Chipsmall Limited consists of a professional team with an average of over 10 year of expertise in the distribution of electronic components. Based in Hongkong, we have already established firm and mutual-benefit business relationships with customers from,Europe,America and south Asia,supplying obsolete and hard-to-find components to meet their specific needs.

With the principle of “Quality Parts,Customers Priority,Honest Operation,and Considerate Service”,our business mainly focus on the distribution of electronic components. Line cards we deal with include Microchip,ALPS,ROHM,Xilinx,Pulse,ON,Everlight and Freescale. Main products comprise IC,Modules,Potentiometer,IC Socket,Relay,Connector.Our parts cover such applications as commercial,industrial, and automotives areas.

We are looking forward to setting up business relationship with you and hope to provide you with the best service and solution. Let us make a better world for our industry!

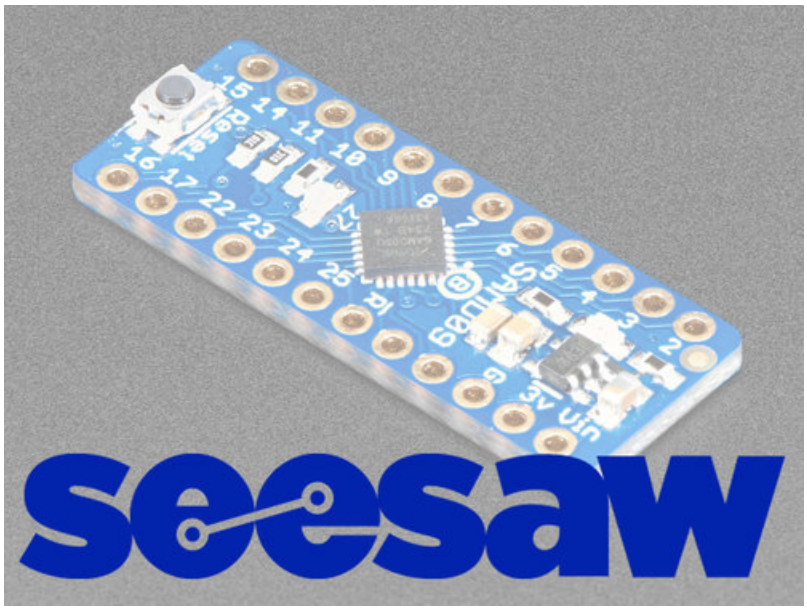
Contact us

Tel: +86-755-8981 8866 Fax: +86-755-8427 6832
Email & Skype: info@chipsmall.com Web: www.chipsmall.com
Address: A1208, Overseas Decoration Building, #122 Zhenhua RD., Futian, Shenzhen, China

adafruit learning system

Adafruit seesaw

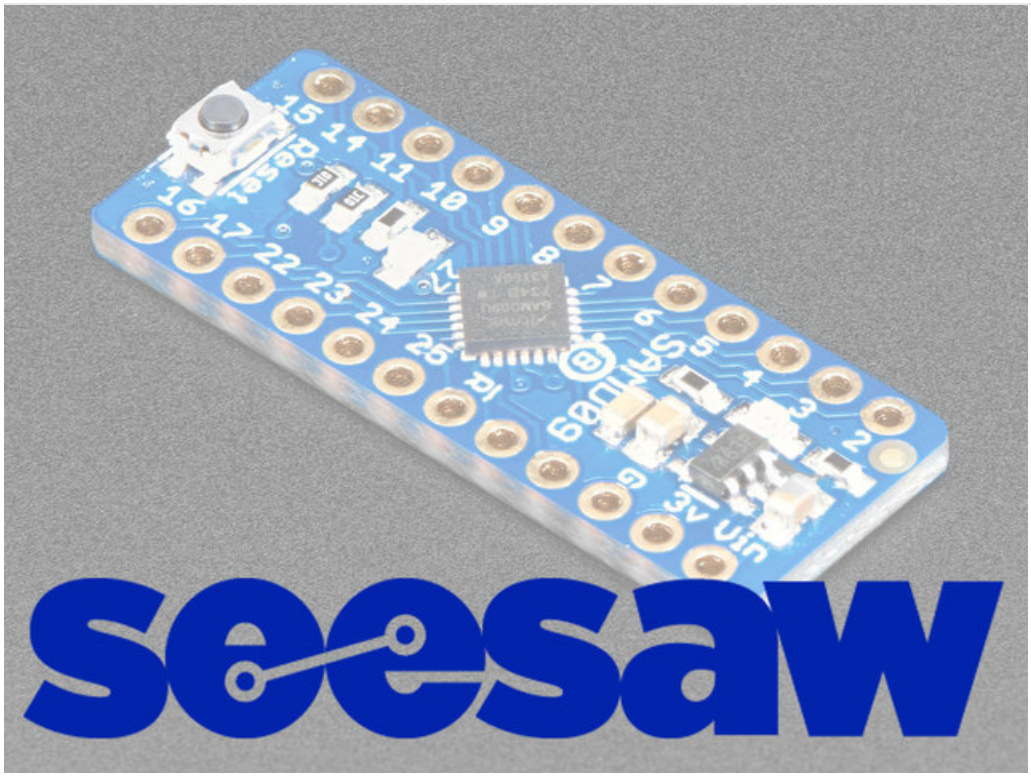
Created by Dean Miller



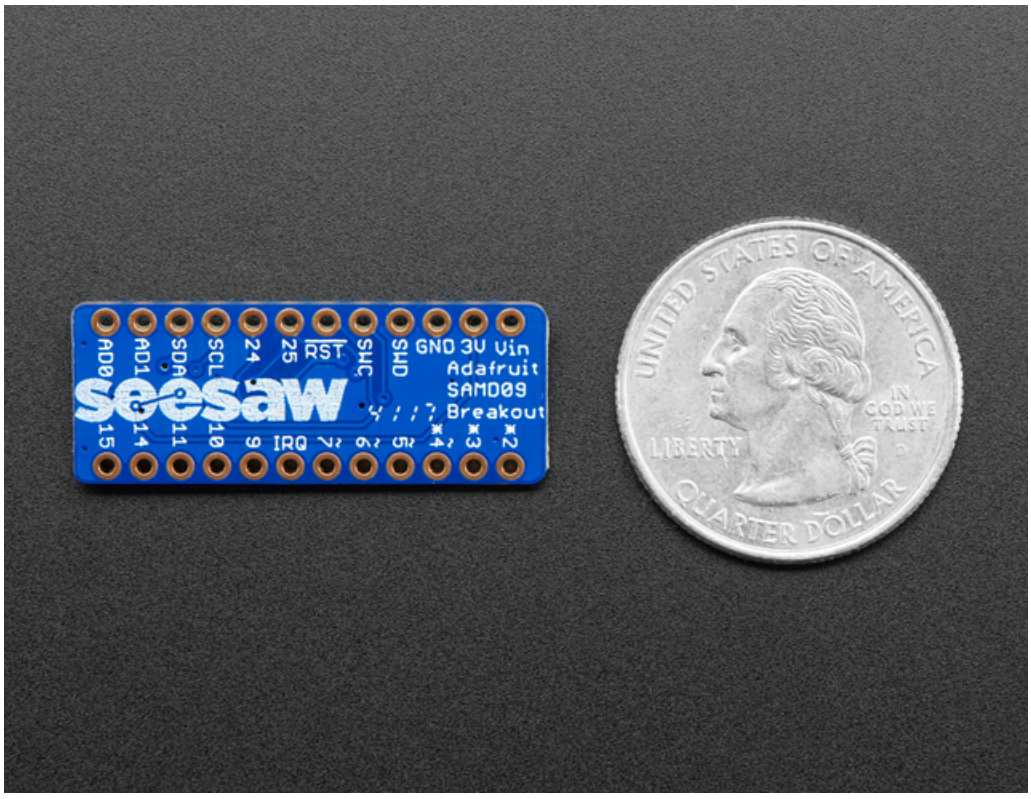
Last updated on 2018-08-12 12:38:05 AM UTC

