

YL9200 使用手册 V2007

第一章 YL9200 开发板套件介绍	3
1.1 YL9200 开发板简介	3
1.2 硬件资源分配	5
1.3 Linux 操作系统支持的驱动	6
第二章 YL9200 开发板硬件电路说明	8
2.1 电源电路	8
2.2 复位电路	9
2.3 SDRAM 电路	9
2.4 NAND FLASH 电路	10
2.5 LCD 及 VGA 显示电路	11
第三章 YL9200 开发板使用和相关资源测试	13
3.1 启动 Linux	13
3.2 在 BIOS 下测试部分资源	13
3.3 Linux 操作系统下的部分资源测试	22
3.4 Linux 系统下的一些应用	32
3.4.1 NAND FLASH 的分区挂载	32
3.4.2 网络通讯地址设置	33
3.4.3 从 PC 机 TELNET 到开发板	34
3.4.4 从开发板 FTP 到 PC 机	35
3.4.5 从 PC 机 FTP 到开发板	36
3.4.6 在开发板上建立 WEB 服务器	37
第四章 烧写 BIOS 及 BIOS 的相关说明	39
4.1 下载运行 BIOS	39
4.2 利用 BIOSBOX 将程序烧写到 Nor Flash	42
4.3 BIOS 的功能说明	46
第五章 烧写和启动 Linux	52
5.1 通过 BIOS 烧写 Linux 内核	52
5.2 通过 BIOS 烧写根文件系统	55
5.2.1 不带 QT 图形界面的根文件系统 myroot.cramfs 的烧写	55
5.2.2 带 QT 图形界面的根文件系统 myqt.cramfs 的烧写	58
5.3 设置 Linux 自启动	61

第六章 Linux 内核的编译及根文件系统的制作	63
6.1 建立交叉编译环境及编译 Linux 内核	63
6.2 制作 cramfs 根文件系统	71
6.3 编译一个 HELLO 的 DEMO 程序.....	73
6.4 利用串口的 ZMODEM 协议进行文件下载.....	74
第七章 YL9200 开发平台出厂设置	81
7.1.标配三星 3.5 寸屏用户	81
7.2.非标配夏普 3.5 寸屏用户	81
7.3.非标配元太 6.4 寸屏（夏普 8 寸屏、夏普 10.4 屏）用户	82
7.4.输出 VGA 显示用户.....	82
附录一 YL9200 开发系统 Qt 嵌入式图形开发	84
基础篇	84
1. 认识 Qt/Embedded 嵌入式工具开发包	85
入门篇	91
1. Qt/Embedded 开发环境的安装	91
2.认识 Qt/Embedded 开发环境	94
实战篇	127
1.嵌入式硬件开发平台的选择	129
2. 安装 Qt/Embedded 工具开发包	131
3. 交叉编译 Qt/Embedded 的库	131
4. 一个 Hello,World 的例子.....	133
5. 发布一个 Qt/Embedded 应用到 YL9200.....	145
6. 添加一个 Qt/Embedded 应用到 QPE.....	152

第一章 YL9200 开发板套件介绍

1.1 YL9200 开发板简介

YL9200 是一款 ARM920T 内核的工业级的开发板，CPU 内嵌 100M 以太网，带有 USB2.0 协议的 USB HOST 和 Device 接口，支持 SD 卡、IIS 音频和全功能 9 线串口等。主频 180MHz，带有 MMU 存储器管理单元。性能稳定，功能强大，是工业控制、网络通讯等应用的首选。

YL9200 V2.00 开发板硬件资源:

中央处理器

—— AT91RM9200 (ARM920T) 主频 180M(可稳定超频到 240MHZ)，工业级；

外部存储器

—— 64M Bytes NAND Flash (用户可自己更换为 16M、64M 或 128M 的 NandFlash)；

—— 64M Bytes SDRAM (2 片 16 位的 SDRAM 芯片组成 32 位接口)；

—— 16M Bytes Nor Flash (一片 intel E28F128)；

2D 图形加速器

—— 外部扩展的 2D 图形加速引擎，具有独立的 2M 字节显示存储器；

—— 一个标准 VGA 接口，最大支持 16BPP 模式 800×600 分辨率；

—— 一个 TFT 液晶屏接口，最大支持 16BPP 模式 800×600 分辨率；

串口

—— 一个串口为标准三线 RS232 接口；

—— 另一个串口为标准 9 线 RS232 接口，可接 Modem；

RS485 接口

—— 一个 RS485 通讯接口；

网口

—— 一个 100M 网口(AT91RM9200 内部 MAC+外部 PHY)，带发送、接收和联结指示灯；

USB 接口

—— 一个 USB HOST (USB 2.0 Full Speed) 接口;

—— 一个 USB Device (USB 2.0 Full Speed) 接口;

CAN 总线接口

—— 一个 CAN 总线接口, 全面支持 CAN2.0A 和 CAN2.0B 协议;

音频接口

—— 采用 IIS 专用接口芯片, 一路立体声音频输出接口可接耳机或音箱;

IDE 接口

—— 40 芯标准连接器, 可挂载 IDE 硬盘, 板上自带标准 IDE 电源接口;

存储卡接口

—— 一个 MMC/SD 卡接口, 可支持 256M 的 MMC 卡;

调试和下载接口

—— 一个 20 芯 Multi-ICE 标准 JTAG 接口, 支持 SDT2.51, ADS1.2 等调试 ;

其他

—— RTC 实时时钟;

—— 一个带功率驱动的蜂鸣器;

—— 四个接在 GPIO 口线上的高亮 LED;

—— 四个接在 GPIO 口线上的小按键;

—— 一个 10PIN 标准连接器, 可接 4×5 矩阵键盘;

—— 一个复位按键, 采用专用芯片进行复位;

—— 一个 50 芯 2 毫米间距用户扩展口, 引出了地址线、数据线、读写、片选、中断、IO 口、5V 和 3.3V 电源、地等用户扩展可能用到的信号;

电源接口

—— 开关电源供电, 输入直流电压范围是 7~15V (推荐使用 12V, 这样 IDE 硬盘可共用电源), 带电源指示灯;

BootLoader

—— 预装 BIOSBOX (在 Nor FLASH 中);

操作系统

—— 预装 linux2.6.13 操作系统;

YL9200 的特点:

- 100M 网口,采用了 RJ45+网络变压器一体化的网口连接器,降低了干扰;
- 通过串口与网口方式下载程序,调试方便;
- 带有一个可接 Modem 的 9 针串口;
- USB HOST 接口可支持 U 盘等 USB 设备;
- 操作系统: linux 2.6.13,支持 jffs2, cramfs 以及 fat 文件系统

用户光盘上提供的开发工具和源代码:

- 1) ADS1.20 安装程序(评估版);
- 2) 板上所有硬件的 demo 演示程序,全部提供源代码。
- 3) Linux_for AT91RM9200 的内核源码包和编译器;
- 4) YL9200 核心板和底板电路原理图 (pdf 格式);
- 5) YL9200 开发板使用手册 (pdf 格式);
- 6) 开发板上所用到的部分芯片手册、资料;

YL9200 套件包括:

- 1) 一块已测试好的 YL9200 开发板
- 2) 一张 YL9200 光盘
- 3) 一条串口线(两边都是母头,交叉串口线)
- 4) 一条网线(交叉网线)
- 5) 一个+12V 直流电源
- 6) 一个包装盒

1.2 硬件资源分配

(1) 跳线说明

跳线名称	说明
JP3	串口 0 引出线
JP6	选择进行串口写烧写还是从外部 NOR FLASH 启动 1+2: 串口烧写 (主要用于烧写 bootloader) 2+3 (EXTER_BOOT) :从外部 NOR FLASH 启动

J9	选择 CAN 总线的匹配电阻，根据用户需要
JP7	决定 CPU 的调试模式，是 JTAG 还是 ICE 模式 1+2 (JTAG): CPU 工作在 JTAG 模式下。 2+3 (ICE): CPU 工作在 MULTIC-ICE 模式下。

(2) 接口说明

接口名称	说明
P1, P2	全功能串口，调试串口
J3	VGA 接口
J2 (LCD)	TFT LCD 接口
J5	IDE 接口
CN2	50 针用户扩展接口
J1	MMC/SD 卡接口
CN3	JTAG 接口
JP2	4*5 键盘扩展口
CON2	音频接口
JP4	RS485 接口
CAN	CAN 接口
CN1	USB HOST 接口
CON1	USB Device 接口
J10	电源开关
J12	电源接口
J16	背光电源接口

1.3 Linux 操作系统支持的驱动

YL9200 开发板提供的嵌入式 Linux 的内核版本是 linux-2.6.13, Linux 下所支持的驱动如下:

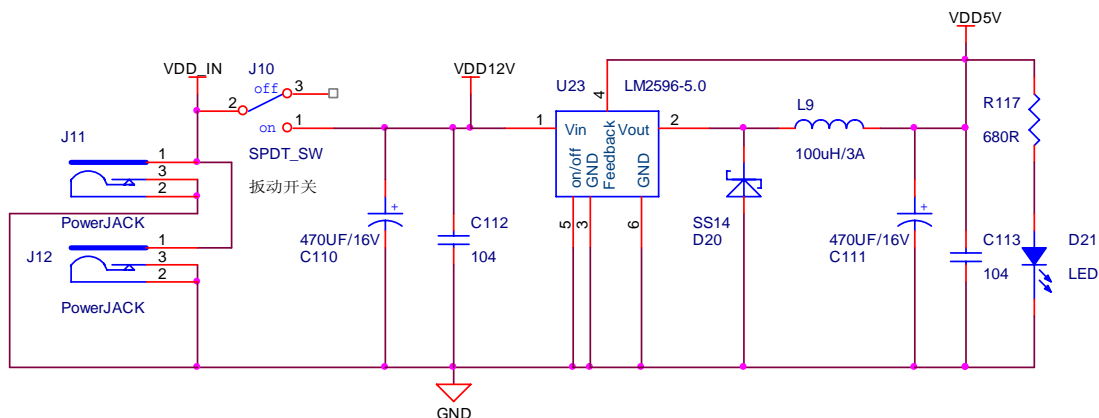
- 串口
- 以太网驱动
- USB HOST

- USB Device
- 音频驱动
- MMC 驱动
- LCD/VGA 驱动
- IDE 驱动
- MTD, (NAND FLASH, NOR FLASH)
- LED 指示灯驱动
- 支持多种文件系统, 如 JFFS2, FAT,CRAMFS 以及 fat 文件系统

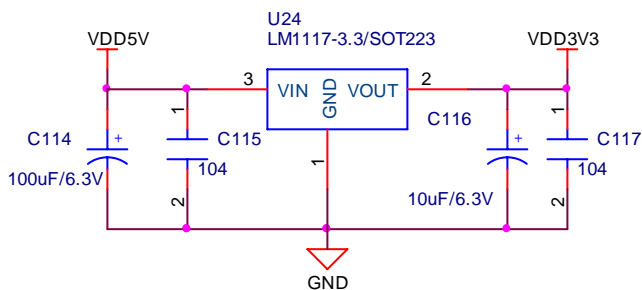
第二章 YL9200 开发板硬件电路说明

2.1 电源电路

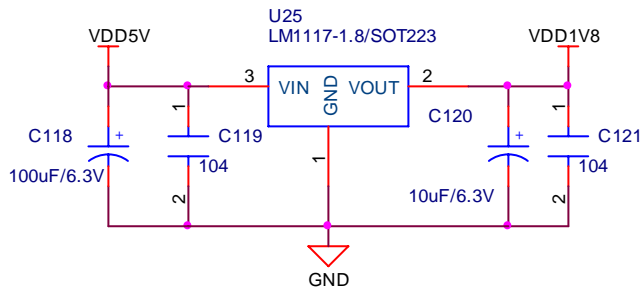
主电源供电电路采用单片开关稳压电源 LM2596-5.0，其最大输出电流可达 3A，具体电路如下：



CPU 及外围 3.3V 器件采用 LM1117-3.3 芯片供电，具体电路如下：

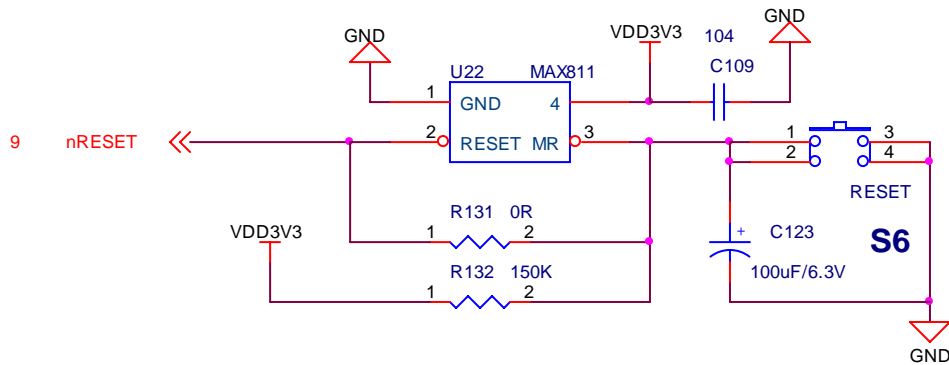


CPU 核心 1.8V 电压采用 LM1117-1.8 进行供电，如下图所示：



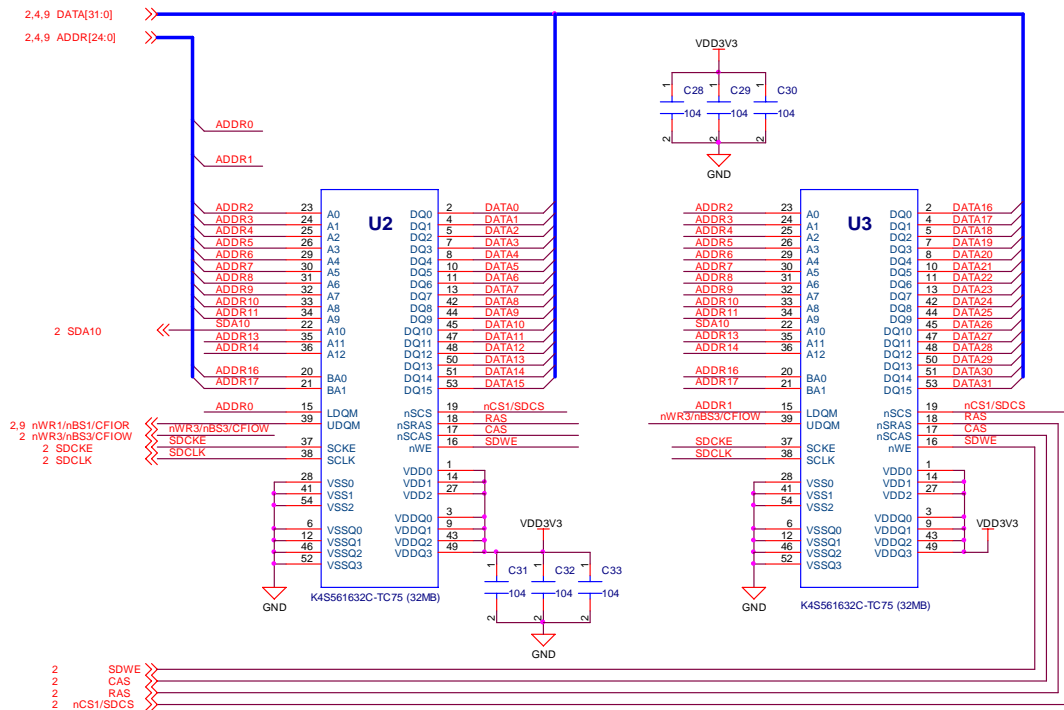
2.2 复位电路

由于 AT91RM9200 的复位时间要求较长，我们采用一个阻容电路来实现上电复位的长复位时间，实际应用过程中可再用专用的复位芯片 MAX811 来实现复位。具体电路如下：（当前开发板上的 MAX811 没有焊接，用 0Ω 电阻短接的）



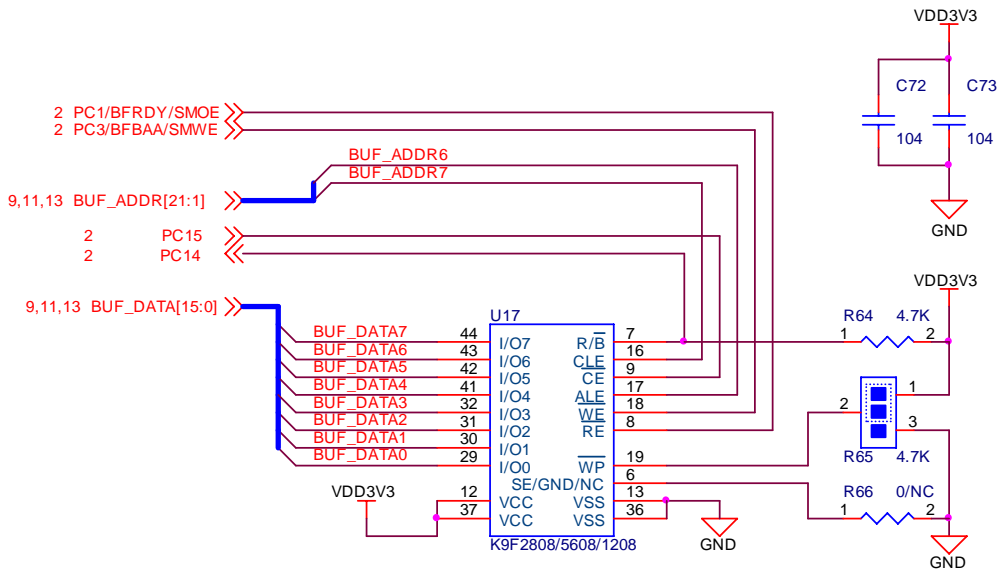
2.3 SDRAM 电路

SDRAM 电路可以使用两片 16M 的 SDRAM 或者是两片 32M 的 SDRAM，具体电路如下：



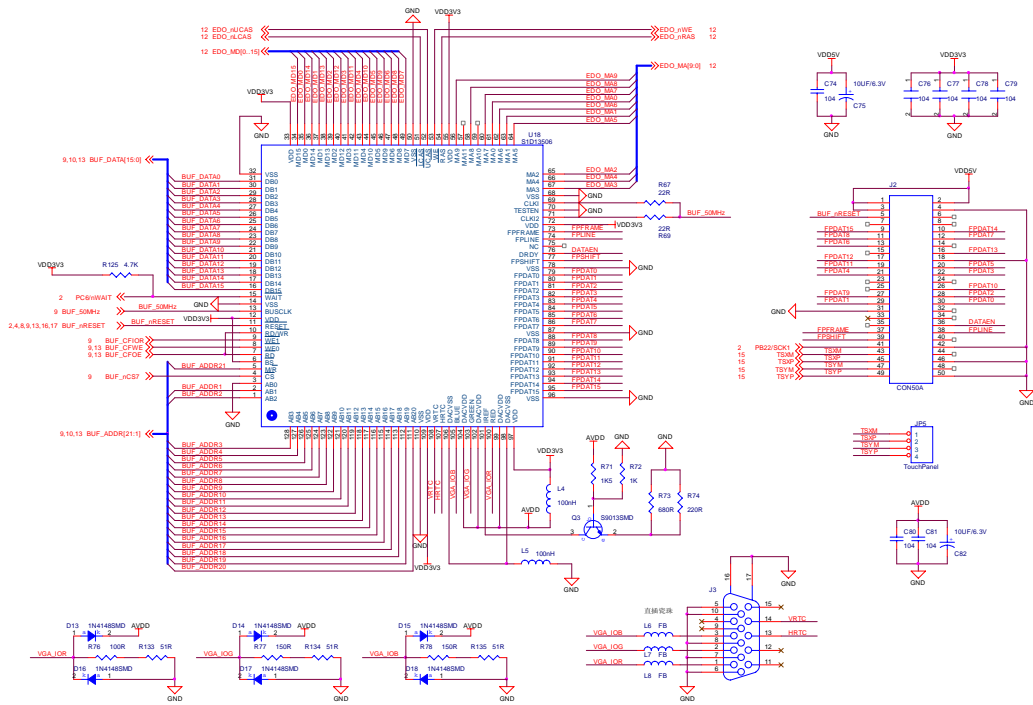
2.4 NAND FLASH 电路

WP#接高电平允许擦除和写入；接低电平禁止擦除和写入。



2.5 LCD 及 VGA 显示电路

由于 AT91RM9200 没有 LCD 控制器,所以我们要使用 LCD 时需要用一个 LCD 控制器完成操作。如下图所示:



其它电路的具体设计请参看光盘中提供的原理图，这里不再多述。

第三章 YL9200 开发板使用和相关资源测试

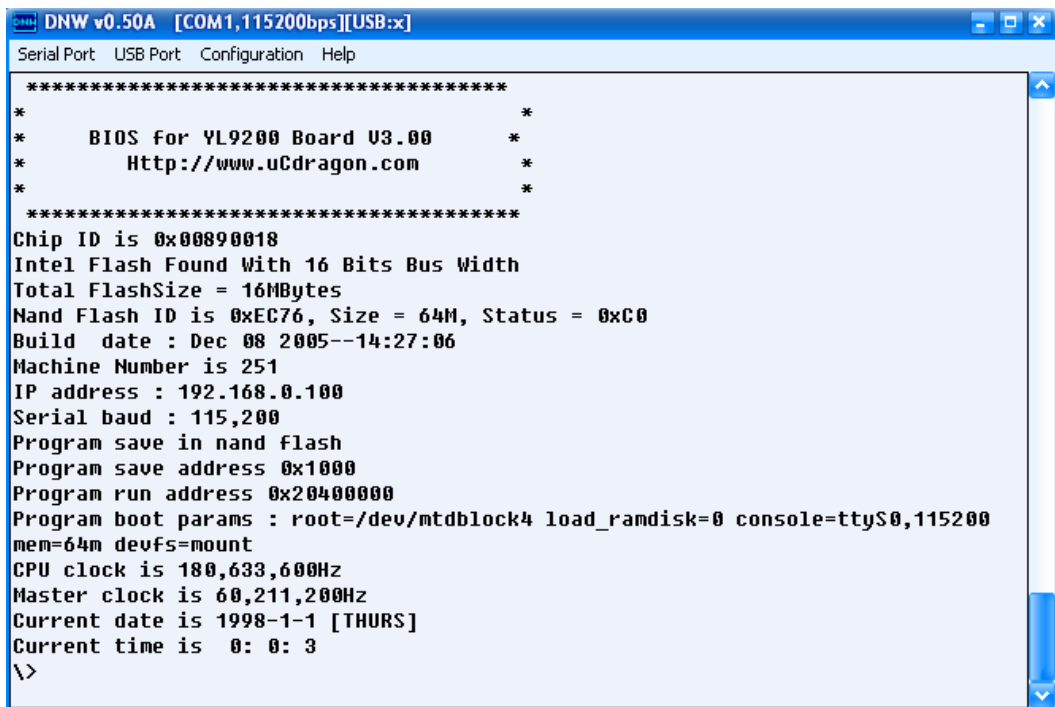
3.1 启动 Linux

在出厂之前，Linux 已经固化在 YL9200 的存储器中了，下面将做好一些准备工作。先用串口线将开发板的串口 P2 与 PC 机的串口连接起来，打开串口工具超级终端或 DNW.exe（在光盘的“实用工具”文件夹下）。

串口工具的参数：波特率 115200，8 位，无奇偶位，停止位 1，无硬件流。接着，接好电源，上电，将启动开发板中的 Linux，这时 D2 LED 指示灯不停的闪烁，并且在超级终端或 DNW.exe 里面有 Linux 启动的相关信息。

3.2 在 BIOS 下测试部分资源

在上电后，在蜂鸣器响过一声后，立即按住开发板上的 S3 按键，将停止装载 Linux，进入 BIOS 的命令状态，如下图所示：**(具体的 BIOS 启动及操作过程请参看第四章 BIOS 烧写及相关说明)**



```
DNW v0.50A [COM1,115200bps][USB:x]
Serial Port  USB Port  Configuration  Help
*****
*
*   BIOS for YL9200 Board V3.00   *
*   Http://www.uCdragon.com     *
*
*****
Chip ID is 0x00890018
Intel Flash Found With 16 Bits Bus Width
Total FlashSize = 16MBytes
Nand Flash ID is 0xEC76, Size = 64M, Status = 0xC0
Build date : Dec 08 2005--14:27:06
Machine Number is 251
IP address : 192.168.0.100
Serial baud : 115,200
Program save in nand flash
Program save address 0x1000
Program run address 0x20400000
Program boot params : root=/dev/mtdblock4 load_ramdisk=0 console=ttyS0,115200
mem=64m devfs=mount
CPU clock is 180,633,600Hz
Master clock is 60,211,200Hz
Current date is 1998-1-1 [THURS]
Current time is 0: 0: 3
\>
```

BIOS 跑起来了，说明串口，蜂鸣器（BIOS 的启动的过程中，蜂鸣器会叫一声），按键以及 LED 指示灯是正常的。

（1）网络测试

BIOS 启动后，可以在 BIOS 的启动信息中看到开发板的 IP address:192.168.0.100 信息，说明开发板的 IP 地址为 192.168.0.100，这时要 PC 的 IP 地址与开发板的 IP 地址要在同一网段，我这里的 PC 机 IP 地址为 192.168.0.7。另外一种方法，是将开发板的 IP 地址设置成与 PC 机的 IP 地址在同一网段（命令 `ipcfg *.*.*.*.*.*`，接着输入命令 `senv` 来保存设置的参数）。

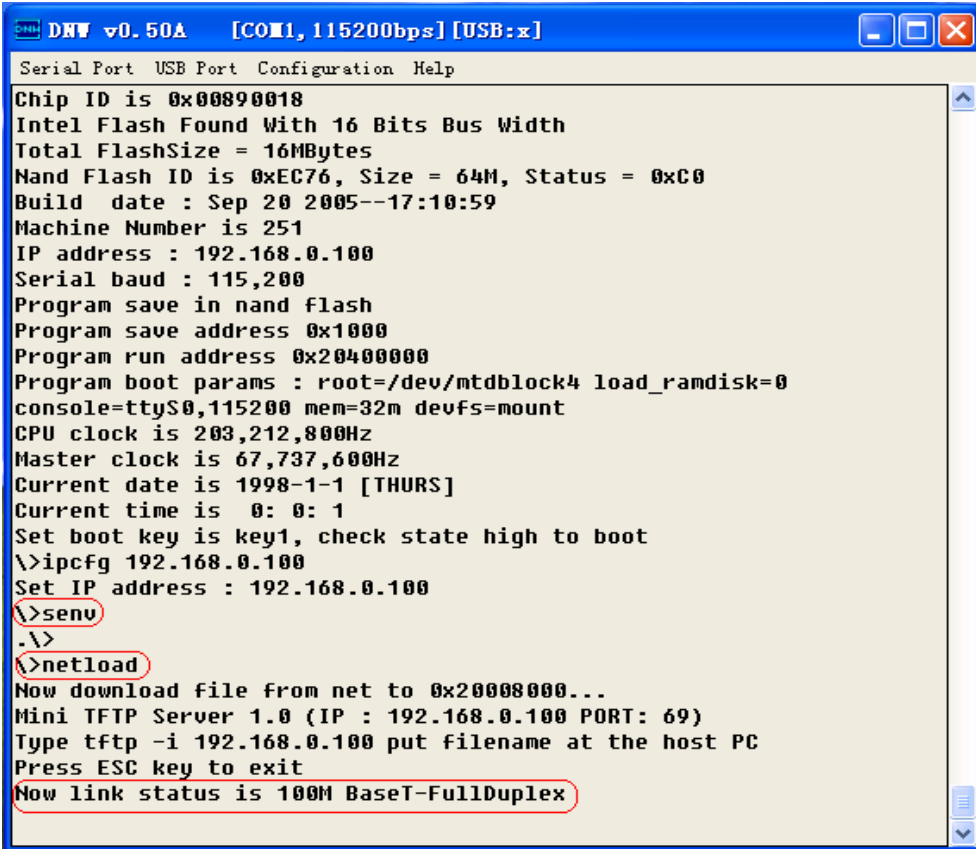
用交叉网线将 PC 的网络接口与开发板的网络接口相连。

IP 地址设置命令：**`ipcfg 192.168.0.100`**

然后输入保存命令：**`senv`**

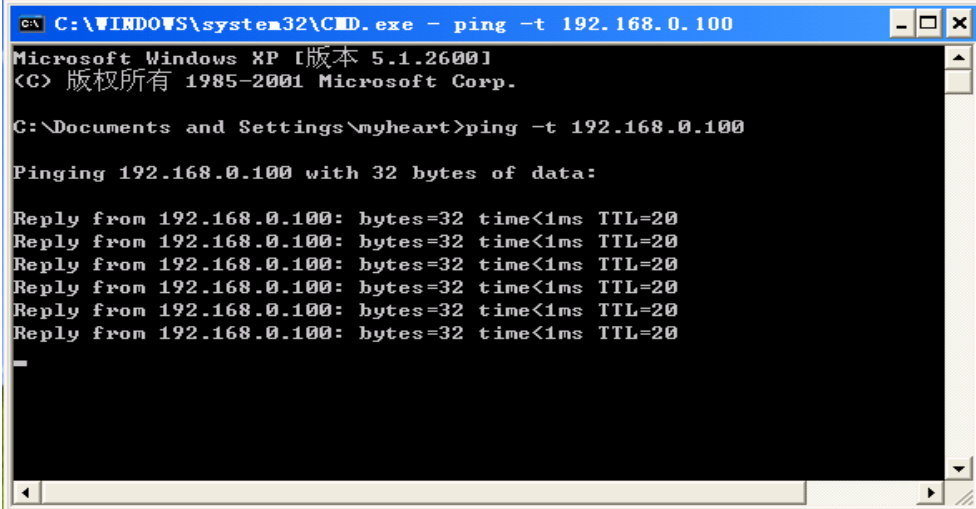
接着输入命令：**`netload`**

界面如下：



```
Serial Port USB Port Configuration Help
Chip ID is 0x00890018
Intel Flash Found With 16 Bits Bus Width
Total FlashSize = 16MBytes
Nand Flash ID is 0xEC76, Size = 64M, Status = 0xC0
Build date : Sep 20 2005--17:10:59
Machine Number is 251
IP address : 192.168.0.100
Serial baud : 115,200
Program save in nand flash
Program save address 0x1000
Program run address 0x20400000
Program boot params : root=/dev/mtdblock4 load_ramdisk=0
console=ttyS0,115200 mem=32m devfs=mount
CPU clock is 203,212,800Hz
Master clock is 67,737,600Hz
Current date is 1998-1-1 [THURS]
Current time is 0: 0: 1
Set boot key is key1, check state high to boot
\>ipcfg 192.168.0.100
Set IP address : 192.168.0.100
\>senv
.\>
\>netload
Now download file from net to 0x20008000...
Mini TFTP Server 1.0 (IP : 192.168.0.100 PORT: 69)
Type tftp -i 192.168.0.100 put filename at the host PC
Press ESC key to exit
Now link status is 100M BaseT-FullDuplex
```

这时，在 PC 机端打开命令窗口，在命令窗口输入 `ping -t 192.168.0.100` 来测试开发板的网络是否是通的，下面是网络成功的信息：



```
C:\WINDOWS\system32\CMD.exe - ping -t 192.168.0.100
Microsoft Windows XP [版本 5.1.2600]
(C) 版权所有 1985-2001 Microsoft Corp.

C:\Documents and Settings\myheart>ping -t 192.168.0.100

Pinging 192.168.0.100 with 32 bytes of data:

Reply from 192.168.0.100: bytes=32 time<1ms TTL=20
Reply from 192.168.0.100: bytes=32 time<1ms TTL=20
Reply from 192.168.0.100: bytes=32 time<1ms TTL=20
Reply from 192.168.0.100: bytes=32 time<1ms TTL=20
Reply from 192.168.0.100: bytes=32 time<1ms TTL=20
Reply from 192.168.0.100: bytes=32 time<1ms TTL=20
-
```

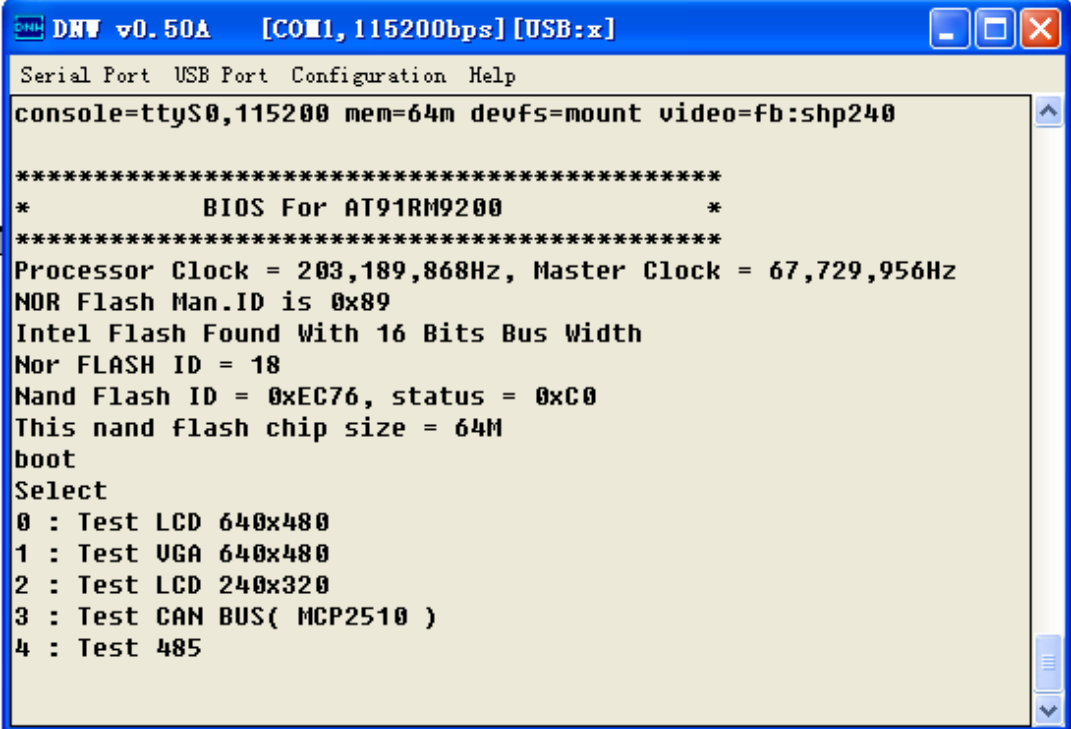
这样，网络接口工作 OK。

(2) VGA 和 CAN 的测试

进入 BIOS 的命令状态，输入命令：

netrun

注意：这个命令是网络下载运行命令，再运行这个命令的时候，首先需要将开发板的 ip 地址设置为 192.168.0.100，同时 PC 机的 IP 地址要与开发板的 IP 地址再同一网段。然后回车。接着点击批处理文件 YL9200_Test.bat \光盘\ 目标代码\YL9200_Test.bat 传输结束后，将自动执行该程序。

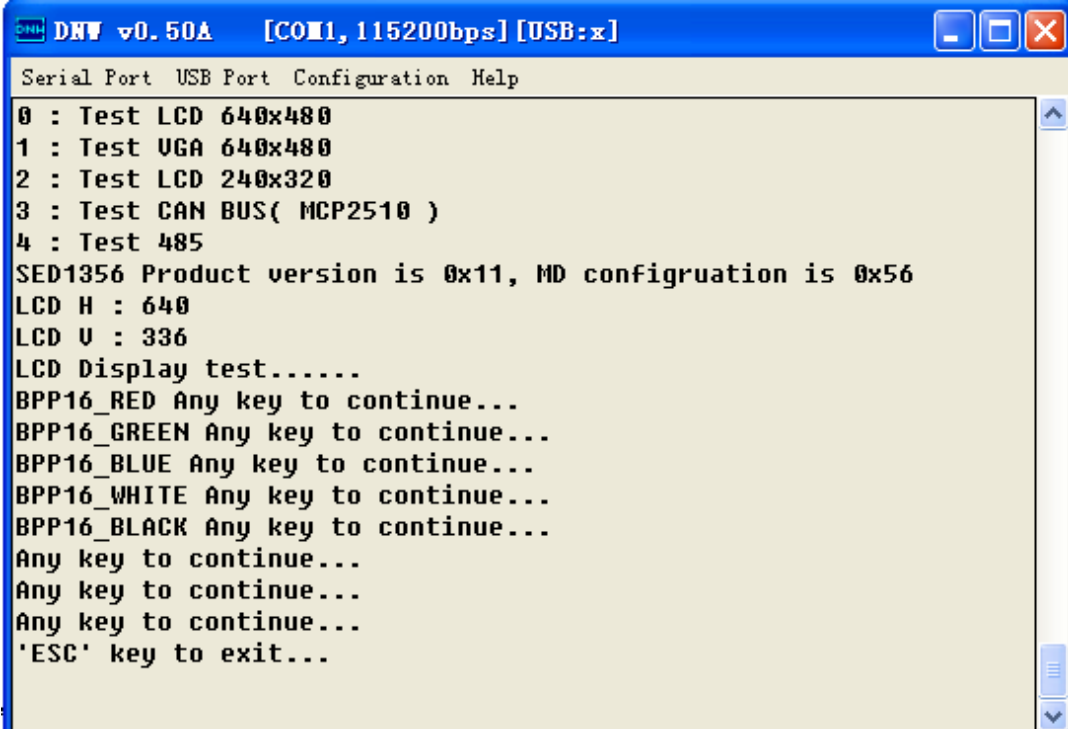


```
Serial Port  USB Port  Configuration  Help
console=ttyS0,115200 mem=64m devfs=mount video=fb:shp240

*****
*                BIOS For AT91RM9200                *
*****
Processor Clock = 203,189,868Hz, Master Clock = 67,729,956Hz
NOR Flash Man.ID is 0x89
Intel Flash Found With 16 Bits Bus Width
Nor FLASH ID = 18
Nand Flash ID = 0xEC76, status = 0xC0
This nand flash chip size = 64M
boot
Select
0 : Test LCD 640x480
1 : Test UGA 640x480
2 : Test LCD 240x320
3 : Test CAN BUS( MCP2510 )
4 : Test 485
```

TFT-LCD 640x480 测试

选择“0”，将进行8寸屏（该8寸LCD屏为选配件，TFT）的LCD测试，接着将8寸LCD用50针排线与YL9200的开发板的J2连接起来，这时按任意键可以看到LCD屏上图片的切换。

