

wiringPi 用户手册

制作：Deruio 工作室

时间：2014 年 8 月 24 日

版本：0.0.1

硬件：Raspberry Pi Model B+

软件：Raspbian

版权声明：本文内容翻译自 <http://wiringPi.com>。

若需最新版本，请及时到网站查阅最新版本。

版本号	修订时间	修订内容
-----	------	------

0.0.1	2014.8.24	第一次发布
-------	-----------	-------

目录

1 wiringPi 简介.....	7
1.1 DevLib 简介	7
1.2 PiFace 简介.....	7
1.3 Gertboard 简介.....	8
1.4 wiringPi 扩展.....	8
1.5 wiringPi 的安装.....	9
1.5.1 使用 git 工具.....	9
1.5.2 离线安装	10
1.5.3 测试 wiringPi 是否安装成功	10
2 wiringPi 设置函数	12
2.1 wiringPiSetup 函数.....	12
2.2 wiringPiSetupGpio 函数.....	12
2.3 wiringPiSetupPhys 函数	13
2.4 wiringPiSetupSys 函数.....	13
3 wiringPi 核心函数	13
3.1 pinMode 函数.....	13
3.2 pullUpDnControl 函数	14
3.3 digitalWrite 函数.....	14
3.4 pwmWrite 函数	15
3.5 digitalRead 函数.....	15
3.6 analogRead 函数.....	15

3.7 analogWrite 函数.....	15
4 Raspberry Pi 专用函数	16
4.1 digitalWriteByte 函数	16
4.2 pwmSetMode 函数	16
4.3 pwmSetRange 函数	16
4.4 pwmSetClock 函数	16
4.5 piBoardRev 函数	17
4.6 wpiPinToGpio 函数.....	17
4.7 physPinToGpio 函数	17
4.8 setPadDrive 函数	17
5 I2C 库.....	18
5.1 wiringPiI2CSetup 函数	18
5.2 wiringPiI2CRead 函数.....	19
5.3 wiringPiI2CWrite 函数.....	19
5.4 wiringPiI2CWriteReg8 和 wiringPiI2CWriteReg16 函数.....	19
5.5 wiringPiI2CReadReg8 和 wiringPiI2CReadReg16 函数.....	19
6 SPI 库.....	20
6.1 int wiringPiSPISetup 函数.....	20
6.2 wiringPiSPIDataRW 函数.....	20
7 串口库.....	21
7.1 serialOpen 函数.....	21
7.2 serialClose 函数.....	21

7.3 serialPutchar 函数.....	21
7.4 serialPuts 函数.....	22
7.5 serialPrintf 函数.....	22
7.6 serialDataAvail 函数.....	22
7.7 serialGetchar 函数.....	22
7.8 serialFlush 函数.....	22
7.9 高级串口控制.....	23
8 软件 PWM 库.....	23
8.1 softPwmCreate 函数.....	24
8.2 softPwmWrite 函数.....	24
9 时间函数.....	25
9.1 millis 函数.....	25
9.2 micros 函数.....	25
9.3 delay 函数.....	26
9.4 delayMicroseconds 函数.....	26
10 优先级/时间/线程.....	26
10.1 piHiPri 函数.....	26
10.2 waitForInterrupt 函数.....	27
10.3 wiringPiISR 函数.....	28
10.4 piThreadCreate 函数.....	28
10.5 piLock 和 piUnlock 函数.....	29
11 转换库.....	30

11.1 shiftIn 函数30

11.2 shiftOut 函数30

12 软件音频库30

12.1 softToneCreate 函数31

12.2 softToneWrite 函数31

1 wiringPi 简介

wiringPi 库是由 Gordon Henderson 所编写并维护的一个用 C 语言写成的类库。起初，主要是作为 BCM2835 芯片的 GPIO 库。而现在，已经非常丰富，除了 GPIO 库，还包括了 I2C 库、SPI 库、UART 库和软件 PWM 库等。

由于其与 Arduino 的 “wiring” 系统较为类似，故以此命名。它是采用 GNU LGPLv3 许可证的，可以在 C 或 C++ 上使用，而且在其他编程语言上也有对应的扩展。

