Adafruit MicroPython TSL2561 Documentation

Release 1.0

Radomir Dopieralski

Jul 10, 2017

Contents

1 *tsl2561* module **3**
- 1.1 TSL2561 . . . 3
- 1.2 Classes for Different Packages . . . 4

2 Usage Examples **5**

3 Indices and tables **7**

Python Module Index **9**

Contents:

CHAPTER 1

tsl2561 module

TSL2561

class `tsl2561.`**`TSL2561`**(*i2c*[, *address*])
The basic class for handling the communication with the sensor.

The `i2c` parameter is an initialized I^2C bus, and the optional address specifies which sensor to connect to, if you have more than one and have changed their addresses with the `Addr` pin.

May raise a `RuntimeError` if connected to a wrong model of sensor.

`active`([*True*|*False*])
Get or set the power status of the sensor.

If called without any parameters, returns the current status (`True` for active, `False` for inactive). If called with a parameter, sets the status.

The functions that require the sensor to be active will activate it for the time necessary, and then return it to the previous state.

`gain`([*1*|*16*])
Get or set the measurement gain of the sensor.

`integration_time`([*0*|*13*|*101*|*402*])
Get or set the integraton time for measurements in milliseconds.

When set to 0, manual integration time is enabled.

Note that with manual integration time, it's not possible to compute the lux values or use autogain.

`sensor_id`()
Return the internal ID of the sensor, describing its model and revision number.

`read`(*[autogain][, raw]*)
Read a measurement from the sensor.

If `raw` is `False` (the default), returns a single floating point value denoting luminosity in lux. If it's `True`, returns a pair of integer numbers, corresponding to the readings of the broadband and infra-red channels, respectively.

If `autogain` is set to `True` (deafult `False`), the sensor may repeat the measurement with a different gain setting if the result is out of bounds for it.

May raise a `ValueError` if the sensor is saturated with `raw=False`.

If at the moment of calling this method, the sensor is inactive, it will be activated, the method will block for a time necessary for the sensor to make the measurement, and then return and deactivate the sensor. If the sensor was active, no waiting takes place – it's up to the user to make sure the sensor had enough time to make the measurement.

threshold(*[cycles] [, min_value][, max_value]*)

Get or set the interrupt settings.

If called without arguments, returns a triple of values.

The `cycles` argument specifies how many cycles are needed to trigger the interrupt. When set to 0, each measurement will do it. If set to 1-15, the interrupt will be triggered only when so many measurements in a row fall out of bounds. When set to -1, interrupts are disabled.

The `min_value` and `max_value` specify the window for the broadband channel to fit in.

Whenever the interrupt is activated, the `Int` pin of the sensor will be pulled low and stay in that state until cleared. You can connect it to one of the pins and set a pin interrupt to detect that event.

Note that for the interrupt pin to work, the sensor must stay active.

interrupt(*False*)

Clears the interrupt state, bringing the `Int` pin back to neutral.

Classes for Different Packages

There are also `TSL2561T`, `TSL2561FN`, `TSL2561CL` and `TSL2561CS` classes for versions of the sensor using different packages. They are the same, except for `TSL2561CS`, which uses different scale for lux computations.

CHAPTER 2

Usage Examples

Connect your sensor in following way:

- `3vo 3V`
- `gnd gnd`
- `sda gpio4`
- `scl gpio5`
- `int` any other pin (optional)

Now, to make basic measurement:

```
import tsl2561
from machine import I2C, Pin
i2c = I2C(Pin(5), Pin(4))
sensor = tsl2561.TSL2561(i2c)
print(sensor.read())
```

To perform continuous measurement:

```
import time
sensor.active(True)
time.sleep_ms(500)
while True:
    print(sensor.read())
    time.sleep_ms(20)
```

To change the gain and integration time:

```
sensor.gain(16)
sensor.integration_time(402)
```

To use the interrupt pin (connected to `gpio0` here):

```
def handler(pin):
    print("interrupt!")
    sensor.interrupt(False)

int_pin = Pin(0, Pin.IN, Pin.PULL_UP)
int_pin.irq(handler=handler, trigger=Pin.IRQ_FALLING)
sensor.active(True)
sensor.threshold(1, 10000, 30000)
```

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

t

tsl2561, 3

Index

A

active() (tsl2561.TSL2561 method), 3

G

gain() (tsl2561.TSL2561 method), 3

I

integration_time() (tsl2561.TSL2561 method), 3
interrupt() (tsl2561.TSL2561 method), 4

R

read() (tsl2561.TSL2561 method), 3

S

sensor_id() (tsl2561.TSL2561 method), 3

T

threshold() (tsl2561.TSL2561 method), 4
TSL2561 (class in tsl2561), 3
tsl2561 (module), 3