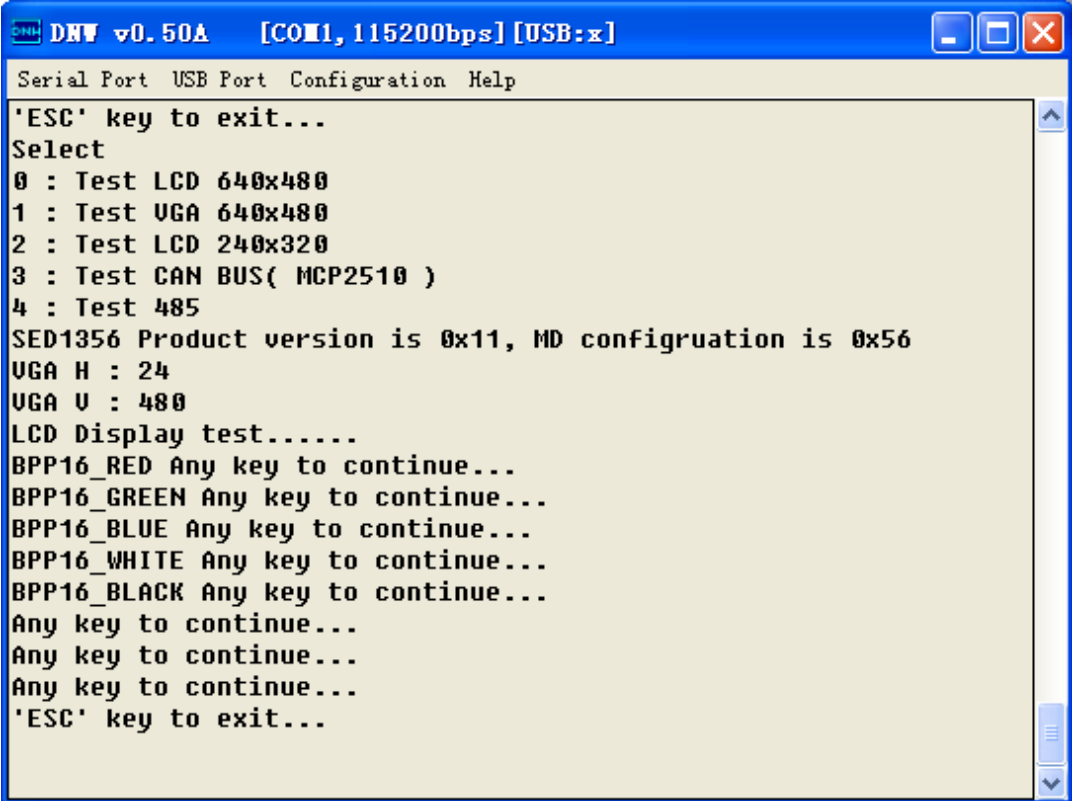


```
Serial Port  USB Port  Configuration  Help
0 : Test LCD 640x480
1 : Test VGA 640x480
2 : Test LCD 240x320
3 : Test CAN BUS( MCP2510 )
4 : Test 485
SED1356 Product version is 0x11, MD configuration is 0x56
LCD H : 640
LCD V : 336
LCD Display test.....
BPP16_RED Any key to continue...
BPP16_GREEN Any key to continue...
BPP16_BLUE Any key to continue...
BPP16_WHITE Any key to continue...
BPP16_BLACK Any key to continue...
Any key to continue...
Any key to continue...
Any key to continue...
'ESC' key to exit...
```

图片显示结束后，按“ESC”键返回主测试菜单。

VGA 640x480 测试

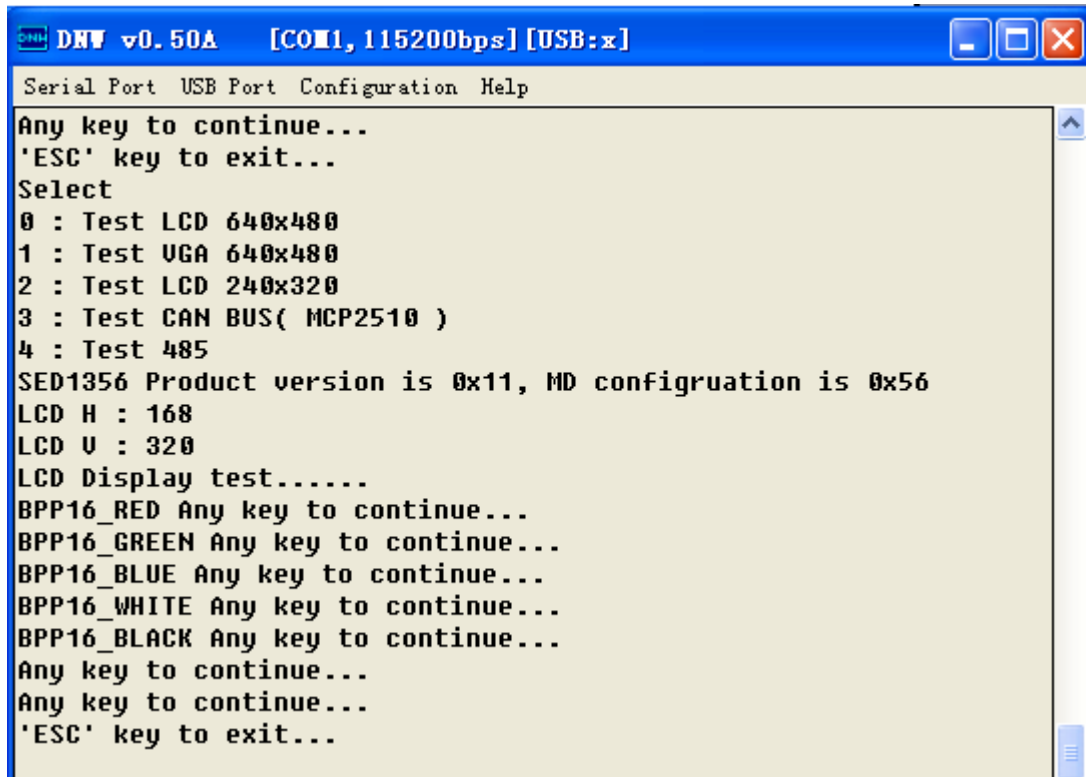
选择“1”，将进行 VGA 测试，这时将 VGA 接口连接好，按任意键将切换 VGA 显示的图形界面。

The image shows a screenshot of a software window titled "DNW v0.50A [COM1, 115200bps] [USB:x]". The window has a blue title bar with standard Windows window controls (minimize, maximize, close). The main content area is a text-based interface with a light beige background. At the top, it says "Serial Port USB Port Configuration Help". Below that, it prompts "'ESC' key to exit...". A "Select" menu is displayed with five options: "0 : Test LCD 640x480", "1 : Test UGA 640x480", "2 : Test LCD 240x320", "3 : Test CAN BUS(MCP2510)", and "4 : Test 485". Below the menu, it shows "SED1356 Product version is 0x11, MD configuration is 0x56". Further down, it displays "UGA H : 24" and "UGA V : 480". A section titled "LCD Display test....." follows, with several prompts: "BPP16_RED Any key to continue...", "BPP16_GREEN Any key to continue...", "BPP16_BLUE Any key to continue...", "BPP16_WHITE Any key to continue...", "BPP16_BLACK Any key to continue...", and "Any key to continue...". The interface ends with "'ESC' key to exit...". A vertical scrollbar is visible on the right side of the text area.

图片显示结束后，按“ESC”键返回主测试菜单。

TFT-LCD 240x320 测试

选择“2”，将进行 TFT LCD 240x320(Sharp 3.5 寸)的测试，这时连接好 3.5 寸的 LCD 屏，接着在 DNW 串口按任意键，可以切换在 LCD 屏上显示的图片，

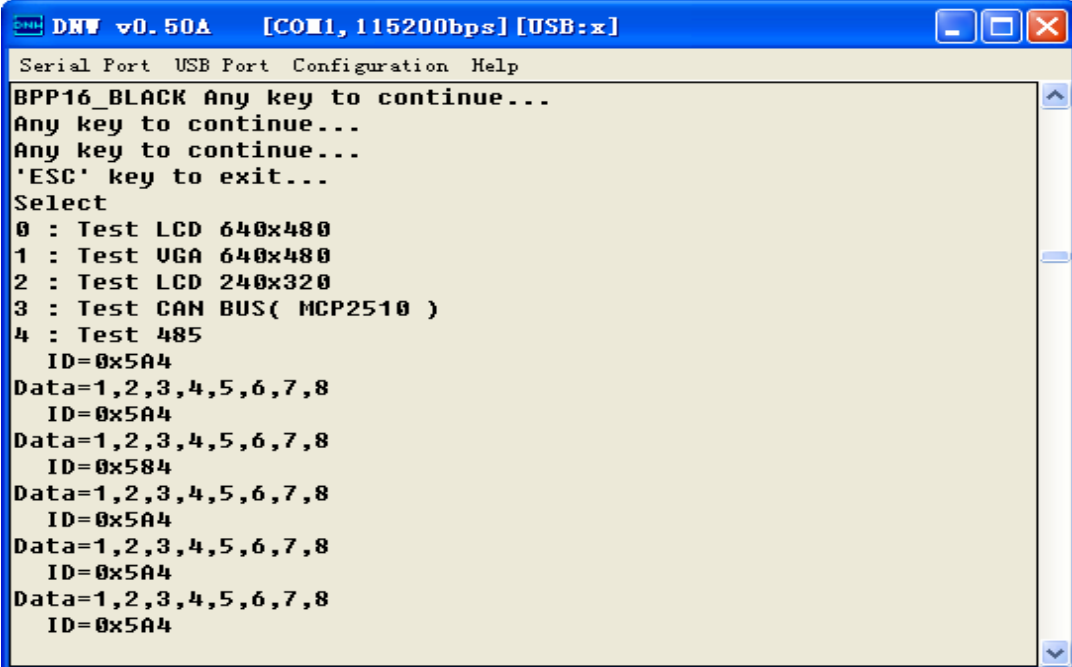
A screenshot of a terminal window titled "DHW v0.50A [COM1, 115200bps] [USB:x]". The window contains a menu of tests and system information. The text displayed is:

```
Serial Port USB Port Configuration Help
Any key to continue...
'ESC' key to exit...
Select
0 : Test LCD 640x480
1 : Test UGA 640x480
2 : Test LCD 240x320
3 : Test CAN BUS( MCP2510 )
4 : Test 485
SED1356 Product version is 0x11, MD configuration is 0x56
LCD H : 168
LCD U : 320
LCD Display test.....
BPP16_RED Any key to continue...
BPP16_GREEN Any key to continue...
BPP16_BLUE Any key to continue...
BPP16_WHITE Any key to continue...
BPP16_BLACK Any key to continue...
Any key to continue...
Any key to continue...
'ESC' key to exit...
```

按“ESC”键可以退出该项测试。

CAN 测试

接着输入“3”，将进行 CAN 总线测试，CAN 测试，主要时让 CAN 工作在回环模式下。



```

DNW v0.50A [COM1, 115200bps] [USB:x]
Serial Port USB Port Configuration Help
BPP16_BLACK Any key to continue...
Any key to continue...
Any key to continue...
'ESC' key to exit...
Select
0 : Test LCD 640x480
1 : Test UGA 640x480
2 : Test LCD 240x320
3 : Test CAN BUS( MCP2510 )
4 : Test 485
   ID=0x5A4
Data=1,2,3,4,5,6,7,8
   ID=0x5A4
Data=1,2,3,4,5,6,7,8
   ID=0x584
Data=1,2,3,4,5,6,7,8
   ID=0x5A4
Data=1,2,3,4,5,6,7,8
   ID=0x5A4
Data=1,2,3,4,5,6,7,8
   ID=0x5A4

```

可以按“ESC”键退出。

RS485 测试

返回主菜单后，接着可以进行 485 测试了

在进行 485 测试前，请用 485 转 232 的转换器，将开发板的 485 接口与一条直连网线连接起。在下面的测试，就是这样前提下进行的。

选择“4”，接着将串口线切换到 485 转换器，然后连接 485 接口，我这里对应的仍是 COM1。（波特率 115200，8 位，无奇偶位，停止位 1，无硬件流）

在 COM1 所对应的 DNW 工具中，输入任意字符，这时将会在本 DNW 中显示输入的字符。

显示信息如下：



```

3 : Test CAN BUS( MCP2510 )
4 : Test 485
RS485 Test
Please switch to RS485 connect
'ESC' key to exit...
fdsyrewyauqdzcbngcjknuh;l9i;ijg,.jhbvkfytiu35q36tyehxfhnbcbjngjhoi78oi6dkfgjnnfd
zd53q6534yhgdrxfgj542452gjcgyufg8iuytkygcknghjghdjtrdjn vbSelect

```

3.3 Linux 操作系统下的部分资源测试

在 BIOS 状态下，可以通过命令 `mrunc` 来启动 Linux,另一种方法是让 Linux 自启动。

进入 Linux 状态，可以看到 QT 图形界面，同时可以进行 USB HOST 测试，USB Device 测试，MMC 测试，音频测试，网络测试以及 IDE 硬盘挂接测试。

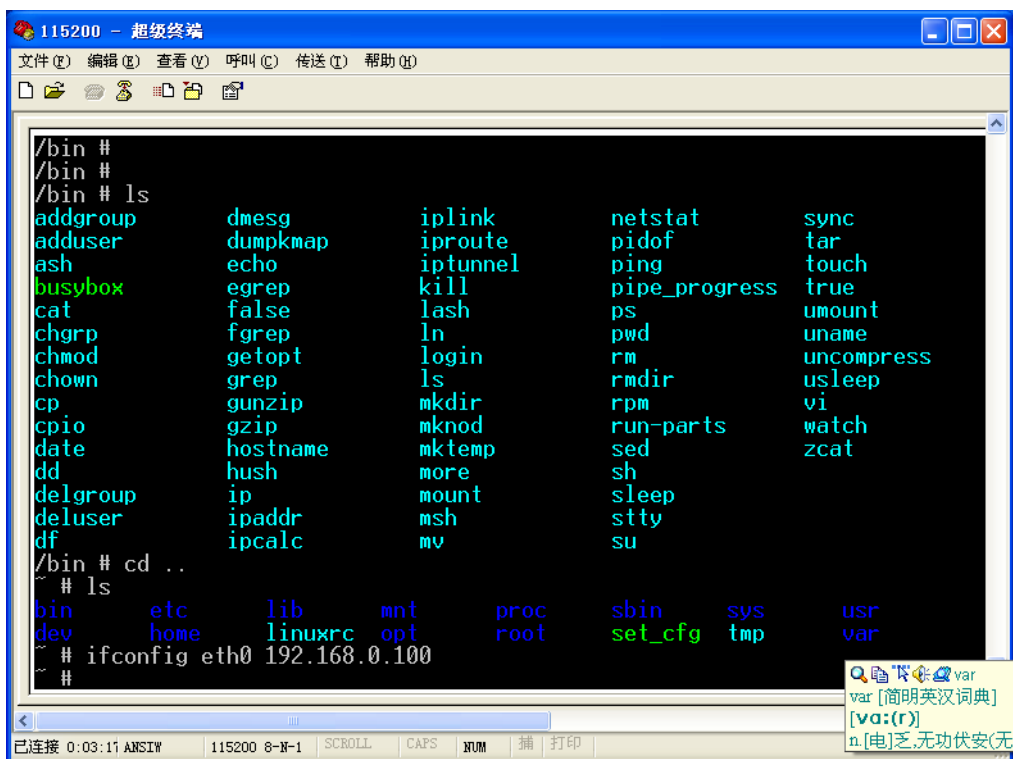
注意在进入开发板的 Linux 环境后，将串口工具换成超级终端。

(1) 网络测试

首先用交叉网线将板子的网络接口与 PC 机的接口连接起来。

必须设置好开发板的 IP 地址（IP 地址必须与 PC 机的地址一致），设置命令如下：

```
ifconfig eth0 192.168.0.100
```

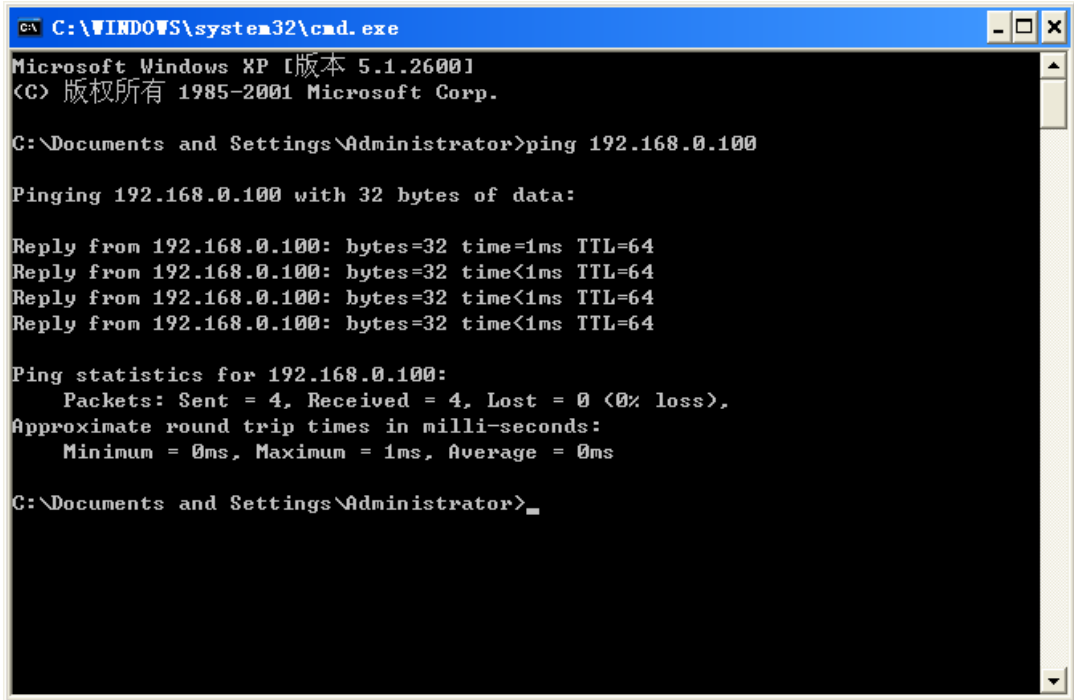


The screenshot shows a terminal window titled "115200 - 超级终端". The terminal output is as follows:

```
/bin #  
/bin #  
/bin # ls  
addgroup          dmesg             iplink            netstat           sync  
adduser           dumpkmap          iptunnel          pidof             tar  
ash               echo              iptunnel          ping             touch  
busybox           grep              kill              pipe_progress    true  
cat               false             lash              ps                umount  
chgrp             fgrep             ln                pwd               uname  
chmod             getopt            login             rm                uncompress  
chown             grep              ls                rmdir            usleep  
cp                gunzip            mkdir             rpm               vi  
cpio              gzip              mknod             run-parts         watch  
date              hostname           mktemp            sed                zcat  
dd                hush              more              sh  
delgroup          ip                mount             sleep  
deluser           ipaddr            msh  
df                ipcalc            mv                su  
/bin # cd ..  
~ # ls  
bin          etc          lib          mnt          proc         sbin         sys          usr  
dev          home        linuxrc     opt          root         set_cfg     tmp          var  
~ # ifconfig eth0 192.168.0.100  
~ #
```

The terminal window also shows a status bar at the bottom with the text "已连接 0:03:11 ANSII 115200 8-N-1 | SCROLL CAPS NUM 捕 打印".

这时在 PC 机端用 `ping -t 192.168.0.100` 命令来测试 Linux 下是否好的。



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [版本 5.1.2600]
(C) 版权所有 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator>ping 192.168.0.100

Pinging 192.168.0.100 with 32 bytes of data:

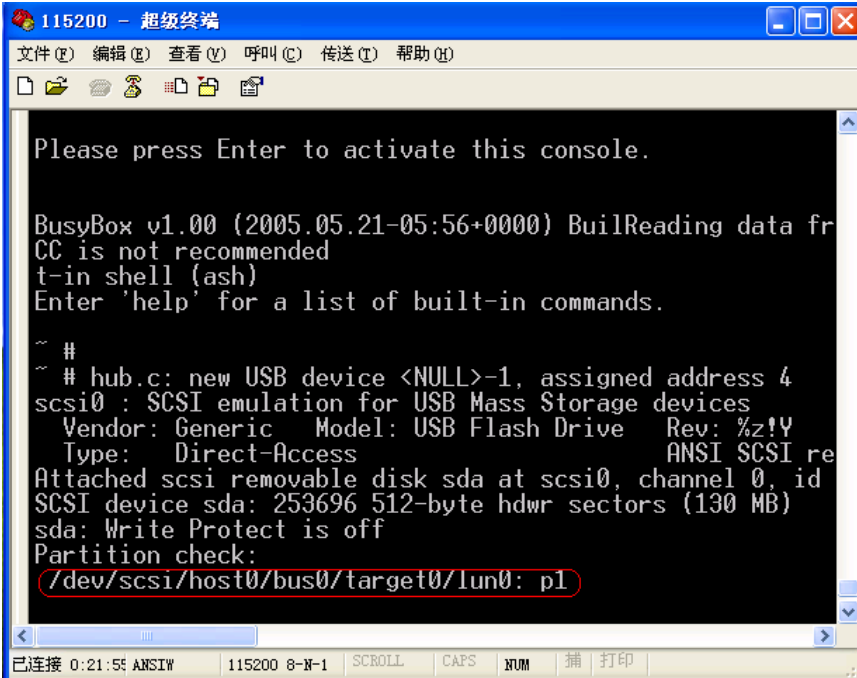
Reply from 192.168.0.100: bytes=32 time=1ms TTL=64
Reply from 192.168.0.100: bytes=32 time<1ms TTL=64
Reply from 192.168.0.100: bytes=32 time<1ms TTL=64
Reply from 192.168.0.100: bytes=32 time<1ms TTL=64

Ping statistics for 192.168.0.100:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 1ms, Average = 0ms

C:\Documents and Settings\Administrator>_
```

(2) USB HOST 测试

Linux 启动后，将 U 盘插入开发板的 USB 接口，如果 Linux 检测到 U 盘，将出现如下信息：



```
115200 - 超级终端
文件(F) 编辑(E) 查看(V) 呼叫(C) 传送(T) 帮助(H)
Please press Enter to activate this console.

BusyBox v1.00 (2005.05.21-05:56+0000) Built-in shell (ash)
CC is not recommended
t-in shell (ash)
Enter 'help' for a list of built-in commands.

~ #
~ # hub.c: new USB device <NULL>-1, assigned address 4
scsi0 : SCSI emulation for USB Mass Storage devices
  Vendor: Generic  Model: USB Flash Drive  Rev: %z!Y
  Type:   Direct-Access          ANSI SCSI re
Attached scsi removable disk sda at scsi0, channel 0, id
SCSI device sda: 253696 512-byte hdwr sectors (130 MB)
sda: Write Protect is off
Partition check:
/dev/scsi/host0/bus0/target0/lun0: p1

已连接 0:21:56 ANSIW 115200 8-N-1 SCROLL CAPS NUM 捕 打印
```

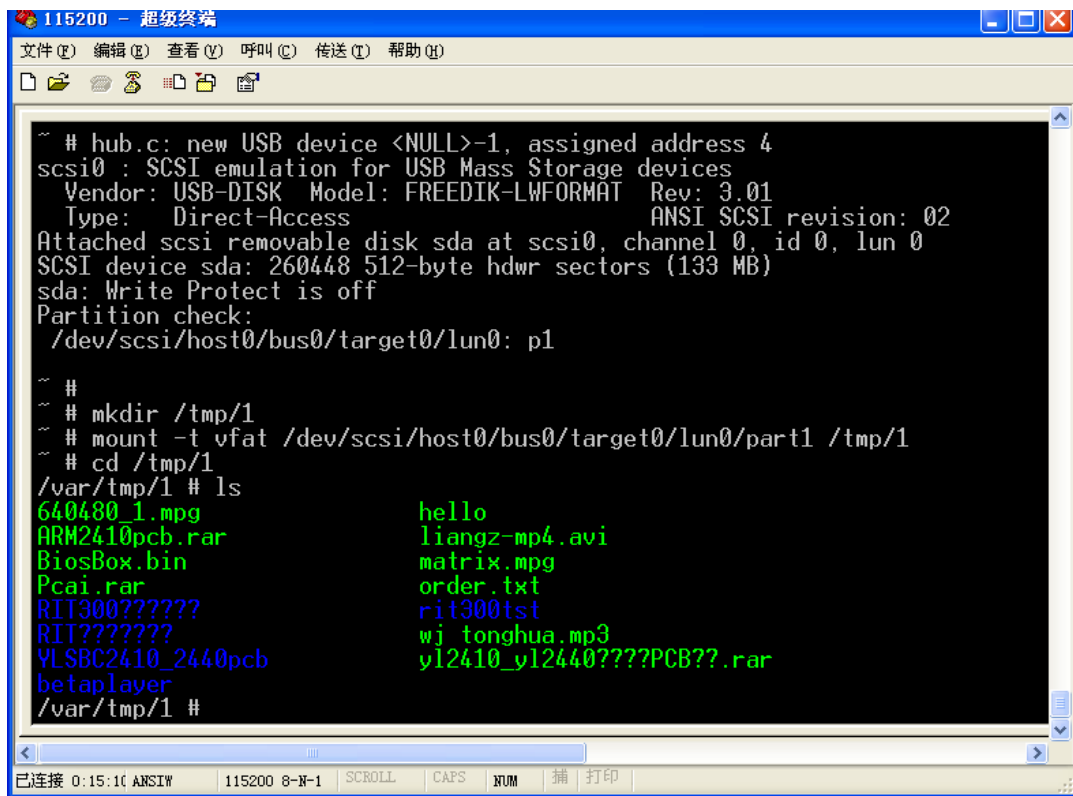
出现“/dev/scsi/host0/bus0/target0/lun0: p1”信息，说明 U 盘被正确检测到，这时就进行 U 盘挂接操作了。

命令如下：

mkdir /tmp/1 ;建立一个用来挂接 U 盘的目录

mount -t vfat /dev/scsi/host0/bus0/target0/lun0/part1 /tmp/1 ;将 U 盘挂接到 /tmp/1 目录下

ls /tmp/1 ; 查看 U 盘下的文件和目录



```
115200 - 超级终端
文件(F) 编辑(E) 查看(V) 呼叫(C) 传送(T) 帮助(H)
~ # hub.c: new USB device <NULL>-1, assigned address 4
scsi0 : SCSI emulation for USB Mass Storage devices
Vendor: USB-DISK Model: FREEDIK-LWFORMAT Rev: 3.01
Type: Direct-Access ANSI SCSI revision: 02
Attached scsi removable disk sda at scsi0, channel 0, id 0, lun 0
SCSI device sda: 260448 512-byte hdwr sectors (133 MB)
sda: Write Protect is off
Partition check:
/dev/scsi/host0/bus0/target0/lun0: p1

~ #
~ # mkdir /tmp/1
~ # mount -t vfat /dev/scsi/host0/bus0/target0/lun0/part1 /tmp/1
~ # cd /tmp/1
/var/tmp/1 # ls
640480_1.mpg          hello
ARM2410pcb.rar       liangz-mp4.avi
BiosBox.bin          matrix.mpg
Pcai.rar             order.txt
RIT300?????         rit300tst
RIT????????         wj_tonghua.mp3
YLSBC2410_2440pcb   yl2410_y12440?????PCB?? .rar
betaplayer
/var/tmp/1 #
```

这样就可以对 U 盘进行操作了，比如，读写，拷贝，删除等等操作。

(3) USB DEVICE 测试

AT91RM9200 芯片内部带有 USB 设备控制器，可以用 USB 设备端连接 PC 与 PC 通讯。在 LINUX 下的 USB 设备驱动 (gadget) 可模拟 U 盘、串口、网口等多类设备，此处只介绍 U 盘模拟方式。在使用 USB 设备前先用 USB 电缆连接 PC 与开发板的 USB 设备端口 CON1。

模拟 U 盘必须使用一个文件来完成开发板和 PC 间的文件传输，这个文件可以是一个普通文件也可以是一个设备文件。使用普通文件时，可以在 /tmp 目录下建一个临时文件，再加载 g_file_storage 模块并指定使用前面建立的文件，下图显示了建立一个 12M 的临时文件和加载驱动模块的操作过程。在 WINDOWS2000/NT/XP 等系统下，PC 机上检测到有新的 U 盘设备插入时能够自动识别和驱动，成功的话在我的电脑里会多出一个可移动磁

盘的图标。

运行的命令如下：

(1) 在 /tmp 目录下，建立一个 12M 的临时文件，命令如下：

```
cd /tmp
```

dd if=/dev/zero of=img bs=1k count=12k ;建立 img 文件，并为它分配 12M 的空间

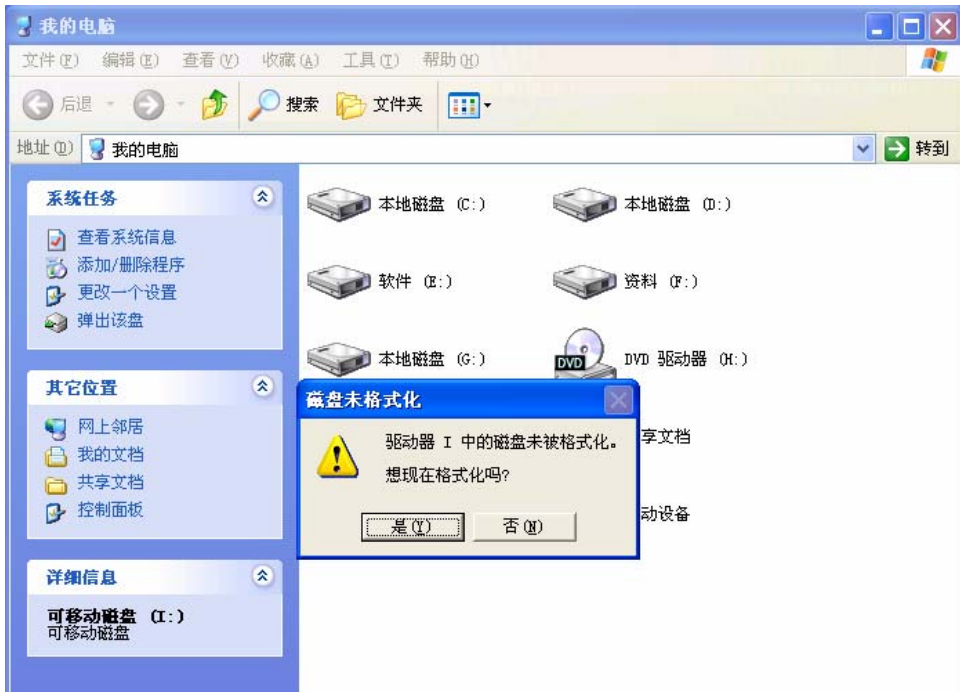
(2) 加载 g_file_storage 模块

```
insmod /usr/g_file_storage.ko file=img
```

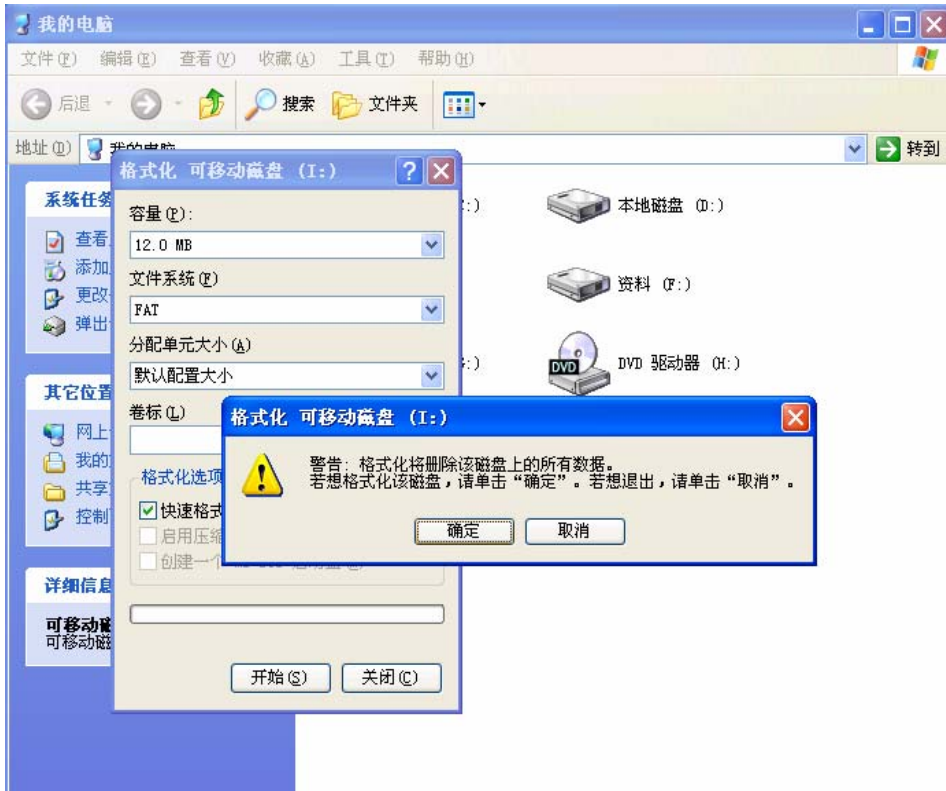
;加载 g_file_storage.ko 模块，模块在 /usr 目录下

```
~ # cd /tmp
/var/tmp # dd if=/dev/zero of=img bs=1k count=12k
12288+0 records in
12288+0 records out
/var/tmp # insmod /usr/g_file_storage.ko file=img
Using /usr/g_file_storage.ko
g_file_storage gadget: File-backed Storage Gadget, version: 20 0c
g_file_storage gadget: Number of LUNs=1
g_file_storage gadget-lun0: ro=0, file: /var/tmp/img
/var/tmp # g_file_storage gadget: full speed config #1
```

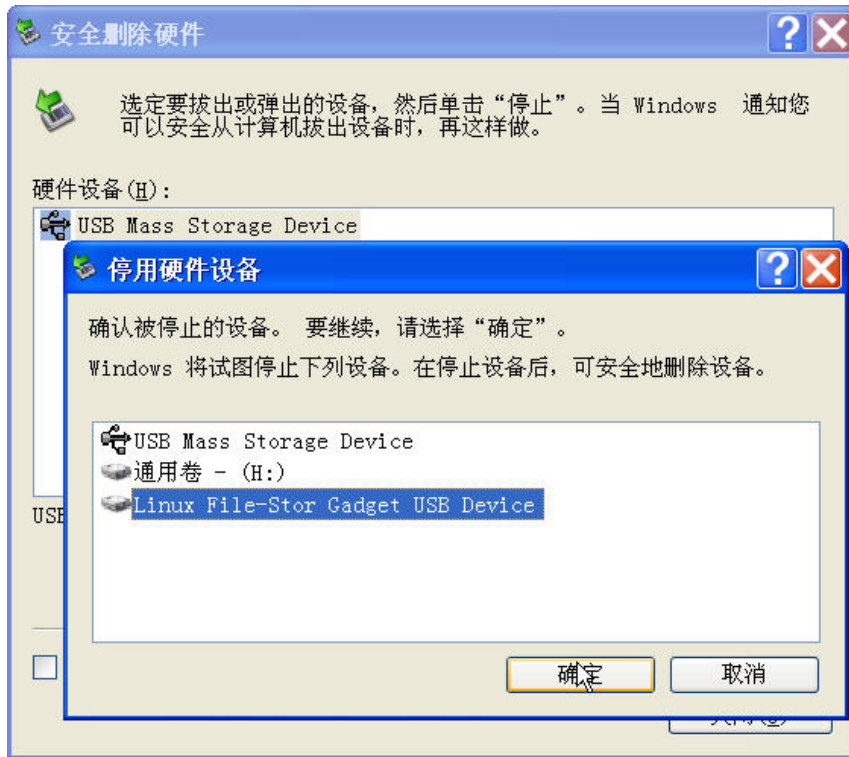
(3) 这时，在 PC 机端将出现一个可移动磁盘的图标。使用临时文件时，PC 机不能识别移动盘的分区信息和文件系统，因此需要先对其进行格式化



选择快速格式化再确定即可



格式化后就可以从 PC 向这个模拟 U 盘拷贝文件了，这里拷贝了一个 mp3 的文件。拷贝完毕后关闭 U 盘浏览窗口，再停止这个 U 盘设备。



(4) 在开发板上要使用 PC 上拷贝的文件，可以做下图的操作。

命令如下：

```
cd /tmp/
```

```
rmmod g_file_storage
```

```
mkdir udisk
```

```
mount -o loop -t vfat img udisk/
```

```
mount
```

```
ls udisk/
```



```
115200 - 超级终端
文件(F) 编辑(E) 查看(V) 呼叫(C) 传送(T) 帮助(H)

/var/tmp # cd /tmp/
/var/tmp # rmdir g_file_storage
/var/tmp # mkdir udisk
/var/tmp # mount -o loop -t vfat img udisk/
/var/tmp # mount
/dev/root on / type cramfs (ro)
none on /dev type devfs (rw)
/proc on /proc type proc (rw,nodiratime)
none on /dev/pts type devpts (rw)
none on /sys type sysfs (rw)
none on /var type ramfs (rw)
none on /var/tmp type tmpfs (rw)
/dev/loop/0 on /var/tmp/udisk type vfat (rw,nodiratime,fmask=
epage=cp437,iocharset=iso8859-1)
/var/tmp # ls udisk
dingxianghua.mp3
/var/tmp #
```

上图中 rmdir 卸载模块，之后会断开与 PC 的 USB 连接，mkdir udisk 建一个临时文件夹，mount -o loop ……可以将一个普通文件当作块设备挂载文件系统，同样用 umount 卸载此文件系统。

(4) 音频测试

音频测试，接上一步。

在进行音频测试前，确保 U 盘里放一个 .mp3 格式的音频文件。

将耳机或音箱连接到开发板的 J6 接口（音频接口）。

输入命令：

