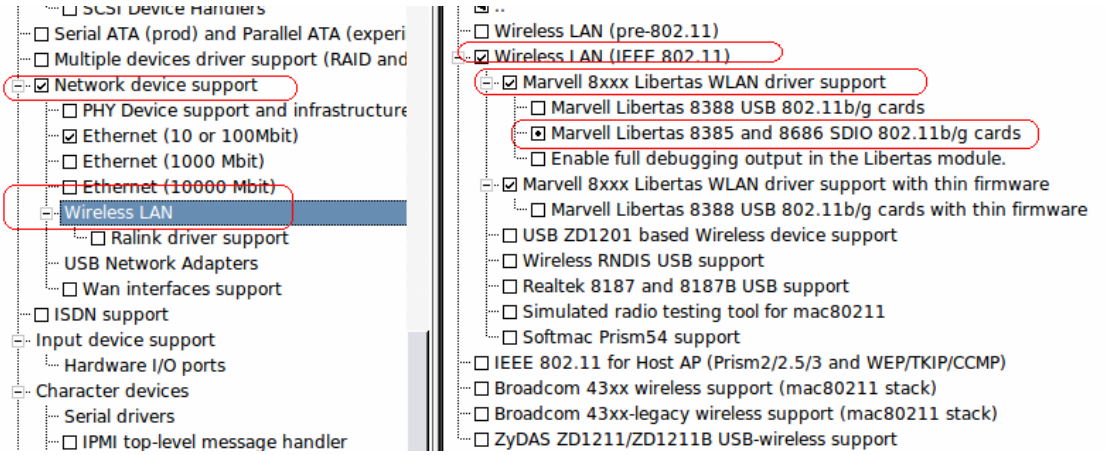
7.11 WIFI 驱动

7.11.1 配置

要使用 WIFI，首先在网络协议层就要配置好 802.11 的支持：



接着就要配置 WIFI 驱动的支持了：



这里将 Marvel Libertas 8385 and 8686 SDIO 802.11 b/g cards 编译为模块。

7.11.2 测试

保存、编译、下载内核，并把 drivers/net/wireless/libertas/libertas_sdio.ko 复制到板子的/root 目录下，并运行下面的加载命令：

insmod /root/libertas_sdio.ko helper_name=/lib/firmware/helper_sd.bin fw_name=/lib/firmware/sd8686.bin

这里还有 2 个固件文件 helper_sd.bin、sd8686.bin 可以从 Marvel 官方网站下载，并放到

lib 下。命令执行结果如下：

```
[root@Real6410 /]# insmod /root/libertas_sdio.ko helper_name=/lib/firmware/helpe
r_sd.bin fw_name=/lib/firmware/sd8686.bin
<6>libertas_sdio: Libertas SDIO driver
libertas_sdio: Libertas SDIO driver
<6>libertas_sdio: Copyright Pierre Ossman
libertas_sdio: Copyright Pierre Ossman
<6>libertas_sdio mmc0:0001:1: firmware: requesting /lib/firmware/helper_sd.bin
libertas_sdio mmc0:0001:1: firmware: requesting /lib/firmware/helper_sd.bin
<6>libertas_sdio mmc0:0001:1: firmware: requesting /lib/firmware/sd8686.bin
libertas_sdio mmc0:0001:1: firmware: requesting /lib/firmware/sd8686.bin
<6>libertas: 00:22:43:6b:0b:e0, fw 9.70.3p24, cap 0x000003a3
libertas: 00:22:43:6b:0b:e0, fw 9.70.3p24, cap 0x000003a3
<6>libertas: PREP_CMD: command 0x00a3 failed: 2
libertas: PREP_CMD: command 0x00a3 failed: 2
<6>libertas: PREP_CMD: command 0x00a3 failed: 2
libertas: PREP_CMD: command 0x00a3 failed: 2
<6>libertas: eth1: Marvell WLAN 802.11 adapter
libertas: eth1: Marvell WLAN 802.11 adapter
[root@Real6410 /]#
```

接着分别运行下面的命令便能找到本地的 WIFI 网络：

```
ifconfig eth1 up
iwlist eth1 scanning
```

运行结果如下：

```
iwconfig  iwevent   iwgetid   iwlist    iwpriv    iwspy
[root@Real6410 /]# ifconfig eth1 up
[root@Real6410 /]# iwlist eth1 scanning
eth1      Scan completed :
          Cell 01 - Address: 00:23:69:CE:54:9C
                    ESSID:"lab410"
                    Mode:Managed
                    Frequency:2.412 GHz (Channel 1)
                    Quality=86/100  Signal level=-64 dBm  Noise level=-96 dBm
                    Encryption key:on
                    Bit Rates:1 Mb/s; 2 Mb/s; 5.5 Mb/s; 11 Mb/s; 18 Mb/s
                              24 Mb/s; 36 Mb/s; 54 Mb/s; 6 Mb/s; 9 Mb/s
                              12 Mb/s; 48 Mb/s
                    IE: WPA Version 1
                        Group Cipher : TKIP
                        Pairwise Ciphers (1) : TKIP
                        Authentication Suites (1) : PSK
          Cell 02 - Address: 94:0C:6D:5A:36:14
                    ESSID:"209"
```

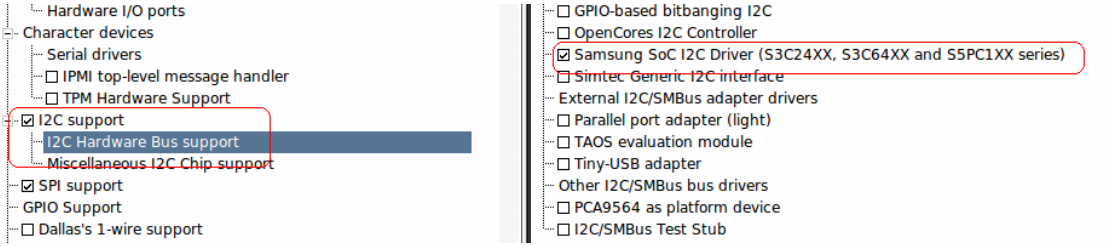
这样便表面 WIFI 模块能够正常工作了。不过能够找到 AP 点到能够上网还有段路要走，因为单单使用 iwlist、iwconfig 等几个简单的配置命令是很难配置好无线网络的，一般的应用都是结合 wpa_supplicant 等程序来管理 WIFI 网络，如 Android 就是这样做的。只是使用简单的 iwconfig 等命令只能连接上那些没有加密的 AP 点，对于有各种加密的 AP 点，请各位读者试着使用 wpa_supplicant。

另外，需要注意的是，WIFI 如果不正常工作后需要断电重启，如果带电复位将不能回复正常。

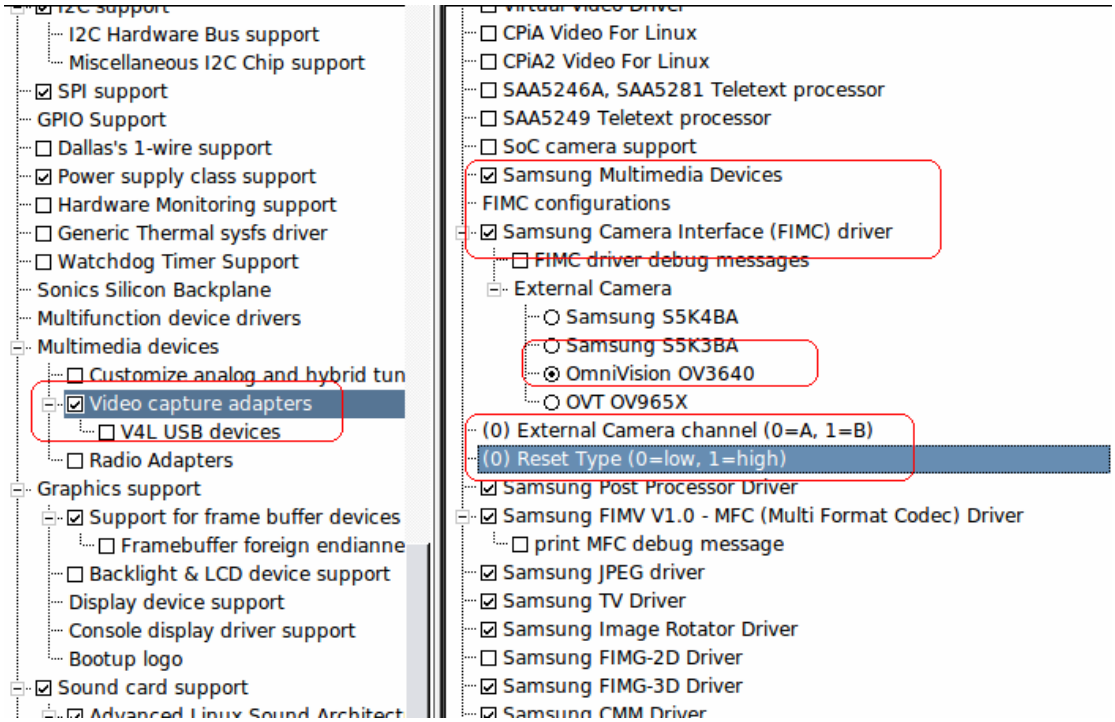
7.12 Camera 驱动

7.12.1 配置

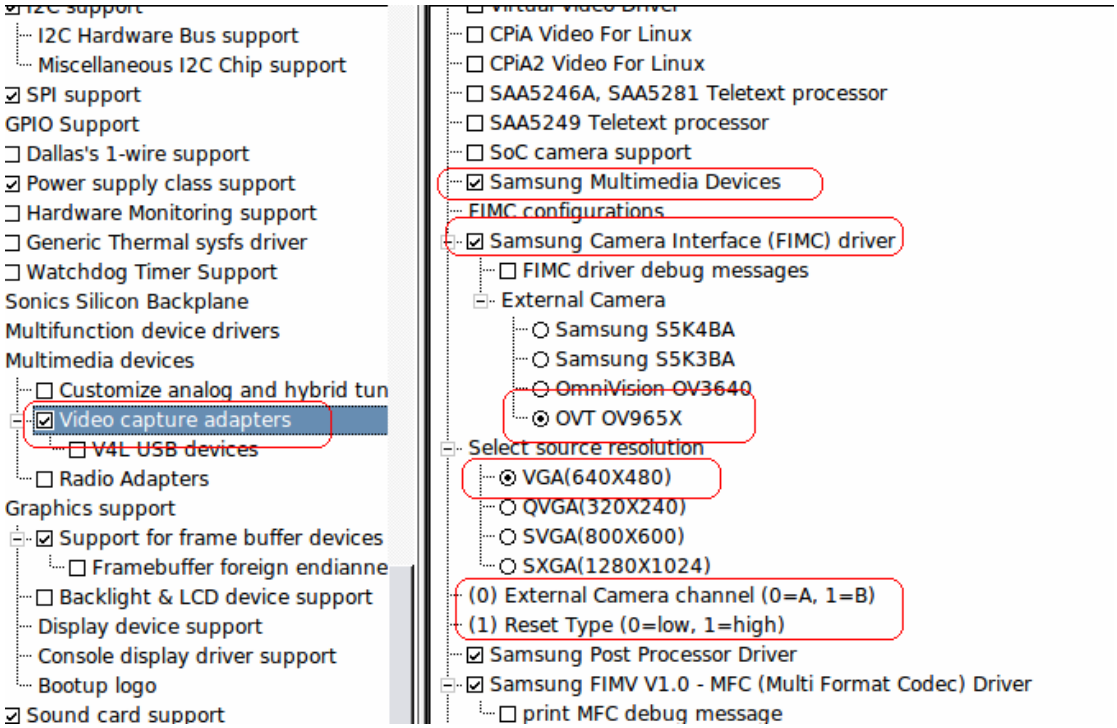
首先要配置 i2c 子系统，因为控制摄像头要用到 i2c 子系统里面的 API 函数。配置情况如下图：



接着配置 Camera 部分，如果是使用 300 万像素的 ov3640 摄像头模组，则配置情况如下图：



如果是使用 130 万像素的 ov9650 摄像头模组，则配置如下：



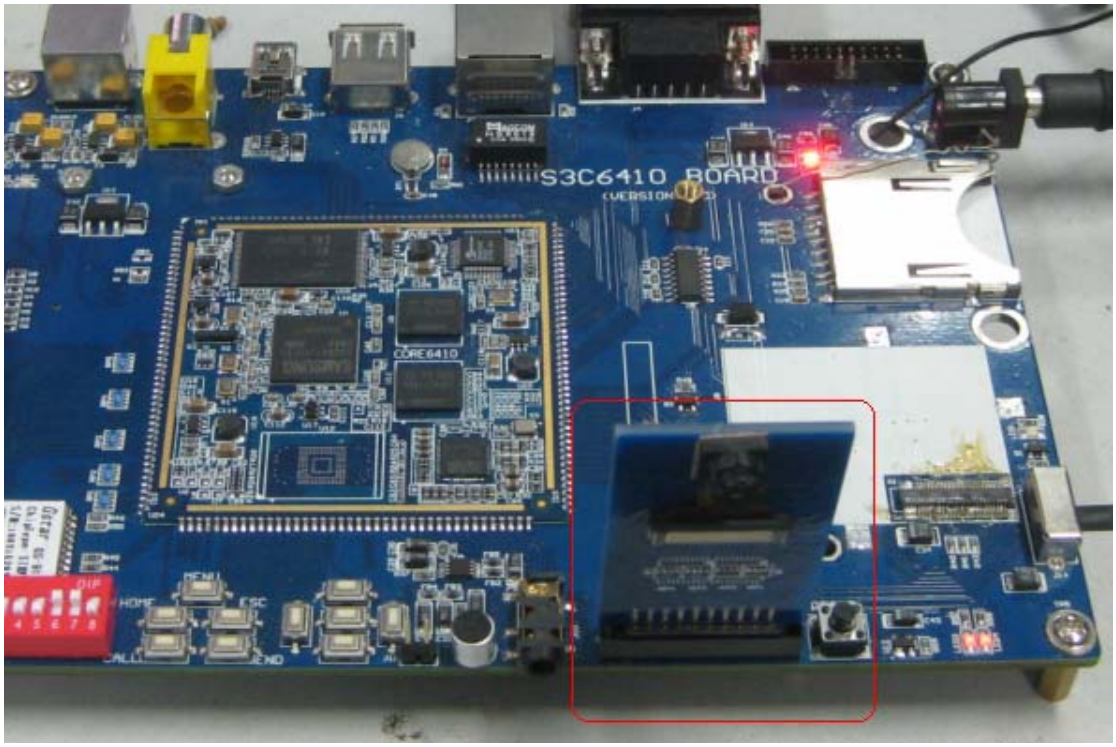
7.12.2 Camera 型号的识别

因为有个别用户不知道自己买的是什么型号的 Camera，而我们目前只提供 ov9650 和 0v3640 这 2 种，简单区别如下：

如果 Camera 的软排线（就是摄像头连到摄像头模组的排线）上有 SOHOO 字样那就是 ov9650，反之就是 ov3640。

7.12.3 测试

在测试摄像头的时候首先要注意摄像头的插法，下图中显示了摄像头位置：



接着启动板子，在板子上运行光盘中 linux\Applications\camera\v4l2grab 程序便可进行测试（文件系统中已经预装了这个程序），输入下面的命令：

