IoT Air Quality System

Uli Raich (uli.raich@gmail.com)

Two lectures on Hardware and Software for
the Internet of Things applied to Air Quality measurement

Lecture 1: Introduction to the hardware
and development environment and access to the “things”

Presented online at the Seminar – Air Quality and
IoT based Air Sensors Nov. 2023

Resources

The slides and the code of all demo programs are available for download at https://github.com/uraich/IoT4AQ

What is IoT?

Wikipedia:

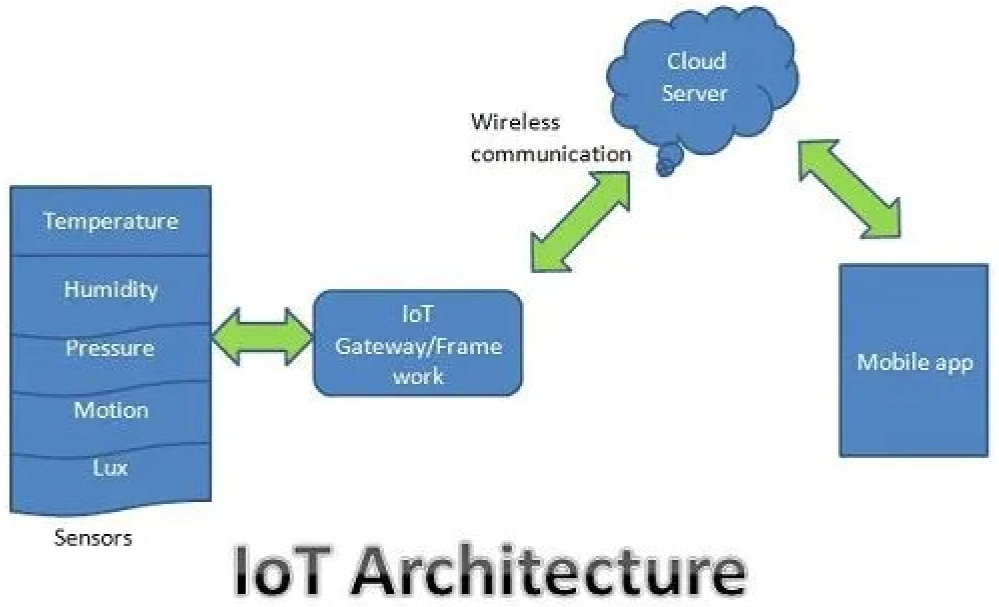
The **Internet of Things (IoT)** describes physical objects (or groups of such objects) with sensors, processing ability, software and other technologies that connect and exchange data with other devices and systems over the internet or any other communication networks

IoT

The name “Internet of Things” already suggests that there are two distinct components to it:

- Things, which are represented by sensors and actuators. We therefore must have a means to read out these sensors and to control the actuators. Hardware interfaces are needed for this.
- Internet, we must be able to send data read from the sensors to the Internet and receive commands for the actuators from it. This means that we need a network interwork and the software to run the network protocols.

IoT system architecture



What is an IoT system composed of?

We need:

- Sensors and actuators
- A processor powerful enough to run the network protocol layers
- A network interface: either Ethernet of WiFi
- Interfaces to the sensors and actuators*:*
  - General Purpose I/O (GPIO) lines
  - I2C, I2S, SPI, serial interfaces
- The cost for the controller should be in a good relation with the cost of *the things*

which micro-contoller?

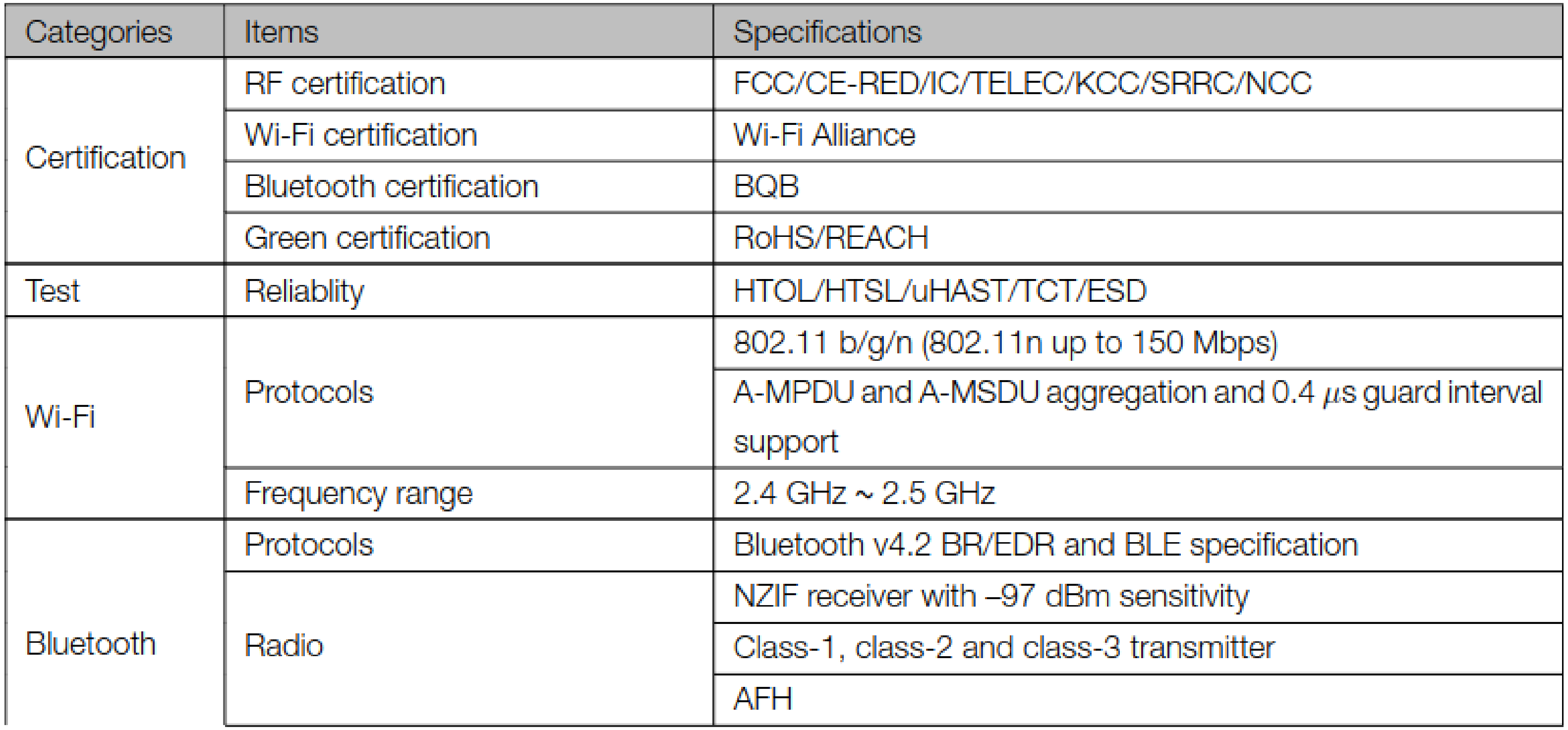
Question of

- Needs (how much memory, which speed, which interfaces …)
- Budget
- Software environment and programming language
  - Most common: Arduino SDK
  - MicroPython
- Taste

The micro-controller

Categories	Items	Specifications
	Audio	CVSD and SBC
Hardware	Module interfaces	SD card, UART, SPI, SDIO, I^2C, LED PWM, Motor PWM, I^2S, IR, pulse counter, GPIO, capacitive touch sensor, ADC, DAC
	On-chip sensor	Hall sensor
	Integrated crystal	40 MHz crystal
	Integrated SPI flash	4 MB
	Operating voltage/Power supply	2.7 V ~ 3.6 V
	Operating current	Average: 80 mA
	Minimum current delivered by power supply	500 mA
	Recommended operating temperature range	–40 °C ~ +85 °C
	Package size	(18.00±0.10) mm x (25.50±0.10) mm x (3.10±0.10) mm

