

嵌入式原理及应用 II: 嵌入式系统

课程实验指导书

张莉君 刘玮

目 录

第一篇 windows 环境下裸机开发实验	3
实验一 ADS 集成开发环境的熟悉	3
实验二 ARM 汇编程序的设计	15
实验三 ARM 汇编与 C 混合编程	17
实验四 ARM 中断实验	18
第二篇 Linux 环境下裸机开发实验	21
实验五 熟悉 Linux 操作系统	21
实验六 Linux 常用命令的熟悉	25
实验七 文本编辑器 Vi 的熟悉	26
实验八 编译工具 GCC 的使用	31
实验九 交叉编译工具的使用	34
实验十 Linux 环境下裸机程序设计及编译	36
第三篇 Linux 操作系统移植与驱动初体验	39
实验十一 Linux 系统移植初体验	39
实验十二 Linux 操作系统下驱动开发初体验	42

第一篇 windows 环境下裸机开发实验

本章的实验是在 windows 操作系统的 ADS 集成开发环境下，实现开发板裸机（开发板上无操作系统）程序的开发。本章实验涉及到 S3C2440 的 GPIO 模块和中断控制模块。

实验一 ADS 集成开发环境的熟悉

实验目的：	熟悉 windows 下 ADS 开发环境
	熟悉 TQ2440 电路板。
硬件连接：	开发板三线连接（电源线、串口线、USB 线），Nor Flash 启动
实验要求：	掌握 ADS 开发环境的使用，完成硬件连接及程序下载。
实验内容：	1. 在 ADS 中的实现新工程创建、添加文件、编译等步骤
	2. 采用 AXD 软件仿真工具调试程序、观察参数等
	3. 可执行程序的下下载
	4. 点亮开发板上 LED 灯

1. 创建一个新的工程

1) 通过 windows 的”开始” → “所有程序”→ “ARM” → “CodeWarrior for ARM Developer Suite”（或在桌面直接打开 ADS 软件），打开软件界面如图 1-1 所示：

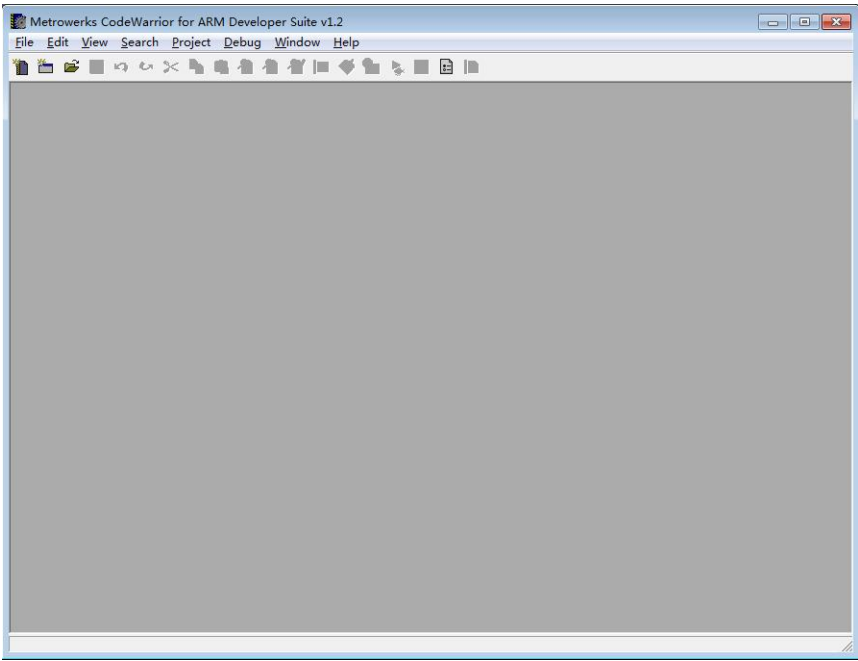


图 1-1 ADS 软件界面

2) 选择“File”→“New”菜单项，则弹出“New”对话框，并选择“Project”选项卡，如图 1-2 所示。在“Project”选项卡中选择“ARM Executable Image”选项，在 “Project name”列表中输入工

程名称（如 ASM），在 Location 列表框中指定工程的保持路径，点击确定就建好了一个工程，如图 1-3 所示。

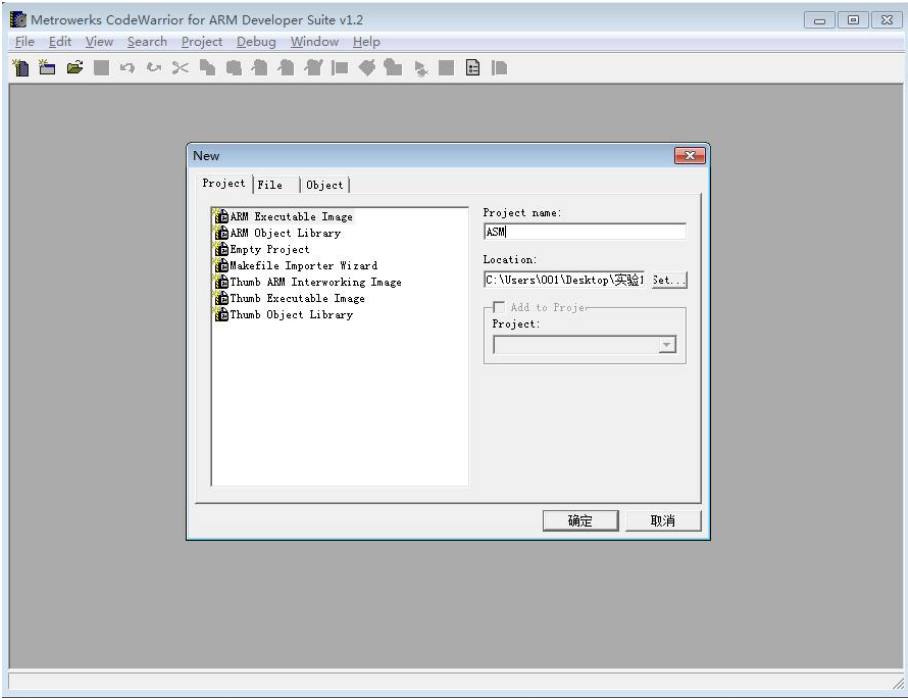


图 1-2 新建工程

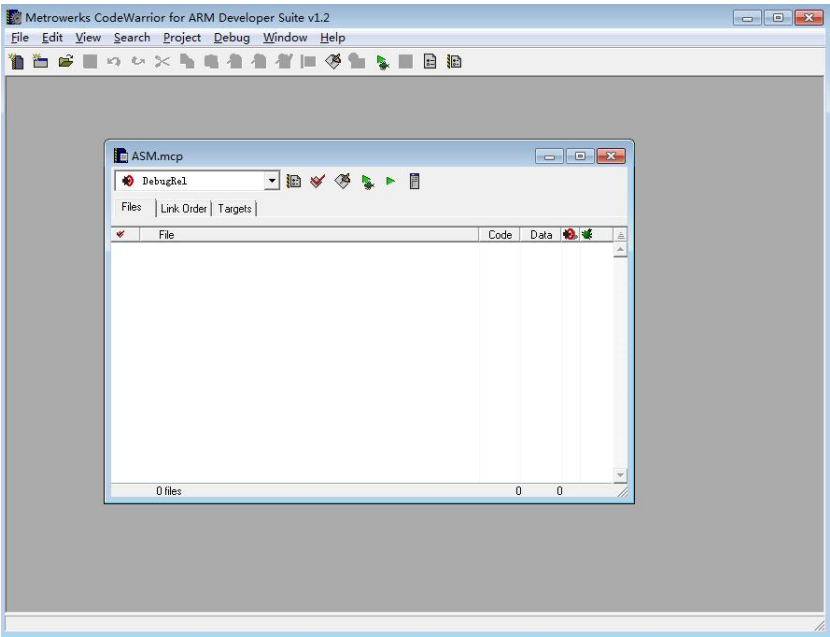


图 1-3 一个空工程

2. 建立一个新的源文件

选择“File”→“New”菜单项，则弹出 New 对话框，并选择“File”选项卡。在“File name”

列表框输入文件名(如 `asm.s`,注:源文件名字可以不同,但是必须.s 结尾),选中“Add to Project”项则将创建的文件添加到指定的工程项目中,这里选中添加到刚创建的 `ASM.mcp` 中,选择“DebugRel”选项和“Debug”选项,如图 1-4 所示。如图 1-5 创建了 `asm.s` 文件后,就可以输入汇编源代码了。

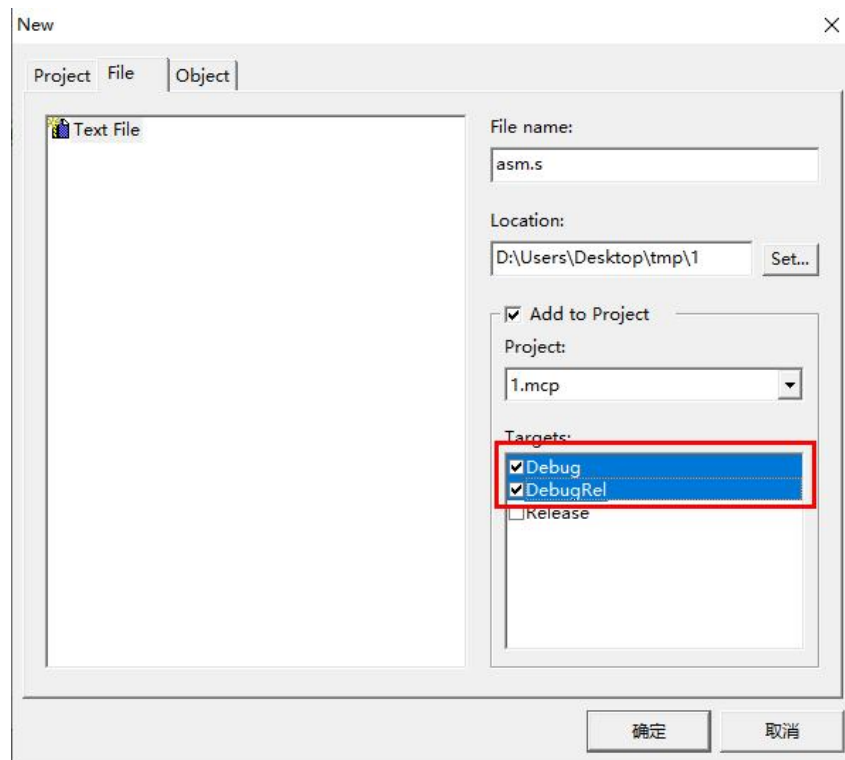


图 1-4 为工程添加文件

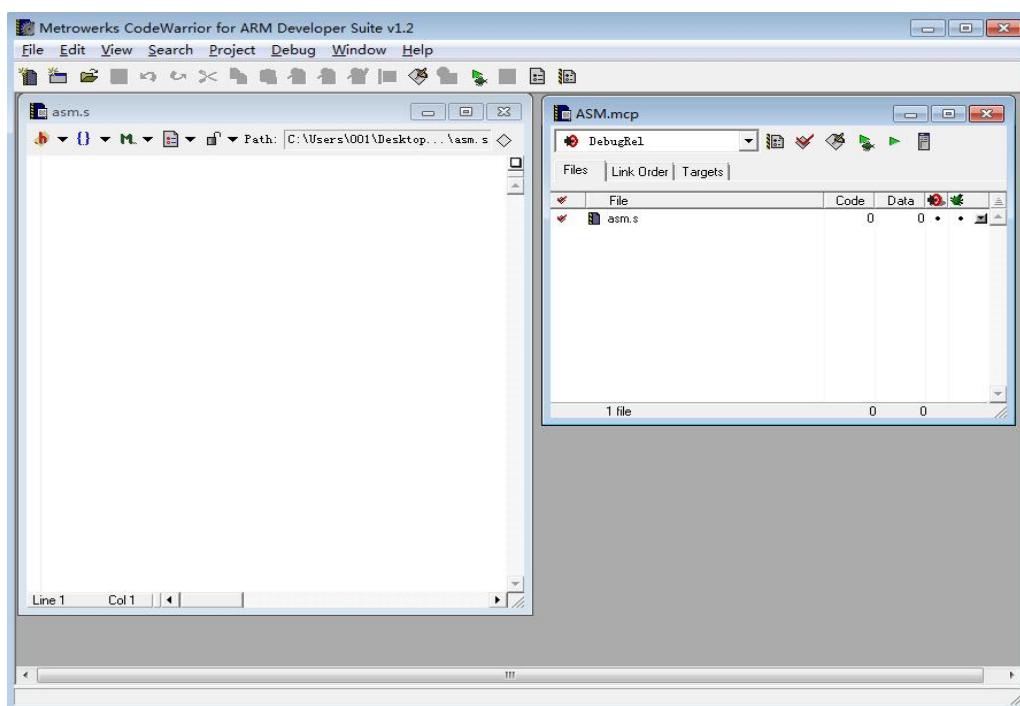


图 1-5 添加文件

3. ADS 工程编译环境的设置

ADS 的工程设置有很多选项，但是大部分选项保持默认即可。这里需要设置与映像文件相关的链接选项。

1) 选中“Edit”→“DebugRel Settings”菜单项，打开“DebugRel Settings”对话框，如图 1-6 所示，选择“Target Settings”，设置“Post-linker”为“ARM FromELF”。

