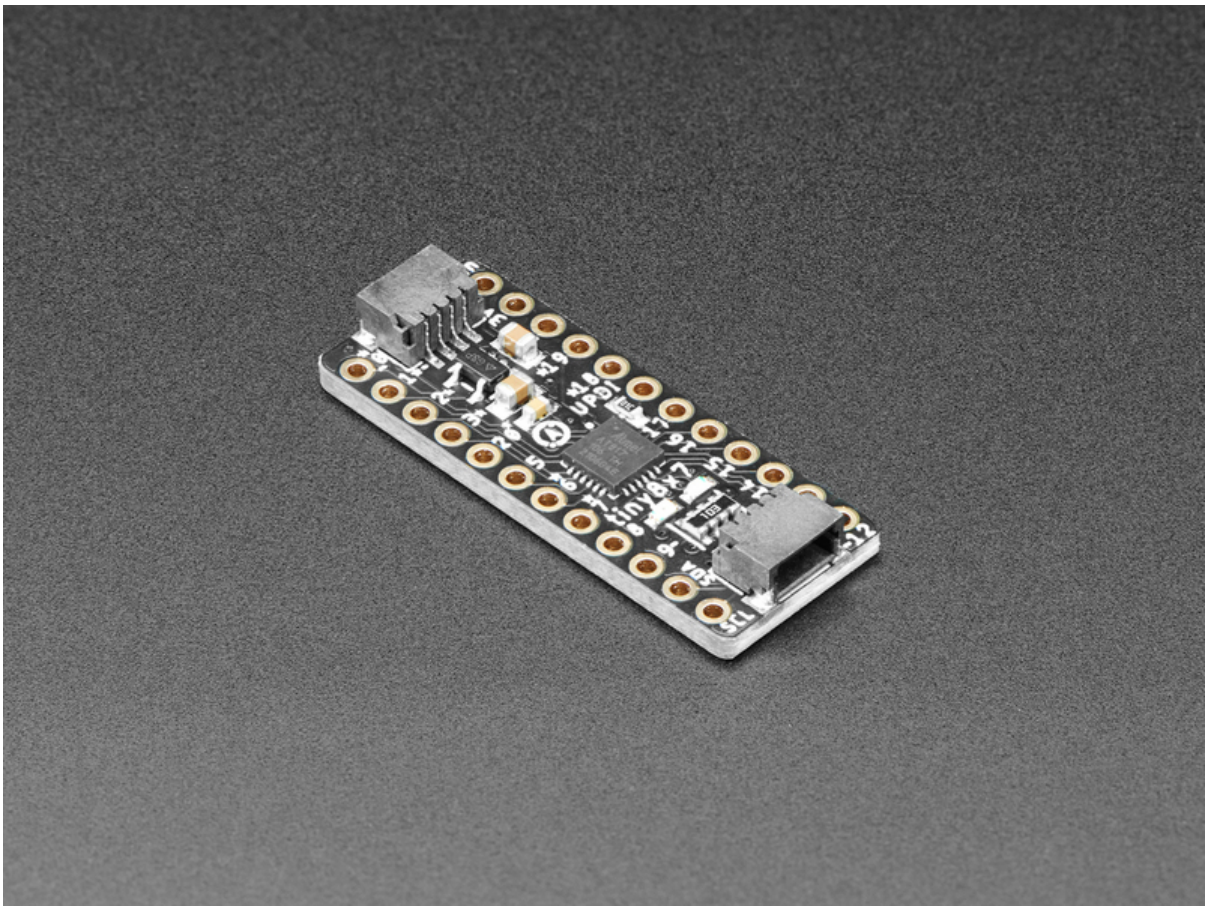
Adafruit ATtiny Breakouts with seesaw

Created by Kattni Rembor

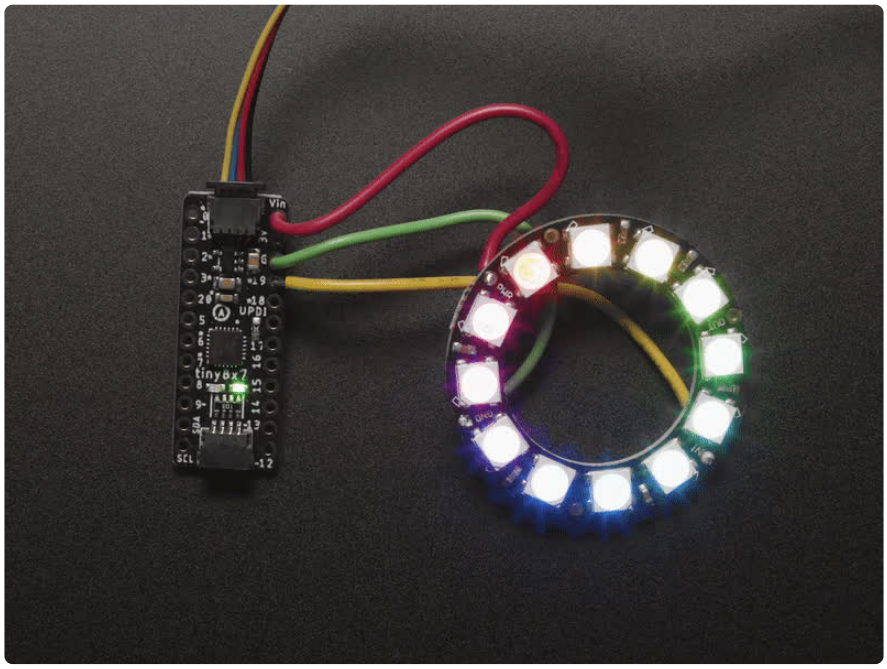


https://learn.adafruit.com/adafruit-attiny817-seesaw

Last updated on 2025-08-07 02:53:03 PM EDT

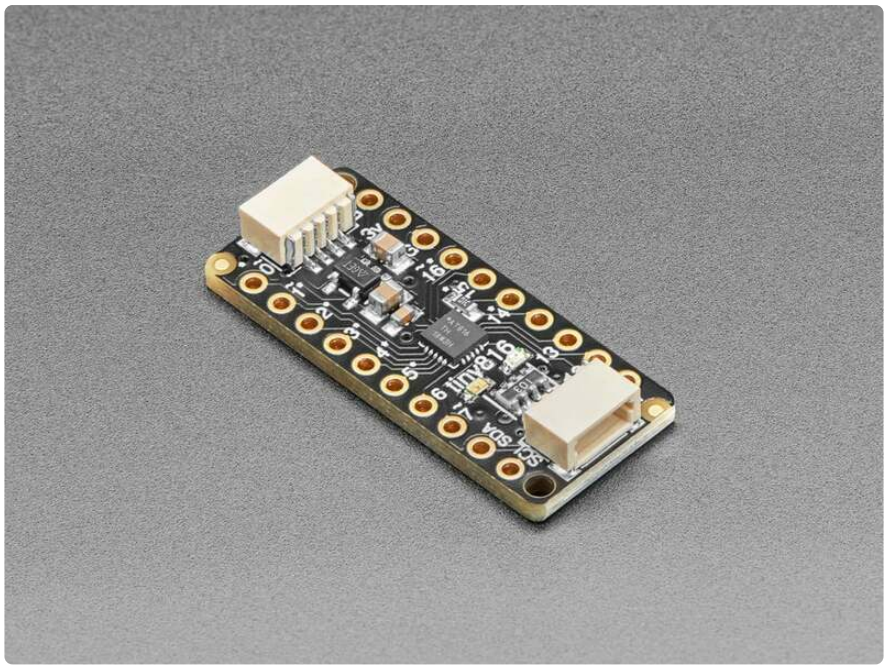
Table of Contents

Overview



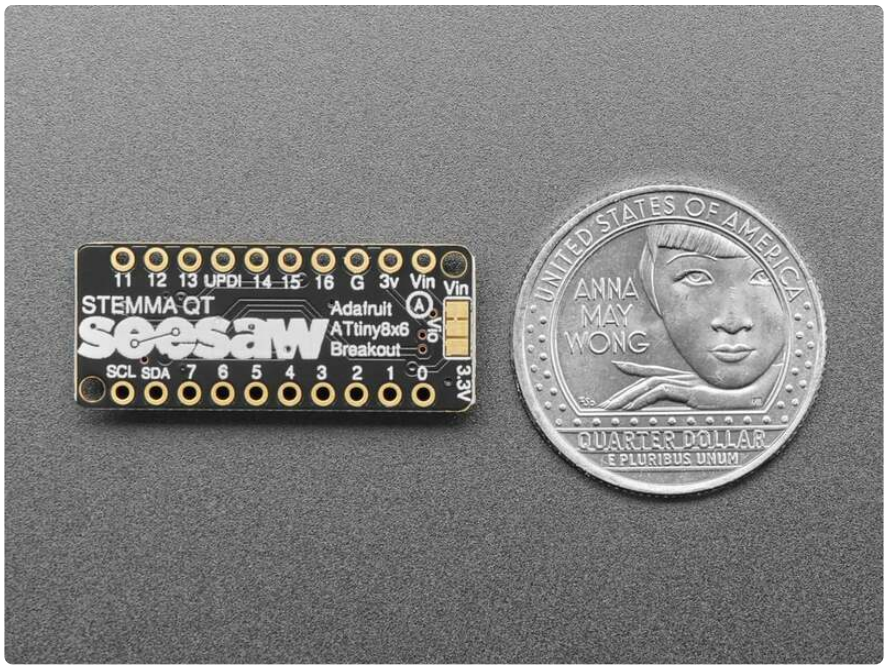
These are the Adafruit ATtiny817, ATtiny816 and ATtiny1616 breakouts with seesaw! These breakout boards are a "three in one" product:

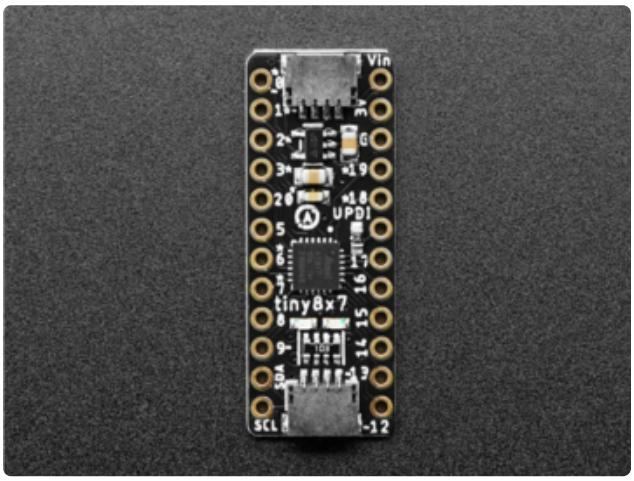
1. The ATtiny817, ATtiny816 and ATtiny1616 are part of the 'next gen' of AVR microcontrollers, and now we have a cute development/breakout board for them, with just enough hardware to get the chip up and running.
2. It's also an Adafruit seesaw board. Adafruit seesaw is a near-universal converter framework 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 ATtinyxxx, [just like we did for the ATSAMD09](http://adafru.it/3657). With the 2021-2022 silicon shortage, we're adapting some of our SAMD09 designs to the ATtinyxxx series and wanted a quick minimal board to test code on.

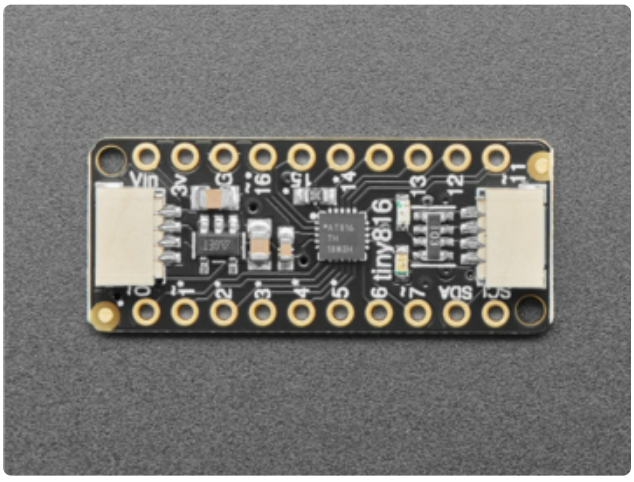
Each breakout comes with the assembled and tested board, as well as some header strips. Each PCB is fairly minimal and contains:





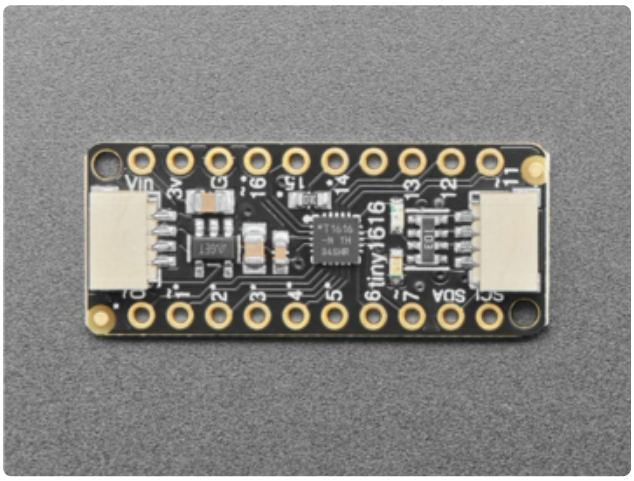
ATtiny817 8-bit microcontroller

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**.



