



## 嵌入式原理及应用 II 实验报告

课程名称: 嵌入式原理及应用II

指导老师: 刘玮、张莉君

姓 名: 廖志豪

班 级: 231212

学 号: 20201003069

2023 年 11 月 2 号

# 目录

<b>1 第一篇 Windows 环境下裸机开发实验</b>	<b>2</b>
1.1 实验二：ARM 汇编程序设计	2
1.2 实验三：ARM 汇编与 C 混合编程	3
1.3 实验四：ARM 中断实验	4
1.4 思考题	5
1.4.1 在嵌入式系统编程当中，汇编语言和 C 语言分别有什么优势？是否可以完全摒弃其中一种语言？为什么？	5
1.4.2 ARM 汇编调用 C 语言以及 C 语言调用 ARM 汇编时，如何传递参数？实验二、三程序中参数是如何传递的？	6
1.4.3 C 语言和汇编语言中是如何操作寄存器的？	6
1.4.4 比较实验二、三和四中 ADS 下的工程设置有何异同，并分析其理由。	6
1.4.5 在中断实验中为什么要把可执行程序下载到 NAND FLASH 中运行，而不是直接下载到 SDRAM 中运行？如果直接下载到 SDRAM 中运行会发生什么情况？	6
1.4.6 结合实验，叙述 NAND FLASH 启动的流程。	6
1.5 体会和建议	7
<b>2 第二、三篇 Linux 环境下裸机开发实验</b>	<b>7</b>
2.1 结合实验五，说说你所理解的 Linux 操作系统	7
2.2 列出你在实验六中执行的一些常规命令，给出该行命令的功能解释，并附上命令行执行截图	8
2.3 列出实验六、实验八的实验过程，编写并编译自己的第一个 C 程序（硬件无关），并附实验截图	9
2.4 列出实验九、十的实验过程，并附实验截图	11
2.5 思考题	14
2.5.1 思考 Windows 环境下与 Linux 环境下开发裸机程序的区别有什么？	14
2.5.2 make 及 makefile 的作用是什么？	14
2.5.3 第二篇实验与第三篇实验的 Linux 系统一样吗？如不同，不同之处有什么？	14
2.5.4 关于 Linux 系统，你还想要了解什么？	14
2.5.5 关于嵌入式 Linux 操作系统的开发，你还希望学习什么？	14
2.6 体会和建议	14

# 1 第一篇 Windows 环境下裸机开发实验

## 1.1 实验二：ARM 汇编程序设计

开发板上 4 个 LED 按二进制流水灯方式闪烁的程序：

```
1 ; 汇编指令实验
2 ; 定义端口B寄存器预定义
3 GPBCON EQU 0x56000010
4 GPBDAT EQU 0x56000014
5 GPBUP EQU 0x56000018
6 ; 该伪指令定义了一个代码段，段名为Init，属性只读
7 AREA Init, CODE, READONLY
8 ENTRY ; 程序的入口点标识
9 ResetEntry
10 ; 下面这三条语句，主要是用来设置GPB5—GPB8为输出属性
11 ldr r0, =GPBCON ; 将寄存器GPBCON的地址存放到寄存器r0中
12 ldr r1, =0x15400 ; 0x 0001 0101 0100 0000 0000
13 ; 将r1中的数据存放到地址为r0的内存单元中
14 str r1, [r0]
15
16 ; 下面这三条语句，设置GPB5—GPB8禁止上拉电阻
17 ldr r0, =GPBUP
18 ldr r1, =0xffff
19 str r1, [r0]
20
21 ldr r2, =GPBDAT ; 将寄存器GPBDAT的地址存放到寄存器r2中
22
23 ; 全灭（低电平点亮LED）
24 ldr r1, =0x1E0 ; 0x 0001 1110 0000
25 str r1, [r2]
26
27 ledloop
28 bl delay
29 sub r1, r1, #32 ; 0x 0010 0000
30 str r1, [r2]
31
32 cmp r1, #0x0 ; 将r0的值与0进行比较
33 bne ledloop ; r1 != 0, 继续循环
34 ; 否则，重置r1
35 ldr r1, =0x1E0 ; 0x 0001 1110 0000
36 b ledloop ; 继续循环
37
38 ; 下面是延迟子程序
```

```

39 delay
40     ldr r3, =0xbffff ;设置延迟的时间
41 delay1
42     sub r3, r3, #1 ;r3=r3-1
43     cmp r3, #0x0 ;将 r3 的值与 0 相比较
44     ;比较的结果不为 0 (r3 不为 0)
45     ;继续调用 delay1, 否则执行下一条语句
46     bne delay1
47
48     mov pc, lr ;返回
49     END ;程序结束符

