Adafruit PCA9685 Library Documentation

Release 1.0

Radomir Dopieralski

Jul 31, 2018

Contents

Driver for the PCA9685, a 16-channel, 12-bit PWM chip

CHAPTER 1

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 2

Usage Example

See examples/pca9685_simpletest.py for a demo of the 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-pca9685 --
↪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/pca9685_simpletest.py

```
# This simple test outputs a 50% duty cycle PWM single on the 0th channel. Connect an␣
↪LED and
# resistor in series to the pin to visualize duty cycle changes and its impact on␣
↪brightness.

from board import SCL, SDA
import busio

# Import the PCA9685 module.
from adafruit_pca9685 import PCA9685

# Create the I2C bus interface.
i2c_bus = busio.I2C(SCL, SDA)

# Create a simple PCA9685 class instance.
pca = PCA9685(i2c_bus)

# Set the PWM frequency to 60hz.
pca.frequency = 60

# Set the PWM duty cycle for channel zero to 50%. duty_cycle is 16 bits to match␣
↪other PWM objects
# but the PCA9685 will only actually give 12 bits of resolution.
pca.channels[0].duty_cycle = 0x7fff
```

Listing 2: examples/calibration.py

```
# This advanced example can be used to compute a more precise reference_clock_speed. Use an
# oscilloscope or logic analyzer to measure the signal frequency and type the results into the
# prompts. At the end it'll give you a more precise value around 25 mhz for your reference clock
# speed.

import time

from board import SCL, SDA
import busio

# Import the PCA9685 module.
from adafruit_pca9685 import PCA9685

# Create the I2C bus interface.
i2c_bus = busio.I2C(SCL, SDA)

# Create a simple PCA9685 class instance.
pca = PCA9685(i2c_bus)

# Set the PWM frequency to 100hz.
pca.frequency = 100

input("Press enter when ready to measure default frequency.")

# Set the PWM duty cycle for channel zero to 50%. duty_cycle is 16 bits to match other PWM objects
# but the PCA9685 will only actually give 12 bits of resolution.
print("Running with default calibration")
pca.channels[0].duty_cycle = 0x7fff
time.sleep(1)
pca.channels[0].duty_cycle = 0

measured_frequency = float(input("Frequency measured: "))
print()

pca.reference_clock_speed = pca.reference_clock_speed * (measured_frequency / pca.frequency)
# Set frequency again so we can get closer. Reading it back will produce the real value.
pca.frequency = 100

input("Press enter when ready to measure coarse calibration frequency.")
pca.channels[0].duty_cycle = 0x7fff
time.sleep(1)
pca.channels[0].duty_cycle = 0
measured_after_calibration = float(input("Frequency measured: "))
print()

reference_clock_speed = measured_after_calibration * 4096 * pca.prescale_reg

print("Real reference clock speed: {0:.0f}".format(reference_clock_speed))
```

Listing 3: examples/servo.py

```
# This example moves a servo its full range (180 degrees by default) and then back.

from board import SCL, SDA
import busio

# Import the PCA9685 module.
from adafruit_pca9685 import PCA9685

# This example also relies on the Adafruit motor library available here:
# https://github.com/adafruit/Adafruit_CircuitPython_Motor
from adafruit_motor import servo

i2c = busio.I2C(SCL, SDA)

# Create a simple PCA9685 class instance.
pca = PCA9685(i2c)
pca.frequency = 50

# To get the full range of the servo you will likely need to adjust the min_pulse and
↪max_pulse to
# match the stall points of the servo.
# This is an example for the Sub-micro servo: https://www.adafruit.com/product/2201
# servo7 = servo.Servo(pca.channels[7], min_pulse=580, max_pulse=2480)
# This is an example for the Micro Servo - High Powered, High Torque Metal Gear:
#   https://www.adafruit.com/product/2307
# servo7 = servo.Servo(pca.channels[7], min_pulse=600, max_pulse=2400)
# This is an example for the Standard servo - TowerPro SG-5010 - 5010:
#   https://www.adafruit.com/product/155
# servo7 = servo.Servo(pca.channels[7], min_pulse=600, max_pulse=2500)
# This is an example for the Analog Feedback Servo: https://www.adafruit.com/product/
↪1404
# servo7 = servo.Servo(pca.channels[7], min_pulse=600, max_pulse=2600)

# The pulse range is 1000 - 2000 by default.
servo7 = servo.Servo(pca.channels[7])

for i in range(180):
    servo7.angle = i
for i in range(180):
    servo7.angle = 180 - i
pca.deinit()
```

5.2 adafruit_pca9685

Driver for the PCA9685 PWM control IC. Its commonly used to control servos, leds and motors.

See also:

The Adafruit CircuitPython Motor library can be used to control the PWM outputs for specific uses instead of generic duty_cycle adjustments.

- Author(s): Scott Shawcroft

5.2.1 Implementation Notes

Hardware:

- Adafruit 16-Channel 12-bit PWM/Servo Driver - I2C interface - PCA9685 (Product ID: 815)

Software and Dependencies:

- Adafruit CircuitPython firmware for the ESP8622 and M0-based boards: https://github.com/adafruit/circuitpython/releases
- 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_pca9685.**PCA9685**(*i2c_bus*, ***, *address=64*, *reference_clock_speed=25000000*)
Initialise the PCA9685 chip at `address` on `i2c_bus`.

The internal reference clock is 25mhz but may vary slightly with environmental conditions and manufacturing variances. Providing a more precise `reference_clock_speed` can improve the accuracy of the frequency and duty_cycle computations. See the `calibration.py` example for how to derive this value by measuring the resulting pulse widths.

Parameters

- **i2c_bus** (*I2C*) – The I2C bus which the PCA9685 is connected to.
- **address** (*int*) – The I2C address of the PCA9685.
- **reference_clock_speed** (*int*) – The frequency of the internal reference clock in Herz.

channels = None
Sequence of 16 *PWMChannel* objects. One for each channel.

deinit()
Stop using the pca9685.

frequency
The overall PWM frequency in herz.

reference_clock_speed = None
The reference clock speed in Hz.

reset()
Reset the chip.

class adafruit_pca9685.**PCAChannels**(*pca*)
Lazily creates and caches channel objects as needed. Treat it like a sequence.

class adafruit_pca9685.**PWMChannel**(*pca*, *index*)
A single PCA9685 channel that matches the `PWMOut` API.

duty_cycle
16 bit value that dictates how much of one cycle is high (1) versus low (0). 0xffff will always be high, 0 will always be low and 0x7fff will be half high and then half low.

frequency
The overall PWM frequency in herz.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

Index