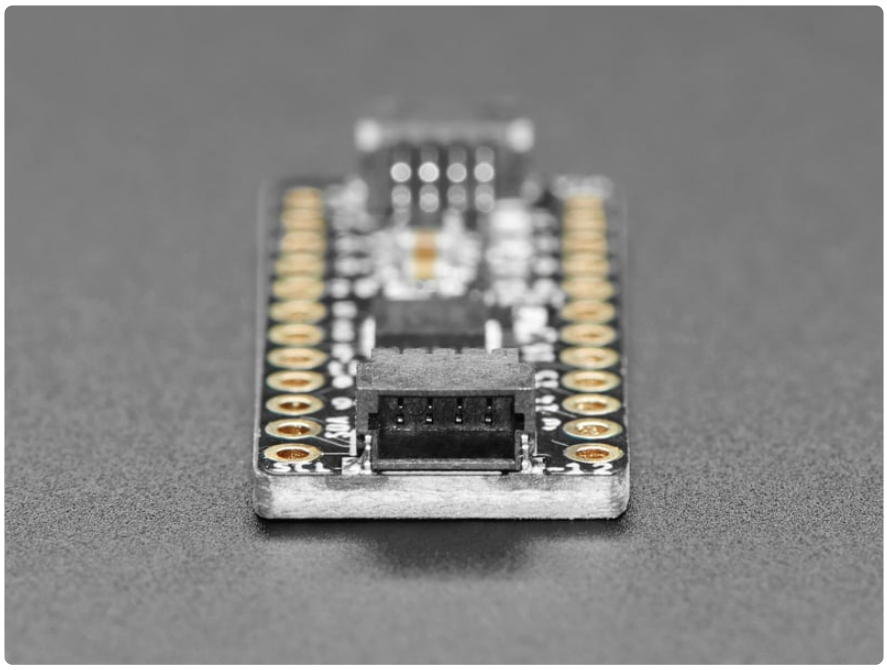
ATtiny816 8-bit microcontroller

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 8 and 9. The default I2C address is **0x49**.



ATtiny1616 8-bit microcontroller

16KB flash, 2KB of RAM, 256 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 8 and 9. The default I2C address is **0x49**.



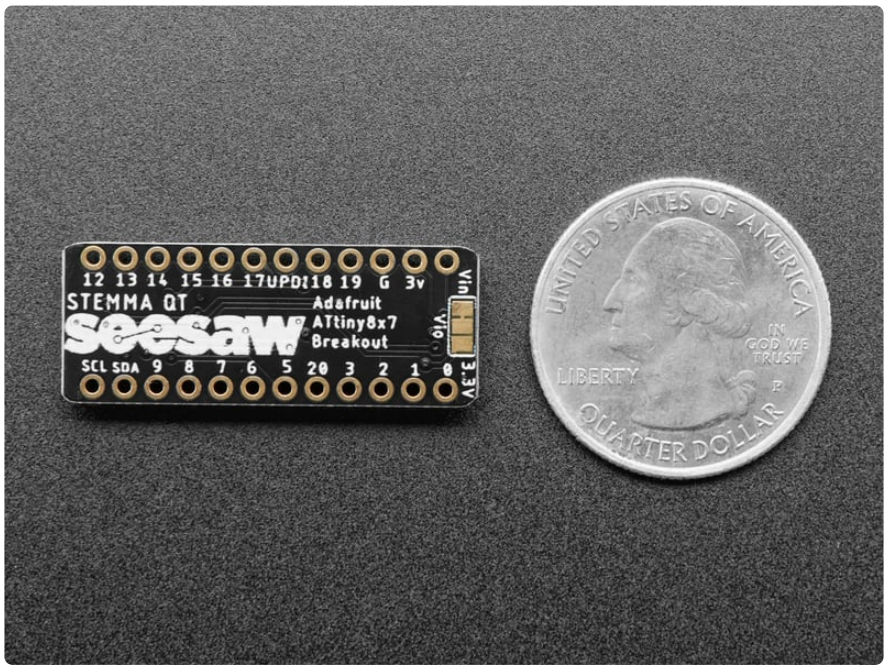
This boards comes pre-programmed with seesaw peripheral code (https://adafru.it/VdL) that will let it act as an "I2C to something" converter, basically a little I2C-controlled friend to do all the timing-sensitive things many microcontrollers and microcomputers are not good at.

For example, using the ATtiny817 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

Using the ATtinyx16 breakouts with pre-burned seesaw firmware gives you:

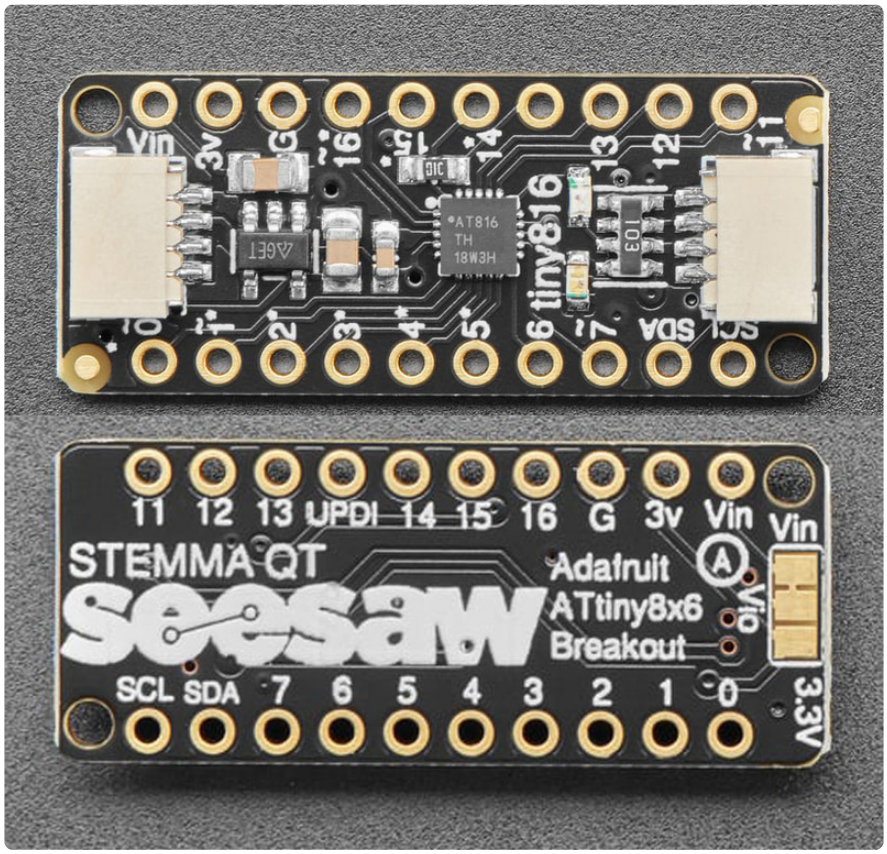
- 12 x GPIO with selectable pullup resistors: 0-5, 6, 8, 11, 14, 15, 16
- 9 x 10-bit ADC inputs - pins 0, 1, 2, 3, 4, 5, 14, 15, 16
- 5 x 8-bit PWM outputs - pins 0, 1, 7, 11, 16
- 1 x NeoPixel output (up to 250 pixels)
- 1 x EEPROM with 127 byte of NVM memory (handy for storing small access tokens or 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 6
- 2 x I2C address selection pins - pins 12 and 13
- 1 x Activity LED on pin 10, 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) to this series of chips. To program the chip you will need a 'UPDI' programmer (https://adafru.it/VdM), which you can make with a USB-to-Serial cable (http://adafru.it/954) and a single 4.7K or 10K resistor (http://adafru.it/2783).

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), 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!

ATtiny816 and ATtiny1616 Breakout Pinouts



The default I2C address is **0x49**.

Power Pins:

- **Vin** - this is the power pin. The ATtinyx16 operates between 1.8V and 5.5V. However, we still included a 3.3V voltage regulator. You can power the board from 2V 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.

- **3V** - This is the 3.3V output from the voltage regulator. You can grab up to 100mA from this if you like.
- **G** - 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

- **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) **-** 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:

- **0-5, 11, 14, 15, 16** - These are the 10 GPIO pins available with selectable pullup resistors.

seesaw Interrupt Pins:

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

seesaw I2C Address Pins:

The default I2C address is **0x49**.

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

seesaw NeoPixel-Capable Pins:

- **0-5, 11, 14, 15, 16** - 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:

- **0-5, 14, 15, 16** - There are nine 10-bit ADC pins.

PWM Pins:

- **0, 1, 7, 11, 16** - There are five 8-bit PWM output pins.

LED Pin

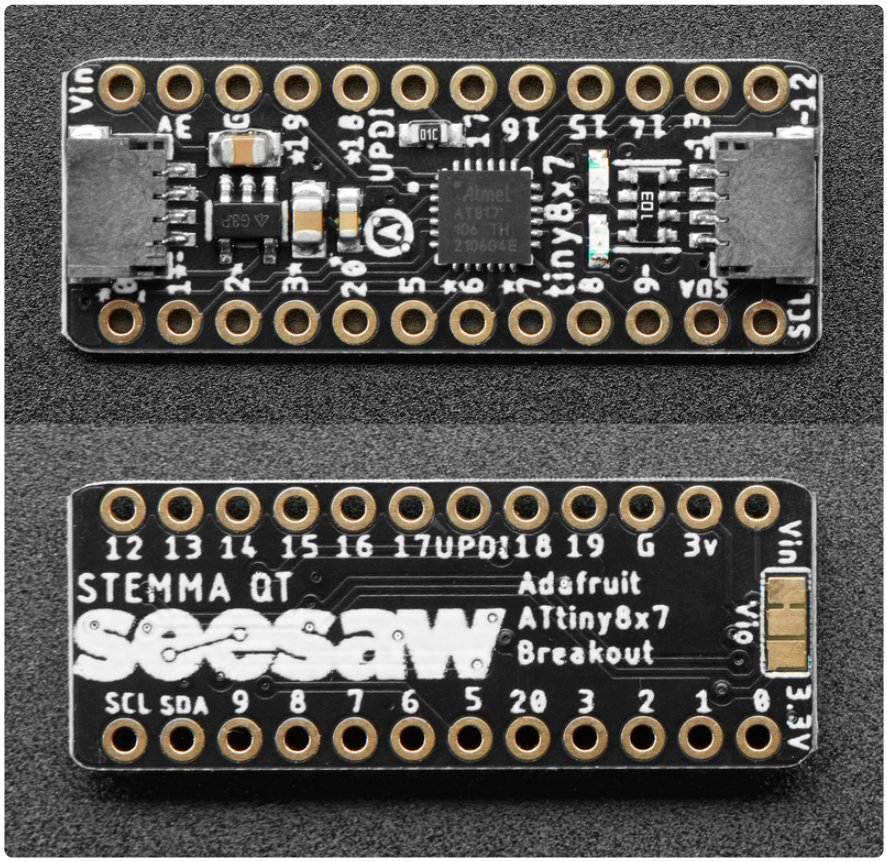
- For the 816 and 1616, there is a red LED that can be turned on by using seesaw Pin **10**. Note the LED pin is different on the 817.

Programming Pins:

- **UPDI** - This is the single-pin **U**nified **P**rogram and **D**ebug **I**nterface. 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!

ATtiny817 Breakout 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) **-** 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.

LED Pin

- For the 817 there is a red LED that can be turned on by using seesaw Pin **5**. Note the LED pin is different on the 816/1616.

Programming Pins:

- **UPDI** - This is the single-pin **U**nified **P**rogram and **D**ebug **I**nterface. 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!

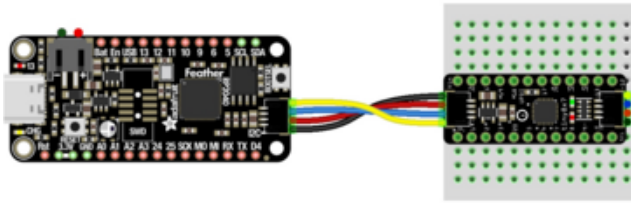
Python & CircuitPython

The Adafruit CircuitPython seesaw library (https://adafru.it/BrW) 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).

