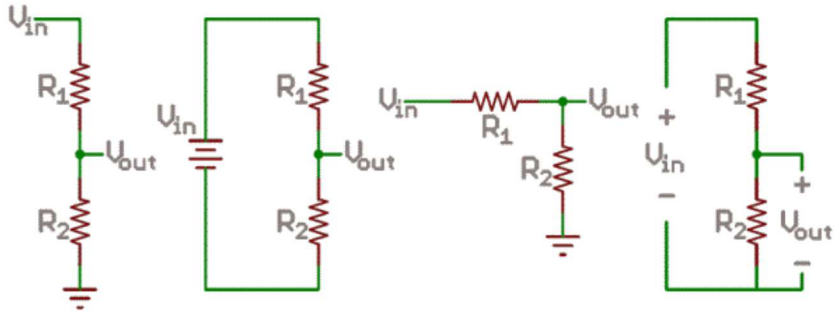
MCP3208 voltage measurement

If you use a MCP3208 (http://ww1.microchip.com/downloads/en/devicedoc/21298c.pdf) ADC one need to add a voltage divider to measure voltages higher than the V_{dd} V_{ref}. This can be done by adding two resistors to the input of the MCP3208. How to calculate see below.

If you want to add an extra security to for instance the Rasberry PI to avoid over voltage (the RPi does not like higher voltages than 3.3v) a zener diode can be added but that will influence the readout as a zener diode will also have some additional current flow. An option to compensate is than to add in the program you use some sort of compensation.

The Circuit

A voltage divider involves applying a voltage source across a series of two resistors. You may see it drawn a few different ways, but they should always essentially be the same circuit.



Examples of voltage divider schematics. Shorthand, longhand, resistors at same/different angles, etc.

We'll call the resistor closest to the input voltage (V_{in}) R_1, and the resistor closest to ground R_2. The voltage drop across R_2 is called V_{out}, that's the divided voltage our circuit exists to make.

That's all there is to the circuit! V_{out} is our divided voltage. That's what'll end up being a fraction of the input voltage.

The Equation

The voltage divider equation assumes that you know three values of the above circuit: the input voltage (V_{in}), and both resistor values (R_1 and R_2). Given those values, we can use this equation to find the output voltage (V_{out}):

$$V_{out} = V_{in} \cdot \frac{R_2}{R_1 + R_2}$$

This equation states that the output voltage is **directly proportional** to the **input voltage** and the **ratio of R_1 and R_2**.

Calculator

Have some fun experimenting with inputs and outputs to the voltage divider equation!
Below, you can plug in numbers for V_{in} and both resistors and see what kind of output voltage they produce.

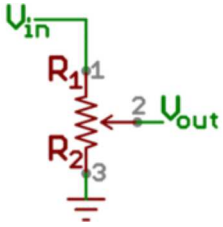
V_{in} = 5 **V**
R_1 = 1700 **Ω**
R_2 = 3300 **Ω**
V_{out} = 3.30 **V**

Or, if you adjust V_{out}, you'll see what resistance value at R_2 is required (given a V_{in} and R_1).

Potentiometers

A potentiometer is a variable resistor which can be used to create an adjustable voltage divider.

Internal to the pot is a single resistor and a wiper, which cuts the resistor in two and moves to adjust the ratio between both halves. Externally there are usually three pins: two pins connect to each end of the resistor, while the third connects to the pot's wiper.



A potentiometer schematic symbol. Pins 1 and 3 are the resistor ends. Pin 2 connects to the wiper.

