MINI2440 用 户 手 册

2011-4-21

copyright@2007-2010

版 权 声 明

本手册版权归属广州友善之臂计算机科技有限公司（以下简称“友善之臂”）所有，并保留一切权力。非经友善之臂同意(书面形式)，任何单位及个人不得擅自摘录本手册部分或全部，违者我们将追究其法律责任。

敬告：

在售开发板的手册会经常更新，请在 http://www.arm9.net网站查看最近更新，并下载最新手册，不再另行通知。

更新说明：

日期	改动
2011-4-21	增加了 W35 LCD 的支持
2011-3-5	- Linux 中增加了 3G 无线上网拨号程序，可以支持 WCDMA, CDMA2000, TD-SCDMA 等多种网络制式，使用方法见 2.4.18 章节 - Linux 中增加了图形界面的 USB WiFi 设置程序，可以支持市面上常见的大部分 USB 无线网卡，详见 2.4.11 章节 - 更新了内核和文件系统，以实现上面两项。
2010-8-17	- 升级了交叉编译器为 arm-linux-gcc-4.4.1(EABI) - 增加 QtE-4.6.3, 可以和 Qtopia-2.2.0 共存，无缝切换 - Linux 中增加了 NEC 4.3"LCD 驱动支持 - WindowsCE5 中解决了串口 2 打开时被占用的问题 - WindowsCE5/6 均增加了英文版本的示例工程，及相应的烧写文件 - 增加了 USB WiFi Kits，可以在 linux 下简单的使用如下类型的 USB 无线网卡 品牌: TP-LINK * TL-WN321G+ * TL-WN322G+ * TL-WN422G+ * TL-WN721N * TL-WN722N 品牌: QCOM * LR802UKG 说明: 1. 使用全新的交叉编译器 arm-linux-gcc-4.4.1 在编译 Qtopia-2.2.0 及 QtE-4.6.x 时，均不会再出现“Segmentation Fault”这样的错误;之前的交叉编译器因为过度优化，在某些时候(一般直接体现在使用大尺寸分辨率时)会出现此错误发生。 2. 因为最新的文件系统集成了 Qtopia-2.2.0 和 Qte-4.6.1 这两种图形系统，直接导致体积倍增，已经无法安装在 64M Nand Flash 版本的开发板中，所以自此以后的更新，将不再考虑 64M Nand Flash 版本的开发板。
2010-3-8	✓ 增加了安装 WindowsCE Embedded 6.0 试用版及补丁等相关文件在微软网站的下载地址(见第九章开头部分) ✓ 增加了安装 Platform Builder 5.0 试用版及补丁等相关文件在微软网站的下载地址(见第十章开头部分)
2010-02-18	在 2.5.18 一节中增加了登录开发板时的 ftp 帐号和密码说明
2010-02-04	✓ 修正了一些笔误 ➢ 增加了 2.8 体验 WindowsCE 5.0 2.8 体验 WindowsCE 5.0 - 164 - 2.8.1 按键测试- 164 - 2.6.2 LED 测试- 165 - 2.8.3 ADC 转换 - 166 -

2.8.4 I2C-EEPROM 读写 - 167 -
2.8.5 PWM 控制蜂鸣器 - 168 -
2.8.6 看门狗 - 169 -
2.8.7 LCD 测试- 171 -
2.6.8 CMOS 摄像头预览拍照 - 171 -
2.8.9 录音测试- 173 -
2.8.10 屏幕旋转并保存 - 174 -
2.8.11 串口助手 - 175 -
2.8.12 触摸屏校正 - 177 -
2.8.13 设置网络参数以连接互联网 - 179 -
2.8.14 背光设置 - 180 -
2.8.15 设置实时时钟并保存 - 182 -
2.8.16 设置程序开机自动运行 - 182 -
2.8.17 使用优盘 - 184 -
2.8.18 使用 SD/MMC 卡 - 185 -
2.8.19 使用 ActiveSync 进行 USB 同步通讯 - 185 -
2.8.20 使用 USB 无线网卡 - 186 -

- 增加了第十章 WindowsCE 5.0 开发指南

第十章 WindowsCE 5.0 开发指南 - 500 -
10.1 基于 WindowsCE5.0 的开发环境- 500 -
10.1.1 安装 Platform Builder 5.0(含 2007 最新补丁) - 500 -
10.1.2 导入安装 BSP - 511 -
10.1.3 安装无线网卡驱动程序 - 514 -
10.1.4 编译内核工程示例 - 517 -
10.1.5 导出 SDK - 521 -
10.1.6 安装 Embedded Visual C++(EVC) - 527 -
10.1.7 安装 EVC 补丁和导出的 SDK - 533 -
10.1.8 定制 CE 内核 - 542 -
10.1.9 制作 WindowsCE 开机画面 StartLogo - 556 -
10.1.10 BootLoader 之 Nboot 的编译和烧写 - 560 -
10.1.11 把 NBOOT 烧写到 Nand Flash - 563 -
10.2 使用 ActiveSync 与 PC 同步 - 565 -
10.2.1 安装 ActiveSync - 565 -
10.2.2 为同步通讯安装 USB 驱动 - 570 -
10.2.3 使用 ActiveSync 同步传输工具复制文件 - 573 -
10.2.4 使用 ActiveSync 与 Platform Builder 连接实现通讯并屏幕截图 - 576 -
10.2.5 使用 ActiveSync 与 Platform Builder 在线编辑注册表- 583 -
10.3 创建 EVC 的 Hello,World, 并编译下载到开发板运行 - 584 -
10.4 创建 VS2005/2008 应用程序, 并编译下载到开发板运行 - 591 -
10.4.1 创建项目 - 592 -

	10.4.2 设置连接开发板 - 594 - 10.4.3 编译下载程序到开发板运行 - 597 - 10.5 LED 驱动程序编写及测试示例 - 598 - 10.5.1 了解硬件连接 - 599 - 10.5.2 编写 LED 流式驱动程序- 600 - 10.5.3 把 LED 驱动程序添加到 BSP 中以编译 - 607 - 10.5.4 编写并编译 LED 测试应用程序 - 609 - 10.5.5 把 LED 测试程序添加到内核，并建立桌面快捷方式 - 612 -

目　　录

MINI2440 用 户 手 册 ..... - 1 -
2011-4-21 ..... - 1 -
第一章 Mini2440 开发板介绍 ..... - 15 -
1.1 Mini2440 开发板简介 ..... - 15 -
1.1.1 Mini2440 开发板外观 ..... - 16 -
1.1.2 Mini2440 开发板硬件资源特性 ..... - 16 -
1.1.3 Linux系统特性 ..... - 18 -
1.1.4 WindowsCE 5.0 系统特性 ..... - 19 -
1.1.5 WindowsCE 6.0 系统特性 ..... - 21 -
1.1.6 资料光盘资源说明 ..... - 22 -
1.2 接口布局及跳线 ..... - 23 -
1.2.1 跳线说明 ..... - 23 -
1.2.2 接口布局 ..... - 23 -
1.3 接口资源说明 ..... - 24 -
1.3.1 地址空间分配和片选信号定义 ..... - 24 -
1.3.2 SDRAM存储系统 ..... - 26 -
1.3.3 FLASH存储系统 ..... - 26 -
1.3.4 电源系统及接口 ..... - 27 -
1.3.5 复位系统 ..... - 29 -
1.3.6 用户LED ..... - 29 -
1.3.7 用户按键 ..... - 30 -
1.3.8 A/D输入测试 ..... - 30 -
1.3.9 PWM控制蜂鸣器 ..... - 31 -
1.3.10 串口 ..... - 31 -
1.3.11 USB接口 ..... - 32 -
1.3.12 LCD接口 ..... - 33 -
1.3.13 EEPROM ..... - 34 -
1.3.14 网络接口 ..... - 34 -
1.3.15 音频接口 ..... - 35 -
1.3.16 JTAG接口 ..... - 36 -
1.3.17 GPIO ..... - 37 -
1.3.18 CMOS CAMERA接口 ..... - 38 -
1.3.19 系统总线接口 ..... - 39 -
第二章 Mini2440 开发板使用说明 ..... - 41 -
2.1 开发板设置及连接 ..... - 41 -
2.1.1 启动模式选择 ..... - 41 -
2.1.2 外部接口连接 ..... - 41 -
2.1.3 设置超级终端 ..... - 41 -
2.2 开发板BIOS功能及使用说明 ..... - 45 -

2.2.1 开机进入BIOS模式 ..... - 45 -
2.2.2 安装USB下载驱动 ..... - 46 -
2.2.3 功能主菜单说明 ..... - 52 -
2.2.4 设置Linux启动参数子菜单功能说明 ..... - 53 -
2.3 非操作系统下的外围资源测试 ..... - 56 -
2.3.1 下载运行测试程序 ..... - 56 -
2.3.2 外围资源测试 ..... - 60 -
2.4 初试Linux之图形界面Qtopia 2.2.0 系统 (预装) ..... - 68 -
2.4.1 触摸屏校正 ..... - 69 -
2.4.2 主要界面说明 ..... - 70 -
2.4.3 播放Mp3 ..... - 71 -
2.4.4 播放视频 ..... - 71 -
2.4.5 图片浏览 ..... - 72 -
2.4.6 自动装载SD卡和优盘 ..... - 75 -
2.4.7 计算器 ..... - 76 -
2.4.8 命令终端 ..... - 77 -
2.4.9 文件浏览器 ..... - 78 -
2.4.10 网络设置 ..... - 79 -
2.4.11 使用无线网络WiFi ..... - 80 -
2.4.12 Ping测试 ..... - 85 -
2.4.13 浏览器 ..... - 86 -
2.4.14 LED测试 ..... - 87 -
2.4.15 EEPROM读写测试 ..... - 88 -
2.4.16 PWM控制蜂鸣器 ..... - 90 -
2.4.17 串口助手 ..... - 91 -
2.4.18 使用 3G上网卡拨号上网 ..... - 93 -
2.4.19 录音 ..... - 97 -
2.4.20 使用USB摄像头拍照 ..... - 98 -
2.4.21 CMOS摄像头预览拍照 ..... - 99 -
2.4.22 LCD测试 ..... - 100 -
2.4.23 背光控制 ..... - 101 -
2.4.24 A/D转换 ..... - 102 -
2.4.25 按键测试 ..... - 103 -
2.4.26 触摸笔测试 ..... - 104 -
2.4.27 条码扫描 ..... - 106 -
2.4.28 语言设置 ..... - 106 -
2.4.29 设置时区-日期-时间-闹钟 ..... - 108 -
2.4.30 屏幕旋转 ..... - 109 -
2.4.31 设置开机自动运行程序 ..... - 111 -
2.4.32 关于关机 ..... - 112 -
2.4.33 看门狗 ..... - 113 -

2.4.34 启动QtE-4.6.3 - 115 -
2.5 通过串口终端操作开发板 - 116 -
2.5.1 播放mp3 - 117 -
2.5.2 如何中止程序的运行 - 117 -
2.5.3 使用优盘/移动硬盘 - 118 -
2.5.4 使用SD卡 - 119 -
2.5.5 如何通过串口与PC互相传送文件 - 121 -
2.5.6 控制板上的LED - 123 -
2.5.7 测试板上的按键 - 124 -
2.5.8 串口 2 和 3 的测试 - 125 -
2.5.9 测试蜂鸣器 - 126 -
2.5.10 控制LCD的背光 - 127 -
2.5.11 测试I2C－EEPROM - 127 -
2.5.12 AD转换测试 - 129 -
2.5.13 CMOS摄像头动态预览 - 129 -
2.5.14 使用telnet上bbs - 130 -
2.5.15 如何设置网络以访问互联网网址 - 132 -
2.5.16 如何设置MAC地址 - 134 -
2.5.17 如何使用Telnet远程登录开发板 - 136 -
2.5.18 使用ftp传递文件 - 137 -
2.5.19 通过网页控制板上的LED - 137 -
2.5.20 如何挂接使用网络文件系统NFS - 138 -
2.5.21 使用USB无线网卡 - 139 -
2.5.22 设置并保存系统实时时钟 - 141 -
2.5.23 如何掉电保存数据到Flash - 141 -
2.5.24 如何设置开机自动运行程序 - 142 -
2.5.25 如何使用命令进行屏幕截图 - 143 -
2.6 预装WindowsCE5/6 系统的使用和设置 - 144 -
2.6.1 按键测试 - 145 -
2.6.2 LED测试 - 146 -
2.6.3 ADC转换 - 146 -
2.6.4 I2C-EEPROM读写 - 147 -
2.6.5 PWM控制蜂鸣器 - 148 -
2.6.6 看门狗 - 149 -
2.6.7 LCD测试 - 151 -
2.6.8 CMOS摄像头预览拍照 - 151 -
2.6.9 录音测试 - 153 -
2.6.10 屏幕旋转并保存 - 154 -
2.6.11 串口助手 - 155 -
2.6.12 触摸屏校正 - 157 -
2.6.13 设置网络参数以连接互联网 - 159 -

2.6.14 背光设置 ..... - 160 -
2.6.15 设置实时时钟并保存 ..... - 162 -
2.6.16 设置程序开机自动运行 ..... - 162 -
2.6.17 使用优盘 ..... - 164 -
2.6.18 使用SD/MMC卡 ..... - 165 -
2.6.19 使用ActiveSync进行USB同步通讯 ..... - 165 -
2.6.20 关于USB无线网卡 ..... - 166 -
2.7 安装使用第三方软件 ..... - 166 -
2.7.1 输入法 ..... - 167 -
2.7.1.1 蒙恬手写输入 ..... - 167 -
2.7.2 实用工具 ..... - 168 -
2.7.2.1 小画笔 ..... - 168 -
2.7.2.2 计算器 ..... - 169 -
2.7.2.3 记事本 ..... - 169 -
2.7.2.4 截图工具 ..... - 170 -
2.7.3 办公软件 ..... - 170 -
2.7.3.1 文字处理浏览器 ..... - 170 -
2.7.3.2 电子表格浏览器 ..... - 171 -
2.7.3.3 幻灯片浏览 ..... - 171 -
2.7.3.5 图片浏览器 ..... - 172 -
2.7.3.5 pdf阅读器 ..... - 172 -
2.7.4 媒体播放 ..... - 173 -
2.7.4.1 TCPMP ..... - 173 -
2.7.4.2 CorePlayer ..... - 173 -
2.7.4.3 Flash播放 ..... - 174 -
2.7.5 图形图像 ..... - 174 -
2.7.5.1 Photoshop ..... - 174 -
2.7.6 网络软件 ..... - 175 -
2.7.6.1 UCWEB浏览器 ..... - 175 -
2.7.7 休闲娱乐 ..... - 176 -
2.7.7.1 BUBBLETS ..... - 176 -
2.7.7.2 中国象棋 ..... - 176 -
2.7.7.3 游戏套装(33 个) ..... - 177 -
2.8 体验WindowsCE 5.0 ..... - 177 -
2.8.1 按键测试 ..... - 177 -
2.6.2 LED测试 ..... - 178 -
2.8.3 ADC转换 ..... - 179 -
2.8.4 I2C-EEPROM读写 ..... - 180 -
2.8.5 PWM控制蜂鸣器 ..... - 181 -
2.8.6 看门狗 ..... - 182 -
2.8.7 LCD测试 ..... - 184 -

2.6.8 CMOS摄像头预览拍照 - 184 -
2.8.9 录音测试 - 186 -
2.8.10 屏幕旋转并保存 - 187 -
2.8.11 串口助手 - 188 -
2.8.12 触摸屏校正 - 190 -
2.8.13 设置网络参数以连接互联网 - 192 -
2.8.14 背光设置 - 193 -
2.8.15 设置实时时钟并保存 - 195 -
2.8.16 设置程序开机自动运行 - 195 -
2.8.17 使用优盘 - 197 -
2.8.18 使用SD/MMC卡 - 198 -
2.8.19 使用ActiveSync进行USB同步通讯 - 198 -
2.8.20 使用USB无线网卡 - 199 -
第三章 备份恢复系统及安装更新 - 201 -
3.1 备份和恢复系统 - 201 -
3.1.1 备份系统 - 201 -
3.1.2 使用备份文件恢复系统 - 206 -
3.2 安装Linux系统 - 209 -
3.2.1 分区 - 209 -
3.2.2 安装bootloader - 211 -
3.2.3 安装Linux内核 - 213 -
3.2.4 安装根文件系统 - 214 -
3.3 安装WinCE系统 - 217 -
3.3.1 安装Bootloader - 217 -
3.3.2 下载烧写BootLogo - 219 -
3.3.3 安装wince内核映象 - 220 -
3.4 下载到内存运行 - 223 -
3.4.1 运行 2440test - 223 -
3.4.2 运行uCos2 - 226 -
3.4.3 运行Linux - 230 -
3.4.4 运行WinCE - 233 -
第四章 ADS1.2 集成开发环境的使用 - 236 -
4.1 使用ADS创建LED工程 - 236 -
4.1.1 建立一个工程 - 236 -
4.1.2 编译和链接工程 - 241 -
4.2 使用H-JTAG进行代码调试 - 248 -
4.2.1 为H-JTAG配置AXD DEBUGGER - 248 -
4.2.4 使用H-JTAG在ADS1.2 环境下进行仿真调试 - 250 -
4.3 编译运行烧写 2440test - 251 -
4.3.1 编译和使用H-JTAG调试 2440test - 251 -
4.3.2 通过USB把 2440test下载到内存运行 - 256 -

4.4.3 把 2440test烧写到Nand Flash运行 - 258 -
4.5 uCos2 的编译和烧写 - 261 -
4.5.1 编译uCos2 - 261 -
4.5.2 把uCos2 下载到内存运行 - 263 -
4.5.3 把uCos2 烧写到Nand Flash运行 - 266 -
第五章 建立Linux开发环境 - 269 -
5.1 图解安装Fedora 9.0 - 269 -
5.2 常用设置和服务 - 283 -
5.2.1 添加新用户 - 284 -
5.2.2 访问Windows系统中的文件 - 286 -
5.3 建立交叉编译环境 - 291 -
5.4 解压安装源代码及其他工具 - 293 -
5.4.1 解压安装源代码 - 293 -
5.4.2 解压创建目标文件系统 - 296 -
5.4.3 解压安装必要实用工具 - 296 -
5.5 配置网络文件系统NFS服务 - 297 -
5.5.1 设置共享目录 - 298 -
5.5.2 和启动NFS服务 - 298 -
5.5.3 通过NFS启动系统 - 299 -
第六章 定制Linux内核及制作文件系统 - 302 -
6.1 使用缺省配置文件配置和编译内核 - 302 -
6.2 各个驱动程序源代码位置 - 305 -
6.3 手工定制Linux内核 - 306 -
6.3.1 配置CPU平台选项 - 306 -
6.3.2 配置各个尺寸的LCD驱动以及背光控制支持 - 309 -
6.3.3 配置触摸屏 - 311 -
6.3.4 配置USB鼠标和键盘 - 312 -
6.3.5 如配置优盘的支持 - 313 -
6.3.6 配置万能驱动USB摄像头 - 315 -
6.3.7 配置CMOS摄像头驱动 - 317 -
6.3.8 配置网卡驱动 - 319 -
6.3.9 配置USB无线网卡驱动 - 322 -
6.3.10 配置音频驱动 - 326 -
6.3.11 配置SD/MMC卡驱动 - 329 -
6.3.12 配置看门狗驱动支持 - 330 -
6.3.13 配置LED驱动 - 331 -
6.3.14 配置按键驱动 - 332 -
6.3.15 配置PWM控制蜂鸣器驱动 - 333 -
6.3.16 配置AD转换驱动 - 333 -
6.3.17 配置串口驱动 - 334 -
6.3.18 如何配置RTC实时时钟驱动 - 335 -

6.3.19 配置I2C-EEPROM驱动支持 - 336 -
6.3.20 配置yaff2s文件系统的支持 - 338 -
6.3.21 配置EXT2/VFAT/ NFS等文件系统 - 341 -
6.3.22 制作Linux logo - 344 -
6.4 制作目标板文件系统映象 - 347 -
第七章 嵌入式Linux应用开发入门指南 - 349 -
7.1Hello,World! - 349 -
7.1.1 Hello,World源代码 - 349 -
7.1.2 编译Hello,World - 350 -
7.1.3 把Hello,World下载到开发板运行 - 350 -
7.2 嵌入式Linux程序开发入门 - 353 -
7.2.1 LED测试程序 - 353 -
7.2.2 测试按键 - 354 -
7.2.3 PWM控制蜂鸣器编程示例 - 356 -
7.2.4 I2C-EEPROM编程示例 - 360 -
7.2.5 串口编程示例 - 363 -
7.2.6 UDP网络编程 - 368 -
7.2.7 数学函数库调用示例 - 374 -
7.2.8 线程编程示例 - 375 -
7.2.9 管道应用编程示例-网页控制LED - 377 -
7.2.10 基于C++的Hello,World - 382 -
7.3 最简单的嵌入式Linux驱动程序模块 - 383 -
7.3.1 Hello,Module源代码 - 383 -
7.3.2 把Hello,Module加入内核代码树，并编译 - 384 -
7.3.3 把Hello, Module下载到开发板并安装使用 - 387 -
7.4 简易Linux驱动程序示例 - 387 -
7.4.1 LED驱动程序 - 388 -
7.4.2 按键驱动程序 - 392 -
7.5 编译Qtopia-2.2.0 - 397 -
7.5.1 解压安装源代码 - 397 -
7.5.2 编译和运行x86 版本的Qtopia-2.2.0 - 397 -
7.5.3 编译和运行arm版本的Qtopia-2.2.0 - 398 -
7.6 编译QtE-4.6.3 - 399 -
7.6.1 解压安装源代码 - 399 -
7.6.2 编译和运行arm版本的QtE-4.6.3 - 400 -
第八章 常见bootloader的配置和编译 - 402 -
8.1 编译vboot - 402 -
8.2 配置和编译vivi - 403 -
8.3 配置和编译U-Boot - 405 -
8.3.1 配置和编译U-Boot - 406 -
8.3.2 把U-Boot烧写到开发板 - 407 -

8.4 使用ADS编译YL-BIOS - 409 -
8.4.1 使用ADS编译YL-BIOS - 409 -
8.4.2 把YL-BIOS下载到内存中运行 - 410 -
8.4.3 烧写YL-BIOS到开发板 - 413 -
第九章 WindowsCE 6.0 开发指南 - 416 -
9.1 建立WindowsCE 6.0 开发环境 - 416 -
9.1.1 安装Visual Studio 2005 及补丁 - 418 -
9.1.2 安装Windows CE 6.0 及补丁 - 432 -
9.1.3 安装第三方软件腾讯QQ - 454 -
9.1.4 安装BSP及内核工程示例 - 461 -
9.1.5 各个驱动程序源代码的位置 - 464 -
9.2 配置和编译WindowsCE 6.0 内核及Bootloader - 465 -
9.2.1 缺省内核工程特性简介 - 465 -
9.2.2 编译缺省内核工程示例 - 466 -
9.2.3 编译带腾讯QQ的内核工程示例 - 476 -
9.2.4 编译和烧写BootLoader之NBOOT - 483 -
9.2.5 在BSP中修改LCD类型及串口输出功能 - 489 -
9.2.6 制作和修改Windows CE启动Logo - 490 -
9.2.7 创建SDK - 494 -
9.2.8 安装SDK - 496 -
9.3 与PC同步(基于Windows 7) - 501 -
9.3.1 安装Windows Mobile设备中心实现PC同步 - 502 -
9.4 通过VS2005 创建应用程序，并编译下载到开发板运行 - 506 -
9.4.1 创建项目 - 506 -
9.4.2 设置连接开发板 - 509 -
9.4.3 编译下载程序到开发板运行 - 512 -
第十章 WindowsCE 5.0 开发指南 - 514 -
10.1 基于WindowsCE5.0 的开发环境 - 514 -
10.1.1 安装Platform Builder 5.0(含 2007 最新补丁) - 514 -
10.1.2 导入安装BSP - 525 -
10.1.3 安装无线网卡驱动程序 - 528 -
10.1.4 编译内核工程示例 - 531 -
10.1.5 导出SDK - 535 -
10.1.6 安装Embedded Visual C++(EVC) - 541 -
10.1.7 安装EVC补丁和导出的SDK - 547 -
10.1.8 定制CE内核 - 556 -
10.1.9 制作WindowsCE开机画面StartLogo - 570 -
10.1.10 BootLoader之Nboot的编译和烧写 - 574 -
10.1.11 把NBOOT烧写到Nand Flash - 577 -
10.2 使用ActiveSync与PC同步 - 579 -
10.2.1 安装ActiveSync - 579 -

10.2.2 为同步通讯安装USB驱动 - 583 -
10.2.3 使用ActiveSync同步传输工具复制文件 - 587 -
10.2.4 使用ActiveSync与Platform Builder连接实现通讯并屏幕截图 - 590 -
10.2.5 使用ActiveSync与Platform Builder在线编辑注册表 - 597 -
10.3 创建EVC的Hello,World，并编译下载到开发板运行 - 598 -
10.4 创建VS2005/2008 应用程序，并编译下载到开发板运行 - 605 -
10.4.1 创建项目 - 606 -
10.4.2 设置连接开发板 - 608 -
10.4.3 编译下载程序到开发板运行 - 611 -
10.5 LED驱动程序编写及测试示例 - 612 -
10.5.1 了解硬件连接 - 613 -
10.5.2 编写LED流式驱动程序 - 614 -
10.5.3 把LED驱动程序添加到BSP中以编译 - 621 -
10.5.4 编写并编译LED测试应用程序 - 623 -
10.5.5 把LED测试程序添加到内核，并建立桌面快捷方式 - 626 -
附录 使用H-JTAG快速烧写BIOS到开发板 - 629 -
1. 1 H-JTAG简介 - 629 -
2. 2 安装并设置H-JTAG - 630 -
2. 3 设置Flash型号并烧写BIOS - 634 -
2. 4 常见问题 - 640 -

第一章 Mini2440 开发板介绍

1.1 Mini2440 开发板简介

Mini2440 是一款真正低价实用的 ARM9 开发板，是目前国内性价比最高的一款学习板；它采用 Samsung S3C2440 为微处理器，并采用专业稳定的 CPU 内核电源芯片和复位芯片来保证系统运行时的稳定性。mini2440 的 PCB 采用沉金工艺的四层板设计，专业等长布线，保证关键信号线的信号完整性，生产采用机器贴片，批量生产；出厂时都经过严格的质量控制，配合这本十分详细的手册，可以迅速帮你掌握嵌入式 Linux 和 WinCE 开发的流程，只要有 C 语言基础的人一般 2 周即可入门。

我们认为，嵌入式开发板不仅仅是一片可以看到“点亮 LED 灯或者 LCD”的电路板，只要你赋予它丰富的灵魂-嵌入式软件，一切可以变得更加精彩，甚至可以重生，因此我们一直在不断的更新和加强嵌入式软件的开发，我们由衷地热爱嵌入式系统这个事业，并为此保持激情和追求完美，目前我们提供的 Linux 和 WindowsCE6 系统，在技术和功能上已经达到国内 2440 开发板中最高水平，它们各自的 bootloader 以及 BSP 均是 100%开放的，任何人均可以在我们网站自由下载。

用户可以到我们网站浏览最新通知及下载更新最新的手册和系统网址：http://www.arm9.net

Mini2440 的众多先进特性，可以概括为：

(1)第一款被 Linux 社区广泛支持的国产 2440 开发板(自从 Linux-2.6.31 开始，Mini2440 被正式加入 Linux 官方内核)

(2)第一款在 Linux 下支持万能 USB 摄像头的 2440 开发板(同类产品至今仍在使用近乎绝迹、拍摄效果奇差的中星微摄像头，并且没有图形界面的应用)

(3)第一款统一采用支持 EABI 标准交叉编译器的开发板(同类产品仍在使用拼凑的各个版本的编译器，导致开发效率低下，并且不利于初学者入门掌握)

(4)第一款提供齐全 BSP(基于 Linux-2.6.29(当前为 Linux-2.6.32.2)和 WindowsCE 6.0)的开发板，并全面配有图形界面实用应用程序(同类产品的部分硬件只是各摆设，并且还在盗用友善之臂早期的命令行测试代码)

(5)第一款支持.NET 3.5 和腾讯 QQ 的 2440 开发板(2009 年 10 月，WindowsCE 6.0 补丁 Release 3 发布)

(6)第一款可以让用户无需编程，即可自由简单定制 WindowsCE 开机画面的开发板，并且可在 10 秒内极速启动系统(同类产品的 WindowsCE 系统启动时间一般在 30 秒甚至 70 秒)

(7)第一款完美支持 USB 烧写更新 Linux(support yaffs2)和 WindowsCE 5.0/6.0 的开发板，并且支持整片 Nand Flash 备份到 PC，真正适合批量生产(同类产品尚无备份功能)

(8)第一款实现 WinCE/Linux 图形界面均可使用 CMOS 摄像头预览并拍照的开发板(同

类产品尚无此功能)

(9)第一款公开所有 BSP 源代码(含 Linux 和 WindowsCE)的开发板(齐全的 BSP 让初学者可以了解更多的底层，让真正做开发的人只需关注上层应用)

(10)第一款内核(含 WinCE 和 Linux)同时支持大页和小页 Nand Flash 的开发板，这样新老用户均可享用最新的软件，并且使用方式相同(同类产品尚无此特性)

(11)第一款在 WindowsCE 和 Linux 下通过简单直观的图形界面，就可设置各种程序开机自动运行的开发板(同类产品尚无此功能)

(12)Mini2440 目前是国内零售销量最多的 2440 开发板，靠口碑相传及朋友间互相介绍，Mini2440 已经被众多用户称赞和认同

1.1.1 Mini2440 开发板外观

1.1.2 Mini2440 开发板硬件资源特性

CPU 处理器

- Samsung S3C2440A，主频 400MHz，最高 533Mhz

SDRAM 内存

- 在板 64M SDRAM

- 32bit 数据总线

- SDRAM 时钟频率高达 100MHz

FLASH 存储

在板 256M/1GB Nand Flash, 掉电非易失(用户可定制 64M/128M/256M/512M/1G)

在板 2M Nor Flash，掉电非易失，已经安装 BIOS

LCD 显示

- 板上集成 4 线电阻式触摸屏接口，可以直接连接四线电阻触摸屏
- 支持黑白、4 级灰度、16 级灰度、256 色、4096 色 STN 液晶屏，尺寸从 3.5 寸到 12.1 寸，屏幕分辨率可以达到 1024x768 象素；
- 支持黑白、4 级灰度、16 级灰度、256 色、64K 色、真彩色 TFT 液晶屏，尺寸从 3.5 寸到 12.1 寸，屏幕分辨率可以达到 1024x768 象素；
- 标准配置为统宝 3.5” 真彩 LCD，分别率 240x320，带触摸屏；

接口和资源

- 1 个 100M 以太网 RJ-45 接口(采用 DM9000 网络芯片)
- 3 个串行口
- 1 个 USB Host
- 1 个 USB Slave B 型接口
- 1 个 SD 卡存储接口
- 1 路立体声音频输出接口，一路麦克风接口；
- 1 个 2.0mm 间距 10 针 JTAG 接口
- 4 USER Leds
- 6 USER buttons(带引出座)
- 1 个 PWM 控制蜂鸣器
- 1 个可调电阻，用于 AD 模数转换测试
- 1 个 I2C 总线 AT24C08 芯片，用于 I2C 总线测试
- 1 个 2.0 mm 间距 20pin 摄像头接口
- 板载实时时钟电池
- 电源接口(5V)，带电源开关和指示灯

系统时钟源

- 12M 无源晶振

实时时钟

- 内部实时时钟（带后备锂电池）

扩展接口

1 个 34 pin 2.0mmGPIO 接口

1 个 40 pin 2.0mm 系统总线接口

规格尺寸

- 100 x 100(mm)

操作系统支持

- Linux2.6.32.2 + Qtopia-2.2.0+QtE-4.6.1(独创双图形系统共存，无缝切换)

- WindowsCE.NET 6.0(R3)

1.1.3 Linux 系统特性

版本

- Linux 2.6.32.2(BSP 可自适应 64M/128M/256M/512M/1GB Nand Flash)

支持的文件系统

- yaffs2(可读写的文件系统，推荐使用)
- cramfs(压缩的只读文件系统，不在线更新数据时推荐使用)
- Ext2
- Fat32
- NFS(网络文件系统，开发驱动程序及应用程序时方便使用)

基本驱动程序(以下驱动均以源代码方式提供)

- 3 串口标准驱动
- DM9000 驱动程序
- 音频驱动(UDA1341)(可录音)
- RTC 驱动(可掉电保存时间)
- 用户 LED 灯驱动
- USB Host 驱动
- 真彩 LCD 驱动(含 1024x768VGA 驱动)
- 触摸屏驱动
- 免驱的万能 USB 摄像头驱动
- USB 鼠标、USB 键盘驱动、优盘、移动硬盘
- SD 卡驱动，可支持高速 SD 卡，最大容量可达 32G
- I2C-EEPROM
- PWM 控制蜂鸣器
- LCD 背光驱动
- A/D 转换驱动
- 看门狗驱动(看门狗复位相当于冷复位)

Linux 应用及服务程序

- busybox1.13(Linux 工具集，包含常用 Linux 命令等)
- Telnet、Ftp、inetd(网络远程登录工具及服务)
- boa(web server)
- madplay(基于控制台的 mp3 播放器)
- snapshot(基于控制台的抓图软件)
- ifconfig、ping、route 等(常用网络工具命令)

嵌入式图形系统平台(以源代码方式提供)

- Qt/Embedded 2.2:分为 x86 和 arm 两个版本
- QtE-4.6.3: 为 ARM 版本，内含简单易用的编译脚本

实用的 Qtopia 测试程序

说明：以下程序均为友善之臂独立自主开发，不提供源代码

- A/D 转换测试
- LED 控制
- Buttons 按键测试
- I2C-EEPROM 读写测试
- LCD 测试
- Ping 测试
- 万能免驱 USB 摄像头动态预览并拍照
- 录音机
- Web 浏览器
- 看门狗测试
- 网络设置(可保存参数)
- 背光控制
- 语言设置：可设置中英文
- 随手写：主要用于测试触摸笔的准确性
- MMC/SD 卡和优盘自动挂载和卸载

1.1.4 WindowsCE 5.0 系统特性

版本

- WindowsCE 5.0 (BSP 可自适应 64M/128M/256M/512M/1G Nand Flash)

特性

- 支持.NET 2.0
- 支持 SQL Mobile
- 支持注册表保存
- 支持快速开机启动(10 秒以内)
- 提供了目前国内最完善的 WindowsCE 5.0 BSP(含 bootloader)，并 100%开放源代码，包括:

  1 - PWM 控制蜂鸣器
  2 - CMOS 摄像头

3 - I2C-EEPROM 读写
4 - ADC 模数转换
5 - 看门狗
6 - 用户按键(6 个)
7 - 用户 LED(4 个)
8 - 背光控制和管理(可设定背光关闭时间,并通过按键，触摸屏，键盘，鼠标唤醒)
9 - RTC
10 - DM9000 网卡
11- 大容量高速 SD 卡，最高可达 32G
12- 优盘、USB 键盘、USB 鼠标等
13- 音频播放和录音
14- 触摸屏
15-LCD 驱动(支持 X35，N35/T35/W35, A70, L80, VGA1024x768，分别对应分辨率:240x320, 800x480, 640x480, 1024x768)
16- 通过简单修改头文件定义即可支持多种型号的 LCD(保持和 WinCE5 一致)
17- LCD 旋转设置
18- 完善的 3 个串口驱动(通过简单修改头文件定义可以指定 UART0 作为 DEBUG 输出或者普通串口，保持和 WinCE5 一致)
19- 通过修改 Nboot 头文件可以方便的自定义进度条的颜色、位置、长宽，以及开机图片的位置、背景
20- 增加了方便的 Logo 制作工具 StartLogoMaker(绿色软件，可运行于 XP 或者 Vista, Win7)
21- 通过 USB 可以烧写普通的 bmp 文件作为开机画面

配合以上全新完善的 BSP，增加了以下各种 WindowsCE 实用小程序，基本和 Qtopia Apps 保持相同或类似的界面:

- (1) LCD-Test: LCD 测试
- (2) PWM-Buzzer: PWM 控制蜂鸣器
- (3) CMOS Camera: CMOS 摄像头动态预览并拍照
- (4) I2C-EEPROM: 读写基于 I2C 总线的 EEPROM 测试
- (5) Watchdog: 看门狗测试
- (6) AD-Convert: ADC 转换测试
- (7) Buttons: 按键测试
- (8) Rotate: 屏幕旋转设定(可保存旋转结果)
- (9) Autorun-Setting: 设定开机自动运行程序
- (11) Recorder: 录音测试 (原有,界面稍做调整)，提供测试源代码
- (12) 串口助手: 提供测试源代码
- (13) LED 测试: 提供测试源代码

1.1.5 WindowsCE 6.0 系统特性

版本

- WindowsCE Embedded 6.0 (BSP 可自适应 64M/128M/256M/512M/1G Nand Flash)

特性

- 基于最新的补丁 Release3(2009.10 发布，用户可自行添加移动版 QQ 组件，比较大)
- 支持.NET 3.5
- 支持全盘目录可读写，可以安装更多第三方软件，如蒙恬手写输入法
- 支持快速开机启动(10 秒以内)
- 提供了目前国内最完善的 WindowsCE 6.0 BSP(含 bootloader)，并 100%开放源代码，包括:

1 - PWM 控制蜂鸣器
2 - CMOS 摄像头
3 - I2C-EEPROM 读写
4 - ADC 模数转换
5 - 看门狗
6 - 用户按键(6 个)
7 - 用户 LED(4 个)
8 - 背光控制和管理(可设定背光关闭时间,并通过按键，触摸屏，键盘，鼠标唤醒)
9 - RTC
10 - DM9000 网卡
11- 大容量高速 SD 卡，最高可达 32G
12- 优盘、USB 键盘、USB 鼠标等
13- 音频播放和录音
14- 触摸屏
15-LCD 驱动(支持 X35，N35/T35/W35，A70，L80，VGA1024x768，分别对应分辨率:240x320, 800x480, 640x480, 1024x768)
16- 通过简单修改头文件定义即可支持多种型号的 LCD(保持和 WinCE5 一致)
17- LCD 旋转设置
18- 完善的 3 个串口驱动(通过简单修改头文件定义可以指定 UART0 作为 DEBUG 输出或者普通串口，保持和 WinCE5 一致)
19- 通过修改 Nboot 头文件可以方便的自定义进度条的颜色、位置、长宽，以及开机图片的位置、背景
20- 增加了方便的 Logo 制作工具 StartLogoMaker(绿色软件，可运行于 XP 或者 Vista, Win7)
21- 通过 USB 可以烧写普通的 bmp 文件作为开机画面

配合以上全新完善的 BSP，增加了以下各种 WindowsCE 实用小程序，基本和 Qtopia Apps 保持相同或类似的界面:

- (1) LCD-Test: LCD 测试

- (2) PWM-Buzzer: PWM 控制蜂鸣器
- (3) CMOS Camera: CMOS 摄像头动态预览并拍照
- (4) I2C-EEPROM: 读写基于 I2C 总线的 EEPROM 测试
- (5) Watchdog: 看门狗测试
- (6) AD-Convert: ADC 转换测试
- (7) Buttons: 按键测试
- (8) Rotate: 屏幕旋转设定(可保存旋转结果)
- (9) Autorun-Setting: 设定开机自动运行程序
- (11) Recorder: 录音测试 (原有,界面稍做调整)，提供测试源代码
- (12) 串口助手: 提供测试源代码
- (13) LED 测试: 提供测试源代码

1.1.6 资料光盘资源说明

本开发板提供一张 DVD 资料光盘，主要有以下内容：
(1) Linux 开发相关的各种源代码和工具
包括：

- 统一的交叉编译器 arm-linux-gcc-4.4.3 with EABI
- 最新的 Linux 内核源代码包 Linux-2.6.32.2，包含十分齐全完善的 BSP
- 嵌入式图形界面 Qte/Qtopia 源代码包：x86-qtopia 和 arm-qtopia，分别对应 PC 版本和 ARM 版本，内含编译脚本
- Busybox 源代码包及缺省配置文件
- Bootloader 源代码(vboot)：使用 arm-linux-gcc 交叉编译器编译
- Linux 编程示例：含串口，PWM，AD，EEPROM，多线程等

(2) WindowsCE5/6 开发相关的各种源代码和软件
包括：

- WindowsCE 5.0/6.0 的 BSP，是目前 2440 开发板中最完善的 BSP，100%开放源代码
- Bootloader 源代码(nboot)：使用 ADS 可以编译，WindowsCE 5.0/6.0 共用相同的 bootloader，均可实现快速启动，快速显示开机 Logo，支持 2M 以内的 24-bit 真彩 bmp 图片(16-bit 的 1024x768 的 bmp 图片大小为 1.5M)
- 基于 WindowsCE 5.0/6.0 制作的 SDK，这是 2 个不同版本的 SDK
- Mini2440.pbxml：内核示例工程，汇集了最常用的组件功能，用户可以在此基础上增删适合自己需要的内核工程
- StartLogoMaker：友善之臂自主开发的简单易用的启动 Logo 制作

(3) uCos2 源代码
uCos2 是由一个网友移植提供的，它的功能和性能都十分有限，在此仅为用户学习参考使用
(4) 2440test 源代码
这是由三星原厂的 2440test 改进而来的一个裸机测试程序，使用 ADS 编译，在此仅为用户学习参考使用。测试项目包括：中断方式按键测试，RTC 实时时钟测试，ADC 数模转换测试，IIS 音频播放 wav 测试，IIS 音频录音测试，触摸屏测试，I2C 总线读写 AT24C08 测试，三星

3.5”LCD、640x480 真彩液晶测试，LCD2VGA 输出测试等

(5)Windows 平台下的实用工具软件

为了方便国内用户更加方便的使用开发板套件，我们制作和搜集了一些常用的 Windows 平台的工具软件，并把它们统一放在光盘的“windows 平台工具”目录中，包括：

- ADS 1.2 集成开发环境安装程序
- H-JTAG：支持烧写各种 NOR Flash
- Vmware：著名的虚拟机软件，版本 VMware-workstation-6.5.1
- Dnw：用来实现 USB 下载
- ActiveSync：微软的 PC 同步软件
- Usb 下载驱动：开发板 BIOS 模式下所需的 PC 端驱动程序

(6)各种数据手册和原理图

包含开发板的原理图(分为 pdf 格式和 protel99se 格式)及封装库，开发板上各个芯片的数据手册，开发板配套的 LCD 数据手册，以及 LCD 驱动板原理图等

(7)其他参考资料

为了方便用户学习使用开发板，我们还不定期在网上搜集一些和学习开发板相关的资料，也有一些是由网友无私提供的。

(8)用户手册

开发板的用户手册，用户可以在我们网站下载最新版本的用户手册。

1.2 接口布局及跳线

1.2.1 跳线说明

开发板上只有一个跳线 J2，它用于选择 LCD 驱动板的输入电压，在标准配置中，所接为 3.5 寸 LCD，电压选择为 5V。

1.2.2 接口布局

Mini2440 接口布局如下图所示，它在十分紧凑的 100 x 100mm 面积上精致安排了开发者常用的各种常用接口，并且还引出了供开发测试需要的富余的 IO 口和总线接口：

1.3 接口资源说明

本小节详细介绍了开发板上每个接口或模块的引脚定义和占用的 CPU 资源，光盘中还有本开发板的完整原理图和封装库(分为 pdf 格式和 Protel99SE 格式)，以供开发板参考使用。

1.3.1 地址空间分配和片选信号定义

S3C2440 支持两种启动模式：一种是从 Nand Flash 启动(MINI2440 即是此种)；一种是从 Nor Flash 启动。在此两种启动模式下，各个片选的存储空间分配是不同的，如下图：

上图中，

左边是 nGCS0 片选的 Nor Flash 启动模式下的存储分配图；

右边是 Nand Flash 启动模式下的存储分配图；

说明：SFR Area 为特殊寄存器地址控制

下面是器件地址空间分配和其片选定义

在进行器件地址说明之前，有一点需要注意，nGCS0 片选的空间在不同的启动模式下，映射的器件是不一样的。由上图可以知道：

- 在 NAND Flash 启动模式下，内部的 4K Bytes BootSram 被映射到 nGCS0 片选的空间；
- 在 Nor Flash 启动模式下(非 Nand Flash 启动模式)，与 nGCS0 相连的外部存储器 Nor Flash 就被映射到 nGCS0 片选的空间

SDRAM 地址空间：0x30000000 ~ 0x34000000

1.3.2 SDRAM 存储系统

Mini2440 使用了两片外接的 32M bytes 总共 64M bytes 的 SDRAM 芯片(型号为：HY57V561620FTP/MT48LC16M16A2)，一般称之为内存，它们并接在一起形成 32-bit 的总线数据宽度，这样可以增加访问的速度；因为是并接，故它们都使用了 nGCS6 作为片选，根据 CPU 手册 5-2 中的介绍可知，这就决定了它们的物理起始地址为 0x30000000，下面是摘自 mini2440 原理图中的 SDRAM 部分原理图。

1.3.3 FLASH 存储系统

Mini2440 具备两种 Flash，一种是 Nor Flash，型号为 SST39VF1601(AMD29LV160DB 与此引脚兼容)，大小为 2Mbyte；另一种是 Nand Flash，型号为 K9F1G08,大小为 128M(旧版本为 K9F1208，大小为 64Mbyte)。S3C2440 支持这两种 Flash 启动系统，通过拨动开关 S2，你可以选择从 NOR 还是从 NAND 启动系统。实际的产品中大都使用一片 Nand Flash 就够了，因为我们为了方便用户开发学习，因此还保留了 Nor Flash。

Nand Flash 不具有地址线，它有专门的控制接口与 CPU 相连，数据总线为 8-bit，但这并不意味着 Nand Flash 读写数据会很慢。大部分的优盘或者 SD 卡等都是 Nand Flash 制成的设备。

从下面的原理图可以看出，Nor Flash 采用了 A1-A22 总共 22 条地址总线和 16 条数据总线与 CPU 连接，请注意地址是从 A1 开始的，这意味着它每次最小的读写单位是 2-byte，因此根据原理图，该设计总共可以兼容支持最大 8Mbyte 的 Nor Flash，实际我们的开发板上只用了 A1-A20 条地址线，因为与 A21、A22 相连的 SST39V1601 的相应引脚是悬空的。

1.3.4 电源系统及接口

本开发板的电源系统比较简单，直接使用外接的 5V 电源，通过降压芯片产生整个系统所需要的三种电压：3.3V、1.8V、1.25V。

请注意，本开发板并非面向手持移动设备设计，因此它并不具备完善的电源管理电路。整个系统的电源通断是由 S1 拨动开关控制的，它不能通过软件实现开关机。

为了方便用户外接其他电源，我们还设计了一个电源接口 CON8，它是一个白色 2.0mm 间距的单排插座，中间均为“地”，两侧均为 5V。注意，这两个 5V 并非是相通的，其中一个连接了外部电源的 5V，另外一个则连接了经过拨动开关 S1 之后的 5V。

它们的连接关系和相应的实物标称见下图：

1.3.5 复位系统

本开发板采用专业的复位芯片 MAX811 实现 CPU 所需要的低电平复位，见下图：

1.3.6 用户 LED

LED 是开发中最常用的状态指示设备，本开发板具有 4 个用户可编程 LED，它们直接与 CPU 的 GPIO 相连接，低电平有效(点亮)，详细的资源占用如下表：

	LED1	LED2	LED3	LED4
GPIO	GPB5	GPB6	GPB7	GPB8
可复用为	nXBACK	nXREQ	nXDACK1	nDREQ1
在原理图中的网络名	nLED_1	nLED_2	nLED_3	nLED_4

1.3.7 用户按键

本开发板总共有 6 个用户测试用按键，它们均从 CPU 中断引脚直接引出，属于低电平触发，这些引脚也可以复用为 GPIO 和特殊功能口，为了用户把它们引出作为其他用途，这 6 个引脚也通过 CON12 引出，6 个按键和 CON12 的定义如下：

	K1	K2	K4	K4	K5	K6
对应的中断	EINT8	EINT11	EINT13	EINT14	EINT15	EINT19
复用的 GPIO	GPG0	GPG3	GPG5	GPG6	GPG7	GPG11
特殊功能口	无	nSS1	SPIMISO1	SPIMOSI1	SPICLK1	TCLK1
对应的 CON12 引脚	CON12.1	CON12.2	CON12.3	CON12.4	CON12.5	CON12.6

说明：CON12.7 为电源(3.3V)，CON12.8 为地(GND)

1.3.8 A/D 输入测试

本开发板总共可以引出 4 路 A/D(模数转换)转换通道，它们位于板上的 CON4-GPIO 接口(详见 GPIO 接口介绍)，为了方便测试， AIN0 连接到了 开发板上的可调电阻 W1，原理图如下所示。

1.3.9 PWM 控制蜂鸣器

开发板的蜂鸣器 SPEAKER 是通过 PWM 控制的，原理图如下所示，其中 GPB0 可通过软件设置为 PWM 输出。

1.3.10 串口

S3C2440 本身总共有 3 个串口 UART0、1、2，其中 UART0,1 可组合为一个全功能的串口，在大部分的应用中，我们只用到 3 个简单的串口功能(本开发板提供的 Linux 和 WinCE 驱动也是这样设置的)，即通常所说的发送(TXD)和接收(RXD)，它们分别对应板上的 CON1、CON2、CON3，这 3 个接口都是从 CPU 直接引出的，是 TTL 电平。为了方便用户使用，其中 UART0 做了 RS232 电平转换，它们对应于 COM0，可以通过附带的直连线与 PC 机互相通讯。

CON1，CON2，CON3 在开发板上的位置和原理图中的连接定义对应关系如下图所示。

1.3.11 USB 接口

本开发板具有两种 USB 接口，一个是 USB Host，它和普通 PC 的 USB 接口是一样的，可以接 USB 摄像头、USB 键盘、USB 鼠标、优盘等常见的 USB 外设，另外一种是 USB Slave，我们一般使用它来下载程序到目标板，当开发板装载了 WinCE 系统时，它可以通过 ActiveSync 软件和 Windows 系统进行同步，当开发板装载了 Linux 系统时，目前尚无相应的驱动和应用。为了方便用户通过程序控制 USB Slave 和 PC 的通断，我们设置了 USB_EN 信号，如图，它使用的 CPU 资源为 GPC5。

我们将提更加广泛的 USB Host 外设应用，请经常留意我们网站的更新信息。

1.3.12 LCD 接口

本开发板的 LCD 接口是一个 41Pin 0.5mm 间距的白色座，其中包含了常见 LCD 所用的大部分控制信号(行场扫描、时钟和使能等)，和完整的 RGB 数据信号(RGB 输出为 8：8：8，即最高可支持 1600 万色的 LCD)；为了用户方便试验，还引出了 PWM 输出(GPB1 可通过寄存器配置为 PWM)， 和复位信号(nRESET)，其中 LCD_PWR 是背光控制信号。

另外，37、38、39、40 为四线触摸屏接口，它们可以直接连接触摸屏使用。

图中的 J2 为 LCD 驱动板供电选择信号，目前我们的驱动板都使用 5V 供电。

1.3.13 EEPROM

本开发板具有一个直接连接 CPU 之 I2C 信号引脚的 EEPROM 芯片 AT24C08，它的容量有 256 byte，在此主要是为了供用户测试 I2C 总线而用，它并没有存储特定的参数。

1.3.14 网络接口

本开发板采用了 DM9000 网卡芯片，它可以自适应 10/100M 网络，RJ45 连接头内部已经包含了耦合线圈，因此不必另接网络变压器，使用普通的网线即可连接本开发板至你的路由器或者交换机。

注意：每个开发板的网络 MAC 地址都是相同的，它可以通过软件设定，对于 Linux

用户，本手册 2.4 章节有相关介绍；对于 WinCE 用户，您可以参考 BSP 里面的 DM9000 驱动代码和注册表文件(platform.reg)。

1.3.15 音频接口

S3C2440 内置 I2S 总线接口，可直接外接 8/16 比特的立体声 CODEC，本开发板采用基于 I2S 总线的 UDA1341 芯片实现音频解码系统，该芯片内部寄存器的初始化和设置则是采用 L3-bus 总线连接控制实现的，在这里我们沿用了三星公板的设计，分别使用 CPU 的 GPB2、GPB3、GPB4 端口模拟实现 L3-Bus 规范的 L3MODE、L3DATA、L3CLOCK，它们在初始化完 UDA1341 以后就不再有用了，因此这三条控制线也可以使用普通的单片机模拟实现。

音频系统的输出为开发板上的常用 3.5mm 孔径插座，输入分为两路，一路为板载麦克风，另一路通过 CON10 白色 2.0mm 插座引出。两路音频输入通道的驱动是有些不同的，目前板载 MIC 对应的通道是可以录音使用的，注意：本开发板并非专业的录音设备，音频输入的处理电路很简单，录音时尽量把音源靠近麦克风。

1.3.16 JTAG 接口

当开发板从贴片厂下线，里面是没有任何程序的，这时我们一般通过 JTAG 接口烧写第一个程序，就是 Supervivi，借助 Supervivi 可以使用 USB 口下载更加复杂的系统程序等，这在后面的章节中你可以看到。

除此之外，JTAG 接口在开发中最常见的用途是单步调试，不管是市面上常见的 JLINK 还是 ULINK，以及其他的仿真调试器，最终都是通过 JTAG 接口连接的。标准的 JTAG 接口是 4 线：TMS、 TCK、TDI、TDO，分别为模式选择、时钟、数据输入和数据输出线，加上电源和地，一般总共 6 条线就够了；为了方便调试，大部分仿真器还提供了一个复位信号。

因此，标准的 JTAG 接口是指是否具有上面所说的 JTAG 信号线，并不是 20Pin 或者 10Pin 等这些形式上的定义表现。这就如同 USB 接口，可以是方的，也可以扁的，还可以是其他形式的，只要这些接口中包含了完整的 JTAG 信号线，都可以称为标准的 JTAG 接口。本开发板提供了包含完整 JTAG 标准信号的 10 Pin JTAG 接口，各引脚定义如图。

说明：对于打算致力于 Linux 或者 WinCE 开发的初学者而言，JTAG 接口基本是没有任何意义和用途的，因为大部分开发板都已经提供了完善的 BSP，这包括最常用的串口和网络以及 USB 通讯口，当系统装载了可以运行的 Linux 或者 WinCE 系统，用户完全可以通过这些高级操作系统本身所具备的功能进行各种调试，这时是不需要 JTAG 接口的；即使你可以进行跟踪，但鉴于操作系统本身结构复杂，接口繁多，单步调试犹如大海捞针，毫无意义

可言。想一想你手头使用的 PC 机就知道了，或许你从没有见过甚至听过有谁会在 PC 主板上插一个仿真器，来调试 PCI 这样接口的 WindowsXP 或者 Linux 驱动。这就是为什么你经常见到或者听到那么多人在讲驱动“移植”，因为大部分人都是参考前辈的实现来做驱动的。

