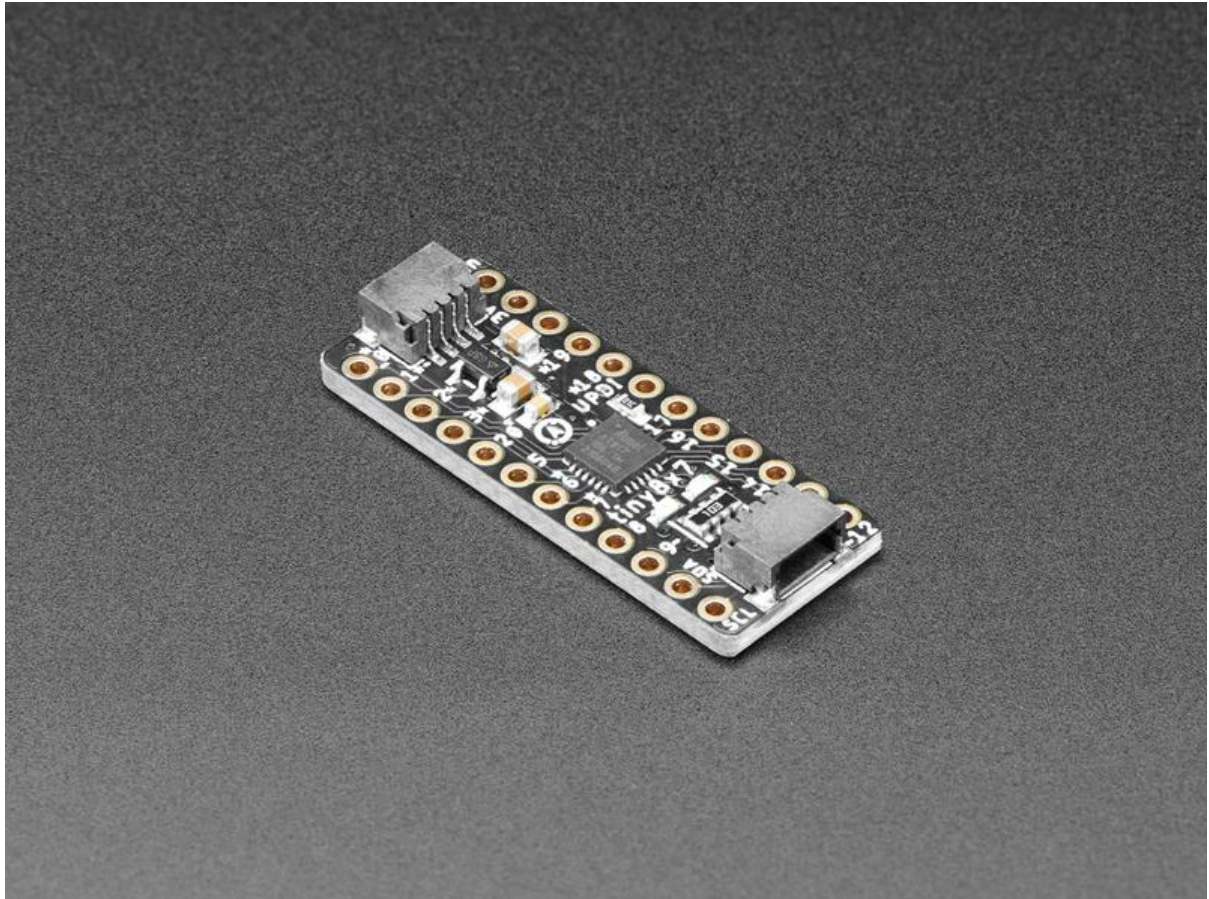




Adafruit ATtiny817 Breakout with seesaw

Created by Kattni Rembor



<https://learn.adafruit.com/adafruit-attiny817-seesaw>

Last updated on 2022-02-14 12:35:03 PM EST

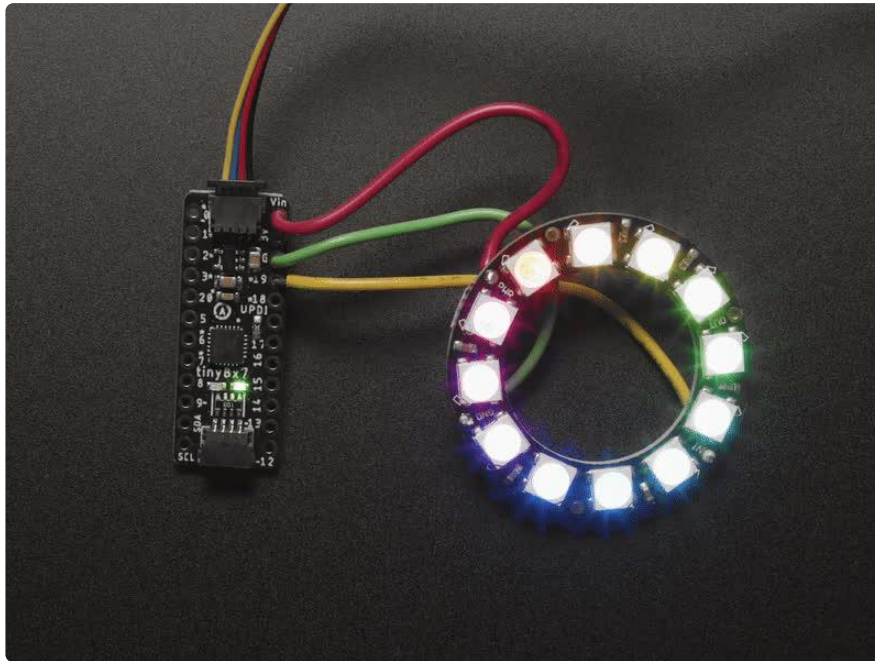
Table of Contents

Overview	5
Pinouts	9
• Power Pins:	9
• I2C Pins / STEMMA QT:	10
• seesaw GPIO Pins:	10
• seesaw Interrupt Pins:	10
• seesaw I2C Address Pins:	11
• seesaw NeoPixel-Capable Pins:	11
• ADC Pins:	11
• PWM Pins:	11
• Programming Pins:	11
Python & CircuitPython	11
• CircuitPython Microcontroller Wiring	12
• Python Computer Wiring	13
• Python Installation of seesaw Library	13
• CircuitPython Usage	13
• Python Usage	14
• Example Code:	14
Analog In	15
• Analog Pins	16
• Example Code	16
Digital Input	17
• Digital Pins	17
• Wiring	17
• Example Code	17
NeoPixel	19
• NeoPixel Pins	19
• Wiring	19
• Example Code	20
PWMOut	21
• PWM Pins	22
• Wiring	22
• Example Code	22
EEPROM	24
• Example Code	24
Python Docs	25
Arduino	25
• Wiring	25
• Library Installation	26
• Load Example	27

Analog In	28
• Analog Pins	28
• Example Code	29
PWM	29
• PWM Pins	30
• Wiring	30
• Example Code	30
NeoPixel	32
• NeoPixel Pins	32
• Wiring	32
• Example Code	33
EEPROM	36
• Example Code	36
Using the Seesaw Platform	37
Reading and Writing Data	37
• Setting the Device Address	38
• I2C Transactions	38
• Writing Data	38
• Reading Data	39
GPIO	39
• Function Registers	40
• GPIO register setup:	41
• DIRSET (0x02, 32 bits, Write Only)	41
• DIRCLR (0x03, 32 bits, Write Only)	41
• GPIO (0x04, 32 bits, Read/Write)	42
• SET (0x05, 32 bits, Write Only)	42
• CLR (0x06, 32 bits, Write Only)	42
• TOGGLE (0x07, 32 bits, Write Only)	42
• INTENSET (0x08, 32 bits, Write Only)	42
• INTENCLR (0x09, 32 bits, Write Only)	43
• INTFLAG (0x0A, 32 bits, Read Only)	43
• PULLENSET (0x0B, 32 bits, Write Only)	43
• PULLENCLR (0x0C, 32 bits, Write Only)	43
Analog to Digital Converter	43
• Function Registers	44
• STATUS (0x00, 8bits, Read Only)	45
•	45
• INTENSET (0x02, 8bits, Write Only)	45
• INTENCLR (0x03, 8bits, Write Only)	46
• WINMODE (0x04, 8bits, Write Only)	46
• WINTHRESH (0x05, 32bits, Write Only)	46
• CHANNEL_0 (0x07, 16bits, Read Only)	46
• CHANNEL_1 (0x08, 16bits, Read Only)	46
• CHANNEL_2 (0x09, 16bits, Read Only)	46
• CHANNEL_3 (0x0A, 16bits, Read Only)	47
Interrupts	47

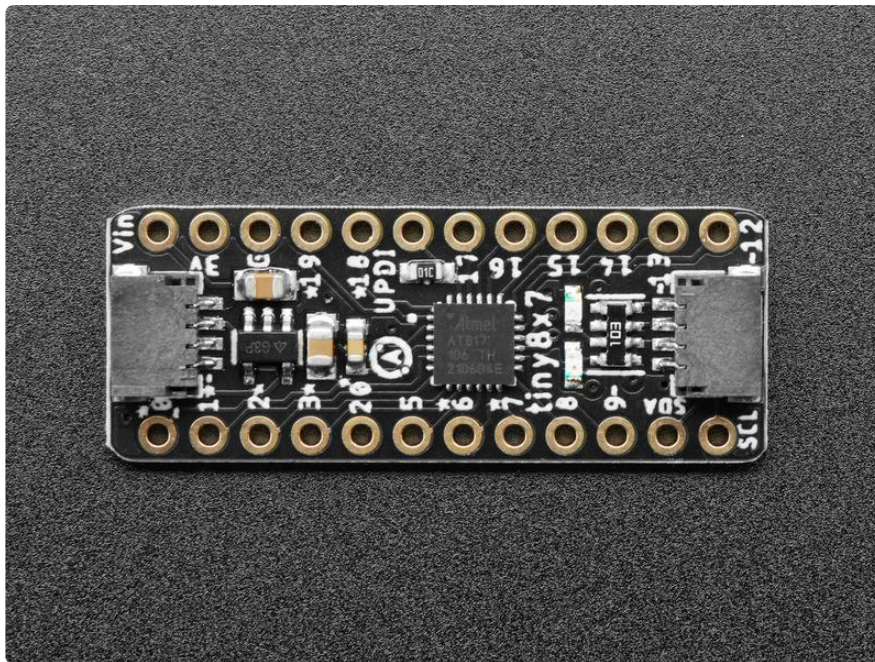
EEPROM	47
• Function Registers	48
• SAMD09	48
• ATtiny817	48
• Function Registers	48
• PIN (0x01, 8bits, Write Only)	49
• SPEED (0x02, 8bits, Write Only)	49
• BUF_LENGTH (0x03, 16bits, Write Only)	49
• BUF (0x04, 32 bytes, Write Only)	49
• SHOW (0x05, no args, Write Only)	50
PWM	50
• Function Registers	50
• PWM_VAL (0x01, 16bits, Write Only)	50
Downloads	51
• Files	51
• Schematic and Fab Print	52