v4l2grab -d /dev/video0 -o figo.jpg -q 70 -m -W 640 -H 480

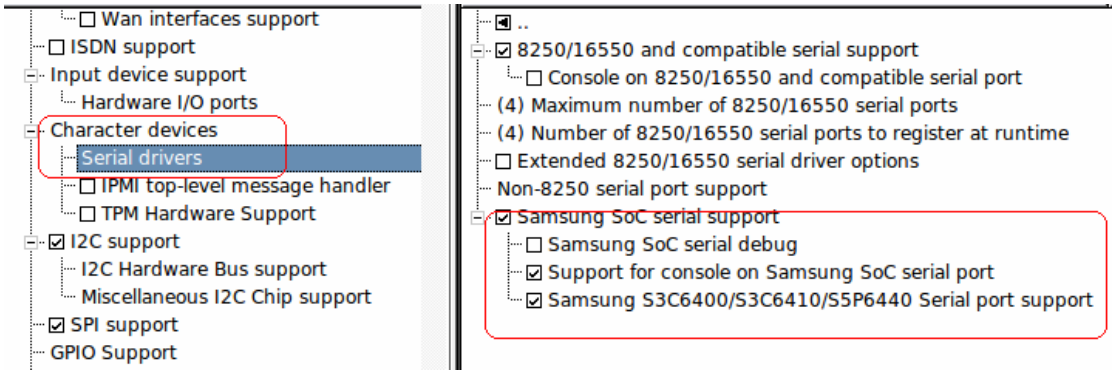
这样就能产生一个图片了：

如果要动态显示摄像头视频，可以使用多媒体测试部分（十一章）的菜单 6 进行测试。

7.13 GPS 驱动

和 GPRS 模块一样，GPS 模块也是连接到串口上的，在电路板上是连接到了 Uart2 中，因此 GPS 的驱动实际上就是串口 2 的驱动。

7.13.1 配置



7.13.2 测试

在板子中运行光盘中的 linux\Applications\gps\gps_test2 程序：

```
gps_test2
```

结果如下：

```
[root@Real6410 /]# gps_test2
Default baudrate is 9600 bps. If not, please enter baudrate as a parameter
read modem
GPGSA,A,1,,,,,,,,,,,,,*1E
DATE     : 0-00-00
TIME     :  00:00:00
Latitude :     0.0000
Longitude:     0.0000
high     :     0.0000
STATUS   :
$GPGSV,3,1,12,20,00,000,,10,00,000,$GPGGA,003833.057,0000.0000,N,00000.0000,E,0,00,,0.0,M,0.0,M,,0000*4A
DATE     : 0-00-00
TIME     :  00:00:00
Latitude :     0.0000
Longitude:     0.0000
high     :     0.0000
STATUS   :
$GPRMC,003833.057,V,0000.0000,N,00000.0000,E,,,280908,,,N*74
```

这样就输出当前的经纬度信息。

第八章 Qtopia-2.2.0 使用教程

8.1 Qtopia-2.2.0 的配置和交叉编译

我们提供的 qtopia-2.2.0 源码包位于 linux\linux-source\qtopia-2.2.0-Real6410.tar.bz2,解压这个源码包到 Ubuntu 的主目录下，得到 qtopia-2.2.0-Real6410，进入 qtopia-2.2.0-Real6410 目录运行以下配置：

echo yes | ./configure -qte '-embedded -no-xft -qconfig qpe -depths 16,32 -system-jpeg -qt-zlib -qt-libpng -gif -no-g++-exceptions -no-qvfb -xplatform linux-arm-g++ -tslib' -qpe 'edition pda -displaysize 480x272 -fontfamilies "helvetica fixed micro smallsmooth smoothtimes unifont" -xplatform linux-arm-g++ -luuid' -qt2 '-no-opengl -no-xft' -dqt '-no-xft -thread'

这样便完成了对 qtopia 的配置，这里可以根据实际使用的 LCD 尺寸修改 displaysize 这个参数。接下来输入

```
make
make install
```

便可以完成对 qtopa 的编译和安装，安装结果就在

qtopia-2.2.0-Real6410/qtopia/image/opt/Qtopia

下面，为了简化这个过程，我们提供了一个脚本 build 完成上面的几个步骤，在该目录下运行./build 即可。

注意：有些用户反映在 Ubuntu-9.10 下编译 qtopia-2.2.0 失败，我也相应地做了一下试验，的确有这个问题出现，这是由于主机环境 Ubuntu 中有相当多的库函数没有预装，我在编译 qtopia-2.20 的时候使用的是 Redhat AS4 的完全安装版，因此编译过程没有出现问题。建议要进行 qtopia-2.2.0 编译的最好切换到 Redhat AS4 中进行。而在 Ubuntu 中编译 QT4 是没有问题的。

8.2 Qtopia-2.2.0 在开发板上的运行

上一步得到了 qtopia-2.2.0-Real6410/qtopia/image/opt/Qtopia/opt 这个结果，这里面实际上包含了所有的应用程序。接着就要把这些代码放到板子的根文件系统中，这里不建议初学的读者从零开始建立一个根文件系统，而直接使用我们在光盘中提供的

linux\linux-source\qtopia.tar.gz

在 Ubuntu 主目录下新建一个目录 rootfs_qtopia，并把上面的源码包解压这个目录，这样讲看到这个目录下有一个 opt 目录，这实际上就是编译 qtopia 后得到的 opt 目录，因此，直接把 qtopia 编译得到的 qtopia-2.2.0-Real6410/qtopia/image/opt 整体复制到 rootfs_qtopia 目录下代替原来的 opt 目录，这样自己的 qtopia 就算安装好了。

接着就要设置好 Qtopia 的运行环境变量，环境变量的设置如下：

```
export TSLIB_TSDEVICE=/dev/input/event1
export TSLIB_CONFFILE=/usr/local/etc/ts.conf
export TSLIB_PLUGINDIR=/usr/local/lib/ts
```

```
export TSLIB_CALIBFILE=/etc/pointercal
export QTDIR=/opt/Qtopia
export QPEDIR=/opt/Qtopia
export PATH=$QTDIR/bin:$PATH
export LD_LIBRARY_PATH=$QTDIR/lib:/usr/local/lib:$LD_LIBRARY_PATH
export QWS_MOUSE_PROTO="TPanel:/dev/input/event1"
# USB:/dev/input/mice"
export QWS_KEYBOARD=TTY:/dev/tty1
export KDEDIR=/opt/kde
export HOME=/root
exec $QPEDIR/bin/qpe
```

我们已经在板子的根文件系统的/bin/下提供了一个名为 qtopia 的可执行脚本来设置上面的参数，并启动 qtopia。默认下这个脚本会在初始化的时候被执行，用户也可以通过运行 qtopia 这个命令来执行这个脚本。

第九章 Qt4 使用教程

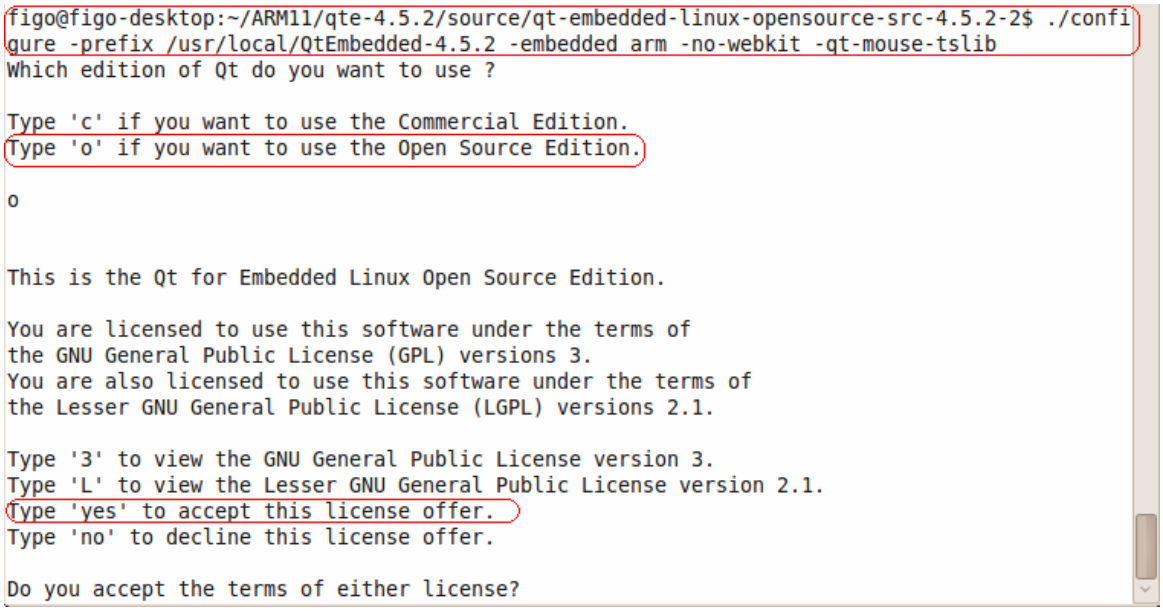
这里使用的是 Qte-4.5.2。Qt 的开发公司奇趣科技在被诺基亚收购后，Qt 代码中就融入了很多诺基亚的风格，相比之前的风格显得更为时尚和人性化，而且更加适合用于手机等移动设备，因此在进行 GUI 的选择的时候不妨考虑一下，这里就是提供一个基于 ARM11 的评估包。

9.1 配置：

解压光盘中的 linux\linux-source\ qt-embedded-linux-opensource-src-4.5.2.tar.gz 到 Ubuntu 下，在终端中进入该目录，并输入以下命令进行 qt4 的配置：

```
./configure -prefix /usr/local/QtEmbedded-4.5.2 -embedded arm -no-webkit -qt-mouse-tslib
```

这时候会要求选择版本和是否接受 license，这里输入 o 和 yes 即可：



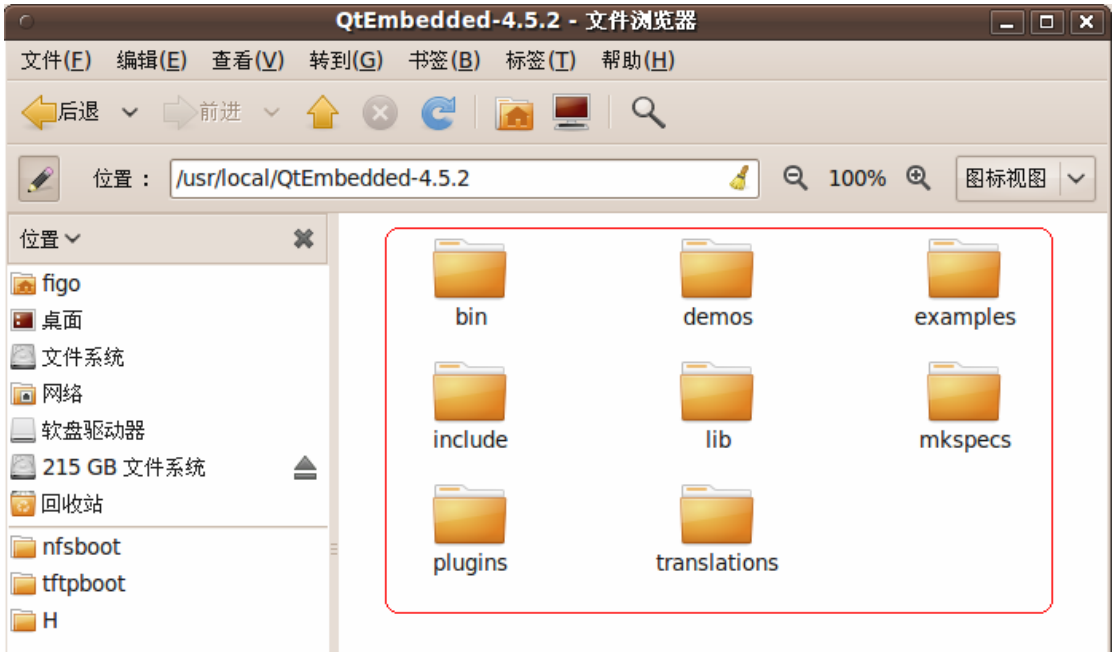
9.2 编译、安装：

这里默认使用 arm-linux-编译器，光盘中提供的 eabi-4.3.2 中已经做了 arm-linux-的软连接，把该编译器的路径添加到 PATH 环境变量即可得到 arm-linux 前缀的 EABI-4.3.2 编译器。在设置好编译器后进行编译：

