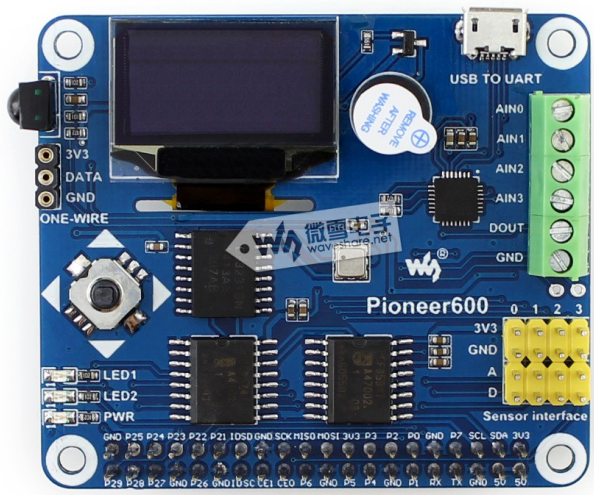
树莓派系列教程8：如何点亮一个LED灯(上)

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
```

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

```
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，添加如下程序并保存退出。

```
#! /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）。

```
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，添加如下程序并保存退出。

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

微雪课堂

```c
#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></sys/stat.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

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
- 树莓派教程系列6：文件共享(samba)
- 树莓派系列教程3：访问树莓派之SSH篇
- 树莓派教程系列7：wiringPi、bcm2835、python库安装
- 树莓派系列教程3：访问树莓派之WiFi篇

最新评论

微雪课堂