

树莓派系列教程汇总

目录

目录.....	1
树莓派系列教程 1: 人生若只如初见	3
树莓派系列教程 2: 树莓派烧写镜像	5
树莓派系列教程 3: 访问树莓派	10
树莓派系列教程 4: 树莓派 raspi-config 配置	15
树莓派系列教程 5: linux 常用命令以及 vi/vim 编辑器	18
树莓派教程系列 6: 文件共享(samba).....	23
树莓派系列教程 7: 如何点亮一个 LED 灯(上)	25
树莓派系列教程 8 : 如何点亮一个 LED 灯(下)	31
wiringPi	31
bcm2835.....	34
python.....	35
树莓派系列教程 9: 按键	37
bcm2835.....	37
wiringPi	38
python.....	39
树莓派系列教程 10: I2C	42
启动 I2C	42
i2c-tools	42

使用 i2c-tools 控制 PCF8574 IO.....	44
树莓派系列教程 11: I2C 编程	45
bcm2835.....	45
wiringPi	46
python.....	47
树莓派系列教程 12: I2C 总线控制 BMP180	50
树莓派系列教程 13: Serial 串口.....	56
释放串口	56
使用 minicom 调试串口	56
串口编程	58
python.....	58
树莓派系列教程 14: 单总线控制 DS18B20.....	60
树莓派系列教程 15: 红外遥控.....	67
树莓派系列教程 16: RTC	73
树莓派系列教程 17: PCF8591 AD/DA	78
DAC	78
ADC	80
树莓派系列教程 18: SPI	83

树莓派系列教程 1：人生若只如初见

Raspberry Pi（中文名为“树莓派”，简称为 RPi，或者 RasPi/RPi）是为学生计算机编程教育而设计，只有信用卡 大小的卡片式电脑，其系统基于 Linux。

第一次接触树莓派的时候也是刚学 linux 系统。抱着玩玩的心态买了一块树莓派 B+，刚拿到手的是有点和想象的不一样，就只有一个盒子装在，真的只有一张信用卡大小，真的很吃惊。自己动手刷了一个系统启动起来的时候真的很神奇，这小小的身板真的是五脏俱全，一个便携的 mini pc. 然后就一发不可收拾，玩得不亦乐乎。SSH 登陆，xrdp 远程桌面，家庭影院，LAMP 服务器，SAMBA 服务器等等，它不但是一个卡式电脑，更重要的是它引出 40 个管脚。这就可以做更多的事情了。就让我们一起走进树莓派的世界.....

项目	A+型	B 型	B+型	2 代 B 型
SoC（系统级芯片）	Broadcom BCM2835（CPU，GPU DSP 和 SDRAM）			Broadcom BCM2836
CPU	ARM1176JZF-S 核心（ARM11 系列）700MHz 单核			ARM Cortex-A7 900MHz 4 核
GPU（图形处理器）	Broadcom VideoCore IV， OpenGL ES 2.0， 1080p 30 h.264/MPEG-4 AVC 高清解码器			
内存	256MB	512MB		1GB
USB 2.0	1（支持 USB hub 扩展）	2（支持 USB hub 扩展）	4（支持 USB hub 扩展）	
视频输出	RCA 视频接口输出(仅 1 代 B 型有此接口),支持 PAL 和 NTSC 制式,支持 HDMI（1.3 和 1.4），分辨率为 640 x 350 至 1920 x 1200 支持 PAL 和 NTSC 制式。			
音频输出	3.5mm 插孔， HDMI（高清晰度多音频/视频接口）			
SD 卡接口	Micro SD 卡接口	标准 SD 卡接口	Micro SD 卡接口	
网络介入	没有(需通过USB)	10/100 以太网接口（RJ45 接口）		
扩展接口	40	26	40	
额定功率	未知，但更低	700 毫安（为 3.5 W）	600 毫安（为 3.0 W）	1000 毫安（为 5.0 W）
电源输入	5v，通过 MicroUSB 或 GPIO 引脚			
总体尺寸	65 × 56 mm	85.60 × 53.98 mm	85 × 56 × 17 mm	
操作系统	Debian GNU/Linux、Fedora、Arch Linux、RISC OS 2 代 B 型还支持 Windows10 和 Snappy Ubuntu Core，官方会持续更新以支持更多操作系统，敬请期待！			



A+型



B型



B+型



2代B型

树莓派系列教程 2：树莓派烧写镜像

树莓派没有硬盘，取而代之的是 TF 卡。只需一个电源一张 TF 卡即可启动树莓派。

1) 下载树莓派系统

树莓派官网下载地址：<http://www.raspberrypi.org/downloads>

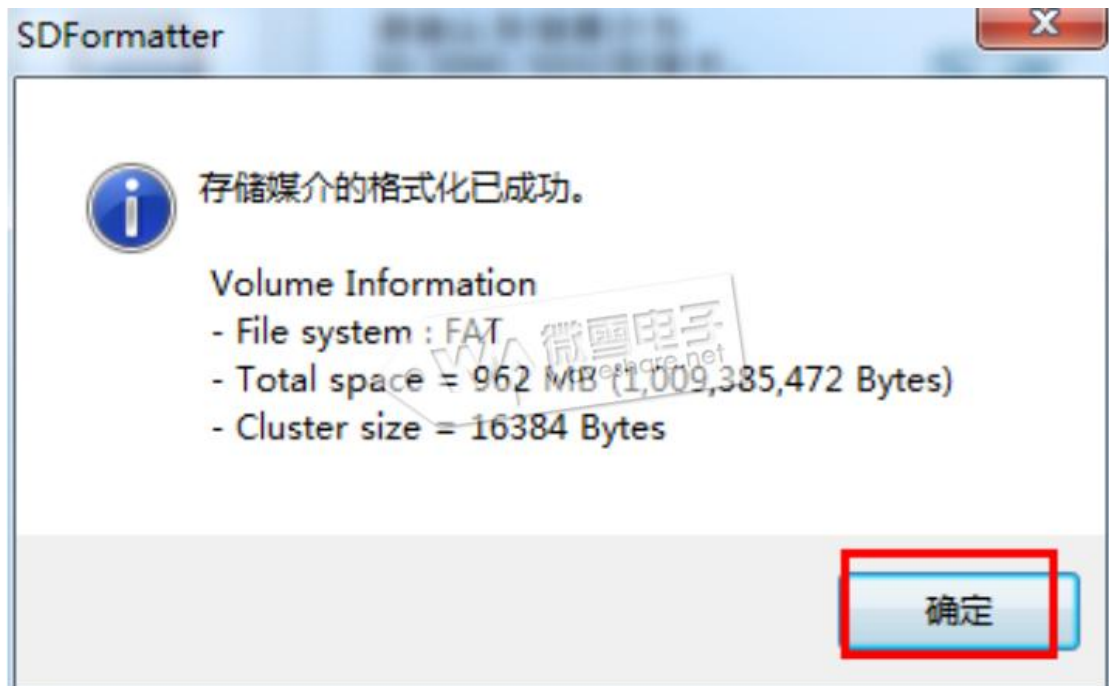
（可下载最新的 Raspbian 树莓派系统也可以在我们公司 wiki 上面下载我们配置过的系统）

2) 格式化 SD 卡

插上 SD 卡到电脑，使用 SDFormatter.exe 软件格式化 SD 卡。

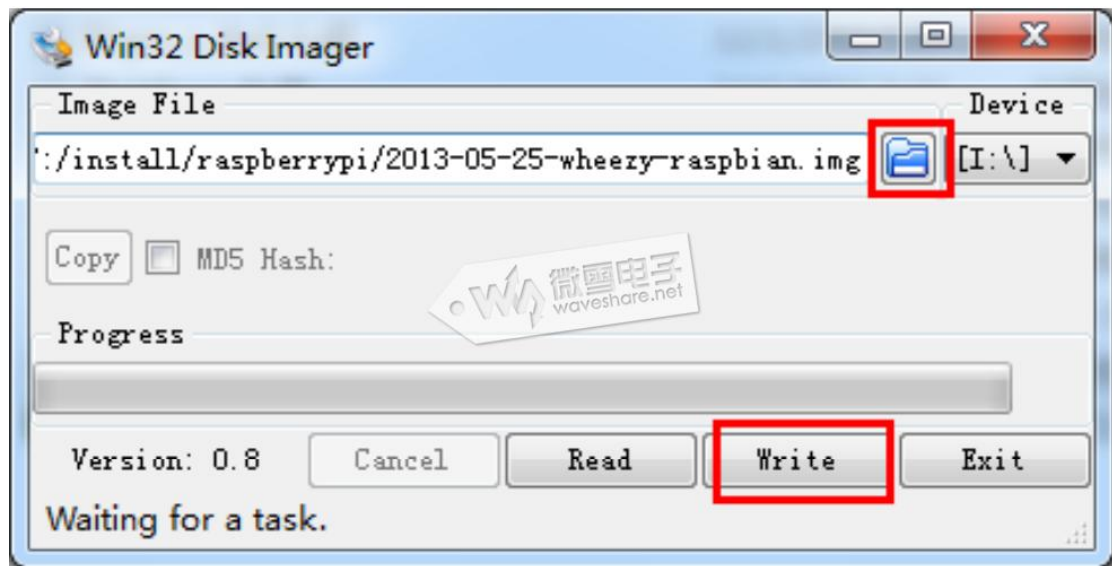






3) 烧写树莓系统

用 Win32DiskImager.exe 烧写镜像。选择要烧写的镜像，点击“Write”进行烧写。



4) 启动树莓派

烧写完后把 SD 卡插入树莓派即可运行。树莓派 raspbian 系统 pi 用户密码默认为 raspberry；root 权限密码为 raspberry。

启动图形界面命令

```
startx
```

重启

```
sudo reboot
```

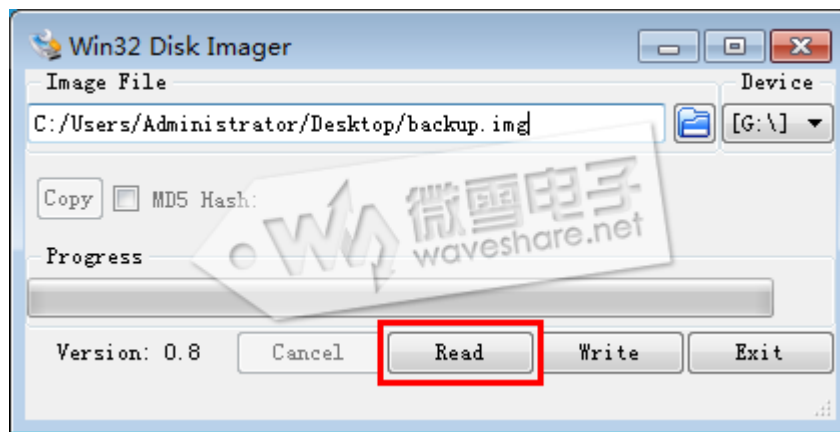
关机

```
sudo poweroff
```

5) 备份树莓派系统

有的时候想装 win10、ubuntu、kodi 等等系统玩玩，但是只有一张卡，又想保留现在的系统，那么现在教你一个方法，就是备份现在的系统。其实很简单，看到这个图像必大家都明白了。先新建一个空白的 .img 后缀的文件，然后选择直接 read 就可以备份系统了，到

