Linux 篇

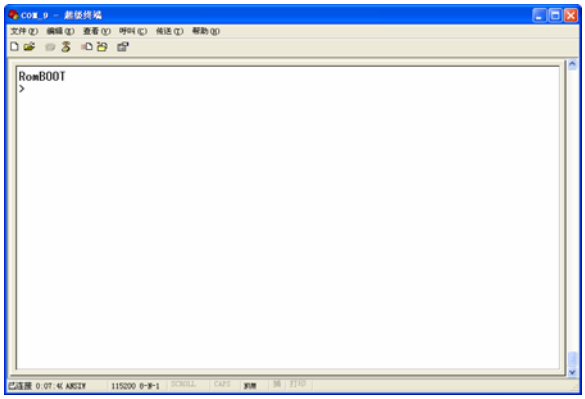
第一章：ATMEL 的 Linux Demo 的烧写、配置以及使用

ATMEL 提供了一个 Linux Demo，我们也可以先测试一下这个 Demo。下面文章中详细叙述了使用 SAM-BA 手动烧写的全过程。再次建议用户不要使用这样的办法，本站提供的测试用的 linux 系统文件包都有一个 bat 文件，只需要在板子进入 SAM-BA 状态，连接好 USB 线后，双击 bat 文件即可完成全部文件的烧写，能有效避免出错。

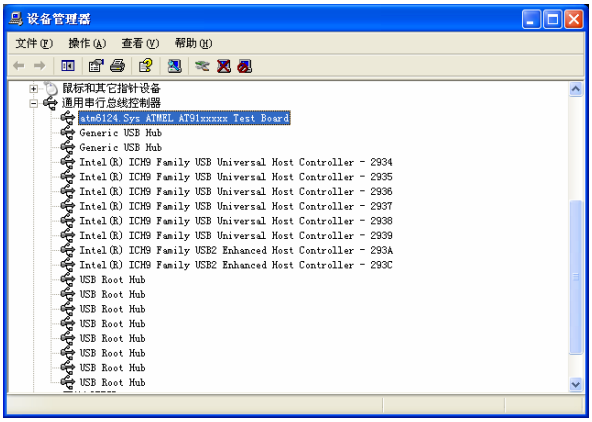
一、烧写

下面我们简单说明一下烧写过程（详细烧写过程请看硬件篇）。

1、断开板上 J21 跳线，然后上电，观察超级终端输出信息，如果正确出现 ROMBOOT 提示即可进行下一步操作：

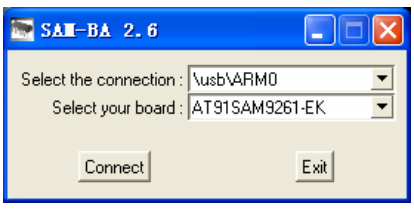


2、插上 USB 线，开发板被识别成：

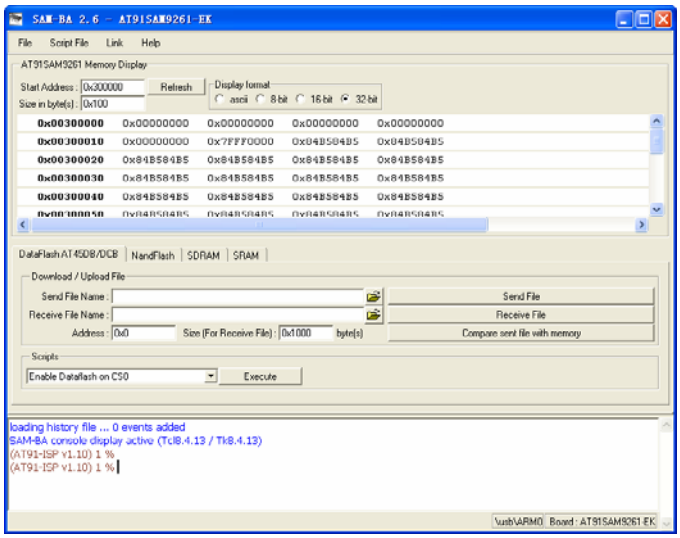


然后插上 J21 跳线至 Data Flash 位置(也即将 data flash 重新连接到系统)。

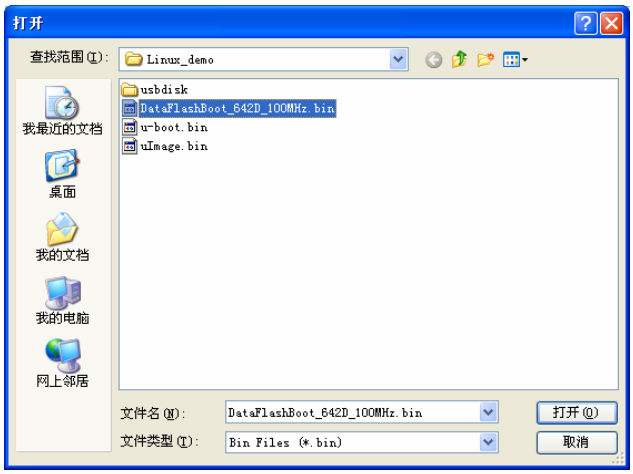
3、打开 SAM-BA，通过 USB 连接：



进入到以下页面：



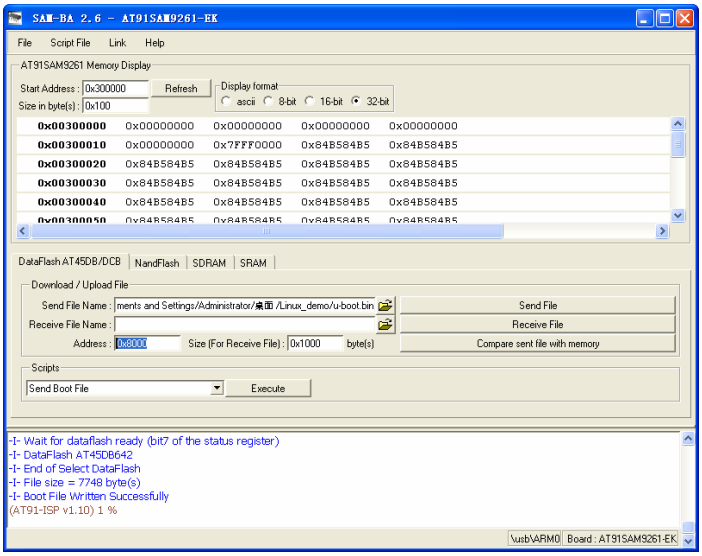
4、烧写 Data Flash。首先选择“Enable Dataflash on CS0”并“Execute”，在下方 log 窗口应该可以看到找到 Dataflash 的提示(如果提示找不到 data flash，请确定在第二步里将 J21 还原)；然后选择“Erase all”并“Execute”，大概 1-2 分钟后完成 Dataflash 擦除操作；接下来就选择“Send boot file”并“Execute”，跳出文件选择框，选择 Linux_demo 下的
“DataFlashBoot_642D_100MHz.bin”：



由于文件比较小，不到 10 秒即可烧写完成。

注意，一级 boot 文件必须使用 send boot file 的方式烧写，直接手动烧写到 0x0 位置将不能启动。

接下来烧写“u-boot.bin”，选择写入地址为 0x8000，点击“send file”按钮发送的“u-boot.bin”，大概 10－20 秒，即可完成烧写。

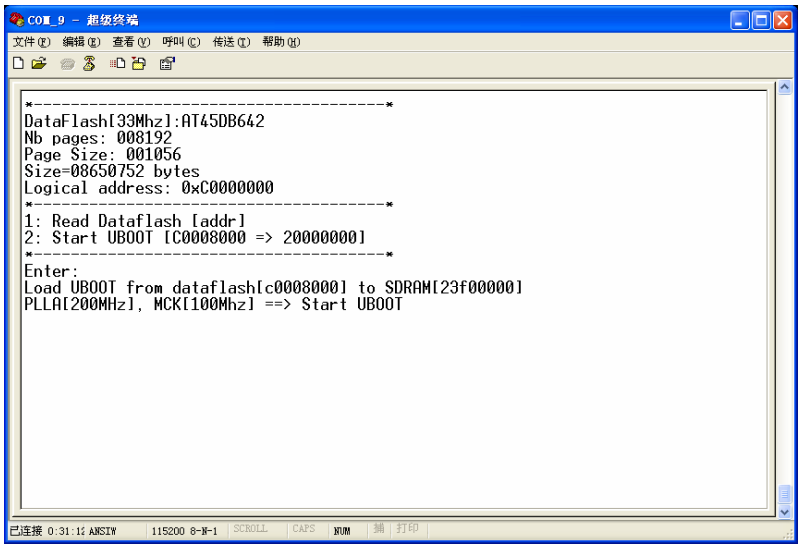


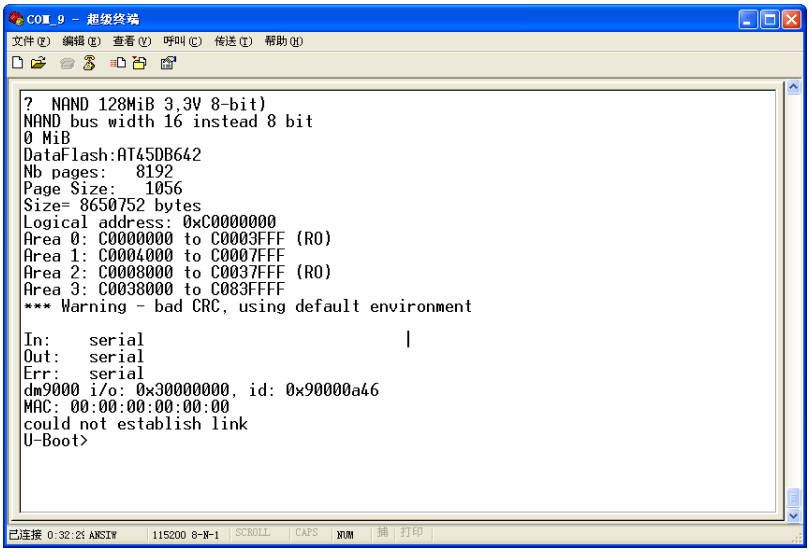
然后烧写“uImage.bin”同样的操作，不过需要将发送地址修改成 0x100000。由于“uImage.bin”较大，发送时间较长，大概会消耗 1－2 分钟。

5、烧写文件系统。由于 ATMEL 提供的 Linux_demo 将文件系统放在 U 盘（Disk on Key）里面，所以我们要准备一个 U 盘，考虑到兼容性问题，建议使用容量较小的早期 U 盘进行测试。请将 U 盘格式化成 FAT 格式，然后把 Linux_demo 下的 usbdisk.zip 解压，里面有三个文件夹，分别是：admin、photos、videos。Videos 文件夹内有两个比较大的 AVI 文件，考虑到播放时间，建议更换成较短的 AVI 文件，比如使用 windows 目录下的几个小 avi 文件，另外也可以放入较短的 mp3 文件进行测试。注意，admin、photos、videos 需要放在 U 盘的根目录！

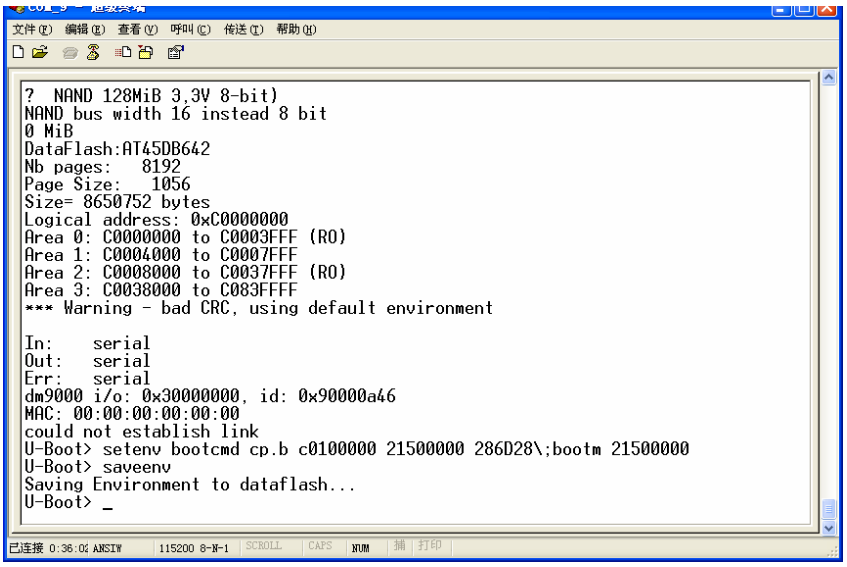
二、配置

烧写完成后拔掉 USB 线缆，按 reset 按钮复位，观察超级终端输出信息：



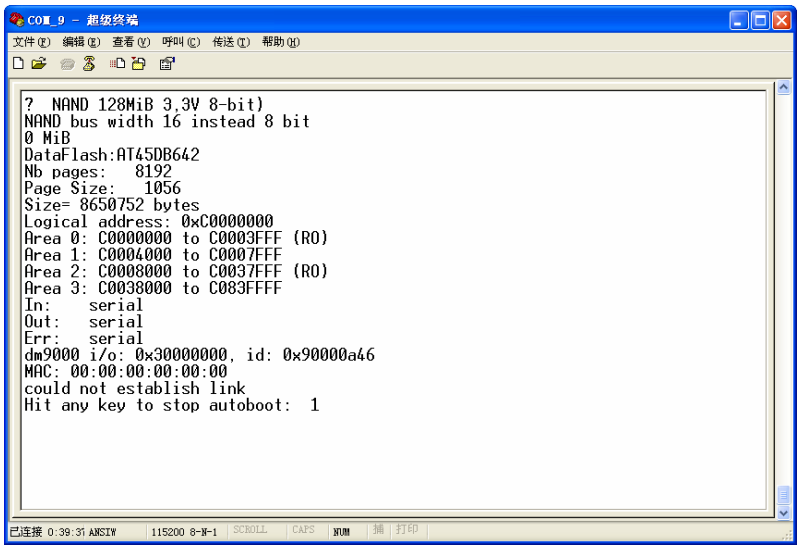


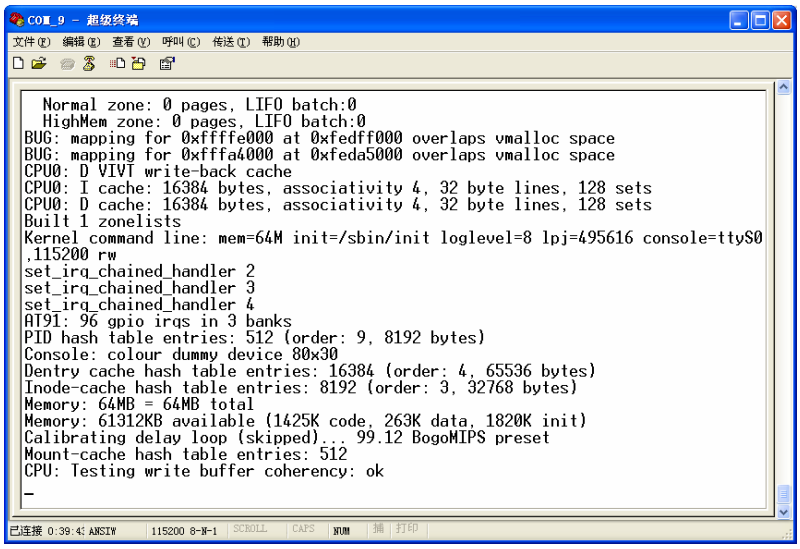
网络可以按照需要决定是否连接，这里我们暂不连接。
然后设置一下启动参数：
在 U-Boot>提示符下输入以下命令：
setenv bootcmd cp.b c0100000 21500000 286D28\;bootm 21500000
然后 saveenv，参数保存到 Dataflash。

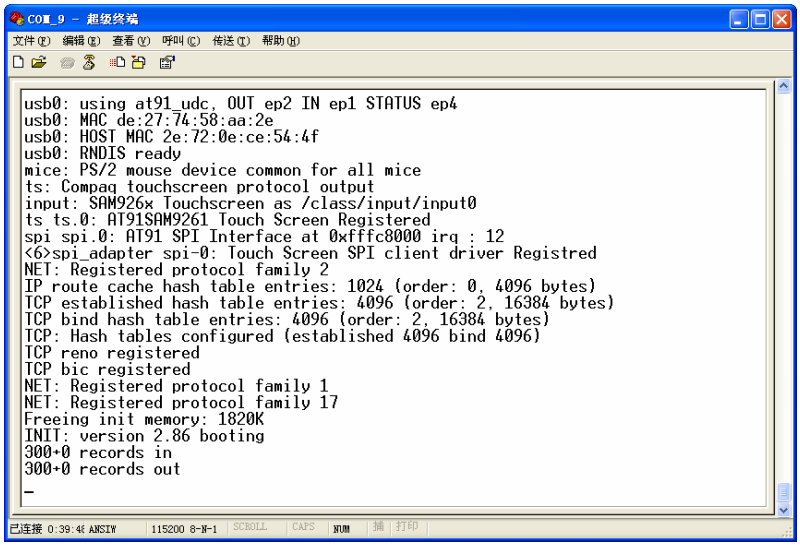


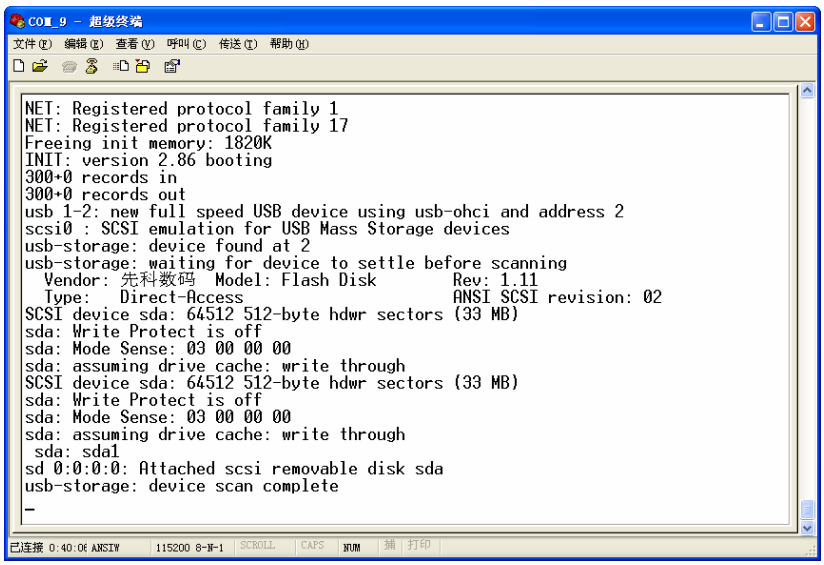
三、使用

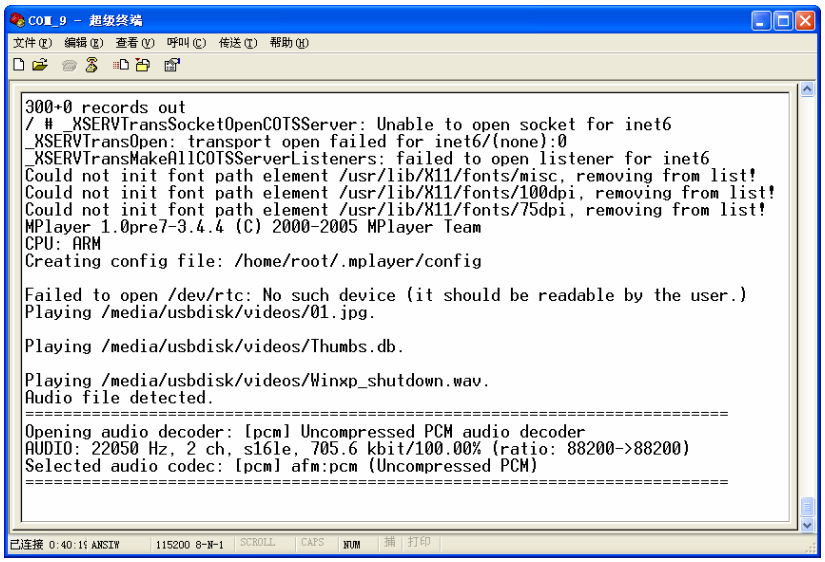
插上 U 盘，按 reset 键复位运行：

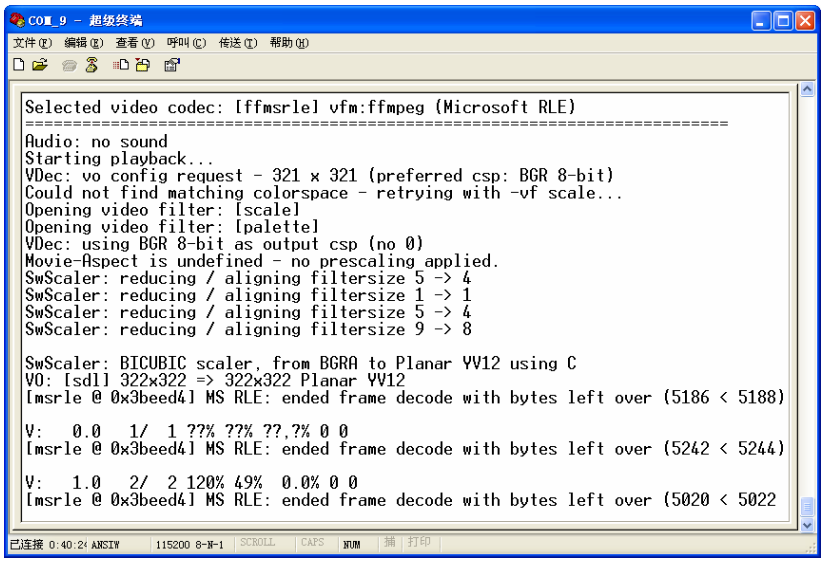


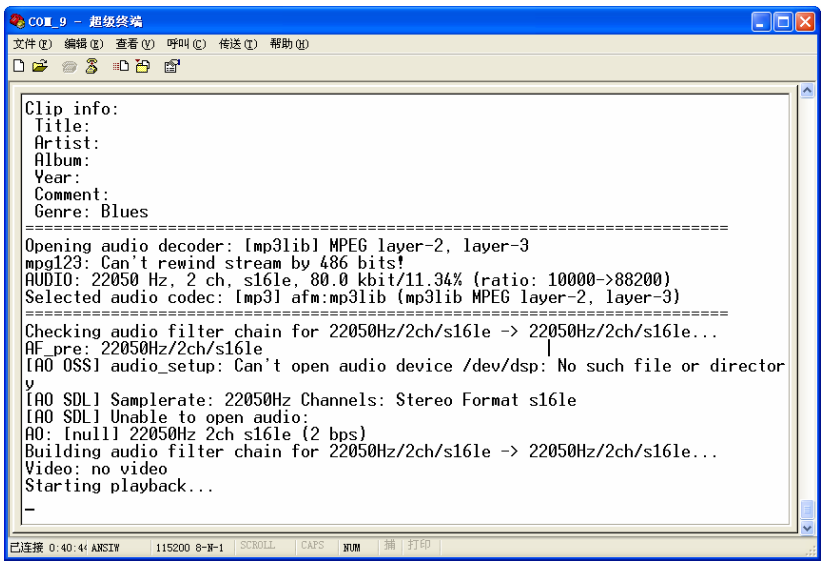

COM_9 - 超级终端
文件(F) 编辑(E) 查看(V) 呼叫(C) 传送(T) 帮助(H)
Normal zone: 0 pages, LIFO batch:0
HighMem zone: 0 pages, LIFO batch:0
BUG: mapping for 0xffffe000 at 0xfedff000 overlaps vmalloc space
BUG: mapping for 0xfffa4000 at 0xfeda5000 overlaps vmalloc space
CPU0: D VIVT write-back cache
CPU0: I cache: 16384 bytes, associativity 4, 32 byte lines, 128 sets
CPU0: D cache: 16384 bytes, associativity 4, 32 byte lines, 128 sets
Built 1 zonelists
Kernel command line: mem=64M init=/sbin/init loglevel=8 lpj=495616 console=ttyS0
,115200 rw
set_irq_chained_handler 2
set_irq_chained_handler 3
set_irq_chained_handler 4
AT91: 96 gpio irqs in 3 banks
PID hash table entries: 512 (order: 9, 8192 bytes)
Console: colour dummy device 80x30
Dentry cache hash table entries: 16384 (order: 4, 65536 bytes)
Inode-cache hash table entries: 8192 (order: 3, 32768 bytes)
Memory: 64MB = 64MB total
Memory: 61312KB available (1425K code, 263K data, 1820K init)
Calibrating delay loop (skipped)... 99.12 BogoMIPS preset
Mount-cache hash table entries: 512
CPU: Testing write buffer coherency: ok
已连接 0:39:4 ANSIW
115200 8-N-1
SCROLL
CAPS
NUM

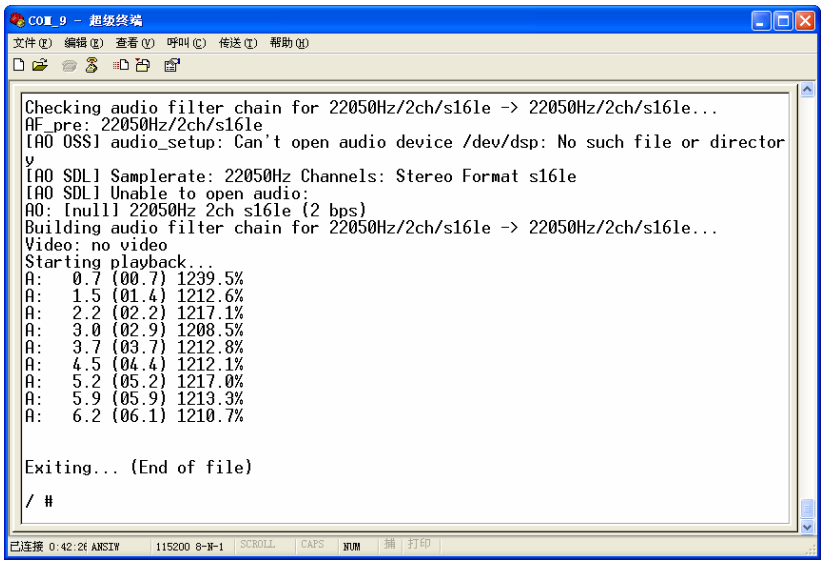

COM_9 - 超级终端
文件(F) 编辑(E) 查看(V) 呼叫(C) 传送(T) 帮助(H)
usb0: using at91_udc, OUT ep2 IN ep1 STATUS ep4
usb0: MAC de:27:74:58:aa:2e
usb0: HOST MAC 2e:72:0e:ce:54:4f
usb0: RNDIS ready
mice: PS/2 mouse device common for all mice
ts: Compaq touchscreen protocol output
input: SAM926x Touchscreen as /class/input/input0
ts ts.0: AT91SAM9261 Touch Screen Registered
spi spi.0: AT91 SPI Interface at 0xfffc8000 irq : 12
<6>spi_adapter spi-0: Touch Screen SPI client driver Registred
NET: Registered protocol family 2
IP route cache hash table entries: 1024 (order: 0, 4096 bytes)
TCP established hash table entries: 4096 (order: 2, 16384 bytes)
TCP bind hash table entries: 4096 (order: 2, 16384 bytes)
TCP: Hash tables configured (established 4096 bind 4096)
TCP reno registered
TCP bic registered
NET: Registered protocol family 1
NET: Registered protocol family 17
Freeing init memory: 1820K
INIT: version 2.86 booting
300+0 records in
300+0 records out
已连接 0:39:4 ANSIW
115200 8-N-1
SCROLL
CAPS
NUM


COM_9 - 超级终端
文件(F) 编辑(E) 查看(V) 呼叫(C) 传送(T) 帮助(H)
NET: Registered protocol family 1
NET: Registered protocol family 17
Freeing init memory: 1820K
INIT: version 2.86 booting
300+0 records in
300+0 records out
usb 1-2: new full speed USB device using usb-ohci and address 2
scsi0 : SCSI emulation for USB Mass Storage devices
usb-storage: device found at 2
usb-storage: waiting for device to settle before scanning
Vendor: 先科数码 Model: Flash Disk Rev: 1.11
Type: Direct-Access ANSI SCSI revision: 02
SCSI device sda: 64512 512-byte hdwr sectors (33 MB)
sda: Write Protect is off
sda: Mode Sense: 03 00 00 00
sda: assuming drive cache: write through
SCSI device sda: 64512 512-byte hdwr sectors (33 MB)
sda: Write Protect is off
sda: Mode Sense: 03 00 00 00
sda: assuming drive cache: write through
sda: sda1
sd 0:0:0:0: Attached scsi removable disk sda
usb-storage: device scan complete
已连接 0:40:0 ANSIW
115200 8-N-1
SCROLL
CAPS
NUM


COM_9 - 超级终端
文件(F) 编辑(E) 查看(V) 呼叫(C) 传送(T) 帮助(H)
300+0 records out
/ # _XSERVTransSocketOpenCOTSServer: Unable to open socket for inet6
_XSERVTransOpen: transport open failed for inet6/(none):0
_XSERVTransMakeAllCOTSServerListeners: failed to open listener for inet6
Could not init font path element /usr/lib/X11/fonts/misc, removing from list!
Could not init font path element /usr/lib/X11/fonts/100dpi, removing from list!
Could not init font path element /usr/lib/X11/fonts/75dpi, removing from list!
MPlayer 1.0pre7-3.4.4 (C) 2000-2005 MPlayer Team
CPU: ARM
Creating config file: /home/root/.mplayer/config
Failed to open /dev/rtc: No such device (it should be readable by the user.)
Playing /media/usbdisk/videos/01.jpg.
Playing /media/usbdisk/videos/Thumbs.db.
Playing /media/usbdisk/videos/Winxp_shutdown.wav.
Audio file detected.
Opening audio decoder: [pcm] Uncompressed PCM audio decoder
AUDIO: 22050 Hz, 2 ch, s16le, 705.6 kbit/100.00% (ratio: 88200->88200)
Selected audio codec: [pcm] afm:pcm (Uncompressed PCM)
已连接 0:40:15 ANSIW
115200 8-N-1
SCROLL
CAPS
NUM


COM_9 - 超级终端
文件(F) 编辑(E) 查看(V) 呼叫(C) 传送(T) 帮助(H)
Selected video codec: [ffmsrle] vfm:ffmpeg (Microsoft RLE)
Audio: no sound
Starting playback...
VDec: vo config request - 321 x 321 (preferred csp: BGR 8-bit)
Could not find matching colorspace - retrying with -vf scale...
Opening video filter: [scale]
Opening video filter: [palette]
VDec: using BGR 8-bit as output csp (no 0)
Movie-Aspect is undefined - no prescaling applied.
SwScaler: reducing / aligning filtersize 5 -> 4
SwScaler: reducing / aligning filtersize 1 -> 1
SwScaler: reducing / aligning filtersize 5 -> 4
SwScaler: reducing / aligning filtersize 9 -> 8
SwScaler: BICUBIC scaler, from BGRA to Planar YV12 using C
VO: [sdl] 322x322 => 322x322 Planar YV12
[msrle @ 0x3beed4] MS RLE: ended frame decode with bytes left over (5186 < 5188)
V: 0.0 1/ 1 ??% ??% ??,?% 0 0
[msrle @ 0x3beed4] MS RLE: ended frame decode with bytes left over (5242 < 5244)
V: 1.0 2/ 2 120% 49% 0.0% 0 0
[msrle @ 0x3beed4] MS RLE: ended frame decode with bytes left over (5020 < 5022
已连接 0:40:24 ANSIW
115200 8-N-1
SCROLL
CAPS
NUM


COM_9 - 超级终端
文件(F) 编辑(E) 查看(V) 呼叫(C) 传送(T) 帮助(H)
Clip info:
Title:
Artist:
Album:
Year:
Comment:
Genre: Blues
Opening audio decoder: [mp3lib] MPEG layer-2, layer-3
mpg123: Can't rewind stream by 486 bits!
AUDIO: 22050 Hz, 2 ch, s16le, 80.0 kbit/11.34% (ratio: 10000->88200)
Selected audio codec: [mp3] afm:mp3lib (mp3lib MPEG layer-2, layer-3)
Checking audio filter chain for 22050Hz/2ch/s16le -> 22050Hz/2ch/s16le...
AF_pre: 22050Hz/2ch/s16le
[AO OSS] audio_setup: Can't open audio device /dev/dsp: No such file or director
y
[AO SDL] Samplerate: 22050Hz Channels: Stereo Format s16le
[AO SDL] Unable to open audio:
AO: [null] 22050Hz 2ch s16le (2 bps)
Building audio filter chain for 22050Hz/2ch/s16le -> 22050Hz/2ch/s16le...
Video: no video
Starting playback...
已连接 0:40:44 ANSIW
115200 8-N-1
SCROLL
CAPS
NUM

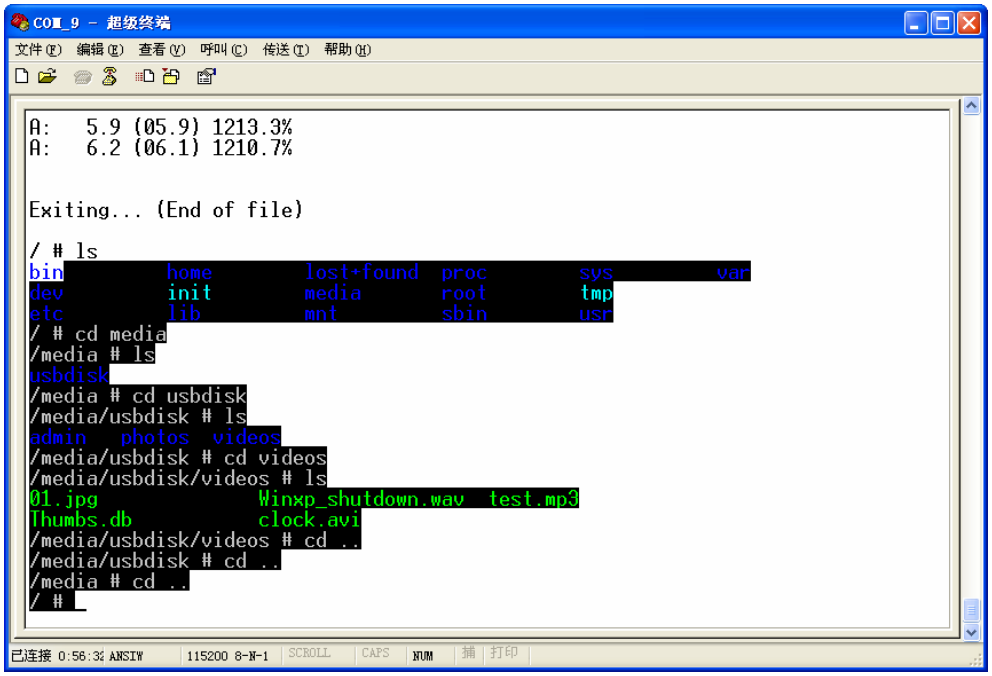


以上一共播放了 3 个文件，分别是一个 wav 文件，一个 avi 文件，一个 mp3 文件。

Linux_demo 下好像没有 AT73C213 的驱动，所以没有音频输出。

播放完成后按回车进入“/ #”提示符下。

下面我们在“/ #”提示符下简单使用几个命令，更多的命令可以参考本手册的附录部分。



ls 和 cd 是最基本的命令，ls 和 windows 下的 dir 命令类似，用来显示当前目录的内容。如果想取得详细的信息，可用 ls -l 命令， 这样就可以显示目录内容

的详细信息。如果目录下的文件太多，用一屏显示不了，可以用 ls -l |more 分屏显示。cd 命令不仅显示当前状态，还改变当前状态，它的用发跟 dos 下的 cd 命令基本一致。cd ..可进入上一层目录；cd -可进入上一个进入的目录；cd ~可进入用户的 home 目录

find 命令用于查找文件。这个命令可以按文件名、建立或修改日期、所有者(通常是建立文件的用户)、文件长度或文件类型进行搜索。 find 命令的基本结构如下： $find

其中指定从哪个目录开始搜索。指定搜索条件。表示找到文件怎么处理。一般来说，要用-print 动作，显示 整个文件路径和名称。如果没有这个动作，则 find 命令进行所要搜索而不显示结果，等于白费劲。 例如，要搜索系统上所有名称为 ye 的文件，可用如下命令：

$find / -name ye -print

这样就可以显示出系统上所有名称为 ye 的文件。

mkdir 这个命令很简单，跟 dos 的 md 命令用法几乎一样，用于建立目录。

cp 命令用于复制文件或目录。cp 命令可以一次复制多个文件，例如：

$cp *.txt *.doc *.bak /home

将当前目录中扩展名为 txt、doc 和 bak 的文件全部复制到/home 目录中。

如果要复制整个目录及其所有子目录，可以用 cp -R 命令。

rm 命令用于删除文件或目录。rm 命令会强制删除文件，如果想要在删除时提示确认，可用 rm -i 命令。如果要删除目录，可用 rm -r 命令。rm -r 命令在删除目录时，每删除一个文件或目录都会显示提示，如果目录太大，响应每个提示是不现实的。这时可以用 rm -rf 命令来强制删除目录，这样即使用了-i 标志也当无效处理。

mv 命令用于移动文件和更名文件。例如：

$mv ye.txt /home

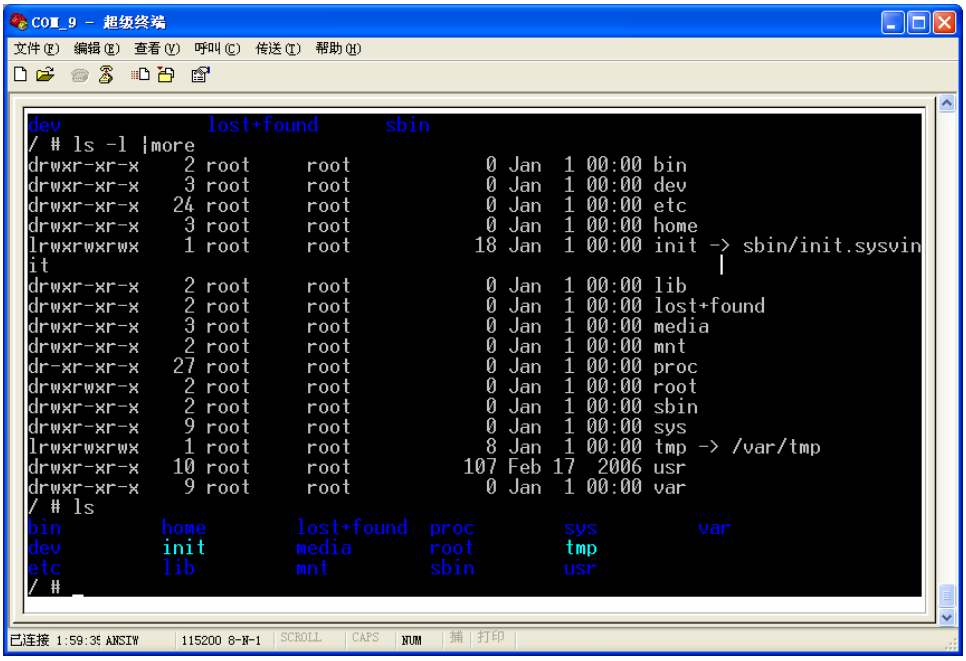
将当前目录下的 ye.txt 文件移动到/home 目录下，

$mv ye.txt ye1.txt

将 ye.txt 文件改名为 ye1.txt。

类似于跟 cp 命令，mv 命令也可以一次移动多个文件，在此不再赘叙。

下面我们再简单介绍一下根目录下的文件夹：



bin 为 Linux 的 bin 文件；
home 为每个用户的根目录；
proc 是虚拟文件系统，是 linux 启动的时候创建的，里面有很多系统的信息；
sys 里面是系统的目录；
dev 里面有所有的设备，linux 下面设备都是文件；
lib 里面有系统的链接库；
usr 就是 user，一些应用程序；
etc 里面 n 多关于系统启动的东西，一些配置信息也在里面；
sbin 里面也是一些可执行文件；
media 为媒体文件，里面为 mount 上来的设备，比如 usbdisk

更多细节可以翻阅 linux 相关文档！

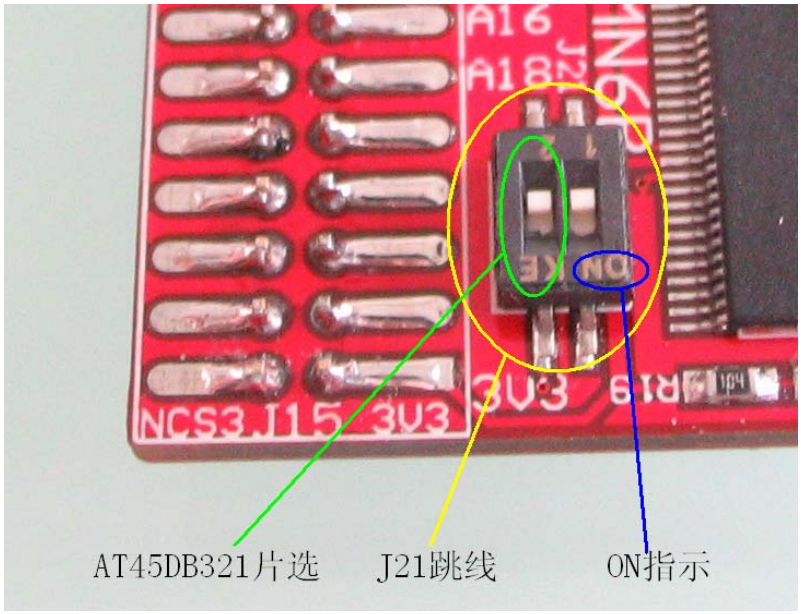
第二章：Mcuzone 的 Linux Demo 的烧写、配置以及使用

考虑到实际使用情况，我们将原本 ATMEL 的 AT91SAM9261-EK 上使用的 8MB NOR FLASH 裁减为 4MB，即将 AT45DB642 更换为 AT45DB321；将原本 ATMEL 的 AT91SAM9261-EK 上使用的 256MB NAND FLASH 裁减为 128MB，即将 K9F2G08 更换为 K9F1G08。

下面我们提供一个基于 AT45DB321 和 K9F1G08 的 Linux 演示案例。

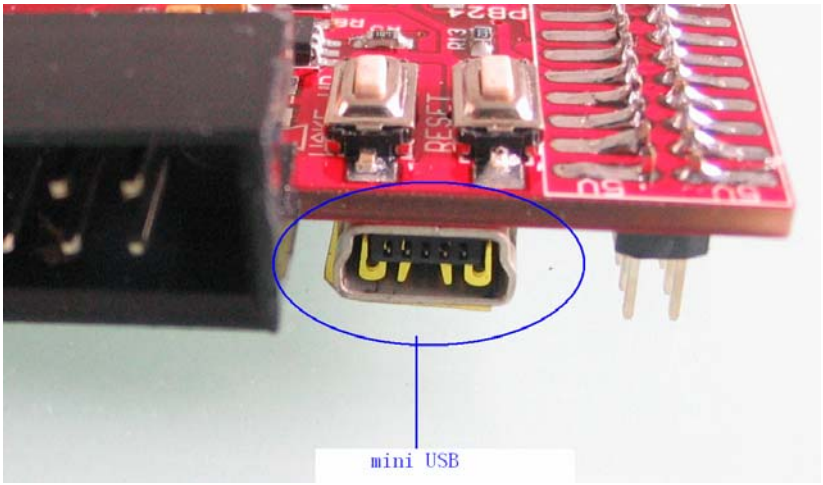
一、烧写一级 boot

在 PC 上安装 SAM-BA 的最新版本（当前版本为 AT91-ISP V1.12）。将核心板上的 NAND FLASH 附近的 J21 拨码开关中的靠近接插件的那一位(下图绿色椭圆框内)拨到 OFF 状态。如下图：



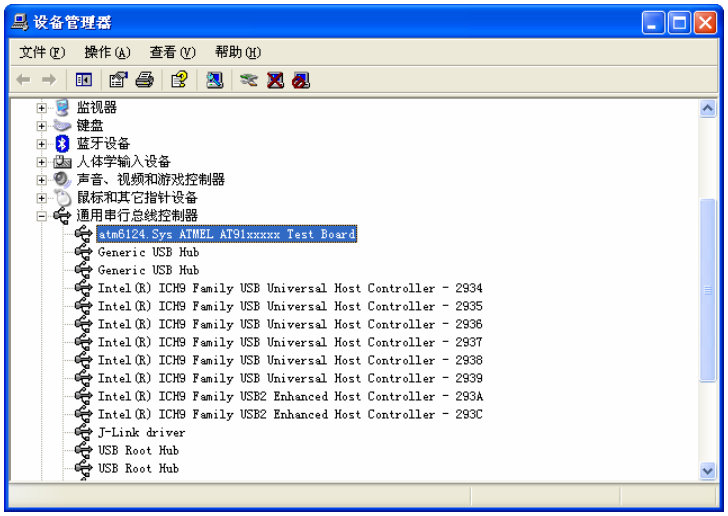
请确认 AT45DB321 的片选在 OFF 状态，即拨位开关 ON 指示的另外一边。然后连接将串口线连接到 VC9261-EK 的底板上的 DBGU 串口 J4。

接下来连接 mini USB 线，插座在核心板反面，Reset 按钮下方，如下图：

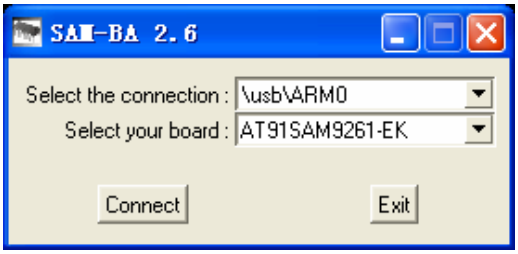


注意：VC9261-EK 底板上的方口 USB Device 接口 J11 和核心板上的 miniUSB 是物理连通的，即既可以通过 miniUSB 来和 PC 连接，也可以通过底板的方口 USB Device 接口来和 PC 连接。

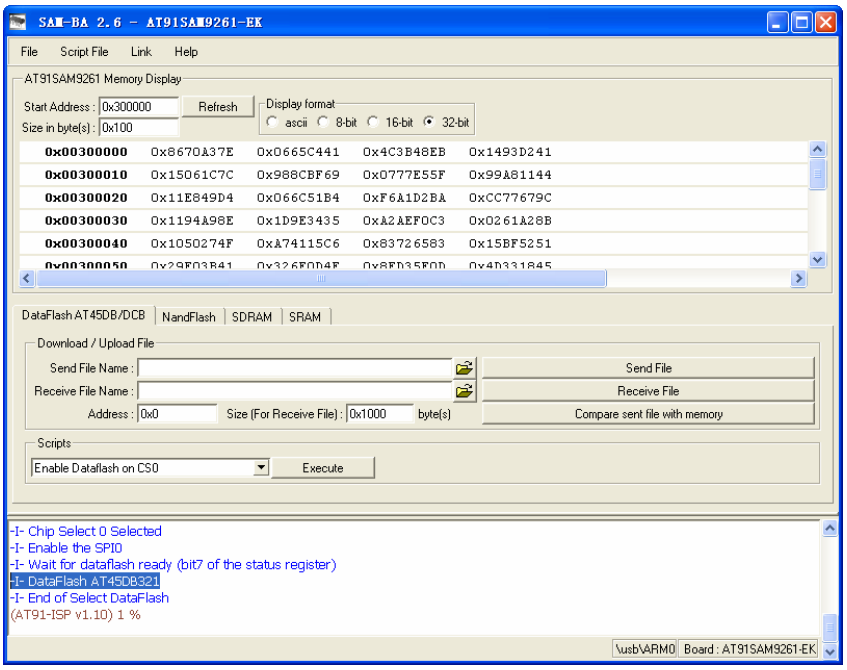
连接 OK 后 PC 即会认出新的设备，自动安装驱动即可。安装完成可以在设备管理器里面看到：



运行 SAM-BA,选择 9261EK：



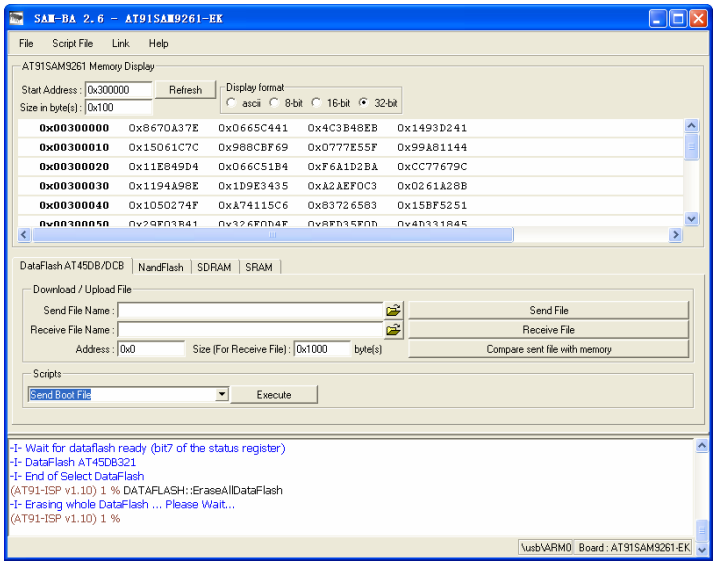
进入 SAM-BA 的界面后，重新将 MN6 下方开关 J21 拨到 ON 处，连接上 Data flash。先选择初始化 SDRAM，然后选择初始化 data flash：

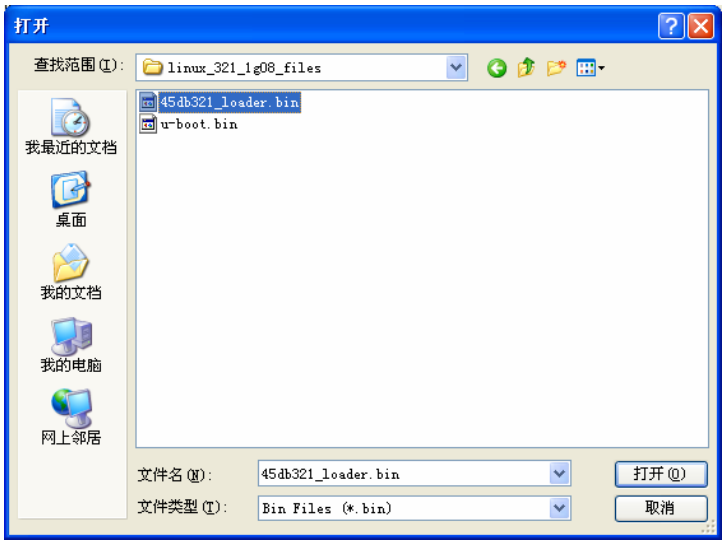


初始化 flash 成功后建议先全片擦除一下，大概会在 20 秒内完成，在 SAM-BA 提示擦除成功后就可以选择烧写一级 boot 文件：

DataFlashBoot_642D_100MHz.bin

在 Scripts 下拉菜单框内选择“Send Boot file”，然后点击“Execute”按钮：

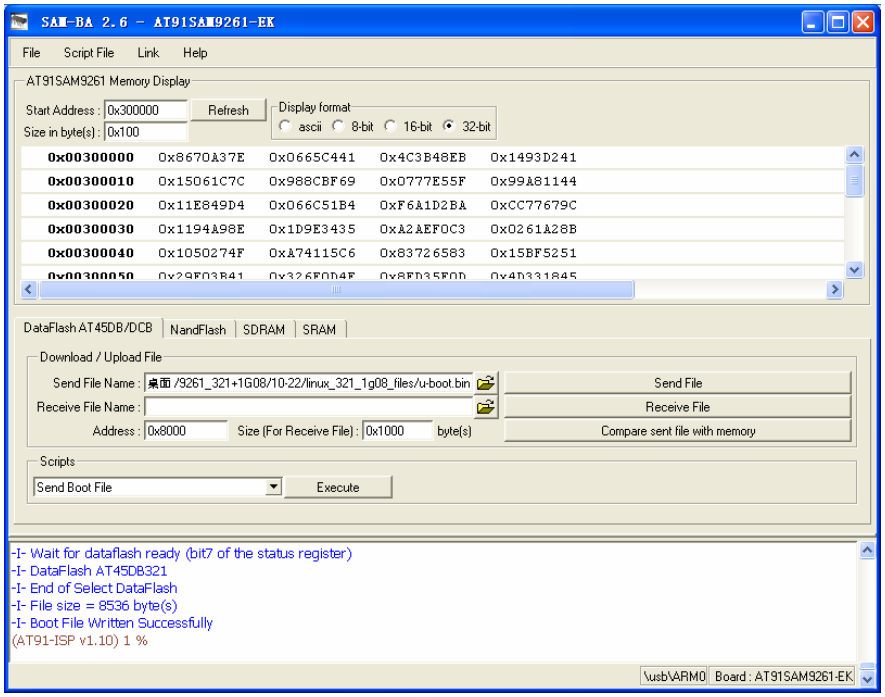




文件较小，很快即可烧录成功。

二、烧写 u-boot

烧写一级 boot 完成后，再选择烧写 u-boot.bin,如下图设置：



文件选择 u-boot.bin，烧写地址为 0x8000，然后点击“Send File”按钮，不到 20 秒即可完成烧写。建议烧写完成后点击一下“Compare sent file with memory”进行校验。

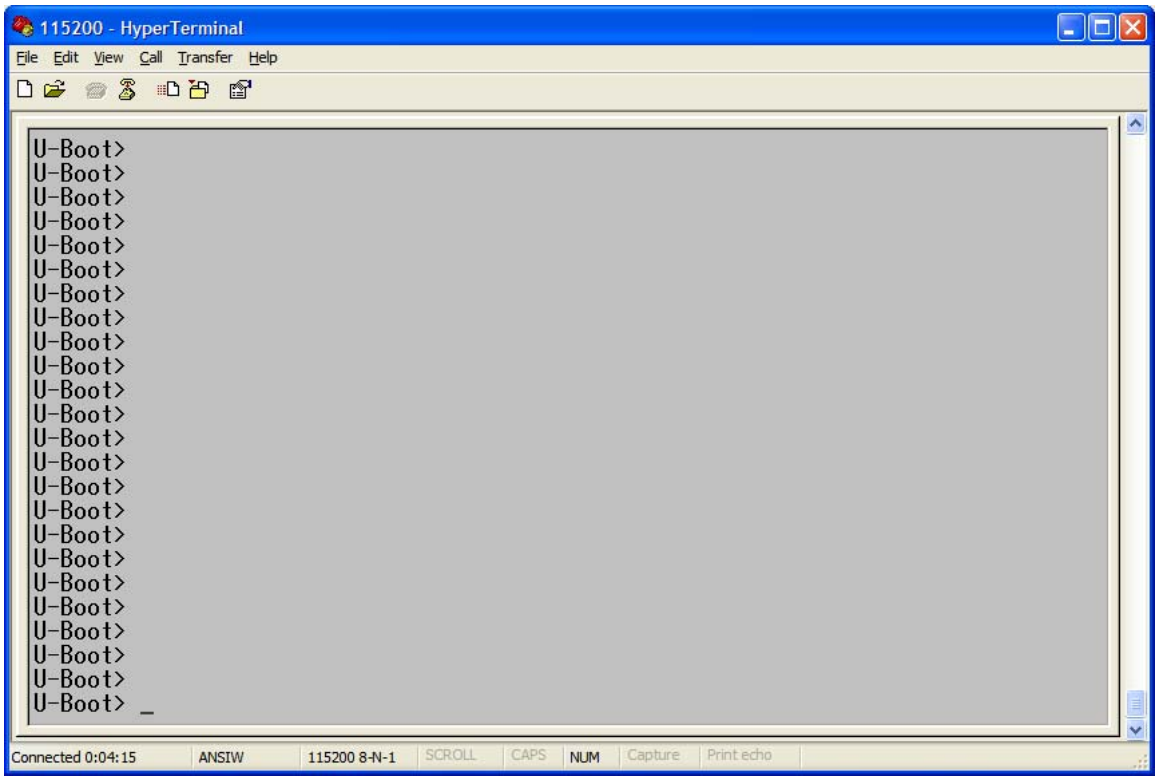
检验通过的话在 SAM-BA 的 log 窗口可以看到提示：

-I- Read File FiletempPcMonitor.testCompare at address 0x8000

校验正确即可关闭 SAM-BA，否则请再烧写一次，直到烧写无误。

退出 SAM-BA 后，拔掉 mini USB 线。连接上 usb device 到 PC，电脑将认出一个串口，请安装 ARK3116 的驱动程序，然后选择自动即可安装上驱动，电脑上将多出一个串口。

在电脑上打开一个超级终端，选择刚才的串口，参数设置为 115200,n,8,1。再次连接 mini USB，u-boot 将启动。如下图：



这说明 u-boot 已经正确运行。

下面可以测试下 NAND，输入下面命令查看 NAND 信息：

```
U-Boot> nand info

Device 0: NAND 128MiB 3,3V 8-bit, sector size 128 KiB
U-Boot>
```

再输入 nand erase clean，将 NAND 执行全片擦除，擦除过程中如果有 bad block，将会输出其信息。

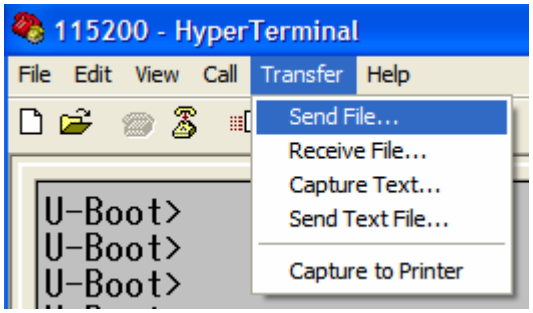
擦除完成后也可以输入 nand bad 查看 bad block 的信息：

```
U-Boot> nand bad

Device 0 bad blocks:
U-Boot>
```

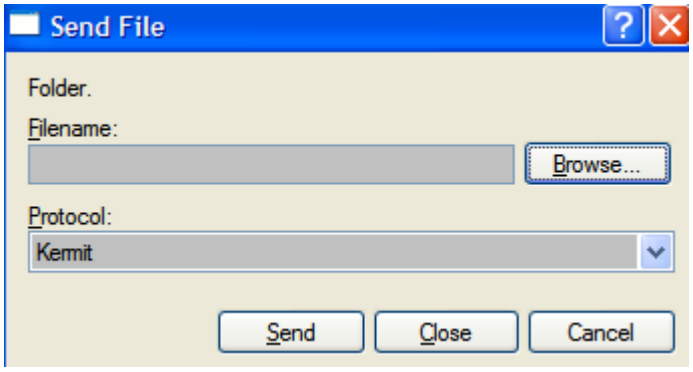
如上的信息表示没有 bad block，可以放心进行下一步烧写操作。

三、烧写 linux 内核

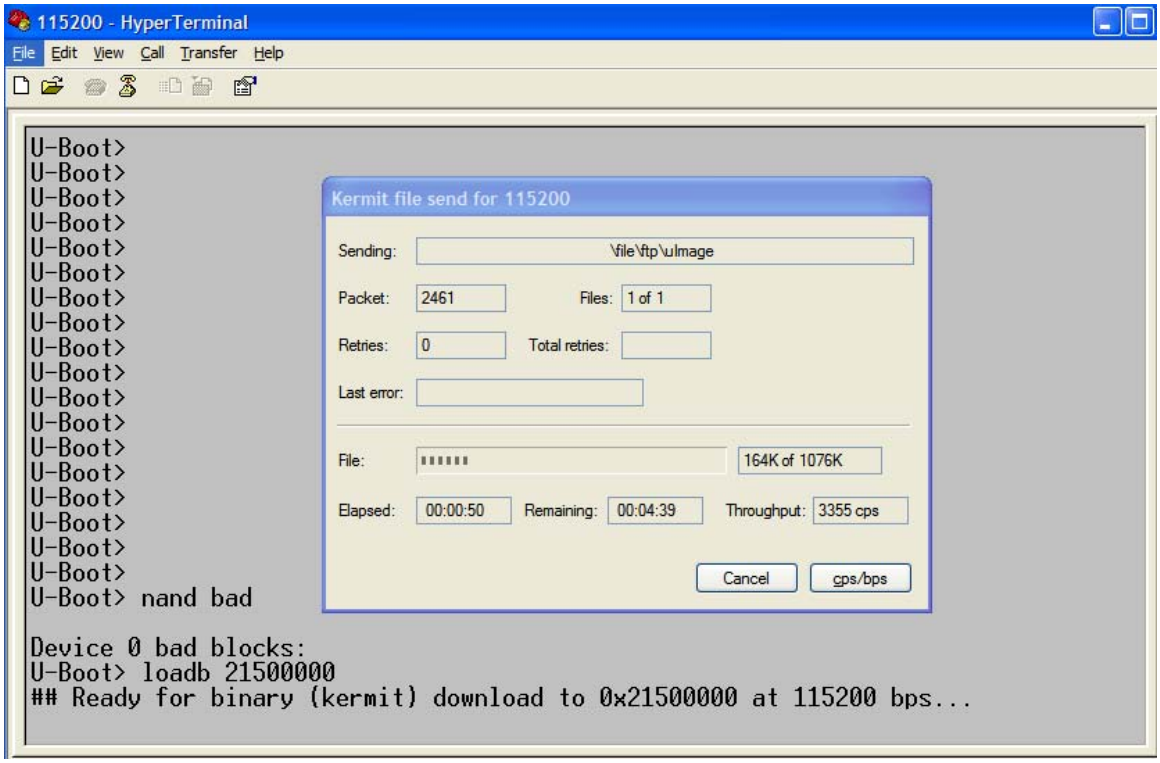
在 u-boot 提示符下输入 loadb 21500000，然后在超级终端的菜单中选择发送文件：



选择需要传输的 linux kernel 的 image，文件名为 uImage 注意选择传输协议为 Kermit:



选择完毕后即可开始传输，文件大小为 1MB 左右，大概会消耗 5 分钟左右。



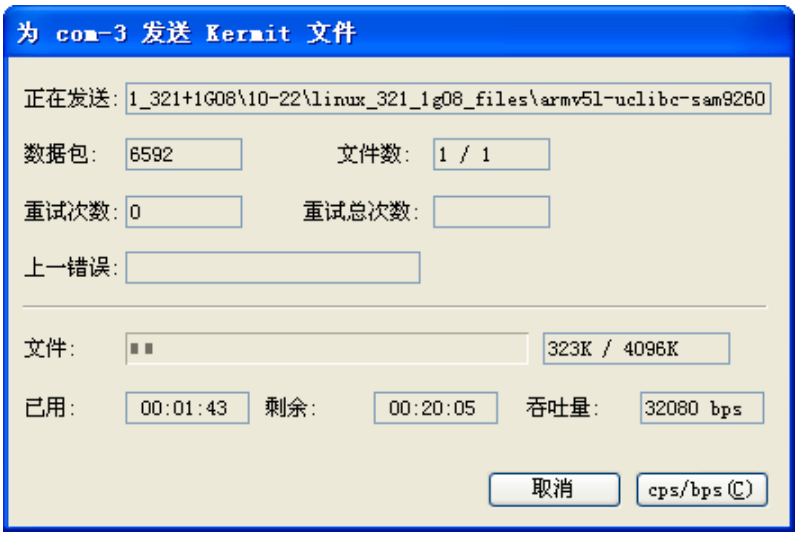
等待其传输完成，即可将其烧写到 NAND，烧写的时候注意避开 NAND 的 bad block。