```

AXD 调试成功截图：

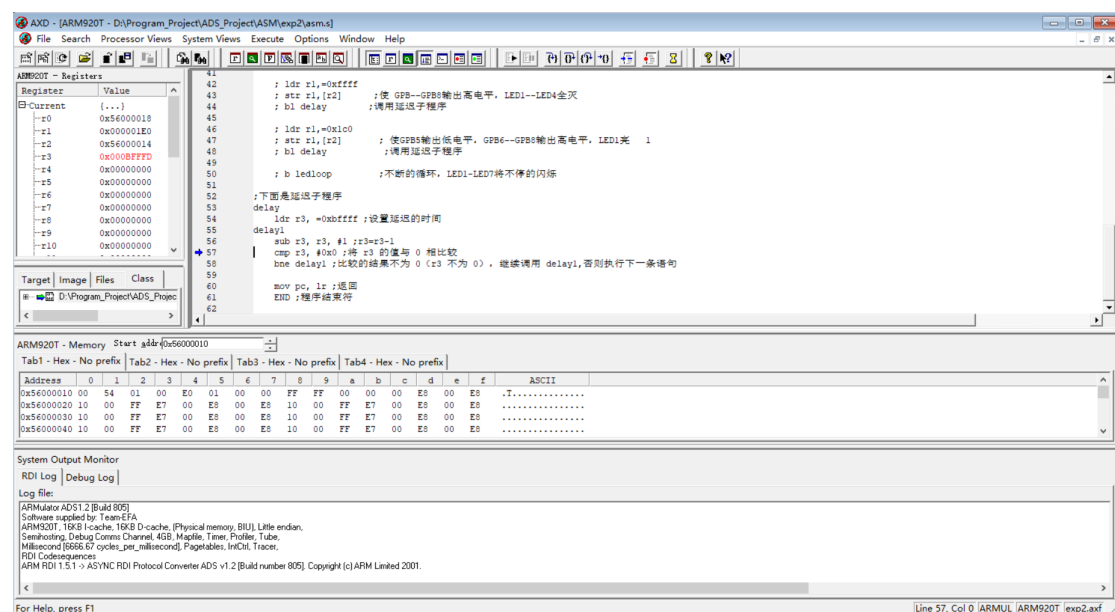


图 1: AXD 调试成功截图

## 1.2 实验三：ARM 汇编与 C 混合编程

软件实现流程图：

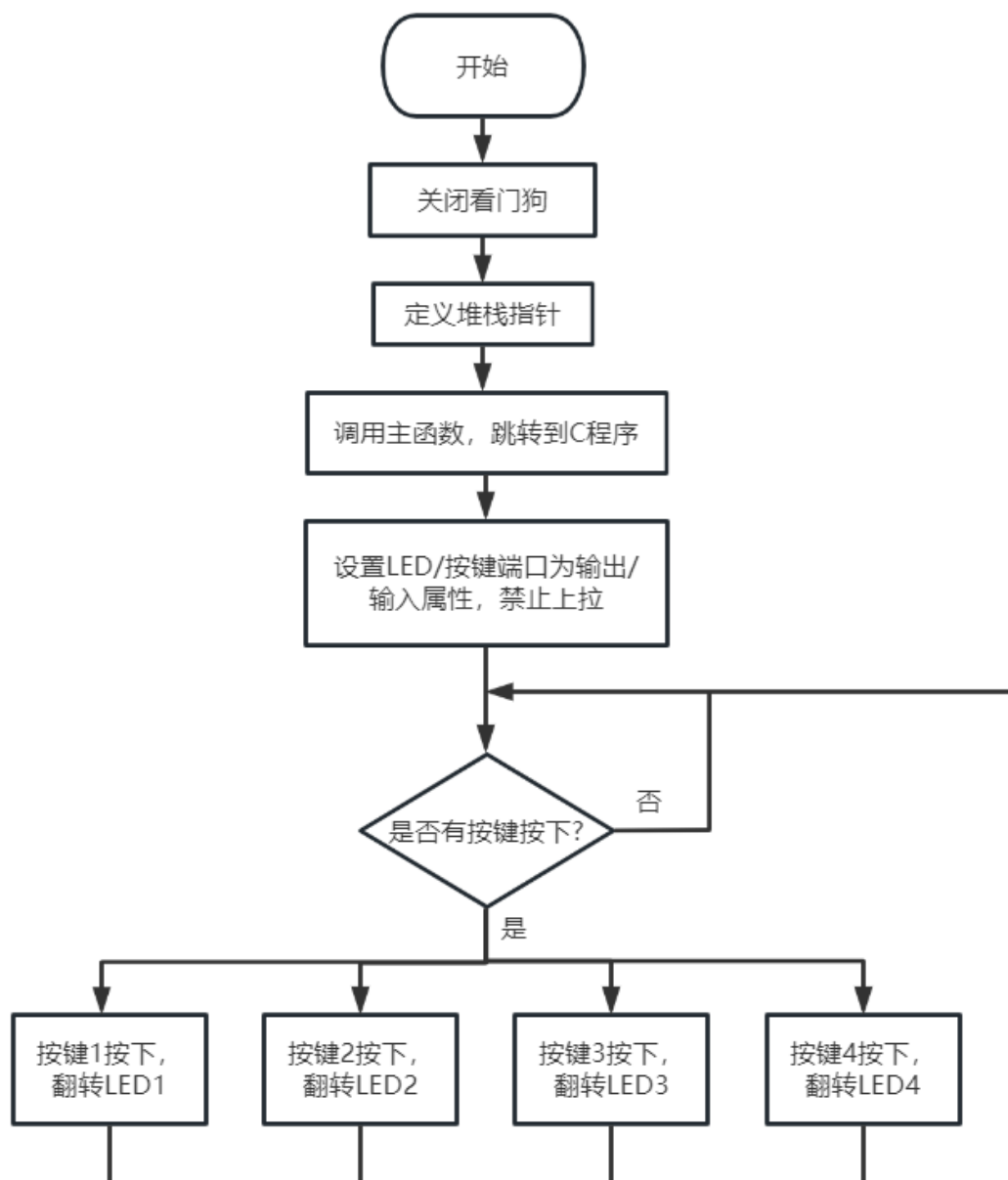


图 2: 实验三-软件实现流程图

### 1.3 实验四：ARM 中断实验

软件实现流程图：

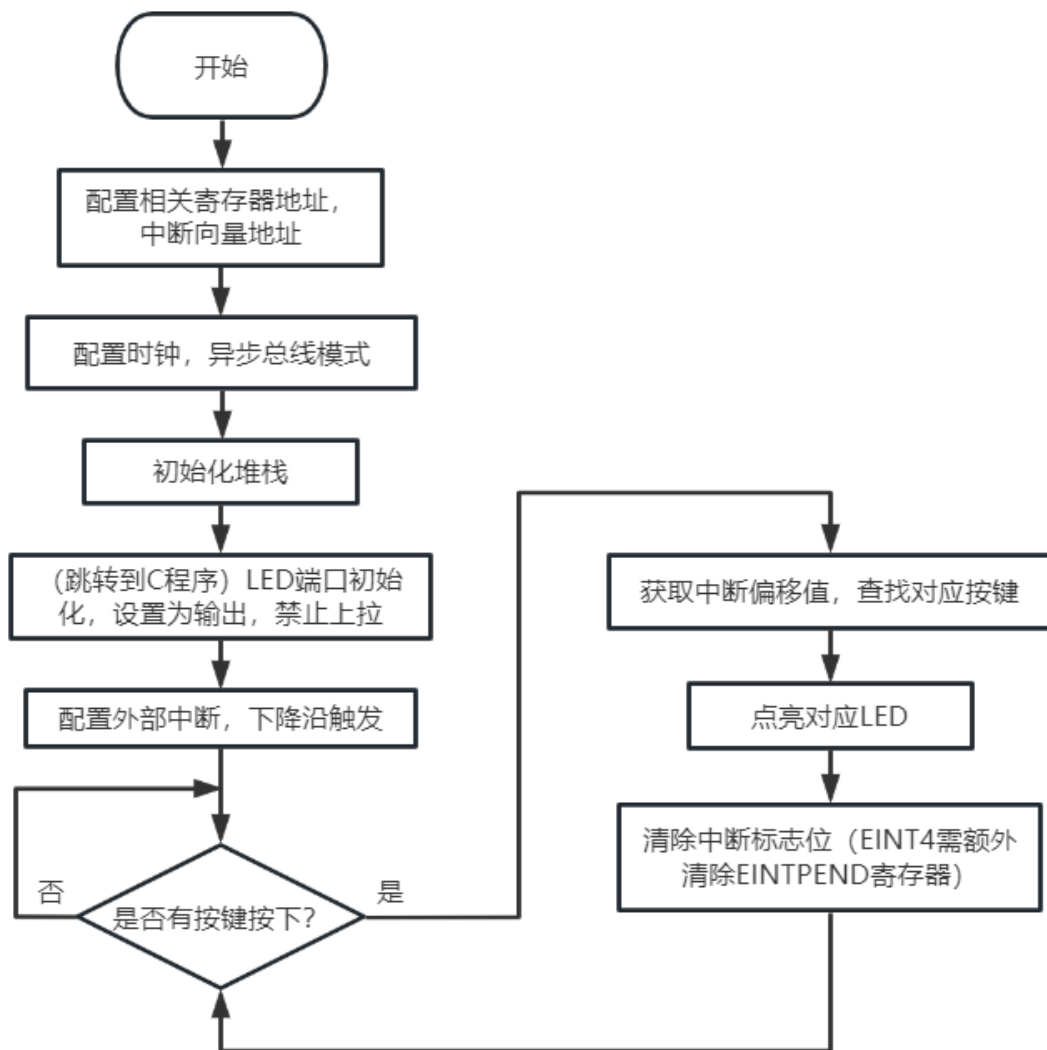


图 3: 实验四-软件实现流程图

## 1.4 思考题

### 1.4.1 在嵌入式系统编程当中, 汇编语言和 C 语言分别有什么优势? 是否可以完全摒弃其中一种语言? 为什么?

优势:

汇编语言: 针对特定硬件的编程语言, 运行速度快、代码执行效率高、实时性强, 同时占用资源少、能够直接控制硬件的工作状态。

C 语言: C 语言属于高级语言, 具有可移植性, 能够采用结构化编程。使用标准 C 语言的程序, 几乎都可以不作改变移植到不同的平台上, 并且程序容易读懂、符合逻辑习惯。

不能完全摒弃其中一种语言。汇编语言效率高, 但是可移植性差, 而 C 语言也存在资源占用多、代码利用率低等问题, 摒弃其中任何一个对于嵌入式开发都是不利的。在实际编程中应该扬长避短, 采用混合编程的形式最大限度提高代码执行效率和可读性。

#### 1.4.2 ARM 汇编调用 C 语言以及 C 语言调用 ARM 汇编时，如何传递参数？实验二、三程序中参数是如何传递的？

ARM 汇编和 C 语言的相互调用中，参数传递需要遵循 ATPCS 规则：