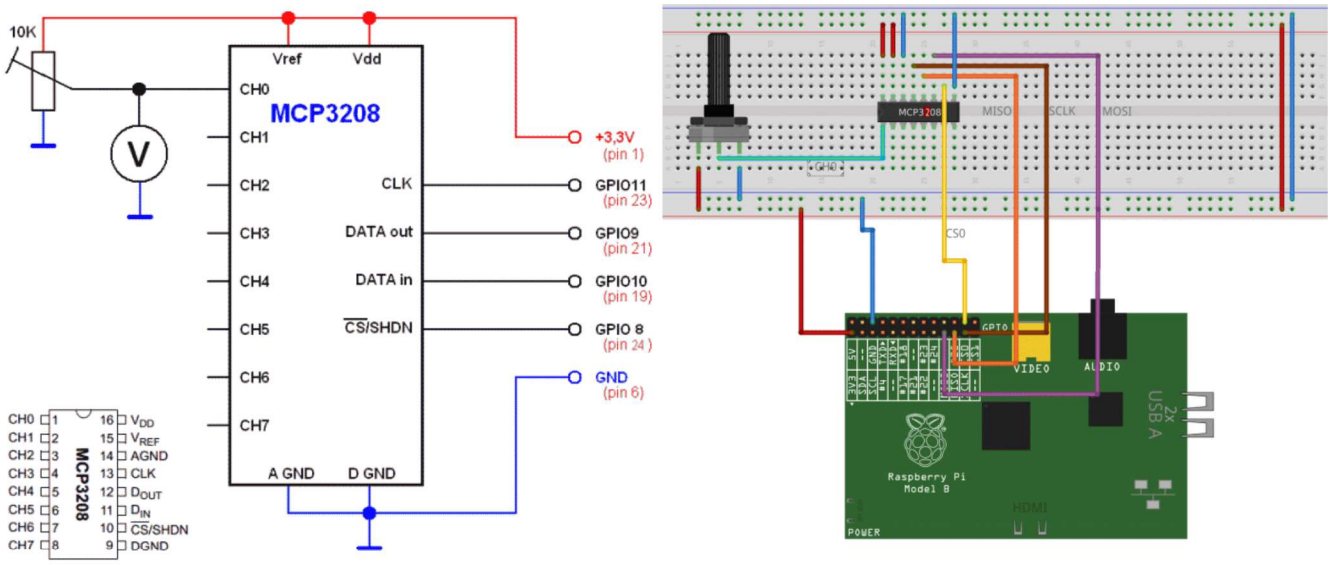
If the outside pins connect to a voltage source (one to ground, the other to V_{in}), the output (V_{out} at the middle pin will mimic a voltage divider. Turn the pot all the way in one direction, and the voltage may be zero; turned to the other side the output voltage approaches the input; a wiper in the middle position means the output voltage will be half of the input.

Potentiometers come in a variety of packages, and have many applications of their own. They may be used to create a reference voltage, adjust radio stations, measure

position on a joystick, or in tons of other applications which require a variable input voltage.

Interface M3208 to the RPi

In this case a potentiometer is used so its easy to play with the different values as with turing the knob it will varry from 0 to V_{dd}



Enable SPI on your Raspberry PI

From the command line you need to edit the following file:
/etc/modprobe.d/raspi-blacklist.conf
After you edit it with your editor jed/nano/vi it should look like this:

---
blacklist spi and i2c by default (many users don't need them)

blacklist spi-bcm2708
blacklist i2c-bcm2708

In most cases it is needed to put a **#** in front of the lines to rem these values out so these drivers are loaded after a next reboot.

A reboot is required or you need to activate it via the **modprobe spi_bcm2708** command
On my PI if you type **lsmod** on the command line it gives:

```
Module                  Size  Used by
spi_bcm2708             4808  0
w1_therm                2870  0
w1_gpio                 2747  0
wire                   25249  2 w1_gpio,w1_therm
cn                      4795  1 wire
nfnetlink_log           8632  0
nfnetlink               5144  1 nfnetlink_log
sg                     19328  0
pl2303                  8727  0
ark3116                 5943  0
usbserial              26435  2 pl2303,ark3116
cpufreq_stats           3692  0
rtc_ds1307              7715  0
bcm2708_wdog            3613  0
i2c_dev                 5277  2
snd_bcm2835            18169  0
snd_soc_pcm512x         8909  0
```

Source C++

For this to work you need first to install WiringPi from here (http://wiringpi.com/download-and-install/) and follow all instructions properly.

```
/*
 * Save as spi-test.c
 * Compile with: gcc -o spi-test spi-test.c -lwiringPi
 *
 * If no value:
 * rmmod spi_bcm2708
 * modprobe spi_bcm2708
 *
 * http://www.icbanq.com/pbloger/board_View.aspx?number=269 (http://www.icbanq.com/pbloger/board_View.aspx?number=269)
 * http://www.raspberrypi.org/forums/viewtopic.php?f=93&t=78551 (http://www.raspberrypi.org/forums/viewtopic.php?f=93&t=78551) (baart)
 * http://www.mikroe.com/add-on-boards/measurement/adc-proto/ (http://www.mikroe.com/add-on-boards/measurement/adc-proto/) (boards)
 *
 */

#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <stdlib.h>

#include <wiringPi.h>
#include <wiringPiSPI.h>

#define CS_MCP3208  8       // BCM_GPIO8

#define SPI_CHANNEL 0
#define SPI_SPEED   100000  // !! Start low here and if all works try to increase if needed on a breadboard I could go upto about 750000


int read_mcp3208_adc(unsigned char adcChannel)
{
  unsigned char buff[3];
  int adcValue = 0;

  buff[0] = 0x06 | ((adcChannel & 0x07) >> 7);
  buff[1] = ((adcChannel & 0x07) << 6);
  buff[2] = 0x00;

  digitalWrite(CS_MCP3208, 0);  // Low : CS Active

  wiringPiSPIDataRW(SPI_CHANNEL, buff, 3);

  buff[1] = 0x0F & buff[1];
  adcValue = ( buff[1] << 8) | buff[2];

  digitalWrite(CS_MCP3208, 1);  // High : CS Inactive

  return adcValue;
}


int main (void)
{
  int adc1Channel = 0;
  int adc1Value   = 0;
  int adc2Channel = 1;
  int adc2Value   = 0;
  int adc3Channel = 2;
  int adc3Value   = 0;
  int adc4Channel = 3;
  int adc4Value   = 0;
  int adc5Channel = 4;
  int adc5Value   = 0;
  int adc6Channel = 5;
  int adc6Value   = 0;
  int adc7Channel = 6;
  int adc7Value   = 0;
  int adc8Channel = 7;
  int adc8Value   = 0;

  if(wiringPiSetup() == -1)
  {
    fprintf (stdout, "Unable to start wiringPi: %s\n", strerror(errno));
    return 1 ;
  }

  if(wiringPiSPISetup(SPI_CHANNEL, SPI_SPEED) == -1)
```

```
{
  fprintf (stdout, "wiringPiSPISetup Failed: %s\n", strerror(errno));
  return 1 ;
}

pinMode(CS_MCP3208, OUTPUT);

while(1)
{
  system("clear");
  printf("\n\nMCP3208 channel output.\n\n");
  adc1Value = read_mcp3208_adc(adc1Channel);
  printf("adc0 Value = %04u", adc1Value);
  printf("\tVoltage = %.3f\n", ((3.3/4096) * adc1Value));
  adc2Value = read_mcp3208_adc(adc2Channel);
  printf("adc1 Value = %04u", adc2Value);
  printf("\tVoltage = %.3f\n", ((3.3/4096) * adc2Value));
  adc3Value = read_mcp3208_adc(adc3Channel);
```

