

Informe sobre el software

Medidor de CO2: Aire Nuevo

Lenguaje de programación

Se está utilizando el lenguaje de programación Arduino para programar las placas, mediante el entorno de desarrollo oficial de Arduino que se puede descargar en

<https://www.arduino.cc/en/software>

Librerías utilizadas

- Para el uso del sensor **MH-Z19C** se está utilizando la librería **mhz19_uart** que se puede descargar desde el siguiente enlace https://github.com/piot-jp-Team/mhz19_uart
- Para el uso del display **LCD LiquidCrystal I2C** se está utilizando la librería **Arduino-LiquidCrystal-I2C-library** que se puede descargar desde el siguiente enlace <https://github.com/fdebrabander/Arduino-LiquidCrystal-I2C-library>

Se recomienda descargar las librerías utilizadas desde los enlaces y no desde el gestor de bibliotecas del **IDE** porque existen varias librerías distintas con nombres iguales y distintos funcionamientos.

Placa utilizada

Actualmente se está utilizando la placa **WEMOS D1 Mini**, para utilizarla se tiene que preparar el entorno de desarrollo, esto se logra siguiendo los pasos a continuación:

1. Se debe abrir el **IDE** e ir a preferencias. Agregar el siguiente enlace https://arduino.esp8266.com/stable/package_esp8266com_index.json a la sección de Gestor de URLs adicionales de tarjetas.
2. Abrir el **Gestor de tarjetas** desde el menú de **Placas** en **Herramientas**, buscar **ESP8266** e instalar el paquete de **ESP8266 Community**.
3. Ir a **Añadir bibliotecas** desde el menú de **Incluir librerías** en **Programa**, e instalar el paquete **ESP8266 Microgear** de **Chavee Issariyapat**.
4. Seleccionar la placa **WEMOS D1 R1** desde el menú de **Placas** en **Herramientas**.

Driver del microcontrolador

El microcontrolador de esta placa es un **CH340G**, se debe instalar un driver para que la computadora reconozca cuando se conecta una placa. Este driver se puede descargar desde http://www.wch.cn/download/CH341SER_EXE.html para su sistema operativo correspondiente.

Funcionamiento del código

Primero incluimos las librerías que vamos a utilizar:

- **pitches.h** solo es un header file que tiene definidas distintas notas musicales para su uso en el buzzer
- **MHZ19_uart.h** permite controlar el sensor
- **Wire.h** es necesaria para la librería que maneja el display
- **LiquidCrystal_I2C.h** como se mencionó antes, es la librería que nos permite controlar el display

```
#include "pitches.h"  
#include <MHZ19_uart.h>  
#include <Wire.h>  
#include <LiquidCrystal_I2C.h>
```

Luego definimos las constantes para los distintos pines.

- **rx_pin** y **tx_pin** son los pines en los que está conectado el sensor
- **pinLed** es el pin donde está conectado un led simple de color rojo
- **pinBuzzer** es el pin en el cual está conectado el buzzer
- **pinCalib** es el pin en el que está conectado un pulsador que se utiliza para la calibración del dispositivo
- **numeroSerie** contiene el número de serie del medidor, este número se imprime tanto por serial como por el display

```
const int rx_pin = 13;  
const int tx_pin = 15;  
const int pinLed = 16;  
const int pinBuzzer = 14;  
const int pinCalib = 12;  
const String numeroSerie = "0000";
```

En loops llevamos un **contador** de cuantas veces se ejecutó el **loop** del programa, luego **iniciamos** los objetos del **sensor** y la **pantalla** usando las librerías mencionadas antes

```
long loops = 0;
MHZ19_uart sensor;
LiquidCrystal_I2C display(0x27,16,2);
```

