**Air Quality Assessment**

**Haris Drapic - hd222dw**

The main goal of my project is building a system which calculates air quality via measured humidity and gas ( more exact [volatile organic compounds](https://www.health.state.mn.us/communities/environment/air/toxins/voc.htm)) in an indoor environment.  
An IoT device will collect gas, humidity from attached sensor(BME680) and transmit collected and calculated values (gas, humidity, calculated air quality) to a cloud platform for storage and presentation.

*Estimated time to complete: ~12h*

**Objective**

The project will aim to create system for measuring air quality at home (or other indoor/semi-indoor environment). Gas([volatile organic compounds](https://www.health.state.mn.us/communities/environment/air/toxins/voc.htm)), humidity and calculated humidity will be measured and sent to a cloud platform for preservation and presentation.

Man worries about air quality in man´s home all the time, especially it is important for some kind of room (garage, basement etc.)   
  
After measured quality is decreased below some threshold than some actuator(some air conditioner or something similar) will turns on. The actuator in my project contains of buzzer(piezo) and LED . If air quality drops below certain value than buzzer will play some song (guess which one?-video on [github](https://github.com/hd1966/desktop-tutorial/tree/main/airquality/videos)) and LED will turns on.

Further on, it will be given an explanation how to build my project and achieve my objective.

**List of material**

**All items used**

| **Image** | **Item** | **What is it for?** |
| --- | --- | --- |
| A picture containing text, electronics  Description automatically generated | FiPy with headers | Micropython programmable microcontroller – development board. In my project it is used to read data from sensor and transmit data over WIFI to two different cloud platforms – adafruit and ubidots. Supports for one certain type of long range communication for different pycom development boards are shown in table 1 . |
| A picture containing text, electronics, circuit  Description automatically generated | Pysense | Pysense is the Expansion Board, intended for pycom devices equipped with different sensors such as temperature sensor , humidity sensor, light sensor ... Read more about pysense [here](https://docs.rs-online.com/59b6/0900766b815d0a8c.pdf) |
| En bild som visar bord  Automatiskt genererad beskrivning | 2xBreadboard | Connection of sensors and the other electronic equipment (led, buzzer, fipy) without need of soldering. |
| Jumper Wires Standard 7" M/M - 30 AWG (30 Pack) | 15 x Jumper Wires male/male | To connect from the breadboard to the pins on the electronic equipment |
| A cell phone on a table  Description automatically generated with medium confidence | BME680 | Temperature , humidity, pressure and gas sensor. Project use gas and humidity sensor for calculation of air quality. BME680 use either [SPI](https://www.circuitbasics.com/basics-of-the-spi-communication-protocol/) (Serial peripheral interface)or [I2C](https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/)(inter-integrated cicuit) communication. In this project I2C communication is used. |
| A picture containing sky, close  Description automatically generated | Buzzer(piezo) | Play some song if air quality is below some threshold |
| A picture containing tableware, kabob, broom  Description automatically generated | 220Ω resistor | We need to limit current on the LED |
|  | LED | To indicate alarm(air quality is below some threshold) |
| A picture containing floor  Description automatically generated | Micro USB 2.0 Cable | Communicating with the device and uploading of code from computer |

**Table 1**

A picture containing chart

Description automatically generated

Table from [development boards](https://static6.arrow.com/aropdfconversion/2d1dfa38786d890f030a8211221075dfe5db664d/pycomwipy3.0userguide.pdf)

**Where to buy them**

Electrokit has a bundle that includes almost everything in list of material except the **BME 680 sensor ( Conrad – only BME680 pre-soldered which I found)** and **buzzer (already had)**.

| **Part** | **Price** | **Link** |
| --- | --- | --- |
| Fipy and sensors bundle | SEK 1499 | [Order at Electrokit](https://www.electrokit.com/produkt/lnu-1dt305-tillampad-iot-fipy-and-sensors-bundle/) |
| BME680 AirQuality sensor | SEK 229 | [Order at Conrad](https://www.conrad.se/p/sensormodul-sen-bme680-1884870) |
|  |  |  |
| Buzer | - | Already had, otherwise couple of SEK |
| **Total SEK** | **1728** |  |

**Computer Setup**

**Install Atom**

I’ve chosen to use **Atom** for this project. Atom is an open source and free IDE. I already have Visual studio code installed at home , but decided to use Atom .   
The first step is to install the IDE itself. Go to [**Atom download page**](https://atom.io/) and download/install Atom for your operating system.

