Raspberry Pi Recipes Part #7

PWMin' is a piece of cake

By **Tony Dixon** (UK)

At this stage of the (pastry) game, in this series we have looked at a lot of digital signals, GPIO, Serial UART, SPI and I²C, as well as some analog signals (via SPI), so where next? This time we'll be adding PWM functionality to our Raspberry Pi.

Hardware PWM interfaces

In pulsewidth modulation (PWM), the signal consists of a rectangular pulse wave where the pulse width is modified or, in technical terms, modulated. This change produces a variation of the average value of the waveform. PWM is mainly used to control the power supplied to electrical devices such as lamps and motors. Broadcom's BCM2835 system on chip (SoC), which is the heart of the Pi, has two hardware PWM channels. One is used by the system to provide audio and the other PWM0 is spare and can be found on the Raspberry Pi Expansion Header on pin 12 (GPIO18), see **Table 1**.

LED dimming

We can use the hardware PWM0 to dim an LED. To demonstrate LED dimming via PWM control we'll use a 330-Ω resistor and an LED connected between PWM0 (GPIO18/pin 12) and ground.

At this point we would be firing up Python and making the RPi.GPIO low-level library do some interesting things, but surprisingly RPi.GPIO doesn't have a method to access the hardware PWM0. So, instead to demonstrate hardware PWM functionality we'll install *wiringPi* written by Gordon Henderson [1]. *wiringPi* is a C-based low level library for the

Raspberry Pi and it is usable by many different programming languages including Python. *wiringPi* is very similar to *wiring*, which forms the software platform for Arduino.
To start we'll have to install some more software. First we'll install Python Package Installer (PiP) with:

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

Once this is installed we'll download and install *wiringPi* for Python by typing:

```
sudo pip install wiringpi2
```

Let's start IDLE, the Python IDE and type the code in **Listing 1**.
This Python program sets up GPIO18/PWM0 as a digital output. It then enters a loop where the modulation to the PWM0 is increased. As the PWM signal increases, so does the brightness of the LED.

Software PWM interfaces

Okay—we got a single hardware PWM on the Expansion Header, but what if we want to use PWM for more than one signal? We could use a chip such as the Texas Instruments PCA9685 16-channel 12-bit PWM to provide additional PWM channels, or we can use software PWM which cost us nothing.
A small warning: normally, software PWM is not a problem when used on an embedded platform such as an Arduino, where the processor only runs its application code. However, using a software PWM on a general-purpose computer such as the Raspberry Pi has the additional complication of a fully blown operating system (OS) juggling many different tasks and programs at the same time. Because of this you cannot guarantee that "software only" PWM timing will not suffer from low resolution and high jitter, as the OS pre-empts the software PWM program to allow other things to run. This could be a problem if we need precision and low jitter. Fortunately, the Pi has several Direct Memory Access (DMA) hardware channels which can be used to provide a hardware timing aid to our software PWM. As the DMA operates separately from the CPU, our timing is now based on hardware so the PWM timings are less prone to interruption by the OS and PWM precision and jitter are now acceptable.

Table 1. Expansion Header Pin Out

Pin Name	Pin Function	Alternative	RPi.GPIO
P1-02	5.0V	-	-
P1-04	5.0V	-	-
P1-06	GND	-	-
P1-08	GPIO14	UART0_TXD	RPi.GPIO8
P1-10	GPIO15	UART0_RXD	RPi.GPIO10
P1-12	GPIO18	**PWM0**	RPi.GPIO12
P1-14	GND	-	-
P1-16	GPIO23		RPi.GPIO16
P1-18	GPIO24		RPi.GPIO18
P1-20	GND	-	-
P1-22	GPIO25		RPi.GPIO22
P1-24	GPIO8	SPI0_CE0_N	RPi.GPIO24
P1-26	GPIO7	SPI0_CE1_N	RPi.GPIO26