1. 当参数不超过 4 个时，可以使用寄存器 R0 R3 来传递参数；
2. 当参数超过 4 个时，还可以使用（满递减）数据栈来传递多出来的参数，入栈的顺序与参数顺序相反，即最后一个字数据先入栈；
3. 返回结果通过 R0 R3 来传递。

实验二、三中主要使用寄存器来传递参数。实验二中 LED 循环点亮子程序 ledloop 中使用 R1 传递 LED 对应各端口输出电平的初始值；延时子程序 delay 中使用 R3 传递需延迟的时间。实验三中延时子程序 delay 使用 R0 传递需延迟的时间。

#### 1.4.3 C 语言和汇编语言中是如何操作寄存器的？

在 C 语言中可以通过宏定义 #define 结合寄存器地址创建“寄存器变量”以间接操作寄存器，此外也可以通过位操作对寄存器进行操作。在汇编语言中，可以直接使用特定的命令来操作寄存器。

#### 1.4.4 比较实验二、三和四中 ADS 下的工程设置有何异同，并分析其理由。

实验二、实验三中“ARM Linker”下“RO Base”的地址设为 0x30000000，而在实验四中设为 0x00000000；实验二、实验三中 DNW UART/USB Options 处 Download Address 设为 0x30000000，而在实验四中设为 0x00000000。这是因为实验二、实验三是从 Nor Flash 启动，此时程序下载到 SDRAM 中地址 0x30000000 处，并从地址 0x30000000 处开始运行；而实验四在 NAND Flash 下运行中断程序，需要将下载地址设为 0x00000000，否则可能会使中断向量表的地址发生偏移，无法正常进入中断子程序，导致程序无法正常运行。