**Install Node.js**

Because PyMakr package on Atom depends on Node.js, you have to install Node.js before you can install package PyMakr.  
Go to [**install node**](https://nodejs.org/en/) and download/install Node.js for your platform.

**PyMakr plugin**

For MicroPython support, we need to install PyMakr.

Graphical user interface, text, application, email

Description automatically generated

Start Atom and open the package manager - File -> Settings -> Install and enter **Pymakr** in the search input field (image above).  
If you connect pycom device with usb cable to your computer you can upload and synchronize code written in the editor to the microcontroller. It is performed by pressing the **Upload** button (see image below).

Graphical user interface, text, application, email

Description automatically generated

**Unpacking the hardware**

Pysense and the FiPy comes in two separates packages.

**Flashing the firmware on the Microcontroller**

Connect pysense to fipy according to image below. It must be done because fipy and pysense must communicate to each other via [UART](https://www.circuitbasics.com/basics-uart-communication/) (universal asynchronous receiver/transmitter).

Diagram

Description automatically generated with medium confidence

Connect Rx and Tx of pysense to Rx and Tx of fipy .Vin and Gnd of pysense is connected to vin and gnd of fipy (see image above).

Connect computer with pysense using the USB cable. **If you want to disconnect usb cable from pysense be extremely cautious, because Usb port on pysense is extremely fragile. It is better to disconnect cable on other end (computer)!!!**

Update the firmware on the fipy. Pycom has a tool and you can download and install that tool for your platform [here](https://docs.pycom.io/updatefirmware/device/) . Follow instructions on same webpage. **Be careful - check nvs partition and config partition(advanced settings), otherwise it can happen that Atom never come to REPL(Read Evaluate Print Line) - it is happened to me!!**

**Connect everything**

It is important, that before you start to connect electronic component, **make sure that power source isn’t attached to the pysense. Disconnect the USB-cable (computer end)!**

**The breadboard**

Main purpose for the breadboard is to test electronic circuits and to experiment with electronic circuits. The breadboard is suitable for these purposes (testing and experimenting with electronic circuits) because breadboard doesn´t require soldering, you only must plug electronic components to holes on the breadboard.

The two bottom and two top horizontal rows of holes(connection points) on the breadboard are so called power rails, because you connect power (gnd and vin see picture below) to these holes. Table

Description automatically generatedThese rows are interconnected horizontally and that connection creates rows of connection points with same electric potential – equipotential points. The connection points (holes) in the middle of the breadboard , placed between power rails, are connected vertically and that creates columns of equipotential connection points. (see image below)

Graphical user interface, application, table

Description automatically generated

According to conventions red color is reserved for Vin (+) and black color for GND(-). I use that also here(image above).

For wiring, it is crucial to understand the connection layout of the breadboard (image above)

**Complete circuit diagram**

A picture containing text, electronics

Description automatically generated

According to conventions, red color is reserved for Vin (+) and black color for GND(-). I use that also here(image above). For the other wires, different from Vin and gnd, I used yellow or orange color on the image above. I used also same colors for jumper wires. You can use colors of your choice, but it is suitable to use those conventions.

At first, connect pysense and fipy as it is described before.

**Attaching the sensor BME680**

The BME680 with its MOX (Metal-oxide) sensor detects VOCs(benzene, ethylene glycol, formaldehyde, methylene chloride, tetrachloroethylene, toluene, xylene, and 1,3-butadiene) in the air. This sensor gives you information of presence of **VOCs** in the surrounding air and this information **is not specific** for a specific gas.

