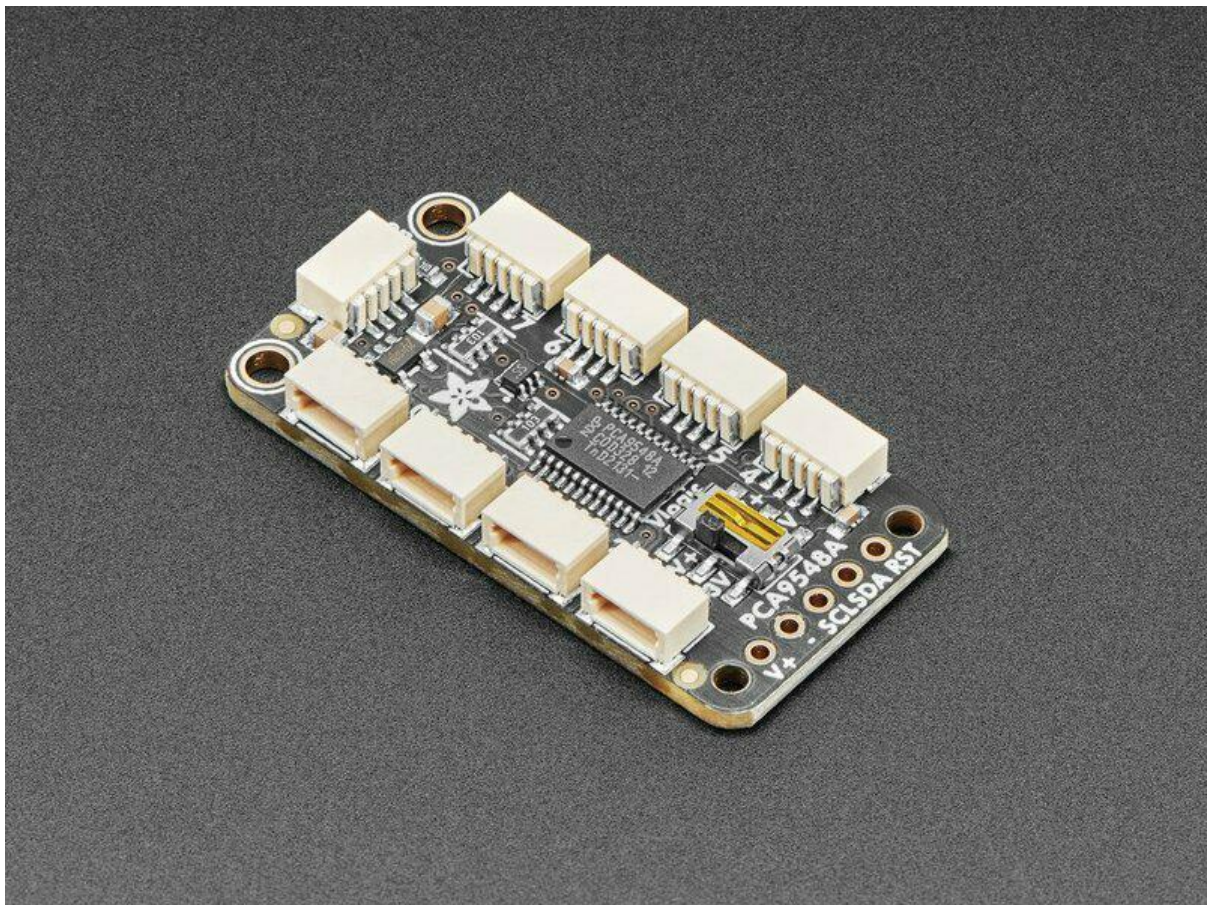




Adafruit PCA9548 8-Channel STEMMA QT / Qwiic I2C Multiplexer

Created by Liz Clark



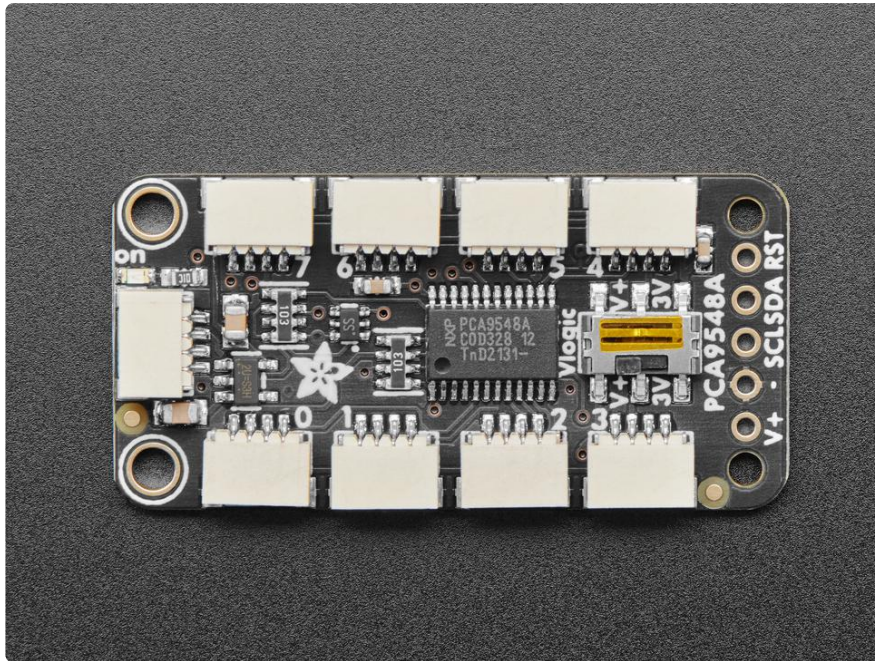
<https://learn.adafruit.com/adafruit-pca9548-8-channel-stemma-qt-qwiic-i2c-multiplexer>

