

校内讲义

嵌入式 Linux UP-CUP2440 开发平台 嵌入式系统实验指导书

赵宏伟 卢欣华

吉 林 大 学

目 录

实验一	ARM 开发工具 ADS 1.2 集成开发环境.....	3
实验二	ARM 汇编语言程序设计	9
实验三	Linux 基本操作及交叉编译环境的建立.....	12
实验四	直流电机及 4*4 矩阵键盘驱动实验.....	39
实验五	LED 数码管实验.....	44
实验六	A/D 和 D/A 接口实验	48
实验七	HTML 网页设计和嵌入式 Web 服务器设计.....	52
实验八	图形界面应用程序设计	59

实验一 ARM 开发工具 ADS 1.2 集成开发环境

预习要求:

- (1) 阅读 ADS 集成开发环境相关资料, 了解 ADS 工程编辑的内容。
- (2) 了解 AXD 调试的内容。

一、 基础性实验

1. 实验目的

熟悉 ADS1.2 集成开发环境的使用方法。

2. 实验设备

- (1) 硬件: PC 机一台。
- (2) 软件: Windows XP 系统, ADS 1.2 集成开发环境。

3. 实验内容

- (1) 建立一个新的汇编工程。
- (2) 建立一个汇编语言源文件, 并添加到工程中。
- (3) 设置编译连接控制选项。
- (4) 编译连接工程。
- (5) 调试工程。

4. 实验步骤

(1) 启动 ADS 1.2 集成开发环境, 点击 WINDOWS 操作系统的[开始]->[程序]->[ARM Developer Suite v1.2]->[CodeWarrior for ARM Developer Suite]启动 Metrowerks CodeWarrior 或双击桌面上的“CodeWarrior for ARM Developer Suite”快捷方式启动。

(2) 选择[File]->[New...], 在 Project 标签中, 使用 ARM Executable Image 工程模板建立一个工程。然后在[Location]项选择工程存放路径, 并在[Project name]项输入工程名称(本例子工程名为 Exp1), 点击[确定]按钮即可建立相应工程, 工程文件名默认后缀为 mcp。

(3) 在主框架下, 点击[File]菜单, 选择[New...], 选择 File 标签, 输入文件全名(如汇编源文件为 test.s, C 语言源文件为 test.c)建立一个新文件。注意: 一定要将文件保存到相应工程的目录下, 请确保勾选“Add to Project”复选框; 下方的“Targets”的三个复选框(Debug、DebugRel、Release)也都勾选上。点击“确定”后, 可以看到工程窗口内多了一个刚加入的文件(本例子是 test.s), 如图 1.1。可以对文件写入源代码, 请参考“5. 实验参考程序”。

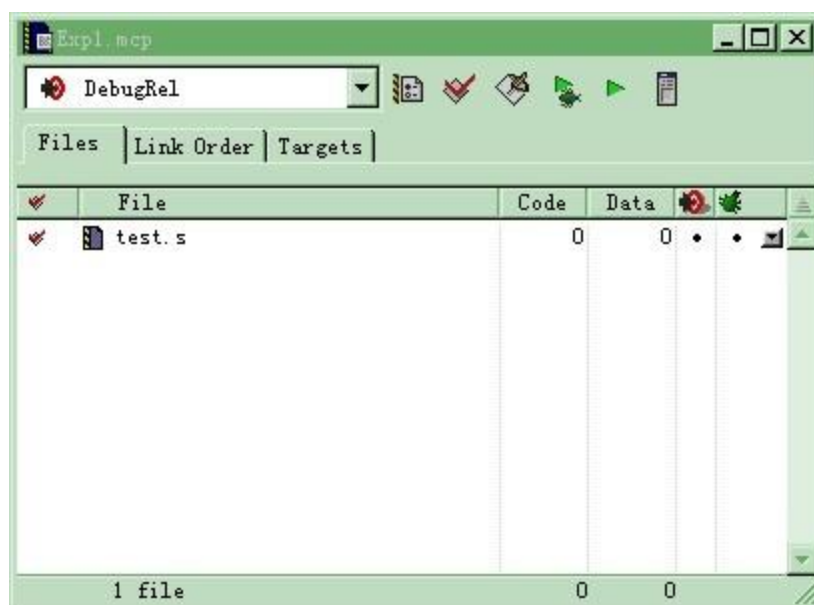


图 1.1 成功建立新文件后的工程窗口

(4) 如果想把一个已经存在的源文件加入工程,在工程窗口中[Files]页空白处点击鼠标右键,选择“Add Files...”,在弹出的对话框中,选择相应的源文件,点击[打开]按钮后,在弹出的对话框中单击[OK]。如需要对文件进行编辑时,在工程对话框中双击要编辑的文件即可。

(5) 选择[Edit]->[DebugRel Settings...],在 DebugRel Settings 对话框的左边单击 ARM Linker 项,然后在 Output 页,“RO Base”和“RW Base”处修改连接地址(见图 1.2),在 Options 页,“Image entry point”处设置调试入口地址(见图 1.3)。

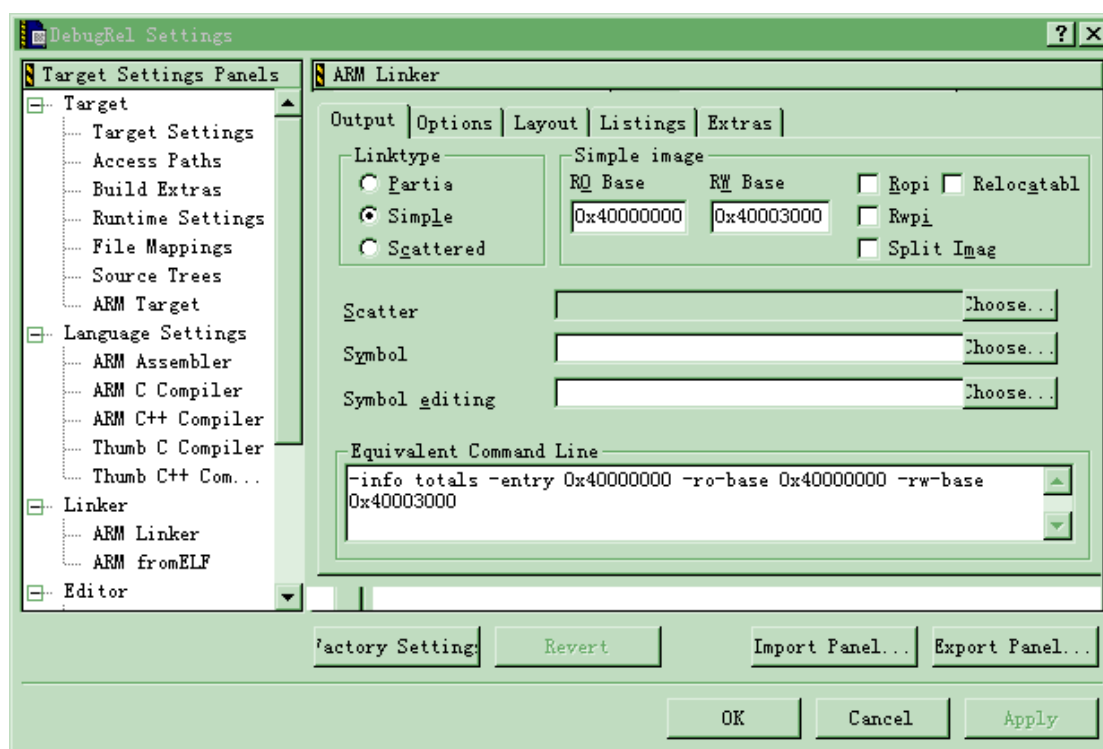


图 1.2 工程连接地址设置

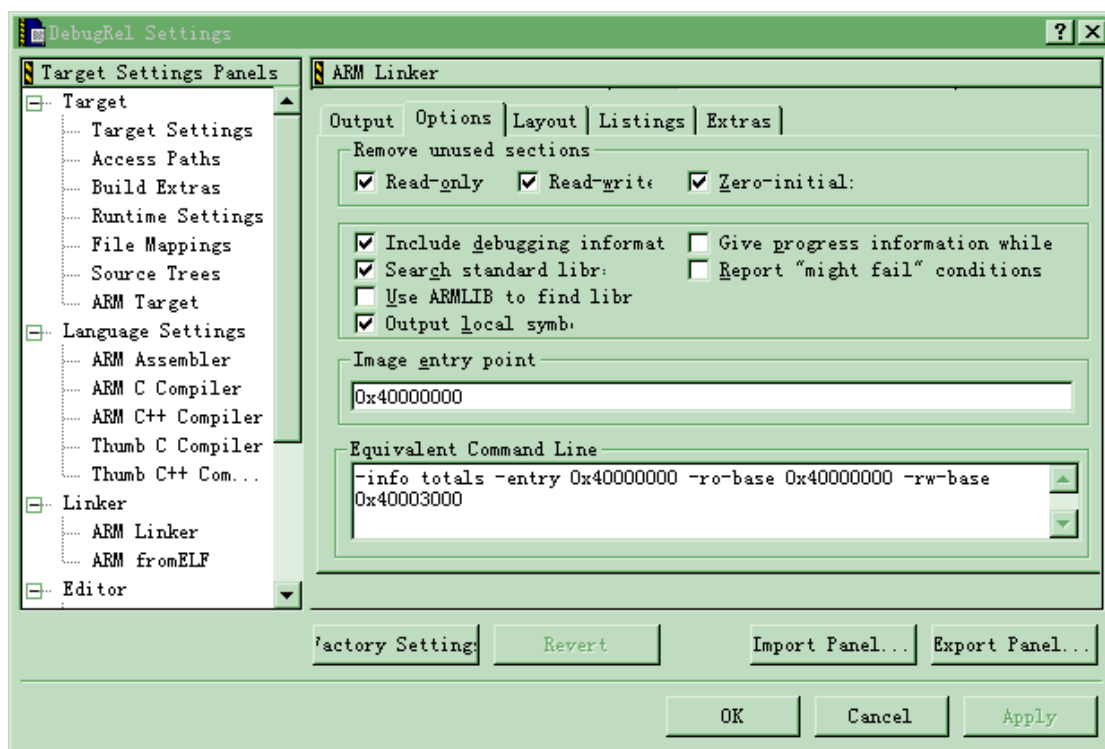



图 1.3 工程调试入口地址设置

(6) 选择[Project]->[Make], 将编译连接整个工程, 如果编译成功, Errors & Warnings 对话框会报告编译错误为 0, 那么就可以对工程进行仿真。

(7) 选择[Project]->[Debug], 或点击  按钮, IDE 环境就会自动启动 AXD 调试软件。

(8) 在 AXD 调试环境中, 选择[Option]->[Configure Target], 弹出 Choose Target 对话框, 如图 1.4 所示, 选择...ARMulate.dll 仿真环境。

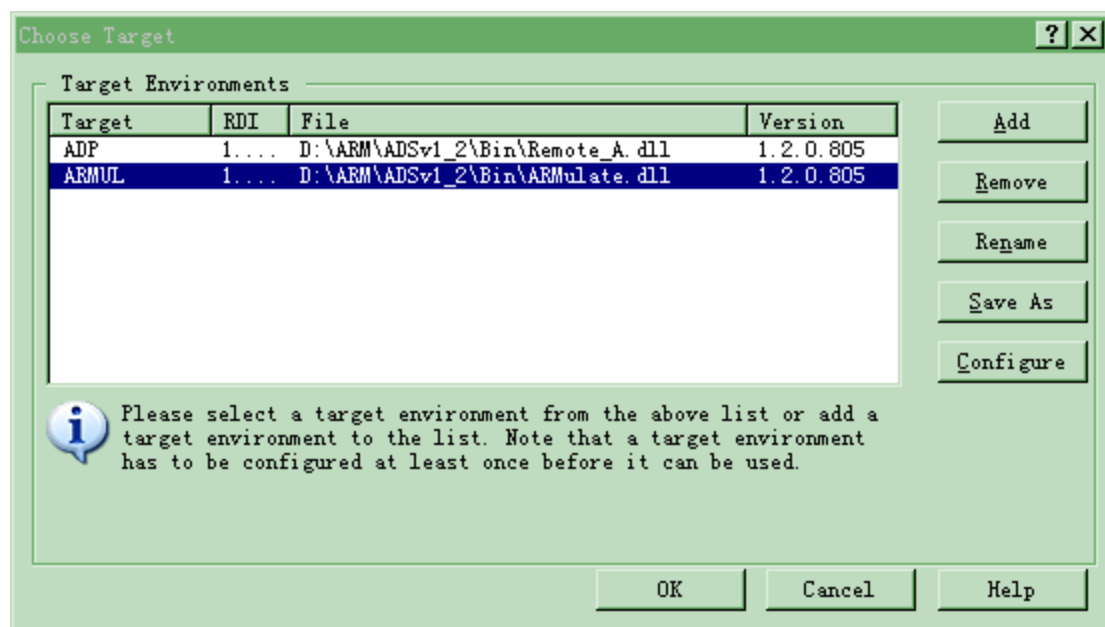


图 1.4 选择仿真环境窗口

(9) 选择[File]->[Load Image], 在工程的 DebugRel 文件夹下找到工程中的映像文件 Exp1.axf, 把映像文件加载到 AXD 环境中。加载成功后如图 1.5 所示。

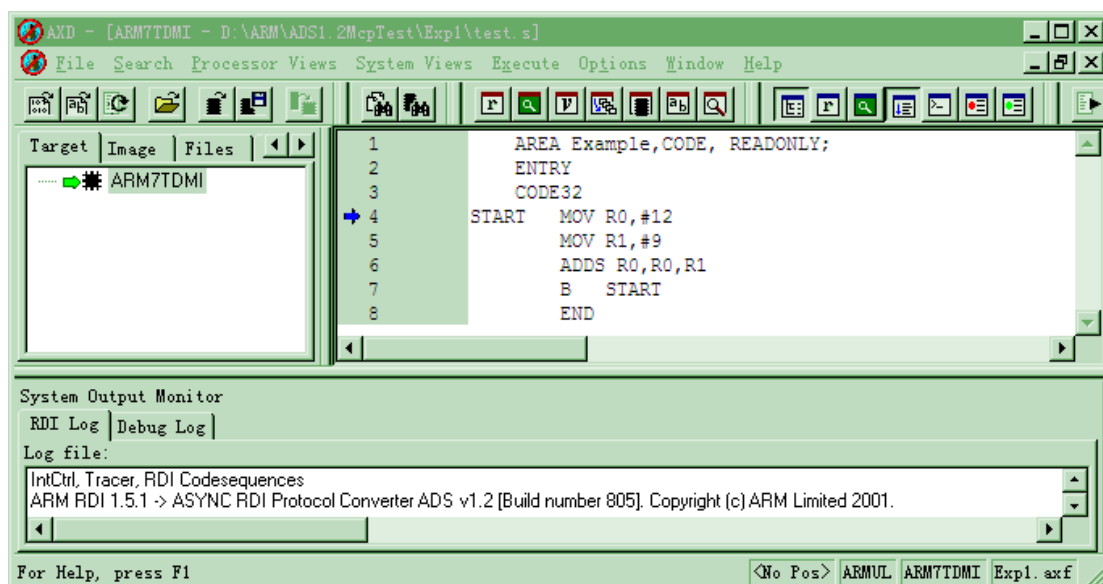


图 1.5 加载映像成功后可以开始调试运行

注意：

(1) 由于 ADS 安装以后默认字体是 Courier，对于中文支持不完善，因此建议修改字体。选择 [Edit]->[Perferences...], 在 Font & Tabs 选项设置字体是 Fixedsys, Script 是 CHINESE_GB2312。由于 Tab 在不同文本编辑器解释不同，建议勾选 “Tab Inserts Spaces” 复选框，使 Tab 键插入的是多个空格。

(2) 在 AXD 调试软件中断点调试方法：在要设置断点的行，双击鼠标即可，如果出现红色实心圆点，那么表示断点设置成功。然后选择 [Execute]->[Go] 全速运行，可以发现程序会在断点处停止，蓝色箭头则表示程序当前执行到的位置。通过断点调试可以观察 ARM 寄存器数值变化。双击断点则可以取消断点设置。

(3) 在 AXD 中观察各寄存器值的调试方法：选择 [Processor views]->[Registers], 打开寄存器观察窗口，监视 R0~R14、PC、CPSR、SPSR 的值。

(4) 在 AXD 中观察存储器值的方法：[Processor views]->[Memory], 打开存储器观察窗口，设置观察地址即可监视地址上的值。显示方式可以改变，在 Memory 窗口中点击鼠标右键，选择显示格式为 8Bit、16Bit 或 32Bit。地址上的值也可以改变，双击要改变的内容即可输入新值。

5. 实验参考程序

```

AREA   Example, CODE, READONLY    ; 声明代码段 Example1
ENTRY                                ; 标识程序入口
CODE32                              ; 声明 32 位 ARM 指令
START  MOV    R0,#12                ; 设置参数
      MOV    R1,#9
      ADDS   R0,R0,R1                ; R0 = R0 + R1
      B      START
      END

```

6. 实验思考题

- (1) 工程模板有什么作用？
- (2) 如何强行重新编译工程的所有文件？

二、 提高性试验

1. 实验目的

- (1) 熟悉 ADS1.2 集成开发环境的使用方法。
- (2) 通过实验了解使用 ADS 1.2 编写 C 语言程序，并进行调试。

(3) 掌握运用 AXD 调试工具断点调试的方法。

2. 实验设备

(1) 硬件：PC 机一台。

(2) 软件：Windows XP 系统，ADS1.2 集成开发环境。

3. 实验内容

利用 ARM Executable Image 工程模板建立一个工程，编写一个 C 语言程序文件，使用加法来计算 $1+2+3+\dots+(N-1)+N$ 的值($N>0$)。

4. 实验步骤

(1) 启动 ADS 1.2 集成开发环境，使用 ARM Executable Image 工程模板建立一个工程 Exp2.mcp，工程文件名默认后缀也为 mcp。

(2) 建立源文件 main.c，并添加到工程中，方法同基础实验，打开刚建立的 main.c 文件，编写实验程序，可参考“5. 实验参考程序”。

(3) 选择[Project]->[Make]，编译连接工程，如果编译成功，Errors & Warnings 对话框会报告编译错误为 0。

(4) 选择[Project]->[Debug]，启动 AXD 进行仿真调试。

(5) 在 AXD 调试环境中，选择[Option]->[Configure Target...]，弹出 Choose Target 对话框，确保选中...ARMulate.dll 仿真环境。

(6) 加载 Exp2 工程中的映像文件 Exp2.axf，加载成功后如图 1.6 所示。

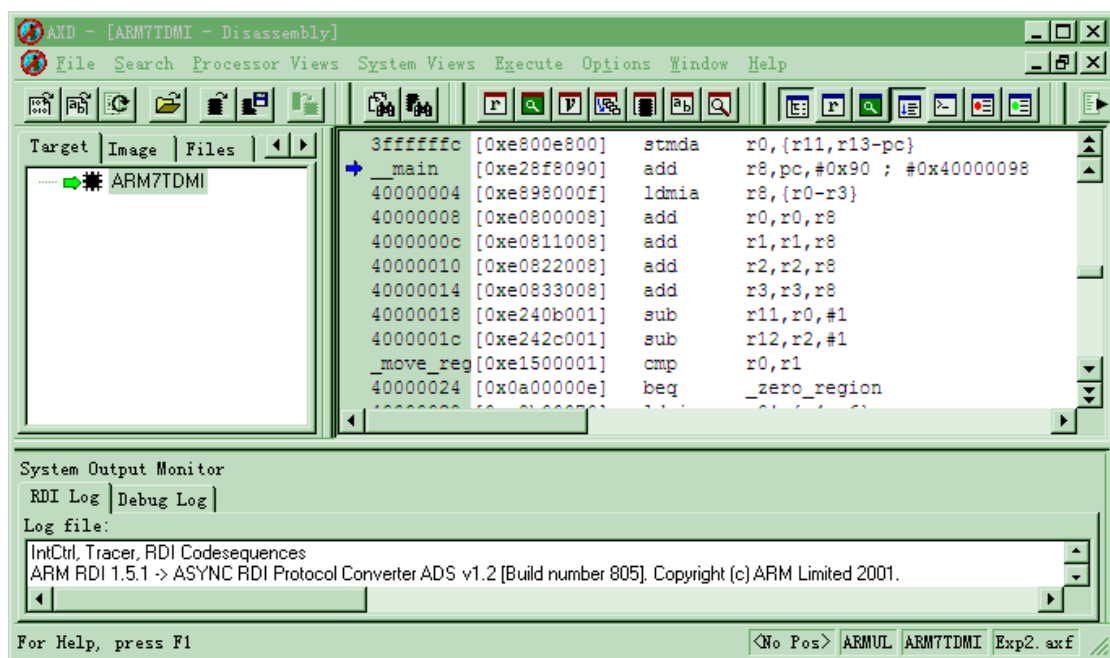



图 1.6 加载 C 语言程序映像成功后的调试窗口

(7) 按[F5]键，或者选择菜单[Execute]->[Go]，或者点击  按钮，全速运行程序，程序会自动在 main() 函数断点处停止。

(8) 通过 AXD 调试软件中的断点调试方法调试程序；并通过选择[Processor views]->[Variable]观察各个变量的值变化检查程序的正确性。

5. 实验参考程序

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

```
#define N 100
```

```
void main(void)
{
    int i;
    int sum=0;
    for(i=0;i<=N;i++)
    {
        sum+=i;
    }
    printf("%d",sum);
    while(1);
}
```

6. 实验思考题

- (1) 在 AXD 调试时如何复位程序？
- (2) 更改变量 N 的值，观察执行结果。

实验二 ARM 汇编语言程序设计

预习要求:

- (1) 仔细阅读 ARM9 指令系统内容。
- (2) 了解 ADS 工程编辑和 AXD 调试的内容。

一、 基础性实验

1. 实验目的

- (1) 熟悉 ADS1.2 集成开发环境及 ARMulator 软件仿真。
- (2) 掌握 ARM9 汇编指令的用法, 能够熟练掌握循环次数已知的循环程序设计。
- (3) 掌握断点调试, 观察 ARM 寄存器数值变化。

2. 实验设备

- (1) 硬件: PC 机一台。
- (2) 软件: Window XP 系统, ADS 1.2 集成开发环境。

3. 实验内容

编写程序计算 $1+2*3+3*4+4*5+\dots+N(N+1)$, 直到 N 等于 10 为止。使用 ADS 1.2 软件仿真, 单步、全速运行程序, 打开寄存器窗口监视 R0, R1, R2, R3 的值, 观察它们的变化。

4. 实验步骤

- (1) 启动 ADS 1.2 集成开发环境, 使用 ARM Executable Image 工程模板建立一个工程。
- (2) 建立汇编源文件 EXAM4_1.s, 编写实验程序, 然后添加到工程中。
- (3) 设置工程连接地址 RO Base 为 0x40000000, RW Base 为 0x40003000。设置调试入口地址 Image entry point 为 0x40000000。

- (4) 编译连接工程, 选择[Project]->[Debug], 启动 AXD 进行软件仿真调试。

注意: 在 AXD 调试环境, 打开[Options]->[Configure Target...], 弹出 Choose Target 窗口, 在“Target Environments”框中选择“...ARMulate.dll”项, 进行 ARMulator 软件仿真。

- (5) 选择[Processor views]->[Registers], 打开寄存器观察窗口, 监视程序运行过程中寄存器值的变化。
- (6) 单步运行至 R1 的值为 0x0000000A, 则表示 N 值已经累加到循环上限 10, 如图 2.1。