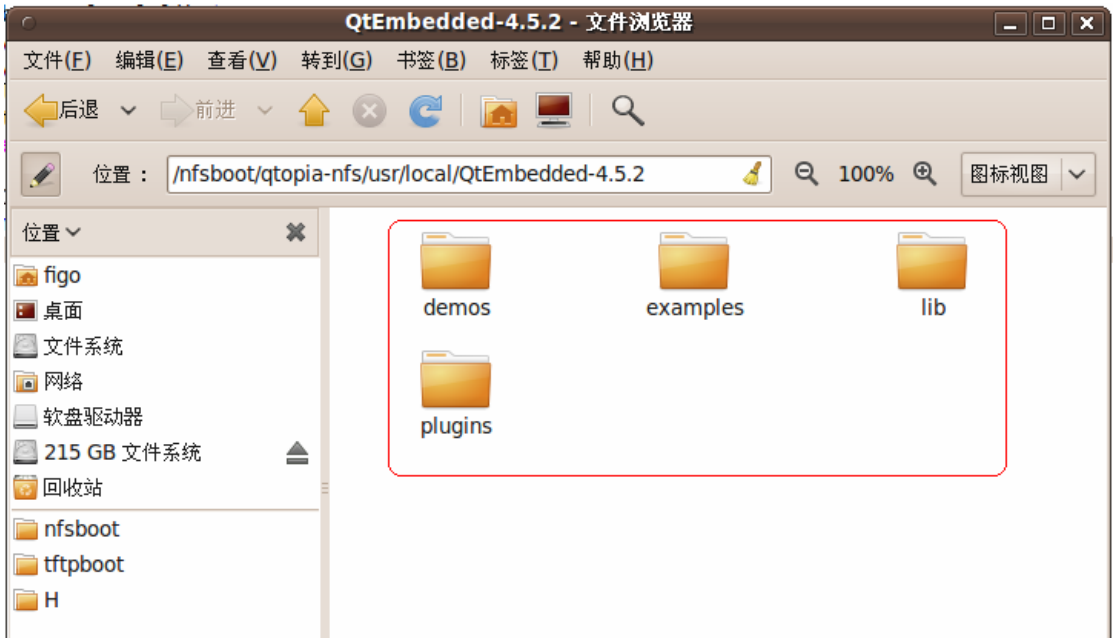
```
make
sudo make install
```

漫长的编译结束后将在主机（Ubuntu）下的/usr/local/QtEmbedded-4.5.2 得到编译后的结果。不过这个编译后的目录内保护了很多不需要放到板子上的内容，可以这个目录下的部分

目录内容进行删减后，如下面的 2 个图所展示的那样。另外即使按照下图进行删减后仍然有相当部分文件可以删除的，不过这里就留给读者去完成了。



删减之前的目录如上图。



删减后的目录如上图。

9.3 设置运行时的环境变量

上面步骤之后已经得到了可用的 Qte，把整个 QtEmbedded-4.5.2 目录复制到板子根目录的/usr/local/路径下。接着在板子的/usr/local/QtEmbedded-4.5.2 中添加脚本文件 setenv-arm.sh，添加如下内容:

```
export QTDIR=/usr/local/QtEmbedded-4.5.2
export QPEDIR=/usr/local/QtEmbedded-4.5.2
export PATH=$QTDIR/bin:$PATH
export LD_LIBRARY_PATH=$QTDIR/lib:/usr/local/lib:$LD_LIBRARY_PATH
export TSLIB_TSDEVICE=/dev/input/event1
export TSLIB_CONFFILE=/usr/local/etc/ts.conf
export TSLIB_PLUGINDIR=/usr/local/lib/ts
export TSLIB_CALIBFILE=/etc/pointercal
export QWS_MOUSE_PROTO="TSLIB:/dev/input/event1 USB:/dev/input/mice"
export QWS_SIZE='480x272'
export QWS_KEYBOARD=TTY:/dev/tty1
```

不过为了防止权限不足的问题，在主机上先给与这个文件可执行属性：

```
sudo chmod 777 setenv-arm.sh
```

注意红色部分字体的设置是非常重要的，主要是指定触摸屏的工作参数。最后一句是自动**运行**特定的程序，保存退出，并进**入板子** shell 界面执**行**这个脚本：

```
cd /usr/local/QtEmbedded-4.5.2
./setenv-arm.sh
```

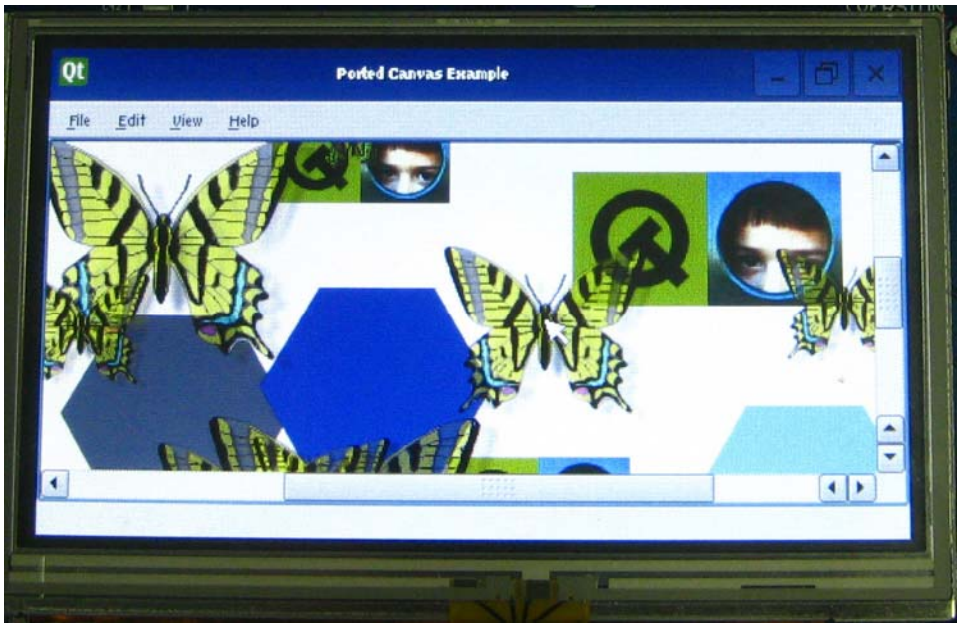
接着便可以运行一下自带的那些测试程序了：

```
/usr/local/QtEmbedded-4.5.2/examples/graphicsv
iew/collidingmice/collidingmice -qws -fn wenquanyi
```

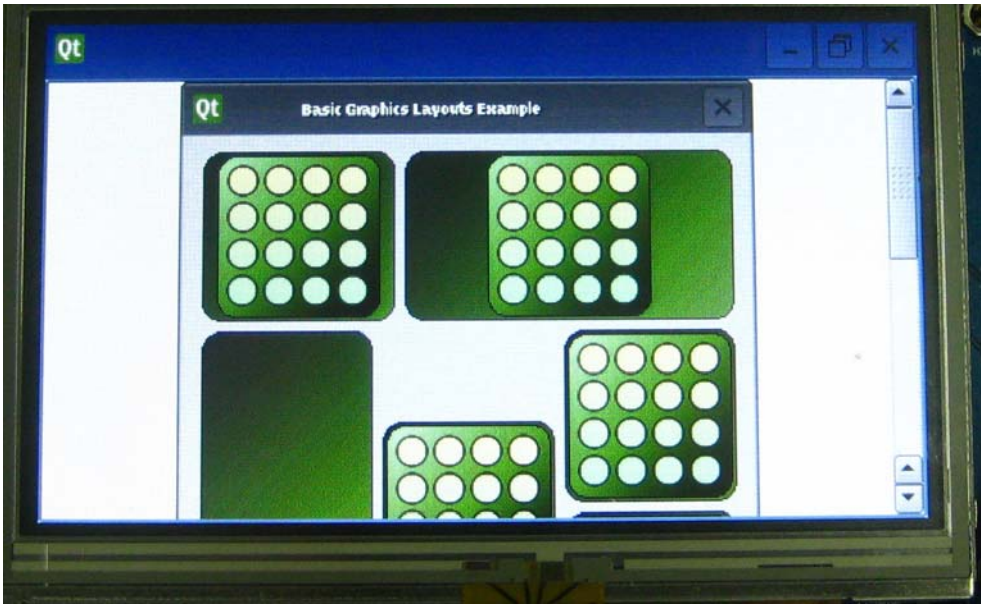
注意：需要提醒用户注意的是，qtopia-2.2.0 的运行和 Qte-4.5.2 的运行需要设置不同的环境变量，因此最好保证两者没有同时运行。如果系统中已经运行了 qtopia-2.2.0，请使用 kill 命令将 qtopia 相关的进程都杀死，kill 命令的使用请参考网上的相关资源。

便能在 LCD 上看到一个漂亮的动画了：

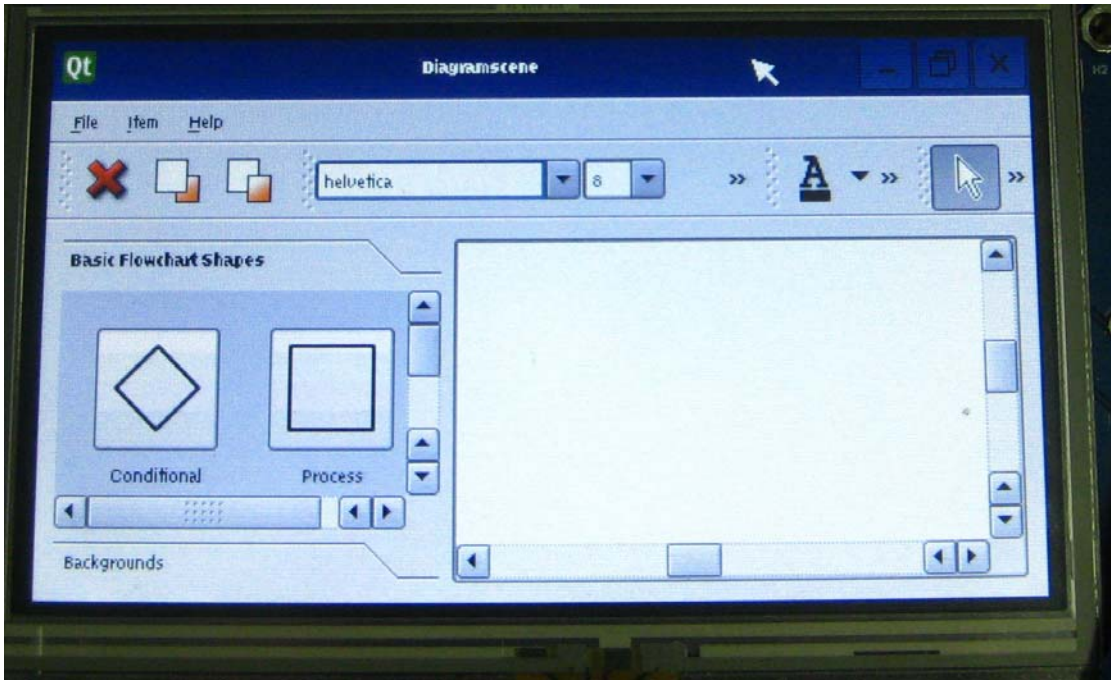
此外，这个包中的 demos、examples 目录下还有很多的测试程序可以运行，这里再贴几张图给读者：



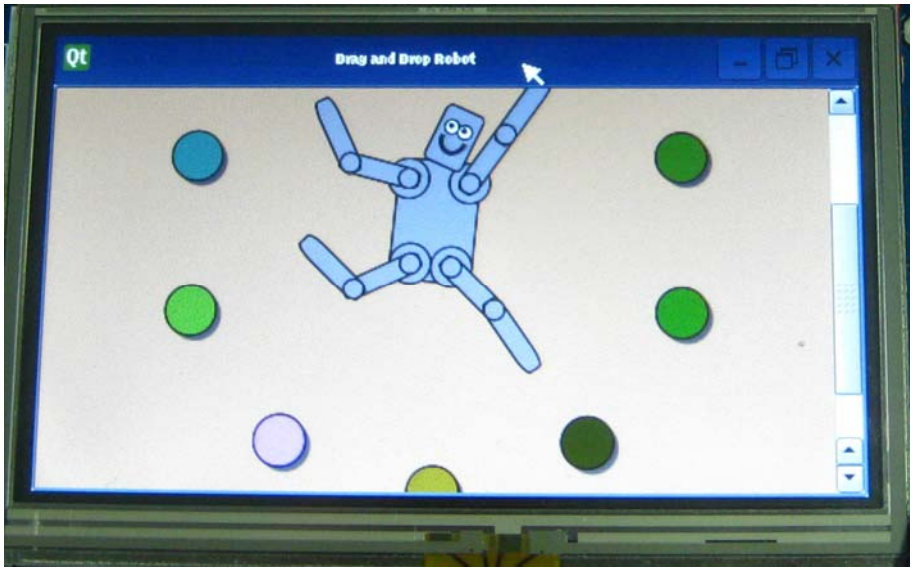
Ported Canvas examples



Basic Graphics layout examples



Diagramscene example



第十章 根文件系统的制作

10.1 cramfs 根文件系统的制作

在我们的板子上，Cramfs 文件系统用来烧写 ubifs，不过也可以用这个文件系统来做一些常规的测试，这里着重说明这个文件系统的制作方式。

首先建议使用我们提供的 mkcramfs 工具，因为官方的版本中有最大文件不超过 8M 的限制。

光盘中的 linux\linux-source\root_mkfs.tar.gz 就是我们所使用的 cramfs 根文件系统包，解压这个包到 Ubuntu 的主目录下，接着运行下面的命令来制作 cramfs：

```
mkcramfs ~/ root_mkfs rootfs.cramfs
```

这样就产生了一个可用的 cramfs，把这个文件烧到 cramfs 即可启动。

10.2 ubifs 根文件系统的制作

Ubifs 格式的镜像暂时还不能像 cramfs 一样制作成镜像文件在 uboot 中烧写进去，而只能制作成一个压缩包，再用 cramfs 中的解压缩工具将这个压缩包解压到 Nandflash 的 ubifs 分区。所以 ubifs 根文件系统的制作实际上就是一个压缩包的制作。

下面以第八章制作好的 rootfs_qtopia 为例讲述如何制作这个压缩包。

首先要在终端中进入 rootfs_qtopia 目录下面：

```
cd ~/rootfs_qtopia     #这里根据自己的实际情况进入 rootfs_qtopia 目录下，这步必须做
```

接下来运行下面命令进行压缩：

```
tar –zcvf qtopia.tar.gz *
```

这样便能在该目录下得到一个可用的 qtopia.tar.gz 了，按照第五章的说明把这个文件烧写到 ubifs 分区即可。

第十一章 多媒体硬件编解码测试

S3C6410 提供了包括对 H264、H263、TV out 等多媒体的硬编解码的支持，使得 S3C6410 在多媒体处理上的性能比一般的软编解码处理器性能高出一截。我们提供了对于这部分功能的测试程序，下面开始介绍如何使用这个测试程序。

该测试程序位于板子根文件系统的/usr/local/multimedia_test_real6410 下，在运行该程序之前最好保证 qtopia-2.2.0 或者 qt4 没有在运行，不然可能出现资源冲突之类的问题。如果这些程序已经在运行，可以在命令行界面中通过 kill 命令来杀死这些进程，该指令的使用可以参考网上相关资料的说明。

测试程序代码位于光盘中的 linux\linux-source\ multimedia_test_real6410.tar.bz2，将此文件复制到 Ubuntu 下的主目录解压即可得到 multimedia_test_real6410 目录。