```
cd /tmp/1 ; 进入 U 盘被挂载的目录  
splay tonghua.mp3 ; 可以是任意的 mp3 音频
```

```

betaplayer
/var/tmp/1 # cp wj tonghua.mp3 tonghua.mp3
cp: wj: No such file or directory
cp: tonghua.mp3: No such file or directory
/var/tmp/1 # cp wj tonghua.mp3 ./tonghua.mp3
cp: wj: No such file or directory
cp: tonghua.mp3: No such file or directory
/var/tmp/1 # cp ./wj tonghua.mp3 ./tonghua.mp3
cp: ./wj: No such file or directory
cp: tonghua.mp3: No such file or directory
/var/tmp/1 # cp ./wj*.mp3 ./tonghua.mp3
/var/tmp/1 # ls
640480_1.mpg          hello
ARM2410pcb.rar       liangz-mp4.avi
BiosBox.bin          matrix.mpg
Pcai.rar             order.txt
RIT300????????      rit300tst
RIT????????         tonghua.mp3
YLSBC2410_2440pcb   wj tonghua.mp3
betaplayer          y12410
/var/tmp/1 # splay tonghua.mp3
Reading data from NAND FLASH without ECC is not recommended
Reading data from NAND FLASH without ECC is not recommended

```

已连接 0:18:21 ANSIIW | 115200 8-N-1 | SCROLL CAPS NUM 捕 打印

(5) MMC 测试

MMC 测试，必须在 Linux 启动前，将 MMC 卡插入 MMC 卡座，Linux 检测到 MMC 卡后，会在 /dev/mmc/ 目录生成一个设备节点 /dev/mmc/disc0/part1。

挂载命令如下：

mkdir /tmp/1 ;建立挂载目录(如果前面建了此目录的话要 umount 卸载掉)
mount -t vfat /dev/mmc/disc0/part1 /tmp/1 ; 将 MMC 存储卡挂载到 /tmp/1 目录下
 这样就可以对 MMC 卡的进行相关操作了，比如读写，删除以及拷贝等等。

(6) IDE 硬盘挂载测试

IDE 测试，必须在 Linux 启动前，用硬盘线将 IDE 硬盘与开发板的 IDE 接口连接起来，并给 IDE 硬盘通电，这时再启动 Linux，如果 IDE 硬盘被正确检测到，Linux 将在 /dev/ide 目录下生成一个设备节点 /dev/ide/host0/bus0/target0/lun0/part1。

IDE 硬盘挂接命令如下:

mkdir /tmp/1 ;建立挂接目录(如果前面建了此目录的话要 `umount` 卸载掉
建立挂接目录

mount -t vfat /dev/ide/host0/bus0/target0/lun0/part1 /tmp/1; 将 IDE 硬盘挂接到/tmp/1 目录下

这样就可以对 IDE 硬盘进行相关操作了, 比如读写, 删除以及拷贝等等。

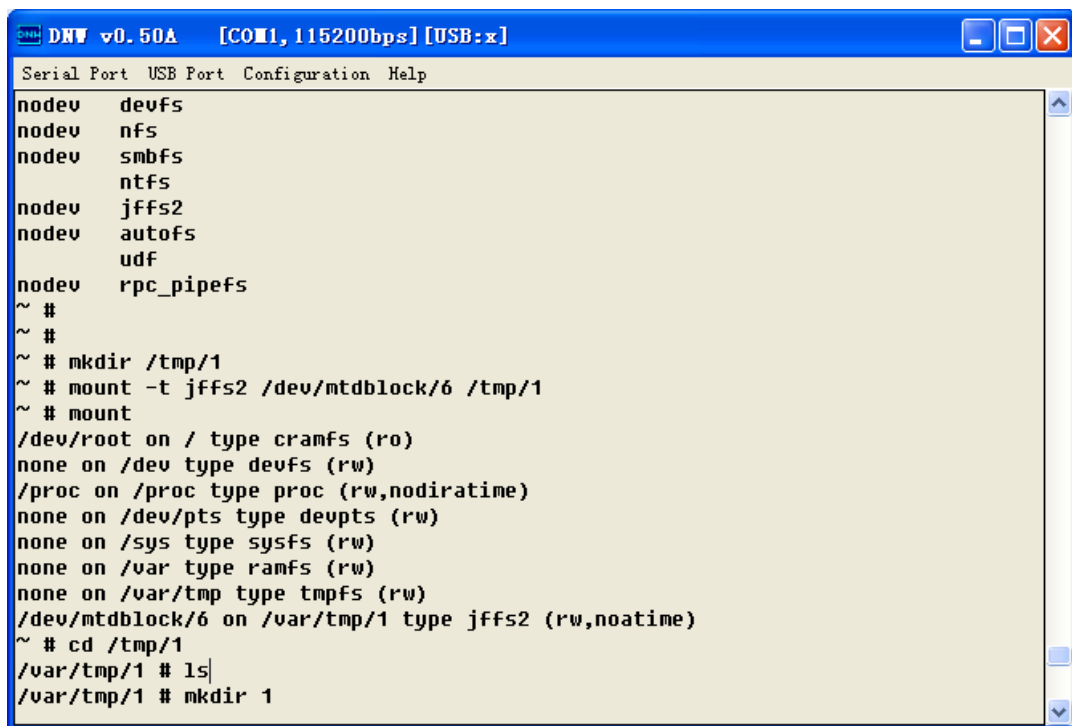
3.4 Linux 系统下的一些应用

3.4.1 NAND FLASH 的分区挂接

在 Linux 启动的时候, 已将 NAND FLASH 分好区了, 分区的信息在 Linux 的启动代码里有显示。

NAND FLASH 分区的挂接命令如下:

mkdir /tmp/1 ; 建立挂接目录
mount -t jffs2 /dev/mtdblock/6 /tmp/1 ; 将 NAND FLASH 的分区挂接到
/tmp/1

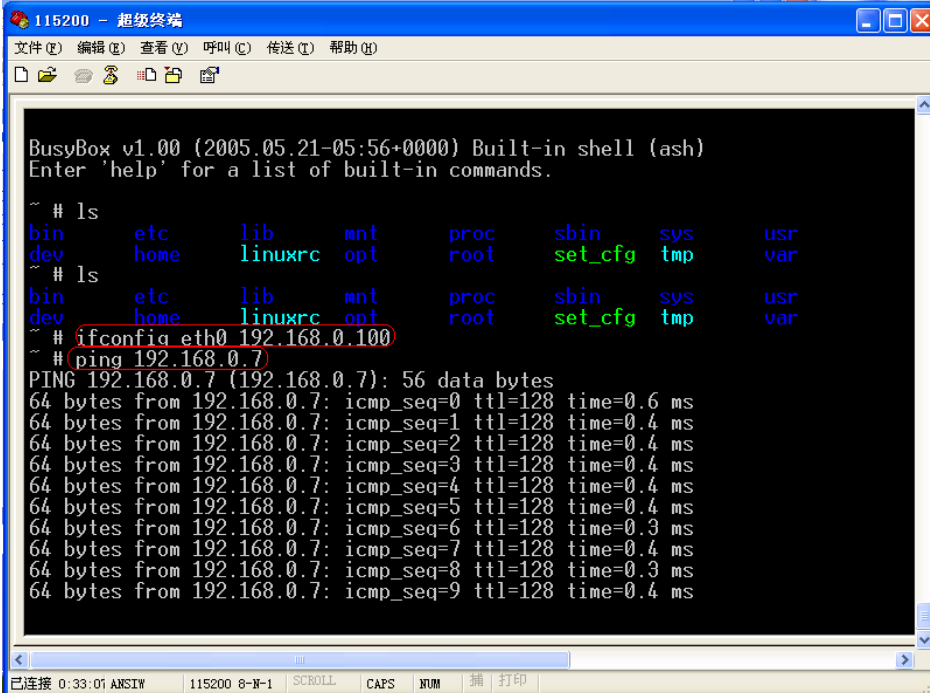


```
Serial Port USB Port Configuration Help
nodev  devfs
nodev  nfs
nodev  smbfs
       ntfs
nodev  jffs2
nodev  autofs
       udf
nodev  rpc_pipefs
~ #
~ #
~ # mkdir /tmp/1
~ # mount -t jffs2 /dev/mtdblock/6 /tmp/1
~ # mount
/dev/root on / type cramfs (ro)
none on /dev type devfs (rw)
/proc on /proc type proc (rw,nodiratime)
none on /dev/pts type devpts (rw)
none on /sys type sysfs (rw)
none on /var type ramfs (rw)
none on /var/tmp type tmpfs (rw)
/dev/mtdblock/6 on /var/tmp/1 type jffs2 (rw,noatime)
~ # cd /tmp/1
/var/tmp/1 # ls|
/var/tmp/1 # mkdir 1
```

3.4.2 网络通讯地址设置

网卡芯片驱动加载成功后，可在 Linux 的 shell 下使用 `ifconfig` 命令来设置网卡芯片的 IP 地址，之后可 PING 局域网中的其他电脑的 IP 地址。

设置 IP 地址命令：*`ifconfig eth0 192.168.0.100`*



```
115200 - 超级终端
文件(F) 编辑(E) 查看(V) 呼叫(C) 传送(T) 帮助(H)
~ # ls
bin      etc      lib      mnt      proc     sbin     sys      usr
dev      home    linuxrc  opt      root     set_cfg  tmp      var
~ # ls
bin      etc      lib      mnt      proc     sbin     sys      usr
dev      home    linuxrc  opt      root     set_cfg  tmp      var
~ # ifconfig eth0 192.168.0.100
~ # ping 192.168.0.7
PING 192.168.0.7 (192.168.0.7): 56 data bytes
64 bytes from 192.168.0.7: icmp_seq=0 ttl=128 time=0.6 ms
64 bytes from 192.168.0.7: icmp_seq=1 ttl=128 time=0.4 ms
64 bytes from 192.168.0.7: icmp_seq=2 ttl=128 time=0.4 ms
64 bytes from 192.168.0.7: icmp_seq=3 ttl=128 time=0.4 ms
64 bytes from 192.168.0.7: icmp_seq=4 ttl=128 time=0.4 ms
64 bytes from 192.168.0.7: icmp_seq=5 ttl=128 time=0.4 ms
64 bytes from 192.168.0.7: icmp_seq=6 ttl=128 time=0.3 ms
64 bytes from 192.168.0.7: icmp_seq=7 ttl=128 time=0.4 ms
64 bytes from 192.168.0.7: icmp_seq=8 ttl=128 time=0.3 ms
64 bytes from 192.168.0.7: icmp_seq=9 ttl=128 time=0.4 ms
```

注意：以后几章节都是工作在 Linux 下，并且 IP 地址为 192.168.0.100。

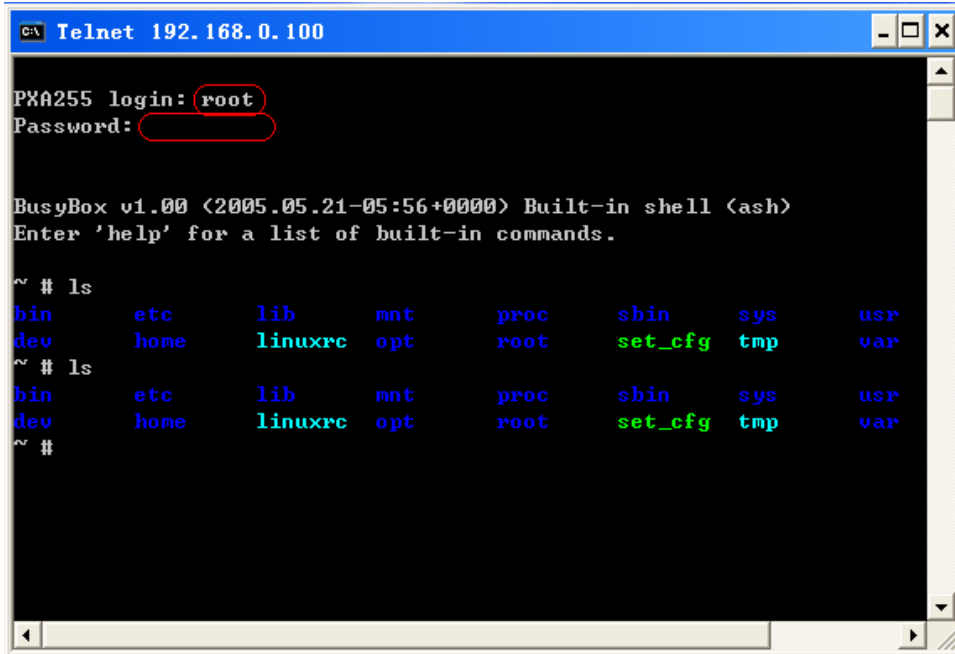
3.4.3 从 PC 机 TELNET 到开发板

在从 PC 机 TELNET 到开发板之前，先要启动 PC 机端的 telnet 服务，这个 TELNET 服务的位置：控制面板—>管理工具—>服务—>Telnet。

接着进入 PC 机的命令窗口，输入命令：

```
telnet 192.168.0.100
```

这时会提示要求输入用户名和密码，用户名：root,密码是 linux，可以用 exit 命令退出登陆。

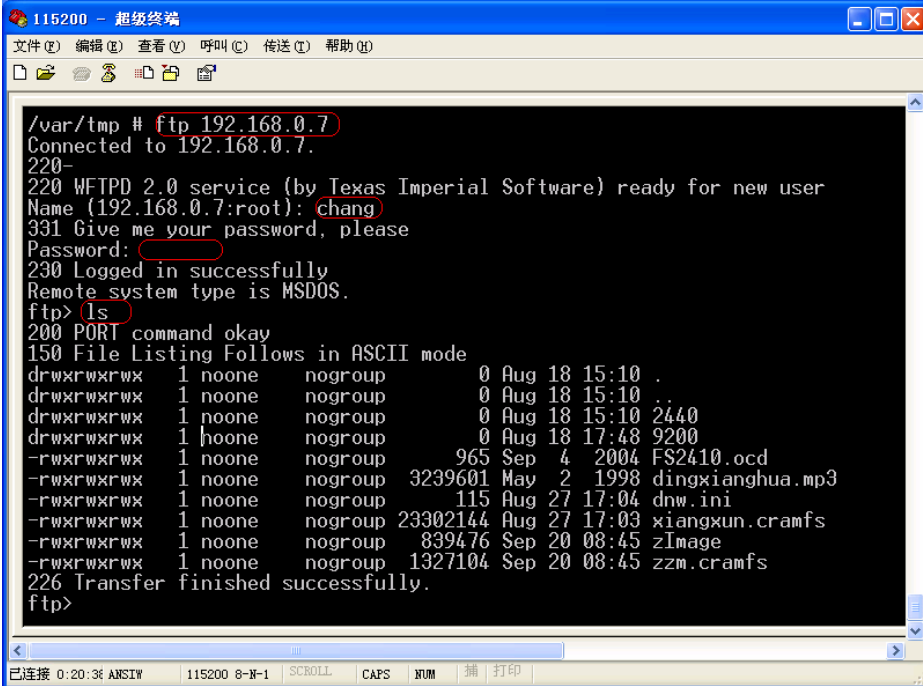


```
C:\ Telnet 192.168.0.100
PXA255 login: root
Password:
BusyBox v1.00 (2005.05.21-05:56+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

~ # ls
bin      etc      lib      mnt      proc     sbin     sys      usr
dev      home    linuxrc  opt      root     set_cfg  tmp      var
~ # ls
bin      etc      lib      mnt      proc     sbin     sys      usr
dev      home    linuxrc  opt      root     set_cfg  tmp      var
~ #
```

3.4.4 从开发板 FTP 到 PC 机

开发板的根文件系统中带有 FTP 客户端程序，因此也可以在开发板上用 FTP 连接 PC 机，用 bye 指令可断开连接。注意，在运行 FTP 客户端程序的时候，在 PC 机上必须要有 FTP 服务端程序运行，至于 FTP 服务端程序的安装，这里就不说明了。



```
115200 - 超级终端
文件(F) 编辑(E) 查看(V) 呼叫(C) 传送(T) 帮助(H)
/var/tmp # ftp 192.168.0.7
Connected to 192.168.0.7.
220-
220 WFTPD 2.0 service (by Texas Imperial Software) ready for new user
Name (192.168.0.7:root): chang
331 Give me your password, please
Password:
230 Logged in successfully
Remote system type is MSDOS.
ftp> ls
200 PORT command okay
150 File Listing Follows in ASCII mode
drwxrwxrwx 1 noone nogroup 0 Aug 18 15:10 .
drwxrwxrwx 1 noone nogroup 0 Aug 18 15:10 ..
drwxrwxrwx 1 noone nogroup 0 Aug 18 15:10 2440
drwxrwxrwx 1 noone nogroup 0 Aug 18 17:48 9200
-rwxrwxrwx 1 noone nogroup 965 Sep 4 2004 FS2410.ocd
-rwxrwxrwx 1 noone nogroup 3239601 May 2 1998 dingxianghua.mp3
-rwxrwxrwx 1 noone nogroup 115 Aug 27 17:04 dnw.ini
-rwxrwxrwx 1 noone nogroup 23302144 Aug 27 17:03 xiangxun.cramfs
-rwxrwxrwx 1 noone nogroup 839476 Sep 20 08:45 zImage
-rwxrwxrwx 1 noone nogroup 1327104 Sep 20 08:45 zzm.cramfs
226 Transfer finished successfully.
ftp>
```

至于 FTP 服务器用户的密码，是自己在 PC 端设置的。

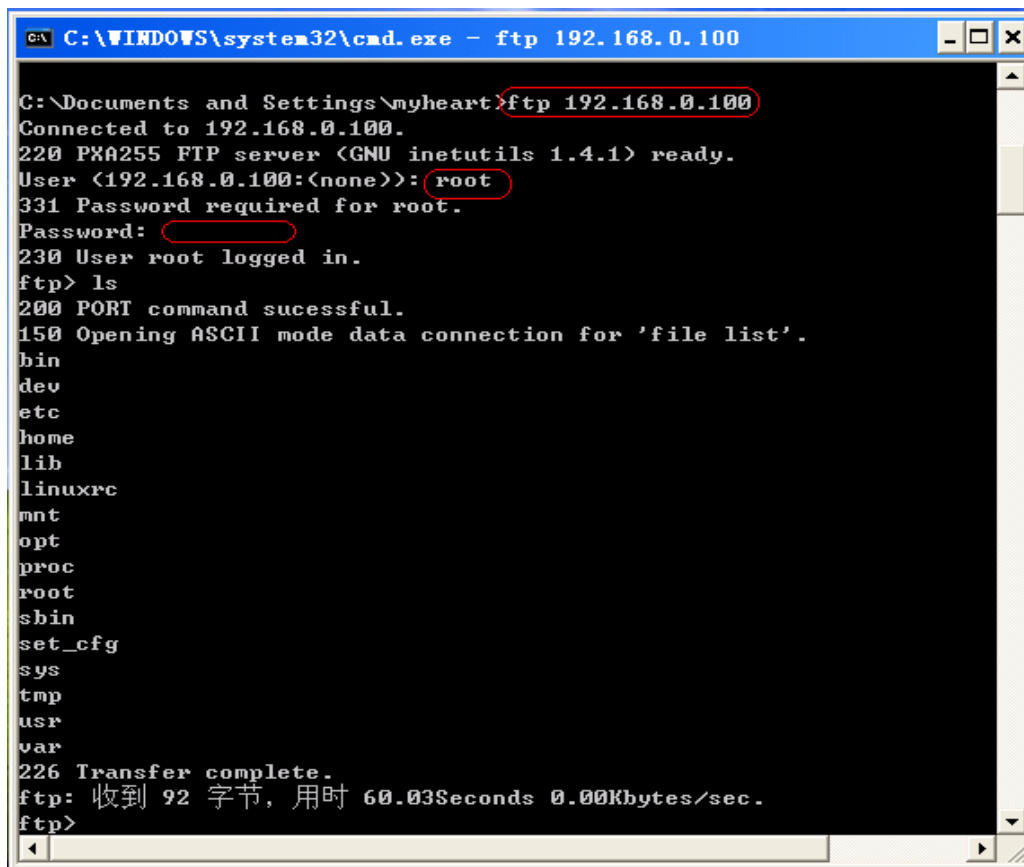
登陆了 PC 机的 FTP 后，就可以从 PC 机上下载和上传文件了，注意，在从 PC 机到开发板的时候要注意，先进入到 /tmp 目录，然后在 FTP 登陆到 PC 机，因为 /tmp 是在 SDRAM 中建立的空间，可以写的，而其他的目录是只读的。

3.4.5 从 PC 机 FTP 到开发板

开发板的 Linux 有一个 FTP 服务端程序，这样就可以从 PC 机 FTP 到开发板。打开 PC 机的命令窗口，输入命令：

[ftp 192.168.0.100](#)

接着回提示要求输入用户名和密码，用户名：root 密码:linux。



```
C:\WINDOWS\system32\cmd.exe - ftp 192.168.0.100
C:\Documents and Settings\myheart>ftp 192.168.0.100
Connected to 192.168.0.100.
220 PXA255 FTP server (GNU inetutils 1.4.1) ready.
User (192.168.0.100:(none)): root
331 Password required for root.
Password:
230 User root logged in.
ftp> ls
200 PORT command successful.
150 Opening ASCII mode data connection for 'file list'.
bin
dev
etc
home
lib
linuxrc
mnt
opt
proc
root
sbin
set_cfg
sys
tmp
usr
var
226 Transfer complete.
ftp: 收到 92 字节, 用时 60.03Seconds 0.00Kbytes/sec.
ftp>
```

3.4.6 在开发板上建立 WEB 服务器

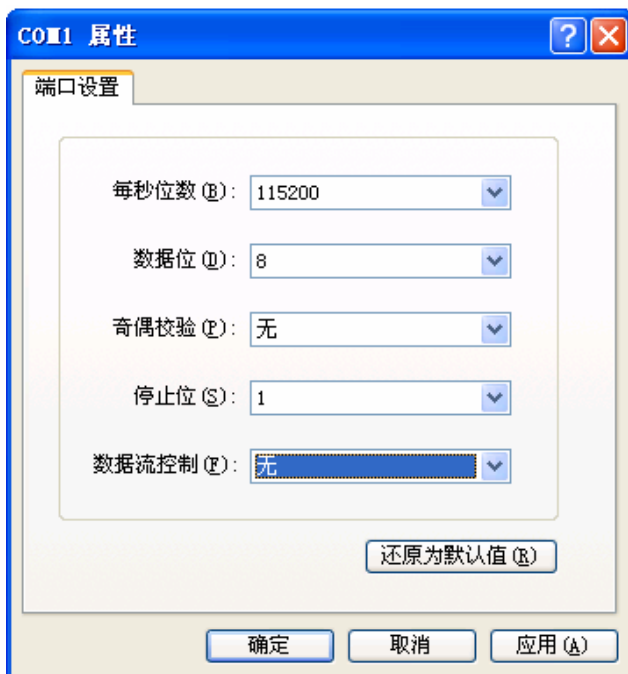
利用 BUSYBOX 所带的 HTTPD 可在开发板上建立一个简单的 WEB 服务器，在 shell 下执行 `httpd -h /home/httpd` 就可以启动 WEB 服务了，在 PC 上打开网络浏览器，在地址栏里直接输入开发板的 IP 地址 192.168.0.100 再回车即可看到开发板上的网页。



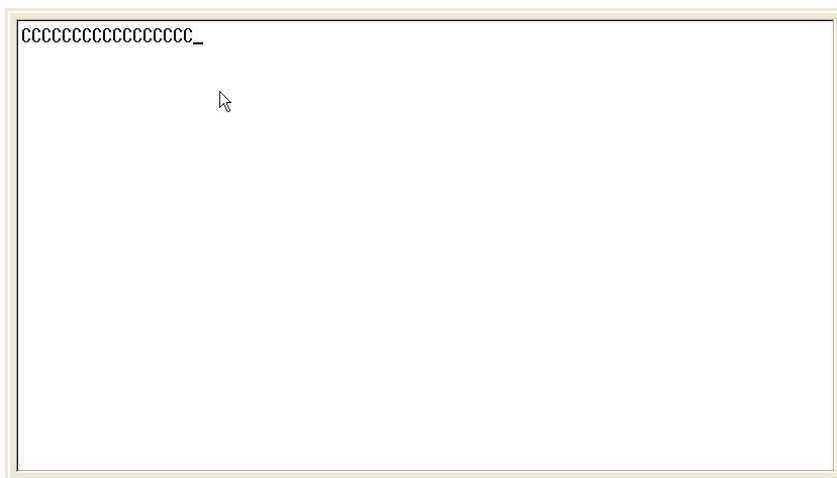
第四章 烧写 BIOS 及 BIOS 的相关说明

4.1 下载运行 BIOS

用交叉串口线将开发板的 P2 串口和 PC 机的串口连接起来，将 JP6 的跳线位置设置在 1+2 (INTER_BOOT) 的地方，选择从串口下载程序，打开超级终端，新建一个串口连接，选择连接开发板的串口，进行如下设置

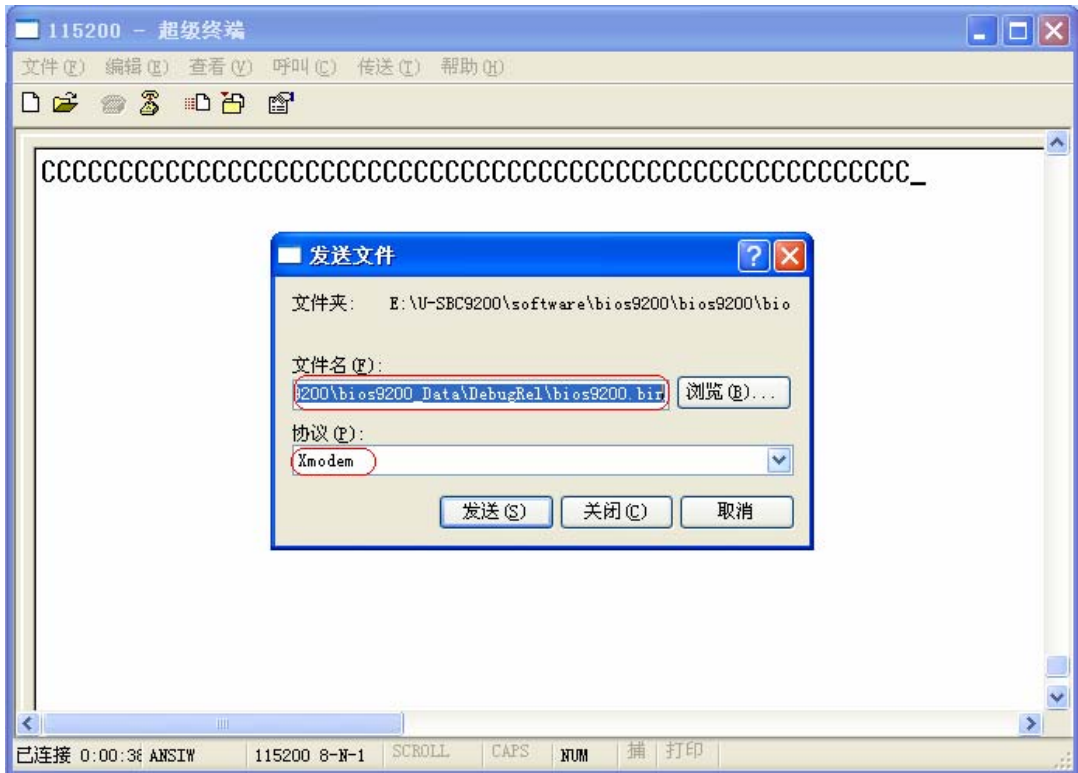


按确定后，再按连接图标。打开开发板电源，然后超级终端会不断显示 C 字符，

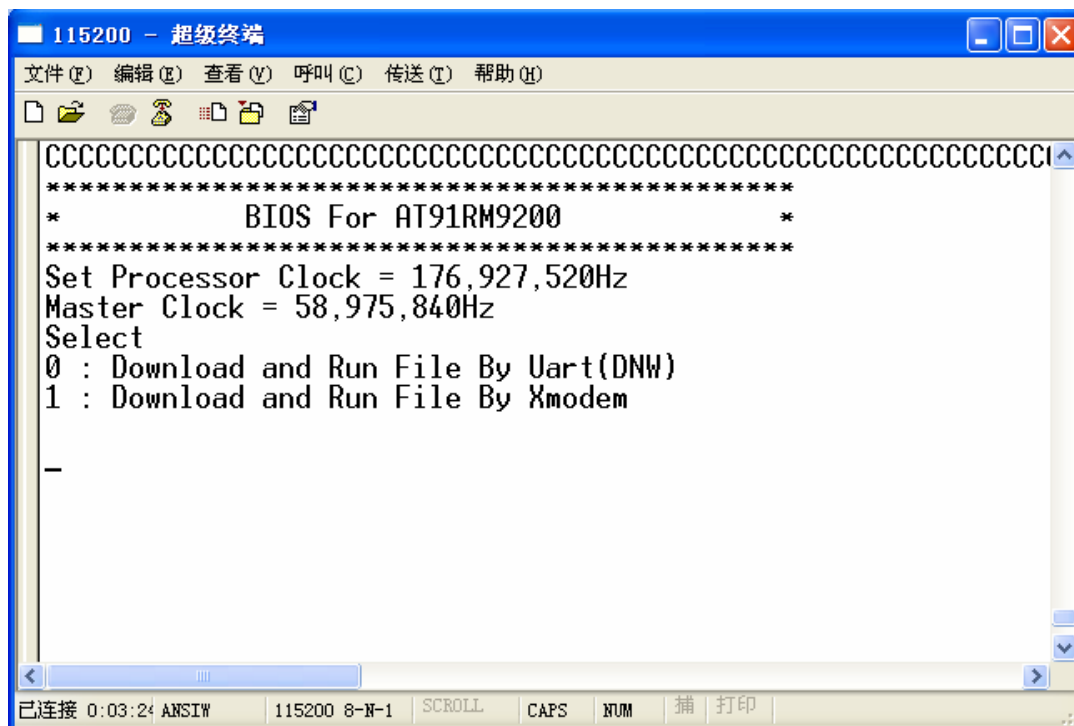


这是 AT91RM9200 片内启动程序在没有检查到任何可用外部存储器带有有效启动程序的情况下运行的 XMODEM 方式下载程序。

这时在超级终端选择 BIOS9200.BIN（此文件在用户光盘的“目标代码”文件夹下）进行发送。



点击文件传送后，接着一直按着开发板上的 S2~S5 中的任意一个键，（因为不按键的话，下载的程序运行后，将会启动 FLASH 里面已固化好的程序），下载结束后，会自动运行，运行的情况如下



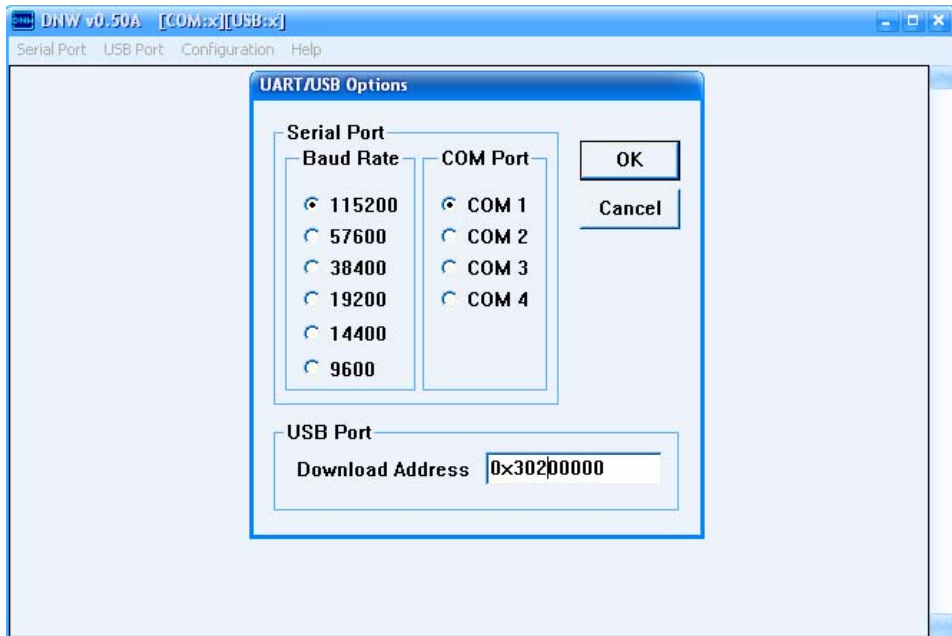
```
115200 - 超级终端
文件(F) 编辑(E) 查看(V) 呼叫(C) 传送(T) 帮助(H)
*****
* BIOS For AT91RM9200 *
*****
Set Processor Clock = 176,927,520Hz
Master Clock = 58,975,840Hz
Select
0 : Download and Run File By Uart(DNW)
1 : Download and Run File By Xmodem
-
已连接 0:03:24 ANSII 115200 8-N-1 SCROLL CAPS NUM 捕 打印
```

当出现上面的界面后，可以松开按键了。

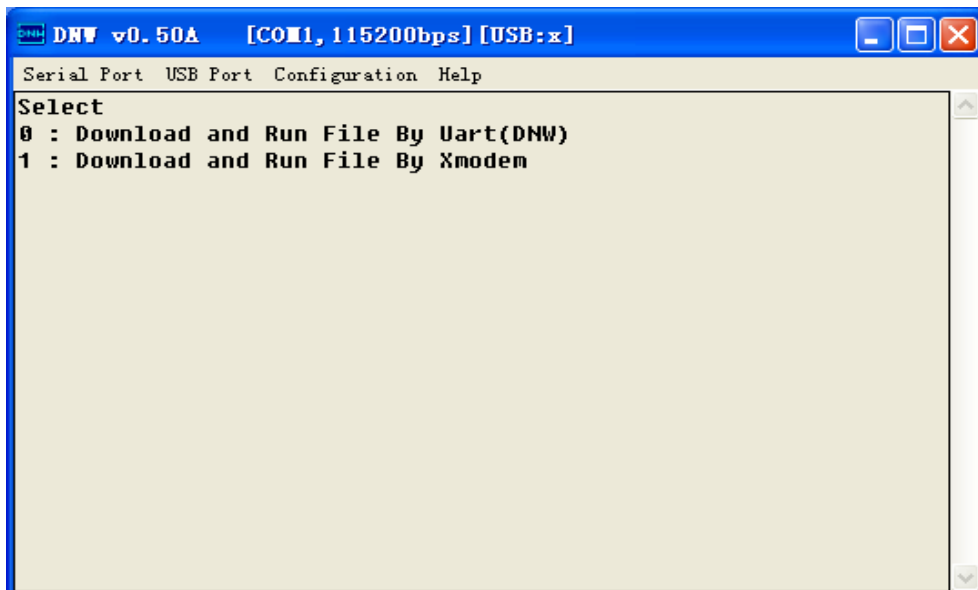
其中 DNW 方式下载是要用 DNW 程序的串口下载方式的，我们可以通过这种方式下载启动程序到 SDRAM 中运行。

4.2 利用 BIOSBOX 将程序烧写到 Nor Flash

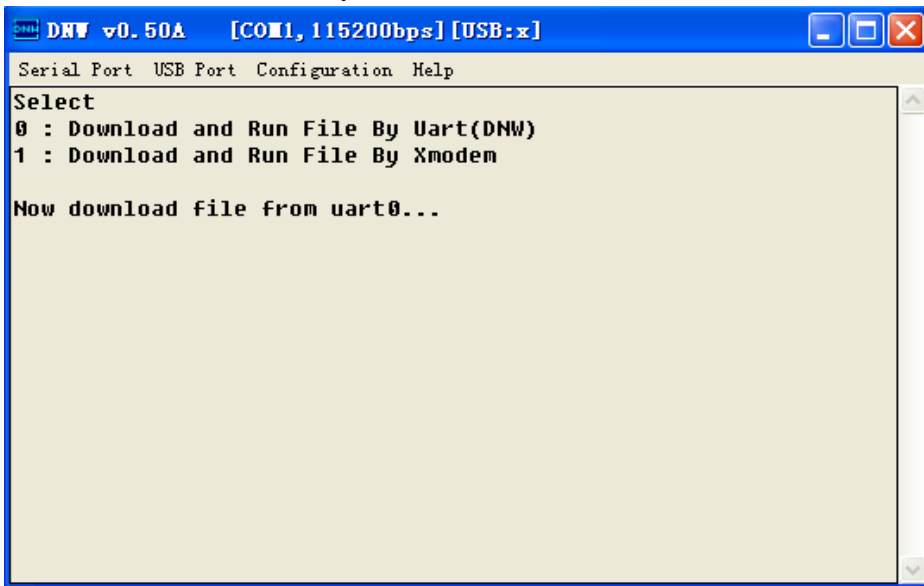
接着 4.1 章节的操作，先在超级终端选择断开连接，然后再打开 DNW 程序，在 Configuration 菜单下选择 Options，作如下设置



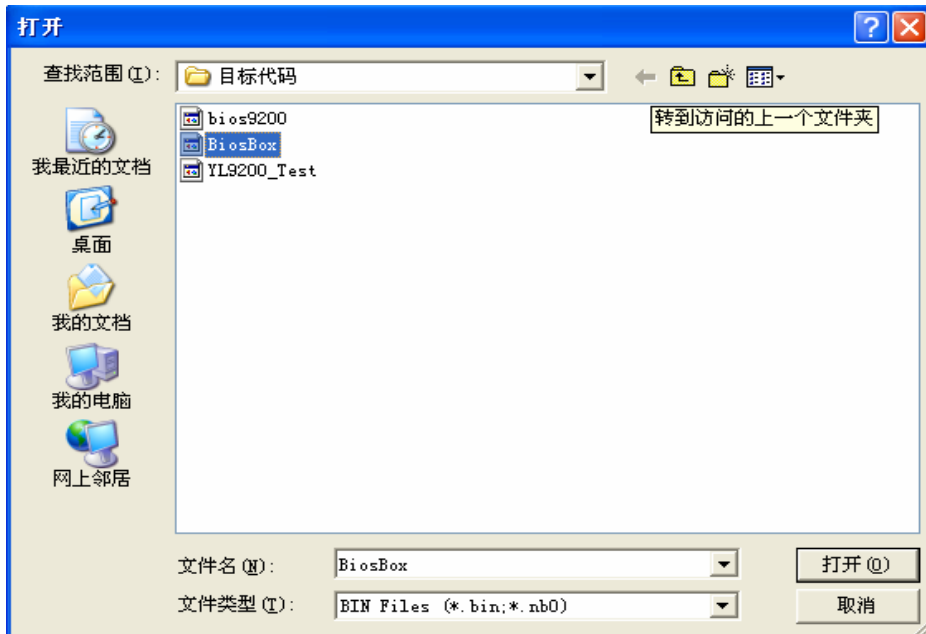
按 OK 后再在 Serial Port 菜单下选择 Connect，敲一下回车键，显示如下



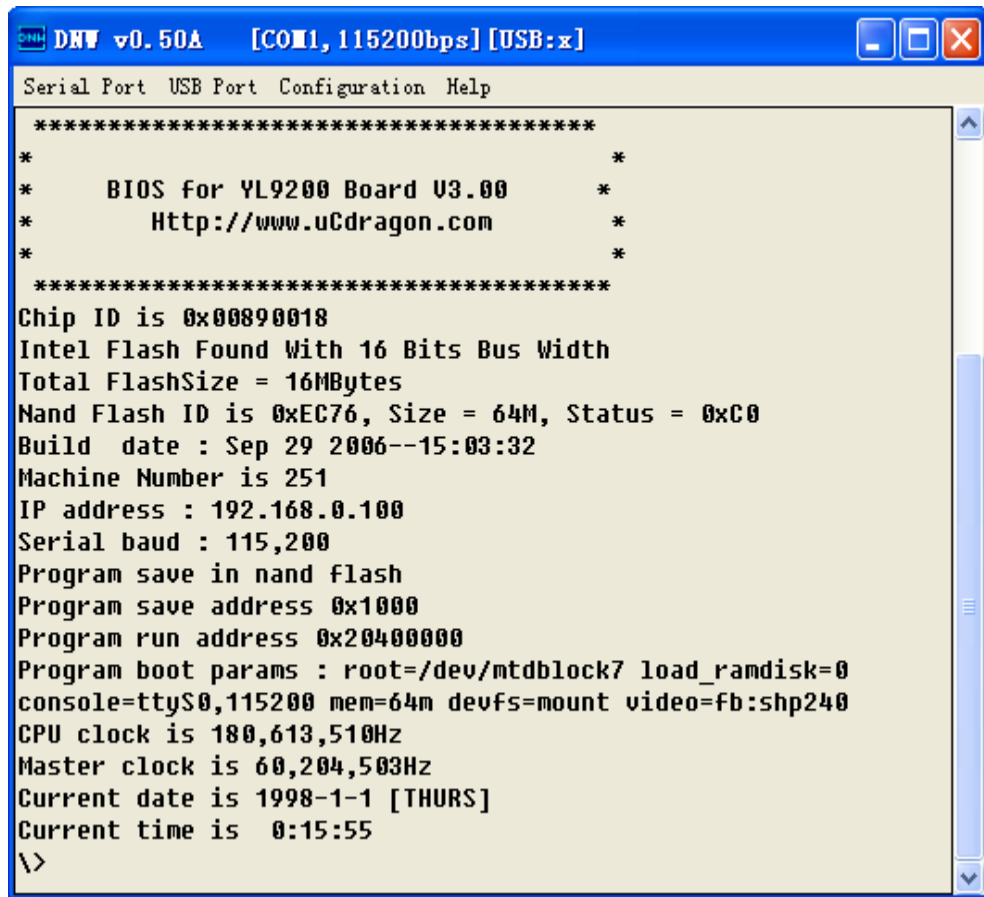
选择 0 : Download and Run File By Uart(DNW)



选择所要下载的文件——BIOSBOX.BIN（此文件在用户光盘的“目标代码”文件夹下）



接收成功显示如下



```
DNW v0.50A [COM1, 115200bps] [USB:x]
Serial Port  USB Port  Configuration  Help
*****
*                               *
*   BIOS for YL9200 Board V3.00   *
*   Http://www.uCdragon.com      *
*                               *
*****
Chip ID is 0x00890018
Intel Flash Found With 16 Bits Bus Width
Total FlashSize = 16MBytes
Nand Flash ID is 0xEC76, Size = 64M, Status = 0xC0
Build date : Sep 29 2006--15:03:32
Machine Number is 251
IP address : 192.168.0.100
Serial baud : 115,200
Program save in nand flash
Program save address 0x1000
Program run address 0x20400000
Program boot params : root=/dev/mtdblock7 load_ramdisk=0
console=ttyS0,115200 mem=64m devfs=mount video=fb:shp240
CPU clock is 180,613,510Hz
Master clock is 60,204,503Hz
Current date is 1998-1-1 [THURS]
Current time is 0:15:55
\>
```

(1) 进入 BIOS 状态，输入命令：

comload;

然后回车。

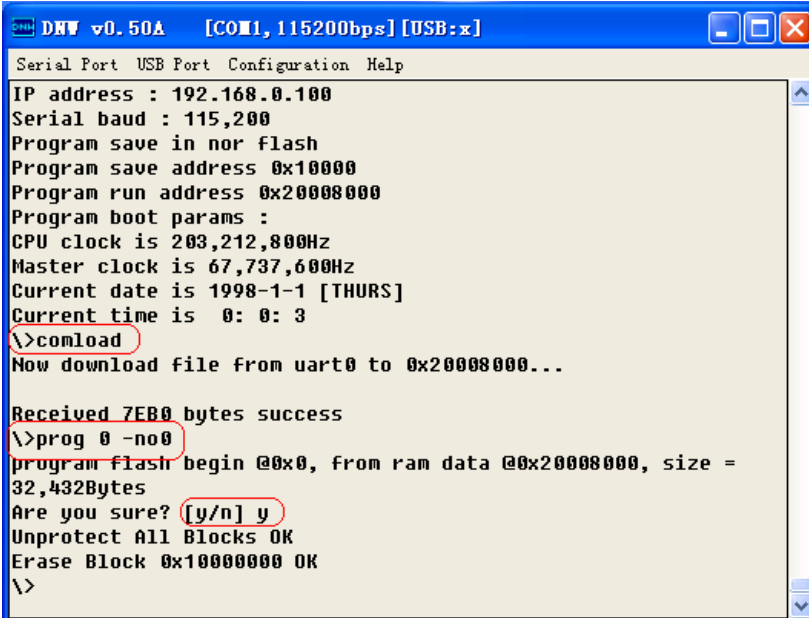
接着选择要传输的文件 BIOSBOX.bin。

文件传输结束后，会自动返回到 BIOS 的命令状态下。

(2) 接着，输入命令：

prog 0 -no0 ;这个命令将刚刚下载的 BIOXBOX.bin 烧写到 Nor Flash 中。

在接着出现的[y/n]下，选择“y”，这样就可以将程序烧写到 Nor Flash 中了。



```
DNW v0.50A [COM1, 115200bps] [USB:x]
Serial Port USB Port Configuration Help
IP address : 192.168.0.100
Serial baud : 115,200
Program save in nor flash
Program save address 0x10000
Program run address 0x20008000
Program boot params :
CPU clock is 203,212,800Hz
Master clock is 67,737,600Hz
Current date is 1998-1-1 [THURS]
Current time is 0: 0: 3
\>comload
Now download file from uart0 to 0x20008000...

