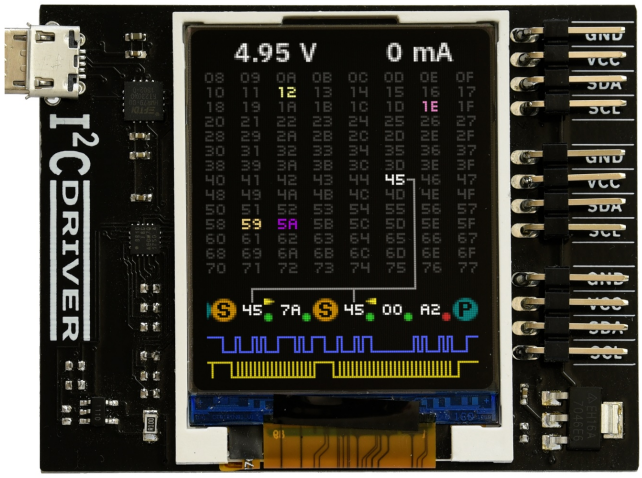


Last updated on 2019 年 6 月 23 日

Contents

1 概论

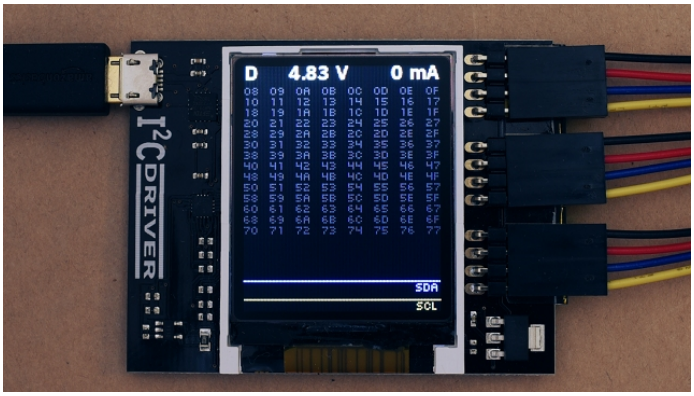
I²CDriver 是一个易于使用，开源的工具，用于控制I²C 设备. 它可以运行于Windows, Mac, and Linux 平台上。它内置彩色屏幕，可以实时显示所有的I²C 活动信息. 由于使用标准的FTDI USB 系列芯片与计算机通讯, 所以不需要另外安装驱动程序. 它还包括了额外的3.3 V 电源，方便用于监控电压和电流.

1.1 特性

- **实时显示**: 精确显示shows you exactly what it's doing all the time
- **支持所有I²C 的特性**: 7- 和10-比特I²C 寻址, 时钟拉伸, 总线仲裁,400至100 KHz的传送速率
- **I²C 上拉电阻**: 可编程自动调整I²C上拉电阻
- **USB 电压监控**: 侦测USB电压供应问题, 低至0.01v
- **目标设备电源监控**: 测量目标设备高侧电流, 精确至5 mA
- **三个I²C 端口**: 三个完全相同的I²C 端口, 各自有独立电源和I²C 信号
- **跳线**: 包括三组高质量彩色编码的100 毫米跳线
- **3.3 V 输出**: 3.3 V 伏特兼容5 伏特
- **高可靠性的组件**: 使用FTDI USB 串口适配芯片, Silicon Labs 汽车级EFM8 控制器
- **开源硬件**: 硬件设计，固件和工具都在BSD 协议下开源
- **控制灵活**: 图形界面, 命令行, C/C++, Python 2/3 ,适用于Windows, Mac, 和Linux

2 入门

当你第一次连接I²CDriver到USB 端口, 屏幕会闪烁白色后，显示如下:



连接三组不同颜色的跳线，按照如下的颜色标签:

GND	黑色
VCC	红色
SDA	蓝色
SCL	黄色

头两个信号分别传送3.3伏特电源

在屏幕的顶端，I²CDriver 不间断的显示所测量的USB 总线电压和输出电流.