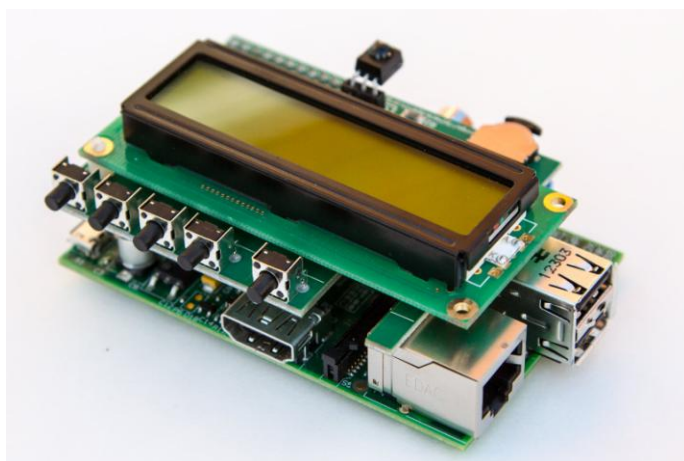
wiringPi 库包含了一个命令行工具 gpio，它可以用来设置 GPIO 管脚，可以用来读写 GPIO 管脚，甚至可以在 Shell 脚本中使用来达到控制 GPIO 管脚的目的。

1.1 DevLib 简介

DevLib 是 wiringPi 的一部分，它可以让您更方便的使用一些流行的外部设备。它支持的设备包括基于日立 HD44780U 芯片的 LCD 显示屏和通用 12864H 驱动芯片的 128x64 像素的显示屏。同时，也支持 DS1302 RTC 时钟芯片、基于 Maxdetect 芯片（如 RHT003）的 Gertboard 板和 PiFace 接口板等等。

1.2 PiFace 简介

wiringPi 对 PiFace 板子提供完全的支持，这里所讲的 PiFace 板子是 PiFace 控制和显示板，如下图：



通过使用 PiFace 板子，你可以不使用键盘和鼠标，仅通过遥控器即可控制树莓派。此外，PiFace 的官网还提供其他的树莓派外围设备，具体可以访问 <http://www.piface.org.uk/>。

1.3 Gertboard 简介

Gertboard 是由 Gert van Loo 所开发的一款树莓派的扩展板，它可以检测和输出模拟电压、驱动电机和检测按键等。

1.4 wiringPi 扩展

Gadgetoid 提供了 Ruby、Perl 和 Python 的扩展，地址：

<https://github.com/WiringPi>

Jeroen Kranssen 提供了 Java 的扩展，地址：

<https://github.com/jkranssen/framboos>

Dave Boulton 提供了 TCL 的扩展，地址：

<https://github.com/davidb24v/WiringPi-Tcl>

Pi4J 是另外一个使用 wiringPi 的 Java 扩展，地址：

<https://github.com/Pi4J/pi4j/>

1.5 wiringPi 的安装

1.5.1 使用 git 工具

如果您尚未安装 git 工具,在 Raspbian 系统中,可以执行如下的命令来安装 git 工具:

```
sudo apt-get install git-core
```

如果您安装时,发生了某些错误,可以尝试更新 apt 库,命令如下:

```
sudo apt-get update
```

或者可能需要对系统进行更新,命令如下:

```
sudo apt-get upgrade
```

安装 git 工具之后,就可以使用如下的命令来获取 wiringPi 了:

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

如果您之前已经使用过 git 的 clone 操作了,那么可以直接下载 wiringPi,而不用再次 clone,命令如下:

```
git pull origin
```

上面的命令,将会为您获取到 wiringPi 的最新版本,接下来,就要编译和安装 wiringPi 了,命令如下:

```
cd wiringPi
```

```
./build
```

新的编译脚本将会编译和安装 wiringPi 到系统中,您不需要再进行其他设置。编译脚本使用到了 sudo 命令,所以,如果有必要,您可以在运行编译脚本前,进行检查。

1.5.2 离线安装

访问下面的链接：

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

将会打开如下的页面：



Gordons Projects

→ [Projects](#) [Top-Level GIT](#)

/ [wiringPi](#) / [summary](#)

[summary](#) | [shortlog](#) | [log](#) | [commit](#) | [commitdiff](#) | [tree](#)

description wiringPi - A 'wiring' like library for the Raspberry Pi
owner GIT
last change Fri, 18 Jul 2014 05:23:57 +0800 (22:23 +0100)

shortlog

2014-07-17	Gordon Henderson	Fixed a bug in the gpio readall command on model B...	master	commit	commitdiff	tree	snapshot
2014-07-14	Gordon Henderson	Updated mostly to the gpio readall command to support...		commit	commitdiff	tree	snapshot
2014-06-27	Gordon Henderson	Fixed a small bug in the ISR code where it was looking...		commit	commitdiff	tree	snapshot
2014-06-24	Gordon Henderson	Bumped version to 2.15		commit	commitdiff	tree	snapshot
2014-06-24	Gordon Henderson	Updates for the Rasperry Pi Compute Module - changes...		commit	commitdiff	tree	snapshot
2014-05-20	Gordon Henderson	changed to pin mode to support softPwm.		commit	commitdiff	tree	snapshot
2013-08-03	Gordon Henderson	Added some tweaks to gpio to set alt modes on pins...		commit	commitdiff	tree	snapshot
2013-07-28	Gordon Henderson	Bumped version		commit	commitdiff	tree	snapshot
2013-07-28	Gordon Henderson	It helps if you add the files into GIT...		commit	commitdiff	tree	snapshot
2013-07-28	Gordon Henderson	Minor changes to the files and removed a bit of debug.		commit	commitdiff	tree	snapshot

