adafruit learning system

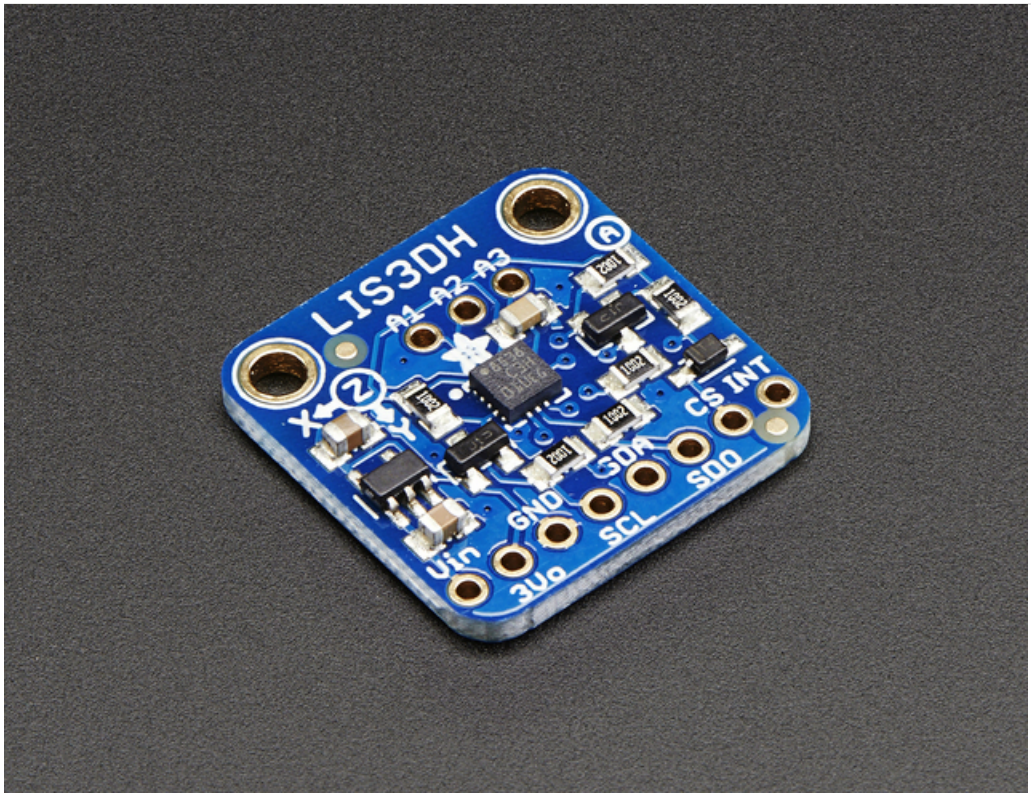
Adafruit LIS3DH Triple-Axis Accelerometer Breakout

Created by lady ada

Last updated on 2017-11-14 02:21:20 AM UTC

Guide Contents

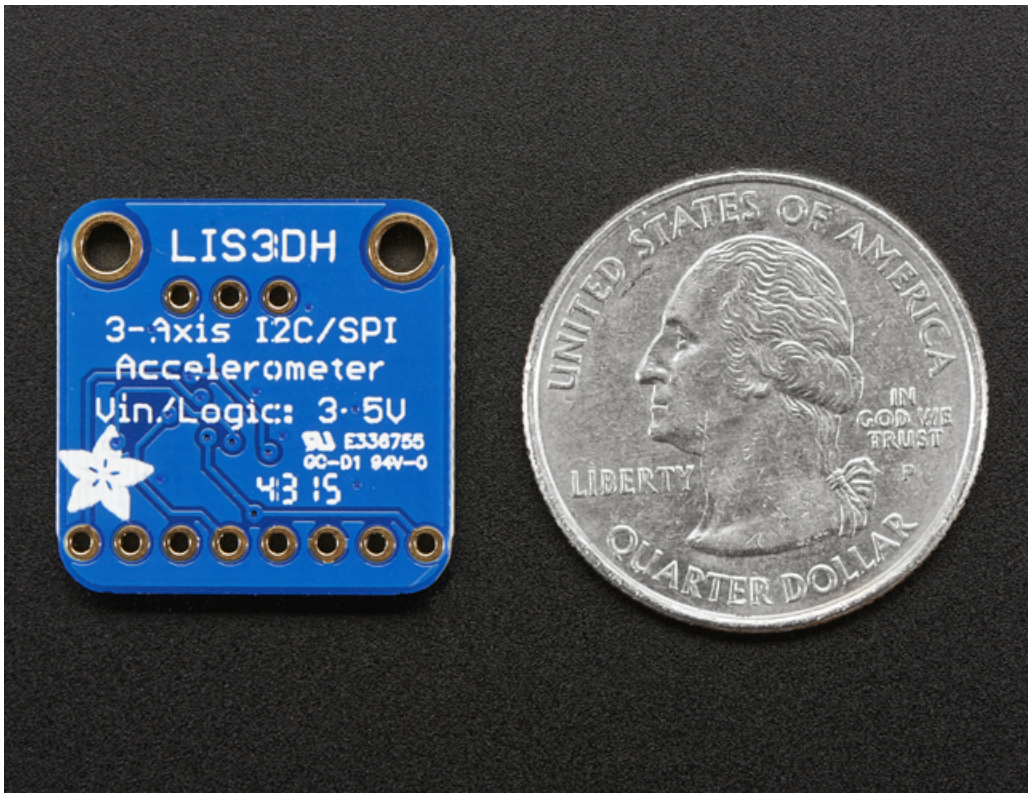
Overview



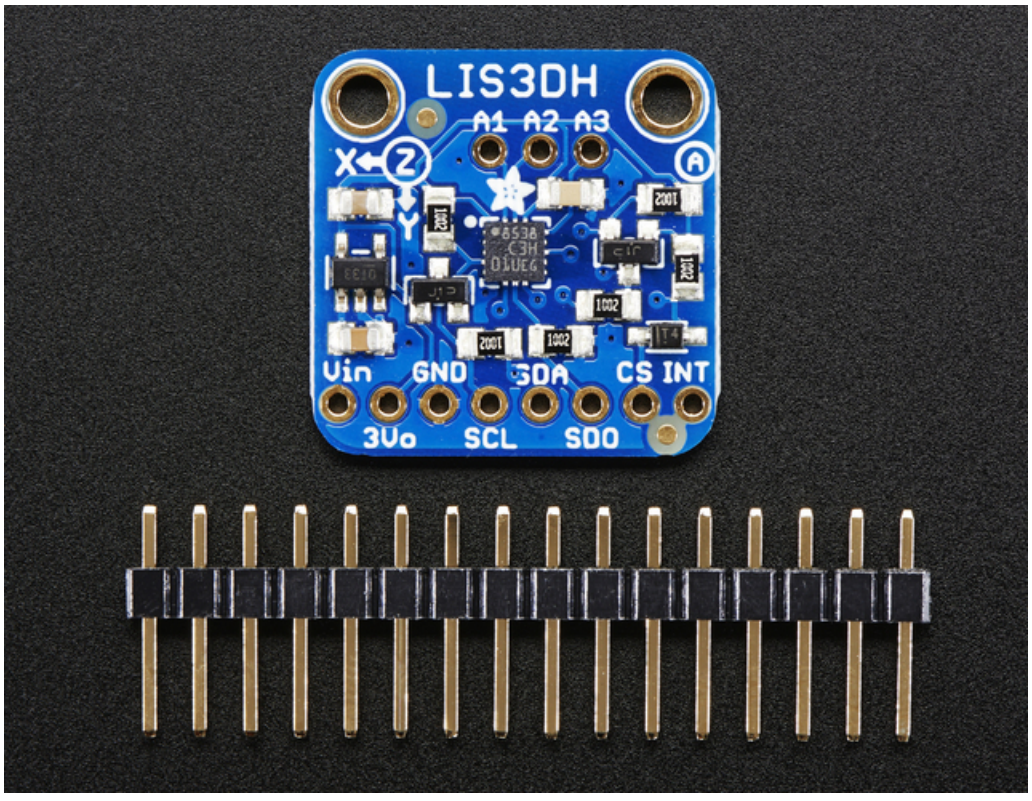
The LIS3DH is a very popular low power triple-axis accelerometer. It's low-cost, but has just about every 'extra' you'd want in an accelerometer:

- Three axis sensing
- ±2g/±4g/±8g/±16g selectable scaling
- Both I2C (2 possible addresses) and SPI interface options
- Interrupt output
- Multiple data rate options 1 Hz to 5Khz
- As low as 2uA current draw (just the chip itself, not including any supporting circuitry)
- Tap, Double-tap, orientation & freefall detection
- 3 additional ADC inputs you can read over I2C

We kept seeing this accelerometer in teardowns of commercial products and figured that if it's the most-commonly used accelerometer, its worth having a breakout board!