3 所需软件安装

I²CDriver 软件源代码在此处. 包括:

- Windows/Mac/Linux 图形工具
- Windows/Mac/Linux 命令行工具
- Python 2 and 3 绑定库
- C/C++ 绑定(Windows/Mac/Linux)

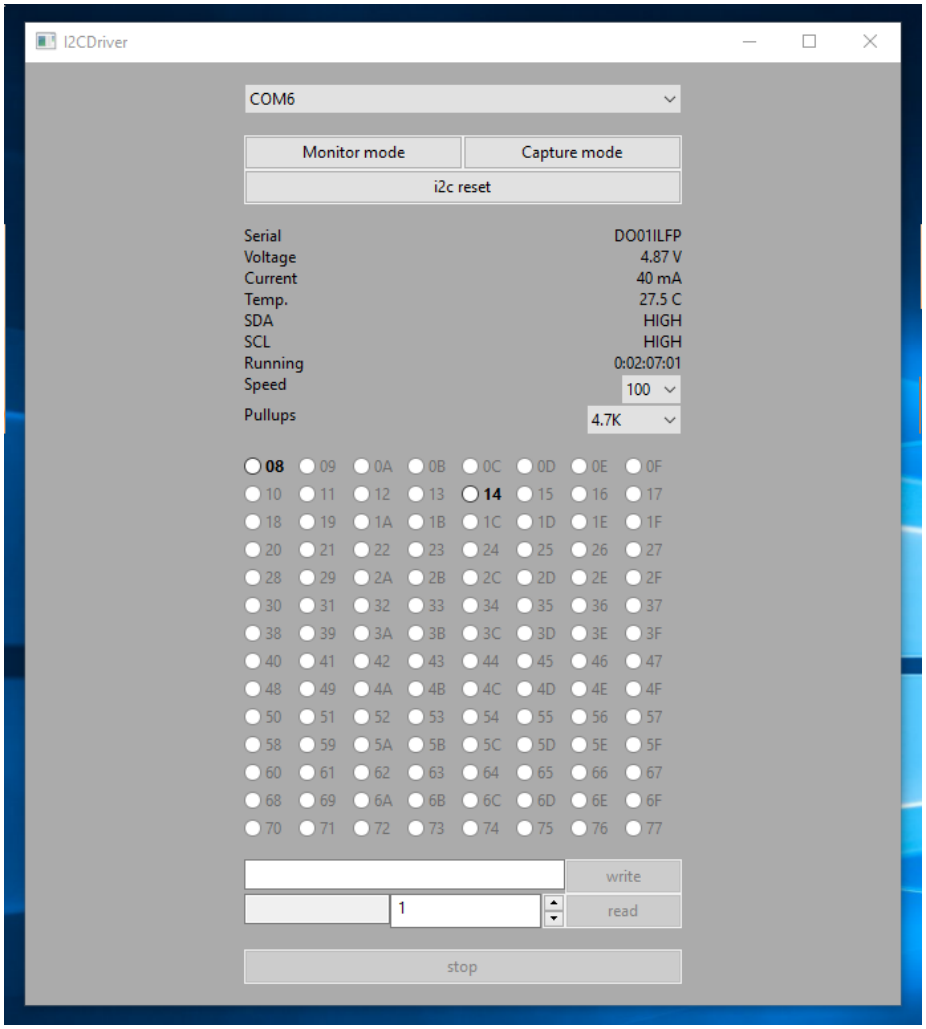
安装图形和命令行工具会由因不同平台而有所差异.

3.1 Windows

这里包含Windows下的图形和命令行工具安装 .

图形界面的快捷方式安装在桌面上：

启动后会开启控制窗口:



若只有一个设备连接，则该I²CDriver设备被自动选择若是超过一个设备，在顶端的下拉菜单中选择对应COM 端口所连接的I²C设备一旦建立连接，你可以选择一个连接的设备并进行读写数据.