#### 1.4.5 在中断实验中为什么要把可执行程序下载到 NAND FLASH 中运行，而不是直接下载到 SDRAM 中运行？如果直接下载到 SDRAM 中运行会发生什么情况？

1. SDRAM 的地址默认是不能执行代码的，这意味着处理器不能通过简单的加载和执行指令来访问 SDRAM 中的代码；
2. 如果程序下载到 SDRAM 中运行，由于中断向量表的固定地址从 0x00000000 开始，位于 0x30000000 以下，而 SDRAM 一般是映射到地址 0x30000000 以后，因此可能会因中断向量地址不对而找不到中断函数，无法正常进入中断，从而导致程序无法正常执行。因此，通常的做法是将存储在 NAND Flash 中的操作系统内核映像复制到 RAM 中运行。

#### 1.4.6 结合实验，叙述 NAND FLASH 启动的流程。

1. 复位，使能 NAND Flash 控制器的自动引导模式；
2. CPU 执行初始化程序，将 NAND Flash 的首 4K 个字节复制到内部缓存 RAM Steppingstone 中，并将控制权转移到该区域（称为引导加载程序区 Bootloader）；

3. CPU 开始执行 Steppingstone 中的引导加载程序。在引导程序中，将程序从 NAND Flash 拷贝到 SDRAM，然后程序跳转到 SDRAM 运行。

## 1.5 体会和建议

### 体会

在第一阶段的嵌入式实验中，我对 Windows 环境下的 ARM 裸机开发流程有了初步的认识。在这段动手实践的过程中，我们小组遇到了各种各样的问题。比如由于领到的板子本身存在一定的缺陷，导致我们在实验刚开始时花费了大量的时间用于开发板接口驱动程序的安装。后来在研究生学长、老师和其他同学的帮助下，我们成功排查出了问题所在，顺利地完成了第一阶段的实验内容。

值得提到的一点是，这次实验的内容和课堂上（包括实验讲解）老师教授的偏理论方面的知识点有着十分紧密的联系。我们小组在完成第四个实验的过程中起初遇到了令人十分费解的困难，这个实验涉及到一个次级中断 EINT4 的配置，我们花了大量的时间一遍又一遍修改源代码，但是运行的效果始终和我们预想的不一致。后来在和其他同学的讨论中发现，我们忽略了 EINT4 的配置过程还需要额外修改其他两个寄存器。在仔细通读了一遍实验讲义后我们终于发现了问题所在，并且最终成功编写出了正确的配置代码，最终按键控制 LED 的效果十分完美。

### 建议

在本次实验中只对开发板上的流水灯和按键部分进行了一些了解和操作，开发板上还有许多其他资源没有被使用，希望可以在实验指导书中增加相关内容。

## 2 第二、三篇 Linux 环境下裸机开发实验

### 2.1 结合实验五，说说你所理解的 Linux 操作系统

#### 历史背景

Linux 操作系统的诞生、发展和成长过程始终依赖着五个重要支柱：Unix 操作系统、MINIX 操作系统、GNU 计划、POSIX 标准和 Internet 网络。Linux 操作系统的历史可以追溯到 1969 年，当时美国 AT&T 公司的贝尔实验室开发了一种叫做 UNIX 的操作系统。

1983 年，芬兰大学生林纳斯·托瓦兹（Linus Torvalds）开始接触计算机科学，并接触到了 UNIX 操作系统。1986 年，托瓦兹开始研究 Minix 操作系统，并在 1990 年发表了一篇名为《操作系统：设计与实现》的文章。1991 年初，托瓦兹购买了最新的 intel 386 的个人计算机，安装了 Minix 系统，开始学习 minix 操作系统。经由 Minix 系统的源码学习到了很多的内核程序设计的设计概念。

第一个与 Linux 有关的消息是在 1991 年 7 月 3 日，在 comp.os.minix 上发布的，当时还不叫 Linux，他说自己正在进行一个全新的操作系统的研发，没有使用 MINIX 一行源代码，并且想到了与 POSIX 兼容的问题。1991 年 10 月 5 日，托瓦兹在 comp.os.minix 上发布消息，正式宣布了 Linux 内核系统的诞生 (Free minix-like kernel sources for 386-AT)。这段消息相当于 Linux 的诞生宣言，并且一直广为流传，因此 10 月 5 日对 Linux 社区来说是个特殊的日子。



## 特点

开源、没有版权、技术社区用户多，开放源码使得用户可以自由裁剪，灵活性高，功能强大，成本低。

## 属性

Linux 是一个基于 POSIX 的多用户、多任务、支持多线程和多 CPU 的操作系统。

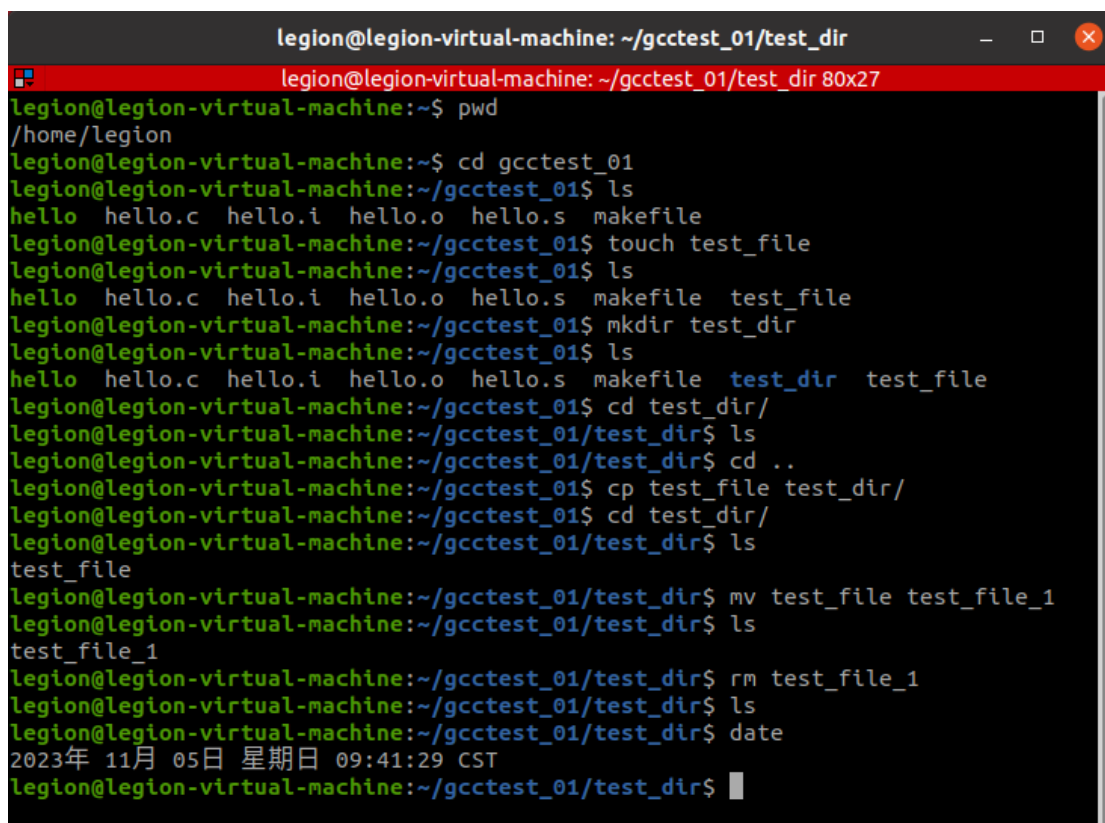
## 2.2 列出你在实验六中执行的一些常规命令，给出该行命令的功能解释，并附上命令行执行截图

