python-periphery Documentation

Release 1.1.0

Vanya Sergeev

Mar 12, 2017

Contents

python-periphery is a pure Python library for GPIO, LED, PWM, SPI, I2C, MMIO, and Serial peripheral I/O interface access in userspace Linux. It is useful in embedded Linux environments (including Raspberry Pi, BeagleBone, etc. platforms) for interfacing with external peripherals. python-periphery is compatible with Python 2 and Python 3, is written in pure Python, and is MIT licensed.

CHAPTER 1

Contents

GPIO

Code Example

```
from periphery import GPIO

# Open GPIO 10 with input direction
gpio_in = GPIO(10, "in")
# Open GPIO 12 with output direction
gpio_out = GPIO(12, "out")

value = gpio_in.read()
gpio_out.write(value)

gpio_in.close()
gpio_out.close()
```

API

class periphery.**GPIO**(*pin*, *direction='preserve'*)

Bases: object

Instantiate a GPIO object and open the sysfs GPIO corresponding to the specified pin, with the specified direction.

direction can be "in" for input; "out" for output, initialized to low; "high" for output, initialized to high; "low" for output, initialized to low, or "preserve" for preserving existing direction. Default is "preserve".

Parameters

- **pin** (*int*) – Linux pin number.
- **direction** (*str*) – pin direction, can be "in", "out", "high", "low", or "preserve".

Returns GPIO object.

Return type *GPIO*

Raises

- `GPIOError` – if an I/O or OS error occurs.
- `TypeError` – if *pin* or *direction* types are invalid.
- `ValueError` – if *direction* value is invalid.

read()
Read the state of the GPIO.

Returns `True` for high state, `False` for low state.

Return type bool

Raises `GPIOError` – if an I/O or OS error occurs.

write(*value*)
Set the state of the GPIO to *value*.

Parameters **value** (*bool*) – `True` for high state, `False` for low state.

Raises

- `GPIOError` – if an I/O or OS error occurs.
- `TypeError` – if *value* type is not bool.

poll(*timeout=None*)
Poll a GPIO for the edge event configured with the .edge property.

timeout can be a positive number for a timeout in seconds, 0 for a non-blocking poll, or negative or None for a blocking poll. Defaults to blocking poll.

Parameters **timeout** (*int, float, None*) – timeout duration in seconds.

Returns `True` if an edge event occurred, `False` on timeout.

Return type bool

Raises

- `GPIOError` – if an I/O or OS error occurs.
- `TypeError` – if *timeout* type is not None or int.

close()
Close the sysfs GPIO.

Raises `GPIOError` – if an I/O or OS error occurs.

fd
Get the file descriptor for the underlying sysfs GPIO "value" file of the GPIO object.

Type int

pin
Get the sysfs GPIO pin number.

Type int

supports_interrupts
Get whether or not this GPIO supports edge interrupts, configurable with the .edge property.

Type bool

direction
Get or set the GPIO's direction. Can be "in", "out", "high", "low".

Direction "in" is input; "out" is output, initialized to low; "high" is output, initialized to high; and "low" is output, initialized to low.

Raises

- `GPIOError` – if an I/O or OS error occurs.
- `TypeError` – if *direction* type is not str.
- `ValueError` – if *direction* value is invalid.

Type str

edge
Get or set the GPIO's interrupt edge. Can be "none", "rising", "falling", "both".

Raises

- `GPIOError` – if an I/O or OS error occurs.
- `TypeError` – if *edge* type is not str.
- `ValueError` – if *edge* value is invalid.

Type str

class `periphery.`**GPIOError**
Bases: `exceptions.IOError`

Base class for GPIO errors.

LED

Code Example

```
from periphery import LED

# Open LED "led0" with initial state off
led0 = LED("led0", False)
# Open LED "led1" with initial state on
led1 = LED("led1", True)

value = led0.read()
led1.write(value)

# Set custom brightness level
led1.write(led1.max_brightness / 2)

led0.close()
led1.close()
```

API

class `periphery.`**LED**(*name*, *brightness=None*)
Bases: `object`

Instantiate an LED object and open the sysfs LED corresponding to the specified name.

brightness can be a boolean for on/off, integer value for a specific brightness, or None to preserve existing brightness. Default is preserve existing brightness.

