



合肥工业大学

# 嵌入式系统原理

## 实 验 指 导 书

合肥工业大学计算机与信息学院

《嵌入式系统原理》课程组

2021年5月

# 目 录

实验一 熟悉 Linux 开发环境 .....	4
一、实验目的 .....	4
二、实验内容 .....	4
三、预备知识 .....	4
四、实验设备及工具（包括软件调试工具） .....	4
五、实验步骤 .....	4
六、实验修改要求 .....	10
实验二 汇编点亮 LED .....	11
一、实验目的 .....	11
二、实验内容 .....	11
三、预备知识 .....	11
四、实验设备及工具（包括软件调试工具） .....	11
五、实验原理 .....	11
六、实验步骤 .....	12
七、实验修改要求 .....	16
实验三 查询方式检测按键 .....	17
一、实验目的 .....	17
二、实验内容 .....	17
三、预备知识 .....	17
四、实验设备及工具（包括软件调试工具） .....	17
五、实验原理 .....	17
六、实验步骤 .....	19
七、实验修改要求 .....	21
实验四 按键中断实验 .....	22
一、实验目的 .....	22
二、实验内容 .....	22
三、预备知识 .....	22
四、实验设备及工具（包括软件调试工具） .....	22
五、实验原理 .....	22
六、实验步骤 .....	24
七、实验修改要求 .....	28
实验五 PWM 定时器实验 .....	29
一、实验目的 .....	29
二、实验内容 .....	29
三、预备知识 .....	29
四、实验设备及工具（包括软件调试工具） .....	29
五、实验原理 .....	29
六、实验步骤 .....	30
七、实验修改要求 .....	35
实验六 串行口通信实验 .....	36
一、实验目的 .....	36
二、实验内容 .....	36

三、预备知识.....	36
四、实验设备及工具（包括软件调试工具） .....	36
五、实验原理.....	36
六、实验步骤.....	39
七、实验修改要求.....	42

# 实验一 熟悉 Linux 开发环境

## 一、实验目的

熟悉Linux开发环境，学会基于Tiny6410的Linux开发环境的配置和使用。  
使用arm-Linux-gcc编译，并使用mini com实现串口方式下载和调试程序。

## 二、实验内容

本次实验使用Fedora（合肥校区）/CentOS（宣城校区）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hello，并在其中编写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。下载已经编译好的文件到目标开发板上运行。

## 三、预备知识

1. 有C语言基础。
2. 会使用Linux下常用的编辑器。
3. 掌握Makefile的编写和使用。
4. 了解Linux下的编译程序与交叉编译的过程。

## 四、实验设备及工具（包括软件调试工具）

硬件：Tiny6410嵌入式实验平台。

软件：PC机操作系统Fedora/CentOS+mini com+ARM-Linux 开发环境

## 五、实验步骤

- 1、登录win7系统，使用管理员权限打开VMware虚拟机软件。确认虚拟机中已安装Fedora(合肥校区)/CentOS（宣城校区）系统，否则请通过镜像文件安装。
- 2、确认虚拟机串口已打开（虚拟机右下角显示0|0图标且不为灰色），否则通过【编辑此虚拟机】选项，使用“添加”功能添加串行通信端口，并确认使用物理串行端口中为“自动选择串口”选项。

3、通过【启动此虚拟机】，启动Fedora（合肥校区）/CentOS\_QT\_6410（宣城校区）系统。合肥校区请选择knight用户，并输入登录密码knight；宣城校区启动后自动以root用户登录。

4、打开终端（terminal），建立hello工作目录。（合肥校区请查看home文件夹，宣城校区请查看root/linux实验例程文件夹。若已存在hello目录，则跳过该步骤。可通过ls命令查看是否存在hello目录。）

```
[root@zxt smile]# mkdir hello
[root@zxt smile]# cd hello
```

此时新建的hello工作目录，会在对应的目录下出现，说明此次操作成功(注意，记清楚所创建目录的位置)，如下图所示：

```
[root@localhost home]# mkdir hello
[root@localhost home]# ls
hello rdy
[root@localhost home]# cd hello
```

5、编写程序源代码。（若虚拟机中已有初始源代码，可直接编辑）

hello.c 源代码较简单，如下：

```
#include <stdio.h>

int main()
{
    printf("Hello,World!\n");

    return 0;
}
```

可以用下面的命令来编写hello.c的源代码，进入hello目录使用vim命令来编辑代码(也可以使用gedit命令来编辑hello.c文件：gedit hello.c)：

```
[root@zxt hello]# vi hello.c
```

按“i”键或者“A”键进入编辑模式，将上面的代码录入进去，完成后按Esc键进入命令状态，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。

## 6、编写Makefile。（若虚拟机中已有Makefile文件，可跳过该步）

要使上面的hello.c程序能够运行，必须要编写一个Makefile文件，Makefile文件定义了一系列的规则，它指明了哪些文件需要编译，哪些文件需要先编译，哪些文件需要重新编译等等更为复杂的命令。使用它带来的好处就是自动编译，只需要敲一个“make”命令整个工程就可以实现自动编译，当然本次实验只有一个文件，它还不能体现出使用Makefile的优越性，但当工程比较大文件比较多时，不使用Makefile几乎是不可能的。下面介绍本次实验用到的Makefile文件。

```
CC=arm-linux-gcc
EXEC=hello
OBS=hello.o
CFLAGS+=
LDFLAGS+=

all:$(EXEC)
$(EXEC):$(OBS)
    $(CC) $(LDFLAGS) -o $@ $(OBS)

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

★注意：“\$(CC) \$(LDFLAGS) -o \$@ \$(OBS)”和“-rm -f \$(EXEC) \*.elf \*.gdb \*.o”前**空白由一个Tab制表符生成，不能单纯由空格来代替**。

与上面编写hello.c的过程类似，用vim来创建一个Makefile文件并将代码录入其中。

```
[root@zxt hello]# vi Makefile
```

## 7、编译应用程序。

在上面的步骤完成后，就可以在hello目录下运行“make”来编译程序。如果进行了修改，重新编译则运行：

```
[root@zxt hello]# make clean
[root@zxt hello]# make
```

★注意：编译、修改程序都是在宿主机（本地PC 机）上进行，不能在mini com下进行。

## 8、下载调试。

① 进入root。（若当前已经是root用户，则此步跳过）

```
[Knight@localhost hello]$ su root
Password:
[root@localhost hello]#
```

终端输入su root，再输入密码knight即可（合肥校区）。

② 首先使用串口线连接开发板的串口（COM0）和PC机的串口；然后将Tiny6410开发板的启动模式拨动开关S2拨至“Nand”位置，表示从Nand Flash启动系统；再将电源拨动开关S1拨至“ON”，启动嵌入式Linux系统运行，此时开发板LCD屏幕左上角显示企鹅图标，待系统初始化完毕显示主页面。（备注：若上电后屏幕黑屏，且模式开关位置正确，则联系实验指导教师为开发板烧写系统）。

③ 在虚拟机的终端中输入minicom命令，若提示Device /dev/ttyUSB0 access failed: No such file or directory错误信息，则命令行输入minicom -s，用键盘方向键选择Serial port setup后回车，然后按“A”键将ttyUSB0（合肥校区）/tty0（宣城校区）修改为ttyS0（合肥校区）/ttyS1（宣城校区）（注意S大写），然后回车返回主界面，再选择Save setup as dfl后回车，最后选择Exit退出。执行minicom命令成功打开串口后，出现下图：



