

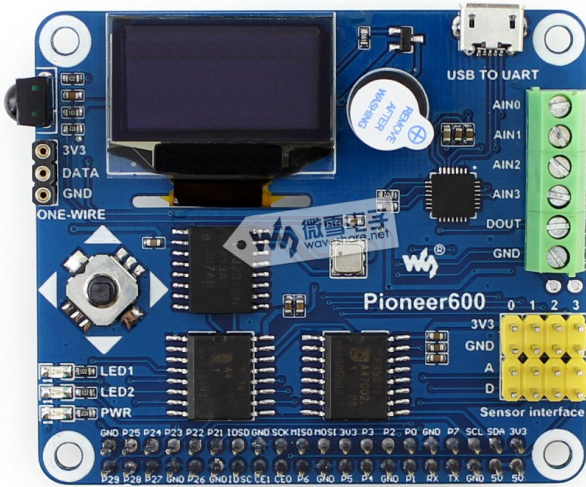
微雪课堂 > 树莓派 > 查看内容

树莓派系列教程8：如何点亮一个LED灯(上)

2015-8-26 18:13 | 发布者: MyMX1213 | 查看: 6322 | 评论: 0 | 原作者: mymx1213

摘要: 本章将简介在树莓派上面通过shell,sysfs,bcm2835,wiringPi,python等不同的编程方式点亮一个LED灯。

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好我千百遍，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

```
1 | 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

```
1 | 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

```
1 | echo 26 > export
```

# 进入GPIO26目录，该目录由上一步操作产生

```
1 | cd gpio26
```

微雪课堂

- 树莓派
- Arduino
- C8051
- PIC
- STM8
- FPGA

树莓派

- 01 Alphabot树莓派教程
- lede是openwrt的一个分
- 有支持pi3
- 02 Alphabot树莓派教程
- 03 树莓派系列教程18：
- 04 树莓派系列教程17：
- 05 树莓派系列教程16：
- 06 树莓派系列教程15：
- 07 树莓派系列教程14：
- 08 树莓派系列教程13：
- 09 树莓派系列教程12：
- 010 树莓派系列教程11
- 011 树莓派系列教程10
- 012 树莓派系列教程9：
- 013 树莓派系列教程8：
- 014 树莓派系列教程8：
- 015 树莓派教程系列7：
- 016 树莓派教程系列6：
- 017 树莓派系列教程5：
- 018 树莓派系列教程4：
- 019 树莓派系列教程3：
- 020 树莓派系列教程3：
- 021 树莓派系列教程3：
- 022 树莓派系列教程2：
- 023 树莓派系列教程1：

微雪课堂 # 运行ls查看gpio26目录中的内容，可查看到如下内容

# active\_low direction edge power subsystem uevent value

```
1 | sudo ls
```

# 设置GPIO26为输出方向

```
1 | echo out > direction
```

# BCM\_GPIO26输出逻辑高电平，LED点亮

```
1 | echo 1 > value
```

# BCM\_GPIO26输出逻辑低电平，LED熄灭

```
1 | echo 0 > value
```

# 返回上一级目录

```
1 | cd ..
```

# 注销GPIO20接口

```
1 | 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

```
pi@raspberrypi ~$ cd /sys/class/gpio
pi@raspberrypi /sys/class/gpio $ ls
export gpiochip0 unexport
pi@raspberrypi /sys/class/gpio $ echo 26 > export
pi@raspberrypi /sys/class/gpio $ ls
export gpio26 gpiochip0 unexport
pi@raspberrypi /sys/class/gpio $ cd gpio26
pi@raspberrypi /sys/class/gpio/gpio26 $ ls
active_low device direction edge subsystem uevent value
pi@raspberrypi /sys/class/gpio/gpio26 $ echo out > direction
pi@raspberrypi /sys/class/gpio/gpio26 $ echo 1 > value
pi@raspberrypi /sys/class/gpio/gpio26 $ echo 0 > value
pi@raspberrypi /sys/class/gpio/gpio26 $ cd ..
pi@raspberrypi /sys/class/gpio $ echo 26 > unexport
pi@raspberrypi /sys/class/gpio $
```

我们可以编写成shell 脚本的形式运行

```
1 | vi led.sh
```

用vi新建led.sh文件，添加如下程序并保存退出。

```
1 | #!/bin/bash
2 | echo Exporting pin $1
3 | echo $1 > /sys/class/gpio/export
4 | echo Setting direction to out.
5 | echo out > /sys/class/gpio/gpio$1/direction
6 | echo Setting pin $2
7 | echo $2 > /sys/class/gpio/gpio$1/value
```

修改文件属性，使文件可执行。

```
1 | 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，执行如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)。

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

```
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。

```
1 | vi led.c
```

使用vi新建led.c文件，添加如下程序并保存退出。

```
001 | #include <sys/stat.h>
002 | #include <sys/types.h>
003 | #include <fcntl.h>
004 | #include <stdio.h>
005 | #include <stdlib.h>
```

