/\*

Noise Pollution Meter. Copyleft (c) 2018 The Things Network Sevilla. https://www.thethingsnetwork.org/community/sevilla/

This program is free software; you can redistribute it and/or modify

it under the terms of the GNU General Public License version 3 as

published by the Free Software Foundation

This program is distributed in the hope that it will be useful,

but WITHOUT ANY WARRANTY; without even the implied warranty of

MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the

GNU General Public License for more details.

You should have received a copy of the GNU General Public License

along with this program. If not, see <http://www.gnu.org/licenses/>.

Authors :

- Alexandre Riasat <alexandre.riasat@podgroup.com>

- Javier Vargas <javier.vargas@podgroup.com>

- J. Félix Ontañón <felix.ontanon@podgroup.com>

\*/

// --------- BEGINING OF CONFIGURABLE FIRMWARE PARAMETERS SECTION ---------

// You can freely modify the parameters below to adapt the behaviour of the

// sound sensor. By default values provided below. Handle with care !

// You can configure the sampling ratio with the three parameters below. The default behaviour is calculating min, max, avg

// values from 60 samples along a minute (one sample a second) and then sleeping for 5 minutes.

// That provides with 12 measurements an hour, 288 a day, 2016 a week (and so on).

// WARNING: Manipulating this parameters can lead into more battery consumption !!

const int Samples = 60; // How many sound measurements before calculating the min, max and avg in dBA. 60 samples as default.

const int SamplesDelay = 1000; // How much time before taking each measurement from the sensor. 1 sample each second as default.

const int SleepTime = 300e6; // How much time to sleep before restarting the cycle. Sleep for 5 minutes as default

// Sound sensor calibration coefficients. This has been calculated for the DFROBOT Gravity Analog Sound Sensor

// DFR0034 https://www.dfrobot.com/product-83.html

// The formula to convet the analog values into dBA is a 3th degree polynomial like this:

// Equation1: sound(v) = p[0] v3 + p[1] v2 + p[2] v+ p[3] [n.u.]

// Equation2: sound-dBA =20 log10( sound(v) ) [dBA]

// The P0-P3 values below was calculated after calibrating the DFR0034 with a certified professional noise meter

// The minimum dBA is 39.39 dBA. We found this calibration offers better accuracy. The trade off is it cannot measure

// under 39.39 dBA, thus this is only useful for outdoor noise pollution measuring, but not indoor.

const float P0 = 4540403.39793356;

const float P1 = -2695328.88406384;

const float P2 = 513679.63231002;

const float P3 = -16110.00641618;

// Find below a miscelanea of firmware parameters

const int WiFiConnectionMaxTime = 30000; // 30 seconds max trying to connect to WiFi before sleeping to retry later.

const int ThingsBoardConnectionMaxTime = 10000; // 10 seconds max trying to connect to WiFi before sleeping to retry later.

const int SerialBaud = 115200; // Serial mode bauds for reporting.

const int SerialDebugMode = 1; // 1 = Print all samples on Serial Monitor, 0 = Print only the calculated report (min, max, avg) on the Serial Monitor.

// --------- END OF CONFIGURABLE FIRMWARE PARAMETERS SECTION ---------

// ESP8266/ESP32 board libraries.

#ifdef ESP8266

#include <ESP8266WiFi.h>

#elif defined(ESP32)

#include <WiFi.h>

#else

#error Only ESP8266 and ESP32 type Boards supported

#endif

// We use ThingsBoard IoT Platform MQTT transport to report measurements in this example.

// Set your ThingsBoard instance credentials on the noisepm/noisepm\_creds.h file

// Train yourself at https://thingsboard.io | https://thingsboard.io/docs/samples/esp8266/temperature/

#include <PubSubClient.h>

// Custom library to define all private credentials. Please replace with yours in that file.

#include <noisepm\_creds.h>

// We'll use WSSID / WPASS constants as WiFi Credentials to report measurements via WiFi ..

const char\* WIFI\_SSID = WIFIAP;

const char\* WIFI\_PASS = WIFIPASS;

// ... and this is to login into ThingsBoard and define the device to report data to.

const char THINGSBOARD\_SERVER[] = THINGSBOARDSERVER;

const char\* THINGSBOARD\_CLIENTID = THINGSBOARDCLIENTID;

const char\* THINGSBOARD\_TOKEN = THINGSBOARDTOKEN;

// Some global private variables.

float sound;

float maximum;

float minimum;

float average;

float Allvalue;

int temp = 1;

float MaxValue = 0;

float window[] = {0, 0, 0, 0, 0};

int valeur ;

int sensor = A0;

// The WiFi Client.