```
Knight@localhost:/home/Knight/hello
File Edit View Terminal Tabs Help

Welcome to minicom 2.3

OPTIONS: I18n
Compiled on Mar 13 2008, 00:58:14.
Port /dev/ttyUSB0

Press CTRL-A Z for help on special keys
```

回车后，出现下图：



```
Knight@localhost:/home/Knight/hello
File Edit View Terminal Tabs Help

Welcome to minicom 2.3

OPTIONS: I18n
Compiled on Mar 13 2008, 00:58:14.
Port /dev/ttyUSB0

Press CTRL-A Z for help on special keys

[root@FriendlyARM /]#
[root@FriendlyARM /]#
```

★必须出现红框中的路径（开发板中嵌入式Linux的路径）才能进行正常的下载，若没有出现，表明串口没有正常连接，需要检查硬件连线及相关串口参数。

**注意：**关闭mini com请先按Ctrl+A，再按X。不要使用界面右上角的×退出，否则会在下一次执行mini com命令时提示串口被锁定，拒绝访问。若出现该问题，可以通过重启虚拟机解决。

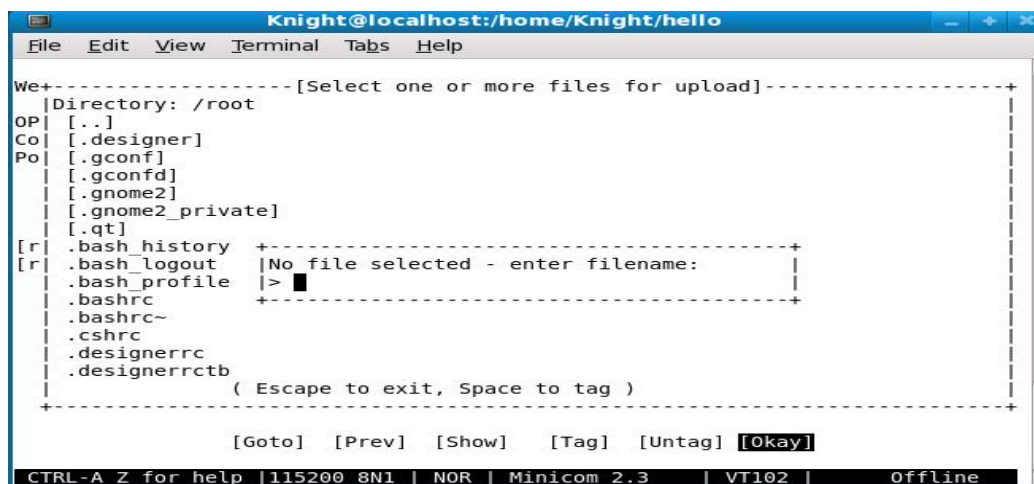
④ 按Ctrl+A，再按S，出现下图：



选择第一个zmodem，回车后，出现下图：

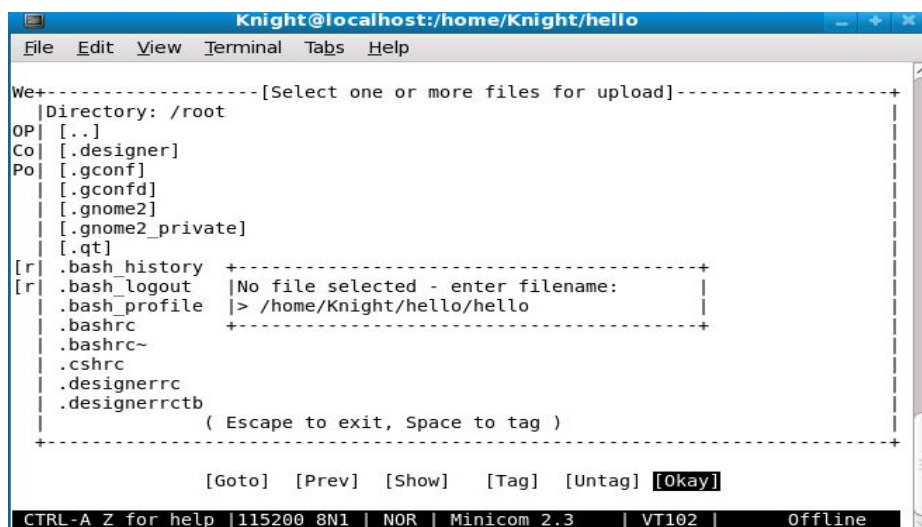


回车后，出现下图：





输入绝对路径或者通过键盘快捷键（双击空格键）选择文件后，出现下图：



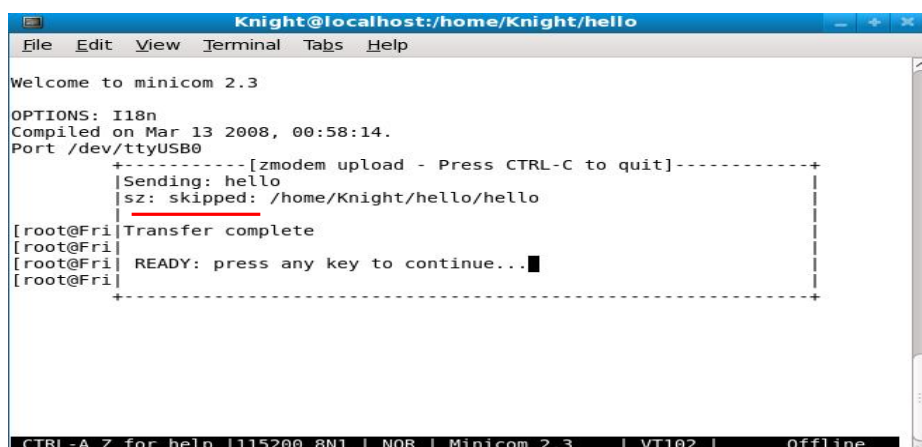
```
Knight@localhost:/home/Knight/hello
File Edit View Terminal Tabs Help

We+-----[Select one or more files for upload]-----+
|Directory: /root
OP| [..]
Co| [.designer]
Po| [.gconf]
| [.gconfd]
| [.gnome2]
| [.gnome2_private]
| [.qt]
[r] .bash_history +-----+
[r] .bash_logout |No file selected - enter filename:
| .bash_profile |> /home/Knight/hello/hello
| .bashrc +-----+
| .bashrc~
| .cshrc
| .designerrc
| .designerrc.tb
|
| ( Escape to exit, Space to tag )
+-----+

[Goto] [Prev] [Show] [Tag] [Untag] [Okay]

CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.3 | VT102 | Offline
```

回车，出现下图：



```
Knight@localhost:/home/Knight/hello
File Edit View Terminal Tabs Help

Welcome to minicom 2.3
OPTIONS: I18n
Compiled on Mar 13 2008, 00:58:14.
Port /dev/ttyUSB0

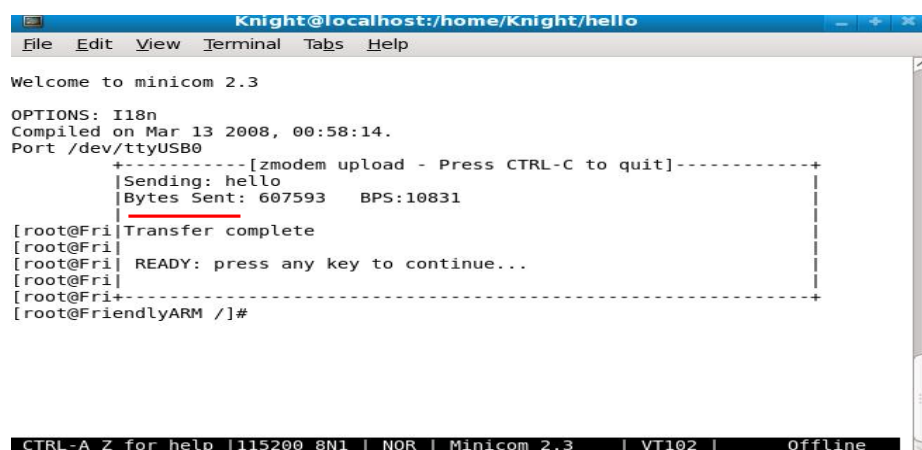
+-----[zmodem upload - Press CTRL-C to quit]-----+
|Sending: hello
|sz: skipped: /home/Knight/hello/hello
+-----+

[root@Fri] Transfer complete
[root@Fri]
[root@Fri] READY: press any key to continue...
[root@Fri]

CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.3 | VT102 | Offline
```

从上图可以看出，文件未被下载，原因是Tiny6410板子已经有了hello可执行文件（之前下载的），若需要下载，则需要删除之前的hello文件，rm hello即可。可通过ls命令查看是否删除成功。

删除之后，继续下载，出现下图，表示下载成功。



```
Knight@localhost:/home/Knight/hello
File Edit View Terminal Tabs Help

Welcome to minicom 2.3
OPTIONS: I18n
Compiled on Mar 13 2008, 00:58:14.
Port /dev/ttyUSB0

+-----[zmodem upload - Press CTRL-C to quit]-----+
|Sending: hello
|Bytes Sent: 607593 BPS:10831
+-----+

[root@Fri] Transfer complete
[root@Fri]
[root@Fri] READY: press any key to continue...
[root@Fri]
[root@Fri]
[root@Fri] FriendlyARM /]#

CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.3 | VT102 | Offline
```

⑤ 运行程序。

在电脑终端（当前路径是开发板嵌入式Linux的路径）中输入./hello，或者在Tiny6410开发板终端输入hello都可。

## 六、实验修改要求

将原程序中显示的“Hello, World!”修改为“Hello, HFUT!”，并增加一行，显示本人学号和实验日期，例如“2019123456|2021-05-08”。

## 实验二 汇编点亮 LED

### 一、实验目的

学会Linux系统中开发汇编程序的步骤和方法。在此基础上，掌握通过汇编程序访问GPIO端口，以实现控制Tiny6410开发板上LED的方法。

### 二、实验内容

本次实验使用Fedora（合肥校区）/CentOS（宣城校区）操作系统环境，安装ARM-Linux的开发库及编译器。学习在Linux下的编程和编译过程，即创建一个新目录leds\_s，使用编辑器建立start.S和Makefile文件，并使用汇编语言编写LED控制程序。编译程序，并下载文件到目标开发板上运行。

### 三、预备知识

- 1、清楚ARM微处理器芯片S3C6410的GPIO口硬件资源及相应寄存器结构。
- 2、有ARM汇编语言基础。
- 3、会使用Linux下常用的编辑器。
- 4、掌握Makefile的编写和使用。
- 5、了解Linux下的编译程序与交叉编译的过程。

### 四、实验设备及工具（包括软件调试工具）

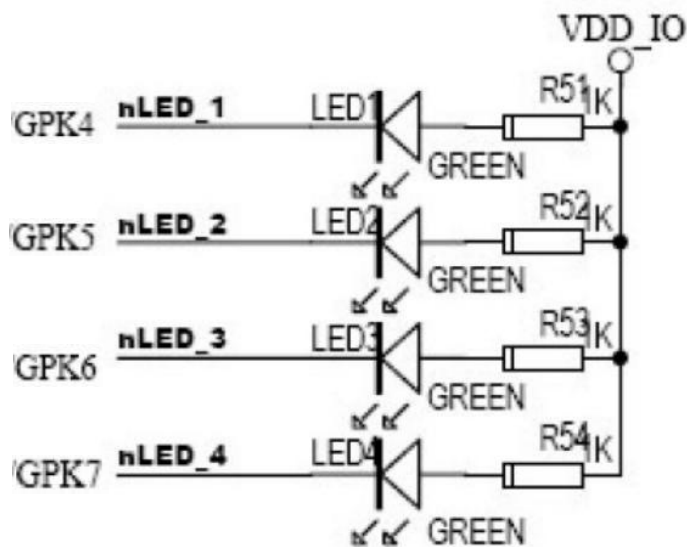
硬件：Tiny6410嵌入式实验平台。

软件：PC机操作系统Fedora/CentOS+Mini Tools+ARM-Linux开发环境

### 五、实验原理

#### 1、LED硬件原理图

Tiny6410开发板上提供了4个可编程用户LED，原理图如下：



LED硬件原理图

可见，LED1、2、3、4分别使用的CPU GPIO端口资源为GP\_K4、5、6、7。

## 2、点亮LED控制原理

点亮4个LED需如下2个步骤：

- 把外设的基地址告诉CPU。对于6410来说，内存的地址范围为0x00000000-0x60000000，外设的地址范围为0x70000000-0x7fffffff。
- 关闭看门狗，防止程序不断重启；设置寄存器GPKCON0，使GP\_K4/5/6/7四个引脚为输出功能；往寄存器GPKDAT写0，使GP\_K4/5/6/7四个引脚输出低电平，4个LED会亮；相反，往寄存器GPKDAT写1，使GP\_K4/5/6/7四个引脚输出高电平，4个LED会灭。

## 六、实验步骤

### 1、建立工作目录leds。（若系统中已建立该目录，可跳过本步骤）

点击【虚拟机】菜单中的【设置】，选择【选项】中的“共享文件夹”，添加Windows系统中的桌面路径为共享文件夹。在Windows系统的桌面上，右键复制leds文件夹，然后进入虚拟机当前用户的Home目录，使用右键粘贴，将文件夹从windows系统复制到虚拟机的系统中。

### 2、编写程序源代码。（若虚拟机中已有初始源代码，可直接编辑）

在Linux下的文本编辑器有许多，常用的是vim和Xwindow界面下的gedit

等，建议在实验中使用vim（需要学习vim的操作方法，请参考相关书籍中的关于vim的操作指南）。

① start.S的汇编源程序如下：

