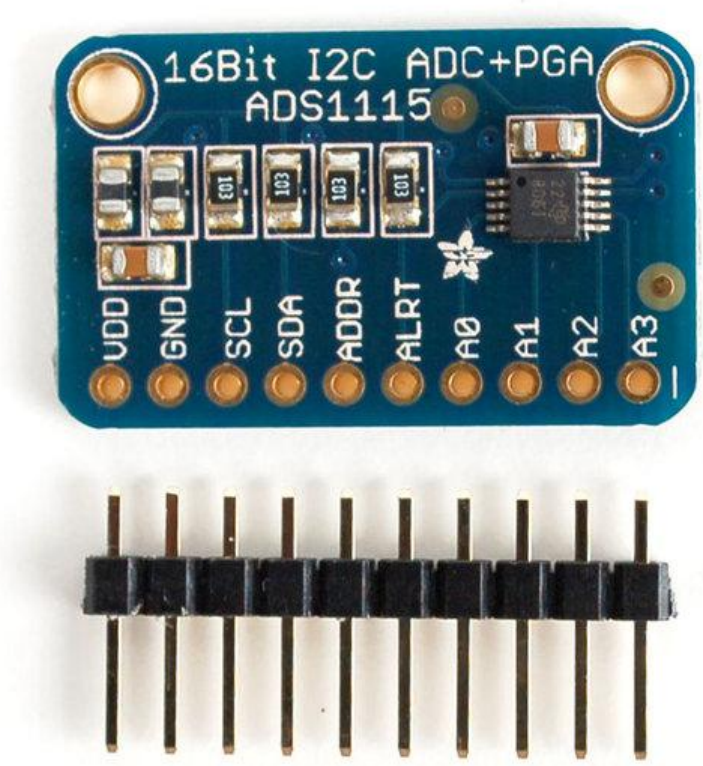
Adafruit 4-Channel ADC Breakouts

Created by Bill Earl

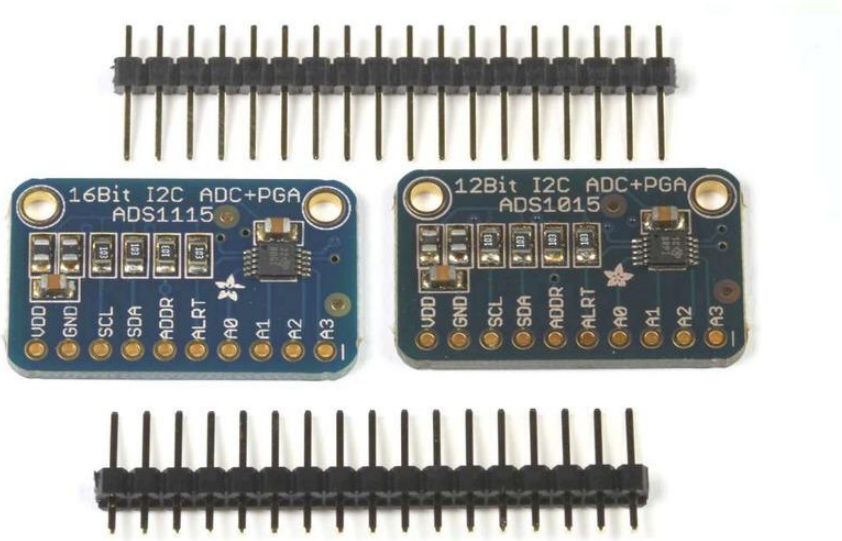


https://learn.adafruit.com/adafruit-4-channel-adc-breakouts

Last updated on 2021-11-15 05:53:04 PM EST

Table of Contents

Overview



The ADS1115 and ADS1015 4-channel breakout boards are perfect for adding high-resolution analog to digital conversion to any microprocessor-based project. These boards can run with power and logic signals between 2v to 5v, so they are compatible with all common 3.3v and 5v processors. As many of 4 of these boards can be controlled from the same 2-wire I2C bus, giving you up to 16 single-ended or 8 differential channels. A programmable gain amplifier provides up to x16 gain for small signals.

These two boards are very similar, differing only in resolution and speed. The ADS1115 has higher resolution and the ADS1015 has a higher sample rate.