时再重装就可以恢复了。



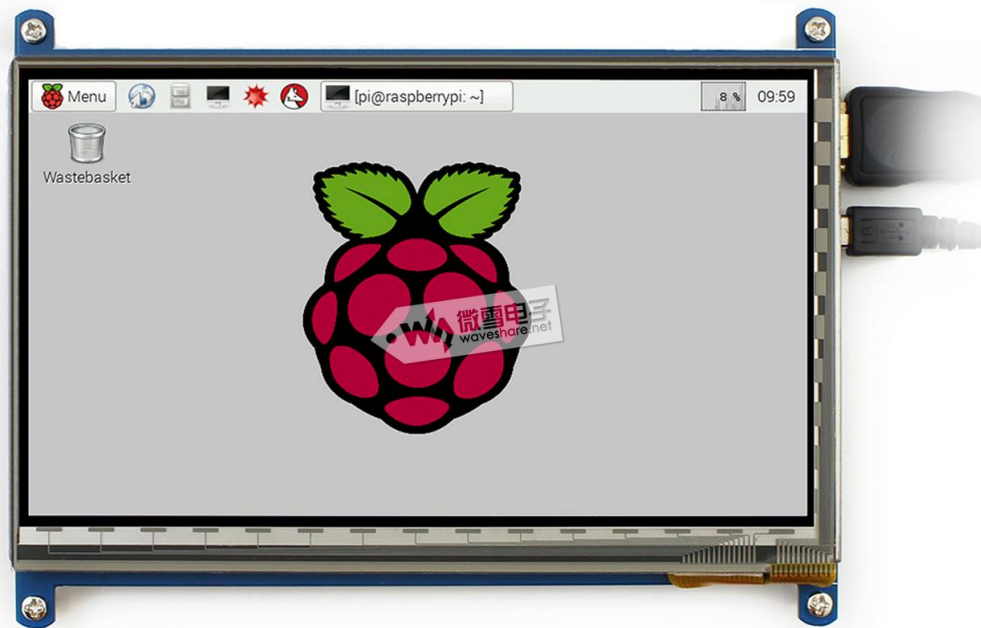
树莓派系列教程 3：访问树莓派

1) 外接 HDMI 显示器，鼠标键盘

如果把树莓派当作一个小电脑，那么可以外接鼠标键盘操作树莓派，通过 HDMI 连接显示器显示界面。

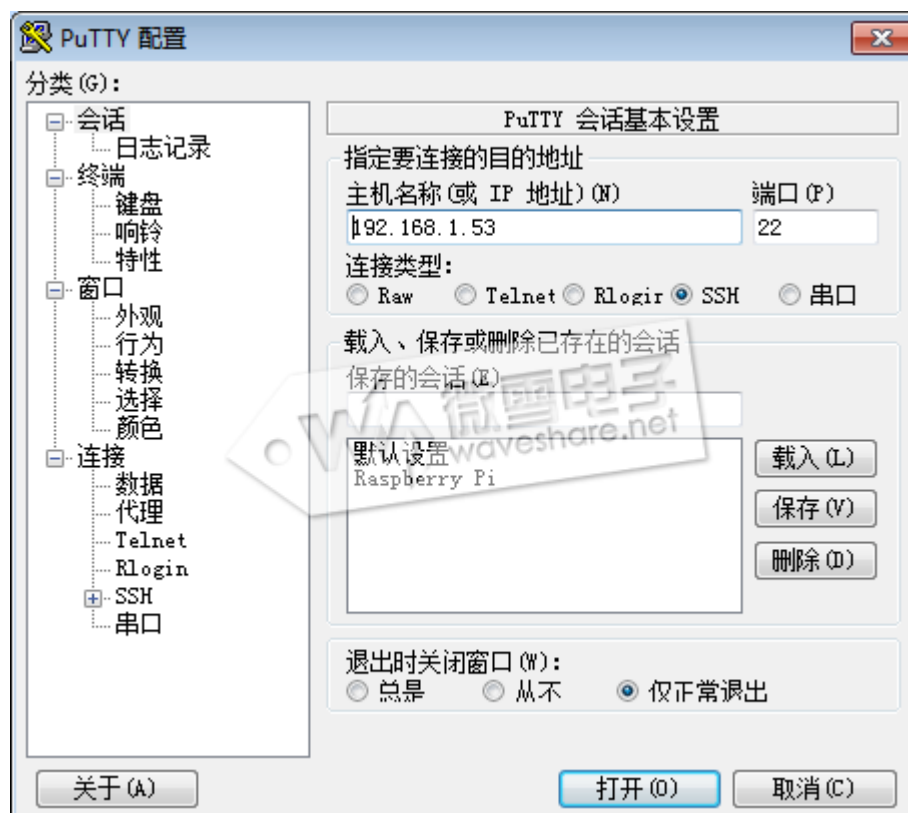
2) 外接 LCD 显示屏

如果你觉得抱着鼠标键盘麻烦，那么你可以接 LCD 触摸屏，触摸控制树莓派，像平板一样玩

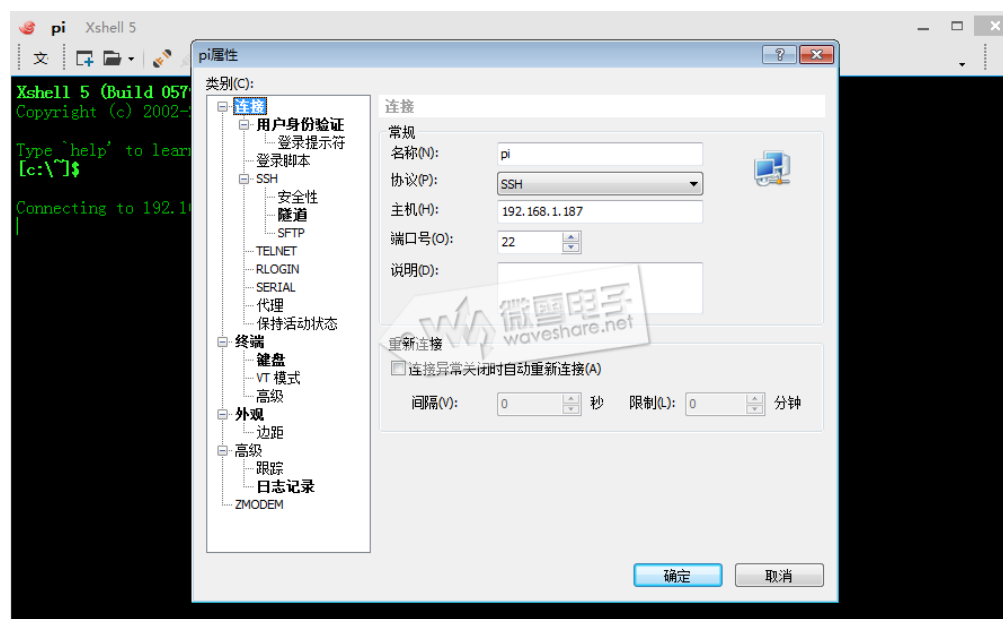


3) SSH

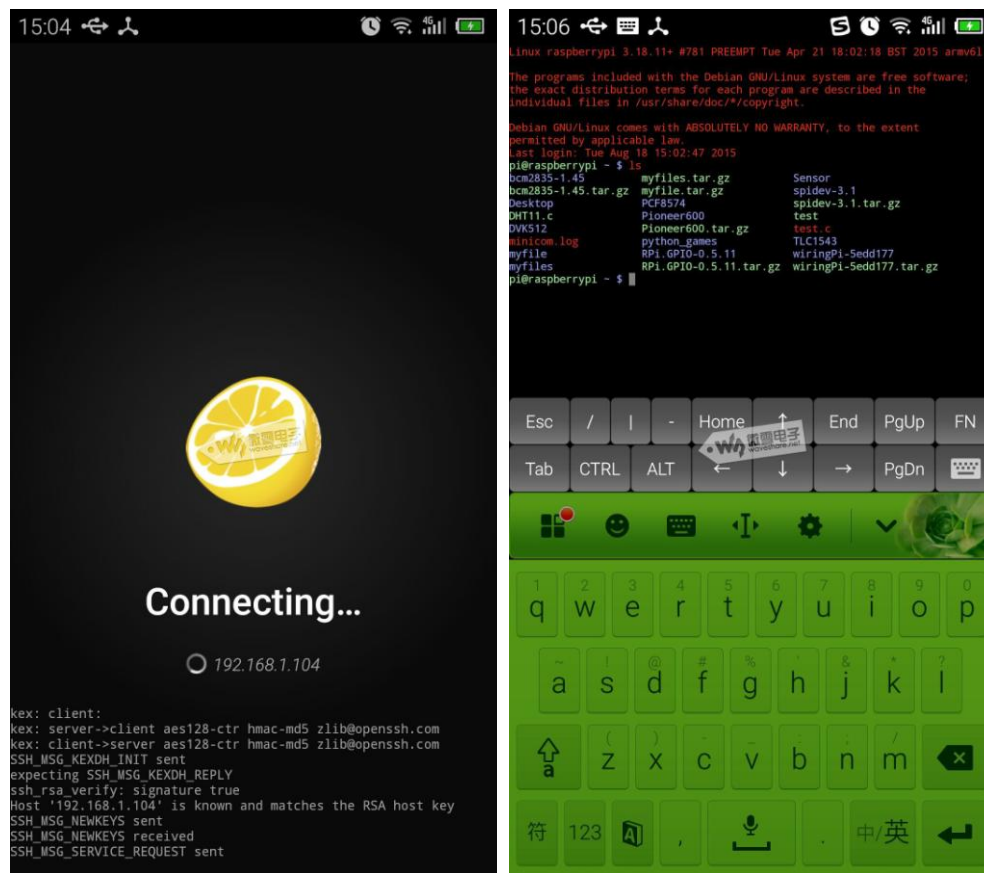
更常用的方式是通过 ssh 控制树莓派，putty, SecureCRT, SSH Secure Shell Client 等 SSH 软件



我个人更加喜欢用 xshell 软件



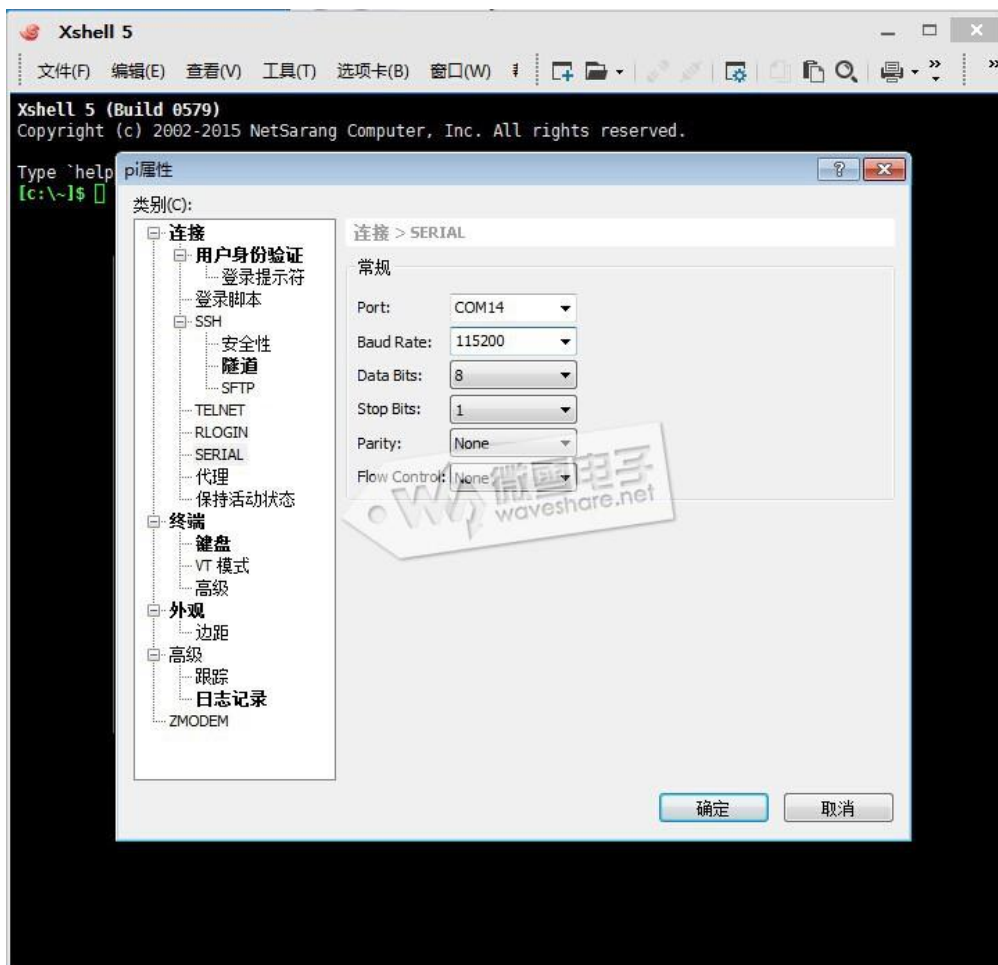
当然你也可以用手机安装 juiceSSH 这个软件通过 SSH 控制树莓派。



4) 串口终端

如果树莓派没有连接网络，又没有显示设备。你可以用串口登陆树莓派。你需要串口转 USB 模块将树莓派连接到电脑。本公司的 Pioneer 600 扩展板板载 UART TO USB 功能。通过

putty,xshell 等软件通过串口登陆树莓派。

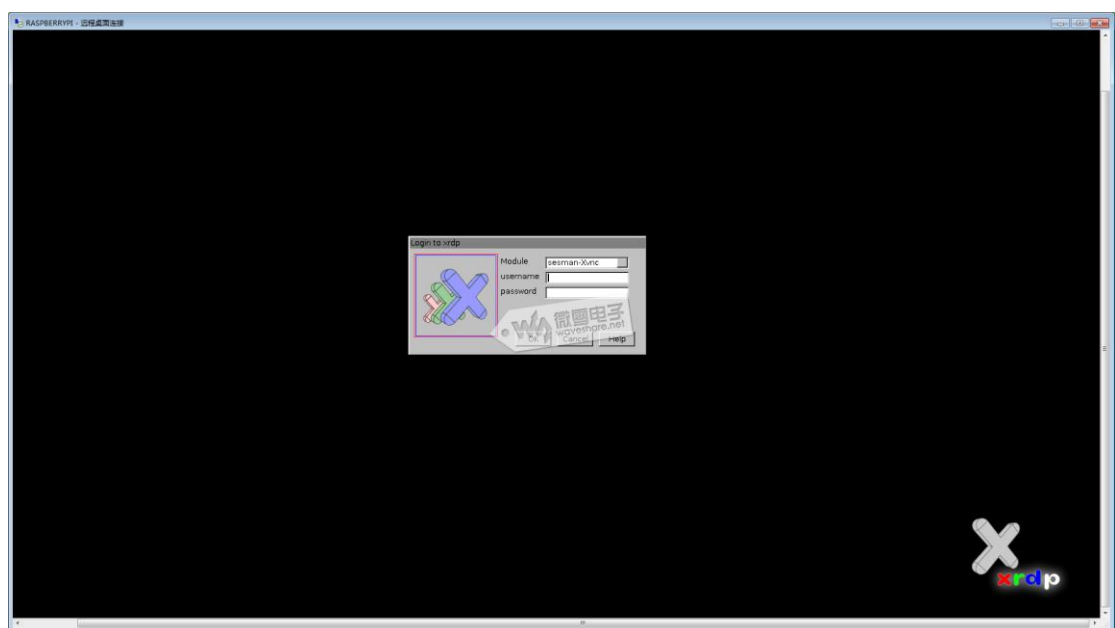


5) 通过远程桌面连接树莓派。

在树莓派命令行下输入以下命令安装 xrdp

```
sudo apt-get install xrdp
```

在 windows 附件中打开远程桌面连接树莓派 IP,输入用户名密码,就可以看到树莓派的界面了。

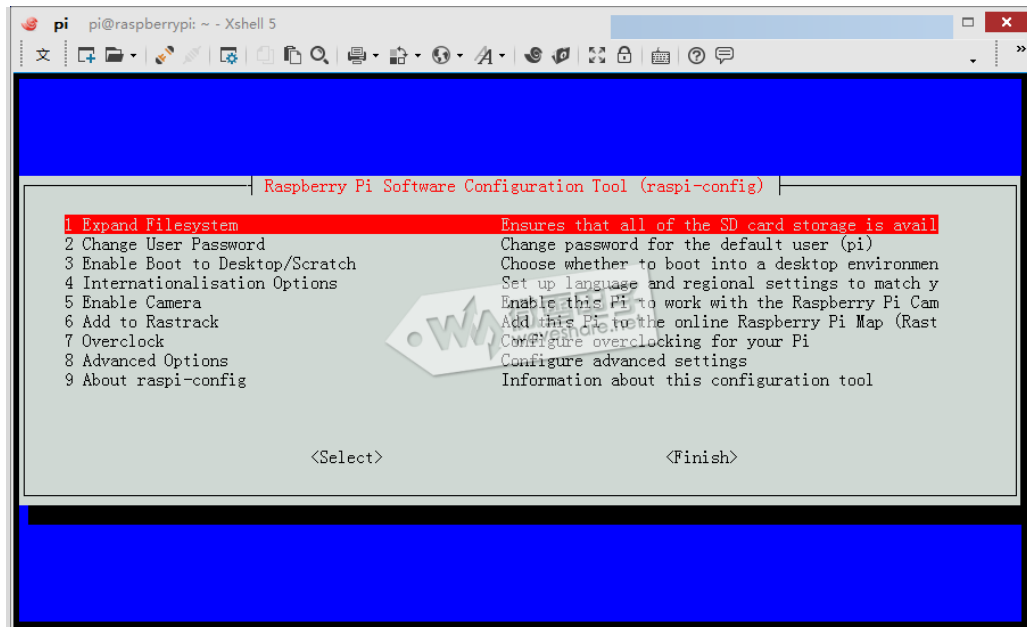


树莓派系列教程 4：树莓派 raspi-config 配置

1) 树莓派 raspi-config 设置

第一次使用树莓派的时候需要进行一些简单的配置，在终端运行以下命令进入配置界面

```
sudo raspi-config
```



1 Expand Filesystem 扩展文件系统,扩展整张 SD 卡空间作为根分区。

2 Change User Password 改变默认 pi 用户的密码，按回车后输入 pi 用户的新密码。

3 Enable Boot to Desktop/Scratch 启动时进入的环境选择

| 1 Console Text console, requiring login(default) 启动时进入字符控制台，需要进行登录（默认项）。

| 2 Desktop log in as user 'pi' at the graphical desktop 启动时进入 LXDE 图形界面的桌面。

└ 3 Scratch Start the Scratch programming environment upon boot 启动时进入 Scratch 编程环境。

4 Internationalisation Options 国际化选项，可以更改默认语言

| I1 Change Locale 语言和区域设置。选中 zh-cn 然后回车

| I2 Change Timezone 设置时区，如果不进行设置，Pi 的时间就显示不正常。

└ I3 Change Keyboard Layout 改变键盘布局

5 Enable Camera 启动 PI 的摄像头模块，如果想启用，选择 Enable，禁用选择 Disable 就行了

6 Add to Rastrack 把你的 PI 的地理位置添加到一个全世界开启此选项的地图。

7 Overclock

- |None 不超频, 运行在 700Mhz, 核心频率 250Mhz, 内存频率 400Mhz, 不增加电压
- |Modest 适度超频, 运行在 800Mhz, 核心频率 250Mhz, 内存频率 400Mhz, 不增加电压
- |Medium 中度超频, 运行在 900Mhz, 核心频率 250Mhz, 内存频率 450Mhz, 增加电压 2
- |High 高度超频, 运行在 950Mhz, 核心频率 250Mhz, 内存频率 450Mhz, 增加电压 6
- |Turbo 终极超频, 运行在 1000Mhz, 核心频率 500Mhz, 内存频率 600Mhz, 增加电压 6

8 Advanced Options 高级设置

- |A1 Overscan 是否让屏幕内容全屏显示
- |A2 Hostname 在网上邻居或者路由器能看到的主机名称
- |A3 Memory Split 内存分配, 选择给 GPU 多少内存
- |A4 SSH 是否运行 SSH 登录, 建议开户此选项, 以后操作 PI 方便, 有网络就行, 不用开屏幕了。
- |A5 Device Tree 时候默认启动设备树
- |A6 SPI 是否默认启动 SPI 内核驱动
- |A7 I2C 是否默认启动 I2C 内核驱动
- |A8 Serial 是否默认启动串口调试
- |A9 Audio 选择声音默认输出到模拟口还是 HDMI 口 i
 - | |0 Auto 自动选
 - | |1 Force 3.5mm ('headphone') jack 强制输出到 3.5mm 模拟口
 - | |2 Force HDMI 强制输出到 HDMI A0 update 把 raspi-config 这个工具自动升级到最新版本
- |9 About raspi-config 关于 raspi-config 的信息。

初次启动树莓派要设置 Internationalisation Options 选项

I1 Change Locale 设置语言, 默认为英文, 若想改中文, 须安装中文字体, 命令如下:

```
sudo apt-get update
```

```
sudo apt-get install ttf-wqy-zenhei ttf-wqy-microhei
```

移动到屏幕底部, 用空格键选中 zh-CN.GBK GBK 和 zh_CN.UTF-8 UTF-8 两项, 然后按回车, 然后默认语言选中 zh-CN.UTF-8, 然后回车。

安装拼音输入法

```
sudo apt-get install scim-pinyin
```

I2 change Timezone 设置时区, 选择 Asia (亚洲) 再选择 shanghai (上海)。

I3 Change Keyboard Layout 改变键盘布局

2) wifi 设置

有线什么的都 OUT 了, 无线才是王道, 有 wifi 的日子才是好日子。本人从来是都一个电源加一个无线网卡玩转树莓派。

运行以下命令查看网卡信息, 若有 wlan0 则已经识别无线网卡

```
ifconfig
```

打开配置文件并修改


```
sudo vi /etc/network/interfaces
```

注释掉 `wpa-conf /etc/wpa_supplicant/wpa_supplicant.conf`

添加下面两句

```
wpa-ssid waveshare_1013 #你要链接的 wifi ssid
```

```
wpa-psk waveshare #wpa 连接密码
```

若要设置静态 IP 地址则修改如下

重启网卡使设置生效

```
sudo service networking restart
```

树莓派系列教程 5: linux 常用命令以及 vi/vim 编辑器

1) linux 常用命令

树莓派系统多都是基于 Linux，而 Linux 多数情况下都是在命令行下输入命令操作。

Unix/Linux 命令参考

FOSSWire.com

文件命令	系统信息
ls - 列出目录	date - 显示当前日期和时间
ls -al - 使用格式化列出隐藏文件	cal - 显示当月的日历
cd dir - 更改目录到 dir	uptime - 显示系统从开机到现在所运行的时间
cd - 更改到 home 目录	w - 显示登录的用户
pwd - 显示当前目录	whoami - 查看你的当前用户名
mkdir dir - 创建目录 dir	finger user - 显示 user 的相关信息
rm file - 删除 file	uname -a - 显示内核信息
rm -r dir - 删除目录 dir	cat /proc/cpuinfo - 查看 cpu 信息
rm -f file - 强制删除 file	cat /proc/meminfo - 查看内存信息
rm -rf dir - 强制删除目录 dir *	man command - 显示 command 的说明手册
cp file1 file2 - 将 file1 复制到 file2	df - 显示磁盘占用情况
cp -r dir1 dir2 - 将 dir1 复制到 dir2; 如果 dir2 不存在则创建它	du - 显示目录空间占用情况
mv file1 file2 - 将 file1 重命名或移动到 file2; 如果 file2 是一个存在的目录则将 file1 移动到目录 file2 中	free - 显示内存及交换区占用情况
ln -s file link - 创建 file 的符号连接 link	
touch file - 创建 file	压缩
cat > file - 将标准输入添加到 file	tar cf file.tar files - 创建包含 files 的 tar 文件 file.tar
more file - 查看 file 的内容	tar xf file.tar - 从 file.tar 提取文件
head file - 查看 file 的前 10 行	tar czf file.tar.gz files - 使用 Gzip 压缩创建 tar 文件
tail file - 查看 file 的后 10 行	tar xzf file.tar.gz - 使用 Gzip 提取 tar 文件
tail -f file - 从后 10 行开始查看 file 的内容	tar cjf file.tar.bz2 - 使用 Bzip2 压缩创建 tar 文件
	tar xjf file.tar.bz2 - 使用 Bzip2 提取 tar 文件
进程管理	gzip file - 压缩 file 并重命名为 file.gz
ps - 显示当前的活动进程	gzip -d file.gz - 将 file.gz 解压缩为 file
top - 显示所有正在运行的进程	
kill pid - 杀掉进程 id pid	网络
killall proc - 杀掉所有名为 proc 的进程	ping host - ping host 并输出结果
bg - 列出已停止或后台的作业	whois domain - 获取 domain 的 whois 信息
fg - 将最近的作业带到前台	dig domain - 获取 domain 的 DNS 信息
fg n - 将作业 n 带到前台	dig -x host - 逆向查询 host
	wget file - 下载 file
文件权限	wget -c file - 断点续传
chmod octal file - 更改 file 的权限	
<ul style="list-style-type: none"> 4 - 读 (r) 2 - 写 (w) 1 - 执行 (x) 	安装
示例:	从源代码安装:
chmod 777 - 为所有用户添加读、写、执行权限	./configure
chmod 755 - 为所有者添加 rwx 权限, 为组和其他用户添加 rx 权限	make
更多选项参阅 man chmod.	make install
	dpkg -i pkg.deb - 安装包 (Debian)
	rpm -Uvh pkg.rpm - 安装包 (RPM)
SSH	
ssh user@host - 以 user 用户身份连接到 host	快捷键
ssh -p port user@host - 在端口 port 以 user 用户身份连接到 host	Ctrl+C - 停止当前命令
ssh-copy-id user@host - 将密钥添加到 host 以实现无密码登录	Ctrl+Z - 停止当前命令, 并使用 fg 恢复
	Ctrl+D - 注销当前会话, 与 exit 相似
	Ctrl+W - 删除当前行中的字
	Ctrl+U - 删除整行
	!! - 重复上次的命令
	exit - 注销当前会话
搜索	* 小心使用。
grep pattern files - 搜索 files 中匹配 pattern 的内容	翻译/Toy <http://LinuxTOY.org>
grep -r pattern dir - 递归搜索 dir 中匹配 pattern 的内容	
command grep pattern - 搜索 command 输出中匹配 pattern 的内容	