Last updated on 2022-11-09 11:13:32 AM EST

Table of Contents

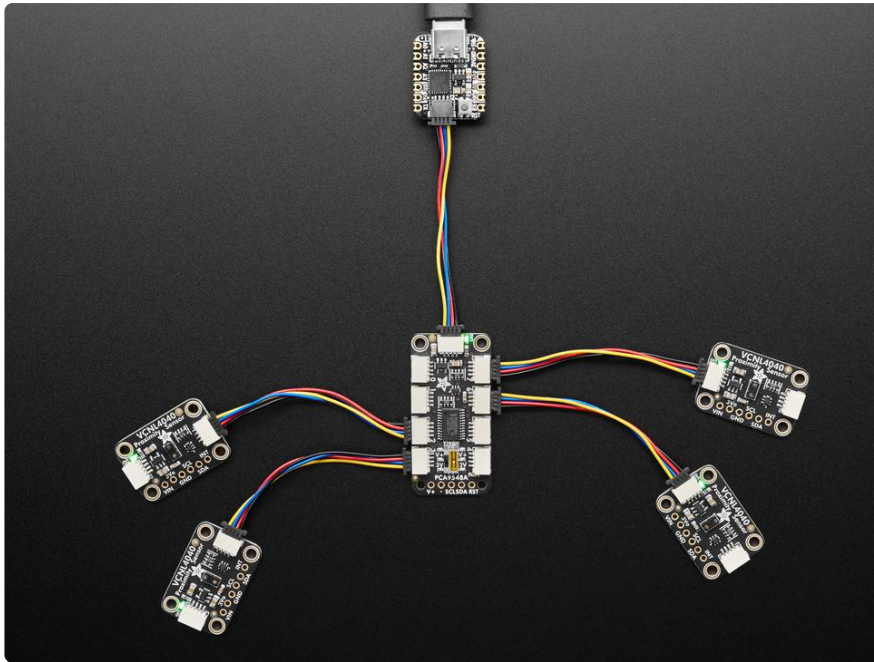
Overview	3
Pinouts	6
<ul style="list-style-type: none">• Power Pins• Vlogic Switch• I2C Logic Pins - Control• I2C Logic Pins - Multiplexed• Address Pins• Reset Pin• Power LED	
CircuitPython & Python	9
<ul style="list-style-type: none">• Why the Adafruit_CircuitPython_TCA9548A Module?• CircuitPython Microcontroller Wiring• Python Computer Wiring• Python Installation of TCA9548A Library• CircuitPython Usage• Python Usage• Simple Test Example Code• Multi-Sensor Example Code	
Python Docs	15
Arduino	15
<ul style="list-style-type: none">• Wiring• Library Installation• I2C Scanner Example Code• Multi-Sensor Example Code	
Downloads	20
<ul style="list-style-type: none">• Files• Schematic and Fab Print	

Overview

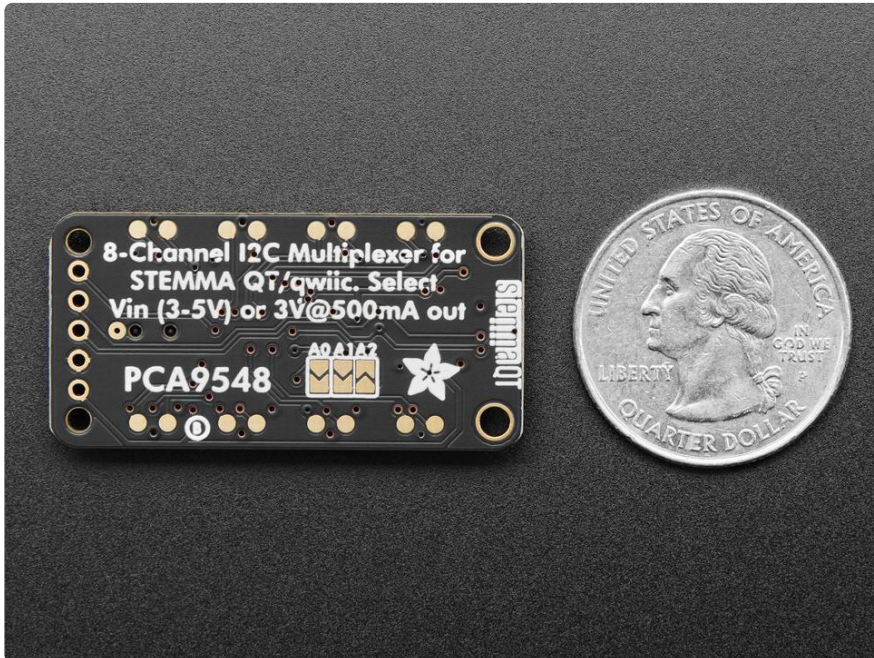


You just found the perfect I2C sensor, available in a handy chainable [Qwiic, or STEMMA QT \(\)](#) package, and you want to wire up two or three or more of them to your microcontroller when you realize "Uh oh, this chip has a fixed I2C address, and from what I know about I2C, you cannot have two devices with the same address on the same SDA/SCL pins!" Are you out of luck? You would be, if you didn't have this ultra-cool Adafruit PCA9548 8 Channel STEMMA QT / Qwiic I2C Multiplexer!

Finally, a way to get up to 8 same-address I2C devices hooked up to one microcontroller - this multiplexer acts as a gatekeeper, shuttling the commands to the selected I2C port with your command.

