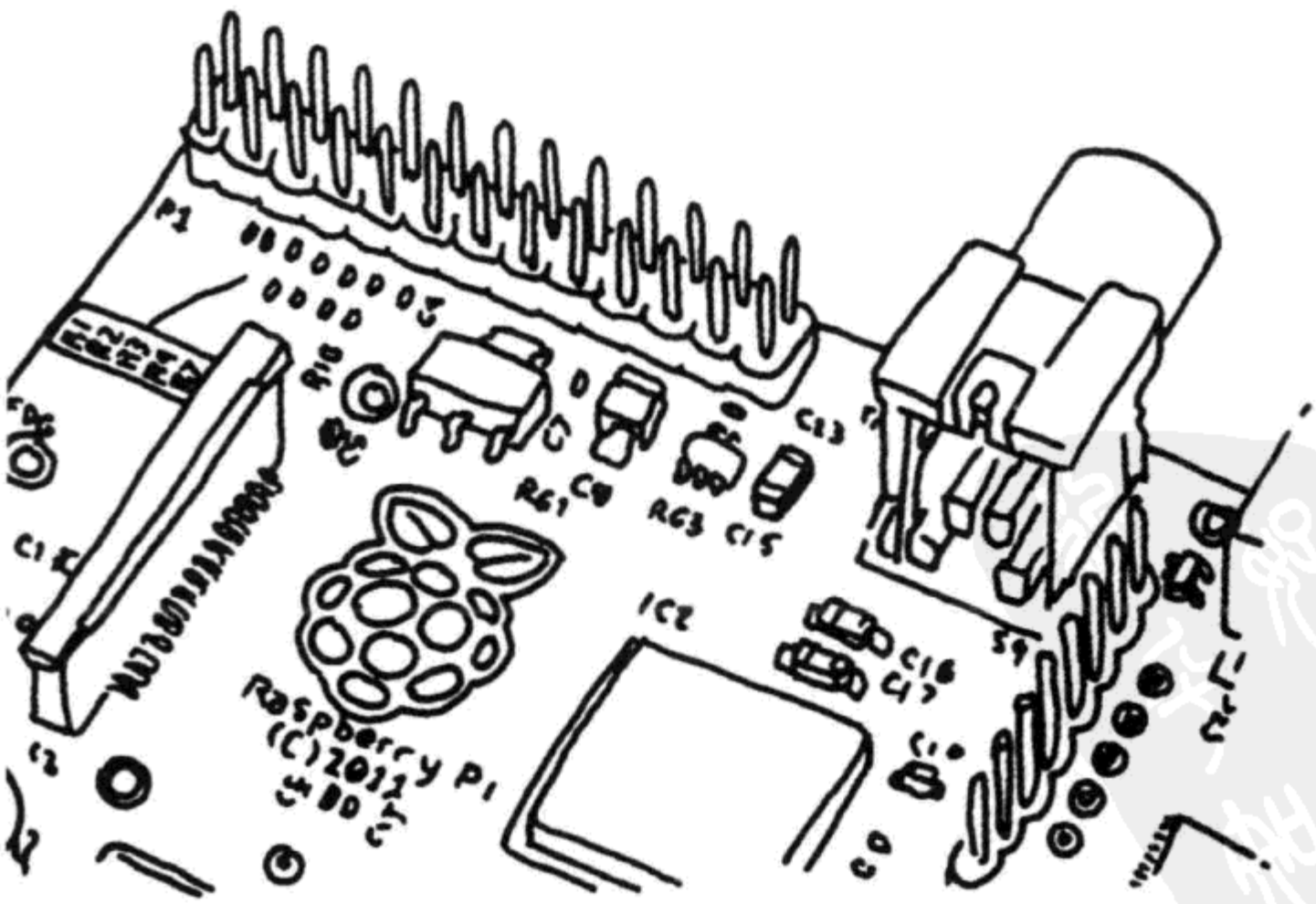
第 8 章

用Python进行输入输出编程

Programming Inputs and Outputs with Python



在第 7 章的最后，我们通过 Shell 脚本进行了一些简单的 Raspberry Pi GPIO 接口的编程。在这一章中，我们会学习如何用 Python 进行 GPIO 编程。与 Shell 脚本一样，使用 Python 也可以通过编写代码去访问和操作 GPIO 接口。