ESP32 network connection

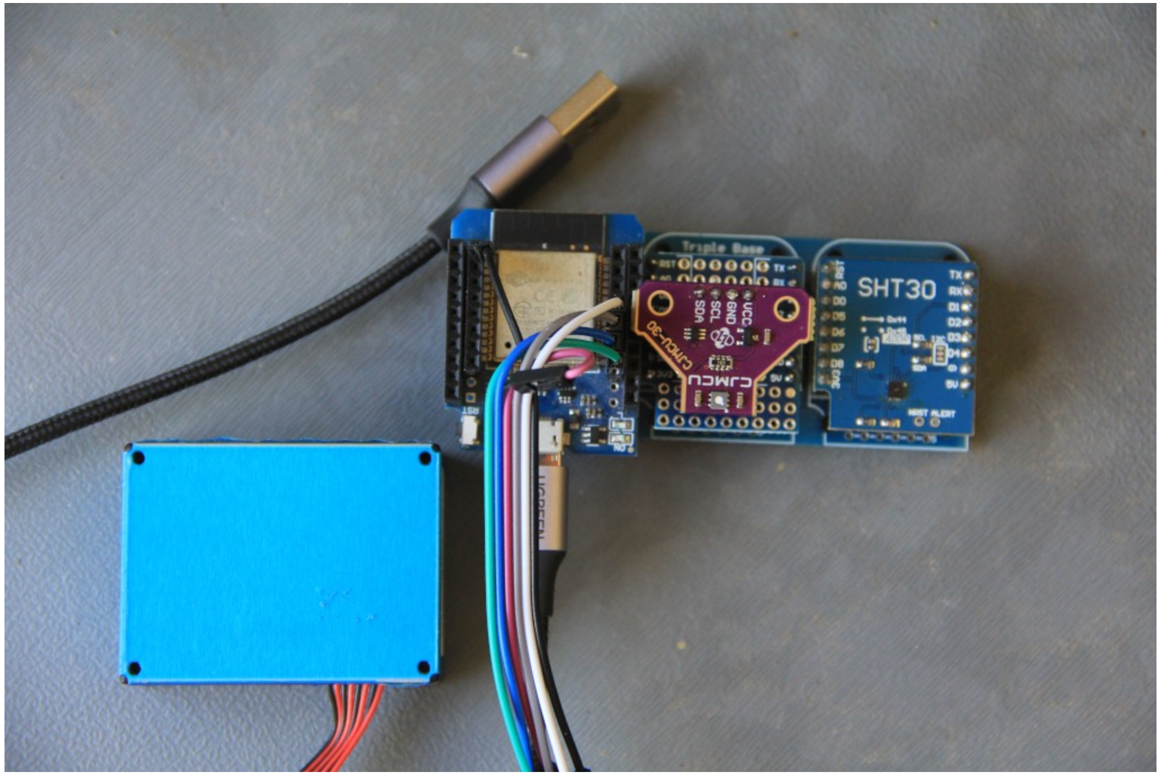


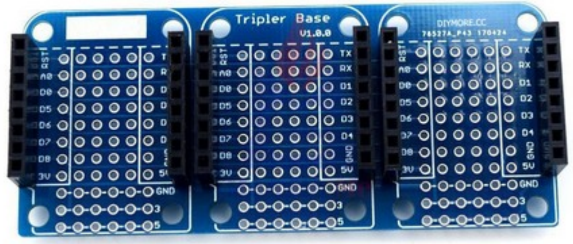
Categories	Items	Specifications
Certification	RF certification	FCC/CE-RED/IC/TELEC/KCC/SRRC/NCC
	Wi-Fi certification	Wi-Fi Alliance
	Bluetooth certification	BQB
	Green certification	RoHS/REACH
Test	Reliablity	HTOL/HTSL/uHAST/TCT/ESD
Wi-Fi	Protocols	802.11 b/g/n (802.11n up to 150 Mbps)
		A-MPDU and A-MSDU aggregation and 0.4 μs guard interval support
	Frequency range	2.4 GHz ~ 2.5 GHz
Bluetooth	Protocols	Bluetooth v4.2 BR/EDR and BLE specification
	Radio	NZIF receiver with –97 dBm sensitivity
		Class-1, class-2 and class-3 transmitter
		AFH

The system with AQ sensors

From left to right:

- The ESP32 CPU card with serial connections to the PlanTower PMS5003 dust sensor
- The SGP30 air quality sensor
- The SHT30 temperature and humidity sensor





Programming the micro-controller

Depending on the processor used, several programming environments are available. For the ESP32 you have:

- esp-idf (ESP integrated development framework) provided by Espressif (not recommended for a beginner programmer)
- The Arduino SDK (Software Development Kit) originally developed for the Atmel Atmega chips but today available for many micro-controllers including the ESP32.
- MicroPython. A stripped down Python-3 interpreter with additional features needed by micro-controllers available for several chips including the ESP32.

In these lectures I use MicroPython

Communication tools

Serial communication (the PC :

- minicom

Interaction with the file system:

- ampy

IDE:

- thonny

Flash Programming:

- esptool

Several other tools are available, I listed only those that will be demonstrated during the presentation

How to interact with MicroPython

The MicroPython interpreter is installed on the ESP32 flash.

The Interpreter itself is written in C and available in OpenSource from github.

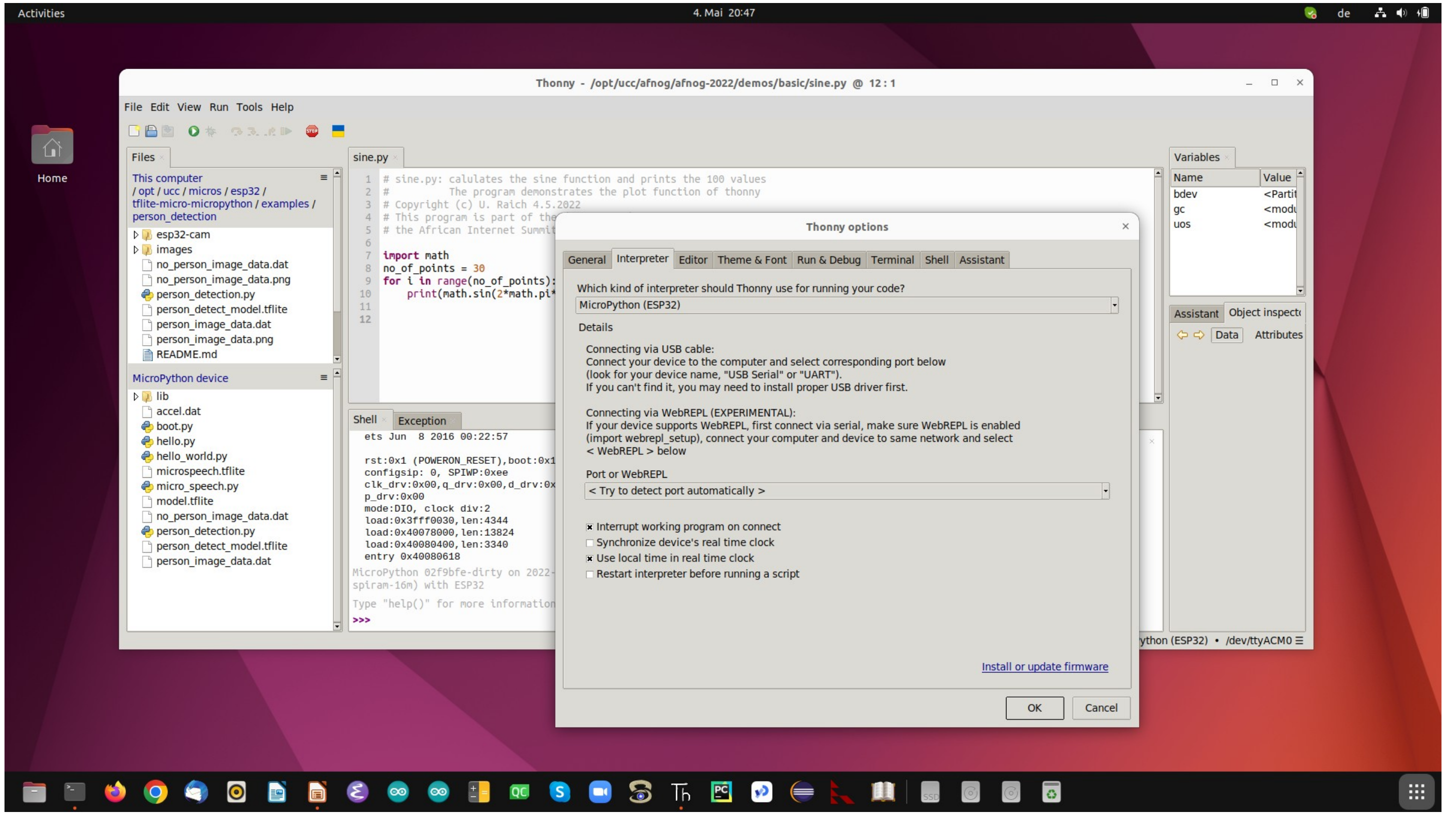
You can build MicroPython from source yourself and install it in the ESP32 flash using esptool.