```
.global _start
_start:
// 把外设的基地址告诉CPU
ldr r0, =0x70000000 //对于6410来说,内存(0x00000000~0x60000000),外设(0x70000000~0x7FFFFFFF)
orr r0, r0, #0x13 //外设大小:256M
mcr p15,0,r0,c15,c2,4 //把r0的值(包括了外设基地址+外设大小)告诉cpu
// 关看门狗
ldr r0, =0x7E004000
mov r1, #0
str r1, [r0]
// 设置GPKCON0
ldr r1, =0x7F008800
ldr r0, =0x11110000
str r0, [r1]
mov r2, #0x1000
led_blink:
// 设置GPKDAT, 使GPK_4/5/6/7引脚输出低电平, LED亮
ldr r1, =0x7F008808
mov r0, #0
str r0, [r1]
// 延时
bl delay
// 设置GPKDAT, 使GPK_4/5/6/7引脚输出高电平, LED灭
ldr r1, =0x7F008808
mov r0, #0xf0
str r0, [r1]
// 延时
bl delay
sub r2, r2, #1
cmp r2, #0
bne led_blink
halt:
b halt
delay:
mov r0, #0x1000000
delay_loop:
cmp r0, #0
sub r0, r0, #1
bne delay_loop
mov pc, lr
```

② Makefile文件如下：

```
led.bin: start.o
arm-linux-ld -Ttext 0x50000000 -o led.elf $^
arm-linux-objcopy -O binary led.elf led.bin
arm-linux-objdump -D led.elf > led_elf.dis
%.o : %.S
arm-linux-gcc -o $@ $< -c

%.o : %.c
arm-linux-gcc -o $@ $< -c

clean:
rm *.o *.elf *.bin *.dis -rf
```

在Makefile所在目录下执行make命令时，系统会执行如下操作：

- 执行arm-Linux-gcc-o \$@ \$< -c命令，将当前目录下存在的汇编文件和C文件编译成.o文件；

- 执行`arm-Linux-ld -Ttext 0x50000000 -o led.elf $^`，将所有.o文件链接成elf文件，`-Ttext 0x50000000`表示程序的运行地址是0x50000000，即程序只有位于该地址上才能正常运行；
- 执行`arm-Linux-objcopy -O binary led.elf led.bin`，将elf文件抽取为可在开发板上运行的bin文件；
- 执行`arm-Linux-objdump -D led.elf > led_elf.dis`，将elf文件反汇编后保存在dis文件中，调试程序时可能会用到。

### 3、编译及下载运行程序。

#### ① 编译代码

确保当前用户为root用户（可使用`su root`命令切换到root用户）的条件下，在Fedora/CentOS的终端中执行如下命令：

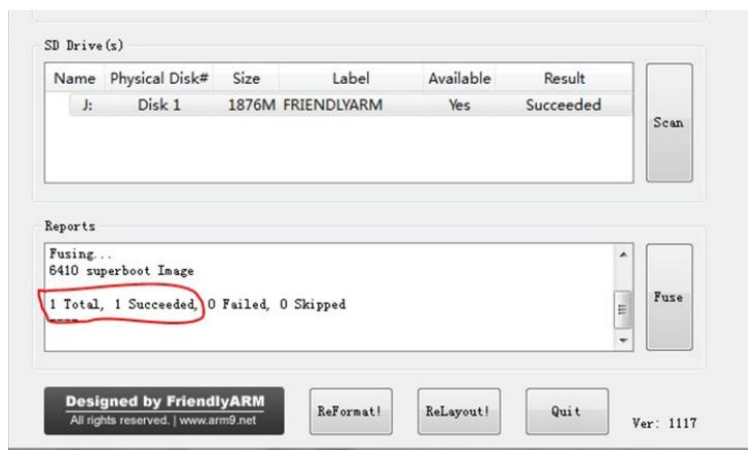
```
# cd leds
```

```
# make
```

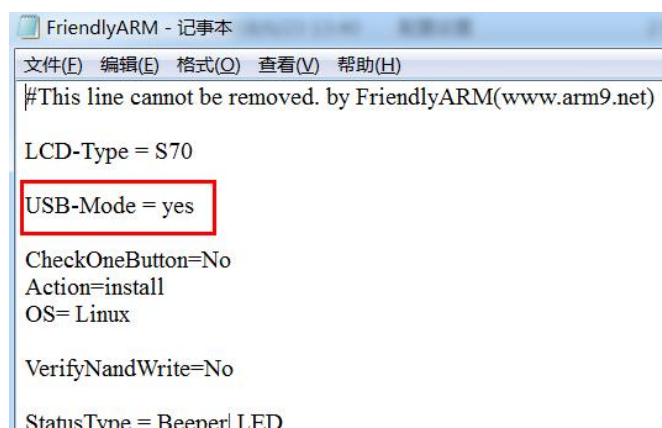
执行 `make` 后会生成 `led.bin` 文件。

② 下载（烧写）和运行程序（实验领取的SD卡已提前完成Superboot-6410.bin烧写和images文件夹复制，故可跳过这两步操作）

在Windows系统中，以管理员权限使用SD-Flasher程序，将引导程序Superboot-6410.bin烧入SD卡。成功烧写后的结果如下图所示：

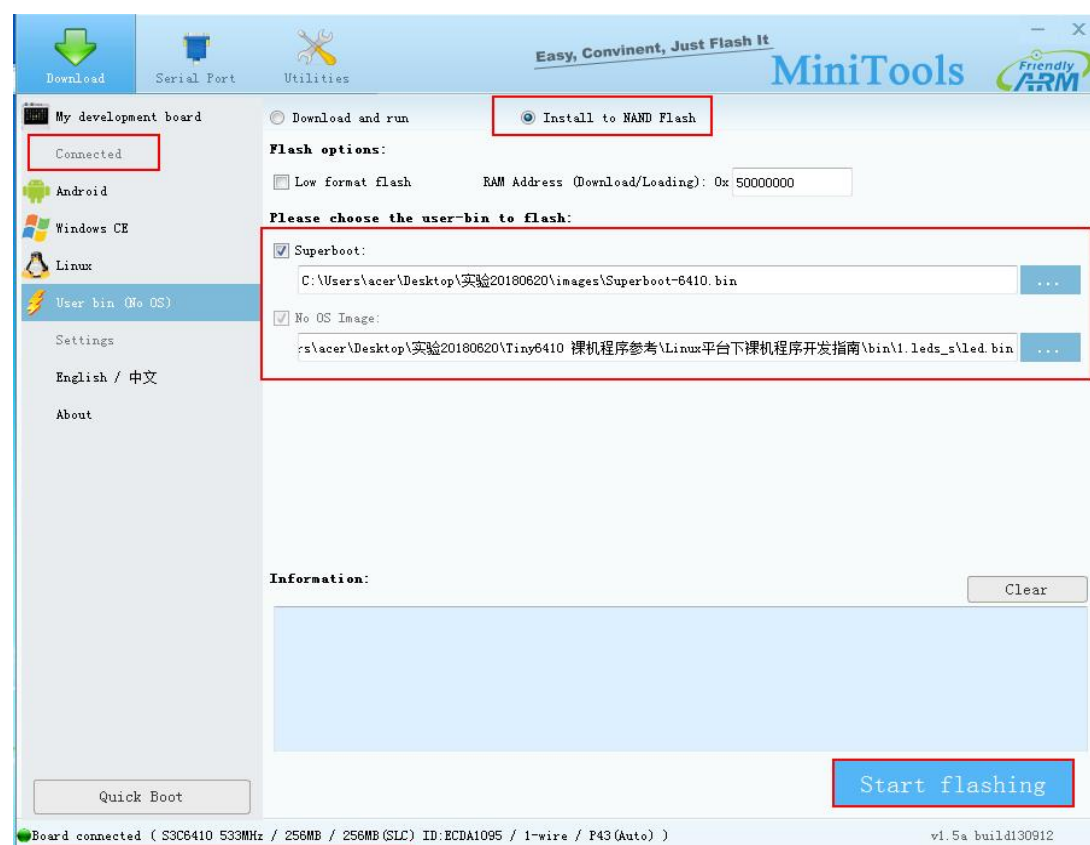


在SD卡中建立一个images文件夹，并把配置文件FriendlyARM.ini 复制到该文件夹中。双击打开SD卡中的该配置文件，在任意位置加入以下内容：USB-Mode = yes （注意字符的大小写）。如下图所示：



首先将烧写完成的SD卡，插入Tiny6410开发板的SD卡槽，并将开发板启动模式拨动开关S2拨动至“SDBOOT”位置，然后打开开发板电源。此时开发板将进入USB下载模式，LCD显示屏上显示“USB Mode: Waiting...”。接着用Mini USB线连接开发板与PC机，LCD上会显示“USB Mode: Connected”。

在Windows系统中，以管理员身份运行Mini Tools软件，打开如下软件界面：



确认界面下方显示绿色图标，表明Mini Tool s已通过Mini USB线与开发板成功连接。接着，选择Install to NAND Flash，表示将裸机程序烧入NAND Flash，但不需要运行。设置好下载地址“RAM Address(Download/Loading)=0x50000000”，同时勾选Superboot，并选择烧写时需要的引导程序Superboot-6410.bin和要烧写的裸机程序（需要提前从Linux系统复制到Windows系统中）。最后，点击【Start flashing】按钮，执行选择的某一裸机程序（bin文件）烧写。成功后显示如下信息：