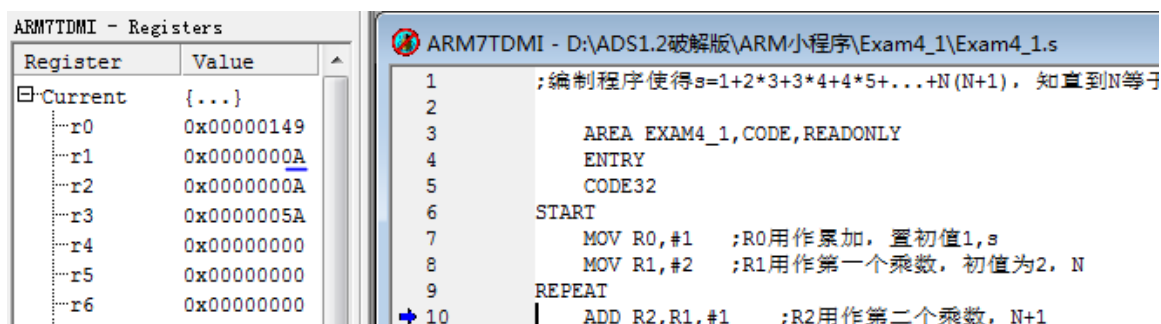


图 2.1 单步调试程序并观察寄存器的值

(7) 继续单步运行程序, 可以设置/取消断点, 或者全速运行程序, 停止程序运行, 调试时观察寄存器的值。ARM 汇编程序结尾处多设为死循环, 使程序不会退出方便调试人员进行调试。如图 2.2。

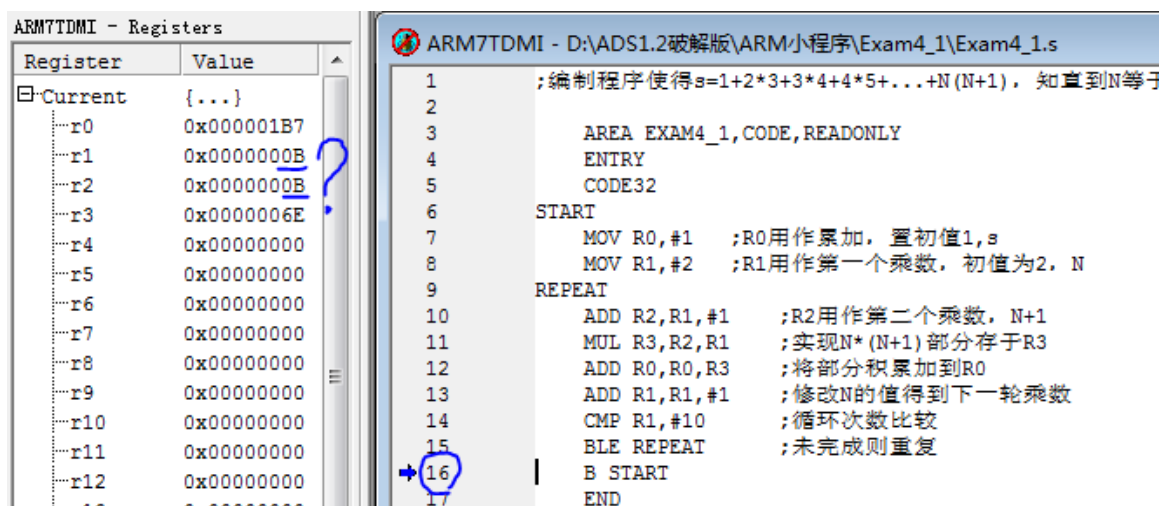


图 2.2 ARM 汇编语言程序结尾多为死循环

5. 实验参考程序

```

AREA EXAM4_1, CODE, READONLY
ENTRY
CODE32
START
    MOV R0, #1 ;R0 用作累加，置初值 1
    MOV R1, #2 ;R1 用作第一个乘数，初值为 2，N
REPEAT
    ADD R2, R1, #1 ;R2 用作第二个乘数，N+1
    MUL R3, R2, R1 ;实现 N*(N+1) 部分存于 R3
    ADD R0, R0, R3 ;将部分积累加到 R0
    ADD R1, R1, #1 ;修改 N 的值得到下一轮乘数
    CMP R1, #10 ;循环次数比较
    BLE REPEAT ;未完成则重复
B START
END

```

6. 实验思考题

- (1) 如图 2.2，R1 的值为什么会等于 11？跟 CMP 语句有关系吗？
- (2) 如果循环次数未知，程序该如何编写？

二、设计性实验

1. 实验目的

- (1) 深入了解循环程序的编写，能够通过条件控制语句来控制循环次数未知的情况。
- (2) 掌握使用 LDR/STR 指令完成存储器的访问。

2. 实验设备

- (1) 硬件：PC 机一台。
- (2) 软件：Window XP 系统，ADS 1.2 集成开发环境。

3. 实验内容

试编写一个汇编程序，求两个数组 DATA1 和 DATA2 对应的数据之和，并把结果存在新的数组 SUM 中，计算一直进行到两数之和为零时结束，并把新数组的长度存在 R0 中。其中 DATA1、DATA2 中数据分别为 {2,5,0,3,-4,5,0,10,9}、{3,5,4,-2,0,8,3,-10,5}。

4. 实验步骤

- (1) 启动 ADS 1.2, 使用 ARM Executable Image 工程模板建立一个工程。
- (2) 建立汇编源文件 EXAM4_2.s, 编写实验程序, 然后添加到工程中。
- (3) 设置工程连接地址 RO Base 为 0x40000000, RW Base 为 0x40003000。设置调试入口地址 Image entry point 为 0x40000000。
- (4) 编译连接工程, 选择[Project]->[Debug], 启动 AXD 进行软件仿真调试。
- (5) 单步调试程序, 当程序执行三条语句后, 寄存器 R0、R1、R2 的变化如下图 2.3。

Register	Value		
Current	{...}	6	CODE32
r0	0x00000000	7	START
r1	0x40003000	8	LDR R1,=DATA1
r2	0x40003024	9	LDR R2,=DATA2
r3	0x40003048	10	LDR R3,=SUM
		11	MOV R0,#0

图 2.3 单步调试程序

可以发现 R0、R1、R2 的值相差 24 (16 进制), 即 36 (10 进制), 因为我们定义的三个数组是 9 个 4 字节长度。最终循环执行 8 次, DATA1 与 DATA2 中数据互补, 跳出循环。

5. 参考代码

```
AREA EXAM4_2, CODE, READONLY
ENTRY
CODE32
START
    LDR R1,=DATA1 ;数组 DATA1 的首地址存放到 R1
    LDR R2,=DATA2 ;数组 DATA2 的首地址存放到 R2
    LDR R3,=SUM    ;数组 SUM 的首地址存放到 R3
    MOV R0,#0      ;计数器 R0 的初始值置 0
LOOP
    LDR R4,[R1],#04 ;取 DATA1 数组的一个数, 同时修改地址指针
    LDR R5,[R2],#04 ;取 DATA2 数组的一个数, 同时修改地址指针
    ADDS R4,R4,R5   ;相加并影响标志位
    ADD R0,R0,#1    ;保存结果到 SUM 中并同时修改地址指针
    STR R4,[R3],#04 ;若相加结果不为 0 则循环
    BNE LOOP
    B START

AREA BlockDtat, DATA, READWRITE ;定义数据段
DATA1 DCB 2,5,0,3,-4,5,0,10,9
DATA2 DCB 3,5,4,-2,0,8,3,-10,5
SUM    DCB 0,0,0,0,0,0,0,0,0
END
```

6. 实验思考题

- (1) 使用 ARM 汇编指令实现循环程序时, 如何在 for, while 结构中实现 break, continue?
- (2) 能否更改数据区的数据定义的默认长度, 使得每个数据最大长度为 2 字节?

实验三 Linux 基本操作及交叉编译环境的建立

预习要求：

- (1) 阅读《UP-CUP S2440 经典 Linux 实验指导书》，了解嵌入式 Linux 开发的相关内容。
- (2) 了解 UP-CUP S2440 教学实验开发平台的硬件结构。
- (3) 阅读如何配置 UP-CUP S2440 教学实验平台开发环境的相关文档资料。

一、 基础性实验

1. 实验目的

熟练掌握 Linux 基本操作命令。

2. 实验设备

- (1) 硬件：PC 机 Pentium 500 以上，硬盘 40G 以上，内存大于 256M。
- (2) 软件：Vmware Workstation8 虚拟机系统，Red Hat Linux 操作系统。

3. 实验内容

- (1) 安装 Vmware Workstation8 虚拟机系统，在虚拟机中运行 Red Hat Linux 操作系统。
- (2) 熟练掌握 Linux 操作系统下的常用命令。

4. 实验步骤

(1) 建立开发环境。操作系统一般使用 REDHAT、Fedora Core、Ubuntu 系列等等，目前市场上的 LINUX 版本比较多，本实验指导是在虚拟机 VM8 中安装 Red Hat Linux 系统作为宿主机环境的，后续实验都是在此指定的环境中进行。其中交叉编译器（如 arm-linux-gcc、arm-uclibc-gcc）的安装也在后面的步骤中介绍。

运行 Vmware Workstation8 虚拟机，本实验已提供配置好的 LINUX 操作系统，无需自行安装操作系统。请打开[REDHAT_2440]文件夹，找到里面扩展名为.vmx 的文件双击，即在虚拟机中运行它。打开 Red Hat Linux 操作系统电源，启动 Linux 操作系统。

- (2) 启动完毕后，输入用户名 root，密码 123456，成功进入 Red Hat Linux 操作系统。如图 3.1。

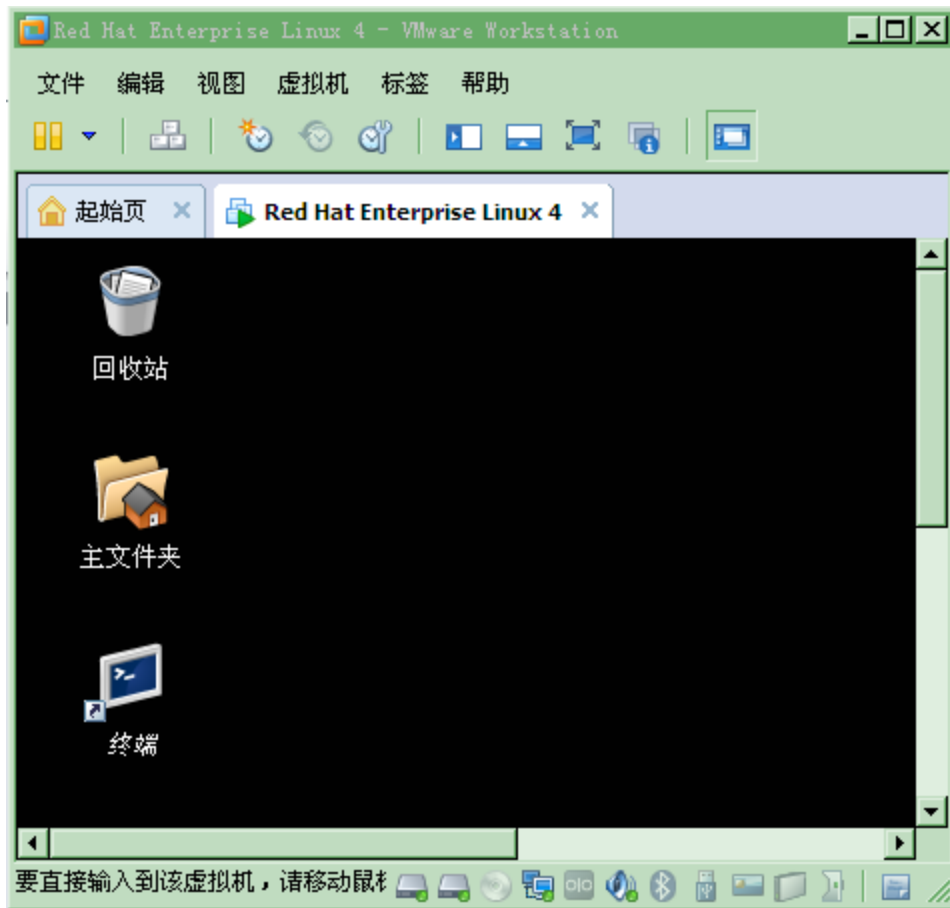


图 3.1 VM8 虚拟机下的 Red Hat Linux 操作系统

(3) 双击桌面上的“终端”图标，打开 Linux 的命令行窗口。提示符为[root@vm-dev ~]#，可以在此提示符下输入各种 Linux 的基本操作命令。

改变目录和查看当前目录命令

(i) 切换目录命令 **cd**

说明：变换工作目录至命令中指定的目录名称，可以是绝对路径或相对路径。若目录名称省略，则变换至使用者的 home directory (也就是刚登录时所在的目录)。

参数：

"~"表示为 home directory。

"."表示目前所在的目录。

".."表示目前目录位置的上一层目录。

范例：

- 跳到 /usr/bin/目录：

cd /usr/bin

- 跳到自己的 home directory：

cd ~

- 跳到当前目录的上上两层：

cd ../../

(ii) 查看当前目录命令 **pwd**

说明：执行 pwd 指令可查看目前所在的工作目录的绝对路径名称。

显示目录和文件的命令

(i) 查看文件和目录命令 **ls**

说明：显示指定工作目录下的内容，列出指定工作目录下的文件及子目录。如果没有指定目录，那么就是当前目录。

参数：

- a 显示所有文件及目录。
- l 除文件名称外，文件权限、拥有者、文件大小等详细信息也一并列出。
- r 将文件以相反次序显示（原来是依英文字母的次序）。
- t 将文件依建立时间之先后次序列出。
- A 同-a，但不列出"."（当前目录）及".."（父目录）。
- F 在列出的文件名称后加一符号；例如可执行文件加"*"，目录则加"/"。
- R 若目录下有文件，则文件全都依序列出。

范例：

- 列出长字符串，包含文件的一些详细信息：
ls -al
- 列出当前工作目录下所有名称是 s 开头的文件，越新的排在越后面：
ls -ltr s*
- 将 /bin 目录以下所有目录及文件详细信息列出：
ls -lR /bin
- 列出当前工作目录下所有文件及目录；目录于名称后加"/"，可执行文件于名称后加"*"：
ls -AF

(ii) 显示目录或文件大小命令 **du**

说明：显示指定的目录或文件所占用的磁盘空间。该命令逐级进入指定目录的每一个子目录并显示该目录占用文件系统数据块（1024 字节）的情况。若没有给出目录或文件名，则对当前目录进行统计。

参数：

- b 或--bytes 显示目录或文件大小时，以 byte 为单位。
- c 或--total 除了显示个别目录或文件的大小外，同时也显示所有目录或文件的总和。
- D 或--dereference -args 显示指定符号连接的源文件大小。
- h 或--human-readable 以 K, M, G 为单位，提高信息的可读性。
- H 或--si 与-h 参数相同，但是 K, M, G 是以 1000 为换算单位。
- k 或--kilobytes 以 1024 bytes 为单位。
- l 或--count-links 重复计算硬件连接的文件。
- m 或--megabytes 以 1MB 为单位。
- s 或--summarize 对每个参数只给出占用的数据块总数。
- S 或--separate-dirs 显示个别目录的大小时，并不含其子目录的大小。
- x 或--one-file-system 以一开始处理时的文件系统为准，若遇上其它不同的文件系统目录则略过。
- exclude=<目录或文件> 略过指定的目录或文件。
- max-depth=<目录层数> 超过指定层数的目录后，予以忽略。

范例：

- 估算目录“dir1”已经使用的磁盘空间：
du -sh dir1
- 以容量大小为依据依次显示文件和目录的大小
du -sk *

修改目录、文件权限和属性命令

(i) 改变指定目录或文件的权限命令 **chmod**

说明：Linux 的文件调用权限分为三级：文件拥有者、群组、其他。通过 chmod 命令可以控制文件的权限如何被他人所调用。

参数:

mode 权限设定字串, 格式如下: [ugoa...][[+|=][rwxX]...][,...], 其中:

u 表示该文件的拥有者, g 表示与该文件的拥有者属于同一个群体(group)者, o 表示其他以外的人, a 表示这三者皆是。

+ 表示增加权限、- 表示取消权限、= 表示唯一设定权限。

r 表示可读取, w 表示可写入, x 表示可执行, X 表示只有当该文件是个子目录或者该文件已经被设定过为可执行。

-c 若该文件权限确实已经更改, 才显示其更改动作。

-f 若该文件权限无法被更改也不要显示错误讯息。

-v 显示权限变更的详细资料。

-R 对目前目录下的所有文件与子目录进行相同的权限变更。

范例:

- 设置某文件权限为只给自己运行, 别人只能读:

chmod u+x filename

- 设置某文件权限为同组的人可以执行:

chmod g+x filename

- 将文件 file1.txt 设为所有人皆可读取:

chmod ugo+r file1.txt 或 chmod a+r file1.txt

- 将文件 file1.txt 与 file2.txt 设为该文件拥有者、与其所属同一个群体者可写入, 但其他以外的人则不可写入:

chmod ug+w,o-w file1.txt file2.txt

- 将 ex1.py 设定为只有该文件拥有者可以执行:

chmod u+x ex1.py

- 将目前目录下的所有文件与子目录皆设为任何人可读取:

chmod -R a+r *

此外 chmod 也可以用数字来表示权限, 如 chmod 777 file。

语法为: chmod abc file。

其中 a,b,c 各为一个数字, 分别表示 User、Group、及 Other 的权限。r=4, w=2, x=1。

若要 rwx 属性则 4+2+1=7; 若要 rw-属性则 4+2=6; 若要 r-x 属性则 4+1=7。

范例:

- **chmod a=rwx file** 和 **chmod 777 file** 效果相同。
- **chmod ug=rwx,o=x file** 和 **chmod 771 file** 效果相同。
- 使某文件具有 root 的权限:

chmod 4755 filename

(ii) 改变文件拥有者属性的命令 **chown**

说明: 将指定文件的拥有者改为指定的用户或组。用户可以是用户名或用户 ID, 组可以是组名或组 ID。一般来说, 这个指令只能由系统管理者(root)使用, 一般使用者没有权限可以改变别人的文件拥有者, 也没有权限可以将自己的文件拥有者改为别人, 只有系统管理者(root)才有这样的权限。

例如 root 用户把自己的一个文件拷贝给用户 xu, 为了让用户 xu 能够存取这个文件, root 用户应该把这个文件的拥有者设为 xu, 否则, 用户 xu 无法存取这个文件。文件是以空格分开的要改变权限的文件列表, 支持通配符。

参数:

-R 递归式地改变指定目录及其下的所有子目录和文件的拥有者。

-v 显示 chown 命令所做的工作。

范例：

- 将文件 file1.txt 的拥有者设为 users 群体的使用者 jessie：
chown jessie:users file1.txt
- 将当前目录下的所有文件与子目录的拥有者皆设为 users 群体的使用者 lampont：
chmod -R lampont:users *

创建和删除目录的命令

(i) 创建目录命令 **mkdir**

说明：建立命令中指定名称的子目录。

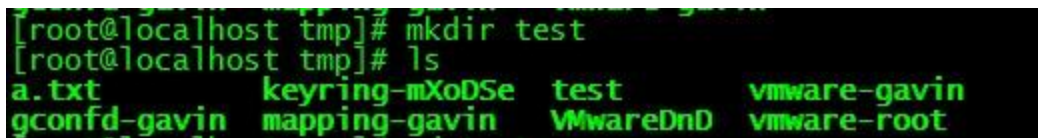
参数：

- p 确保目录名称存在，不存在的话就创建一个。

范例：

- 在工作目录下，建立一个名为 test 的子目录（如图 3.2）：

mkdir test



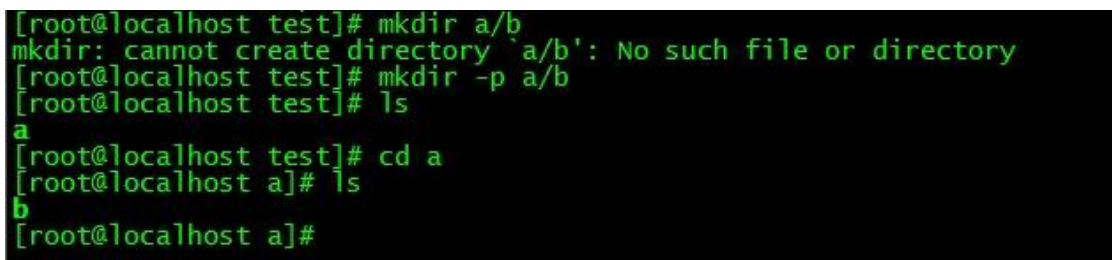
```
[root@localhost tmp]# mkdir test
[root@localhost tmp]# ls
a.txt          keyring-mXoD5e  test           vmware-gavin
gconfd-gavin  mapping-gavin  VMwareDnD     vmware-root
```

图 3.2 创建一个 test 子目录

- 在 a 目录下建立一个名为 b 的子目录。若 a 目录不存在，则建立一个：

mkdir -p a/b

注：本例若不加-p，且原本 BBB 目录不存在，则产生错误（如图 3.3）。



```
[root@localhost test]# mkdir a/b
mkdir: cannot create directory 'a/b': No such file or directory
[root@localhost test]# mkdir -p a/b
[root@localhost test]# ls
a
[root@localhost test]# cd a
[root@localhost a]# ls
b
[root@localhost a]#
```

图 3.3 创建多层目录

- 同时创建两个目录：

mkdir dir1 dir2

(ii) 删除空的目录命令 **rmdir**

说明：删除空的目录。

参数：

- p 子目录被删除后也成为空目录的话，则顺便把此子目录一并删除。

范例：

- 将工作目录下，名为 AAA 的子目录删除：

rmdir AAA

- 在工作目录下的 BBB 目录中，删除名为 Test 的子目录。若 Test 删除后，BBB 目录成为空目录，则 BBB 也删除。

rmdir -p BBB/Test

(iii) 删除文件或目录命令 **rm**

说明：删除一个目录中的一个或多个文件或目录，也可以将某个目录及其下的所有文件及子目录均删除。如果没有使用-r 参数，则 rm 不会删除目录。

参数:

-f 即使原文件属性设为只读, 也直接删除, 有强制的意思, 无需逐一确认。忽略不存在的文件, 不会出现任何警告信息。

-i 互动模式: 删除前, 会逐一询问是否删除。

-r 将参数中列出的全部目录和子目录均递归地删除。

范例:

- 删除所有 C 语言程序文件, 并且在删除前逐一询问确认:

rm -i *.c

- 将 Finished 子目录及子目录中所有文件删除 (rmdir 命令只能删除空目录, 删除非空目录则只能使用 rm -r):

rm -r Finished

创建、删除、重命名和复制文件命令

(i) 改变文件的时间记录命令 **touch**

说明: 改变文件的时间记录。命令 **ls -l** 可以显示文件的时间记录。

参数:

-a 改变文件的读取时间记录。

-m 改变文件的修改时间记录。

-c 假如目的文件不存在, 不会建立新的文件。与--no-create 的效果一样。

-f 不使用, 是为了与其他 unix 系统的相容性而保留。

-r 使用参考文件的时间记录。

-d 设定时间与日期, 可以使用各种不同的格式。

-t 设定文件的时间记录, 格式与 date 指令相同。

--no-create 不会建立新文件。

--help 列出指令格式。

--version 列出版本讯息。

范例:

- 将文件的时间记录改为现在的时间。若文件不存在, 系统会建立一个新的文件。

touch file

- 将 file 的时间记录改为 5 月 6 日 18 点 3 分, 公元两千年。时间的格式可以参考 date 指令, 至少需输入 MMDDHHmm, 即月日时与分。时间也可以使用 am, pm 或是 24 小时的格式, 日期也可以使用其他格式如 6 May 2000。

touch -c -t 05061803 file

touch -c -t 050618032000 file

touch -d "6:03pm" file

touch -d "05/06/2000" file

touch -d "6:03pm 05/06/2000" file

- 将 file 的时间记录改变成与 referencefile 文件一样:

touch -r referencefile file

(ii) 文本编辑器 **vi** (创建一个新的文件)

说明:

vi 有三种工作模式:

插入模式 (Insert mode): 用来输入和编辑文件的模式, 屏幕上会显示用户的键入, 按键不是被解释为命令执行, 而是作为文本写到用户的文件中。

底行模式 (last line mode): 将文件保存或退出 vi, 也可以设置编辑环境, 如寻找字符串、列出行号等。

命令行模式 (command mode): 控制屏幕光标的移动, 字符、字或行的删除, 移动复制某区段及进入

Insert mode 下，或者到 last line mode。

运行 vi 后，首先进入命令行模式。此时输入的任何字符都被视为指令，键入的命令不会在屏幕上显示。

从命令行模式切换到插入模式，则可以按“Insert”键或“i”键，就可以开始输入文字了。如果发现输错了字，想用光标键往回移动，将该字删除，就要切换回命令行模式，按一下“ESC”键，再删除文字。

状态行是屏幕底部一行，被 vi 编辑器用来反馈编辑操作结果。错误消息或提供信息的信息会在状态行中显示出来。

操作方式：

启动 vi:

在系统提示符下键入“vi”+“空格”+“文件名”，如：vi test.txt，vi 可以自动载入所要编辑的文件或是创建一个新文件。

退出 vi:

在命令行模式下键入“:”进入底行模式，如下命令可以退出 vi:

:q 如果用户只是读文件的内容而未对文件进行修改，可以在命令模式下输入“:q”退出 vi。

:q! 如果用户对文件的内容作了修改，又决定放弃对文件的修改，则用“:q!”命令。

:w! 强行保存一个 vi 文件，如果该文件已存在，则进行覆盖。

:wq 保存文件并退出 vi。

:w filename 相当于“另存为”。

:n,m w filename 将第 n-m 行的文本保存到指定的文件 filename 中。

命令行模式的功能键:

1) 插入模式

按[i]切换进入插入模式[Insert mode]，按“i”进入插入模式后是从光标当前位置开始输入文件；

按[a]进入插入模式后，是从目前光标所在位置的下一个位置开始输入文字；

按[o]进入插入模式后，是插入新的一行，从行首开始输入文字。

2) 从插入模式切换为命令行模式

按[ESC]键。

3) 移动光标

vi 可以直接用键盘上的光标来上下左右移动，但正规的 vi 是用小写英文字母[h]、[j]、[k]、[l]，分别控制光标左、下、上、右移一格。

按[ctrl]+[b]: 屏幕往“后”移动一页。

按[ctrl]+[f]: 屏幕往“前”移动一页。

按[ctrl]+[u]: 屏幕往“后”移动半页。

按[ctrl]+[d]: 屏幕往“前”移动半页。

按数字[0]: 移到文章的开头。

按[G]: 移动到文章的最后。

按[\$]: 移动到光标所在行的“行尾”。

按[^]: 移动到光标所在行的“行首”。

按[w]: 光标跳到下个字的开头。

按[e]: 光标跳到下个字的字尾。

按[b]: 光标回到上个字的开头。

按[#l]: 光标移到该行的第#个位置，如：5l,56l。

4) 删除文字

[x]: 每按一次，删除光标所在位置的“后面”一个字符。

[#x]: 例如，[6x]表示删除光标所在位置的“后面”6个字符。

[X]: 大写的 X，每按一次，删除光标所在位置的“前面”一个字符。

[#X]: 例如，[20X]表示删除光标所在位置的“前面”20个字符。

[dd]: 删除光标所在行。

[#dd]: 从光标所在行开始删除#行

5) 复制

[yw]: 将光标所在之处到字尾的字符复制到缓冲区中。

[#yw]: 复制#个字到缓冲区。

[yy]: 复制光标所在行到缓冲区。

[#yy]: 例如, [6yy]表示拷贝从光标所在的该行“往下数”6行文字。

[p]: 将缓冲区内的字符贴到光标所在位置。注意: 所有与“y”有关的复制命令都必须与“p”配合才能完成复制与粘贴功能。

6) 替换

[r]: 替换光标所在处的字符。

[R]: 替换光标所到之处的字符, 直到按下[ESC]键为止。

7) 回复上一次操作

[u]: 如果误执行一个命令, 可以马上按下[u], 回到上一个操作。按多次“u”可以执行多次回复。

8) 更改

[cw]: 更改光标所在处的字到字尾处。

[c#w]: 例如, [c3w]表示更改3个字。

9) 跳至指定的行

[ctrl]+[g]列出光标所在行的行号。

[#G]: 例如, [15G], 表示移动光标至文章的第15行行首。

底行模式下的命令简介:

在使用底行模式 (last line mode) 之前, 请先按[ESC]键确定已经处于命令行模式 (command mode) 下后, 再按[:]冒号即可进入 (last line mode)。

1) 列出行号

[set nu]: 输入[set nu]后, 会在文件中的每一行前面列出行号。

2) 跳到文件中的某一行

[#]: [#]号表示一个数字, 在冒号后输入一个数字, 再按回车键就会跳到该行了。如输入数字15, 再回车, 就会跳到文章的第15行。

3) 查找字符

[/]关键字]: 先按[/]键, 再输入想寻找的字符, 如果第一次找的关键字不是想要的, 可以一直按[n]会往后寻找到要的关键字为止。

[?]关键字]: 先按[?]键, 再输入想寻找的字符, 如果第一次找的关键字不是想要的, 可以一直按[n]会往前寻找到要的关键字为止。

4) 保存文件

[w]: 在冒号输入字母[w]就可以将文件保存起来。

5) 退出 vi

[q]: 按[q]就是退出, 如果无法离开 vi, 可以在[q]后跟一个[!]强制离开 vi。

[wq]: 一般建议离开时, 搭配[w]一起使用, 这样在退出的时候还可以保存文件。

(iii) 重命名或移动文件命令 mv

说明: 将一个文件重命名为另一文件, 或将数个文件移至另一目录下。

参数:.

-i 若目的地已有同名文件, 则先询问是否覆盖旧文件。

范例:

● 将文件 aaa 更名为 bbb:

mv aaa bbb

- 将所有的 C 语言程序文件移至 Finished 子目录中：

mv -i *.c

- 将当前目录下的 hello.txt 文件移动到 b 目录中，并改名为 hello1.txt（如图 3.4）：

mv hello.txt ./b/hello1.txt

```
[root@localhost test]# cd a
[root@localhost a]# touch hello.txt
[root@localhost a]# ls
b  hello.txt
[root@localhost a]# mv hello.txt ./b/hello1.txt
[root@localhost a]# cd b
[root@localhost b]# ls
hello1.txt
```

图 3.4 移动文件并改名

(iv) 文件复制命令 **cp**

说明：将一个文件复制成另一文件，或将数个文件复制到另一目录。

参数：

- a 尽可能将文件状态、权限等资料都照原状予以复制。
- r 若 source 中含有目录名，则将目录下的文件都依序复制到目的地。
- f 若目的地已经有相同名称的文件存在，则在复制前先予以删除再进行复制。

范例：

- 将文件 aaa 复制（此文件已存在），并命名为 bbb：
cp aaa bbb
- 将所有的 C 语言程序文件复制到 Finished 子目录中：
cp *.c Finished
- 将本目录下的 a.txt 文件复制到 test2 目录下，并重命名为 b.txt（如图 3.5）：
cp test1/a.txt test2/b.txt

```
[root@localhost tmp]# cp test1/a.txt test2/b.txt
[root@localhost tmp]# ls
a.txt  gconfd-gavin  keyring-mXoDSe  mapping-gavin  test1  test2  VMwareDnD  vmware-gavin  vmware-root
[root@localhost tmp]# cd test2
[root@localhost test2]# ls
b.txt
[root@localhost test2]# ll
```

图 3.5 复制文件并重命名

显示文件内容的命令

(i) 显示指定文件内容命令 **cat**

说明：将某个文件的内容显示出来，也可以把两个文件串连接后显示出来。屏幕上命令行加 “>filename” 链接到另一个文件。

参数：

- n 或 --number 由 1 开始对所有输出的行数编号。
- b 或 --number-nonblank 和 -n 相似，只不过对于空白行不编号。
- s 或 --squeeze-blank 当遇到有连续两行以上的空白行，换为一行的空白行。

范例：

- 把 textfile1 的文件内容加上行号后输入 textfile2 这个文件里：
cat -n textfile1 > textfile2
- 把 textfile1 和 textfile2 的文件内容加上行号（空白行不加）之后将内容附加到 textfile3 里：
cat -b textfile1 textfile2 >> textfile3
- 清空/etc/test.txt 文件的内容：
cat /dev/null > /etc/test.txt

- 参数-n 和-b 的区别如图 3.6:

```
[root@localhost test2]# cat -b /etc/issue
 1 CentOS release 5.5 (Final)
 2 Kernel \r on an \m

[root@localhost test2]# cat -n /etc/issue
 1 CentOS release 5.5 (Final)
 2 Kernel \r on an \m
 3
[root@localhost test2]#
```

图 3.6 cat 命令参数-n 和-b 的区别

(ii) 分页显示文件内容 **more**

说明：类似 cat 命令，不过会以一页一页的形式显示，方便使用者逐页阅读。最基本的指令就是按空白键（space）就往下一页显示；按 b 键就会往回（back）一页显示，按 Enter 键代表向下滚动一行；按 q 键表示离开 more 命令。

参数：

-num 一次显示的行数。

-d 提示使用者，在画面下方显示 [Press space to continue, 'q' to quit.]，如果使用者按错键，则会显示 [Press 'h' for instructions.]。

-f 计算行数时，以实际上的行数，而非自动换行过后的行数（有些单行字数太长的会被扩展为两行或两行以上）。

-p 不以卷动的方式显示每一页，而是先清除萤幕后再显示内容。

-c 跟-p 相似，不同的是先显示内容再清除其他旧资料。

-s 当遇到有连续两行以上的空白行，就代换为一行的空白行。

+num 从第 num 行开始显示。

范例：

- 逐页显示 testfile 文件的内容，如有连续两行以上空白行则以一行空白行显示。

more -s testfile

- 从第 20 行开始显示 testfile 文件的内容。

more +20 testfile

查找命令

(i) 查找指定目录或文件的命令 **find**

说明：将文件系统内符合命令中参数的文件列出来。可以指定文件的名称、类别、时间、大小、权限等不同信息的组合，只有完全相符的文件才会被列出来。

参数：

-mount, -xdev: 只检查和指定目录在同一个文件系统下的文件，避免列出其它文件系统下的文件。

-amin n: 在过去 n 分钟内被读取过的文件。

-anewer file: 比文件 file 更晚被读取过的文件。

-atime n: 在过去 n 天读取过的文件。

-cmin n: 在过去 n 分钟内被修改过的文件。

-cnewer file: 比文件 file 更新的文件。

-ctime n: 在过去 n 天修改过的文件。

-empty: 空的文件。

-ipath p, -path p: 路径名称符合 p 的文件，ipath 会忽略大小写。

-name name, -iname name: 文件名称符合 name 的文件，iname 会忽略大小写。

-size n: 文件大小是 n 单位, b 代表 512 位元组的区块, c 表示字元数, k 表示 kilo bytes, w 是二个位元组。

-type c: 文件类型是 c 的文件。类型如下:

- d: 目录
- c: 字型装置文件
- b: 区块装置文件
- p: 具名贮列
- f: 一般文件
- l: 符号连结
- s: socket

-pid n: process id 是 n 的文件。

范例:

- 将当前目录及其子目录下所有扩展名是 c 的文件列出来:
find . -name "*.c"
- 将当前目录及其下子目录中所有一般文件列出:
find . -ftype f
- 将当前目录及其子目录下所有最近 20 分钟内更新过的文件列出:
find . -ctime -20

(ii) 查找指定文件源和二进制文件和手册等 **whereis**

说明: **whereis** 指令会在特定目录中查找符合条件的文件。这些文件应是原始代码、二进制文件、或是帮助文件。

参数:

- b 只查找二进制文件。
- B<目录> 只在设置的目录下查找二进制文件。
- f 不显示文件名前的路径名称。
- m 只查找说明文件。
- M<目录> 只在设置的目录下查找说明文件。
- s 只查找原始代码文件。
- S<目录> 只在设置的目录下查找原始代码文件。
- u 查找不包含指定类型的文件。

挂载一个文件系统命令 **mount**

说明: 将某个文件的内容解读成文件系统, 然后将其挂在目录的某个位置之上。当这个命令执行成功后, 直到使用 **umount** 将这个文件系统移除为止, 这个命令之下的所有文件将暂时无法被调用。

参数:

- V 显示程序版本。
- h 显示辅助讯息。
- s-r 等于 -o ro
- w 等于 -o rw
- t 指定文件系统的型态, 通常不必指定。**mount** 会自动选择正确的型态。
- o async 打开非同步模式, 所有的文件读写动作都会用非同步模式执行。
- o sync 在同步模式下执行。
- o auto
- o noauto 打开/关闭自动挂上模式。
- o exec
- o noexec 允许执行文件被执行。

-o suid

-o nosuid 允许执行文件在 root 权限下执行。

-o user

-o nouser 使用者可以执行 mount/umount 的动作。

-o remount 将一个已经挂下的文件系统重新用不同的方式挂上。例如原先是只读的系统，现在用可读写的模式重新挂上。

-o ro 用只读模式挂上。

-o rw 用可读写模式挂上。

范例：

- 将/dev/hda1 挂在/mnt 之下：

mount /dev/hda1 /mnt

- 将/dev/hda1 用只读模式挂在/mnt 之下：

mount -o ro /dev/hda1 /mnt

二、设计性实验

1. 实验目的

(1) 了解 UP-CUP S2440 教学实验开发平台的硬件结构。

(2) 熟悉嵌入式 Linux 开发环境，学会基于 UP-CUP 经典 2440 教学科研平台的 Linux 开发环境的配置和使用。

(3) 利用 arm-linux-gcc 交叉编译器编译程序，使用基于 NFS 的挂载方式进行实验，了解嵌入式开发的基本过程。

(4) 能够在 UP-CUP S2440 教学实验平台上运行第一个程序。

2. 实验设备

(1) 硬件：UP-CUP S2440 嵌入式实验平台一套，PC 机 Pentium 500 以上，硬盘 40G 以上，内存大于 256M。

(2) 软件：Vmware Workstation8 虚拟机系统，Red Hat Linux 操作系统，MiniCom/Xshell 超级终端，ARM-LINUX 交叉编译开发环境。

3. 实验内容

本次实验使用 Red Hat Linux 操作系统环境，安装 ARM-Linux 的开发库及编译器。创建一个新目录，并在其中编写第一个程序文件 hello.c 和 Makefile 文件。

学习在 Linux 下的编程和编译过程，以及 ARM 开发板的使用和开发环境的设置。将已经编译好的文件通过 NFS 方式挂载到目标开发板上运行。

4. 实验步骤

(1) 连接 UP-CUP S2440 型开发板。

在 UP-CUP S2440 开发板和 PC 机之间连接好串口线和网线，连接好开发板的 12V 电源线。开发板上一共有两个串口，串口 RS232-0 和串口 RS232-1，一般默认的连接串口 RS232-0。连接好三条线后，按下 2440 核心板上的电源拨码开关启动开发板，在 PC 机的设备管理器中查明 Windows XP 分配给开发板的串口设备号，如“COM3”等，在下一步配置串口超级终端中需要使用此串口设备号来连接开发板。

(2) 配置串口超级终端。

开发板主要使用串口 RS232-0 来进行信息输出与反馈，因此可以在 PC 机端使用一些串口工具连接开发板的串口，观察开发板串口的输出信息，以及在串口终端中控制开发板上程序的运行。本实验我们采用 PC 机上已事先装好的 Xshell 超级终端工具，下面我们只需要对它进行一些必要的配置以连接 UP-CUP S2440 开发板。

点击[开始]->[程序]->[Xmanager Enterprise]->[Xshell]，运行 Xshell 超级终端工具，如图 3.7 所示：

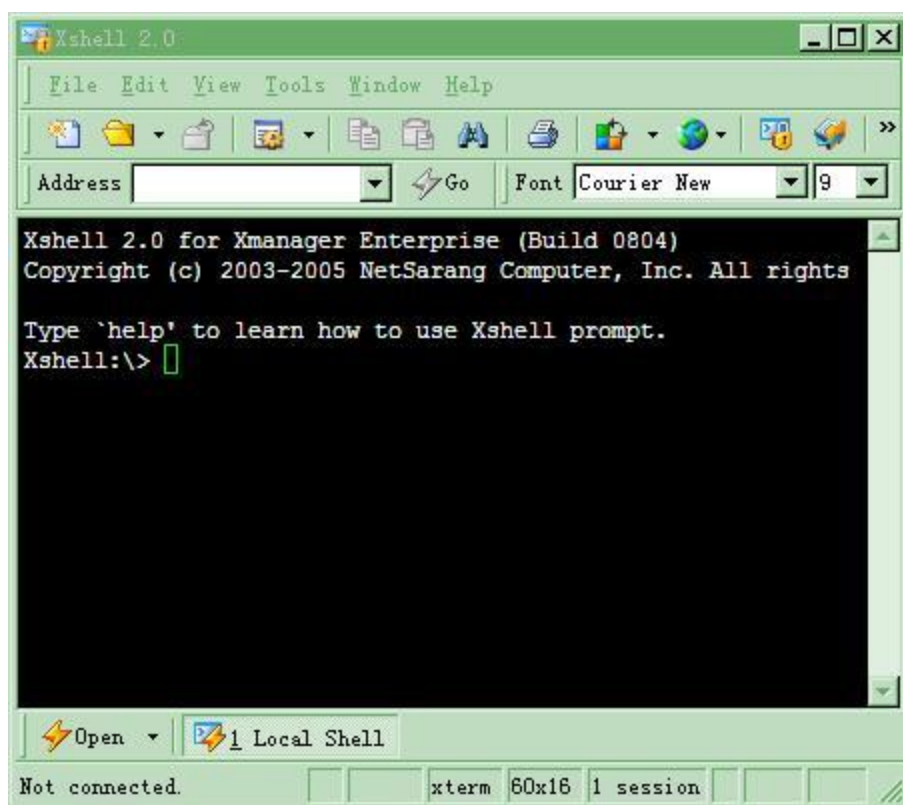


图 3.7 运行 Xshell 超级终端工具

点击[File]->[New...], 打开“New Session 属性”对话框, 新建一个 Session 连接开发板, “Name”输入“COM2440”, “Method”选择“SERIAL”串口连接, 如图 3.8:

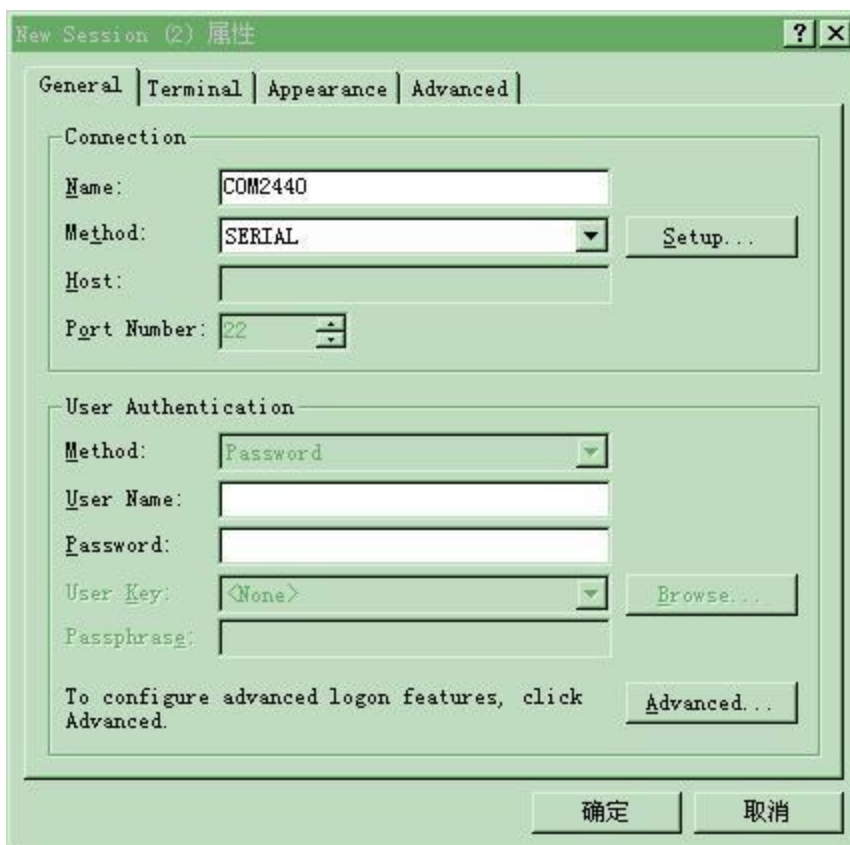


图 3.8 新建一个 Session

点击“Setup”按钮，打开“Advanced Serial Options”对话框，继续配置此连接。Port 端口选择“COM4”，即在上一步骤中查看好的 Windows XP 分配给开发板的串口设备号，请根据实际硬件连接选择。Baud Rate 波特率选择“115200”，Data Bits 数据位选择“8”位，Stop Bits 停止位选择“1”位，Parity 校验位选择“None”，如图 3.9：

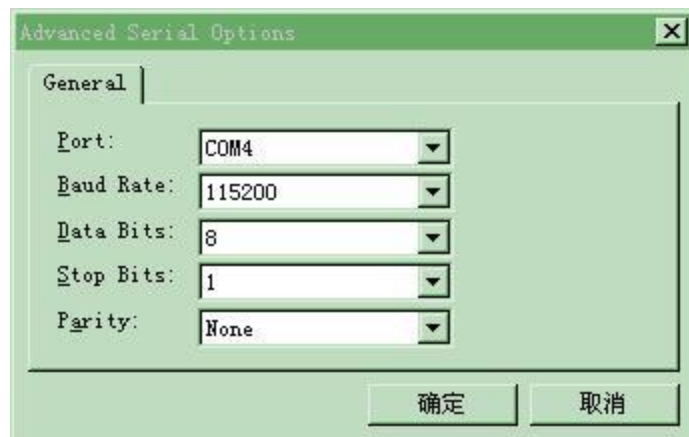


图 3.9 配置新建 Session 属性

点击确定，“Session”窗口新建了一个会话 COM2440，点击“Connect”按钮，连接开发板，如图 3.10：

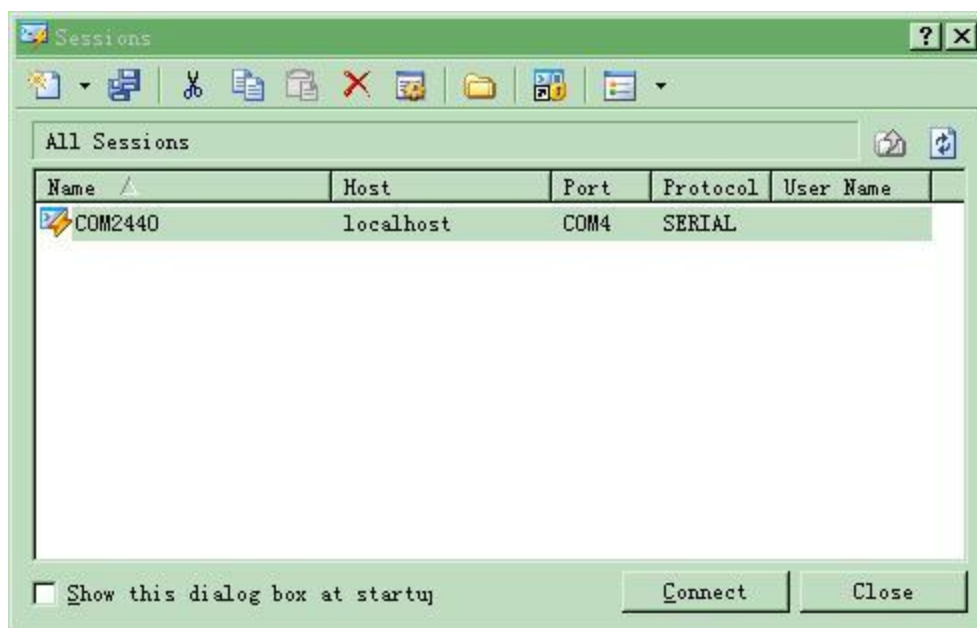


图 3.10 新建 Session 成功

由于步骤(1)已连接好开发板，可以重新手动按下 2440 核心板上的电源拨码开关，重新启动开发板，如果连接成功，超级终端显示系统启动的串口信息如图 3.11 所示：

