PowerSense

Working Environment

Operating System: `Ubuntu 18.04`

OS: `Mongoose OS(mos)`

Debugging Tools: `Saleae Logic Analyser`

Hardware:

```
ESP32
ATM90E26
```

How to build

Setup Mongoose OS

The Mongoose OS setup Instructions are available in quick start guide.

1. Setup Mongoose OS according to the Host operating system.
2. The application was originally built using mos version `2.14.0`. After mos is installed, switch to this version.

```
mos update 2.14.0
```

3. Doing Local build on Linux requires docker to be installed. Please refer to the installation Instructions on docker website.

Building Application:

To build the application, clone the repo and run this terminal command inside `repo/firmware/`:

For local build:

```
$ sudo mos build --local --platform esp32
```

This will fetch the mos sources and build the application. First build will take time as it fetches all the required libraries.

The final application will be available inside build folder named `fw.zip`.

For online build:

```
$ sudo mos build
```

Program Flow

Overview:

The whole application is event driven.

At bootup, two timers are started. One ticking every second and the other every 15s.

- The timer ticking every 1 second, checks and maintains the network connectivity.
- Timer ticking every 15 second, pushes the data to the cloud.

Steps:

Step 1: Connect to WiFi.

For this, the esp32 starts and AP (`ssid=powersense, password=password`). Connect to the AP and Sign In to the network. Give the WIFI credentials and done.

Step 2: Connect to MQTT.

After WiFi is successfully connected, the device connects to the `nodesense.baseapp.com` through MQTT.

Step 3: Poll ATM90E26 every 15s via SPI and get the data. The readings are pushed to `nodesense.baseapp.com`.

Production

Overview:

1. Each device has a unique ID (12 characters) which is made of two parts, unique password (8 characters) and unique HID (4 characters).
2. The HID is used to get UID from the cloud.
3. The UID is then used for MQTT connection.
4. When flashing the device in production, use `build_script.py` inside `repo/firmware/esp32/`. This scripts picks up a unique ID from `sheet.csv` and flashes the device with it.

```
$ python3 build_script.py <startingID>
```

For example:

```
$ python3 build_script.py 1000
```