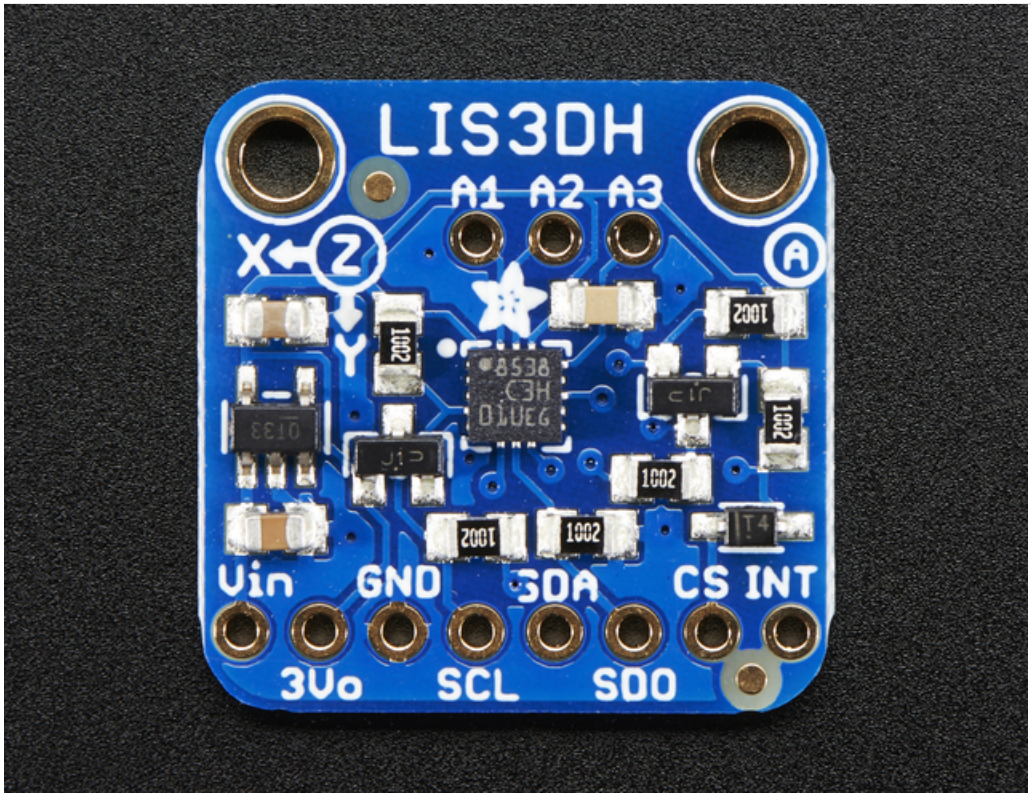
This sensor communicates over I2C *or* SPI (our library code supports both) so you can share it with a bunch of other sensors on the same I2C bus. There's an address selection pin so you can have two accelerometers share an I2C bus.



To get you going fast, we spun up a breakout board for this little guy. Since it's a 3V sensor, we add a low-dropout

3.3V regulator and level shifting circuitry on board. That means its perfectly safe for use with 3V or 5V power and logic. It's fully assembled and tested. Comes with a bit of 0.1" standard header in case you want to use it with a breadboard or perfboard. Two 2.5mm (0.1") mounting holes for easy attachment.

Pinouts



The little chip in the middle of the PCB is the actual LIS3DH sensor that does all the motion sensing. We add all the extra components you need to get started, and 'break out' all the other pins you may want to connect to onto the PCB. For more details you can check out the schematics in the Downloads page.

Power Pins

The sensor on the breakout requires 3V power. Since many customers have 5V microcontrollers like Arduino, we tossed a 3.3V regulator on the board. Its ultra-low dropout so you can power it from 3.3V-5V just fine.

- **Vin** - this is the power pin. Since the chip uses 3 VDC, we have included a voltage regulator on board that will take 3-5VDC and safely convert it down. To power the board, give it the same power as the logic level of your microcontroller - e.g. for a 5V micro like Arduino, use 5V
- **3Vo** - this is the 3.3V output from the voltage regulator, you can grab up to 100mA from this if you like
- **GND** - common ground for power and logic

I2C Pins

- **SCL** - I2C clock pin, connect to your microcontrollers I2C clock line. Has a 10K pullup already on it.
- **SDA** - I2C data pin, connect to your microcontrollers I2C data line. Has a 10K pullup already on it.
- To use I2C, keep the **CS** pin either disconnected or tied to a high (3-5V) logic level.
- **SDO** - When in I2C mode, this pin can be used for address selection. When connected to **GND** or left open, the address is **0x18** - it can also be connected to 3.3V to set the address to **0x19**

SPI pins:

All pins going into the breakout have level shifting circuitry to make them 3-5V logic level safe. Use whatever logic

level is on **Vin!**

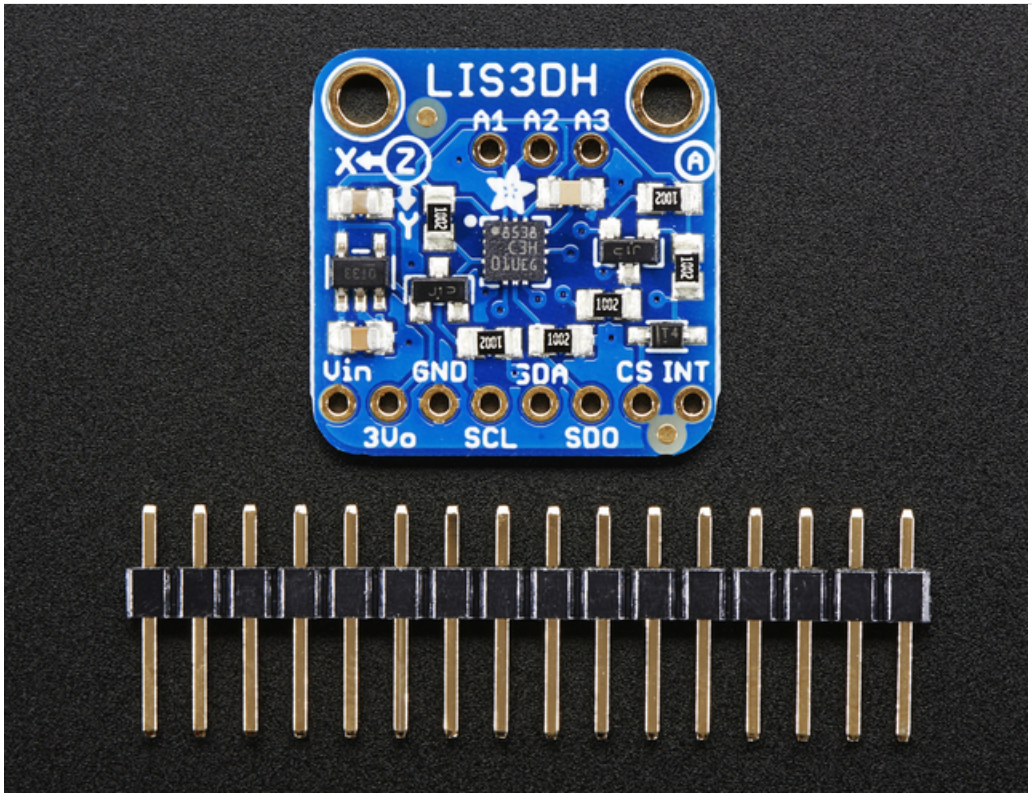
- **SCL** - this is the **S**PI **Cl**oc**k** pin, its an input to the chip
- **SDA** - this is the **S**erial **Da**ta In / **M**aster **O**ut **S**lave **I**n pin, for data sent from your processor to the LIS3DH
- **SDO** - this is the **S**erial **D**ata **O**ut / **M**aster **I**n **S**lave **O**ut pin, for data sent from the LIS3DH to your processor. It's 3.3V logic level out
- **CS** - this is the **C**hip **S**elect pin, drop it low to start an SPI transaction. Its an input to the chip