La siguiente función es la alarma que se utiliza en el medidor, se le pasan dos parámetros, cantidad de veces que se quiere que suene y cuanto debe durar el pitido, la duración se mide en **milisegundos**. Luego el programa **prende** el **led**, **hace sonar** el **buzzer** con la nota **C7** durante el tiempo definido. La nota se puede cambiar y poner otra en base al archivo **pitches.h** o simplemente escribir el número de la frecuencia, en este caso usamos **C7** porque es la frecuencia en la que mejor funciona el **buzzer** que estamos utilizando). Una vez que transcurre el tiempo definido, el **buzzer deja de sonar** y se **apaga** el **led**, luego, hay una **pausa** con la misma duración que la nota, para que al repetir la nota sea un **intervalo del mismo tiempo**.

```
void alarma(int veces, int duracionNota) {
    for(int i=0; i<veces; i++)
    {
        digitalWrite(pinLed, HIGH);
        tone(pinBuzzer, NOTE_C7, duracionNota);
        delay(duracionNota);
        noTone(pinBuzzer);
        digitalWrite(pinLed, LOW);
        delay(duracionNota);
    }
}
```

La siguiente función permite que llamandola diciéndole donde se quiere posicionar el **cursor**, y el **mensaje** a imprimir, se pueda imprimir mensajes a la pantalla con una sola línea.

Se le debe pasar por parametro la **posición** (de 0 a 15) y la **línea** (de 0 a 1) con el mensaje en formato **string**

```
void displayPrint(int posicion, int linea, String texto) {
    display.setCursor(posicion, linea);
    display.print(texto);
}
```

Esta función sirve para dibujar el **logo** de la **UNAHUR** en el **display**. Para ello primero creamos unos **caracteres personalizados** en **bytes**, luego posicionamos el **cursor** del **display** en las posiciones correspondientes y los escribimos, en este caso **no** es posible reutilizar la función **displayPrint()** porque para imprimir caracteres personalizados se debe utilizar la función **write()** de la librería del display, en cambio, en los mensajes regulares se utiliza **print()**

```
void logoUNAHUR() {
    byte UNAHUR1[] = {
        B11100,
        B11110,
        B11111,
        B11111,
        B11111,
        B11111,
        B11111,
        B01111,
        B00111
    };
    byte UNAHUR2[] = {
        B11111,
        B11111,
        B11111,
        B11111,
        B11111,
        B11111,
        B11111,
        B11111,
        B11111
    };
    byte UNAHUR3[] = {
        B00111,
        B01111,
        B11111,
        B11111,
        B11111,
        B11111,
        B11111,
        B11110,
        B11100
    };
    display.createChar(0, UNAHUR1);
    display.createChar(1, UNAHUR2);
    display.createChar(2, UNAHUR3);
    display.setCursor(13, 0);
    display.write(0);
    display.setCursor(14, 0);
    display.write(1);
    display.setCursor(15, 0);
```

```

display.write(2);
display.setCursor(13, 1);
display.write(2);
display.setCursor(14, 1);
display.write(1);
display.setCursor(15, 1);
display.write(0);
}

```

Para imprimir el **CO2** en pantalla utilizamos la siguiente función, se le pasa por parametro el **valor que recibió el sensor** y se imprime tanto por **Serial** como por **display**. En esta misma función llamamos a **logoUNAHUR()**

```

void imprimirCO2(int co2ppm) {
    Serial.print("CO2: " + String(co2ppm) + "ppm \n");
    displayPrint(0, 1, "          ");
    displayPrint(0, 1, "CO2: " + String(co2ppm) + "ppm");
    logoUNAHUR();
}

```

La siguiente función la usamos para **calibrar** el equipo, primero definimos la cantidad de **segundos a esperar**, este número es el tiempo que el equipo va a esperar para ejecutar la **calibración**, lo **recomendado** por el fabricante es que sean **al menos 20 minutos**, nosotros esperamos **30 minutos** para estar más **seguros**. Este número se maneja en segundos así que convertimos 30 minutos a **1800 segundos**.

Luego se imprime un mensaje notificando que **comienza la calibración** que se puede ver durante 10 segundos, luego empieza el proceso de calibración. **Mientras los segundos pasados son menores a los segundos a esperar** mencionados anteriormente, se va mostrando por **pantalla** y por **serial** cuantos minutos van transcurriendo, **cada un minuto** se muestra el **CO2** que está sensando en el momento.

Una vez **pasados los 30 minutos**, se ejecuta la función **calibrateZero()** de la librería del **sensor** y se notifica por **serial** y por **pantalla**, luego de un minuto se ejecuta **de nuevo** la misma función y notifica que se hizo una **segunda calibración**. Esto es para estar seguros en caso de que hubiera algún problema con la primera, **no es necesario** que pasen **30 minutos entre ambas calibraciones**, los 30 minutos **solo son necesarios desde que se inicia hasta la primer calibración**.

