Raspberry Pi Workshop: Lab Materials

Dario Schor, Troy Denton

October 18, 2013

Contents

Apparatus

The following components are required for this lab

- 1x Raspberry Pi
- 1x SD Card
- 1x USB Micro-B cable
- 1x Workstation (PC) c/w SD card writer
- 1x HDMI to DVI cable
- 1x DVI Monitor
- 1x USB Mouse
- 1x USB Keyboard
- 1x Ethernet Cable (c/w network connection)
- 1x MCP3001 IC
- 3x LEDs
- 1x 10K-Ohm 1/4W resistor
- 3x 220-Ohm 1/4W resistor
- 1x pushbutton switch
- Assorted jumper wires

Exercise 1 - Preparing the OS

The Raspberry Pi is nothing more than a CPU, GPU, and some peripheral devices. To bring the Raspberry Pi into a usable state, it requires an *Operating System* - software that schedules and executes essential routines and user programs. The Raspberry Pi is configured to boot from an SD card - that is to say, it reads a *bootable image* from an SD card. Essentially, it reads in CPU instructions from the SD card that further instruct it to load up the rest of the Operating System software.
To put an Operating System on an SD card, you typically download a compiled image and *flash* it to the SD card. The procedure (for a computer running Windows) is as follows:

1. Download the Raspbian image from `http://downloads.raspberrypi.org/raspbian_latest`
2. Extract the .zip file
3. Install Win32DiskImager
4. Run Win32DiskImager as Adminstrator (from right-click menu)
5. Insert the SD card into your SD card reader, if you have not done so
6. In Win32DiskImager, "Browse" to the extracted .img file
7. Select the SD card reader as the destination device - make sure you don't select your hard drive!

For Linux, Mac OSX, and Unix-variants, the procedure is slightly different:

1. Download the Raspbian image from `http://downloads.raspberrypi.org/raspbian_latest`
2. Extract the .zip file (`unzip -x file.zip`)
3. Insert the SD card if you have not done so
4. use `dmesg | tail` to determine the path to your SD card (should be /dev/sdX)
5. use **sudo** `dd bs=4M if=/path/to/rpi.img of=/dev/(SD Card)` to flash the image - this will take a while!

Finally, plug your Raspberry Pi into a power supply (micro-B USB cable), keyboard, and monitor, and you should see the Raspberry Pi logo and some scrolling text on the screen. If this did not work, please consult one of the volunteers.

Exercise 2 - Shell Basics

This section describes some basic commands for interacting with a Linux shell - specifically, we will be using the *bash* shell. This is the default shell on many popular Linux distributions. If you have not yet logged in to your raspberry pi, do so no - the username is *pi*, and the password is *raspberry*. Plug your raspberry pi into a monitor, power source (USB Cable) and keyboard, and you will eventually be greeted by a login prompt.
Listing 1 gives some examples on "getting around" in the shell. The commands used - for example *cd*, are what you would type at the command-line prompt after logging in to the raspberry pi.

Listing 1: Basic navigation commands

```
#the pwd command lists the directory you are currently 'in'
pwd
```

```
#the cd command changes your current directory
#directories can be absolute
cd /home/pi
#or directories can be relative
cd /
cd home
cd pi

#the ls command lists the files in your current directory
ls
#ls -l gives even more information.
#You can also provide a path that you wish to 'inspect'
ls -l /etc
```

In Linux, Unix, and their variants, everything in the Operating System is a file - everything from programs, to input and output devices, are represented as files in a directory-tree structure. If you have a mouse handy, give Listing 2 a try:

Listing 2: Viewing Mouse Output

```
# make sure the mouse is plugged in before running this command.
# once you run this command, move the mouse to view the output.
# to stop viewing the output, hit Ctrl+C
sudo cat /dev/input/mouse0
```

When using a graphical desktop, the Operating System reads the same file to know when the mouse is moving.
Listing 3 shows some linux commands for basic input and output.