Received 7EB0 bytes success
\>prog 0 -no0
program flash begin @0x0, from ram data @0x20008000, size =
32,432Bytes
Are you sure? [y/n] y
Unprotect All Blocks OK
Erase Block 0x10000000 OK
\>
```

(3) 选择外部启动模式

选择外部启动模式，这时将启动刚刚烧写到 Nor Flash 中的 BIOSBOX 程序，要从外部启动，必须将 JP6 的跳线的位置设置在 **2+3 (EXTER BOOT)** 的位置。

这时在上电复位，按复位键，将工作在外部启动模式下，从 Nor Flash 中启动程序。这时程序，也是跑的 BIOS，在这个启动模式下，可以烧写 NAND FLASH.

注意：此开发板只支持从 Nor Flash 启动。

4.3 BIOS 的功能说明

(注：以下命令所带参数中地址和长度都属 16 进制，不必在前面加 0x)

help 和 **?** 可以列出所有命令并给出简单的说明；

date 命令可以显示和设置当前日期，只输入 date 命令则显示日期，输入 date 2004-6-8 则设置当前日期为 2004 年 6 月 8 日。

time 命令可以显示和设置当前时间，只输入 **time** 命令则显示时间，输入 **date 14: 4: 30** 则设置当前时间为 14: 4: 30。

setweek n 可设置星期几，**n** 从 1 到 7 表示星期一到星期日。

clock 可以显示当前的工作频率。

setmclk 可以改变 CPU 工作频率，具体参数设置可见芯片手册，注意不要使频率超出工作范围。频率参数属于可以保存和调入的参数，这次设置和保存后下次复位 BIOS 会自动调入这写参数初始化 CPU。

setbaud 可改控制串口的波特率，改完后要在 PC 上相应改变串口通讯波特率后再敲回车。

ipcfg 可显示和修改 TFTP 下载时所用的 IP 地址，只输入 **IPCFG** 则显示当前 IP 地址，输入 **IPCFG192.168.2.223** 则将 IP 地址改为 192.168.2.223。

netload 启动 TFTP 接收，若没带地址参数，则使用缺省下载地址 0x0c008000，若指定地址，下载数据保存到指定地址开始的 SDRAM 中去，如 **netload c300000**。启动 **tftp** 接收后，要在 PC 端执行 **tftp** 下载程序，在 WIN2000 或 WINXP 下，直接输入 **tftp -i xxx.xxx.xxx.xxx put 文件名** 即可；在 WIN98 下，使用 CDROM 里所带的 TFTP 程序；在 LINUX 下，使用 CDROM 里所带的 TFTPMD 程序。注意进行 TFTP 传输时要保证 PC 机和开发板处于同一个 IP 段内。

netrun 或者是 **g** 启动 TFTP 接收完数据后会自动运行下载到的程序，缺省下载地址和指定参数同 **netload**。

comload 启动串口下载（DNW 程序的串口下载），缺省下载地址和指定参数同 **netload**。

comrun 启动串口下载（DNW 程序的串口下载）并在接收完数据后自动运行下载的程序，缺省下载地址和指定参数同 **netload**。

rx 启动 XMODEM 方式下载，可在超级终端内选择 1K XMODEM 或 XMODEM 发送数据到开发板上，缺省下载地址和指定参数同 **netload**。

rxrun 在启动 XMODEM 方式接收完数据后自动运行下载到的程序，缺省下载地址和指定参数同 netload。

prog 可以烧写 NOR FLASH，目前支持 SST39VF160。prog 命令完整的参数是 prog addr1 addr2 length [-no0],其中 addr1 是要烧写的 FLASH 的地址，大于等于 0，小于 200000，字对齐，addr2 是 sdram 中要烧进 flash 的数据区起始地址，length 是要烧写的长度，-no0 表示要把数据烧进 Nor Flash 0 地址开始的地方时，是否修改 0 地址的指令，因为 CPU 复位总是从 0 开始执行的，当用 Nor Flash 启动时，若用 prog 命令将下载到的程序烧入 Nor Flash 0 地址开始的地方并在命令最后指定-no0,那么在复位后，就不会再运行 Bios 而直接启动用户程序了，若不在 prog 命令最后加-no0, 则 BIOS 可以烧写 NOR FLASH 0 地址的数据前，将 0 地址的指令改为直接跳转到 0x1f0000 处即 Bios 的驻留地址，并保存原程序 0 地址将要跳转到的地址，以后在执行 boot 指令时再跳转过去执行用户烧入的程序。运行 BIOS。在运行 BIOS 下载完数据后，也可不带参数直接执行 prog 命令，缺省的 NORFLASH 地址是用户程序存储地址 prog_s_addr（见后面 setpa 命令），sdram 中数据起始地址和数据长度在接收成功后自动设定了。

ap 指令自动下载完数据并将数据烧写到 NORFLASH 的 0 地址处，缺省为 TFTP 下载，指定-c 表示串口下载（DNW 方式），-x 表示 XMODEM 下载，-b 表示不修改 0 地址的指令。

backup 可用在第一次烧写完 BIOS 到 NORFLASH 0 地址后上电执行时将 BIOS 本身拷贝到 0x1f0000 处。

copy 将 NORFLASH 某地址的数据拷贝到另一地址。

boot 可运行用户通过 BIOS 下载烧写到 0 地址并修改过 0 地址跳转地址的程序，见 prog.

run 可运行存储器中的程序，缺省地址就是缺省下载地址，也可指定运行地址。

move addr1 addr2 size 可将存储器中 addr1 开始的长度为 size 的数据拷贝到 addr2 开始的地址去。

mrun 可自动执行 move 的过程并运行程序，比如在 FS44B0 中我们将 UCLINUX 内核保存在 Nor Flash 的 0x10000 开始的地方，长度为 800K，它的运行地址是 0x0c300000,那么 MRUN 就可以完成拷贝的操作并直接运行。MRUN 内部使用的参数见 setpa 命令。

md 显示存储器中的数据，可以带地址参数。

memd 可显示单个存储器单元中的内容，-c 参数表示 8 位数据，-s 参数表示 16 位数据,-l 参数表示 32 位数据，后面跟存储器地址。

mems 可修改单个存储器单元中的内容，-c,-s,-l 参数同上，后面跟存储器地址和要写入的内容。

machine 可设置机器号，适用于 LINUX 此参数可保存。

setpa 有几个参数

Usage : setpa -s[-r][-i][-ni][-nor][-nand] [address]

-s save address

-r run address

-i initrd save address

-ni disable initrd

-nor use nor flash to save

-nand use nand flash to save

其中-s 表示用户程序在 FLASH 中的存储地址，如上面所说的将 UCLINUX 内核保存到 Nor Flash 的 0x10000 处，为使 mrun 正确运行，我们就要设置 setpa -s 10000

-r 表示用户程序的运行地址，如上面所说的将 UCLINUX 内核的运行地址是 0x0c300000,为使 mrun 正确运行，我们就要设置 setpa -r c300000

-i 表示使用 initrd(对于 linux 或 uClinux),它的存储地址是多少。

-noi 表示取消 initrd。

-nor 表示用户程序存储在 Nor Flash 中，-nand 表示用户程序存储在 Nand Flash 中。注意使用 Nand Flash 存储时，前述保存地址 1000 表示 Nand 分区 1，2000 表示 Nand 分区 2，依此类推，Nand 分区见 nfpert 命令。

setpa 设置的参数都是可以保存的。

setbp 可以设置启动命令（对于 UCLINUX 和 LINUX）,可以保存。

Usage : setpa -s[-r][-i][-ni][-nor][-nand] [address]

-s save address
-r run address
-i initrd save address
-ni disable initrd
-nor use nor flash to save
-nand use nand flash to save

-s 表示 mrun 运行的程序是存储器在 flash 的什么位置,

对于 nor flash 是 nor flash 中的地址,

对于 nand flash 1000 表示分区 0,2000 表示分区 2.

-r 表示存储的程序要读到 sdram 中什么位置再运行.对于 uClinux 是 c300000.

-i 表示 initrd 存储在 flash 的什么位置,如同-s.

-ni 表示取消 initrd

-nand 表示用 NAND FLASH 作为内核及 initrd 的存储介质.

bootkey 可设置 BIOS 复位运行后检查哪个按键状态来自动启动存储在 Flash 中的用户程序,即自动调用 mrun 指令,按键编号 1~4,状态 0 表示低启动,1 表示高启动。比如要在复为后检测到按键 3 为低时启动,可执行 bootkey 3 0。此参数也可保存,注意实现自动启动的前提是先烧写好 Flash 和用 setpa 命令设置好各个参数,bootkey 命令最后可带-b 参数,表示自动运行 boot 指令,缺省情况下是运行 mrun 指令。

nfpart 可在 Bios 中对 Nand Flash 简单分区,比如 Nand Flash 大小是 32M,要分为 0~0x30000,0x30000~0x200000,0x200000~0x800000,0x800000~0x1000000,0x1000000~0x2000000 这样 5 个分区,可以执行 nfpart 30000 200000 800000 1000000 2000000,分区最多为 8 个,分区参数可以保存。

nferase 可以擦除 NAND FLASH 分区,块有错误时会有提示。

nfprog 可以将下载的数据写入 NAND FLASH 分区,也可指定烧入数据的起始地址和长度,烧写有错误也会有提示。

nfload 可以将 NAND FLASH 分区的数据全部读入 sdram 中, 可以指定 sdram 地址和 NAND FLASH 分区。

senv 命令可以保存所有保存的参数到 FLASH 中, 下次复位运行 BIOS 后会调入这些参数。

defset 命令用来设置一些默认参数, 比如 NAND FLASH 的分区, Linux 的启动参数等等之类的。

注意: 有些命令不能在 YL9200 上使用, 比如 **usbload** 之类的。

第五章 烧写和启动 Linux

这一章节，主要介绍如何烧写 Linux 内核和 Linux 的根文件系统，主要是利用 BIOS 来进行烧写。

串口使用 DNW 工具。

5.1 通过 BIOS 烧写 Linux 内核

(1) 进入 BIOS 后，输入命令：

defset ;主要用来设置默认参数

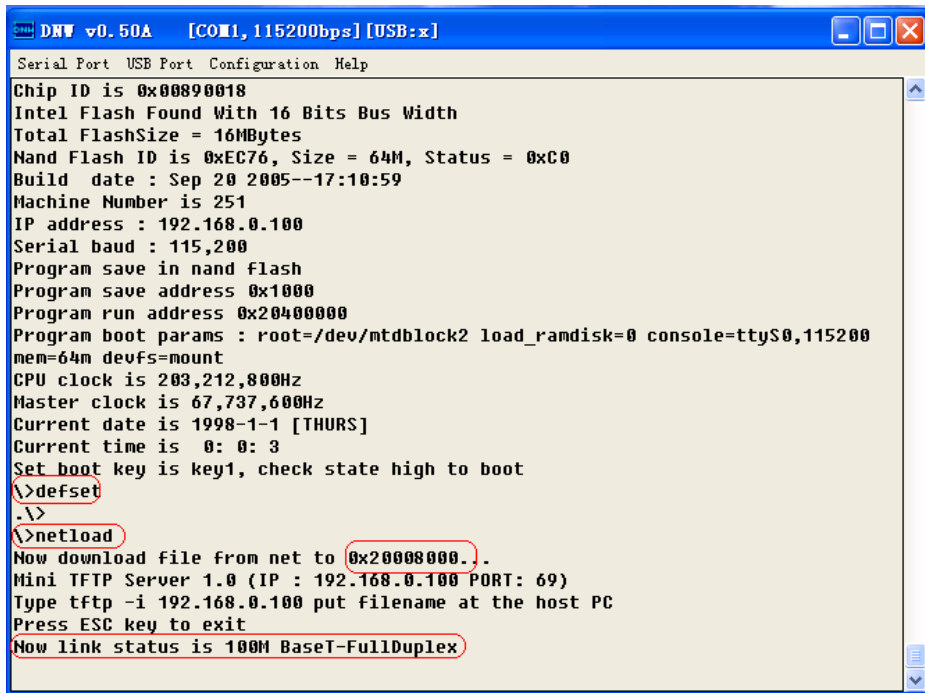
接着连好交叉网线，PC 机的地址要与开发板的地址在同一网段，（开发板的 IP 地址：192.168.0.100，PC 机的 IP 地址：192.168.0.7，当然你可以设置成其他的，最好与我们这里设置一样）。

(2) 接着输入命令：

netload ;启动 TFTP 来传输 Linux 内核文件

然后回车。

输入上面命令后，信息如下：



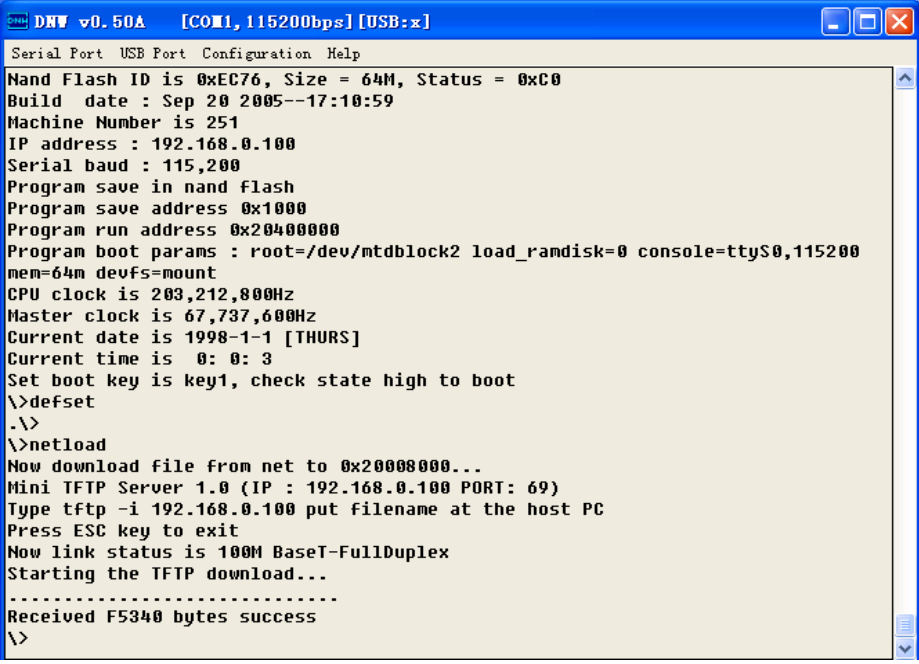
```
DNW v0.50A [COM1, 115200bps] [USB:x]
Serial Port  USB Port  Configuration  Help
Chip ID is 0x00890018
Intel Flash Found With 16 Bits Bus Width
Total FlashSize = 16MBytes
Nand Flash ID is 0xEC76, Size = 64M, Status = 0xC0
Build date : Sep 20 2005--17:10:59
Machine Number is 251
IP address : 192.168.0.100
Serial baud : 115,200
Program save in nand flash
Program save address 0x1000
Program run address 0x20400000
Program boot params : root=/dev/mtdblock2 load_ramdisk=0 console=ttyS0,115200
mem=64m devfs=mount
CPU clock is 203,212,800Hz
Master clock is 67,737,600Hz
Current date is 1998-1-1 [THURS]
Current time is 0: 0: 3
Set boot key is key1, check state high to boot
\>defset
.\>
.\>netload
Now download file from net to 0x20008000.-
Mini TFTP Server 1.0 (IP : 192.168.0.100 PORT: 69)
Type tftp -i 192.168.0.100 put filename at the host PC
Press ESC key to exit
Now link status is 100M BaseT-FullDuplex
```

上面的信息显示，程序将下载到 0x20008000 地址处（可以在 netload *****，来改变下载的地址）。IP 地址为 192.168.0.100。

(3) 点击光盘下的“目标代码”文件夹下的 zImage.bat 批处理文件，

注意：在进行 TFTP 传输操作前，要将防火墙关闭掉

点击 zImage.bat 批处理文件后，将会启动 TFTP 文件传输，传输成功后，将自动返回到 BIOS 的命令状态下。



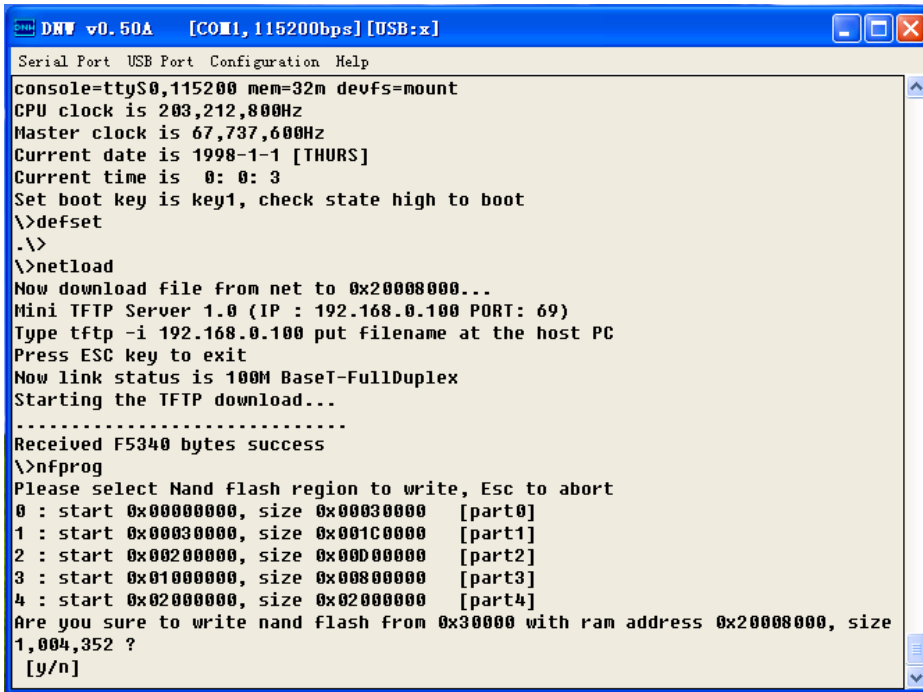
```
DNW v0.50A [COM1, 115200bps] [USB:x]
Serial Port  USB Port  Configuration  Help
Nand Flash ID is 0xEC76, Size = 64M, Status = 0xC0
Build date : Sep 20 2005--17:10:59
Machine Number is 251
IP address : 192.168.0.100
Serial baud : 115,200
Program save in nand flash
Program save address 0x1000
Program run address 0x20400000
Program boot params : root=/dev/mtdblock2 load_ramdisk=0 console=ttyS0,115200
mem=64m devfs=mount
CPU clock is 203,212,800Hz
Master clock is 67,737,600Hz
Current date is 1998-1-1 [THURS]
Current time is 0: 0: 3
Set boot key is key1, check state high to boot
\>defset
.\>
\>netload
Now download file from net to 0x20008000...
Mini TFTP Server 1.0 (IP : 192.168.0.100 PORT: 69)
Type tftp -i 192.168.0.100 put filename at the host PC
Press ESC key to exit
Now link status is 100M BaseT-FullDuplex
Starting the TFTP download...
.....
Received 5340 bytes success
\>
```

(4) 接着，输入命令：

nfprog ;用这个命令将下载的 Linux 内核（可以是其他程序）烧写到 NAND
; FLASH 分区

然后回车。

输入上述命令后，会提示选择要烧写的 NAND FLASH 分区，这时选择“1”，将 Linux 的内核烧写到 NAND FLASH 的分区 1 中。



```
DNW v0.50A [COM1,115200bps] [USB:x]
Serial Port  USB Port  Configuration  Help
console=ttyS0,115200 mem=32m devfs=mount
CPU clock is 203,212,800Hz
Master clock is 67,737,600Hz
Current date is 1998-1-1 [THURS]
Current time is 0: 0: 3
Set boot key is key1, check state high to boot
\>defset
.-\>
\>netload
Now download file from net to 0x20000000...
Mini TFTP Server 1.0 (IP : 192.168.0.100 PORT: 69)
Type tftp -i 192.168.0.100 put filename at the host PC
Press ESC key to exit
Now link status is 100M BaseT-FullDuplex
Starting the TFTP download...
.....
Received F5340 bytes success
\>nfprog
Please select Nand flash region to write, Esc to abort
0 : start 0x00000000, size 0x00030000 [part0]
1 : start 0x00030000, size 0x001C0000 [part1]
2 : start 0x00200000, size 0x00D00000 [part2]
3 : start 0x01000000, size 0x00800000 [part3]
4 : start 0x02000000, size 0x02000000 [part4]
Are you sure to write nand flash from 0x30000 with ram address 0x20000000, size
1,004,352 ?
[y/n]
```

接着在 “[y/n]” 提示下输入 “y”，烧写结束后，会自动返回到 BIOS 的命令状态下。

5.2 通过 BIOS 烧写根文件系统

5.2.1 不带 QT 图形界面的根文件系统 myroot.cramfs 的烧写

这一章节，主要介绍如何烧写不带 QT 图形界面的根文件系统。

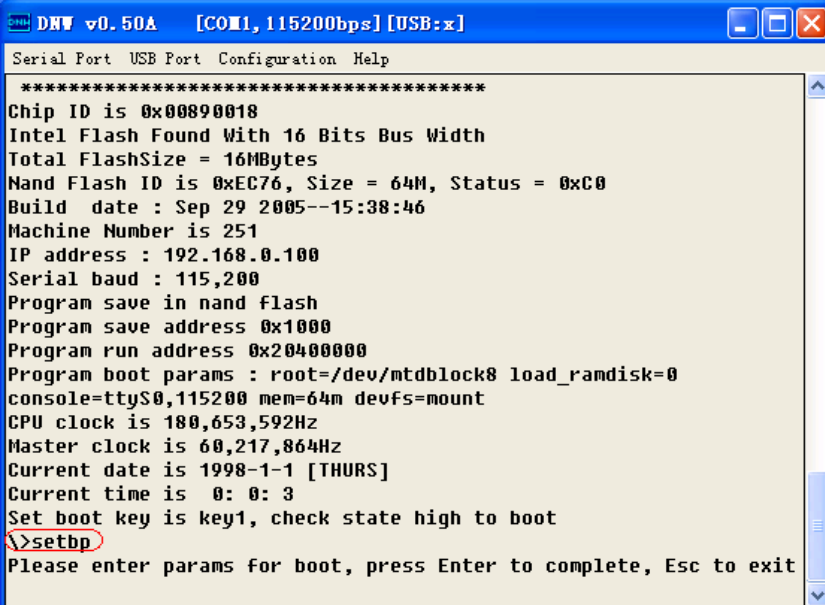
步骤与烧写 Linux 内核是一样的，不同的地方是点击的批处理文件是 myrootcramfs.BAT，烧写的分区是 2，其他过程是一样的。(注意，根文件系统烧写的区域不能有坏块，如果有可以换其他的分区，同时内核的启动参数 (/dev/mtdblockX) 也需要改动! [X=分区号+3])

这样内核和根文件系统烧写好，就可以启动 Linux 了。

设置内核传递参数，命令序列如下：

setbp ;

输入上面这个命令后, 会出现如下的界面:



The screenshot shows a terminal window titled "DNW v0.50A [COM1, 115200bps] [USB:x]". The window contains the following text:

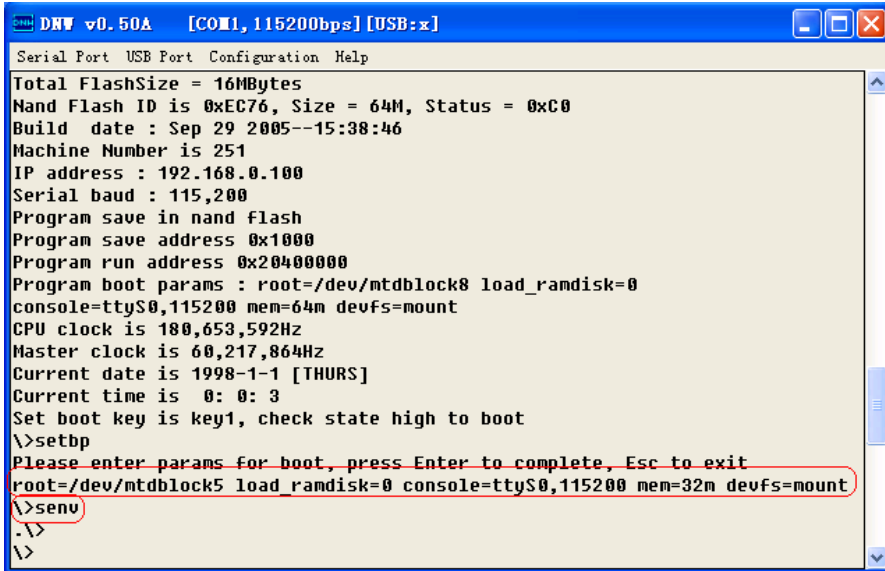
```
Serial Port  USB Port  Configuration  Help
*****
Chip ID is 0x00890018
Intel Flash Found With 16 Bits Bus Width
Total FlashSize = 16MBytes
Nand Flash ID is 0xEC76, Size = 64M, Status = 0xC0
Build date : Sep 29 2005--15:38:46
Machine Number is 251
IP address : 192.168.0.100
Serial baud : 115,200
Program save in nand flash
Program save address 0x1000
Program run address 0x20400000
Program boot params : root=/dev/mtdblock8 load_ramdisk=0
console=ttyS0,115200 mem=64m devfs=mount
CPU clock is 180,653,592Hz
Master clock is 60,217,864Hz
Current date is 1998-1-1 [THURS]
Current time is 0: 0: 3
Set boot key is key1, check state high to boot
\\>setbp
Please enter params for boot, press Enter to complete, Esc to exit
```

接着输入参数:

root=/dev/mtdblock5 load_ramdisk=0 console=ttyS0,115200 mem=64m devfs=mount

接着输入命令:

senv ;保存设置的参数

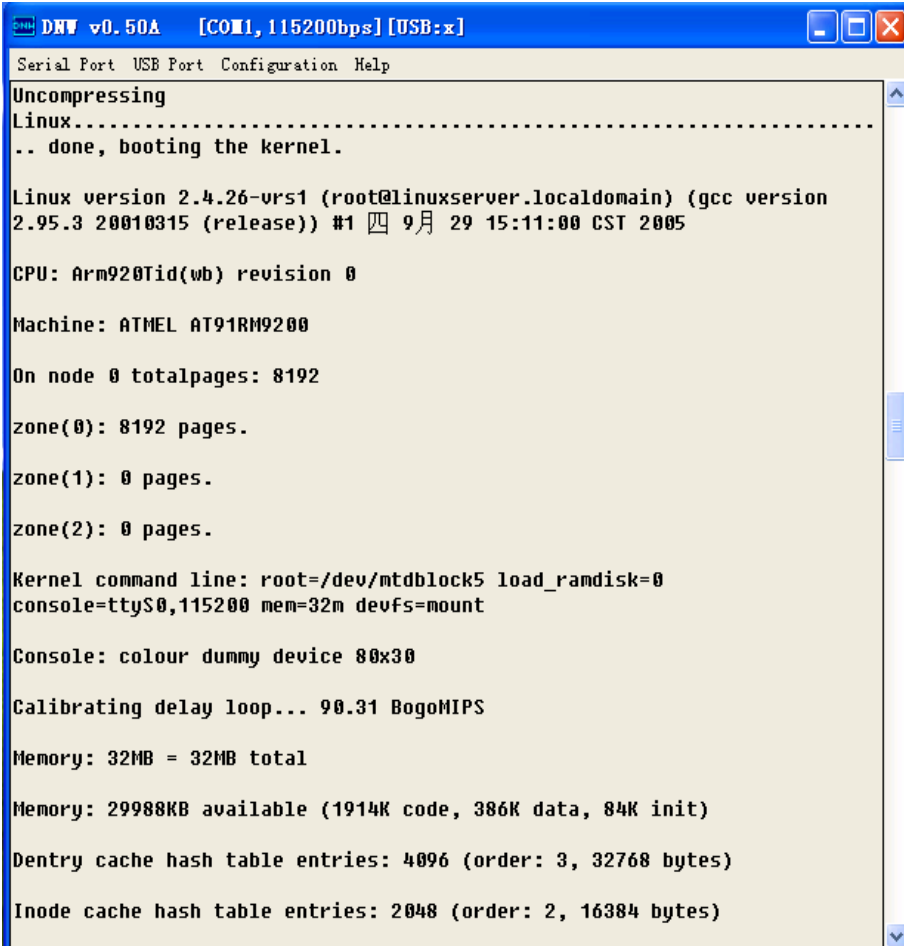
A screenshot of a terminal window titled "DNW v0.50A [COM1, 115200bps] [USB:x]". The window contains the following text:

```
Serial Port USB Port Configuration Help
Total FlashSize = 16MBytes
Nand Flash ID is 0xEC76, Size = 64M, Status = 0xC0
Build date : Sep 29 2005--15:38:46
Machine Number is 251
IP address : 192.168.0.100
Serial baud : 115,200
Program save in nand flash
Program save address 0x1000
Program run address 0x20400000
Program boot params : root=/dev/mtdblock8 load_ramdisk=0
console=ttyS0,115200 mem=64m devfs=mount
CPU clock is 180,653,592Hz
Master clock is 60,217,864Hz
Current date is 1998-1-1 [THURS]
Current time is 0: 0: 3
Set boot key is key1, check state high to boot
\\setbp
Please enter params for boot, press Enter to complete, Esc to exit
root=/dev/mtdblock5 load_ramdisk=0 console=ttyS0,115200 mem=32m devfs=mount
\\senv
.\\
\\>
```

在 BIOS 命令状态下，可以通过输入命令：

mrn ; 这个命令可以在 BIOS 下运行 Linux

这时将启动 Linux 操作，启动信息如下：



```
DNV v0.50A [COM1, 115200bps] [USB:x]
Serial Port  USB Port  Configuration  Help
Uncompressing
Linux..... .. done, booting the kernel.

Linux version 2.4.26-urs1 (root@linuxserver.localdomain) (gcc version
2.95.3 20010315 (release)) #1 四 9月 29 15:11:00 CST 2005

CPU: Arm920Tid(wb) revision 0

Machine: ATMEL AT91RM9200

On node 0 totalpages: 8192

zone(0): 8192 pages.

zone(1): 0 pages.

zone(2): 0 pages.

Kernel command line: root=/dev/mtdblock5 load_ramdisk=0
console=ttyS0,115200 mem=32m devfs=mount

Console: colour dummy device 80x30

Calibrating delay loop... 90.31 BogoMIPS

Memory: 32MB = 32MB total

Memory: 29988KB available (1914K code, 386K data, 84K init)

Dentry cache hash table entries: 4096 (order: 3, 32768 bytes)

Inode cache hash table entries: 2048 (order: 2, 16384 bytes)
```

5.2.2 带 QT 图形界面的根文件系统 myqt.cramfs 的烧写

这一章节，主要介绍如何烧写带 QT 图形界面的根文件系统。

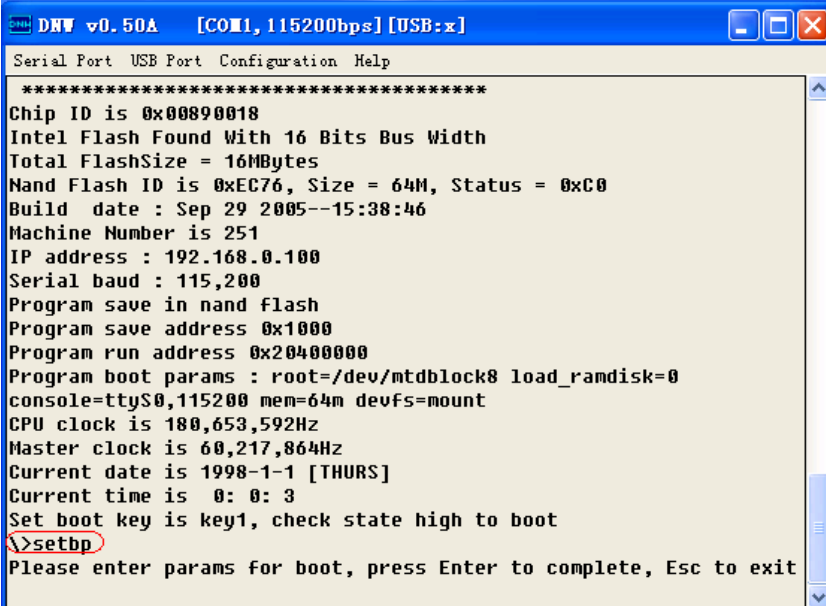
步骤与烧写 Linux 内核是一样的，不同的地方是点击的批处理文件是 myqt.cramfs.BAT，烧写的分区是 4，其他过程是一样的。(注意，根文件系统烧写的区域不能有坏块，如果有可以换其他的分区，同时内核的启动参数 (/dev/mtdblockX) 也需要改动! [X=分区号+3])

这样内核和根文件系统烧写好，就可以启动 Linux 了。

设置内核传递参数，命令序列如下：

setbp ;

输入上面这个命令后，会出现如下的界面：



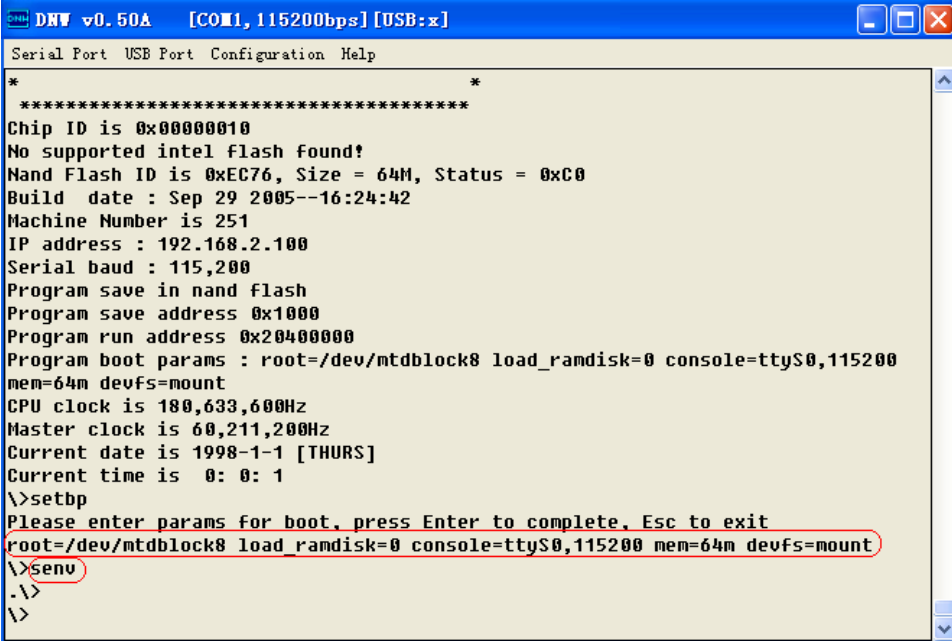
```
DNW v0.50A [COM1, 115200bps] [USB:x]
Serial Port  USB Port  Configuration  Help
*****
Chip ID is 0x00890018
Intel Flash Found With 16 Bits Bus Width
Total FlashSize = 16MBytes
Nand Flash ID is 0xEC76, Size = 64M, Status = 0xC0
Build date : Sep 29 2005--15:38:46
Machine Number is 251
IP address : 192.168.0.100
Serial baud : 115,200
Program save in nand flash
Program save address 0x1000
Program run address 0x20400000
Program boot params : root=/dev/mtdblock8 load_ramdisk=0
console=ttyS0,115200 mem=64m devfs=mount
CPU clock is 180,653,592Hz
Master clock is 60,217,864Hz
Current date is 1998-1-1 [THURS]
Current time is 0: 0: 3
Set boot key is key1, check state high to boot
\>setbp
Please enter params for boot, press Enter to complete, Esc to exit
```

接着输入参数：

root=/dev/mtdblock7 load_ramdisk=0 console=ttyS0,115200 mem=64m devfs=mount

接着输入命令：

senv ;保存设置的参数

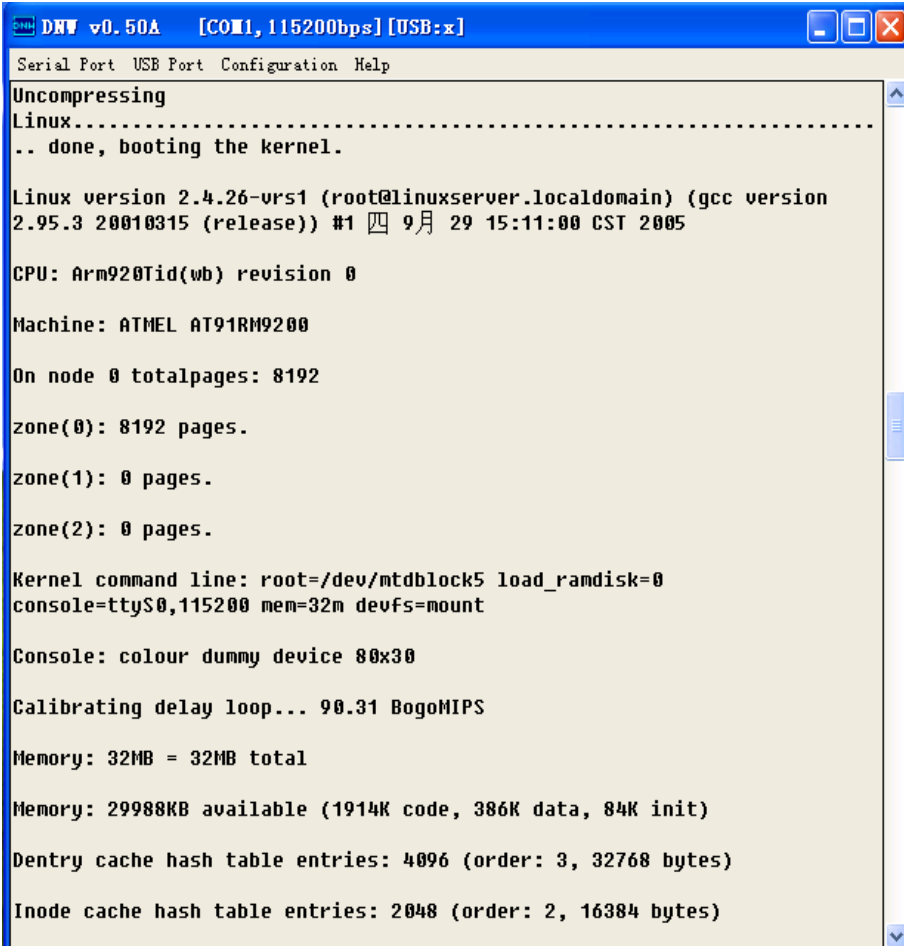


```
DHW v0.50A [COM1, 115200bps] [USB:x]
Serial Port  USB Port  Configuration  Help
*
*****
Chip ID is 0x00000010
No supported intel flash found!
Nand Flash ID is 0xEC76, Size = 64M, Status = 0xC0
Build date : Sep 29 2005--16:24:42
Machine Number is 251
IP address : 192.168.2.100
Serial baud : 115,200
Program save in nand flash
Program save address 0x1000
Program run address 0x20400000
Program boot params : root=/dev/mtdblock8 load_ramdisk=0 console=ttyS0,115200
mem=64m devfs=mount
CPU clock is 180,633,600Hz
Master clock is 60,211,200Hz
Current date is 1998-1-1 [THURS]
Current time is 0: 0: 1
\>setbp
Please enter params for boot, press Enter to complete, Esc to exit
root=/dev/mtdblock8 load_ramdisk=0 console=ttyS0,115200 mem=64m devfs=mount
\>seuv
.\>
\>
```

在 BIOS 命令状态下，可以通过输入命令：

mrun ; 这个命令可以在 BIOS 下运行 Linux

这时将启动 Linux 操作，启动信息如下：



```
DNU v0.50A [COM1, 115200bps] [USB:x]
Serial Port  USB Port  Configuration  Help
Uncompressing
Linux.....
.. done, booting the kernel.

