**INTERFACING ADS1115 WITH RASPBERRY PI**

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!

**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://learn.adafruit.com/adafruits-raspberry-pi-lesson-4-gpio-setup/configuring-i2c).  Don't move forward until I2C is enabled and you've checked the ADC is visible with the i2cdetect –y 1 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 (Pin 5)
* ADS1x15 SDA to Raspberry Pi SDA (Pin 3)

**Library Install**

After you've wired the ADS1x15 to the Raspberry Pi  you're ready to install the [Adafruit ADS1x15 Python library](https://github.com/adafruit/Adafruit_Python_ADS1x15).

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://github.com/adafruit/Adafruit_Python_ADS1x15) 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:

1. sudo apt-get update
2. sudo apt-get install build-essential python-dev python-smbus git
3. cd ~
4. git clone https://github.com/adafruit/Adafruit\_Python\_ADS1x15.git
5. cd Adafruit\_Python\_ADS1x15
6. 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://pypi.python.org/pypi) connect to a terminal on the Raspberry Pi and execute the following commands:

1. sudo apt-get update
2. sudo apt-get install build-essential python-dev python-smbus python-pip
3. sudo pip install adafruit-ads1x15

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

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:

1. 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://github.com/adafruit/Adafruit_Python_ADS1x15/tree/master/examples)**and will need to download it to the Pi manually.**

We'll start by looking at the [simpletest.py](https://github.com/adafruit/Adafruit_Python_ADS1x15/blob/master/examples/simpletest.py) 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:

1. nano simpletest.py

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

1. # Create an ADS1115 ADC (16-bit) instance.
2. adc = Adafruit\_ADS1x15.ADS1115()
4. # Or create an ADS1015 ADC (12-bit) instance.
5. #adc = Adafruit\_ADS1x15.ADS1015()
7. # Note you can change the I2C address from its default (0x48), and/or the I2C
8. # bus by passing in these optional parameters:
9. #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.

1. # Create an ADS1115 ADC (16-bit) instance.
2. # adc = Adafruit\_ADS1x15.ADS1115()
4. # Or create an ADS1015 ADC (12-bit) instance.
5. adc = Adafruit\_ADS1x15.ADS1015()
7. # Note you can change the I2C address from its default (0x48), and/or the I2C
8. # bus by passing in these optional parameters:
9. #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:

1. 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:

1. # Choose a gain of 1 for reading voltages from 0 to 4.09V.
2. # Or pick a different gain to change the range of voltages that are read:
3. # - 2/3 = +/-6.144V
4. # - 1 = +/-4.096V
5. # - 2 = +/-2.048V
6. # - 4 = +/-1.024V
7. # - 8 = +/-0.512V
8. # - 16 = +/-0.256V
9. # See table 3 in the ADS1015/ADS1115 datasheet for more info on gain.
10. 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:

1. print('Reading ADS1x15 values, press Ctrl-C to quit...')
2. # Print nice channel column headers.
3. print('| {0:>6} | {1:>6} | {2:>6} | {3:>6} |'.format(\*range(4)))
4. print('-' \* 37)
5. # Main loop.
6. while True:
7. # Read all the ADC channel values in a list.
8. values = [0]\*4
9. for i in range(4):
10. # Read the specified ADC channel using the previously set gain value.
11. values[i] = adc.read\_adc(i, gain=GAIN)
12. # Note you can also pass in an optional data\_rate parameter that controls
13. # the ADC conversion time (in samples/second). Each chip has a different
14. # set of allowed data rate values, see datasheet Table 9 config register
15. # DR bit values.
16. #values[i] = adc.read\_adc(i, gain=GAIN, data\_rate=128)
17. # Each value will be a 12 or 16 bit signed integer value depending on the
18. # ADC (ADS1015 = 12-bit, ADS1115 = 16-bit).
19. # Print the ADC values.
20. print('| {0:>6} | {1:>6} | {2:>6} | {3:>6} |'.format(\*values))
21. # Pause for half a second.
22. 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://github.com/adafruit/Adafruit_Python_ADS1x15/blob/master/examples/differential.py) 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://github.com/adafruit/Adafruit_Python_ADS1x15/blob/master/examples/continuous.py) 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://github.com/adafruit/Adafruit_Python_ADS1x15/blob/master/examples/comparator.py) 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.