Connecting multiple sensors over I2C in embedded Linux: an example using a LIDAR-Lite and PX4Flow

Trevor Avant

Abstract

This document describes the process of connecting two sensors, a LIDAR-Lite and a PX4Flow optic flow sensor, together over I2C to a Gumstix embedded computer.

Hardware

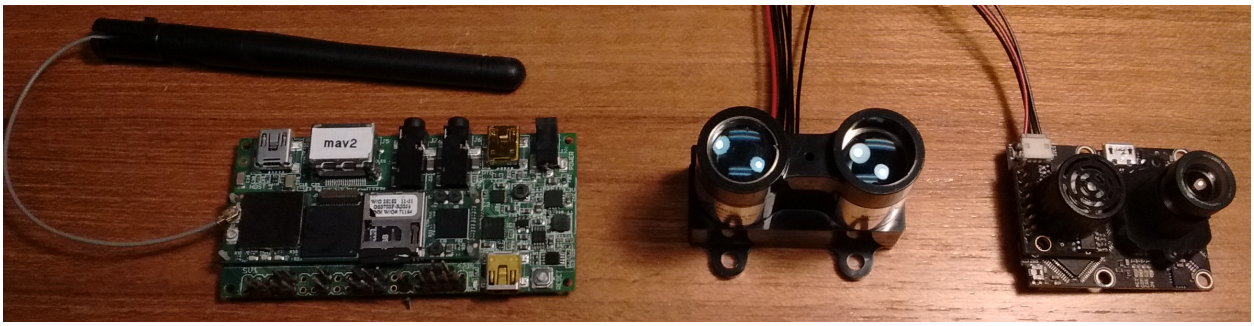

Figure 1: Hardware used in the project. Left to right: Gumstix with expansion board, LIDAR-Lite (version 1), PX4Flow (version 1.3).

We used a Gumstix Overo FireSTORM COM embedded computer which runs a Linux operating system. We connected this device to a Gumstix SUMMIT expansion board to easily access the I2C output and other outputs. More specifically, the Overo's i2c-3 bus can be accessed via the 40 pin header on the expansion board. The goal of the project was to connect two sensors to the embedded computer over I2C: a LIDAR-Lite (version 1) and a PX4Flow optic flow sensor (version 1.3).

One of the main challenges of interfacing the sensors with the embedded computer is that the i2c-3 bus on the Overo uses 1.8V logic (meaning a binary 1 is represented as 1.8V and a binary 0 is represented as 0V). However, the LIDAR-Lite and PX4Flow both use a 5V logic I2C. Accordingly, it is necessary to use a logic level shifting circuit to convert the 1.8V logic on the Overo to the 5V logic on the sensors.

I2C in Embedded Linux

In Overo's Linux environment, we wrote all code to do the I2C communication in user space in the C language. In Linux, communication over I2C can be accomplished by using the `ioctl` function, which can be included in a C file by the command `#include <sys/ioctl.h>`.

Using `ioctl` to perform I2C operations in user space is not obvious. We have found that a good way to learn how is by examining the i2c-tools package. The i2c-tools source code can be downloaded using `sudo apt-get source i2c-tools`. A particularly useful file in the source code is /include/linux/i2c-dev.h, which shows how to use I2C commands with SMBus functionality. The i2c-tools package can be downloaded and installed, but we chose to use just copy the declarations and definitions we needed from /include/linux/i2c-dev.h. Doing it this way makes it a little more transparent as to how `ioctl` is being used. For information about the C code we used, see the Appendix.

LIDAR-Lite and PX4Flow over I2C

In all I2C communications, the master (in this case the Overo) must specify whether the command sent to the slave (in this case the LIDAR-Lite or PX4Flow) is a "read" or "write" operation. The format of these read and write operations are dictated by the slave device, and can be determined from the slave device's documentation.

In order to obtain sensor data from the PX4Flow, the master must send a write command specifying what information is to be obtained, and then send a read command in which the master reads the appropriate data. There can be no stop bit between the write and read commands (for definition of a "stop bit" in I2C protocol, consult a reference on I2C communication). This write/read sequence can be accomplished using the `I2C_RDWR` protocol of the ioctl function (i.e. a command of the form `ioctl(file, I2C_RDWR, struct i2c_rdwr_ioctl_data *msgset)`). This function is designed to send combined commands consisting of read and write operations to a device, with no stop bits in between. Note that the `I2C_SMBUS` protocol of `ioctl` (i.e. commands of the form `ioctl(file, I2C_SMBUS, struct i2c_smbus_ioctl_data *args)`) cannot be used as we do not believe this function can implement combined read/write sequences with no stop bits in between. Also note that the PX4Flow does not require read and write operations to be sent to specific registers on the device.