输入 nand write 21500000 200000 110000，即可将刚才传输的 image 烧写到 NAND 2MB 开始的地方，为了输入方便，烧写长度直接写了个整数值。

```
Device 0 bad blocks:
U-Boot> loadb 21500000
## Ready for binary (kermit) download to 0x21500000 at 115200 bps...
## Total Size      = 0x0010cce4 = 1101028 Bytes
## Start Addr      = 0x21500000
U-Boot> nand write 21500000 200000 110000

NAND write: device 0 offset 2097152, size 1114112 ...  1114112 bytes written: OK

U-Boot> _
```

四、烧写根文件系统

继续使用 Kermit 协议传输文件系统，文件名为 armv5l-uclibc-sam9260，传输完成后将其烧写到 NAND 8MB 开始的地方，文件大小为 4MB 左右，大概耗时半小时，请耐心等待烧写完成。




```
U-Boot> loadb 21500000
## Ready for binary (kermit) download to 0x21500000 at 115200 bps...
## Total Size      = 0x00400000 = 4194304 Bytes
## Start Addr      = 0x21500000
U-Boot> nand write 21500000 800000 400000_
```

五、设置 u-boot 参数

为了正确启动系统，必须要设置一些 u-boot 的参数，下面的参数用于启动 Linux：

```
U-Boot> set ker nand read 21500000 200000 110000
U-Boot> set fs nand read 21100000 800000 400000
U-Boot> set bootcmd run boot
U-Boot> set boot run fs\;run ker\;bootm 21500000
U-Boot> set bootargs mem=64M console=ttyS0,115200 initrd=0x21100000,4M root=/dev
/ram0 rw
U-Boot>
```

如果不想输入，可以直接从本文档中复制，然后粘贴到终端：

U-Boot> set ker nand read 21500000 200000 110000

U-Boot> set fs nand read 21100000 800000 400000

U-Boot> set bootcmd run boot

U-Boot> set boot run fs\;run ker\;bootm 21500000

U-Boot> set bootargs mem=64M console=ttyS0,115200 initrd=0x21100000,4M root=/dev/ram0 rw

以下参数用于配置网络，这里以 PC 与目标板均连接到宽带路由器为例，宽带路由器连接到 Internet：

U-Boot> setenv ipaddr 192.168.1.21 ← 目标板的 IP 地址

U-Boot> setenv gateway 192.168.1.254 ← 网关

U-Boot> setenv serverip 192.168.1.20 ← 服务器地址

U-Boot> setenv netmask 255.255.255.0 ← 子网掩码

U-Boot> setenv ethaddr ??:??:??:??:??:?? ← 目标板网卡 MAC 地址，请输入有意义的值，不要使用 01:02:…这种，可以根据 PC 网卡的 MAC，修改最后一位数字。

设置完成后输入 saveenv，即可将参数保存。

六、运行 Linux

在 u-boot 下输入命令 boot，即可启动 Linux。在登录画面，输入 root 即可进入 linux 命令行模式。

使用 ifconfig 命令可以看到系统的网络设置：

```
# ifconfig -a
eth0      Link encap:Ethernet  HWaddr
          inet addr:192.168.1.21  Bcast:192.168.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:97 errors:0 dropped:0 overruns:0 frame:0
          TX packets:27 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:7310 (7.1 KiB)  TX bytes:2654 (2.5 KiB)
          Interrupt:107 Base address:0xe000

lo        Link encap:Local Loopback
          LOOPBACK  MTU:16436  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

# _
```

如果 eth0 的网络地址为无效，可以使用下面命令设置 ip 地址：

```
# ifconfig eth0 ipaddr 192.168.1.21
# ping -c 10 192.168.1.254
PING 192.168.1.254 (192.168.1.254): 56 data bytes
84 bytes from 192.168.1.254: icmp_seq=0 ttl=255 time=3.0 ms
84 bytes from 192.168.1.254: icmp_seq=1 ttl=255 time=0.9 ms
84 bytes from 192.168.1.254: icmp_seq=2 ttl=255 time=0.6 ms
84 bytes from 192.168.1.254: icmp_seq=3 ttl=255 time=0.6 ms
84 bytes from 192.168.1.254: icmp_seq=4 ttl=255 time=1.3 ms
84 bytes from 192.168.1.254: icmp_seq=5 ttl=255 time=0.8 ms
84 bytes from 192.168.1.254: icmp_seq=6 ttl=255 time=0.6 ms
84 bytes from 192.168.1.254: icmp_seq=7 ttl=255 time=0.7 ms
84 bytes from 192.168.1.254: icmp_seq=8 ttl=255 time=1.4 ms
84 bytes from 192.168.1.254: icmp_seq=9 ttl=255 time=0.9 ms

--- 192.168.1.254 ping statistics ---
10 packets transmitted, 10 packets received, 0% packet loss
round-trip min/avg/max = 0.6/1.0/3.0 ms
# _
```

为了网络 ok，还需要设置 DNS，使用如下命令：

```
# vi /etc/resolv.conf
```

该文档中有 3 个默认的 DNS，删除其中两个，剩下一个修改为网络中的 DNS，比如设置为网络中的路由器地址。关于 VI 的使用，可以查找相关资料。

```
nameserver 10.159.254.133
nameserver 10.159.254.131
nameserver 10.159.252.43
```

然后 ping 一下 mcuzone 的服务器，结果 ping 不通，需要添加 gateway：

```
# ping -c 10 www.mcuzone.com
PING www.mcuzone.com (220.189.255.38): 56 data bytes
ping: sendto: Network is unreachable
# route add default gw 192.168.1.254_
```

```
# ping -c 10 www.mcuzone.com
PING www.mcuzone.com (220.189.255.38): 56 data bytes
84 bytes from 220.189.255.38: icmp_seq=0 ttl=115 time=221.2 ms
84 bytes from 220.189.255.38: icmp_seq=1 ttl=113 time=219.6 ms
84 bytes from 220.189.255.38: icmp_seq=2 ttl=113 time=219.4 ms
84 bytes from 220.189.255.38: icmp_seq=3 ttl=113 time=218.8 ms
84 bytes from 220.189.255.38: icmp_seq=4 ttl=113 time=224.5 ms
84 bytes from 220.189.255.38: icmp_seq=5 ttl=115 time=221.8 ms
84 bytes from 220.189.255.38: icmp_seq=6 ttl=115 time=219.4 ms
84 bytes from 220.189.255.38: icmp_seq=7 ttl=115 time=218.9 ms
84 bytes from 220.189.255.38: icmp_seq=8 ttl=115 time=222.6 ms
84 bytes from 220.189.255.38: icmp_seq=9 ttl=115 time=218.0 ms

--- www.mcuzone.com ping statistics ---
10 packets transmitted, 10 packets received, 0% packet loss
round-trip min/avg/max = 218.0/220.4/224.5 ms
# _
```

然后可以通过 date 命令查看和修改时间：

```
# date
Thu Jan  1 00:42:36 UTC 1970
# date --help
BusyBox v1.1.3 (2006.07.31-20:37+0000) multi-call binary

Usage: date [OPTION]... [MMDDhhmm[[CC]YY][.ss]] [+FORMAT]

# date 102023002007
Sat Oct 20 23:00:00 UTC 2007
# _
```

注：linux ping 命令详解

功能说明：检测主机。

语　　法：ping [-dfnqrRv][-c<完成次数>][-i<间隔秒数>][-I<网络界面>][-l<前置载入>][-p<范本样式>][-s<数据包大小>][-t<存活数值>][主机名称或 IP 地址]

补充说明：执行 ping 指令会使用 ICMP 传输协议，发出要求回应的信息，若远端主机的网络功能没有问题，就会回应该信息，因而得知该主机运作正常。

参　　数：
-d 使用 Socket 的 SO_DEBUG 功能。
-c<完成次数> 设置完成要求回应的次数。
-f 极限检测。

-i<间隔秒数> 指定收发信息的间隔时间。
-I<网络界面> 使用指定的网络界面送出数据包。
-l<前置载入> 设置在送出要求信息之前，先行发出的数据包。
-n 只输出数值。
-p<范本样式> 设置填满数据包的范本样式。
-q 不显示指令执行过程，开头和结尾的相关信息除外。
-r 忽略普通的 Routing Table，直接将数据包送到远端主机上。
-R 记录路由过程。
-s<数据包大小> 设置数据包的大小。
-t<存活数值> 设置存活数值 TTL 的大小。
-v 详细显示指令的执行过程。

七、Linux 应用

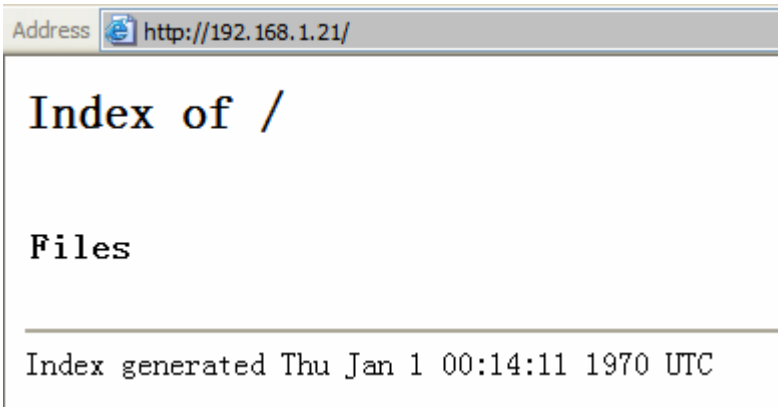
首先测试下 boa 服务器，首先要修改 boa 的 conf 文件：

```
# vi /etc/boa/boa.conf _
```

按 esc 键，然后输入:90，光标停到 90 行首，按 x 键，删除“#”，然后输入:wq，保存退出。然后就可以启动 boa：

```
# boa
#
```

在 PC 上打开一个网页，浏览目标板地址，可以看到一个网页：



但是没有内容。此时连接上 U 盘，Linux 将能识别出设备：

```
# usb 1-2: new full speed USB device using at91_ohci and address 2
usb 1-2: configuration #1 chosen from 1 choice
scsi0 : SCSI emulation for USB Mass Storage devices
scsi 0:0:0:0: Direct-Access     USB NAND FLASH DISK       0.20 PQ: 0 ANSI: 2
SCSI device sda: 128000 512-byte hdwr sectors (66 MB)
sda: Write Protect is off
sda: assuming drive cache: write through
SCSI device sda: 128000 512-byte hdwr sectors (66 MB)
sda: Write Protect is off
sda: assuming drive cache: write through
 sda:<7>usb-storage: queuecommand called
 sda1
sd 0:0:0:0: Attached scsi removable disk sda

#
```

连接成功后，可以使用下面命令查看系统的 partitions：

```
# cat /proc/partitions
major minor  #blocks  name

  31     0        256 mtdblock0
  31     1     130816 mtdblock1
   8     0      64000 sda
   8     1      63984 sda1
#
```

然后到/mnt 下建立一个文件夹并 mount usb 设备：

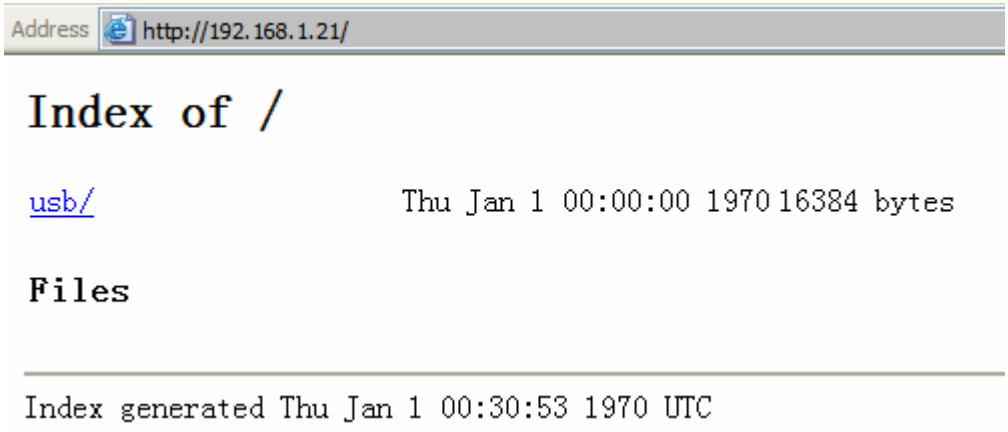
```
# cd /mnt
# mkdir usb
# mount -t vfat /dev/sda1 /mnt/usb
# cd usb
# ls
AT91SAM9260_V1.0 - SCH.pdf  res
Linux-2.6.20-v1             tst2
Ref_circuit_PI(030210).pdf  tst9261
# _
```

mount 成功后到/mnt/usb 下就可以浏览 U 盘上的文件。

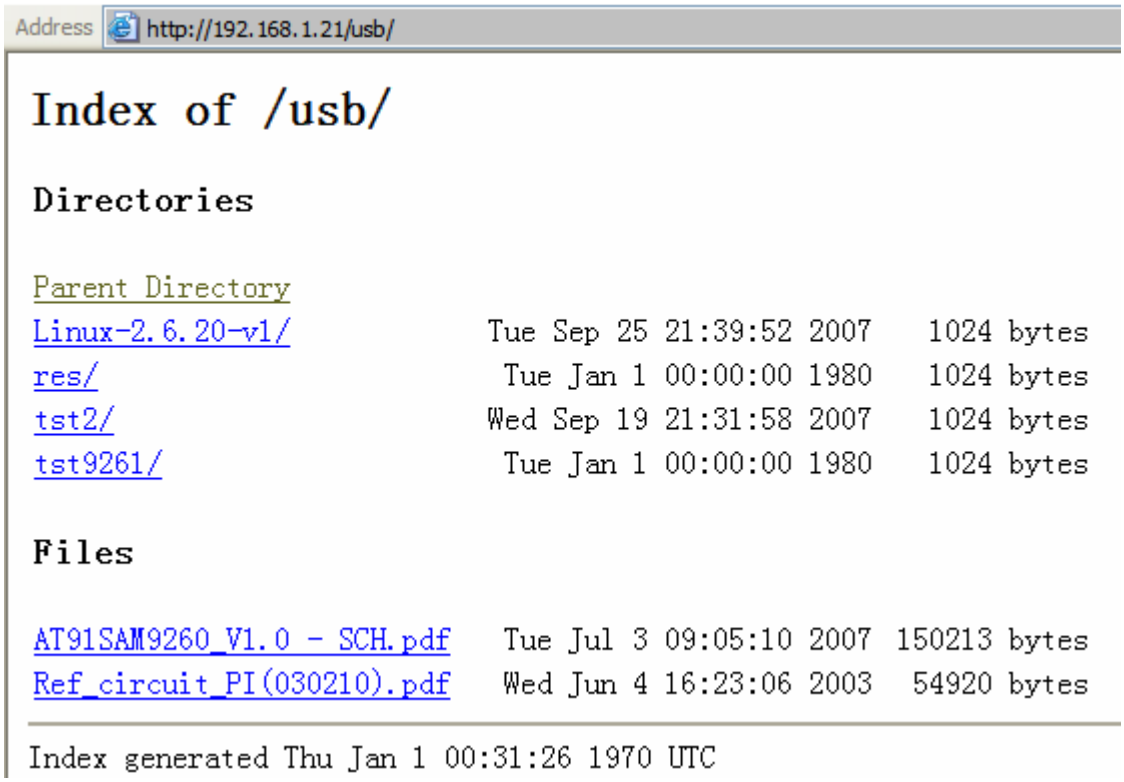
下面到 boa 的目录，建立一个 link：

```
# cd /home/httpd/html/
# ls
# ln -s /mnt/usb/ ./usb
# _
```

再打开网页，可以看到 usb 设备：

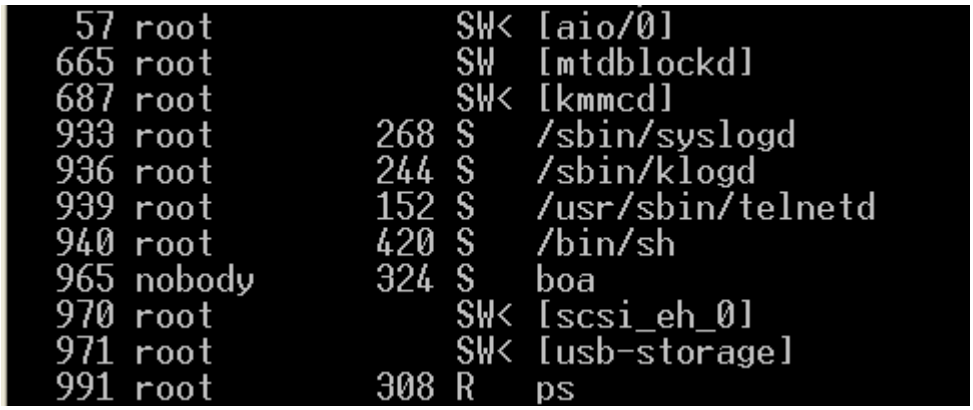


点击 usb 进去，就可以看到 U 盘上的文件：

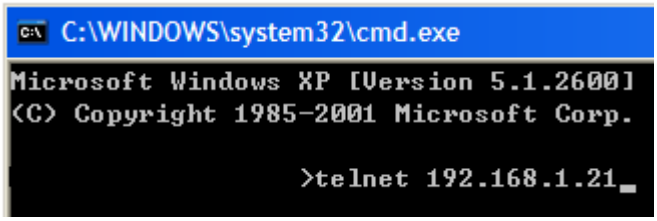


为了多用户登录，也为了方便查看信息，毕竟终端窗口浏览信息有限，可以使用 telnet 登录到目标板。

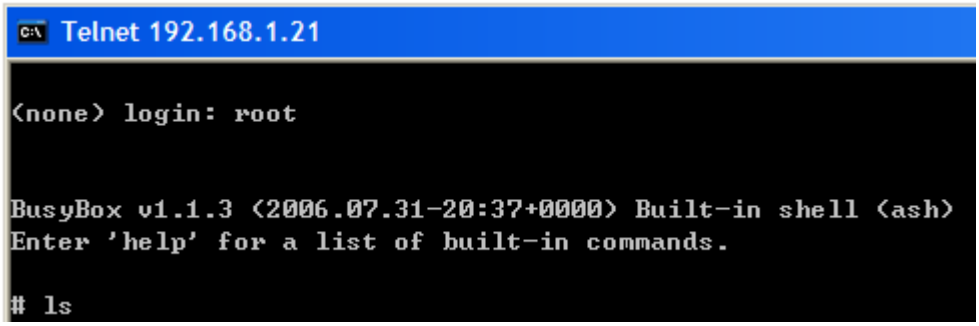
首先在 linux 下输入 ps，默认情况下 telnetd 服务已经开启：



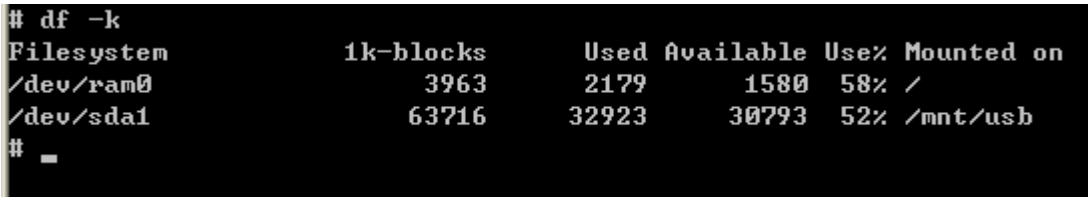
在 XP 上开一个 cmd 窗口，输入 telnet：



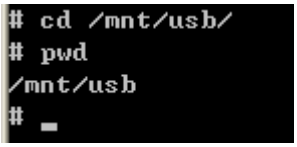
登录时输入 root 即可：



使用 df －k 命令可以看到系统中已经 mount 的文件系统信息：

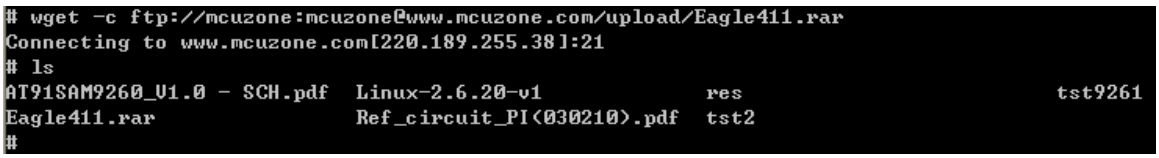


再到 usb 文件夹下下载文件：



输入下面命令，从 mcuzone 的 ftp 上下载一个文件：

等待一段时间(取决于网络连接速度)，文件下载就会完成：



注意：现在我们还提供了一个修改自 linux4sam.org 的 Linux，这个 linux 位于光盘的\Linux\Mcuzone\修改自 linux4sam，将压缩包解压后只要运行里面的.bat 文件即可完成 linux 的烧写和配置工作，重启后即可直接进入 linux 系统。系统过程大概需要花费 2－3 分钟。

第三章 为 SAM926X 编译 U-boot

本文叙述 U-boot 的编译过程，包括编译器的安装，U-boot 编译及使用。

一， 准备工作

1. 建立开发环境

请先按照《搭建基于 VPC 的 Linux 平台》一文（位于附录）建立好开发环境，使得 linux 上的工作文件夹作为一个网络驱动器，挂接到 XP 主机上。

2. 下载相关文件

请到本站 ftp(ftp://www.mcuzone.com，用户名：mcuzone 密码:mcuzone)
下载下列文件：

AT91Bootstrap1.2.zip	241 KB	WinRAR ZIP 档案...	2007-8-26 14:35
arm-linux-cross-2.95.3.tar.bz2	35,424 KB	WinRAR 档案文件	2007-6-20 11:51
arm-linux-gcc-3.4.1.tar.bz2	41,744 KB	WinRAR 档案文件	2007-6-13 15:42
arm-softfloat-linux-gnu.tar.gz	89,735 KB	WinRAR 档案文件	2007-6-13 16:47
gcc-3.3.6-glib-2.3.6.tar.bz2	79,272 KB	WinRAR 档案文件	2007-6-13 16:09
u-boot-1.1.5-atmel.tar.bz2	6,837 KB	WinRAR 档案文件	2007-3-8 11:15

在 Linux 虚拟机上以 root 登录。

在 XP 上那个网络驱动器上新建文件夹 source，将下载的文件都放到该文件夹。

如果 source 文件夹在 Linux 上创建，那么需要修改权限，使得 XP 用户可以修改。

```
[root@RH9 work]# chown -R nobody:nobody ./source/
```

在 Linux 下即可看到这些文件：

```
[root@RH9 source]# pwd
/usr/work/source
[root@RH9 source]# ls
arm-linux-cross-2.95.3.tar.bz2  arm-softfloat-linux-gnu.tar.gz  u-boot-1.1.5-atmel.tar.bz2
arm-linux-gcc-3.4.1.tar.bz2     gcc-3.3.6-glib-2.3.6.tar.bz2
[root@RH9 source]#
```

二， 准备编译

1. 展开 U-boot 源码包

到 work 文件夹下，展开 U-boot 源码包：

```
[root@RH9 work]# tar -jxvf ./source/u-boot-1.1.5-atmel.tar.bz2
```

完成后多出一个相应的文件夹：

```
[root@RH9 work]# ls
source  u-boot-1.1.5
```

修改文件夹权限，使得在 XP 端可以编辑源代码：

```
[root@RH9 work]# chown -R nobody:nobody ./u-boot-1.1.5/
```

检查一下展开出来的代码，可以看到 ATMEL 修改后的所有 board 文件夹：

```
[root@RH9 work]# ls ./u-boot-1.1.5/board/atmel/
at91rm9200df  at91rm9200dk  at91rm9200ek  at91sam9260ek  at91sam9261ek  at91sam9263ek  atstk1000
[root@RH9 work]#
```

2. 展开并编译 bootstrap

根据 SAM9261 的 boot 顺序以及 boot 的限制，必须先用一个最初的 boot 代码完成处理器的初始化及 SDRAM 的初始化，再将 U-boot 代码复制到 SDRAM 中运行。

对于 bootstrap 的代码，其大小有严格的限制，就是不能超过片内 RAM 区域的大小，二是要按照 SAM-BA 所要求的格式编写。

ATMEL 已经提供了现成的 bootstrap 代码，就是 AT91Bootstrap1.2.zip，可以阅读文档 doc6277.pdf 来了解其工作过程。这里仅以 SAM9261(U-boot 置于 dataflash)为例来说明如何编译。

注意：如无必要，就不需要自行编译 bootstrap。

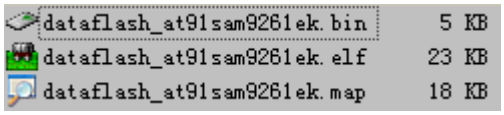
该 bootstrap 代码使用 arm-elf 工具链编译，因此需要在 PC 上安装该工具链，一个简单的方法就是安装 WinARM。

安装了 WinARM 后开启一个 cmd 窗口，使用下列命令检查安装结果：

```
C:\>arm-elf-gcc -v
Using built-in specs.
Target: arm-elf
Configured with: ../gcc-4.1.2/configure --target=arm-elf --prefix=/c/WinARM --disable-nls --disable-shared --disable-thr
eads --with-gcc --with-gnu-ld --with-gnu-as --enable-languages=c,c++ --enable-interwork --enable-multilib --with-newlib
--disable-libssp --disable-libstdcxx-pch --disable-libmudflap --disable-win32-registry --with-cpu=arm7tdmi --with-newlib
 --program-prefix=arm-elf- -v
Thread model: single
gcc version 4.1.2 (WinARM 4/2007)
```

说明安装成功，环境变量也以添加。

将 AT91Bootstrap1.2.zip 在 PC 机上展开，到 board 目录下面可以看到对应的开发板名字，进入 at91sam9261ek 文件夹，可以看到有两个配置，就是 dataflash 与 nandflash。现在先编译 dataflash 的版本，进入 dataflash 文件夹，可以看到已经编译成功的输出：



这些是 ATMEL 预先编译好的，可以直接使用。为了编译自己的代码，先将其保存到另外的文件夹。

然后在该文件夹下新建一个文本文件，文件内容为 cmd，保存后将文件改名为 bat 文件，比如 start.bat。

双击该 bat 文件，将运行一个 cmd 窗口，在窗口中输入 make 回车即可开始编译，很快就会完成，新的目标文件将生成。

现在要看一下当前文件夹下的 at91sam9261ek.h，注意下面的参数：

```
/* ********************************************************************* */
/* BootStrap Settings                                                    */
/*                                                                       */
/* ********************************************************************* */
#define AT91C_SPI_PCS_DATAFLASH   AT91C_SPI_PCS0_DATAFLASH  /* Boot on SPI NCS0 */

#define IMG_ADDRESS     0x8000      /* Image Address in DataFlash */
#define IMG_SIZE    0x32000     /* Image Size in DataFlash    */

#define MACH_TYPE       0x350       /* AT91SAM9261-EK */
#define JUMP_ADDR   0x23F00000    /* Final Jump Address         */
```

bootstrap 从 IMG_ADDRESS 开始读取代码，长度为 IMG_SIZE，并将其复制到 SDRAM 的 JUMP_ADDR 处。这些参数在 U-boot 编译和烧写的时候都需要保持一致。

3. 安装编译器

可以使用 arm-softfloat-linux-gnu-的工具链来编译 U-boot,因此需要安装该工具链。此工具链可以自行编译，也可以使用现成。自行编译可以使用 crosstool，其使用方法可以到 crosstool 的网站上看 howto。

本文以从网上编译好的工具链为例说明如何安装和使用。

由于目前还不清楚该工具链应该被安装到什么地方，因此可以在本地先将其展开：

```
[root@RH9 work]# tar -xjvf ./source/arm-softfloat-linux-gnu.tar.gz
```

完成后到新建立的文件夹下检查：

```
[root@RH9 work]# /usr/work/arm-softfloat-linux-gnu/bin/arm-softfloat-linux-gnu-gcc -v
Reading specs from /usr/work/arm-softfloat-linux-gnu/bin/../lib/gcc/arm-softfloat-linux-gnu/3.4.1
/specs
Configured with: /opt/crosstool-0.42/build/arm-softfloat-linux-gnu/gcc-3.4.1-glibc-2.3.3/gcc-3.4.
1/configure --target=arm-softfloat-linux-gnu --host=i686-host_pc-linux-gnu --prefix=/opt/crosstoo
l/gcc-3.4.1-glibc-2.3.3/arm-softfloat-linux-gnu --with-float=soft --with-headers=/opt/crosstool/g
cc-3.4.1-glibc-2.3.3/arm-softfloat-linux-gnu/arm-softfloat-linux-gnu/include --with-local-prefix=
/opt/crosstool/gcc-3.4.1-glibc-2.3.3/arm-softfloat-linux-gnu/arm-softfloat-linux-gnu --disable-nl
s --enable-threads=posix --enable-symvers=gnu --enable-__cxa_atexit --enable-languages=c,c++ --en
able-shared --enable-c99 --enable-long-long
Thread model: posix
gcc version 3.4.1
[root@RH9 work]#
```

可以看到该工具链正是使用 crosstool 编译产生。

--target 指明了目标的体系结构

--prefix 指明了安装路径

--with-headers 指明了头文件所在路径

按照上面的信息建立相关的文件夹：

```
[root@RH9 arm-softfloat-linux-gnu]# mkdir /opt/crosstool
[root@RH9 arm-softfloat-linux-gnu]# mkdir /opt/crosstool/gcc-3.4.1-glibc-2.3.3/
```

到新建的目录下重新展开编译器：

```
[root@RH9 work]# cd /opt/crosstool/gcc-3.4.1-glibc-2.3.3/
[root@RH9 gcc-3.4.1-glibc-2.3.3]# ls
[root@RH9 gcc-3.4.1-glibc-2.3.3]# tar -xjvf /usr/work/source/arm-softfloat-linux-gnu.tar.gz
```

完成后即可：

```
[root@RH9 gcc-3.4.1-glibc-2.3.3]# ls
arm-softfloat-linux-gnu
[root@RH9 gcc-3.4.1-glibc-2.3.3]#
```

原来 work 文件夹下的可以删除：

```
[root@RH9 work]# rm -Rf arm-softfloat-linux-gnu/
[root@RH9 work]# ls
source  u-boot-1.1.5
[root@RH9 work]#
```

4. 添加环境变量

为了使用方便，需要将工具链添加到用户的环境变量。可以使用如下方式：

运行：

```
[root@RH9 work]# vi ~/.bashrc
```

在文件末尾添加：

```
# Export Path
export PATH=$PATH:/opt/crosstool/gcc-3.4.1-glibc-2.3.3/arm-softfloat-linux-gnu/bin
```

保存后运行 source 使得新的设置对当前用户生效：

```
[root@RH9 work]# source ~/.bashrc
```

检查一下：

```
[root@RH9 work]# echo $PATH
/usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
:/usr/X11R6/bin:/root/bin:/opt/crosstool/gcc-3.4.1-glibc-2.3.3/arm-softfloat-linux-gnu/bin
```

环境变量设置成功，现在可以直接运行下：

```
[root@RH9 work]# arm-softfloat-linux-gnu-gcc -v
Reading specs from /opt/crosstool/gcc-3.4.1-glibc-2.3.3/arm-softfloat-linux-gnu/lib/gcc/arm-softf
loat-linux-gnu/3.4.1/specs
Configured with: /opt/crosstool-0.42/build/arm-softfloat-linux-gnu/gcc-3.4.1-glibc-2.3.3/gcc-3.4.
1/configure --target=arm-softfloat-linux-gnu --host=i686-host_pc-linux-gnu --prefix=/opt/crosstoo
l/gcc-3.4.1-glibc-2.3.3/arm-softfloat-linux-gnu --with-float=soft --with-headers=/opt/crosstool/g
cc-3.4.1-glibc-2.3.3/arm-softfloat-linux-gnu/arm-softfloat-linux-gnu/include --with-local-prefix=
/opt/crosstool/gcc-3.4.1-glibc-2.3.3/arm-softfloat-linux-gnu/arm-softfloat-linux-gnu --disable-nl
s --enable-threads=posix --enable-symvers=gnu --enable-__cxa_atexit --enable-languages=c,c++ --en
able-shared --enable-c99 --enable-long-long
Thread model: posix
gcc version 3.4.1
```

OK。

三， 编译 U-boot

1. 配置参数

进入 U-boot 源码所在文件夹：

```
[root@RH9 work]# ls
source  u-boot-1.1.5
[root@RH9 work]# cd u-boot-1.1.5/
```

检查 Makefile 中工具链的配置:

```
[root@RH9 u-boot-1.1.5]# vi Makefile
```

默认的情况下 arm 的工具链已经被 ATMEL 设置为 arm-softfloat-linux-gnu-

```
128 ifeq ($(ARCH),arm)
129 CROSS_COMPILE = arm-softfloat-linux-gnu-
130 endif
```

如果不是需要修改为这个设置。

下面需要检查 U-boot 的 link 地址，以使其与 bootstrap 一致。

```
[root@RH9 work]# cat ./u-boot-1.1.5/board/atmel/at91sam9261ek/config.mk
TEXT_BASE = 0x23f00000
```

以上输出说明 link 起始地址在 0x23F00000，与 bootstrap 一致。如果不一致，则需要这两者之一。

注意：修改代码的时候请保持文件的 UNIX 格式。

2. 编译

下面以 9261 板子为例来说明如何编译。

运行：

```
[root@RH9 u-boot-1.1.5]# make at91sam9261ek_config
Configuring for at91sam9261ek board...
[root@RH9 u-boot-1.1.5]#
```

完成对 9261ek 板子的设置，注意_config 之前的 board 名字必须与 board 文件夹下的一致。

配置完成后即可进行编译：

```
[root@RH9 u-boot-1.1.5]# make
```

几分钟之后编译即会完成：

```
arm-softfloat-linux-gnu-objcopy --gap-fill=0xff -O srec u-boot u-boot.srec
arm-softfloat-linux-gnu-objcopy --gap-fill=0xff -O binary u-boot u-boot.bin
gzip -c u-boot.bin > u-boot-at91sam9261ek.gz
cp u-boot-at91sam9261ek.gz /tftpboot/u-boot-at91sam9261ek-2007-08-26.gz
cp u-boot.bin /tftpboot/u-boot-at91sam9261ek-2007-08-26.bin
```

主要生成下列文件：

```
-rwxr-xr-x    1 root     root       531846 Aug 26 12:54 u-boot
-rwxr-xr-x    1 root     root       191728 Aug 26 12:54 u-boot.bin
-rw-r--r--    1 root     root       109033 Aug 26 12:54 u-boot.map
```

其中 u-boot 是一个 elf 文件，可以用于调试：

```
[root@RH9 u-boot-1.1.5]# file u-boot
u-boot: ELF 32-bit LSB executable, ARM, version 1 (ARM), statically linked, not stripped
```

u-boot.bin 是 binary 格式的输出文件，可以直接到板子上运行。

u-boot.ma 是 map 文件，提供了目标文件的一些信息。

同时，生产的 bin 文件会被复制到/tftpboot 下以方便使用 tftp 下载测试。

四， 使用 U-boot

先在 PC 上安装 SAM-BA，这样可以使用其进行下载。

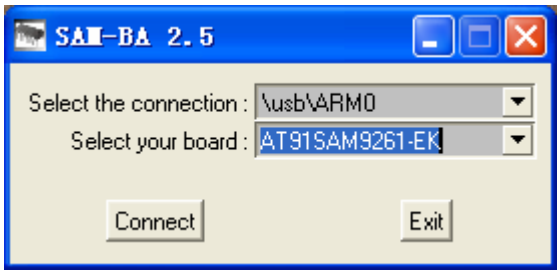
现以核心板的 JTAG 插座在左侧来描述：

找到核心板左上角的 J4，将 J4 右边的那个开关拨到上方，使得板子从内部 ROM 启动；

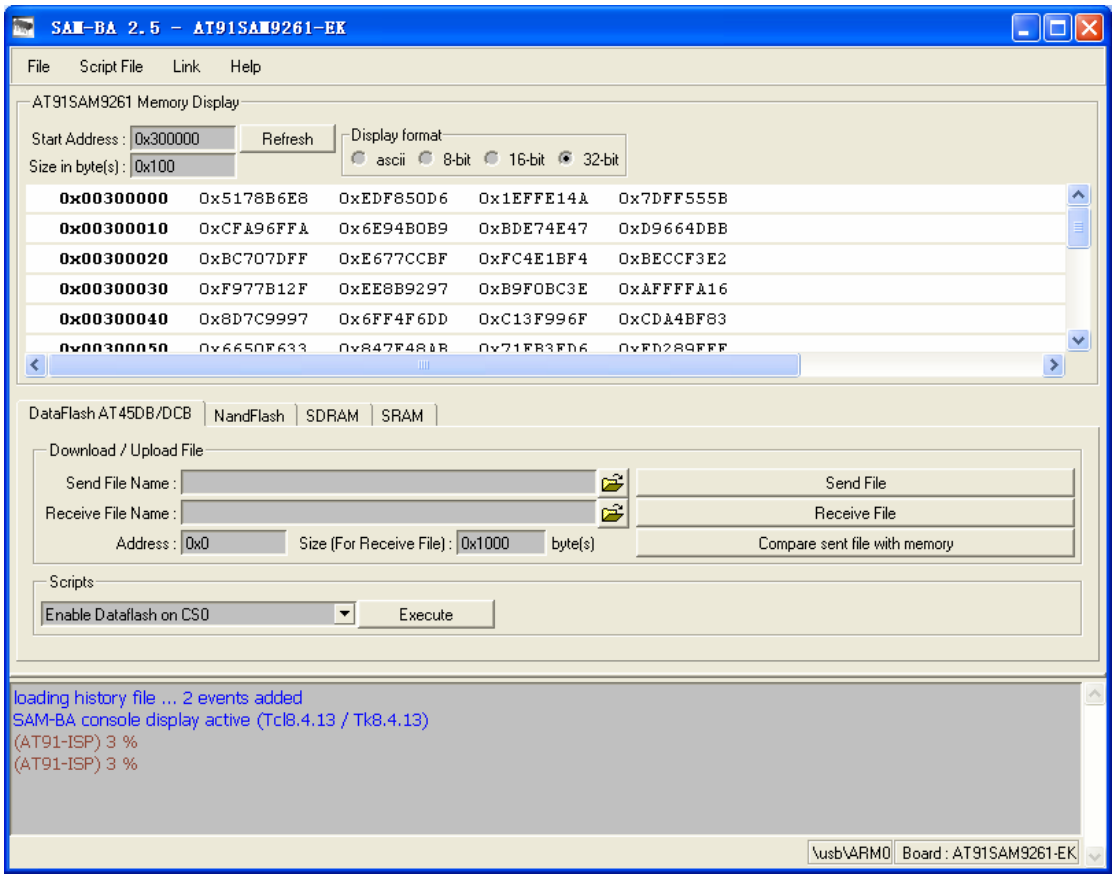
找到核心板右下角的 J21，将其下面的那个开关拨到右方，断开 dataflash 的 CS。

使用 mini USB 线连接 PC 与 9261 开发板，PC 端将出现一个新的 USB 设备，如果安装驱动，一路 next 即可。

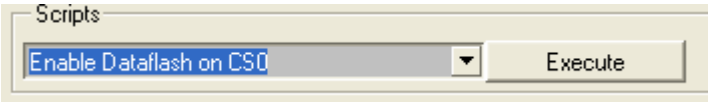
运行 SAM-BA,等待一段时间，在 SAM-BA 窗口中如下选择：



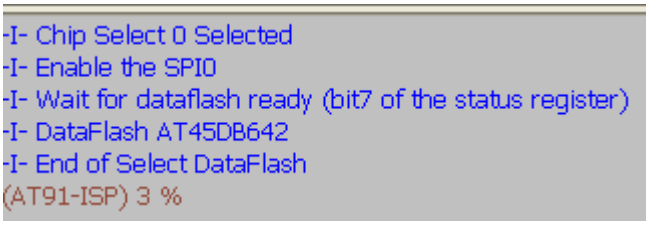
选择 connect，即会出现主界面：



此时再将 J21 下方的开关拨到左侧，运行下面的脚本：

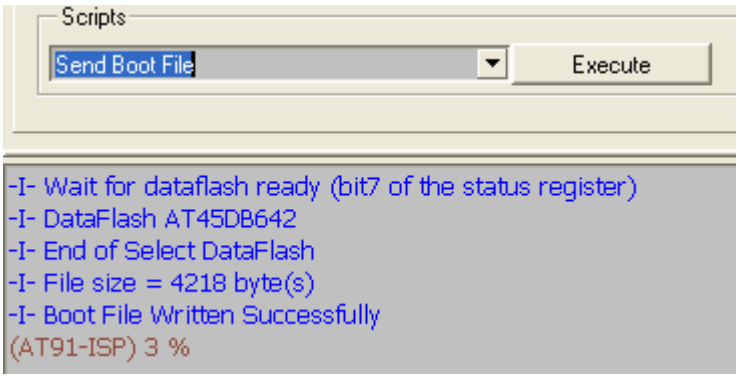


下面是正确的输出信息：

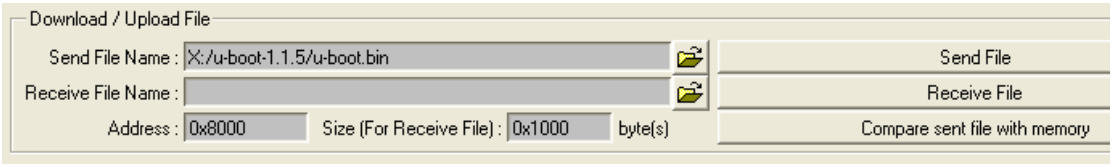


AT45DB642 已被正确探测到。

下面选择发送启动文件，在弹出的对话框中选择 ATMEL 提供的基于 dataflash 的 bootstrap，点击 execute 即可烧写 boot 文件。脚本会在保留位置填写对应的文件长度等信息。



再选择刚才编译好的 u-boot.bin，地址要选择 0x8000：

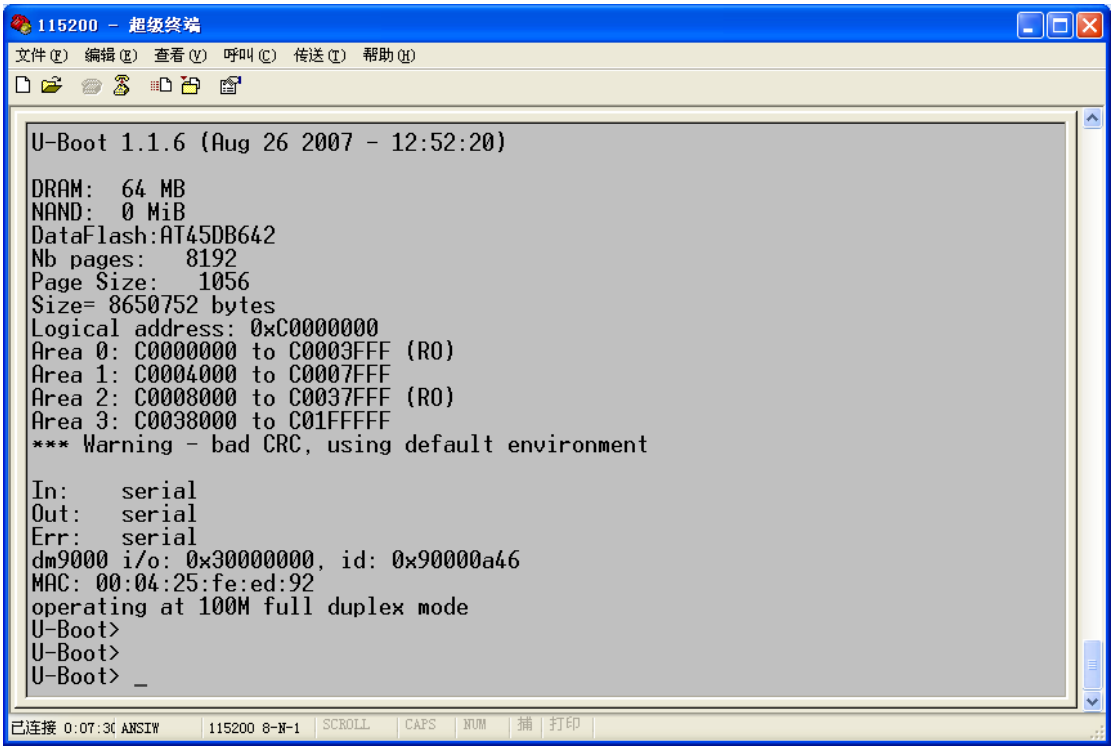


点击 Send File，文件就将被下载到 dataflash 上 0x8000 地址开始的地方。

完成后可以关闭 SAM-BA，用 USB 线连接底板的 J4 与 PC，安装好 USB 转串口的驱动程序，PC 上多一个串口，在设备管理器中记录下串口号。

打开一个超级中断，使用上面那个串口，设置为 115200,n,8,1。

拔掉 mini USB 线，然后再联接上，中断窗口里可以看到 U-boot 的启动画面：



之所以出现 bad CRC，原因在于没有在存储区域找到环境变量。

第四章：为 SAM926X 编译 Linux

本章以 9261 为例，说明 SAM926x Linux 的编译过程，Linux 版本选择了 2.6.20，开发环境选择了 Virtual PC，其它软件可以类推。

一，准备工作

1. 使用 Virtual PC 安装 Linux 虚拟机

   请参照本站另一篇文档《基于 VPC 建立 ARM_Linux 开发环境》。

2. 下载相关软件包

   下载下列软件(可在本站 ftp 上下载)，并传输到 Linux 虚拟机下。

   linux-2.6.20-atmel9261-nfs.tar.bz2

   arm-linux-gcc-3.4.1.tar.bz2

   将后者展开到/usr/local/arm 下，

   tar xjvf arm-linux-gcc-3.4.1.tar.bz2

   并添加其路径到系统路径:

   在~/.bashrc 文件(使用 bash shell)中添加/usr/local/arm/3.4.1/bin

```
# Export Path
export PATH=$PATH:/opt/crosstool/gcc-3.4.1-glibc-2.3.3/arm-softfloat-linux-gnu/bin
export PATH=$PATH:/usr/local/arm/3.4.1/bin
[root@RH9 linux-2.6.20]#
```

   然后运行 source ~/.bashrc，新改动立即生效。

   可以运行些相关的命令检查效果。

```
[root@RH9 linux-2.6.20]# arm-linux-gcc
arm-linux-gcc: no input files
[root@RH9 linux-2.6.20]# arm-linux-gcc -v
Reading specs from /usr/local/arm/3.4.1/lib/gcc/arm-linux/3.4.1/specs
Configured with: /work/crosstool-0.27/build/arm-linux/gcc-3.4.1-glibc-2.3.2/gcc-3.4.1/configure -
-target=arm-linux --host=i686-host_pc-linux-gnu --prefix=/usr/local/arm/3.4.1 --with-headers=/usr
/local/arm/3.4.1/arm-linux/include --with-local-prefix=/usr/local/arm/3.4.1/arm-linux --disable-n
ls --enable-threads=posix --enable-symvers=gnu --enable-__cxa_atexit --enable-languages=c,c++ --e
nable-shared --enable-c99 --enable-long-long
Thread model: posix
gcc version 3.4.1
[root@RH9 linux-2.6.20]# which arm-linux-gcc
/usr/local/arm/3.4.1/bin/arm-linux-gcc
[root@RH9 linux-2.6.20]#
```

   为了方便，在/usr 下建立新文件夹，名为 work，将 linux 源代码放置到该路径下，然后运行命令展开压缩包:

   cd /usr/work

   tar xjvf linux-2.6.20-atmel9261-nfs.tar.bz2

   解压缩完成后将在当前路径下生成 linux-2.6.20 文件夹。

3. 完成编译 U-boot 的工作

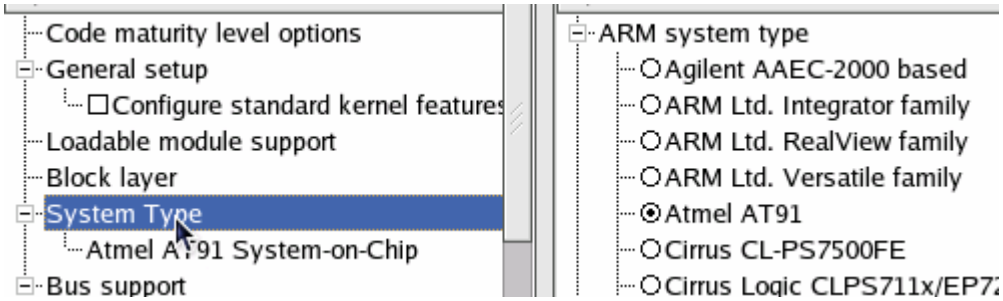
   请参照本站另一篇文章《为 SAM926X 编译 U-boot》为 9261 编译好 u-boot。

二，编译 Linux

5. 配置 Linux

在编译之前，必须要对 Linux 进行相应的配置，以使得 Linux 适应目标板，并对相关功能进行裁减。

运行 make xconfig，检查 System Type：

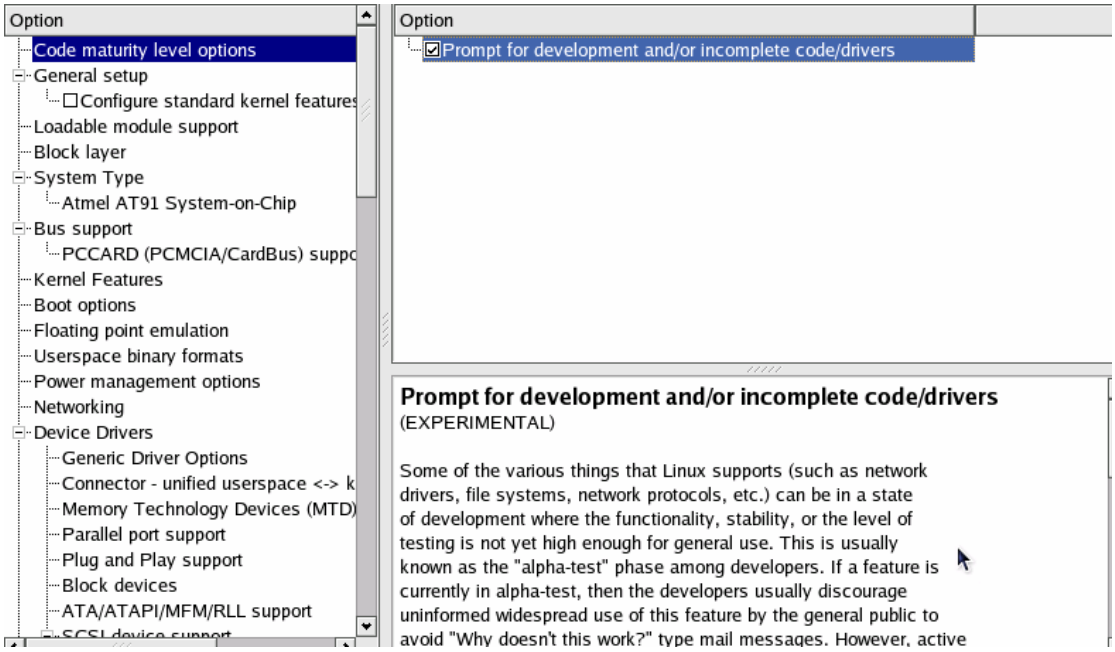


如果不是 ARM 类型的，请退出，然后输入下面命令：

Make ARCH=arm CROSS_COMPILE=arm-linux- xconfig

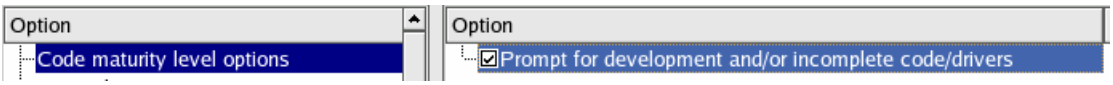
上面命令将设置当前编译的目标器件和使用的工具链。然后即可以对每项进行配置。

点击左侧的大项，右侧就会出现可以配置的小项目，同时下方出现该项的解释。

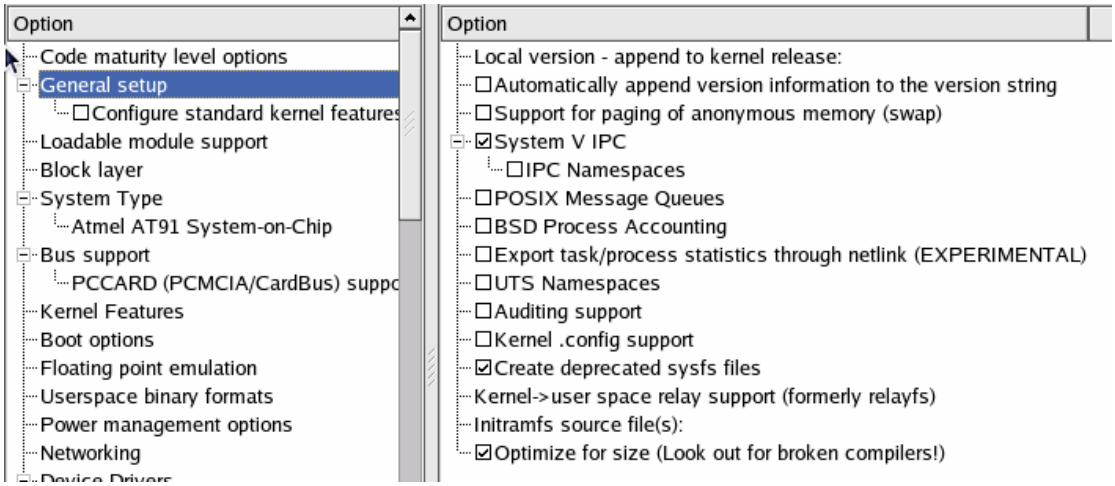


将方框勾选，将使得该项目被编译入内核；不勾选，则不选用该功能；如果是一个点号，则该功能被编译为模块，可以动态加载。

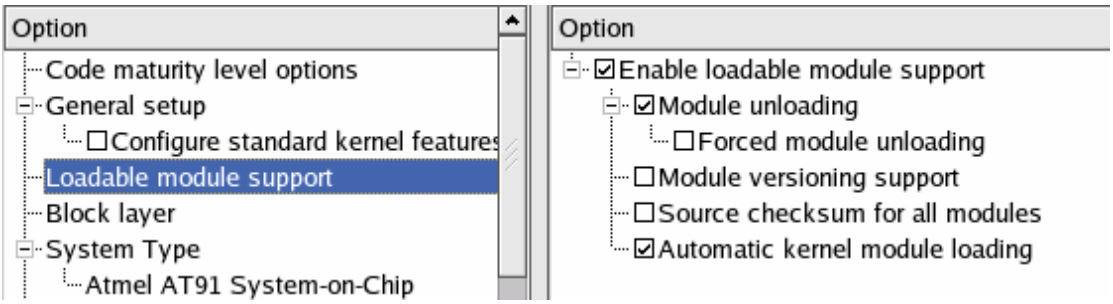
下面简要介绍下有关的一些选项。**注意：**所有图片仅供参考。



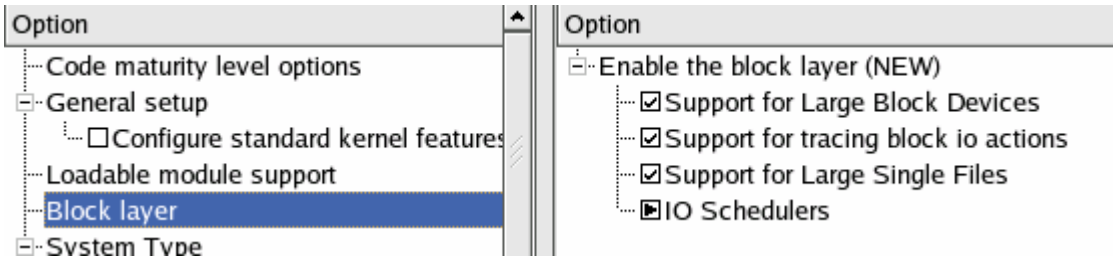
选中该选项将会编译一些还处于开发状态的代码。



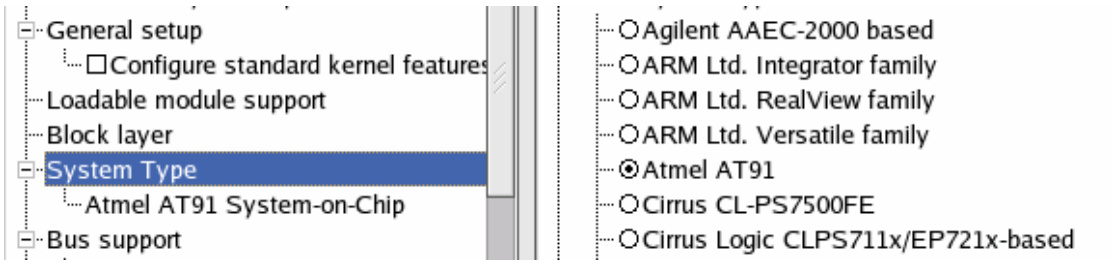
此菜单包含一些通用的配置选项。



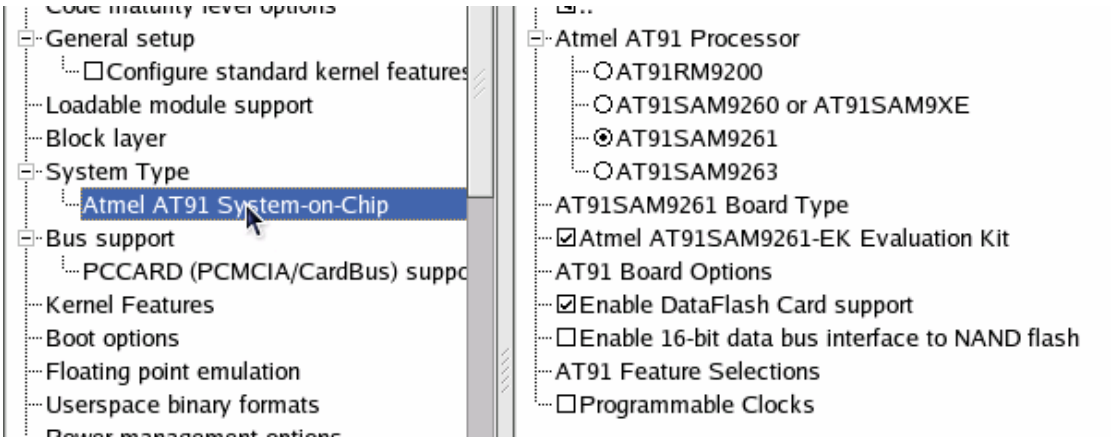
此菜单包含支持动态加载模块的一些配置选项。



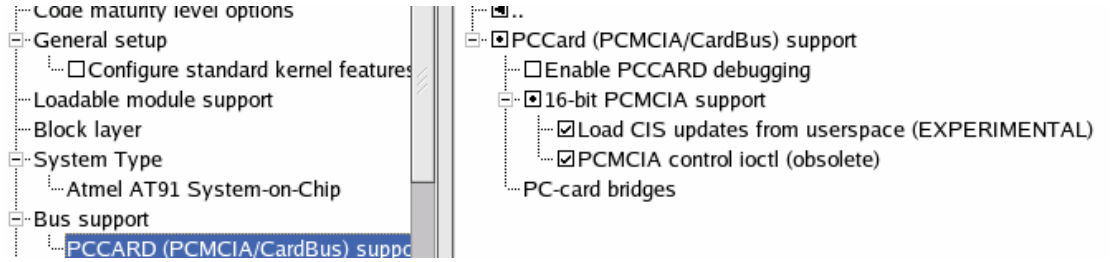
此菜单可以设置对 block 设备的一些选项。



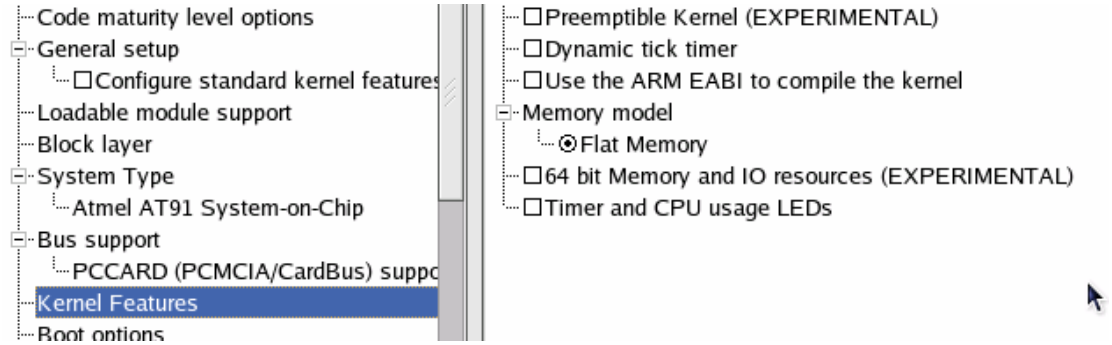
此菜单可以指定选用的 ARM 处理器，选择了 AT91 后，还可以进行一些具体设置：



