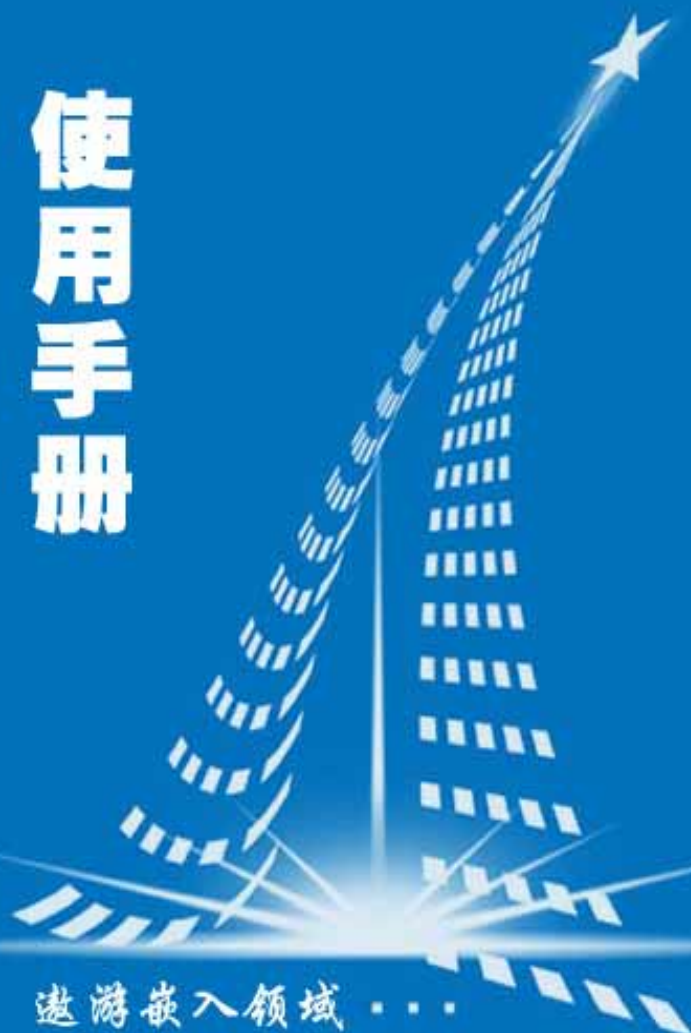


LJD-2410DVK-I

使用手册



遨游嵌入领域...

北京蓝海微芯科技发展有限公司

<http://www.bluemcu.com>

目 录

第一部分 硬件资源和非操作系统的软件开发	3
第一章 LJD-2410DVK-I 开发板硬件和软件资源介绍	3
1.1 LJD-2410DVK-I 开发板简介	3
1.2 操作系统支持的驱动	6
第二章 快速开始	6
2.1 非操作系统的软件编程	6
第三章 s3c2410 体系结构与外设	11
3.1 S3C2410 处理器简介	11
3.2 地址空间分配以及跳线设置	12
第四章 ADS1.2 集成开发环境	14
4.1 建立工程	14
第二部分 基于操作系统的软件开发	23
第五章 Linux 交叉编译环境的建立和内核编译	23
5.1 建立嵌入式 linux 开发环境	23
5.2 文件系统的制作	28
第六章 通过 BIOS 下载 linux 内核和文件文件系统	30
6.1 烧些内核（目标代码里的 zImage）	30
6.2 烧写文件系统（目标代码里的 myrootfs.cramfs）	32
第七章 linux 下的接口测试和应用软件开发	33
7.1 linux 下的接口测试	33
7.2 linux 下的简单应用程序的编写实例	35
第八章 简单的 linux 驱动程序的开发	37
第九章 图形界面开发入门	38
9.1 引言	38
9.2 图形用户界面 MiniGUI 简介	38
9.3 MiniGUI 在 S3C2410 开发板上的移植过程	39
第十章 PB4.2 的安装和镜像文件的定制	41
10.1 PB4.2 开发环境的安装	41
10.2 安装 BSP 包	43
10.3 创建新的 Platform 并生成新的 Image 文件	44
10.4 下载烧写内核镜像	51
附录 一 如何将开发板恢复到出厂设置	54
附录 二 开发板常见问题	55

版权声明

本手册版权归北京蓝海微芯科技发展有限公司所有,并保留一切权利。非经作者本人同意(书面形式),任何单位和个人不可擅自抄录本手册或全部,以任何形式用于商业目的,但可以自由传播。本产品所使用的嵌入式操作系统Linux/minigui/QT为开放源代码的自由软件,并遵循GNU 通用公共许可协议GPL。

版权所有Copyright© 2005 北京蓝海微芯科技发展有限公司

Copyright© 2005 Blue Ocean MicroChip Technology Co.,Ltd. All Rights Reserved

Support : BBS:<http://www.bluemcu.com>

Tel: 010-82684385/86/87

第一部分 硬件资源和非操作系统的软件开发

第一章 LJD-2410DVK-I 开发板硬件和软件资源介绍

1.1 LJD-2410DVK-I 开发板简介

中央处理器

— CPU: 三星S3C2410 , 主频203MHz ;

外部存储器

— 内存 : 64M 字节 ;

— NAND Flash : 64M 字节 (K9F1208 , 用户可自己更换为16M、32M 或128M的NandFlash)

串口

— 两个三线异步串行口 ;

网络接口

— 一个10M 网口 , 采用CS8900Q3 , 带联接和传输指示灯 ;

USB 接口

— 两个USB1.1 HOST 接口 ;

— 一个USB1.1 Device 接口 ;

红外通讯口

— 一个IRDA 红外线数据通讯口 ;

音频接口

— 采用S 接口芯片UDA1341 , 一路立体声音频输出接口可接耳机或音箱 ;

—— 支持录音，板子自带驻机体话筒可直接录音，另有一路话筒输入接口可接麦克风；存储

接口

—— 一个SD 卡接口，可接256M SD 卡；

—— 一个10针的AD接口；

—— 一个IDE 接口(接口信号均加了74LVTH162245 驱动)，可直接挂接硬盘；

LCD 和触摸屏接口

—— 板上集成了4 线电阻式触摸屏接口的相关电路；

—— 一个50 芯LCD 接口引出了LCD 控制器的全部信号，并且这些信号引脚都加了

74LV3245电平转换驱动，所以LCD 输出更加稳定可靠；

—— 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，屏幕分辨率可达到1024×768 像素；

—— 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，屏幕分辨率可达到1024×768 像素；

—— 标准配置为夏普256K 色240x320/3.5 英寸TFT 液晶屏，带触摸屏；

时钟源

—— 内部实时时钟（带有后备锂电池）；

复位电路

—— 一个复位按键，并采用专用复位芯片进行复位，稳定可靠；

调试及下载接口

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

电源接口

—— 5V 电源供电，带电源开关和指示灯；

其他

—— 四个小按键；

—— 一个蜂鸣器（带使能控制的短路块）；

—— 一个50 芯2 毫米间距双排标准连接器用作扩展口，引出了地址线、数据线、读写、片选、中断、IO 口、ADC、5V 和3.3V 电源、地等用户扩展可能用到的信号；；

操作系统

——支持linux（2.4版本内核）和WINCE4.2.NET

用户光盘上提供的开发工具和源代码：

- 1) ADS1.20 安装程序(评估版)；
- 2) 使用并支持ADS1.20 的JTAG 调试软件ARM DEBUG；
- 3) 烧写FLASH 的工具软件sjf.exe；
- 4) 串口工具软件sscom32.exe、DNW.exe、tftp.exe；
- 5) 64K 色（RGB565）图片字模软件；
- 6) USB Device 接口驱动程序；
- 7) BIOS 源代码（ADS1.20 的项目文件）；
- 8) 测试程序(ADS1.20 的项目文件，包含全部源代码)；
- 9) Linux for S3c2410 内核源码包以及编译工具，含CS8900驱动，UART 驱动USB HOST & DEVICE 驱动；
- 10) WINCE4.2.NET 板级支持包BSP for S3c2410；
- 11) 已经编译好并可在LJD-2410DVK-I 上运行的wince 内核，基于我公司提供的BSP；
- 12) 核心板和底板电路原理图（pdf 格式）；
- 13) 开发板使用手册（pdf 格式）；
- 14) 开发板上所用到的主要芯片手册；
- 15) Nandflash直接启动WINCE得启动BOOTLOADER和BIN文件。

LJD-2410DVK-I 套件包括:

- 1) 一块已测试好的LJD-2410DVK-I 开发板（包括LJD-2410DVK-I 核心板与底板）
- 2) LJD-2410DVK-I 用户光盘
- 3) 3.5" TFT 彩色LCD 板一块，带触摸屏(选配)
- 4) 一个SUPER JTAG 调试头（带20 芯排线）
- 5) 一条并口线（一边是公头一边是母头，一对一）
- 6) 一条串口线(两边都是母头，交叉串口线)
- 7) 一条网线(交叉网线线)
- 8) USB 线一条

- 9) 触摸笔一支（选配）
- 10) 一个 + 5V 直流电源
- 11) 一个包装盒

1.2 操作系统支持的驱动

支持Linux 和WINCE 操作系统。

Linux 支持的驱动

- 10M以太网
- USB HOST
- 音频
- LCD 和触摸屏
- 通用串口驱动
- CF 卡和IDE 驱动
- MMC 驱动

WINCE 下的驱动

- 10M 网络
- USB HOST
- 音频
- SD 卡
- LCD 和触摸屏驱动
- USB DEIVCE

第二章 快速开始

2.1 非操作系统的软件编程

将开发板连接好电源和串口线，设置串口终端软件（DNW），如图2-1所示：

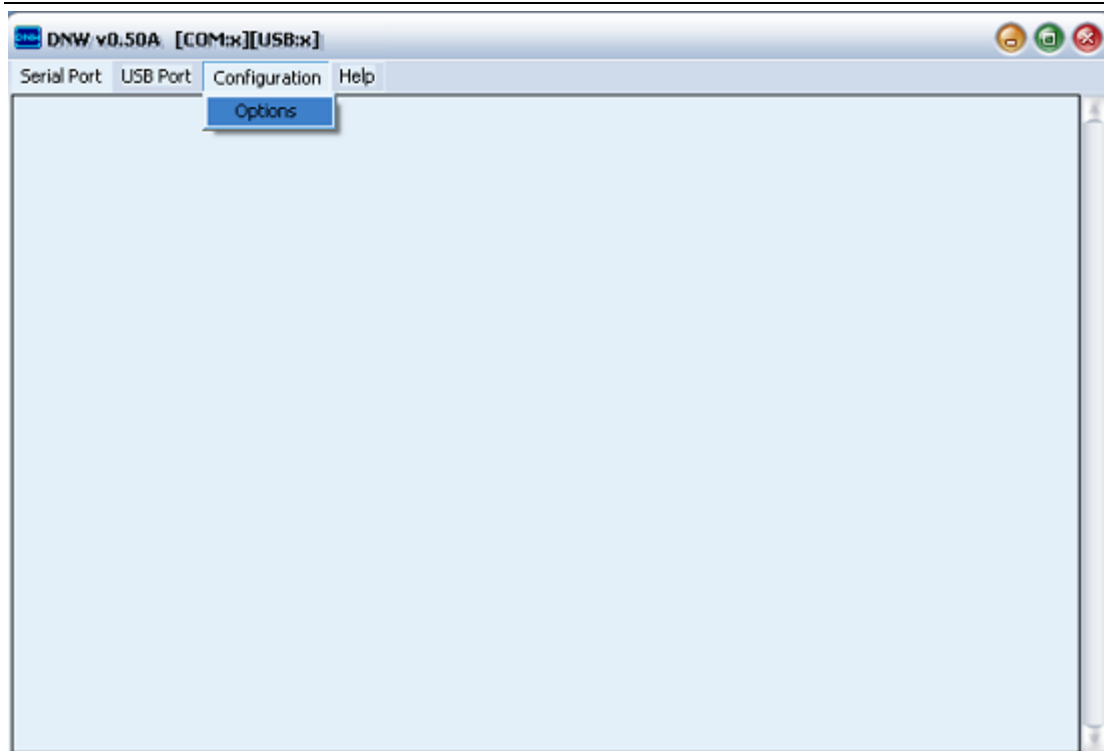


图 2-1

设置串口号和波特率，特别要注意 USB 的下载地址如图 2-2 所示：

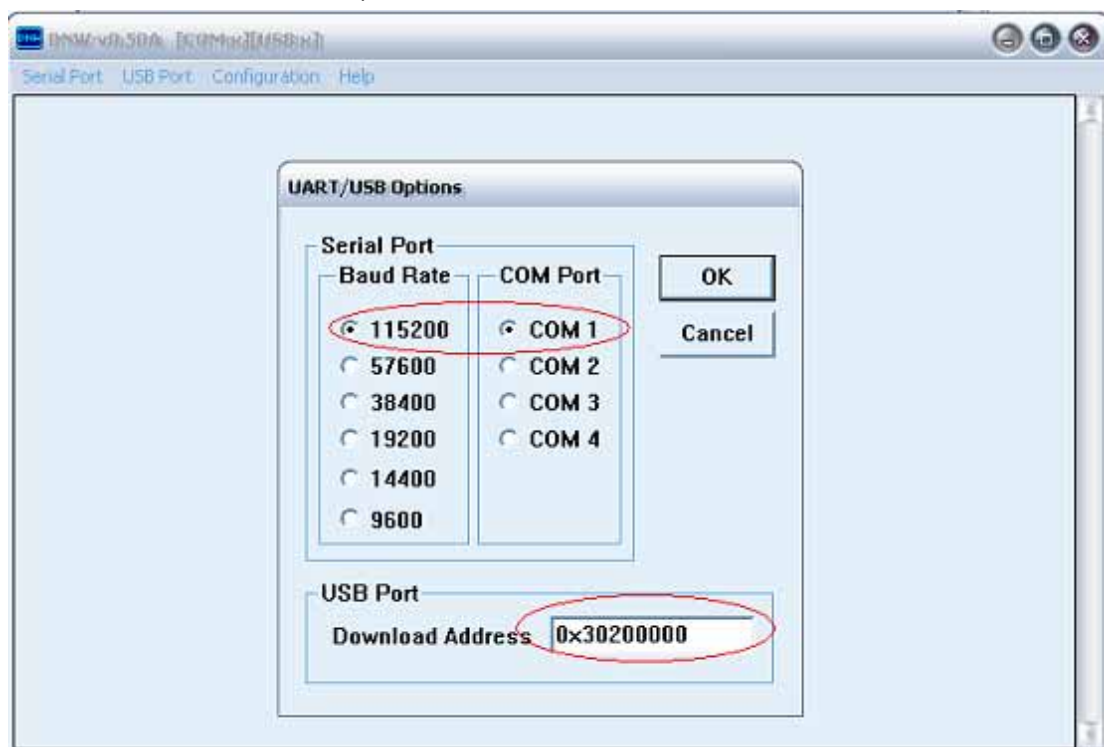


图 2-2

然后单击 OK，单击 Serial Port 菜单下的 Connect:

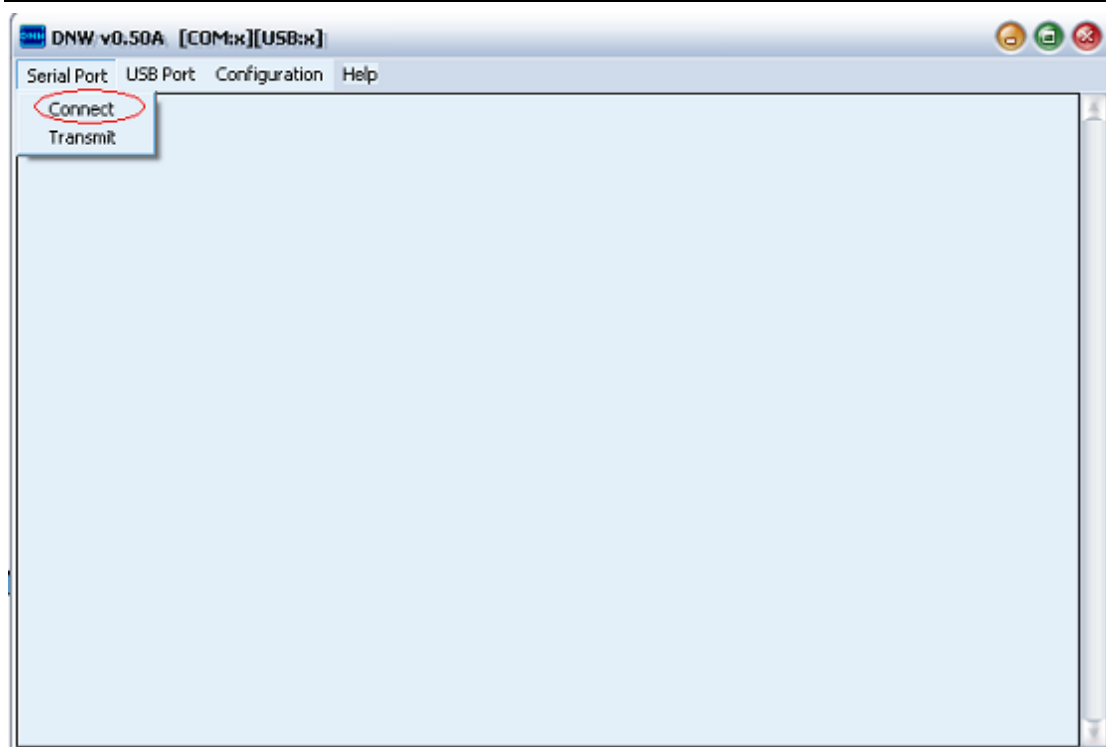


图 2-3

连接好的效果如下图 2-4 所示：

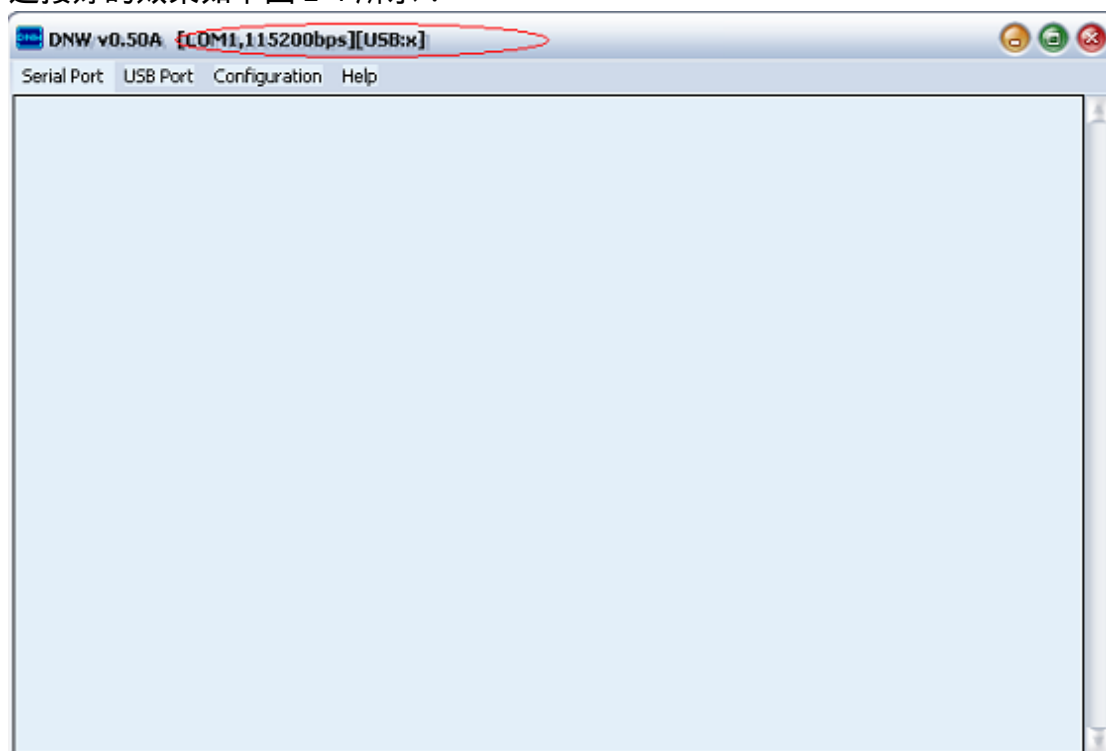


图 2-4

打开开发板电源显示如下信息：

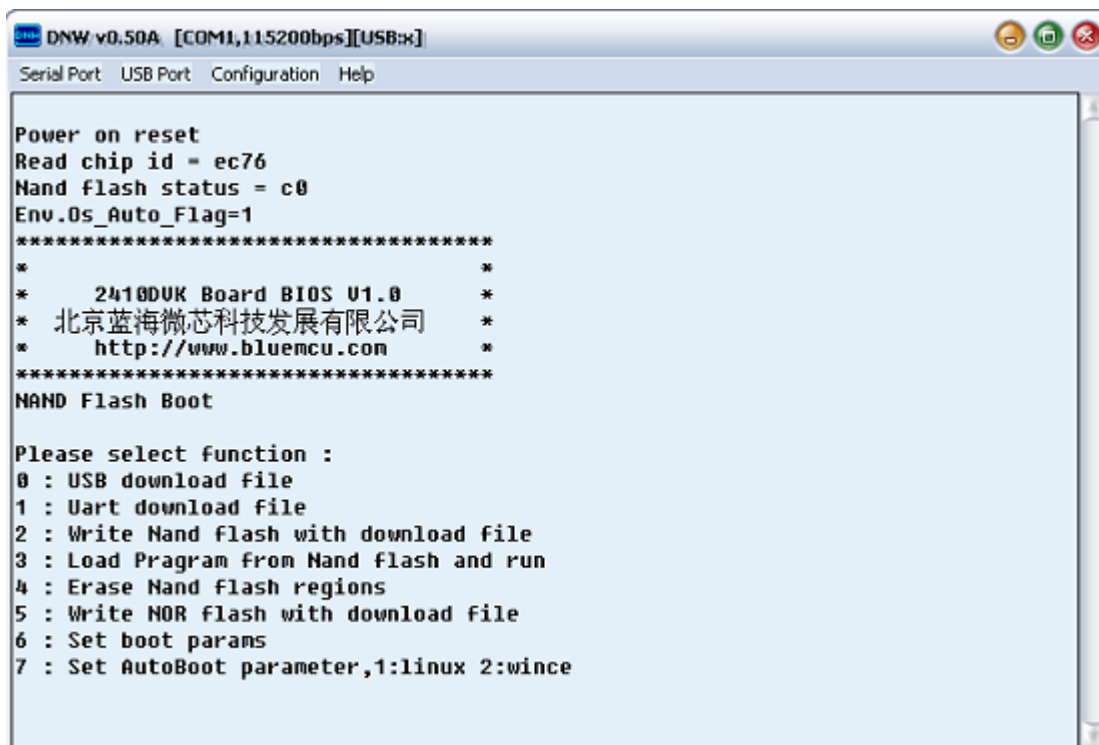
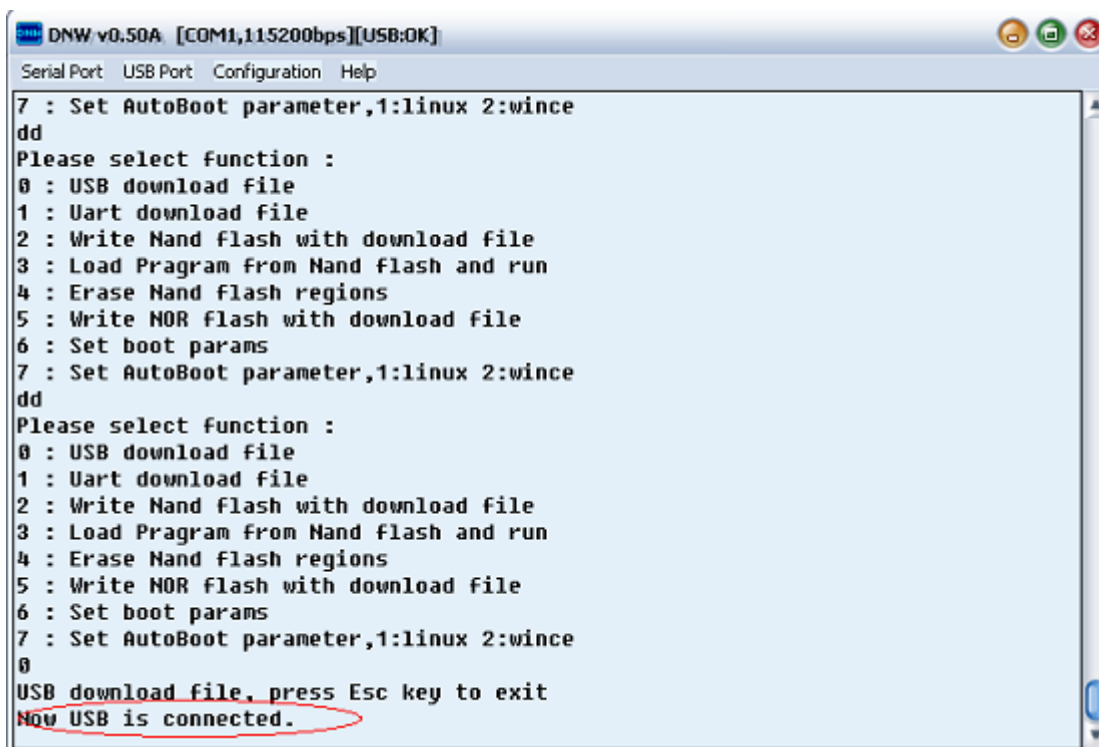


图 2-5

显示开发板的启动信息，我们在终端里输入0，选择利用USB下载文件，然后连接好USB线，系统会提示发现新的USB设备，光盘里提供了USB的驱动，驱动安装完成后提示如下图2-6:



然后我们利用DNW的USB接口传输文件，如下图所示：

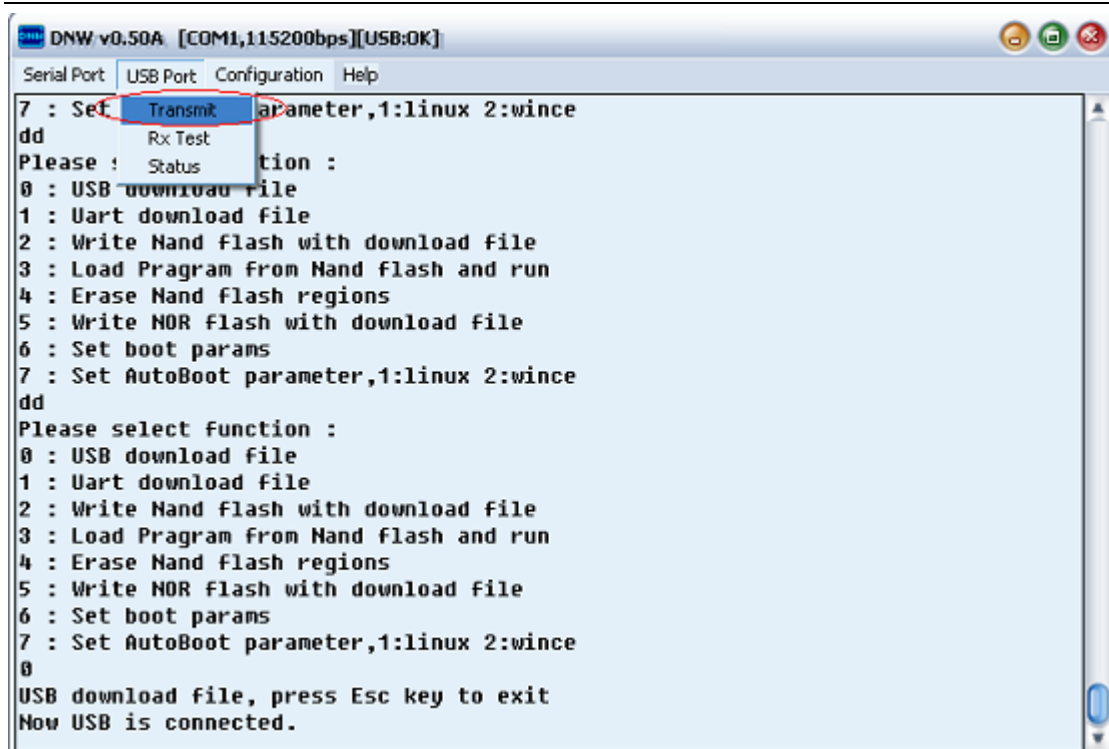


图 2-7

在光盘的目标文件夹下有一个名为First_project.bin的文件，选中它通过USB下载完成以后的状态如下图2-8所示：

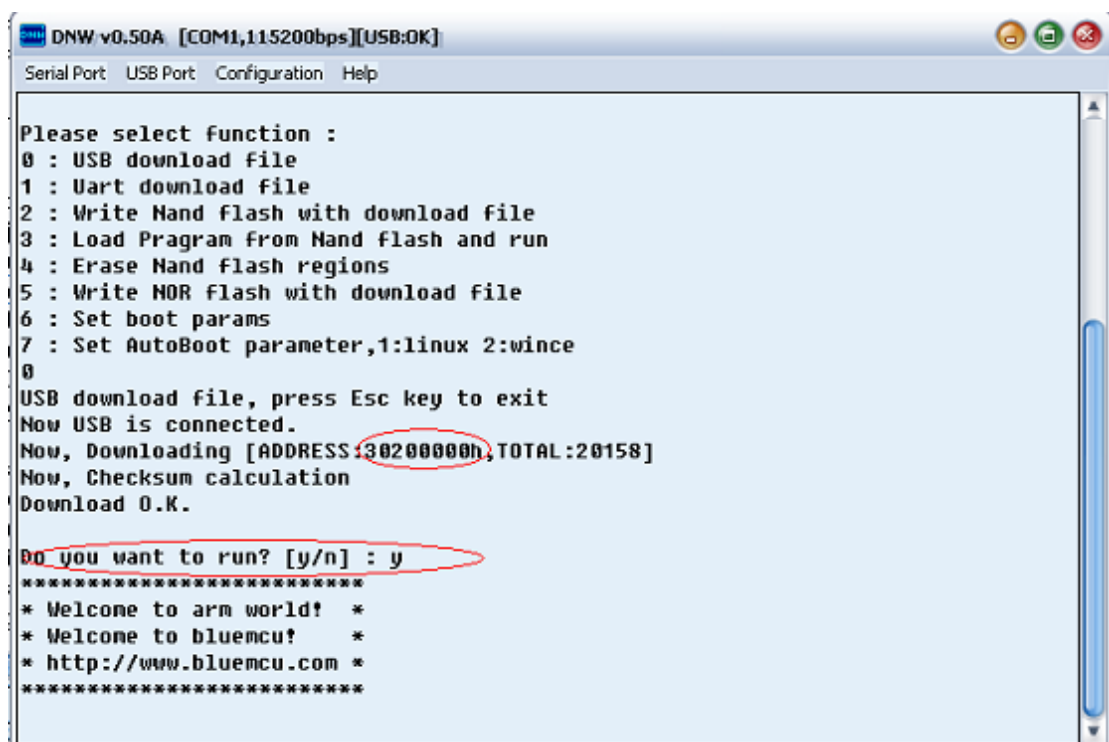


图 2-8

这样我们就将已经利用ADS1.2编译好的二进制程序下载到SDRAM中运行了，运行的效果如图2-8所示。

第三章 s3c2410 体系结构与外设

本章首先介绍一下有关S3C2410的内部资源和体系结构，详细资料请读者参考其它资料。然后介绍了开发板的地址空间分配和跳线设置。

3.1 S3C2410 处理器简介

S3C2410 是韩国三星公司的一款基于ARM920T 内核的16/32 位RISC 嵌入式微处理器，主要面向手持设备以及高性价比，低功耗的应用。运行的频率可以达到203MHz。ARM920T 核由ARM9TDMI，存储管理单元（MMU）和高速缓存三部分组成。其中MMU 可以管理虚拟内存，高速缓存由独立的16KB 地址和16KB 数据高速Cache 组成。ARM920T 有两个协处理器：CP14 和CP15。CP14 用于调试控制，CP15 用于存储系统控制以及测试控制。S3C2410 的资源包括：

- 1个LCD 控制器（支持STN 和TFT 带有触摸屏的液晶显示屏）。
- SDRAM 控制器。
- 3个通道的UART。
- 4个通道的DMA。
- 4个具有PWM 功能的计时器和一个内部时钟。
- 8 通道的10 位ADC。
- 触摸屏接口。
- IIS 总线接口。
- 2个USB 主机接口，1 个USB 设备接口。
- 2个SPI 接口。
- SD 接口和MMC 卡接口。
- 看门狗计数器。
- 117 个通用I/O 口和24 位外部中断源。
- 8 通道10 位AD 控制器。

S3C2410 集成了一个具有日历功能的RTC 和具有PLL（MPLL 和UPLL）的芯片时钟

发生器。MPLL 产生主时钟，能够使处理器工作频率最高达到203MHz。

S3C2410 将系统的存储空间分为8 组（Bank），每组的大小是128MB，共1GB。Bank0 到 Bank5 的开始地址是固定的，用于ROM 或SRAM。Bank6 和Bank7 用于ROM，SRAM 或SDRAM，这两个组可编程且大小相同。Bank7 的开始地址是Bank6 的结束地址，灵活可变。所有内存块的访问周期都可编程。S3C2410 采用Ngcs[7:0]8 个通用片选信号选择这些组。S3C2410 支持从NAND FLASH 启动，NAND FLASH 具有容量大，比NOR Flash 价格低等特点。系统采用NAND Flash 与SDRAM 组合，可以获得非常高的性价比。S3C2410 具有三种启动方式可以通过OM[1：0]管脚进行选择：

OM[1：0]=00 时处理器从NAND Flash 启动；

OM[1：0]=01 时处理器从16 位宽的ROM 启动；

OM[1：0]=10 时处理器从32 位宽的ROM 启动。

用户可以将引导代码和操作系统镜像存放在外部的NAND Flash 中，并从NAND Flash启动。当处理器在这种启动模式下复位时，内置的NAND Flash 将访问控制接口，并将代码自动加载到内部SRAM（此时该SRAM 定位于起始地址空间0x00000000,容量为4KB）并且运行。之后，SRAM 中的引导程序将操作系统镜像加载到SDRAM 中，操作系统就能够在SDRAM 中运行。启动完毕后，4KB 的启动SRAM 就可以用于其它用途。如果从其它方式启动，启动ROM 就要定位于内存的起始地址空间0x00000000,处理器直接在ROM 上运行启动程序，而4KB 启动SRAM被定位于内存地址的0x40000000 处。S3C2410 对于片内的各个部件采用了独立的电源供给方式：

内核采用1.8V 供电；

存储单元采用3.3V 独立供电，对于一般SDRAM 可以采用3.3V，对于移动SDRAM 可以采用Vdd 等于1.8/2.5V；□ Vddq 等于3.0/3.3V；I/O 采用独立3.3V 供电。

3.2 地址空间分配以及跳线设置

S3C2410 支持两种启动模式：

一种是从NAND FLASH 启动；一种是从外部nGCS0 片选的Nor Flash 启动。

在这两种启动模式下，各片选的存储空间分配是不同的，这两种启动模式的存储分配图

如下：

0xFFFF_FFFF	Not Used	Not Used
0x6000_0000	SFR Area	SFR Area
0x4800_0000		
0x4000_0FFF	BootSRAM (4KBytes)	Not Used
0x4000_0000	SDRAM (BANK7, nGCS7)	SDRAM (BANK7, nGCS7)
0x3800_0000	SDRAM (BANK6, nGCS6)	SDRAM (BANK6, nGCS6)
0x3000_0000	SROM (BANK5, nGCS5)	SROM (BANK5, nGCS5)
0x2800_0000	SROM (BANK4, nGCS4)	SROM (BANK4, nGCS4)
0x2000_0000	SROM (BANK3, nGCS3)	SROM (BANK3, nGCS3)
0x1800_0000	SROM (BANK2, nGCS2)	SROM (BANK2, nGCS2)
0x1000_0000	SROM (BANK1, nGCS1)	SROM (BANK1, nGCS1)
0x0800_0000	SROM (BANK0, nGCS0)	BootSRAM (4KBytes)
0x0000_0000		
OM[1:0] == 01, 10		OM[1:0] == 00
a) Not using NAND flash for booting ROM		a) Using NAND flash for booting ROM

图 3-1

SEL401 接AB，从NAND FLASH 启动。

接BC，从NOR FLASH 启动。

a)图是nGCS0 片选的Nor Flash 启动模式下的存储分配图；

b)图是NAND FLASH 启动模式下的存储分配图；

说明：SFR Area 为特殊寄存器地址空间。

S3C2410开发板跳线分配表：

表3-1

跳线名SEL401	功能实现	功能实现
1	AB:NANDFLASH启动	BC:Norflash启动
2	AB: PS2键盘有效	BC:串口1有效
3	AB: PS2键盘有效	BC:串口1有效

4	AB：红外线有效	BC:串口2有效
5	AB:红外线有效	BC：串口2有效
6	AB:USB Slave有效	BC: USB host有效
7	AB:USB slave有效	BC: USB host有效

第四章 ADS1.2 集成开发环境

ADS(ARM Developer Suite)是 ARM 公司推出的功能强大的 ARM 开发集成开发工具，目前最流行的版本是 1.2 版本。

4.1 建立工程

建立一个新的工程步骤如下：

单击 File 下的 New 后如图 4-1 所示：该对话框有 3 个选项卡，即 Project、File、Object。

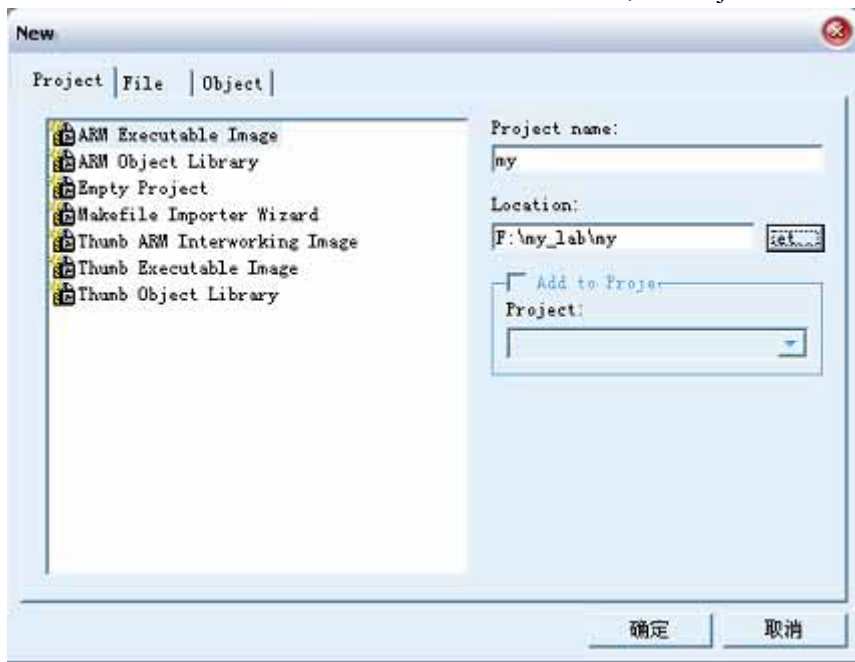


图 4-1

ARM Executable Image：用于由 ARM 指令的代码生成一个 ELF 格式的可执行映像文件。其它的工程格式读者可以参考其它的资料。这里我们选择 ARM Executable Image ,在“ Project name ”中输入工程文件的名字，例如 my，点击 Setting 按钮，浏览工程保存的路径，设置好以后点击确定。（详细设置方法可以参考多媒体演示文档）

点击确定后会出现 my.mcp 的窗口，如图 5-2 所示

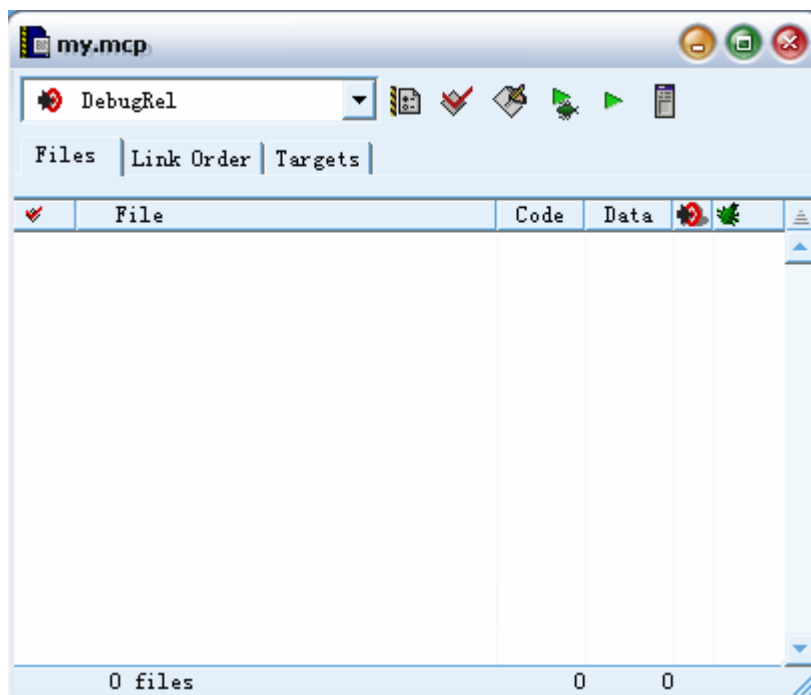


图 4-2

在窗口中点击右键选择“Add File ...”可以把源程序添加到工程中。我们就可以编写自己的程序了，大家可以参考多媒体演示。

这里我们打开一个已经编写好的工程，就是上面我们曾经试验过的 First_project.bin 的源代码。我们来讨论如何将它编译成二进制文件，并在开发板上运行。打开工程 First_project.mcp 如图 4-3 所示：

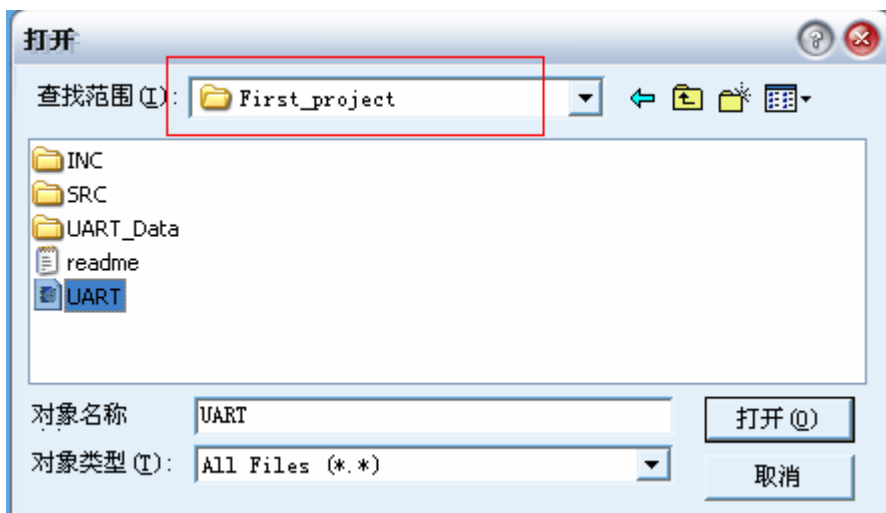


图 4-3

工程的所有文件已经添加到工程中我们选择图 4-4 种所示的选项，将所有源文件选中，准备

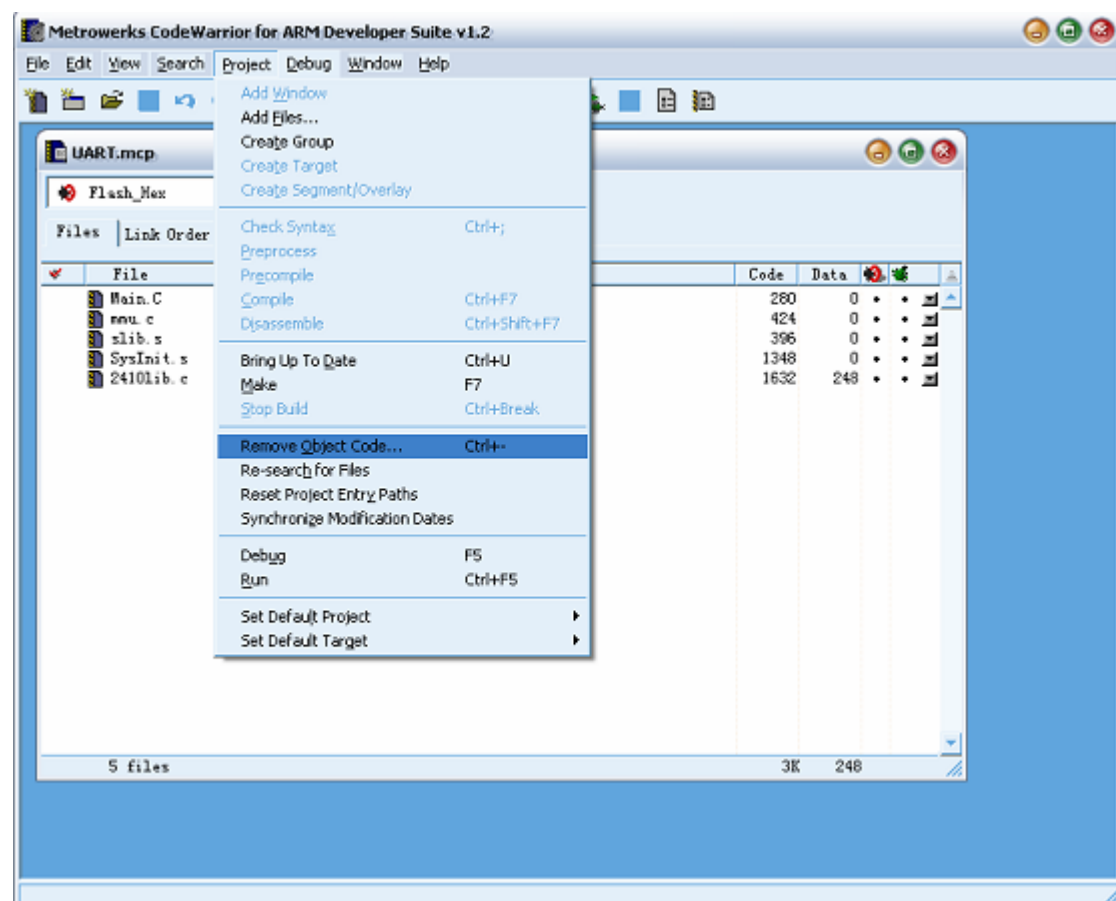


图 4-4

编译,在编译之前我还要有一些设置需要读者注意:下面我们简单讲述一下如何生成目标文件的配置。首先讲述一下如何进行生成目标的配置。点击 Edit 菜单,选择“ DebugRel Settings ...”(注意,这个选项会因用户选择的不同目标而有所不同)。如图 4-5 所示:

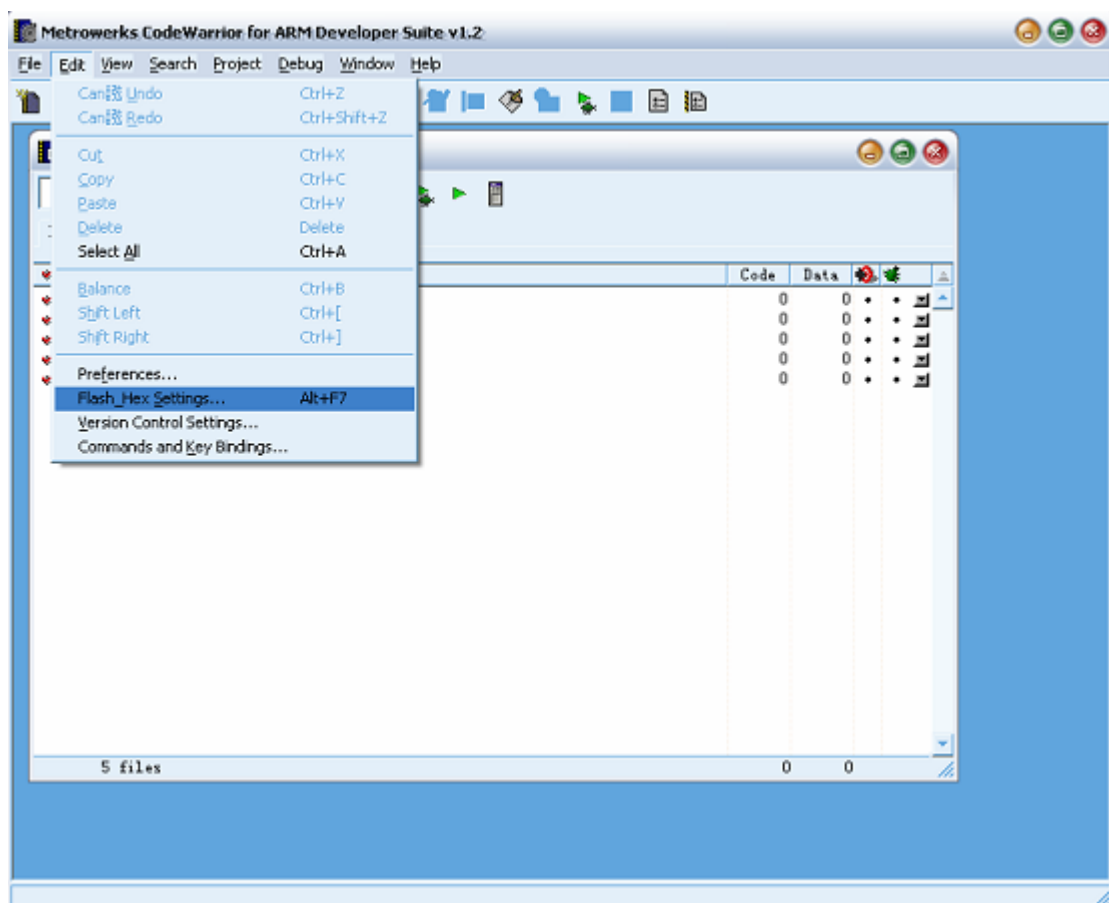


图 4-5

弹出如下图 4-6 所示的对话框：

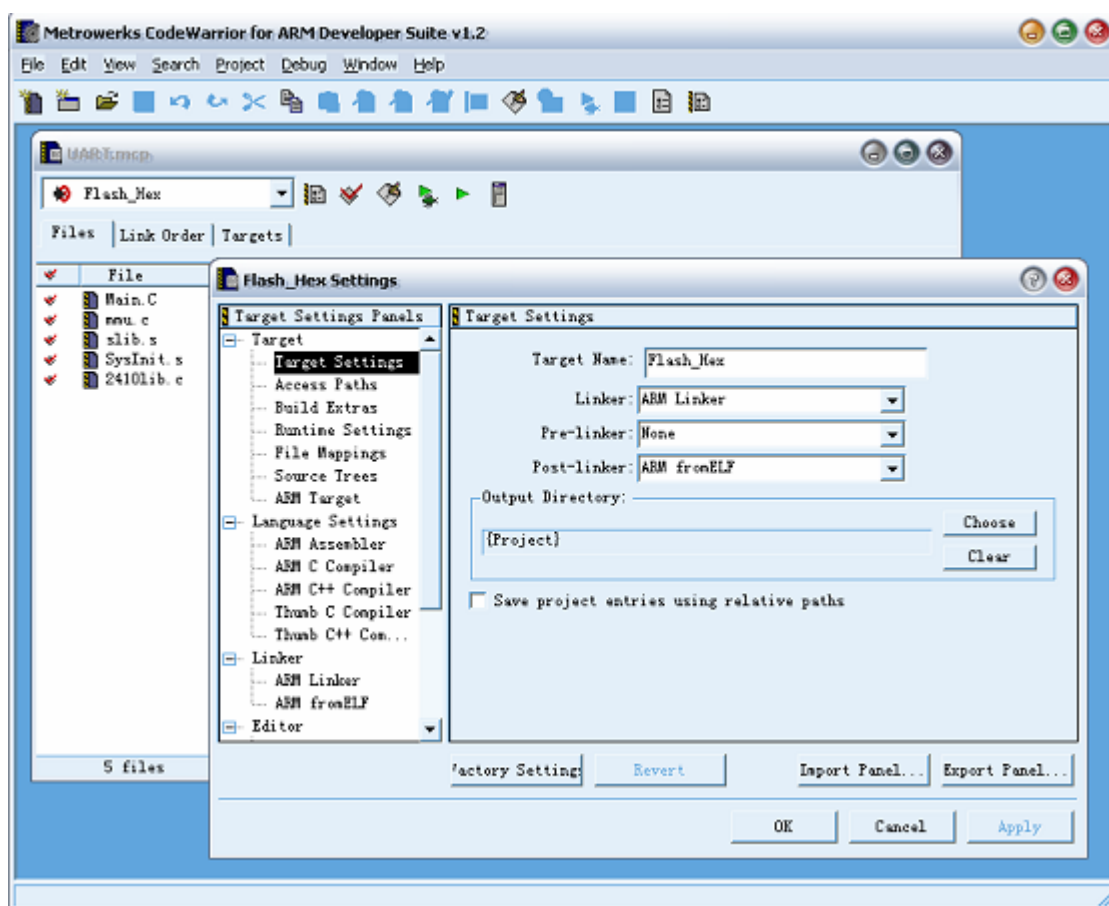


图 4-6

Target设置选项：Target Settings选项里的显示了当前工程的一些基本设置。target 设置选项。Target Name文本框显示了当前的目标设置。Linker 选项供用户选择要使用的链接器。在这里默认选择的是ARM Linker，使用该链接器，将使用armlink 链接编译器和汇编器生成的工程中的文件相应的目标文件。这个设置中还有两个可选项，None 不是不用任何链接器，如果使用它，工程中的所有文件都不会被编译器或汇编器处理。ARM Librarian 表示将编译或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CodeWarrior IDE 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，所以在这里选择ARM fromELF，表示在链接生成映像文件后，再调用FromELF 命令将含有调试信息的ELF 格式的映像文件转换成其它格式的文件。