JTAG 仅对那些不打算采用操作系统，或者采用简易操作系统(例如 uCos2 等)的用户有用。大部分开发板所提供的 Bootloader 或者 BIOS 已经是一个基本完好的系统了，因此也不需要单步调试。

1.3.17 GPIO

GPIO 是通用输入输出口的简称，本开发板带有一个 34 Pin 2.0mm 间距的 GPIO 接口，标称为 CON4，如图。

实际上，CON4 不仅包含了很多富余的 GPIO 引脚，还包含了一些其他 CPU 引脚，如 AD0-AIN3，CLKOUT 等。你所看到的图中的 SPI 接口、I2C 接口、GPB0 和 GPB1 等，它们其实也是 GPIO，不过是以特殊功能接口来标称定义的，这些都可以通过相应的 CPU 寄存器来设置更改它们的用途，详细的接口资源见下表。

CON4	网络名称	说明(有些端口可复用)	CON4	网络名称	说明(有些端口可复用)
1	VDD5V	5V 电源(输入或者输出)	**2**	VDD33V	3.3V 电源(输出)
3	GND	地	**4**	nRESET	复位信号(输出)
5	AIN0	AD 输入通道 0	**6**	AIN1	AD 输入通道 1

7	AIN2	AD 输入通道 2	**8**	AIN3	AD 输入通道 3
9	EINT0	EINT0/GPF0	**10**	EINT1	EINT1/GPF1
11	EINT2	EINT2/GPF2	**12**	EINT3	EINT3/GPF3
13	EINT4	EINT4/GPF4	**14**	EINT5	EINT5/GPF5
15	EINT6	EINT6/GPF6	**16**	EINT8	EINT8/GPG0
17	EINT9	EINT9/GPG1	**18**	EINT11	EINT11/GPG3/nSS1
19	EINT13	EINT13/GPG5/SPIMISO1	**20**	EINT14	EINT14/GPG6/SPIMOSI1
21	EINT15	EINT15/GPG7/SPICLK1	**22**	EINT17	EINT17/GPG9/nRST1
23	EINT18	EINT18/GPG10/nCTS1	**24**	EINT19	EINT19/GPG11
25	SPIMISO	SPIMISO /GPE11	**26**	SPIMOSI	SPIMOSI /EINT14/GPG6
27	SPICLK	SPICLK /GPE13	**28**	nSS_SPI	nSS_SPI /EINT10/GPG2
29	I2CSCL	I2CSCL/GPE14	**30**	I2CSDA	I2CSDA/GPE15
31	GPB0	TOUT0/ GPB0	**32**	GPB1	TOUT1/ GPB1
33	CLKOUT0	CLKOUT0/GPH9	**34**	CLKOUT1	CLKOUT1/GPH10

1.3.18 CMOS CAMERA 接口

S3C2440 带有 CMOS 摄像头接口，在开发板上通过标称为 CAMERA 的接口引出。它是一个 20 脚 2.0mm 间距的针座，用户可以直接使用我们提供的 CAM130 摄像头模块；其实 CAM130 摄像头模块上面没有任何电路，它只是一个转接板，它直接连接使用了型号为 ZT130G2 摄像头模块，它们的定义如下图所示。

说明：CAMERA 接口是一个复用端口，它可以通过设置相应的寄存器改为 GPIO 使用，下表是它对应引脚的 GPIO 列表

CAMERA	网络名称	可复用为	CAMERA	网络名称	可复用为
1	I2CSDA	GPE15	**2**	I2CSCL	GPE14
3	EINT20	GPG12	**4**	CAMRST	GPJ12

5	CAMCLK	GPJ11	**6**	CAM_HREF	GPJ10
7	CAM_VSYNC	GPJ9	**8**	CAM_PCLK	GPJ8
9	CAMDATA7	GPJ7	**10**	CAMDATA6	GPJ6
11	CAMDATA5	GPJ5	**12**	CAMDATA4	GPJ4
13	CAMDATA3	GPJ3	**14**	CAMDATA2	GPJ2
15	CAMDATA1	GPJ1	**16**	CAMDATA0	GPJ0
17	VDD33V	3.3V 电源	**18**	VDD_CAM	VDD_CAM
19	VDD18V	1.8V 电源	**20**	GND	地

1.3.19 系统总线接口

本开发板上的系统总线接口为 CON5，它总共包含 16 条数据线(D0-D15)、8 条地址线(A0-A6, A24)、还有一些控制信号线(片选、读写、复位等)，CON5 可以向外提供 5V 电压输出；实际上，很少有用户通过总线扩展外设。下面是 CON5 的详细引脚定义说明。

CON5	网络名称	说明(有些端口可复用)	CON5	网络名称	说明(有些端口可复用)
1	VDD5V	5V 电源(输入或者输出)	**2**	GND	地
3	EINT17	中断 17(输入)	**4**	EINT18	中断 18(输入)
5	EINT3	中断 3(输入)	**6**	EINT9	中断 9(输入)
7	nGCS1	片选 1 对应物理地址: 0x08000000	**8**	nGCS2	片选 2 对应物理地址: 0x10000000
9	nGCS3	片选 3 对应物理地址: 0x18000000	**10**	nGCS5	片选 2 对应物理地址: 0x28000000
11	LnOE	读使能信号	**12**	LnWE	写使能

13	nWAIT	等待信号	**14**	nRESET	复位
15	nXDACK0	nXDACK0	**16**	nXDREQ0	nXDREQ0
17	LADDR0	地址 0	**18**	LADDR1	地址 1
19	LADDR2	地址 2	**20**	LADDR3	地址 3
21	LADDR4	地址 4	**22**	LADDR5	地址 5
23	LADDR6	地址 6	**24**	LADDR24	地址 24
25	LDATA0	数据线 0	**26**	DATA1	数据线 1
27	LDATA2	数据线 2	**28**	DATA3	数据线 3
29	LDATA4	数据线 4	**30**	DATA5	数据线 5
31	LDATA6	数据线 6	**32**	DATA7	数据线 7
33	LDATA8	数据线 8	**34**	DATA9	数据线 9
35	LDATA10	数据线 10	**36**	DATA11	数据线 11
37	LDATA12	数据线 12	**38**	DATA13	数据线 13
39	LDATA14	数据线 14	**40**	DATA15	数据线 15

第二章 Mini2440 开发板使用说明

出厂之前，如果客户未加说明，我们一般已经烧写缺省的 linux 系统(包含三个文件，对应的光盘二进制文件是 **supervivi**、zImage_t35/x35/w35、rootfs-qtopia-qt4.img)，请注意以下的操作是基于 WindowsXP 环境的。

注意：光盘中的 supervivi-64M 适用于老版本的 64M 版 mini2440，supervivi-128M 适用于 128M/256M/512M/1GB 版 mini2440

2.1 开发板设置及连接

2.1.1 启动模式选择

本开发板的启动模式选择，是通过拨动开关 S2 来决定的：

根据目标板提示：

S2 接到 Nor Flash 标识一侧时，系统将从 Nor Flash 启动；

S2 接到 Nand Flash 标识一侧时，系统将从 Nand Flash 启动。

出厂的时候开发板的 Nor Flash 和 Nand Flash 已经烧入了相同的 BIOS(因为该 BIOS 同时支持这两种 Flash，只是开机后表现形式不同，请参考“开发板 BIOS 功能及使用说明”一节)，S2 已经被接到 Nand Flash 一侧，系统一开机就从 Nand Flash 启动运行系统。

2.1.2 外部接口连接

- 请使用我们提供的直连串口线连接开发板的串口 0 和 PC 机的串口
- 用我们提供的交叉网线将开发板的网络接口与 PC 相连
- 用我们提供的 5V 电源适配器连接到板上的 5V 输入插座
- 把音箱或者耳机的插头接入板上的音频输出口(绿色)
- 如果您有液晶屏，请按照数据线头的方向与开发板的 LCD 接口相连
- 用 USB 电缆连接开发板和 PC

2.1.3 设置超级终端

注意：有的用户使用 USB 转串口线来扩展串口，但注意有的 USB 转串口线是会出现乱码的，这说明它的性能和功能并不好，我们的代理大部分都提供了这样的转接线，用户可以直接联系代理购买可用的转接线。

另外，请务必使用随机附带的串口直连线，或许其他线是不能正常通讯的，请使用万用表检测确定为直连线即可。

为了通过串口连接开发板，必须使用一个模拟终端程序，几乎所有的类似软件都可以使用，其中MS-Windows 自带的超级终端是最常用的选择，当你安装Windows9x 时需要自定义选择安装该项，Windows2000 及更高版本则已经缺省安装。

一般桌面版Linux系统也自带了类似的串口终端软件，叫minicom，它是基于命令行的程序，使用比较复杂一些，感兴趣的用户可以在网上找一下这方面的介绍。

在此着重介绍一下Windows 自带的超级终端程序并以WindowsXP 为例，或许其他Windows 版本的程序界面有所不同。 超级终端程序通常位于"**开始->程序->附件->通讯**"中，选择运行该程序，一般会跳出如图所示窗口，询问你是否要将Hypertrm作为默认的telnet程序，此时你不需要，因此点“否”按钮。

接下来，会跳出如下窗口，点“取消”

此时系统提示“确认取消”，点“是”即可，接着点提示窗口的“确定”，进入下一步。

超级终端会要求你为新的连接取一个名字，如图所示，这里我取了”ttyS0”，Windows 系统会禁止你取类似”COM1”这样的名字，因为这个名字被系统占用了。

当你命名完以后，又会跳出一个对话框，你需要选择连接开发板的串口，我这里选择了串口 1，如图所示：

最后，最重要的一步是设置串口，注意必须选择无流控制，否则，或许你只能看到输出而不能输入，另外板子工作时的串口波特率是 115200，如图所示。

当所有的连接参数都设置好以后，打开电源开关，系统会出现 vivi 启动界面。选择超级终端“文件”菜单下的“另存为…”，保存该连接设置，以便于以后再连接时就不必重新执行以上设置了。

2.2 开发板 BIOS 功能及使用说明

2.2.1 开机进入 BIOS 模式

说明:本开发板所附光盘提供了两个 supervivi 文件：supervivi-64M 和 supervivi-128M 其中，supervivi-64M 适用于 64M Nand Flash 版 mini2440/micro2440；supervivi-128M 适用于 128M /256M/512M/1GB 版本的 Nand Flash 版 mini2440/micro2440

Supervivi 在出厂的时候已经预装入板子的 Nor Flash 中，设置拨动开关 S2 为 NOR Flash 启动，即可进入 BIOS 模式，此时开发板上的绿色 LED1 会呈现闪烁状态，其启动界面如下图：

Supervivi 简介：

开发板采用的 BIOS 是基于三星原来的 bootloader 之 vivi 改进而来，名为 **Supervivi**，它采用功能菜单的方式，并可以和原来的命令交互模式互相切换。

Supervivi 可以使用 JTAG 板(一般借助 H-JTAG 软件)直接烧写入 Nor Flash 中使用，也可以直接烧入 Nand Flash 中运行。当烧入 Nor Flash 并从中时，将会出现菜单模式；当烧入 Nand Flash 并从中运行时，按下开发板上的任意一个按键，也可以出现菜单模式，否则会启动开发板上预装的操作系统(Linux/WindowsCE)。

Supervivi 的菜单模式主要为烧写系统和调试而用，也可以设置参数和进行分区等，它采用 USB 下载的方式，因此搭建烧写环境极为简单，并且下载速度快，使用十分方便。

如果 Supervivi 被烧写入 Nor Flash(默认)，您不仅可以用它来方便的下载更新 linux 和 WinCE 系统，还可以烧写其他任何支持 Nand Flash 启动的操作系统和非操作系统到 Nand Flash，如 uCos2，U-boot，Nboot，2440test 等，然后再选择系统从 Nand Flash 启动，这样您就可以使用各种各样的系统了，我们将会逐步增加这方面的 Demo 文件，请留意我们的网站信息。

如果 Supervivi 被烧写入 Nand Flash，它可以自动识别您烧写的 Linux 或者 WindowsCE 系统、或者其他系统，并快速自动启动它们。在本手册的“安装和更新系统”一节，我们就直接使用它来作为 bootloader。

另外，使用 Download & Run 功能，您还可以把程序下载到内存马上运行，这对于开发调试是极有帮助的，这样，您甚至不使用仿真器都可以了，我们光盘中的 2440test 程序就是这样一个例子。

使用 supervivi 还可以把 Linux 内核文件 zImage 直接下载到内存中运行，如果您在 supervivi 中设定好网络启动参数，则还可以通过网络启动整个系统；同样的，suerpvivi 也可以把 WinCE 的运行时映像文件 NK.nb0 下载在内存中运行。

2.2.2 安装 USB 下载驱动

注意：此处安装的 USB 驱动仅在 BIOS 模式下有用，它需要配合 dnw.exe 软件使用，进入 Linux 或者 WinCE 系统都不会使用到该驱动。

说明：安装 USB 下载驱动不需要连接开发板，该安装是独立进行的。

双击运行光盘中的“**windows 平台工具\usb 下载驱动\ FriendlyARM USB Download Driver Setup_20090421.exe**”安装程序，开始安装 USB 下载驱动。

出现如图安装界面：

点“下一步”继续：

此时会跳出警告信息提示：

点“仍然继续”，USB 下载驱动会很快安装完毕，如图：

下面我们检测一下 USB 驱动：

首先设置开发板的拨动开关 S2 为 Nor Flash 启动，连接好附带的 USB 线和电源(可以不必连接串口线)。

打开电源开关 S1，如果您是第一次使用，WindowsXP 系统会提示您发现了新的 USB

设备，并出现如图界面，在此选择“否，暂时不(T)”，点“下一步”继续。

出现如图提示，选择“自动安装软件”，点“下一步”继续

出现如图警告界面，点“仍然继续”

至此，第一次使用 USB 下载驱动的步骤就结束了。

此时打开光盘中的 dnw.exe 下载软件，可以看到 USB 连接 OK，如图。

在计算机设备管理器中，你也可以看到相关的 USB 下载驱动信息，如图：

2.2.3 功能主菜单说明

注意：以下通过 USB 下载的功能均配合 DNW 这个程序使用，以下截图所用软件版本可能教老，请以实际为准。

功能**[x]**：对 Nand Flash 进行默认分区,相当于执行命令行的 bon part 0 320k 2368k，此命令仅对 Linux 系统有效。

功能**[v]**：通过 USB 下载 Linux bootloader 到 Nand Flash 的 bootloader 分区

功能**[k]**：通过 USB 下载 Linux 内核到 Nand Flash 的 kernel 分区

功能**[y]**：通过 USB 下载 yaffs 文件系统映象到 Nand Flash 的 root 分区

功能**[a]**：通过 USB 下载用户程序到 Nand Flash 中，一般这样的用户程序为 bin 可执行文件，如 2440test(需要支持超过 4K 限制)、uCos2(开发板中带的 uCos2 支持 nand flash 启动)、U-Boot 等；当然也可以是其他任意大小的 bin 程序。

功能**[n]**：通过 USB 下载 WinCE 之启动程序 Nboot 到 Nand Flash 的 Block0

功能**[l]**：通过 USB 下载 WinCE 启动时的开机 Logo(bmp 格式的图片)

功能**[w]**：通过 USB 下载 WinCE 发行映象 NK.bin 到 Nand Flash

功能**[d]**：通过 USB 下载程序到指定内存地址(通过 DNW 的 Configuration->Option 选项指定运行地址)，并运行。对于本开发板，SDRAM 的物理起始地址是 0x30000000，结束地址是 0x34000000，大小为 64Mbytes，另外 BIOS 本身占用了 0x33DE8000 以上的空间，因此在用 BIOS 的 USB 下载功能时应指定地址在 **0x30000000 - 0x33DE8000** 之间。

功能**[z]**：通过USB下载Linux内核映像文件zImage到内存中，下载地址为0x30008000。

功能**[g]**：运行内存中的 Linux 内核映像，该功能一般配合功能[z]一起使用。

功能**[f]**：擦除 Nand Flash，执行此功能将会擦除整片 Nand Flash 中的数据。(如果您是第一次使用本开发板，请不必担心误操作，您可以根据本手册第三章的步骤恢复到出厂状态)

功能**[b]**：启动系统，如果烧入了 linux 或者 wince，执行从命令将自动辨认识别启动系统。

功能**[s]**：设置 linux 启动参数，详细见子菜单说明

功能**[u]**：备份整个 Nand Flash 中的内容，通过 USB 上传到 PC 存储为一个文件，该功能类似于 PC 系统中经常用的 Ghost 工具。

功能**[r]**：使用备份出来的文件恢复到 Nand Flash

功能**[i]**：版本信息

功能**[q]**：返回 vivi 的命令交互模式，如图

在交互模式下输入 **menu** 命令，则可以返回到菜单模式。

2.2.4 设置 Linux 启动参数子菜单功能说明

通过该子菜单功能，可以更加灵活的启动 Linux 系统，在 BIOS 主菜单执行功能号[s]，进入设置 Linux 启动参数子菜单，如图：

(1)浏览当前参数设置[v]

输入“v”可以浏览当前启动参数设置情况：

(2)设置参数[s]

输入“s”，可以对上面列出的参数进行设置，比较常用的参数有(其他参数建议不要更改)：

- Mach_type
- Linux command line

下面我们分别举例说明如何设置：

开发板默认的 MACH_TYPE 为 1999，假设你编译的内核使用的 MACH_TYPE 是 2000，则可以通过修改 mach_type 参数来正常启动内核，根据提示先输入参数的名字“mach_type”，再输入参数值“2000”(引号不要输入)，更改后记得输入“w”保存设置，如图：

Linux_cmd_line 是经常用到的一个内核启动参数，例如要把内核的启动信息和登录终端改为串口 1（默认是串口 0），则这样修改：

通过浏览参数，可以看到原来的参数：

Linux_cmd_line：noinitrd root=/dev/mtdblock2 init=/linuxrc console=ttySAC0

输入“s”后，根据提示输入要修改的参数“**linux_cmd_line**”，回车，再输入参数值为(因为该参数串中有空格，因此需要输入双引号括起来)：

“noinitrd root=/dev/mtdblock2 init=/linuxrc console=ttySAC1,115200”

如图所示：

这样系统启动的时候，内核的启动信息和登录信息都将在串口 1 出现，而 vivi 的输出信息不会改变，还是从串口 0 出来。

(3)保存配置[w]

当设置更改之后，可以输入“w”保存所作的更改。

(4)恢复默认值[r]

输入“r”可以恢复出厂时的内核启动参数。

(5)返回主菜单[q]

输入“q”可以返回 BIOS 功能主菜单。

2.3 非操作系统下的外围资源测试

在非操作系统下，主要测试 PWM 控制蜂鸣器，RTC 实时时钟测试，AD 转换测试，按键，触摸屏，各种 LCD，红外测试，I2C 总线测试，音频输入与输出，SD 卡功能。

2.3.1 下载运行测试程序

说明：2440test 是一个裸机测试程序，它不是一个操作系统，该程序由三星原厂的同名文件修改而来，我们根据实际情况，更改了输出菜单和各项测试内容，使其更加简洁明了，

为了方便用户测试和使用，我们分别编译出了针对不同类型 LCD 显示输出的可执行二进制文件(见下表)，用户使用本节的步骤通过 USB 下载到内存中即可运行，区别之处在于默认显示 LCD 显示输出的类型不同；实际上它们都是使用相同的代码编译出的，只需在头文件中更改一下 LCD 类型(2440test\inc\Option.h 中“LCD_TYPE” 定义)即可。

文件名	说明	备注
2440test_W35.bin	默认显示输出支持 3.5”LCD(LQ035Q1DG06 或兼容，分辨率 320x240，横向)	(1) 因为它们的代码都是相同的，以下我们把这几个针对不同显示输出的测试文件统称为 **2440test.bin** (2) **2440test 可以兼容 64M/128M 友善之臂 2440 开发板**
2440test_N35.bin	默认显示输出支持 NEC 3.5”LCD	
2440test_x35.bin	默认显示输出支持 Sony 3.5”LCD	
2440test_T35.bin	默认显示输出支持统宝 3.5”LCD	
2440test_L80.bin	默认显示输出支持 Sharp 8”LCD(或兼容)	
2440test_A70.bin	默认显示输出支持群创 7”LCD	
2440test_VGA1024x768.bin	默认显示输出支持 VGA(分辨率：1024x768@70Hz)	

在光盘“images\2440test\”目录中找到 **2440test.bin** 文件，通过 BIOS 下载运行该测试程序，步骤如下：

(1)连接好开发板电源，串口线，USB 线，并**设置拨动开关 S2 为 Nor Flash 启动系统**，分别打开串口超级终端和 DNW，上电启动开发板。

(2)保证 USB 驱动已经安装好(前面已经详细介绍了 USB 驱动的安装方法)，这时可以看到 DNW 的标题栏显示[USB：OK]，如果没有安装好驱动会显示[USB：x]，如图所示：

(3)点 DNW 菜单 Configuration，设置 USB 下载运行地址为 0x30000000

(4)这时在超级终端的 BIOS 功能菜单中选择功能号[d]，出现 USB 下载等待提示信息：

(5)点击 DNW 程序的“USB Port”→“Transmit”，选择 2440test.bin 这个映象文件(在光盘的 images\2440test\目录下面)，接着点“打开”，这样就开始下载了。

(6)下载结束后，会自动运行，出现如下界面，同时在 LCD 上会出现一副“向日葵“画面：

若使用 NEC3.5 寸屏(**2440test.bin 默认**)，会出现如下界面：

说明：2440test 运行起来之后，supervivi 就失去了对 CPU 的控制权，而 2440test 本身是没有 USB 部分支持的，因此此时会出现 USB 断开的情况。

2.3.2 外围资源测试

测试程序运行后，就可以进行相应的外围资源测试了，通过选择测试程序主菜单相应的选项，可以执行测试。

(1)蜂鸣器测试(Test PWM)

在主菜单中，输入“1”，再按“回车”键(即 Enter 键)，将开始进行蜂鸣器测试，蜂鸣器测试运行起来后，将会听到蜂鸣器发出声音。

按“—”键，蜂鸣器频率会降低，按“+”键频率升高，按“ESC”键可以推出该测试，并返回到主菜单中。

(2)实时时钟测试

在测试程序主菜单中，选择“2”，再按“回车”键，可以看到秒钟在不断的变化，这说明 CPU 的 RTC 在正常工作（注意：该时间并不是当前的时间，因为测试程序对其初始化并进行了赋值）

按“ESC”键退出该测试，并返回到主菜单。

(3)AD 测试

在主菜单中，输入“3”，再按“回车”键，开始执行 AD 测试。

用户可以使用螺丝刀调节开发板上的 W1 或者 W2(这两个可调电阻接了 AIN0 和 AIN1)，可以看到 AD 的值在跟随调节电压在不断的变化。

按“ESC”键退出该测试，并返回到主菜单。

(4)按键测试

在主菜单中输入“4”，再按“回车”键，开始执行按键测试，此时按开发板上的 K1-K6 按键进行测试，可以看到串口终端打印相应的按键信息。

按“ESC”键退出该测试，并返回到主菜单。

(5)触摸屏测试

如果选购了 LCD 液晶屏，请使用附带的电缆连接开发板上的 LCD 接口。在主菜单中输入“5”，按“回车”开始进行触摸屏测试，这时用附带的触摸笔点击触摸屏，可以看到串口终端打印触摸点的坐标信息。

按“ESC”键退出该测试，并返回到主菜单。

(6)LCD 或 VGA 模块输出测试

请烧写相应的 2440test 测试程序，主菜单中输入“6”按“回车”键开始执行测试，接着按照提示按任意键，LCD 将不断变化显示，直到最后显示一幅图片结束，并返回主菜单。

(7)I2C 测试

在主菜单中输入“7”，按“回车”键开始执行测试，程序将对 I2C 总线的芯片 AT24C08 进行读写，该测试主要是通过向 AT24C08 写入 0x-0xFF，然后读取出来。

这个测试结束后，会自动回到主菜单。

(8)音频输出测试

先将音箱接到开发板的绿色耳机孔座，在主菜单中输入“8”，按“回车”开始音频输出测试，这时可以从音箱听到 XP 的启动声音。

按“+”或者“－”可以增加或者减小音量，按“ESC”键退出测试，返回主菜单。

(9)SD 卡测试

注意：本测试会破坏 SD 卡中的数据，试用前请备份好 SD 卡中的数据。

先将 SD 卡插入开发板的 SD 卡插座。

在主菜单中输入“9”，按“回车”开始执行测试，程序将对 SD 卡进行读写，并出现如下界面：

该界面显示 SD 卡读写成功，测试完毕，自动退回到主菜单。

(11)测试 CMOS 摄像头

如果您选购了本公司提供的 CAM130 型号的 CMOS 摄像头，可以进行本功能测试。

开机之前，把 CAM130 摄像头模块按照板上箭头方向插到开发板的“CAMERA”排座上，在主菜单中输入“10”，按“回车”开始执行测试，

注意：如果使用的是 7 寸屏或者 VGA 输出模块，LCD 显示界面会有所不同。

使用 NEC3.5”寸屏时，CMOS 摄像头效果：

2.4 初试 Linux 之图形界面 Qtopia 2.2.0 系统 (预装)

说明：Qtopia 2.2.0 是 Qt 公司基于 Qt/Embedded 2.3 库开发的 PDA 版(也是最终版)图形界面系统；自从 Qtopia 2.2.0 之后，Qt 公司就再也没有提供 PDA 版的图形系统了。最新版的 Qtopia 只有手机版本，而且 Qt 公司自从 2009.3 开始已经停止了所有 Qtopia PDA 版和手机版图形系统的授权，但依然继续开发 Qt/Embedded (简称 QtE)库系统。

QtE的最新版本请到 http://qt.nokia.com/查看，本开发板所移植的版本为QtE-4.6.3。

出厂之前，本开发板一般都预装了 Linux+Qtopia 2.2.0+QtE-4.6.3 图形界面，它包含了很多实用的小程序，拿到开发板后，你只要接上电源并开机就可以进行各项功能测试了，这不需要和电脑进行任何连接。

如果你使用的是 VGA 输出模块连接了显示器，还需要准备一个 USB 鼠标插到开发板的 USB Host 端口

另外，本系统支持 USB 鼠标和触摸屏共存，并支持 USB 鼠标和键盘热插拔，你可以同时使用它们。

现在让我们先睹为快吧：

注意：本小节内容的介绍均基于开发板 + 3.5” LCD 的屏幕截图。

使用 3.5 寸平屏时，开机后会先后出现如下显示界面：

2.4.1 触摸屏校正

说明：如果按照下面的步骤没有校正准确，可以接上 USB 鼠标，在“设置”中打开“重校正”，再重新开始。

在两种情况下可以会出现触摸屏校正界面：

1. 如果你按照第三章的步骤重新安装了 Qtopia 系统，重启系统时首先出现如下触摸校正界面，依屏幕提示点击屏幕任何地方开始进行校正；然后依照屏幕提示，使用触摸屏逐步点击“十”型交叉点即可。如果你使用的是横屏 W35，则出现横向的校正界面，如下图：

2. 进入系统后，点“开始->设置”切换到“设置”界面，再点“重校正”图标也会出现校正界面；然后依照屏幕提示，使用触摸屏逐步点击“十”型交叉点即可。

说明：因为我们的演示测试程序都是根据 240x320 的竖屏设计的，对于使用横屏 W35 的用户，为了达到更好的效果，建议您进入系统后先进行旋转(出厂时均已做了旋转设置)，旋转设置如下图所示(如果选择另外一个方向旋转，就会出现乱码，这是 Qtopia 本身的 bug 造成的)，旋转设置完毕，点“设置”->“关机”->“Restart Server”重新启动 Qtopia，即可看到竖屏效果了：

2.4.2 主要界面说明

进入 Qtopia 系统后主界面如下图所示：

可以看到 Qtopia 系统界面上方有五个图标， 它们代表了五类程序/文档，单击任何一个图标都可以进入相应的子类界面，它们都是类似的。

另外点系统界面左下角的“开始”图标，也可以出现五个子类选择菜单，它们和系统界面上方的图标是对应的。

五个子类界面分别如下图所示，根据标题名称，其意自明。

其中“友善之臂”组中所有程序均为友善之臂公司自主开发或移植而来，仅供测试使用，其他子类中程序或为系统自带。

2.4.3 播放 Mp3

在子类“应用程序”中单击“音乐”图标，出现播放器界面，在“Audio”列表中选择一首 mp3 歌曲文件，再点上方的“播放”按钮，开始播放 Mp3 文件。

说明：Audio 列表中的音频文件对应“Documents”子类中的所有有效音频文件。

提示：也可以在“Documents”中直接点击相应的文件名开始播放。

2.4.4 播放视频

在子类“应用程序”中单击“视频”图标，出现播放器界面，在“Video”列表中选择一个视频文件，再点上方的“播放”按钮，开始播放视频。

说明：Video 列表中的音频文件对应“Documents”子类中的所有有效视频文件。

提示：也可以在“Documents”中直接点击相应的文件名开始播放。

2.4.5 图片浏览

在子类“应用程序”中单击“图片”图标打开图片浏览器，首先映入眼帘的是“文档”组中各个图片的缩略图，如果你插入了含有图片的 SD 卡或者优盘，其中的图片文件也会一并全部显示出来，下图是系统自带的 3 张图片，和插入含有其他图片的 SD 卡之后的截图。

Qtopia 2.2.0 系统的图片浏览器比以前的 Qtopia 1.7.0 有了很多改善，它可以对图片进行简单的编辑，而且使用起来也更方便，下面做一些简单的说明。

编辑图片

选择一张图片，单击打开，再点工具栏的笔形编辑按钮，进入编辑状态。

在编辑状态，点工具栏的彩色的圆形按钮，进行颜色调整，见下图。

旋转图片

在编辑状态，点工具栏的顺时针旋转按钮，可以依次以 90 度旋转图片。

剪切图片

在编辑状态，点工具栏的“剪刀”按钮，这时图片会被“蒙”上一层浅色的网，用触摸笔在这层网上任意区域画出一个“矩形”，抬起触笔，在所画的“矩形”上点击一下，这个区域就被单独选择出来了，这时你还可以对这个选区进行其他的调整。

缩放图片

在编辑状态，点工具栏的“放大镜”按钮，可以实现对图片的缩放，如图。

说明：“图片”浏览器列表中的图片文件对应“Documents”子类中的所有有图片文件。

提示：也可以在“Documents”中直接点击相应的文件名开始浏览。

2.4.6 自动装载 SD 卡和优盘

在任何界面状态下，插入普通或者高速 SD 卡(实测最大容量为 32G)或者 USB 移动存储器，稍等片刻，即可在任务栏右下角出现移动存储的图标出现，本系统支持二者同时挂载，点移动存储器的图标，出现如图所示，这时可以像在 Windows 中一样安全移除它们。

MMC/SD 卡或者优盘中的所有文件会在“文档”组中全部显示出来，并且**支持中文文件名显示**，它是不显示目录名称的，如果你的文件太多，那么其列表也是相当可观的。

说明：支持 MMC/SD 卡或者优盘自动挂载是通过友善之臂开发的一个 Qtopia 2.2.0 插件实现的，目前它只识别 MMC/SD 卡或优盘的第一个分区，并且格式为常见的 VFAT/FAT32/FAT16，如果你的优盘或者 SD 卡不能识别，请检查是否为 VFAT/FAT32/FAT16 格式。

此时点“应用程序”组的“系统信息”->“Storage”可以看到 SD 卡和优盘的容量信息，如图：

2.4.7 计算器

在子类“应用程序”中单击“计算器”图标，出现计算器界面，用户可以通过下拉列表选择 Simple, Fraction, Scientific, Conversion 等类型的计算器，如图：

2.4.8 命令终端

“终端”是 Linux 系统中通常用到的交互操作界面，通过“终端”可以运行很多 Linux 命令，查看系统信息等等。

有很多途径可以设置或打开一个“终端”：

在 Linux 系统启动的时候，可以把终端指向串口输出，这样就形成了串口终端，它的输入和输出都是通过串口进行的，无需图形界面，这是嵌入式 Linux 开发中最常用的方式。

在系统启动的时候，也可以把终端输出指向图形显示设备（如 LCD 或者 CRT 等），而把键盘设定为输入，这样就形成了一套独立的“输入输出系统”，它无需借助另外的 PC 即可操作。

当使用了图形显示设备，并且系统软件中增加了图形用户界面(GUI)时，就可以建立一个基于 GUI 系统的“命令终端窗口”，这时既可以通过标准的实体硬件键盘进行交互，也可以通过虚拟的“软键盘”进行交互，此处所讲的就是这种终端方式。

在子类“应用程序”中单击“终端”图标，出现命令终端窗口界面，此时可以接上 USB 键盘(**不要在启动之前接 USB 键盘，否则不能使用**)或者使用屏幕下方的软键盘输入 Linux 命令，你还可以点 Option 菜单中的某些选项进行设置，以改变显示的模式，如图。

2.4.9 文件浏览器

在“友善之臂”程序组中点“文件浏览器”图标打开它，如图：

通过文件浏览器，你可以浏览管理开发板中的目录及文件。

说明：Qtopia-2.2.0 原始版本是不带文件浏览器的，我们移植了 Qtopia-1.7.0 中自带的文件管理器，它们的功能和界面是完全相同的。

2.4.10 网络设置

在子类“友善之臂”程序中，点“网络设置”图标打开相应的界面，如图：

在这里，你可以进行常见的网络参数设置：

- 静态的 IP 地址 – 出厂缺省为 192.168.1.230
- 子网掩码 – 出厂缺省为 255.255.255.0
- 网关 – 出厂缺省为 192.168.1.1
- DNS 解析服务器 IP – 出厂缺省为 192.168.1.1，和网关地址相同
- 网卡的 MAC 地址 – 此地址由驱动程序通过软件设定，是可以修改的，本开发板出厂时所有 MAC 地址都是相同的，为 08:90:90:90:90:90

点“Save”按钮可以保存以上参数，并马上生效，重新启动开发板也可以保留此次的更改设定，与该设置程序相对应的参数文件为/etc/eth0-setting

说明：/etc/eth0-setting 参数文件在系统重装后是不存在的，点“Save”按钮会自动生成；开发板出厂之前需经过测试，因此这个文件是存在的。另外，命令终端的 ifconfig 程序所作的 IP 地址更改对该配置文件没有影响。

其实，Qtopia 本身带有一个网络设置的程序，但配置界面有些复杂，有用户反应其设置也不能有效，为了保持 Qtopia 系统的代码原始性，我们对此并没有深入检测，所以另外自己开发了上面介绍的“网络设置”程序。

Qtopia 自带的网络设置界面如下(注意：其设置不一定有效，本公司不对该程序的设置提供咨询)：

2.4.11 使用无线网络 WiFi

本章节介绍在 Linux 下如何进行无线网络设备，缺省安装的 Linux 系统可以支持市面上大部分 USB WiFi 无线网卡。

2.4.11.1 启动无线网络设备程序

在子类“友善之臂”程序中，找到“无线网络设置”图标并点击：

2.4.11.2 搜索无线 AP(或无线路由器)并连接

启动无线网络设置程序时，将会自动搜索无线 AP（或无线路由器），并列出这些 AP 的 SSID 以及信号强度，如下图所示，如果搜索不到你的无线 AP（或无线路由器），可点击“Scan”按钮重新搜索：

搜索到你的无线 AP 后，如果要进行 WiFi 连接，点击列表上的 ESSID 名称即可，点击后将会弹出以下界面，提示你输入无线 AP 的密码：

输入无线 AP 的密码，如果没有密码则留空即可，点击“Connect”开始连接，连接过程过界面如下所示：

连接完成后，所连接的无线 AP 后面会显示 Connected 字样，如下所示：

如果你启动了以太网，则有可能会弹出以下对话框，让你关闭以太网(ifconfig eth0 down)，否则一些网络程序可能会优先使用以太网而不是 WiFi，在下图中选择 Yes 关闭以太网，想恢复以太网时，可启动友善之臂程序组下的“网络设置”程序 Save 一次即可，或在终端上输入 ifconfig eth0 up 开启：

点击“Net Detail”可查看网络的详细信息，如 IP 地址，DNS 等：

连接成功后，点击主界面上的“Close”按钮可将无线设备程序最小化在状态栏上，如下图所示的状态栏上的 WIFI 小图标，可以通过点击这个小图标返回设置程序主界面：

试试打开浏览用 WiFi 上网：

2.4.11.3 断开无线连接

在无线网络设置程序主界面上点击“Disconnect”即可断开无线连接，如下图所示：

2.4.11.4 配置 IP 地址

在无线网络设置程序主界面上点击“Configure IP”按钮，将启动 IP 配置界面，如下图所示：

点击界面顶部的下拉框，可以选择使用 DHCP 自动分配 IP 地址、或者手动设置 IP 地址这两种方式，如下图所示：

设置完成后，点 Save 保存即可。

2.4.12 Ping 测试

请连接好开发板附带的网线，并设置好有效的网关，DNS 等参数，就可以通过图形界面的 ping 程序来测试网络连通性了。在程序子类“友善之臂”中点“Ping 测试”图标，打开相应界面，如下图：

因为已经设置好了 DNS，所以可支持字符域名和数字 IP 两种方式。默认的 ping 测试次数为 4 次，当勾选上方的“ping forever”后，可以一直 ping，测试结果如下图。

重要提示：**要 ping 互联网域名，必须要设置好正确有效的网关和 DNS，并且保证你的网络确实可以连通互联网。**

点“Start”按钮开始 ping，点“Stop”按钮停止 ping，要关闭“Ping 测试”界面，必须先停止 ping。

说明：ping 是计算机系统中最常见的网络测试工具，不管是各个发行版本的 Linux 系统，还是各种 MS Windows 系统，都可以在命令终端输入“ping”命令。以上的“Ping 测试”程序实际就是调用命令行的 ping，把结果通过图形界面显示出来。

2.4.13 浏览器

在“友善之臂”程序组中，点“浏览器”打开它，点开界面下方的软键盘，在界面上方的地址栏中输入一个网址，再点键盘上的“Ret”(回车)按钮，可以打开相应的网站。

说明：本开发板所用的网络浏览器为 Konqueror/Embedded，它是一个开发源代码的浏览器，具体的移植步骤见附录 1 中 Qtopia 的相关脚本(build-all 脚本中包含了移植该浏览器的所有步骤)。

2.4.14 LED 测试

在“友善之臂”程序中点“LED 测试”图标，打开如下界面：

可以看到界面中只有“Stop led-player”按钮有效，这是因为系统启动的时候开启了 led-player 服务， 开机后你所见到的“流水灯”效果就是这个服务控制的，要单独控制某个 LED，需要先关闭这个服务，释放 LED 资源。÷

操作：

点“Stop led-player”按钮，这时它变为“Start led-player”，同时板上所有灯关闭、“LED Control”框中所有按钮由灰色变为有效(如下图)。

这时点“All On”按钮可以点亮所有 LED，点“All Off”可以关闭所有 LED，勾选左边任意一个框可以点亮相应的 LED，取消勾选左边任意一个框可以熄灭相应的 LED。

当关闭“LED 测试”界面时，会重新开启 led-player 服务。

2.4.15 EEPROM 读写测试

在“友善之臂”程序中点“I2C-EEPROM 测试”图标，打开如下界面：

由上到下依次可以看到：写 EEPROM 按钮、写入字符编辑区、读取 EEPROM 按钮、读出字符编辑区。

点开任务栏上的“软键盘”按钮，在“写入字符编辑区”输入一些 ASC 字符，点“Write Data below into EEPROM via IIC”按钮，这时该按钮变为进度条，并指示正在写入的进度，；点“Read EEPROM Data via IIC”按钮，这时按钮变为进度条，并指示正在读取的进度。如图。

2.4.16　PWM 控制蜂鸣器

在“友善之臂”程序中点“PWM-蜂鸣器测试”图标，打开如下界面：

程序中默认的 PWM 输出为 1000Hz，点“Start”按钮开始驱动蜂鸣器发声，此时可以通过点击“＋”或者“－”按钮改变 PWM 输出的频率，同时也可以听到蜂鸣器输出声音的改变。点“Stop”按钮中止 PWM 输出。

2.4.17 串口助手

提示：在使用该程序之前请连接好你需要测试的串口

- 开发板上的 CON1,2,3 分别对应 CPU 的 UART0,1,2，其中 UART0 已经转换为 RS232 电平，并通过 DB9 插座 COM0 输出，它在启动时已经被设置为 console 终端，因此不能直接使用该程序测试。其他两个串口 CON2,3 需要增加 RS232 转换才可以和 PC 的串口连接使用。(本公司提供了型号为“OneCom”RS232 转换模块，如图)，连接 PC 时请注意你使用的串口线类型(交叉或直连)；不管你使用的是何种串口线，你也可以根据实际情况设置 OneCom 模块上的 COM.2 和 COM.3 跳线。
- 本程序也支持市面上常见的 USB 转串口线，因为目前的笔记本大都没有串口，为了方便开发，我们的很多代理都提供了类似的转接线。把 USB 转串口线接到开发板的 USB Host 端口上，你就可以扩充开发板的串口了。它对应的设备名一般为**/dev/ttyUSB0,1,2,3** 等，这意味着你可以通过 USB Hub 扩展多个 USB 转串口。

在“友善之臂”程序组中点“串口助手”图标，打开相应的程序界面，如图：

从该程序窗口的标题可以看到，默认设置为“**ttySAC1 115200 8N1 [C]**”，它表示默认端口的设置：

- 串口设备：/dev/ttySAC1，它对应 CPU 的第二个串口 UART1
- 波特率：115200
- 数据位：8
- 流控制：无
- 停止位：1
- [C]：表示字符模式，如果是[H]则表示 16 进制模式

上图中有两个“编辑框”区域，上面的“编辑框”是用于显示接收到的数据，它实际上是不能编辑的；下面的“编辑框”可以通过 USB 键盘或者 Qtopia 的软键盘获取输入。

点 **Connect** 按钮，以打开开发板串口/dev/ttySAC1，在窗口下面的编辑框输入一些字符，点 **Send** 按钮，就可以向与它相连的串口设备发送数据了，下图显示的是通过 Windows 超级终端接收的数据截图(注意：与此终端对应的串口也应该设置为 115200 8N1)。

点 Disconnect 断开连接，再点 Setting…按钮进入端口设置界面，如图，在此列出了最常见的一些串口设置参数项：

Comm Port：可以选择 CPU 自带的/dev/ttySAC0,1,2 三个串口；也可以选择 USB 转串口对应的/dev/ttyUSB0,1,2,3

Speed：可以选择各种常见的波特率

Data：可以选择数据位是 8 位/7 位，常见的是 8 位

Hex：此项表示采用 16 进制输入或者显示数据。

2.4.18 使用 3G 上网卡拨号上网

在中国，有三种制式的 3G 无线网络：

- WCDMA：运营商为中国联通，这是最成熟的一种 3G 网，可世界通用
- CDMA2000: 运营商为中国电信
- TD-SCDMA: 运营商为中国移动，中国独有的一种 3G 网制式

对应这三种 3G 无线网，有不同的 3G Modem 可用，最常见的就是 USB 接口的 3G Modem，通常称之为“USB 3G 上网卡”或“USB 上网卡”，我们开发的拨号程序，可以自动探测并支持上百种 USB 上网卡 (在本节后面我们给出了一份支持的 3G 上网卡型号清单)。

下面以 HUAWEI E1750 为例介绍一下它的使用步骤，其他的请根据实际情况以此类推。

Step1：先准备一张可用的 3G SIM 卡，你可以到相应的运营商营业厅去办理

Step2：把 SIM 卡插入上网卡

Step3：把上网卡插到开发板的 USB Host 接口

Step4：开机并打开 3G 拨号程序，这里会把检测到的型号自动列出来，点 OK 继续

Step5：点“Connect”开始连接，连接成功会显示如图界面，为获取到的 IP 地址和 DNS 地址等信息

Step6：把点“Minimize”拨号程序隐藏，就可以使用浏览器进行上网了。

下面给出 Mini6410 支持的 USB 3G 网卡的型号清单，我们推荐你选购经过友善之臂测试过的型号：

经过友善之臂测试 OK 的 USB 3G 网卡型号：

Huawei E169 (CDMA2000)
Huawei E1750/E1550 (WCDMA)
ZTE AC581 (CDMA2000)
ZTE AC8710 (CDMA2000)
ZTE MU351 (TD-SCDMA)

其它支持的 USB 3G 网卡型号：

ZTE 6535-Z
ZTE AC2710 (EVDO)
ZTE AC2726
ZTE K3520-Z
ZTE K3565
ZTE MF110 (Variant)
ZTE MF112

ZTE MF620 (aka "Onda MH600HS")
ZTE MF622 (aka "Onda MDC502HS")
ZTE MF628
ZTE MF638 (aka "Onda MDC525UP")
ZTE WCDMA Stick from BNSL
HuaXing E600 (NXP Semiconductors "Dragonfly")
Huawei E1612
Huawei E1690
Huawei E180
Huawei E270+ (HSPA+ modem)
Huawei E630
Huawei EC168C (from Zantel)
Huawei K3765
Huawei K4505
Huawei R201
Huawei U7510 / U7517
Huawei U8110 (Android smartphone)
Onda MW833UP
A-Link 3GU
AT&T USBConnect Quicksilver (made by Option, HSO driver)
AVM Fritz!Wlan USB Stick N
Alcatel One Touch X020 (aka OT-X020, aka MBD-100HU, aka Nuton 3.5G), works with Emobile D11LC
Alcatel X200/X060S
Alcatel X220L, X215S
AnyDATA ADU-500A, ADU-510A, ADU-510L, ADU-520A
Atheros Wireless / Netgear WNDA3200
BSNL Capitel
BandLuxe C120
BandRich BandLuxe C170, BandLuxe C270
Beceem BCSM250
C-motech CGU-628 (aka "Franklin Wireless CGU-628A" aka "4G Systems XS Stick W12")
C-motech CHU-629S
C-motech D-50 (aka "CDU-680")
Cricket A600
EpiValley SEC-7089 (featured by Alegro and Starcomms / iZAP)
Franklin Wireless U210
Hummer DTM5731
InfoCert Business Key (SmartCard/Reader emulation)
Kyocera W06K CDMA modem

LG HDM-2100 (EVDO Rev.A USB modem)
LG L-05A
LG LDU-1900D EV-DO (Rev. A)
LG LUU-2100TI (aka AT&T USBConnect Turbo)
Motorola 802.11 bg WLAN (TER/GUSB3-E)
MyWave SW006 Sport Phone/Modem Combination
Nokia CS-10
Nokia CS-15
Novatel MC990D
Novatel U727 USB modem
Novatel U760 USB modem
Novatel Wireless Ovation MC950D HSUPA
ONDA MT505UP (most likely a ZTE model)
Olivetti Olicard 100 and others
Olivetti Olicard 145
Option GlobeSurfer Icon 7.2
Option GlobeSurfer Icon 7.2, new firmware (HSO driver)
Option GlobeTrotter EXPRESS 7.2 (aka "T-Mobile wnw Express II")
Option GlobeTrotter GT MAX 3.6 (aka "T-Mobile Web'n'walk Card Compact II")
Option GlobeTrotter HSUPA Modem (aka "T-Mobile Web'n'walk Card Compact III')
Option iCON 210
Option iCON 225 HSDPA
Philips TalkTalk (NXP Semiconductors "Dragonfly")
Rogers Rocket Stick (a Sony Ericsson device)
Royaltek Q110 - UNCONFIRMED!
ST Mobile Connect HSUPA USB Modem
Sagem F@ST 9520-35-GLR
Samsung GT-B3730
Samsung SGH-Z810 USB (with microSD card)
Samsung U209
Sierra Wireless AirCard 881U (most likely 880U too)
Sierra Wireless Compass 597
Siptune LM-75 ("LinuxModem")
Solomon S3Gm-660
Sony Ericsson MD300
Sony Ericsson MD400
Toshiba G450
UTStarcom UM175 (distributor "Alltel")
UTStarcom UM185E (distributor "Alltel")
Vertex Wireless 100 Series

Vodafone (Huawei) K4605
Vodafone (ZTE) K3805-Z
Vodafone MD950 (Wisue Technology)
Zydas ZD1211RW WLAN USB, Sphairon HomeLink 1202 (Variant 1)
Zydas ZD1211RW WLAN USB, Sphairon HomeLink 1202 (Variant 2)

2.4.19 录音

在“友善之臂”程序组中，点击打开“录音机”程序，出现如下界面：

根据提示，点“REC”按钮开始录音，这时对着板上的麦克风说话，可以看到录音的波形，点“STOP”按钮结束录音，如图：

此时可以点“PLAY”按钮播放刚才的录音，同时录制的音频文件将以“WAV”格式自动存储到“文档”中：

说明：Qtopia 2.2.0 系统自带了一个录音程序，中文名为“语音便签”，但它不能正常使用板上的麦克风进行录制，为了保持代码的原始性，我们没有对此做任何修改。

2.4.20 使用 USB 摄像头拍照

问：我需要准备什么型号的 USB 摄像头？

答：你能买到的任何型号，最新的内核已经包含了 USB 摄像头万能驱动

把摄像头插到开发板的 USB Host 端口，然后在“友善之臂”程序组中，点击打开“USB 摄像头”程序，你将会看到动态的预览界面，调节一下摄像头的对焦，拍好姿势，点下“Snap”按钮就可以拍照了，拍摄的照片将会保存到“文档”组中。

本程序还可以调节亮度、对比度和伽马值；每个型号的摄像头在出厂时已经设定了最佳值或者缺省值，程序开启时会读取它们并以此设定。

注意：虽然内核中已经包含了万能驱动，但各个型号的摄像头输出格式却不尽相同，受条件所限，我们不可能收集到所有型号的摄像头，本程序基本可以支持市面上常见的各种型号摄像头，如果你的不支持，可以邮件和我们联系。

2.4.21 CMOS 摄像头预览拍照

本程序需要使用本公司提供的 CMOS 摄像头模块 CAM130，开机之前，把摄像头模块插到开发板的 CAMERA 接口上，在“友善之臂”程序组中找到“CMOS 摄像头”，点击打开它，如图：

点 Snap 按钮可以对当前的动态预览进行拍照，拍照后 Snap 按钮变为 Continue，点击它可以继续动态预览，同时照片会保存到“文档”组中(实际位于开发板的 /root/Documents/image/jpeg 目录中)，在“文档”点击刚刚拍摄的照片，会自动使用 Qtopia 的“图片”程序打开，如图。

2.4.22 LCD 测试

LCD 测试程序是为了检测 LCD 有无坏点而做，对于学习型开发板来讲，我们最大可

容许不超过 3 个坏点。

在“友善之臂”程序组中，点击打开“LCD 测试”，如图：

本程序有自动和手动两种测试模式：

Auto-loop 是自动循环模式，执行它将会按图中依次循环全屏显示红、黄、白、青、蓝、绿、粉红、黑共八种颜色，此过程中，点击屏幕任何地方就可以返回。

Manual-control 是手动模式，执行之后，每点击一次触摸屏，就会切换到另一种颜色，直至红、黄、白、青、蓝、绿、粉红、黑八色完成一个循环，然后返回。

2.4.23 背光控制

在“友善之臂”程序组中，点击打开“背光控制”程序界面，如图：

按照提示，在屏幕上蓝色区域部分，任意点击一下，即可关闭 LCD 背光。注意：这里仅仅是关闭了背光，系统各个程序仍在执行之中(如果对着灯光斜视 LCD，你会看到这个程序的深暗界面)。在蓝色区域再点击一下，背光又会被点亮了。

2.4.24 A/D 转换

Samsung S3C2440 芯片内部总共有 8 路 A/D 转换通道，但其转换器只有一个。在常见的设计中，一般 AIN4、AIN5、AIN6、AIN7 被用作了四线电阻触摸的 YM、YP、XM、XP 通道(见 S3C2440 芯片手册 16-5 章节)；本开发板引出了剩余的 AIN0-3，它们位于 CON4 接口，见本手册 1.3.章节，为了方便测试，其中 AIN0 直接和一个可调电阻 W1 连接。

它们如何共用一个转换器呢？请看如下操作：

在“友善之臂”程序组中，点击打开“A/D 转换”，如图：

这时旋转板上的 W1 可调电阻，可以看到不断变化的转换结果，因为是 10 位精度的转换器，故转换值最小时会接近 0，最大时会接近 1024。

当你点击触摸屏时，A/D 转换器将会选择触摸屏通道，这时转换结果会显示为“－1”，当触摸笔离开触摸屏，A/D 转换器又会选择板上的 AIN0 通道了。

说明：W1 可调电阻被 LCD 面板覆盖了，需要取下 LCD 面板才可以操作。

2.4.25 按键测试

说明：在本开发板系统中，按键并没有任何专用功能，它仅仅为测试底层驱动而用。

在“友善之臂”程序组中，点击打开“Buttons”程序，如图：

此时按下开发板上的任意按键(可以是多个)，相应的按键图标就会变为蓝色，松开后恢复为灰色，如图。

2.4.26 触摸笔测试

要测试校正后触摸笔的准确性，可以在 LCD 上任意地方画线，对比笔针所到之处画线是否有偏移，是否有抖动，这可以通过“随手写”程序来测试。

在“友善之臂”程序组中点“随手写”图标，打开如下界面：

“随手写”是由友善之臂开发的一个简易画图程序，打开后它出现一个浅黄色的画图区，用触摸笔可以在上面任意涂画(画笔颜色为黑色，宽度为 1 pixel)；点 File->Save 可以把所画保存为 png 格式的图片文件(保存位置：“Documents”，在板子中的位置：**/Documents/image/png/**)，文件以 001 起始，总共可以保存 999 个文件。

可以看到书写比较流畅，也没有毛刺出现，这说明触摸笔是很准确的。

2.4.27 条码扫描

本开发板支持 USB 接口的条码扫描器(英文名称：Barcode Scanner)，它其实是一个 HID 设备，类似 USB 键盘。因此，在任何适合 USB 键盘输入的地方，都可以直接使用这种条码扫描器。

需要注意的是：目前必须在图形界面启动之后插入 USB 扫描器才能正常使用，不能在开机之前就插入。

接上 USB 扫描器，打开“应用程序”中的“文本编辑器”，使用扫描器扫描一些条码(诸如快递单上、杂志等很多商品上都可以找到)，可以看到条码数字被准确扫描到编辑器中了。

2.4.28 语言设置

Qtopia 2.2.0 自带了“语言”设置，但已经和 Qtopia 1.7.0 有所不同，它只有英文支持，因此我们重新开发了一个“语言设置”，它位于“友善之臂”程序组，点击打开它。

目前仅有 3 种语言可以选择：英文、中文和日文。试试选择“英文”：点“English”可选项，再点“OK”，会跳出一个提示信息，询问你是否要改变语言，如果“Yes”，Qtopia 系统就会重新启动，如果“No”则返回，(注意：中文和日文版本只是翻译了程序名称)，如图：

2.4.29 设置时区-日期-时间-闹钟

本开发板在出厂的时候日期是不准确的,不过你可以自己调整一下,因为 CPU 内置了实时时钟(RTC),开发板上有备份电池,所以调整后的时间在关机后是可以”保存”的,当然这需要相应的驱动支持了，调整时间的步骤如下：