然后，你会下载到一个.tar.gz 的压缩包，名字可能为 wiringPi-df45388.tar.gz，由于不同的发行版本，后面的 df45388 字符串可能会不同。下载完成后，你可以执行下面的命令来安装 wiringPi：

```
tar xzf wiringPi-df45388.tar.gz
```

```
cd wiringPi-df45388
```

```
./build
```

请注意，实际的文件名可能会不同，因此需要做相对的改变。

1.5.3 测试 wiringPi 是否安装成功

打开命令终端，可以通过 gpio 命令来检查 wiringPi 是否安装成功，运行下面的命令：

```
gpio -v
```

gpio readall

运行第一条命令后，您可能得到如下的输出结果：

```
pi@raspberrypi:~/picode/wiringPi$ gpio -v
gpio version: 2.20
Copyright (c) 2012-2014 Gordon Henderson
This is free software with ABSOLUTELY NO WARRANTY.
For details type: gpio -warranty

Raspberry Pi Details:
Type: Model B+, Revision: 1.2, Memory: 512MB, Maker: Sony
```

运行第二条命令，你可以获取 wiringPi 与树莓派的 GPIO 接口之间的对应关系，如下

图：

```
pi@raspberrypi:~/picode/wiringPi$ gpio readall
```

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

上图中的 Physical 列即代表树莓派物理上接口 J8 的管脚定义，wPi 列即代表在 wiringPi

中所对应的数值，BCM 列即代表在 BCM2835 的 GPIO 寄存器中的偏移地址，即在

BCM2835 C Library 中对应的 GPIO 的数值。

如果上述两条命令，您均能得到类似于上图中的输出结果，证明您的 wiringPi 已经安装成功了。

2 wiringPi 设置函数

有四个函数来对 wiringPi 进行初始化，它们是：

```
int wiringPiSetup(void);
```

```
int wiringPiSetupGpio(void);
```

```
int wiringPiSetupPhys(void);
```

```
int wiringPiSetupSys(void);
```

您的程序在开始需要调用上面的一个函数来进行初始化，否则，您的程序可能不能正常工作。

在 wiringPi 的 v1 版本中，如果无论何种原因，这些函数执行失败，将会返回一个错误代码。v2 版本中，一直会返回 0。再和许多 wiringPi 的用户进行讨论后，很多人并不在意检查返回值，如果这些设置函数返回失败，那么就应该停止程序的执行。

如果您想使用 v1 版本，您需要设置一个环境变量，其名为 WIRINGPI_CODES。它的值可以是任何值，只要这个环境变量存在即可。

2.1 wiringPiSetup 函数

该函数初始化 wiringPi，并假定程序将使用 wiringPi 的管脚定义图。具体管脚映射，可以通过 gpio readall 命令来查看。

该函数需要 root 权限。

2.2 wiringPiSetupGpio 函数

该函数与 wiringPiSetup 函数类似，区别在于假定程序使用的是 Broadcom 的 GPIO 管脚定义，而没有重新映射。

该函数需要 root 权限，需要注意 v1 和 v2 版本的树莓派是不同的。

2.3 wiringPiSetupPhys 函数

该函数与 wiringPiSetup 函数类似，区别在于允许程序使用物理管脚定义，仅支持 P1 接口。

该函数需要 root 权限。

2.4 wiringPiSetupSys 函数

该函数初始化 wiringPi，使用/sys/class/gpio 接口，而不是直接通过操作硬件来实现。该函数可以使用非 root 权限用户，在此种模式下的管脚号是 Broadcom 的 GPIO 管脚号，与 wiringPiSetupGpio 函数类似，需要注意 v1 和 v2 板子的不同。

在此种模式下，在运行程序前，您需要通过/sys/class/gpio 接口导出要使用的管脚。你可以在一个独立的 shell 脚本中来导出将要使用的管脚，或者使用系统的 system()函数来调用 gpio 命令。

3 wiringPi 核心函数

这些函数可以直接在树莓派上使用，也可以在外部 GPIO 模块上使用。并不是所有的模块都支持所有函数，如 PiFace 预配置了固定的输入和输出管脚，并且树莓派板上并没有硬件上的模拟管脚。

3.1 pinMode 函数

该函数的原型为：`void pinMode(int pin, int mode);`

使用该函数可以将某个管脚设置为 INPUT (输入)、OUTPUT (输出)、PWM_OUTPUT (脉冲输出) 或者 GPIO_CLOCK (GPIO 时钟)。