Linux version 2.4.26-vrs1 (root@linuxserver.localdomain) (gcc version
2.95.3 20010315 (release)) #1 四 9月 29 15:11:00 CST 2005

CPU: Arm920Tid(wb) revision 0

Machine: ATMEL AT91RM9200

On node 0 totalpages: 8192

zone(0): 8192 pages.

zone(1): 0 pages.

zone(2): 0 pages.

Kernel command line: root=/dev/mtdblock5 load_ramdisk=0
console=ttyS0,115200 mem=32m devfs=mount

Console: colour dummy device 80x30

Calibrating delay loop... 90.31 BogoMIPS

Memory: 32MB = 32MB total

Memory: 29988KB available (1914K code, 386K data, 84K init)

Dentry cache hash table entries: 4096 (order: 3, 32768 bytes)

Inode cache hash table entries: 2048 (order: 2, 16384 bytes)
```

5.3 设置 Linux 自启动

将 Linux 内核和根文件系统烧写好后，现在来设置 Linux 自启动参数，这样的话，上电就可以自启动 Linux，或者自己的应用程序。

首先进入 BIOS 的命令状态，输入命令：

bootkey 1 1 ; 这个命令就可以设置 Linux 的自启动了

接着输入命令：

senv ; 保存设置的参数

当上述参数设置好，只要一上电，将会启动 Linux 系统，如果要重新回到 BIOS 的命令状态，需要上电听到蜂鸣器响一声的时候，按住开发板的 S3 按键。

设置 Linux 自启动后，接上你所设置的参数，就可以看到对应输出设备的右上角有一个器器企鹅的图标，说明在 Linux 下就只有支持 frambuffer，启动结束后会自动启动 QT 图形界面。

第六章 Linux 内核的编译及根文件系统的制作

注意：这一章节的操作是在 PC 的 Linux 操作系统下进行的
PC 机的 Linux 操作系统是 RED HAT 9.0。

6.1 建立交叉编译环境及编译 Linux 内核

在编译 LINUX 内核前先要在 PC 机的 LINUX 操作系统下安装 ARM 的交叉编译工具。在光盘上提供的编译工具是 cross-3.3.2 的编译器。安装方法是先进入/usr/local 目录，

```
cd /usr/local
```

建立一个 arm 的目录，

```
mkdir arm
```

再进入 arm 目录，

```
cd arm
```

将光盘下的“Linux 源码和工具”下的 cross-3.3.2.tar.bz2 编译工具拷贝到/usr/local/arm 目录（也就是刚刚建立 arm 的目录下）

在/usr/local/arm 目录下执行解压操作，

```
tar jxvf cross-3.3.2.tar.bz2
```

解压完成后会在 arm 目录下生成一个 3.3.2 的子目录，编译工具都解压在此目录下。要想把 arm 编译器的路径加在系统命令搜索路径中去，可以编辑/etc/bashrc 文件，在最后加上一行

```
export PATH=/usr/local/arm/2.95.3/bin:$PATH
```

安装好编译工具后，在某个分区内解压内核压缩包，比如在/home 分区内。

```
cd /home
```

然后将光盘下的“Linux 源码和工具”下的 linux-2.6.13-vrs1-hzh.tar.gz 源码包拷贝到/home 目录下。

然后执行解压命令：

```
tar zxvf linux-2.6.13-vrs1-hzh.tar.gz
```

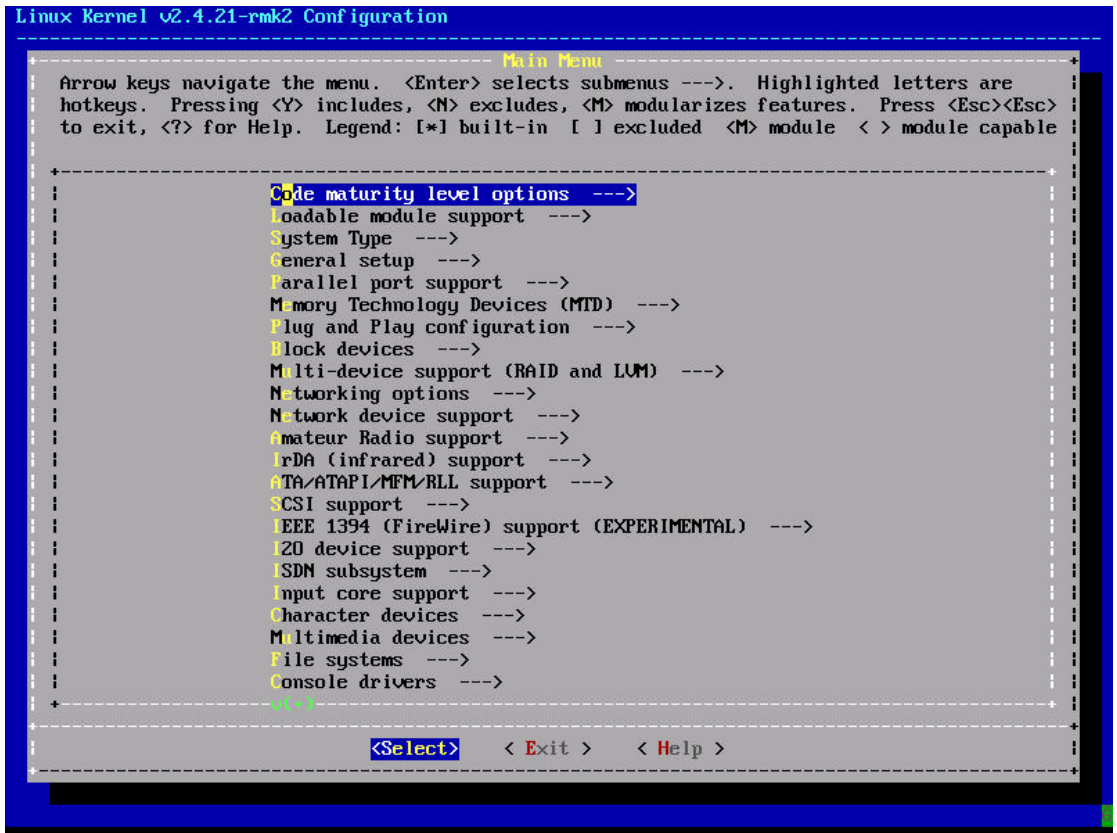
解压完后会在/home 目录下生成 linux-2.6.13-vrs1-hzh 的目录，LINUX 内核都包含在此目录下。再进入此目录

```
cd linux-2.6.13-vrs1-hzh
```

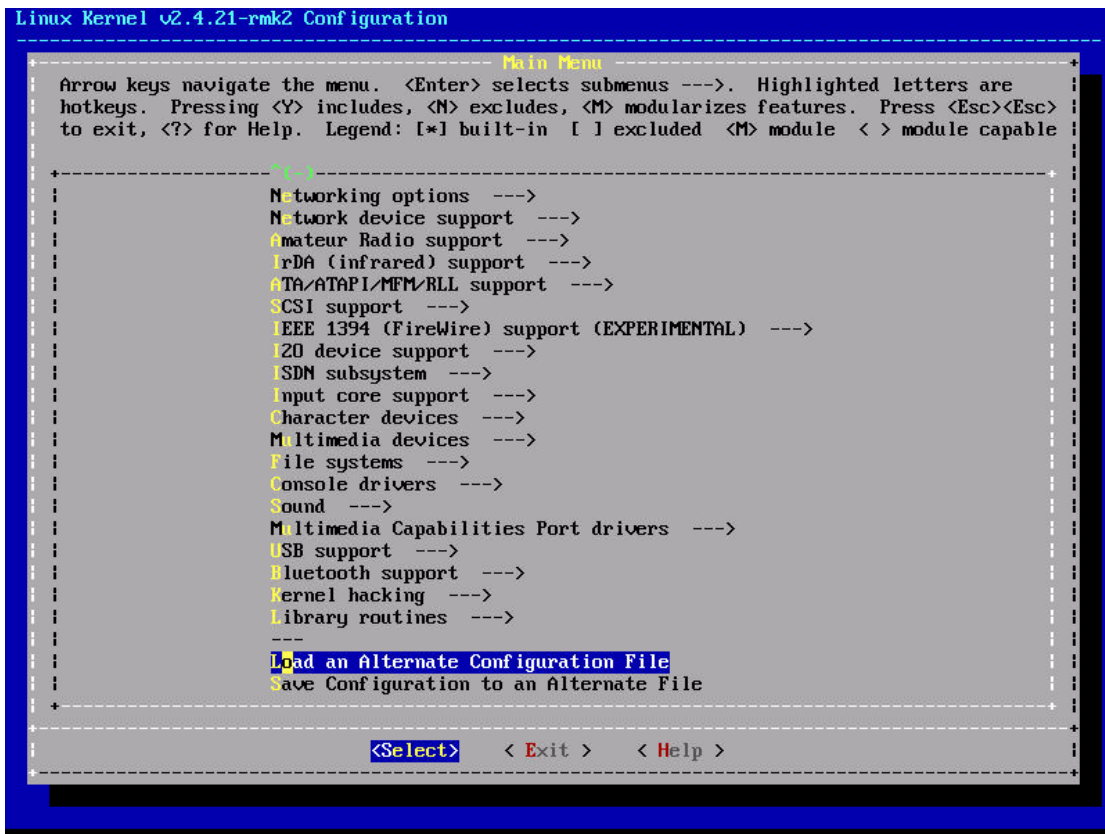
执行

```
make menuconfig
```

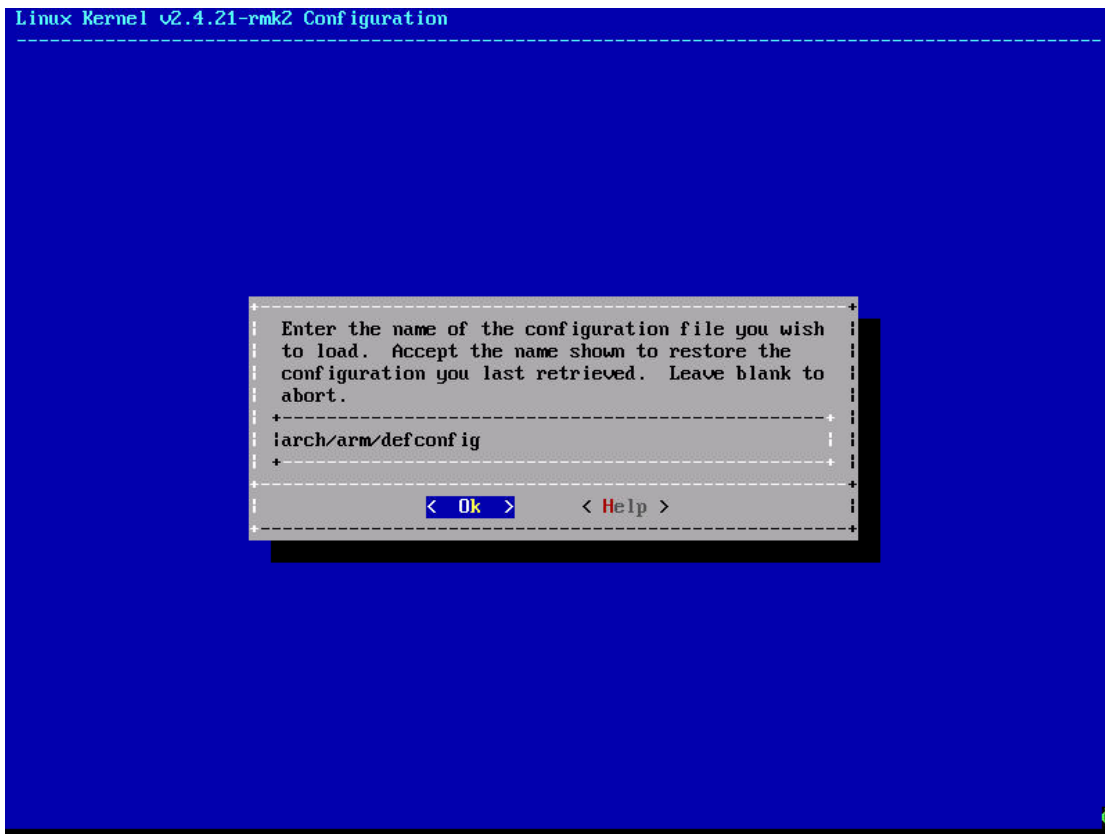
出现如下图的配置菜单



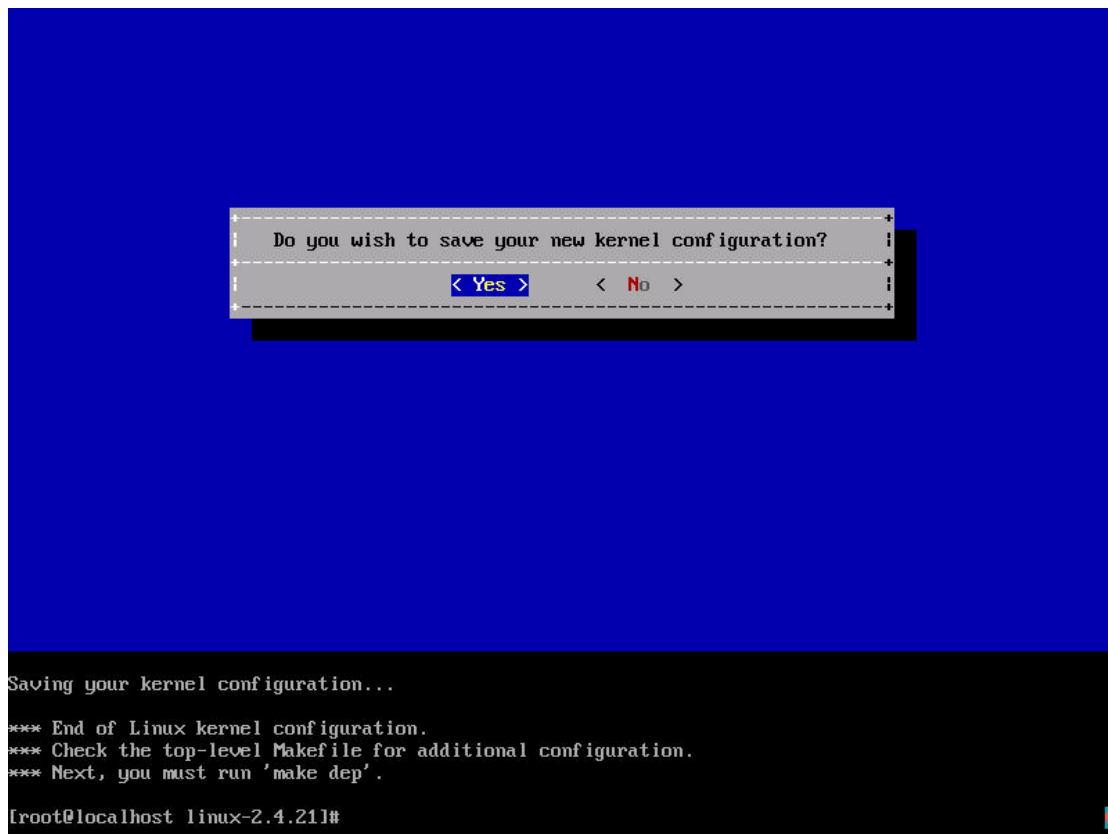
移到最下面，选择 Load an Alternate Configuration File



出现如下文件名输入框



将 arch/arm/defconfig 改为 kernel_9200.cfg，再按 Ok 退出
回到主菜单后再按右键移动到 Exit 退出配置，出现保存配置的确框



```
Do you wish to save your new kernel configuration?
< Yes > < No >

Saving your kernel configuration...
*** End of Linux kernel configuration.
*** Check the top-level Makefile for additional configuration.
*** Next, you must run 'make dep'.
[root@localhost linux-2.4.21]#
```

选到 Yes 后回车就保存配置了。

接下来执行 make dep，编译文件依赖关系

```
Saving your kernel configuration...

*** End of Linux kernel configuration.
*** Check the top-level Makefile for additional configuration.
*** Next, you must run 'make dep'.

[root@localhost linux-2.4.21]# make dep
gcc -Wall -Wstrict-prototypes -O2 -fomit-frame-pointer -o scripts/mkdep scripts/mkdep.c
  Making asm-arm/arch -> asm-arm/arch-at91rm9200 symlink
  Making asm-arm/proc -> asm-arm/proc-armv symlink
rm -f include/asm
( cd include ; ln -sf asm-arm asm)
make[1]: Entering directory `/work/linux-2.4.21/arch/arm/tools'
/work/linux-2.4.21/scripts/mkdep -D__KERNEL__ -I/work/linux-2.4.21/include -Wall -Wstrict-prototypes
  -Wno-trigraphs -Os -fno-strict-aliasing -fno-common -Uarm -fno-common -pipe -mapcs-32 -D__LINUX_ARM
_ARCH__=4 -march=armv4 -mtune=arm9tdmi -mshort-load-bytes -msoft-float -Uarm -- getconstants.c \
sed s,getconstants.o,constants.h, > .depend
make all
make[2]: Entering directory `/work/linux-2.4.21/arch/arm/tools'
awk -f gen-mach-types mach-types > /work/linux-2.4.21/include/asm-arm/mach-types.h
/usr/local/arm/2.95.3/bin/arm-linux-gcc -D__KERNEL__ -I/work/linux-2.4.21/include -Wall -Wstrict-pro
totypes -Wno-trigraphs -Os -fno-strict-aliasing -fno-common -Uarm -fno-common -pipe -mapcs-32 -D__LI
NUX_ARM_ARCH__=4 -march=armv4 -mtune=arm9tdmi -mshort-load-bytes -msoft-float -Uarm -S -o constants.
h.tmp.1 getconstants.c
```

make dep 结束后执行 make zImage

```

make -C arch/arm/nwfppe fastdep
make[2]: Entering directory `/work/linux-2.4.21/arch/arm/nwfppe'
/work/linux-2.4.21/scripts/mkdep -D__KERNEL__ -I/work/linux-2.4.21/include -Wall -Wstrict-prototypes
-Wno-trigraphs -Os -fno-strict-aliasing -fno-common -Uarm -fno-common -pipe -mapcs-32 -D__LINUX_ARM
_ARCH__=4 -march=armv4 -mtune=arm9tdmi -mshort-load-bytes -msoft-float -Uarm -nostdinc -iwithprefix
include -- ARM-gcc.h double_cpdo.c entry26.S entry.S extended_cpdo.c fpa11.c fpa11_cpdo.c fpa11_cpdo
t.c fpa11_cpdt.c fpa11.h fpmodule.c fpmodule.h fpopcode.c fpopcode.h fpsr.h milieu.h single_cpdo.c s
oftfloat.c softfloat.h > .depend
make[2]: Leaving directory `/work/linux-2.4.21/arch/arm/nwfppe'
make -C arch/arm/fastfppe fastdep
make[2]: Entering directory `/work/linux-2.4.21/arch/arm/fastfppe'
/work/linux-2.4.21/scripts/mkdep -D__KERNEL__ -I/work/linux-2.4.21/include -Wall -Wstrict-prototypes
-Wno-trigraphs -Os -fno-strict-aliasing -fno-common -Uarm -fno-common -pipe -mapcs-32 -D__LINUX_ARM
_ARCH__=4 -march=armv4 -mtune=arm9tdmi -mshort-load-bytes -msoft-float -Uarm -nostdinc -iwithprefix
include -- CPDO.S CPDT.S CPRT.S entry.S module.c > .depend
make[2]: Leaving directory `/work/linux-2.4.21/arch/arm/fastfppe'
make -C arch/arm/common fastdep
make[2]: Entering directory `/work/linux-2.4.21/arch/arm/common'
/work/linux-2.4.21/scripts/mkdep -D__KERNEL__ -I/work/linux-2.4.21/include -Wall -Wstrict-prototypes
-Wno-trigraphs -Os -fno-strict-aliasing -fno-common -Uarm -fno-common -pipe -mapcs-32 -D__LINUX_ARM
_ARCH__=4 -march=armv4 -mtune=arm9tdmi -mshort-load-bytes -msoft-float -Uarm -nostdinc -iwithprefix
include -- pcipool.c > .depend
make[2]: Leaving directory `/work/linux-2.4.21/arch/arm/common'
make[1]: Leaving directory `/work/linux-2.4.21'
scripts/mkdep -- `find /work/linux-2.4.21/include/asm /work/linux-2.4.21/include/linux /work/linux-2
.4.21/include/scsi /work/linux-2.4.21/include/net /work/linux-2.4.21/include/math-emu \{ -name SCCS
-o -name .svn \} -prune -o -follow -name \*.h ! -name modversions.h -print` > .hdepend
scripts/mkdep -- init/*.c > .depend
[root@localhost linux-2.4.21]# make zImage
gcc -Wall -Wstrict-prototypes -O2 -fomit-frame-pointer -o scripts/split-include scripts/split-includ
e.c
scripts/split-include include/linux/autoconf.h include/config
/usr/local/arm/2.95.3/bin/arm-linux-gcc -D__KERNEL__ -I/work/linux-2.4.21/include -Wall -Wstrict-pro
totypes -Wno-trigraphs -Os -fno-strict-aliasing -fno-common -Uarm -fno-common -pipe -mapcs-32 -D__LI
NUX_ARM_ARCH__=4 -march=armv4 -mtune=arm9tdmi -mshort-load-bytes -msoft-float -Uarm -DKBUILD_BASEN
AME=main -c -o init/main.o init/main.c

```

编译结束后在 arch/arm/boot 目录下得到压缩内核文件 zImage，

```

h/arm/kernel/init_task.o init/main.o init/version.o init/do_mounts.o \
--start-group \
  arch/arm/kernel/kernel.o arch/arm/mm/mm.o arch/arm/mach-at91rm9200/at91rm9200.o kernel/kerne
l.o mm/mm.o fs/fs.o ipc/ipc.o arch/arm/common/nopci.o \
  drivers/i2c/i2c.o drivers/serial/serial.o drivers/char/char.o drivers/block/block.o drivers
/misc/misc.o drivers/net/net.o drivers/ide/idedriver.o drivers/scsi/scsidrv.o drivers/cdrom/driver.o
drivers/mtd/mtdlink.o drivers/usb/usbdrv.o drivers/media/media.o drivers/input/inputdrv.o drivers/a
t91/at91drv.o \
  net/network.o \
  arch/arm/nwfp/math-emu.o arch/arm/lib/lib.a /work/linux-2.4.21/lib/lib.a \
--end-group \
-o vmlinux
/usr/local/arm/2.95.3/bin/arm-linux-gcc -o vmlinux `grep -v '\(compiled\)\|\(\.o\$\)\|\([aUw] \)\|\(\.
.ng\)\|\(LASH|RL|D|I\)' `i sort > System.map
make[1]: Entering directory `/work/linux-2.4.21/arch/arm/boot'
make[2]: Entering directory `/work/linux-2.4.21/arch/arm/boot/compressed'
/usr/local/arm/2.95.3/bin/arm-linux-gcc -D __ASSEMBLY__ -D __KERNEL__ -I/work/linux-2.4.21/include -ma
pcs-32 -D __LINUX_ARM_ARCH__=4 -march=armv4 -msoft-float -traditional -c head.S
/usr/local/arm/2.95.3/bin/arm-linux-gcc -D __KERNEL__ -I/work/linux-2.4.21/include -O2 -DSTDC_HEADERS
-maps-32 -D __LINUX_ARM_ARCH__=4 -march=armv4 -mtune=arm9tdmi -mshort-load-bytes -msoft-float -Uarm
-fpic -Uarm -D __KERNEL__ -I/work/linux-2.4.21/include -c -o misc.o misc.c
/usr/local/arm/2.95.3/bin/arm-linux-gcc -D __ASSEMBLY__ -D __KERNEL__ -I/work/linux-2.4.21/include -ma
pcc-32 -D __LINUX_ARM_ARCH__=4 -march=armv4 -msoft-float -c -o head-at91rm9200.o head-at91rm9200.S
/usr/local/arm/2.95.3/bin/arm-linux-objcopy -O binary -R .note -R .comment -S /work/linux-2.4.21/vml
inux piggy
gzip -9 < piggy > piggy.gz
/usr/local/arm/2.95.3/bin/arm-linux-ld -r -o piggy.o -b binary piggy.gz
rm -f piggy piggy.gz
/usr/local/arm/2.95.3/bin/arm-linux-ld -p -X -T vmlinux.lds head.o misc.o head-at91rm9200.o piggy.o
/usr/local/arm/2.95.3/lib/gcc-lib/arm-linux/2.95.3/libgcc.a -o vmlinux
make[2]: Leaving directory `/work/linux-2.4.21/arch/arm/boot/compressed'
/usr/local/arm/2.95.3/bin/arm-linux-objcopy -O binary -R .note -R .comment -S compressed/vmlinux zIm
age
make[1]: Leaving directory `/work/linux-2.4.21/arch/arm/boot'
[root@localhost linux-2.4.21]# ls arch/arm/boot/zImage -l
-rwxr-xr-x 1 root root 828536 Jun 29 14:23 arch/arm/boot/zImage
[root@localhost linux-2.4.21]#

```

可以在 BIOSBOX 里将此文件下载后烧入到 NANDFLASH 里去或直接运行，注意下载后
要直接运行时指定地址为 20400000，如 comrun 20400000、rxrun 20400000、netrun 20400000
等。

6.2 制作 cramfs 根文件系统

上面的过程主要用于调试，掉电之后不能保存，要将程序固化，需要将该程序和模块添
加到根文件系统中。这里我们将以 YL9200 光盘附带的 myroot.cramfs（在“目标代码”
文件夹下）根文件系统的添加为例。

(1) 将 myroot.cramfs 拷贝到任意目录下

(2) 在该目录下建立两个文件：

mkdir chang

mkdir guo

(3) 将 myroot.cramfs 挂接到 chang 目录

`mount myroot.cramfs chang -o loop`

(4) 将 chang 目录下的内容压缩

cd chang

tar -cvf /chang 的上一级目录的绝对路径/1.tar ./

这样将在 chang 的上一级目录产生一个 1.tar 的包

(5) 将包解压到 guo 目录下。

`umount chang` ; 卸载挂载

cd .. ; 进入上一级目录

mv 1.tar guo ;

cd guo ;

tar -xvf 1.tar ; 将打包的根文件系统的里的内容解压

rm 1.tar

(6) 经过上面的步骤就可以将自己的驱动和应用程序添加到 cramfs 根文件系统中了

现在将开始制作 cramfs 根文件系统

先将 mkcramfs 文件拷贝到 guo 所在的目录

在这个目录下运行命令：

mkcramfs guo myroot1.cramfs

运行成功后，会在该目录下生成 myroot1.cramfs 根文件系统

- (7) 根文件系统制作成功后，就可以将 myroot1.cramfs 烧写到相应的地方，关于根文件系统的烧写，在“YL9200 使用手册”中有如何烧写根文件系统的说明。
- (8) 上面的步骤教你如何将自己的驱动和引用程序添加到根文件系统中。

6.3 编译一个 HELLO 的 DEMO 程序

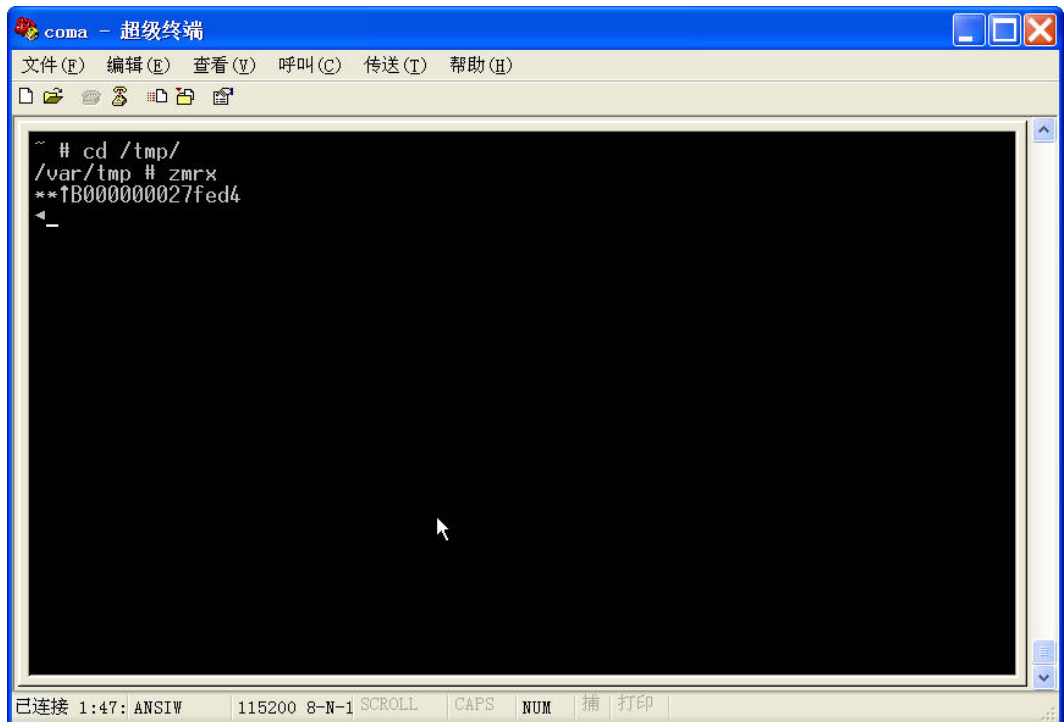
先在 Linux 下用文本编辑器编辑一个简单的 C 程序。

```
#include <stdio.h>
int main()
{
    printf("hello \n");
    return 0;
}
```

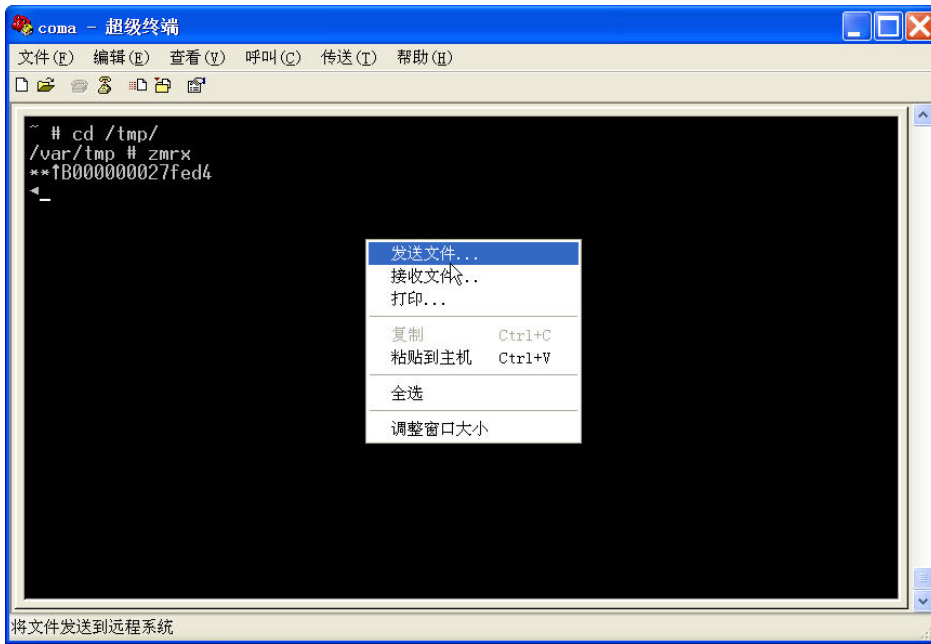
保存退出后，先用 GCC 编译并运行一下，之后再用 arm-linux-gcc 编译此程序，最后生成的可执行程序可以下载到开发板上运行。

编译 hello 的命令：

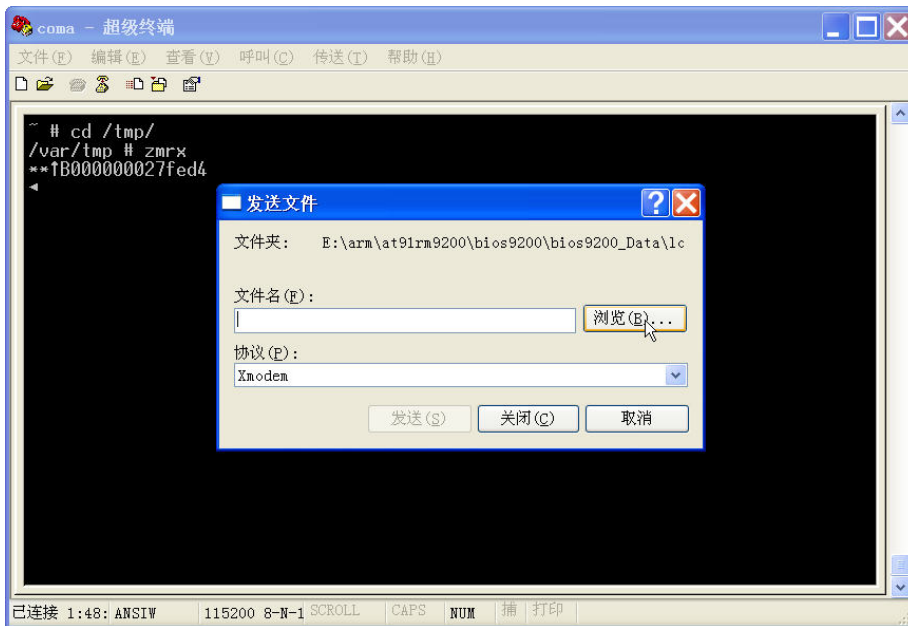
```
arm-linux-gcc -Wall -Os -s -o hello hello.c
```

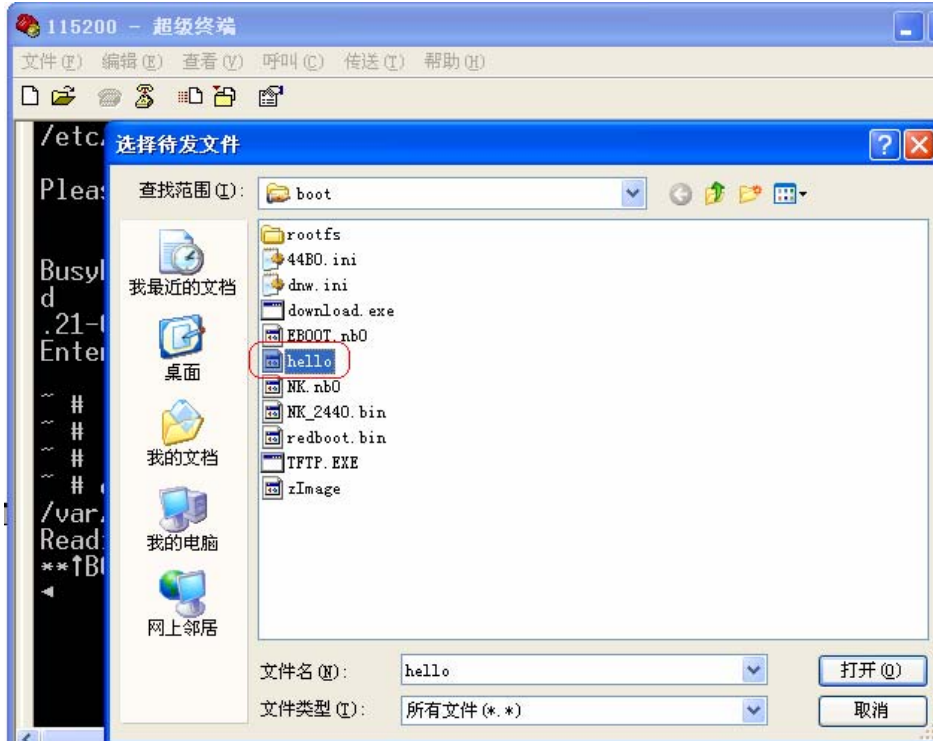
在超级终端里点击鼠标右键



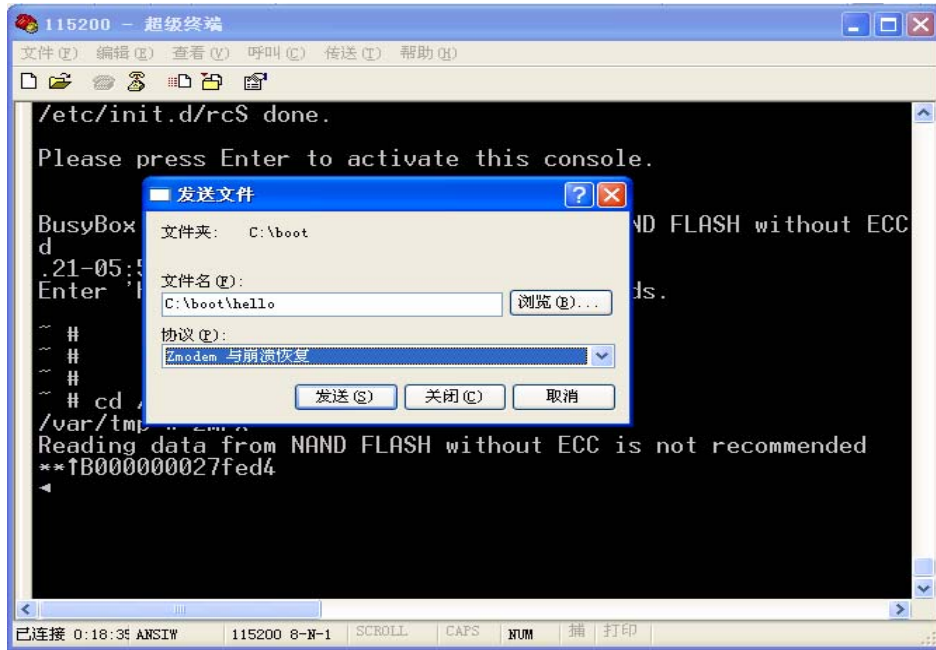
选择发送文件



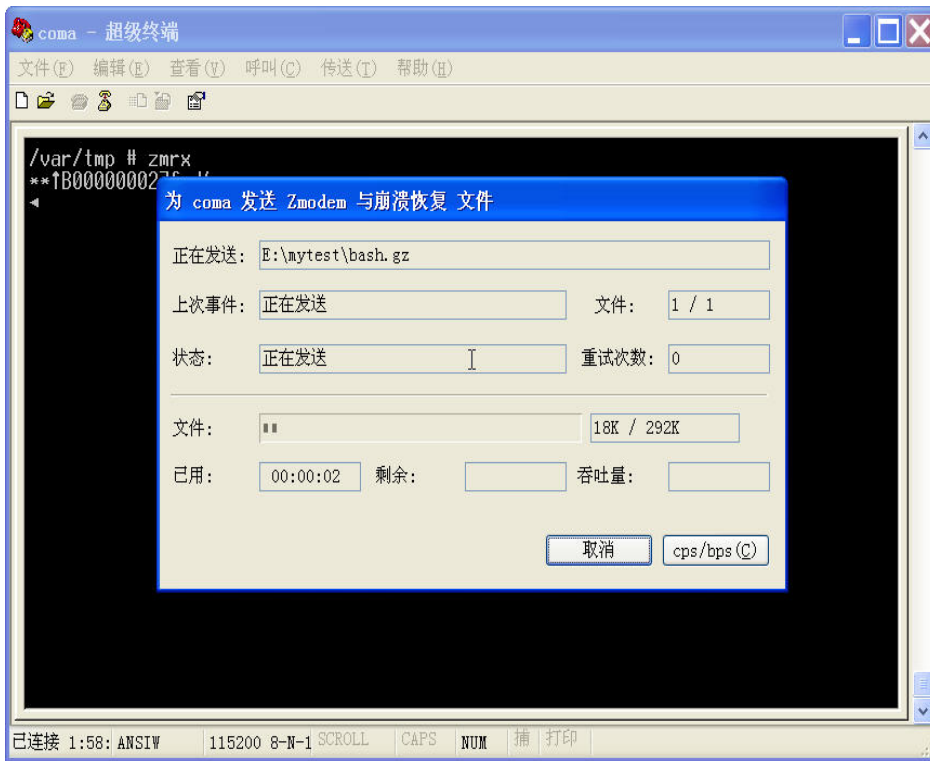
再选择浏览找到要发送的文件 hello



回到上一个对话框，选择以 **Zmodem 与崩溃恢复** 方式发送，设置好后按发送



下图为发送界面



发送完毕后，它会自动结束。返回到 shell 状态。

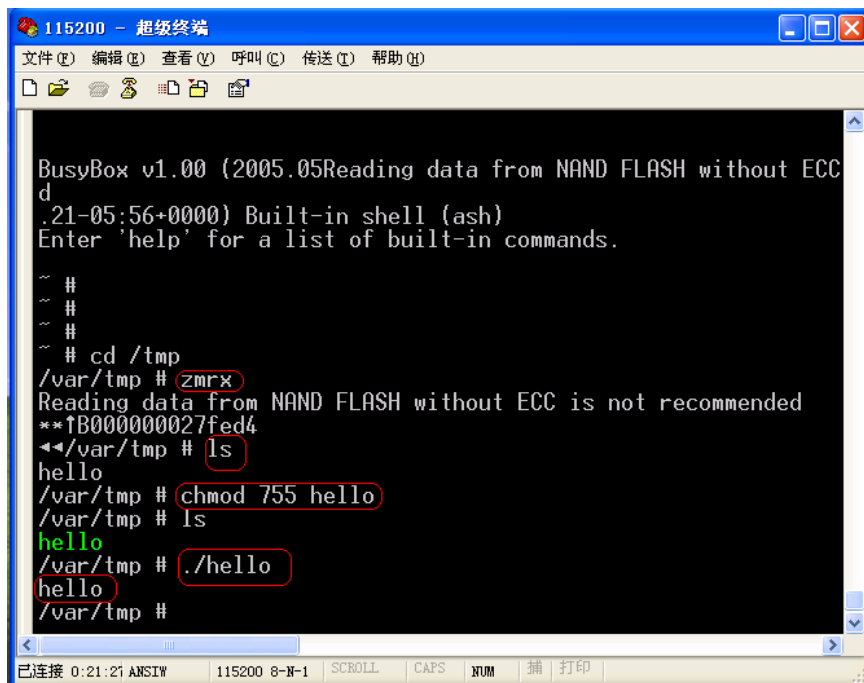
Hello 下载完成后，现在可以执行它了。

执行命令如下：