HostLoc.com

上图为 linux 常用命令。这里我就不一一介绍了，只是简单介绍几条命令。

- 查看操作系统版本
`cat /proc/version`
- 查看主板版本
`cat /proc/cpuinfo`
- 查看 SD 存储卡剩余空间
`df -h`
- 查看 ip 地址
`ifconfig`
- 压缩: `tar -zcvf filename.tar.gz dirname`
- 解压: `tar -zxvf filename.tar.gz`
- linux 系统常用 apt(Advanced Package Tool)高级软件工具来安装软件

```
sudo apt-get install xxx #安装软件。
```

```
sudo apt-get update #更新软件列表。
```

```
sudo apt-get upgrade #更新已安装软件。
```

```
sudo apt-get remove xxx #删除软件。
```

例如:运行以下两个命令安装 `sl`,`cmatrix`

```
sudo apt-get install sl
```

```
sudo apt-get install cmatrix
```

安装完成后运行以下两个命令

```
sl
```

```
cmatrix
```

(别问我这两个软件有什么用,我是不会告诉你的。试一下你就知道了。)

`sudo` 是增加用户权限,在命令行前面添加 `sudo` 相当于以 `root` 用户运行这条命令。可以运行 `sudo su` 直接切换到 `root` 用户操作。我个人喜欢在普通用户登陆。`"$"`为普通用户,`"#"`为超级用户。

```
pi@raspberrypi ~ $ sudo su
```

```
root@raspberrypi:/home/pi# su pi
```

```
pi@raspberrypi ~ $
```

2) vi/vim 编辑器

linux 常用的编辑工具有 `nano` ,`vi/vim` (`vim` 是 `vi` 的增强版)等。新手建议使用 `nano` 编辑器,简单易用。我个人则更加喜欢使用 `vi/vim` 编辑器,如果要使用 `vi` 编辑器首先得重新安装 `vi` 编辑器,因为树莓派自带的编辑器比较坑,谁用谁知道。

首先删除默认 `vi` 编辑器

```
sudo apt-get remove vim-common
```

然后重装 `vim`

```
sudo apt-get install vim
```

为方便使用还得在 `/etc/vim/vimrc` 文件后面添加下面三句

```
set nu #显示行号
```

syntax on #语法高亮

```
set tabstop=4 #tab 退四格
```

效果如下

```
pi pi@raspberrypi: ~/Pioneer600/LED/python - Xshell 5
文件(F) 编辑(E) 查看(V) 工具(T) 选项卡(B) 窗口(W) 帮助(H) [Icons] [Search] [Print] [Fullscreen] [Refresh] [Close All] [More]

#!/usr/bin/python
# -*- coding:utf-8 -*-
import RPi.GPIO as GPIO
import time

LED = 20

GPIO.setmode(GPIO.BCM)
GPIO.setup(LED, GPIO.OUT)
try:
    while True:
        GPIO.output(LED, GPIO.HIGH)
        time.sleep(1)
        GPIO.output(LED, GPIO.LOW)
        time.sleep(1)
except:
    print("except")
    GPIO.cleanup()
~
~
~
~
~
~
~
~

~/led.py" 18L, 284C 1,1 全部
```

```
pi pi@raspberrypi: ~/Pioneer600/LED/python - Xshell 5
文件(F) 编辑(E) 查看(V) 工具(T) 选项卡(B) 窗口(W) 帮助(H) [Icons] [Search] [Print] [Fullscreen] [Refresh] [Close All] [More]

1 #!/usr/bin/python
2 # -*- coding:utf-8 -*-
3 import RPi.GPIO as GPIO
4 import time
5
6 LED = 20
7
8 GPIO.setmode(GPIO.BCM)
9 GPIO.setup(LED, GPIO.OUT)
10 try:
11     while True:
12         GPIO.output(LED, GPIO.HIGH)
13         time.sleep(1)
14         GPIO.output(LED, GPIO.LOW)
15         time.sleep(1)
16 except:
17     print("except")
18     GPIO.cleanup()
~
~
~
~
~
~
~
~

~/led.py" 18L, 284C 1,1 全部
```

vi 有 3 个模式：插入模式、命令模式、低行模式。

插入模式：在此模式下可以输入字符，按 **ESC** 将回到命令模式。

命令模式：可以移动光标、删除字符等。

低行模式：可以保存文件、退出 vi、设置 vi、查找等功能(低行模式也可以看作是命令模式里

的)

打开文件、保存、关闭文件(vi 命令模式下使用):

```
vi filename //打开 filename 文件
:w //保存文件
:q //退出编辑器, 如果文件已修改请使用下面的命令
:q! //退出编辑器, 且不保存
:wq //退出编辑器, 且保存文件
```

插入文本或行(vi 命令模式下使用, 执行下面命令后将进入插入模式, 按 ESC 键可退出插入模式):

```
a //在当前光标位置的右边添加文本
i //在当前光标位置的左边添加文本
A //在当前行的末尾位置添加文本
I //在当前行的开始处添加文本(非空字符的行首)
O //在当前行的上面新建一行
o //在当前行的下面新建一行
R //替换(覆盖)当前光标位置及后面的若干文本
J //合并光标所在行及下一行为一行(依然在命令模式)
```

删除、恢复字符或行(vi 命令模式下使用):

```
x //删除当前字符
nx //删除从光标开始的 n 个字符
dd //删除当前行
ndd //向下删除当前行在内的 n 行
u //撤销上一步操作
U //撤销对当前行的所有操作
```

复制、粘贴(vi 命令模式下使用):

```
yy //将当前行复制到缓存区
nyy //将当前行向下 n 行复制到缓冲区
yw //复制从光标开始到词尾的字符
nyw //复制从光标开始的 n 个单词
y^ //复制从光标到行首的内容
y$ //复制从光标到行尾的内容
```

p //粘贴剪切板里的内容在光标后

P //粘贴剪切板里的内容在光标前

设置行号(vi 命令模式下使用)

:set nu //显示行号

:set nonu //取消显示行号

新手使用 vi 可能不习惯，慢慢的被虐多了就觉习惯了。顺便提醒一句，linux 系统是区分大小写的。另附 vi/vim 键盘图一张。

version 1.1
April 1st, 06
翻译: 2006-5-22

vi / vim 键盘图

Esc 命令模式	~ 转换大小写 跳转到标注	! 外部过滤器	@ 运行宏	# prev ident	\$ 行末	% 整句匹配	^ "款" 行首	& 重复 :s	* next ident	(句首) 下一句首	= "soft" bol down 前一行行首	+ 后一行行首
1	2	3	4	5	6	7	8	9	0	- 前一 行首	= 自动 格式化		
Q 切换到 ex模式	W 下一 单词	E 词尾	R 转换 模式	T back 'till	Y 拷贝 行	U 撤消 行内命令	I 到行首 插入	O 分段 (前)	P 粘贴 (前)	{ 段首	}	段尾	
q 录制 宏	w 下一 单词	e 词尾	r 替换 字符	t 'till	y 拷贝 行	u 撤消 命令	i 插入 模式	o 分段 (后)	p 粘贴 (后)	[杂项]	杂项	
A 在行末 附加	S 删除行 并插入	D 删除 至行末	F 行内字符 反向查找	G 文尾/ 行号	H 屏幕 顶行	J 合并 两行	K 帮助	L 屏幕 底行	:	ex 命令	" 寄存器 标识	.	行首/ 列
a 附加	s 删除字符 并插入	d 删除	f 行内字符 查找	g 附加 命令	h 向左 移动	j 向下 移动	k 向上 移动	l 向右 移动	;	重复 命令	.	跳转到标 注的行首	! 未使用!
Z 退出	X 退格	C 修改 至行末	V 可视 模式	B 前单 词	W 屏幕 中间行	M 屏幕 中间行	< 反向进	> 缩进	?	向前 搜索	.	向后 搜索	
Z 附加 命令	x 删除 (字符)	c 修改	v 可视 模式	b 前单 词	w 屏幕 中间行	m 设置 标注	< 反向 命令	> 重复 命令	/	向前 搜索	.	向后 搜索	

动作 移动光标, 或者定义操作的范围

命令 直接执行的命令,
红色命令 进入编辑模式

操作 后面跟随表示操作范围的指令

extra 特殊功能,
需要额外的输入

q. 后跟字符参数

w,e,b命令
小写(b): quux(foo, bar, baz)
大写(B): QUUX(foo, BAR, BAZ)

主要ex命令:
:w (保存), :q (退出), :q! (不保存退出)
:e f (打开文件 f),
:%s/x/y/g ('y' 全局替换 'x'),
:h (帮助 in vim), :new (新建文件 in vim)

其它重要命令:
CTRL-R: 重复 (vim),
CTRL-F/-B: 上翻/下翻,
CTRL-E/-Y: 上滚/下滚,
CTRL-V: 块可视模式 (vim only)

可视模式:
漫游后对选中的区域执行操作 (vim only)

备注:
(1) 在 拷贝/粘贴/删除 命令前使用 "x (x=a..z,*)
使用命令的寄存器(剪贴板)
(如: "ay\$ 拷贝剩余的行内容至寄存器 'a')
(2) 命令前添加数字
多遍重复操作
(e.g.: 2p, d2w, 5i, d4j)
(3) 重复本字符在光标所在行执行操作
(dd = 删除本行, >> = 行首缩进)
(4) ZZ 保存退出, ZQ 不保存退出
(5) zt: 移动光标所在行至屏幕顶端,
zb: 底端, zz: 中间
(6) gg: 文首 (vim only),
gf: 打开光标处的文件名 (vim only)

原图: www.viemu.com 翻译: fdl (linuxsir)

树莓派教程系列 6：文件共享(samba)

我们使用树莓派的时候经常要在 windows 和树莓派之间进行文件传输，使用 samba 服务可实现文件共享。在 windows 的网上邻居即可访问树莓派文件系统，非常方便。

- 1) 运行以下命令安装 samba 软件

```
sudo apt-get install samba samba-common-bin
```

- 2) 安装完成后，修改配置文件/etc/samba/smb.conf

```
sudo vi /etc/samba/smb.conf
```

下面的配置是让用户可以访问自己的 home 目录。

- a) 开启用户认证，找到“##### Authentication #####”，将“# security = user ”的#号去掉。
- b) 配置每个用户可以读写自己的 home 目录，在“[homes]”节中，把 “read only = yes” 改为 “read only = no”。

- 3) 重启 samba 服务

```
sudo /etc/init.d/samba restart
```

或

```
sudo service samba restart
```

- 4) 添加默认用户 pi 到 samba

```
sudo smbpasswd -a pi
```

输入密码确定即可。

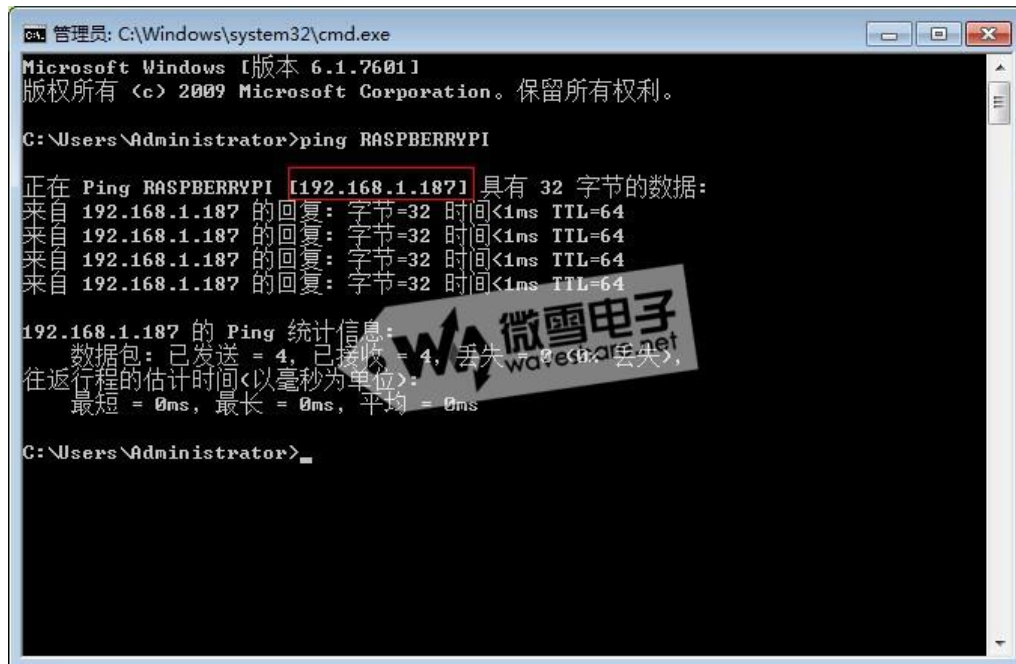
- 5) 访问树莓派文件

使用文件浏览器打开 ip 地址\\192.168.1.110\pi (ip 地址改为树莓派 IP 地址)，输入用户密码，则可以访问树莓派 home 目录。

在 windows 网络中你会发现多了台电脑 RASPBERRYPI，以后直接点击即可访问树莓派存储器，或在 windows 开始菜单中运行打开\\RASPBERRYPI。

(提示：如果树莓派开机启动 Samba 服务器，而又不知道树莓派 IP, 可以在 windows 命令

行中 ping RASPBERRYPI 返回树莓派的 ip 地址。)



The screenshot shows a Windows command prompt window titled "管理员: C:\Windows\system32\cmd.exe". The window displays the output of the command "ping RASPBERRYPI". The output shows that the IP address 192.168.1.187 is associated with RASPBERRYPI. The ping results show four successful replies with a time of less than 1ms and a TTL of 64. The statistics section shows that 4 packets were sent and received, with 0ms round-trip time.

```
C:\Users\Administrator>ping RASPBERRYPI

正在 Ping RASPBERRYPI [192.168.1.187] 具有 32 字节的数据:
来自 192.168.1.187 的回复: 字节=32 时间<1ms TTL=64
来自 192.168.1.187 的回复: 字节=32 时间<1ms TTL=64
来自 192.168.1.187 的回复: 字节=32 时间<1ms TTL=64
来自 192.168.1.187 的回复: 字节=32 时间<1ms TTL=64

192.168.1.187 的 Ping 统计信息:
    数据包: 已发送 = 4, 已接收 = 4, 丢失 = 0 (0% 丢失),
    往返行程的估计时间(以毫秒为单位):
        最短 = 0ms, 最长 = 0ms, 平均 = 0ms

C:\Users\Administrator>
```


树莓派系列教程 7：如何点亮一个 LED 灯(上)

树莓派的强大之处不单单是因为它是一个卡式电脑,更重要的是个引出 **GPIO**, 可以通过编程控制 **GPIO** 管脚输出高低电平。学过 51 单片机的孩童第一个程序就是点亮一个 LED 灯, 从此就点亮我们的人生, 从此 code 奸我千百遍, 我待 code 如初见。今天我们就来探讨一下树莓派点亮一个 LED 灯的 n 种方法。从这一章开始我们将教大家如何在树莓派编程, 在学习树莓派编程前, 你需要一块树莓扩展板。本教程是 WaveShare 设计的 Pioneer600 扩展板为例。Pioneer600 扩展板包括了 **GPIO, I2C, SPI, Serial** 等接口的器件, 是学习树莓派编程很好的扩展板。关于 Pioneer600 扩展的详细资料看网站。

1) 通过 shell 脚本操作 GPIO

```
# 进入 GPIO 目录
cd /sys/class/gpio

# 运行 ls 命令查看 gpio 目录中的内容, 可以查看到 export gpiochip0 unexport 三个文件
sudo ls

# GPIO 操作接口从内核空间暴露到用户空间
# 执行该操作之后, 该目录下会增加一个 gpio26 文件
echo 26 > export

# 进入 GPIO26 目录, 该目录由上一步操作产生
cd gpio26

# 运行 ls 查看 gpio26 目录中的内容, 可查看到的内容如下
# active_low direction edge power subsystem uevent value
sudo ls

# 设置 GPIO26 为输出方向
echo out > direction

# BCM_GPIO26 输出逻辑高电平, LED 点亮
echo 1 > value

# BCM_GPIO26 输出逻辑低电平, LED 熄灭
echo 0 > value

# 返回上一级目录
cd ..

# 注销 GPIO20 接口
echo 20> unexport
```

注: echo 命令为打印输出, 相当于 C 语言的 printf 函数的功能, >符号为 IO 重定向符号, IO 重定向是指改变 linux 标准输入和输出的默认设备, 指向一个用户定义的设备。例如 echo 20 > export 便是把 20 写入到 export 文件中