To obtain sensor data from the LIDAR-Lite over I2C, the master must send a write command, wait some amount of time for the sensor to do some processing, and then send a read command to finally obtain the data. This requires separate write and read commands to be sent to the LIDAR-Lite (unlike the combined write/read operation for the PX4Flow). The `I2C_SMBUS` protocol of the `ioctl` function can be used to obtain this data. As noted in the `ioctl` function's documentation, the `I2C_SMBUS` protocols are "not meant to be called directly", and instead should be called with the help of an access function.

Logic Level Shifting Circuit

As previously noted, the Overo has 1.8V logic while the LIDAR-Lite and PX4Flow use 5V logic. I2C communication between a master and slave with the same logic voltages simply requires the use of two pull-up resistors: one from the supply voltage to the clock line, and one from the supply voltage to the data line. When the master and slave operate at different logic levels, as in this case, a more complicated circuit is required to ensure the logic signals match.

There are a variety of circuits which can perform logic level shifting. We used the circuit described in application note AN10441 provided by NXP (the company which originally created I2C). This circuit uses MOS-FETs in combination with four pull-up resistors. Choosing values for pull-up resistors depends on properties of master and slave devices such as the bus capacitance and the values of internal pull-up resistors on either the master or slave (if such internal pull-ups exist or are enabled). The basic limiting factors for pull-up resistors is as follows.

- *pull-up resistance too low*: the clock and data lines cannot obtain a low enough voltage to register as a logic 0
- *pull-up resistance too high*: the clock and data lines will not be able to return to a high state (logic 1) from a low state (logic 0) fast enough before the next clock cycle begins

It is possible to calculate appropriate values for pull-up resistors based on characteristics of the circuit (e.g. internal pull-up resistance, bus capacitance, etc.). However, depending on the circuit you are working with, this may be a challenge if the circuit is complex or has unknown characteristics (e.g. a manufacturer does not specify if a sensor has internal pull-up resistors). Instead of calculating pull-up values directly, we used a trial and error method by building a logic level shifting circuit with pull-up resistors of variable resistance. Thus, we built the circuit in application note AN10441, and used potentiometers wired as variable resistors for the pull-up resistors (those labeled as R_p in NXP's diagram). This circuit is shown in Fig. 2. Note that Adafruit makes a breakout board which replicates the AN10441 circuit with 10k resistors (`https://www.adafruit.com/product/757`). However, as will be shown in the next section, fixed 10k resistors will not work for all sensor configurations in this project.

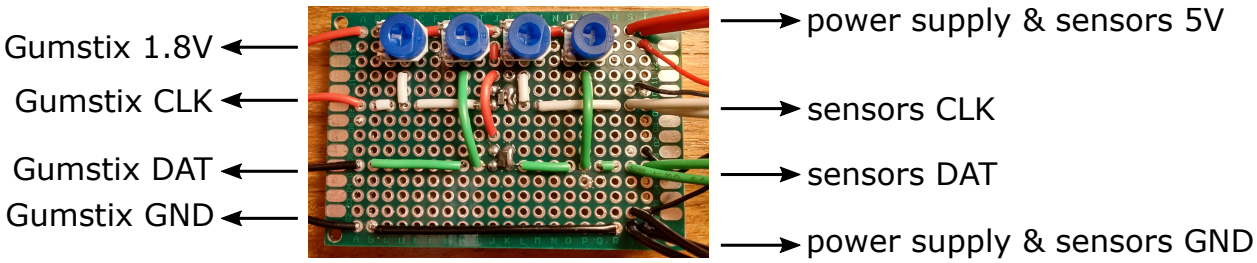

Figure 2: Logic level shifting circuit in NXP's application note AN10441, with variable resistors (blue components) used as the pull-up resistors R_p. The MOS-FETs are the small, black, rectangular components. CLK denotes the I2C clock line, DAT denotes the I2C data line, and "sensors" denotes the LIDAR-Lite and PX4Flow.

