CodeBug I2C Tether Documentation

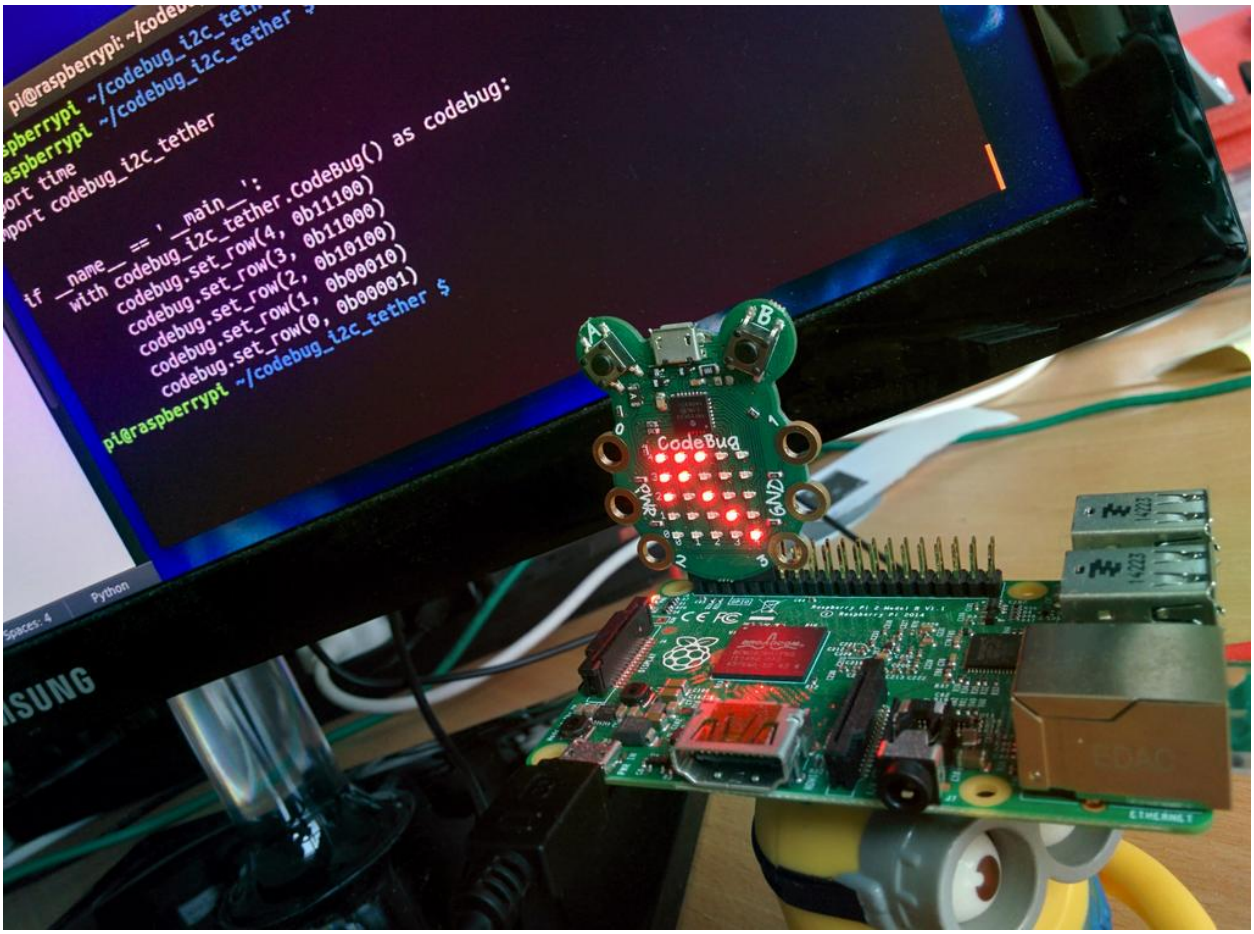
Release 0.3.0

Thomas Preston

January 21, 2017

Contents

The *codebug_i2c_tether* Python module provides functions and classes for interacting with CodeBug when tethered over I2C.