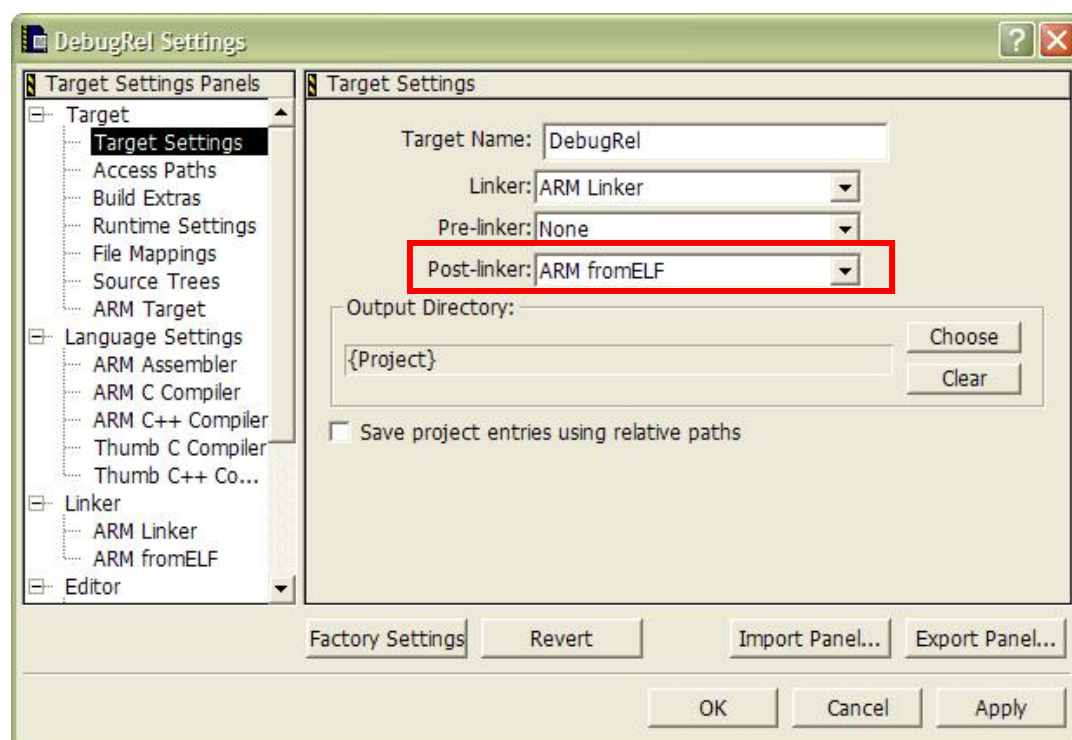


图 1-6 Target Settings 设置

2) “Language Settings”级联菜单中的“ARM Assemble”的“Target”→“Architecture or Processor”项选择 ARM920T，如图 1-7。

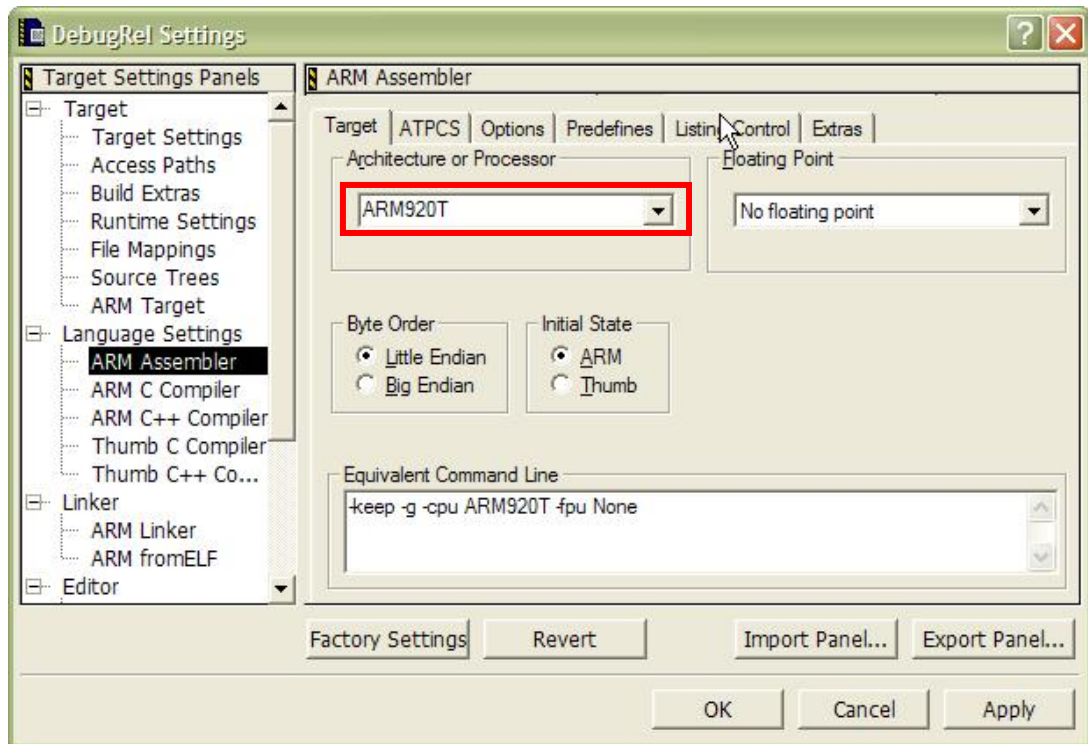


图 1-7 Language Settings 设置

3) 选择“Linker”级联菜单下的“ARM Linker”项，并在“Output”选项卡中的“RO Base”列表框中输入“0x30000000”，如图 1-8 所示。

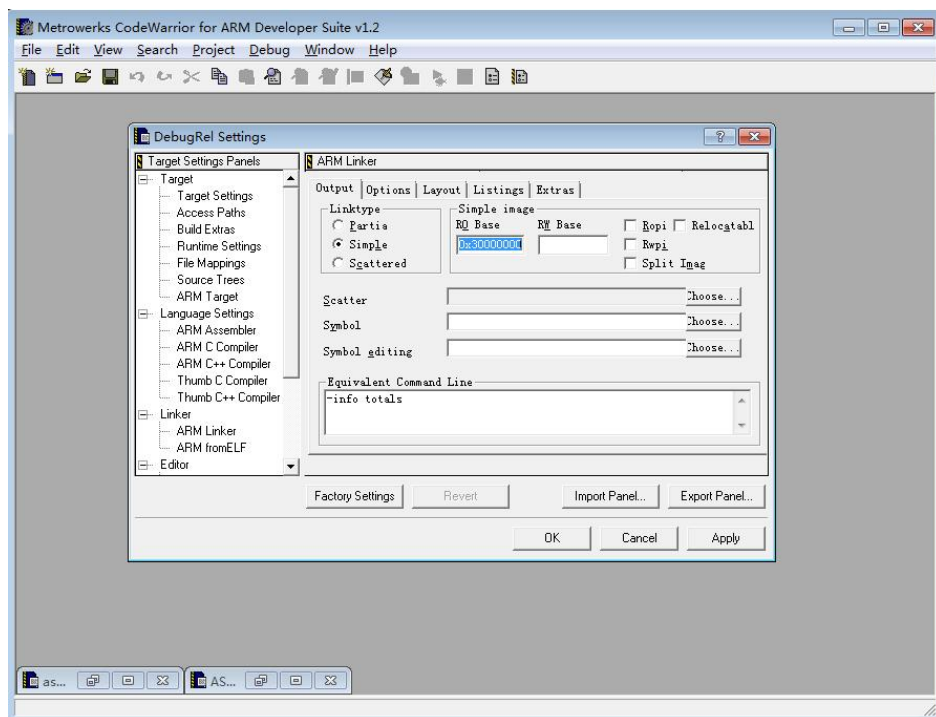


图 1-8 ARM Linker 设置

4) 设置映像文件的入口地址，如图 1-9 所示，在“ARM Linker”的“Layout”选项中，asm.o

为 ASM.s 编译生成的目标文件，Init 为代码段的段名。

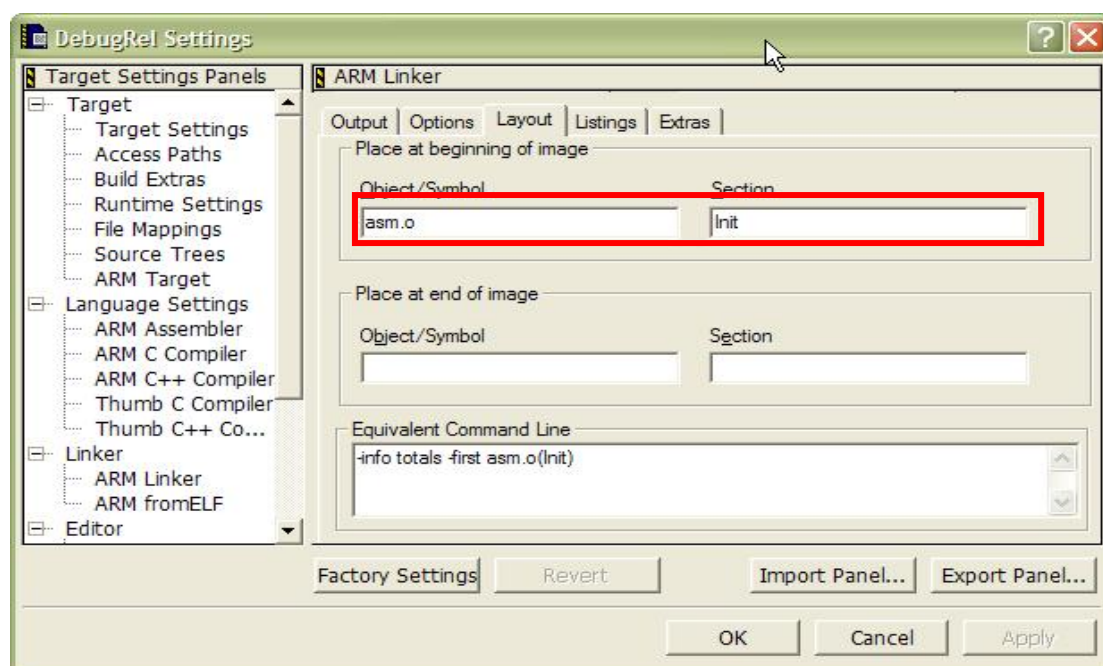


图 1-9 ARM 映像文件入口地址设置

5) 在“ARM fromELF”中，设置输出文件名字，如图 1-10 所示。

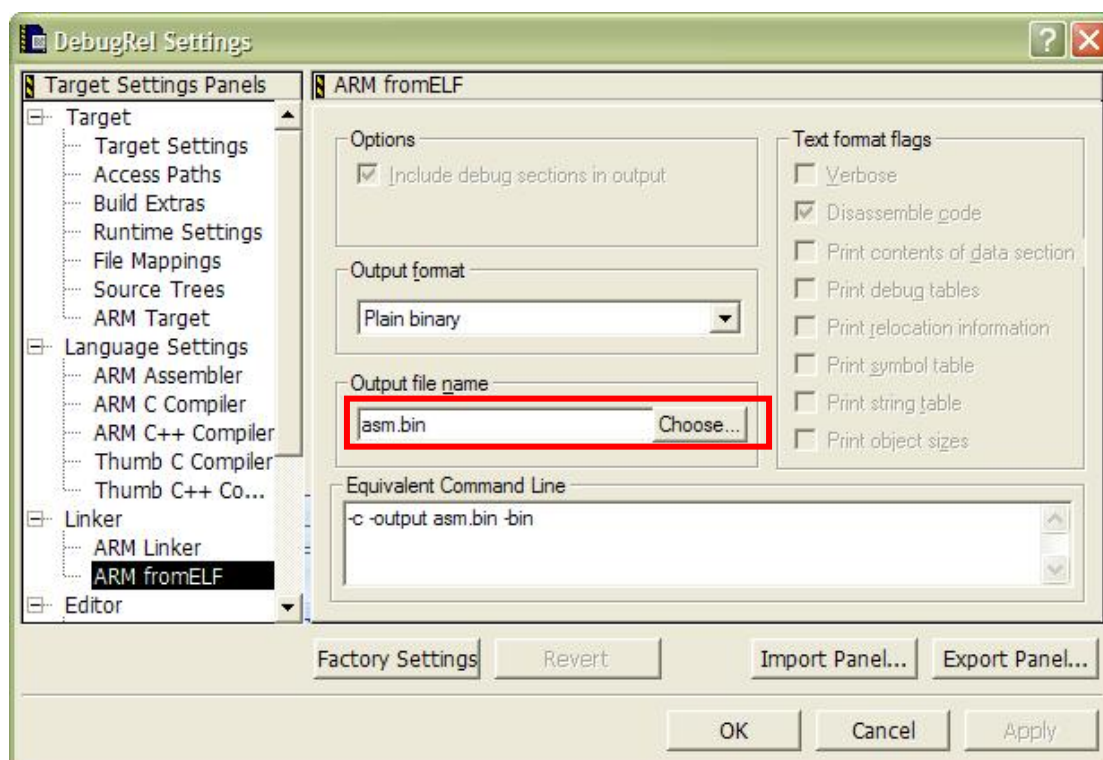


图 1-10 输出文件名设置

4. 编译、链接

当设置完成后，在建立好的文件中输入实验汇编源代码后，选择 **Project → Make** 菜单项（或者单击工程项目窗口中的 **Make** 按钮）即可完成编译、链接，生成目标映像文件，编译结果如下图 1-11、1-12 所示。