ADS1115 Features:

- Resolution: 16 Bits
- Programmable Sample Rate: 8 to 860 Samples/Second
- Power Supply/Logic Levels: 2.0V to 5.5V
- Low Current Consumption: Continuous Mode: Only 150µA Single-Shot Mode: Auto Shut-Down
- Internal Low-Drift Voltage Reference
- Internal Oscillator
- Internal PGA: up to x16
- I2C Interface: 4-Pin-Selectable Addresses
- Four Single-Ended or 2 Differential Inputs
- Programmable Comparator

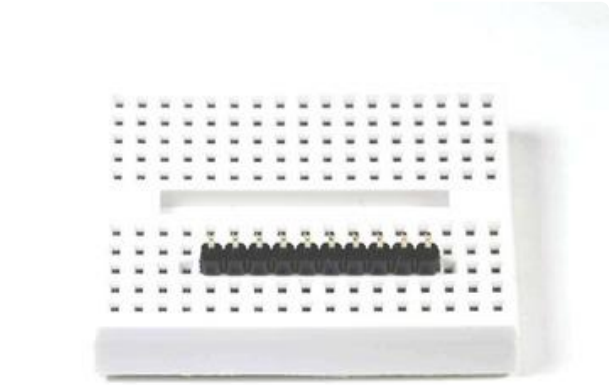
ADS1015 Features:

- Resolution: 12 Bits
- Programmable Sample Rate: 128 to 3300 Samples/Second
- Power Supply/Logic Levels: 2.0V to 5.5V
- Low Current Consumption: Continuous Mode: Only 150µA Single-Shot Mode: Auto Shut-Down
- Internal Low-Drift Voltage Reference
- Internal Oscillator
- Internal PGA: up to x16
- I2C Interface: 4-Pin-Selectable Addresses
- Four Single-Ended or 2 Differential Inputs
- Programmable Comparator

Assembly and Wiring

Assembly:

The board comes with all surface-mount parts pre-soldered. For breadboard use, the included header-strip should be soldered on:

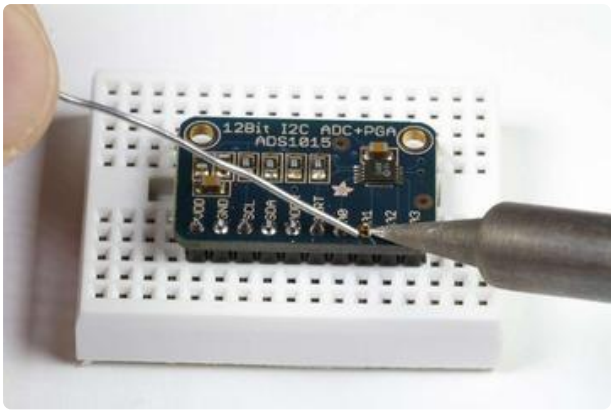


Prepare the header strip

Cut the supplied header strip to length and insert it long-pins-down in your breadboard to hold it for soldering.

Position the breakout board

Place the breakout board on the header pins.



Solder!

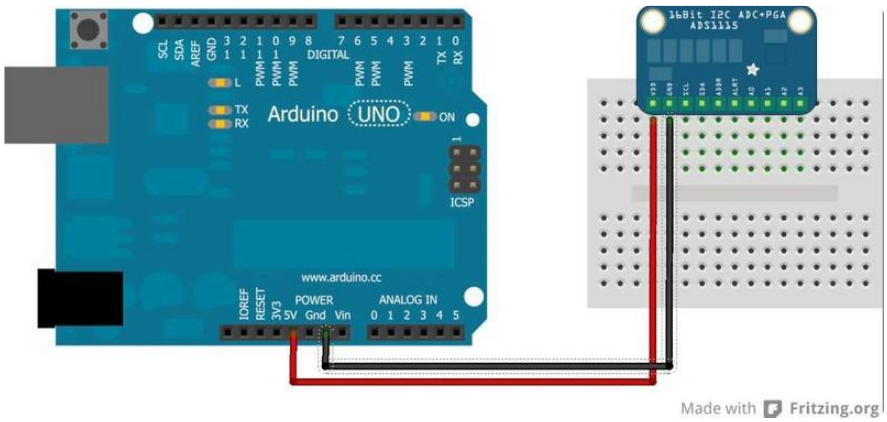
Solder each pin for a good electrical connection.

Wiring:

Power

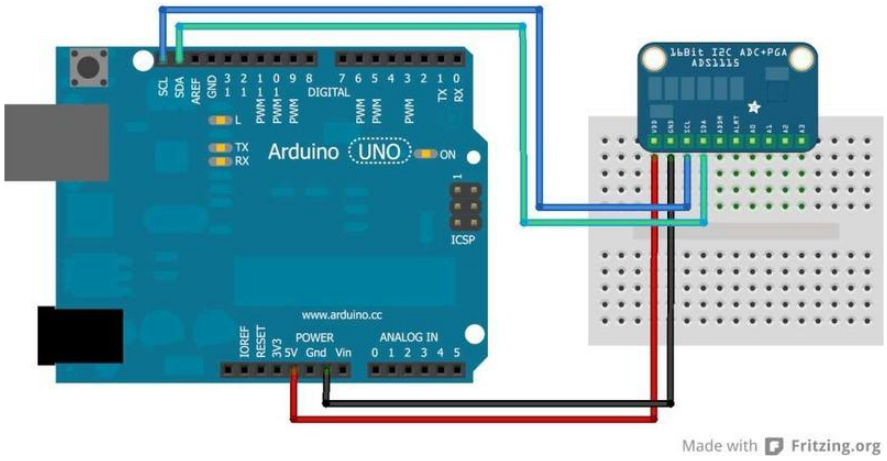
First connect VDD and GND. These boards will work with either a 3.3v or a 5v supply. The diagram below shows connection to the Arduino 5v pin.