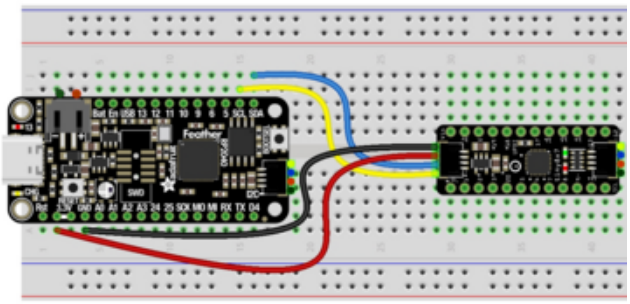
CircuitPython Microcontroller Wiring

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.

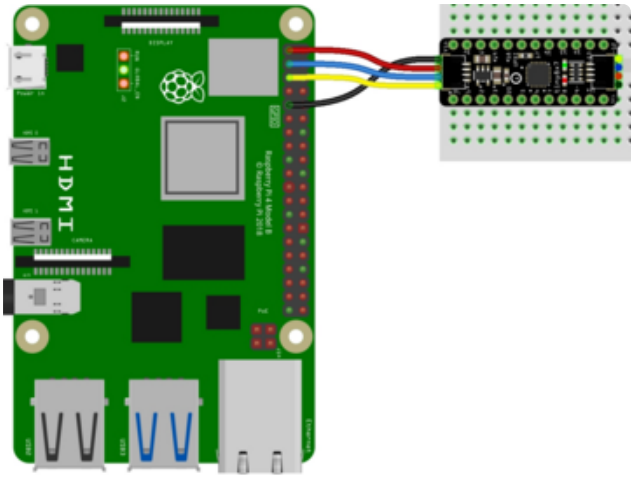


Plug a [STEMMA QT to male jumper wire cable **(http://adafru.it/4209)**](http://adafru.it/4209) into the **STEMMA QT connector on the breakout.**
Feather 3V to **breakout VIN (red wire)**
Feather GND to **breakout GND (black wire)**
Feather SCL to **breakout SCL (yellow wire)**
Feather SDA to **breakout 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 **(http://adafru.it/4209)**](http://adafru.it/4209) **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)!

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`

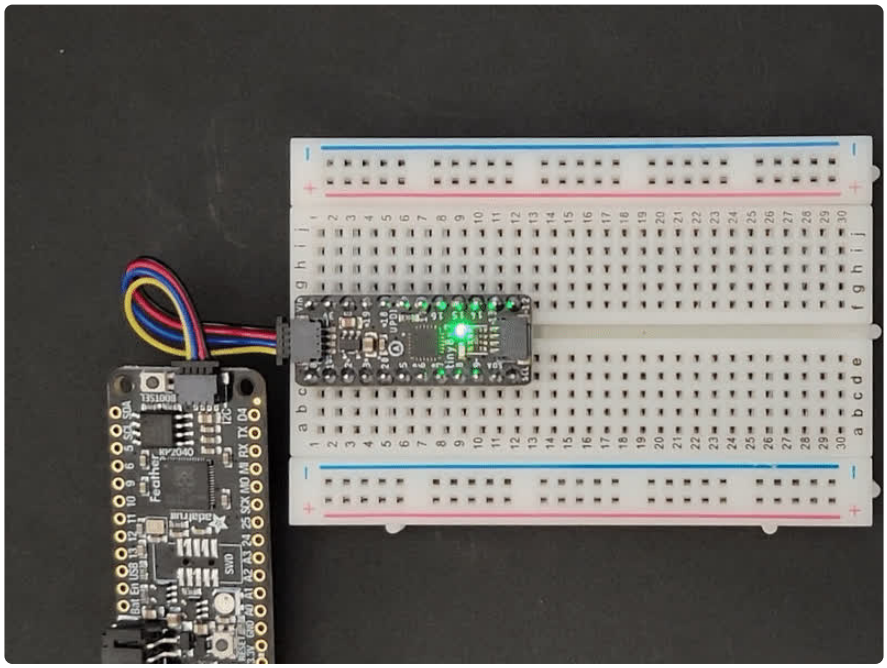
Example Code:

> The example below is for the 817. If you are using the 816/1616, please change seesaw Pin 5 to Pin 10. No other changes are needed.

```
# 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

i2c = board.I2C()  # uses board.SCL and board.SDA
# i2c = board.STEMMA_I2C()  # For using the built-in STEMMA QT connector on a
microcontroller
ss = Seesaw(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 ATtinyxxx breakouts 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) to get set up.

> Though the Fritzing diagrams depict an ATtiny817, the examples work as written with the ATtiny816 and ATtiny1616.

Analog Pins

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

- **0-3, 6, 7, 18-20**

The ATtiny816 and ATtiny1616 breakout with seesaw firmware provides analog on the following pins:

- **0-5, 14-16**

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.analoginput import AnalogInput
from adafruit_seesaw.seesaw import Seesaw

i2c = board.I2C()  # uses board.SCL and board.SDA
# i2c = board.STEMMA_I2C()  # For using the built-in STEMMA QT connector on a
microcontroller
ss = Seesaw(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 ATtinyxxx breakouts!

Digital Input

The seesaw firmware that ships with the ATtinyxxx breakouts 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) to get set up.

> Though the Fritzing diagrams depict an ATtiny817, the examples work as written with the ATtiny816 and ATtiny1616.

Digital Pins

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

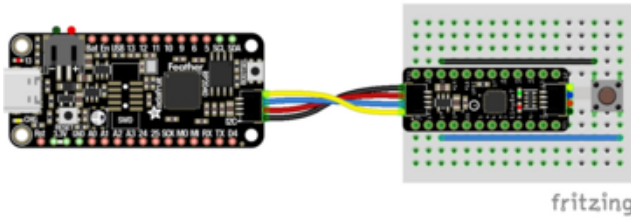
- **0-3, 5-9, 12-14, 18-20**

There are 12 pins on the ATtinyx16 breakouts that can be used with digitalio:

- **0-6, 8, 11, 14-16**

Wiring

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



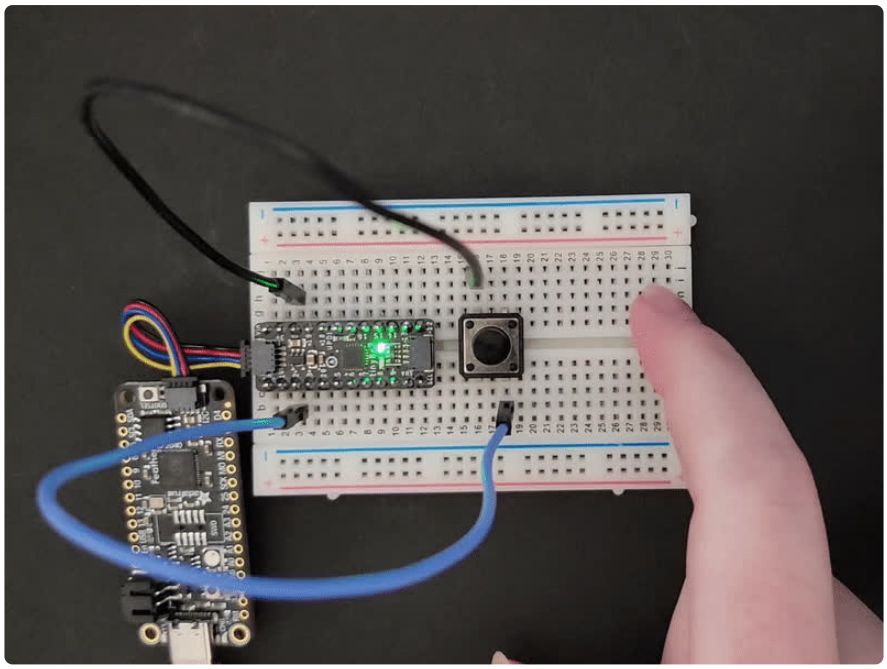
Use a **STEMMA QT cable** to connect the **STEMMA QT connector on the Feather** to the **STEMMA QT connector on the breakout**.
Connect **one leg of button** to **breakout GND**
Connect **opposite leg of button** to **breakout pin 2**

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.digitalio import DigitalIO
from adafruit_seesaw.seesaw import Seesaw

i2c = board.I2C()  # uses board.SCL and board.SDA
# i2c = board.STEMMA_I2C()  # For using the built-in STEMMA QT connector on a
microcontroller
ss = Seesaw(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 ATtinyxxx breakouts!

NeoPixel

The seesaw firmware that ships with the ATtinyxxx breakouts include 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) to get set up.

NeoPixel Pins

Only one pin can be used at a time for powering NeoPixels on the ATtinyxxx breakouts!

There are 15 pins on the ATtiny817 breakout that can be used for powering NeoPixels:

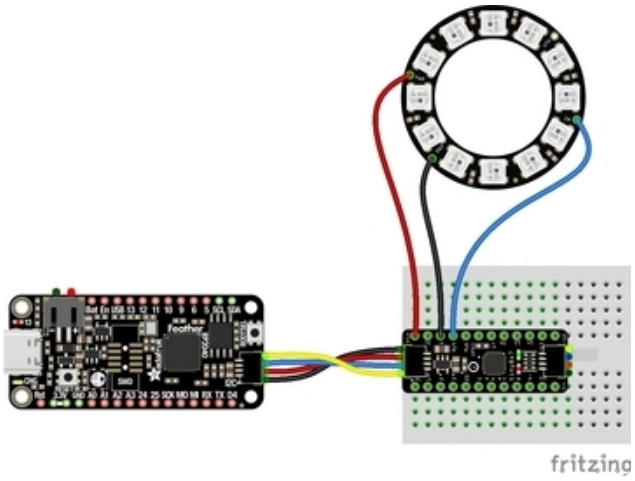
- **0-3, 5-9, 12-14, 18-20**

There are 12 pins on the ATtinyx16 breakouts that can be used for powering NeoPixels:

- **0-6, 8, 11, 14-16**

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**

If you are using an ATtinyx16, there is no pin 19. Connect the NeoPixel IN pin to one of the 12 NeoPixel capable pins listed above.

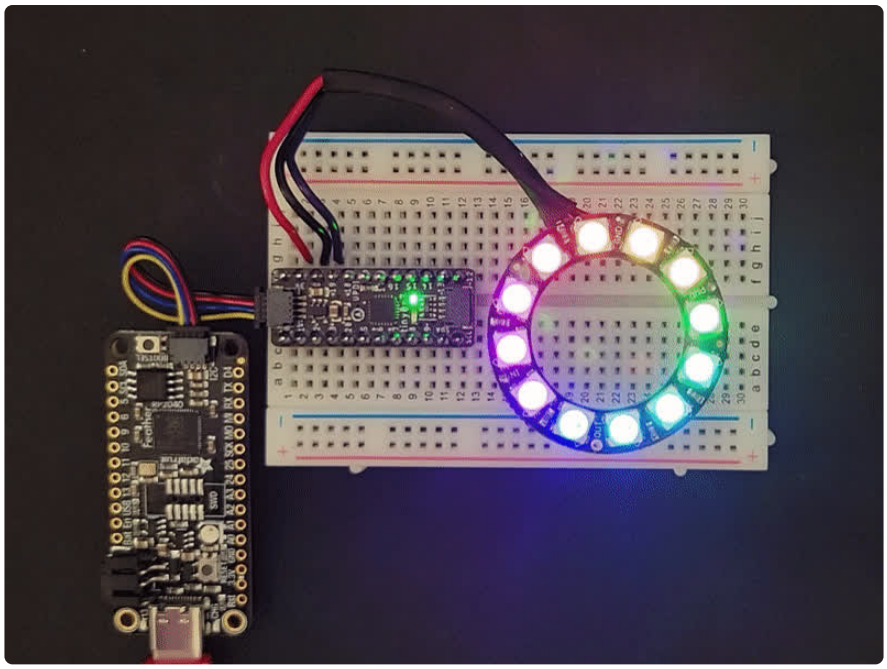
Example Code

Update your **code.py** to the following. If you are using an ATtinyx16, update `NEOPIXEL_PIN` to be one of the NeoPixel capable pins listed above to match your wiring.

```
# 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/neopixel

import time

import board
from rainbowio import colorwheel

from adafruit_seesaw import neopixel, seesaw

i2c = board.I2C()  # uses board.SCL and board.SDA
# i2c = board.STEMMA_I2C()  # For using the built-in STEMMA QT connector on a
microcontroller
ss = seesaw.Seesaw(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 ATtinyxxx breakouts!