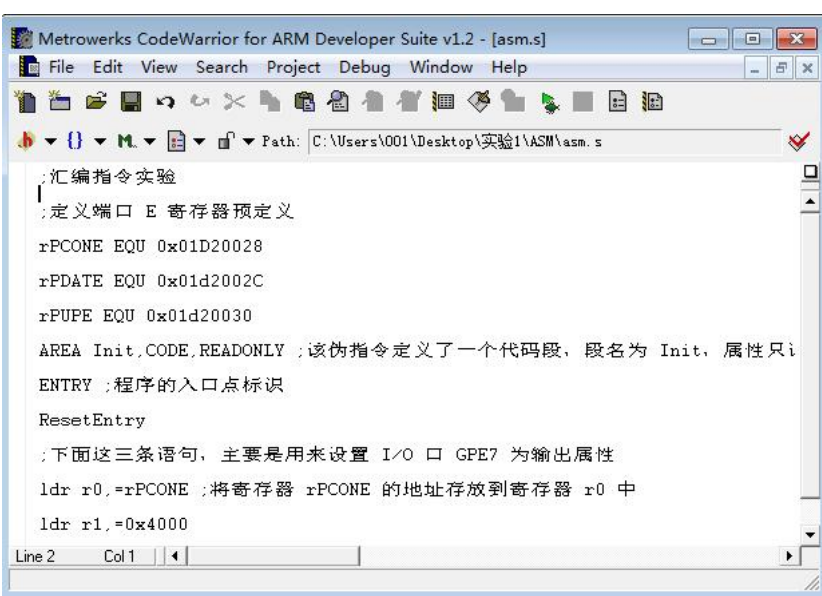


图 1-11 输入代码

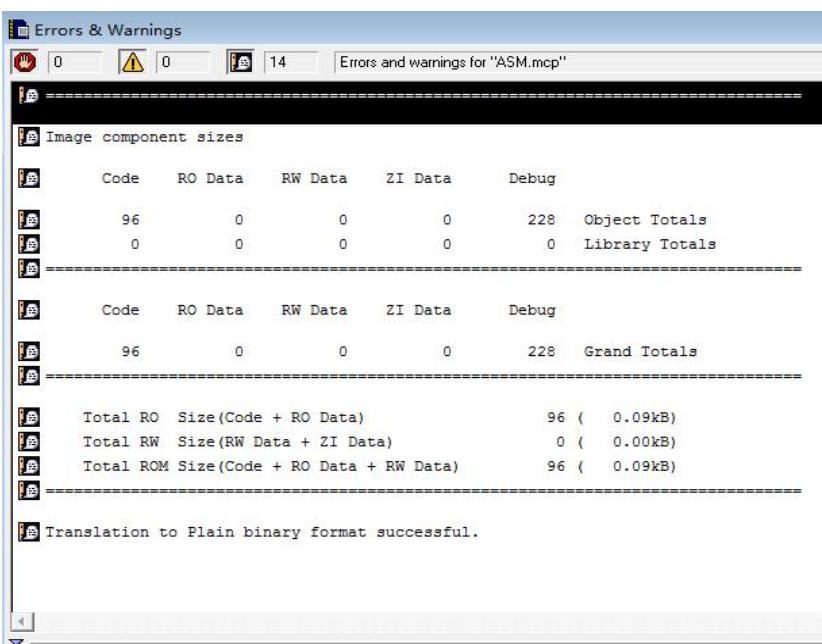


图 1-12 编译连接输出

5. AXD 仿真调试

选择“Project→Debug”，启动 AXD 进行软件仿真调试，如图 1-13、1-14 所示。打开寄存器窗口（Processor view）。选择“Current”项监视 R0、R1 和 R2 的值。打开存储器观察窗口（Memory），设置观察地址 0x56000010（地址不唯一，视具体情况而定），显示方式 Sizewei32Bit，监视其值的变化。

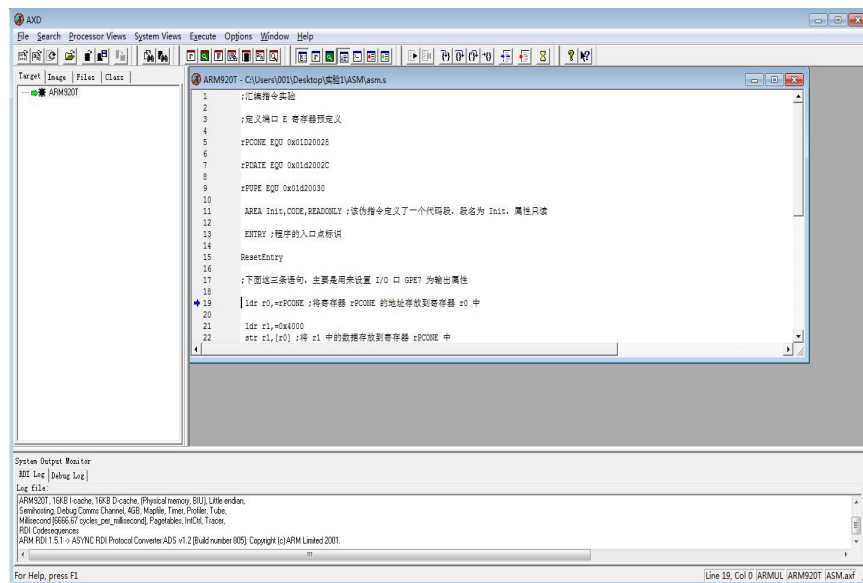


图 1-13 仿真调试加载代码

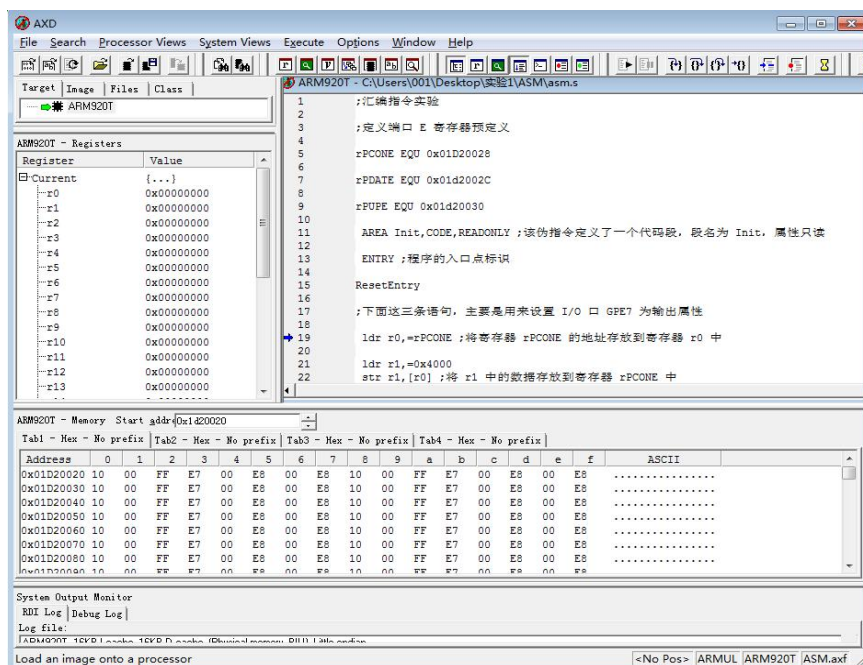
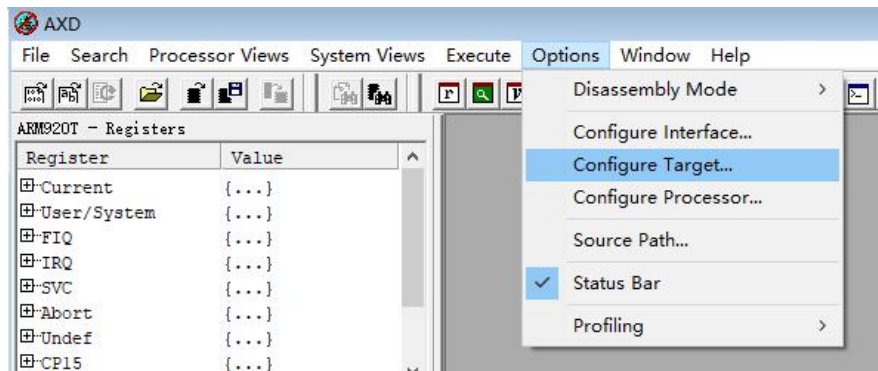
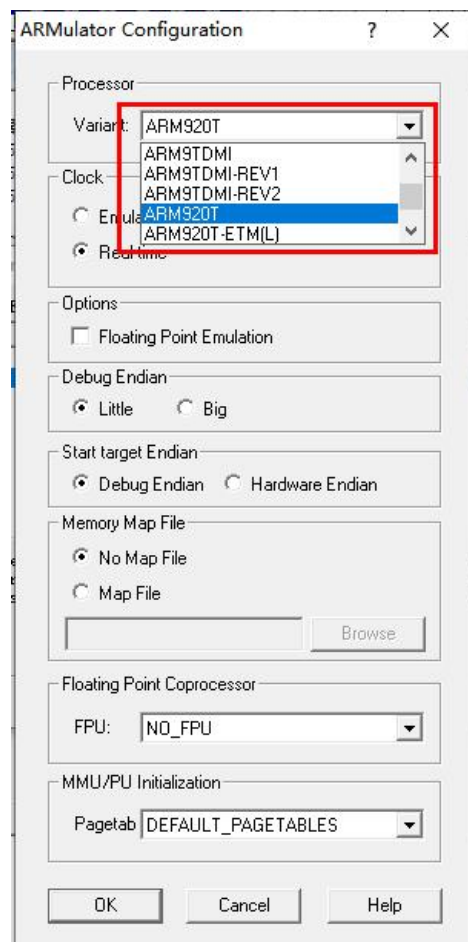
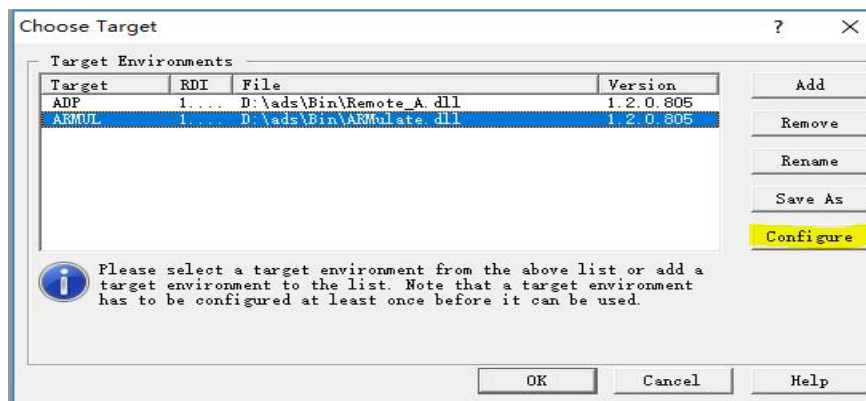


图 1-14 运行程序并观察

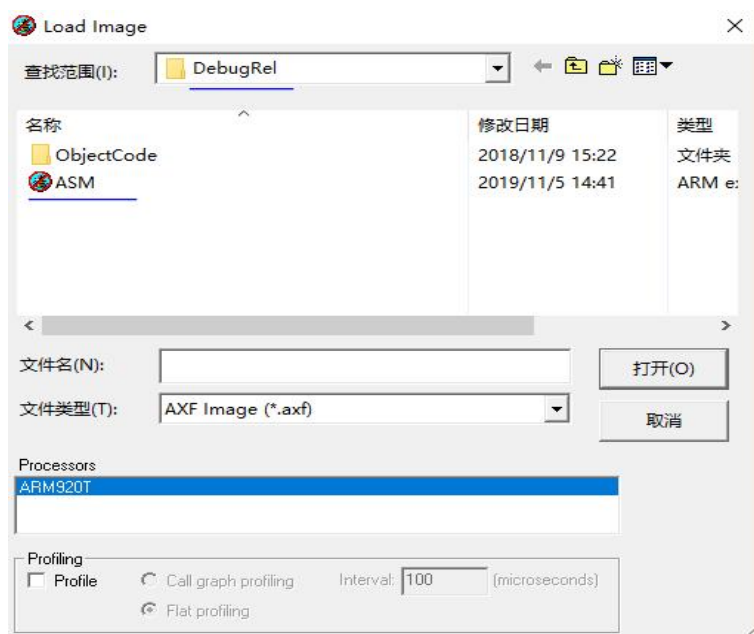
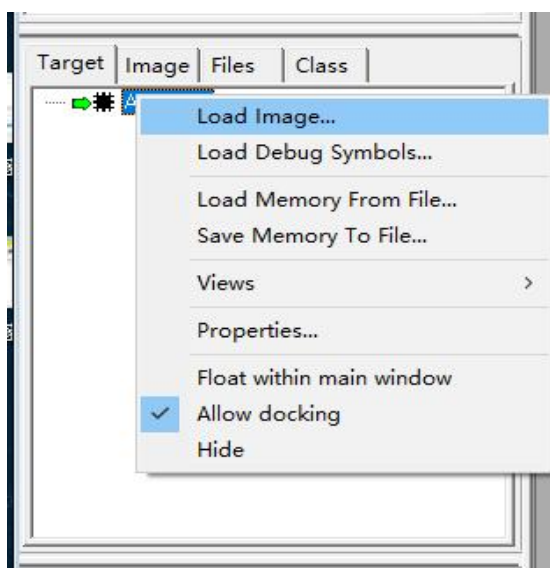
1) 打开 options



2) 配置环境，选择 Configure-->ARM920T

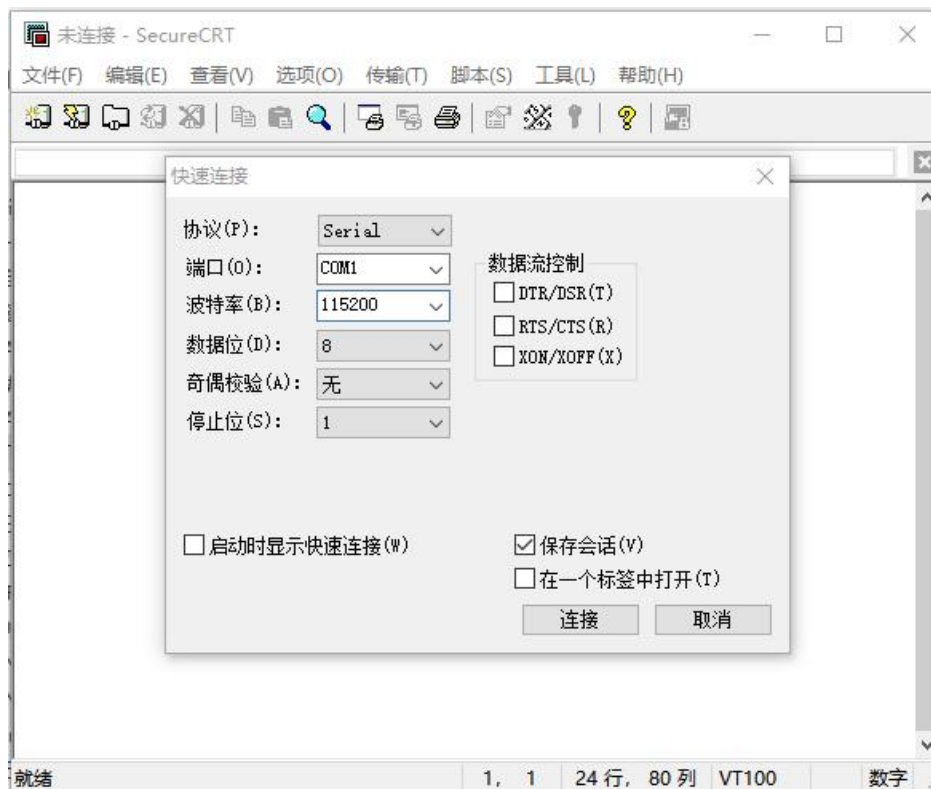


3) 加载调试映像文件



6. 下载可执行程序到开发板

- 1) 硬件连接：将 PC 机串口线连接到开发板串口，连接 USB 下载线到开发板。开发板右上角有启动模式开关，把开关拨到 NOR 选项，选择从 Nor Flash 启动。确认已连接好开发板电源、串口线、USB 下载线。
- 2) 运行串口超级终端（推荐 SecureCRT），选择正确的串口号，并将串口设置如下：波特率（115200）、奇偶校验（none）、数据位（8）和停止位数（1）、无流控，打开串口，不选择选项“RTS/CTS(R)”，如下图所示。