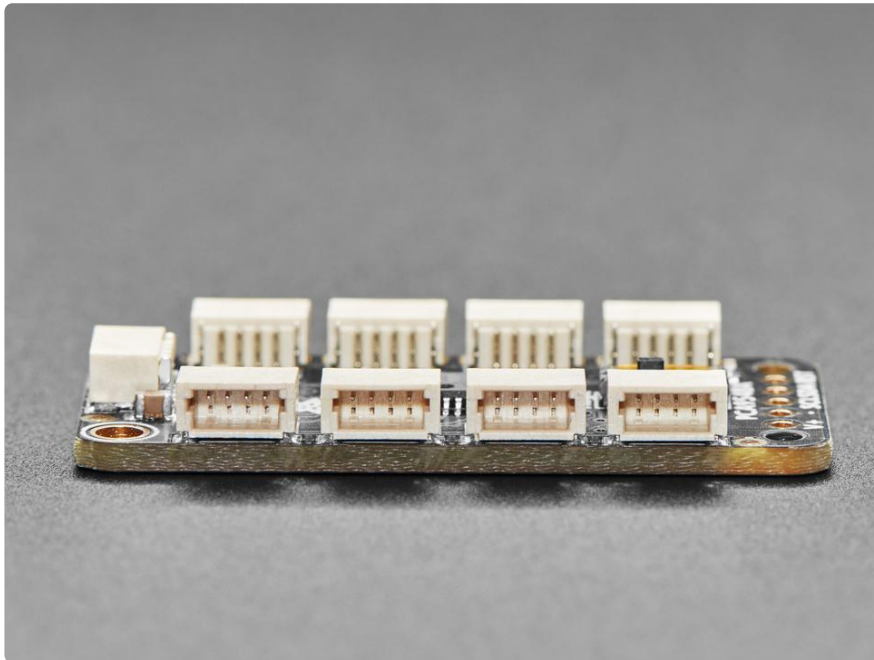


In case you're wondering why this uses the PCA9548A not the [TCA9548A](#) (), the PCA9548 is the 'fraternal twin sister' of the TCA9548 but is easier to get during the great chip shortage of 2022. It works exactly the same, just can't go down to 1.8V power which is OK because QT boards are 3V or 5V only anyways. [You can still use any example code or library for the TCA9548](#) ()

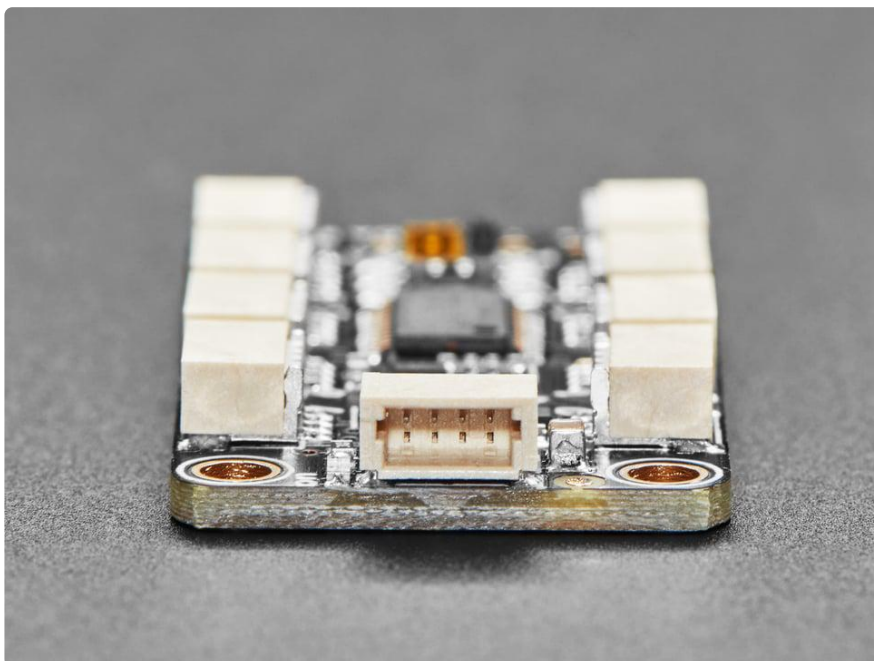


Using it is fairly straight-forward: the multiplexer itself is on I2C address 0x70 (but can be adjusted from 0x70 to 0x77 using jumpers on the back) and you simply write a single byte with the desired multiplexed output number to that port, and bam - any future I2C packets will get sent to that port. In theory, you could have 8 of these

multiplexers on each of 0x70-0x77 addresses in order to control 64 of the same-I2C-addressed-part.



The Adafruit STEMMA QT / Qwiic PCA9548A Mux Breakout - 8 Channel has eight JST SH 1mm connectors in two rows of four, all with the power, ground, and SDA/SCL pins connected. There's one port at the end that connects to your I2C controller (there are also breadboard breakout pins if you need them). Use this breakout to add as many I2C devices to the bus as you need. Complete with mounting holes so the board can be added to any system. A small power LED lets you know that the hub board has connectivity.

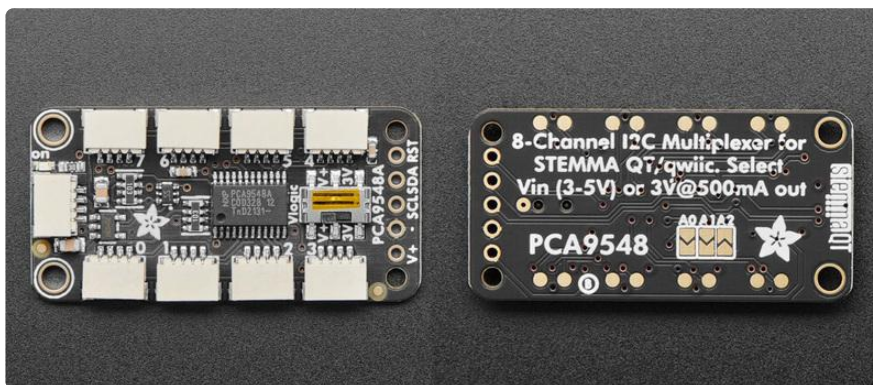


There's even an onboard 3.3V 500mA regulator, so if you're using this with a 5V microcontroller like an Arduino 328-compatible, you can level shift all the QT ports to have 3V power and logic level.