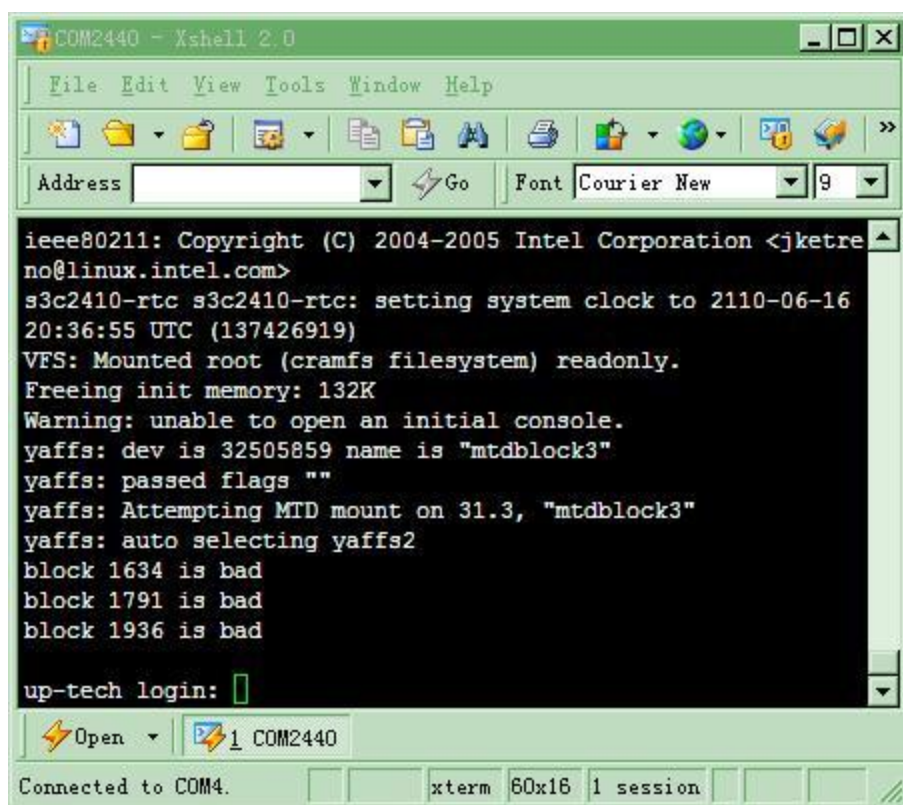


图 3.11 Xshell 连接开发板成功

在以后的实验中采用 Xshell 超级终端工具作为范例进行说明。系统上电后，进入 BOOTLOADER 引导 LINUX 内核，如果在上电后的 3 秒内按下回车键即可进入 U-BOOT 控制界面（该功能界面可以实现系统内核、文件系统的烧写），否则系统启动后默认引导 LINUX 操作系统，如图 3.12:

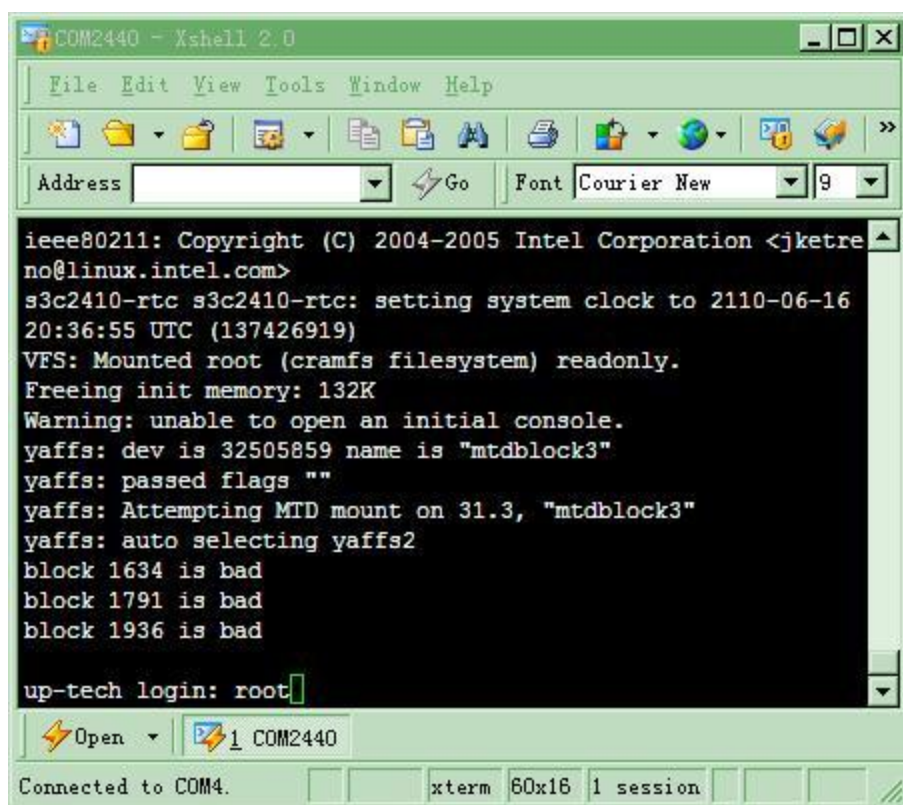


图 3.12 登陆 ARM 设备端 LINUX 操作系统

接着输入 root 用户，会自动登陆到/root 目录下，随机附带的接口测试程序存放在开发板的/root 目录中，如图 3.13 的登陆信息：

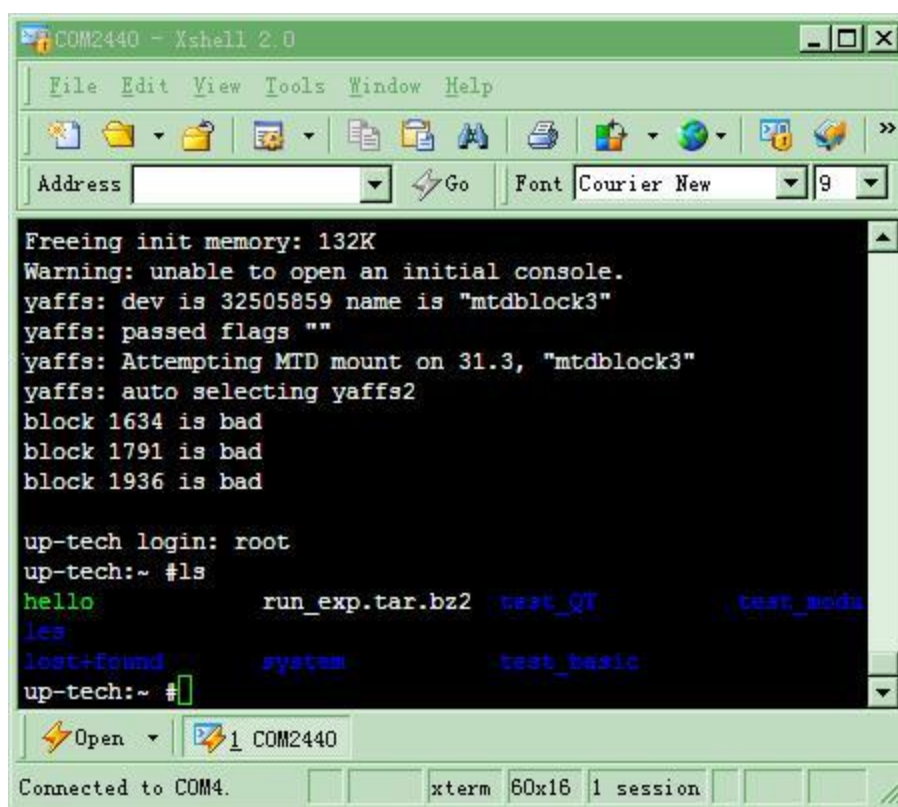
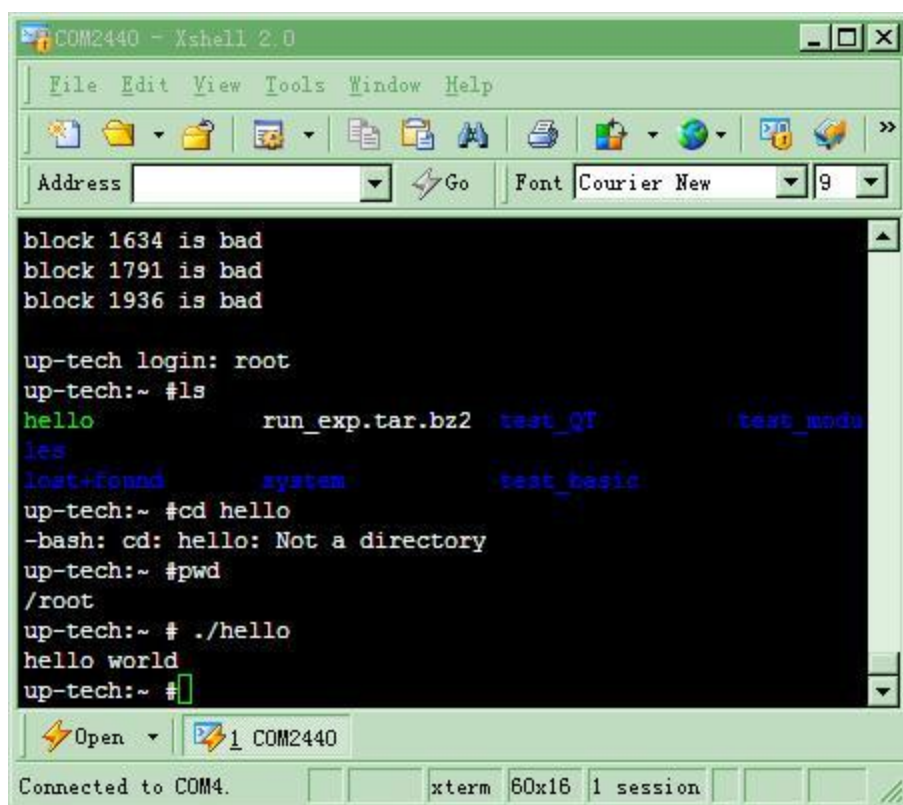


图 3.13 附带测试程序

此时可以看到在 up-tech:~根目录下开发板自带了一个名为“hello”的例子程序，可以用以下命令试着运行此程序，运行结果是屏幕输出一行语句“hello world”，配置超级终端连接开发板成功。如图 3.14。

```
up-tech:~# ./hello
```



```
COM2440 - Xshell 2.0
File Edit View Tools Window Help
Address [ ] Go Font Courier New 9
block 1634 is bad
block 1791 is bad
block 1936 is bad

up-tech login: root
up-tech:~ #ls
hello          run_exp.tar.bz2  test_QT        test_woda
ls
test_found     system          test_basic
up-tech:~ #cd hello
-bash: cd: hello: Not a directory
up-tech:~ #pwd
/root
up-tech:~ # ./hello
hello world
up-tech:~ #
```

图 3.14 例子程序 hello 的运行结果

(3) 开启虚拟机 VM8，运行 Ret Hat Linux 操作系统，输入用户名和口令进入操作系统界面。详细步骤参照基础实验的步骤(1)。

(4) 设置开发板、虚拟机、Windows XP 的 IP 地址。

首先打开超级终端，输入 ifconfig 命令，此时可以看到厂商已设好的开发板网络 IP 为 192.168.1.193，如图 3.15：

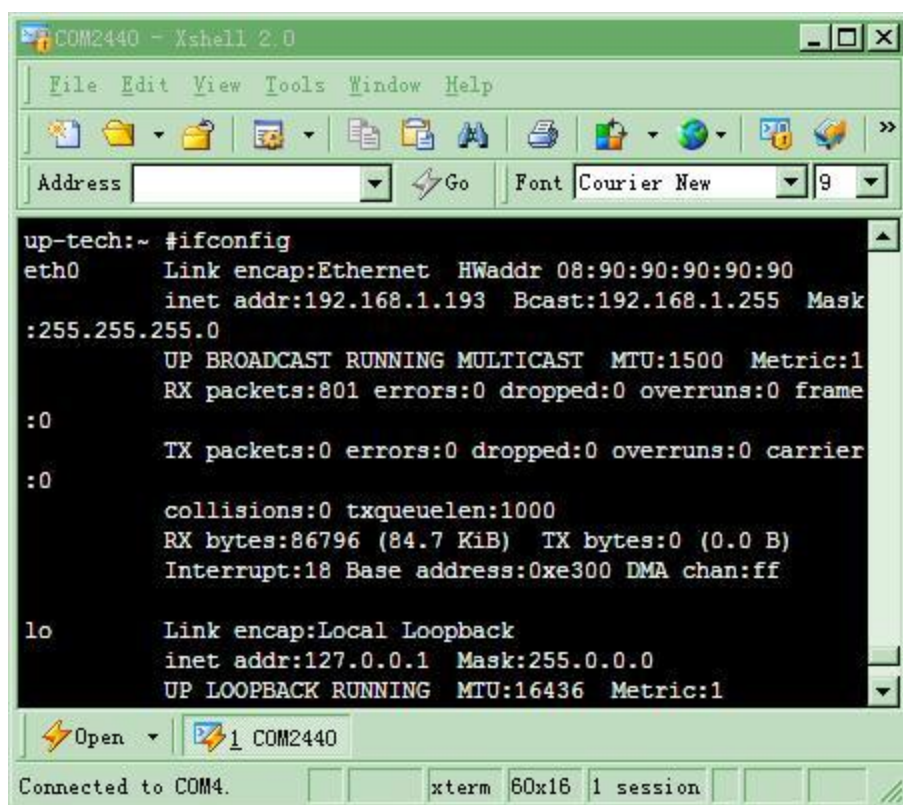


图 3.15 查看开发板的端口网络地址

接下来设置虚拟机的 IP 地址。在虚拟机的 Linux 操作系统中点击左下方红帽子标志->[系统设置]->[网络]，打开“网络配置”对话框设置 IP，如图 3.16：

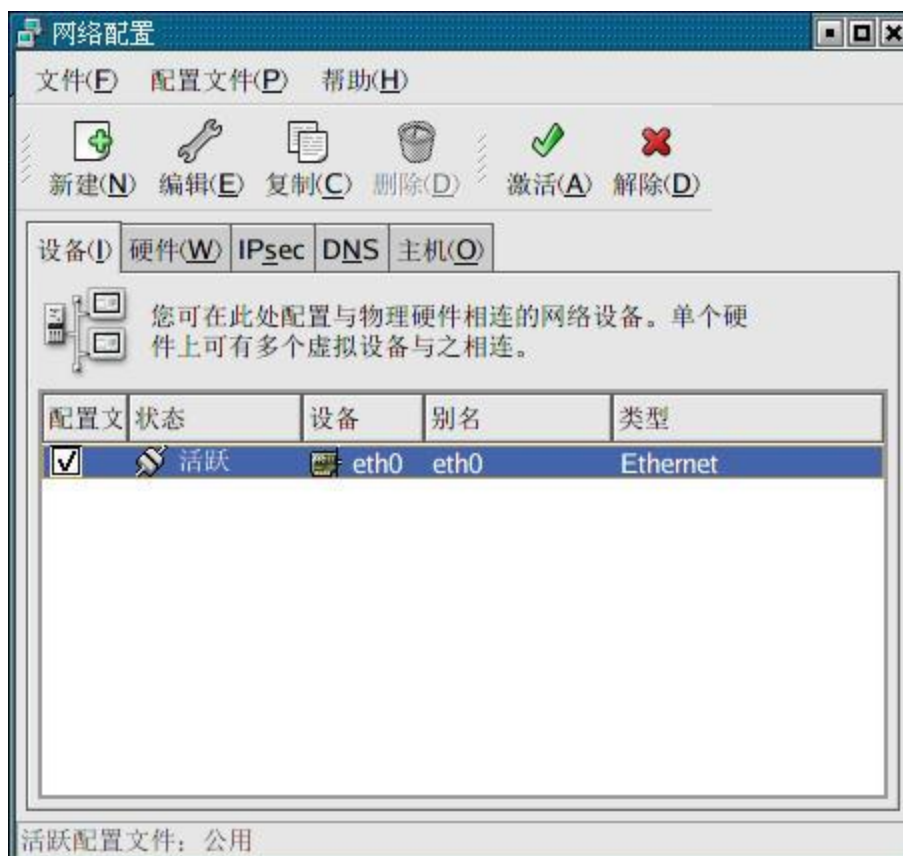


图 3.16 “网络配置”对话框

双击状态为活跃的配置，打开“以太网设置”对话框配置其属性。勾选“当计算机启动时激活设备”复选框和“允许所有用户启用和禁用该设备”复选框，按照图 3.17 手工设置 IP 地址、子网掩码和默认网关，确保此虚拟机的 IP 地址要与开发板的 IP 地址同属于一个码段。

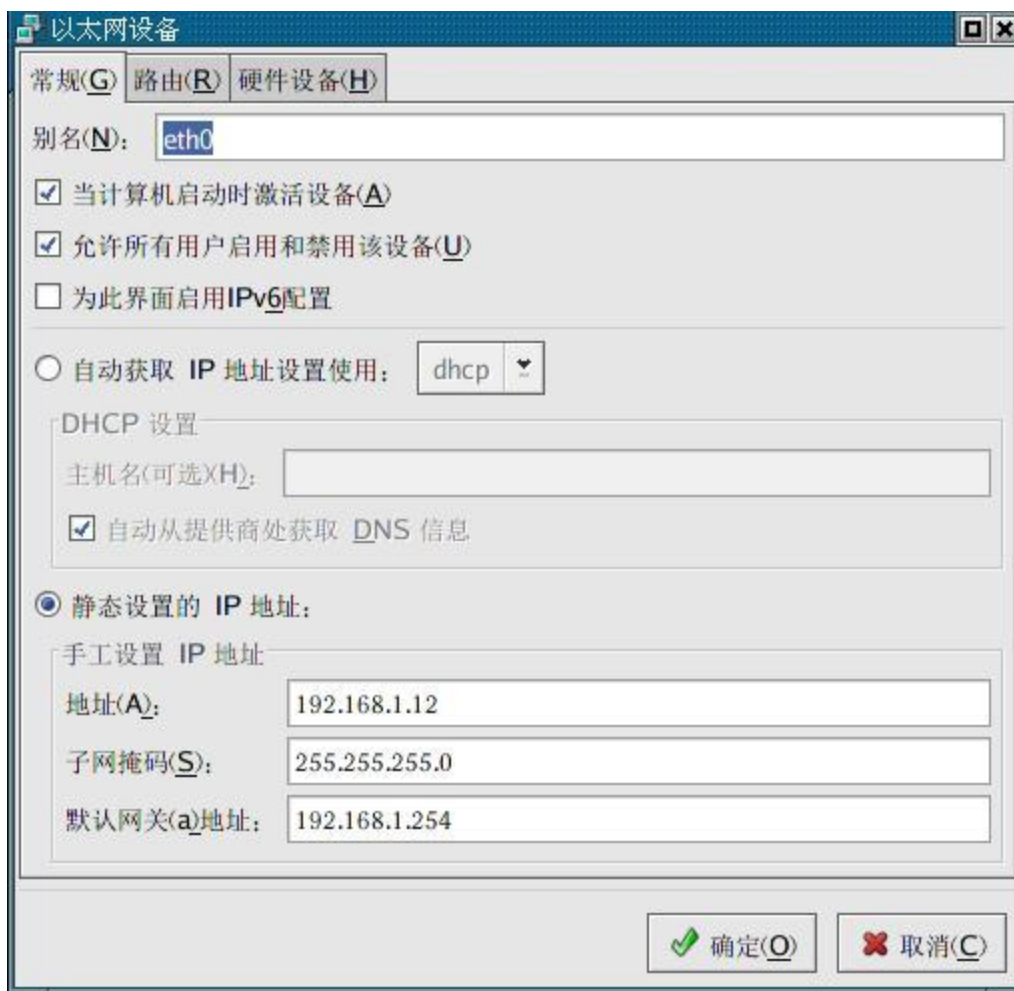



图 3.17 配置虚拟机 IP 地址

配置完毕后，点击确定保存设置，回到“网络配置”对话框，点击  按钮，并对修改进行保存，虚拟机的 IP 地址配置完成。

最后设置 Windows XP 的 IP 地址。右击桌面“网络邻居”图标，点击“属性”打开“本地连接 属性”对话框，双击“Internet 协议 (TCP/IP)”设置 Windows XP 的 IP 地址，如图 3.18，也要确保设置的 IP 地址与开发板的 IP 地址同属于一个码段。点击确定保存配置。

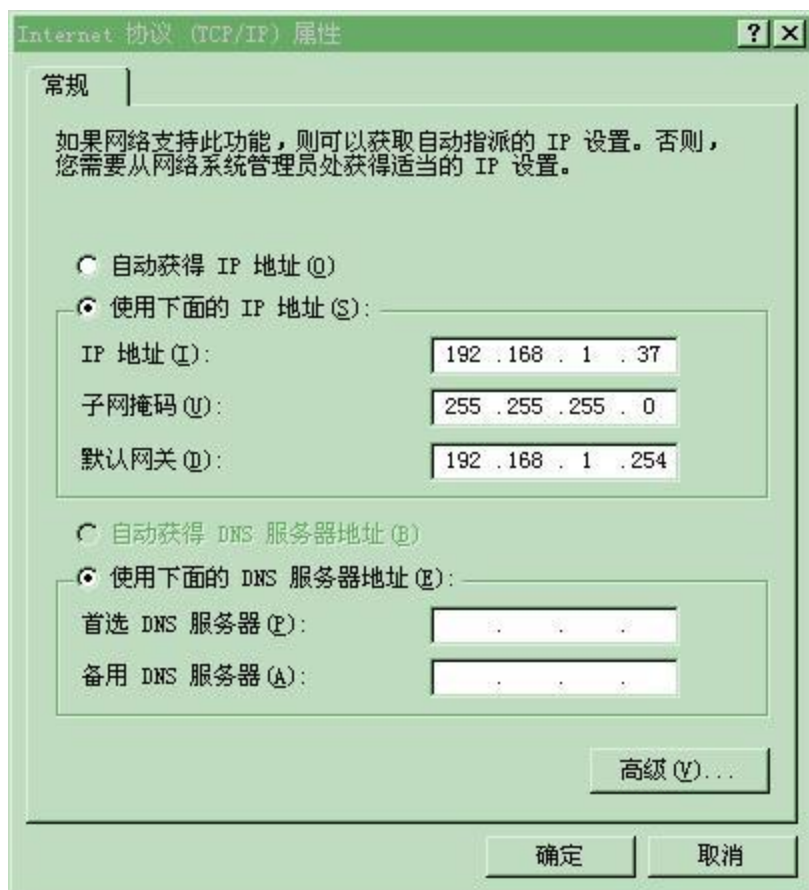


图 3.18 Windows XP 的 IP 地址设置

此时打开超级终端，输入 ping 命令可以检验网络是否完好，即检查开发板、虚拟机、Windows 三方的网络是否连通。通过 Ctrl+C 命令终止 ping 命令。如图 3.19。