需要注意的是仅有管脚 1 (BCM_GPIO 18) 支持 PWM_OUTPUT 模式，仅有管脚 7 (BCM_GPIO 4) 支持 CLOCK 输出模式。

在 Sys 模式下，这个函数没有影响。你可以通过调用 gpio 命令在 shell 脚本中来设置管脚的模式。

3.2 pullUpDnControl 函数

该函数的原型为：void pullUpDnControl(int pin, int pud);

使用该函数可以设置指定管脚使用上拉或者下拉电阻模式，通常当需要管脚作为输入引脚时，需要设定此项。不同于 Arduino，BCM2835 有内部上拉和下拉电阻，这两种模式。

参数 pud 可以为 PUD_OFF (无上拉或下拉电阻)、PUD_DOWN (内部下拉至地线) 或者 PUD_UP (内部上拉至 3.3V)。在树莓派上，内部上拉和下拉电阻有接近 50KΩ。

该函数在 Sys 模式下无作用。如果你需要激活上拉或下拉电阻的话，在启动程序前，可以通过在脚本中调用 gpio 命令来实现。

3.3 digitalWrite 函数

该函数的原型为：void digitalWrite(int pin, int value);

使用该函数可以向指定的管脚写入 HIGH (高) 或者 LOW (低)，写入前，需要将管脚设置为输出模式。

wiringPi 将任何的非 0 值作为 HIGH (高) 来对待，因此，0 是唯一能够代表 LOW (低) 的数值。

3.4 pwmWrite 函数

该函数的原型为：`void pwmWrite(int pin, int value);`

使用该函数可以将值写入指定管脚的 PWM 寄存器中。树莓派板上仅有一个 PWM 管脚，即管脚 1 (BCM_GPIO 18 , 物理管脚号为 12)。可设置的值为 0~1024，其他 PWM 设备可能有不同的 PWM 范围。

当在 Sys 模式时，该函数不可用来控制树莓派的板上 PWM。

3.5 digitalRead 函数

该函数的原型为：`void digitalRead(int pin);`

使用该函数可以读取指定管脚的值，读取到的值为 HIGH (1) 或者 LOW (0)，该值取决于该管脚的逻辑电平的高低。

3.6 analogRead 函数

该函数的原型为：`int analogRead(int pin);`

该函数返回所指定的模拟输入管脚的值。你需要注册额外的模拟模块来启用该函数，比如 Gertboard，quick2Wire 模拟板等。

3.7 analogWrite 函数

该函数的原型为：`void analogWrite(int pin, int value);`

该函数将指定的值写入到指定的管脚。你需要注册额外的模拟模块来启用该函数，比如 Gertboard 等。

4 Raspberry Pi 专用函数

这些函数并不是 wiringPi 的核心函数集中的函数，但是，特定的适用于树莓派。一些外部硬件驱动模块可能也提供一些与此相类似的函数。

4.1 digitalWriteByte 函数

该函数的原型为：`void digitalWriteByte(int value);`

该函数将一个 8 位的字节写入到前 8 个 GPIO 管脚中。这是一次性设置 8 个管脚的最快的方法，尽管将会花费两个写入操作到树莓派的 GPIO 硬件上。

4.2 pwmSetMode 函数

该函数的原型为：`void pwmSetMode(int mode);`

PWM 产生器可以运行在 “balanced”（平衡）和 “mark : space”（标记和空格）模式，后者是传统的工作模式。然而，树莓派的默认工作模式是 “balanced”，你可以通过将 mode 参数设置为 PWM_MODE_BAL 或者 PWM_MODE_MS 来切换到不同的模式上。

4.3 pwmSetRange 函数

该函数的原型为：`void pwmSetRange(unsigned int range);`

该函数用来设置 PWM 发生器的范围寄存器，它的默认值是 1024.

4.4 pwmSetClock 函数

该函数的原型为：`void pwmSetClock(int divisor);`

该函数用来设置 PWM 时钟的分频值。

需要注意的是，PWM 控制函数在 Sys 模式下，是不可用的。要了解更多关于 PWM 系统的信息，请阅读 Broadcom ARM 外设手册。

4.5 piBoardRev 函数

该函数的原型为：`int piBoardRev(void);`

该函数返回树莓派的硬件版本，可能为 1 或者 2.当从版本 1 到版本 2 时，一些 BCM_GPIO 管脚号可能会有所改变，所以，如果你正在使用 BCM_GPIO 管脚号的话，你需要注意到这些。

