View on GitHub

# Introduction to Raspberry Pi

## Artisan's Asylum

## PWM and Servo Control

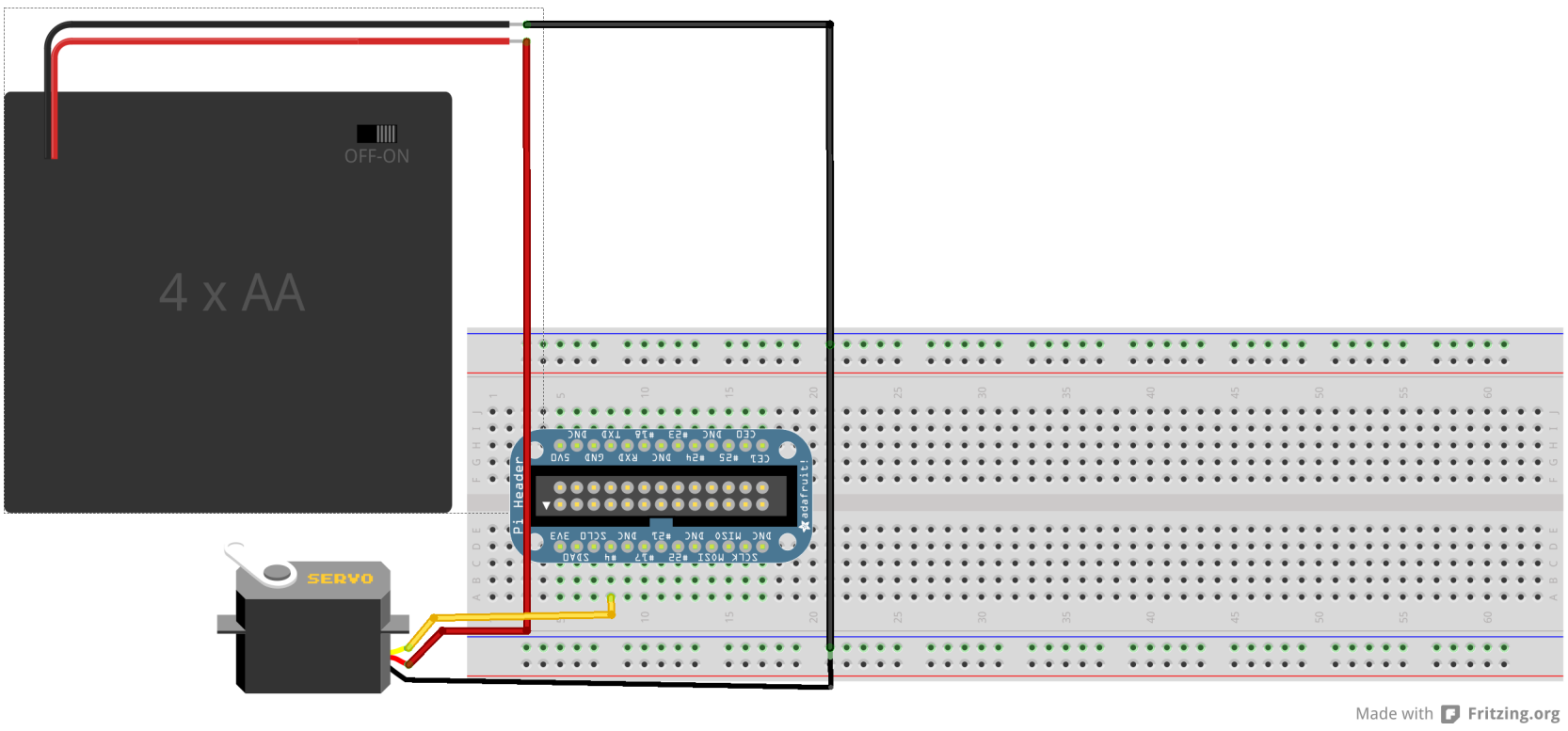
Using the GPIO ports provides binary on/off control. Often, more detailed control is required. Dimming LEDs or controlling motor speed are two such examples. One possible way to slow down a motor is to run it on lower voltage (3 volts instead of 5). Lowering the voltage is not possible with the Raspberry Pi (or most microcontrollers) without specialized circuitry. Instead, the Raspberry Pi switches the output on and off very rapidly, making it appear to the motor like the voltage is lower. This technique is called Pulse Width Modulation (PWM). In addition to motor speed control, PWM can also be used to steer a servo motor. Servos are a special class of motors, which do not (usually) spin continuously, but rather set an output angle (position). They are often used for steering robots or adjusting control surfaces on RC planes.

The Raspberry Pi contains a single hardware PWM/servo driver. The pure hardware PWM from the hardware driver provides a very accurate and clean PWM signal. Support for this driver is limited, currently supported by the semi-outdated Adafruit Occidentalis distribution and the more up to date WiringPi library. As there is only a single PWM channel, only 1 servo can be controlled.

To control more than 1 servo, some timing accuracy can be traded off for more channels. Libraries like RPIO and Servoblaster (what we'll use today) are not a pure hardware solution (they are driven by DMA transfers). Their signal is slightly less clean and might be affected by other programs running on the Raspberry Pi.

## Hardware connections

Connect your servo as shown below. For testing purposes, or teeny servos, you can maybe skate by using the 5V connection from the Raspberry Pi. Otherwise, you almost definitely need a battery or external power supply as shown below. Be sure the battery ground is connected to the Raspberry Pi ground.