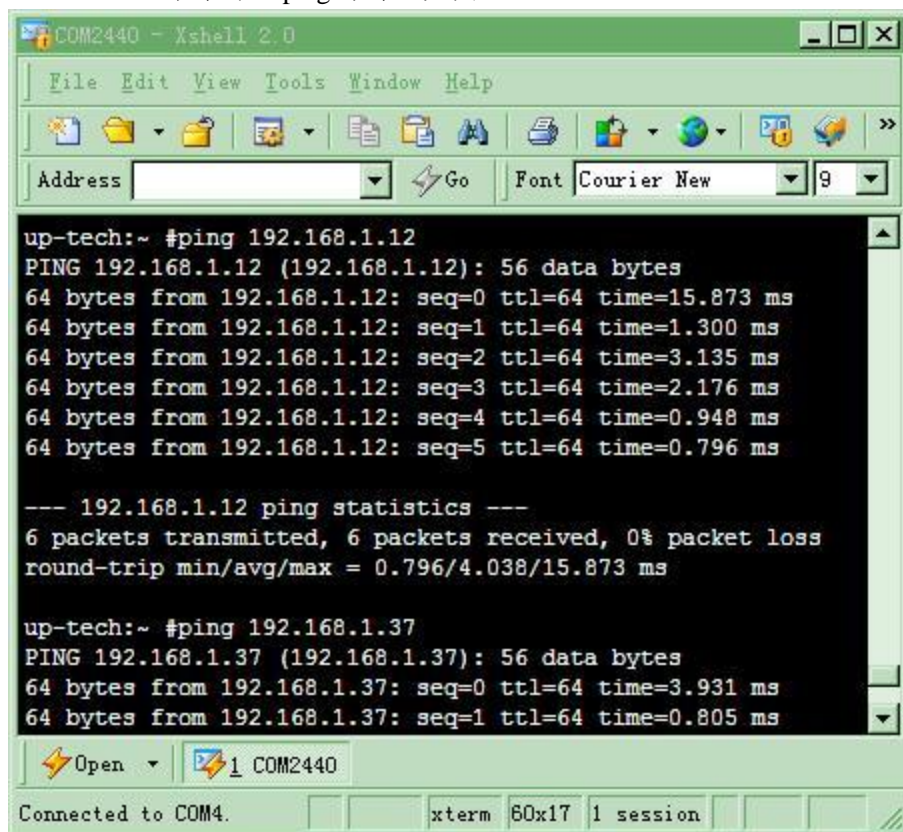


图 3.19 开发板连通虚拟机、Windows XP

注意：

如果在超级终端输入 ping 命令无法连通虚拟机或 Windows，请尝试以下三种方法使得连通成功：

- ① 确保 Windows 的无线网络已关闭，而本地连接是畅通的；
- ② 在 VM8 虚拟机的终端下输入以下命令重启网络服务，以确保虚拟机设置的 IP 地址有效：

```
[root@vm-dev ~]# /etc/init.d/network restart
```

- ③ 点击 VM8 虚拟机菜单栏的[虚拟机]->[设置]，打开“虚拟机设置”对话框，点击“硬件”标签内的“网络适配器”，确保右侧显示出来的网络连接方式为“桥接”。

- (5) 拷贝 UP-CUP S2440 型产品光盘的代码资源到虚拟机及交叉编译器的安装。

UP-CUP S2440 开发平台光盘内的 UP-CUP2440 Linux 目录下有自动安装脚本执行文件 install.sh，该执行文件用于初次在虚拟机 Linux 上自动安装光盘的代码资源及交叉编译环境。将 UP-CUP S2440 型开发平台附带开发工具光盘插入 CDRROM，用 cd 命令进入光盘的 UP-CUP2440 Linux 目录下，用以下命令执行 install.sh 脚本，即可在虚拟机上安装产品交叉编译环境。

```
[root@vm-dev UP-CUP2440]# ./install.sh
```

安装完毕后将在虚拟机 Linux 的根目录上产生/UP-CUP2440 光盘目录，该目录下存放开发板配置资源代码及工具环境，同时交叉编译器的解压及安装实际上也已经在 install.sh 脚本执行的时候安装完毕。如图 3.20。

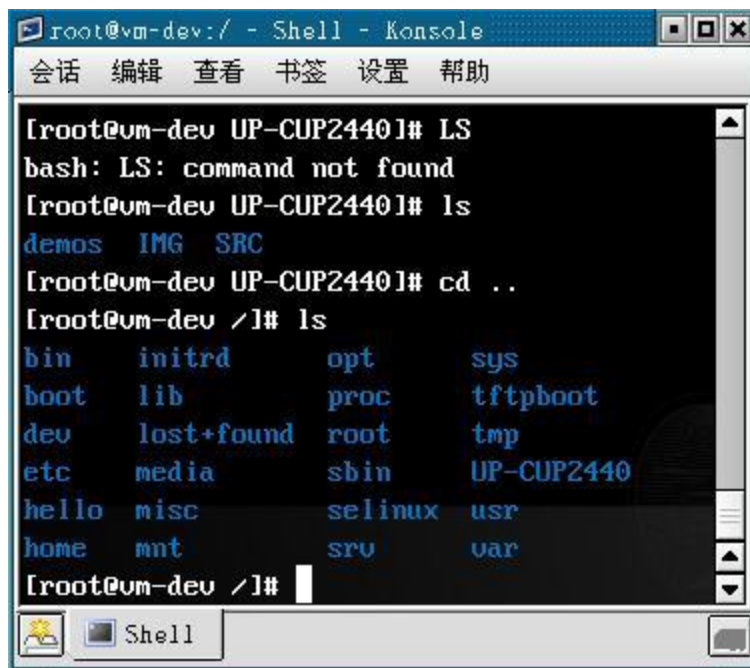


图 3.20 虚拟机根目录增加 UP-CUP2440 目录

- (6) 配置 Samba 服务。

Samba 服务主要用于在虚拟机 Linux 与 Windows XP 之间建立共享目录以实现通讯。在虚拟机的 Linux 操作系统中点击左下方红帽子标志->[系统设置]->[服务器设置]->[Samba]，打开“Samba 服务器配置”对话框，如图 3.21 所示：

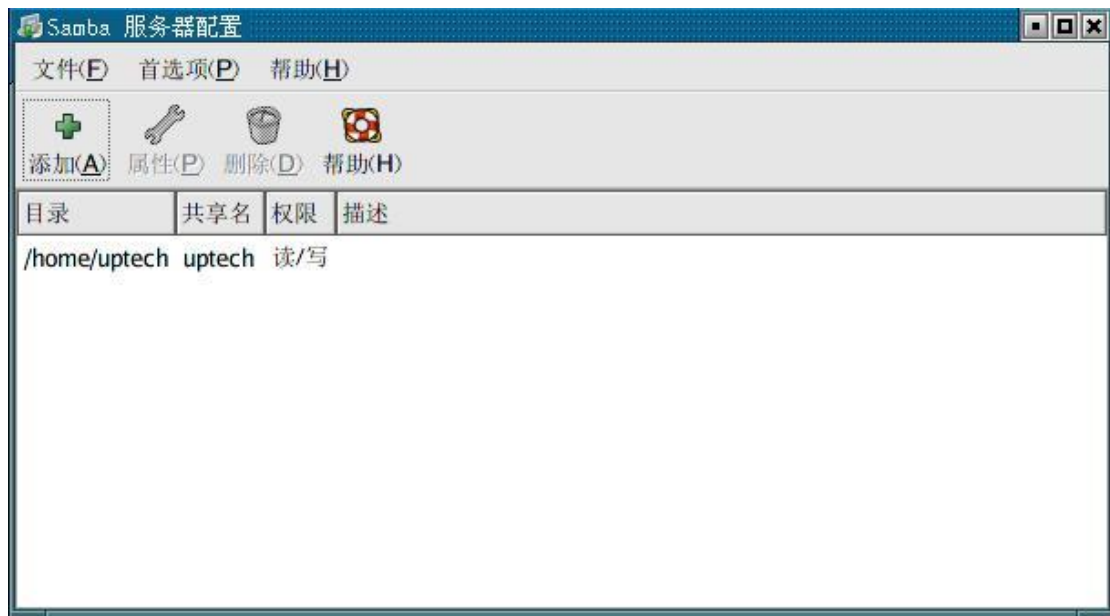


图 3.21 Samba 服务器配置对话框

可以看到已存在一个默认的共享目录/home/uptech，权限为读/写，也可以新建别的共享目录，本例使用此默认的共享目录。选中此目录，点击“属性”按钮，打开“编辑 Samba 共享”对话框，确保各个属性设置正确，如图 3.22、3.23：



图 3.22 配置 Samba 共享目录属性 1



图 3.23 配置 Samba 共享目录属性 2

Samba 服务配置完毕后，在 Windows XP 下的[开始]->[运行]输入“\\192.168.1.12”，即虚拟机的 IP 地址，这样就通过网络访问模式从 Windows 查看到了虚拟机的共享目录 uptech，用户就可以把 Windows 下的资源拷贝到虚拟机下。如图 3.24:



图 3.24 从 Windows 查看虚拟机的共享目录 uptech

至此，嵌入式 Linux 的开发环境已全部建立完毕。下面我们开始在 Linux 下编写第一个程序文件 hello.c 和 Makefile 文件，并将已经编译好的文件通过 NFS 方式挂载到目标开发板上运行。

(7) 在虚拟机端的 UP-CUP2440 目录下建立工作目录 hello。

在虚拟机的终端下依次输入以下命令建立一个目录 hello，并进入此新建的目录下：

```
[root@vm-dev /]# cd UP-CUP2440
[root@vm-dev UP-CUP2440]# mkdir hello
[root@vm-dev UP-CUP2440]# cd hello
```

(8) 编写程序源代码。

在 Linux 下的文本编辑器有许多，在开发过程中推荐使用 vim/vi，请参阅基础实验的 vi 编辑器的操作方法。使用以下命令来编写 hello.c 的源代码：

```
[root@vm-dev hello]# vi hello.c
```

进入 vi 编辑器后，按“i”或者“a”进入编辑模式，本例中的第一个程序 `hello.c` 的源代码非常简单，请参阅“5. 实验参考程序”。将代码录入完毕后，按“Esc”键进入命令行模式，再用命令“:wq”保存并退出。这样便在当前目录下建立了一个名为 `hello.c` 的文件。

(9) 编写 Makefile 文件。

Makefile 文件是在 Linux 系统下进行程序编译的规则文件，通过 Makefile 文件来指定和规范程序编译和组织的规则。

Makefile 文件的具体内容，可以在 Xshell 超级终端通过以下命令找到本次实验目录下的 Makefile 文件，用 vi 编辑器查看其中的内容。

```
[root@vm-dev /]# cd/UP-CUP2440/SRC/exp/basic/01_hello/
```

```
[root@vm-dev 01_hello]# ls
```

```
hello  hello.c  hello.o  Makefile
```

Makefile 文件的内容如下：

*****Makefile 源文件*****

```
TOPDIR = ../
```

```
include $(TOPDIR)Rules.mak
```

```
EXEC = hello
```

```
OBJS = hello.o
```

```
all: $(EXEC)
```

```
$(EXEC): $(OBJS)
```

```
$(CC) $(LD_FLAGS) -o $@ $(OBJS)
```

```
install:
```

```
$(EXP_INSTALL) $(EXEC) $(INSTALL_DIR)
```

```
clean:
```

```
-rm -f $(EXEC) *.elf *.gdb *.o
```

与上面编写 `hello.c` 的过程类似，用 vi 编辑器来创建一个 Makefile 文件并将代码录入其中：

```
[root@vm-dev hello]# vi Makefile
```

(10) 编译应用程序。

上面的步骤完成后，接着就可以在 `hello` 目录下运行 `make` 命令来编译本例中的第一个程序。如果对源文件进行了修改，重新编译再运行：

```
[root@vm-dev hello]# make clean
```

```
[root@vm-dev hello]# make
```

`make clean` 命令在第一次编译程序时候无需使用，在多次编译程序的时候可以用该命令来清除上次编译程序过程中生成的中间文件。这样可以避免一些非改动的 `make` 编译错误提示。

注意：编译、修改程序都是在虚拟机（PC 机 VM8 下的 Red Hat Linux）上进行，不能在 ARM 终端即开发板上进行。

(11) 运行编译好的程序。

配置 NFS 共享目录的步骤如下：

在虚拟机的 Linux 操作系统中点击左下方红帽子标志->[系统设置]->[服务器设置]->[NFS]，打开“NFS 服务器配置”对话框，如图 3.25 所示：

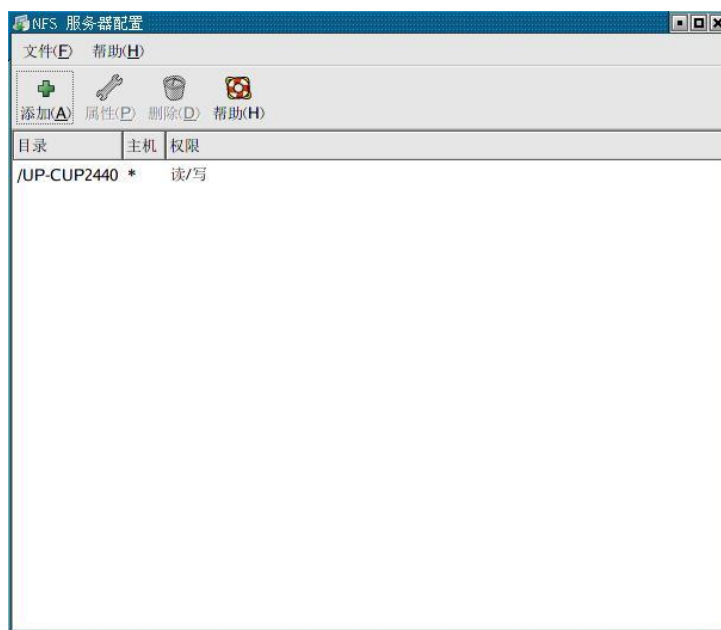


图 3.25 NFS 配置对话框

可以看到已存在一个默认的共享目录/UP-CUP2440，权限为读/写，也可以新建别的共享目录，本例使用此默认的共享目录。选中此目录，点击“属性”按钮，打开“编辑 NFS 共享”对话框，确保各个属性设置正确，如图 3.26、3.27：



图 3.26 配置 NFS 共享目录属性 1



图 3.27 配置 NFS 共享目录属性 2

如果没有默认的共享目录，点击“添加”按钮，添加一个 NFS 共享目录。各个参数说明如下：

目录：设置虚拟机端共享目录文件夹。例如，使用产品光盘安装后产生的光盘目录：/UP-CUP2440，该 NFS 共享目录在后续实验都将用到，建议新创建的 NFS 目录与本实验指导一致，方便后续实验的进行。

主机：可以访问该共享目录的机器的 IP 地址，例如：*，指所有的主机均允许访问。

基本权限：读写权限。

点击确定按钮，成功建立了一个新的 NFS 共享目录。

虚拟机端 NFS 服务目录建立好后，即可通过 Xshell 超级终端在 UP-CUP2440 平台端运行 mount 命令挂载虚拟机端的 NFS 共享目录。在 Xshell 超级终端输入以下命令：

```
up-tech:~ #mount -t nfs -o nolock,rsize=4096,wsize=4096 192.168.1.12:/UP-CUP2440 /mnt/nfs/
```

注意：192.168.1.12 为虚拟机端的 IP 地址。/UP-CUP2440 目录为虚拟机端 NFS 共享目录，/mnt/nfs 目录为开发板端临时挂载目录。

挂载成功后即可在开发板的/mnt/nfs 下访问虚拟机的/UP-CUP2440 目录下的文件内容。在超级终端输入以下命令可以看到开发板端的/mnt/nfs 目录和虚拟机端的/UP-CUP2440 目录内容是一样的，说明挂载成功：

```
up-tech:~ #cd /mnt/nfs
```

```
up-tech:/mnt/nfs #ls
```

```
IMG      SRC      demos
```

如果挂载失败，而且使用 ping 命令测试虚拟机与开发板通讯正常，可以在虚拟机端使用如下命令关闭默认路由，之后再重新挂载。

```
[root@vm-dev ~]#route del default
```

接下来我们可以进入超级终端的 NFS 共享实验目录运行编译好的程序。在 Xshell 超级终端输入以下命令进入/mnt/nfs 目录下的实验目录，运行刚刚编译好的 hello 程序，查看运行结果。

```
up-tech:/ #cd /mnt/nfs/hello/
```

```
up-tech:/mnt/nfs/hello #ls
```

```
Makefile  hello      hello.c    hello.o
```

执行程序用./表示执行当前目录下 hello 程序，执行结果为在超级终端的屏幕上输出一行字符：“hello world”：

```
up-tech:/mnt/nfs/hello #./hello
```

```
hello world
```

5. 实验参考程序

*****hello.c 源文件*****

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

```
main()
```

```
{  
    printf("hello world\n");  
}
```

6. 实验思考题

- (1) **Makefile** 文件是如何工作的？其中的宏定义分别是什么意思？
- (2) 简述嵌入式 **Linux** 开发的一般流程。

实验四 直流电机及 4*4 矩阵键盘驱动实验

预习要求：

- (1) 阅读直流电动机脉宽调制 (PWM) 电路原理的相关资料。
- (2) 了解 UP-CUP2440 教学实验开发平台的硬件结构。
- (3) 阅读模块驱动方法的相关资料，掌握交叉编译的概念及方法，学习键盘驱动的方法。

一、 基础性实验

1. 实验目的

- (1) 熟悉 ARM 本身自带的脉宽调制 PWM，掌握相应寄存器的配置。
- (2) Linux 下编程实现 ARM 系统的 PWM 输出，从而控制直流电机。
- (3) 了解直流电机的工作原理，学会用软件的方法实现步进电机的脉冲分配。
- (4) 掌握带有 PWM 的 CPU 编程实现其相应功能的主要方法。

2. 实验设备

- (1) 硬件：UP-CUP2440 型嵌入式实验平台，PC 机 Pentium 500 以上，硬盘 40G 以上，内存大于 256M。
- (2) 软件：Vmware Workstation8 虚拟机系统，Red Hat Linux 操作系统，MiniCom/Xshell 超级终端，

ARM-LINUX 交叉编译开发环境。

3. 实验内容

驱动直流电机的驱动模块，编写应用程序实现控制直流电机的转动速度和转动方向。

学习直流电机的工作原理，了解实现电机转动对于系统的软件和硬件要求。学习 ARM PWM 的生成方法。使用 Linux 操作系统环境及 ARM 编译器，编译直流电机的驱动模块和应用程序。运行程序以实现直流电机的调速转动。

4. 实验步骤

- (1) 启动虚拟机运行 Red Hat Linux 操作系统，在虚拟机的终端输入以下命令，在 UP-CUP2440 目录下建立工作目录 07_dcmotor：

```
[root@vm-dev UP-CUP2440]# mkdir 07_dcmotor
[root@vm-dev UP-CUP2440]# cd 07_dcmotor
```

- (2) 编写程序源代码。

在虚拟机的 Linux 下用文本编辑器 vim 或 vi 输入程序的源代码，关于 vi 编辑器的使用方法请参照实验三的基础实验。使用下面的命令来编写 dcm_main.c 的源代码，进入 07_dcmotor 目录后，使用 vi 命令来编辑代码：

```
[root@vm-dev 07_dcmotor]# vi dcm_main.c
```

按“i”或者“a”进入编辑模式，输入源代码，请参阅本次实验的实验目录：UP-CUP2440/SRC/exp/basic/07_dcmotor 下的同名源文件。完成后按 Esc 键进入命令行模式，再用命令“:wq”保存并退出。这样便在当前目录下建立了一个名为 dcm_main.c 的文件。

- (3) 编写 Makefile 文件。

与上一步编写 dcm_main.c 的过程类似，在 07_dcmotor 目录下用 vi 命令来创建一个 Makefile 文件并将代码录入到其中，具体源代码也请参阅本次实验的实验目录：UP-CUP2440/SRC/exp/basic/07_dcmotor 下的同名源文件。

```
[root@vm-dev 07_dcmotor]# vi Makefile
```

- (4) 编译应用程序。

上面的步骤完成后，就可以在 07_dcmotor 目录下运行“make”命令来编译程序，如果进行了修改，重新编译再运行，如图 4.1：

```
[root@vm-dev 07_dcmotor]# make clean
[root@vm-dev 07_dcmotor]# make

[root@vm-dev 07_dcmotor]# make clean
rm -f *.o a.out da *.gdb dcm_main
[root@vm-dev 07_dcmotor]# make
arm-linux-gcc -c -o dcm_main.o dcm_main.c
arm-linux-gcc dcm_main.o -o dcm_main
[root@vm-dev 07_dcmotor]# ls
dcm_main dcm_main.c dcm_main.o drivers Makefile
[root@vm-dev 07_dcmotor]#
```

图 4.1 虚拟机的 Linux 终端下编译程序

(5) NFS 挂载实验目录测试。

启动 UP-CUP2440 实验系统，连好网线、串口线和电源线。通过串口的 Xshell 超级终端挂载虚拟机的实验目录 UP-CUP2440。

```
up-tech:~ #mount -t nfs -o nolock,rsize=4096,wsize=4096 192.168.1.12:/UP-CUP2440 /mnt/nfs/
```

(6) 进入串口终端的 NFS 共享实验目录。

使用以下命令进入 NFS 挂载的实验目录，在超级终端的/mnt/nfs 目录下：

```
up-tech:/ # cd /mnt/nfs/SRC/exp/basic/07_dcmotor/
up-tech:/mnt/nfs/SRC/exp/basic/07_dcmotor # ls
Makefile dcm_main dcm_main.c dcm_main.o drivers
```

```
up-tech:~ #cd /mnt/nfs/SRC/exp/basic/07_dcmotor/
up-tech:/mnt/nfs/SRC/exp/basic/07_dcmotor #ls
Makefile dcm_main dcm_main.c dcm_main.o drivers
up-tech:/mnt/nfs/SRC/exp/basic/07_dcmotor #
```

(7) 使用 insmod 命令加载驱动。

在超级终端下用以下命令加载驱动程序：

```
up-tech:/mnt/nfs/SRC/exp/basic/07_dcmotor # insmod drivers/s3c2440-dc-motor.ko
s3c2440-dc-motor device initialized
up-tech:/mnt/nfs/SRC/exp/basic/07_dcmotor #insmod drivers/s3c2440-dc-motor.ko
s3c2440-dc-motor device initialized
up-tech:/mnt/nfs/SRC/exp/basic/07_dcmotor #
```

(8) 执行程序。

执行程序用./表示执行当前目录下的 dcm_main 程序，命令如下：

```
up-tech:/mnt/nfs/SRC/exp/basic/07_dcmotor # ./dcm_main
```

执行结果如下图，按下 Ctrl+C 停止程序：


```

up-tech:/mnt/nfs/SRC/exp/basic/07_dcmotor #./dcm_main
S3c2440 DC Motor device open now!
setpwm = -512
setpwm = -511
setpwm = -510
setpwm = -509
setpwm = -508
setpwm = -507
setpwm = -506
setpwm = -505
setpwm = -504
setpwm = -503
setpwm = -502
setpwm = -501
setpwm = -500
setpwm = -499
setpwm = -498
setpwm = -497
setpwm = -496
setpwm = -495
S3c2440 DC Motor device release!

