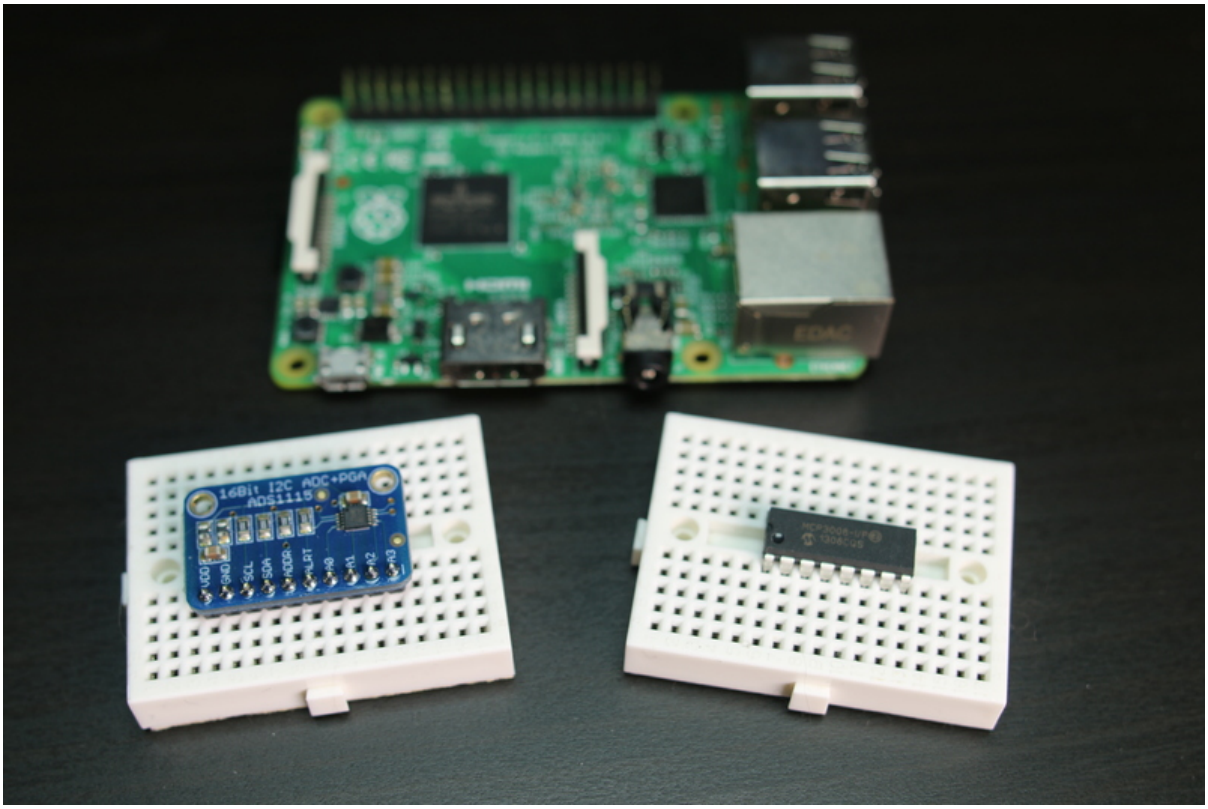
Raspberry Pi Analog to Digital Converters

Created by Tony DiCola



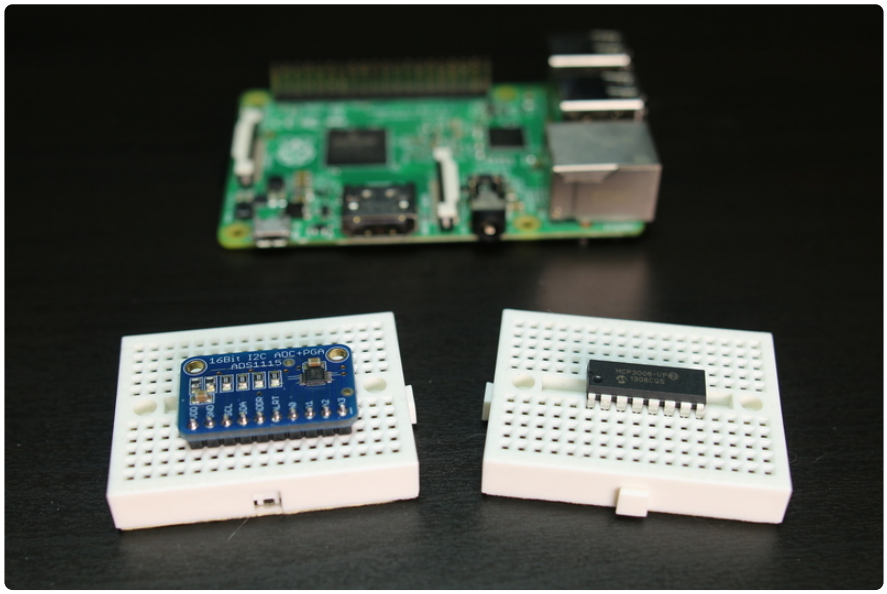
https://learn.adafruit.com/raspberry-pi-analog-to-digital-converters

Last updated on 2024-06-03 01:52:30 PM EDT

Table of Contents

Overview

The examples in this guide are no longer supported. Check out the MCP3008 and ADS1x15 guides for CircuitPython and Python usage.



