UNIVERSIDAD DEL VALLE DE GUATEMALA
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE ELECTRÓNICA, MECATRÓNICA

Nombre: María Alejandra Rodríguez Roldán
Carné: 21620
Curso: Electrónica Digital 2
Sección: 10
Catedrático: Pablo Mazariegos y José Morales
Fecha: 30/08/2023

Proyecto 1 – Señales de PWM

Utilizando el microcontrolador ESP32

Entrega y revisión final: semana del 29 de agosto de 2023

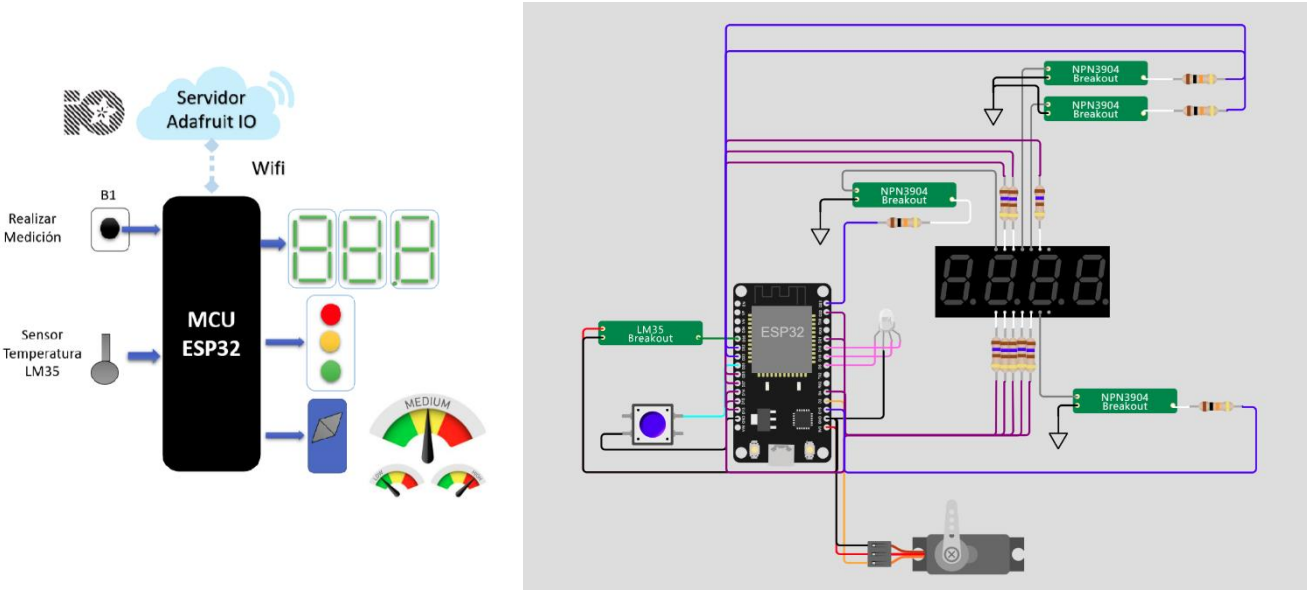

Figura 1. Diagrama y esquemático del circuito con los componentes necesarios

- **Pseudocódigo o diagrama de flujo:**

1. Incluir las librerías necesarias
2. Definir los canales de Adafruit
3. Definir los pines que se utilizarán del ESP32
4. Definir los prototipos de funciones
5. Colocar las variables globales e inicializarlas
6. En el setup colocar:
    a. Inicializar el botón
    b. Configurar el PWM para las leds
    c. Configurar los 7 segmentos de los displays con la librería creada
    d. Configurar el multiplexeo de los displays
    e. Configurar el PWM para el servo
    f. Iniciar la comunicación serial
    g. Conectar con Adafruit
    h. Conectar a los canales de Adafruit
    i. Esperar a que se conecte con Adafruit
    j. Colocar el estado inicial de los leds