11.1.配置编译运行 multimedia_test_real6410

11.1.1.配置 multimedia_test_real6410

1.我们提供的 multimedia_test_real6410 测试程序支持多种 LCD 尺寸，包括 4.3”、5”、7”等，用户可以根据自己 LCD 的实际情况对这个尺寸进行定义，定义方式是修改 multimedia_test_real6410 根目录下的 Makefile 文件中的下面的部分：

env.txt | Makefile

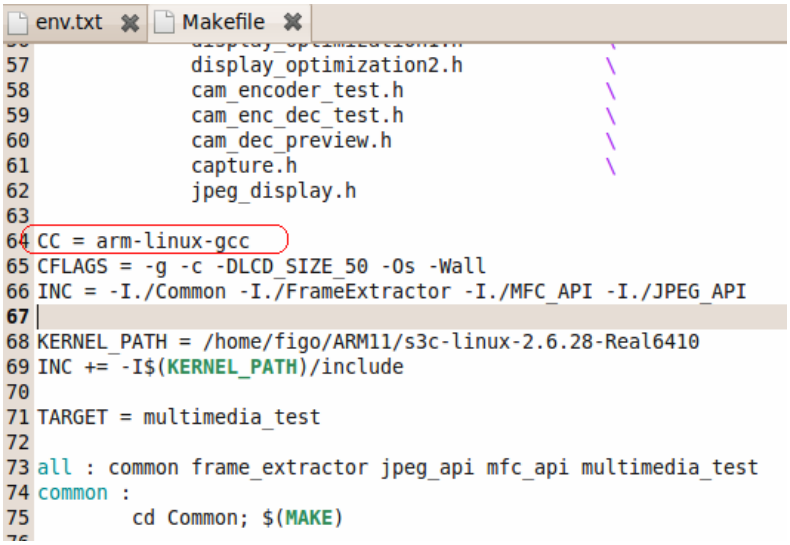
```
7               display_optimization2.h             \
8               cam_encoder_test.h                  \
9               cam_enc_dec_test.h                  \
0               cam_dec_preview.h                   \
1               capture.h                           \
2               jpeg_display.h
3
4 CC = arm-linux-gcc
5 CFLAGS = -g -c -DLCD_SIZE_50 -Os -Wall
6 INC = -I./Common -I./FrameExtractor -I./MFC_API -I./JPEG_API
7
8 KERNEL_PATH = /home/figo/ARM11/s3c-linux-2.6.28-Real6410
9 INC += -I$(KERNEL_PATH)/include
0
1 TARGET = multimedia_test
2
3 all : common frame_extractor jpeg_api mfc_api multimedia_test
4 common :
5         cd Common; $(MAKE)
6
7 frame_extractor :
8         cd FrameExtractor; $(MAKE)
```

如红色部分字体所示，根据自己的实际情况做相应的修改：

- 如果使用的是 5”的 LCD，那么红色部分编辑为-DLCD_SIZE_43
- 如果使用的是 5”的 LCD，那么红色部分编辑为-DLCD_SIZE_50
- 如果使用的是 5”的 LCD，那么红色部分编辑为-DLCD_SIZE_70（默认是 7”）

2.修改完尺寸后，接着要修改的是交叉编译器，同样是修改 Makefile，不过需要修改该测试包下的多个 Makefile，这里就不一一说明每个 Makefile 的修改了，只说明根目录下的

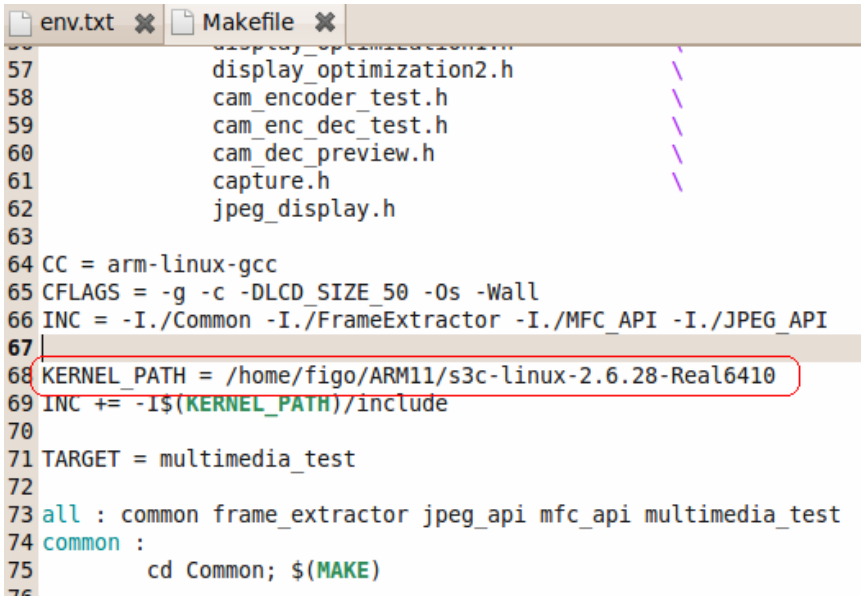
Makefile 的修改：



如上图的红色圈部分，根据自己的实际情况设置好交叉编译器，因为我已经将该编译器的路径添加到 PATH 变量中，所以只要把上面的 CC 赋值为 arm-linux-gcc 即可。如果没有添加到环境变量需要填写完整的路径。

其他目录下的 Makefile 也要做类似的修改，这里不进一步说明。

3.最后一个要修改的地方就是内核路径，只需要在测试程序根目录下的 Makefile 中进修改即可：



如上图，按照自己的实际情况修改为自己的实际路径即可。

11.1.2.编译 multimedia_test_real6410

上面的工作做完后，在终端中进入该测试目录下运行 make 即可完成测试程序的编译，在根目录下得到的 multimedia_test 便是所需要的测试程序。

11.1.3.运行 multimedia_test_real6410

把上一步得到的 multimedia_test 文件和相同目录下的 TestVectors 文件夹一起复制到板子文件系统的/usr/local/multimedia_test_real6410 目录下。

从新制作文件系统并烧入板子后进入该目录，并运行这个程序：

```
cd /usr/local/multimedia_test_real6410
./ multimedia_test
```

```
[root@Real6410 /]# cd usr/local/multimedia_test_real6410/
[root@Real6410 multimedia_test_real6410]# ./multimedia_test
======= Real6410 Media Demo Application ========
=                                              =
=  1.   H.264 display                          =
=  2.   MPEG4 display                          =
=  3.   H.263 display                          =
=  4.   VC-1  display                          =
=  5.   4-windows display                      =
=  6.   Camera preview & MFC encoding          =
=  7.   MFC decoding & Camera preview          =
=  8.   Camera input and JPEG encoding         =
=  9.   JPEG decoding and display              =
=  10.  H.264 decoding thru TVOUT              =
=  11.  MFC decoding & Camera preview thru TV  =
=  12.  Exit                                   =
=                                              =
================================================
Select number -->
```

11.2.H264 硬解码测试

选择 1：

```
=  8.   Camera input and JPEG encoding         =
=  9.   JPEG decoding and display              =
=  10.  H.264 decoding thru TVOUT              =
=  11.  MFC decoding & Camera preview thru TV  =
=  12.  Exit                                   =
=                                              =
================================================
Select number --> 1
======= H.264 File Decodec Test ========
        ########<STREAMINFO> width=320   height=240.

[1. H.264 display]
Using IP          : MFC, Post processor, LCD
Input filename    : wanted.264
Input vector size : VGA(640x480)
Display size      : WVGA(800x480)
Bitrate           : 971 Kbps
FPS               : 30
```

测试截图如下：

11.3.MPEG4 硬解码测试

选择 1：

```
=  6.   Camera preview & MFC encoding            =
=  7.   MFC decoding & Camera preview            =
=  8.   Camera input and JPEG encoding           =
=  9.   JPEG decoding and display                =
=  10.  H.264 decoding thru TVOUT                =
=  11.  MFC decoding & Camera preview thru TV    =
=  12.  Exit                                     =
=                                                =
================================================
Select number --> 2

[2. MPEG4 display]
Using IP          : MFC, Post processor, LCD
Input filename    : shrek.m4v
Input vector size : QVGA(320x240)
Display size      : WVGA(800x480)
Bitrate           : 482 Kbps
FPS               : 24
```

测试截图如下：

11.4.H263 硬解码测试

选择 1：

```
=  6.   Camera preview & MFC encoding            =
=  7.   MFC decoding & Camera preview            =
=  8.   Camera input and JPEG encoding           =
=  9.   JPEG decoding and display                =
=  10.  H.264 decoding thru TVOUT                =
=  11.  MFC decoding & Camera preview thru TV    =
=  12.  Exit                                     =
=                                                =
================================================
Select number -->3

[3. H.263 display]
Using IP          : MFC, Post processor, LCD
Input filename    : iron.263
Input vector size : QVGA(320x240)
Display size      : WVGA(800x480)
Bitrate           : 460 Kbps
FPS               : 30
```

测试截图如下：

11.5.VC-1 硬解码测试

选择 1：

```
=  6.   Camera preview & MFC encoding            =
=  7.   MFC decoding & Camera preview            =
=  8.   Camera input and JPEG encoding           =
=  9.   JPEG decoding and display                =
=  10.  H.264 decoding thru TVOUT                =
=  11.  MFC decoding & Camera preview thru TV    =
=  12.  Exit                                     =
=                                                =
================================================
Select number --> 4

[4. VC-1 display]
Using IP          : MFC, Post processor, LCD
Input filename    : test2_0.rcv
Input vector size : QVGA(320x240)
Display size      : WVGA(800x480)
Bitrate           : 460 Kbps
FPS               : 30
```

测试截图如下：

11.6.四窗口显示测试

选择 1：

```
================================================
Select number --> 5
====== 4 Windows muti-Format decode Test ======

[4-windows display]
Using IP : MFC, Post processor, LCD
*************************************************************************
*                               *                                       *
* Frame buffer      : 0         * Frame buffer      : 1                 *
* Codec             : H.264     * Codec             : MPEG4             *
* Input filename    : veggie.264* Input filename    : shrek.m4v         *
* Input vector size : QVGA      * Input vector size : QVGA              *
* Display size      : 400x240   * Display size      : 400x240           *
* Bitrate           : 460 Kbps  * Bitrate           : 482 Kbps          *
* FPS               : 30        * FPS               : 24                *
*                               *                                       *
*************************************************************************
*                               *                                       *
* Frame buffer      : 2         * Frame buffer      : 3                 *
```

测试截图如下：

11.7.摄像头预览与 MFC 硬编码测试

选择 1：

```
=  8.   Camera input and JPEG encoding          =
=  9.   JPEG decoding and display               =
=  10.  H.264 decoding thru TVOUT               =
=  11.  MFC decoding & Camera preview thru TV   =
=  12.  Exit                                    =
=                                               =
================================================
Select number --> 6
==== Camera Preview & Encode to H264 Test ====
V4L2 : Camera Input(V4L2_INPUT_TYPE_CAMERA )
s3c-fimc: external camera initialized

[8. Camera preview & MFC encoding]
Using IP          : MFC, Post processor, LCD, Camera
Display size      : VGA(640x480)

e : Encoding
x : Exit
Select ==>
```

测试截图如下：

11.8.MFC 硬解码与摄像头预览测试

选择 1：

```
=  8.   Camera input and JPEG encoding          =
=  9.   JPEG decoding and display               =
=  10.  H.264 decoding thru TVOUT               =
=  11.  MFC decoding & Camera preview thru TV   =
=  12.  Exit                                    =
=                                               =
================================================
Select number -> 7
====== Camera Preview & Decode Test ======

VIDIOC_ENUMINPUT = -1068476902, ret = 0
[9. MFC decoding & Camera preview]
Using IP            : MFC, Post processor, LCD, Camera
Camera preview size : QVGA(320x240)
Display size        : WVGA(800x480)

e : Encoding
x : Exit
Select ==>
```

测试截图如下：

11.9.摄像头输入与 JPEG 硬编码测试

选择 1：

```
=  8.   Camera input and JPEG encoding           =
=  9.   JPEG decoding and display                =
=  10.  H.264 decoding thru TVOUT                =
=  11.  MFC decoding & Camera preview thru TV    =
=  12.  Exit                                     =
=                                                =
==========================================
Select number -> 8
====== Camera Preview & JPEG Encode ======

VIDIOC_ENUMINPUT = 0
[11. Camera input & JPEG encoding]
Using IP            : Post processor, LCD, Camera, JPEG
Camera preview size : VGA(640x480)
Capture size        : VGA(640x480)

c : Capture
x : Exit
Select ==>
```

测试截图如下：

11.10.JPEG 解码与显示测试

选择 1：

```
=                                                 =
=  1.   H.264 display                             =
=  2.   MPEG4 display                             =
=  3.   H.263 display                             =
=  4.   VC-1  display                             =
=  5.   4-windows display                         =
=  6.   Camera preview & MFC encoding             =
=  7.   MFC decoding & Camera preview             =
=  8.   Camera input and JPEG encoding            =
=  9.   JPEG decoding and display                 =
=  10.  H.264 decoding thru TVOUT                 =
=  11.  MFC decoding & Camera preview thru TV     =
=  12.  Exit                                      =
=                                                 =
=================================================
Select number -> 9
======= JPEG decodec Test ========
======= Real6410 Media Demo Application ========
=                                                 =
```

测试截图如下：

11.11.H264 解码与 TV 输出测试

选择 1：

```
=  12.  Exit                                            =
=                                                       =
=======================================================
Select number -> 10
        <STREAMINFO> width=320   height=240. s3c_tvscaler_init
TV-OUT: VIDIOC_ENUMINPUT : index = 1
TV-OUT: VIDIOC_S_INPUT
TV-OUT: VIDIOC_ENUMOUTPUT : index = 0
TV-OUT: VIDIOC_S_OUTPUT
C: VIDIOC_S_FMT
TV-OUT: MFC path operation set

Device file open
V4L2 APPL : Name of the interface is S3C TV-OUT driveTVENCODER
V4L2 APPL : [1]: IN channel name Memory input (MSDMA)
V4L2 APPL : DMA INPUT
V4L2 APPL : [0]: OUT channel name TV-OUT
V4L2 APPL : TV OUT
```