Overview

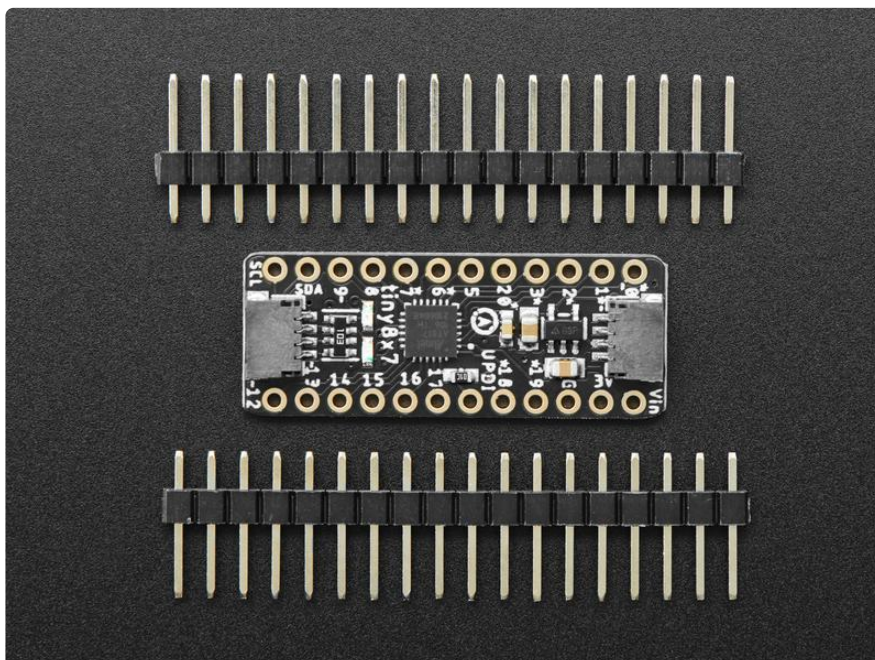


This is the Adafruit ATtiny817 breakout with seesaw! This breakout board is a "three in one" product:

1. The ATtiny817 is part of the 'next gen' of AVR microcontrollers, and now we have a cute development/breakout board for it, with just enough hardware to get the chip up and running.
2. It's also an Adafruit seesaw board. Adafruit seesaw is a near-universal converter framework which allows you to 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.
3. Finally, with STEMMA QT connectors on it, you could use it as either an I2C controller or peripheral with plug-and-play support.



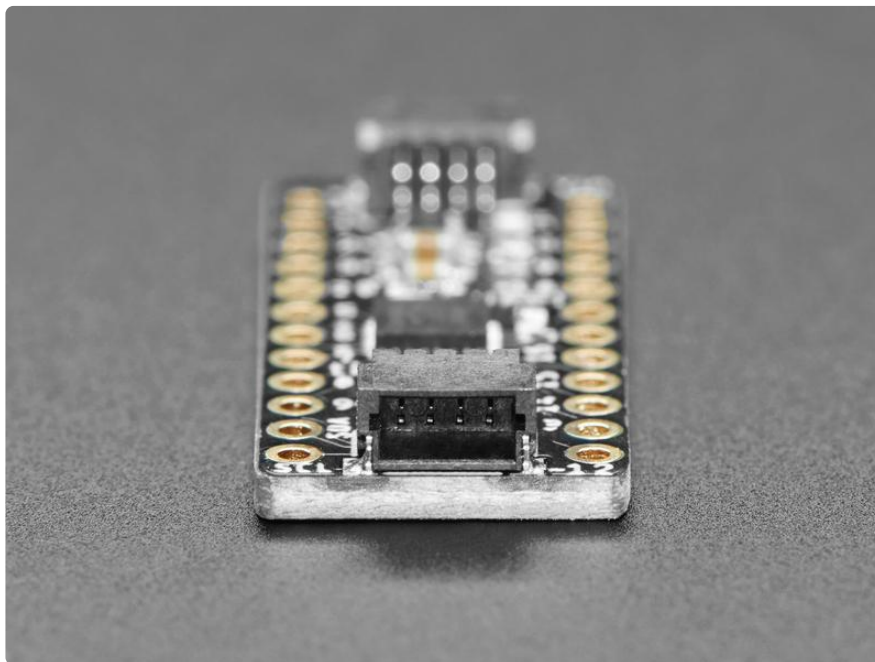
We primarily designed this board for our own use: it's a mini dev board that lets us design with the ATtiny817 [just like we did for the ATSAM09 \(https://adafruit.it/VdK\)](https://adafruit.it/VdK). With the 2021 silicon shortage, we're adapting some of our SAMD09 designs to the ATTiny8xx series and wanted a quick minimal board to test code on.



Each breakout comes with the assembled and tested board, as well as some header strips. Each PCB is fairly minimal and contains:

- ATtiny817 8-bit microcontroller
 - 8KB flash, 512 bytes of RAM, 128 bytes of EEPROM

- Internal oscillator can run up to 20MHz
- Internal hardware multiplier
- Can run from 2V to 5V power/logic (check the datasheet for max speed at desired power)
- 3.3V regulator - by default we run at the Vin voltage, which can be 5V, but there's a solder jumper on the bottom if you'd like to select 3V logic.
- Green power LED
- Red indicator LED
- Two STEMMA QT I2C connectors with 10K pullup resistors, connected to pins 10 and 11. The default I2C address is 0x49.

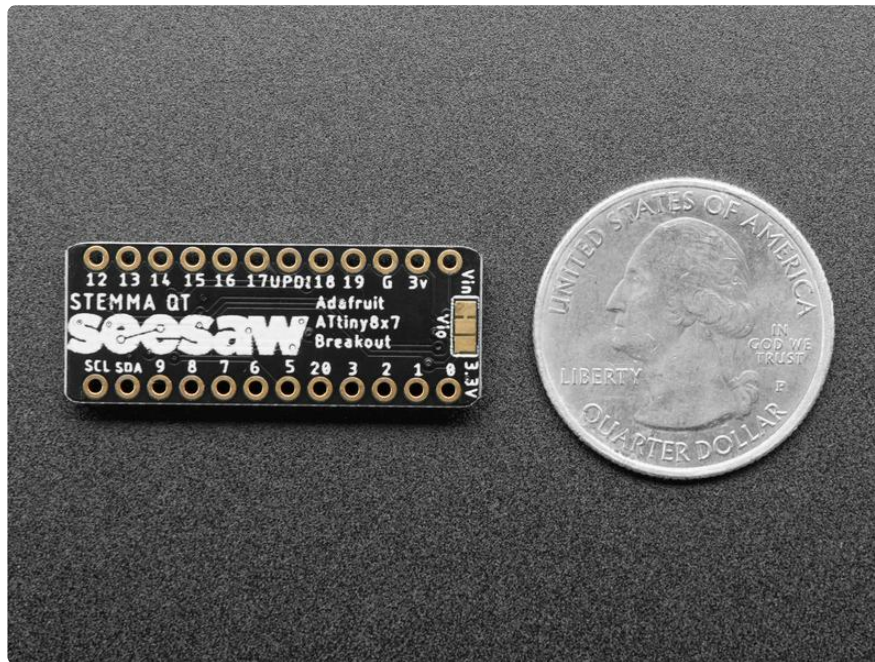


[This board comes pre-programmed with seesaw peripheral code \(https://adafru.it/VdL\)](https://adafru.it/VdL) that will let it act as an "I2C to something" converter, basically a little I2C-controlled friend to do all the timing-sensitive things many microcontrollers and microcomputers are not good at.

For example, using this breakout with the pre-burned seesaw firmware gives you

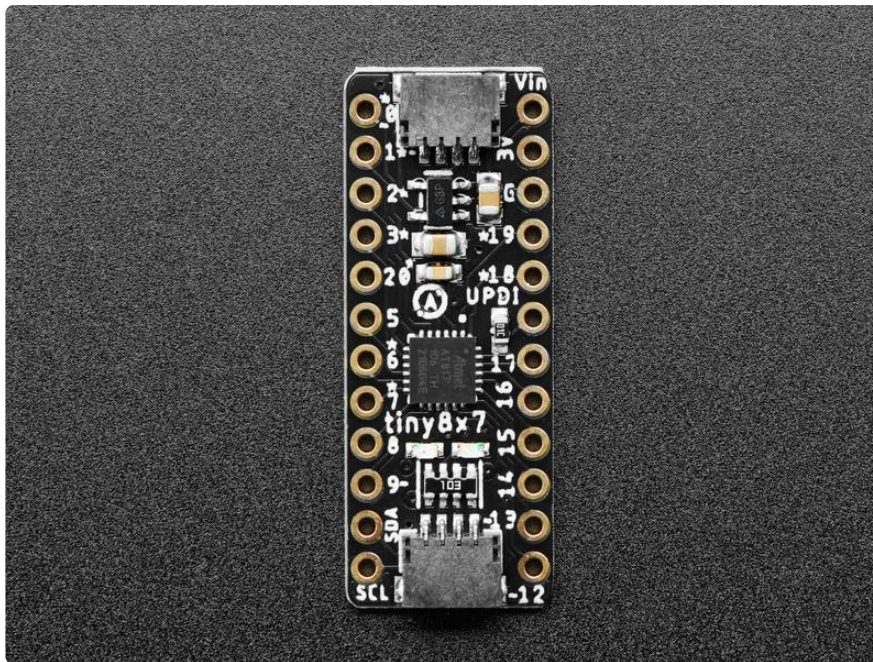
- 14 x GPIO with selectable pullup resistors
- 9 x 10-bit ADC inputs - pins 0, 1, 2, 3, 6, 7, 18, 19, 20
- 5 x 8-bit PWM outputs - pins 0, 1, 9, 12, 13
- 1 x NeoPixel output (up to 60 pixels)
- 1 x EEPROM with 127 byte of NVM memory (handy for storing small access tokens or MAC addresses) - last byte of EEPROM is used for I2C address selection
- 1 x Interrupt output that can be triggered by any of the accessories - pin 15

- 2 x I2C address selection pins - pins 16 and 17
- 1 x Activity LED on pin 5, tied active low