The Raspberry Pi is an excellent small board computer that you can use to control digital inputs & outputs. However what do you do when you want to read an analog signal, like what you might get from a thermistor, potentiometer, or many other types of sensors? Don't give up! By connecting a small analog to digital converter (ADC) chip to the Pi you can open up the world of analog signals to your Raspberry Pi programs!

This guide will show you a couple great options for reading analog values from Python with a Raspberry Pi. You can use a simple MCP3008 (http://adafru.it/856) analog to digital converter (ADC) to read up to 8 channels of analog input with 10-bit precision. Or use a fancier ADS1x15 (https://adafru.it/lfD) series ADC to read 4 channels with 12 to 16-bit precision, and a programmable gain stage--wow, fancy! You'll be up and running in no time with a Python library and examples to read analog inputs from both these ADCs.

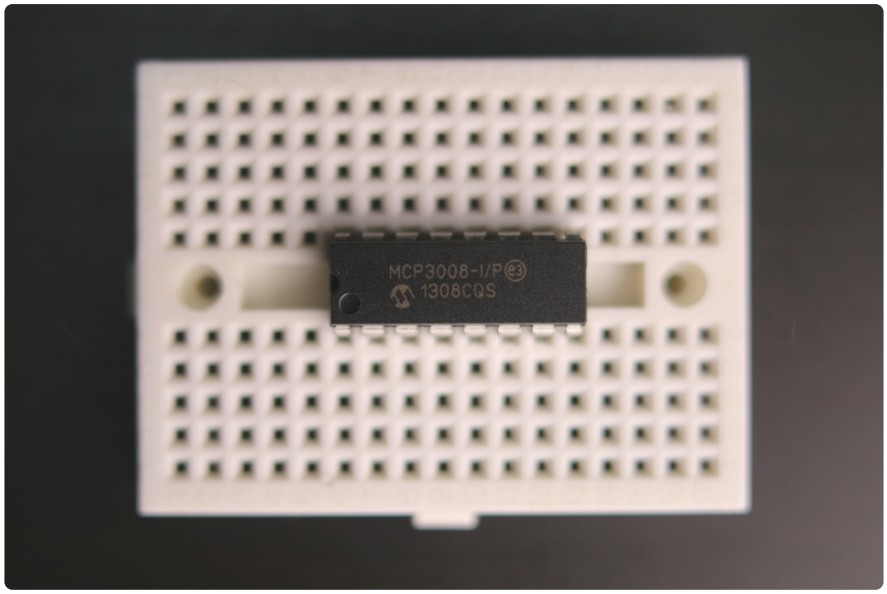
Before you get started you'll want to be familiar with the basics of using a Raspberry Pi like loading an operating system, setting up networking, and connecting to a terminal using SSH. Check out the Learn Raspberry Pi series (https://adafru.it/jcW) for more information.

Also make sure your Raspberry Pi is running the latest Raspbian Jessie (https://adafru.it/fQi) operating system (either the full or lite version) before starting the guide.

Continue on to learn about using the MCP3008 (https://adafru.it/lfE), or the ADS1015 / ADS1115 (https://adafru.it/lfF) analog to digital convert with the Raspberry Pi.

MCP3008

The examples in this guide are no longer supported. Check out the MCP3008 guide for CircuitPython and Python usage: https://learn.adafruit.com/mcp3008-spi-adc/python-circuitpython



The MCP3008 (http://adafru.it/856) is a low cost 8-channel 10-bit analog to digital converter. The precision of this ADC is similar to that of an Arduino Uno, and with 8 channels you can read quite a few analog signals from the Pi. This chip is a great option if you just need to read simple analog signals, like from a temperature or light sensor. If you need more precision or features, check out the ADS1x115 series on the next page.

Before you use the MCP3008 it will help to skim this older Raspberry Pi MCP3008 guide (https://adafru.it/lfG) for more information about using it with the Raspberry Pi. However **don't** use the code from the older guide as it's deprecated. This guide will show you an easier way to install and use new Python code to talk to the MCP3008 ADC.