Guide Contents

Overview



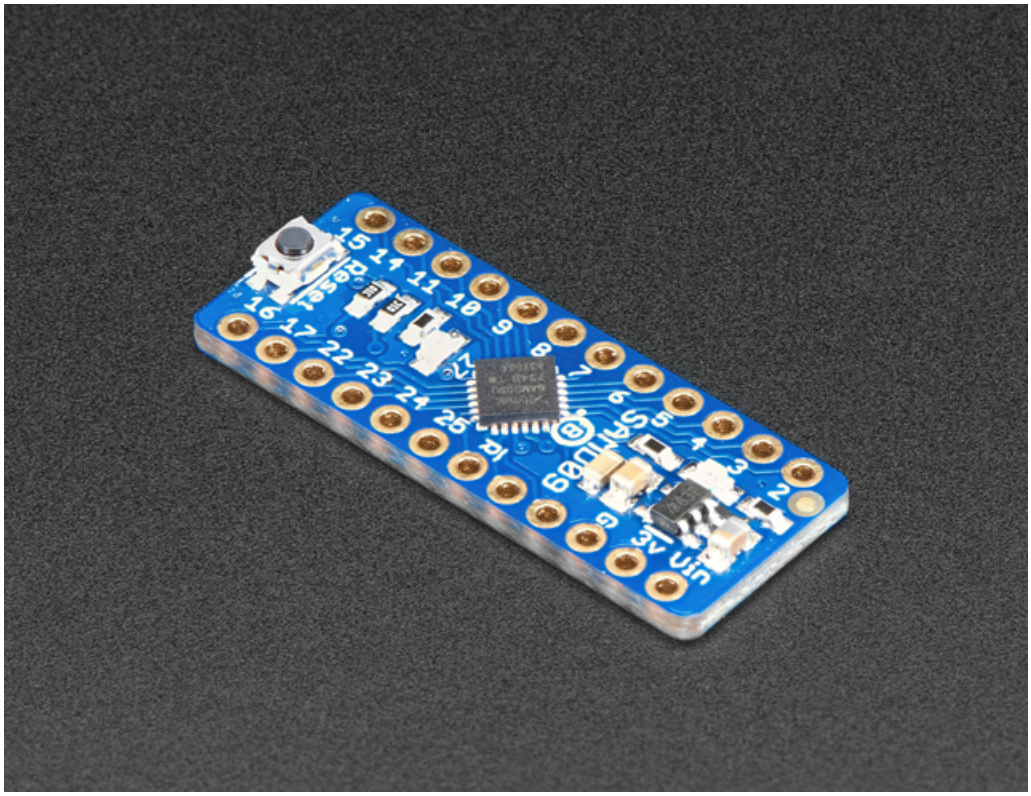
Adafruit seesaw is a near-universal converter framework which allows you to add add and extend hardware support to any I2C-capable microcontroller or microcomputer. Instead of getting separate I2C GPIO expanders, ADCs, PWM drivers, etc, seesaw can be configured to give a wide range of capabilities.



For example, our ATSAMD09 breakout with seesaw gives you

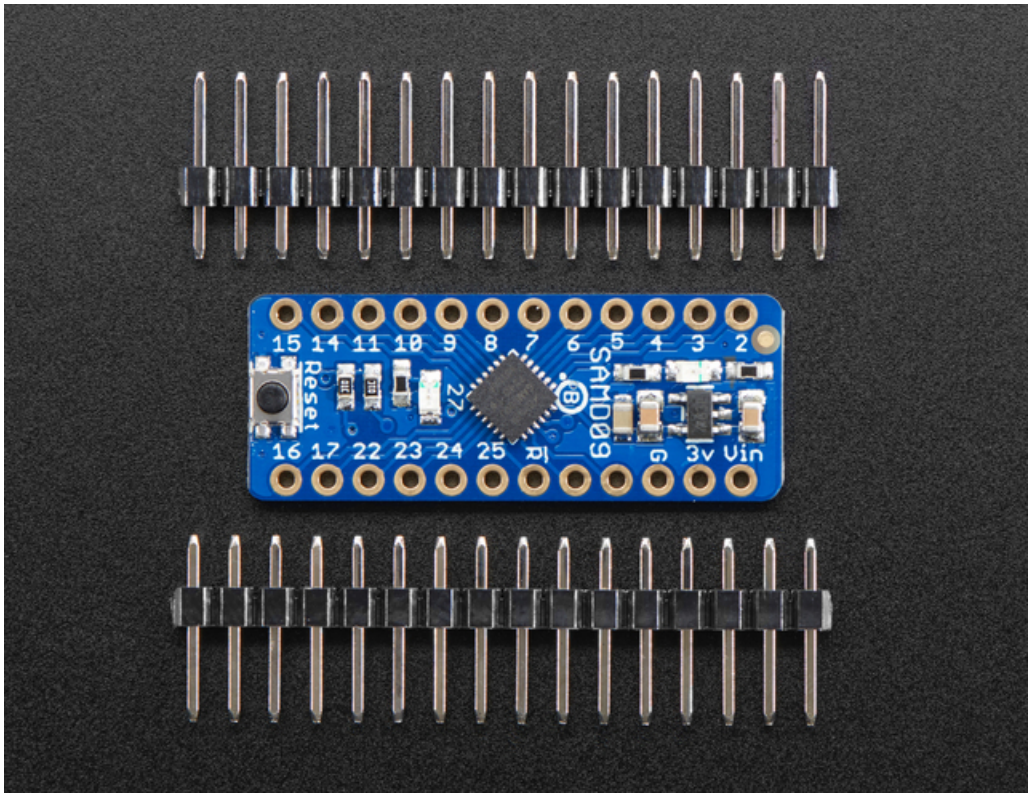
- 3 x 12-bit ADC inputs
- 3 x 8-bit PWM outputs
- 7 x GPIO with selectable pullup or pulldown
- 1 x NeoPixel output (up to 340 pixels)
- 1 x EEPROM with 64 byte of NVM memory (handy for storing small access tokens or MAC addresses)
- 1 x Interrupt output that can be triggered by any of the accessories
- 2 x I2C address selection pins
- 1 x Activity LED

But you can reprogram and reconfigure the chip to have more or less of each peripheral - as long as it fits into the ATSAMD09D14's firmware! For example, there's also a UART converter but it isn't included in the default firmware.