The first step is connect the **BME sensor** which has six pins.  
A picture containing wooden, wood

Description automatically generated

VCC (Voltage at the Common Collector, ‘+’) ,GND,SCL,SDA,SDO(used only for SPI) and CS(chip select) - these pins are marked and you can see that on picture above.

This sensor is enabled for spi or i2c communication. For I2C use following connection:

* Using jumper wire, connect a GND pin on the sensor with gnd rail.
* Using jumper wire, connect a VCC pin on the sensor with vin rail.
* Using jumper wire, connect a SCL pin on the sensor wire with P10 (SCL) on fipy.
* Using jumper wire, connect a SDA pin on the sensor wire with P9 (SDA) on fipy.
* Using jumper wire, connect a CS pin on the sensor wire with P11 on fipy.

I use I2C, but it is possible to use SPI , then SDO must be connected to P8 on fipy.

**Attaching buzzer and LED**

A picture containing text, gear

Description automatically generated

Buzzers generate sounds. In these projects buzzer playing some melody.Read more about [buzzers](https://www.seeedstudio.com/blog/2020/12/22/introduction-to-buzzers-piezo-and-magnetic-buzzers/). Using jumper wire, connect one pin of buzzer to GND rail and the other pin of buzzer to P23 on fipy.

Using jumper wire ,connect LED’s cathode (shorter leg) via resistor(limiting current) to gnd rail. Using jumper wire ,connect LED’s anode (longer leg) to P22 of fipy.

Before doing anything more, please ensure that all the wiring you’ve done so far is identical to the complete circuit diagram given before.Connect a jumper wire from the **Vin** pin on the Pycom device to the **one** of the two power rails and from the **GND** pin on the Pycom device to the **other** of the two power rails.

**Platform**

I’ve chosen two platforms to use for this project:

* [Ubidots](https://ubidots.com/)
* [Adafruit](https://io.adafruit.com/)

Ubidots and Adafruit are a cloud based platforms. On these platforms you can send, store and present data from your sensors on dashboards.

With ubidots, you can send sensor’s data using [HTTP requests](https://www.tutorialspoint.com/http/http_requests.htm)(POST,HEAD,…) over WIFI .

With adafruit, you can send sensor’s data using [MQTT](http://www.steves-internet-guide.com/mqtt/) .

Ubidots has a STEM edition . This edition is free of charge in case your project is for educational use.

Adafruit also have [free edition](https://io.adafruit.com/) (at the bottom of webpage).

These free editions are enough for this project in a development setting i.e. for educational purpose.

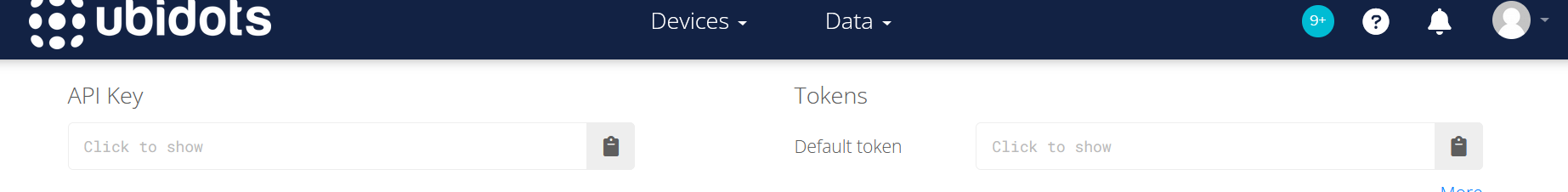
Comparison between those two platforms is shown in table below:

|  |  |
| --- | --- |
| Ubidots(stem) | Adafruit (free) |
|  Devices: **3** devices free   Variables: Up to **10** **variables** per device.   Data Rate: **1 request per second**, across all of your devices.   Data Retention: **1 month.**   Dashboards: Up to **3** dashboards, with up to **10** widgets each. | * **30** data points per minute * **30** days of data storage * Actions every **15 minutes** * **5** dashboard limit * **10** feed limit |