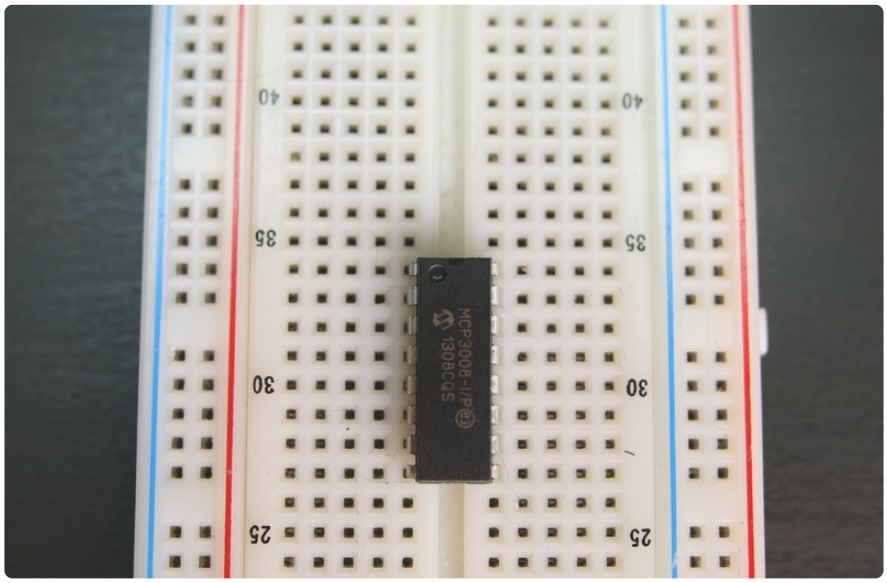
The MCP3008 datasheet (https://adafru.it/aHE) is also an important resource to skim and have handy.

Wiring

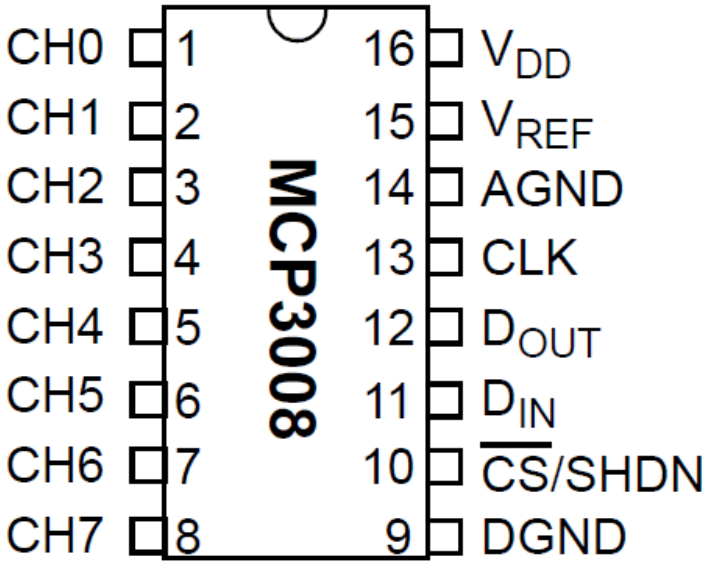
The MCP3008 connects to the Raspberry Pi using a SPI serial connection. You can use either the hardware SPI bus, or any four GPIO pins and software SPI to talk to the MCP3008. Software SPI is a little more flexible since it can work with any pins on the Pi, whereas hardware SPI is slightly faster but less flexible because it only works with specific pins. If you aren't sure which to use I recommend software SPI as it's easier to setup.

Before you can wire the chip to the Pi you first need to place it into a breadboard. If you haven't used bare DIP chips like the MCP3008 before you want to press it into the breadboard so its legs straddle the empty channel in the middle of the breadboard. This way you can access each of the legs of the chip from the breadboard.

Note that the orientation of the chip matters! Be sure to place it with the half circle indention and dot towards the top. See the photo below for an example:



Once the chip is in the breadboard then you're ready to connect it to the Pi. Each of the legs of the MCP3008 chip have the following names:



Remember the orientation of the chip matters and you MUST have it with the half circle indentation towards the top like the diagram above!

Software SPI

To connect the MCP3008 to the Raspberry Pi with a software SPI connection you need to make the following connections:

- **MCP3008 VDD** to **Raspberry Pi 3.3V**
- **MCP3008 VREF** to **Raspberry Pi 3.3V**
- **MCP3008 AGND** to **Raspberry Pi GND**
- **MCP3008 DGND** to **Raspberry Pi GND**
- **MCP3008 CLK** to **Raspberry Pi pin 18**
- **MCP3008 DOUT** to **Raspberry Pi pin 23**
- **MCP3008 DIN** to **Raspberry Pi pin 24**
- **MCP3008 CS/SHDN** to **Raspberry Pi pin 25**

Note that you can swap the MCP3008 CLK, DOUT, DIN, and CS/SHDN pins to any other free digital GPIO pins on the Raspberry Pi. You'll just need to modify the example code to use your pins.

Hardware SPI

To use hardware SPI first make sure you've enabled SPI using the raspi-config tool (https://adafru.it/lfH). Be sure to answer yes to both enabling the SPI interface and loading the SPI kernel module, then reboot the Pi. Check you can see a /dev/spidev0.0 and /dev/spidev0.1 device when you run the **ls -l /dev/spi*** command before continuing.

Now wire the MCP3008 to the Raspberry Pi as follows:

- **MCP3008 VDD** to **Raspberry Pi 3.3V**
- **MCP3008 VREF** to **Raspberry Pi 3.3V**
- **MCP3008 AGND** to **Raspberry Pi GND**
- **MCP3008 DGND** to **Raspberry Pi GND**
- **MCP3008 CLK** to **Raspberry Pi SCLK**
- **MCP3008 DOUT** to **Raspberry Pi MISO**
- **MCP3008 DIN** to **Raspberry Pi MOSI**
- **MCP3008 CS/SHDN** to **Raspberry Pi CE0**

Library Install

After you've wired the MCP3008 to the Raspberry Pi with either the software or hardware SPI wiring you're ready to install the Adafruit MCP3008 Python library (https://adafru.it/lfl).