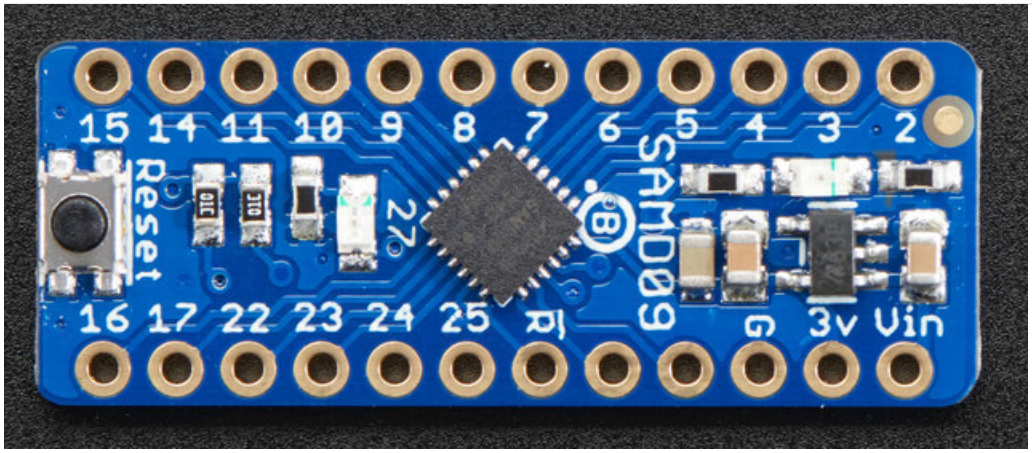


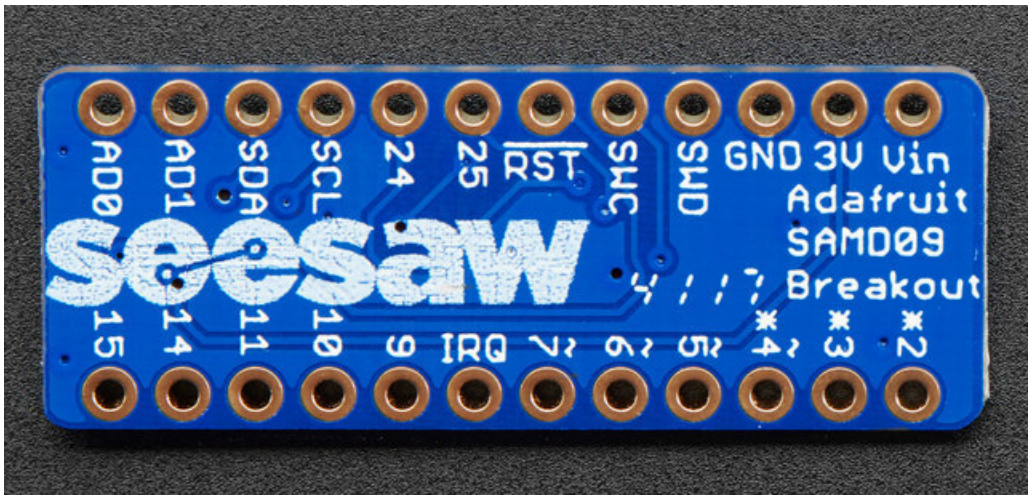
The ATSAMD09 breakout is great for development of seesaw capabilities (we use it in-house for our design work) or you can use it as-is to give your Raspberry Pi or ESP8266 more hardware support! Each breakout comes with the assembled and tested board, as well as some header strips.

Please note: The boards do not come with a bootloader. If you want to do development using seesaw you'll need to pick up a J-Link (https://adafru.it/BrS) and we recommend a SWD adapter breakout (https://adafru.it/u7d) - at this time our project is for Atmel Studio but you could probably get it working with arm gcc and a Makefile. We don't provide any support for custom builds of seesaw - we think this is cool and useful for the Maker community!


15 14 11 10 9 8 7 6 5 4 3 2
Reset
SAMD09
16 17 22 23 24 25 A1 G 3v Vin

Pinouts





Power Pins:

- **Vin** - this is the power pin. Since the ATSAMD09 uses 3.3V, we have included an on-board voltage regulator that will take 3-5VDC and safely convert it down. You can power from 3.3V to 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

Logic Pins:

- **23 / SCL** - this is the I2C clock pin, connect to your microcontrollers I2C clock line. There is a 10K pullup on this pin to 3.3V. I2C is 'open drain' which means as long as you don't add an extra pullup you can use with 5V logic devices.
- **22 / SDA** - this is the I2C data pin, connect to your microcontrollers I2C data line. There is a 10K pullup on this pin to 3.3V. I2C is 'open drain' which means as long as you don't add an extra pullup you can use with 5V logic devices.
- **RST** - this is the reset pin. Pulling this pin to ground resets the device.

GPIO Pins:

- Pins **9, 10, 11, 14, 15, 24, and 25** can be used as GPIO.

Neopixel Pins:

- Pins **9, 10, 11, 14, 15, 24, and 25** can be used as the NeoPixel output.

Address Pins:

- **16 / AD0** - this is the ADDR0 pin. Connect this to ground to increment the devices I2C address by 1.
- **17 / AD1** - this is the ADDR1 pin. Connect this to ground to increment the devices I2C addres by 2.

ADC Pins:

- **2** - this pin can be configured as an ADC input.
- **3** - this pin can be configured as an ADC input.
- **4** - this pin can be configured as an ADC input.

PWM Pins:

- **5** - this pin can be configured as a PWM output.
- **6** - this pin can be configured as a PWM output.
- **7** - this pin can be configured as a PWM output.

Interrupt Pins:

- **8 / IRQ** - this pin gets pulled low by the seesaw to signal to your host microcontroller that an interrupt has occurred.

Programming Pins:

- **SWD** - this pin connects to SWDIO of an SWD compatible programmer to program the device over SWD.
- **SWC** - this pin connects to SWCLK of an SWD compatible programmer to program the device over SWD.
- **RST** - this pin connects to RESET of an SWD compatible programmer to program the device over SWD.