```

5. 实验思考题

- (1) 简述 PWM 的基本原理，思考其基本参数的变化对电机转动有什么影响？
- (2) 尝试使用实验箱上的电位器旋钮控制直流电机的转向和转速。

二、 设计性实验

1. 实验目的

- (1) 了解 4*4 矩阵键盘协议和接口。
- (2) 了解模块驱动方法，掌握交叉编译的概念及方法。

2. 实验设备

- (1) 硬件：UP-CUP2440 型嵌入式实验平台，PC 机 Pentium 500 以上，硬盘 40G 以上，内存大于 256M。
- (2) 软件：Vmware Workstation8 虚拟机系统，Red Hat Linux 操作系统，MiniCom/Xshell 超级终端，

ARM-LINUX 交叉编译开发环境。

3. 实验内容

加载 UP-CUP2440 型开发板上 4*4 矩阵键盘驱动模块，在超级终端上接收键盘的按键，并输出相应的信息。

4. 实验步骤

- (1) 进入实验目录。

启动虚拟机运行 Red Hat Linux 操作系统，本次实验的实验目录为 /UP-CUP2440/SRC/exp/driver/keyboard，通过以下命令进入实验目录：

```
[root@vm-dev ~]# cd /UP-CUP2440/SRC/exp/driver/keyboard
```

实验目录的内容如下：

```

up-tech:/mnt/nfs/SRC/exp/driver/keyboard #ls
Makefile      get_key.c     get_key.o     kbd_types.h  keyboard.h    main.c
driver        get_key.h     getkey        keyboard.c   keyboard.o    main.o
up-tech:/mnt/nfs/SRC/exp/driver/keyboard #

```

也可以通过在虚拟机的终端输入以下命令，在 UP-CUP2440 目录下建立自己的工作目录 keyboard，命令如下：

```
[root@vm-dev UP-CUP2440]# mkdir keyboard
```

```
[root@vm-dev UP-CUP2440]# cd keyboard
```

- (2) 编写程序源代码和 Makefile 文件。

进入 keyboard 目录后，使用 vi 命令来编辑以下几个源代码文件并保存（可参阅步骤(1)中实验目录下的源文件）：

```
[root@vm-dev keyboard]# vi main.c
[root@vm-dev keyboard]# vi keyboard.c
[root@vm-dev keyboard]# vi keyboard.h
[root@vm-dev keyboard]# vi get_key.c
[root@vm-dev keyboard]# vi get_key.h
[root@vm-dev keyboard]# vi kbd_types.h
[root@vm-dev keyboard]# vi Makefile
```

注意 Makefile 中定义的内核目录和编译器宏的指定。

(3) 清除中间代码，重新编译。

```
[root@vm-dev keyboard]# make clean
rm -f getkey *.elf *.gdb *.o
[root@vm-dev keyboard]# make
arm-linux-gcc -c -o keyboard.o keyboard.c
arm-linux-gcc -c -o get_key.o get_key.c
get_key.c: In function `get_line':
get_key.c:81: warning: return makes integer from pointer without a cast
arm-linux-gcc -c -o main.o main.c
arm-linux-gcc -o getkey keyboard.o get_key.o main.o -lpthread
[root@vm-dev keyboard]# ls
```

```
driver  get_key.c  get_key.o  keyboard.c  keyboard.o  main.o
getkey  get_key.h  kbd_types.h  keyboard.h  main.c  Makefile
```

当前目录下生成应用测试程序 getkey。

(4) 启动 UP-CUP2440 实验系统，连好网线、串口线、电源线。通过串口的 Xshell 超级终端挂载虚拟机的实验目录 UP-CUP2440。

```
up-tech:~ #mount -t nfs -o nolock,rsize=4096,wsize=4096 192.168.1.12:/UP-CUP2440 /mnt/nfs/
```

(5) 进入串口终端的 NFS 共享实验目录。

```
up-tech:~ # cd /mnt/nfs/SRC/exp/driver/keyboard/
up-tech:/mnt/nfs/SRC/exp/driver/keyboard # ls
driver  get_key.c  get_key.o  keyboard.c  keyboard.o  main.o
getkey  get_key.h  kbd_types.h  keyboard.h  main.c  Makefile
```

```
up-tech:/mnt/nfs/SRC/exp/driver/keyboard #ls
Makefile      get_key.c      get_key.o      kbd_types.h    keyboard.h      main.c
getkey        get_key.h      getkey         keyboard.c     keyboard.o     main.o
```

(6) 手动加载驱动程序 mega8.ko。

```
up-tech:/mnt/nfs/SRC/exp/driver/keyboard/ # insmod driver/mega8.ko
no PS/2 device found on PS/2 Port 0!
no PS/2 device found on PS/2 Port 1!
input: s3c2410_mouse as /class/input/input0
input: s3c2410_keyboard as /class/input/input1
initialization success!
```

加载成功后会在 UP-CUP2440 型系统/dev 目录下自动建立设备节点 Mega8-kbd，可以通过以下命令查看设备属性：

```
up-tech:/mnt/nfs/SRC/exp/driver/keyboard/ # ll /dev/Mega8-kbd
crw-r--r--  1  root    root    250,  0 May 11 15:34 /dev/Mega8-kbd

up-tech:/mnt/nfs/SRC/exp/driver/keyboard #ll /dev/Mega8-kbd
crw-rw----  1  root    root    250,  0 May 11 15:34 /dev/Mega8-kbd
```

(7) 执行应用程序，测试该驱动及设备。

```
up-tech:/mnt/nfs/SRC/exp/driver/keyboard/ # ./getkey
```

keyboard is opened

keyboard opened!

按下开发板上的矩阵键盘按键，会显示出相应的键值，按下 0 键退出程序，如下图所示：

```
up-tech:/mnt/nfs/SRC/exp/driver/keyboard # ./getkey
keyboard is opened
keyboard opened!
which key you press is 5
which key you press is 8
which key you press is 7
which key you press is 1
which key you press is 3
which key you press is 6
which key you press is 2
which key you press is 4
which key you press is 9
which key you press is 0
up-tech:/mnt/nfs/SRC/exp/driver/keyboard #
```

实验五 LED 数码管实验

预习要求：

- (1) 阅读 LED、LED 数码管电路相关资料。
- (2) 了解 UP-CUP2440 教学实验开发平台 LED 数码管的硬件结构。

一、 基础性实验

1. 实验目的

- (1) 学习 LED 的相关知识。
- (2) 掌握 74HC273 芯片的工作原理。
- (3) 了解 SPI 接口的相关知识。

2. 实验设备

- (1) 硬件：UP-CUP2440 型嵌入式实验平台，PC 机 Pentium 500 以上，硬盘 40G 以上，内存大于 256M。
- (2) 软件：Vmware Workstation8 虚拟机系统，Red Hat Linux 操作系统，MiniCom/Xshell 超级终端，ARM-LINUX 交叉编译开发环境。

3. 实验内容

编写程序实现控制开发板上的 LED 数码管从小到大循环显示数字。

学习 LED 相关知识，了解 74HC273 芯片对 LED 点亮的工作机制，熟练阅读 74HC273 芯片资料，掌握对它的使用。

4. 实验步骤

- (1) 启动 Linux 虚拟机，通过虚拟机的终端输入以下命令，在 UP-CUP2440 目录下建立工作目录 08_led：

```
[root@vm-dev UP-CUP2440]# mkdir 08_led  
[root@vm-dev UP-CUP2440]# cd 08_led
```

本次实验的实验目录为 UP-CUP2440/SRC/exp/basic/08_led，可以在此实验目录下找到本次实验的驱动程序。

```
up-tech:/mnt/nfs/SRC/exp/basic/08_led #ls  
Makefile  driver  test_led  test_led.c  test_led.o  
up-tech:/mnt/nfs/SRC/exp/basic/08_led #
```

- (2) 编写程序源代码。

使用 vi 编辑器来编辑源代码，进入 08_led 目录后，通过以下命令创建一个新文件 test_led.c：

```
[root@vm-dev 08_led]# vi test_led.c
```

按“i”或者“a”进入 vi 编辑器的编辑模式，请参阅本次实验的实验目录：UP-CUP2440/SRC/exp/basic/08_led 下的同名源文件输入源代码，完成后按 Esc 键进入命令行模式，再用命令“:wq”保存并退出。这样便在当前目录下建立了一个名为 test_led.c 的文件。

- (3) 编写 Makefile 文件。

与上一步编写 test_led.c 文件的过程类似，在 08_led 目录下用 vi 编辑器创建一个 Makefile 文件并将其源代码录入其中，请参阅本次实验的实验目录：UP-CUP2440/SRC/exp/basic/08_led 下的同名源文件。

```
[root@vm-dev 08_led]# vi Makefile
```

- (4) 清除中间代码，编译应用程序。

在上面的步骤完成后，就可以在 08_led 目录下运行“make”命令来编译新建立的程序。如果进行了修改，需要重新编译再运行：

```
[root@vm-dev 08_led]# make clean
rm -f *.o *~ core .depend
[root@vm-dev 08_led]# make
arm-linux-gcc -c -o test_led.o test_led.c
arm-linux-gcc test_led.o -o test_led
[root@vm-dev 08_led]# ls
Makefile driver test_led test_led.c test_led.o
编译成功后，当前目录下生成了一个可执行程序 test_led。
```

```
[root@vm-dev 08_led]# make clean
rm -f *.o *~ core .depend
[root@vm-dev 08_led]# make
arm-linux-gcc -c -o test_led.o test_led.c
arm-linux-gcc test_led.o -o test_led
[root@vm-dev 08_led]# ls
driver Makefile test_led test_led1 test_led.c test_led.o
[root@vm-dev 08_led]#
```

(5) NFS 挂载实验目录测试。

启动 UP-CUP2440 实验系统，连好网线、串口线、电源线。通过串口的 Xshell 超级终端挂载虚拟机的实验目录 UP-CUP2440。

```
up-tech:~ #mount -t nfs -o nolock,rsize=4096,wsize=4096 192.168.1.12:/UP-CUP2440 /mnt/nfs/
```

(6) 进入串口终端的 NFS 共享实验目录。

```
up-tech:~ # cd /mnt/nfs/SRC/exp/basic/08_led/
```

```
up-tech:/mnt/nfs/SRC/exp/basic/08_led # ls
```

```
Makefile driver test_led test_led.c test_led.o
```

(7) 使用 insmod 命令加载驱动程序。

```
up-tech:/mnt/nfs/SRC/exp/basic/08_led #insmod driver/s3c2440-led.ko
```

```
s3c2440_led device initialized
```

```
up-tech:/mnt/nfs/SRC/exp/basic/08_led #insmod driver/s3c2440-led.ko
s3c2440_led device initialized
up-tech:/mnt/nfs/SRC/exp/basic/08_led #
```

(8) 执行程序。

```
up-tech:/mnt/nfs/SRC/exp/basic/08_led #./test_led
```

```
led device open sucess!
```

```
will enter TUBE LED,please waiting .....
```

```
DOT buffer is ff
```

```
DOT buffer is ff00
```

```
DOT buffer is c0
```

```
DOT buffer is c090
```

```
DOT buffer is f9
```

开发板上的 8 字数码管从小到大依次显示不同的数字，按下 Ctrl+C 结束程序，如下图所示：

```

up-tech:/mnt/nfs/SRC/exp/basic/08_led #./test_led
led device open sucess!
will enter TUBE LED ,please waiting .....
DOT buffer is ff
DOT buffer is ff00
DOT buffer is c0
DOT buffer is c090
DOT buffer is f9
DOT buffer is f980
DOT buffer is a4
DOT buffer is a4f8
DOT buffer is b0
DOT buffer is b082

```

5. 实验思考题

- (1) LED 是如何正常工作的？
- (2) 74HC273 芯片的特点及使用方法是什么？

二、 设计性实验

1. 实验目的

- (1) 熟练掌握 ADS1.2 下的 ARMulator 软件仿真。
- (2) 掌握使用 LDR/STR 指令完成存储器的访问。

2. 实验设备

- (1) 硬件：PC 机一台。
- (2) 软件：Window XP 系统，ADS 1.2 集成开发环境。

3. 实验内容

SRC DCD 1,2,3,4,5,6,7,8,1,2,3,4,5,6,7,8,1,2,3,4

DST DCD 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

内存中定义两组数据如上，要求将数据从源数据区复制到目的数据区，复制时以 8 个字节为单位进行，对于最后所剩不足 8 个字节的数据，以字为单位进行复制。

4. 实验步骤

- (1) 启动 ADS 1.2 集成开发环境，使用 ARM Executable Image 工程模板建立一个工程。
- (2) 建立汇编源文件，编写实验程序，然后添加到工程中。
- (3) 设置工程连接地址 RO Base 为 0x40000000, RW Base 为 0x40003000。设置调试入口地址 Image entry point 为 0x40000000。
- (4) 编译连接工程，选择[Project]->[Debug]，启动 AXD 进行软件仿真调试。

注意：在 AXD 调试环境，打开[Options]->[Configure Target...], 弹出 Choose Target 窗口，在 “Target Environments” 框中选择 “...ARMulate.dll” 项，进行 ARMulator 软件仿真。

(5) 选择[Processor views]->[Registers]，打开寄存器观察窗口；再选择[Processor views]->[Memory]，打开内存观察窗口，设置观察地址为 0x40003000，显示方式为 32Bit，监视程序运行过程中寄存器值、内存中 0x40003000 地址上值的变化。

(6) 可以单步运行程序，可以设置/取消断点，或者全速运行程序，停止程序运行，调试时观察寄存器和 0x40003000 地址上的值。

5. 实验参考程序

```

AREA EXAM5, CODE, READONLY
NUM EQU 20
ENTRY

START

```

LDR R0,=SRC

LDR R1,=DST

MOV R2,#NUM

MOV SP,#0x400

blockcopy

MOVSB R3,R2,LSR #3 ;需要对进行的以 8 字为单位的数据复制

BEQ copywords ;对剩余不足 8 字节数据跳转 copywords,

;以字为单位复制

STMFD SP!,{R4-R11}

outcopy

LDMIA R0!,{R4-R11} ;从数据源取出一个字数据到 R3 并更新 R0

STMIAR1!,{R4-R11} ;将字数据存储在目标地址并更新 R1

SUBS R3,R3,#1

BNE outcopy

LDMFD SP!, {R4-R11}

copyw ords

ANDS R2,R2,#7 ;剩余不足 8 个字的数据的字数

BEQ STOP ;数据复制完则跳转到 stop

wordcopy

```
LDR R3,[R0],#4
```

STR R3,[R1],#4

SUBS R2,R2,#1

BNE wordcopy

STOP

B STOP

AREA BlockData,DATA,READWRITE

SRCDCD 1,2,3,4,5,6,7,8,1,2,3,4,5,6,7,8,1,2,3,4

[illegible]

END

6. 实验思考题

(1) LDR 伪指令与 LDR 加载指令的功能和应用有何区别？

(2) 若使用 LDRB/STRB 代替程序中的所有加载/存储指令 (LDR/STR)，程序会得到正确的执行吗？

(3) STMFD 与 LDMFD 指令分别起到什么作用?

实验六 A/D 和 D/A 接口实验

预习要求:

- (1) 了解 A/D 转换器和 D/A 转换器的类型及其重要指标。
- (2) 阅读 A/D 和 D/A 转换原理的相关资料。

一、 基础性实验

1. 实验目的

- (1) 学习 A/D 接口原理, 了解实现 A/D 系统对于系统的软件和硬件要求。
- (2) 阅读 ARM 芯片文档, 掌握 ARM 的 A/D 相关寄存器的功能, 熟悉 ARM 系统硬件的 A/D 相关接口。
- (3) 了解在 Linux 环境下对 S3C2440 芯片的 8 通道 10 位 A/D 的操作与控制。

2. 实验设备

- (1) 硬件: UP-CUP2440 型嵌入式实验平台, PC 机 Pentium 500 以上, 硬盘 40G 以上, 内存大于 256M。
- (2) 软件: Vmware Workstation8 虚拟机系统, Red Hat Linux 操作系统, MiniCom/Xshell 超级终端,

ARM-LINUX 交叉编译开发环境。

3. 实验内容

利用外部模拟信号编程实现 ARM 循环, 采集全部前 3 路通道, 并且在 Xshell 超级终端上显示。

4. 实验步骤

- (1) 启动 Linux 虚拟机, 通过虚拟机的终端输入以下命令, 在 UP-CUP2440 目录下建立工作目录 03_ad:

```
[root@vm-dev UP-CUP2440]# mkdir 03_ad
[root@vm-dev UP-CUP2440]# cd 03_ad
```

本次实验的实验目录为 UP-CUP2440/SRC/exp/basic/03_ad, 可以在此实验目录下找到本次实验的驱动程序及源代码文件。

```
up-tech:/mnt/nfs/SRC/exp/basic/03_ad #ls
Makefile      driver        main.c        readme.txt
ad             hardware.h    main.o        s3c2440-adc.h
up-tech:/mnt/nfs/SRC/exp/basic/03_ad #
```

- (2) 编写程序源代码。

建立完工作目录之后, 在虚拟机的终端下用 vi 编辑器来编写 main.c、hardware.h、s3c2440-adc.h 的源代码, 具体代码可参照本次实验目录 UP-CUP2440/SRC/exp/basic/03_ad 下的源代码文件:

```
[root@vm-dev 03_ad]# vi main.c
[root@vm-dev 03_ad]# vi hardware.h
[root@vm-dev 03_ad]# vi s3c2440-adc.h
```

按“i”或者“a”进入 vi 编辑器的编辑模式, 请参阅本次实验的实验目录: UP-CUP2440/SRC/exp/basic/03_ad 下的同名源文件输入源代码, 完成后按 Esc 键进入命令行模式, 再用命令“:wq”保存并退出。这样便在当前目录下建立了若干源文件。

- (3) 编写 Makefile 文件。

与上面编写源文件的过程类似, 在 03_ad 目录下用 vi 编辑器来创建一个 Makefile 文件并将源代码录入其中, 请参阅本次实验的实验目录: UP-CUP2440/SRC/exp/basic/03_ad 下的同名源文件。

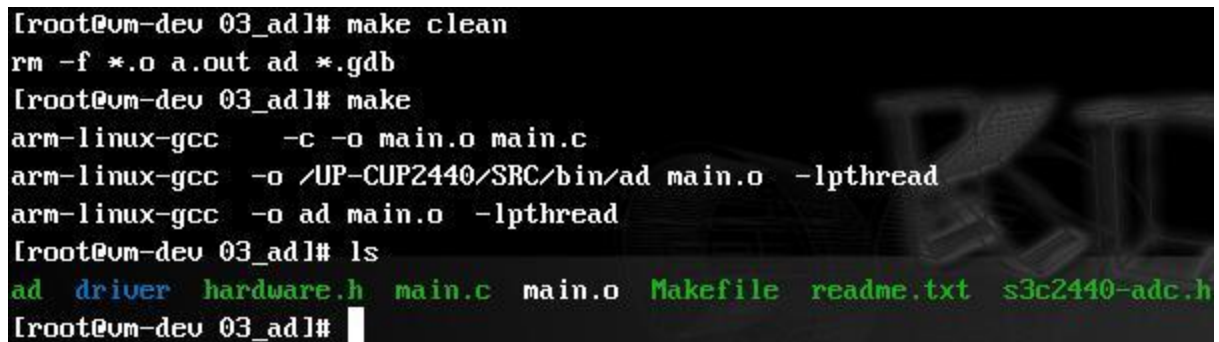
```
[root@vm-dev 03_ad]# vi Makefile
```

- (4) 清除中间代码, 编译应用程序。

在上面的步骤完成后，就可以在 03_ad 目录下运行“make”命令来编译新建立的程序。如果进行了修改，需要重新编译再运行：

```
[root@vm-dev 03_ad]# make clean
rm -f *.o a.out ad *.gdb
[root@vm-dev 03_ad]# make
arm-linux-gcc      -c -o main.o main.c
arm-linux-gcc      -o /UP-CUP2440/SRC/bin/ad main.o -lpthread
arm-linux-gcc      -o ad main.o -lpthread
[root@vm-dev 03_ad]# ls
Makefile  ad  driver  hardware.h  main.c  main.o  readme.txt  s3c2440-adc.h
```

编译成功后，当前目录下生成了一个可执行程序 ad。



```
[root@vm-dev 03_ad]# make clean
rm -f *.o a.out ad *.gdb
[root@vm-dev 03_ad]# make
arm-linux-gcc      -c -o main.o main.c
arm-linux-gcc      -o /UP-CUP2440/SRC/bin/ad main.o -lpthread
arm-linux-gcc      -o ad main.o -lpthread
[root@vm-dev 03_ad]# ls
ad  driver  hardware.h  main.c  main.o  Makefile  readme.txt  s3c2440-adc.h
[root@vm-dev 03_ad]#
```

(5) NFS 挂载实验目录测试。

启动 UP-CUP2440 实验系统，连好网线、串口线、电源线。通过串口的 Xshell 超级终端挂载虚拟机的实验目录 UP-CUP2440。

```
up-tech:~ #mount -t nfs -o nolock,rsz=4096,wsz=4096 192.168.1.12:/UP-CUP2440 /mnt/nfs/
```

(6) 进入串口终端的 NFS 共享实验目录。

```
up-tech:~ # cd /mnt/nfs/SRC/exp/basic/03_ad
up-tech:/mnt/nfs/SRC/exp/basic/03_ad # ls
```

```
Makefile  ad  driver  hardware.h  main.c  main.o  readme.txt  s3c2440-adc.h
```

(7) 使用 insmod 命令加载驱动程序。

```
up-tech:/mnt/nfs/SRC/exp/basic/03_ad #insmod driver/s3c2440-adc.ko
add s3c2440_adc ok!
```

```
up-tech:/mnt/nfs/SRC/exp/basic/03_ad #insmod driver/s3c2440-adc.ko
add s3c2440_adc ok!
up-tech:/mnt/nfs/SRC/exp/basic/03_ad #
```

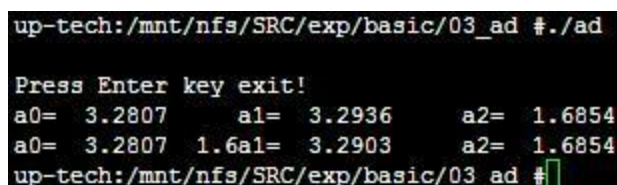
(8) 执行程序。

```
up-tech:/mnt/nfs/SRC/exp/basic/03_ad #./ad
```

Press Enter key exit!

```
a0= 0.0011    a1= 0.8089    a2= 0.6187
```

可以通过调节开发板上的三个电位器（ADC POT0-2），来查看 a0、a1、a2 的变化，如下图所示：



```
up-tech:/mnt/nfs/SRC/exp/basic/03_ad #./ad

Press Enter key exit!
a0= 3.2807    a1= 3.2936    a2= 1.6854
a0= 3.2807    1.6a1= 3.2903    a2= 1.6854
up-tech:/mnt/nfs/SRC/exp/basic/03_ad #
```