命令行工具`i2ccl` 已经被安装. 比如显示状态信息:

```
c:\>"c:\Program Files\Excamera Labs\I2CDriver\i2ccl.exe" COM6 i
uptime 8991 4.957 V 30 mA 25.8 C SDA=1 SCL=1 speed=100 kHz
```

查看下面以获取更多命令行的用法.

3.2 Linux

图形界面工具包含在`i2cdriver` Python package, 同时兼容Python 2 and 3. 若要安装，打开一个shell prompt:

```
sudo pip install i2cdriver
```

然后运行:

```
i2cgui.py
```

使用命令行工具克隆I2CDriver 源码库, 然后:

```
cd i2cdriver/c
make -f linux/Makefile
sudo make -f linux/Makefile install
i2ccl /dev/ttyUSB0 i
```

你会看到如下:

```
uptime 1651 4.971 V 0 mA 21.2 C SDA=1 SCL=1 speed=100 kHz
```

3.3 MacOS

图形界面工具包含在`i2cdriver` Python package, 同时兼容Python 2 and 3. 若要安装，打开一个shell prompt:

```
sudo pip install i2cdriver
```

Then run it with

```
i2cgui.py
```

使用命令行工具，克隆I2CDriver 源码库, 然后:

```
cd i2cdriver/c
make -f linux/Makefile
sudo make -f linux/Makefile install
i2ccl /dev/cu.usbserial-DO00QS8D i
```

(用你真实的I²CDriver's ID 来替换DO00QS8D) 你会看到:

```
uptime 1651 4.971 V 5 mA 21.2 C SDA=1 SCL=1 speed=100 kHz
```

请注意你所使用的端口是/dev/cu.usbserial-XXXXXXXX, 如这里 所解释的那样.

4 APIs

4.1 Python 2 和3

I²CDriver 绑定可以通过pip 安装:

```
pip install i2cdriver
```

然后你就可以用Python读取LM75B 温度传感器:

```
>>> import i2cdriver
>>> i2c = i2cdriver.I2CDriver("/dev/ttyUSB0")
>>> d=i2cdriver.EDS.Temp(i2c)
>>> d.read()
17.875
>>> d.read()
18.0
```

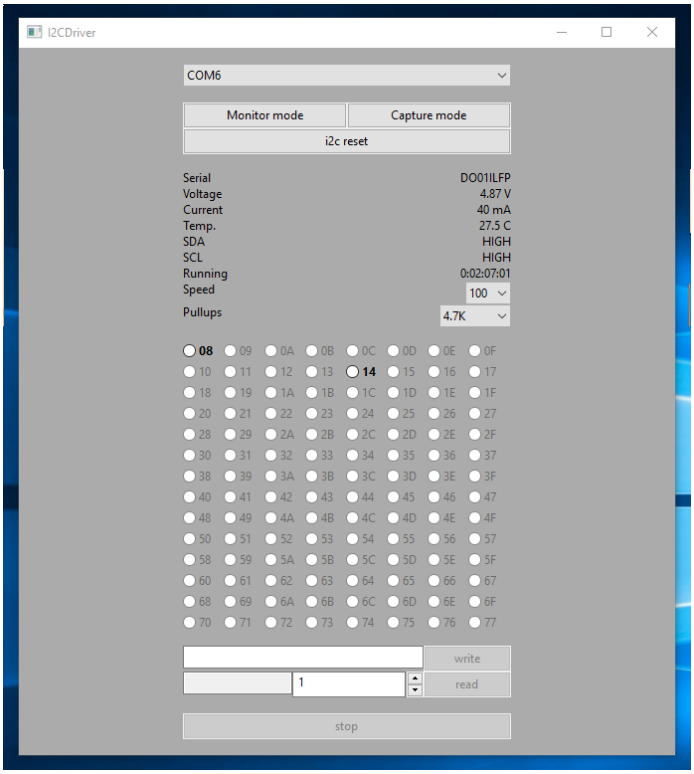
你可以打印出总线信息，使用:

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

Python 图形界面(使用wxPython) 可以这样启动:

```
python i2cgui.py
```

根据你所安装的系统不同，会有:



更多例子请参考: samples folder in the repository.