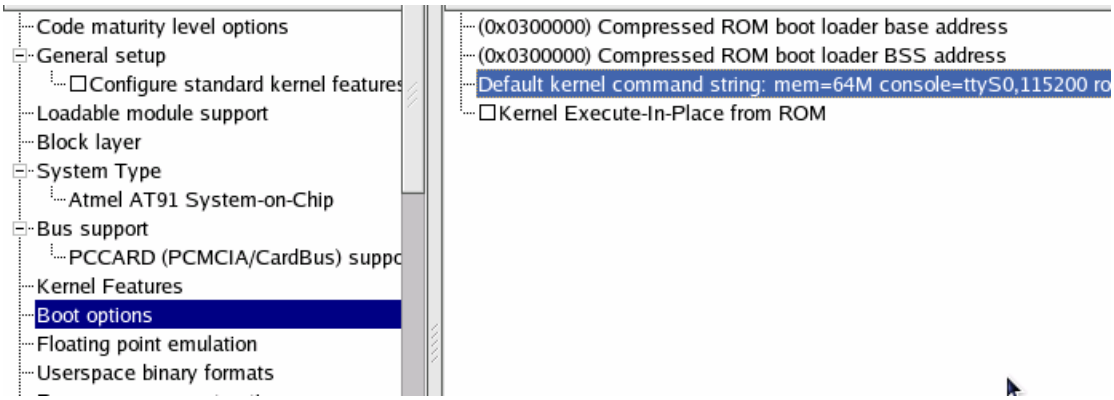
这里可以选择 AT91 的具体型号，以及一些相关的设置。



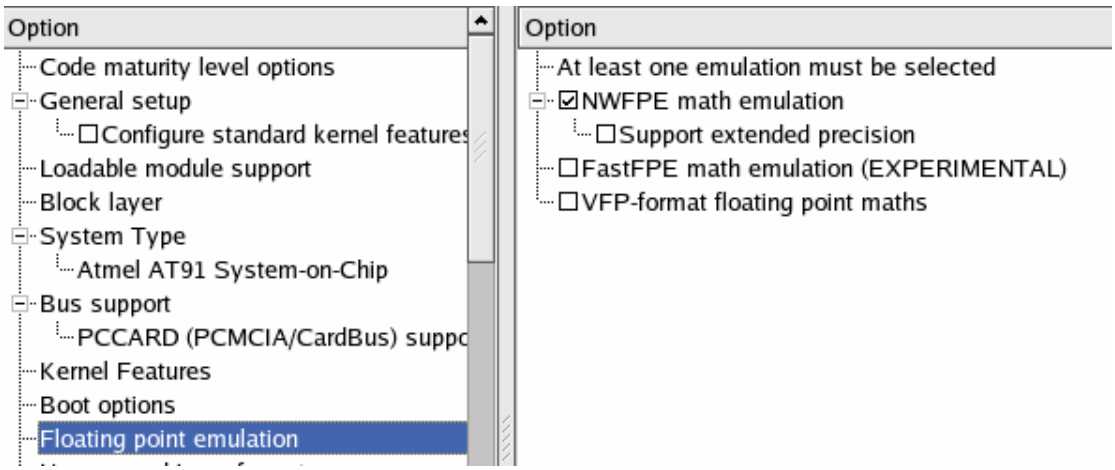
此菜单可以配置系统中的各种总线。



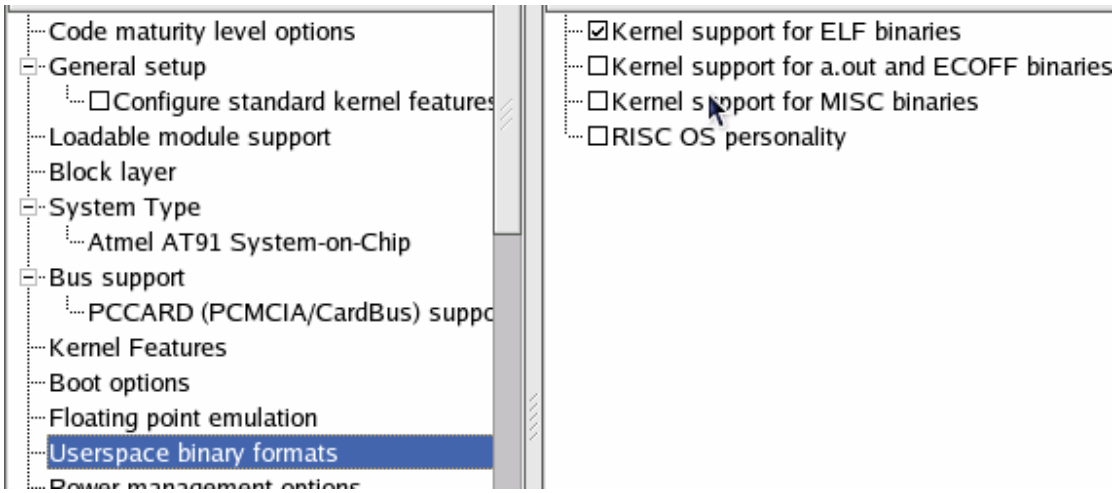
此菜单包含内核特性的相关选项。



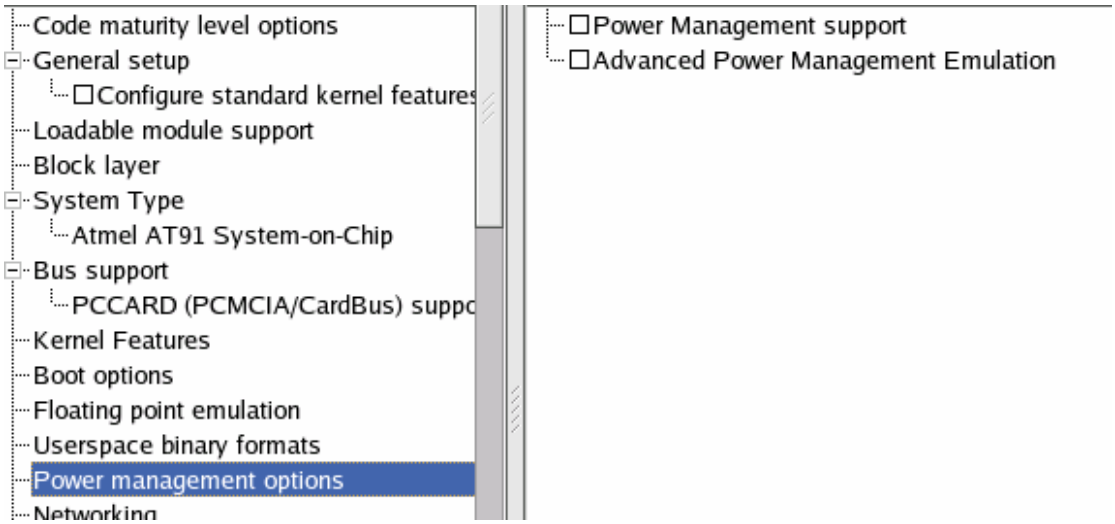
此菜单包含内核启动相关的选项。



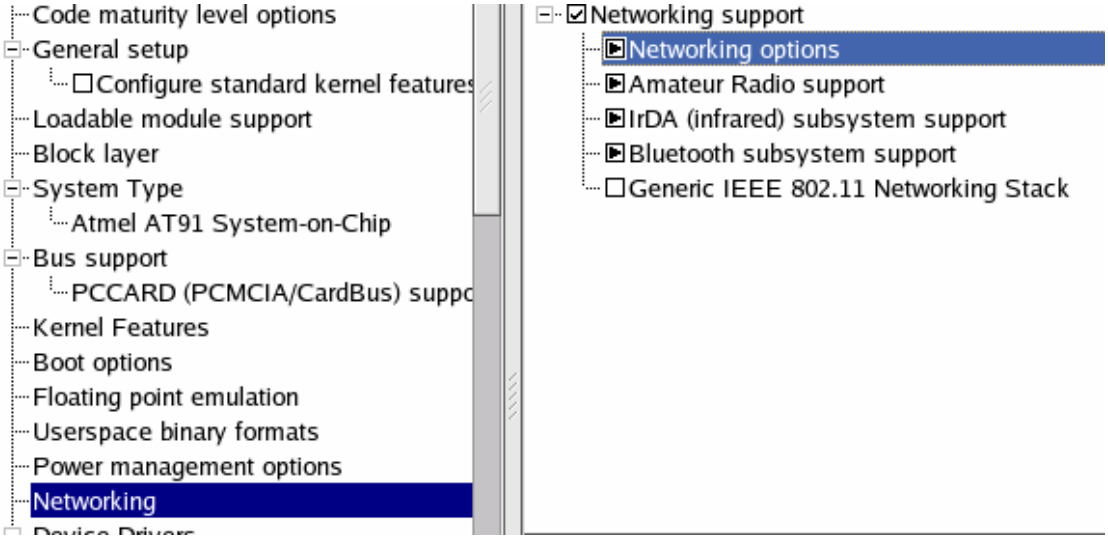
此菜单提供对浮点运算进行模拟实现的相关选项。



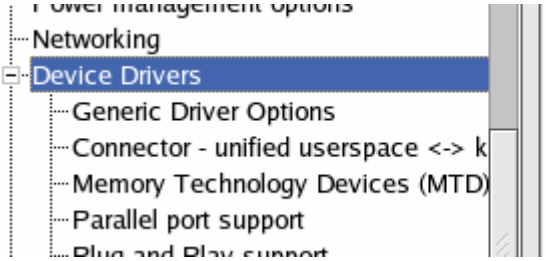
此菜单可设置所支持的应用程序的格式。



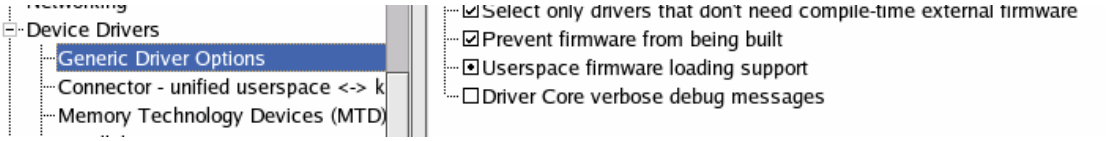
此菜单包含电源管理相关的选项。



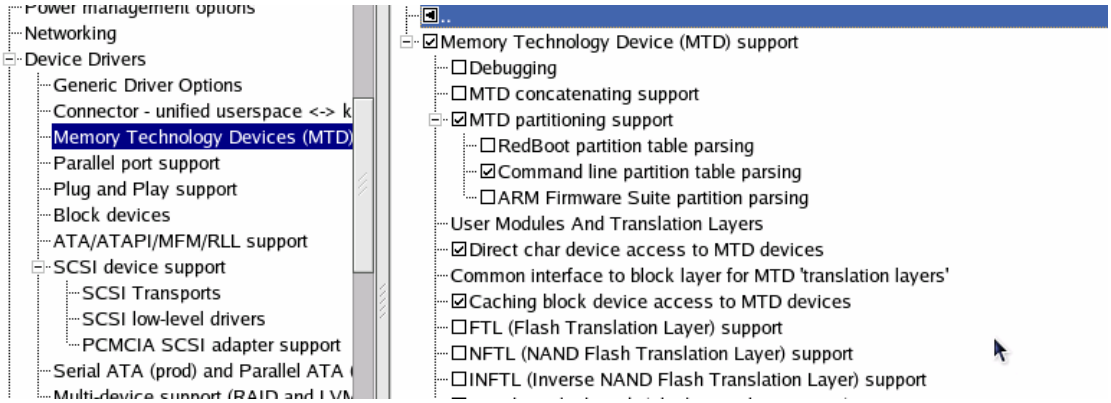
此菜单可以设置所支持的网络协议。点击没项左侧的箭头还会出现详细的设置菜单。



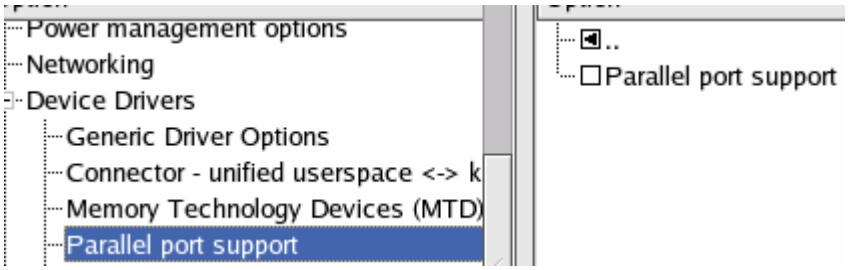
此菜单可以配置各种设备驱动程序。其包含很多子菜单，可以对不同的设备进行配置：



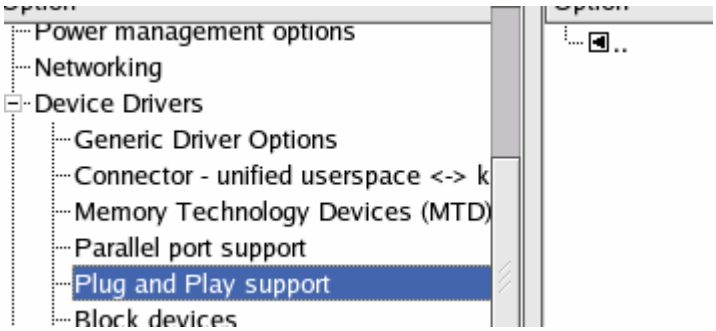
此菜单对应 drivers/base 目录的配置选项，包含 Linux 驱动程序基本和通用的配置。



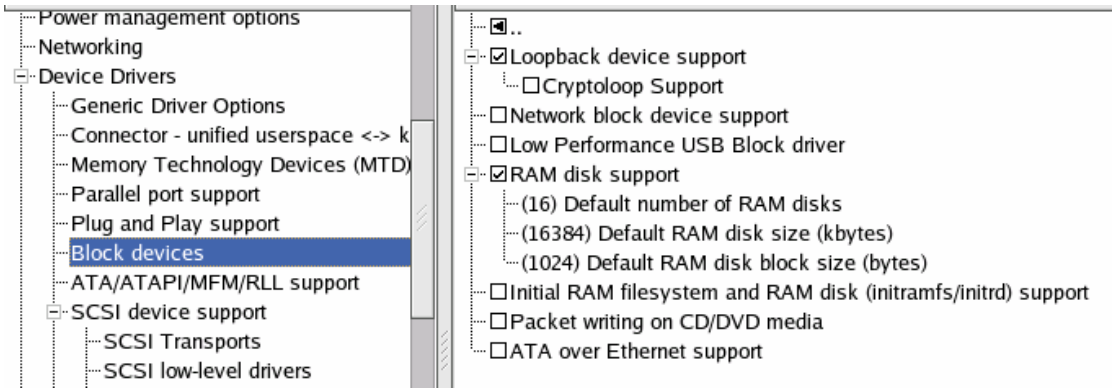
此菜单对应 drivers/mtd 目录的配置选项，包含 MTD 设备驱动的配置选项。



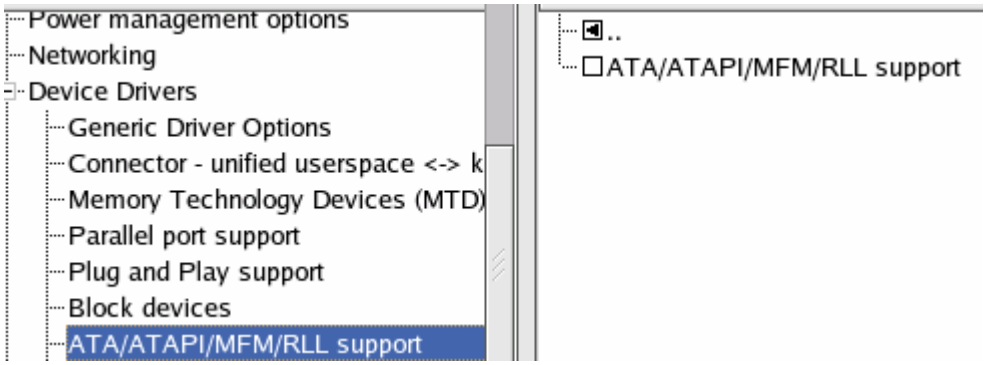
此菜单对应 drivers/parport 目录的配置选项，包含并口设备的驱动程序。



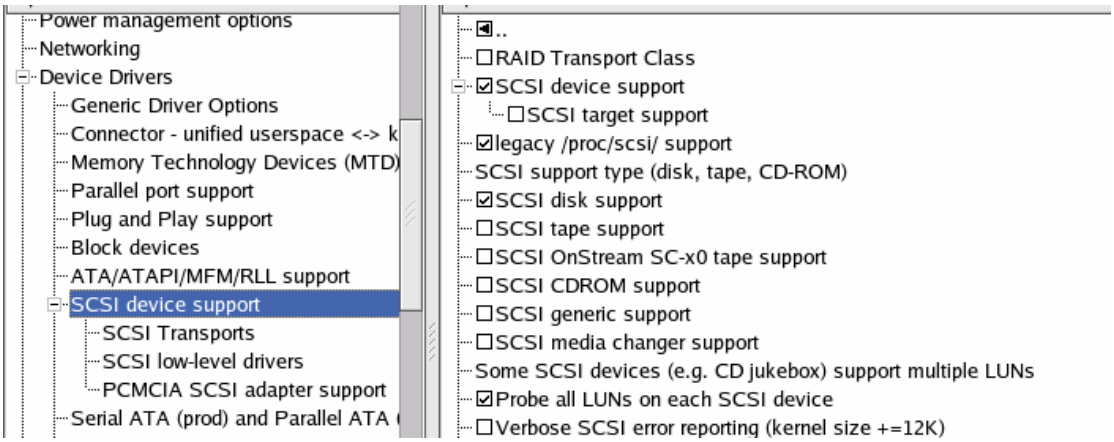
此菜单对应 drivers/pnp 目录的配置选项，包含计算机外围设备的热插拔功能。



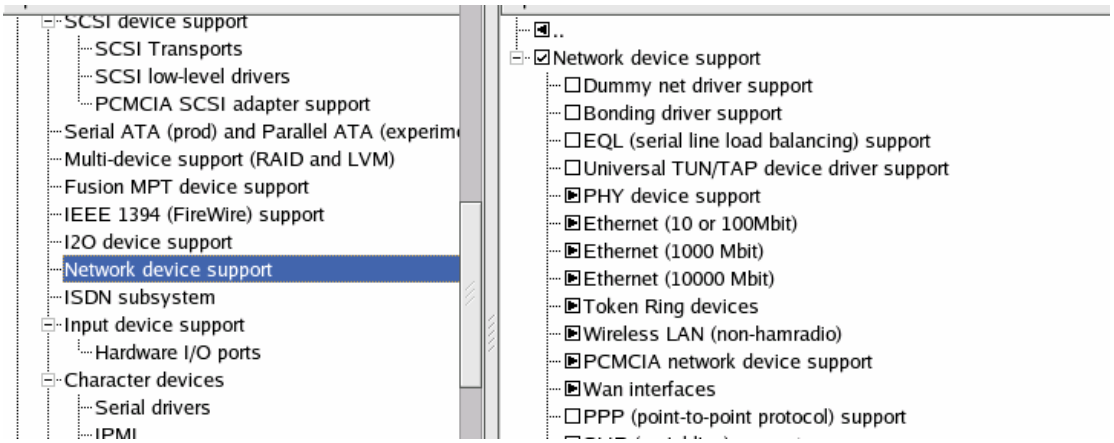
此菜单对应 drivers/block 目录的配置选项，包含 RAMDISK 等驱动程序。



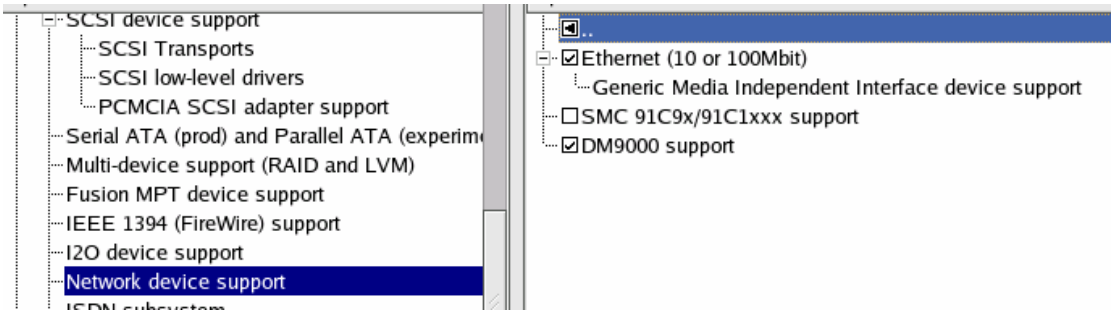
此菜单对应 drivers/ide 目录的配置选项，包含各种 ATA/ATAPI 设备的驱动。

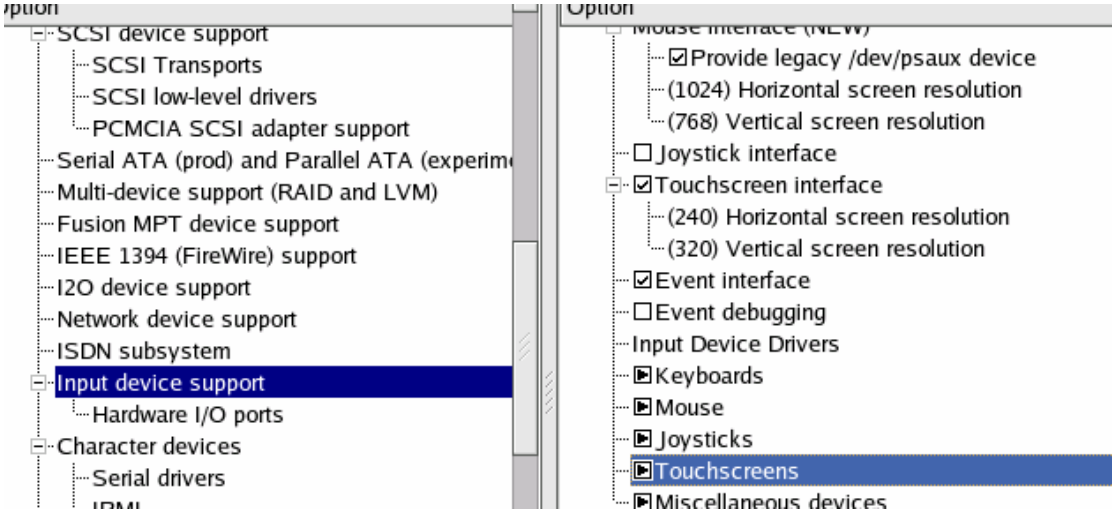


此菜单对应 drivers/scsi 目录的配置选项，包含各类 SCSI 接口设备的驱动程序。

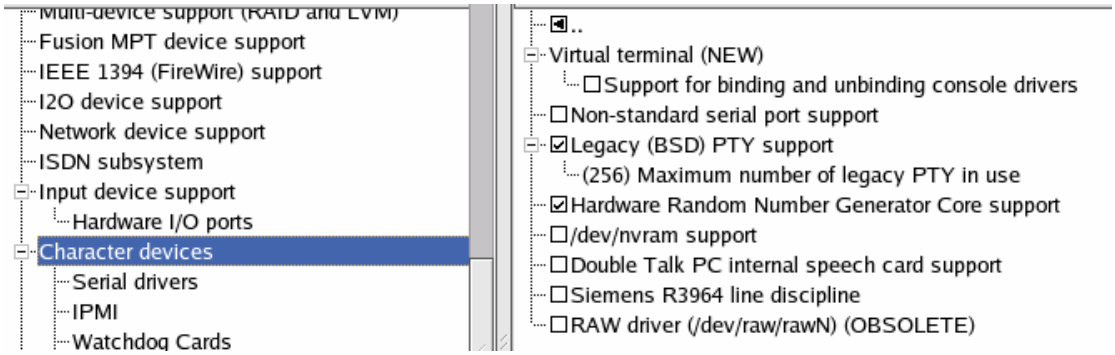


此菜单对应 drivers/net 目录的配置选项，包含各类网络设备的驱动程序。针对 9261 板子的配置，需要配置 DM9000 的驱动:

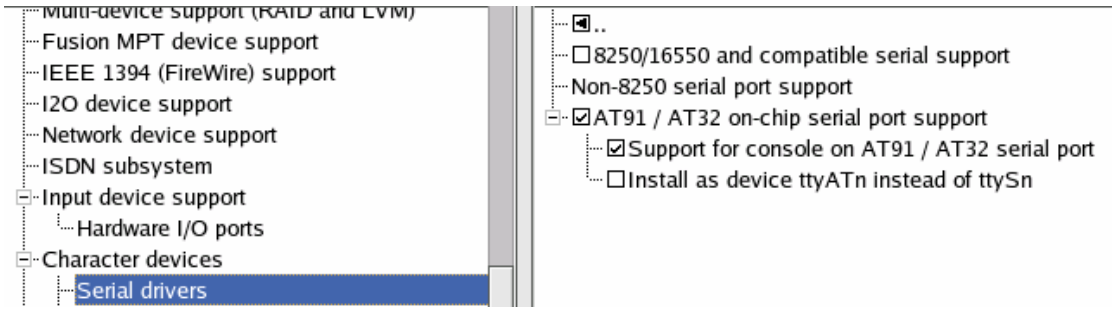


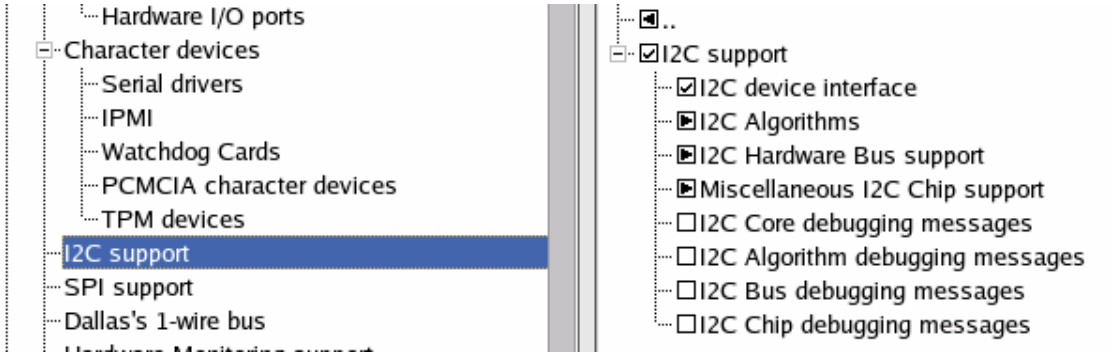


此菜单对应 drivers/input 目录的配置选项，包含各种输入设备的驱动程序，比如对 touchscreen 的支持。



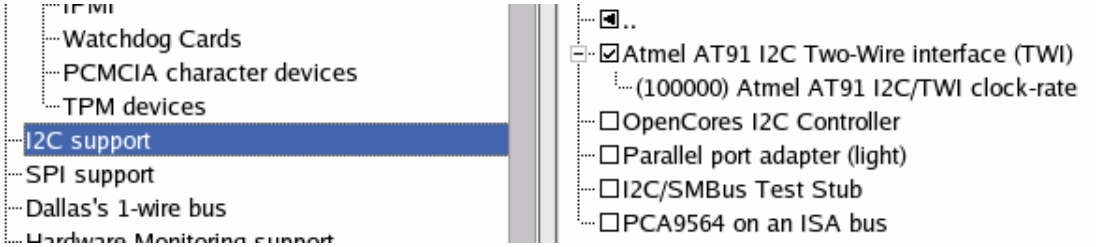
此菜单对应 drivers/char 目录的配置选项，包含各种字符设备的驱动程序。串口即是一种字符设备，可以在此菜单下设置，但其对应的目录为 drivers/serial。

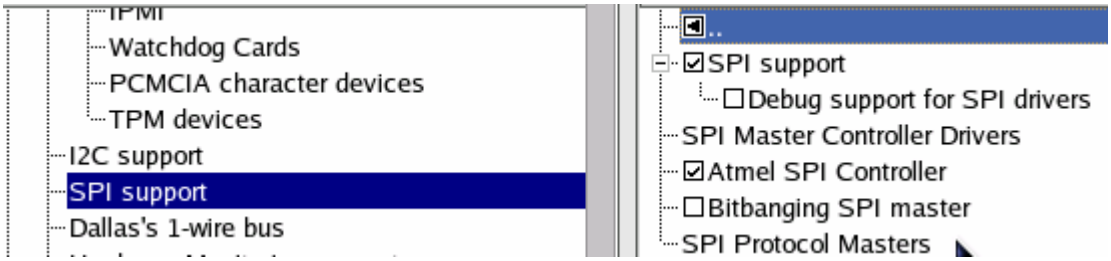




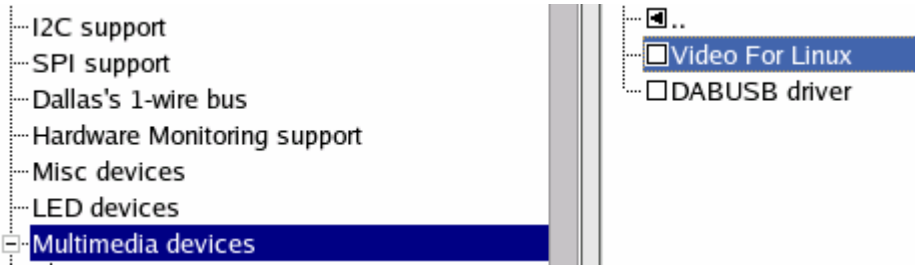
此菜单对应 drivers/i2c 目录的配置选项，包含 I2C 设备的驱动。可以为 9261 选择

对应的配置。

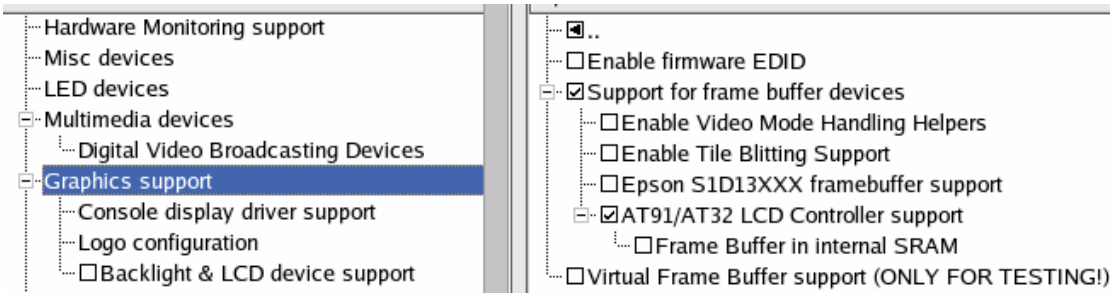




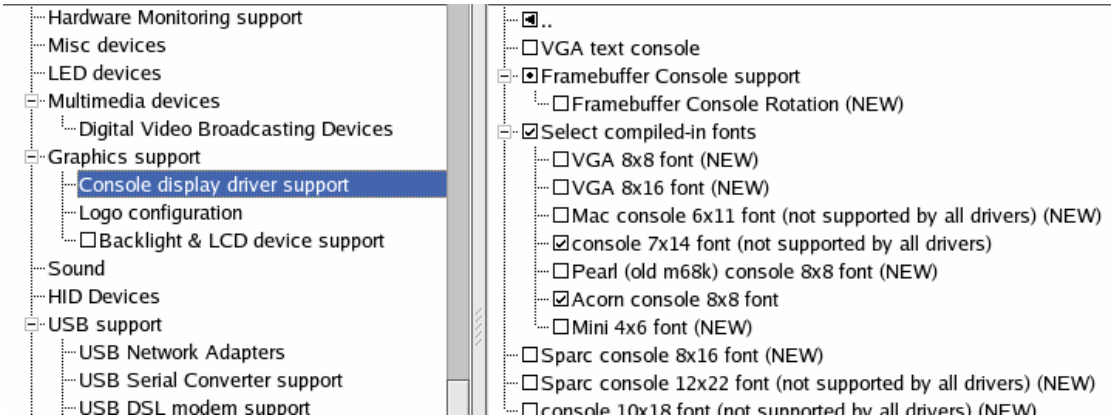
此菜单对应 drivers/spi 目录的相关配置，包含 SPI 接口设备的驱动。

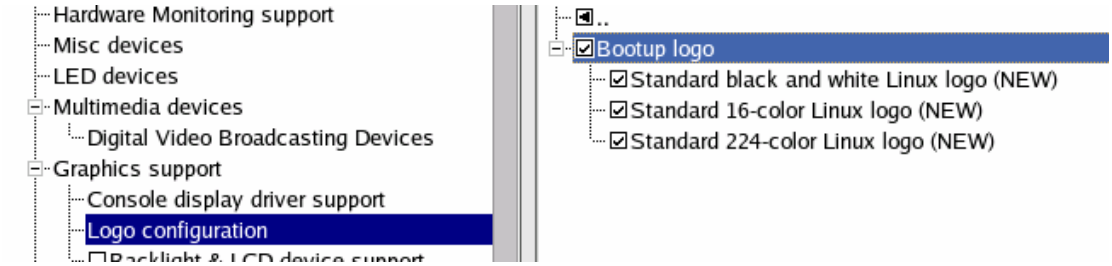


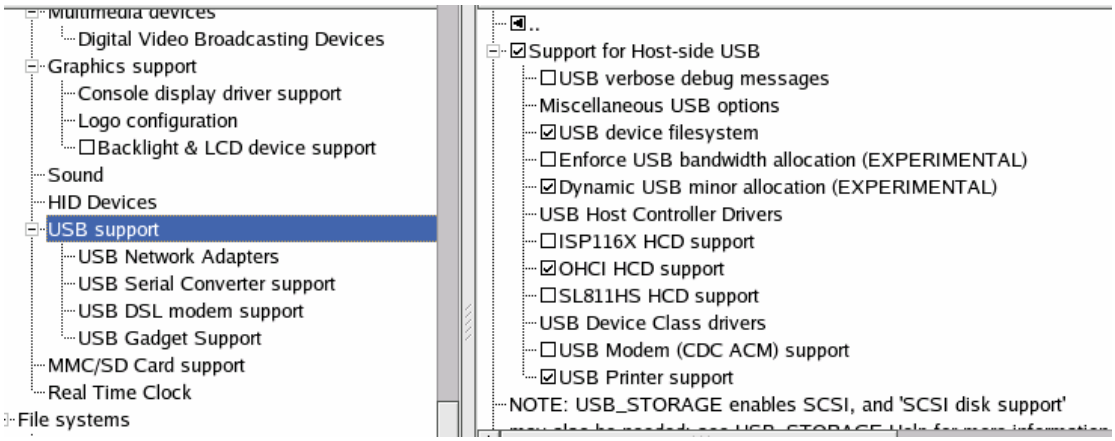
此菜单对应 drivers/media 目录的配置选项，提供视频/音频接收设备和摄像头的驱动程序。



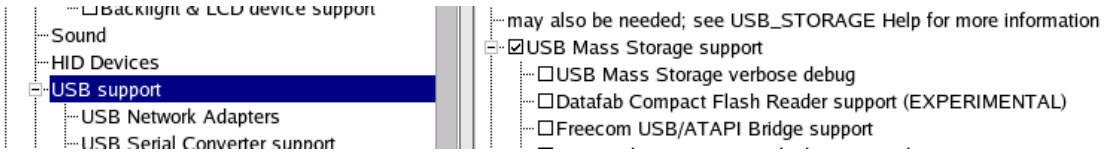
此菜单对应 drivers/video 目录的配置选项，包含 FrameBuffer 的驱动程序。在其子菜单中可以配置 display console 和 logo。

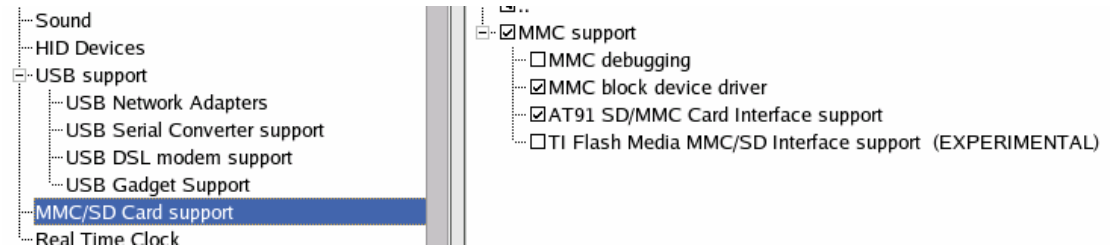




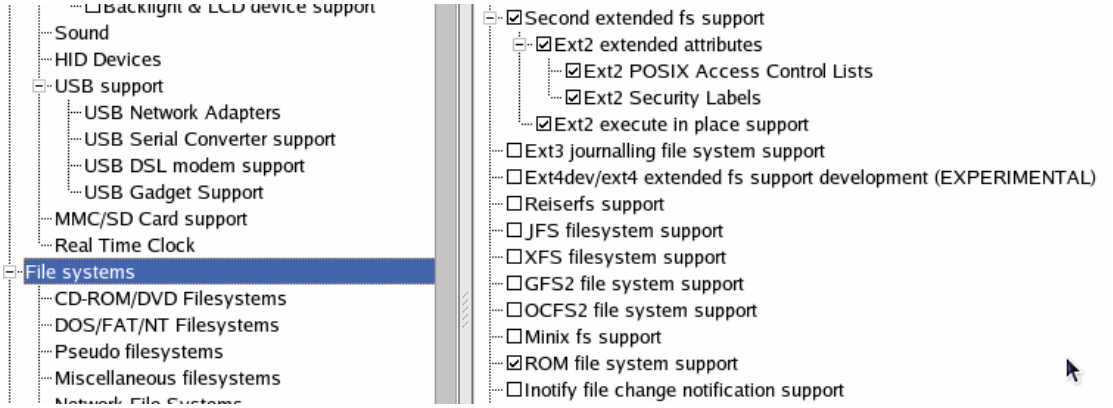


此菜单对应 drivers/usb 目录的配置选项，提供 USB HOST 和 DEVICE 设备的支持。比如 U 盘：

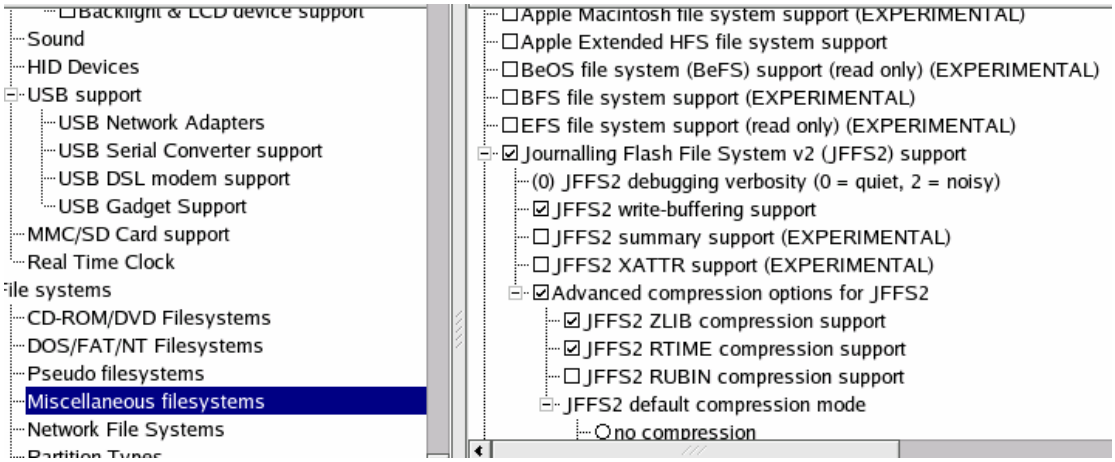




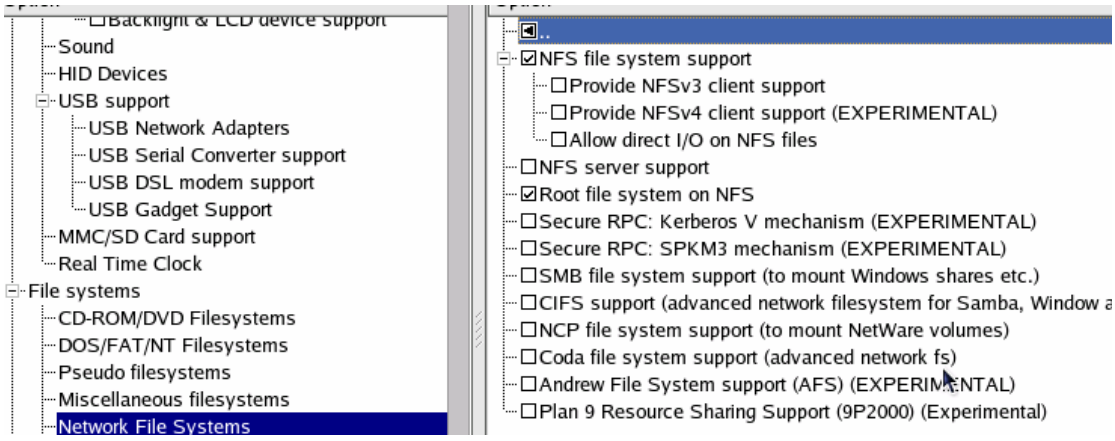
此菜单对应 drivers/mmc 目录的配置选项，包含对 MMC/SD 卡的支持。

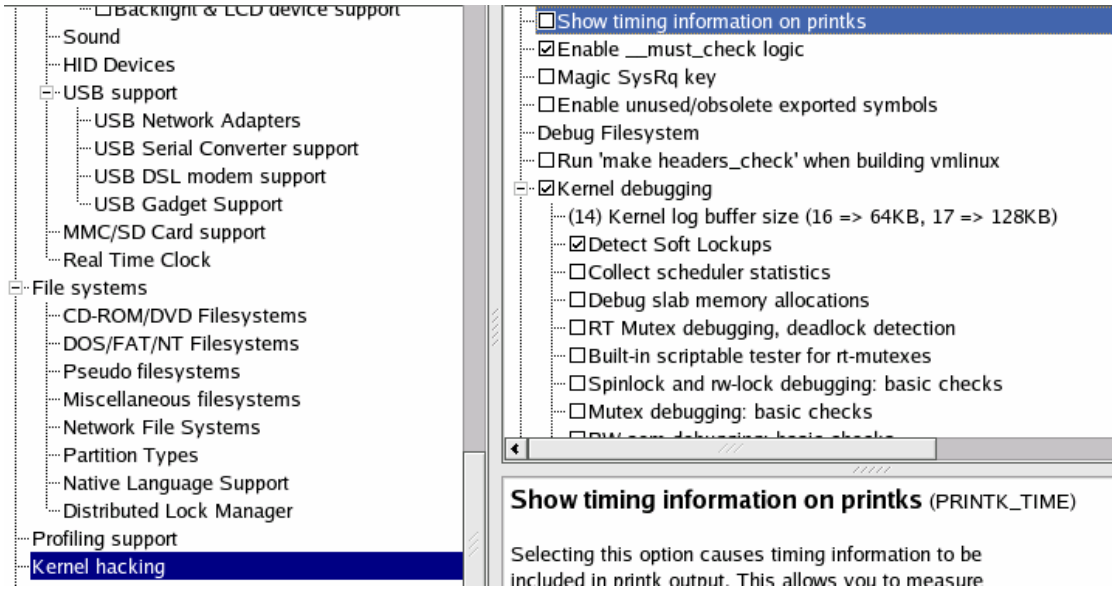


此菜单包含各种文件系统支持的选项。可以设置 JFFS2 的支持：

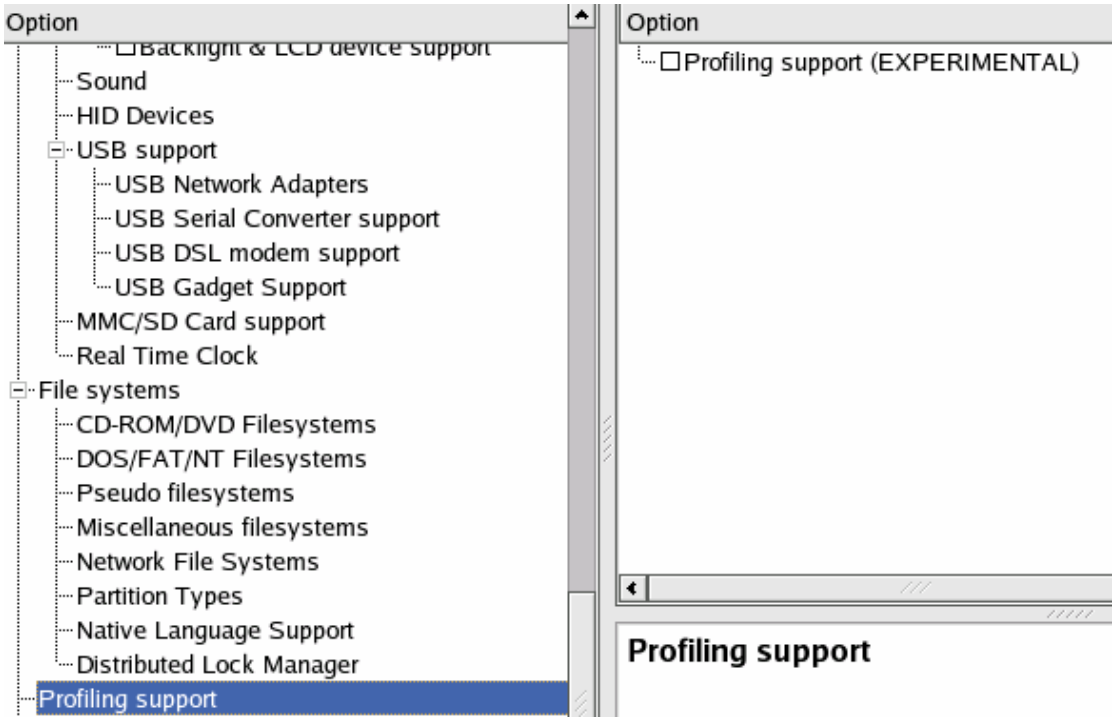


以及 NFS 的支持：

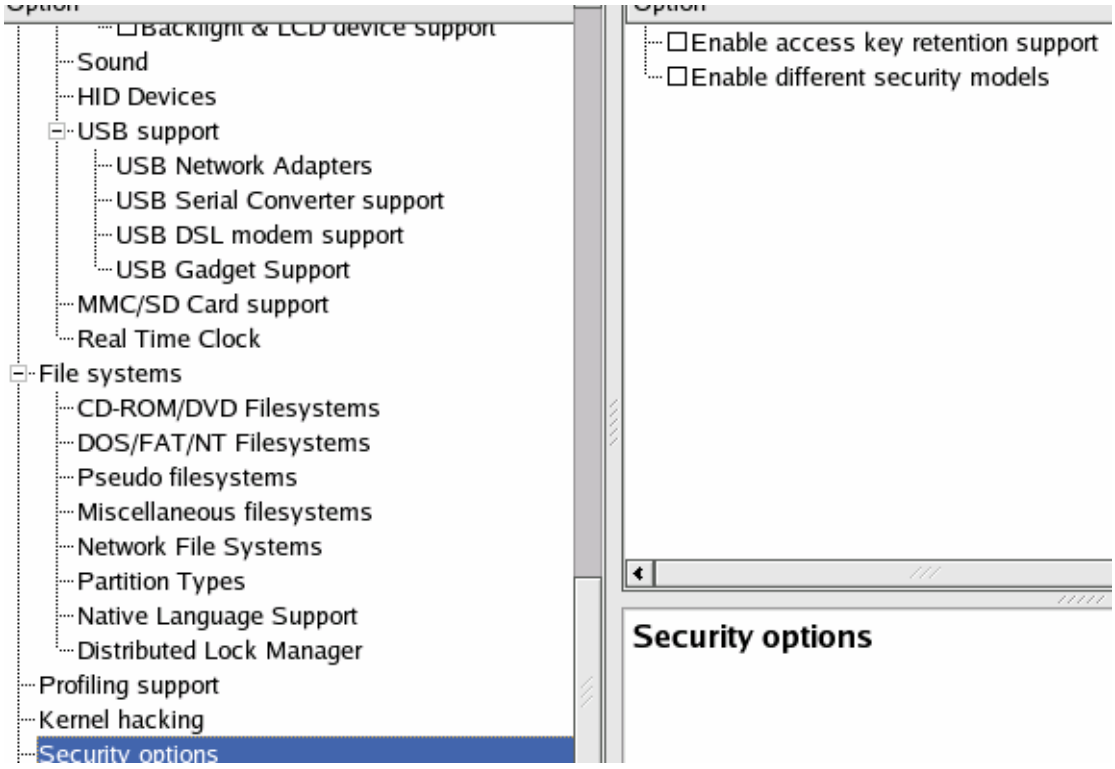




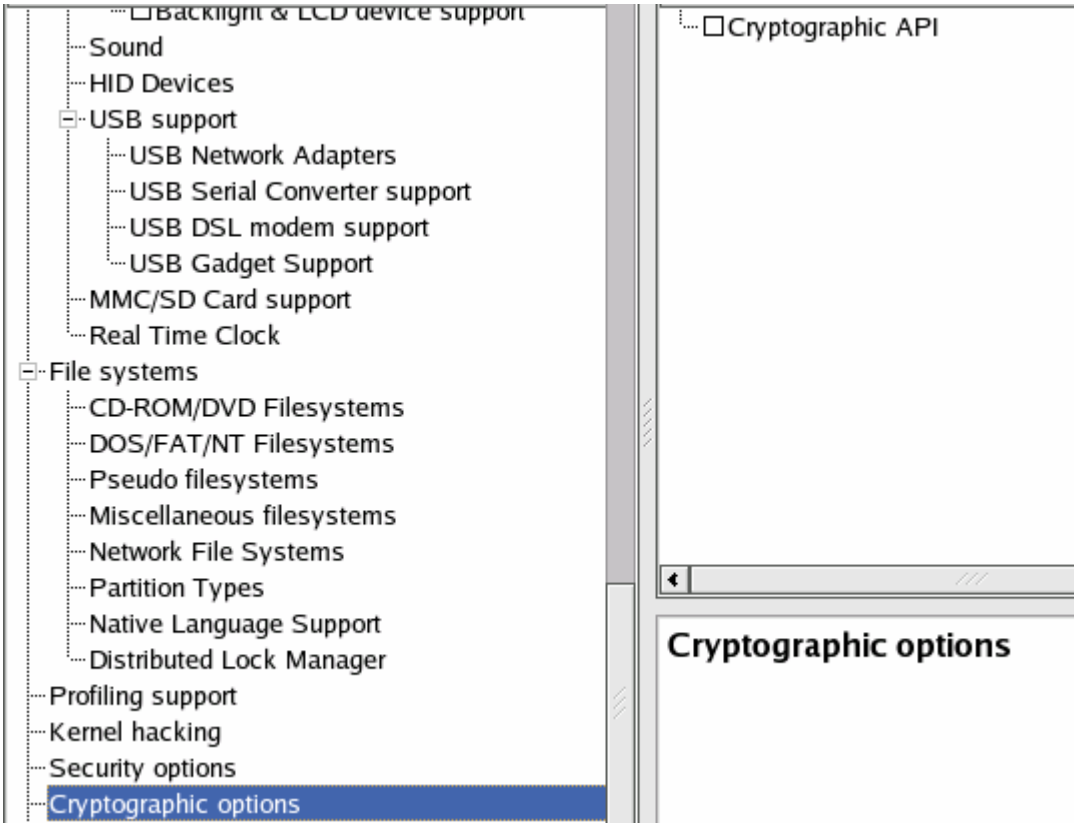
此菜单包含各种内核调试的选项。



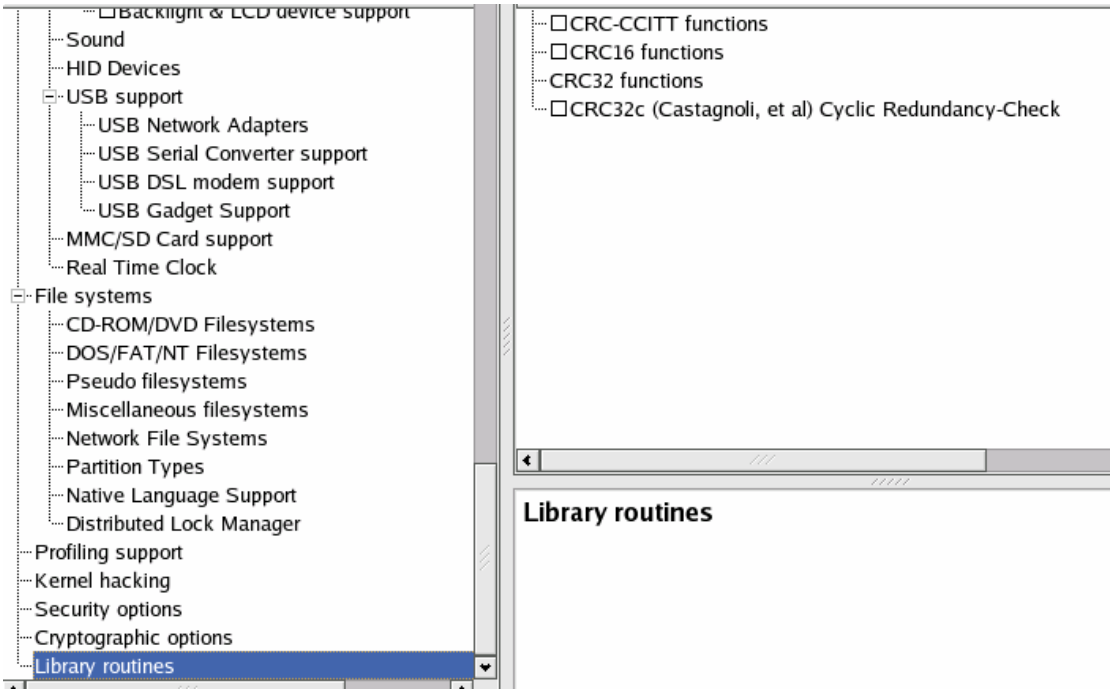
此菜单包含用于系统测试的工具选项。



此菜单包含安全性相关的选项。



此菜单包含加密算法。



此菜单包含一些实用的函数库。

各项配置完成后，可以选择 File->Save，然后退出。

6. 编译 Linux

在命令行输入 make(或者 make ARCH=arm CROSS_COMPILE=arm-linux-)即可开始编译:

```
[root@RH9 linux-2.6.20]# make
```

等待一段时间后，编译即会完成，并生成相关的文件。

```
[root@RH9 boot]# pwd
/hd2nd/9261linux_vert/linux-2.6.20/arch/arm/boot
[root@RH9 boot]# ls
bootp  compressed  Image  install.sh  Makefile  uImage  zImage
```

7. 生成 u-boot 可用的 image

在编译完成的 u-boot 文件夹的 tools 目录下有用于生成 u-boot 可用的 image 的工具 mkimage。

```
[root@RH9 boot]# ls /usr/work/at91_u-boot/u-boot-1.1.5/tools/
bddb        easylogo       environment.o   img2srec          Makefile.win32  scripts
bmp_logo    env            gdb             img2srec.c        mkimage         setlocalversion
bmp_logo.c  envcrc         gen_eth_addr    img2srec.o        mkimage.c       updater
bmp_logo.o  envcrc.c       gen_eth_addr.c  inca-swap-bytes.c mkimage.o       zImage
crc32.c     envcrc.o       gen_eth_addr.o  logos             mpc86x_clk.c    zImage.img
crc32.o     environment.c  img2brec.sh     Makefile          ncb.c
[root@RH9 boot]#
```

复制该文件到 Linux 对应的 boot 目录(见上图)下:

```
[root@RH9 boot]# cp /usr/work/at91_u-boot/u-boot-1.1.5/tools/mkimage ./
[root@RH9 boot]# ls
bootp  compressed  Image  install.sh  Makefile  mkimage  uImage  zImage
[root@RH9 boot]# pwd
/hd2nd/9261linux_vert/linux-2.6.20/arch/arm/boot
[root@RH9 boot]#
```

在当前路径下运行如下命令，生成 u-boot 可用的 uImage:

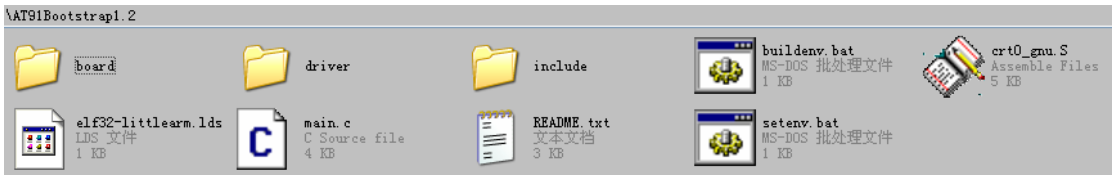
```
[root@RH9 boot]# ./mkimage  -A arm -O linux -T kernel -C none -a 0x20008000 -e 0x20008000 -n 'Lin
ux-2.6.20' -d ./zImage ./uImage
Image Name:   Linux-2.6.20
Created:      Thu Jan 17 22:09:34 2008
Image Type:   ARM Linux Kernel Image (uncompressed)
Data Size:    1294760 Bytes = 1264.41 kB = 1.23 MB
Load Address: 0x20008000
Entry Point:  0x20008000
[root@RH9 boot]# ls
bootp  compressed  Image  install.sh  Makefile  mkimage  uImage  zImage
[root@RH9 boot]# ls -al uImage
-rw-r--r--    1 root     root      1294824 Jan 17 22:09 uImage
```

三、运行 Linux

到了这里，应该具有了 u-boot 及 uImage，再加上一级 boot 和 rootfs，Linux 系统即可以在 9261 上运行。一级 boot 和 rootfs 均有现成的可用。

1. 编译一级 boot

一级 boot 会被 SAM9261 片上的 boot rom 加载到内部 ram 运行，它将初始化相关硬件，例如 SDRAM，然后加载 u-boot 到 SDRAM 中运行。ATMEL 提供了一级 boot 的源代码，用户可以自行编译，需要 arm-elf-工具链。其源代码包为 AT91Bootstrap1.2(20080117)。下载该源码包后将其展开，如下图：



如果安装了 GCC 的工具链，可以到相应的 board 的相应启动类型(dataflash or NAND)目录下，以 NAND 为例：

```
[redacted]/AT91Bootstrap1.2/board/at91sam9261ek/nandflash
$ ls
Makefile  at91sam9261ek.h  nandflash_at91sam9261ek.bin  nandflash_at91sam9261ek.elf  nandflash_at91sam9261ek.map
```

检查 arm-elf-工具链：

```
$ arm-elf-gcc -v
Using built-in specs.
Target: arm-elf
Configured with: ../gcc-4.2.2/configure --target=arm-elf --prefix=/c/gnuarm-4.2.2 --enable-interwork --enable-multilib --with-float=soft --disable-newlib-supplied-syscalls --with-newlib --with-headers=../newlib-1.15.0/newlib/libc/include --enable-languages=c,c++
Thread model: single
gcc version 4.2.2
```

工具链 OK，输入 make，即可开始编译。编译完成后会生成相关的 bin 文件。

如果需要修改相关的代码在 NAND 上的地址，可以修改 nandflash 目录下的 at91sam9261ek.h 文件。相关的参数定义如下：

```
/* ******************************************************************** */
/* BootStrap Settings                                                   */
/*                                                                      */
/* ******************************************************************** */
#define IMG_ADDRESS     0x20000   /* Image Address in NandFlash */
#define IMG_SIZE    0x30000   /* Image Size in NandFlash    */

#define MACH_TYPE       0x350       /* AT91SAM9261-EK */
#define JUMP_ADDR   0x23F00000  /* Final Jump Address         */
```

注释很详细，这里就不多说了。

2. 准备 rootfs

开始阶段 rootfs 可以使用现成的，比如 ATMEL 提供的 armv5l-uclibc-sam9260。

3. 运行 Linux

OK，到了这里，基本上 Linux 运行所需要的文件都已齐备。下面需要将相关文件烧写到板子上。

第五章 在 SAM9261 上调试 U-boot

本文叙述 U-boot 调试过程，调试软件选择了 ARM 的 AXD 与 RVDEBUG，仿真器选择的是 jlink，其他仿真器可以类推。

五， 准备工作

1. 安装调试软件

PC 上的调试软件建议安装 ADS 或者 RealView 2.2，以后者为好。

2. 安装调试器及相关软件

由于使用的 jlink，首先必须安装 jlink 的软件，可以到 segger 网站下载。安装完成后需要将 jlink 整合到 axd 及 rvdebug 中。

连接上 jlink 到 PC 后选择自动安装驱动即可。

然后连接好 jlink 到 9261 板子 JTAG 口的 20 芯排线，给目标板上电(连接 mini USB 线)。

到 jlink 软件安装目录下运行 jlink.exe:

出现：

```
Feature(s) : RDI,FlashDL,FlashBP,JFlash,GDBFull
VTarget = 3.539V
JTAG speed: 30 kHz
Info: CP15.0.0: 0x41069265: ARM, Architecure 5TEJ
Info: CP15.0.1: 0x1D152152: ICache: 16kB (4*128*32), DCache: 16kB (4*128*32)
Found 1 JTAG device, Total IRLen = 4:
 Id of device #0: 0x0792603F
Found ARM with core Id 0x0792603F (ARM9)
  ETM V1.3: 4 pairs addr.comp, 2 data comp, 8 MM decs, 2 counters, sequencer
J-Link>
```

那么就说明核心已经能被正确识别并连接。

3. 完成编译 U-boot 的工作

请参照本站另一片文章《为 SAM926X 编译 U-boot》为 9261 编译好 u-boot。

六， 使用 AXD 调试

8. 编写 AXD 格式的初始化脚本

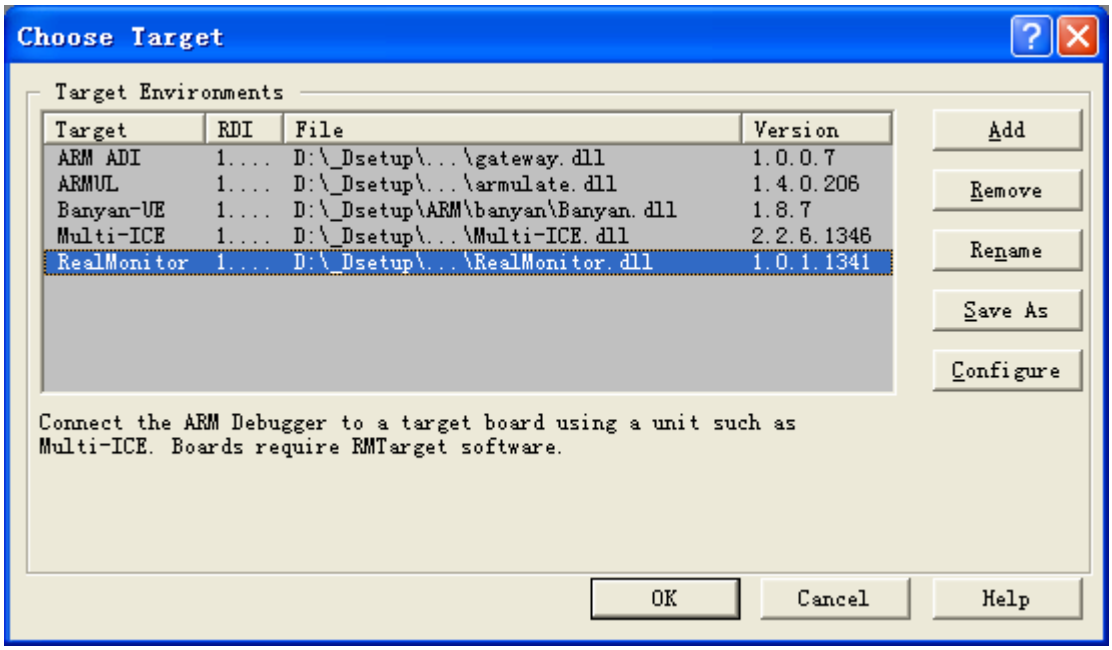
由于要将 u-boot 加载到 sdram 中运行，且 u-boot 中不会再执行 low level 的初始化，因此相关的初始化必须由脚本完成，或者由复位的时候可运行的代码完成。

9. 加载 u-boot 的 elf 文件

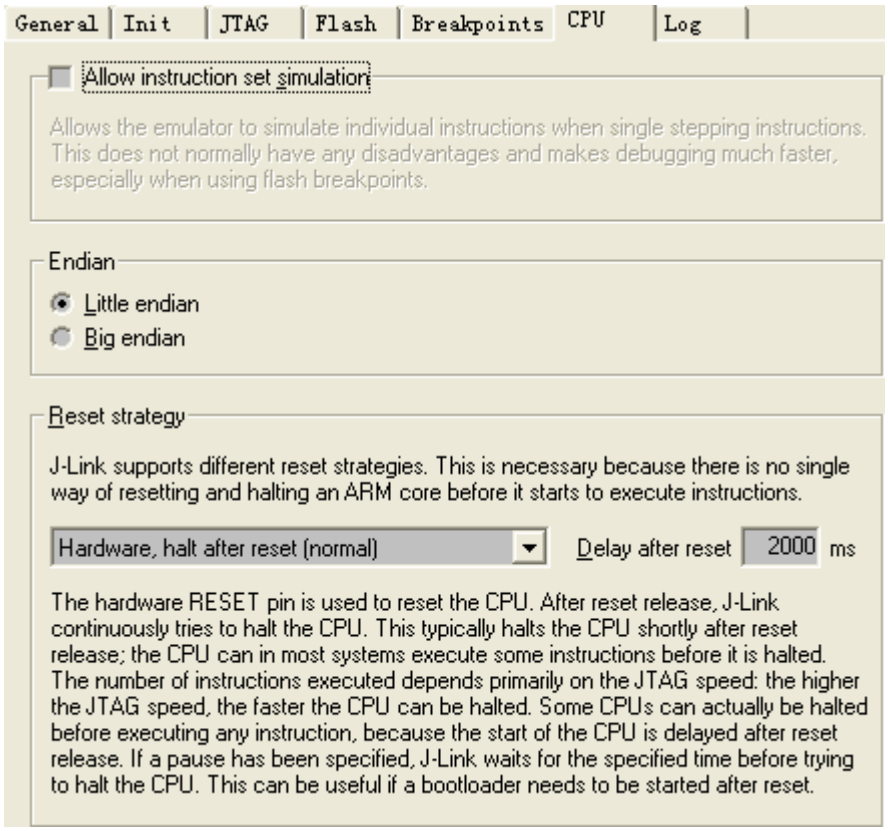
连接底板的 J4 到 PC，出现一个 usb 转的串口，打开超级终端，并连接到该串口。