7. En el loop colocar:
    a. Llamar a la función que va a controlar a los displays
    b. Leer el estado del botón
    c. Con el estado del botón leído, si el botón está presionado y si el botón anterior no está presionado:
        i. Se activa una bandera
        ii. Si la bandera está activa:
            1. Se lee la temperatura con una función que se crea
            2. Se manda el valor de temperatura a Adafruit
            3. Se niega la bandera para que se pueda presionar el botón
    d. Se modifican el estado de los leds basado en los datos que se mandan desde Adafruit
8. Se crean los prototipos de funciones
    a. Leer el valor del ADC en crudo para la calibración (readADC_Cal)
        i. Se calibra el ADC
        ii. Se convierte el ADC a un valor de voltaje
        iii. Se calcula la temperatura usando el voltaje
        iv. Regresa el valor de temperatura
    b. Configurar los leds con PWM (configurarPWM)
        i. Aquí se coloca como están configurados, por el canal que los controla, la frecuencia y la resolución
    c. Asociar la temperatura con los leds y el servo
        i. Leer el valor del sensor LM35
        ii. Calibrar el ADC, llamar a la función readADC_Cal
        iii. Calcular la temperatura en Celsius
        iv. Actualizar los leds en base a la temperatura leída y mapear el servo a una posición fija
            1. Menor a 37 y el servo en 45 grados
            2. Entre 37 y 37.5 y el servo en 90 grados
            3. Menorr a 37.5 y el servo en 135 grados
    d. Crear la función que va a manejar los datos que envíe Adafruit
        i. Actualizar los leds de acuerdo a los datos recibidos
    e. Función que dice como encender los displays en base a la librería que contiene la configuración de los displays y el encendido de los segmentos de acuerdo al número
        i. Convertir la temperatura a dígitos individuales
        ii. Mostrar la temperatura en los displays

- **Explicaciones del código:**

```
//**************************************************************************
************************
//BE3029 - Proyecto 1 - Digital 2
//Sensor de temperatura
//María Alejandra Rodríguez Roldán
//No. 21620
//**************************************************************************
************************

//**************************************************************************
************************
//Librerías
//**************************************************************************
************************
#include <Arduino.h> //Para trabajar con framework arduino
#include "driver/ledc.h" //Para poder controlar PWM
#include "esp_adc_cal.h" //Librería para utilización de ADC de ESP32
#include "display7.h" //Librería hecha para controlar los displays de 7
segmentos
#include "config.h" //Archivo de librería de Adafruit y el WIFI, aquí está
el usuario y contraseña de Adafruit así como la de internet
```

Se coloca el encabezado del proyecto y se mandan a llamar las librerías necesarias para que el código se ejecute correctamente.

```
//**************************************************************************
************************
// Conexión con ADAFRUIT IO
//**************************************************************************
************************
//Conexión de los canales con Adafruit
AdafruitIO_Feed *tempCanal = io.feed("Sensor"); //Conexión con canal que
recibe los datos de temperatura
AdafruitIO_Feed *LedCanal = io.feed("canalLed"); //Conexión con canal que va
a mandar datos del led
AdafruitIO_Feed *ServoCanal = io.feed("RELOJ"); //Conexión que recibe las
posiciones del servo
```

Se realiza la conexión con los canales creados en la interfaz de adafruit para poder enviar y recibir datos.

```
//************************************************************************
**********************
// Definición de etiquetas
//************************************************************************
**********************
#define LM35_Sensor1 35 //Sensor de temperatura
#define pwmChannel 0 // 16 canales 0-15
#define ledRChannel 1 //Canal de PWM para led rojo
#define ledGChannel 2 //Canal de PWM para led verde
#define ledBChannel 3 //Canal de PWM para led azul
#define freqPWM 5000 //Frencuencia en Hz para el uso de las leds
#define freqPWMS 50   // Frecuencia en Hz (se ajusta a 50Hz para controlar
el servomotor)
#define resolution 8 //Resolución de 1 a 16 bits para la led RGB
#define resolutionS 10 // 1-16 bits de resolución del servo
#define pinLedR 5 //Conexión de pin al ESP 32 para el led rojo
#define pinLedB 18 //Conexión de pin al ESP 32 para el led azul
#define pinLedG 19 //Conexión de pin al ESP 32 para el led verde
#define pinPWM 15 //GPIO para tener la salida del PWM
#define pinPWMS 2    // GPIO 2 para tener la salida del PWM del servo
#define toma_TEMP 25 //Botón para la toma de temperatura

#define DA 27 //Pin A intercontado con multiplexeo para los displays
#define DB 13 //Pin B intercontado con multiplexeo para los displays
#define DC 12 //Pin C intercontado con multiplexeo para los displays
#define DD 22 //Pin D intercontado con multiplexeo para los displays
#define DE 4 //Pin E intercontado con multiplexeo para los displays
#define DF 26 //Pin F intercontado con multiplexeo para los displays
#define DG 21 //Pin G intercontado con multiplexeo para los displays
#define pPunto 14 //Pin punto interconectado con multiplexeo para los
displays
//El pin del punto no lo coloco, debido a que siempre estará encendido en el
mismo display, por lo que lo conecté directamente a voltaje
#define display1 23 //Conexión del display 1, para controlarlo
individualmente
#define display2 32 //Conexión del display 2, para controlarlo
individualmente
#define display3 33 //Conexión del display 3, para controlarlo
individualmente
#define display4 15 //Conexión del display 4, para controlarlo
individualmente
```