### Linux 常规命令

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
9. date 获取当前系统时间

### 部分常规命令运行测试

打开 Linux 的终端，在终端中输入指令操作文件系统。通过 pwd 指令可以查看当前的工作空间为 /home/legion。通过 touch 指令创建名为 test\_file 的新文件，通过 mkdir 指令创建名为 test\_dir 的新文件夹。通过 cp 指令，将 test\_file 文件复制到 test\_dir 文件夹中，并通过 cd 和 ls 指令跳转查看相应文件夹下的文件列表。通过 mv 指令将 test\_file 的文件名修改为 test\_file1。通过 date 指令可以查看当前的时间。通过以上操作完成了对 Linux 常规命令的测试，具体测试情况如下图所示：



```
legion@legion-virtual-machine: ~/gcctest_01/test_dir
legion@legion-virtual-machine: ~/gcctest_01/test_dir 80x27
legion@legion-virtual-machine:~$ pwd
/home/legion
legion@legion-virtual-machine:~$ cd gcctest_01
legion@legion-virtual-machine:~/gcctest_01$ ls
hello hello.c hello.i hello.o hello.s makefile
legion@legion-virtual-machine:~/gcctest_01$ touch test_file
legion@legion-virtual-machine:~/gcctest_01$ ls
hello hello.c hello.i hello.o hello.s makefile test_file
legion@legion-virtual-machine:~/gcctest_01$ mkdir test_dir
legion@legion-virtual-machine:~/gcctest_01$ ls
hello hello.c hello.i hello.o hello.s makefile test_dir test_file
legion@legion-virtual-machine:~/gcctest_01$ cd test_dir/
legion@legion-virtual-machine:~/gcctest_01/test_dir$ ls
legion@legion-virtual-machine:~/gcctest_01/test_dir$ cd ..
legion@legion-virtual-machine:~/gcctest_01$ cp test_file test_dir/
legion@legion-virtual-machine:~/gcctest_01$ cd test_dir/
legion@legion-virtual-machine:~/gcctest_01/test_dir$ ls
test_file
legion@legion-virtual-machine:~/gcctest_01/test_dir$ mv test_file test_file_1
legion@legion-virtual-machine:~/gcctest_01/test_dir$ ls
test_file_1
legion@legion-virtual-machine:~/gcctest_01/test_dir$ rm test_file_1
legion@legion-virtual-machine:~/gcctest_01/test_dir$ ls
legion@legion-virtual-machine:~/gcctest_01/test_dir$ date
2023年 11月 05日 星期日 09:41:29 CST
legion@legion-virtual-machine:~/gcctest_01/test_dir$
```

图 4: 指令运行测试-命令行执行截图

## 2.3 列出实验六、实验八的实验过程，编写并编译自己的第一个 C 程序（硬件无关），并附实验截图

实验六：Linux 常用命令的熟悉，实验过程见上（图4）。

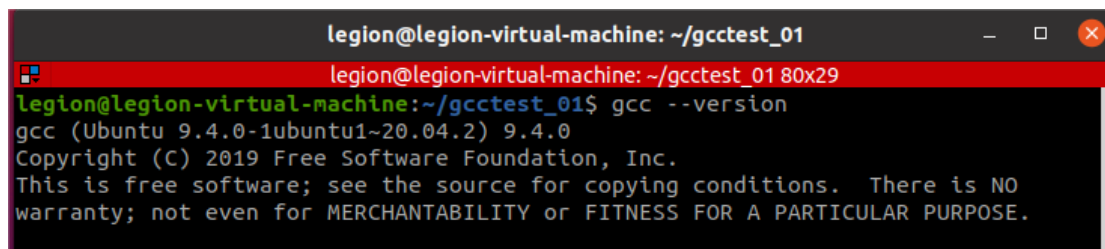
实验八：编译工具 GCC 的使用，实验过程如下：

### 实验环境配置

实验环境：Windows 10, VMware 17, Ubuntu 20.04

### GCC 编译及结果截图

GCC 安装测试：GCC 编译器是 Linux 系统下最常用的 C/C++ 编译器，根据帮助文档完成虚拟机的安装和 Linux 系统的搭建，通过终端安装 GCC 编译工具，并通过指令 `gcc --version` 确认 GCC 安装情况和查看 GCC 版本信息，测试结果如图所示：