Of course, because [STEMMA QT is Qwiic compatible \(\)](#), it will [work with any and all STEMMA QT or Qwiic boards and parts we have in the Adafruit shop \(\)](#).

Comes with only the assembled PCB, no [cables or sensors included \(we have tons available though!\) \(\)](#)

Pinouts



The default I2C address is 0x70.

Power Pins

- V+ - this is the power pin. Since the multiplexer chip uses 3-5 VDC, to power the board, give it the same power as the logic level of your microcontroller - e.g. for a 5V micro like Arduino, use 5V
This connection is shared with the STEMMA QT port on the opposite side of the board.
- - - common ground for power and logic.
This connection is shared with the STEMMA QT port on the opposite side of the board.

Vlogic Switch

- Vlogic - this switch selects between supplying the 8 STEMMA QT ports with either the incoming voltage on V+ (3-5 VDC) or level shifting to 3.3V 500mA (labeled 3V). For example, if you're using this with a 5V microcontroller like an

Arduino 328-compatible, you can level shift all the QT ports to have 3V power and logic level.

I2C Logic Pins - Control

- SCL - I2C clock pin, connect to your microcontroller I2C clock line. This pin is level shifted so you can use 3-5V logic, and there's a 10K pullup on this pin. This connection is shared with the STEMMA QT port on the opposite side of the board.
- SDA - I2C data pin, connect to your microcontroller I2C data line. This pin is level shifted so you can use 3-5V logic, and there's a 10K pullup on this pin. This connection is shared with the STEMMA QT port on the opposite side of the board.
- [STEMMA QT \(\)](#) - These connectors allow you to connect to dev boards with STEMMA QT connectors or to other things with [various associated accessories \(\)](#). There's one port at the end that connects to your microcontroller. The other eight connectors in two rows of four are discussed below.

I2C Logic Pins - Multiplexed

- STEMMA QT Ports 0 to 7 - There are eight JST SH 1mm connectors in two rows of four, all with the power, ground, and SDA/SCL pins connected. There are 8 sets of SDA and SCL pins, from SD0/SC0 to SD7/SC7. These are the multiplexed pins. Each one is a completely separate I2C bus set. You can have 8 I2C devices with identical addresses, as long as they are on one I2C bus each. The power input for the connectors is the output from the Vlogic switch, and is either V+ or 3.3V 500mA.

These ports do not have any pullups installed, so if you are using a chip or breakout without I2C pullups be sure to add them!

Address Pins

On the back of the board are three address jumpers, labeled A0, A1, and A2, between the board name and Adafruit logo on the board silk. These jumpers allow you to chain

up to 8 of these boards on the same pair of I2C clock and data pins. To do so, you solder the jumpers "closed" by connecting the two pads.

The default I2C address is 0x70. The other address options can be calculated by "adding" the A0/A1/A2 to the base of 0x70.

A0 sets the lowest bit with a value of 1, A1 sets the next bit with a value of 2 and A2 sets the next bit with a value of 4. The final address is $0x70 + A2 + A1 + A0$ which would be 0x77.

So for example if A2 is soldered closed and A0 is soldered closed, the address is $0x70 + 4 + 1 = 0x75$.

If only A0 is soldered closed, the address is $0x70 + 1 = 0x71$

If only A1 is soldered closed, the address is $0x70 + 2 = 0x72$

If only A2 is soldered closed, the address is $0x70 + 4 = 0x74$

The table below shows all possible addresses, and whether the pin(s) should be high (closed) or low (open).

ADDR	A0	A1	A2	ADDR	A0	A1	A2
0x70	L	L	L	0x74	L	L	H
0x71	H	L	L	0x75	H	L	H
0x72	L	H	L	0x76	L	H	H
0x73	H	H	L	0x77	H	H	H

Reset Pin

- RST - Reset pin for resetting the multiplexer chip. Pulled high by default, connect to ground to reset.

Power LED

- Power LED - In the upper left corner, above the STEMMA connector, on the front of the board, is the power LED, labeled on. It is the green LED.

CircuitPython & Python

It's easy to use the PCA9548 with Python or CircuitPython, and the [Adafruit_CircuitPython_TCA9548A \(\)](#) module. This module allows you to easily write Python code that allows you to multiplex up to 8 STEMMA boards with the PCA9548 I2C multiplexer. You can use this multiplexer with any CircuitPython microcontroller board or with a computer that has GPIO and Python [thanks to Adafruit_Blinka, our CircuitPython-for-Python compatibility library \(\)](#).

If you're curious why you'd need an I2C multiplexer, be sure to [check out this guide \(\)](#) that goes in depth on working with multiple copies of the same I2C device, which most likely have the same I2C address.

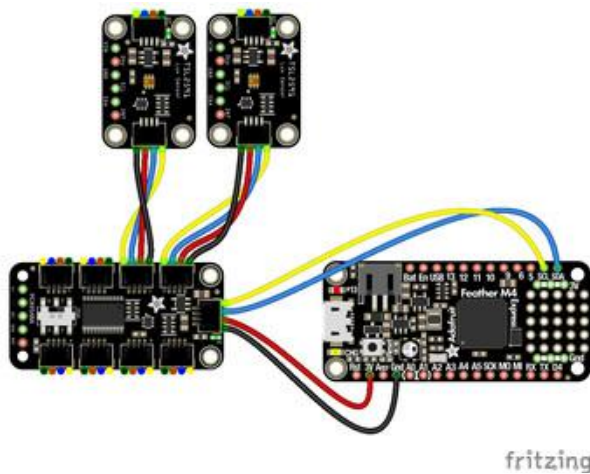
Working with Multiple Same Address
I2C Devices Learn Guide

Why the Adafruit_CircuitPython_TCA9548A Module?