点击任务栏右下角时间显示区域，会出现一个菜单，在其中选择“Set time…”，打开时间设置界面，如图，在此界面你可以设置时区，调整日期和时间等。

在菜单中选择“Clock…”，出现钟表界面，点其中的“Stop Watch”打开“秒表”程序。

另外你还可以设置闹钟，如图，闹钟时间到，从音频输出可以听过“嘀嘀”声，它会持续一分钟，并出现如下界面，点“OK”可以结束闹钟声。

2.4.30 屏幕旋转

在“设置”程序组点“旋转”图标，进入相应的界面，如图，你可以选择 4 个界面旋

转方向，如图。

选择好你想要的方向，点“OK”返回，即可看到结果，如图。

说明：有时需要重新启动 Qtopia 系统才可以看到旋转的效果，因为这个程序是 Qtopia 自带的，为保持其代码原始性，我们没有去修改它。另外，旋转的效果完全是 Qtopia 软件实现的，与 LCD 的底层驱动无关。

2.4.31 设置开机自动运行程序

“开机自动运行”可以让你设定任何 Qtopia 自带的程序或者用户自己的 Qtopia 程序为开机后自动运行，它就像 Windows 系统中的“程序->启动”功能。

在“友善之臂”程序组中，点击“自动运行设置”图标打开它，如图：

图中左边一列是要设定的程序名，它包含了所有的 Qtopia 程序(包括用户 Qtopia 程序)，右边一列表明该程序的状态是否已经被设置为“开机自动运行”，这个状态是唯一的，所有程序只能有一个被标识。

在左侧列表中选择一个程序，如“串口助手”，这时它的状态被标识为“Auto Start”，点“Save”按钮，会跳出保存成功的提示，这时关闭该程序，再重新启动系统(复位或者点“设置->关机->Reboot)，就可以看到开机后串口助手已经自动打开了执行了。

另外，要取消“开机自动运行”程序，可以点“Clean”，然后点“Close”，这时会出现如图对话框，点“Yes”退出即可。

2.4.32 关于关机

在“设置”程序组中，有个“关机”图标，打开它，如图，其中有四个关机选项：Shutdown：按下此按钮，Linux 系统就会逐项关闭各个程序和服务，直到整个系统关

闭，这时 CPU 已经完全不工作了，此时整个系统的功耗是最低的。因为本开发板并无相应的硬件关机电路，因此你仍然可以看到板上的电源灯在亮。

Reboot：这是“热”重启按钮，如果你使用的是 NOR FLASH 模式，系统会逐项关闭各个应用和服务，重启后会停留在 Supervivi 的菜单模式；如果你使用的是 NAND FLASH 模式，系统会逐项关闭各个应用和服务，然后自动重启重新进入 Qtopia 系统。

注意：Reboot 和后面所介绍的“看门狗”是完全不同的功能，“看门狗”属于“冷”启动，它不会逐项关闭各个应用和服务，而是直接复位重启。

Restart Server 是指重新启动 Qtopia 图形系统，此时并不会影响到基本的 Linux 系统；

Terminates Server 是指关闭 Qtopia 图形系统，点击它之后图形界面就完全失效了，此时屏幕上的显示是遗留在内存中的数据，并不是有效的图形系统界面。

在“设置”程序组中，还有个“亮度和电源”的图标，因为本开发板并没有相应的电源管理电路，因此它们也是无效的。

说明：最原始的 Qtopia 2.2.0 源代码系统是无法有效执行“Shutdown”和“Reboot”的，我们对此进行了改进。

2.4.33 看门狗

看门狗是嵌入式系统中最常见的功能之一，S3C2440 芯片本身就带有看门狗，最新的内核中已经包含了它的驱动，现在我们就在应用程序中启动它。

点“友善之臂”程序组，点击打开“看门狗”，如图：

注意，先不要点“Start”按钮，请看红色区域的提示：

一旦启动了看门狗，它就无法停止了，只有不停的去喂它，否则系统就会复位重启，我们在此设定的倒数时间为 15 秒。

为了形象表示喂狗的动作，当喂狗时我们就丢给它一只骨头吃，如果一直点“Feed”按钮，它就一直有骨头吃，这样系统也不会复位重启，如图：

2.4.34 启动 QtE-4.6.3

为了方便用户使用，我们实现了在同一个文件系统中让 Qtopia-2.2.0 和 QtE-4.6.3 共存，在 Qtopia-2.2.0 系统中，通过点一个普通的应用程序图标就可以启动 QE-4.6.3；当关闭 QtE-4.6.3 程序时，又可以返回到 Qtopia-2.2.0，如图

QtE-4.6.3 系统运行的界面如下图，它是一个 CoverFlow 效果的简易程序管理器，你使用触摸笔可以左右拖动，并点击其中一个 Cover 来运行它

点“Exit Embedded Demo”可以退出 QtE-4.6.3，并返回 Qtopia-2.2.0

到此为止，我们就介绍完了大部分和硬件相关的基于图形界面的测试，系统里面还有其他一些小程序，那些都是系统自带的，如果你有兴趣可以自己尝试一下。

2.5 通过串口终端操作开发板

说明：每个期望学习 Linux 的嵌入式爱好者都应该学会熟练使用终端控制台的操作，所有平台的 Linux 指令都是相似的，超过 99%的命令是相同的；当你能够在命令行运指如飞的时候，你就已经深深爱上了 Linux。

进行本小节的操作之前，请先按照 2.1.3 章节正确设置好超级终端。

下图是通过串口终端显示的 Linux 登录界面，实际可能与此不完全相同，但基本都是类似的，根据提示，按下回车，就可以开始 Linux 控制台之旅了。

2.5.1 播放 mp3

测试程序名称： madplay		备注
源代码包的名称	madplay.tgz	
源代码包在光盘中的位置	Linux\madplay.tgz	
交叉编译器	Arm-linux-gcc-4.4.3 with EABI	
其他：		

madplay 是我们移植的一个基于控制台下的 mp3 播放器。它有多种播放控制模式，最简单的使用方法是：

#madplay your.mp3

该命令将以缺省模式播放 your.mp3 文件(开发板中并无 your.mp3 文件，这里只是举例说明)。

注意：系统启动后将会自动播放“/”目录下的 shanghaitan.mp3 文件。

可以运行“madplay -h”查看其使用帮助。

2.5.2 如何中止程序的运行

要中止程序的运行，可以在终端控制台下同时按下 **Ctrl+c**，注意：先按 **Ctrl**，不要放开，再按下 **c** 键即可。

例如：我们刚刚使用 madplay 命令播放了 mp3，如果要中止这个程序的运行，可以按

下 **Ctrl+c** 键。

另外，如果程序是在后台运行，可以使用 **kill** 命令杀掉该进程

2.5.3 使用优盘/移动硬盘

插入优盘之后，系统会自动创建一个/udisk 目录，并自动挂载优盘到上面，此时在串口会出现类似如下信息：

实际上优盘设备对应的设备名为**/dev/udisk**。进入/udisk 目录，就可以看到里面的文件了。注意：如果你的优盘无法识别，请检查一下它是不是 FAT32/VFAT 格式的

2.5.4 使用 SD 卡

和使用优盘一样，SD 卡也是自动识别挂载的，插入 SD 卡之后可以在串口看到如下信息：

系统会会自动创建/sdcard 目录，并把 SD 设备挂载到上面，如图：

2.5.5 如何通过串口与 PC 互相传送文件

当通过串口终端登录系统之后，可以使用 **rz** 或者 **sz** 命令通过串口与 PC 互相传送文件，具体操作如下。

(1)使用 sz 向 PC 发送文件

在超级终端窗口中，点鼠标右键，在弹出的菜单中选择“接收文件”开始设置接收文件目录和协议，如图所示。

然后在终端的命令行输入“**sz /shanghaitan.mp3**”命令，开始向 PC 传送位于“/”目录的 shanghaitan.mp3 文件(或者其他文件，改一下路径和文件名就可以了)，因为该文件比较大，所以需要多等几分钟，发送完毕，系统会自动保存文件到您设置的目录里面，如图。

(2)使用 rz 命令下载文件到开发板

在串口终端输入“rz”命令，开始接收从 PC 传过来的文件。

然后在超级终端窗口中，点鼠标右键，在弹出的菜单中选择“发送文件”，设置好要发送的文件和使用的协议，如图所示，开始向开发板发送文件。

点“发送”，开发板开始接收文件，如图所示。

接送完毕，将会在当前目录下得到同样文件名的文件，您可以使用 md5sum 命令验证该文件是否和源文件相同。

2.5.6 控制板上的 LED

测试程序名称： led-player leds		备注
源代码文件名	led-player.c led.c	
源代码在光盘中的位置	解压 linux\examples.tgz 可得	
交叉编译器	Arm-linux-gcc-4.4.3 with EABI	
开发板上对应的设备名	/dev/leds	
对应的内核驱动源代码	Linux-2.6.32.2/drivers/char/mini2440_leds.c	
其他：		

程序名称： leds.cgi		备注
源代码或者源代码包的名称	Leds.cgi	在开发板中
源代码或者源代码包的位置	位于开发板中的\www 目录中	
说明： leds.cgi 是一个 shell 脚本文件，它并不是二进制程序，该脚本通过 leds.html 被调用，其中使用的是最普通的网页设计技术。 解压光盘中的 root_default.tgz 也可以在其中的 www 目录得到 leds.cgi 和 leds.html 文件，它们都是脚本，本身就是源代码，使用任何文本编辑器(如 Windows 的“记事本”)都可以打开。		

说明：Led-player 和通过网页控制 LED 均为友善之臂早期为 SBC2410 开发的简易示例程序，因其硬件无关性，所以可以方便的移植到其他系统。目前市面上有的书籍，部分 2410/2440 开发板厂商均采用了这个典型的管道应用示例。

(1) LED 服务器

开机进入系统后，将会自动运行运行一个 LED 服务程序(**/etc/rc.d/init.d/leds**)，它其实是调用了 **led-player** 的一个脚本，led-player 开始运行后，将会在/tmp 目录下创建一个 **led-control** 管道文件，向该管道发送不同的参数可以改变 led 的闪烁模式：

#echo 0 0.2 > /tmp/led-control

运行该命令后，4 个用户 led 将会以每个间隔 0.2 秒的时间运行跑马灯。

#echo 1 0.2 >/tmp/led-control

运行该命令后，4 个用户 led 将会以间隔 0.2 秒的时间运行累加器。

#/etc/rc.d/init.d/leds stop

运行该命令后，4 个用户 led 将会停止闪动。

#/etc/rc.d/init.d/leds start

运行该命令后，4 个用户 led 将会重新开始闪动。

(2)单独控制 LED

/bin/leds 是一个可以控制单个 led 的实用程序，要使用 leds 必须先停止 led-player，如

下命令：

#/etc/rc.d/init.d/leds stop

该命令将停止 led-player 对 led 的操纵。led 的使用方法如下：

[root@fa /]# **led**

Usage: leds led_no 0|1

led_no 是要操作的 led(可为 0，1，2，3)，0 和 1 分别代表关闭和点亮。

#led 2 1

将点亮 LED3

2.5.7 测试板上的按键

测试程序名称： buttons		备注
测试程序源代码文件名	Buttons_test.c	
测试程序源代码位置	解压 linux\examples.tgz 可得	
交叉编译器	Arm-linux-gcc-4.4.3 with EABI	
开发板上对应的设备名	/dev/buttons	
对应的内核驱动源代码	Linux-2.6.32.2/drivers/char/mini2440_buttons.c	
其他：		

在命令行输入“**buttons**”命令，然后按开发板上的按键，可以显示对应的键值，如图

2.5.8 串口 2 和 3 的测试

测试程序名称： armcomtest		备注
测试程序源代码文件名	Comtest.c	
测试程序源代码位置	解压 linux\examples.tgz 可得	
交叉编译器	Arm-linux-gcc-4.4.3 with EABI	
开发板上对应的设备名	/dev/ttySAC0,1,2 或 /dev/ttyUSB0,1,2,3…	
对应的内核驱动源代码	Linux-2.6.32.2/drivers/serial/s3c2440.c	
其他：		

说明：armcomtest 是友善之臂为了方便测试而开发的 linux 下的简易实用串口终端程序，它使用标准的系统调用，和硬件无关，该程序可以在大部分 armv4 平台系统上运行使用，该程序不提供源代码。

提示：

系统启动后，串口 0,1,2 对应的设备名分别为**/dev/ttySAC0,1,2**

测试串口 2 需要借助另一台带有串口的 PC，使用我们提供的串口线和扩展小板(选购配件)，连接好 COM2 和另一台 PC 的串口，并如前所述设置该 PC 的超级终端为波特率 115200，无流控制，其他默认。

在命令行下输入：

#armcomtest –d /dev/ttySAC1 -o

这时如果输入字符会在另一台 PC 的超级终端出现，反之亦然。

如果要测试串口 3，则需要连接扩展小板的 COM3，并在命令行输入：

#armcomtest –d /dev/ttySAC2 -o

下面是测试时的界面：

2.5.9 测试蜂鸣器

测试程序名称： pwm_tset		备注
测试程序源代码文件名	Pwm_test.c	
测试程序源代码位置	解压 linux\examples.tgz 可得	
交叉编译器	Arm-linux-gcc-4.4.3 with EABI	
开发板上对应的设备名	/dev/pwm	
对应的内核驱动源代码	Linux-2.6.32.2/drivers/char/pwm.c	
其他：		

在命令行种输入：pwm_test
可以听到蜂鸣器的发出的声音，按“+”或者“-”可以改变输出的频率，如图。
按 ESC 键中止该测试。

2.5.10 控制 LCD 的背光

提示：
LCD 背光设备文件：/dev/backlight

在命令行种输入：echo 0 > /dev/backlight 可以关闭 LCD 背光。
在命令行种输入：echo 1 > /dev/backlight 可以打开 LCD 背光。

2.5.11 测试 I2C－EEPROM

测试程序名称： i2c		备注
测试程序源代码文件名	Eeprom.c 24cXX.c	
测试程序源代码位置	解压 linux\examples.tgz 可得	
交叉编译器	Arm-linux-gcc-4.4.3 with EABI	
开发板上对应的设备名	/dev/i2c/0	
对应的内核驱动源代码	Linux-2.6.32.2/drivers/i2c/busses/i2c-s3c2440.c	
其他：		

提示：
I2C-EEPROM 的设备文件名：/dev/i2c/0

在命令行种输入：i2c –w 可以向板子的 24C08 器件中写入数据（0x00-0xff）

在命令行中输入：i2c –r 可以从板子的 24C08 器件中读出输出

2.5.12 AD 转换测试

测试程序名称： adc-test		备注
测试程序源代码文件名	Adc-test.c	
测试程序源代码位置	解压 linux\examples.tgz 可得	
交叉编译器	Arm-linux-gcc-4.4.3 with EABI	
开发板上对应的设备名	/dev/adc	
对应的内核驱动源代码	Linux-2.6.32.2/drivers/char/mini2440_adc.c	
其他：		

在命令行输入 adc-test 命令，可以进行 ADC 转换测试，调节开发板板上的可调电阻 W1，可以看到从串口终端输出的转换结果。

2.5.13 CMOS 摄像头动态预览

测试程序名称： camtest		备注
测试程序源代码文件名	camtest.c	
测试程序源代码位置	解压 linux\examples.tgz 可得	
交叉编译器	Arm-linux-gcc-4.4.3 with EABI	
开发板上对应的设备名	/dev/camera	

对应的内核驱动源代码	Linux-2.6.32.2/drivers/media/video/s3c2440camif.c	
其他：		

把 CAM130 摄像头模块插到开发板的 CAMERA 接口上，打开电源进入命令行终端，在命令行输入 camtest 命令，可以看到 CMOS 摄像头的动态预览画面，如图

2.5.14 使用 telnet 上 bbs

telnet 是一个经常被使用的远程登录工具，使用 telnet 功能，可以从开发板登录到其他提供了 telnet 服务器的主机，如果您接入开发板的网络可以上互联网，则可以通过 telnet 命令登录外部的 bbs。

首先，确认开发板的 IP 地址是否为 192.168.1.230，并且是否和局域网内其他主机相通，如图为成功的信息。

然后设置路由 IP：**route add default gw 192.168.1.1**

最后使用 telnet 命令登录您要登录的主机，在此登录的是华南木棉 bbs。

2.5.15 如何设置网络以访问互联网网址

首先要确保你的网络环境可以正常登陆互联网，请记下你的网络环境所使用的网关 IP 地址，比如在我这里是 192.168.1.1，然后使用 route 进行设置：

 # route add default gw 192.168.1.1

这时你就可以直接访问互联网上的数字 IP 地址了，比如 ping 一下华南木棉的 BBS(其 IP 地址为 202.112.17.137)：

#ping 202.112.17.137

如图所示表示可以 ping 通外面的网络：

要能 ping 通外部网络的实名网址，还需要设置好域名解析服务器，先查看一下您当前网络所使用的 DNS 服务器 IP 地址(可以询问您的网络管理员)：

比如，我这里 DNS 服务器的 IP 为“202.96.128.86”，则在开发板中这样设置：

#rm /etc/resolv.conf ;首先删除以前的配置文件

#touch /etc/resolv.conf ；重新生成一个 resolv.conf 文件

#echo nameserver 202.96.128.86 >> /etc/resolv.conf ;使用实际的 DNS 服务器 IP 配置 resolv.conf 文件

可以这里主要是修改/etc/resolv.conf 文件，当然你也可以直接使用 vi 进行修改。

全部过程如下图所示：

2.5.16 如何设置 MAC 地址

开发板中所使用的 MAC 地址是“软”性的，因此你可以通过 ifconfig 命令对它进行重置，以适应于在同一个网络环境中使用多片开发板的情况，具体操作如下：

首先使用 ifconfig 查看一下当前的 mac 地址，运行：

#ifconfig ;注意后面不要跟任何内容

可以看到当前的 mac 地址为“08：90：90：90：90：90”，这是在网卡驱动中默认的 mac 地址，它已经被写死到内核中，除非更改网卡的驱动源代码并重新编译得到新内核。要在运行的系统中动态更改 mac 地址，先关闭当前网络，并使用 ifconfig 重置 mac 地址：

#ifconfig eth0 down

#ifconfig eth0 hw ether 00:11:AA:BB:CC:DD **;提示：a,b,c,d,e,f 可以为小写**

再开启网络，并使用 ifconfig 查看设置以后的 mac 地址，使用 ping 检验网络是否依然可通：

#ifconfig eth0 up

#ifconfig

#ping 192.168.1.1

2.5.17 如何使用 Telnet 远程登录开发板

开发板开机正常运行后，其实已经启动了一个 Telnet 服务，因此用户也可以通过网络远程登录开发板。

在 Windows 的命令行窗口输入“**telnet 192.168.1.230**”，如图出现登录界面，输入“root”(不需要密码)进入系统。

2.5.18 使用 ftp 传递文件

无论在 Linux 系统还是 windows 系统中，一般安装后都自带一个命令行的 ftp 命令程序，使用 ftp 可以登录远程的主机，并传递文件，这需要主机提供 ftp 服务和相应的权限；本开发板不仅带有 ftp 命令，还在开机时启动了 ftp 服务。为了方便测试，我们可以从 PC 机的命令行窗口登录开发板，并向开发板传递文件。

注意：请确保您执行 ftp 所在的目录有需要上传的文件，这里是 test.mp3

说明：登录开发板的 ftp 帐号为：plg 密码为：plg

传送完毕，您可以在串口终端看到目标板的/home/plg 目录下多了一个 test.mp3 文件。

2.5.19 通过网页控制板上的 LED

在 web server 测试页面中点“网络控制 LED 测试”项，会出现 LED 测试控制页面，如图

您可以使用网页中的各个测试项目进行测试，其中的“LED 测试”将会通过 CGI 程序来控制板上的 LED 灯，其中包括 2 种方式的显示类型和三种不同的显示速度。

如果要停止 web 服务器，则在命令提示符下输入以下命令：

#/etc/rc.d/init.d/httpd stop

要重新启动则输入：

#/etc/rc.d/init.d/httpd start

2.5.20 如何挂接使用网络文件系统 NFS

在进行该测试之前，请先按照 4.3 一节搭建好 NFS 服务器系统，然后在命令行输入以下命令（假定服务器的 IP 地址为 192.168.1.111）：

#mount –t nfs –o nolock 192.168.1.111:/opt/FriendlyARM/mini2440/root_qtopia /mnt

挂接成功，您就可以进入/mnt 目录进行操作了，如下图所示。

取消挂接的命令如下：

#umount /mnt

2.5.21 使用 USB 无线网卡

Linux-2.6.32.2 内核已经自带了大部分 USB 无线网卡的驱动，其中包括目前最流行、性价比最高的 TP-Link USB WiFi 无线网卡：TL-WN321G+，本开发板的缺省内核映象已经加入了本模块的支持。

提示：本手册的 6.3 章节有关于在内核中配置无线网卡驱动的说明

把 USB 无线网卡插入开发板，会出现如下信息：

[root@FriendlyARM /]# usb 1-1: new full speed USB device using s3c2410-ohci and address 2
usb 1-1: New USB device found, idVendor=148f, idProduct=2573
usb 1-1: New USB device strings: Mfr=1, Product=2, SerialNumber=0
usb 1-1: Product: 54M.USB.....
usb 1-1: Manufacturer: Ralink
usb 1-1: configuration #1 chosen from 1 choice
wmaster0 (rt73usb): not using net_device_ops yet
wlan0 (rt73usb): not using net_device_ops yet

[root@FriendlyARM /]#

这说明已经识别到该无线网卡，然后通过步骤命令开始配置。

(1)首先关闭开发板的有线网卡 DM9000

[root@FriendlyARM /]# ifconfig eth0 down

(2)加载 USB WiFi 无线网卡

[root@FriendlyARM /]# ifconfig wlan0 up
rt73usb 1-1:1.0: firmware: requesting rt73.bin
[root@FriendlyARM /]#

(3)扫描可用的无线网络

[root@FriendlyARM /]# iwlist scanning | grep ESSID
lo　　Interface doesn't support scanning.

eth0　　Interface doesn't support scanning.

wmaster0　Interface doesn't support scanning.

　　ESSID:"FRIENDLY-ARM"
　　ESSID:"NETGEAR"
　　ESSID:"TP-LINK"
[root@FriendlyARM /]#

(4)选择要连接的无线网络

[root@FriendlyARM /]# iwconfig wlan0 essid "FRIENDLY-ARM"

(5)输入该网络的安全密码

[root@FriendlyARM /]# iwconfig wlan0 key s:12345

(6)连接到指定的 AP(无线路由)

[root@FriendlyARM /]# iwconfig wlan0 ap auto

(7)设置无线网卡的 IP 地址

[root@FriendlyARM /]# ifconfig wlan0 192.168.1.120

(8)使用 ping 命令检测无线网连通状况

[root@FriendlyARM /]# ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1): 56 data bytes
64 bytes from 192.168.1.1: seq=0 ttl=64 time=42.804 ms
64 bytes from 192.168.1.1: seq=1 ttl=64 time=5.020 ms
64 bytes from 192.168.1.1: seq=2 ttl=64 time=5.021 ms

以上整个过程如下所示：

[root@FriendlyARM /]# usb 1-1: new full speed USB device using s3c2410-ohci and address 2
usb 1-1: New USB device found, idVendor=148f, idProduct=2573
usb 1-1: New USB device strings: Mfr=1, Product=2, SerialNumber=0
usb 1-1: Product: 54M.USB.....
usb 1-1: Manufacturer: Ralink
usb 1-1: configuration #1 chosen from 1 choice

```
wmaster0 (rt73usb): not using net_device_ops yet
wlan0 (rt73usb): not using net_device_ops yet

[root@FriendlyARM /]# ifconfig eth0 down
[root@FriendlyARM /]# ifconfig wlan0 up
rt73usb 1-1:1.0: firmware: requesting rt73.bin
[root@FriendlyARM /]# iwconfig wlan0 key s:12345
[root@FriendlyARM /]# iwconfig wlan0 essid "FRIENDLY-ARM"
[root@FriendlyARM /]# iwconfig wlan0 ap auto
[root@FriendlyARM /]# ifconfig wlan0 192.168.1.120
[root@FriendlyARM /]# ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1): 56 data bytes
64 bytes from 192.168.1.1: seq=0 ttl=64 time=42.804 ms
64 bytes from 192.168.1.1: seq=1 ttl=64 time=5.020 ms
64 bytes from 192.168.1.1: seq=2 ttl=64 time=5.021 ms
^C
--- 192.168.1.1 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 5.020/17.615/42.804 ms
[root@FriendlyARM /]#
```

2.5.22 设置并保存系统实时时钟

Linux 中更改时间的方法一般使用 **date** 命令，为了把 S3C2440 内部带的时钟与 linux 系统时钟同步，一般使用 **hwclock** 命令，下面是它们的使用方法：

(1) **date -s 042916352007** #设置时间为 2007-04-29 16:34

(2) **hwclock -w** #把刚刚设置的时间存入 S3C2440 内部的 RTC

(3).开机时使用 **hwclock -s** 命令可以恢复 linux 系统时钟为 RTC，一般把该语句放入 **/etc/init.d/rcS** 文件自动执行。

注意：我们提供的系统已经把 hwclock –s 命令写入 rcS 文件。

2.5.23 如何掉电保存数据到 Flash

由于本系统采用了可读写文件系统 yaffs(在嵌入式系统中，专门管理 Flash 存储器的一种文件系统)，因此可以很方便的动态保存数据，掉电后不会丢失。开机后在串口终端运行以下命令：

#cp / shanghaitan.mp3 /home/plg

此时将在/home/fa 目录下复制一个同样的文件，然后关机，重新开启系统，可以查看到/home/plg 目录下的文件依然存在。

2.5.24 如何设置开机自动运行程序

借助启动脚本可以设置各种程序开机后自动运行，也可以设置其他系统设置，这有点类似于 Windows 系统中的 Autobat 自动批处理文件，启动脚本的位于板子的/etc/init.d/rcS，内容如下(实际内容可能与此不完全一致)：

```
#! /bin/sh

PATH=/sbin:/bin:/usr/sbin:/usr/bin:/usr/local/bin:
runlevel=S
prevlevel=N
umask 022
export PATH runlevel prevlevel

#
#     Trap CTRL-C &c only in this shell so we can interrupt subprocesses.
#
trap ":" INT QUIT TSTP
/bin/hostname FriendlyARM

/bin/mount -n -t proc none /proc
/bin/mount -n -t sysfs none /sys
/bin/mount -n -t usbfs none /proc/bus/usb
/bin/mount -t ramfs none /dev

echo /sbin/mdev > /proc/sys/kernel/hotplug
/sbin/mdev -s
/bin/hotplug
# mounting file system specified in /etc/fstab
mkdir -p /dev/pts
mkdir -p /dev/shm
/bin/mount -n -t devpts none /dev/pts -o mode=0622
/bin/mount -n -t tmpfs tmpfs /dev/shm
/bin/mount -n -t ramfs none /tmp
/bin/mount -n -t ramfs none /var
mkdir -p /var/empty
mkdir -p /var/log
mkdir -p /var/lock
mkdir -p /var/run
mkdir -p /var/tmp
```

```
/sbin/hwclock -s

syslogd
/etc/rc.d/init.d/netd start
echo "                                    " > /dev/tty1
echo "Starting networking..." > /dev/tty1
sleep 1
/etc/rc.d/init.d/httpd start
echo "                                    " > /dev/tty1
echo "Starting web server..." > /dev/tty1
sleep 1
/etc/rc.d/init.d/leds start
echo "                                    " > /dev/tty1
echo "Starting leds service..." > /dev/tty1
echo "                                    "
sleep 1

/sbin/ifconfig lo 127.0.0.1
```

/etc/init.d/ifconfig-eth0　；设置开机静态 IP 地址，这是一个脚本，你可以使用 vi 打开并编辑

```
#/bin/qtopia &
echo "                                                  " > /dev/tty1
echo "Starting Qtopia, please waiting..." > /dev/tty1
```

2.5.25 如何使用命令进行屏幕截图

使用 snapshot 命令可以对当前的 LCD 显示进行截图，并保存为 png 格式的图片。

#snapshot pic.png

执行该命令将把当前 LCD 显示进行抓图，并保存为 pic.png 文件。

2.6 预装 WindowsCE5/6 系统的使用和设置

WindowsCE6 的烧写文件位于光盘“\images\wince6.0”目录中。

请按照“安装和更新系统程序”一章下载烧写您所需要的 wince 映象文件(此示例中烧写的是 NK_T35.bin，如果你采用的是横向的 W35 LCD，烧写文件是 NK_W35.bin，开机画面可能仅显示一部分，请自行选择合适的图片即可)。安装完毕，请把开发板的 S2 开关设置为 Nand Flash 启动系统。

如果你在安装时选择的 bootloader 是 Nboot，开机启动系统你将会看到一个开机图片，然后会有进度条指示系统正在装载中，装载完毕即可出现 WindowsCE 的启动画面，然后进入桌面系统，如图：

开机 Logo 和启动进度条

CE 启动画面

进入系统

2.6.1 按键测试

测试程序名称： buttons		备注
驱动程序源代码在 BSP 中的位置	mini2440\Src\Drivers\Userkey	
开发编译工具	Visual Studio 2005 with Platform Builder 6	
开发板上对应的设备名	/dev/buttons	
其他：		

说明：在本开发板系统中，按键并没有任何专用功能，它仅仅为测试底层驱动而用。在“友善之臂”程序组中，点击打开“Buttons”程序，如图：

此时按下开发板上的任意按键(可以是多个)，相应的按键图标就会变为蓝色，松开后恢复为灰色，如图。

2.6.2 LED 测试

测试程序名称： LED-Test.exe		备注
驱动程序源代码在 BSP 中的位置	Mini2440\SRC\DRIVERS\LEDdriver	
开发编译工具	Visual Studio 2005 with Platform Builder 6	
开发板上对应的设备名		
其他：		

在“友善之臂”程序中点“LED 测试”图标，打开如下界面：

此时，点击程序中的按钮，可以任意控制开发板上的四个 LED 的亮和灭。

2.6.3 ADC 转换

测试程序名称： AD-Convert		备注
驱动程序源代码在 BSP 中的位置	Mini2440\SRC\DRIVERS\Touch	
开发编译工具	Visual Studio 2005 with Platform Builder 6	
开发板上对应的设备名		
其他：ADC 驱动和触摸屏驱动共享一个源代码		

Samsung S3C2440 芯片内部总共有 8 路 A/D 转换通道，但其转换器只有一个。在常见的设计中，一般 AIN4、AIN5、AIN6、AIN7 被用作了四线电阻触摸的 YM、YP、XM、XP 通道(见 S3C2440 芯片手册 16-5 章节)；本开发板引出了剩余的 AIN0-3，它们位于 CON4 接口，见本手册 1.3.章节，为了方便测试，其中 AIN0 直接和一个可调电阻 W1 连接。

它们如何共用一个转换器呢？请看如下操作：

在“友善之臂”程序组中，点击打开“A/D 转换”，如图：

这时旋转板上的 W1 可调电阻，可以看到不断变化的转换结果，因为是 10 位精度的转换器，故转换值最小时会接近 0，最大时会接近 1024。

当你点击触摸屏时，A/D 转换器将会选择触摸屏通道，这时转换结果会显示为“－1”，当触摸笔离开触摸屏，A/D 转换器又会选择板上的 AIN0 通道了。

说明：W1 可调电阻被 LCD 面板覆盖了，需要取下 LCD 面板才可以操作。

2.6.4 I2C-EEPROM 读写

测试程序名称：　I2C-EEPROM.exe		备注
驱动程序源代码在 BSP 中的位置	\Mini2440\SRC\DRIVERS\IIC	
开发编译工具	Visual Studio 2005 with Platform Builder 6	
开发板上对应的设备名	/dev/buttons	
其他：		

在“友善之臂”程序组中点“I2C-EEPROM 测试”图标，打开如下界面：

由上到下依次可以看到：写 EEPROM 按钮、写入字符编辑区、读取 EEPROM 按钮、读出字符编辑区。

点开任务栏上的“软键盘”按钮，在“写入字符编辑区”输入一些 ASC 字符，点“Write”按钮，这时该按钮变为进度条，并指示正在写入的进度，；点“Read”按钮，这时按钮变为进度条，并指示正在读取的进度。如图。

2.6.5 PWM 控制蜂鸣器

测试程序名称：　PWM-Buzzer.exe		备注
驱动程序源代码在 BSP 中的位置	Mini2440\SRC\DRIVERS\PWM	
开发编译工具	Visual Studio 2005 with Platform Builder 6	

开发板上对应的设备名		
其他：		

在“友善之臂”程序组中点“PWM-buzzer”图标，打开如下界面：

程序中默认的 PWM 输出为 1000Hz，点“Start”按钮开始驱动蜂鸣器发声，此时可以通过点击列表框的“上”或者“下”按钮改变 PWM 输出的频率，同时也可以听到蜂鸣器输出声音的改变。点“Stop”按钮中止 PWM 输出。

2.6.6 看门狗

测试程序名称：　Watchdog.exe		备注
驱动程序源代码在 BSP 中的位置	Mini2440\SRC\DRIVERS\WDT	
开发编译工具	Visual Studio 2005 with Platform Builder 6	

开发板上对应的设备名		
其他：		

看门狗是嵌入式系统中最常见的功能之一，S3C2440 芯片本身就带有看门狗，我们提供的最新 WindwsCE6 BSP 中已经包含了它的驱动，现在我们就在应用程序中启动它。

点“友善之臂”程序组，点击打开“看门狗”，如图：

注意，先不要点“Start”按钮，请看红色区域的提示：

一旦启动了看门狗，它就无法停止了，只有不停的去喂它，否则系统就会复位重启，我们在此设定的倒数时间为 15 秒。

为了形象表示喂狗的动作，当喂狗时我们就丢给它一只骨头吃，如果一直点“Feed”按钮，它就一直有骨头吃，这样系统也不会复位重启，如图：

2.6.7 LCD 测试

测试程序名称：　LCD-Test.exe		备注
驱动程序源代码在 BSP 中的位置	Mini2440\SRC\DRIVERS\Display	
开发编译工具	Visual Studio 2005 with Platform Builder 6	
开发板上对应的设备名		
其他：		

LCD 测试程序是为了检测 LCD 有无坏点而做，对于学习型开发板来讲，我们最大可容许不超过 3 个坏点。

在“友善之臂”程序组中，点击打开“LCD 测试”，如图：

本程序有自动和手动两种测试模式：

Auto-loop 是自动循环模式，执行它将会按图中依次循环全屏显示红、黄、白、青、蓝、绿、粉红、黑共八种颜色，此过程中，点击屏幕任何地方就可以返回。

Manual-control 是手动模式，执行之后，每点击一次触摸屏，就会切换到另一种颜色，直至红、黄、白、青、蓝、绿、粉红、黑八色完成一个循环，然后返回。

2.6.8 CMOS 摄像头预览拍照

测试程序名称：　CMOS-Camera.exe		备注
驱动程序源代码在 BSP 中的位置	Mini2440\SRC\DRIVERS\Camera	
开发编译工具	Visual Studio 2005 with Platform Builder 6	
开发板上对应的设备名		
其他：		

本程序需要使用本公司提供的 CMOS 摄像头模块 CAM130，开机之前，把摄像头模块插到开发板的 CAMERA 接口上，在“友善之臂”程序组中找到“CMOS 摄像头”，点击打开它，如图：

当点击运行 CMOS-Camera 程序时，它将开始初始化 CMOS 摄像头设备，此时界面如左下图，如果 CMOS 摄像头已经接上，则稍等片刻，就会看到动态的影像了，如图：

点 Snap 按钮可以对当前的动态预览进行拍照，拍照后 Snap 按钮变为 Continue，点击它可以继续动态预览，同时照片会保存到“文档”组中(实际位于开发板的 My Documents 目录中，格式为 bmp)，因为本开发板并没有预装图片浏览器，你可以把该图片文件通过 SD 卡或者优盘，或者同步的方式转移到 PC 上浏览。

2.6.9 录音测试

测试程序名称： Recorder.exe		备注
驱动程序源代码在 BSP 中的位置	Mini2440\SRC\DRIVERS\Wavedev	
开发编译工具	Visual Studio 2005 with Platform Builder 6	
开发板上对应的设备名		
其他：		

在“友善之臂”程序组中，点击打开“Recorder”程序，出现如下界面：

根据提示，点“录音”按钮开始录音，这时对着板上的麦克风说话，程序开始录音，点“停止”按钮结束录音，如图：

此时可以点“播放”按钮会循环播放刚才的录音，

说明：该录音程序并不保存录音结果。

2.6.10 屏幕旋转并保存

说明：屏幕旋转的驱动程序已经包换在 LCD 驱动中，它不需要特别的硬件支持，是纯软件实现的，因此无需另外的单独驱动。

LCD 驱动的位置：mini2440\Src\Drivers\Display

系统启动后，在“友善之臂”程序组点击运行“Rotate”，运行界面如左下图，

点击其中的“Rotate”按钮，屏幕会按逆时针方向 90 度，如右下图。

此时点 Close 按钮，该旋转结果会保存在注册表中，下次开机运行时将会以“横屏”的方式出现，如右上图

2.6.11 串口助手

测试程序名称： SerialPort.exe		备注
驱动程序源代码在 BSP 中的位置	Mini2440\SRC\DRIVERS\Serial	
开发编译工具	Visual Studio 2005 with Platform Builder 6	
开发板上对应的设备名		
其他：		

说明：本开发板提供的 BSP 包含三个串口的标准驱动，要测试串口 2,3，需要使用串口扩展小板。

在“友善之臂”程序组点击运行“SerialPort”，运行界面如左下图，

点“设置”按钮，打开设置窗口，设置串口号为 COM2，波特率为 115200，其他设置如图(右上)，点确定返回主窗口。

同时，连接好扩展板的 COM2 到 PC 一端，并在 PC 端把相应的串口作相同的设置。

在主窗口中点“打开端口”按钮，此时该按钮会变为“关闭端口”，在“发送区”输入一些字符，点“发送”按钮，如左下图，这时会在 PC 端的串口终端接收到从开发板发送来的字符，如右下图：

然后，在串口调试助手的主窗口点“接收”按钮（该按钮会改变为“不接收”），在 PC 端的串口终端输入一些字符(通过超级终端是无法看到的)，在输入的同时，我们看到输入的字符会在开发板串口调试助手的接收区显示，如图：

我们还可以使用同样的方法测试 COM3，在此就不作详细的说明了。

2.6.12 触摸屏校正

触摸屏驱动程序源代码位于 BSP 的如下目录：
Mini2440\SRC\DRIVERS\Touch

缺省安装的 wince 系统的触摸屏校正参数一般适用于统宝 3.5”LCD，但因为每个触摸屏的物理特性不同，有时可能不太准确，特别是不同尺寸的时候，这时就需要重新校正，如下步骤。

请先接上 USB 鼠标，点开始->设置->控制面板，找到“笔针”图标，双击打开“笔针属性”窗口，点“校准”选项卡的“再校准”按钮，开始重新校正。

根据系统提示，使用五点校正法用触摸笔开始校正，校正完毕，将会跳出如下窗口，这时随便点一个位置即可返回“笔针属性”窗口，点“OK”保存退出。

如果您想保存本次校准的参数，请点“开始->挂起”，然后再重新开机就可以了。

2.6.13 设置网络参数以连接互联网

测试程序名称：　系统自带的网络设置		备注
驱动程序源代码在 BSP 中的位置	Mini2440\SRC\DRIVERS\dm9000	
开发编译工具	Visual Studio 2005 with Platform Builder 6	
开发板上对应的设备名		
其他：		

只有正确设置了 IP 地址和网关以及 DNS 等参数，才可以使用开发板连接使用互联网或者局域网。

设置网络参数的步骤和标准的 Windows 系统十分相似，点“开始”->“控制面板”，找到网络设置选项，并找到相应的网卡 DM9CE1 如图。

双击打开 DM9CE1 图标，如图，在这里，你可以设置静态的 IP 地址，也可以设置动态获取 IP 的方式，按照你所在的网络环境实际参数填写即可，如图为缺省的设置参数。

确定网络设置参数无误，就可以打开浏览器上互联网了。

2.6.14 背光设置

测试程序名称： 系统自带的背光设置		备注
驱动程序源代码在 BSP 中的位置	Mini2440\SRC\DRIVERS\Backlight	
开发编译工具	Visual Studio 2005 with Platform Builder 6	
开发板上对应的设备名		
其他：		

我们提供的最新 WindowsCE6 已经支持标准的系统背光开关设置，在此，你可以设置

背光常亮，或者延时关闭，通过触摸屏、鼠标、板上的按键等都可以唤醒背光，实际上，此处实现的背光设置已经属于电源管理的一部分，只不过本开发板仅有外部电源支援。

要进入背光设置界面，有 2 种方式：

可以在空白的桌面处通过点击鼠标右键，再点属性

也可以点“开始”->“控制面板”->“显示”

如图为背光设置界面的缺省设置

可见，使用外部电源供电时，如果在 1 分钟之内触摸屏或者按键等 HMI 设备没有动作，背光将会自动熄灭，你也可以把时间改为其他数值，设置完毕，只需耐心等待即可看到结果了。

为了让背光常亮，可以把上面界面中“使用外部电源并且设备空闲”取消就可以了，如下图

2.6.15 设置实时时钟并保存

测试程序名称：　系统自带的时间设置		备注
驱动程序源代码在 BSP 中的位置	Mini2440\SRC\COMMON\Rtc	
开发编译工具	Visual Studio 2005 with Platform Builder 6	
开发板上对应的设备名		
其他：　注意此设备驱动和其他设备不太相同，它并不在常规的“设备驱动”目录中		

点任务栏右下角的时间，出现时间设置窗口，根据提示进行设置就可以了，设置完毕，点“OK”退出，设置时间不需要点“挂起”保存。

2.6.16 设置程序开机自动运行

有很多用户希望在开机时就可以运行自己的程序，有很多办法可以实现，通过 google 搜索一下关键词“WinCE 开机自动运行”你就会得到很多结果，我们选择通过修改注册表的方式实现了这一功能，并且只需简单的设定就可以，无需手工修改注册表。当然，这是一个我们专门制作的小程序，它的使用方法如下。

在“友善之臂”程序组中点“AutoRun-Setting”并运行它，如图

可以看到，缺省的系统中没有任何 Apps 被设定，点“Browser”找到一个需要开机自动运行的程序，在此我们以“AD-Convert“为例，如图

说明：AD-Convert.exe 程序位于\Windows 目录下，其他位置的同名文件均为快捷方式。

点“OK”返回设置界面，再点“Close”关闭该设置程序。

好了，现在直接关闭电源或者按复位键，稍等片刻，你就可以看到 AD-Convert 程序已经自动运行了。

2.6.17 使用优盘

在 wince 中使用优盘和使用标准的 windows 使用优盘类似，当 WINCE 系统启动以后，把优盘插入 USB Host 接口，这时板子给优盘供电，优盘的指示灯会闪烁，等待几秒系统就自动加载优盘了，这时可以双击桌面的“我的电脑”图标，打开资源管理器，可以看到优盘的盘符：硬盘，双击硬盘即可进入优盘进行数据读写了。如下图所示。

2.6.18 使用 SD/MMC 卡

说明：因为采用了最新架构的 BSP，目前开发板最大可以支持 32G 普通和高速 SD 卡。

把 SD/MMC 卡插入板上的 SD 插槽，资源管理器中就可以看到 SD 卡的盘符：Storage Card，双击打开进入该目录，就可以对 SD/MMC 卡进行读写了。

2.6.19 使用 ActiveSync 进行 USB 同步通讯

注意：安装同步连接的 USB 驱动时，请使用光盘中的“\windows 平台工具\CE 用同步 USB 驱动”

我们假定您已经按照 9.3 节安装好驱动和 ActiveSync 同步软件。

在默认情况下，当 WINCE 系统启动以后，USB Device 连接和设置就已经做好了，这时可以直接用 USB 连接开发板和 PC 机，即可看到如下成功连接的情况：

2.6.20 关于 USB 无线网卡

目前我们尚未发现驱动支持 WinCE6 系统的 USB 无线网卡，欲使用 USB 无线网卡的用户可以安装光盘为 2009-8-23 日期版本的 WinCE5 系统。

2.7 安装使用第三方软件

预装的 WinCE6 系统可以支持运行很多第三方软件，我们在互联网上淘到一些常用的并集合放在光盘中，仅供用户测试验证使用，有些可能涉及到版权，如果您把这些软件直接用于终端产品，请和软件的原作者或厂家联系。

下面这些软件均可直接复制安装到开发板中的任何文件夹中运行，并且掉电后不会丢失，为了方便管理，我们建议用户自行建立目录归类。因仅为测试验证，我们将不对这些软件作详细的使用说明，请您自行把玩其中的功能细节。

第三方软件在光盘中的位置：**mini2440-20100110\WindowsCE 第三方软件**

把“WindowsCE 第三方软件”整个目录复制到一张 SD 卡或者优盘中，或者直接通过同步中心复制到开发板中，就可以进行下面的步骤了。本示例中是把所有软件复制到开发板的\Software 目录中进行的。

2.7.1 输入法

2.7.1.1 蒙恬手写输入

双击运行“蒙恬输入法.cab”安装输入法，一切按照缺省设置安装即可，安装完毕，“蒙恬输入法.cab”会自动被删除。

安装完毕，打开桌面的“Microsoft WordPad”程序，此时点任务栏右边的键盘图标，可以出现“蒙恬输入法”选项，点击选择，即可开始进行手写输入了，如图

2.7.2 实用工具

2.7.2.1 小画笔

在\software\实用工具\小画笔\目录中，双击运行“画笔”程序，如图

2.7.2.2 计算器

在\software\实用工具\小画笔\目录中，双击运行“画笔”程序，如图

2.7.2.3 记事本

在\software\实用工具\小画笔\目录中，双击运行“画笔”程序，如图

2.7.2.4 截图工具

在\software\实用工具\小画笔\目录中，双击运行“画笔”程序，如图

2.7.3 办公软件

2.7.3.1 文字处理浏览器

在\software\办公软件\OFFIC 目录中，双击运行“docviewer.exe”程序，并打开一个 word 文件，如图

2.7.3.2 电子表格浏览器

在\software\办公软件\OFFIEC 目录中，双击运行“XLS.exe”程序，并打开一个 Excel 文件，如图

2.7.3.3 幻灯片浏览

在\software\办公软件\OFFIEC 目录中，双击运行“presviewer.exe”程序，并打开一个 ppt 文件，如图

2.7.3.5 图片浏览器

在\software\办公软件\OFFIEC 目录中，双击运行“imageviewer.exe”程序，程序会自动打开 My Documents 目录中的图片并预览，如图

2.7.3.5 pdf 阅读器

在\software\办公软件\OFFIEC 目录中，双击运行“pdfviewer.exe”程序，并打开一个 pdf 文件，如图

2.7.4 媒体播放

2.7.4.1 TCPMP

在\software\媒体播放\TCPMP 目录中，双击运行“PLAYER”程序，并打开一个视频文件进行播放，如图

2.7.4.2 CorePlayer

在\software\媒体播放\CorePlayer 目录中，双击运行“PLAYER”程序，并打开一个视频文件进行播放，如图

2.7.4.3 Flash 播放

在\software\媒体播放\Flash 播放器目录中，双击运行“swFlash32.exe”程序，并打开一个视频文件进行播放，如图

2.7.5 图形图像

2.7.5.1 Photoshop

在\Software\图形图像\Photoshop 目录中，双击运行“PocketArtist.exe”程序，并打开一个图片进行编辑，如图

2.7.6 网络软件

2.7.6.1 UCWEB 浏览器

连接好网线，并设置好正确的网络参数，在\Software\网络软件\UCWEB6.7 目录中，双击运行“UCWEB.EXE”程序，并打开一个网站进行浏览，如图

2.7.7 休闲娱乐

2.7.7.1 BUBBLETS

在\Software\休闲游戏\BUBBLETS 目录中，双击运行“BUBBLETS.EXE”程序， 即可打开该游戏，如图

2.7.7.2 中国象棋

在\Software\休闲游戏\中国象棋目录中，双击运行“中国象棋.exe”程序， 即可打开该游戏，如图

2.7.7.3 游戏套装(33 个)

我们还搜集了其他 33 个小游戏，它们大部分都可以在开发板上运行，在此就不一一介绍了，感兴趣的用户可以自行试用。

2.8 体验 WindowsCE 5.0

WindowsCE5 的烧写文件位于光盘“\images\wince5.0”目录中。

请按照“安装和更新系统程序”一章下载烧写您所需要的 wince 映象文件(此示例中烧写的是 NK_T35.bin，如果你采用的是横向的 W35 LCD，烧写文件是 NK_W35.bin，开机画面可能仅显示一部分，请自行选择合适的图片即可)。安装完毕，请把开发板的 S2 开关设置为 Nand Flash 启动系统。

如果你在安装时选择的 bootloader 是 Nboot，开机启动系统你将会看到一个开机图片，然后会有进度条指示系统正在装载中，装载完毕即可出现 WindowsCE 的启动画面，然后进入桌面系统，如图：

开机 Logo 和启动进度条

CE 启动画面

进入系统

2.8.1 按键测试

测试程序名称： buttons		备注
驱动程序源代码在 BSP 中的位置	mini2440\Src\Drivers\Userkey	
开发编译工具	Platform Builder 5 with 2007 Patch	
开发板上对应的设备名	/dev/buttons	
其他：		

说明：在本开发板系统中，按键并没有任何专用功能，它仅仅为测试底层驱动而用。在“友善之臂”程序组中，点击打开“Buttons”程序，如图：

此时按下开发板上的任意按键(可以是多个)，相应的按键图标就会变为蓝色，松开后恢复为灰色，如图。

2.6.2 LED 测试

测试程序名称：　LED-Test.exe		备注
驱动程序源代码在 BSP 中的位置	Mini2440\SRC\DRIVERS\LEDdriver	
开发编译工具	Platform Builder 5 with 2007 Patch	
开发板上对应的设备名		
其他：		

在“友善之臂”程序中点“LED 测试”图标，打开如下界面：

此时，点击程序中的按钮，可以任意控制开发板上的四个 LED 的亮和灭。

2.8.3 ADC 转换

测试程序名称：　AD-Convert		备注
驱动程序源代码在 BSP 中的位置	Mini2440\SRC\DRIVERS\Touch	
开发编译工具	Platform Builder 5 with 2007 Patch	
开发板上对应的设备名		
其他：ADC 驱动和触摸屏驱动共享一个源代码		

Samsung S3C2440 芯片内部总共有 8 路 A/D 转换通道，但其转换器只有一个。在常见的设计中，一般 AIN4、AIN5、AIN6、AIN7 被用作了四线电阻触摸的 YM、YP、XM、XP 通道(见 S3C2440 芯片手册 16-5 章节)；本开发板引出了剩余的 AIN0-3，它们位于 CON4 接口，见本手册 1.3.章节，为了方便测试，其中 AIN0 直接和一个可调电阻 W1 连接。

它们如何共用一个转换器呢？请看如下操作：

在“友善之臂”程序组中，点击打开“A/D 转换”，如图：

这时旋转板上的 W1 可调电阻，可以看到不断变化的转换结果，因为是 10 位精度的转换器，故转换值最小时会接近 0，最大时会接近 1024。

当你点击触摸屏时，A/D 转换器将会选择触摸屏通道，这时转换结果会显示为“－1”，当触摸笔离开触摸屏，A/D 转换器又会选择板上的 AIN0 通道了。

说明：W1 可调电阻被 LCD 面板覆盖了，需要取下 LCD 面板才可以操作。

2.8.4 I2C-EEPROM 读写

测试程序名称：　I2C-EEPROM.exe		备注
驱动程序源代码在 BSP 中的位置	\Mini2440\SRC\DRIVERS\IIC	
开发编译工具	Platform Builder 5 with 2007 Patch	
开发板上对应的设备名	/dev/buttons	
其他：		

在“友善之臂”程序组中点“I2C-EEPROM 测试”图标，打开如下界面：

由上到下依次可以看到：写 EEPROM 按钮、写入字符编辑区、读取 EEPROM 按钮、读出字符编辑区。

点开任务栏上的“软键盘”按钮，在“写入字符编辑区”输入一些 ASC 字符，点“Write”按钮，这时该按钮变为进度条，并指示正在写入的进度，；点“Read”按钮，这时按钮变为进度条，并指示正在读取的进度。如图。

2.8.5 PWM 控制蜂鸣器

测试程序名称：　PWM-Buzzer.exe		备注
驱动程序源代码在 BSP 中的位置	Mini2440\SRC\DRIVERS\PWM	
开发编译工具	Platform Builder 5 with 2007 Patch	

开发板上对应的设备名		
其他：		

在“友善之臂”程序组中点“PWM-buzzer”图标，打开如下界面：

程序中默认的 PWM 输出为 1000Hz，点“Start”按钮开始驱动蜂鸣器发声，此时可以通过点击列表框的“上”或者“下”按钮改变 PWM 输出的频率，同时也可以听到蜂鸣器输出声音的改变。点“Stop”按钮中止 PWM 输出。

2.8.6 看门狗

测试程序名称：　Watchdog.exe		备注
驱动程序源代码在 BSP 中的位置	Mini2440\SRC\DRIVERS\WDT	
开发编译工具	Platform Builder 5 with 2007 Patch	

开发板上对应的设备名		
其他：		

看门狗是嵌入式系统中最常见的功能之一，S3C2440 芯片本身就带有看门狗，我们提供的最新 WindwsCE6 BSP 中已经包含了它的驱动，现在我们就在应用程序中启动它。

点“友善之臂”程序组，点击打开“看门狗”，如图：

注意，先不要点“Start”按钮，请看红色区域的提示：

一旦启动了看门狗，它就无法停止了，只有不停的去喂它，否则系统就会复位重启，我们在此设定的倒数时间为 15 秒。

为了形象表示喂狗的动作，当喂狗时我们就丢给它一只骨头吃，如果一直点“Feed”按钮，它就一直有骨头吃，这样系统也不会复位重启，如图：

2.8.7 LCD 测试

测试程序名称：　LCD-Test.exe		备注
驱动程序源代码在 BSP 中的位置	Mini2440\SRC\DRIVERS\Display	
开发编译工具	Platform Builder 5 with 2007 Patch	
开发板上对应的设备名		
其他：		

LCD 测试程序是为了检测 LCD 有无坏点而做，对于学习型开发板来讲，我们最大可容许不超过 3 个坏点。

在“友善之臂”程序组中，点击打开“LCD 测试”，如图：

本程序有自动和手动两种测试模式：

Auto-loop 是自动循环模式，执行它将会按图中依次循环全屏显示红、黄、白、青、蓝、绿、粉红、黑共八种颜色，此过程中，点击屏幕任何地方就可以返回。

Manual-control 是手动模式，执行之后，每点击一次触摸屏，就会切换到另一种颜色，直至红、黄、白、青、蓝、绿、粉红、黑八色完成一个循环，然后返回。

2.6.8 CMOS 摄像头预览拍照

测试程序名称：　CMOS-Camera.exe		备注
驱动程序源代码在 BSP 中的位置	Mini2440\SRC\DRIVERS\Camera	
开发编译工具	Platform Builder 5 with 2007 Patch	
开发板上对应的设备名		
其他：		