连接好 miniUSB 的电源线，板上的程序会开始运行，如果是原来的 u-boot,记得要在 linux 或 win ce 起来之前将其停住。

运行 AXD，在菜单 Options->Config Target 中添加 jlink 的 dll(jlinkrdi.dll,在 jlink 软件的安装目录下)：

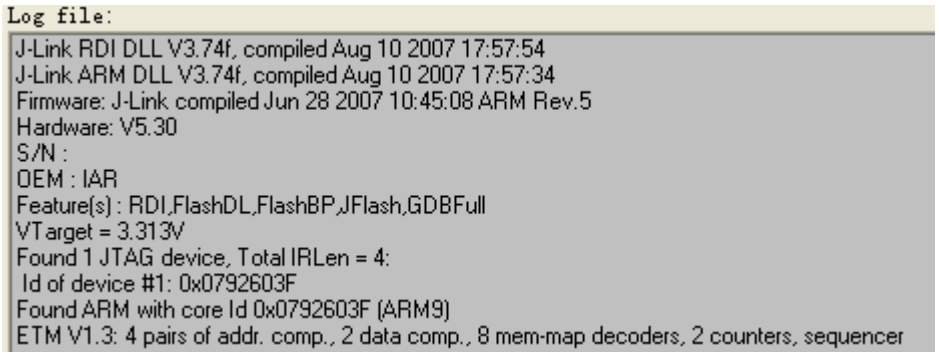


在 jlink 的设置选项卡中设置复位方式：

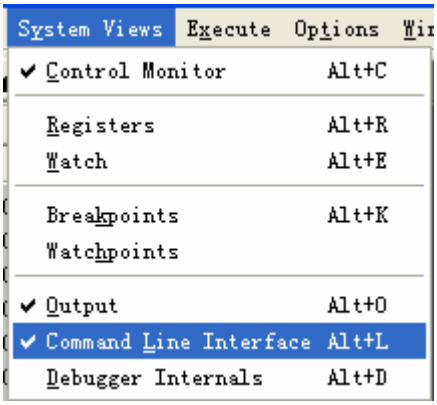


复位后的延时可以稍长。设置这个时间的目的就是让复位之后，系统中现有的代码有机会运行一段时间，这段时间正好完成系统相关的初始化,这样就可以不必使用初始化文件。

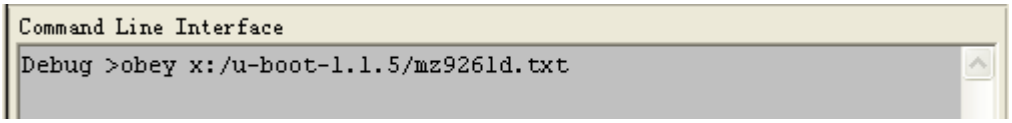
添加了 dll 后就可以进行联接，成功后如图：



选择菜单

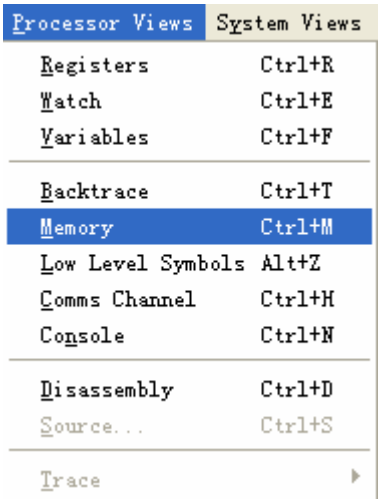


打开命令行窗口，然后在此窗口中运行 9261 的初始化脚本：

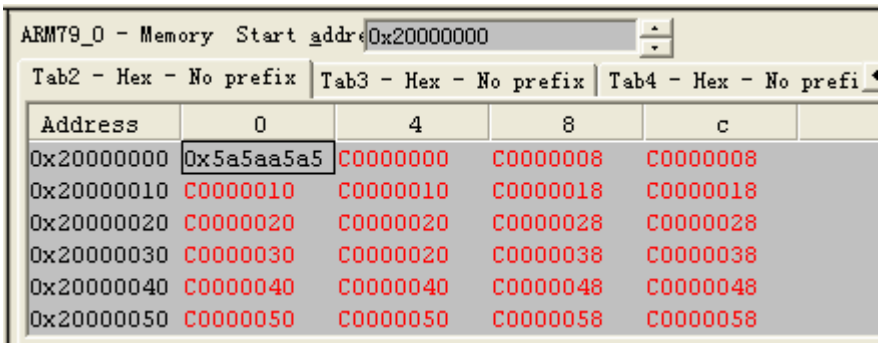


如果板子上有可以运行的代码，比如已经烧写了代码到 dataflash 中，那么根据前面的 jlink 设置，该代码会运行并初始化好环境，就不需要再运行初始化脚本。

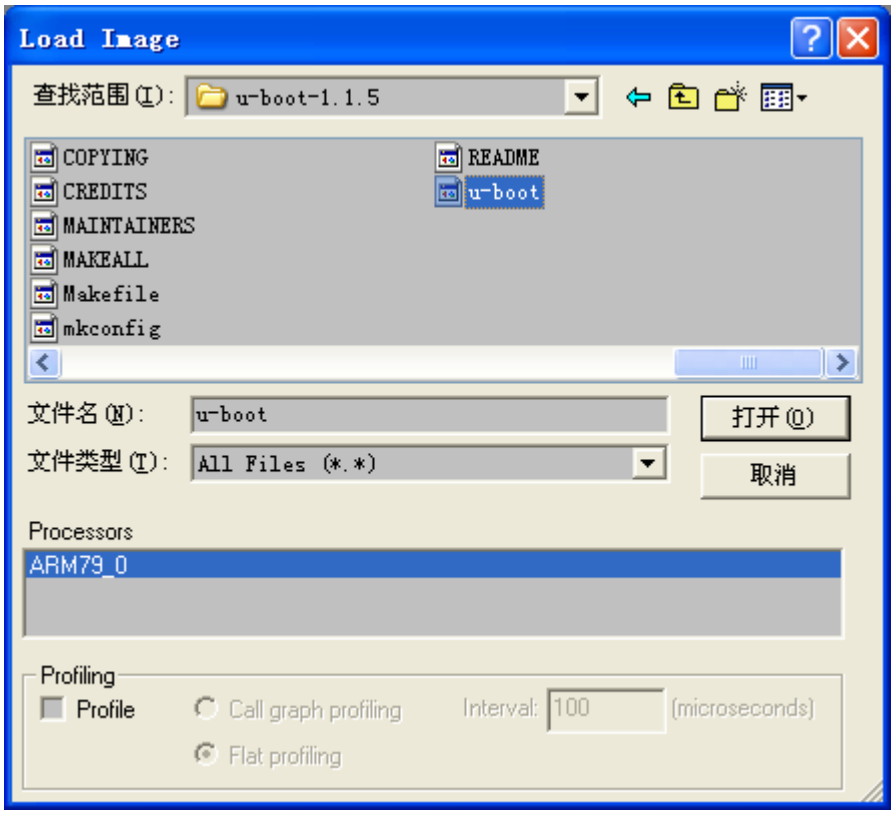
为了检验初始化结果，可以打开内存窗口:



查看 SDRAM，向某个地址写入一个值，看看是否正确初始化：



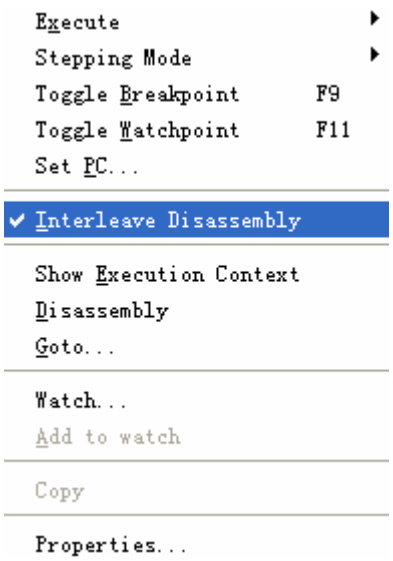
选择菜单 File-> Load Image…,加载编译生成的 u-boot 文件:



加载完成后 AXD 会试图显示第一个源代码文件，如果找不到就会出现：

手动指定到./cpu/arm926ejs/start.S 即可。以后也可能出现这个情况，也可以手动指定。

在代码窗口的右键菜单中选择反汇编模式：



检查代码是否正确加载：

```
52          .globl _start
53          _start:
54              b   reset
23f00000 [0xea000012]   b       (pc)+0x50 ; #0x23f00050
55              ldr pc, _undefined_instruction
23f00004 [0xe59ff014]   ldr     pc,0x23f00020 ; = #0x23f000a0
56              ldr pc, _software_interrupt
23f00008 [0xe59ff014]   ldr     pc,0x23f00024 ; = #0x23f00100
57              ldr pc, _prefetch_abort
23f0000c [0xe59ff014]   ldr     pc,0x23f00028 ; = #0x23f00160
58              ldr pc, _data_abort
23f00010 [0xe59ff014]   ldr     pc,0x23f0002c ; = #0x23f001c0
59              ldr pc, _not_used
23f00014 [0xe59ff014]   ldr     pc,0x23f00030 ; = #0x23f00220
60              ldr pc, _irq
23f00018 [0xe59ff014]   ldr     pc,0x23f00034 ; = #0x23f00280
61              ldr pc, _fiq
23f0001c [0xe59ff014]   ldr     pc,0x23f00038 ; = #0x23f002e0
```

从反汇编结果可以看到代码被正确加载，且代码 link 在 0x23F00000 位置。然后就可以使用 AXD 提供的功能调试代码。比如单步，跳出等等。

10. 代码调试

如果不像看汇编代码，可以直接到 start.S 的 185 行打个断点：

```
185             ldr pc, _start_armboot
186
187         _start_armboot:
188             .word start_armboot
189
```

然后点击运行，即会停在该行。

按 F8 单步，选择./lib_arm/board.c，会停在 C 函数 start_armboot 处。这个函数中有相当多的初始化代码。

下面以调试 NAND 代码为例，大体讲下调试的过程。

在 nand_init(L305, board.c)处打断点并直接运行到该处：

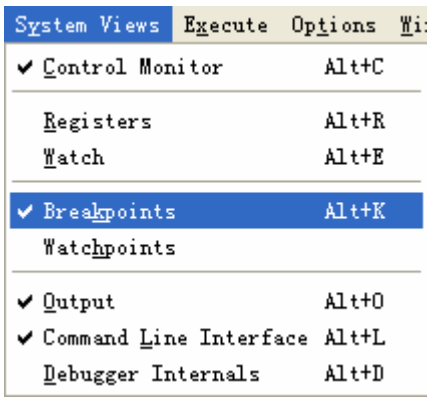
```
302
303         #if (CONFIG_COMMANDS & CFG_CMD_NAND)
304             puts ("NAND:  ");
23f00a28 [0xe59f0114]   ldr     r0,0x23f00b44 ; = #0x23f20610
23f00a2c [0xeb003f77]   bl      puts
305             nand_init();        /* go init the NAND */
23f00a30 [0xeb001262]   bl      nand_init
306         #endif
```

此时超级终端中能收到前面的函数执行完后所打印出的信息：

```
U-Boot 1.1.6 (Aug 26 2007 - 12:52:20)

DRAM:  64 MB
NAND:
```

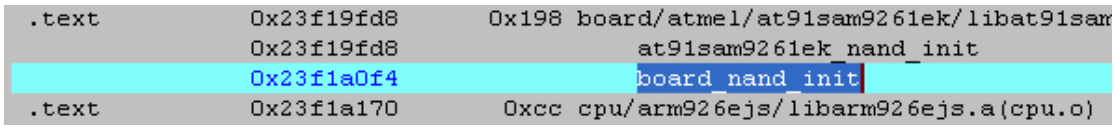
单步跟进，打开./drivers/nand/nand.c。

打开断点窗口：

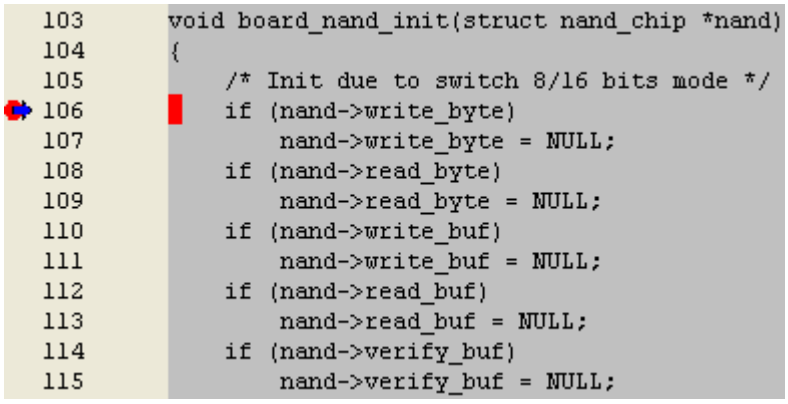


在窗口右键 add：

添加一个断点，地址为 0x23f1a0f4，其实就是 board_nand_init 函数。地址信息来源于 map 文件：



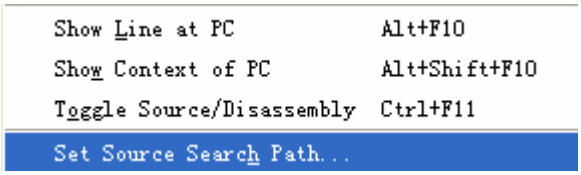
点运行后会执行到文件./board/atmel/at91sam9261ek/nand.c。



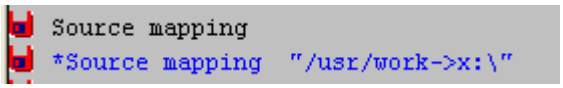
三，使用 RVDEBUG 调试

1. 配置参数

打开 RVDEBUG，连接到 jlink。在 debug 菜单下选择配置文件搜索路径：

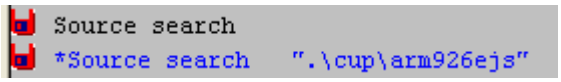


根据 u-boot 代码所在设置好 source 的 mapping，



其中/usr/work 是在 linux 上编译时 u-boot 源代码文件夹，x:为那个文件映射到本地网络驱动器的盘符。

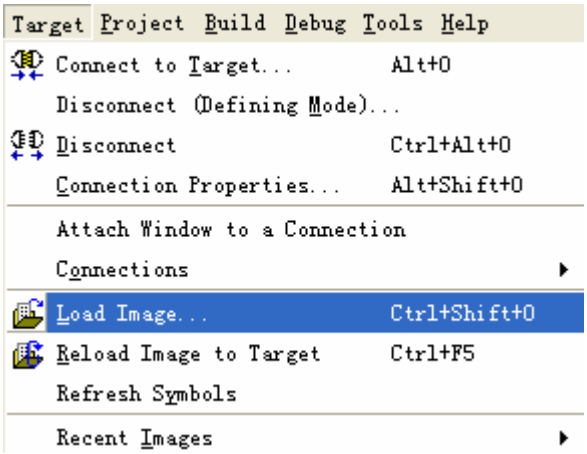
必要的时候也可以设置具体的搜索路径：



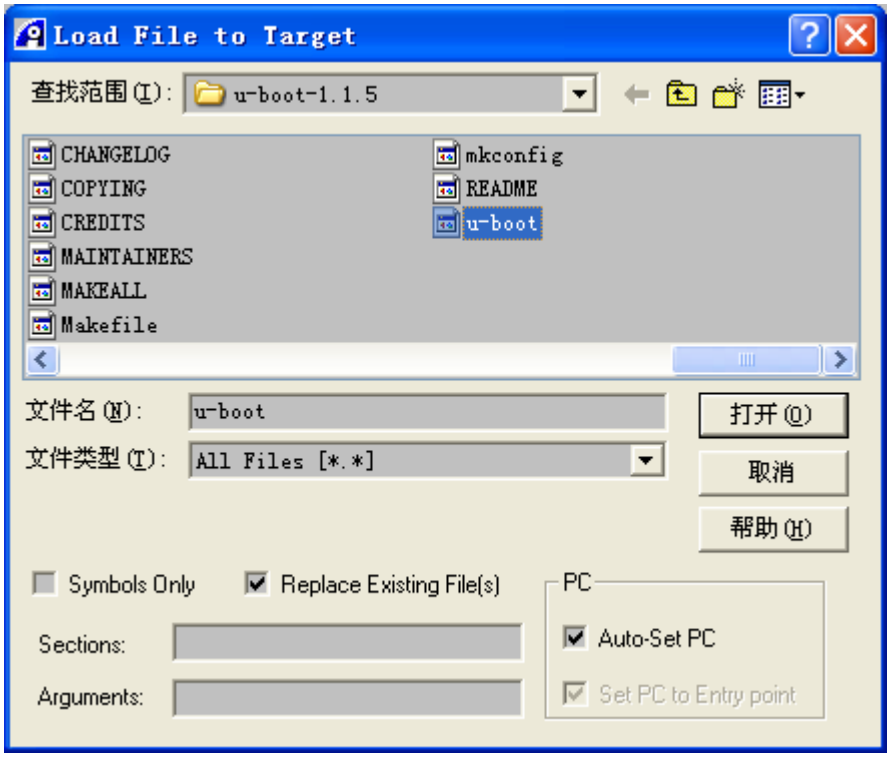
这个路径时基于 u-boot 的 elf 文件所在路径的一个相对路径。

2．加载文件

选择 target 下的菜单:

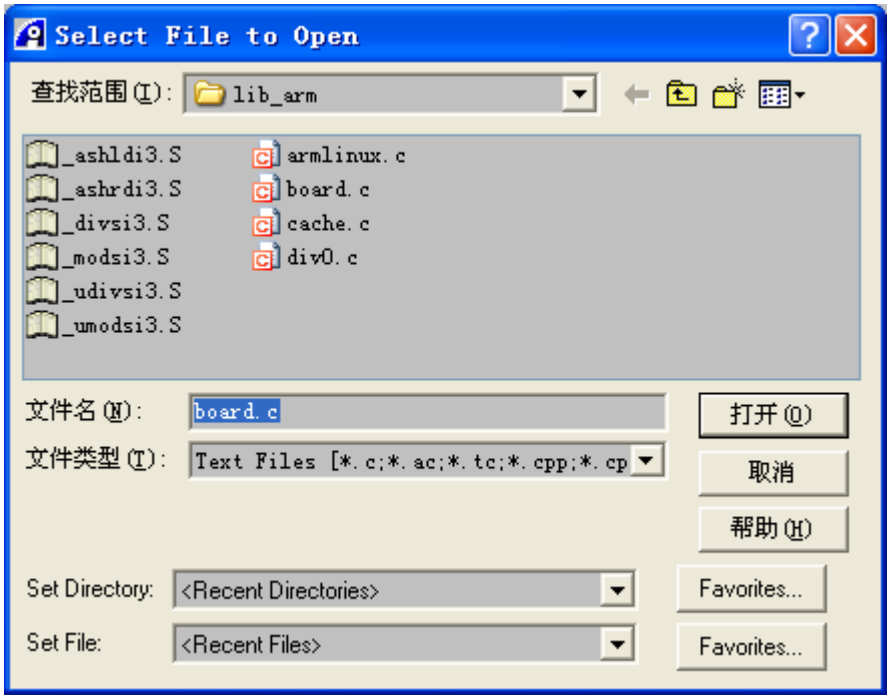


加载 elf 文件:

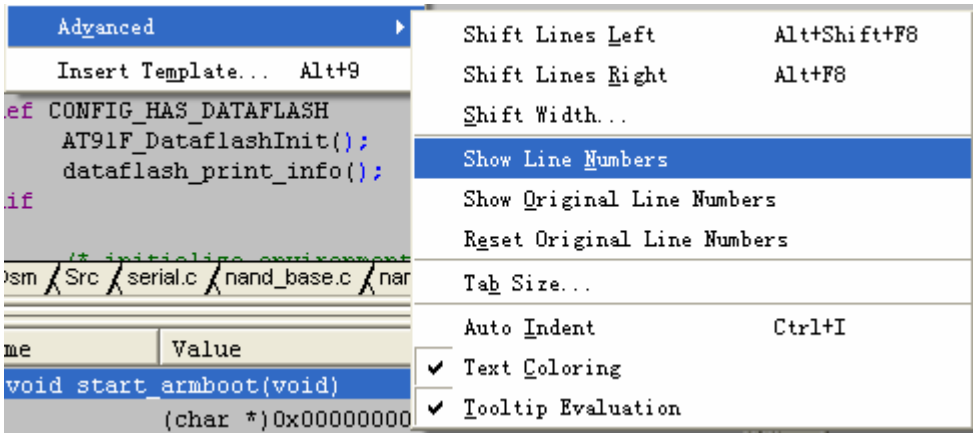


完成后即可在调试窗口看到代码，进入调试状态。

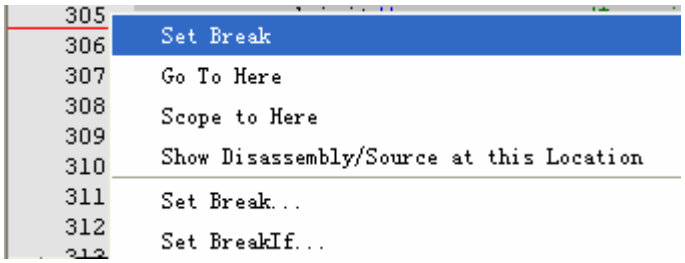
选择 File 下的 Open 打开 board.c：



打开文件后选择 Edit 下功能，显示行数：



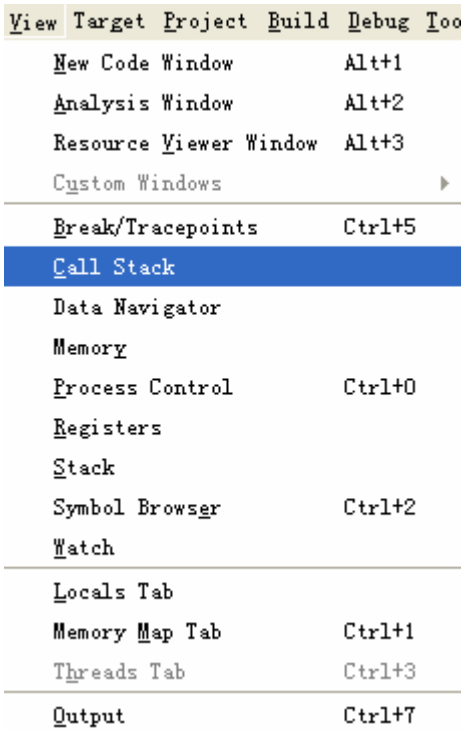
在到 board.c 的 305 行，在左侧函数位置单击鼠标左键：



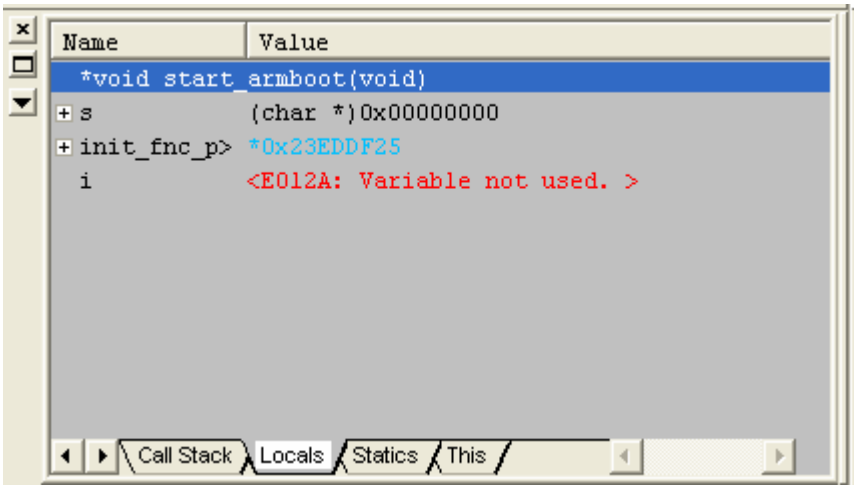
选择 Set Break 即可打断点。

在代码中可以使用单步等常规调试功能。**注意**：RVDEBUG 的调试快捷键与 AXD 不同。

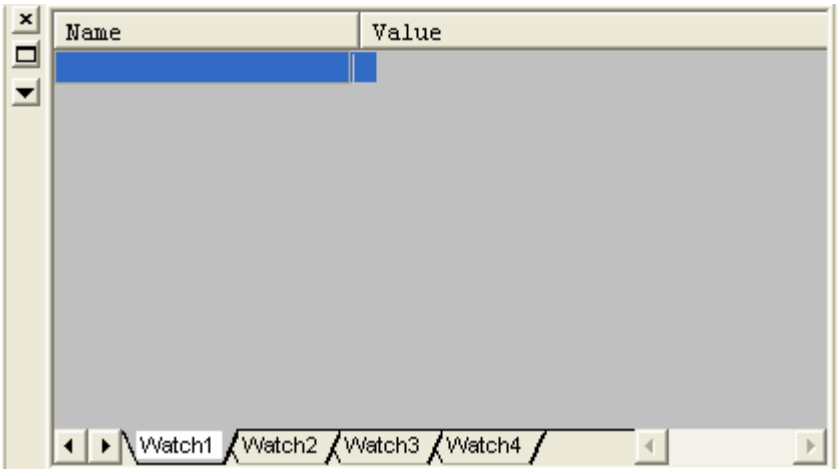
在 view 菜单下可以打开相应的资源查看窗口：



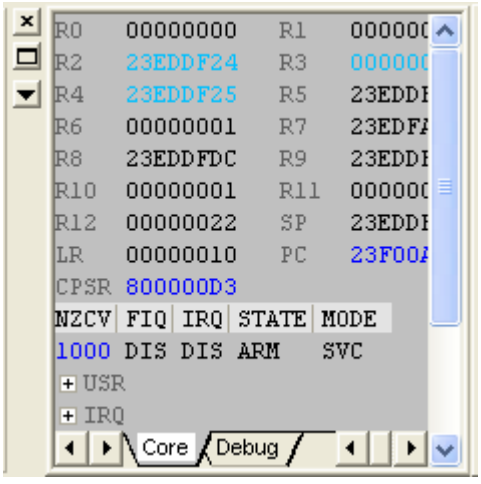
比如 call stack 窗口：



watch 窗口：



Register 窗口：



四，一些事项

U-boot 在默认的情况下，编译的时候会进行优化，这会对调试造成影响，有的时候源码调试会比较困难。为了调试方便，可以关闭优化功能生成代码，专门用于调试，调试完成后再修改成优化方式。由于关闭优化会使得代码体积变大，因此可能需要关注

一下可能产生的问题。

关闭优化可以通过修改 U-boot 根目录下的 config.mk 来实现：

```
128 RANLIB  = $(CROSS_COMPILE)RANLIB
129
130 ARFLAGS = crv
131 RELFLAGS= $(PLATFORM_RELFLAGS)
132 DBGFLAGS= -g # -DDEBUG
133 OPTFLAGS= -Os #-fomit-frame-pointer
134 ifndef LDSCRIPT
135 #LDSCRIPT := $(TOPDIR)/board/$(BOARDDIR)/u-boot.lds.debug
```

将 133 行的 Os 改成 O0，先 make clean，然后重新编译，即可用新的 U-boot 调试。

第六章：SAM9261 上的 Linux 初步应用 1

本章叙述 9261 Linux 的初步应用，包括了 Linux 下的 LCD 参数的修改，MTD 设备的使用，添加和使用板子上的 LED，以及 u-boot 下 U 盘的使用。开发使用的 Linux 版本选择了 2.6.20, 开发环境选择了 Virtual PC，其它软件可以类推。

一，准备工作

1. 使用 Virtual PC 安装 Linux 虚拟机

   请参照本站另一篇文档《搭建基于 VPC 的 Linux 平台》。

2. 下载相关软件包

   下载下列软件(可在本站 ftp 上下载)，并传输到 Linux 虚拟机下。

   linux-2.6.20-atmel9261-nfs.tar.bz2

   arm-linux-gcc-3.4.1.tar.bz2

   将后者展开到/usr/local/arm 下，

   tar xjvf arm-linux-gcc-3.4.1.tar.bz2

   并添加其路径到系统路径:

   在~/.bashrc 文件(使用 bash shell)中添加/usr/local/arm/3.4.1/bin

```
# Export Path
export PATH=$PATH:/opt/crosstool/gcc-3.4.1-glibc-2.3.3/arm-softfloat-linux-gnu/bin
export PATH=$PATH:/usr/local/arm/3.4.1/bin
[root@RH9 linux-2.6.20]#
```

   然后运行 source ~/.bashrc，新改动立即生效。

   可以运行相关的命令检查效果。

```
[root@RH9 linux-2.6.20]# arm-linux-gcc
arm-linux-gcc: no input files
[root@RH9 linux-2.6.20]# arm-linux-gcc -v
Reading specs from /usr/local/arm/3.4.1/lib/gcc/arm-linux/3.4.1/specs
Configured with: /work/crosstool-0.27/build/arm-linux/gcc-3.4.1-glibc-2.3.2/gcc-3.4.1/configure -
-target=arm-linux --host=i686-host_pc-linux-gnu --prefix=/usr/local/arm/3.4.1 --with-headers=/usr
/local/arm/3.4.1/arm-linux/include --with-local-prefix=/usr/local/arm/3.4.1/arm-linux --disable-n
ls --enable-threads=posix --enable-symvers=gnu --enable-__cxa_atexit --enable-languages=c,c++ --e
nable-shared --enable-c99 --enable-long-long
Thread model: posix
gcc version 3.4.1
[root@RH9 linux-2.6.20]# which arm-linux-gcc
/usr/local/arm/3.4.1/bin/arm-linux-gcc
[root@RH9 linux-2.6.20]#
```

   为了开发方便，可在 /usr 下建立新文件夹，名为 work, 将 linux 源代码放置到该路径下，然后运行命令展开压缩包:

   cd /usr/work

   tar xjvf linux-2.6.20-atmel9261-nfs.tar.bz2

   解压缩完成后将在当前路径下生成 linux-2.6.20 文件夹。

3. 完成编译 U-boot 的工作

请参考本站另一篇文章**《为 SAM926X 编译 U-boot》**为 9261 编译好 u-boot。

4. 完成 Linux 对 9261 的基本配置和编译，验证 Linux 可以运行
请参照本站另一篇文章**《为 SAM9261 编译 Linux》**为 9261 编译好 Linux。

二，修改 Linux 下的 LCD 参数

ATMEL 所提供的 Linux 代码包，已经做到了支持 LCD 显示，但为了使用到本站的板子上，还需要一些改动和配置。

1. 修改源代码

由于本站使用的 LCD 与官方的竖屏有一些驱动参数不同，需要做相应的改动。首先打开\arch\arm\mach-at91rm9200\board-sam9261ek.c，找到 at91_tft_vga_modes[]，如下图：

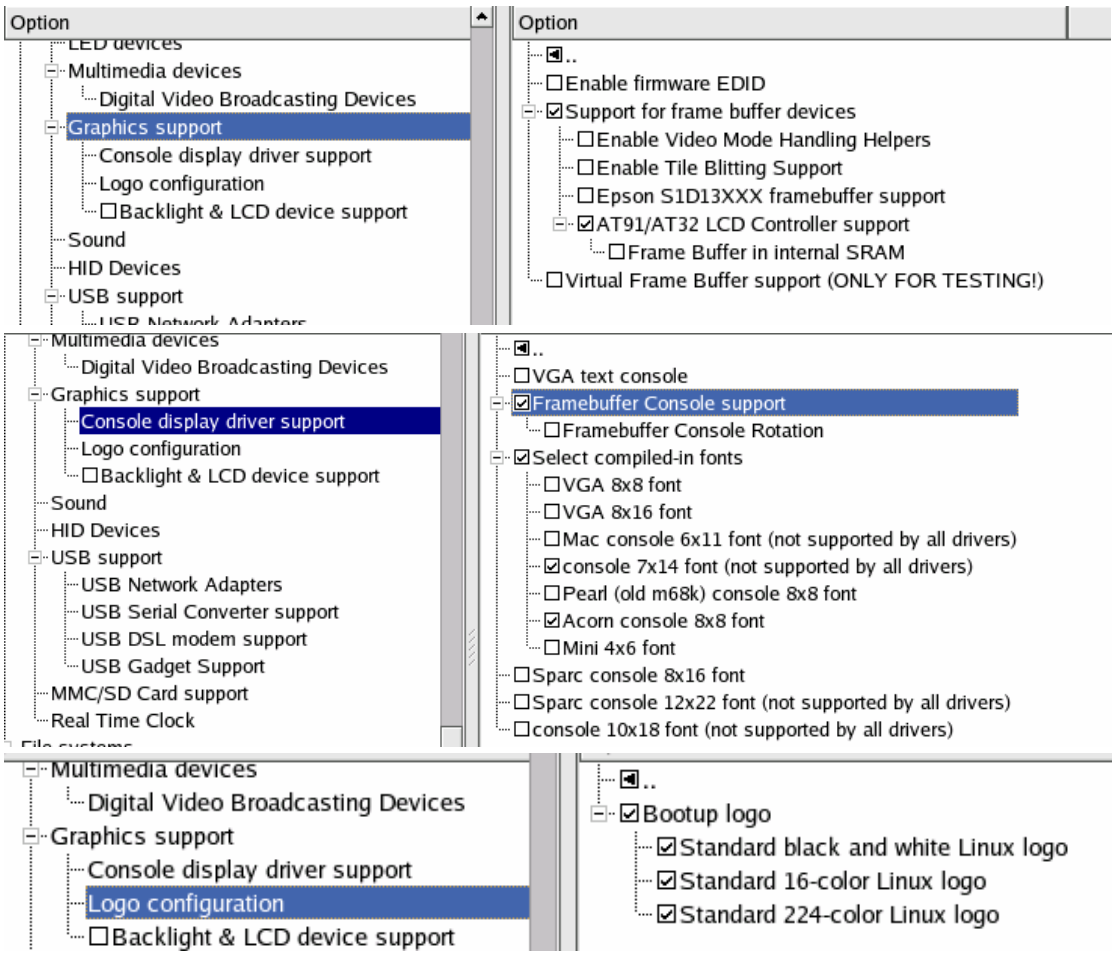
```
/*
 * LCD Controller
 */
#if defined(CONFIG_FB_ATMEL) || defined(CONFIG_FB_ATMEL_MODULE)
static struct fb_videomode at91_tft_vga_modes[] = {
  {
          .name           = "TX09D50VM1CCA @ 60",
    .refresh  = 60,
    .xres   = 240,    .yres   = 320,
    .pixclock = KHZ2PICOS(4965),

    .left_margin  = 1,     .right_margin = 33,
    .upper_margin = 1,     .lower_margin = 0,
    .hsync_len  = 5,     .vsync_len  = 1,

    .sync   = FB_SYNC_HOR_HIGH_ACT | FB_SYNC_VERT_HIGH_ACT,
    .vmode    = FB_VMODE_NONINTERLACED,
  },
};
```

这里定义了 LCD 的相关参数，需要按照目前使用的 LCD 进行修改。为了保留原始的代码便于跟踪改动，可以使用#if。修改后的效果如下：

```
#if 1
          .name           = "TX09D50VM1CCA @ 60",
    .refresh  = 60,
    .xres   = 240,    .yres   = 320,
    .pixclock = KHZ2PICOS(4965),

    .left_margin  = 48,    .right_margin = 16,
    .upper_margin = 31,    .lower_margin = 12,
    .hsync_len  = 96,    .vsync_len  = 2,

    .sync   = FB_SYNC_HOR_HIGH_ACT | FB_SYNC_VERT_HIGH_ACT,
    .vmode    = FB_VMODE_NONINTERLACED,

#else
          .name           = "TX09D50VM1CCA @ 60",
    .refresh  = 60,
    .xres   = 240,    .yres   = 320,
    .pixclock = KHZ2PICOS(4965),

    .left_margin  = 1,     .right_margin = 33,
    .upper_margin = 1,     .lower_margin = 0,
    .hsync_len  = 5,     .vsync_len  = 1,

    .sync   = FB_SYNC_HOR_HIGH_ACT | FB_SYNC_VERT_HIGH_ACT,
    .vmode    = FB_VMODE_NONINTERLACED,
#endif
```

注意：上图修改的参数适用于本站提供的竖屏，仅供参考。

运行 make xconfig，使能 Framebuffer 设备驱动，并选择正确的控制器：



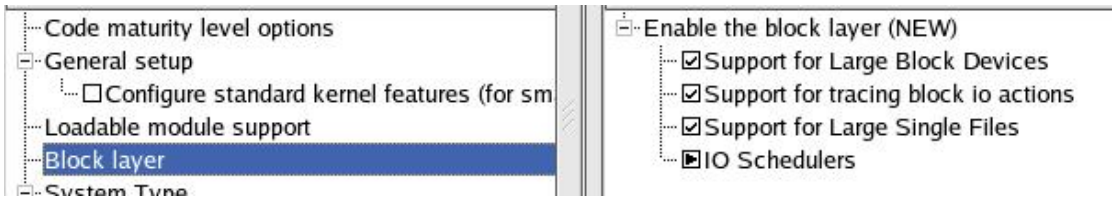
这步配置完成后，可以选择 File->Save，下面会继续配置别的项目。

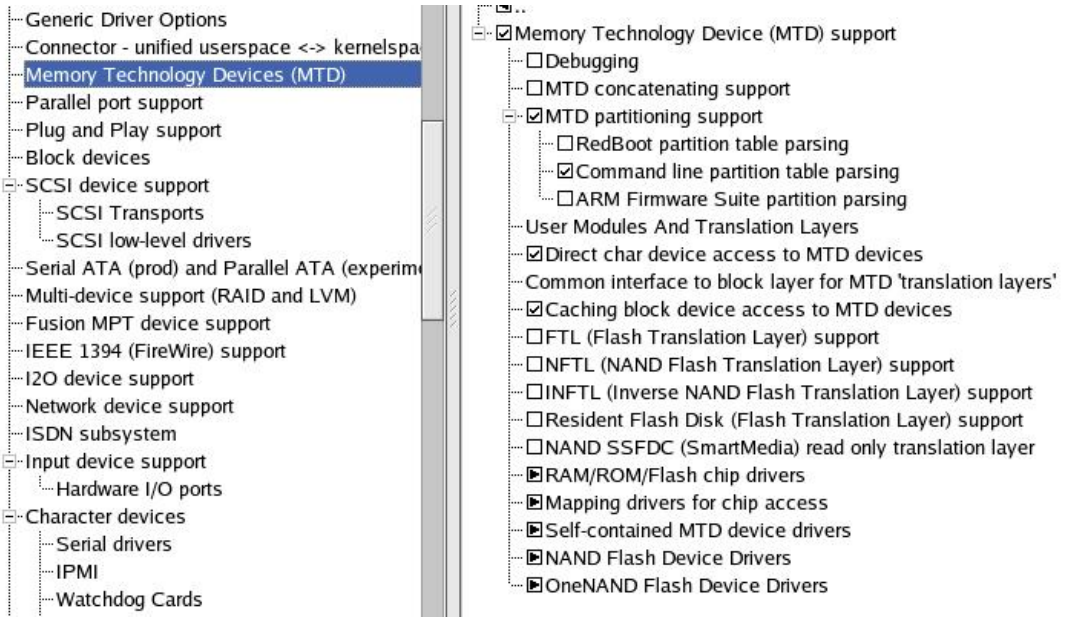
三，使用 MTD 设备

1. 配置 Linux

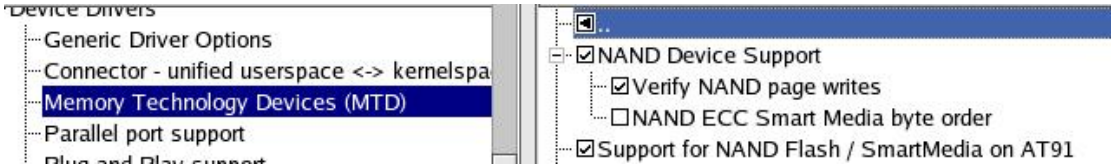
为了使用基于 NAND 的 MTD 设备，必须进行相关的配置。

具体设置如下：

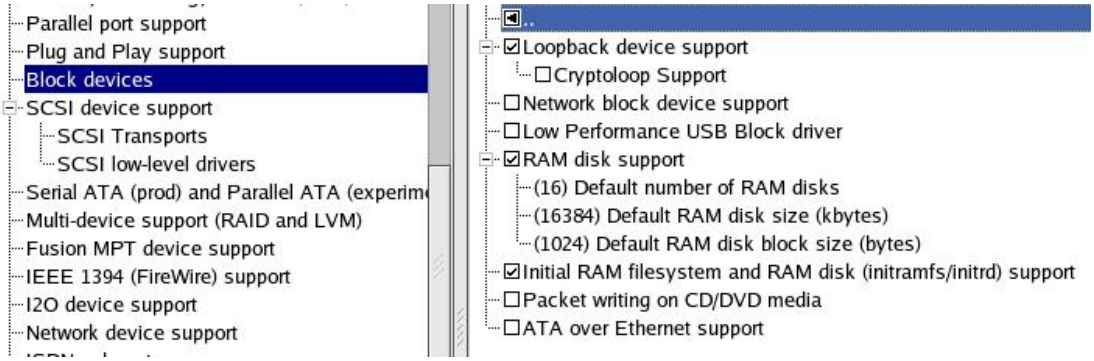




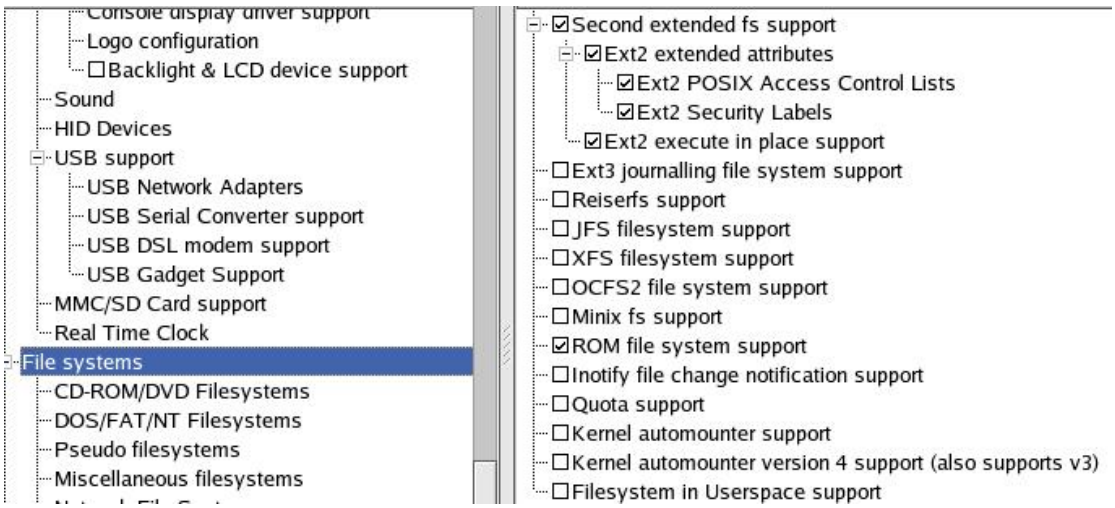
选择 NAND Flash Device Drivers:

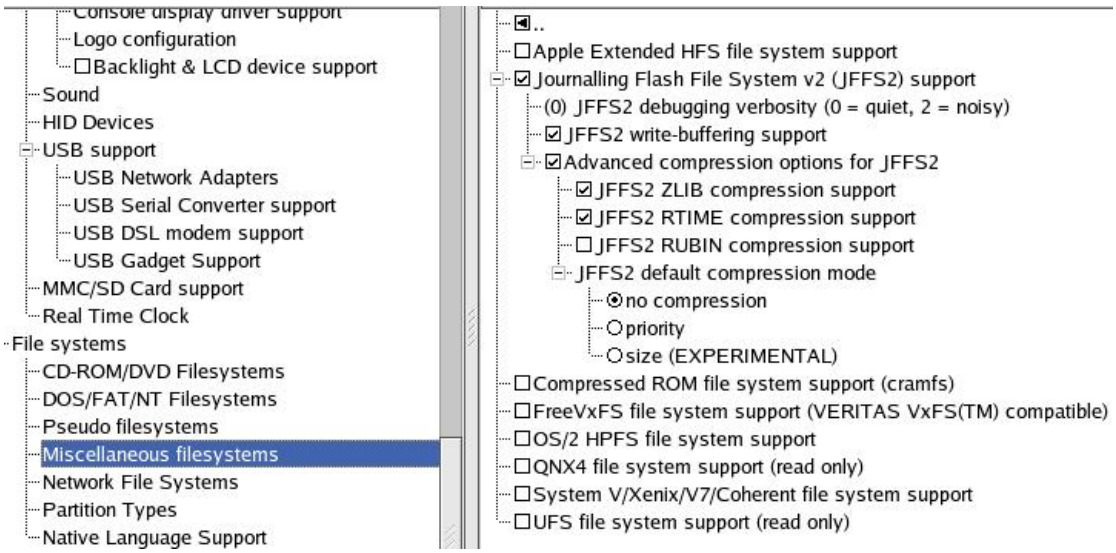


选择 RAM DISK 的支持：



配置 JFFS2 的支持：





OK，保存配置。

2. 修改源代码

针对板子上 128MB 的 NAND，可以进行一定的分区，以配置各分区的大小。

打开 arch/arm/mach-at91rm9200/board-sam9261ek.c，找到 ek_nand_partition[]，修改分区大小，

```
/*
 * NAND flash
 */
static struct mtd_partition __initdata ek_nand_partition[] = {
#if 1
  {
    .name = "kernel",
    .offset = 0,
    .size = 8192 * 1024,
  },
  {
    .name = "rootfs",
    .offset = 8192 * 1024 ,
    .size = MTDPART_SIZ_FULL,
  },
#else
  {
    .name = "kernel",
    .offset = 0,
    .size = 256 * 1024,
  },
  {
    .name = "rootfs",
    .offset = 256 * 1024 ,
    .size = MTDPART_SIZ_FULL,
  },
#endif
};
```

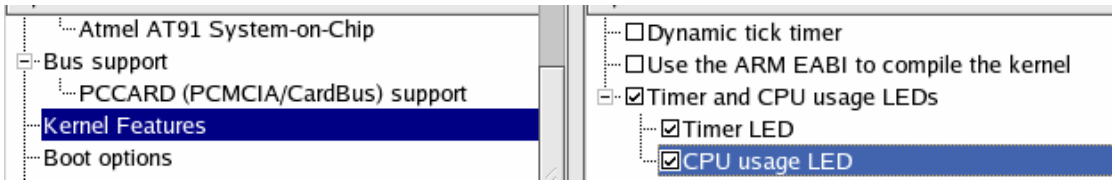
如上设置，NAND 被分成两个分区，第一个分区名为”kernel”，从 0 开始，8MB，NAND 剩下的部分作为第二个分区，名为”rootfs”。

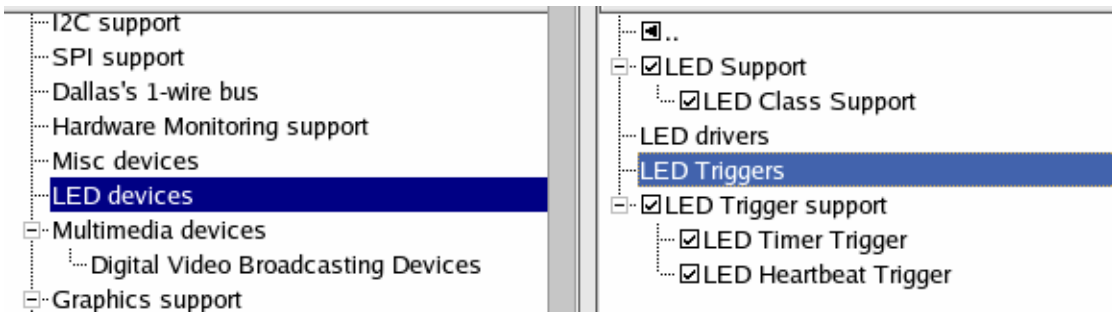
修改完成后需要保存代码。

四，配置和使用 LED

1. 配置 Linux

首先配置一些内核参数：





配置完成，保存后退出 xconfig。

2. 修改源代码

配置完成后还需要添加 LED 支持的相关代码。这部分的代码中，很多已经实现，还有一些可以参考其它板子(例如 at91rm9200_devices.c, board_ek.c)。

具体的修改如下：

1) 打开文件 arch/arm/mach-at91rm9200/board-sam9261ek.c，找到 ek_board_init(void)。首先添加 LED 的资源，方便起见，可以将代码直接添加到 ek_board_init()之前：

```
#if defined(CONFIG_LEDS)
/* LED */
static struct at91_gpio_led ek9261_leds[] = {
  [0] = {
    .name    = "led0",
    .gpio    = AT91_PIN_PA13,
    .trigger  = "heartbeat",
  },
  [1] = {
    .name    = "led1",
    .gpio    = AT91_PIN_PA14,
    .trigger  = "timer",
  }
};
#endif

static void __init ek_board_init(void)
{
  /* Serial */
  at91_add_device_serial();
```

这个数组包含 LED 的名称，指定的 IO，以及关联的驱动事件。

2) 在板子初始化时，初始化 LED。这段代码可以添加到 ek_board_init()的最后：

```
  /* AT73C213 & SSC1 port */
  at91_add_device_ssc1_at73c213();

#if defined(CONFIG_LEDS)
  /* LEDs */
  at91_gpio_leds(ek9261_leds, ARRAY_SIZE(ek9261_leds));
#warning ++++++++++++++++ CONFIG_LEDS ++++++++++++++++++++
#endif

#if defined(CONFIG_NEW_LEDS)
#warning ---------------- CONFIG_NEW_LEDS ----------------
#endif
}
```

3) 初始化相关 IO，在 board-sam9261ek.c 中查找 ek_map_io(void)，为 LED 添加下列代码：

```
static void __init ek_map_io(void)
{
  /* Initialize processor: 18.432 MHz crystal */
  at91sam9261_initialize(18432000);

#if defined(CONFIG_LEDS)
  /* Setup the LEDs */
  /*
   * Name              IO              LED on Board
   * CPU LED:      AT91_PIN_PA13          DS5
   * TIMER LED:    AT91_PIN_PA14          DS4
   */
  at91_init_leds(AT91_PIN_PA13, AT91_PIN_PA14);
#endif

  /* Setup the serial ports and console */
  at91_init_serial(&ek_uart_config);
}
```

4) 修改/arch/arm/mach-at91rm9200 下的 Makefile，为 9261ek 板子添加 led 的支持。

```
# LEDs support
led-$(CONFIG_ARCH_AT91RM9200DK) += leds.o
led-$(CONFIG_MACH_AT91RM9200EK) += leds.o
led-$(CONFIG_MACH_CSB337) += leds.o
led-$(CONFIG_MACH_CSB637) += leds.o
led-$(CONFIG_MACH_KB9200) += leds.o
led-$(CONFIG_MACH_KAFA)   += leds.o
obj-$(CONFIG_MACH_AT91SAM9261EK) += leds.o
obj-$(CONFIG_LEDS) += $(led-y)

```

3. 编译 Linux

在命令行输入 make 即可开始编译：

```
[root@RH9 linux-2.6.20]# make
```

等待一段时间后，编译即会完成，并生成相关的文件。

```
[root@RH9 boot]# pwd
/hd2nd/9261linux_vert/linux-2.6.20/arch/arm/boot
[root@RH9 boot]# ls
bootp  compressed  Image  install.sh  Makefile  uImage  zImage
```

4. 生成 u-boot 可用的 image

在编译完成的 u-boot 文件夹的 tools 目录下有用于生成 u-boot 可用的 image 的工具 mkimage。

```
[root@RH9 boot]# ls /usr/work/at91_u-boot/u-boot-1.1.5/tools/
bddb        easylogo       environment.o   img2srec          Makefile.win32  scripts
bmp_logo    env            gdb             img2srec.c        mkimage         setlocalversion
bmp_logo.c  envcrc         gen_eth_addr    img2srec.o        mkimage.c       updater
bmp_logo.o  envcrc.c       gen_eth_addr.c  inca-swap-bytes.c mkimage.o       zImage
crc32.c     envcrc.o       gen_eth_addr.o  logos             mpc86x_clk.c    zImage.img
crc32.o     environment.c  img2brec.sh     Makefile          ncb.c
[root@RH9 boot]#
```

复制该文件到 Linux 对应的 boot 目录(见上图)下:

```
[root@RH9 boot]# cp /usr/work/at91_u-boot/u-boot-1.1.5/tools/mkimage ./
[root@RH9 boot]# ls
bootp  compressed  Image  install.sh  Makefile  mkimage  uImage  zImage
[root@RH9 boot]# pwd
/hd2nd/9261linux_vert/linux-2.6.20/arch/arm/boot
[root@RH9 boot]#
```

在当前路径下运行如下命令，生成 u-boot 可用的 uImage:

```
[root@RH9 boot]# ./mkimage  -A arm -O linux -T kernel -C none -a 0x20008000 -e 0x20008000 -n 'Lin
ux-2.6.20' -d ./zImage ./uImage
Image Name:   Linux-2.6.20
Created:      Thu Jan 17 22:09:34 2008
Image Type:   ARM Linux Kernel Image (uncompressed)
Data Size:    1294760 Bytes = 1264.41 kB = 1.23 MB
Load Address: 0x20008000
Entry Point:  0x20008000
[root@RH9 boot]# ls
bootp  compressed  Image  install.sh  Makefile  mkimage  uImage  zImage
[root@RH9 boot]# ls -al uImage
-rw-r--r--    1 root     root      1294824 Jan 17 22:09 uImage
```

五，运行 Linux

到了这里，应该具有了新编译的 Linux kernel。再加上一级 boot，u-boot 和 rootfs，Linux 系统即可以在 9261 上运行。

一级 boot，u-boot 和 rootfs 均可以使用以前编译和获取的。

4. 准备相关代码

将一级 boot 和 u-boot 都烧写到 9261 板子的 data flash 上。相关的烧录地址如下：

0x00000000　　DataFlashBoot_642D_100MHz.bin

（0x00004000　　u-boot 环境变量）

0x00008000　　u-boot.bin

注意：下载一级 boot 时需要选择 send boot file。

由于 9261 不支持从 NAND 启动，一级 boot 文件需要烧写到 data flash。烧写和运行的步骤请参考本站另一篇文章《**9261 核心板 Linux 测试 Rev1.0——基于 AT45DB321+K9F1G08**》。

5. 准备 U 盘

找一个 U 盘，可以格式化为 FAT32 格式。将前面编译生成的 uImage 放到根目录，同时也将 armv5l-uclibc-sam9260 放到 U 盘根目录。

6. 设置 U-boot 环境变量

为了从 USB 上启动 Linux，还需要设置一些 U-boot 的环境变量。

将烧好的板子和 PC 用串口线连接，打开超级终端，按下复位键，停在 U-boot 提示符下：

```
Hit any key to stop autoboot:  0
U-Boot>
U-Boot>
```

将 U 盘接到板子的 USB HOST 插座上，输入：

usb start

```
U-Boot> usb start
(Re)start USB...
USB:   scanning bus for devices... 2 USB Device(s) found
       scanning bus for storage devices... 1 Storage Device(s) found
```

OK，U 盘被找到并正确识别，可以看看文件目录(下面的命令以 U 盘连接在上方的 USB HOST 插座为例)：

```
U-Boot> fatls usb 0:1 /
  1307192   uimage
  1097960   uimage3.bin
  4194304   armv5l-uclibc-sam9260

3 file(s), 0 dir(s)

U-Boot>
```

该命令列出了 U 盘根目录下的文件。

以上步骤说明 U 盘可以被 U-boot 正确识别和应用。下面开始设置环境变量。

在 U-boot 命令行下执行以下各条命令：

```
set usb_init usb start
set usb_ls fatls usb 0:1 /
set ker fatload usb 0:1 21500000 uImage
set fs fatload usb 0:1 21100000 armv5l-uclibc-sam9260
set bootcmd run usb_init\;run usb_ls\;run fs\; run ker\;bootm 21500000
set bootargs mem=64M console=tty0,115200 initrd=0x21100000,4M root=/dev/ram0 rw
```

然后输入：

saveenv

保存所有的变量。

再次输入：

printenv

该命令将显示当前的环境变量：

```
U-Boot> printenv
bootdelay=3
baudrate=115200
usb_init=usb start
usb_ls=fatls usb 0:1 /
fs=fatload usb 0:1 21100000 armv5l-uclibc-sam9260
bootcmd=run usb_init;run usb_ls;run fs; run ker;bootm 21500000
ker=fatload usb 0:1 21500000 uImage
bootargs=mem=64M console=tty0,115200 initrd=0x21100000,4M root=/dev/ram0 rw
stdin=serial
stdout=serial
stderr=serial

Environment size: 336/16380 bytes
U-Boot> _
```

7. 运行 Linux

以上步骤之后，接上竖屏，按下板子上的 reset 键，板子复位，U-boot 会自动从 U 盘加载 Linux 并运行：

```
Hit any key to stop autoboot:  0
(Re)start USB...
USB:   scanning bus for devices... 2 USB Device(s) found
       scanning bus for storage devices... 1 Storage Device(s) found
  1307192   uimage
  1097960   uimage3.bin
  4194304   armv5l-uclibc-sam9260

3 file(s), 0 dir(s)

reading armv5l-uclibc-sam9260
................................................................................
................................................................................
................................................................................
................................................................................
................................................................................
.........

4194304 bytes read
reading uImage
....................................._
```

启动过程中可以在 LCD 上看到 Linux 的小企鹅 logo 及输出信息，并可以观察到板子上的两个 LED 闪烁。

8. 使用 MTD 设备

进入 Linux 后，在命令行下可以查看 MTD 的信息：

```
# cat /proc/mtd
dev:    size   erasesize  name
mtd0: 00800000 00020000 "kernel"
mtd1: 07800000 00020000 "rootfs"
#
```

设备与代码修改的效果相同。

也可以使用下面的命令：

```
# cat /proc/partitions
major minor  #blocks  name

  31     0       8192 mtdblock0
  31     1     122880 mtdblock1
   8     0      64000 sda
   8     1      63984 sda1
# _
```

将显示 MTD 设备和 U 盘的信息。

执行下面命令以格式化设备：

```
# cd /usr/sbin/
# ls
boa             flash_lock      jffs2dump       nftldump
chroot          flash_otp_dump  jffs3dump       rdate
crond           flash_otp_info  mkfs.jffs       sumtool
doc_loadbios    flash_unlock    mkfs.jffs2      telnetd
docfdisk        flashcp         mtd_debug       udhcpd
flash_erase     ftl_check       nanddump
flash_eraseall  ftl_format      nandwrite
flash_info      inetd           nftl_format
# flash_eraseall /dev/mtd1
Erasing 128 Kibyte @ 77e0000 -- 99 % complete.
#
```

使用下面命令 mount 新的 MTD 设备到系统：

```
# cd /mnt
# mkdir mtd_mnt
# ls
mtd_mnt
# cd mtd_mnt/
# cd ..
# ls -al
drwxr-xr-x    3 4994     90           1024 Jan  1 00:12 .
drwxr-xr-x   21 4994     90           1024 Oct  2  2006 ..
drwxr-xr-x    2 root     root         1024 Jan  1 00:12 mtd_mnt
# mount -v -t jffs2 /dev/mtdblock1 /mnt/mtd_mnt/
# cd mtd_mnt/
# ls -al
drwxr-xr-x    3 root     root            0 Jan  1 00:00 .
drwxr-xr-x    3 4994     90           1024 Jan  1 00:12 ..
# _
```

为了测试方便，需要设置正确的系统时间：

```
# date 031613532008
Sun Mar 16 13:53:00 UTC 2008
#
```

然后尝试在 MTD 设备上建立一些文件：

```
# cd /mnt/mtd_mnt/
# vi test.txt_
```

使用 vi 随意输入一定的内容，比如：

```
This is a test file on /dev/mtdblock1.
SAM9261EK NAND flash
www.mcuzone.com
~
~
```

然后保存，可以看到文件创建成功：

```
# ls -al
drwxr-xr-x    3 root     root             0 Mar 16 13:57 .
drwxr-xr-x    3 4994     90            1024 Jan  1  1970 ..
-rw-r--r--    1 root     root            76 Mar 16 13:57 test.txt
# cat test.txt
This is a test file on /dev/mtdblock1.
SAM9261EK NAND flash
www.mcuzone.com
#
```

也可以试着从别的地方复制一些文件到该文件夹：

```
# cp /bin/* ./
# ls
addgroup    cp          egrep       kill        mv          run-parts   touch
adduser     cpio        false       ln          netstat     sed         true
ash         date        fdflush     login       pidof       sh          umount
bash        dd          fgrep       ls          ping        sleep       uname
busybox     delgroup    getopt      mkdir       ps          stty        usleep
cat         deluser     grep        mknod       pwd         su          vi
chgrp       df          gunzip      mktemp      rm          sync        zcat
chmod       dmesg       gzip        more        rmdir       tar
chown       echo        hostname    mount       rpm         test.txt
# _
```

使用如下命令，可以看到系统中文件系统的使用情况：

```
# df -k
Filesystem           1k-blocks      Used Available Use% Mounted on
/dev/ram0                 3963      2091      1668  56% /
/dev/mtdblock1          122880     33076     89804  27% /mnt/mtd_mnt
#
```

由此，MTD 设备被 mount 到系统，并被正确的使用了起来。

与此类似，可以尝试 mount U 盘到系统：

```
# pwd
/mnt
# mkdir USB
# ls
USB  mtd
# mount -t vfat /dev/sda1 /mnt/USB/
# cd USB/
# ls -al
drwxr-xr-x    2 root     root           512 Jan  1 00:00 .
drwxr-xr-x    4 4994     90            1024 Jan  1 00:04 ..
-rwxr-xr-x    1 root     root       4194304 Oct  2  2006 armv5l-uclibc-sam9260
-rwxr-xr-x    1 root     root       1307192 Mar 16  2008 uImage
-rwxr-xr-x    1 root     root       1097960 Sep 20  2007 uImage3.bin
# _
```

9. 验证 MTD 设备

由于上面的测试文件是建立在 NAND 上，因此系统关机不会影响到这些文件。可以测试如下。

给系统断电，等待一会(仅为测试)，再次上电。

Linux 启动后，执行如下命令：

```
# df -k
Filesystem           1k-blocks      Used Available Use% Mounted on
/dev/ram0                 3963      2090      1669  56% /
# _
```