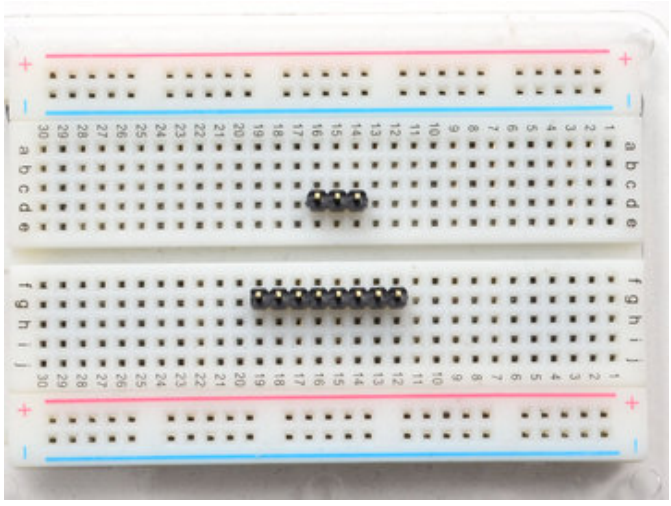
If you want to connect multiple LIS3DH's to one microcontroller, have them share the SDI, SDO and SCK pins. Then assign each one a unique CS pin.

Other Pins

- **INT** is the interrupt output pin. You can configure the interupt to trigger for various 'reasons' such as motion, tilt, taps, data ready etc. This pin is 3.3V logic output.

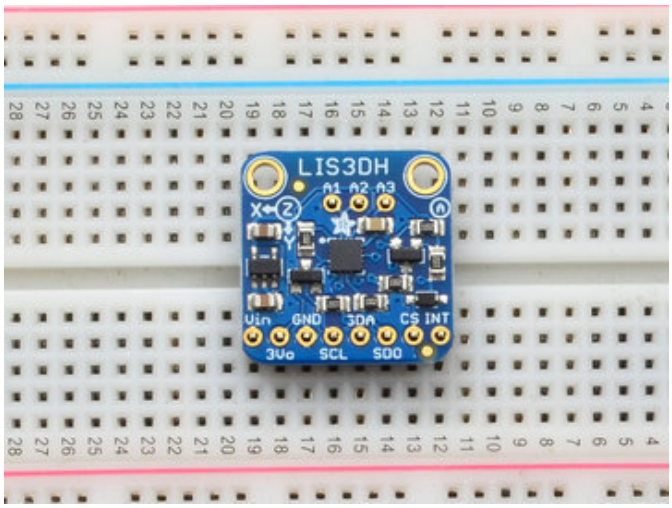
Assembly





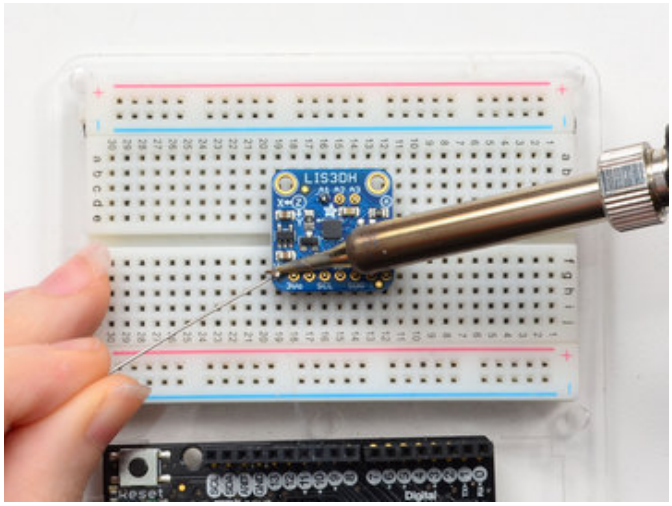
Prepare the header strip:

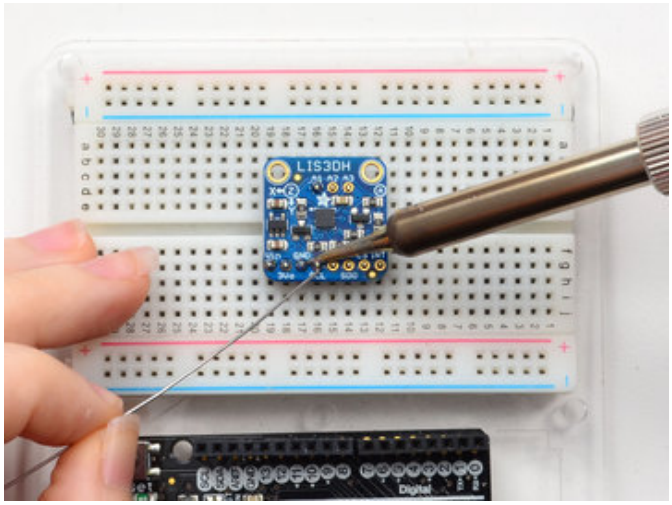
Cut/break the header strip into two pieces, one 3 pin and one 8 pin. It will be easier to solder if you insert it into a breadboard - **long pins down**

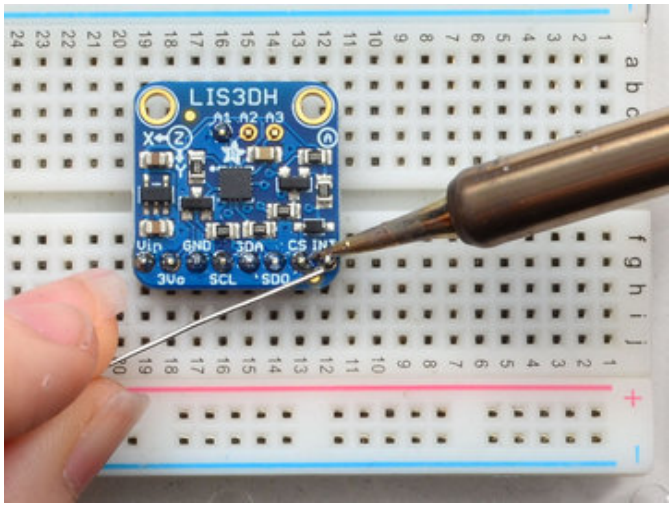


Add the breakout board:

Place the breakout board over the pins so that the short pins poke through the breakout pads