Language Settings

鼠标点中Language Settings选项出现图4-7所示的界面

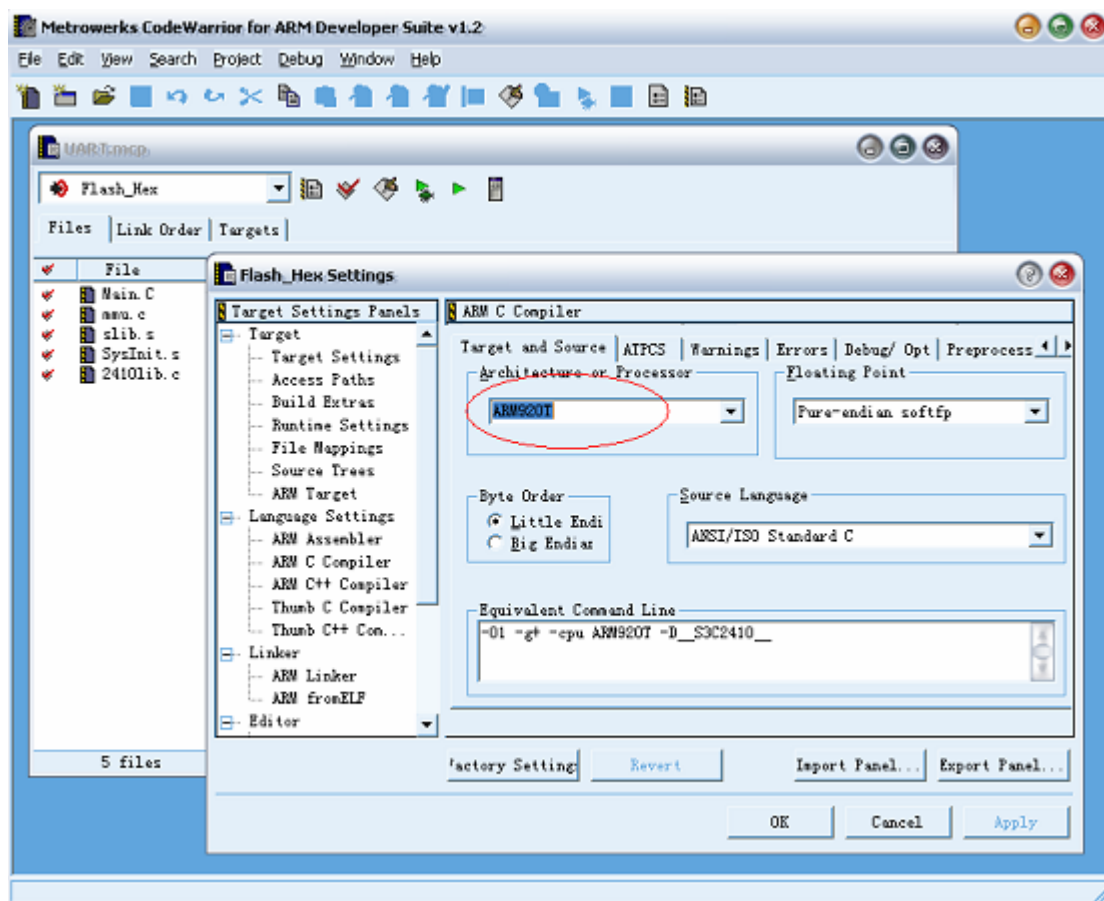


图 4-7

因为本例中包含有汇编源代码，所以要用到汇编器。ADS集成开发环境的汇编器是 arm asm，

默认的ARM 体系结构是ARM920T。字节顺序默认就是小端模式。其它设置，就用默认值即可。

还有一个需要注意的就是ARM C 编译器，它实际就是调用的命令行工具armcc。使用默认的设置就可以了。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，这行文字就显示的是相应的编译或链接选项，由于有了CodeWarrior，开发人员可以不用再去查看繁多的命令行选项，只要在界面中选中或撤消某个选项，软件就会自动生成相应的代码，为不习惯在DOS 下键入命令行的用户提供了极大的方便。

Linker 设置

鼠标选中ARM Linker，出现如图4-8所示对话框。这里详细介绍该对话框的主要的标签页选项，因为这些选项对最终生成的文件有着直接的影响。

在标签页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，使用的是链接器选项中指定的地址映像方式。Scattered 方式使得链接器要根据scatter格式文件中指定的地址映像，生成复杂的ELF 格式的映像文件。这个选项一般情况下，使用不太多。

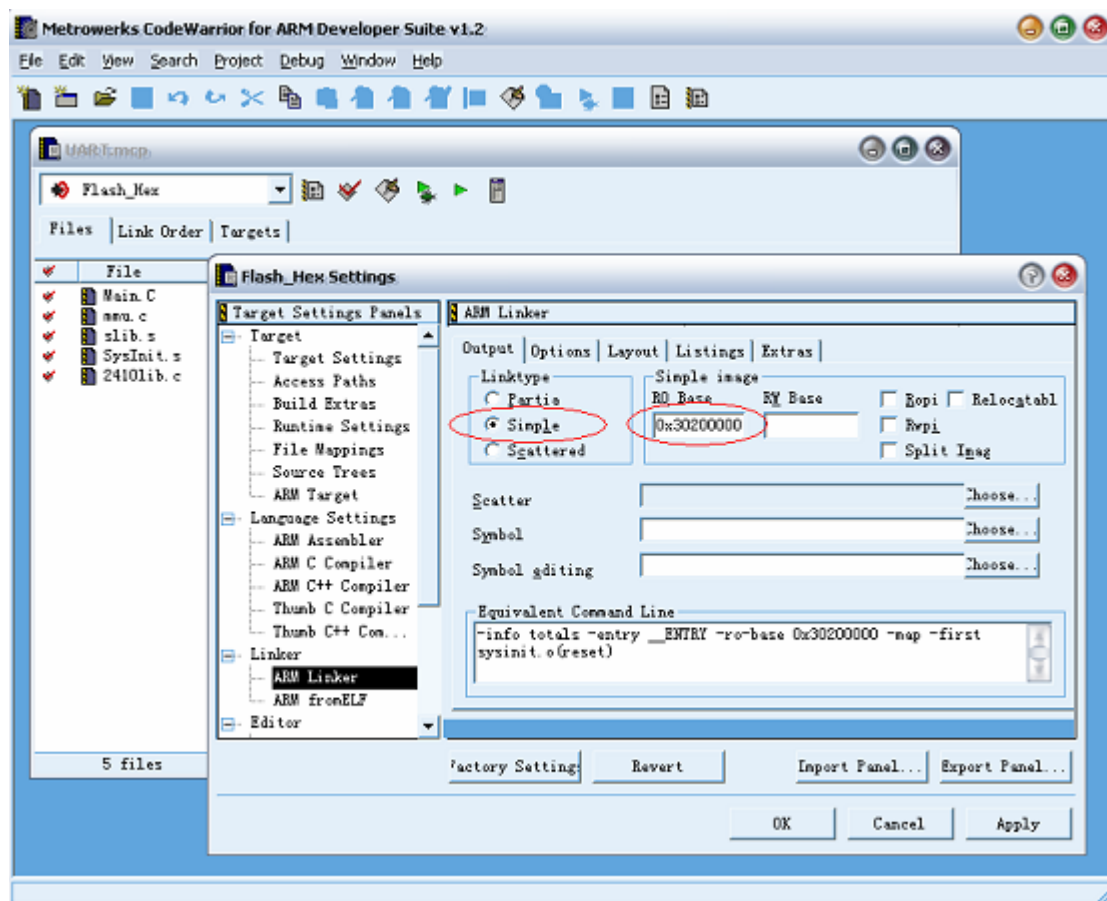


图 4-8

在Linker 下还有一个ARM fromELF，如图4-9所示：

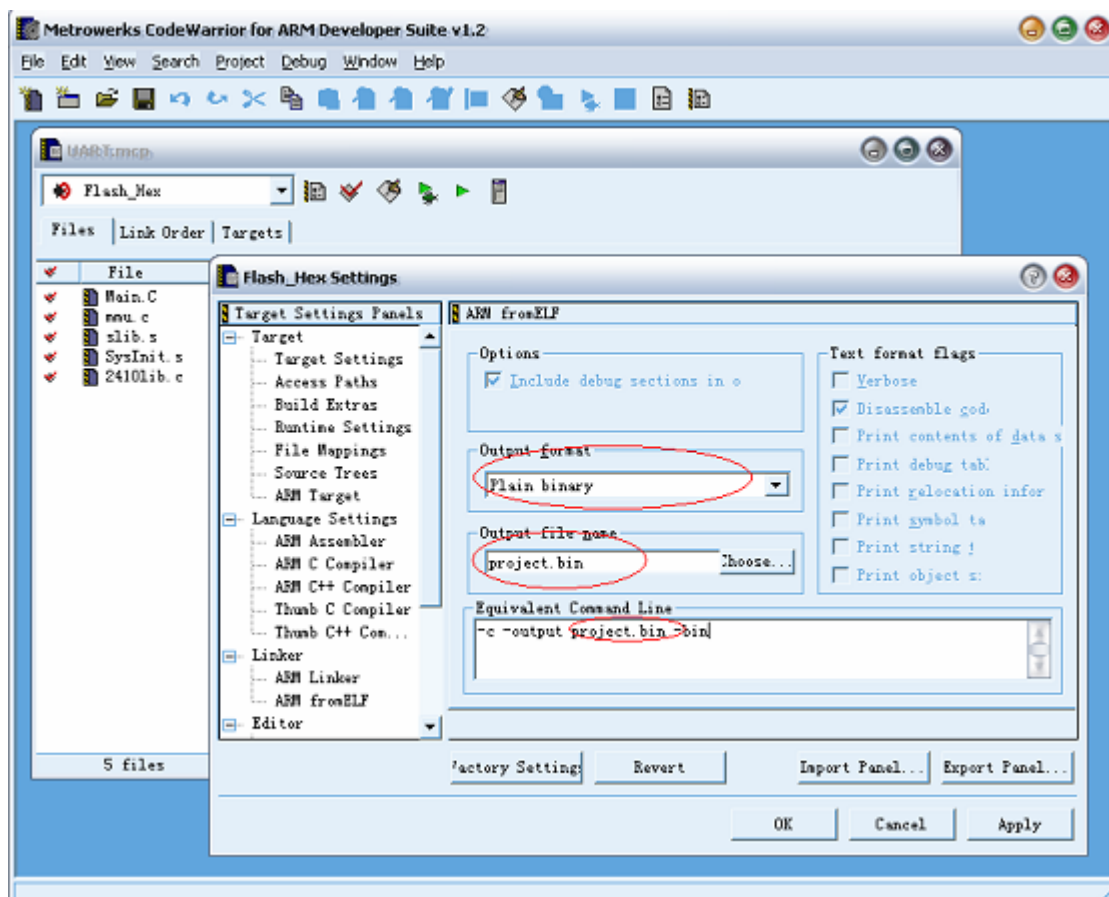


图 4-9

fromELF 是一个实用工具，它实现将链接器，编译器或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，可以被烧些的目标板的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。

我们完成后就可以点击 Make 来完成编译，如图 4-10 所示：生成我们的二进制文件，这样就和前面的通过 USB 下载二进制文件到 SDRSAM 运行联起来了。

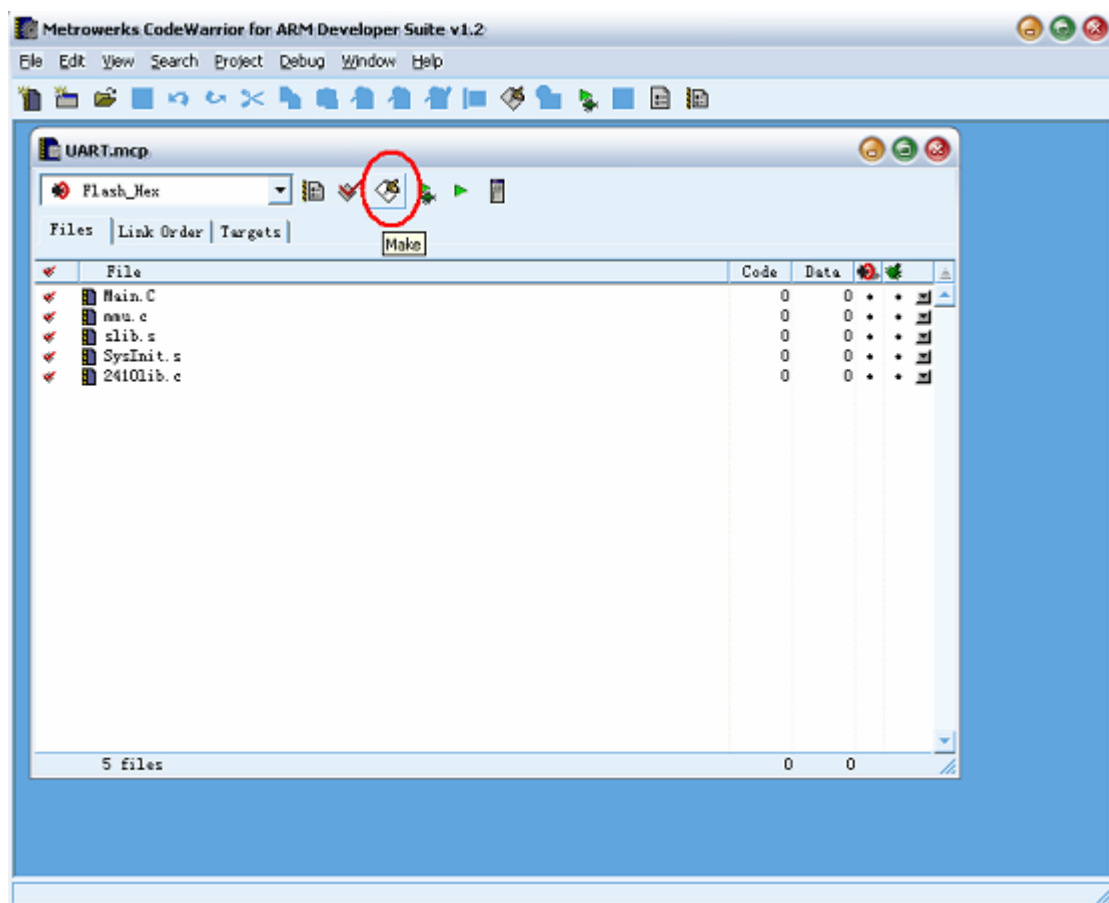


图 4-10

接下来我们就可以下载运行我们自己的程序啦！（上面的设置若是读者还是不清楚，可以参考我们的视频教程关于 ADS1.2 调试的部分）

第二部分 基于操作系统的软件开发

本部分主要是介绍基于 Linux 和 Wince 两种系统的软件开发，以我们的目标板为基础，使读者了解嵌入式系统的开发过程。

第五章 Linux 交叉编译环境的建立和内核编译

5.1 建立嵌入式 linux 开发环境

这一章的工作是在PC 机上的LINUX 操作系统平台下进行的。

工作环境：redhat 9.0 (linux 操作系统)，可以运行在虚拟主机上（详细的方法参考嵌入式的开发过程）

5.1.1. 安装编译工具

编译嵌入式linux 内核前，要先安装交叉编译工具toolchain，随板光盘已附带两个编译工具。一个是cross-2.95.3.tar.bz2，安装此编译器只需要在/usr/local 目录下建一个arm的目录，先将光盘目录“Linux 源码和工具”下“toolchain”文件夹下的cross-2.95.3.tar.bz2 拷贝到某个目录下，然后进入该目录，执行解包命令：

```
tar zxvf cross-2.95.3.tar.bz2
```

即可，之后可编辑/etc/profile 文件，修改系统当前的环境变量，在最后增加路径：

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

以后编译内核或其他应用程序均可用arm-linux- 来指定交叉编译器。

5.1.2. 编译内核

1) 解压内核包

先将光盘目录的“Linux 源码和工具”的Linux 内核包拷贝到某个目录下，进入这个目录，然后解压这个Linux 源码包，命令为：

```
tar jxvf LJD2410DVK.tar.bz2
```

解压后会生成一个名为 LJD2410DVK 的目录

2) 编辑 Makefile 文件

进入解压的目录后，运行命令：

vi Makefile 如下图5-3所示

```

mengyang@localhost/2.4.18-rmk7
文件(F) 编辑(E) 查看(V) 终端(T) 转到(G) 帮助(H)
Your kernel configuration changes were NOT saved.

make: warning: Clock skew detected. Your build may be incomplete.
[root@localhost 2.4.18-rmk7]# ls
arch          fs            kernel_2410.cfg      mm             Rules.make
COPYING       include       kernel_2410.cfg.old  net            scripts
CREDITS       init          lib                  README         user
Documentation ipc           MAINTAINERS          REPORTING-BUGS zImage
drivers        kernel        Makefile             rm

[root@localhost 2.4.18-rmk7]# cd ..
[root@localhost /]# ls
2.4.18-rmk7    boot          home              lost+found      opt             sbin            usr
2.4.18-rmk7.tar.bz2 dev            initrd            misc             proc            tftpboot        var
bin            etc            lib               mnt              root            tmp

[root@localhost /]# cd 2.4.18-rmk7
[root@localhost 2.4.18-rmk7]# ls
arch          fs            kernel_2410.cfg      mm             Rules.make
COPYING       include       kernel_2410.cfg.old  net            scripts
CREDITS       init          lib                  README         user
Documentation ipc           MAINTAINERS          REPORTING-BUGS zImage
drivers        kernel        Makefile             rm

[root@localhost 2.4.18-rmk7]# vi M
MAINTAINERS Makefile
[root@localhost 2.4.18-rmk7]# vi Makefile
    
```

图 5-3

找到 `CROSS_COMPILE = /work/cgf/2410/2.95.3/bin/arm-linux-` 这行，将它修改为 `CROSS_COMPILE = arm-linux-` 如图5-4所示：

```

mengyang@localhost/2.4.18-rmk7
文件(F) 编辑(E) 查看(V) 终端(T) 转到(G) 帮助(H)

PATCHLEVEL = 4
SUBLEVEL = 18
EXTRAVERSION = -rmk7-pxa1

KERNELRELEASE=$(VERSION).$(PATCHLEVEL).$(SUBLEVEL)$(EXTRAVERSION)

#ARCH := $(shell uname -m | sed -e s/i.86/i386/ -e s/sun4u/sparc64/ -e s/arm.*/a
rm/ -e s/sa110/arm)
ARCH := arm
KERNELPATH=kernel-$(shell echo $(KERNELRELEASE) | sed -e "s/-//g")

CONFIG_SHELL := $(shell if [ -x "$$BASH" ]; then echo $$BASH; \
    else if [ -x /bin/bash ]; then echo /bin/bash; \
    else echo sh; fi ; fi)
TOPDIR := $(shell /bin/pwd)

HPATH      = $(TOPDIR)/include
FINDHPATH  = $(HPATH)/asm $(HPATH)/linux $(HPATH)/scsi $(HPATH)/net

HOSTCC     = gcc
HOSTCFLAGS = -Wall -Wstrict-prototypes -O2 -fomit-frame-pointer

CROSS_COMPILE = arm-linux-

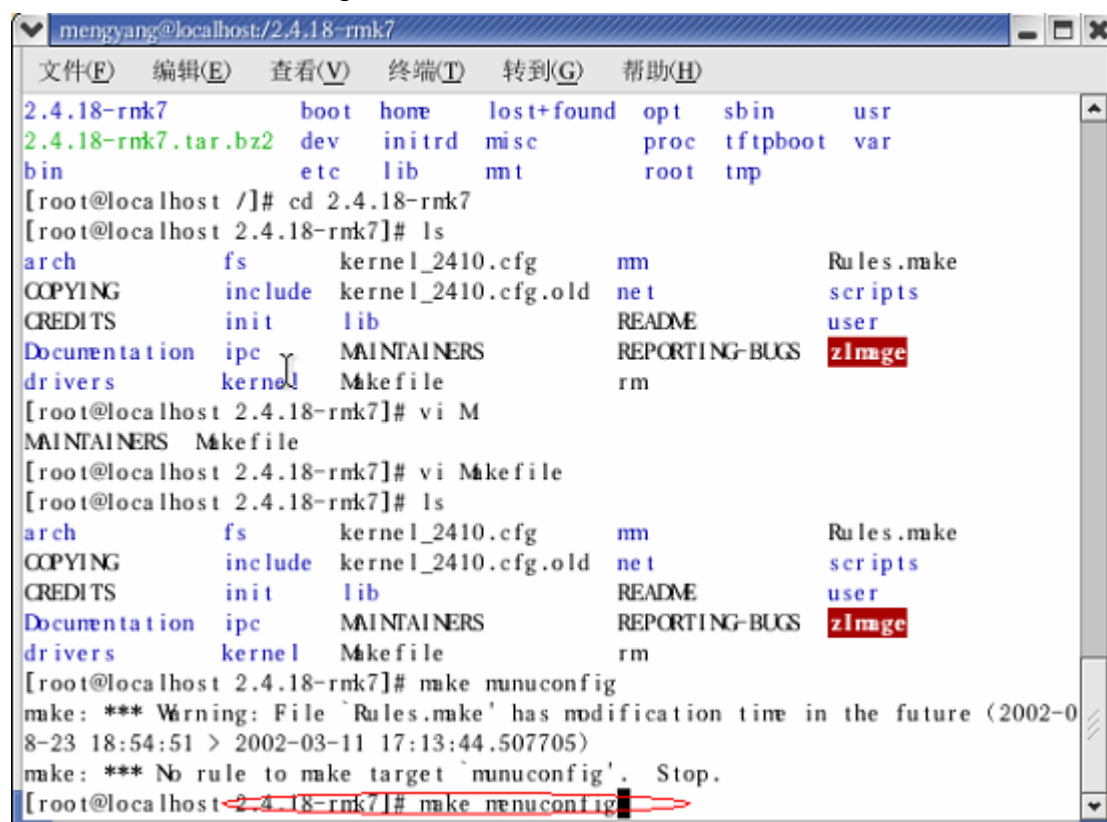
"Makefile" [已转换] 544L, 17380C          23,1          0%
    
```

