

Contents

1	Serial communication with I2C	1
1.1	Notes	1
1.2	Parts	2
1.3	Procedure	2
1.3.1	Enable I2C on Raspberry Pi OS	2
1.3.2	Install the library	2
1.3.3	Connect the SSD1306 OLED screen	3
1.3.4	Scan the I2C bus and find OLED screen address	3
1.3.5	Send text and images to the OLED display	4
1.3.6	Examine I2C communication in <code>piscope</code>	4
1.3.7	Use terminal-based utilities	5

1 Serial communication with I2C

In this lab, we'll learn how to connect external parts to a single board computer using a digital communication bus, specifically: using an I2C communication bus. At the end of this lab, you should be able to:

- Connect a peripheral device or sensor to the I2C bus on the Raspberry Pi.
- Use a datasheet to identify the registers, and values to write to those registers, to configure and use a peripheral device or sensor.
- Inspect the digital waveform transmitted over the I2C bus, and identify important features (most and least significant bits, start condition, address, clock, etc.)

1.1 Notes

- In this lab, you will create some breadboard circuits with exposed pins and wires. Please be especially careful not to accidentally create connections that shouldn't be connected (e.g. short circuits). Also, check your work carefully before connecting any breadboard circuit to a board, to avoid damaging the board.
- Read each subsection of this lab manual in its entirety before you start following the instructions in it. Some instructions are modified by explanations that come afterwards.
- Although you may work with a partner, this collaboration is limited to discussion. Your partner is not allowed to construct or modify your circuit, log in to your Pi, or run commands or write code on your Pi. Similarly, you are not allowed to do these things for your partner. (You *are* encouraged to collaborate by screen-sharing or showing video of your circuit to debug and discuss problems together.)
- For your lab report, you must submit data, code, and screenshots from your own experiment. You are not allowed to use your lab partner's data, code, or screenshots.
- For any question in the lab report that is marked "Individual work", you should *not* collaborate with your lab partner or anyone else (even via discussion). You can use your notes, the lab manual, or the lecture slides and video to help you answer these questions.

1.2 Parts

In this lab, we'll use the following parts:



- Pi, SD card, and power supply. We will insert the SD card, connect the power supply, and log in to the Pi over SSH.
- Breadboard and jumper cables
- 0.96" OLED module

1.3 Procedure

1.3.1 Enable I2C on Raspberry Pi OS

The I2C driver is disabled by default on Raspberry Pi OS. To enable it, run

```
sudo raspi-config
```

and choose  , select "Yes", then "OK", then "Finish".

Verify that it is enabled by running

```
lsmod | grep i2c
```

This lists all currently loaded kernel modules (including drivers) and then filters the output to show only those with `i2c` in the name. You should see an entry for `i2c_dev` and `i2c_bcm2835`.

1.3.2 Install the library

We will use the Adafruit library for the SSD1306 device. To install it, run

```
mkdir ~/lab-i2c
cd ~/lab-i2c
git clone https://github.com/adafruit/Adafruit_Python_SSD1306.git
cd Adafruit_Python_SSD1306
sudo python3 setup.py install
```

1.3.3 Connect the SSD1306 OLED screen

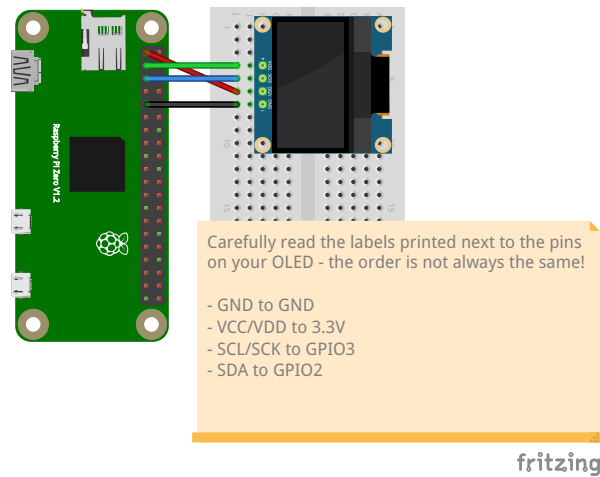


Figure 1: Connect the Pi to the OLED screen.