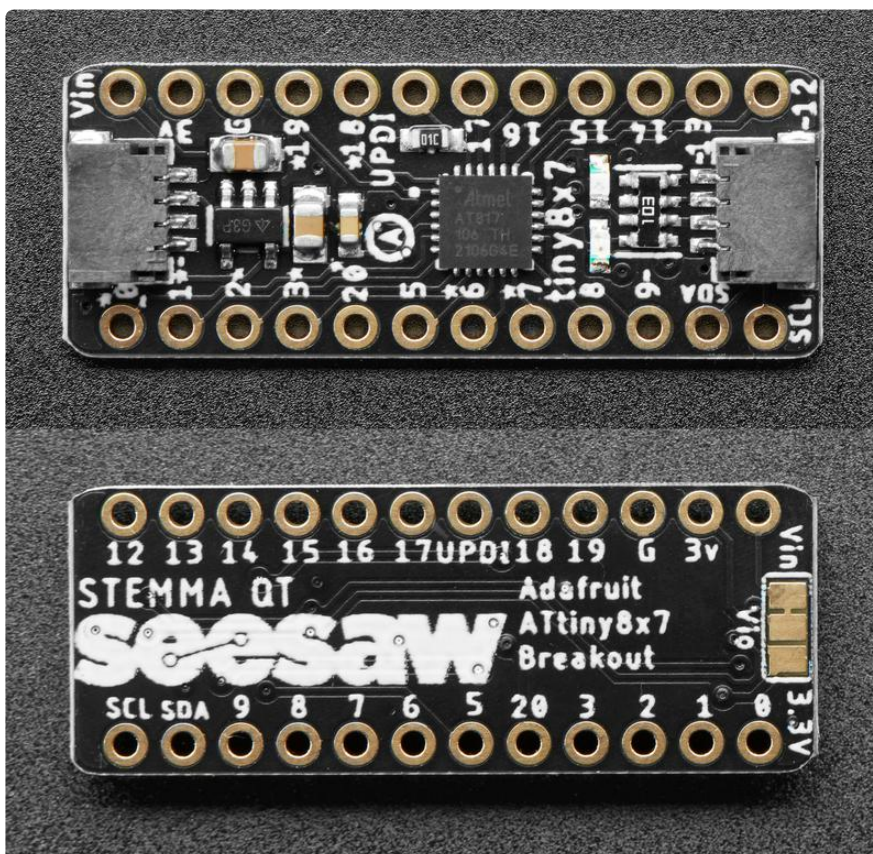


Of course you can configure or reprogram the chip to however you want to use it - [we like using SpenceKonde's megaTinyCore which brings Arduino peripheral support \(https://adafru.it/VdM\)](https://adafru.it/VdM) to this series of chips. To program the chip [you will need a 'UPDI' programmer \(https://adafru.it/VdM\)](https://adafru.it/VdM), which you can make with a [USB-to-Serial cable \(https://adafru.it/dDd\)](https://adafru.it/dDd) and [a single 4.7K or 10K resistor \(https://adafru.it/BHo\)](https://adafru.it/BHo).

Please note: The boards do not come with a bootloader. If you want to do development on seesaw (e.g. changing the configuration) you need a separate UPDI programming setup! [The firmware we put on is available as this example sketch \(https://adafru.it/VdN\)](https://adafru.it/VdN), compiled using the megaTinyCore. We don't provide any support for custom builds of seesaw - we think this is cool and useful for the Maker community!



Pinouts



Power Pins:

- Vin - this is the power pin. The ATtiny8x7 operates between 1.8V and 5.5V. However, we still included a 3.3V voltage regulator. You can power from 3.3V to

5V. It defaults to Vin voltage, which can be between 3.3V and 5V, or cut the jumper on the back and bridge the other pad to select 3.3V logic.

- 3Vo - This is the 3.3V output from the voltage regulator. You can grab up to 100mA from this if you like.
- GND - This is common ground for power and logic.

If you swap the jumper on the back to select 3.3V logic, the ATtiny should be kept at the (pre-programmed default) 10MHz and not set faster

I2C Pins / STEMMA QT:

The default I2C address is 0x49.

- SCL - This is the I2C clock pin. Connect to your microcontrollers I2C clock line. There is a 10K pullup on this pin to Vin.
- SDA - This is the I2C data pin. Connect to your microcontrollers I2C data line. There is a 10K pullup on this pin to Vin.
- [STEMMA QT \(https://adafru.it/Ft4\)](https://adafru.it/Ft4) - These I2C connectors, located on each end of the board, have 10K pullup resistors. They allow you to connect to development boards with STEMMA QT connectors, or to other things, with [various associated accessories \(https://adafru.it/Ft6\)](#).

seesaw GPIO Pins:

- 0, 1, 2, 3, 20, 5, 6, 7, 8, 9, 12, 13, 14, 18, 19 - These are the 15 GPIO pins available.

Pins 15, 16 and 17 can NOT be used as GPIO pins with the seesaw firmware!

seesaw Interrupt Pins:

- 15 - This pin can be set to pull low by the breakout to signal to your host microcontroller that an interrupt has occurred.

seesaw I2C Address Pins:

The default I2C address is 0x49.

- 16 / AD0 - This is the ADDR0 pin. Connect this to ground to increment the device's I2C address by 1.
- 17 / AD1 - this is the ADDR1 pin. Connect this to ground to increment the device's I2C address by 2.

seesaw NeoPixel-Capable Pins:

- 0, 1, 2, 3, 20, 5, 6, 7, 8, 9, 12, 13, 14, 18, 19 - Any GPIO pin can be a NeoPixel pin, however, you can only use one pin at a time. There is only one NeoPixel buffer, and therefore only one NeoPixel output. It can drive up to 60 pixels.

ADC Pins:

- 0, 1, 2, 3, 6, 7, 18, 19, 20 - There are nine 10-bit ADC pins.

PWM Pins:

- 0, 1, 9, 12, 13 - There are five 8-bit PWM output pins.

Programming Pins:

- UPDI - This is the single-pin Unified Program and Debug Interface. This pin is for external programming or on-chip-debugging. It uses the reset pin on the ATtiny.

There is no reset on this breakout! It is used for the UPDI programming pin, and cannot be used to reset the board!

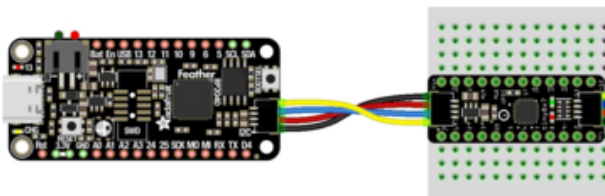
Python & CircuitPython

The [Adafruit CircuitPython seesaw library \(https://adafru.it/BrW\)](https://adafru.it/BrW) makes it easy to do all kinds of things with your ATtiny817 Breakout. This section includes a quick way to verify your breakout is hooked up properly and functioning.

You can use this sensor 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) (<https://adafru.it/BSN>).

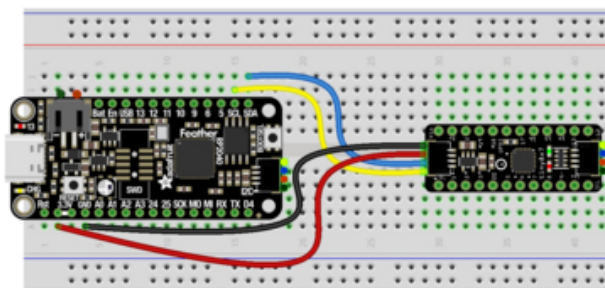
CircuitPython Microcontroller Wiring

First wire up a ATtiny817 breakout to your microcontroller board exactly as shown below. Here's an example of wiring a Feather RP2040 to the breakout using I2C via the STEMMA QT connector.



- Use a STEMMA QT cable to connect the STEMMA QT connector on the Feather to the STEMMA QT connector on the breakout.

Here's an example connecting the breakout STEMMA QT connector to a solderless breadboard.

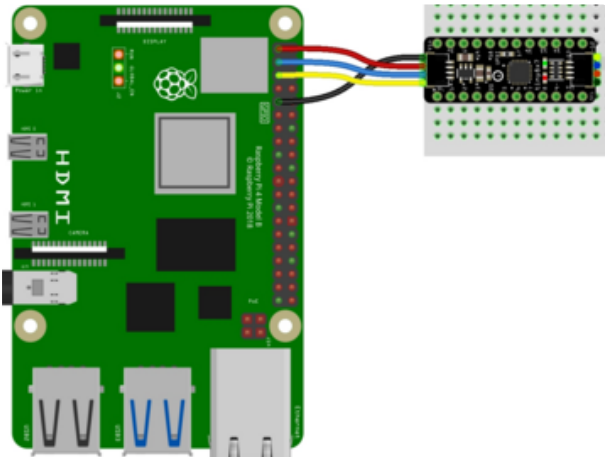


- Plug a [STEMMA QT to male jumper wire cable](https://adafru.it/FA-) (<https://adafru.it/FA->) into the STEMMA QT connector on the breakout.
- Feather 3V to breakout VIN (red wire)
- Feather GND to breakout GND (black wire)
- Feather SCL to breakout SCL (yellow wire)
- Feather SDA to breakout SDA (blue wire)

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) (<https://adafru.it/BSN>).

Here's the Raspberry Pi wired with I2C.



- Plug a [STEMMA QT to male jumper wire cable](https://adafru.it/FA-) (<https://adafru.it/FA->) into the STEMMA QT connector on the breakout.
- Pi 3V3 to breakout VIN (red wire)
- Pi GND to breakout GND (black wire)
- Pi SCL to breakout SCL (yellow wire)
- Pi SDA to breakout SDA (blue wire)

Python Installation of seesaw Library

You'll need to install the Adafruit_Blinka 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) (<https://adafru.it/BSN>)!

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

- `pip3 install adafruit-circuitpython-seesaw`

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!

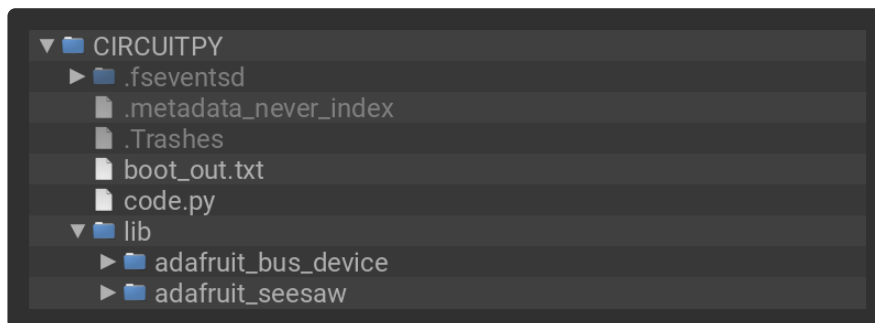
CircuitPython Usage

To use with CircuitPython, you need to first install the seesaw library into the lib folder on your CIRCUITPY drive. Then you need to update code.py with the example script.

Thankfully, you can do this in one go. In the example below, click the Download Project Bundle button below to download the necessary libraries and the code.py file in a zip file. Extract the contents of the zip file, and copy the entire lib folder and the code.py file to your CIRCUITPY drive.