3) 打开开发板左下角的电源开关，可以在超级终端中看到如下信息：

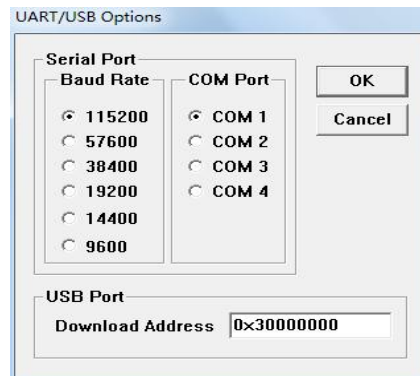
```
##### Boot for Nor Flash Main Menu #####
##### Embedsky USB download mode #####

[1] Download u-boot or STEPLDR.nb1 or other bootloader to Nand Flash
[2] Download Eboot (eboot.nb0) to Nand Flash
[3] Download Linux Kernel (zImage.bin) to Nand Flash
[4] Download CRAMES image to Nand Flash
[6] Download YAFFS image (root.bin) to Nand Flash
[7] Download Program (uCOS-II or TQ2440_Test) to SDRAM and Run it
[8] Boot the system
[9] Format the Nand Flash
[0] Set the boot parameters
[a] Download User Program (eg: uCOS-II or TQ2440_Test)
[b] Download LOGO Picture (.bin) to Nand Flash
[l] Set LCD Parameters
[n] Enter TFTP download mode menu
[o] Download u-boot to Nor Flash
[r] Reboot u-boot
[t] Test Linux Image (zImage)
[q] quit from menu
Enter your selection:
```

在超级终端中输入 7，选择下载测试程序到 SDRAM 并执行。如 USB 下载线连接正常打印信息如下：

```
[7] Download Program (uCOS-II or TQ2440_Test) to SDRAM and Run it
[8] Boot the system
[9] Format the Nand Flash
[0] Set the boot parameters
[a] Download User Program (eg: uCOS-II or TQ2440_Test)
[b] Download LOGO Picture (.bin) to Nand Flash
[l] Set LCD Parameters
[n] Enter TFTP download mode menu
[o] Download u-boot to Nor Flash
[r] Reboot u-boot
[t] Test Linux Image (zImage)
[q] quit from menu
Enter your selection: 7
USB host is connected. waiting a download.
```

4) 使用 USB 口下载程序：打开 DNW，菜单栏中 Configuration--->Option:



打开 DNW 选择 USB Port--->Transmit/Restore, 打开文件选择对话框, 选择刚编译好的可执行文件 ASM.bin。下载完成后, 超级终端打印信息如下:

```
Enter your selection: 7
USB host is connected. waiting a download.

Now, Downloading [ADDRESS:30000000h,TOTAL:106]
RECEIVED FILE SIZE: 106 (0KB/S, 1S)
## starting application at 0x30000000 ...
```

此时程序已经下载到 SDRAM 中地址为 0x30000000 处, 并从地址 0x30000000 处开始运行。

按上述流程操作后, 观察 LED 是否点亮。

实验二 ARM 汇编程序的设计

实验目的:	掌握 ARM 汇编编程
	掌握 ARM 的 GPIO 口的使用
	理解 TQ2440 电路板中 4 个 LED 灯的 GPIO 连接。
硬件连接:	开发板三线连接（电源线、串口线、USB 线），Nor Flash 启动
实验要求:	结合 TQ2440 开发板底板原理图及所给部分代码，实现 4 个 LED 灯二进制流水灯点亮。
实验内容:	1.使用 ADS 编写代码
	2.使用 ADS 编译生成可执行程序
	3.下载可执行程序到开发板，观察实验结果

开发板上 LED 灯的连接原理图如图 2-1 和 2-2 所示。

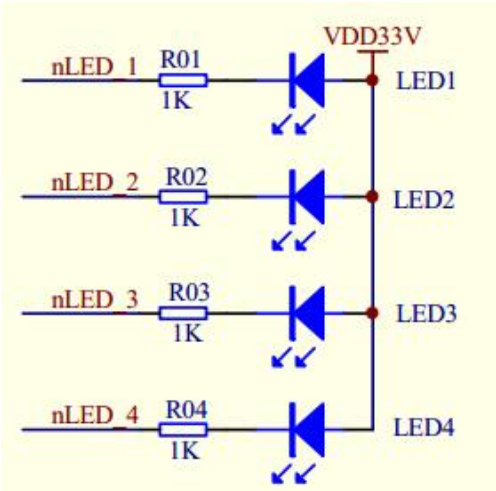


图 2-1：LED 灯连接原理图

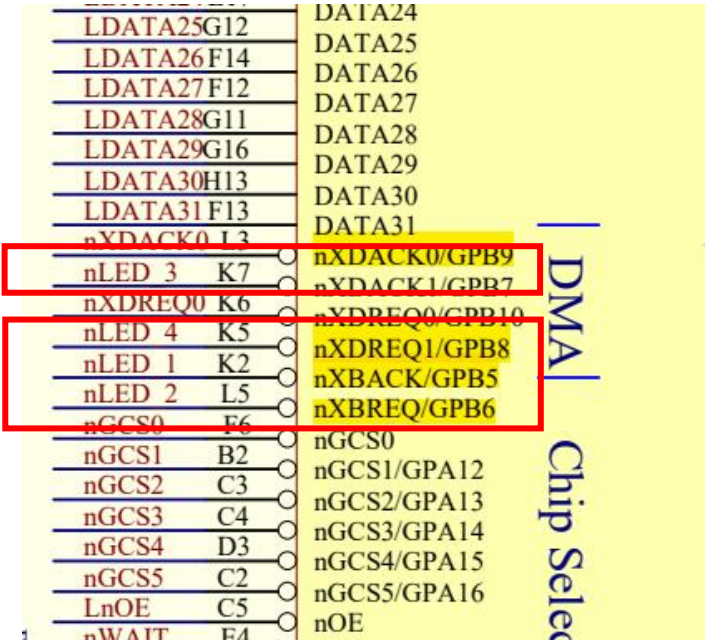


图 2-2 LED 灯 2440 端连接原理图

实验三 ARM 汇编与 C 混合编程

实验目的:	掌握 ARM 汇编与 C 语言程序的混合编程
	理解 TQ2440 电路板中 4 个 LED 灯的 GPIO 连接。
	理解 TQ2440 电路板中 4 个开关的 GPIO 连接。
硬件连接:	开发板三线连接（电源线、串口线、USB 线），Nor Flash 启动
实验要求:	结合 TQ2440 开发板底板原理图及所给部分代码，实现 4 个按键控制 4 个 LED 灯的亮灭，本实验采用查询的方式实现开关状态的检测。
实验内容:	1.使用 ADS 编写代码
	2.使用 ADS 编译生成可执行程序
	3.下载可执行程序到开发板，观察实验结果

开发板上 LED 灯和开关的连接原理图如图 2-1，2-2，2-3 和 2-4 所示。

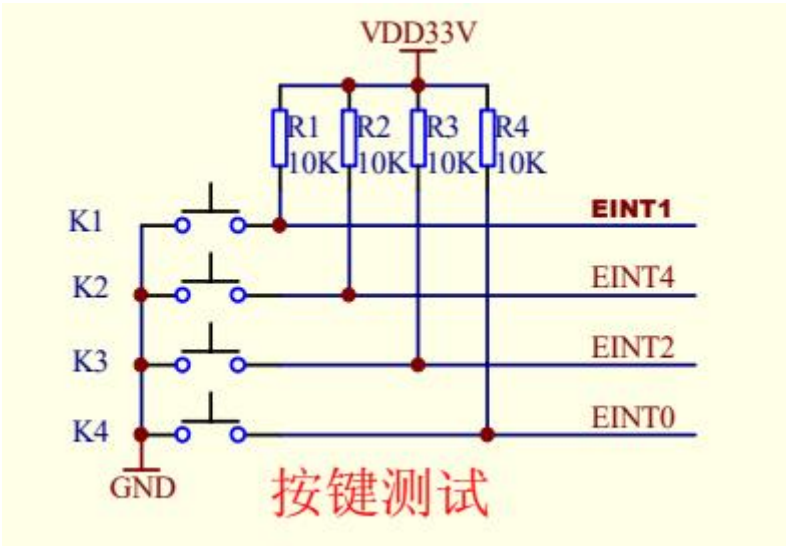


图 2-3 开关连接原理图

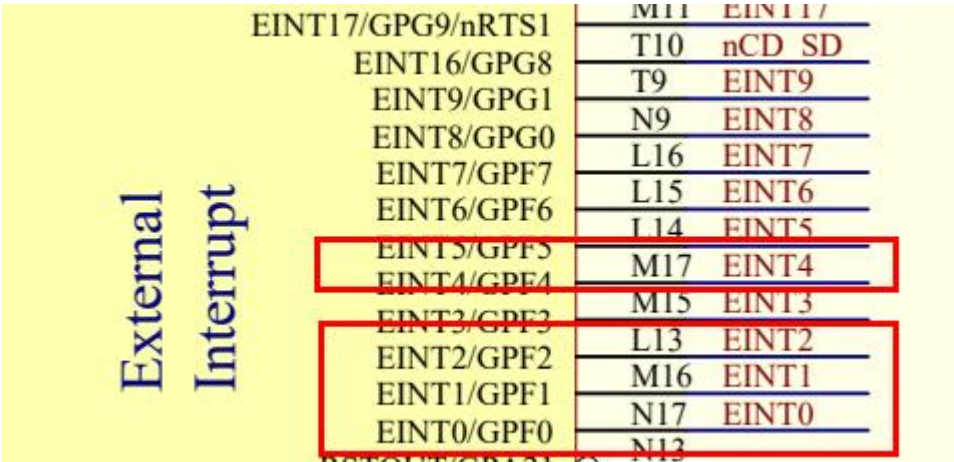


图 2-4 开关在 2440 端连接图

实验四 ARM 中断实验

实验目的：	掌握 ARM 的中断控制器的原理及使用
	理解 TQ2440 电路板中 4 个 LED 灯的 GPIO 连接。
	理解 TQ2440 电路板中 4 个开关的 GPIO 连接。
硬件连接：	开发板三线连接（电源线、串口线、USB 线），Nor Flash 下载程序，Nand Flash 启动运行程序。
实验要求：	结合 TQ2440 开发板底板原理图及所给部分代码，实现 4 个按键分别控制 4 个 LED 灯的亮灭，本实验采用中断的方式实现开关状态的检测。
实验内容：	1.使用 ADS 编写代码
	2.使用 ADS 编译生成可执行程序
	3.下载可执行程序到开发板，观察实验结果

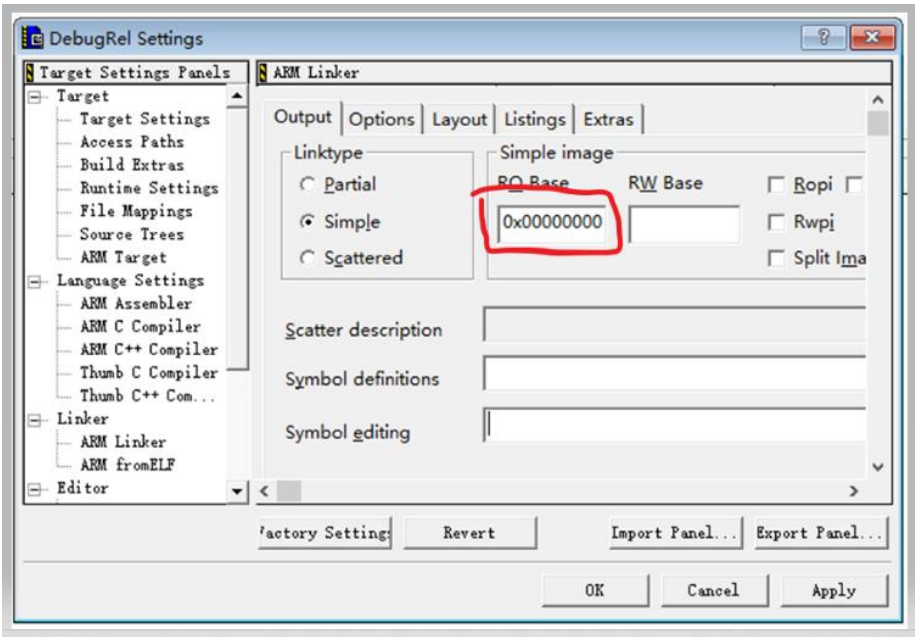
开发板上 LED 灯和开关的连接原理图如图 2-1，2-2，2-3 和 2-4 所示。