PWMOut

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

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

PWM Pins

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

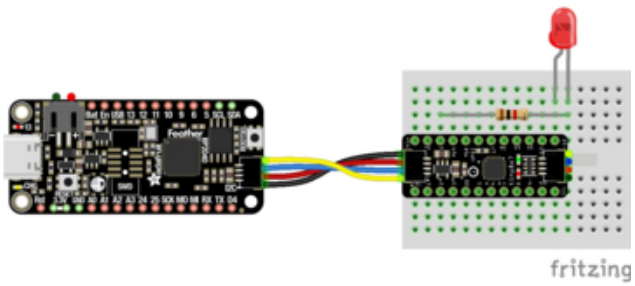
- **0, 1, 9, 12, 13**

The ATtiny816 and ATtiny1616 breakouts with seesaw firmware provides PWM on the following pins:

- **0, 1, 7, 11, 16**

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 **1kΩ resistor**
Connect **1kΩ resistor** to **breakout GND**

If you are using an ATtinyx16, pin 12 is not a PWM pin. Connect the anode (+) pin of the LED to one of the 5 PWM capable pins listed above.

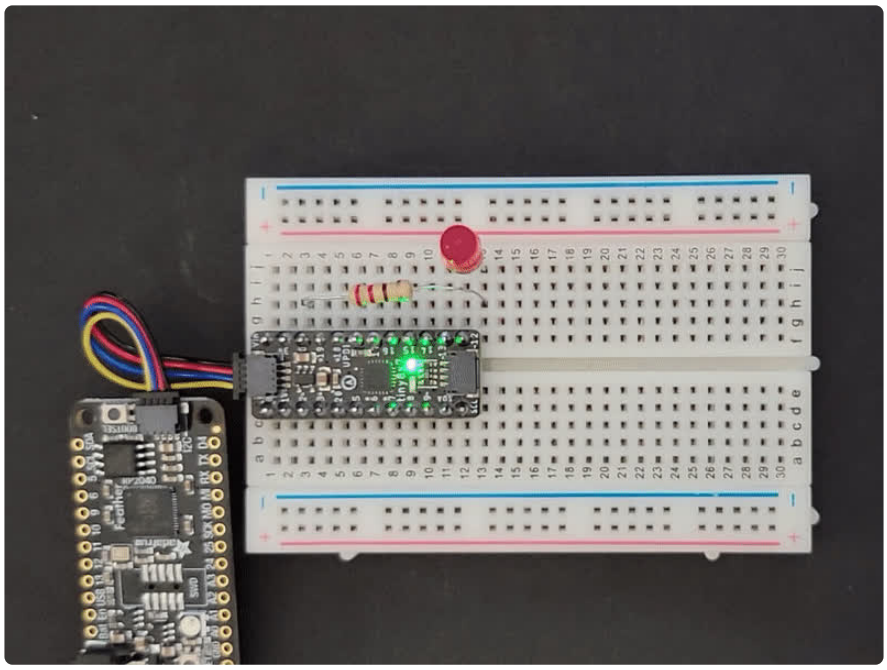
Example Code

Update your **code.py** to the following. If you are using an ATtinyx16, update `PWM_PIN` to be one of the PWM capable pins listed above to match your wiring.

```
# 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 pwmout, seesaw

i2c = board.I2C()  # uses board.SCL and board.SDA
# i2c = board.STEMMA_I2C()  # For using the built-in STEMMA QT connector on a
microcontroller
ss = seesaw.Seesaw(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 0f 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 ATtinyxxx breakouts!

EEPROM

The seesaw firmware that ships with the ATtinyxxx breakouts 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) 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()  # uses board.SCL and board.SDA
# i2c_bus = board.STEMMA_I2C()  # For using the built-in STEMMA QT connector on a
microcontroller
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 disab
le.
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)

Arduino

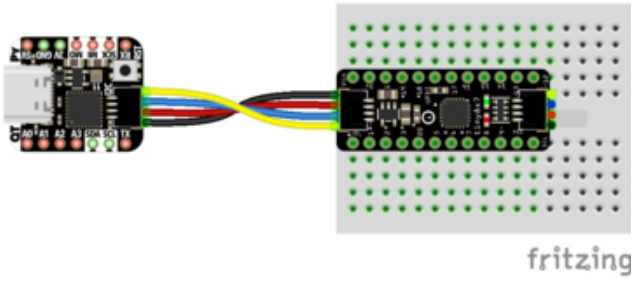
The Adafruit Seesaw library makes it easy to use your ATtinyxxx breakout with Arduino. Install the necessary libraries, and load the examples to use your ATtinyxxx breakout for all sorts of purposes. This example is designed to verify that your breakout is working - it simply blinks the built-in LED.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

> Though the Fritzing diagrams depict an ATtiny817, the examples work as written with the ATtiny816 and ATtiny1616.

Wiring

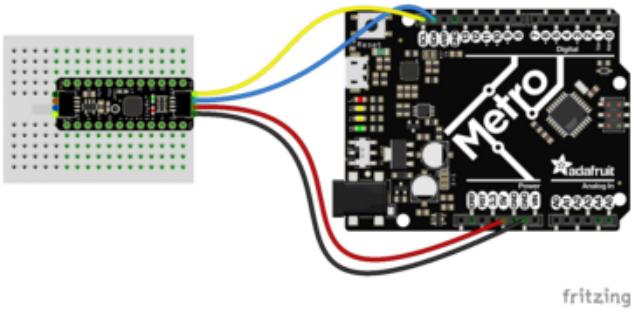
You can use many different Arduino boards, but this example shows the QT Py M0, and the Adafruit Metro. Connect up the ATtiny817 breakout as shown below.

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**.

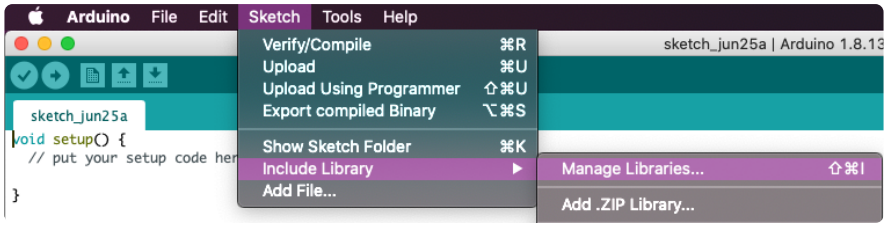
Here is the Metro wired up:



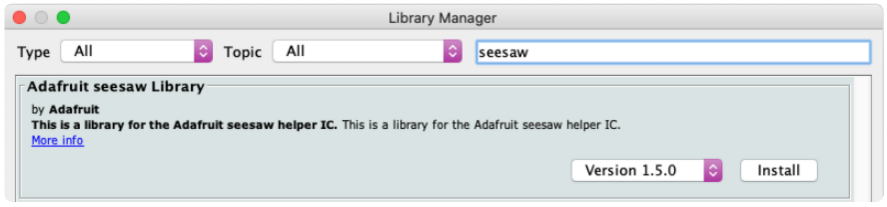
Plug a [STEMMA QT to male jumper wire cable (http://adafru.it/4209)] into the **STEMMA QT connector on the breakout.**
Metro 5V to **breakout VIN (red wire)**
Metro GND to **breakout GND (black wire)**
Metro SCL to **breakout SCL (yellow wire)**
Metro SDA to **breakout SDA (blue wire)**

Library Installation

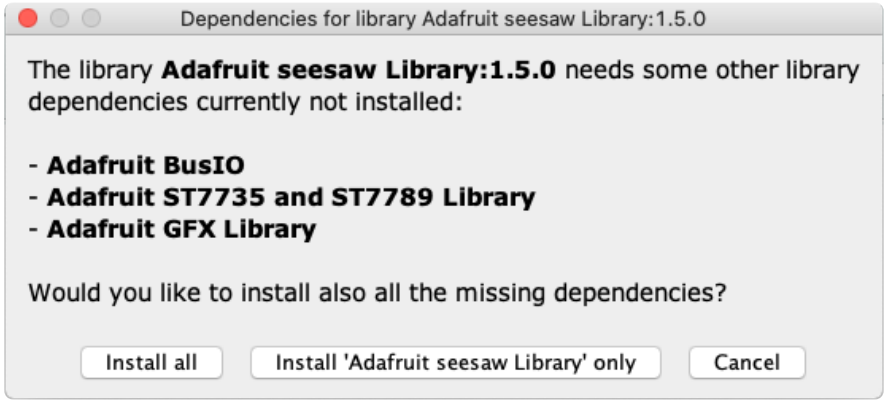
You can install the **Adafruit Seesaw Library** for Arduino using the Library Manager in the Arduino IDE:



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.

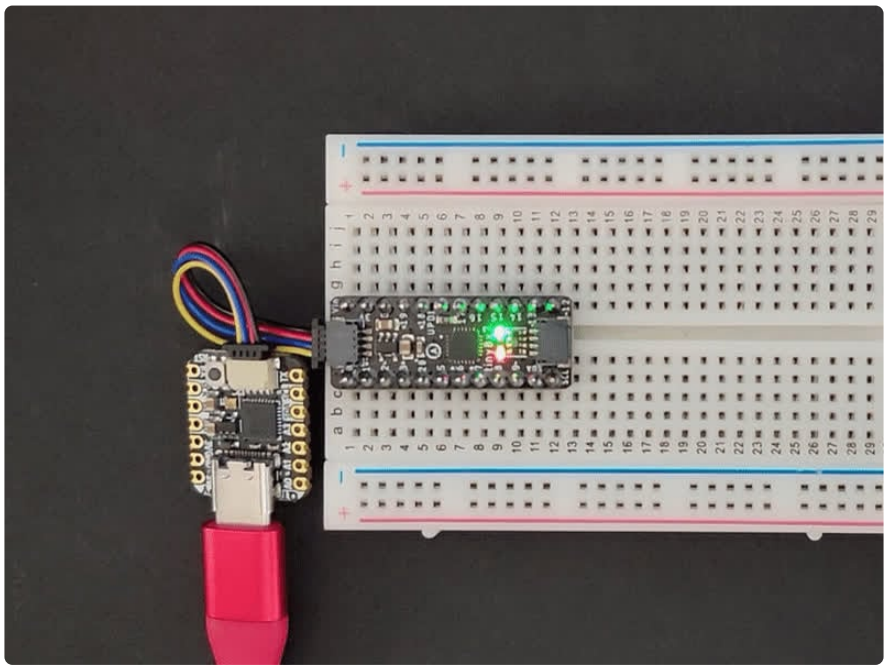
If using the STEMMA QT version of the board, you'll need to use Wire1 like this: Adafruit_seesaw ss(&Wire1) ;

The example below is for the 817. If you are using the 816/1616, please change `#define BLINK_PIN 5` to `#define BLINK_PIN 10`. No other changes are needed.

```
/*
 * 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 ATtinyxxx breakouts 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) to get set up.

Analog Pins

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

- **0-3, 6, 7, 18-20**

The ATtiny816 and ATtiny1616 breakout with seesaw firmware provides analog on the following pins:

- **0-5, 14-16**

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 Attinyxy7, analog in can be 0-3, 6, 7, 18-20
// on Attinyxy6, analog in can be 0-5, 14-16
#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) to get set up.

PWM Pins

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

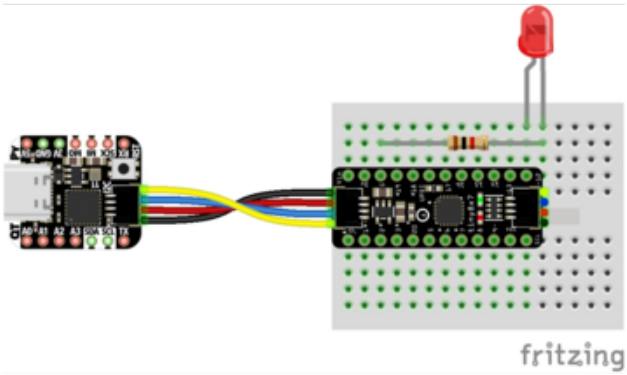
- **0, 1, 9, 12, 13**

The ATtiny816 and ATtiny1616 breakouts with seesaw firmware provides PWM on the following pins:

- **0, 1, 7, 11, 16**

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 **1kΩ resistor**
Connect **1kΩ resistor** to **breakout GND**

If you are using an ATtinyx16, pin 12 is not a PWM pin. Connect the anode (+) pin of the LED to one of the 5 PWM capable pins listed above.

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 ATtinyxxx breakouts.