The absolute maximum analog input voltage is VDD + 0.3v. To avoid damage to the chip, do not attempt to measure voltages greater than VDD.



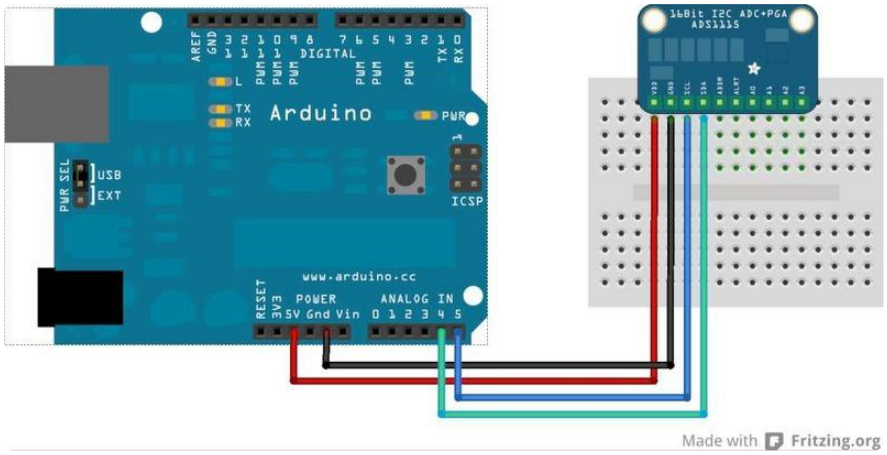
I2C Connections

I2C requires just 2 pins to communicate. These can be shared with other I2C devices. For R3 and later Arduinos (including MEGA and DUE models), connect SDA->SDA and SCL->SCL.



I2C "Classic"

For older Arduino boards without dedicated SDA and SCL pins, connect as shown below. (For older Arduino Megas, SDA and SCL are on pins 20 and 21)

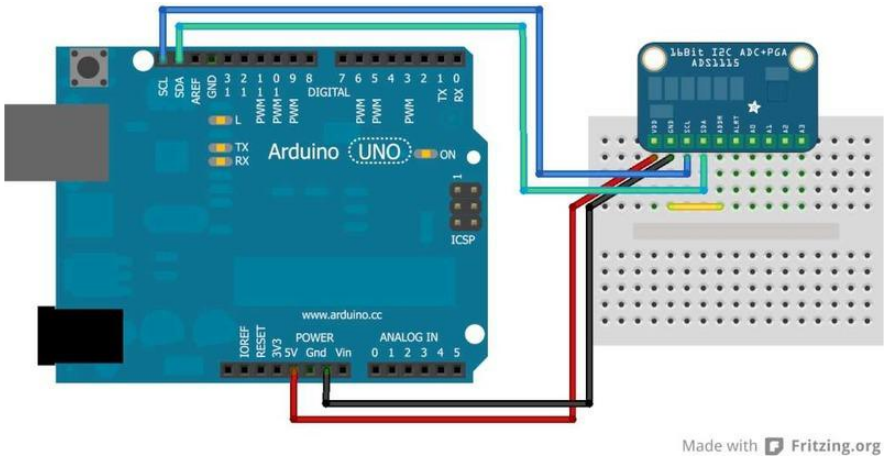


I2C Addressing

The ADS11x5 chips have a base 7-bit I2C address of 0x48 (1001000) and a clever addressing scheme that allows four different addresses using just one address pin (named ADR for ADdRess). To program the address, connect the address pin as follows:

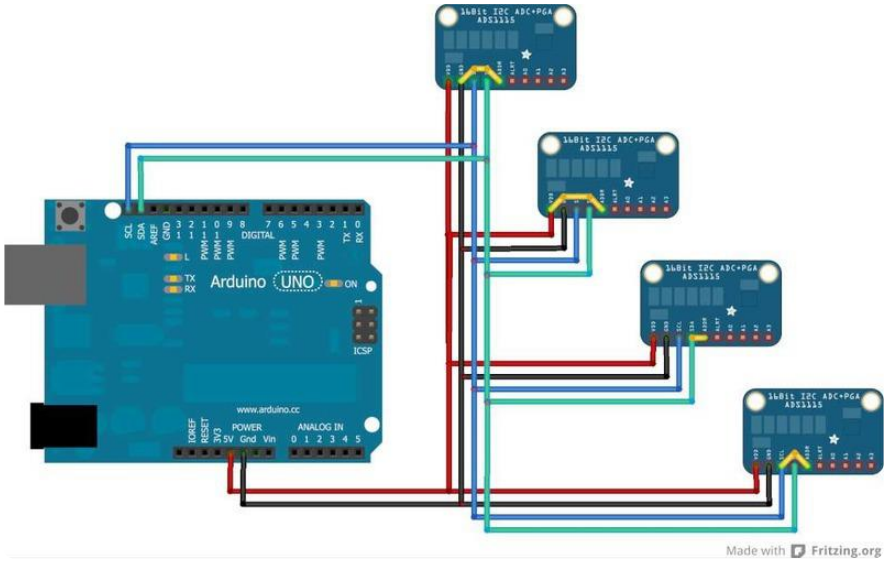
- 0x48 (1001000) ADR -> GND
- 0x49 (1001001) ADR -> VDD
- 0x4A (1001010) ADR -> SDA
- 0x4B (1001011) ADR -> SCL

The following diagram shows one board addressed as 0x48:

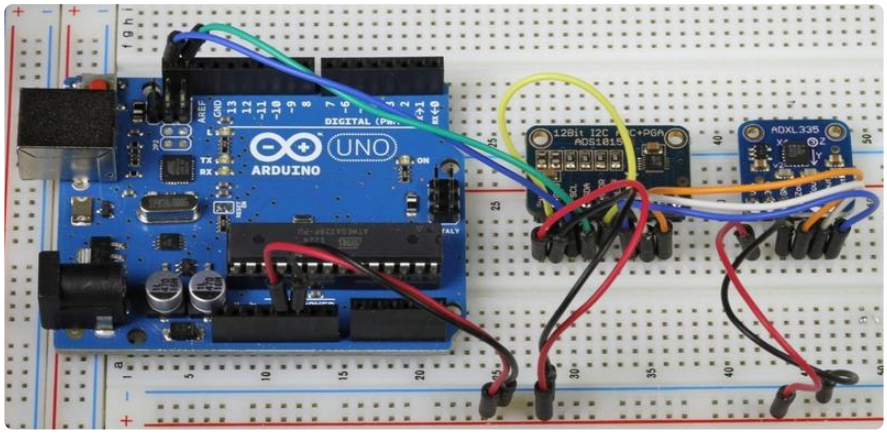


Multiple Boards

By assigning each board a different address, up to 4 boards can be connected as below:



Signal Connections



Single Ended vs. Differential Inputs:

The ADS1x15 breakouts support up to 4 SIngle Ended or 2 Differential inputs.

Single Ended inputs measure the voltage between the analog input channel (A0-A3) and analog ground (GND).

Differential inputs measure the voltage between two analog input channels. (A0&A1 or A2&A3).

Which should I use?

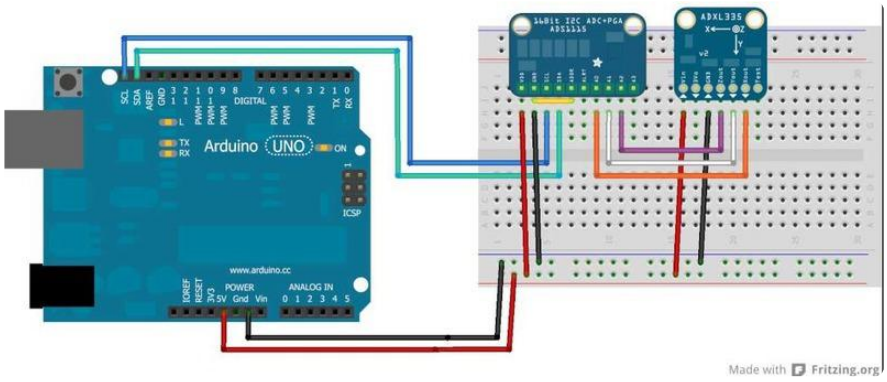
Single ended inputs give you twice as many inputs. So why would you want to use differential inputs?

Single ended inputs can, by definition, only measure positive voltages. Without the sign bit, you only get an effective 15 bit resolution.

In addition to providing the full 16 bits of resolution and the ability to measure negative voltages, Differential measurements offer more immunity from electromagnetic noise. This is useful when using long signal wires or operating in an electrically noisy environment. This is also desirable when dealing with small signals requiring high gain, since the gain will amplify the noise as well as the signal.