1. 开发板硬件连接

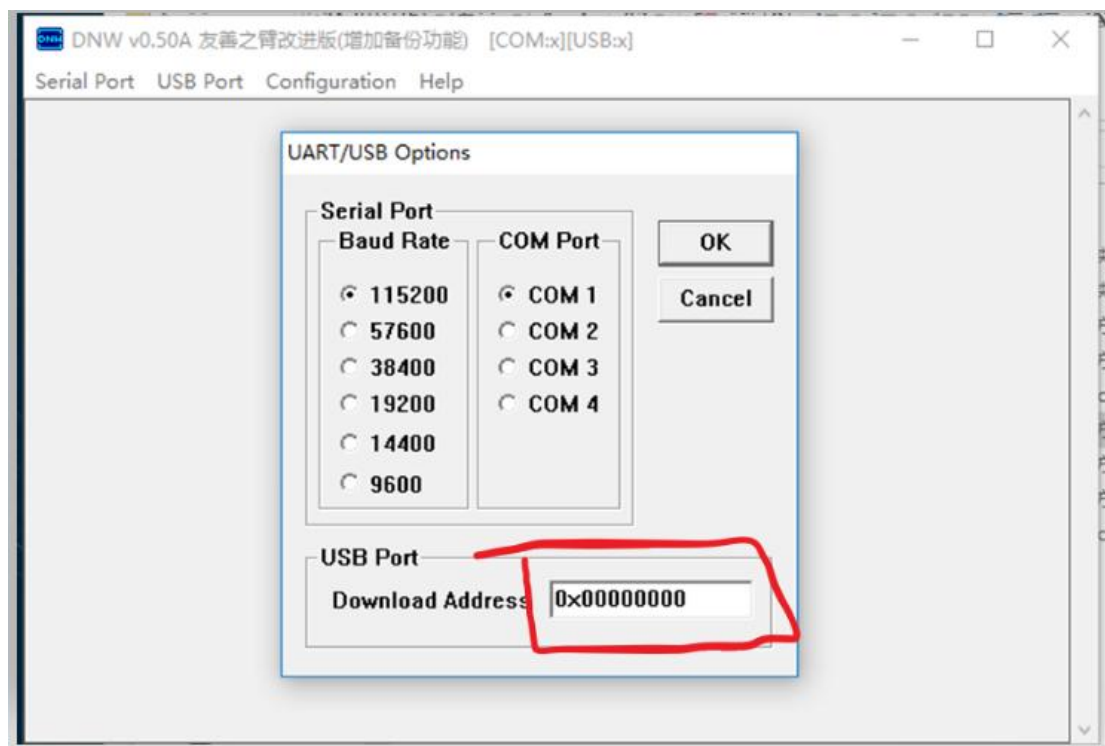
将 PC 机串口线连接到开发板串口，连接 USB 下载线到开发板 USB。启动模式开关拨到 NOR 选项，选择从 Nor Flash 启动。确认已连接好开发板电源、串口线、USB 下载线。

2. 修改参数

修改 RO Base 值为 0x00000000。



在 DNW 中设置 Download Address 为 0x00000000。



3. Nor Flash 启动模式下载程序

运行串口超级终端，打开开发板左下角的电源开关，可以在超级终端中看到如下信息：

```
##### Boot for Nor Flash Main Menu #####
##### EmbedSky USB download mode #####

[1] Download u-boot or STEPLDR.nb1 or other bootloader to Nand Flash
[2] Download Eboot (eboot.nb0) to Nand Flash
[3] Download Linux kernel (zImage.bin) to Nand Flash
[5] Download CRAMFS image to Nand Flash
[6] Download YAFFS image (root.bin) to Nand Flash
[7] Download Program (uCOS-II or TQ2440_Test) to SDRAM and Run it
[8] Boot the system
[9] Format the Nand Flash
[0] Set the boot parameters
[a] Download User Program (eg: uCOS-II or TQ2440_Test)
[b] Download Logo picture (.bin) to Nand Flash
[l] Set LCD Parameters
[n] Enter TFTP download mode menu
[o] Download u-boot to Nor Flash
[r] Reboot u-boot
[t] Test Linux Image (zImage)
[q] quit from menu
Enter your selection:
```

在超级终端中输入 a，选择下载测试程序到 Nand Flash 中。

```
[a] Download User Program (eg: uCOS-II or TQ2440_Test)
[b] Download LOGO Picture (.bin) to Nand Flash
[l] Set LCD Parameters
[n] Enter TFTP download mode menu
[o] Download u-boot to Nor Flash
[r] Reboot u-boot
[t] Test Linux Image (zImage)
[q] quit from menu
Enter your selection: a
USB host is connected. waiting a download.
```

使用 USB 口下载程序 DNW，配置好参数，选择 USB Port--->Transmit/Restore，下载刚编译好的可执行文件。下载完成后，超级终端打印信息如下：

```
Enter your selection: a
USB host is connected. waiting a download.

Now, Downloading [ADDRESS:30000000h,TOTAL:546]
RECEIVED FILE SIZE:      546 (0KB/S, 1S)

NAND erase: device 0 offset 0x0, size 0x20000
Erasing at 0x0 -- 100% complete.
OK

NAND write: device 0 offset 0x0, size 0x20000

Writing data at 0x1f800 -- 100% complete.
131072 bytes written: OK
```

4. Nand Flash 模式下运行程序

关闭开发板电源，将启动模式更改至“Nand Flash”启动模式，再上电并运行程序，按键控制 LED 灯，观察实验结果。

第二篇 Linux 环境下裸机开发实验

本章实验是在 Linux 操作系统下，采用文本编辑器、代码编译器和工程管理等开发工具，实现开发板裸机（开发板上无操作系统）程序的开发。本章实验涉及到 S3C2440 的 GPIO 模块。

实验五 熟悉 Linux 操作系统

实验目的：	熟悉 Linux 操作系统环境的搭建
硬件连接：	无
实验要求：	了解虚拟机+Linux 操作系统的环境搭建
实验内容：	1. （安装）使用虚拟机
	2. 虚拟机下（安装）启动 Linux 操作系统

1. 打开虚拟机

打开虚拟机 VMware，界面如下图 5-1 所示，选择要运行的操作系统，本实验中选择的 Linux 系统为 Red Hat 操作系统，打击绿色箭头按钮开启此操作系统。

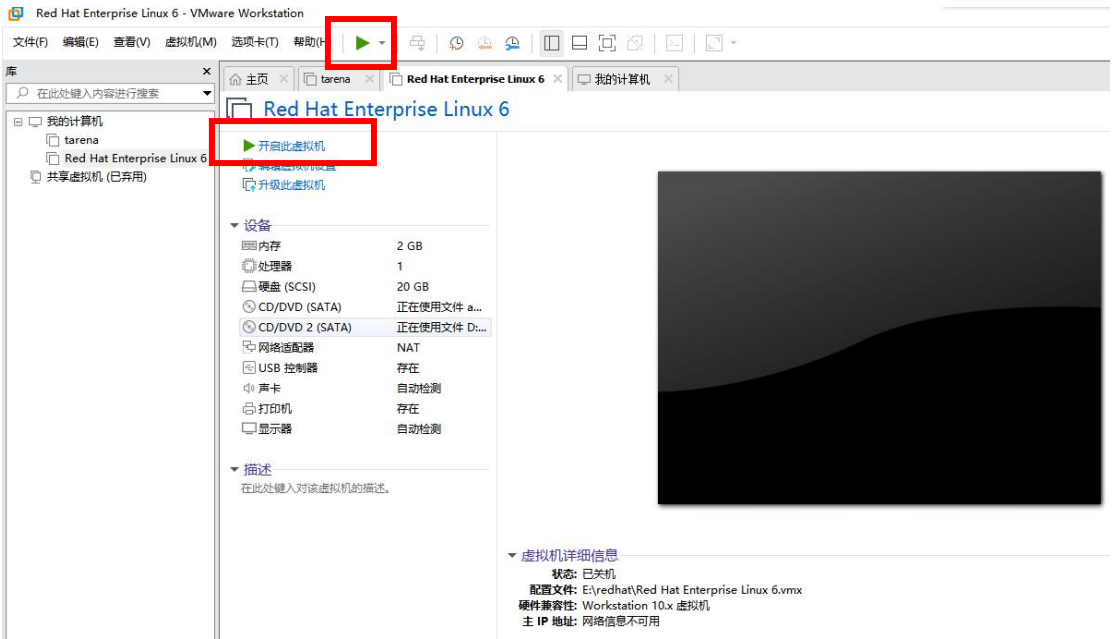


图 5-1：VMware 虚拟机界面

2. 登录 Linux 操作系统

进入登陆界面后如图 5-2 所示，选择 Other，输入用户名：`root`，密码为 1。登录后，会出现警告提示，提示是以特权用户的身份登录，选择继续即可。



图 5-2 Red Hat 登录界面

进入 Linux 系统桌面环境如图 5-3 所示

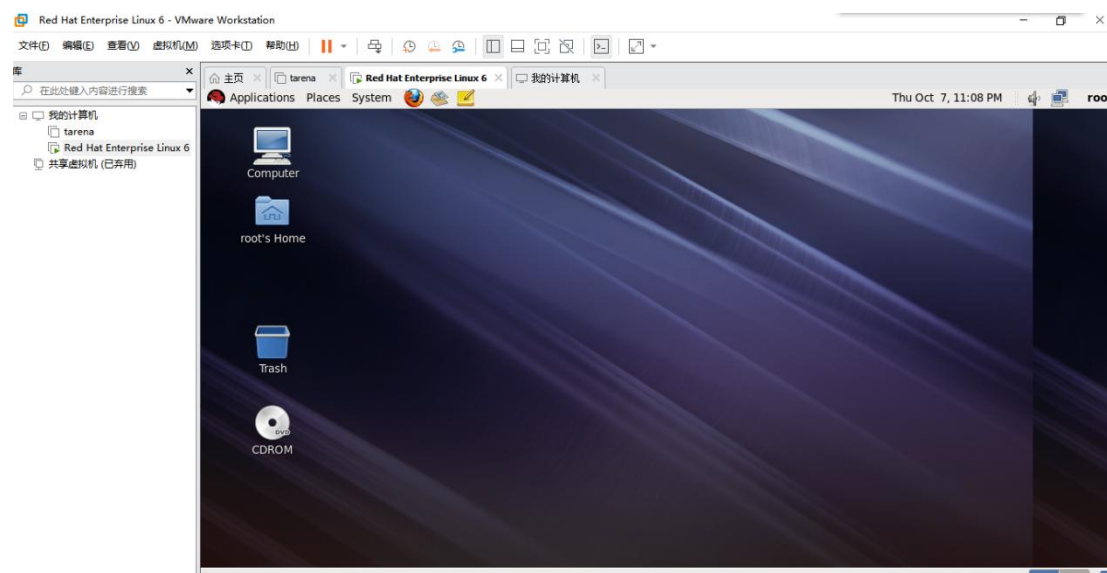


图 5-3 Red Hat 桌面环境

3. 打开命令终端 (Terminal)