> This example will not work with the ATtinyxxx breakouts 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 ATtinyxxx, such as pin 1:

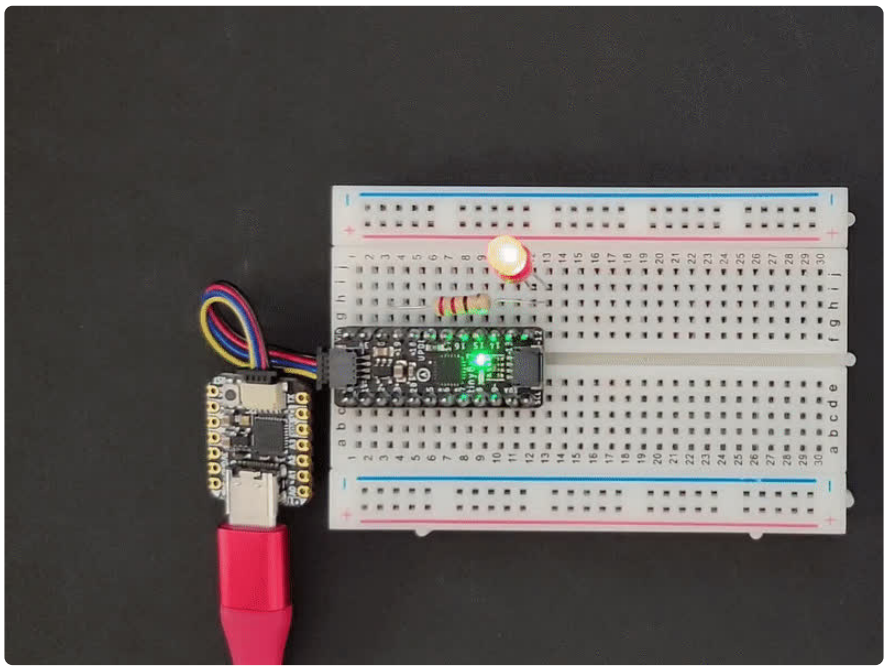
```
int led = 1;           // the PWM pin the LED is attached to
```

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 ATtinyxy7 breakout these are pins 0, 1, 9, 12, 13
  On the ATtinyxy6 breakout these are pins 0, 1, 7, 11, 16
*/

#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 ATtinyxxx breakouts 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) to get set up.

NeoPixel Pins

Only one pin can be used at a time for powering NeoPixels on the ATtinyxxx breakouts!

There are 15 pins on the ATtiny817 breakout that can be used for powering NeoPixels:

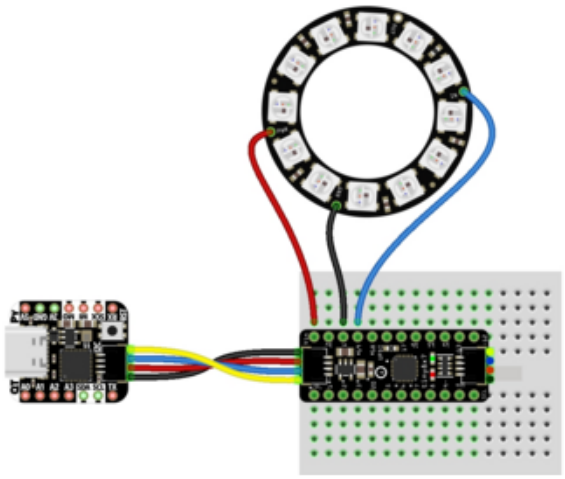
- **0-3, 5-9, 12-14, 18-20**

There are 12 pins on the ATtinyx16 breakouts that can be used for powering NeoPixels:

- **0-6, 8, 11, 14-16**

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**

> If you are using an ATtinyx16, there is no pin 19. Connect the NeoPixel IN pin to one of the 12 NeoPixel capable pins listed above.

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 ATtinyxxx breakout.

> This example will not work with the ATtinyxxx 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 ATtinyxxx, such as pin 8:

`#define PIN 8`

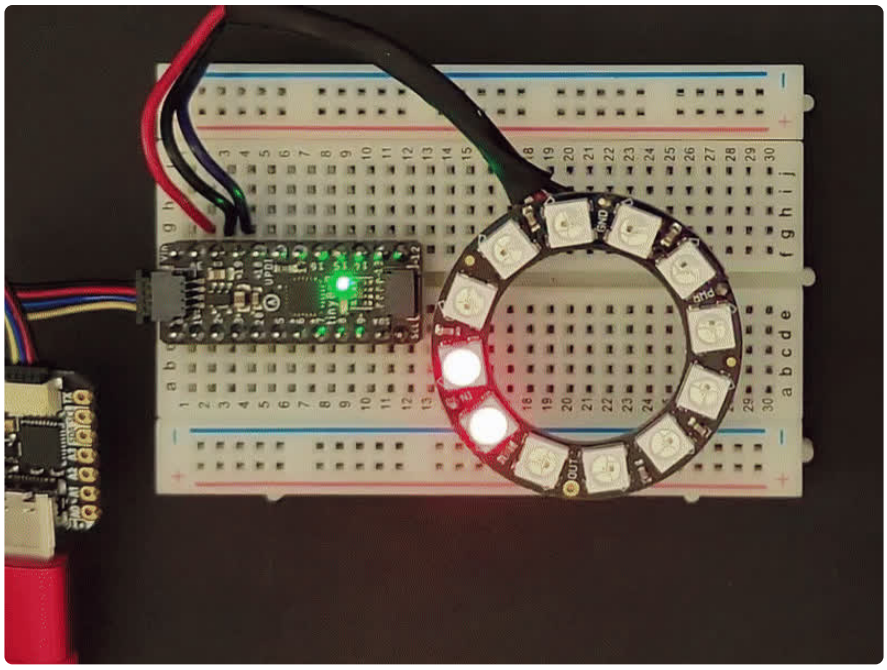
You must do this before uploading the sketch, or it won't work with your setup!

```
#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 ATtinyxxx breakouts 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) 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.

Repo Summary

There are numerous code repositories with "seesaw" in the name. The firmware repositories contain the code running on the seesaw device itself. The support libraries are used to interact with *any* seesaw device.

If you are an end user of a seesaw device, then it is the support libraries you want.

There is no support for custom firmware development.

Firmware

- seesaw (https://adafru.it/D1-) - for SAMD based boards
- Adafruit_seesawPeripheral (https://adafru.it/VdL) - for ATtiny 1-series based boards

Support Libraries

- Adafruit_CircuitPython_seesaw (https://adafru.it/BrW) - for use with CircuitPython or Blinka
- Adafruit_Seesaw (https://adafru.it/BrV) - for use with Arduino

---

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.

The I2C bus on the SAMD09 is 3.3V logic level, but all boards other than the SAMD09 breakout have level shifting so you can use 3 or 5V logic. Boards with Attiny chips are 3 or 5V safe so you can use either for power and logic

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 theSeeSaw breakouts we use **0x49**) but other boards will have a different base address. Check the board documentation for the default base I2C address, or you can plug it in and scan the I2C bus to find it (https://adafru.it/YEf).

This address can be modified using the address select pins, there can be multiple address pins. If address select pin 0 (**A0**) is tied to ground on boot, the I2C address is incremented by 1. If address select pin 1 (**A1**) 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

On both the SAMD09 and Attiny817 breakouts, the default A0 pin is **16**, default A1 pin is **17**. On boards where the chips are embedded, there may be as many as 4 address pins, they'll be labeled with jumpers

The base 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.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

Base Register Summary

The following table summarizes the **module base registers addresses**. Further details about the function registers associated for each base register are covered in later sections.

Not all seesaw products implement all of these features.

Base Register Address	Module
0x00	Status
0x01	GPIO
0x02 - 0x07	SERCOM
0x08	PWM
0x09	ADC
0x0A	DAC
0x0B	Interrupt
0x0C	DAP
0x0D	EEPROM
0x0E	NeoPixel

0x0F	Touch
0x10	Keypad
0x11	Encoder

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&lt;num; i++){
    buf[i] = Wire.read();
  }
}
```

GPIO

The GPIO module provides every day input and outputs. You'll get 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.

On SAMD09-based boards the GPIO is 3V only. On ATtiny-based boards, the GPIO logic is whatever the power pin is, 3V or 5V.

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 on SAMD09:

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

GPIO register setup on ATTiny8x7:

([this is the same as the megaTinyCore pin mapping](https://adafru.it/YEh) (https://adafru.it/YEh))

- Bit 0: PA4
- Bit 1: PA5
- Bit 2: PA6
- Bit 3: PA7
- Bit 4: PB7
- Bit 5: PB6
- Bit 6: PB5
- Bit 7: PB4
- Bit 8: PB3
- Bit 9: PB2
- Bit 10: PB1
- Bit 11: PB0
- Bit 12: PC0
- Bit 13: PC1
- Bit 14: PC2
- Bit 15: PC3
- Bit 16: PC4
- Bit 17: PC5
- Bit 18: PA1
- Bit 19: PA2

- Bit 20: PA3

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 of the seesaw device. On the Attiny series please wait at least 250uS between command write and data read to allow the data to be read and formatted for retrieval. **Reading this register will also reset the IRQ pin if it was configured.**

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. **This will also reset the IRQ pin if it was configured.**

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 SAMD09 seesaw has 4 ADC inputs, the Attiny8x7 has 11 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 uS 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.

SAMD09 ADC channels are:

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

ATtiny8x7 ADC channels are:

- Channel 0: PA4
- Channel 1: PA5
- Channel 2: PA6
- Channel 3: PA7
- Channel 6: PB5
- Channel 7: PB4
- Channel 10: PB1
- Channel 11: PB0
- Channel 18: PA1
- Channel 19: PA2
- Channel 20: PA3

(These are the same as the Arduino GPIO pin names for the ADCs in megaTinyCore (https://adafru.it/YEh))

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
...	...	...	...
0x1B	CHANNEL_20	16-bit	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

CHANNEL_1 (0x08, 16bits, Read Only)

ADC value for channel 1

CHANNEL_2 (0x09, 16bits, Read Only)

ADC value for channel 2

CHANNEL_3 (0x0A, 16bits, Read Only)

ADC value for channel 3

...

CHANNEL_20 (0x1B, 16bits, Read Only)

ADC value for channel 20

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 inconsistant 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 170 RGB or 127 RGBW pixels. The output pin as well as the communication protocol frequency are configurable. Note: older firmware is limited to 63 pixels max.

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 LE, 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. 2 Bytes, little endian order

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

0x08	**Base Register Address**

This module provides support for Pulse Width Modulation (PWM) output.

Nomenclature

The seesaw PWM Function Registers are defined using a generic nomenclature as follows:

- **PWM Number** - Specifies a specific PWM output (pin).
- **PWM Value** - Specifies the PWM **duty cycle**.
- **PWM Frequency** - Specifies the PWM **frequency**.

See the Port Specific Details section below for further information.

Function Register Summary

Register Address	Register Name	Register Size	Access
0x01	PWM_VAL	3 bytes	W
0x02	PWM_FREQ	3 bytes	W

Function Register Description

0x01 - PWM_VAL

Byte 0	Byte 1	Byte 2
PWM Number	PWM Value MSB	PWM Value LSB

Sets the **PWM Value** for a specified **PWM Number**. The first byte written is the **PWM Number**. The next two bytes are the 16 bit **PWM Value**, most significant byte (MSB) followed by least significant byte (LSB).

0x02 - PWM_FREQ

Byte 0	Byte 1	Byte 2
PWM Number	PWM Frequency MSB	PWM Frequency LSB

Sets the **PWM Frequency** for a specified **PWM Number**. The first byte written is the **PWM Number**. The next two bytes are the 16 bit **PWM Frequency**, most significant byte (MSB) followed by least significant byte (LSB).

Port Specific Details

SAMD

PWM outputs are available on pins PA04, PA05, PA06, and PA07. The **PWM Number** for each is shown in the table below.

PWM Number	Output Pin
0	PA04

1	PA05
2	PA06
3	PA07

The full 16 bit **PWM Value** is used. This value should be an unsigned integer ranging from 0 for full off to 65535 for full on.

The **PWM Frequency** is a 16 bit unsigned integer value which specifies the frequency in hertz (Hz).

ATtiny

The **PWM Number** is the Arduino GPIO pin number.

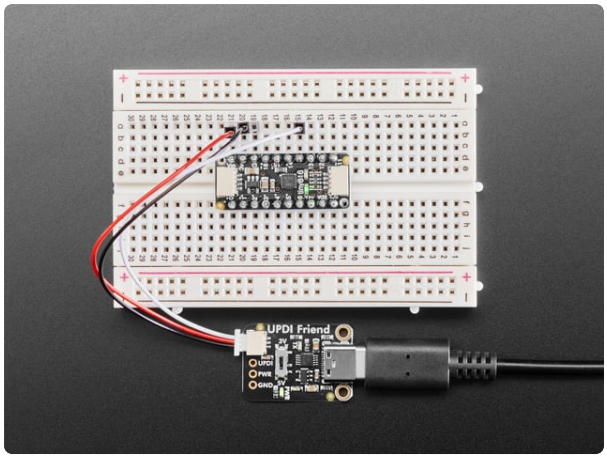
Currently, only the MSB of the 16 bit **PWM Value** is used. This is due to the 8 bit limit of analogWrite() used internally. However, a full 16 bit value should be sent from 0 for full off to 65535 for full on.

The **PWM Frequency** is a 16 bit unsigned integer value which specifies the frequency in hertz (Hz). (**NOTE**: uses tone() internally)

Advanced: Reprogramming with UPDI

> We don't provide any support for custom builds of seesaw - but we think this information is cool and useful for the Maker community!