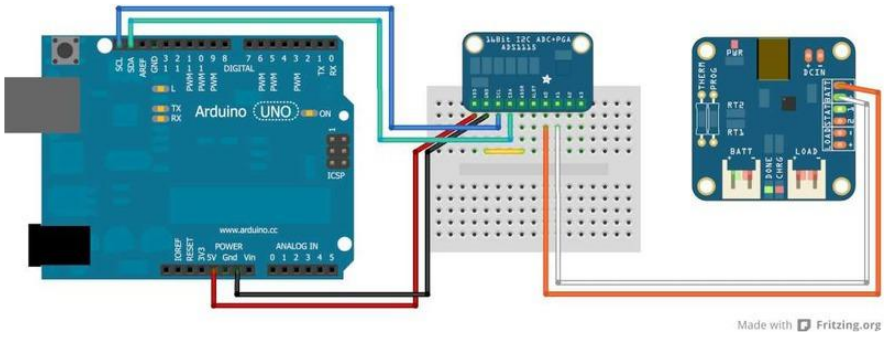
Single Ended Connections:

Connect the signal wire to one of the analog input channels (A0 - A3). Connect the ground wire to GND. This diagram shows how to connect an ADXL335 to for measurement of the X, Y and Z axis on analog channels A0, A1 and A2.



Differential Connections:

Differential measurements use a pair of input pins, either A0&A1 or A2&A3. The following diagram shows connections for differential measurement of the battery voltage on a LiPo charger board.



All input signals to these devices must be between ground potential and VCC. If your source signal produces negative voltages, they must be offset to fall within the GND to VCC range of the ASD1x15.

Arduino Code

The Adafruit_ADS1X15 library supports both single-ended and differential readings as well as comparator operations on both the ADS1015 and ADS1115 breakout boards. The library uses the wiring library for I2C communication, so wiring.h must be included.

Construction and Initialization:

Adafruit_ADS1015();
Construct an instance of an ADS1015

Adafruit_ADS1115();
Construct an instance of an ADS1115

begin();
Initialize the ADC for operation using the default address and I2C bus.

begin(0x49);
Initialize the ADC for operation using specified address of 0x49.

Example:

The following examples assume an ADS1015 and use a 3 mV/bit scaling factor. For the higher-resolution ADS1115, the scaling factor would be 188uV/bit.

```
#include &lt;Wire.h&gt;
#include &lt;Adafruit_ADS1X15.h&gt;

Adafruit_ADS1015 ads1015;    // Construct an ads1015
Adafruit_ADS1115 ads1115;    // Construct an ads1115

void setup(void)
{
  ads1015.begin();  // Initialize ads1015 at the default address 0x48
  ads1115.begin(0x49);  // Initialize ads1115 at address 0x49
}
```

Single Ended Conversion:

uint16_t readADC_SingleEnded(uint8_t channel);

Perform a single-ended analog to digital conversion on the specified channel.

Example:

```
#include &lt;Wire.h&gt;
#include &lt;Adafruit_ADS1X15.h&gt;

Adafruit_ADS1015 ads1015;

void setup(void)
{
  Serial.begin(9600);
  Serial.println("Hello!");

  Serial.println("Getting single-ended readings from AIN0..3");
  Serial.println("ADC Range: +/- 6.144V (1 bit = 3mV)");
  ads1015.begin();
}

void loop(void)
{
  int16_t adc0, adc1, adc2, adc3;

  adc0 = ads1015.readADC_SingleEnded(0);
  adc1 = ads1015.readADC_SingleEnded(1);
  adc2 = ads1015.readADC_SingleEnded(2);
  adc3 = ads1015.readADC_SingleEnded(3);
  Serial.print("AIN0: "); Serial.println(adc0);
  Serial.print("AIN1: "); Serial.println(adc1);
  Serial.print("AIN2: "); Serial.println(adc2);
  Serial.print("AIN3: "); Serial.println(adc3);
  Serial.println(" ");

  delay(1000);
}
```

Differential Conversion:

int16_t readADC_Differential_0_1(void);
Perform a differential analog to digital conversion on the voltage between channels 0 and 1.

int16_t readADC_Differential_2_3(void);
Perform a differential analog to digital conversion on the voltage between channels 2 and 3.