用 Python 而不是 Shell 脚本的好处在于，Python 脚本更容易编写，可读性也更强。有一些封装好的 Python 模块可以让你只用很简单的代码就能完成较为复杂的操作。表 3.2 中列出了一些有用的 Python 模块，通过使用其中的 raspberry-gpio-python 模块（*http://code.google.com/p/raspberry-gpio-python/*）可以非常方便地访问和控制 GPIO 接口。在本章中，你会学到如何去使用这个模块。

在 Python 中安装并测试 GPIO

在最新的 Raspbian 系统中已经预装了 GPIO 模块。如果还在使用较早版本的 Raspbian，你需要自行安装。可以通过 Python 的交互式解释器（参考第 3 章中的内容，交互式解释器允许你直接输入单行 Python 代码并立即运行，不需要先把代码写入文件再执行）来验证你是否已经安装了这个模块。

1. 从命令行以 `root` 身份启动 Python 的交互式解释器（因为 raspberry-gpio-python 需要 `root` 权限来操作 GPIO 接口，所以在启动 Python 的交互式解释器时，需要在前面添加 `sudo` 命令）。

```
pi@raspberrypi ~ $ sudo python
Python 2.7.3rc2 (default, May 6 2012, 20:02:25)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more
information.
>>>
```

2. 在 >>> 提示符后面，尝试导入这个模块：

```
>>> import RPi.GPIO as GPIO
```

3. 如果没有出错，证明这个模块已经正确安装了。

如果在导入 GPIO 模块时出错了，可以通过 Raspberry Pi 的软件包管理器 apt-get 命令很容易地安装这个模块。

如果你的系统中没有安装 raspberry-gpio-python，可以用下面的步骤来安装。

1. 退出 Python 解释器（按 Control-D 或输入 exit() 并回车），更新 apt-get 软件包列表，然后执行安装命令来安装 raspberry-gpio-python 包：

```
>>> exit()
pi@raspberrypi ~ $ sudo apt-get update
pi@raspberrypi ~ $ sudo apt-get install python-rpi.gpio
```

2. 安装完成后，重新运行 Python 的交互式解释器并导入模块。

```
pi@raspberrypi ~ $ sudo python
Python 2.7.3rc2 (default, May 6 2012, 20:02:25)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more
information.
>>> import RPi.GPIO as GPIO
>>>
```

在本章中，我们使用 Python 2.7。不使用 Python 3 的原因是，我们所用到的一些模块在 Raspberry Pi 上默认只安装在 Python 2.x 的环境中。当你在 Raspberry Pi 的命令行上运行 `python` 命令时，目前的行为是默认运行 Python 2.7，但以后可能会改成 Python 3（任何时候，你都可以显式地输入 `python2.7` 而不是 `python` 命令来运行 Python 2.7）。

Python 2.7 与 Python 3 的最明显的区别之一是向屏幕输出文本的方式：在 Python 2.x 中使用 `print "Hello, World!"` 这样的形式，而在 Python 3 中使用 `print("Hello, Wrold!")`。

如果输入 `import` 命令后没有出错，你就可以继续下面的实验了。

1. 在使用 GPIO 接口前，先要告诉 GPIO 模块你打算以何种方式来指代 GPIO 接口。在第 7 章中，我们所使用的 GPIO 接口编号与它们在主板上的引脚排列没有直接的关系，本质上是在使用 Broadcom 芯片的引脚编号。使用 GPIO 的 Python 模块时，你可以使用芯片引脚编号也可以使用主板引脚编号来访问 GPIO 接口。如果要使用主板引脚编号，使用 `GPIO.setmode(GPIO.BOARD)` 命令来设置 GPIO 模块。不过在本章中，我们还是使用与第 7 章中一样的芯片引脚编号（`GPIO.setmode(GPIO.BCM)`）来操作 GPIO 接口，这种编号方式也是 Adafruit Pi Cobler 等套件板上标注引脚编号的方式：

```
>>> GPIO.setmode(GPIO.BCM)
```