Pin	Board Revision 1		Board Revision 2	
Name	Pin Function	Alternative	Pin Function	Alternative
P1-01	3.3V	-	3.3V	-
P1-03	GPIO0	I2C0_SDA	GPIO2	I2C1_SDA
P1-05	GPIO1	I2C0_SCL	GPIO3	I2C1_SCL
P1-07	GPIO4	GPCLK0	GPIO4	GPCLK0
P1-09	GND	-	GND	-
P1-11	GPIO17	RTS0	GPIO17	RTS0
P1-13	GPIO21		GPIO27	
P1-15	GPIO22		GPIO22	
P1-17	3.3V	-	3.3V	-
P1-19	GPIO10	SPI0_MOSI	GPIO10	SPI0_MOSI
P1-21	GPIO9	SPI0_MISO	GPIO9	SPI0_MISO
P1-23	GPIO11	SPI0_SCLK	GPIO11	SPI0_SCLK
P1-25	GND	-	GND	-

Note: I2C0_SDA and I2C0_SCL (GPIO0 & GPIO1) and I2C1_SDA and I2C1_SCL (GPIO2 & GPIO3) have 1.8-kΩ pull-up resistors to 3.3 V.

Listing 1. Dimming an LED

```
#!/usr/bin/python

import wiringpi2 as gpio
import time

#set up gpio
gpio.wiringpiPiSetupGpio ()
gpio.pinMode (18,2)

while True:
  gpio.pwmWrite (18,0)
  for n in range (0,1024):
    gpio.pwmWrite (18,n)
    time.sleep (0.01)
```

DC motor control

Another common use for PWM is to control the rotational speed of an electric motor.
The H bridge circuit depicted in **Figure 1** allows a motor to be controlled in both directions, but if we apply a PWM signal to one of the switching quadrants we can also control the speed as well in the direction of rotation.

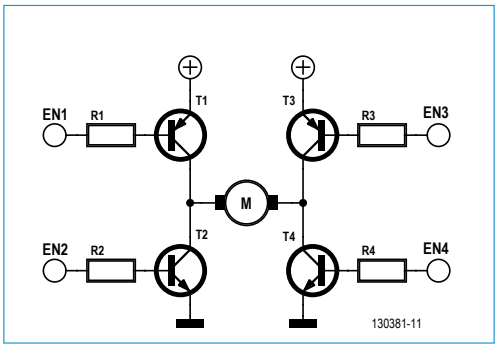

Figure 1.
H bridge schematic.

PiiBOT: *rolling* out the dough

The PiiBOT is a 4-wheel driven robot operated remotely using a Nintendo Wii controller, as can be seen in **Figure 2**. This *roller* features two L293D DC motor controller chips used to drive four small brushed DC motors and a Bluetooth USB dongle to accept instructions from the Wii controller.

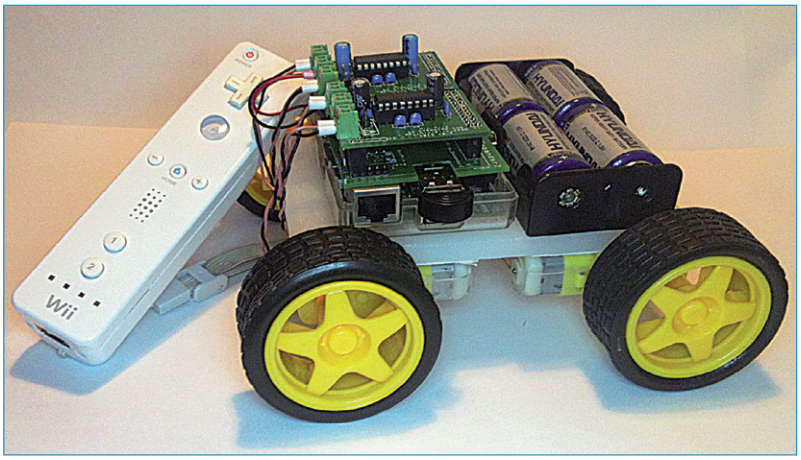

Figure 2.
PiiBOT, a simple 4-wheel roller.