当前只有 ramdisk。下面需要 mount MTD 设备，因为 mount 点存在于 ramdisk，掉电后已丢失。

再次 mount，可以看到以前创建的文件均正常。MTD 设备被系统识别并使用。

```
# mkdir /mnt/mtd
# mount -t jffs2 /dev/mtdblock1 /mnt/mtd/
# cd /mnt/mtd/
# ls
addgroup   cp         egrep      kill       mv         run-parts  touch
adduser    cpio       false      ln         netstat    sed        true
ash        date       fdflush    login      pidof      sh         umount
bash       dd         fgrep      ls         ping       sleep      uname
busybox    delgroup   getopt     mkdir      ps         stty       usleep
cat        deluser    grep       mknod      pwd        su         vi
chgrp      df         gunzip     mktemp     rm         sync       zcat
chmod      dmesg      gzip       more       rmdir      tar
chown      echo       hostname   mount      rpm        test.txt
# cat test.txt
This is a test file on /dev/mtdblock1.
SAM9261EK NAND flash
www.mcuzone.com
# df -k
Filesystem           1k-blocks      Used Available Use% Mounted on
/dev/ram0                 3963      2091      1668  56% /
/dev/mtdblock1          122880     33080     89800  27% /mnt/mtd
#
```

第七章：使用 busybox 制作根文件系统

本文叙述使用 busybox 创建 Linux 的根文件系统的过程。包含了 busybox 的编译，初始化文件的建立，mtd 工具的编译及 jffs2 格式根文件映像的生成。Linux 版本选择了 2.6.24，开发环境选择了 Virtual PC，其它软件可以类推。

一，准备工作

1．使用 Virtual PC 安装 Linux 虚拟机

请参照本站另一篇文档《搭建基于 VPC 的 Linux 平台》。

2． 下载相关软件包

下载下列软件(可在本站 ftp 上下载)，并传输到 Linux 虚拟机下。

busybox-1.11.2.tar.bz2

zlib-1.2.3.tar.gz

mtd-snapshot-20050519.tar.bz2

arm-2008q1-126-arm-none-linux-gnueabi-i686-pc-linux-gnu.tar.bz2

将软件包展开，并将安装工具链 arm-none-linux-gnueabi，添加其路径到系统 PATH。

3． 准备相关环境

请参照本站另一篇文章《为 SAM926X 编译 U-boot》为 9261 编译好 u-boot。

请参照本镇另一篇文章《9261 上的 Linux 初步应用 3》设置好主机的 NFS 等环境。

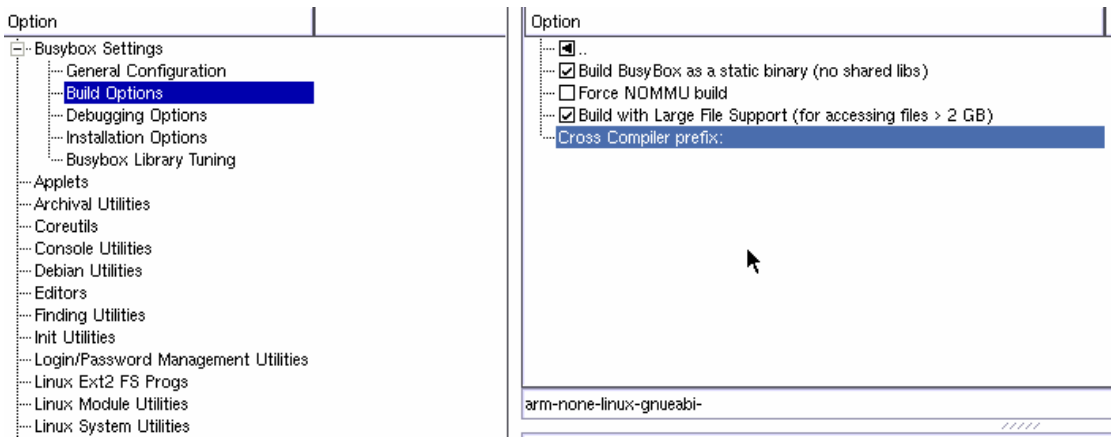
二，配置和编译 busybox

1，配置 busybox

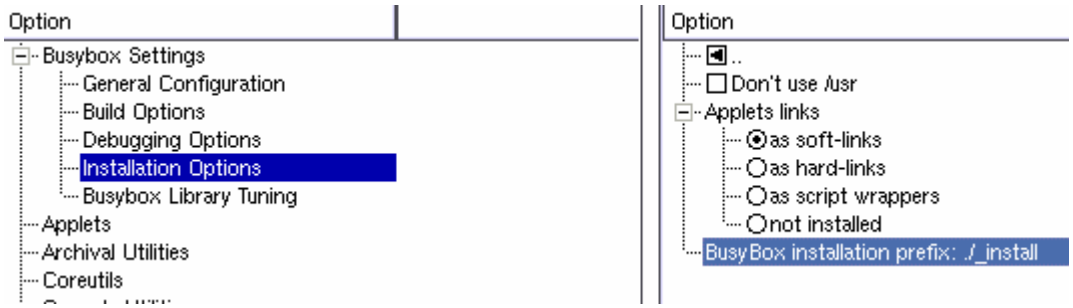
在编译之前，必须要对 busybox 进行相应的配置：

```
[root@Linux4SAM busybox-1.11.2]# ls
applets     console-tools  editors     INSTALL     Makefile         modutils       README    sysklogd
arch        coreutils      examples    libbb       Makefile.custom  networking     runit     testsuite
archival    debianutils    findutils   libpwdgrp   Makefile.flags   nobody:nobody  scripts   TODO
AUTHORS     docs           include     LICENSE     Makefile.help    printutils     selinux   TODO_config_nommu
Config.in   e2fsprogs      init        loginutils  miscutils        procps         shell     util-linux
[root@Linux4SAM busybox-1.11.2]# make xconfig
```

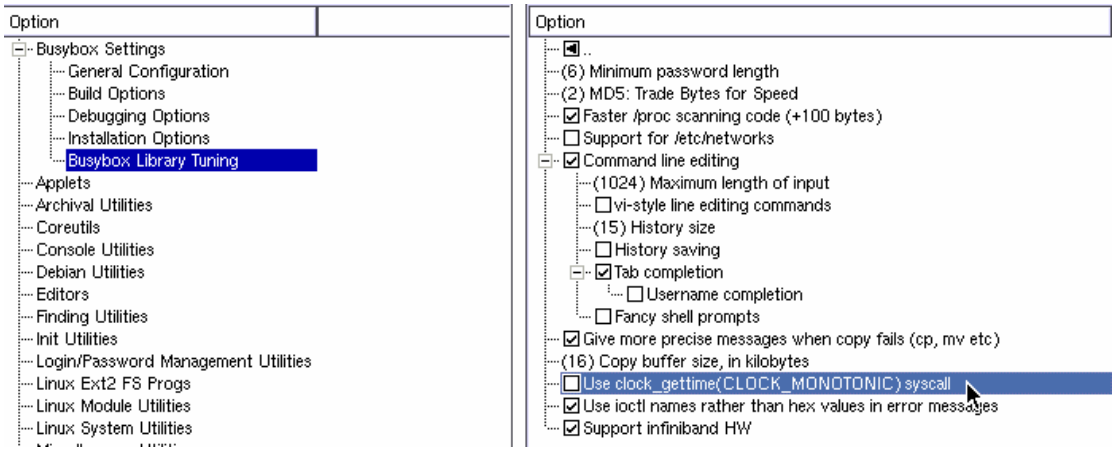
配置编译为静态链接，设置工具链为 arm-none-linux-gnueabi:



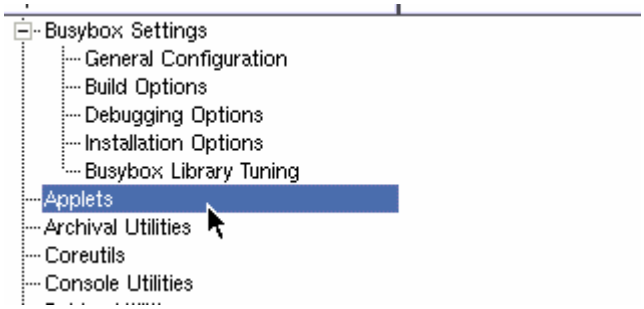
设置安装路径：



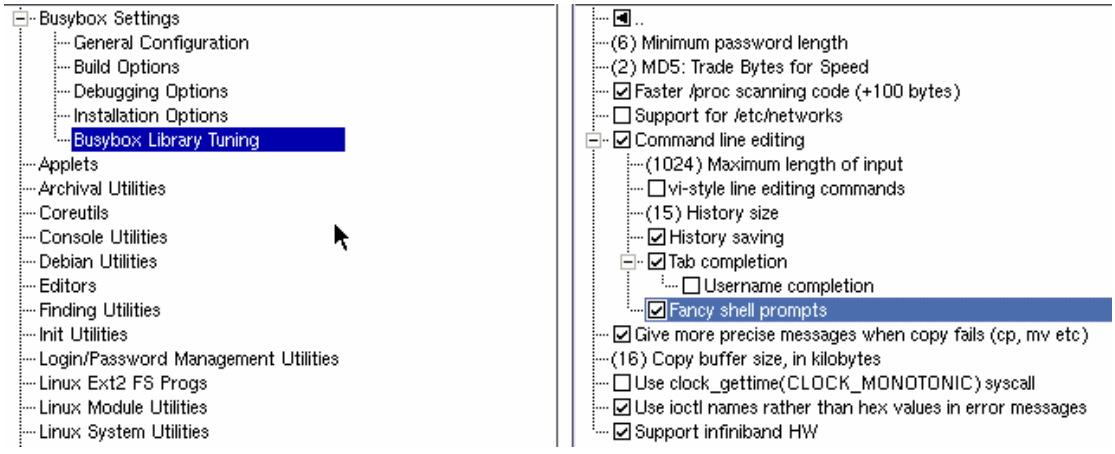
一些设置：

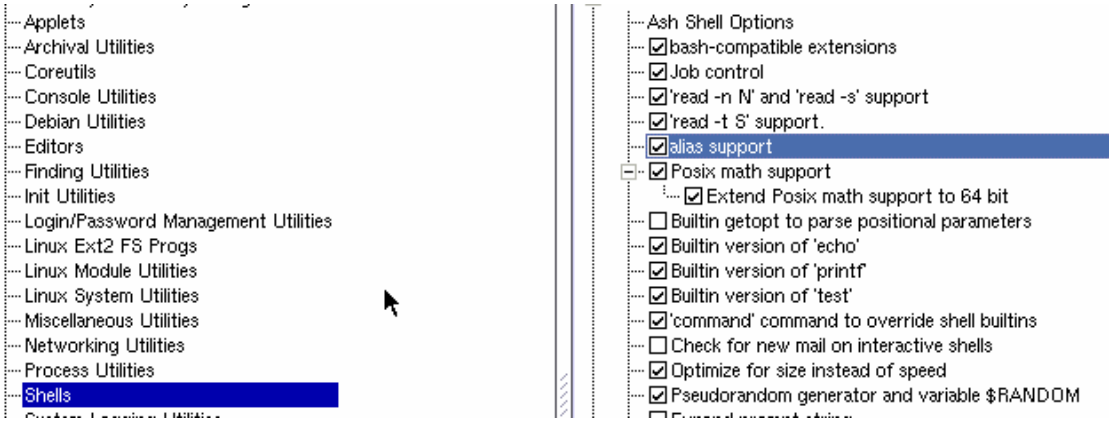


Applets 选项以下都是一些具体的 applet 配置选项，根据实际需要选择：

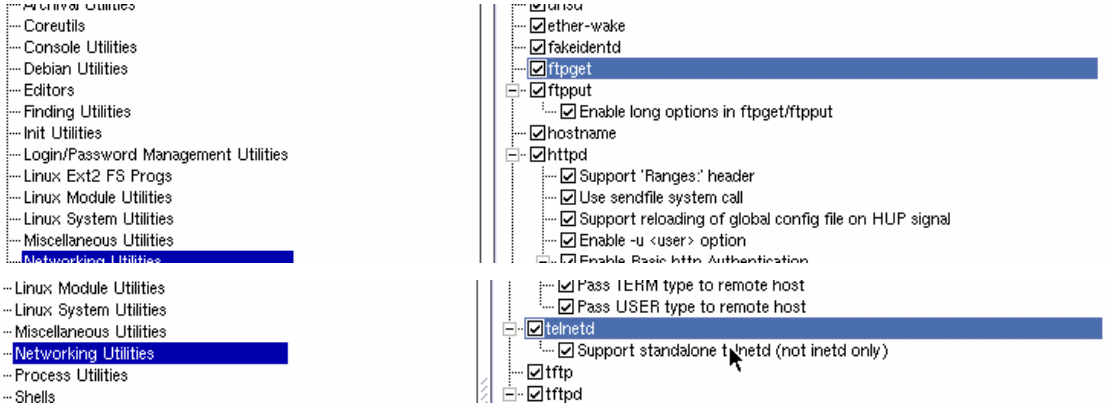


配置 prompt 以及 alias：





网络设置中的 ftp 和 telnetd 相关服务：



设置完成，保存退出。

2，编译 busybox

在命令行输入 make 即可开始编译:

```
[root@Linux4SAM busybox-1.11.2]# make
```

等待一段时间后，编译即会完成，输入下面命令：

```
[root@Linux4SAM busybox-1.11.2]# make install

  ./_install/usr/sbin/readprofile -> ../../bin/busybox
  ./_install/usr/sbin/setlogcons -> ../../bin/busybox
  ./_install/usr/sbin/telnetd -> ../../bin/busybox
  ./_install/usr/sbin/udhcpd -> ../../bin/busybox

----------------------------------------------------
You will probably need to make your busybox binary
setuid root to ensure all configured applets will
work properly.
----------------------------------------------------

[root@Linux4SAM busybox-1.11.2]# ls _install/
bin  linuxrc  sbin  usr
[root@Linux4SAM busybox-1.11.2]#
```

在该过程，一系列 softlink 被生成,完成后可以看到生成了相应的目录结构。

三，制作根文件系统

1，建立根文件系统结构

在 linux 虚拟机的 NFS 所在目录建立一个 rootfs 文件夹:

```
[root@Linux4SAM busybox-1.11.2]# mkidr /nfs4arm/rootfs/
```

建立相关文件夹：

```
[root@Linux4SAM busybox-1.11.2]# cd /nfs4arm/rootfs/
[root@Linux4SAM rootfs]# mkdir bin dev etc home lib mnt proc sbin sys tmp var usr
[root@Linux4SAM rootfs]# ls
bin  dev  etc  home  lib  mnt  proc  sbin  sys  tmp  usr  var
[root@Linux4SAM rootfs]#
```

2，复制文件

首先将 busybox 编译后生成的文件都复制到 rootfs：

```
[root@Linux4SAM rootfs]# cp -a /usr/work/app/busybox-1.11.2/_install/* ./
[root@Linux4SAM rootfs]# ls bin
addgroup  chown     dmesg     getopt    iplink    login       mt            ps          sleep  umount
adduser   cp        dumpkmap  grep      iproute   ls          mv            pwd         stat   uname
ash       cpio      echo      gunzip    iprule    mkdir       netstat       rm          stty   uncompress
busybox   date      ed        gzip      iptunnel  mknod       nice          rmdir       su     usleep
cat       dd        egrep     hostname  kill      mktemp      pidof         rpm         sync   vi
catv      delgroup  false     ip        linux32   more        ping          run-parts   tar    watch
chgrp     deluser   fdflush   ipaddr    linux64   mount       pipe_progress setarch     touch  zcat
chmod     df        fgrep     ipcalc    ln        mountpoint  printenv      sh          true
[root@Linux4SAM rootfs]# ls usr/sbin/
brctl     chroot  dhcprelay  fakeidentd  httpd  lpd    readprofile  telnetd
chpasswd  crond   dnsd       fbset       inetd  rdate  setlogcons   udhcpd
[root@Linux4SAM rootfs]#
```

复制编译器所带的库文件到 lib：

```
[root@Linux4SAM rootfs]# ls /opt/codesourcery/arm-none-linux-gnueabi/libc/armv4t/lib/
ld-2.5.so               libcrypt-2.5.so  libm.so.6             libnss_hesiod-2.5.so  libresolv-2.5.so
ld-linux.so.3           libcrypt.so.1    libnsl-2.5.so         libnss_hesiod.so.2    libresolv.so.2
libanl-2.5.so           libc.so.6        libnsl.so.1           libnss_nis-2.5.so     librt-2.5.so
libanl.so.1             libdl-2.5.so     libnss_compat-2.5.so  libnss_nisplus-2.5.so librt.so.1
libBrokenLocale-2.5.so  libdl.so.2       libnss_compat.so.2    libnss_nisplus.so.2   libSegFault.so
libBrokenLocale.so.1    libgcc_s.so      libnss_dns-2.5.so     libnss_nis.so.2       libthread_db-1.0.so
libc-2.5.so             libgcc_s.so.1    libnss_dns.so.2       libpcprofile.so       libthread_db.so.1
libcidn-2.5.so          libm-2.5.so      libnss_files-2.5.so   libpthread-2.5.so     libutil-2.5.so
libcidn.so.1            libmemusage.so   libnss_files.so.2     libpthread.so.0       libutil.so.1
[root@Linux4SAM rootfs]# cp -a /opt/codesourcery/arm-none-linux-gnueabi/libc/armv4t/lib/* ./lib
```

```
[root@Linux4SAM lib]# ls
ld-2.5.so               libc.so.6             libnss_compat.so.2    libpcprofile.so     libtdb.so.1
ld-linux.so.3           libdl-2.5.so          libnss_dns-2.5.so     libpthread-2.5.so   libthread_db-1.0.so
libanl-2.5.so           libdl.so.2            libnss_dns.so.2       libpthread.so.0     libthread_db.so.1
libanl.so.1             libgcc_s.so           libnss_files-2.5.so   libresolv-2.5.so    libutil-2.5.so
libBrokenLocale-2.5.so  libgcc_s.so.1         libnss_files.so.2     libresolv.so.2      libutil.so.1
libBrokenLocale.so.1    libm-2.5.so           libnss_hesiod-2.5.so  librt-2.5.so        libwbclient.so
libc-2.5.so             libmemusage.so        libnss_hesiod.so.2    librt.so.1          libwbclient.so.0
libcidn-2.5.so          libm.so.6             libnss_nis-2.5.so     libSegFault.so      modules
libcidn.so.1            libnsl-2.5.so         libnss_nisplus-2.5.so libtalloc.so
libcrypt-2.5.so         libnsl.so.1           libnss_nisplus.so.2   libtalloc.so.1
libcrypt.so.1           libnss_compat-2.5.so  libnss_nis.so.2       libtdb.so
[root@Linux4SAM lib]# du -m
1       ./modules/2.6.26
1       ./modules
4       .
[root@Linux4SAM lib]#
```

3，新建相关文件

首先删除 rootfs 下的 linuxrc 文件。

创建为 shell 导入全局变量的/etc/profile 文件:

```
[root@Linux4SAM etc]# cat profile
#!/bin/sh
#/etc/profile:system-wide .profile file for the Bourne shells

echo "Processing /etc/profile... "
# no-op

# Set search library path
echo "Set search library path in /etc/profile"
export LD_LIBRARY_PATH=/lib:/usr/lib

# host name
HOSTNAME=`/bin/hostname`
export HOSTNAME

# Set user path
echo "Set user path in /etc/profile"
PATH=/bin:/sbin:/usr/bin:/usr/sbin
export PATH

# alias
alias ll="ls -al"

# prompt
USER="`id -un`"
LOGNAME=$USER
PS1='[\u@\h \W]\$'
#export PS1

echo "Welcome to Linux"
[root@Linux4SAM etc]#
```

创建初始化文件/etc/inittab：

```
[root@Linux4SAM etc]# cat inittab
::sysinit:/etc/init.d/rcS

# The following 2 does not need login
#::respawn:-/bin/sh
#::respawn:/sbin/getty 115200 ttyS0 vt100 -n -l /bin/sh

# login needed
::respawn:/sbin/getty 115200 ttyS0 vt100

#ttyS0::respawn:/sbin/getty 115200 ttyS0
::restart:/sbin/init
::ctrlaltdel:/sbin/reboot
::shutdown:/bin/umount -a -r
::shutdown:/sbin/swapoff -a
```

以及/etc/fstab:

```
[root@Linux4SAM etc]# cat fstab
# stock fstab
#rootfs               /                    auto       defaults              1  1
proc                 /proc                proc       defaults              0  0
devpts               /dev/pts             devpts     mode=0620,gid=5       0  0
usbfs                /proc/bus/usb        usbfs      defaults              0  0
tmpfs                /var/volatile        tmpfs      size=1M               0  0

# uncomment this if your device has a SD/MMC/Transflash slot
#/dev/mmcblk0p1       /media/card          auto       defaults,sync,noauto  0  0
[root@Linux4SAM etc]#
```

使用了 1MB 内存作为 tmpfs，可以用于存放一些运行时的 log(关机会丢失，但可以避免反复读写 NAND)。

增加初始化脚本/etc/init.d/rcS:

```
[root@Linux4SAM init.d]# cat rcS
#! /bin/sh
runlevel=S
export runlevel

# host name
/bin/hostname mcuzone

# mount /proc if not
[ -d "/proc/1" ] || mount /proc

echo "----------mount all----------"
/bin/mount -av

# read the busybox docs: mdev.txt
echo "----------Starting mdev------"
echo "This may take some time ...  "
/bin/mount -t sysfs sysfs /sys
/bin/echo /sbin/mdev > /proc/sys/kernel/hotplug
mdev -s

# when mdev is mounted, /sys can be umounted
#/bin/umount /sys

# local service
/etc/init.d/rc.local

echo "************************************************************"
echo " www.mcuzone.com "
echo " Linux for at91SAM "
echo "************************************************************"
```

修改其属性：

```
[root@Linux4SAM etc]# ls -al ./init.d/rcS
-rw-r--r--    1 root     root           326 Sep  7 20:05 ./init.d/rcS
[root@Linux4SAM etc]# chmod 775 ./init.d/rcS
[root@Linux4SAM etc]# ls -al ./init.d/rcS
-rwxrwxr-x    1 root     root           326 Sep  7 20:05 ./init.d/rcS
[root@Linux4SAM etc]#
```

建立/etc/init.d/rc.local，启动本地服务，下面例子中会启动 telnetd:

```
[root@Linux4SAM init.d]# cat rc.local
#!/bin/sh
echo "************************************************************"
echo "local service:"
echo "telnetd start"
telnetd

echo "boa start"
mkdir /var/log/boa
boa

echo "Samba start"
nmbd -D
smbd -D

[root@Linux4SAM init.d]#
```

完成后需要修改其属性 775。

创建 mdev 配置文件/etc/mdev.conf，内容可以为空：

```
[root@Linux4SAM rootfs]# touch etc/mdev.conf
[root@Linux4SAM rootfs]#
```

配置 name server 配置文件/etc/resolv.conf:

```
[root@Linux4SAM rootfs]# cat etc/resolv.conf
nameserver 192.168.1.1

[root@Linux4SAM rootfs]#
```

创建/etc/host.conf:

```
[root@Linux4SAM etc]# cat host.conf
order hosts,bind
[root@Linux4SAM etc]#
```

创建/etc/hosts:

```
[root@Linux4SAM etc]# cat hosts
127.0.0.1        localhost:localdomain          localhost
[root@Linux4SAM etc]#
```

创建网络应用程序用到的标准服务端口映射表/etc/services:

```
[root@Linux4SAM etc]# cat services
tcpmux  1/tcp
tcpmux  1/udp
ftp-data        20/tcp
ftp     21/tcp
ssh     22/tcp
ssh     22/udp
telnet  23/tcp
nameserver      42/tcp  name
syslog  514/udp
[root@Linux4SAM etc]#
```

建立库相关的/etc/ld.so.conf：

```
[root@Linux4SAM etc]# cat ld.so.conf
/lib
/usr/lib
[root@Linux4SAM etc]#
```

建立用户和组相关文件

```
[root@Linux4SAM etc]# cat passwd
root::0:0:root:/home/root:/bin/sh
nobody:*:65534:65534:nobody:/nonexistent:/bin/sh
[root@Linux4SAM etc]#
```

```
[root@Linux4SAM etc]# cat group
root:*:0:
nobody:*:65534:
[root@Linux4SAM etc]#
```

为 root 建立 home 目录：

```
[root@Linux4SAM home]# pwd
/nfs4arm/rootfs/home
[root@Linux4SAM home]# mkdir root
[root@Linux4SAM home]#
```

4，建立设备节点

在/dev 文件夹下建立相关设备节点：

```
[root@Linux4SAM dev]# pwd
/nfs4arm/rootfs/dev
[root@Linux4SAM dev]# mknod console c 5 1
[root@Linux4SAM dev]# mknod null c 1 3
[root@Linux4SAM dev]# ls -al
total 8
drwxr-xr-x    2 root     root          4096 Sep  7 20:20 .
drwxr-xr-x   14 nobody   nobody        4096 Sep  7 19:53 ..
crw-r--r--    1 root     root       5,    1 Sep  7 20:20 console
crw-r--r--    1 root     root       1,    3 Sep  7 20:20 null
[root@Linux4SAM dev]#
```

5，安装 APP

复制一些 app 到系统：

```
[root@Linux4SAM usr]# ls
bin  info  lib  libexec  man  sbin  share
[root@Linux4SAM usr]# mkdir testapp
[root@Linux4SAM usr]# cd testapp/
[root@Linux4SAM testapp]# pwd
/nfs4arm/rootfs/usr/testapp
[root@Linux4SAM testapp]#
```

```
[root@Linux4SAM testapp]# ls -al
total 16
drwxr-xr-x    2 root     root         4096 Sep  7 21:22 .
drwxr-xr-x   10 root     root         4096 Sep  7 21:21 ..
-rwxr-xr-x    1 root     root         5185 Sep  7 21:22 hello
[root@Linux4SAM testapp]# file hello
hello: ELF 32-bit LSB executable, ARM, version 1 (SYSV), for GNU/Linux 2.6.14,
t stripped
[root@Linux4SAM testapp]#
```

四，测试根文件系统

为了测试 rootfs，可以先用 NFS 方式启动系统。

在 uboot 下设置 linux 启动参数：

```
U-Boot> setenv bootargs bootargs=mem=64M console=ttyS0,115200 root=/dev/nfs rw n
fsroot=192.168.1.5:/nfs4arm/rootfs ip=192.168.1.100:192.168.1.1:255.255.255.0
```

指定根文件系统，也就是 rootfs 采用 nfs 方式。

连接开发板到网络，启动后的效果：

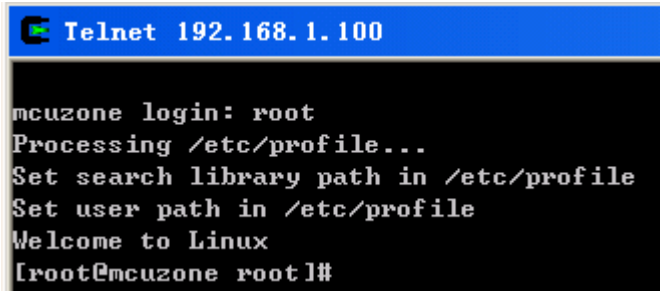
```
VFS: Mounted root (nfs filesystem).
Freeing init memory: 152K
init started: BusyBox v1.11.2 (2008-09-09 21:57:17 CST)
starting pid 837, tty '': '/etc/init.d/rcS'
----------mount all----------
----------Starting mdev------
This may take some time ...
************************************************************
local service:
telnetd start
boa start
Samba start
************************************************************
 www.mcuzone.com
 Linux for at91SAM
************************************************************
starting pid 854, tty '': '/sbin/getty 115200 ttyS0 vt100 '

mcuzone login:
```

使用 root 登陆。

```
  846 root      1712 S    telnetd
  849 nobody    1908 S    boa
  851 root      5032 S    nmbd -D
  853 root      8624 S    smbd -D
  854 root      1712 S    -sh
  855 root      8624 S    smbd -D
  856 root      1712 S    -sh
  861 root      1712 R    ps
```

Telnetd, boa,Samba 已经运行。

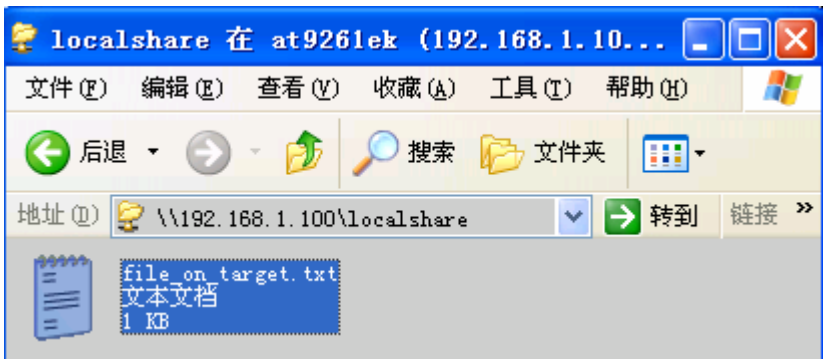
在 pc 上 telnet 上开发板：



到/etc 下运行 ldconfig 生成 ld.so.cache：

在 PC 上打开一个网页，浏览开发板 ip 地址：

浏览板子上的网络共享文件夹：



使用 mplayer：

```
[root@mcuzone mtd]#mplayer -vf rotate=2 ETERNAL_LEGEND.wmv
```

LCD 显示：

五，生成 jffs2

1，编译 zlib

将 zlib 展开，首先配置：

```
[root@Linux4SAM zlib-1.2.3]# ./configure -prefix=/opt/codesourcery/arm-none-linux-gnueabi --shared
Checking for gcc...
Checking for shared library support...
Building shared library libz.so.1.2.3 with gcc.
Checking for unistd.h... Yes.
Checking whether to use vs[n]printf() or s[n]printf()... using vs[n]printf()
Checking for vsnprintf() in stdio.h... Yes.
Checking for return value of vsnprintf()... Yes.
Checking for errno.h... Yes.
Checking for mmap support... Yes.
[root@Linux4SAM zlib-1.2.3]#
```

修改 Makefile，使用 arm-none-linux-gnueabi-gcc 编译:

```
CC=arm-none-linux-gnueabi-gcc

CFLAGS=-fPIC -O3 -mcpu=arm926ej-s -DUSE_MMAP
#CFLAGS=-O -DMAX_WBITS=14 -DMAX_MEM_LEVEL=7
#CFLAGS=-g -DDEBUG
#CFLAGS=-O3 -Wall -Wwrite-strings -Wpointer-arith -Wconversion \
#           -Wstrict-prototypes -Wmissing-prototypes

LDFLAGS=-L. libz.so.1.2.3
LDSHARED=arm-none-linux-gnueabi-gcc -shared -Wl,-soname,libz.so.1
CPP=arm-none-linux-gnueabi-gcc -E

LIBS=libz.so.1.2.3
SHAREDLIB=libz.so
SHAREDLIBV=libz.so.1.2.3
SHAREDLIBM=libz.so.1
```

输入命令 make，然后 make install：

```
[root@Linux4SAM zlib-1.2.3]# make install
cp zlib.h zconf.h /opt/codesourcery/arm-none-linux-gnueabi/include
chmod 644 /opt/codesourcery/arm-none-linux-gnueabi/include/zlib.h /opt/codesourcery/arm-none-linux-gnueabi/include/zconf
.h
cp libz.so.1.2.3 /opt/codesourcery/arm-none-linux-gnueabi/lib
cd /opt/codesourcery/arm-none-linux-gnueabi/lib; chmod 755 libz.so.1.2.3
cd /opt/codesourcery/arm-none-linux-gnueabi/lib; if test -f libz.so.1.2.3; then \
  rm -f libz.so libz.so.1; \
  ln -s libz.so.1.2.3 libz.so; \
  ln -s libz.so.1.2.3 libz.so.1; \
  (ldconfig || true)  >/dev/null 2>&1; \
fi
cp zlib.3 /opt/codesourcery/arm-none-linux-gnueabi/share/man/man3
chmod 644 /opt/codesourcery/arm-none-linux-gnueabi/share/man/man3/zlib.3
[root@Linux4SAM zlib-1.2.3]#
```

安装后的效果:

```
[root@Linux4SAM zlib-1.2.3]# ls /opt/codesourcery/arm-none-linux-gnueabi/include/
c++  zconf.h  zlib.h
[root@Linux4SAM zlib-1.2.3]# ls /opt/codesourcery/arm-none-linux-gnueabi/lib
armv4t  boards  ldscripts  libiberty.a  libsupc++.a  libz.so  libz.so.1  libz.so.1.2.3  thumb2
[root@Linux4SAM zlib-1.2.3]#
```

2，编译 MTD 工具

将 mtd-snapshot-20050519.tar.bz2 展开，首先编译开发板上可用的版本。

cd 到 util 路径，修改 Makefile，修改工具链，并设置好 DESTDIR：

```
DESTDIR=/nfs4arm/rootfs
SBINDIR=/usr/sbin
MANDIR=/usr/man
INCLUDEDIR=/usr/include
CROSS=arm-none-linux-gnueabi-
CC := $(CROSS)gcc
CFLAGS := -I../include -O2 -Wall

TARGETS = ftl_format flash_erase flash_eraseall nanddump doc_loadbios \
        mkfs.jffs ftl_check mkfs.jffs2 flash_lock flash_unlock flash_info \
        flash_otp_info flash_otp_dump mtd_debug flashcp nandwrite \
        jffs2dump jffs3dump \
        nftldump nftl_format docfdisk \
        sumtool #jffs2reader
```

输入 make 开始编译，完成后生成一些列工具：

```
[root@Linux4SAM util]# ls
checkfs         crc32.c           flash_eraseall    flash_otp_lock.c   jffs3dump     mkfs.jffs2.1    nftldump.c
compr.c         crc32.h           flash_eraseall.c  flash_otp_write.c  jffs3dump.c   mkfs.jffs2.c    nftl_format
compr.h         crc32.o           flash_eraseall.o  flash_unlock       jffs3dump.o   mkfs.jffs2.o    nftl_format.c
compr_lzari.c   CVS               flash_erase.c     flash_unlock.c     jffs3.h       mkfs.jffs.c     summary.h
compr_lzari.o   device_table.txt  flash_info        ftl_check          jffs-dump.c   mtd_debug       sumtool
compr_lzo.c     docfdisk          flash_info.c      ftl_check.c        jittertest    mtd_debug.c     sumtool.c
compr_lzo.o     docfdisk.c        flash_lock        ftl_format         MAKEDEV       mtd-utils.spec  sumtool.o
compr.o         doc_loadbios      flash_lock.c      ftl_format.c       Makefile      nanddump
compr_rtime.c   doc_loadbios.c    flash_otp_dump    jffs2dump          Makefile.am   nanddump.c
compr_rtime.o   flashcp           flash_otp_dump.c  jffs2dump.c        mkfs.ffs2.c   nandwrite
compr_zlib.c    flashcp.c         flash_otp_info    jffs2dump.o        mkfs.jffs     nandwrite.c
compr_zlib.o    flash_erase       flash_otp_info.c  jffs2reader.c      mkfs.jffs2    nftldump
[root@Linux4SAM util]# file mkfs.jffs2
mkfs.jffs2: ELF 32-bit LSB executable, ARM, version 1 (SYSV), for GNU/Linux 2.6.14, dynamically linked (uses shared libs
), not stripped
[root@Linux4SAM util]#
```

运行 make install 安装相应的路径：

```
[root@Linux4SAM util]# make install
mkdir -p /nfs4arm/rootfs//usr/sbin
install -m0755 -oroot -groot ftl_format flash_erase flash_eraseall n
flash_lock flash_unlock flash_info flash_otp_info flash_otp_dump mtd
p nftl_format docfdisk sumtool  /nfs4arm/rootfs//usr/sbin/
mkdir -p /nfs4arm/rootfs//usr/man/man1
gzip -c mkfs.jffs2.1 > /nfs4arm/rootfs//usr/man/man1/mkfs.jffs2.1.gz
mkdir -p /nfs4arm/rootfs//usr/include/mtd
install -m0644 -oroot -groot ../include/mtd/*.h /nfs4arm/rootfs//usr
[root@Linux4SAM util]# ls /nfs4arm/rootfs/usr/sbin/
boa         dnsd          flashcp         flash_otp_info  inetd
brctl       docfdisk      flash_erase     flash_unlock    jffs2dump
chpasswd    doc_loadbios  flash_eraseall  ftl_check       jffs3dump
chroot      fakeidentd    flash_info      ftl_format      lpd
crond       fbset         flash_lock      httpd           mkfs.jffs
dhcprelay   fbsnap        flash_otp_dump  iconvconfig     mkfs.jffs2
You have new mail in /var/spool/mail/root
[root@Linux4SAM util]#
```

编译 ARM 版本完成后，开始编译 PC 版本工具，使得在 PC 上可以生成 jffs2 的文件映像。首先修改 Makefile：

```
# $Id: Makefile,v 1.55 2005/02/08 17:45:52 nico Exp $
#DESTDIR=/nfs4arm/rootfs
SBINDIR=/usr/sbin
MANDIR=/usr/man
INCLUDEDIR=/usr/include
#CROSS=arm-none-linux-gnueabi-
CC := $(CROSS)gcc
CFLAGS := -I../include -O2 -Wall

TARGETS = ftl_format flash_erase flash_eraseall nanddu
        mkfs.jffs ftl_check mkfs.jffs2 flash_lock flas
        flash_otp_info flash_otp_dump mtd_debug flashc
        jffs2dump jffs3dump \
        nftldump nftl_format docfdisk \
        sumtool #jffs2reader
```

然后 make 即可生成可执行文件：

```
[root@Linux4SAM util]# file mkfs.jffs2
mkfs.jffs2: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV),
ed libs), not stripped
[root@Linux4SAM util]#
```

3，生成 jffs2 格式的 rootfs

使用命令 du –k (-m)看看当前文件系统的大小：

```
3900    ./usr/info
4       ./usr/share
12      ./usr/testapp
24      ./usr/include/mtd
28      ./usr/include
21708   ./usr
28796   .
[root@Linux4SAM rootfs]#
```

运行下面命令将 rootfs 的文件夹做成 jffs2 格式的根文件系统。-e 参数指明擦除大小为 128KB，-s 指明 page 大小为 2048：

./mkfs.jffs2 -r /nfs4arm/rootfs/ -o /nfs4arm/rootfs_mcuzone.jffs2 -e 0x20000 -s 0x800 -n

```
[root@Linux4SAM util]# ./mkfs.jffs2 -r /nfs4arm/rootfs/ -o /nfs4arm/rootfs_mcuzone.jffs2 -e 0x20000 -s 0x800 -n
```

```
-rw-r--r--    1 root     root      15350316 Sep 11 20:24 rootfs_mcuzone.jffs2
```

```
[root@Linux4SAM nfs4arm]# ls *.jffs2
rootfs_mcuzone.jffs2
[root@Linux4SAM nfs4arm]#
```

运行开发板，首先使用 NFS 方式。一路 cd 到/usr/sbin 下:

```
[root@mcuzone sbin]#pwd
/usr/sbin
[root@mcuzone sbin]#ls
boa             flash_erase     inetd           nmbd
brctl           flash_eraseall  jffs2dump       nscd
chpasswd        flash_info      jffs3dump       rdate
chroot          flash_lock      lpd             readprofile
crond           flash_otp_dump  mkfs.jffs       rpcinfo
dhcprelay       flash_otp_info  mkfs.jffs2      setlogcons
dnsd            flash_unlock    mplayer         smbd
doc_loadbios    flashcp         mtd_debug       sumtool
docfdisk        ftl_check       nanddump        telnetd
fakeidentd      ftl_format      nandwrite       udhcpd
fbset           httpd           nftl_format     zdump
fbsnap          iconvconfig     nftldump        zic
[root@mcuzone sbin]#_
```

看看 mtd info：

```
[root@mcuzone sbin]#mtd_debug  info /dev/mtd0
mtd.type = MTD_NANDFLASH
mtd.flags =
mtd.size = 67108864 (64M)
mtd.erasesize = 131072 (128K)
mtd.oobblock = 2048 (2K)
mtd.oobsize = 64
mtd.ecctype = (unknown ECC type - new MTD API maybe?)
regions = 0

[root@mcuzone sbin]#
```

首先擦除/dev/mtd0:

```
[root@mcuzone sbin]#flash_eraseall /dev/mtd0
Erasing 128 Kibyte @ 3fe0000 -- 99 % complete.
[root@mcuzone sbin]#_
```

在 NFS 环境下，可以将 jffs2 的映像复制到根文件系统的/usr/testapp 下：

```
[root@Linux4SAM nfs4arm]# cp /tftpboot/rootfs_mcuzone.jffs2 /nfs4arm/rootfs/usr/testapp/
[root@Linux4SAM nfs4arm]#
```

那么在板子上即可看到该文件：

```
[root@mcuzone testapp]#ls -al
drwxr-xr-x    2 root     root          4096 Sep 13  2008 .
drwxr-xr-x   11 root     root          4096 Sep 13  2008 ..
-rwxr-xr-x    1 root     root          5185 Sep  7  2008 hello
-rw-r--r--    1 root     root      15706060 Sep 13  2008 rootfs_mcuzone.jffs2
[root@mcuzone testapp]#_
```

下面需要将映像烧录到 NAND 上：

```
[root@mcuzone testapp]#nandwrite -p /dev/mtd0 rootfs_mcuzone.jffs2 _
```

烧录完成后可以重启开发板，在 u-boot 下修改启动命令参数：

```
bootargs=mem=64M console=ttyS0,115200 root=/dev/mtdblock0 rw rootfstype=jffs2 ip
=192.168.1.100:192.168.1.1:192.168.1.1:255.255.255.0

Environment size: 355/16380 bytes
U-Boot> _
```

Boot 完成后的 log：

```
VFS: Mounted root (jffs2 filesystem).
Freeing init memory: 152K
init started: BusyBox v1.11.2 (2008-09-09 21:57:17 CST)
starting pid 837, tty '': '/etc/init.d/rcS'
----------mount all----------
----------Starting mdev------
This may take some time ...
************************************************************
local service:
telnetd start
boa start
Samba start
UDC MSD start
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: /dev/mtdblock1
************************************************************
 www.mcuzone.com
 Linux for at91SAM
************************************************************
starting pid 857, tty '': '/sbin/getty 115200 ttyS0 vt100 '

mcuzone login:
```

jffs2 文件系统 mount OK。

文件系统的情况：

```
[root@mcuzone root]#df -k
Filesystem           1k-blocks      Used Available Use% Mounted on
rootfs                   65536     17108     48428  26% /
/dev/root                65536     17108     48428  26% /
tmpfs                     1024        24      1000   2% /var/volatile
[root@mcuzone root]#cat /proc/mtd
dev:    size   erasesize  name
mtd0: 04000000 00020000 "Partition 1"
mtd1: 04000000 00020000 "Partition 2"
mtd2: 00420000 00000210 "spi0.0-AT45DB321x"
[root@mcuzone root]#cat /proc/partitions
major minor  #blocks  name

  31     0      65536 mtdblock0
  31     1      65536 mtdblock1
  31     2       4224 mtdblock2
[root@mcuzone root]#
```

六，一些事项

如果编译遇到问题可以先检查一下环境，然后就是工具链。请按照上文描述的步骤操作。

第八章：创建 yaffs2 文件系统

本文叙述基于 yaffs2 创建 Linux 的根文件系统的过程。包含了 yaffs2 的配置与编译，yaffs2 格式根文件映像的生成。Linux 版本选择了 2.6.26，开发环境选择了 Virtual PC，其它软件可以类推。

一，准备工作

1. 使用 Virtual PC 安装 Linux 虚拟机
   请参照本站另一篇文档《搭建基于 VPC 的 Linux 平台》。

2. 下载相关软件包
   下载下列软件(可在本站 ftp 上下载)，并传输到 Linux 虚拟机下。
   yaffs2.tar.gz

3. 准备相关环境
   请参照本镇另一篇文章《9261 上的 Linux 初步应用 3》设置好主机的 NFS 等环境。

二，配置和编译 yaffs2

1．配置 yaffs2

首先展开 yaffs2 的压缩包：

```
[root@Linux4SAM work]# tar xzvf yaffs2.tar.gz
```

由于现在的 linux 内核(2.6.26)还没有包含对 yaffs2 的支持，需要对 linux 内核做一些修改。

首先到 linux 内核下的 fs 目录下创建 yaffs 文件夹：

```
[root@Linux4SAM fs]# mkdir yaffs
[root@Linux4SAM fs]# pwd
/usr/work/linux-2.6.26/fs
[root@Linux4SAM fs]# cd yaffs/
[root@Linux4SAM yaffs]# pwd
/usr/work/linux-2.6.26/fs/yaffs
[root@Linux4SAM yaffs]#
```

从 yaffs2 文件夹下复制相关的 yaffs 文件到 yaffs 文件夹下：

```
[root@Linux4SAM yaffs]# cp -f ../../../yaffs2/*.h ./
[root@Linux4SAM yaffs]# cp -f ../../../yaffs2/*.c ./
[root@Linux4SAM yaffs]#
```

复制 yaffs2 下的 Makefile.kernel 到 kernel 下的 yaffs 下，并重命名为 Makefile：

```
[root@Linux4SAM yaffs]# cp ../../../yaffs2/Makefile.kernel ./Makefile
[root@Linux4SAM yaffs]# pwd
/usr/work/linux-2.6.26/fs/yaffs
[root@Linux4SAM yaffs]#
```

复制 Kconfig 文件到 yaffs 文件夹：

```
[root@Linux4SAM yaffs]# cp ../../../yaffs2/Kconfig ./
[root@Linux4SAM yaffs]# pwd
/usr/work/linux-2.6.26/fs/yaffs
[root@Linux4SAM yaffs]#
```

修改 kernel 的 fs 文件夹下的 Makefile，将 yaffs 加入内核编译：

```
120 obj-$(CONFIG_OCFS2_FS)     += ocfs2/
121 obj-$(CONFIG_GFS2_FS)            += gfs2/
122 obj-$(CONFIG_YAFFS_FS)            += yaffs/
```

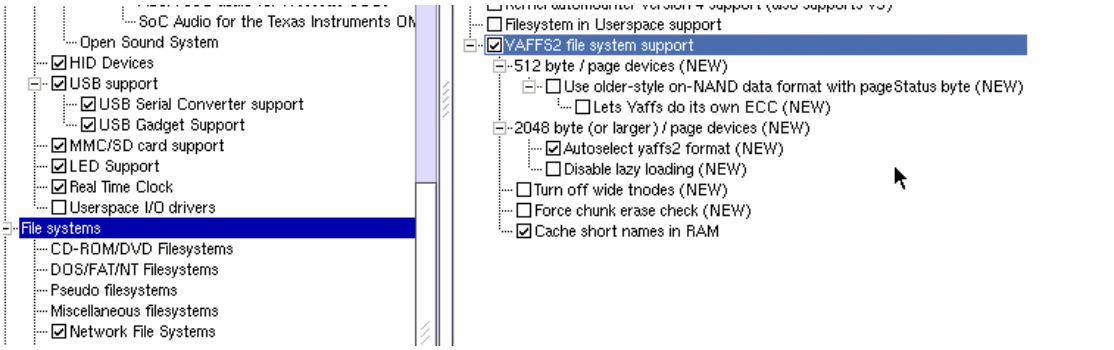
修改 kernel 的 fs 文件夹下的 Kconfig，使得可以配置 yaffs：

```
2153
2154 endif # NETWORK_FILESYSTEMS
2155
2156 source "fs/yaffs/Kconfig"
2157
2158 if BLOCK
2159 menu "Partition Types"
2160
2161 source "fs/partitions/Kconfig"
2162
2163 endmenu
2164 endif
2165
2166 source "fs/nls/Kconfig"
2167 source "fs/dlm/Kconfig"
2168
```

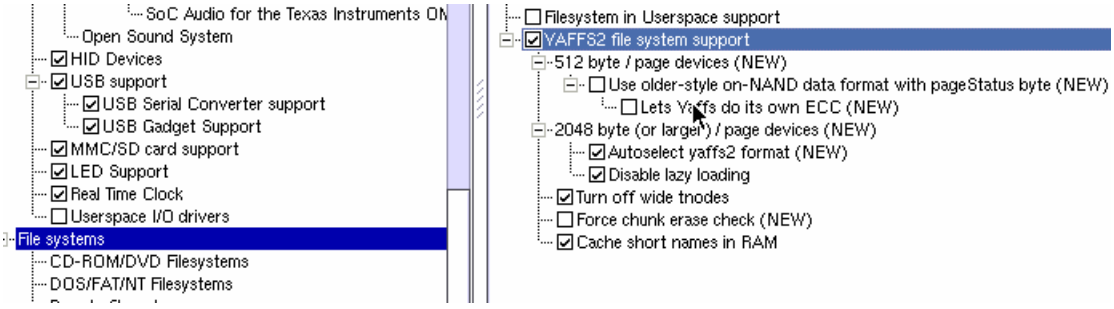
2. 编译 yaffs2

在 linux 下运行配置程序：

由于板子上使用了 2K 页的 NAND 因此需要选择对应 2048 的选项。



完成的配置项：



配置完成后保存后退出，开始编译 kernel。
Kernel 生成后需要将其转换成 u-boot 格式。

三，制作根文件系统

1．编译 yaffs2 工具

在 yaffs2/utils 下编译工具:

```
[root@Linux4SAM utils]# pwd
/usr/work/yaffs2/utils
[root@Linux4SAM utils]# make
```

生成的工具：

```
[root@Linux4SAM utils]# ls
Makefile          mkyaffs2image.o  mkyaffsimage.o  yaffs
mkyaffs2image     mkyaffsimage     yaffs_ecc.c     yaffs
mkyaffs2image.c   mkyaffsimage.c   yaffs_ecc.o     yaffs
[root@Linux4SAM utils]# file mkyaffs2image
mkyaffs2image: ELF 32-bit LSB executable, Intel 80386,
hared libs), not stripped
[root@Linux4SAM utils]#
```

注意 mkyaffs2image.c，该文件将文件夹输出为 yaffs2 的 image，由于 yaffs2 的代码使用 mtd 的驱动来读写 NAND，因此可能需要改动代码来使得生成的 image 和 mtd 的格式一致，特别是 NAND 的 oob 格式。否则可能造成 mount 后只能看到一个 lost+found 文件夹。

2．生成 yaffs2 根文件系统映像

使用工具生成根文件系统映像：

```
[root@Linux4SAM nfs4arm]# pwd
/nfs4arm
[root@Linux4SAM nfs4arm]# mkyaffs2image rootfs/ rootfs_mz.yaffs2
```

文件生成：

```
Operation complete.
557 objects in 45 directories
14791 NAND pages
[root@Linux4SAM nfs4arm]#
```

生成的根文件系统：

```
[root@Linux4SAM nfs4arm]# pwd
/nfs4arm
[root@Linux4SAM nfs4arm]# mkyaffs2image rootfs/ rootfs_mz.yaffs
[root@Linux4SAM nfs4arm]# ls -al rootfs_mz.yaffs2
-rw-------    1 root     root      31238592 Sep 16 13:35 rootfs_mz.yaffs2
[root@Linux4SAM nfs4arm]#
```

由于 yaffs2 不支持数据压缩，因此生成的 image 比 jffs2 格式要大。

3．使用 yaffs2 文件系统

首先将支持 yaffs 的新 kernel 烧录到板子上，使用 NFS 方式启动，在内核可以看到 yaffs 的支持：

```
[root@mcuzone root]#cat /proc/filesystems
nodev   sysfs
nodev   rootfs
nodev   bdev
nodev   proc
nodev   sockfs
nodev   usbfs
nodev   pipefs
nodev   anon_inodefs
nodev   tmpfs
nodev   inotifyfs
nodev   devpts
        ext2
        cramfs
nodev   ramfs
        msdos
        vfat
nodev   nfs
nodev   jffs2
        yaffs
        yaffs2
nodev   rpc_pipefs
[root@mcuzone root]#
```

为了将 yaffs2 的 image 烧录到板子上作为根文件系统，需要先擦除 mtd0：

```
[root@mcuzone root]#flash_eraseall /dev/mtd0
Erasing 128 Kibyte @ 3fe0000 -- 99 % complete.
[root@mcuzone root]#
```

在 linux PC 上将前面生成的 yaffs2 的 image 放到 NFS 的 tmp 目录下：

```
[root@Linux4SAM nfs4arm]# pwd
/nfs4arm
[root@Linux4SAM nfs4arm]# cp rootfs_mz.yaffs2 rootfs/tmp/
[root@Linux4SAM nfs4arm]# cd rootfs/tmp/
[root@Linux4SAM tmp]# ls
rootfs_mz.yaffs2
[root@Linux4SAM tmp]#
```

在开发板的/tmp 下就可以看到这个文件，使用 nandwrite 将其写到 mtd0:

```
[root@mcuzone /tmp]#ls
rootfs_mz.yaffs2
[root@mcuzone /tmp]#nandwrite -a -o /dev/mtd0 rootfs_mz.yaffs2
```

烧写完毕可以 mount 一下，进行测试：

```
[root@mcuzone /tmp]#mount -t yaffs2 /dev/mtdblock0 /mnt/mtd/
yaffs: dev is 32505856 name is "mtdblock0"
yaffs: passed flags ""
yaffs: Attempting MTD mount on 31.0, "mtdblock0"
[root@mcuzone /tmp]#
```

在 mount 成功后可以先看看 yaffs 的信息：

```
[root@mcuzone root]#cat /proc/yaffs
YAFFS built:Sep 14 2008 23:14:00
$Id: yaffs_fs.c,v 1.69 2008/08/28 02:42:11 charles Exp $
$Id: yaffs_guts.c,v 1.59 2008/07/21 01:03:19 charles Exp $

Device 0 "Partition 1"
startBlock..... 0
endBlock..... 511
totalBytesPerChunk. 2048
nDataBytesPerChunk. 2048
chunkGroupBits..... 0
chunkGroupSize..... 1
nErasedBlocks..... 248
nReservedBlocks.... 5
blocksInCheckpoint. 0
nTnodesCreated..... 1200
nFreeTnodes..... 30
nObjectsCreated.... 1400
nFreeObjects..... 137
nFreeChunks..... 17455
nPageWrites..... 0
nPageReads..... 0
nBlockErasures..... 0
nGCCopies..... 0
garbageCollections. 0
```

可以看到所 mount 的 mtdblock0 所在 NAND 的信息。

到/mnt/mtd 目录下浏览 yaffs2 文件系统中的文件：

```
[root@mcuzone /tmp]#cd /mnt/mtd/
[root@mcuzone mtd]#ls
bin         home        mnt         sys         var
dev         lib         proc        tmp
etc         lost+found  sbin        usr
[root@mcuzone mtd]#
```

四，Yaffs2 根文件系统

测试 ok 后，就可以将 mtd0 上的 yaffs2 文件系统作为根文件系统。

修改 u-boot 下的启动参数：

```
U-Boot> setenv bootargs mem=64M console=ttyS0,115200 root=/dev/mtdblock0 rw root
fstype=yaffs2 ip=192.168.1.100:192.168.1.1:192.168.1.1:255.255.255.0_
```

启动过程中可以 yaffs2 文件系统被识别并作为根文件系统：

```
     bootserver=192.168.1.1, rootserver=192.168.1.1, rootpath=
yaffs: dev is 32505856 name is "mtdblock0"
yaffs: passed flags ""
yaffs: Attempting MTD mount on 31.0, "mtdblock0"
VFS: Mounted root (yaffs2 filesystem).
Freeing init memory: 132K
init started: BusyBox v1.11.2 (2008-09-09 21:57:17 CST)
starting pid 843, tty '': '/etc/init.d/rcS'
----------mount all----------
----------Starting mdev------
This may take some time ...
************************************************************
local service:
telnetd start
boa start
Samba start
UDC MSD start
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: /dev/mtdblock1
************************************************************
 www.mcuzone.com
 Linux for at91SAM
```

系统的 mount 信息：

```
[root@mcuzone /]#mount
rootfs on / type rootfs (rw)
/dev/root on / type yaffs2 (rw)
proc on /proc type proc (rw)
devpts on /dev/pts type devpts (rw,gid=5,mode=620)
usbfs on /proc/bus/usb type usbfs (rw)
tmpfs on /var/volatile type tmpfs (rw,size=1024k)
sysfs on /sys type sysfs (rw)
[root@mcuzone /]#_
```

```
[root@mcuzone /]#df
Filesystem           1k-blocks      Used Available Use% Mounted on
rootfs                   65536     31804     33732  49% /
/dev/root                65536     31804     33732  49% /
tmpfs                     1024        24      1000   2% /var/volatile
[root@mcuzone /]#_
```

第九章：SAM9261 上的 Linux 初步应用 2

本章叙述 9261 Linux 下 APP 编程，包括了一个最简单的 hello world 的例子，以及将可执行程序下载到板子上运行的不同方法，包括使用 U 盘，tftp 方式，以及将板子作为 U 盘的方法。这些方法都侧重于使用 windows 作为主机。由于 APP 开发和底层关系小，本文基于 9261 开发板，但也适用于其它开发板。本文中开发使用的 Linux 版本选择了 2.6.24, 开发环境选择了 Virtual PC，其它软件可以类推。

一，准备工作

1. 使用 Virtual PC 安装 Linux 虚拟机

   请参照本站另一篇文档《搭建基于 VPC 的 Linux 平台》。

2. 下载相关软件包

   下载下列软件(可在本站 ftp 上下载)，并传输到 Linux 虚拟机下。

   linux4sam-angstrom-at91sam9261ek.zip

   arm-2007q1-10-arm-none-linux-gnueabi-i686-pc-linux-gnu.tar.bz2

   linux-2.6.24.tar.bz2

   2.6.24-at91.patch.gz

   2.6.24.at91.2-exp.patch.gz

   将 arm-2007q1-10-arm-none-linux-gnueabi-i686-pc-linux-gnu.tar.bz2 展开到/opt/codesourcery 下，并添加其路径到系统路径:

```
[root@Linux4SAM opt]# pwd
/opt
[root@Linux4SAM opt]# ls -al
total 16
drwxr-xr-x    4 root     root         4096 Aug 15 22:03 .
drwxr-xr-x   21 root     root         4096 Aug 24 12:34 ..
drwxr-xr-x    8 root     root         4096 Jul 27 13:10 arm-2008q1
lrwxrwxrwx    1 root     root           10 Jul 27 13:08 codesourcery -> arm-2008q1
drwxr-xr-x    2 nobody   nobody       4096 Aug 15 22:02 crosstool
[root@Linux4SAM opt]# cd codesourcery/
[root@Linux4SAM codesourcery]# ls
arm-none-linux-gnueabi  bin  include  lib  libexec  share
[root@Linux4SAM codesourcery]#
```

```
# Export Path
export PATH=$PATH:/opt/codesourcery/bin
export PATH=$PATH:/usr/local/arm/3.4.1/bin

[root@Linux4SAM codesourcery]#
```

3. 修改和编译 linux

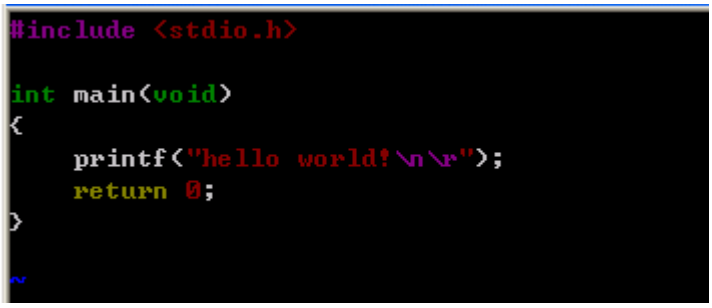
   为了在本站的板子上运行，需要修改一些 linux 的代码，过程可以参考本站另外一篇文档《9261 上的 Linux 初步应用 1》。

二，编译应用

这里以一个极其简单的例子来说明代码编程。该例子只是向标准输出打印 Hello World。

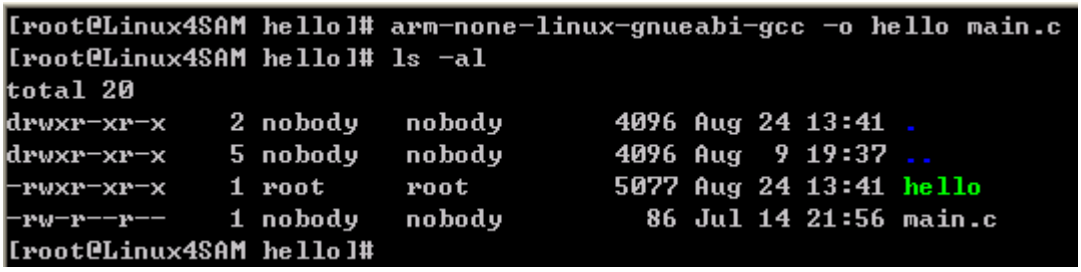
1. 编辑代码

在 linux 下编辑代码，可以使用 vi，如下图：

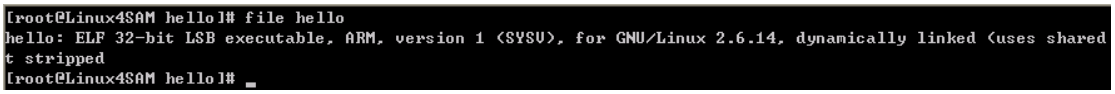


如果觉得使用 vi 不顺手，可以在 windows 下编辑好代码然后传输到 Linux 下。将代码保存为 main.c。

使用下面的命令进行编译:



编译完成就会生成 hello 这个文件，文件属性：



三，运行 Hello World 例程

代码编译完成后，需要将可执行文件放到板子上运行。下面介绍几种将文件上传到板子运行的办法。

1. U 盘

将 hello 复制到 U 盘，然后将 U 盘连接到开发板：

U 盘将被识别，并自动 mount 到/media/sda?(sda1,sda2,…):

```
root@at91sam9263ek:/media/sda1$ ls -al hello
-rwxr-xr-x    1 root     root          5077 Aug 24 13:41 hello
root@at91sam9263ek:/media/sda1$
```

开始运行：

```
root@at91sam9263ek:/media/sda1$ ls -al hello
-rwxr-xr-x    1 root     root          5077 Aug 24 13:41 hello
root@at91sam9263ek:/media/sda1$ ./hello
hello world!
root@at91sam9263ek:/media/sda1$ _
```

输出了期望的结果。

2. tftp

由于板子上有网络，也可以使用网络的方式。

首先将开发板和 PC 以网线相连，PC 端将出现连接，在 linux 端需要进行一些设置。

先看看网络配置情况：

```
root@at91sam9263ek:~$ ifconfig -a
eth0      Link encap:Ethernet  HWaddr 3A:1F:34:08:54:54
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
          Interrupt:107 Base address:0x6000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:8 errors:0 dropped:0 overruns:0 frame:0
          TX packets:8 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:576 (576.0 B)  TX bytes:576 (576.0 B)
```

设置 ip 地址：

```
root@at91sam9263ek:~$ ifconfig eth0 ipaddr 192.168.1.100
```

尝试 ping 主机：

```
eth0: link up, 100Mbps, full-duplex, lpa 0x45E1
root@at91sam9263ek:~$ ping 192.168.1.10
PING 192.168.1.10 (192.168.1.10): 56 data bytes
64 bytes from 192.168.1.10: icmp_seq=0 ttl=128 time=4.6 ms
64 bytes from 192.168.1.10: icmp_seq=1 ttl=128 time=0.3 ms
64 bytes from 192.168.1.10: icmp_seq=2 ttl=128 time=0.3 ms
64 bytes from 192.168.1.10: icmp_seq=3 ttl=128 time=0.3 ms
64 bytes from 192.168.1.10: icmp_seq=4 ttl=128 time=0.3 ms

--- 192.168.1.10 ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 0.3/1.1/4.6 ms
root@at91sam9263ek:~$ _
```