Example:

```
#include &lt;Wire.h&gt;
#include &lt;Adafruit_ADS1X15.h&gt;

Adafruit_ADS1015 ads1015;

void setup(void)
{
  Serial.begin(9600);
  Serial.println("Hello!");

  Serial.println("Getting differential reading from AIN0 (P) and AIN1 (N)");
  Serial.println("ADC Range: +/- 6.144V (1 bit = 3mV)");
  ads1015.begin();
}

void loop(void)
```

```
{
  int16_t results;

  results = ads1015.readADC_Differential_0_1();
  Serial.print("Differential: "); Serial.print(results); Serial.print("(");
Serial.print(results * 3); Serial.println("mV)");

  delay(1000);
}
```

Comparator Operation:

Comparator mode allows you to compare an input voltage with a threshold level and generate an alert signal (on the ALRT pin) if the threshold is exceeded. This pin can be polled with a digital input pin, or it can be configured to generate an interrupt.

void startComparator_SingleEnded(uint8_t channel, int16_t threshold);
Set the threshold and channel for comparator operation.

int16_t getLastConversionResults();
Get the last conversion result and clear the comparator.

Example:

```
#include &lt;Wire.h&gt;
#include &lt;Adafruit_ADS1X15.h&gt;

Adafruit_ADS1015 ads1015;

void setup(void)
{
  Serial.begin(9600);
  Serial.println("Hello!");

  Serial.println("Single-ended readings from AIN0 with &gt;3.0V comparator");
  Serial.println("ADC Range: +/- 6.144V (1 bit = 3mV)");
  Serial.println("Comparator Threshold: 1000 (3.000V)");
  ads1015.begin();

  // Setup 3V comparator on channel 0
  ads1015.startComparator_SingleEnded(0, 1000);
}

void loop(void)
{
  int16_t adc0;

  // Comparator will only de-assert after a read
  adc0 = ads1015.getLastConversionResults();
  Serial.print("AIN0: "); Serial.println(adc0);

  delay(100);
}
```

Adjusting Gain

To boost small signals, the gain can be adjusted on the ADS1x15 chips in the following steps:

- GAIN_TWOTHIRDS (for an input range of +/- 6.144V)
- GAIN_ONE (for an input range of +/-4.096V)
- GAIN_TWO (for an input range of +/-2.048V)
- GAIN_FOUR (for an input range of +/-1.024V)
- GAIN_EIGHT (for an input range of +/-0.512V)
- GAIN_SIXTEEN (for an input range of +/-0.256V)

adsGain_t getGain(void)

Reads the current gain value (default = 2/3x)

```
adsGain_t gain = getGain();
```

void setGain(adsGain_t gain)

Sets the gain for the ADS1x15

```
ads1015.setGain(GAIN_TWOTHIRDS);  // 2/3x gain +/- 6.144V  1 bit = 3mV (default)

// ads1015.setGain(GAIN_ONE);     // 1x gain   +/- 4.096V  1 bit = 2mV
// ads1015.setGain(GAIN_TWO);     // 2x gain   +/- 2.048V  1 bit = 1mV
// ads1015.setGain(GAIN_FOUR);    // 4x gain   +/- 1.024V  1 bit = 0.5mV
// ads1015.setGain(GAIN_EIGHT);   // 8x gain   +/- 0.512V  1 bit = 0.25mV
// ads1015.setGain(GAIN_SIXTEEN); // 16x gain  +/- 0.256V  1 bit = 0.125mV
```

Example

If we had an analog sensor with an output voltage ~1V (a TMP36, for example), we could set the gain on the ADC to GAIN_FOUR, which would give us a +/-1.024V range. This would push the 1V input signal over the entire 12-bit or 16-bit range of the ADC, compared to the very limited range 1V would cover without adjusting the gain settings

```
// Set the gain to 4x, for an input range of +/- 1.024V
// 1-bit = 0.5V on the ADS1015 with this gain setting
ads1015.setGain(GAIN_FOUR);
```

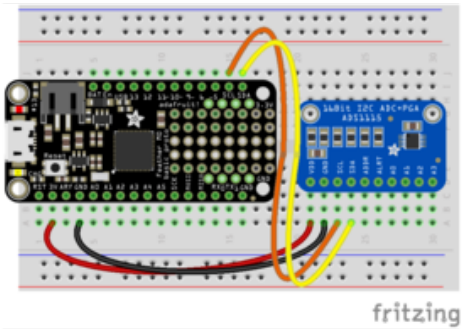
Python & CircuitPython

It's easy to use the ADS1115 and ADS1015 ADC with CircuitPython and the Adafruit CircuitPython ADS1x15 (https://adafru.it/C1n) module. This module allows you to easily write Python code that reads the analog input values.

You can use this ADC with any CircuitPython microcontroller board or with a computer that has GPIO and Python thanks to Adafruit_Blinka, our CircuitPython-for-Python compatibility library (https://adafru.it/BSN).