```

005 #include <string.h>
006 #include <unistd.h>
007
008 #define TN 0
009
010 #define OUT 1
011
012 #define LOW 0
013 #define HIGH 1
014
015 #define POUT 26
016 #define BUFFER_MAX 3
017 #define DIRECTION_MAX 48
018
019 static int GPIOExport(int pin)
020 {
021     char buffer[BUFFER_MAX];
022     int len;
023     int fd;
024
025     fd = open("/sys/class/gpio/export", O_WRONLY);
026     if (fd < 0) {
027         fprintf(stderr, "Failed to open export for writing!\n");
028         return(-1);
029     }
030
031     len = snprintf(buffer, BUFFER_MAX, "%d", pin);
032     write(fd, buffer, len);
033
034     close(fd);
035     return(0);
036 }
037
038 static int GPIOUnexport(int pin)
039 {
040     char buffer[BUFFER_MAX];
041     int len;
042     int fd;
043
044     fd = open("/sys/class/gpio/unexport", O_WRONLY);
045     if (fd < 0) {
046         fprintf(stderr, "Failed to open unexport for writing!\n");
047         return(-1);
048     }
049
050     len = snprintf(buffer, BUFFER_MAX, "%d", pin);
051     write(fd, buffer, len);
052
053     close(fd);
054     return(0);
055 }
056
057 static int GPIODirection(int pin, int dir)
058 {
059     static const char dir_str[] = "in?out";
060     char path[DIRECTION_MAX];
061     int fd;
062
063     snprintf(path, DIRECTION_MAX, "/sys/class/gpio/gpio%d/direction", pin);
064     fd = open(path, O_WRONLY);
065     if (fd < 0) {
066         fprintf(stderr, "failed to open gpio direction for writing!\n");
067         return(-1);
068     }
069
070     if (write(fd, &dir_str[dir == IN ? 0 : 3], dir == IN ? 2 : 3) < 0) {
071         fprintf(stderr, "failed to set direction!\n");
072         return(-1);
073     }
074
075     close(fd);
076     return(0);
077 }
078
079 static int GPIORead(int pin)
080 {
081     char path[DIRECTION_MAX];
082     char value_str[3];
083     int fd;
084
085     snprintf(path, DIRECTION_MAX, "/sys/class/gpio/gpio%d/value", pin);
086     fd = open(path, O_RDONLY);
087     if (fd < 0) {
088         fprintf(stderr, "failed to open gpio value for reading!\n");
089         return(-1);
090     }
091
092     if (read(fd, value_str, 3) < 0) {
093         fprintf(stderr, "failed to read value!\n");
094         return(-1);
095     }
096
097     close(fd);
098     return(atoi(value_str));
099 }
100
101 static int GPIOWrite(int pin, int value)
102 {
103     static const char s_values_str[] = "01";
104     char path[DIRECTION_MAX];
105     int fd;
106
107     snprintf(path, DIRECTION_MAX, "/sys/class/gpio/gpio%d/value", pin);
108     fd = open(path, O_WRONLY);
109     if (fd < 0) {
110         fprintf(stderr, "failed to open gpio value for writing!\n");
111         return(-1);
112     }
113
114     if (write(fd, &s_values_str[value == LOW ? 0 : 1], 1) < 0) {
115         fprintf(stderr, "failed to write value!\n");
116         return(-1);
117     }

```

微雪课堂

```
close(fd);
return(0);
}
```

```
122
123 int main(int argc, char *argv[])
124 {
125     int i = 0;
126
127     GPIOExport(POUT);
128     GPIODirection(POUT, OUT);
129
130     for (i = 0; i < 20; i++) {
131         GPIOWrite(POUT, i % 2);
132         usleep(500 * 1000);
133     }
134
135     GPIOUnexport(POUT);
136     return(0);
137 }</unistd.h></string.h></stdlib.h></stdio.h></fcntl.h></sys/types.h></sys/stat.h>
```

编译并执行程序

```
1 gcc led.c -o led
2 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。

```
1 vi Makefile
```

使用vi 新建Makefile文件，添加以下代码并保存退出。

```
1 led:led.c
2 gcc led.c -o led
3 clean:
4 rm led
```

运行以下命令即可编译led.c程序

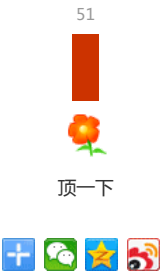
```
1 make
```

运行以下命令即可删除编译产生的可执行文件led

```
1 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>



刚表态过的朋友 (51 人)

邀请 收藏

上一篇：树莓派教程系列6：文件共享(samba)  
下一篇：树莓派系列教程8：如何点亮一个LED灯 ( 下 )

相关阅读

- 树莓派系列教程2：树莓派烧写镜像
- 树莓派系列教程3：访问树莓派之SSH篇
- 树莓派系列教程3：访问树莓派之WiFi篇

- 树莓派教程系列6：文件共享(samba)
- 树莓派教程系列7：wiringPi、bcm2835、python库安装

最新评论

验证码  [换一个](#)

评论