我们可以编写成 `shell` 脚本的形式运行

```
vi led.sh
```

用 `vi` 新建 `led.sh` 文件，添加以下程序并保存退出。

```
#!/bin/bash
echo Exporting pin $1
echo $1 > /sys/class/gpio/export
echo Setting direction to out.
echo out > /sys/class/gpio/gpio$1/direction
echo Setting pin $2
echo $2 > /sys/class/gpio/gpio$1/value
```

修改文件属性，使文件可执行。

```
chmod +x led.sh
```

程序第一句注销表明这个是一个 `bash shell` 文件，通过 `/bin/bash` 程序执行。

`$1` 代表第一个参数，`$2` 代表第二个参数，执行以下两个命令可点亮和熄灭 LED(Pioneer600 扩展板 LED1 接到树莓派 BCM 编码的 26 号管脚)。

```
sudo ./led.sh 26 1
```

```
sudo ./led.sh 26 0
```

2) 通过 `sysfs` 方式操作 GPIO

通过上面的操作，我们可以发现在 `linux` 系统中，读写设备文件即可操作对应的设备。所以说在 `linux` 的世界，一切的都是文件。下面我们可以通过 `C` 语言读写文件的方式操作 GPIO。

```
vi led.c
```

使用 `vi` 新建 `led.c` 文件，添加以下程序并保存退出。

```
#include <sys/stat.h>
#include <sys/types.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#define IN 0
#define OUT 1

#define LOW 0
#define HIGH 1

#define POUT 26
#define BUFFER_MAX 3
```

```
#define DIRECTION_MAX 48

static int GPIOExport(int pin)
{
    char buffer[BUFFER_MAX];
    int len;
    int fd;

    fd = open("/sys/class/gpio/export", O_WRONLY);
    if (fd < 0) {
        fprintf(stderr, "Failed to open export for writing!\n");
        return(-1);
    }

    len = snprintf(buffer, BUFFER_MAX, "%d", pin);
    write(fd, buffer, len);

    close(fd);
    return(0);
}

static int GPIOUnexport(int pin)
{
    char buffer[BUFFER_MAX];
    int len;
    int fd;

    fd = open("/sys/class/gpio/unexport", O_WRONLY);
    if (fd < 0) {
        fprintf(stderr, "Failed to open unexport for writing!\n");
        return(-1);
    }

    len = snprintf(buffer, BUFFER_MAX, "%d", pin);
    write(fd, buffer, len);

    close(fd);
    return(0);
}

static int GPIODirection(int pin, int dir)
{

```

```
static const char dir_str[] = "in\0out";
char path[DIRECTION_MAX];
int fd;

snprintf(path, DIRECTION_MAX, "/sys/class/gpio/gpio%d/direction", pin);
fd = open(path, O_WRONLY);
if (fd < 0) {
    fprintf(stderr, "failed to open gpio direction for writing!\n");
    return(-1);
}

if (write(fd, &dir_str[dir == IN ? 0 : 3], dir == IN ? 2 : 3) < 0) {
    fprintf(stderr, "failed to set direction!\n");
    return(-1);
}

close(fd);
return(0);
}

static int GPIORead(int pin)
{
    char path[DIRECTION_MAX];
    char value_str[3];
    int fd;

    snprintf(path, DIRECTION_MAX, "/sys/class/gpio/gpio%d/value", pin);
    fd = open(path, O_RDONLY);
    if (fd < 0) {
        fprintf(stderr, "failed to open gpio value for reading!\n");
        return(-1);
    }

    if (read(fd, value_str, 3) < 0) {
        fprintf(stderr, "failed to read value!\n");
        return(-1);
    }

    close(fd);
    return(atoi(value_str));
}
```

```
static int GPIOWrite(int pin, int value)
{
    static const char s_values_str[] = "01";
    char path[DIRECTION_MAX];
    int fd;

    snprintf(path, DIRECTION_MAX, "/sys/class/gpio/gpio%d/value", pin);
    fd = open(path, O_WRONLY);
    if (fd < 0) {
        fprintf(stderr, "failed to open gpio value for writing!\n");
        return(-1);
    }

    if (write(fd, &s_values_str[value == LOW ? 0 : 1], 1) < 0) {
        fprintf(stderr, "failed to write value!\n");
        return(-1);
    }

    close(fd);
    return(0);
}

int main(int argc, char *argv[])
{
    int i = 0;

    GPIOExport(POUT);
    GPIODirection(POUT, OUT);

    for (i = 0; i < 20; i++) {
        GPIOWrite(POUT, i % 2);
        usleep(500 * 1000);
    }

    GPIOUnexport(POUT);
    return(0);
}</unistd.h></string.h></stdlib.h></stdio.h></fcntl.h></sys/types.h>
```

编译并执行程序

```
gcc led.c -o led
```

```
sudo ./led
```

如果没有意外，我们可以看到接到 BCM 编码的 26 号管脚的 LED, 闪烁 10 次后自动退出程序。

以上第一条命令是使用 `gcc` 编译器将 `led.c` 源文件，编译成 `led` 可执行文件，在目录下面我们可以发现编程后生产的 `led` 可执行文件。我们也可以编写 `Makefile` 文件, 下次直接运行 `make` 命令即可编译程序。

`vi Makefile`

使用 `vi` 新建 `Makefile` 文件，添加以下代码并保存退出。

```
led:led.c
    gcc led.c -o led
clean:
    rm led
```

运行以下命令即可编译 `led.c` 程序

`make`

运行以下命令即可删除编译产生的可执行文件 `led`

`make clean`

余下教程都为方便解说都是运行 `gcc` 命令编译程序，如果想编写 `Makefile` 文件可查看 `Pioneer600` 示例程序。

关于 `sysfs` 操作 `GPIO` 方式，详情请参考以下网站：

<https://bitbucket.org/xukai871105/rpi-gpio-sysfs>

树莓派系列教程 8 ： 如何点亮一个 LED 灯（下）

上一章我们讲解了在 linux 系统下如何通过读写设备文件的方式控制 GPIO 点亮 LED 灯，本章我们继续讲解如果通过使用中间层库函数编程控制 GPIO。

WIRINGPI

WiringPi 是应用于树莓派平台的 GPIO 控制库函数，WiringPi 中的函数类似于 Arduino 的 wiring 系统。官网：<http://wiringpi.com/>

1、wiringPi 安装

（1）方案 1-使用 GIT 工具

通过 GIT 获得 wiringPi 的源码

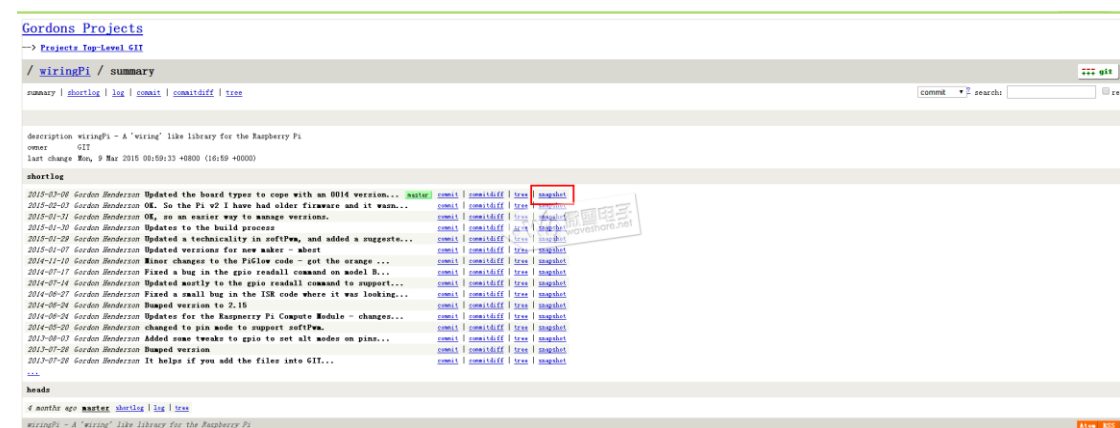
```
git clone git://git.drogon.net/wiringPi
cd wiringPi
./build
```

build 脚本会帮助你编译和安装 wiringPi

（2）方案 1-使用 GIT 工具

我们可以在网站上直接下载最新版本编译使

用, <https://git.drogon.net/?p=wiringPi;a=summary>



在 windows 上下载 wiringPi 库，并复制到树莓派中，运行如下命令解压安装。（xxx 代表版本号）

```
tar -zxvf wiringPi-xxx.tar.gz
cd wiringPi-xxx
./build
```

详细安装教程请参考 wiringPi 官网: <http://wiringpi.com/download-and-install/>

2、测试

wiringPi 包括一套 gpio 命令, 使用 gpio 命令可以控制树莓派上的各种接口, 通过以下指令可以测试 wiringPi 是否安装成功。

```
<div style="text-align: left;"><span style="font-size: 9pt; line-height: 1.8em;">gpio -v</span></div>gpio readall
```

```
pi@raspberrypi: ~ - Xshell 5
pi@raspberrypi ~$ gpio -v
gpio version: 2.26
Copyright (c) 2012-2015 Gordon Henderson
This is free software with ABSOLUTELY NO WARRANTY.
For details type: gpio -warranty

Raspberry Pi Details:
Type: Model 2, Revision: 1.1, Memory: 1024MB, Maker: Sony
pi@raspberrypi ~$ gpio readall
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| BCM | wPi | Name | Mode | V | Physical | V | Mode | Name | wPi | BCM |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 2 | 8 | 3.3v | ALTO | 1 | 3 | 4 | 5v | 5v | 15 | 14 | |
| 3 | 9 | SDA.1 | ALTO | 1 | 5 | 6 | 0v | 0v | 16 | 15 |
| 4 | 7 | SCL.1 | IN | 0 | 7 | 8 | 1 | TXD | 17 | 14 |
| 17 | 0 | GPIO.0 | OUT | 0 | 9 | 10 | 1 | RXD | 18 | 15 |
| 27 | 2 | GPIO.2 | IN | 0 | 11 | 12 | 0 | GPIO.1 | 19 | 18 |
| 22 | 3 | GPIO.3 | IN | 0 | 13 | 14 | 0 | 0v | 20 | 18 |
| 10 | 12 | MOSI | ALTO | 0 | 15 | 16 | 0 | IN | GPIO.4 | 21 | 23 |
| 9 | 13 | MISO | ALTO | 0 | 17 | 18 | 0 | IN | GPIO.5 | 22 | 24 |
| 11 | 14 | SCLK | ALTO | 0 | 19 | 20 | 0 | IN | GPIO.6 | 23 | 25 |
| 0 | 30 | SDA.0 | IN | 1 | 21 | 22 | 1 | CE0 | 24 | 8 |
| 5 | 21 | GPIO.21 | OUT | 1 | 23 | 24 | 1 | CE1 | 25 | 7 |
| 6 | 22 | GPIO.22 | IN | 1 | 25 | 26 | 1 | SCL.0 | 26 | 1 |
| 13 | 23 | GPIO.23 | IN | 0 | 27 | 28 | 1 | 0v | 27 | 12 |
| 19 | 24 | GPIO.24 | IN | 0 | 29 | 30 | 0 | IN | GPIO.26 | 28 | 16 |
| 26 | 25 | GPIO.25 | IN | 0 | 31 | 32 | 0 | IN | GPIO.27 | 29 | 16 |
| | | 0v | | | 33 | 34 | 0 | IN | GPIO.28 | 30 | 20 |
| | | | | | 35 | 36 | 0 | IN | GPIO.29 | 31 | 21 |
| | | | | | 37 | 38 | 0 | IN | | | |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| BCM | wPi | Name | Mode | V | Physical | V | Mode | Name | wPi | BCM |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
pi@raspberrypi ~$
```

从上图可以知道树莓派管脚有三种编号方式, 下面以 Pioneer600 扩展板的 LED1 为例, 第一种为板上编号 (37), 即中间两列, 表示第几号插针。第二中为 wiringPi 编码 (25), 使用 wiringPi 库编程时是用 wiringPi 编号, 第三种为 BCM 编号, 这个是更加 BCM 管脚来编号的, 使用 bcm2835 库或 python 的 RPi.GPIO 编程时使用 BCM 编号。

我们可以使用 gpio 命令操作树莓派 GPIO, 下面我们可以 gpio 命令控制 Pioneer600 扩展的 LED1.

```
gpio -g mode 26 out
```


设置管脚为输出模式，`-g` 参数表示是以 BCM 编号方式，如果去掉这个参数则以 `wiringPi` 编号方式，即为 25。

```
gpio -g write 26 1
```

设置管脚为高电平，点亮 LED。

```
gpio -g write 26 0
```

设置管脚为低电平，熄灭 LED，

```
gpio -g read 26
```

读取管脚当前状态

更多 `gpio` 命令请查

看：<https://projects.drogon.net/raspberry-pi/wiringpi/the-gpio-utility/>

3、wiringPi 程序：

```
#include<wiringpi.h>
char LED = 25;

int main(void)
{
    if(wiringPiSetup() < 0)return 1;
    pinMode (LED,OUTPUT) ;

    while(1)
    {
        digitalWrite(LED, 1) ;
        delay (200);
        digitalWrite(LED, 0) ;
        delay (200);
    }
}
```

使用 `vi` 将代码添加到 `led.c` 文件中，运行如下命令编译并执行程序。按 `Ctrl+C` 终止程序。

```
gcc -Wall led.c -o led -lwiringPi
```

```
sudo ./led
```

注：（1）`-Wall` 表示编译时显示所有警告，`-lwiringPi` 表示编译时动态加载 `wiringPi` 库

（2）终止程序后，LED 的状态为不确定，这和 `python` 程序相比显得有点不足。

关于更多 `wiringPi` 的库函数，可参看 `wiringPi` 官网，也可参考 `wiringPi` 用户手册。

http://wenku.baidu.com/link?url=U_APBvE_ga5pSSwPwWABIGJymLVwyC-0W9AEOT2cjhlZzoLywa0-QpElyNT2yHvNV0P7BbqTZCgG0ctaQZLi_ovkAGXREB0E6h68eTt-Q3y

BCM2835

bcm2835 库是树莓派 cpu 芯片的库函数，相当于 stm32 的固件库一样，底层是直接操作寄存器。而 wiringPi 库和 python 的 RPi.GPIO 库其底层都是通过读写 linux 系统的设备文件操作设备。

1、安装 bcm2835 库

从 bcm2835 官网 (<http://www.airspayce.com/mikem/bcm2835/>) 下载最新版本的库，然后解压安装。

```
tar -zxvf bcm2835-1.xx.tar.gz
cd bcm2835-1.xx
./configure
make
sudo make check
sudo make install
```

2、示例程序

新建名为 led.c 的文件，添加以下程序。

```
#include <bcm2835.h>

#define PIN 26
int main(int argc, char **argv)
{
    if (!bcm2835_init())return 1;
    bcm2835_gpio_fsel(PIN, BCM2835_GPIO_FSEL_OUTP);

    while (1)
    {
        bcm2835_gpio_write(PIN, HIGH);
        bcm2835_delay(500);
        bcm2835_gpio_write(PIN, LOW);
        bcm2835_delay(500);
    }
    bcm2835_close();
    return 0;
}
```

编译并执行程序，按 Ctrl+C 可结束程序。

```
gcc -Wall led.c -o led -lbcm2835
sudo ./led
```

- 注：（1）-lbcm2835 表示动态加载 bcm2835 库
（2）注意 bcm2835 程序管脚使用 bcm 编号，和 wiringPi 编号不一样。
（3）和 wiringPi 一样，程序结束时 GPIO 的状态不确定。

PYTHON

1、安装 RPi.GPIO

- （1）先安装 python-dev, 输入以下指令。

```
sudo apt-get install python-dev
```

- （2）安装 RPi.GPIO

```
<span style="font-size: 9pt; line-height: 1.8em;">#下载
wget
https://pypi.python.org/packages/source/R/RPi.GPIO/RPi.GPIO-0.5.11.tar.gz
#解压:
tar -zxvf RPi.GPIO-0.5.11.tar.gz
#进入解压之后的目录 :
cd RPi.GPIO-0.5.3a
#启动安装 :
sudo python setup.py install</span>
```

2. 示例程序

新建 led.py 文件，添加如下代码并保存。

```
#!/usr/bin/python
# -*- coding:utf-8 -*-
import RPi.GPIO as GPIO
import time

LED = 26

GPIO.setmode(GPIO.BCM)
GPIO.setup(LED,GPIO.OUT)
try:
    while True:
        GPIO.output(LED,GPIO.HIGH)
        time.sleep(1)
        GPIO.output(LED,GPIO.LOW)
        time.sleep(1)
except:
    print("except")
```

```
<span style="font-size: 9pt; line-height: 25.2000007629395px;">
</span>GPIO.cleanup()
```

执行程序，按 **Ctrl+C** 结束程序

```
sudo python led.py
```

注：（1）`#!/usr/bin/env python`，定义 python 解析脚本的绝对路径。
（2）`# -*- coding: utf-8 -*-`，python 文件为 `utf-8` 格式，否则无法写入中文注释。
（3）`GPIO.setmode(GPIO.BCM)`，采用 `bcm` 编号方式。
（4）python 程序使用 `try except` 语言，当按下 **Ctrl+C** 结束程序是会触发异常，程序执行 `gpio.cleanup()` 语句清楚 GPIO 管脚状态。

对 python 语言不是很熟悉的孩童，可以查看 Python 基础教程：

<http://www.runoob.com/python/python-tutorial.html>

这里有一个 python 语言的 `wiringPi` 库，有兴趣的可以参考：

<https://github.com/WiringPi/WiringPi2-Python>

树莓派系列教程 9：按键

上两章我们讲解了在树莓派上如何点亮一个 LED 灯，这一章我们讲解一下按键以及事件中断。

BCM2835

```
#include <bcm2835.h>
#include <stdio.h>

#define KEY 20
int main(int argc, char **argv)
{
    if (!bcm2835_init())return 1;
    bcm2835_gpio_fsel(KEY, BCM2835_GPIO_FSEL_INPT);
    bcm2835_gpio_set_pud(KEY, BCM2835_GPIO_PUD_UP);
    printf("Key Test Program!!!!\n");
    while (1)
    {
        if(bcm2835_gpio_lev(KEY) == 0)
        {
            printf ("KEY PRESS\n") ;
            while(bcm2835_gpio_lev(KEY) == 0)
                bcm2835_delay(100);
        }
        bcm2835_delay(100);
    }
    bcm2835_close();
    return 0;
}</stdio.h></bcm2835.h>
```