该模块带有丰富的帮助文档:

```
>>> help(i2cdriver)
```

可以显示API文档.

4.1.1 参考

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

A connected I2CDriver.

Variables

- `product` – product code e.g. 'i2cdriver1'
- `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

`__init__(`*port='/dev/ttyUSB0'*, *reset=True*`)`

Connect to a hardware i2cdriver.

Parameters

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

`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])
```

(continues on next page)

(continued from previous page)

```
>>> 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

If device 0x75 has a 16-bit 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` – sequence of values to write

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

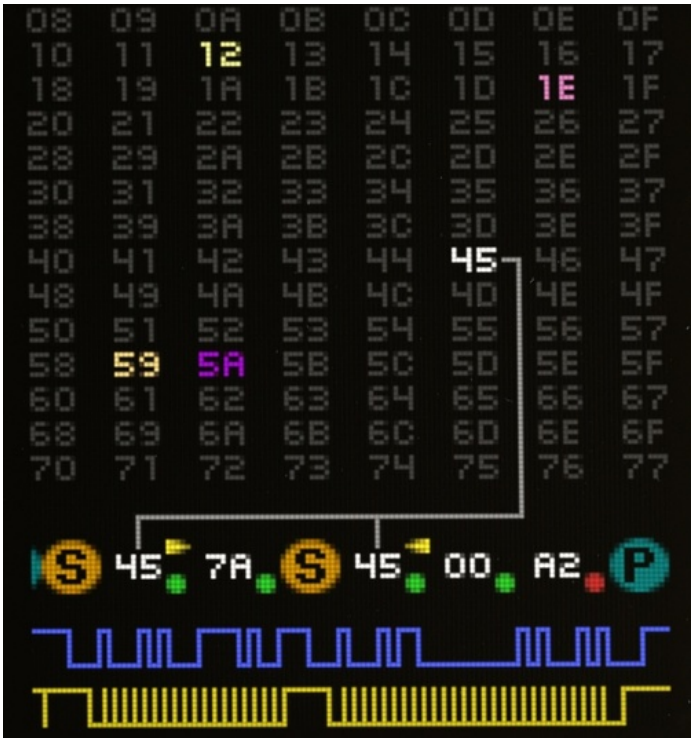
4.2 C/C++

I²CDriver 包含在一个拥有单一头文件的源文件中. 二者都在这个位置. 用法和Python API 类似且容易明白.

5 使用I²CDriver

5.1 显示

屏幕上的显示有三个部分. 第一个部分是热量图，显示所有112个合法I²C 地址. 正在活跃的设备是白色的. 不活跃的设备会淡化成黄色, 紫色, 直至蓝色. 中间部分是当前I²C 信息的符号解释。细节如下。 最底下两行代表SDA (蓝色) 和SCL (黄色) 信号.



符号解释部分显示出I²C的交易. 开始和停止如下所示：(S) 和(P) 符号. 在(S) 之后，地址字节显示出来(向右的箭头代表写,而向左的箭头代表读). 有灰色线连接到热量图中的地址字节. 随后的是一系列的数据字节. Following this is a series of data bytes. 每一字节是16进制, 均带有绿色点(ACK)，或是红色点(NACK).