Se definen todas las etiquetas que tienen asignado un pin en el ESP32 y que van a funcionar como entrada o salida del circuito. Además, hay etiquetas que ayudan a la configuración del PWM. Aquí se define en donde está el botón, el led RGB, el servo y los displays de 4 segmento, tanto el COM como cada segmento multiplexeado.

```
//*************************************************************************
**********************
// Prototipos de funciones
//*************************************************************************
**********************
uint32_t readADC_Cal(int ADC_Raw); //Función para leer el sensor de
temperatura con ADC de ESP32
void configurarPWM(void); //Función para configurar el PWM de las leds, el
del servo está configurado en el setup
void temperatura_led(void); //Función para indicar la led que tiene que
encender según la temperatura
void displaysvalor(void); //Función para mostrar el valor de temperatura en
los displays
void handleMessage(AdafruitIO_Data *data); //Función para recibir el mensaje
que manda Adafruit
```

En los prototipos de funciones, se crean las funciones necesarias para que el código funcione adecuadamente. En este caso se crearon 4 funciones para leer el ADC, configurar el PWM, leer el valor de temperatura e indicar que hacen las leds y el servo, mostrar el valor de temperatura en los displays y la que dice que hacer con los datos que manda Adafruit.

```
//*************************************************************************
**********************
// Variables Globales
//*************************************************************************
**********************
int LM35_Raw_Sensor1 = 0; //En donde inicia el sensor
float LM35_TempC_Sensor1 = 0.0; //Temperatura en grados centígrados
float Voltage = 0.0; //Variable para utilizar en la función y obtener
temperatura
int botonpresionado; //Almacena el estado del botón
int botonanterior = 1; //Variable bandera para el botón y revisar el estado
anterior
bool bandera = false; //Variable bandera para el botón
bool estadoR = false; //Estado de la led roja en Adafruit
bool estadoB = false; //Estado de la led azul en Adafruit
bool estadoG = false; //Estado de la led verde en Adafruit

//Variables para identificar la temperatura, según la tabla de proyecto
```

```
const float TEMP_LOW = 24.0; //Valor mínimo de temperatura para considerarlo
en estado bajo
const float TEMP_MEDIUM = 25.0; //Valor medio de temperatura para
considerarlo en estado medio
const float TEMP_HIGH = 26.0; //Valor medio de temperatura para considerarlo
en estado alto
int SERVO_LOW = 45; //Posición específica del servo para el estado bajo
int SERVO_MEDIUM = 90; //Posición específica del servo para el estado medio
int SERVO_HIGH = 135; //Posición específica del servo para el estado alto
```

Aquí se colocan las variables globales que se utilizarán en todo el código, además algunas de ellas empiezan en valor 0 y otras ya tiene valores preestablecidos para asegurar que el código cumpla con las condiciones.

```
//*************************************************************************
**********************
// Configuración
//*************************************************************************
**********************
void setup() {
  //Se configura el botón con resistencia interna pull - up
  pinMode(toma_TEMP, INPUT_PULLUP);
  //Se llama la función que configura las leds para controlarlas con PWM
  configurarPWM();
  //Se llama la función que configura los displays de 7 segmentos
  configdisplay7(DA, DB, DC, DD, DE, DF, DG,pPunto);

  //Se configuran los displays como salidas
  pinMode(display1, OUTPUT);
  pinMode(display2, OUTPUT);
  pinMode(display3, OUTPUT);
  pinMode(display4, OUTPUT);

  //Configuración de los transistores del multiplexeo, para saber en que
estado inician, por ser cátodo se coloca LOW
  digitalWrite(display1, LOW);
  digitalWrite(display2, LOW);
  digitalWrite(display3, LOW);
  digitalWrite(display4, LOW);


  //Configuración del PWM para el servo motor
  // Paso 1: Configurar el módulo PWM
  ledcSetup(pwmChannel, freqPWMS, resolutionS);
```

```
  // Paso 2: seleccionar en qué GPIO tendremos nuestra señal PWM
  ledcAttachPin(pinPWMS, pwmChannel);
  //Paso 3: Establecer la posición del servo motor inicialmente
  ledcWrite(0, map(180,0,180,0, 1023));

  //Se inicia la comunicación
  Serial.begin(115200);

  //Se coloca la configuración para conectar con Adafruit
  while(! Serial);
  Serial.print("Conectando con Adafruit IO \n");
  io.connect();

  //Se establece la conexión con el canal del led para recibir los datos de
Adafruit
  LedCanal->onMessage(handleMessage);

  //Se configura para que aparezcan puntitos (........), mientras conecta
con Adafruit
  while(io.status()< AIO_CONNECTED){
    Serial.print(".");
    delay(500);
  }

  //Para recibir los datos del canal de los leds
  LedCanal->get();

  //Imprime el estado en que está con Adafruit
  Serial.println();
  Serial.println(io.statusText());
  //Estados para el encendido y apagado de los leds con Adafruit
  estadoR = false;
  estadoB = false;
  estadoG = false;
}
```