2. 把 GPIO 25 设置为输出状态：

```
>>> GPIO.setup(25, GPIO.out)
```

3. 把 LED 接到 GPIO 25 接口（与“使用输入输出接口”一节中的做法一样）。

4. 点亮 LED：

```
>>> GPIO.output(25, GPIO.HIGH)
```

5. 熄灭 LED：

```
>>> GPIO.output(25, GPIO.HIGH)
```

6. 退出 Python 交互式解释器：

```
>>> exit()
pi@raspberrypi ~ $
```

在第 7 章，我们强调过 Raspberry Pi 的数字输入输出接口信号应该是 3.3V 或接地。在电子学中，我们把这些信号称为高电平或低电平。请记住，不是所有的电路都使用 3.3V 作为高电平信号，有些电路会使用 1.8V 或 5V。如果你要把 Raspberry Pi 与其他数字设备通过 GPIO 接口相连，必须确认那些设备也是以 3.3V 作为工作电压。

以上步骤让你初步了解了如何在 Python 的交互式解释器中用单行 Python 命令来控制 GPIO 接口。在第 7 章中，我们使用 Shell 脚本来操作 GPIO 接口，下面我们也会用 Python 脚本来自动读取或控制这些 GPIO 接口。

让 LED 闪烁

要通过 Python 让 LED 闪烁起来，你需要使用先前在交互式解

释器中实验过的命令和另外一些命令。在下面的操作步骤中，我们假设你使用的是桌面环境（图 8.1）。但如果你喜欢用命令行来编写和运行 Python 脚本的话，也完全没有问题。