所以以上的例子表明:

- 开始, 写地址45
- 写字节7A
- 重复开始从地址45读
- 读到字节00
- 读到字节A2
- 停止

上述的顺序是非常典型的从I²C读寄存器的操作。请注意最终的NACK(红色点)并非错误条件，而是读操作中最后一个字节的标准方法。

5.2 图形界面

(待完成)

5.3 命令行工具i2ccl

i2ccl 在所有平台上都是相同的. 头一个参数是串口，具体取决于你的操作系统。所有后面的参数都是控制参数，他们是:

i	显示状态信息(uptime, 电压, 电流, 温度)
d	设备扫描
w *dev bytes*	写*bytes* 到I²C 设备*dev*
p	发送一个STOP
r *dev N*	读N bytes 从I²C 设备*dev*, 然后STOP
m	进入I²C 总线监控模式

例如下面的例子:

```
i2ccl /dev/ttyUSB0 r 0x48 2
```

从地址0x48的I²C 设备读取两个字节。

所以从LM75B 温度传感器 你会看到:

```
0x16,0x20
```

表示温度是22 °C.

I²C 通常有多个寄存器. 若要读LM75B的寄存器3，你需要首先写寄存器地址3, 然后读两个字节:

```
i2ccl /dev/ttyUSB0 w 0x48 3 r 0x48 2
0x50,0x00
```

这显示寄存器3的内容是0x5000.

5.4 监控模式

在监控模式, I²CDriver 并不会通过I²C总线写入任何数据. 相反的是，它监控总线的流量并在显示屏幕上画出来。这使它成为一个调试和纠错I²C软硬件的理想工具。要显出它是否在监控模式，I²CDriver 屏幕的左上侧的字符会从D切换到M.

有以下方式可以进入监控模式:

- 通过命令行工具:

  ```
  i2ccl m
  ```

- 在图形界面，在”Monitor” 的选项上打勾
- 在Python:

  ```
  i2c.monitor(True)
  ```

  然后退出监控模式:

```
i2c.monitor(False)
```

- 开启一个终端并于I²CDriver (at 1000000 8N1) 连接，键入m 字符, 然后键入任何字符以退出监控模式

5.5 捕获模式

在捕获模式，I²CDriver 并不通过I²C 总线写任何数据. 相反，它监控总线交通并把数据传回PC，作为记录.

5.5.1 命令行

有一个Python 的例子程序用于在命令行捕获数据。capture.py.

以地址作为参数就可以把I²CDriver放入到捕获模式：Running it with the I²CDriver address as an argument puts the I²CDriver into capture mode: 屏幕的左上侧字符会从D 变成C.

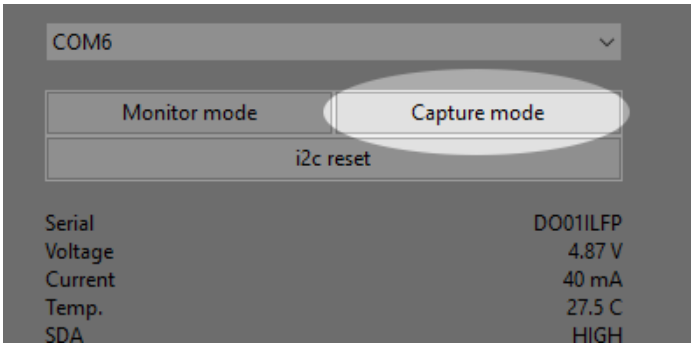
```
$ python samples/capture.py /dev/ttyUSB0

Now capturing traffic to
    standard output (human-readable)
    log.csv
Hit CTRL-C to leave capture mode
<START 0x14 WRITE ACK>
<WRITE 0x02 ACK>
<WRITE 0x22 ACK>
<STOP>
^C
Capture finished
```

运行的时候，它显示所有的流量到标准输出终端。它也会把总结写入log.csv, 它可以被其他任何接受CSV 格式的工具检验和处理

	A	B	C	D
1	START	WRITE	20	ACK
2	BYTE	WRITE	2	ACK
3	BYTE	WRITE	34	ACK
4	STOP			
5				
6				

5.5.2 图形工具

图形工具也支持捕获到CSV文件。



点击”Capture mode”开始捕获，提示用户选择目标CSV文件屏幕的左上侧字符会从D 变成C. 捕获一直持续到再次点击“Capture mode”