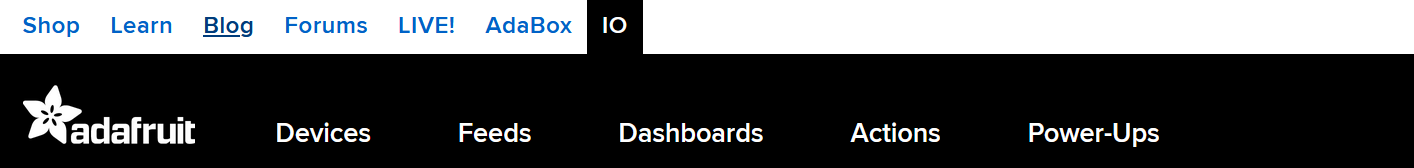
Go to Ubidots and sign up for a STEM account [here](https://industrial.ubidots.com/accounts/signup_industrial/) and go to [adafruit](https://io.adafruit.com/) sign up for a free account (sign up now at the bottom of the webpage)

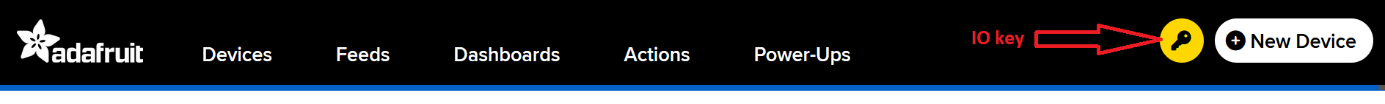
You are going to need your **access token(ubidots)** in order to identify your account on ubidots when you are sending data via http requests to ubidots. If you want to obtain your token , [sign in](https://stem.ubidots.com/accounts/signin/) to your account, press arrow(see picture below) and then choose username API credentials. Click to show , in textbox beside default token, to see access token .

Graphical user interface, application

Description automatically generated 

You are going to need your **IO key** in order to identify your account on adafruit when you are sending data mqtt to adafruit. If you want to obtain your IO key , [sign in](https://accounts.adafruit.com/users/sign_in) to your account, press IO and then press IO key(picture below).





**The code**

BME680

I used the bme680.py from [Robert Hammelrath](https://github.com/robert-hh/BME680-Micropython) . After that I created bme.py with same class. In BME class constructor, an object of class BME680\_I2C (if you want to use I2C) or BME680\_SPI(if you want to use SPI) is created . Commented out rows are for SPI.

*def \_\_init\_\_(self, calibration\_time=500):*

*self.sensor = BME680\_I2C(I2C())*

*#cs = Pin("P11", Pin.OUT, value=1) - for SPI*

*#spi = SPI(0, mode=SPI.MASTER, baudrate=400000, pins=("P10", "P9", "P8"))*

*#self.sensor = BME680\_SPI(spi, cs)*

*self.vars = []*

*#register variables (voc, airquality and humidity)*

*self.humidity\_var = self.\_register\_var(constants.BME\_HUMIDITY\_VAR\_NAME)*

*self.gas\_var = self.\_register\_var(constants.BME\_GAS\_VAR\_NAME)*

*self.airquality\_var = self.\_register\_var(constants.BME\_AIRQUALITY\_VAR\_NAME)*

*# calibration*

*self.\_warm\_up(calibration\_time)*

This class also read values from BME680 and calculate air quality with weighting 25% humidity and 75% gas(voc) . calculation distance measured humidity and ideal humidity (40 = indoor humidity).

To calculate air quality , calculate first distance between measured humidity and hum\_baseline(40.0 - ideal indoor humidity) and distance between measured gas(voc) and gas\_baseline (average value of voc, calculated in calibration step), then use weighting and calculate air quality

*Platform* is abstract, base class that allows for setup and transmission of variables to a given platform. Sensor (BME680) create dictionary with variable’s name (gas,humidity,airquality) as keys in that dictionary.  
Platforms(platformubidots and platfformadafruit) receives that dictionary of variables and their task is to transmit those to a destination (adafruit or ubidots cloud platform).