You will need some infrastructure to accomplish this:

- xtensa-esp32-gcc cross compilation chain
- esp-idf
- cmake
- a dedicated Python virtual environment
- esptool to erase and program the flash memory

You can also pick up a firmware binary and flash it.

thonny



```
# sine.py: calulates the sine function and prints the 100 values
#          The program demonstrates the plot function of thonny
# Copyright (c) U. Raich 4.5.2022
# This program is part of the
# the African Internet Summit

import math
no_of_points = 30
for i in range(no_of_points):
    print(math.sin(2*math.pi*
```

```
ets Jun  8 2016 00:22:57

rst:0x1 (POWERON_RESET),boot:0x1
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x
p_drv:0x00
mode:DIO, clock div:2
load:0x3fff0030,len:4344
load:0x40078000,len:13824
load:0x40080400,len:3340
entry 0x40080618
MicroPython 02f9bfe-dirty on 2022-
spiram-16m) with ESP32

Type "help()" for more information
>>>
```

Thonny options

General | Interpreter | Editor | Theme & Font | Run & Debug | Terminal | Shell | Assistant

Which kind of interpreter should Thonny use for running your code?
MicroPython (ESP32)

Details

Connecting via USB cable:
Connect your device to the computer and select corresponding port below
(look for your device name, "USB Serial" or "UART").
If you can't find it, you may need to install proper USB driver first.

Connecting via WebREPL (EXPERIMENTAL):
If your device supports WebREPL, first connect via serial, make sure WebREPL is enabled
(import webrepl_setup), connect your computer and device to same network and select
< WebREPL > below

Port or WebREPL
< Try to detect port automatically >

- [x] Interrupt working program on connect
- [ ] Synchronize device's real time clock
- [x] Use local time in real time clock
- [ ] Restart interpreter before running a script

Install or update firmware

OK Cancel

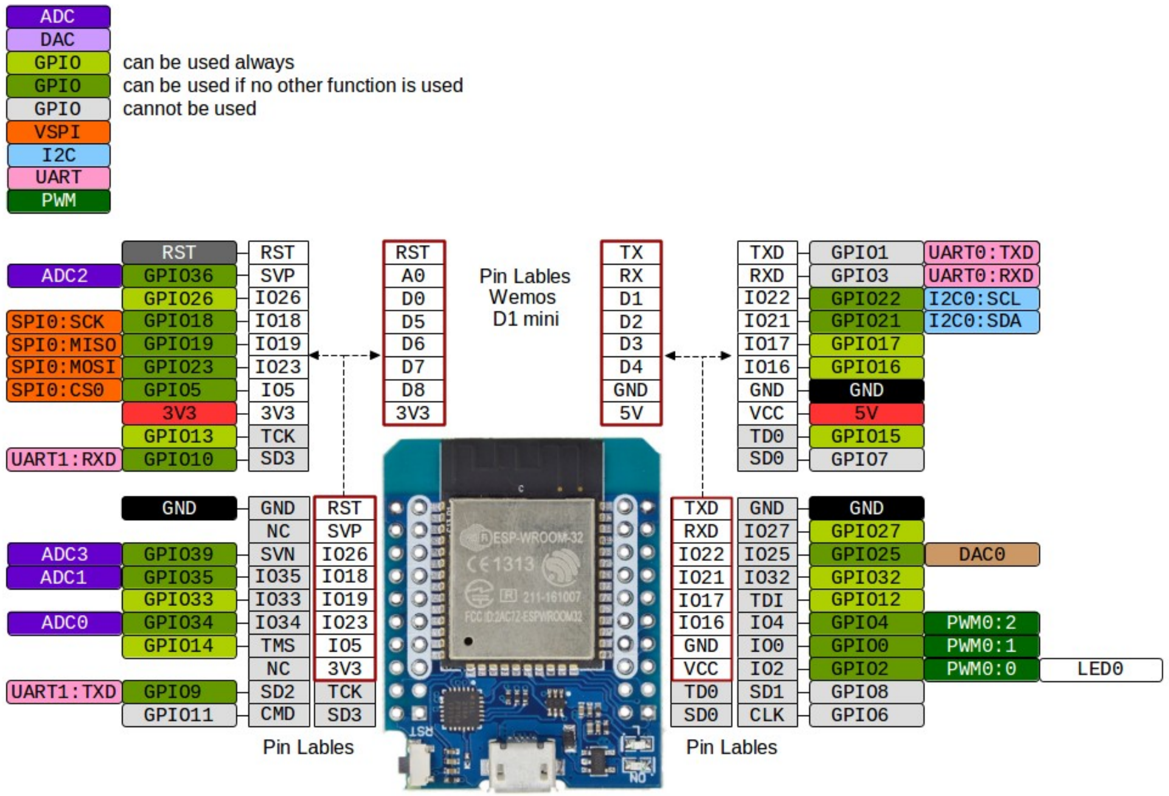
d

Interfacing to the “things”

The ESP32 has a big number of interfaces implemented on the chip:

- GPIO pins
- PWM
- Capacitive touch sensor
- I2C (**I**nter **I**ntegrated **C**ircuit) interface
- SPI (**S**erial **P**eripheral **I**nterface) interface
- Analogue to digital converter
- Digital to analogue converter
- Timer

ESP32 pinout


ADC
DAC
GPIO can be used always
GPIO can be used if no other function is used
GPIO cannot be used
VSPI
I2C
UART
PWM
RST RST RST TX TXD GPIO1 UART0:TXD
ADC2 GPIO36 SVP A0 RX RXD GPIO3 UART0:RXD
GPIO26 IO26 D0 D1 IO22 GPIO22 I2C0:SCL
SPI0:SCK GPIO18 IO18 D5 D2 IO21 GPIO21 I2C0:SDA
SPI0:MISO GPIO19 IO19 D6 D3 IO17 GPIO17
SPI0:MOSI GPIO23 IO23 D7 D4 IO16 GPIO16
SPI0:CS0 GPIO5 IO5 D8 GND GND GND
3V3 3V3 3V3 5V VCC 5V
GPIO13 TCK TDO GPIO15
UART1:RXD GPIO10 SD3 SDO GPIO7
Pin Lables Wemos D1 mini
GND GND RST TXD GND GND
NC SVP RXD IO27 GPIO27
ADC3 GPIO39 SVN IO26 IO22 IO25 GPIO25 DAC0
ADC1 GPIO35 IO35 IO18 IO21 IO32 GPIO32
GPIO33 IO33 IO19 IO17 TDI GPIO12
ADC0 GPIO34 IO34 IO23 IO16 IO4 GPIO4 PWM0:2
GPIO14 TMS IO5 GND IO0 GPIO0 PWM0:1
NC 3V3 VCC IO2 GPIO2 PWM0:0 LED0
UART1:TXD GPIO9 SD2 TCK TDO SD1 GPIO8
GPIO11 CMD SD3 SDO CLK GPIO6
Pin Lables
Pin Lables