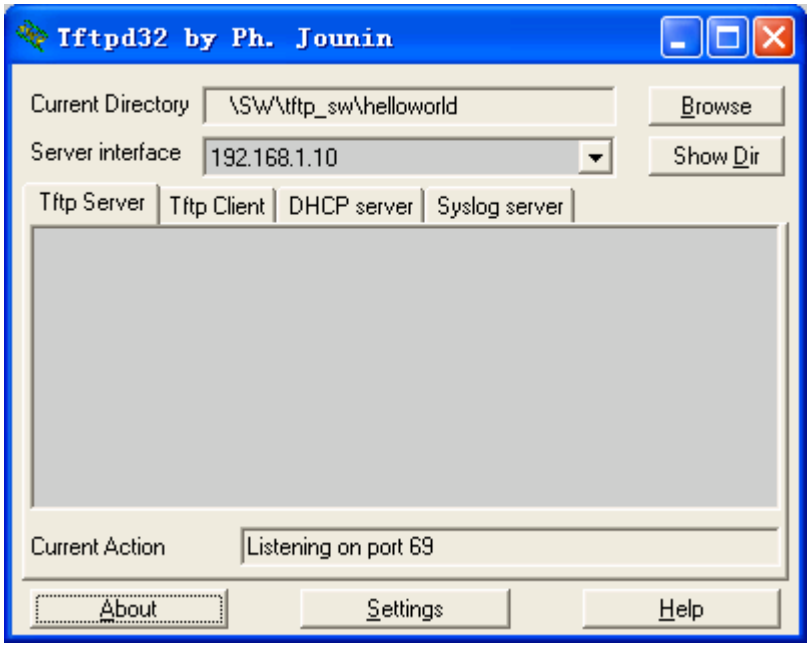
在主机端 ping 开发板：

```
$ ping 192.168.1.100

Pinging 192.168.1.100 with 32 bytes of data:

Reply from 192.168.1.100: bytes=32 time<1ms TTL=64
Reply from 192.168.1.100: bytes=32 time<1ms TTL=64
Reply from 192.168.1.100: bytes=32 time<1ms TTL=64
Reply from 192.168.1.100: bytes=32 time<1ms TTL=64

Ping statistics for 192.168.1.100:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms
```

网络畅通。

将 hello 放到 tftpd32 所在目录：

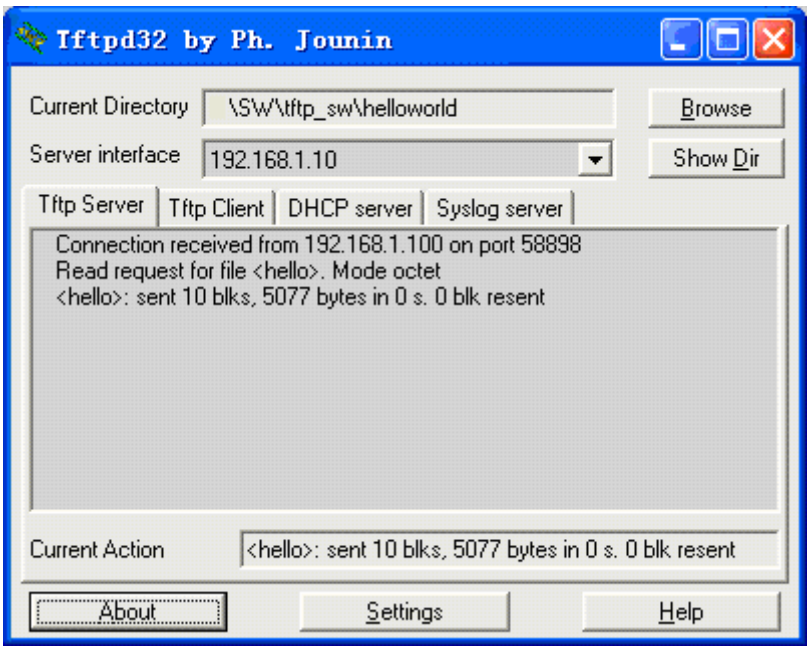
运行 tftpd32：



在开发板上运行 tftp：

```
root@at91sam9263ek:~$ pwd
/home/root
root@at91sam9263ek:~$ tftp -g -r hello 192.168.1.10
root@at91sam9263ek:~$ ls -al hello
-rw-r--r--    1 root     root          5077 Aug 24 17:15 hello
root@at91sam9263ek:~$
```

文件已经下载到了板子上，同时看看 tftpd32 的输出：



这里给出了一些信息。

在开发板上运行 hello，注意文件属性：

```
root@at91sam9263ek:~$ pwd
/home/root
root@at91sam9263ek:~$ tftp -g -r hello 192.168.1.10
root@at91sam9263ek:~$ ls -al hello
-rw-r--r--    1 root     root          5077 Aug 24 17:15 hello
root@at91sam9263ek:~$ ./hello
-sh: ./hello: Permission denied
root@at91sam9263ek:~$ id
uid=0(root) gid=0(root)
root@at91sam9263ek:~$ chmod 777 hello
root@at91sam9263ek:~$ ls -al hello
-rwxrwxrwx    1 root     root          5077 Aug 24 17:15 hello
root@at91sam9263ek:~$ ./hello
hello world!
root@at91sam9263ek:~$ _
```

tftp 的用法如下：

接收文件:tftp –g –r remote_file_name server_ip

上传文件:tftp –p –l local_file_name server_ip

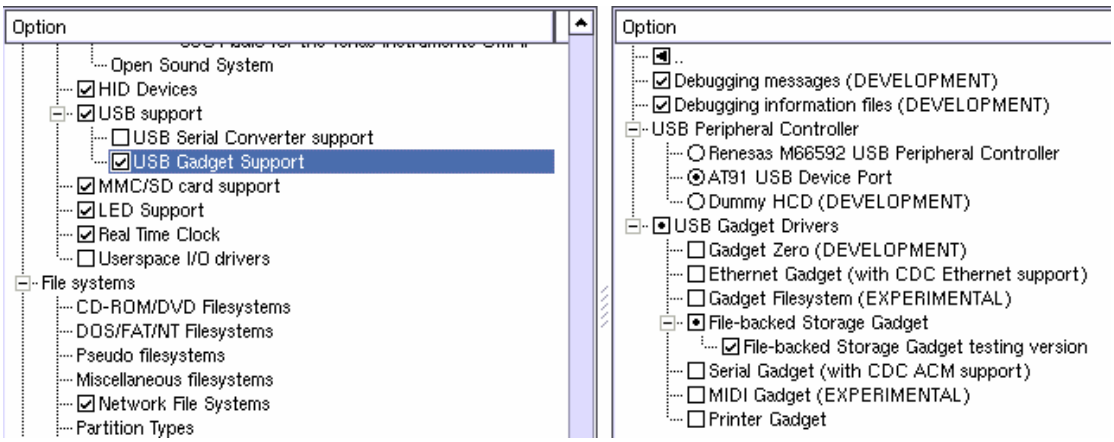
例如:

```
root@at91sam9263ek:/var/volatile/log$ tftp -p -l messages 192.168.1.10
```

1. 将板子作为 U 盘

9261 有一个 USB Device 接口，可以使用此接口，将开发板作为一个 USB Mass Storge 的设置，使得 PC 将开发板识别为一个 U 盘，在 PC 上可以将文件传输到这个 U 盘上，然后在 linux 下 mount 这个设备。

首先需要在 linux 下配置 USB Gadget 设备：



编译完成后，将在 linux 的\drivers\usb\gadget 下生成模块 g_file_storage.ko。将此文件传输到开发板：

```
root@at91sam9263ek:/lib/modules/2.6.20$ ls -al g_file_storage.ko
-rwxr-xr-x    1 root     root        38977 Aug 24 17:07 g_file_storage.ko
root@at91sam9263ek:/lib/modules/2.6.20$ _
```

将 NAND 的一部分作为 Mass Storge Device 的存储空间，根据板子对 NAND 的划分：

```
root@at91sam9263ek:/lib/modules/2.6.20$ cat /proc/mtd
dev:    size   erasesize  name
mtd0: 04000000 00020000 "Partition 1"
mtd1: 04000000 00020000 "Partition 2"
mtd2: 00420000 00000210 "spi0.0-AT45DB321x"
root@at91sam9263ek:/lib/modules/2.6.20$
```

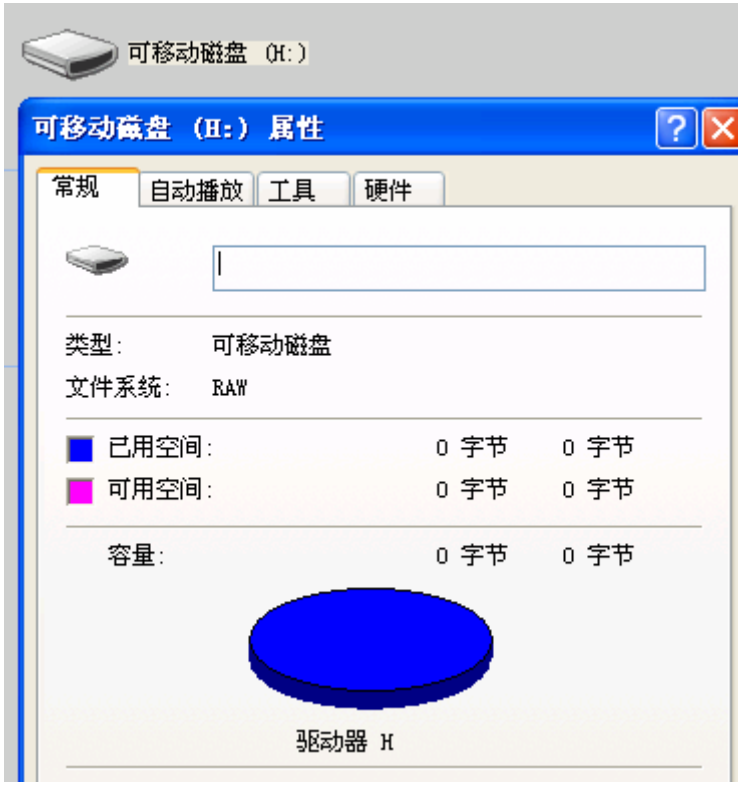
由于 MTD0 被 rootfs 占用，因此使用 MTD1，运行下面命令加载模块：

```
root@at91sam9263ek:/lib/modules/2.6.20$ insmod g_file_storage.ko file=/dev/mtdbl
ock1 stall=0 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: /dev/mtdblock1
root@at91sam9263ek:/lib/modules/2.6.20$ _
```

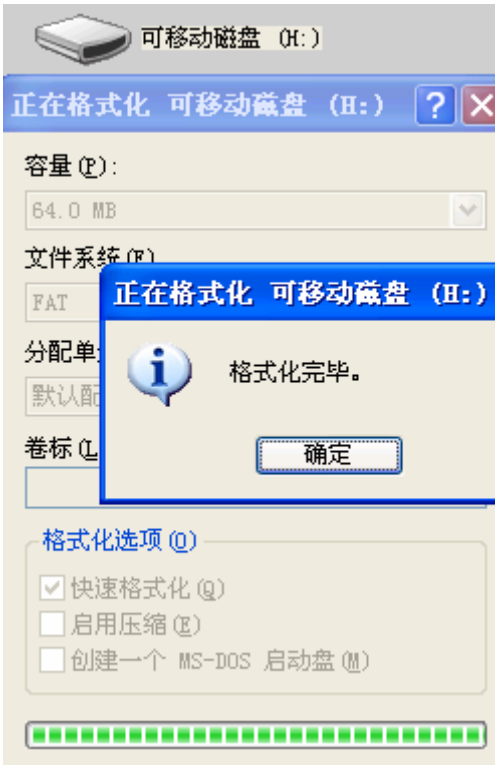
加载完成后，接上 USB Device 插座的连接线到 PC。开发板将输出如下信息：

```
root@at91sam9263ek:/lib/modules/2.6.20$ g_file_storage gadget: full speed config
 #1
```

PC 端将识别出一个移动磁盘：



但此时还不能使用，需要格式化：



完成后再检查一次属性：

设备已经可用使用。

在 PC 上将 hello 放到该盘符下，文件传输完成后在 PC 上将移动磁盘弹出。

回到开发板命令行，开始 mount mtd1 到文件系统：

```
root@at91sam9263ek:/mnt$ mkdir MSD
root@at91sam9263ek:/mnt$ mount -t vfat /dev/mtdblock1 /mnt/MSD/
root@at91sam9263ek:/mnt$ cd MSD/
root@at91sam9263ek:/mnt/MSD$ ls -al hello
-rwxr-xr-x    1 root     root          5077 Aug 24 13:41 hello
root@at91sam9263ek:/mnt/MSD$
```

Mount 完成后，文件系统可用，信息如下：

```
root@at91sam9263ek:/mnt/MSD$ df -k
Filesystem           1k-blocks      Used Available Use% Mounted on
/dev/mtdblock0           65536     28648     36888  44% /
tmpfs                       40         0        40   0% /mnt/.psplash
/dev/mtdblock0           65536     28648     36888  44% /dev/.static/dev
tmpfs                     2048        44      2004   2% /dev
tmpfs                    30908        80     30828   0% /var/volatile
tmpfs                    30908         0     30908   0% /media/ram
/dev/mtdblock1           65264     43374     21890  66% /mnt/MSD
root@at91sam9263ek:/mnt/MSD$
```

同样开始运行：

```
root@at91sam9263ek:/mnt/MSD$ ./hello
hello world!
root@at91sam9263ek:/mnt/MSD$ _
```

四，一些事项

本文以一个非常简单的例子介绍了 Linux 下 APP 的开发，运行的过程，主要侧重在以 Windows PC 为开发机的环境。

如果编译遇到问题可以先检查一下环境，然后就是工具链。

请按照上文描述的步骤操作。

第十章：SAM9261 上的 Linux 初步应用 3

本文叙述 9261 Linux 下基于 Boa 的 web server 应用。包括了 boa 的编译，配置运行以及 web server 网页测试(静态页面与 CGI)。开发使用的 Linux 版本选择了 2.6.24。本文中的开发基于 Linux 主机，包括了 linux 主机的 tftp 和 NFS 配置，以及开发板上的网络配置。在文章最后给出了 linux4sam 网站所提供的 linux image 的使用方法。开发环境选择了 Virtual PC，其它软件可以类推。

一，准备工作

1. 使用 Virtual PC 安装 Linux 虚拟机
   请参照本站另一篇文档**《基于 VPC 建立 ARM_Linux 开发环境》**。

2. 下载相关软件包
   1) Boa 源代码
   boa-0.94.13.tar.gz
   2) 工具链
   arm-2008q1-126-arm-none-linux-gnueabi-i686-pc-linux-gnu.tar.bz2
   将工具链安装到/opt/codesourcery/，并将其路径添加到系统路径。

```
[root@Linux4SAM work]# which arm-none-linux-gnueabi-gcc
/opt/codesourcery/bin/arm-none-linux-gnueabi-gcc
[root@Linux4SAM work]#
```

二，编译 Boa

首先使用展开源码包：

```
[root@Linux4SAM app]# tar xzvf boa-0.94.13.tar.gz
```

```
[root@Linux4SAM app]# cd boa-0.94.13
[root@Linux4SAM boa-0.94.13]# ls
boa.conf  ChangeLog  contrib  CREDITS  docs  examples  extras  Gnu_License  README  src
[root@Linux4SAM boa-0.94.13]#
```

1. 修改 Makefile

使用下面命令生成 Makefile：

```
[root@Linux4SAM boa-0.94.13]# cd src
[root@Linux4SAM src]# ./configure
```

完成后可以看到生成的 Makefile：

```
[root@Linux4SAM src]# ls Makefile
Makefile
[root@Linux4SAM src]#
```

针对 ARM 环境编译，需要修改 Makefile：
找到 CC 和 CPP：

```
21 srcdir = .
22 VPATH = .:./../extras
23 LDFLAGS =  -g
24 LIBS =
25 CFLAGS = -g -O2 -pipe -Wall -I.
26
27 # Change these if necessary
28
29 YACC = bison -y
30 LEX = flex
31 CC = gcc
32 CPP = gcc -E
33
34 SOURCES = alias.c boa.c buffer.c cgi.c cgi_header.c config.c escape.c \
35         get.c hash.c ip.c log.c mmap_cache.c pipe.c queue.c read.c \
36         request.c response.c select.c signals.c util.c sublog.c
37
38 OBJS = y.tab.o lex.yy.o $(SOURCES:.c=.o) timestamp.o
39
40 all:    boa boa_indexer
41
42 boa:    $(OBJS)
43         $(CC) -o $@ $^ $(LDFLAGS) $(LIBS)
44
45 boa_indexer:    index_dir.o escape.o
46         $(CC) -o $@ $^ $(LDFLAGS) $(LIBS)
```

根据前面所安装的工具链，将其修改为：

```
31 CC = arm-none-linux-gnueabi-gcc -mcpu=arm926ej-s
32 CPP = arm-none-linux-gnueabi-gcc -mcpu=arm926ej-s -E
33
```

2. 编译代码

输入 make 即可开始编译：

```
[root@Linux4SAM src]# make
```

编译开始后不久就会遭遇一个错误：

```
arm-none-linux-gnueabi-gcc -mcpu=arm926ej-s  -g -O2 -pipe -Wall -I.   -c -o response.o response.c
arm-none-linux-gnueabi-gcc -mcpu=arm926ej-s  -g -O2 -pipe -Wall -I.   -c -o select.o select.c
arm-none-linux-gnueabi-gcc -mcpu=arm926ej-s  -g -O2 -pipe -Wall -I.   -c -o signals.o signals.c
arm-none-linux-gnueabi-gcc -mcpu=arm926ej-s  -g -O2 -pipe -Wall -I.   -c -o util.o util.c
util.c:100:1: error: pasting "t" and "->" does not give a valid preprocessing token
make: *** [util.o] Error 1
[root@Linux4SAM src]#
```

打开文件看看什么问题：

```
90 char *get_commonlog_time(void)
91 {
92     struct tm *t;
93     char *p;
94     unsigned int a;
95     static char buf[30];
96     int time_offset;
97
98     if (use_localtime) {
99         t = localtime(&current_time);
100        time_offset = TIMEZONE_OFFSET(t);
101    } else {
102        t = gmtime(&current_time);
103        time_offset = 0;
104    }
105
```

从错误信息看，是编译器认为宏展开错误，可以将宏展开后带入。

搜索宏定义：

```
[root@Linux4SAM src]# grep -in TIMEZONE_OFFSET *
compat.h:120:#define TIMEZONE_OFFSET(foo) foo##->tm_gmtoff
compat.h:122:#define TIMEZONE_OFFSET(foo) timezone
util.c:100:         time_offset = TIMEZONE_OFFSET(t);
[root@Linux4SAM src]#
```

可以看到宏定义在 compat.h 内。根据其定义，直接修改 util.c 内的源代码:

```
 95      static char buf[30];
 96      int time_offset;
 97
 98      if (use_localtime) {
 99          t = localtime(&current_time);
100          //time_offset = TIMEZONE_OFFSET(t);
101          time_offset = t->tm_gmtoff;
102      } else {
103          t = gmtime(&current_time);
104          time_offset = 0;
105      }
```

保存后重新编译，完成后会生成 boa 可执行文件。

```
root@Linux4SAM:/usr/work/app/boa-0.94.13/src
File  Edit  View  Terminal  Go  Help
[root@Linux4SAM src]# ls -al boa
-rwxr-xr-x    1 root     root       195950 Aug 30 14:27 boa
[root@Linux4SAM src]# file boa
boa: ELF 32-bit LSB executable, ARM, version 1 (SYSV), for GNU/Linux 2.6.14, dynamically linked (uses shared libs), not stri
ped
[root@Linux4SAM src]#
```

三，运行 Boa

1. 配置 Boa

通过 nfs 可以将 boa 及相关文件传输到开发板。首先在 PC 上将相关文件传输到 NFS(NFS 服务的配置请看下一章)共享目录下：

```
[root@Linux4SAM src]# cp boa /nfs4arm/
[root@Linux4SAM src]# cd ..
[root@Linux4SAM boa-0.94.13]# ls
boa.conf  ChangeLog  contrib  CREDITS  docs  examples  extras  Gnu_License  README  src
[root@Linux4SAM boa-0.94.13]# cp boa.conf /nfs4arm/
[root@Linux4SAM boa-0.94.13]# cp /etc/mime.types /nfs4arm/
[root@Linux4SAM boa-0.94.13]#
```

在开发板上将相应文件分别放到不同目录：

```
root@at91sam9263ek:/mnt/nfs$ ls
boa                 file_from_client.txt  nfs_tst.txt
boa.conf            mime.types
root@at91sam9263ek:/mnt/nfs$ cp boa /bin/
root@at91sam9263ek:/mnt/nfs$ mkdir /etc/boa
root@at91sam9263ek:/mnt/nfs$ cp boa.conf /etc/boa/
root@at91sam9263ek:/mnt/nfs$ cp mime.types /etc/
root@at91sam9263ek:/mnt/nfs$
```

对 boa 的配置集中在文件/etc/boa/boa.conf 中:

Port：设置监听端口，默认 80 即可

ErrorLog 与 AccessLog 可以设置 log 的位置。

ServerName 选项需要打开。

Group 改为 nobody。

ScriptAlias 由/cgi-bin/ /usr/lib/cgi-bin/修改为/cgi-bin/ /var/www/cgi-bin/。

配置文件修改完成后保存，创建下列文件夹:

```
root@at91sam9263ek:~$ mkdir -m 777 /var/log/boa
root@at91sam9263ek:~$ mkdir -m 777 /var/www
root@at91sam9263ek:~$ mkdir -m 777 /var/www/cgi-bin
root@at91sam9263ek:~$
```

2. 静态网页测试

为了方便测试，可以直接找一个网页，另存为 html 文件，然后将其传输到开发板。

这里将 mcuzone 首页保存为 html 文件：

将文件通过 linux 虚拟机传输到开发板：

```
root's X desktop (Linux4SAM:2)
root@Linux4SAM:/usr/work/app/boa-0.94.13
File  Edit  View  Terminal  Go  Help
[root@Linux4SAM boa-0.94.13]# ls ../../trans/
index.files  index.html
[root@Linux4SAM boa-0.94.13]# mv ../../trans/* /nfs4arm/
[root@Linux4SAM boa-0.94.13]# ls ../../trans/
[root@Linux4SAM boa-0.94.13]# ls /nfs4arm/
boa  boa.conf  file_from_client.txt  index.files  index.html  mime.types  nfs_tst.txt
[root@Linux4SAM boa-0.94.13]#
```

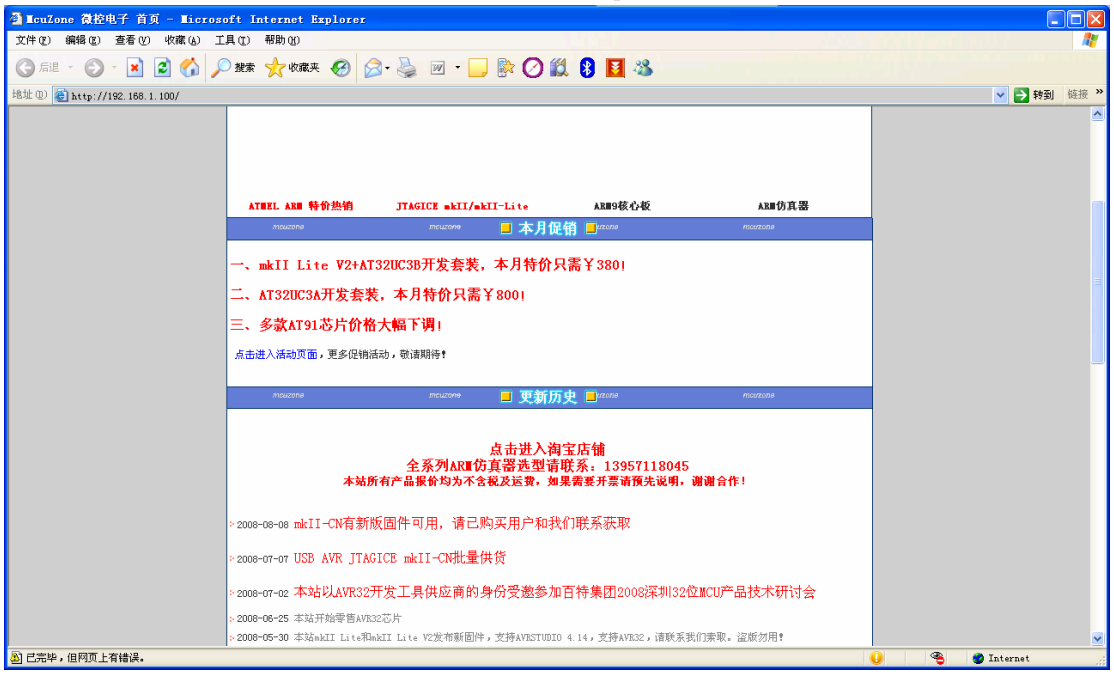
在开发板上继续操作：

```
root@at91sam9263ek:/mnt/nfs$ ls
boa                 index.files          nfs_tst.txt
boa.conf            index.html
file_from_client.txt  mime.types
root@at91sam9263ek:/mnt/nfs$ cp index.html /var/www/
root@at91sam9263ek:/mnt/nfs$ cp -R index.files/ /var/www/
root@at91sam9263ek:/mnt/nfs$ ls -al /var/www/
drwxrwxrwx    4 root     root             0 Aug 30 15:31 .
drwxr-xr-x    8 root     root             0 Aug 30 15:16 ..
drwxrwxrwx    2 root     root             0 Aug 30 15:16 cgi-bin
drwxr-xr-x    2 root     root             0 Aug 30 15:31 index.files
-rwxr--r--    1 root     root         22026 Aug 30 15:30 index.html
root@at91sam9263ek:/mnt/nfs$ _
```

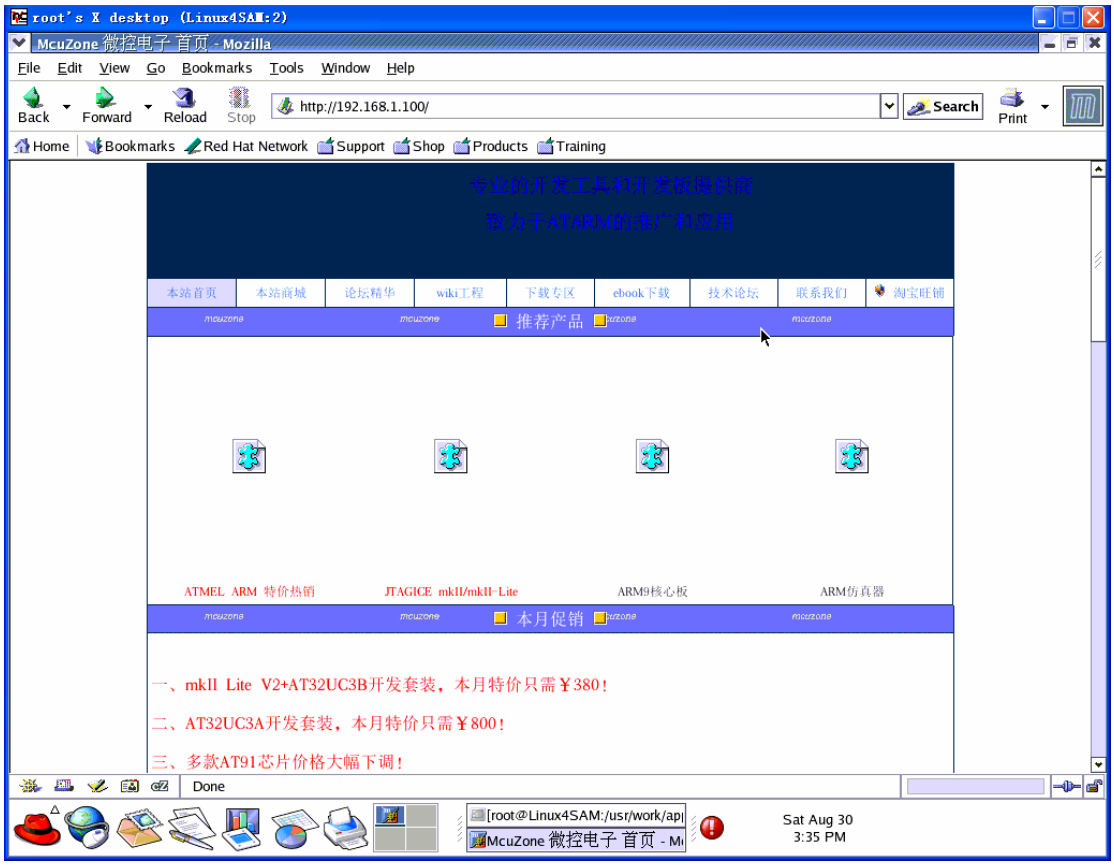
启动 Boa：

```
root@at91sam9263ek:/mnt/nfs$ which boa
/bin/boa
root@at91sam9263ek:/mnt/nfs$ boa
root@at91sam9263ek:/mnt/nfs$ _
```

在 PC 上打开浏览器，输入开发板 ip 地址：



在 linux 虚拟机上也可以浏览：



通过这样的步骤，boa 就已经在板子上运行起来，静态网页得到支持。

在开发板上可以通过下面的 log 来了解 boa 的工作情况：

```
root@at91sam9263ek:/var/volatile/log/boa$ ls -al
drwxrwxrwx    2 root     root            80 Aug 30 15:32 .
drwxr-xr-x    3 root     root           100 Aug 30 15:16 ..
-rw-------    1 root     root         13439 Aug 30 16:26 access_log
-rw-------    1 root     root          3042 Aug 30 16:12 error_log
root@at91sam9263ek:/var/volatile/log/boa$ _
```

3. CGI 测试

创建一个 index.html，内容如下：

index.html

```
1  <html>
2  <head>
3  <title>MCUZone</title>
4  </head>
5  <body>
6  <H1 ALIGN=CENTER>MCUZone: Boa CGI test</H1>
7  <A HREF="http://192.168.1.100/cgi-bin/getdate">Get Current Date from board</A>
8  </body>
9  </html>
```

将此文件传输到开发板：

```
root@at91sam9263ek:/var/www$ ls
cgi-bin
root@at91sam9263ek:/var/www$ cp /mnt/nfs/index.html ./
root@at91sam9263ek:/var/www$ cat index.html
<html>
<head>
<title>MCUZone</title>
</head>
<body>
<H1 ALIGN=CENTER>MCUZone: Boa CGI test</H1>
<A HREF="http://192.168.1.100/cgi-bin/getdate">Get Current Date from board</A>
</body>
</html>root@at91sam9263ek:/var/www$
```

在 cgi-bin 下创建一个 shell 脚本：

```
root@at91sam9263ek:/var/www/cgi-bin$ vi getdate.sh
```

```
#!/bin/sh
echo Content-type:text/plain
echo
/bin/date
~
```

```
root@at91sam9263ek:/var/www/cgi-bin$ ls -al
drwxrwxrwx    2 root     root             0 Aug 30 16:10 .
drwxrwxrwx    3 root     root             0 Aug 30 16:07 ..
-rw-r--r--    1 root     root            54 Aug 30 16:10 getdate.sh
root@at91sam9263ek:/var/www/cgi-bin$ chmod 777 getdate.sh
root@at91sam9263ek:/var/www/cgi-bin$ ls -al
drwxrwxrwx    2 root     root             0 Aug 30 16:10 .
drwxrwxrwx    3 root     root             0 Aug 30 16:07 ..
-rwxrwxrwx    1 root     root            54 Aug 30 16:10 getdate.sh
root@at91sam9263ek:/var/www/cgi-bin$ _
```

创建一个 softlink(也可以修改 index.html 里指明执行的文件)：

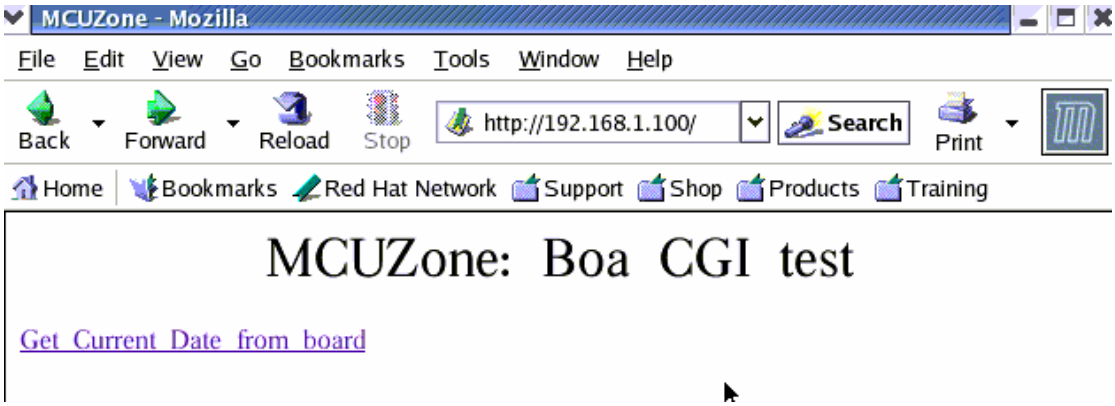
```
root@at91sam9263ek:/var/www/cgi-bin$ ln -s getdate.sh getdate
root@at91sam9263ek:/var/www/cgi-bin$ ls -al
drwxrwxrwx    2 root     root            0 Aug 30 16:15 .
drwxrwxrwx    3 root     root            0 Aug 30 16:07 ..
lrwxrwxrwx    1 root     root           10 Aug 30 16:15 getdate -> getdate.sh
-rwxrwxrwx    1 root     root           54 Aug 30 16:10 getdate.sh
root@at91sam9263ek:/var/www/cgi-bin$
```

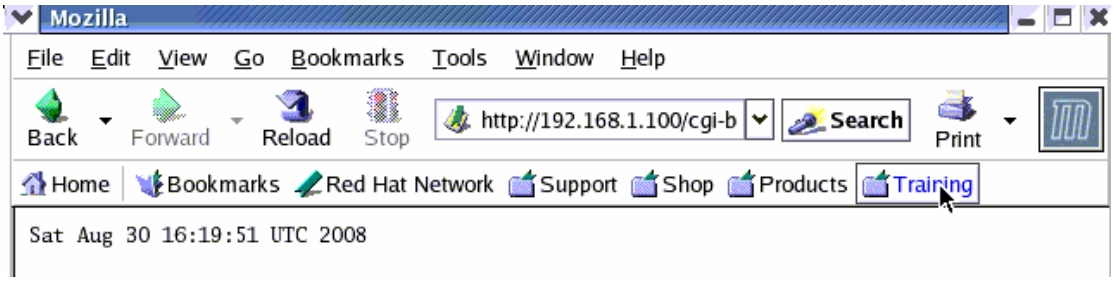
在 PC 浏览器上打开开发板 ip 地址：

点击进入链接：

对应的脚本执行，获得了板子上的时间。

再看看在 linux 虚拟机上的效果：





四，配置 Linux 网络

1. 配置 Linux 主机

为了方便与开发板通过网络传输文件，在 linux 虚拟机上可以尝试开启 tftp 与 NFS 服务。

1) 启用 tftp 服务

默认情况下 tftp 服务被禁用。通过浏览其配置文件即可发现：

```
[root@Linux4SAM root]# cat /etc/xinetd.d/tftp
# default: off
# description: The tftp server serves files using the trivial file transfer \
#       protocol.  The tftp protocol is often used to boot diskless \
#       workstations, download configuration files to network-aware printers, \
#       and to start the installation process for some operating systems.
service tftp
{
        socket_type             = dgram
        protocol                = udp
        wait                    = yes
        user                    = root
        server                  = /usr/sbin/in.tftpd
        server_args             = -s /tftpboot
        disable                 = yes
        per_source              = 11
        cps                     = 100 2
        flags                   = IPv4
}
[root@Linux4SAM root]#
```

将 disable 项修改为 no，tftp 服务的根文件夹可以可以通过 server_args 修改，参数-s 指定 chroot， -c 指定可以创建文件：

```
[root@Linux4SAM tftpboot]# cat /etc/xinetd.d/tftp
# default: off
# description: The tftp server serves files using the trivial file transfer \
#       protocol.  The tftp protocol is often used to boot diskless \
#       workstations, download configuration files to network-aware printers, \
#       and to start the installation process for some operating systems.
service tftp
{
        socket_type             = dgram
        protocol                = udp
        wait                    = yes
        user                    = root
        server                  = /usr/sbin/in.tftpd
        server_args             = -u nobody -c -s /tftpboot
        disable                 = no
        per_source              = 11
        cps                     = 100 2
        flags                   = IPv4
}
[root@Linux4SAM tftpboot]#
```

修改完成后使用下面命令启动 tftp 服务：

```
root@Linux4SAM:~
File  Edit  View  Terminal  Go  Help
[root@Linux4SAM root]# mkdir tftpboot
[root@Linux4SAM root]# chmod o+w /tftpboot/
[root@Linux4SAM root]# service xinetd restart
Stopping xinetd:                                           [  OK  ]
Starting xinetd:                                           [  OK  ]
[root@Linux4SAM root]#
```

```
root's X desktop (Linux4SAM:2)
root@Linux4SAM:/
File  Edit  View  Terminal  Go  Help
[root@Linux4SAM /]# chmod 777 -R tftpboot/
[root@Linux4SAM /]#
```

看看服务状态:

```
[root@Linux4SAM tftpboot]# netstat -nlp | grep 69
udp        0      0 0.0.0.0:69              0.0.0.0:*                           16570/xinetd
udp        0      0 0.0.0.0:69              0.0.0.0:*                           16432/in.tftpd
[root@Linux4SAM tftpboot]#
```

新建一个测试文件 tftp_pc.txt：

```
root@Linux4SAM:/
File  Edit  View  Terminal  Go  Help
[root@Linux4SAM /]# ls -l tftpboot/
total 8
-rw-r--r--    1 root     root           18 Aug 30 12:43 tftp_pc.txt
drwxr-xr-x    3 root     root         4096 May  9 06:43 X86PC
[root@Linux4SAM /]# chown -R nobody:nobody tftpboot/
[root@Linux4SAM /]# ls -l tftpboot/
total 8
-rw-r--r--    1 nobody   nobody         18 Aug 30 12:43 tftp_pc.txt
drwxr-xr-x    3 nobody   nobody       4096 May  9 06:43 X86PC
[root@Linux4SAM /]# cat tftpboot/tftp_pc.txt
this is a tftp pc
[root@Linux4SAM /]#
```

在开发板上测试一下：

```
root@at91sam9263ek:~$ ls
mnt        test.txt
root@at91sam9263ek:~$ tftp -g -r tftp_pc.txt 192.168.1.5
root@at91sam9263ek:~$ ls tftp_pc.txt
tftp_pc.txt
root@at91sam9263ek:~$ cat tftp_pc.txt
this is a tftp pc
root@at91sam9263ek:~$
```

2) 启用 NFS 服务

编辑/etc 下的文件 exports：

```
root@Linux4SAM:/tftpboot
File  Edit  View  Terminal  Go  Help
[root@Linux4SAM tftpboot]# ls -al /etc/exports
-rw-r--r--    1 root     root             0 Jan 13  2000 /etc/exports
[root@Linux4SAM tftpboot]# vi /etc/exports
```

```
root@Linux4SAM:~/nfs4arm
File  Edit  View  Terminal  Go  Help
[root@Linux4SAM nfs4arm]# cat /etc/exports
/nfs4arm 192.168.1.*(rw,sync,no_root_squash)
[root@Linux4SAM nfs4arm]# exportfs -r
```

/nfs4arm 也称为服务器输出共享目录。

括号内的参数意义描述如下：

rw：读/写权限，只读权限的参数为 ro；

sync：数据同步写入内存和硬盘，也可以使用 async，此时数据会先暂存于内存中，而不立即写入硬盘。

no_root_squash：NFS 服务器共享目录用户的属性，如果用户是 root，那么对于这个共享目录来说就具有 root 的权限。

修改保存后即可启动 nfs 服务：

```
root's X desktop (Linux4SAM:2)
root@Linux4SAM:/mnt/nfs_t
File  Edit  View  Terminal  Go  Help
[root@Linux4SAM nfs_t]# service portmap start
Starting portmapper:                                       [  OK  ]
[root@Linux4SAM nfs_t]# service nfs start
Starting NFS services:                                     [  OK  ]
Starting NFS quotas:                                       [  OK  ]
Starting NFS daemon:                                       [  OK  ]
Starting NFS mountd:                                       [  OK  ]
[root@Linux4SAM nfs_t]#
```

在 linux 虚拟机上先做一个本地验证。将 nfs 所在文件夹 mount 到本机：

```
root's X desktop (Linux4SAM:2)
root@Linux4SAM:/mnt/nfs_t
File  Edit  View  Terminal  Go  Help
[root@Linux4SAM /]# mkdir nfs4arm
[root@Linux4SAM /]# chown -R nobody:nobody nfs4arm/
[root@Linux4SAM /]# cd nfs4arm/
[root@Linux4SAM nfs4arm]# vi nfs_tst.txt
[root@Linux4SAM nfs4arm]# cat nfs_tst.txt
test file nfs
[root@Linux4SAM nfs4arm]# exportfs -r
[root@Linux4SAM nfs4arm]# cat /var/lib/nfs/xtab
[root@Linux4SAM nfs4arm]# mount -t nfs 192.168.1.5:/nfs4arm /mnt/nfs_t
[root@Linux4SAM nfs4arm]# cd /mnt/nfs_t/
[root@Linux4SAM nfs_t]# ls -al
total 12
drwxr-xr-x    2 nobody   nobody       4096 Aug 30 13:52 .
drwxr-xr-x    5 root     root         4096 Aug 30 13:35 ..
-rw-r--r--    1 root     root           14 Aug 30 13:52 nfs_tst.txt
[root@Linux4SAM nfs_t]# cat nfs_tst.txt
test file nfs
[root@Linux4SAM nfs_t]#
```

可以看到成功的 mount 到了主机文件系统。下面再到开发板上测试：

```
root@at91sam9263ek:~$ cd /mnt/
root@at91sam9263ek:/mnt$ mkdir nfs
root@at91sam9263ek:/mnt$ mount -t nfs 192.168.1.5:/nfs4arm /mnt/nfs
root@at91sam9263ek:/mnt$ cd nfs/
root@at91sam9263ek:/mnt/nfs$ ls -al
drwxr-xr-x    2 99       99           4096 Aug 30 05:52 .
drwxr-xr-x    6 root     root            0 Aug 30 13:56 ..
-rw-r--r--    1 root     root           14 Aug 30 05:52 nfs_tst.txt
root@at91sam9263ek:/mnt/nfs$ cat nfs_tst.txt
test file nfs
root@at91sam9263ek:/mnt/nfs$ _
```

Mount 情况:

```
root@at91sam9263ek:/mnt/nfs$ df -k
Filesystem           1k-blocks      Used Available Use% Mounted on
/dev/mtdblock0           65536     28700     36836  44% /
tmpfs                       40         0        40   0% /mnt/.psplash
/dev/mtdblock0           65536     28700     36836  44% /dev/.static/dev
tmpfs                     2048        44      2004   2% /dev
tmpfs                    30612        80     30532   0% /var/volatile
tmpfs                    30612         0     30612   0% /media/ram
/dev/mtdblock1           65264     43530     21734  67% /mnt/MSD
192.168.1.5:/nfs4arm  15377856   8009928   6586768  55% /mnt/nfs
root@at91sam9263ek:/mnt/nfs$
```

在开发板上创建一个文件:

```
root@at91sam9263ek:/mnt/nfs$ ls -al
drwxr-xr-x    2 99       99           4096 Aug 30 05:58 .
drwxr-xr-x    6 root     root            0 Aug 30 13:56 ..
-rw-r--r--    1 root     root           29 Aug 30 05:58 file_from_client.txt
-rw-r--r--    1 root     root           14 Aug 30 05:52 nfs_tst.txt
root@at91sam9263ek:/mnt/nfs$ cat file_from_client.txt
This file created on client.
```

在 PC 上浏览:

```
root's X desktop (Linux4SAM:2)
root@Linux4SAM:/nfs4arm
File  Edit  View  Terminal  Go  Help
[root@Linux4SAM nfs4arm]# ls -al
total 16
drwxr-xr-x    2 nobody   nobody       4096 Aug 30 13:58 .
drwxr-xr-x   22 root     root         4096 Aug 30 13:51 ..
-rw-r--r--    1 root     root           29 Aug 30 13:58 file_from_client.txt
-rw-r--r--    1 root     root           14 Aug 30 13:52 nfs_tst.txt
[root@Linux4SAM nfs4arm]# cat file_from_client.txt
This file created on client.
[root@Linux4SAM nfs4arm]#
```

Linux 虚拟机上的共享情况:

```
root's X desktop (Linux4SAM:2)
root@Linux4SAM:/nfs4arm
File  Edit  View  Terminal  Go  Help
[root@Linux4SAM nfs4arm]# ls -al
total 16
drwxr-xr-x    2 nobody   nobody       4096 Aug 30 13:58 .
drwxr-xr-x   22 root     root         4096 Aug 30 13:51 ..
-rw-r--r--    1 root     root           29 Aug 30 13:58 file_from_client.txt
-rw-r--r--    1 root     root           14 Aug 30 13:52 nfs_tst.txt
[root@Linux4SAM nfs4arm]# cat file_from_client.txt
This file created on client.
[root@Linux4SAM nfs4arm]# cat /var/lib/nfs/xtab
/nfs4arm        192.168.1.100(rw,sync,wdelay,hide,secure,no_root_squash,no_all_squash,subtree_check,secure_locks,mapping=id
tity,anonuid=-2,anongid=-2)
[root@Linux4SAM nfs4arm]#
```

2. 配置 linux 开发板

将开发板与路由器相连，启动 linux。设置开发板的 ip 地址：

```
root@at91sam9263ek:/etc/init.d$ ifconfig eth0 192.168.1.100
root@at91sam9263ek:/etc/init.d$ ifconfig -a
eth0      Link encap:Ethernet  HWaddr 3A:1F:34:08:54:54
          inet addr:192.168.1.100  Bcast:192.168.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:237 errors:0 dropped:0 overruns:0 frame:0
          TX packets:237 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:29412 (28.7 KiB)  TX bytes:27401 (26.7 KiB)
          Interrupt:107 Base address:0x6000
```

假定系统内的 ip 地址为：路由器 192.168.1.1；linux 虚拟机为 192.168.1.5；开发板为 192.168.1.100。

在板子和 pc 上通过 ping 的方式来测试网络连接：

```
root@at91sam9263ek:/etc/init.d$ ping 192.168.1.5
PING 192.168.1.5 (192.168.1.5): 56 data bytes
64 bytes from 192.168.1.5: icmp_seq=0 ttl=64 time=0.8 ms
64 bytes from 192.168.1.5: icmp_seq=1 ttl=64 time=0.7 ms
64 bytes from 192.168.1.5: icmp_seq=2 ttl=64 time=0.9 ms
64 bytes from 192.168.1.5: icmp_seq=3 ttl=64 time=1.9 ms
^C
--- 192.168.1.5 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 0.7/1.0/1.9 ms
root@at91sam9263ek:/etc/init.d$
```

```
root's X desktop (Linux4SAM:2)
root@Linux4SAM:~
File  Edit  View  Terminal  Go  Help
[root@Linux4SAM root]# ping 192.168.1.100
PING 192.168.1.100 (192.168.1.100) 56(84) bytes of data.
64 bytes from 192.168.1.100: icmp_seq=1 ttl=64 time=3.23 ms
64 bytes from 192.168.1.100: icmp_seq=2 ttl=64 time=0.622 ms
64 bytes from 192.168.1.100: icmp_seq=3 ttl=64 time=0.566 ms
64 bytes from 192.168.1.100: icmp_seq=4 ttl=64 time=0.569 ms
64 bytes from 192.168.1.100: icmp_seq=5 ttl=64 time=0.521 ms

--- 192.168.1.100 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4036ms
rtt min/avg/max/mdev = 0.521/1.103/3.237/1.067 ms
[root@Linux4SAM root]#
```

设置板子的路由：

```
root@at91sam9263ek:~$ route add default gw 192.168.1.1
root@at91sam9263ek:~$ route
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.1.0     *               255.255.255.0   U     0      0        0 eth0
default         192.168.1.1     0.0.0.0         UG    0      0        0 eth0
root@at91sam9263ek:~$
```

测试 ping 外网：

```
root@at91sam9263ek:~$ ping www.mcuzone.com
ping: www.mcuzone.com: Host name lookup failure
root@at91sam9263ek:~$ ping 220.189.255.38
PING 220.189.255.38 (220.189.255.38): 56 data bytes
64 bytes from 220.189.255.38: icmp_seq=0 ttl=120 time=33.5 ms
64 bytes from 220.189.255.38: icmp_seq=1 ttl=120 time=31.1 ms
64 bytes from 220.189.255.38: icmp_seq=2 ttl=120 time=31.9 ms
64 bytes from 220.189.255.38: icmp_seq=3 ttl=120 time=30.7 ms
^
--- 220.189.255.38 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 30.7/31.8/33.5 ms
root@at91sam9263ek:~$ _
```

可以发现直接使用域名无效，这就需要设置 nameserver：

在/etc 文件夹下创建文件 resolv.conf,其内容如下图，保存后即可。

```
root@at91sam9263ek:/etc$ cat resolv.conf
nameserver 192.168.1.1
root@at91sam9263ek:/etc$ ping www.mcuzone.com
PING www.mcuzone.com (220.189.255.38): 56 data bytes
64 bytes from 220.189.255.38: icmp_seq=0 ttl=120 time=231.5 ms
^
--- www.mcuzone.com ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 231.5/231.5/231.5 ms
root@at91sam9263ek:/etc$ ping google.com
PING google.com (64.233.167.99): 56 data bytes
64 bytes from 64.233.167.99: icmp_seq=0 ttl=237 time=510.9 ms
^
--- google.com ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 510.9/510.9/510.9 ms
root@at91sam9263ek:/etc$ ping google.com.cn
PING google.com.cn (209.85.165.99): 56 data bytes
64 bytes from 209.85.165.99: icmp_seq=0 ttl=238 time=328.5 ms
^
--- google.com.cn ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 328.5/328.5/328.5 ms
root@at91sam9263ek:/etc$
```

再次测试，即可看到域名可被解析。

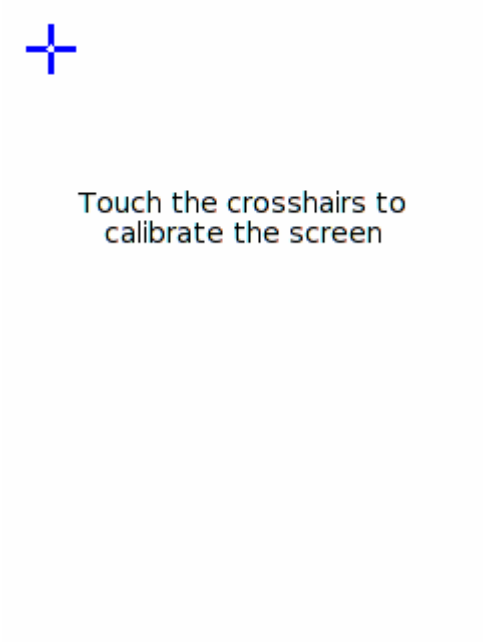
五，使用 linux4sam 提供的 linux image

1. 烧写 linux image

下载了 linux image 的整个压缩包后，将其烧写到板子上。压缩包里有 bat 文件，在板子连接上 sam-ba 之后，可以直接点击该 bat 文件完成所有的烧写工作。

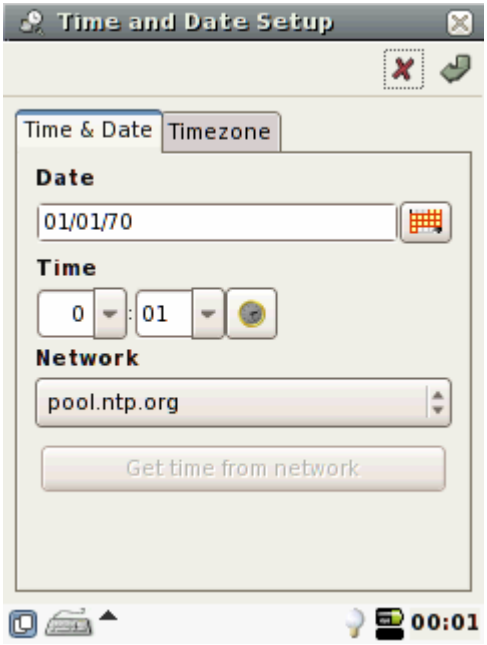
2. 校准触摸屏

烧写完成后第一次开机会要求校准触摸屏，按照屏幕提示，点击显示的”+”即可完成。



3. 设置系统时间

登陆完成后会被要求设置系统时间，可以在图形界面中设置：



也可以使用命令行设置时间，并将时间写入 rtc(系统将 SAM9 的 rtt 作为 RTC)：

```
root@at91sam9263ek:~$ date 083019382008
Sat Aug 30 19:38:00 UTC 2008
root@at91sam9263ek:~$ hwclock -w
root@at91sam9263ek:~$ hwclock -r
Sat Aug 30 19:38:10 2008  0.000000 seconds
root@at91sam9263ek:~$ _
```

完成后按复位键使得系统重启，从启动信息中可以看到系统时间将通过 rtc 同步，同时系统也不会提示输入时间。

```
NET: Registered protocol family 17
RPC: Registered udp transport module.
RPC: Registered tcp transport module.
rtc-at91sam9 at91_rtt.0: setting system clock to 2008-08-30 19:39:10 UTC (122012
5150)
```

登陆完成后的系统桌面如下图：

4. 配置网络

首先连接上网线到路由器。

浏览当前的网络配置：

```
root@at91sam9263ek:~$ ifconfig -a
eth0      Link encap:Ethernet  HWaddr 3A:1F:34:08:54:54
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
          Interrupt:21 Base address:0xc000
```

设置 ip 地址：

```
root@at91sam9263ek:~$ ifconfig eth0 192.168.1.100
root@at91sam9263ek:~$ eth0: link up (100/Full)

root@at91sam9263ek:~$
```

使用 ping 测试：

```
root@at91sam9263ek:~$ ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1): 56 data bytes
64 bytes from 192.168.1.1: icmp_seq=0 ttl=64 time=5.8 ms
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=0.7 ms
^
--- 192.168.1.1 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.7/3.2/5.8 ms
root@at91sam9263ek:~$ ping 192.168.1.5
PING 192.168.1.5 (192.168.1.5): 56 data bytes
64 bytes from 192.168.1.5: icmp_seq=0 ttl=64 time=17.3 ms
64 bytes from 192.168.1.5: icmp_seq=1 ttl=64 time=0.7 ms
64 bytes from 192.168.1.5: icmp_seq=2 ttl=64 time=0.8 ms
^
--- 192.168.1.5 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.7/6.2/17.3 ms
root@at91sam9263ek:~$
```

5. 使用 U 盘播放视频

在 PC 上复制一些视频和音频文件到 U 盘，将 U 盘连接到板子的 USB HOST 接口。系统会将 U 盘自动 mount 到/media/sda?(?=1,…):

```
root@at91sam9263ek:~$ usb 1-2: new full speed USB device using at91_ohci and add
ress 3
usb 1-2: configuration #1 chosen from 1 choice
scsi0 : SCSI emulation for USB Mass Storage devices
scsi 0:0:0:0: Direct-Access     USB NAND FLASH DISK       0.20 PQ: 0 ANSI: 2
sd 0:0:0:0: [sda] 128000 512-byte hardware sectors (66 MB)
sd 0:0:0:0: [sda] Write Protect is off
sd 0:0:0:0: [sda] Assuming drive cache: write through
sd 0:0:0:0: [sda] 128000 512-byte hardware sectors (66 MB)
sd 0:0:0:0: [sda] Write Protect is off
sd 0:0:0:0: [sda] Assuming drive cache: write through
 sda: sda1
sd 0:0:0:0: [sda] Attached SCSI removable disk
FAT: invalid media value (0xb9)
VFS: Can't find a valid FAT filesystem on dev sda.
FAT: invalid media value (0xb9)
VFS: Can't find a valid FAT filesystem on dev sda.
```

出现的错误并不影响使用：

```
:~$ cd /media/sda1/
:/media/sda1$ ls
          SHTxx_E_lo_0505_v.pdf
          V_trailer_320_reflect.avi
c         g_file_storage.ko
          hello
c         lcm_test
doc       mv1.asf
          usart_tst
          yx.mp3
:/media/sda1$ _
```

连接好耳机，使用下面命令播放 mp3：

```
root@at91sam9263ek:/media/sda1$ mplayer yx.mp3 _
```

播放过程中将输出相关的信息：

```
==========================================================================
Forced audio codec: mad
Opening audio decoder: [libmad] libmad mpeg audio decoder
AUDIO: 44100 Hz, 2 ch, s16le, 128.0 kbit/9.07% (ratio: 16000->176400)
Selected audio codec: [mad] afm: libmad (libMAD MPEG layer 1-2-3)
==========================================================================
AO: [oss] 44100Hz 2ch s16le (2 bytes per sample)
Video: no video
Starting playback...
A:   2.7 (02.7) of 386.0 (06:26.0) 19.3%
```

使用下面的命令播放视频：

```
root@at91sam9263ek:/media/sda1$ mplayer -vf rotate=2 -loop 0 MiDieXiang_JAY.mp4
```

参数 rotate 将旋转视频，以适应板子上的竖屏，loop 参数控制播放重复次数，为 0 则表示无限循环。

播放中终端上也有信息显示：

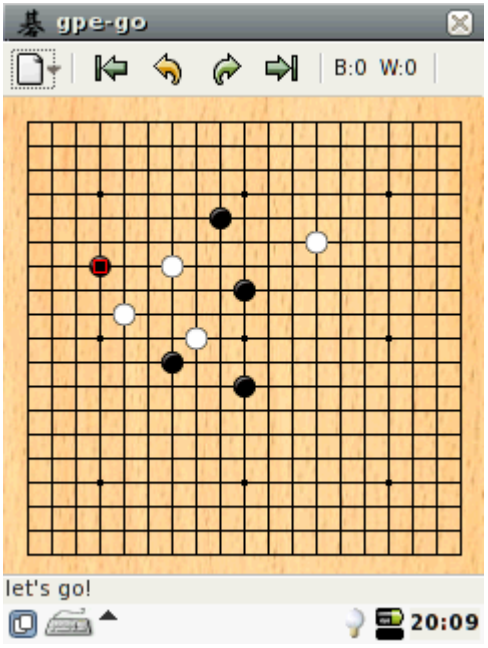
```
VO XOverlay need a subdriver
[VO_SDL] SDL initialization failed: No available video device.
Opening video filter: [rotate=2]
==========================================================================
Opening video decoder: [ffmpeg] FFmpeg's libavcodec codec family
Selected video codec: [ffodivx] vfm: ffmpeg (FFmpeg MPEG-4)
==========================================================================
==========================================================================
Forced audio codec: mad
Opening audio decoder: [faad] AAC (MPEG2/4 Advanced Audio Coding)
FAAD: compressed input bitrate missing, assuming 128kbit/s!
AUDIO: 22050 Hz, 2 ch, s16le, 128.0 kbit/18.14% (ratio: 16000->88200)
Selected audio codec: [faad] afm: faad (FAAD AAC (MPEG-2/MPEG-4 Audio) decoder)
==========================================================================
AO: [oss] 22050Hz 2ch s16le (2 bytes per sample)
Starting playback...
VDec: vo config request - 320 x 240 (preferred colorspace: Planar YV12)
Could not find matching colorspace - retrying with -vf scale...
Opening video filter: [scale]
New_Face failed. Maybe the font path is wrong.
Please supply the text font file (~/.mplayer/subfont.ttf).t.
subtitle font: load_sub_face failed.bgr555 special converter
A:  10.5 V:  10.4 A-V:  0.127 ct: -0.154 154/154 46% 17% 26.6% 38 0
```

LCD 显示(竖屏上的横向显示)：

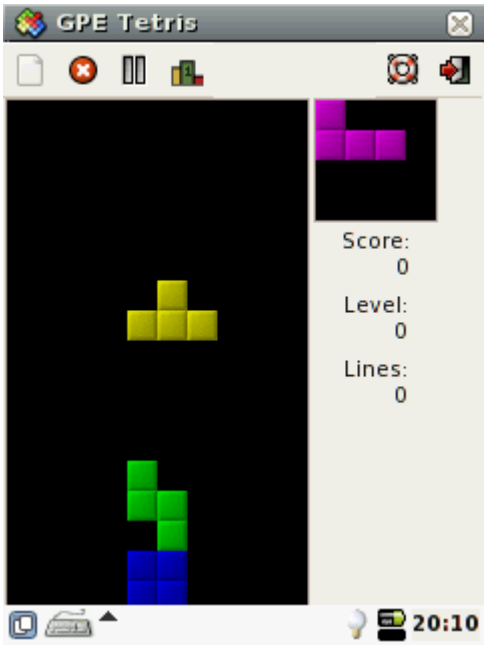
6. 游戏功能

在桌面上点击 Games 进入游戏菜单：

其中 GO 是一个棋类游戏,该游戏可以用来测试触摸屏：



而 Tetris 就是俄罗斯方块:

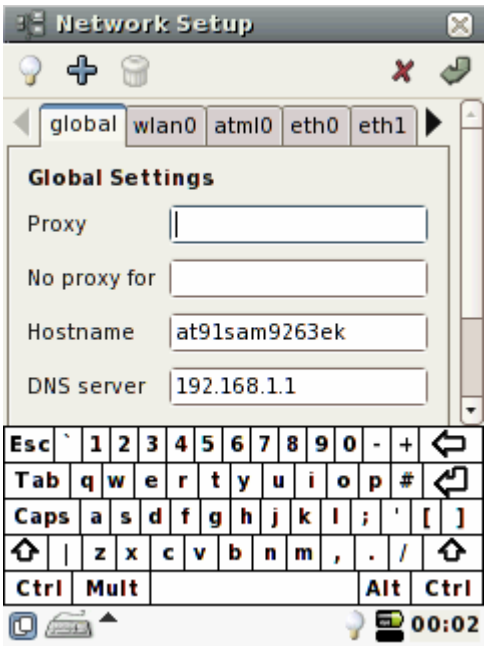


7. 系统设置

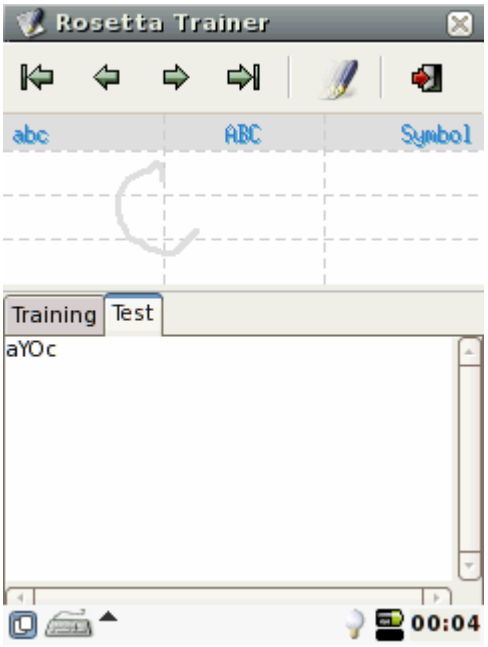
在设置菜单中可以对系统各方面进行配置


Settings
Owner Informa...
Package Manager
Panel Manager
PC/CF Cards
Rosetta Trainer
Screen Setup
Serial Ports
Sleep Config
Sound Settings
USB Settings
User Access
User Manager
20:24

例如网络配置:


Network Setup
global
wlan0
atml0
eth0
eth1
Global Settings
Proxy
No proxy for
Hostname
at91sam9263ek
DNS server
192.168.1.1
00:02