本程序需要使用本公司提供的 CMOS 摄像头模块 CAM130，开机之前，把摄像头模块插到开发板的 CAMERA 接口上，在“友善之臂”程序组中找到“CMOS 摄像头”，点击打开它，如图：

当点击运行 CMOS-Camera 程序时，它将开始初始化 CMOS 摄像头设备，此时界面如左下图，如果 CMOS 摄像头已经接上，则稍等片刻，就会看到动态的影像了，如图：

点 Snap 按钮可以对当前的动态预览进行拍照，拍照后 Snap 按钮变为 Continue，点击它可以继续动态预览，同时照片会保存到“文档”组中(实际位于开发板的 My Documents 目录中，格式为 bmp)，因为本开发板并没有预装图片浏览器，你可以把该图片文件通过 SD 卡或者优盘，或者同步的方式转移到 PC 上浏览。

2.8.9 录音测试

测试程序名称： Recorder.exe		备注
驱动程序源代码在 BSP 中的位置	Mini2440\SRC\DRIVERS\Wavedev	
开发编译工具	Platform Builder 5 with 2007 Patch	
开发板上对应的设备名		
其他：		

在“友善之臂”程序组中，点击打开“Recorder”程序，出现如下界面：

根据提示，点“录音”按钮开始录音，这时对着板上的麦克风说话，程序开始录音，点“停止”按钮结束录音，如图：

此时可以点“播放”按钮会循环播放刚才的录音，

说明：该录音程序并不保存录音结果。

2.8.10 屏幕旋转并保存

说明：屏幕旋转的驱动程序已经包换在 LCD 驱动中，它不需要特别的硬件支持，是纯软件实现的，因此无需另外的单独驱动。

LCD 驱动的位置：mini2440\Src\Drivers\Display

系统启动后，在“友善之臂”程序组点击运行“Rotate”，运行界面如左下图，

点击其中的“Rotate”按钮，屏幕会按逆时针方向 90 度，如右下图。

此时点 Close 按钮，该旋转结果会保存在注册表中，下次开机运行时将会以“横屏”的方式出现，如右上图

2.8.11 串口助手

测试程序名称： SerialPort.exe		备注
驱动程序源代码在 BSP 中的位置	Mini2440\SRC\DRIVERS\Serial	
开发编译工具	Platform Builder 5 with 2007 Patch	
开发板上对应的设备名		
其他：		

说明：本开发板提供的 BSP 包含三个串口的标准驱动，要测试串口 2,3，需要使用串口扩展小板。

在“友善之臂”程序组点击运行“SerialPort”，运行界面如左下图，

点“设置”按钮，打开设置窗口，设置串口号为 COM2，波特率为 115200，其他设置如图(右上)，点确定返回主窗口。

同时，连接好扩展板的 COM2 到 PC 一端，并在 PC 端把相应的串口作相同的设置。

在主窗口中点“打开端口”按钮，此时该按钮会变为“关闭端口”，在“发送区”输入一些字符，点“发送”按钮，如左下图，这时会在 PC 端的串口终端接收到从开发板发送来的字符，如右下图：

然后，在串口调试助手的主窗口点“接收”按钮（该按钮会改变为“不接收”），在 PC 端的串口终端输入一些字符(通过超级终端是无法看到的)，在输入的同时，我们看到输入的字符会在开发板串口调试助手的接收区显示，如图：

我们还可以使用同样的方法测试 COM3，在此就不作详细的说明了。

2.8.12 触摸屏校正

触摸屏驱动程序源代码位于 BSP 的如下目录：
Mini2440\SRC\DRIVERS\Touch

缺省安装的 wince 系统的触摸屏校正参数一般适用于统宝 3.5”LCD，但因为每个触摸屏的物理特性不同，有时可能不太准确，特别是不同尺寸的时候，这时就需要重新校正，如下步骤。

请先接上 USB 鼠标，点开始->设置->控制面板，找到“笔针”图标，双击打开“笔针属性”窗口，点“校准”选项卡的“再校准”按钮，开始重新校正。

根据系统提示，使用五点校正法用触摸笔开始校正，校正完毕，将会跳出如下窗口，这时随便点一个位置即可返回“笔针属性”窗口，点“OK”保存退出。

如果您想保存本次校准的参数，请点“开始->挂起”，然后再重新开机就可以了。

2.8.13 设置网络参数以连接互联网

测试程序名称：　系统自带的网络设置		备注
驱动程序源代码在 BSP 中的位置	Mini2440\SRC\DRIVERS\dm9000	
开发编译工具	Platform Builder 5 with 2007 Patch	
开发板上对应的设备名		
其他：		

只有正确设置了 IP 地址和网关以及 DNS 等参数，才可以使用开发板连接使用互联网或者局域网。

设置网络参数的步骤和标准的 Windows 系统十分相似，点“开始”->“控制面板”，找到网络设置选项，并找到相应的网卡 DM9CE1 如图。

双击打开 DM9CE1 图标，如图，在这里，你可以设置静态的 IP 地址，也可以设置动态获取 IP 的方式，按照你所在的网络环境实际参数填写即可，如图为缺省的设置参数。

确定网络设置参数无误，就可以打开浏览器上互联网了。

2.8.14 背光设置

测试程序名称： 系统自带的背光设置		备注
驱动程序源代码在 BSP 中的位置	Mini2440\SRC\DRIVERS\Backlight	
开发编译工具	Platform Builder 5 with 2007 Patch	
开发板上对应的设备名		
其他：		

我们提供的最新 WindowsCE6 已经支持标准的系统背光开关设置，在此，你可以设置

背光常亮，或者延时关闭，通过触摸屏、鼠标、板上的按键等都可以唤醒背光，实际上，此处实现的背光设置已经属于电源管理的一部分，只不过本开发板仅有外部电源支援。

要进入背光设置界面，有 2 种方式：

可以在空白的桌面处通过点击鼠标右键，再点属性

也可以点“开始”->“控制面板”->“显示”

如图为背光设置界面的缺省设置

可见，使用外部电源供电时，如果在 1 分钟之内触摸屏或者按键等 HMI 设备没有动作，背光将会自动熄灭，你也可以把时间改为其他数值，设置完毕，只需耐心等待即可看到结果了。

为了让背光常亮，可以把上面界面中“使用外部电源并且设备空闲”取消就可以了，如下图

2.8.15 设置实时时钟并保存

测试程序名称：　系统自带的时间设置		备注
驱动程序源代码在 BSP 中的位置	Mini2440\SRC\COMMON\Rtc	
开发编译工具	Platform Builder 5 with 2007 Patch	
开发板上对应的设备名		
其他：　注意此设备驱动和其他设备不太相同，它并不在常规的“设备驱动”目录中		

点任务栏右下角的时间，出现时间设置窗口，根据提示进行设置就可以了，设置完毕，点“OK”退出，设置时间不需要点“挂起”保存。

2.8.16 设置程序开机自动运行

有很多用户希望在开机时就可以运行自己的程序，有很多办法可以实现，通过 google 搜索一下关键词“WinCE 开机自动运行”你就会得到很多结果，我们选择通过修改注册表的方式实现了这一功能，并且只需简单的设定就可以，无需手工修改注册表。当然，这是一个我们专门制作的小程序，它的使用方法如下。

在“友善之臂”程序组中点“AutoRun-Setting”并运行它，如图

可以看到，缺省的系统中没有任何 Apps 被设定，点“Browser”找到一个需要开机自动运行的程序，在此我们以“AD-Convert“为例，如图

说明：AD-Convert.exe 程序位于\Windows 目录下，其他位置的同名文件均为快捷方式。

点“OK”返回设置界面，再点“Close”关闭该设置程序。

好了，现在直接关闭电源或者按复位键，稍等片刻，你就可以看到 AD-Convert 程序已经自动运行了。

2.8.17 使用优盘

在 wince 中使用优盘和使用标准的 windows 使用优盘类似，当 WINCE 系统启动以后，把优盘插入 USB Host 接口，这时板子给优盘供电，优盘的指示灯会闪烁，等待几秒系统就自动加载优盘了，这时可以双击桌面的“我的电脑”图标，打开资源管理器，可以看到优盘的盘符：硬盘，双击硬盘即可进入优盘进行数据读写了。如下图所示。

2.8.18 使用 SD/MMC 卡

说明：因为采用了最新架构的 BSP，目前开发板最大可以支持 32G 普通和高速 SD 卡。

把 SD/MMC 卡插入板上的 SD 插槽，资源管理器中就可以看到 SD 卡的盘符：Storage Card，双击打开进入该目录，就可以对 SD/MMC 卡进行读写了。

2.8.19 使用 ActiveSync 进行 USB 同步通讯

注意：安装同步连接的 USB 驱动时，请使用光盘中的“\windows 平台工具\CE 用同步 USB 驱动”

我们假定您已经按照第九章的步骤安装好驱动和 ActiveSync 同步软件。

在默认情况下，当 WINCE 系统启动以后，USB Device 连接和设置就已经做好了，这时可以直接用 USB 连接开发板和 PC 机，即可看到如下成功连接的情况：

2.8.20 使用 USB 无线网卡

我们预装的 WINCE5 系统已经集成了无线网卡驱动(型号为: VNUWLC41)，您可以直接把无线网卡模块插入板子上的 USB Host 接口，将会跳出如下窗口，同时列表中会出现您附近能够提供无线网络的服务器组名。

双击打开您要连接的无线网络服务提供组进行设置，这里选择的是“FriendlyARM”，输入该组的网络密钥(如果没有进行安全设置，就不用输入了)，这里的密码是明文方式显示的。点“OK”设置完毕，返回上一个窗口。

可以看到系统正在尝试连接，过一会就可以看到已经与 FriendlyARM 无线网组连接上了。

这时选择点击“IP 信息”选项卡，可以查看该无线网络已经被分配的信息，点“更新”按钮以更新信息，如下图。

第三章 备份恢复系统及安装更新

本节内容主要通过图解介绍如何通过 USB 备份和恢复您的开发板系统程序，以及如何安装更新系统。

很多人特别是初学者认为，安装嵌入式系统是十分复杂的事情，需要很高的技术水平，特别是 Linux 系统，总以为要输入很多的命令进行配置安装。通过本节的介绍，你将会发现安装和更新系统简直比在 PC 上安装 Windows 系统还要简单，只要几分钟的时间，输入不超过 5 个字母，您就可以随意更新您的系统了。

如果你认为快速的输入一些命令行指令是一件很酷的事情，可以参考附录 2。

提示：本节内容的各个小节均具有独立性，用户可以根据自己的目的需要进行独立阅读操作。

本小节的工作环境为 Windows/2000/XP。

3.1 备份和恢复系统

在开发过程中，我们经常需要不断的烧写和更新系统进行调试，有时您的系统程序会配合的很好，让你有大功告成的感觉，这时我们都希望能保存当前的目标系统程序以供以后参考，或者完全复制当前的系统装入到另一个目标板；特别是当项目要求紧迫，而重新构建一个同样的系统又很复杂的时候，这时使用系统自带的备份和恢复功能，就能够快速有效地解决您的问题。

注意：对于 64M NandFlash 版 mini2440/micro2440 仅能实现本机备份和恢复；对于 128M-1Gb Nand Flash 版 mini2440/micro2440 可以实现异机备份恢复

另外，本开发板预装的系统一般为 Linux，如果你不小心删除或者改动了它，可以依照下面 3.2 章节的内容恢复到出厂状态，所附光盘并没有提供此处所指的备份文件。

3.1.1 备份系统

注意：本小节假定您已经按照前面的方法安装了 USB 驱动，并把开发板设置为 Nor Flash 启动，设置方法请参考前面的章节。

另外，备份并不会破坏任何 Flash 的数据，备份之前请检查您的系统是否能够正常运行使用，以便参考。

(1) 连接好串口，打开超级终端，上电启动开发板，进入 BIOS 功能菜单：

(2) 选择功能号[u]开始备份 Nand Flash 内容到文件，如图所示：

(3) 打开 DNW 程序，接上 USB 电缆，如果 DNW 标题栏提示[USB：OK]，说明 USB 连接成功，这时选择 DNW 菜单的 Usb Port → Backup NandFlash to File，如图所示：

跳出文件保存窗口，选择所要保存文件的位置，并命名(这里保存文件为 backup.bin)，如图：

系统开始备份，备份的进度如图所示：

备份完毕，DNW 窗口会出现如图信息：

最后备份生成的文件大小为 66M byte，这是因为包含了 Nand Flash 的所有字节信息，关于 Nand Flash 更多的介绍请查看相应的数据手册。

提示：如果你使用的是其他容量的 Nand Flash 版 mini2440/micro2440，最后生成的文件大小则不同。

3.1.2 使用备份文件恢复系统

注意：本小节假定您已经按照前面的方法安装了 USB 驱动，并把开发板设置为 Nor Flash 启动，设置方法请参考前面的章节。

另外，要使用备份文件恢复系统，必须先按照上一步创建生成备份文件。恢复系统会擦除整片 Nand Flash！

下面我们开始介绍如何使用备份文件恢复系统。

(1) 连接好串口，打开超级终端，上电启动开发板，进入 BIOS 功能菜单：

(2) 选择功能号[r]开始使用备份文件恢复整个 Nand Flash，如图所示：

(3) 打开 DNW 程序，接上 USB 电缆，如果 DNW 标题栏提示[USB：OK]，说明 USB 连接成功，这时选择 DNW 菜单的 Usb Port → Transmit/Restore，如图所示：

跳出文件选择窗口，选择要使用的备份文件(如上一步骤所生成的 backup.bin)，点“打开”开始恢复系，如图所示：

备份完毕，就可以把开发板设置为 Nand Flash 启动，按复位或者电源开关重新启动系统了。

3.2 安装 Linux 系统

注意：本小节假定您已经按照前面的方法安装了 USB 驱动，并把开发板设置为 NOR Flash 启动，系统更新和安装完毕请设置为 Nand Flash 启动，设置方法请参考前面的章节。

说明：安装 Linux 所需要的二进制文件位于光盘的 **images\linux** 目录中。
安装 Linux 系统主要有以下步骤：
(1)对 Nand Flash 进行分区
(2)安装 bootloader
(3)安装内核文件
(4)安装文件系统

下面是详细的步骤。

3.2.1 分区

提示：分区将会擦除 Nand Flash 里面的所有数据
(1) 连接好串口，打开超级终端，上电启动开发板，进入 BIOS 功能菜单：

(2) 选择功能号**[f]**开始对 Nand Flash 进行分区，如图所示。

说明：有的 Nand Flash 分区时会出现坏区报告提示，因为 supervivi 会对坏区做检测记录，因此这将不会影响板子的正常使用。

提示：普通的 Nand Flash 并不能保证所有扇区都是完好的，如果有坏区，系统软件会对它们做检测处理，而不会影响整个软件系统的使用。保证完全无坏区的 Nand Flash 另有型号，请参考光盘中的 Flash 选项指南(**Samsung_Nand_Flash.pdf**)，这种 Flash 一般没有现货，而且订货周期长，价格昂贵，一般场合很少用到；其他品牌的 Nand Flash 也与此类似。

3.2.2 安装 bootloader

注意：老用户必须先更新 NOR FLASH 里面的 BIOS 为最新才可以进行下面的步骤。

我们针对 Linux 系统提供了两种 bootloader：vboot 和 supervivi。

vboot 是一个十分简易的开源软件，由友善之臂设计制作，它可以兼容启动 64M/128M-1Gb Nand Flash 版 mini2440/micro2440；之前我们使用的是 vivi，它是由三星原厂提供的，它的结构复杂，只能启动 64M 版本，因此我们目前已经弃用了。

Supervivi 由 vivi 发展而来，针对 64M 和 128M-1GB 开发板分别有 supervivi-64M 和 supervivi-128M 两个文件，它们的用法和功能是一样的，我们统称为 supervivi，只是在选择具体的文件时有所区分；它并不是开源的。

(1) 打开 DNW 程序，接上 USB 电缆，如果 DNW 标题栏提示[USB：OK]，说明 USB 连接成功，这时根据菜单选择功能号**[v]**开始下载 supervivi

(3)点击“USB Port->Transmit/Restore”选项，并选择打开文件 supervivi(该文件位于光盘的 images/linux/目录)开始下载。

(4)下载完毕，BIOS 会自动烧写 supervivi 到 Nand Flash 分区中，并返回到主菜单。

3.2.3 安装 Linux 内核

说明：Linux 内核可以自适应 64M/128M-1Gb Nand Flash 版 mini2440/micro2440

(1)在 BIOS 主菜单中选择功能号**[k]**，开始下载 linux 内核 zImage

(2) 点击“USB Port->Transmit”选项，并选择打开相应的内核文件 zImage(该文件位于光盘的 images\linux\目录)开始下载。

内核文件说明：

zImage_w35 – 适用于 3.5”横屏(LQ035Q1DG06 或兼容)

zImage_x35 – 适用于 Sony 3.5”LCD

zImage_n35 – 适用于 NEC3.5”LCD

zImage_t35 – 适用于统宝 3.5”LCD

zImage_l80 – 适用于 Sharp 8”LCD(或兼容)

zImage_a70 – 适用于 7 寸真彩屏，分辨率为 800x480

zImage_VGA1024x768 – 适用于 VGA 模块输出，分辨率为 1024x768

实际可能与此不完全相同，请参考 images\linux 目录下的 readme.txt 文件说明

(3) 下载完毕，BIOS 会自动烧写内核到 Nand Flash 分区中，并返回到主菜单，如图：

3.2.4 安装根文件系统

说明：针对 64M/128M-1Gb mini2440/micro2440，有不同的文件系统烧写映象文件：

root_qtopia-64M.img 和 root_qtopia-128M.img，实际上它们的内容都是完全相同的，只是制作工具(mkyaffs2image)不同，我们把文件系统统称为 root-qtopia.img。

(1)在 BIOS 主菜单中选择功能号**[y]**，开始下载 yaffs 根文件系统映象文件

(2) 点击“USB Port->Transmit/Restore”选项，并选择打开相应的文件系统映象文件 root_qtopia.img(该文件位于光盘的 images\linux 目录)开始下载。

根文件系统映象文件说明：

root_qtopia-64M.img

-缺省安装的文件系统映象文件，可以同时支持 USB 鼠标和触摸屏，并自动识别 VGA 模块输出和 NFS 启动，适用于 64M Nand Flash 版 mini2440/micro2440

root_qtopia-128M.img

-缺省安装的文件系统映象文件，可以同时支持 USB 鼠标和触摸屏，并自动识别 VGA 模块输出和 NFS 启动，适用于 128M-1GB Nand Flash 版 mini2440/micro2440

(3) 下载过程如图所示，下载完毕，BIOS 会自动烧写内核到 Nand Flash 分区中，并返回到主菜单，如图：

提示：此过程大概需要 2-3 分钟，下载的文件越大，下载和烧写的时间就会越长。

注意：下载完毕，请拔下 USB 连接线，如果不取下来，有可能在复位或者启动系统的时候导致您的电脑死机。

在 BIOS 主菜单中选择功能号**[b]**，将会启动系统。

如果您把开发板的启动模式设置为 Nand Flash 启动，则系统会在上电后自动启动。

3.3 安装 WinCE 系统

说明：

安装 WinCE5 所需要的二进制文件位于光盘的“\images\wince5.0”目录中

安装 WinCE6 所需要的二进制文件位于光盘的“\images\wince6.0”目录中

因为安装 WindowsCE5/6 的步骤和方式都是完全一样的，只是选择的文件不同，下面统称为 WindowsCE

安装 WindowsCE 系统主要有以下步骤：

(1)安装 bootloader

(2)安装 BootLogo(bmp 格式)

(3)安装 WindowsCE 内核映象

下面是详细的步骤：

提示：请先连接好串口，打开超级终端，上电启动开发板，进入 BIOS 功能菜单

3.3.1 安装 Bootloader

注意：老用户必须先更新 NOR FLASH 里面的 BIOS 为最新才可以进行下面的步骤。

本开发板提供了两种 bootloader 均可启动 WINCE：supervivi 和 nboot.bin，它们的说明如下：

Bootloader 种类:	Supervivi-64M 或 Supervivi-128M	Nboot.bin
二进制映象文件的位置	\images\wince5.0 \images\wince6.0	\images\wince5.0 \images\wince6.0
源代码位置	无	WindowsCE5.0\NBOOT 或 WindowsCE6.0\NBOOT
项目文件	无	Nboot.mcp
编译器	Arm-linux-gcc	ADS1.2
功能描述	见前面章节 BIOS 功能介绍。 注意：supervivi 的总体启动速度比 nboot 要快 2-3 秒，但不支持开机	● 支持快速启动 WindowsCE(一般 6-10 秒，视内核大小而定) ● 支持开机 Logo(通过 USB 下载普通

	Logo 和进度条	的 bmp 格式真彩图片) ● 带进度条(用户可根据源代码配置文件自行定义进度条的各个属性：颜色、长宽、起始位置等)

说明：
- supervivi 由友善之臂维护和发展，不提供源代码
- NBOOT 的配置和编译见 4.6 章节
- 此处提供的 Nboot 分别有 Nboot_W35.bin, **Nboot_N35.bin**、**Nboot_X35.bin**、**Nboot_T35.bin**、**Nboot_L80.bin**、**Nboot_A70.bin**、**Nboot_VGA1024x768.bin**，它们分别对应于不同的 LCD 型号，因为 Nboot 并不能自动识别 LCD，为了用户使用方便，我们根据配置文件分别做了编译。

下面以 Nboot_T35.bin 为例(以下简称 Nboot.bin)，介绍 Nboot.bin 的下载烧写步骤：

(1) 打开 DNW 程序，接上 USB 电缆，如果 DNW 标题栏提示[USB：OK]，说明 USB 连接成功，这时根据菜单选择功能号**[n]**开始下载 Nboot.bin

(3)点击“USB Port->Transmit”选项，并选择打开文件 Nboot.bin(该文件位于光盘的\images\wince6.0 目录)开始下载。

(4)下载完毕，BIOS 会自动把 Nboot.bin 烧写到 Nand Flash 的 Block 0

3.3.2 下载烧写 BootLogo

说明：WindowsCE 系统的启动过程有两种 Logo：BootLogo 和 StartLogo。其中 BootLogo 是由 Nboot 加载显示的，用户可以通过修改 Nboot 的源代码调整 BootLogo 的显示位置和背景色；StartLogo 则属于 BSP 的一部分，它是一个数组文件(StartLogo.c)，位于“mini2440\Src\Kernel\Oal”目录，由该目录下的 init.c 文件实现加载显示，StartLogo.c 文件可以通过本光盘中的 StartLogoMaker.exe 工具制作生成。

另外，**BootLogo 通过 Supervivi 的[l]功能下载烧写到 Nand Flash，它必须是 24bit 真彩 bmp 图片(一般 bmp 都是真彩的)，并且不能大于 2M，可以支持 240x320 – 1024x768 各个尺寸的真彩图片。**

(1)在 BIOS 主菜单中选择功能号**[l]**，开始下载 bmp 图片作为 BootLogo(光盘中已经准备好一个 BootLogo.bmp 图片)

(2) 点击“USB Port->Transmit/Restore”选项，并选择文件 bootlogo.bmp

(3) 下载完毕，BIOS 会自动烧写 bootlogo.bmp 到 Nand Flash 中，并返回到主菜单。

3.3.3 安装 wince 内核映象

说明：Supervivi 已经集成了 WinCE 内核烧写功能，不需要加载其他的跳板文件(诸如

Eboot 等)，Supervivi 先通过 USB 下载 NK.bin 到内存中，然后格式化 Nand Flash，再创建系统分区(NK.bin 所占用的 NandFlash 空间，BINFS 格式，为只读)和用户分区(对应于 WinCE 启动后的 ResidentFlash 目录，FAT 格式，可读写)

我们提供的 WinCE 系统可以自适应支持 64M/128M-1Gb mini2440/micro2440，因此烧写的 WinCE 内核文件是一样的。

(1)在 BIOS 主菜单中选择功能号**[w]**，开始下载 WINCE 内核

(2) 点击“USB Port->Transmit/Restore”选项，并选择打开相应的内核文件 NK.bin(该文件位于光盘的“\images\wince5.0”目录或者“\images\wince6.0”目录)开始下载。

WINCE 内核文件说明：

NK_W35.bin – 适用于横向显示 3.5”LCD(LQ035Q1DG06 或兼容)

NK_X35.bin –适用于 Sony　3.5" LCD

NK_N35.bin –适用于 NEC 3.5" LCD

NK_T35.bin – 适用于统宝 3.5”LCD

NK_L80.bin – 适用于 Sharp 8”LCD(或兼容)

NK_A70.bin – 适用于 7”LCD

NK_VGA1024x768.bin – 适用于 VGA 模块输出，分辨率为 1024x768

实际可能与此不完全相同，请参考“images\wince5.0”或者“images\wince6.0”目录下的 readme.txt 文件说明

下载完毕，BIOS 会开始格式化 Nand Flash，并依次创建分区烧写 WinCE 内核文件，烧写完毕，会自动启动 WinCE 系统，整个过程串口信息如下图：

3.4 下载到内存运行

Supervivi 支持很多程序下载到内存中运行，例如 2440test、uCos2、Linux 内核、WinCE 内核等。对于简单的系统，一般可以通过 USB 下载到内存中运行调试，对于比较高级完善的系统则烧写到 Nand Flash 中运用其本身的功能进行进一步的调试。

3.4.1 运行 2440test

文件信息		备注
在光盘中的位置	**光盘\images\2440test**	2440test_W35.bin 适用于 W35 2440test_N35.bin 适用于 NEC3.5”LCD 2440test_T35.bin 适用于统宝 3.5”LCD 2440test_A70.bin 适用于群创 7”LCD 2440test_L80.bin 适用于 Sharp 8”LCD(或兼容) 2440test_VGA1024x768.bin 适用于 VGA 模块，分辨率 1024x768
指定下载运行地址	**0x30000000**	
对应的源代码位置	非操作系统示例代码\2440test	
项目名称	2440test.mcp	默认项目使用 NEC3.5 寸屏
编译工具	ADS1.2	
说明： ● 若要烧写到 Nand Flash 运行，需选择 supervivi 的[a]功能，无需指定下载地址 ● 通过修改“\非操作系统示例代码\2440test\inc\Option.h”中 LCD_TYPE 的定义，可以编译出可以编译出适用于不同 LCD 型号的目标文件		

(1)连接好开发板电源，串口线，USB 线，并**设置拨动开关 S2 为 Nor Flash 启动系统**，分别打开串口超级终端和 DNW，上电启动开发板。

(2)保证 USB 驱动已经安装好(前面已经详细介绍了 USB 驱动的安装方法)，这时可以看到 DNW 的标题栏显示[USB：OK]，如果没有安装好驱动会显示[USB：x]，如图所示：

(3)点 DNW 菜单 Configuration，设置 USB 下载运行地址为 0x30000000

(4)这时在超级终端的 BIOS 功能菜单中选择功能号[d]，出现 USB 下载等待提示信息：

(5)点击 DNW 程序的“USB Port”→“Transmit”，选择 2440test_T35.bin 映象文件(在光盘的 images\2440test 目录下面，你也可以根据实际的 LCD 型号选择相应的文件)，接着点“打开”，这样就开始下载了。

(6)下载结束后，会自动运行，出现如下界面，就可以按照 2.3 章节来操作测试了：

同时在 LCD 上会出现一副图片显示。

3.4.2 运行 uCos2

文件信息		备注
在光盘中的位置	**\images\uCos22440uCos2.bin**	2440uCos2_W35.bin 适用于 W35 2440uCos2_N35.bin 适用于 NEC3.5”LCD 2440uCos2_X35.bin 适用于 Sony 3.5”LCD 2440uCos2_T35.bin 适用于统宝 3.5”LCD 2440uCos2_L80.bin 适用于 Sharp 8”LCD(或兼容) 2440uCos2_A70.bin 适用于群创 7”LCD 2440uCos2_VGA1024x768.bin 适用于 VGA 显示输出，分辨率：1024x768
指定下载运行地址	**0x30000000**	
对应的源代码位置	uCos2\uCos2	
项目名称	uCOS_2440.mcp	默认项目使用 NEC3.5 寸屏
编译工具	ADS1.2	
说明： ● 若要烧写到 Nand Flash 运行，需选择 supervivi 的[a]功能，无需指定下载地址 ● 通过修改“uCos2\uCos2\S3C2440\includes\option.h”中 LCD_TYPE 的定义，可以编译出适用于不同 LCD 型号的目标文件		

(1)连接好开发板电源，串口线，USB 线，并**设置拨动开关 S2 为 Nor Flash 启动系统**，分别打开串口超级终端和 DNW，上电启动开发板。

(2)保证 USB 驱动已经安装好(前面已经详细介绍了 USB 驱动的安装方法)，这时可以看到 DNW 的标题栏显示[USB：OK]，如果没有安装好驱动会显示[USB：x]，如图所示：

(3)点 DNW 菜单 Configuration，设置 USB 下载运行地址为 0x30000000

(4)这时在超级终端的 BIOS 功能菜单中选择功能号[d]，出现 USB 下载等待提示信息：

(5)点击 DNW 程序的“USB Port”→“Transmit”，选择 2440uCos2_T35.bin 映象文件(在光盘的 images\uCos2 目录下面，你也可以根据实际的 LCD 型号选择相应的文件)，接着点“打开”，这样就开始下载了。

(6)下载结束后，会自动运行，出现如下界面，同时 LCD 上会出现一幅图片并且屏幕上会显示一些汉字和英文：

3.4.3 运行 Linux

文件信息		备注
在光盘中的位置	**\images\linux**	zImage_W35 适用于 W35(见面板标识) zImage_N35 适用于 NEC3.5”LCD zImage_X35 适用于 Siny 3.5”LCD zImage_T35 适用于统宝 3.5”LCD zImage_L80 适用于 Sharp 8”LCD(或兼容) zImage_A70 适用于群创 7”LCD zImge_VGA1024x768 适用于 VGA 显示输出，分辨率：1024x768
指定下载运行地址	**0x30008000**	该地址无需通过 dnw 指定
对应的源代码包位置	**linux-2.6.32.2-mini2440-20100106.tgz**	因为内核经常更新，请以最新日期为准
项目名称	无	
编译工具	Arm-linux-gcc-4.4.3	
说明：配置和编译内核见第 8 章		

说明：在内存中运行 linux 系统，一般是指 linux 内核(具体为 zImage 文件)，文件系统是无法通过网络或者 USB 下载到内存中运行的。一般是借助于 linux 的启动命令，指定 NFS(网络文件系统)，或者使用开发板“本地”文件系统，如 yaffs(可通过 supervivi 的“y”命令烧写 root_default.img 或者其他文件系统映象文件)。

如何通过 linux 命令指定 NFS 启动系统？

在本开发板中，首先进入 supervivi 菜单，按“q”键进入 supervivi 的命令行模式，输入(详细见 5.1.4 章节)：

Supervivi>param set linux_cmd_line "console=ttySAC0 root=/dev/nfs nfsroot=192.168.1.111:/opt/FriendlyARM/mini2440/root_qtopia ip=192.168.1.70:192.168.1.111:192.168.1.111:255.255.255.0:MINI2440.arm9.net:eth0:off"

下面是使用 USB 下载 linux 内核到开发板中，并启动运行的步骤，这里使用的是“本地”文件系统 root_qtopia.img：

(1)连接好开发板电源，串口线，USB 线，并**设置拨动开关 S2 为 Nor Flash 启动系统**，分别打开串口超级终端和 DNW，上电启动开发板。

(2)保证 USB 驱动已经安装好(前面已经详细介绍了 USB 驱动的安装方法)，这时可以看到 DNW 的标题栏显示[USB：OK]，如果没有安装好驱动会显示[USB：x]，如图所示：

(3)这时在超级终端的 BIOS 功能菜单中选择功能号[z]，出现 USB 下载等待提示信息：

(4)点击 DNW 程序的“USB Port”→“Transmit”，选择 zImage_t35 或者 zImage_A70 这个映象文件(在光盘的 images\linux 目录下面)，接着点“打开”，这样就开始下载了。

说明(仅供参考)：功能[z]实际是把 zImage 文件下载到内存地址为 0x30008000 的地方，大小为 0x200000。按[q]进入 supervivi 的命令行模式，输入“**load ram 0x30008000 0x200000 u**”也可以实现相同的功能。

(5)下载结束后，回到 supervivi 菜单，这时按功能号[g]，就可以启动系统了。

说明（仅供参考）：功能[g]的功能实际是 supervivi 的命令行“**boot ram**”，在 supervivi 的命令行输入“**boot ram**”可以达到相同的功能效果。

若出现如下类似界面，则说明没有指定好文件系统，可以在 supervivi 菜单中选择[y]烧写一个 root_qtopia.img，或者使用 NFS 启动系统：

3.4.4 运行 WinCE

说明：WindowsCE5/6 的执行映象文件有两种：NK.bin 和 NK.nb0，它们是同时生成的。其中 NK.bin 一般称为发行版映象文件，它在真正烧写到 Nand Flash 之前会按实际大小转换成和 NK.nb0 相同的结构格式；NK.nb0 是固定大小的可运行时映象文件，它可以直接烧写到 Nand Flash 运行，也可以通过 USB 等方式下载到内存中运行，其运行地址是 0x30200000

文件信息		备注
在光盘中的位置	**\images\ wince5.0** **或** **\images\ wince6.0**	NK_W35.nb0 适用于 W35(见面板标识) NK_N35.nb0 适用于 NEC3.5”LCD NK_X35.nb0 适用于 Sony 3.5”LCD NK_T35.nb0 适用于统宝 3.5”LCD NK_L80.nb0 适用于 Sharp8”LCD(或兼容) NK_A70.nb0 适用于群创 7”LCD NK_VGA1024x768.nb0 适用于 VGA 显示输出，分辨率：1024x768
指定下载运行地址	**0x30200000**	必须在 dnw 中指定此下载运行地址，详见以下操作
对应的源代码包位置	**WindowsCE5.0\mini2440** **或者** **WindowsCE6.0\mini2440**	
项目名称	**mini2440.pbxml**	
编译工具	Platform Builder 5.0/6.0	
说明：mini2440 目录(有日期尾缀)是本开发板的 wince 5.0/6.0 BSP，mini2440.pbxml 是相应的项目文件，按照 9/10 章节步骤可以编译出相应的 wince 内核映象文件 nk.bin 和 nk.nb0.		

注意：在内存中运行 nk.nb0，在其启动的时候，因为 wince 启动过程中目录的创建，会破坏 Nand Flash 中一些内容，从而导致原来的系统不再可用，在此请注意！

下面是使用 USB 下载 WINCE 内核到开发板内存中运行的步骤：

(1)连接好开发板电源，串口线，USB 线，并**设置拨动开关 S2 为 Nor Flash 启动系统**，分别打开串口超级终端和 DNW，上电启动开发板。

(2)保证 USB 驱动已经安装好(前面已经详细介绍了 USB 驱动的安装方法)，这时可以看到 DNW 的标题栏显示[USB：OK]，如果没有安装好驱动会显示[USB：x]，如图所示：

(3)点 DNW 菜单 Configuration，设置 USB 下载运行地址为 0x30200000

(4)这时在超级终端的 BIOS 功能菜单中选择功能号[d]，出现 USB 下载等待提示信息：

(5)点击 DNW 程序的“USB Port”→“Transmit”，选择 NK.nb0 映象文件(在光盘的“images\wince5.0”目录或“images\wince6.0”目录下面)，接着点“打开”，这样就开始下载了。

(6)下载结束后，会自动运行，不再返回 supervivi 菜单。这时 PC 机有可能会出现 USB 无法识别的提示，只要把 USB 拔下来，重新插上，就可以看到同步连接了。

第四章 ADS1.2 集成开发环境的使用

ARM ADS 的全称为 ARM Developer Suite，它是 ARM 公司推出的新一代 ARM 集成开发环境，我们使用的 ADS 为 1.2 版本，它取代了早期的 ADS1.1 和 ADS1.0，它可以安装在 WindowsNT/2000/98/95/XP 上面使用。

4.1 使用 ADS 创建 LED 工程

本节通过一个简单的具体实例，介绍如何使用 ADS 集成开发环境。包括如何创建一个新的工程，如何配置编译选项，并编译生成可以直接烧写到 Flash 中的 bin 格式二进制可执行文件。

4.1.1 建立一个工程

在 ADS 集成开发环境中，点 File->New，打开如图所示窗口：

可以看到有 7 种工程类型可以选择：

ARM Excuteable Image：用于由 ARM 指令的代码生成一个 ELF 格式的可以执行映象

文件。

ARM Object Library：用于由 ARM 指令的代码生成一个 armar 格式的目标文件库。

Empty Project：用于创建一个不包含任何库或者源文件的工程。

Makefile Importer Wizard：用于将 Visual C 的 nmake 或者 GNU make 文件转入到 CodeWarrior IDE 工程文件。

Thumb ARM Excutable Image：用于由 ARM 指令和 Thumb 指令的混和代码生成一个可执行的 ELF 格式的映象文件。

Thumb Excutable image：用于由 Thumb 指令创建一个可执行的 ELF 格式的映象文件。

Thumb Object Library：用于由 Thumb 指令的代码生成一个 armar 格式的目标文件库。

我们在这里选择 ARM Executable Image，在“Project name：”中输入工程文件名，本例为“**myled**”，点击“Location：”文本框的“Set”按钮，浏览选择想要保存该工程的路径(本例为“**D:\work**”)，将这些设置好之后，点击“确定”，即可创建一个新的名为 myled 的工程。

这个时候会出现 myled.mcp 窗口，如图所示，同时会在 D:\work 目录下创建一个工程目录 myled，而 myled.mcp 会出现在“**D:\work\myled**”目录中。

对于本例，我们将已经准备好的源文件及其目录（位于光盘的“**非操作系统示例代码\myled**”文件夹中）一起复制到 myled 工程目录，如图

然后在 myled.mcp 项目窗口中，点鼠标右键或者 ADS 菜单 **Project->Add Files…**，如图开始添加该项目索需要的源代码。

点击“打开”按钮确定，这时会跳出如下图所示的提示选择窗口

这里请注意，我们在新建一个工程时，ADS 默认的 target 时 DebugRel，另外还有两个可用的 target，分别为 Realse 和 Debug，它们的含义分别为：

DebugRel：使用该目标选项，在生成目标的时候，会为每一个源文件生成调试信息。

Debug：使用该目标选项，在生成目标的时候，会为每一个源代码生成最完整的调试信息。

Release：使用该目标选项，在生成目标的时候，不会生成任何调试信息。

在本例中，我们使用默认的 DebugRel 选项。

然后把 main.c 也加入到 myled.mcp 项目工程：

到目前为止，一个完整的工程就已经建立了，下面我们开始对该工程进行编译和链接的配置。

4.1.2 编译和链接工程

在进行编译和链接之前，首先需要对生成的目标进行配置，点 Edit 菜单，选择“DebugRel Setting…”（注意：这个选项会因为用户选择的不同目标而有所不同），出现如图所示的设置窗口。

这里的设置有很多，我们主要介绍最常用的一些选项。

- **Target Setting**

Target Name 文本框显示了当前的目标设置。

Linker 选项为用户提供了要使用的连链接器，在这里选择默认的 ARM Linker，使用该链接器，将使用 armlink 链接编译器和汇编器生成相应的工程目标文件。

在 Linker 设置中，还有两个可选项，None 代表不对生成的各个源代码目标文件进行链接，ARM Librarian 表示将编译或者汇编得到的目标文件转换为 ARM 库文件，对于本例，使用默认的链接器 ARM Linker。

Pre-Linker：目前 ADS 并不支持该选项。

Post-Linker：选择在链接完成后，还要对输出文件进行的操作。因为在本例中，希望生成一个可以烧写到 Flash 中去的二进制代码，所以在此选择 ARM fromELF，表示在链接生成映象文件后，再调用 fromELF 命令将含有调试信息的 ELF 格式的映象文件转换为其他格式的文件。

Target Setting 选择最后设置如图所示：

- **Language Settings**

因为在本例中包含汇编代码，所有要用到汇编器，点接选择 **ARM Assembler**，在右侧出现相应的设置选项，在 ADS 集成开发环境中用的汇编器是 armasm，默认的 ARM 体系结构是 ARM7TDMI，在此要改为 ARM920T，字节顺序默认是小端模式，其他设置，采用默认值即可，如图所示：

本例中还包含了 C 语言代码，因此还需要设置 ARM C Compiler 选项，点接选择 ARM C Compiler，在右侧出现相应的设置选项，在 ADS 集成开发环境中用的汇编器是 armcc，默认的 ARM 体系结构是 ARM7TDMI，在此要改为 **ARM920T**，字节顺序默认是小端模式，其他设置，采用默认值即可，如图所示：

细心的读者可能会注意到，在设置框的右下脚，当对某项设置进行了修改，该行中的某个选项就会发生相应的改动，实际上，这行文字显示的就是相应的编译或者链接选项，由于有了 CodeWarrior，开发人员可以不用再去产科繁多的命令行选项，只要在界面中选择或者

撤销某个选项，软件就会自动生成相应的代码，该命令框为习惯在 DOS 下键入命令行的用户提供了极大的方便。

- **Linker 设置**

提示：如果您对 ADS 设置不熟悉，可以使用我们缺省的项目文件，以下部分仅供参考。

点接选择 ARM Linker，在右侧出现相应的设置选项，我们在此详细介绍这些设置框，因为这些选项对最终生成的文件有着直接的影响。

在标签 Output 中，Linktype 中提供了三种链接方式。Partial 方式表示链接器只进行部分链接，经过部分链接生成的目标文件，可以作为以后进一步链接时的输入文件。Simple 方式是默认的链接方式，也是最为频繁使用的链接方式，它链接生成简单的 ELF 格式的目标文件，使用的是链接器中指定的地址映象方式。Scattered 方式使得链接器要根据 scatter 格式文件指定的地址映象，生成复杂的 ELF 格式的映象文件，这个选项一般很少用到。

在本例中，我们选择使用 Simple 方式，这里有一些设置：

RO Base: 这个文本框设置包含 RO 段的加载域和运行域为同一个地址，默认是 0x8000。这里用户要根据自己的硬件实际 SDRAM 地址空间来修改这个地址，保证这里填写的地址，是程序运行时，SDRAM 地址空间所能到达的范围，针对本目标板，SDRAM 的空间范围是 0x3000000-0x34000000，因此这里设置为 0x30000000。

RW Base: 这个文本框设置了包含 RW 和 ZI 输出段的运行域地址。如果选中 split 选项，链接器生成的映象文件将包含两个加载域和两个运行域，此时在 RW Base 中所输入的地址为包含 RW 和 ZI 输出段的域设置了加载域和运行域地址。

Ropi：选中这个设置将告诉链接器使包含有 RO 输出段的运行域位置无关。使用这个选项，链接器将保证下面的操作：检查各段时间的重寻址是否有效；确保任何由 armlink 自身生成的代码是只读位置无关的。

Rwpi：选中该选项将会告诉链接器使包含 RW 和 ZI 输出段的运行域无关。如果这个选项没有被选中，域就标识为绝对。每一个可写的输入段必须是和读写位置无关的。如果这个选项被选中，链接器将进行下面的操作：

检查可读/写属性的运行域的输入段是否设置了位置无关属性；

检查在各段之间的重地址是否有效；

在 Region$$Table 和 ZISection$$Table 中添加基于静态存储器 sb 的选项。

该选项要求 RW Base 有值，如果没有给他指定数值的话，默认为 0。

Split Image：选择这个选项把包含 RO 和 RW 的输出段的加载域分成 2 各加载域，一个是包含 RO 输出段的域，一个是包含 RW 输出段的域。

这个选项要求 RW Base 有值，如果没有给 RW Base 选项的值，则默认是 0。

Relocatable：选择这个选项保留了映像文件的重寻址偏移量。这些偏移量为程序加载器提供了有用信息。在 Options 选项中，需要读者引起主义的是 Image entry point 文本框。它指定映像文件的初始入口点地址值，当映像文件被加载程序加载时，加载程序会跳转到该地址处执行。如果需要，用户可以在这个文本框中输入下面格式的入口点：

入口点地址：这是一个数值，例如 -entry 0x0

符号：该选项指定映像文件的入口点为该符号所代表的地址处，比如：-entry int_handler ，如果该符号有多处定义存在，armlink 将产生出错信息。

offset+object(section)：该选项指定在某个目标文件的段的内部的某个偏移量处为映像文件的入口地址，例如：-entry8+startup(startuoseg)在此处指定的入口点用于设置 ELF 映像文件的入口地址。需要引起注意的是，这里不可以用符号 main 作为入口点地址符号，否则将会出现类似“Image dose not have an entry point(Not specified or not set due to multiple choice)”的错误信息。在 Layout 选项中，需要设置 asm.o 目标文件中的 Init 为整个文件的入口点。关于 ARM Linker 的设置还很多，对于想进一步深入了解的读者，可以查看帮助文件，都有很详细的介绍。

在 Linker 下还有一个 ARM fromELF：

fromELF 是一个实用工具，它实现将链接器，编译器和汇编器的输出代码进行格式转换的功能。例如，将 ELF 格式的可执行映像文件转换成可以烧写到 ROM 的二进制格式文件；对输出文件进行反汇编，从而提取出有关目标文件的大小，符号和字符串表以及重寻址等信息。只有在 Target 设置中选择了 Post-linker，才金可以使用该选项。

在 Output format 下拉框中，为用户提供了多种可以转换的目标格式，本例选择 Plain binary，这是一个二进制格式的可执行文件，可以被烧写到目标板的 Flash 中。

在 Output file name 文本域输入期望生成的输出文件存放的路径，或通过点击 Choose... 按钮从文件对话框中选择输出文件。如果在这个文本域不输入路径名，则生成的二进制文件存放在工程所在的目录下。进行好这些相关的设置后，以后在对工程进行 make 的时候，CodeWarrior IDE 就会在链接完成后调用 fromELF 来处理生成的映像文件。

对于本例的工程而言，到此，就完成了 make 之前的设置工作了。

点击 CodeWarrior IDE 的菜单 Project 下的 make 菜单，就可以对工程进行编译和链接了。编译链接结果如图：

最后在“D:\work\myled\myled_Data\DebugRel”目录下生成 myled.bin，同时还有 myled.axf 文件，它是用于调试的，如图：

4.2 使用 H-JTAG 进行代码调试

请先按照 2.7 一节安装 H-JTAG。

4.2.1 为 H-JTAG 配置 AXD DEBUGGER

(1)运行 AXD Debugger

如图所示打开运行 ADS1.2 软件的调试软件－AXD Debugger：

AXD Debugger 主界面：

(2)设置 AXD Debugger

点菜单 Options->Confiuguer Target…，出现如下窗口：

在该窗口中点击 Add 按钮，跳出选择文件对话框，找到 H-JTAG 安装目录，选择并打开里面的 H-JTAG.dll 文件，如图。

此时会在 Choose Target 窗口中多了一项 H-JTAG，如图，点 OK 返回 AXD Debugger 主界面。

4.2.4 使用 H-JTAG 在 ADS1.2 环境下进行仿真调试

关闭并重新启动 AXD Debugger，点击菜单 File->Load Image，找到您想调试的调试目标文件(*.axf 格式)，如我们前面编译生成的 myled.axf 文件，打开它就自动启动目标映象通过 Jtag 下载至目标板，这时在 AXD Debugger 底部的状态栏会出现下载过程提示，下载完毕就可以进行单步或者全速调试了，调试过程中您可以看到 CPU 各个寄存器的值，也可以设置断点等，详细的用法请参考 ADS 附带的英文说明手册，这基本上和常见的 VisualC++等集成开发环境是类似的。

4.3 编译运行烧写 2440test

2440test 是源自三星的一个非操作系统测试程序，里面集成了很多小型的测试程序，涉及到 GPIO 的配置，中断的编写，常见接口的测试使用等，其中每一部分的测试代码都有很强的独立性，非常适合“ARM 基础性”练习实验。2440test 是基于 ADS1.2 开发环境创建的，它编译出的二进制文件不能下载到 Linux 或者 wince 系统中运行，只能下载到内存指定地址（这里是 0x30000000）运行，也可以烧写到 Nand Flash 中运行。

2440test 的代码仅供学习参考使用，它在某些方面可能是不完善的，我们不对它的原理做任何解释说明，也没有计划去深入完善它。

注意：经友善之臂改进，2440test 目前可以烧写到 64M/128M-1GB Nand Flash 版的 mini2440/micro2440 中，并自动适应运行。

本开发板提供的 2440test 对于开机后 LCD 显示的支持，可以通过\2440test\inc\Option.h 中的 LCD_TYPE 定义来选择（通过去掉前面的注释符”//”可以选择你需要的 LCD 型号）：

//#define LCD_N35
//#define LCD_L80
//#define LCD_T35
#define LCD_X35
//#define LCD_W35
//#define LCD_A70
//#define LCD_VGA1024768

光盘中默认的是 LCD_TYPE_N35

下面是 2440test 的编译、运行和烧写的方法，代码的原理性请用户自行理解。

4.3.1 编译和使用 H-JTAG 调试 2440test

使用开发板附带的 JTAG 小板连接开发板的 JTAG 接口，使用附带的串口线连接开发板的串口，并接上打开电源。这时打开 H-JTAG 软件，它会自动检测到目标板，如图。

把光盘中“非操作系统示例代码”文件夹中的 2440test 目录复制到硬盘的某一个目录(在此为 D:\work)，去掉其只读属性，运行 ADS1.2 集成开发环境，点 **File->Open…**打开 2440test.mcp 文件，如图。

点 Project->Debug 或者按 F5 键开始编译 2440test 项目，编译完毕会自动启动 AXD Debugger 调试器，并把编译好的 2440test.axf 映象通过 JTAG 下载到内存中，如图。

注意：因为 2440test.axf 比较大，因此下载需要等待一段时间。

下载完毕如图所示。

这时点菜单 Execute->Go 或者按 F5 按键，程序将跳转到 Main 函数处，我们就可以进行单步运行调试了。

如果您这时再全速执行程序，可以看到从串口终端打印处如下信息，这和我们之前使用的非操作系统测试程序是一样的。

4.3.2 通过 USB 把 2440test 下载到内存运行

使用 USB 下载运行 2440test 程序不需要并口和 JTAG 板，借助 Supervivi 的“Download & Run”功能就可以了，下面是详细的操作步骤：

(1)连接好开发板电源，串口线，USB 线，并**设置开发板为 NOR Flash 启动系统**，分别打开串口超级终端和 DNW，上电启动开发板。

(2)保证 USB 驱动已经安装好(前面已经详细介绍了 USB 驱动的安装方法)，这时可以看到 DNW 的标题栏显示[USB：OK]，如果没有安装好驱动会显示[USB：x]，如图所示：

(3)点 DNW 菜单 Configuration，设置 USB 下载运行地址为 0x30000000

(4)这时在超级终端的 BIOS 功能菜单中选择功能号[d]，出现 USB 下载等待提示信息：

(5)点击 DNW 程序的“USB Port”→“Transmit”，如图选择刚刚编译出的映象文件(光盘“\images\2440test”目录中有已经编译好的可执行文件)，这样就开始下载了

(6)下载结束后，会自动运行，串口出现如下信息，同时 LCD 上会显示一幅图片：

4.4.3 把 2440test 烧写到 Nand Flash 运行

说明：2440test 可以自动适应 64M/128M Nand Flash mini2440/micro2440，对于同一种 LCD 型号套餐，烧写的文件是一样的。

使用 Supervivi 的功能号[a]可以把 2440test.bin 可执行程序烧写到 Nand Flash 中运行，

步骤如下：

(1)连接好开发板电源，串口线，USB 线，并**设置开发板为 NOR Flash 启动系统**，分别打开串口超级终端和 DNW，上电启动开发板。

(2)保证 USB 驱动已经安装好(前面已经详细介绍了 USB 驱动的安装方法)，这时可以看到 DNW 的标题栏显示[USB：OK]，如果没有安装好驱动会显示[USB：x]，如图所示：

(3)这时在超级终端的 BIOS 功能菜单中选择功能号[a]，出现 USB 下载等待提示信息：

(4)点击 DNW 程序的“USB Port”→“Transmit”，如图选择刚刚编译出的映象文件(光盘“\images\2440test\”目录中有已经编译好的可执行文件)，这样就开始下载了，下载完毕，Supervivi 会把它自动烧写到 Nand Flash 起始块为 0 的地方，即 Block 0 开始处。

烧写完毕，把**开发板设置为 Nand Flash 启动**，重新开机或者复位开发板就可以了。

4.5 uCos2 的编译和烧写

说明：本手册不详细讲述 uCos2 在 2440 上的移植过程及原理，这方面的资料在网上和书店里有很多，虽然都是针对三星 2410 系统的，但同样适用于 2440。

注意：经友善之臂改进，2440test 目前可以烧写到 64M/128M-1GB Nand Flash 版的 mini2440/micro2440 中，并自动适应运行。

uCos2 的源代码位于光盘的“uCos2”目录，为了方便您的使用，我们分别以目录和压缩文件的方式提供其源代码：您可以把源代码目录复制到硬盘中，去掉其只读属性使用(以下示例就是如此)，也可以解压源代码包使用。

本开发板提供的 uCos2 对于开机后 LCD 显示的支持，可以通过 uCos2\S3C2440\includes\option.h 中的 LCD_TYPE 定义来选择（通过去掉前面的注释符”//”可以选择你需要的 LCD 型号）：

```
//#define LCD_N35
//#define LCD_L80
//#define LCD_T35
#define LCD_X35
//#define LCD_W35
//#define LCD_A70
//#define LCD_VGA1024768
```

光盘中默认的是 LCD_TYPE_N35

4.5.1 编译 uCos2

把光盘中“uCos2”目录夹中的“uCos2”文件夹复制到硬盘的某一个目录(在此为 D:\work)，去掉其只读属性，运行 ADS1.2 集成开发环境，点 **File->Open…**打开 uCOS_2440.mcp 文件，如图。

你可能会看到出现如下的一个提示窗口，可以不必理会：

这时点菜单 Project→Make 或者直接按 F7 键，开始编译 uCos2 项目。
编译完毕，在 D:\work\uCos2\uCOS_2440_Data\DebugRel 目录下会生成 2440ucos2.bin

可执行文件，如图。

4.5.2 把 uCos2 下载到内存运行

(1)连接好开发板电源，串口线，USB 线，并**设置开发板为 Nor Flash 启动系统**，分别打开串口超级终端和 DNW，上电启动开发板。

(2)保证 USB 驱动已经安装好(前面已经详细介绍了 USB 驱动的安装方法)，这时可以看到 DNW 的标题栏显示[USB：OK]，如果没有安装好驱动会显示[USB：x]，如图所示：

(3)点 DNW 菜单 Configuration，设置 USB 下载运行地址为 0x30000000

(4)这时在超级终端的 BIOS 功能菜单中选择功能号[d]，出现 USB 下载等待提示信息：

(5)点击 DNW 程序的“USB Port”→“Transmit”，如图选择刚刚编译出的映象文件(光盘“images\uCos2\”目录中有已经编译好的可执行文件)，这样就开始下载了

(6)下载结束后，会自动运行，串口出现如下信息，同时会在 LCD 上显示一幅图片：

4.5.3 把 uCos2 烧写到 Nand Flash 运行

上面的步骤是把可执行文件下载到内存中执行，为了脱离 PC 运行，需要使用 Supervivi 的功能[a](Absolute User Application)把它烧写到 Nand Flash 中，步骤如下：

(1)连接好开发板电源，串口线，USB 线，并**设置开发板为 Nor Flash 启动系统**，分别打开串口超级终端和 DNW，上电启动开发板。

(2)保证 USB 驱动已经安装好(前面已经详细介绍了 USB 驱动的安装方法)，这时可以看到 DNW 的标题栏显示[USB：OK]，如果没有安装好驱动会显示[USB：x]，如图所示：

(3)这时在超级终端的 BIOS 功能菜单中选择功能号[a]，出现 USB 下载等待提示信息：

(4)点击 DNW 程序的“USB Port”→“Transmit”，如图选择刚刚编译出的映象文件(光盘“\images\uCos2\”目录中有已经编译好的可执行文件)，这样就开始下载了，下载完毕，

supervivi 会把它自动烧写到 Nand Flash 起始 block 0 中；把 S2 开关拨至“NAND”一侧，选择从 Nand Flash 启动系统；这时重新开机或者复位，就可以看到和在内存中完全一样的 uCos2 运行界面了。

第五章 建立 Linux 开发环境

本章从在虚拟机/PC 机上安装 **Fedora 9.0** 开始，详细介绍了如何建立 Linux 开发环境。

注意： 自从Linux-2.6.29+Qtopia-2.2.0 开始(**本开发板所配最新内核为Linux-2.6.32.2**)，我们就基于Fedora9 平台做开发，所有的配置和编译脚本也基于此平台，我们没有在其他平台上测试过。如果你对Linux开发很熟悉，相信你会根据错误提示逐步找到原因并解决，它们一般是你选用的平台缺少了某些库文件或者工具等原因造成的；否则，我们建议初学者使用和我们一致的平台，即Fedora 9(全称为：Fedora-9-i386-DVD.iso)，你可以在我们网站下载(http://www.arm123.com.cn/iso/Fedora-9-i386-DVD.iso ，不保证长期有效)，也可以在其他地方获取，它们都是一样的，安装时请务必参考我们手册提供的步骤，这是我们经过严格测试的，以免遗漏一些开发时所需要的组件。

Linux 的发行版本众多，我们无法为此一一编写文档解释安装方法，请谅解。

我们为什么选择 Fedora 9:

根据我们的测试，Fedora 9 经过比较简单的安装和设置，依然可以使用 root 用户登录(大多数开发均需要此用户权限)，Fedora 10 及其以后的版本则需要经过稍微复杂的设置才能使用 root，这不利于不了解 Linux 的初学者，Fedora 8 及其以前的版本则相对老了一些。并且按照我们手册提供的步骤安装 Fedora 9，可以比较完美配合我们提供的开发软件包，不再需要其他补丁之类的繁琐设置(ubuntu 就需要经常这样更新设置)，因此我们认为 Fedora 9 是最适合初学者的开发平台。

5.1 图解安装 Fedora 9.0

用户可以在此下载 Fedora 9.0 的 DVD 光盘映象文件：

http://www.arm123.com.cn/iso/Fedora-9-i386-DVD.iso

(如果下载地址有变，将不再另外通知)

Step1： 将的安装光盘放到光驱中，将 BIOS 改为从光盘启动，启动后系统将会出现如下界面，按回车继续。

Step2：然后进入下一步，检查安装盘，一般不需要检测，所以选择了 Skip（跳过）

Step3：过一会儿就进入安装图形化画面，点击 Next 即可.