Connect the OLED screen to your Pi:

- **GND to GND**
- **VCC/VDD to 3.3V**
- **SCL/SCK to GPIO3** on the Pi
- **SDA to GPIO2** on the Pi

Carefully read the labels printed next to the pins on your OLED to identify each pin - the order is not necessarily the same as in the diagram.

Note: An I2C bus requires pull up resistors on the SDA and SCL lines. However, the Raspberry Pi already has pull-ups on its I2C lines, so we won't need external pull-up resistors.

1.3.4 Scan the I2C bus and find OLED screen address

When connecting a new I2C device, the first challenge is to make sure we can address it. We can use a utility called `i2cdetect` to scan the entire I2C address space, and look for a response from each address, to identify the address of every device connected to the bus.

Run

```
i2cdetect -y 1
```

Your output should look like this:

```
    0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
10:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
20:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
30:  --  --  --  --  --  --  --  --  --  --  --  3c  --  --  --
40:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
50:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
60:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
70:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
```

We can see that a device with I2C address 3c (the OLED screen) is connected to the bus. No other devices are connected.

The library we are using is configured to use the OLED screen at address 3c - if you open the file

```
~/lab-i2c/Adafruit_Python_SSD1306/Adafruit_SSD1306/SSD1306.py
```

you can find the line where the address is defined:

```
SSD1306_I2C_ADDRESS = 0x3C    # 011110+SA0+RW - 0x3C or 0x3D
```

If your OLED screen is at a different address, edit this line. Then run

```
cd ~/lab-i2c/Adafruit_Python_SSD1306
sudo python3 setup.py install
```

to reinstall the library.

1.3.5 Send text and images to the OLED display

The library we are using comes with some sample scripts we can use to send text and images to the OLED. Run

```
cd ~/lab-i2c/Adafruit_Python_SSD1306/examples
```

to change to this directory. Then open the file `image.py`.

By default, the sample scripts are set up to use a different version of the OLED screen - a 128x32 pixel version, while ours is 128x64. To change this, find the line

```
disp = Adafruit_SSD1306.SSD1306_128_32(rst=RST)
```

and add a # at the beginning to comment it out. Then find the line

```
# disp = Adafruit_SSD1306.SSD1306_128_64(rst=RST)
```

and remove the #. Save and close the file.

Finally run

```
python3 image.py
```

to run the script. You should see the image appear on the OLED.

1.3.6 Examine I2C communication in piscope

Run

```
sudo pigpiod
```

if it is not already running, and open PiScope:

```
piscope
```

Leave PiScope open to monitor the I2C bus on GPIO pins 2 and 3.

Run

```
python3 image.py
```

again, and use the zoom and arrow buttons to find the relevant parts of the I2C session. See if you can find one complete exchange, including both the start condition and stop condition, in the same screenshot. (You can use the first exchange, which is relatively short.)

1.3.7 Use terminal-based utilities

We can also interact with I2C devices directly from the terminal. For example, from the “Command table” on page 28 of the datasheet, we find that we can set the display on or off;

- 0xAE or 0b10101110 to set the display off
- 0xAF or 0b10101111 to set the display on

Make sure your OLED display is currently showing the image of the cat, and that `piscope` is running. Then, from a terminal, run

```
i2cset 1 0x3C 0x00 0xAE
```

and choose `y` when prompted to continue. You should see your OLED display go off.

The `i2cset` utility accepts four arguments:

- an integer identifying the I2C bus - in this case, 1
- the address of the I2C peripheral device to write to - here, 3C
- the register address to write to - here 00
- the data to write - here, AE

To turn your display back on, run

```
i2cset 1 0x3C 0x00 0xAF
```

and the cat image should re-appear.

Take a screenshot in `piscope` showing the complete `i2cset` exchange, where you turn the screen off.

Lab report: Annotate your `piscope` screenshot:

- Mark the start condition and stop condition.
- Number each clock cycle. Then label each byte with its value in binary and hex, and label each ACK/NACK bit - note whether it is ACK or NACK.
- Indicate which byte is the peripheral address, which byte is the register address, and which byte is the data.
- In the first byte, identify the 7-bit peripheral address, and the value of the R/W bit.

Lab report (individual work): Why do the clock pulses come in groups of 9, rather than 8 (i.e. one pulse per byte)?