Circuit description

Figure 3 shows the circuit schematic for our motor controller Pi shield [2]. It has two L293D chips. The L293D is a veteran of many small motor projects. It has two full H bridge circuits, each capable of sourcing 600 mA DC (1.2 A peak) and it's very easy to interface to. Each H bridge has an *enable* (EN1/2) signal and two directional input (IN1/3 and IN2/4) control signals, as shown in **Figure 4**.
External power (4.5 V to 36 VDC) can be supplied via connector K4. Alternatively, power can be selected internally via jumper J1 from the Pi's 5 V supply, but it is recommended to use an external supply via K4.
For control we require six GPIO signals per

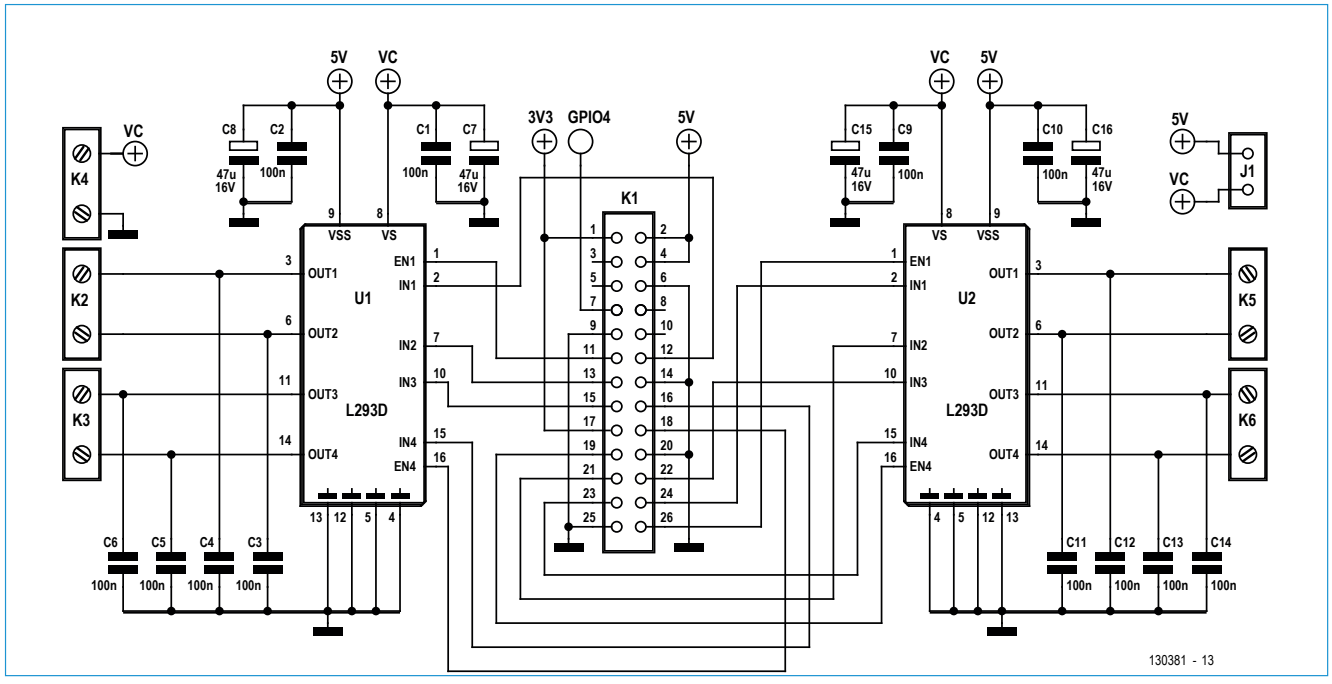

Figure 3. Schematic of the PiiBOT's add-on motor controller shield.

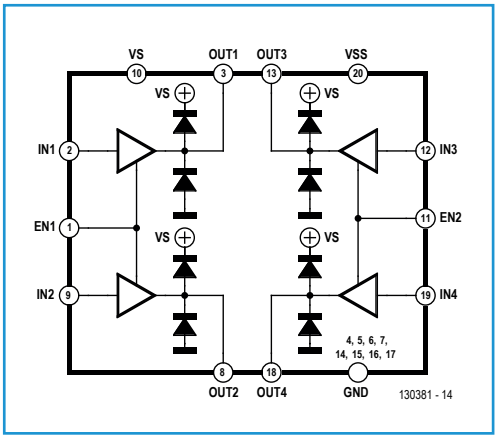

Figure 4. L293D motor controller.