Step4：选择安装过程用什麽语言，这里选择的是英文

Step5：选键盘，我们一般选美式键盘即可

Step6：开始设置网络

点“Edit”按钮，不要设置为 DHCP，我们一般使用静态的 IP，对照下面进行填写，分别输入 IP 和子网掩码

点“OK”返回，开始设置机器名和网关以及 DNS 等

Step7：设置时区，如果你不使用虚拟机安装，“System clock uses UTC”选项可以去掉，如图

Step8：设置 root 用户密码，必须是 6 位数以上。

Step9：设置分区，一般选择默认即可，注意要备份好硬盘数据

点“Next”会出现警告信息，告诉你继续执行会格式化分区中的所有数据，一般我们在 Vmware 虚拟机中使用，因此可以选“Write changes to disk”，之后开始进行格式化操作。

这是格式化的进程图：

Step11：选择安装类型，选择如图，点“Next”开始定制。

Step12：在 Servers 项中，选择如图

Step13：开始安装系统，此过程时间会比较长，请耐心等待。

Step14：安装完毕，如图

Step15：接上一步，按“Reboot“按钮重启系统，出现第一次使用的界面，如图。

Step16：一些授权信息，不必理会，继续下一步

Step17：创建用户，在此我们不需要创建任何新的用户，点“Forward”继续

这时会出现提示信息让你确认，点“Continue”继续下一步。

Step18：设置日期和时间，不必理会，继续下一步

Step19：列出了本机的一些硬件信息，采用缺省设置，点“Finish”

出现提示信息，如图选择，进行下一步

Step20：出现登录界面，我们要以 root 用户进行登录，因此先输入 root

再输入刚才设定的密码

登录后会出现一个提示，以后如果你以 root 用户登录，每次都会出现这个提示，每次均点“Continue”即可。

这是登录后的界面，它和 Windows 或者 Ubuntu 是十分类似的。

5.2 常用设置和服务

现在介绍一些开发中常用的设置和服务，其实还有很多 Linux 命令也是需要掌握的，用户可以自己找本 Linux 的书籍看看。

5.2.1 添加新用户

为了方便开发，我们通常创建一个普通权限的用户，步骤如下。

Step1：如图打开用户和组管理器：

Step2：出现用户管理窗口

Step3：点工具栏的“Add User”按钮，添加新用户，并设置密码：

点“OK”返回，可以看到已经增加了 plg 用户，同时/home 目录下也增加了 plg 用户目录，如图：

点 Add User 按钮，出现添加新用户窗口，按提示操作就可以了。

5.2.2 访问 Windows 系统中的文件

无论你使用的是虚拟机还是真实的 Fedora9 系统，都可以很方便的访问 Windows 中的共享文件，前提是两个系统之间的网络是互通的。

提示：要在虚拟机中使用网络，最简单的方式是设置“Guest”为“Bridges”方式的网络连接，如图：

访问 Windows 系统中共享文件的步骤如下：

Step1：在 Windows 中设置共享文件夹“share_f9”(示例)

Step2：在 Fedora9 系统中如图操作：

打开如图窗口：

在 Service type 列表中选择 Windows share，如图：

输入所要共享 Windows 主机的 IP 地址和共享文件夹的名字：

点“Connect”，会出现如下提示窗口：

不必理会，直接点“Connect”即可，就可以看到 Windows 共享文件中的内容了，在此你可以像操作其他目录一样来使用它。

如果你想在命令行使用这个目录，可以这样操作：
说明：在控制台下，TAB 键是一个很好使用的小技巧。

要想断开共享目录，只要在桌面的共享文件夹上用右键如图操作就可以了：

5.3 建立交叉编译环境

在 Linux 平台下，要为开发板编译内核，图形界面 Qtopia，bootloader，还有其他一些应用程序，均需要交叉编译工具链。

之前的系统，要使用不同的编译器版本才能正常编译各个部分，因此要在开发过程不断切换设置，这十分不利于初学者使用，也降低了开发的效率；自从 Linux-2.6.29 开始(本开发板所配内核已为最新的 Linux-2.6.32.2)，我们把交叉编译器统一为 arm-linux-gcc-4.4.3，下面是它的安装设置步骤。

Step1：将光盘目录 linux\中的 arm-linux-gcc-4.4.3.tgz 复制到某个目录下如 tmp\，然后进入到该目录，执行解压命令：

#cd /tmp

#tar xvzf arm-linux-gcc-4.4.3.tgz　–C /

注意：C 后面有个空格，并且 C 是大写的，它是英文单词“Change”的第一个字母，在此是改变目录的意思。

执行该命令，将把 arm-linux-gcc 安装到**/usr/loca/arm/4.4.3** 目录。

解压过程如图所示：

Step2：把编译器路径加入系统环境变量，运行命令

#gedit /root/.bashrc

编辑/root/.bashrc 文件，在最后一行添加：

export PATH=$PATH: /opt/FriendlyARM/toolschain/4.4.3/bin

如图，保存退出。

重新登录系统(不必重启机器，**开始->logout** 即可)，使以上设置生效，在命令行输入 arm-linux-gcc –v，会出现如下信息，这说明交叉编译环境已经成功安装。

```
root@tom:~
File  Edit  View  Terminal  Tabs  Help
[root@tom ~]# arm-linux-gcc -v
Using built-in specs.
Target: arm-none-linux-gnueabi
Configured with: /opt/FriendlyARM/mini2440/build-toolschain/working/src/gcc-4.4.3/configure --build=
i386-build_redhat-linux-gnu --host=i386-build_redhat-linux-gnu --target=arm-none-linux-gnueabi --pre
fix=/opt/FriendlyARM/toolschain/4.4.3 --with-sysroot=/opt/FriendlyARM/toolschain/4.4.3/arm-none-linu
x-gnueabi//sys-root --enable-languages=c,c++ --disable-multilib --with-arch=armv4t --with-cpu=arm920
t --with-tune=arm920t --with-float=soft --with-pkgversion=ctng-1.6.1 --disable-sjlj-exceptions --ena
ble-__cxa_atexit --with-gmp=/opt/FriendlyARM/toolschain/4.4.3 --with-mpfr=/opt/FriendlyARM/toolschai
n/4.4.3 --with-ppl=/opt/FriendlyARM/toolschain/4.4.3 --with-cloog=/opt/FriendlyARM/toolschain/4.4.3
--with-mpc=/opt/FriendlyARM/toolschain/4.4.3 --with-local-prefix=/opt/FriendlyARM/toolschain/4.4.3/a
rm-none-linux-gnueabi//sys-root --disable-nls --enable-threads=posix --enable-symvers=gnu --enable-c
99 --enable-long-long --enable-target-optspace
Thread model: posix
gcc version 4.4.3 (ctng-1.6.1)
[root@tom ~]#
```

5.4 解压安装源代码及其他工具

本小节将解压安装开发学习过程所用到的全部源代码以及其他一些小工具，这包括：

- Linux 内核源代码
- 嵌入式图形界面 Qropia-2.2.0 源代码(分为 x86 和 arm 平台两个版本)
- 嵌入式图形界面 QtE-4.6.3 源代码(ARM 版本)
- busybox-1.13 源代码
- Linux 编程示例源代码(均为友善之臂自主开发并开放)
- 用以启动 Linux 的 bootloader 之 vboot
- 其他开源的 bootloader(适用于 Linux)
- 其他开源软件源代码，如 boa(web server), madplay(一个命令行 mp3 播放器)
- 目标文件系统目录
- 目标文件系统映象制作工具 **mkyaffsimage**
- 图形界面的 Linux logo 制作工具 **logomaker**

注意：所有的源代码和工具都是通过解压方式安装的，所有的源代码均使用统一的编译器 arm-linux-gcc-4.3.2 编译(见上一节)

下面是详细的解压安装过程，并有简要的介绍。

5.4.1 解压安装源代码

首先创建工作目录/opt/FriendlyARM/mini2440

在命令行执行 mkdir –p /opt/FriendlyARM/mini2440 ，如图，后面步骤的所有源代码

都会解压安装到此目录中：

(1)准备好 Linux 源代码包

在 Fedora9 系统中/tmp 目录中创建一个临时目录**/tmp/linux**

#mkdir /tmp/linux

把光盘中 linux 目录中的所有文件都复制到**/tmp/linux** 目录中

说明：这样做是为了统一下面的操作步骤，其实你可以使用其他目录，也可以直接从光盘解压安装。

(2)解压安装 Linux 内核源代码

在工作目录/opt/FriendlyARM/mini2440 中执行：

#cd /opt/FriendlyARM/mini2440

#tar xvzf /tmp/linux/linux-2.6.32.2-mini2440-20100106.tar.gz

将创建生成 linux-2.6.32.2 目录，里面包含了完整的 Linux-2.6.32.2 内核源代码

说明：20100106 是我们的发行更新日期标志，请以光盘中实际日期尾缀为准。

(3)解压安装嵌入式图形系统 qtopia 源代码

在工作目录/opt/FriendlyARM/mini2440 中执行：

#cd /opt/FriendlyARM/mini2440

#tar xvzf /tmp/linux/x86-qtopia.tgz

#tar xvzf /tmp/linux/arm-qtopia.tgz

将创建 x86-qtopia 和 arm-qtopia 两个目录，并内含相应的全部源代码。

说明：20100108 是我们的发行更新日期标志，请以光盘中实际日期尾缀为准。和之前的 qtopia 源代码包不同，现在的源代码包不再区分 mouse(鼠标支持)和 tp(触摸屏支持)，此系统可以支持二者共存，因此只有一个源代码包，其中也包含了嵌入式浏览器 konquor 的源代码。

另外，为了方便用户学习开发使用，此源代码包相比 Qt 公司的原始版本已经打过补丁，并做了诸多改进，它们都是源代码方式，我们不再一一赘述，感兴趣者可自行比较。

(4)解压安装嵌入式图形系统 QtE-4.6.1 源代码

在工作目录/opt/FriendlyARM/mini2440 中执行：

#cd /opt/FriendlyARM/mini2440

#tar xvzf /tmp/linux/arm-qte-4.6.3-20100802.tar.gz

将创建 arm-qte-4.6.3 目录，并内含相应的全部源代码。

说明： arm-qte 压缩包后面或许会有日期尾缀，它是我们的发行或更新日期，请以光盘中实际日期尾缀为准。

(5)解压安装 busybox 源代码

Busybox是一个轻型的linux命令工具集，在此使用的是busybox-1.13.3 版本。用户可以从其官方网站下载最新版本(http://www.busybox.net)。

在工作目录/opt/FriendlyARM/mini2440 中执行：

#cd /opt/FriendlyARM/mini2440

#tar xvzf /tmp/linux/busybox-1.13.3-mini2440.tgz

将创建 busybox-1.13.3 目录，内含相应版本的全部源代码。

说明：为了方便用户编译使用，我们做了一个缺省的配置文件 fa.config。

(6)解压安装 Linux 示例程序

在工作目录/opt/FriendlyARM/mini2440 中执行：

#cd /opt/FriendlyARM/mini2440

#tar xvzf /tmp/linux/examples-20100108.tgz

将创建 examples 目录，并包含初学 linux 编程代码示例。

说明：20100108 是我们的发行更新日期标志，请以光盘中实际日期尾缀为准。examples 目录中的代码均为友善之臂自主开发，并全部以源代码方式提供，它们都是一些基于命令行的小程序。

(7)解压安装 vboot 源代码

为了实现自动适应支持 64M/128M mini2440/micro2440，我们专门为 Linux 系统设计了一个简易的 bootloader：vboot，而不再使用以前的 vivi。

在工作目录/opt/FriendlyARM/mini2440 中执行：

#cd /opt/FriendlyARM/mini2440

#tar xvzf /tmp/linux/vboot-src-20100106.tar.gz

将创建 vboot 目录，里面包含该 bootloader 的源代码和 Makefile 文件。

说明：20100106 是我们的发行更新日期标志，请以光盘中实际日期尾缀为准

(8)解压安装其他其他开源 bootloader 源代码

除了 vboot，本开发板还提供了另外三种开源的 Bootloader(详见本手册第八章节)，其中 vivi 和 u-boot 是在 Linux 平台下设计编译的。

在工作目录/opt/FriendlyARM/mini2440 中执行：

#cd /opt/FriendlyARM/mini2440

#tar xvzf /tmp/linux/bootloader.tgz

将创建 bootloader 目录，里面包含 vivi 和 u-boot 两种 bootloader 的源代码。

说明：此处的 vivi 仅适用于 64M Nand Flash 的 mini2440/micro2440 板，u-boot 是由网友提供的，我们并没有使用过，对其功能和性能均不了解。

5.4.2 解压创建目标文件系统

在工作目录/opt/FriendlyARM/mini2440 中执行：

#cd /opt/FriendlyARM/mini2440

#tar xvzf /tmp/linux/ rootfs_qtopia_qt4-20100816.tar.gz

将创建 rootfs_qtopia_qt4 目录，该目录和目标板上使用的文件系统内容是完全一致的。

说明：20100816 是我们的发行更新日期标志，请以光盘中实际日期尾缀为准。以前的目标文件系统有 4 个：root_default, root_nfs, root_qtopia_tp, root_qtopia_mouse，它们分别是为实现不同的启动方式和功能外设而创建的，现在我们把它统一为一种，它包含了完整的 qtopia 测试系统，最新的 busybox，还有常用的命令行工具等，和之前的相比，它具有如下特性：

- 自动识别 NFS 启动或本地启动
- 可支持 USB 鼠标和触摸屏共存
- 自动识别所接的输出显示模块是否接了触摸屏，以判断在第一次开机使用时是否要进行校正。如果没有连接，会自动进入系统，使用鼠标即可；否则会先校正触摸屏。
- 自动识别普通或者高速 SD 卡(最大可支持 32G)和优盘
- 包含双图形系统 Qtopia-2.2.0 和 QtE-4.6.1

5.4.3 解压安装必要实用工具

(1)目标文件系统映象制作工具 mkyaffs2image

要把上一步中的 rootfs_qtopia_qt4 目录烧写入目标板中使用，就需要使用相应的 mkyaffs2image 工具了，它是一个命令行的程序，使用它可以把主机上的目标文件系统目录制作成一个映象文件，以烧写到开发板中。

针对 64M 或 128M/256M/512M/1GB 的 mini2440/mcro2440，分别有 2 套制作工具：mkyaffs2image 和 mkyaffs2image-128M。其中 mkyaffs2image 是制作适用于 64M 版本文件系统映象的工具，它沿用了以前的名字；mkyaffs2image-128M 是制作适用于 128M/256M/512M/1GB 版本文件系统映象的工具，为了便于区分，我们把它命名为此。

在工作目录/opt/FriendlyARM/mini2440 中执行：

#cd /opt/FriendlyARM/mini2440

#tar xvzf /tmp/linux/mkyaffs2image.tgz –C /

注意：C 是大写的，C 后面有个空格，C 是改变解压安装目录的意思

说明：以前的内核系统支持的是 yaffs 文件系统，现在使用的是 yaffs2 文件系统，因此需要不同的制作工具，我们在此把它称为 mkyaffs2image，按照上面的命令解压后它会被安装到/usr/sbin 目录下，并产生 2 个文件：mkyaffs2image 和 mkyaffs2image-128M。

(2)解压安装 LogoMaker

在工作目录/opt/FriendlyARM/mini2440 中执行：

#cd /opt/FriendlyARM/mini2440

#tar xvzf /tmp/linux/logomaker.tgz –C /

注意：C 是大写的，C 后面有个空格，C 是改变解压安装目录的意思

说明：LogoMaker 是友善之臂开发的一个 linux logo 简易制作工具，网上有很多资料介绍如何使用命令行的工具把 bmp, jpg, png 等格式的图片转换为 linux logo 文件，在此我们设计了一个图形化的版本，它是基于 Fedora9 开发。

执行以上命令，LogoMaker 将会被安装到/usr/sbin 目录下，它只有一个文件，安装完之后在命令行输入 logomaker 可出现如下界面，在后面的章节我们会介绍它的使用方法：

5.5 配置网络文件系统 NFS 服务

如果您已经按照以上章节介绍的方法完全安装好了 Fedora 9，则 NFS 相关软件都已经缺省安装好了，请按照以下步骤建立和配置 NFS 服务。

5.5.1 设置共享目录

注意：要使用共享目录，必须先按照 5.4.2 章节解压安装好 root_qtopia 目标板文件系统包。

(1)设置共享目录

运行命令

#gedit /etc/exports

编辑 nfs 服务的配置文件(*注意：第一次打开时该文件是空的*)，添加以下内容：

/opt/FriendlyARM/mini2440/rootfs_qtopia_qt4　　*(rw,sync,no_root_squash)

其中：

/opt/FriendlyARM/mini2440/rootfs_qtopia_qt4 表示 nfs 共享目录，它可以作为开发板的根文件系统通过 nfs 挂接；

* 表示所有的客户机都可以挂接此目录

rw 表示挂接此目录的客户机对该目录有读写的权力

no_root_squash 表示允许挂接此目录的客户机享有该主机的 root 身份

5.5.2 和启动 NFS 服务

可以通过命令行和图形界面两种方式启动 NFS 服务，我们建立 NFS 服务的目的是通过网络对外提供目录共享服务，但默认安装的 Fedora 系统开启了防火墙，这会导致 NFS 服务无法正常使用。因此先关闭防火墙，在命令行输入“lokkit”命令，打开防火期设置界面：

选择其中(*)Disabled，然后选择“OK”退出，这样就永久的关闭了防火墙。

下面是启动 NFS 服务的方法和步骤：

(1)通过命令启动和停止 nfs 服务

在命令行下运行：

#/etc/init.d/nfs start

这将启动 nfs 服务，可以输入以下命令检验 nfs 该服务是否启动。

mount -t nfs localhost: **/opt/FriendlyARM/mini2440/rootfs_qtopia_qt4**　/mnt/

如果没有出现错误信息，您将可以浏览到 **/mnt** 目录中的内容和 **/opt/FriendlyARM/mini2440/rootfs_qtopia_qt4** 是一致的。

使用这个命令可以停止 nfs 服务：

#/etc/init.d/nfs stop

(2)通过图形界面启动 NFS 服务

为了在每次开机时系统都自动启动该服务，可以输入

 # serviceconf

打开系统服务配置窗口，在左侧一栏找到 nfs 服务选项框，并选中它，然后点工具栏的“Enable”启动它，如图。

5.5.3 通过 NFS 启动系统

当 NFS 服务设置好并启动后，我们就可以把 NFS 作为根文件系统来启动开发板了。通过使用 NFS 作为根文件系统，开发板的“硬盘”就可以变得很大，因为您使用的是主机的

硬盘，这是使用 Linux 作为开发经常使用的方法，

设置目标板启动模式为 Nand Flash 启动，连接好电源，串口线，网线；打开串口终端，**在开机或者复位的时候按下开发板上的 K1-K6 任意按键**，这样我们就进入了 vivi 模式，输入以下命令：

Supervivi> param set linux_cmd_line "console=ttySAC0 root=/dev/nfs nfsroot=192.168.1.111:/opt/FriendlyARM/mini2440/rootfs_qtopia_qt4 ip=192.168.1.70:192.168.1.111:192.168.1.111:255.255.255.0:sbc2440.arm9.net:eth0:off"

其中 ，param set linux_cmd_line 是设置启动 linux 时的命令参数。其各参数的含义如下：

nfsroot 是自己开发主机的 IP 地址。

“ip=”后面：

第一项(192.168.1.70)是目标板的临时 IP(注意不要和局域网内其他 IP 冲突)；

第二项(192.168.1.111)是开发主机的 IP；

第三项(192.168.1.111)是目标板上网关(GW)的设置；

第四项(255.255.255.0)是子网掩码；

第五项是开发主机的名字(一般无关紧要，可随便填写)

eth0 是网卡设备的名称。

由于该命令比较长，容易输入错误，我们已经把它写入了光盘的 nfs.txt 文件中，这样您直接复制过来就可以了。

然后输入 boot，按回车就可以通过 nfs 启动系统了。

ttyS0 - 超级终端
文件(F) 编辑(E) 查看(V) 呼叫(C) 传送(T) 帮助(H)
#0: S3C24XX_UDA134X (UDA134X)
TCP cubic registered
RPC: Registered udp transport module.
RPC: Registered tcp transport module.
s3c2410-rtc s3c2410-rtc: setting system clock to 2103-10-01 10:38:28 UTC (
8308)
eth0: link down
IP-Config: Complete:
device=eth0, addr=192.168.1.70, mask=255.255.255.0, gw=192.168.1.111,
host=sbc2440, domain=, nis-domain=arm9.net,
bootserver=192.168.1.111, rootserver=192.168.1.111, rootpath=
Looking up port of RPC 100003/2 on 192.168.1.111
eth0: link up, 100Mbps, full-duplex, lpa 0x45E1
Looking up port of RPC 100005/1 on 192.168.1.111
VFS: Mounted root (nfs filesystem) on device 0:11.
Freeing init memory: 124K
hwclock: settimeofday() failed: Invalid argument
[25/Aug/1967:04:10:24 +0000] boa: server version Boa/0.94.13
[25/Aug/1967:04:10:24 +0000] boa: server built Mar 26 2009 at 15:28:42.
[25/Aug/1967:04:10:24 +0000] boa: starting server pid=482, port 80
Try to bring eth0 interface up.....NFS root ...Done
Please press Enter to activate this console.
已连接 0:48:41 ANSIW
115200 8-N-1
SCROLL
CAPS
NUM
捕
打印

第六章 定制 Linux 内核及制作文件系统

Linux 内核看似很庞大，其实初学者以及致力于应用开发的用户，不必学习之初就一头钻进浩如烟海的内核代码中问剑；但熟练配置内核的一些常用选项，并编译出来下载到开发板中运行试用，是你迈向 Linux 系统顶峰的必行之路。

本小节中所有步骤内容均不涉及代码编写，但请初学者不要忽视它们；学习 Linux 不像单片机系统，你不必从“零”代码开始，一切先从学会配置、编译、下载运行开始。

下面操作均基于 Fedora 9 平台操作。

注意：本开发板所配最新内核版本为 Linux-2.6.32.2

6.1 使用缺省配置文件配置和编译内核

为了方便用户能够编译出和光盘烧写文件完全一致的内核，我们针对不同的 LCD 输出分别做了相应的内核配置文件：

config_mini2440_w35 – 适用于横屏 3.5” LCD(面板标识为 W35)的内核配置文件
config_mini2440_x35 – 适用于 Sony 3.5” LCD 的内核配置文件
config_mini2440_t35 – 适用于统宝 3.5” LCD 的内核配置文件
config_mini2440_l80 – 适用于 Sharp 8” LCD(或兼容)的内核配置文件
config_mini2440_n35 – 适用于 NEC3.5” LCD 的内核配置文件
config_mini2440_n43 – 适用于 NEC4.3” LCD 的内核配置文件
config_mini2440_a70 – 适用于群创 7” LCD 的内核配置文件
config_mini2440_vga1024x768 – 适用于 VGA 显示输出(分辨率 1024x768(模块的内核配置文件

在内核目录中可以使用 ls 命令看到它们的存在：

```
root@tom:/opt/FriendlyARM/mini2440/linux-2.6.32.2
File  Edit  View  Terminal  Tabs  Help
linux-2.6.32.2/drivers/firewire/core-topology.c
linux-2.6.32.2/drivers/firewire/core-cdev.c
linux-2.6.32.2/drivers/firewire/Kconfig
linux-2.6.32.2/drivers/firewire/sbp2.c
linux-2.6.32.2/drivers/firewire/core.h
linux-2.6.32.2/drivers/firewire/core-iso.c
linux-2.6.32.2/drivers/firewire/ohci.c
linux-2.6.32.2/drivers/firewire/core-device.c
linux-2.6.32.2/drivers/clocksource/
linux-2.6.32.2/drivers/clocksource/sh_tmu.c
linux-2.6.32.2/drivers/clocksource/Makefile
linux-2.6.32.2/drivers/clocksource/sh_cmt.c
linux-2.6.32.2/drivers/clocksource/scx200_hrt.c
linux-2.6.32.2/drivers/clocksource/acpi_pm.c
linux-2.6.32.2/drivers/clocksource/cyclone.c
linux-2.6.32.2/drivers/clocksource/sh_mtu2.c
linux-2.6.32.2/drivers/clocksource/tcb_clksrc.c
[root@tom mini2440]#
[root@tom mini2440]#
[root@tom mini2440]# ls
linux-2.6.32.2
[root@tom mini2440]# cd linux-2.6.32.2/
[root@tom linux-2.6.32.2]# ls
arch                 config_mini2440_t35          Documentation  init    MAINTAINERS  REPORTING-
block                config_mini2440_vga1024x768  drivers        ipc     Makefile     samples
config_mini2440_a70  COPYING                      firmware       Kbuild  mm           scripts
config_mini2440_l80  CREDITS                      fs             kernel  net          security
config_mini2440_n35  crypto                       include        lib     README       sound
[root@tom linux-2.6.32.2]#
```

执行以下命令来使用缺省配置文件 config_w35

#cp config_mini2440_w35 .config　　；注意：**w35** 后面有个空格，然后有个“**.**”开头的 **config**

然后执行“**make menuconfig**”，出现配置内核界面：

```
root@tom:/opt/FriendlyARM/mini2440/linux-2.6.32.2
File  Edit  View  Terminal  Tabs  Help
.config - Linux Kernel v2.6.32.2 Configuration

                    Linux Kernel Configuration
  Arrow keys navigate the menu.  <Enter> selects submenus --->.  Highlighted letters are
  hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes features.  Press
  <Esc><Esc> to exit, <?> for Help, </> for Search.  Legend: [*] built-in  [ ] excluded
  <M> module  < > module capable

        General setup  --->
        [*] Enable loadable module support  --->
        -*- Enable the block layer  --->
            System Type  --->
            Bus support  --->
            Kernel Features  --->
            Boot options  --->
            CPU Power Management  --->
            Floating point emulation  --->
            Userspace binary formats  --->
            Power management options  --->
        [*] Networking support  --->
            Device Drivers  --->
            File systems  --->
            Kernel hacking  --->
        v(+)

                <Select>    < Exit >    < Help >
```

这时不用做任何更改，在主菜单里选择**<Exit>**退出，这样做是为了生成相应配置的头文件。

输入以下命令，开始编译内核：

#make zImage

```
root@tom:/opt/FriendlyARM/mini2440/linux-2.6.32.2
File  Edit  View  Terminal  Tabs  Help
make[1]: `include/asm-arm/mach-types.h' is up to date.
  CHK     include/linux/utsrelease.h
  SYMLINK include/asm -> include/asm-arm
  CC      kernel/bounds.s
  GEN     include/linux/bounds.h
  CC      arch/arm/kernel/asm-offsets.s
  GEN     include/asm/asm-offsets.h
  CALL    scripts/checksyscalls.sh
  CC      scripts/mod/empty.o
  HOSTCC  scripts/mod/mk_elfconfig
  MKELF   scripts/mod/elfconfig.h
  HOSTCC  scripts/mod/file2alias.o
  HOSTCC  scripts/mod/modpost.o
  HOSTCC  scripts/mod/sumversion.o
  HOSTLD  scripts/mod/modpost
  HOSTCC  scripts/kallsyms
  HOSTCC  scripts/pnmtologo
  HOSTCC  scripts/conmakehash
  CC      init/main.o
  CHK     include/linux/compile.h
  UPD     include/linux/compile.h
  CC      init/version.o
  CC      init/do_mounts.o
  LD      init/mounts.o
  CC      init/noinitramfs.o
  CC      init/calibrate.o
  LD      init/built-in.o
  LD      usr/built-in.o
```

编译结束后，会在 **arch/arm/boot** 目录下生成 linux 内核映象文件：**zImage**

```
root@tom:/opt/FriendlyARM/mini2440/linux-2.6.32.2
File  Edit  View  Terminal  Tabs  Help
  MODPOST vmlinux.o
  GEN     .version
  CHK     include/linux/compile.h
  UPD     include/linux/compile.h
  CC      init/version.o
  LD      init/built-in.o
  LD      .tmp_vmlinux1
  KSYM    .tmp_kallsyms1.S
  AS      .tmp_kallsyms1.o
  LD      .tmp_vmlinux2
  KSYM    .tmp_kallsyms2.S
  AS      .tmp_kallsyms2.o
  LD      vmlinux
  SYSMAP  System.map
  SYSMAP  .tmp_System.map
  OBJCOPY arch/arm/boot/Image
  Kernel: arch/arm/boot/Image is ready
  AS      arch/arm/boot/compressed/head.o
  GZIP    arch/arm/boot/compressed/piggy.gz
  AS      arch/arm/boot/compressed/piggy.o
  CC      arch/arm/boot/compressed/misc.o
  LD      arch/arm/boot/compressed/vmlinux
  OBJCOPY arch/arm/boot/zImage
  Kernel: arch/arm/boot/zImage is ready
[root@tom linux-2.6.32.2]# ls arch/arm/boot/zImage
arch/arm/boot/zImage
[root@tom linux-2.6.32.2]# ls arch/arm/boot/zImage -l
-rwxr-xr-x 1 root root 2517028 2010-01-08 12:53 arch/arm/boot/zImage
[root@tom linux-2.6.32.2]#
```

您可以使用第三章节介绍的方法把 zImage 下载到开发板测试。

6.2 各个驱动程序源代码位置

说明：解压内核源代码(linux-2.6.32.2 开头的 tgz 压缩文件)可以找到如下驱动，mini2440 提供基于 linux-2.6.32.2 内核 100%完全可以使用的驱动源代码，绝无库文件，敬请放心使用。

(1)DM9000 网卡驱动
Linux-2.6.32.2/drivers/net/dm9000.c
(2)串口(包括三个串口驱动 0,1,2，对应设备名/dev/ttySAC0,1,2)
Linux-2.6.32.2/drivers/serial/s3c2440.c
(3)实时时钟 RTC 驱动
Linux-2.6.32.2/drivers/rtc/rtc-s3c.c
(4)LED 驱动
Linux-2.6.32.2/drivers/char/mini2440_leds.c
(5)按键驱动
Linux-2.6.32.2/drivers/char/mini2440_buttons.c
(6)触摸屏驱动
Linux-2.6.32.2/drivers/input/touchscreen/s3c2410_ts.c
(7)yaffs2 文件系统源代码目录
Linux-2.6.32.2/fs/yaffs2
(8)USB 鼠标、键盘源代码
Linux-2.6.32.2/drivers/usb/hid
(9)SD/MMC 卡驱动源代码目录(支持高速最大容量 32G SD 卡)
Linux-2.6.32.2/drivers/mmc
(10)Nand Flash 驱动
Linux-2.6.32.2/drivers/mtd/nand
(11)UDA1341 音频驱动目录
Linux-2.6.32.2/sound/soc/s3c24xx
(12)LCD 驱动
Linux-2.6.32.2/drivers/video/s3c2410fb.c
(13)优盘支持驱动
Linux-2.6.32.2/drivers/usb/storage
(14)万能 USB 摄像头驱动
Linux-2.6.32.2/drivers/media/video/gspca
(15)I2C-EEPROM 驱动
inux-2.6.32.2/drivers/i2c
(16)背光驱动
Linux-2.6.32.2/drivers/video/mini2440_backlight.c
(17)PWM 控制蜂鸣器驱动
Linux-2.6.32.2/drivers/char/mini2440_pwm.c
(18)看门狗驱动
Linux-2.6.32.2/drivers/watchdog/s3c2410_wdt.c

(19)AD 转换驱动
Linux-2.6.32.2/drivers/char/mini2440_ad.c
(20)CMOS 摄像头驱动
Linux-2.6.32.2/drivers/media/video/s3c2440camif.c
(21)USB 无线网卡驱动(型号：TL-WN321G+)
Linux-2.6.32.2/drivers/net/wireless/rt2x00
(22)USB 转串口驱动
Linux-2.6.32.2/drivers/usb/serial/pl2302.c

6.3 手工定制 Linux 内核

上面我们使用缺省文件配置和编译了内核，其实 Linux 内核的配置选项有很多，下面我们就常见的一些选项分别予以图解，帮助你尽快熟悉内核配置，以便定制自己需要的内核。

注意：为了方便你尽快熟悉各个内核配置选项，请务必按照 6.1 介绍的方法先加载一个缺省的配置文件如 config_mini2440_w35，否则下面的选项有可能是不会出现的。

运行 make menuconfig 后，进入内核配置主菜单

6.3.1 配置 CPU 平台选项

在主菜单里面，选择 System Type，按回车进入

可以看到，系统大部分使用了标注了 S3C2410 的选项，这是因为 S3C2410 和 S3C2440 的很多寄存器地址等地址和设置是完全相同的。

如果您要选择板级选项，使用上下方向控制键一直找到 S3C2440 机器平台选项，可以进入 S3C2400 Machines 子菜单

可以看到里面有很多常见的使用 S3C2440 的目标板平台选项，在此选“FriendlyARM Mini2440 development board”，如图

它们分别对应于 **arch/arm/mach-s3c2440/mach-*** 开头的文件，在此对应于 **mach-mini2440.c**。另外，在这个文件中，还会用到一个机器码 **MACH_TYPE**，该机器码的定义文件为 **arch/arm/tools/mach-types**，我们开发板的机器码为 **1999**，它还对应于 vivi 源代码中 **include/platform/smdk2440.h** 文件的 **MACH_TYPE**

6.3.2 配置各个尺寸的 LCD 驱动以及背光控制支持

在主菜单里面，选择 Device Drivers，按回车进入，并找到如图选项，按回车进入：

找到如图选项，再按回车进入

出现类似如图界面，并找到如图选项，选中如图 Backlight(背光控制)

再选中 LCD select，按回车进入，如图，可以看到我们加载的默认配置 config_mini2440_t35 在此选择 统宝 3.5” LCD(3.5 inch 240x320 Toppoly LCD)，你还可以根据需要改为其他型号的 LCD

选择完毕，一直按照下方的提示返回到 Device Drivers 配置菜单。

6.3.3 配置触摸屏

注意：如果你选择了 VGA1024x768 显示输出模块，是不需要配置此项的。

在 Device Drivers 菜单里面，选择 Input device support，按回车进入

找到并选择 Touchscreens 选项，按回车进入，如图：

然后如图选择

选择完毕，按<Exit>一直返回 Device Drivers 菜单。

6.3.4 配置 USB 鼠标和键盘

在 Device Drivers 菜单里面，找到如图选项，并选择进入

选择如图“*”号所指示的选项

这样就选择配置了 USB 键盘和鼠标，然后选择<Exit>返回 Deice Drivers 菜单。

6.3.5 如配置优盘的支持

因为优盘用到了 SCSI 命令，所以我们先增加 SCSI 支持。
在 Device Drivers 菜单里面，选择 SCSI device support，按回车进入

在出现的次菜单中，选择如图

返回 Device Drivers 菜单，再选择 USB support，按回车进入 USB support 菜单找到并选中

<*> USB Mass Storage support

然后选择<Exit>返回 Device Drivers 菜单

6.3.6 配置万能驱动 USB 摄像头

在 Device Drivers 菜单里面，选择 Multimedia devices，回车进入

选择如图“*”号选项，并选择 Video capture adapters 进入

出现如图菜单，找到如图选项并进入

出现如图菜单，选择如图“*”号选项，再选 GSPCA based webcams 进入

GSPCA 是一个法国程序员在业余时间制作的一个万能 USB 摄像头驱动程序，在此你可以选择所有类型 USB 摄像头的支持，如图

需要注意的是：虽然这里选择了众多型号的摄像头驱动，但每个型号的 Video 输出格式并不完全相同，这需要在高层应用中根据实际情况分别做处理，才能正常使用这些驱动。

一直选择<Exit>返回 Device Drivers 菜单，再选择<Exit>返回到主菜单。

6.3.7 配置 CMOS 摄像头驱动

本开发板配用的 CMOS 摄像头模块 CAM130，其内部使用的 OV9650 芯片，因此我们需要为此配置驱动程序，如下步骤：

在 Device Drivers 菜单里面，选择 Multimedia devices，回车进入

选择如图“*”号选项，并选择 Video capture adapters 进入，找到 OV9650 芯片驱动并选中它，如图：

注意：我们为 CAM130 模块设计的驱动程序，既不属于 VL4 体系，也不属于 V4L2 体系，它就是一个简单的字符设备，这样做是为了方便移植。

6.3.8 配置网卡驱动

要配置网卡驱动，首先要配置网络协议支持
在主菜单中，选择 Netwoking support，回车进入

出现如图子菜单，如图选择 Networking options 并进入

一般我们选择 TCP/IP 协议就够了，但推荐使用我们缺省配置的几个选项，如图

选择完毕，一直退回到主菜单，并选择进入 Device Drivers 菜单。
找到 Network device support，选择进入

找到并进入 Ethernet (10 or 100Mbit) 选项

选中：

<*> Generic Media Independent Interface device support

<*> DM9000 support

选择<Exit>一直返回到 Device Drivers 菜单。

6.3.9 配置 USB 无线网卡驱动

本开发板采用 Linux-2.6.32.2 内核，它已经包含了多种型号的 USB 无线网卡驱动，在我们提供的缺省配置中，也已经包含了大部分常见的网卡型号，如 TP-Link 系列，VIA 系列等，下面是它的驱动配置说明。

在主菜单中，选择 Netwoking support，回车进入

出现如图子菜单，如图选择 Wireless 并进入开始配置无线网络协议

选择如图*各项配置：

退回到内核配置主菜单，选择 Device Drivers 并进入，开始配置无线网卡驱动，如图

进入网络设备子菜单，找到如图无线网络设备子项，并进入

再选择 Wireless LAN(IEEE 802.11)子项，并进入

可以看到已经配置了以芯片厂商为分类方式的常见各种 USB 无需网卡类新，如图为 Ralink 公司芯片方案的 USB 无线网卡驱动支持

选择<Exit>一直返回到 Device Drivers 菜单。

6.3.10 配置音频驱动

在 Device Drivers 菜单中，选择 Sound card supprt，并进入

再作如图选择，并进入

在出现的菜单中，选择 ALSA 接口支持(Advanced Linux Sound Architecture)，并进入

选择 OSS Mixer API 以增加老式的 OSS API 支持，如图

选择 ALSA for Soc audio support，并进入

选择 ALSA 接口驱动支持，如图

选择完毕，一直按<Exit>返回到 Device Drivers 菜单

6.3.11 配置 SD/MMC 卡驱动

在 Device Drivers 菜单中，选择 SD/MMC 设备选项并按回车进入

选择如图<*>各项，如图，这样就配置好了 MMC/SD 卡驱动，它可以支持高速大容

量 SD 卡，最大可达到 32G。

按<Exit>返回到 Device Drivers 菜单。

6.3.12 配置看门狗驱动支持

在 Device Drivers 菜单中，选择 Watchdog 选项并按回车进入

选中如图所示看门狗驱动支持

按<Exit>返回到 Device Drivers 菜单。

6.3.13 配置 LED 驱动

在 Device Drivers 菜单中，选择进入 Character devices - - ->，找到并选中 LEDs 驱动支持，如图。

6.3.14 配置按键驱动

在 Device Drivers 菜单中，选择进入 Character devices - - ->，找到并选中 Buttons 驱动支持，如图。

6.3.15 配置 PWM 控制蜂鸣器驱动

依然在 Character devices 菜单中，找到并选中 buzzer 选项，如图

6.3.16 配置 AD 转换驱动

依然在 Character devices 菜单中，找到并选中 ADC 选项，如图

6.3.17 配置串口驱动

依然在 Character devices 菜单中，选择进入 Serial drivers - - ->

选择如图选项，来配置串口驱动

6.3.18 如何配置 RTC 实时时钟驱动

依然在 Device Drivers 菜单中，选择 Real Time Clock 选项并进入

如图选择 2440 系统的 RTC 驱动支持

返回到主菜单。

6.3.19 配置 I2C-EEPROM 驱动支持

在 Device Drivers 菜单中，找到 I2C support 项，选择进入

在菜单中再选择如图，并进入 I2C Hardware Bus support 子项

再选择 S3C2410 I2C Driver 即可，如图

6.3.20 配置 yaff2s 文件系统的支持

要使用 yaffs2 文件系统，需要先配置 nand flash 驱动支持，在 Device drivers 菜单中选择 MTD 选项如图，并按回车进入

注意子菜单中<*>号的选项，不要取消

找到 NAND Device Support 选项并进入

如图选择 Nand Flash 驱动支持

返回到内核配置主菜单，并找到 File systems 选项进入

找到如图选项 Miscellaneous filesystems 并进入

找到 YAFFS2 支持选项，如图选择

然后<Exit>返回到 File systems 菜单进行下一步

6.3.21 配置 EXT2/VFAT/ NFS 等文件系统

在 File System 菜单中，如图选择 Network File Systems 文件系统的支持并进入。

选择如图选项，这样配置编译出的内核就可以通过 NFS 启动系统了。

为了支持优盘或者 SD 卡等存储设备常用的 FAT32 文件系统，还需要配置与此相关的文件系统支持，如图，在 File Systems 菜单中选择 DOS/FAT/NT Filesystems 选项并进入

在此我们选择了常用的 VFAT 文件系统格式，它可以支持 FAT32

返回到内核配置主菜单，至此，您已经了解内核的大部分常用选项的配置，更多的内核选项需要您在学习中逐步实践和摸索。

6.3.22 制作 Linux logo

当你启动开发板的 Linux 系统时会在液晶屏上看到这样一个图像

这是 Linux 系统的启动 Logo，它在内核中其实是一个特殊格式的图像文件。它在内核中的位置是：**linux-2.6.32.2/drivers/video/logo/linux_logo_clut224.ppm**

有很多方法可以把普通的图片转换为 logo 文件，我们设计了一个简单易用的图形界面的制作工具 LogoMaker，它基于 Fedora 9 平台开发，可以支持 bmp，png，jpg 等格式的文件转换，下面是它的使用方法。

请先按照第五章节安装好 logomaker 工具程序，在任意命令行输入 logomaker，就可以启动它，打开时它会显示一幅缺省的花朵图片，如图，。

点 File->Open a picture file…或者使用快捷键 Ctrl+O 可以打开一个图片文件，在跳出的文件打开窗口中选择一个图片：

图片会显示在 logomaker 窗口中，如图

这时点 File->Convert the picture to a Linux Logo File，或者使用快捷键 Crtl+C 会跳出文件保存目录窗口，不需要输入任何东西，选择要保存的目录即可，文件名将会自动保存为 **linux_logo_clut224.ppm**，使用这个文件代替 linux-2.6.32.2/drivers/video/logo 目录下的同名文件即可。

LogoMaker 的简易说明如图所示(点菜单“help->About”可以打开它)：

6.4 制作目标板文件系统映象

注意：请务必先按照 5.4.3 章节的介绍安装 mkyaffs2image 文件系统制作工具。

在此，我们以制作测试用的 rootfs_qtopia_qt4.img 文件系统映象为例，来介绍 yaffs2 文件系统映象的制作。

进入/opt/FriendlyARM/mini2440 工作目录，执行以下命令：

#cd /opt/FriendlyARM/mini2440

#mkyaffs2image-128M　rootfs_qtopia_qt4　rootfs_qtopia_qt4.img

这样会在当前目录下生成 rootfs_qtopia_qt4.img 映像文件，它和光盘中的同名文件是一样的，你可以通过 USB 烧写到 128M/256M/512M/1GB Nand Flash 的 mini2440/micro2440 系统中使用。

如图：

如果你使用的是 64M Nand Flash 版本的开发板，则需要先删除目标文件系统中的部分内容才可以，否则生成的映像文件太大，无法烧写到开发板中；另外，要使用请使用 **mkyaffs2image** 制作工具来制作，在此简单介绍如下：

(1) 要制作一个只包含 qtopia-2.2.0 图形系统的目标文件系统，只需要删除”rootfs_qtopia_qt4/usr/local/Trolltech”整个目录的内容就可以了。

(2)要制作一个只包含 QtE-4.6.1 图形系统的目标文件系统，只需要把”/opt”目录中的内容全部删除就可以了。

第七章 嵌入式 Linux 应用开发入门指南

本节内容通过嵌入式 Linux 开发最简单的例子，介绍了如何编写和编译 Linux 应用程序，并下载到开发板运行起来，并介绍了如何制作和装载驱动程序模块，以及移植一些常见的开源软件。

嵌入式 Linux 资源丰富，我们不可能介绍到每一个细节，本文旨在提供一些嵌入式 Linux 经常用到的方法，为你打开奇妙世界的大门。

注意：以下示例程序所使用的编译器为 arm-linux-gcc-4.3.2 with EABI，如果你使用了其他版本的交叉编译器，编译完有可能无法在开发板上运行。

要检查交叉编译器的版本类型，可在终端运行命令：arm-linux-gcc –v，如图

7.1Hello,World!