Arduino Wiring & Test

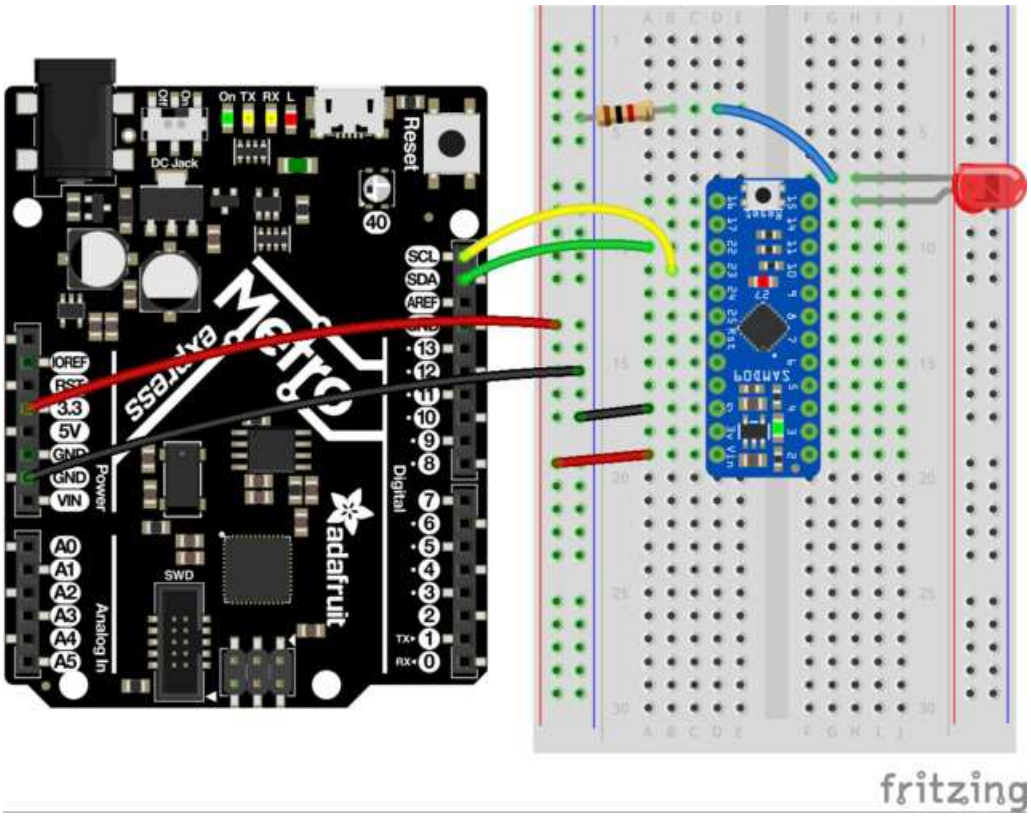
Arduino Wiring & Test

You can easily wire this breakout to any microcontroller, we'll be using an Adafruit Metro M0 Express (Arduino compatible) with the Arduino IDE. But, you can use any other kind of microcontroller as well as long as it has I2C clock and I2C data lines.

I2C Wiring

- Connect **Vin** to the power supply, 3-5V is fine.
- Connect **GND** to common power/data ground
- Connect the **SCL** pin (**23**) to the I2C clock **SCL** pin on your Microcontroller.
- Connect the **SDA** pin (**22**) to the I2C data **SDA** pin on your Microcontroller.
- Connect the positive (long lead) of an LED to pin **15** on the seesaw breakout and the other lead to **GND** through a **1k ohm resistor.**

This seesaw uses I2C address **0x49** by default. You can change this by grounding the **AD0/16** and/or **AD1/15** pins, but we recommend not doing that until you have it working



Download Adafruit_Seesaw library

To begin reading sensor data, you will need to download Adafruit_Seesaw 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

https://adafru.it/A0t

https://adafru.it/A0t

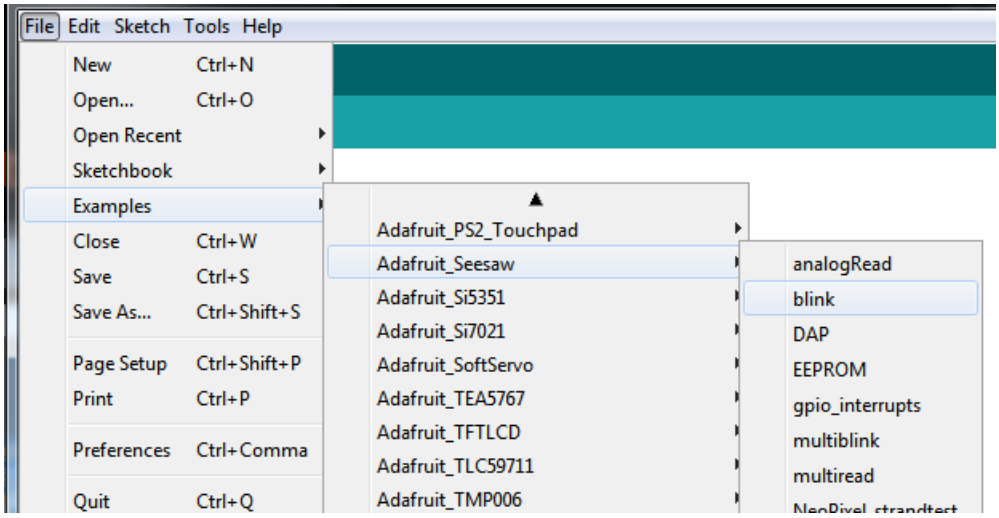
Rename the uncompressed folder **Adafruit_Seesaw** and check that the **Adafruit_Seesaw** folder contains **Adafruit_Seesaw .cpp** and **Adafruit_Seesaw .h**

