// This code includes snippets from https://github.com/empierre/arduino/blob/master/AirQuality-Multiple\_Gas\_Sensor1\_4.ino

// for the MQ-135 gas sensor, and uses Adafruit graphics libraries for the OLED display.

// License: Attribution-NonCommercial-ShareAlike 3.0 Unported (CC BY-NC-SA 3.0)

#include <Wire.h>

#include <Adafruit\_SSD1306.h>

#include <Adafruit\_GFX.h>

// OLED display TWI address

#define OLED\_ADDR 0x3C

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Hardware Related Macros\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define MQ135\_SENSOR (2) //I have it connected to Analog pin A2

#define RL\_VALUE (1000) //define the load resistance on the board, in ohms

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Software Related Macros\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define CALIBRATION\_SAMPLE\_TIMES (50) //define how many samples you are going to take in the calibration phase

#define CALIBRATION\_SAMPLE\_INTERVAL (500) //define the time interal(in milisecond) between each samples in the

//cablibration phase

#define READ\_SAMPLE\_INTERVAL (50) //define how many samples you are going to take in normal operation

#define READ\_SAMPLE\_TIMES (5) //define the time interal(in milisecond) between each samples in

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Application Related Macros\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#define GAS\_CO2 (2)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Globals\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

float CO2Curve[2] = {113.7105289, -3.019713765}; //MQ135

float Ro = 10000; //Ro is initialized to 10 kilo ohms

unsigned long SLEEP\_TIME = 2; // Sleep time between reads (in seconds)

//VARIABLES

float Ro4 = 2.511; //MQ135 2.51 this has to be tuned 10K Ohm

float RL4 = 1.000; //MQ135 FC-22

int val = 0; //variable to store the value coming from the sensor

int cur = 0; //current measured value

int peak = 0; //peak measured value for comparison

Adafruit\_SSD1306 display(-1);

#if (SSD1306\_LCDHEIGHT != 64)

#error("Height incorrect, please fix Adafruit\_SSD1306.h!");

#endif

#define CHILD\_ID\_MQ135 4

void setup()

{

// initialize and clear display

display.begin(SSD1306\_SWITCHCAPVCC, OLED\_ADDR);

display.clearDisplay();

display.display();

// display a line of text

display.setTextSize(2);

display.setTextColor(WHITE);

display.setCursor(0,0);

display.print("SAFETY");

display.setTextSize(1);

display.println("Calibrating");

display.println("CO2 Sensor");

display.println("Please wait");

// update display with all of the above graphics

display.display();

delay(2000);

pinMode( 4, INPUT); // set digital pin 4 to input mode

Ro4 = MQCalibration(MQ135\_SENSOR,400,RL4,CO2Curve);

// Serial.begin(9600); // sets the serial port to 9600

// Serial.print("MQ135 Ro4 value:");

// Serial.println(Ro4); // print the calculated value of Ro4

display.clearDisplay();

display.display();

}

void loop()

{

// MQ135 CO2

// Serial.print("MQ135 :");

// Serial.print("CO2 :");

// Serial.print(MQGetGasPercentage(MQRead(MQ135\_SENSOR,RL4),Ro4,GAS\_CO2,MQ135\_SENSOR) );

// Serial.println( "ppm" );

display.setTextSize(2);

display.setTextColor(WHITE);

display.setCursor(0,0); display.print("CO2"); display.setCursor(65,0); display.print("MAX");

cur = MQGetGasPercentage(MQRead(MQ135\_SENSOR,RL4),Ro4,GAS\_CO2,MQ135\_SENSOR);

if (cur > peak) peak = cur;

display.setCursor(0,21); display.print(cur); display.setCursor(65,21); display.print(peak);

display.setCursor(0,41); display.print("ppm"); display.setCursor(65,41); display.print("ppm");

display.display();

// delay(100); // delay between display updates

display.clearDisplay();

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* MQResistanceCalculation \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Input: raw\_adc - raw value read from adc, which represents the voltage

Output: the calculated sensor resistance

Remarks: The sensor and the load resistor forms a voltage divider. Given the voltage across the load resistor and its resistance, the resistance of the sensor could be derived.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

float MQResistanceCalculation(int raw\_adc,float rl\_value)

{

return (long)((long)(1024\*1000\*(long)rl\_value)/raw\_adc-(long)rl\_value);

;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* MQCalibration \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Input: mq\_pin - analog channel

Output: Ro of the sensor

Remarks: This function assumes that the sensor is in clean air. It use

MQResistanceCalculation to calculates the sensor resistance in clean air. .

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

float MQCalibration(int mq\_pin, double ppm, double rl\_value,float \*pcurve )

{

int i;

float val=0;

for (i=0;i<CALIBRATION\_SAMPLE\_TIMES;i++) { //take multiple samples

val += MQResistanceCalculation(analogRead(mq\_pin),rl\_value);

delay(CALIBRATION\_SAMPLE\_INTERVAL);

}

val = val/CALIBRATION\_SAMPLE\_TIMES; //calculate the average value

//Ro = Rs \* sqrt(a/ppm, b) = Rs \* exp( ln(a/ppm) / b )

return (long)val\*exp((log(pcurve[0]/ppm)/pcurve[1]));

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* MQRead \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Input: mq\_pin - analog channel

Output: Rs of the sensor

Remarks: This function use MQResistanceCalculation to caculate the sensor resistenc (Rs).

The Rs changes as the sensor is in the different consentration of the target

gas. The sample times and the time interval between samples could be configured

by changing the definition of the macros.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

float MQRead(int mq\_pin,float rl\_value)

{

int i;

float rs=0;

for (i=0;i<READ\_SAMPLE\_TIMES;i++) {

rs += MQResistanceCalculation(analogRead(mq\_pin),rl\_value);

delay(READ\_SAMPLE\_INTERVAL);

}

rs = rs/READ\_SAMPLE\_TIMES;

return rs;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* MQGetGasPercentage \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Input: rs\_ro\_ratio - Rs divided by Ro

gas\_id - target gas type

Output: ppm of the target gas

Remarks: This function passes different curves to the MQGetPercentage function which

calculates the ppm (parts per million) of the target gas.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int MQGetGasPercentage(float rs\_ro\_ratio, float ro, int gas\_id, int sensor\_id)

{

if (sensor\_id == MQ135\_SENSOR ){

if ( gas\_id == GAS\_CO2 ) {

return MQGetPercentage(rs\_ro\_ratio,ro,CO2Curve); //MQ135

}

}

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* MQGetPercentage \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Input: rs\_ro\_ratio - Rs divided by Ro

pcurve - pointer to the curve of the target gas

Output: ppm of the target gas

Remarks: By using the slope and a point of the line. The x(logarithmic value of ppm)

of the line could be derived if y(rs\_ro\_ratio) is provided. As it is a

logarithmic coordinate, power of 10 is used to convert the result to non-logarithmic

value.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int MQGetPercentage(float rs\_ro\_ratio, float ro, float \*pcurve)

{

return (double)(pcurve[0] \* pow(((double)rs\_ro\_ratio/ro), pcurve[1]));

}