Listing 3: Basic input and output

```
# Anything following the hash symbol (#) is a comment

# the echo command prints text to "Standard output"
echo "lol"

#you can redirect output to a file using the > operator
echo "lol" > /home/pi/lol.txt

#if you want to read the file, you can use the cat command
cat /home/pi/lol.txt
```

Use the cd and ls commands to explore the following directories:

- /
- /etc
- /usr/bin

Exercise 3 - GPIO Output

There are many ways to control GPIO (General Purpose Input/Output) on the Raspberry Pi, the following being the most popular:

- C/C++ libraries
- Python libraries
- Filesystem access

Each method has it's merits - in this lab we will be primarily using the Filesystem methodology due to it's ease of use.
Under the filesystem methodology, if we wish to use a GPIO pin, we must first *export* it. This tells the Operating System to create the necessary files to control the GPIO pin. Listing 4 shows how this is done, using GPIO pin 4 as an example.

Listing 4: Exporting a GPIO pin for use

```
echo 4 > /sys/class/gpio/export
```

A pin need only be exported once to use it. Once exported, GPIO pins can be set up as either an input or an output, but not both. Their configuration as input or output can however change at any time. To configure a particular pin for output, we can use the `echo` command to write to a file - Listing 5 configures pin 4 as an output:

Listing 5: Setting the direction of a GPIO pin

```
echo "out" > /sys/class/gpio/gpio4/direction
```

Once a pin is configured as an output, you can assign it a logic value of 0 or 1 - this will determine the output voltage for that pin (0 or 3.3VDC, respectively). Listing 6 shows examples of setting the output value on pin 4:

Listing 6: Setting an output value on a GPIO pin

```
echo 1 > /sys/class/gpio/gpio4/value
```

Hook up an LED ala Figure 1 and observe the GPIO pin toggling as you `echo` 1 and 0 to the value file:

Exercise 4 - GPIO Input

Similar to our GPIO Output example, we can read the status of a GPIO pin through the filesystem. What we mean by this, is that you can have the raspberry pi detect the presence or absence of a voltage on a particular GPIO pin.
The first step to reading a pin as an input, is to export the pin - so that the Operating System knows to handle it via the filesystem. If you have not yet done so, please run the code in Listing 4 now. To configure a GPIO pin as an input, we simple write the value *in* to its "direction" file, much like we did in our output example. See Listing 7 below:

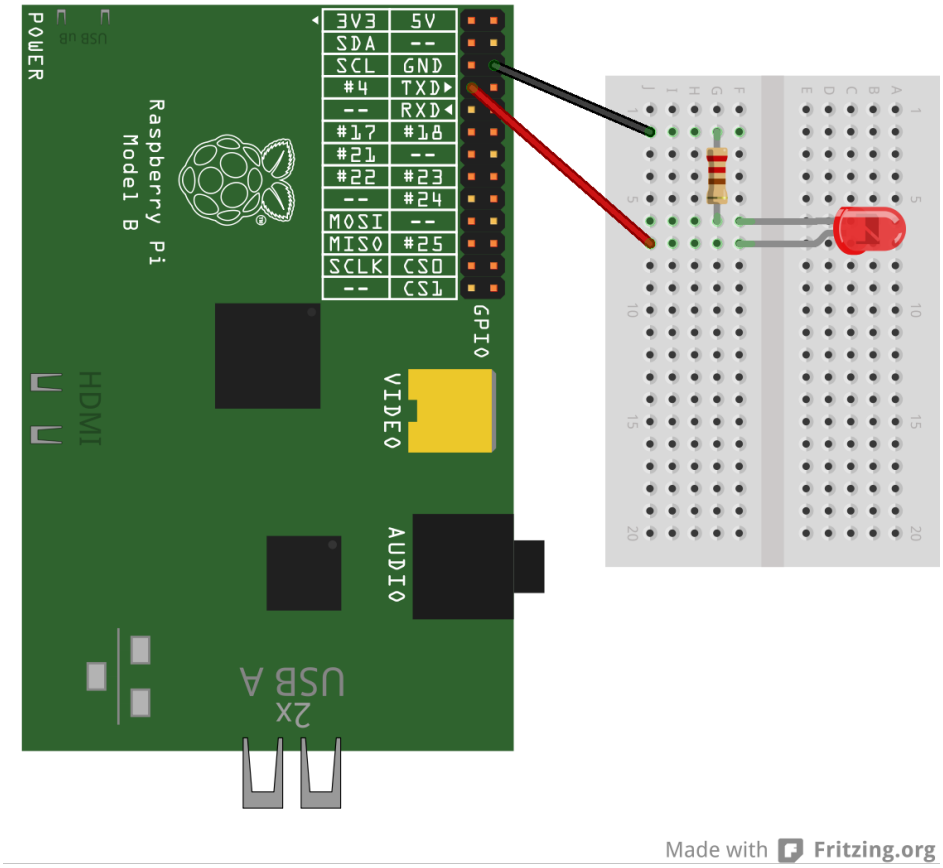

Figure 1: Connecting an LED to GPIO 4

Listing 7: Setting a GPIO pin as an input

```
echo "in" > /sys/class/gpio/gpio4/direction
```

Once the pin is set as an input, you can read the value of the pin by simply reading the "value" file, as shown in Listing 8:

Listing 8: Reading in a value from a GPIO pin

```
cat /sys/class/gpio/gpio4/value
```

Use the circuit diagrammed in 2 to toggle the value of GPIO pin 4, combined with the code in Listing 8 to observe the GPIO working.

Exercise 5 - Creating Shell Scripts

So far, all we've done is run commands on the terminal - but what if we want to have a program control our GPIO pins? One approach is to use what is known as a *Shell Script*, which is simply a series of command-line commands in a file. Take a look at Listing 9 for an overview of a shell script:

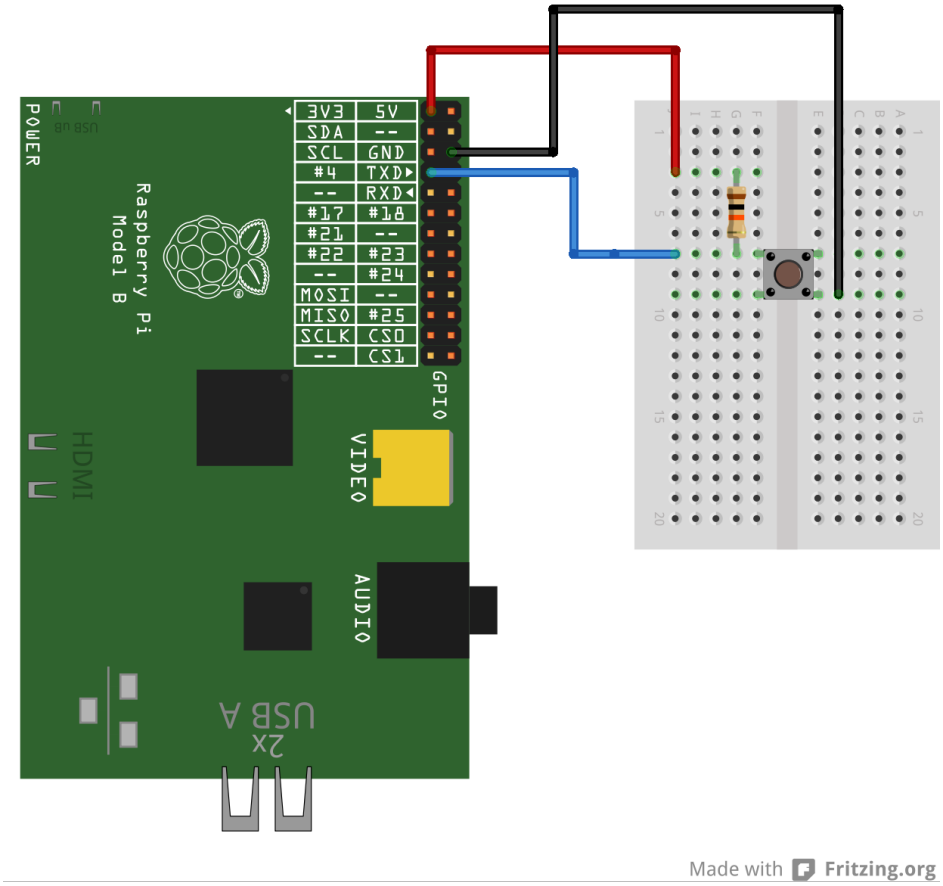

Figure 2: Connecting a switch to GPIO 4

Listing 9: Basic shell script examples

```
#!/bin/bash
# if the first line starts with #!, the following path lists the
# executable file used to run the rest of the file - this is also
# known as the 'interpreter'.  In this case we are using bash -
# which is also the program used when you are interacting with the
# shell.  This allows us to use command-line commands in our program

# lines beginning with a # symbol - without a ! symbol - are comments,
# and aren't run as part of the program!

echo "testing.."
sleep 2
echo "some more output..."
sleep 2
echo "writing this to a file..." > /home/pi/test.txt
sleep 2

# variables in bash script files are created, assigned, and used as so:
TESTVARIABLE=15
echo "the contents of TESTVARIABLE: $TESTVARIABLE"
sleep 2
```

```
# the if [] command lets us perform conditional statements
# the -e switch is used to check if a file exists
# to see what other options there are for the if[] command,
# refer to the man page for 'test' (run 'man test' in the terminal)
if [ -e /home/pi/test.txt ]; then


# if we place commands within $( ), this lets us execute that command
# and use the result
     TESTVARIABLE=$(cat /home/pi/test.txt)
     echo "the contents of TESTVARIABLE: $TESTVARIABLE"
fi

# we can use the ! operator instead the if[] statement to make it test
# the opposite - in this case, it's chekcing if /home/pi/test does
# not exist
if [ ! -e /home/pi/test.txt]; then
     echo "/home/pi/text.txt does not exist!"
fi

sleep 2


# the mkdir command creates a new directory
mkdir /home/pi/testdir
for i in $( seq 1 10 )
do

#you can embed a variable directly into filenames and directories
     echo "test number $i..." > /home/pi/testdir/test$1.txt

done
```

To create a shell script, simply open a new file and start typing! To edit and create a new file, the `nano` program is recommended. `cd` to your *home directory* (/home/pi) and run `nano test.sh` to start editing a new file named test.sh, stored at */home/pi/test.sh*. Enter some of the commands from 9 to start experimenting. To run your script, you will first have to add *execution permission* to your script. Execute `chmod +x /home/pi/test.sh` to tell the Operating System that this file is meant to be run as a program. Then, enter `/home/pi/test.sh` on the command line, which will actually run your program. Note that you only need to add the execution permission once.
Now that we have some shell scripting experience under our belts, lets start building some tools to use in later examples:

1. Hook up the circuit in Figure 1. Create the script /opt/workshop/gpio_out.sh. Have it accept two arguments on the command line: the first argument is the GPIO pin to modify, and the second argument is the value to set it to. For example: **sudo** `/opt/workshop/gpio_out.sh 4 1` will set GPIO pin 4 to value 1. Hint: the first argument is stored in the variable `$1`, and the second argument is stored in `$2`.

2. Hook up the circuit in Figure 2. Create the script /opt/workshop/gpio_in.sh. Have it accept one argument on the command line - the GPIO pin to read. For example: `/opt/workshop/gpio_in.sh 4`

should return '0' or '1' as appropriate.

3. Modify both scripts so that they export the GPIO pin before doing any reading or writing, **but only if it has not been exported yet! Hint: If the pin has already been exported, the directory /sys/class/gpio/gpioN will already exist, where N is the pin number.**

Don't forget to **sudo** `chmod +x /opt/workshop/gpio_out.sh`, as well as for `gpio_in.sh`!
Note that you will have to prefix your commands with **sudo** to edit and run files in */opt*. For example: **sudo** `nano /opt/workshop/gpio_in.sh`, and **sudo** `/opt/workshop/gpio_in.sh 4`. This is because your user - *pi* - does not regularly have the account permissions to manipulate files in */opt*. **sudo** bypasses this by temporarily giving you permission, provided you can supply your password.

Exercise 6 - Installing Apache

For those of you familiar with Ubuntu, or other Operating Systems with a built-in package manager, you'll be glad to know that the Raspbian Operating System (What we are running on our Raspberry Pi's) also features a package manager! To complete further examples, we need to install the *apache* webserver. Luckily, this only takes a few commands with our package manager (Listing 10). The update may take a while, this might be a good time to get up and stretch your legs...