```
Information:
Set User-Bin's Address Succeeded
Send File completed, Waitting...
Installing bootloader...
Send File completed, Waitting...
Downloading User-Bin succeed
Installing User-Bin succeed
All operations was completed successfully.
```

关闭开发板电源，首先将拨动开关S2拨动至“NAND”位置，然后重新开启开发板电源，则开发板运行刚刚烧入的裸机程序。

## 七、实验修改要求

在延时不变的条件，将原程序中“四个LED全亮，然后全灭”的显示状态，修改为“四个全亮，然后1和2亮（3和4灭），然后3和4亮（1和2灭），最后全灭”。



# 实验三 查询方式检测按键

## 一、实验目的

学会Linux系统中开发C程序的步骤和方法。在此基础上，掌握通过汇编程序实现Tiny6410初始化及通过C程序实现通用输入输出端口(GPIO)数据操作的方法。

## 二、实验内容

本次实验使用Fedora/CentOS操作系统环境, 安装ARM-Linux的开发库及编译器。学习在Linux下的编程和编译过程, 即创建一个新目录key\_led, 使用编辑器建立start.S、main.c和Makefile等文件, 并使用C语言编写查询处理程序。编译程序, 并下载文件到目标开发板上运行。

## 三、预备知识

- 1、清楚ARM微处理器芯片S3C6410的GPIO口硬件资源及相应寄存器结构。
- 2、有ARM汇编和C语言基础。
- 3、会使用Linux下常用的编辑器。
- 4、掌握Makefile的编写和使用。
- 5、了解Linux下的编译程序与交叉编译的过程。

## 四、实验设备及工具（包括软件调试工具）

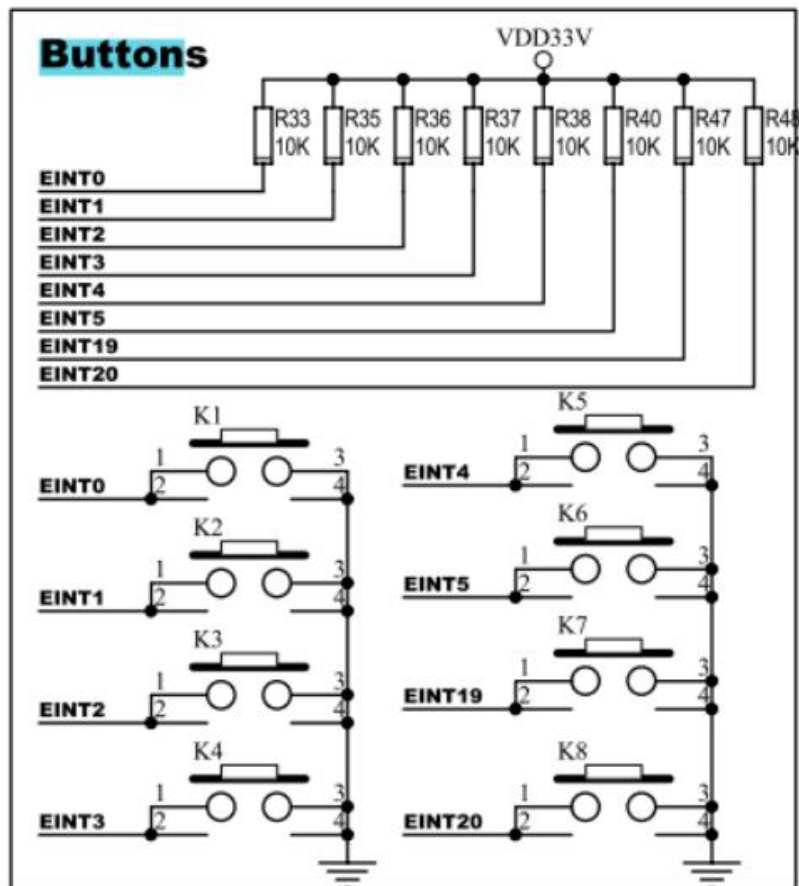
硬件：Tiny6410嵌入式实验平台。

软件：PC机操作系统Fedora/CentOS+Mini Tools+ARM-Linux开发环境

## 五、实验原理

### 1、按键硬件原理图

Tiny6410中共有8个用户按键（☆说明：部分型号的底板上，只有K1~K4四个按键），原理图如下：



按键原理图

相关的引脚信息如下图：

按键	K1	K2	K4	K4	K5	K6	K7	K8
对应的中断	EINT0	EINT1	EINT2	EINT3	EINT4	EINT5	EINT19	EINT20
可复用为GPIO	GPN0	GPN1	GPN2	GPN3	GPN4	GPN5	GPL11	GPL12

按键引脚图

## 2、查询按键原理

- 把外设的基地址告诉CPU。对于6410来说，内存的地址范围为0x00000000-0x60000000, 外设的地址范围为0x70000000-0x7fffffff。
- 关闭看门狗，防止程序不断重启；设置寄存器GPNCON，使接按键K1~K4的GPN0~GPN3为输入功能。设置寄存器GPKCON0，使接LED1~LED4的GPK4~GPK7为输出功能。
- 由原理图可知，按键按下时为低电平。读取寄存器GPNDATA，依次判断哪个按键按下，并点亮对应的LED灯。

## 六、实验步骤

1、建立工作目录key\_led。（若系统中已建立该目录，可跳过本步骤）

建立方法同实验二。

2、编写程序源代码。（若虚拟机中已有初始的源代码，可直接编辑）

在Linux下的文本编辑器有许多，常用的是vim和Xwindow界面下的gedit等，建议在实验中使用vim（需要学习vim的操作方法，请参考相关书籍中的关于vim的操作指南）。

① start.S的汇编源程序如下：

```
.global _start

_start:

    // 把外设的基地址告诉CPU
    ldr r0, =0x70000000
    orr r0, r0, #0x13
    mcr pl5,0,r0,c15,c2,4

    // 关看门狗
    ldr r0, =0x7E004000
    mov rl, #0
    str rl, [r0]

    // 设置栈
    ldr sp, =0x0C002000

    // 开启icaches
#ifdef CONFIG_SYS_ICACHE_OFF
    bic r0, r0, #0x00001000 @ clear bit 12 (I) I-cache
#else
    orr r0, r0, #0x00001000 @ set bit 12 (I) I-cache
#endif
    mcr pl5, 0, r0, c1, c0, 0

    // 调用c函数点灯
    bl main

halt:
    b halt
```

② main.c文件如下：

```

#define GPKCON0 (*(volatile unsigned long *)0x7F008800)
#define GPKDAT  (*(volatile unsigned long *)0x7F008808)

#define GPNCON  (*(volatile unsigned long *)0x7F008830)
#define GPNDAT  (*(volatile unsigned long *)0x7F008834)

void main(void)
{
    int dat = 0;

    // 配置GPK4-7为输出功能
    GPKCON0 = 0x11110000;

    // 所有LED熄灭
    GPKDAT = 0x000000f0;

    // 配置GPN为输入功能
    GPNCON = 0;

    // 轮询的方式查询按键事件
    while(1)
    {
        dat = GPNDAT;

        if(dat & (1<<0))          // KEY1被按下, 则LED1亮, 否则LED1灭
            GPKDAT |= 1<<4;
        else
            GPKDAT &= ~(1<<4);

        if(dat & (1<<1))          // KEY2被按下, 则LED2亮, 否则LED2灭
            GPKDAT |= 1<<5;
        else
            GPKDAT &= ~(1<<5);

        if(dat & (1<<2))          // KEY3被按下, 则LED3亮, 否则LED3灭
            GPKDAT |= (1<<6);
        else
            GPKDAT &= ~(1<<6);

        if(dat & (1<<3))          // KEY4被按下, 则LED4亮, 否则LED4灭
            GPKDAT |= 1<<7;
        else
            GPKDAT &= ~(1<<7);
    }
}

```

③ Makefile文件如下:

```

key.bin: start.o main.o
arm-linux-ld -Ttext 0x50000000 -o key.elf $^
arm-linux-objcopy -O binary key.elf key.bin
arm-linux-objdump -D key.elf > key_elf.dis
%.o : %.S
arm-linux-gcc -o $@ $< -c