6 范例

Python `samples` 目录包含多个短小的例子，均使用来自Electric Dollar Store I²C 的模块:

模块	功能	范例
DIG2	2-digit 7-seg display	EDS-DIG2.py
LED	RGB LED	EDS-LED.py
POT	potentiometer	EDS-POT.py
BEEP	Piezo beeper	EDS-BEEP.py
REMOTE	IR remote receiver	EDS-REMOTE.py
EPROM	CAT24C512 64 Kbyte EPROM	EDS-EPROM.py
MAGNET	LIS3MDL magnetometer	EDS-MAGNET.py
TEMP	LM75B temperature sensor	EDS-TEMP.py
ACCEL	RT3000C Accelerometer	EDS-ACCEL.py
CLOCK	HT1382 real-time clock	EDS-CLOCK.py

所有的范例都使用I²CDriver 的命令行程序. 例如:

```
python EDS-LED.py COM16
```

也包含了一些小程序，演示了各类模块的组合.

6.1 彩色罗盘

源代码: EDS-color-compass.py

彩色罗盘使用了LED和磁铁, 读取当前磁场方向并在LED上渲染成某一颜色。当你旋转这个模块时，颜色会随之变化。假如有一特别的方向时纯红色，其他颜色就代表了其他的方向。代码会读取磁场的方向，并把数值定义到0-255 之间，设置LED为相应的色彩。

6.2 鸡蛋计时器

源代码: EDS-egg-timer.py

该例子使用POT, DIG2 和BEEPER 模块制作一个简单的厨房蛋形定时器。旋转POT 设置倒数秒数。一旦释放，就开始计时。当它到达0时，开始闪烁，并响铃。

6.3 Take-a-ticket

Source code: EDS-take-a-ticket.py

该范例展示了商铺或熟食柜台所使用的Take-a-ticket显示。This demo runs a take-a-ticket display for a store or deli counter, 它使用了REMOTE, DIG2 and BEEP 模块. 它显示两位“正在服务”号码，每次遥控器的‘+’键被按动，它就会加一并发出蜂鸣声，下一个客户就可以被提醒。每次遥控器的‘-‘键被按动，它就会减一并发出蜂鸣声，

7 注意事项

7.1 端口名称

I²CDriver 所显示的串口名称取决于你所在的操作系统

在**Windows**, 它的名字可能会是COM1, COM2, COM3 等. 你可以使用设备管理器或者MODE 命令来显示可用端口名称该文章 描述了如何为某一设备设置固定端口.

在**Linux** 上, 端口名称显示为/dev/ttyUSB0, 1, 2 等. 实际的号码取决于设备连接的顺序。然而它也有可能显示如下端口名称：

```
/dev/serial/by-id/usb-FTDI_FT230X_Basic_UART_DO00QS8D-if00-port0
```

DO00QS8D 就是I²CDriver 的端口号码(在每一个I²CDriver设备的下方打印出来). 这个长一些，但是它可以确保每一个设备有相同的名称。

与**Mac OS** 类似, I²CDriver 设备显示为/dev/cu.usbserial-DO00QS8D.

7.2 减少USB 延迟时间

通过设置USB 的延迟时间到最小1 ms, I²CDriver 的性能可以得以提高。这可以提高双向的I²C 流量高达10倍

在**Linux** 平台上:

```
setserial /dev/ttyUSB0 low_latency
```

在**Windows** 和**Mac OS** 平台，参考该文章.

7.3 温度传感器

温度传感器位于EFM8控制器内，在出厂校正误差为2 °C.