You can install the library from the Python package index with a few commands, or you can install the library from its source on GitHub. Pick one of these options below. If you aren't sure I recommend installing from source on GitHub (https://adafru.it/lfl) because it will also download examples to use the library.

Note that before you install the library your Raspberry Pi **must** be connected to the internet through a wired or wireless network connection.

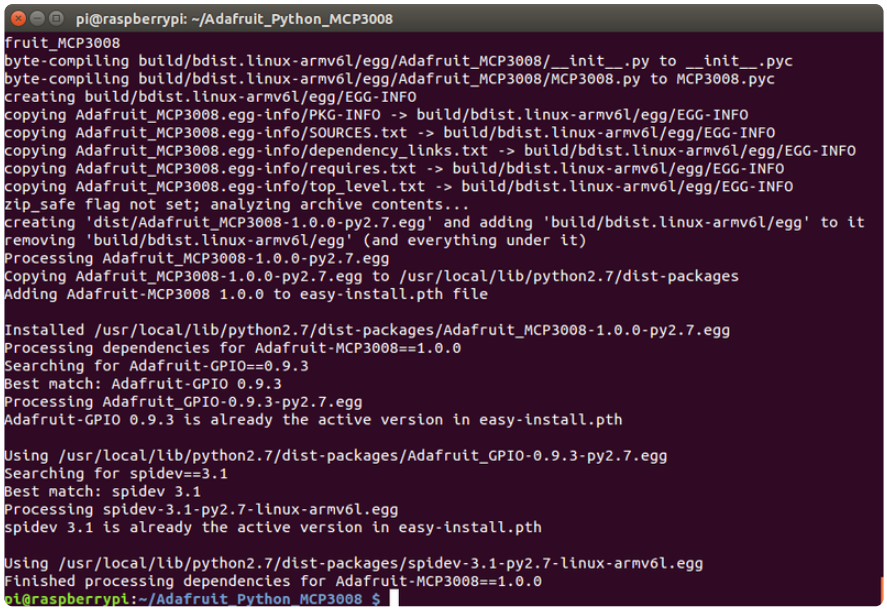
Source Install

To install from the source on Github connect to a terminal on the Raspberry Pi and run the following commands:

```
sudo apt-get update
sudo apt-get install build-essential python-dev python-smbus git
```

```
cd ~
git clone https://github.com/adafruit/Adafruit_Python_MCP3008.git
cd Adafruit_Python_MCP3008
sudo python setup.py install
```

You should see the library install succeed and finish with a message similar to the following:



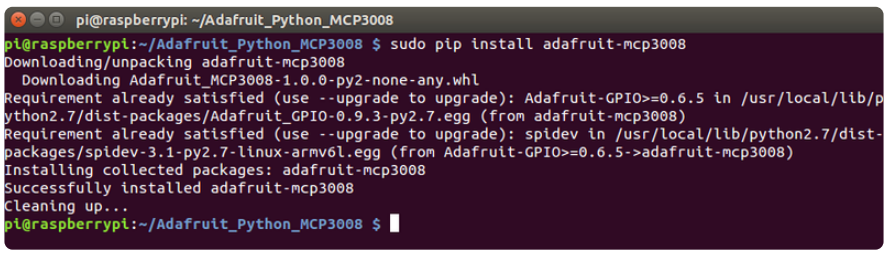
If you see an error go back and carefully check all the previous commands were run, and that they didn't fail with an error.

Python Package Index Install

To install from the Python package index (https://adafru.it/lfJ) connect to a terminal on the Raspberry Pi and execute the following commands:

```
sudo apt-get update
sudo apt-get install build-essential python-dev python-smbus python-pip
sudo pip install adafruit-mcp3008
```

You should see a message like the following that the library was successfully installed:



Note that if you install from the Python package index you **won't** have the example code for the library. You'll need to download these MCP3008 examples (https://adafru.it/lfK) to the Pi manually and run them in the next section.

Library Usage

To learn how to use the library I'll walk through some of the example code included with it. These examples are in the examples folder if you downloaded and installed the library from source. Change to that folder by running on the Pi:

```
cd ~/Adafruit_Python_MCP3008/examples
```

Note: If you installed the library from the Python package index using the pip command you won't have the example code (https://adafru.it/lfK) and will need to download it to the Pi manually.

We'll start by looking at the simpletest.py (https://adafru.it/lfL) example which is a basic example of reading and displaying the ADC channel values. First let's open the file to configure it to use software or hardware SPI. Run the following command to open the file in the nano text editor:

```
nano simpletest.py
```

Now scroll down to the following block of code near the top:

```
# Software SPI configuration:
CLK  = 18
MISO = 23
MOSI = 24
CS   = 25
mcp = Adafruit_MCP3008.MCP3008(clk=CLK, cs=CS, miso=MISO, mosi=MOSI)

# Hardware SPI configuration:
# SPI_PORT   = 0
# SPI_DEVICE = 0
# mcp = Adafruit_MCP3008.MCP3008(spi=SPI.SpiDev(SPI_PORT, SPI_DEVICE))
```

By default this section of code configures the chip to use the software SPI configuration described in the previous section. If you used different pins for your software SPI setup be sure to change the values of CLK, MISO, MOSI, CS to the pins you used.