4.6 wpiPinToGpio 函数

该函数的原型为：`int wpiPinToGpio(int wPiPin);`

该函数返回所指定的 wiringPi 管脚所对应的 BCM_GPIO 管脚号。需要考虑到不同的版本中的 wiringPi 管脚定义的差别。

4.7 physPinToGpio 函数

该函数的原型为：`int physPinToGpio(int physPin);`

该函数返回指定 P1 接口的物理管脚所对应的 BCM_GPIO 管脚号。

4.8 setPadDrive 函数

该函数的原型为：`void setPadDrive(int group, int value);`

该函数设置指定管脚组的驱动强度。树莓派上共有 3 组管脚组，驱动强度的范围为 0~7。

不用使用该函数，除非你知道为什么要设置驱动强度。

5 I2C 库

wiringPi 包含了一个 I2C 库，来让您能够更轻松的使用树莓派的板上 I2C 接口。在使用 I2C 接口之前，您可能需要使用 gpio 命令来加载 I2C 驱动到内核中：

```
gpio load i2c
```

如果你需要的波特率不是 100Kbps，那么您可以使用如下命令设置波特率为 1000Kbps：

```
gpio load i2c 1000
```

使用 I2C 库，需要包含 wiringPiI2C.h 文件。并且编译时，同样需要使用 -lwiringPi 来连接到 wiringPi 库。

您仍然可以使用标准的系统命令来检测 I2C 设备，如 i2cdetect 命令，需要注意的是，在 v1 版本的树莓派上是 0，v2 版本上是 1，如下：

```
i2cdetect -y 0 #Rev 1
```

```
i2cdetect -y 1 #Rev 2
```

当然，您也可以使用 gpio 命令来调用 i2cdetect 命令，从而检测 I2C 设备，这样就不用在乎您的树莓派版本了，如下：

```
gpio i2cdetect
```

5.1 wiringPiI2CSetup 函数

该函数的原型为：`int wiringPiI2CSetup(int devId);`

该函数使用指定设备标示号来初始化 I2C 系统。参数 devId 是 I2C 设备的地址，可以通过 i2cdetect 命令可以查到该地址。该函数会获取树莓派的版本并依据此打开 /dev 目录下对应的设备。

返回值是标准的 Linux 文件句柄，如果有错误，则返回-1。

比如，流行的 MCP23017 GPIO 扩展器的设备 ID 是 0x20，所以，你需要将这个数值传递给 wiringPiI2CSetup()。

5.2 wiringPiI2CRead 函数

该函数的原型为：int wiringPiI2CRead(int fd)；

简单的设备读操作。一些设备可以直接读取，而不需要发送任何寄存器地址。

5.3 wiringPiI2CWrite 函数

该函数的原型为：int wiringPiI2CWrite(int fd, int data)；

简单的设备写操作。一些设备可以接受数据，而不需要发送任何内部寄存器地址。

5.4 wiringPiI2CWriteReg8 和 wiringPiI2CWriteReg16 函数

该函数的原型为：

```
int wiringPiI2CWriteReg8(int fd, int reg, int data);
```

```
int wiringPiI2CWriteReg16(int fd, int reg, int data);
```

使用这两个函数，可以写一个 8 位或 16 位数值到指定的设备寄存器。

5.5 wiringPiI2CReadReg8 和 wiringPiI2CReadReg16 函数

该函数的原型为：

```
int wiringPiI2CReadReg8(int fd, int reg);
```

```
int wiringPiI2CReadReg16(int fd, int reg);
```

使用这两个函数，可以从指定的设备寄存器读取一个 8 位或 16 位的数值。

6 SPI 库

wiringPi 库包含一个更易使用的 SPI 库，来帮助您使用树莓派上的板上 SPI 接口。

在使用 SPI 接口前，你需要使用 gpio 命令来加载 SPI 驱动到内核中：

```
gpio load spi
```

如果您需要的缓冲区大于 4KB,需要在命令行进行指定缓冲区的大小,单位是 KB:

```
gpio load spi 100
```

上述命令将会分配 100KB 的缓冲区。(您可能很少需要改变这项设置,默认值对于绝大多数应用程序来说已经足够了)。

为了使用 SPI 库,你也需要在你的程序中添加如下语句:

```
#include <wiringPiSPI.h>
```

程序在编译连接时，仍然需要添加-lwiringPi 选项。

6.1 int wiringPiSPISetup 函数

该函数的原型为：int wiringPiSPISetup(int channel, int speed);

使用该函数可以初始化一个 SPI 通道，树莓派有两个 SPI 通道（0 和 1）。speed 参数是一个整数值，其范围为 500000~32000000，代表 SPI 时钟速度，单位是 Hz。

返回值是一个标准的 Linux 设备文件描述符，若返回值为-1，则失败。若失败，可以使用标准 errno 全局变量来查看失败原因。

6.2 wiringPiSPIDataRW 函数

该函数的原型为：int wiringPiSPIDataRW(int channel, unsigned char* data, int len) ;