Checking the system

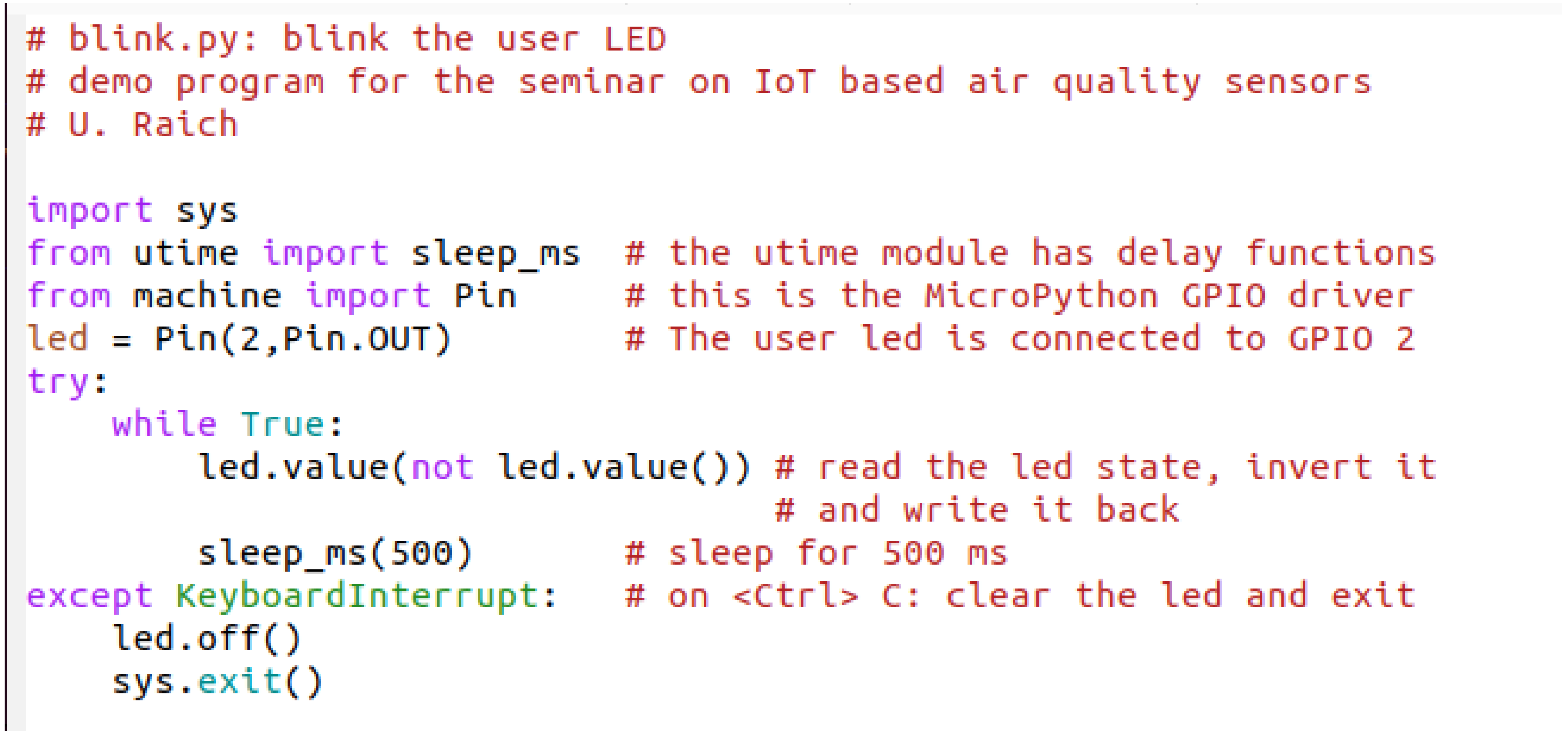
In C the ubiquitous “hello world” program checks if

- You can edit a program
- You can compile and link it
- You can run it

On embedded systems the simplest program possible is often the “blinking led” program

It shows that you can access the system and that interfacing works

IoT “Hello world”



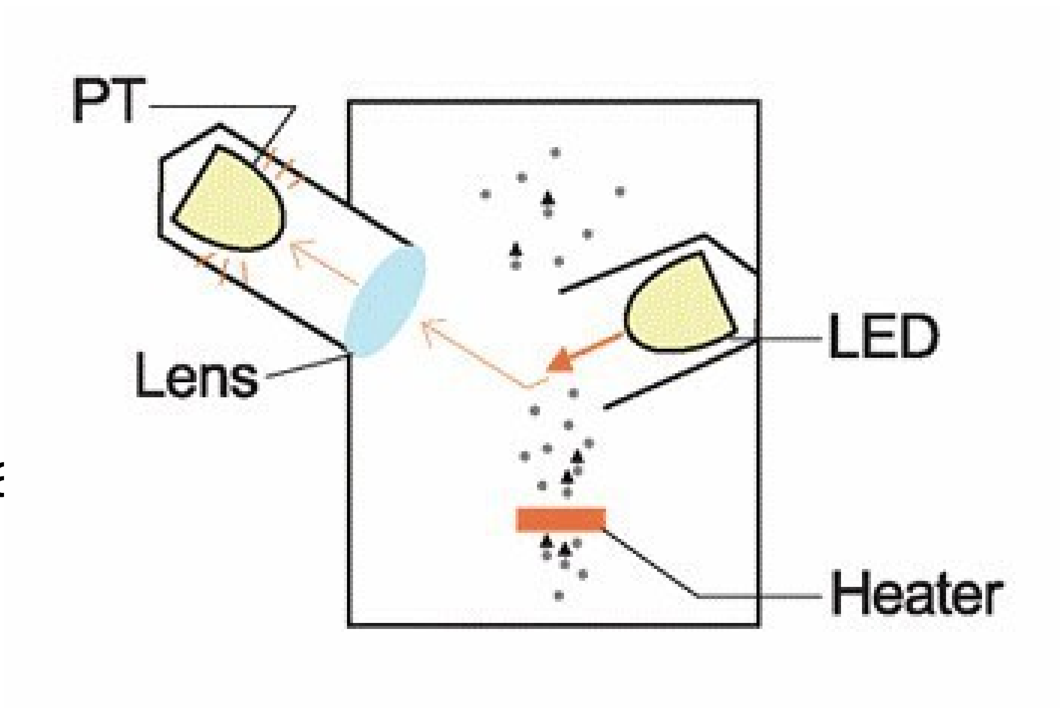
```python
# blink.py: blink the user LED
# demo program for the seminar on IoT based air quality sensors
# U. Raich

import sys
from utime import sleep_ms  # the utime module has delay functions
from machine import Pin     # this is the MicroPython GPIO driver
led = Pin(2,Pin.OUT)        # The user led is connected to GPIO 2
try:
    while True:
        led.value(not led.value()) # read the led state, invert it
                                   # and write it back
        sleep_ms(500)       # sleep for 500 ms
except KeyboardInterrupt:   # on <Ctrl> C: clear the led and exit
    led.off()
    sys.exit()
```