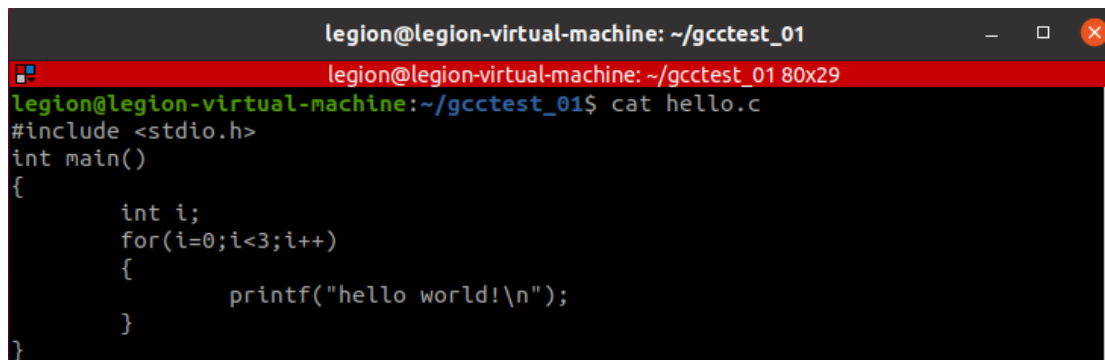


```
legion@legion-virtual-machine: ~/gcctest_01
legion@legion-virtual-machine: ~/gcctest_01 80x29
legion@legion-virtual-machine:~/gcctest_01$ gcc --version
gcc (Ubuntu 9.4.0-1ubuntu1~20.04.2) 9.4.0
Copyright (C) 2019 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

图 5: GCC 安装版本信息

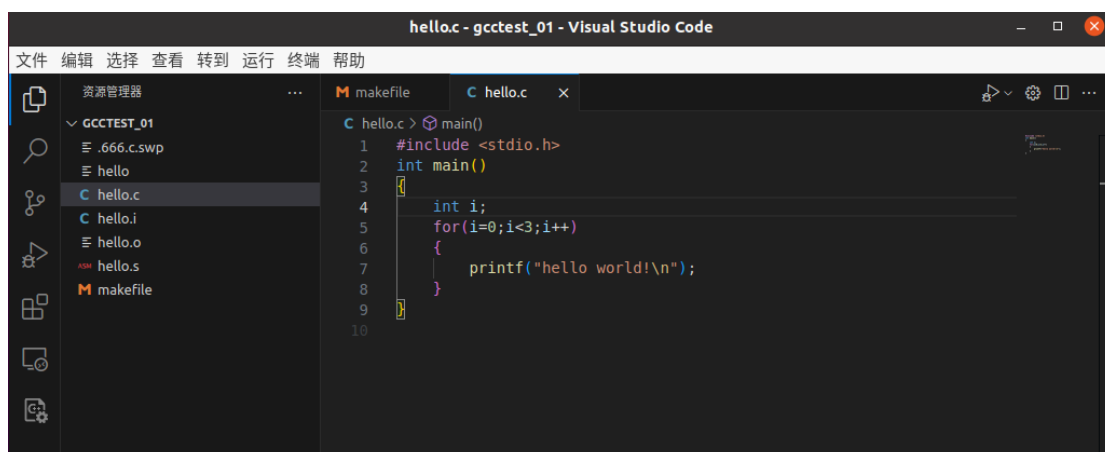
## 源文件编辑

在终端中通过指令 `mkdir` 创建名为 `gcctest_01` 的文件夹，在文件夹中通过指令 `vim hello.c` 创建同名文件并在 `vim` 编辑器中进行代码编辑。在编辑器中输入 `c` 语言代码，并通过 `wq` 指令保存代码并退出。通过指令 `cat hello.c` 便可查看文件中的代码。也可以在文件夹 `gcctest_01` 使用指令 `code .` 即可通过 `vscode` 打开文件进行代码编辑。测试结果如图 3，图 4 所示。



```
legion@legion-virtual-machine: ~/gcctest_01
legion@legion-virtual-machine: ~/gcctest_01 80x29
legion@legion-virtual-machine: ~/gcctest_01$ cat hello.c
#include <stdio.h>
int main()
{
    int i;
    for(i=0;i<3;i++)
    {
        printf("hello world!\n");
    }
}
```

图 6: 终端指令 `cat` 显示代码



```
hello.c - gcctest_01 - Visual Studio Code
文件 编辑 选择 查看 转到 运行 终端 帮助
资源管理器
GCCTEST_01
  .666.c.swp
  hello
  C hello.c
  C hello.i
  hello.o
  hello.s
  makefile
C hello.c > main()
1 #include <stdio.h>
2 int main()
3 {
4     int i;
5     for(i=0;i<3;i++)
6     {
7         printf("hello world!\n");
8     }
9 }
10
```

图 7: `vscode` 编辑代码

## GCC 编译测试

GCC 编译流程包括以下 4 个步骤（以源文件 `hello.c` 为例）：

1. 预处理（进行宏替换），指令：`gcc -e hello.c -o hello.i`
2. 编译（生成汇编），指令：`gcc -s hello.i -o hello.s`
3. 汇编（生成机器可识别代码），指令：`gcc -c hello.s -o hello.o`
4. 连接（生成可执行文件或库文件），指令：`gcc hello.o -o hello`

开启已安装 GCC 的 Ubuntu 终端，可以使用上述指令编译文件。具体测试流程图所示：

```
legion@legion-virtual-machine: ~/gcctest_01
legion@legion-virtual-machine: ~/gcctest_01 80x29
legion@legion-virtual-machine:~/gcctest_01$ ls
hello.c  makefile
legion@legion-virtual-machine:~/gcctest_01$ gcc -E hello.c -o hello.i
legion@legion-virtual-machine:~/gcctest_01$ ls
hello.c  hello.i  makefile
legion@legion-virtual-machine:~/gcctest_01$ gcc -S hello.i -o hello.s
legion@legion-virtual-machine:~/gcctest_01$ ls
hello.c  hello.i  hello.s  makefile
legion@legion-virtual-machine:~/gcctest_01$ gcc -c hello.s -o hello.o
legion@legion-virtual-machine:~/gcctest_01$ ls
hello.c  hello.i  hello.o  hello.s  makefile
legion@legion-virtual-machine:~/gcctest_01$ gcc hello.o -o hello
legion@legion-virtual-machine:~/gcctest_01$ ls
hello  hello.c  hello.i  hello.o  hello.s  makefile
legion@legion-virtual-machine:~/gcctest_01$ ./hello
hello world!
hello world!
hello world!
```

图 8: GCC 编译测试

为了简化编译流程, GCC 也提供了指令 `gcc -o hello hello.c`, 能够一步到位将 .c 文件编译为可执行文件, 但此方法不会生成编译流程中的中间文件, 不便于实验中理解 GCC 编译流程, 测试结果如图所示:

```
legion@legion-virtual-machine: ~/gcctest_01
legion@legion-virtual-machine: ~/gcctest_01 80x24
legion@legion-virtual-machine:~$ ls
公共的  视频  文档  音乐  catkin_ws  gcctest_02  roctest_03  test02_vscode
模板    图片  下载  桌面  gcctest_01  roctest_01  snap
legion@legion-virtual-machine:~$ cd gcctest_01
legion@legion-virtual-machine:~/gcctest_01$ ls
hello.c  makefile
legion@legion-virtual-machine:~/gcctest_01$ cat hello.c
#include <stdio.h>
int main()
{
    int i;
    for(i=0;i<3;i++)
    {
        printf("hello world!\n");
    }
}
legion@legion-virtual-machine:~/gcctest_01$ gcc -o hello hello.c
legion@legion-virtual-machine:~/gcctest_01$ ls
hello  hello.c  makefile
legion@legion-virtual-machine:~/gcctest_01$ ./hello
hello world!
hello world!
hello world!
```

图 9: GCC 单指令编译测试

## 2.4 列出实验九、十的实验过程, 并附实验截图

### 交叉编译工具 arm-linux-gcc 和 GUN make 工具的使用

交叉编译是指在一个平台上生成另一个平台可执行的应用程序。交叉编译工具 arm-linux-gcc 是在桌面 Linux 下生成 arm 开发板可执行的应用程序。GUN make 是 Linux 程

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。为了使用 make 命令完成工程项目内所有源文件的自动编译，需要我们投入一些时间去完成一个或者多个 Makefile 文件的编写。

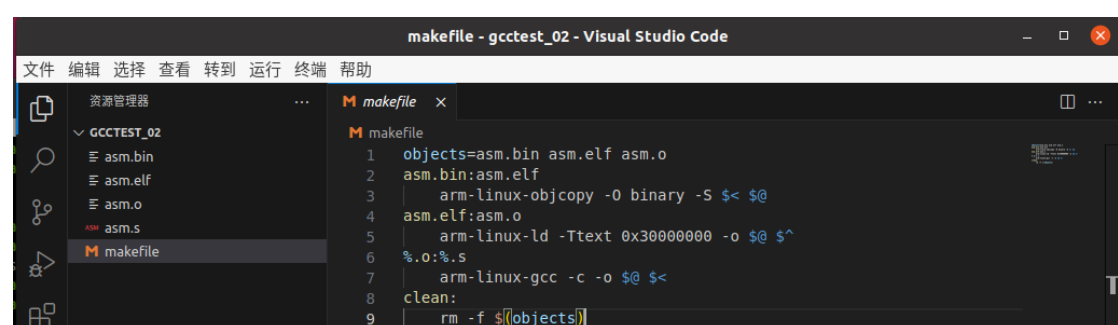
根据实验帮助文档在 Linux 终端下安装交叉编译工具 arm-linux-gcc，并通过命令 arm-linux-gcc -v 获取其安装版本信息和安装情况，测试结果如下图所示：

```
legion@legion-virtual-machine:~/gcctest_01$ arm-linux-gcc -v
Using built-in specs.
Target: arm-none-linux-gnueabi
Configured with: /opt/FriendlyARM/mini2440/build-toolschain/working/src/gcc-4.4.3/configure --build=i386-build_redhat-linux-gnu --host=i386-build_redhat-linux-gnu --target=arm-none-linux-gnueabi --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=arm920t --with-tune=arm920t --with-float=soft --with-pkgversion=ctng-1.6.1 --disable-sjlj-exceptions --enable-__cxa_atexit --with-gmp=/opt/FriendlyARM/toolschain/4.4.3 --with-mpfr=/opt/FriendlyARM/toolschain/4.4.3 --with-ppl=/opt/FriendlyARM/toolschain/4.4.3 --with-cloog=/opt/FriendlyARM/toolschain/4.4.3 --with-mpc=/opt/FriendlyARM/toolschain/4.4.3 --with-local-prefix=/opt/FriendlyARM/toolschain/4.4.3/arm-none-linux-gnueabi//sys-root --disable-nls --enable-threads=posix --enable-symvers=gnu --enable-c99 --enable-long-long --enable-target-optspace
Thread model: posix
gcc version 4.4.3 (ctng-1.6.1)
```

图 10: arm-linux-gcc 安装版本信息

## makefile 文件编写

首先通过指令 touch makefile 创建一个名为 makefile 的 Makefile 文件，通过指令 vscode 开启 vscode 编写编译所需要的依赖项和目标文件并保存，回到终端后切换到 gcctest\_02 文件夹下，通过指令 ls 可以查看当前文件夹中的文件，通过指令 make 便可完成对 asm.s 的编译工作，再次使用指令 ls 可以看到生成了名为 asm.bin 的二进制文件和两个中间文件，此时便通过 Makefile 文件完成了对源文件的编译工作。



```
makefile - gcctest_02 - Visual Studio Code
文件 编辑 选择 查看 转到 运行 终端 帮助
资源管理器
GCCTEST_02
asm.bin
asm.elf
asm.o
asm.s
makefile
makefile
1 objects=asm.bin asm.elf asm.o
2 asm.bin:asm.elf
3 arm-linux-objcopy -O binary -S $< $@
4 asm.elf:asm.o
5 arm-linux-ld -Ttext 0x30000000 -o $@ $^
6 %.o:%.s
7 arm-linux-gcc -c -o $@ $<
8 clean:
9 rm -f $[objects]
```

图 11: Makefile 文件编辑