En el setup se realiza toda la configuración de los componentes que están conectados en el circuito. Aquí se coloca que tipo de resistencia tiene el botón y se configura como input. También se configuran los displays y los PWM que van a controlar a las leds como al servo motor. Además, está la comunicación con Adafruit.

```
//*****************************************************************************
************************
// Loop Principal
//*****************************************************************************
************************
void loop() {
  //Se llama la función que va a poner el valor de temperatura en los
displays
  displaysvalor();

  //Leer el estado del botón
  botonpresionado = digitalRead(toma_TEMP);

  //Se establece el condicional de la bandera del botón para evitar rebote
  if (botonpresionado == LOW && botonanterior == HIGH) {
    bandera = true;
    //Si se cumple, se llama a la función para leer la temperatura
    if (bandera) {
      //Función que trabaja con el ADC para obtener un valor del sensor
      temperatura_led();
      delay(1000);
      //Se pone a funcionar Adafruit solamente cuando se presiona el botón
      io.run();
      //Manda los valores de temperatura a Adafruit
      Serial.print("sending ->");
      Serial.println(LM35_TempC_Sensor1);
      tempCanal-> save(LM35_TempC_Sensor1);
      //Desactiva la bandera
      bandera = false;
    }
  }

  //Configuración de los estados de los leds para saber que hacer con los
valores que envía Adafruit
  //Led Rojo
  if(estadoR == true) {
    ledcWrite(ledRChannel, 255);
  }
  //Led Azul
  if(estadoB == true) {
    ledcWrite(ledBChannel, 255);
  }
  //Led Verde
  if(estadoG == true) {
```

```
    ledcWrite(ledGChannel, 255);
  }
}
```

En el loop principal se coloca lo que se va a estar realizando repetitivamente, mientras el microcontrolador ESP32 esté conectado. Aquí se lee el estado del botón, se activa y desactiva la bandera dependiendo si se cumple la condición del estado del botón actual y la del estado del botón anterior. Si la bandera se activa se llama a la función que lee la temperatura y se activa Adafruit para mandar los valores de temperatura. También se colocaron los condicionales para recibir los datos de Adafruit y encender los leds según corresponda.

```
//************************************************************
// Función para configurar módulo PWM de la led RGB
//************************************************************
//Esta es la función que configura el led RGB para que sea controlado por
PWM y así poder variar su brillo
void configurarPWM(void) {
  // Paso 1: Configurar el módulo PWM
  ledcSetup(pwmChannel, freqPWM, resolution);
  ledcSetup(ledRChannel, freqPWM, resolution);
  ledcSetup(ledGChannel, freqPWM, resolution);
  ledcSetup(ledBChannel, freqPWM, resolution);

  // Paso 2: seleccionar en qué GPIO tendremos nuestra señal PWM
  ledcAttachPin(pinPWM, pwmChannel);
  ledcAttachPin(pinLedR, ledRChannel);
  ledcAttachPin(pinLedG, ledGChannel);
  ledcAttachPin(pinLedB, ledBChannel);
}
```

Esta función configura el led RGB para que pueda ser controlado por un canal de PWM. Además, se establecen con las variables ya definidas, así como lo es la resolución y la frecuencia. Además, se indica en que pin está conectado cada color de led.