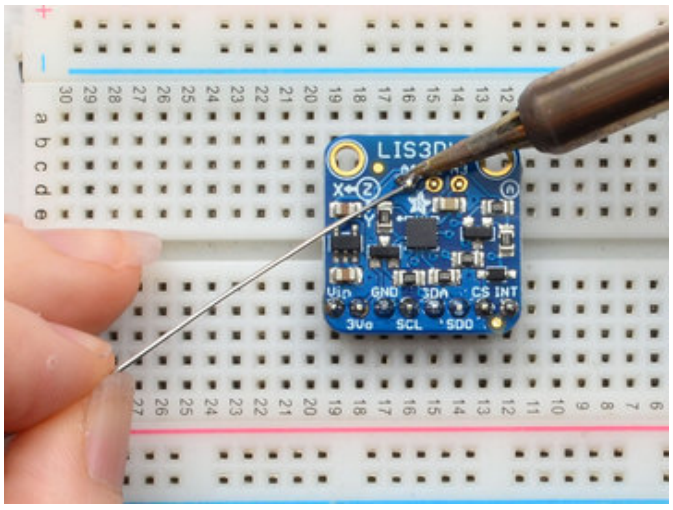




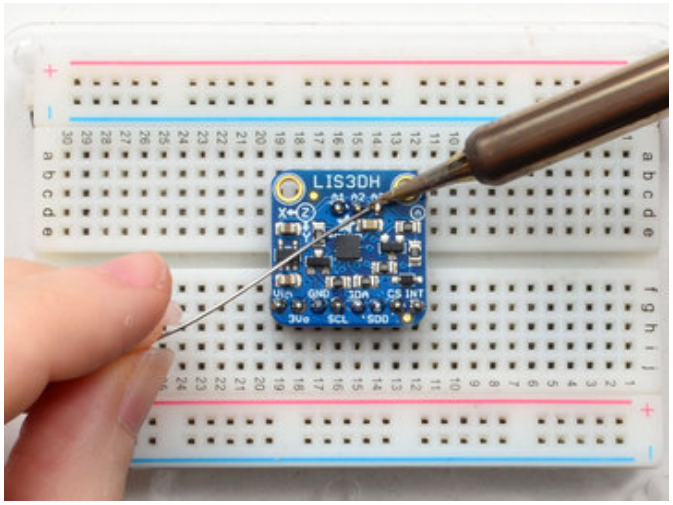
And Solder!

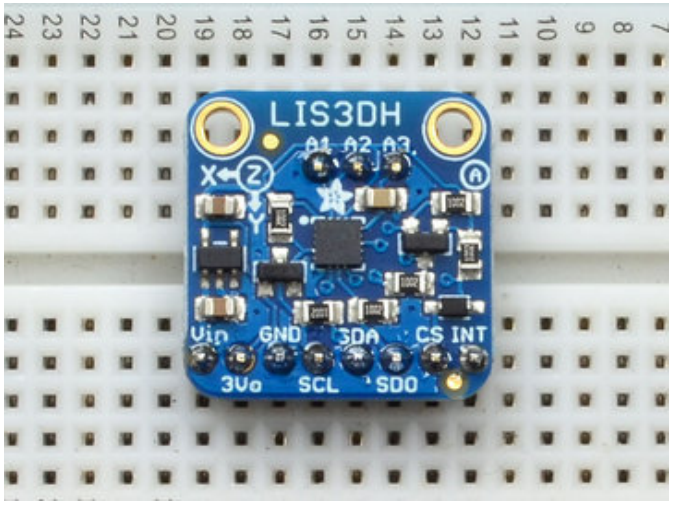
Be sure to solder all pins for reliable electrical contact. We'll start with the main 8 pins for power & the I2C/SPI interface.

(For tips on soldering, be sure to check out our Guide to Excellent Soldering (https://adafru.it/aTk).



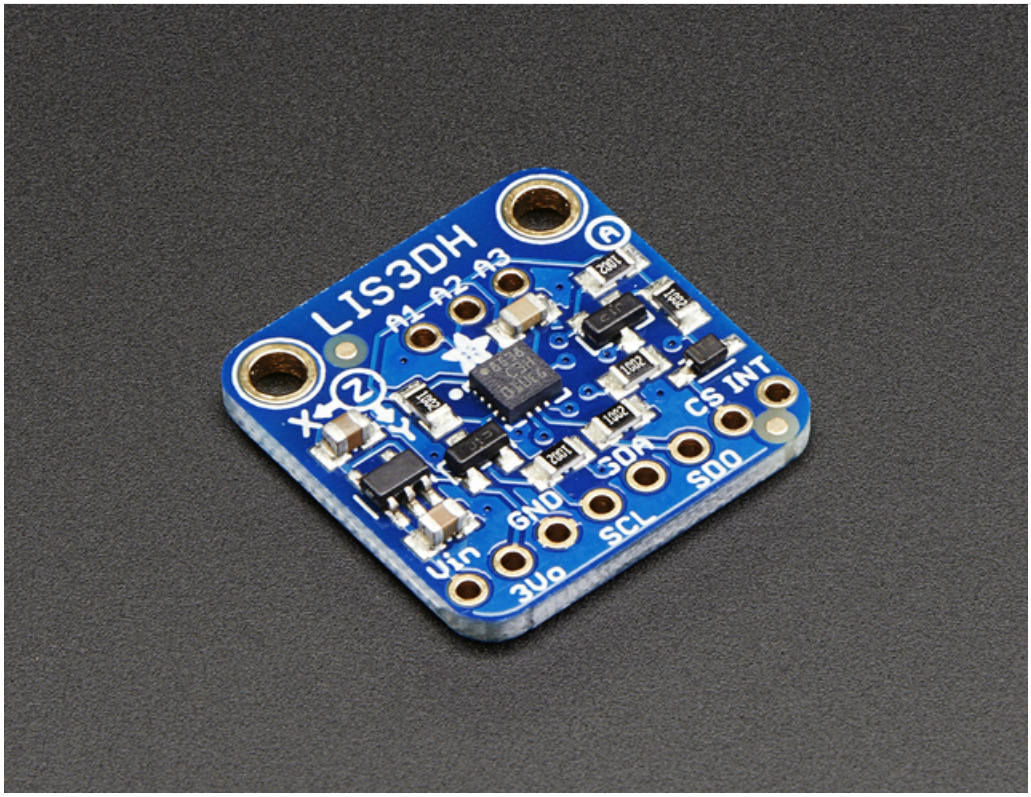
If you want to have the board be more mechanically stable, or want to use the ADC pins, solder in the 3 ADC pads too