7.1.1 Hello,World 源代码

Hello,World 源代码位于光盘的 linux/examples.tgz 包中，如果您按第五章节的步骤安装了开发环境，它将位于**/opt/FriendlyARM/mini2440/examples/hello** 目录，其源代码如下：

Hello, World 源代码:

```
#include <stdio.h>

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

7.1.2 编译 Hello,World

首先进入测试程序源代码目录

#cd /opt/FriendlyARM/mini2440/examples/hello

然后，使用命令行进行手工交叉编译示例程序

#arm-linux-gcc –o hello main.c

或者借助编译脚本进行编译

#make

最后将生成 hello 可执行文件，使用 file 命令可以检查你生成的 hello 可执行文件是否为 ARM 体系和格式版本，能在开发板上正常运行的可执行文件一般如图所示：

7.1.3 把 Hello,World 下载到开发板运行

将编译好的可执行文件下载到目标板目前主要四种方式：

第一种：复制到介质(如优盘)

第二种：通过网络传送文件到开发板(推荐使用)

第三种：通过串口传送文件到开发板

第四种：通过 NFS(网络文件系统)直接运行

下面分别进行介绍：

(1)使用优盘

方法：先把编译好的可执行程序复制到优盘，再把优盘插到目标板上并挂载它，然后

把程序拷贝到目标板的可执行目录/bin

步骤：

Step1：复制程序到优盘

把优盘插到 PC 的 USB 接口，执行以下命令把程序复制到优盘

#mount /dev/sda1 /mnt　　；挂接优盘

#cp hello /mnt　　；复制刚才编译好的程序到优盘

#umount /mnt　　；卸载优盘

Step2：把程序从优盘拷贝到目标板并执行

把优盘插入到开发板的 USB Host 接口，优盘会自动挂载到/udisk 目录，执行以下命令就可以运行 hello 程序了。

#cd /udisk

#./hello　　；执行 hello 程序

注意：如果此时强制拔出优盘，需要退回到根目录，再执行 umount /udisk 方可为下一次做好自动挂载的准备。

(3) 使用 ftp 传送文件(推荐使用)

方法：使用 ftp 登录目标板，把编译好的程序上传；然后修改上传后目标板上的程序的可执行属性，并执行。

首先，在 PC 端执行，如图所示

```
root@tom:/opt/FriendlyARM/mini2440/examples/hello
File Edit View Terminal Tabs Help
[root@tom hello]# ls
hello  hello.c  Makefile
[root@tom hello]# ftp 192.168.1.230          1. 登录开发板
Connected to 192.168.1.230 (192.168.1.230).
220 FriendlyARM FTP server (Version 6.4/OpenBSD/Linux-ftpd-0.17) ready.
Name (192.168.1.230:root): plg
331 Password required for plg. 2.输入用户名和密码，均为 plg
Password:
230 User plg logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> bin   3. 设置文件传送格式
200 Type set to I.
ftp> put hello  4. 上传hello
local: hello remote: hello
227 Entering Passive Mode (192,168,1,230,171,47)
150 Opening BINARY mode data connection for 'hello'.
226 Transfer complete.
5061 bytes sent in 0.000144 secs (35145.83 Kbytes/sec)
ftp> by    5. 退出登录
221 Goodbye.
[root@tom hello]#
```

然后，在目标板一端执行，如图所示

```
COM1 (1) - CRT
File Edit View Options Transfer Script Window Help
[root@FriendlyARM plg]# cd
[root@FriendlyARM /]# ls
bin              lib              opt              tmp
dev              linuxrc          proc             usr
etc              lost+found       sbin             var
home             mnt              shanghaitan.mp3  www
[root@FriendlyARM /]# cd /home/plg/   进入 /home/plg，因为文件上传至此
[root@FriendlyARM plg]# ls            查看该目录下的文件，可以看到hello
hello
[root@FriendlyARM plg]# chmod +x hello   改变文件的执行权限
[root@FriendlyARM plg]# ls
hello                                 可以看到文件的属性已经变了
[root@FriendlyARM plg]# ./hello
hello, FriendlyARM!                   执行hello
[root@FriendlyARM plg]#
Ready   Serial: COM1   4,  25   18 Rows,  75 Cols   Linux
```

(4)通过串口传送文件到开发板

通过 2.5 章节我们学会了如何通过串口传送文件到开发板，您也可以通过相同的方法

传送 hello 可执行程序，具体步骤在此不再详细描述，记得传送完毕把文件的属性改为可执行才能正常运行。

#chmod +x hello

说明：有些用户使用 USB 转串口线，因为有些转接器性能是不太好的，所以有时会出现“传输超时”或者根本无法传输到开发板的现象，因此我们建议使用 ftp 传送到开发板。

(4)通过网络文件系统 NFS 执行

Linux 中最常用的方法就是采用 NFS 来执行各种程序，这样可以不必花费很多时间下载程序，虽然在此下载 hello 程序用不了多久，一旦您的应用程序变得越来越大，您就会发现使用 NFS 运行的方便所在。

如同前面所讲述的那样，请先按照 4.3 一节搭建好 NFS 服务器系统，然后在命令行输入以下命令（假定服务器的 IP 地址为 192.168.1.111）：

#mount –t nfs –o nolock 192.168.1.111:/opt/FriendlyARM/mini2440/root_qtopia /mnt

挂接成功，您就可以进入/mnt 目录进行操作了，在您的 PC Linux 终端把 hello 复制到 **opt/FriendlyARM/mini2440/root_qtopia** 目录，然后在开发板的串口终端执行

#cd /mnt

#./hello

7.2 嵌入式 Linux 程序开发入门

声明：以下 Linux 示例程序均为友善之臂原创，我们发现有的开发板厂商或者个人修改了 copyright 说明，据为己有，虽然国内对这种抄袭行为基本没有法律约束，但我们对这种无耻的盗窃行为予以鄙视，并敬告大家要尊重原厂家的辛苦劳动。

7.2.1 LED 测试程序

程序源代码说明：

驱动源代码所在目录	**/opt/FriendlyARM/mini2440/linux-2.6.32.2/drivers/char**
驱动程序名称	mini2440_leds.c
设备类型	misc
设备名	/dev/leds
测试程序源代码目录	**/opt/FriendlyARM/mini2440/examples/leds**
测试程序名称	led.c
测试程序可执行文件名称	led
说明：LED 驱动已经被编译到缺省内核中，因此不能再使用 insmod 方式加载。	

测试程序清单：

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

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

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

/* 检查 led 控制的两个参数，如果没有参数输入则退出。*/
if (argc != 3 || sscanf(argv[1], "%d", &led_no) != 1 || sscanf(argv[2],"%d", &on) != 1 ||
      on < 0 || on > 1 || led_no < 0 || led_no > 3) {
      fprintf(stderr, "Usage: leds led_no 0|1\n");
      exit(1);
}

/*打开/dev/leds 设备文件*/
fd = open("/dev/leds0", 0);
if (fd < 0) {
      fd = open("/dev/leds", 0);
}
if (fd < 0) {
      perror("open device leds");
      exit(1);
}

/*通过系统调用 ioctl 和输入的参数控制 led*/
ioctl(fd, on, led_no);
/*关闭设备句柄*/
close(fd);
return 0;
}
```

你可以按照上面的 hello 程序的步骤手工编译出 led 可执行文件，然后下载到开发板运行它。

7.2.2 测试按键

程序源代码说明：

驱动源代码所在目录	**/opt/FriendlyARM/mini2440/linux-2.6.32.2/drivers/char**
驱动程序名称	mini2440_buttons.c
设备类型	misc
设备名	/dev/buttons
测试程序源代码目录	**/opt/FriendlyARM/mini2440/examples/buttons**
测试程序源代码名称	buttons_test.c
测试程序可执行文件名称	buttons
说明：按键驱动已经被编译到缺省内核中，因此不能再使用 insmod 方式加载。	

测试程序清单：

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/select.h>
#include <sys/time.h>
#include <errno.h>

int main(void)
{
    int buttons_fd;
    char buttons[6] = {'0', '0', '0', '0', '0', '0'};

    buttons_fd = open("/dev/buttons", 0);
    if (buttons_fd < 0) {
        perror("open device buttons");
        exit(1);
    }

    for (;;) {
        char current_buttons[6];
        int count_of_changed_key;
        int i;
        if (read(buttons_fd, current_buttons, sizeof current_buttons) != sizeof current_buttons)
{
            perror("read buttons:");
            exit(1);
```

```
            }

            for (i = 0, count_of_changed_key = 0; i < sizeof buttons / sizeof buttons[0]; i++) {
                if (buttons[i] != current_buttons[i]) {
                    buttons[i] = current_buttons[i];
                    printf("%skey %d is %s", count_of_changed_key? ", ": "", i+1, buttons[i] ==
'0' ? "up" : "down");
                    count_of_changed_key++;
                }
            }
            if (count_of_changed_key) {
                printf("\n");
            }


        }

        close(buttons_fd);
        return 0;
    }
```

你可以按照上面的 hello 程序的步骤手工编译出 buttons 可执行文件，然后下载到开发板运行它

7.2.3 PWM 控制蜂鸣器编程示例

程序源代码说明：

驱动源代码所在目录	**/opt/FriendlyARM/mini2440/linux-2.6.32.2/drivers/char**
驱动程序名称	mini2440_pwm.c
设备类型	misc
设备名	/dev/pwm
测试程序源代码目录	**/opt/FriendlyARM/mini2440/examples/pwm**
测试程序源代码名称	pwm_test.c
测试程序可执行文件名称	Pwm_test
说明：PWM 控制蜂鸣器驱动已经被编译到缺省内核中，因此不能再使用 insmod 方式加载。	

测试程序源代码清单：

```
#include <stdio.h>
#include <termios.h>
#include <unistd.h>
#include <stdlib.h>
```

```
#define PWM_IOCTL_SET_FREQ          1
#define PWM_IOCTL_STOP              2

#define  ESC_KEY         0x1b

static int getch(void)
{
        struct termios oldt,newt;
        int ch;

        if (!isatty(STDIN_FILENO)) {
                fprintf(stderr, "this problem should be run at a terminal\n");
                exit(1);
        }
        // save terminal setting
        if(tcgetattr(STDIN_FILENO, &oldt) < 0) {
                perror("save the terminal setting");
                exit(1);
        }

        // set terminal as need
        newt = oldt;
        newt.c_lflag &= ~( ICANON | ECHO );
        if(tcsetattr(STDIN_FILENO,TCSANOW, &newt) < 0) {
                perror("set terminal");
                exit(1);
        }

        ch = getchar();

        // restore termial setting
        if(tcsetattr(STDIN_FILENO,TCSANOW,&oldt) < 0) {
                perror("restore the termial setting");
                exit(1);
        }
        return ch;
}

static int fd = -1;
static void close_buzzer(void);
```

```
static void open_buzzer(void)
{
	fd = open("/dev/pwm", 0);
	if (fd < 0) {
		perror("open pwm_buzzer device");
		exit(1);
	}

	// any function exit call will stop the buzzer
	atexit(close_buzzer);
}

static void close_buzzer(void)
{
	if (fd >= 0) {
		ioctl(fd, PWM_IOCTL_STOP);
		close(fd);
		fd = -1;
	}
}

static void set_buzzer_freq(int freq)
{
	// this IOCTL command is the key to set frequency
	int ret = ioctl(fd, PWM_IOCTL_SET_FREQ, freq);
	if(ret < 0) {
		perror("set the frequency of the buzzer");
		exit(1);
	}
}
static void stop_buzzer(void)
{
	int ret = ioctl(fd, PWM_IOCTL_STOP);
	if(ret < 0) {
		perror("stop the buzzer");
		exit(1);
	}
}

int main(int argc, char **argv)
{
```

```
        int freq = 1000 ;

        open_buzzer();

        printf( "\nBUZZER TEST ( PWM Control )\n" );
        printf( "Press +/- to increase/reduce the frequency of the BUZZER\n" ) ;
        printf( "Press 'ESC' key to Exit this program\n\n" );


        while( 1 )
        {
            int key;

            set_buzzer_freq(freq);
            printf( "\tFreq = %d\n", freq );

            key = getch();

            switch(key) {
            case '+':
                if( freq < 20000 )
                    freq += 10;
                break;

            case '-':
                if( freq > 11 )
                    freq -= 10 ;
                break;

            case ESC_KEY:
            case EOF:
                stop_buzzer();
                exit(0);

            default:
                break;
            }
        }
    }
```

7.2.4 I2C-EEPROM 编程示例

程序源代码说明：

驱动源代码所在目录	**/opt/FriendlyARM/mini2440/linux-2.6.32.2/drivers/i2c/busses**
驱动程序名称	I2c-s3c2410.c
设备类型	字符设备
设备名	/dev/i2c/0
测试程序源代码目录	**/opt/FriendlyARM/mini2440/examples/i2c**
测试程序源代码名称	eeprog.c 24cXX.c
测试程序可执行文件名称	I2c
说明：I2C 驱动已经被编译到缺省内核中，因此不能再使用 insmod 方式加载。	

测试程序清单：

注意：以下程序还需同目录下 24cXX.c 程序的支持

```
/*****************************************************************************
        copyright                    : (C) by 2009 Guangzhou FriendlyARM, in China
        email                        : capbily@163.com
        website            : http://www.arm9.net

 *****************************************************************************/

#include <stdio.h>
#include <fcntl.h>
#include <getopt.h>
#include <unistd.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include "24cXX.h"

#define usage_if(a) do { do_usage_if( a , __LINE__); } while(0);
void do_usage_if(int b, int line)
{
        const static char *eeprog_usage =
                "I2C-24C08(256 bytes) Read/Write Program, ONLY FOR TEST!\n"
                "FriendlyARM Computer Tech. 2009\n";
        if(!b)
                return;
        fprintf(stderr, "%s\n[line %d]\n", eeprog_usage, line);
```

```
        exit(1);
}


#define die_if(a, msg) do { do_die_if( a , msg, __LINE__); } while(0);
void do_die_if(int b, char* msg, int line)
{
        if(!b)
                return;
        fprintf(stderr, "Error at line %d: %s\n", line, msg);
        fprintf(stderr, "        sysmsg: %s\n", strerror(errno));
        exit(1);
}


static int read_from_eeprom(struct eeprom *e, int addr, int size)
{
        int ch, i;
        for(i = 0; i < size; ++i, ++addr)
        {
                die_if((ch = eeprom_read_byte(e, addr)) < 0, "read error");
                if( (i % 16) == 0 )
                        printf("\n %.4x|    ", addr);
                else if( (i % 8) == 0 )
                        printf("    ");
                printf("%.2x ", ch);
                fflush(stdout);
        }
        fprintf(stderr, "\n\n");
        return 0;
}

static int write_to_eeprom(struct eeprom *e, int addr)
{
        int i;
        for(i=0, addr=0; i<256; i++, addr++)
        {
                if( (i % 16) == 0 )
                        printf("\n %.4x|    ", addr);
                else if( (i % 8) == 0 )
                        printf("    ");
```

```
            printf("%.2x ", i);
            fflush(stdout);
            die_if(eeprom_write_byte(e, addr, i), "write error");
        }
        fprintf(stderr, "\n\n");
        return 0;
}

int main(int argc, char** argv)
{
        struct eeprom e;
        int op;

        op = 0;

        usage_if(argc != 2 || argv[1][0] != '-' || argv[1][2] != '\0');
        op = argv[1][1];

        fprintf(stderr, "Open /dev/i2c/0 with 8bit mode\n");
        die_if(eeprom_open("/dev/i2c/0", 0x50, EEPROM_TYPE_8BIT_ADDR, &e) < 0,
                "unable to open eeprom device file "
                "(check that the file exists and that it's readable)");
        switch(op)
        {
        case 'r':
            fprintf(stderr, "    Reading 256 bytes from 0x0\n");
            read_from_eeprom(&e, 0, 256);
            break;
        case 'w':
            fprintf(stderr, "    Writing 0x00-0xff into 24C08 \n");
            write_to_eeprom(&e, 0);
            break;
        default:
            usage_if(1);
            exit(1);
        }
        eeprom_close(&e);

        return 0;
}
```

7.2.5 串口编程示例

程序源代码说明：

驱动源代码所在目录	**/opt/FriendlyARM/mini2440/linux-2.6.32.2/drivers/serial/**
驱动程序名称	S3c2440.c
设备名	/dev/ttySAC0,1,2
测试程序源代码目录	**/opt/FriendlyARM/mini2440/examples/comtest**
测试程序源代码名称	comtest.c
测试程序可执行文件名称	armcomtest
说明：测试程序编译后可得到 x86 版本和 arm 版本，其源代码是完全一样的。	

说明：comtest 程序是友善之臂早期开发的一个串口测试程序，它其实是一个十分简易的串口终端程序，类似于 linux 中的 minicom，该程序与硬件无关，因此相同的代码不仅适用于任何 Arm-linux 开发板平台，也可以在 PC linux 上运行使用，方法都是完全一样的。通过该程序你可以了解串口编程的一些常见关键设置，对于 linux 下串口编程很有帮助和借鉴意义，该程序虽然十分短小，但设计极为严谨巧妙，我们对此就不详细解释了，下面是它的完整源代码：

注：本程序版权归属友善之臂所有，任何单位或个人转载或复制均需注明出处，且不得用于商业用途。

```
# include <stdio.h>
# include <stdlib.h>
# include <termio.h>
# include <unistd.h>
# include <fcntl.h>
# include <getopt.h>
# include <time.h>
# include <errno.h>
# include <string.h>

static void Error(const char *Msg)
{
        fprintf (stderr, "%s\n", Msg);
        fprintf (stderr, "strerror() is %s\n", strerror(errno));
        exit(1);
}
static void Warning(const char *Msg)
{
         fprintf (stderr, "Warning: %s\n", Msg);
}
```

```
static int SerialSpeed(const char *SpeedString)
{
        int SpeedNumber = atoi(SpeedString);
#       define TestSpeed(Speed) if (SpeedNumber == Speed) return B##Speed
        TestSpeed(1200);
        TestSpeed(2400);
        TestSpeed(4800);
        TestSpeed(9600);
        TestSpeed(19200);
        TestSpeed(38400);
        TestSpeed(57600);
        TestSpeed(115200);
        TestSpeed(230400);
        Error("Bad speed");
        return -1;
}

static void PrintUsage(void)
{

      fprintf(stderr, "comtest - interactive program of comm port\n");
      fprintf(stderr, "press [ESC] 3 times to quit\n\n");

      fprintf(stderr, "Usage: comtest [-d device] [-t tty] [-s speed] [-7] [-c] [-x] [-o] [-h]\n");
      fprintf(stderr, "                -7 7 bit\n");
      fprintf(stderr, "                -x hex mode\n");
      fprintf(stderr, "                -o output to stdout too\n");
      fprintf(stderr, "                -c stdout output use color\n");
      fprintf(stderr, "                -h print this help\n");
      exit(-1);
}

static inline void WaitFdWriteable(int Fd)
{
        fd_set WriteSetFD;
        FD_ZERO(&WriteSetFD);
        FD_SET(Fd, &WriteSetFD);
        if (select(Fd + 1, NULL, &WriteSetFD, NULL, NULL) < 0) {
          Error(strerror(errno));
        }
```

```
}

int main(int argc, char **argv)
{
        int CommFd, TtyFd;

        struct termios TtyAttr;
        struct termios BackupTtyAttr;

        int DeviceSpeed = B115200;
        int TtySpeed = B115200;
        int ByteBits = CS8;
        const char *DeviceName = "/dev/ttyS0";
        const char *TtyName = "/dev/tty";
        int OutputHex = 0;
        int OutputToStdout = 0;
        int UseColor = 0;

        opterr = 0;
        for (;;) {
                int c = getopt(argc, argv, "d:s:t:7xoch");
                if (c == -1)
                        break;
                switch(c) {
                case 'd':
                        DeviceName = optarg;
                        break;
                case 't':
                        TtyName = optarg;
                        break;
                case 's':
                if (optarg[0] == 'd') {
                DeviceSpeed = SerialSpeed(optarg + 1);
                } else if (optarg[0] == 't') {
                TtySpeed = SerialSpeed(optarg + 1);
                } else
                        TtySpeed = DeviceSpeed = SerialSpeed(optarg);
                        break;
        case 'o':
                OutputToStdout = 1;
                break;
```

```
case '7':
        ByteBits = CS7;
        break;
         case 'x':
                OutputHex = 1;
                break;
case 'c':
        UseColor = 1;
        break;
         case '?':
         case 'h':
         default:
        PrintUsage();
         }
 }
 if (optind != argc)
         PrintUsage();

 CommFd = open(DeviceName, O_RDWR, 0);
 if (CommFd < 0)
Error("Unable to open device");
 if (fcntl(CommFd, F_SETFL, O_NONBLOCK) < 0)
      Error("Unable set to NONBLOCK mode");



 memset(&TtyAttr, 0, sizeof(struct termios));
 TtyAttr.c_iflag = IGNPAR;
 TtyAttr.c_cflag = DeviceSpeed | HUPCL | ByteBits | CREAD | CLOCAL;
 TtyAttr.c_cc[VMIN] = 1;

 if (tcsetattr(CommFd, TCSANOW, &TtyAttr) < 0)
         Warning("Unable to set comm port");

 TtyFd = open(TtyName, O_RDWR | O_NDELAY, 0);
 if (TtyFd < 0)
Error("Unable to open tty");

 TtyAttr.c_cflag = TtySpeed | HUPCL | ByteBits | CREAD | CLOCAL;
 if (tcgetattr(TtyFd, &BackupTtyAttr) < 0)
Error("Unable to get tty");
```

```
 if (tcsetattr(TtyFd, TCSANOW, &TtyAttr) < 0)
Error("Unable to set tty");


 for (;;) {
unsigned char Char = 0;
fd_set ReadSetFD;

void OutputStdChar(FILE *File) {
        char Buffer[10];
        int Len = sprintf(Buffer, OutputHex ? "%.2X    " : "%c", Char);
        fwrite(Buffer, 1, Len, File);
}

FD_ZERO(&ReadSetFD);

FD_SET(CommFd, &ReadSetFD);
FD_SET( TtyFd, &ReadSetFD);
#   define max(x,y) ( ((x) >= (y)) ? (x) : (y) )
if (select(max(CommFd, TtyFd) + 1, &ReadSetFD, NULL, NULL, NULL) < 0) {
        Error(strerror(errno));
}
#   undef max

if (FD_ISSET(CommFd, &ReadSetFD)) {
        while (read(CommFd, &Char, 1) == 1) {

       WaitFdWriteable(TtyFd);
       if (write(TtyFd, &Char, 1) < 0) {
              Error(strerror(errno));
       }
       if (OutputToStdout) {
              if (UseColor)
             fwrite("\x1b[01;34m", 1, 8, stdout);
              OutputStdChar(stdout);
              if (UseColor)
             fwrite("\x1b[00m", 1, 8, stdout);
              fflush(stdout);
       }
        }
```

```
        }

        if (FD_ISSET(TtyFd, &ReadSetFD)) {
            while (read(TtyFd, &Char, 1) == 1) {
                static int EscKeyCount = 0;
            WaitFdWriteable(CommFd);
                if (write(CommFd, &Char, 1) < 0) {
                Error(strerror(errno));
            }
            if (OutputToStdout) {
                if (UseColor)
                fwrite("\x1b[01;31m", 1, 8, stderr);
                OutputStdChar(stderr);
                if (UseColor)
                fwrite("\x1b[00m", 1, 8, stderr);
                fflush(stderr);
                }

                if (Char == '\x1b') {
                    EscKeyCount ++;
                    if (EscKeyCount >= 3)
                        goto ExitLabel;
                } else
                    EscKeyCount = 0;
            }
            }

        }

    ExitLabel:
        if (tcsetattr(TtyFd, TCSANOW, &BackupTtyAttr) < 0)
        Error("Unable to set tty");

        return 0;
    }
```

7.2.6 UDP 网络编程

程序源代码说明：

驱动源代码所在目录	**/opt/FriendlyARM/mini2440/linux-2.6.32.2/drivers/net/**

驱动程序名称	dm9000.c
该驱动的主设备号	无
设备名	eth0 (网络设备并不在/dev 目录中出现)
测试程序源代码目录	**/opt/FriendlyARM/mini2440/examples/udptak**
测试程序源代码名称	udptalk.c
测试程序可执行文件名称	udptalk.c
说明：测试程序编译后可得到 x86 版本和 arm 版本，其源代码是完全一样的。	

TCP/IP 提供了无连接的传输层协议：UDP(User Datagram Protocol，即用户数据报协议)。UDP 与 TCP 有很大的区别，因为无连接的 socket 编程与面向连接的 socket 编程也有很大的差异。由于不用建立连接，因此每个发送个接收的数据报都包含了发送方和接收方的地址信息。

在发送和接收数据之前，先要建立一个数据报方式的套接字，该 socket 的类型为 SOCK_DGRAM，用如下的调用产生：

sockfd=socket(AF_INET, SOCK_DGRAM, 0);

由于不需要建立连接，因此产生 socket 后就可以直接发送和接收了。当然，要接收数据报也必须绑定一个端口，否则发送方无法得知要发送到哪个端口。Sendto 和 recvfrom 两个系统调用分别用于发送和接收数据报，其调用格式为：

int sendto(int s, const void *msg, int len, unsigned int flags, const struct sockaddr *to, int tolen);

int recvfrom(int, s, void *buf, int len, unsigned int flags, struct sockaddr *from, int fromlen);

其中 s 为所使用的 socket，msg 和 buf 分别为发送和接收的缓冲区指针，len 为缓冲区的长度，flags 为选项标志，此处还用不到，设为 0 即可。to 和 from 就是发送的目的地址和接收的来源地址，包含了 IP 地址和端口信息。tolen 和 fromlen 分别是 to 和 from 这两个 socket 地址结构的长度。这两个函数的返回值就是实际发送和接收的字节数，返回-1 表示出错。

使用无连接方式通信的基本过程如图所示。

UDP 通信的基本过程

上图描述的是通信双方都绑定自己地址端口的情形，但在某些情况下，也可能有一方不用绑定地址和端口。不绑定的一方的地址和端口由内核分配。由于对方无法预先知道不绑定的一方的端口和 IP 地址(假设主机有多个端口，这些端口分配了不同的 IP 地址)，因此只能由不绑定的一方先发出数据报，对方根据收到的数据报中的来源地址就可以确定回送数据报所需要的发送地址了。显然，在这种情况下对方必须绑定地址和端口，并且通信只能由非绑定方发起。

与 read()和 write()相似，进程阻塞在 recvfrom()和 sendto()中也会发生。但与 TCP 方式不同的是，接收到一个字节数为 0 的数据报是有可能的，应用程序完全可以将 sendto()中的 msg 设为 NULL，同时将 len 设为 0。

下面是一个基于以上原理分析的一个 udp 编程的例子：

```
/*
 *      udptalk ： Example for Matrix V    ;说明：本程序同样适用于 mini2440
 *
 *      Copyright (C) 2004 capbily - friendly-arm
 *      capbily@hotmail.com
 */
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <stdio.h>

#define BUFLEN 255
```

```
int main(int argc, char **argv)
{
    struct sockaddr_in peeraddr, /*存放谈话对方 IP 和端口的 socket 地址*/
                        localaddr;/*本端 socket 地址*/
    int sockfd;
    char recmsg[BUFLEN+1];
    int socklen, n;

    if(argc!=5){
        printf("%s <dest IP address> <dest port> <source IP address> <source port>\n",
argv[0]);
        exit(0);
    }

    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if(sockfd<0){
        printf("socket creating err in udptalk\n");
        exit(1);
    }
    socklen = sizeof(struct sockaddr_in);
    memset(&peeraddr, 0, socklen);
    peeraddr.sin_family=AF_INET;
    peeraddr.sin_port=htons(atoi(argv[2]));
    if(inet_pton(AF_INET, argv[1], &peeraddr.sin_addr)<=0){
        printf("Wrong dest IP address!\n");
        exit(0);
    }
    memset(&localaddr, 0, socklen);
    localaddr.sin_family=AF_INET;
    if(inet_pton(AF_INET, argv[3], &localaddr.sin_addr)<=0){
        printf("Wrong source IP address!\n");
        exit(0);
    }
    localaddr.sin_port=htons(atoi(argv[4]));
    if(bind(sockfd, &localaddr, socklen)<0){
        printf("bind local address err in udptalk!\n");
        exit(2);
    }

    if(fgets(recmsg, BUFLEN, stdin) == NULL) exit(0);
    if(sendto(sockfd, recmsg, strlen(recmsg), 0, &peeraddr, socklen)<0){
```

```
        printf("sendto err in udptalk!\n");
        exit(3);
    }

    for(;;){
        /*recv&send message loop*/
        n = recvfrom(sockfd, recmsg, BUFLEN, 0, &peeraddr, &socklen);
        if(n<0){
            printf("recvfrom err in udptalk!\n");
            exit(4);
        }else{
            /*成功接收到数据报*/
            recmsg[n]=0;
            printf("peer:%s", recmsg);
        }
        if(fgets(recmsg, BUFLEN, stdin) == NULL) exit(0);
        if(sendto(sockfd, recmsg, strlen(recmsg), 0, &peeraddr, socklen)<0){
            printf("sendto err in udptalk!\n");
            exit(3);
        }
    }
}
```

将 udptalk.c 编译好后就可以运行了，/opt/FriendlyARM/mini2440/examples/udptalk 目录下的 Makefile 指定了两个编译目标可执行文件，一个用于在主机端的 x86-udptalk，一个是用于开发板的 arm-udptalk，运行 make 命令将把这两个程序一起编译出来。可以把 arm-udptalk 使用上面介绍的方法下载到开发板中(预装的 Linux 不含该程序)，假设主机的 IP 地址为 192.168.1.108，开发板的 IP 地址为 192.168.1.230。

在主机的终端上输入：

#./x86-udptalk 192.168.1.230 2000 192.168.1.108 2000

在开发板上的终端输入

#arm-udptalk 192.168.1.108 2000 192.168.1.230 2000

则运行结果分别如图所示：

在主机上运行 **x86-udptalk**

在开发板上运行 **arm-udptalk**

7.2.7 数学函数库调用示例

程序源代码说明：

测试程序源代码目录	**/opt/FriendlyARM/mini2440/examples/math**
测试程序源代码名称	mathtest.c
测试程序可执行文件名称	mathtest
注意：使用数学函数的关键是要包含其头文件 math.h，并且在编译的时候加入数学函数库 libm。	

程序清单：

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h> ;注意：一定要包含此头文件

int main(void)
{
```

```
        double a=8.733243;

        printf("sqrt(%f)=%f\n", a, sqrt(a));

        return 0;
}
```

相应的 Makefile 内容：

```
CROSS=arm-linux-

all: mathtest

#注意：该处包含了数学函数库 libm，红色部分
mathtest:
        $(CROSS)gcc -o mathtest main.c -lm

clean:
        @rm -vf mathtest *.o *~
```

你可以按照上面的 hello 程序的步骤手工编译出 mathtest 可执行文件，然后下载到开发板运行它

7.2.8 线程编程示例

程序源代码说明：

测试程序源代码目录	**/opt/FriendlyARM/mini2440/examples/pthread**
测试程序源代码名称	pthread_test.c
测试程序可执行文件名称	pthread_test
注意：使用线程的关键是要包含其头文件 pthread.h，并且在编译的时候加入线程库 libpthread。	

程序清单：

```
#include<stddef.h>
#include<stdio.h>
#include<unistd.h>
#include"pthread.h" ;注意：一定要包含此头文件

void reader_function(void);
void writer_function(void);
char buffer;
```

```
int buffer_has_item=0;
pthread_mutex_t mutex;
main()
{
	pthread_t reader;
	pthread_mutex_init(&mutex,NULL);
	pthread_create(&reader,NULL,(void*)&reader_function,NULL);
	writer_function();
}
void writer_function(void)
{
	while(1)
	{
		pthread_mutex_lock(&mutex);
		if(buffer_has_item==0)
		{
			buffer='a';
			printf("make a new item\n");
			buffer_has_item=1;
		}
		pthread_mutex_unlock(&mutex);
	}
}
void reader_function(void)
{
	while(1)
	{
		pthread_mutex_lock(&mutex);
		if(buffer_has_item==1)
		{
			buffer='\0';
			printf("consume    item\n");
			buffer_has_item=0;
		}
		pthread_mutex_unlock(&mutex);
	}
}
```

相应的 Makefile 内容：

```
CROSS=arm-linux-
```

```
all: pthread

#注意：该处包含了线程库 libphread，红色部分
pthread:
        $(CROSS)gcc -static -o pthread main.c -lpthread

clean:
        @rm -vf pthread *.o *~
```

你可以按照上面的 hello 程序的步骤手工编译出 pthread 可执行文件，然后下载到开发板运行它

7.2.9 管道应用编程示例-网页控制 LED

程序源代码说明：

测试程序源代码目录	**/opt/FriendlyARM/mini2440/examples/led-player**
测试程序源代码名称	led-player.c
测试程序可执行文件名称	led-player
说明：见以下描述	

开机后我们可以通过网页发送命令控制开发板上的 LED 闪烁模式，其实这是进程间通信共享资源的一个典型例子，进程间通信就是 IPC(InterProcess Communication)，进程间通信的目的一般有：

(1)数据传输
(2)共享数据
(3)通知事件
(4)资源共享
(5)进程控制.

Linux 支持多种 IPC 机制，信号和管道是其中的两个。关于更详细的进程间通信的介绍，一般 Linux 编程的书上都有介绍，在此我们不多说了。

通过网页来控制 LED 的闪烁模式就是通过管道机制来实现的，其中 LED 是共享资源，led-player 是一个后台程序，它启动的时候就创建了一个命名管道/tmp/ led-control(当然该管道也可以通过命令 mknod 来创建，那样程序就要改写了，有兴趣的可以自己试试)， 并一直监测输入该管道的数据，根据不同的参数(模式 type 和周期 period)来改变 LED 的显示模式；leds.cgi 是一个网关程序，它接收从网页发送过来的字符形式指令（ping 代表跑马灯模式或者乒乓模式，counter 代表计数器模式，stop 代表停止模式，slow 代表周期为 0.25m，normal 代表周期为 0.125m，fast 代表周期为 0.0625m），并对这些指令进行赋值转换为实际数字，然后调用 echo 命令输送到管道/tmp/ led-control 以此实现对 LED 的控制，以下是各自的程序清单。

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

```
#include <stdlib.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/select.h>
#include <sys/time.h>
#include <string.h>

static int led_fd;
static int type = 1;

static void push_leds(void)
{
  static unsigned step;
  unsigned led_bitmap;
  int i;

  switch(type) {
  case 0:
        if (step >= 6) {
              step = 0;
        }
        if (step < 3) {
              led_bitmap = 1 << step;
        } else {
              led_bitmap = 1 << (6 - step);
        }
        break;
  case 1:
        if (step > 255) {
              step = 0;
        }
        led_bitmap = step;
        break;
  default:
        led_bitmap = 0;
  }
  step++;
  for (i = 0; i < 4; i++) {
```

```
        ioctl(led_fd, led_bitmap & 1, i);
        led_bitmap >>= 1;
  }
}

int main(void)
{
  int led_control_pipe;
  int null_writer_fd; // for read endpoint not blocking when control process exit

  double period = 0.5;

  led_fd = open("/dev/leds0", 0);
  if (led_fd < 0) {
        led_fd = open("/dev/leds", 0);
  }
  if (led_fd < 0) {
        perror("open device leds");
        exit(1);
  }
  unlink("/tmp/led-control");
  mkfifo("/tmp/led-control", 0666);

  led_control_pipe = open("/tmp/led-control", O_RDONLY | O_NONBLOCK);
  if (led_control_pipe < 0) {
        perror("open control pipe for read");
        exit(1);
  }
  null_writer_fd = open("/tmp/led-control", O_WRONLY | O_NONBLOCK);
  if (null_writer_fd < 0) {
        perror("open control pipe for write");
        exit(1);
  }

  for (;;) {
        fd_set rds;
        struct timeval step;
        int ret;

        FD_ZERO(&rds);
        FD_SET(led_control_pipe, &rds);
```

```
        step.tv_sec    = period;
        step.tv_usec = (period - step.tv_sec) * 1000000L;

        ret = select(led_control_pipe + 1, &rds, NULL, NULL, &step);
        if (ret < 0) {
            perror("select");
            exit(1);
        }
        if (ret == 0) {
            push_leds();
        } else if (FD_ISSET(led_control_pipe, &rds)) {
            static char buffer[200];
            for (;;) {
                char c;
                int len = strlen(buffer);
                if (len >= sizeof buffer - 1) {
                    memset(buffer, 0, sizeof buffer);
                    break;
                }
                if (read(led_control_pipe, &c, 1) != 1) {
                    break;
                }
                if (c == '\r') {
                    continue;
                }
                if (c == '\n') {
                    int tmp_type;
                    double tmp_period;
                    if (sscanf(buffer,"%d%lf", &tmp_type, &tmp_period) == 2) {
                        type = tmp_type;
                        period = tmp_period;
                    }
                    fprintf(stderr, "type is %d, period is %lf\n", type, period);
                    memset(buffer, 0, sizeof buffer);
                    break;
                }
                buffer[len] = c;
            }
        }
    }
```

```
  close(led_fd);
  return 0;
}
```

使用 make 指令可以直接编译出 led-player 可执行文件，它被作为一个服务器放置在开发板的/sbin 目录中。

Leds.cgi 网关程序源代码(该程序在开发板上的位置：/www/leds.cgi)，可见该网关程序其实就是一个 shell 脚本，它被网页 leds.html 调用为一个执行“action”：

```
#!/bin/sh

type=0
period=1

case $QUERY_STRING in
        *ping*)
                type=0
                ;;
        *counter*)
                type=1
                ;;
        *stop*)
                type=2
                ;;
esac

case $QUERY_STRING in
        *slow*)
                period=0.25
                ;;
        *normal*)
                period=0.125
                ;;
        *fast*)
                period=0.0625
                ;;
esac

/bin/echo $type $period > /tmp/led-control
```

```
echo "Content-type: text/html; charset=gb2312"
echo
/bin/cat led-result.template

exit 0
```

7.2.10 基于 C++的 Hello,World

程序源代码说明：

测试程序源代码目录	**/opt/FriendlyARM/mini2440/examples/c++**
测试程序源代码名称	cplus.c++
测试程序可执行文件名称	cplus
说明：无	

程序清单及注释：

程序清单：

```
#include <iostream>
#include <cstring>
using namespace std;

class String
{
   private:
            char *str;
   public:
        String(char *s)
        {
            int lenght=strlen(s);
            str = new char[lenght+1];
            strcpy(str, s);
        }
        ~String()
        {
       cout << "Deleting str.\n";
       delete[] str;
        }
        void display()
        {
       cout << str <<endl;
        }
```

```
};

int main(void)
{
    String s1="I like FriendlyARM.";
    cout << "s1=";
    s1.display();
    return 0;
    double num, ans;

    cout << "Enter num:";
}
```

你可以按照上面的 hello 程序的步骤手工编译出 cplus 可执行文件，然后下载到开发板运行它

7.3 最简单的嵌入式 Linux 驱动程序模块

7.1 一节我们介绍了一个简单的 Linux 程序 Hello,World，它是运行于用户态的应用程序，现在我们再介绍一个运行于内核态的 Hello, World 程序，它其实是一个最简单的驱动程序模块。

7.3.1 Hello,Module 源代码

程序源代码说明：

源代码所在目录	**/opt/FriendlyARM/mini2440/linux-2.6.32.2/drivers/char**
源代码文件名称	mini2440_hello_module.c
该驱动的主设备号	无
设备名	无
测试程序源代码目录	无
测试程序名称	无
测试程序可执行文件名称	无
说明：该驱动装载后不会在 **dev** 下创建任何设备节点。	

源代码内容如下：

```
#include <linux/kernel.h>
#include <linux/module.h>


static int __init mini2440_hello_module_init(void)
{
	printk("Hello, Mini2440 module is installed !\n");
	return 0;
}

static void __exit mini2440_hello_module_cleanup(void)
{
	printk("Good-bye, Mini2440 module was removed!\n");
}

module_init(mini2440_hello_module_init);
module_exit(mini2440_hello_module_cleanup);
MODULE_LICENSE("GPL");
```

7.3.2 把 Hello,Module 加入内核代码树，并编译

一般编译 2.6 版本的驱动模块需要把驱动代码加入内核代码树，并做相应的配置，如下步骤(**注意：实际上以下步骤均已经做好，你只需要打开检查一下直接编译就可以了**)：

Step1：编辑配置文件 Kconfig，加入驱动选项，使之在 make menuconfig 的时候出现

打开 linux-2.6.32.2/drivers/char/Kconfig 文件，添加如图所示：

```
root@tom:/opt/FriendlyARM/mini2440/linux-2.6.29/drivers/char
File  Edit  View  Terminal  Tabs  Help
          /dev/kmem device is rarely used, but can be used for certain
          kind of kernel debugging operations.
          When in doubt, say "N".

config LEDS_MINI2440
        tristate "LED Support for Mini2440/QQ2440 GPIO LEDs"
        depends on ARCH_S3C2410
        help
          This option enables support for LEDs connected to GPIO lines
          on Mini2440/QQ2440 boards.

config MINI2440_HELLO_MODULE
        tristate "Mini2440/QQ2440 module sample"
        depends on ARCH_S3C2410
        default m if MACH_FRIENDLY_ARM_MINI2440
        help
          Mini2440/QQ2440 module sample.

config MINI2440_BUTTONS
        tristate "Buttons driver for FriendlyARM Mini2440/QQ2440 development boa
rds"
        depends on MACH_FRIENDLY_ARM_MINI2440
        default y if MACH_FRIENDLY_ARM_MINI2440
```

保存退出，这时在 linux-2.6.32.2 目录位置运行一下 make menuconfig 就可以在 Device Drivers → Character devices 菜单中看到刚才所添加的选项了，按下空格键将会选择为<M>，此意为要把该选项编译为模块方式；再按下空格会变为<*>，意为要把该选项编译到内核中，在此我们选择<M>，如图：

```
root@tom:/opt/FriendlyARM/mini2440/linux-2.6.29
File  Edit  View  Terminal  Tabs  Help
.config - Linux Kernel v2.6.29 Configuration

                    Character devices
  Arrow keys navigate the menu.  <Enter> selects submenus --->.
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
  <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
  for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >

      -*- Virtual terminal
      [ ]   Support for binding and unbinding console drivers
      [ ] /dev/kmem virtual device support
      <*> LED Support for Mini2440/QQ2440 GPIO LEDs
      <M> Mini2440/QQ2440 module sample
      <*> Buttons driver for FriendlyARM Mini2440/QQ2440 development bo
      <*> Buzzer driver for FriendlyARM Mini2440/QQ2440 development boa
      [*] ADC driver for FriendlyARM Mini2440/QQ2440 development boards
      [ ] Non-standard serial port support
          Serial drivers  --->
      v(+)

                <Select>    < Exit >    < Help >
```

Step2：通过上一步，我们虽然可以在配置内核的时候进行选择，但实际上此时执行编译内核还是不能把 mini2440_hello_module.c 编译进去的，还需要在 Makefile 中把内核配置

选项和真正的源代码联系起来，打开 linux-2.6.32.2/drivers/char/Makefile，如图添加并保存退出：

```
root@tom:/opt/FriendlyARM/mini2440/linux-2.6.29
File  Edit  View  Terminal  Tabs  Help
obj-$(CONFIG_IPMI_HANDLER)      += ipmi/

obj-$(CONFIG_HANGCHECK_TIMER)   += hangcheck-timer.o
obj-$(CONFIG_TCG_TPM)           += tpm/

obj-$(CONFIG_PS3_FLASH)         += ps3flash.o

obj-$(CONFIG_JS_RTC)            += js-rtc.o
js-rtc-y = rtc.o

obj-$(CONFIG_LEDS_MINI2440)     += mini2440_leds.o
obj-$(CONFIG_MINI2440_HELLO_MODULE) += mini2440_hello_module.o
obj-$(CONFIG_MINI2440_BUTTONS)  += mini2440_buttons.o
obj-$(CONFIG_MINI2440_BUZZER)   += mini2440_pwm.o
obj-$(CONFIG_MINI2440_ADC)      += mini2440_adc.o

# Files generated that shall be removed upon make clean
clean-files := consolemap_deftbl.c defkeymap.c

quiet_cmd_conmk = CONMK   $@
      cmd_conmk = scripts/conmakehash $< > $@

$(obj)/consolemap_deftbl.c: $(src)/$(FONTMAPFILE)
```

Step3：这时回到 linux-2.6.32.2 源代码根目录位置，执行 make modules，就可以生成我们所需要的内核模块文件 mini2440_hello_module.ko 了，如图：

至此，我们已经完成了模块驱动的编译。

```
root@tom:/opt/FriendlyARM/mini2440/linux-2.6.29
File  Edit  View  Terminal  Tabs  Help
make[1]: `include/asm-arm/mach-types.h' is up to date.
  CHK     include/linux/utsrelease.h
  SYMLINK include/asm -> include/asm-arm
  CALL    scripts/checksyscalls.sh
<stdin>:1097:2: warning: #warning syscall fadvise64 not implemented
<stdin>:1265:2: warning: #warning syscall migrate_pages not implemented
<stdin>:1321:2: warning: #warning syscall pselect6 not implemented
<stdin>:1325:2: warning: #warning syscall ppoll not implemented
<stdin>:1365:2: warning: #warning syscall epoll_pwait not implemented
  CC [M]  drivers/char/mini2440_hello_module.o
  CC [M]  drivers/scsi/scsi_wait_scan.o
  Building modules, stage 2.
  MODPOST 2 modules
  CC      drivers/char/mini2440_hello_module.mod.o
  LD [M]  drivers/char/mini2440_hello_module.ko
  CC      drivers/scsi/scsi_wait_scan.mod.o
  LD [M]  drivers/scsi/scsi_wait_scan.ko
[root@tom linux-2.6.29]# ls drivers/char/mini2440_hello_module.*
drivers/char/mini2440_hello_module.c
drivers/char/mini2440_hello_module.ko
drivers/char/mini2440_hello_module.mod.c
drivers/char/mini2440_hello_module.mod.o
drivers/char/mini2440_hello_module.o
[root@tom linux-2.6.29]#
```

7.3.3 把 Hello, Module 下载到开发板并安装使用

在此使用 rz 命令把编译出的 mini2440_hello_module.ko 下载到板子中，并把它移动到 **/lib/modules/2.6.29.4-FriendlyARM** 目录然后在板子中现在执行

#modprobe mini2440_hello_module

可以看到该模块已经被装载了(注意：使用 modprobe 命令加载模块不需要加“ko”尾缀)

再执行以下命令，可以看到该模块被卸载

#rmmod mini2440_hello_module

注意：要能够正常卸载模块，必须把模块放入开发板的 **/lib/modules/2.6.29.4-FriendlyARM** 目录

另外需要注意的是：因为我们的内核有时会升级更新，如果内核版本已经改变，请依照具体的内核版本重新建立一个模块存放目录，在此为**/lib/modules/2.6.29.4-FriendlyARM**

整个过程如下图：

7.4 简易 Linux 驱动程序示例

在上一小节，我们介绍了最简单的 Hello,Module 驱动程序模块，它只是从串口输出一些信息，并未对应板上的硬件进行操作，下面的几个例子都是和硬件密切相关的实际驱动，

在嵌入式 Linux 系统中，大部分的硬件都需要类似的驱动才能操作，比如触摸屏、网卡、音频等，在这里我们介绍的是一些简单典型的例子，实际上复杂的驱动都有参考代码，不必从头写驱动。

在本节中，我们不介绍驱动程序理论概念之类的内容，那些在网上或者书籍中都有比较系统的描述。

7.4.1 LED 驱动程序

程序源代码说明：

驱动源代码所在目录	**/opt/FriendlyARM/mini2440/linux-2.6.32.2/drivers/char**
驱动程序名称	mini2440_leds.c
设备号	Led 属于 misc 设备，设备自动生成
设备名	/dev/leds
测试程序源代码目录	**/opt/FriendlyARM/mini2440/examples/leds**
测试程序名称	led.c
测试程序可执行文件名称	led
说明：LED 驱动已经被编译到缺省内核中，因此不能再使用 insmod 方式加载。	

要写实际的驱动，就必须了解相关的硬件资源，比如用到的寄存器，物理地址，中断等，在这里，LED 是一个很简单的例子，它用到了如下硬件资源。

开发板上所用到的 4 个 LED 的硬件资源

LED	对应的 IO 寄存器名称	对应的 CPU 引脚
LED1	GPB5	K2
LED2	GPB6	L5
LED3	GPB7	K7
LED4	GPB8	K5

要操作所用到的 IO 口，就要设置它们所用到的寄存器，我们需要调用一些现成的函数或者宏，例如：s3c2410_gpio_cfgpin

为什么是 S3C2410 的呢？因为三星出品的 S3C2440 芯片所用的寄存器名称以及资源分配大部分和 S3C2410 是相同的，在目前各个版本的 Linux 系统中，也大都采用了相同的函数定义和宏定义。

它们从哪里定义？细心的用户或许很快就想到它们和体系结构有关，因此你可以在 linux-2.6.32.2/arch/arm/mach-s3c2410/include/mach/hardware.h 文件中找到该函数的定义，关于该函数的实际实现则可以在 linux-2.6.32.2/arch/arm/plat-s3c24xx/gpio.c 中找到，它的内容如下：

```
void s3c2410_gpio_cfgpin(unsigned int pin, unsigned int function)
{
    void __iomem *base = S3C24XX_GPIO_BASE(pin);
    unsigned long mask;
```

```
    unsigned long con;
    unsigned long flags;

    if (pin < S3C2410_GPIO_BANKB) {
        mask = 1 << S3C2410_GPIO_OFFSET(pin);
    } else {
        mask = 3 << S3C2410_GPIO_OFFSET(pin)*2;
    }

    switch (function) {
    case S3C2410_GPIO_LEAVE:
        mask = 0;
        function = 0;
        break;

    case S3C2410_GPIO_INPUT:
    case S3C2410_GPIO_OUTPUT:
    case S3C2410_GPIO_SFN2:
    case S3C2410_GPIO_SFN3:
        if (pin < S3C2410_GPIO_BANKB) {
            function -= 1;
            function &= 1;
            function <<= S3C2410_GPIO_OFFSET(pin);
        } else {
            function &= 3;
            function <<= S3C2410_GPIO_OFFSET(pin)*2;
        }
    }

    /* modify the specified register wwith IRQs off */

    local_irq_save(flags);

    con    = __raw_readl(base + 0x00);
    con &= ~mask;
    con |= function;

    __raw_writel(con, base + 0x00);

    local_irq_restore(flags);
}
```

实际上，我们并不需要关心这些，写驱动时只要会使用他们就可以了，除非你所使用的 CPU 体系平台尚没有被 Linux 所支持，因为大部分常见的嵌入式平台都已经有了很完善的类似定义，你不需要自己去编写。

在下面的驱动程序清单中，你可以看到 s3c2410_gpio_cfgpin 被调用的情况。除此之外，你还需要调用一些和设备驱动密切相关的基本函数，如注册设备 misc_register，填写驱动函数结构 file_operations，以及像 Hello,Module 中那样的 module_init 和 module_exit 函数等。

有些函数并不是必须的，随着你对 Linux 驱动开发的进一步了解和阅读更多的代码，你自然明白。下面是我们为 LED 编写的驱动代码清单，

驱动程序清单：

```
#include <linux/miscdevice.h>
#include <linux/delay.h>
#include <asm/irq.h>
#include <mach/regs-gpio.h>
#include <mach/hardware.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/init.h>
#include <linux/mm.h>
#include <linux/fs.h>
#include <linux/types.h>
#include <linux/delay.h>
#include <linux/moduleparam.h>
#include <linux/slab.h>
#include <linux/errno.h>
#include <linux/ioctl.h>
#include <linux/cdev.h>
#include <linux/string.h>
#include <linux/list.h>
#include <linux/pci.h>
#include <asm/uaccess.h>
#include <asm/atomic.h>
#include <asm/unistd.h>


#define DEVICE_NAME "leds"

static unsigned long led_table [] = {
    S3C2410_GPB5,
      S3C2410_GPB6,
      S3C2410_GPB7,
```

```
        S3C2410_GPB8,
};

static unsigned int led_cfg_table [] = {
        S3C2410_GPB5_OUTP,
        S3C2410_GPB6_OUTP,
        S3C2410_GPB7_OUTP,
        S3C2410_GPB8_OUTP,
};

static int sbc2440_leds_ioctl(
        struct inode *inode,
        struct file *file,
        unsigned int cmd,
        unsigned long arg)
{
        switch(cmd) {
        case 0:
        case 1:
                if (arg > 4) {
                        return -EINVAL;
                }
                s3c2410_gpio_setpin(led_table[arg], !cmd);
                return 0;
        default:
                return -EINVAL;
        }
}

static struct file_operations dev_fops = {
        .owner  =   THIS_MODULE,
        .ioctl  =   sbc2440_leds_ioctl,
};

static struct miscdevice misc = {
        .minor = MISC_DYNAMIC_MINOR,
        .name = DEVICE_NAME,
        .fops = &dev_fops,
};

static int __init dev_init(void)
```

```
{
    int ret;

    int i;

    for (i = 0; i < 4; i++) {
        s3c2410_gpio_cfgpin(led_table[i], led_cfg_table[i]);
        s3c2410_gpio_setpin(led_table[i], 0);
    }

    ret = misc_register(&misc);

    printk (DEVICE_NAME"\tinitialized\n");

    return ret;
}

static void __exit dev_exit(void)
{
    misc_deregister(&misc);
}

module_init(dev_init);
module_exit(dev_exit);
MODULE_LICENSE("GPL");
MODULE_AUTHOR("FriendlyARM Inc.");
```

7.4.2 按键驱动程序

程序源代码说明：

驱动源代码所在目录	**/opt/FriendlyARM/mini2440/linux-2.6.32.2/drivers/char**
驱动程序名称	mini2440_buttons.c
该驱动的主设备号	Misc 设备，设备号将自动生成
设备名	/dev/buttons
测试程序源代码目录	**/opt/FriendlyARM/mini2440/examples/buttons**
测试程序源代码名称	buttons_test.c
测试程序可执行文件名称	buttons
说明：按键驱动已经被编译到缺省内核中，因此不能再使用 insmod 方式加载。	

开发板所用到的按键资源如下：

按键	对应的 IO 寄存器名称	对应的中断
K1	GPG0	EINT8
K2	GPG3	EINT11
K3	GPG5	EINT13
K4	GPG6	EINT14
K5	GPG7	EINT15
K6	GPG11	EINT19

按键驱动源代码清单及注释：

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <linux/init.h>
#include <linux/delay.h>
#include <linux/poll.h>
#include <linux/irq.h>
#include <asm/irq.h>
#include <linux/interrupt.h>
#include <asm/uaccess.h>
#include <mach/regs-gpio.h>
#include <mach/hardware.h>
#include <linux/platform_device.h>
#include <linux/cdev.h>
#include <linux/miscdevice.h>

#define DEVICE_NAME		"buttons"

struct button_irq_desc {
	int irq;
	int pin;
	int pin_setting;
	int number;
	char *name;
};

static struct button_irq_desc button_irqs [] = {
	{IRQ_EINT8 , S3C2410_GPG0 ,	S3C2410_GPG0_EINT8	, 0, "KEY0"},
	{IRQ_EINT11, S3C2410_GPG3 ,	S3C2410_GPG3_EINT11 , 1, "KEY1"},
	{IRQ_EINT13, S3C2410_GPG5 ,	S3C2410_GPG5_EINT13 , 2, "KEY2"},
	{IRQ_EINT15, S3C2410_GPG7 ,	S3C2410_GPG7_EINT15 , 3, "KEY3"},
	{IRQ_EINT14, S3C2410_GPG6 ,	S3C2410_GPG6_EINT14 , 4, "KEY4"},
```

```
        {IRQ_EINT19, S3C2410_GPG11,    S3C2410_GPG11_EINT19, 5, "KEY5"},
    };

    static volatile char key_values [] = {'0', '0', '0', '0', '0', '0'};

    static DECLARE_WAIT_QUEUE_HEAD(button_waitq);

    static volatile int ev_press = 0;


    static irqreturn_t buttons_interrupt(int irq, void *dev_id)
    {
        struct button_irq_desc *button_irqs = (struct button_irq_desc *)dev_id;
        int down;

        // udelay(0);
        down = !s3c2410_gpio_getpin(button_irqs->pin);

        if (down != (key_values[button_irqs->number] & 1)) { // Changed

       key_values[button_irqs->number] = '0' + down;

            ev_press = 1;
            wake_up_interruptible(&button_waitq);
        }

        return IRQ_RETVAL(IRQ_HANDLED);
    }


    static int s3c24xx_buttons_open(struct inode *inode, struct file *file)
    {
        int i;
        int err;

        for (i = 0; i < sizeof(button_irqs)/sizeof(button_irqs[0]); i++) {
            err          =          request_irq(button_irqs[i].irq,          buttons_interrupt,
IRQ_TYPE_EDGE_BOTH,
                                button_irqs[i].name, (void *)&button_irqs[i]);
            if (err)
                break;
```

```
        }

        if (err) {
                i--;
                for (; i >= 0; i--) {
                disable_irq(button_irqs[i].irq);
                        free_irq(button_irqs[i].irq, (void *)&button_irqs[i]);
                }
                return -EBUSY;
        }

        ev_press = 1;

        return 0;
}


static int s3c24xx_buttons_close(struct inode *inode, struct file *file)
{
        int i;

        for (i = 0; i < sizeof(button_irqs)/sizeof(button_irqs[0]); i++) {
        free_irq(button_irqs[i].irq, (void *)&button_irqs[i]);
        }

        return 0;
}


static int s3c24xx_buttons_read(struct file *filp, char __user *buff, size_t count, loff_t
*offp)
{
        unsigned long err;

        if (!ev_press) {
        if (filp->f_flags & O_NONBLOCK)
                return -EAGAIN;
        else
                wait_event_interruptible(button_waitq, ev_press);
        }
```

