AdafruitTMP117 Library Documentation

Release 1.0

Bryan Siepert

May 24, 2021

CONTENTS

CircuitPython library for the TI TMP117 Temperature sensor

CHAPTER

ONE

DEPENDENCIES

This driver depends on:

- Adafruit CircuitPython
- Bus Device
- Register

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

CHAPTER

TWO

INSTALLING FROM PYPI

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-tmp117
```

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

```
sudo pip3 install adafruit-circuitpython-tmp117
```

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-tmp117
```

CHAPTER

THREE

USAGE EXAMPLE

```
import time
import board
import adafruit_tmp117

i2c = board.I2C()  # uses board.SCL and board.SDA
tmp117 = adafruit_tmp117.TMP117(i2c)

while True:
    print("Temperature: %.2f degrees C"%tmp117.temperature)
    time.sleep(1)
```

CHAPTER

FOUR

CONTRIBUTING

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

CHAPTER

FIVE

DOCUMENTATION

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

CHAPTER

SIX

TABLE OF CONTENTS

6.1 Simple test

Ensure your device works with this simple test.

Listing 1: examples/tmp117_simpletest.py

```
# SPDX-FileCopyrightText: 2020 Bryan Siepert, written for Adafruit Industries
#
# SPDX-License-Identifier: Unlicense
import time
import board
import adafruit_tmp117

i2c = board.I2C()  # uses board.SCL and board.SDA
tmp117 = adafruit_tmp117.TMP117(i2c)

while True:
    print("Temperature: %.2f degrees C" % tmp117.temperature)
    time.sleep(1)
```

6.2 Temperature limits and alerts

Set high and low temperature limits and be alerted when they are surpassed.

Listing 2: examples/tmp117_limits_test.py

```
# SPDX-FileCopyrightText: 2020 Bryan Siepert, written for Adafruit Industries
#
# SPDX-License-Identifier: Unlicense
import time
import board
from adafruit_tmp117 import TMP117, AlertMode

i2c = board.I2C()  # uses board.SCL and board.SDA

tmp117 = TMP117(i2c)

tmp117.high_limit = 25
```

(continues on next page)

(continued from previous page)

```
tmp117.low_limit = 10

print("\nHigh limit", tmp117.high_limit)
print("Low limit", tmp117.low_limit)

# Try changing `alert_mode`  to see how it modifies the behavior of the alerts.
# tmp117.alert_mode = AlertMode.WINDOW #default
# tmp117.alert_mode = AlertMode.HYSTERESIS

print("Alert mode:", AlertMode.string[tmp117.alert_mode])
print("\n\n")
while True:
    print("Temperature: %.2f degrees C" % tmp117.temperature)
    alert_status = tmp117.alert_status
    print("High alert:", alert_status.high_alert)
    print("Low alert:", alert_status.low_alert)
    print("")
    time.sleep(1)
```

6.3 Measurement averaging and rate

Adjust the number of samples averaged for every reported temperature, and adjust the time beween new measurement reports

Listing 3: examples/tmp117_rate_and_averaging_test.py

```
# SPDX-FileCopyrightText: 2020 Bryan Siepert, written for Adafruit Industries
#
# SPDX-License-Identifier: Unlicense
# pylint:disable=no-member

# This example is best viewed using a serial plotter
# such as the one built into the Mu editor.
import time
import board
from adafruit_tmp117 import TMP117, AverageCount, MeasurementDelay

i2c = board.I2C()  # uses board.SCL and board.SDA
tmp117 = TMP117(i2c)

# uncomment different options below to see how it affects the reported temperature
# tmp117.averaged_measurements = AverageCount.AVERAGE_1X
# tmp117.averaged_measurements = AverageCount.AVERAGE_8X
# tmp117.averaged_measurements = AverageCount.AVERAGE_32X
# tmp117.averaged_measurements = AverageCount.AVERAGE_64X

# tmp117.measurement_delay = MeasurementDelay.DELAY_0_0015_S
# tmp117.measurement_delay = MeasurementDelay.DELAY_0_125_S
# tmp117.measurement_delay = MeasurementDelay.DELAY_0_250_S
# tmp117.measurement_delay = MeasurementDelay.DELAY_0_500_S
```

(continues on next page)