该函数执行一个同时读写操作，通过选定的 SPI 总线。缓冲区中的数据，将会被 SPI 总线的返回数据所覆盖。若需要简单的读写操作，可以使用标准的系统函数：read()和 write()。

7 串口库

wiringPi 包含了一个简单的串口处理库。使用它可以操作板上的串口或者 USB 串口设备，这两者并无特殊的差别。只需要在初始化函数中指定设备名称即可。

为了能够正常使用串口库，您的程序必须包含下面的文件：

```
#include <wiringSerial.h>
```

7.1 serialOpen 函数

该函数的原型为：int serialOpen(char* device, int baud);

该函数会初始化并打开串口设备，同时设置通讯的波特率。默认将端口设置为“raw”模式（字符未经过转换），并且读超时为 10 秒。返回值是一个文件描述符，如果失败的话，返回值为-1，这种情况下，将会依据不同的失败原因来设置 errno 变量。

7.2 serialClose 函数

该函数的原型为：void serialClose(int fd);

使用指定的文件描述符关闭设备。

7.3 serialPutchar 函数

该函数的原型为：void serialPutchar(int fd, unsigned char c);

将单个字节写入指定设备的文件描述符。

7.4 serialPuts 函数

该函数的原型为：`void serialPuts(int fd, char* s);`

该函数将一个以 0 结尾的字符串写入指定设备的文件描述符。

7.5 serialPrintf 函数

该函数的原型为：`void serialPrintf(int fd, char* message, ...);`

与系统的 `printf` 函数类似，区别在于将内容写入到了串口设备。

7.6 serialDataAvail 函数

该函数的原型为：`int serialDataAvail(int fd);`

返回等待读取的字符数，如果发生错误，则返回-1，此种情况下，`errno` 将会被设置为错误发生原因。

7.7 serialGetchar 函数

该函数的原型为：`int serialGetchar(int fd);`

返回串口设备的下一个待读取字符。如果没有数据，该函数将会等待 10 秒，10 秒后若仍无数据则会返回-1。

7.8 serialFlush 函数

该函数的原型为：`void serialFlush(int fd);`

抛弃所有已接收的数据或者等待写入指定设备完成。

7.9 高级串口控制

wiringSerial 库的目的是提供简单的控制，对于大多数应用程序来说，已足够。然而，如果您需要更高级的控制，比如校验位等，那么您需要使用旧方法来设置。

例如，需要设置数据位为 7 位、偶校验，那么需要在程序中如下设置：

```
#include <termios.h>
```

在函数中，添加下面的代码：

```
struct termios options;
```

```
tcgetattr(fd, &options);
```

```
options.c_cflag &= ~CSIZE;
```

```
options.c_cflag |= CS7;
```

```
options.c_cflag |= PARENB;
```

```
tcsetattr(fd, &options);
```

上面代码中的变量 fd，即是 serialOpen()函数的返回值。

如果需要更多关于 tcgetattr 的信息，可以使用 man tcgetattr 命令。

8 软件 PWM 库

wiringPi 中包含了一个软件驱动的 PWM 处理库，可以在任意的树莓派 GPIO 上输出 PWM 信号。

但是，也有一些限制。为了维护较低的 CPU 使用率，最小的脉冲宽度是 100 微秒，结合默认的建议值为 100，那么最小的 PWM 频率是 100Hz。如果需要更高的频率，可以使用更低的数值。如果看脉冲宽度的驱动代码，你会发现低于 100 微秒，wiringPi 是在软件

循环中实现的，这就意味着 CPU 使用率将会动态增加，从而使得控制其他管脚成为不可能。

需要注意的是，当其他程序运行在更高的实时的优先级，Linux 可能会影响产生信号的精度。

尽管有这些限制，控制 LED 或电机还是可以的。

使用前，需要包含相应的文件：

```
#include <wiringPi.h>
```

```
#include <softPwm.h>
```

当编译程序时，必须加上 pthread 库，如下：

```
gcc -o myprog myprog.c -lwiringPi -lpthread
```

必须使用 wiringPiSetup()、wiringPiSetupGpio()或者 wiringPiSetupPhys()函数来初始化 wiringPi。wiringPiSetupSys()是不够快的，因此，必须使用 sudo 命令来运行程序。

一些外部扩展模块有可能足够快的处理软件 PWM，在 MCP23S17 GPIO 扩展器上已经测试通过了。

8.1 softPwmCreate 函数

该函数的原型为：`int softPwmCreate(int pin, int initialValue, int pwmRange);`

该函数将会创建一个软件控制的 PWM 管脚。可以使用任何一个 GPIO 管脚，pwmRange 参数可以为 0（关）~100（全开）。

返回值为 0，代表成功，其他值，代表失败。

8.2 softPwmWrite 函数

该函数的原型为：`void softPwmWrite(int pin, int value);`

该函数将会更新指定管脚的 PWM 值。value 参数的范围将会被检查，如果指定的管脚之前没有通过 softPwmCreate 初始化，将会被忽略。

9 时间函数

尽管 Linux 系统提供了很多系统函数来实现定时和休眠，但是，有时使用起来会感到非常容易混淆，特别对于 Linux 新手来说。因此，为了方便使用，特意从 Arduino 平台移植了代码。