The Ubidots platform using POST request to transmit data and requires urequests.py.  
It can be downloaded from Jose Reyes Garcia Delgado’s Github [here](https://github.com/jotathebest/micropython-lib/blob/master/urequests/urequests.py). For sending data using WiFi over HTTP to ubidots platform [following tutorial](https://help.ubidots.com/en/articles/961994-connect-any-pycom-board-to-ubidots-using-wi-fi-over-http) is used.

For sending data to Adafruit MQTT(feeds) is used. Because of that umqtt.py requires.

For connection with wifi används airqualitywifi.py.

One class for buzzer is created. If air quality is decreased below defined threshold then some melody is played with the buzzer.

All constants (variable’s names, credentials for wifi, info needed for connection to ubidots, adafruit ) are placed here. Couple constants are listed below:

AUTH\_WIFI\_SSID = '<YOUR\_WIFI\_SSID>'

AUTH\_WIFI\_PWD = '<YOUR\_WIFI\_PASSWORD>'

AUTH\_UBIDOTS\_ACCESS\_TOKEN = '<YOUR\_UBIDOTS\_ACCESS\_TOKEN>'

AUTH\_DEVICE\_NAME = "<YOUR\_DEVICE\_NAME>" # device name in ubidots account

…

All code is available at [here.](https://github.com/hd1966/desktop-tutorial) Feel free to use however you like!

**Create new project and lib folder**

The first step is to create a new project in Atom(File -> Add project folder). I created folder airquality and whole project will be placed in that folder. After that , right clicked on project name (airquality) and create new folder(New folder – Shift +a) lib(see picture below).

Graphical user interface, application

Description automatically generated

Place bme.py, bme680.py , airqualityplatform.py, urequests.py,airqualitywifi.py, umqtt.py, constants.py in lib folder.

.

**The application itself**

Finally, we will develop the application itself. The application will consist of two files.

* main.py
* boot.py

Create these two files in the root folder of your project (one level above the *lib* folder).

Chart

Description automatically generated with low confidence

**boot.py**

boot.py is the first file executed on device in your application.

*from lib.senap\_wifi import wifi\_enable*

*import pycom*

*import config*

*pycom.heartbeat(False)*

*pycom.rgbled(config.CFG\_LED\_BOOT)*

*wifi\_enable(config.CFG\_WIFI\_CONNECT\_TIMEOUT)*

Turn off the device’s heartbeat , enable WIFI and define color on rgbled(default green in constants).

**Main.py**

main.py - instantiates bme680,buzzer ,led , PlatformUbidots and PlatformAdafruit.

*from lib.airqualityplatforms import PlatformUbidots*

*from lib.airqualityplatforms import PlatformAdafruit*

*import lib.bme*

*from lib.airqualitywifi import wifi\_is\_connected*

*import pycom*

*import machine*

*import time*

*import lib.constants as constants*

*from machine import I2C, Pin,SPI*

*from lib.bme import BME*

*from lib.umqtt import MQTTClient # For use of MQTT protocol to talk to Adafruit IO*

*from lib.buzzer import Buzzer*

*#led*

*led=Pin("P22", Pin.OUT,pull = Pin.PULL\_DOWN)*

*#buzzer*

*buzz=Buzzer(Pin("P23"))*

*cs=Pin("P11", Pin.OUT, value=1)*

*bme = BME(calibration\_time=5)*

*# Dictionary of sensor data*

*sensor\_data = {}*

*# platforms in this case ubidots and adafruit*

*all\_platforms = []*

*# List of all sensor's variables*

*sensor\_var\_list = bme.get\_vars()*

*# Add ubidots platform*

*all\_platforms.append(PlatformUbidots())*

*adf=PlatformAdafruit()*

*# Add adafruiplatform platform*

*all\_platforms.append(adf)*

*# Allow all platforms to prepare, pass the variable list in case any platform needs it*

*for platform in all\_platforms:*

*platform.init\_with\_context(sensor\_var\_list)*

*# Main loop*

*while True:*

*led.value(0)*

*# Reset device in case we lost WIFI*

*if not wifi\_is\_connected():*

*machine.reset()*

*adf.getClient().check\_msg()# Action a message if one is received. Non-blocking.*