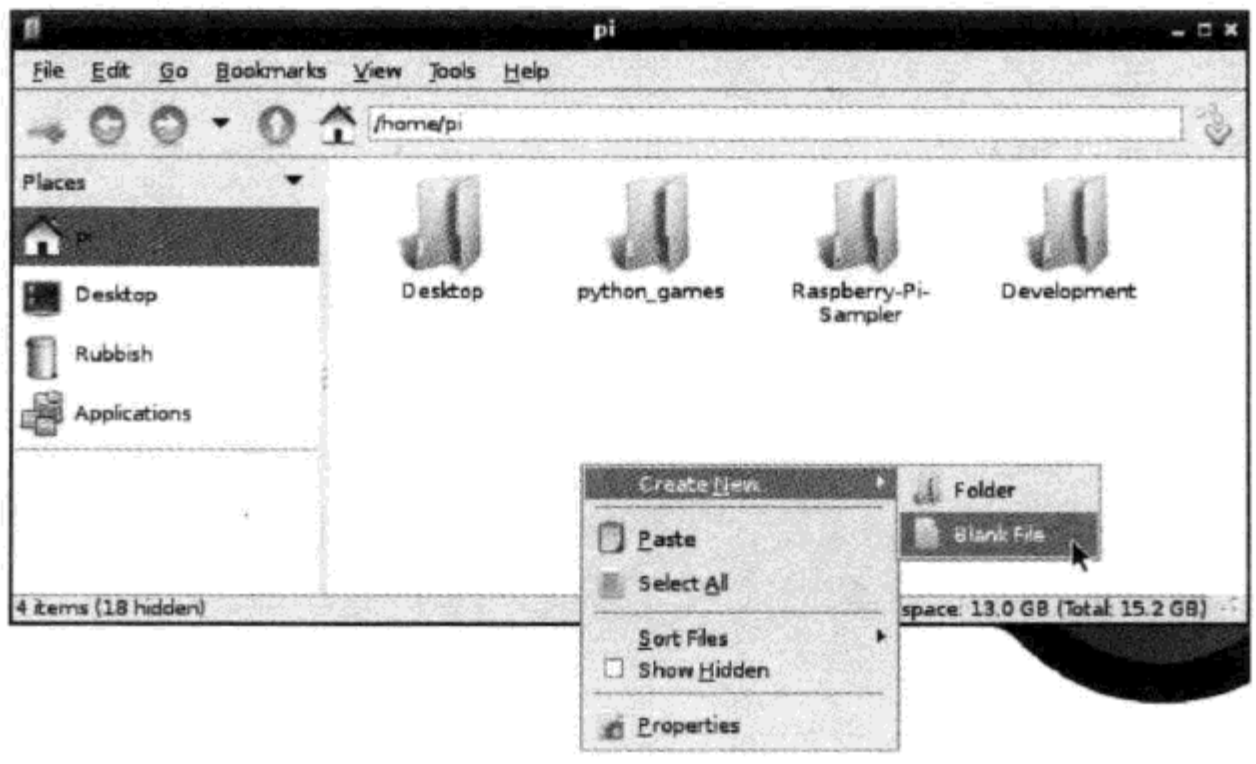


图8.1　在用户主目录中创建一个新文件

1. 通过任务栏上的按钮打开文件管理器（File Manager）。

2. 确认当前目录是你的主目录（默认为 */home/pi*）。如果不是，点 Places 列表下的主目录图标。

3. 在你的主目录中创建一个文件，命名为 *blink.py*。操作方法是在你的主目录窗口中点击鼠标右键，选 Create New…然后选 Blank File。把这个文件命名为 *blink.py*。

4. 双击 *blink.py*，通过默认的 Leafpad 文本编辑器打开它。

5. 输入以下的代码并保存文件：

```
import RPi.GPIO as GPIO❶
import time❷

GPIO.setmode(GPIO.BCM)❸
GPIO.setup(25, GPIO.OUT)❹
```

```
while True:❺
    GPIO.output(25, GPIO.HIGH)❻
    time.sleep(1)❼
    GPIO.output(25, GPIO.LOW)❽
    time.sleep(1)❾
```

❶ 导入控制 GPIO 所需的代码。

❷ 导入 sleep 函数所需的代码。

❸ 设置使用芯片引脚编号。

❹ 把 GPIO 25 接口设置为输出模式。

❺ 开始一个无限循环，执行下面缩进的代码。

❻ 点亮 LED。

❼ 暂停 1s。

❽ 熄灭 LED。

❾ 暂停 1s。

请牢记：在 Python 中，代码的缩进位置非常重要。

6. 打开 LX 终端（LXTerminal），执行下面的代码把当前工作目录切换到你的主目录，并执行刚才输入的脚本：

```
pi@raspberrypi ~/Development $ cd ~
pi@raspberrypi ~ $ sudo python blink.py
```

7. 你的 LED 应该开始闪烁了！

8. 按 Control-C 中断脚本的运行并返回命令行。

你可以尝试在 time.sleep() 函数中指定小于 1 的小数使 LED 闪烁得更快。你也可以尝试连接更多的 LED，并让它们以特

定的模式来闪烁。你可以参考图 7.2 使用以下独立的 GPIO 接口中的任意一些：4，17，18，21，22，23，24 或 25。

读取按钮状态

如果你想在按钮按下时触发特定的操作，实现这个目标的方法之一称为轮询。轮询的意思就是不断地检查某一个状态，在我们的例子中就是指不断检查相关的 GPIO 接口是被连接到了 3.3V 还是接地。下面的实验可以实现在按下按钮时屏幕上显示一些文本。

1. 用“数字信号输入：读取按钮状态”（第 7 章）一节中相同的方法连接一个按钮，使用 GPIO 24 接口作为输入。别忘了在输入接口与地线之间加上下拉电阻。

2. 在你的主目录中创建一个新文件，名为 *button.py*，在文本编辑器中打开它。

3. 输入下面的代码：

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
GPIO.setup(24, GPIO.IN)❶

count = 0❷

while True:
    inputValue = GPIO.input(24)❸
    if (inputValue == True):❹
        count = count + 1❺
        print("Button pressed " + str(count) + " times.")❻
    time.sleep(.01)❼
```

❶ 把 GPIO 24 接口设置为输入接口。

❷ 创建一个名为 count 的变量，并把值设为 0。

❸ 把 GPIO 24 接口上读到的状态存入 `inputValue` 变量。

❹ 检查读到的值是否为 `True`（意味着按钮被按下）。

❺ 如果按钮被按下，计数器 `counter` 加 `1`。

❻ 在屏幕上显示文字。

❼ 稍稍暂停一会儿，释放处理器的计算资源，使得其他程序可以有机会被执行到。

4. 回到 LX 终端（LXTerminal），运行这个脚本：