Links:

- CodeBug Website
- GitHub

Contents:

CHAPTER 1

Installation

1.1 Setting up CodeBug

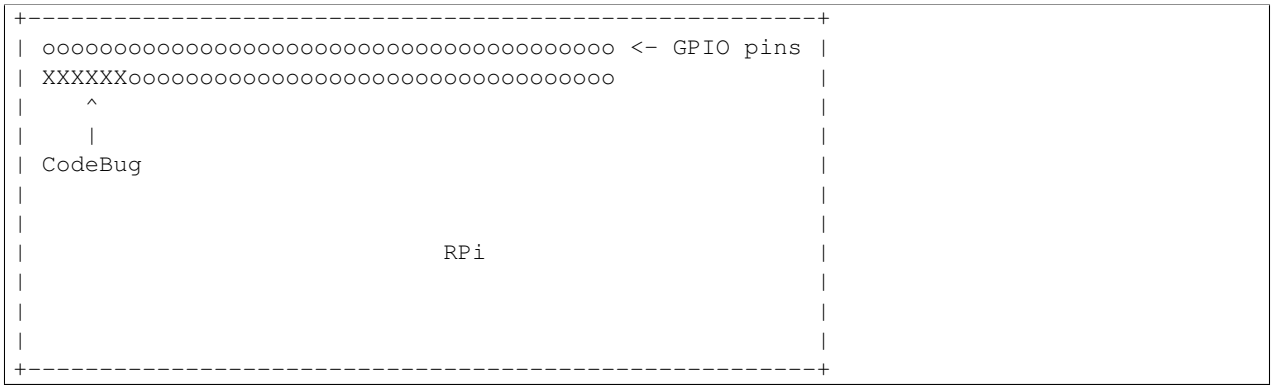
In order to use CodeBug with codebug_i2c_tether you need to program CodeBug with `codebug_i2c_tether.cbg` ().

To do this, hold down button A and plug in CodeBug via USB — it should appear as a USB drive — then copy the `codebug_i2c_tether.cbg` file onto it. CodeBug is now ready to be used via serial USB. Press button B to exit programming mode.