这部分由于手头上没有 TV 所以没办法看到实际效果，请有 TV 的读者自行测试。

11.12.通过 TV 输出的 MFC 解码与摄像头预览

选择 1：

```
=                                                 =
===================================================
Select number -> 11
=======Camera Preview & TVOUT Test ======== s3c_tvscaler_init
TV-OUT: VIDIOC_ENUMINPUT : index = 1
TV-OUT: VIDIOC_S_INPUT
TV-OUT: VIDIOC_ENUMOUTPUT : index = 0
TV-OUT: VIDIOC_S_OUTPUT
C: VIDIOC_S_FMT
TV-OUT: LCD path operation set

V4L2 APPL : Name of the interface is S3C TV-OUT driveTVENCODER
V4L2 APPL : [1]: IN channel name Memory input (MSDMA)
V4L2 APPL : DMA INPUT
V4L2 APPL : [0]: OUT channel name TV-OUT
V4L2 APPL : TV OUT

[13. MFC decoding & Camera preview]
Using IP                    : MFC, Post processor, LCD, Camera, TV scaler/encoder
```

这部分由于手头上没有 TV 所以没办法看到实际效果，请有 TV 的读者自行测试。

附录一 Linux 启动画面的制作

在做嵌入式 linux 系统时常常希望能自己制作液晶的启动画面，这里将讲述 logo 的自定义方法。

linux 下一般图片使用 PNG 格式比较多，这里就以 PNG 格式为例，不是 PNG 格式的图片，可以用 GIMP 转换一下：
首先将 png 图片转成 pnm
pngtopnm utulinux_logo.png > utulinux_logo.pnm
然后将 pnm 图片的颜色数限制在 224
pnmquant 224 utulinux_logo.pnm > utulinux_logo_224.pnm
最后将 pnm 图片转换成我们需要的 ppm
pnmtoplainpnm utulinux_logo_224.pnm > utulinux_logo_224.ppm

然 utulinux_logo_224.ppm 替换 linuxsrc/drivers/video/logo 中对应的图像就 OK 了。

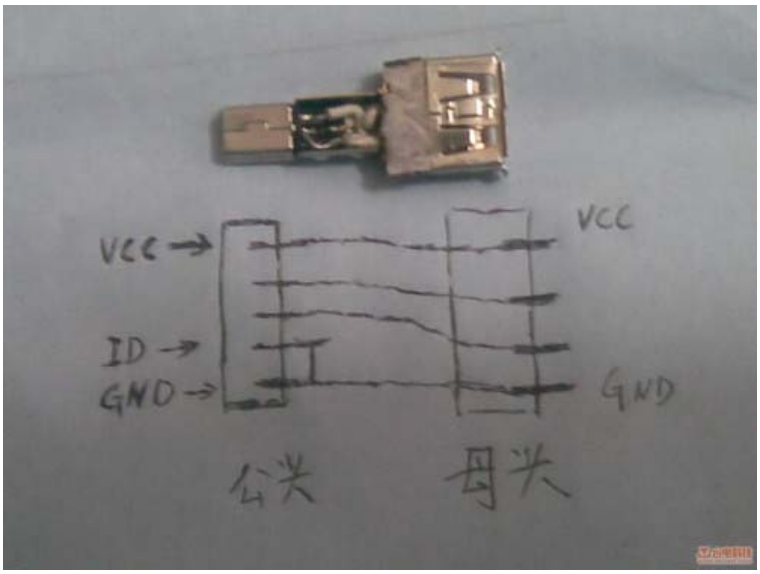
在 suse10.2 下没有这些工具，但是可以使用 GIMP 完成所有工作：
1、在右键的 image/mode/indexed..中修改最大颜色数为 224
2、保存为 ppm 格式，选择 asiic 存放格式
3、代替 logo 目录下相应图形即可

注意：可以使用 ACD 将 windows 常用图片格式转换为 png 格式

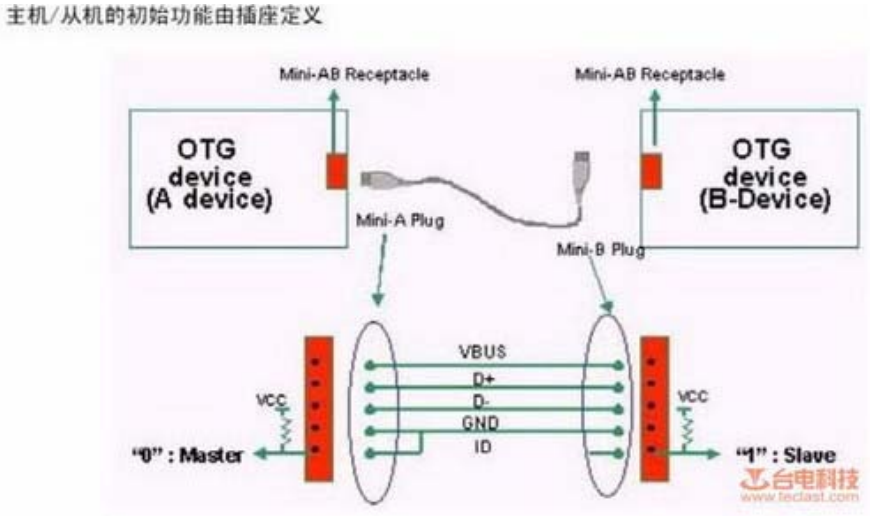
附录二 DIY 一根 OTG 线

首先，你需要两个 usb 接头，一个 usb 大头的母头（被 U 盘插），一个小头的 T 型头（插 M50），然后网上找了两张图，可以看到，公头的那边有五个触点，母头只有四个，因为公头那边有一个 id 端，用于主机判断电平，高则普通 usb 状态，低则 usbhost 状态，所以我们将它接地，让它低电平，就是 usbhost 状态了，接下来就是看图连线了，会用烙铁就可以了，还有就是连接线尽量短，我之前失败了就是由于连线太长，信号衰减太大，连接失败！记得别接错了，有一张图是我画的，从那个角度看过去，两个接头的线正好顺序连接，直接对过去连，注意 id 线怎么连。

（你的机器如果无法连接，有可能是机器的 otg 供电电路没有加上，请外接电源试试，我只能保证 21 周以及以后生产的机器有 otg 供电,我的 sn:2183M092190065）



主机/从机的初始功能由插座定义



附录三 移植 qtopia-free-2.2.0

5.5.1.1 下载源码

到官方网站下载 qtopia-2.2.0 源代码包：
http://www.qtopia.org.cn/ftp/mirror/ftp.trolltech.com/qt/source/
解压源代码包到目录/root 下，目录为/root/qtopia-free-2.2.0

5.5.1.2 修改源码

A．修改文件
qtopia-free-2.2.0/qtopia/mkspecs/qws/linux-arm-g++/qmake.conf
将此行
QMAKE_LIBS_QT = -lqte
修改为
QMAKE_LIBS_QT= -lqte -lpng -lts -lz -luuid -ljpeg

B．如果想让 Qtopia 自己支持触摸屏，需要修改 qtopia-free-2.2.0/qtopia/src/**qt**/qconfig-qpe.h 文件，定义相应的宏，在最后加上
#define QT_QWS_IPAQ
#define QT_QWS_IPAQ_RAW
在板子运行的时候，加上如下环境变量：
exportQWS_MOUSE_PROTO=TPanel:/dev/h3600_tsraw（触摸屏具体名称和位置可能需要根据实际情况修改）
export QWS_MOUSE_PROTO=USB:/dev/input/mice

C。如果想让 Qtopia 支持鼠标，需要修改$QPEDIR/src/**qt**/qconfig-qpe.h 文件
注释如下部分：
/*
#ifndef QT_NO_QWS_CURSOR
#define QT_NO_QWS_CURSOR
#endif
#ifndef QT_NO_QWS_MOUSE_AUTO
#define QT_NO_QWS_MOUSE_AUTO
#endif
#ifndef QT_NO_QWS_MOUSE_PC
#define QT_NO_QWS_MOUSE_PC
#endif
*/
如果想让 Qtopia 支持 USB 标准键盘，在板子运行的时候，加上如下环境变量：
export QWS_KEYBOARD=USB:/dev/input/event0

5.5.1.3 设置环境变量

打开一个终端并进入/root/qtopia-free-2.2.0 目录下，分别输入以下命令设置好各个环境变量：

export QPEDIR=$PWD/qtopia
export QTDIR=$PWD/qt2
export LD_LIBRARY_PATH=$QTDIR/lib/$QPEDIR/lib
export TMAKEDIR=$PWD/tmake
export TMAKEPATH=$TMAKEDIR/lib/qws/linux-arm-g++

5.5.1.4 准备配置文件

cp $QPEDIR/src/**qt**/qconfig-qpe.h $QTDIR/src/tools
cd $QPEDIR/src/libraries/qtopia
cp custom-linux-ipaq-g++.cpp custom-linux-arm-g++.cpp
cp custom-linux-ipaq-g++.h custom-linux-arm-g++.h

5.5.1.5 开始 configure

cd /root/qtopia-free-2.2.0
echo yes | ./configure -qte '-embedded -no-xft -qconfig qpe -depths 16,24 -system-jpeg -qt-zlib -qt-libpng -gif -no-g++-exceptions -no-qvfb -xplatform linux-arm-g++ -tslib' -qpe 'edition pda -displaysize 480x272 -fontfamilies "helvetica fixed micro smallsmooth smoothtimes unifont" -xplatform linux-arm-g++ -luuid' -qt2 '-no-opengl -no-xft' -dqt '-no-xft -thread'
之后输入 make、make install 后便可以在/root/qtopia-free-2.2.0/qtopia/image/opt/Qtopia 得到编译后的结果，把 Qtopia 目录复制到板子的根目录下，并重新命名为 qtopia-2.2.0。
需要提醒的是，在编译过程中可能会出现段错误之类的提示，只要重新编译即可。
对于上面的过程已经制作成一个脚本 build，在解压缩我们提供的 qtopia-free-2.2.0 后便能在这个目录下找到这个脚本，执行这个脚本就能完成环境变量的设置和配置编译了。

5.5.1.6 运行 qtopia

在运行 qtopia 之前需要设置多个环境变量：
export QTDIR=/qtopia-2.2.0
export QPEDIR=/qtopia-2.2.0
export LD_LIBRARY_PATH=/qtopia-2.2.0/lib
export QWS_MOUSE_PROTO=TPanel:/dev/ts0
export TSLIB_CALIBFILE=/etc/pointercal
export TSLIB_FBDEVICE=/dev/fb0
export TSLIB_CONFFILE=/etc/ts.conf
export HOME=/root
/qtopia-2.2.0/bin/qpe –qws
对于上面的各个环境变量的意义相信见名思意就行，而 HOME 这个变量需要稍微解释一下。Qtopia 的主界面中有 Settings 和 Documents 这 2 项，这 2 项实际上就是分别对应于$HOME/Settings 和$HOME/Documents 目录，默认下 HOME 就是根目录，所以在 qtopia 运行起来后会在根目录下建立这 2 个文件夹，如果要自定义这个目录可以在这里设置。
对于上面的这些命令已经写在了/etc/runqpe 中，并由/etc/rCS 自动执行，不需要用户输入。

5.5.1.7 制作含有 qtopia 的文件系统

在复制编译之后的 qtopia 到需要制作成根文件镜像的 root_qtopia 目录下之后，使用 mkyaffs2image 命令制作成为 yaffs2 镜像，并使用 tftp 下载到板子中即可。

附录四 vmware下安装Ubuntu教程

0、 预备知识

什么是 Ubuntu。如果不了解这一点，本文的内容似乎与您无关，请无视之。

另外，VMware 的虚拟显卡不支持 3D 图形加速，如果想体验 Ubuntu 的 3D 桌面还是真正的装一回 Ubuntu 吧。

1、 安装 VMware

理论上讲，软件与硬件是可以在功能上相互转化的。“虚拟机”就是这样一种以软件手段来模拟硬件的工具。喜欢玩电子游戏的朋友一定知道模拟器的概念吧，没错，说白了虚拟机其实就是模拟器的一种，只不过模拟的是 PC 而已。目前虚拟机已经被广泛用作进行与操作系统、网络等有关的实验，当然如果你喜欢，你也可以装个其他的操作系统运行你现有操作系统上玩不了的游戏^_^

虚拟机之家是国内有规模的虚拟机资讯网站，也许你能从中了解更多有关虚拟机的信息。

我们目前最常用的虚拟机是 VMware Workstation，本文中姑且省略为 VMware，建议安装版本 5.5.3。另有最新版本 6.0.0 比较庞大，很多功能并不是初阶常用到的，就免了。

VMware 在网上到处都可以搜到下载，同时汉化包也比较完备。先下载、安装、注册程序，再打汉化补丁即可，注意小心别同时误装汉化补丁中附带的流氓软件。

2、 下载 Ubuntu

目前 Ubuntu 的最新稳定版本是 7.04，最新稳定的 LTS 版本（Ubuntu 的 “LTS” 版本拥有长期支持，桌面版本为 3 年，服务器版本为 5 年）是 6.06，最新测试版本是 7.10 Tribe 4，开发代号是 Gutsy Gibbon，意思是“勇敢的长臂猿”，汗~

Ubuntu7.04 和 Ubuntu6.06 的下载地址；

Ubuntu7.10 Tribe 4 的下载地址。

另外说一下有关 Ubuntu 各个版本的区别，帮助您确定您要下载哪一个——

A、 按支持的时间划分

普通版：提供 18 个月的在线更新支持；

LTS 版：上面说了，桌面版本提供 3 年，服务器版本提供 5 年的在线更新支持。

B、 按应用划分

桌面版：应用于台式机、笔记本等私人、家用、办公商务等用途；

服务器版：用于服务器，据说没有图形界面。

C、 按安装方式划分

Live CD 版：光盘中是一个完整 Ubuntu 操作系统，通过光盘启动后可以直接进入，从而允许你在安装之前事先进行一番体验。而且安装到硬盘的界面就是基于这个光盘系统的。但由于是光盘启动，运行的效率就打折扣了；

Alternate desktop CD 版：安装界面是文字形式，允许用户做更多的定制工作，运行效率也比 Live CD 高些，不需事先体验的老用户和机器老的用户推荐采用。

P.S. 衍生版本说明

要知道，由 Linus Torvalds 及其合作者开发的 Linux 并不是一个完整的操作系统，而是一个类 Unix（Unix-like）的操作系统内核。我们常说的 Linux 操作系统其实应该被完整地称为 GNU/Linux。