7.4 原始协议

I²CDriver 使用了一套串行协议，收发I²C 命令. 以1M baud, 8 bits, no parity, 1 stop bit (1000000 8N1) 的配置，连接I²CDriver 因为很多I²CDriver 命令是ASCII字符，你可以采用支持1M波特率的终端程序，以互动的方式来控制它。比如，键入u 和s 切换CS 信号线，而键入? 显示状态信息.

命令如下:

?	状态信息
e *byte*	回显*byte*
1	设置速度到100 KHz
4	设置速度到400 KHz
s *addr*	发送START/addr, 返回状态
0x80-bf	读取1-64 字节, 以NACK 结尾
0xc0-ff	写入1-64 字节
a *N*	读取N 个字节, 每一个字符都有ACK
p	发送STOP
x	重置I²C 总线
r	读取寄存器
d	扫描设备，返回112 状态字节
m	进入监控模式
c	进入捕获模式
b	进入bitbang 模式
i	离开bitmang 模式, 返回I²C 模式
u *byte*	设置上拉控制线
v	开始电压测量
w	读取电压测量结果

以下的例子:

主设备端应该发送:

`s 0x90`	开始写入设备45
`0xc0 0x7a`	写入一个字节
`s 0x91`	开始读取设备45
`0x80`	读取一个字节
`p`	停止

回应状态总是80个字符（以空格填充不足或断句），比如::

```
[i2cdriver1 DO01JUOO 000000061 4.971 000 23.8 I 1 1 100 24 ffff        ]
```

按照以下的方式解释：

identifier	always `i2cdriver1`
serial	serial code identifier
uptime	I²CDriver uptime 0-999999999, in seconds
voltage	USB bus voltage, in volts
current	attached device current, in mA
temperature	junction temperature, in °C
mode	current mode, `I` for I²C, `B` for bitbang
SDA	SDA line state, 0 or 1
SCL	SCL line state, 0 or 1
speed	I²C bus speed, in KHz
pullups	pullup state byte
crc	16-bit CRC of all input and output bytes (CRC-16-CCITT)

The sample `confirm.py` shows the CRC-16-CCITT calculation.

7.5 上拉电阻

I²CDriver 拥有六个可编程上拉电阻, SDA 和SCL各有三个. 共有六个控制位可以启用或禁用上拉电阻:

bit	resistor
0	2.2K to SDA
1	4.3K to SDA
2	4.7K to SDA
3	2.2K to SCL
4	4.3K to SCL
5	4.7K to SCL

启动后两个4.7K 电阻是启用的。通过设置这些电阻的不同组合，一系列的上拉strength可以被实现：

4.7K	4.3K	2.2K	pull-up strength
0	0	0	0 (i.e. no pull-up)
0	0	1	2.2K
0	1	0	4.3K
0	1	1	1.5K
1	0	0	4.7K
1	0	1	1.5K
1	1	0	2.2K
1	1	1	1.1K

所以就有下面的控制组合:

3-bit value	Resistance
0	0
1	2.2K
2	4.3K
4	4.7K
5	1.5K
7	1.1K

在Python, 上拉电阻可以由`setpullups()` 方法来控制, 可以通过读取`pullups` 变量得到状态。二者都如上所示。

图形界面工具也提供了上拉电阻的控制。它为SDA和SCL设置同样的上拉力量。

7.6 规范

7.6.1 DC 特性

	min	typ	max	units
电压精确度		0.01		V
电流精确度		5		mA
温度精确度		± 2		°C
SDA,SCL				
低压			0.6	V
高压	2.7		5.8	V
输出电流			470	mA
输出功率		25		mA

7.6.2 AC 特性

	min	typ	max	units
I²C 速度	100		400	Kbps
运行时间精度		150		ppm
累计运行时间重置		31.7		years
开机时间			200	ms

8 技术支持

如需技术和产品支持，请电邮: support@i2cdriver.com

I²CDriver 由Excamera Labs 制造和维护.

Index