Adafruit SimpleIO Library Documentation

Release 1.0

Scott Shawcroft

Mar 08, 2018

Contents

1 Dependencies **3**

2 Usage Example **5**

3 Contributing **7**

4 API Reference **9**

4.1 `simpleio` - Simple, beginner friendly IO. 9

Python Module Index **13**

SimpleIO features a number of helpers to simplify hardware interactions. Many of the functions and classes are inspired by Arduino APIs to make it easier to move to CircuitPython from Arduino.

CHAPTER 1

Dependencies

This driver depends on:

- Adafruit CircuitPython

Please ensure all dependencies are available on the CircuitPython filesystem. This is easily achieved by downloading the Adafruit library and driver bundle.

CHAPTER 2

Usage Example

TODO

CHAPTER 3

Contributing

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

CHAPTER 4

API Reference

4.1 simpleio - Simple, beginner friendly IO.

The *simpleio* module contains classes to provide simple access to IO.

class simpleio.**DigitalIn**(*pin*)
Simple digital input that is valid until soft reset.

value
The digital logic level of the input pin.

class simpleio.**DigitalOut**(*pin*)
Simple digital output that is valid until soft reset.

value
The digital logic level of the output pin.

class simpleio.**Servo**(*pin*, *min_pulse=0.5*, *max_pulse=2.5*)
Easy control for hobby (3-wire) servos

Parameters

- **pin** (*Pin*) – PWM pin where the servo is located.
- **min_pulse** (*int*) – Pulse width (microseconds) corresponding to 0 degrees.
- **max_pulse** (*int*) – Pulse width (microseconds) corresponding to 180 degrees.

Example for Metro M0 Express:

```
import simpleio
import time
from board import *

pwm = simpleio.Servo(D9)

while True:
    pwm.angle = 0
```

```
    print("Angle: ", pwm.angle)
    time.sleep(2)
    pwm.angle = pwm.microseconds_to_angle(2500)
    print("Angle: ", pwm.angle)
    time.sleep(2)
```

angle
Get and set the servo angle in degrees

deinit()
Detaches servo object from pin, frees pin

microseconds_to_angle(*us*)
Converts microseconds to a degree value

simpleio.**bitWrite**(*x*, *n*, *b*)
Based on the Arduino bitWrite function, changes a specific bit of a value to 0 or 1. The return value is the original value with the changed bit. This function is written for use with 8-bit shift registers

Parameters

- **x** – numeric value
- **n** – position to change starting with least-significant (right-most) bit as 0
- **b** – value to write (0 or 1)

simpleio.**map_range**(*x*, *in_min*, *in_max*, *out_min*, *out_max*)
Maps a number from one range to another. Note: This implementation handles values < in_min differently than arduino's map function does.

Returns Returns value mapped to new range

Return type float

simpleio.**shift_in**(*data_pin*, *clock*, *msb_first=True*)
Shifts in a byte of data one bit at a time. Starts from either the LSB or MSB.

> **Warning:** Data and clock are swapped compared to other CircuitPython libraries in order to match Arduino.

Parameters

- **data_pin** (*DigitalInOut*) – pin on which to input each bit
- **clock** (*DigitalInOut*) – toggles to signal data_pin reads
- **msb_first** (*bool*) – True when the first bit is most significant

Returns returns the value read

Return type int

simpleio.**shift_out**(*data_pin*, *clock*, *value*, *msb_first=True*)
Shifts out a byte of data one bit at a time. Data gets written to a data pin. Then, the clock pulses hi then low

> **Warning:** Data and clock are swapped compared to other CircuitPython libraries in order to match Arduino.

Parameters

- **data_pin** (*DigitalInOut*) – value bits get output on this pin
- **clock** (*DigitalInOut*) – toggled once the data pin is set
- **msb_first** (*bool*) – True when the first bit is most significant
- **value** (*int*) – byte to be shifted

Example for Metro M0 Express:

```
import digitalio
import simpleio
from board import *
clock = digitalio.DigitalInOut(D12)
data_pin = digitalio.DigitalInOut(D11)
latchPin = digitalio.DigitalInOut(D10)
clock.direction = digitalio.Direction.OUTPUT
data_pin.direction = digitalio.Direction.OUTPUT
latchPin.direction = digitalio.Direction.OUTPUT

while True:
    valueSend = 500
    # shifting out least significant bits
    # must toggle latchPin.value before and after shift_out to push to IC chip
    # this sample code was tested using
    latchPin.value = False
    simpleio.shift_out(data_pin, clock, (valueSend>>8), msb_first = False)
    latchPin.value = True
    time.sleep(1.0)
    latchPin.value = False
    simpleio.shift_out(data_pin, clock, valueSend, msb_first = False)
    latchPin.value = True
    time.sleep(1.0)

    # shifting out most significant bits
    latchPin.value = False
    simpleio.shift_out(data_pin, clock, (valueSend>>8))
    latchPin.value = True
    time.sleep(1.0)
    latchpin.value = False
    simpleio.shift_out(data_pin, clock, valueSend)
    latchpin.value = True
    time.sleep(1.0)
```

simpleio.**tone**(*pin*, *frequency*, *duration=1*, *length=100*)

Generates a square wave of the specified frequency on a pin

Parameters

- **Pin** (*Pin*) – Pin on which to output the tone
- **frequency** (*float*) – Frequency of tone in Hz
- **length** (*int*) – Variable size buffer (optional)
- **duration** (*int*) – Duration of tone in seconds (optional)

Python Module Index

S

simpleio, 9

Index

A

angle (simpleio.Servo attribute), 10

B

bitWrite() (in module simpleio), 10

D

deinit() (simpleio.Servo method), 10
DigitalIn (class in simpleio), 9
DigitalOut (class in simpleio), 9

M

map_range() (in module simpleio), 10
microseconds_to_angle() (simpleio.Servo method), 10

S

Servo (class in simpleio), 9
shift_in() (in module simpleio), 10
shift_out() (in module simpleio), 10
simpleio (module), 9

T

tone() (in module simpleio), 11

V

value (simpleio.DigitalIn attribute), 9
value (simpleio.DigitalOut attribute), 9