Your CIRCUITPY/lib folder should contain the following folders:

- adafruit_bus_device/
- adafruit_seesaw/



Python Usage

Once you have the library `pip3` installed on your computer, copy or download the following example to your computer, and run the following, replacing code.py with whatever you named the file: `python3 code.py`

Example Code:

```
# SPDX-FileCopyrightText: 2021 Kattni Rembor for Adafruit Industries
# SPDX-License-Identifier: MIT
"""
Simple seesaw test for ATtiny8x7 breakout using built-in LED on pin 5.
"""
import time
import board
from adafruit_seesaw.seesaw import Seesaw

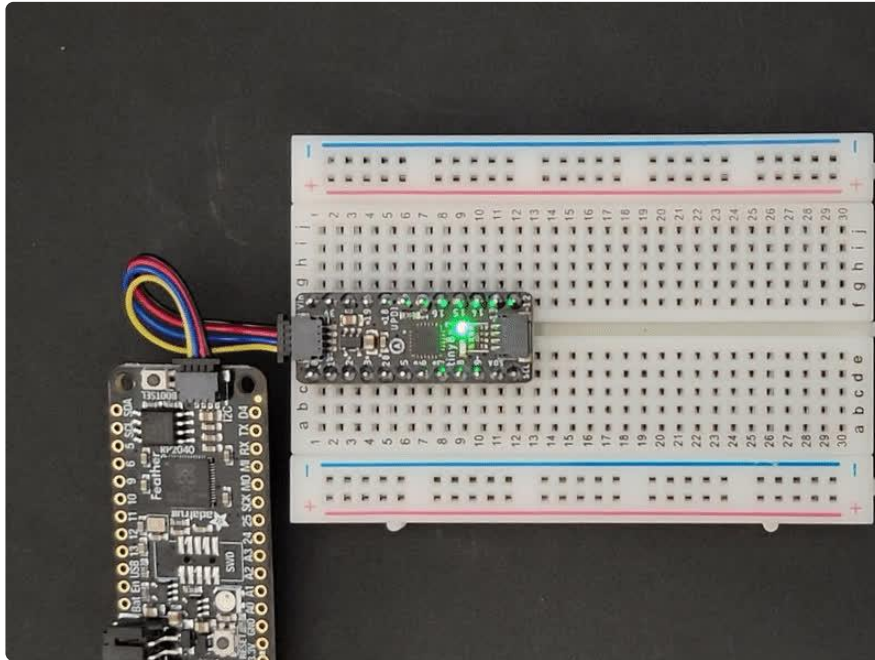
ss = Seesaw(board.I2C())

ss.pin_mode(5, ss.OUTPUT)

while True:
    ss.digital_write(5, False) # Turn the LED on (the built-in LED is active low!)
    time.sleep(1) # Wait for one second
    ss.digital_write(5, True) # Turn the LED off
    time.sleep(1) # Wait for one second
```

If running CircuitPython: Once everything is saved to the CIRCUITPY drive, the built-in LED will begin blinking!

If running Python: Once you run the example, the built-in LED will begin blinking!



In this example, you first import the required modules and library. Then you initialise the seesaw and provide it the I2C board object.

Next, you setup pin 5 as an output.

Inside the loop, you first set the LED to `False` to turn it on. `False` is the voltage level. You set it to `False` to turn it on because the LED is tied active low. Then, you wait for one second. Next, you turn off the LED by setting it to `True`, and then wait one more second before beginning the loop again.

That's all there is to blinking the built-in LED on the ATtiny817 breakout using CircuitPython and the seesaw library!

Analog In

The seesaw firmware that ships with the ATtiny817 breakout includes analog capabilities on specific pins. This simple example reads the analog value of a pin.

Follow the instructions on the [Python & CircuitPython page \(https://adafru.it/Vsf\)](https://adafru.it/Vsf) to get set up.

Analog Pins

The ATtiny817 breakout with seesaw firmware provides analog on the following pins:

- 0, 1, 2, 3, 6, 7, 18, 19, 20

Example Code

Update your code.py to the following.

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

# Simple seesaw test reading analog value
# on SAMD09, analog in can be pins 2, 3, or 4
# on Attiny8x7, analog in can be pins 0, 1, 2, 3, 6, 7, 18, 19, 20

import time
import board
from adafruit_seesaw.seesaw import Seesaw
from adafruit_seesaw.analoginput import AnalogInput

ss = Seesaw(board.I2C())

analogin_pin = 2
analog_in = AnalogInput(ss, analogin_pin)

while True:
    print(analog_in.value)
    time.sleep(0.1)
```

Now, connect to the serial console to see the values printed out.



First you import the necessary modules and libraries. Then, you instantiate the seesaw on I2C.

Next, you choose the analog pin from which you would like to read values, in this case pin 2, and then set it up as an `AnalogInput`.

Inside the loop, you read the analog pin value and print it to the serial console. Finally, you include a short delay to keep the values readable.

That's all there is to using CircuitPython seesaw analog with the ATtiny817 breakout!

Digital Input

The seesaw firmware that ships with the ATtiny817 breakout includes digital capabilities on specific pins. This example blinks the turns on the built-in LED when a button is pressed.

Follow the instructions on the [Python & CircuitPython page \(https://adafru.it/Vsf\)](https://adafru.it/Vsf) to get set up.

Digital Pins

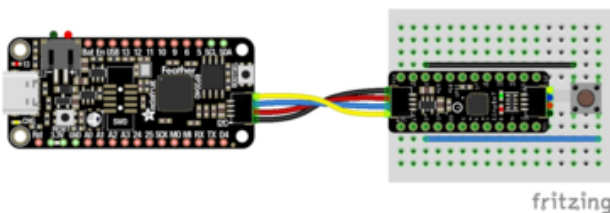
There are 15 pins on the ATtiny817 breakout that can be used with digitalio:

- 0, 1, 2, 3, 20, 5, 6, 7, 8, 9, 12, 13, 14, 18, 19

Pins 15, 16 and 17 can NOT be used as digital pins with the seesaw firmware!

Wiring

Connect a button up to your already wired up breakout as follows.



- Use a STEMMA QT cable to connect the STEMMA QT connector on the Feather to the STEMMA QT connector on the breakout.
- Connect one leg of button to breakout GND
- Connect opposite leg of button to breakout pin 2

Example Code

Update your code.py to the following.

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

# Simple seesaw test using an LED attached to Pin 5 and a button on pin 2
#
# See the seesaw Learn Guide for wiring details.
# For SAMD09:
# https://learn.adafruit.com/adafruit-seesaw-atsamd09-breakout?
# view=all#circuitpython-wiring-and-test
# For ATtiny8x7:
# https://learn.adafruit.com/adafruit-attiny817-seesaw/digital-input

import time
import board
import digitalio
from adafruit_seesaw.seesaw import Seesaw
from adafruit_seesaw.digitalio import DigitalIO

ss = Seesaw(board.I2C())

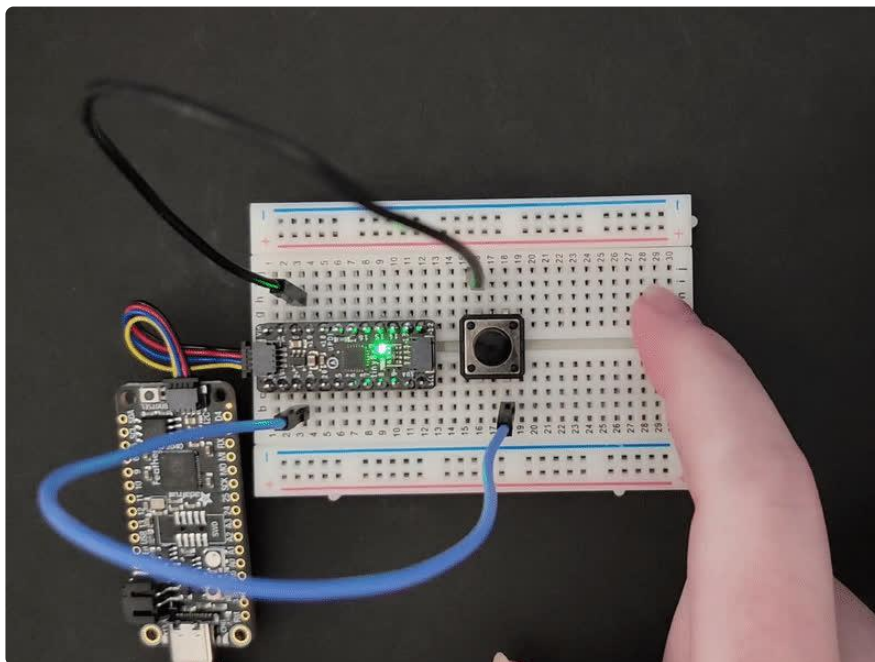
button_pin = 2
led_pin = 5

button = DigitalIO(ss, button_pin)
button.direction = digitalio.Direction.INPUT
button.pull = digitalio.Pull.UP

led = DigitalIO(ss, led_pin)
led.direction = digitalio.Direction.OUTPUT

while True:
    # simply set the LED to the same 'value' as the button pin
    led.value = button.value
    time.sleep(0.1)
```

Now, press the button to see the little red LED light up! Release the button to turn it off.



First, you import all the necessary modules and libraries. Then you instantiate the seesaw on I2C.

Next, you choose the button pin and LED pin. The button is on pin 2, and the built-in LED is on pin 5.

Then you create the button object, and set it up as an input with a pullup. Following that, you create the LED object and set up the pin as an output.

Finally, inside the loop, you simply set the LED state equal to the button state with a 0.1 second delay for debounce.

That's all there is to using CircuitPython seesaw digitalio with the ATtiny817 breakout!

NeoPixel

The seesaw firmware that ships with the ATtiny817 breakout includes the ability to power up to 60 NeoPixels on one pin. This example displays a rainbow across a NeoPixel ring.

Follow the instructions on the [Python & CircuitPython page \(https://adafru.it/Vsf\)](https://adafru.it/Vsf) to get set up.

NeoPixel Pins

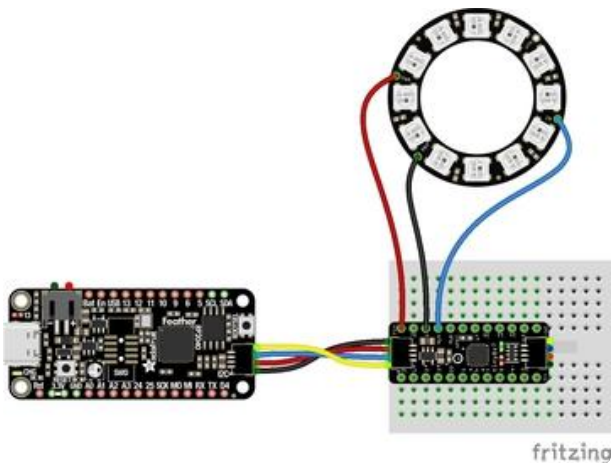
There are 15 pins on the ATtiny817 breakout that can be used for powering NeoPixels. However, you can only use one pin at a time! The available pins are:

- 0, 1, 2, 3, 20, 5, 6, 7, 8, 9, 12, 13, 14, 18, 19

Pins 15, 16 and 17 can NOT be used as NeoPixel pins with the seesaw firmware!

Wiring

As stated above, you can use many different pins for powering NeoPixels, but this example uses pin 19. Connect a NeoPixel ring (or strip) to the breakout as follows.



- Use a STEMMA QT cable to connect the STEMMA QT connector on the Feather to the STEMMA QT connector on the breakout.
- Connect NeoPixel ring GND to breakout GND
- Connect NeoPixel ring IN (data in) to breakout pin 19
- Connect NeoPixel ring power to breakout VIN

Example Code

Update your code.py to the following.