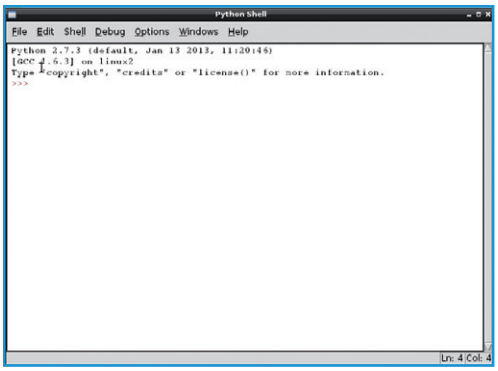

Figure 5. Python shell.

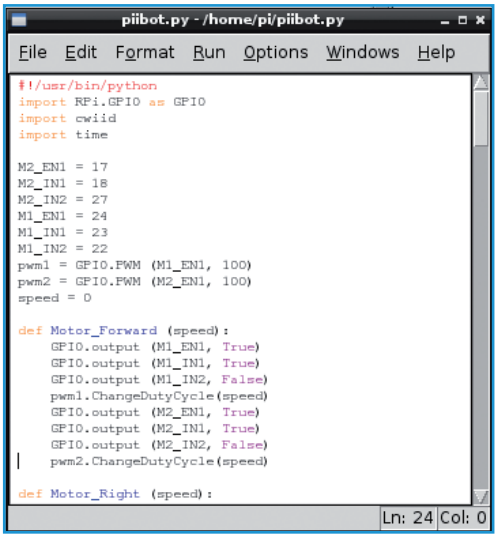

Figure 6. IDLE editor with the "piibot.py" script.

Table 2. GPIO Usage

Motors 1 and 2 (U1 L293D)		Motors 3 and 4 (U2 L293D)	
Motor Function	**GPIO**	**Motor Function**	**GPIO**
EN1	GPIO17	EN1	GPIO7
IN1	GPIO18	IN1	GPIO8
IN2	GPIO27	IN2	GPIO9
EN2	GPIO24	EN2	GPIO10
IN3	GPIO22	IN3	GPIO25
IN4	GPIO23	IN4	GPIO11

L293D chip, making 12 GPIO signals in total from our Pi to control our 4-wheel robot. **Table 2** provides an overview of all these signals.
To control motor speed we'll PWM the *enable* signal of each H bridge. As the circuit has four *enable* signals we'll be using four software PWM channels to achieve this.

Installing Bluetooth drivers and Python's CWii library

To use the Nintendo Wii controller we first need to install Bluetooth drivers for our USB Bluetooth module. We only need basic Bluetooth communication, so we'll use "`--no-install-recommends`" option when we type our command, as shown below:

```
sudo apt-get install --no-install-recommends bluetooth
```

With the Bluetooth dongle plugged into our Pi, we can test the interface by typing:

```
sudo service bluetooth status
```

If everything is good we should get a response showing:

```
[ ok ] bluetooth is running.
```

Once the Bluetooth drivers are installed we can download CWii [4], the Python Wii library. Simply type the following commands:

```
sudo apt-get install python-cwiid
```

With everything installed we should be good to use the Wii Controller with our Pi, or more specifically our PiiBOT. More information on how to read data from such a controller may be found in this link [5].

Example program: "piibot.py"

With our circuit built we are almost ready to write our Python control program. But before starting we need to install the drivers and libraries required for our Nintendo Wii Controller. The instructions are included in the text box, right on this same article.

Let's double click IDLE icon on your Pi's desktop to start the Python Shell and IDE (**Figure 5**).

Select "File" option from the menu and create a new program. This will start the IDLE editor (**Figure 6**), where we'll have to type the program shown in **Listing 2**. Due to its length, it may also be downloaded from the Raspberry Pi series' support page at Elektor. LABS [3].