Parameters

- **name** (*str*) – Linux led name.
- **brightness** (*bool, int, None*) – Initial brightness.

Returns LED object.

Return type *LED*

Raises

- LEDError – if an I/O or OS error occurs.
- TypeError – if *name* or *brightness* types are invalid.
- ValueError – if *brightness* value is invalid.

read()
Read the brightness of the LED.

Returns Current brightness.

Return type int

Raises LEDError – if an I/O or OS error occurs.

write(*brightness*)
Set the brightness of the LED to *brightness*.

brightness can be a boolean for on/off, or integer value for a specific brightness.

Parameters **brightness** (*bool, int*) – Brightness value to set.

Raises

- LEDError – if an I/O or OS error occurs.
- TypeError – if *brightness* type is not bool or int.

close()
Close the sysfs LED.

Raises LEDError – if an I/O or OS error occurs.

fd
Get the file descriptor for the underlying sysfs LED "brightness" file of the LED object.

Type int

name
Get the sysfs LED name.

Type str

max_brightness
Get the LED's max brightness.

Type int

brightness
Get or set the LED's brightness.

Value can be a boolean for on/off, or integer value a for specific brightness.

Raises

- `LEDError` – if an I/O or OS error occurs.
- `TypeError` – if *brightness* type is not bool or int.
- `ValueError` – if *brightness* value is invalid.

Type int

class `periphery.`**`LEDError`**

Bases: `exceptions.IOError`

Base class for LED errors.

PWM

Code Example

```
from periphery import PWM

# Open PWM channel 0, pin 10
pwm = PWM(0, 10)

# Set frequency to 1 kHz
pwm.frequency = 1e3
# Set duty cycle to 75%
pwm.duty_cycle = 0.75

pwm.enable()

# Change duty cycle to 50%
pwm.duty_cycle = 0.50

pwm.close()
```

API

class `periphery.`**`PWM`**(*channel*, *pin*)

Bases: `object`

Instantiate a PWM object and open the sysfs PWM corresponding to the specified channel and pin.

Parameters

- **`channel`** (*int*) – Linux channel number.
- **`pin`** (*int*) – Linux pin number.

Returns PWM object.

Return type *PWM*

Raises

- `PWMError` – if an I/O or OS error occurs.
- `TypeError` – if *channel* or *pin* types are invalid.
- `ValueError` – if PWM channel does not exist.

close()
Close the sysfs PWM.

enable()
Enable the PWM output.

disable()
Disable the PWM output.

channel
Get the sysfs PWM channel number.

Type int

pin
Get the sysfs PWM pin number.

Type int

period
Get or set the PWM's output period in seconds.

Raises

- `PWMError` – if an I/O or OS error occurs.
- `TypeError` – if value type is not int or float.

Type int, float

duty_cycle
Get or set the PWM's output duty cycle as a ratio from 0.0 to 1.0.

Raises

- `PWMError` – if an I/O or OS error occurs.
- `TypeError` – if value type is not int or float.
- `ValueError` – if value is out of bounds of 0.0 to 1.0.

Type int, float

frequency
Get or set the PWM's output frequency in Hertz.

Raises

- `PWMError` – if an I/O or OS error occurs.
- `TypeError` – if value type is not int or float.

Type int, float

polarity
Get or set the PWM's output polarity. Can be "normal" or "inversed".

Raises

- `PWMError` – if an I/O or OS error occurs.
- `TypeError` – if value type is not str.
- `ValueError` – if value is invalid.

Type str

enabled
Get or set the PWM's output enabled state.

Raises

- `PWMError` – if an I/O or OS error occurs.
- `TypeError` – if value type is not bool.

Type bool

class `periphery.`**PWMError**
Bases: `exceptions.IOError`

Base class for PWM errors.