(continued from previous page)

```
# tmp117.measurement_delay = MeasurementDelay.DELAY_1_S
# tmp117.measurement_delay = MeasurementDelay.DELAY_4_S
# tmp117.measurement_delay = MeasurementDelay.DELAY_8_S
# tmp117.measurement_delay = MeasurementDelay.DELAY_16_S

print(
    "Number of averaged samples per measurement:",
    AverageCount.string[tmp117.averaged_measurements],
)
print(
    "Minimum time between measurements:",
    MeasurementDelay.string[tmp117.measurement_delay],
    "seconds",
)
print("")

while True:
    print("Temperature:", tmp117.temperature)
    time.sleep(0.01)
```

6.4 Temperature offset

Set an offset that will be applied to each measurement, to account for measurement biases in the sensor's environment

Listing 4: examples/tmp117_offset_test.py

```
# SPDX-FileCopyrightText: 2020 Bryan Siepert, written for Adafruit Industries
#
# SPDX-License-Identifier: Unlicense
import time
import board
import adafruit_tmp117

i2c = board.I2C()  # uses board.SCL and board.SDA

tmp117 = adafruit_tmp117.TMP117(i2c)

print("Temperature without offset: %.2f degrees C" % tmp117.temperature)
tmp117.temperature_offset = 10.0
while True:
    print("Temperature w/ offset: %.2f degrees C" % tmp117.temperature)
    time.sleep(1)
```

6.5 Single Measurement Test

Take different sample number and average to give a single temperature measure

Listing 5: examples/tmp117_single_measurement_test.py

```
# SPDX-FileCopyrightText: 2020 Bryan Siepert, written for Adafruit Industries
#
# SPDX-License-Identifier: Unlicense
import board
from adafruit_tmp117 import TMP117, AverageCount

i2c = board.I2C()  # uses board.SCL and board.SDA
tmp117 = TMP117(i2c)

# uncomment different options below to see how it affects the reported temperature
# and measurement time

# tmp117.averaged_measurements = AverageCount.AVERAGE_1X
# tmp117.averaged_measurements = AverageCount.AVERAGE_8X
# tmp117.averaged_measurements = AverageCount.AVERAGE_32X
# tmp117.averaged_measurements = AverageCount.AVERAGE_64X

print(
    "Number of averaged samples per measurement:",
    AverageCount.string[tmp117.averaged_measurements],
)
print(
    "Reads should take approximately",
    AverageCount.string[tmp117.averaged_measurements] * 0.0155,
    "seconds",
)

while True:
    print("Single measurement: %.2f degrees C" % tmp117.take_single_measurememt())
    # time.sleep(1)
```

6.6 adafruit_tmp117

CircuitPython library for the TI TMP117 Temperature sensor

- Author(s): Bryan Siepert, Ian Grant

parts based on SparkFun_TMP117_Arduino_Library by Madison Chodikov @ SparkFun Electronics: https://github.com/sparkfunX/Qwiic_TMP117 https://github.com/sparkfun/SparkFun_TMP117_Arduino_Library

Serial number register information: https://e2e.ti.com/support/sensors/f/1023/t/815716?TMP117-Reading-Serial-Number-from-EEPROM

6.6.1 Implementation Notes

Hardware:

- Adafruit TMP117 ±0.1°C High Accuracy I2C Temperature Sensor (Product ID: 4821)

Software and Dependencies:

- Adafruit CircuitPython firmware for the supported boards: https://circuitpython.org/downloads
- Adafruit's Bus Device library: https://github.com/adafruit/Adafruit_CircuitPython_BusDevice
- Adafruit's Register library: https://github.com/adafruit/Adafruit_CircuitPython_Register

class adafruit_tmp117.**AlertMode**
Options for *alert_mode*. See *alert_mode* for more information.

class adafruit_tmp117.**AlertStatus**(*high_alert*, *low_alert*)
Create new instance of AlertStatus(high_alert, low_alert)

property high_alert
Alias for field number 0

property low_alert
Alias for field number 1

class adafruit_tmp117.**AverageCount**
Options for *averaged_measurements*

class adafruit_tmp117.**MeasurementDelay**
Options for *measurement_delay*

class adafruit_tmp117.**MeasurementMode**
Options for *measurement_mode*. See *measurement_mode* for more information.

class adafruit_tmp117.**TMP117**(*i2c_bus*, *address=72*)
Library for the TI TMP117 high-accuracy temperature sensor

property alert_mode
Sets the behavior of the *low_limit*, *high_limit*, and *alert_status* properties.