WiFiClient espClient;

// The ThingsBoard MQTT Pub/Sub client

PubSubClient client(THINGSBOARD\_SERVER, 1883, espClient);

void InitWifi() {

WiFi.persistent(false);

WiFi.mode(WIFI\_OFF);

WiFi.mode(WIFI\_STA);

WiFi.begin(WIFI\_SSID, WIFI\_PASS);

unsigned long wifiConnectStart = millis();

while (WiFi.status() != WL\_CONNECTED) {

if (millis() - wifiConnectStart > WiFiConnectionMaxTime) {

if (WiFi.status() == WL\_CONNECT\_FAILED) {

Serial.println("Failed to connect to WIFI. Please verify credentials: ");

Serial.println();

Serial.print("SSID: ");

Serial.println(WIFI\_SSID);

Serial.print("Password: ");

Serial.println(WIFI\_PASS);

Serial.println();

}

Serial.println();

Serial.println("Failed to connect to WiFi. Putting device to sleep before retrying.");

Serial.println("Please check your WiFi configuration parameters as well.");

ESP.deepSleep(SleepTime); // going to sleep

} else {

delay(500);

Serial.println(".w.");

}

}

Serial.println();

Serial.println("WiFi connected");

Serial.println("IP address: ");

Serial.println(WiFi.localIP());

Serial.println();

}

void ConnectToThingsBoard() {

unsigned long thingsBoardConnectStart = millis();

while (!client.connected()) {

// To ensure there's WiFI acquired before trying connect via MQTT

if (WiFi.status() != WL\_CONNECTED) {

Serial.println("No Wifi. Putting device to sleep.");

ESP.deepSleep(SleepTime); // going to sleep

}

if (millis() - thingsBoardConnectStart > ThingsBoardConnectionMaxTime) {

Serial.print("Failed to connect Thingsboard.");

Serial.println();

Serial.println("Putting device to sleep before retrying.");

Serial.println("Please check your credentials as well.");

ESP.deepSleep(SleepTime); // going to sleep

} else {

if (client.connect(THINGSBOARD\_CLIENTID, THINGSBOARD\_TOKEN, NULL)) {

Serial.println("Connected to ThingsBoard");

} else {

Serial.println(".t.");

delay(2000);

}

}

}

}

void report(double maxi, double minim , double avg)

{

// Prepare a JSON payload string

String payload = "{";

payload += "\"maxi\":"; payload += maxi; payload += ",";

payload += "\"minim\":"; payload += minim; payload += ",";

payload += "\"avg\":"; payload += avg;

payload += "}";

// Send payload

char attributes[100];

payload.toCharArray( attributes, 100 );

client.publish( "v1/devices/me/telemetry", attributes );

Serial.println("Reported!");

}

void UpdateMax (float Value) {

MaxValue = 0;

for (int i = 4; i > 0; i--) {

window[i] = window[i - 1];

}

window[0] = Value;

for (int a = 0; a < 5; a ++) {

if (window[a] > MaxValue) {

MaxValue = window[a];

}

}

}

void Sampling(int Sample\_D, int n\_Sample, int Sleep\_t, int Mode) {

Allvalue = 0;

maximum = 0;

minimum = 0;

average = 0;

int i = 0;

for (i = 0; i < n\_Sample ; i++) {

float OldvoltageValue;

OldvoltageValue = analogRead(sensor) \* (3.3 / 1024);

if (OldvoltageValue <= 0.039) {

OldvoltageValue = 0.039;

}

UpdateMax(OldvoltageValue);

sound = P0 \* pow(MaxValue, 3) + P1 \* pow(MaxValue, 2) + P2 \* MaxValue + P3;

sound = 20 \* log10(sound);

if (i == 0) {

maximum = sound; // For the first meausure

minimum = sound;

} else if (sound < minimum) { // If the actually sound is lower than the lowest sound measured

minimum = sound;

} else if (sound > maximum) { // If the actually sound is higher than the highest sound measured

maximum = sound;

}

Allvalue = Allvalue + sound; // All the values for this sampling

if (Mode == true) {

average = Allvalue / (i + 1); // Makes the average of the previous measures

Serial.println("");

Serial.print(average);

Serial.print(",");

Serial.print(maximum);

Serial.print(",");

Serial.print(minimum);

}

delay(Sample\_D);

}

average = Allvalue / n\_Sample; // Makes the average of this sample

if (Mode == false) {

Serial.println("");

Serial.print(average);

Serial.print(",");

Serial.print(maximum);

Serial.print(",");

Serial.print(minimum);

}

report(maximum, minimum, average);

delay(1000); // to make sure that it is reported

ESP.deepSleep(Sleep\_t); // going to sleep

}

void setup() {

Serial.begin(SerialBaud);

Serial.print("Connecting to WiFi: ");

Serial.println(WIFI\_SSID);

InitWifi();

client.connect(THINGSBOARD\_SERVER);

Serial.print("Connecting to ThingsBoard: ");

Serial.println(THINGSBOARD\_SERVER);

Serial.println(THINGSBOARD\_CLIENTID);

Serial.println(THINGSBOARD\_TOKEN);

if ( !client.connected() ) {

ConnectToThingsBoard();

}

}

void loop() {

Sampling(SamplesDelay, Samples, SleepTime, SerialDebugMode);

client.loop();

}