图 5-4

修改完成后按ESC退出编辑状态，按”shift”+”.”进入命令提示行，然后键入wq！保存修改并且退出vi编辑器。

3) 配置内核

输入命令：make menuconfig



```

mengyang@localhost/2.4.18-rnk7
文件(F) 编辑(E) 查看(V) 终端(T) 转到(G) 帮助(H)
2.4.18-rnk7      boot  home  lost+found  opt  sbin  usr
2.4.18-rnk7.tar.bz2 dev  initrd  misc      proc  tftpboot  var
bin             etc   lib    mt        root  tmp
[root@localhost /]# cd 2.4.18-rnk7
[root@localhost 2.4.18-rnk7]# ls
arch          fs          kernel_2410.cfg  mm          Rules.make
COPYING       include    kernel_2410.cfg.old  net        scripts
CREDITS       init       lib             README     user
Documentation  ipc        MAINTAINERS      REPORTING-BUGS  zImage
drivers       kernel     Mkefile          rm
[root@localhost 2.4.18-rnk7]# vi M
MAINTAINERS  Mkefile
[root@localhost 2.4.18-rnk7]# vi Mkefile
[root@localhost 2.4.18-rnk7]# ls
arch          fs          kernel_2410.cfg  mm          Rules.make
COPYING       include    kernel_2410.cfg.old  net        scripts
CREDITS       init       lib             README     user
Documentation  ipc        MAINTAINERS      REPORTING-BUGS  zImage
drivers       kernel     Mkefile          rm
[root@localhost 2.4.18-rnk7]# make mnuconfig
make: *** Warning: File `Rules.make' has modification time in the future (2002-0
8-23 18:54:51 > 2002-03-11 17:13:44.507705)
make: *** No rule to make target `mnuconfig'. Stop.
[root@localhost 2.4.18-rnk7]# make menuconfig

```

图 5-5

进入配置栏的的“Load an Alternate Configuration File”如下图5-7所示：

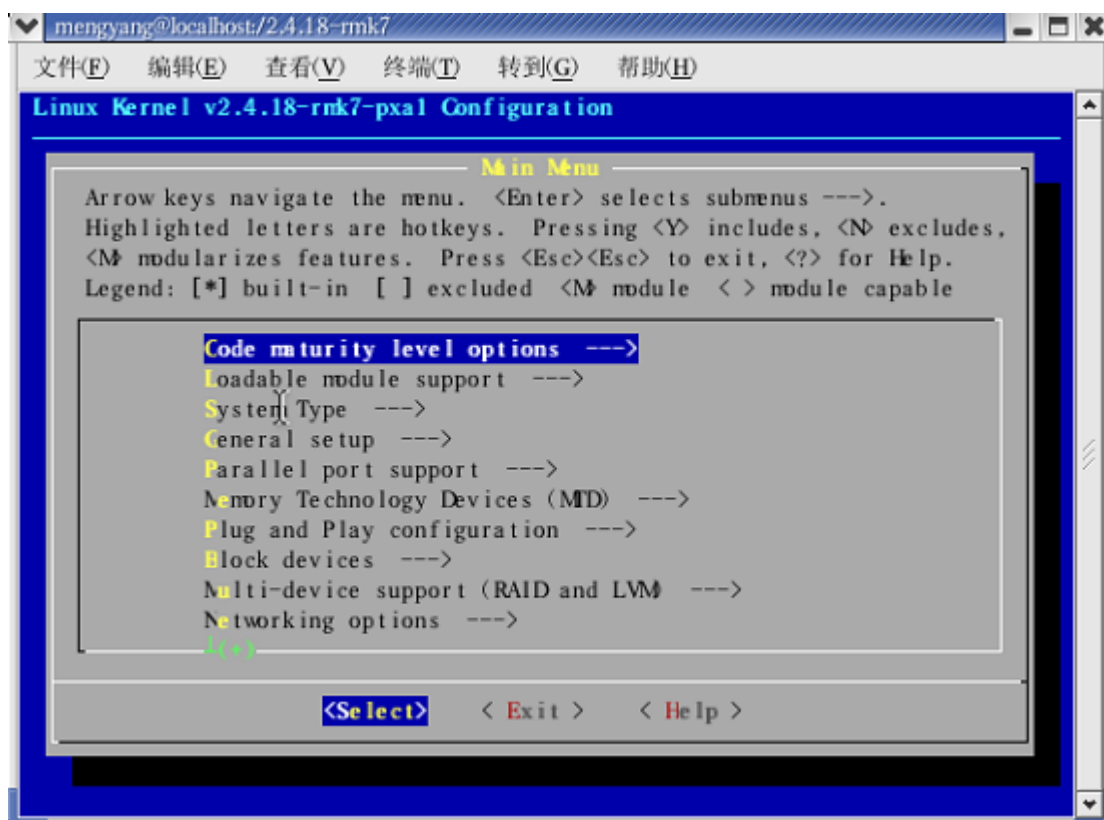


图 5-6

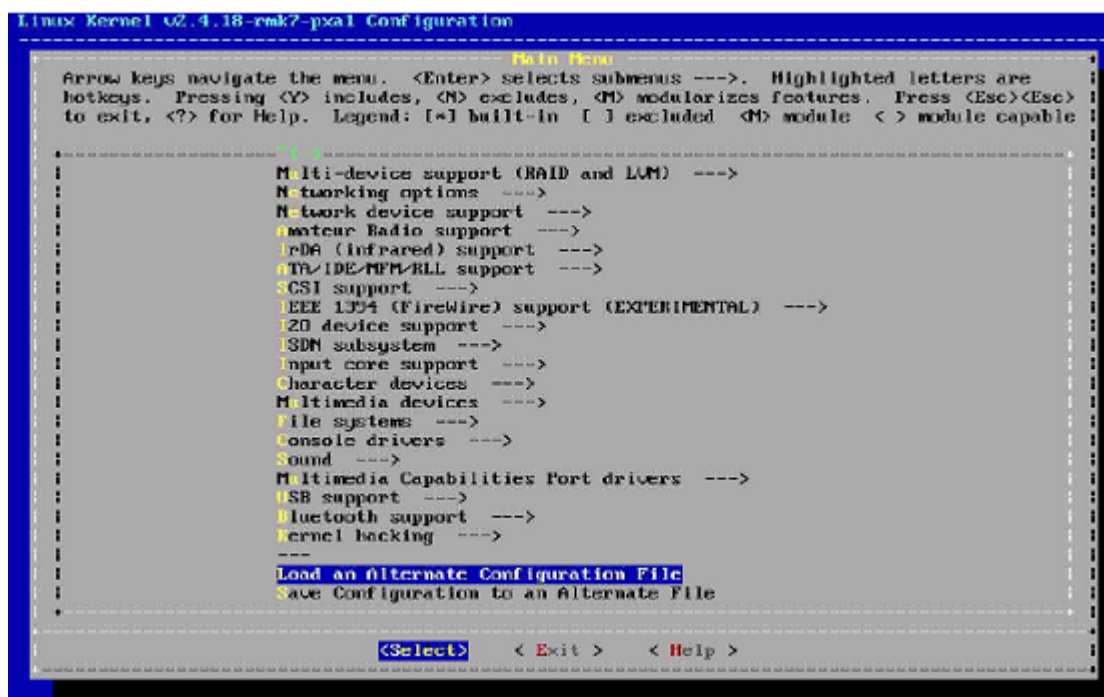


图 5-7

输入配置文件名kernel_2410.cfg 并回车,在主菜单里选择Exit 退出并保存设置。如图5-8 所示:

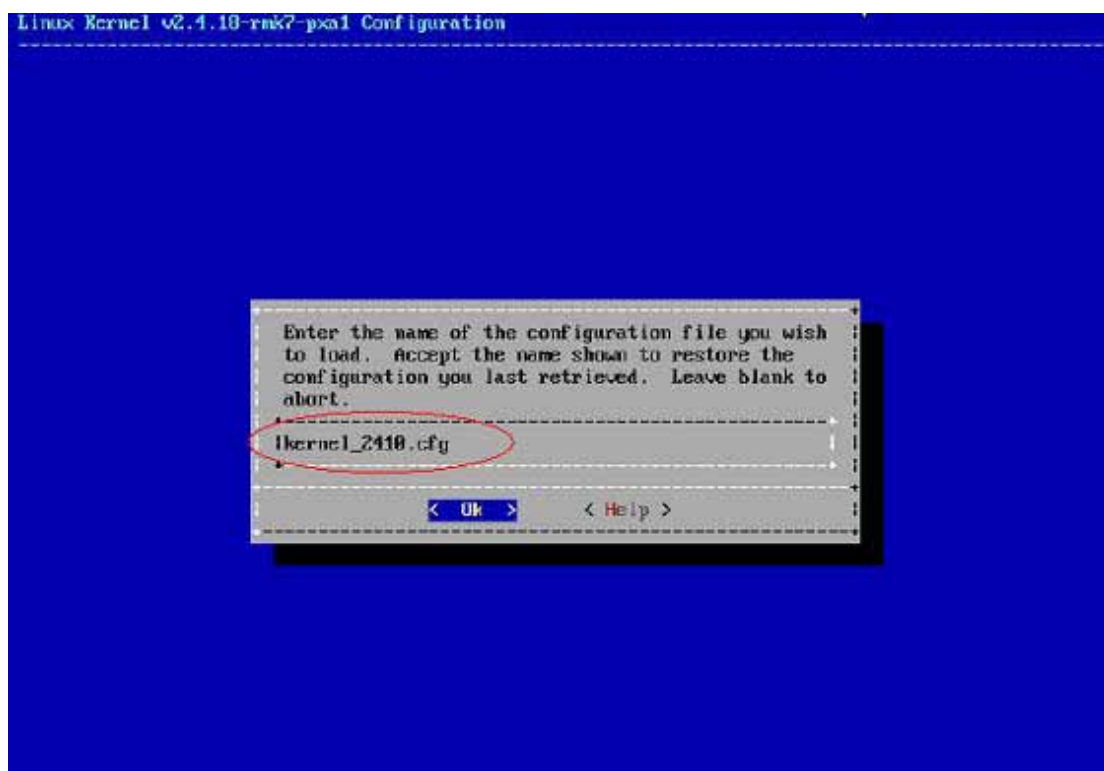


图 5-8

保存后退出。会到命令行执行 make dep 命令建立依赖关系。如图 5-9 所示：

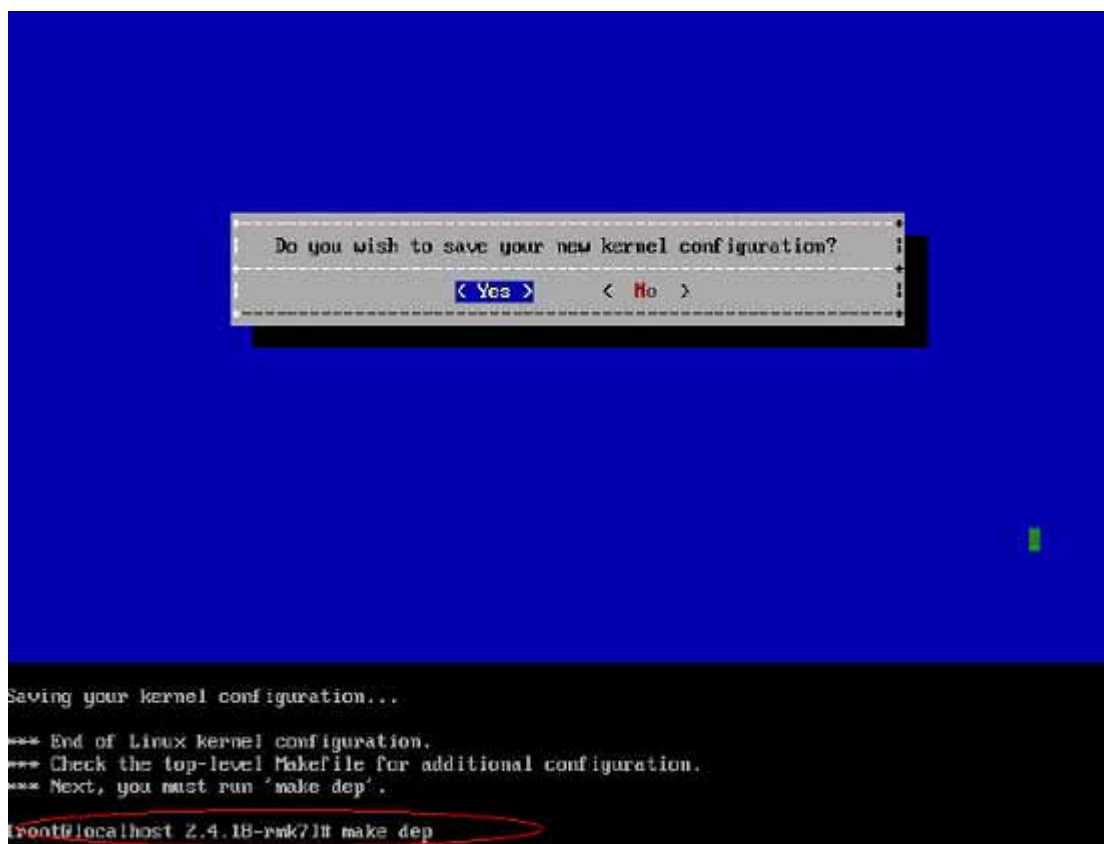


图 5-9

完成后输入 make zImage ,编译的时间比较长 ,根据 PC 机的配置有关大概需要 10 分钟左右。

编译完成后可在arch/arm/boot 目录下得到linux 内核压缩映像zImage。如图5-10所示：

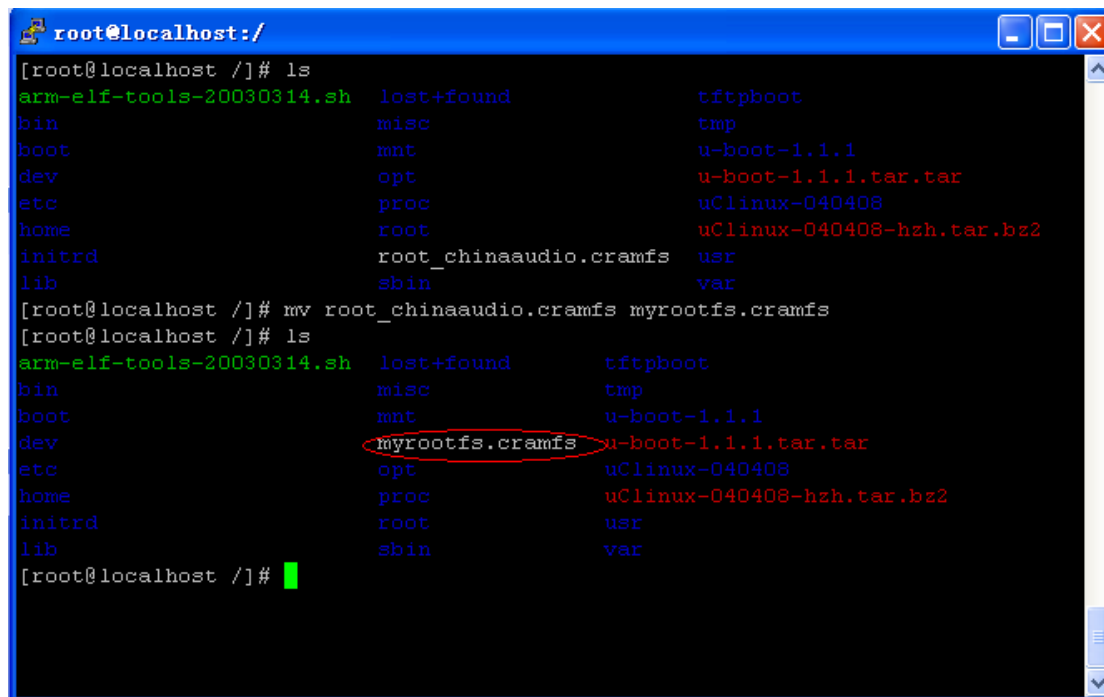
```
make[1]: Leaving directory '/work/2.4.18-rmk7/arch/arm/fastfp'
arm-linux-ld -p -X -T arch/arm/vmlinux.lds arch/arm/kernel/head-armv.o arch/arm/kernel/init_task.o 1
init/main.o init/version.o \
--start-group \
arch/arm/kernel/kernel.o arch/arm/mm/mm.o arch/arm/mach-s3c2410/s3c2410.o kernel/kernel.o mm
mm.o fs/fs.o ipc/ipc.o \
drivers/serial/serial.o drivers/char/char.o drivers/block/block.o drivers/misc/misc.o drive
rs/net/net.o drivers/media/media.o drivers/scsi/scsidrv.o drivers/sound/sounddrivers.o drivers/mtd/m
tdlink.o drivers/net/wireless/wireless_net.o drivers/video/video.o drivers/usb/usbdrv.o drivers/inputs
t/inputdrv.o \
net/network.o \
arch/arm/nvfp/math-emu.o arch/arm/lib/lib.a /work/2.4.18-rmk7/lib/lib.a \
--end-group \
-o vmlinux
arm-linux-mm vmlinux | grep -v '\(compiled\)\|(\.o$)\|(\[aDw]\)\|(\.\.ng$)\|(\.LASHLIB)\)' |
sort > System.map
make[1]: Entering directory '/work/2.4.18-rmk7/arch/arm/boot'
make[2]: Entering directory '/work/2.4.18-rmk7/arch/arm/boot/compressed'
arm-linux-gcc -D__ASSEMBLY__ -D__KERNEL__ -I/work/2.4.18-rmk7/include -mapcs-32 -march=armv4 -mno-fp
u -msoft-float -traditional -c head.S
arm-linux-gcc -D__KERNEL__ -I/work/2.4.18-rmk7/include -O2 -DSTDC_HEADERS -mapcs-32 -march=armv4 -m
ne=arm9tdmi -mshort-load-bytes -msoft-float -fpic -D__KERNEL__ -I/work/2.4.18-rmk7/include -c -o m
isc.o misc.c
arm-linux-gcc -D__ASSEMBLY__ -D__KERNEL__ -I/work/2.4.18-rmk7/include -mapcs-32 -march=armv4 -mno-fp
u -msoft-float -c -o head-s3c2410.o head-s3c2410.S
arm-linux-objcopy -O binary -R .note -R .comment -S /work/2.4.18-rmk7/vmlinux piggy
gzip -9 < piggy > piggy.gz
arm-linux-ld -r -o piggy.o -b binary piggy.gz
rm -f piggy piggy.gz
arm-linux-ld -p -X -T vmlinux.lds head.o misc.o head-s3c2410.o piggy.o /usr/local/arm/2.95.3/lib/gcc
-2.95.3/lib/arm-linux/2.95.3/libgcc.a -o vmlinux
make[2]: Leaving directory '/work/2.4.18-rmk7/arch/arm/boot/compressed'
arm-linux-objcopy -O binary -R .note -R .comment -S compressed/vmlinux zImage
make[1]: Leaving directory '/work/2.4.18-rmk7/arch/arm/boot'
root@localhost 2.4.18-rmk714 ls arch/arm/boot/zImage -l
-rwxr-xr-x 1 root root 817532 May 6 21:45 arch/arm/boot/zImage
root@localhost 2.4.18-rmk71#
```

图 5-10

5.2 文件系统的制作

上面的过程主要用于调试，掉电之后不能保存，要将程序固化，需要将该程序和模块添加到根文件系统中。这里我们以向LJD-2410DVK-I 光盘附带的myrootfs.cramfs 根文件系统的添加为例。以下的操作是redhat9下进行的，环境请读者自行配置。

(1) 将myrootfs.cramfs拷贝到任意目录下如图5-11所示：



```

root@localhost: /
[ root@localhost / ]# ls
arm-elf-tools-20030314.sh  lost+found  tftpboot
bin                       misc        tmp
boot                     mnt         u-boot-1.1.1
dev                     opt         u-boot-1.1.1.tar.tar
etc                     proc        uClinux-040408
home                   root        uClinux-040408-hzh.tar.bz2
initrd                root_chinaaudio.cramfs  usr
lib                   sbin        var
[ root@localhost / ]# mv root_chinaaudio.cramfs myrootfs.cramfs
[ root@localhost / ]# ls
arm-elf-tools-20030314.sh  lost+found  tftpboot
bin                       misc        tmp
boot                     mnt         u-boot-1.1.1
dev                     myrootfs.cramfs  u-boot-1.1.1.tar.tar
etc                     opt         uClinux-040408
home                   proc        uClinux-040408-hzh.tar.bz2
initrd                root        usr
lib                   sbin        var
[ root@localhost / ]#

```

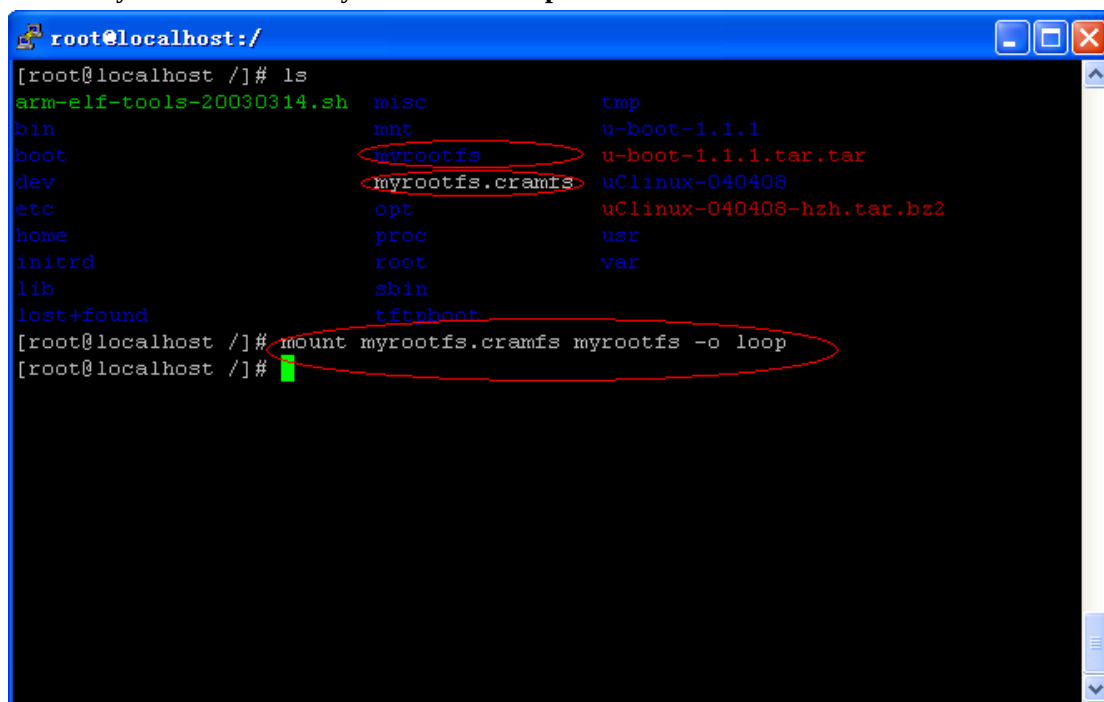
图 5-11

(2) 在该目录下建立一个目录：

mkdir myrootfs

(3) 将myrootfs.cramfs 挂接到myrootfs 目录，如图5-12所示：

mount myrootfs.cramfs myrootfs -o loop



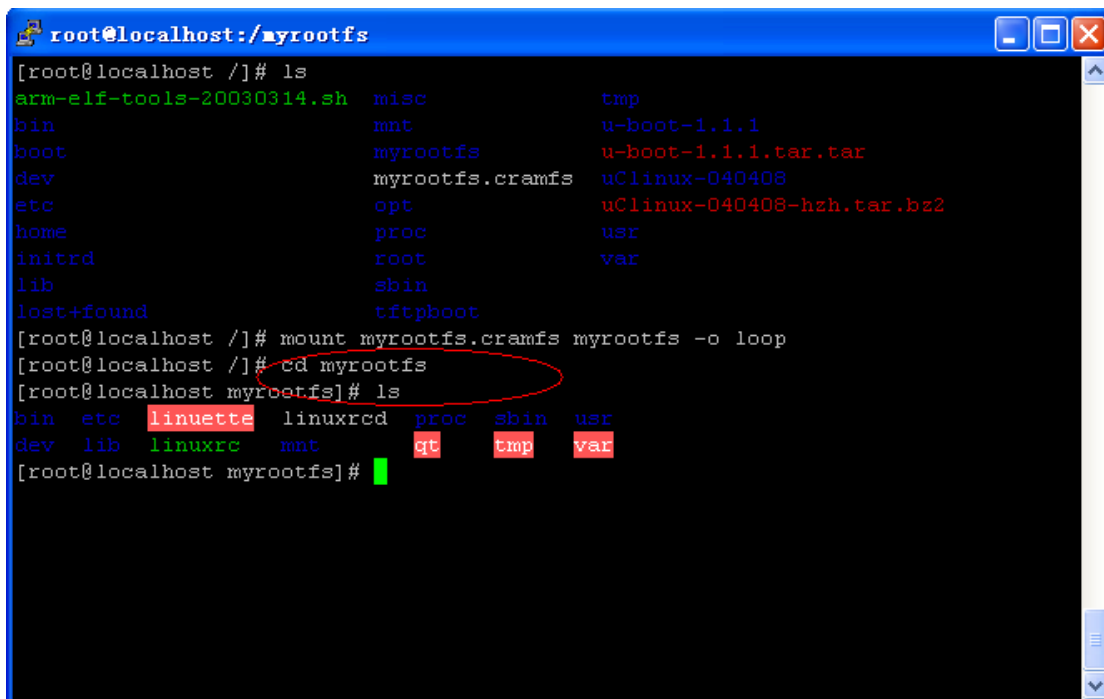
```

root@localhost: /
[ root@localhost / ]# ls
arm-elf-tools-20030314.sh  misc        tmp
bin                       mnt         u-boot-1.1.1
boot                     myrootfs    u-boot-1.1.1.tar.tar
dev                     myrootfs.cramfs  uClinux-040408
etc                     opt         uClinux-040408-hzh.tar.bz2
home                   proc        usr
initrd                root        var
lib                   sbin
lost+found            tftpboot
[ root@localhost / ]# mount myrootfs.cramfs myrootfs -o loop
[ root@localhost / ]#

