i2cEncoder Library Documentation

Release 1.0

Ben Shockley

Feb 11, 2022

Contents

This is a CircuitPython helper library for the i2c Encoder made by Simon Caron: http://www.duppa.net/i2c-encoder-v2-1/

CHAPTER 1

Dependencies

This driver depends on:

- Adafruit CircuitPython
- Bus Device

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

CHAPTER 2

Installing from PyPI

Note: This library is not available on PyPI yet. Install documentation is included as a standard element. Stay tuned for PyPI availability!

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-i2cencoderlibv21
```

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

```
sudo pip3 install adafruit-circuitpython-i2cencoderlibv21
```

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-i2cencoderlibv21
```

CHAPTER 3

Usage Example

```
# imports
import time
import struct
import busio
import board
import digitalio
import i2cencoderlibv21

# Setup the Inturrpt Pin from the encoder.
INT = digitalio.DigitalInOut(board.A3)
INT.direction = digitalio.Direction.INPUT
INT.pull = digitalio.Pull.UP

# Initialize the device.
i2c = busio.I2C(board.SCL, board.SDA)
encoder = i2cencoderlibv21.I2CEncoderLibV21(i2c, 0x21)

def EncoderChange():
    encoder.write_rgb_code(0x00FF00)
    valBytes = struct.unpack('>i', encoder.readCounter32())
    print ('Changed: {}'.format(valBytes[0]))

def EncoderIncrement():
    print ('Incrementing...')

def EncoderPush():
    encoder.write_rgb_code(0x0000FF)
    print ('Encoder Pushed!')

def EncoderRelease():
    encoder.write_rgb_code(0x00FFFF)
    print ('Encoder Released!')

def EncoderDoublePush():
```

(continues on next page)

(continued from previous page)

```
    encoder.write_rgb_code(0xFF00FF)
    print ('Encoder Double Push!')

def EncoderMax():
    encoder.write_rgb_code(0xFF0000)
    print ('Encoder max!')

def EncoderMin():
    encoder.write_rgb_code(0xFF0000)
    print ('Encoder min!')

def EncoderFade():
    encoder.write_rgb_code(0x000000)

def Encoder_INT():
    encoder.update_status()

# Start by resetting the encoder. Reset takes 400us , so let us give it time to␣
↪settle.
encoder.reset()
time.sleep(.1)

# When the board was initialized, the default config was loaded.
# Here we can override that config if we want.
encconfig = (i2cencoderlibv21.INT_DATA | i2cencoderlibv21.WRAP_DISABLE
             | i2cencoderlibv21.DIRE_RIGHT | i2cencoderlibv21.IPUP_DISABLE
             | i2cencoderlibv21.RMOD_X1 | i2cencoderlibv21.RGB_ENCODER)
encoder.begin(encconfig)

# Setup other varibles
encoder.write_counter(0)
encoder.write_max(10)
encoder.write_min(-10)
encoder.write_step_size(1)
encoder.write_antibounce_period(25)
encoder.write_double_push_period(50)
encoder.write_fade_rgb(5)

# Declare callbacks
encoder.onChange = EncoderChange
encoder.onIncrement = EncoderIncrement
encoder.onButtonRelease = EncoderRelease
encoder.onButtonPush = EncoderPush
encoder.onButtonDoublePush = EncoderDoublePush
encoder.onMax = EncoderMax
encoder.onMin = EncoderMin
encoder.onFadeProcess = EncoderFade

# Autoconfigure the interrupt register according to the callbacks declared.
# Must be called after declaring callbacks.
encoder.autoconfig_interrupt()

while True:
    if not INT.value:       #If INT pin goes LOW - we know the encoder status changed.
        Encoder_INT()
```

CHAPTER 4

Contributing

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

CHAPTER 5

Documentation

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

CHAPTER 6

Table of Contents

6.1 Simple test

Ensure your device works with this simple test.