Dust Sensors

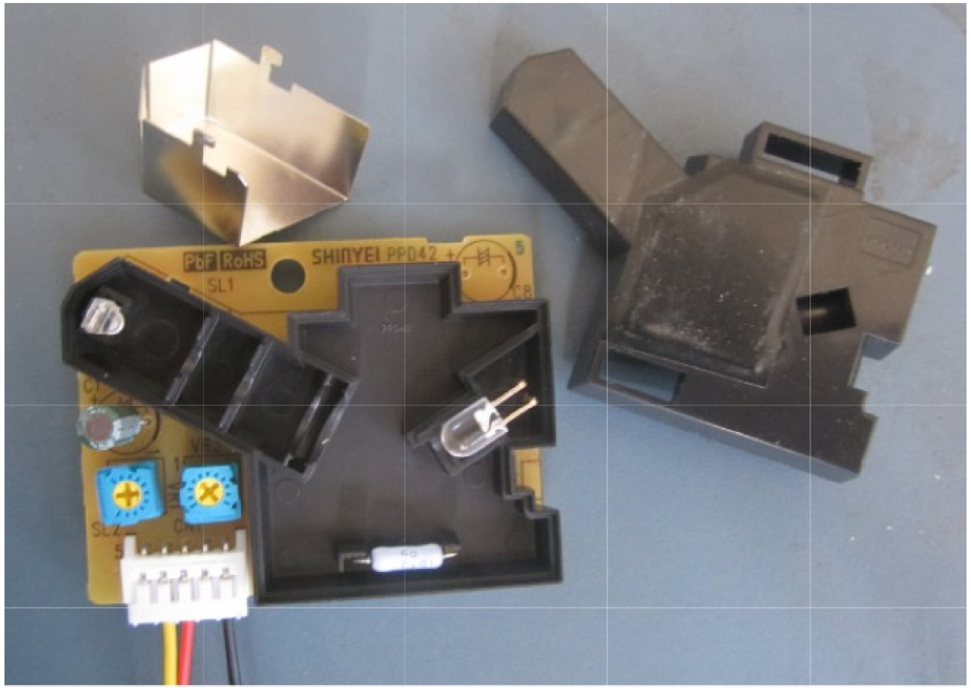
The working principle:

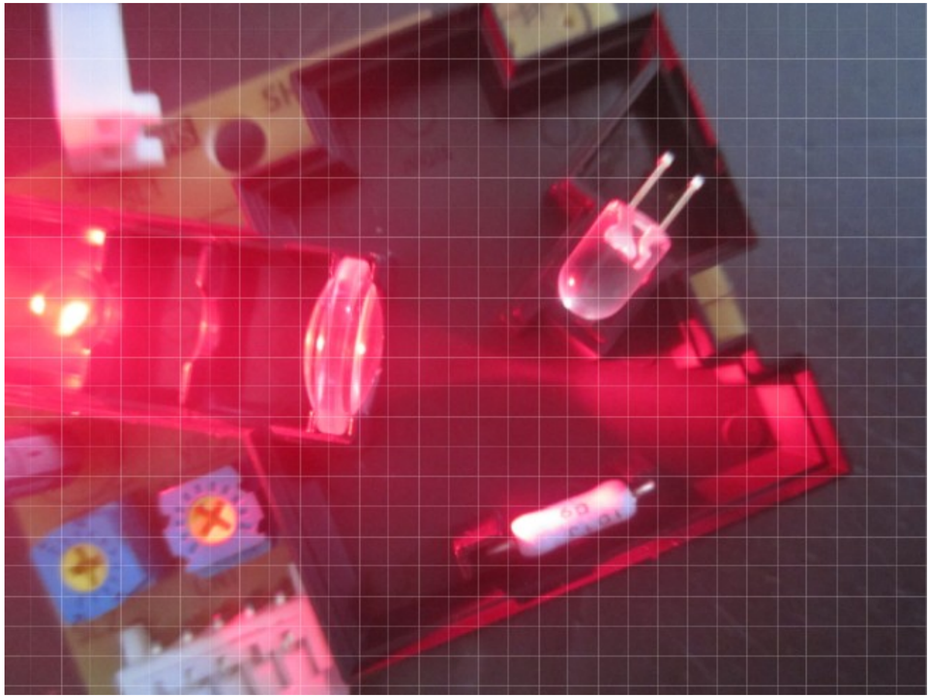
Light from an LED or Laser is scattered by the dust particle and the scattered light is detected by a photo diode.

The air flow is maintained through a ventilator or through convection by heater



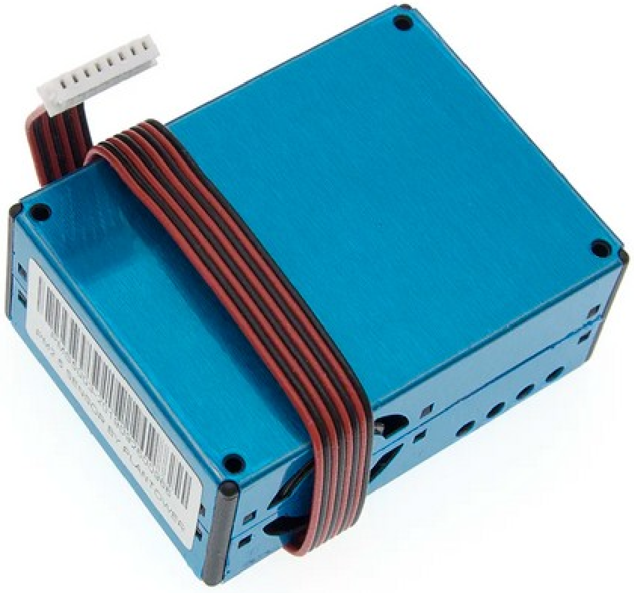
Dismounted sensor





Different sensor types

Sharp GP2Y1051


PlanTower PME 5003

DMS 501A

Differences between the dust sensors

- Air flow (ventilator on convection)
- Light source (LED, laser, light wave length)
- Interface:
  - analogue output, needs an Analogue to Digital Converter (ADC)
  - PWM output signal
  - a micro-controller transforming the signal and sending a serial text string

PWM

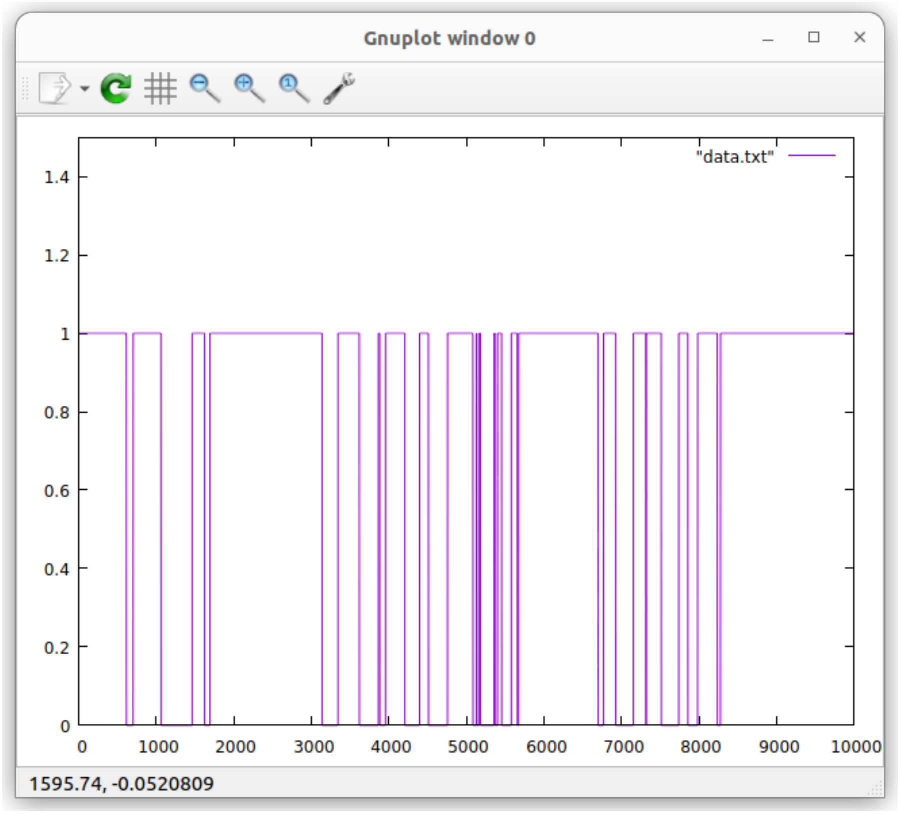
The amount of dust particles is given by the time the output signal is low.

Very clean air results in a signal that is constantly high.

We will concentrate on the PMS 5003 because it is the easiest to read out.