```
chmod 755 hello
```

```
./hello
```

这时可以看到 hello 运行的情况。



```
BusyBox v1.00 (2005.05Reading data from NAND FLASH without ECC
d
.21-05:56+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

~ #
~ #
~ #
~ # cd /tmp
/var/tmp # zmrxx
Reading data from NAND FLASH without ECC is not recommended
**↑B000000027fed4
←/var/tmp # ls
hello
/var/tmp # chmod 755 hello
/var/tmp # ls
hello
/var/tmp # ./hello
hello
/var/tmp #
```

注意，接收文件的目录必须是可写的，并且此目录下不能存在与要接收的文件名称相同的文件，有的话先删除或改名。另外 BUSYBOX 自带的 rx 命令也是一个接收程序，但是使用 xmodem 协议。也可以使用 ftp 的方法进行传送文件，或用 U 盘。

第七章 YL9200 开发平台出厂设置

7.1. 标配三星 3.5 寸屏用户

- 1、 复位让系统进行 BIOS 命令提示状态；
- 2、 在 BIOS 命令提示状态下输入 defset，然后通过网络传输 **Linux 内核 zImage** 到板子中的 SDRAM 内存中，传输完毕后自动返回命令提示状态下输入 nfprog 回车后出来烧写到的 Nand Flash 分区的选择，这里 zImage 对应选择 1，所以输入“1”选择第二个区块，输入“y”确认将它烧写到 nandflash (K9F1208) 中。(具体如何烧写 zImage，请看 5.1 章节)
- 3、 在 BIOS 命令提示状态下通过网络传输 **Linux 内核 myqt. cramfs** 到板子中的 SDRAM 内存中，传输完毕后自动返回命令提示状态下输入 nfprog 回车后出来烧写到的 Nand Flash 分区的选择，这里 myqt. cramfs 对应选择 4，所以输入“4”选择第五个区块，输入“y”确认将它烧写到 nandflash (K9F1208) 中。(具体如何烧写 myqt. cramfs，请看 5.2.2 章节)
- 4、 参数的设置：在 BIOS 命令提示状态下输入 bootkey 1 1，回车再输入 senv 保存参数，复位系统自启动 Linux，这时可在三星屏上显示 QT 图形界面了。

7.2. 非标配夏普 3.5 寸屏用户

- 1、 复位让系统进行 BIOS 命令提示状态；
- 2、 在 BIOS 命令提示状态下输入 defset，然后通过网络传输 **Linux 内核 zImage** 到板子中的 SDRAM 内存中，传输完毕后自动返回命令提示状态下输入 nfprog 回车后出来烧写到的 Nand Flash 分区的选择，这里 zImage 对应选择 1，所以输入“1”选择第二个区块，输入“y”确认将它烧写到 nandflash (K9F1208) 中。(具体如何烧写 zImage，请看 5.1 章节)
- 3、 在 BIOS 命令提示状态下通过网络传输 **Linux 内核 myqt. cramfs** 到板子中的 SDRAM 内存中，传输完毕后自动返回命令提示状态下输入 nfprog 回车后出来烧写到的 Nand Flash 分区的选择，这里 myqt. cramfs 对应选择 4，所以输入“4”选择第五个区块，输入“y”确认将它烧写到 nandflash (K9F1208) 中。(具体如何烧写 myqt. cramfs，请看 5.2.2 章节)
- 4、 参数的设置：在 BIOS 命令提示状态下输入 setbp，进入 Linux 启动参数设置，输入“

root=/dev/mtdblock7 load_ramdisk=0 console=ttyS0,115200 mem=64m devfs=mount video=fb:shp240”即把参数 video=fb:ct35tf05 改成 video=fb:shp240,回车再输入 senv 保存参数。

在 BIOS 命令提示状态下输入 bootkey 1 1, 回车再输入 senv 保存参数, 复位系统自启动 Linux, 这时可在三星屏上显示 QT 图形界面了。

7.3.非标配元太 6.4 寸屏（夏普 8 寸屏、夏普 10.4 屏）用户

- 1、复位让系统进行 BIOS 命令提示状态;
- 2、在 BIOS 命令提示状态下输入 defset, 然后通过网络传输 Linux 内核 zImage 到板子中的 SDRAM 内存中, 传输完毕后自动返回命令提示状态下输入 nfprog 回车后出来烧写到的 Nand Flash 分区的选择, 这里 zImage 对应选择 1, 所以输入“1”选择第二个区块, 输入“y”确认将它烧写到 nandflash (K9F1208) 中。(具体如何烧写 zImage, 请看 5.1 章节)
- 3、在 BIOS 命令提示状态下通过网络传输 Linux 内核 myqt.cramfs 到板子中的 SDRAM 内存中, 传输完毕后自动返回命令提示状态下输入 nfprog 回车后出来烧写到的 Nand Flash 分区的选择, 这里 myqt.cramfs 对应选择 4, 所以输入“4”选择第五个区块, 输入“y”确认将它烧写到 nandflash (K9F1208) 中。(具体如何烧写 myqt.cramfs, 请看 5.2.2 章节)
- 4、参数的设置: 在 BIOS 命令提示状态下输入 setbp, 进入 Linux 启动参数设置, 输入“root=/dev/mtdblock7 load_ramdisk=0 console=ttyS0,115200 mem=64m devfs=mount”即把参数 video=fb:ct35tf05 去掉, 回车再输入 senv 保存参数。

在 BIOS 命令提示状态下输入 bootkey 1 1, 回车再输入 senv 保存参数, 复位系统自启动 Linux, 这时可在三星屏上显示 QT 图形界面了。

7.4.输出 VGA 显示用户

- 1、复位让系统进行 BIOS 命令提示状态;
- 2、在 BIOS 命令提示状态下输入 defset, 然后通过网络传输 Linux 内核 zImage 到板子中的 SDRAM 内存中, 传输完毕后自动返回命令提示状态下输入 nfprog 回车后出来烧写到的

Nand Flash 分区的选择，这里 zImage 对应选择 1，所以输入“1”选择第二个区块，输入“y”确认将它烧写到 nandflash (K9F1208) 中。(具体如何烧写 zImage，请看 5.1 章节)

- 3、在 BIOS 命令提示状态下通过网络传输 **Linux 内核 myqt.cramfs** 到板子中的 SDRAM 内存中，传输完毕后自动返回命令提示状态下输入 nfp prog 回车后出来烧写到的 Nand Flash 分区的选择，这里 myqt.cramfs 对应选择 4，所以输入“4”选择第五个区块，输入“y”确认将它烧写到 nandflash (K9F1208) 中。(具体如何烧写 myqt.cramfs，请看 5.2.2 章节)
- 4、参数的设置：在 BIOS 命令提示状态下输入 setbp，进入 Linux 启动参数设置，输入“root=/dev/mtdblock7 load_ramdisk=0 console=ttyS0,115200 mem=64m devfs=mount video=fb:vga”即把参数 video=fb:ct35tf05 改成 video=fb:vga，回车再输入 senv 保存参数。

在 BIOS 命令提示状态下输入 bootkey 1 1，回车再输入 senv 保存参数，复位系统自启动 Linux，这时可在三星屏上显示 QT 图形界面了。

附录一 YL9200 开发系统 Qt 嵌入式图形开发

基础篇

Qt 是 Trolltech 公司的一个标志性产品。Trolltech 公司 1994 年成立于挪威，但是公司的核心开发团队已经在 1992 年开始了 Qt 产品的研发，并于 1995 年推出了 Qt 的第一个商业版，直到现在 Qt 已经被世界各地的跨平台软件开发人员使用，而 Qt 的功能也得到了不断的完善和提高。

Qt 是一个支持多操作系统平台的应用程序开发框架，它的开发语言是 C++。Qt 最初主要是为跨平台的软件开发者提供统一的，精美的图形用户编程接口，但是现在它也提供了统一的网络和数据库操作的编程接口。正如微软当年为操作系统提供了友好，精致的用户界面一样，今天由于 Trolltech 的跨平台开发框架 Qt 的出现，也使得 UNIX、LINUX 这些操作系统以更加方便、精美的人机界面走近普通用户。

Qt 是以工具开发包的形式提供给开发者的，这些工具开发包包括了图形设计器，Makefile 制作工具，字体国际化工具，Qt 的 C++类库等等；谈到 C++的类库我们自然会想到 MFC，是的，Qt 的类库也是等价于 MFC 的开发库，但是 Qt 的类库是支持跨平台的类库，也就是说 Qt 类库封装了适应不同操作系统的访问细节，这正是 Qt 的魅力所在。

目前，Qt 可以支持的操作系统平台如下：

- ◆ MS/Windows 95、Windows 98、WindowsNT 4.0、Windows 2000、Windows XP；
- ◆ Unix/X11 Linux、Sun Solaris、HP-UX、Compaq True64Unix、IBM AIX、SGI IRIX 和很多其它 X11 平台；
- ◆ Macintoshi Mac OSX；

◆ 嵌入式的，包含有 FramBuffer 的 Linux 平台。

Qt 的资源

trolltech 的主页: <http://www.trolltech.com/>

支持匿名访问的 FTP: <ftp://ftp.trolltech.com>

新闻组服务器: nntp.trolltech.com

非官方的 Qt 文档中文翻译小组: <http://www.qiliang.net/qt/index.html>

1. 认识 Qt/Embedded 嵌入式工具开发包

1.1 介绍

Qt/Embedded 是一个为嵌入式设备上的图形用户接口和应用开发而订做的C++工具开发包。它通常可以运行在多种不同的处理器上部署的嵌入式Linux操作系统上。如果不考虑X窗口系统的需要,居于Qt/Embedded 的应用程序可以直接对缓冲帧进行写操作。除了类库以外, Qt/Embedded还包括了几个提高开发速度的工具, 使用标准的Qt API, 我们可以非常熟练的在Windows和Unix编程环境里开发应用程序。

Qt/Embedded 是一组用于访问嵌入式设备的 Qt C++ API; Qt/Embedded 的Qt/X11, Qt/Windows 和Qt/Mac版本提供的都是相同的API和工具。Qt/Embedded还包括类库以及支持嵌入式开发的工具。

Qt/Embedded提供了一种类型安全的被称之为信号与插槽的真正的组件化编程机制, 这种机制和以前的回调函数有所不同。Qt/Embedded还提供了一个通用的widgets类, 这个类可以很容易的被子类化为客户自己的组件或是对话框。针对一些通用的任务, Qt还预先为客户定制了类似于消息框和向导这样的对话框。

运行Qt/Embedded 所需的系统资源可以很小, 相对X窗口下的嵌入解决方案而言, Qt/Embedded只要求一个较小的存储空间(Flash)和内存。Qt/Embedded可以运行在不同的处理器上部署的Linux系统, 只要这个系统有一个线性地址的缓冲帧并支持C++的编

译器。您可以选择不编译Qt/Embedded某些你不需要的功能，从而大大减小了它的内存占有量。

Qt/Embedded包括了它自身的窗口系统，并支持多种不同的输入设备。

开发者可以使用他们熟悉的开发环境来编写代码。Qt的图形设计器 (designer) 可以用来可视化地设计用户接口，设计器中有一个布局系统，它可以使您设计的窗口和组件自动根据屏幕空间的大小而改变布局。开发者可以选择一个预定义的视觉风格，或是建立自己独特的视觉风格。使用UNIX/LINUX操作系统的用户，可以在工作站上通过一个虚拟缓冲帧的应用程序仿真嵌入式系统的显示终端。

Qt/Embedded也提供了许多特定用途的非图形组件，例如国际化，网络和数据库交互组件。

Qt/Embedded是成熟可靠的工具开发包，它在世界各地被广泛使用。除了在商业上的许多应用以外，Qt/Embedded还是为小型设备提供的Qtopia应用环境的基础。Qt/Embedded以简洁的系统，可视化的表单设计和详致的API让编写代码变得愉快和舒畅。

1.2 系统要求

Qt/Embedded很省内存，因为它不需要一个X 服务器或是 Xlib库，相反它可以直接的写缓冲帧，它的内存消耗可以通过不编译某些不使用的功能来动态调节，它甚至可以
把全部的应用功能编译链接到一个简单的静态链接的可执行程序中，从而能够最大的节省内存。

Qt/Embedded可以运行在被Linux支持的所有的处理器上，当然所说的“Linux支持某个处理器”是指已经有一个Linux的c++编译器支持产生该处理器的目标代码以及Linux操作系统已经顺利移植到这个处理器上。目前Qt/Embedded可以运行在Intel x86, MIPS, ARM, StrongARM, Motorola 68000 and PowerPC等处理器上。

Qt/Embedded的应用程序可以直接的写内核缓冲帧,它支持的线性的缓冲帧包括1, 4, 8, 15, 16, 24和32位深度以及VGA16的缓冲帧,任何被内核支持的图形卡也可以工作,通过对Qt/Embedded进行客户化的定制可以使得它从图形加速系统获得好处。Qt/Embedded对显示屏幕的尺寸并无限制,另外它还有许多先进的功能,例如反别名字体、alpha-blended位图和屏幕旋转等。

Qt/Embedded的库可以通过在编译时去除不需要的功能来进行精简。例如,要想不编译QListView,可以通过定义一个QT_NO_LISTVIEW 的预处理标记来达到此目的;如果不想编译支持国际化的功能,那么可以定义QT_NO_I18N 的预处理标记。Qt/Embedded 提供了大约200个可配置的特征,由此在Intel x86平台上库的大小范围会在700KB到5000KB之间。大部分客户选择的配置使得库的大小在1500 KB 到 4000 KB 之间。

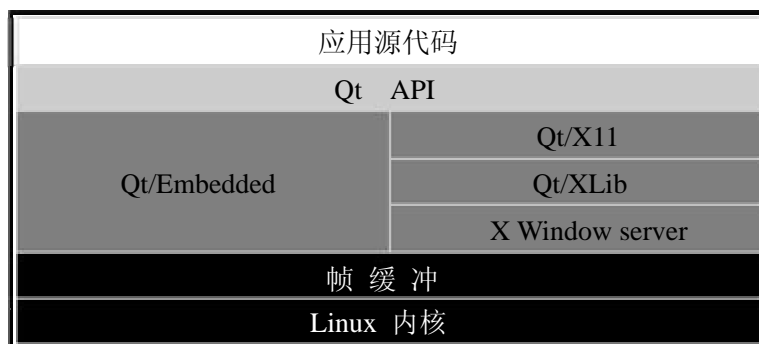
Qt/Embedded 还使用了一些节省内存空间的技术,例如隐式共享(写时复制)和缓存。在Qt中有超过20个类,包括 **QBitmap**, **QMap**, **QPalette**, **QPicture**, **QPixmap** 和 **QString**,使用了隐式共享技术,目的是避免不需要的复制和最小的内存需求。隐式共享过程会自动的发生,从而使编程更简单并且避免了处理指针和最优化时带来的危险。

许多的Qt组件都会被编译成为库的形式或者是插件的形式,客户视觉组件,数据库驱动,字体格式读写,图形格式转换,文本解码和窗体可以被编译成插件,从而减小核心库的大小并提供了更大的弹性。作为一种选择,如果对组件和应用了解得很深入的话,它们也可以被编译并和Qt/Embedded的库静态链接到一个简单的可执行程序,从而节省ROM, RAM和CPU的使用。

1.3 架构

Qt/Embedded 为带有轻量级窗口系统的嵌入式设备提供了一个标准的 Qt API。Qt/Embedded 的面向对象的设计使得它一直能不断的向前支持像键盘，鼠标和图形加速卡这样的额外设备。

通过使用Qt/Embedded，开发者可以感受到在Qt/X11，Qt/Windows和 Qt/Mac等不同的版本下使用相同的API编程带来的便利。



图一：Qt/Embedded与 Qt/X11 的Linux版本的比较

使用单一的API进行跨平台的编程可以有很多好处。提供嵌入式设备和桌面计算机环境下应用的公司可以培训开发人员使用同一套工具开发包，这有利于开发人员之间共享开发经验与知识，也使得管理人员在分配开发人员到项目中的时候增加灵活性。更进一步来说，针对某个平台而开发的应用和组件也可以销售到Qt支持的其它平台上，从而以低廉的成本扩大了产品的市场。

1.3.1 窗口系统

一个 Qt/Embedded窗口系统包含了一个或多个进程，其中的一个进程可作为服务器。该服务进程会分配客户显示区域，以及产生鼠标和键盘事件。该服务进程还能够提供输入方法和一个用户接口给运行起来的客户应用程序。该服务进程其实就是一个

有某些额外权限的客户进程。任何程序都可以在命令行上加上“-qws”的选项来把它作为一个服务器运行。

客户与服务器之间的通信使用共享内存的方法实现，通信量应该保持最小，例如客户进程直接访问帧缓冲来完成全部的绘制操作，而不会通过服务器，客户程序需要负责绘制它们自己的标题栏和其它式样。这就是Qt/Embedded库内部层次分明的处理过程。

客户可以使用QCOP通道交换消息。服务进程简单的广播QCOP消息给所有监听指定通道的应用进程，接着应用进程可以把一个插槽连接到一个负责接收的信号上，从而对消息做出响应。消息的传递通常伴随着二进制数据的传输，这是通过一个QDataStream类的序列化过程来实现的，有关这个类的描述，见28页的“非图形类”。

QProcess类提供了另外一种异步的进程间通信的机制。它用于启动一个外部的程序并且通过写一个标准的输入和读取外部程序的标准输出和错误码来和它们通信。

1.3.2 字体

Qt/Embedded支持四种不同的字体格式：True Type字体(TTF), Postscript Type1字体，位图发布字体(BDF)和Qt的预呈现(Pre-rendered)字体(QPF)。Qt还可以通过增加QFontFactory的子类来支持其它字体，也可以支持以插件方式出现的反别名字体。

每个TTF或者TYPE1类型的字体首次在图形或者文本方式的环境下被使用时，这些字体的字形都会以指定的大小被预先呈现出来，呈现的结果会被缓冲。根据给定的字体尺寸(例如10或12点阵)预先呈现TTF或者TYPE1类型的字体文件并把结果以QPF的格式保存，这样将可以节省内存和CPU处理时间。QPF文件包含了一些必要的字体，这些字体可以通过makeqpf工具取得，或者通过运行程序时加上“-savefonts”选项获

取。如果应用程序中使用到的字体都是QPF格式，那么Qt/Embedded将被重新配置，并排除对TTF和TYPE1类型的字体的编译，这样就可以减少Qt/Embedded的库的大小和存储字体的空间。例如一个10点阵大小的包含所有ASII字符的QPF字体文件的大小为1300字节，这个文件可以直接从物理存储格式映射成为内存存储格式。

Qt/Embedded的字体通常包括Unicode字体的一部分子集，ASII和Latin-1。一个完整的16点阵的Unicode字体的存储空间通常超过1M，我们应尽可能存储一个字体的子集，而不是存储所有的字，例如在您的应用中，您仅仅需要以Cappuccino字体、粗体的方式显示您的产品的名称，但是您却有一个包含了全部字形的字体文件。

1.3.3 输入设备

Qt/Embedded 3.0支持几种鼠标协议：BusMouse, IntelliMouse, Microsoft和MouseMan. Qt/Embedded 还支持 NEC Vr41XX 和 iPAQ 的触摸屏。通过从**QWMouseEventHandler**或者**QcalibratedMouseEventHandler**派生子类，开发人员可以让Qt/Embedded支持更多的客户指示设备。

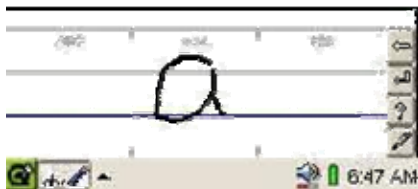
Qt/Embedded支持标准的101键盘和Vr41XX按键，通过子类化**QWSKeyboardHandler** 可以让Qt/Embedded支持更多的客户键盘和它的非指示设备。

1.3.4 输入方法

对于非拉丁语系字符（例如阿拉伯，中文，希伯来和日语）的输入法，需要把它写成过滤器的方式，并改变键盘的输入。输入法的作者应该对全部的Qt API的使用有完全的认识。

在一个无键盘的设备上，输入法成了唯一的输入字符的手段。QtPia提供了四种输入方法：笔迹识别器，图形化的标准键盘，Unicode键盘，居于字典方式提取的键盘。这些

键盘的样式如下所示



笔迹识别器



Unicode 输入



标准键盘



字典式提取键盘

1.3.5 屏幕加速

通过子类化 `QScreen` 和 `QgfxRaster` 可以实现硬件加速，从而为屏幕操作带来好处。Trolltech提供了Mach64 和 Voodoo3 视频卡的硬件加速的驱动例子，同时可以按照协议编写其它的驱动程序。

入门篇

1. Qt/Embedded 开发环境的安装

一般来说，居于 Qt/Embedded 开发的应用程序最终会发布到安装有嵌入式 Linux 操作系统的小型设备上，所以使用装有 Linux 操作系统的 PC 机或者工作站来完成 Qt/Embedded 开发当然是最理想的环境，尽管 Qt/Embedded 也可以安装在 Unix 和

Windows 系统上。

下面将介绍如何在一台装有 Linux 操作系统的机器上建立 Qt/Embedded 开发环境。

首先，您需要拥有三个软件安装包：tmake 工具安装包，Qt/Embedded 安装包，Qt 的 X11 版的安装包。由于上述这些软件安装包有许多不同的版本，您要注意由于版本的不同导致这些软件在使用时可能造成的冲突，为此将告诉您一些基本的安装原则：当您选择或下载了 Qt/Embedded 的某个版本的安装包之后，您下一步要选择安装的 Qt for X11 的安装包的版本必须比您最先下载的 Qt/Embedded 的版本要旧，这是因为 Qt for X11 的安装包的两个工具 uic 和 designer 产生的源文件会和 Qt/Embedded 的库一起被编译链接，本着“向前兼容”的原则，Qt for X11 的版本应比 Qt/Embedded 的版本旧。

将以下面所列版本的安装包，一步一步介绍 Qt/Embedded 开发环境建立的过程（这些软件可以免费从 trolltech 的 WEB 或 FTP 服务器上下载），

- ◆ tmake 1.11 或更高版本；（生成 Qt/Embedded 应用工程的 Makefile 文件）
- ◆ Qt/Embedded 2.3.7 （Qt/Embedded 安装包）
- ◆ Qt 2.3.2 for X11；（Qt 的 X11 版的安装包，它将产生 x11 开发环境所需要的两个工具）

1.1 安装 tmake

在 Linux 命令模式下运行以下命令：

```
tar xzf tmake-1.11.tar.gz
export TMAKEDIR=$PWD/tmake-1.11
export TMAKEPATH=$TMAKEDIR/lib/qws/linux-x86-g++
export PATH=$TMAKEDIR/bin:$PATH
```

1.2 安装 Qt/Embedded 2.3.7

在 Linux 命令模式下运行以下命令：

```
tar xzf qt-embedded-2.3.7.tar.gz
cd qt-2.3.7
export QTDIR=$PWD
export QTEDIR=$QTDIR
export PATH=$QTDIR/bin:$PATH
export LD_LIBRARY_PATH=$QTDIR/lib:$LD_LIBRARY_PATH
./configure -qconfig local -qvfb -depths 4,8,16,32
make sub-src
cd ..
```

上述命令 `./configure -qconfig -qvfb -depths 4,8,16,32` 指定 Qt 嵌入式开发包生成虚拟缓冲帧工具 `qvfb`，并支持 4, 8, 16, 32 位的显示颜色深度。另外我们也可以在 `configure` 的参数中添加 `-system-jpeg` 和 `gif`，使 Qt/Embedded 平台能支持 `jpeg`、`gif` 格式的图形。

上述命令 `make sub-src` 指定按精简方式编译开发包，也就是说有些 Qt 类未被编译。Qt 嵌入式开发包有 5 种编译范围的选项，使用这些选项，可控制 Qt 生成的库文件的大小，但是您的应用所使用到的一些 Qt 类将可能因此在 Qt 的库中找不到链接。编译选项的具体用法可运行 `./configure -help` 命令查看。

1.3 安装 Qt/X11 2.3.2

在 Linux 命令模式下运行以下命令：

```
tar xzf qt-x11-2.3.2.tar.gz
cd qt-2.3.2
export QTDIR=$PWD
```

```
export PATH=$QTDIR/bin:$PATH
export LD_LIBRARY_PATH=$QTDIR/lib:$LD_LIBRARY_PATH
./configure --no-opengl
make
make -C tools/qvfb
mv tools/qvfb/qvfb bin
cp bin/uic $QTDIR/bin
cd ..
```

根据开发者本身的开发环境，也可以在 `configure` 的参数中添加别的参数，比如 `--no-opengl` 或 `--no-xfs`，可以键入 `./configure --help` 来获得一些帮助信息。

2. 认识 Qt/Embedded 开发环境

Qt/Embedded 的开发环境可以取代那些我们熟知的 UNIX 和 WINDOWS 开发工具。它提供了几个跨平台的工具使得开发变得迅速和方便，尤其是它的图形设计器。Unix 下的开发者可以在 PC 机或者工作站使用虚拟缓冲帧，从而可以仿真一个和嵌入式设备的显示终端大小，像素相同的显示环境。

嵌入式设备的应用可以在安装了一个跨平台开发工具链的不同的平台上编译。最通常的做法是在一个 UNIX 系统上安装跨平台的带有 `libc` 库的 GNU `c++` 编译器和二进制工具。在开发的许多阶段，一个可替代的做法是使用 Qt 的桌面版本，例如 Qt/X11 或是 Qt/Windows 来进行开发。这样开发人员就可以使用他们熟悉的开发环境，例如微软的 Visual C++ 或者 Borland C++；在 UNIX 操作系统下，许多环境也是可用的，例如 Kdevelop，它也支持交互式开发。

如果 Qt/Embedded 的应用是在 UNIX 平台下开发的话，那么它就可以在开发的机器上以一个独立的控制台或者虚拟缓冲帧的方式来运行，对于后者来说，其实是有一个 X11 的应用程序虚拟了一个缓冲帧。通过指定显示设备的宽度，高度和颜色深度，

虚拟出来的缓冲帧将和物理的显示设备在每个像素上保持一致。这样每次调试应用时开发人员就不用总是刷新嵌入式设备的 FLASH 存储空间，从而加速了应用的编译、链接和运行周期。

运行 Qt 的虚拟缓冲帧工具的方法是：在 Linux 的图形模式下运行命令：

```
qvfb (回车)
```

当 Qt 嵌入式的应用程序要把显示结果输出到虚拟缓冲帧时，我们在命令行运行这个程序时，在程序名后加上 -qws 的选项。例如：`$> hello -qws`

2.1 QT 的支撑工具

Qt 包含了许多支持嵌入式系统开发的工具，其中一些工具我们会在别的地方介绍。有两个最实用的工具（除了上面我们提到的虚拟缓冲帧）是 qmake 和 Qt designer(图形设计器)。

qmake 是一个为编译 Qt/Embedded 库和应用而提供的 Makefile 生成器。它能够根据一个工程文件 (.pro) 产生不同平台下的 Makefile 文件。qmake 支持跨平台开发和影子生成 (shadow builds)，影子生成是指当工程的源代码共享给网络上的多台机器时，每台机器编译链接这个工程的代码将在不同的子路径下完成，这样就不会覆盖别人的编译链接生成的文件。qmake 还易于在不同的配置之间切换。

开发者可以使用 Qt 图形设计器可视化地设计对话框而不需编写一行代码。使用 Qt 图形设计器的布局管理可以生成具有平滑改变尺寸的对话框，qmake 和 Qt 图形设计器是完全集成在一起的。

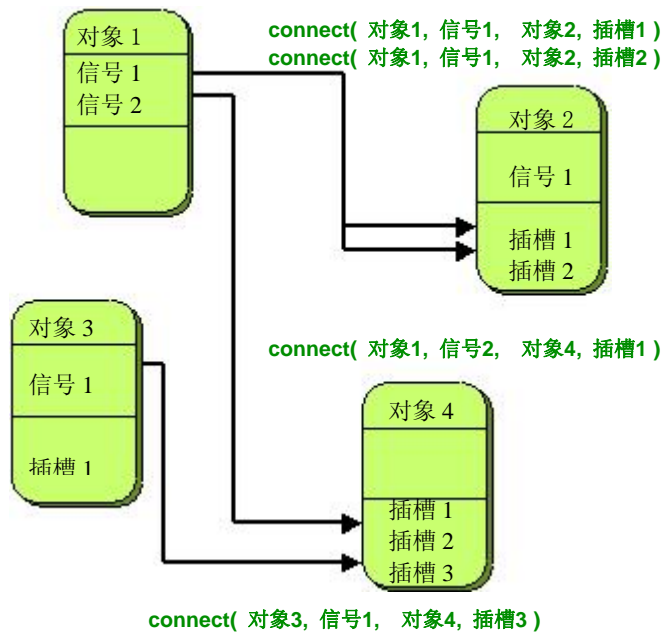
2.2 信号与插槽

信号与插槽机制提供了对象间的通信机制，它易于理解和使用，并完全被 Qt 图形设计器所支持。

图形用户接口的应用需要对用户的动作做出响应。例如，当用户点击了一个菜单项或是工具栏的按钮时，应用程序会执行某些代码。大部分情况下，是希望不同类型的对象之间能够进行通信。程序员必须把事件和相关代码联系起来，这样才能对事件做出响应。以前的工具开发包使用的事件响应机制是易崩溃的，不够健壮的，同时也不是面向对象的。 Trolltech 已经创立了一种新的机制，叫做“信号与插槽”。信号与插槽是一种强有力的对象间通信机制，它完全可以取代原始的回调和消息映射机制；信号与插槽是迅速的，类型安全的，健壮的，完全面向对象并用 C++来实现的一种机制。

在以前，当使用回调函数机制来把某段响应代码和一个按钮的动作相关联时，通常把那段响应代码写成一个函数，然后把这个函数的地址指针传给按钮，当那个按钮被按下时，这个函数就会被执行。对于这种方式，以前的开发包不能够确保回调函数被执行时所传递进来的函数参数就是正确的类型，因此容易造成进程崩溃，另外一个问题是，回调这种方式紧紧的绑定了图形用户接口的功能元素，因而很难把开发进行独立的分类。

Qt 的信号与插槽机制是不同的。Qt 的窗口在事件发生后激发信号。例如一个按钮被点击时会激发一个“clicked”信号。程序员通过建立一个函数（称作一个插槽），



图一 一些信号与插槽连接的抽象图

然后调用 `connect()` 函数把这个插槽和一个信号连接起来，这样就完成了一个事件和响应代码的连接。信号与插槽机制并不要求类之间互相知道细节，这样就可以相对容易的开发出代码可高重用的类。信号与插槽机制是类型安全的，它以警告的方式报告

类型错误，而不会使系统产生崩溃。

例如，如果一个退出按钮的 `clicked()` 信号被连接到了一个应用的退出函数 `quit()` 插槽。那么一个用户点击退出键将使应用程序终止运行。上述的连接过程用代码写出来就是这样

```
connect( button, SIGNAL(clicked()), qApp, SLOT(quit()) );
```

可以在 Qt 应用程序的执行过程中增加或是减少信号与插槽的连接。

信号与插槽的实现扩展了 C++ 的语法，同时也完全利用了 C++ 面向对象的特征。信号与插槽可以被重载或者重新实现，它们可以定义为类的公有，私有或是保护成员。

2.2.1 信号与插槽的例子

如果一个类要使用信号与插槽机制，它就必须是从 `QObject` 或者 `QObject` 的子类继承，而且在类的定义中必须加上 `Q_OBJECT` 宏。信号被定义在类的信号部分，而插槽则定义在 `public slots`, `protected slots` 或者 `private slots` 部分。

下面定义一个使用到信号与插槽机制的类。

```
class BankAccount : public QObject
{
    Q_OBJECT
public:
    BankAccount() { curBalance = 0; }
    int balance() const { return curBalance; }
public slots:
    void setBalance( int newBalance );
signals:
    void balanceChanged( int newBalance );
private:
```

```
    int curBalance;  
};
```

和大部分的 C++ 的类一样，**BankAccount** 类有一个构造函数，还有一个取值的函数 **balance()**，一个设置值的函数 **setBalance(int newBalance)**。

这个类有一个信号 **balanceChanged()**，这个信号声明了它在 **BankAccount** 类的成员 **curBalance** 的值被改变时产生。信号不需要被实现，当信号被激发时，和该信号连接的插槽将被执行。

上面用来设置值的函数 **setBalance(int newBalance)** 定义在类的 “public slots” 部分，因此它是一个插槽。插槽是一个需要实现的标准的成员函数，它可以像其它函数一样被调用，也可以和信号相连接。

下面就是该插槽函数 **setBalance(int newBalance)** 的实现代码：

```
void BankAccount::setBalance( int newBalance )  
{  
    if ( newBalance != curBalance )  
    {  
        curBalance = newBalance;  
        emit balanceChanged( curBalance );  
    }  
}
```

其中的一段代码