SPI

Code Example

```
from periphery import SPI

# Open spidev1.0 with mode 0 and max speed 1MHz
spi = SPI("/dev/spidev1.0", 0, 1000000)

data_out = [0xaa, 0xbb, 0xcc, 0xdd]
data_in = spi.transfer(data_out)

print("shifted out [0x%02x, 0x%02x, 0x%02x, 0x%02x]" % tuple(data_out))
print("shifted in  [0x%02x, 0x%02x, 0x%02x, 0x%02x]" % tuple(data_in))

spi.close()
```

API

class `periphery.`**SPI**(*devpath*, *mode*, *max_speed*, *bit_order='msb'*, *bits_per_word=8*, *extra_flags=0*)
Bases: `object`

Instantiate a SPI object and open the spidev device at the specified path with the specified SPI mode, max speed in hertz, and the defaults of "msb" bit order and 8 bits per word.

Parameters

- **devpath** (*str*) – spidev device path.
- **mode** (*int*) – SPI mode, can be 0, 1, 2, 3.
- **max_speed** (*int, float*) – maximum speed in Hertz.
- **bit_order** (*str*) – bit order, can be "msb" or "lsb".
- **bits_per_word** (*int*) – bits per word.
- **extra_flags** (*int*) – extra spidev flags to be bitwise-ORed with the SPI mode.

Returns SPI object.

Return type *SPI*

Raises

- `SPIError` – if an I/O or OS error occurs.
- `TypeError` – if *devpath*, *mode*, *max_speed*, *bit_order*, *bits_per_word*, or *extra_flags* types are invalid.
- `ValueError` – if *mode*, *bit_order*, *bits_per_word*, or *extra_flags* values are invalid.

transfer(*data*)
Shift out *data* and return shifted in data.

Parameters **data** (*bytes, bytearray, list*) – a byte array or list of 8-bit integers to shift out.

Returns data shifted in.

Return type bytes, bytearray, list

Raises

- `SPIError` – if an I/O or OS error occurs.
- `TypeError` – if *data* type is invalid.
- `ValueError` – if data is not valid bytes.

close()
Close the spidev SPI device.

Raises `SPIError` – if an I/O or OS error occurs.

fd
Get the file descriptor of the underlying spidev device.

Type int

devpath
Get the device path of the underlying spidev device.

Type str

mode
Get or set the SPI mode. Can be 0, 1, 2, 3.

Raises

- `SPIError` – if an I/O or OS error occurs.
- `TypeError` – if *mode* type is not int.
- `ValueError` – if *mode* value is invalid.

Type int

max_speed
Get or set the maximum speed in Hertz.

Raises

- `SPIError` – if an I/O or OS error occurs.
- `TypeError` – if *max_speed* type is not int or float.

Type int, float

bit_order
Get or set the SPI bit order. Can be "msb" or "lsb".

Raises

- `SPIError` – if an I/O or OS error occurs.
- `TypeError` – if *bit_order* type is not str.
- `ValueError` – if *bit_order* value is invalid.

Type str

bits_per_word
Get or set the SPI bits per word.

Raises

- `SPIError` – if an I/O or OS error occurs.
- `TypeError` – if *bits_per_word* type is not int.
- `ValueError` – if *bits_per_word* value is invalid.

Type int

extra_flags
Get or set the spidev extra flags. Extra flags are bitwise-ORed with the SPI mode.

Raises

- `SPIError` – if an I/O or OS error occurs.
- `TypeError` – if *extra_flags* type is not int.
- `ValueError` – if *extra_flags* value is invalid.

Type int

class `periphery.`**SPIError**
Bases: `exceptions.IOError`

Base class for SPI errors.

I2C

Code Example

```
from periphery import I2C

# Open i2c-0 controller
i2c = I2C("/dev/i2c-0")

# Read byte at address 0x100 of EEPROM at 0x50
msgs = [I2C.Message([0x01, 0x00]), I2C.Message([0x00], read=True)]
i2c.transfer(0x50, msgs)
print("0x100: 0x%02x" % msgs[1].data[0])

i2c.close()
```

API

class `periphery.`**I2C**(*devpath*)
Bases: `object`

Instantiate an I2C object and open the i2c-dev device at the specified path.

