ENVIRONMENTAL MONITORING

PHASE 3:DEVELOPMENT PART 1

ABSTRACT:

- We had some Environment click sensors handy, so we decided to hook them up to the Arduino MKR1000 and visualise them on WolkAbout IoT Platform.
- The idea was to do a measurement every minute and publish the results once every 15 minutes.
- If the publishing of sensor readings fails (due to a busy network or some other issue), then the results should be persisted in the Flash memory of the device, With a potential maximum of 96 writes a day.

PYTHON SCRIPT:

```
import time

import random

def read_sensor_data():

 temperature = random.uniform(10, 40)

 humidity = random.uniform(20, 80)

 return temperature, humidity

def process_data(temperature, humidity):

print(f"Temperature: {temperature}°C")

 print(f"Humidity: {humidity}%")

while True:

 temperature, humidity = read_sensor_data()

 process_data(temperature, humidity)

 time.sleep(10)
```

ARDUINO UNO R3:

```
#include <Adafruit_Sensor.h>
#include <Adafruit_BME680.h>
#include <bme680_defs.h>
#include <bme680.h>

#include <WiFi101.h>
#include <RTCZero.h>
#include <FlashStorage.h>

#include "WolkConn.h"
#include "MQTTClient.h"
/*Number of outbound_message_t to store*/
#define STORAGE_SIZE 32
#define SEALEVELPRESSURE_HPA (1013.25)

/*Circular buffer to store outbound messages to persist*/
typedef struct
{
boolean valid;
outbound_message_t outbound_messages[STORAGE_SIZE];
uint32_t head;
uint32_t tail;
boolean empty;
boolean full;
} Messages;
static Messages data;
/*Connection details*/
const char* ssid = "ssid";
const char* wifi_pass = "wifi_pass";
const char *device_key = "device_key";
const char *device_password = "device_password";
const char* hostname = "api-demo.wolkabout.com";
int portno = 1883;
WiFiClient espClient;
PubSubClient client(espClient);
```

```
/* WolkConnect-Arduino Connector context */
static wolk_ctx_t wolk;
/* Init flash storage */
FlashStorage(flash_store, Messages);
/*Init i2c sensor communication*/
Adafruit_BME680 bme;
RTCZero rtc;
bool read;
/*Read sensor every minute. If you change this parameter
make sure that it's <60*/
const byte readEvery = 1;
bool publish;
/*Publish every 10 minutes. If you change this parameter
make sure that it's <60*/
const byte publishEvery = 10;
byte publishMin;
/*Flash storage and custom persistence implementation*/
void _flash_store()
{
data.valid = true;
flash_store.write(data);
}
void increase_pointer(uint32_t* pointer)
{
if ((*pointer) == (STORAGE_SIZE - 1))
{
        (*pointer) = 0;
}
else
{
(*pointer)++;
}
}
void _init()
{
data = flash_store.read();
if (data.valid == false)
{
 data.head = 0;
```

```
data.tail = 0;
data.empty = true;
data.full = false;
  }
}
bool _push(outbound_message_t* outbound_message)
{
if(data.full)
{
increase_pointer(&data.head);
}
memcpy(&data.outbound_messages[data.tail], outbound_message,
sizeof(outbound_message_t));
increase_pointer(&data.tail);
data.empty = false;
data.full = (data.tail == data.head);
return true;
}
bool _peek(outbound_message_t* outbound_message)
{
memcpy(outbound_message, &data.outbound_messages[data.head],
sizeof(outbound_message_t));
return true;
}
bool _pop(outbound_message_t* outbound_message)
{
memcpy(outbound_message, &data.outbound_messages[data.head],
sizeof(outbound_message_t));
increase_pointer(&data.head);
data.full = false;
data.empty = (data.tail == data.head);
return true;
}
bool _is_empty()
{
 return data.empty;
}
void init_wifi()
{
```