Listing 10: Installing apache

```
#this command updates our local list of available packages and updates
sudo apt-get update

#this command installs the actual updates
sudo apt-get upgrade

#this command installs apache for us
sudo apt-get install apache2

sudo apt-get install php5
```

We want apache to be able to run the scripts we created - */opt/workshop/gpio_in.sh* and */opt/workshop/gpio_out.sh* - so we need to give the user that runs apache the proper permissions to do so. This is accomplished by running the command *sudo visudo* and adding the following line to the bottom of the file that comes up for editing:

```
www-data ALL=(root)NOPASSWD: /opt/workshop/*
```

Note: hit Ctrl+O to save the file (output file), and Ctrl+x to exit.

Exercise 7 - PHP and AJAX Calls

PHP is a scripting language that is easily integrated with apache. PHP is widely used for AJAX - *Asynchronous Javascript and XML* - requests. Although the name suggests usage of XML, it is not required - AJAX simply refers to asynchronous web traffic.
If we want our PHP scripts to take arguments - similar to what we did to read variables in bash scripts - we need to introduce POST variables. POST variables allow a web browser, or any HTTP client to send pieces of data that have a name and a value. These are generally known as *key-value* pairs.
Consider the following code listings:

Listing 11: Basic PHP backend script

```php
<?php

#the 'NAME' variable we send via $.post is accessed as so:
echo "Hello, ".$_POST["NAME"]."!\r\n";

#the 'exec' command allows php to run a shell command, and return the result
echo "The current date is: ".exec(date)."\r\n";


?>
```

Listing 12: Basic javascript-based AJAX client

```html
<html>
<head>
<title>Test</title>
<script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jquery
    /1.10.2/jquery.min.js"></script>
<script type="text/javascript">

function buttonClick()
{
     $.post("ex_7.php",{

          NAME: "TROY"

     }).done(function(result){

          alert("The server returned: " + result);

     });
}

</script>
</head>
<body>
<button onClick=buttonClick()>Click me!</button>
</body>
</html>
```

When the user clicks the button labelled "Click me!", the button's *onClick* function gets executed. This function uses the jQuery library to perform a POST to test.php, and alerts the user with the response in a pop-up box. You may find the javascript syntax odd if you've never encountered it - we will not have you perform javascript development in this lab, as such no explanation will be listed here. The instructors will gladly answer any questions you have during or after the laboratory session. Note: jQuery is a popular javascript library that provides easy-to-use AJAX functions, among many, many more things!

1. Perform the following commands to test out the AJAX example:

Listing 13: Installing AJAX example

```
cd /var/www
sudo wget http://troydenton.ca/rpi/ex7.tar.gz
sudo tar xvzf ex7.tar.gz
sudo chown www-data:www-data *
```

Use the `ifconfig eth0` command to determine your Raspberry Pi's IP address, and navigate to http://*IP address*/ex_7-1.html in a web browser.

2. Write a PHP script - /var/www/gpio_out.php - that accepts the POST variables STATE and PIN, and sets GPIO Pin #PIN to state STATE, where STATE is '1' or '0'. Test this using the 'GPIO Output' tab of http://*IP address*/index.php

3. Write a PHP script - /var/www/gpio_in.php - that accepts the POST variable PIN, and returns the logic value of pin #PIN. Test this using the 'GPIO Input' tab of http://*IP address*/index.php

Hint: to use the exec() call with our /opt/workshop scripts, we will need to use sudo. The -t option can be used to bypass the password prompt, if configured properly via visudo.

Exercise 8 - SPI

In order to read analog ports using the Raspberry Pi, one needs additional hardware to convert the analog readings and then transmit that to the processor. Microchip's MCP3001 is a small analog-to-digital converter (ADC) integrated chip with 10-bit resolution that can convert a value and transmit it to a processor using the Serial Peripheral Interface (SPI) bus.

SPI

SPI is a serial synchronous communication protocol developed by Motorola. This master/slave protocol uses 4 wires as described below:

- SCLK - serial clock (sent from master)
- MOSI - master output / slave input (sent from master)
- MISO - master input / slave output (sent from slave)
- CS - chip select or slave select

The protocol can work with one or more slaves and expand to more CS pins as needed. In essence, it operates as a full duplex protocol controlled by the master. When the master wants to send a byte, it generates an 8-bit clock used to synchronize the transmission. Then the data is sent serially by shifting out bits from the LSB to the MSB. In this lab, we will be reading a byte. To do that, the master will send a dummy byte that generates the clock signal to send the data. This is shown graphically in Fig. 3.