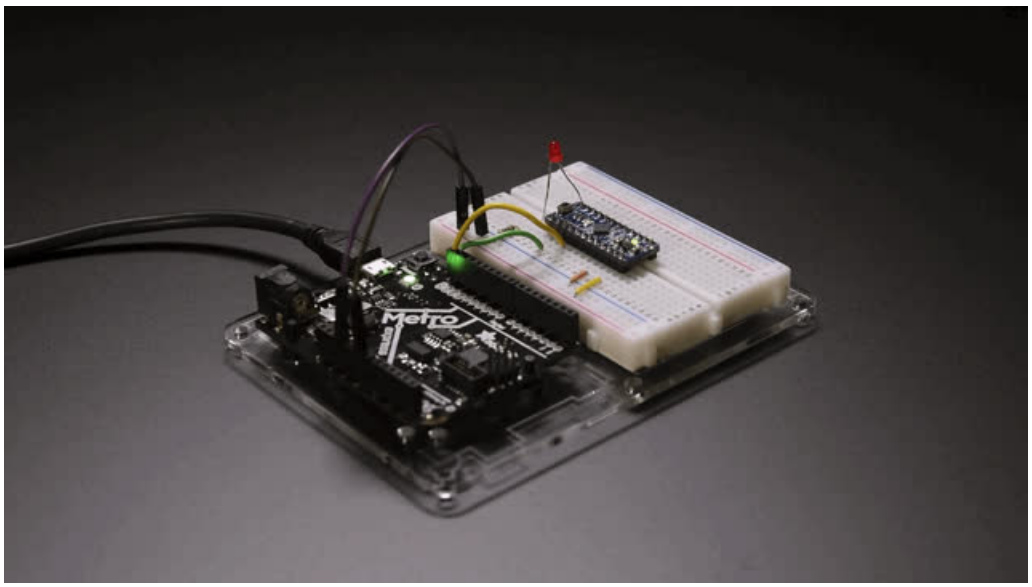
Place the **Adafruit_Seesaw** library folder your **arduinosketchfolder/libraries/** folder.
You may need to create the **libraries** subfolder if its your first library. Restart the IDE.

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

Load Test Example

Open up **File->Examples->Adafruit_Seesaw->digital->blink** and upload to your Arduino wired up to the seesaw breakout. If everything is wired up correctly, the led should blink on and off repeatedly.





Documentation

see here (https://adafru.it/BrT) for documentation of the seesaw python API.

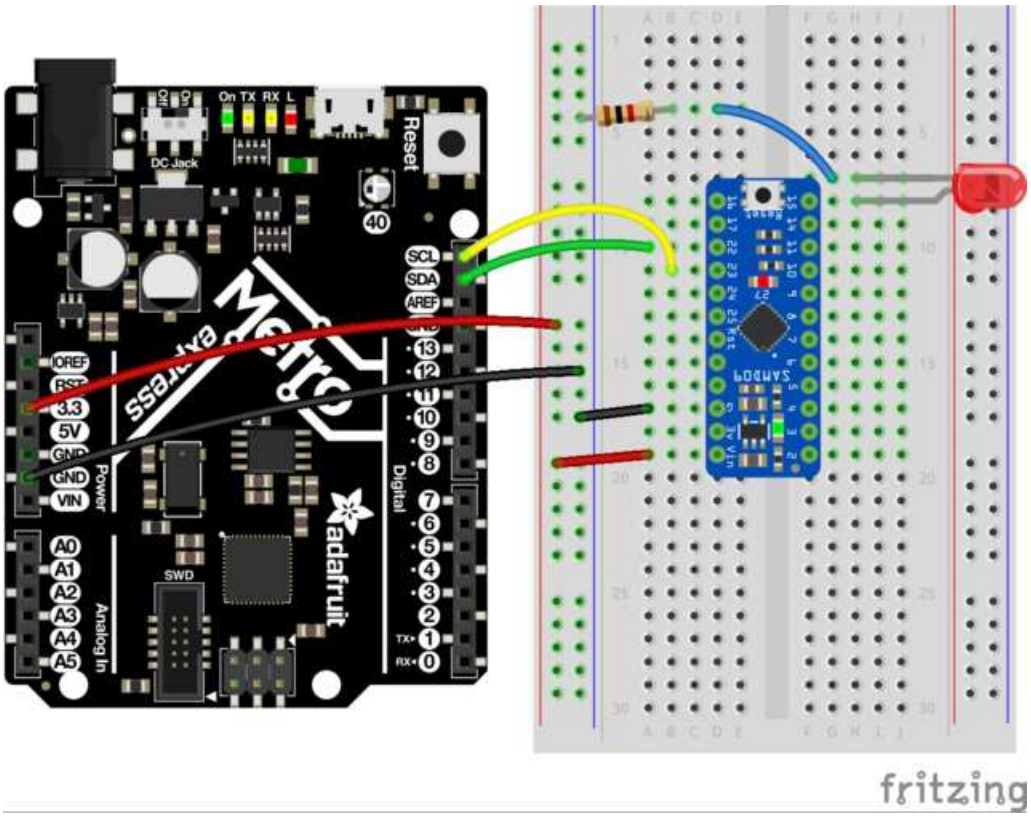
CircuitPython Wiring & Test

You can easily wire this breakout to a microcontroller running CircuitPython. We will be using a Metro M0 Express.

I2C Wiring

- Connect **Vin** to the power supply, 3-5V is fine.
- Connect **GND** to common power/data ground
- Connect the **SCL** pin (**23**) to the I2C clock **SCL** pin on your CircuitPython board, usually marked SCL. On a Gemma M0 this would be **Pad #2/ A1**
- Connect the **SDA** pin (**22**) to the I2C data **SDA** pin on your CircuitPython board, usually marked SDA. On an Gemma M0 this would be **Pad #0/A2**
- Connect the positive (long lead) of an LED to pin **15** on the samd09 breakout and the other lead to **GND** through a **1k ohm resistor.**

The seesaw uses I2C address **0x49** by default. You can change this by grounding the **AD0/16** and/or **AD1/15** pins, but we recommend not doing that until you have it working



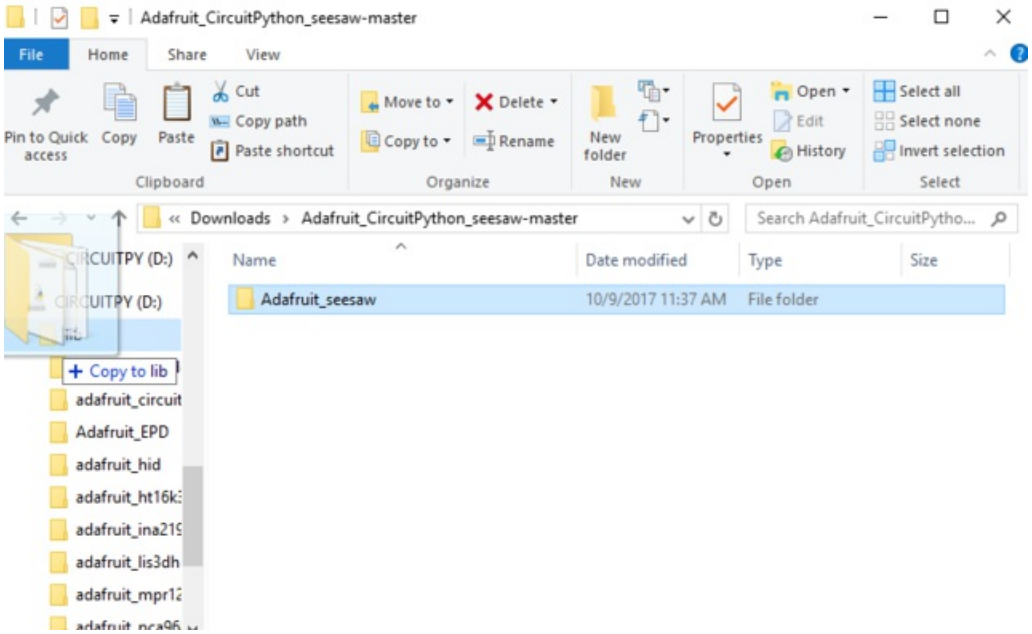
Download Adafruit_CircuitPython_Seesaw library

To begin using the seesaw, you will need to download Adafruit_CircuitPython_Seesaw 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

https://adafru.it/A0u

https://adafru.it/A0u

Extract the zipped folder and rename the **folder it contains** to **Adafruit_seesaw**. drag the **Adafruit_seesaw** folder to the **lib** folder that appears on the **CIRCUITPY** drive. You'll also need the **adafruit_busdevice** driver.