If you used hardware SPI then you'll need to comment out the software SPI section and uncomment the hardware SPI section. The configuration should look like this for hardware SPI:

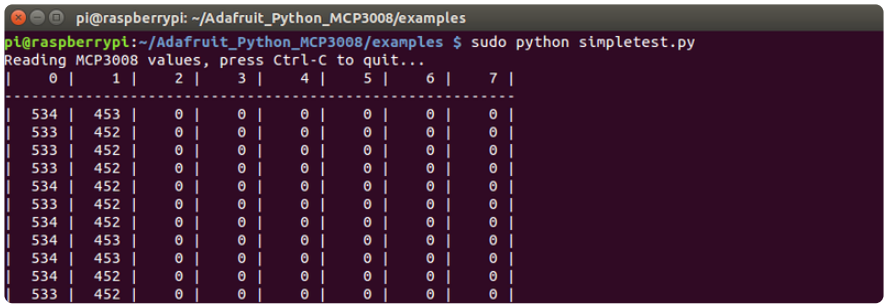
```
# Software SPI configuration:
# CLK  = 18
# MISO = 23
# MOSI = 24
# CS   = 25
# mcp = Adafruit_MCP3008.MCP3008(clk=CLK, cs=CS, miso=MISO, mosi=MOSI)

# Hardware SPI configuration:
SPI_PORT   = 0
SPI_DEVICE = 0
mcp = Adafruit_MCP3008.MCP3008(spi=SPI.SpiDev(SPI_PORT, SPI_DEVICE))
```

Now save the file by pressing **Ctrl-o**, **enter**, then **Ctrl-x** to quit. You can run the **simpletest.py** code by executing at the terminal:

```
sudo python simpletest.py
```

The example will print out a table with all of the ADC channels and their values. Every half second a new row will be printed with the latest channel values. For example you might see output like:



Each column represents a different channel and the header on the first row shows the channel number (from 0 to 7, 8 channels total). The value for each channel is the ADC value for that channel. This is a number that ranges from 0 to 1023, where 0 means the signal is at a ground level, and 1023 means it's at the AREF value (3.3V) or higher. In between values are proportional to each other, so a value of 512 is about 3.3 / 2 or 1.65 volts.

Press **Ctrl-c** to stop the example.

Try connecting a potentiometer to one of the analog inputs. Connect the middle leg of the potentiometer (the wiper) to an analog input, then connect one of the other legs to Pi 3.3V and the other leg to Pi ground. Run the example and twist the potentiometer around. You should see the ADC value change and get lower as the voltage from the potentiometer decreases, and get higher as the voltage increases!

To understand how the code works open the **simpletest.py** example in nano again. Now scroll down to the main loop at the bottom:

```
print('Reading MCP3008 values, press Ctrl-C to quit...')
# Print nice channel column headers.
print('| {0:&gt;4} | {1:&gt;4} | {2:&gt;4} | {3:&gt;4} | {4:&gt;4} | {5:&gt;4} |
{6:&gt;4} | {7:&gt;4} |'.format(*range(8)))
print('-' * 57)
# Main program loop.
while True:
    # Read all the ADC channel values in a list.
    values = [0]*8
    for i in range(8):
        # The read_adc function will get the value of the specified channel (0-7).
        values[i] = mcp.read_adc(i)
    # Print the ADC values.
    print('| {0:&gt;4} | {1:&gt;4} | {2:&gt;4} | {3:&gt;4} | {4:&gt;4} | {5:&gt;4}
| {6:&gt;4} | {7:&gt;4} |'.format(*values))
    # Pause for half a second.
    time.sleep(0.5)
```

The code might look a little complicated but most of that complication is from printing the table. Notice this line that reads an ADC channel value and saves it in a list:

values[i] = mcp.read_adc(i)

This line is calling the **read_adc()** function from the MCP3008 Python library. The function takes one parameter, the channel number to read (a value of 0 to 7). As a result the function will return the current ADC value of that channel.