The PCA9548 is the 'fraternal twin sister' of the TCA9548 but it is easier to get during the great chip shortage of 2022. It works exactly the same, it just can't go down to 1.8V power which is OK because QT boards are 3V or 5V only anyways. [You can still use any example code or library for the TCA9548 \(\)](#)

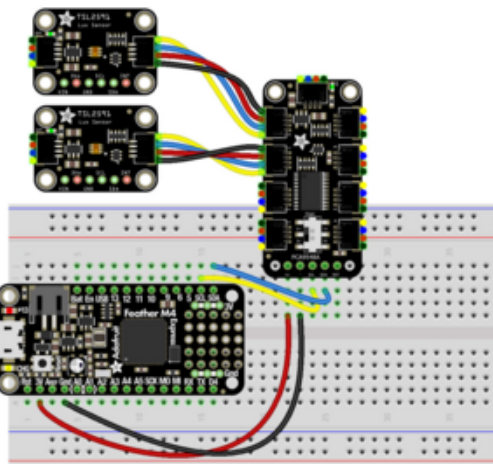
CircuitPython Microcontroller Wiring

First, wire up a PCA9548 to your board exactly as shown below. Here's an example of wiring a Feather M4 to the PCA9548 with I2C using one of the handy [STEMMA QT \(\)](#) connectors. Then, plug two TSL2591 STEMMA boards into the PCA9548 via STEMMA plug 0 and STEMMA plug 1:



Board 3V to mux VIN (red wire)
 Board GND to mux GND (black wire)
 Board SCL to mux SCL (yellow wire)
 Board SDA to mux SDA (blue wire)
 TSL2591 1 to mux STEMMA port 0
 TSL2591 2 to mux STEMMA port 1

You can also use standard 0.100" pitch headers to wire it up on a breadboard:

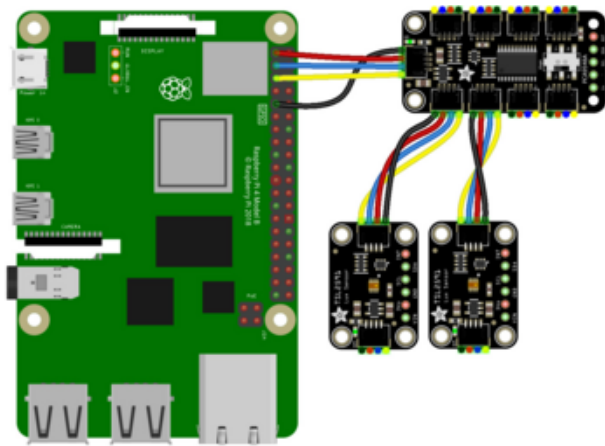


Board 3V to mux VIN (red wire)
 Board GND to mux GND (black wire)
 Board SCL to mux SCL (yellow wire)
 Board SDA to mux SDA (blue wire)
 TSL2591 1 to mux STEMMA port 0
 TSL2591 2 to mux STEMMA port 1

Python Computer Wiring

Since there's dozens of Linux computers/boards you can use, below shows wiring for Raspberry Pi. For other platforms, [please visit the guide for CircuitPython on Linux to see whether your platform is supported \(\)](#).

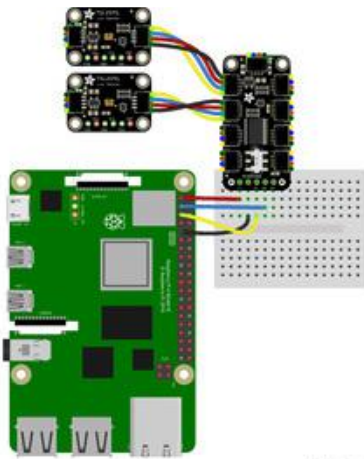
Here's the Raspberry Pi wired to the I2C multiplexer using I2C and a [STEMMA QT \(\)](#) connector:



fritzing

- Pi 3V to mux VIN (red wire)
- Pi GND to mux GND (black wire)
- Pi SCL to mux SCL (yellow wire)
- Pi SDA to mux SDA (blue wire)
- TSL2591 1 to mux STEMMA port 0
- TSL2591 2 to mux STEMMA port 1

Finally here is an example of how to wire up a Raspberry Pi to the I2C multiplexer using a solderless breadboard:



fritzing

- Pi 3V to mux VIN (red wire)
- Pi GND to mux GND (black wire)
- Pi SCL to mux SCL (yellow wire)
- Pi SDA to mux SDA (blue wire)
- TSL2591 1 to mux STEMMA port 0
- TSL2591 2 to mux STEMMA port 1

Python Installation of TCA9548A 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 \(\)!](#)

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

- `sudo pip3 install adafruit-circuitpython-tca9548a`

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

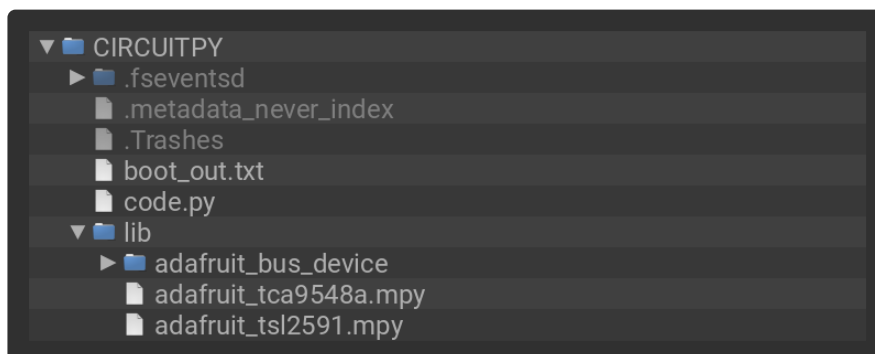
CircuitPython Usage

To use with CircuitPython, you need to first install the TCA9548A library, and its dependencies, into the lib folder on your CIRCUITPY drive. Then you need to update code.py with the example script.