5. 实验思考题

- (1) 逐次逼近型的 A/D 转换器原理是什么？
- (2) ARM 的 A/D 功能的相关寄存器有哪几个，对应的地址是什么？
- (3) 如何启动 ARM 开始转换 A/D，有几种方式？转换开始时，ARM 是如何知道转换哪路通道的？如何判断转换结束？

二、提高性实验

1. 实验目的

- (1) 学习 D/A 转换原理。
- (2) 掌握 MAX504 D/A 转换芯片的使用方法。
- (3) 掌握不带有 D/A 的 CPU 扩展 D/A 功能的主要方法。
- (4) 了解 D/A 驱动程序加入内核的方法。

2. 实验设备

- (1) 硬件：UP-CUP2440 型嵌入式实验平台，PC 机 Pentium 500 以上，硬盘 40G 以上，内存大于 256M。
- (2) 软件：Vmware Workstation8 虚拟机系统，Red Hat Linux 操作系统，MiniCom/Xshell 超级终端，ARM-LINUX 交叉编译开发环境。

3. 实验内容

学习 D/A 接口原理，了解实现 D/A 系统对于系统的软件和硬件要求。阅读 MAX504 芯片文档，掌握其使用方法。

4. 实验步骤

- (1) 启动 Linux 虚拟机，通过虚拟机的终端输入以下命令，在 UP-CUP2440 目录下建立工作目录 04_da：

```
[root@vm-dev UP-CUP2440]# mkdir 04_da
[root@vm-dev UP-CUP2440]# cd 04_da
```

本次实验的实验目录为 UP-CUP2440/SRC/exp/basic/04_da，可以在此实验目录下找到本次实验的驱动程序及源代码文件。

```
up-tech:/mnt/nfs/SRC/exp/basic/04_da #ls
Makefile  da_main  da_main.c  da_main.o  driver
up-tech:/mnt/nfs/SRC/exp/basic/04_da #
```

- (2) 编写程序源代码。

建立完工作目录之后，在虚拟机的终端下用 vi 编辑器来编写 main.c 源代码，具体代码请参阅本次实验的实验目录：UP-CUP2440/SRC/exp/basic/04_da 下的同名源文件。

```
[root@vm-dev 04_da]# vi da_main.c
```

按“i”或者“a”进入 vi 编辑器的编辑模式，请参阅本次实验的实验目录：UP-CUP2440/SRC/exp/basic/04_da 下的同名源文件输入源代码，完成后按 Esc 键进入命令行模式，再用命令“:wq”保存并退出。这样便在当前目录下建立了 da_main.c 源文件。

- (3) 编写 Makefile 文件。

与上面编写 dc_main.c 的过程类似，在 04_da 目录下用 vi 编辑器来创建一个 Makefile 文件并将源代码录入其中，请参阅本次实验的实验目录：UP-CUP2440/SRC/exp/basic/04_da 下的同名源文件。。

```
[root@vm-dev 04_da]# vi Makefile
```

- (4) 清除中间代码，编译应用程序。

在上面的步骤完成后，就可以在 04_da 目录下运行“make”命令来编译新建立的程序。如果进行了修改，需要重新编译再运行：

```
[root@vm-dev 04_da]# make clean
rm -f *.o a.out da *.gdb da_main
[root@vm-dev 04_da]# make
arm-linux-gcc -c -o da_main.o da_main.c
```

```
arm-linux-gcc da_main.o -o da_main
```

```
[root@vm-dev 04_da]# ls
```

```
Makefile da_main da_main.c da_main.o driver
```

编译成功后，当前目录下生成了一个可执行程序 da_main。

```
[root@vm-dev 04_da]# make clean
rm -f *.o a.out da *.gdb da_main
[root@vm-dev 04_da]# make
arm-linux-gcc -c -o da_main.o da_main.c
arm-linux-gcc da_main.o -o da_main
[root@vm-dev 04_da]# ls
da_main da_main.c da_main.o driver Makefile
[root@vm-dev 04_da]#
```

(5) NFS 挂载实验目录测试。

启动 UP-CUP2440 实验系统，连好网线、串口线、电源线。通过串口的 Xshell 超级终端挂载虚拟机的实验目录 UP-CUP2440。

```
up-tech:~ #mount -t nfs -o nolock,rsz=4096,wsz=4096 192.168.1.12:/UP-CUP2440 /mnt/nfs/
```

(6) 进入串口终端的 NFS 共享实验目录。

```
up-tech:/ #cd /mnt/nfs/SRC/exp/basic/04_da/
```

```
up-tech:/mnt/nfs/SRC/exp/basic/04_da #ls
```

```
Makefile da_main da_main.c da_main.o driver
```

(7) 使用 insmod 命令加载驱动程序。

```
up-tech:/mnt/nfs/SRC/exp/basic/04_da #insmod driver/s3c2440-da.ko
```

```
s3c2440-da device initialized
```

```
up-tech:/mnt/nfs/SRC/exp/basic/04_da #insmod driver/s3c2440-da.ko
s3c2440-da device initialized
up-tech:/mnt/nfs/SRC/exp/basic/04_da #
```

(8) 执行程序。

```
up-tech:/mnt/nfs/SRC/exp/basic/04_da #./da_main 1
```

```
Current Voltage is 1.000000 v
```

```
up-tech:/mnt/nfs/SRC/exp/basic/04_da #./da_main 2.66
```

```
Current Voltage is 2.660000 v
```

此时可用示波器或万用表测开发板上 DA 的输出电压，如下图所示：

```
up-tech:/mnt/nfs/SRC/exp/basic/04_da #./da_main 1
cmd 18
cmd 16
Current Voltage is 1.000000 v
up-tech:/mnt/nfs/SRC/exp/basic/04_da #./da_main 2.66
cmd 18
cmd 16
Current Voltage is 2.660000 v
up-tech:/mnt/nfs/SRC/exp/basic/04_da #
```

5. 实验思考题

(1) D/A 转换器的分类有哪几种？

(2) MAX504 的特点及使用方法是什么？

实验七 HTML 网页设计和嵌入式 Web 服务器设计

预习要求：

- (1) 阅读 HTML 标记语言的相关资料。
- (2) 了解嵌入式 WEB 服务器架构。
- (3) 了解 SOCKET 编程的相关知识。
- (4) 阅读 HTTP 协议的相关内容，学习几个重要的网络函数的使用方法。

一、 基础性实验

1. 实验目的

了解 HTML 标记语言的语法规则，掌握在 Windows 环境下编写一个 HTML 网页的方法。

2. 实验设备

- (1) 硬件：PC 机一台。
- (2) 软件：Windows XP 系统。

3. 实验内容

编写一个 HTML 静态网页文件，包含文本、图像、表格、超级链接和框架等基本元素，要求进行合理的布局，并设置有较为美观的背景和颜色。

第一个 HTML 例子：

HTML 的语法就是给文本加上表明文本含义的标签，让用户（人或程序）能对文本得到更好的理解。下面是一个简单的 HTML 文档，可以在记事本或其他文本编辑软件中输入以下代码，保存为扩展名 html/htm 的文件，再用浏览器打开这个文件，即可以查看自己写的网页文件。

例 1：

```
<html>
  <head>
    <title>第一个 Html 文档</title>
  </head>
  <body>
    欢迎访问<a href="http://arm.net">嵌入式系统的网页</a>!
  </body>
</html>
```

所有的 HTML 文档都应该有一个<html>标签，<html>标签可以包含两个部分：<head>和<body>。

<head>标签用于包含整个文档的一般信息，比如文档的标题（<title>标签用于包含标题）、对整个文档的描述、文档的关键字等等。用浏览器打开本例子文件时，注意浏览器的标题栏被设置为“第一个 Html 文档”，就是用<title>标签进行设置的。文档的具体内容放在<body>标签里。

<a>标签用于表示链接，在浏览器中查看 HTML 文档时，点击<a>标签括起来的内容时，通常会跳转到另一个页面。这个要跳转到的页面的地址由<a>标签的 href 属性指定。上面的中，href 属性的值就是 http://arm.net。

常用标签介绍：

● 文本

文本标签用于改变字体、字号、文字颜色等文字属性。

在记事本中输入以下语句，保存为扩展名 html/htm 的文件，并用浏览器打开这个文件查看效果：

例 2:

6

4

红色的 5

黑体的字

加粗、下划线、斜体字也是常用的文字效果，分别用标签、<u>、<i>表示：

Bold

<i>italic</i>

<u>underline</u>

一段较长的文本，如果有合适的小标题的话，就可以快速地对它的内容进行大致的了解。在 HTML 里，用来表示标题的标签有：<h1>，<h2>，<h3>，<h4>，<h5>，<h6>，它们分别表示一级标题，二级标题，三级标题等等，见下面的例子：

例 3:

<h1>HTML 30 分钟教程</h1>

<h2>什么是 HTML</h2>

...

<h2>HTML 是什么样的</h2>

...

● 图片

<hr>标签用于在页面上添加横线。可以通过指定 width 和 color 属性来控制横线的长度和颜色。

例如：<hr width="90%" color="red" />，在网页上显示一条长度为网页宽度 90%，颜色为红色的横线。

标签用于在页面上添加图片，src 属性指定图片的地址，如果无法打开 src 指定的图片，浏览器通常会在页面上需要显示图片的地方显示 alt 属性定义的文本。

例 4:

● 链接

超级链接用<a>标签表示，href 属性指定了链接到的地址。<a>标签可以包含文本，也可以包含图片。

例 5:

嵌入式系统的网站

● 分段与换行

由于 HTML 文档会忽略空白符，所以要想保证正常的分段换行的话，必须指出哪些文字是属于同一段落的，这就用到了标签<p>。

例 6:

<p>这是第一段。</p>

<p>这是第二段。</p>

除了用<p>标签划分段落，
标签也常常被使用。
只表示换行，不表示段落的开始或结束，所以通常没有结束标签。

例 7:

这是第一段。

这是第二段。

这是第三段。

有时要把文档看作不同的部分组合起来的，比如一个典型的页面可能包括三个部分：页头，主体，页脚。<div>标签专门用于标明不同的部分：

例 8:

```
<div>页头内容</div>
<div>主体内容</div>
<div>页脚内容</div>
```

- 表格

HTML 文档在浏览器里通常是从左到右，从上到下地显示的，到了窗口右边就自动换行。为了实现分栏的效果进行页面的布局，很多人使用表格（<table>）进行页面排版。

<table>标签里通常会包含几个<tr>标签，<tr>代表表格里的一行。<tr>标签又会包含<td>标签，每个<td>代表一个单元格。

例 9:

```
<table>
  <tr>
    <td>2000</td><td>悉尼</td>
  </tr>
  <tr>
    <td>2004</td><td>雅典</td>
  </tr>
  <tr>
    <td>2008</td><td>北京</td>
  </tr>
</table>
```

<tr>标签还可以被<table>里的<thead>或<tbody>或<tfoot>包含。它们分别代表表头、表正文、表脚。在打印网页的时候，如果表格很大，一页打印不完，<thead>和<tfoot>将在每一页出现。

<th>和<td>非常相似，也用在<tr>里边，不同的是<th>代表这个单元格是它所在的行或列的标题。

例 10:

```
<table>
  <thead>
    <tr><th>时间</th><th>地点</th></tr>
  </thead>
  <tbody>
    <tr><td>2000</td><td>悉尼</td></tr>
    <tr><td>2004</td><td>雅典</td></tr>
    <tr><td>2008</td><td>北京</td></tr>
  </tbody>
</table>
```

- 列表

表格用于表示二维数据（行，列），一维数据则用列表来表示。列表可以分为无序列表（）、有序列表（）和定义列表（<dl>）。前两种列表更常见一些，都用标签包含列表项目。

无序列表表示一系列类似的项目，它们之间没有先后顺序。

例 11:

```
<ul>
  <li>苹果</li>
  <li>桔子</li>
  <li>桃</li>
</ul>
```

有序列表中各个项目间的顺序是很重要的，浏览器通常会自动给它们产生编号。

例 12:

```
<ol>
  <li>打开冰箱门</li>
  <li>把大象赶进去</li>
  <li>关上冰箱门</li>
</ol>
```

● 框架

框架使一个窗口里能同时显示多个文档。主框架页里面没有<body>标签，取代它的是<frameset>。<frameset>标签的属性 Rows 和 Cols 用于指定框架集(frameset)里有多少行(列)，以及每行(列)的高度(宽度)。

<frameset>标签可以包含<frame>标签，每个<frame>标签代表一个文档(src 属性指定文档的地址)。如果觉得这样的页面还不够复杂的话，还可以在<frameset>标签里包含<frameset>标签。

例 13:

```
<frameset rows="15%,*">
  <frame src="top.html" name=title scrolling=no>
  <frameset cols="20%,*">
    <frame src="left.html" name=sidebar>
    <frame src="right.html" name=recipes>
  </frameset>
</frameset>
```

● 背景和颜色

颜色由红色、绿色、蓝色混合而成。颜色值一个十六进制符号来定义，这个符号由红色、绿色和蓝色的值组成(RGB)。每种颜色的最小值是 0 (十六进制: #00)。最大值是 255 (十六进制: #FF)。大多数的浏览器都支持颜色名集合。仅仅有 16 种颜色名被 HTML4.0 标准所支持。它们是: aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, purple, red, silver, teal, white, yellow。如果需要使用其它的颜色，需要使用十六进制的颜色值。

<body>拥有两个配置背景的标签，分别是背景颜色标签<bgcolor>和背景图像标签<background>。

背景颜色属性<bgcolor>将背景设置为某种颜色。属性值可以是十六进制数、RGB 值或颜色名。

例 14:

```
<body bgcolor="#000000">
<body bgcolor="rgb(0,0,0)">
<body bgcolor="black">
```

以上的代码均将背景颜色设置为黑色。

背景属性<background>将背景设置为图像，属性值为图像的 URL。如果图像尺寸小于浏览器窗口，那么图像将在整个浏览器窗口进行复制。

例 15:

```
<body background="clouds.gif">
<body background="http://www.w3school.com.cn/clouds.gif">
```

URL 可以是相对地址，如第一行代码。也可以使绝对地址，如第二行代码。

搭配良好的背景和颜色，使页面中的文字更易于阅读；搭配得不好的背景和颜色，会使得页面中的文字难于阅读。

4. 实验步骤

(1) 点击 Windows 的[开始]->[程序]->[附件]->[记事本]，打开记事本程序，新建一个扩展名为 html 或 htm 的文件。

(2) 使用记事本来编写 HTML。可以使用专业的 HTML 编辑器来编辑 HTML，比如 Adobe Dreamweaver、Microsoft Expression Web、CoffeeCup HTML Editor 等。不过，使用一款简单的文本编辑器是学习 HTML 的更好方法，本实验推荐使用记事本来编辑 HTML 文件。

(3) 用 HTML 标记语言编写一个静态网页文件，要包含文本、图像、表格、超级链接和框架等基本元素，用表格等标签进行合理的布局，并设置有较为美观、便于阅读的背景和颜色。

(4) 在 IE 浏览器中运行这个写好的 HTML 文件，观察其效果。

5. 实验思考题

(1) 如果要在表格中显示背景色，如何编写 HTML？

(2) 如何在一幅图片上设置超级链接？

二、提高性实验

1. 实验目的

(1) 掌握在 ARM 设备上实现一个简单 WEB 服务器的过程。

(2) 学习在 ARM 设备上的 SOCKET 网络编程。

(3) 学习 Linux 下的 signal() 函数的使用。

2. 实验设备

(1) 硬件：UP-CUP2440 型嵌入式实验平台，PC 机 Pentium 500 以上，硬盘 40G 以上，内存大于 256M。

(2) 软件：Vmware Workstation8 虚拟机系统，Red Hat Linux 操作系统，MiniCom/Xshell 超级终端，

ARM-LINUX 交叉编译开发环境。

3. 实验内容

学习使用 socket 进行通讯编程的过程，了解一个实际的网络通讯应用程序整体设计。编写一个 WEB 服务器程序，在 PC 计算机上使用浏览器测试嵌入式 WEB 服务器的功能。

4. 实验步骤

(1) 启动 Linux 虚拟机，通过虚拟机的终端输入以下命令，在 UP-CUP2440 目录下建立工作目录 05_httpd：

```
[root@vm-dev UP-CUP2440]# mkdir 05_httpd
```

```
[root@vm-dev UP-CUP2440]# cd 05_httpd
```

本次实验的实验目录为 UP-CUP2440/SRC/exp/basic/05_httpd，可以在此实验目录下找到本次实验的驱动程序及源代码文件。

```
up-tech:/mnt/nfs/SRC/exp/basic/05_httpd #ls
copy.c      httpd.c      index.html
Makefile    copy.o      httpd        httpd.o
```

(2) 编写程序源代码。

建立完工作目录之后，在虚拟机的终端下用 vi 编辑器来编写 httpd.c、copy.c 源代码，具体代码请参阅本次实验的实验目录：UP-CUP2440/SRC/exp/basic/05_httpd 下的同名源文件。

```
[root@vm-dev 05_httpd]# vi httpd.c
```

```
[root@vm-dev 05_httpd]# vi copy.c
```

按“i”或者“a”进入 vi 编辑器的编辑模式，请参阅本次实验的实验目录：UP-CUP2440/SRC/exp/basic/05_httpd 下的同名源文件输入源代码，完成后按 Esc 键进入命令行模式，再用命令“:wq”保存并退出。这样便在当前目录下建立了 httpd.c 和 copy.c 源文件。

(3) 编写 Makefile 文件。

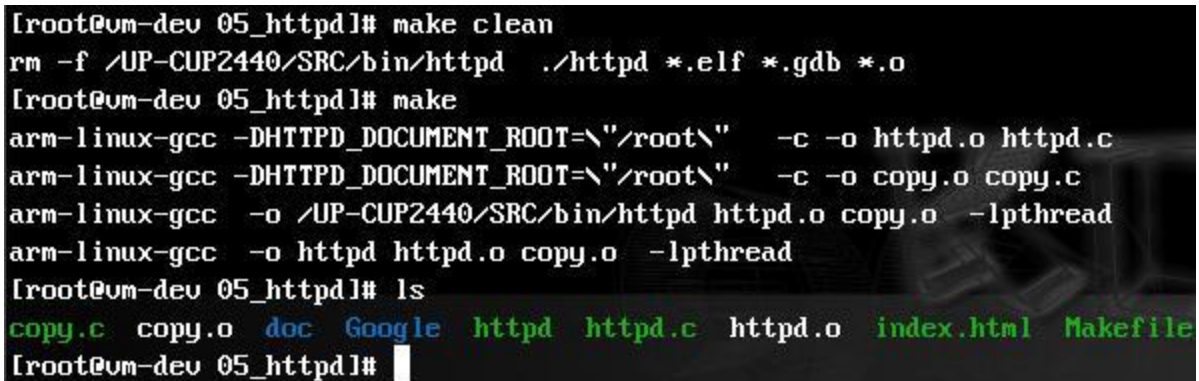
与上面编写 httpd.c 的过程类似，在 05_httpd 目录下用 vi 编辑器来创建一个 Makefile 文件并将源代码录入其中，请参阅本次实验的实验目录：UP-CUP2440/SRC/exp/basic/05_httpd 下的同名源文件。

```
[root@vm-dev 05_httpd]# vi Makefile
```

(4) 清除中间代码，编译应用程序。

在上面的步骤完成后，就可以在 05_httpd 目录下运行“make”来编译新建立的程序了。如果进行了修改，需要重新编译再运行：

```
[root@vm-dev 05_httpd]# make clean
rm -f ../bin/httpd ../httpd *.elf *.gdb *.o
[root@vm-dev 05_httpd]# make
arm-linux-gcc -DHTTPD_DOCUMENT_ROOT=\"/root\" -c -o httpd.o httpd.c
arm-linux-gcc -DHTTPD_DOCUMENT_ROOT=\"/root\" -c -o copy.o copy.c
arm-linux-gcc -o ../bin/httpd httpd.o copy.o -lpthread
arm-linux-gcc -o httpd httpd.o copy.o -lpthread
[root@vm-dev 05_httpd]# ls
Google Makefile copy.c copy.o doc httpd httpd.c httpd.o index.html
编译成功后，当前目录下生成了一个可执行程序 httpd。
```



```
[root@vm-dev 05_httpd]# make clean
rm -f /UP-CUP2440/SRC/bin/httpd ../httpd *.elf *.gdb *.o
[root@vm-dev 05_httpd]# make
arm-linux-gcc -DHTTPD_DOCUMENT_ROOT=\"/root\" -c -o httpd.o httpd.c
arm-linux-gcc -DHTTPD_DOCUMENT_ROOT=\"/root\" -c -o copy.o copy.c
arm-linux-gcc -o /UP-CUP2440/SRC/bin/httpd httpd.o copy.o -lpthread
arm-linux-gcc -o httpd httpd.o copy.o -lpthread
[root@vm-dev 05_httpd]# ls
copy.c copy.o doc Google httpd httpd.c httpd.o index.html Makefile
[root@vm-dev 05_httpd]#
```

(5) NFS 挂载实验目录测试。

启动 UP-CUP2440 实验系统，连好网线、串口线、电源线。通过串口的 Xshell 超级终端挂载虚拟机的实验目录 UP-CUP2440。

```
up-tech:~ #mount -t nfs -o nolock,rsiz=4096,wsiz=4096 192.168.1.12:/UP-CUP2440 /mnt/nfs/
```

(6) 进入串口终端的 NFS 共享实验目录。

```
up-tech:/ # cd /mnt/nfs/SRC/exp/basic/05_httpd/
```

```
up-tech:/mnt/nfs/SRC/exp/basic/05_httpd #ls
```

```
Google copy.c doc httpd.c index.html Makefile copy.o httpd httpd.o
```

(7) 执行程序，启动 HTTP 服务器。

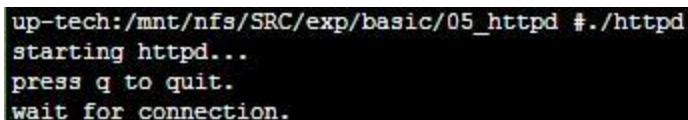
```
up-tech:/mnt/nfs/SRC/exp/basic/05_httpd #./httpd
```

```
starting httpd...
```

```
press q to quit.
```

```
wait for connection.
```

此时 UP-CUP2440 型设备端 HTTP 服务器启动并等待连接。



```
up-tech:/mnt/nfs/SRC/exp/basic/05_httpd #./httpd
starting httpd...
press q to quit.
wait for connection.
```

(8) 浏览网页。

打开 PC 机 Windows XP 系统上的 IE 浏览器，在地址栏输入 UP-CUP2440 型设备的 IP 地址：192.168.1.193 (ARM 端 IP 地址需根据具体情况而设定，可以使用 ifconfig 命令查看和设置)。如图所示：



输入正确的 IP 地址即可访问 UP-CUP2440 型设备上 HTTP 提供的服务。可以尝试把“基础实验”中完成的网页替换成实验目录中的默认网页，浏览自己的网页。

实验八 图形界面应用程序设计

预习要求:

- (1) 了解 QT 编程的相关资料。
- (2) 学会使用 Qt Designer 编写程序。

一、 基础性实验

1. 实验目的

- (1) 了解在 Linux 下安装 QT 的基本步骤。
- (2) 学会 QT 环境在 X11 平台下程序设计的方法。

2. 实验设备

- (1) 硬件: UP-CUP2440 型嵌入式实验平台, PC 机 Pentium 500 以上, 硬盘 40G 以上, 内存大于 256M。
- (2) 软件: Vmware Workstation8 虚拟机系统, Red Hat Linux 操作系统, MiniCom/Xshell 超级终端, ARM-LINUX 交叉编译开发环境, QT 软件包: qt-x11-opensource-src-4.4.0.tar.gz。