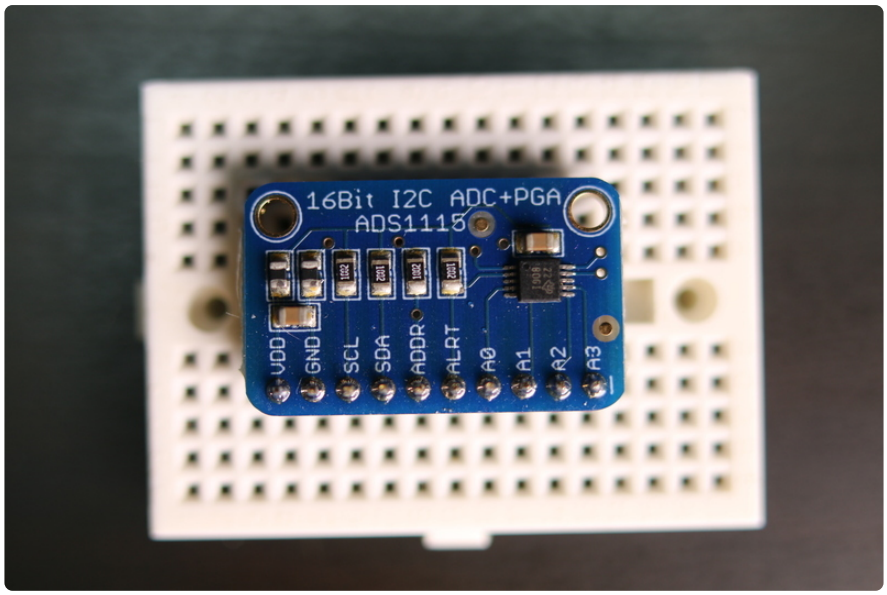
Reading an ADC channel in your own code is as easy as calling the **read_adc()** function! Pass in the channel to read and it will return the value. That's all there really is to using the MCP3008 library to read an analog value!

If you're curious you can examine and run the differential.py (https://adafru.it/lfM) example just like you ran simpletest.py. Modify the configuration to suite your wiring, either software or hardware SPI. Then when you run the example it will call the **read_adc_difference()** function and use it to read the voltage difference between channel 0 and 1 of the chip. Sometimes it's useful to read the difference of two signals to help reduce noise and other artifacts from analog signals.

That's all there is to the MCP3008 Python library!

ADS1015 / ADS1115

The examples in this guide are no longer supported. Check out the ADS1x15 guide for CircuitPython and Python usage: https://learn.adafruit.com/adafruit-4-channel-adc-breakouts/python-circuitpython



The ADS1015 and ADS1115 are great analog to digital converters that are easy to use with the Raspberry Pi using its I2C communication bus. The ADS1015 is a 12-bit ADC with 4 channels, and the ADS1115 is a higher precision 16-bit ADC with 4 channels. Both have a programmable gain from 2/3x to 16x so you can amplify small signals and read them with higher precision. If you're looking for a nice step up from the MCP3008 or other simple ADCs, the ADS1x15 series is a great option!

Before you get started be sure to follow the ADS1x15 guide (https://adafru.it/lfD) to assemble your ADC board by soldering on headers.

You might also be interested in the datasheets for these chips:

- ADS1015 Datasheet (https://adafru.it/lfN)
- ADS1115 Datasheet (https://adafru.it/lfO)

Wiring

Both the ADS1015 and ADS1115 use the same I2C communication protocol to read analog values. You can wire each chip to the Pi in exactly the same way as described below.

Before you wire the ADC to the Pi make sure to enable I2C on the Raspberry Pi using raspi-config (https://adafru.it/dEO). Don't move forward until I2C is enabled and you've checked the ADC is visible with the i2cdetect command.

Connect the ADC to the Pi as follows:

- **ADS1x15 VDD** to **Raspberry Pi 3.3V**
- **ADS1x15 GND** to **Raspberry Pi GND**
- **ADS1x15 SCL** to **Raspberry Pi SCL**
- **ADS1x15 SDA** to **Raspberry Pi SDA**

Library Install

After you've wired the ADS1x15 to the Raspberry Pi you're ready to install the Adafruit ADS1x15 Python library (https://adafru.it/lfP).

You can install the library from the Python package index with a few commands, or you can install the library from its source on GitHub. Pick one of these options below. If you aren't sure I recommend installing from source on GitHub (https://adafru.it/lfP) because it will also download examples to use the library.

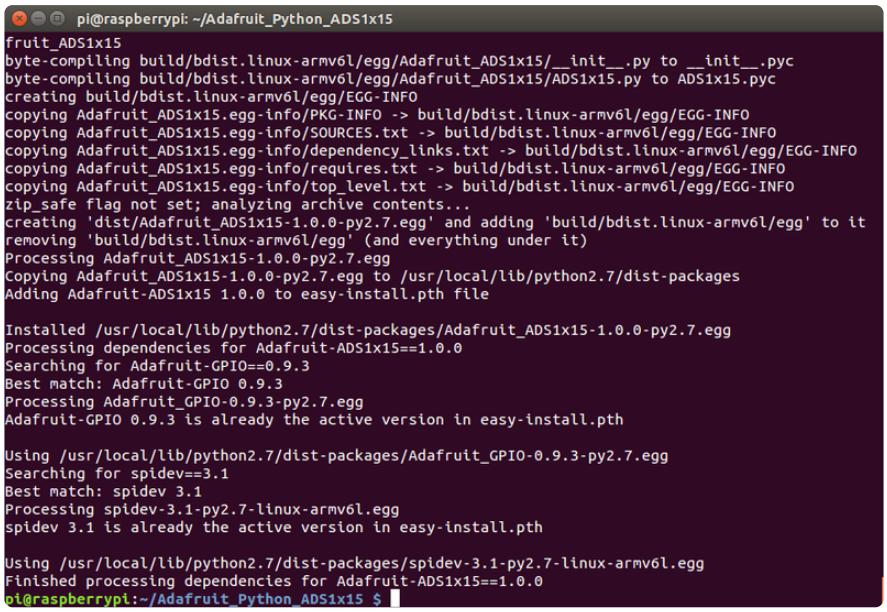
Note that before you install the library your Raspberry Pi **must** be connected to the internet through a wired or wireless network connection.