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

# Simple seesaw test writing NeoPixels
# Can use any valid GPIO pin, up to 60 pixels!
#
# See the seesaw Learn Guide for wiring details.
# For SAMD09:
# https://learn.adafruit.com/adafruit-seesaw-atsamd09-breakout?
# view=all#circuitpython-wiring-and-test
# For ATtiny8x7:
# https://learn.adafruit.com/adafruit-attiny817-seesaw/neoixel

import time
import board
from rainbowio import colorwheel
from adafruit_seesaw import seesaw, neopixel

ss = seesaw.Seesaw(board.I2C())

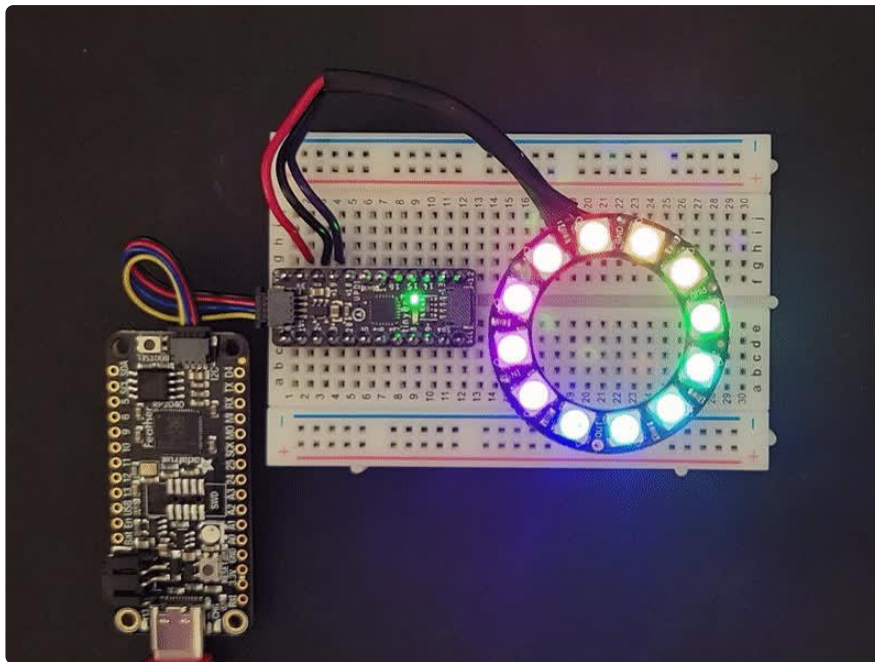
NEOPIXEL_PIN = 19 # Can be any pin
NEOPIXEL_NUM = 12 # No more than 60 pixels!

pixels = neopixel.NeoPixel(ss, NEOPIXEL_PIN, NEOPIXEL_NUM)
pixels.brightness = 0.3 # Not so bright!

color_offset = 0 # Start at red

# Cycle through all colors along the ring
while True:
    for i in range(NEOPIXEL_NUM):
        rc_index = (i * 256 // NEOPIXEL_NUM) + color_offset
        pixels[i] = colorwheel(rc_index & 255)
        color_offset += 1
    time.sleep(0.01)
```

The NeoPixel ring lights up in a rainbow!



First you import all the necessary modules and libraries. Then you instantiate the seesaw on I2C.

Next, you set the NeoPixel pin, and the number of pixels to match your wiring and pixel number. This example uses pin 19 and a 12-pixel ring.

Then, you create the pixels object with the pin and number you set above.

Before the loop, you set pixel brightness to 30% and create a `color_offset` variable and set it to 0 to start the colorwheel at red.

Inside the loop, you display the rainbow with a slight delay. Increase this to slow down the rainbow if desired.

That's all there is to using CircuitPython seesaw NeoPixel with the ATtiny817 breakout!

PWMOut

The seesaw firmware that ships with the ATtiny817 breakout includes PWM capabilities on specific pins. This example fades an external LED.

Follow the instructions on the [Python & CircuitPython page \(https://adafru.it/Vsf\)](https://adafru.it/Vsf) to get set up.

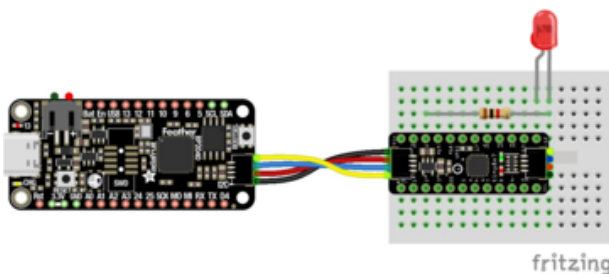
PWM Pins

The ATtiny817 breakout with seesaw firmware provides PWM on the following pins:

- 0, 1, 9, 12, 13

Wiring

This example uses an external LED. Wire it up as follows.



- Use a STEMMA QT cable to connect the STEMMA QT connector on the Feather to the STEMMA QT connector on the breakout.
- Connect the + leg (longer leg) of LED to breakout pin 12
- Connect the - leg (shorter leg) of LED to 1k Ω resistor
- Connect 1k Ω resistor to breakout GND

Example Code

Update your code.py to the following.

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

# Simple seesaw test for writing PWM outputs
# On the SAMD09 breakout these are pins 5, 6, and 7
# On the ATtiny8x7 breakout these are pins 0, 1, 9, 12, 13
#
# See the seesaw Learn Guide for wiring details.
# For SAMD09:
# https://learn.adafruit.com/adafruit-seesaw-atsamd09-breakout?
view=all#circuitpython-wiring-and-test
# For ATtiny8x7:
# https://learn.adafruit.com/adafruit-attiny817-seesaw/pwmout

import time
import board
from adafruit_seesaw import seesaw, pwmout

ss = seesaw.Seesaw(board.I2C())

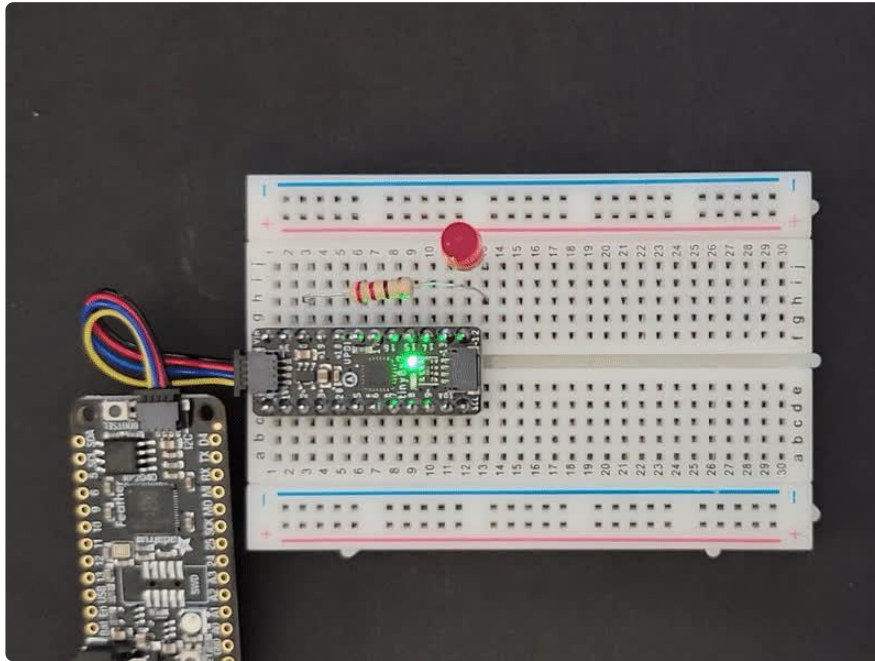
PWM_PIN = 12 # If desired, change to any valid PWM output!
led = pwmout.PWMOut(ss, PWM_PIN)
```

```

delay = 0.01
while True:
    # The API PWM range is 0 to 65535, but we increment by 256 since our
    # resolution is often only 8 bits underneath
    for cycle in range(0, 65535, 256): #
        led.duty_cycle = cycle
        time.sleep(delay)
    for cycle in range(65534, 0, -256):
        led.duty_cycle = cycle
        time.sleep(delay)

```

The brightness of the LED will fade up and then down, and repeat!



First, you import all the necessary modules and libraries. Then you instantiate the seesaw on I2C.

Next, you choose a pin for the LED, and then create the LED `PWMOut` object and provide it the `PWM_PIN`.

Before the loop, you set a delay of 0.01 seconds.

Inside the loop, you initially cycle up through the PWM range (0 - 65535) in increments of 256, and once complete, cycle down from the max back to 0 in increments of 256. The step is limited to increments of 256 because the resolution is often only 8 bits underneath. Both cycles have the delay included.

That's all there is to using CircuitPython seesaw PWM to fade an LED on the ATtiny817 breakout!

EEPROM

The seesaw firmware that ships with the ATtiny817 breakout provides access to the 128 byte EEPROM. This example reads from and writes to the EEPROM.

Follow the instructions on the [Python & CircuitPython page \(https://adafru.it/Vsf\)](https://adafru.it/Vsf) to get set up.

Example Code

Update your code.py to the following.

Before saving the file, connect to the serial console. The print statements are only sent the first time the code runs, so if you connect after you save the file, you may not see them printed to the serial console.

```
# SPDX-FileCopyrightText: 2021 ladyada for Adafruit Industries
# SPDX-License-Identifier: MIT

# Simple seesaw test reading and writing the internal EEPROM
# The ATtiny8xx series has a true 128 byte EEPROM, the SAMD09 mimics it in flash
# with 64 bytes
# THE LAST BYTE IS USED FOR I2C ADDRESS CHANGE!

import time
import board
from adafruit_seesaw import seesaw

i2c_bus = board.I2C()
ss = seesaw.Seesaw(i2c_bus)

value = ss.eeprom_read8(0x02) # Read from address 2
print("Read 0x%02x from EEPROM address 0x02" % value)

print("Incrementing value")
ss.eeprom_write8(0x02, (value + 1) % 0xFF)

value = ss.eeprom_read8(0x02) # Read from address 2
print("Second read 0x%02x from EEPROM address 0x02" % value)

while True:
    # Do not write EEPROM in a loop, it has 100k cycle life
    time.sleep(1)
```

The print statements will show up in the serial console, but only the first time the code is run. If you connect to the serial console after saving, you may not see anything as there are no prints in the loop.


```
Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.  
code.py output:  
Read 0xff from EEPROM address 0x02  
Incrementing value  
Second read 0x01 from EEPROM address 0x02
```

First, you import all the necessary modules and libraries, and you instantiate the seesaw on I2C.

Next, you read the value from address 2.

Then, you increment that value by +1.

Finally, you read the new value from address 2.

Inside the loop, is a 1 second delay. Do not write EEPROM in a loop, because it has a 100k cycle life.