需要注意的是，尽管你没有使用任何的输入或输出函数，你仍然需要调用 wiringPi 中的设置函数来进行初始化操作。如果你的程序中不需要 root 权限，你可以仅仅使用 wiringPiSetupSys 函数，不要忘记加上 `#include <wiringPi.h>`。

9.1 millis 函数

该函数的原型为：`unsigned int millis(void);`

该函数返回从调用 wiringPiSetup 函数后的毫秒数。返回值是一个无符号 32 位整数。在 49 天后，将会溢出。

9.2 micros 函数

该函数的原型为：`unsigned int micros(void);`

该函数返回从调用 wiringPiSetup 函数后的微秒数。返回值是一个无符号 32 位整数，在接近 71 分钟后，将会溢出。

9.3 delay 函数

该函数的原型为：`void delay(unsigned int howLong);`

该函数将会中断程序执行至少 `howLong` 毫秒。因为 Linux 是多任务的原因，中断时间可能会更长。需要注意的是，最长的延迟值是一个无符号 32 位整数，其大约为 49 天。

9.4 delayMicroseconds 函数

该函数的原型为：`void delayMicroseconds(unsigned int howLong);`

该函数将会中断程序执行至少 `howLong` 微秒。因为 Linux 是一个多任务的系统，因此中断时间可能会更长。需要注意的是，最长的延迟值是一个无符号 32 位整数，其大约为 71 分钟。

延迟低于 100 微秒，将会使用硬件循环来实现；超过 100 微秒，将会使用系统的 `nanosleep()` 函数来实现。

10 优先级/时间/线程

wiringPi 提供一些函数来允许程序管理优先级和从程序内部启动新线程。线程将会和主程序一同运行，可以用作不同的用途。要了解更多关于线程的知识，可以搜索“Posix Threads”关键字。

10.1 piHiPri 函数

该函数的原型为：`int piHiPri(int priority);`

该函数会将程序（或多线程程序中的线程）移动到一个更高的优先级，并且会开启实时调度功能。

priority 参数可以为 0~99, 0 为默认值, 99 为最大值。更改优先级并不会让程序运行更快, 但是当其他程序正在运行时, 会给予一个更大的时间片。优先级参数可以与其他线程同时工作, 因此, 可以将一个程序设置为优先级 1, 其他的设置为优先级 2, 这与设置一个优先级为 10 和另一个优先级为 90 是同样的效果 (只要没有其他程序有更高的优先级)。

该函数返回 0 代表执行成功, 返回-1 代表执行失败。如果返回错误, 程序可以通过查看全局变量 errno 来得到失败原因。

需要注意的是, 只有作为 root 权限运行的程序才能改变其优先级。如果是非 root 权限, 则没有任何改变。

10.2 waitForInterrupt 函数

该函数的原型为: `int waitForInterrupt(int pin, int timeout);`

当调用该函数后, 程序会一直在指定的管脚等待中断事件的发生。timeout 参数的单位是毫秒, 其值为-1, 代表永远等待。

如果有错误发生, 函数将会返回-1。如果超时, 将会返回 0, 如果等待中断事件成功, 将会返回 1。

在调用 waitForInterrupt 函数前, 首先必须初始化 GPIO 管脚。可以通过在脚本或使用 system() 函数来调用 gpio 命令来实现。

例如, 在 GPIO 管脚 0 上等待一个下降沿中断, 可以使用如下的命令来设置硬件参数:

```
gpio edge 0 falling
```

当然, 上述命令需要在运行程序前执行。

10.3 wiringPiISR 函数

该函数的原型为：`int wiringPiISR(int pin, int edgeType, void (*function)(void));`

该函数会在指定管脚注册一个中断事件的函数，当指定管脚发生中断事件时，会自动调用该函数。`edgeType` 参数可以为 `INT_EDGE_FALLING`（下降沿）、`INT_EDGE_RISING`（上升沿）、`INT_EDGE_BOTH`（上升沿或者下降沿）或者 `INT_EDGE_SETUP`。如果是 `INT_EDGE_SETUP`，将不会初始化该管脚，因为它假定已经在别处设置过该管脚（比如使用 `gpio` 命令），但是，如果指定另外的类型，指定管脚将会被导出并初始化。完成此操作使用的是 `gpio` 命令，所以，必须保证 `gpio` 命令是可用的。