More on the other sensor on the Twiki:

https://afnog.iotworkshop.africa/do/view/IoT_Course_English/DustSensors



The PMS5003 dust sensor

The PMS5003 sends its measurements to the micro-controller over a simple serial line.

```
import machine

PMS5003_MSG_LENGTH = 32
uart2 = machine.UART(2,baudrate=9600,rx=17,tx=16,timeout=10000)
while True:
    rawline = uart2.read(PMS5003_MSG_LENGTH)
    print(rawline)
```

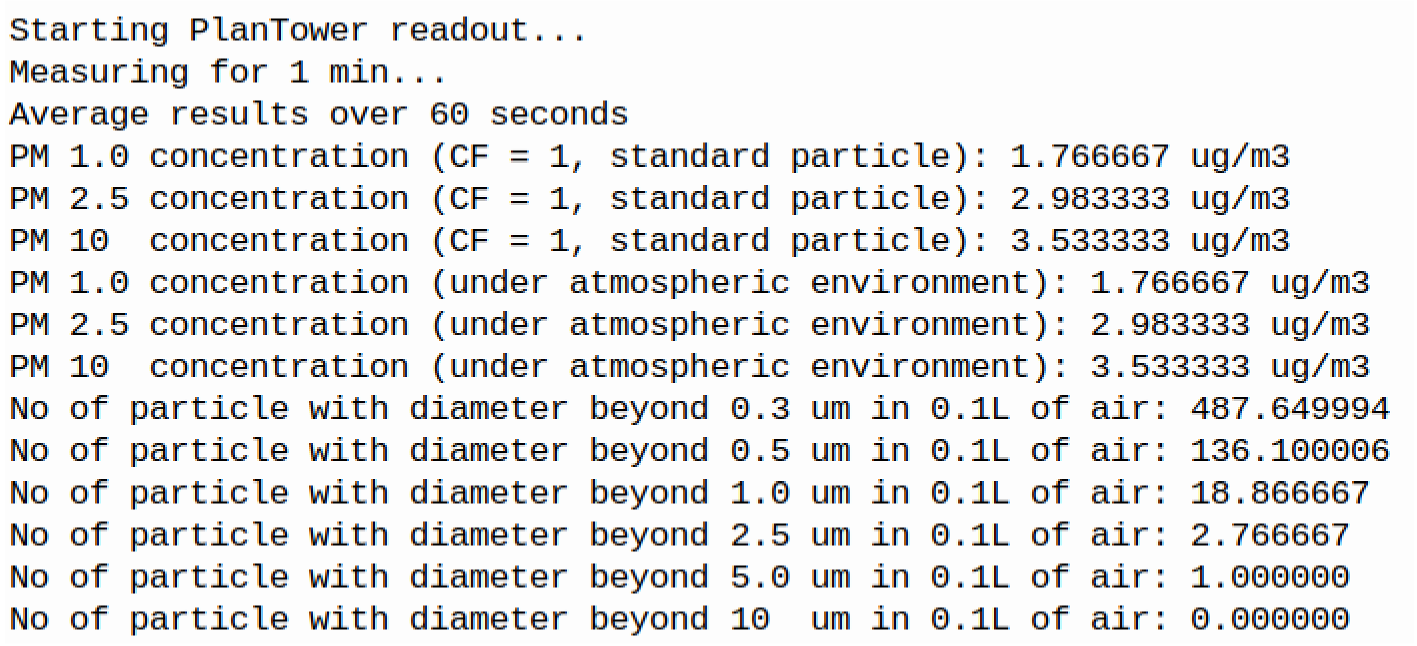
```
>>> %run -c $EDITOR_CONTENT
 b'BM\x00\x1c\x00\x02\x00\x03\x00\x03\x00\x02\x00\x03\x00\x03\x01\xf2\x0
 0\x98\x00\x13\x00\x02\x00\x00\x00\x00\x97\x00\x02\xf2'
```

Meaning of the serial message

Bytes	meaning
0,1	Magic word: ‘BM’
2,3	Frame length: data + checksum = 28
4,5	PM1.0 concentration [µg/m^3]
]6,7	PM2.5 concentration [µg/m^3]
7,9	PM10 concentration [µg/m^3]
...	
16,17	No of particles <0.3 µm in 0.1 L of air
...	
30,31	checksum

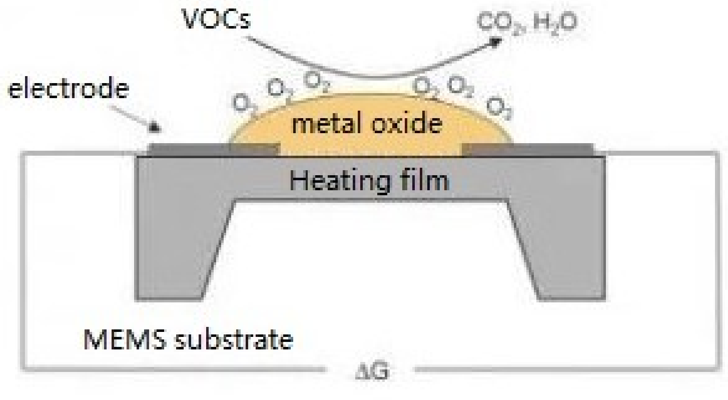
OAD

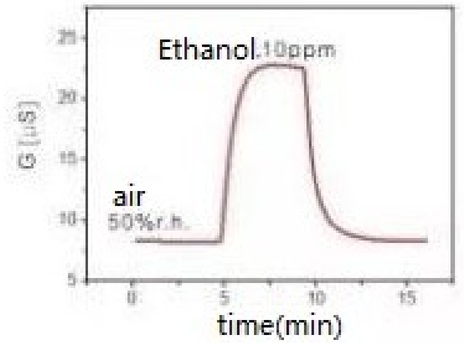
Results from the dust measurement



```
Starting PlanTower readout...
Measuring for 1 min...
Average results over 60 seconds
PM 1.0 concentration (CF = 1, standard particle): 1.766667 ug/m3
PM 2.5 concentration (CF = 1, standard particle): 2.983333 ug/m3
PM 10  concentration (CF = 1, standard particle): 3.533333 ug/m3
PM 1.0 concentration (under atmospheric environment): 1.766667 ug/m3
PM 2.5 concentration (under atmospheric environment): 2.983333 ug/m3
PM 10  concentration (under atmospheric environment): 3.533333 ug/m3
No of particle with diameter beyond 0.3 um in 0.1L of air: 487.649994
No of particle with diameter beyond 0.5 um in 0.1L of air: 136.100006
No of particle with diameter beyond 1.0 um in 0.1L of air: 18.866667
No of particle with diameter beyond 2.5 um in 0.1L of air: 2.766667
No of particle with diameter beyond 5.0 um in 0.1L of air: 1.000000
No of particle with diameter beyond 10  um in 0.1L of air: 0.000000
```

Metal Oxide gas sensors





In clean air, donor electrons in metal oxide are attracted toward oxygen which is adsorbed on the surface of the sensing material, preventing electric current flow.

In the presence of reducing gases, the surface density of adsorbed oxygen decreases as it reacts with the reducing gases. Electrons are then released into the tin dioxide, allowing current to flow freely through the sensor. The current flow will give the equivalent readings of the measured gas.