这里的 GNU 是指 Richard Stallman 于 1984 年发起的 GNU（GNU’s Not Unix）计划，它的目标是完成一套基于自由软件的完整操作系统——HURD。该计划的参与者中云集了诸多掌握核心技术的顶尖高手，更重要的是，他们信仰技术上的共产主义。为了保证 GNU 软件可以自由地“使用、复制、修改和发布”，所有 GNU 软件都包含一份被称之为 GNU 通用公共许可证（GNU General Public License，GPL）的协议条款。

Linux 并不是 GNU 计划的一部分。到 1991 年 Linux 的第一个版本公开发行时，GNU 计划已经完成除了 HURD 操作系统内核之外的大部分软件，其中包括了一个壳程序（shell），C 语言程序库以及一个 C 语言编译器。Linus Torvalds 及其合作者加入了这些软件从而完成了 Linux 操作系统，并宣布在 GNU 通用公共许可证（GPL）下发行。

正是由于 Linux 使用了许多 GNU 程序，Richard Stallman 认为应该将该操作系统完整的称为“GNU/Linux”。

我们现在所称的 Linux 系统或 GNU/Linux 系统，实际上包括使用 Linux 内核的若干操作系统发行版本。比较知名的有 Ubuntu、openSUSE、Fedora/Red Hat、Debian 等，它们大都使用 XFree86 或 X.org 服务器作为图像系统，并使用 GNOME 和 KDE 等桌面环境。其中 Ubuntu 系就是基于 Debian 发展出来的一系列 GNU/Linux 发行版本。

Ubuntu：使用 GNOME 桌面环境，这个桌面环境是 GNU 计划的一部分；

Kubuntu：使用 KDE 桌面环境，据说效果比较华丽，但系统开销相对大些；

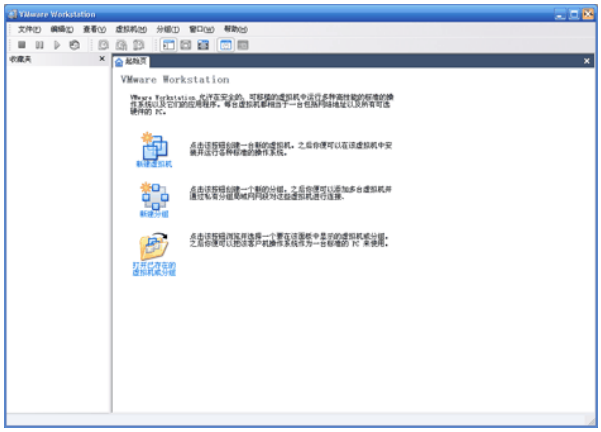
Xubuntu：使用 Xface 桌面环境，比较轻量，适合配置较低的老机器使用；

Edubuntu：同样使用 GNOME 桌面环境，界面风格稍微卡通一点，适合儿童使用，并且集合了很多寓教于乐的软件。

3、 创建虚拟机

这里我采用的环境为 Windows XP SP2 下的 VMware Workstation 5.5.3 汉化版；为了减少在线更新组件的工作量，我选择安装最新的 Ubuntu7.10 Tribe 4 桌面 Live CD 版，网络环境是普通家用宽带（adsl）。

在以后的叙述中我将现实中的真实的计算机称为“宿主机”，将存在于 VMware 中的虚拟计算机称为“虚拟机”。



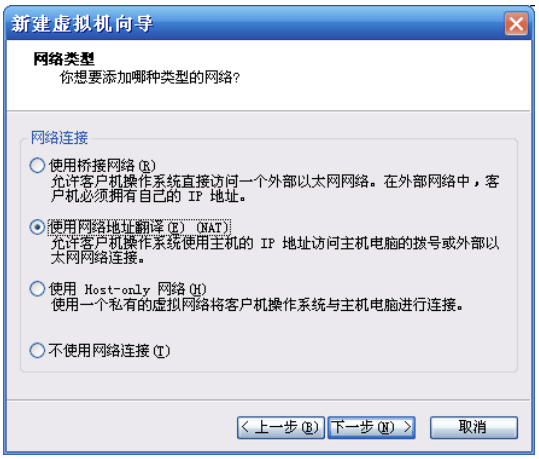
Step1：双击 VMware 图标后展现在我们眼前的是这款虚拟机的初始界面。我们能够看到它的功能十分的强大，如果你愿意，甚至可以创建若干台虚拟电脑并将它连成一个虚拟的网络，当然这需要你的真实的机器足够强劲。在 VMware 初始界面上点击新建虚拟机的图标，会弹出新建虚拟机向导，别犹豫，下一步；

Step2：虚拟机配置——典型——下一步

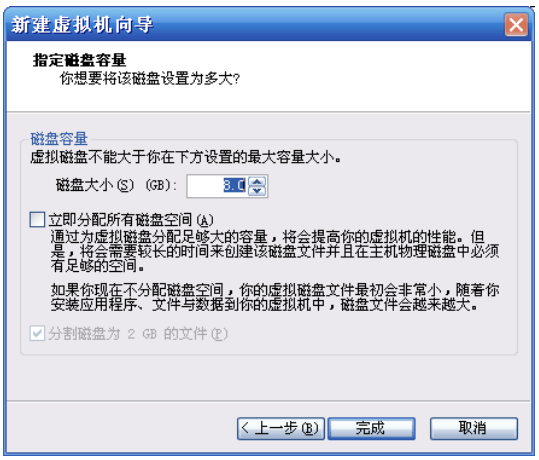
Step3：客户机操作系统——Linux——Ubuntu——下一步

Step4：虚拟机名称、位置——Ubuntu（当然也可以写别的）、默认位置（推荐）——下一步

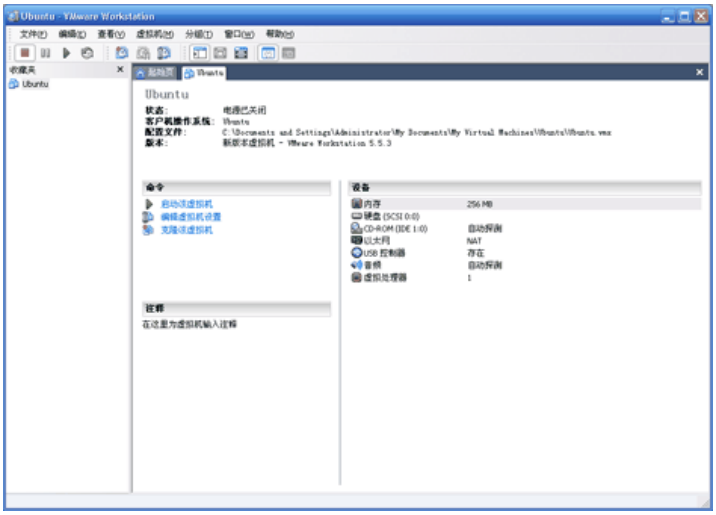
Step5：网络连接——a、如果你有一个外网固定 IP（不是 adsl 随机分配给你一次一变的那种），那么就选择“使用桥接网络（Bridge）”，之后需要在虚拟机的操作系统中进行相应设置才能上网：设置一个与宿主机同网段且未使用的 IP 地址，其余如子网掩码、DNS、网关等与宿主机相同；b、如果你是家庭 adsl 的用户（使用 DHCP 上网），那么就选择“使用网络地址翻译（NAT）”，这样只要宿主机可以上网，虚拟机不用特殊设置（Ubuntu 默认开启 DHCP 服务）就可以共享宿主机的网络。我的网络环境就是 adsl，自然选它；c、如果你硬要将虚拟机与宿主机联局域网，而不接入互联网，就选“使用 host-only 网络”；d、不使用网络连接。可为什么不呢？——下一步



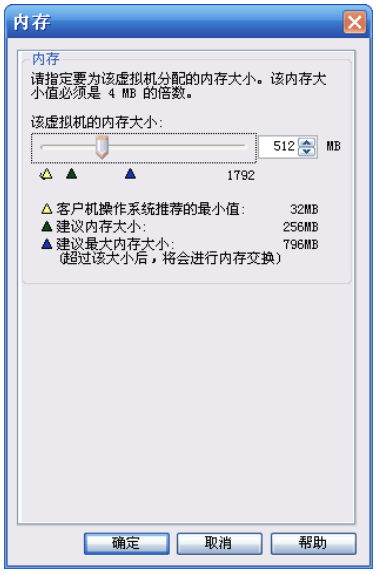
Step6：磁盘容量——默认设置即可——完成



经过上面的步骤，我们已经拥有了一台还没装任何操作系统的 VMware 牌的虚拟 PC 裸机，你可以在设备面板上清晰地看到并编辑它的配置。



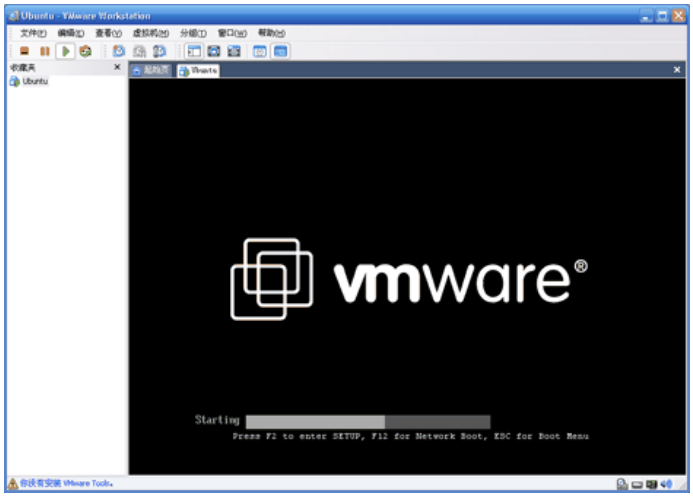
在设备面板上双击任一选项，你都可以对选定对象进行编辑，比如我宿主机的内存是 1G 的，为了运行虚拟机更流畅些，就可以双击设备——内存，把虚拟机内存从缺省的 256M 调高到 512M。



4、 安装 Ubuntu

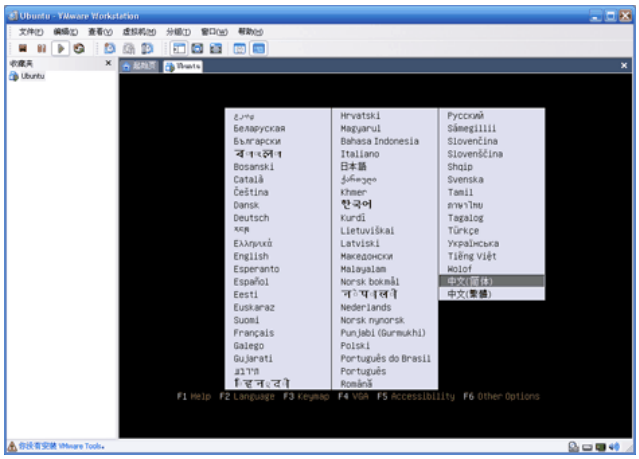
Step1：下载下来的 Ubuntu 是 ISO 格式。我们在设备——CD-ROM 上双击，设置“连接”为“使用 ISO 镜像”，载入 Ubuntu 的光盘镜像，确定。至此我们算是把 Ubuntu 的光盘放到了虚拟机的光驱里。

Step2：点击面板上的命令——启动该虚拟机，开启虚拟机电源。我们看见 VMware 牌 PC 机的开机自检画面。

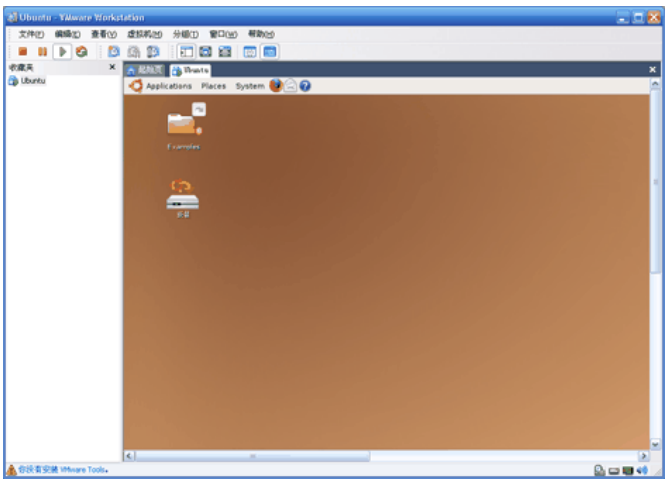


Step3：紧接着，系统由“放到”光驱里的 Ubuntu 安装盘引导，进入 Ubuntu 的安装界面。

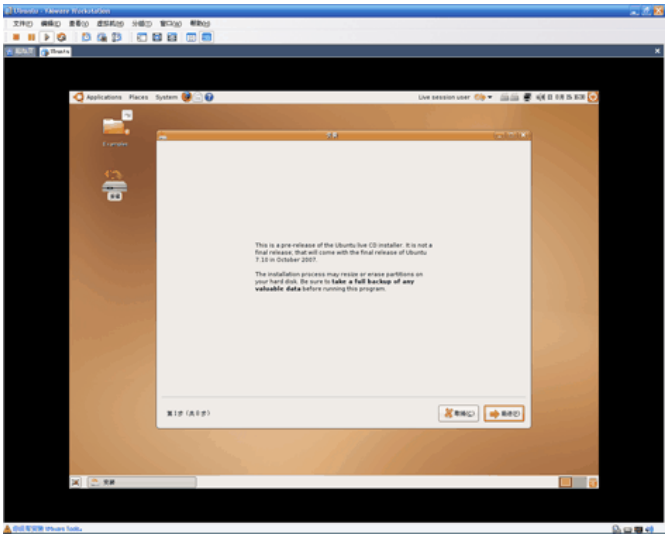
Step4：鼠标单击画面，进入对虚拟机的操作中（按 Ctrl+Alt 回归宿主机）。按 F2 键选择 Language——中文（简体），画面登时友好了许多。选择第一个选项“启动或安装 Ubuntu”吧。



Step5：经过一通曾相识的开机画面，我们进入了 Ubuntu。没有安装怎么就进入了呢？因为我们选择的版本是 Live CD，安装之前允许你领略光盘中现成的 Ubuntu，我们将在这个环境下完成硬盘安装。你完全可以在这个装在光盘中的系统里遨游——这是个完整的 Ubuntu 系统——但由于是光盘的缘故，读取效率绝不会有在硬盘中运行的那么流畅，而且也不可能保留你对它所做的任何改动，因此，我们还是赶紧结束这样的折磨吧。双击桌面上的“安装”图标，开始我们的安装之旅。



