How to Scan and Detect I2C Addresses

Created by Carter Nelson

https://learn.adafruit.com/scanning-i2c-addresses

Last updated on 2023-11-28 04:35:04 PM EST
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>3</td>
</tr>
<tr>
<td><strong>I2C Basics</strong></td>
<td>3</td>
</tr>
<tr>
<td>• Terminology</td>
<td></td>
</tr>
<tr>
<td>• The I2C Bus</td>
<td></td>
</tr>
<tr>
<td>• Pull Up Resistors</td>
<td></td>
</tr>
<tr>
<td>• Addresses</td>
<td></td>
</tr>
<tr>
<td>• Hex Address Nomenclature</td>
<td></td>
</tr>
<tr>
<td><strong>I2C Address Scanning</strong></td>
<td>6</td>
</tr>
<tr>
<td><strong>Arduino</strong></td>
<td>6</td>
</tr>
<tr>
<td>• Specify Alternate I2C Bus</td>
<td></td>
</tr>
<tr>
<td>• Normal Behavior</td>
<td></td>
</tr>
<tr>
<td>• Missing Device or Pull Ups</td>
<td></td>
</tr>
<tr>
<td><strong>Arduino Using TestBed</strong></td>
<td>9</td>
</tr>
<tr>
<td>• Scan Results</td>
<td></td>
</tr>
<tr>
<td><strong>CircuitPython</strong></td>
<td>10</td>
</tr>
<tr>
<td>• I2C Bus Selection</td>
<td></td>
</tr>
<tr>
<td><strong>Raspberry Pi</strong></td>
<td>12</td>
</tr>
<tr>
<td>• Normal Behavior</td>
<td></td>
</tr>
<tr>
<td>• Missing Device or Pull Ups</td>
<td></td>
</tr>
<tr>
<td><strong>Troubleshooting</strong></td>
<td>13</td>
</tr>
<tr>
<td>• Correct Addresses Show Up</td>
<td></td>
</tr>
<tr>
<td>• Incorrect Addresses Show Up</td>
<td></td>
</tr>
<tr>
<td>• No Addresses Show Up</td>
<td></td>
</tr>
<tr>
<td>• &quot;No pull up found on SDA or SCL&quot;</td>
<td></td>
</tr>
<tr>
<td>• &quot;TimeoutError: Clock stretch too long&quot;</td>
<td></td>
</tr>
</tbody>
</table>
A lot of breakout boards use Inter-Integrate Circuit (I2C) to communicate with the host microcontroller. This is a nice general purpose communication bus that only requires two pins for the data transmission. When using breakouts that include STEM MA or STEMMA QT connectors, usage is essentially plug-and-play.

However, sometimes things just don't seem to work for some reason. When that happens, the best initial troubleshooting step is to do an I2C scan. This is a simple "Hey! Who's out there?" sort of sanity check.

In this guide, we show how to perform an I2C scan on various platforms and provide some guidance on how to use the results to aid in troubleshooting.

I2C Basics

Terminology

Here we define some terms related to I2C.

- SDA - Serial DAta line. This is the line the 1's and 0's go down to communicate data.
- SCL - Serial CLock line. This is a square wave clock signal that is used to coordinate when to read the SDA line.
- VCC - The supply voltage for power, typically 3.3V or 5V.
• GND - Ground
• Address - Each I2C device has a specific numerical address.
• Pull Up Resistors - Resistors required between SCL and VCC as well as between SDA and VCC.

The I2C Bus

Here is a simple diagram showing the general arrangement of an I2C bus with several devices attached.

There are two main items that are important here:

1. Pull up resistors between SCL/SDA and VCC
2. Unique addresses for each device

Most I2C issues relate to something wrong with these two factors.

Pull Up Resistors

Only one set of pull up resistors are needed for the entire I2C bus. A stand alone device, like an iPhone, will have taken care of this internally. In that case, the microcontroller and all the I2C devices are self contained within the iPhone itself. But for maker applications, the microcontroller and I2C devices are typically provided as
separate items. In that case, where are the pull up resistors? It depends, but in general:

- Adafruit I2C breakouts include pull up resistors.
- Mix mode breakouts may or may not include pull up resistors.
- Raspberry Pi's include pull up resistors on their I2C pins.

It's OK to have more than one pull up resistor. That would happen if you chained more than one Adafruit I2C breakout together, since each includes pull ups. The resistors will act in parallel and thus reduce the overall pull up resistance. But I2C can work with a range of pull up values, so this is generally not an issue.

Addresses

Each device on the I2C bus needs a unique address. How do you determine what the address is for any given breakout? The datasheet is the main source, but it can be a bit buried. Sometimes the address will be mentioned on the product page or in the associated learn guide. Some breakouts will print the address on the PCB itself. The source code for the sensor's library is another option, but again, can be a bit buried. That's why there's this convenient list of nothing but I2C addresses:

I2C Addresses - The List

But it requires manually updating, so may not be 100% complete.