You're done! Check your solder joints visually and continue onto the next steps

Wiring & Test

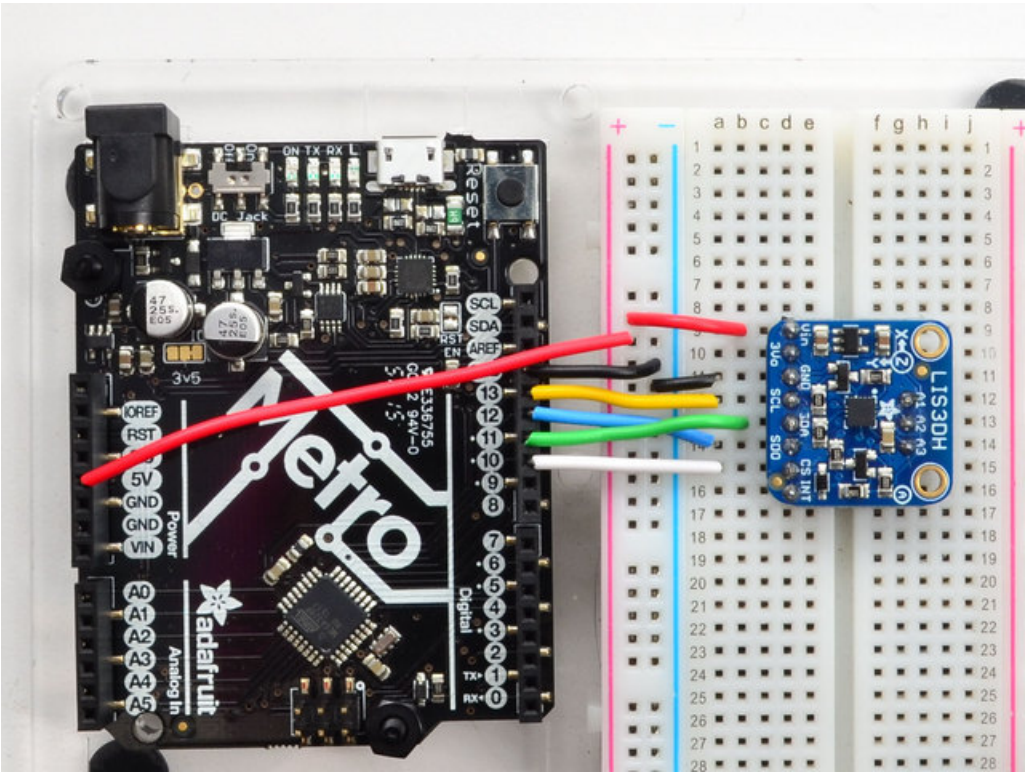


You can easily wire this breakout to any microcontroller, we'll be using an Arduino. For another kind of microcontroller, as long as you have 4 available pins it is possible to 'bit-bang SPI' or you can use two I2C pins, but usually those pins are fixed in hardware. Just check out the library, then port the code.

I2C Wiring

Use this wiring if you want to connect via I2C interface

- Connect **Vin** to the power supply, 3-5V is fine. Use the same voltage that the microcontroller logic is based off of. For most Arduinos, that is 5V
- Connect **GND** to common power/data ground
- Connect the **SCL** pin to the I2C clock **SCL** pin on your Arduino. On an UNO & '328 based Arduino, this is also known as **A5**, on a Mega it is also known as **digital 21** and on a Leonardo/Micro, **digital 3**
- Connect the **SDA** pin to the I2C data **SDA** pin on your Arduino. On an UNO & '328 based Arduino, this is also known as **A4**, on a Mega it is also known as **digital 20** and on a Leonardo/Micro, **digital 2**



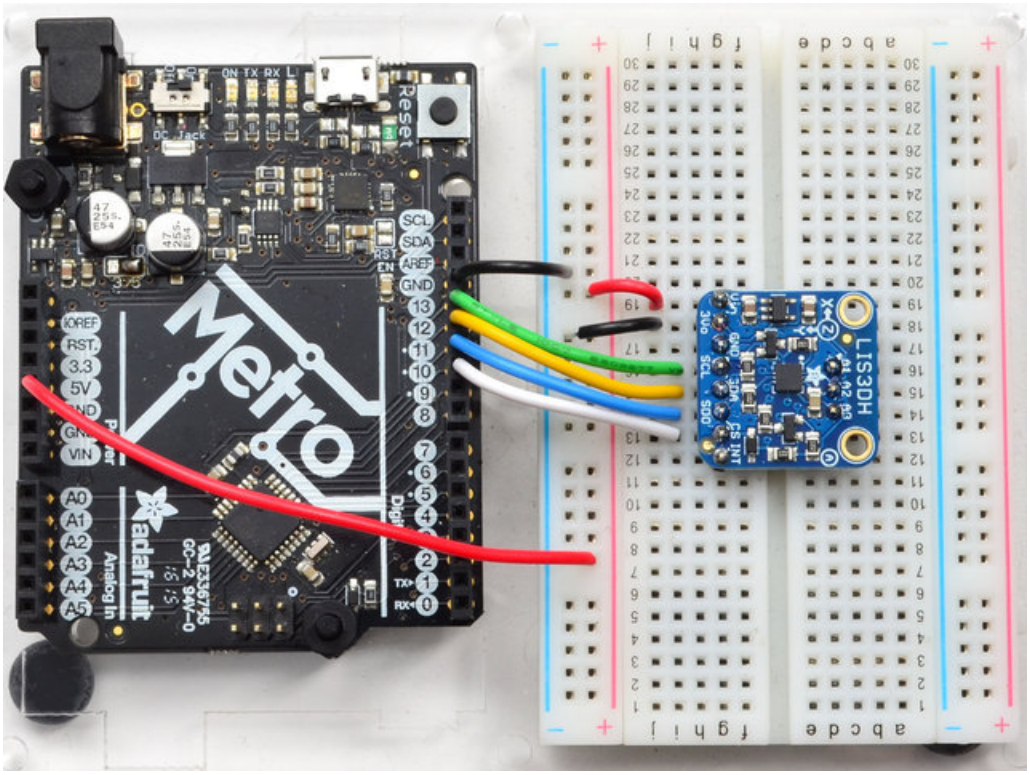
SPI Wiring

Since this is also an SPI-capable sensor, we can use hardware or 'software' SPI. To make wiring identical on all Arduinos, we'll begin with 'software' SPI. The following pins should be used:

- Connect **Vin** to the power supply, 3V or 5V is fine. Use the same voltage that the microcontroller logic is based off of. For most Arduinos, that is 5V
- Connect **GND** to common power/data ground
- Connect the **SCL (SCK)** pin to **Digital #13** but any pin can be used later
- Connect the **SDO** pin to **Digital #12** but any pin can be used later
- Connect the **SDA (SDI)** pin to **Digital #11** but any pin can be used later
- Connect the **CS** pin **Digital #10** but any pin can be used later

Later on, once we get it working, we can adjust the library to use hardware SPI if you desire, or change the pins to other pins.

Note: Photo below incorrectly shows GND to 3Vo. The correct connection is GND to GND as listed above!



Download Adafruit_LIS3DH library

To begin reading sensor data, you will need to download Adafruit_LIS3DH from our github repository. You can do that by visiting the github repo and manually downloading or, easier, just click this button to download the zip

Download Adafruit LIS3DH library

https://adafru.it/jzc

Rename the uncompressed folder **Adafruit_LIS3DH** and check that the **Adafruit_LIS3DH** folder contains **Adafruit_LIS3DH.cpp** and **Adafruit_LIS3DH.h**

Place the **Adafruit_LIS3DH** library folder your ***arduinosketchfolder*/libraries/** folder.
You may need to create the **libraries** subfolder if its your first library. Restart the IDE.

We also have a great tutorial on Arduino library installation at:
http://learn.adafruit.com/adafruit-all-about-arduino-libraries-install-use

Download Adafruit_Sensor

The LIS3DH library uses the Adafruit_Sensor support backend so that readings can be normalized between sensors. You can grab Adafruit_Sensor from the github repo or just click the button below.

