PyonAir - pycom Flow Chart

User Button

Initialise an interrupt triggered by the button labelled 'CONFIG' on the expansion board.
A press shorter than a second triggers a soft reset, while holding down the button for 3 seconds enters configuration mode.
Should the soft reset fail, the user will have to press the reset button on the pycom board.

Configuration

Upon pressing the config button for longer than 3 seconds, the LED flashes blue and configuration mode is entered. During configuration mode the device turns into an access point with the name: "*NewPyonAir*", and password: "*newpyonair*", or whatever the device name and password were set previously.
The user has to connect to the device over WiFi and navigate to http://192.168.4.10 in their browser. Once the user has connected and opened the configurations page, the LED flashes green. To configure the device, all fields have to be filled in, and the 'Save' button has to be pressed, when the device saves the new configuration to the SD card, sets time on the RTC module, reboots and continues to run according to the new configurations.
Note, that once configuration mode is entered, the device will reboot, even if the process does not succeed, when red light flashes for 3 seconds before the device proceeds to reboot.

Preferences

There are a range of preferences on the configurations page for the user to choose from.
The device is quite modular, meaning that most of its functions can be turned on and off independently from one-another, and the frequency of data acquisition can also be specified. (Attention should to be paid to the units of time given for each period/interval.)
The user will also have to assign a unique number for each initialised sensor. It is recommended that LoRa is enabled. To register the device on TTN, the device EUI is required, which can also be obtained from the configurations page.

Overwrite Configurations

Read a JSON file from flash if it exists. Overwrite the contents of the config dict with the contents of the JSON file for debug purposes. Developer use only.

Read Config File

Read a file (config.txt) that contains a JSON style dictionary containing all the configurations. The file is converted into a python dict object that is shared globally.
If the file does not exist, it creates an empty one.

Check Configurations

Check if all the keys in the configuration have a value assigned. The config dict is compared to a default one to see if all keys are present. It is also checked if the device ID in the config dict does not match the ID of the device to see if the SD card has recently been moved from one device to another.
Should any of the checks fail, the config dict is reset with the default one, configuration mode is entered, and the user is forced to configure the device.

Initialise File System

Check if all required directories exist on the SD card. If a directory is missing, create it. All files are deleted from the 'Current' and 'Processing' directories (except the LoRa Ring Buffer) to clean up data from the previous boot.

Initialise LoRaWAN

If LoRaWAN is enabled by the user, the device attempts to connect to The Things Network using OTAA.
In the LoRa send method, the most recent message is popped from the stack and sent over LoRa to the given port with the required format.
The stack (LoRa Ring Buffer) stores unsent messages, to make sure everything will be sent, even if there is no connection for a period of time. Messages older than a month are automatically deleted, and at full capacity it is guaranteed that 22 days worth of backed-up data is dealt with, provided that a stable connection returns.

LoRa Downlink Commands

LoRaWAN is capable of receiving downlink messages from the gateway for a brief amount of time after having transmitted an uplink message. The device currently accepts four commands; rebooting the device, triggering an OTA software update, updating the WiFi credentials, and doing all three simultaneously.
It is not recommended that downlink messages require confirmation and it is important to note that commands only get sent after sending an uplink message, therefore it could take up to two intervals for the command to execute.

PM Sensors

Start a thread for each enabled PM sensor, which reads and logs sensor data to the terminal and to a file in the 'Current' directory. The reading is triggered by a periodic interrupt every second. Upon initialising the thread there is a warm-up period specified by the user, during which no readings are logged to make sure the sensors have time to stabilise.
Missed readings are logged, and a red LED blink is scheduled to let the user know something went wrong. If the device keeps blinking red, the sensors are worth checking.
If none of the PM sensors are enabled, a transistor makes sure that nothing gets powered by the PM1 and PM2 Grove sockets.

Indicator LEDs

The RGB LED is used extensively to indicate the current status of the device, and is extremely useful when dealing with errors. (See exception handling in the main.) The user should always go through the initialisation phase upon boot, paying close attention to the LED. If there is a periodic red blink or no green heartbeat every 5 seconds, something has probably gone wrong, and requires immediate attention.

Start Heartbeat

Pycom's built-in heartbeat has to be turned off and back on again every time the LEDs are used, and it often causes an error, which forces the device to reboot itself. Therefore, the heartbeat is simulated using a green LED blink every 5 seconds triggered by a periodic timer interrupt from the main instead of the built-in pycom method.

Update Time via GPS

If getting current time in Stage 1 did not involve using the GPS, then start a thread to update time from the GPS. This is necessary, because the time acquired by the GPS module is more accurate than the time set by configuring the device over WiFi.

Main

Stage 1

Initialisation of critical features

Try: Amber light flashes

- Status Logger
- User Button
- Mount SD Card

Except: Red light blinks

Execution halted, probably an SD Card failure.
Device automatically reboots after 3 minutes.
Reboot can be triggered with the user/reboot buttons.

Stage 2

System management and UTC acquisition

Try: Amber light flashes

- Read Config File
- Set Version Numbers
- Overwrite Configurations (debug)
- Get Current Time
- Check Configurations
- Check for Update

Except: Yellow light blinks

Execution halted, probably failed to acquire UTC.
Device automatically reboots after 3 minutes.
Reboot can be triggered with the user/reboot buttons.
Blocking configuration mode can be entered by pressing the user button for 3 seconds - indicated by blue LED flashing.