You can reprogram these ATtiny breakouts to run your own firmware. However, 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! You can build this setup with a 1K resistor and a USB to Serial cable or with the UPDI Friend board.



[Adafruit UPDI Friend - USB Serial UPDI Programmer]

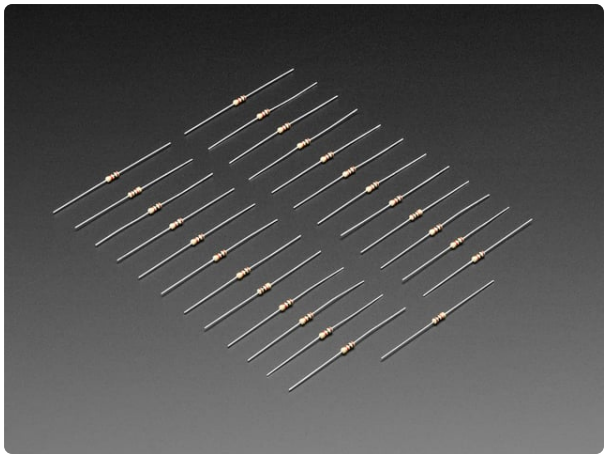
UPDI stands for

https://www.adafruit.com/product/5879

[USB to TTL Serial Cable - Debug / Console Cable for Raspberry Pi]

The cable is easiest way ever to connect to your microcontroller/Raspberry Pi/WiFi router serial console port. Inside the big USB plug is a USB<->Serial conversion chip and at...

https://www.adafruit.com/product/954

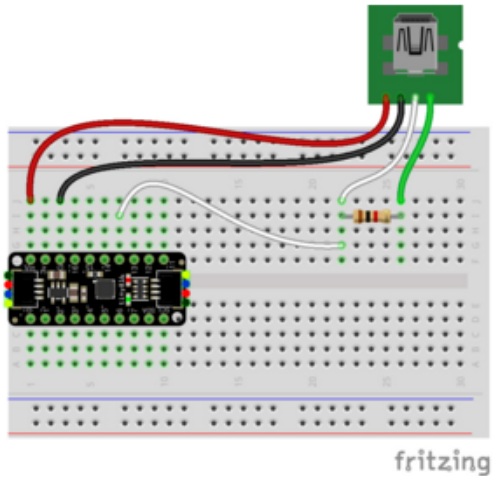


[Through-Hole Resistors - 1.0K ohm 5% 1/4W - Pack of 25]

ΩMG! You're not going to be able to resist these handy resistor packs! Well, axially, they do all of the resisting for you!This is a 25 Pack of...

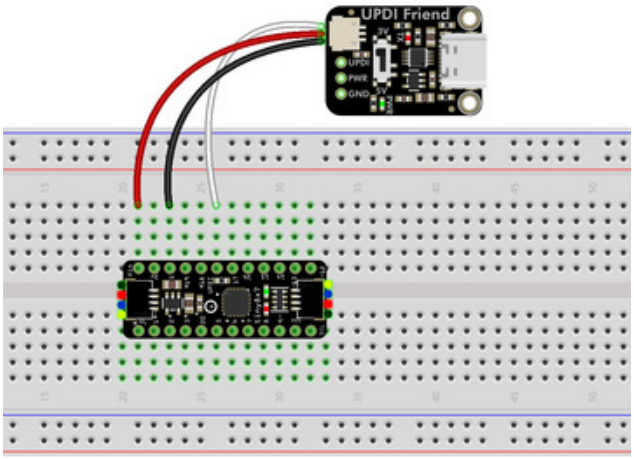
https://www.adafruit.com/product/4294

Building a UPDI Programmer



USB to Serial cable power to **ATtiny VIN (red wire)**
USB to Serial cable ground to **ATtiny G (black wire)**
USB to Serial cable RX to **1K resistor (white wire)**
USB to Serial cable TX to **1K resistor (green wire)**
1K resistor to **ATtiny UPDI pin (white wire)**

Wiring with the UPDI Friend



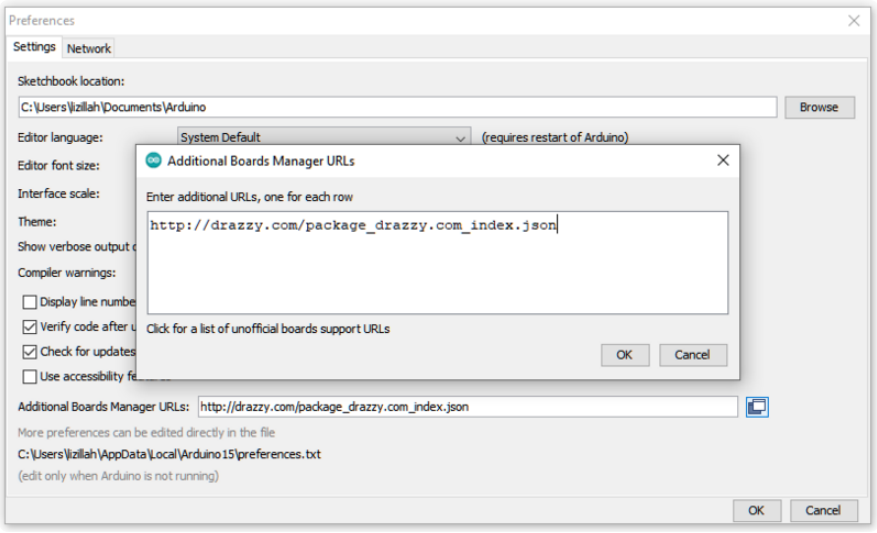
UPDI Friend PWR to **ATtiny VIN (red wire)**
UPDI Friend GND to **ATtiny G (black wire)**
UPDI Friend UPDI to **ATtiny UPDI pin (white wire)**

Install megaTinyCore

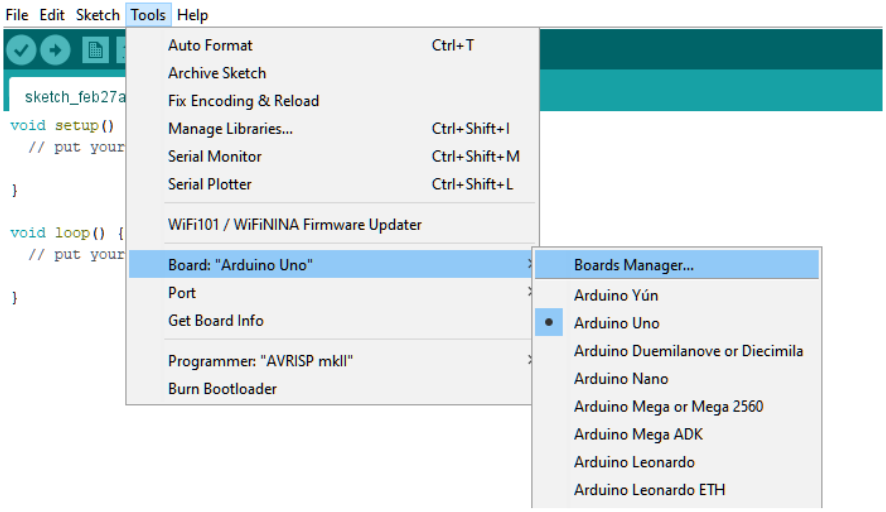
You can compile code for the ATtiny using the megaTinyCore (https://adafru.it/VdM) board support package in the Arduino IDE. There are detailed installation instructions (https://adafru.it/19e4) in the megaTinyCore GitHub repository. The following steps outline how to install it using the Boards Manager.

In the Arduino IDE, go to **Preferences** and add the megaTinyCore boards manager URL to the **Additional Boards Manager URLs list**:

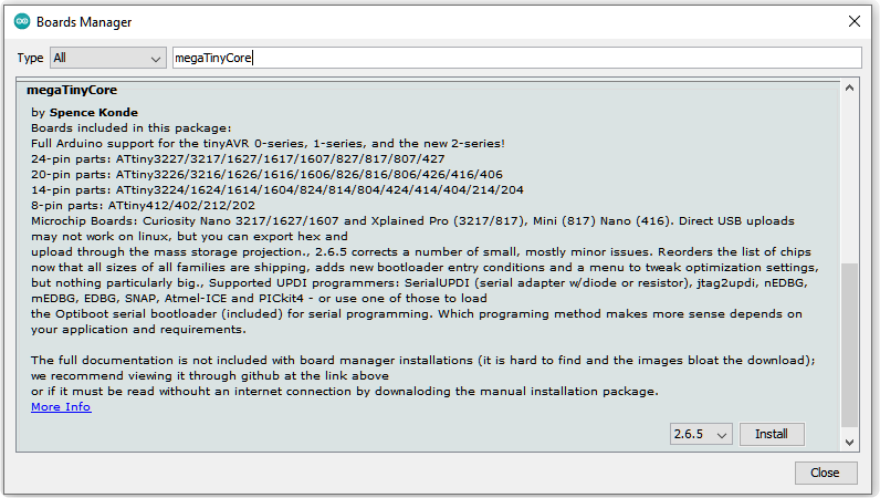
```
http://drazzy.com/package_drazzy.com_index.json
```



Go to **Tools** - **Board** - **Boards Manager...**

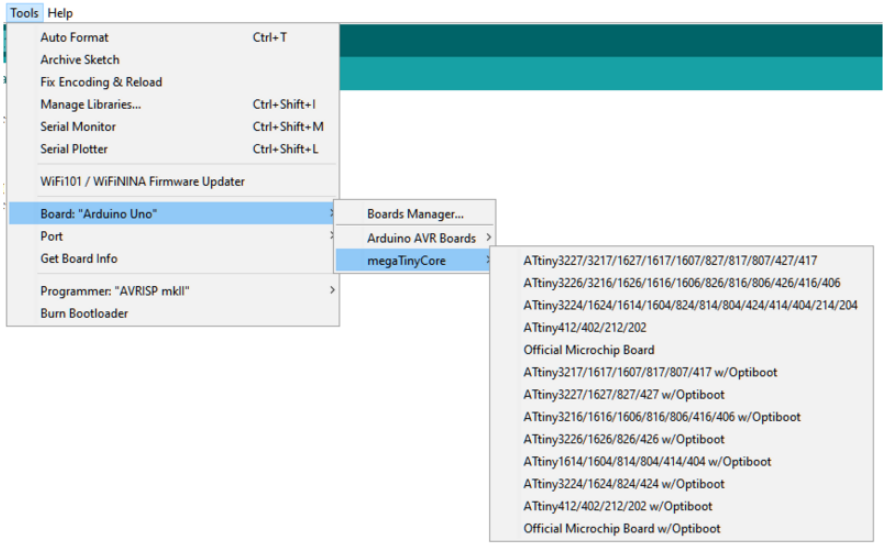


In the **Boards Manager**, search for **megaTinyCore**. Click **Install** to install the board support package.

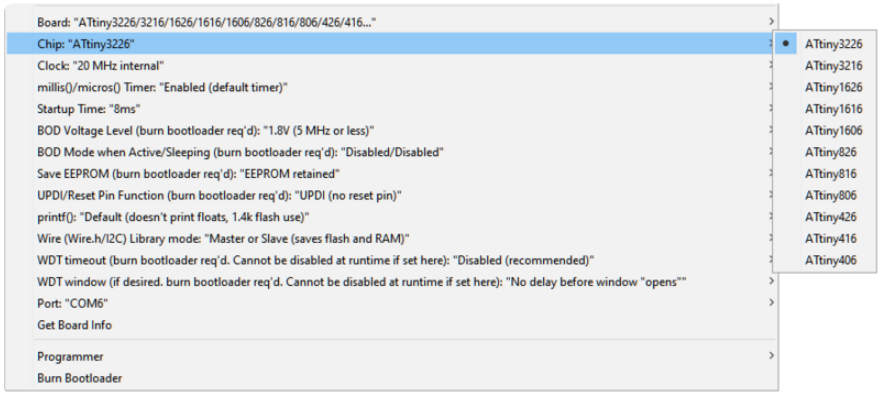


Program the ATtiny

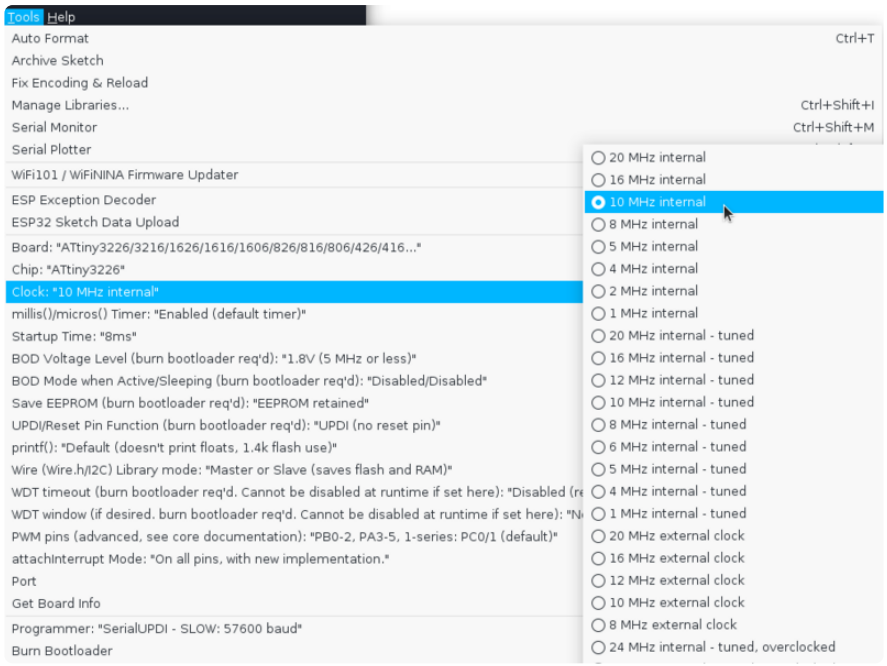
After the megaTinyCore finishes installing, go to **Tools - Board - megaTinyCore** and **select the board list** that includes your ATtiny.