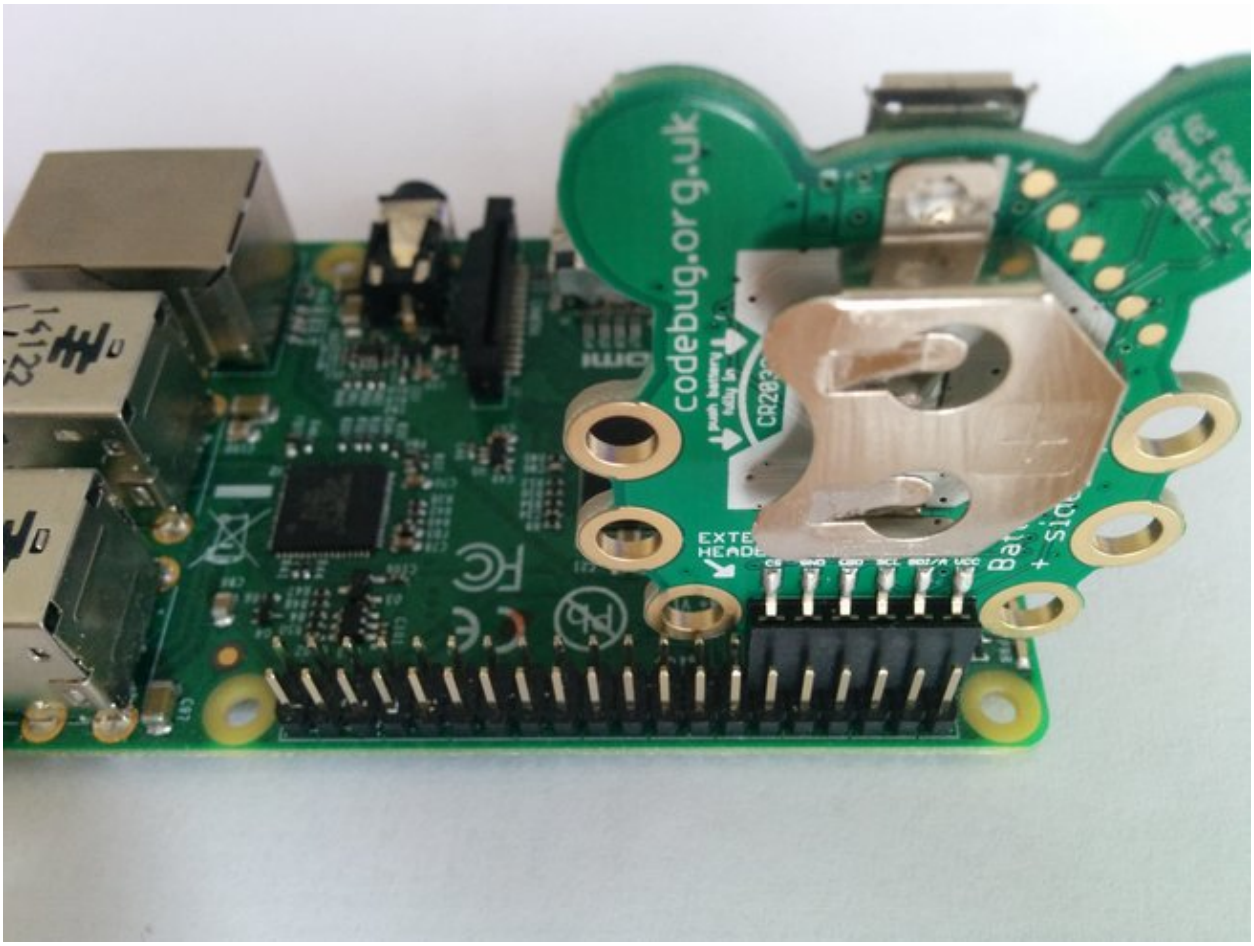
Note: When CodeBug is connected to a computer via USB is should now appear as a serial device. To reprogram CodeBug: hold down button A and (re)plug it into a USB port.

Unplug CodeBug, remove the battery (if one is inserted) and then plug CodeBug's extension header into the I2C GPIO pins on the Raspberry Pi. These are the left-most inner row GPIO pins on the Raspberry Pi with CodeBug facing in.

Here is a diagram of this configuration:



And here is a picture of the same thing:



1.2 Install codebug_i2c_tether on a Raspberry Pi

First, make sure you have enabled I2C by running:

```
sudo raspi-config
```

and then navigating to:

```
Advanced Options > Would you like the I2C interface to be enabled? > Yes
Would you like the I2C kernel module to be loaded by default? > Yes
```

Then reboot.

1.2.1 Install Python

Python should already be installed but for good measure:

```
sudo apt-get install python3
```

To install pip, securely download get-pip.py.

Then run the following:

```
sudo python3 get-pip.py
```

1.2.2 Install codebug_i2c_tether

To install codebug_i2c_tether, open up a terminal and type:

```
pip3 install codebug_i2c_tether
```

To test it has worked, plug in CodeBug and open a Python shell by typing:

```
python3
```

Your command prompt should have changed to:

```
>>> _
```

Now type:

```
>>> import codebug_i2c_tether
>>> with codebug_i2c_tether.CodeBug() as codebug:
...     codebug.set_pixel(2, 2, 1)
...
```

The middle pixel on your CodeBug should light up.

See *Examples* for more ways to use codebug_i2c_tether.