编译并执行，按下按键会看到窗口显示“KEY PRESS”，按 Ctrl+C 结束程序。

```
gcc -Wall key.c -o key -lbcm2835
```

```
sudo ./key
```

注：（1）bcm2835_gpio_fsel(KEY, BCM2835_GPIO_FSEL_INPT);设置管脚为输入模式

（2）bcm2835_gpio_set_pud(KEY, BCM2835_GPIO_PUD_UP);设置为上拉模式

（3）bcm2835_gpio_lev(KEY); 读取管脚状态

WIRINGPI

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

char KEY = 29;

int main()
{
    if (wiringPiSetup() < 0)return 1 ;
    pinMode (KEY,INPUT);
    pullUpDnControl(KEY, PUD_UP);
    printf("Key Test Program!!!\n");
    while(1)
    {
        if (digitalRead(KEY) == 0)
        {
            printf ("KEY PRESS\n") ;
            while(digitalRead(KEY) == 0)
                delay(100);
        }
        delay(100);
    }
}
}</wiringpi.h></stdio.h>
```

编译并执行，按下按键会看到窗口显示”KEY PRESS”，按 Ctrl+C 结束程序。

```
gcc -Wall key.c -o key -wiringPi
```

```
sudo ./key
```

注：（1）pinMode (KEY,INPUT);设置管脚为输入模式

（2）pullUpDnControl(KEY, PUD_UP);设置为上拉模式

（3）digitalRead(KEY); 读取管脚状态

通过中断的方式编程

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

```
#define button 29
char flag = 0;
void myInterrupt()
{
    flag ++;
}

int main()
{
    if(wiringPiSetup() < 0)return 1;
    if(wiringPiISR(button,INT_EDGE_FALLING,&myInterrupt) < 0)
    {
        printf("Unable to setup ISR \n");
    }
    printf("Interrupt test program\n");
    while(1)
    {
        if(flag)
        {
            while(digitalRead(button) ==0);
            printf("button press\n");
            flag = 0;
        }
    }
}
</wiringpi.h></stdio.h>
```

编译并执行

```
gcc -Wall Interrupt.c -o Interrupt -lwiringPi
```

```
sudo ./Interrupt
```

注：（1）`wiringPiISR(button,INT_EDGE_FALLING,&myInterrupt)`；设置中断下降沿触发，`myInterrupt` 为中断处理函数。

PYTHON

```
#!/usr/bin/python
# -*- coding:utf-8 -*-
import RPi.GPIO as GPIO
```

```
import time

KEY = 26

GPIO.setmode(GPIO.BCM)
GPIO.setup(KEY,GPIO.IN,GPIO.PUD_UP)
while True:
    time.sleep(0.05)
    if GPIO.input(KEY) == 0:
        print("KEY PRESS")
        while GPIO.input(KEY) == 0:
            time.sleep(0.01)
```

执行程序，按下按键会看到窗口显示“KEY PRESS”，按 Ctrl+C 结束程序。

```
sudo python key.py
```

注：（1）GPIO.setup(KEY,GPIO.IN,GPIO.PUD_UP) 设置管脚为上拉输入模式

（2）GPIO.input(KEY) 读取管脚值

通过中断模式编程

```
#!/usr/bin/python
# -*- coding:utf-8 -*-
import RPi.GPIO as GPIO
import time

KEY = 26

def MyInterrupt(KEY):
    <span style="font-size: 9pt; line-height: 25.2000007629395px;">
    </span>print("KEY PRESS")

GPIO.setmode(GPIO.BCM)
GPIO.setup(KEY,GPIO.IN,GPIO.PUD_UP)
GPIO.add_event_detect(KEY,GPIO.FALLING,MyInterrupt,200)

while True:
    <span style="font-size: 9pt; line-height: 25.2000007629395px;">
    </span>time.sleep(1)
```

注：（1）def MyInterrupt(KEY)：定义中断处理函数

(2) `GPIO.add_event_detect(KEY,GPIO.FALLING,MyInterrupt,200)` 增加事件检测，下降沿触发，忽略由于开关抖动引起的小于 `200ms` 的边缘操作。

关于树莓派事件中断编程请参考：

<http://www.guokr.com/post/480073/focus/1797650173/>

树莓派系列教程 10: I2C

启动 I2C

执行如下命令进行树莓派配置

```
sudo raspi-config
```

选择 Advanced Options -> I2C ->yes 启动 i2c 内核驱动

除了启动 i2c 内核驱动外，还需修改配置文件，运行如下命令打开配置文件。

```
sudo nano /etc/modules
```

增加以下两行并保存退出。

```
i2c-bcm2708
```

```
i2c-dev
```

运行 lsmod 命令查看 i2c 时候启动

I2C-TOOLS

安装 i2c-tools，这个工具在 I2c 硬件监控设备识别和故障诊断是非常重要的。

```
sudo apt-get install i2c-tools
```

【i2c-tools 官网】<http://www.lm-sensors.org/wiki/i2cToolsDocumentation>

I2c-tools 仅有四条命令，下面逐条介绍。

1、i2c-tool 查询 i2c 设备：

```
i2cdetect -y 1
```

-y 代表取消用户交互过程，直接执行指令；

1 代表 I2C 总线编号；

上图是树莓派接上 Pioneer 600 扩展板检测到的 i2c 设备，

0x20 是 PCF8974 IO 扩展芯片的地址，

0x48 是 PCF8591 AD/DA 转换芯片的地址

0x68 是 DS3231 RTC 时钟芯片的地址

0x77 是 BMP180 压强传感器的地址

2、扫描寄存器内容：

```
i2cdump -y 1 0x68
```

-y 代表取消用户交互过程，直接执行指令；

1 代表 I2C 总线编号；

0x68 代表 I2C 设备从机地址，此处表示 DS3231 RTC 时钟芯片

3、寄存器内容写入：

```
i2cset -y 1 0x68 0x00 0x13
```

-y 代表曲线用户交互过程，直接执行指令

1 代表 I2C 总线编号

0x68 代表 I2C 设备地址，此处表示 DS3231 RTC 时钟芯片

0x00 代表存储器地址，

0x13 代表存储器地址中的具体内容

4、寄存器内容读出：

```
i2cget -y 1 0x68 0x00
```

-y 代表曲线用户交互过程，直接执行指令

1 代表 I2C 总线编号

0x68 代表 I2C 设备地址，此处表示 DS3231 RTC 时钟芯片

0x00 代表存储器地址

使用 I2C-TOOLS 控制 PCF8574 IO

PCF8574 是 I2C 总线 8 位 IO 扩展芯片, 初始状态 IO 为高电平. PCF8574 和其他 I2C 芯片不同, 该芯片没有寄存器, 直接传输一个字节即控制 IO 输出状态. Pioneer 600 扩展板, LED2 接到 PCF8574 的 p4 管脚. 低电平点亮 LED.

写 IO 管脚:

```
i2cset -y 1 0x20 0xEF
```

0x20 代表 I2C 设备地址, 此处表示 PCF8574 芯片

0xEF 传输的内容, 此处表示 PCF8574 的 P4 管脚输出低电平, 其他管脚输出高电平

可以看到 Pioneer 600 扩展板的 LED2 点亮

读 IO 管脚:

```
i2cget -y 1 0x20
```

运行这条命令即可读取 PCF8574 IO 管脚状态

熄灭 LED2

```
i2cset -y 1 0x20 0xFF
```

同理, Pioneer 600 扩展板, 蜂鸣器接到 PCF8574 的 P7 管脚, 可以用 i2c-tool 控制蜂鸣器

蜂鸣器响:

```
i2cset -y 1 0x20 0xEF
```

蜂鸣器停:

```
i2cset -y 1 0x20 0xFF
```

树莓派系列教程 11：I2C 编程

例程是通过 i2c 控制 pcf8574 IO,使 Pioneer 600 扩展板的 LED2 闪烁。

BCM2835

```
#include <bcm2835.h>

int main(int argc, char **argv)
{
    char buf[1];

    if (!bcm2835_init())return 1;
    bcm2835_i2c_begin();
    bcm2835_i2c_setSlaveAddress(0x20); //i2c address
    bcm2835_i2c_set_baudrate(10000); //1M baudrate

    while(1)
    {
        buf[0] = 0xEF;    //LED ON
        bcm2835_i2c_write(buf,1);
        bcm2835_delay(500);
        buf[0] = 0xFF;    //LED OFF
        bcm2835_i2c_write(buf,1);
        bcm2835_delay(500);
    }
    bcm2835_i2c_end();
    bcm2835_close();
    return 0;
} </bcm2835.h>
```

编译并执行，扩展板上的 LED2 灯开始闪烁了，Ctrl +C 结束程序

```
gcc -Wall pcf8574 -o pfc8574 -lbcm2835
```

```
sudo ./pcf8574
```

注：（1）bcm2835_i2c_begin(); 启动 i2c 操作，设置 I2C 相关引脚为复用功能

（2）bcm2835_i2c_setSlaveAddress(0x20); 设置 I2C 从机设备的地址，此处为 0x20。即 PCF8574 的地址。

(3) `bcm2835_i2c_write(buf,1)`; 传输字节到 i2c 从设备, `buf` 为要传输的数据, `1` 表示传输一个字节

更多 `bcm2835` 库 i2c 操作函数请查看:

http://www.airspayce.com/mikem/bcm2835/group__i2c.html#ga1309569f7363853333f3040b1553ea64

WIRINGPI

```
#include <wiringpi.h>
#include <wiringpi2c.h>

int main (void)
{
    int fd;
    wiringPiSetup();
    fd = wiringPiI2CSetup(0x20);

    while (1)
    {
        wiringPiI2CWrite(fd,0xEF); //LED ON
        delay(500);
        wiringPiI2CWrite(fd,0xFF); //LED OFF
        delay(500);
    }
    return 0;
}
</wiringpi2c.h></wiringpi.h>
```

编译并执行, 扩展板上的 LED2 灯开始闪烁了, `Ctrl +C` 结束程序

```
gcc -Wall pcf8574 -o pcf8574 -lbcm2835
```

```
sudo ./pcf8574
```

注: (1) `fd = wiringPiI2CSetup(0x20)`; 初始化 I2C 设备, `0x20` 为 PCF8574 的 I2C 地址, 返回值是标准的 Linux 文件句柄, 如果错误则返回 -1. 由此可知, `wiringPi` 底层也是通过 `sysfs` 方式操作 I2C 设备 `/dev/i2c-1`

`wiringPi` 还有 `pcf8574` 的扩展库, 也可以调用 `pcf8574` 的库操作 IO.

```
#include <wiringpi.h #include="" <pcf8574.h="">

#define EXTEND_BASE 64
```

```
#define LED_EXTEND_BASE + 4
int main (void)
{
    wiringPiSetup();
    pcf8574Setup(EXTEND_BASE,0x20);
    pinMode(LED,OUTPUT);
    while (1)
    {
        digitalWrite(LED,LOW); //LED ON
        delay(500);
        digitalWrite(LED,HIGH); //LED OFF
        delay(500);
    }
    return 0;
}</wiringpi.h>
```

编译并执行

```
gcc -Wall pcf8574.c -o pcf8474 -lwiringPi
sudo ./pcf8574
```

更多 bcm2835 库 i2c 操作函数请查看:

<http://wiringpi.com/reference/i2c-library/>

<http://wiringpi.com/extensions/i2c-pcf8574/>

PYTHON

首先执行如下命令安装 **smbus** 库

```
sudo apt-get install python-smbus
```

编辑程序

```
#!/usr/bin/python
# -*- coding:utf-8 -*-
import smbus
import time

address = 0x20

bus = smbus.SMBus(1)
```

```
while True:
    bus.write_byte(address,0xEF)
    time.sleep(0.5)
    bus.write_byte(address,0xFF)
    time.sleep(0.5)
```

执行程序：

```
sudo python pcf8574.py
```

注：（1） `import smbus` 导入 `smbus` 模块

（2） `bus = smbus.SMBus(1)` 在树莓派版本 2 中，I2C 设备位于 `/dev/I2C-1`，所以此处的编号为 1

python 封装 SMBUS 操作函数具体代码请查看：<https://github.com/bivab/smbus-cffi>

四、sysfs

从上面编程，我们可以发现，`wiring`,`python` 程序都是通过读写 `i2c` 设备文件 `/dev/I2C-1` 操作 `i2c` 设备。故我们也可以用 `c` 语言读写文件的形式操作 `i2c` 设备。

```
#include <linux i2c-dev.h>
#include <errno.h>
#define I2C_ADDR 0x20
#define LED_ON 0xEF
#define LED_OFF 0xFF
int main (void) {
    int value;
    int fd;

    fd = open("/dev/i2c-1", O_RDWR);
    if (fd < 0) {
        printf("Error opening file: %s\n", strerror(errno));
        return 1;
    }
    if (ioctl(fd, I2C_SLAVE, I2C_ADDR) < 0) {
        printf("ioctl error: %s\n", strerror(errno));
        return 1;
    }
    while(1)
    {
        if(value == LED_ON)value = LED_OFF;
```



```
        else value = LED_ON;
        if( write( fd , &value, 1 ) != 1) {
            printf("Error writing file: %s\n", strerror(errno));
        }
        usleep(1000000);
    }
    return 0;
}</errno.h></linux>
```

编译并执行

```
gcc -Wall pcf8574.c -o pcf8574
sudo ./pcf8574
```

注：（1）`fd = open("/dev/i2c-1", O_RDWR)`；打开设备，树莓派版本 2 的 I2C 设备位于 `/dev/i2c-1`

（2）`ioctl(fd, I2C_SLAVE, I2C_ADDR)`；设置 I2C 从设备地址，此时 PCF8574 的从机地址为 `0x20`。

（3）`write(fd , &value, 1)`；向 PCF8574 写入一个字节，`value` 便是写入的内容，写入的长度为 1。

树莓派系列教程 12: I2C 总线控制 BMP180

通过上一章,相信各位对树莓派 I2C 编程有一定的了解了,今天我们继续使用 I2C 来控制 BMP180 压强传感器。BMP180 压强传感器操作原理比较简单,开机先通过 I2C 读取出 AC1,AC2,AC3,AC4,AC5,AC6,B1,B2,MB,MC,MD 等寄存器的值,这些寄存器的值作为校准时使用。如何读取温度寄存器,压强寄存器的值,根据下图公式算出测得的当前温度和压强。

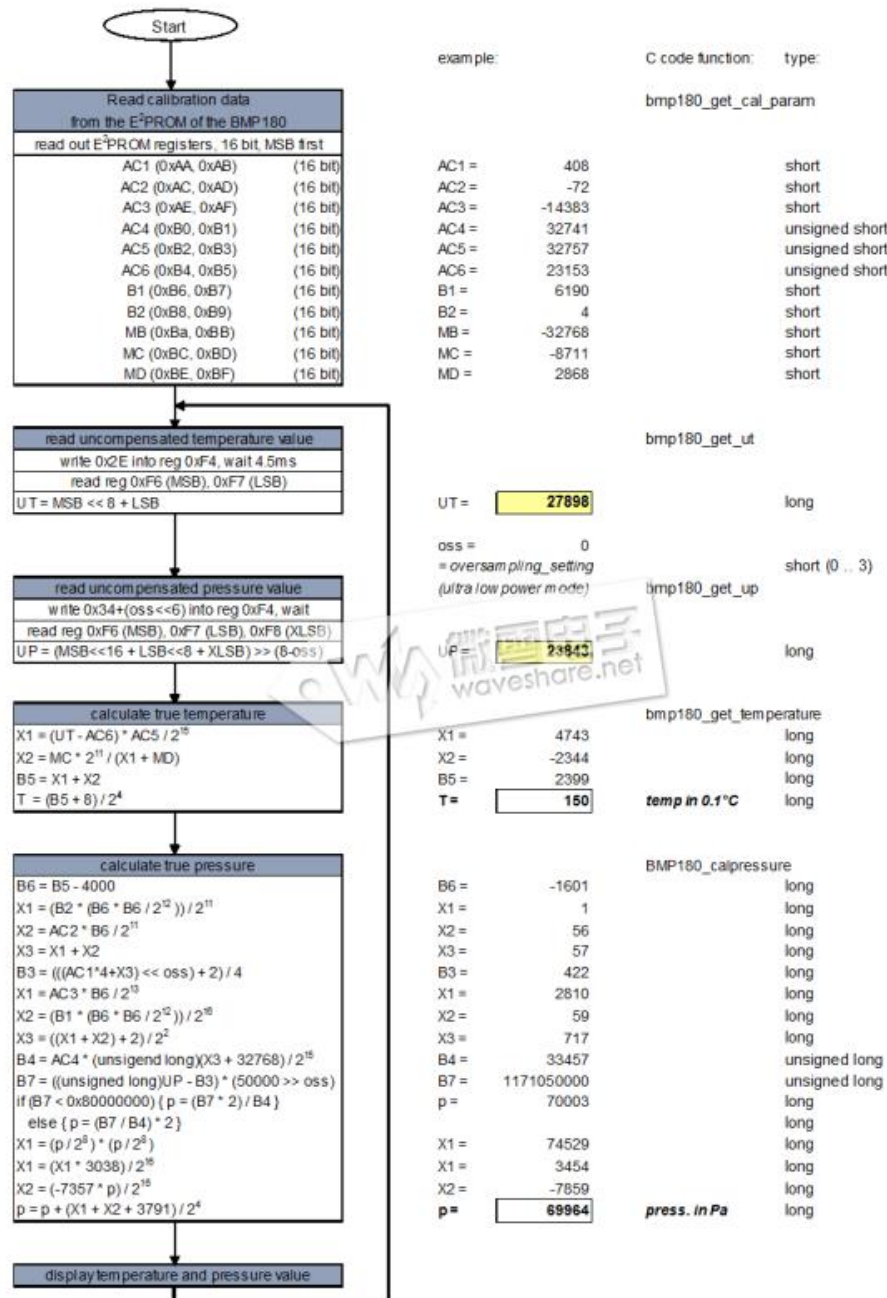


Figure 4: Algorithm for pressure and temperature measurement

本章主要讲解 python 程序，使大家熟悉 python 编程。关于 bcm2835,wiringpi 程序具体可参看 Pioneer600 示例程序。

驱动文件 bmp180.py