Parte 1 Sensor de Temperatura.
Diseñe e implemente una rutina en el cual mediante un botón pueda adquirir la señal de un sensor de temperatura LM35 utilizando un ADC.

```
//************************************************************
// Función para leer el sensor de temperatura
//************************************************************
//Esta función utiliza el ADC, para obtener el dato del sensor, se utiliza
ADC_RAW como variable par a
uint32_t readADC_Cal(int ADC_Raw) {
```

```
  esp_adc_cal_characteristics_t adc_chars;
  esp_adc_cal_characterize(ADC_UNIT_1, ADC_ATTEN_DB_11, ADC_WIDTH_BIT_12,
1100, &adc_chars);
  return(esp_adc_cal_raw_to_voltage(ADC_Raw, &adc_chars));
}
```

Esta función se diseñó para leer el valor crudo del sensor de temperatura a través del convertidor analógico a digital siendo el ADC en el ESP32. Luego de leer el valor en crudo, se convierte en una lectura de voltaje en mV. Se utilizan variables de número entero de 32 bits. Además, se utilizan funciones de la librería "esp_adc_cal.h" para almacenar características de calibración.

Parte 2 Semáforo de Temperatura.
Diseñe e implemente tres señales PWM para controlar un LED RGB o en su defecto 3 leds de color Rojo, Verde y Azul en donde dependiendo de la temperatura colocará un color indicador

Temperatura	Color de semáforo
t < 37.0 °C	Verde
37.0 °C < t < 37.5 °C	Amarillo
t < 37.5 °C	Rojo

Parte 3 Reloj del semáforo de temperatura.
Diseñe e implemente una rutina la cual dependiendo del valor de la temperatura obtenida mueva la posición del eje de un servo motor modelando un reloj de temperatura que vaya de la mano con el semáforo de temperatura.


```
//*****************************************************************************
// Función para actualizar el semáforo de leds y el movimiento del servo
//*****************************************************************************
//Esta función utiliza la función readADC_Cal para convertir el dato en
crudo a un valor de temperatura
void temperatura_led(void){
  // Leer el pin LM35_Sensor1 ADC
  LM35_Raw_Sensor1 = analogRead(LM35_Sensor1);
  //Calibrar ADC y tomar el voltaje en mV
  Voltage = readADC_Cal(LM35_Raw_Sensor1);
  // TempC = función con respecto al voltaje
  LM35_TempC_Sensor1 = ((Voltage/4095)*3.25) / 0.01; //De ser necesario se
multiplica por un factor para que lea correctamente la temperatura

  // Imprimir las lecturas, para saber si el sensor funciona
  Serial.print("Lectura de la temperatura = ");
  Serial.print(LM35_TempC_Sensor1);
  Serial.print(" °C \n");
```

```
  //Para el semáforo de temperatura, al estar utilizando un led RGB, los
colores indican lo siguiente:
  //En la guía del proyecto, VERDE es menor a 37.0 °C, AMARILLO es mayor a
37.0 °C y menor a 37.5 °C y ROJO es mayor a 37.5 °C.
  //Con el LED RGB VERDE es menor a 37.0 °C, AZUL es mayor a 37.0 °C y menor
a 37.5 °C y ROJO es mayor a 37.5 °C.
  //Para probar su funcionamiento, los valores, mínimo, máximo y medio
pueden variar debido a que la temperatura ambiental
  //generalmente es de 23 a 26 grados.

  //Primer condicional, este nos dice que si se cumple que la temperatura
tomada es menor a temp_low, se enciende la led verde y el servo está en 45
  if (LM35_TempC_Sensor1 < TEMP_LOW) {
    ledcWrite(ledRChannel, 0);
    ledcWrite(ledGChannel, 255);
    ledcWrite(ledBChannel, 0);
    ledcWrite(pwmChannel, map(SERVO_LOW, 0, 180, 30, 115));
    Serial.print("LED Verde encendido \n");
    //Se configura para enviar el valor de la posición del servo a Adafruit
    Serial.print("sending ->");
    Serial.println(SERVO_LOW);
    ServoCanal-> save(SERVO_LOW);
  //Segundo condicional, este nos dice que si se cumple que la temperatura
tomada es menor a temp_medium, pero mayor a temp_low, se enciende la led
azul y el servo está en 90
  } else if (LM35_TempC_Sensor1 >= TEMP_LOW && LM35_TempC_Sensor1 <
TEMP_MEDIUM) {
    ledcWrite(ledRChannel, 0);
    ledcWrite(ledGChannel, 0);
    ledcWrite(ledBChannel, 255);
    ledcWrite(pwmChannel, map(SERVO_MEDIUM, 0, 180, 30, 115));
    //Se configura para enviar el valor de la posición del servo a Adafruit
    Serial.print("LED Azul encendido \n");
    Serial.print("sending ->");
    Serial.println(SERVO_MEDIUM);
    ServoCanal-> save(SERVO_MEDIUM);
  //Tercer condicional, este nos dice que si se cumple que la temperatura
tomada es menor a temp_high, se enciende la led roja y el servo está en 135
  } else if (LM35_TempC_Sensor1 >= TEMP_MEDIUM && LM35_TempC_Sensor1 <=
TEMP_HIGH) {
    ledcWrite(ledRChannel, 255);
    ledcWrite(ledGChannel, 0);
    ledcWrite(ledBChannel, 0);
```

```
    ledcWrite(pwmChannel, map(SERVO_HIGH, 0, 180, 30, 115));
    //Se configura para enviar el valor de la posición del servo a Adafruit
    Serial.print("LED Rojo encendido \n");
    Serial.print("sending ->");
    Serial.println(SERVO_HIGH);
    ServoCanal-> save(SERVO_HIGH);
  }
}
```