这里的管脚号可以使用的模式有：`wiringPi` 模式、`BCM_GPIO` 模式、物理或 `Sys` 模式。

这个函数在所有的 `GPIO` 模式均可正常工作，并且不需要 `root` 权限。

注册函数在中断触发时，将会被调用。在调用注册函数前，中断事件将会从分配器中清除，所以，即使有后续的触发发生，在处理完之前，也不会错过此次触发。（当然，如果在正在处理触发时，有不止一个的中断发生，已经发生的中断将会被忽略）。

这个函数将会运行在一个高的优先级（如果程序使用 `sudo` 或者 `root` 用户），并且与主程序同时执行。它可以访问全局变量、打开文件句柄等等。

查看 `isr.c` 代码，从而了解如何使用该函数。

10.4 piThreadCreate 函数

该函数的原型为：`int piThreadCreate(name);`

该函数将会创建一个线程，该线程是程序中的一个使用 `PI_THREAD` 声明的函数。该函数与主程序可同时运行。当主程序在执行其他任务时，可能需要等待一个中断来运行该函数。被创建的线程可以用来监听一个事件、或者使用全局变量来返回主程序的一个动作、或者其

他线程。

线程函数需要采用如下的声明方式：

```
PI_THREAD (myThread)

{

    // 在此处添加线程代码，可与主程序一同运行，也可能是一个无限循环

}
```

可以采用如下的方式来在主程序中启动：

```
x = piThreadCreate(myThread);

if(x != 0)

    printf( "it didn' t start" );
```

创建线程的方法与 Linux 的 Posix 线程的创建机制并无不同。

10.5 piLock 和 piUnlock 函数

该函数的原型为：

```
void piLock(int keyNum);

void piUnlock(int keyNum);
```

这两个函数可以使您从主程序中同步变量到程序内的线程中。keyNum 参数可以为 0~3，它代表一个键而已。当其他进程试图锁定同样的键时，它将会被停滞，直到第一个进程解锁同样的键。

你需要使用这些函数来保证当主程序和线程进行数据交换时，能够得到有效的数据，否则，可能当线程在执行数据被复制和改变数据时，数据未复制完整或者无效。

查看 wfi.c 代码，来了解如何使用。

11 转换库

wiringPi 包含了一个简单的转换库，使用该库可以从移位寄存器（如 74x595）移出或移入 8 位数据。

为了使用该库，需要包含如下头文件：

```
#include <wiringPi.h>
```

```
#include <wiringShift.h>
```

11.1 shiftIn 函数

该函数的原型为：`uint8_t shiftIn(uint8_t dPin, uint8_t cPin, uint8_t order);`

该函数将 dPin 上的接收的 8 位数据进行转换，时钟信号将通过 cPin 管脚输出。order 参数可以为 LSBFIRST 或者 MSBFIRST。在 cPin 变高后，数据会重新采样。（cPin 为高，采样数据；cPin 为低，重复 8 位数据）。读取到的 8 位数据将会被作为返回值。

11.2 shiftOut 函数

该函数的原型为：`void shiftOut(uint8_t dPin, uint8_t cPin, uint8_t order, uint8_t val);`

该函数将 8 位数值 val 通过 dPin 管脚发送出去，时钟信号将通过 cPin 管脚输出。order 参数可以为 LSBFIRST 或者 MSBFIRST。数据在上升或下降沿时被输出。

12 软件音频库

wiringPi 包含了一个软件驱动的声音处理库，可以在任意的树莓派的 GPIO 上输出音调

或方波信号。

当然也有一些限制。为了维护一个低的 CPU 使用率，最小脉冲宽度为 100 微秒，即是最大频率是 $1/0.0002=5000\text{Hz}$ 。

为了使用软件音频库，需要添加相应的头文件：

```
#include <wiringPi.h>
```

```
#include <softTone.h>
```

编译时，需要加入 -lpthread 选项，如下：

```
gcc -o myprog myprog.c -lwiringPi -lpthread
```

必须使用 wiringPiSetup()、wiringPiSetupGpio() 或者 wiringPiSetupPhys() 函数中的一个来初始化。wiringPiSetupSys() 函数不够快，因此你必须使用 sudo 命令来运行程序。

12.1 softToneCreate 函数

该函数的原型为：`int softToneCreate(int pin);`

该函数穿件一个软件控制的音频管脚。可以使用任何一个 GPIO 管脚，管脚号需在 wiringPiSetup 函数中初始化。

返回值为 0，代表成功。返回其他值，代表失败。

12.2 softToneWrite 函数

该函数的原型为：`void softToneWrite(int pin, int freq);`

该函数将为更新指定管脚的音频频率值。音频将会一直被播放，直到将其频率设置为 0。