%.o : %.c
arm-linux-gcc -o $@ $< -c

clean:
rm *.o *.elf *.bin *.dis -rf

```

### 3、编译及下载运行程序。

#### ① 编译代码

确保当前用户为root用户（可使用su root命令切换到root用户）的条件下，在Fedora/CentOS的终端中执行如下命令：

```
# cd key_led
```

```
# make
```

执行make后会生成key.bin文件。

#### ② 下载（烧写）和运行程序

按实验二给出的方法下载程序后，关闭开发板电源，首先**将启动模式拨动开关S2拨动至“ NAND” 位置**，然后重新开启开发板电源。当未按下任何按键时，四个LED灯全灭，当按下KEY1~KEY4任意一个按键时，则对应的LED被点亮。

### 七、实验修改要求

**当K1按下时，LED1和LED2被点亮；当K2按下时，LED3和LED4被点亮；当K3按下时，LED1和LED3被点亮；当K4按下时，LED2和LED4被点亮。**

# 实验四 按键中断实验

## 一、实验目的

学会Linux系统中开发C程序的步骤和方法。在此基础上，掌握通过汇编程序实现Tiny6410中断控制器初始化及通过C程序实现中断处理的方法。

## 二、实验内容

本次实验使用Fedora/CentOS操作系统环境, 安装ARM-Linux的开发库及编译器。学习在Linux下的编程和编译过程, 即创建一个新目录irq, 使用编辑器建立start.S、main.c、irq.c和Makefile等文件, 并使用C语言编写中断处理程序。编译程序, 并下载文件到目标开发板上运行。

## 三、预备知识

- 1、清楚ARM微处理器芯片S3C6410的外部中断口硬件资源及相应寄存器结构。
- 2、有ARM汇编和C语言基础。
- 3、会使用Linux下常用的编辑器。
- 4、掌握Makefile的编写和使用。
- 5、了解Linux下的编译程序与交叉编译的过程。

## 四、实验设备及工具（包括软件调试工具）

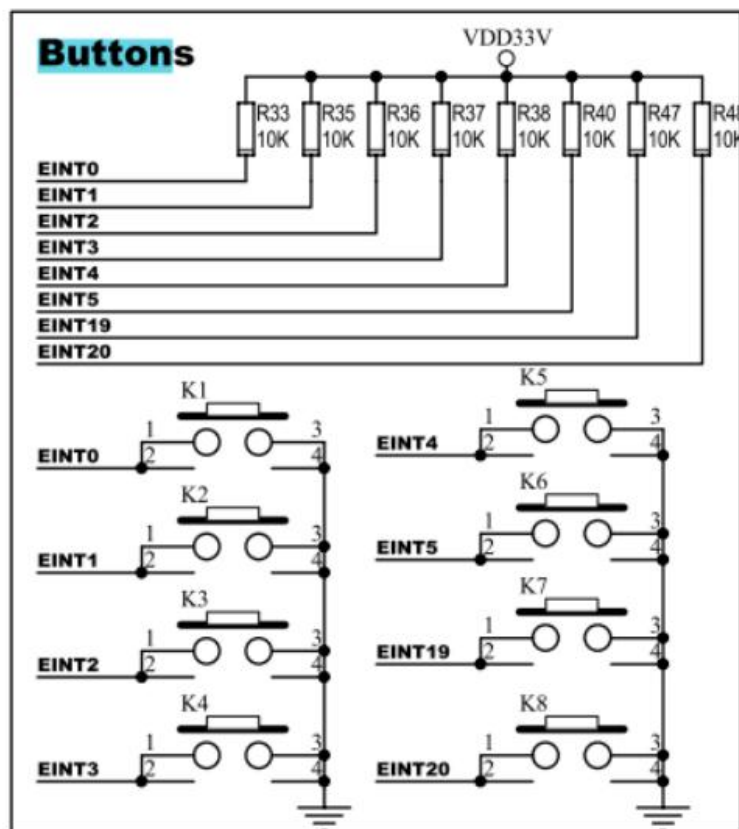
硬件：Tiny6410嵌入式实验平台。

软件：PC机操作系统Fedora/CentOS+Mini Tools+ARM-Linux开发环境

## 五、实验原理

### 1、按键硬件原理图

Tiny6410中共有8个用户按键（☆说明：部分型号的底板上，只有K1~K4四个按键），原理图如下：



按键原理图

相关的引脚信息如下图：

按键	K1	K2	K4	K4	K5	K6	K7	K8
对应的中断	EINT0	EINT1	EINT2	EINT3	EINT4	EINT5	EINT19	EINT20
可复用为 GPIO	GPN0	GPN1	GPN2	GPN3	GPN4	GPN5	GPL11	GPL12

按键引脚图

## 2、中断控制原理

- 把外设的基地址告诉CPU。对于6410来说，内存的地址范围为0x00000000-0x60000000, 外设的地址范围为0x70000000-0x7fffffff。
- 关闭看门狗，防止程序不断重启；设置寄存器GPNCON，使接按键K0~K5的GPN0~GPN5为中断功能；设置寄存器EINTOCON0，使按键的中断触发信号采用下降沿触发；设置寄存器EINTOMASK，使按键对应的中断使能；设置VIC0INTSELECT，使对应中断源工作于IRQ模式。
- 在中断处理函数中，查询寄存器EINTOPEND，判断哪个中断发生，进而识别对应的按键。最后，向寄存器EINTOPEND对应位写“1”，以实现清除中断标记。

## 六、实验步骤

1、建立工作目录irq。（若系统中已建立该目录，可跳过本步骤）

建立方法同实验二。

2、编写程序源代码。（若虚拟机中已有初始的源代码，可直接编辑）

在Linux下的文本编辑器有许多，常用的是vim和Xwindow界面下的gedit等，建议在实验中使用vim（需要学习vim的操作方法，请参考相关书籍中的关于vim的操作指南）。

① start.S的汇编源程序如下：

```
.global _start
.global asm_k1_irq
.extern do_irq
_start:

reset:
    // 外设地址告诉cpu
    ldr r0, =0x70000000
    orr r0, r0, #0x13
    mcr p15,0,r0,c15,c2,4

    // 关看门狗
    ldr r0, =0x7E004000
    mov r1, #0
    str r1, [r0]

    // 使能VIC
    mrc p15,0,r0,c1,c0,0
    orr r0,r0,#(1<<24)
    mcr p15,0,r0,c1,c0,0

    // 设置栈
    ldr sp, =8*1024

    // 初始化时钟
    bl clock_init

    // 初始化ddr
    bl sdram_init

    // 初始化nandflash
    bl nand_init

    // 初始化irq
    bl irq_init

    // 开中断
    //mov r0, #0x53
    //msr CPSR_cxsf, r0
    mrs r0,cpsr
    bic r0,r0,#0x80
    msr cpsr_c,r0
```



```

// 重定位, 把程序的代码段、数据段复制到它的链接地址去
adr r0, _start
ldr r1, =_start
ldr r2, =_bss_start
sub r2, r2, r1
cmp r1, r1
beq clean_bss
bl copy2ddr
cmp r0, #0
bne halt

// 清BSS, 把BSS段对应的内存清零
clean_bss:
ldr r0, =bss_start
ldr r1, =bss_end
mov r3, #0
cmp r0, r1
beq on_ddr
clean_loop:
str r3, [r0], #4
cmp r0, r1
bne clean_loop

on_ddr:
// 跳转
ldr pc, =main

// 中断异常
asm_k1_irq:
.word irq
irq:
/* 保存现场 */
ldr sp, =0x54000000
sub lr, lr, #4
stmfd sp!, {r0-r12, lr}
/* 处理异常 */
bl do_irq
/* 恢复现场 */
ldmfd sp!, {r0-r12, pc}^ /* ^表示把spsr恢复到cpsr */

halt:
b halt

```

② irq.c文件如下:

```

#include "stdio.h"
#define GPKCON0 (*(volatile unsigned long *)0x7F008800)
#define GPKDATA (*(volatile unsigned long *)0x7F008808)
#define GPNCON (*(volatile unsigned long *)0x7F008830)
#define GPNDAT (*(volatile unsigned long *)0x7F008834)
#define EINT0CON0 (*(volatile unsigned long *)0x7F008900)
#define EINT0MASK (*(volatile unsigned long *)0x7F008920)
#define EINT0PEND (*(volatile unsigned long *)0x7F008924)
#define PRIORITY (*(volatile unsigned long *)0x7F008280)
#define SERVICE (*(volatile unsigned long *)0x7F008284)
#define SERVICEPEND (*(volatile unsigned long *)0x7F008288)
#define VIC0IRQSTATUS (*(volatile unsigned long *)0x71200000)
#define VIC0FIQSTATUS (*(volatile unsigned long *)0x71200004)
#define VIC0RAWINTR (*(volatile unsigned long *)0x71200008)
#define VIC0INTSELECT (*(volatile unsigned long *)0x7120000c)
#define VIC0INTENABLE (*(volatile unsigned long *)0x71200010)
#define VIC0INTENCLEAR (*(volatile unsigned long *)0x71200014)
#define VIC0PROTECTION (*(volatile unsigned long *)0x71200020)
#define VIC0SWPRIORITYMASK (*(volatile unsigned long *)0x71200024)
#define VIC0PRIORITYDAISY (*(volatile unsigned long *)0x71200028)

#define VIC0ADDRESS (*(volatile unsigned long *)0x71200f00)

typedef void (isr) (void);
extern void asm_k1_irq();

```

```

void irq_init(void)
{
    /* 配置GPN0~5引脚为中断功能 */
    GPNCON &= ~(0xff);
    GPNCON |= 0xaa;

    /* 设置中断触发方式为：下降沿触发 */
    EINT0CON0 &= ~(0xff);
    EINT0CON0 |= 0x33;