```
import time
import smbus

# BMP085 default address.
BMP180_I2CADDR      = 0x77

# Operating Modes
BMP180_ULTRALOWPOWER  = 0
BMP180_STANDARD       = 1
BMP180_HIGHRES        = 2
BMP180_ULTRAHIGHRES   = 3

# BMP085 Registers
BMP180_CAL_AC1        = 0xAA # R   Calibration data (16 bits)
BMP180_CAL_AC2        = 0xAC # R   Calibration data (16 bits)
BMP180_CAL_AC3        = 0xAE # R   Calibration data (16 bits)
BMP180_CAL_AC4        = 0xB0 # R   Calibration data (16 bits)
BMP180_CAL_AC5        = 0xB2 # R   Calibration data (16 bits)
BMP180_CAL_AC6        = 0xB4 # R   Calibration data (16 bits)
BMP180_CAL_B1         = 0xB6 # R   Calibration data (16 bits)
BMP180_CAL_B2         = 0xB8 # R   Calibration data (16 bits)
BMP180_CAL_MB         = 0xBA # R   Calibration data (16 bits)
BMP180_CAL_MC         = 0xBC # R   Calibration data (16 bits)
BMP180_CAL_MD         = 0xBE # R   Calibration data (16 bits)
BMP180_CONTROL        = 0xF4
BMP180_TEMPDATA       = 0xF6
BMP180_PRESSUREDATA   = 0xF6

# Commands
BMP180_READTEMPCMD    = 0x2E
BMP180_READPRESSURECMD = 0x34

class BMP180(object):
    def __init__(self, address=BMP180_I2CADDR, mode=BMP180_STANDARD):
        self._mode = mode
        self._address = address
```

```
self._bus = smbus.SMBus(1)
# Load calibration values.
self._load_calibration()
def _read_byte(self,cmd):
    return self._bus.read_byte_data(self._address,cmd)

def _read_u16(self,cmd):
    MSB = self._bus.read_byte_data(self._address,cmd)
    LSB = self._bus.read_byte_data(self._address,cmd+1)
    return (MSB << 8) + LSB

def _read_s16(self,cmd):
    result = self._read_u16(cmd)
    if result > 32767:result -= 65536
    return result

def _write_byte(self,cmd,val):
    self._bus.write_byte_data(self._address,cmd,val)

def _load_calibration(self):
    "load calibration"
    self.cal_AC1 = self._read_s16(BMP180_CAL_AC1)    # INT16
    self.cal_AC2 = self._read_s16(BMP180_CAL_AC2)    # INT16
    self.cal_AC3 = self._read_s16(BMP180_CAL_AC3)    # INT16
    self.cal_AC4 = self._read_u16(BMP180_CAL_AC4)    # UINT16
    self.cal_AC5 = self._read_u16(BMP180_CAL_AC5)    # UINT16
    self.cal_AC6 = self._read_u16(BMP180_CAL_AC6)    # UINT16
    self.cal_B1 = self._read_s16(BMP180_CAL_B1)      # INT16
    self.cal_B2 = self._read_s16(BMP180_CAL_B2)      # INT16
    self.cal_MB = self._read_s16(BMP180_CAL_MB)      # INT16
    self.cal_MC = self._read_s16(BMP180_CAL_MC)      # INT16
    self.cal_MD = self._read_s16(BMP180_CAL_MD)      # INT16

def read_raw_temp(self):
    """Reads the raw (uncompensated) temperature from the sensor."""
    self._write_byte(BMP180_CONTROL, BMP180_READTEMPCMD)
    time.sleep(0.005) # Wait 5ms
    MSB = self._read_byte(BMP180_TEMPDATA)
    LSB = self._read_byte(BMP180_TEMPDATA+1)
    raw = (MSB << 8) + LSB
    return raw
```

```
def read_raw_pressure(self):
    """Reads the raw (uncompensated) pressure level from the sensor."""
    self._write_byte(BMP180_CONTROL, BMP180_READPRESSURECMD + (self._mode
<< 6))
    if self._mode == BMP180_ULTRALOWPOWER:
        time.sleep(0.005)
    elif self._mode == BMP180_HIGHRES:
        time.sleep(0.014)
    elif self._mode == BMP180_ULTRAHIGHRES:
        time.sleep(0.026)
    else:
        time.sleep(0.008)
    MSB = self._read_byte(BMP180_PRESSUREDATA)
    LSB = self._read_byte(BMP180_PRESSUREDATA+1)
    XLSB = self._read_byte(BMP180_PRESSUREDATA+2)
    raw = ((MSB << 16) + (LSB << 8) + XLSB) >> (8 - self._mode)
    return raw

def read_temperature(self):
    """Gets the compensated temperature in degrees celsius."""
    UT = self.read_raw_temp()

    X1 = ((UT - self.cal_AC6) * self.cal_AC5) >> 15
    X2 = (self.cal_MC << 11) / (X1 + self.cal_MD)
    B5 = X1 + X2
    temp = ((B5 + 8) >> 4) / 10.0
    return temp

def read_pressure(self):
    """Gets the compensated pressure in Pascals."""
    UT = self.read_raw_temp()
    UP = self.read_raw_pressure()

    X1 = ((UT - self.cal_AC6) * self.cal_AC5) >> 15
    X2 = (self.cal_MC << 11) / (X1 + self.cal_MD)
    B5 = X1 + X2

    # Pressure Calculations
    B6 = B5 - 4000
    X1 = (self.cal_B2 * (B6 * B6) >> 12) >> 11
    X2 = (self.cal_AC2 * B6) >> 11
```

```

X3 = X1 + X2
B3 = (((self.cal_AC1 * 4 + X3) << self._mode) + 2) / 4

X1 = (self.cal_AC3 * B6) >> 13
X2 = (self.cal_B1 * ((B6 * B6) >> 12)) >> 16
X3 = ((X1 + X2) + 2) >> 2
B4 = (self.cal_AC4 * (X3 + 32768)) >> 15
B7 = (UP - B3) * (50000 >> self._mode)

if B7 < 0x80000000:
    p = (B7 * 2) / B4
else:
    p = (B7 / B4) * 2
X1 = (p >> 8) * (p >> 8)
X1 = (X1 * 3038) >> 16
X2 = (-7357 * p) >> 16

p = p + ((X1 + X2 + 3791) >> 4)
return p

def read_altitude(self, sealevel_pa=101325.0):
    """Calculates the altitude in meters."""
    # Calculation taken straight from section 3.6 of the datasheet.
    pressure = float(self.read_pressure())
    altitude = 44330.0 * (1.0 - pow(pressure / sealevel_pa, (1.0/5.255)))
    return altitude

def read_sealevel_pressure(self, altitude_m=0.0):
    """Calculates the pressure at sealevel when given a known altitude in
    meters. Returns a value in Pascals."""
    pressure = float(self.read_pressure())
    p0 = pressure / pow(1.0 - altitude_m/44330.0, 5.255)
    return p0

```

主文件 bmp180_example.py

```

#!/usr/bin/python

import time
from BMP180 import BMP180

# Initialise the BMP085 and use STANDARD mode (default value)
# bmp = BMP085(0x77, debug=True)

```

```
bmp = BMP180()

# To specify a different operating mode, uncomment one of the following:
# bmp = BMP085(0x77, 0) # ULTRALOWPOWER Mode
# bmp = BMP085(0x77, 1) # STANDARD Mode
# bmp = BMP085(0x77, 2) # HIRES Mode
# bmp = BMP085(0x77, 3) # ULTRAHIRES Mode
while True:
    temp = bmp.read_temperature()

# Read the current barometric pressure level
    pressure = bmp.read_pressure()

# To calculate altitude based on an estimated mean sea level pressure
# (1013.25 hPa) call the function as follows, but this won't be very accurate
    altitude = bmp.read_altitude()

# To specify a more accurate altitude, enter the correct mean sea level
# pressure level. For example, if the current pressure level is 1023.50 hPa
# enter 102350 since we include two decimal places in the integer value
# altitude = bmp.readAltitude(102350)

    print "Temperature: %.2f C" % temp
    print "Pressure:    %.2f hPa" % (pressure / 100.0)
    print "Altitude:    %.2f\n" % altitude
time.sleep(1)
```

树莓派系列教程 13: SERIAL 串口

树莓派的串口默认为串口终端调试使用，如要正常使用串口则需要修改树莓派设置。关闭串口终端调试功能后则不能再通过串口登陆访问树莓派，需从新开启后才能通过串口控制树莓派。

释放串口

执行如下命令进入树莓派配置

```
sudo raspi-config
```

选择 **Advanced Options** ->**Serial** ->**no** 关闭串口调试功能

使用 MINICOM 调试串口

设置完之后串口便可以正常使用了，便可测试一下树莓派的 UART 是否正常工作，**Pioneer600** 扩展板带有 USB 转 UART 功能，用 USB 线连接到电脑。**minicom** 便是一个简单好用的工具。**minicom** 是 linux 平台串口调试工具，相当于 windows 上的串口调试助手。

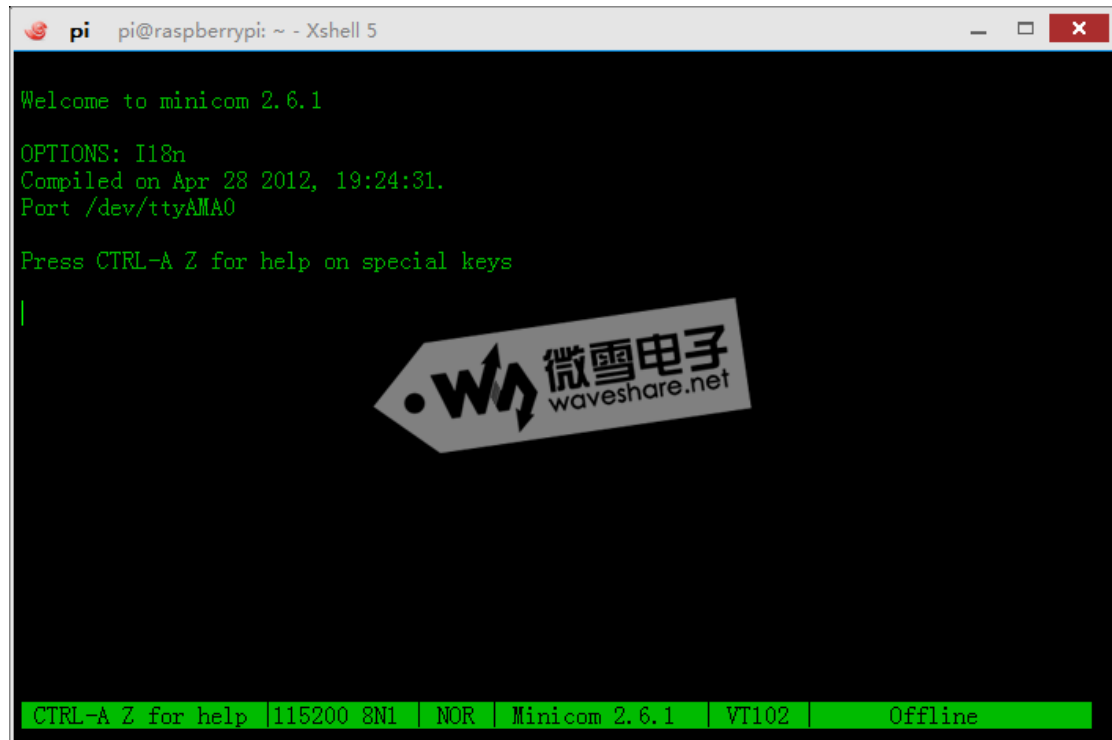
1、minicom 安装

```
sudo apt-get install minicom
```

2、minicom 启动

```
minicom -D /dev/ttyAMA0
```

默认波特率为 **115200**，如需设置波特率为 **9600** 加参数 **-b 9600**，**-D** 代表端口，**/dev/ttyAMA0** 类似于 windows 中的 **COM1**，



同时在 windows 也打开串口助手。设置波特率为 115200，选择对应的串口号



3、串口数据传输

直接在 **minicom** 控制台中输入内容即可通过串口发送数据，在 **windows** 串口助手中会接到到输入的内容。同理，在 **windows** 串口助手中发送数据会在 **minicom** 控制台上显示。如果 **minicom** 打开了回显（先 **Ctrl+A**，再 **E**）可在控制台中观察到输出内容，如果回显关闭 **minicom** 控制台不会显示你输入的内容。先 **Ctrl+A**，再 **Q**，退出 **minicom**。

串口编程

1、wiringPi

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

int main()
{
    int fd;
    if(wiringPiSetup() < 0)return 1;
    if((fd = serialOpen("/dev/ttyAMA0",115200)) < 0)return 1;
    printf("serial test start ...\n");
    serialPrintf(fd,"Hello World!!!\n");
    while(1)
    {
        serialPutchar(fd,serialGetchar(fd));
    }
    serialClose(fd);
    return 0;
}</wiringserial.h></wiringpi.h></stdio.h>
```

编译并执行,在 window 下打开串口助手会接收到” Hello World!!!”,发送数据会返回到显示窗口。

```
gcc -Wall uart.c -o uart -lwiringPi
sudo ./uart
```

PYTHON

首先运行如下命令安装 **python serial** 扩展库。

```
sudo apt-get install python-serial
```

编写程序

```
#!/usr/bin/python
# -*- coding:utf-8 -*-import serial

ser = serial.Serial("/dev/ttyAMA0",115200)

print('serial test start ...')
ser.write("Hello Wrold !!!\n")
try:
    while True:
        ser.write(ser.read())
except KeyboardInterrupt:
    if ser != None:
        ser.close()
```

执行程序，实验结果和上面一样。

```
sudo python uart.py
```

注：(1) `ser = serial.Serial("/dev/ttyAMA0",115200)` 打开串口，波特率为 115200

(2) `ser.write(ser.read())` 接收字符并回传

(3) `ser.close()` 关闭串口

总结：通过上面两个程序我们可以发现和 `i2c` 一样，`wiringPi`,`python` 程序都是读写串口设备文件 `/dev/ttyAMA0` 操作串口，故我们也可以通过 `sysfs` 的形式编程操作串口，在这里我就不详细介绍了。

树莓派系列教程 14：单总线控制 DS18B20

DS18B20 是一个比较常用的温度传感器，采用单总线控制，以前用单片机编程控制时严格按照单总线的时序控制，今天来看看在 linux 系统下如何控制 DS18B20, 体验一下在 linux 世界，一切都是文件。

一、修改配置文件

```
sudo vi /boot/config.txt
```

（注：运行 `sudo raspi-config` 实际上也是修改这个文件，例如设置 Advanced Options -> I2C 启动 i2C 内核驱动, 就是修个 `dtparam=i2c_arm=on` 这一行）

在 `/boot/config.txt` 文件后面添加下面这一句，这一句就是树莓派添加 Device Tree 设备，`dtoverlay=w1-gpio-pullup, gpioin=4` 表示添加单总线设备，`gpioin=4` 默认管脚为 4，如果 DS18B20 接到其他管脚则需要修改这个值，Pioneer 600 扩展板 DS18B20 默认接到 4，故不用修改。（注：管脚为 BCM 编号）

```
dtoverlay=w1-gpio-pullup, gpioin=4
```

```
53 # Additional overlays and parameters are documented /boot/overlays/README
54 dtoverlay=w1-gpio-pullup,gpioin=4
55 dtparam=spi=on
56 dtparam=i2c_arm=on
```

在 `/boot/overlays/README` 中有关于树莓派 Device Tree 的详细介绍，在其中我们找到下面关于 `w1-gpio-pullup` 设备的介绍如下图。

```
398 File:    w1-gpio-pullup-overlay.dtb
399 Info:    Configures the w1-gpio Onewire interface module.
400         Use this overlay if you *do* need a GPIO to drive an external pullup.
401 Load:    dtoverlay=w1-gpio-pullup, <param>=<val>
402 Params:   gpioin      GPIO for 1-wire (default "4")
403
404         pullup        Non-zero, "on", or "y" to enable the parasitic
405                        power (2-wire, power-on-data) feature
406
407         extpullup      GPIO for external pullup (default "5")
```

二、查看模块是否启动

重启树莓派是设置生效，运行 `lsmod` 命令，如果发现红色方框的两个模块说明模块已启动。

如果没有发现，也可以运行如下命令加载模块

```
sudo modprobe w1_gpio
```

```
sudo modprobe w1_therm
```

```
pi@raspberrypi ~$ lsmod
Module                  Size  Used by
cfg80211                386508  0
rfkill                  16651  1 cfg80211
i2c_dev                 6027  0
snd_bcm2835             18649  0
snd_pcm                 73475  1 snd_bcm2835
snd_seq                 53078  0
snd_seq_device          5628  1 snd_seq
snd_timer               17784  2 snd_pcm,snd_seq
snd                     51038  5 snd_bcm2835,snd_pcm,snd_seq,snd_seq_device
i2c_bcm2708             4990  0
spi_bcm2708             5137  0
w1_therm                2559  0
w1_gpio                 3465  0
wire                    25680  2 w1_gpio,w1_therm
cn                      4636  1 wire
uio_pdrv_genirq         2958  0
uio                     8119  1 uio_pdrv_genirq
pi@raspberrypi ~$ |
```