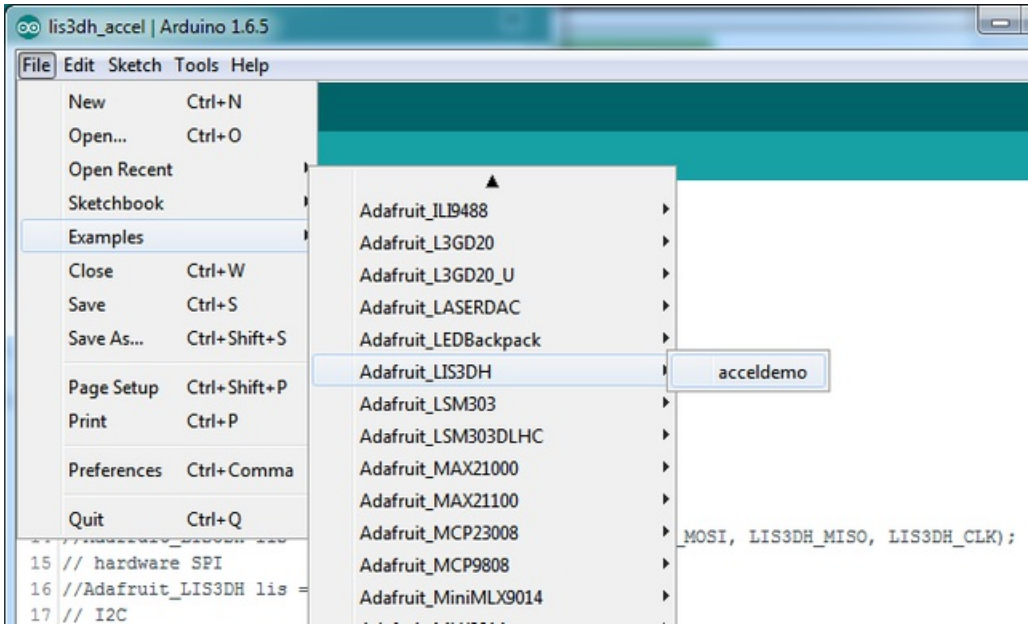
Download Adafruit Sensor library

https://adafru.it/cMO

Install it just like you did with the Adafruit_LIS3DH library above!

Accelerometer Demo

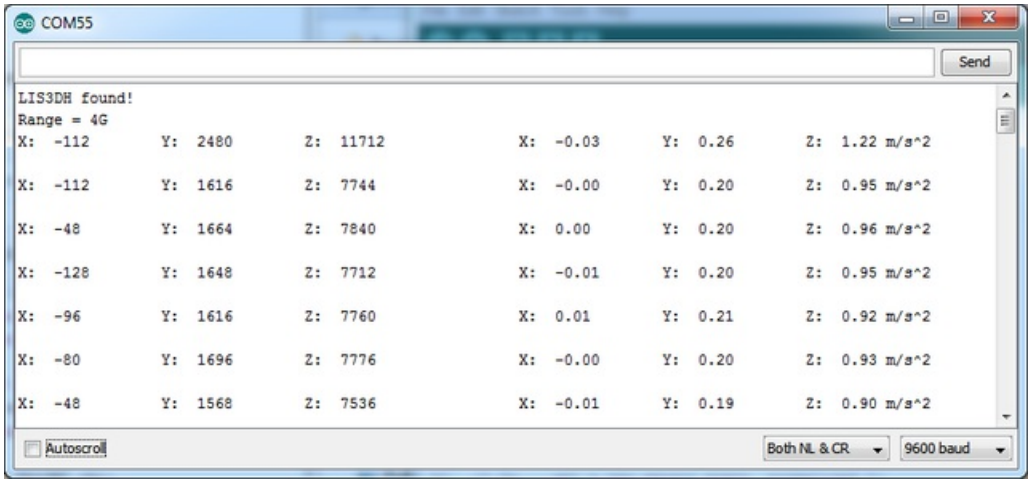
Open up **File->Examples->Adafruit_LIS3DH->acceldemo** and upload to your Arduino wired up to the sensor



Depending on whether you are using I2C or SPI, change the pin names and comment or uncomment the following lines.

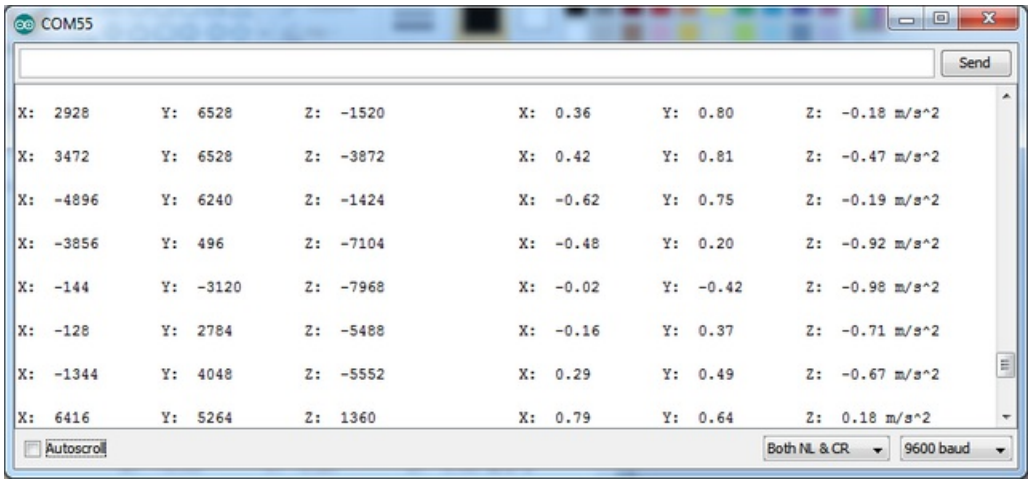
```
// Used for software SPI
#define LIS3DH_CLK 13
#define LIS3DH_MISO 12
#define LIS3DH_MOSI 11
// Used for hardware & software SPI
#define LIS3DH_CS 10

// software SPI
//Adafruit_LIS3DH lis = Adafruit_LIS3DH(LIS3DH_CS, LIS3DH_MOSI, LIS3DH_MISO, LIS3DH_CLK);
// hardware SPI
//Adafruit_LIS3DH lis = Adafruit_LIS3DH(LIS3DH_CS);
// I2C
Adafruit_LIS3DH lis = Adafruit_LIS3DH();
```

Once uploaded to your Arduino, open up the serial console at 9600 baud speed to see data being printed out



Normally, sitting on a table, you'll see X and Y are close to **0** and Z will be about ~**1 m/s^2** because the accelerometer is measuring the force of gravity! (My table is a bit tilted so Y is closer to 0.2 m/s^2.

You can also move around the board to measure your movements and also try tilting it to see how the gravity 'force' appears at different axes.



Not that you'll see there's *two* sets of data!

Accelerometer ranges

Accelerometeres don't 'naturally' spit out a "meters per second squared" value. Instead, they give you a raw value. In this sensor's case, its a number ranging from -32768 and 32767 (a full 16-bit range). Depending on the sensor range, this number scales between the min and max of the range. E.g. if the accelerometer is set to +-**2g** then 32767 is +2g of force, and -32768 is -2g. If the range is set to +-**16g**, then those two number correlate to +16g and -16g respectively. So knowing the range is key to deciphering the data!

You can set, and get the range with:

```
lis.setRange(LIS3DH_RANGE_4_G);   // 2, 4, 8 or 16 G!
Serial.print("Range = "); Serial.print(2 << lis.getRange());
```

In the first line, you can use **LIS3DH_RANGE_2_G**, **LIS3DH_RANGE_4_G**, **LIS3DH_RANGE_8_G**,

or **LIS3DH_RANGE_16_G**

When reading the range back from the sensor, **0** is ±2g, **1** is ±4g, **2** is ±8g and **3** is ±16g range

Raw data readings

You can get these raw readings by calling `lis.read()` which will take a snapshot at that moment in time. You can then grab the x, y and z data by reading the signed 16-bit values from `lis.x`, `lis.y` and `lis.z`. When you are done with that data, call `read()` again to get another snapshot.

Normalized readings

If you dont want to noodle around with range readings and scalings, you can use **Adafruit_Sensor** to do the normalization for you. Its a nice way to have consistant readings betwixt multiple types of accelerometers.