```
emit balanceChanged( curBalance );
```

它的作用是当 **curBalance** 的值被改变后，将新的 **curBalance** 的值作为参数去激活 **balanceChanged ()** 信号。对于关键词 “emit”，它和信号、插槽一样是由 Qt 提供的，这些关键词都会被 c++ 的预处理机制转换为 c++ 代码。

一个对象的信号可以被多个不同的插槽连接，而多个信号也可以被连接到相同的插槽。当信号和插槽被连接起来时，应当确保它们的参数类型是相同的，如果插槽的参数个数小于和它连接在一起的信号的参数个数，那么从信号传递插槽的多余的参数将被忽略。

2.2.2 元对象编译器

信号与插槽机制是以纯 C++代码来实现的，实现的过程使用到了 Qt 开发工具包提供的预处理器和元对象编译器（moc）。

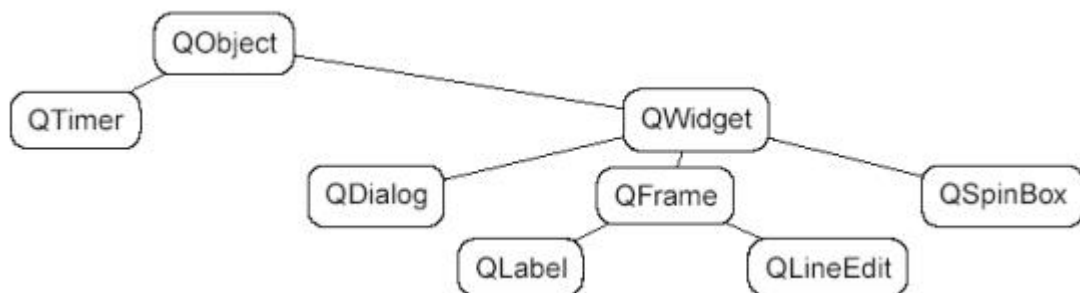
moc 读取应用程序的头文件，并产生支持信号与插槽的必要的代码。开发者没必要编辑或是浏览这些自动产生的代码，当有需要时，qmake 生成的 Makefile 文件里会显式的包含了运行 moc 的规则。

除了可以处理信号与插槽机制之外，moc 还支持翻译机制，属性系统和运行时的信息。

2.3 窗体

Qt 拥有丰富的满足不同需求的窗体（按钮，滚动条等等），Qt 的窗体使用起来很灵活，为了满足特别的要求，它很容易就可以被子类化。

窗体是 `QWidget` 类或它子类的实例，客户自己的窗体类需要从 `QWidget` 它的子类继承。



图二 摘录的 Qwidget 类的继承图

一个窗体可以包含任意数量的子窗体，子窗体可以显示在父窗体的客户区，一个没父窗体的窗体我们称之为顶级窗体（一个“窗口”），一个窗体通常有一个边框和标题栏作为装饰。Qt 并未对一个窗体有什么限制，任何类型的窗体可以是顶级窗体，任何类型的窗体可以是别的窗体的子窗体。在父窗体显示区域的子窗体的位置可以通过布局管理自动的进行设置，也可以人为的指定。当父窗体无效，隐藏或被删除后，它的子窗体都会进行同样的动作。

标签，消息框，工具栏等等，并未被限制使用什么颜色，字体和语言。Qt的文本呈现窗体可以使用HTML子集显示一个多语言的宽文本。

2.3.1 一个 Hello 的例子

下面是一个显示“Hello Qt/Embedded!” 的程序的完整的源代码：



图三 Hello Qt/Embedded

```
#include <qapplication.h>
```

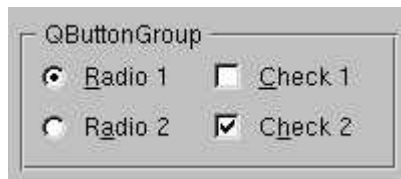
```
#include <qlabel.h>
int main( int argc, char **argv )
{
    QApplication app( argc, argv );
    QLabel *hello = new QLabel( "<font color=blue>Hello"
    " <i>Qt/Embedded!</i></font>", 0 );
    app.setMainWidget( hello );
    hello->show();
    return app.exec();
}
```

2.3.2 通用窗体

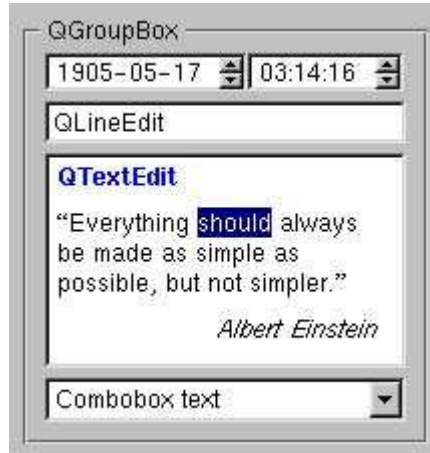
下面是一些主要的 Qt 窗体的截屏图，这些窗体使用了窗口样式。



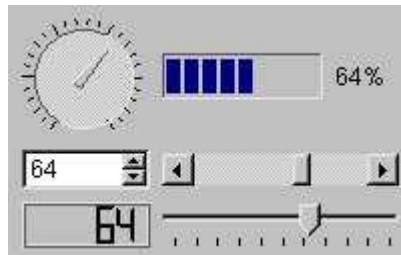
图四 使用了 QHBox 进行排列一个标签和一个按钮



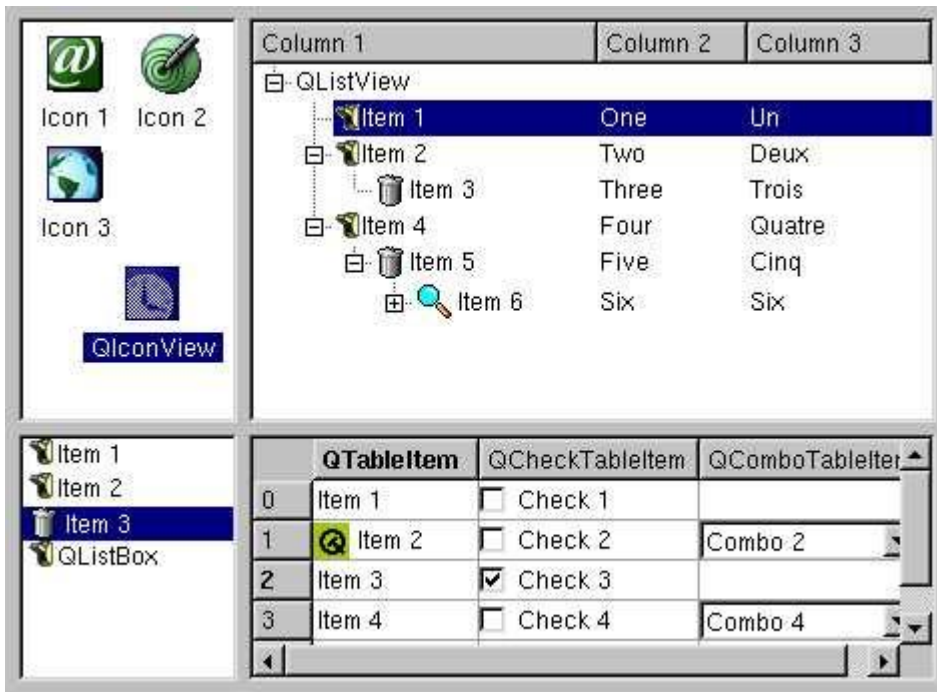
图五 使用了 QbuttonGroup 的两个单选框和两个复选框



图六 使用了QgroupBox进行排列的日期类QDateTimeEdit, 一个行编辑框类QLineEdit, 一个文本编辑类QTextEdit 和一个组合框类QComboBox



图七 以QGrid排列的一个 QDial, 一个QProgressBar, 一个QSpinBox, 一个QScrollBar, 一个QLCDNumber和一个Qslider



图八 以QGrid排列的一个QIconView, 一个 QListView, 一个 QListBox 和一个 QTable

有些时候在进行字符输入时, 希望输入的字符满足了某种规则才能使输入被确认。Qt提供了解决的办法, 例如QComboBox, QLineEdit 和 QspinBox 的字符输入可以通过 Qvalidator的子类来进行约束和有效性检查。

通过继承QScrollView, QTable, QListView, QTextEdit 和其它窗体就能够显示大量的数据, 并且自动的拥有了一个滚动条。

许多Qt创建的窗体能够显示图像, 例如按钮, 标签, 菜单项等等。Qimage类支持几种图形格式的输入、输出和操作, 它目前支持的图形格式有BMP, GIF*, JPEG, MNG, PNG, PNM, XBM 和 XPM。

2.3.3 画布

QCanvas 类提供了一个高级的平面图形编程接口，它可以处理大量的像线条、矩形、椭圆、文本、位图、动画等这些画布项，画布项可以较容易的做成交互式的（例如做成支持用户移动的）。

画布项是QcanvasItem子类的实例，它们比窗体类QWidget更显得轻量级，它们能够被快速的移动，隐藏和显示。Qcanvas可以更有效的支持冲突检测，它能够列出一个指定区域里面的所有的画布项。QcanvasItem可以被子类化，从而可以提供更多的客户画布项类型，或者扩展已有的画布项的功能。

Qcanvas对象是由QcanvasView进行绘制的，QcanvasView对象可以以不同的译文、比例、旋转角度，剪切方式去显示同一个画布。

Qcanvas 对象是理想的数据表现方式，它已经被消费者用于绘制地图和显示网络拓扑结构。它也可用于制作快节奏的且有大量角色的平面游戏。



图九 在 Qtopia 中用 QCanvas 实现的小行星游戏

2.3.4 客户窗体

通过对Qwidget或者它的子类进行子类化，可以建立自己的客户窗体或者对话框。下面是一个完整的源代码例子，它示例了如何通过子类化窗体，绘制一个模拟的时钟。AnalogClock 窗体类是Qwidget的子类，它显示当前时间，并且可以自动地更新时间。



图十 模拟钟窗体

在 analogclock.h 头文件中, **AnalogClock** 以这样地形式定义: :

```
#include <qwidget.h>
class AnalogClock : public QWidget
{
    public:
    AnalogClock( QWidget *parent = 0, const char *name = 0 );
    protected:
    virtual void timerEvent( QTimerEvent *event );
    virtual void paintEvent( QPaintEvent *event );
};
```

AnalogClock 类继承了QWidget, 它有一个典型的窗体类构造函数, 这个函数有父窗口对象指针和名字指针两个参数。(如果设置了名字的话, 测试和调试起来就会容易些)

timerEvent() 函数是从QObject (QWidget的父类) 对象继承而来的, 这个函数会被系统定期调用。paintEvent() 函数是从QWidget 继承而来的并且当窗体需要重画时这个函数就会被调用。

timerEvent() 和 paintEvent() 函数是“事件句柄”的两个例子。应用对象以重载父类对象的虚拟函数events (QEvent objects) 的形式接收系统的事件。大约有超过50个的系统事件是较常用的, 例如 MouseButtonPress, MouseButtonRelease, KeyPress, KeyRelease, Paint, Resize 和Close. 对象可以对发给它们的事件做出响应或者筛选一些事件后再发送给别的对象。

analogclock.cpp 文件是定义在analogclock.h中的函数的实现源文件

```
#include <qdatetimer.h>
#include <qpainter.h>
#include "analogclock.h"
AnalogClock::AnalogClock( QWidget *parent, const char *name )
```

```
: QWidget( parent, name )
{
    startTimer( 12000 );
    resize( 100, 100 );
}
void AnalogClock::timerEvent( QTimerEvent * )
{
    update();
}
void AnalogClock::paintEvent( QPaintEvent * )
{
    QCOORD hourHand[8] = { 2, 0, 0, 2, -2, 0, 0, -25 };
    QCOORD minuteHand[8] = { 1, 0, 0, 1, -1, 0, 0, -40 };
    QTime time = QTime::currentTime();
    QPainter painter( this );
    painter.setWindow( -50, -50, 100, 100 );
    painter.setBrush( black );
    for ( int i = 0; i < 12; i++ )
    {
        painter.drawLine( 44, 0, 46, 0 );
        painter.rotate( 30 );
    }
    painter.save();
    painter.rotate( 30 * (time.hour() % 12) + time.minute() / 2 );
    painter.drawConvexPolygon( QPointArray(4, hourHand) );
    painter.restore();
    painter.save();
    painter.rotate( 6 * time.minute() );
    painter.drawConvexPolygon( QPointArray(4, minuteHand) );
    painter.restore();
}
```

构造函数设置窗口的尺寸大小为100 x 100，并且告诉系统每隔12秒调用一次

timerEvent() 函数，从而对模拟钟的窗体进行刷新。

在 timerEvent() 函数中，通过调用QWidget的函数 update() 就可以告诉Qt，窗体需要立即重画，紧接着Qt就会产生一个绘制事件并且调用paintEvent() 函数。

在paintEvent() 函数中，一个QPainter对象用于在窗体上绘制12个刻度以及分针，时针。QPainter类提供了一种统一的方式用于绘制窗体，位图，矢量图等，它提供了绘制点，线，椭圆，多边形，弧，贝塞尔曲线等功能，一个QPainter的坐标系可以被转变，缩放，旋转，和剪切，这样对象就可以根据它在窗口或者窗体上的位置绘制出一个剪切的视图。剪切可以使窗体绘制时减少闪烁。使用QPainter 的子类QDirectPainter可以锁定和直接访问帧缓冲区域。

文件 analogclock.h 和 analogclock.cpp 完全的定义和实现了AnalogClock 客户窗体类，这个窗体是现在就可以使用的。

```
#include <qapplication.h>
#include "analogclock.h"
int main( int argc, char **argv )
{
    QApplication app( argc, argv );
    AnalogClock *clock = new AnalogClock;
    app.setMainWidget( clock );
    clock->show();
    return app.exec();
}
```

2.3.5 主窗口

QMainWindow类是为应用的主窗口提供一个摆放相关窗体的框架。

一个主窗口包含了一组标准窗体的集合。主窗口的顶部包含一个菜单栏，它的下方

放置着一个工具栏，工具栏可以移动到其它的停靠区域。主窗口允许停靠的位置有顶部，左边，右边和底部。工具栏可以被拖放到一个停靠的位置，从而形成一个浮动的工具面板。主窗口的下方，也就是在底部的停靠位置之下有一个状态栏。主窗口的中间区域可以包含其它的窗体。提示工具和“这是什么”帮助按钮以旁述的方式阐述了用户接口的使用方法。

对于小屏幕的设备，使用Qt图形设计器定义的标准的Qwidget模板比使用主窗口类更好一些。典型的模板包含有菜单栏，工具栏，可能没有状态栏（在必要的情况下，可以用任务栏，标题栏来显示状态）

2.3.6 菜单

弹出式菜单QpopupMenu类以垂直列表的方式显示菜单项，它可以是单个的（例如下文相关菜单），可以以菜单栏的方式出现，或者是别的弹出式菜单的子菜单出现。

每个菜单项可以有一个图标，一个复选框和一个加速器（快捷键），菜单项通常对应一个动作（例如存盘），分隔器通常显示成一条竖线，它用于把一组相关联的动作菜单分立成组。

下面是一个建立包含有 New, Open 和 Exit 菜单项的文件菜单的例子。

```
QPopupMenu *fileMenu = new QPopupMenu( this );
fileMenu->insertItem( "&New", this, SLOT(newFile()), CTRL+Key_N );
fileMenu->insertItem( "&Open...", this, SLOT(open()), CTRL+Key_O );
fileMenu->insertSeparator();
fileMenu->insertItem( "E&xit", qApp, SLOT(quit()), CTRL+Key_Q );
```

当一个菜单项被选中，和它相关的插槽将被执行。加速器(快捷键)很少在一个没有键盘输入的设备上使用，Qt/Embedded 的典型配置并未包含对加速器的支持。上面出现的代码“&New”意思是在桌面机器上以“**New**”的方式显示出来，但是在嵌入式设备上，它只

会显示为“New”。

QMenuBar类实现了一个菜单栏，它会自动的设置几何尺寸并在它的父窗体的顶部显示出来，如果父窗体的宽度不够宽以致不能显示一个完整的菜单栏，那么菜单栏将会分为多行显示出来。Qt内置的布局管理能够自动的调整菜单栏。

Qt的菜单系统是非常灵活的，菜单项可以被动态的使能，失效，添加或者删除。通过子类化QCustomMenuItem，我们可以建立客户化外观和功能的菜单项。

2.3.7 工具栏

QToolButton类实现了一个带有图标，3维边框和可选标签的工具栏按钮。切换工具栏按钮具有开、关的特征，其它的按钮则执行一个命令。不同的图标用来表示按钮的活动，无效、使能模式，或者是开或关的状态。如果你仅为按钮指定了一个图标，那么Qt会使用可视提示来表现按钮不同的状态，例如按钮失效时显示灰色。

工具栏按钮通常以一排的形式显示在工具栏上，对于一个有几组工具栏的应用，用户可以随便的到处移动这些工具栏，工具栏差不多可以包含所有的窗体，例如QComboBoxes 和 QSpinBoxes。

2.3.8 旁述

现代的应用主要使用旁述的方式去解释用户接口的用法。Qt 提供了两种旁述的方式：“提示栏”和“这是什么”帮助按钮。

“提示栏”是小的，通常是黄色的矩形，当鼠标在窗体的一些位置游动时它就会自动出现。它主要用于解释工具栏按钮，特别是那些缺少文字标签说明的工具栏按钮的用途。下面就是如何设置一个“存盘”按钮的提示的代码。

```
QToolTip::add( saveButton, "Save" );
```

当提示字符出现之后，你还可以在状态栏显示更详细的文字说明。

对于一些没有鼠标的设备（例如那些使用触点输入的设备），就不会有鼠标的光标在窗体上进行游动，这样就不能激活提示栏。对于这些设备也许就需要使用“这是什么”帮助按钮，或者使用一种姿态来表示输入设备正在进行游动，例如用按下或者握住的姿态来表示现在正在进行游动。

“这是什么”帮助按钮和提示栏有些相似，只不过前者是要用户点击它才会显示旁述。在小屏幕设备上，要想点击“这是什么”帮助按钮，具体的方法是，在靠近应用的 X 窗口的关闭按钮“x”附近你会看到一个“？”符号的小按钮，这个按钮就是“这是什么”帮助按钮。一般来说，“这是什么”帮助按钮按下后要显示的提示信息应该比提示栏要多一些。下面是设置一个存盘按钮的“这是什么”文本提示信息的方法：

```
QWhatsThis::add( saveButton, "Saves the current file." );
```

QToolTip 和 **QWhatsThis** 类提供了虚拟函数以供开发者重新实现更多的特定的用途。

Qtopia并未使用上述提及的两种帮助（旁述）机制。它在应用窗口的标题栏上放置一个“？”符号的按钮来代替上述的旁述机制，这个“？”按钮可以启动一个浏览器来显示和当前应用相关的HTML页面。Qtopia使用按下和握住的姿态来调用上下文菜单（右击）和属性对话框。

2.3.9 动作

应用程序通常提供给用户几种不同的方式去执行特别的动作。例如，大部分应用提供了一个“Save”动作给用于存盘的菜单(File|Save)以及工具栏（一个“软盘”图

标的工具栏按钮)和快捷键(Ctrl+S)。Qaction类可以让上述过程变得简洁,它允许程序员在一个地方定义一个动作,然后把这个动作加入到菜单或者工具栏,这个过程与把菜单项加入到菜单的道理是一样的。

下面的代码实现了一个“Save”菜单项和一个“Save”工具按钮,旁述系统和快捷键可以很容易的添加进去,但是我们没添加,因为它们很少在嵌入式设备上使用。

```
QAction *saveAct = new QAction( this );
saveAct->setText( "Save" );
saveAct->setIconSet( QPixmap("save.png") );
connect( saveAct, SIGNAL(activated()), this, SLOT(save()) );
saveAct->addTo( fileMenu );
saveAct->addTo( toolbar );
```

除了不用复制代码,使用 QAction 还可以确保菜单的状态与工具栏按钮的状态保持一致,必要的时候还可显示提示栏。使一个动作(Action)失效将导致和该动作相关联的菜单项以及工具按钮的失效。同样的,如果用户切换一个工具按钮的状态,那么相关的菜单项的也会跟着被选中或不选中。

2.4 对话框

使用 Qt 图形设计器这个可视化设计工具用户可以建立自己的对话框。Qt 使用布局管理自动的设置窗体与别的窗体之间相对的尺寸和位置,这样可以确保对话框能够最好的利用屏幕上的可用空间。使用布局管理意味着按钮和标签可以根据要显示的文字自动的改变自身大小,而用户完全不用考虑文字是那一种语言。

2.4.1 布局

Qt 的布局管理用于组织管理一个父窗体区域内的子窗体。它的特点是可以自动

的设置子窗体的位置和大小，并可判断出一个顶级窗体的最小和缺省的尺寸，当窗体的字体或内容变化后，它可以重置一个窗体的布局。

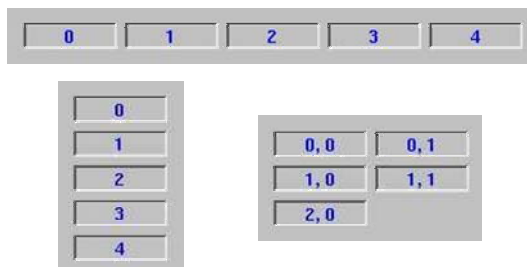
使用布局管理，开发者可以编写独立于屏幕大小和方向之外的程序，从而不需要浪费代码空间和重复编写代码。对于一些国际化的应用程序，使用布局管理，可以确保按钮和标签在不同的语言环境下有足够的空间显示文本，不会造成部分文字被剪掉。

布局管理使得提供部分用户接口组件，例如输入法和任务栏变得更容易。我们可以通过一个例子说明这一点，当Qtopia的用户正在输入文字时，输入法会占用一定的文字空间，应用程序这时也会根据可用的屏幕尺寸的变化调整自己。



图十 Qtopia 的布局管理

Qt提供了三种用于布局管理的类：`QHBoxLayout`、`QVBoxLayout` 和 `QGridLayout`。



图十一 QHBoxLayout, QVBoxLayout 和 QGridLayout 的布局效果

`QHBoxLayout` 布局管理把窗体按照水平方向从左至右排成一行

`QVBoxLayout` 布局管理把窗体按照垂直方向从上至下排成一列

`QGridLayout` 布局管理以网格的方式来排列窗体，一个窗体可以占据多个网格。

在多数情况下，布局管理在管理窗体时执行最优化的尺寸，这样窗口看起来就更好看而且可以尺寸变化会更平滑。使用以下的机制可以简化窗口布局的过程：

- 1、为一些子窗口设置一个最小的尺寸，一个最大的或者固定的尺寸。
- 2、增加拉伸项 (*stretch items*) 或者间隔项 (*spacer item*)。拉伸项和间隔项可以填充一个排列的空间。
- 3、改变子窗口的尺寸策略，程序员可以调整窗体尺寸改变时的一些策略。子窗体可以被设置为扩展，紧缩和保持相同尺寸等策略。
- 4、改变子窗口的尺寸提示。`QWidget::sizeHint()` 和 `QWidget::minimumSizeHint()` 函数返回一个窗体根据自身内容计算出的首选尺寸和首选最小尺寸，我们在建立窗体时可考虑重新实现这两个函数。
- 5、设置拉伸比例系数。设置拉伸比例系数是指允许开发者设置窗体之间占据空间大小的比例系数，例如我们设定可用空间的 $2/3$ 分配给窗体 A，剩下的 $1/3$ 则分配给窗体 B。

布局管理也可按照从右至左，从下到上的方式来进行。当一些国际化的应用需要

支持从右至左阅读习惯的语言文字（例如阿拉伯和希伯来）时，使用从右至左的布局排列是更方便的。

布局是可以嵌套的和随意进行的。下面是一个对话框的例子，它以两种不同尺寸大小来显示：



图十二 小的对话框和大的对话框

这个对话框使用了三种排列方式。QVBoxLayout管理一组按钮，QHBoxLayout管理一个显示国家名称的列表框和右边那组按钮，QVBoxLayout管理窗体上剩下的组件“Now please select a country”标签。在“< Prev”和“Help”按钮之间放置了一个拉伸项（stretch items），使得两者之间保持了一定比例的间隔。

建立这个对话框窗体和布局管理的实现代码如下：

```
QVBoxLayout *buttonBox = new QVBoxLayout( 6 );  
buttonBox->addWidget( new QPushButton("Next >", this) );  
buttonBox->addWidget( new QPushButton("< Prev", this) );
```

```
buttonBox->addStretch( 1 );  
  
buttonBox->addWidget( new QPushButton("Help", this) );  
  
QListBox *countryList = new QListBox( this );  
countryList->insertItem( "Canada" );  
  
/* ... */  
  
countryList->insertItem( "United States of America" );  
  
QHBoxLayout *middleBox = new QHBoxLayout( 11 );  
middleBox->addWidget( countryList );  
middleBox->addLayout( buttonBox );  
  
QVBoxLayout *topLevelBox = new QVBoxLayout( this, 6, 11 );  
topLevelBox->addWidget( new QLabel("Now please select a country", this) );  
topLevelBox->addLayout( middleBox );
```

使用 Qt 图形设计器设计的这个对话框，显示如下



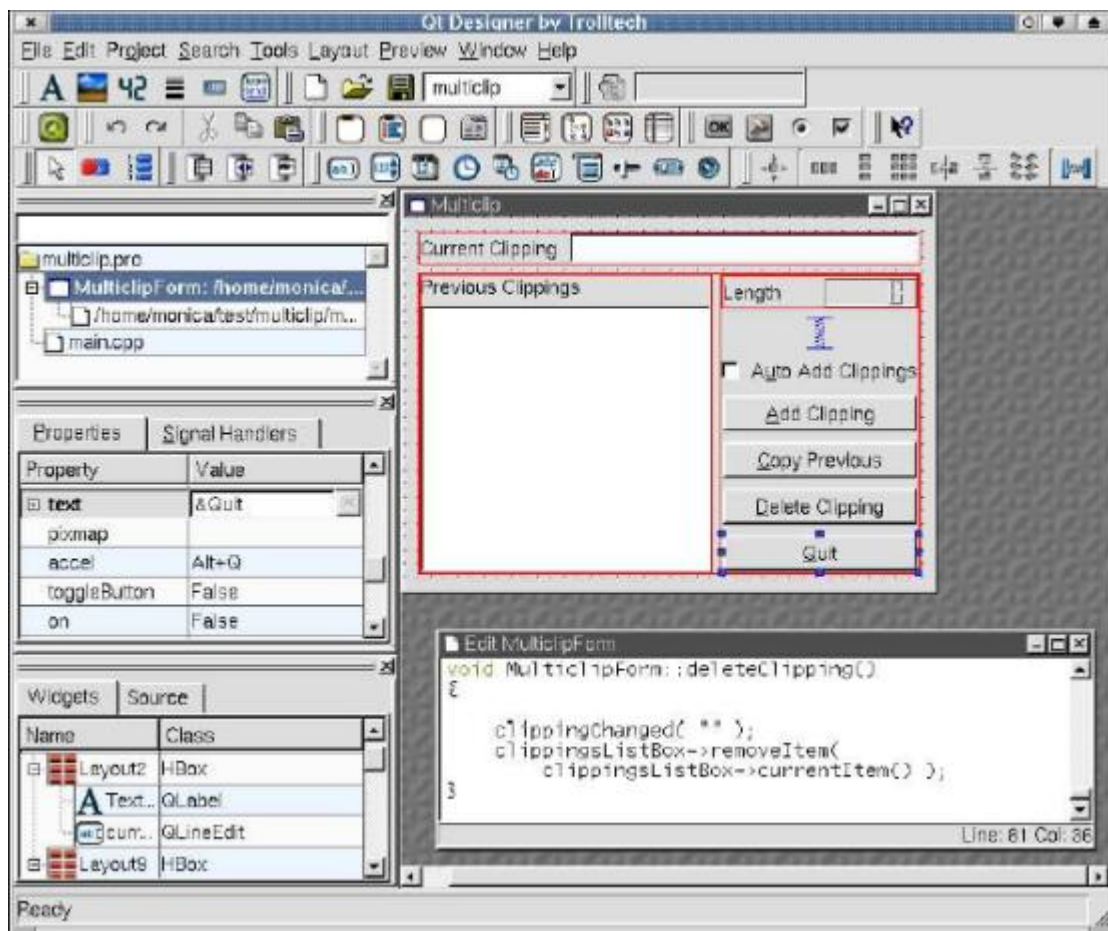
图十三 Qt 图形设计器中使用了布局的对话框

2.4.2 Qt 图形设计器

Qt 图形设计器是一个具有可视化用户接口的设计工具。Qt 的应用程序可以完全用源代码来编写，或者使用 Qt 图形设计器来加速开发工作。启动 Qt 图形设计器的方法是：在 Linux 命令模式下，键入以下命令（假设 Qt X11 安装在 /usr/local 下）：

```
cd qt-2.3.2/bin  
./designer
```

这样就可以启动一个与 Windows 下的 Delphi 相类似界面。下图是使用 Qt 图形设计器设计一个表单的截屏图。



图十四 Qt 图形设计器

开发者点击工具栏上的代表不同功能的子窗体/组件的按钮，然后把它放到一个表单上面，这样就可以把一个子窗体/组件放到表单上了。开发者可以使用属性对话框来设置子窗体的属性。精确的设置子窗体的位置和尺寸大小是没必要的。开发者可以选择一组窗体，然后对他们进行排列。例如，我们选定了一些按钮窗体，然后使用“水平排列（lay out horizontally）”选项对它们进行一个接一个的水平排列。这样

做使得设计工作变得更快，而且完成后的窗体将能够按照属性设置的比例填充窗口的可用尺寸范围。

使用 Qt 图形设计器进行图形用户接口的设计可以消除应用的编译，链接和运行时间，同时使得修改图形用户接口的设计变得更容易。Qt 图形设计器的预览功能可以使开发者能够在开发阶段看到各种样式的图形用户界面，包括客户样式的用户界面。通过 Qt 集成的功能强大的数据库类，Qt 图形设计器还可提供生动的数据库数据浏览和编辑操作。

开发者可以建立同时包含有对话框和主窗口的应用，其中主窗口可以放置菜单，工具栏，旁述帮助等等的子窗口部件。Qt 图形设计器提供了几种表单模板，如果窗体会被多个不同的应用反复使用，那么开发者也可建立自己的表单模板，以确保窗体的一致性。

Qt 图形设计器使用向导来帮助人们更快更方便的建立包含有工具栏、菜单和数据库等方面的应用。程序员可以建立自己的客户窗体，并把它集成到 Qt 图形设计器中。

Qt 图形设计器设计的图形界面以扩展名为“ui”的文件进行保存，这个文件有良好的可读性，这个文件可被 uic (Qt 提供的用户接口编译工具) 编译成为 C++ 的头文件和源文件。Qmake 工具在它为工程生成的 Makefile 文件中自动的包含了 uic 生成头文件和源文件的规则。

另一种可选的做法是，在应用程序运行期间载入 ui 文件，然后把它转变为具备原先全部功能的表单。这样开发者就可以在程序运行期间动态修改应用的界面，而不需重新编译应用，另一方面，也使得应用的文件尺寸减小了。

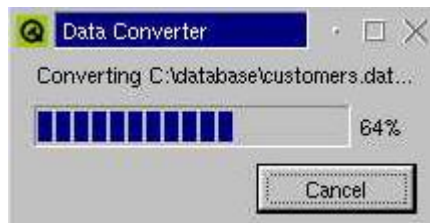
2.4.3 建立对话框

Qt 为许多通用的任务提供了现成的包含了实用的静态函数的对话框类，下边是一些 Qt 的标准的对话框的截屏图。

QMessageBox类是一个用于向用户提供信息或是给用户进行一些简单选择（例如“yes”或“no”）的对话框类。



图十五 一个 QMessageBox 对话框
progressDialog对话框包含了一个进度栏和一个“Cancel”按钮



图十六 一个 QprogressDialog 对话框

Qwizard类提供了一个向导对话框的框架



图十七 一个向导类

Qt提供的对话框还包括[QColorDialog](#), [QFileDialog](#), [QFontDialog](#) 和 [QprintDialog](#)。这些类通常适用于桌面应用，一般不会在Qt/Embedded中编译使用它们。

2.5 外形与感觉

Qt桌面应用随着执行环境（例如*Windows XP*, *Mac OS X*, *Linux*）的不同而具有不同的样式，或者叫做外形与感觉。Qt/Embedded 的应用可以使用不同操作环境提供的样式，也可以以静态或插件的方式使用客户样式。开发者可以设计自己窗体的样式和窗口装饰。

2.5.1 窗体样式

一个样式是一个实现Qt窗体外形与视觉效果的一类[Qstyle](#)的子类。Qt/Embedded 程序员可以自由的使用和修改目前存在的样式，也可以使用Qt样式设计引擎实现自己的样

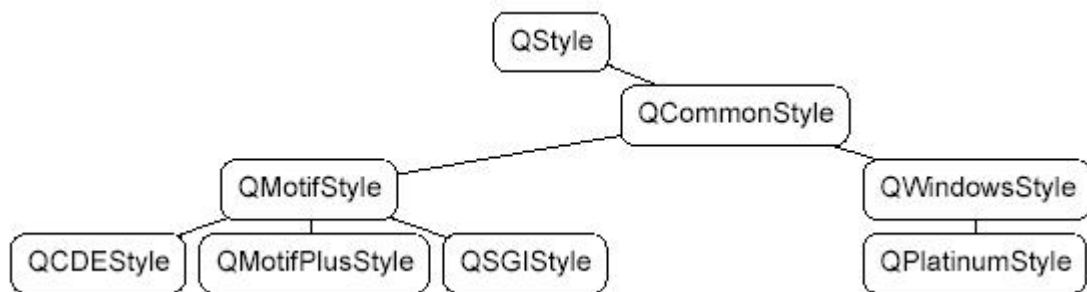
式。目前Qt/Embedded提供的样式类型有EmbeddedareWindows, Motif, MotifPlus, CDE, Platinum 和 SGI。样式可以被动态的设置到一个应用中,甚至可以设置到特定的窗体上。



图十八 以不同样式建立的下拉框

通过给一组相关的应用编写一个客户样式,可以让这些应用的外观具有与众不同的感觉。建立客户样式可以通过子类化QStyle, QCommonStyle或者QCommonStyle的派生类来实现。通过重新实现现有样式基类的一两个虚函数可以很容易对现有样式做一些小的修改。

一个样式可以编译成为一个插件,在Qt图形设计器中,把样式做成插件的话,开发者就可以以设备的客户样式来预览设计出来的表单。样式插件使得开发者不需重新编译就可改变设备的外观。



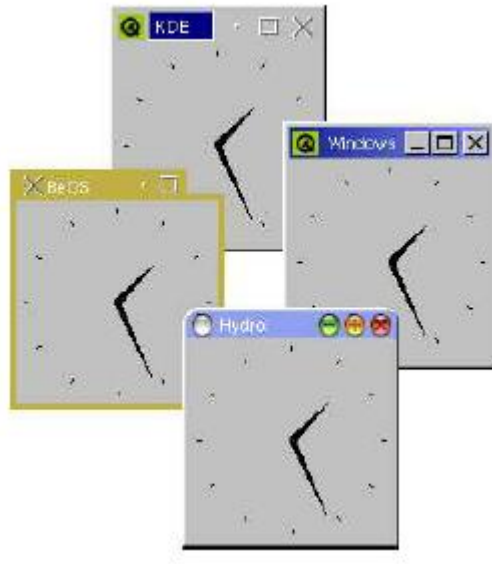
图十九 Qstyle 的继承图

Qt发布的窗体都是样式已知的,当它们的样式改变后,窗体会自动重绘。客户窗体和对话框大多数都是Qt原先发布的窗体和布局的重新整合。它们也是自动的获得原

先的样式。在少数场合，当有必要写一个外观上很随意的窗体时，开发者可以使用 `QStyle` 去写一些用户接口元素，而不是直接的绘制固有的矩形。

2.5.2 窗口装饰

顶级窗口的装饰是由一个标题栏和一个框架组成的。`Qt/Embedded` 包括了这些窗口样式：`BeOS`，`Hydro`，`KDE` 和 `Windows`。



图二十 不同的平台上的窗口风格

如果需要的话，我们可以配置不同的窗口使用不同的窗口装饰。通过子类化 `QWSDecoration`，我们可以建立客户的装饰样式，并且把它们以插件的形式进行发布。为了在窗口管理器之外进行更多的控制行为，开发者需要子类化 `QWSManager`。

2.6 国际化

Qt/Embedded 完全支持 Unicode，一个国际标准的字符集。开发者在他们的应用中可以自由的混合使用被 Unicode 字符集支持的语言，例如阿拉伯文，英文，中文，希伯来文，日文和俄文等。为了有助于公司将产品推向国际市场，Qt 还提供了将应用翻译成支持多种语言环境的工具。

2.6.1 Unicode

Qt 使用 **QString** 存储 Unicode 编码的字符串，**QString** 取代了粗糙的 `const char *`；它提供了用于处理 **QString** 和 `const char *` 之间相互转换的构造函数和操作符。因为 Qt 使用了隐式共享(写时复制)技术来减少内存的使用, 所以直接复制 **QString** 的值是不会产生问题的。为有效率的存储 ASCII 码字符串，Qt 还提供了 **QString** 类。

Qt 为所有要显示在屏幕上的文本，包括最简单的文字标签到最复杂的宽文本编辑器，提供了一个强大的 Unicode 文本呈现引擎。这个引擎支持一些先进的特征，例如特殊的间隔线、双向写和区别标记。它几乎支持世界上所有的书写系统，包括阿拉伯文，中文，古斯拉夫文，英文，希腊文，希伯来文，日文，韩文，拉丁和越南文。体现这个引擎的最优化性能的常用的例子就是：在带有加速功能的文字的下方显示一条下划线（例如 **File**）。

QtextCodec 的子类用于处理不同编码类型的字符集之间的转换。Qt 3.0 支持 37 种不同的编码方式，包括中文的 Big5 和 GBK, 日文的 EUC-JP、JIS 和 Shift-JIS, 俄罗斯的 KOI8-R 和 ISO 8859 系列。它们可以以库的一部分或者插件的形式编译，或者使用“feature”机制去除这些编译。

2.6.2 应用的翻译

Qt 提供了相应的工具和函数用于帮助开发者以他们的本地语言推出应用。

要使一个字符串可以被翻译，您需要把这个字符串作为一个参数放到 `tr()` 函数中调用。例如：

```
saveButton->setText( tr("Save") );
```

Qt 尝试寻找字符串“Save”的译文，如果找到的话，就会把它的译文显示出来，如果找不到，就用原来的字符串“Save”进行显示。例如把英文作为源语言，就需要把这个源语言翻译成中文，即中文为翻译的目标语言，以此类推。这样当调用 `tr()` 函数后，函数的参数的原先的缺省编码就会转变为 Unicode 编码。`Tr()` 函数的通常用法为：

```
Context::tr("source text", "comment")
```

上面的“Context”是指一个 `QObject` 对象的子类的名称。如果在一个包含了 `tr()` 成员函数的类的上下文环境中使用 `tr()` 函数时，“Context”通常可以省略掉。例如：