Parameters **devpath** (*str*) – i2c-dev device path.

Returns I2C object.

Return type *I2C*

Raises I2CError – if an I/O or OS error occurs.

transfer(*address*, *messages*)
Transfer *messages* to the specified I2C *address*. Modifies the *messages* array with the results of any read transactions.

Parameters

- **address** (*int*) – I2C address.
- **messages** (*list*) – list of I2C.Message messages.

Raises

- I2CError – if an I/O or OS error occurs.
- TypeError – if *messages* type is not list.
- ValueError – if *messages* length is zero, or if message data is not valid bytes.

close()
Close the i2c-dev I2C device.

Raises I2CError – if an I/O or OS error occurs.

fd
Get the file descriptor of the underlying i2c-dev device.

Type int

devpath
Get the device path of the underlying i2c-dev device.

Type str

class Message(*data*, *read=False*, *flags=0*)
Instantiate an I2C Message object.

Parameters

- **data** (*bytes, bytearray, list*) – a byte array or list of 8-bit integers to write.
- **read** (*bool*) – specify this as a read message, where *data* serves as placeholder bytes for the read.
- **flags** (*int*) – additional i2c-dev flags for this message.

Returns Message object.

Return type *Message*

Raises TypeError – if *data*, *read*, or *flags* types are invalid.

class periphery.**I2CError**
Bases: exceptions.IOError

Base class for I2C errors.

MMIO

Code Example

```
from periphery import MMIO

# Open am335x real-time clock subsystem page
rtc_mmio = MMIO(0x44E3E000, 0x1000)

# Read current time
rtc_secs = rtc_mmio.read32(0x00)
rtc_mins = rtc_mmio.read32(0x04)
rtc_hrs = rtc_mmio.read32(0x08)

print("hours: %02x minutes: %02x seconds: %02x" % (rtc_hrs, rtc_mins, rtc_secs))

rtc_mmio.close()

# Open am335x control module page
ctrl_mmio = MMIO(0x44E10000, 0x1000)

# Read MAC address
mac_id0_lo = ctrl_mmio.read32(0x630)
mac_id0_hi = ctrl_mmio.read32(0x634)

print("MAC address: %04x%08x" % (mac_id0_lo, mac_id0_hi))

ctrl_mmio.close()
```

API

class periphery.**MMIO**(*physaddr*, *size*)

Bases: object

Instantiate an MMIO object and map the region of physical memory specified by the address base *physaddr* and size *size* in bytes.

Parameters

- **physaddr** (*int, long*) – base physical address of memory region.
- **size** (*int, long*) – size of memory region.

Returns MMIO object.

Return type *MMIO*

Raises

- MMIOError – if an I/O or OS error occurs.
- TypeError – if *physaddr* or *size* types are invalid.

read32(*offset*)

Read 32-bits from the specified *offset* in bytes, relative to the base physical address of the MMIO region.

Parameters **offset** (*int, long*) – offset from base physical address, in bytes.

Returns 32-bit value read.

Return type int

Raises

- `TypeError` – if *offset* type is invalid.
- `ValueError` – if *offset* is out of bounds.

read16(*offset*)
Read 16-bits from the specified *offset* in bytes, relative to the base physical address of the MMIO region.

Parameters **offset** (*int, long*) – offset from base physical address, in bytes.

Returns 16-bit value read.

Return type int

Raises

- `TypeError` – if *offset* type is invalid.
- `ValueError` – if *offset* is out of bounds.

read8(*offset*)
Read 8-bits from the specified *offset* in bytes, relative to the base physical address of the MMIO region.

Parameters **offset** (*int, long*) – offset from base physical address, in bytes.

Returns 8-bit value read.

Return type int

Raises

- `TypeError` – if *offset* type is invalid.
- `ValueError` – if *offset* is out of bounds.

read(*offset*, *length*)
Read a string of bytes from the specified *offset* in bytes, relative to the base physical address of the MMIO region.