三、 读取温度

如果没有问题，在 `/sys/bus/w1/devices` 中发现一个 `28-xxxx` 开头的文件夹，这个就是 DS18B20 的 ROM，每个 DS18B20 都一样，在这个文件夹中读取 `w1_slave` 文件则会返回当前温度值。操作如下图：

```
sudo modprobe w1-gpio
sudo modprobe w1-therm
cd /sys/bus/w1/devices
cd 28-00000xxx
cat w1_slave
```

```
pi@raspberrypi ~$ sudo modprobe w1-gpio
pi@raspberrypi ~$ sudo modprobe w1-therm
pi@raspberrypi ~$ cd /sys/bus/w1/devices/
pi@raspberrypi /sys/bus/w1/devices $ ls
28-00000674869d w1_bus_master1
pi@raspberrypi /sys/bus/w1/devices $ cd 28-00000674869d
pi@raspberrypi /sys/bus/w1/devices/28-00000674869d $ ls
driver id name subsystem uevent w1_slave
pi@raspberrypi /sys/bus/w1/devices/28-00000674869d $ cat w1_slave
e8 01 4b 46 7f ff 08 10 97 : crc=97 YES
e8 01 4b 46 7f ff 08 10 97 t=30500
pi@raspberrypi /sys/bus/w1/devices/28-00000674869d $
```

返回数据中，第一行最后的 YRS 表示 CRC 校验成功，数据有效。第二行最后 `t=30500` 表示当前温度为 30.5 摄氏度。

如果接多个 DS18B20，将会看到多个 `28-xxxx` 的文件，分别对应各个 DS18B20。

四、软件编程

1、sysfs

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <dirent.h>
#include <string.h>
#include <time.h>

int main(int argc, char *argv[])
{
    char path[50] = "/sys/bus/w1/devices/";
    char rom[20];
    char buf[100];
    DIR *dirp;
    struct dirent *direntp;
    int fd = -1;
    char *temp;
    float value;

    system("sudo modprobe w1-gpio");
    system("sudo modprobe w1-therm");
    if((dirp = opendir(path)) == NULL)
    {
        printf("opendir error\n");
        return 1;
    }

    while((direntp = readdir(dirp)) != NULL)
    {
        if(strstr(direntp->d_name,"28-00000"))
        {
            strcpy(rom,direntp->d_name);
            printf(" rom: %s\n",rom);
        }
    }
    closedir(dirp);

    strcat(path,rom);
```

```
strcat(path, "/w1_slave");
while(1)
{
    if((fd = open(path, O_RDONLY)) < 0)
    {
        printf("open error\n");
        return 1;
    }

    if(read(fd, buf, sizeof(buf)) < 0)
    {
        printf("read error\n");
        return 1;
    }

    temp = strchr(buf, 't');
    sscanf(temp, "t=%s", temp);
    value = atof(temp)/1000;
    printf(" temp : %3.3f °C\n", value);

    sleep(1);
}
return 0;
}</time.h></string.h></dirent.h></fcntl.h></unistd.h></stdlib.h></stdio.h>
```

编译并执行,结果如图

```
gcc -Wall ds18b20.c -o ds18b20
```

```
sudo ds18b20
```

```
pi@raspberrypi ~/Pioneer600/DS18B20/fs $ sudo ./ds18b20
rom: 28-00000674869d
temp : 29.000 °C
temp : 29.062 °C
temp : 29.125 °C
temp : 29.187 °C
temp : 29.187 °C
temp : 29.187 °C
```



注: (1) `system("sudo modprobe w1-gpio");system("sudo modprobe w1-therm");`在程序的开头运行了一下 `modprobe` 命令

(2) `dirp = opendir(path)` 打开 `/sys/bus/w1/devices/` 文件路径

(3) `direntp = readdir(dirp)` 读取当前路径下的文件或文件夹

(4) `strstr(direntp->d_name, "28-00000")`

查找 28-00000 开头的文件, `strstr` 为字符串操作函数, 上面这条语句表示文件名字是否包含字符串“28-00000”, 如果匹配则返回第一次匹配的地址, 没有搜索到则返回 `NULL`.

(5) `strcpy(rom, direntp->d_name);` `strcpy` 为字符串复制函数。将包含 28-00000 的文件名复制到 `rom` 字符串

(6) `strcat(path, rom);` `strcat(path, "/w1_slave");` `strcat` 为字符串连接函数, 此时 `path` 的值为 `/sys/bus/w1/devices/28-00000xxxx/w1_slave`

(7) `fd = open(path, O_RDONLY); read(fd, buf, sizeof(buf))` 打开文件并读取数据

(8) `temp = strchr(buf, 't');` 查找字符‘t’第一次出现的位置,

(9) `sscanf(temp, "t=%s", temp);` `sscanf` 函数是从一个字符串中读进与指定格式相符的数据, 此处为从第二行数据中扫描出温度值

(10) `value = atof(temp)/1000;` `atof` 函数把字符串转化为浮点数。

2、python

```
import os
import glob
import time

os.system('modprobe w1-gpio')
os.system('modprobe w1-therm')

base_dir = '/sys/bus/w1/devices/'
device_folder = glob.glob(base_dir + '28*')[0]
device_file = device_folder + '/w1_slave'

def read_rom():
    name_file=device_folder+'/name'
    f = open(name_file, 'r')
    return f.readline()

def read_temp_raw():
    f = open(device_file, 'r')
    lines = f.readlines()
    f.close()
```



```

    return lines

def read_temp():
    lines = read_temp_raw()
    while lines[0].strip()[-3:] != 'YES':
        time.sleep(0.2)
        lines = read_temp_raw()
    equals_pos = lines[1].find('t=')
    if equals_pos != -1:
        temp_string = lines[1][equals_pos+2:]
        temp_c = float(temp_string) / 1000.0
        temp_f = temp_c * 9.0 / 5.0 + 32.0
        return temp_c, temp_f

print(' rom: ' + read_rom())
while True:
    print(' C=%3.3f  F=%3.3f'% read_temp())
    time.sleep(1)

```

运行程序,运行结果如图

```
sudo python ds18b20.py
```



```

pi@raspberrypi ~/Pioneer600/DS18B20/python $ sudo python ds18b20.py
rom: 28-00000674869d

C=29.875  F=85.775
C=29.875  F=85.775
C=29.937  F=85.887
C=29.937  F=85.887
C=30.000  F=86.000

```

注：（1）程序的开头运行了一下 modprobe 命令

```
device_folder = glob.glob(base_dir + '28*')[0]
```

```
device_file = device_folder + '/w1_slave'
```

定义设备文件夹和设备文件，glob.glob（base_dir + '28*'）函数为获得 base_dir 路径下所有 28 开头的文件。

（3）while lines[0].strip()[-3:] != 'YES': 判断 w1-value 第一行的最后三个字符是否为‘YES’

（4）equals_pos = lines[1].find('t=') 查找第二行中‘t=’出现的位置

(5) `temp_string = lines[1][equals_pos+2:]` 取温度数据

总结：对比上面两个程序，我们可以发现 `python` 程序更加简单方便。

树莓派系列教程 15：红外遥控

上一章我们介绍了如果通过树莓派 `device tree`, 将在 `ds18b20` 添加到 `linux` 系统中, 并通过命令行读取温度数据, 这一章我们也通过 `device tree` 添加红外接收

`lirc` 为 `linux` 系统中红外遥控的软件, 树莓派系统已经有这个模块, 我们只需设置一下就而已使用。

```
sudo vi /boot/config.txt
```

在文件后面添加下面这一行

```
dtoverlay=lirc-rpi,gpio_in_pin=18
```

红外默认输出是 18 管脚, 如果红外接收头接到其他管脚则需修改对应管脚, (管脚为 BCM 编码), Pioneer 600 接收头默认接到 18 管脚故只需要添加

```
dtoverlay=lirc-rpi
```

在 `/boot/overlay/README` 文件中我们可以找到详细说明。

```
64 This causes the file /boot/overlays/lirc-rpi-overlay.dtb to be loaded. By
65 default it will use GPIOs 17 (out) and 18 (in), but this can be modified using
66 DT parameters:
67
68     dtoverlay=lirc-rpi,gpio_out_pin=17,gpio_in_pin=18
69
```

安装 `lirc` 软件

```
sudo apt-get install lirc
```

运行 `lsmod` 命令查看设备是否已启动, 如若没有找到可运行 `sudo modprobe lirc_rpi` 加载驱动。

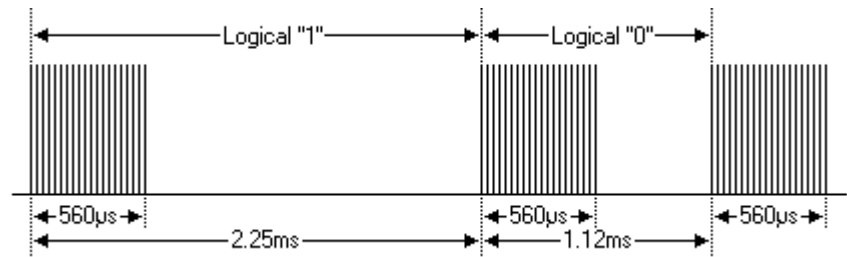
```
pi@raspberrypi / $ lsmod
Module                  Size  Used by
cfg80211                386508  0
rfkill                  16651  1 cfg80211
i2c_dev                 6027  0
snd_bcm2835             18649  3
snd_pcm                 73475  1 snd_bcm2835
snd_seq                 53078  0
snd_seq_device          5628  1 snd_seq
snd_timer               17784  2 snd_pcm, snd_seq
snd                     51038  11 snd_bcm2835, snd_timer, snd_pcm, snd_seq, snd_seq_device
wl_therm                2559  0
spi_bcm2708             5137  0
i2c_bcm2708             4990  0
wl_gpio                 3465  0
lirc_rpi                6646  0
wire                    25680  2 wl_gpio, wl_therm
cn                      4636  1 wire
lirc_dev                8181  1 lirc_rpi
uio_pdrv_genirq         2958  0
rc_core                 16932  1 lirc_dev
uio                     8119  1 uio_pdrv_genirq
pi@raspberrypi / $
```

运行 `sudo mode2 -d /dev/lirc0`, 按遥控上任何键, 查看是否接到类似脉冲。

```
pi@raspberrypi / $ sudo mode2 -d /dev/lirc0
space 16777215
pulse 9085
space 4559
pulse 558
space 578
pulse 539
space 606
pulse 541
space 581
pulse 567
space 579
pulse 538
space 609
pulse 539
space 581
pulse 566
space 579
pulse 538
```

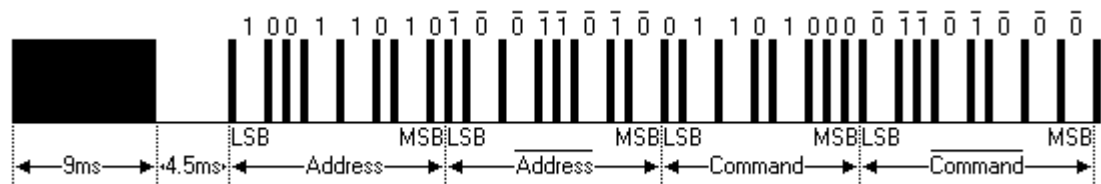
如有接到脉冲测 lirc 正常使用。

采用脉宽调制的串行码, 以脉宽为 0.565ms、间隔 0.56ms、周期为 1.125ms 的组合表示二进制的"0"; 以脉宽为 0.565ms、间隔 1.685ms、周期为 2.25ms 的组合表示二进制的"1"



协议:

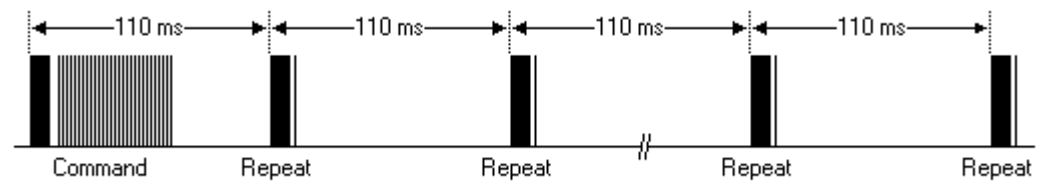
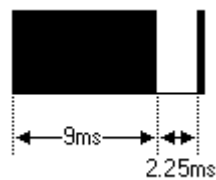
上述“0”和“1”组成的 32 位二进制码经 38kHz 的载频进行二次调制以提高发射效率，达到降低电源功耗的目的。然后再通过红外发射二极管产生红外线向空间发射，如下图。



| 引导码 | 用户识别码 | 用户识别码反码 | 操作码 | 操作码反码 |

一个命令只发送一次，即使遥控器上的按键一直按着。但是会每 110ms 发送一次代码，直到遥控器按键释放。

重复码比较简单：一个 9ms 的 AGC 脉冲、2.25ms 间隔、560uS 脉冲。



bcm2835 程序:

[代码]php 代码:

```
#include <bcm2835.h>
#include <stdio.h>
#define PIN 18
#define IO bcm2835_gpio_lev(PIN)
unsigned char i,idx,cnt;
unsigned char count;
unsigned char data[4];

int main(int argc, char **argv)
{
    if (!bcm2835_init())return 1;
    bcm2835_gpio_fsel(PIN, BCM2835_GPIO_FSEL_INPT);
    bcm2835_gpio_set_pud(PIN, BCM2835_GPIO_PUD_UP);
    printf("IRM Test Program ... \n");

    while (1)
    {
        if(IO == 0)
        {
            count = 0;
            while(IO == 0 && count++ < 200)    //9ms
                delayMicroseconds(60);

            count = 0;
            while(IO == 1 && count++ < 80)      //4.5ms
                delayMicroseconds(60);

            idx = 0;
            cnt = 0;
            data[0]=0;
            data[1]=0;
            data[2]=0;
            data[3]=0;
            for(i =0;i<32;i++)
            {
                count = 0;
                while(IO == 0 && count++ < 15) //0.56ms
                    delayMicroseconds(60);

                count = 0;
                while(IO == 1 && count++ < 40) //0: 0.56ms; 1: 1.69ms
                    delayMicroseconds(60);
```

```

        if (count > 25) data[idx] |= (1<<cnt); if(cnt==" 7")="" {=""
cnt="0;" idx++;="" }="" else="" cnt++;="" if(data[0]+data[1]==" 0xff="" &&=""
data[2]+data[3]=="0xFF)" check="" printf("get="" the="" key:=""
0x%02x\n",data[2]);="" bcm2835_close();="" return="" 0;="" <=""
pre=""></cnt);></stdio.h></bcm2835.h><cnt); if(cnt==" 7")="" {="" cnt="0;"
idx++;="" }="" else="" cnt++;="" if(data[0]+data[1]==" 0xff="" &&=""
data[2]+data[3]=="0xFF)" check="" printf("get="" the="" key:=""
0x%02x\n",data[2]);="" bcm2835_close();="" return="" 0;="" }<=""
pre=""></cnt);>


```

编译并执行，按下遥控按键，终端会显示接收到按键的键值。

```

gcc -Wall irm.c -o irm -lbcm2835
sudo ./irm

```



```

pi@raspberrypi /Pioneer3/IRM/bcm2835 $ sudo ./irm
irm test start:
Get the key: 0x0c
Get the key: 0x0c
Get the key: 0x1c
Get the key: 0x5a
Get the key: 0x52
Get the key: 0x08

```

python 程序 v

```

#!/usr/bin/python
# -*- coding:utf-8 -*-
import RPi.GPIO as GPIO
import time

PIN = 18;

GPIO.setmode(GPIO.BCM)
GPIO.setup(PIN,GPIO.IN,GPIO.PUD_UP)
print('IRM Test Start ...')
try:
    while True:
        if GPIO.input(PIN) == 0:
            count = 0
            while GPIO.input(PIN) == 0 and count < 200: #9ms
                count += 1
                time.sleep(0.00006)
            count = 0

```

```

while GPIO.input(PIN) == 1 and count < 80: #4.5ms
    count += 1
    time.sleep(0.00006)

idx = 0
cnt = 0
data = [0,0,0,0]
for i in range(0,32):
    count = 0
    while GPIO.input(PIN) == 0 and count < 15: #0.56ms
        count += 1
        time.sleep(0.00006)


    count = 0
    while GPIO.input(PIN) == 1 and count < 40: #0: 0.56mx #1: 1.69ms
        count += 1
        time.sleep(0.00006)

    if count > 8:
        data[idx] |= 1>>cnt
    if cnt == 7:
        cnt = 0
        idx += 1
    else:
        cnt += 1
    if data[0]+data[1] == 0xFF and data[2]+data[3] == 0xFF: #check
        print("Get the key: 0x%02x" %data[2])
except KeyboardInterrupt:
GPIO.cleanup();

```

执行，按下遥控按键，终端会显示接到按键的键值。

```
sudo python irm.py
```



```

pi@raspberrypi ~/Pioneer600/IRM/python $ sudo python irm.py
IRM Test Start ...
Get the key: 0x0c
Get the key: 0x0c
Get the key: 0x18
Get the key: 0x5e
Get the key: 0x1c
Get the key: 0x42
Get the key: 0x52
|

```


树莓派系列教程 16：RTC

树莓派本身没有 RTC 功能，若树莓派不联网则无法从网络获取正确时间，Pioneer 600 扩展板上带有高精度 RTC 时钟 DS3231 芯片，可解决这个问题。

一、配置 RTC

1、 修改配置文件

```
sudo vi /boot/config.txt
```

添加 RTC 设备 ds3231

```
dtoverlay=i2c-rtc,ds3231
```

重启树莓派生效设置，开机后可以运行 `lsmod` 命令查看时候有 `rtc-1307` 模块。

（注：ds3231 为 i2c 控制，故应打开树莓派 I2C 功能）

2、 读取 RTC 时钟，

```
sudo hwclock -r
```

读取系统时间

```
date
```

3、 设置 RTC 时间

```
sudo hwclock -set -date="2015/08/12 18:00:00"
```

4、 更新 RTC 时间到系统

```
sudo hwclock -s
```

5、 读取 RTC 时间及系统时间

```
sudo hwclock -r;date
```

```

pi@raspberrypi ~$ sudo hwclock -r
2015年08月12日 星期三 17时52分20秒 -0.996846 seconds
pi@raspberrypi ~$ date
2015年 08月 12日 星期三 17:52:23 CST
pi@raspberrypi ~$ sudo hwclock --set --date="2015/08/12 18:00:00"
pi@raspberrypi ~$ sudo hwclock -r,date
2015年08月12日 星期三 18时00分11秒 -0.182734 seconds
2015年 08月 12日 星期三 17:53:29 CST
pi@raspberrypi ~$ sudo hwclock -s
pi@raspberrypi ~$ sudo hwclock -r,date
2015年08月12日 星期三 18时03分08秒 -0.600901 seconds
2015年 08月 12日 星期三 18:03:08 CST
pi@raspberrypi ~$

```

二、编程控制

我们也可以通过 I2C 编程读写 RTC 时间，运行 `i2cdetect -y 1` 命令我们可以看到下图，我们发现 `ds3231` 的 i2c 地址 `0x68` 的位置显示 `UU`，此时 `ds3231` 作为树莓派的硬件时钟，不能通过 i2c 编程控制，必须将刚才配置文件中的设置注释掉才能用。