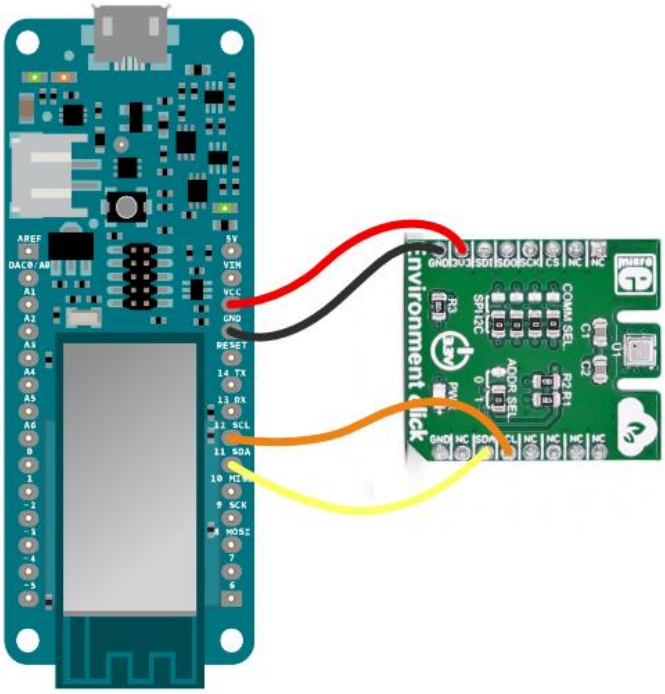
```
 if ( WiFi.status() != WL_CONNECTED) {
 while (WiFi.begin(ssid, wifi_pass) != WL_CONNECTED) {
 delay(1000);
    }
  }
}
void setup_wifi()
{
delay(10);
if ( WiFi.status() != WL_CONNECTED) {
int numAttempts = 0;
while (WiFi.begin(ssid, wifi_pass) != WL_CONNECTED) {
numAttempts++;
if(numAttempts == 10){
Serial.println("Couldn't reach WiFi!");
break;
      }
      delay(1000);
    }
  }
}

void setup() {
pinMode(LED_BUILTIN, OUTPUT);
digitalWrite(LED_BUILTIN, LOW);
/*Initialize the circular buffer structure*/
_init();
init_wifi();
wolk_init(&wolk, NULL, NULL, NULL, NULL,
            device_key, device_password, &client, hostname,
portno, PROTOCOL_JSON_SINGLE, NULL, NULL);
wolk_init_custom_persistence(&wolk, _push, _peek, _pop,
_is_empty);

/*The on board LED will turn on if something went wrong*/
if(!bme.begin())
{
 digitalWrite(LED_BUILTIN, HIGH);
}
```

```
/*Sensor init*/
bme.setTemperatureOversampling(BME680_OS_8X);
bme.setHumidityOversampling(BME680_OS_2X);
bme.setPressureOversampling(BME680_OS_4X);
bme.setIIRFilterSize(BME680_FILTER_SIZE_3);
bme.setGasHeater(320, 150); // 320*C for 150 ms
delay(200);

read = true;
publish = true;
/*Get current epoch from server*/
wolk_connect(&wolk);
delay(100);
wolk_update_epoch(&wolk);
while (!(wolk.pong_received)) {
wolk_process(&wolk);
digitalWrite(LED_BUILTIN, HIGH);
delay(1000);
}
digitalWrite(LED_BUILTIN, LOW);
wolk_disconnect(&wolk);
rtc.begin();
rtc.setEpoch(wolk.epoch_time);

rtc.setAlarmTime(rtc.getHours(), (rtc.getMinutes() +
readEvery) % 60, rtc.getSeconds());
rtc.enableAlarm(rtc.MATCH_MMSS);
rtc.attachInterrupt(alarmMatch);
publishMin = (rtc.getMinutes() + publishEvery) % 60;
WiFi.lowPowerMode();
}
void loop()
{
if(read)
{
read = false;
if (!bme.performReading()) {
      digitalWrite(LED_BUILTIN, HIGH);
}
```

```
wolk_add_numeric_sensor_reading(&wolk, "T", bme.temperature,
rtc.getEpoch());
wolk_add_numeric_sensor_reading(&wolk, "H", bme.humidity,
rtc.getEpoch());
wolk_add_numeric_sensor_reading(&wolk, "P", bme.pressure /
100.0, rtc.getEpoch());
wolk_add_numeric_sensor_reading(&wolk, "GR",
bme.gas_resistance, rtc.getEpoch());
wolk_add_numeric_sensor_reading(&wolk, "A",
bme.readAltitude(SEALEVELPRESSURE_HPA), rtc.getEpoch());
/*set new alarm*/
int alarmMin = (rtc.getMinutes() + readEvery) % 60;
rtc.setAlarmMinutes(alarmMin);
delay(100);
}
if(publish)
{
publish = false;
setup_wifi();
wolk_connect(&wolk);
if(!wolk.is_connected)
{
flash_store();
}
delay(100);
if(wolk_publish(&wolk) == W_TRUE)
{
_flash_store();
}
 /*set new publish time*/
publishMin = (rtc.getMinutes() + publishEvery) % 60;
delay(100);
wolk_disconnect(&wolk);
delay(100);
}
delay(100);
/*Timed interrupt routine*/
void alarmMatch()
{
```

```
read = true;
if(publishMin == rtc.getMinutes())
{
publish = true;
}
}
```

IOT DEVICE:



THESE CODE ARE THEROY ARE INCLUDED IN PHASE3:ENVIRONMENTAL MONITORING

BY :

B.Swetha

(422621104046)