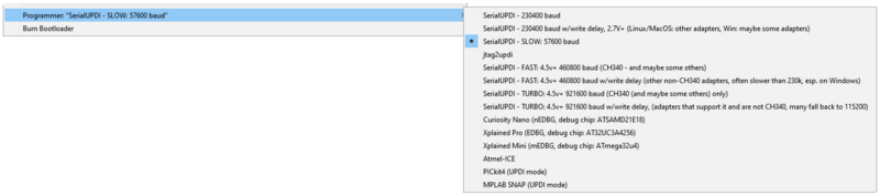
Then, **select the chip** of your ATtiny.



Next, **set clock** if needed. The default 20 MHz clock option is only valid if powering with 5V. If powering with 3.3V, select 10 MHz for the clock.

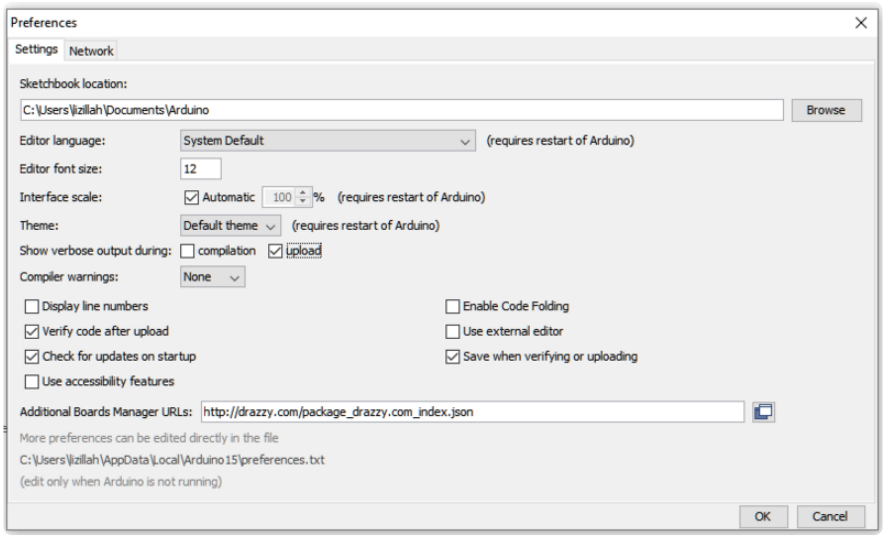


Under **Programmer**, select **SerialUPDI - SLOW: 57600 baud**.

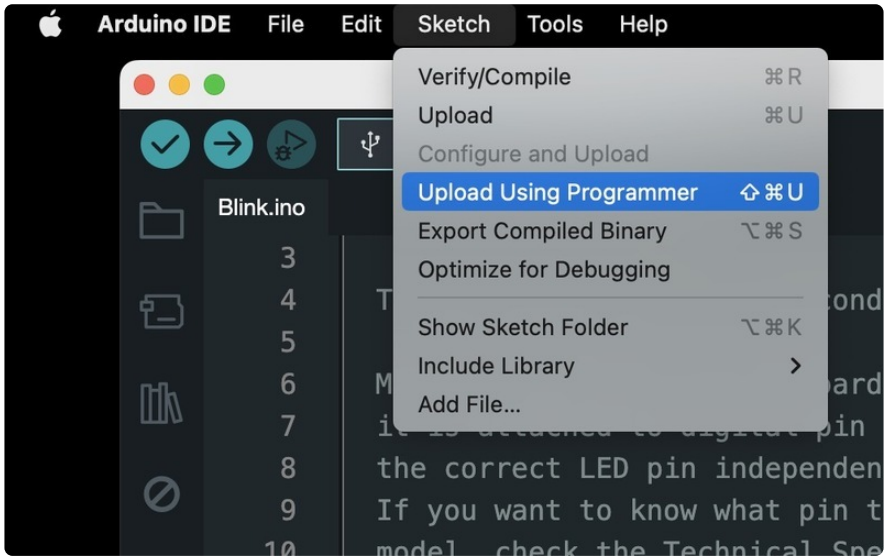


Other options can be left at their defaults.

Finally, go to **Preferences** and **check ON Show verbose output during upload**. This will give you the upload details and progress in the monitor at the bottom of the Arduino IDE, which is very useful for debugging.



Now you can compile code with the megaTinyCore to upload to the ATtiny with the UPDI programmer by going to **Sketch** > **Upload Using Programmer**.



Blink Test

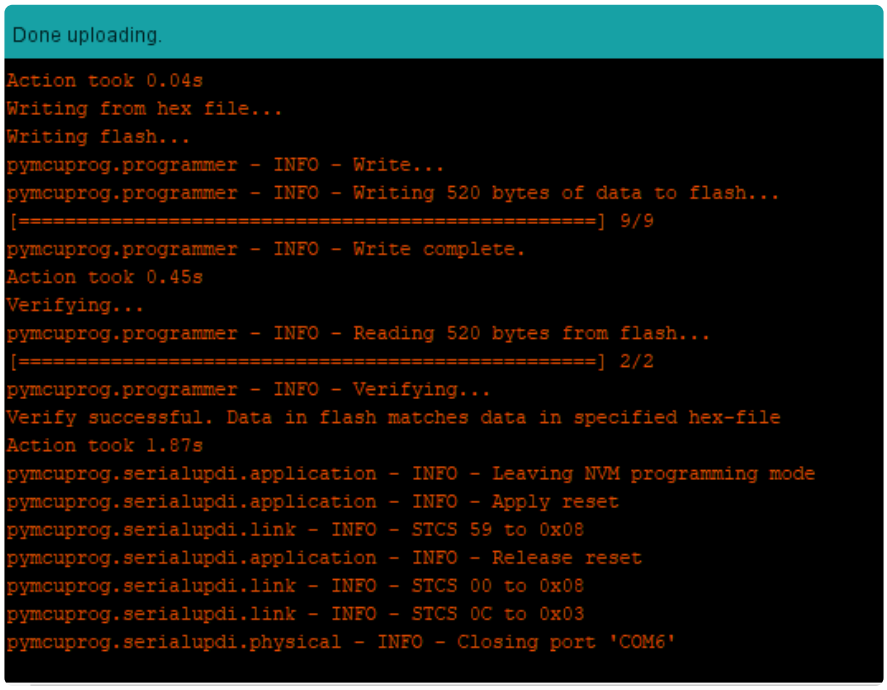
This simple example blinks the ATTiny816 onboard red indicator LED on pin 10.

```
void setup() {
  pinMode(10, OUTPUT);
}

void loop() {
  digitalWrite(10, HIGH);
  delay(1000);
  digitalWrite(10, LOW);
  delay(1000);
}
```

If using this with the ATTiny817, you'll want to update the pin to 5, as shown below.

```
void setup() {
  pinMode(5, OUTPUT);
}

void loop() {
  digitalWrite(5, HIGH);
  delay(1000);
  digitalWrite(5, LOW);
  delay(1000);
}
```

Upload the sketch using the UPDI programmer. You should see this output in the monitor after the upload has completed successfully:



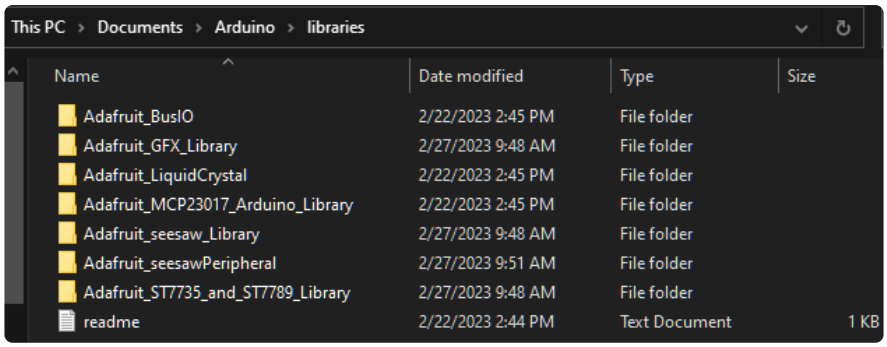
The red LED on the ATtiny should be blinking!

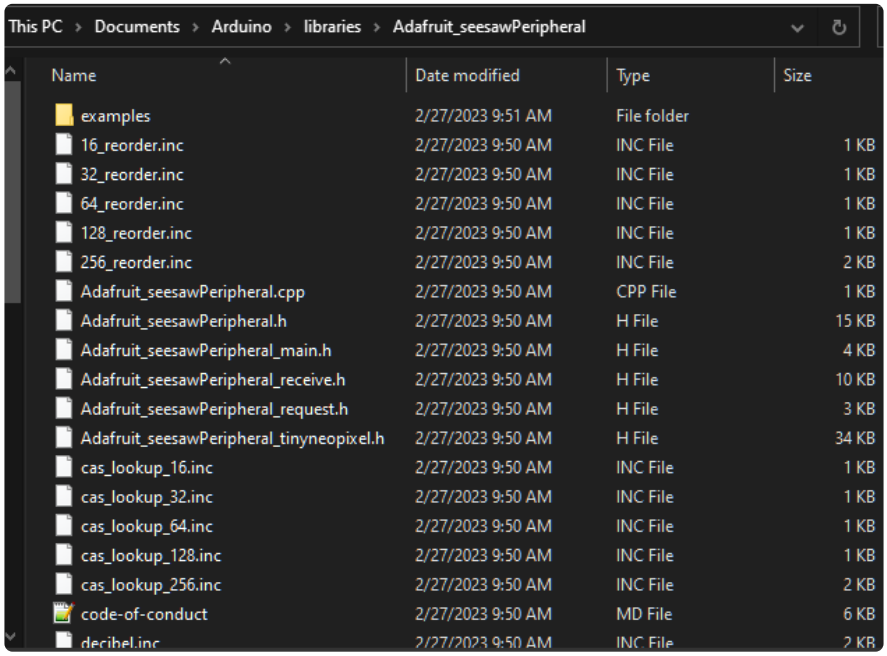
Reloading the seesaw Firmware

You can reinstall the default seesaw Firmware using the UPDI programmer, megaTinyCore (https://adafru.it/VdM) and the Adafruit_seesawPeripheral library (https://adafru.it/VdL).

Install the Libraries

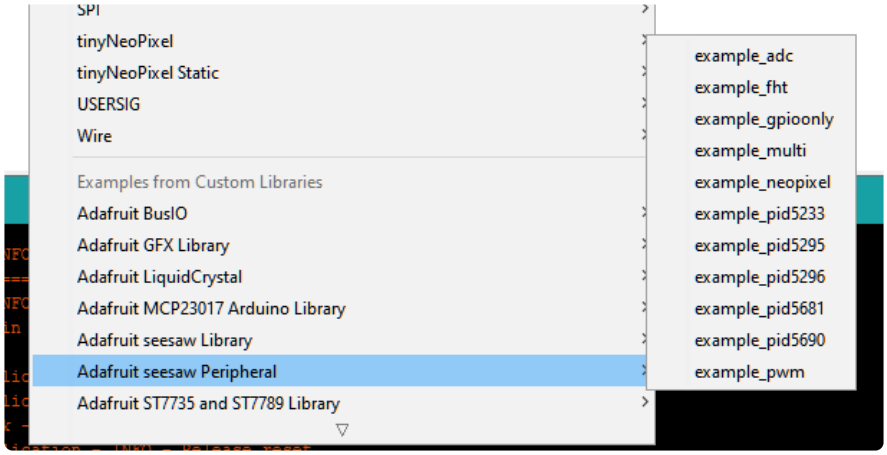
Install the Adafruit_seesawPeripheral library (https://adafru.it/VdL), Adafruit_BusIO library (https://adafru.it/GxD) and the Adafruit_Seesaw library (https://adafru.it/BrV) by cloning the repository and adding the folder to your Arduino **libraries** folder on your computer.