Rosetta Trainer，一个手写识别程序：



8. 一些应用

Other 菜单下包含了一些应用，比如计算器，文本编辑器等。

计算器：

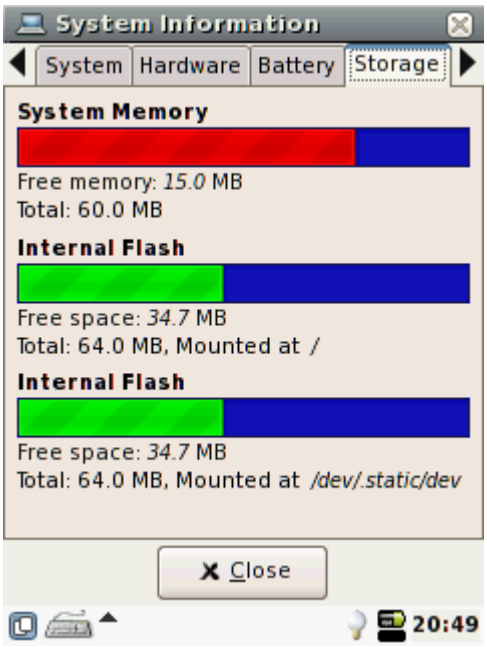

gcalculator v0.7
0.835673682896
1/x
x^2
SQRT
C
AC
INV
HYP
sin
cos
tan
e
EE
log
ln
x^y
PI
n!
(
)
/
MS
7
8
9
*
MR
4
5
6
-
M+
1
2
3
+
<-
0
.
+/-
=
welcome to gcalculator v0.7
20:26

文本编辑器：

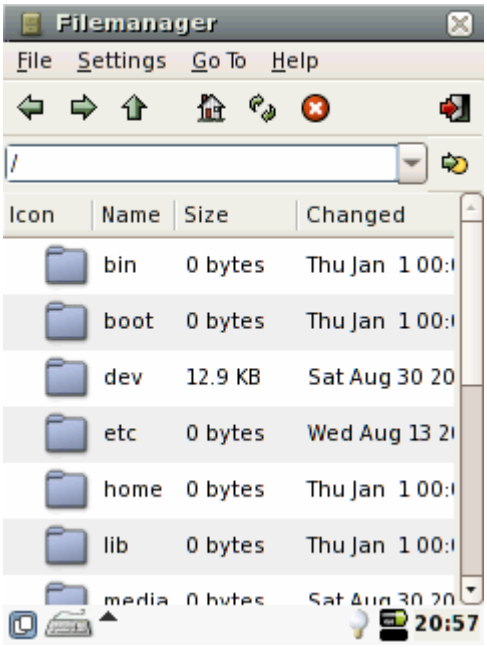

Editor - Untitled *
mcuzone
Esc ` 1 2 3 4 5 6 7 8 9 0 - +
Tab q w e r t y u i o p #
Caps a s d f g h j k l ; ' [ ]
| z x c v b n m , . /
Ctrl Mult Alt Ctrl
20:48

浏览系统信息：

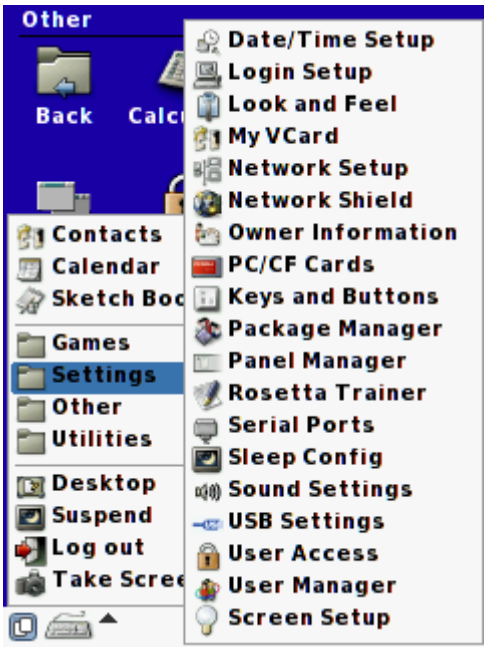

System Information
System
Hardware
Battery
Storage
Device information
Device Name
AT91SAM9263-EK
Manufacturer: ATMEL
RAM: 60 MB
Storage: 132 MB
Close
20:49


System Information
System
Hardware
Battery
Storage
System Memory
Free memory: 15.0 MB
Total: 60.0 MB
Internal Flash
Free space: 34.7 MB
Total: 64.0 MB, Mounted at /
Internal Flash
Free space: 34.7 MB
Total: 64.0 MB, Mounted at /dev/.static/dev
Close
20:49

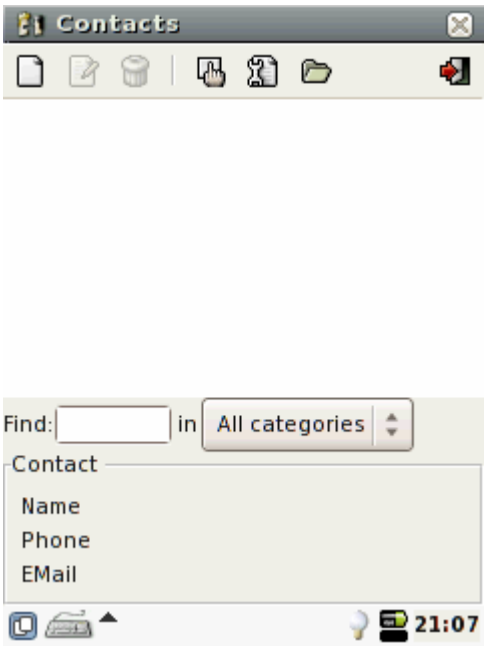
文件管理器：



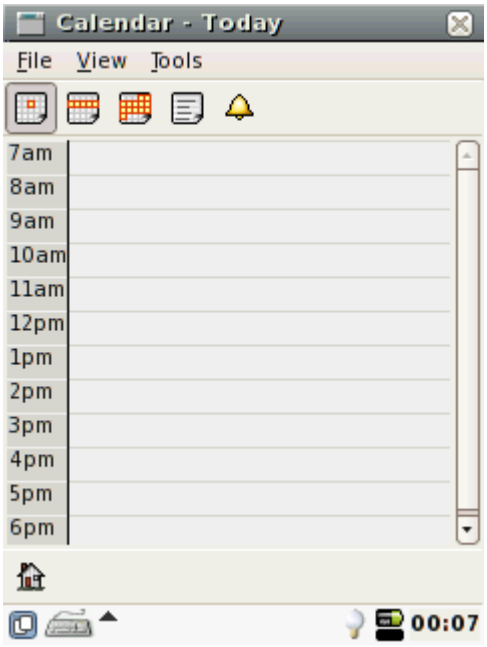
点击左下角图标将出现”开始菜单”:


Other
Back
Calc
Contacts
Calendar
Sketch Boc
Games
Settings
Other
Utilities
Desktop
Suspend
Log out
Take Scree
Date/Time Setup
Login Setup
Look and Feel
My VCard
Network Setup
Network Shield
Owner Information
PC/CF Cards
Keys and Buttons
Package Manager
Panel Manager
Rosetta Trainer
Serial Ports
Sleep Config
Sound Settings
USB Settings
User Access
User Manager
Screen Setup

联系人：


Contacts
Find:
in
All categories
Contact
Name
Phone
EMail
21:07

日程表:



六，一些事项

如果编译遇到问题可以先检查一下环境，然后就是工具链。
如果网络连接问题，请检查各 pc 上的防火墙设置。
请按照上文描述的步骤操作。

第十一章：SAM9261 上的 Linux 初步应用 4

基于前面的 linux 平台，下面我们演示一个 BT 下载应用，软件使用 transmissionbt。

首先复制 transmission 的_install 目录下的 web 到开发板目录下：

```
[root@mcuzone transmission]#pwd
/var/www/transmission
[root@mcuzone transmission]#ls
web
[root@mcuzone transmission]#ls web/
LICENSE      images       index.html   javascript   stylesheets
[root@mcuzone transmission]#
```

```
[root@mcuzone share]#pwd
/usr/share
[root@mcuzone share]#ls -al
drwxr-xr-x    1 root     root         2048 Sep 11  2008 .
drwxr-xr-x    1 root     root         2048 Sep 27  2008 ..
lrwxrwxrwx    1 root     root           22 Jan  1 00:02 transmission -> /var/www/transmission/
[root@mcuzone share]#
```

运行 tansimission-daemon：

```
[root@mcuzone temp]#transmission-daemon -f -T
Transmission 1.34 (6778) started
Searching for web interface file "/home/root/.local/share/transmission/web/javas
cript/transmission.js"
Searching for web interface file "/usr/local/share//transmission/web/javascript/
transmission.js"
Searching for web interface file "/usr/share//transmission/web/javascript/transm
ission.js"
Serving the web interface files from "/usr/share//transmission/web"
Loaded 2 torrents
Port Forwarding: Opened port 51413 to listen for incoming peer connections
```

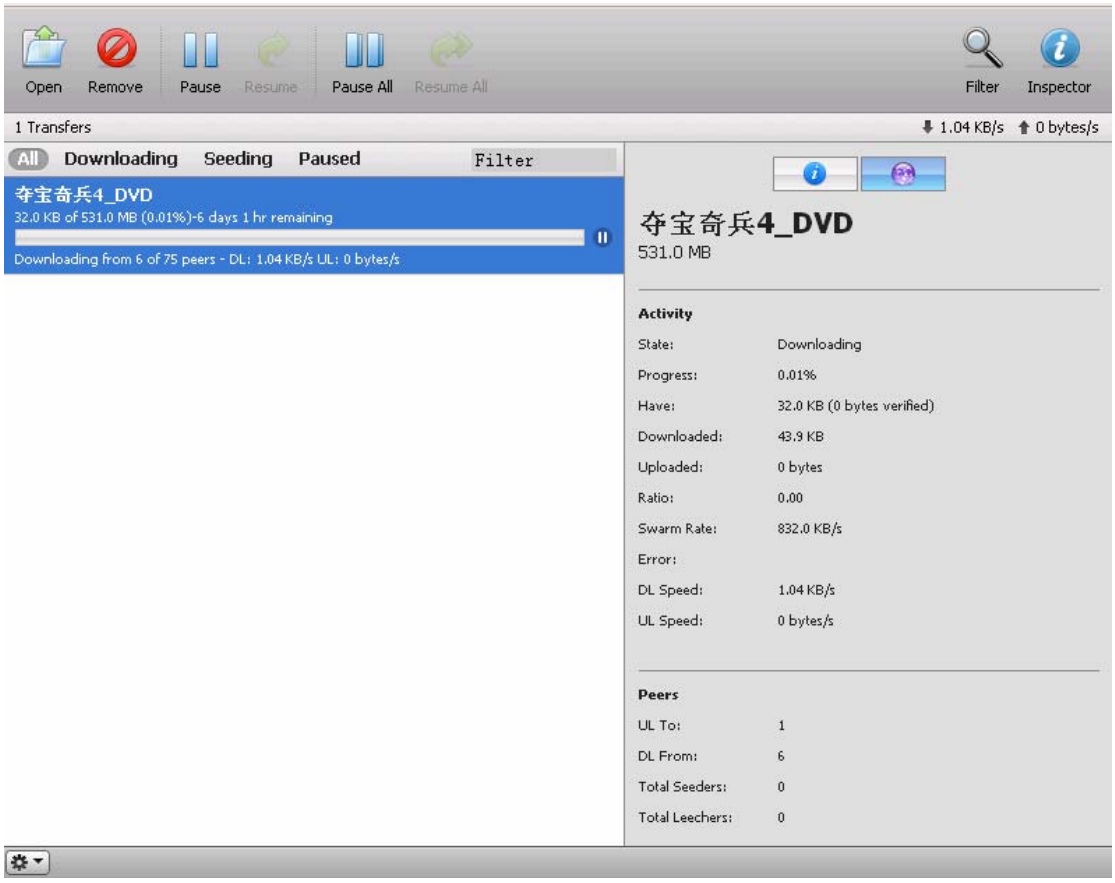
设置下载目录:

```
[root@mcuzone sbin]#transmission-remote -w /mnt/sda1/
localhost:9091 responded: "success"
[root@mcuzone sbin]#
```

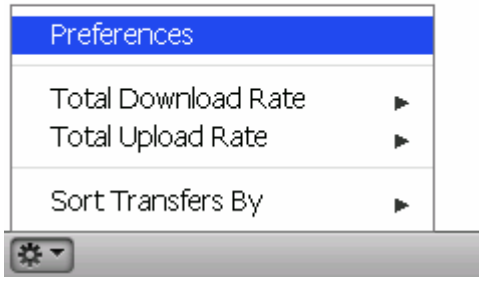
打开 firefox，输入 http://Server_IP:9091:，选择 open：

上传 BT 的 torrent 种子文件即可开始下载文件：

下载的详细信息：



点击页面左下角图标可以控制限速功能：



第十二章：在 9261 上使用 USB 无线网卡

－基于 Virtual PC

本文叙述基于 9261 开发板，在 Linux 下使用 USB 无线网卡的过程。包含了 Linux 内核的配置，USB 网卡驱动和应用程序的配置与编译。无线网卡硬件为 RA2571WF。Linux 版本选择了 2.6.26，开发环境选择了 Virtual PC，其它软件可以类推。

一，准备工作

1. 使用 Virtual PC 安装 Linux 虚拟机

   请参照本站另一篇文档《搭建基于 VPC 的 Linux 平台》。

2. 下载相关软件包

   下载下列软件(可在本站 ftp 上下载)，并传输到 Linux 虚拟机下。

   2008_0506_RT73_Linux_STA_Drv1.1.0.1.tar.bz2

   wireless_tools.29.tar.gz

   RT71W_Firmware_V1.8.zip

3. 准备相关环境

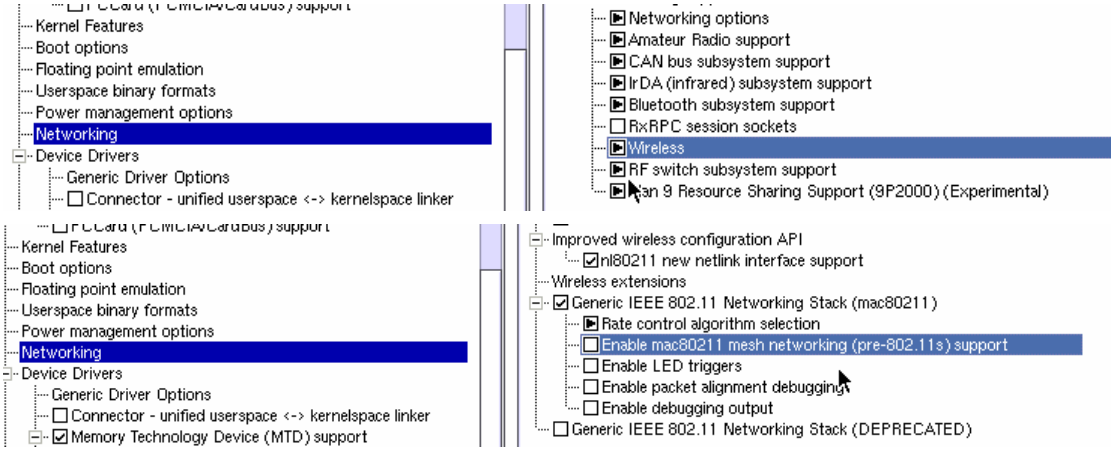
   请参照本镇另一篇文章《9261 上的 Linux 初步应用 3》设置好主机的 NFS 等环境。

二，配置和编译驱动

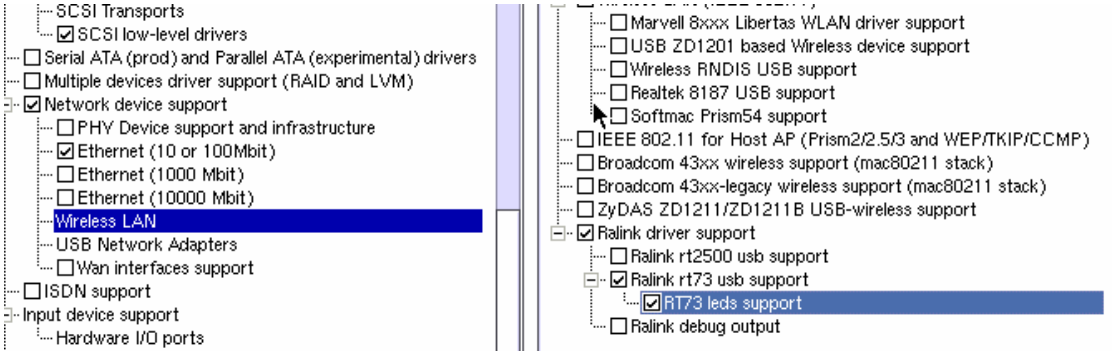
1．方法一：配置内核

Linux2.6.26 中包含对 rt73 的支持，所以可以直接使用 。

配置无线网络支持：



选择对 rt73 的支持：



以此新内核启动，然后连接上无线网卡。网卡的识别信息：

```
[root@mcuzone root]#usb 1-1: new full speed USB device using at91_ohci and addre
ss 2
usb 1-1: configuration #1 chosen from 1 choice
Registered led device: rt73usb-phy0:radio
Registered led device: rt73usb-phy0:assoc
Registered led device: rt73usb-phy0:quality
```

使用 ifconfig –a 浏览网络信息：

```
lo        Link encap:Local Loopback
          LOOPBACK  MTU:16436  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

wlan0     Link encap:Ethernet  HWaddr 00:17:9A:
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

wmaster0  Link encap:UNSPEC  HWaddr                               -00-00-00-00-00-00-00
-00
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

展开 RT71W_Firmware_V1.8.zip，将 rt73.bin 复制到 NFS 目录/lib/firmware 下，修改其权限为 775。

```
[root@Linux4SAM lib]# pwd
/nfs4arm/rootfs/lib
[root@Linux4SAM lib]# mkdir firmware
[root@Linux4SAM lib]# cp /usr/work/app/wifiUSB/rt73.bin ./
[root@Linux4SAM lib]# mv rt73.bin firmware/
[root@Linux4SAM lib]# ls firmware/
rt73.bin
[root@Linux4SAM lib]#
```

配置 ip：

```
[root@mcuzone root]#ifconfig wlan0 inet 192.168.1.100 up
firmware: requesting rt73.bin
[root@mcuzone root]#_
```

```
[root@mcuzone root]#ifconfig
wlan0     Link encap:Ethernet  HWaddr 00:17:
          inet addr:192.168.1.100  Bcast:192.168.1.255  Mask:255.255.255.0
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

使用 iwconfig(编译方法在后面章节描述)配置参数：

```
[root@mcuzone root]#iwconfig wlan0 essid "[redacted]"
[root@mcuzone root]#iwconfig wlan0 key s:[redacted]
[root@mcuzone root]#
```

essid 与 key 分别设置无线网络的 ssid 与密钥。

为了使用无线连接，必要的话，需要设置 DNS 与默认网关。

使用 iwconfig 浏览连接状况：

```
[root@mcuzone root]#iwconfig
lo        no wireless extensions.

eth0      no wireless extensions.

wmaster0  no wireless extensions.

wlan0     IEEE 802.11  ESSID:"[redacted]"
          Mode:Managed  Frequency:2.462 GHz  Access Point: [redacted]
          Bit Rate=1 Mb/s   Tx-Power=19 dBm
          Retry min limit:7   RTS thr:off   Fragment thr=2352 B
          Encryption key:[redacted]
          Link Quality=70/100  Signal level=-58 dBm
          Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
          Tx excessive retries:0  Invalid misc:0   Missed beacon:0

[root@mcuzone root]#
```

可以看到已经连接到了 AP，下面就可以使用 ping 测试。

```
1032 bytes from 192.168.1.5: seq=430 ttl=64 time=5.467 ms
1032 bytes from 192.168.1.5: seq=431 ttl=64 time=6.198 ms
1032 bytes from 192.168.1.5: seq=432 ttl=64 time=6.906 ms
1032 bytes from 192.168.1.5: seq=433 ttl=64 time=6.650 ms
1032 bytes from 192.168.1.5: seq=434 ttl=64 time=6.381 ms
1032 bytes from 192.168.1.5: seq=435 ttl=64 time=7.109 ms
1032 bytes from 192.168.1.5: seq=436 ttl=64 time=6.829 ms
```

添加 default gw：

```
[root@mcuzone root]#route add default gw 192.168.1.1
```

Ping 外网测试：

```
[root@mcuzone root]#ping www.mcuzone.com
PING www.mcuzone.com (220.189.255.38): 56 data bytes
64 bytes from 220.189.255.38: seq=0 ttl=120 time=60.430 ms
64 bytes from 220.189.255.38: seq=1 ttl=120 time=38.644 ms
64 bytes from 220.189.255.38: seq=2 ttl=120 time=52.409 ms
64 bytes from 220.189.255.38: seq=3 ttl=120 time=60.168 ms
64 bytes from 220.189.255.38: seq=4 ttl=120 time=176.858 ms
64 bytes from 220.189.255.38: seq=5 ttl=120 time=78.588 ms
64 bytes from 220.189.255.38: seq=6 ttl=120 time=109.314 ms
64 bytes from 220.189.255.38: seq=7 ttl=120 time=79.033 ms
```

2．方法二：配置 rt73 驱动

如果要从源代码编译，可以采用这个方法。

展开源代码包 2008_0506_RT73_Linux_STA_Drv1.1.0.1.tar.bz2 后,在 Module 文件下将 Makefile.6 复制为 Makefile,然后修改相关内容：

平台为 arm:

```
4  ######################
5  PLATFORM=arm
6  #PLATFORM=CMPC
7
```

平台设置，其中指明了内核文件所在目录：

```
042  endif
043  ifeq ($(PLATFORM),arm)
044  LINUX_SRC = /usr/work/linux-2.6.26
045  WFLAGS += -mcpu=arm926ej-s -fsigned-char
046  endif
047
048  EXTRA_CFLAGS += -I$(LINUX_SRC)/include
```

编译选项：

```
all:
        make -C $(LINUX_SRC) CROSS_COMPILE=arm-none-linux-gnueabi- ARCH=arm SUBDIRS=$(shell pwd
```

输入 make，开始编译：

```
[root@Linux4SAM Module]# ls -al rt73.ko
-rwxrwxr-x    1 root     root       387953 Sep 28 21:42 rt73.ko
[root@Linux4SAM Module]#
```

将此文件复制到 NFS 的对应目录下，连接无线网卡，加载模块：

```
[root@mcuzone 2.6.26]#pwd
/lib/modules/2.6.26
[root@mcuzone 2.6.26]#ls
g_file_storage.ko  gspca.ko           rt73.ko
[root@mcuzone 2.6.26]#insmod rt73.ko
idVendor = 0x7d1, idProduct = 0x3c03
usbcore: registered new interface driver rt73
[root@mcuzone 2.6.26]#_
```

DWL-G122 被正确识别，同时多出一个 rausb0 的连接：

```
rausb0    Link encap:Ethernet  HWaddr 00:17:[illegible]C4[illegible]
          BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)
```

通过 ifconfig 设置 rausb0 的 ip 地址，iwconfig 设置无线参数，即可通过无线网卡连接。

三，编译无线工具

1．编译

展开源码包 wireless_tools.29.tar.gz 后，修改 Makefile，指明编译工具和安装路径：

```
ifndef PREFIX
  PREFIX = $(PWD)/_install
endif

## Compiler to use (modify this for cross comp
CC = arm-none-linux-gnueabi-gcc
## Other tools you need to modify for cross c
AR = arm-none-linux-gnueabi-ar
RANLIB = arm-none-linux-gnueabi-ranlib
```

建立_install 文件夹，然后，make，make install 即可生成相应的 lib：

```
[root@Linux4SAM _install]# pwd
/usr/work/app/wifiUSB/wireless_tools.29/_install
[root@Linux4SAM _install]# ls lib/
libiw.so  libiw.so.29
[root@Linux4SAM _install]# ls sbin/
ifrename  iwconfig  iwevent  iwgetid  iwlist  iwpriv  iwspy
[root@Linux4SAM _install]#
```

将 lib 与 sbin 复制到 NFS 的对应目录下即可。

第十三章 使用 u-boot 测试 9261 开发板

本文叙述使用 u-boot 对本站的 VC9261，SAM9261EK 开发板(以下简称作 9261 开发板)进行基本功能测试的过程。包含了 LCD，NAND，SDRAM，USB Host，以及网络测试。推荐用户在购买了板子后按照本文的内容进行测试，一来可以确认板子的功能，二来可以熟悉板子的操作。

一， 准备工作

1. 安装 SAM-BA

软件的安装和使用请参考本站文档《SAM-BA 中文用户手册 2.0》。

2. 下载相关代码

请到本站的 ftp(ftp://www.mcuzone.com, 用户名与密码均为 mcuzone) 的 /download/ARM_Linux/9261/目录下载下列文件：

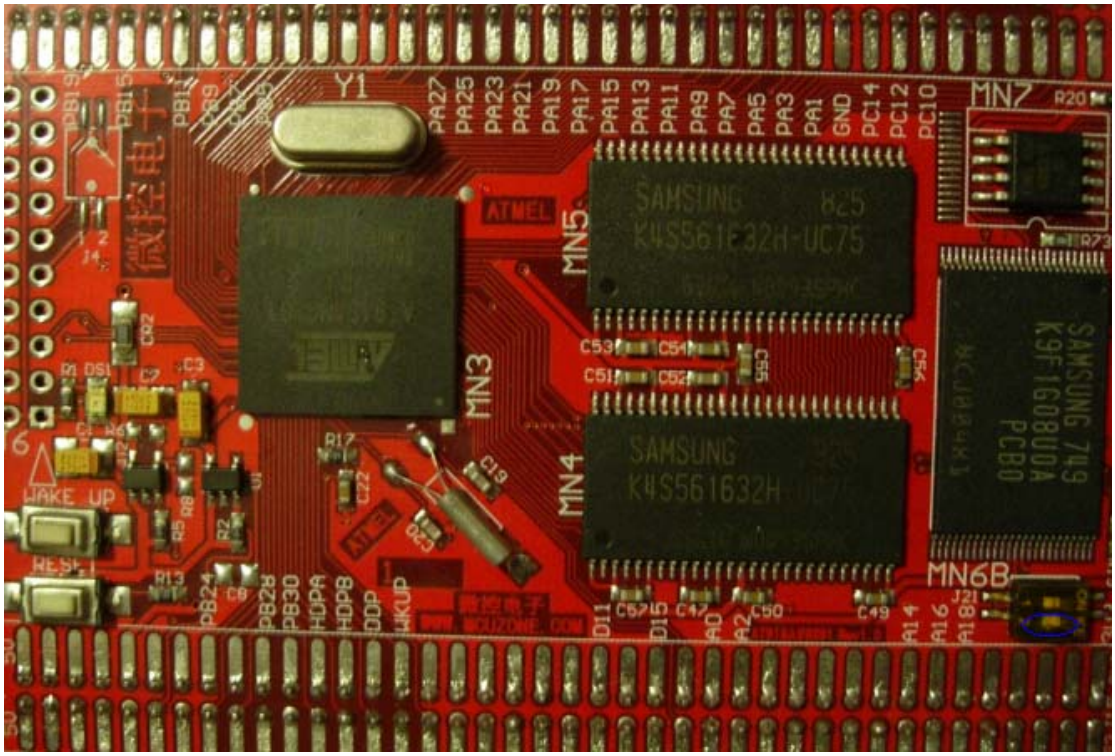
dataflash_at91sam9261ek.bin

u-boot134_mcuzone_logo.bin

二， 烧写代码

1. 配备 9261 核心板的开发板

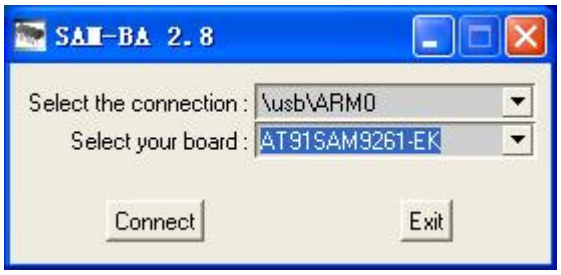
对于使用 9261 核心板的开发板，控制开发板进入 ROM boot 的方式如下：



图中蓝色圈内就是控制 ROM BOOT 的跳线。拨向右侧(以上图为准)就是从 data flash 启动，左侧就是从 ROM 启动。照片中是跳在右侧，将其拨到左侧。

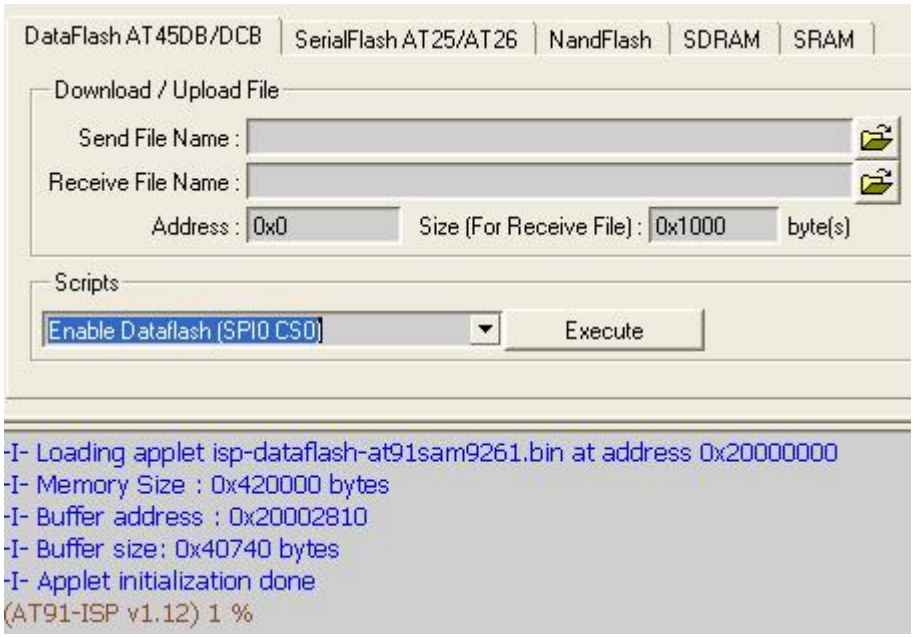
跳线设置后，给开发板上电，接上 USB 线到 PC。如果是首次使用，PC 会提示找到新硬件，点击下一步安装驱动即可。然后运行 SAM-BA，选择开发板为

AT91SAM9261-EK。

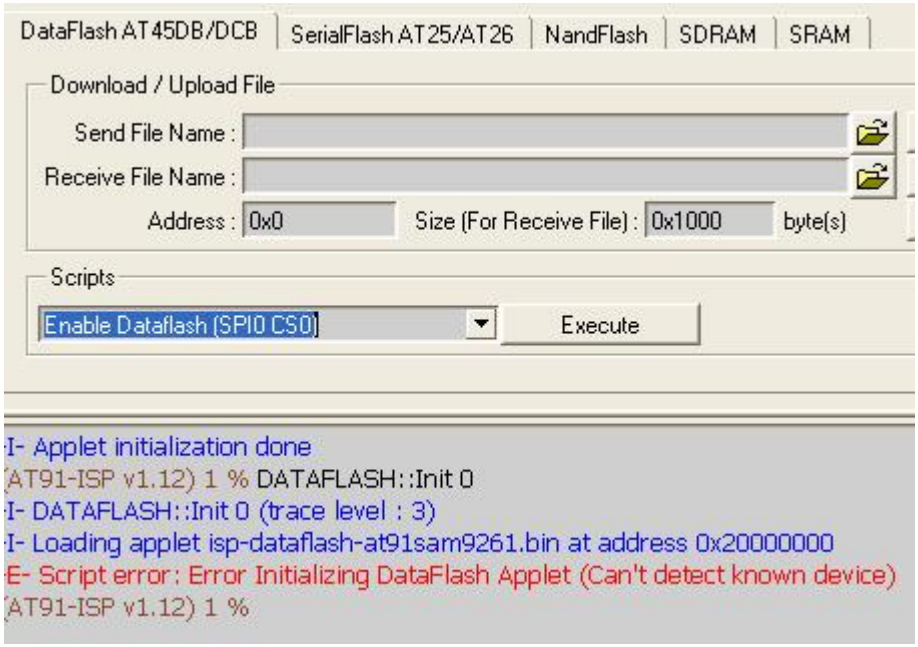


连接成功后，重新将跳线拨到 data flash 一侧。

选择 data flash 的 tab，选择初始化 data flash：

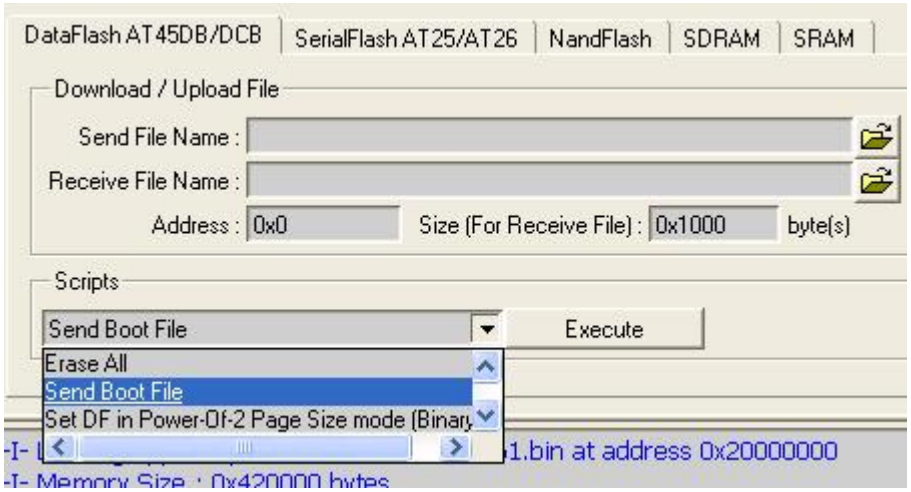


如果出现如下错误提示:



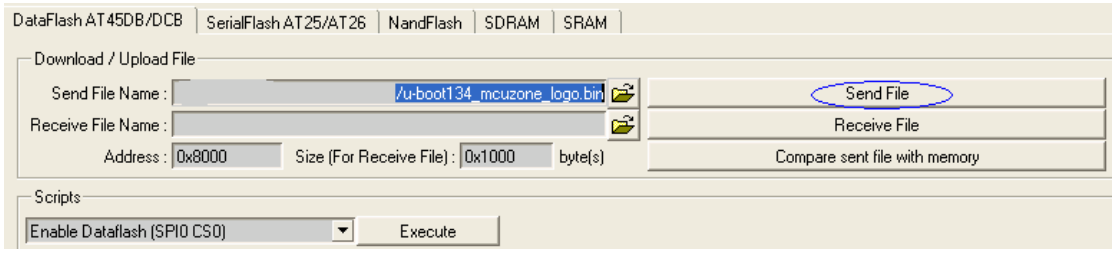
请检查是否忘记将 data flash 跳线拨到 data flash 侧。

选择 Send Boot File，烧写一级 boot：



在弹出的窗口中选择一级 boot 文件(也就是 dataflash_at91sam9261ek.bin)即可开始烧写。由于一级 boot 文件很小，烧写很快结束。

下面的工作是烧写 u-boot，设置地址为 0x8000，这个地址在编译一级 boot 的时候确定，具体可以看一级 boot，也就是 bootstrap 源代码中各种配置的头文件。



选择 u-boot 二进制文件，本文中使用本站编译的 u-boot 文件。相较于 u-boot 网站的代码，针对本站的硬件做了一些改动。

烧写 U-boot 需要若干秒时间，烧写完毕不要忘记选择”compare sent file with memory” 进行校验，如果校验出错，就需要再烧写一遍，直到烧写无误。

烧写完成后关闭 SAM-BA，关闭开发板电源，移除 USB 线。

将开发板的 DBGU 与 PC 连接好，打开超级终端，设置为 115200,n,8,1:

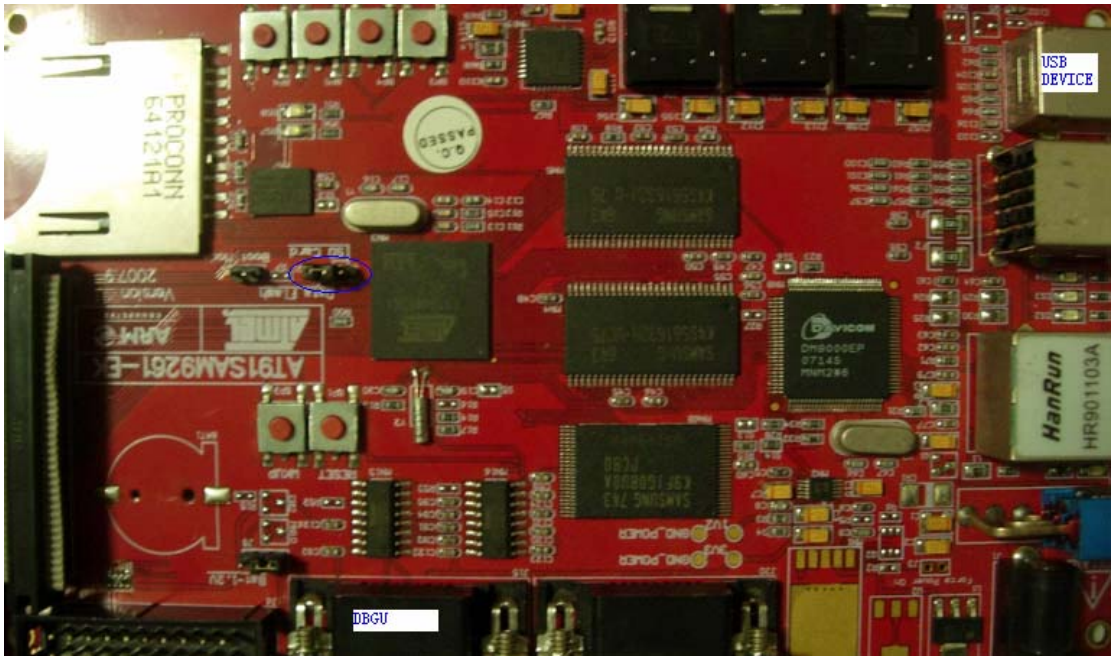
再次打开开发板电源，超级终端将收到如下内容：

```
U-Boot 1.3.4 (Dec 30 2008 - 09:19:23)

DRAM:  64 MB
NAND:  128 MiB
DataFlash:AT45DB321
Nb pages:   8192
Page Size:    528
Size= 4325376 bytes
Logical address: 0xC0000000
Area 0: C0000000 to C0003FFF (RO) Bootstrap
Area 1: C0004000 to C0007FFF      Environment
Area 2: C0008000 to C003FFFF (RO) U-Boot
Area 3: C0040000 to C023FFFF      Kernel
Area 4: C0240000 to C041FFFF      FS
In:    serial
Out:   serial
Err:   serial
dm9000 i/o: 0x30000000, id: 0x90000a46
DM9000: running in 16 bit mode
MAC: 00:0f:b0:f2:a3:ee
operating at 100M full duplex mode
Hit any key to stop autoboot:  0
U-Boot> _
```

这说明烧写正确，开发板基本正常。

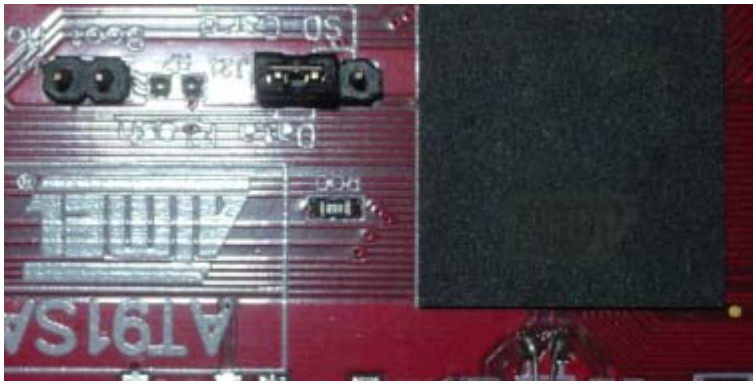
2. 整板的 9261EK

对于整板的开发板，控制开发板进入 ROM boot 的方式如下：



上图中蓝色圈所示即是控制 ROM boot 的跳线，短接左侧(以上图为准)两个针就是选择从 data flash 启动，为了从 ROM 启动，需要去除该跳线：

跳线设置后，烧写操作与上一节相同。连接上 SAM-BA 后记得短接上 data flash boot 跳线：



三， 测试开发板硬件功能

1. 测试 NAND

依次运行下面命令，可以获得 NAND 的结构信息及坏块信息：

```
VC9261> nand info

Device 0: NAND 128MiB 3,3V 8-bit, sector size 128 KiB
VC9261> nand bad

Device 0 bad blocks:
VC9261>
```

上图显示，板子上的 NAND 容量是 128MB(K9F1G08U0A)，没有坏块。

注意：NAND 上有一些坏块是正常现象。

2. 测试 data flash

因为 u-boot 已经通过 data flash 启动，因此 data flash 硬件是正常的。可以通过运行命令 flinfo 以显示 flash 信息：

```
VC9261> flinfo
DataFlash:AT45DB321
Nb pages:    8192
Page Size:    528
Size= 4325376 bytes
Logical address: 0xC0000000
Area 0: C0000000 to C0003FFF (RO) Bootstrap
Area 1: C0004000 to C0007FFF      Environment
Area 2: C0008000 to C003FFFF (RO) U-Boot
Area 3: C0040000 to C024FFFF      Kernel
Area 4: C0250000 to C041FFFF      FS
VC9261> _
```

上图显示了 data flash 的名字，物理结构以及逻辑分区表。

3. 测试 USB Host

准备一个 U 盘，将其连接到开发板的 USB HOST 插座上，运行 usb start 命令扫描 USB 设备，然后通过命令 fatinfo 浏览 U 盘系统信息：

```
VC9261> usb start
(Re)start USB...
USB:   scanning bus for devices... 2 USB Device(s) found
       scanning bus for storage devices... 1 Storage Device(s) found
VC9261> fatinfo usb 0:1
Interface:  USB
  Device 0: Vendor: USB NAND Rev: 0.20 Prod: FLASH DISK
            Type: Removable Hard Disk
            Capacity: 62.5 MB = 0.0 GB (128000 x 512)
Partition 1: Filesystem: FAT16 "NO NAME    "
VC9261> _
```

上面的输出显示了 USB 系统扫描到了一个 USB mass storage 设备，其可用容量为 62.5MB，以及其它一些硬件信息。

使用命令 fatls 浏览 U 盘上的目录结构：

```
VC9261> fatls
usage: fatls <interface> <dev[:part]> [directory]
```

比如查看根目录文件：

```
VC9261> fatls usb 0:1 /
            gongzi/
            1124/
  1934396   uimage27_svga.bin
  1934392   uimage27_vga.bin
            media/
            net/
            linux/
            ar/

2 file(s), 6 dir(s)

VC9261>
```

查看 media 文件夹：

```
VC9261> fatls usb 0:1 /media
            ./
            ../
  6770876   eternal_legend.wmv
 10464986   midiexiang_jay.mp4
 14159872   mv1.asf
  6184119   yx.mp3
  6801954   v_trailer_320_reflect.avi

5 file(s), 2 dir(s)

VC9261> _
```

对应输出文件大小及文件名。

4. 测试 SDRAM

默认情况下，u-boot 运行于 SDRAM 的高位地址，从 0x23F00000 开始，为了测试低位地址，可以使用命令 mtest，该命令将使用不同的数据测试其余的 SDRAM 空间。*注意：该测试极其耗时。*

```
VC9261> mtest
Pattern FFFFFFFE  Writing...  Reading..._
```

这个过程无异常才能说明 SDRAM 空间 OK。

5. 测试 NET

通过网线将开发板连接到路由器，假定路由器 IP 为 192.168.1.1 且支持 DHCP 服务。

注意：请检测相关 PC 的防火墙设置，或者关闭防火墙进行测试。

可以通过下面命令手动设置开发板网卡参数：

```
VC9261> setenv ipaddr 192.168.1.100
VC9261> setenv netmask 255.255.255.0
VC9261> setenv gateway 192.168.1.1
VC9261> setenv ethaddr 3a:1f:34:08:54:54
```

注意：ethaddr 也就是网卡的 MAC 地址，尽量不要使用随意的值，比如 01:02:03:04:05:06 之类，可以按照 PC 网卡，改变一两个数字即可，否则在某些路由器网络中会有问题。

使用 ping 命令测试：

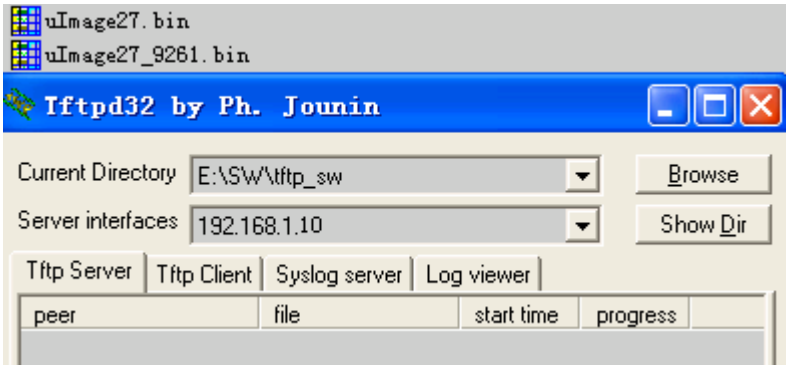
```
VC9261> ping 192.168.1.1
dm9000 i/o: 0x30000000, id: 0x90000a46
DM9000: running in 16 bit mode
MAC: 3a:1f:34:08:54:54
operating at 100M full duplex mode
host 192.168.1.1 is alive
VC9261> _
```

这说明网卡部分的连接没有问题。

进一步测试可以使用 tftp，这里需要设置 tftp server 的 IP：

```
U-Boot> setenv serverip 192.168.1.10
```

这里的 tftp server 为网络中另外一台 Windows PC，IP 为 192.168.1.10。在该 PC 上运行 tftpd32 软件：



在 u-boot 下运行 tftp 0x22200000 uImage27_9261.bin 从 server 上下载文件 uImage27_9261.bin 到板子的 0x22200000 地址：

```
U-Boot> tftp 0x22200000 uImage27_qvga_mzlogo.bin
dm9000 i/o: 0x30000000, id: 0x90000a46
DM9000: running in 16 bit mode
MAC: 3a:1f:34:08:54:54
operating at 100M full duplex mode
TFTP from server 192.168.1.6; our IP address is 192.168.1.100
Filename 'uImage27_qvga_mzlogo.bin'.
Load address: 0x22200000
Loading: ################################################################################
         ################################################################################
         ##
done
Bytes transferred = 1936284 (1d8b9c hex)
```

Tftp 下载的速度比较快，推荐在开发阶段使用。

下载完毕使用 bootm 0x22200000 就可以启动刚才下载的 kernel。

6. 测试 LCD

将开发板断电，连接上 LCD 模块，重新上电，u-boot 会在 LCD 上输出 mcuzone 的 logo 及板子的信息：

由于颜色转换和拍摄原因，上图的 logo 有色差。

四， 启动 Linux

使用 u-boot 启动 Linux，可以参考本站其它的 Linux 应用文章。

第十四章 使用 Linux 测试 9261 开发板

本文叙述使用 Linux 对本站的 VC9261，SAM9261EK 开发板(以下简称作 9261 开发板)进行基本功能测试的过程。包含了 LCD，NAND，SDRAM，USB Host，以及网络测试。推荐用户在购买了板子后按照本文的内容进行测试，一来可以确认板子的功能，二来可以熟悉板子的操作。

一， 准备工作

1. 安装 SAM-BA

软件的安装和使用请参考本站文档《SAM-BA 中文用户手册 2.0》。

2. 下载相关代码

请到本站的 ftp(ftp://www.mcuzone.com, 用户名与密码均为 mcuzone) 的 /download/ARM_Linux/mcuzone/目录下载下列文件：

9261_ctrl_demo.rar

basefs_9261.rar

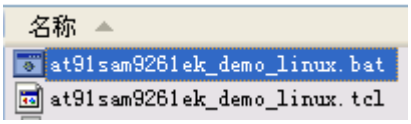
下载后展开压缩包。

3. 熟悉通过 SAM-BA 烧写板子的流程

该部分内容可以参考文档《使用 u-boot 测试 9261 开发板》。

二， 测试 LCD 及触摸屏

使得开发板进入 SAM-BA 状态，然后连接上 USB 线到 PC，不要运行 SAM-BA 软件，而是直接运行批处理文件 at91sam9261ek_demo_linux.bat，如下图：

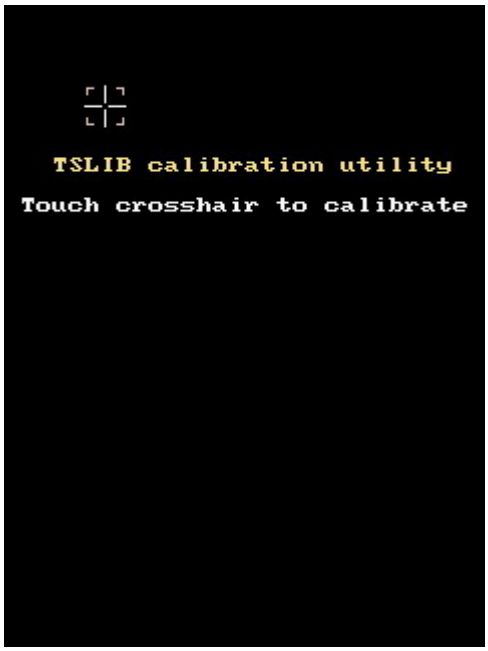


该批批处理文件将完成所有文件的烧写，包括 bootstrap，u-boot, kernel, rootfs。正确烧写完成后，移除 USB 线，开发板断电。

连接串口线到板子的 DBGU，PC 端运行超级终端，设置为 115200,n,8,1。连接好 LCD 模块到开发板。给板子上电，系统将开始启动。超级终端就能收到系统启动的信息。

1. 校准触摸屏

Linux 启动后将首先运行 LCD 的校准程序：



依次点击 5 个测试点，完成触摸屏的校准。

2. 测试 LCD

校准完成后，屏幕上将显示一个欢迎界面：

该界面同时用于 GUI 控件的演示。在屏幕上触摸时，对应触摸点会显示光标。中间的 gif 动画会循环播放。进度条上方的文字指示当前被按下的按键号，最下方的两个按钮可以控制上方进度条的增减。

运行中的截图：

三， 测试外设功能

使用与上一节相同的办法将 basefs_9261.rar 烧写到板子。

连接串口线到 PC，通过网线将板子连接到路由器。这里假设路由器 IP 为 192.168.1.1。开发板 IP 在本例中已经设置为 192.168.1.100。如果需要修改，请在 u-boot 下停止自动启动，修改环境变量中的 bootargs：

```
bootargs=mem=64M console=ttyS0,115200 root=/dev/mtdblock0 rw rootfstype=jffs2 ip
=192.168.1.100:192.168.1.1:192.168.1.1:255.255.255.0
bootcmd= cp.b 0xC0040000 0x22200000 0x0016C9B8; bootm 0x22200000
```

1. 测试 SD 卡

将 SD 卡连接到开发板，然后给开发板上电，Linux 启动信息中将提示：

```
mmc0: host does not support reading read-only switch. assuming write-enable.
mmc0: new SD card at address 4719
mmcblk0: mmc0:4719 SD256 249088KiB
 mmcblk0: p1
```

上图显示 SD 卡被正确识别。登陆系统后运行如下命令：

```
[root@mcuzone root]#cat /proc/partitions
major minor  #blocks  name

  31     0      65536 mtdblock0
  31     1      65536 mtdblock1
 179     0     249088 mmcblk0
 179     1     249037 mmcblk0p1
[root@mcuzone root]#
```

可以看到相应分区，然后可以使用如下命令 mount SD 卡：

```
[root@mcuzone /mnt]#ls
card
[root@mcuzone /mnt]#mount -t vfat /dev/mmcblk0p1 /mnt/card/
[root@mcuzone /mnt]#cd card
[root@mcuzone card]#pwd
/mnt/card
[root@mcuzone card]#ls
9261_24_log.txt                ch18_tty.pdf
```

这里，SD 卡被 mount 到/mnt/card 下，其内容就可以通过该路径进行访问。

2. 测试 USB Host

准备一个 U 盘，将其连接到开发板的 USB HOST 插座上：

```
[root@mcuzone card]#usb 1-1: new full speed USB device using at91_ohci and addre
ss 2
usb 1-1: configuration #1 chosen from 1 choice
scsi0 : SCSI emulation for USB Mass Storage devices
scsi 0:0:0:0: Direct-Access     USB NAND FLASH DISK       0.20 PQ: 0 ANSI: 2
sd 0:0:0:0: [sda] 128000 512-byte hardware sectors (66 MB)
sd 0:0:0:0: [sda] Write Protect is off
sd 0:0:0:0: [sda] Assuming drive cache: write through
sd 0:0:0:0: [sda] 128000 512-byte hardware sectors (66 MB)
sd 0:0:0:0: [sda] Write Protect is off
sd 0:0:0:0: [sda] Assuming drive cache: write through
 sda: sda1
sd 0:0:0:0: [sda] Attached SCSI removable disk
```

系统检测到 U 盘，运行如下命令：

```
[root@mcuzone card]#cat /proc/partitions
major minor  #blocks  name

  31     0      65536 mtdblock0
  31     1      65536 mtdblock1
 179     0     249088 mmcblk0
 179     1     249037 mmcblk0p1
   8     0      64000 sda
   8     1      63984 sda1
```

使用 mount 命令将 U 盘 mount 到某个文件夹：

```
[root@mcuzone /mnt]#ls
card  msd
[root@mcuzone /mnt]#mount -t vfat /dev/sda1 /mnt/msd/
[root@mcuzone /mnt]#cd msd/
```

```
[root@mcuzone msd]#ls uI*
uImage27_svga.bin  uImage27_vga.bin
[root@mcuzone msd]#pwd
/mnt/msd
[root@mcuzone msd]#_
```

此后，U 盘的内容即可通过/mnt/msd 文件夹访问。

3. 测试网络

注意：请检测相关 PC 的防火墙设置，或者关闭防火墙进行测试。

首先 ping 路由器：

```
[root@mcuzone root]#ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1): 56 data bytes
64 bytes from 192.168.1.1: seq=0 ttl=64 time=3.787 ms
64 bytes from 192.168.1.1: seq=1 ttl=64 time=0.801 ms
^
--- 192.168.1.1 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.801/2.294/3.787 ms
[root@mcuzone root]#ping google.com.cn
PING google.com.cn (203.208.33.100): 56 data bytes
64 bytes from 203.208.33.100: seq=0 ttl=245 time=181.594 ms
64 bytes from 203.208.33.100: seq=1 ttl=245 time=306.039 ms
64 bytes from 203.208.33.100: seq=2 ttl=245 time=228.488 ms
64 bytes from 203.208.33.100: seq=3 ttl=245 time=286.344 ms
^
--- google.com.cn ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 181.594/250.616/306.039 ms
[root@mcuzone root]#login[336]: root login on 'pts/0'
_
```

Ping 外网的 IP 也可以通过，就说明网络设置是 OK 的。

板子上的 telnetd 开机后就会运行：

```
 327 root      2972 S    telnetd
 330 root      2976 S    -sh
```

在网络内另一台主机上 telnet 上开发板：

```
Telnet 192.168.1.100

mcuzone login: root
Processing /etc/profile ...
Set search library path in /etc/profile
Set user path in /etc/profile
Welcome to Linux
[root@mcuzone root]#
```

使用 root 登陆即可远程控制开发板。

4. 测试简单 C 程序

到/usr/testapp 下运行 hello：

```
[root@mcuzone testapp]#ls
hello      key        usart_tst
[root@mcuzone testapp]#./hello
hello world!
result=2146226
```

该程序只是不停更新一个计数器。该程序能运行，说明程序运行环境基本 OK。使用 ctrl+c 按键可以中止程序的运行。

```
[root@mcuzone testapp]#ldd hello
        libc.so.6 => /lib/libc.so.6 (0x40026000)
        /lib/ld-linux.so.3 (0x40000000)
[root@mcuzone testapp]#_
```

5. 测试按键

运行下面命令了解按键的资源分配：

```
[root@mcuzone testapp]#cat /proc/bus/input/devices
I: Bus=0019 Vendor=0001 Product=0001 Version=0100
N: Name="gpio-keys"
P: Phys=gpio-keys/input0
S: Sysfs=/class/input/input0
U: Uniq=
H: Handlers=event0
B: EV=3
B: KEY=f 0 0 0 0 0 0 0 0

[root@mcuzone testapp]#_
```

```
[root@mcuzone testapp]#cat /proc/interrupts
           CPU0
  1:    1101394         AIC  at91_tick, rtc0, ttyS0
  9:        122         AIC  mmc0
 10:          0         AIC  at91_udc
 20:        328         AIC  ohci_hcd:usb1
 21:          0         AIC  atmel_lcdfb
 56:          0        GPIO  Button 3
 57:          0        GPIO  Button 2
 58:          0        GPIO  Button 1
 59:          1        GPIO  Button 0
 93:          0        GPIO  at91_udc
107:          0        GPIO  eth0
Err:          0
[root@mcuzone testapp]#
```

到/usr/testapp 下运行 key，按动板子上的按键，检查相应输出：

```
[root@mcuzone testapp]#./key /dev/event0
AT91SAM input event device[/dev/event0] opened[3]
key[0x103] down
key[0x103] up
key[0x103] down
key[0x103] up
key[0x102] down
key[0x102] up
key[0x101] down
key[0x101] up
key[0x100] down
key[0x100] up
key[0x100] down
key[0x100] up
```

6. 测试串口

DBGU 是默认输出设备，也就是 ttyS0，当前的系统控制台足以证明其功能 OK。

硬件上的 USART0 在 Linux 下被识别为 ttyS1:

```
[root@mcuzone testapp]#ls -al /dev/ttyS*
crw-------    1 root     root       4,  64 Jan  1 00:26 /dev/ttyS0
crw-rw----    1 root     root       4,  65 Jan  1 00:00 /dev/ttyS1
[root@mcuzone testapp]#_
```

到/usr/testapp 下运行 usart_tst，该程序从 ttyS1 采集以回车结束的字符串并回显。

```
[root@mcuzone testapp]#./usart_tst
/dev/ttyS1 opened
/dev/ttyS1 test starts, press any key then ENTER
press Z to end test
Please change line to /dev/ttyS1
```

看到该段提示后将开发板上的串口插座拔下，连接到 USART0 对应的插座上，然后在超级终端中输入字符，按回车即可看到字符回显：

```
press Z to end test
Please change line to /dev/ttyS1

Get user input, size = 3
gh

  Get user input, size = 3
sd

  Get user input, size = 3
65

  Get user input, size = 3
78

  Get user input, size = 3
90

  Get user input, size = 3
-=

  Get user input, size = 11
qwertyuiop
```

测试完成后请将串口线还原，重新连接到 DBGU 上。可以将串口线连接到 DBGU 后通过 ctrl+c 中止测试。

7. 测试 NAND

板载的 128MB NAND 默认情况下被分成两个 64MB 的分区：

```
[root@mcuzone testapp]#cat /proc/mtd
dev:    size   erasesize  name
mtd0: 04000000 00020000 "Partition 1"
mtd1: 04000000 00020000 "Partition 2"
[root@mcuzone testapp]#
```

其中分区 0 对应的 mtdblock0 已经被用作了 rootfs，分区 1 可以供用户使用。

使用命令 mount 可以将其以 jffs2 文件系统 mount 到系统文件夹：

```
[root@mcuzone /mnt]#mkdir mtd
[root@mcuzone /mnt]#mount -t jffs2 /dev/mtdblock1 /mnt/mtd/
[root@mcuzone /mnt]#df
Filesystem           1k-blocks      Used Available Use% Mounted on
rootfs                   65536      5716     59820   9% /
/dev/root                65536      5716     59820   9% /
tmpfs                     1024         0      1024   0% /var/volatile
/dev/mmcblk0p1          248776     37260    211516  15% /mnt/card
/dev/sda1                63716     58503      5213  92% /mnt/msd
/dev/mtdblock1           65536      1792     63744   3% /mnt/mtd
[root@mcuzone /mnt]#_
```

用户可以对/mnt/mtd 进行读写，写入的数据将保存在 NAND 上。

8. 测试 RTC

首先检查设备节点：

```
[root@mcuzone /dev]#ls -al rtc*
crw-rw----    1 root     root     254,   0 Jan  1 00:00 rtc0
[root@mcuzone /dev]#_
```

通过下面命令更新系统时间并同步到 rtc：

```
[root@mcuzone /dev]#date 021221002009
Thu Feb 12 21:00:00 UTC 2009
[root@mcuzone /dev]#hwclock -w
[root@mcuzone /dev]#_
```

下次启动时系统将从 rtc 同步时间：

```
rtc-at91sam9 at91_rtt.0: setting system clock to 2009-02-12 21:05:45 UTC (123447
2745)
```

第十五章 在 Linux 下驱动 9261EK 上的 LED

本文叙述在 Linux 下驱动本站的 VC9261 上的 LED，包含 kernel 驱动模块的编程，编译，加载，对应的应用程序的编程及使用，也包含了部分在 user space 访问 GPIO 寄存器的方法。以控制 GPIO 为例简要说明 Linux 下的驱动的编程及使用。

一， 准备工作

1. 在开发板上安装 Linux
   请按照本站其它文档，在开发板上建立 Linux 运行环境。

2. 下载相关代码
   请到本站的 ftp(ftp://www.mcuzone.com, 用户名与密码均为 mcuzone) 的 /download/ARM_Linux/mcuzone/目录下载下列文件：
   led9261drv.tar.gz
   下载后展开压缩包。

3. 在 PC 上搭建 Linux 开发环境
   请参考本站文档《基于 VPC 建立 ARM_Linux 开发环境》,《为 9261 编译 Linux》。

二， 内核驱动

将从 mcuzone FTP 上下载的压缩包展开，上传到虚拟机目录下，假定为 /usr/work/drvlinux/led:

```
root@u810:/usr/work/drvlinux/led# ls -al
total 28
drwxrwxr-x 2 nobody nogroup 4096 2009-03-30 20:02 .
drwxrwxr-x 3 nobody nogroup 4096 2009-02-28 04:05 ..
-rwxrwxr-x 1 nobody nogroup 4168 2009-03-29 12:10 led9261.c
-rwxrwxr-x 1 nobody nogroup 1299 2009-03-29 12:14 led_test.c
-rwxrwxr-x 1 nobody nogroup  428 2009-03-29 20:52 Makefile
-rwx------ 1 nobody nogroup 1253 2009-03-29 16:58 usrleddrv.c
root@u810:/usr/work/drvlinux/led#
```

led9261.c 为内核驱动程序，led_test.c 是用于测试内核驱动的用户程序，Makefile 用于编译内核驱动模块，usrleddrv.c 是从 user space 直接通过控制 GPIO 寄存器控制 LED 的例子程序。

1. 编译内核驱动

首先核对 Makefile：

```
root@u810:/usr/work/drvlinux/led# cat Makefile

obj-m := led9261.o
KERNELDIR := /usr/work/linux-2.6.27
PWD := $(shell pwd)

CROSS_COMPILE := arm-none-linux-gnueabi-
#CROSS_COMPILE := arm-linux-
CC := $(CROSS_COMPILE)gcc
LD := $(CROSS_COMPILE)ld

modules:
        $(MAKE) -C $(KERNELDIR) SUBDIRS=$(PWD) ARCH=arm modules

modules_install:
        $(MAKE) -C $(KERNELDIR) SUBDIRS=$(PWD) ARCH=arm modules_install

clean:
        rm -rf *.o *~ core .depend .*.cmd *.ko *.mod.c .tmp_versions *.symvers
root@u810:/usr/work/drvlinux/led#
```