CircuitPython Microcontroller Wiring

First wire up the ADC to your board exactly as shown on the previous pages for Arduino using an I2C interface. Here's an example of wiring a Feather M0 to the ADS1115 with I2C:

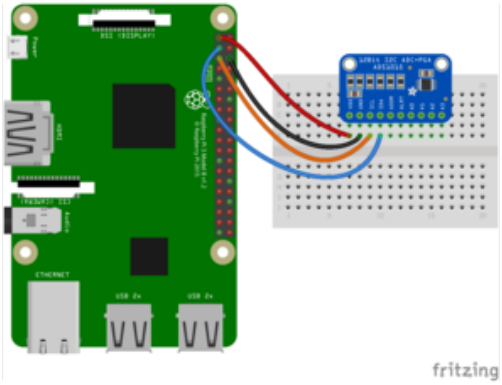


- Board 3V to ADS1115 VDD - Remember the maximum input voltage to any ADC channel cannot exceed this VDD 3V value!
- Board GND to ADS1115 GND
- Board SCL to ADS1115 SCL
- Board SDA to ADS1115 SDA

Python Computer Wiring

Since there's dozens of Linux computers/boards you can use we will show wiring for Raspberry Pi. For other platforms, please visit the guide for CircuitPython on Linux to see whether your platform is supported (https://adafru.it/BSN).

Here's the Raspberry Pi wired to the ADS1015 with I2C:



- Pi 3V to ADS1015 VDD - Remember the maximum input voltage to any ADC channel cannot exceed this VDD 3V value!
- Pi GND to ADS1015 GND
- Pi SCL to ADS1015 SCL
- Pi SDA to ADS1015 SDA

CircuitPython Installation of ADS1x15Library

Next you'll need to install the Adafruit CircuitPython ADS1x15 (https://adafru.it/C1n) library on your CircuitPython board.

First make sure you are running the latest version of Adafruit CircuitPython (https://adafru.it/tBa) for your board.

Next you'll need to install the necessary libraries to use the hardware--carefully follow the steps to find and install these libraries from Adafruit's CircuitPython library bundle (https://adafru.it/zdx). For example the Circuit Playground Express guide has a great page on how to install the library bundle (https://adafru.it/Bf2) for both express and non-express boards.

Remember for non-express boards like the Trinket M0, Gemma M0, and Feather/Metro M0 basic you'll need to manually install the necessary libraries from the bundle:

- adafruit_ads1x15
- adafruit_bus_device

You can also download the adafruit_ads1x15 folder from its releases page on Github (https://adafru.it/C1o).

Before continuing make sure your board's lib folder or root filesystem has the adafruit_ads1x15 and adafruit_bus_device files and folders copied over.

Next connect to the board's serial REPL (https://adafru.it/pMf)so you are at the CircuitPython >>> prompt.

Python Installation of ADS1x15 Library

You'll need to install the Adafruit_Blinka (https://adafru.it/BJX) library that provides the CircuitPython support in Python. This may also require enabling I2C on your platform and verifying you are running Python 3. Since each platform is a little different, and Linux changes often, please visit the CircuitPython on Linux guide to get your computer ready (https://adafru.it/BSN)!

Once that's done, from your command line run the following command:

- `sudo pip3 install adafruit-circuitpython-ads1x15`

If your default Python is version 3 you may need to run 'pip' instead. Just make sure you aren't trying to use CircuitPython on Python 2.x, it isn't supported!

CircuitPython & Python Usage

To demonstrate the usage of the ADC we will initialize it and read the ADC channel values interactively using the REPL. First run the following code to import the necessary modules and initialize the I2C bus:

```
import board
import busio
i2c = busio.I2C(board.SCL, board.SDA)
```

Next, import the module for the board you are using. For the ADS1015, use:

```
import adafruit_ads1x15.ads1015 as ADS
```

OR, for the ADS1115, use:

```
import adafruit_ads1x15.ads1115 as ADS
```

Note that we are renaming each import to ADS for convenience.

The final import needed is for the ADS1x15 library's version of AnalogIn:

```
from adafruit_ads1x15.analog_in import AnalogIn
```

which provides behavior similar to the core AnalogIn library (https://adafru.it/Bep), but is specific to the ADS1x15 ADC's.

OK, now we can actually create the ADC object. For the ADS1015, use:

```
ads = ADS.ADS1015(i2c)
```

OR, for the ADS1115, use:

```
ads = ADS.ADS1115(i2c)
```

Now let's see how to get values from the board. You can use these boards in either single ended or differential mode. The usage for the two modes are slightly different, so we'll go over them separately.

Single Ended Mode

For single ended mode we use AnalogIn to create the analog input channel, providing the ADC object and the pin to which the signal is attached. Here, we use pin 0:

```
chan = AnalogIn(ads, ADS.P0)
```

To set up additional channels, use the same syntax but provide a different pin.