Step6：双击“安装”后，首先弹出个洋文对话框（可能会由于分辨率的原因使虚拟机的桌面显示不全，遇到这种情况可以点击屏幕左上角：System——首选项——屏幕分辨率，设定 Resolution 到合适的分辨率，Apply——keep resolution 即可。再不行就调整一下 VMware 的窗口大小吧。关于分辨率带来的苦恼会随着 VMware Tools 的安装而消失，埋个伏笔先^_^），同时有中文说明这是 8 个步骤中的第 1 步。洋文的大意是马上就要安装 Ubuntu 的这个版本了，请提前对硬盘数据做好备份之类。虚拟的机器，无视之，直接前进。



Step7：第 2 步是选择语言。缺省就是中文（简体），继续前进。

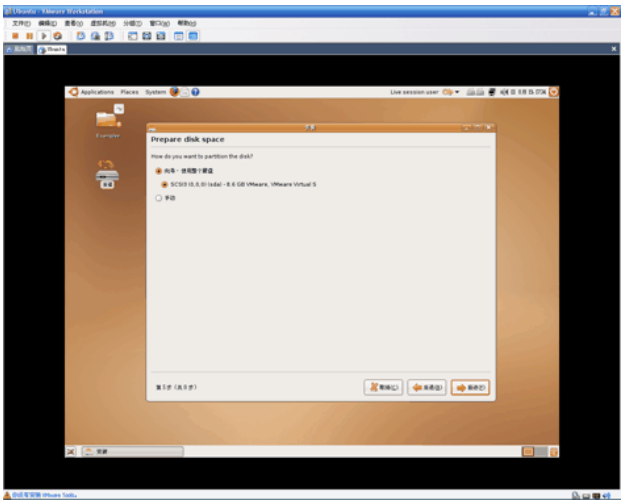
Step8：第 3 步设置时区。缺省就是中国上海（不用找了，找不到北京的。但你如果喜欢选重庆也可以，一样的效果）的东八区，时间与实际时间有出入，没关系，安装之后在系统中

调整吧，继续前进。

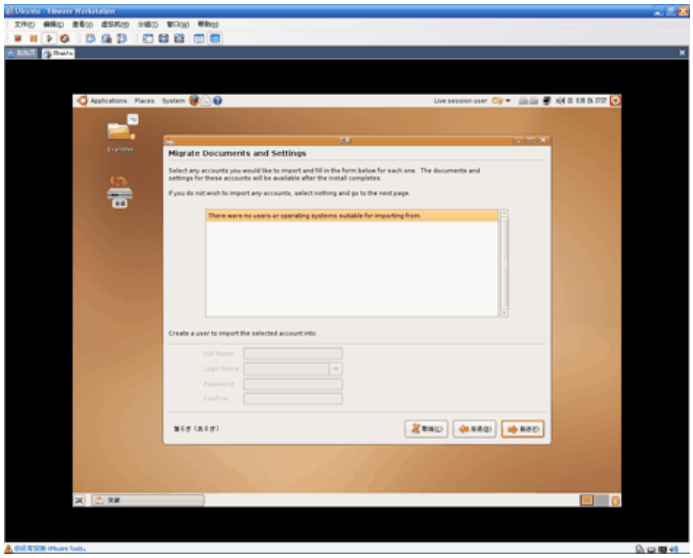
Step9：第 4 步选择键盘种类。我用的是普通的 U.S. English 键盘，缺省设置就行了，直接点前进。

Step10：第 5 步是磁盘分区。磁盘分区是整个安装过程中至关重要的一步，因为 Linux 的分区和文件结构同我们熟悉的 Windows 截然不同，很多 Linux 初心者就是因为不能适应陌生的文件结构，由满怀信心的 Linux 新手变成垂头丧气的 Linux 苦手。叹一个！关于 Ubuntu 分区的知识，强烈建议先读 linux-partition-and-file-system.html" target=_blank>这篇文章。

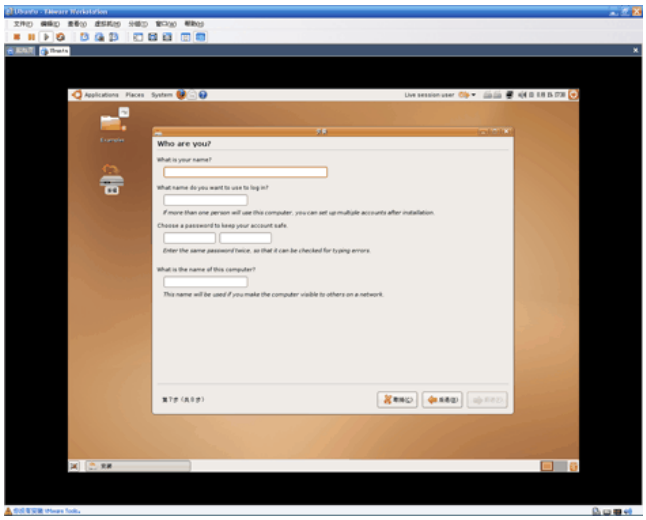
当然我们这里是谈在虚拟机中安装 Ubuntu，完全规避了分区风险，因此使用缺省设置“使用整个硬盘”就可以了（我们虚拟机默认的虚拟硬盘空间是 8G，目前空空如也。它在宿主机 Windows 系统下以一个文件的形式存在，搞成什么样子也不用担心，删除掉就全都清净了），点击前进。



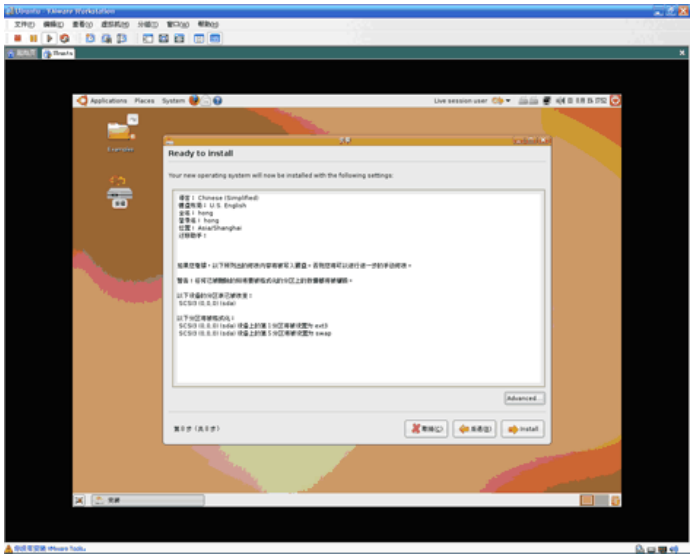
Step11：第 6 步是文件迁移向导。据说可以帮你识别出的原有操作系统（Win XP）环境下的硬盘内容并进行搬家。前面说过，我们的虚拟机硬盘目前空空如也，没什么好迁移的，那么继续前进。



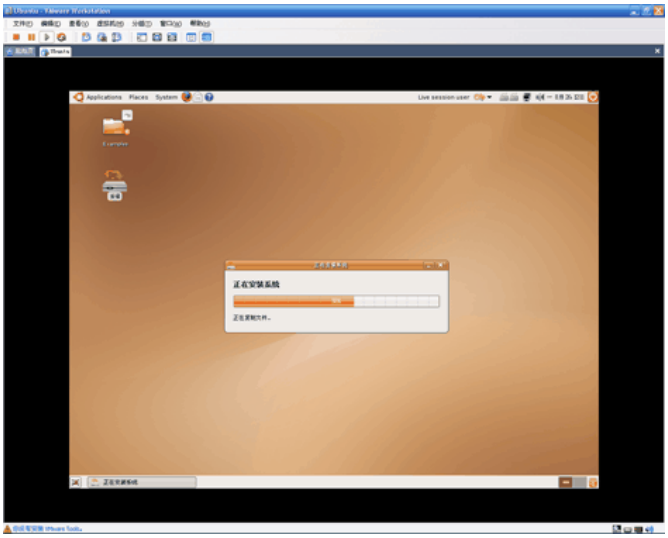
Step12：第 7 步是填写你的账号信息。第一行是你的名字，第二行是你的登录账号（根据你前面填写的名字自动生成推荐，可修改），第三行是输入登录密码并重复之，第四行是计算机名称（根据你前面填写的名字自动生成推荐，可修改）。其中尤其要对第二行和第三行中填写的登录账号和登录密码熟记之，以后我们很多操作要用到它。



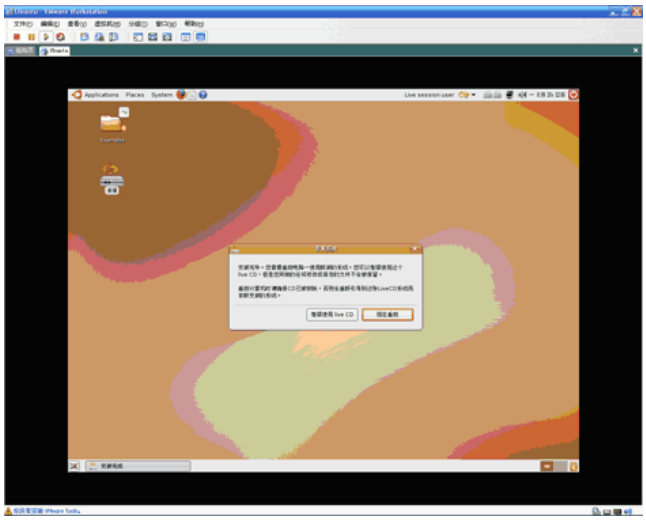
Step13：最后一步是确认。信息栏中很负责任的列举出前面你所做出的各种设定，仔细查看后点击 Install 进行安装。



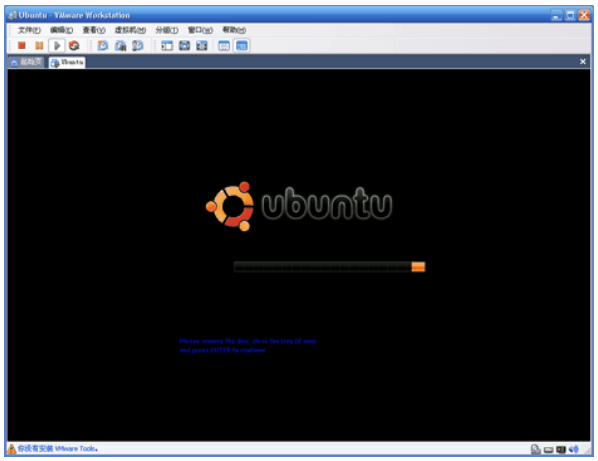
Step14：跟进度条相面吧。除了等，不用你做什么。



Step15：OK，现在 Ubuntu 已经安装到你的 VMware 牌 PC 里了。对话框说：你可以继续在 Live CD 中“慢”游，不过期间对系统所做的任何变更都不会被保留；重启前注意弹出安装盘，否则又会从光盘引导进 Live CD 了。还磨蹭什么？点击重启吧。



Step16：重启的过程类似 Windows，退出系统，再进入系统。在系统退出的最后时刻，进度条不动了，下面显示出一行蓝色小字，大意是请移除安装盘。



这时双击编辑 VMware 窗口的右下角光驱图标，把连接从“使用 ISO 镜像”改为“使用物理驱动器”，确定后点进虚拟机窗口，按回车继续重启

重启完成。展现在我们眼前的是登录画面，没忘记我们安装时设置的登录账号和密码吧？填写进入桌面。至此，Ubuntu 就已经成功安装完成了。

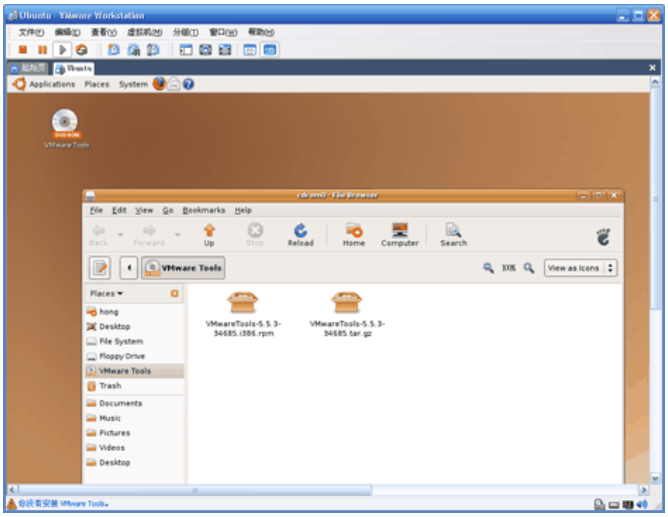
5、 安装 VMware Tools

在前述对 VMware 的操作过程中，你也许已经注意到在 VMware 窗口左下方的状态栏上有“你没有安装 VMware Tools”的蓝色提示。

VMware Tools 何许物也？它是 VMware 提供的一套很贴心的程序，用于解决虚拟机的分辨率问题（我们前面有提过）、改善鼠标的性能（还记得我们前面不断的用鼠标单击和 Ctrl+Alt 在虚拟与现实间切换的费劲场景吗？），并且能将虚拟机的剪贴板内容直接粘贴到宿主机中。当然，不安装 VMware Tools 也不会带来什么灾难，只不过从用户体验方面考虑，装上 VMware Tools 会给你对虚拟机的操作带来尽可能多的方便。

VMware Tools 必须在虚拟机已经开启且已安装操作系统的前提下才能安装。如果你注意过 VMware Workstation 的安装目录，会发现一些命名为 windows.iso、linux.iso、freebsd.iso、solaris.iso 的光盘镜像，这些就是 VMware Tools 在各种操作系统下的安装文件。VMware Tools 就是通过光盘镜像的方式加载到相对应操作系统下来运行安装的。

闲话少叙，点击 VMware 菜单的——虚拟机——安装 VMware Tools，在弹出的对话框中选择“安装”。这时，在 Ubuntu 下会自动加载 Linux 版的 VMware Tools 的安装光盘镜像。你会看到虚拟机的桌面上出现了一个名为 VMware Tools 的光盘图标，并且被自动打开。其中包括 VMwareTools-5.3.3-34685-i386.rpm 和 VMwareTools-5.3.3-34685.tar.gz 两个文件。



.rpm 文件是给 Red Hat 准备的，我们只需要.tar.gz 的那个。

点击 Ubuntu 桌面左上角的 Applications——附件——终端，会弹出一个貌似 Windows 记事本的程序，这就是大名鼎鼎的“终端”。它在 Ubuntu 中的角色类似 Windows 里的 MS DOS 或命令提示符，我们以后会经常性的同它打交道。