Thankfully, we 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 and file:

- adafruit_bus_device/
- adafruit_tca9548a.mpy
- adafruit_tsl2591.mpy



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
```

Simple Test Example Code

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

# This example shows using TCA9548A to perform a simple scan for connected devices
import board
import adafruit_tca9548a
```

```

# Create I2C bus as normal
i2c = board.I2C() # uses board.SCL and board.SDA

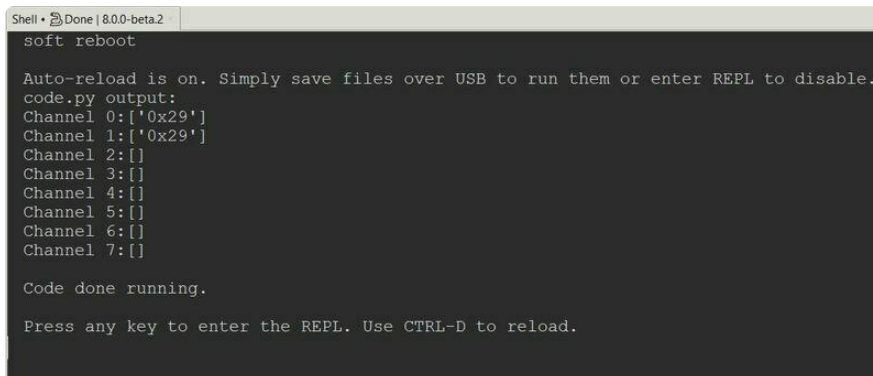
# Create the TCA9548A object and give it the I2C bus
tca = adafruit_tca9548a.TCA9548A(i2c)

for channel in range(8):
    if tca[channel].try_lock():
        print("Channel {}: ".format(channel), end="")
        addresses = tca[channel].scan()
        print([hex(address) for address in addresses if address != 0x70])
        tca[channel].unlock()

```

If running CircuitPython: Once everything is saved to the CIRCUITPY drive, [connect to the serial console \(\)](#) to see the data printed out!

If running Python: The console output will appear wherever you are running Python.



```

Shell • Done | 8.0.0-beta.2
soft reboot

Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
code.py output:
Channel 0: ['0x29']
Channel 1: ['0x29']
Channel 2: []
Channel 3: []
Channel 4: []
Channel 5: []
Channel 6: []
Channel 7: []

Code done running.

Press any key to enter the REPL. Use CTRL-D to reload.

```

In this simple test for the PCA9548, an I2C scan is performed for all eight of its ports. If any devices are connected, then the I2C address will be printed to the REPL next to the channel number. If no device is connected, then the port will print with empty brackets ([]).

Multi-Sensor Example Code

```

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

# This example shows using two TSL2491 light sensors attached to TCA9548A channels
# 0 and 1.
# Use with other I2C sensors would be similar.
import time
import board
import adafruit_tsl2591
import adafruit_tca9548a

# Create I2C bus as normal
i2c = board.I2C() # uses board.SCL and board.SDA

# Create the TCA9548A object and give it the I2C bus
tca = adafruit_tca9548a.TCA9548A(i2c)

# For each sensor, create it using the TCA9548A channel instead of the I2C object

```

```

tsl1 = adafruit_tsl2591.TSL2591(tca[0])
tsl2 = adafruit_tsl2591.TSL2591(tca[1])

# After initial setup, can just use sensors as normal.
while True:
    print(tsl1.lux, tsl2.lux)
    time.sleep(0.1)

```

In the multi-sensor example, the PCA9548 is used as an I2C multiplexer with two TSL2591 light sensors. When the connected sensors are instantiated over I2C, the I2C pins declared are the ports from `0` and `1` on the PCA9548.

In the example, the first TSL2591 light sensor, instantiated as `tsl1`, is plugged into port 0 (`tca[0]`) and the second TSL2591 light sensor, instantiated as `tsl2`, is plugged into port 1 (`tca[1]`).

```

# Create the TCA9548A object and give it the I2C bus
tca = adafruit_tca9548a.TCA9548A(i2c)

# For each sensor, create it using the TCA9548A channel instead of the I2C object
tsl1 = adafruit_tsl2591.TSL2591(tca[0])
tsl2 = adafruit_tsl2591.TSL2591(tca[1])

```

In the loop, the readings from the two light sensors are printed to the REPL every `0.1` seconds.

```

while True:
    print(tsl1.lux, tsl2.lux)
    time.sleep(0.1)

```

```

Shell • 22@code.py KeyboardInterrupt | 8.0.0-beta.2
14.5833 22.5792
13.3333 16.2432
10.8333 19.6416
10.8333 23.2128
6.25 27.7056
7.08333 26.9568
7.08333 18.6624
5.0 16.5888
4.16667 16.5888
4.16667 17.0496
4.16667 18.8928
4.16667 22.0608
4.16667 35.136
4.16667 22.176
6.66667 24.9984
8.33333 24.8832

```

Python Docs

[Python Docs \(\)](#)

Arduino

Using the PCA9548 I2C multiplexer with Arduino involves wiring up the I2C multiplexer to your Arduino-compatible microcontroller and running the provided example code.