```
legion@legion-virtual-machine: ~/gcctest_02
legion@legion-virtual-machine: ~/gcctest_02 80x24
legion@legion-virtual-machine:~/gcctest_02$ ls
asm.s  makefile
legion@legion-virtual-machine:~/gcctest_02$ make
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
legion@legion-virtual-machine:~/gcctest_02$ ls
asm.bin  asm.elf  asm.o  asm.s  makefile
legion@legion-virtual-machine:~/gcctest_02$
```

图 12: Makefile 文件编译

## 下载 LED 流水灯程序到开发板并运行

将 Linux 下交叉编译生成的可执行文件拷贝到 Windows 操作系统下，启动开发板，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。

## Linux 系统下驱动程序的使用

将 Linux 系统移植到开发板上：依次分别烧写 u-boot、内核 zImage、文件系统 rootfs，将开发板启动开关拨到 Nand，启动开发板，将所给源码中对应的三个文件拷贝到 U 盘。将 U 盘插入开发板的 USB 接口，测试 LED 驱动程序，实现对 LED 的操作，如下图所示：

```
Serial-COM3 - SecureCRT
文件(F) 编辑(E) 查看(V) 选项(O) 传输(T) 脚本(S) 工具(L) 帮助(H)
Serial-COM3
sd 0:0:0:0: [sda] 122880000 512-byte hardware sectors (62915 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!
ls
bin      lib      opt      sddisk   usr
dev      linuxrc  proc     sys      var
etc      lost+found  root    tmp      web
home     mnt      sbin     udisk
[root@EmbedSky /]# cd udisk/
[root@EmbedSky /udisk]# ls
EmbedSky_hello.ko      leds
EmbedSky_leds.ko        连接设备? [0m
System Volume Information  连接设备? [0m
[root@EmbedSky /udisk]# insmod EmbedSky_hello.ko

Hello, I Love CUG!

This is first driver program.

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

Exit!

Goodbye CUG!

[root@EmbedSky /udisk]# insmod EmbedSky_leds.ko
TQ2440 LED Driver, (c) 2008,2009 www.cug.edu.cn
tq2440-leds initialized
[root@EmbedSky /udisk]# ls /dev/tq2440-leds
/dev/tq2440-leds
[root@EmbedSky /udisk]# ./leds 5 0
[root@EmbedSky /udisk]# ./leds 5 1
[root@EmbedSky /udisk]# ./leds 2 1
[root@EmbedSky /udisk]#
```

图 13: LED 驱动测试

## 2.5 思考题

### 2.5.1 思考 Windows 环境下与 Linux 环境下开发裸机程序的区别有什么？

环境不同：Windows 环境下开发程序需要在特定软件上（ADS 或是 Keil）编写源代码，通过编译生成可执行文件，而 Linux 环境下只需安装交叉编译工具即可完成上述操作；

可视化程度不同：Windows 环境下编程软件拥有丰富的 UI 设计和交互操作，而 Linux 环境下一般只有单一命令行输入的交互，且 Linux 区分大小写，windows 在 dos 界面命令下不区分大小写；

开发流程不同：Windows 下使用 ADS 工具开发流程一般为：编辑 → 编译 → 执行 → 烧写到开发板启动并运行。Linux 下使用命令终端开发流程一般为：编辑 → 写 makefile，执行 make 编译 → 烧到开发板启动并运行。

### 2.5.2 make 及 makefile 的作用是什么？

在 Linux 系统中，专门提供了一个 make 命令来自动维护目标文件。因为在 Linux 中，一个文件被创建或更新后有一个最后修改时间，make 命令就是通过这个最后修改时间来判断此文件是否被修改，而对没修改的文件则不做更改，并且 make 命令不会漏掉一个需要更新的文件。

makefile 定义了一系列的规则来指定，哪些文件需要先编译，哪些文件需要后编译，哪些文件需要重新编译，甚至于进行更复杂的功能操作，因为 makefile 就像一个 Shell 脚本一样，其中也可以执行操作系统的命令。makefile 带来的好处就是“自动化编译”，一旦写好，只需要一个 make 命令，整个工程完全自动编译，极大地提高了软件开发的效率。

### 2.5.3 第二篇实验与第三篇实验的 Linux 系统一样吗？如不同，不同之处有什么？

不一样，第二篇实验的 Linux 系统是在虚拟机上运行的桌面 Linux 系统，是软件公司或个人计算机开发的操作系统；而第三篇实验的 Linux 系统是在开发板上运行的系统，是经过裁剪定制，仅保留部分核心功能的 Linux 系统，即嵌入式 Linux。

### 2.5.4 关于 Linux 系统，你还想要了解什么？

Linux 在日常工作中的应用，Linux 系统下基于其他编程语言（如 Python）的开发方式。

### 2.5.5 关于嵌入式 Linux 操作系统的开发，你还希望学习什么？

搭建基于 Linux 的个人服务器，在嵌入式 Linux 系统中部署 AI 应用。

## 2.6 体会和建议

### 体会

经过本阶段的实验，我对 Linux 操作系统的使用有了一定的了解和熟悉。相比常见的 Windows 操作系统，Linux 中的多数工作都可以在命令终端下完成，通过命令行的形式让计算机执行我们想要的操作，比如打开一个文件或应用程序等。这种高效的操作方式让我耳目一新，勾起了我对 Linux 操作系统的浓厚兴趣。在未来，我会继续深入了解 Linux，并努力将它融入到日常学习工作的应用场景中，利用 Linux 提高学习和工作效率。

## 建议

目前实验室提供的开发板版本太旧，且其中有一些板子存在问题，有条件的话建议把这些设备更新换代一下。另外，也许可以考虑把课内实验和实习内容整合一下，把这两个实践内容合并到一起。