Adafruit_Sensor uses event (`sensors_event_t`) objects to 'snapshot' the data:

```
/* Get a new sensor event */
sensors_event_t event;
lis.getEvent(&event);
```

Once you've `getEvent`'d the data, read it out from the event object with `event.acceleration.x`, `event.acceleration.y` & `event.acceleration.z`

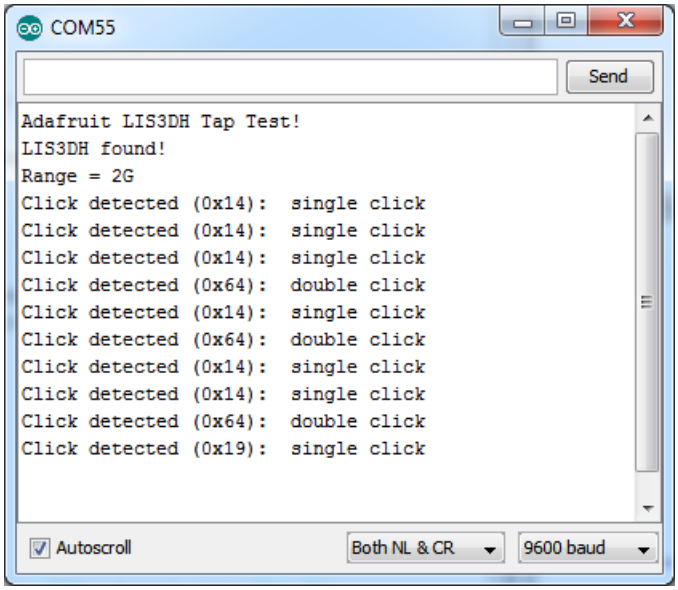
The key point is that those values are floating point, and are going to be in m/s^2 No matter the range!

It's up to you whether you want the raw numbers or normalized data - there's times you want to keep it simple and avoid floating point numbers, and other times you may want to avoid doing the math for force conversion.

Tap & Double tap detection

One of the neat extras in the LIS3DH is tap & double-tap detection. By tapping the accelerometer or the PCB or something the PCB is attached to you can create a type of interface.

Open up the **tapdemo** demo to run it! It defaults to I2C so change to SPI if you're using that interface.



The tap detection works by looking for when one of the axes has an acceleration higher than a certain **threshhold** for longer than a certain **timelimit**. The threshhold is in 'raw' values so you have to adjust it based on the scale/range you've configured for the sensor:

```
// Adjust this number for the sensitivity of the 'click' force
// this strongly depend on the range! for 16G, try 5-10
// for 8G, try 10-20. for 4G try 20-40. for 2G try 40-80
#define CLICKTHRESHHOLD 80
```

Larger numbers are less sensitive, you'll really need to just tweak as necessary for your project.

You'll also have to turn on click detection. This will also se the **INT** pin to pulse high whenever a tap or double tap is detected. If you turn on double tap detection it will still 'decode' single taps but the INT pin wont pulse on them.

```
  // 0 = turn off click detection & interrupt
  // 1 = single click only interrupt output
  // 2 = double click only interrupt output, detect single click
  // Adjust threshhold, higher numbers are less sensitive
  lis.setClick(2, CLICKTHRESHHOLD);
```

setClick can actually take many more arguments, the full list is:

```
  void setClick(uint8_t c, uint8_t clickthresh, uint8_t timelimit = 10, uint8_t timelatency = 20, uint8_
```

The **timelimit timelatency** and **timewindow** are in units of "ODR" so if you change the sample frequency of the LIS3DH you'll have to adjust these as necessary. The App note has way more details on this, with pretty graphs & eveything!

Figure 15. Single and double click recognition

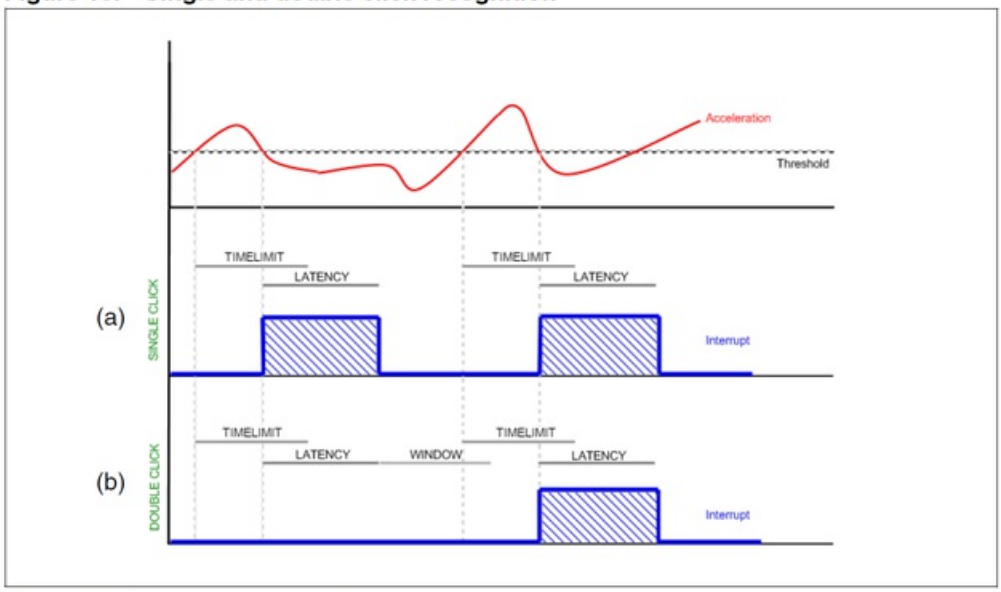


Figure 15 illustrates a single click event (a) and a double click event (b). The device is able to distinguish between (a) and (b) by changing the settings of the TAP_CFG register from single to double click recognition.

You can get the current click status register with `lis.getClick()`. Note that will return the raw 8-bit reg known as **LIS3DH_REG_CLICKSRC**

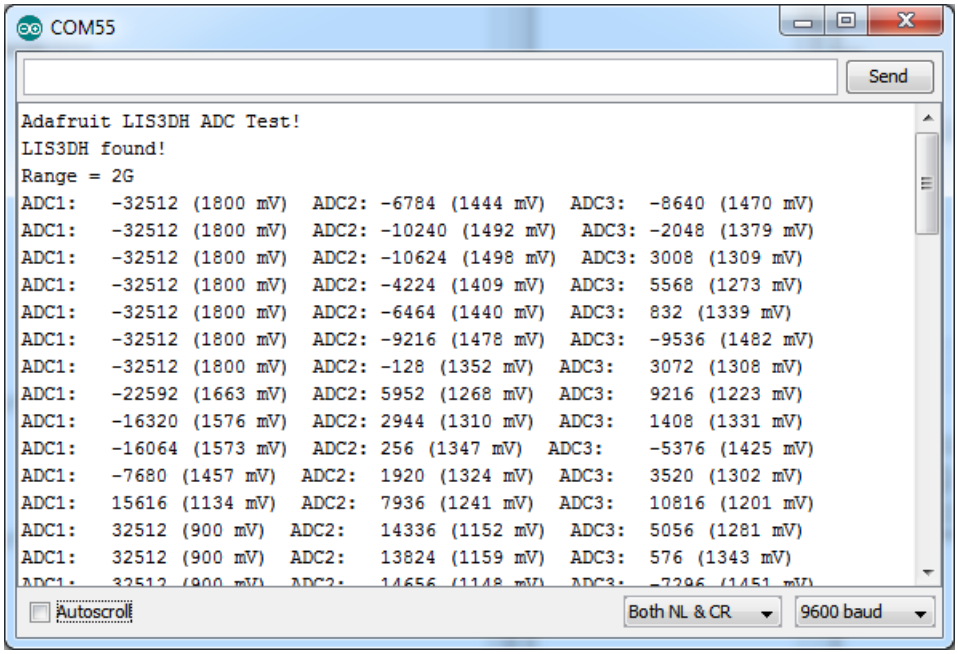
Even though that register has 'axis' bits that theoretically tell you which axis the click is from, unless you bolt the breakout to a huge thing like a table and thwack the table, any 'direct hits' to the sensor itself will register on any/all axes because of the small scale.

