Python LoRa Module - lora.py

This library provides access to one LoRa (Long Range Radio) module on the PITS Zero board or the add-on board for the PITS+ board.

Basic Usage

```
from lora import *
import time

print("Create LoRa object")
mylora = LoRa(Channel=0, Frequency=434.450, Mode=1)

print("Send message")
mylora.send_text("$$Hello World\n")

while mylora.is_sending():
    time.sleep(0.01)
print("DONE")
```

The LoRa object is **non-blocking**. The calling code can either poll to find out when the transmission has completed, or can be notified via a callback.

Reference

Object Creation

```
LoRa(Channel=0, Frequency=434.450, Mode=1)
```

Channel should match the number of the position occupied by the LoRa module (as labelled on the LoRa board).

The **frequency** is in MHz and should be selected carefully:

- If you are using RTTY also, it should be at least 25kHz (0.025MHz) away from the RTTY frequency
- It should be different to that used by any other HAB flights that are airborne at the same time and within 400 miles (600km)
- It should be legal in your country (for the UK see https://www.ofcom.org.uk/*data/assets/pdffile/0028/84970/ir*2030-june2014.pdf)

Mode should be either **0** (best if you are not sending image data over LoRa) or **1** (best if you are).

When setting up your receiver, use matching settings.

Primary Functions

These are identical those of the RTTY module.

```
send_packet(packet, callback=None)
```

Sends a binary packet **packet** which should be a **bytes object**. Normally this would be a 256-byte SSDV packet (see the camera.py module).

callback, if used, is called when the packet has been completely set and the RTTY object is ready to accept more data to transmit.

```
send_text(sentence, callback=None)
```

Sends a text string **sentence**. Normally this would be a UKHAS-compatible HAB telemetry sentence but it can be anything. See the telemetry.py module for how to create compliant telemetry sentences.

callback is as for **send_packet()**

```
is_sending()
```

returns **True** if the RTTY module is busy sending data, or **False** if not.

Secondary Functions

These are not needed for most purposes, but may be useful for some users.

```
SetLoRaFrequency(Frequency)
```

Sets the frequency in MHz.

```
SetStandardLoRaParameters(self, Mode)
```

Sets the various LoRa parameters to one of the following standard combinations:

- 0: EXPLICIT*MODE, ERROR*CODING*48, BANDWIDTH20K8, SPREADING*11, LDO On
- 1: IMPLICIT*MODE, ERROR*CODING*45, BANDWIDTH20K8, SPREADING*6, LDO Off
- 2: EXPLICIT*MODE, ERROR*CODING*48, BANDWIDTH62K5, SPREADING*8, LOD Off

these have the following common usages:

- 0: Slow mode for sending telemetry with the best range
- 1: Fast mode for sending images and telemetry at the highest rate
- 2: Intermittent mode for sending brief messages periodically

  SetLoRaParameters(ImplicitOrExplicit, ErrorCoding, Bandwidth, SpreadingFactor, LowDataRateOptimize)

Sets the LoRa parameters to any combination supported by the device. See the LoRa technical manual for details.

Dependencies

- Needs **SPI** to be enabled

Testing

Run the supplied test program:

```
python3 test_lora.py
```