Source Install

To install from the source on Github connect to a terminal on the Raspberry Pi and run the following commands:

```
sudo apt-get update
sudo apt-get install build-essential python-dev python-smbus git
cd ~
git clone https://github.com/adafruit/Adafruit_Python_ADS1x15.git
cd Adafruit_Python_ADS1x15
sudo python setup.py install
```

You should see the library install succeed and finish with a message similar to the following:



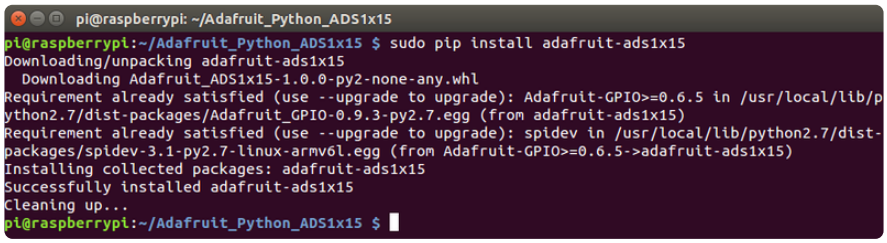
If you see an error go back and carefully check all the previous commands were run, and that they didn't fail with an error.

Python Package Index Install

To install from the Python package index (https://adafru.it/lfJ) connect to a terminal on the Raspberry Pi and execute the following commands:

```
sudo apt-get update
sudo apt-get install build-essential python-dev python-smbus python-pip
sudo pip install adafruit-ads1x15
```

You should see a message like the following that the library was successfully installed:



Note that if you install from the Python package index you **won't** have the example code for the library. You'll need to download these ADS1x15 examples (https://adafru.it/lfQ) to the Pi manually and run them in the next section.

Library Usage

To learn how to use the library I'll walk through some of the example code included with it. These examples are in the **examples** folder if you downloaded and installed the library from source. Change to that folder by running on the Pi:

```
cd ~/Adafruit_Python_ADS1x15/examples
```

Note: If you installed the library from the Python package index using the pip command you won't have the example code (https://adafru.it/lfQ) and will need to download it to the Pi manually.

We'll start by looking at the simpletest.py (https://adafru.it/lfR) example which is a basic example of reading and displaying the ADC channel values. First let's open the file to configure which chip we're using. Run the following command to open the file in the nano text editor:

```
nano simpletest.py
```

Now scroll down to the following block of code near the top:

```
# Create an ADS1115 ADC (16-bit) instance.
adc = Adafruit_ADS1x15.ADS1115()

# Or create an ADS1015 ADC (12-bit) instance.
#adc = Adafruit_ADS1x15.ADS1015()

# Note you can change the I2C address from its default (0x48), and/or the I2C
# bus by passing in these optional parameters:
#adc = Adafruit_ADS1x15.ADS1015(address=0x49, bus=1)
```

This code configures the example to use either an ADS1115 chip, or the ADS1015 chip. The top uncommented line is choosing to use the ADS1115 chip by creating an ADS1115 object. Below it is a commented line that instead creates an ADS1015 object to use the ADS1015 chip. Uncomment the right line depending on the chip you're using.

For example if you're using the ADS1015 the code would look like:

```
# Create an ADS1115 ADC (16-bit) instance.
# adc = Adafruit_ADS1x15.ADS1115()

# Or create an ADS1015 ADC (12-bit) instance.
adc = Adafruit_ADS1x15.ADS1015()

# Note you can change the I2C address from its default (0x48), and/or the I2C
```

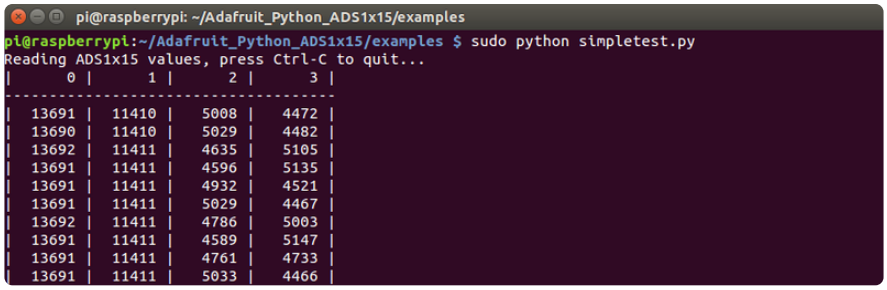
```
# bus by passing in these optional parameters:
#adc = Adafruit_ADS1x15.ADS1015(address=0x49, bus=1)
```

The last commented line shows more advanced usage like choosing a custom I2C address or bus number. You don't normally need to change these values.