When set to `AlertMode.WINDOW`, the *high_limit* property will unset when the measured temperature goes below *high_limit*. Similarly *low_limit* will be True or False depending on if the measured temperature is below (`False`) or above(`True`) *low_limit*.

When set to `AlertMode.HYSTERESIS`, the *high_limit* property will be set to `False` when the measured temperature goes below *low_limit*. In this mode, the *low_limit* property of *alert_status* will not be set.

The default is `AlertMode.WINDOW`

property alert_status
The current triggered status of the high and low temperature alerts as a AlertStatus named tuple with attributes for the triggered status of each alert.

```
import board
import adafruit_tmp117
i2c = board.I2C()  # uses board.SCL and board.SDA

tmp117 = adafruit_tmp117.TMP117(i2c)

tmp117.high_limit = 25
```

(continues on next page)

(continued from previous page)

```
tmp117.low_limit = 10

print("High limit", tmp117.high_limit)
print("Low limit", tmp117.low_limit)

# Try changing `alert_mode`  to see how it modifies the behavior of the alerts.
# tmp117.alert_mode = AlertMode.WINDOW #default
# tmp117.alert_mode = AlertMode.HYSTERESIS

print("Alert mode:", AlertMode.string[tmp117.alert_mode])
print("")
print("")
while True:
    print("Temperature: %.2f degrees C" % tmp117.temperature)
    alert_status = tmp117.alert_status
    print("High alert:", alert_status.high_alert)
    print("Low alert:", alert_status.low_alert)
    print("")
    time.sleep(1)
```

property averaged_measurements

The number of measurements that are taken and averaged before updating the temperature measurement register. A larger number will reduce measurement noise but may also affect the rate at which measurements are updated, depending on the value of *measurement_delay*

Note that each averaged measurement takes 15.5ms which means that larger numbers of averaged measurements may make the delay between new reported measurements to exceed the delay set by *measurement_delay*

property high_limit

The high temperature limit in degrees Celsius. When the measured temperature exceeds this value, the *high_alert* attribute of the *alert_status* property will be True. See the documentation for *alert_status* for more information

initialize()

Configure the sensor with sensible defaults. *initialize* is primarily provided to be called after *reset*, however it can also be used to easily set the sensor to a known configuration

property low_limit

The low temperature limit in degrees Celsius. When the measured temperature goes below this value, the *low_alert* attribute of the *alert_status* property will be True. See the documentation for *alert_status* for more information

property measurement_delay

The minimum amount of time between measurements in seconds. Must be a *MeasurementDelay*. The specified amount may be exceeded depending on the current setting off *averaged_measurements* which determines the minimum time needed between reported measurements.

property measurement_mode

Sets the measurement mode, specifying the behavior of how often measurements are taken.

measurement_mode must be one of:

Mode	Behavior
`MeasurementMode.CONTINUOUS`	Measurements are made at the interval determined by *averaged_measurements* and *measurement_delay*. *temperature* returns the most recent measurement
`MeasurementMode.ONE_SHOT`	Take a single measurement with the current number of *averaged_measurements* and switch to `SHUTDOWN` when finished. *temperature* will return the new measurement until *measurement_mode* is set to `CONTINUOUS` or `ONE_SHOT` is set again.
`MeasurementMode.SHUTDOWN`	The sensor is put into a low power state and no new measurements are taken. *temperature* will return the last measurement until a new *measurement_mode* is selected.

reset()
Reset the sensor to its unconfigured power-on state

property serial_number
A 48-bit, factory-set unique identifier for the device.

take_single_measurememt()
Perform a single measurement cycle respecting the value of *averaged_measurements*, returning the measurement once complete. Once finished the sensor is placed into a low power state until `take_single_measurement()` or *temperature* are read.

Note: if *averaged_measurements* is set to a high value there will be a notable delay before the temperature measurement is returned while the sensor takes the required number of measurements

property temperature
The current measured temperature in degrees Celsius

property temperature_offset
User defined temperature offset to be added to measurements from *temperature*

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

INDEX