Reading the 3 ADC pins

Finally, there are 3 extra 10-bit analog-digital-converter pins available. The details on these pins is scarce but we do have code for reading data for them. This might be handy if you have a device without any ADC's

While you can put 0-3.3V into the ADC pins, the valid reading range is only ~0.9V to 1.8V

You can load the **readadc** demo sketch to start reading



I connected a trimpot from ground (left pin) to 3.3V (right pin) and conneted the wiper (center pin) to **ADC1**

For pins that don't have anything connected, such as ADC2 and ADC3 you'll notice the values flicker around a lot. If you don't like this, just tie the pins to **GND** and it will keep the values 'steady'

The values of the ADC range from -32512 to 32512 **but** note that they correspond to ~1.8V to ~0.9V. You can map the raw values to an approx voltage using

```
volt = map(adc, -32512, 32512, 1800, 900);
```

Which will convert the raw **adc** value into **volt** in units of millivolts

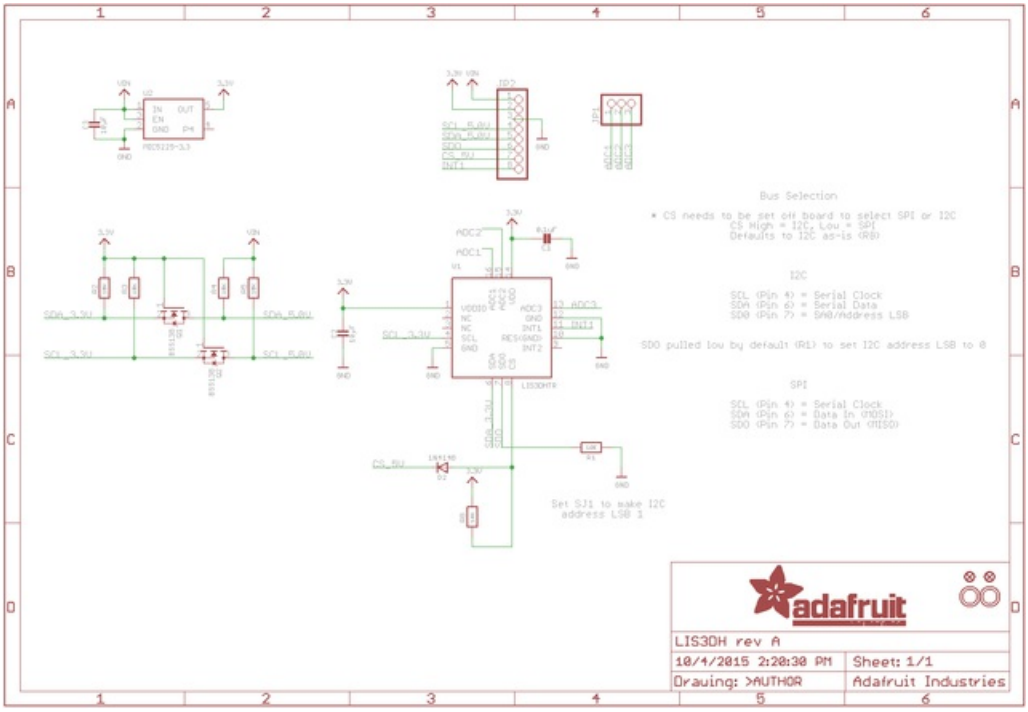
Downloads

Datasheets

- LIS3DH datasheet
- LIS3DH app note
- ST design resources
- Fritzing object available in the Adafruit Fritzing Library
- EagleCAD PCB files on GitHub

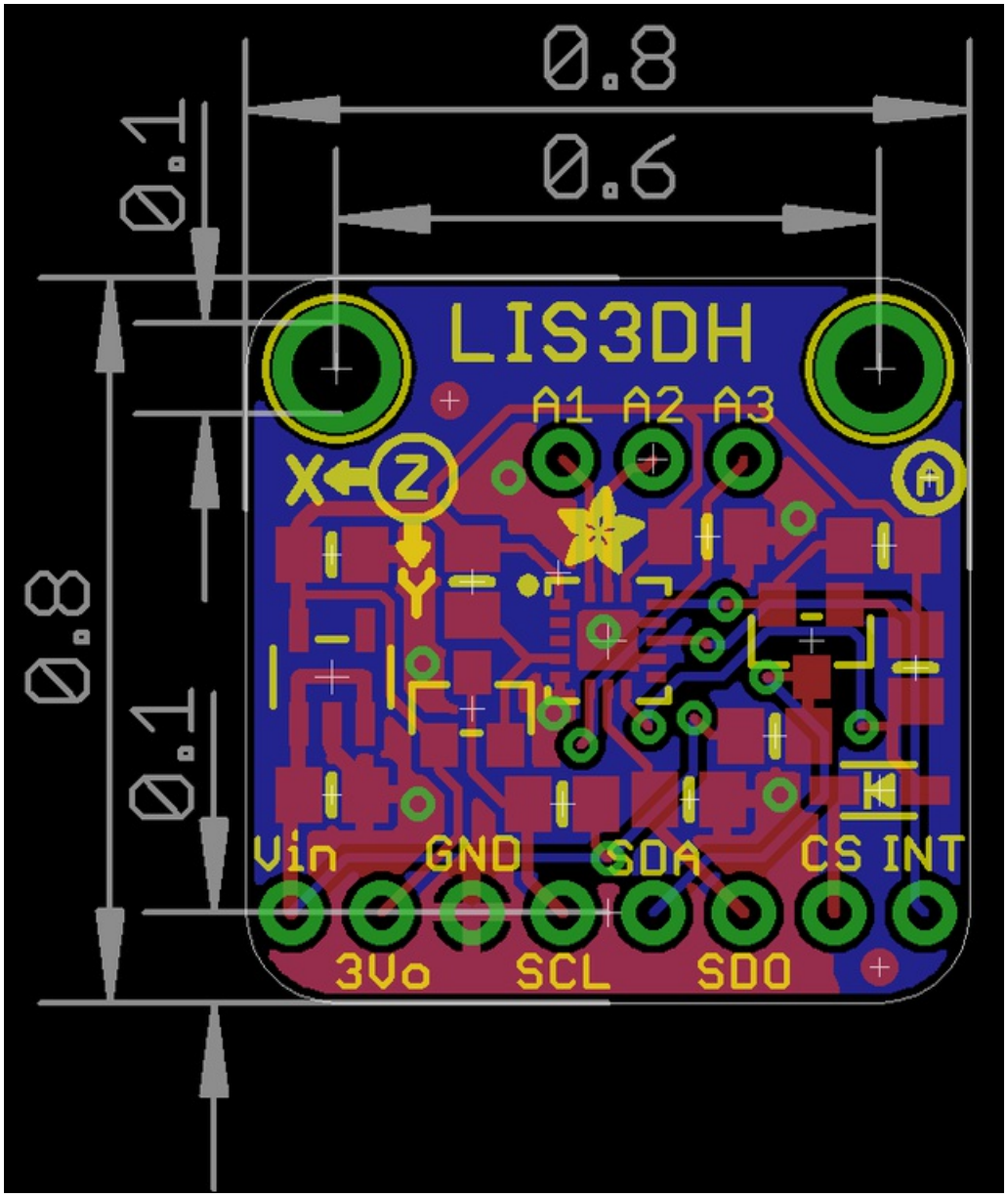
Schematic

click to enlarge



Fabrication Print

Dimensions in inches


0.8
0.6
0.1
0.8
0.1
LIS3DH
A1 A2 A3
X
Z
Y
Vin GND SDA CS INT
3Vo SCL SDO