Now save the file by pressing **Ctrl-o**, **enter**, then **Ctrl-x** to quit. You can run the **simpletest.py** code by executing at the terminal:

```
sudo python simpletest.py
```

The example will print out a table with all of the ADC channels and their values. Every half second a new row will be printed with the latest channel values. For example you might see output like:



Each column represents a different channel and the header on the first row shows the channel number (from 0 to 3, 4 channels total). The value for each channel is the ADC value for that channel. This is a number that ranges from -32768 to 32767 on the 16-bit ADS1115 or -2048 to 2047 on the 12-bit ADS1015. A value of 0 means the signal is at a ground (reference) level, 32767 (or 2047 on the ADS105) means it's at or higher than the maximum voltage value for the current gain (4.096V by default), and -32768 (or -2048 on the ADS1015) means it's a negative voltage below the reference voltage (e.g. if you are in differential mode). In between values are proportional to each other, so a value of 16384 is about 4.096 / 2 or 2.048 volts.

Press **Ctrl-c** to stop the example.

Try connecting a potentiometer to one of the analog inputs. Connect the middle leg of the potentiometer (the wiper) to an analog input, then connect one of the other legs to Pi 3.3V and the other leg to Pi ground. Run the example and twist the potentiometer around. You should see the ADC value change and get lower as the voltage from the potentiometer decreases, and get higher as the voltage increases!

To understand how the code works open the **simpletest.py** example in nano again. Scroll down to this section of code:

```
# Choose a gain of 1 for reading voltages from 0 to 4.09V.
# Or pick a different gain to change the range of voltages that are read:
#  - 2/3 = +/-6.144V
#  -   1 = +/-4.096V
#  -   2 = +/-2.048V
#  -   4 = +/-1.024V
#  -   8 = +/-0.512V
#  -  16 = +/-0.256V
# See table 3 in the ADS1015/ADS1115 datasheet for more info on gain.
GAIN = 1
```

The code configures the gain that the ADC will use when reading signals. Gain will amplify signals so it's easier to read small weak signals. The gain also controls the range of voltages that can be read by the chip. With a gain value of 1 notice the range of voltage is 4.096 volts. This means the chip can read values from -4.096 volts to +4.096 volts. You can choose higher gains to read weaker signals with higher precision (since more bits will be used to represent a smaller range of values).

Choosing the right gain for your signal is very important and worth thinking about the possible range of voltages you might see. Too high a gain will push your signal beyond the ADC's max voltage and produce inaccurate results, and too low a gain will bury your signal in noise making it difficult to read. Choose carefully!

Now scroll down to the main loop at the bottom:

```
print('Reading ADS1x15 values, press Ctrl-C to quit...')
# Print nice channel column headers.
print('| {0:&gt;6} | {1:&gt;6} | {2:&gt;6} | {3:&gt;6} |'.format(*range(4)))
print('-' * 37)
# Main loop.
while True:
    # Read all the ADC channel values in a list.
    values = [0]*4
    for i in range(4):
        # Read the specified ADC channel using the previously set gain value.
        values[i] = adc.read_adc(i, gain=GAIN)
        # Note you can also pass in an optional data_rate parameter that controls
        # the ADC conversion time (in samples/second). Each chip has a different
        # set of allowed data rate values, see datasheet Table 9 config register
        # DR bit values.
        #values[i] = adc.read_adc(i, gain=GAIN, data_rate=128)
        # Each value will be a 12 or 16 bit signed integer value depending on the
        # ADC (ADS1015 = 12-bit, ADS1115 = 16-bit).
    # Print the ADC values.
    print('| {0:&gt;6} | {1:&gt;6} | {2:&gt;6} | {3:&gt;6} |'.format(*values))
    # Pause for half a second.
    time.sleep(0.5)
```

The code might look a little complicated but most of that complication is from printing the table. Notice this line that reads an ADC channel value and saves it in a list:

values[i] = adc.read_adc(i, gain=GAIN)

This line is calling the **read_adc()** function from the ADS1x15 Python library. The function takes one parameter, the channel number to read (a value of 0 to 3), and optionally a gain value (the default is 1). As a result the function will return the current ADC value of that channel.

Reading an ADC channel in your own code is as easy as calling the **read_adc()** function! Pass in the channel to read and it will return the value. That's all there really is to using the ADS1x15 library to read an analog value!

If you're curious you can examine and run the differential.py (https://adafru.it/lfS) example just like you ran simpletest.py. Modify the configuration to choose your chip. Then when you run the example it will call the **read_adc_difference()** function and use it to read the voltage difference between channel 0 and 1 of the chip. Sometimes it's useful to read the difference of two signals to help reduce noise and other artifacts from analog signals.

In addition the continuous.py (https://adafru.it/lfT) example demonstrates how to turn on continuous read mode and read a stream of ADC values. This is useful if you're just reading values from the chip and don't want to worry about calling the **read_adc()** function all the time. See the comments in the example for an explanation of how it works.

Finally the comparator.py (https://adafru.it/lfU) example shows a more advanced mode of the chips where they can enable an ALERT output pin when the ADC value falls within a specific range. Again the comments and datasheet should help you use this functionality.

That's all there is to the ADS1x15 Python library!