Adafruit Bus Device Library Documentation

Release 1.0

Scott Shawcroft and Tony Dicola

Feb 08, 2019

Contents

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.

CHAPTER 1

Installation

This library is **NOT** built into CircuitPython to make it easy to update. To install it either follow the directions below or install the library bundle.

To install:

1. Download and unzip the latest release zip.
2. Copy the unzipped `adafruit_bus_device` to the `lib` directory on the `CIRCUITPY` drive.

CHAPTER 2

Usage Example

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

CHAPTER 3

Contributing

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

CHAPTER 4

Building locally

To build this library locally you'll need to install the circuitpython-build-tools package.

```
python3 -m venv .env
source .env/bin/activate
pip install circuitpython-build-tools
```

Once installed, make sure you are in the virtual environment:

```
source .env/bin/activate
```

Then run the build:

```
circuitpython-build-bundles --filename_prefix adafruit-circuitpython-busdevice --
↪library_location .
```

4.1 Sphinx documentation

Sphinx is used to build the documentation based on rST files and comments in the code. First, install dependencies (feel free to reuse the virtual environment from above):

```
python3 -m venv .env
source .env/bin/activate
pip install Sphinx sphinx-rtd-theme
```

Now, once you have the virtual environment activated:

```
cd docs
sphinx-build -E -W -b html . _build/html
```

This will output the documentation to `docs/_build/html`. Open the index.html in your browser to view them. It will also (due to -W) error out on any warning like Travis will. This is a good way to locally verify it will pass.

CHAPTER 5

Table of Contents

5.1 Simple test

Ensure your device works with this simple test.

Listing 1: examples/read_register_i2c.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/read_register_spi.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))
```

5.2 adafruit_bus_device.i2c_device - I2C Bus Device

class adafruit_bus_device.i2c_device.**I2CDevice**(*i2c*, *device_address*, *, *debug=False*)

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

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*, ***kwargs*)

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

write(*buf*, ***kwargs*)
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
- **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=True*)
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`. Transmits a stop bit after the write, if `stop` is set.

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
- **in_start** (*int*) – Index to start writing at
- **in_end** (*int*) – Index to write up to but not include
- **stop** (*bool*) – If true, output an I2C stop condition after the buffer is written

5.3 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 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index