Python Docs

[Python Docs \(https://adafru.it/C5y\)](https://adafru.it/C5y)

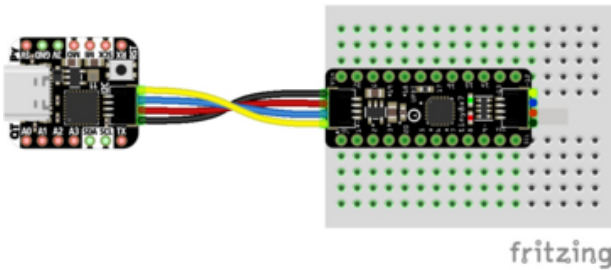
Arduino

The Adafruit Seesaw library makes it easy to use your ATtiny817 breakout with Arduino. Install the necessary libraries, and load the examples to use your ATtiny817 breakout for all sorts of purposes. This example is designed to verify that your breakout is working - it simply blinks the built-in LED. No hardware other than the breakout and microcontroller is needed.

Wiring

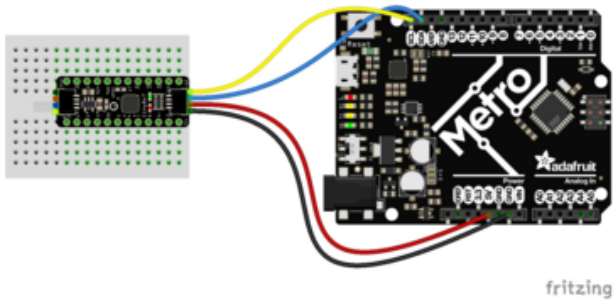
You can use many different Arduino boards, but this example shows the QT Py M0, and the Adafruit Metro. Connect up the ATtiny817 breakout as shown below.

Here is the QT Py wired up:



- Use a STEMMA QT cable to connect the STEMMA QT connector on the QT Py to the STEMMA QT connector on the breakout.

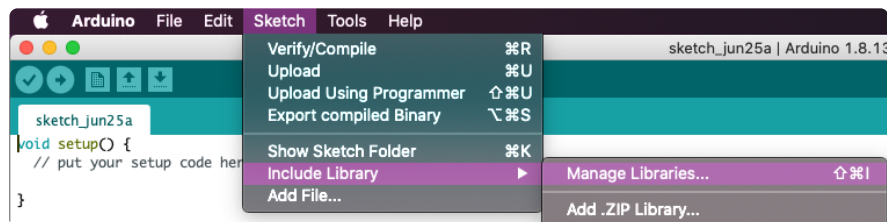
Here is the Metro wired up:



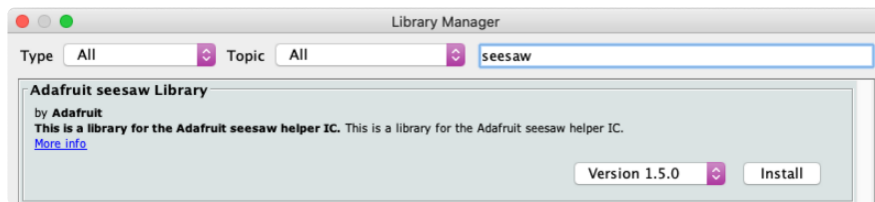
- Plug a [STEMMA QT to male jumper wire cable \(https://adafru.it/FA-\)](https://adafru.it/FA-) into the STEMMA QT connector on the breakout.
- Metro 5V to breakout VIN (red wire)
- Metro GND to breakout GND (black wire)
- Metro SCL to breakout SCL (yellow wire)
- Metro SDA to breakout SDA (blue wire)

Library Installation

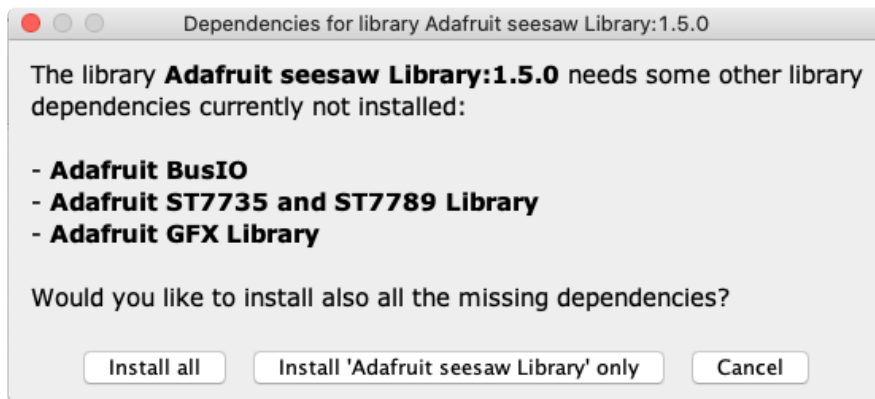
You can install the Adafruit Seesaw Library for Arduino using the Library Manager in the Arduino IDE:



Click the Manage Libraries ... menu item, search for seesaw, and select the Adafruit Seesaw library:



If asked to install dependencies, choose Install all.



Load Example

Open up File -> Examples -> Adafruit Seesaw -> digital -> attiny_blink and upload to your Arduino wired up to the breakout.

Upload the sketch to your board and open up the Serial Monitor (Tools->Serial Monitor). You should see **seesaw started ok!**. If you don't, check your wiring.

```
/*
 * This example shows how to blink a pin on a seesaw.
 * It is written to use the built-in LED on the ATtiny817 breakout with seesaw.
 */

#include "Adafruit_seesaw.h"

Adafruit_seesaw ss;

#define BLINK_PIN 5

void setup() {
  Serial.begin(115200);

  while (!Serial) delay(10); // wait until serial port is opened

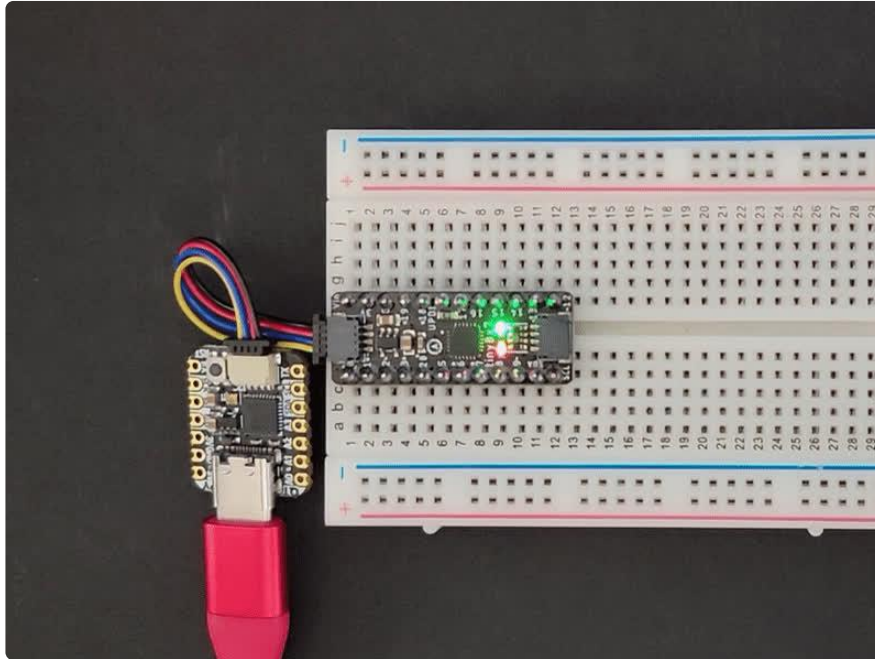
  if(!ss.begin()){
    Serial.println("seesaw not found!");
    while(1) delay(10);
  }

  Serial.println(F("seesaw started OK!"));

  ss.pinMode(BLINK_PIN, OUTPUT);
}
```

```
void loop() {  
  ss.digitalWrite(BLINK_PIN, LOW); // turn the LED on (the LED is tied low)  
  delay(1000); // wait for a second  
  ss.digitalWrite(BLINK_PIN, HIGH); // turn the LED off  
  delay(1000);  
}
```

Once you've successfully loaded the sketch onto your board, the little red LED will begin blinking!



Analog In

The seesaw firmware that ships with the ATtiny817 breakout includes analog capabilities on specific pins. This example reads the analog value of a pin.

Follow the steps on the [Arduino page \(https://adafru.it/VrF\)](https://adafru.it/VrF) to get set up.

Analog Pins

The ATtiny817 breakout with seesaw firmware provides analog on the following pins:

- 0, 1, 2, 3, 6, 7, 18, 19, 20

Example Code

Open up File -> Examples -> Adafruit Seesaw -> analog -> analogRead and upload to your Arduino wired up to the breakout.

```
/*
 * This example shows how read the ADC on a seesaw.
 * The default ADC pins on the SAMD09 Breakout are 2, 3, and 4.
 */

#include "Adafruit_seesaw.h"

Adafruit_seesaw ss;
// on SAMD09, analog in can be 2, 3, or 4
// on Attiny8x7, analog in can be 0-3, 6, 7, 18-20
#define ANALOGIN 2

void setup() {
  Serial.begin(115200);

  while (!Serial) delay(10); // wait until serial port is opened

  if(!ss.begin()){
    Serial.println(F("seesaw not found!"));
    while(1) delay(10);
  }

  Serial.println(F("seesaw started OK!"));
}

void loop() {
  Serial.println(ss.analogRead(ANALOGIN));
  delay(50);
}
```

Once you've successfully loaded the sketch onto your board, open the Serial Monitor (Tools->Serial Monitor). Your output should look something like the following.



PWM

The seesaw firmware that ships with the ATtiny817 breakout includes PWM capabilities on specific pins. This example fades an external LED.

Follow the steps on the [Arduino page \(https://adafru.it/VrF\)](https://adafru.it/VrF) to get set up.

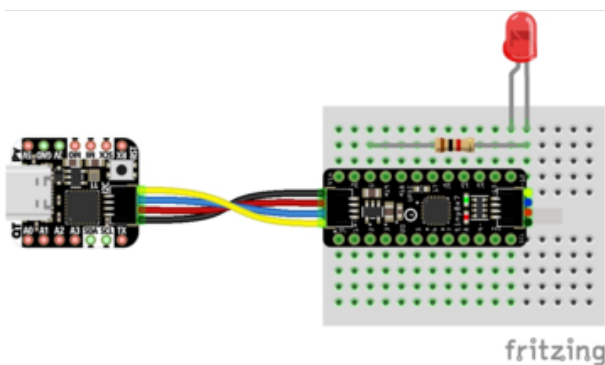
PWM Pins

The ATtiny817 breakout with seesaw firmware provides PWM on the following pins:

- 0, 1, 9, 12, 13

Wiring

Follow the instructions to add an external LED to your existing setup.



- Use a STEMMA QT cable to connect the STEMMA QT connector on the Qt Py to the STEMMA QT connector on the breakout.
- Connect the + leg (longer leg) of LED to breakout pin 12
- Connect the - leg (shorter leg) of LED to 1k Ω resistor
- Connect 1k Ω resistor to breakout GND

Example Code

Open up File -> Examples -> Adafruit Seesaw -> analog -> Fade.

Before you upload it to your microcontroller, you must make a change for it to work with the ATtiny817 breakout.