```

图 5-12

(4) 浏览文件系统里面的内容如图 5-13 所示



```

root@localhost:~/myrootfs
[root@localhost /]# ls
arm-elf-tools-20030314.sh  misc      tmp
bin                       mnt       u-boot-1.1.1
boot                     myrootfs  u-boot-1.1.1.tar.tar
dev                     myrootfs.cramfs  uClinux-040408
etc                     opt       uClinux-040408-hzh.tar.bz2
home                    proc      usr
initrd                  root      var
lib                     sbin
lost+found              tftpboot
[root@localhost /]# mount myrootfs.cramfs myrootfs -o loop
[root@localhost /]# cd myrootfs
[root@localhost myrootfs]# ls
bin  etc  linuette  linuxrc  proc  shbin  usr
dev  lib  linuxrc  mnt      qt     tmp     var
[root@localhost myrootfs]#
    
```

图 5-13

这样我们就可以添加自己的应用程序到我们的文件系统里了。

(5) 制作文件系统

先将mkcramfs (在toolchain目录下)文件拷贝到myrootfs目录所在的目录在这个目录下运行命令：

mkcramfs myrootfs myrootfs.cramfs

运行成功后，会在该目录下生成myrootfs.cramfs 根文件系统。到这里我们自己的文件系统制作完成。

第六章 通过 BIOS 下载 linux 内核和文件文件系统

6.1 烧些内核 (目标代码里的 zImage)

根据第二章的内容配置开发板包括：串口 电源 usb 连接和驱动程序，上电后连接 USB 后如下图所示：

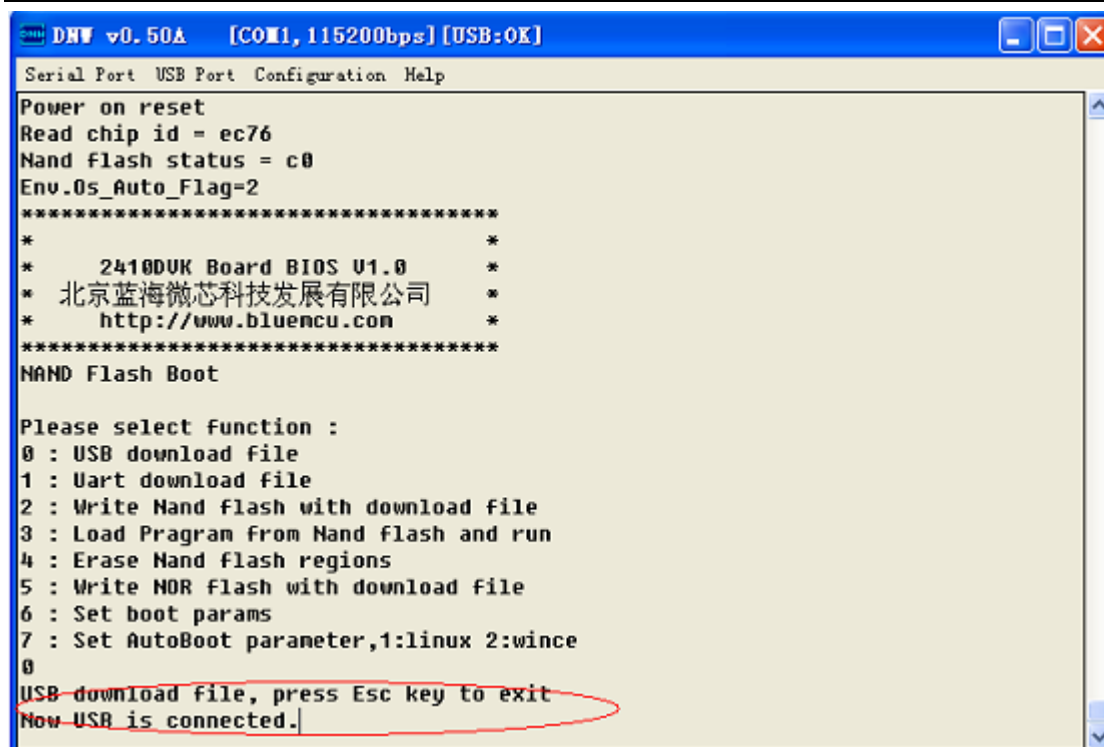


图 5-14

按照第二章的步骤把生成的内核和文件系统下载并烧写到相应的位置.如图 5-15 所示：

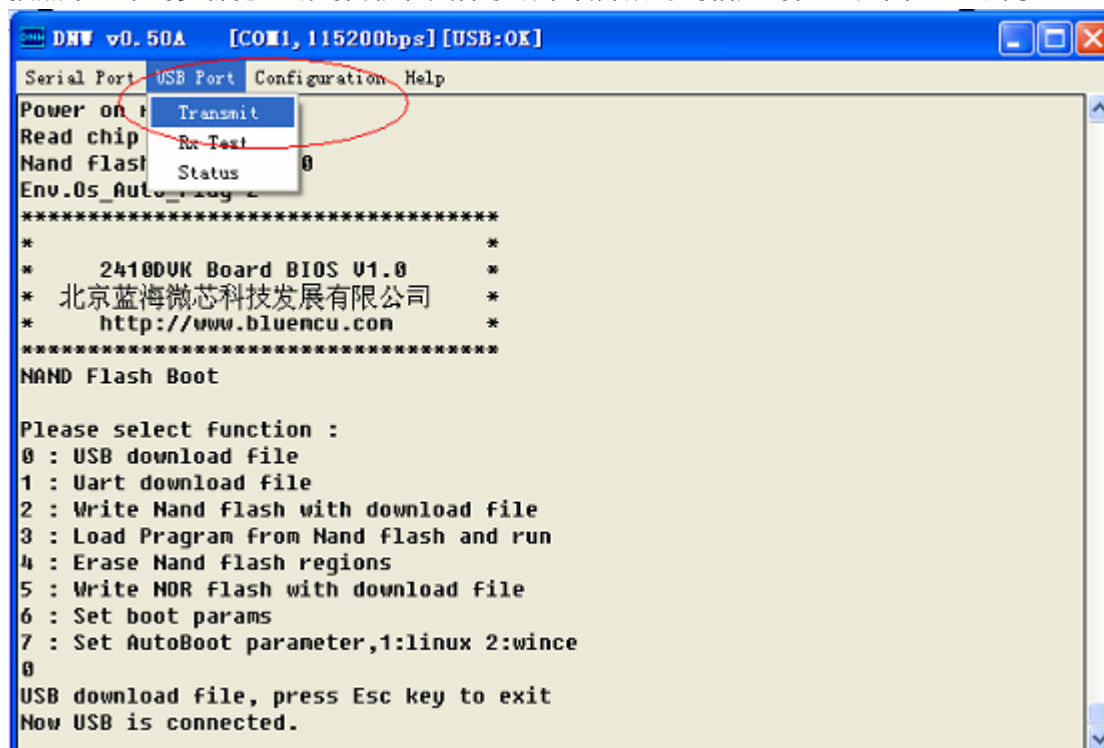


图 5-15

下载内核到 SDRAM , 注意 USB 的下载运行地址是 0x30200000。

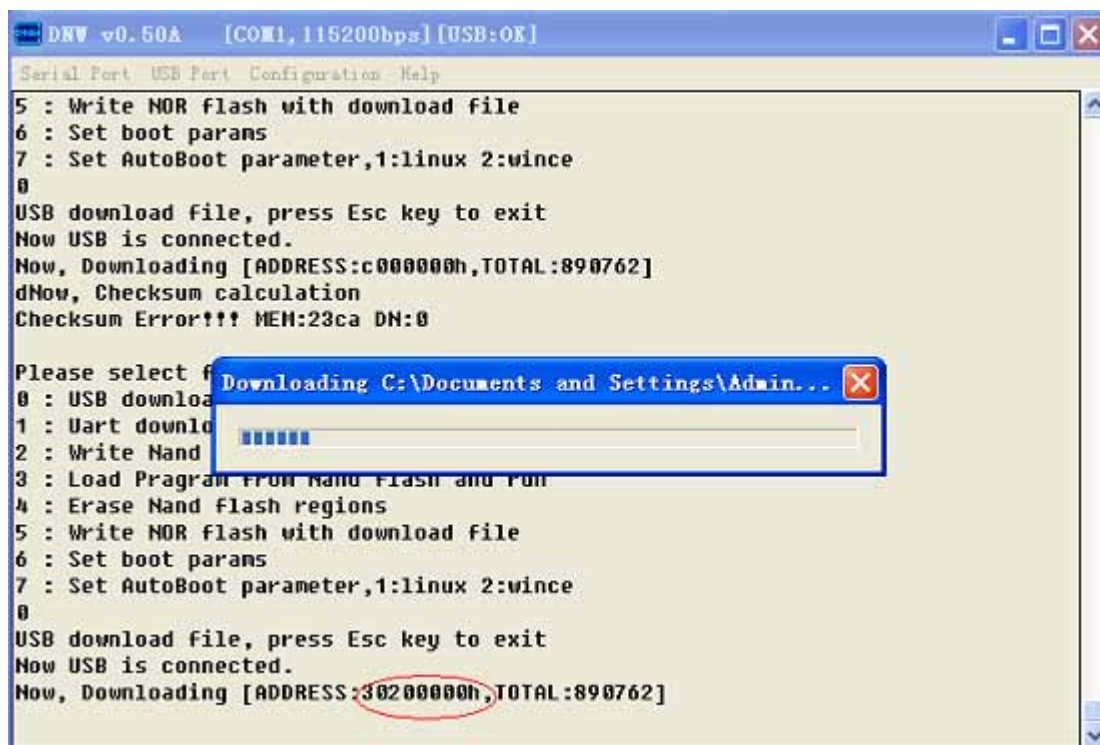


图 5-16

下载完成后提示 Do you want to run? [y/n] : n 然后与选择 2 将文件烧写到 NANDFLASH

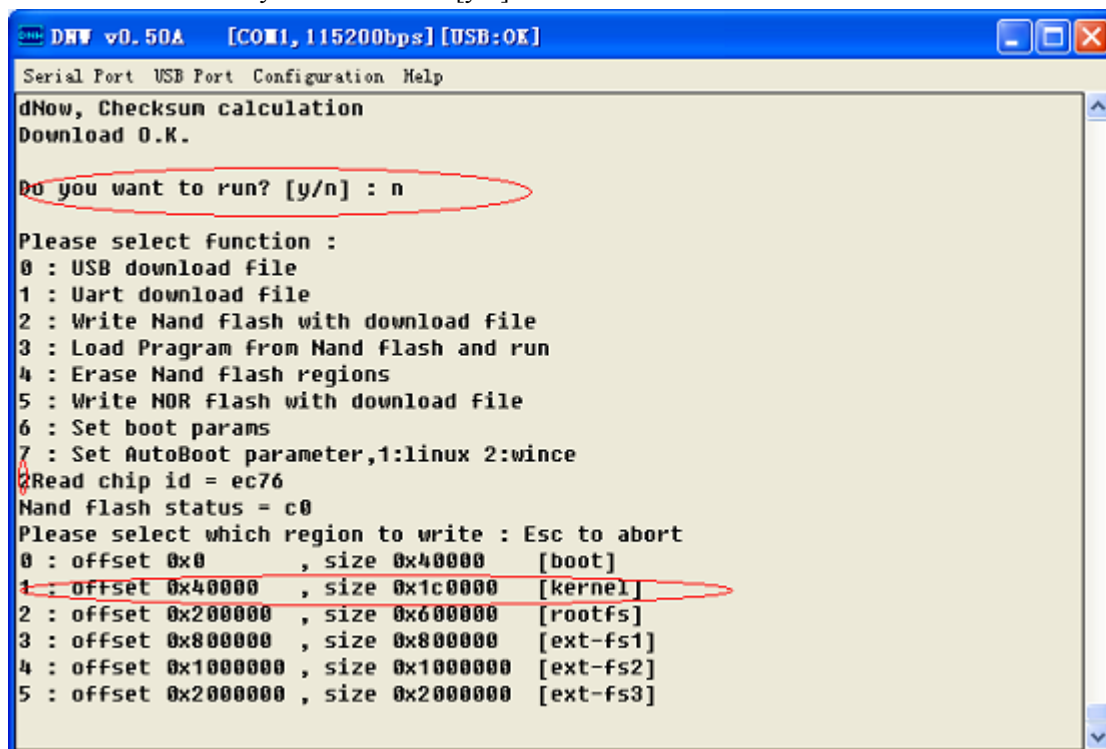


图 5-17

选择分区 1。到这里内核已经烧到 FLASH 里了。

6.2 烧写文件系统（目标代码里的 myrootfs.cramfs）

按照同样的方法烧写文件系统,不同的是要烧写到 2 分区。

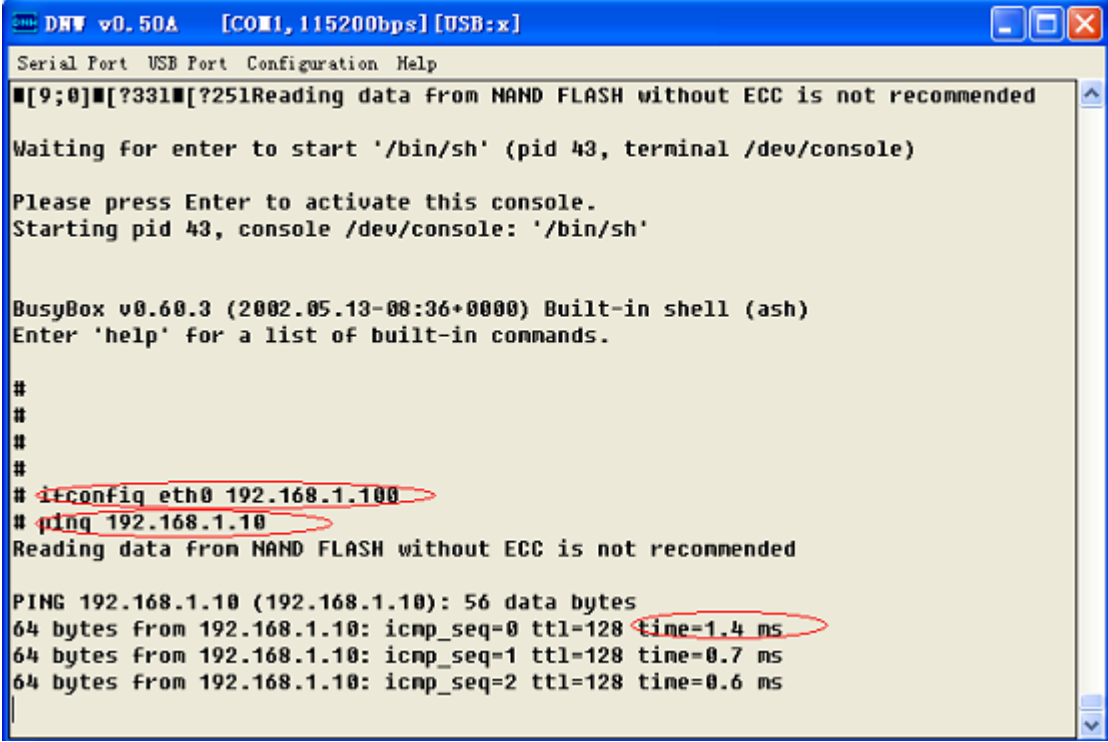
烧写完成后复位开发板,在 BIOS 里选择 3 启动 linux 操作系统。

第七章 linux 下的接口测试和应用软件开发

7.1 linux 下的接口测试

7.1.1 网络测试

配置本地计算机的 ip 地址 192.168.1.10，设置开发板的 ip 地址为 192.168.1.100
如下图所示：



```
DHW v0.50A [COM1,115200bps] [USB=x]
Serial Port USB Port Configuration Help
[9;0][?33l[?25lReading data from NAND FLASH without ECC is not recommended

Waiting for enter to start '/bin/sh' (pid 43, terminal /dev/console)

Please press Enter to activate this console.
Starting pid 43, console /dev/console: '/bin/sh'

BusyBox v0.60.3 (2002.05.13-08:36+0000) Built-in shell (ash)
Enter 'help' for a list of built-in commands.

#
#
#
#
# ifconfig eth0 192.168.1.100
# ping 192.168.1.10
Reading data from NAND FLASH without ECC is not recommended

PING 192.168.1.10 (192.168.1.10): 56 data bytes
64 bytes from 192.168.1.10: icmp_seq=0 ttl=128 time=1.4 ms
64 bytes from 192.168.1.10: icmp_seq=1 ttl=128 time=0.7 ms
64 bytes from 192.168.1.10: icmp_seq=2 ttl=128 time=0.6 ms
```

图 7-1

出现了如上图所示，表示网络已经开始正常工作了。

7.1.2 usb host 接口测试



```

DNV v0.50A [COM2, 115200bps] [USB:x]
Serial Port USB Port Configuration Help
[root@none] /]#
[root@none] /]#
[root@none] /]#
[root@none] /]#
[root@none] /]#
[root@none] /]# hub.c: USB new device connect on bus1/1, assigned device
number 2

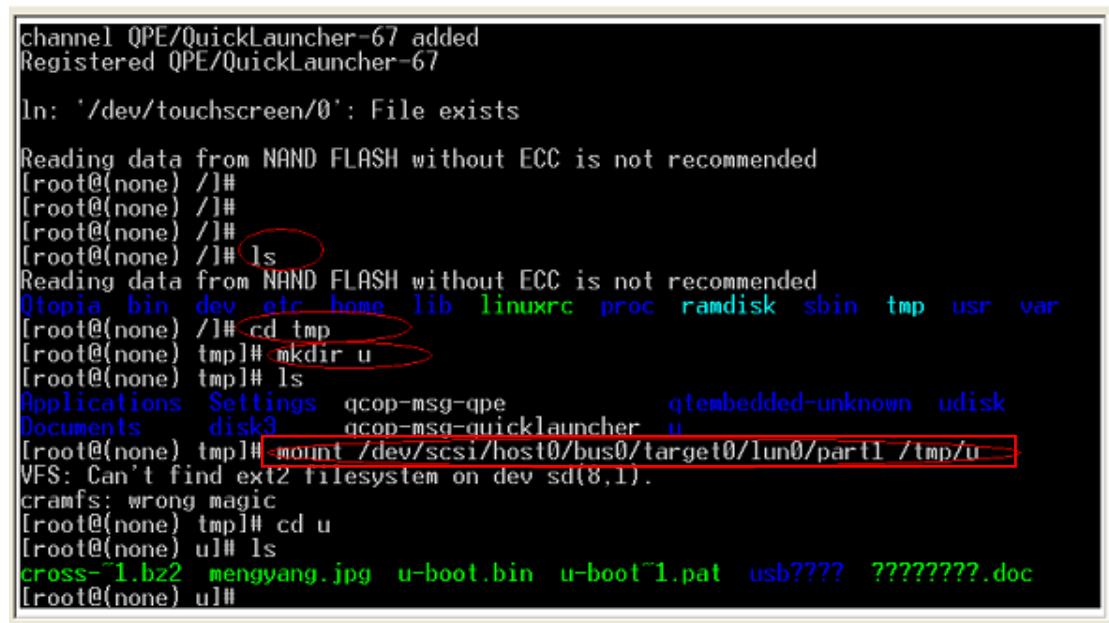
scsi0 : SCSI emulation for USB Mass Storage devices

Vendor:          Model: SAGE DISK X64M   Rev:  00
Type:   Direct-Access          ANSI SCSI revision: 02

Attached scsi removable disk sda at scsi0, channel 0, id 0, lun 0
SCSI device sda: 128000 512-byte hdwr sectors (66 MB)
sda: Write Protect is off
Partition check:
/dev/scsi/host0/bus0/target0/lun0: p1
    
```

图 7 - 2

这样系统已经检测到 usb 设备，接下来我们需要挂载 u 盘



```

channel QPE/QuickLauncher-67 added
Registered QPE/QuickLauncher-67

ln: '/dev/touchscreen/0': File exists

Reading data from NAND FLASH without ECC is not recommended
[root@none] /]#
[root@none] /]#
[root@none] /]#
[root@none] /]# ls
Reading data from NAND FLASH without ECC is not recommended
Qtopia bin dev etc home lib linuxrc proc ramdisk sbin tmp usr var
[root@none] /]# cd tmp
[root@none] tmp]# mkdir u
[root@none] tmp]# ls
Applications Settings qcop-msg-qpe          qtembedded-unknown udisk
Documents      disk3      qcop-msg-quicklauncher u
[root@none] tmp]# mount /dev/scsi/host0/bus0/target0/lun0/part1 /tmp/u
VFS: Can't find ext2 filesystem on dev sd(8,1).
cramfs: wrong magic
[root@none] tmp]# cd u
[root@none] u]# ls
cross-1.bz2 mengyang.jpg u-boot.bin u-boot~1.pat usb???? ?????.doc
[root@none] u]#
    
```

图 7 - 3

上图所示，我们已经可以读写 u 盘里面的内容了，下面我们在 u 盘里建立一个文件，证明 u 盘是可以读写，然后卸载 u 盘操作如下图 7 - 4 所示：

```

Applications  Settings  qcop-msg-qpe          qtembedded-unknown  udisk
Documents    disk3          qcop-msg-quicklauncher  u
[root@(none) tmp]# mount /dev/scsi/host0/bus0/target0/lun0/part1 /tmp/u
VFS: Can't find ext2 filesystem on dev sd(8,1).
cramfs: wrong magic
[root@(none) tmp]# cd u
[root@(none) u]# ls
cross~1.bz2  mengyang.jpg  u-boot.bin  u-boot~1.pat  usb????  ????????.doc
[root@(none) u]# Reading data from NAND FLASH without ECC is not recommended
Cannot suspend - no APM support in kernel

[root@(none) u]#
[root@(none) u]#
[root@(none) u]#
[root@(none) u]# mkdir mengyang
[root@(none) u]# ls
cross~1.bz2  mengyang.jpg  u-boot~1.pat  ????????.doc
mengyang    u-boot.bin    usb????
[root@(none) u]# cd mengyang
[root@(none) mengyang]# ls
[root@(none) mengyang]# cd /
[root@(none) /]# umount /tmp/u
Reading data from NAND FLASH without ECC is not recommended
[root@(none) /]#
    
```

图 7-4

7.2 linux 下的简单应用程序的编写实例

编辑源代码

在PC 上编辑以下源代码 ,并保存为hello.c ,工作环境为运行在虚拟机上的redhat9.0 利用linux的编辑软件vi编写自己的应用程序 ,vi的常用命令读者可以参考文档最后的附录首先在linux下shell输入命令vi如下图7-5所示 :

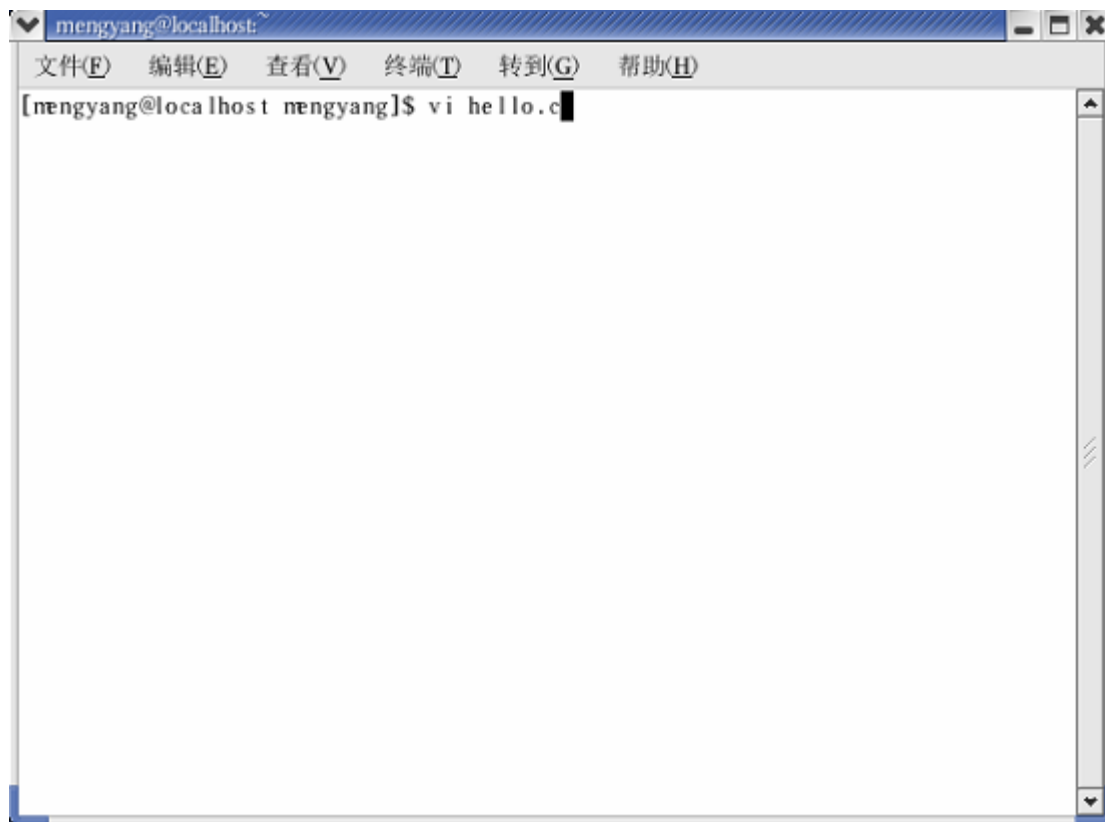


图 7-5

回车后如下图7-6所示 :