Parameters

- **offset** (*int, long*) – offset from base physical address, in bytes.
- **length** (*int*) – number of bytes to read.

Returns bytes read.

Return type bytes

Raises

- `TypeError` – if *offset* type is invalid.
- `ValueError` – if *offset* is out of bounds.

write32(*offset*, *value*)
Write 32-bits to the specified *offset* in bytes, relative to the base physical address of the MMIO region.

Parameters

- **offset** (*int, long*) – offset from base physical address, in bytes.
- **value** (*int, long*) – 32-bit value to write.

Raises

- `TypeError` – if *offset* or *value* type are invalid.

- `ValueError` – if *offset* or *value* are out of bounds.

write16(*offset*, *value*)
Write 16-bits to the specified *offset* in bytes, relative to the base physical address of the MMIO region.

Parameters

- **offset** (*int, long*) – offset from base physical address, in bytes.
- **value** (*int, long*) – 16-bit value to write.

Raises

- `TypeError` – if *offset* or *value* type are invalid.
- `ValueError` – if *offset* or *value* are out of bounds.

write8(*offset*, *value*)
Write 8-bits to the specified *offset* in bytes, relative to the base physical address of the MMIO region.

Parameters

- **offset** (*int, long*) – offset from base physical address, in bytes.
- **value** (*int, long*) – 8-bit value to write.

Raises

- `TypeError` – if *offset* or *value* type are invalid.
- `ValueError` – if *offset* or *value* are out of bounds.

write(*offset*, *data*)
Write a string of bytes to the specified *offset* in bytes, relative to the base physical address of the MMIO region.

Parameters

- **offset** (*int, long*) – offset from base physical address, in bytes.
- **data** (*bytes, bytearray, list*) – a byte array or list of 8-bit integers to write.

Raises

- `TypeError` – if *offset* or *data* type are invalid.
- `ValueError` – if *offset* is out of bounds, or if data is not valid bytes.

close()
Unmap the MMIO object's mapped physical memory.

base
Get the base physical address of the MMIO region.

Type int

size
Get the mapping size of the MMIO region.

Type int

pointer
Get a ctypes void pointer to the memory mapped region.

Type ctypes.c_void_p

class periphery.**MMIOError**
Bases: exceptions.IOError

Base class for MMIO errors.

Serial

Code Example

```
from periphery import Serial

# Open /dev/ttyUSB0 with baudrate 115200, and defaults of 8N1, no flow control
serial = Serial("/dev/ttyUSB0", 115200)

serial.write(b"Hello World!")

# Read up to 128 bytes with 500ms timeout
buf = serial.read(128, 0.5)
print("read %d bytes: _%s_" % (len(buf), buf))

serial.close()
```

API

class periphery.**Serial**(*devpath*, *baudrate*, *databits=8*, *parity='none'*, *stopbits=1*, *xonxoff=False*, *rtscts=False*)
Bases: object

Instantiate a Serial object and open the tty device at the specified path with the specified baudrate, and the defaults of 8 data bits, no parity, 1 stop bit, no software flow control (xonxoff), and no hardware flow control (rtscts).

Parameters

- **devpath** (*str*) – tty device path.
- **baudrate** (*int*) – baudrate.
- **databits** (*int*) – data bits, can be 5, 6, 7, 8.
- **parity** (*str*) – parity, can be "none", "even", "odd".
- **stopbits** (*int*) – stop bits, can be 1 or 2.
- **xonxoff** (*bool*) – software flow control.
- **rtscts** (*bool*) – hardware flow control.

Returns Serial object.

Return type *Serial*

Raises

- SerialError – if an I/O or OS error occurs.
- TypeError – if *devpath*, *baudrate*, *databits*, *parity*, *stopbits*, *xonxoff*, or *rtscts* types are invalid.
- ValueError – if *baudrate*, *databits*, *parity*, or *stopbits* values are invalid.

read(*length*, *timeout=None*)
Read up to *length* number of bytes from the serial port with an optional timeout.

timeout can be positive for a timeout in seconds, 0 for a non-blocking read, or negative or None for a blocking read that will block until *length* number of bytes are read. Default is a blocking read.