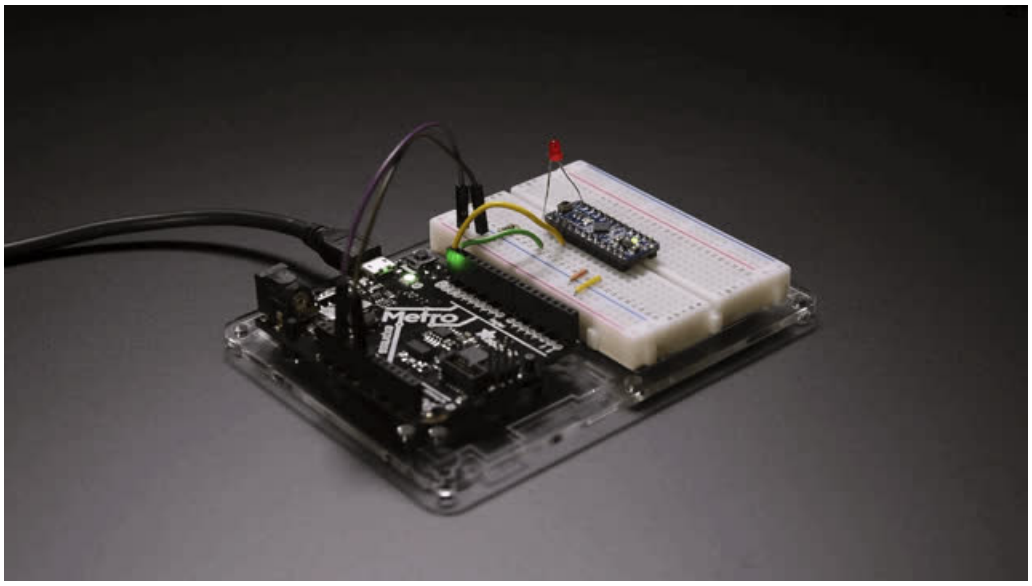


Our CircuitPython library may change APIs so consider this beta!

Open the **code.py** file on the **CIRCUITPY** drive and copy and paste the following code:

```
from board import *
import busio
import adafruit_seesaw
import time

myI2C = busio.I2C(SCL, SDA)

ss = adafruit_seesaw.Seesaw(myI2C)

ss.pin_mode(15, ss.OUTPUT);


while True:
 ss.digital_write(15, True)   # turn the LED on (True is the voltage level)
 time.sleep(1)                # wait for a second
 ss.digital_write(15, False)  # turn the LED off by making the voltage LOW
 time.sleep(1)
```

The LED attached to pin 15 should blink on and off repeatedly.



Python Docs

Python Docs (https://adafru.it/C5y)

Raspberry Pi Wiring & Test

The Raspberry Pi also has an I2C interface that can be used to communicate with this seesaw

Install Python Software

Once your Pi is all set up, and you have internet access set up, lets install the software we will need. First make sure your Pi package manager is up to date

```
sudo apt-get update
```

Next, we will install the Raspberry Pi library and Adafruit_GPIO which is our hardware interfacing layer

```
sudo apt-get install -y build-essential python-pip python-dev python-smbus git
git clone https://github.com/adafruit/Adafruit_Python_GPIO.git
cd Adafruit_Python_GPIO
sudo python setup.py install
```

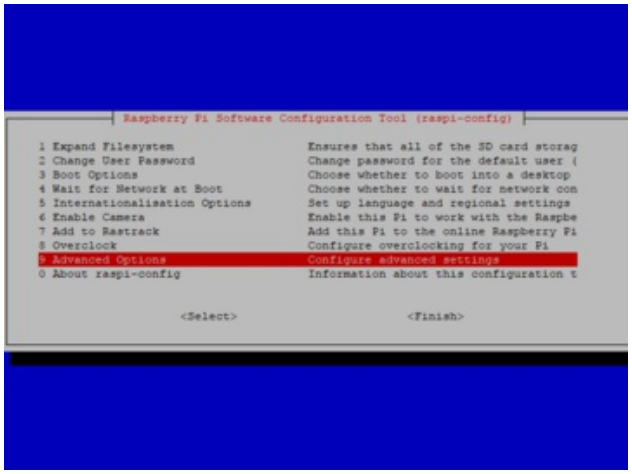
Next install the adafruit seesaw python library.

```
sudo pip install Adafruit-seesaw
```

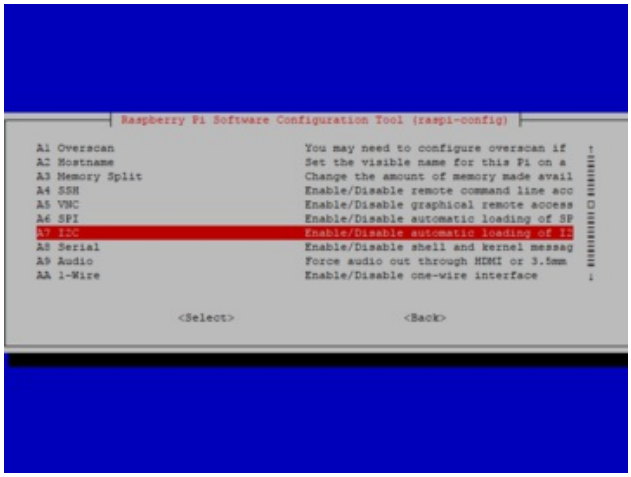
Enable I2C

We need to enable the I2C bus so we can communicate with the seesaw.

```
sudo raspi-config
```



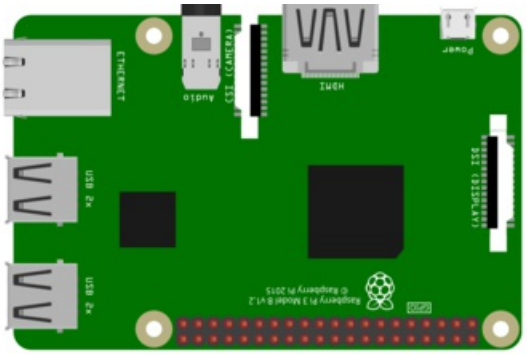
select Advanced options, enable I2C, and then finish.