But also, don't worry - you don't need to know the address to perform an I2C scan. It just helps to know what to expect from the scan results.

Knowing your device I2C address is not required to perform a scan.

There are some devices that have more than one I2C address! Especially address 0x00 (which is special)
Hex Address Nomenclature

By convention, I2C addresses are reported in hexadecimal (0x) notation. That is what the leading 0x indicates. We won’t discuss hex notation here, just keep in mind that when you see 0x77, it is not the same as 77.

I2C Address Scanning

OK, let’s get into how to perform an I2C address scan. While simple, the exact process is unique depending on what platform you are using. The following sections detail the process for the main scenarios of interest - Arduino, CircuitPython, and Raspberry Pi.

Arduino

There is an excellent I2C scanner sketch available from the old Arduino Playground here:

Arduino I2C Scanner

Here is a slightly modified version that allows easily specifying an alternate I2C bus. See below for more info. The default Wire should hopefully work for most boards though.

```cpp
// i2c_scanner
// Modified from https://playground.arduino.cc/Main/I2cScanner/

#include <Wire.h>

void setup() {
  Wire.begin();
  Serial.begin(9600);
  while (!Serial)
    delay(10);
  Serial.println("I2C Scanner");
}
```

While addresses are rarely referred in decimal, they are sometimes referenced in ‘8-bit’ mode rather than 7-bit mode. In that case, a device with 7-bit address 0x39 would be referred to as 0x39 x 2 = 0x72
void loop() {
    byte error, address;
    int nDevices;

    Serial.println("Scanning...");

    nDevices = 0;
    for(address = 1; address < 127; address++)
    {
        // The i2c_scanner uses the return value of
        // the Wire.endTransmission to see if
        // a device did acknowledge to the address.
        WIRE.beginTransmission(address);
        error = WIRE.endTransmission();

        if (error == 0)
        {
            Serial.print("I2C device found at address 0x");
            if (address<16)
                Serial.print("0");
            Serial.print(address,HEX);
            Serial.println("!");

            nDevices++;
        }
        else if (error==4)
        {
            Serial.print("Unknown error at address 0x");
            if (address<16)
                Serial.print("0");
            Serial.println(address,HEX);
        }
    }

    if (nDevices == 0)
        Serial.println("No I2C devices found\n");
    else
        Serial.println("done\n");

delay(5000);            // wait 5 seconds for next scan
}

Load and run that sketch as you would any other sketch onto your Arduino board.

Specify Alternate I2C Bus

The example sketch above uses the default Wire bus. To run the I2C scan on a different bus, change this line of code:

```c
#define WIRE Wire
```

to specify the bus to use. For example, to use Wire1, change the line to this:

```c
#define WIRE Wire1
```
**Arduino DUE Caution**

Note that the Arduino DUE places `Wire1` where most other boards place `Wire`. Be sure to change scanner sketch code as needed.

Normal Behavior

If all goes well, you should get a list of addresses for each device found. In the example below, an [Adafruit BMP280 breakout](https://www.adafruit.com/product/2281) is attached to an Arduino UNO.

The BMP280’s I2C address of `0x77` shows up as expected.
Missing Device or Pull Ups

If the device is disconnect and/or the pull up resistors are missing, the results will look like this:

![Image of scanning output]

Note that the sketch runs just fine, but reports no addresses.

A mis-wired board may hang the sketch after the "Scanning..." text is printed

Arduino Using TestBed

This Arduino I2C scanner sketch uses the [Adafruit TestBed](https://adafruit.github.io/testbed/) library. This library takes care of a lot of the board specific boiler plate setup, so should be able to be used as is on most platforms.

To use, first install the Adafruit TestBed library using the Arduino Library Manager:

Tools > Manage Libraries...

Search for "testbed" to find the library and then click Install.
Then the I2C scanner can be found in the library examples.

File > Examples > Adafruit TestBed > I2C_Scan

Once the sketch is uploaded to the board, open the Arduino Serial Monitor to see the scan results.

Scan Results

The board's primary I2C port, aka `Wire`, will be scanned. Additionally, if a secondary I2C port, aka `Wire1`, exists for the board, then it will also be scanned.

Here is example output for a board with two I2C ports, with a single I2C device attached to each:

---

**CircuitPython**

An I2C scanner example for CircuitPython is provided in the CircuitPython Essentials guide. Here's the code:

```python
# SPDX-FileCopyrightText: 2017 Limor Fried for Adafruit Industries
#
```
Use the [Welcome to CircuitPython()](https://circuitpython.readthedocs.io/en/latest/) guide to learn how to load and run this on your board.