3. 实验内容

在本机宿主机 Linux 下编译 X11 环境的 QT 库, 编译和运行一个程序并显示运行结果。

本次实验主要学习在嵌入式 Linux 系统下如何构建图形用户界面程序, 着重学习 QT 程序设计的基本方法, 以及 QT 设计当中常用的工具与命令。通过本次实验, 可以基本掌握嵌入式开发中图形用户界面设计的基本方法, 实现图形界面与用户的交互。

4. 实验步骤

- (1) 建立 QT-X11 实验目录。

打开 Linux 虚拟机的终端, 在/home 目录下建立 uptech 子目录。所有 QT 相关实验都放在该目录下完成, 命令如下:

```
[root@vm-dev /]# cd /home/
```

```
[root@vm-dev home]# mkdir uptech
```

接着在 uptech 目录下建立 QT4 目录, 命令如下:

```
[root@vm-dev home]# cd uptech/
```

```
[root@vm-dev uptech]# mkdir QT4/
```

再在 QT4 目录下建立 QT-X11 目录 for_x11:

```
[root@vm-dev uptech]# cd QT4/
```

```
[root@vm-dev QT4]# mkdir for_x11
```

后续所有关于 QT-X11 实验环境都建在此目录(/home/uptech/QT4/for_x11/)下进行。

- (2) 配置编译 QT-X11 环境。

①进入 for_x11 目录, 将 QT-X11 压缩包(/UP-CUP2440/SRC/gui/qt-x11-opensource-src-4.4.0.tar.gz)拷贝到新建的目录下并解压, 命令如下:

```
[root@vm-dev for_x11]# tar xzvf /UP-CUP2440/SRC/gui/qt-x11-opensource-src-4.4.0.tar.gz -C /home/uptech/QT4/for_x11/
```

这样在实验目录的 for_x11 下产生解压后的 qt-x11-opensource-src-4.4.0 目录。所有 QT-X11 环境源码都存放在此。

- ②进入 qt-x11-opensource-src-4.4.0 目录执行 configure 文件, 配置 QT-X11 环境, 命令如下:

```
[root@vm-dev for_x11]# cd qt-x11-opensource-src-4.4.0
```

```
[root@vm-dev qt-x11-opensource-src-4.4.0]# ./configure --prefix /usr/local/Trolltech/Qt-x11-4.4.0
```

其中-prefix 参数指定 QT-X11 环境的安装目录。Configure 的其他具体配置用户可以通过--help 命令查看：

```
[root@vm-dev qt-x11-opensource-src-4.4.0]# ./configure --help
```

配置结束时在输出的许可证提示符下输入“yes”同意回车即可进入 QT-X11 库的配置过程，注意 yes 大小写及全称。

③编译 QT-X11 库。在当前 qt-x11-opensource-src-4.4.0 目录下使用命令“gmake”编译 QT-X11 库。

```
[root@vm-dev qt-x11-opensource-src-4.4.0]# gmake
```

由于 QT 库环境的庞大体积，编译 QT-X11 库环境时间比较长，具体时间因机器配置而异（大约 3 个小时左右）。因此为了节省实验时间，本次实验前各个计算机已完成这一步骤。

④安装 QT-X11 库。在编译完之后，输入 gmake install 命令进行 QT-X11 库的安装，此时便会在/usr/local/目录下产生 Trolltech/Qt-x11-4.4.0 目录。

```
[root@vm-dev qt-x11-opensource-src-4.4.0]# gmake install
```

(3) 运行 QT-X11 环境自带例程测试 QT-X11 环境的搭建是否成功。

如果上面各步都能够成功的编译通过，下面就可以通过运行 Qt/x11 自带的 demo 来查看运行结果。

在 QT-X11 库的 examples 目录下存放很多例程。进入 examples/widgets/wiggly 目录，执行编译出来的可执行程序 wiggle，如下命令：

```
[root@vm-dev qt-x11-opensource-src-4.4.0]# cd examples/widgets/wiggly/
```

```
[root@vm-dev wiggly]# ls
```

Makefile dialog.cpp dialog.h main.cpp wiggly wiggly.debug wiggly.pro wigglywidget.cpp wigglywidget.h

```
[root@vm-dev wiggly]# ./wiggly
```

弹出 Hello world!对话框，表明 QT-X11 环境已经搭建完毕。



(4) 编写 Hello QT。

将上面的步骤完成后，就已经建立好了在本机上开发 Qt 应用程序的环境，下面的步骤通过编写一个“Hello QT!”的小程序来了解 Qt 程序设计。

①在/home/uptech/目录下建立一个名为 hello 的目录，在此目录下建立一个名为 hello.cpp 的 C++源文件，具体命令如下：

```
[root@vm-dev uptech]# cd /home/uptech/
```

```
[root@vm-dev uptech]# ls
```

QT4

```
[root@vm-dev uptech]# mkdir hello
```

```
[root@vm-dev uptech]# ls
```

QT4 hello

②使用 vim/vi 编辑器编译 hello.cpp 源文件，注意是 CPP 为后缀的 C++源文件，具体源代码请参照“5. 实验参考程序”。命令如下：

```
[root@vm-dev uptech]# cd hello/
```

```
[root@vm-dev hello]# vi hello.cpp
```

进入 vim 编辑界面之后，按“a”开始编写程序，编写完成再按 Esc 键退出编辑，按下“:wq”进行保存并退出。

③编译 hello.cpp 程序。

编译 hello.cpp QT 程序要用到编译 QT-X11 环境的时候安装目录下（即 prefix 参数所指定的路径：/usr/local/Trolltech/Qt-x11-4.4.0/）的一些工具，如 qmake 等，因此为准确起见，使用工具的绝对路径。当然可以通过修改 PATH 环境变量来设置工具目录，这样可以免去输入超长绝对路径字符串的麻烦。使用前确保 qmake 版本是 4.4.0 的 QT-X11 库配套工具，可以使用 qmake -v 来查看版本。

首先在 hello 目录下执行如下命令生成工程文件 hello.pro：

```
[root@vm-dev hello]# /usr/local/Trolltech/Qt-x11-4.4.0/bin/qmake -project
[root@vm-dev hello]# ls
```

hello.cpp hello.pro

再使用 qmake 命令生成 Makefile 文件：

```
[root@vm-dev hello]# /usr/local/Trolltech/Qt-x11-4.4.0/bin/qmake
[root@vm-dev hello]# ls
```

Makefile hello.cpp hello.pro

最后用 make 命令编译程序，之后在当前目录下生成了 hello 可执行程序。

```
[root@vm-dev hello]# make
g++ -c -pipe -O2 -Wall -W -D_REENTRANT -DQT_NO_DEBUG -DQT_GUI_LIB -DQT_CORE_LIB
-DQT_SHARED -I/usr/local/Trolltech/Qt-x11-4.4.0/mkspecs/linux-g++ -I.
-I/usr/local/Trolltech/Qt-x11-4.4.0/include/QtCore
-I/usr/local/Trolltech/Qt-x11-4.4.0/include/QtGui
-I/usr/local/Trolltech/Qt-x11-4.4.0/include -I. -I. -I. -o hello.o hello.cpp
g++ -Wl,-rpath,/usr/local/Trolltech/Qt-x11-4.4.0/lib -o hello hello.o
-L/usr/local/Trolltech/Qt-x11-4.4.0/lib -lQtGui
-L/usr/local/Trolltech/Qt-x11-4.4.0/lib -L/usr/X11R6/lib -lpng -lSM -lICE -lXi
-lXrender -lXrandr -lfreetype -lfontconfig -lXext -lX11 -lQtCore -lz -lm -lrt -ldl
-lpthread
[root@vm-dev hello]# ls
```

Makefile hello hello.cpp hello.o hello.pro

④运行程序。

```
[root@vm-dev hello]# ./hello
```



QT 也支持 XML，可以把程序的第 6 行替换成下面的语句：

```
QLabel *label = new QLabel("<h2><i>Hello</i> " " "<font color=red>Qt! </font></h2>");
```

重新编译运行程序，发现界面拥有了简单的 HTML 风格，如下图：



5. 实验参考程序

*****hello.cpp 源文件*****

```
#include <QApplication>
```

```
#include <QLabel>
```

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

```

{
    QApplication app(argc, argv);
    QLabel *label = new QLabel ("Hello Qt!");
    label->show ();
    return app. exec ();
}

```

6. 实验思考题

- (1) 在本机 X11 环境下如何搭建 QT 库？
- (2) 如何编译 QT-X11 版本的程序？

二、 提高性实验

1. 实验目的

- (1) 了解 Qt Designer 的使用基本步骤。
- (2) 学会在使用 Qt Designer 编写程序，编译，本机上运行。

2. 实验设备

- (1) 硬件：UP-CUP2440 型嵌入式实验平台，PC 机 Pentium 500 以上，硬盘 40G 以上，内存大于 256M。
- (2) 软件：Vmware Workstation8 虚拟机系统，Red Hat Linux 操作系统，MiniCom/Xshell 超级终端，ARM-LINUX 交叉编译开发环境。

3. 实验内容

在 Linux 下使用 Qt Designer 设计 QT 程序界面，编写程序在本机上编译并运行。

4. 实验步骤

- (1) 进入/home/uptech 实验目录进行界面设计，并建立实验目录 testqt-x11:

```
[root@vm-dev /]# cd /home/uptech/
```

```
[root@vm-dev uptech]# mkdir testqt-x11
```

```
[root@vm-dev uptech]# cd testqt-x11/
```

- (2) 使用 designer 编辑界面程序控件，这同样使用绝对路径，使用如下命令即会弹出 designer 界面:

```
[root@vm-dev testqt-x11]# /usr/local/Trolltech/Qt-x11-4.4.0/bin/designer
```

- (3) 选择一个窗口布局 Widget 点击->创建，如图 8.1:

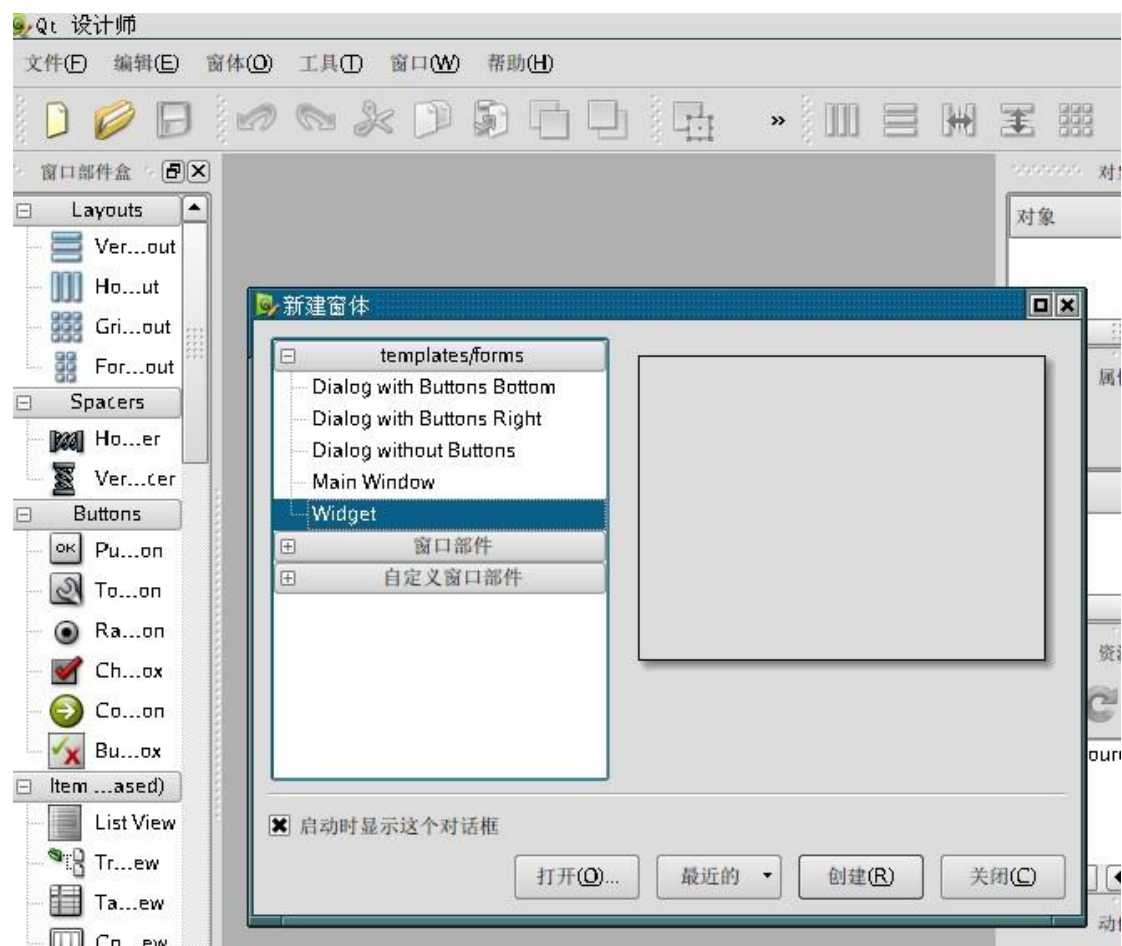


图 8.1 新建窗体

(4) 拖拽左侧几个简单的控件(TextEdit、PushButton、TextLabel)进行界面设计。如图 8.2:

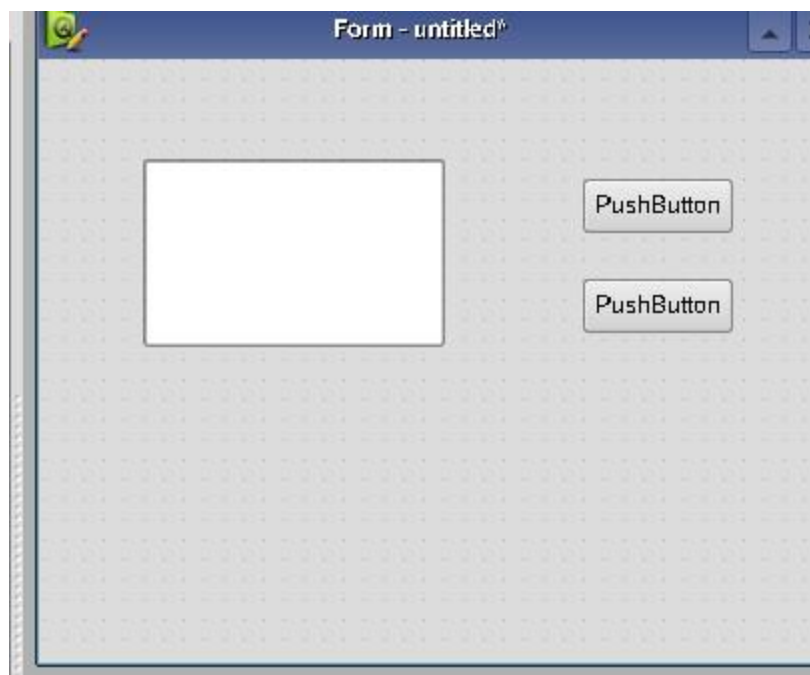


图 8.2 添加控件

(5) 右击控件来初始化控件及相关属性内容。如图 8.3:

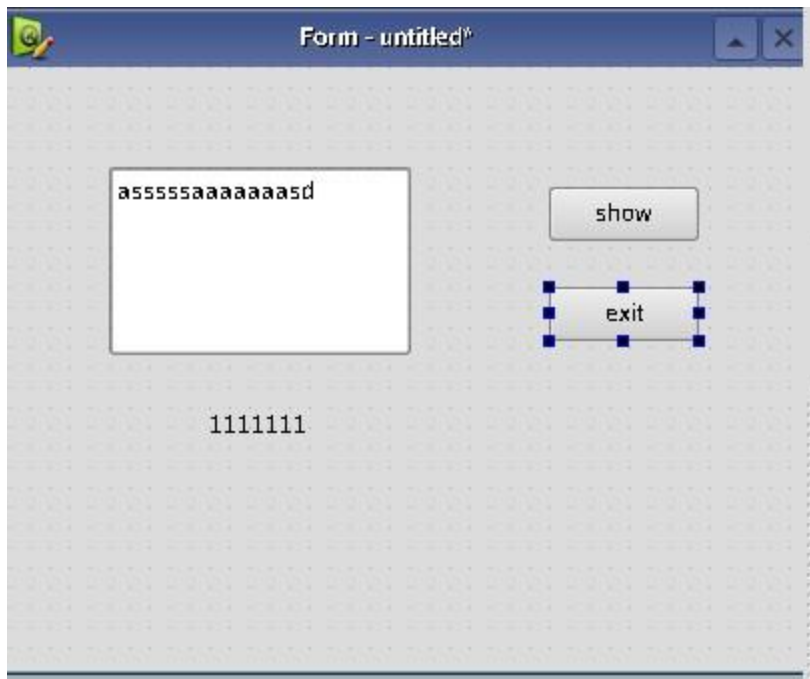


图 8.3 改变控件属性

(6) 可以通过 Qt Designer 工具栏上的快捷方式在不同设计模式间切换。在选择信号槽连接界面时候，左下角有个选项要选中，才会显示 Designer 全部信号与槽。如图 8.4: (show 按钮与文本编辑框的连接 clicked->clear, exit 按钮与 Form 的连接 clicked->close)。

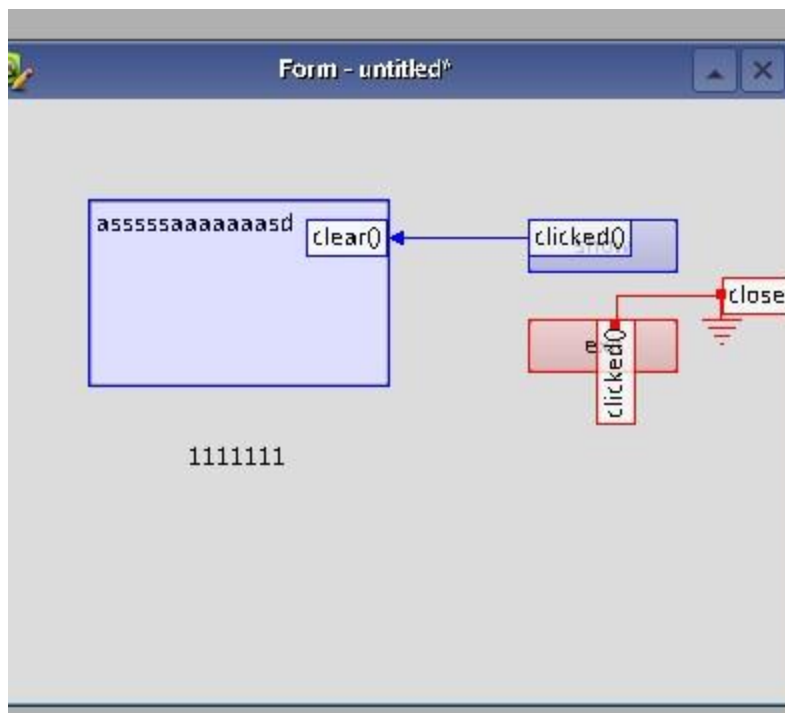


图 8.4 建立信号与槽的连接

退出界面编辑保存为 UI 格式名称为: testx11.ui。

(7) 在 testqt-x11 目录下使用 vi 命令编辑 main.cpp 函数。详细代码请参阅“5.实验参考程序”。

[root@vm-dev testqt-x11]# vi main.cpp

(8) 使用 qmake -project 命令编译程序生成工程文件.pro。


```
[root@vm-dev testqt-x11]# /usr/local/Trolltech/Qt-x11-4.4.0/bin/qmake -project
[root@vm-dev testqt-x11]# ls
```

```
main.cpp  testqt-x11.pro  testx11.ui
```

(9) 使用 qmake 命令生成 Makefile 文件:

```
[root@vm-dev testqt-x11]# /usr/local/Trolltech/Qt-x11-4.4.0/bin/qmake
[root@vm-dev testqt-x11]# ls
```

```
Makefile  main.cpp  testqt-x11.pro  testx11.ui
```

(10) make 编译生成可执行文件:

```
[root@vm-dev testqt-x11]# make
```

```
/usr/local/Trolltech/Qt-x11-4.4.0/bin/uic testx11.ui -o ui_testx11.h
```

```
g++ -c -pipe -O2 -Wall -W -D_REENTRANT -DQT_NO_DEBUG -DQT_GUI_LIB -DQT_CORE_LIB
-DQT_SHARED -I/usr/local/Trolltech/Qt-x11-4.4.0/mkspecs/linux-g++ -I.
```

```
-I/usr/local/Trolltech/Qt-x11-4.4.0/include/QtCore
```

```
-I/usr/local/Trolltech/Qt-x11-4.4.0/include/QtGui
```

```
-I/usr/local/Trolltech/Qt-x11-4.4.0/include -I. -I. -I. -o main.o main.cpp
```

```
g++ -WL,-rpath,/usr/local/Trolltech/Qt-x11-4.4.0/lib -o testqt-x11 main.o
```

```
-L/usr/local/Trolltech/Qt-x11-4.4.0/lib -lQtGui
```

```
-L/usr/local/Trolltech/Qt-x11-4.4.0/lib -L/usr/X11R6/lib -lpng -lSM -lICE -lXi
```

```
-lXrender -lXrandr -lfreetype -lfontconfig -lXext -lX11 -lQtCore -lz -lm -lrt -ldl
```

```
-lpthread
```

```
[root@vm-dev testqt-x11]# ls
```

```
Makefile  main.cpp  main.o  testqt-x11  testqt-x11.pro  testx11.ui  ui_testx11.h
```

程序编译成功后, 在当前目录生成与目录名同名的可执行 QT 程序 testqt-x11。

(11) 执行编译好的程序测试观察效果。

```
[root@vm-dev testqt-x11]# ./testqt-x11/testqt-x11
```

```
oot@vm-dev testqt-x11# vi main.cpp
```

```
oot@vm-dev testqt-x11# ls
```

```
in.cpp  testx11.
```

```
oot@vm-dev testq
```

```
oot@vm-dev testq
```

```
in.cpp  testqt-x
```

```
oot@vm-dev testq
```

```
oot@vm-dev testq
```

```
in.cpp  Makefile
```

```
oot@vm-dev testq
```

```
sr/local/Trollte
```

```
+ -c -pipe -O2 -
```

```
r/local/Trolltec
```

```
e/QtCore -I/usr/
```

```
/include/QtGui -
```

```
4.4.0/include -I
```

```
+ -Wl,-rpath,/us
```

```
ch/Qt-x11-4.4.0/
```

```
-lICE -pthread
```

```
-lz -lm -pthread -lgthread-2.0 -lglib-2.0 -lrt -ldl -lpthread
```

```
oot@vm-dev testqt-x11# ls
```

```
in.cpp  main.o  Makefile  testqt-x11  testqt-x11.pro  testx11.ui  ui_testx11.h
```

```
oot@vm-dev testqt-x11# ./testqt-x11
```



5. 实验参考程序

*****main.cpp 源文件*****

```
#include "ui_testx11.h"
```

```
int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QWidget *widget = new QWidget;
    Ui::Form ui;
    ui.setupUi(widget);
    widget->show();
    return app.exec();
}
```

6. 实验思考题

- (1) 如何使用 Qt Designer 设计 QT 程序?
- (2) 如何在控件中实现信号和槽的连接?