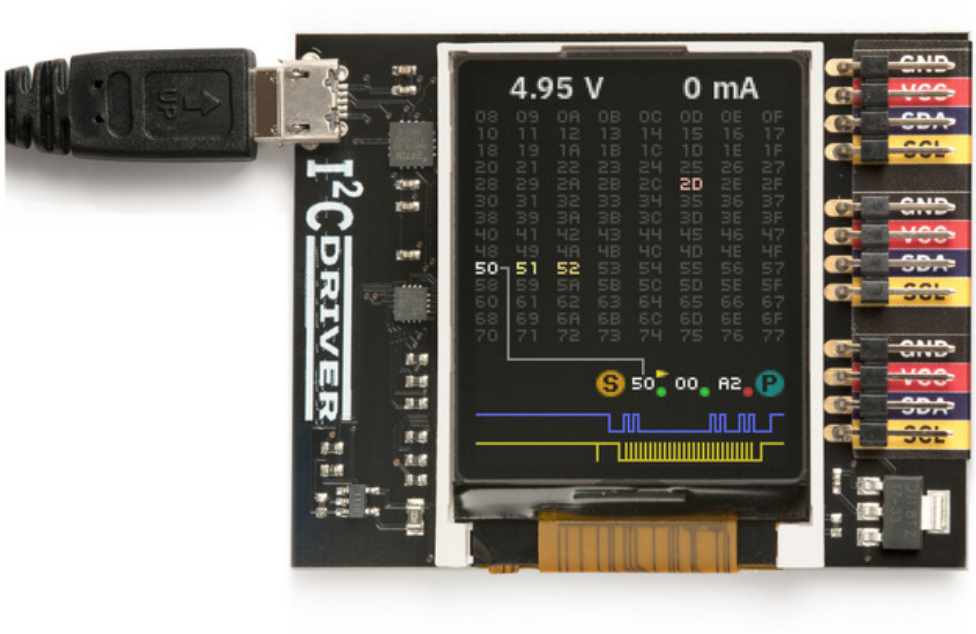
i2cdriver

Feb 07, 2023

Contents:



I²CDriver is an easy-to-use, open source tool for controlling I²C devices over USB. It works with Windows, Mac, and Linux, and has a built-in color screen that shows a live "dashboard" of all the I²C activity.

The I²CDriver User Guide has complete information on the hardware:

https://i2cdriver.com/i2cdriver.pdf

CHAPTER 1

System Requirements

Because it is a pure Python module, `i2cdriver` can run on any system supported by `pyserial`. This includes:

- Windows 7 or 10
- Mac OS
- Linux, including all Ubuntu distributions

Both Python 2.7 and 3.x are supported.

1.1 Installation

The `i2cdriver` package can be installed from PyPI using `pip`:

```
$ pip install i2cdriver
```

1.2 Quick start

To connect to the I2CDriver and scan the bus for connected devices:

```
>>> import i2cdriver
>>> i2c = i2cdriver.I2CDriver("/dev/ttyUSB0")
>>> i2c.scan()
-- -- -- -- -- -- -- --
-- -- -- -- -- -- -- --
-- -- -- -- 1C -- -- --
-- -- -- -- -- -- -- --
-- -- -- -- -- -- -- --
-- -- -- -- -- -- -- --
-- -- -- -- -- -- -- --
-- -- -- -- -- -- -- --
```

(continues on next page)

(continued from previous page)

```
48 -- -- -- -- -- -- --
-- -- -- -- -- -- -- --
-- -- -- -- -- -- -- --
-- -- -- -- -- -- -- --
68 -- -- -- -- -- -- --
-- -- -- -- -- -- -- --
[28, 72, 104]
```

To read the temperature in Celsius from a connected LM75 sensor:

```
>>> d=i2cdriver.EDS.Temp(i2c)
>>> d.read()
17.875
>>> d.read()
18.0
```

The User Guide at https://i2cdriver.com has more examples.