*# New dictionary each cycle, populated by sensors*

*sensor\_data = {}*

*read\_result=bme.read(sensor\_data)*

*if sensor\_data[constants.BME\_AIRQUALITY\_VAR\_NAME] <95.0:*

*buzz.play\_bls()#play song*

*led.value(1)*

*# Iterate all registered platforms and transmit data from sensors*

*for platform in all\_platforms:*

*platform.transmit(sensor\_data)*

*# Sleep until next cycle*

time.sleep(constants.CFG\_SENSOR\_TRANSMISSION\_INTERVAL)

The application fills a list of sensors and a list of platforms. For each transmission, the sensors are read and passed to all platforms (adafruit and ubidots in my case).  
If wifi is lost, the device is reset in order to invoke the connection process again (in boot.py)

**Transmitting data/connectivity**

**Upload your project**

Connect the device to your computer, wait until its successfully connected and then press the **Upload** button in Atom to upload the project to your device.

As I said I use WiFi in my my project. As I also said I use MQTT to transport data to Adafruit and HTTP request method (POST) to transport data to ubidots.

If everything is working correctly, you should see something similar to this in your terminal.

Graphical user interface, text, application, email

Description automatically generated

Transmission interval which decide how often sensor data is sent, is defined with constant CFG\_SENSOR\_TRANSMISSION\_INTERVAL . This constant is in constants.py file. I tried with several values - 30 seconds, 1 minute, 10 seconds etc.

**Presenting the data**

When logged in to your account in Ubidots, you select need to create new Device. Press on + on image below.Graphical user interface, text, application, website

Description automatically generated Inside of device create several variables . Press on + sign.

A picture containing timeline

Description automatically generated

After that, you add widgets to your dashboard and bind them to the variables that you have sent to the platform.

Graphical user interface, application

Description automatically generated

Below, I’ve chosen to present humidity ,airquality,gas(voc) as line charts

.

Chart, line chart

Description automatically generatedChart, line chart

Description automatically generatedChart, line chart

Description automatically generated

It is almost equal with adafruit but you need to create feeds instead of variables. Then you can create dashboard, add block(same as widget in ubidots) to dashboard and bind them to the feeds.

Graphical user interface

Description automatically generated

Graphical user interface, application

Description automatically generated

Below, I’ve chosen to present humidity ,airquality,gas(voc) as line charts

**Graphical user interface

Description automatically generated**

**Graphical user interface

Description automatically generated**

**Chart

Description automatically generated**

**Test the project**

I test project in two different ways.

I touched BME680 with fingers (see video on my [github](https://github.com/hd1966/desktop-tutorial)) . As a result of this touch measured humidity is increased and air quality is decreased. After some time , humidity take normal value again and aur quality too. You can see that on line chart below.

Chart

Description automatically generated

I used the lighter in the vicinity of bme680. As a result of that , gas\_voc is slightly increased and air quality slightly decreased.

**Finalizing the design**

A picture containing text, electronics

Description automatically generated

The project was very interesting and I learned me a lot on the way!  
As I already mentioned(read flashing firmware to microcontroller!), I had some minor problem in project implementation . I also needed certain amount of time to find on the internet, version of bme680 which works without error(I2Bus Error). Except these 2 short test which I´ve done, this project also offers many opportunities to test(test project in different environments = garage, basement etc.). It is also possible to extend this project in the future. New platforms can be added, for example datacake is same as adafruit – only constants are different(token , topics etc). You can add following in constants.py

# Datacake credentials

SERIAL\_NUMBER = '…'

MQTT\_BROKER = 'mqtt.datacake.co'

TOKEN = '…'

PORT = 1883

TOPIC1 = '…'

TOPIC2 = '…'

TOPIC3 = '…'

Then use TOPIC1 or TOPIC2 or TOPIC3 instead of feeds here

self.client.publish(topic=constants.**AIO\_FEEDS['humidity']**, msg=str(humidity)) # Send to Adafruit IO

The rest of the code is same.

New type of network (lorawan) can be added.