```
Context: : func1 ()  
{  
    setText( tr("Save") );  
}
```

“source text”是指要翻译的文本的内容

“comment”是一个可选项，它用于给手工翻译者提供一些额外的信息。

说了半天，还有一个重要的内容，就是 `tr()` 函数如何寻找到译文。要把一个源字符串翻译为和它对应的译文（目标语言的字符串）时，需要让 Qt 知道这些译文放在哪里。Qt 规定了译文存储在 `QTranslator` 对象中，这个对象是从一个内存映射文件（扩展名为 `qm`）中读取译文。每个 `qm` 文件包含了某种语言的译文信息。所以开发者需要建立一个 `qm` 文件来存储应用中需要翻译的字符串的译文。

Qt 提供了 3 种工具帮助人们建立译文存储文件（`.qm` 文件），这 3 个工具是 `lupdate`,

Qt Linguist 和 `lrelease`.

1) `lupdate` 自动地从源代码文件 (.cpp) 和界面接口文件 (.ui) 中获取所有需要翻译的对象, 即上述的 (“Context”); 同时还获取要翻译的所有的字符串 (源语言的字符串), 即上述的 “source text”; “comment” 选项如果被使用的话, 也会被纳入收集的范围。当这些信息收集完毕后, `lupdate` 最后会生成一个 .ts 文件 (翻译源文件), 这个文件是直接可阅读的。

2) 开发者使用 *Qt Linguist* 工具提供的良好的人机界面打开一个 .ts 文件 (翻译源文件), 然后开发者根据每一个 source text 填写上相对应的译文, 这样一个 .ts 文件 (翻译源文件) 就包含了完整的 “Context”, “source text” 和译文信息。

3) 最后通过运行 `lrelease` 去把一个 .ts 文件 (翻译源文件) 压缩为一个 .qm 文件 (译文存储文件), 生成的 .qm 文件可用在嵌入式设备上。

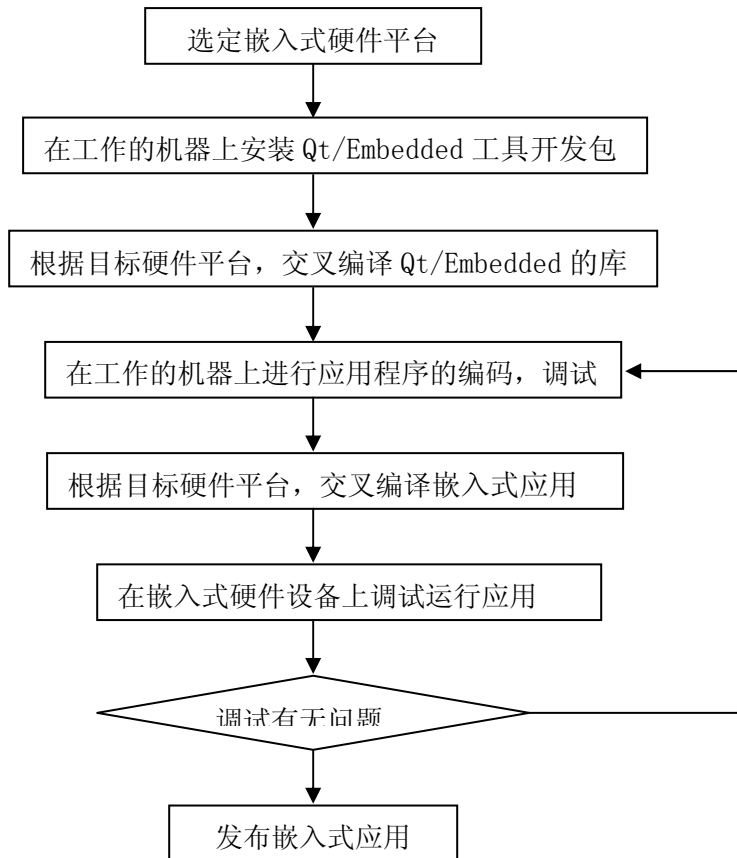
在一个应用的生存期间里, 上述的步骤有可能根据需要会被反复执行。多次运行 `lupdate` 是很安全的, 您可以重复使用已经存在的译文 (.qm) 文件, 或者当您不想使用某些翻译文件时您可以标记这些翻译源文件为旧文件, 而不需要删除它们。

实战篇

前面已经详细介绍了 Qt 嵌入式工具开发包的安装和使用方法, 但是这些介绍对于要真正进行一次商业的嵌入式应用开发来说并不足够。嵌入式应用的开发工作基本上是在工作站或是 PC 机上完成的, 在工作的机器上调试运行嵌入式应

用, 并将输出结果显示在一个仿真小型设备显示终端的模拟器上。在开发的后期, 要根据我们选择的嵌入式硬件平台, 将嵌入式应用编译链接成适合在这个硬件平台上运行的二进制目标代码, 另外由于应用使用到了 Qt/Embedded 的库, 所以还要将 Qt/Embedded 库的源代码编译链接成为适合在这个硬件平台上使用的二进制目标代码库。当一个 Qt/Embedded 应用被部署到小型设备上, 并可靠的运行, 这样一个开发过程才宣告结束。

使用 Qt/Embedded 开发一个嵌入式应用的过程大体可用下面的流程图表示:



图一 Qt/Embedded 应用开发的一般流程

在以下的篇幅，将按照上述的流程完整的介绍使用 Qt/Embedded 开发一个嵌入式应用例子的过程。

1. 嵌入式硬件开发平台的选择

嵌入式系统的核心部件是各种类型的嵌入式处理器，目前据不完全统计，全世界嵌入式处理器的品种总量已经超过 1000 多种，流行体系结构有 30 几个系列，如何在种类纷繁的嵌入式处理器中选择适合应用需求的处理器呢？在应用的需求分析过程中，有几项因素决定了应该选择什么样的嵌入式处理器：①嵌入式处理器能否在技术上实现应用②嵌入式处理器的成本是否符合应用要求。对于嵌入式处理器能否在技术上实现应用需要考虑到应用在运行时特点，比如应用对实时性的要求，对计算量和计算速度的要求，对外围接口电路的要求，对图形用户接口的要求等。当选定了一个嵌入式处理器之后，还要考虑选用什么样的操作系统，在选择操作系统时也要注意嵌入式处理器能否被所选的操作系统支持，有没有合适的编译器能够生成支持这种操作系统和嵌入式处理器的二进制目标代码。

近年来，随着 EDI 的推广和 VLSI 设计的普及化，及半导体工艺的迅速发展，在一个硅片上实现一个更为复杂的系统的时代已来临，这就是 System On Chip(SOC)。各种通用处理器内核将作为 SOC 设计公司的标准库，和许多其它嵌入式系统外设一样，成为 VLSI

设计中一种标准的器件，用标准的 VHDL 等语言描述，存储在器件库中。用户只需定义出其整个应用系统，仿真通过后就可以将设计图交给半导体工厂制作样品。这样除个别无法集成的器件以外，整个嵌入式系统大部分均可集成到一块或几块芯片中去，应用系统电路板将变得很简洁，对于减小体积和功耗、提高可靠性非常有利。

SOC 可以分为通用和专用两类。通用系列包括 Infineon 的 TriCore, Motorola 的 M-Core, 某些 ARM 系列器件, Echelon 和 Motorola 联合研制的 Neuron 芯片等。专用 SOC 一般专用于某个或某类系统中，不为一般用户所知。一个有代表性的产品是 Philips 的 Smart XA, 它将 XA 单片机内核和支持超过 2048 位复杂 RSA 算法的 CCU 单元制作在一块硅片上，形成一个可加载 JAVA 或 C 语言的专用的 SOC, 可用于公众互联网如 Internet 安全方面。

由于 SOC 芯片集成了处理器和许多常用的外围接口芯片，使得它的功能变得很强大，能够应用的领域和场合变得很广，所以嵌入式开发板的厂商选择 SOC 芯片作为开发板的核心芯片。三星的 S3C9200 就是一款带有 ARM920T 处理器内核的 SOC 芯片，由于它主频高，内置 LCD 和触摸屏控制器，以及声音控制器等外围电路，因而用在对图形用户接口有较高要求的场合是非常合适的。因为支持 ARM9 处理器的 linux 编译器早已发布，所以 S3C9200 可以很好的支持 linux 和 Qt/Embedded 的运行。

深圳优龙科技有限公司设计开发的 YL9200 嵌入式开发板是一个成熟稳定的嵌入式开发硬件平台，它使用了三星的 S3C9200 作为核心芯片。以下是 YL9200 的一些资料（更详细的资料可参考优龙公司的《YL9200 使用手册》）。在本文提及的 Qt/Embedded 应用例子都是居于 YL9200 嵌入式硬件开发平台的。

2. 安装 Qt/Embedded 工具开发包

要进行 Qt/Embedded 开发，就需要在工作的机器上安装 Qt/Embedded 工具开发包，这一步已在《Qt 嵌入式开发（入门篇）》的第一节中谈及，在此不在赘述。

3. 交叉编译 Qt/Embedded 的库

开发居于 Qt/Embedded 的应用程序要使用到 Qt/Embedded 的库，编写的 Qt 嵌入式应用程序最终是在 YL9200 开发板上运行的，因此在把 Qt 嵌入式应用程序编译成支持 YL9200 的目标代码之前，需要两样东西，一个是 arm9 的 linux 编译器，另一个是经 arm9 的 linux 编译器编译过的 Qt/Embedded 的库。

①安装交叉编译工具

需要arm9的linux编译器去编译工程并产生arm9处理器的目标代码，却是在一台PC机或者工作站上（这个机器有可能是x86的处理器）使用这个编译器，这个过程被称为交叉编译。交叉编译的实际定义是在一个处理器平台上编译产生一个工程代码的另一个处理器的目标代码。关于如何安装一个arm9的linux编译器请参考《YL9200使用手册》第七章“如何编译linux”，但是需要特别提醒读者注意，使用toolchain做为交叉编译工具时，最好使用cross-3.3.2及其以后的版本，这样才能对qt/embedded有良好支持。

②交叉编译Qt/Embedded库

当有了arm9的linux编译器之后，就可以使用这个编译器交叉编译Qt/Embedded库的

源代码，从而产生一个以ARM9为目标代码的Qt/Embedded库。具体过程如下

1、解包 Qt/Embedded（以 Qt/Embedded2.3.7 为例）

解包这个 Qt/Embedded2.3.7 压缩包时，应把它解压缩到不同于您的机器的处理器使用的 Qt/Embedded2.3.7 的安装路径。

在 Linux 命令模式下运行以下命令：

```
tar xzf qt-embedded-2.3.7.tar.gz
```

2、配置 Qt/Embedded2.3.7 的安装

Qt/Embedded 的安装选项有很多个，您可以在命令行下直接输入“./configure”来运行配置，这时安装程序会一步一步提示你输入安装选项。您也可以在“./configure”后输入多个安装选项直接完成安装的配置。在这些选项中有一个选项决定了编译 Qt/Embedded 库的范围，即可以指定以最小，小，中，大，完全 5 种方式编译 Qt/Embedded 库。另外 Qt/Embedded 的安装选项还允许我们自己定制一个配置文件，来有选择的编译 Qt/Embedded 库，这个安装选项是“-qconfig local”；当我们指定这个选项时，Qt/Embedded 库在安装过程中会寻找 qt-2.3.7/src/tools/qconfig-local.h 这个文件，如找到这个文件，就会以该文件里面定义的宏，来编译链接 Qt/Embedded 库。

由于使用优龙公司的 YL9200 嵌入式开发板，所以为了支持触摸屏的显示，需要定义一些宏，为了避免初学者在一开始就直接接触到 Qt/Embedded 的复杂的宏编译选项，把这些宏定义到一个名为 qconfig-local.h 的安装配置文件中，当使用者在安装 Qt/Embedded 的时候，需要把这个文件复制到 Qt/Embedded 的安装路径的/src/tools 子路径下。例如

您安装的是 Qt/Embedded2.3.7，那么您应该把提供给您的 qconfig-local.h 安装配置文件复制到 qt-2.3.7/src/tools 路径下。

具体过程如下：

```
cd qt-2.3.7
export QTDIR=$PWD
export QTEDIR=$QTDIR
cp /配置文件所在路径/qconfig-local.h ./src/tools
make clean
./configure -xplatform linux-arm-g++ -shared -debug (接下行)
-qconfig local -qvfb -depths 4,8,16,32
make
cd ..
```

4. 一个 Hello, World 的例子

下面将通过编写一个跳动的“Hello, World”字符串，来讲解 Qt 嵌入式应用的开发过程。

① 生成一个工程文件（.pro 文件）

一个应用通常对应一个工程文件，生成一个工程文件，并对它做一些简单的编辑，然后

使用一个专门的工具（例如 tmake）处理这个工程文件，就可以生成一个 Makefile 文件。

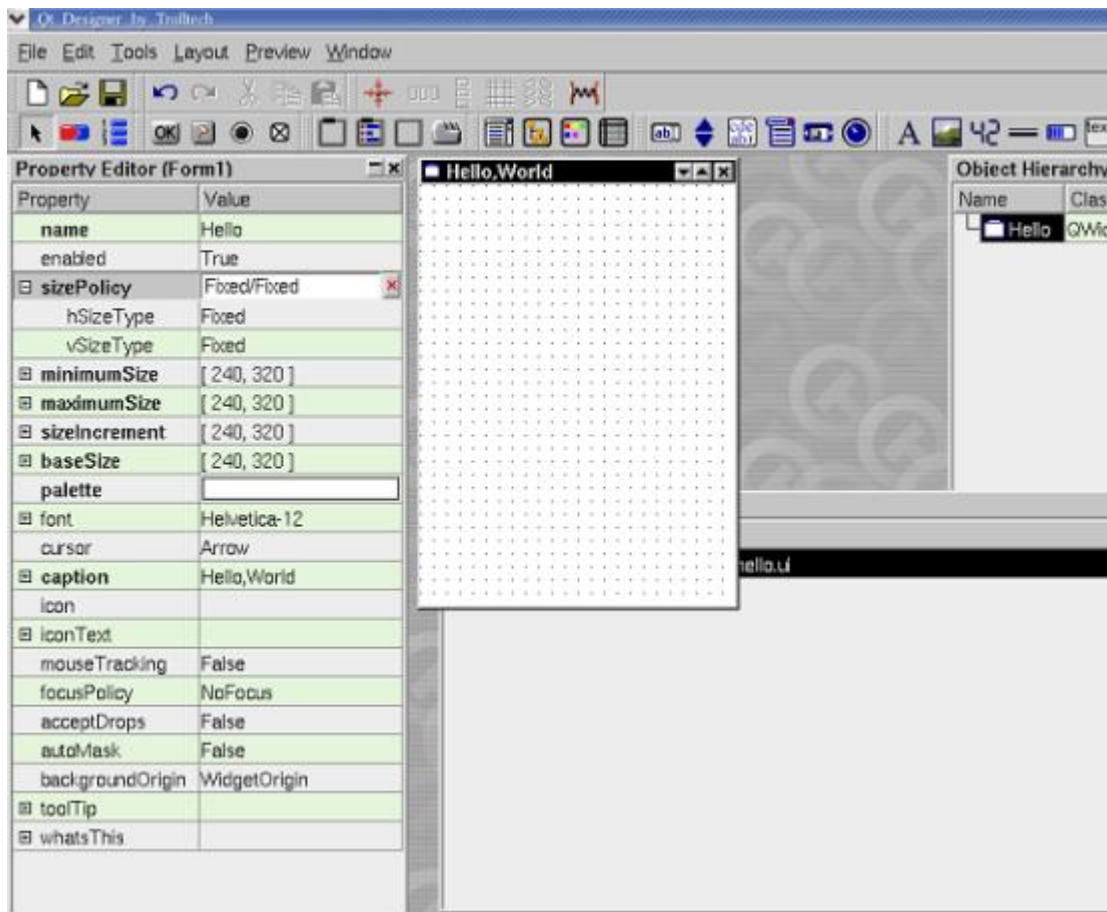
产生一个工程文件的其中一个方法是使用 progen 命令（progen 程序可在 tmake 的安装路径下找到）。下面使用 progen 产生一个名为 hello 的工程文件

```
progen -t app.t -o hello.pro
```

产生的 hello.pro 工程文件并不完整，开发者还需手动添加工程所包含的头文件，源文件等信息。

② 新建一个窗体

在 qt-2.3.x for x11 的安装路径的 bin 目录下运行 “./designer” 命令，就启动了一个 Qt 图形编辑器。点击编辑器的 “new” 菜单，弹出了一个 “new Form” 对话框，在这个对话框里选择 “Widget”，然后点击 “OK” 按钮，这样就新建了一个窗体。接着对这个窗体的属性进行设置，注意把窗体的 “name” 属性设为 “Hello”；窗体的各种尺寸设为宽 “240”，高 “320”，目的是使窗体大小和 YL9200 带的显示屏大小一致；窗体背景颜色设置为白色。具体设置可参见下图：



图二 Hello 窗体的属性设置

设置完成后，将其保存为 `hello.ui` 文件，这个文件就是 Hello 窗体的界面存储文件。

③ 生成 Hello 窗体类的头文件和实现文件

下面根据上述的界面文件 `hello.ui` 使用 `uic` 工具产生出 Hello 窗体类的头文件和实现文件，具体方法是：

```
cd qt-2.3.7/bin
uic -o hello.h hello.ui
uic -o hello.cpp -impl hello.h hello.ui
```

这样就得到了 Hello 窗体类的头文件 hello.h 和实现文件 hello.cpp。下面就可以根据我们要实现的具体功能，在 hello.cpp 文件里添加相应的代码。

比如要在 Hello 的窗体上显示一个动态的字符串“Hello, World”，那么需要重新实现 paintEvent(QPaintEvent *)方法，同时还需要添加一个定时器 QTimer 实例，以周期性刷新屏幕，从而得到动画得效果。下面是修改后的 hello.h 和 hello.cpp 文件。

```
/*
** 以下是 hello.h 的代码
**
/
#ifndef HELLO_H
#define HELLO_H

#include <qvariant.h>
#include <qwidget.h>
class QVBoxLayout;
class QHBoxLayout;
class QGridLayout;

class Hello : public QWidget
{
    Q_OBJECT

public:
    Hello( QWidget* parent = 0, const char* name = 0, WFlags fl = 0 );
```



```
    ~Hello();

//以下是手动添加的代码
signals:
    void clicked();
protected:
    void mousePressEvent( QMouseEvent * );
    void paintEvent( QPaintEvent * );
private slots:
    void animate();
private:
    QString t;
    int     b;
};
#endif // HELLO_H
/*****
*
**  以下是 hello.cpp 源代码
*****/

#include "hello.h"

#include <qlayout.h>
#include <qvariant.h>
#include <qtooltip.h>
#include <qwhatsthis.h>
#include <qpushbutton.h>
#include <qtimer.h>
#include <qpainter.h>
#include <qpixmap.h>

/*
```

```
* Constructs a Hello which is a child of 'parent', with the
* name 'name' and widget flags set to 'f'
*/
Hello::Hello( QWidget* parent, const char* name, WFlags fl )
    : QWidget( parent, name, fl )
{
    if ( !name )
        setName( "Hello" );
    resize( 240, 320 );
    setMinimumSize( QSize( 240, 320 ) );
    setMaximumSize( QSize( 240, 320 ) );
    setSizeIncrement( QSize( 240, 320 ) );
    setBaseSize( QSize( 240, 320 ) );
    QPalette pal;
    QColorGroup cg;
    cg.setColor( QColorGroup::Foreground, black );
    cg.setColor( QColorGroup::Button, QColor( 192, 192, 192 ) );
    cg.setColor( QColorGroup::Light, white );
    cg.setColor( QColorGroup::Midlight, QColor( 223, 223, 223 ) );
    cg.setColor( QColorGroup::Dark, QColor( 96, 96, 96 ) );
    cg.setColor( QColorGroup::Mid, QColor( 128, 128, 128 ) );
    cg.setColor( QColorGroup::Text, black );
    cg.setColor( QColorGroup::BrightText, white );
    cg.setColor( QColorGroup::ButtonText, black );
    cg.setColor( QColorGroup::Base, white );
    cg.setColor( QColorGroup::Background, white );
    cg.setColor( QColorGroup::Shadow, black );
    cg.setColor( QColorGroup::Highlight, black );
    cg.setColor( QColorGroup::HighlightedText, white );
    pal.setActive( cg );
    cg.setColor( QColorGroup::Foreground, black );
    cg.setColor( QColorGroup::Button, QColor( 192, 192, 192 ) );
    cg.setColor( QColorGroup::Light, white );
```

```
cg.setColor( QColorGroup::Midlight, QColor( 220, 220, 220) );
cg.setColor( QColorGroup::Dark, QColor( 96, 96, 96) );
cg.setColor( QColorGroup::Mid, QColor( 128, 128, 128) );
cg.setColor( QColorGroup::Text, black );
cg.setColor( QColorGroup::BrightText, white );
cg.setColor( QColorGroup::ButtonText, black );
cg.setColor( QColorGroup::Base, white );
cg.setColor( QColorGroup::Background, white );
cg.setColor( QColorGroup::Shadow, black );
cg.setColor( QColorGroup::Highlight, black );
cg.setColor( QColorGroup::HighlightedText, white );
pal.setInactive( cg );
cg.setColor( QColorGroup::Foreground, QColor( 128, 128, 128) );
cg.setColor( QColorGroup::Button, QColor( 192, 192, 192) );
cg.setColor( QColorGroup::Light, white );
cg.setColor( QColorGroup::Midlight, QColor( 220, 220, 220) );
cg.setColor( QColorGroup::Dark, QColor( 96, 96, 96) );
cg.setColor( QColorGroup::Mid, QColor( 128, 128, 128) );
cg.setColor( QColorGroup::Text, black );
cg.setColor( QColorGroup::BrightText, white );
cg.setColor( QColorGroup::ButtonText, QColor( 128, 128, 128) );
cg.setColor( QColorGroup::Base, white );
cg.setColor( QColorGroup::Background, white );
cg.setColor( QColorGroup::Shadow, black );
cg.setColor( QColorGroup::Highlight, black );
cg.setColor( QColorGroup::HighlightedText, white );
pal.setDisabled( cg );
setPalette( pal );
QFont f( font() );
f.setFamily( "adobe-helvetica" );
f.setPointSize( 29 );
f.setBold( TRUE );
setFont( f );
```

```
    setCaption( tr( "" ) );

//以下是手动添加的代码
    t = "Hello, World";
    b = 0;
    QTimer *timer = new QTimer(this);
    connect( timer, SIGNAL(timeout()), SLOT(animate()) );
    timer->start( 40 );
}
/*
 * Destroys the object and frees any allocated resources
 */
Hello::~Hello()
{
}
/*
    This private slot is called each time the timer fires.
 */

//以下至结尾是手动添加的代码
void Hello::animate()
{
    b = (b + 1) & 15;
    repaint( FALSE );
}

/*
    Handles mouse button release events for the Hello widget.

    We emit the clicked() signal when the mouse is released inside
    the widget.
 */
```

```
void Hello::mouseReleaseEvent( QMouseEvent *e )
{
    if ( rect().contains( e->pos() ) )
        emit clicked();
}

/* Handles paint events for the Hello widget.

Flicker-free update. The text is first drawn in the pixmap and the
pixmap is then blt'ed to the screen.
*/

void Hello::paintEvent( QPaintEvent * )
{
    static int sin_tbl[16] = {
        0, 38, 71, 92, 100, 92, 71, 38, 0, -38, -71, -92, -100, -92, -71, -38};
    if ( t.isEmpty() )
        return;
    // 1: Compute some sizes, positions etc.
    QFontMetrics fm = fontMetrics();
    int w = fm.width(t) + 20;
    int h = fm.height() * 2;
    int pmx = width()/2 - w/2;
    int pmy = height()/2 - h/2;

    // 2: Create the pixmap and fill it with the widget's background
    QPixmap pm( w, h );
    pm.fill( this, pmx, pmy );

    // 3: Paint the pixmap. Cool wave effect
```

```
QPainter p;
int x = 10;
int y = h/2 + fm.descent();
int i = 0;
p.begin( &pm );
p.setFont( font() );
while ( !t[i].isNull() ) {
    int i16 = (b+i) & 15;
    p.setPen( QColor((15-i16)*16, 255, 255, QColor::Hsv) );
    p.drawText( x, y-sin_tbl[i16]*h/800, t.mid(i,1), 1 );
    x += fm.width( t[i] );
    i++;
}
p.end();
// 4: Copy the pixmap to the Hello widget
bitBlt( this, pmx, pmy, &pm );
}
```

④ 编写主函数 main()

一个 Qt/Embedded 应用程序应该包含一个主函数，主函数所在的文件名是 main.cpp。主函数是应用程序执行的入口点。以下是 Hello, World 例子的主函数文件 main.cpp 的实现代码。

```
/*
** 以下是 main.cpp 源代码
**
#include "hello.h"
#include <qapplication.h>
```

```
/*
   The program starts here. It parses the command line and builds a message
   string to be displayed by the Hello widget.
*/
#define QT_NO_WIZARD

int main( int argc, char **argv )
{
    QApplication a(argc, argv);
    Hello dlg;
    QObject::connect( &dlg, SIGNAL(clicked()), &a, SLOT(quit()) );
    a.setMainWidget( &dlg );
    dlg.show();
    return a.exec();
}
```

⑤ 编辑工程文件 hello.pro 文件

到目前为止，为 Hello, World 例子编写了一个头文件和两个源文件，这 3 个文件应该被包括在工程文件中，因此还需要编辑 hello.pro 文件，加入这 hello.h, hello.cpp, main.cpp 这三个文件名。具体定义如下

```
/*
*****
** 以下是 hello.pro 文件的内容
*****
/

TEMPLATE    = app
CONFIG      = qt warn_on release
HEADERS     = hello.h
SOURCES     = hello.cpp \
```

```
main.cpp  
INTERFACES =
```

⑥ 生成 Makefile 文件

编译器是根据 Makefile 文件内容来进行编译的，所以需要生成 Makefile 文件。Qt 提供的 tmake 工具可以帮助我们从工程文件（.pro 文件）中产生 Makefile 文件。结合当前例子，要从 hello.pro 生成一个 Makefile 文件的做法是：首先查看环境变量 \$TMAKEPATH 是否指向 arm 编译器的配置目录，在命令行下输入以下命令

```
echo $TMAKEPATH
```

如果返回的结果的末尾不是/qws/linux-arm-g++ 的字符串，那您需要把环境变量 \$TMAKEPATH 所指的目录设置为指向 arm 编译器的配置目录，过程如下，

```
export TMAKEPATH = /tmake 安装路径/qws/linux-arm-g++
```

同时，应确保当前的 QTDIR 环境变量指向 Qt/Embedded 的安装路径，如果不是，则需要执行以下过程

```
export QTDIR = ...../qt-2.3.7
```

上述步骤完成后，就可以使用 tmake 生成 Makefile 文件，具体做法是在命令行输入以下命令：

```
tmake -o Makefile hello.pro
```


这样就可以看到当前目录下新生成了一个名为 Makefile 的文件。下一步，需要打开这个文件，做一些小的修改。

(1) 将 `LINK = arm-linux-gcc` 这句话改为

```
LINK = arm-linux-g++
```

这样做是因为要是用 `arm-linux-g++` 进行链接

(2) 将 `LIBS = $(SUBLIBS) -L$(QTDIR)/lib -lm -lqte` 这句话改为

```
LIBS = $(SUBLIBS) -L/usr/local/arm/2.95.3/lib -L$(QTDIR)/lib -lm  
-lqte
```

这是因为链接时要用到交叉编译工具 `toolchain` 的库。

⑦ 编译链接整个工程

最后就可以在命令行下输入 `make` 命令对整个工程进行编译链接了。

```
make
```

`make` 生成的二进制文件 `hello` 就是可以在 YL9200 上运行的可执行文件。下面将介绍如何将这个文件发布到 YL9200 上。

5. 发布一个 Qt/Embedded 应用到 YL9200

在优龙公司的《YL9200 使用手册》的第五章“Linux 的引导和烧写”介绍了如何把 Linux 的引导和内核的映像文件烧写到 YL9200 的 FLASH 上。一个 Qt/Embedded 应用的运行需要有 Linux 操作系统和 Qt/Embedded 库的支持。所以我们除了要烧写 Linux 到 YL9200 的 FLASH 存储空间之外，我们还要烧写 Qt/Embedded 的二进制库到 YL9200 的 FLASH。

一般来说，先把 Qt/Embedded 的二进制库复制到某个目录下，然后再把这个目录制成某种类型的根文件系统，最后把这个根文件系统烧写到 YL9200 的 FLASH 上，这个过程可能需要一些制作根文件系统的工具，例如 mkcramfs。在优龙公司的 YL9200 开发套件中附有的一张光盘中，有一个叫 Linux 的文件夹，包含了一个名为 qtopia.cramfs 的根文件系统映象，这个根文件系统包含了 trolltech 公司使用 Qt/Embedded 开发的一个 PDA 应用环境（简称 QPE），当用户把这个根文件系统映象烧写到 YL9200 的 FLASH 存储空间后，在 YL9200 的根文件系统下就包含了一个 Qt/Embedded 2.3.7 版本的二进制库文件。

鉴于目前发行的套件中所安装的 arm linux 均采用只读文件系统作为其根文件系统，因此其目录大多是不可写的。只有 /var， /tmp 是 RAM 盘可写，但板子一掉电里面的内容就丢失了，因此只能作临时文件保存，无法永久的保存数据，例如配置文件等。但是 /var， /tmp 这些目录可作为调试程序的目录，可以把编译好的 Qt/Embedded 应用程序传送到这些目录，并运行。

下面简单介绍一下如何把一个 Qt/Embedded 应用程序传送到 arm linux 的根文件系统的 /tmp 目录。

- ① 用 YL9200 开发套间中的串口线连接好开发用的机器的串口和 YL9200 开发板的串口。
- ② 在 WINDOWS 操作系统的程序|附件|通讯中运行一个超级终端，并设定通过串口 x 与 YL9200 进行连接。具体设置过程见下列图示：



图三 输入连接名称



图四 输入连接所使用的串口的端口号



图五 设置串口通讯的参数

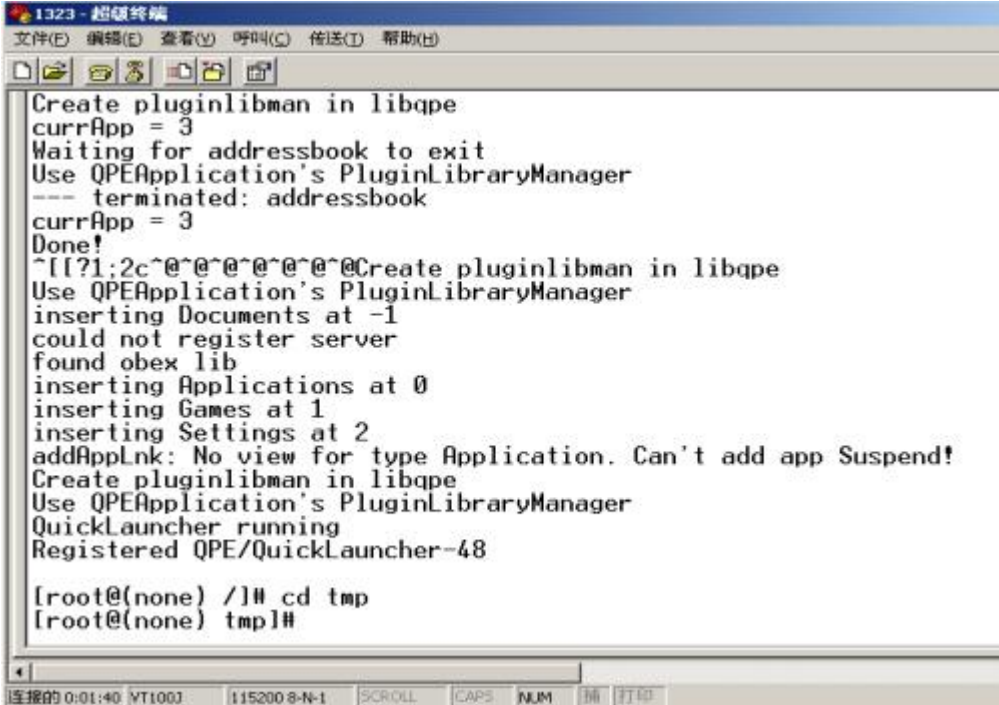
- ③ 打开 YL9200 的电源，或者复位 YL9200，这时可在超级终端的窗口中看到 BIOS 引导信息，如下图

```
channel QPE/Application/quicklauncher added
channel QPE/QuickLauncher-43 added
Registered QPE/QuickLauncher-43
Cannot suspend - no APM support in kernel

Power on reset
*****
*                               *
*   FS2410 Board BIOS           *
*                               *
*****
Please visit Http://www.uCdragon.com for more details
NAND Flash Boot

Please select function :
0 : USB download file
1 : Uart download file
2 : Write Nand flash with download file
3 : Load Program from Nand flash and run
4 : Erase Nand flash regions
5 : Write NOR flash with download file
6 : Set boot params
7 : Test Power off
```

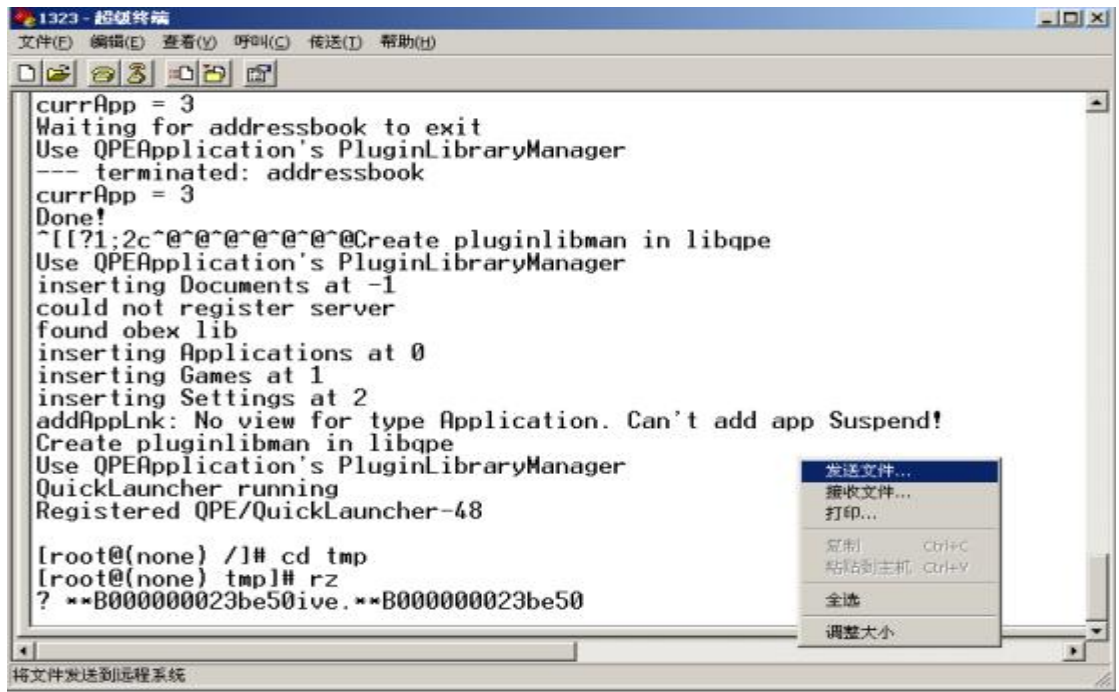
- ④ 待 Linux 引导完成后，在当前文件系统的根目录下，运行“cd tmp”命令，进入到 tmp 目录，如下图



```
1323 - 超频终端
文件(F) 编辑(E) 查看(V) 呼叫(C) 传送(T) 帮助(H)
Create pluginlibman in libqpe
currApp = 3
Waiting for addressbook to exit
Use QPEApplication's PluginLibraryManager
--- terminated: addressbook
currApp = 3
Done!
^[[?1:2c^@^@^@^@^@^@Create pluginlibman in libqpe
Use QPEApplication's PluginLibraryManager
inserting Documents at -1
could not register server
found obex lib
inserting Applications at 0
inserting Games at 1
inserting Settings at 2
addAppLnk: No view for type Application. Can't add app Suspend!
Create pluginlibman in libqpe
Use QPEApplication's PluginLibraryManager
QuickLauncher running
Registered QPE/QuickLauncher-48

[root@(none) /]# cd tmp
[root@(none) tmp]#
```

⑤ 在当前命令行下输入“rz”命令，开始一个文件传输过程；接着点击鼠标右键，在弹出的菜单中，点击“发送文件”子菜单，然后选择你要传送到 tmp 目录的 Qt/Embedded 应用程序。



例如，下图是正在传送 Hello, World 例子的可执行文件到 tmp 目录



⑥ 最后就可以在 tmp 目录下运行传送过来的一个 Qt/Embedded 应用程序。

6. 添加一个 Qt/Embedded 应用到 QPE

前面提到了优龙光盘里面有一个名为 qtopia.cramfs 的根文件系统映象，烧写这个根文件系统到 YL9200 的 FLASH，就会安装有 QT PDA 应用环境（简称 QPE）。可以在 QPE 里添加我们自己编写的应用，不过添加的过程需要重新交叉编译 QPE 的源文件。下面我们介绍如何在 QPE 里添加我们编写的 Hello, World 的例子。

① 在工作的机器上解包 qtopia

```
tar zxvf qtopia-free-1.7.x.tar.gz
cd qtopia-free-1.7.x
export QTDIR=$QTDIR
export QPEDIR=$PWD
export PATH=$QPEDIR/bin:$PATH
```

注意在上面已经设定环境变量 QPEDIR 为 QPE 的安装（解包）路径。

② 建立 Hello, World 的例子程序的图标文件

方法是：制作一个 32 X 32 大小的 PNG 格式的图标文件，将该文件存放在 \$QPEDIR/pics/inline 目录下，然后使用以下命令将 \$QPEDIR/pics/inline 目录下的所有图形文件转换成为一个 c 语言的头文件，这个头文件包含了该目录下的图形文件的 rgb 信息。

```
qembed --images $QPEDIR/pics/inline/*.*
> $QPEDIR/src/libraries/qtopia/inlinepics_p.h
```


注意上述的 qembed 是在 qt3. x. x for x11 上才发布有的工具

③ 交叉编译 qtopia

在\$QPEDIR 路径下，运行以下命令

```
cd src
./configure -platform linux-arm-g++
make
cd ..
```

④ 建立应用启动器 (.desktop) 文件

方法是：建立一个文本文件，在文件中添加以下的内容，这些内容指明了应用的名
称，图标名等信息，然后将文件更名为 xxxx.desktop, 保存在\$QPEDIR/apps/applications
目录下。

以下是例子程序的启动器文件 (hello.desktop):

```
[Desktop Entry]
Comment=A Hello Program
Exec=hello
Icon=Hello
Type=Application
Name=hello
```

① 建立根文件系统

在这里利用原有的 qtopia.cramfs 的根文件系统映像，把新建的应用的相关文
件添加到这个根文件系统中。首先要把 qtopia.cramfs 的根文件系统 mount 到工作机器
上来，然后复制这个文件系统的内容到一个临时的 temp 目录，这时可以在 temp\Qtopia
目录下看到一个 qtopia-free-1.7.0 目录，这就是 qtopia.cramfs 的根文件系统里的 qpe

安装目录，接着把新建的应用的相关文件（包括启动器文件，包含了图标的库文件 libqte.so.*，和应用程序的可执行文件）复制到 temp/Qtomia/ qtopia-free-1.7.0 的对应的目录。具体过程如下

```
mkdir /mnt/cram
mount -t cramfs qtopia.cramfs /mnt/cram -o loop
cp -ra /mnt/cram temp/
cp $QPEDIR/apps/Applications/hello.desktop （接下行）
    temp/Qtomia/qtopia-free-1.7.0/apps/Applications/
cp $QPEDIR/lib/libqpe.so.* （接下行）
    temp/Qtomia/qtopia-free-1.7.0/lib/
cp hello temp/Qtomia/qtopia-free-1.7.0/bin （拷贝可执行文件）
mkcramfs temp xxxxxx.cramfs （生成新的根文件系统）
```

将生成的新的根文件系统烧写到 YL9200 的 FLASH 根文件系统区，复位，OK，就可以看到 QPE 里有我们编写的应用的图标了，点击这个图标，程序就成功运行了。当然为了使我们的应用和原先的 QPE 的应用具备统一的界面风格，在编写自己的应用的主函数文件（main.cpp）时，不妨使用 QPE 提供的宏，具体可参考 QPE 应用程序的源文件。下面是我们编写的 Hello, world 程序在 qvfb 的截屏图。



小记

开发 Qt/Embedded 应用对初学者可能是一项艰苦的工作过程，因为需要安装和设置很多的内容，有时候某一过程没有进行可能会导致一些莫名其妙的出错提示。尽管我们的开发文档详细的介绍了嵌入式 Qt 的开发过程，然而还是不能保证初学者一步一步按照我们所描述的去做便可以在编译应用时万无一失，因为 linux 开发包之间有一定的依赖性，这些开发包又从属不同的开发商或组织。我们的建议是您在您的机器上安装 linux 并准备进行开发时，至少要安装一个工作站版的 linux。您需要注意您的机器上安装有 libjpeg, e2fsprogs,

Freetypes 等开发包有否安装，因为上述谈到的 Qt 嵌入式的软件都用到了这些开发包。

您不必为这些繁琐的工作感到沮丧，因为这些工作会让你有机会认识嵌入式软件工作的底层细节。您会把嵌入式软件的开发过程当作是在玩一个锻炼智力的玩具吗？您很快会有喜欢上这种玩具的感觉的！