Now you can read the raw value and voltage of the channel using either the the value or voltage property.

```
print(chan.value, chan.voltage)
```

```
Adafruit CircuitPython 3.1.1 on 2018-11-02; Adafruit ItsyBitsy M4 Express with s
amd51g19
>>> import board
>>> import busio
>>> i2c = busio.I2C(board.SCL, board.SDA)
>>> import adafruit_ads1x15.ads1015 as ADS
>>> from adafruit_ads1x15.analog_in import AnalogIn
>>> ads = ADS.ADS1015(i2c)
>>> chan = AnalogIn(ads, ADS.P0)
>>> print(chan.value, chan.voltage)
808 1.61679
>>>
```

Differential Mode

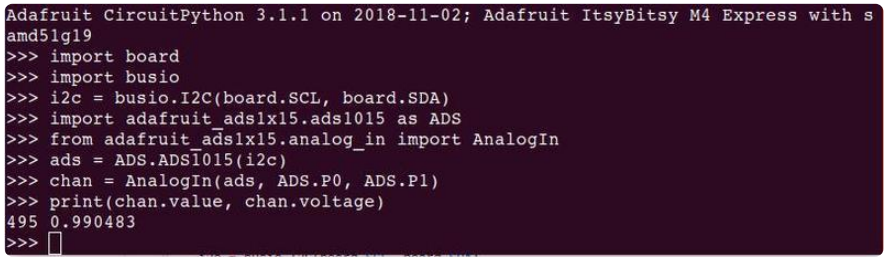
For differential mode, you provide two pins when setting up the ADC channel. The reading will be the difference between the two. Here, we use pin 0 and 1:

```
chan = AnalogIn(ads, ADS.P0, ADS.P1)
```

You can create more channels by doing this again with different pins. However, note that not all pin combinations are possible. See the datasheets for details.

Once the channel is created, getting the readings is the same as before:

```
print(chan.value, chan.voltage)
```



Gain

Both the ADS1015 and the ADS1115 have a Programmable Gain (PGA) that you can set to amplify the incoming signal before it reaches the ADC. The available settings and associated Full Scale (FS) voltage range are shown in Table 3 of the datasheet.

Table 3. PGA Gain Full-Scale Range

PGA SETTING	FS (V)
2/3	±6.144V(1)
1	±4.096V(1)
2	±2.048V
4	±1.024V
8	±0.512V
16	±0.256V

You set the gain to one of the values using the `gain` property, like this:

```
ads.gain = 16
```

Note that setting `gain` will affect the raw ADC `value` but not the `voltage` (expect for variance due to noise). For example:

```
&gt;&gt;&gt; ads.gain
1
&gt;&gt;&gt; chan.value, chan.voltage
(84, 0.168082)
```

```
&gt;&gt;&gt; ads.gain = 16
&gt;&gt;&gt; ads.gain
16
&gt;&gt;&gt; chan.value, chan.voltage
(1335, 0.167081)
&gt;&gt;&gt;
```

The `value` changed from 84 to 1335, which is pretty close to 84 x 16 = 1344. However, the `voltage` returned in both cases is still the actual input voltage of ~0.168 V.

Single Mode

Single mode (not to be confused with single-ended mode) always waits until the analog to digital conversion is completed by the ADC to read the value. This is slower, but it is really the only way to read from more than one pin on the ADC. This is the default mode, but if you ever need to manually set it, here's how:

```
ads.mode = Mode.SINGLE
```

Continuous Mode

In continuous mode, the board will read the latest value that the ADS1x15 device has converted. This is faster as it does not have to wait for the ADC to finish converting the value, but it only really works when reading data from one of the four pins on the ADC. You can set it to continuous mode using the line below:

```
ads.mode = Mode.CONTINUOUS
```

Challenges to Reading Quickly

The best approach to reading data from the ADC quickly would be to use interrupts which would enable the digital value to be recorded from the desired pin as soon as the ADC has converted it. However, CircuitPython does not support the use of interrupts, so this isn't possible. To partially solve this, the continuous mode was added, which will always return the most recently converted value with very low latency since it does not wait for the new value to be ready.

More Info

The above examples cover the basic setup and usage using default settings. For more details, see the documentation (https://adafru.it/C1p).

Python Docs

Python Docs (https://adafru.it/C1p)

Downloads

Software

- ADS1x15 Library for Arduino (https://adafru.it/aSt)

Files

- Board Files and Schematics (https://adafru.it/aSu)
- Fritzing library (https://adafru.it/aP3)
- ADS1015 Data Sheet (https://adafru.it/aSv)
- ADS1115 Data Sheet (https://adafru.it/aSw)

Schematic (Identical For Both)