```
pi@raspberrypi ~ $ sudo python button.py
```

5. 按动按钮。如果你的电路和程序一切都正常，每次屏幕上都会显示出几行 “Button pressed * times”（按钮被按下了 * 次）的提示。

在这个例子中，程序 1s 内会检测 100 次按钮的状态，所以你每按一下按钮都可能显示出多行信息来（除非你的动作快到无与伦比）。代码中的 `time.sleep(.01)` 决定了多久检测一次按钮的状态。

为什么不能不间断地检测按钮状态呢？如果你把代码中的 `time.sleep(.01)` 这行去除，检测循环会更快地运行，所以当按钮按下时你也可以更快地知道。这样做有一些缺点：你会占用过多的处理器时间，其他程序就难以正常工作，同时也会增加 Raspberry Pi 的功耗。因为 *button.py* 与其他程序一起共享 Raspberry Pi 的硬件资源，所以你要保证不能让一个程序消耗完所有的资源。

添加一些代码，可以让程序更好地检测单次按钮动作：

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
GPIO.setup(24, GPIO.IN)

count = 0
```

```
while True:
    inputValue = GPIO.input(24)
    if (inputValue == True):
        count = count + 1
        print("Button pressed " + str(count) + " times.")
        time.sleep(.3)❶
    time.sleep(.01)
```

❶ 更好地处理单次按钮动作。

新添加的代码可以更好地保证每次按动按钮只被记录一次，但这不是一个完美的解决方案，如果你按住按钮不放，那 1s 内会被检测成 3 次按钮动作。如果你快速按动按钮，有几次动作就不会被检测到，因为它无法检测 1s 内 3 次以上的按钮动作。

这些都是通过轮询的方式检查数字输出接口状态时会遇到的问题。解决这些问题的一个常见思路是使用中断。中断是指在硬件检测到某个引脚电平状态变化时，回调一段特定的代码的方式。目前，RPi.GPIO 模块中对中断的支持还处于实验性阶段[①]，可以参考这个模块的文档（*http://pypi.python.org/pypi/RPi.GPIO*）来了解如何使用这个功能。

项目：简易发音板

我们已经学习了如何在 Raspberry Pi 上读取输入接口的状态，现在你可以用 Python 的 Pygame 模块中的 sound 函数来制作一块发音板。所谓发音板，是指在你按动上面的按钮时可以播放一段指定的声音的设备。为了制作发音板，你需要以下的材料：

- 3 个按钮开关；
- 母头对公头的连接线；

① 0.5.0a 以后版本的 RPi.GPIO 模块已经正式支持中断，可以使用 wait_for_edge() 函数进行边缘检测或用 add_event_detect() 和回调函数实现中断处理。——译者注

- 适当长度的连接线；
- 面包板；
- 3 个 10kΩ 的电阻；
- 音箱，或者具备 HDMI 接口并内置音箱的显示器。

你还需要几个未压缩的 *.wav* 格式的声音文件。Raspberry Pi 的系统中自带了几个声音文件，你可以直接用来测试。只要你的发音板可以正常工作了，把这个声音文件替换成你所想要的声音是一件很容易的事情，当然，你也许需要先把它们从其他格式转换为 *.wav* 格式。下面先来搭建电路。

1. 用母头对公头连接线把 Raspberry Pi 的接地接口与面包板的电源总线负极相连。

2. 用母头对公头连接线把 Raspberry Pi 的 3.3V 电源接口与面包板的电源总线正极相连。

3. 在面包板上安装 3 个按钮开关，横跨在面包板的中央绝缘条两边。

4. 用连接线把电源负极与按钮上部的引脚相接。

5. 安装下拉电阻，用 10kΩ 的电阻把按钮下部的引脚与接地相连。

6. 用母头对公头连接线把每个按钮下部的引脚（与 10kΩ 电阻相连的那个）与 Raspberry Pi 的 GPIO 接口相连。在这个项目中，我们使用 GPIO 23、24 和 25 号接口。

