Adafruit Bus Device Library Documentation

Release 1.0

Scott Shawcroft and Tony Dicola

Apr 10, 2020

Contents

1 Usage Example **3**

2 Contributing **5**

3 Documentation **7**

4 Table of Contents **9**

4.1 Simple test 9

4.2 `adafruit_bus_device.i2c_device` - I2C Bus Device 10

4.3 `adafruit_bus_device.spi_device` - SPI Bus Device 11

5 Indices and tables **13**

Python Module Index **15**

Index **17**

The `I2CDevice` and `SPIDevice` helper classes make managing transaction state on a bus easy. For example, they manage locking the bus to prevent other concurrent access. For SPI devices, it manages the chip select and protocol changes such as mode. For I2C, it manages the device address.

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-circuitpython-busdevice
```

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

```
sudo pip3 install adafruit-circuitpython-busdevice
```

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-circuitpython-busdevice
```

CHAPTER 1

Usage Example

See examples/read_register_i2c.py and examples/read_register_spi.py for examples of the module's usage.

CHAPTER 2

Contributing

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

CHAPTER 3

Documentation

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

CHAPTER 4

Table of Contents

4.1 Simple test

Ensure your device works with this simple test.

Listing 1: examples/busdevice_read_register_i2c_simpletest.py

```
import busio
import board
from adafruit_bus_device.i2c_device import I2CDevice

DEVICE_ADDRESS = 0x68  # device address of DS3231 board
A_DEVICE_REGISTER = 0x0E  # device id register on the DS3231 board

# The follow is for I2C communications
comm_port = busio.I2C(board.SCL, board.SDA)
device = I2CDevice(comm_port, DEVICE_ADDRESS)

with device as bus_device:
    bus_device.write(bytes([A_DEVICE_REGISTER]))
    result = bytearray(1)
    bus_device.readinto(result)

print("".join("{:02x}".format(x) for x in result))
```

Listing 2: examples/busdevice_read_register_spi_simpletest.py

```
import busio
import board
import digitalio
from adafruit_bus_device.spi_device import SPIDevice

DEVICE_ADDRESS = 0x68  # device address of BMP280 board
A_DEVICE_REGISTER = 0xD0  # device id register on the BMP280 board
```

(continues on next page)

(continued from previous page)

```
8
9  # The follow is for SPI communications
10 cs = digitalio.DigitalInOut(board.A2)
11 comm_port = busio.SPI(board.SCK, MOSI=board.MOSI, MISO=board.MISO)
12 device = SPIDevice(comm_port, cs)
13
14 # pylint: disable-msg=no-member
15 with device as bus_device:
16     bus_device.write(bytes([A_DEVICE_REGISTER]))
17     result = bytearray(1)
18     bus_device.readinto(result)
19
20 print("".join("{:02x}".format(x) for x in result))
```

4.2 adafruit_bus_device.i2c_device - I2C Bus Device

class adafruit_bus_device.i2c_device.**I2CDevice**(*i2c*, *device_address*, *probe=True*)

Represents a single I2C device and manages locking the bus and the device address.

Parameters

- **i2c** (*I2C*) – The I2C bus the device is on
- **device_address** (*int*) – The 7 bit device address
- **probe** (*bool*) – Probe for the device upon object creation, default is true

---

Note: This class is **NOT** built into CircuitPython. See *here for install instructions*.

Example:

```
import busio
from board import *
from adafruit_bus_device.i2c_device import I2CDevice

with busio.I2C(SCL, SDA) as i2c:
    device = I2CDevice(i2c, 0x70)
    bytes_read = bytearray(4)
    with device:
        device.readinto(bytes_read)
    # A second transaction
    with device:
        device.write(bytes_read)
```

readinto(*buf*, ***, *start=0*, *end=None*)

Read into `buf` from the device. The number of bytes read will be the length of `buf`.

If `start` or `end` is provided, then the buffer will be sliced as if `buf[start:end]`. This will not cause an allocation like `buf[start:end]` will so it saves memory.