The Raspberry Pi has a set of dedicated SPI pins, but unlike the GPIOs, there are no commands to access these pins. One option would be to write a full application in C, C++, Python, or other language. In this workshop, we will utilize a pre-existing application called **spincl**. To install this tool, run the commands from the following listings.

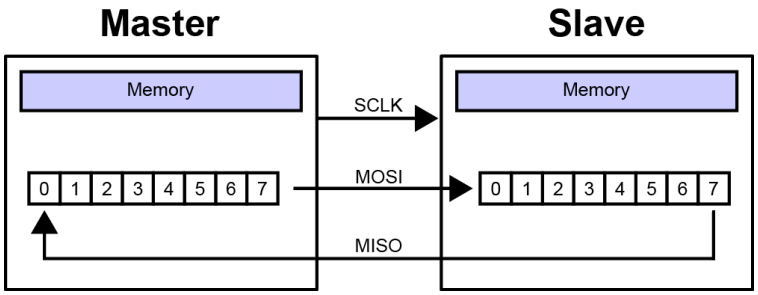

Figure 3: SPI Protocol (credit: Wikipedia)

Listing 14: Enable SPI pins.

```
# open an editor called nano to edit the configuration file to manage serial ports
sudo nano /etc/modprobe.d/raspi-blacklist.conf

# use the arrow keys to move around and put a '#' in front  of both blacklists
# when you are done editing, hit CTRL+O to save, then CTRL+X to exit
```

Listing 15: Install spincl.

```

# go to the workshop directory
cd /opt/workshop

#install dependancies for spincl
sudo wget http://www.airspayce.com/mikem/bcm2835/bm2835-1.30.tar.gz
sudo tar xvzf bcm2835-1.30.tar.gz
cd bcm2835-1.3.0
sudo ./configure
sudo make
sudo make install


# download the spincl application source code
sudo wget http://ipsolutionscorp.com/?ddownload=650 --output-document=spincl.tar.
    gz

# decompress the application files
sudo tar -xzvf spincl.tar.gz

# go into the spincl directory
cd spincl

# clean the installation (it is better to recompile for your platform)
sudo make clean

# compile the application
sudo make

# run application to see help menu and instructions on how it works
./spincl
```

Circuit

Build the circuit shown in Figures 4 and 5 and try reading the analog reading using **spincl** with SPI mode 0, clock divider 15, chip select 0, and chip polarity low. Remember that this is a 10-bit converter, therefore you need to read 2 bytes to get the full value. Once this works on the command line, write a full shell script to read and print two bytes.

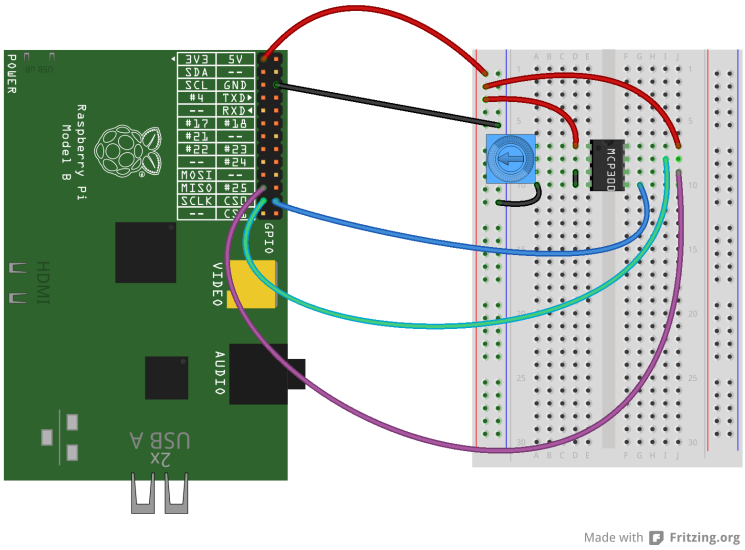

Figure 4: MCP3001 & Raspberry Pi circuit.