# Servoblaster

Follow these instructions to install Servoblaster:

```
$ git clone /github.com/richardghirst/PiBits.git
$ cd PiBits/ServoBlaster/user
~/PiBits/ServoBlaster/user$ make
~/PiBits/ServoBlaster/user$ sudo make install
~/PiBits/ServoBlaster/user$ ls -l /dev/servoblaster
prw-rw-rw- 1 root root 0 Sep 17 00:55 /dev/servoblaster
```

Servoblaster is controlled by writing into the /dev/servoblaster file. The content is written as `<servo-number>=<servo-position>`. The first field written is the servo number. The following table shows which output pin each servo channel is connected to.

| Servo number | GPIO number | Pin in P1 header |
|---|---|---|
| 0 | 4 | P1-7 |
| 1 | 17 | P1-11 |
| 2 | 18 | P1-12 |
| 3 | 21/27 | P1-13 |
| 4 | 22 | P1-15 |
| 5 | 23 | P1-16 |
| 6 | 24 | P1-18 |
| 7 | 25 | P1-22 |

The allowable position values depend on your servo, for mine values between 80 and 249 were accepted. The servo specification often provides the number of steps the servo supports.

## Source Code

```
#!/usr/bin/env python
#
# Drives Servoblaster controlled servo
# https://github.com/richardghirst/PiBits/tree/master/ServoBlaster
#


import time

# From /dev/servoblaster.cfg:
# Servo Channel 0 => GPIO 4
servoChannel = 2

def setServo(servoChannel, position):

    servoStr ="%u=%u\n" % (servoChannel, position)


    with open("/dev/servoblaster", "wb") as f:
        f.write(servoStr)


if __name__ == '__main__':
    val = 50
    direction = 1
    while True:
        #print val
        setServo(servoChannel, val)
        time.sleep(.01)

        if val == 249:
            direction -= 1
        elif val == 50:
            direction = 1

        val += direction
```

This Gist brought to you by gist-it. servoblaster.py view raw

## Appendix A: Adafruit Occidentalis PWM Instructions

The steps to set up the PWM driver are as follows (must be run as root, i.e. with sudo):

1. Set the frequency in the `/sys/class/rpi-pwm/pwm0/frequency` file. Somewhere in the 1000Hz-5000Hz range is recommended
2. Turn on the PWM by writing a value of 1 into the `/sys/class/rpi-pwm/pwm0/active` file

3. Adjust the duty cycle by writing a value between 1-100 in the `/sys/class/rpi-pwm/pwm0/duty` file

## PWM Example Code

```
#!/usr/bin/env python
#
# Demonstrates PWM  control
#
# MUST use pin 18

import time


def togglePwm(enabled):
    value = "0"
    if enabled:
        value = "1"

    with open("/sys/class/rpi-pwm/pwm0/active", "w") as f:
        f.write(str(value))

def setPwmFrequency(freq=1000):
    with open("/sys/class/rpi-pwm/pwm0/frequency", "w") as f:
        print str(freq)
        f.write(str(freq))


# Duty cycle can vary between 1-100
def setPwmDutyCycle(duty):
    if duty <= 0:
        duty = 1
    elif duty > 100:
        duty = 100

    with open("/sys/class/rpi-pwm/pwm0/duty", "w") as f:
        f.write(str(duty))

def pwmTest():
    setPwmFrequency()
    #togglePwm(True)

    while True:
        for i in range(0, 95, 1):
            print "Brightness:", i
            setPwmDutyCycle(i)
            time.sleep(.01)

    togglePwm(False)
```

```
if __name__ == "__main__":
    pwmTest()
```

## Servo Control

Controlling a servo is very similar to controlling PWM. The same comments apply to the use of hardware or software servos as for PWM. The biggest difference is that the Raspberry Pi must be specifically put into servo mode (it defaults to pwm mode, so no change is required for pwm). Then, instead of writing to the "duty" file for control, instead the "servo" file is used. The units of the servo file are in steps from 0-32. A setting of 0 will turn the servo to 0 degrees, while a setting of 32 will turn the servo to 180 degrees. The maximum servo value can be changed by modifying the "max_servo" file, but this is not usually necessary.

### Servo Example

```
#!/usr/bin/env python
#
# Demonstrates Servo control
#
# MUST use pin 18

import time

def setupServo():
    with open("/sys/class/rpi-pwm/pwm0/mode", "w") as f:
        f.write("servo")

def toggleServo(enabled):
    value = "0"
    if enabled:
        value = "1"

    with open("/sys/class/rpi-pwm/pwm0/active", "w") as f:
        f.write(value)


# Duty cycle can vary between 1-180
def setServoPosition(pos):
    if pos <= 0:
        pos = 1
    elif pos > 180:
        pos = 180

    with open("/sys/class/rpi-pwm/pwm0/servo", "w") as f:
        f.write(str(pos))

def servoTest():
    setupServo()
```

```
    toggleServo(True)

    while True:
        for i in range(0, 32, 1):
            print "Servo position (degrees): ",i
            setServoPosition(i)
            time.sleep(.2)

        toggleServo(False)

if __name__ == "__main__":
    servoTest()
```

This Gist brought to you by gist-it. servo_test.py view raw

Eric Friedrich

github.com/raspberrypi-aa
twitter.com/limited00

Slate theme maintained by Jason Costello

Published with GitHub Pages