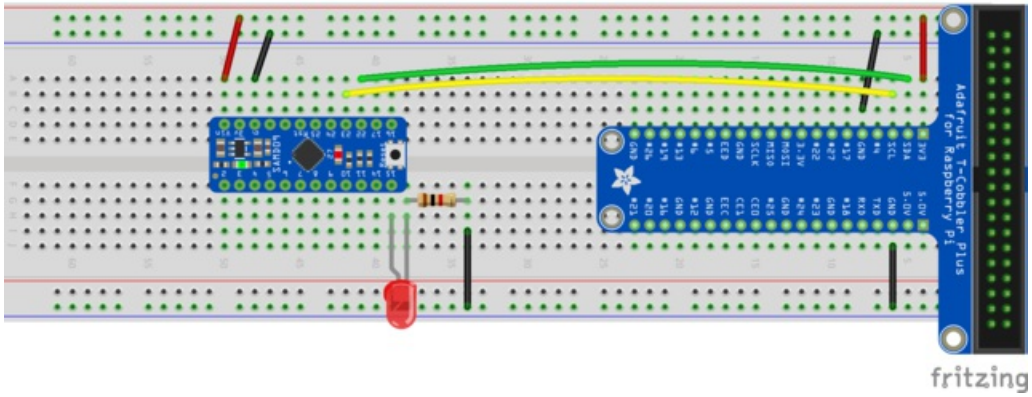


With the Pi powered off, we can wire up the sensor to the Pi Cobbler like this:

- Connect **Vin** to the 3V or 5V power supply (either is fine)
- Connect **GND** to the ground pin on the Cobbler
- Connect **SDA** (**22**) to **SDA** on the Cobbler
- Connect **SCL** (**23**) to **SCL** on the Cobbler
- Connect the positive (long lead) of an LED to pin **15** on the samd09 breakout and the other lead to **GND** through a **1k ohm resistor.**

You can also use direct wires, we happen to have a Cobbler ready. remember you can plug the cobbler into the bottom of the PiTFT to get access to all the pins!

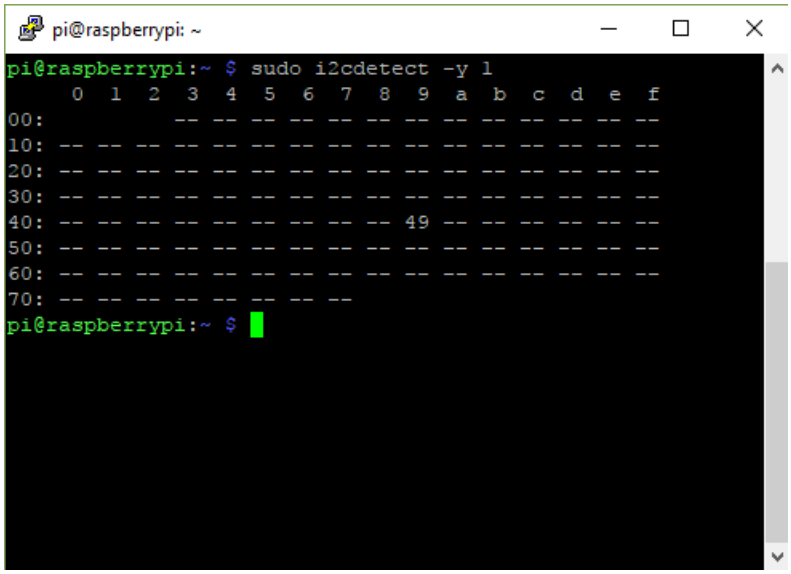




Now you should be able to verify that the sensor is wired up correctly by asking the Pi to detect what addresses it can see on the I2C bus:

```
sudo i2cdetect -y 1
```

It should show up under it's default address (**0x49**). If you don't see 49, check your wiring, did you install I2C support, etc?

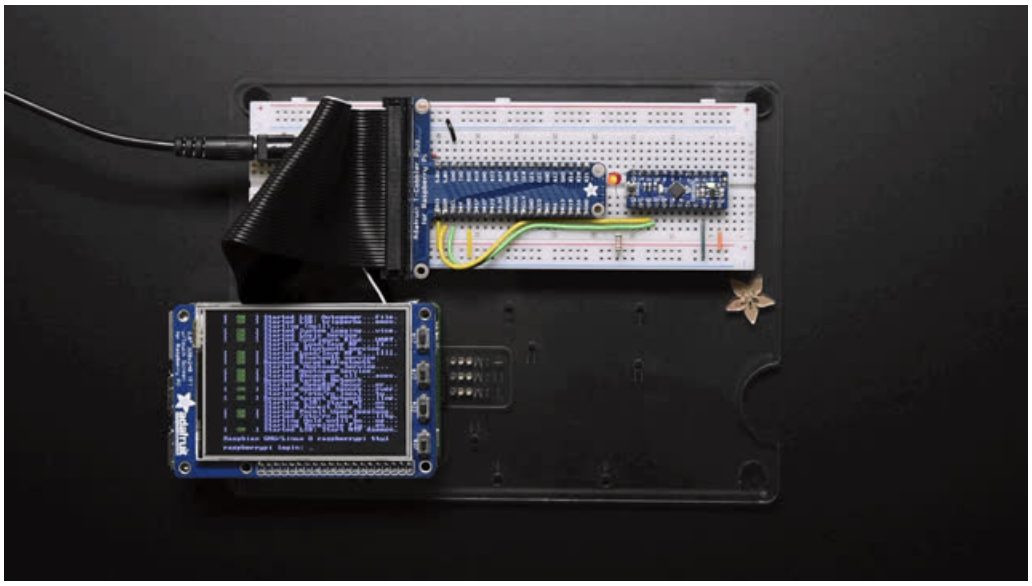


Run example code

At long last, we are finally ready to run our example code

```
cd ~/
git clone https://github.com/adafruit/Adafruit_Python_seesaw.git
cd Adafruit_Python_seesaw/examples
sudo python blink.py
```

If everything is set up correctly, the LED attached to pin 15 on the SAMD09 breakout should blink on and off repeatedly. Press CTRL + C to stop the program running once you are satisfied with the blinking.



Documentation

See here (https://adafru.it/BrT) for documentation of the seesaw python API.

Using the Seesaw Platform

The sections under this heading contain more detailed information about how the Seesaw platform works. **If you are using our Arduino, CircuitPython, or Python API you can skip these sections.** These sections are intended for people who either want to understand and modify seesaw, or who want to make their own API for a platform that is no officially supported by Adafruit such as C/C++ on Raspberry Pi.