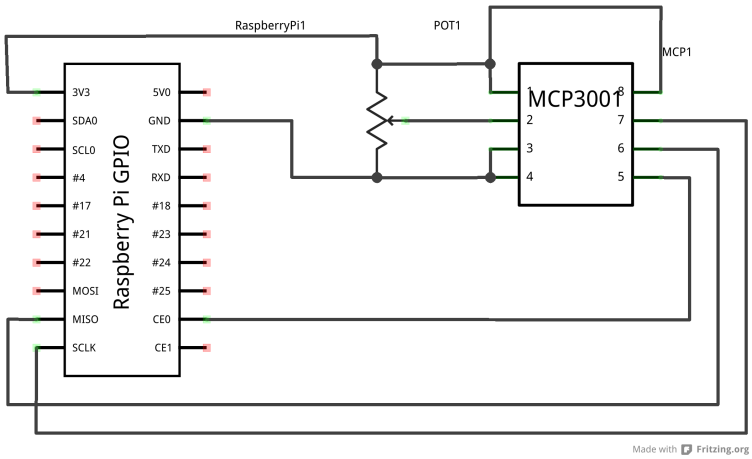

Figure 5: MCP3001 & Raspberry Pi circuit.

Web Interface for ADC

To integrate this with the web interface you will need to write **/var/www/spi_in.php**. Use the PHP function hexdec() to convert the values to numbers and then shift the bits to read the ten bits in the middle of the 2 byte transmission (i.e., XXX0123456789XXX).

Exercise 9 - Python and GPIO

Python is an interpreted language that has grown very popular in recent years - it aims to have clear and concise syntax, without comprimising on ability. Many open-source software products use python because of these characteristics - it is heavily used in Ubuntu, as well as Raspbian!
Python itself comes with the Raspbian Operating System, but if weant to control GPIO pins, we need to install a software library:

Listing 16: Installing the RPi.GPIO Python Library

```
cd /home/pi
wget http://pypi.python.org/packages/source/R/RPi.GPIO/RPi.GPIO-0.5.3a.tar.gz
tar xvzf RPi.GPIO-0.5.3a.tar.gz
cd RPi.GPIO-0.5.3a
sudo apt-get install python-dev
sudo python setup.py install
```

Hook up the circuit described in Figure 6 to use with the following code listing:

Listing 17: Controlling a GPIO pin with python

```
import RPi.GPIO as GPIO
import time

try:
     # Set up the GPIO channels

     # This sets the pin naming conventions to agree
     # with what we have been using so far
     GPIO.setmode(GPIO.BCM)


     GPIO.setup(18, GPIO.OUT)


     # Output to pin 18
     while True:
          GPIO.output(18, True)
          time.sleep(1)
          GPIO.output(18, False)
          time.sleep(1)

except KeyboardInterrupt:
     GPIO.cleanup()
```

If you save the file in your home directory as 'blink1.py', you can run the program with **sudo** `/home/pi/blink1.py`. The program will stop when you hit CTRL+C.