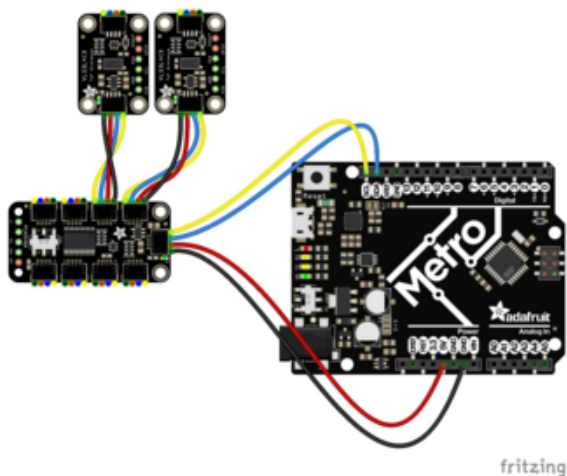
If you're curious why you'd need an I2C multiplexer, be sure to [check out this guide \(\)](#) that goes in depth on working with multiple copies of the same I2C device, which most likely have the same I2C address.

Working with Multiple Same Address I2C Devices Learn Guide

Wiring

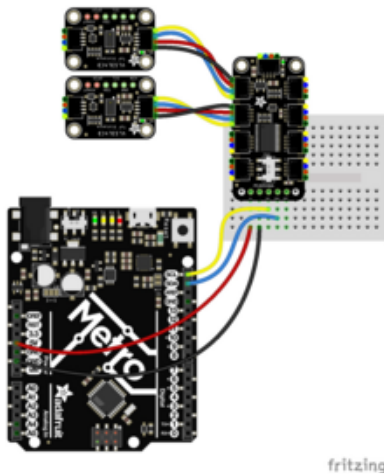
Wire as shown for a 5V board like an Uno. If you are using a 3V board, like an Adafruit Feather, wire the board's 3V pin to the PCA9548 VIN.

Here is an Adafruit Metro wired up to the PCA9548 using the STEMMA QT connector, along with two VL53L4CD STEMMA boards plugged into port 0 and port 1 on the PCA9548:



Board 5V to mux VIN (red wire)
Board GND to mux GND (black wire)
Board SCL to mux SCL (yellow wire)
Board SDA to mux SDA (blue wire)
VL53L4CD 1 to mux STEMMA port 0
VL53L4CD 2 to mux STEMMA port 1

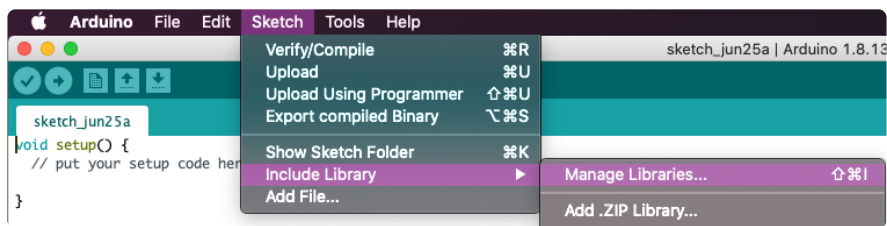
Here is an Adafruit Metro wired up using a solderless breadboard:



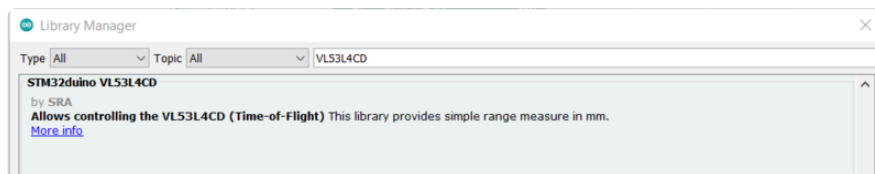
Board 5V to mux VIN (red wire)
 Board GND to mux GND (black wire)
 Board SCL to mux SCL (yellow wire)
 Board SDA to mux SDA (blue wire)
 VL53L4CD 1 to mux STEMMA port 0
 VL53L4CD 2 to mux STEMMA port 1

Library Installation

The [Multi-Sensor example \(\)](#) uses two VL53L4CD time of flight sensors. You can install the VL53L4CD library for Arduino using the Library Manager in the Arduino IDE.



Click the Manage Libraries ... menu item, search for VL53L4CD, and select the STM32duino VL53L4CD library:



I2C Scanner Example Code

```
/**
 * TCA9548 I2CScanner.ino -- I2C bus scanner for Arduino
 *
 * Based on https://playground.arduino.cc/Main/I2cScanner/
 *
 */

#include "Wire.h"

#define PCAADDR 0x70

void pcselect(uint8_t i) {
  if (i > 7) return;
```



```

Wire.beginTransmission(PCADDR);
Wire.write(1 &&& i);
Wire.endTransmission();
}

// standard Arduino setup()
void setup()
{
  while (!Serial);
  delay(1000);

  Wire.begin();

  Serial.begin(115200);
  Serial.println("\nPCAScanner ready!");

  for (uint8_t t=0; t<8; t++) {
    pcaselect(t);
    Serial.print("PCA Port #"); Serial.println(t);

    for (uint8_t addr = 0; addr<=127; addr++) {
      if (addr == PCADDR) continue;

      Wire.beginTransmission(addr);
      if (!Wire.endTransmission()) {
        Serial.print("Found I2C 0x"); Serial.println(addr,HEX);
      }
    }
  }
  Serial.println("\ndone");
}

void loop()
{
}

```

```

COM5

PCAScanner ready!
PCA Port #0
Found I2C 0x29
PCA Port #1
Found I2C 0x29
PCA Port #2
PCA Port #3
PCA Port #4
PCA Port #5
PCA Port #6
PCA Port #7
done

```

Upload the sketch to your board and open up the Serial Monitor (Tools -> Serial Monitor) at 115200 baud. You should see the 8 ports print to the Serial Monitor. If an I2C device is plugged into one of the ports, its address will be printed below the port number.