Parameters

- **buffer** (*bytearray*) – buffer to write into
- **start** (*int*) – Index to start writing at

- **end** (*int*) – Index to write up to but not include; if None, use `len(buf)`

write(*buf*, ***, *start=0*, *end=None*, *stop=True*)
Write the bytes from `buffer` to the device. Transmits a stop bit if `stop` is set.

If `start` or `end` is provided, then the buffer will be sliced as if `buffer[start:end]`. This will not cause an allocation like `buffer[start:end]` will so it saves memory.

Parameters

- **buffer** (*bytearray*) – buffer containing the bytes to write
- **start** (*int*) – Index to start writing from
- **end** (*int*) – Index to read up to but not include; if None, use `len(buf)`
- **stop** (*bool*) – If true, output an I2C stop condition after the buffer is written

write_then_readinto(*out_buffer*, *in_buffer*, ***, *out_start=0*, *out_end=None*, *in_start=0*, *in_end=None*, *stop=False*)
Write the bytes from `out_buffer` to the device, then immediately reads into `in_buffer` from the device. The number of bytes read will be the length of `in_buffer`.

If `out_start` or `out_end` is provided, then the output buffer will be sliced as if `out_buffer[out_start:out_end]`. This will not cause an allocation like `buffer[out_start:out_end]` will so it saves memory.

If `in_start` or `in_end` is provided, then the input buffer will be sliced as if `in_buffer[in_start:in_end]`. This will not cause an allocation like `in_buffer[in_start:in_end]` will so it saves memory.

Parameters

- **out_buffer** (*bytearray*) – buffer containing the bytes to write
- **in_buffer** (*bytearray*) – buffer containing the bytes to read into
- **out_start** (*int*) – Index to start writing from
- **out_end** (*int*) – Index to read up to but not include; if None, use `len(out_buffer)`
- **in_start** (*int*) – Index to start writing at
- **in_end** (*int*) – Index to write up to but not include; if None, use `len(in_buffer)`
- **stop** (*bool*) – Deprecated

4.3 adafruit_bus_device.spi_device - SPI Bus Device

class adafruit_bus_device.spi_device.**SPIDevice**(*spi*, *chip_select=None*, ***, *baudrate=100000*, *polarity=0*, *phase=0*, *extra_clocks=0*)
Represents a single SPI device and manages locking the bus and the device address.

Parameters

- **spi** (*SPI*) – The SPI bus the device is on
- **chip_select** (*DigitalInOut*) – The chip select pin object that implements the DigitalInOut API.
- **extra_clocks** (*int*) – The minimum number of clock cycles to cycle the bus after CS is high. (Used for SD cards.)

Note: This class is **NOT** built into CircuitPython. See *here for install instructions*.

Example:

```
import busio
import digitalio
from board import *
from adafruit_bus_device.spi_device import SPIDevice

with busio.SPI(SCK, MOSI, MISO) as spi_bus:
    cs = digitalio.DigitalInOut(D10)
    device = SPIDevice(spi_bus, cs)
    bytes_read = bytearray(4)
    # The object assigned to spi in the with statements below
    # is the original spi_bus object. We are using the busio.SPI
    # operations busio.SPI.readinto() and busio.SPI.write().
    with device as spi:
        spi.readinto(bytes_read)
    # A second transaction
    with device as spi:
        spi.write(bytes_read)
```

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

adafruit_bus_device.i2c_device, 10
adafruit_bus_device.spi_device, 11

Index

A

adafruit_bus_device.i2c_device (*module*), 10
adafruit_bus_device.spi_device (*module*), 11

I

I2CDevice (*class in adafruit_bus_device.i2c_device*), 10

R

readinto() (*adafruit_bus_device.i2c_device.I2CDevice method*), 10

S

SPIDevice (*class in adafruit_bus_device.spi_device*), 11

W

write() (*adafruit_bus_device.i2c_device.I2CDevice method*), 11
write_then_readinto() (*adafruit_bus_device.i2c_device.I2CDevice method*), 11