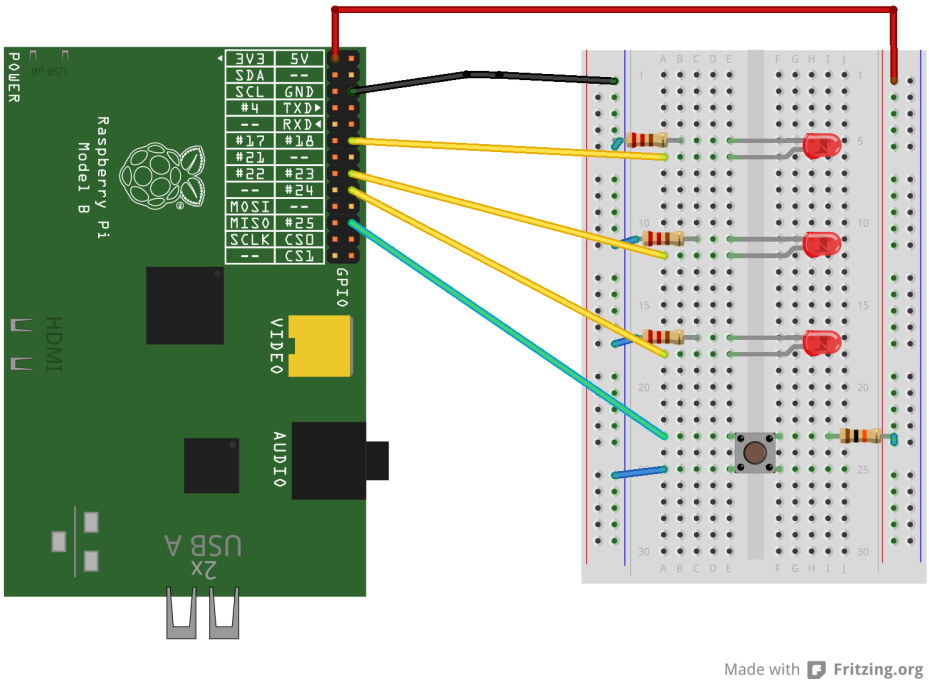

Figure 6: Connecting Multiple GPIO Pins

1. Enter the example program and observe the output - make the LED blink twice as fast
2. Make the program blink the 3 LEDs in sequence
3. Listing 18 demonstrates how to read an input in Python - make the program blink the 3 LEDs in sequence when a user presses the button

Listing 18: Reading an Input in Python

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


print "Hit Ctrl+C to stop...";
try:
     # Set up the GPIO channels - 3 outputs, one input
     GPIO.setmode(GPIO.BCM)
     GPIO.setup(25, GPIO.IN)

     while True:

          #do nothing while the button is not pushed.
          #ie, this will loop forever until GPIO.input(25) becomes 0
          while (GPIO.input(25) == 1):
               pass
          print "Ding!";
```

```
except KeyboardInterrupt:
    GPIO.cleanup()
```

Exercise 10 - Python and SPI

Python can also be used to read SPI, given the proper libraries. You can install the spidev libraries by following Listing 19.

Listing 19: Installing spidev

```
cd /home/pi
sudo apt-get install git
git clone git://github.com/doceme/py-spidev.git
cd py-spidev
make
sudo make install

#this needs to be run before running python SPI examples,
# if you have been running the spincl program previously
sudo rmmod spidev
sudo rmmod spi_bcm2708
sudo modprobe spi_bcm2708
```

This is only one way to access the SPI bus in python - there are likely better ways. But this will work for our experimenting purposes today. See listing 20 for the code, and Figure 4 for the connection diagram.

Listing 20: Reading the ADC with SPIDEV

```
#!/usr/bin/python

import spidev
import time

spi = spidev.SpiDev()
spi.open(0,0)  #open spi channel 0
spi.mode = 0   #mode (0,0) = 0, (0,1) = 1, (1,0)  = 2, (1,1) = 3



def read_mcp3001():
    reading = spi.readbytes(2)
    #    format of bytes read in (2 bytes = 16 bits):
    #    XXX0123456789XXX  relevant bits
    #    0001111111111000  bitmask  (= 0x1ff8)
    hi = reading[0] & 0x1f
    lo = reading[1] & 0xf8
    vin = (hi<<5) + (lo>>3)
    vin = vin * 3.3/1024     #VREF = 3.3, 10 bit adc gives us (2^10 = 1024) steps
        .
    return vin
```

```
try:
     #enter an infinite loop while, and continuously output values
     while True:
          reading = read_mcp3001() #get a reading!
          print "vin: %fV" % reading    #output the reading
          time.sleep(0.5)               #wait for half a second

except KeyboardInterrupt:
     spi.close()
```

1. Implement Listing 20 and Figure 4 - adjust the potentiometer while the program is running and observe the output
2. Implement thresholds - have the program print “Too High” for voltages over 3.0VDC, and “Too Low” for voltages under 1.0VDC
3. Hook up some LEDs - ala earlier examples - and have them turn on when the voltage exceeds 3.0VDC. Have them turn off when the voltage falls below 1.0VDC
4. Hook up a row of LEDs, and have them display the magnitude of the voltage like a ‘Bar Graph’

   This concludes the prepared material - thanks for coming out!