Stage 3

Initialisation of LoRaWAN, Sensors and the Event Scheduler

Try: Amber light flashes

- Initialise File System
- Get Dict of Sensors
- Initialise LoRaWAN
- Initialise Temperature Sensor
- Initialise PM Sensors
- Start Event Scheduler
- Start Heartbeat
- Update Current Time via GPS

Successful Initialisation
Green light blinks 3x

Except: Red light flashes

Initialisations of further threads are halted.
Already initialised threads and timed interrupts continue to run.
Device does not reboot automatically.
Reboot can be triggered with the user/reboot buttons.
Parallel Configuration mode can be entered by pressing the user button for 3 seconds - indicated by blue LED flashing.

Status Logger

The Status Logger is the main logger used to log all exceptions and information to the SD card (status_logger.txt), and to the terminal as well.
Levels: Critical, Error, Warning, Info and Debug

SD Card

An 16GB or larger SD Card is used to log all data, errors and configurations.
A 16GB card should last about 5 years.

Version Numbers

There are two types of Version Numbers that are set by the developer here.
The first one is the format version, which is used to tell the decoder at the back-end what format scheme the device uses to encode the LoRa messages.
The second one is the code version, which is the git tag corresponding to current commit. The code version is important to compare differences upon the event of an OTA update.

Modular Code

The code was designed modular to encourage people to fork and contribute to the project. To add support for a new sensor look for the following parts of the code to change: main.py/Stage 3/call initialisation, intialisation.py, strings.py, config_page.py, and lastly add the thread that uses your sensor's library.

Get Current Time

Get current time is a critical part of the initialisation process. Try to read current UTC time from the external RTC module and update time on the pycom board.
If the RTC module is not connected, or it has not yet been configured, then try to get current time from the GPS module and update both the external RTC and the RTC on the pycom board. A blue light blinks while the GPS operates.
If none of these succeed, an exception is raised (see Except of Stage 2). At this point the user is given a table of possible issues and solutions by the logger on how to get UTC to continue the execution

Check for Update

Check if an update has recently been triggered on the device. If there is an update scheduled, the current code version is compared to the one on a server, yellow light flashes, and the software is updated over WiFi. The device then proceeds to reboot.

Get Sensors

Get a dictionary of sensors from the configuration. This is a dictionary of sensors, which are selected by the user.

Temperature & Humidity Sensor

If the temperature & humidity sensor is enabled by the user, a class object of the chosen sensor is initialised (default to SHT35). A periodic interrupt is triggered from the main thread, which reads the current temperature and humidity, and logs it to the terminal and to a file in the 'Current' directory. The period is set by the user, and it is recommended to be more than 3 minutes to save processing power and mitigate heating effects. Missed readings are logged, and a red LED blink is scheduled to let the user know something went wrong. If the device keeps blinking, the sensors are worth checking.

Calculating Averages

Calculating averages happens at the end of each interval. It takes the raw data files of enabled sensors (PM1, PM2, TEMP) from the 'Current' directory, and moves it to the 'Processing' directory. It then takes the average of requested columns and constructs a LoRa message with a given format based on the type of sensors that are enabled. The LoRa message is pushed to the LoRa Ring Buffer, and a copy is appended to a file separated by months in the 'Averages' folder in the 'Archive' directory. The raw data is then moved from the 'Processing' directory to the 'Archive'.

Event Scheduler

The Event Scheduler is responsible for triggering periodic events such as calculating sensor averages, getting position over the GPS, and sending data over LoRaWAN. Upon initialisation it calculates when the first event should occur, so the interval (s) is divisible by the number of seconds in the day that will have passed by the first event. The first event then starts a periodic interrupt with the given interval to set up the event queue.

Get Position via GPS

Upon a GPS event blue light blinks, a transistor is turned on to power the GPS module and a serial bus is initialised to communicate with the sensor. Since the maximum number of UART buses are 3 on the pycom board, and each PM sensor uses one, the terminal output is disabled while the GPS operates. The GPS continuously receives and parses sentences from the satellites until the desired quality is reached. The desired quality depends on the percentage of time elapsed from the timeout. When latitude, longitude and altitude are updated, a LoRa message is constructed with the given format, and pushed to the LoRa Ring Buffer. The message is also logged to 'Archive'.
The device also recognises if the GPS is enabled in configurations, but the module itself is not connected. In that case the thread times out in 10 seconds instead of the timeout set by the user.

Schedule LoRa Messages

The Event Scheduler is also responsible to limit airtime over LoRa. Currently, the Fair Access Policy on TTN limits the airtime to 30s per day. The event scheduler keeps track of how many messages it has sent on the day, and schedules 1-4 randomly timed messages during an interval. The number of messages depends on the size of the Lora Stack, the number of messages sent on the day, and also on the length of the interval.
Real-time transmission is guaranteed if the interval is less than 4 minutes when the stack is empty, and less than 15 minutes when the stack is full. The absolute minimum interval is 1 minute, provided that it takes 30 seconds for the LoRa thread to finish, and there needs to be time for at least two, to send the message of the current interval, and potentially a message sitting deeper in the stack.