图 8.2 展示了完成后的电路图。我们使用 Fritzing 软件来制作这个图片，Fritzing（*http://fritzing.org*）是一个用于硬件设计的开源软件。

把电路搭好后，就该开始编写代码了。

1. 在你的主目录中创建一个新的目录，名为 *soundboard*。

2. 打开这个目录并在里面创建一个名为 *soundboard.py* 的文件。

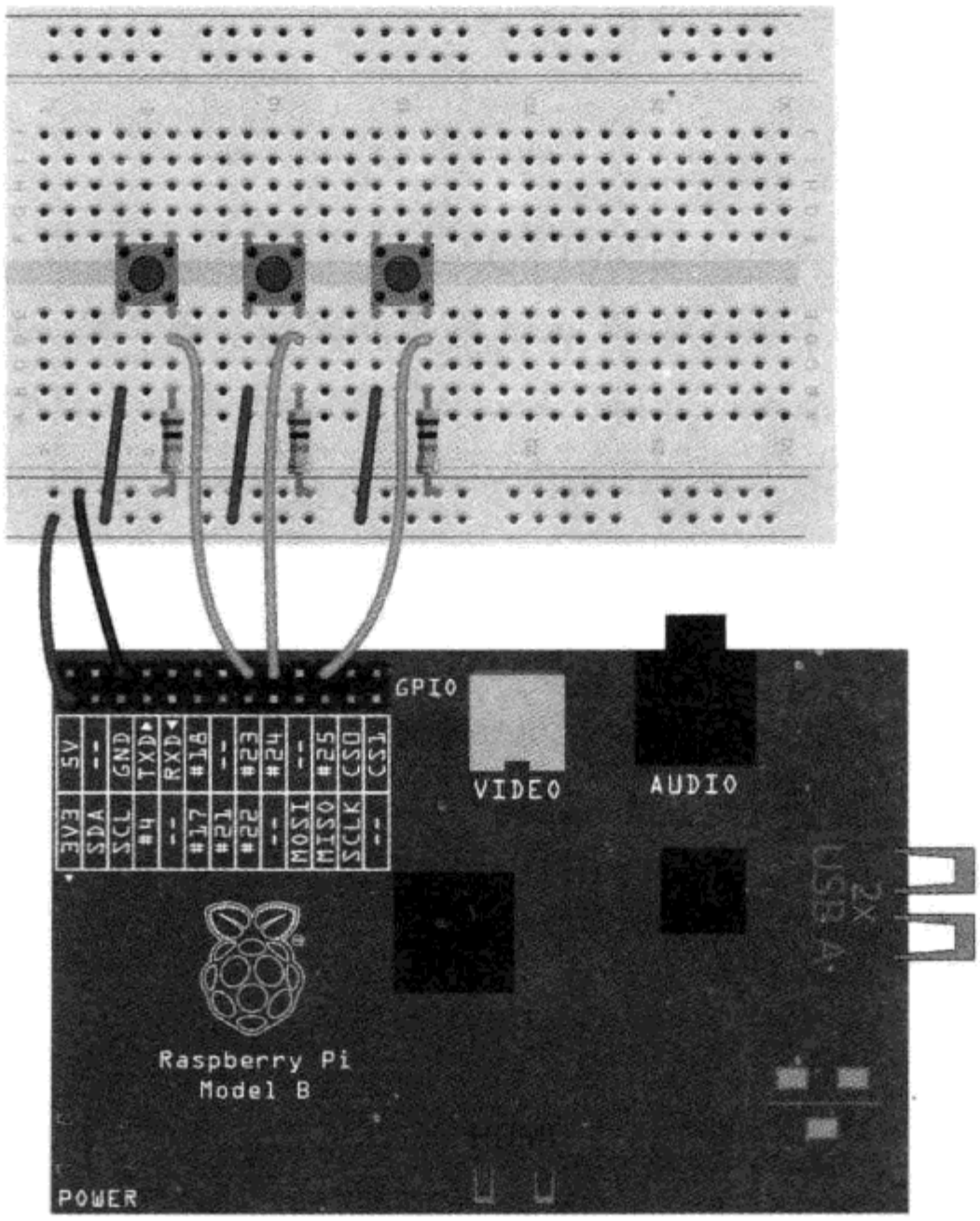


图8.2　发音板项目完成后的电路图

3. 打开 *soundboard.py* 并输入下面的代码：

```
import pygame.mixer
from time import sleep
import RPi.GPIO as GPIO
from sys import exit

GPIO.setmode(GPIO.BCM)
GPIO.setup(23, GPIO.IN)

GPIO.setup(24, GPIO.IN)
GPIO.setup(25, GPIO.IN)
```

```
    pygame.mixer.init(48000, -16, 1, 1024)❶
    soundA = pygame.mixer.Sound("/usr/share/sounds/alsa/Front_
Center.wav")❷
    soundB = pygame.mixer.Sound("/usr/share/sounds/alsa/Front_
Left.wav")
    soundC = pygame.mixer.Sound("/usr/share/sounds/alsa/Front_
Right.wav")

    soundChannelA = pygame.mixer.Channel(1)❸
    soundChannelB = pygame.mixer.Channel(2)
    soundChannelC = pygame.mixer.Channel(3)

    print "Soundboard Ready."❹

    while True:
        try:
            if (GPIO.input(23) == True):❺
                soundChannelA.play(soundA)❻
            if (GPIO.input(24) == True):
                soundChannelB.play(soundB)
            if (GPIO.input(25) == True):
                soundChannelC.play(soundC)
            sleep(.01)❼
        except KeyboardInterrupt:❽