图 7-6

在上面的环境里输入i进入编辑状态键入如下代码，

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

Step2：编译hello

使用以下命令编译：

```
#arm-linux-gcc -o hello hello.c
```

将生成hello 可执行文档。

Step3：下载并运行

把优盘插入PC 的USB，然后执行以下命令把hello 复制到优盘

```
#mkdir /mnt/u
```

```
#mount -t vfat /dev/sda1 /mnt/u
```

```
#cp hello /mnt/u
```

```
#umount /mnt/u
```

把优盘拔下来插入到开发板的USB HOST（下面的口），按照以下命令操作(下面的命令是在超级终端或者Minicom对开发板进行操作)：

```
#mkdir /tmp/u
```

```
#mount -t vfat /dev/sda1 /tmp/u ; 挂接优盘
```

```
#cd /tmp
```

```
#./hello ; 执行hello
```

注意，Linux对U盘的支持不是很好，如果以上命令不成功的话，您可能要更换U盘。

另外，读者还可以利用 NFS 的方式与 PC 机连接，更加简单的完成 Linux 应用程序的在线调试。详细的方法请读者登陆我公司论坛。

注意：以下第八、九章涉及到的内容提供论坛技术支持，仅限于交流学习之用。

第八章 简单的 linux 驱动程序的开发

简单起见，我们以一个按键的实现（图6.3.）为例来讲述驱动的编写。平台上的一个按键和外部中断IRQ6 相连，当按键按下时，引脚输入低电平触发中断。下面我们来看驱动程序的设计。

设备初始化

驱动程序在init_keyboard()中实现向系统注册主次设备号，设备名，并初始化寄存器，如下：

```
void init_dev_set(void)
{
    ICR = 0x00; //低电平触发中断
    PDDIR = 0x80; //设置PD7 为输入
    PDSEL = 0x80; //PD7 作为I/O 与外部连接
    PDKBEN = 0x00; //键盘中断使能
```

```
int init_keyboard(void)
{
    #define keyboard_major 50 //手动分配主设备号为50
    #define keyboard_minor 0 //次设备号为0
    int rc;
    rc=register_chrdev(keyboard_major, " keyboard", & keyboard_fops);
    //向系统注册字符设备
    if(rc<0) //register_chrdev()的返回值小于零，注册失败
        printk("Panic! Could not register keyboard-Driver\n");
    else
        init_dev_set();
    return rc;
};
```

□ 注册中断和中断处理程序

在open 函数中向内核注册中断，如下：

```
static int keyboard_open(struct inode *inode, struct file *file)
{
    rc=request_irq(IRQ_MACHSPEC|IRQ6_IRQ_NUM, keyboard_interrupt, IRQ_FLG_STD, "
keyboard-IRQ", NULL/*Userdata!!!*/); //向内核注册中断
```

```
if(rc) //返回值不为零，则注册失败
{
    printk("keyboard-Driver: Error while installing interrupt handler\n");
    return -ENODEV;
};
MOD_INC_USE_COUNT;
return 0;
}

static void keyboard_interrupt(int irq, void *dev_id, struct pt_regs *regs)
{ //中断处理程序
    ISR |= (1<<18);
    wake_up_interruptible(&wq); /*唤醒队列*/
#ifdef DEBUG
    printk("I've woken up the process\n");
#endif
    return 0;
}

□ read 的实现
static int keyboard_read(struct inode *inode, struct file *file, char *buffer, int size)
{
    char * ch;
#ifdef DEBUG
    printk("I'm reading the device!\n");/*调用读函数时的调试信息*/
#endif
    interruptible_sleep_on(&wq);
#ifdef DEBUG
    printk("I'm wake up!\n");
#endif
    return 0;
}
```

第九章 图形界面开发入门

9.1 引言

自 MiniGUI 从 1998 年底推出以来,越来越多的人开始选择 MiniGUI 在 Linux 上开发实时嵌入式系统。MiniGUI 系统也逐渐成熟,应用到各种嵌入式系统中,包括 PDA、机顶盒、DVD、MP4 播放器等。本文讲解基于 Linux 和 MiniGUI 的嵌入式系统软件开发,,其内容不仅仅限于 MiniGUI 的编程,还会涉及到一些 Linux 下嵌入式系统软件开发的技巧,包括交叉编译环境的建立,根文件系统的生成等。

9.2 图形用户界面 MiniGUI 简介

MiniGUI 是由北京飞漫软件技术有限公司拥有版权并主持和维护的自由软件,遵循 GPL 条款发布(1.2.6 及之前的版本以 LGPL 条款发布),其目标是为实时嵌入式

Linux/uClinux、eCos、uC/OS-II、VxWorks 等操作系统建立一个快速、稳定和轻量级的图形用户界面支持系统。与QT/Embedded、Microwindows等其他GUI相比,MiniGUI最大的特点是占用资源少,在主频为66M的ARM7内核也能够稳定的运行。经过几年的发展已经发布了很多的版本,现在比较稳定的是1.6.x版本,而且源码是开放的,方便我们大家学习的研究。本文就是采用的此版本。

9.3 MiniGUI 在 S3C2410 开发板上的移植过程

9.3.1 MiniGUI 运行的环境的搭建

在 Linux/uClinux 上运行,要求 Linux FrameBuffer 驱动程序的功能正常。在 PC 上,如果显示芯片是 VESA 兼容的,则可以通过 Linux 的 VESA FrameBuffer 驱动程序获得较好的支持。对没有 FrameBuffer 支持的 Linux/uClinux 系统,需要编写特定的图形引擎才能运行 MiniGUI。MiniGUI 1.6.x 中已包含了针对非 FrameBuffer 驱动的图形引擎实现,另外在没有 FrameBuffer支持的系统上运行也可以借助虚拟软件实现相同的功能,在这里只是在有FrameBuffer支持的linux操作系统上的MiniGUI的移植。用到的交叉编译工具链是 cross-2.95.3.tar.bz2交叉编译器是 arm-linux-gcc2.95.3,这些工具软件通常在网上下载或是由硬件供应商会提供。在这里需要注意的是开发板上的linux的glibc的版本一定要和 cross-2.95.3.tar.bz2中的glibc一致,我们使用的是glibc-2.2.3。

9.3.2 交叉编译 MiniGUI

首先,从网上下载三个源码包 (<http://www.minigui.org/>),同时可以下载到相应的用户手册。

libminigui-1.6.x.tar.gz : MiniGUI 函数库源代码,其中包括 libminigui、libmgext 和 libvcongui。

minigui-res-1.6.tar.gz : MiniGUI 所使用的资源,包括基本字体、图标、位图和鼠标光标。

mg-samples-1.6.x.tar.gz : 《MiniGUI 编程指南》的配套示例程序。

先安装MiniGUI的函数库,我们先在PC的根目录下建立一个名为minigui的目录,将下载的三个源码包全部复制到minigui目录下,再在minigui目录下建立一个名为miniguitmp的目录,用来存放生成的文件和minigui库函数。

解压libminigui-1.6.x.tar.gz (tar zxf libminigui-1.6.x.tar.gz)

生成libminigui-1.6.x.目录,编写脚本文件配置lib内容如下:

```
./configure--host=arm-unknown-linux--enable=jpgsupport=no  
--enable-pngsupport=no--enable-gifsupport=no --disable-lite  
--prefix=/minigui/miniguitmp --enable-smdk2410ial=yes
```

保存为setup.sh

host用于指明宿主机的类型disable-lite用来指定生成基于线程的minigui版本而不是生成基于进程的minigui版本prefix用于指定MiniGUI函数库的安装路径/minigui/miniguitmp

进入该目录运行./configure脚本文件,我们为了适合我们的开发板做了适当的修改,其他的开发板类似。

根据PC机的交叉编译环境安装的路径修改libminigui-1.6.x.目录下的configure文件,在文件的开头处加入编译器的安装路径,然后保存,笔者主机上交叉编译器安装路径是 /usr/local/arm/,所以修改如下:

```
CC=/usr/local/arm/2.95.3/bin/arm-linux-gcc
CPP=/usr/local/arm/2.95.3/bin/cpp
LD=/usr/local/arm/2.95.3/bin/arm-linux-ld
AR=/usr/local/arm/2.95.3/bin/arm-linux-ar
RANLIB=/usr/local/arm/2.95.3/bin/arm-linux-ranlib
STRIP=/usr/local/arm/2.95.3/bin/arm-linux-strip
```

然后运行脚本文件

```
./ setup.sh
make
make install
```

如果运行成功，就会生成定制的MakeFile文件，执行make和make install 后相应的函数库就安装到/minigui/miniguitmp/lib下，在执行make install命令是用户一定要具有root权限。

下面安装MiniGUI资源，这些资源包括基本字体、图标、位图和鼠标光标等等。解压minigui-res-1.6.tar.gz (ar xzf minigui-res-1.6.tar.gz)，进入生成目录，修改configure.linux文件，指明TOPDIR=/minigui162/miniguitmp

执行

```
make install
```

就可以把相关文件拷贝到/minigui/miniguitmp/目录下的相关目录中去。

执行后会在/minigui/miniguitmp/ 下生成usr/local/lib/minigui/res 相关目录, 以下为res目录下的子目录，用户有兴趣可以进入以下目录查看其中的内容。

```
bmp cursor font icon imetab
```

9.3.3 制作带有MiniGUI的文件系统映象 (root_minigui.cramfs)

CRAMFS是Linus Torvalds撰写的只具备基本特性的文件系统。CRAMFS是一个简单的，经压缩以及只读的文件系统，通常要为文件系统建立映象，首先要建立并安装CRAMFS工具，在这里我们使用mkcramfs工具建立自己的文件系统映象。

开发板上的linux，自带有VESA FrameBuffer设备驱动程序，并且初始状态已经激活，这样MiniGUI就可以使用FrameBuffer作为图象引擎来显示图象。有关FrameBuffer支持的详细情况请读者参考《S3C2410开发板用户手册》北京蓝海微芯科技发展有限公司。

将/minigui/miniguitmp/lib中所有的库文件复制到root_minigui的/usr/lib中，将/minigui/miniguitmp/usr/local/lib/minigui目录复制到root_minigui的/usr/lib目录中，在/root_dir/root_minigui/usr/local 下执行ln -s /usr/lib lib,生成一个链接文件。修改/minigui/miniguitmp/etc/MiniGUI.cfg文件，

找到

```
ial_engine=console
mdev=/dev/mouse
```

改为

```
ial_engine=SMDK2410
```

```
mdev=/dev/ts
```

然后保存退出。

把/minigui162/miniguitmp/etc/MiniGUI.cfg文件复制到

/root_dir/root_minigui/usr/local/目录中。

以上把 MiniGUI 运行库和其它资源环境已经拷贝到 root_minigui 文件系统中，同时把 mg-samples-str-1.6.2/src/下可执行文复制到/usr/local/bin 下。然后编译文件系统，在 /root_dir 执行

```
[root@localhost root_dir]# mkcramfs root_minigui root_minigui.cramfs
```

就可以生成 root_minigui.cramfs 文件系统映象

把 root_minigui.cramfs 复制到 PC 机的/tftpboot 下，通过 tftp 命令把文件系统下载到 2410 开发板上。

重启开发板后执行：

```
#cp /usr/local/bin/* /tmp
```

//将可执行文件

复制到/tmp 临时目录下

```
#cd /tmp
```

//用 cd 命令去到

/tmp 目录下

经过以上工作，用户就可以在/tmp 目录下执行自己的应用程序了。

例如：

执行

```
#chmod 777 helloworld
```

//改变权限,使其为可执行

```
#./helloworld
```

//执行 helloworld

这样我们就成功的将带有minigui界面的文件系统成功的移植到了我们的开发板上了。

第十章 PB4.2 的安装和镜像文件的定制

10.1 PB4.2 开发环境的安装

Microsoft® Windows® CE .NET 是 Windows CE 3.0 的后续产品，它不仅是一个功能强劲的实时嵌入式操作系统，而且提供了众多强大工具，允许用户利用它快速开发出下一代的智能化小体积连接设备。借助于完善的操作系统功能和开发工具，Windows CE .NET 为开发人员提供了构建、调试和部署基于 Windows CE.NET 的定制设备所需的一切特性。平台开发工具 Platform Builder 是一个完全集成的开发环境（IDE），并且包括一个软件开发工具包（SDK）导出工具。Windows CE .NET 支持 Microsoft eMbedded Visual C++®和 Microsoft Visual Studio® .NET，为面向 Microsoft .NET Compact Framework（Microsoft .NET Framework 的一个子集）的 Web 服务和应用程序开发提供了一个完整的开发环境。利用这些工具，开发人员可以迅速开发出能够在最新硬件上运行各种应用程序的智能化设计。

为了建立WINCE4.2的应用开发环境，您需要准备好由微软公司发布的两张PB（Platform Builder 4.2）安装光盘，这两张光盘包含了微软公司的Windows® CE .NET 4.2 操作系统安装程序，以及把操作系统编译移植到指定目标硬件平台的工具 - 平台建立器

(Platform Builder 4.2)。下面介绍如何从这两张光盘安装WINCE4.2操作系统和 Platform Builder 4.2。运行SETUP.EXE，输入序列号在出现安装选项时选择ARMV4如图10-1所示

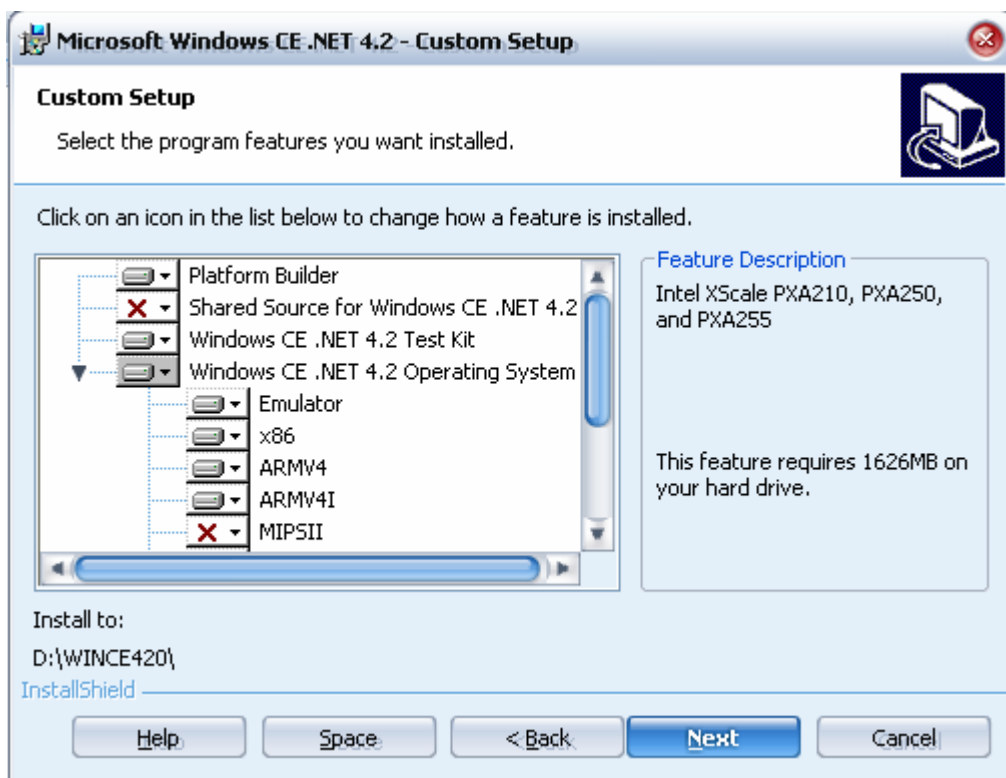


图 10-1

然后一路 NEXT,最后单击 Finsh 完成安装。

安装完成以后运行程序界面如下图 10-2 所示：

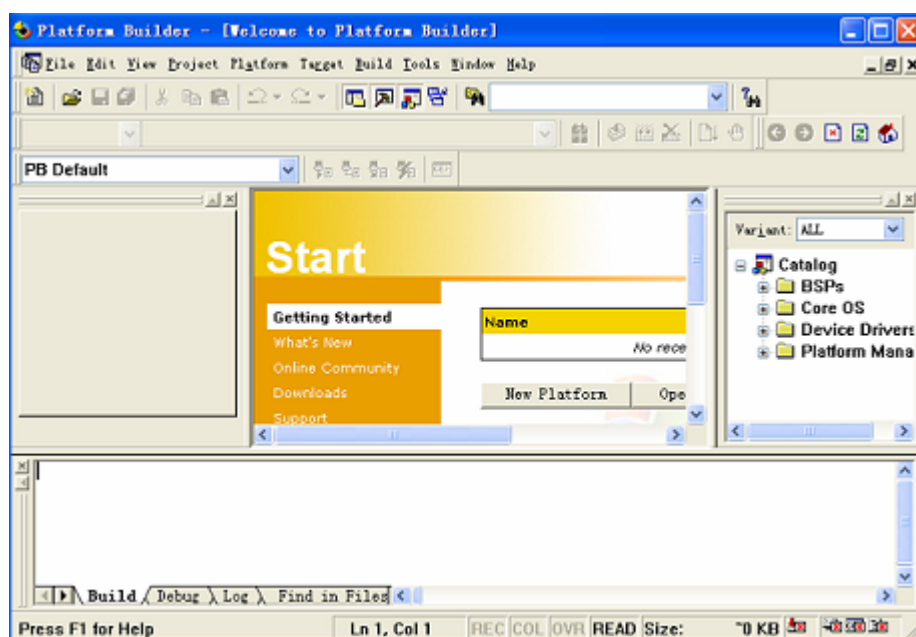


图 10-2

10.2 安装 BSP 包

将光盘里的 smdk2410 整个目录复制到 x:\WINCE420\PLATFORM 下,去掉只读属性,新建工程如图 10-3 所示:

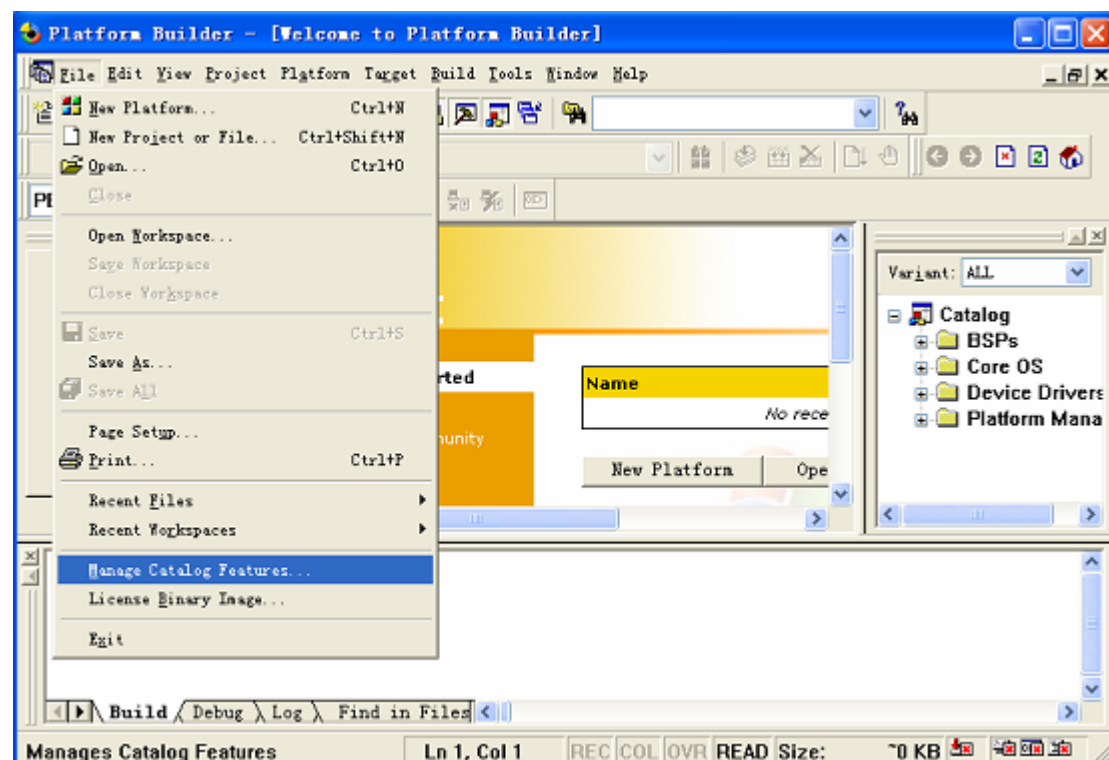


图 10-3

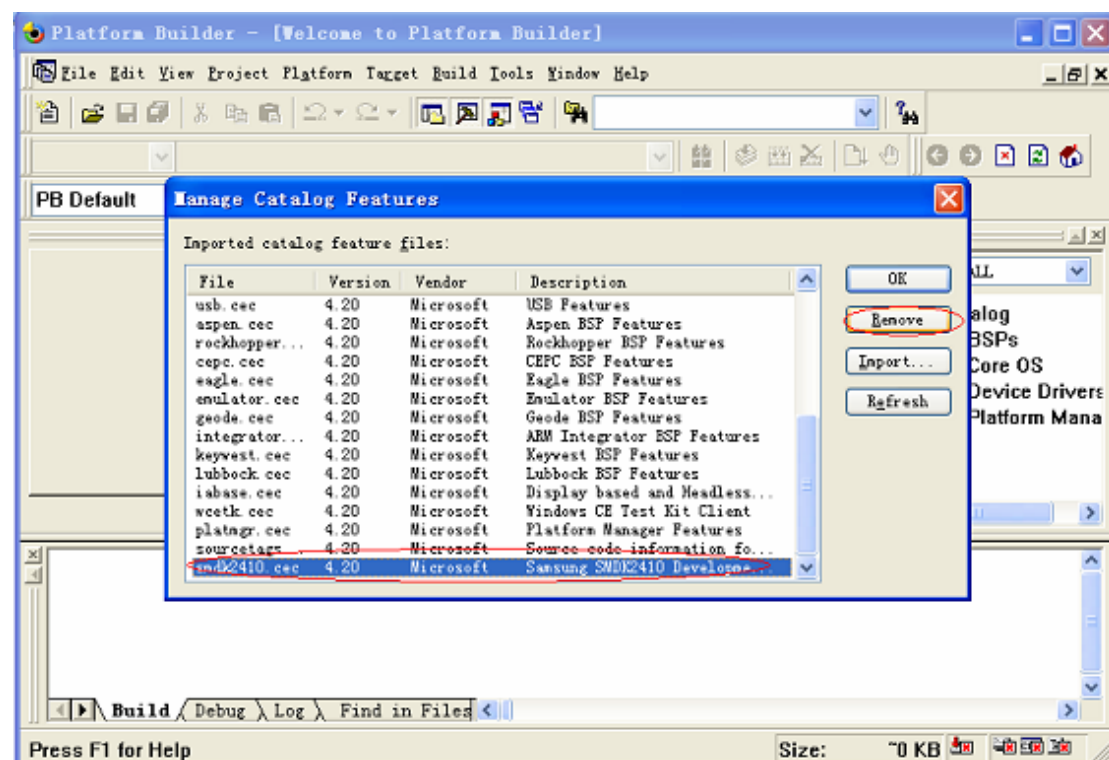


图 10-4

点击按钮“ Yes ”确定移除操作。然后 ,您需要“ 导入 ”(Import)新的“ SMDK2410.CEC ”文件。新的“ SMDK2410.CEC ”文件是从2410用户光盘的wince目录中的wincebsp.rar 压缩包解压出来的 ,一般我们将这个压缩包解压缩到wince420的安装目录下的 PLATFORM 子目录 ,例如WINCE420\PLATFORM ;然后就可以点击“ Import ” 按钮导入新的“ SMDK2410.CEC ”文件了 ,打开WINCE420\PLATFORM\SMDK2410目录 ,选中该目录下的“ SMDK2410.CEC ”文件 ,导入以后的状态如下图10-5所示 :

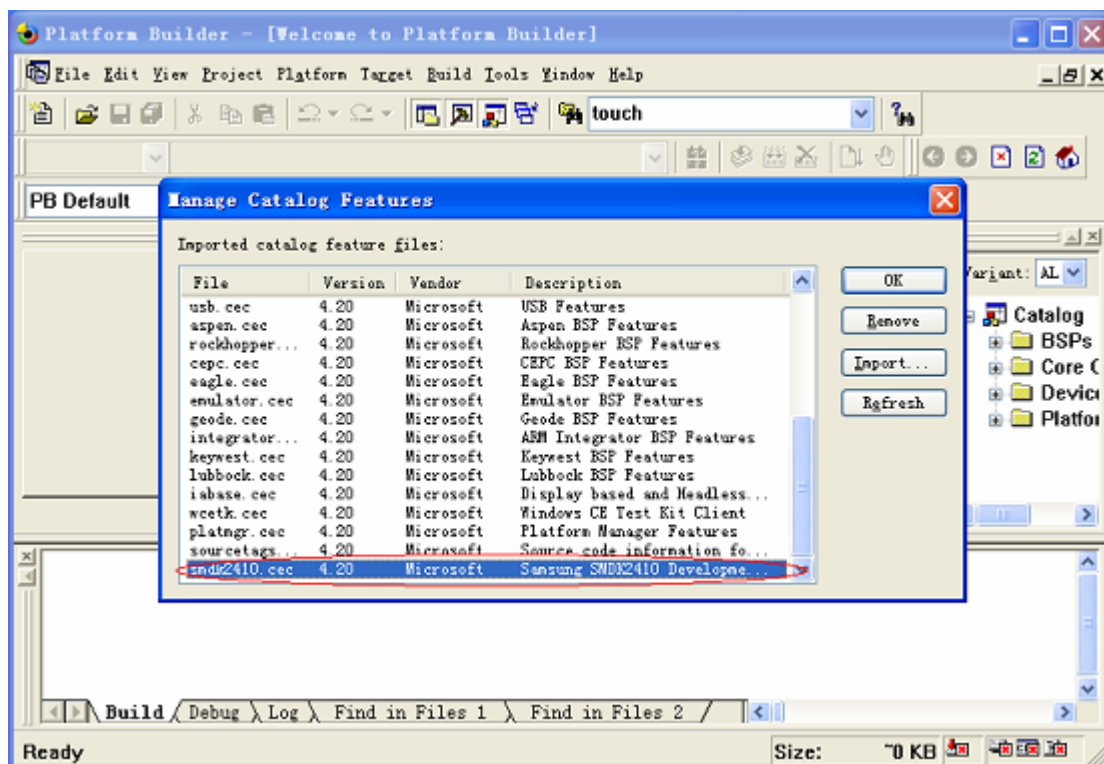


图 10-5

单击 OK 即完成 BSP 的安装。

10.3 创建新的 Platform 并生成新的 Image 文件

在 Platform Builder 下 ,点击 “ FILE ” 菜单下的 “ New Paltform” ,如下图10-6 :