Multi-Sensor Example Code

```
/**
 * PCA9548 I2C Multi Sensor Example
 *
 * Using two VL53L4CD sensors on ports 0 and 1
 *
 */

#include <Arduino.h>
#include <Wire.h>
#include <vl53l4cd_class.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>
#include <assert.h>

#define PCAADDR 0x70
#define DEV_I2C Wire

#define SerialPort Serial

// create two instances of the sensor
VL53L4CD tof1(&DEV_I2C, A1);
VL53L4CD tof2(&DEV_I2C, A1);

void pcaselect(uint8_t i) {
    if (i > 7) return;

    Wire.beginTransaction(PCAADDR);
    Wire.write(1 << i);
    Wire.endTransmission();
}

// standard Arduino setup()
void setup()
{
    while (!Serial);
    delay(1000);

    Wire.begin();

    Serial.begin(115200);
    Serial.println("\nMultiSensor PCA9548");

    // define the port on the PCA9548 for the first sensor
    pcaselect(0);
    // setup the first sensor
    tof1.begin();
    tof1.VL53L4CD_Off();
    tof1.InitSensor();
    tof1.VL53L4CD_SetRangeTiming(200, 0);
    tof1.VL53L4CD_StartRanging();

    // define the port on the PCA9548 for the 2nd sensor
    pcaselect(1);

    // setup the 2nd sensor
    tof2.begin();
    tof2.VL53L4CD_Off();
    tof2.InitSensor();
    tof2.VL53L4CD_SetRangeTiming(200, 0);
    tof2.VL53L4CD_StartRanging();
}
```

```

void loop()
{
  uint8_t NewDataReady = 0;
  VL53L4CD_Result_t results;
  uint8_t status;
  char report[64];

  // define port on the PCA9584
  pcselect(0);

  // loop for time of flight sensor 1
  do {
    status = tof1.VL53L4CD_CheckForDataReady(&NewDataReady);
  } while (!NewDataReady);

  if ((!status) && (NewDataReady != 0)) {
    // (Mandatory) Clear HW interrupt to restart measurements
    tof1.VL53L4CD_ClearInterrupt();

    // Read measured distance. RangeStatus = 0 means valid data
    tof1.VL53L4CD_GetResult(&results);
    snprintf(report, sizeof(report), "ToF 1 - Status = %3u, Distance = %5u mm,
Signal = %6u kcps/spad\r\n",
             results.range_status,
             results.distance_mm,
             results.signal_per_spad_kcps);
    SerialPort.println(report);
  }

  // define port on PCA9548
  pcselect(1);

  // loop for time of flight sensor 2
  do {
    status = tof2.VL53L4CD_CheckForDataReady(&NewDataReady);
  } while (!NewDataReady);

  if ((!status) && (NewDataReady != 0)) {
    // (Mandatory) Clear HW interrupt to restart measurements
    tof2.VL53L4CD_ClearInterrupt();

    // Read measured distance. RangeStatus = 0 means valid data
    tof2.VL53L4CD_GetResult(&results);
    snprintf(report, sizeof(report), "ToF 2 - Status = %3u, Distance = %5u mm,
Signal = %6u kcps/spad\r\n",
             results.range_status,
             results.distance_mm,
             results.signal_per_spad_kcps);
    SerialPort.println(report);
  }
}

```

```
COM5
MultiSensor PCA9548
ToF 1 - Status = 0, Distance = 46 mm, Signal = 3317 kcps
ToF 2 - Status = 4, Distance = 1995 mm, Signal = 2 kcps
ToF 1 - Status = 0, Distance = 60 mm, Signal = 2269 kcps
ToF 2 - Status = 7, Distance = 0 mm, Signal = 10 kcps
ToF 1 - Status = 2, Distance = 163 mm, Signal = 165 kcps
ToF 2 - Status = 0, Distance = 100 mm, Signal = 65 kcps
ToF 1 - Status = 2, Distance = 296 mm, Signal = 15 kcps
ToF 2 - Status = 0, Distance = 44 mm, Signal = 1971 kcps
ToF 1 - Status = 2, Distance = 421 mm, Signal = 4 kcps
ToF 2 - Status = 0, Distance = 102 mm, Signal = 864 kcps
ToF 1 - Status = 0, Distance = 173 mm, Signal = 91 kcps
ToF 2 - Status = 2, Distance = 208 mm, Signal = 25 kcps
ToF 1 - Status = 0, Distance = 75 mm, Signal = 640 kcps
ToF 2 - Status = 4, Distance = 1998 mm, Signal = 2 kcps
ToF 1 - Status = 0, Distance = 74 mm, Signal = 873 kcps
ToF 2 - Status = 4, Distance = 0 mm, Signal = 2 kcps
ToF 1 - Status = 0, Distance = 148 mm, Signal = 208 kcps
ToF 2 - Status = 7, Distance = 227 mm, Signal = 127 kcps
ToF 1 - Status = 0, Distance = 283 mm, Signal = 35 kcps
ToF 2 - Status = 0, Distance = 58 mm, Signal = 263 kcps
```

Upload the sketch to your board and open up the Serial Monitor (Tools -> Serial Monitor) at 115200 baud. You should see readings from the two VL53L4CD time of flight sensors print to the Serial Monitor.

Downloads

Files

- [PCA9548 Datasheet \(\)](#)
- [EagleCAD PCB files on GitHub \(\)](#)
- [Fritzing object in the Adafruit Fritzing Library \(\)](#)

Schematic and Fab Print