This example will not work with the ATtiny817 breakout as is. You must update the led pin before uploading the sketch!

Update the following line:

```
int led = 6;           // the PWM pin the LED is attached to
```

To one of the PWM capable pins on the ATtiny817, such as pin 12 to match the wiring diagram:

```
int led = 12;           // the PWM pin the LED is attached to
```

You must do this before uploading the sketch, or it won't work with your setup!

```
/*
  Fade

  This example shows how to fade an LED on pin 6 of a seesaw board using the
  analogWrite()
  function.

  The analogWrite() function uses PWM, so if you want to change the pin you're
  using, be sure to use another PWM capable pin.
  On the SAMD09 breakout these are pins 5, 6, and 7
  On the ATtiny8x7 breakout these are pins 0, 1, 9, 12, 13
*/

#include "Adafruit_seesaw.h"

Adafruit_seesaw ss;

int led = 6;           // the PWM pin the LED is attached to
int brightness = 0;    // how bright the LED is
int fadeAmount = 5;    // how many points to fade the LED by

// the setup routine runs once when you press reset:
void setup() {
  Serial.begin(115200);

  while (!Serial) delay(10); // wait until serial port is opened

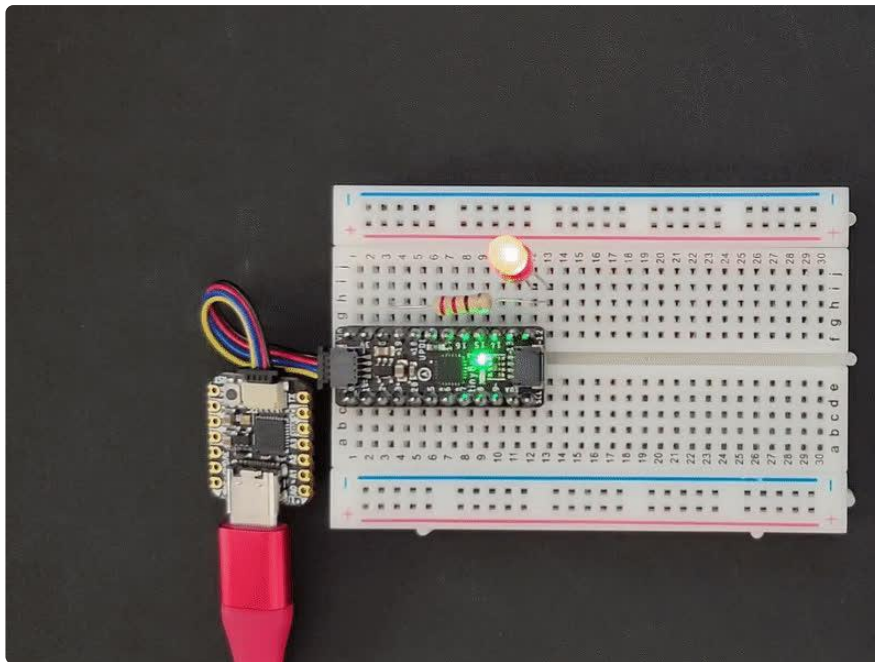
  if(!ss.begin()){
    Serial.println("seesaw not found!");
    while(1) delay(10);
  }
}

// the loop routine runs over and over again forever:
void loop() {
  // set the brightness of the LED:
  ss.analogWrite(led, brightness);

  // change the brightness for next time through the loop:
  brightness = brightness + fadeAmount;

  // reverse the direction of the fading at the ends of the fade:
  if (brightness <= 0 || brightness >= 255) {
    fadeAmount = -fadeAmount;
  }
  // wait for 30 milliseconds to see the dimming effect
  delay(30);
}
```

Once you've successfully uploaded the sketch to your board, open the Serial Monitor (Tools->Serial Monitor), and the LED will begin to fade bright and dim!



NeoPixel

The seesaw firmware that ships with the ATtiny817 breakout includes the ability to power up to 60 NeoPixels on one pin. This example displays a series of color wipes, a theatre chase effect in different colors, and a rainbow across a NeoPixel ring.

Follow the steps on the [Arduino page \(https://adafru.it/VrF\)](https://adafru.it/VrF) to get set up.

NeoPixel Pins

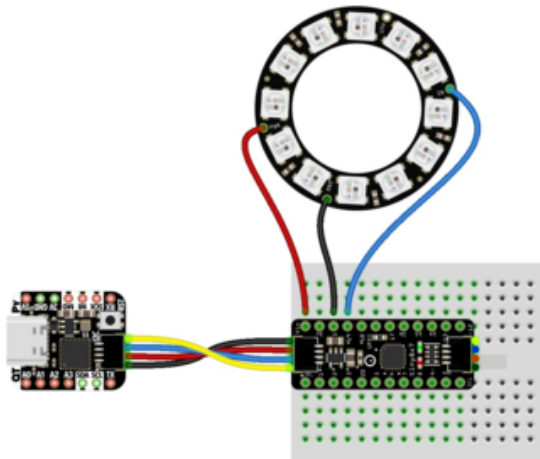
There are 15 pins on the ATtiny817 breakout that can be used for powering NeoPixels. However, you can only use one pin at a time! The available pins are:

- 0, 1, 2, 3, 20, 5, 6, 7, 8, 9, 12, 13, 14, 18, 19

Pins 15, 16 and 17 can NOT be used as NeoPixel pins with the seesaw firmware!

Wiring

Follow the instructions to add an external LED to your existing setup.



- Use a STEMMA QT cable to connect the STEMMA QT connector on the Feather to the STEMMA QT connector on the breakout.
- Connect NeoPixel ring GND to breakout GND
- Connect NeoPixel ring IN (data in) to breakout pin 19
- Connect NeoPixel ring power to breakout VIN

Example Code

Open up File -> Examples -> Adafruit Seesaw -> NeoPixel -> NeoPixel_strandtest.

Before you upload it to your microcontroller, you must make a change for it to work with the ATtiny817 breakout.

This example will not work with the ATtiny817 breakout as is. You must update the pin before uploading the sketch!

Update the following line:

```
#define PIN 10
```

To one of the NeoPixel capable pins on the ATtiny817, such as pin 19 to match the wiring diagram:

```
#define PIN 19
```

You must do this before uploading the sketch, or it won't work with your setup!

```
#include <seesaw_neopixel.h>
#define PIN 10

// Parameter 1 = number of pixels in strip
// Parameter 2 = Arduino pin number (most are valid)
// Parameter 3 = pixel type flags, add together as needed:
//   NEO_KHZ800  800 KHz bitstream (most NeoPixel products w/WS2812 LEDs)
//   NEO_KHZ400  400 KHz (classic 'v1' (not v2) FLORA pixels, WS2811 drivers)
//   NEO_GRB     Pixels are wired for GRB bitstream (most NeoPixel products)
//   NEO_RGB     Pixels are wired for RGB bitstream (v1 FLORA pixels, not v2)
//   NEO_RGBW    Pixels are wired for RGBW bitstream (NeoPixel RGBW products)
seesaw_NeoPixel strip = seesaw_NeoPixel(12, PIN, NEO_GRB + NEO_KHZ800);
```

```

// IMPORTANT: To reduce NeoPixel burnout risk, add 1000 uF capacitor across
// pixel power leads, add 300 - 500 Ohm resistor on first pixel's data input
// and minimize distance between Arduino and first pixel. Avoid connecting
// on a live circuit...if you must, connect GND first.

void setup() {
  Serial.begin(115200);

  while (!Serial) delay(10); // wait until serial port is opened

  if(!strip.begin()){
    Serial.println("seesaw not found!");
    while(1) delay(10);
  }

  Serial.println(F("seesaw started OK!"));

  strip.show(); // Initialize all pixels to 'off'
}

void loop() {
  // Some example procedures showing how to display to the pixels:
  colorWipe(strip.Color(255, 0, 0), 50); // Red
  colorWipe(strip.Color(0, 255, 0), 50); // Green
  colorWipe(strip.Color(0, 0, 255), 50); // Blue
  //colorWipe(strip.Color(0, 0, 0, 255), 50); // White RGBW
  // Send a theater pixel chase in...
  theaterChase(strip.Color(127, 127, 127), 50); // White
  theaterChase(strip.Color(127, 0, 0), 50); // Red
  theaterChase(strip.Color(0, 0, 127), 50); // Blue

  rainbow(20);
  rainbowCycle(20);
  theaterChaseRainbow(50);
}

// Fill the dots one after the other with a color
void colorWipe(uint32_t c, uint8_t wait) {
  for(uint16_t i=0; i<strip.numPixels(); i++) {
    strip.setPixelColor(i, c);
    strip.show();
    delay(wait);
  }
}

void rainbow(uint8_t wait) {
  uint16_t i, j;

  for(j=0; j<256; j++) {
    for(i=0; i<strip.numPixels(); i++) {
      strip.setPixelColor(i, Wheel((i+j) & 255));
    }
    strip.show();
    delay(wait);
  }
}

// Slightly different, this makes the rainbow equally distributed throughout
void rainbowCycle(uint8_t wait) {
  uint16_t i, j;

  for(j=0; j<256*5; j++) { // 5 cycles of all colors on wheel
    for(i=0; i< strip.numPixels(); i++) {
      strip.setPixelColor(i, Wheel(((i * 256 / strip.numPixels()) + j) & 255));
    }
    strip.show();
    delay(wait);
  }
}