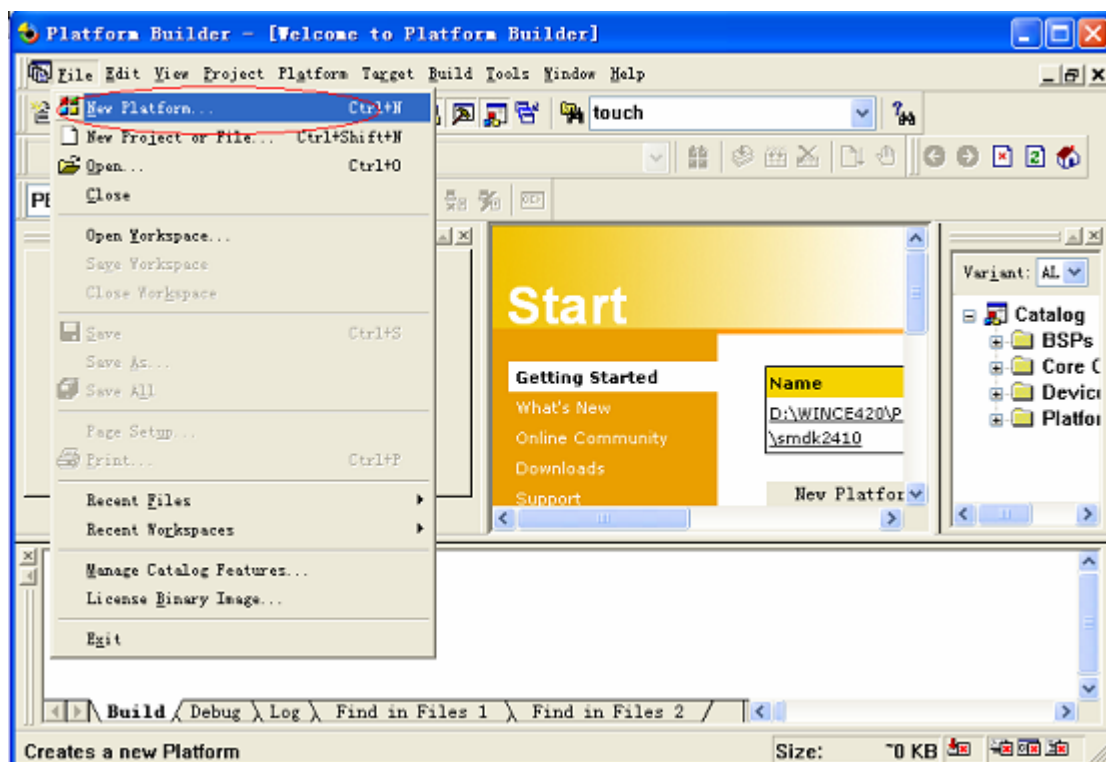


图 10-6

出现如图 10-7 所示：单击 NEXT

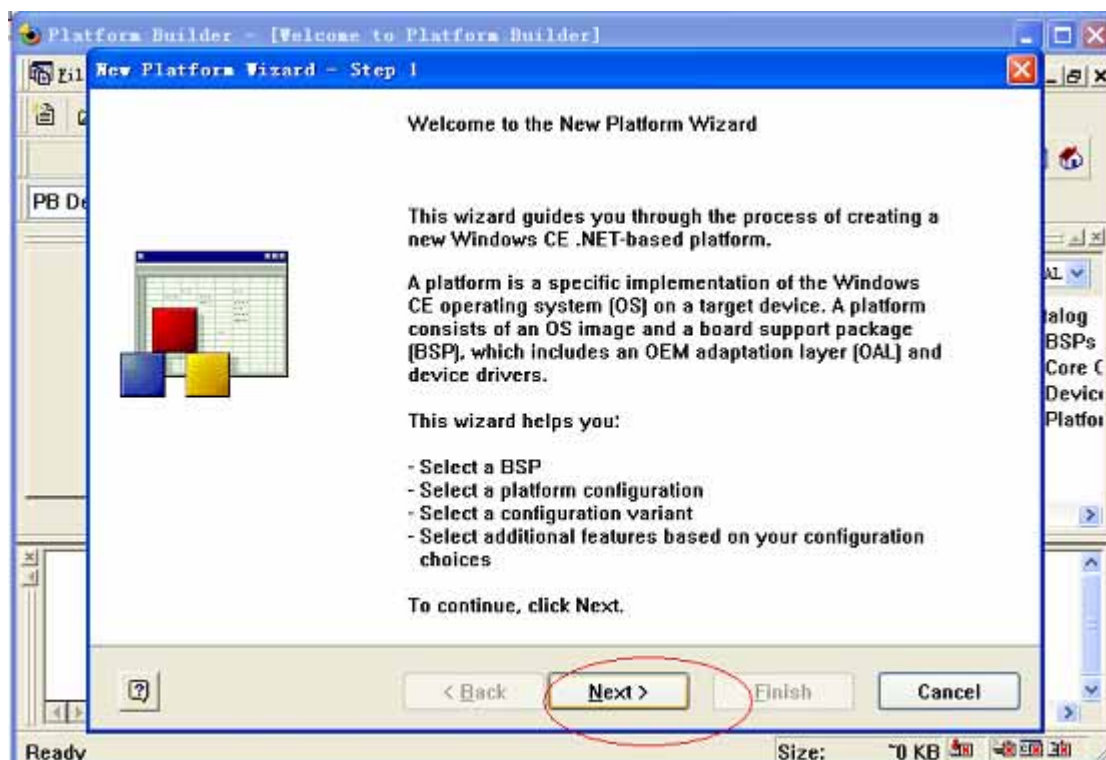


图 10-7

选择如图 10-8 所示的内容后单击 NEXT

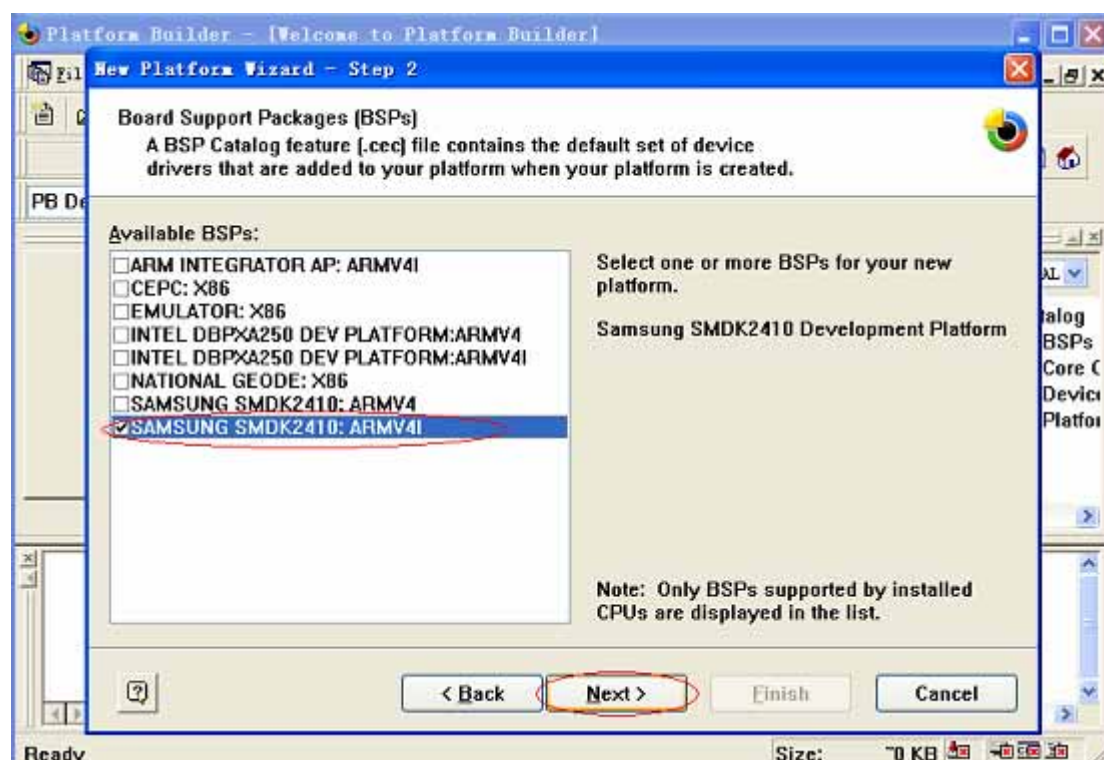


图 10-8

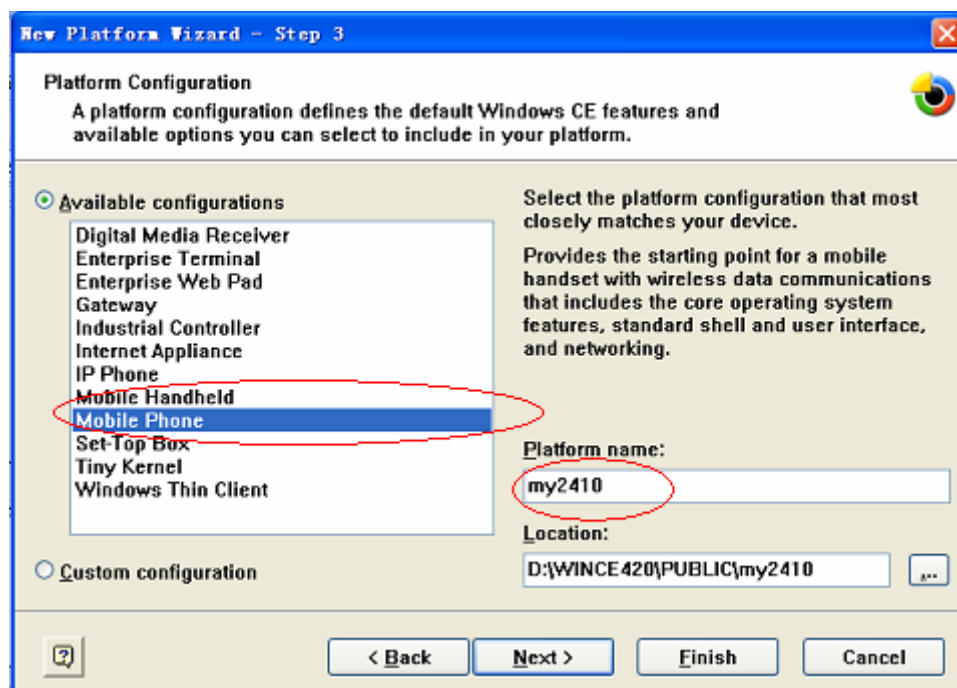


图 10-9

选择工程的类型和工程的名称。

定制在 wince 下经常用到的软件，如图 10-10 所示：

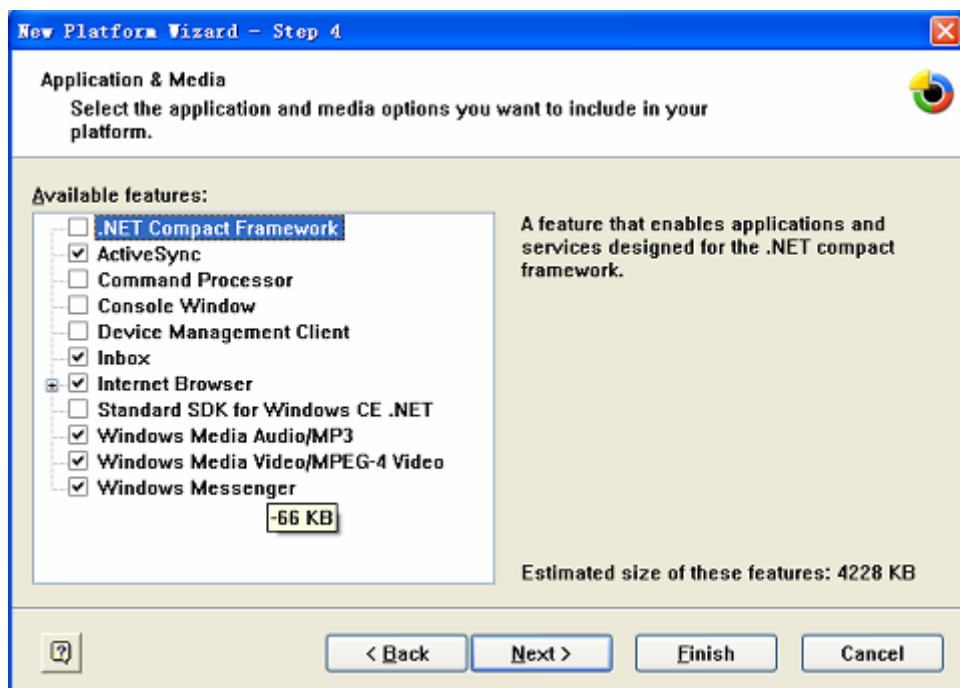


图 10-10

接下来我们一直 NEXT 直到完成为止。

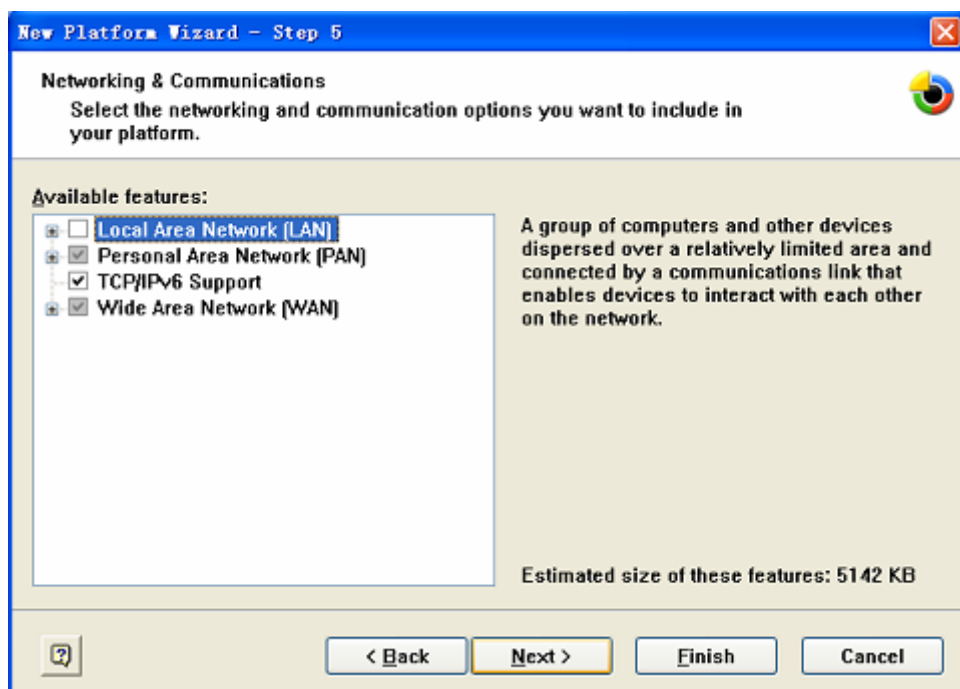


图 10-11

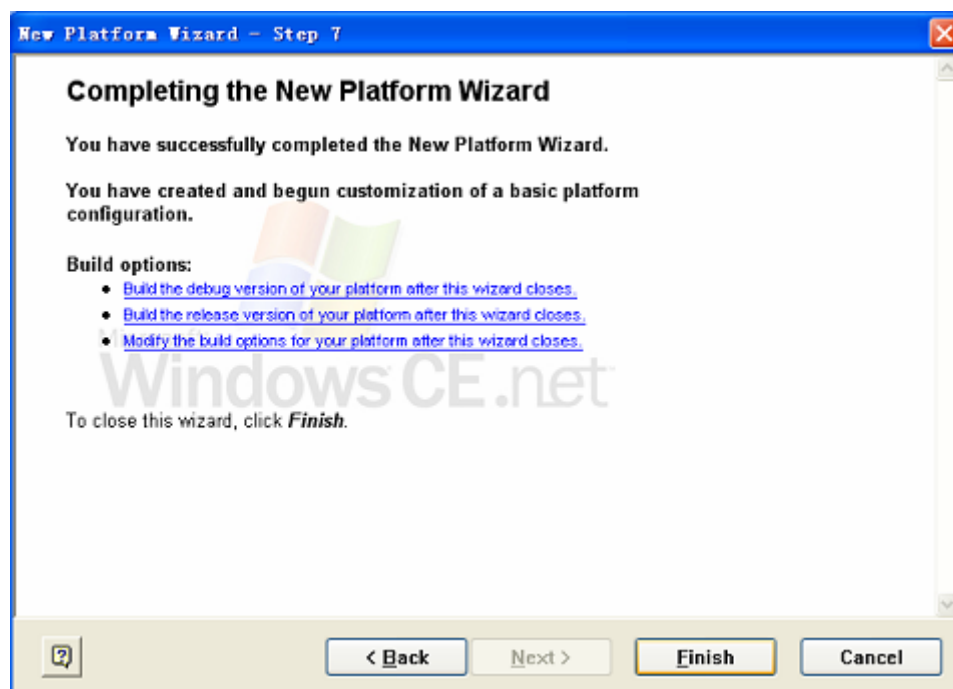


图 10-12

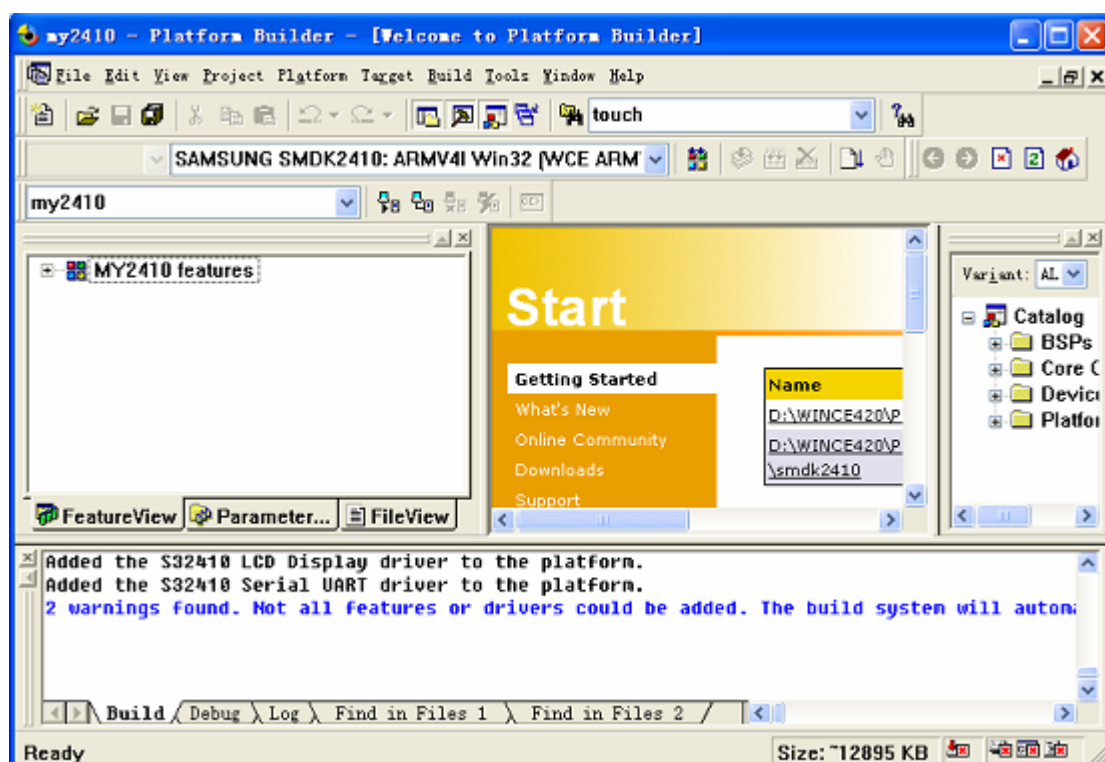


图 10-13

到这里新 Platform 平台建立完成。

选择 “ Platform ” 菜单中的 “ Setting ”，如下图：

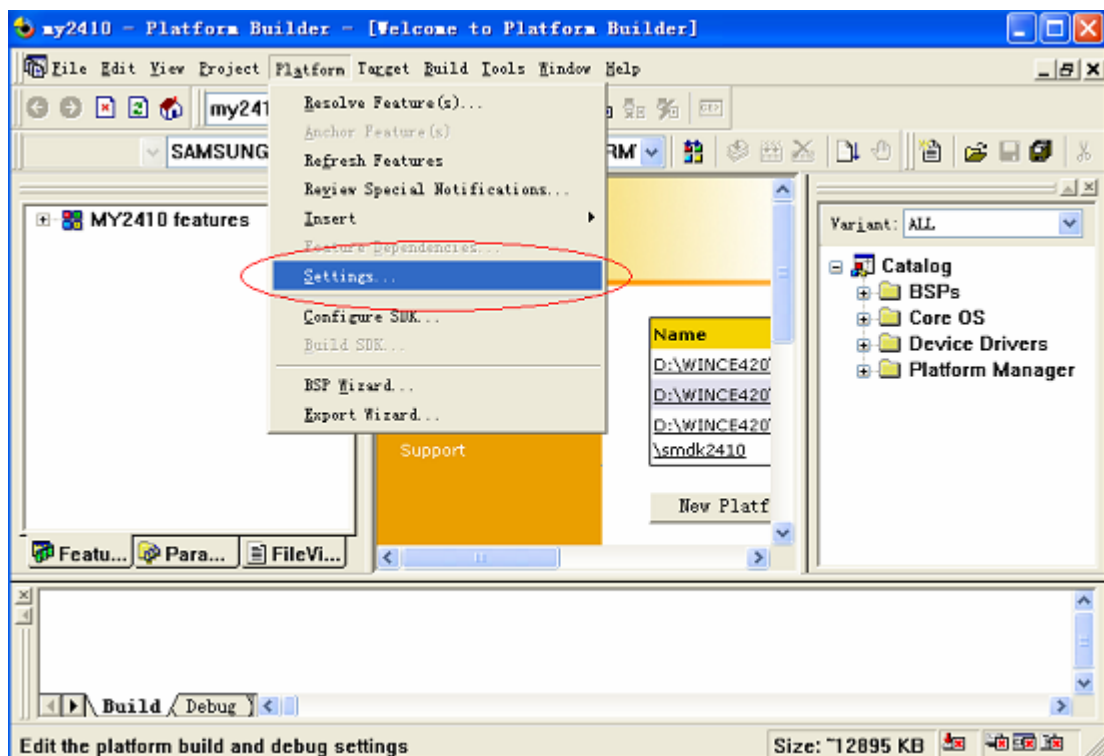


图 10-14

在 Build Options 标签页里，将 Build Options 设置成下图所示，并点击“OK”按钮：

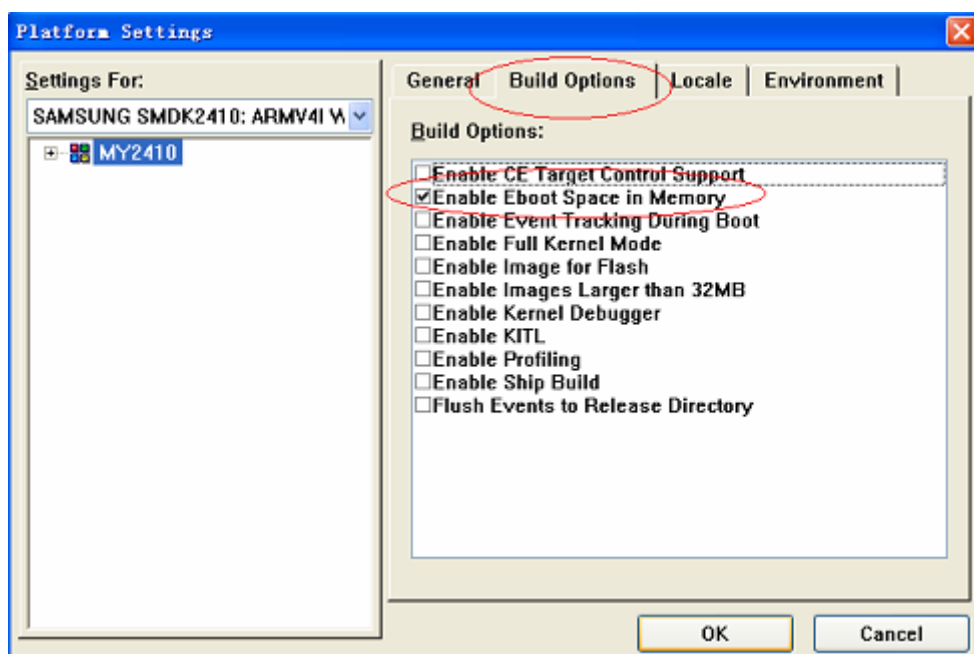


图 10-15

然后展开右边的组件列表：

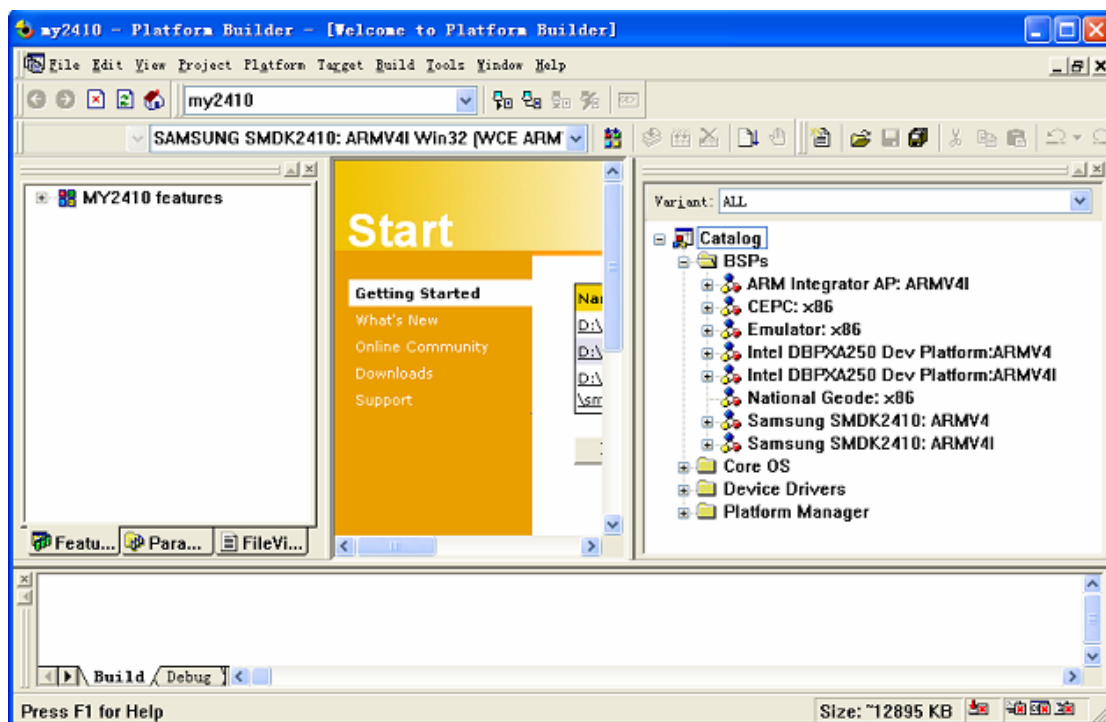


图 10-16

下面以加入 USB 鼠标为例讲解在 PB 中添加组件的过程，Catalog-CoreOS-Display based devices-Core OS Services-USB Host Support-USB Human Input Device[HID] Class Drivers-USB Mouse Only 上单击右键 Add to Platform