Experimental Investigation of the Circuit

Working Pull-Up Resistances

Using the aforementioned circuit with variable resistors, it is easy to test the range of working pull-up resistors for different sensor combinations. Note that all tests in this section were performed at an I2C clock speed of 400 kbit/sec. Table 1 shows the working ranges for different sensor configurations. Interestingly, any sensor

Device Tested	Devices Connected	Minimum Pull-Up (Ω)	Maximum Pull-Up (Ω)
LL	LL	1k	∞
PX	PX	5k	18k
LL	LL & PX	1k	∞
PX	LL & PX	12k	∞

*"LL" denotes LIDAR-Lite and "PX" denotes PX4Flow

Table 1: Minimum and maximum pull-up resistor values R_p for working operation of I2C devices using the level shifting circuit in NXP's AN10441 application note. A value of ∞ denotes that the pull-up resistors have been removed entirely from the circuit (i.e. an open circuit).

configuration in which the LIDAR-Lite is connected will work even if there is an infinite pull-up resistance, i.e. all four pull-up resistors are removed from the circuit entirely! Although this may seem perplexing at first, the 1.8V side of the clock and data lines is still connected to the 5V side through the MOS-FETs. We believe the circuit still works due to internal pull-up resistors on the computer and/or sensors. As a next step, we will gather more insight into the inner workings of the I2C circuit by probing it with an oscilloscope.

LIDAR-Lite

Fig. 3 shows the clock and data lines when only the LIDAR-Lite is connected by itself. First, let's consider the clock lines. There are two main trends to compare in the different plots. The first is how close to low (0V) the waveform gets. With the $< 1k\Omega$ pull-ups, the voltage only gets down to about 2.5V, which is why the I2C communication fails. For the rest of the pull-up values, the clock line gets very close to 0V. The other trend is how fast the waveform reaches high (5V). It is clear that as the pull-up resistance is increased, the clock line is slower to rise to 5V. And actually, with $5k\Omega$ and $\infty k\Omega$ pull-ups, the clock line is so slow to rise it can't even reach 5V in one clock cycle. Interestingly, despite the imperfect waveforms of the $5k\Omega$ and $\infty k\Omega$ plots, the I2C communications still work. Additionally, note that the beginning of the waveform for the $\infty k\Omega$ pull-ups show that the high voltage has actually dropped to about 3V.

Now let's consider the data line in the plots of Fig. 3. The data line shows similar trends to the clock line. As the pull-up resistors are increased, the waveform is slower to rise to 5V. For the $\infty k\Omega$ resistance, the voltage only reaches about 3V. However, unlike the clock line, the data line always maintains a low voltage close to 0V.

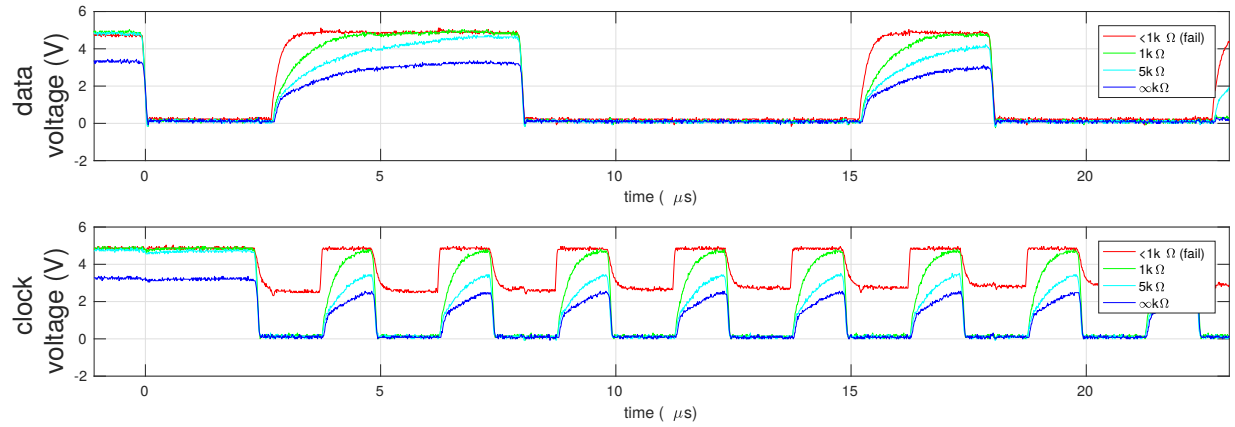

Figure 3: I2C waveform for various pull-up resistors when the LIDAR-Lite is connected by itself. Note that the bits of data transfer can be deduced by comparing the clock cycles to the data line. This sequence is 1100010...