在终端界面中依次运行（运行，就是在“$”提示符后面输入一段命令，回车之）如下命令（$是自带的，不用专门输入了）——

$ tar zxf /media/cdrom/VMwareTools-5.3.3-34685.tar.gz（回车后系统会把那个.tar.gz 文件解压缩）

$ cd vmware-tools-distrib（回车后目录将转换到解压缩后的那个文件夹）

$ sudo ./vmware-install.pl（回车后会提示输入你的密码，表明你将以更高级权限执行一个动作——安装软件；再次回车后安装开始）

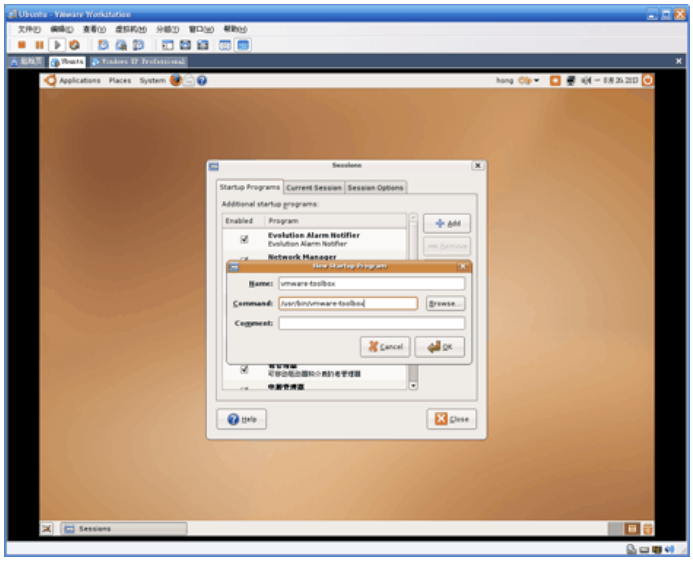
经过一番确认回车后（相当于在 Windows 里安装软件时所填答的诸多对话框，我们这里均采用缺省设置），直到最后出现“Enjoy——the VMware team”的字样后，VMwareTools 终于安装完成了。

如果你使用的是 VMware Workstation 6.0.0，现在你应该已经能看到 VMware Tools 安装后的效果了：鼠标再也不需要用 Ctrl+Alt 切换于虚拟与现实之间，一切过渡得那么自然；虚拟机的分辨率也能依窗口的大小来自动适应，一切显得那样和谐。但我们这里使用的是 VMware Workstation 5.5.3，所以暂时你还看不到什么，让我们来做一下最后的设置吧。

在 VMware Workstation 5.5.3 里，Ubuntu 下的 VMware Tools 安装之后是要靠手动执行 /usr/bin/vmware-toolbox 才能显示出效果的。你可以通过在终端中输入如下命令来运行它：

$ /usr/bin/vmware-toolbox

但如果你嫌麻烦，我们就需要让 vmware-toolbox 实现开机自动运行。点击 Ubuntu 菜单：System——首选项——会话——Startup Programs，Add 一个 Name 叫“vmware-toolbox”，Command 是“/usr/bin/vmware-toolbox”的启动程序。这样每次开机后就能自动运行 VMware Tools 了。重启 Ubuntu 看看效果吧！（注意：**vmware-toolbox 并不是在后台隐藏运行的，启动后不要关闭**）



VMware Tools 固然是个好东西，但也有副作用，比如说：**虚拟机中的鼠标的滚轮不好使了**。我们这样解决这个问题，还是打开终端，输入：

$ sudo gedit /etc/X11/xorg.conf

这个命令使系统以 root 权限打开鼠标配置文件/etc/X11/xorg.conf。把文件中的

Option “Protocol” “ps/2”

改成

Option “Protocol” “IMPS/2”

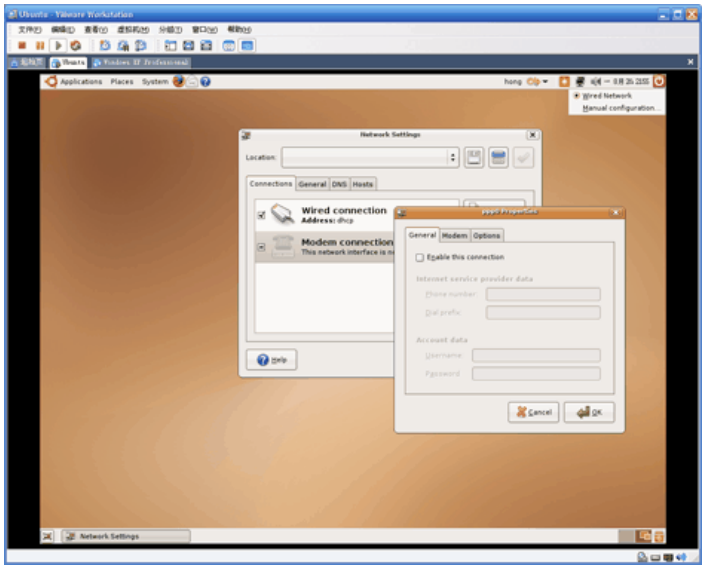
重启 Ubuntu 搞定。

6、 Ubuntu 上网、设置源

上网，这既是一个大问题又是一个小问题。

Ubuntu 是一个比较依赖网络的操作系统，它的几乎所有部分都可以从网上获得更新和扩充，因此联网是我们的系统时刻保持青春活力的重要保证。但至少在版本 6.06 LTS 的时候，Ubuntu 还没有默认集成一个图形化的 adsl 拨号程序，于是上网就成了初学者的老大难问题。

好在我们这里讲的是 7.10 版本，它里面已经集成了图形化的包括拨号在内的网络设置工具，暂时解放了我们这些终端命令的苦手，使上网设置变成了一个小问题。



前面提到我建立这台虚拟机时设置的网络连接形式是 NAT，也就是说共享宿主机的网络。而我的宿主机是连接在一个通过路由器自动拨号的 adsl 家庭小局域网中的，也就是说宿主机和虚拟机不用手动拨号就都可以连接到互联网了。

如果你需要在虚拟机上手动拨号或者需要手动设置虚拟机的 IP 等网络信息（比如选择的连接方式不是 NAT 而是桥接），你可以直接单击桌面右上角的黑色小电脑图标并单击选择“Manual configuration…”进行设置。

确认联网后我们有必要进行一下软件源的设置。我们说过，Ubuntu 的全部都可以通过网络进行更新或扩展，这些可供 Ubuntu 下载使用的软件通过多种形式分散或集中的库存在全球各地，这些远端的可用程序仓库对我们来说就是可供在线安装、更新的软件源。

在诸多的软件源中，我们可以选择一个连接起来速度最快的。软件源通过 System——系统管理——软件源进行设置，具体步骤这里已经写的很清楚了，不再赘述。

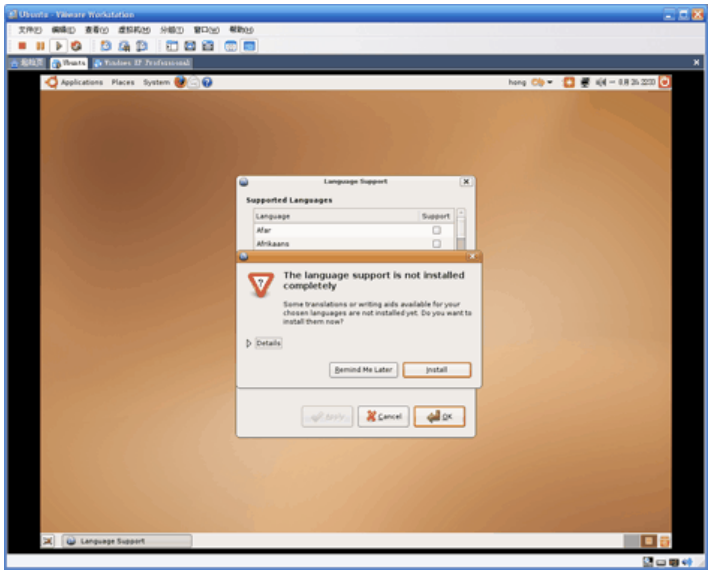
我这里最快的软件源竟然在美国，汗一个！

7、 让你的 Ubuntu 说中文

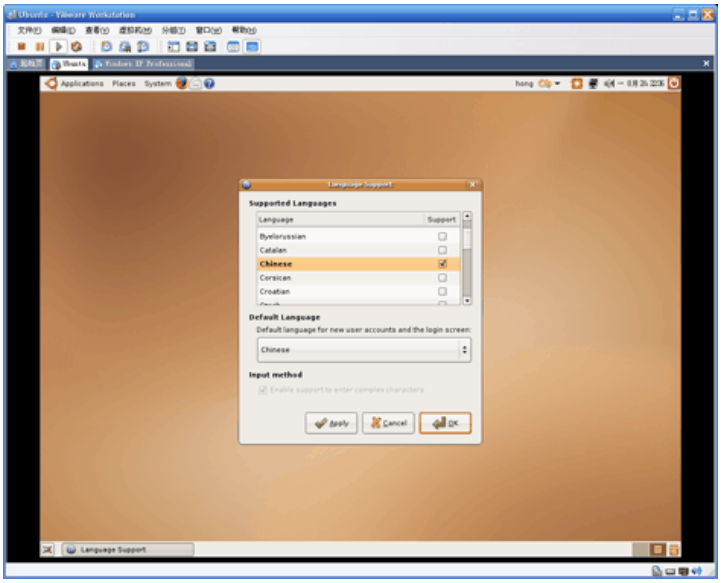
Ubuntu 装到了这个份儿上，心里不禁打鼓：明明选择了中文简体，可怎么还是这样面貌可憎，华夷夹杂呢？Ubuntu 的光盘版本是全球统一的，并不像商业软件那样会为每一个语言地区都单独发行一套版本。因此光盘中的本地化内容往往并不全面，这就需要我们在线下载系统各个组成部分的中文语言包来加以完善。

点击 System——系统管理——Language Support，输入密码后（很多涉及安装、删除、修改、重设等更高权限的操作都需要输入密码）会弹出一个对话框说你的英文语言包没有安装完全，问是不是要在线安装。我们这里讲本地化，不是英国化，点 Remind Me Later 忽略之，

进入语言设定窗口。



在上面 Supported Languages 中勾选 Chinese，单击 Apply 后系统开始自动下载中文语言包（我这次共需要下载 22 个）。



在一番漫长的等待后，你会发现，英文的地方变中文了，字体小的地方变正常了，中文输入法也能使用了，总之我们的 Ubuntu 变可爱了。

附录五 uboot 命令使用教程

Printenv 打印环境变量。

```
Uboot> printenv
baudrate=115200
ipaddr=192.168.1.1
ethaddr=12:34:56:78:9A:BC
serverip=192.168.1.5
Environment size: 80/8188 bytes
```

Setenv 设置新的变量

```
Uboot> setenv myboard AT91RM9200DK
Uboot> printenv
baudrate=115200
ipaddr=192.168.1.1
ethaddr=12:34:56:78:9A:BC
serverip=192.168.1.5
myboard=AT91RM9200DK
Environment size: 102/8188 bytes
```

Saveenv 保存变量

命令将当前定义的所有的变量及其值存入 flash 中。用来存储变量及其值的空间只有 8k 字节，应不要超过。

Loadb 通过串口 Kermit 协议下载二进制数据。

Tftp 通过网络下载程序，需要先设置好网络配置

```
Uboot> setenv ethaddr 12:34:56:78:9A:BC
Uboot> setenv ipaddr 192.168.1.1
Uboot> setenv serverip 192.168.1.254      （tftp 服务器的地址）
下载 bin 文件到地址 0x20000000 处。
Uboot> tftp 20000000 application.bin （application.bin 应位于 tftp 服务程序的目录）

Uboot> tftp 32000000 vmlinux
把 server（IP=环境变量中设置的 serverip）中/tftpdroot/ 下的 vmlinux 通过 TFTP 读入到物理内存 32000000 处。
```

Md 显示内存区的内容。

Mm 修改内存，地址自动递增。

Nm 修改内存，地址不自动递增。

Mw 用模型填充内存

mw 32000000 ff 10000(把内存 0x32000000 开始的 0x10000 字节设为 0xFF)

Cp 拷贝一块内存到另一块

Cmp 比较两块内存区

这些内存操作命令后都可加一个后缀表示操作数据的大小，比如 cp.b 表示按字节拷贝。

Protect 写保护操作

protect on 1:0-3(就是对第一块 FLASH 的 0-3 扇区进行保护)
protect off 1:0-3 取消写保护

Erase 擦除扇区。

erase: 删除 FLASH 的扇区
erase 1:0-2(就是对每一块 FLASH 的 0-2 扇区进行删除)

对 DataFlash 的操作

U-Boot 在引导时如果发现 NPCS0 和 NPCS3 上连有 DataFlash，就会分配虚拟的地址给它，具体为 ：
0xC0000000---NPCS0
0xD0000000---NPCS3

run 执行设置好的脚本

```
Uboot> setenv flashit tftp 20000000 mycode.bin\; erase 10020000 1002FFFF\;
cp.b 20000000 10020000 8000
Uboot> saveenv
Uboot> run flashit
```

bootcmd 保留的环境变量,也是一种脚本

如果定义了该变量，在 autoboot 模式下，将会执行该脚本的内容。

Go 执行内存中的二进制代码，一个简单的跳转到指定地址

Bootm 执行内存中的二进制代码

要求二进制代码为制定格式的。通常为 **mkimage** 处理过的二进制文件。
起动 UBOOT TOOLS 制作的压缩 LINUX 内核, bootm 3200000

Bootp 通过网络启动，需要提前设置好硬件地址。

? 得到所有命令列表

help help usb, 列出 USB 功能的使用说明

ping 注：只能开发板 PING 别的机器

usb

usb start: 起动 usb 功能
usb info: 列出设备
usb scan: 扫描 usb storage(u 盘)设备

kgo 起动没有压缩的 linux 内核

kgo 32000000

fatls 列出 DOS FAT 文件系统

fatls usb 0 列出第一块 U 盘中的文件

fatload 读入 FAT 中的一个文件

fatload usb 0:0 32000000 aa.txt 把 USB 中的 aa.txt 读到物理内存 0x32000000 处!

flinfo 列出 flash 的信息

nfs

nfs 32000000 192.168.0.2:aa.txt
把 192.168.0.2(LINUX 的 NFS 文件系统)中的 NFS 文件系统中的 aa.txt 读入内存 0x32000000 处