里面指定了内核所在的文件夹(KERNELDIR)，需要根据实际开发 PC 上的位置设置，还有就是交叉编译的工具链，需要指定，这里使用 arm-none-linux-gnueabi-。

```
root@u810:/usr/work/drvlinux/led# 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 --wit
h-gnu-as --with-gnu-ld --enable-languages=c,c++ --enable-shared --enable-symvers=gnu --enable-__cxa_atexit --with-pkgver
sion='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-ar
m-none-linux-gnueabi-i686-pc-linux-gnu/usr --with-mpfr=/scratch/julian/lite-respin/linux/obj/host-libs-2008q3-72-arm-non
e-linux-gnueabi-i686-pc-linux-gnu/usr --disable-libgomp --enable-poison-system-directories --with-build-time-tools=/scra
tch/julian/lite-respin/linux/install/arm-none-linux-gnueabi/bin --with-build-time-tools=/scratch/julian/lite-respin/linu
x/install/arm-none-linux-gnueabi/bin
Thread model: posix
gcc version 4.3.2 (Sourcery G++ Lite 2008q3-72)
root@u810:/usr/work/drvlinux/led#
```

核对无误后输入 make：

```
root@u810:/usr/work/drvlinux/led# make
make -C /usr/work/linux-2.6.27 SUBDIRS=/usr/work/drvlinux/led ARCH=arm modules
make[1]: Entering directory `/usr/work/linux-2.6.27'
  CC [M]  /usr/work/drvlinux/led/led9261.o
/usr/work/drvlinux/led/led9261.c:20:2: warning: #warning build for Linux-2.6.27 or above
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /usr/work/drvlinux/led/led9261.mod.o
  LD [M]  /usr/work/drvlinux/led/led9261.ko
make[1]: Leaving directory `/usr/work/linux-2.6.27'
root@u810:/usr/work/drvlinux/led#
```

如果编译提示缺失头文件，请检查 Makefile 里指定的 kernel 文件夹是否有对应的头文件。

编译无误将会生成对应的内核驱动.ko 文件：

```
root@u810:/usr/work/drvlinux/led# ls
led9261.c   led9261.mod.c  led9261.o   Makefile       Module.symvers
led9261.ko  led9261.mod.o  led_test.c  modules.order  usrleddrv.c
root@u810:/usr/work/drvlinux/led# file led9261.ko
led9261.ko: ELF 32-bit LSB relocatable, ARM, version 1 (SYSV), not stripped
root@u810:/usr/work/drvlinux/led#
```

将 led9261.ko 复制到 NFS 目录。

2. 编译测试程序

使用下面命令编译驱动的测试程序：

```
root@u810:/usr/work/drvlinux/led# arm-none-linux-gnueabi-gcc -o led_test led_test.c
root@u810:/usr/work/drvlinux/led# file led_test
led_test: ELF 32-bit LSB executable, ARM, version 1 (SYSV), for GNU/Linux 2.6.14, dynamically linked (uses shared libs),
 not stripped
root@u810:/usr/work/drvlinux/led# _
```

将测试程序 led_test 复制到 NFS 目录。

3. 测试驱动程序

在开发板上运行 linux，通过 NFS 将开发机上的驱动 mount 到开发板，比如/usr/drvtst:

```
[root@mcuzone drvtst]#pwd
/usr/drvtst
[root@mcuzone drvtst]#ls led*
led9261.ko  led_test
[root@mcuzone drvtst]#ldd led_test
        libc.so.6 => /lib/libc.so.6 (0x40026000)
        /lib/ld-linux.so.3 (0x40000000)
[root@mcuzone drvtst]#_
```

在开发板上建立 kernel 版本号对应的模块目录：

```
[root@mcuzone drvtst]#cat /proc/version
Linux version 2.6.27 (root@u810) (gcc version 4.3.2 (Sourcery G++ Lite 2008q3-72
) ) #15 Sun Mar 29 20:28:14 CST 2009
[root@mcuzone drvtst]#_
```

```
[root@mcuzone drvtst]#ls /lib/modules/
2.6.27
[root@mcuzone drvtst]#_
```

使用 insmod 加载模块：

```
[root@mcuzone drvtst]#insmod --help
BusyBox v1.13.3 (2009-03-29 21:37:07 CST) multi-call binary

Usage: insmod [-knqrsv] MODULE [symbol=value...]

Options:
        -n      Dry run
        -q      Quiet
        -r      Remove module (stacks) or do autoclean
        -s      Report via syslog instead of stderr
        -v      Verbose

[root@mcuzone drvtst]#
```

运行命令：

```
[root@mcuzone drvtst]#insmod led9261.ko
register led device MAJOR=252 MINOR=0
[root@mcuzone drvtst]#lsmod
led9261 2636 0 - Live 0xbf000000
[root@mcuzone drvtst]#
```

注意其中的 MAJOR 与 MINOR，下面需要使用这两个参数创建设备节点：

```
[root@mcuzone drvtst]#mknod led7 c 252 0
[root@mcuzone drvtst]#ls -al /dev/led7
crw-r--r--    1 root     root      252,   0 Jan  1 00:03 /dev/led7
[root@mcuzone drvtst]#
```

取 led7 这个名字是因为测试程序里试图打开这个设备。

运行下面命令，也可以看到对应的设备：

```
[root@mcuzone root]#cat /proc/devices
Character devices:
  1 mem
  2 pty
  3 ttyp
  4 /dev/vc/0
  4 tty
  4 ttyS
```

```
189 usb_device
252 led7
253 usb_endpoint
254 rtc
```

运行测试程序 led_test：

```
[root@mcuzone drvtst]#./led_test
Example: ./led_test [ON / OFF]
[root@mcuzone drvtst]#./led_test ON
intend LED state [ON]

/dev/led7 opened, fd = 3
/dev/led7 closed
[root@mcuzone drvtst]#./led_test OFF
intend LED state [OFF]

/dev/led7 opened, fd = 3
/dev/led7 closed
[root@mcuzone drvtst]#_
```

输入 led_test ON/OFF 就可以控制 LED 的亮灭。

三， 在 user space 控制 GPIO

为了控制连接在 GPIO 上的 LED，也可以通过 mmap 将 GPIO 相关寄存器的地址映射到 user space，然后通过读写对应寄存器完成对 LED 的控制。

相关的代码可以参考 usrleddrv.c。

9. 编译程序

使用下面命令编译：

```
root@u810:/usr/work/drvlinux/led# arm-none-linux-gnueabi-gcc -o usrleddrv usrleddrv.c
root@u810:/usr/work/drvlinux/led# file usrleddrv
usrleddrv: ELF 32-bit LSB executable, ARM, version 1 (SYSV), for GNU/Linux 2.6.14, dynamically linked (uses shared libs)
, not stripped
root@u810:/usr/work/drvlinux/led#
```

将 usrleddrv 复制到 NFS 目录下。

10．测试程序

在开发板那上 mount 开发机上的 NFS 目录，直接运行 usrleddrv 即可看到对应的 led 按照程序设定闪烁。

```
[root@mcuzone drvtst]#ldd usrleddrv
        libc.so.6 => /lib/libc.so.6 (0x40026000)
        /lib/ld-linux.so.3 (0x40000000)
[root@mcuzone drvtst]#./usrleddrv
```

第十六章 在 Linux 下使用 9263EK 上的 EEPROM

本文叙述在 Linux 下使用本站的 VC9263EK 上的 EEPROM(24C02)，包含 kernel 驱动模块的配置，对应 EEPROM 信息的添加，以及应用程序的编程及使用。

一， 准备工作

1. 在开发板上安装 Linux

请按照本站其它文档，在开发板上建立 Linux 运行环境。

2. 下载相关代码

请到本站的 ftp(ftp://www.mcuzone.com, 用户名与密码均为 mcuzone) 的 /download/ARM_Linux/kernel_src/目录下载下列文件：

linux-2.6.27.tar.bz2

下载后展开压缩包。

3. 在 PC 上搭建 Linux 开发环境

请参考本站文档《基于 VPC 建立 ARM_Linux 开发环境》,《为 9263 编译 Linux》。

二， 内核驱动

为了使用 24C02，需要在开发板文件中提供对应 EEPROM 的信息，然后在 kernel 的配置中使能 I2C，并选择 EEPROM 的支持。

1. 配置目标版

修改文件/arch/arm/mach-at91/board-sam9263ek.c，添加 24C02 的信息：

```
245  /*
246   * I2C devices
247   */
248  #if 0
249  static struct at24_platform_data at24c512 = {
250      .byte_len = SZ_512K / 8,
251      .page_size = 128,
252      .flags = = AT24_FLAG_ADDR16,
253  };
254
255
256  static struct i2c_board_info __initdata ek_i2c_devices[] = {
257      {
258          I2C_BOARD_INFO("24c512", 0x50),
259          .platform_data = &at24c512,
260      },
261      /* more devices can be added using expansion connectors */
262  };
263  #else
264  static struct at24_platform_data at24c02 = {
265      .byte_len = SZ_1K * 2 / 8,
266      .page_size = 8,
267      .flags = = 0,
268  };
269
270
271  static struct i2c_board_info __initdata ek_i2c_devices[] = {
272      {
273          I2C_BOARD_INFO("24c02", 0x50),
274          .platform_data = &at24c02,
275      },
276      /* more devices can be added using expansion connectors */
277  };
278  #endif
```

并在板子初始化时，初始化 i2c：

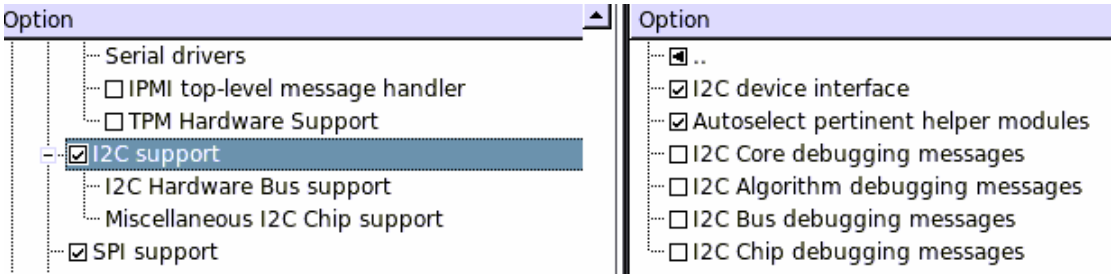
```
483      /* NAND */
484      ek_add_device_nand();
485      /* I2C */
486  #if 0
487      at91_add_device_i2c(NULL, 0);
488  #else
489      at91_add_device_i2c(ek_i2c_devices, ARRAY_SIZE(ek_i2c_devices));
490  #endif
491      /* LCD Controller */
492      at91_add_device_lcdc(&ek_lcdc_data);
```

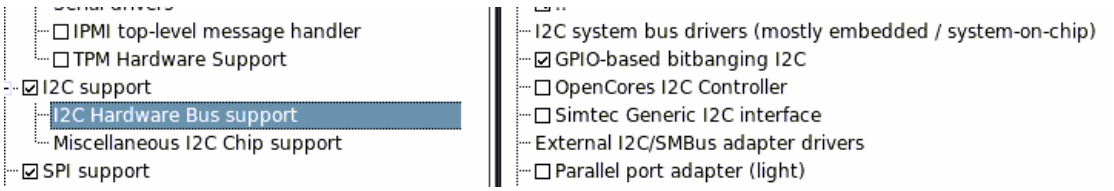
注意添加 i2c 设备对应头文件：

```
28   #include <linux/spi/ads7846.h>
29   #include <linux/i2c/at24.h>
30   #include <linux/fb.h>
```

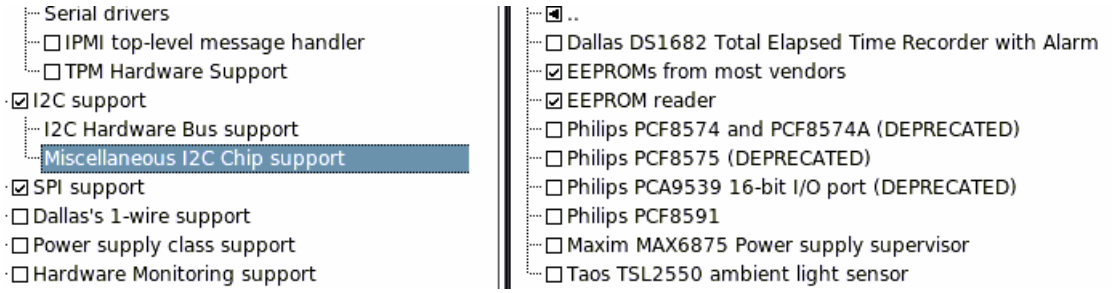
2. 配置内核模块

配置内核，选择 I2C 和 EEPROM 的支持：





选择 EEPROM 的支持：



3. 编译新内核

使用以上设置，编译内核，烧写到开发板并启动。

启动信息里将提示 I2C 模块被加载：

```
103  i2c /dev entries driver
104  at24 0-0050: 256 byte 24c02 EEPROM (writable)
105  i2c-gpio i2c-gpio: using pins 68 (SDA) and 69 (SCL)
106  cpuidle: using governor ladder
```

三， 访问 EEPROM

为了验证内核编译以及硬件连接的正确性，可以通过/sys 目录下的相关节点来测试 EEPROM。

11. 写入 EEPROM

使用下面命令向 EEPROM 写入数据：

```
[root@mcuzone root]#echo "mcuzone: 24c02 on SAM9263ek in Linux" > /sys/bus/i2c/devices/0-0050/eeprom
[root@mcuzone root]#
```

12. 读取 EEPROM

通过下面命令读取数据：

```
[root@mcuzone root]#cat /sys/bus/i2c/devices/0-0050/eeprom
mcuzone: 24c02 on SAM9263ek in Linux
```

关机重启，可以看到数据仍被保存。

可以看到，通过这种方式，不需要其它程序就可以通过访问文件来访问 24C02 的整个硬件。

四， 通过用户程序访问

利用内核提供的 char device 接口，也可以从用户程序访问 24C02。

首先看到/dev 下的 i2c 设备接口：

```
[root@mcuzone /dev]#pwd
/dev
[root@mcuzone /dev]#ls -al i2c*
crw-rw----    1 root     root      89,   0 Apr 19  2009 i2c-0
[root@mcuzone /dev]#
```

该接口的使用可以参考内核文档：

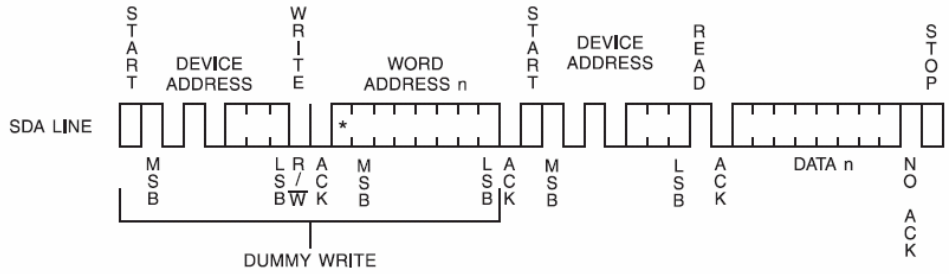
/linux/Documentation/i2c/dev-interface

据此，可以先初始化设备接口：

```
40
41      fd = open("/dev/i2c-0", O_RDWR);
42
43      if(0 > fd)
44      {
45          perror("open i2c device");
46          exit(-1);
47      }
48
49      printf("%s opened[%d]\n\r", I2C_DEV_NODE, fd);
50
51      res = ioctl(fd, I2C_SLAVE_FORCE, EEP_24C_ADDR);
52      printf("\n%d\n", res);
53
54      res=ioctl(fd, I2C_TIMEOUT, 1);
55      printf("\n%d\n", res);
56
57      res=ioctl(fd, I2C_RETRIES, 10);
58      printf("\n%d\n", res);
```

对照 24C02 的读取时序，可以实现读操作：

Figure 5. Random Read



(* = DON'T CARE bit for 1K)

据此，读操作可以编程如下：

```
59      buf[0] = 0;
60      if (write(fd, buf, 1))
61      {
62          printf("from 0\n\r");
63      }
64
65      if (read(fd, buf, 128))
66      {
67          printf("Read %d char: [0x%x] [%c] [%s]\n", size, buf[0], buf[0], buf);
68      }
69
```

使用编译器编译该代码：

```
i2c_eep.c
root@u810:/usr/work/applinux/i2c_eep# arm-none-linux-gnueabi-gcc -o i2c_tst i2c_eep.c
root@u810:/usr/work/applinux/i2c_eep# ls
i2c_eep.c  i2c_tst
root@u810:/usr/work/applinux/i2c_eep# file i2c_tst
i2c_tst: ELF 32-bit LSB executable, ARM, version 1 (SYSV), for GNU/Linux 2.6.14, dynamically linked (uses shared libs),
not stripped
root@u810:/usr/work/applinux/i2c_eep#
```

将程序下载到开发板运行：

```
[root@mcuzone testapp]#ldd i2c_tst
        libc.so.6 => /lib/libc.so.6 (0x40026000)
        /lib/ld-linux.so.3 (0x40000000)
[root@mcuzone testapp]#./i2c_tst
/dev/i2c-0 opened[3]

0

0

0
from 0
Read 1 char: [0x6d] [m] [mcuzone: 24c02 on SAM9263ek in Linux

            ]
[root@mcuzone testapp]#_
```

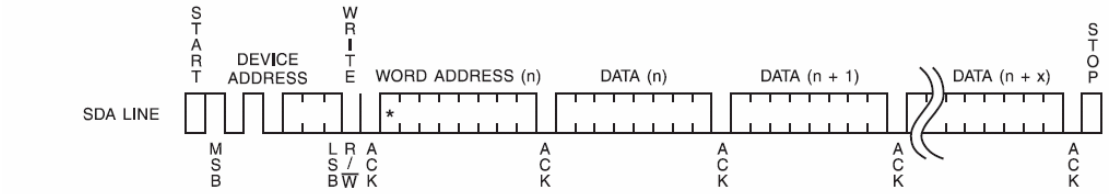
OK，刚才通过文件写入的数据被读取了出来。

对于 EEPROM，数据写入前需要确定芯片的页大小：

AT24C02, 2K SERIAL EEPROM: Internally organized with 32 pages of 8 bytes each, the 2K requires an 8-bit data word address for random word addressing.

然后根据页写的时序：

Figure 3. Page Write



(* = DON'T CARE bit for 1K)

进行写操作编程：

```
73      /* write date from address 36 */
74      memset(buf, ' ', BUFF_SIZE);
75      buf[0] = 1; /* start address */
76      strcpy(&buf[1], "0123456789abcdef");
77      printf("%s\n\r", buf);
78      size = write(fd, buf, 16);
79      if (size)
80      {
81          printf("24C02 write size[%d]\n\r", size);
82      }
```

编译后可以与读程序配合测试：

```
[root@mcuzone testapp]#./i2c_tst
/dev/i2c-0 opened[3]

0

0

0
@0123456789abcdef
24C02 write size[16]
[root@mcuzone testapp]#./i2c_r
/dev/i2c-0 opened[3]

0

0

0
from 0
Read [256] char: [0x74] [t] [t0123456789abcdejklmnopqrstuvwxyz
st]
[root@mcuzone testapp]#_
```

可以看到相关数据的写入情况。

五， 一些事项

可以看到，使用 I2C 的 char 设备用户接口，可以方便的控制读取及写入的位置，更适合在不同地址存储不同配置的系统，而文件操作方式更适合保存单一的文本信息。

请保证 Linux 开发环境的正确，否则可能会出现问题。

如果出现问题，可以在代码中打印调试信息，以定位问题。

第十七章 使用 GDB 调试 Linux 应用程序

本文演示使用 GDB 调试本站 ARM 开发板上的 Linux 应用程序的过程。包含源程序编译，GDB，DDD，insight 的使用。调试器与开发板使用网络连接。PC 上的 Linux 发行版本选择 Ubuntu8.10。

一， 准备工作

1. 在开发板上建立 Linux 运行环境

在 Linux 上编译 9261 的 Linux kernel，rootfs，并与开发 PC 机连接通过 NFS 启动。

请参考本站文档:《基于 VPC 建立 ARM_Linux 开发环境》,《使用 busybox 制作根文件系统》,《为 9261 编译 Linux》。

2. 下载相关代码

请到本站的 ftp(ftp://www.mcuzone.com, 用户名与密码均为 mcuzone) 的 /download/ARM_Linux/tools/目录下载下列文件：

insight-6.8.tar.bz2

下载后展开压缩包。

二， 编译源代码

编写一个简单的 hello 程序：

main.c

```
#include <stdio.h>

int main(void)
{
    int a, b, c, d;

    a = 1;
    b = 2;
    c = 3;
    d = 0;

    printf("hello world!\n\r");

    while(1)
    {
        //d = a * b + c;

        a ++; b ++; c ++;

        printf("result=%d\n\r", d);

        d = a * b + c;
    }

    return 0;
}
```

使用如下命令编译：

```
root@u810:/usr/work/applinux/hello# arm-none-linux-gnueabi-gcc -mcpu=arm926ej-s -g -O2 -o dbgtst main.c
root@u810:/usr/work/applinux/hello# file dbgtst
dbgtst: ELF 32-bit LSB executable, ARM, version 1 (SYSV), for GNU/Linux 2.6.14, dynamically linked (uses shared libs), n
ot stripped
root@u810:/usr/work/applinux/hello# cp dbgtst /nfsroot/baseroot/usr/testapp/
root@u810:/usr/work/applinux/hello#
```

注意其中的-g 参数，指明生成 debug 信息。

将生成的 dbgtst 文件复制到 rootfs 的目录下，比如/usr/testapp。

启动开发板，以 NFS 方式 mount 根文件系统(rootfs)。

本文档中开发板的 IP 为 192.168.1.100，Linux server 的 IP 为 192.168.1.5。

三， 使用 GDB

从 arm-none-linux-gnueabi 的工具链相应目录下复制 gdbserver 到开发板：

```
root@u810:/opt/arm-2008q3/arm-none-linux-gnueabi/libc/usr/bin# ls
catchsegv  gencat   getent  ldd     localedef  pcprofiledump  sprof     xtrace
gdbserver  getconf  iconv   locale  mtrace     rpcgen         tzselect
root@u810:/opt/arm-2008q3/arm-none-linux-gnueabi/libc/usr/bin# file gdbserver
gdbserver: ELF 32-bit LSB executable, ARM, version 1 (SYSV), for GNU/Linux 2.6.14, dynamically linked (uses shared libs)
, not stripped
root@u810:/opt/arm-2008q3/arm-none-linux-gnueabi/libc/usr/bin# pwd
/opt/arm-2008q3/arm-none-linux-gnueabi/libc/usr/bin
root@u810:/opt/arm-2008q3/arm-none-linux-gnueabi/libc/usr/bin#
```

将其放到开发板的/usr/bin 下：

```
[root@mcuzone testapp]#which gdbserver
/usr/bin/gdbserver
[root@mcuzone testapp]#
```

在开发板上运行 gdbserver：

```
[root@mcuzone testapp]#pwd
/usr/testapp
[root@mcuzone testapp]#ldd dbgtst
        libc.so.6 => /lib/libc.so.6 (0x40026000)
        /lib/ld-linux.so.3 (0x40000000)
[root@mcuzone testapp]#gdbserver 192.168.1.5:2345 dbgtst
Process dbgtst created; pid = 433
Listening on port 2345
```

此时 gdbserver 将在 2345 端口等待远端连接。

切换到 Linux server 上，运行 arm-none-linux-gnueabi-gdb：

```
root@u810:/usr/work/applinux/hello# which arm-none-linux-gnueabi-gdb
/opt/arm-2008q3/bin/arm-none-linux-gnueabi-gdb
root@u810:/usr/work/applinux/hello# arm-none-linux-gnueabi-gdb dbgtst
GNU gdb (Sourcery G++ Lite 2008q3-72) 6.8.50.20080821-cvs
Copyright (C) 2008 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=i686-pc-linux-gnu --target=arm-none-linux-gnueabi".
For bug reporting instructions, please see:
<https://support.codesourcery.com/GNUToolchain/>...
(gdb) target remote 192.168.1.100:2345
Remote debugging using 192.168.1.100:2345
warning: Unable to find dynamic linker breakpoint function.
GDB will be unable to debug shared library initializers
and track explicitly loaded dynamic code.
0x400007b0 in ?? ()
(gdb)
```

通过 arm-none-linux-gnueabi-gdb dbgtst 启动命令行调试器，在(gdb)提示符下运行命令 target remote 192.168.1.100:2345 将连接到开发板的 gdbserver，下面就可以用 gdb

的命令开始调试。

例如 l 命令列出源代码：

```
(gdb) l
3       int main(void)
4       {
5           int a, b, c, d;
6
7           a = 1;
8           b = 2;
9           c = 3;
10          d = 0;
11
12          printf("hello world!\n\r");
(gdb)
```

使用 b 命令设置断点：

```
(gdb) b 7
Breakpoint 1 at 0x8374: file main.c, line 7.
(gdb)
```

使用 c 运行程序：

```
(gdb) c
Continuing.
warning: .dynamic section for "/lib/libc.so.6" is not at the expected address (wrong library or version m
smatch?)
Error while mapping shared library sections:
/lib/ld-linux.so.3: No such file or directory.

Breakpoint 1, main () at main.c:12
12          printf("hello world!\n\r");
(gdb)
```

再次设置断点：

```
(gdb) l
17
18              a ++; b ++; c ++;
19
20              printf("result=%d\n\r", d);
21
22              d = a * b + c;
23          }
24
25          return 0;
26      }
(gdb) b 18
Breakpoint 2 at 0x8384: file main.c, line 18.
(gdb) c
Continuing.

Breakpoint 2, main () at main.c:18
18              a ++; b ++; c ++;
(gdb)
```

使用 n 单步运行：

```
(gdb) c
Continuing.

Breakpoint 2, main () at main.c:18
18              a ++; b ++; c ++;
(gdb) n
20              printf("result=%d\n\r", d);
(gdb)
```

使用 p 打印变量：

```
(gdb) n
20              printf("result=%d\n\r", d);
(gdb) p a
No symbol "a" in current context.
(gdb) p b
$1 = 6461
(gdb) p c
No symbol "c" in current context.
(gdb) n
22              d = a * b + c;
(gdb) n

Breakpoint 2, main () at main.c:18
18              a ++; b ++; c ++;
(gdb) p d
$2 = 41744522
```

由于优化的原因，不是所有的变量都可以通过 p 查看。

调试结束，通过 disconnect 断开 gdbserver 的连接，停止调试，并使用 q 退出 gdb：

```
(gdb) disconnect
Ending remote debugging.
(gdb) q
root@u810:/usr/work/applinux/hello#
```

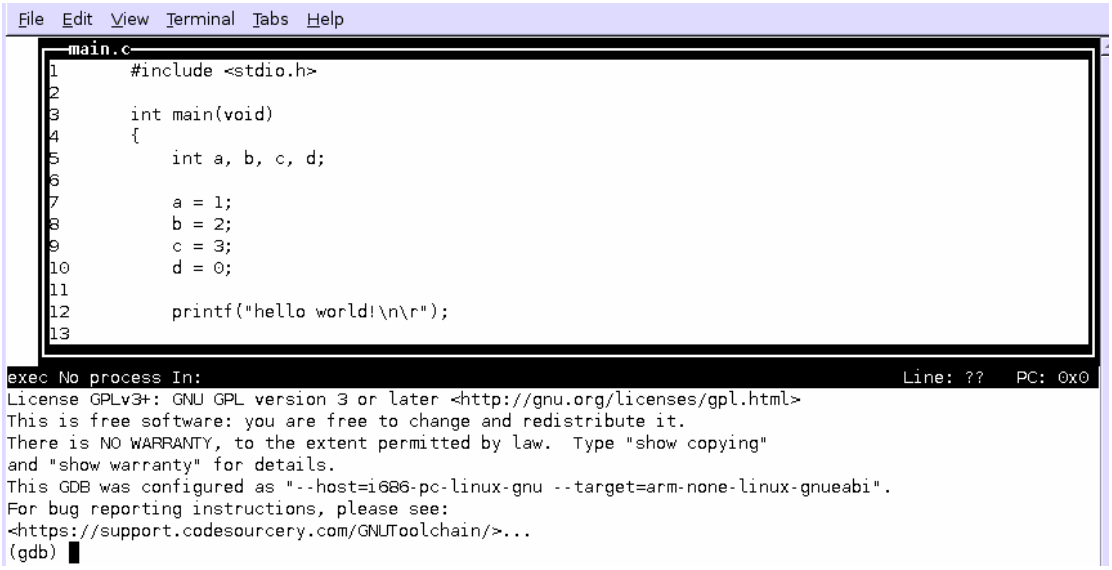
开发板上的 gdbserver 也有相应提示：

```
result=41731601
readchar: Got EOF
Remote side has terminated connection.  GDBserver will reopen the connection.
Listening on port 2345
```

在命令行下，也可以使用 arm-none-linux-gnueabi-gdbtui，

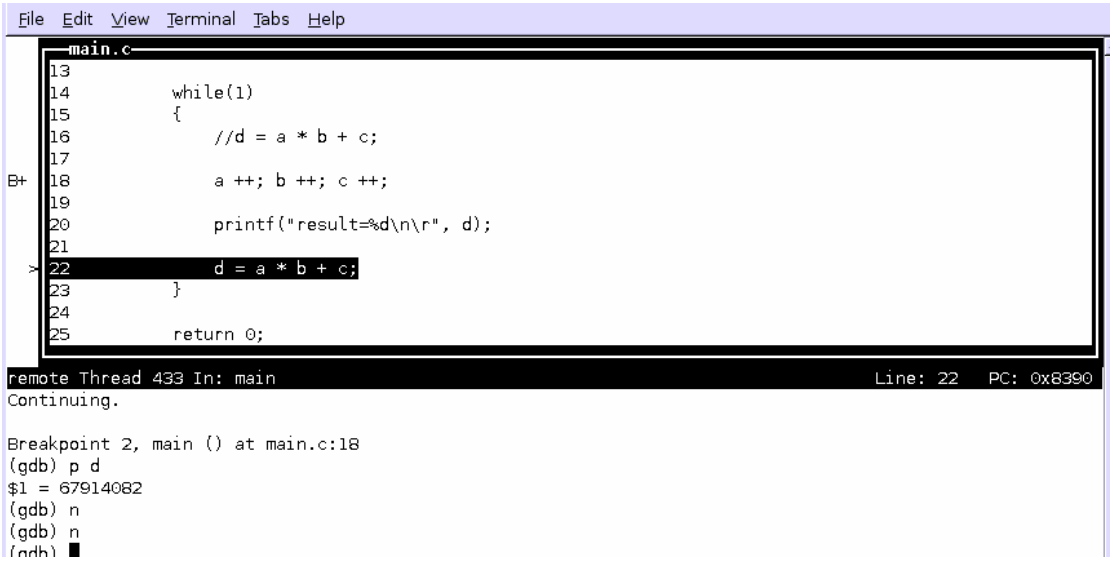
```
root@u810:/usr/work/applinux/hello# which arm-none-linux-gnueabi-gdbtui
/opt/arm-2008q3/bin/arm-none-linux-gnueabi-gdbtui
root@u810:/usr/work/applinux/hello# arm-none-linux-gnueabi-gdbtui dbgtst
```

运行起来后如下图：

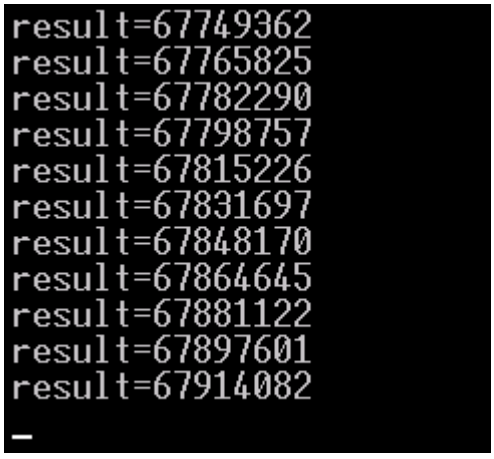


窗口上方会有代码的显示，下面仍然是命令行方式操作。

比纯命令行稍有改进，该方式可以显示代码运行的位置：



调试过程中被调试程序在开发板上也有相应的输出：



调试完成需要断开连接。

四， 使用 DDD

如果觉得命令行界面不是很方便，也可以选择 DDD 这个前端。

首先需要在宿组机上安装 DDD，

```
root@u810:~# apt-get install ddd
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  lesstif2
Suggested packages:
  ddd-doc pydb glibc-doc gnuplot
The following NEW packages will be installed:
  ddd lesstif2
0 upgraded, 2 newly installed, 0 to remove and 286 not upgraded.
Need to get 2078kB of archives.
After this operation, 5579kB of additional disk space will be used.
Do you want to continue [Y/n]?
```

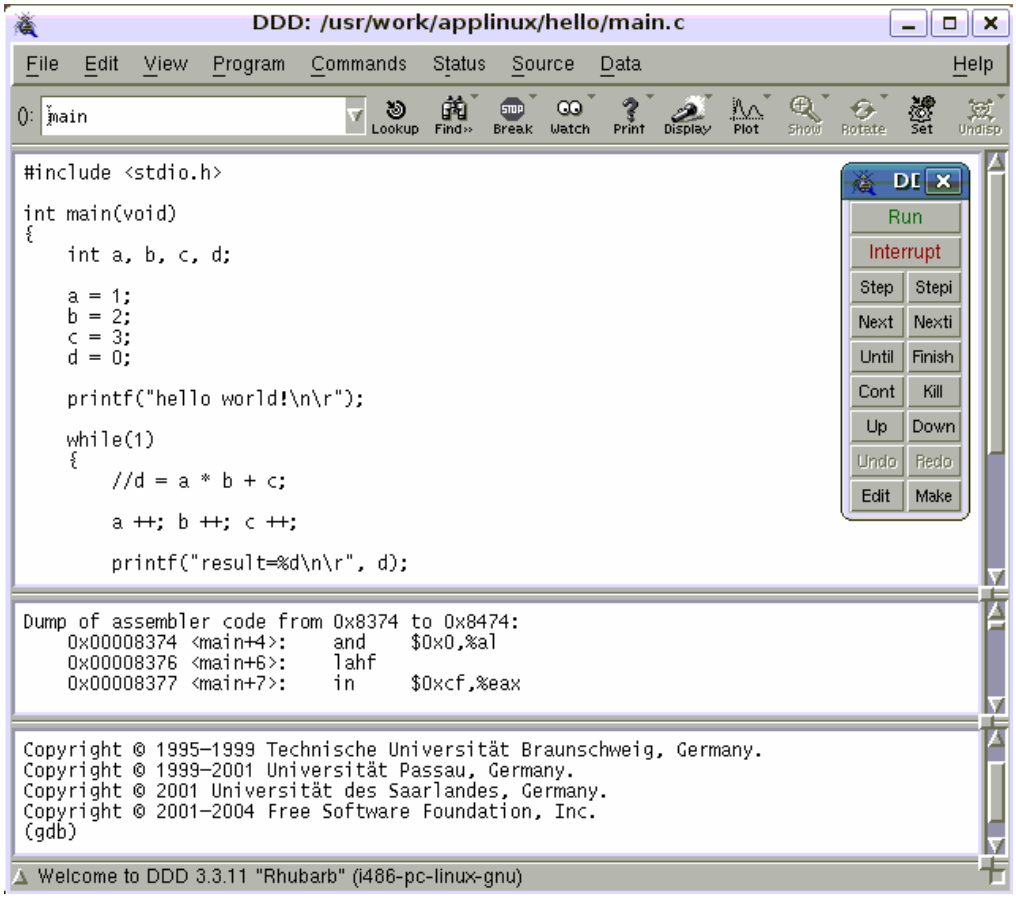
输入 Y 或者直接回车，即可开始从网络安装。

安装完成：

```
root@u810:~# which ddd
/usr/bin/ddd
root@u810:~#
```

输入 ddd 开始运行：

```
root@u810:/usr/work/applinux/hello# ddd -debugger arm-none-linux-gnueabi-gdb dbgtst
```



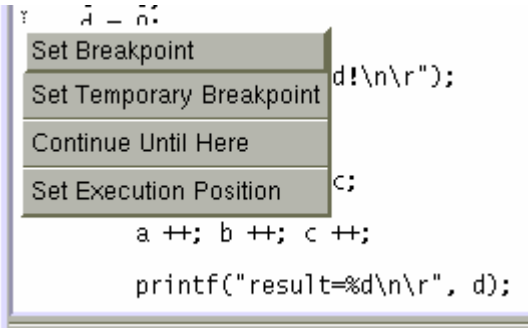
上方显示源代码，最下面是命令行。

在命令行输入：

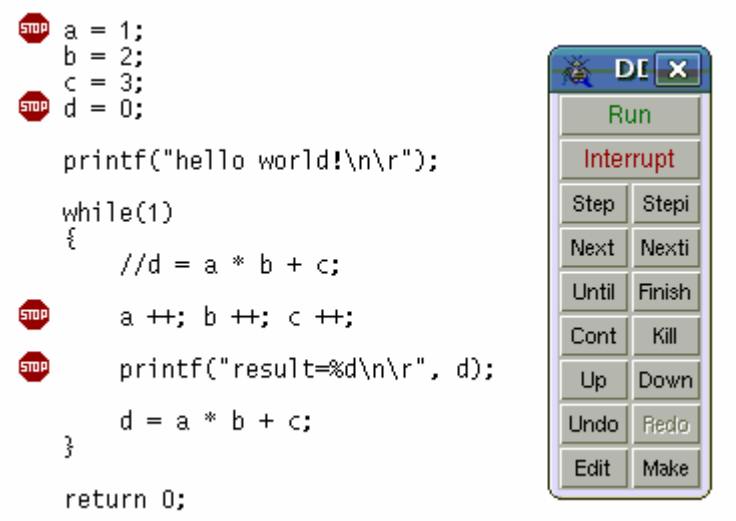
```
Copyright © 2001-2004 Free Software Foundation, Inc.
(gdb) target remote 192.168.1.100:2345 dbgtst
```

连接到开发板上等待连接的 gdbserver。

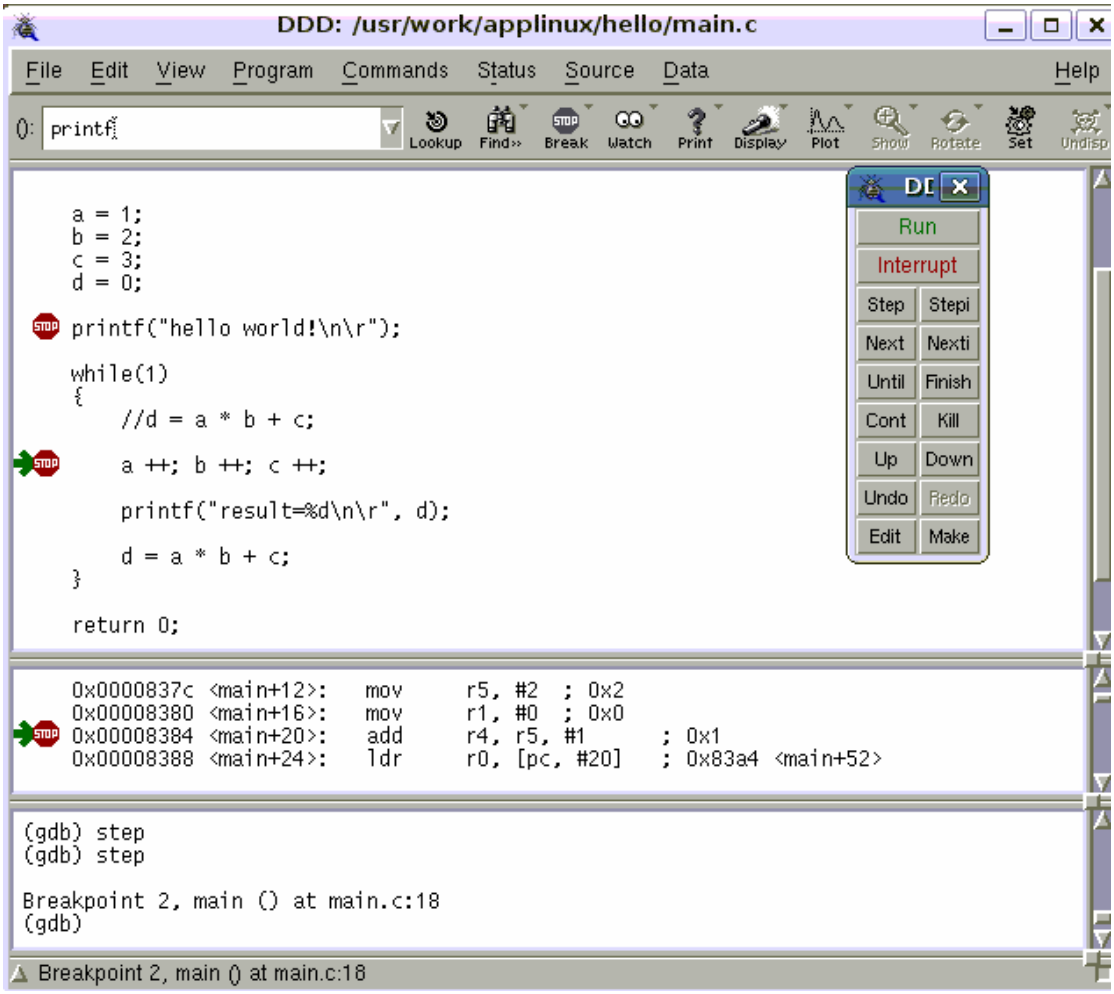
连接上后，可以用图形方式设置断点：



使用命令按钮控制代码运行：



代码运行如下图：



调试完成断开连接。

五， 使用 insight

如果觉得 DDD 的界面仍然不 OK，可以使用 insight。

展开 insight 源码后，使用下面命令配置：

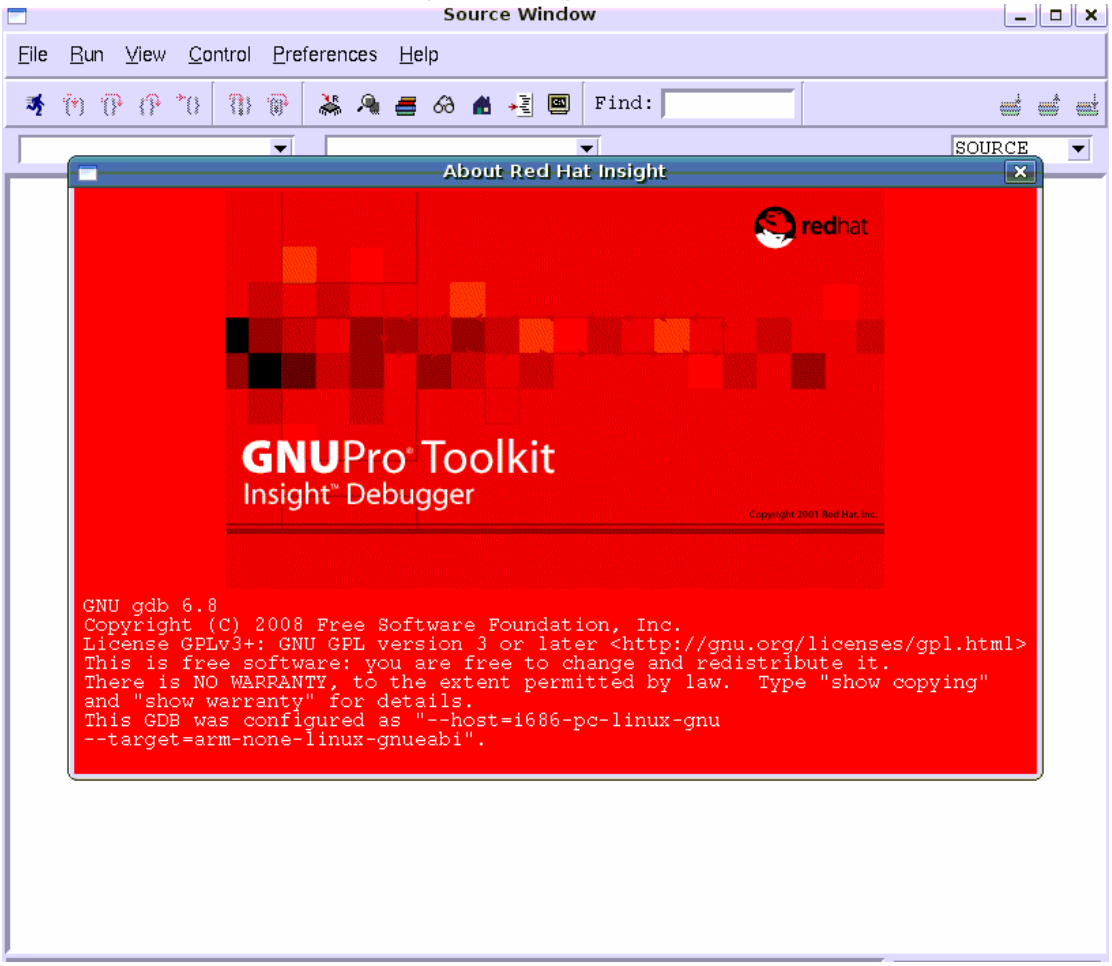
```
root@u810:/usr/work/insight-6.8# ./configure --target=arm-none-linux-gnueabi --enable-sim --prefix=$PWD/_install --disable-werror
```

然后是 make，make install。

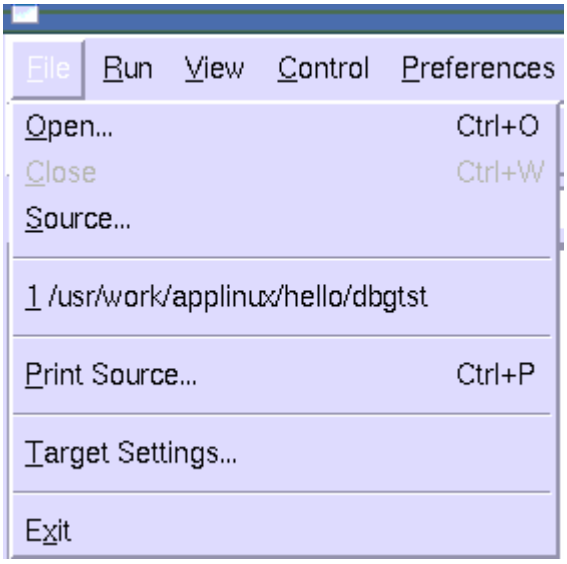
编译获得的可执行文件：

```
root@u810:/usr/work/insight-6.8/_install/bin# ls -al
total 65652
drwxr-xr-x 2 root root     4096 2009-05-01 22:33 .
drwxr-xr-x 8 root root     4096 2009-05-01 22:32 ..
-rwxr-xr-x 1 root root 18587166 2009-05-01 22:33 arm-none-linux-gnueabi-gdb
-rwxr-xr-x 1 root root 18587211 2009-05-01 22:33 arm-none-linux-gnueabi-gdbtui
-rwxr-xr-x 1 root root 18587237 2009-05-01 22:33 arm-none-linux-gnueabi-insight
-rwxr-xr-x 1 root root  2793211 2009-05-01 22:33 arm-none-linux-gnueabi-run
-rwxr-xr-x 1 root root  2717044 2009-05-01 22:32 tclsh8.4
-rwxr-xr-x 1 root root  5848936 2009-05-01 22:33 wish8.4
root@u810:/usr/work/insight-6.8/_install/bin#
```

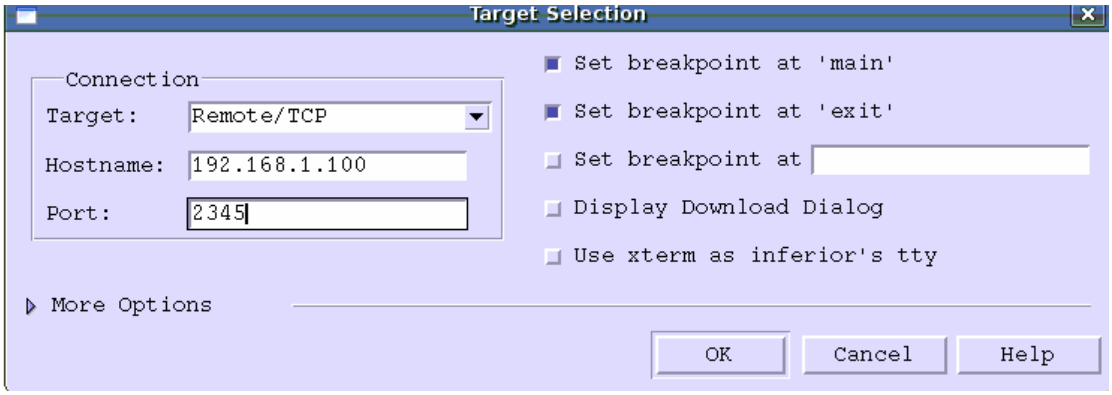
直接运行 arm-none-linux-gnueabi-insight：



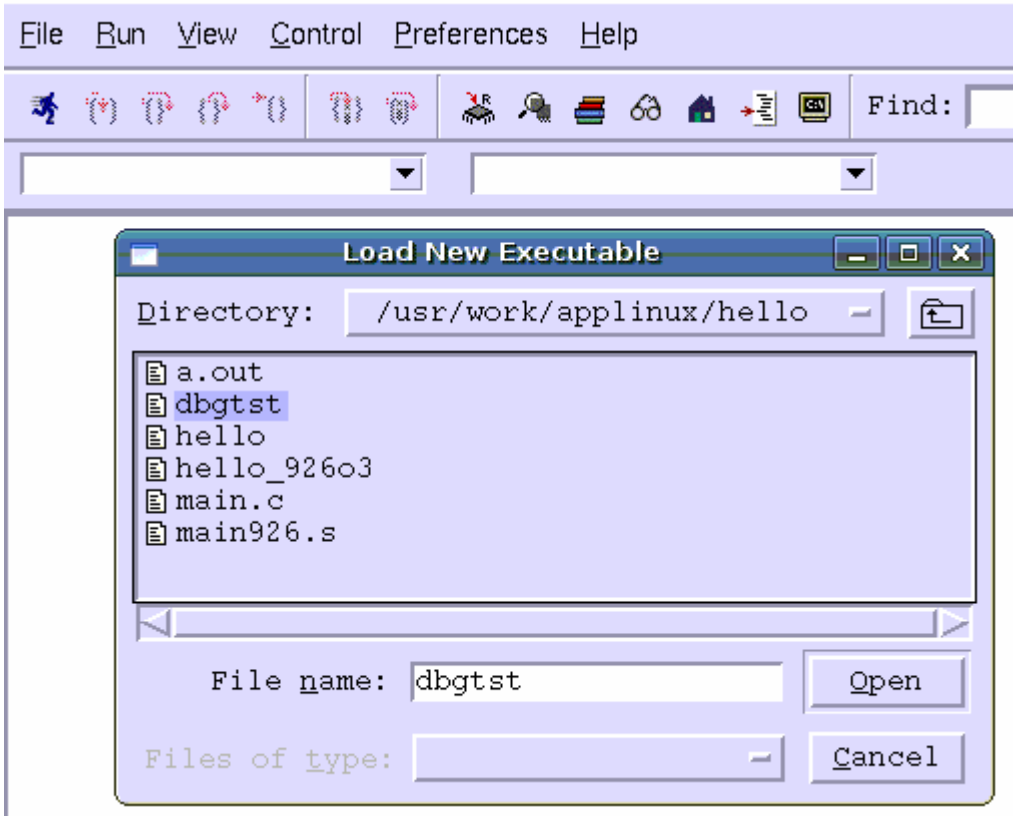
配置 target：



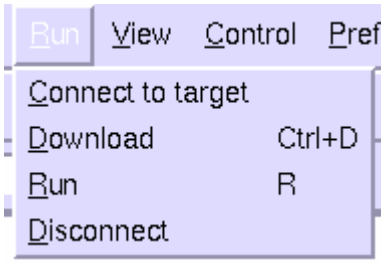
此时需要使得远端的 gdbserver 处于等待连接的状态。



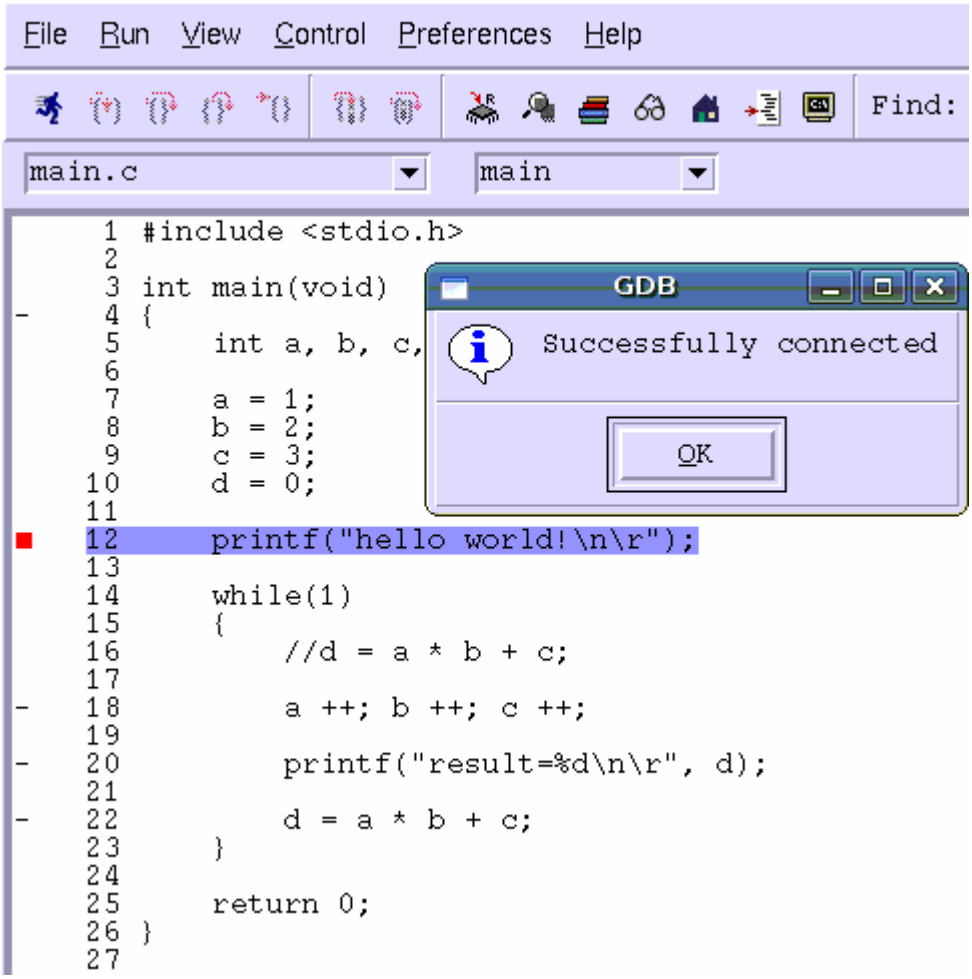
使用 file 下的 open 命令打开调试文件：



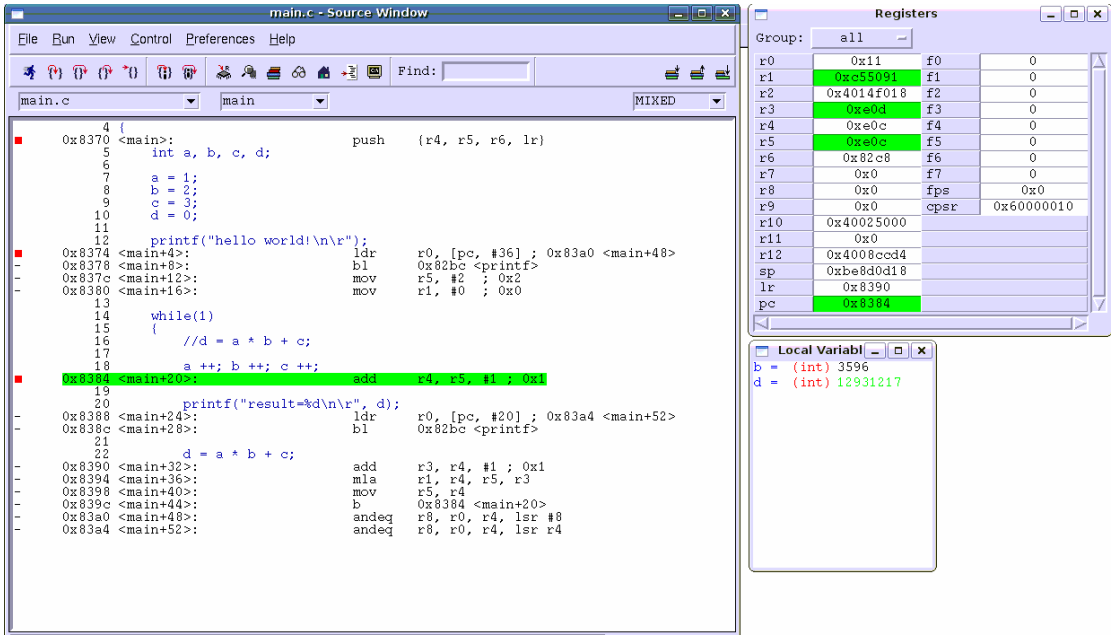
使用 Run 下的命令连接到 target：



成功连接：



点击代码左侧短横线可以设置断点，并可以打开寄存器和变量查看窗口：



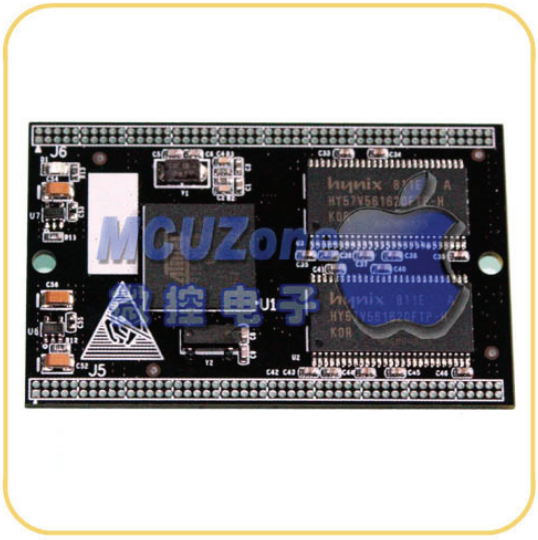
使用工具栏上的调试控制按钮可以控制程序的运行，使用右侧的下拉栏，可以选择代码查看的方式。

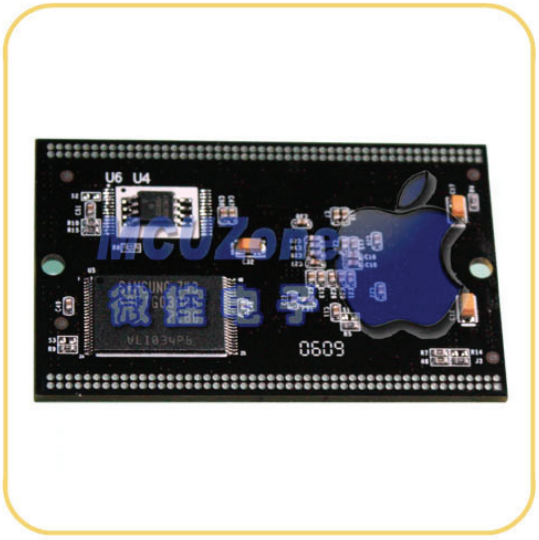
调试完成使用 Run 里的命令断开连接即可。

六， 总结

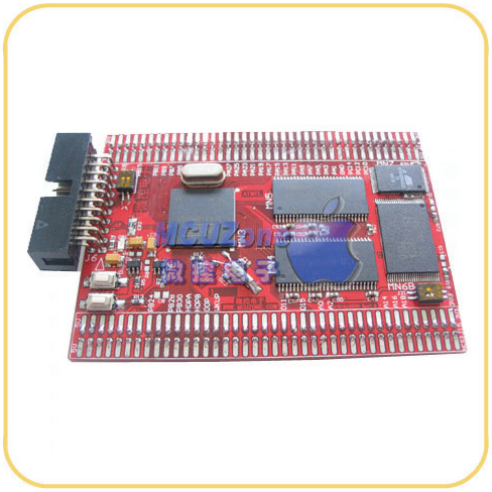
比较上面的几种调试方式，可以看见 insight 的界面最为友好，而 gdb 对系统的依赖最低，无需图形界面的支持。

MDK9261 核心板

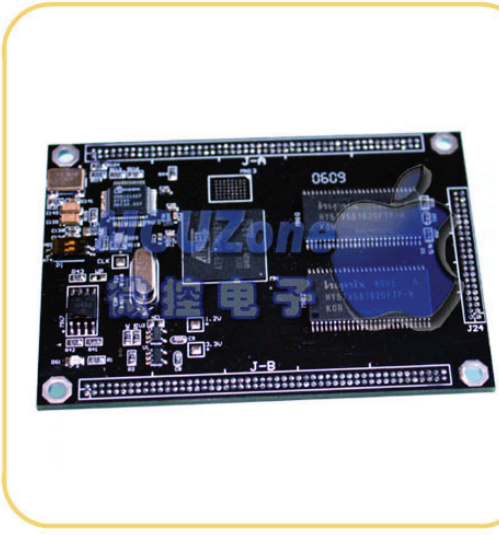




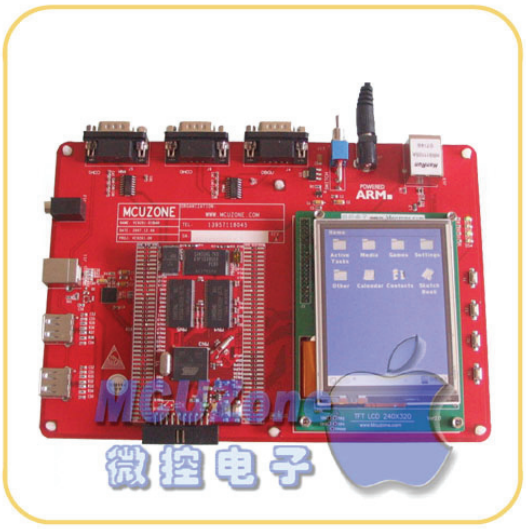
VC9261 核心板



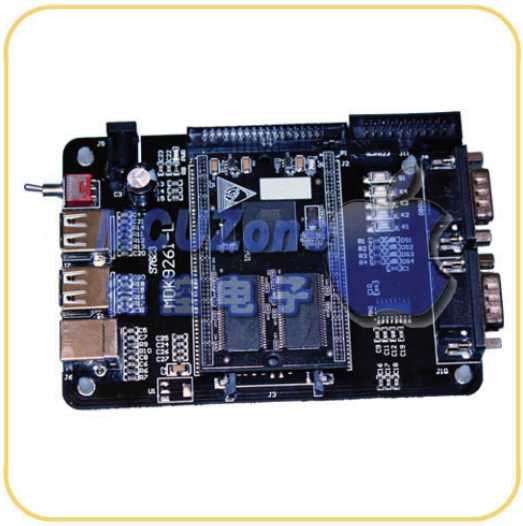
MDK9263 核心板



VC9261-EK 开发板



MDK9261-L 开发板（￥400）



AT9261-EK 开发板