Cuando ambas calibraciones son ejecutadas, se notifica por **serial**, por **pantalla**, y suena una **alarma** de **cinco** pitidos indicando que el proceso fue **finalizado**.

```

void calibrar()
{
    const long segundosEspera = 1800;
    long segundosPasados = 0;
    Serial.print("COMIENZA CALIBRACION \n");
    display.clear();
    displayPrint(0, 0, "COMIENZA");
    displayPrint(0, 1, "CALIBRACION");
    delay(10000); // Espera 10 segundos

    while(segundosPasados <= segundosEspera) {
        if (++segundosPasados % 60 == 0) {
            Serial.print(String(segundosPasados / 60) + " minutos \n");
            Serial.print("CO2: " + String(sensor.getPPM()) + "ppm \n");
            display.clear();
            displayPrint(0, 0, String(segundosPasados / 60));
            displayPrint(7, 0, "minutos");
            displayPrint(0, 1, "CO2: ");
            displayPrint(8, 1, String(sensor.getPPM()));
            displayPrint(12, 1, "ppm");
        }
        else {
            display.clear();
            displayPrint(0, 0, "CALIBRANDO");
            displayPrint(0, 1, String(segundosPasados / 60));
            displayPrint(7, 1, "minutos");
        }
        delay(1000);
    }
    sensor.calibrateZero();
    Serial.print("PRIMERA CALIBRACION \n");
    display.clear();
    displayPrint(0, 0, "PRIMERA");
    displayPrint(0, 1, "CALIBRACION");
    alarma(1, 250);
    delay(60000);
    sensor.calibrateZero();
    Serial.print("SEGUNDA CALIBRACION \n");

    display.clear();
    displayPrint(0, 0, "SEGUNDA");
    displayPrint(0, 1, "CALIBRACION");
    alarma(1, 250);
    delay(10000);
}

```

```

Serial.print("CALIBRACION TERMINADA \n");
display.clear();
displayPrint(0, 0, "CALIBRACION");
displayPrint(0, 1, "TERMINADA");
alarma(5, 250);
delay(10000);
}

```

Para el mensaje que va moviéndose tenemos que crear un **array** con los caracteres y dos funciones, **scrollingText()** se encarga de imprimir todos los caracteres una vez, y **aireNuevo()** se encarga de hacer esa impresión durante 11 veces para imprimir los 10 caracteres y que queden de nuevo en la posición inicial con un espacio en blanco al final.

```

const int STR_LEN 12
char str_to_print[STR_LEN]={'A','i','r','e',' ','N','u','e','v','o'};

void scrollingText(uint8_t scrolled_by) {
    for (uint8_t i=0;i<11;i++) {
        display.setCursor(i,0);
        if (scrolled_by>=11) scrolled_by=0;
        if (scrolled_by<10) display.print(str_to_print[scrolled_by]);
        else display.print(' ');
        scrolled_by++;
    }
}

void aireNuevo() {
    for (uint8_t i=0;i<STR_LEN;i++) {
        scrollingText(i);
        delay(500);
    }
}

```

Para las siguientes funciones tengo que mencionar un poco como es la estructura del código de **Arduino**. Todo programa de arduino es llamado **Sketch**, estos se dividen en dos partes principales, el **setup()** y el **loop()**, antes de estos se puede crear otras funciones que se quieran usar, variables, etc. Pero el **setup()** y el **loop()** son dos funciones **necesarias** para el funcionamiento de un programa de arduino.

En el **setup()** se debe **configurar el funcionamiento**, se configura si los pines son entrada o salida, se limpia la pantalla del display, etc. En el **loop()** está el **código principal** que se ejecuta en **bucle** mientras el dispositivo esté encendido.

En nuestro **setup** tenemos la **configuración de los pines** del **led**, el **buzzer** y el **pulsador para la calibración**. El **led** y el **buzzer** están como **salida** y el **pulsador** como **INPUT_PULLUP**, esto básicamente indica que en ese pin tenemos un pulsador para que se pueda sensor cuando está siendo presionado.