打开命令终端：与 Windows 不同，在 Linux 中的工作大多是在 Linux 终端中用命令

完成的。单击 Applications->System Tools->Terminal，打开命令行终端，如图 5-4 所示：

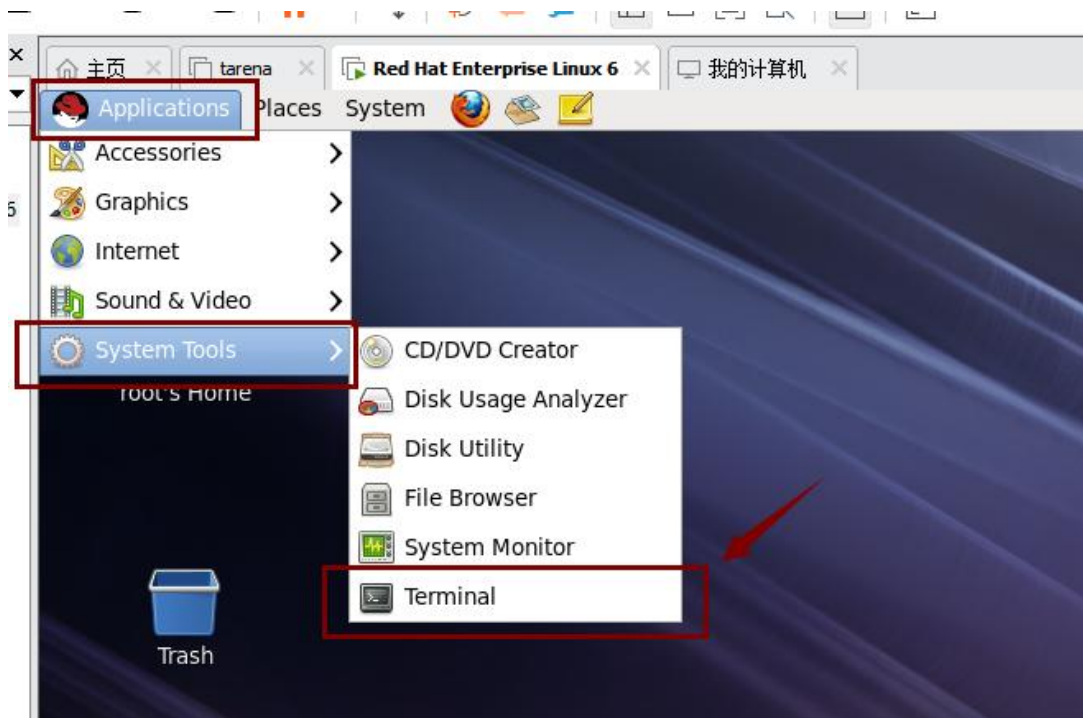


图 5-4 打开命令终端

试着在 Linux 终端中写入第一个命令 `date`，回车后，终端显示出当前的日期和时间如图 5-5 所示。

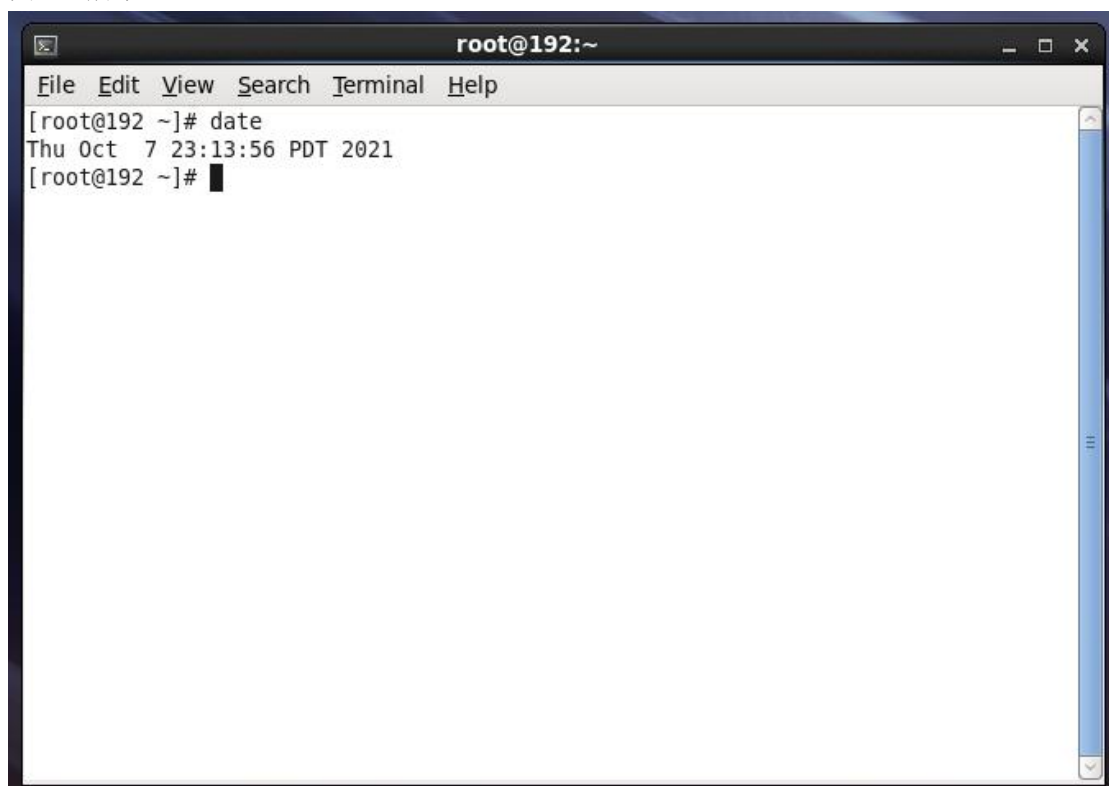


图 5-5 命令行输入命令

注意：虚拟机已安装 VMwareTools，可以在 windows 系统和虚拟机之间实现文件拖拽功能。

试着把 windows 系统下的一个文件拖拽到虚拟机的 Linux 系统中。在本实验后面的小节中，都以这种方式实现两个系统之间的文件复制。

实验六 Linux 常用命令的熟悉

实验目的:	熟练 Linux 的常用命令
硬件连接:	无
实验要求:	掌握常用命令的使用
实验内容:	1. pwd 显示工作目录
	2. cd 切换工作目录
	3. ls 列出目录内容
	4. touch 创建文件
	5. mkdir 创建目录
	6. cp 复制文件或目录
	7. mv 移动或更名现有的文件或目录
	8. rm 删除文件或目录

Linux 命令行的一般格式为:

命令名 [选择项] [参数]

注意: Linux 终端下的命令区分大小写, 而 windows 系统命令行不区分大小写。

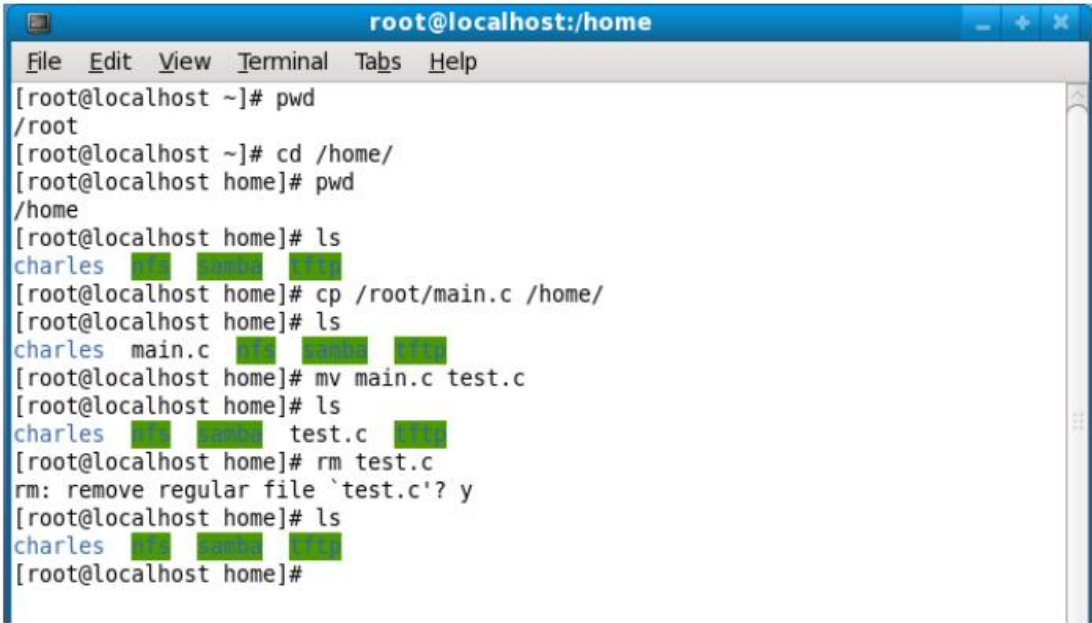


图 6-1 在命令终端输入命令

实验七 文本编辑器 Vi 的熟悉

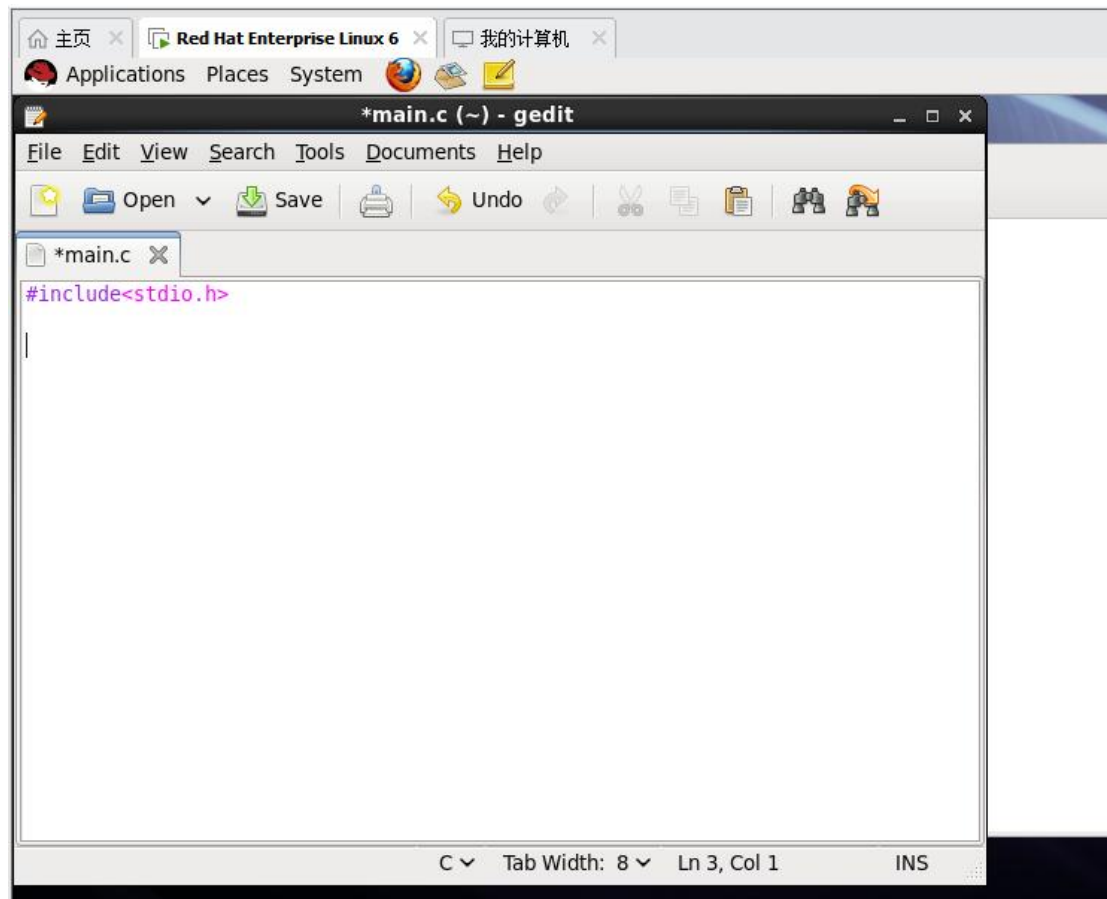
实验目的:	掌握 Vi 和 gedit 编辑器的使用
硬件连接:	无
实验要求:	熟练掌握 Vi 的使用，三种模式的切换，常用快捷键； 熟练掌握 gedit 的使用。

1. gedit 编辑器

1) Linux 下的 gedit 类似 windows 下文本编辑器，使用方法简单，用 gedit 编辑，执行命令：

```
#gedit main.c
```

gedit 界面如图 7-1 所示。



2) 输入 main.c 内容如下：

```
#include <stdio.h>
```

```
int main(int argc, char** argv)
```

```
{
```

```
    int i;
```

```

printf("Print the external argument of the main function:\n");
for(i=0;i<argc;i++)
    printf("arg[%d]=%s\n",i+1,argv[i]);
return 0;
}

```

2. Vi 编辑器

1) Linux 下 Vi 的是命令行下的文本编辑器，在串口终端的嵌入式 Linux 系统中经常会用到，功能强大。

2) Vi 有 3 种模式，分别为命令行模式、插入模式及底行模式。各模式的功能及切换如下：

①命令行模式：用户在使用 vi 编辑文件时，最初进入的模式。在该模式中可以通过上下移动光标进行删除符或整行删除等操作，也可以进行复制、粘贴等操作，但无法编辑文件。用户可按 a、o、i 键进入插入模式。按冒号进入底行模式。

②插入模式：只用在该模式下，用户才能进行文字编辑输入，用户可按 ESC 键返回命令行模式。

③底行模式：在命令行模式下，用户按冒号可进入底行模式。此时光标位于屏幕的底行。用户可以进行保存或退出操作，也可以设置编辑环境，如寻找字符串、列出行号等。

3) 下面几个表分类列出了 vi 下常用的命令。

编辑命令：

命令	功能
[N] x	(Expurgate)删除从光标位置开始的连续 N 个字符（并复制到编辑缓冲区）
[N] dd	(Delete)删除从光标位置开始的连续 N 个行（并复制到编辑缓冲区）
[N] yy	(Yank)复制从光标位置开始的连续 N 行到编辑缓冲区
p	(Put)从编辑缓冲区复制文本到当前光标位置（即粘贴）
u	(Undo)取消上一次操作（即恢复功能）

光标命令：

命令	功能
h	方向键，向左移动光标一个字符的位置，相当于键“←”

j	方向键，向下移动光标到下一行的位置，相当于键“↓”
k	方向键，向上移动光标到上一行的位置，相当于键“↑”
l	方向键，向右移动光标一个字符的位置，相当于键“→”
:N	移动光标到第 N 行（N 待定）
1G	移动光标到文件的第 1 行
G	移动光标到文件的最后 1 行
:set nu	设置显示行号
:set nonu	取消显示行号

文件命令：

命令	功能
:q	(Quit)退出没有被修改的文件 (若文件被修改了而没有保存，则此命令无效)
:q!	强制退出，且不保存修改过的部分
:w	(Write)保存文件，但不退出
:w!	强制保存文件，但不退出
:x	(Exit)保存文件并退出
:x!	强制保存文件并退出
:w File	另存为 File 给出的文件名，不退出
:w! File	强制另存为 File 给出的文件名，不退出
:r File	(Read)读入 File 指定的文件内容插入到光标位置

状态命令：

命令	功能
a	(Append)进入编辑状态，从当前光标之后的位置开始插入键盘输入的字符。

i	(Insert)进入编辑状态，从当前光标之后的位置开始插入键盘输入的字符。
o	(Open)进入编辑状态，并插入一新行，光标移到该新行的行首，以后键盘输入的字符将插入到光标位置。
ESC	进入命令状态。
:! command	在 vi 中执行外部命令 command ，按回车键可以返回继续工作。