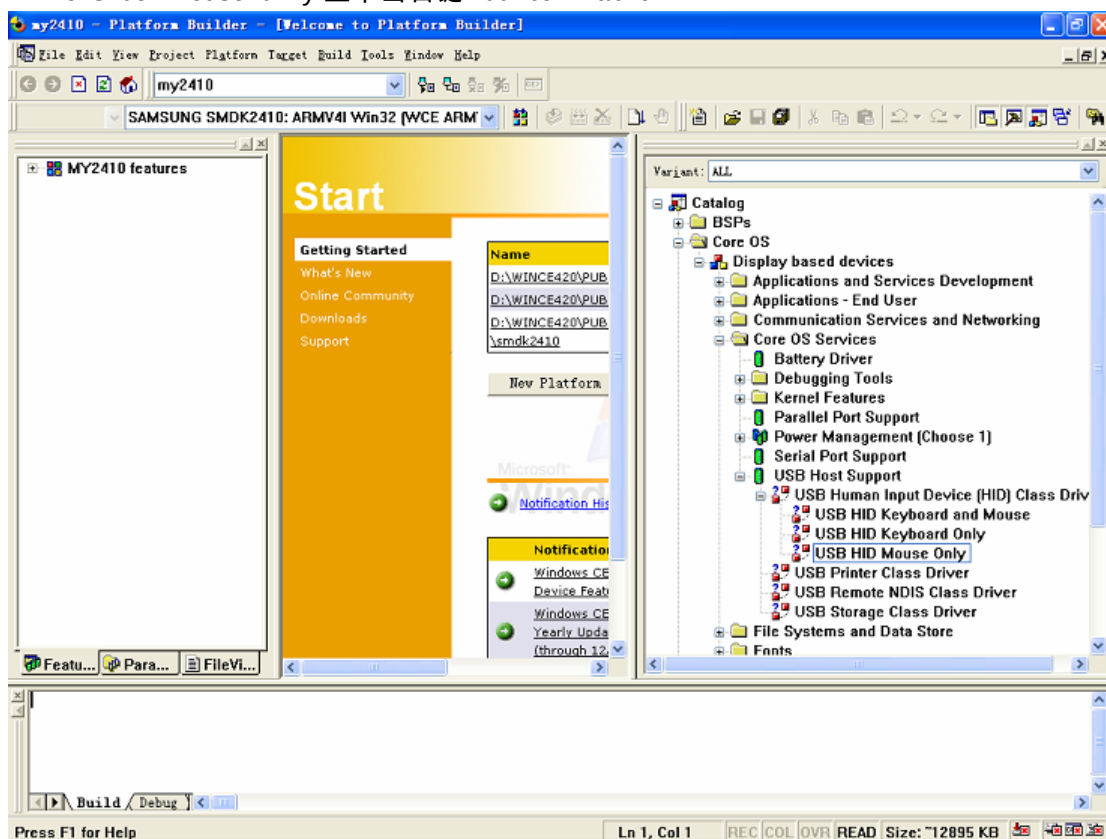


图 10-17

其他的常用组件已经添加到平台里了，若是用户还需要其他的标准功能，可以自行添加。

然后编译内核：如图 10-18 所示；

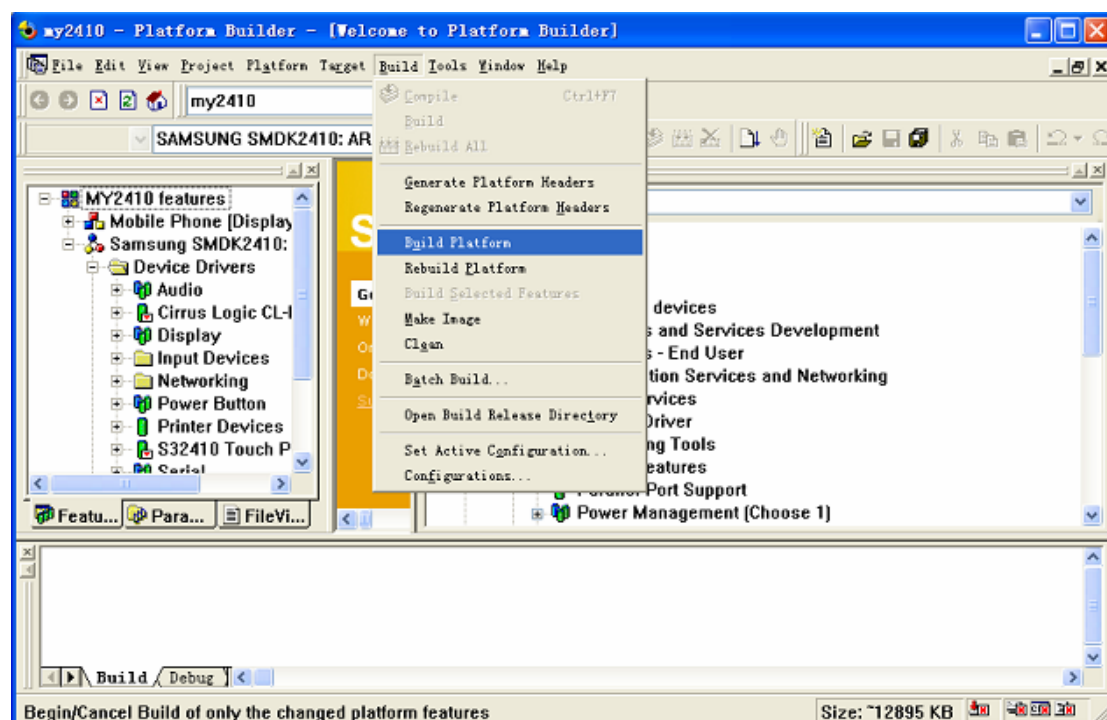


图 10-18

10.4 下载烧写内核镜像

按照第二章的内连接串口和 usb 设备如下图所示：



图 10-19



图 10-20

选择刚才生成的 NK.NB0 文件，或是在光盘里的目标代码里选择 NK.NB0 开始下载如下图 10-21 所示：

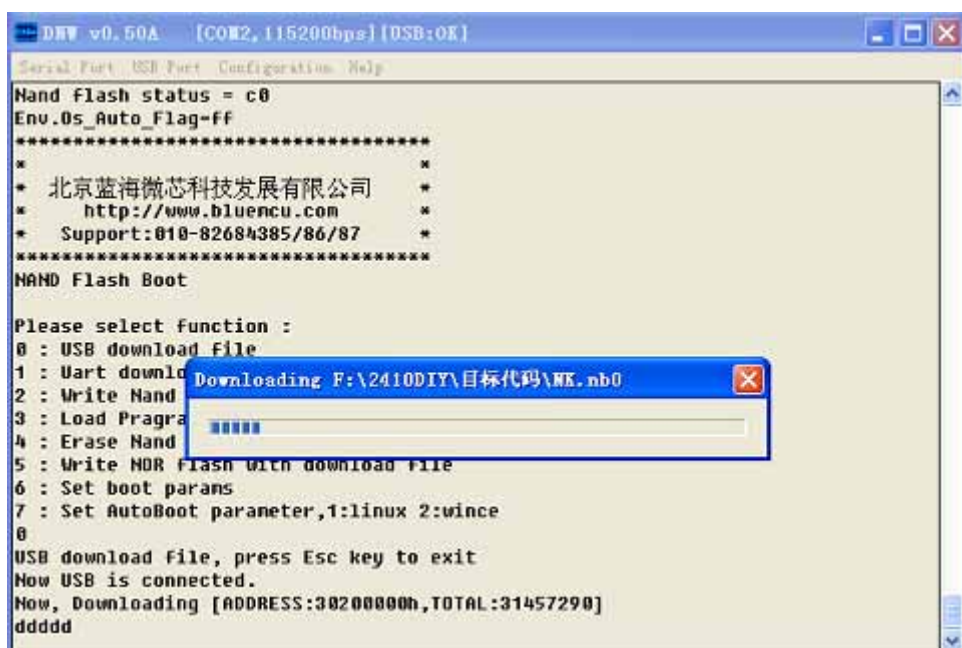


图 10-21

下载完成后系统提示是否运行程序，选择 n，如图 10-22 所示：

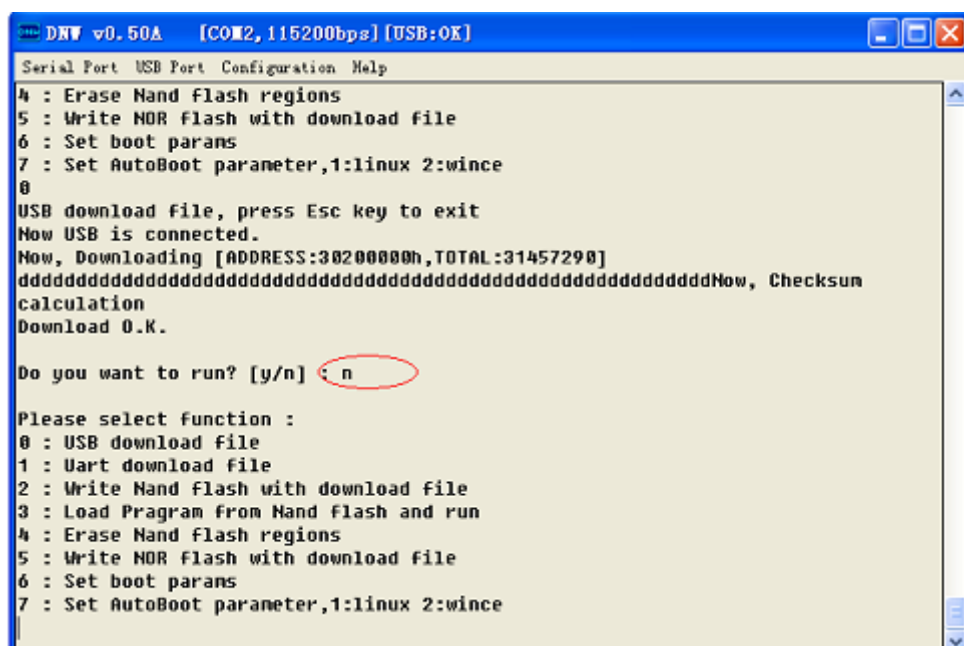


图 10-22

然后回到了 BIOS 的主界面，选择 2 Write Nand flash with download file
 然后选择 5 : offset 0x2000000 , size 0x1ffc000 [ext-fs3]，将 wince 内核镜像烧写到 nand flash 的分区 5 如下图 10-23 所示：

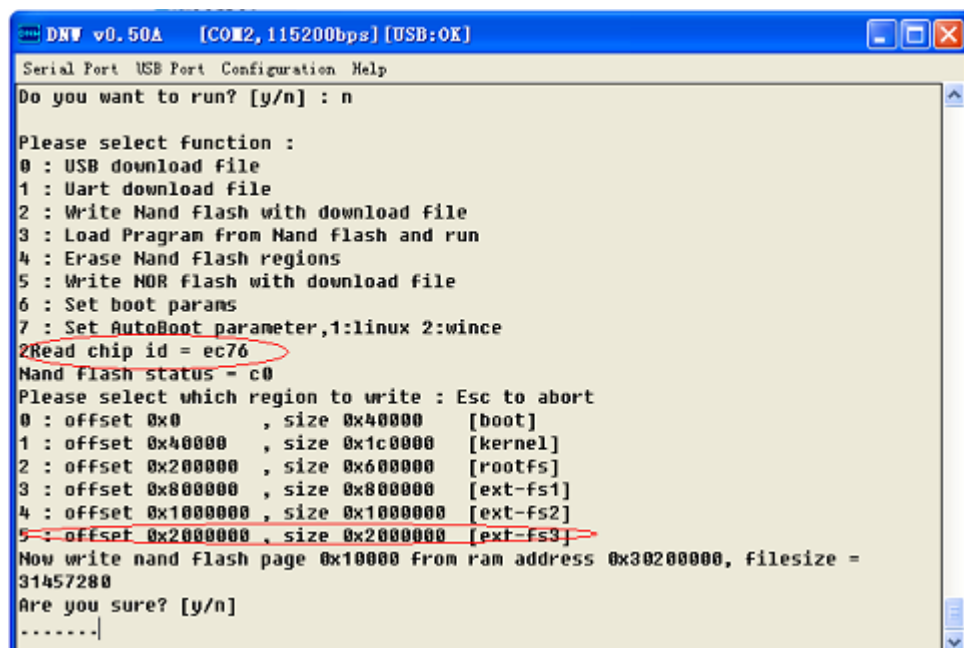


图 10-23

烧写完成后，选择 7 : Set AutoBoot parameter, 1:linux 2:wince，然后选择 2，复位开发板，如下图 10-24 所示：

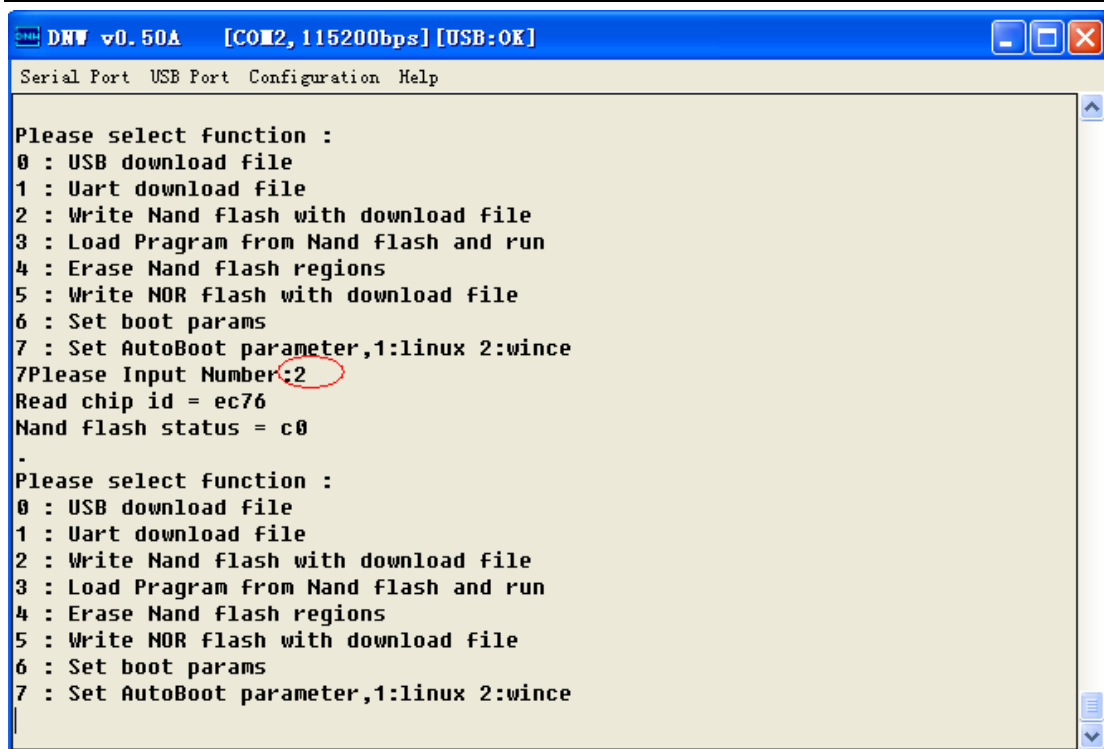


图 10 - 24

附录 一 如何将开发板恢复到出厂设置

(1) BIOS 的烧写到出厂的设置

连接 JTAG 的 JP3 和开发板的 JTAG 调试接口，在光盘的 BIOS 烧写目录双击 LJD2410DVK_BIOS.BAT 文件，如下图附-1 所示：

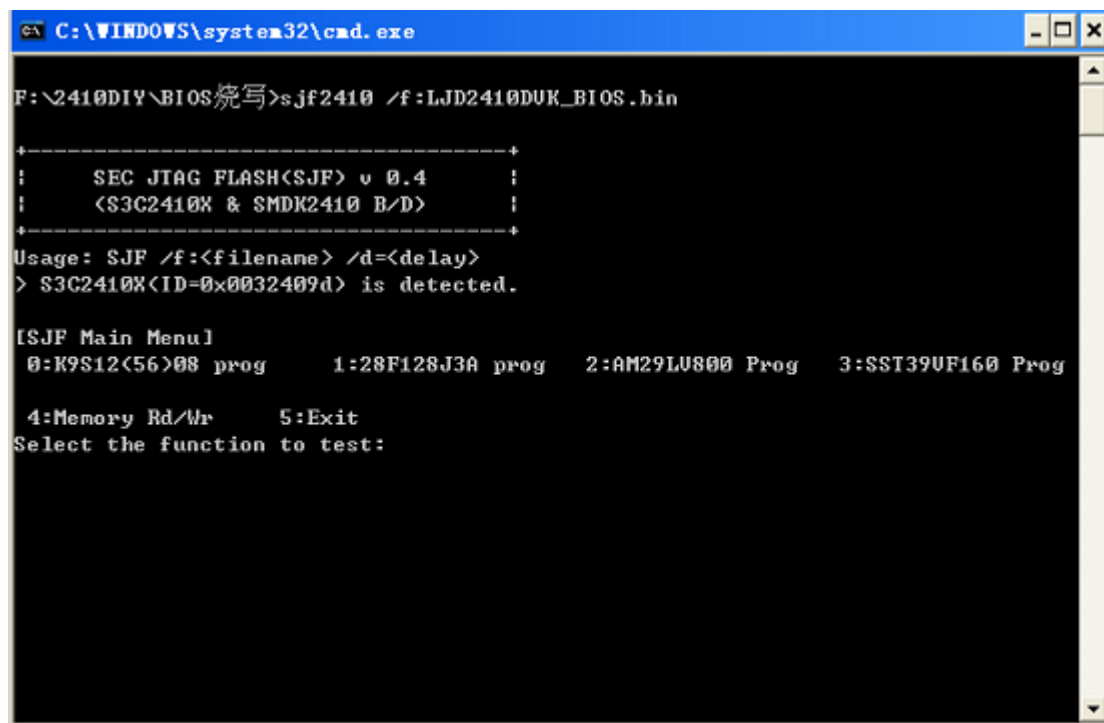
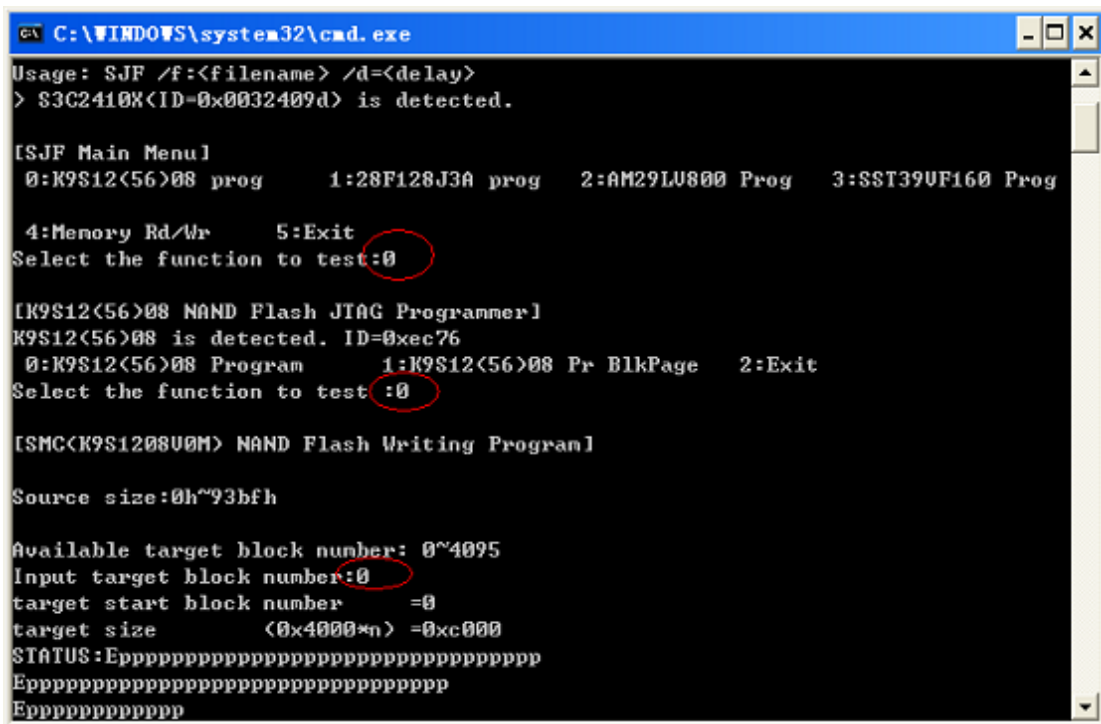


图 附-1



```

C:\WINDOWS\system32\cmd.exe
Usage: SJF /f:<filename> /d=<delay>
> S3C2410X<ID=0x0032409d> is detected.

[ SJF Main Menu ]
0:K9S12<56>08 prog      1:28F128J3A prog      2:AM29LV800 Prog      3:SST39VF160 Prog
4:Memory Rd/Wr      5:Exit
Select the function to test:0

[K9S12<56>08 NAND Flash JTAG Programmer]
K9S12<56>08 is detected. ID=0xec76
0:K9S12<56>08 Program      1:K9S12<56>08 Pr BlkPage      2:Exit
Select the function to test:0

[SMC<K9S1208U0M> NAND Flash Writing Program]

Source size:0h~93bfbh

Available target block number: 0~4095
Input target block number:0
target start block number =0
target size      <0x4000*n> =0xc000
STATUS:EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
EEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEEE
EEEEEEEEEEEEEEEE
    
```

图 附-2

按照上图所示一直选择 0 然后回车，即可完 BIOS 的烧写，烧写完成后选择 2，推出程序。复位开发板。

- (2) 烧写 Linux 内核和文件系统到 nandflash
详细步骤参考第六章的内容
- (3) 烧写 wince 内核镜像
详细步骤参考第十章的内容

附录二 开发板常见问题

(1) USB设备不能识别

检查USB驱动是否已经安装。

若已经安装过驱动，还是提示无法识别，用户可以按照下面的步骤解决：

断开USB线，复位开发板，在BIOS里选择0：USB download file，从新连接USB线，问题即可解决。

(2) 系统不能启动

当系统启动的时候提示USB错误，而导致系统不能启动时，请检查USB(Device)线是否断开。

(3) 串口没有打印信息

复位开发板，查看核心板上的发光二极管是否闪烁，若是闪烁证明系统已经正常启动，请仔细检查串口号和波特率（115200）

(4) 计算机没有串口

现在很多笔记本电脑没有串口，通常我们采取USB转串口线来解决这个问题。

(5) ADS1.2编译BIOS源码报错

当我们编译代码时，提示Can't open...filename,我们在键盘上按Ctrl+'-',然后选择All targets，点击make，即可完成编译。