CHAPTER 2

Examples

2.1 Basics

```
>>> import codebug_i2c_tether

>>> codebug = codebug_i2c_tether.CodeBug()  # create a CodeBug object
>>> codebug.open()

>>> codebug.set_pixel(2, 1, 1)  # turn on the LED at (2, 1)
>>> codebug.set_pixel(0, 0, 0)  # turn off the LED at (0, 0)
>>> codebug.get_pixel(0, 1)     # return the LED state at (0, 1)
1

>>> codebug.clear()  # turn off all LEDs

>>> codebug.set_row(0, 5)         # set row 0 to 5  (binary: 00101)
>>> codebug.set_row(1, 0x1a)      # set row 1 to 26 (binary: 11010)
>>> codebug.set_row(2, 0b10101)   # set row 2 to 21 (binary: 10101)
>>> bin(codebug.get_row(2))
'0b10101'

>>> codebug.set_col(0, 0x1f)  # turn on all LEDs in column 0
>>> codebug.get_col(0)
31

>>> codebug.get_input('A')  # returns the state of button 'A'
0
>>> codebug.get_input(0)  # returns the state of input 0
0

>>> codebug.set_leg_io(0, 0)  # set leg 0 to output
>>> codebug.set_output(0, 1)  # turn leg 0 'on' (1)
```

2.2 Sprites

You can use the sprites library to quickly draw things on CodeBug's display.

```
>>> import codebug_i2c_tether
>>> import codebug_i2c_tether.sprites
```

```
>>> # create a 3x3 square with the middle pixel off
>>> square_sprite = codebug_i2c_tether.sprites.Sprite(3, 3)
>>> square_sprite.set_row(0, 0b111)
>>> square_sprite.set_row(1, 0b101)
>>> square_sprite.set_row(2, 0b111)

>>> # draw it in the middle of CodeBug
>>> codebug = codebug_i2c_tether.CodeBug()
>>> codebug.open()
>>> codebug.draw_sprite(1, 1, square_sprite)

>>> # write some text
>>> message = codebug_i2c_tether.sprites.StringSprite('Hello CodeBug!')
>>> codebug.draw_sprite(0, 0, message)
>>> # move it along
>>> codebug.draw_sprite(-2, 0, message)
```

You can do some more interesting things with Sprites:

```
>>> import codebug_i2c_tether.sprites

>>> sprite = codebug_i2c_tether.sprites.Sprite(10, 10)

>>> # basic gets and sets
>>> sprite.set_pixel(0, 0, 1)
>>> sprite.get_pixel(0, 0)
1
>>> sprite.set_row(0, 0, 0b1111111111)
>>> sprite.get_row(0)
1023
>>> sprite.set_col(0, 0, 0b1111111111)
>>> sprite.get_col(0)
1023

>>> # transform the sprite
>>> sprite.invert_horizontal()
>>> sprite.invert_vertical()
>>> sprite.invert_diagonal()
>>> sprite.rotate90()
>>> sprite.rotate90(rotation=2)  # rotate 180 degrees

>>> # clone or extract parts of the sprite
>>> dolly_sprite = sprite.clone()
>>> rectangle = sprite.get_sprite(3, 3, 5, 2)

>>> # draw other sprites
>>> sprite.render_sprite(1, 1, rectangle)
```

You can also change the direction text is written in:

```
>>> from codebug_i2c_tether.sprites import StringSprite
>>> left_to_right_msg = StringSprite('Hello CodeBug!')
>>> right_to_left_msg = StringSprite('Hello CodeBug!', direction='L')
>>> top_to_bottom_msg = StringSprite('Hello CodeBug!', direction='D')
>>> bottom_to_top_msg = StringSprite('Hello CodeBug!', direction='U')
```

2.3 Analogue Input

You can read analogue inputs from all 8 of CodeBug's I/O legs/extension pins:

```
>>> import codebug_i2c_tether
>>> from codebug_i2c_tether import (IO_DIGITAL_INPUT,
...                                 IO_ANALOGUE_INPUT,
...                                 IO_PWM_OUTPUT,
...                                 IO_DIGITAL_OUTPUT)
...
>>> codebug = codebug_i2c_tether.CodeBug()
>>> codebug.open()
>>> codebug.set_leg_io(0, IO_ANALOGUE_INPUT)
>>> codebug.read_analogue(0)
128
```