For a non-blocking or timeout-bound read, read() may return data whose length is less than or equal to the requested length.

Parameters

- **length** (*int*) – length in bytes.
- **timeout** (*int, float, None*) – timeout duration in seconds.

Returns data read.

Return type bytes

Raises `SerialError` – if an I/O or OS error occurs.

write(*data*)
Write *data* to the serial port and return the number of bytes written.

Parameters **data** (*bytes, bytearray, list*) – a byte array or list of 8-bit integers to write.

Returns number of bytes written.

Return type int

Raises

- `SerialError` – if an I/O or OS error occurs.
- `TypeError` – if *data* type is invalid.
- `ValueError` – if data is not valid bytes.

poll(*timeout=None*)
Poll for data available for reading from the serial port.

timeout can be positive for a timeout in seconds, 0 for a non-blocking poll, or negative or None for a blocking poll. Default is a blocking poll.

Parameters **timeout** (*int, float, None*) – timeout duration in seconds.

Returns `True` if data is available for reading from the serial port, `False` if not.

Return type bool

flush()
Flush the write buffer of the serial port, blocking until all bytes are written.

Raises `SerialError` – if an I/O or OS error occurs.

input_waiting()
Query the number of bytes waiting to be read from the serial port.

Returns number of bytes waiting to be read.

Return type int

Raises `SerialError` – if an I/O or OS error occurs.

output_waiting()
Query the number of bytes waiting to be written to the serial port.

Returns number of bytes waiting to be written.

Return type int

Raises `SerialError` – if an I/O or OS error occurs.

close()
Close the tty device.

Raises `SerialError` – if an I/O or OS error occurs.

fd
Get the file descriptor of the underlying tty device.

Type int

devpath
Get the device path of the underlying tty device.

Type str

baudrate
Get or set the baudrate.

Raises

- `SerialError` – if an I/O or OS error occurs.
- `TypeError` – if *baudrate* type is not int.
- `ValueError` – if *baudrate* value is not supported.

Type int

databits
Get or set the data bits. Can be 5, 6, 7, 8.

Raises

- `SerialError` – if an I/O or OS error occurs.
- `TypeError` – if *databits* type is not int.
- `ValueError` – if *databits* value is invalid.

Type int

parity
Get or set the parity. Can be “none”, “even”, “odd”.

Raises

- `SerialError` – if an I/O or OS error occurs.
- `TypeError` – if *parity* type is not str.
- `ValueError` – if *parity* value is invalid.

Type str

stopbits
Get or set the stop bits. Can be 1 or 2.

Raises

- `SerialError` – if an I/O or OS error occurs.
- `TypeError` – if *stopbits* type is not int.

- `ValueError` – if *stopbits* value is invalid.

Type int

xonxoff
Get or set software flow control.

Raises

- `SerialError` – if an I/O or OS error occurs.
- `TypeError` – if *xonxoff* type is not bool.

Type bool

rtscts
Get or set hardware flow control.

Raises

- `SerialError` – if an I/O or OS error occurs.
- `TypeError` – if *rtscts* type is not bool.

Type bool

class `periphery.`**SerialError**
Bases: `exceptions.IOError`

Base class for Serial errors.

Version and Helper Functions

`periphery.`**__version__ = '1.1.0'**
Module version string.

`periphery.`**version = (1, 1, 0)**
Module version tuple.

`periphery.`**sleep**(*seconds*)
Sleep for the specified number of seconds.

Parameters **seconds** (*int, long, float*) – duration in seconds.

`periphery.`**sleep_ms**(*milliseconds*)
Sleep for the specified number of milliseconds.

Parameters **milliseconds** (*int, long, float*) – duration in milliseconds.

`periphery.`**sleep_us**(*microseconds*)
Sleep for the specified number of microseconds.

Parameters **microseconds** (*int, long, float*) – duration in microseconds.

Python Module Index

p

Index

Symbols

B

C

D

E

F

G

I

L

M

N

O

P

R

S

T

V

W

X