PX4Flow

Let's also look at the clock and data waveforms when the PX4Flow is connected by itself. The waveforms are shown in Fig. 4. Similar trends are apparent in these waveforms compared to those of the LIDAR-Lite.

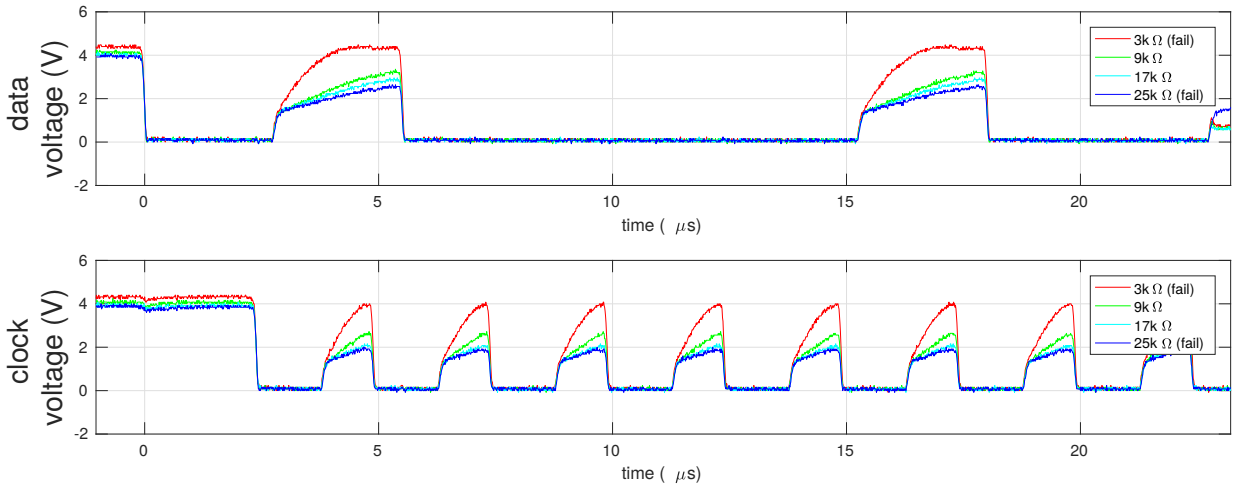

Figure 4: I2C waveform when the PX4Flow is connected by itself.

For example, large pull-up resistors lead to slower transitions from logic low to logic high. Also note that in the $3k\Omega$ case, the circuit has failed, even though the clock and data lines still seem to get close to 0V at low voltage. It is most likely that this case failed because the low voltage was too high, but it cannot be distinguished by the naked eye. Unlike the LIDAR-Lite, the PX4Flow does have a maximum working resistance. For the $9k\Omega$, $17k\Omega$ and $25k\Omega$ waveforms, the clock and data lines only reach about half of the high voltage before they are brought back down to 0V by the next clock cycle. However, the $9k\Omega$ and $17k\Omega$ waveforms are successful while the $25k\Omega$ is not. These results suggest that a waveform which appears to be quality, as in the $3k\Omega$ case, may not actually be successful. On the other hand, a waveform which does not appear to be quality, as in the $9k\Omega$ and $17k\Omega$ case, may be successful. Therefore, it may not be possible to judge the quality of an I2C communication based on the waveforms alone.

Effect of Clock Speed

Success of I2C communications depend on the I2C clock speed. A configuration which does not work at a fast clock speed may work if the speed is decreased. This is because the waveform will have more time to rise from low voltage. The disadvantage of reducing the clock speed is that communication is slower. An example is shown for the PX4Flow in Fig. 5. Since the pull-up resistors are the same in Fig. 5, the