    /* 禁止屏蔽中断 */
    EINT0MASK &= ~(0x0f);

    // Select INT_EINT0 mode as irq
    VIC0INTSELECT = 0;

    /* 在中断控制器里使能这些中断 */
    VIC0INTENABLE |= (0x3); /* bit0: eint0~3, bit1: eint4~11 */

    isr** isr_array = (isr**)(0x71200100);

    isr_array[0] = (isr*)asm_k1_irq;

    /*将GPK4-GPK7配置为输出口*/
    GPKCON0 = 0x11110000;

    /*熄灭四个LED灯*/
    GPKDATA = 0xf0;
}

void do_irq(void)
{
    int i = 0;
    //GPKDATA = 0x00;

    /* 分辨是哪个中断 */
    for (i = 0; i < 4; i++)
    {
        if (EINT0PEND & (1<<i))
        {
            GPKDATA &= ~(1<<(i+4));
        }
        else
        {
            GPKDATA |= 1<<(i+4);
        }
    }
    if (EINT0PEND & 0x8)
    {
        GPKDATA = 0xf0;
    }

    /* 清中断 */
    EINT0PEND = 0x3f;
    VIC0ADDRESS = 0;
}

```

③ main.c文件如下：

```
#include "stdio.h"

int main()
{
    while (1)
    {
        █
    }

    return 0;
}
```

④ Makefile文件如下：

```
CC      = arm-linux-gcc
LD       = arm-linux-ld
AR       = arm-linux-ar
OBJCOPY = arm-linux-objcopy
OBJDUMP  = arm-linux-objdump

INCLUDEDIR := $(shell pwd)/include
CFLAGS     := -Wall -Os -fno-builtin
CPPFLAGS   := -nostdinc -I$(INCLUDEDIR)

export CC AR LD OBJCOPY OBJDUMP INCLUDEDIR CFLAGS CPPFLAGS

objs := start.o sdram.o clock.o uart.o irq.o  main.o lib/libc.a nand.o

irq.bin: $(objs)
    ${LD} -Tirq.lds -o irq.elf $^
    ${OBJCOPY} -O binary -S irq.elf $@
    ${OBJDUMP} -D irq.elf > irq.dis

.PHONY : lib/libc.a
lib/libc.a:
    cd lib; make; cd ..

%.o:%.c
    ${CC} $(CPPFLAGS) $(CFLAGS) -c -o $@ $<

%.o:%.S
    ${CC} $(CPPFLAGS) $(CFLAGS) -c -o $@ $<

clean:
    make clean -C lib
    rm -f *.bin *.elf *.dis *.o
```

3、编译及下载运行程序。

① 编译代码

确保当前用户为root用户（可使用su root命令切换到root用户）的条件下，在Fedora/CentOS的终端中执行如下命令：

```
# cd irq
```

```
# make
```

执行make后会生成irq.bin文件。

## ② 下载（烧写）和运行程序

按实验二给出的方法下载程序后，关闭开发板电源，首先将启动模式拨动开关S2拨动至“ NAND” 位置，然后重新开启开发板电源。当按下K1按键时，LED1点亮；按下K2按键时，LED2点亮；按下K3按键时，LED3点亮；按下K4按键时，所有LED熄灭。

## 七、实验修改要求

当K1按下时，LED1和LED2被点亮；当K2按下时，LED3和LED4被点亮；当K3按下时，LED1和LED3被点亮；当K4按下时，LED2和LED4被点亮。

# 实验五 PWM 定时器实验

## 一、实验目的

学会Linux系统中开发C程序的步骤和方法。在此基础上，掌握通过C程序实现Tiny6410定时器初始化及中断处理的方法。

## 二、实验内容

本次实验使用Fedora/CentOS操作系统环境, 安装ARM-Linux的开发库及编译器。学习在Linux下的编程和编译过程, 即创建一个新目录timer, 使用编辑器建立start.S、main.c、timer.c和Makefile等文件。编译程序, 并下载文件到目标开发板上运行。

## 三、预备知识

- 1、清楚ARM微处理器芯片S3C6410的定时器硬件资源及相应寄存器结构。
- 2、有ARM汇编和C语言基础。
- 3、会使用Linux下常用的编辑器。
- 4、掌握Makefile的编写和使用。
- 5、了解Linux下的编译程序与交叉编译的过程。

## 四、实验设备及工具（包括软件调试工具）

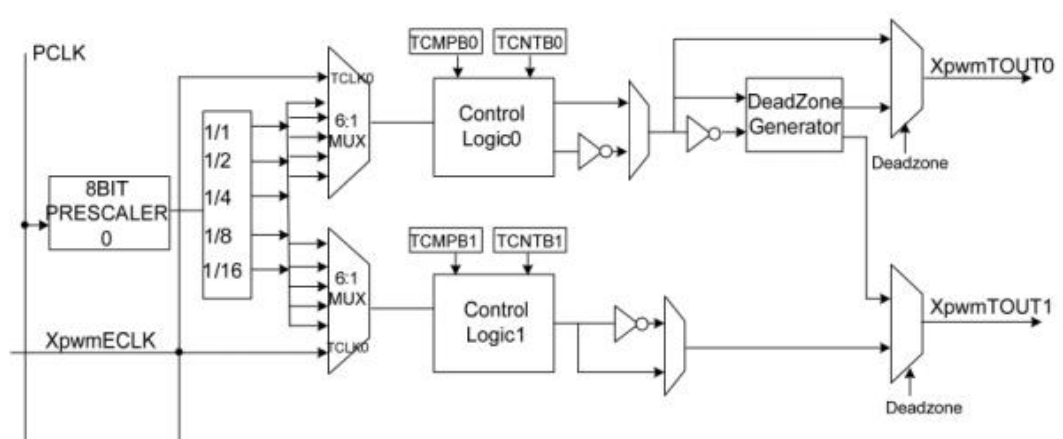
硬件：Tiny6410嵌入式实验平台。

软件：PC机操作系统Fedora/CentOS+Mini Tools+ARM-Linux开发环境

## 五、实验原理

### 1、PWM定时器

S3C6410内部包含五个32位定时器0~4。这些计时器通过产生内部定时中断来与ARM微处理器交互。



定时器0和1的内部结构图

## 2、中断控制原理

- 把外设的基地址告诉CPU。对于6410来说，内存的地址范围为0x00000000-0x60000000, 外设的地址范围为0x70000000-0x7fffffff。
- 关闭看门狗，防止程序不断重启；设置CPSR，允许IRQ中断；**设置中断控制器中的寄存器VICINTENABLE，允许定时器0中断**；设置寄存器TCFG0和TCFG1，写入相应的两次分频系数；设置寄存器TCNTB0和TCMPB0，写入计数初值和PWM比较值；设置寄存器TCON，手动更新计数值，自动重载计数初值，并启动定时器；**设置寄存器TINT\_CSTAT，允许定时器0中断**。
- 在中断处理函数中，执行相应的处理。最后，向寄存器TINT\_CSTAT对应位写“1”，以实现清除中断标记。

## 六、实验步骤

- 1、建立工作目录timer。（若系统中已建立该目录，可跳过本步骤）
- 2、编写程序源代码。（若虚拟机中已有初始的源代码，可直接编辑）

在Linux下的文本编辑器有许多，常用的是vim和Xwindow界面下的gedit等，建议在实验中使用vim（需要学习vim的操作方法，请参考相关书籍中的关于vim的操作指南）。

- ① start.S的汇编源程序如下：

```

.globl _start
.global asm_timer_irq
.extern do_irq
_start:

reset:
    // 外设地址告诉cpu
    ldr r0, =0x70000000
    orr r0, r0, #0x13
    mcr p15,0,r0,c15,c2,4

    // 关看门狗
    ldr r0, =0x7E004000
    mov r1, #0
    str r1, [r0]

    //Enabel VIC
    mrc p15,0,r0,c1,c0,0
    orr r0,r0,#(1<<24)
    mcr p15,0,r0,c1,c0,0

    // 设置栈
    ldr sp, =8*1024

    // 初始化时钟
    bl clock_init

    // 初始化ddr
    bl sdram_init

    // 初始化nandflash
    bl nand_init

    // 初始化中断
    bl irq_init

    // 开中断
    //mov r0, #0x53
    //msr CPSR_cxsf, r0
    mrs r0,cpsr

```

```

// 开中断
//mov r0, #0x53
//msr CPSR_cxsf, r0
mrs r0,cpsr
bic r0,r0,#0x80
msr cpsr_c,r0

// 重定位，把程序的代码段、数据段复制到它的链接地址去
adr r0, _start
ldr r1, =_start
ldr r2, =bss_start
sub r2, r2, r1
cmp r0,r1
beq clean_bss
bl copy2ddr
cmp r0, #0
bne halt

// 清BSS，把BSS段对应的内存清零
clean_bss:
ldr r0, =bss_start
ldr r1, =bss_end
mov r3, #0
cmp r0, r1
beq on_ddr
clean_loop:
str r3, [r0], #4
cmp r0, r1
bne clean_loop

on_ddr:
// 跳转
ldr pc, =main

// 中断异常
asm_timer_irq:
.word irq
irq:
/* 1. 保存现场 */
ldr sp, =0x54000000
sub lr, lr, #4
stmfd sp!, {r0-r12, lr} /* lr就是swi的下一条指令地址 */
/* 2. 处理异常 */
bl do_irq
/* 3. 恢复现场 */
ldmfd sp!, {r0-r12, pc}^ /* ^表示把spsr恢复到cpsr */

halt:
b halt

```



② timer.c文件如下:

```
#define          PWMTIMER_BASE          (0x7F006000)
#define          TCFG0                   ( *((volatile unsigned long *) (PWMTIMER_BASE+0x00)) )
#define          TCFG1                   ( *((volatile unsigned long *) (PWMTIMER_BASE+0x04)) )
#define          TCON                     ( *((volatile unsigned long *) (PWMTIMER_BASE+0x08)) )
#define          TCNTB0                   ( *((volatile unsigned long *) (PWMTIMER_BASE+0x0C)) )
#define          TCMPB0                   ( *((volatile unsigned long *) (PWMTIMER_BASE+0x10)) )
#define          TCNT00                   ( *((volatile unsigned long *) (PWMTIMER_BASE+0x14)) )
#define          TCNTB1                   ( *((volatile unsigned long *) (PWMTIMER_BASE+0x18)) )
#define          TCMPB1                   ( *((volatile unsigned long *) (PWMTIMER_BASE+0x1C)) )
#define          TCNT01                   ( *((volatile unsigned long *) (PWMTIMER_BASE+0x20)) )
#define          TCNTB2                   ( *((volatile unsigned long *) (PWMTIMER_BASE+0x24)) )
#define          TCMPB2                   ( *((volatile unsigned long *) (PWMTIMER_BASE+0x28)) )
#define          TCNT02                   ( *((volatile unsigned long *) (PWMTIMER_BASE+0x2C)) )
#define          TCNTB3                   ( *((volatile unsigned long *) (PWMTIMER_BASE+0x30)) )
#define          TCMPB3                   ( *((volatile unsigned long *) (PWMTIMER_BASE+0x34)) )
#define          TCNT03                   ( *((volatile unsigned long *) (PWMTIMER_BASE+0x38)) )
#define          TCNTB4                   ( *((volatile unsigned long *) (PWMTIMER_BASE+0x3C)) )
#define          TCNT04                   ( *((volatile unsigned long *) (PWMTIMER_BASE+0x40)) )
#define          TINT_CSTAT               ( *((volatile unsigned long *) (PWMTIMER_BASE+0x44)) )

typedef void (isr) (void);
extern void asm_timer_irq();

void irq_init(void)
{
    /* 在中断控制器里使能timer0中断 */
    VIC0INTENABLE |= (1<<23);

    VIC0INTSELECT =0;

    isr** isr_array = (isr**)(0x7120015C);

    isr_array[0] = (isr*)asm_timer_irq;

    /*将GPK4-GPK7配置为输出口*/
    GPKCON0 = 0x11110000;

    /*熄灭四个LED灯*/
    GPKDATA = 0xff;
}
// timer0中断的中断处理函数
void do_irq()
{
    unsigned long uTmp;
    GPKDATA = ~GPKDATA;
    //清timer0的中断状态寄存器
    uTmp = TINT_CSTAT;
    TINT_CSTAT = uTmp;
    VIC0ADDRESS=0x0;
}
```

```

// 初始化timer
void timer_init(unsigned long utimer,unsigned long uprescaler,unsigned long udivider,unsigned long utcntb,unsigned long utcmpb)
{
    unsigned long temp0;

    // 定时器的输入时钟 = PCLK / ( {prescaler value + 1} ) / {divider value} = PCLK/(65+1)/16=62500hz

    //设置预分频系数为66
    temp0 = TCFG0;
    temp0 = (temp0 & ~(0xff00ff)) | ((uprescaler-1)<<0);
    TCFG0 = temp0;

    // 16分频
    temp0 = TCFG1;
    temp0 = (temp0 & ~(0xf<<4*utimer)) & (~(1<<20)) | (udivider<<4*utimer);
    TCFG1 = temp0;

    // 1s = 62500hz
    TCNTB0 = utcntb;
    TCMFB0 = utcmpb;

    // 手动更新
    TCON |= 1<<1;

    // 清手动更新位
    TCON &= ~(1<<1);

    // 自动加载和启动timer0
    TCON |= (1<<0)|(1<<3);

    // 使能timer0中断
    temp0 = TINT_CSTAT;
    temp0 = (temp0 & ~(1<<utimer))|(1<<(utimer));
    TINT_CSTAT = temp0;
}

```

③ main.c文件如下：

```

#include "stdio.h"

void timer_init(unsigned long utimer,unsigned long uprescaler,unsigned long udivider,unsigned long utcntb,unsigned long utcmpb);

int main()
{
    timer_init(0,65,4,62500,0);

    while (1)
    {

    }

    return 0;
}

```

#### ④ Makefile文件如下：

```
CC      = arm-linux-gcc
LD      = arm-linux-ld
AR      = arm-linux-ar
OBJCOPY = arm-linux-objcopy
OBJDUMP = arm-linux-objdump

INCLUDEDIR := $(shell pwd)/include
CFLAGS     := -Wall -Os -fno-builtin
CPPFLAGS   := -nostdinc -I$(INCLUDEDIR)

export CC AR LD OBJCOPY OBJDUMP INCLUDEDIR CFLAGS CPPFLAGS

objs := start.o clock.o uart.o timer.o sdram.o main.o lib/libc.a nand.o

timer.bin: $(objs)
    ${LD} -Ttimer.lds -o timer.elf $^
    ${OBJCOPY} -O binary -S timer.elf $@
    ${OBJDUMP} -D timer.elf > timer.dis

.PHONY : lib/libc.a
lib/libc.a:
    cd lib; make; cd ..

%.o:%.c
    ${CC} ${CPPFLAGS} ${CFLAGS} -c -o $@ $<

%.o:%.S
    ${CC} ${CPPFLAGS} ${CFLAGS} -c -o $@ $<

clean:
    make clean -C lib
    rm -f *.bin *.elf *.dis *.o
```

### 3、编译及下载运行程序。

#### ① 编译代码

确保当前用户为root用户（可使用su root命令切换到root用户）的条件下，在Fedora/CentOS的终端中执行如下命令：