### I2C Bus Selection

The I2C scanner code above tries to automatically detect the presence of the most commonly used I2C buses. It then runs an I2C scan on each valid bus found. Therefore, the code should work as is, without any need to modify it, on most boards.
Here is example output for a QT Py RP2040 with two I2C devices attached - one to the STEMMA QT port and the other to the SCL/SDA header pins. These are two different I2C busses on the QT Py RP20404. So it scans both:

```python
Checking board.I2C()...ADDED.
Checking board.STEMMA_I2C()...ADDED.
Checking busio.I2C(board.GP1, board.GP0)...SKIPPED.
----------------------------------------
I2C SCAN
----------------------------------------
board.I2C() addresses found: ['0x76']
board.STEMMA_I2C() addresses found: ['0x23']
board.I2C() addresses found: ['0x76']
board.STEMMA_I2C() addresses found: ['0x23']
board.I2C() addresses found: ['0x76']
board.STEMMA_I2C() addresses found: ['0x23']
```

**Raspberry Pi**

To do an I2C scan on a Raspberry Pi the `i2cdetect` command is used. If not already done, be sure to enable I2C on the Raspberry Pi via `raspi-config`. If the `i2cdetect` command is not found, install it with:

```bash
sudo apt-get install i2c-tools
```

And then to run a scan, use `i2cdetect` with the following command line parameters:

```bash
i2cdetect -y 1
```

On modern Raspberry Pi OS releases, you do not need to run the command with `sudo`. The `-y` disables interactive mode, so it just goes ahead and scans. The `1` specifies the I2C bus.

**Normal Behavior**

If all goes well, you should get a list of addresses for each device found. In the example below, an [Adafruit BMP280 breakout](https://www.adafruit.com/product/4085) is attached to a Raspberry Pi 4.
The BMP280's I2C address of **0x77** shows up as expected.

### Missing Device or Pull Ups

If the device is disconnect and/or the pull up resistors are missing, the results will look like this:

![I2C scan results](image)

The scan runs fine, but no addresses are shown. Keep in mind that Raspberry Pi's include pull up resistors on the SCL and SDA pins.

### Troubleshooting

OK, so you've done a scan and gotten some results. How do those help with troubleshooting. Here we discuss various scenarios and what they imply.

#### Correct Addresses Show Up

In general, you should be all good. There's nothing more to be done with an I2C scan. Go ahead and start trying to use the device, preferably using an example from the devices library.

If you get additional errors when trying to run the library example, post in the Adafruit forums with details.

#### Incorrect Addresses Show Up

Weird, but can happen in rare cases. Double check the device and make sure it is what you think it is. Some devices can have more than one I2C address, typically settable somehow by the users. Check if the incorrect address is one of these alternate addresses.

If everything seems correct but you are still getting an incorrect address, post in the Adafruit forums with details.
No Addresses Show Up

Try just scanning again. Some devices can be weird and not show up on first scan. If that still does not work, then: Check your wiring and/or your soldering. This accounts for the vast majority of issues.

Wiring

Wiring issues can be incorrect pins or even just bad wires. Double and triple check you are using the correct pins. If you are using a breadboard, try moving to a different location. Breadboards can wear out with time. Also try totally different wires. The pre-crimped prototype wires commonly used can also wear out over time.

Soldering

This is unfortunately a very common issue for beginners. Soldering is not super tough, but does take a little getting used to. There’s a good general guide on soldering as well as a guide specific to the commonly used headers pins.

This photo from the first guide is an excellent summary of common soldering issues:

If you’ve checked all this and are still not getting the expected results, post in the Adafruit forums with details. Photos are very helpful when doing so.

"No pull up found on SDA or SCL"

This error message is unique to CircuitPython, which performs a check on the SDA and SCL pins to determine if the pull up resistors are present. This is most likely a connection issue, either wiring/soldering or connected to the wrong pins. See info above about checking wiring and soldering.
If wiring and soldering seem good, then it could be that the pins being used by the code are not the same pins the device is attached to. Some boards have multiple I2C ports and/or the I2C pins can show up in multiple locations. This can lead to general confusion about which specific pins should be used, esp. when using the built ins like `board.I2C()`. Double check the board's pinout information with special attention on where the SDA and SCL pins are located. For boards with STEMMA QT connectors, check if there is a `board.STEMMA_I2C()` defined.

"TimeoutError: Clock stretch too long"

This error message is unique to CircuitPython. Clock stretching is discussed elsewhere and CircuitPython has safeguards against overly long stretches. If you're using an I2C device that is known to have long clock stretches, then it may be hitting CircuitPython limits.

However, an electrical short between the SDA and SCL pins can potentially cause this error to show up. In that case, it will typically occur when the I2C bus is created, i.e. on the code line with `board.I2C()`, or `busio.I2C()`, or `bitbangio.I2C()`, etc. Even a tiny solder whisker can create enough of an electrical short to cause this to occur. So check for any obvious shorts, but also carefully examine any soldering as well.