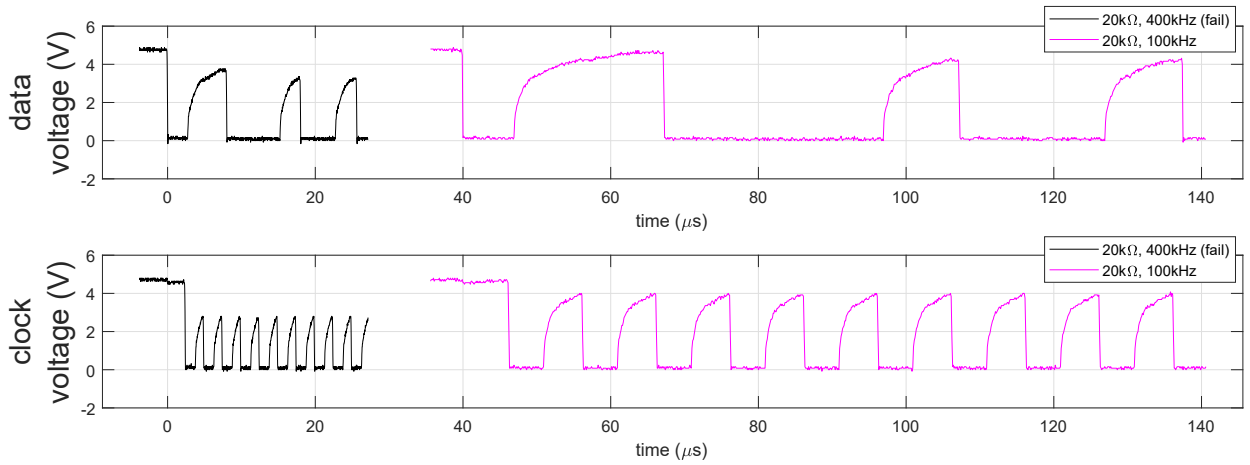

Figure 5: When the PX4Flow is connected by itself, $20k\Omega$ pull-up resistors prevent the clock and data lines from nearing high voltage. However, if the clock speed is reduced, the lines have more time to rise, and the circuit works.

waveforms follow the same curve as they rise from low voltage. To see this, we can superimpose the clock and data curves, which is show in Fig. 6.

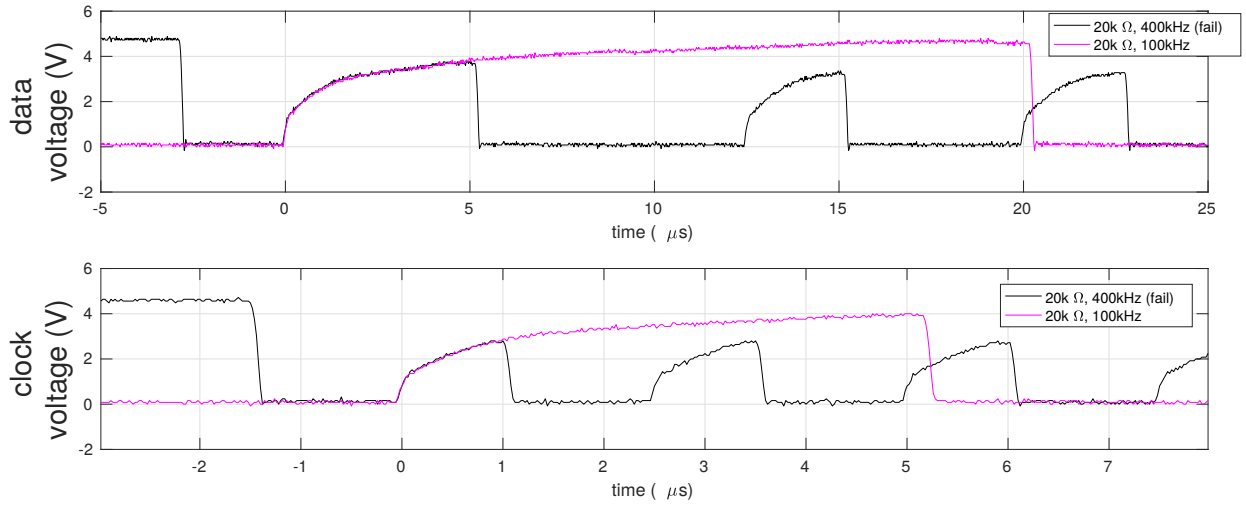

Figure 6: The same waveform as Fig. 5, but with the clock and data lines shifted to show that the waveforms rise from low voltage along the same curve.

Conclusion

We have described the process of connecting two sensors over I2C to a computer running embedded Linux. Since the sensors operate at different logic voltage levels than the computer, a converter circuit is required.

Due to certain characteristics of the computer and sensors (e.g. whether there are internal pull-ups on any of the devices), it may be unclear which size pull-up resistors are needed for the circuit. So, it is helpful to first build the circuit with variable resistors so that appropriate values can be determined.

Appendix

The C codes we used to communicate with the LIDAR-Lite and PX4Flow (named LL.c and PX.c) are located in the same directory as this PDF.