```
        ev_press = 0;

        err = copy_to_user(buff, (const void *)key_values, min(sizeof(key_values), count));

        return err ? -EFAULT : min(sizeof(key_values), count);
}

static unsigned int s3c24xx_buttons_poll( struct file *file, struct poll_table_struct *wait)
{
        unsigned int mask = 0;
        poll_wait(file, &button_waitq, wait);
        if (ev_press)
                mask |= POLLIN | POLLRDNORM;
        return mask;
}


static struct file_operations dev_fops = {
        .owner      =      THIS_MODULE,
        .open       =      s3c24xx_buttons_open,
        .release =      s3c24xx_buttons_close,
        .read       =      s3c24xx_buttons_read,
        .poll       =      s3c24xx_buttons_poll,
};

static struct miscdevice misc = {
        .minor = MISC_DYNAMIC_MINOR,
        .name = DEVICE_NAME,
        .fops = &dev_fops,
};

static int __init dev_init(void)
{
        int ret;

        ret = misc_register(&misc);

        printk (DEVICE_NAME"\tinitialized\n");

        return ret;
}
```

```
static void __exit dev_exit(void)
{
        misc_deregister(&misc);
}

module_init(dev_init);
module_exit(dev_exit);
MODULE_LICENSE("GPL");
MODULE_AUTHOR("FriendlyARM Inc.");
```

7.5 编译 Qtopia-2.2.0

因为配置编译 Qtopia-2.2.0 的过程比较复杂，为了便于初学者学习和使用方便，我们把配置和编译的步骤制作成一个 **build** 脚本，只要执行该脚本即可编译整个 Qtopia 平台和应其自带的各个小程序；通过”**run**”脚本则可以运行它们，x86 和 arm 版本的步骤基本相同，而脚本内容则有稍微不同，下面是详细的步骤。

7.5.1 解压安装源代码

见 5.4 章节

7.5.2 编译和运行 x86 版本的 Qtopia-2.2.0

进入工作目录，执行以下命令

#cd /opt/FriendlyARM/mini2440/x86-qtopia

#./build-all　　(该过程比较长，需要运行大概 30 分钟左右)

说明：./build-all 将自动编译完整的 Qtopia 和嵌入式浏览器，你还可以先后执行**./build** 和**./build-konq** 脚本命令分别编译它们。

要运行你刚刚编译出的 Qtopia 系统十分简单，在刚刚编译完的命令终端下输入如下命令：

#./run ；注意，“/”前面有个“.”，这表示在当前目录执行

这时你可以看到如下界面

按照提示点击运行就可以看到 Qtopia 系统了，如下，
注意：我们并没有制作 x86 版本的中文系统。

7.5.3 编译和运行 arm 版本的 Qtopia-2.2.0

请确认你所使用的编译器版本为 arm-linux-gcc-4.4.3，运行平台为 Fedora 9，进入工作目录，执行以下命令

#cd /opt/FriendlyARM/mini2440/arm-qtopia

#./build-all (该过程比较长，需要运行大概 30 分钟左右)

#./mktarget (制作适用于根文件系统的目标板二进制映象文件包，将生成 **target-qtopia-konq.tgz**)

说明：./build-all 将自动编译完整的 Qtopia 和嵌入式浏览器，并且编译生成的系统支持 Jpeg、GIF、PNG 等格式的图片，您还可以先后执行**./build** 和**./build-konq** 脚本命令分别编译它们。

你可以删除开发板中原有的 Qtopia 系统，只要把/opt 目录下的所有文件都删除就可以了。然后把刚刚生成的 target-qtopia-konq.tgz 通过优盘或者其他方式解压到开发板的根目录，假定我们已经通过 ftp 把它传到了/home/plg 目录下，然后在开发板命令终端执行：

#tar xvzf /home/plg/target-qtopia-konq.tgz –C /

其中"C"是 Change 的意思，"C"后面的"/"代表要解压到根目录下，执行完毕，重启开发板，就可以看到所有的界面都已经变为英文的，并且"FriendlyARM"标签下只有一个浏览器程序，这就是你自己编译得到的整个 Qtopia 系统了，如下图：

注意：新系统可能会使用预装系统的触摸屏校正参数/etc/pointercal，你也可以在删除旧系统的时候，一并删除它，这样开机后就会进入校正界面了。

上面的过程看似很简单，其实所有的秘密都在 build-all 脚本中，网上也有很多关于移植的文章，但本质的步骤都是脚本所记录的那些，你可以使用记事本工具打开自行查看一下，只不过其中没有那些嘻哈打诨的辞藻罢了。

7.6 编译 QtE-4.6.3

7.6.1 解压安装源代码

见 5.4 章节

7.6.2 编译和运行 arm 版本的 QtE-4.6.3

注意，请务必使用我们光盘中提供的交叉编译器 arm-linux-gcc-4.4.3，并基于 Fedora9 平台，Fedora9 的安装步骤见 5.1，我们不能保证所制作的脚本可以在其他平台环境下顺利执行，对此我们也不提供支持。

和 Qtopia-2.2.0 十分类似，我们也为 QtE-4.6.3 的编译制作了现成的脚本 build-all，进入源代码目录执行：

#cd /opt/FriendlyARM/mini6410/linux/arm-qte-4.6.3

#./build-all

这个过程将十分漫长，根据机器配置不同，会有不同的编译时间，请耐心等待。

当顺利执行完毕，再运行 mktarget 脚本，将会从编译好的目标文件目录中，提取出必要的 QtE-4.6.3 库文件和可执行二进制示例，并打包为 **target-qte-4.6.3.tgz**，把它在开发板的根目录下解压，就可以使用了，如下命令

#tar xvzf target-qte-4.6.3.tgz –C /

这样，就会在/usr/local/目录下创建生成 Trolltech 目录，它里面包含了运行所需要的所有库文件和可执行程序。

使用说明：因为目标板中预装的 Linux 已经包含了 QtE-4.6.3，测试之前，你可以先删除原有的，它位于**/usr/local/Trolltech** 目录，把该目录使用 **rm** 命令完全删除就可以了。

要运行 QtE-4..6.3，建议先停止正在运行的 Qtopia-2.2.0，点“设置”中的“关机”可出现如下界面，点“Terminate Server”即可关闭 Qtopia-2.2.0 系统。

也可以使用其他的方法，比如在启动脚本/etc/init.d/rcS 中注释掉 qtopia 启动项，再重新系统系统；或者使用 killall 命令杀死相关的进程(比较多；甚至是直接删除/opt 目录中的所有内容重启。

关闭 qtopia-2.2.0 之后，在命令行输入 qt4 命令，即可启动刚刚解压安装的 QtE-4.6.3 了，如图：

Path Stroking

第八章 常见 bootloader 的配置和编译

在 S3C2440/2410 系统中，常见的 bootloader 一般有

- Vivi – 由三星提供，韩国 mizi 公司原创，开放源代码，必须使用 arm-linux-gcc 进行编译，目前已经基本停止发展，主要适用于三星 S3C24xx 系列 ARM 芯片，用以启动 Linux 系统，支持串口下载和网络文件系统启动等常用简易功能。
- Supervivi – 由友善之臂提供并积极维护，它基于 vivi 发展而来，不提供源代码，在保留原始 vivi 功能的基础上，整合了诸多其他实用功能，如**支持 CRAMFS，YAFFS 文件系统**，USB 下载，自动识别并启动 Linux，WinCE, uCos，Vxwork 等多种嵌入式操作系统，下载程序到内存中执行，并**独创了系统备份和恢复功能**，非常适合在批量生产中使用，是目前 2440/2410 系统中功能最强大最好用的 bootloader。
- Vboot – 由友善之臂制作并开源提供，它的功能很简单，只是启动 Linux 系统，vboot 可以自动适应支持 64M/128M Nand Flash 的 mini2440/micro2440 板。
- YL-BIOS – 深圳优龙基于三星的监控程序 24xxmon 改进而来，**提供源代码，可以使用 ADS 进行编译**，整合了 USB 下载功能，仅支持 CRAMFS 文件系统，并增加了手工设置启动 Linux 和 WinCE，下载到内存执行测试程序等多种实用功能。因其开源性，故该 bootloader 被诸多其他嵌入式开发板厂商所采用，需要注意的是大部分是未经优龙公司授权的。
- U-Boot – 一个开源的专门针对嵌入式 Linux 系统设计的最流行 bootloader，必须使用 arm-linux-gcc 进行编译，具有**强大的网络功能(失去网络，U-Boot 基本丧失其最独到的优势，在 2440/2410 系统中网络是添加网络芯片外扩的，毫无疑问会增加成本，而 USB 是内置的，只需几个电阻配置)**，支持网络下载内核并通过网络启动系统，U-Boot 处于更加活跃的更新发展之中，但对于 2440/2410 系统来说，它尚未支持 Nand Flash 启动，国内已经有人为此自行加入了这些功能，本章节中的 U-Boot 即是如此。

Bootloader 以其本身的含义来讲就是下载和启动系统，它类似于 PC 中的 BIOS，大部分芯片厂商所提供的嵌入式系统都提供有这样的程序，而且都比较成熟，大可不必自行编写。我们所改进的 supervivi 目标是使之更加人性化，更加适合于批量生产需要。

下面我们使用图解的方式介绍一下那些开源 BIOS 的配置和编译过程，其使用方法暂时不做介绍。

8.1 编译 vboot

在第五章节中我们已经解压安装好了 vivi 的源代码，它位于 **/opt/FriendlyARM/mini2440/vboot** 目录中。

编译 vboot 十分简单，进入该目录，运行以下命令：

#cd /opt/FriendlyARM/mini2440/vboot

#make

将会在当前目录下生成 vboot.bin，它和光盘中提供的是完全一样的目标文件，如图：

```
root@tom:/opt/FriendlyARM/mini2440/vboot
File  Edit  View  Terminal  Tabs  Help
244x_lib.h  def.h       head.S      Makefile  nand.c   parameters.h
[root@tom vboot]#
[root@tom vboot]# pwd
/opt/FriendlyARM/mini2440/vboot
[root@tom vboot]# ls
244x_lib.c  def.h       main.c    nand.c        s3c2440.h
244x_lib.h  hardware.h  Makefile  nand.h        smdk2440.h
bitfield.h  head.S      mem.lds   parameters.h
[root@tom vboot]# make
arm-linux-gcc -mabi=aapcs-linux -mno-thumb-interwork -Os -Wall -c hea
d.S 244x_lib.c nand.c main.c
arm-linux-ld -T mem.lds -Bstatic head.o 244x_lib.o nand.o main.o
arm-linux-objcopy -O binary -S a.out vboot.bin -R .comment -R .stab -
R .stabstr
rm *.o a.out
[root@tom vboot]# ls
244x_lib.c  def.h       main.c    nand.c        s3c2440.h
244x_lib.h  hardware.h  Makefile  nand.h        smdk2440.h
bitfield.h  head.S      mem.lds   parameters.h  vboot.bin
[root@tom vboot]# ls -l vboot.bin
-rwxr-xr-x 1 root root 3143 2009-07-19 11:05 vboot.bin
[root@tom vboot]#
```

此时已经在当前目录下生成了 vboot.bin，您可以参考前面的章节把 vboot.bin 烧写到目标板的 Nand Flash 用以启动 Linux 系统。

8.2 配置和编译 vivi

注意：本开发板的所有 Linux 软件源代码均使用统一的编译器 arm-linux-gcc-4.3.2，其安装和设置见第五章节；本小节的 vivi 仅适合于 64M Nand Flash 版本的 mini2440/micro2440。

在第五章节中我们已经解压安装好了 vivi 的源代码，它位于 **/opt/FriendlyARM/mini2440/bootloader/vivi** 目录中。

进入该目录，运行以下命令：

#cd /opt/FriendlyARM/mini2440/bootloader/vivi

#cp fa.config .config ；使用缺省友善之臂提供的配置文件 fa.config

#make menuconfig

出现以下界面：

不需要更改任何配置，按左右方向键，选择 **<Exit>**，如图：

选择**<Yes>**，按回车退出，这样做是为了按照缺省配置自动生成头文件。然后执行“**make**”开始编译，执行结果如下：

#make

```
root@tom:/opt/FriendlyARM/mini2440/bootloader/vivi
File  Edit  View  Terminal  Tabs  Help
40'
make[1]: Leaving directory `/opt/FriendlyARM/mini2440/bootloader/vivi/arch/s3c24
40'
/usr/local/arm/4.3.2/bin/arm-linux-gcc -D__ASSEMBLY__ -I/opt/FriendlyARM/mini244
0/bootloader/vivi/include -I/usr/local/arm/4.3.2/arm-none-linux-gnueabi/libc/usr
/include -msoft-float   -c -o arch/s3c2440/head.o arch/s3c2440/head.S
arch/s3c2440/head.S:453:8: warning: extra tokens at end of #endif directive
/usr/local/arm/4.3.2/bin/arm-linux-ld -v -Tarch/vivi.lds -Bstatic  \
                arch/s3c2440/head.o \
                arch/s3c2440/s3c2440.o init/main.o init/version.o lib/lib.o \
                 drivers/serial/serial.o drivers/mtd/mtd.o \
                lib/priv_data/priv_data.o \
                -o vivi-elf -L/usr/local/arm/4.3.2/arm-none-linux-gnueabi/libc/a
rmv4t/lib -L/usr/local/arm/4.3.2/arm-none-linux-gnueabi/libc/armv4t/usr/lib -L/u
sr/local/arm/4.3.2/lib/gcc/arm-none-linux-gnueabi/4.3.2/armv4t/ -lgcc -lc
GNU ld (Sourcery G++ Lite 2008q3-72) 2.18.50.20080215
/usr/local/arm/4.3.2/bin/arm-linux-nm -v -l vivi-elf > vivi.map
/usr/local/arm/4.3.2/bin/arm-linux-objcopy -O binary -S vivi-elf vivi -R .commen
t -R .stab -R .stabstr
[root@tom vivi]# ls
arch        drivers     init        net         test   vivi-elf
ChangeLog   fa.config   lib         Rules.make  util   vivi.map
COPYING     include     Makefile    scripts     vivi
[root@tom vivi]#
```

此时已经在当前目录下生成了 vivi，您可以参考前面的章节把 vivi 烧写到目标板的 Nand Flash 运行。

8.3 配置和编译 U-Boot

说明：本小节只介绍 U-Boot 的配置编译和烧写，关于其使用方法的详细介绍，我们将会在以后的手册更新中添加，用户也可以自行到网上查找相关资料。

本光盘中的 U-Boot 具有以下功能特性：

1. 同时支持 S3C2410 和 S3C2440
2. 支持串口 xmodem 协议
3. 支持 USB 下载，可以在 PC 上使用 dnw 传数据
4. 支持网卡芯片 CS8900
5. 支持 NAND Flash 读写
6. 支持从 Nor/Nand Flash 启动
7. 支持烧写 yaffs 文件系统映象
8. 可以直接下载到内存运行
9. 即可以支持 CS8900，又可以支持 DM9000，但是，不能同时支持；要选择支持哪个网卡芯片，需要在 include/configs/100ask24x0.h 中进行配置，如下：

```
#if 0
#define CONFIG_DRIVER_CS8900   1     /* we have a CS8900 on-board */
#define CS8900_BASE          0x19000300
#define CS8900_BUS16         1 /* the Linux driver does accesses as shorts */
#endif
```

```
#if !defined(CONFIG_DRIVER_CS8900)
#define CONFIG_DRIVER_DM9000        1
#define CONFIG_DM9000_USE_16BIT     1
#define CONFIG_DM9000_BASE          0x20000000
#define DM9000_IO                   0x20000000
#define DM9000_DATA                 0x20000004
#endif
```

下面是具体的编译方法和烧写步骤。

8.3.1 配置和编译 U-Boot

注意：本开发板的所有 Linux 软件源代码均使用统一的编译器 arm-linux-gcc-4.3.2，其安装和设置见第五章节。

在第五章节中我们已经解压安装好了 u-boot 的源代码，它位于 **/opt/FriendlyARM/mini2440/bootloadder/u-boot-1.1.6** 目录中。

进入该目录，运行以下命令：

```
#cd /opt/FriendlyARM/mini2440/bootloader/u-boot-1.1.6
#make open24x0_config        ；配置 U-Boot
#make
```

就可以开始编译了，编译完毕，如图所示生成 u-boot.bin

8.3.2 把 U-Boot 烧写到开发板

要烧写 U-Boot，需要把开发板拨动开关 S2 设置为 Nor Flash 启动，连接好串口和 USB 线，打开超级终端，打开电源，串口显示如图：

选择功能号“a”，打开 DNW，确认 USB 连接正常 OK，点 UsbPort→Transmit/Restore，选择刚才所编译的 u-boot.bin，下载和烧写很快就会结束。

把开发板启动模式改为 Nand Flash 启动，重新复位或者重启开机电源开关，在串口终端可以看到如图信息，如果开发板中已经安装了 linux 系统，U-Boot 将会自动启动它，否则会进入 U-Boot 的功能菜单(也可以根据提示，在开机后 3 秒中内按任意键进入)：

U-Boot 的功能菜单如图，您可以按照功能选项进行测试，它和 supervivi 基本是类似的.

注意：本开发板提供的 linux 内核并不能直接用于 u-boot，因为我们公司是不使用 u-boot 的，并且对其各个参数设置并不了解，关于 U-Boot 的使用方法用户可以参考网上的资料。

8.4 使用 ADS 编译 YL-BIOS

注意：要编译使用优龙的 BIOS，首先要安装好 ADS 开发环境。

说明：对于 YL-BIOS 我们没有做太多测试和说明，毕竟这是第三方 BIOS，也非主流，在此仅作参考，本公司不提供关于该 BIOS 的任何技术支持和咨询。

8.4.1 使用 ADS 编译 YL-BIOS

(1)YL-BIOS2440 源代码位于光盘的/OpenSourceBootloader 文件夹中，把它复制到您的电脑中，如“D:\work”，去掉其只读属性。

(2)使用 ADS 打开 YL2440A_MON.mcp 文件：

(3)不要修改任何设置，直接点击 Project→Make 或者按 F7 键开始编译，最后在“D:\work\YL-BIOS2440\YL2440A_MON_Data\DebugRel”目录下生成 yl-bios2440.bin 文件：

8.4.2 把 YL-BIOS 下载到内存中运行

把开发板拨动开关 S2 设置为 Nor Flash 启动，并打开超级终端，进入功能菜单模式，并输入功能命令“d”，如图：

打开运行 DNW，设置下载地址为 0x30100000 (这是 YL-BIOS 的默认运行地址)：

点 Usb Port → Transmit/Restore，并选择刚才所编译出的 yl-bios2440.bin 开始通过 USB 下载到内存中：

下载很快结束，这时串口终端出现如图所示：

按一下空格键就可以进入 YL-BIOS 的菜单了，如图：

8.4.3 烧写 YL-BIOS 到开发板

YL-BIOS 还可以直接烧写到 Nand Flash 中运行使用。

(1)和上面的步骤类似，首先确定设置开发板为 Nor Flash 启动模式，打开超级终端进入 supervivi 的 BIOS 模式，并输入功能命令“a”：

(2)打开 DNW，确认 USB 连接正常 OK，点 UsbPort→Transmit/Restore，选择刚才所编译的 yl-bios2440.bin，下载和烧写很快就会结束。

(3)把开发板启动模式跳线改为 Nand Flash 启动，重新复位或者重启开机电源开关，在串口终端可以看到如图信息：

按任意键进入 YL-BIOS 的菜单模式，如图：

注意事项：

至此你已经可以编译、烧写并启动了 YL-BIOS，在此之后你在 Nand Flash 中将使用一个全新的第三方 BIOS，你不能再继续使用 supervivi 烧写内核和文件系统，并用 YL-BIOS 启动它们，因为其采用的一些参数和友善之臂并不相同；你必须按照 YL-BIOS 的功能手册来操作剩余的部分，关于如何进一步使用 YL-BIOS 请用户自行查找资料解决。

当然你也可以使用 SJF2440 工具软件来烧写 YL-BIOS，那将会十分慢，但不能使用 H-JTAG，因为它尚不支持 Nand Flash 烧写。

第九章 WindowsCE 6.0 开发指南

9.1 建立 WindowsCE 6.0 开发环境

注意：以下软件和步骤均基于 Microsoft Windows 7 系统（旗舰版），其他 Windows 系统未经测试。建议把安装软件复制到硬盘安装(ISO 光盘映象文件可借助虚拟光驱)

Windows CE 6.0 的安装过程十分繁琐，并且对开发主机的要求比较高(否则会很慢)，我们建议用户特别是初学者务必按照我们介绍的步骤安装开发环境。

这里是我们采用的开发主机的关键配置，仅供参考：

CPU：Intel Core Duo E8400

内存：DDR2 4GB

硬盘空间：500GB

安装所需的软件列表如下(本公司并不提供 Windows Embedded 6.0 CE 6 的安装文件，用户可以到微软网站自行下载它的试用版)：

- ✓ Visual Studio 2005
  (试用版下载地址：
  http://download.microsoft.com/download/e/1/4/e1405d9e-47e3-404c-8b09-489437b27fb0/En_vs_2005_Pro_90_Trial.img)
- ✓ Visual Studio 2005 Service Pack 1(文件名：VS80sp1-KB926601-X86-ENU.exe)

下载地址：
http://www.microsoft.com/downloads/details.aspx?familyid=bb4a75ab-e2d4-4c96-b39d-37baf6b5b1dc&displaylang=en

- ✓ Visual Studio 2005 Service Pack 1 Update for Windows Vista
  (文件名：VS80sp1-KB932232-X86-ENU.exe)
  下载地址：
  http://www.microsoft.com/downloads/details.aspx?FamilyID=90E2942D-3AD1-4873-A2EE-4ACC0AACE5B6&displaylang=en)
- ✓ Visual Studio 2005 Service Pack 1 ATL Security Update
  (文件名：VS80sp1-KB971090-X86-INTL.exe)
  下载地址：
  http://www.microsoft.com/downloads/details.aspx?familyid=7C8729DC-06A2-4538-A90D-FF9464DC0197&displaylang=en
- ✓ Windows Embedded CE 6.0
  试用版下载地址：
  http://www.microsoft.com/downloads/details.aspx?displaylang=en&FamilyID=7e286847-6e06-4a0c-8cac-ca7d4c09cb56
- ✓ Windows Embedded CE 6.0 Platform Builder Service Pack 1
  下载地址：
  http://www.microsoft.com/downloads/details.aspx?FamilyId=BF0DC0E3-8575-4860-A8E3-290ADF242678&displaylang=en
- ✓ Windows Embedded CE 6.0 R2
  下载地址：
  http://www.microsoft.com/downloads/details.aspx?FamilyId=F41FC7C1-F0F4-4FD6-9366-B61E0AB59565&displaylang=en
- ✓ Windows Embedded CE 6.0 R3
  下载地址：
  http://www.microsoft.com/downloads/details.aspx?FamilyID=BC247D88-DDB6-4D4A-A595-8EEE3556FE46&displaylang=ja&displaylang=en
- ✓ 腾讯 QQ(第三方软件)
  下载地址：
  http://www.microsoft.com/downloads/details.aspx?FamilyID=527042f7-bb5b-4831-a6ad-5081808824ec&displaylang=en
- ✓ WesttekFileViewers6.exe(office 文件浏览器，亦属于第三方软件)
  下载地址：
  http://www.microsoft.com/downloads/details.aspx?FamilyID=d2fd14eb-7d5c-428b-951c-343f910047c1&displaylang=en

以上列表顺序基本也说明了这些软件的安装顺序：先安装 Visual Studio 2005 及补丁，再安装 Windows CE 6.0 及补丁，最后安装第三方软件。

说明：Windows CE 6.0 所使用的 Platform Builder 和以往的 Windows CE 5.0/4.2 等均不同，它并不是独立的开发平台软件，而是作为 VS2005 的一个插件来安装使用的，因此必须先安装 VS2005，以后所有的内核配置编译等开发都基于 VS2005 进行。

下面是详细的步骤。

9.1.1 安装 Visual Studio 2005 及补丁

Step1：打开 Visual Studio 2005 文件夹，找到 setup.exe，双击运行开始安装。

Step2：出现如图界面，点“Install Visual Studio 2005”，继续

Step3：出现如图界面，稍等片刻，点“Next”继续

Step4：出现如图界面，注意点选红色框的，并输入序列号，点“Next”继续

Step5：出现如图界面，选择安装类型，在此选择完全安装，即“Full”，点“Next”继续

Step6：出现如图界面，开始正式安装 Visual Studio 2005，此过程较长，请耐心等待。

Step7：Visual Studio 2005 安装完毕，出现如下画面，点“Finish”结束安装。

Windows Embedded CE 6.0 Setup
Completing the Windows Embedded CE 6.0 Setup Wizard
Click Finish to exit the Setup Wizard.
< Back
Finish
Cancel

接着会出现如图界面，点“Exit”退出即可。

Step8：现在开始安装第一个补丁文件 Visual Studio 2005 Service Pack 1，双击运行 VS80sp1-KB926601-X86-ENU.exe 开始安装，出现如图界面

Step9：须稍等片刻，出现如图画面，点“OK”开始正式安装

Step10：接受安装许可协议，点“I accept”继续

Step11：出现安装过程界面，此过程较长，请耐心等待

Step12：安装完毕，出现如下界面，点“OK”结束本补丁的安装

Step13：接下来安装第二个补丁 Visual Studio 2005 Service Pack 1 Update for Windows Vista，双击运行 VS80sp1-KB932232-X86-ENU.exe，依次出现如图界面

Step14：稍等片刻，出现如图界面，点“OK”继续

Step15：出现安装许可协议界面，点“I accept”继续

Step16：出现安装过程界面，此过程较长，请耐心等待

Step17：安装完毕，出现如下界面，点“OK”结束本补丁的安装

Step18：接下来安装第三个补丁 Visual Studio 2005 Service Pack 1 ATL Security Update，双击运行 VS80sp1-KB971090-X86-INTL.exe，依次出现如图界面

Step19：稍等片刻，出现如图界面，点“OK”继续

Step20：出现安装许可协议界面，点“I accept”继续

Step21：出现安装过程界面，此过程较长，请耐心等待

Step22：安装完毕，出现如下界面，点“OK”结束本补丁的安装

至此，基于 Windows7 平台的 Visual Studio 2005 及其补丁已经完全安装完毕。

9.1.2 安装 Windows CE 6.0 及补丁

接下来开始安装 Windows CE 6.0 的 Platform Builder，注意：在 Windows 7 系统上安装 Windows CE 6.0 及其补丁需要管理员的权限来安装，请不要双击运行安装文件执行安装，否则到后面无法安装成功，请按照如下步骤启动安装文件。

Step1：点“开始”->“程序”->“附件”，找到“命令行提示符”，然后点右键出现菜单，点“以管理员身份运行”，如图

Step2：出现命令行窗口，进入相应的安装目录，并输入安装程序名“Windows Embedded CE 6.0.msi”，开始安装，如图

Step3：出现如图界面，点“Next”继续

Step4：输入产品密钥，点“Next”继续

Step5：出现安装许可协议界面，选择“I accept”，点“Next”继续

Step6：选择及设置如图，点“Next”继续

Step7：出现如图界面，选择如图，点“Next”继续

Step8：出现如图界面，点“Install”继续

Step9：开始正式安装，如图，此过程时间较长，请耐心等待

Step10：安装结束，出现如图界面，点“Finish”结束安装。

Step11：接下来安装Windows CE 6.0的第一个补丁“Windows Embedded CE 6.0 Platform Builder Service Pack 1.msi”，按照本小节开头 Step1 的方法以管理员的身份进入命令行窗口，并进入相应的目录，输入“Windows Embedded CE 6.0 Platform Builder Service Pack 1.msi”开始安装，如图

Step12：出现如图界面，点“Next”继续

Step13：出现如图界面，选“I accept”，并点“Next”继续

Step14：出现如图界面，点“Next”继续

Step15：开始正式安装，如图，此过程时间较长，请耐心等待

Step16：安装结束，出现如图界面，点“Finish”结束安装。

Step17：接下来安装 Windows CE 6.0 的第二个补丁“Windows Embedded CE 6.0 R2.msi”，按照本小节开头 Step1 的方法以管理员的身份进入命令行窗口，并进入相应的目录，输入“Windows Embedded CE 6.0 R2.msi”开始安装，如图

说明：有的用户可能会下载到单独的 Windows Embedded CE 6.0 R2.msi”安装文件，大小为 50MB 左右，但此补丁似乎是不完整的，在安装时可能会遇到缺少“help.cab”文件的问题，导致无法顺利安装，因此我们建议用户使用我们提供的 R2 补丁，它总共 122 个文件，大概 1.01GB。

Step18：出现如图界面，点“Next”继续

Step19：出现如图界面，选“I accept”，并点“Next”继续

Step20：出现如图界面，不用做任何改动，点“Next”继续

Step21：出现如图界面，点“Next”继续

Step22：开始正式安装，此过程时间较长，请耐心等待

Step23：安装结束，出现如图界面，点“Finish”结束安装

Step24：现在开始安装 Windows CE 6.0 的第三个补丁 R3，按照本小节开头 Step1 的方法以管理员的身份进入命令行窗口，并进入相应的目录，输入“Windows Embedded CE 6.0 R2.msi”开始安装，如图

说明：有的用户可能会下载到单独的 Windows Embedded CE 6.0 R3”安装文件，它其实是一个光盘映象文件，为了用户使用方便，我们把它提取出来，做成普通的目录文件方式，它总共有 166 个文件，大小大概是 1.14GB

Step25：出现如图界面，点“Next”继续

Step26：出现如图界面，选“I accept”，并点“Next”继续

Step27：出现如图界面，点“Next”继续

Step28：开始正式安装，此过程时间较长，请耐心等待

Step29：安装结束，出现如图界面，点“Finish”结束安装

9.1.3 安装第三方软件腾讯 QQ

在 Windows CE 6.0 R3 补丁中，微软还正式提供了可选的第三方的软件，分别有腾讯 QQ 和 File Viewers，我们已经把它们放入资料光盘，用户也可以在微软网站下载到它们，地址如下：

http://www.microsoft.com/downloads/details.aspx?displaylang=en&FamilyID=bc247d88-ddb6-4d4a-a595-8eee3556fe46#filelist (此地址有可能会失效，请用户到微软网站自行查询)

我们后面的例子中，实际中用到腾讯 QQ，因此我们只安装 QQ，其他软件用户可以自行安装测试。

说明：

Step1：进入 QQ 安装目录，双击运行 setup.exe 开始安装，如图

Step2：出现如图界面，点“Next”继续

Step3：使用缺省配置，不作任何改动，点“Next”继续

Step4：出现如图界面，选择“I accpet”，点“Next”继续

Step5：出现如图界面，点“Next”继续

Step6：出现如图界面，稍等片刻

Step7：出现如图界面，点“Close”结束安装

9.1.4 安装 BSP 及内核工程示例

Mini2440 的BSP和示例工程等文件只有一个安装文件mini2440-ce6-suite-1033.exe(尾缀 1033 是年周日期版本标识)，其中包含所有的BSP源代码以及两个内核工程示例，用户可以在 http://www.arm9.net网站点“下载”查找最新的版本，下面是详细的安装过程。

注意：请以以下步骤安装 BSP，建议不要改变路安装径，否则有可能无法编译通过。

Step1：找到 mini2440-ce6-suite-1033.exe 可执行安装文件，并双击运行

Step2：保持各项设置不变，点“Install”继续

Step3：出现安装过程界面，因为安装的文件很小，安装会很快结束

Step4：安装结束，出现如图界面，点“Close”结束安装

安装完毕，会在 WinCE600\PLATFORM 目录下创建 mini2440 BSP 目录，如图

并在 WinCE600\OSDesigns 目录下分别创建三个内核示例工程文件目录，如图

其中:

Mini2440 目录中包含的工程文件，可以用来编译生成光盘中对应的中文版 WinCE 内核映像

Mini2440-with-QQ 目录中的工程文件，可以用来编译生成包含腾讯 QQ 的 WinCE 内核映像

Mini2440-en 目录中包含的工程文件，可以用来生成英文版的 WinCE 内核映像

至此，Windows CE 6.0 的开发环境就已经完全创建了。

9.1.5 各个驱动程序源代码的位置

Mini2440 目前拥有最齐全的 BSP，也就是驱动程序，并且每个驱动基本都有相应的图形界面测试程序。各个驱动程序的源代码位置说明如下：

(1) LED 驱动

\Mini2440\SRC\DRIVERS\LEDdriver

(2) 按键驱动

\Mini2440\SRC\DRIVERS\Userkey

(3)PWM 控制蜂鸣器驱动

\Mini2440\SRC\DRIVERS\PWM

(4)ADC 转换驱动

\Mini2440\SRC\DRIVERS\Touch

说明：ADC 驱动实际和触摸屏驱动在同一个文件中实现

(5)I2C 驱动

\Mini2440\SRC\DRIVERS\IIC

(6)RTC 驱动

\Mini2440\SRC\COMMON\Rtc

说明：RTC 驱动的位置和其他有所不同，这里遵循该 BSP 原有的目录结构

(7)串口驱动(包含三个串口)

\Mini2440\SRC\DRIVERS\Serial

(8)触摸屏驱动

\Mini2440\SRC\DRIVERS\Touch

(9)USB 驱动

\Mini2440\SRC\DRIVERS\Usb

说明：包含 USB Slave 和 USB Host(可连接 USB 鼠标、键盘、优盘等外设)

(10)SD 卡驱动

\Mini2440\SRC\DRIVERS\SDHC

说明：支持高速大容量 SD 卡，最高可达 32GB

(11)DM9000 网卡驱动

\Mini2440\SRC\DRIVERS\dm9000

(12)音频驱动

\Mini2440\SRC\DRIVERS\Wavedev

说明：该驱动支持录音和放音

(13)LCD 驱动

\Mini2440\SRC\DRIVERS\Display

(14)背光驱动

\Mini2440\SRC\DRIVERS\Backlight

说明：该背光仅实现开关，并无调节作用，主要是开发板的硬件目前并不支持背光调节

(15)CMOS 摄像头驱动

\Mini2440\SRC\DRIVERS\Camera

9.2 配置和编译 WindowsCE 6.0 内核及 Bootloader

因为 Windows CE6 的内核配置比较复杂，很容易因配置不对而导致无法编译通过，众所周知 Windows CE 平台的编译是十分耗时的，因此我们特意制作了 2 个内核工程示例，以便供用户参考，用户按照下面的步骤直接打开编译就可以了，光盘中 images\wince6.0 目录中有相应的编译好的内核映象文件。

需要说明的是，因据此 BSP 编译出的内核映象不能大于 30MB，而腾讯 QQ 所占用的空间又比较大，因此我们制作了 2 个示例工程：mini2440 和 mini2440-with-QQ，前者包含了一些常见的配置选项，后者删除了 SQL 相关组件，添加了腾讯 QQ，其他部分则是大部分相同的，用户可自行对比。我们推荐用户使用 mini2440(以下均称为缺省内核工程示例)，毕竟在开发板上安装腾讯 QQ 只是测试性质的。

9.2.1 缺省内核工程特性简介

缺省配置的内核选择了用户常见的一些特性，大致如下：

9.2.2 编译缺省内核工程示例

现在，我们启动 VS2005 来编译刚刚安装的 mini2440 BSP，第一次启动 VS2005 时有些事项要注意一下，如下步骤：

Step1：点“开始”->“程序”->“Microsoft Visual Studio 2005”-> “Microsoft Visual Studio 2005”(下称 VS2005)，如图

Step2：这时会出现如下提示窗口，请先不要点“Continue”，在这里微软建议你采用管理员身份运行该程序，因此点“Exit Visual Studio”退出。

Step3：我们先把 VS2005 设置为管理员执行权限，点“开始”->“程序”->“VS2005”->“VS2005”，然后右键出现如图菜单，点“属性”

Step4：出现如图窗口，点“兼容性”选项卡，并作如图勾选，点“确定”返回。

Step5：这时再点“开始”->“程序”->“VS2005”->“VS2005”，又会出现刚才的提示窗口，如图，点“Contonue”继续，此时将以管理员身份运行 VS2005

Step6：出现如图界面，这是 VS2005 的工作界面，在此就不再对该界面赘述了，请用户参考常用的 VS2005 资料即可

Step7：点 File->Open->Project/Solution…，如图

Step8：出现文件选择窗口，找到 mini2440 的缺省内核项目文件(路径为：C:\WINCE600\OSDesigns\Mini2440)，点“Open”打开它，如图

Step9：稍等片刻，mini2440 的缺省内核项目被载入工作区，出现如图界面

Step10：点“Build->Advanced Build Commands->Clean Sysgen”开始编译内核，如图，此过程较长，请耐心等待

Step11：编译完毕，结果如图所示，此时会生成内核映象文件 NK.bin 和 NK.nb0，路径如下：C:\WINCE600\OSDesigns\Mini2440\Mini2440\RelDir\Mini2440_ARMV4I_Release

Mini2440 - Microsoft Visual Studio
File Edit View Project Build Debug Target Tools Window Community Help
Mini2440 Af
Platform Builder (_TGTC
Device: CE Device
Solution Explorer - Mini2440
Solution 'Mini2440' (1 project)
Mini2440
C:/WINCE600
PLATFORM
ARUBABOARD
CEPC
COMMON
DEVICEEMULATOR
H4SAMPLE
MAINSTONEIII
Solution Explorer
Catalog Items View
Class View
Start Page
Microsoft Visual Studio 2005
Recent Projects
Mini2440
MSDN: Windows Embedded Developer Center
Windows Embedded Partner Program
Tue, 22 Jan 2008 14:56:53 -0800 - Accelerate your sc
and your business by connecting with partners, OEM
Distributors, and Microsoft.
Download Windows Embedded CE 6.0 Platform B
Output
Show output from: Build
makeimg: Change directory to C:\WINCE600.
makeimg: run command: cmd /C C:\WINCE600\public\common\oak\misc\pbpostmakeimg
驱动器 C 中的卷没有标签。
卷的序列号是 A89F-95D1
C:\WINCE600\OSDesigns\Mini2440\Mini2440\RelDir\Mini2440_ARMV4I_Release 的目录
2009/01/26 17:01 30,462,591 NK.bin
1 个文件 30,462,591 字节
0 个目录 79,042,859,008 可用字节
BLDDEMO: Mini2440 build complete.
Mini2440 - 0 error(s), 64 warning(s)
========== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==========
Code Definition Window
Call Browser
Output
Build succeeded
Ln 2394
Col 1
Ch 1

9.2.3 编译带腾讯 QQ 的内核工程示例

编译带腾讯 QQ 支持的内核工程和上面的步骤类似，只不过不必再重新设置 VS2005 “管理员执行权限”的属性，步骤如下

Step1：点“开始->程序->VS2005->VS2005”，打开 VS2005 工作界面

Step2：点 File->Open->Project/Solution…，如图

Step3：出现文件选择窗口，找到 mini2440 的缺省内核项目文件(路径为：C:\WINCE600\OSDesigns\Mini2440-with-QQ)，点“Open”打开它，如图

Step4：稍等片刻，mini2440 的缺省内核项目被载入工作区，出现如图界面

Step5：点“Build->Advanced Build Commands->Clean Sysgen”开始编译内核，如图，此过程较长，请耐心等待

Step6：编译完毕，结果如图所示，此时会生成内核映象文件 NK.bin 和 NK.nb0，路径如下：C:\WINCE600\OSDesigns\Mini2440-with-QQ\Mini2440-with-QQ\RelDir\Mini2440_ARMV4I_Release

Mini2440-with-QQ - Microsoft Visual Studio
File Edit View Project Build Debug Target Tools Window Community Help
Mini2440 AF
Platform Builder (_TGTC
Device: CE Device
Solution Explorer - Mini2440-with-QQ
Mini2440-with-QQ
C:/WINCE600
PLATFORM
ARUBABOARD
CEPC
COMMON
DEVICEEMULATOR
Solution Ex...
Catalog Ite...
Class View
Start Page
Microsoft Visual Studio 2005
Recent Projects
Mini2440-with-QQ
Mini2440
MSDN: Windows Embedded Dev
Windows Embedded Partner Pro
Tue, 22 Jan 2008 14:56:53 -0800 -
and your business by connecting
Output
Show output from: Build
卷的序列号是 A89F-95D1
C:\WINCE600\OSDesigns\Mini2440-with-QQ\Mini2440-with-QQ\RelDir\Mini2440_ARMV4I_Release 的目录
2009/01/26 17:32 24,739,219 NK.bin
1 个文件 24,739,219 字节
0 个目录 78,387,290,112 可用字节
BLDDEMO: Mini2440-with-QQ build complete.
Mini2440-with-QQ - 0 error(s), 66 warning(s)
========== Build: 1 succeeded or up-to-date, 0 failed, 0 skipped ==========
Code Definition Window
Call Browser
Output
Build succeeded
Ln 2512
Col 1
Ch 1
INS

9.2.4 编译和烧写 BootLoader 之 NBOOT

说明：我们提供的 Nboot 可同时适用于 WindowsCE5/6，因此在光盘中的所有 Nboot 项目源代码都是相同的，只不过为了方便组织使用，特意增加了拷贝，并分别放在了不同的目录。

编译 Nboot 需要使用 ADS 集成开发环境，详细的步骤见本手册第四章，在 Windows 7 系统上，你可以通过安装 Windows XP Mode 来创建 Windows XP 环境。

Nboot 是一个十分简单的 bootloader，其大小不到 4K，一般被烧写到 Nand Flash 的 Block 0 位置用来启动 WinCE 内核，Nboot 原由三星提供，我们对此做了很多改进，目前有如下特色功能：

- 自适应支持 64M/128M/256M/1G mini2440/micro2440
- 支持开机画面快速显示
- 支持加载 WinCE 内核的动态进度条
- 启动 WinCE 仅需 5-10 秒，视内核大小而定

需要注意的是，Nboot 并不具备烧写功能，它只能读取已经烧写处理好的文件：开机画面(BootLogo)和 WinCE 内核。

Nboot 具有很方便的定制性，你可以通过头文件定义修改开机画面的显示位置、背景，以及进度条的颜色、位置、长宽等，这些定义位于 **option.h** 文件中，如下：

```
//通过更改定义，选择相应的 LCD 型号，此处默认选择 W35，表示横屏 W35(LQ035Q1DG06)
//#define LCD_N35
//#define LCD_L80
//#define LCD_T35
//#define LCD_X35
#define LCD_W35
//#define LCD_A70
//#define LCD_VGA1024768

//设置背景色
#define BACKGROUND_R    0x00
#define BACKGROUND_G    0x00
#define BACKGROUND_B    0x00

//设置进度条的颜色
#define PROGRESS_BAR_R 0xFF
#define PROGRESS_BAR_G 0xFF
#define PROGRESS_BAR_B 0x00

//设置开机图片的位置
#define LOGO_POS_TOP     0
#define LOGO_POS_LEFT    0

//设置启动条的位置和长宽
#define PROGRESS_BAR_TOP       260
#define PROGRESS_BAR_LEFT       20
#define PROGRESS_BAR_WIDTH     200
#define PROGRESS_BAR_HEIGHT     12
```

下面介绍 Nboot 的编译方法和步骤：

编译 Nboot

把光盘中“WindowsCE6.0”目录中的文件夹“NBOOT”文件夹复制到硬盘的某一个目录(在此为 D:\work)，去掉其只读属性，运行 ADS1.2 集成开发环境，点 **File->Open…**打开 nboot.mcp 文件，如图。

这时点菜单 Project→Make 或者直接按 F7 键，开始编译 nboot 项目，编译完毕如图：

在 D:\work\NBOOT\nboot_Data\DebugRel 目录下会生成 nboot.bin 可执行文件，如图。

把 NBOOT 烧写到开发板的 Nand Flash

(1)连接好开发板电源，串口线，USB 线，并**设置拨动开关 S2 为 Nor Flash 启动系统**，分别打开串口超级终端和 DNW，上电启动开发板。

(2)保证 USB 驱动已经安装好(前面已经详细介绍了 USB 驱动的安装方法)，这时可以看到 DNW 的标题栏显示[USB：OK]，如果没有安装好驱动会显示[USB：x]，如图所示：

(3)这时在超级终端的 BIOS 功能菜单中选择功能号[n]，出现 USB 下载等待提示信息：

(4)点击 DNW 程序的“USB Port”→“Transmit”，如图选择刚刚编译出的映象文件(光

盘“images\wince6.0”目录中有已经编译好的可执行文件)，这样就开始下载了，瞬间就可下载完毕，supervivi 会把它自动烧写到 Nand Flash 起始 block 0 中；把 S2 开关拨至“NAND”一侧，选择从 Nand Flash 启动系统；如果系统中已经烧写好了开机图片文件和 WINCE 内核映象文件，马上就会开机画面和进度条，稍等片刻就看到 wince 启动了。

使用 NBOOT 启动 wince 的串口信息如下：

9.2.5 在 BSP 中修改 LCD 类型及串口输出功能

说明：我们提供的 BSP 目前支持以下型号的液晶屏：

- 横屏 3.5 寸 LCD 带触摸(LQ035Q1DG06 或兼容，简称 W35)
- NEC3.5 寸屏带触摸
- 统宝 3.5”LCD 带触摸
- Sharp 8”LCD(或兼容)带触摸
- 7 寸屏带触摸
- VGA 模块显示输出，分辨率 1024x768

通过修改 mini2440\Src\Inc\options.h 头文件中 LCD_TYPE 的定义，可以选择相应的 LCD 类型：

//#define LCD_N35 适用于 NEC3.5”LCD
//#define LCD_L80 适用于 Sharp 8”LCD(或兼容)
//#define LCD_T35 适用于统宝 3.5”LCD
//#define LCD_X35 适用于 Sony 3.5”LCD(LQ035Q1DG06 或兼容，简称 W35)
#define LCD_W35 适用于横屏 3.5”LCD
//#define LCD_A70 适用于群创 7”LCD
//#define LCD_VGA1024768 适用于 LCD2VGA 模块，分辨率为 1024x768

提示：光盘中缺省 LCD 型号是 LCD_T35。

在 options.h 文件中，用户也可以修改串口的输出功能：作为普通串口功能或者调试输出(仅限于串口 1 和 2)，如下定义：

#define KITL_NONE
//#define KITL_SERIAL_UART0
//#define KITL_SERIAL_UART1
//#define KITL_USBSERIAL

//#define KITL_ETHERNET

这里缺省的定义是作为普通串口功能，如果要把串口 1 作为调试信息输出使用，则应该定义为：

```
//#define KITL_NONE
#define KITL_SERIAL_UART0
//#define KITL_SERIAL_UART1
//#define KITL_USBSERIAL
//#define KITL_ETHERNET
```

9.2.6 制作和修改 Windows CE 启动 Logo

在前面的章节，我们提到过：

WindowsCE 系统的启动过程有两种 Logo：BootLogo 和 StartLogo。其中 BootLogo 是由 Nboot 加载显示的，用户可以通过修改 Nboot 的源代码调整 BootLogo 的显示位置和背景色；StartLogo 则属于 BSP 的一部分，它是一个数组文件(StartLogo.c)，位于“mini2440\Src\Kernel\Oal”目录，由该目录下的 init.c 文件实现加载显示，StartLogo.c 文件可以通过本光盘中的 StartLogoMaker.exe 工具制作生成。

StartLogoMaker 由友善之臂开发的 Linux Logo 制作工具 LogoMaker(运行于 Fedora9)移植而来，是一个“绿色软件”，它不需要安装，直接复制到 WindowsXP/Vista 平台即可运行，使用它可以把 bmp,jpg,png 等格式的图片转换为 mini2440 BSP 所需要的数组文件 StartLogo.c，使用新生成的文件替换 BSP 中的同名文件，即可更换 WindowsCE 的启动画面，StartLogo.c 数组的头部内容如下：

```
// Automatic generated by StartLogo.exe from FriendlyARM Co., Ltd.

static const unsigned short StartLogoData[] = {
240, 320,
0x965, 0x945, 0x164, 0x9C4, 0x1246, 0x22CA, 0x22A8, 0x2AA7,
```

下面是使用 StartLogoMaker.exe 制作 StartLogo.c 的步骤：

Step1：双击运行“windows 平台工具\StartLogoMaker”中的 StartLogoMaker.exe 程序，打开如图界面：

Step2：点 File->Open 打开一个图片文件，也可以在工具栏点 图标打开文件选择窗口：

Step3：点 File->Convert，或者点工具栏的图标 打开文件输出选择窗口：

点“确定”在相应的目录会生成 StartLogo.c 文件：

Step5：把生成的文件替换 BSP 中的同名文件(位于 mini2440-BSP\ Src\Kernel\Oal 目录中)，重新编译内核，并烧写到板子中运行，即可看到你自己制作的 WinCE 启动画面了：

9.2.7 创建 SDK

SDK 适用于：当开发主机只安装了 VS2005，但没有安装 Windows CE 6.0 的 Platform Builder 插件时，这时开发人员想通过 VS2005 开发 mini2440 的应用程序，就需要一个 SDK，它类似于 Embedded Visual C++所需的 SDK。

当你编译完缺省内核，此时可以通过 VS2005 平台创建相应的 SDK，注意：这里的 SDK 仅适用于 VS2005 开发环境，它不能安装到 EVC，也不能安装到 VS2008，下面是创建 SDK 的详细步骤。

Step1：运行 VS2005 并打开已经编译过的缺省内核示例工程 mini2440,找到如图位置，并右键点击“Mini2440-CE6-SDK”出现菜单，点 Build 开始创建 SDK

Step2：稍等片刻，SDK 创建完毕，如图所示

Step3：在 C:\WINCE600\OSDesigns\Mini2440\Mini2440\SDKs\SDK1 目录下，可以看到已经生成 Mini2440-CE6-SDK.msi 安装文件

9.2.8 安装 SDK

要通过 VS2005 为 mini2440 开发应用程序，需要先安装刚才生产的 SDK，步骤如下

Step1：双击运行 Mini2440-CE6-SDK.msi，出现如下界面，点“Next”继续

Step2：如图选择“I accept”，点“Next”继续

Step3：出现如图界面，输入用户名和公司名，点“Next”继续

Step4：出现如图界面，点“Complete”继续

Step5：出现如图界面，点“Next”继续

Step6：出现如图界面，点“Install”继续

Step7：出现如图安装进度界面，稍等片刻

Step8：出现安装结束界面，点“Finish”结束

至此，SDK 已经安装完毕。

9.3 与 PC 同步(基于 Windows 7)

注意：在 Windows 7 系统中，无需安装 ActiveSync 软件！

在 Windows 7 系统中，开发板与 PC 的同步是通过“Windows Mobile 设备中心”(下称“同步中心”)来实现并管理的，它类似于以前的 ActiveSync，它的界面如图所示。

“同步中心”并非在 Windows 7 中自带，而是首次连接移动设备时通过互联网下载安装的，下面是详细的步骤。

说明：如果开发板安装了 WinCE6，用户依然可以通过 Windows XP 系统的 ActiveSync 与之相连，具体步骤可以参考老版本的用户手册，在此介绍的步骤仅适用于 Windows 7 系统。

9.3.1 安装 Windows Mobile 设备中心实现 PC 同步

当开发板中安装并运行 WinCE6 系统后，第一次和基于 Windows 7 系统的 PC 通过 USB 连接时，会弹出如图窗口

很快，就会在桌面上出现如下提示窗口

此时要保证你的网络是和互联网连通的，系统会自动下载并安装配置相关的软件，如图

安装完毕，出现如下界面开始自动配置

出现“软件许可协议”窗口，点“接受”继续

之后，很快就和开发板设备连接成功了，如图

点“不设置就进行连接”按钮，继续，出现如图界面

此时，点“文件管理”之“浏览设备上的内容”就会像打开目录一样打开开发板的根目录，如果开发板上插了优盘或者 SD 卡，也会像优盘一样出现相应的图标。

在此，我们打开“\”文件夹，它表示了整个开发板的目录内容，如下图，这时，你就可以通过拖放向开发板中复制文件了，当然也可以从开发板中读取文件。

9.4 通过 VS2005 创建应用程序，并编译下载到开发板运行

下面是使用 VS2005 的基本开发步骤：

9.4.1 创建项目

(1)打开运行 VS2008，点菜单文件→新建→项目，如图：

(2)出现“新建项目”窗口，在“项目类型”一栏中点击选择“Visual Basic → 智能设备”，在“模板”一栏中选择“智能设备项目”，在“名称”一栏中输入“my2440”，其他采用缺省设置，点“确定”继续：

(3)出现“添加新只能设备项目”窗口，在“目标平台”下拉列表中选择“Windows CE”，在“.NET Compact Framework 版本”下拉列表中选择“.NET Compact Framework Version 2.0”，在“模板”中选择“设备应用程序”，点“确定”继续：

(4)出现如图工作界面

这时，可以调整一下“Form”的大小，通过“工具箱”添加一些控件到“Form”中，如图：

(5)双击控件“Button1”出现代码设计窗口，输入以下简单代码，它将实现：当点击 Button1 按钮时，窗口标题会显示代码中设置的内容，如图：

保存好以上创建的项目文件。

9.4.2 设置连接开发板

(1) 确认 PC 同步已经建立并连接正常(可以不必连接网线)，如图

(2)点 VS2008 菜单“工具”→ “选项”，出现“选项”窗口，
在左侧一栏中选择“设备工具”→ “设备”，
在右侧中的各个下拉列表中作如图选择：

点“属性”按钮，出现“Windows CE 设备 属性窗口”，选择如图：

点“配置”按钮，出现“配置 TCP/IP 传输”窗口，如图选择，点“确定”返回 VS2008 工作界面。

(3)点 VS2008 菜单“工具”→“连接到设备”，出现“连接到设备”设置窗口，如图进行选择设置：

点“连接”按钮，此时 VS2008 开始和开发板进行连接握手：

稍等一会，出现连接成功的提示，点“关闭”按钮返回 VS2008 工作主界面：

9.4.3 编译下载程序到开发板运行

(1)接上面的步骤，点菜单“调试”→ “启动调试”或者直接按 F5 键开始调试过程。

(2)出现“部署 my2440”窗口，选择“Windows CE 设备”，并点“部署”按钮开始部署，如图

(3)如果程序没有问题，等待一会它会直接下载到开发板并运行，点击一下“Button1”按钮，窗口标题改变如图，这正如我们在代码中的设置， 如图：

通过 VS2008，你不仅可以编写 Visual Basic 程序，还可以使用 Visual C#和 Visual C++进行程序设计，通过同步连接开发板，你还可以实现单步调试运行。

第十章 WindowsCE 5.0 开发指南

10.1 基于 WindowsCE5.0 的开发环境

本小节介绍如何建立 WindowsCE 5.0 的开发环境，这包括安装 Platform Builder 5.0 及其补丁，导入开发板所用的 BSP 和编译示例工程。

说明：Platform Builder 5 不能在 Windows Vista/2007 平台上安装使用，以下操作环境均基于 Windows XP.

10.1.1 安装 Platform Builder 5.0(含 2007 最新补丁)

注意: 本公司并不提供 Windows Embedded 6.0 CE 6 的安装文件，用户可以到微软网站自行下载它的试用版

- ✓ 试用版下载地址：

http://www.microsoft.com/downloads/details.aspx?FamilyID=486E8250-D311-4F67-9FB3-23E8B8944F3E&displaylang=en

- ✓ Platform Builder 5.0 的 2008 补丁下载地址：

http://www.microsoft.com/downloads/details.aspx?displaylang=en&FamilyID=6dccd9fc-a7ac-4fa3-a9bd-fcc7a83f5311

说明：以下安装步骤使用的补丁文件是 2007 年的，但此补丁已经在微软网站无法找到下载链接，微软建议直接使用 2008 年的补丁，我们尚未验证 2008 的补丁是否可用，Platform Builder 5.0 之 2007 年的补丁安装文件位于光盘“\WindowsCE5.0\PB5 补丁 2007”目录。

安装提示：安装 PB5 时请把安装文件复制到硬盘安装，否则有可能会不能顺利执行以下步骤。

本节以图例介绍如何在 WindowsXP 上安装 Platform Builder 5.0(简称 PB5)，它用来开发和定制 WINCE 内核，并可以用来调试内核等，安装 PB 一般需要 5-7G 的硬盘空间。

Step1：安装PB5需要dotnet framework1.1，如果系统中没有安装此组件，将会出现如图提示：

在PB5的安装光盘中可以找到此组件的安装文件，双击运行按照提示安装即可，如图：

Step2：在PB5安装光盘中找到Microsoft Windows CE 5.0.msi.exe，双击运行开始安装PB5：

Step3：出现安装向导窗口，点“Next”继续：

Step4：出现“License Agreement”窗口，选择“I accept the terms in the license agreement”，并点“Next”继续：

Step5：输入用户信息和序列号，点“Next”继续：

Step6：选择安装类型为定制安装，如图，点“Next”继续

Step7：选择安装目录，这里按默认，点“Next”继续

Step8：在定制安装中选择您所需要的系统平台，在这里一定要选择安装ARMV4I，最好也选择安装“Shared Source for Windows CE 5.0”，如图.点"Next"继续

Step9：出现许可协议窗口，选择“I accept the terms in the license agreement”，并点“Next”继续：

Step10：出现如图提示窗口，点“Next”继续

Step11：进入安装进程界面，此过程时间较长，请耐心等待，如图

Step12：安装完毕，如图

Step13：现在开始安装PB5补丁程序，其安装文件位于光盘WindowsCE 5.0\PB5补丁2007文件夹中，双击该程序开始安装：

Step14：出现安装向导界面，如图，点“Next”继续

Step15：出现许可协议窗口，选择“I accept the terms in the license agreement”，并点“Next”继续：

Step16：出现如图公共代码许可协议窗口，选择“I accpet”，点“Next”继续

Step17：出现定制安装界面，选择默认安装全部组件，点“Next”继续

Step18：准备好安装，点“Next”继续

Step19：安装过程如图所示，该过程比较长，请耐心等待

Step20：安装过程会出现如下提示界面，不必理会，点“确定”即可

Step21：安装完毕，如图

10.1.2 导入安装 BSP

WindowsCE5 的 BSP 位于光盘目录：WindowsCE5.0\mini2440-ce5-bsp-20100816

其中目录最后面的数字为发行或者更新日期，请以实际为准，为了描述方法，我们以下把该目录简称为 mini2440

说明： BSP 目前支持以下型号的液晶屏：

- 横屏 W35(LQ035Q1DG06 或兼容)，简称为 W35
- NEC3.5 寸屏带触摸，简称为 N35
- 统宝 3.5”LCD 带触摸，简称为 T35
- Sharp 8”LCD(或兼容)带触摸，简称为 L80
- 7 寸屏带触摸，简称为 A70
- VGA 模块显示输出，分辨率 1024x768，简称为 VGA1024x768

通过修改 mini2440\Src\Inc\options.h 头文件中 LCD_TYPE 的定义，可以选择相应的 LCD 类型：

//#define LCD_N35 适用于 NEC3.5”LCD
//#define LCD_L80 适用于 Sharp 8”LCD(或兼容)
//#define LCD_T35 适用于统宝 3.5”LCD
//#define LCD_X35 适用于 Sony 3.5”LCD
#define LCD_W35 适用于横屏 W35(LQ035Q1DG06 或兼容，简称 W35)
//#define LCD_A70 适用于群创 7”LCD
//#define LCD_VGA1024768 适用于 LCD2VGA 模块，分辨率为 1024x768

光盘中缺省 LCD 型号是 LCD_T35。

要使用 PB5 编译 WINCE 内核映象，需要安装对应目标板的 BSP，并进行一些设置，请按照以下步骤安装 BSP：

Step1：把光盘WindowsCE 5.0目录里面的mini2440文件夹复制“C:\WINC**E500\PLA**TFORM ”目录下，并去掉只读属性。

Step6：打开“Platform Builder 5.0”，选择“File”菜单下的“Manage CatalogFeatures”

点“Import按钮”，导入“platform\mini2440\mini2440.cec”文件

Step7：在“Catalog”的Third Party下，将会自动添加“Mini2440 From FriendlyARM:ARMV4I”项，BSP安装完成。

10.1.3 安装无线网卡驱动程序

说明：无线网卡驱动并不是必须要安装的，如果不安装，使用 mini2440.pbxml 工程示例编译出的 WinCE 内核将不具备无线网卡驱动。

另外，此无线网卡驱动仅支持 VNT-6656G6A40 型号的 USB 无线网卡

无线网卡驱动程序位于光盘WindowsCE驱动程序模块\无线网卡\文件夹中，它是一个安装文件“**VNUWLC5-ARM.msi**”。下面是安装步骤：

Step1：双击运行安装程序，打开安装向导，点“Next”继续

Step2：开始安装，并提示将安装到C:\WINCE500目录中，如图

Step3：安装进程如图，安装很快就会结束

Step4: 安装结束，点“Finish”结束安装

Step5: 这时打开PB5，会看到Catalog一栏出现如图选项，如图

10.1.4 编译内核工程示例

Step1: 在 C:\WINCE420\PBWorkspaces 目录(如果没有，可以手工创建一个)中创建一个文件夹“mini2440”，把光盘中 WindowsCE 5.0 目录下的 mini2440.pbxml 文件 C:\WINCE420\PBWorkspaces\mini2440 目录，并去掉只读属性。

Step2: 在 PB 中点 File → Open Workspace…，打开刚刚复制的项目文件，注意是 pbxml 结尾的。

说明：也可以通过双击打开 mini2440.pbxml 文件。

如图为打开的项目文件之 PB 界面：

Step3: 打开后，点 Build OS → Sysgen 开始编译，或者点工具栏的 图标开始进行编译，该过程比较长。

Step4: 编译完毕，就会生成“nk.bin”和“nk.nb0”两个文件，其中 nk.bin 是发行版本，nk.nb0 是内存中运行版本，我们一般使用 nk.bin。它们位于 **C:\WINCE500\PBWorkspaces\mini2440\RelDir\mini2440_ARMV4I_Release**

编译过程中有可能会出现如图这样的警告信息，这个是正常的，不必理会。

10.1.5 导出 SDK

我们可以把定制好的内核工程导出为 SDK 安装文件，它用来提供给应用开发人员，里面主要包含与定制平台有关的头文件、库、一些文档等内容。应用开发人员可以通过安装 SDK 在 Embedded Visual C++(以下简称 EVC)中开发基于此平台的应用程序。

说明：光盘“\WindowsCE5.0\SDK”目录中有已经制作好的 SDK 安装文件，您可以直接使用而不必自己制作。

下面是具体的导出步骤。

Step1: 首先打开并确定已经编译好工程示例，点 Platform → SDK → New SDK…如图：

Step2: 跳出“Export SDK Wizard”向导窗口，点“下一步”继续：

Step3: 进入“Prodect Properties”配置窗口，可以根据实际情况填写配置，点“下一步”继续，如图：

Step4: 进入“Development Language”配置窗口，选择开发语言支持，点“下一步“继续：

Step5: 配置完毕，点“Fnish“按钮结束。

Step6: 点 Platform → SDK → Configure SDK…，出现设置窗口，在这里你可以对刚才的初始配置进行更加详细的设置，点“CPU“选项卡，出现如图界面：

Step7: 点“Edit”按钮，出现如图界面，并如图选择：

Step8: 点“OK”返回 PB5 主界面，再点 Platform → SDK → Build SDK…：

Step9: 出现编译向导窗口，并同时开始编译制作 SDK：

Step10: 大概几分钟时间，编译完毕，此时点“Done“按钮结束：

Step11: 根据提示，最后在如图目录生成 SDK 安装文件：

10.1.6 安装 Embedded Visual C++(EVC)

为了开发基于 API 的 WinCE 应用程序，需要安装 EVC 集成开发环境和相应的 SDK 及补丁，下面是详细的 EVC 安装步骤：

(1)EVC 安装文件位于光盘 Embedded VisualC++\目录中，双击 setup.exe 开始安装

(2)出现安装向导界面，点“Next”继续

(3)出现用户许可协议，选择“I accept the agreement”点“Next”继续

(4)输入序列号和用户信息，点“Next”继续

(5)选择安装组件，选择默认全部安装即可，点“Next”继续

(6)选择安装目录，这里按默认，点“Next”继续

(7)跳出提示窗口，选择“是”继续

(8)开始安装 EVC 管理器，出现如图过程窗口，等待即可

(9)EVC 平台管理器安装完毕，点“OK”结束

(10)出现 EVC 安装界面，点“Continue”继续

(11)出现产品 ID 号提示信息窗口，点“OK”继续

(12)选择安装组件和安装路径，按默认即可，点“Next”继续

(13)开始安装进程，如图

(14)安装完毕，点“OK”结束

注意：此时如果点开始→程序→Microsoft eMbedded Visual C++ 4.0 → eMbedded Visual C++ 4.0 快捷菜单，尚不能正常运行，会出现以下提示窗口，因此你还需要按照下面的章节继续安装 SDK 才可以。

10.1.7 安装 EVC 补丁和导出的 SDK

为了能够正常使用我们导出的 SDK 安装文件，**必须要先安装 EVC 的 SP4 补丁文件，它位于\Embedded VisualC++\SP\evc4sp4\DISK1 目录中**，下面的步骤我们先安装补丁文件，再安装刚才我们导出的 SDK，如图：

(1)双击运行 SP4 的安装程序 setup，出现安装向导界面，点“Next”继续

(2)出现用户许可协议窗口，如图选择，点“Next”继续

(3)准备安装 SP4，点“Next”继续

(4)开始安装进程，如图

(5)安装结束，如图

(6)现在我们开始安装刚才导出的 SDK，你可以使用自己导出的，也可以在光盘中找到我们已经做好的(它位于光盘的 WindowsCE5.0\SDK 文件夹中，名字为“mini2440_SDK.msi.exe”)，双击运行它，出现安装向导窗口，点“Next”继续：

(7)这时会出现如下提示窗口，不必理会，点“Close”关闭掉，继续

(8)出现用户许可协议窗口，选择“Accpet”，点“Next”继续：

(9)出现“Customer Information”窗口，输入相应信息，点“Next”继续：

(10)出现安装类型窗口，点“Complete”完全安装继续：

(11)出现安装目录选择窗口，按默认即可，点“Next”继续：

(12) 在Ready to Install对话框中,点击“Install”安装，如图：

(13)系统开始安装，如图：

(14)安装完毕，点“Finish”结束：

现在，你就可以点开始→程序→Microsoft eMbedded Visual C++ 4.0 → eMbedded Visual C++ 4.0 快捷菜单，打开 EVC 集成开发环境了，如图：

10.1.8 定制 CE 内核

通过以上步骤，我们了解了如何安装 BSP，编译内核工程，导出 SDK 等常用操作，现在我们使用图解的方式介绍一下如何定制适合于自己的 WinCE 内核工程。

注意：以下步骤主要是为说明定制 WinCE 内核的相关步骤和组件，根据以下步骤建立的项目不一定能顺利编译通过，请用户自行酌情增减。

(1)打开 PB5，点 File → New Platform…，跳出内核定制向导，点“下一步”继续：

(2)出现工程命名设置窗口，输入“my2440”，点“Next”继续：

(3)选择所要使用的 BSP，在这里当然选择 2440 的，如图，点“Next”继续：

(4)出现模板选择窗口，我们选择手持设备“Mobile Handheld”，如图，点“Next”继续：

(5)出现应用程序定制窗口，这里出现的都是常见的一些应用程序，在这里**必须选择红色圈号里面的“Standard SDK for WindowsCE”**，如图，点“Next”继续：

(6)出现网络有关的一些设置，使用缺省即可，点“Next”继续：

(7)出现和蓝牙、红外有关的一个窗口，这里没有任何选项，因此点“Next”继续：

(8)出现向导结束窗口，点“Finish”结束向导：

(9)现在回到 PB5 的主窗口，可以看新的工程文件已经创建，我们还需要为编译做一下准备：

(10)点 Platform → Setting…，打开工程设置窗口：

(11)在设置窗口中，点“Locale”选项卡来设置目标内核的语言，这里选择中文：

(12)点“Build Options”选项卡，去掉“Enable CE Target Control Support”和“Enable KITL”这两个设置，其他使用缺省，并点“OK”完成设置，如图：

(13)加入 USB 鼠标和键盘的支持，在 Catalog 一栏依次点击展开 Core OS → Windows CE device → Core OS Services → USB Host Support → USB Human Input Device(HID) Class Driver，点右键选择“Add to OS Design”，并展开其子项添加“USB HID Keyboard and Mouse”，如图：

(14)使用同样的方法可以添加 USB 存储设备的支持，如图：

(15)添加文件系统的支持，点 Core OS → File Systems and Data Store → File System – Internal (Choose I) → RAM and ROM File System，为了支持优盘所使用的 FAT32 文件系统，你还需要添加途中矩形方框中的 FAT File System，如图：

(16)添加注册表保存卡支持，点 Core OS → File Systems and Data Store → Registry Storage (Choose I) → Hive-based Registry，如图：

(17)修改默认的 IP 地址，打开 platform.reg 文件，找到如图红色框所示的位置，在这里你可以修改开机后默认的 IP 地址，网关设置，DNS 等设置：

(18) 修改桌面背景图片，准备一个 bmp 格式的图片，复制到 C:\WINCE500\Platform\SMDK2440\Files 目录中，并命名为 bliss.bmp，如图：

打开 platform.bib 文件，在 FILES 栏目中把 bliss.bmp 添加进去，如图：

(19)保存以上设置，点 File → Save 即可，再点 Build OS → Sysgen 或者点工具栏的 按钮就开始编译内核了，编译完毕如图所示：

关于更多的内核配置和更改用户可以自己摸索尝试，或者参考相关网上资料等，在此不再叙述。

10.1.9 制作 WindowsCE 开机画面 StartLogo

在前面的章节，我们提到过：

WindowsCE 系统的启动过程有两种 Logo：BootLogo 和 StartLogo。其中 BootLogo 是由 Nboot 加载显示的，用户可以通过修改 Nboot 的源代码调整 BootLogo 的显示位置和背景色；StartLogo 则属于 BSP 的一部分，它是一个数组文件(StartLogo.c)，位于“mini2440\Src\Kernel\Oal”目录，由该目录下的 init.c 文件实现加载显示，StartLogo.c 文件可以通过本光盘中的 StartLogoMaker.exe 工具制作生成。

StartLogoMaker 由友善之臂开发的 Linux Logo 制作工具 LogoMaker(运行于 Fedora9)移植而来，是一个“绿色软件”，它不需要安装，直接复制到 WindowsXP/Vista 平台即可运行，使用它可以把 bmp,jpg,png 等格式的图片转换为 mini2440 BSP 所需要的数组文件 StartLogo.c，使用新生成的文件替换 BSP 中的同名文件，即可更换 WindowsCE 的启动画面，StartLogo.c 数组的头部内容如下：