4) 使用 vi 编辑 main.c 源文件，在终端执行如下命令：

```
#vi main.c
```

刚启动时 vi 处于“命令模式”，如图 7-2 所示，此时不能输入文本，可执行命令。

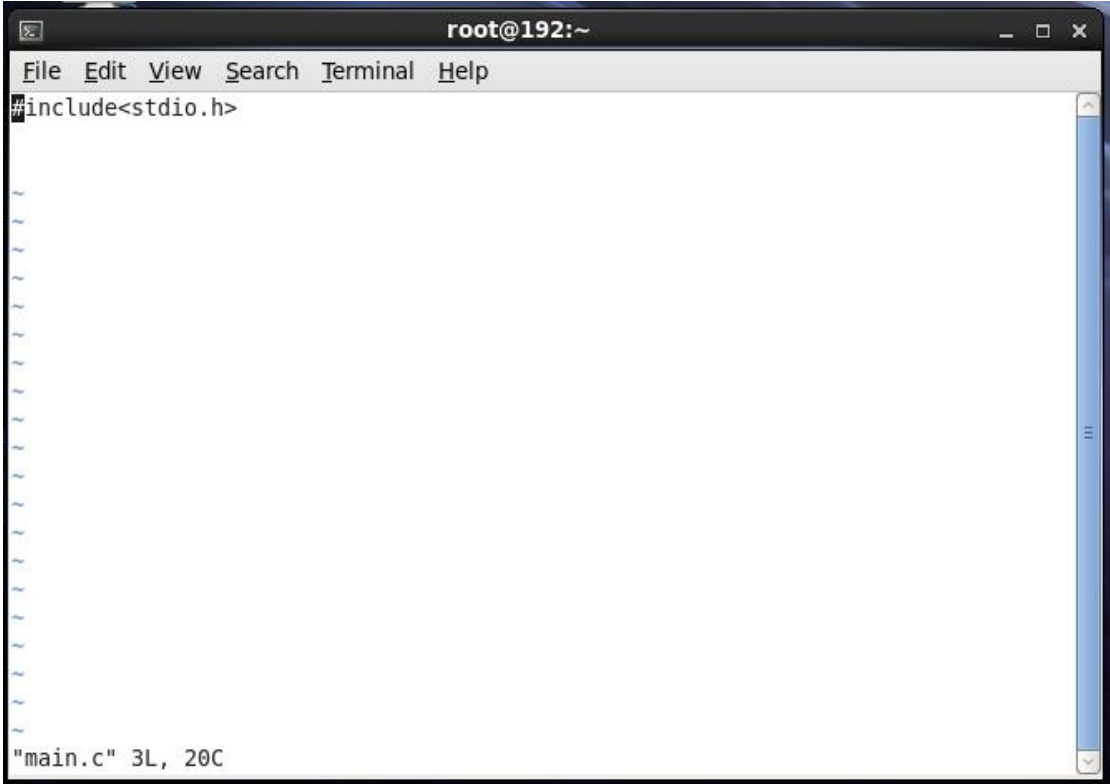


图 7-2 vi 命令行模式

按 a, o, i 键切换到插入模式，底部出现“INSERT”，如图 7-3 所示，此时可输入代码。



图 7-3 vi 插入模式

当输入代码完毕，可按 ESC 键返回命令模式，再按“:”进入“底行模式”，输入 wq 保存退出，如图 7-4 所示。

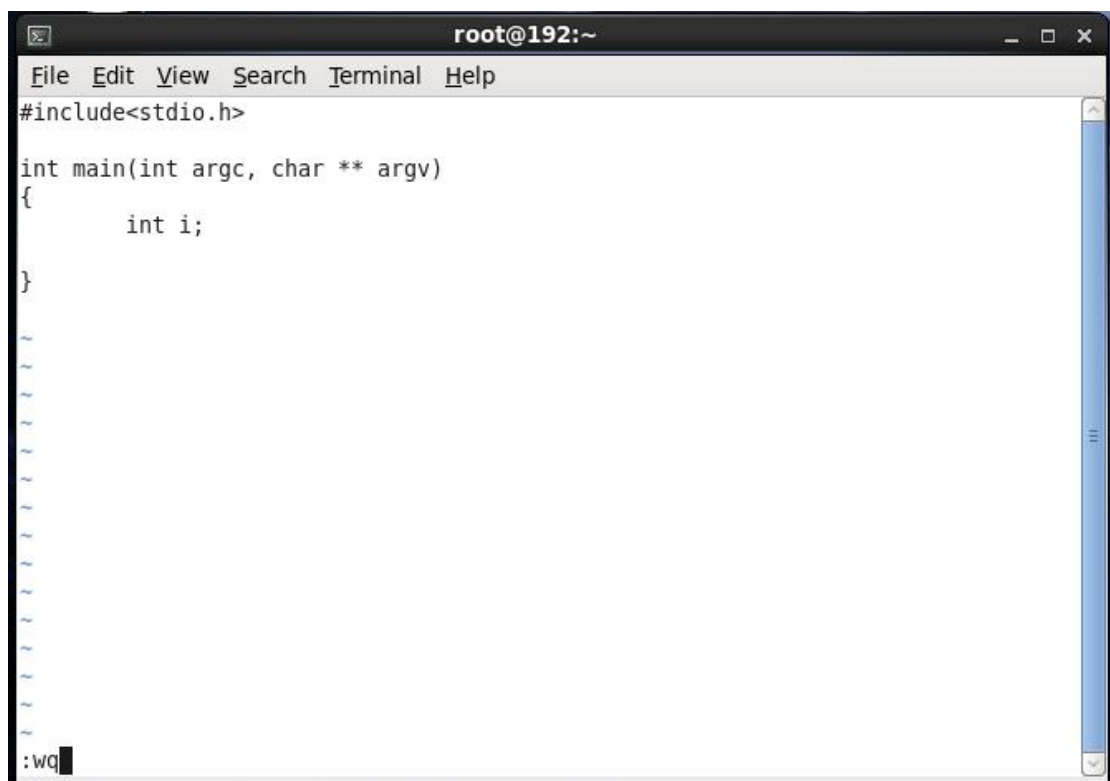


图 7-4 vi 底行模式

实验八 编译工具 GCC 的使用

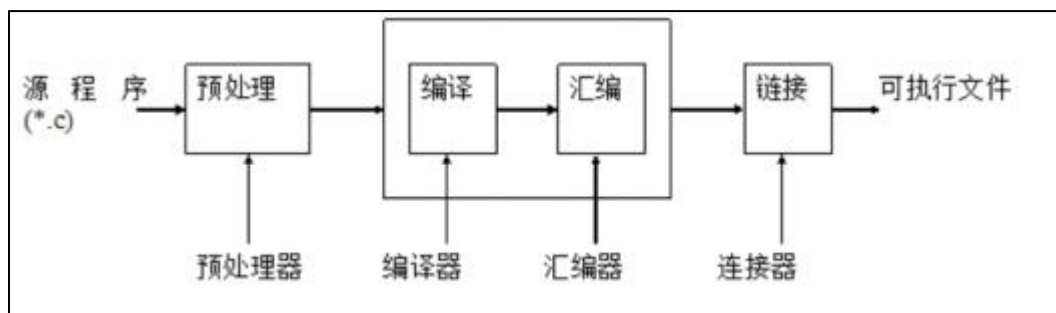
实验目的:	掌握编译工具 gcc 的使用
硬件连接:	无
实验要求:	使用 gcc 编译器编译代码并执行。

1) 一般来说,系统安装后就已经安装和设定好了 gcc。在 shell 的提示符(即命令行终端)下键入 gcc-v, 屏幕上就会显示出目前正在使用的 gcc 的版本信息。



```
root@192:~  
File Edit View Search Terminal Help  
[root@192 ~]# date  
Thu Oct 7 23:13:56 PDT 2021  
[root@192 ~]# gedit main.c  
[root@192 ~]# vi main.c  
[root@192 ~]# gcc -v  
Using built-in specs.  
Target: i686-redhat-linux  
Configured with: ../configure --prefix=/usr --mandir=/usr/share/man --infodir=/usr/share/info --with-bugurl=http://bugzilla.redhat.com/bugzilla --enable-bootstrap --enable-shared --enable-threads=posix --enable-checking=release --with-system-zlib --enable-__cxa_atexit --disable-libunwind-exceptions --enable-gnu-unique-object --enable-languages=c,c++,objc,obj-c++,java,fortran,ada --enable-java-awt=gtk --disable-dssi --with-java-home=/usr/lib/jvm/java-1.5.0-gcj-1.5.0.0/jre --enable-libgcj-multifile --enable-java-maintainer-mode --with-ecj-jar=/usr/share/java/eclipse-ecj.jar --disable-libjava-multilib --with-ppl --with-cloog --with-tune=generic --with-arch=i686 --build=i686-redhat-linux  
Thread model: posix  
gcc version 4.4.6 20120305 (Red Hat 4.4.6-4) (GCC)  
[root@192 ~]#
```

2) gcc 可以使程序员灵活地控制编译过程。编译过程一般可以分为下面四个阶段,每个阶段分别调用不同的工具进行处理,如下图所示。



Linux 系统中可执行文件有两种格式。第一种格式是 a.out 格式,这种格式用于早期的 Linux 系统以及 Unix 系统的原始格式。a.out 来自于 Unix C 编译程序默认的可执行文件名。当使用共享库时,a.out 格式就会发生问题。第二种新的文件格式被称为可执行和连接的格式(ELF)。这种格式很容易实现共享库。本 Linux 系统 gcc 产生的是*.elf 可执行文件。

3) gcc 的使用格式如下:

```
#gcc [options] [filenames]
```

其中 `filenames` 为所要编译的程序源文件。

当使用 `gcc` 时, `gcc` 会完成预处理、编译、汇编和连接。前三步分别生成目标文件, 连接时, 把生成的目标文件链接成可执行文件。`gcc` 可以针对支持不同的源程序文件进行不同处理, 文件格式以文件的后缀来识别。

`gcc` 总体选项列表如下所示:

选项	说明
-c	只是编译不连接, 生成目标文件“.o”
-S	只是编译不汇编, 生成汇编代码
-E	只进行预编译, 不做其他处理
-g	在可执行程序中包含标准调试信息
-o file	把输出文件输出到 <code>file</code> 里
-v	打印出编译器内部编译各过程的命令行信息和编译器的版本
-I dir	在头文件的搜索路径列表中添加 <code>dir</code> 目录
-L dir	在库文件的搜索路径列表中添加 <code>dir</code> 目录
-static	链接静态库
-llibrary	连接名为 <code>library</code> 的库文件

使用 `gcc` 把上一个实验编辑的 `main.c` 源文件编译为可执行文件并执行。

确认 `main.c` 在当前工作目录下, 执行以下命令:

```
#gcc -o test main.c
```

执行 `ls`, 查看当前目录, 发现新增一个文件 `test`。执行命令

```
#file test
```

可以查看 `test` 文件详细信息。执行可执行文件 `test`:

```
#./test 123 456 abc def ghi
```

注意: `./test` 是运行 `test` 可执行文件的命令(也算一个参数), 后面的 `123 456 abc def ghi` 是传递给 `test` 的 5 个参数, 在本测试程序中, 这些参数可以任意给定。

运行结果如下:


```
[root@localhost ~]# ./test 123 456 abc def ghi
```

Print the external arguments of the main function:

```
arg[1]=./test
```

```
arg[2]=123
```

```
arg[3]=456
```

```
arg[4]=abc
```

```
arg[5]=def
```

```
arg[6]=ghi
```

实验九 交叉编译工具的使用

实验目的:	熟悉交叉编译工具 arm-linux-gcc 的安装和使用
硬件连接:	无
实验要求:	在 Linux 系统下安装交叉编译工具; 修改系统环境变量。

1. 交叉编译工具介绍

所谓交叉编译,简单的说,就是在一个平台上生成另一个平台可执行的应用程序。在上一节中使用的 gcc 不是交叉编译工具,因为它是在 PC 机的 Linux 系统(桌面 Linux 系统)下生成本地可执行的应用程序。而 arm-linux-gcc 是在桌面 Linux 下生成 arm 开发板可执行的应用程序。我们之前使用的 ADS 编译源文件,同样也属于交叉编译。

2. 安装交叉编译工具

在 Linux 环境下编译生成 arm 开发板能运行的程序,就需要安装交叉编译工具 arm-linux-gcc。解压缩实验提供的交叉编译工具压缩包,输入解压命令如下:

```
# tar xvfz arm-linux-gcc-4.4.3-20100728.tar.gz -C /
```

注意: -C / 表示解压缩到根目录,否则默认解压缩到当前工作目录。

3. 修改系统环境变量

将交叉编译器目录加入到系统路径。

解压缩后,还不能直接使用,因为系统无法找到 arm-linux-gcc 的可执行程序,需要添加环境变量,修改/etc/profile:

```
# vi /etc/profile
```

在文件最后添加一行(本实验虚拟机中已经添加,若没有则自行添加):

```
export PATH=/opt/FriendlyARM/toolschain/4.4.3/bin/:$PATH
```

其中/opt/FriendlyARM/toolschain/4.4.3/bin/是解压缩后交叉编译工具可执行程序的路径,实验中以实际路径为准。修改后保存文件,执行以下命令使环境变量立即生效:

```
# source /etc/profile
```

查看修改的环境变量是否生效,可以执行命令:

```
# echo $PATH
```

修改的环境变量生效后,安装交叉编译工具的工作就完成了。

查看交叉编译工具的信息,可以执行命令:

```
# arm-linux-gcc -v
```

```
root@192:~  
File Edit View Search Terminal Help  
[root@192 ~]# 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-g  
nu --target=arm-none-linux-gnueabi --prefix=/opt/FriendlyARM/toolschain/4.4.3 --  
with-sysroot=/opt/FriendlyARM/toolschain/4.4.3/arm-none-linux-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-s  
jlj-exceptions --enable-__cxa_atexit --with-gmp=/opt/FriendlyARM/toolschain/4.4.  
3 --with-mpfr=/opt/FriendlyARM/toolschain/4.4.3 --with-ppl=/opt/FriendlyARM/tool  
schain/4.4.3 --with-cloog=/opt/FriendlyARM/toolschain/4.4.3 --with-mpc=/opt/Frie  
ndlyARM/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-sy  
mvers=gnu --enable-c99 --enable-long-long --enable-target-optspace  
Thread model: posix  
gcc version 4.4.3 (ctng-1.6.1)  
[root@192 ~]#
```

实验十 Linux 环境下裸机程序设计及编译

实验目的:	理解 Makefile 规则及掌握 GNU make 工具的使用
硬件连接:	开发板三线连接（电源线、串口线、USB 线），Nor Flash 启动
实验要求:	熟悉掌握交叉编译工具 arm-linux-gcc 的使用，熟练掌握工程管理 GNU make 工具的使用
实验内容:	1. 使用交叉编译工具编译程序；
	2. 编写 makefile 文件
	3. 使用工程管理工作 make 生成可执行程序；
	4. 下载程序到板上运行，并观察 LED 流水灯。

参考实验程序路径： ...\\linux_asm

linux 系统下没有像 ADS 这样的集成开发环境，编译源码的方法是使用交叉编译工具 arm-linux-gcc 和工程管理工作 GUN make 实现的。

1. GUN make 工具介绍

GUN make 是 Linux 程序员用于构建和管理源代码工程的工具。整个工程的编译只需要一个命令就可以完成编译、连接以至于最后的执行。不过这需要我们投入一些时间去完成一个或者多个称之为 Makefile 文件的编写。Makefile 文件描述了整个工程的编译、连接等规则。其中包括：工程中的哪些源文件需要编译以及如何编译、需要创建那些库文件以及如何创建这些库文件、如何最后产生我们想要得可执行文件。

2. Makefile 规则

以上一小节 main.c 为例，为它编写一个 Makefile，内容如下：

```
test:main.c
gcc -o test main.c
clean:
rm -f test
```

这是一个最简单的 Makefile 例子，执行 make 命令即可编译程序，执行 make clean 即可清除编译出来的结果。

make 命令根据文件更新的时间戳来决定哪些文件需要重新编译，这使得可以避免编译已经编译过得、没有变化的程序，可以大大提高编译效率。

一个 Makefile 文件包含一系列的“规则”，规则的格式如下：

```
TARGET... : PREREQUISITES...
COMMAND
```

target: 规则的目标文件。通常是最后需要生成的文件名或者为了实现这个目的而必需

的中间过程文件名。可以是.o文件、也可以是最后的可执行程序的文件名等。另外，目标也可以是一个make执行的动作的名称，如目标“clean”，我们称这样的目标是“伪目标”。

prerequisites: 规则的依赖。生成规则目标所需要的文件名列表。通常一个目标依赖于一个或者多个文件。

command: 规则的命令行。是规则所要执行的动作（任意的shell命令或者是可在shell下执行的程序）。它限定了make执行这条规则时所需要的动作。

一个规则可以有多个命令行，每一条命令占一行。**注意：每一个命令行必须以[Tab]字符开始**，[Tab]字符告诉make此行是一个命令行。make按照命令完成相应的动作。这也是书写Makefile中容易产生，而且比较隐蔽的错误。