Once we're all set, we should make sure that it's saved before switching to LX Terminal, typing the following command to make our program an executable:

```
chmod +x piibot.py
```

Once done, we can run our program by typing:

```
sudo ./piibot.py
```

When our program is running we'll be asked to pair the Wii controller with our Pi. To do this, buttons 1 and 2 on the Wii controller must be pressed at the same time. Once the controller is paired, our PiiBOT will be ready to roll!

(130381)

Internet Links

[1] WiringPi GPIO Library: http://wiringpi.com

[2] MiniPiio Motor293D add-on board: www.dtronixs.com

[3] Raspberry Pi Support Site at Elektor. LABS: www.elektor-labs.com/RPi

[4] CWiid library for Nintendo Wii Controller: http://abstrakraft.org/cwiid/

[5] Nintendo Wii Remote, Python and The Raspberry Pi: http://bit.ly/RPi-Wii

Listing 2. "piibot.py" (Download the program at [3])

```
#!/usr/bin/python
import RPi.GPIO as GPIO
import cwiid
import time

M1_EN1 = 24
M1_IN1 = 23
M1_IN2 = 22
M2_EN1 = 17
M2_IN1 = 18
M2_IN2 = 27
M3_EN1 = 7
M3_IN1 = 8
M3_IN2 = 9
M4_EN1 = 10
M4_IN1 = 25
M4_IN2 = 11

speed = 40

def Motor_Setup ():
    print 'Setting up..'
    GPIO.setwarnings (False)
    # Configure GPIO
    GPIO.setmode (GPIO.BCM)
    GPIO.setup (M1_EN1, GPIO.OUT)
    GPIO.setup (M1_IN1, GPIO.OUT)
    GPIO.setup (M1_IN2, GPIO.OUT)
    GPIO.setup (M2_EN1, GPIO.OUT)
    GPIO.setup (M2_IN1, GPIO.OUT)
    GPIO.setup (M2_IN2, GPIO.OUT)
    GPIO.setup (M3_EN1, GPIO.OUT)
    GPIO.setup (M3_IN1, GPIO.OUT)
    GPIO.setup (M3_IN2, GPIO.OUT)
    GPIO.setup (M4_EN1, GPIO.OUT)
    GPIO.setup (M4_IN1, GPIO.OUT)
    GPIO.setup (M4_IN2, GPIO.OUT)

    print "ready"

def Motor_Forward (speed):
    print 'Forward, speed = ', speed
    GPIO.output (M1_IN1, True)
    GPIO.output (M1_IN2, False)
    pwm1.ChangeDutyCycle(speed) # M1 EN1
```

```
    GPIO.output (M2_IN1, True)
    GPIO.output (M2_IN2, False)
    pwm2.ChangeDutyCycle(speed) # M2 EN1
    GPIO.output (M3_IN1, True)
    GPIO.output (M3_IN2, False)
    pwm3.ChangeDutyCycle(speed) # M3 EN1
    GPIO.output (M4_IN1, True)
    GPIO.output (M4_IN2, False)
    pwm4.ChangeDutyCycle(speed) # M4 EN1

def Motor_Right (speed):
    print 'Right, speed = ', speed
    GPIO.output (M1_IN1, True)
    GPIO.output (M1_IN2, False)
    pwm1.ChangeDutyCycle(speed)
    GPIO.output (M2_IN1, False)
    GPIO.output (M2_IN2, False)
    pwm2.ChangeDutyCycle(0)
    GPIO.output (M3_IN1, True)
    GPIO.output (M3_IN2, False)
    pwm3.ChangeDutyCycle(speed)
    GPIO.output (M4_IN1, False)
    GPIO.output (M4_IN2, False)
    pwm4.ChangeDutyCycle(0)

def Motor_Left (speed):
    print 'Left, speed = ', speed
    GPIO.output (M1_IN1, False)
    GPIO.output (M1_IN2, False)
    pwm1.ChangeDutyCycle(0)
    GPIO.output (M2_IN1, True)
    GPIO.output (M2_IN2, False)
    pwm2.ChangeDutyCycle(speed)
    GPIO.output (M3_IN1, False)
    GPIO.output (M3_IN2, False)
    pwm3.ChangeDutyCycle(0)
    GPIO.output (M4_IN1, True)