Upload the seesaw Firmware

In the Arduino IDE, go to **File - Examples - Adafruit seesaw Peripheral**.



The `example_pid####` sketches contain the default seesaw firmware for each seesaw breakout. The `pid####` portion of the name corresponds to the Adafruit product ID of the board. For example, to upload the firmware to the ATtiny816 breakout, you would select `example_pid5681`.

After selecting the corresponding firmware sketch, upload it to your board using the UPDI programmer. You can confirm that the seesaw firmware was successfully uploaded by connecting the ATtiny breakout to a board via I2C and performing an I2C scan. The address `0x49` should appear.

---

megaTinyCore Docs

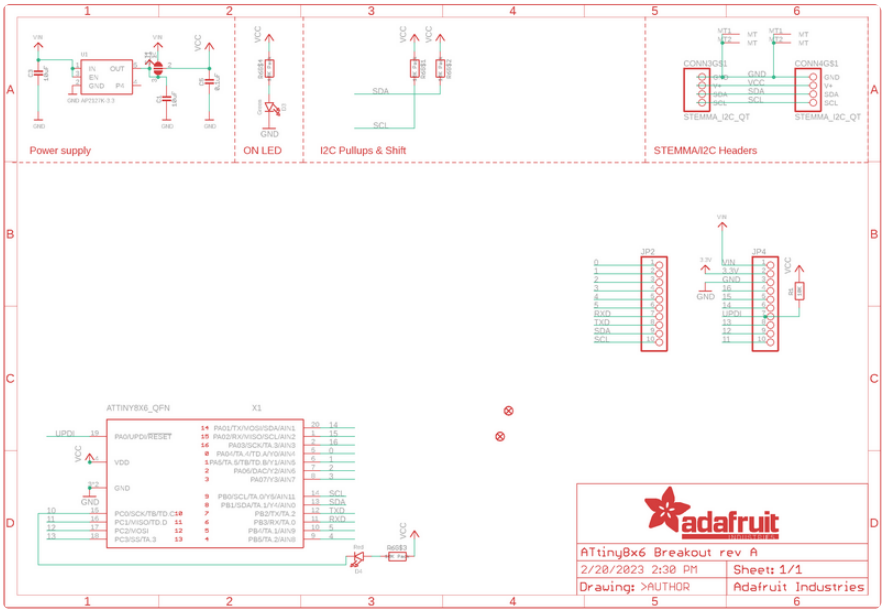
megaTinyCore Docs (https://adafru.it/VdM)

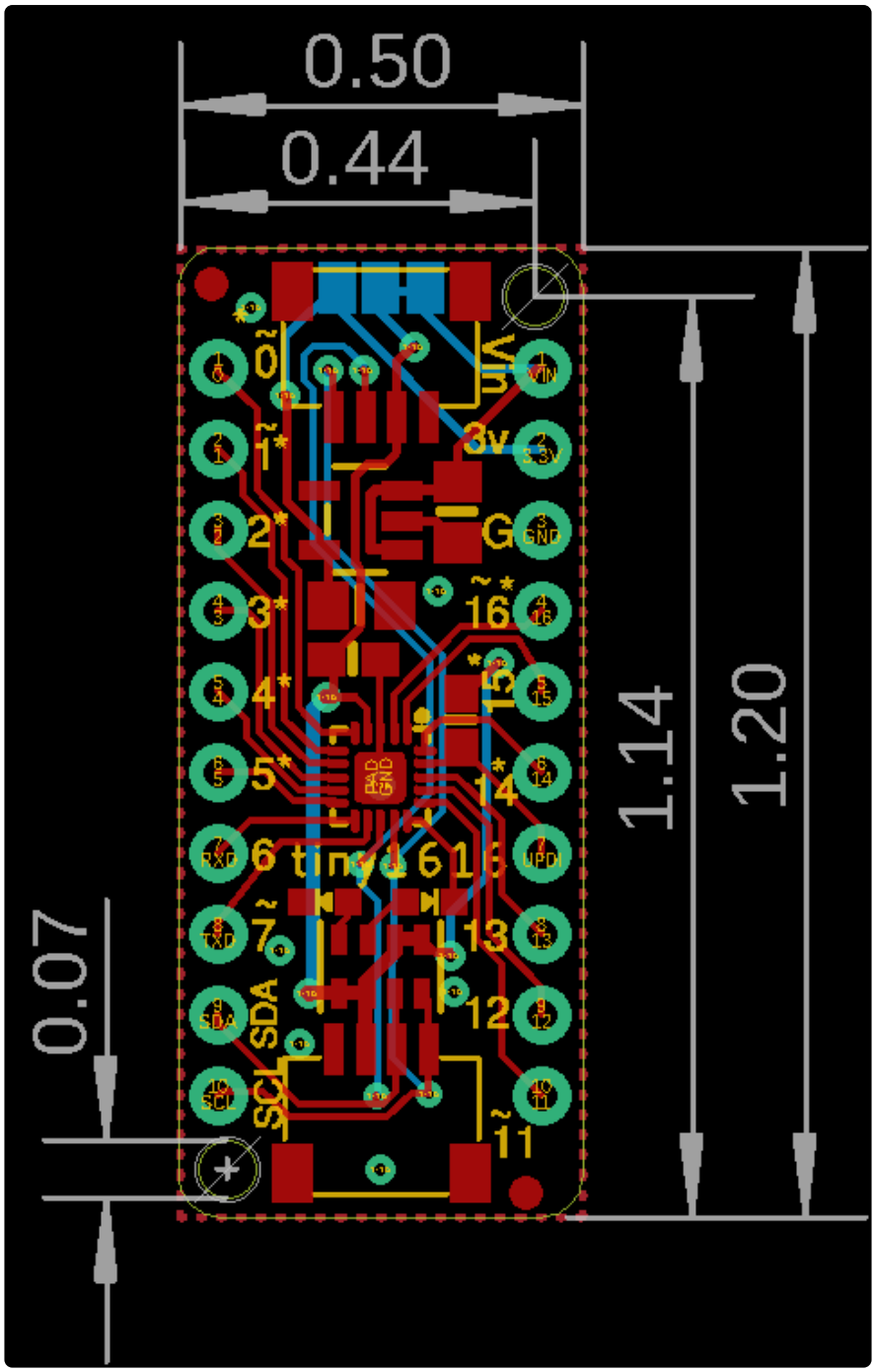
Downloads

Files

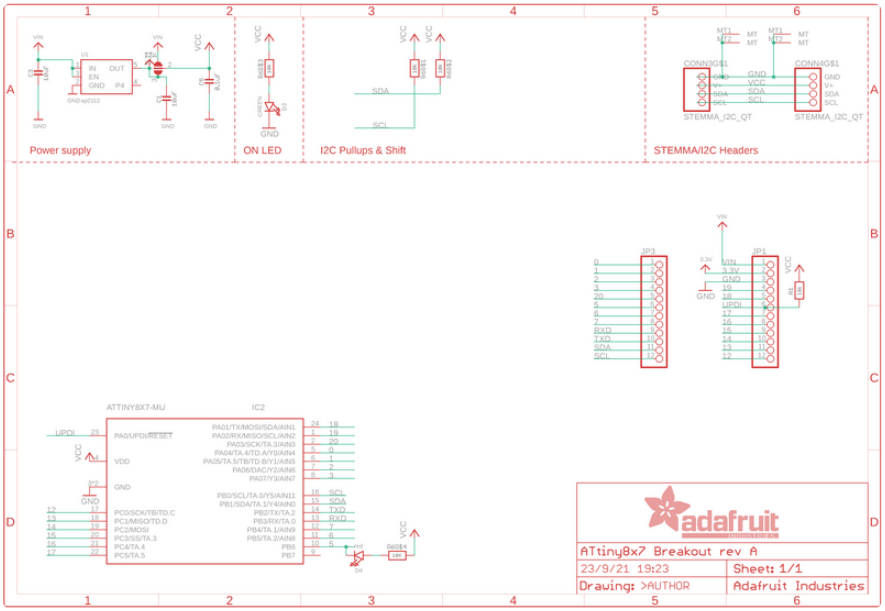
- ATtiny8x7 datasheet (https://adafru.it/VhF)
- ATtiny816 Datasheet (https://adafru.it/18EI)
- ATtiny1616 Datasheet (https://adafru.it/18HB)
- ATtiny817 EagleCAD PCB files on GitHub (https://adafru.it/Via)
- ATtinyx16 EagleCAD PCB files on GitHub (https://adafru.it/19e5)
- ATtiny817 Fritzing object in the Adafruit Fritzing Library (https://adafru.it/Vib)
- ATtinyx16 Fritzing object in the Adafruit Fritzing Library (https://adafru.it/19e6)

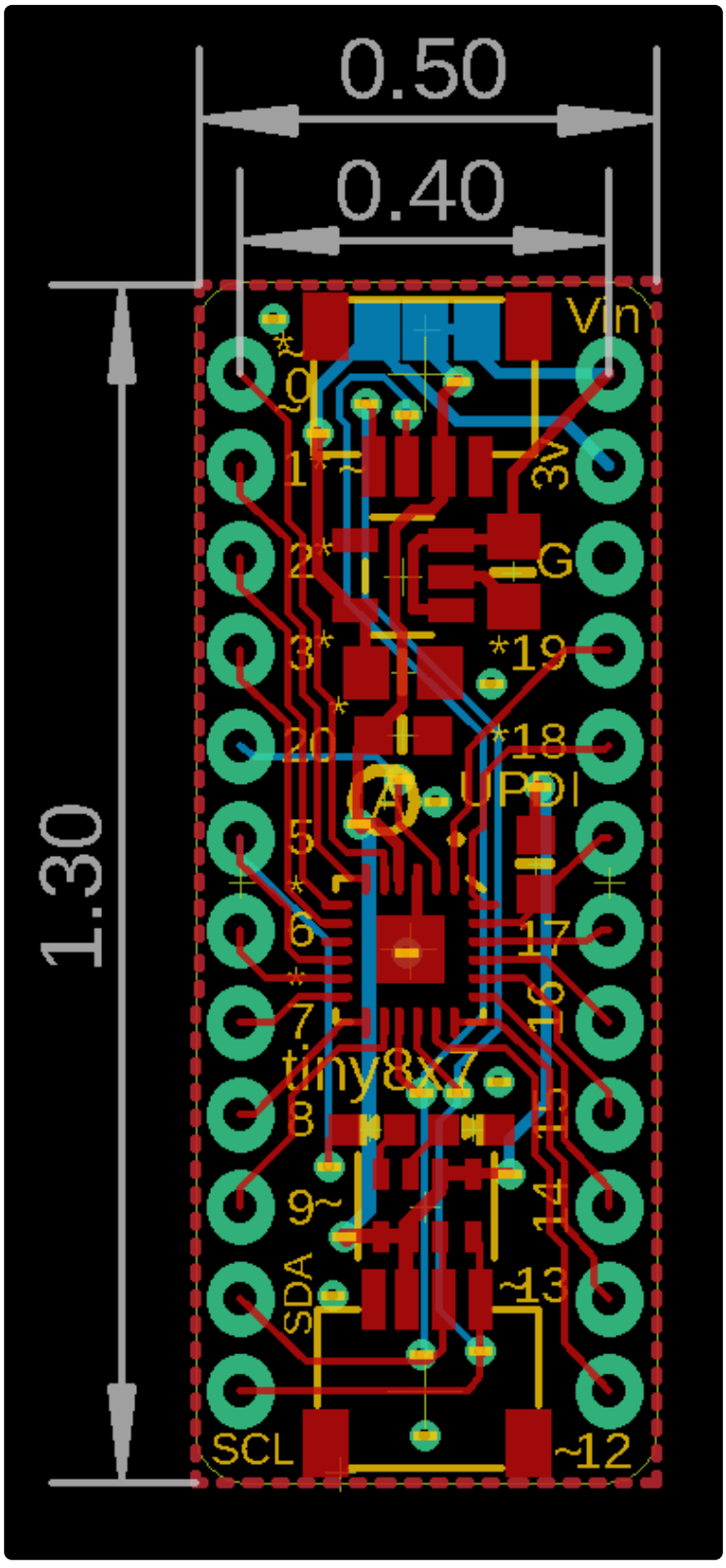
ATtinyx16 Breakout Schematic and Fab Print




0.50
0.44
1.14
1.20
0.07
VIN
3V
G
SDA
SCL
tiny1616

ATtiny817 Breakout Schematic and Fab Print




0.50
0.40
1.30
Vin
3V
G
*19
*18
PDI
17
16
15
14
~13
~12
SCL
SDA
9~
8
7
6
5
20
3*
2*
1~
tiny8x7