```
# cd timer
```

```
# make
```

执行make后会生成timer.bin文件。

#### ② 下载（烧写）和运行程序

按实验二给出的方法下载程序后，关闭开发板电源，首先将启动模式拨动开关S2拨动至“NAND”位置，然后重新开启开发板电源。初始状态，四个LED灯都为灭。每当定时器0的1秒定时达到后，四个LED切换状态（灭→亮/亮→灭）。

### 七、实验修改要求

将1秒定时到达后四个LED灯同时亮/灭，修改为四个LED灯按流水灯方式循环显示（LED1→LED2→LED3→LED4→LED1→……）。

# 实验六 串行口通信实验

## 一、实验目的

学会Linux系统中开发C程序的步骤和方法。在此基础上，掌握通过C程序实现Tiny6410串口数据接收与发送的方法。

## 二、实验内容

本次实验使用Fedora/CentOS操作系统环境, 安装ARM-Linux的开发库及编译器。学习在Linux下的编程和编译过程, 即创建一个新目录uart\_putchar, 使用编辑器建立start.S、main.c、uart.c和Makefile等文件。编译程序, 并下载文件到目标开发板上运行。通过串口线连接PC机与Tiny6410, 实现两者之间字符的传输。

## 三、预备知识

- 1、清楚ARM微处理器芯片S3C6410的串口硬件资源及相应寄存器结构。
- 2、有ARM汇编和C语言基础。
- 3、会使用Linux下常用的编辑器。
- 4、掌握Makefile的编写和使用。
- 5、了解Linux下的编译程序与交叉编译的过程。

## 四、实验设备及工具（包括软件调试工具）

硬件：Tiny6410嵌入式实验平台，串口线。

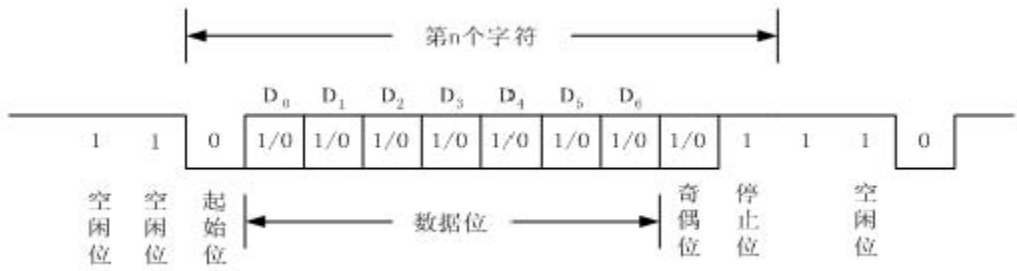
软件：PC机操作系统Fedora/CentOS+Mini Tools+ARM-Linux开发环境，串口调试助手。

## 五、实验原理

### 1、通用异步收发器UART

通用异步收发器（Universal Asynchronous Receiver and Transmitter，

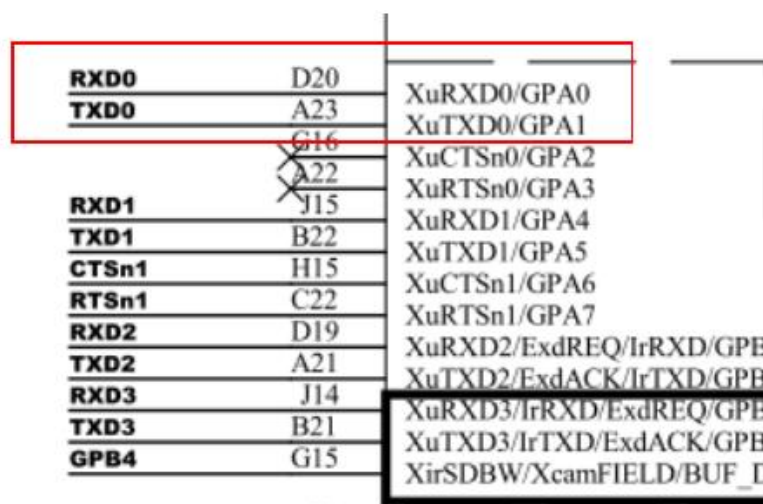
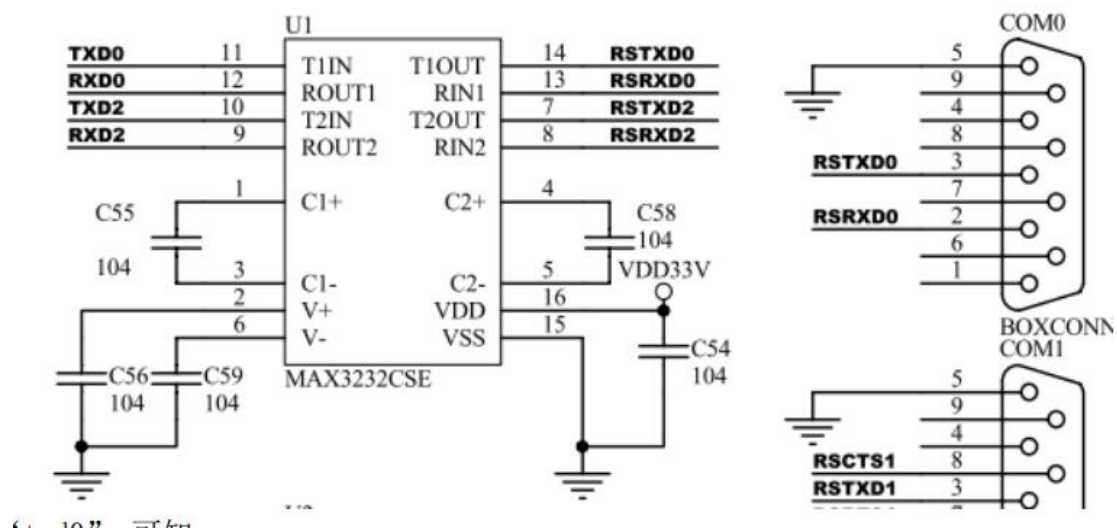
UART) 使用异步串行通信方式传输数据。异步串行通信方式是将传输数据的每个字符一位接一位(例如先低位、后高位)地传送。数据的各不同位可以分时使用同一传输通道, 因此串行通信可以减少信号连线, 最少用两根线即可进行。接收方对于同一根线上一连串的数字信号, 首先要分割成位, 再按位组成字符。为了恢复发送的信息, 双方必须协调工作。通信双方使用各自的时钟信号, 而且允许时钟频率有一定误差, 因此实现较容易。但是由于每个字符都要独立确定起始和结束(即每个字符都要重新同步), 字符和字符间还可能有长度不定的空闲时间, 因此效率较低。



串行通信字符格式

上图给出异步串行通信中一个字符的传送格式。开始前, 线路处于空闲状态, 送出连续“1”。传送开始时, 首先发一个“0”作为起始位, 然后出现在通信线上的是字符的二进制编码数据。每个字符的数据位长可以约定为5 位、6位、7 位或8 位, 一般采用ASCII 编码。后面是奇偶校验位, 根据约定, 用奇偶校验位将所传字符中为“1”的位数凑成奇数个或偶数个, 也可以约定不要奇偶校验, 这样就取消奇偶校验位。最后是表示停止位的“1”信号, 这个停止位可以约定持续1 位、1.5位或2 位的时间宽度。至此一个字符传送完毕, 线路又进入空闲, 持续为“1”。经过一段随机的时间后, 下一个字符开始传送才又发出起始位。每一个数据位的宽度等于传送波特率的倒数。异步串行通信中, 常用的波特率为50, 95, 110, 150, 300, 600, 1200, 2400, 4800, 9600, 115200 等。

S3C6410的UART提供了四个独立的异步串行通信端口。每个异步串行端口通过中断或者直接存储器存取(DMA)模式来操作, 以实现在CPU和UART之间传输数据。每个UART的通道包含了两个64字节收发FIFO存储器。



UART引脚连接图

## 2、串口收发原理

- 把外设的基地址告诉CPU。对于6410来说，内存的地址范围为0x00000000-0x60000000, 外设的地址范围为0x70000000-0x7fffffff。
- 关闭看门狗，防止程序不断重启；对于串口0，设置寄存器GPACON，使GPA0和GPA1工作于串行通信功能；设置寄存器ULCON0、UCON0、UFCON0和UMCON0，写入串行通信帧格式等相关参数（8个数据位，无奇偶校验，1个停止位，无流控等）；设置寄存器UBRDIV0和UDIVSL0T0，写入串行通信波特率。
- 对于串口0，查询寄存器UFSTAT0的低八位是否为0，以判断是否接收到数据，并从寄存器URXH0中获取收到的数据；查询寄存器UFSTAT0的D14位，以判断能否发送数据，并向寄存器UTXH0写入发送的数据。

## 六、实验步骤

- 1、建立工作目录uart\_putchar。（若系统中已建立该目录，可跳过本步骤）
- 2、编写程序源代码。（若虚拟机中已有初始的源代码，可直接编辑）

在Linux下的文本编辑器有许多，常用的是vim和Xwindow界面下的gedit等，建议在实验中使用vim（需要学习vim的操作方法，请参考相关书籍中的关于vim的操作指南）。

① start.S的汇编源程序如下：

```
.global _start

_start:

    // 把外设的基地址告诉CPU
    ldr r0, =0x70000000
    orr r0, r0, #0x13
    mcr p15,0,r0,c15,c2,4

    // 关看门狗
    ldr r0, =0x7E004000
    mov r1, #0
    str r1, [r0]

    // 设置栈
    ldr sp, =0x0C002000

    // 开启icaches
#ifdef CONFIG_SYS_ICACHE_OFF
    bic r0, r0, #0x00001000    @ clear bit 12 (I) I-cache
#else
    orr r0, r0, #0x00001000    @ set bit 12 (I) I-cache
#endif
    mcr p15, 0, r0, c1, c0, 0

    // 设置时钟
    bl clock_init

    // 调用c函数点灯
    bl main

halt:
    b halt
```



② uart.c文件如下:

```
#include "uart.h"

#define ULCON0      (*(volatile unsigned long *)0x7F005000)
#define UCON0       (*(volatile unsigned long *)0x7F005004)
#define UFCON0      (*(volatile unsigned long *)0x7F005008)
#define UMCN0       (*(volatile unsigned long *)0x7F00500C)
#define UTRSTAT0    (*(volatile unsigned long *)0x7F005010)
#define UFSTAT0     (*(volatile unsigned long *)0x7F005018)
#define UTXH0       (*(volatile unsigned char *)0x7F005020)
#define URXH0       (*(volatile unsigned char *)0x7F005024)
#define UBRDIV0     (*(volatile unsigned short *)0x7F005028)
#define UDIVSLOT0   (*(volatile unsigned short *)0x7F00502C)
#define GPACON      (*(volatile unsigned long *)0x7F008000)

void init_uart(void)
{
    /* 1. 配置引脚 */
    GPACON &= ~0xff;
    GPACON |= 0x22;

    /* 2. 设置数据格式等 */
    ULCON0 = 0x3;           // 数据位:8, 无校验, 停止位: 1, 8n1
    UCON0 = 0x5;            // 时钟: PCLK, 禁止中断, 使能UART发送、接收
    UFCON0 = 0x01;          // FIFO ENABLE
    UMCN0 = 0;              // 无流控

    /* 3. 设置波特率 */
    // DIV_VAL = (PCLK / (bps x 16) ) - 1 = (66500000/(115200x16))-1 = 35.08
    // DIV_VAL = 35.08 = UBRDIVn + (num of 1's in UDIVSLOTn)/16
    UBRDIV0 = 35;
    UDIVSLOT0 = 0x1;
}

/* 接收一个字符 */
char getchar(void)
{
    while ((UFSTAT0 & 0x7f) == 0); // 如果RX FIFO空, 等待
    return URXH0;                  // 取数据
}

/* 发送一个字符 */
void putchar(char c)
{
    while (UFSTAT0 & (1<<14)); // 如果TX FIFO满, 等待
    UTXH0 = c;                  // 写数据
}
```



③ main.c文件如下：

```
#include "uart.h"

int main()
{
    char c;

    init_uart();
    while (1)
    {
        c = getchar();
        putchar(c+1);
    }

    return 0;
}
```

④ Makefile文件如下：

```
uart.bin: start.o main.o uart.o clock.o
    arm-linux-ld -Ttext 0x50000000 -o uart.elf $^
    arm-linux-objcopy -O binary uart.elf uart.bin
    arm-linux-objdump -D uart.elf > uart.dis

%.o : %.S
    arm-linux-gcc -o $@ $< -c

%.o : %.c
    arm-linux-gcc -o $@ $< -c -fno-builtin

clean:
    rm *.o *.elf *.bin *.dis -rf
```

3、编译及下载运行程序。

① 编译代码

在Fedora/CentOS的终端中执行如下命令：

```
# cd uart_putchar
```

```
# make
```

执行make后会生成uart.bin文件。

② 下载（烧写）和运行程序

按实验二给出的方法下载程序后，关闭开发板电源，首先将启动模式拨动

开关S2拨动至“ NAND” 位置，然后用串口线连接PC机与Ti ny6410的COM0/COM3口，再重新开启开发板电源。接着运行PC机上的串口调试助手软件，将波特率设置为115200，其它串口通信参数保持默认，最后打开串口，在发送区输入“ abcd” 并点击发送。若串口通信正常，则接收区会显示“ bcde” 。

## 七、实验修改要求

如果Ti ny6410开发板收到的是大写字母，则将其转换为小写字母后发送回去；如果收到的是小写字母，则将其转换为大写字母后发送回去；如果收到的是数字，则把数字乘以2后发送回去。