Listing 1: examples/i2cencoder_simpletest.py

```
# Simple test for an RGB encoder.

# Author: Ben Shockley

# imports
import time
import struct
import busio
import board
import digitalio
import i2cencoderlibv21

# Setup the Inturrpt Pin from the encoder.
INT = digitalio.DigitalInOut(board.A3)
INT.direction = digitalio.Direction.INPUT
INT.pull = digitalio.Pull.UP

# Initialize the device.
i2c = busio.I2C(board.SCL, board.SDA)
encoder = i2cencoderlibv21.I2CEncoderLibV21(i2c, 0x21)

def EncoderChange():
    encoder.write_rgb_code(0x00FF00)
    valBytes = struct.unpack('>i', encoder.readCounter32())
    print ('Changed: {}'.format(valBytes[0]))

def EncoderIncrement():
```

(continues on next page)

(continued from previous page)

```
    print ('Incrementing...')

def EncoderPush():
    encoder.write_rgb_code(0x0000FF)
    print ('Encoder Pushed!')

def EncoderRelease():
    encoder.write_rgb_code(0x00FFFF)
    print ('Encoder Released!')

def EncoderDoublePush():
    encoder.write_rgb_code(0xFF00FF)
    print ('Encoder Double Push!')

def EncoderMax():
    encoder.write_rgb_code(0xFF0000)
    print ('Encoder max!')

def EncoderMin():
    encoder.write_rgb_code(0xFF0000)
    print ('Encoder min!')

def EncoderFade():
    encoder.write_rgb_code(0x000000)

def Encoder_INT():
    encoder.update_status()

# Start by resetting the encoder. Reset takes 400us , so let us give it time to␣
↪settle.
encoder.reset()
time.sleep(.1)

# When the board was initialized, the default config was loaded.
# Here we can override that config if we want.
encconfig = (i2cencoderlibv21.INT_DATA | i2cencoderlibv21.WRAP_DISABLE
             | i2cencoderlibv21.DIRE_RIGHT | i2cencoderlibv21.IPUP_DISABLE
             | i2cencoderlibv21.RMOD_X1 | i2cencoderlibv21.RGB_ENCODER)
encoder.begin(encconfig)

# Setup other varibles
encoder.write_counter(0)
encoder.write_max(10)
encoder.write_min(-10)
encoder.write_step_size(1)
encoder.write_antibounce_period(25)
encoder.write_double_push_period(50)
encoder.write_fade_rgb(5)

# Declare callbacks
encoder.onChange = EncoderChange
encoder.onIncrement = EncoderIncrement
encoder.onButtonRelease = EncoderRelease
encoder.onButtonPush = EncoderPush
encoder.onButtonDoublePush = EncoderDoublePush
encoder.onMax = EncoderMax
encoder.onMin = EncoderMin
```

(continues on next page)

(continued from previous page)

```
encoder.onFadeProcess = EncoderFade

# Autoconfigure the interrupt register according to the callbacks declared.
# Must be called after declaring callbacks.
encoder.autoconfig_interrupt()

while True:
    if not INT.value:       #If INT pin goes LOW - we know the encoder status changed.
        Encoder_INT()
```

6.2 i2cencoderv21

A CircuitPython library for the I2CEncoder v2.1 board from Simon Caron.

- Author(s): Ben Shockley & Simon Caron

6.2.1 Implementation Notes

Hardware: Software and Dependencies: * Adafruit CircuitPython firmware for the supported boards:

https://github.com/adafruit/circuitpython/releases

- Adafruit's Bus Device library: https://github.com/adafruit/Adafruit_CircuitPython_BusDevice

class i2cencoderlibv21.**I2CEncoderLibV21**(*i2c_bus*, *address*)

Helper library for the i2c Encoder from Simon Caron http://www.duppa.net/i2c-encoder-v2-1/

Parameters

- **i2c_bus** (*I2C*) – The I2C bus the encoder is connected to.
- **address** (*byte*) – The I2C slave address of the encoder
- **config** (*byte*) – Initial setup config. Default is CONFIG_DEFAULT

autoconfig_interrupt()
Automatically configures the interrupt register according to the callback(s) declared.

begin(*config=32*)
Set the encoder's general configuration.

Parameters **or 2-byte config** (*1*) – Configuration to be set.

read_eeprom(*add*)
Read the data in the EEPROM

Parameters **add** (*byte*) – registry address from which to read.

reset()
Reset the encoder.

setInterrupts(*interrupts*)
Set the Interrupt Configuration Manually.

Parameters **interrupts** (*byte*) – byte contining interrupt configuration.

update_status()
Runs the callback for the encoder status.

write_eeprom(*add*, *data*)
write the data to the EEPROM

Parameters

- **add** (*byte*) – registry address where to write.
- **data** (*byte*) – 1-Byte of data to write.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

i

Index