Reading and Writing Data

The SeeSaw operates as an I2C slave device using standard I2C protocol. It uses the **SDA** and **SCL** pins to communicate with the host system.

We do not use clock stretching. The I2C is 3.3V logic level (but often times a 5V microcontroller will work fine with 3.3V logic levels and it is an open-drain protocol). Only 7-bit addressing is supported.

I2C pullup resistors are included in our SeeSaw boards but if you are DIY'ing, be sure to add your own! 2.2K - 10K is a good range.

Setting the Device Address

Standard 7-bit addressing is used. The seesaw's default I2C address is initially configured in the compiled firmware (e.g for the SAM09 SeeSaw breakout we use **0x49**) but other boards will have a different base address.

This address can be modified using the address select pins. If address select pin 0 (**PA16**) is tied to ground on boot, the I2C address is incremented by 1. If address select pin 1 (**PA17**) is pulled low, the I2C address is incremented by 2. If both address select pins are pulled low, the I2C address is incremented by 3. Thus you can, with the same hardware, have up to 4 devices

The I2C address can also be modified by writing a new address to EEPROM. See the EEPROM section for more information.

I2C Transactions

We recommend using 100KHz I2C, but speeds of up to 400KHz are supported. You may want to decrease the SDA/SCL pullups to 2.2K from 10K in that case.

Writing Data

A seesaw write command consists of the standard I2C write header (with the R/W bit set to 1), followed by 2 **register bytes** followed by zero or more **data bytes**.

The first register byte is the **module base register address**. Each module (GPIO, ADC, DAC, etc.) has it's own unique 8 bit base identifier.

The second register byte is the **module function register address**. This byte specifies the desired register within the module to be written.

Thus we have up to 254 modules available (0x00 is reserved) and 255 functions per module - plenty to allow all sorts of different capabilities!

In code, this may look like this (using the Arduino wire I2C object):

```
void Adafruit_seesaw::write(uint8_t moduleBase, uint8_t moduleFunction, uint8_t *buf, uint8_t num)
{
 Wire.beginTransmission((uint8_t)_i2caddr);
 Wire.write((uint8_t)moduleBase); //module base register address
 Wire.write((uint8_t)moduleFunction); //module function register address
 Wire.write((uint8_t *)buf, num); //data bytes
 Wire.endTransmission();
}
```

The Arduino UNO Wire library implementation has a limit of 32 bytes per transaction so be aware you may not be able to read/write more than that amount. We have designed the library to work within those constraints

Reading Data

A register read is accomplished by first sending the standard I2C write header, followed by the two **register bytes** corresponding to the data to be read. Allow a short delay, and then send a standard I2C read header (with the R/W bit set to 0) to read the data.

The length of the required delay depends on the data that is to be read. These delays are discussed in the sections specific to each module.

In code, this may look like this (using the Arduino wire I2C object):

```
void Adafruit_seesaw::read(uint8_t moduleBase, uint8_t moduleFunction, uint8_t *buf, uint8_t num, uint16_
{
  Wire.beginTransmission((uint8_t)_i2caddr);
  Wire.write((uint8_t)moduleBase); //module base register address
  Wire.write((uint8_t)moduleFunction); //module function register address
  Wire.endTransmission();

  delayMicroseconds(delay);

  Wire.requestFrom((uint8_t)_i2caddr, num);

  for(int i=0; i<num; i++){
    buf[i] = Wire.read();
  }
}
```

GPIO

The GPIO module provides every day input and outputs. You'll get 3.3V logic GPIO pins that can act as outputs or inputs. With pullups or pulldowns. When inputs, you can also create pin-change interrupts that are routed the the IRQ pin.

The module base register address for the GPIO module is **0x01**.

Function Registers

Register Address	Function Name	Register Size	Notes
0x02	DIRSET	32 bits	Write Only
0x03	DIRCLR	32 bits	Write Only
0x04	GPIO	32 bits	Read/Write
0x05	SET	32 bits	Write Only
0x06	CLR	32 bits	Write Only
0x07	TOGGLE	32 bits	Write Only
0x08	INTENSET	32 bits	Write Only
0x09	INTENCLR	32 bits	Write Only
0x0A	INTFLAG	32 bits	Read Only
0x0B	PULLENSET	32 bits	Write Only
0x0C	PULLENCLR	32 bits	Write Only

Writes of GPIO function registers should contain **4 data bytes** (32 bits) following the initial register data bytes. Each bit in these registers represents a GPIO pin on **PORTA** of the seesaw device.

If the corresponding pin does not exist on the SeeSaw device, then reading or writing the bit has no effect.

We decided to go with this method to make GPIO toggling fast (rather than having one i2c transaction per individual pin control) but the host processor will need to do a little work to keep the pins identified.

GPIO register setup:

Bit 31	Bit 30	Bit 29	Bit 28	Bit 27	. . .	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PA31	PA30	PA29	PA28	PA27	. . .	PA04	PA03	PA02	PA01	PA00

DIRSET (0x02, 32 bits, Write Only)

Writing a 1 to any bit in this register sets the direction of the corresponding pin to OUTPUT.

Writing zeros to this register has no effect.

DIRCLR (0x03, 32 bits, Write Only)

Writing a 1 to any bit in this register sets the direction of the corresponding pin to INPUT.

Writing zeros to this register has no effect.

GPIO (0x04, 32 bits, Read/Write)

When this register is written, all bits that are set to 0 will have their corresponding pins set LOW.

All bits that are set to 1 will have their corresponding pins set HIGH.

Reading this register reads all pins on PORTA of the seesaw device.

SET (0x05, 32 bits, Write Only)

Writing a 1 to any bit in this register writes the corresponding pin HIGH.

Writing zeros to this register has no effect.

CLR (0x06, 32 bits, Write Only)

Writing a 1 to any bit in this register writes the corresponding pin LOW.

Writing zeros to this register has no effect.

TOGGLE (0x07, 32 bits, Write Only)

Writing a 1 to any bit in this register toggles the corresponding pin.

Writing zeros to this register has no effect.

INTENSET (0x08, 32 bits, Write Only)

Writing a 1 to any bit in this register enables the interrupt on the corresponding pin. When the value on this pin changes, the corresponding bit will be set in the **INTFLAG** register.

Writing zeros to this register has no effect.

INTENCLR (0x09, 32 bits, Write Only)

Writing a 1 to any bit in this register disables the interrupt on the corresponding pin.

Writing zeros to this register has no effect.

INTFLAG (0x0A, 32 bits, Read Only)

This register hold the status of all GPIO interrupts. When an interrupt fires, the corresponding bit in this register gets set. Reading this register clears all interrupts.

Writing to this register has no effect.

PULLENSET (0x0B, 32 bits, Write Only)