Thomas Kidd and Hunter Green

Overview

To test the LoRa module RN2903's range and data rate we set up our architecture in the following way:

We have two devices, master and slave, where the master sends the different configuration settings to the slave, and the slave logs which configurations have been received. This will help us test the range of the device.

Architecture

In our architecture, our synchronization method is using the python time library where all time is measured from January 1, 1970 and each second has 50 to a 100 tics on UNIX. We believe that this provides enough accuracy for our synchronization technique.

How we want to structure our data

We currently have 1225 different settings we can use on the RN2903. The settings constitute of the following:

- spreading factor (sf): 7, 8, 9, 10, 11, 12
- bandwidth (bw): 125 kHz, 250 kHz, 500 kHz
- coding rate (cr): 4/5, 4/6, 4/7, 4/8
- frequency (freq): 915 MHz
- power (pwr) 2 db, 3 db, 4 db, 5 db, 6 db, 7 db, 8 db, 9 db, 10 db, 11 db, 12 db, 13 db, 14 db, 15 db, 16 db, 17 db, 20 db
- modulation (mod): lora

We know that the receiver (slave) can receive messages with different power readings as long as the spreading factor, modulation, frequency, and bandwidth is the same. Because of this we only need to update the configuration of the receiver 72 different times. We call each set of the 17 configurations a batch. We have 72 batches.

Old method

Our original synchronization method on the master is the following:
- **starting the program on both the master and the slave**: When the second on the clock turns to 0, the master and the slave loads the configuration settings for the first batch of configurations to be sent and received respectively.
- **receiving consecutive batches on both the master and the slave**: to ensure that our program works, we send new batches every 30 seconds. This allows enough time for most of the batches to send, and for the configurations to update on the master and the slave. This is NOT the optimal speed we want to do our testing at, but it allows us to debug any issues we currently have.

Problem

We were running into timing issues between the master and slave. The slave was unable to go back into receive mode fast enough since our code was having trouble reading the serial line from the RN2903 module. We found the issue in our while loop. Our While loop to check the time had no `time.sleep()` in it. And because threads in python are executed in the off time of the main thread (our program) the threads responsible for reading and writing the serial line were unable to return the value quick enough. Since our slave program depends on the `ok` message from the RN2903 after we send it `radio rx 0` to receive consecutive messages, we were unable to quickly receive them.

```
for row_dict in configurations:

        #print(YELLOW, row_dict['sf'], row_dict['cr'],
row_dict['pwr'], RESET)

        row_dict["bw"] = int(row_dict["bw"])

        print(row_dict)

        if send_config(radio_rn2903, row_dict):

            current_time = datetime.now().second

            print(GREEN, "Receiving batch: ",config_counter,
RESET)

            config_counter += 1

            while current_time != 0 and current_time != 30:

                start_time = 60 - current_time

                #print(f"Next config in: {start_time}\r")

                time.sleep(.1)
                current_time = datetime.now().second

                # print(current_time)

            time.sleep(1)
```

As you can see the `time.sleep(.1)` in the while loop, allows the other threads to also run.

The `time.sleep(1)` is for ensuring that the configurations are not skipped over.

Moving on

To speed up the process of sending different configurations we want to implement MQTT to have two way communication between the master and the slave allowing us to sync the two devices easily. Further we also want to make testing faster by only testing certain configurations to figure out quickly which ones fail at certain distances.