```



```

//Theatre-style crawling lights.
void theaterChase(uint32_t c, uint8_t wait) {
  for (int j=0; j<10; j++) { //do 10 cycles of chasing
    for (int q=0; q < 3; q++) {
      for (uint16_t i=0; i < strip.numPixels(); i=i+3) {
        strip.setPixelColor(i+q, c);    //turn every third pixel on
      }
      strip.show();

      delay(wait);

      for (uint16_t i=0; i < strip.numPixels(); i=i+3) {
        strip.setPixelColor(i+q, 0);    //turn every third pixel off
      }
    }
  }
}

//Theatre-style crawling lights with rainbow effect
void theaterChaseRainbow(uint8_t wait) {
  for (int j=0; j < 256; j++) { // cycle all 256 colors in the wheel
    for (int q=0; q < 3; q++) {
      for (uint16_t i=0; i < strip.numPixels(); i=i+3) {
        strip.setPixelColor(i+q, Wheel( (i+j) % 255));    //turn every third pixel
on
      }
      strip.show();

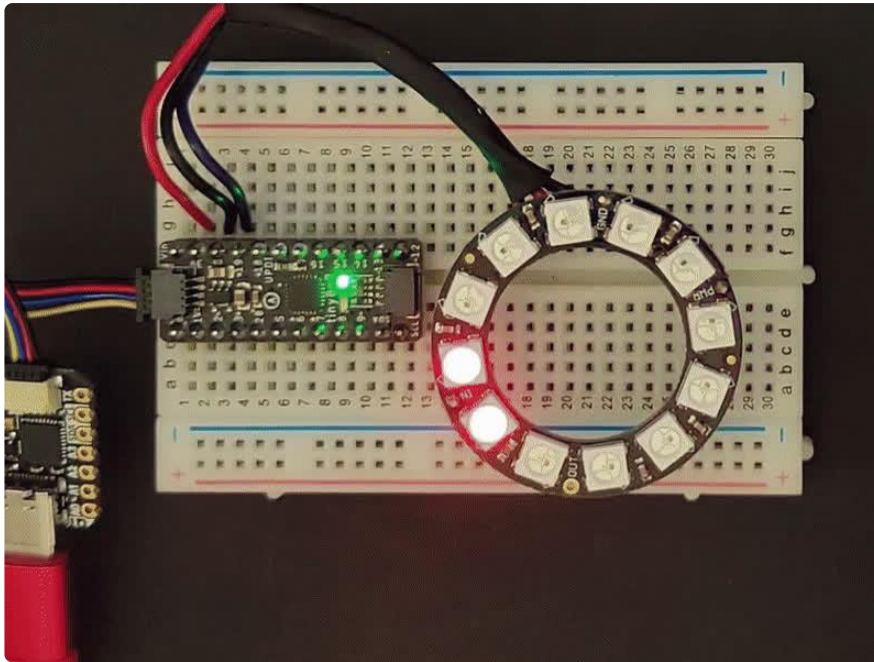
      delay(wait);

      for (uint16_t i=0; i < strip.numPixels(); i=i+3) {
        strip.setPixelColor(i+q, 0);    //turn every third pixel off
      }
    }
  }
}

// Input a value 0 to 255 to get a color value.
// The colours are a transition r - g - b - back to r.
uint32_t Wheel(byte WheelPos) {
  WheelPos = 255 - WheelPos;
  if(WheelPos < 85) {
    return strip.Color(255 - WheelPos * 3, 0, WheelPos * 3);
  }
  if(WheelPos < 170) {
    WheelPos -= 85;
    return strip.Color(0, WheelPos * 3, 255 - WheelPos * 3);
  }
  WheelPos -= 170;
  return strip.Color(WheelPos * 3, 255 - WheelPos * 3, 0);
}

```

Once successfully uploaded, your NeoPixel light show will begin! Color wipe, followed by theatre chase, followed by full rainbow, followed by rainbow cycle.



EEPROM

The seesaw firmware that ships with the ATtiny817 breakout provides access to the 128 byte EEPROM. This example reads from and writes to the EEPROM.

Follow the steps on the [Arduino page \(https://adafru.it/VrF\)](https://adafru.it/VrF) to get set up.

Example Code

Open up File -> Examples -> Adafruit Seesaw -> EEPROM.

```
/*
 * This example shows how to read and write EEPROM data. Try writing
 * then removing power from both devices, commenting out the write, and
 * uploading again.
 */

#include "Adafruit_seesaw.h"

Adafruit_seesaw ss;

void setup() {
  uint8_t eepromval;

  Serial.begin(115200);

  while (!Serial) delay(10); // wait until serial port is opened

  if(!ss.begin()){
    Serial.println(F("seesaw not found!"));
    while(1) delay(10);
  }

  Serial.println(F("seesaw started OK!"));
```

```

Serial.print(F("Initial read from address 0x02...0x"));
eepromval = ss.EEPROMRead8(0x02);
Serial.println(eepromval, HEX);

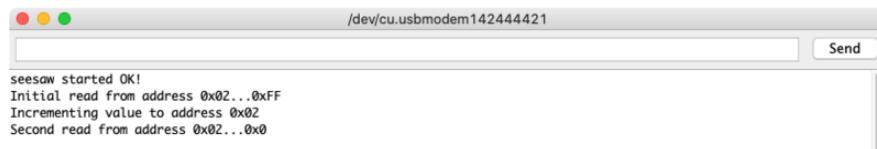
Serial.println(F("Incrementing value to address 0x02"));
ss.EEPROMWrite8(0x02, eepromval+1);

Serial.print(F("Second read from address 0x02...0x"));
Serial.println(ss.EEPROMRead8(0x02), HEX);
}

void loop() {
  //DONT WRITE EEPROM IN A LOOP!!!! YOU WILL DESTROY YOUR FLASH!!!
}

```

Once you've successfully uploaded the sketch to your board, open the Serial Monitor (Tools->Serial Monitor), and see the info printed out!



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 not officially supported by Adafruit such as C/C++ on Raspberry Pi.

Reading and Writing Data

The SeeSaw operates as an I2C secondary 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 0), 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.beginTransaction((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 1) 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_t delay)
{
    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)

Writing a 1 to any bit in this register enables the internal pullup or pulldown on the corresponding pin. The pull direction (up/down) is determined by the GPIO (output) value - if the corresponding GPIO register bit is low, its a pulldown. High, its a pullup.

Writing zeros to this register has no effect.

PULLENCLR (0x0C, 32 bits, Write Only)

Writing a 1 to any bit in this register disables the pull up/down on the corresponding pin.

Writing zeros to this register has no effect.

Analog to Digital Converter

The ADC provides the ability to measure analog voltages at 10-bit resolution. The seesaw has 4 ADC inputs.

The module base register address for the ADC is 0x09

Conversions can be read by reading the corresponding CHANNEL register.

When reading ADC data, there should be at least a 500 microsecond delay between writing the register number you would like to read from and attempting to read the data.

Allow a delay of at least 1ms in between sequential ADC reads on different channels.

ADC channels are:

Channel 0	PA02
Channel 1	PA03
Channel 2	PA04
Channel 3	PA05

Function Registers

Register Address	Register Name	Register Size	Notes
0x00	STATUS	8 bits	Read Only
0x02	INTENSET	8 bits	Write Only
0x03	INTENCLR	8 bits	Write Only
0x04	WINMODE		Write Only
0x05	WINTHRESH	32 bits	Write Only

0x07	CHANNEL_0	16 bits	Read Only
0x08	CHANNEL_1	16 bits	Read Only
0x09	CHANNEL_2	16 bits	Read Only
0x0A	CHANNEL_3	16 bits	Read Only

Window mode or ADC interrupts is not yet supported as of the time of writing this guide.

STATUS (0x00, 8bits, Read Only)

This register contains status information on the ADC

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	WINMON_INT	ERROR

INTENSET (0x02, 8bits, Write Only)

Writing a 1 to any bit in this register enables the corresponding interrupt.

Writing zeros to this register has no effect.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	WINMON
----------	----------	----------	----------	----------	----------	----------	--------

INTENCLR (0x03, 8bits, Write Only)

Writing a 1 to any bit in this register enables the corresponding interrupt.

Writing zeros to this register has no effect.

WINMODE (0x04, 8bits, Write Only)

Writing 1 to this register sets window control.

WINTHRESH (0x05, 32bits, Write Only)

This register sets the threshold values for window mode.

Bits 31 - 16	Bits 15 - 0
High Threshold	Low Threshold

CHANNEL_0 (0x07, 16bits, Read Only)

ADC value for channel 0 (PA02).

CHANNEL_1 (0x08, 16bits, Read Only)

ADC value for channel 1 (PA03).

CHANNEL_2 (0x09, 16bits, Read Only)

ADC value for channel 2 (PA04).

CHANNEL_3 (0x0A, 16bits, Read Only)

ADC value for channel 3 (PA05).

Interrupts

The seesaw has a configurable interrupt pin that can be triggered through various channels.

Once the interrupt is triggered, it can be only be cleared when the conditions of it's source module(s) have been met (e.g. data has been read, an interrupt has been cleared by reading an INTFLAG register).

See individual module sections for details on their available interrupt configurations.

The hardware interrupt pin is available on PA08 (#8)

EEPROM

The EEPROM module provides persistent storage of data across reboots.

On the SAMD09, there are 64 bytes of emulated EEPROM available for use. Byte 63 (0x3F) can be written to change the devices default I2C address.

On the ATtiny817, there are 128 bytes of actual EEPROM available for use. Byte 127 (0x7F) can be written to change the device's default I2C address.

The module base register address for the EEPROM module is 0x0D

The SAMD09 does not have true EEPROM, but flash memory on the seesaw that performs the same function. Performing a chip erase will erase all data stored in the emulated EEPROM. Also, be aware that the emulated EEPROM has a limited write/erase cycle lifespan. Care should be taken to not write/erase too many times or you will get inconsistent results and possibly damage the FLASH! The FLASH is rated for 100,000 cycles

Function Registers

SAMD09

Register Address	Function Name	Register Size	Notes
0x00 - 0x3E	General Purpose EEPROM	8 bits each	Read/Write
0x3F	I2C Address	8 bits	Read/Write

ATtiny817

Register Address	Function Name	Register Size	Notes
0x00 - 0x7E	General purpose EEPROM	8 bits each	Read/write
0x7F	I2C Address	8 bits	Read/write

The seesaw has built in NeoPixel support for up to 63 pixels (RGB pixels, less if RGBW). The output pin as well as the communication protocol frequency are configurable.

The module base register address for the NeoPixel module is 0x0E.

Function Registers

Register Address	Register Name	Register Size	Notes
------------------	---------------	---------------	-------

0x01	PIN	8 bits	Write Only
0x02	SPEED	8 bits	Write Only
0x03	BUF_LENGTH	16 bits	Write Only
0x04	BUF	32 bytes	Write Only
0x05	SHOW	none	Write Only

PIN (0x01, 8bits, Write Only)

This register sets the pin number (PORTA) that is used for the NeoPixel output.

SPEED (0x02, 8bits, Write Only)

The protocol speed.

0x00 = 400khz

0x01 = 800khz (default)

BUF_LENGTH (0x03, 16bits, Write Only)

the number of bytes currently used for the pixel array. This is dependent on when the pixels you are using are RGB or RGBW.

BUF (0x04, 32 bytes, Write Only)

The data buffer. The first 2 bytes are the start address, and the data to write follows. Data should be written in blocks of maximum size 30 bytes at a time.

Bytes 0 - 1	Bytes 2 - 32
Start address	Data

SHOW (0x05, no args, Write Only)

Sending the SHOW command will cause the output to update. There's no arguments/ data after the command

PWM

The PWM module provides up to 4 8-bit PWM outputs.

The module base register address for the PWM module is 0x08.

PWM outputs are available on pins PA04, PA05, PA06, and PA07.

Function Registers

Register Address	Register Name	Register Size	Notes
0x01	PWM_VAL	16 bits	Write Only

PWM_VAL (0x01, 16bits, Write Only)

Byte 0	Byte 1
PWM Number	PWM Value

The first byte written should be the PWM number you would like to write to. The second byte should be the pwm value.

PWM Number	Output Pin
0	PA04
1	PA05
2	PA06
3	PA07

Downloads

Files

- [ATtiny8x7 datasheet \(https://adafru.it/VhF\)](https://adafru.it/VhF)
- [EagleCAD PCB files on GitHub \(https://adafru.it/Via\)](https://adafru.it/Via)
- [Fritzing object in the Adafruit Fritzing Library \(https://adafru.it/Vib\)](https://adafru.it/Vib)

Schematic and Fab Print