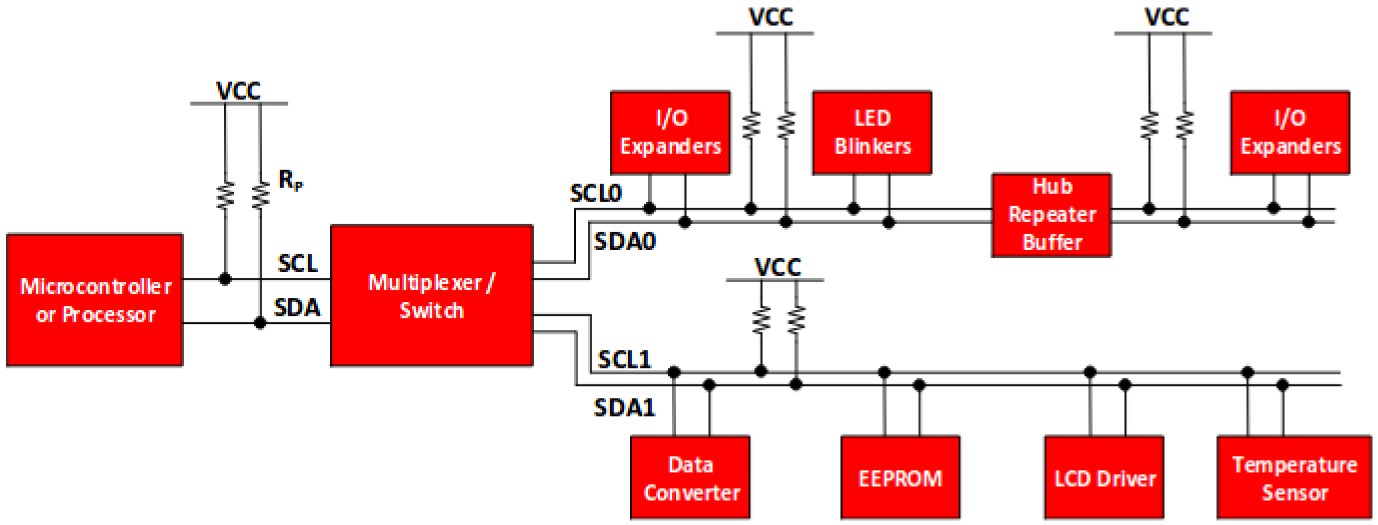
The Sensirion SGP 30

The SGP30 is a MOS sensor providing output signals for volatile organic compounds (VOCs) and for the CO_2 concentration.

It uses an I2C interface to communicate with the micro-controller

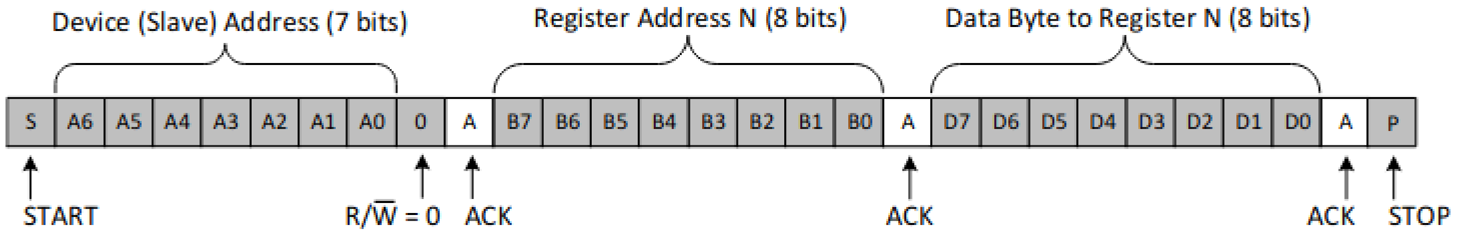
The I2C bus

The Inter-Integrated Circuit (I2C) bus was invented by Philips in the early 1980. It is used by many sensors to communicate their data to the CPU.

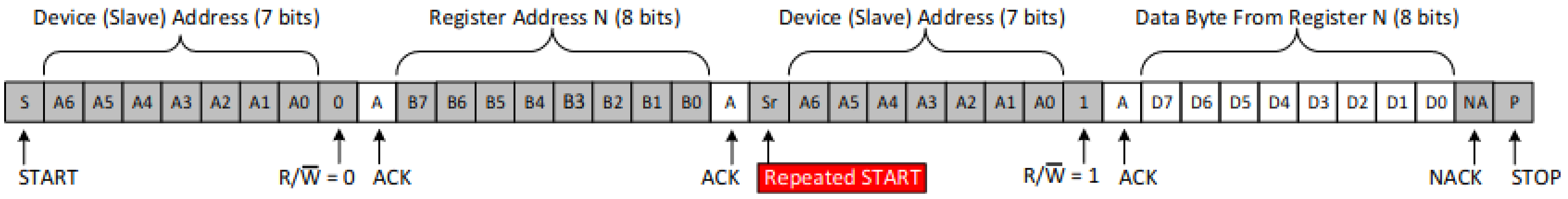


The I2C protocol

Write to the bus


Device (Slave) Address (7 bits)
Register Address N (8 bits)
Data Byte to Register N (8 bits)
S A6 A5 A4 A3 A2 A1 A0 0 A B7 B6 B5 B4 B3 B2 B1 B0 A D7 D6 D5 D4 D3 D2 D1 D0 A P
START
R/W̄ = 0
ACK
ACK
ACK
STOP

Read from the bus


Device (Slave) Address (7 bits)
Register Address N (8 bits)
Device (Slave) Address (7 bits)
Data Byte From Register N (8 bits)
S A6 A5 A4 A3 A2 A1 A0 0 A B7 B6 B5 B4 B3 B2 B1 B0 A Sr A6 A5 A4 A3 A2 A1 A0 1 A D7 D6 D5 D4 D3 D2 D1 D0 NA P
START
R/W̄ = 0
ACK
ACK
Repeated START
R/W̄ = 1
ACK
NACK
STOP

7 bit device address + R/W bit

SHT30 and SGP30

In addition to dust concentration we want to measure

- eCO2: the CO2 concentration in the air in ppm
- TVOC: the total amount of volatile organic compounds in ppb

This can be accomplished with the SGP30 air quality sensor

The SGP30 needs to know the temperature and humidity in the air, which can be measured with an SHT30 temperature and humidity sensor.
We log these measurements as well.

I2C addressing

The I2C bus uses the master/slave paradigm. The CPU acts as the master and the sensors as slaves.
The I2C bus uses 7 address bits (the 8th bit is the R/W line). A maximum of 128 slaves can therefore be connected to a single master.
The ESP32 has two hardware I2C buses, but MicroPython also provides a software I2C implementation using bit-banging on any GPIO line.
The pins for SCL (the I2C clock line) and SDA (the I2C data line) are

- SCL: GPIO 22
- SDA: GPIO 21

Control and readout of an I2C based sensor requires a driver accessing it using the MicroPython I2C driver calls. An example for the SHT30 temperature and humidity sensor is explained here.

MicroPython’s I2C driver

From the MicroPython doc on the I2C bus for the ESP32

On the ESP32 CPU:

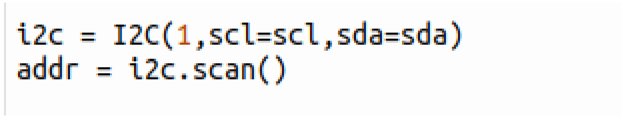
scl : GPIO 22
sda: GPIO 21

```
from machine import Pin, SoftI2C

i2c = SoftI2C(scl=Pin(5), sda=Pin(4), freq=100000)

i2c.scan()              # scan for devices

i2c.readfrom(0x3a, 4)   # read 4 bytes from device with address 0x3a
i2c.writeto(0x3a, '12') # write '12' to device with address 0x3a

buf = bytearray(10)     # create a buffer with 10 bytes
i2c.writeto(0x3a, buf)  # write the given buffer to the peripheral
```

```
from machine import Pin, I2C

i2c = I2C(0)
i2c = I2C(1, scl=Pin(5), sda=Pin(4), freq=400000)
```