```
// Automatic generated by StartLogo.exe from FriendlyARM Co., Ltd.

static const unsigned short StartLogoData[] = {
```

240, 320,
0x965, 0x945, 0x164, 0x9C4, 0x1246, 0x22CA, 0x22A8, 0x2AA7,

下面是使用 StartLogoMaker.exe 制作 StartLogo.c 的步骤：

Step1：双击运行“windows 平台工具\StartLogoMaker”中的 StartLogoMaker.exe 程序，打开如图界面：

Step2：点 File->Open 打开一个图片文件，也可以在工具栏点 图标打开文件选择窗口：

Step3：点 File->Convert，或者点工具栏的图标 打开文件输出选择窗口：

点“确定”在相应的目录会生成 StartLogo.c 文件：

Step5：把生成的文件替换 BSP 中的同名文件(位于 mini2440-BSP\ Src\Kernel\Oal 目录中)，重新编译内核，并烧写到板子中运行，即可看到你自己制作的 WinCE 启动画面了：

10.1.10 BootLoader 之 Nboot 的编译和烧写

说明：我们提供的 Nboot 可同时适用于 WindowsCE5/6，因此在光盘中的所有 Nboot 项目源代码都是相同的，只不过为了方便组织使用，特意增加了拷贝，并分别放在了不同的目录。

编译 Nboot 需要使用 ADS 集成开发环境，详细的步骤见本手册第四章，在 Windows 7 系统上，你可以通过安装 Windows XP Mode 来创建 Windows XP 环境。

Nboot 是一个十分简单的 bootloader，其大小不到 4K，一般被烧写到 Nand Flash 的 Block 0 位置用来启动 WinCE 内核，Nboot 原由三星提供，我们对此做了很多改进，目前有如下特色功能：

- 自适应支持 64M/128M mini2440/micro2440
- 支持开机画面快速显示
- 支持加载 WinCE 内核的动态进度条
- 启动 WinCE 仅需 5-10 秒，视内核大小而定

需要注意的是，Nboot 并不具备烧写功能，它只能读取已经烧写处理好的文件：开机画面(BootLogo)和 WinCE 内核。

Nboot 具有很方便的定制性，你可以通过头文件定义修改开机画面的显示位置、背景，以及进度条的颜色、位置、长宽等，这些定义位于 **option.h** 文件中，如下：

```
//通过更改定义，选择相应的 LCD 型号，此处默认选择 T35：
//#define LCD_N35
//#define LCD_L80
#define LCD_T35
//#define LCD_X35
//#define LCD_W35
//#define LCD_A70
//#define LCD_VGA1024768

//设置背景色
#define BACKGROUND_R    0x00
#define BACKGROUND_G    0x00
#define BACKGROUND_B    0x00

//设置进度条的颜色
#define PROGRESS_BAR_R 0xFF
#define PROGRESS_BAR_G 0xFF
#define PROGRESS_BAR_B 0x00

//设置开机图片的位置
```

```
#define LOGO_POS_TOP      0
#define LOGO_POS_LEFT     0

//设置启动条的位置和长宽
#define PROGRESS_BAR_TOP       260
#define PROGRESS_BAR_LEFT       20
#define PROGRESS_BAR_WIDTH     200
#define PROGRESS_BAR_HEIGHT     12
```

下面介绍 Nboot 的编译方法和步骤：

编译 Nboot

把光盘中“WindowsCE5.0”目录中的文件夹“NBOOT”文件夹复制到硬盘的某一个目录(在此为 D:\work)，去掉其只读属性，运行 ADS1.2 集成开发环境，点 **File->Open…**打开 nboot.mcp 文件，如图。

这时点菜单 Project→Make 或者直接按 F7 键，开始编译 nboot 项目，编译完毕如图：

在 D:\work\NBOOT\nboot_Data\DebugRel 目录下会生成 nboot.bin 可执行文件，如图。

10.1.11 把 NBOOT 烧写到 Nand Flash

(1)连接好开发板电源，串口线，USB 线，并**设置拨动开关 S2 为 Nor Flash 启动系统**，分别打开串口超级终端和 DNW，上电启动开发板。

(2)保证 USB 驱动已经安装好(前面已经详细介绍了 USB 驱动的安装方法)，这时可以看到 DNW 的标题栏显示[USB：OK]，如果没有安装好驱动会显示[USB：x]，如图所示：

(3)这时在超级终端的 BIOS 功能菜单中选择功能号[n]，出现 USB 下载等待提示信息：

(4)点击 DNW 程序的“USB Port”→“Transmit”，如图选择刚刚编译出的映象文件(光

盘“images\wince5.0”目录中有已经编译好的可执行文件)，这样就开始下载了，瞬间就可下载完毕，supervivi 会把它自动烧写到 Nand Flash 起始 block 0 中；把 S2 开关拨至“NAND”一侧，选择从 Nand Flash 启动系统；如果系统中已经烧写好了开机图片文件和 WINCE 内核映象文件，马上就会开机画面和进度条，稍等片刻就看到 wince 启动了。

使用 NBOOT 启动 wince 的串口信息如下：

10.2 使用 ActiveSync 与 PC 同步

使用微软提供的工具 ActiveSync，可以让开发板与 PC 之间十分方便的进行通讯连接，从而实现文件上传，远程调试等功能。

10.2.1 安装 ActiveSync

在光盘的“windows 平台工具”目录中 ActiveSync 文件夹，双击运行里面的 ActiveSync_4.1_setup.exe 开始安装。

如图选择“我接受该许可协议中的条款”，点“下一步”继续

输入用户名和单位名称，点“下一步”继续

选择要安装的目的路径，这里使用缺省值，点“下一步”继续。

出现如下界面，点“安装”开始进行安装。

出现安装过程界面，如下

安装完毕，点“完成”安装完毕。

这时会自动运行 ActiveSync，点“取消“，同时在任务栏出现相应的图标托盘，出现如下界面

10.2.2 为同步通讯安装 USB 驱动

确认板子里面已经烧写好了我们提供的 WINCE 映象文件，并开机运行，系统起来以后，接上 USB 电缆，并与 PC 连接，如果以前没有安装过这个驱动，计算机会出现“发现新硬件”的提示，这时就可以按照本小节的步骤安装驱动了，我们用到的驱动程序的位置在光盘的“\Windows 平台工具\CE 用同步 USB 驱动”目录中：

Step1：找到新的硬件，请如图进行选择，点“下一步”继续。

Step2：选择“从列表或指定位置安装”，点“下一步”继续，如图。

Step3：点“浏览”并定位到
光盘\Windows 平台工具\CE 用同步 USB 驱动，如图

Step4：系统将会自动搜索一会，然后出现如下界面，点“仍然继续”，系统将自动安装完毕 USB 驱动。

Step5：同时 ActiveSync 会自动跳出运行，如果您对使用 ActiveSync 还不熟悉，请点击“取消”。

10.2.3 使用 ActiveSync 同步传输工具复制文件

经过上一小节的 USB 驱动安装，实际上您已经看到开发板已经同 PC 连接好了，我们现在看看如何使用它。

(1)查看 PC 端的连接情况

当我们看到如下窗口跳出时，我们也可以注意到 PC 任务栏的右下角的 ActiveSync 也变成了绿色，这说明一切准备就绪。

实际上，ActiveSync 安装完毕后，在“我的电脑”里会出现一个“移动设备”图标，现在我们双击打开它，您将看到目标板的所有目录，如下

这时，双击打开 ResidentFlash 目录，在您的电脑里面找一首 mp3 文件或者其他小于 30M 的文件并拖动到打开的 ResidentFlash 文件夹，如图

这样，我们就能像操作 Windows 里面的目录一样，对开发板里面的目录进行文件复制等操作了。

(2) 查看开发板一端的情况

接上一步，我们复制了一首 Mp3 文件到 ResidentFlash 目录，是不是真的有了呢？

接上 USB 鼠标或者使用触摸笔，点击打开 我的电脑->ResidentFlash 目录，可以看到如下界面。

10.2.4 使用 ActiveSync 与 Platform Builder 连接实现通讯并屏幕截图

首先确认开发板已经和 PC 之间可以使用 ActiveSync 连接成功，同时网络连接也没有问题(至关重要!)，如下图

我们先设置开发板上的 WINCE 连接。

打开 **开始->设置->网络和拨号连接**，点“**新建连接**”设置对话框中，选择连接类型为“直接连接”，如下图

改连接名称为“我的 usb 连接”或者不改也可以，点“下一步”，在出现的“选择设备”下拉列表中选择“S3C2440 USB Cable:”，如下图

点右上角的“OK”，这时出现“我的 usb 连接”图标，如下图

打开“开始->设置->控制面板”，双击打开“PC 连接”图标，进入“PC 连接属性”设置对话框，选中“启用与桌面计算机的直接连接”的复选框，然后再点“更改连接”按钮，如下图

在“更改连接”设置对话框的下拉列表中选择刚刚新建的连接“我的 usb 连接”，然后点“OK”退出设置，这样就完成了开发板这端的设置。

现在，我们再来设置 PC 端，请务必确认 ActiveSync 已经和开发板连接成功。

点 PB 菜单 Tools -> Remote zoom-in，打开远程图片缩放工具，这个程序可以实现对远程移动设备显示屏幕的截屏。

Remote zoom-in 窗口打开之后，先要配置一下平台管理器，点击菜单 **Connection->Configure windows ce platform manager**，如下图

在弹出的对话框中添加一个新设备，并且取名为“mini2440”，如下图

点右边的“Propertions…”按钮，设置“mini2440”设备平台的属性，如下图

点“Transport”下拉框右边的“Configure”按钮，开始设置 TCP/IP 传输，在 HOST IP 一项中输入您的主机 IP 地址，请保证开发板 WINCE 的 IP 地址(默认为 192.168.1.230)与 PC 主机的 IP 地址在同一个网段，如图

设置结束，点 OK 按钮返回到 Remote Zoom-in 主窗口，点菜单 Connection->Connect to Device，开始连接开发板。

这时一般会出现如下图所示界面，显示正在连接。

当连接成功，会出现和目标板当前屏幕一样的几个界面窗口，如下图

这时点 File-Save as 可以保存当然的屏幕截图。

10.2.5 使用 ActiveSync 与 Platform Builder 在线编辑注册表

当您学会使用上一小节的内容实现截屏之后，您还可以在 PB 的 Tools 菜单中点击“Remote Registry Editor”来运行远程注册表编辑工具查看并修改 wince 的注册表内容，如下图，在此就不作更详细的解释了。

10.3 创建 EVC 的 Hello,World，并编译下载到开发板运行

我们将不需要编写任何代码，使用向导创建一个最简单的 EVC 应用程序，并通过 USB 同步连接的方式下载到开发板运行起来，这是我们的第一个基于 EVC 的 Hello, World 程序，在这里我们主要向您展示了最基本的开发步骤。

说明：使用基于 mini2440_SDK 的 EVC 不需要安装庞大的 PB5 平台即可进行应用开发。

(1)打开运行 EVC 集成开发环境，点 File → New… 如图

(2)出现程序向导窗口，选择 WCE MFC AppWizard(exe)，在 Project Name 中并输入项目名字，在 Location 中选择存放位置，如图，点“OK”继续

(3)出现如图窗口，选择语言设置为英语，其他为默认，点“Next”继续

(4)出现如图窗口，所有设置保持默认，点“Next”继续

(5)出现如图界面，保持默认设置，点“Next”继续

(6)出现如图界面，点“Finish”继续

(7)出现 New Project Information 窗口，点“OK”完成向导

(8)回到 EVC 主界面，点 Build → Set Active Platform…以设置目标板连接

(9)因为我们只安装了导出的 mini2440_SDK，因此在这里只有一项选择，如图选择，点“OK”返回即可

(10)确认 PC 同步已经建立并连接正常(同时必须连接好网线)，如图

(11)在 EVC 主界面，点 Build → Rebuild All，如图，系统开始编译你所创建的工程文件：

(12)编译完毕，系统会自动下载下载编译好的可执行文件到开发板的根目录(打开“我的电脑”就可以看到)，双击并运行它，如图：

10.4 创建 VS2005/2008 应用程序，并编译下载到开发板运行

注意：必须要安装 PB5 的补丁程序，才能在定制内核时出现.NET 2.0 选项，并使选择使用它才能支持 VS2005/2008 应用程序。(缺省的内核稍写文件和示例工程已经添加了该选项)如图：

下面是使用 VS2008 的基本开发步骤：

10.4.1 创建项目

(1)打开运行 VS2008，点菜单文件→新建→项目，如图：

(2)出现“新建项目”窗口，在“项目类型”一栏中点击选择“Visual Basic → 智能设备”，在“模板”一栏中选择“智能设备项目”，在“名称”一栏中输入“my2440”，其他采用缺省设置，点“确定”继续：

(3)出现“添加新只能设备项目”窗口，在“目标平台”下拉列表中选择“Windows CE”，在“.NET Compact Framework 版本”下拉列表中选择“.NET Compact Framework Version 2.0”，在“模板”中选择“设备应用程序”，点“确定”继续：

(4)出现如图工作界面

这时，可以调整一下“Form”的大小，通过“工具箱”添加一些控件到“Form”中，如图：

(5)双击控件“Button1”出现代码设计窗口，输入以下简单代码，它将实现：当点击 Button1 按钮时，窗口标题会显示代码中设置的内容，如图：

保存好以上创建的项目文件。

10.4.2 设置连接开发板

(1) 确认 PC 同步已经建立并连接正常(可以不必连接网线)，如图

(2)点 VS2008 菜单“工具”→ “选项”，出现“选项”窗口，
在左侧一栏中选择“设备工具”→ “设备”，
在右侧中的各个下拉列表中作如图选择：

点“属性”按钮，出现“Windows CE 设备 属性窗口”，选择如图：

点“配置”按钮，出现“配置 TCP/IP 传输”窗口，如图选择，点“确定”返回 VS2008 工作界面。

(3)点 VS2008 菜单“工具”→“连接到设备”，出现“连接到设备”设置窗口，如图进行选择设置：

点“连接”按钮，此时 VS2008 开始和开发板进行连接握手：

稍等一会，出现连接成功的提示，点“关闭”按钮返回 VS2008 工作主界面：

10.4.3 编译下载程序到开发板运行

(1)接上面的步骤，点菜单“调试”→ “启动调试”或者直接按 F5 键开始调试过程。

(2)出现“部署 my2440”窗口，选择“Windows CE 设备”，并点“部署”按钮开始部署，如图

(3)如果程序没有问题，等待一会它会直接下载到开发板并运行，点击一下“Button1”按钮，窗口标题改变如图，这正如我们在代码中的设置， 如图：

通过 VS2008，你不仅可以编写 Visual Basic 程序，还可以使用 Visual C#和 Visual C++进行程序设计，通过同步连接开发板，你还可以实现单步调试运行。

10.5　LED 驱动程序编写及测试示例

LED 驱动程序在 BSP 中的位置：\mini2440\Src\Drivers\LEDdriver
LED 驱动程序文件的名称：LEDDriver.cpp
LED 测试程序位于光盘目录：\WindowsCE 应用程序示例代码\LEDtest

本小节以一个简单的 LED 驱动程序作为例子，介绍了 Windows CE 下常规流设备驱动程序编写的一般步骤和方法，用户可以通过更改其中的代码实现类似的其他 IO 控制功能。

10.5.1 了解硬件连接

开发板上 4 个 LED 引脚连接如图所示：

可见它们使用了 CPU 的 GPB 端口，关于 GPB 端口的描述及其使用在 S3C2440 数据手册中有如下描述：

GPB 端口描述(见 S3C2440.pdf 第 276 页)：

GPB8	Input/output	nXDREQ1
GPB7	Input/output	nXDACK1
GPB6	Input/output	nXBREQ
GPB5	Input/output	nXBACK

GPBCON 端口描述(见 S3C2440.pdf 第 284 页)：

GPB8	[17:16]	00 = Input 10 = nXDREQ1	01 = Output 11 = Reserved
GPB7	[15:14]	00 = Input 10 = nXDACK1	01 = Output 11 = Reserved
GPB6	[13:12]	00 = Input 10 = nXBREQ	01 = Output 11 = reserved
GPB5	[11:10]	00 = Input 10 = nXBACK	01 = Output 11 = reserved

GPBDAT 端口描述(见 S3C2440.pdf 第 284 页)：

GPBDAT	Bit	Description
GPB[10:0]	[10:0]	When the port is configured as input port, the corresponding bit is the pin state. When the port is configured as output port, the pin state is the same as the corresponding bit. When the port is configured as functional pin, the undefined value will be read.

因此，要使用 GPB5~8，需要：
(1) 设置 GPBCON，以使引脚功能为 IO 功能输出
(2) 要点亮 LED, GPBDAT 对应的位要设置为 0
(3) 要熄灭 LED, GPBDAT 对应的位要设置为 1

10.5.2 编写 LED 流式驱动程序

流式驱动是 WinCE 驱动程序的一种常规方式，应用程序通过文件系统，透过 DeviceManager 以访问文件的形式访问驱动程序，调用 IOCTL 向驱动程序下达指令。所有的流式驱动程序都需实现一组统一的接口。

流式驱动框架：

流式驱动程序接口函数(摘自 PB5 帮助文档)

Programming element	Description
XXX_Close (Device Manager)	This function is required to access the device with CreateFile. If you implement **XXX_Close**, you must implement **XXX_Open**.
XXX_Deinit (Device Manager)	This function is required by drivers loaded by ActivateDeviceEx, ActivateDevice, or RegisterDevice.

XXX_Init (Device Manager)	This function initializes a device. It is called by Device Manager. This function is required by drivers loaded by **ActivateDeviceEx**, **ActivateDevice**, or **RegisterDevice**.
XXX_IOControl (Device Manager)	This function sends a command to a device. This function might or might not be required, depending on the device capabilities that the driver exposes. This function requires an implementation of **XXX_Open** and **XXX_Close**.
XXX_Open (Device Manager)	This function opens a device for reading, writing, or both. An application indirectly invokes this function when it calls **CreateFile** to obtain a handle to a device. This function is required to access the device with **CreateFile**.
XXX_PowerDown (Device Manager)	Optional. This function ends power to the device. It is useful only with devices that can be shut off under software control.
XXX_PowerUp (Device Manager)	Optional. This function restores power to a device.
XXX_Read (Device Manager)	This function reads data from the device identified by the open context. This function might or might not be required, depending on the device capabilities that the driver exposes. This function requires an implementation of **XXX_Open** and **XXX_Close**.
XXX_Seek (Device Manager)	This function moves the data pointer in the device. This function might or might not be required, depending on the device capabilities that the driver exposes. This function requires an implementation of **XXX_Open** and **XXX_Close**.
XXX_Write (Device Manager)	This function writes data to the device. This function might or might not be required, depending

	on the device capabilities that the driver exposes. This function requires an implementation of **XXX_Open** and **XXX_Close**.

据此，我们要实现的 LED 驱动主要源代码如下：

```
#include <windows.h>
//#include <ceddk.h>
#include <nkintr.h>
#include <pm.h>
#include "pmplatform.h"
#include "Pkfuncs.h"
#include "s2440.h"

#define IO_CTL_LED_1_ON 0x01
#define IO_CTL_LED_2_ON 0x02
#define IO_CTL_LED_3_ON 0x03
#define IO_CTL_LED_4_ON 0x04
#define IO_CTL_LED_ALL_ON 0x05
#define IO_CTL_LED_1_OFF 0x06
#define IO_CTL_LED_2_OFF 0x07
#define IO_CTL_LED_3_OFF 0x08
#define IO_CTL_LED_4_OFF 0x09
#define IO_CTL_LED_ALL_OFF 0x0a

volatile IOPreg        *s2440IOP = (IOPreg *)IOP_BASE;
volatile INTreg        *s2440INT = (INTreg *)INT_BASE;

BOOL mInitialized;
void Virtual_Alloc();                          // Virtual allocation

void Virtual_Alloc()
{

      // GPIO Virtual alloc
 s2440IOP = (volatile IOPreg *) VirtualAlloc(0,sizeof(IOPreg),MEM_RESERVE, PAGE_NOACCESS);
 if(s2440IOP == NULL) {
       RETAILMSG(1,(TEXT("For s2440IOP: VirtualAlloc faiLED!\r\n")));
 }
 else {
       if(!VirtualCopy((PVOID)s2440IOP,(PVOID)(IOP_BASE),sizeof(IOPreg),PAGE_READWRITE   |
```

```
PAGE_NOCACHE )) {
                RETAILMSG(1,(TEXT("For s2440IOP: VirtualCopy faiLED!\r\n")));
            }
        }


    }

    BOOL WINAPI
    DllEntry(HANDLE  hinstDLL,
                DWORD dwReason,
                LPVOID /* lpvReserved */)
    {
     switch(dwReason)
     {
     case DLL_PROCESS_ATTACH:
            DEBUGREGISTER((HINSTANCE)hinstDLL);
            return TRUE;
     case DLL_THREAD_ATTACH:
            break;
     case DLL_THREAD_DETACH:
            break;
     case DLL_PROCESS_DETACH:
            break;
    #ifdef UNDER_CE
     case DLL_PROCESS_EXITING:
            break;
     case DLL_SYSTEM_STARTED:
            break;
    #endif
     }

     return TRUE;
    }


    BOOL LED_Deinit(DWORD hDeviceContext)
    {
     BOOL bRet = TRUE;

     RETAILMSG(1,(TEXT("USERLED: LED_Deinit\r\n")));
```

```
 return TRUE;
}

BOOL LEDGpioInit()
{
 RETAILMSG(1,(TEXT("LED_Gpio_Setting----\r\n")));
 s2440IOP->rGPBCON   = (s2440IOP->rGPBCON   &~(3 << 10)) | (1<< 10); // GPB5 == OUTPUT.
 s2440IOP->rGPBCON   = (s2440IOP->rGPBCON   &~(3 << 12)) | (1<< 12); // GPB6 == OUTPUT.
 s2440IOP->rGPBCON   = (s2440IOP->rGPBCON   &~(3 << 14)) | (1<< 14); // GPB7 == OUTPUT.
 s2440IOP->rGPBCON   = (s2440IOP->rGPBCON   &~(3 << 16)) | (1<< 16); // GPB8 == OUTPUT.
 return TRUE;
}




DWORD LED_Init(DWORD dwContext)
{

 RETAILMSG(1,(TEXT("LED_Init----\r\n")));

 // 1. Virtual Alloc
 Virtual_Alloc();

 LEDGpioInit();


 mInitialized = TRUE;
 return TRUE;
}

//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
BOOL LED_IOControl(DWORD hOpenContext,
                   DWORD dwCode,
                   PBYTE pBufIn,
                   DWORD dwLenIn,
                   PBYTE pBufOut,
                   DWORD dwLenOut,
                   PDWORD pdwActualOut)
```

```
{
 LEDGpioInit();
 switch(dwCode)
 {
 case IO_CTL_LED_1_ON:
        s2440IOP->rGPBDAT=s2440IOP->rGPBDAT&~(0x1<<5);
        break;
 case IO_CTL_LED_2_ON:
        s2440IOP->rGPBDAT=s2440IOP->rGPBDAT&~(0x1<<6);
        break;
 case IO_CTL_LED_3_ON:
        s2440IOP->rGPBDAT=s2440IOP->rGPBDAT&~(0x1<<7);
        break;
 case IO_CTL_LED_4_ON:
        s2440IOP->rGPBDAT=s2440IOP->rGPBDAT&~(0x1<<8);
        break;
 case IO_CTL_LED_ALL_ON:
        s2440IOP->rGPBDAT=s2440IOP->rGPBDAT&~(0xF<<5);
        break;
 case IO_CTL_LED_1_OFF:
        s2440IOP->rGPBDAT=s2440IOP->rGPBDAT|(0x1<<5);
        break;
 case IO_CTL_LED_2_OFF:
        s2440IOP->rGPBDAT=s2440IOP->rGPBDAT|(0x1<<6);
        break;
 case IO_CTL_LED_3_OFF:
        s2440IOP->rGPBDAT=s2440IOP->rGPBDAT|(0x1<<7);
        break;
 case IO_CTL_LED_4_OFF:
        s2440IOP->rGPBDAT=s2440IOP->rGPBDAT|(0x1<<8);
        break;
 case IO_CTL_LED_ALL_OFF:
        s2440IOP->rGPBDAT=s2440IOP->rGPBDAT|(0xF<<5);
        break;
 default:
        break;
 }

 RETAILMSG(1,(TEXT("LED:Ioctl code = 0x%x\r\n"), dwCode));
 return TRUE;
}
```

```
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
DWORD LED_Open(DWORD hDeviceContext, DWORD AccessCode, DWORD ShareMode)
{
  RETAILMSG(1,(TEXT("USERLED: LED_Open\r\n")));
  return TRUE;
}

//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
BOOL LED_Close(DWORD hOpenContext)
{
  s2440IOP->rGPBDAT=s2440IOP->rGPBDAT&(0xF<<5);
  RETAILMSG(1,(TEXT("USERLED: LED_Close\r\n")));
  return TRUE;
}

//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
void LED_PowerDown(DWORD hDeviceContext)
{
  RETAILMSG(1,(TEXT("USERLED: LED_PowerDown\r\n")));

  //RETAILMSG(1,(TEXT("CAMERA: LED_PowerDown, m_Dx = D%u, init %d \r\n"), m_Dx,
mInitialized));
}

//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
void LED_PowerUp(DWORD hDeviceContext)
{
  RETAILMSG(1,(TEXT("USERLED: LED_PowerUp\r\n")));

}

//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
DWORD LED_Read(DWORD hOpenContext, LPVOID pBuffer, DWORD Count)
{
  RETAILMSG(1,(TEXT("USERLED: LED_Read\r\n")));
```

```
 return TRUE;
}

//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
DWORD LED_Seek(DWORD hOpenContext, long Amount, DWORD Type)
{
 RETAILMSG(1,(TEXT("USERLED: LED_Seek\r\n")));
 return 0;
}

//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
DWORD LED_Write(DWORD hOpenContext, LPCVOID pSourceBytes, DWORD NumberOfBytes)
{
 RETAILMSG(1,(TEXT("USERLED: LED_Write\r\n")));
 return 0;
}
```

10.5.3 把 LED 驱动程序添加到 BSP 中以编译

LED 驱动程序必须要编译到内核中才能使用，要把 LED 驱动程序编译到内核中，需要做如下步骤：

(1) 建立 LED 驱动程序目录

在 mini2440\DRIVERS 下建立 LEDdriver 目录，并在 dirs 文件中加入此目录，使系统编译 bsp 的时候可以编译这个文件

(2)为 LED 驱动创建 Makefile 文件

可以参考其他驱动程序示例创建 Makefile 文件。

在 mini2440\DRIVERS\LEDriver\目录中建立 makefile 文件，内容如下：

```
#
# DO NOT EDIT THIS FILE!!!    Edit .\sources. if you want to add a new source
# file to this component.    This file merely indirects to the real make file
# that is shared by all the components of Peg
#
!INCLUDE $(_MAKEENVROOT)\makefile.def
```

(3)为 LED 驱动创建 source 文件

可以参考其他驱动程序实例创建 source 文件。

在 mini2440\DRIVERS\LEDriver\目录中建立 source 文件，内容如下：

```
!if 0
```

```
        File:     sources

        Author: jeffmi

        Copyright (c) 1995-2002 Microsoft Corporation.    All rights reserved.
    !endif

    RELEASETYPE=PLATFORM
    TARGETNAME=LEDDriver
    TARGETTYPE=DYNLINK
    DLLENTRY=DllEntry

    TARGETLIBS= \
        $(_COMMONSDKROOT)\lib\$(_CPUINDPATH)\coredll.lib \

    MSC_WARNING_LEVEL = $(MSC_WARNING_LEVEL) /W3 /WX

    INCLUDES= \
        $(_TARGETPLATROOT)\inc; \
        $(_COMMONOAKROOT)\inc; \

$(_PUBLICROOT)\common\oak\inc;$(_PUBLICROOT)\common\sdk\inc;$(_PUBLICROOT)\co
mmon\ddk\inc;    \
        ..\..\inc

    SOURCES= \
        leddriver.cpp \
```

(4)编写 leddriver.def 导出 Dll 符号

可以参考其他驱动程序示例创建 leddriver.def 文件。

```
;
; Windows CE LED Driver. Written by capbily

LIBRARY userLED
EXPORTS
LED_Close
LED_Deinit
LED_Init
LED_IOControl
LED_Open
LED_PowerDown
```

LED_PowerUp
LED_Read
LED_Seek
LED_Write

(5)把 LED 驱动加入内核

在配置文件 platform.bib 中加入以下内容：

```
;leddriver
leddriver.dll                $(_FLATRELEASEDIR)\leddriver.dll  NK SH
```

(6)把 LED 驱动加入注册表

在注册表文件 platform.reg 中加入以下内容：

```
[HKEY_LOCAL_MACHINE\Drivers\BuiltIn\LEDdriver]
    "Prefix"="LED"
    "Dll"="LEDdriver.dll"
```

(7)重新编译内核

重新编译内核，点 PB5 的主菜单 Build OS → Sysgen 即可，这样就可以生成包含以上 LED 驱动的内核映象文件 NK.bin 和 NK.nb0 了。

10.5.4 编写并编译 LED 测试应用程序

示例文件在光盘中的位置：**\Windows CE5.0\LEDtest**
示例文件所使用的编译器：eMbedded Visual C++ 4.0 + SP4 补丁 + Mini2440-CE5-SDK
SP4 补丁在光盘中的位置：\Embedded VisualC++\SP\evc4sp4\DISK1
Mini2440-CE5-SDK 安装文件在光盘中的位置：\WindowsCE5.0\SDK

要在应用程序中调用驱动程序，IO Control 是最常用的方法，它的一般流程如下：
在应用程序中：

```
HANDLE
leddriver=CreateFile(L"LED1:",GENERIC_READ|GENERIC_WRITE,0,NULL,OPEN_E
XISTING,0,NULL );
ReadFile(leddriver，……..)
DeviceIoControl(leddriver,IO_CTL_LED_CLEAR,NULL,0,NULL,0,NULL,NULL);
```

在驱动程序中：
当 CreateFile 被调用时，调用 XXX_Open
当 ReadFile 被调用时，调用 XXX_Read
当 DeviceIoControl 被调用时，调用 XXX_IOContrl,根据 IO_CTL_XXX 的数值，决定执行的操作。

在 LED 驱动中，实现 10 个 IOControl 分支，处理 10 个命令请求，对应点亮 1~4 号灯，，点亮所有 LED，关闭 1~4 号等，关闭所有 LED 灯。

LED 测试程序的主要代码如下：

```
#define IO_CTL_LED_1_ON 0x01
#define IO_CTL_LED_2_ON 0x02
#define IO_CTL_LED_3_ON 0x03
#define IO_CTL_LED_4_ON 0x04
#define IO_CTL_LED_ALL_ON 0x05
#define IO_CTL_LED_1_OFF 0x06
#define IO_CTL_LED_2_OFF 0x07
#define IO_CTL_LED_3_OFF 0x08
#define IO_CTL_LED_4_OFF 0x09
#define IO_CTL_LED_ALL_OFF 0x0a

void CKEYTESTDlg::OnButton1()
{
// TODO: Add your control notification handler code here
DeviceIoControl(leddriver,IO_CTL_LED_1_ON, NULL,0,NULL,0,NULL,NULL);

}

void CKEYTESTDlg::OnButton2()
{
// TODO: Add your control notification handler code here
DeviceIoControl(leddriver,IO_CTL_LED_2_ON,NULL,0,NULL,0,NULL,NULL);
}

void CKEYTESTDlg::OnButton3()
{
// TODO: Add your control notification handler code here
DeviceIoControl(leddriver,IO_CTL_LED_3_ON,NULL,0,NULL,0,NULL,NULL);
}

void CKEYTESTDlg::OnButton4()
{
// TODO: Add your control notification handler code here
DeviceIoControl(leddriver,IO_CTL_LED_4_ON,NULL,0,NULL,0,NULL,NULL);
}

void CKEYTESTDlg::OnButton5()
{
// TODO: Add your control notification handler code here
DeviceIoControl(leddriver,IO_CTL_LED_ALL_ON,NULL,0,NULL,0,NULL,NULL);
}
```

```
void CKEYTESTDlg::OnButton6()
{
// TODO: Add your control notification handler code here
DeviceIoControl(leddriver,IO_CTL_LED_1_OFF,NULL,0,NULL,0,NULL,NULL);
}

void CKEYTESTDlg::OnButton7()
{
// TODO: Add your control notification handler code here
DeviceIoControl(leddriver,IO_CTL_LED_2_OFF,NULL,0,NULL,0,NULL,NULL);
}

void CKEYTESTDlg::OnButton8()
{
// TODO: Add your control notification handler code here
DeviceIoControl(leddriver,IO_CTL_LED_3_OFF,NULL,0,NULL,0,NULL,NULL);
}

void CKEYTESTDlg::OnButton9()
{
// TODO: Add your control notification handler code here
DeviceIoControl(leddriver,IO_CTL_LED_4_OFF,NULL,0,NULL,0,NULL,NULL);
}

void CKEYTESTDlg::OnButton10()
{
// TODO: Add your control notification handler code here
DeviceIoControl(leddriver,IO_CTL_LED_ALL_OFF,NULL,0,NULL,0,NULL,NULL);
}
```

要编译光盘中测 LEDtest 测试程序，步骤如下：
(1)把 LEDtest 复制到硬盘的某个目录，比如 D:\Work，去掉其只读属性。
(2)打开运行 EVC，点 File→ Open，找到 LEDtest 工程文件，如图：

(3)参考9.4一节的方法先设置好开发板连接(**ActiveSync 同步和网络都必须连接正常**)，点菜单 Build → Rebuild All 开始重新编译 LEDtest.exe 程序，编译完毕，EVC 会自动连接下载到开发板的\Windows 目录，同时在 D:\work\lcdtest\ARMV4IRel 目录下生成 LCDtest.exe 可执行程序，点击运行 LCDtest 测试程序，如图，可以按照提示进行控制开发板上的 LED。如图：

10.5.5 把 LED 测试程序添加到内核，并建立桌面快捷方式

为了把上一小节编译出的 LEDtest.exe 可执行程序永久写入到 NK.bin/NK.nb0 中，以便每次开机时都能从桌面的快捷方式中执行它，我们采用以下方法步骤：

(1)把 LEDtest.exe 复制到 mini2440\FILES 目录中。

(2)使用 Windows 自带的记事本程序创建 LEDtest.lnk 快捷方式文件：

.lnk 文件其实是一个文本文件，它包含用于链接目标的命令行以及命令行的长度，其格式为“<length>#<command line>”，其中 length 是#后所有字符的个数，在此，LEDtest.lnk 的内容如下：

20#\Windows\LEDtest.exe

注意：当把 LEDtest.lnk 后缀改为.lnk 后，使用记事本一般就无法打开了。

我们把这个文件也放入 mini2440\FILES 目录中。

(3)打开 platform.bib 文件，添加如下内容：

```
LEDtest.exe      $(_FLATRELEASEDIR)\QQtest.exe      NK    U
LEDtest.lnk      $(_FLATRELEASEDIR)\QQtest.lnk      NK    U
```

这样，执行 SYSGEN 的时候会把这两个文件加入到内核中，最后他们会存在于开发板的\Windows 目录中。

(4)打开 platform.dat，加入以下内容：

Directory("\windows\桌面"):-File("LED 测试.lnk","\windows\LEDtest.lnk")

这将会在桌面出现名称位“LED 测试”的快捷方式，它是 LEDtest.lnk 的一个拷贝，其内容和 LEDtest.lnk 是一样的。

(5)最后执行菜单 Builder → Sysgen，生成 NK.bin 和 NK.nb0，把它们烧写或者下载到开发板启动后，就会在桌面看到“LED 测试”快捷方式了。

结束语

本手册详细地向您介绍了如何开发和使用 Linux 或者 WindowsCE，这些都是一些基础性的东西，实际上，使用任何开发板都有一个类似的过程；我们相信，只要有兴趣，并且耐心的按照本手册去做了，您一定会掌握它，并体会到其中的乐趣；俗话说“师父领进门，修行在个人”，在技术不断更新和变革的今天，我们每个人都需要不断的学习和补充，我们希望能和您

一起成长。

附录 使用 H-JTAG 快速烧写 BIOS 到开发板

1. 1 H-JTAG 简介

说明：关于部分 H-JTAG 的说明摘 H-Jtag 说明手册，请参考其官方网站 http://www.hjtag.com

当前 ARM 的学习与开发非常流行，由于 ARM 的软件开发相对以前单片机而言更加复杂，硬件上的考虑也比较多，因此选择一个好的调试方法将可以使得开发的除错过程变得更加直接和简单。

现在市面上有很多可用于 ARM 调试的仿真器出售，然而其价格往往都比较贵。这些仿真器一般都有其专用的软件和硬件，在速度和 flash 编程等方面有各自的优势。然而对初学者而言，这些仿真器的成本都太高。而简易仿真器的出现，使得大家可以使用甚至自制 ARM 仿真器硬件。

有了调试器的硬件，还要加上调试代理软件，作为中介，将调试器前端软件（比如 AXD）的调试信息与目标板上的目标芯片交互，才能最终完成仿真的任务。目前，可以免费使用的简易 ARM 仿真器的代理软件很多，差别也比较大，主要表现在易用程度，目标器件支持，调试速度等方面。H-JTAG 作为近来新推出的简易 ARM 仿真器调试代理，其支持器件比较多，支持的调试器前端软件也比较多，特别是支持 keil，其调试速度也很有优势。

H-JTAG 是由 twentyone 推出的一款免费调试代理软件。官方主页为：http://www.hjtag.com/

目前的版本为 0.4.4，支持下列特性(更新的版本请到 H-JTAG 网站下载试用)：

1. 支持 RDI 1.5.0 与 1.5.1;
2. 支持 ARM7 与 ARM9（包括 ARM9E-S 与 ARM9EJ-S）;
3. 支持 thumb 与 arm 指令集;
4. 支持 little-endian 与 big-endian;
5. 支持 semihosting;
6. 支持 wiggler, sjf-jtag 以及用户自定义的简易调试器硬件接口;
7. 支持 WINDOWS 9.X/NT/2000/XP;
8. 支持 flash 器件的编程

光盘中还提供了最新的 H-JTAG 软件，它可以支持更多型号的 NOR Flash 烧写，因时间关系，我们暂时没有对其做详细的安装使用说明，它们的步骤和界面都是大同小异的，请用户自行尝试使用。

注：本开发板所用的 JTAG 板即为 SJF-JTAG

2.2 安装并设置 H-JTAG

注意：以下过程所使用的并口是计算机主板自带的，使用 USB 扩展的并口一般是不行的，使用 PCI 扩展的并口有可能不行，我们对此没有测试过。

(1)安装 H-JTAG

H-JTAG 安装文件位于光盘的“**Windows 平台工具\H-JTAG**”目录，双击运行，按照其提示安装即可。

安装完毕，会在桌面生成 H-JTAG 和 H-Flasher 快捷方式，双击运行 H-JTAG，程序将自动检测是否连接了 JTAG 设备，因为之前我们还没有做任何设置，所以会跳出一个提示窗口：

点击确定，进入程序主界面，因为没有连接任何目标器件，因此显示如图所示：

(2)设置 JTAG 端口

在 H-JTAG 主界面的菜单里点 Setting->Jtag Settings，作如下图所示设置，点 OK 返回主界面。

(3)设置初始化脚本

把光盘“**Windows 平台工具\H-JTAG**”目录中的 **FriendlyARM2440.his** 和 H-Flasher_mini2440.hfc 文件复制到 H-JTAG 的安装目录，如图：

在 H-JTAG 的主界面，点 **Script->Init Script**，跳出 Init Script 窗口，点该窗口下面的 Load 按钮，找到并选择打开刚刚复制的 **FriendlyARM2440.his** 文件，如图：

这时，Init Script 窗口会被载入的脚本填充，如图，注意不要点选“Enable Auto Init”，点 OK 退回 H-JTAG 主界面：

Idx	Cmd	Width	Address	Value
1	Setmem	32-Bit	0x53000000	0x00000000
2	Setmem	32-Bit	0x4A000008	0xFFFFFFFF
3	Setmem	32-Bit	0x4A00001C	0x000007FF
4	Setmem	32-Bit	0x53000000	0x00000000
5	Setmem	32-Bit	0x56000050	0x000055AA
6	Setmem	32-Bit	0x4C000014	0x00000007
7	Setmem	32-Bit	0x4C000000	0x00FFFFFF
8	Setmem	32-Bit	0x4C000004	0x00061012
9	Setmem	32-Bit	0x4C000008	0x00040042
10	Setmem	32-Bit	0x48000000	0x22111120
11	Setmem	32-Bit	0x48000004	0x00002F50

(4)检测目标器件

注意：以下过程所使用的并口是计算机主板自带的，使用 USB 扩展的并口一般是不行的，使用 PCI 扩展的并口有可能不行，我们对此没有测试过。

使用开发板附带的 JTAG 小板连接开发板的 JTAG 接口，并接上打开电源。点主菜单 **Operations->Detect Target**，或者点工具栏相应的图标也可以，这时就可以看到已经检测到目标器件了。

提示：如果没有设置初始化脚本，也可以检测到 CPU，但无法进行下面的单步调试。

2. 3 设置 Flash 型号并烧写 BIOS

注意：执行以下步骤之前，要确保开发板选择从 **Nor Flash** 启动，切记！

接上面的步骤：

(1)点 H-JTAG 主菜单的 Flasher → Start H-Flasher 打开 H-Flasher 烧写程序窗口，如图：

(2)在 H-Flasher 窗口菜单中选择“Load”，出现打开文件选择窗口，选择上面步骤复制的 H-Flasher_mini2440.hfc 文件，如图：

因目标板上所使用的 NOR Flash 型号不同，Flasher 烧写器的初始化脚本也可能会使用 H-Flasher_S29AL016T.hfc/H-Flasher_SST39VF1601.hfc，请用户根据实际情况选择。

(3)可以看到 Flash 初始文件已经被载入并显示在 H-Flasher 窗口标题栏中，如图：

这时，点 H-Flasher 左侧导航栏的“4 Programming”，出现如图界面：

(4) 点一下“Check”按钮，H-Flasher 将会探测到开发板所用的 Nor Flash 所用的型号为 SST39VF1601，如图(注意：早期的 mini2440/micro24440 使用兼容芯片 AM29LV160DB，图未变)：

(5)点 Type 下拉列表，选择“Plain Binary Format”，如图：

再点 Src File 右侧的浏览按钮，选择所要烧写的文件 supervivi，并在 Dst Addr 一栏中输入烧写的起始地址“0”,如图：

(6)点“Program”按钮开始烧写 supervivi，如图：

烧写结束，“Stop”按钮会变为“Close”，点“Close”结束烧写并取下 JTAG 烧写板，如图：

(7)至此，您已经把 BIOS 烧写入 Nor Flash 中，如果您需要烧写更多的开发板，无需重复以上步骤(下次打开运行 H-JTAG 时会自动载入上次的配置)，可以直接接上 Jtag 线，打开电源，点“Check”先检测一下 Flash，再点“Program”就可以开始新的烧写了。

注意：目前 H-JTAG 只能用于烧写 Nor Flash，并不能直接烧写 Nand Flash，一些开发板厂商为了节省成本，很多都省掉了 Nor Flash，因此并不能用本节介绍的步骤快速简单的烧写 BIOS。

2.4 常见问题

其实我们不推荐初学者使用 H-JTAG 烧写 NOR FLASH，一旦操作错误就会导致无法正常进行后面步骤的操作。

有的用户可能尝试烧写了其他程序或者文件到 NOR FLASH 中，这时再使用 H-JTAG 烧写 Supervivi 有可能会失败，通常是因为系统复位或者开机后就马上执行 NOR FLASH 中的程序了，这导致 H-JTAG 无法正常执行。

可以尝试这样解决：在复位之后快速点 H-Flaser 的“Progarm”按钮，防止 NOR FLASH 中的程序进一步执行。