Luego **iniciamos** el canal de **serial**, **limpiamos la pantalla** y **prendemos la luz trasera** de esta. Suena una **alarma** indicando que el dispositivo se encendió y se muestra el **número de serie** y un cartel indicando que el equipo está iniciando. Esto se muestra tanto por serial como por pantalla. Imprimimos en pantalla el **logo de la UNAHUR** y luego de 10 segundos limpiamos la pantalla e **iniciamos** el **sensor** indicándole los pines en los que está conectado.

Deshabilitamos la calibración automática del sensor con **setAutoCalibration(false)** porque puede traer problemas y **más preciso** calibrarlo manualmente con el pulsador.

Luego **imprimimos** tanto por serial como por pantalla un mensaje indicando que el dispositivo se **está calentando**, esto es porque el sensor necesita **un minuto** de calentamiento para poder empezar a funcionar **correctamente**. Una vez pasado este minuto, se vuelve a limpiar la pantalla y suena una **alarma** de **tres** pitidos indicando que ya **terminó el calentamiento**.

```
void setup() {
  pinMode(pinLed, OUTPUT);
  pinMode(pinBuzzer, OUTPUT);
  pinMode(pinCalib, INPUT_PULLUP);
  Serial.begin(115200);
  display.begin();
  display.clear();
  display.backlight();
  alarma(1, 250);
  Serial.print("N° de serie " + numeroSerie + "\n");
  Serial.print("INICIANDO \n");
  displayPrint(0, 0, "N/S: " + numeroSerie);
  displayPrint(0, 1, "INICIANDO");
  logoUNAHUR();
  delay(10000);
  display.clear();
  sensor.begin(rx_pin, tx_pin);
  sensor.setAutoCalibration(false);
  Serial.print("Calentando, espere 1 minuto \n");
  displayPrint(0, 0, "Calentando");
  displayPrint(0, 1, "Espere 1 minuto");
  delay(60000);
  display.clear();
  alarma(3, 250);
}
```


En el **loop** primero tenemos un **chequeo** para revisar el **estado del pulsador**, si está siendo **pulsado**, suena una **alarma** y se **ejecuta** la **calibración**.

Luego revisamos cuantas **veces** se ejecutó el **loop**, si fue **ejecutado 30 veces** se muestra por pantalla y serial, un **mensaje presentando al medidor**.

Después se limpia la pantalla. En la **línea superior** se imprime el mensaje **“Aire Nuevo”** y en la **línea inferior** se imprime el **CO2**, a la **derecha** de ambas líneas se dibuja el **logo** de la **UNAHUR**.

En caso de que el **CO2** exceda las **800ppm** (partes por millón) suena una **alarma** de **dos** pitidos **lentos**. Si el **CO2** excede los **1000ppm**, suena una alarma de **cuatro** pitidos un poco **más rápidos**. Si el **CO2** excedió los **1200ppm**, empieza a sonar un **pitido rápido** que **no cesa hasta que las partes por millón desciendan de los 1200**. Luego ejecuta el **mensaje animado** y se **espera 5 segundos** y para volver a ejecutar el loop. **Entre mediciones hay 10 segundos**, 5 segundos transcurren en el mensaje animado y 5 se esperan.

```
void loop() {
  if (digitalRead(pinCalib) == LOW) {
    alarma(1, 250);
    calibrar();
  }
  if(++loops % 30 == 0) {
    Serial.print("AireNuevo UNAHUR \n");
    Serial.print("MEDIDOR de CO2 \n");
    display.clear();
    displayPrint(0, 0, "AireNuevo UNAHUR");
    displayPrint(0, 1, "MEDIDOR de CO2");
    delay(5000);
    loops = 0;
  }
  display.clear();
  displayPrint(0, 0, "Aire Nuevo");
  while(sensor.getPPM() >= 1200) {
    alarma(1, 250);
    imprimirCO2(sensor.getPPM());
  }
  int co2ppm = sensor.getPPM();
  imprimirCO2(co2ppm);
  if(co2ppm >= 1000){
    alarma(4, 500);
  }
  else if(co2ppm >= 800){
    alarma(2, 1000);
  }
}
```



```
aireNuevo();  
delay(5000);  
}
```