1.3 Module Contents

class i2cdriver.**I2CDriver**(*port='/dev/ttyUSB0'*, *reset=True*)

A connected I2CDriver.

Parameters

- **port** (*str*) – The USB port to connect to
- **reset** (*bool*) – Issue an I2C bus reset on connection

After connection, the following object variables reflect the current values of the I2CDriver. They are updated by calling *getstatus()*.

Variables

- **product** – product code e.g. 'i2cdriver1' or 'i2cdriverm'
- **serial** – serial string of I2CDriver
- **uptime** – time since I2CDriver boot, in seconds
- **voltage** – USB voltage, in V
- **current** – current used by attached device, in mA
- **temp** – temperature, in degrees C
- **scl** – state of SCL
- **sda** – state of SDA
- **speed** – current device speed in KHz (100 or 400)
- **mode** – IO mode (I2C or bitbang)
- **pullups** – programmable pullup enable pins
- **ccitt_crc** – CCITT-16 CRC of all transmitted and received bytes

setspeed(*s*)

Set the I2C bus speed.

Parameters **s** (*int*) – speed in KHz, either 100 or 400

setpullups(*s*)
Set the I2CDriver pullup resistors

Parameters **s** – 6-bit pullup mask

scan(*silent=False*)
Performs an I2C bus scan. If silent is False, prints a map of devices. Returns a list of the device addresses.

```
>>> i2c.scan()
-- -- -- -- -- -- -- --
-- -- -- -- -- -- -- --
-- -- -- -- 1C -- -- --
-- -- -- -- -- -- -- --
-- -- -- -- -- -- -- --
-- -- -- -- -- -- -- --
-- -- -- -- -- -- -- --
-- -- -- -- -- -- -- --
48 -- -- -- -- -- -- --
-- -- -- -- -- -- -- --
-- -- -- -- -- -- -- --
-- -- -- -- -- -- -- --
68 -- -- -- -- -- -- --
-- -- -- -- -- -- -- --
[28, 72, 104]
```

reset()
Send an I2C bus reset

start(*dev*, *rw*)
Start an I2C transaction

Parameters

- **dev** – 7-bit I2C device address
- **rw** – read (1) or write (0)

To write bytes `[0x12,0x34]` to device `0x75`:

```
>>> i2c.start(0x75, 0)
>>> i2c.write([0x12,034])
>>> i2c.stop()
```

read(*l*)
Read l bytes from the I2C device, and NAK the last byte

write(*bb*)
Write bytes to the selected I2C device

Parameters **bb** – sequence to write

stop()
stop the i2c transaction

regrd(*dev*, *reg*, *fmt='B'*)
Read a register from a device.

Parameters

- **dev** – 7-bit I2C device address
- **reg** – register address 0-255

- **fmt** – `struct.unpack()` format string for the register contents, or an integer byte count

If device 0x75 has a 16-bit unsigned big-endian register 102, it can be read with:

```
>>> i2c.regrd(0x75, 102, ">H")
4999
```

regwr(*dev*, *reg*, *vv*)
Write a device's register.

Parameters

- **dev** – 7-bit I2C device address
- **reg** – register address 0-255
- **vv** – value to write. Either a single byte, or a sequence

To set device 0x34 byte register 7 to 0xA1:

```
>>> i2c.regwr(0x34, 7, 0xa1)
```

If device 0x75 has a big-endian 16-bit register 102 you can set it to 4999 with:

```
>>> i2c.regwr(0x75, 102, struct.pack(">H", 4999))
```

monitor(*s*)
Enter or leave monitor mode

Parameters **s** – `True` to enter monitor mode, `False` to leave

getstatus()
Update all status variables

class i2cdriver.**START**(*addr*, *rw*, *ack*)

class i2cdriver.**STOP**

class i2cdriver.**BYTE**(*b*, *rw*, *ack*)

Index