Scanning the I2C bus

This is the code to read the I2C addresses of all modules connected to the bus

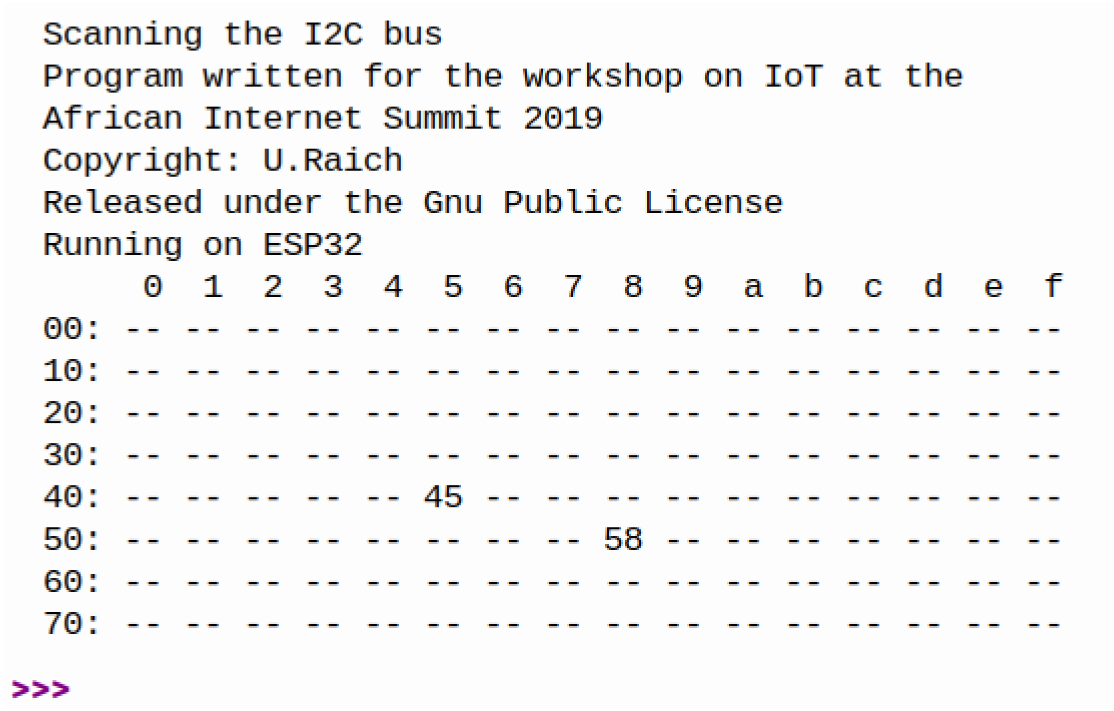


```
i2c = I2C(1,scl=scl,sda=sda)
addr = i2c.scan()
```

A list of all valid addresses is returned.

In our case we find:

- 0x45: SHT30
- 0X58: SGP30



```
Scanning the I2C bus
Program written for the workshop on IoT at the
African Internet Summit 2019
Copyright: U.Raich
Released under the Gnu Public License
Running on ESP32
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- 45 -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- 58 -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --

>>>
```

d

How to get the driver?

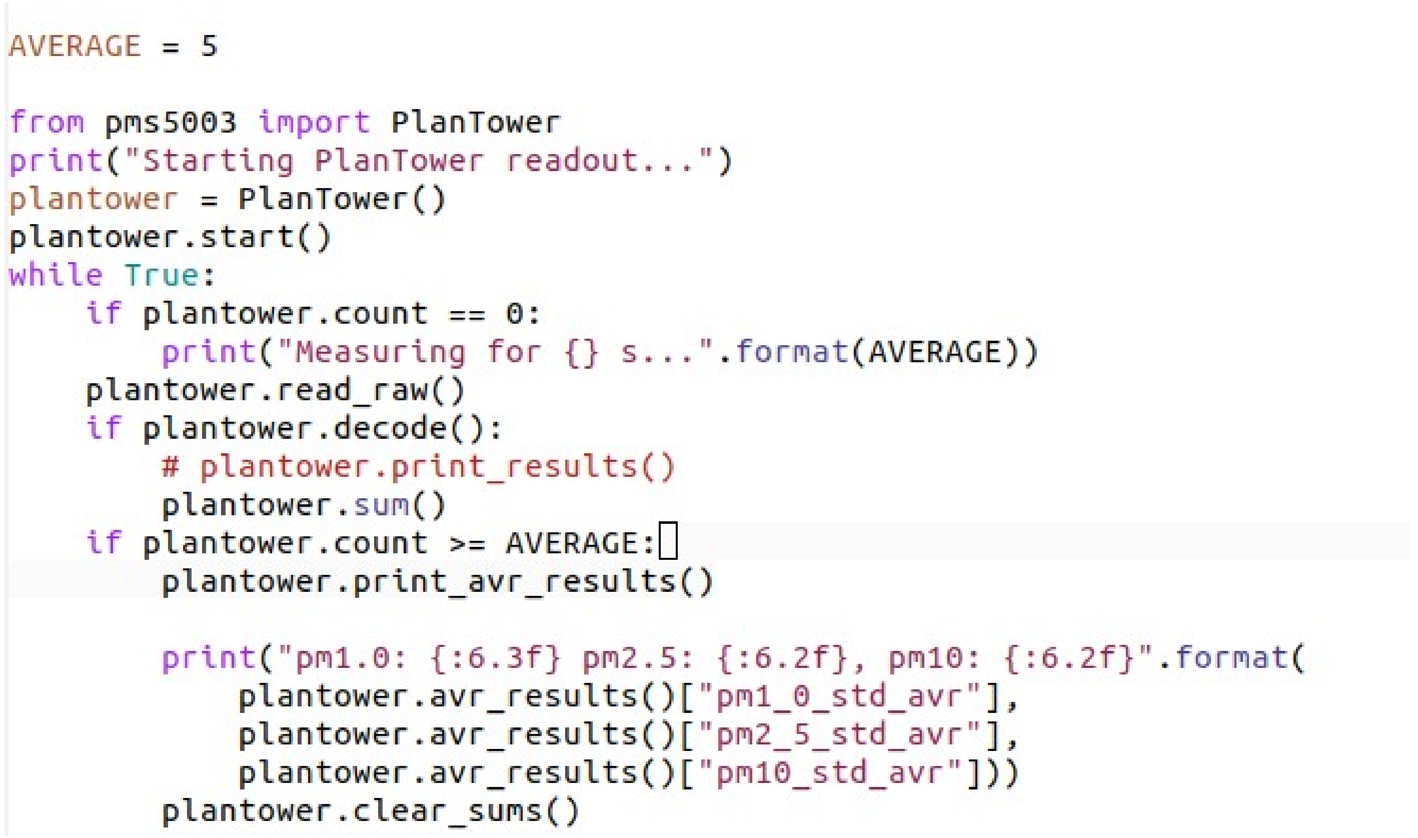
There are several possibilities:

- It is already implemented in the interpreter: You are lucky! Just use it
- Somebody has published his driver in github: Clone it and install it on /lib of the ESP32
- There is a driver for CircuitPython (Adafruit’s version of MicroPython): You can easily port this to MicroPython
- You are unlucky and you will have to write the driver yourself!

Drivers are classes

Drivers are often implemented as Python classes
When an instance of a class is created the __init__ method its called
This method initializes the I2C bus and performs any initialization that the device may need.
The ESP32 has 2 hardware I2C interfaces and I2C bus 1 is reserved for users.
MicroPython includes a driver for the I2C bus whose methods are used to communicate with the I2C device.
The sensor functions and its control is provided through registers, which must be read and written. A typical sensor can have a big number of such registers (up to ~ 100) and each bit within the register may have a specific meaning. The sensor data sheet describes all register details.
Drivers are either frozen into the firmware binary or uploaded into the /lib folder in flash.
Usually a driver is provided with a few simple demo programs showing its use.

How to use the driver



```
AVERAGE = 5

from pms5003 import PlanTower
print("Starting PlanTower readout...")
plantower = PlanTower()
plantower.start()
while True:
    if plantower.count == 0:
        print("Measuring for {} s...".format(AVERAGE))
    plantower.read_raw()
    if plantower.decode():
        # plantower.print_results()
        plantower.sum()
    if plantower.count >= AVERAGE:
        plantower.print_avr_results()

        print("pm1.0: {:6.3f} pm2.5: {:6.2f}, pm10: {:6.2f}".format(
            plantower.avr_results()["pm1_0_std_avr"],
            plantower.avr_results()["pm2_5_std_avr"],
            plantower.avr_results()["pm10_std_avr"]))
        plantower.clear_sums()
```