Python code

For this to work you need to install the Python GPIO package from here (http://sourceforge.net/projects/raspberry-gpio-python/files/latest/download) and install in to your RPi.

Working on my Python 2.7.3 version.

```
#!/usr/bin/env python
#
#
import time
import os
import sys
import RPi.GPIO as GPIO
from decimal import *
import atexit
import signal
import subprocess

CONTROL_C = False

def program_exit():
    # You may do some clean-up here, but you don't have to.
    print "\n"
    print "Exiting application... Thnxs                                                "
    GPIO.cleanup()
    subprocess.call('setterm -cursor on', shell=True)
    subprocess.call('spincl -ib', shell=True)
    print " "

def ctrlCHandler(*whatever):
    # Just sets the value of CONTROL_C
    global CONTROL_C
    CONTROL_C = True

THREEPLACES = Decimal(10) ** -3

GPIO.setwarnings(False)

GPIO.setmode(GPIO.BCM)

subprocess.call('setterm -cursor off', shell=True)
subprocess.call('spincl -ib', shell=True)

# read SPI data from MCP3208 chip, 8 possible adc's (0 thru 7)
def readadc(adcnum, clockpin, mosipin, misopin, cspin):
        if ((adcnum > 7) or (adcnum < 0)):
                return -1
        GPIO.output(cspin   , True)

        GPIO.output(clockpin, False)  # start clock low
        GPIO.output(cspin   , False)  # bring CS low

        commandout = adcnum
        commandout |= 0x18            # start bit + single-ended bit
        commandout <<= 3              # we only need to send 5 bits here
        for i in range(5):
                if (commandout & 0x80):
                        GPIO.output(mosipin, True )
                else:
                        GPIO.output(mosipin, False)
                commandout <<= 1
                GPIO.output(clockpin, True )
                GPIO.output(clockpin, False)

        adcout = 0

        # read in one empty bit, one null bit and 12 ADC bits
        # pro desetibitovy prevodnik tu bylo puvodne cislo 12
        for i in range(14):
                GPIO.output(clockpin, True )
                GPIO.output(clockpin, False)
                adcout <<= 1
                if (GPIO.input(misopin)):
                        adcout |= 0x1

        GPIO.output(cspin, True)

        adcout >>= 1 # first bit is 'null' so drop it
        return adcout

# change these as desired - they're the pins connected from the SPI port on the ADC to the RPi
SPICLK  = 11
```

```
SPIMISO = 9
SPIMOSI = 10
SPICS   = 8

# set up the SPI interface pins

GPIO.setup(SPIMOSI, GPIO.OUT)
GPIO.setup(SPIMISO, GPIO.IN )
GPIO.setup(SPICLK,  GPIO.OUT)
GPIO.setup(SPICS,   GPIO.OUT)

# 10k trim pot connected to adc #0
potentiometer_adc = 0;

subprocess.call('clear', shell=True)

# You must check CONTROL_C in your program
```

Here (http://ipsolutionscorp.com/raspberry-pi-spi-utility/) I found a SPI utility that communicates from the command line. **spincl -ib** will reset the SPI properly.
Its needed as Python GPIO somehow locks up the bcm2708_spi.0 making the C code program not to work after running. Still investigating why.

If someone has any questions you can contact me here (/contact).
Also if you have improvements on the above coding I would be more than happy to include them.