想要完整的了解Makefile的规则，可以查看GUN make中文手册.pdf。

3. 编写流水灯 Makefile 管理工程代码

为了更清晰的了解Linux环境与Windows环境下应用程序开发的区别，在这里仍然使用实验二中led流水灯的例子，来说明Linux环境下arm裸机程序的开发流程。

首先需要说明的是，ADS开发环境的arm汇编伪指令与Linux环境下arm GUN汇编伪指令不同（具体参看教科书P85页）。所以不能直接使用实验一中的源代码，在编译前需要修改伪指令。

源码修改完成后，还需要编写Makefile文件，内容如下：

```
objects=asm.elf asm.o asm.bin
asm.bin:asm.elf
    arm-linux-objcopy -O binary -S $< $@
asm.elf:asm.o
    arm-linux-ld -Ttext 0x30000000 -o $@ $^
%.o:%.s
    arm-linux-gcc -c -o $@ $<
clean:
    rm -f $(objects)
```

上述Makefile文件中，“\$@”、“\$<”、“\$^”、“%”称为自动变量。“\$@”表示规则的目标文件名；“\$<”表示第一个依赖的文件名；“\$^”表示所有依赖的名字，名字之间用空格隔开；“%”表示通配符，它和一个字符串中任意个数的字符相匹配。

4. 执行 Makefile 文件编译代码

假设已经把本实验程序文件夹 `linux_asm` 复制到 Linux 系统目 `/home/cuglab` 下，依次执行以下命令：

```
# cd /home/cuglab/linux_asm
```

```
# make
```

make 打印信息如下：

```
arm-linux-gcc -c -o asm.o asm.s
```

```
arm-linux-ld -Ttext 0x30000000 -o asm.elf asm.o
```

```
arm-linux-objcopy -O binary -S asm.elf asm.bin
```

这些打印信息就是执行 `make` 后，真正运行的编译源文件的命令。通过打印信息我们可以看到生成可执行文件 `asm.bin` 的过程。编译完成后，`/home/cuglab/linux_asm` 目录下生成了新文件，其中 `asm.bin` 就是需要下载到开发板运行的可执行文件，按照实验一的步骤，把它下载到开发板 SDRAM 中测试。

下面简单介绍执行 `make` 后运行的三条编译命令的作用：

`arm-linux-gcc` 用法与 `gcc` 相同，编译选项可参考 `gcc`。在本实验中的作用就是编译 `asm.s` 汇编源代码。

`arm-linux-ld` 用于将多个目标文件、库文件连接成可执行文件。它可以直接指定代码段、数据段、`bss` 段的起始地址，格式如下：

```
-Ttext startaddr
```

```
-Tdata startaddr
```

```
-Tbss startaddr
```

其中的 `startaddr` 分别表示代码段、数据段和 `bss` 段的起始地址，它是一个 16 进制数。本实验中是将可执行程序代码段的起始地址设置为 `0x30000000`。

`arm-linux-objcopy` 被用来复制一个目标文件的内容到另一个文件中，可以使用不同于源文件的格式来输出目标文件，即可以进行格式转换。在本实验中用来将 ELF 格式的可执行文件转换为二进制文件。

5. 下载并运行执行程序

将 Linux 下交叉编译生成的可执行二进制文件拷贝到 windows 操作系统下，在串口超级终端运行界面选择 7 将文件下载到 SDRAM 中并执行，并观察 LED 灯的状态（此步操作同实验一的第 6 步）。

第三篇 Linux 操作系统移植与驱动初体验

本章实验实现将嵌入式 Linux 系统移植到 ARM 开发板上，并基于嵌入式 Linux 操作系统实现 LED 驱动程序的加载及使用。因为课程时间的限制，这里只对上述内容做了个初步的体验，对基于 Linux 操作系统的驱动程序及应用程序开发建立基本的框架概念。

实验十一 Linux 系统移植初体验

实验目的：	了解嵌入式 Linux 操作系统的移植
硬件连接：	开发板三线连接（电源线、串口线、USB 线），在 Nor Flash 模式下加载操作系统内核及文件系统，在 Nand Flash 模式下启动操作系统。
实验要求：	初步了解嵌入式 Linux 操作系统的移植步骤及加载过程。

1. 烧写 bootload 程序

烧写 u-boot: 将启动开关拨到 nor 启动，选择选项 1，下载 u-boot.bin 到 nandflash, u-boot 下载完成后，出现如下界面：

```
[1] Download u-boot or other bootloader to Nand Flash
[2] Download Eboot (eboot.nb0) to Nand Flash
[3] Download Linux Kernel (zImage.bin) to Nand Flash
[4] Download winCE NK.bin to Nand Flash
[5] Download CRAMFS image to Nand Flash
[6] Download YAFFS image (root.bin) to Nand Flash
[7] Download Program (uCOS-II or TQ2440_Test) to SDRAM and Run it
[8] Boot the system
[9] Format the Nand Flash
[0] Set the boot parameters
[a] Download User Program (eg: uCOS-II or TQ2440_Test)
[b] Download LOGO Picture (.bin) to Nand Flash
[l] Set LCD Parameters
[n] Enter TFTP download mode menu
[o] Download u-boot to Nor Flash
[r] Reboot u-boot
[t] Test Linux Image (zImage)
[q] quit from menu
Enter your selection: 1
USB host is connected. waiting a download.

Now, Downloading [ADDRESS:30000000h,TOTAL:239218]
RECEIVED FILE SIZE: 239218 (233KB/S, 1S)

NAND erase: device 0 offset 0x0, size 0x40000
Erasing at 0x20000 -- 100% complete.
OK

NAND write: device 0 offset 0x0, size 0x3a668
Writing data at 0x3a000 -- 100% complete.
239208 bytes written: OK
```

2. 烧写操作系统内核

烧写操作系统内核 zImage: 保持开关为 nor 启动，选择选项 3，下载 zImage.bin 到 nandflash, 内核下载完成后，出现如下界面：

```

[1] Download u-boot or other bootloader to Nand Flash
[2] Download Eboot (eboot.nb0) to Nand Flash
[3] Download Linux Kernel (zImage.bin) to Nand Flash
[4] Download winCE NK.bin to Nand Flash
[5] Download CRAMFS image to Nand Flash
[6] Download YAFFS image (root.bin) to Nand Flash
[7] Download Program (uCOS-II or TQ2440_Test) to SDRAM and Run it
[8] Boot the system
[9] Format the Nand Flash
[0] Set the boot parameters
[a] Download User Program (eg: uCOS-II or TQ2440_Test)
[b] Download LOGO Picture (.bin) to Nand Flash
[l] Set LCD Parameters
[n] Enter TFTP download mode menu
[o] Download u-boot to Nor Flash
[r] Reboot u-boot
[t] Test Linux Image (zImage)
[q] quit from menu
Enter your selection: 3
USB host is connected. waiting a download.

Now, Downloading [ADDRESS:30000000h,TOTAL:1923054]
RECEIVED FILE SIZE: 1923054 (625KB/S, 3S)

NAND erase: device 0 offset 0x200000, size 0x300000
Erasing at 0x4e0000 -- 100% complete.
OK

NAND write: device 0 offset 0x200000, size 0x1d57e4
writing data at 0x3d5000 -- 100% complete.
1923044 bytes written: OK

```

3. 烧写根文件系统

文件系统 rootfs: 保持开关为 nor 启动, 选择选项 6, 下载 rootfs.bin 到 nandflash, 文件的下载需要的时间稍长, 文件系统下载完成后, 出现如下界面:

```

[1] Download u-boot or other bootloader to Nand Flash
[2] Download Eboot (eboot.nb0) to Nand Flash
[3] Download Linux Kernel (zImage.bin) to Nand Flash
[4] Download winCE NK.bin to Nand Flash
[5] Download CRAMFS image to Nand Flash
[6] Download YAFFS image (root.bin) to Nand Flash
[7] Download Program (uCOS-II or TQ2440_Test) to SDRAM and Run it
[8] Boot the system
[9] Format the Nand Flash
[0] Set the boot parameters
[a] Download User Program (eg: uCOS-II or TQ2440_Test)
[b] Download LOGO Picture (.bin) to Nand Flash
[l] Set LCD Parameters
[n] Enter TFTP download mode menu
[o] Download u-boot to Nor Flash
[r] Reboot u-boot
[t] Test Linux Image (zImage)
[q] quit from menu
Enter your selection: 6
USB host is connected. waiting a download.

Now, Downloading [ADDRESS:30000000h,TOTAL:58884682]
TQ2440 board's Nand is 256MB or 1GB, Please choose correct yaffs image.
Image Length = 58884672, StartAddr = 0x500000, Length = 0xfb00000.
writing data at 0xda59800 -- 85% complete.
skipping bad block 0xdb00000
writing data at 0x10000000 -- 100% complete.
Yaffs Images written to Nand Flash complete!

```


4. 启动操作系统

这样就将 linux 系统通过 usb 下载到开发板了，将开发板启动开关拨到 nand，启动开发板，将会在超级终端看到系统正常启动起来。如下图所示：

```
[root@EmbedSky /]# ls
bin          linuxrc      root         tq2440_serial0
dev          lost+found  sbin         udisk
etc          mnt         sddisk      usr
home        opt         sys         var
lib         proc        tmp         web
```

实验十二 Linux 操作系统下驱动开发初体验




实验目的：	了解嵌入式 Linux 操作系统下驱动程序的加载及使用。
硬件连接：	开发板三线连接（电源线、串口线、USB 线）在 Nand Flash 模式下进入操作系统。
实验要求：	初步了解基于嵌入式 Linux 操作系统的驱动程序加载及使用过程。

第一篇和第二篇的实验都是裸机程序实验，本小节实验是在开发板嵌入式 linux 系统下加载驱动程序以及设计应用程序调用驱动程序。实验十一中已将嵌入式 Linux 系统烧写在 Nand Flash 中。

LED 对于嵌入式 Linux 系统来说它是一个硬件设备，在 Linux 系统中想要对硬件设备操作，就需要相关的硬件驱动程序。在本实验中，LED 的驱动程序为 EmbedSky_leds.ko，它是以内核模块的形式动态加载的。其应用测试程序为 leds。

1. 将所给源码中的三个文件拷贝到 U 盘

将 EmbedSky_hello.ko、EmbedSky_leds.ko、leds 拷贝到 u 盘。其中 EmbedSky_hello.ko 是 hello world 驱动程序，EmbedSky_leds.ko 是 LED 灯的驱动程序，leds 是应用程序（程序可以通过调用了 LED 灯驱动程序来控制亮灭）。

 EmbedSky_hello.ko	2019/11/13 14:50	KO 文件	3 KB
 EmbedSky_leds.ko	2019/11/13 14:50	KO 文件	5 KB
 leds	2019/11/13 14:18	文件	4 KB

2. 将 U 盘插入开发板的 usb 接口

将 U 盘插入开发板的 usb 接口，如下所示：

```
[root@EmbedSky /]# usb 1-1: new full speed USB device using s3c2410-ohci and address 2
usb 1-1: configuration #1 chosen from 1 choice
scsi0 : SCSI emulation for USB Mass Storage devices
scsi 0:0:0:0: Direct-Access Kingston DataTraveler 2.0 PMAP PQ: 0 ANSI: 4
sd 0:0:0:0: [sda] 15240576 512-byte hardware sectors (7803 MB)
sd 0:0:0:0: [sda] write Protect is off
sd 0:0:0:0: [sda] Assuming drive cache: write through
sd 0:0:0:0: [sda] 15240576 512-byte hardware sectors (7803 MB)
sd 0:0:0:0: [sda] write Protect is off
sd 0:0:0:0: [sda] Assuming drive cache: write through
sda: sda1
sd 0:0:0:0: [sda] Attached SCSI removable disk
FAT: utf8 is not a recommended IO charset for FAT filesystems, filesystem will be case sensitive!
[root@EmbedSky /]#
```

3. 在开发板 Linux 下查询 U 盘文件

切换到 `udisk` 目录，如果 U 盘加载成功，可以查到拷贝到 U 盘里面的文件。

```
[root@EmbedSky /]# cd udisk/
[root@EmbedSky /udisk]# ls
EmbedSky_hello.ko      System volume Information
EmbedSky_leds.ko       leds
```

4. 加载 hello 驱动程序（硬件无关）

加载第一个驱动程序，命令 `insmod EmbedSky_hello.ko`，终端打印如下：

```
[root@EmbedSky /udisk]# insmod EmbedSky_hello.ko
Hello,I Love CUG!
This is first driver program.
```

5. 卸载驱动程序

卸载该驱动程序，命令 `rmmod EmbedSky_hello.ko`，终端打印如下：

```
[root@EmbedSky /udisk]# rmmod EmbedSky_hello.ko
Exit!
```

6. 加载 LED 驱动程序（硬件相关）

加载第二个驱动程序，命令 `insmod EmbedSky_leds.ko`，终端打印如下：

```
[root@EmbedSky /udisk]# insmod EmbedSky_leds.ko
TQ2440 LED Driver, (c) 2008,2009 www.cug.edu.cn
tq2440-leds initialized
```

7. 查看设备节点

查看设备节点，命令 `ls /dev/tq2440-leds`，终端打印如下：

```
[root@EmbedSky /udisk]# ls /dev/tq2440-leds
/dev/tq2440-leds
```

8. 测试应用程序

运行 leds 需要给它传递两个参数，其执行格式为:

```
./leds led_no 0|1
```

其中参数“led_no”表示 led 灯编号，四个 led 分别对应 1、2、3、4，当该参数为 5 时表示同时控制四个灯；参数“0|1”表示控制 led 亮(1)灭(0)。

在该目录下执行命令：

- ① `./leds 5 0` 会看到开发板上的 4 个 led 灯都灭了。
- ② `./leds 5 1` 会看到开发板上的 4 个 led 灯都亮了。
- ③ `./leds 1 1` 点亮第一个 led 灯
- ④ `./leds 1 0` 熄灭第一个 led 灯
- ⑤ `./leds 2 1` 点亮第二个 led 灯

.....