            exit()
```

❶ 初始化 Pygame 的混音器。

❷ 加载声音文件。

❸ 设置 3 个通道，每段声音对应一个通道，这样就可以做到同时播放不同的声音。

❹ 显示提示用户发音板已经启动完毕（这里使用了 Python 2 语法）。

❺ 如果对应接口是高电平，执行下面的代码。

❻ 播放声音。

❼ 检测按钮的间隔不要太短，以保证不会占用太多的处理器资源。

❽ 这个异常捕获代码可以保证在按下 Control-C 时程序可以正常退出，而不会显示一堆用于调试的栈信息。

4. 到命令行下把当前工作目录切换到 *soundboard.py* 所在的目录，然后用 Python 2 执行这个脚本：

```
pi@raspberrypi ~/soundboard $ sudo python soundboard.py
```

5. 看到屏幕提示“Soundboard Ready”后，按动面包板上的按钮就可以播放相应的声音。

Pygame 有支持 Python 3 的版本，但在 Raspberry Pi 上默认只安装了支持 Python 2 的版本。

根据 Raspberry Pi 设置的不同，声音可能会从 HDMI 接口传到显示器上输出，也可能是通过板载的 3.5mm 模拟音频接口输出。如果改变声音输出的接口，可以先按 Control-C 退出程序，然后在命令行上输入下面的命令把声音设置为从模拟音频接口输出：

```
pi@raspberrypi ~/soundboard $ sudo amixer cset numid=3 1
```

如果要把声音设置为通过 HDMI 接口输出，使用命令：

```
pi@raspberrypi ~/soundboard $ sudo amixer cset numid=3 2
```

当然，默认的那些声音并不是很有意思，你可以把它们替换成任意你喜欢的声音，如掌声、笑声、警报声或铃声。把这些声音对应的文件放入你的 *soundboard* 目录并修改代码让它使用这些文件即可。如果想让你的发音板发出更多种类的声音，也可以通过安装更

多的按钮并修改相应的代码来实现。

进一步学习

RPi.GPIO

（*http://code.google.com/p/raspberry-gpio-python/*）

RPi.GPIO 库还在不断开发中，你可以访问它的主页来了解最新的进展。

Using the MCP3008

（*http://learn.adafruit.com/reading-a-analog-in-and-controlling-audio-volume-with-the-raspberry-pi/overview*）

Adafruit 的一篇不错的教程，介绍了如何使用 MCP3008 模数转换器在 Raspberry Pi 上使用输出模拟信号的传感器。