2.4 PWM Output

You can drive one synchronised PWM (Pulse Width Modulation) signal out of the first three legs on CodeBug. That is, the same PWM signal will be driven out of legs configured as PWM output:

```
>>> import codebug_i2c_tether
>>> from codebug_i2c_tether import (IO_DIGITAL_INPUT,
...                                 IO_ANALOGUE_INPUT,
...                                 IO_PWM_OUTPUT,
...                                 IO_DIGITAL_OUTPUT,
...                                 T2_PS_1_1,
...                                 T2_PS_1_4,
...                                 T2_PS_1_16)

>>> codebug = codebug_i2c_tether.CodeBug()
>>> codebug.open()
>>> # configure legs 0 and 1 to be PWM output
>>> codebug.set_leg_io(0, IO_PWM_OUTPUT)
>>> codebug.set_leg_io(1, IO_PWM_OUTPUT)
>>> # shortcut method to specify a frequency (the note C == 1046 Hz)
>>> codebug.pwm_freq(1046)
>>> time.sleep(2)
>>> codebug.pwm_off()
```

Or you can be more specific with the duty cycle and timing:

```
>>> # pwm on with 1:4 prescaler and 75% duty cycle @ ~977Hz
>>> # Timer 2 prescale: 4Mhz clock / 4 = 1MHz timer speed
>>> # full_period: 255 << 2 = 1024  (timer resets at this count; PWM = 1)
>>> # on_period: 765 (PWM goes to zero at this count; PWM = 0)
>>> # therefore duty cycle here is 75%
>>> codebug.pwm_on(T2_PS_1_4, 255, 765)
>>> time.sleep(2)
>>> codebug.pwm_off()
```

2.5 Servos

It is possible to drive up to eight servos from CodeBug. Servos typically operate by sending them a PWM (Pulse Width Modulation) signal with a 20ms period and a 1-2ms duty cycle which controls the angle of the servo. For example:

```
 Pulse Length
  <-1-2ms->
1+---------+
 |         |
0+         +-------------------------------------+
  <------------------20ms (50Hz)----------------->
                     PWM Period
```

A duty cycle of 1ms will typically correspond to 0° rotation and a duty cycle of 2ms will typically correspond to 180° rotation. Although the precise values may differ depending on the type of servo.

In order to drive servos from CodeBug you can call the *servo_set()* method which takes the servo index (which leg you are driving the servo from) and the the pulse length specified in N 0.5μs. For example:

```
>>> import codebug_i2c_tether
>>> from codebug_i2c_tether import (IO_DIGITAL_OUTPUT, scale)

>>> # init CodeBug and configure leg 0 to be digital output
>>> codebug = codebug_i2c_tether.CodeBug()
>>> codebug.open()
>>> codebug.set_leg_io(0, IO_DIGITAL_OUTPUT)

>>> # set servo on leg 0 with pulse length of 1ms (2000 * 0.5µs)
>>> codebug.servo_set(0, 2000)

>>> # stop driving the servo on leg 0
>>> codebug.servo_set(0, 0)
```

You can use the scale function to easily calculate the required pulse length value like so:

```
>>> import codebug_i2c_tether
>>> from codebug_i2c_tether import (IO_DIGITAL_OUTPUT, scale)

>>> # scale 50 in the range 0-100 to the range 0-255
>>> scale(50, 0, 100, 0, 255)
127

>>> # scale 10 in the range 0-30 to the range 100-400
>>> scale(10, 0, 30, 100, 400)
200

>>> # scale 90° in the range 0-180° to the range 2000-4000 * 0.5µs
>>> scale(90, 0, 180, 2000, 4000)
3000

>>> # init CodeBug and configure leg 0 to be digital output
>>> codebug = codebug_i2c_tether.CodeBug()
>>> codebug.open()
>>> codebug.set_leg_io(0, IO_DIGITAL_OUTPUT)

>>> # drive the servo to be at 90 degrees
>>> codebug.servo_set(0, scale(90, 0, 180, 2000, 4000))
```

CHAPTER 3

Indices and tables

- genindex
- modindex
- search