```

pi@raspberrypi ~$ i2cdetect -y 1
   0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:  -- -- -- -- -- -- -- -- -- -- -- -- -- --
10:  -- -- -- -- -- -- -- -- -- -- -- -- -- --
20:  20 -- -- -- -- -- -- -- -- -- -- -- -- -- --
30:  -- -- -- -- -- -- -- -- -- -- -- -- -- --
40:  -- -- -- -- -- -- -- -- -- -- -- -- -- --
50:  -- -- -- -- -- -- -- -- -- -- -- -- -- --
60:  -- -- -- -- -- -- -- -- UU -- -- -- -- --
70:  -- -- -- -- -- -- -- -- 77 -- -- -- -- --
pi@raspberrypi ~$

```

```
sudo vi /boot/config.txt
```

找到刚才的设置，在前面加 '#' 注释掉

```
#dtoverlay=i2c-rtc,ds3231
```

重启后再运行 `i2cdetect -y 1` 此时发现 `ds3231` 可以通过 i2c 编程控制

```

pi@raspberrypi ~$ i2cdetect -y 1
   0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:  -- -- -- -- -- -- -- -- -- -- -- -- -- --
10:  -- -- -- -- -- -- -- -- -- -- -- -- -- --
20:  20 -- -- -- -- -- -- -- -- -- -- -- -- -- --
30:  -- -- -- -- -- -- -- -- -- -- -- -- -- --
40:  -- -- -- -- -- -- -- -- 48 -- -- -- -- --
50:  -- -- -- -- -- -- -- -- -- -- -- -- -- --
60:  -- -- -- -- -- -- -- -- 68 -- -- -- -- --
70:  -- -- -- -- -- -- -- -- 77 -- -- -- -- --
pi@raspberrypi ~$

```

1、bcm2835

```
#include <bcm2835.h>
#include <stdio.h>
#include <unistd.h>

//regaddr,seconds,minutes,hours,weekdays,days,months,yeas
char  buf[]={0x00,0x00,0x00,0x18,0x04,0x12,0x08,0x15};
char  *str[] ={"SUN","Mon","Tues","Wed","Thur","Fri","Sat"};
void pcf8563SetTime()
{
    bcm2835_i2c_write(buf,8);
}

void pcf8563ReadTime()
{
    buf[0] = 0x00;
    bcm2835_i2c_write_read_rs(buf ,1, buf,7);
}

int main(int argc, char **argv)
{
    if (!bcm2835_init())return 1;
    bcm2835_i2c_begin();
    bcm2835_i2c_setSlaveAddress(0x68);
    bcm2835_i2c_set_baudrate(10000);
    printf("start.....\n");

    pcf8563SetTime();
    while(1)
    {
        pcf8563ReadTime();
        buf[0] = buf[0]&0x7F; //sec
        buf[1] = buf[1]&0x7F; //min
        buf[2] = buf[2]&0x3F; //hour
        buf[3] = buf[3]&0x07; //week
        buf[4] = buf[4]&0x3F; //day
        buf[5] = buf[5]&0x1F; //mouth
```

```
//year/month/day
printf("20%02x/%02x/%02x  ",buf[6],buf[5],buf[4]);
//hour:minute/second
printf("%02x:%02x:%02x  ",buf[2],buf[1],buf[0]);
//weekday
printf("%s\n",str[(unsigned char)buf[3]-1]);
bcm2835_delay(1000);
}


bcm2835_i2c_end();
bcm2835_close();

return 0;
}</unistd.h></stdio.h></bcm2835.h>
```

编译并执行

```
gcc -Wall ds3231.c -o ds3231 -lbcm2835
```

```
sudo ./ds3231
```



```
pi@raspberrypi ~/Pioneer600/DS3231/bcm2835 $ sudo ./ds3231
start.....
2015/08/12 18:00:00 Wed
2015/08/12 18:00:01 Wed
2015/08/12 18:00:02 Wed
2015/08/12 18:00:03 Wed
2015/08/12 18:00:04 Wed
2015/08/12 18:00:05 Wed
2015/08/12 18:00:06 Wed
2015/08/12 18:00:07 Wed
```

2、python

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import smbus
import time

address = 0x68
register = 0x00
```

```
#sec min hour week day mout year
NowTime = [0x00,0x00,0x18,0x04,0x12,0x08,0x15]
w = ["SUN","Mon","Tues","Wed","Thur","Fri","Sat"];
#/dev/i2c-1
bus = smbus.SMBus(1)
def ds3231SetTime():
    bus.write_i2c_block_data(address,register,NowTime)

def ds3231ReadTime():
    return bus.read_i2c_block_data(address,register,7);

ds3231SetTime()
while 1:
    t = ds3231ReadTime()
    t[0] = t[0]&0x7F #sec
    t[1] = t[1]&0x7F #min
    t[2] = t[2]&0x3F #hour
    t[3] = t[3]&0x07 #week
    t[4] = t[4]&0x3F #day
    t[5] = t[5]&0x1F #mouth

    print("20%x/%x/%x %x:%x:%x %s" %(t[6],t[5],t[4],t[2],t[1],t[0],w[t[3]-1]))
    time.sleep(1)
```

执行程序

```
sudo python ds3231.py
```



```
pi@raspberrypi ~/Pioneer600/DS3231/python $ sudo python ds3231.py
2015/8/12 18:0:0 Wed
2015/8/12 18:0:1 Wed
2015/8/12 18:0:2 Wed
2015/8/12 18:0:3 Wed
2015/8/12 18:0:4 Wed
2015/8/12 18:0:5 Wed
2015/8/12 18:0:6 Wed
^CTraceback (most recent call last):
  File "ds3231.py", line 29, in <module>
    time.sleep(1)
KeyboardInterrupt
pi@raspberrypi ~/Pioneer600/DS3231/python $ |
```

树莓派系列教程 17: PCF8591 AD/DA

树莓派本身没有 AD/DA 功能，如果树莓派外接模拟传感器，则必须外接 AD/DA 功能扩展板才能用。Pioneer 600 扩展带有 AD/DA 芯片 PCF8591,pcf8591 带 1 通道 8 位 DA,4 通道 8 位 AD, 通过 I2C 控制。

DAC

1、bcm2835 程序

```
#include <bcm2835.h>
#include <stdio.h>
#include <unistd.h>

int main(int argc, char **argv)
{
    char Buf[]={0,0};
    unsigned char value=0;

    if (!bcm2835_init())return 1;
    bcm2835_i2c_begin();
    bcm2835_i2c_setSlaveAddress(0x48);
    bcm2835_i2c_set_baudrate(10000);
    printf("start.....\n");

    while(1)
    {
        Buf[0] = 0x40;
        Buf[1] = value++;
        bcm2835_i2c_write(Buf,2);
        printf("AOUT: %d\n",value);

        bcm2835_delay(20);
    }

    bcm2835_i2c_end();
    bcm2835_close();
    return 0;
}
</unistd.h></stdio.h></bcm2835.h>
```

编译并执行

```
gcc -Wall pcf8591.c -o pcf8591 -lbcm2835
```

```
sudo ./ pcf8591
```

2、Python 程序

```
#!/usr/bin/python
# -*- coding:utf-8 -*-
import smbus
import time

address = 0x48
cmd = 0x40
value = 0

bus = smbus.SMBus(1)
while True:
    bus.write_byte_data(address,cmd,value)
    value += 1
    if value == 256:
        value =0
    print("AOUT:%3d" %value)
    time.sleep(0.01)
```

执行程序

```
sudo python pcf8591
```

3、wiringPi 程序

```
#include <wiringpi.h>
#include <pcf8591.h>
#include <stdio.h>

#define Address 0x48
#define BASE 64
#define A0 BASE+0
#define A1 BASE+1
#define A2 BASE+2
#define A3 BASE+3

int main(void)
{
    unsigned char value;
```

```
wiringPiSetup();
pcf8591Setup(BASE,Address);

while(1)
{
    analogWrite(A0,value);
    printf("AOUT: %d\n",value++);
    delay(20);
}
}
</stdio.h></pcf8591.h></wiringpi.h>
```

编译并执行程序

```
gcc -Wall pcf8591.c -o pcf8591 -lbcm2835 -lwiringPi
```

```
sudo ./ pcf8591
```

ADC

1、bcm2835 程序

```
#include <bcm2835.h>
#include <stdio.h>
#include <unistd.h>

int main(int argc, char **argv)
{
    char Buf[]={0};
    unsigned char i;

    if (!bcm2835_init())return 1;
    bcm2835_i2c_begin();
    bcm2835_i2c_setSlaveAddress(0x48);
    bcm2835_i2c_set_baudrate(10000);
    printf("start.....\n");

    while(1)
    {
        for(i = 0;i < 4;i++)
        {
            Buf[0] = i;
            bcm2835_i2c_write_read_rs(Buf,1,Buf,1);
        }
    }
}
```



```

        bcm2835_i2c_read (Buf,1);
        printf("AIN%d:%5.2f  ",i,(double)Buf[0]*3.3/255);
    }
    printf("\n");
    bcm2835_delay(100);
}

bcm2835_i2c_end();
bcm2835_close();
return 0;
}
</unistd.h></stdio.h></bcm2835.h>

```

编译并执行

```
gcc -Wall pcf8591.c -o pcf8591 -lbcm2835
```

```
sudo ./ pcf8591
```

2、Python 程序

```

#!/usr/bin/python
# -*- coding:utf-8 -*-
import smbus
import time

address = 0x48
A0 = 0x40
A1 = 0x41
A2 = 0x42
A3 = 0x43
bus = smbus.SMBus(1)
while True:
    bus.write_byte(address,A0)
    value = bus.read_byte(address)
    print("AOUT:%1.3f  " %(value*3.3/255))
    time.sleep(0.1)

```

执行程序

```
sudo python pcf8591
```

3、wiringPi 程序

```

#include <wiringpi.h>
#include <pcf8591.h>
#include <stdio.h>

```

```
#define Address 0x48
#define BASE 64
#define A0 BASE+0
#define A1 BASE+1
#define A2 BASE+2
#define A3 BASE+3

int main(void)
{
    int value;
    wiringPiSetup();
    pcf8591Setup(BASE,Address);

    while(1)
    {
        value = analogRead(A0);
        printf("Analog: %dmv\n",value*3300/255);
        delay(1000);
    }
}
</stdio.h></pcf8591.h></wiringpi.h>
```

编译并执行程序

```
gcc -Wall pcf8591.c -o pcf8591 -lwiringPi
```

```
sudo ./ pcf8591
```

树莓派系列教程 18: SPI

一、开启树莓派 spi 功能

```
sudo raspi-config
```

选择 Advanced Options -> SPI -> yes 启动 SPI 内核驱动

运行 lsmod 命令，可以看到 spi 模块已启动

```
pi@raspberrypi ~/Pioneer600/OLED/bcm2835 $ lsmod
Module                  Size      Used by
cfg80211                386508    0
rfkill                  16651     1  cfg80211
i2c_dev                 6027      0
snd_bcm2835             18649     3
snd_pcm                 73475     1  snd_bcm2835
snd_seq                 53078     0
snd_seq_device          5628     1  snd_seq
snd_timer               17784     2  snd_pcm, snd_seq
snd                     51038    11  snd_bcm2835, snd_timer, snd_pcm, snd_seq, snd_seq_device
i2c_bcm2708              4990      0
wl_therm                 2559      0
spi_bcm2708              5137      0
wl_gpio                  3465      0
lirc_rpi                 6646      0
wire                     25680     2  wl_gpio, wl_therm
cn                       4636     1  wire
lirc_dev                 8181     1  lirc_rpi
uio_pdrv_genirq          2958      0
uio                      8119     1  uio_pdrv_genirq
rc_core                  16932     1  lirc_dev
```

在/dev 路径下面，我们可以发现两个 spi 设备

```
pi@raspberrypi /dev $ ls
autofs          loop4           ram12           stdout          tty28           tty49           urandom
block           loop5           ram13           tty             tty29           tty5            vc-cma
btrfs-control   loop6           ram14           tty0            tty3            tty50           vchiq
bus             loop7           ram15           tty1            tty30           tty51           vcio
cachefiles      loop-control    ram2            tty10           tty31           tty52           vc-mem
char            MAKEDEV         ram3            tty11           tty32           tty53           vcs
console         mapper          ram4            tty12           tty33           tty54           vcs1
cpu_dma_latency mem             ram5            tty13           tty34           tty55           vcs2
cuse            memory_bandwidth ram6            tty14           tty35           tty56           vcs3
disk            mmcblk0         ram7            tty15           tty36           tty57           vcs4
fb0             mmcblk0p1       ram8            tty16           tty37           tty58           vcs5
fd              mmcblk0p2       ram9            tty17           tty38           tty59           vcs6
full            net             random          tty18           tty39           tty6            vcsa
fuse            network_latency raw             tty19           tty4            tty60           vcsa1
i2c-1           network_throughput rfkill          tty2            tty40           tty61           vcsa2
input           null            root            tty20           tty41           tty62           vcsa3
kmsg            ppp             shm             tty21           tty42           tty63           vcsa4
lirc0           ptmx            snd             tty22           tty43           tty7            vcsa5
log             pts             sndstat         tty23           tty44           tty8            vcsa6
loop0           ram0            spidev0.0       tty24           tty45           tty9            vcsa
loop1           ram1            spidev0.1       tty25           tty46           ttyAMA0         vchi
loop2           ram10           stderr          tty26           tty47           ttyprintk       xconsole
loop3           ram11           stdin           tty27           tty48           uinput          zero
pi@raspberrypi /dev $
```

spi 管脚如下图红框所示，左边方框的管脚分别为 MOSI MISO SCLK, 左边 CE0,CE1 为两个片选管脚，分别对应上图中的 spidev0.0,spidev0.1 两个设备。对这两个文件读写操作即可控制 spi 设备。

```
pi@raspberrypi /dev $ gpio readall
```

Pi 1						Pi 2					
BCM	wPi	Name	Mode	V	Physical	BCM	wPi	Name	Mode	V	Physical
2	8	3.3v			1	2		5v			
3	9	SDA.1	ALTO	1	3	4		5V			
4	7	SCL.1	ALTO	1	5	6		0v			
		GPIO. 7	IN	1	7	8	1	ALTO	TxD	15	14
		0v			9	10	1	ALTO	RxD	16	15
17	0	GPIO. 0	OUT	0	11	12	1	IN	GPIO. 1	1	18
27	2	GPIO. 2	IN	0	13	14		0v			
22	3	GPIO. 3	IN	0	15	16	0	IN	GPIO. 4	4	23
		3.3v			17	18	0	IN	GPIO. 5	5	24
10	12	MOSI	ALTO	0	19	20	0	IN	GPIO. 6	6	25
9	13	MISO	ALTO	0	21	22	0	IN	CE0	10	8
11	14	SCLK	ALTO	0	23	24	0	IN	CE1	11	7
		0v			25	26	1	IN	SCL.0	31	1
0	30	SDA.0	IN	1	27	28	1	IN	0v		
5	21	GPIO. 21	OUT	1	29	30		IN	GPIO. 26	26	12
6	22	GPIO. 22	IN	1	31	32	0	IN	0v		
13	23	GPIO. 23	IN	0	33	34		IN	GPIO. 27	27	16
19	24	GPIO. 24	OUT	1	35	36	1	OUT	GPIO. 28	28	20
26	25	GPIO. 25	IN	1	37	38	0	IN	GPIO. 29	29	21
		0v			39	40	0	IN			
BCM	wPi	Name	Mode	V	Physical	BCM	wPi	Name	Mode	V	Physical

```
pi@raspberrypi /dev $
```

二、SPI 编程

Pioneer 600 扩展板配备一款 0.96inch,128864 分辨率的 oled 显示屏，通过 SPI 控制，先上显示效果图。由于程序过长，在这里我就不把代码全部贴出来了，只是简单讲解一下 spi 的操作函数。如果各位有兴趣可以下载程序，研究一下。如果程序有什么写得不好的地方，还望各位指正。

1、bcm2835

```
bcm2835_spi_begin();           //启动 spi 接口，设置 spi 对应管脚为复用功能
bcm2835_spi_setBitOrder(BCM2835_SPI_BIT_ORDER_MSBFIRST);    //高位先传输
bcm2835_spi_setDataMode(BCM2835_SPI_MODE0);           //spi 模式 0
bcm2835_spi_setClockDivider(BCM2835_SPI_CLOCK_DIVIDER_128); //分频，
bcm2835_spi_chipSelect(BCM2835_SPI_CS0);               //设置片选
bcm2835_spi_setChipSelectPolarity(BCM2835_SPI_CS0, LOW); //设置片选低电平有效
```

```
uint8_t bcm2835_spi_transfer(uint_t value) 传输一个字节  
void bcm2835_spi_transfernb(char *tbuf,char *rbuf,uint32_t len) 传输 n 字节
```

2、python

安装 spi 库

<https://pypi.python.org/pypi/spidev/3.1>

下载 spidev 库,复制到树莓派,并行运行如下命令解压安装

```
tar -zxvf spidev-3.1.tar.gz
```

```
cd spidev
```

```
sudo python setup.py install
```

安装 imaging 库

```
sudo apt-get install python-imaging
```

python-spidev 的使用方法:

```
import spidev
```

导入库

```
bus=0  
device=0  
spi=SPI.SpiDev(bus,device)
```

打开 spi 设备,此处设备为/dev/spi-decv0.0

```
spi.readbytes(n)
```

从 SPI 设备读取 n 字节

```
spi.writebytes(list of value)
```

将列表的数据写到 SPI 设备

```
spi.xfer(list of values[, speed_hz, delay_usec, bits_per_word])
```

执行 SPI 传输。