AdafruitPureIO Library Documentation

Release 1.0

Melissa LeBlanc-Williams

May 25, 2023

Contents

Pure python (i.e. no native extensions) access to Linux IO including I2C and SPI. Drop in replacement for smbus and spidev modules.

CHAPTER 1

Dependencies

This driver depends on:

- Python 3.5 or higher

CHAPTER 2

Installing from PyPI

On supported GNU/Linux systems like the Raspberry Pi, you can install the driver locally from PyPI. To install for current user:

```
pip3 install Adafruit-PureIO
```

To install system-wide (this may be required in some cases):

```
sudo pip3 install Adafruit-PureIO
```

To install in a virtual environment in your current project:

```
mkdir project-name && cd project-name
python3 -m venv .env
source .env/bin/activate
pip3 install Adafruit-PureIO
```

CHAPTER 3

Contributing

Contributions are welcome! Please read our Code of Conduct before contributing to help this project stay welcoming.

CHAPTER 4

Documentation

For information on building library documentation, please check out this guide.

CHAPTER 5

Table of Contents

5.1 Adafruit_PureIO.smbus

Pure python (i.e. no native extensions) access to Linux IO I2C interface that mimics the Python SMBus API.

- Author(s): Tony DiCola, Lady Ada, Melissa LeBlanc-Williams

5.1.1 Implementation Notes

Software and Dependencies:

- Linux and Python 3.5 or Higher

class Adafruit_PureIO.smbus.**SMBus**(*bus=None*)

I2C interface that mimics the Python SMBus API but is implemented with pure Python calls to ioctl and direct /dev/i2c device access.

close()

Close the smbus connection. You cannot make any other function calls on the bus unless open is called!

open(*bus*)

Open the smbus interface on the specified bus.

process_call(*addr*, *cmd*, *val*)

Perform a smbus process call by writing a word (2 byte) value to the specified register of the device, and then reading a word of response data (which is returned).

read_block_data(*addr*, *cmd*)

Perform a block read from the specified cmd register of the device. The amount of data read is determined by the first byte send back by the device. Data is returned as a bytearray.

read_byte(*addr*)

Read a single byte from the specified device.

read_byte_data(*addr*, *cmd*)

Read a single byte from the specified cmd register of the device.

read_bytes(*addr*, *number*)
Read many bytes from the specified device.

read_i2c_block_data(*addr*, *cmd*, *length=32*)
Perform a read from the specified cmd register of device. Length number of bytes (default of 32) will be read and returned as a bytearray.

read_word_data(*addr*, *cmd*)
Read a word (2 bytes) from the specified cmd register of the device. Note that this will interpret data using the endianness of the processor running Python (typically little endian)!

write_block_data(*addr*, *cmd*, *vals*)
Write a block of data to the specified cmd register of the device. The amount of data to write should be the first byte inside the vals string/bytearray and that count of bytes of data to write should follow it.

write_byte(*addr*, *val*)
Write a single byte to the specified device.

write_byte_data(*addr*, *cmd*, *val*)
Write a byte of data to the specified cmd register of the device.

write_bytes(*addr*, *buf*)
Write many bytes to the specified device. buf is a bytearray

write_i2c_block_data(*addr*, *cmd*, *vals*)
Write a buffer of data to the specified cmd register of the device.

write_quick(*addr*)
Write a single byte to the specified device.

write_word_data(*addr*, *cmd*, *val*)
Write a word (2 bytes) of data to the specified cmd register of the device. Note that this will write the data in the endianness of the processor running Python (typically little endian)!

class Adafruit_PureIO.smbus.**i2c_msg**
Linux i2c_msg struct.

class Adafruit_PureIO.smbus.**i2c_rdwr_ioctl_data**
Linux i2c data struct.

Adafruit_PureIO.smbus.**make_i2c_rdwr_data**(*messages*)
Utility function to create and return an i2c_rdwr_ioctl_data structure populated with a list of specified I2C messages. The messages parameter should be a list of tuples which represent the individual I2C messages to send in this transaction. Tuples should contain 4 elements: address value, flags value, buffer length, ctypes c_uint8 pointer to buffer.

5.2 Adafruit_PureIO.spi

Pure python (i.e. no native extensions) access to Linux IO SPI interface that is similar to the SpiDev API. Based heavily on https://github.com/tomstokes/python-spi/.

- Author(s): Tom Stokes, Melissa LeBlanc-Williams

5.2.1 Implementation Notes

Software and Dependencies:

- Linux and Python 3.5 or Higher

class Adafruit_PureIO.spi.**SPI**(*device*, *max_speed_hz=None*, *bits_per_word=None*, *phase=None*, *polarity=None*, *cs_high=None*, *lsb_first=None*, *three_wire=None*, *loop=None*, *no_cs=None*, *ready=None*)

This class is similar to SpiDev, but instead of opening and closing for each call, it is set up on initialization making it fast.

bits_per_word
Number of bits per word of SPI transfer.

A value of 0 is equivalent to 8 bits per word

cs_high
SPI chip select active level

loop
SPI loopback mode

lsb_first
Bit order of SPI word transfers

max_speed_hz
Maximum SPI transfer speed in Hz.

Note that the controller cannot necessarily assign the requested speed.

mode
Mode that SPI is currently running in

no_cs
No chipselect. Single device on bus.

phase
SPI clock phase bit

polarity
SPI polarity bit

readbytes(*length*, *max_speed_hz=0*, *bits_per_word=0*, *delay=0*)
Perform half-duplex SPI read as a binary string

ready
Slave pulls low to pause

three_wire
SPI 3-wire mode

transfer(*data*, *max_speed_hz=0*, *bits_per_word=0*, *delay=0*)
Perform full-duplex SPI transfer

writebytes(*data*, *max_speed_hz=0*, *bits_per_word=0*, *delay=0*)
Perform half-duplex SPI write.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index