    GPIO.output (M4_IN2, False)
    pwm4.ChangeDutyCycle(speed)

def Motor_Reverse (speed):
    print 'Reverse, speed = ', speed
    GPIO.output (M1_IN1, False)
    GPIO.output (M1_IN2, True)
    pwm1.ChangeDutyCycle(speed)
    GPIO.output (M2_IN1, False)
    GPIO.output (M2_IN2, True)
    pwm2.ChangeDutyCycle(speed)
    GPIO.output (M3_IN1, False)
    GPIO.output (M3_IN2, True)
    pwm3.ChangeDutyCycle(speed)
    GPIO.output (M4_IN1, False)
    GPIO.output (M4_IN2, True)
    pwm4.ChangeDutyCycle(speed)

def Motor_Stop ():
    GPIO.output (M1_IN1, False)
    GPIO.output (M1_IN2, False)
    pwm1.ChangeDutyCycle(0)
    GPIO.output (M2_IN1, False)
    GPIO.output (M2_IN2, False)
    pwm2.ChangeDutyCycle(0)
    GPIO.output (M3_IN1, False)
    GPIO.output (M3_IN2, False)
    pwm3.ChangeDutyCycle(0)
    GPIO.output (M4_IN1, False)
    GPIO.output (M4_IN2, False)
    pwm4.ChangeDutyCycle(0)


# Main Program
Motor_Setup ()
pwm1 = GPIO.PWM (M1_EN1, 100)
pwm2 = GPIO.PWM (M2_EN1, 100)
pwm3 = GPIO.PWM (M3_EN1, 100)
pwm4 = GPIO.PWM (M4_EN1, 100)
pwm1.start (0)
pwm2.start (0)
pwm3.start (0)
pwm4.start (0)

Motor_Stop ()

button_delay = 0.1

print 'Press 1 + 2 on your Wii Remote'
time.sleep(1)

# Connect to the Wii Remote
try:
    wii=cwiid.Wiimote()
except RuntimeError:
    print 'Error Connecting to Wii Remote'
    quit()

print 'Wii Remote connected'

wii.rpt_mode = cwiid.RPT_BTN
```

```
# Loop
try:
    while True:

        buttons = wii.state['buttons']

        stop = True

        if (buttons & cwiid.BTN_LEFT):
            Motor_Left (speed)
            time.sleep(button_delay)
            stop = False

        if(buttons & cwiid.BTN_RIGHT):
            Motor_Right(speed)
            time.sleep(button_delay)
            stop = False

        if (buttons & cwiid.BTN_UP):
            Motor_Forward (speed)
            time.sleep(button_delay)
            stop = False

        if (buttons & cwiid.BTN_DOWN):
            Motor_Reverse (speed)
            time.sleep(button_delay)
            stop = False


        if (buttons & cwiid.BTN_MINUS):
            speed = speed - 1
            if (speed < 0):
                speed = 0
            print 'speed =', speed
            time.sleep(button_delay)

        if (buttons & cwiid.BTN_PLUS):
            speed = speed + 1
            if (speed > 100):
                speed = 100
            print 'speed =', speed
            time.sleep(button_delay)

        if (stop == True):
            Motor_Stop ()

except KeyboardInterrupt:
    pass

print "End"
GPIO.output (M1_EN1, False)
GPIO.output (M1_IN1, False)
GPIO.output (M1_IN2, False)
GPIO.output (M2_EN1, False)
GPIO.output (M2_IN1, False)
GPIO.output (M2_IN2, False)
GPIO.output (M3_EN1, False)
GPIO.output (M3_IN1, False)
GPIO.output (M3_IN2, False)
GPIO.output (M4_EN1, False)
GPIO.output (M4_IN1, False)
GPIO.output (M4_IN2, False)

pwm1.stop ()
pwm2.stop ()
pwm3.stop ()
pwm4.stop ()

GPIO.cleanup
exit (wii)
```