En esta función se combina la parte 2 y 3 del proyecto. Incluso hay parte de Adafruit. Está diseñada para actualizar un semáforo de leds y controlar el movimiento de un servo motor en base a la lectura de temperatura obtenida del sensor. Primero se lee el valor crudo del sensor usando analogRead, luego con la función creada de readADC_Cal se convierte el valor crudo en voltaje. Con el voltaje obtenido se utiliza una función que relaciona el voltaje que está leyendo con la cantidad de datos que puede leer y con el voltaje que debería de estar leyendo, esta ecuación devuelve el valor de temperatura real.

Así mismo, se imprime el valor de temperatura en el monitor serial y luego hay 3 condicionales que controlan el comportamiento de las leds y del servo motor.

- Si la temperatura es menor que TEMP_LOW, se enciende el LED verde y el servo se posiciona a 45 grados.
- Si la temperatura está entre TEMP_LOW y TEMP_MEDIUM, se enciende el LED azul y el servo se posiciona a 90 grados.
- Si la temperatura está entre TEMP_MEDIUM y TEMP_HIGH, se enciende el LED rojo y el servo se posiciona a 135 grados.
- En cada caso, se ajusta la intensidad de los LEDs usando ledcWrite y se mueve el servo motor utilizando el valor de ángulo correspondiente mapeado a un rango adecuado para el servo.

Además, también en los condicionales se codifica que envíe la posición del servo motor a Adafruit.

Parte 4 Despliegue de temperatura.

Diseñe e implemente una rutina para desplegar el valor de temperatura en 3 displays de 7 segmentos. Utilice el punto del primer display para desplegar un decimal. Utilice multiplexeo para poder desplegar en los 3 displays con ayuda de transistores. Tome en cuenta que el circuito puede llegar a cambiar dependiendo si tiene displays de ánodo común o cátodo común.

Creación de la librería para control de los displays:

```
//*************************************************************************
**********************
//Creación de librería de 7 segmentos
//*************************************************************************
**********************
#ifndef __DISPLAY7_H__
```

```
#define __DISPLAY7_H__

//**************************************************************************
**********************
//Librerías
//**************************************************************************
**********************
#include <Arduino.h>

//**************************************************************************
**********************
//Variables globales
//**************************************************************************
**********************
extern uint8_t pinA, pinB, pinC, pinD, pinE, pinF, pinG, pindP;

//**************************************************************************
**********************
//Prototipos de funciones
//**************************************************************************
**********************
//Función para configurar el display de 7 segmentos
void configdisplay7(uint8_t pA, uint8_t pB, uint8_t pC, uint8_t pD, uint8_t
pE, uint8_t pF, uint8_t pG, uint8_t pdP);

//Función para desplegar el valor al 7 segmentos
void valor(uint8_t valor);

//Función para desplegar el punto
void verpunto(boolean punto);

#endif // __DISPLAY7_H__
```

Aquí se crea la librería para el control de los displays de 7 segmentos. Además, se declaran las funciones necesarias y las variables globales. Las variables globales representan los pines de conexión de los displays. También está la declaración de una función para configurarlos, una para ver los valores y una para ver el punto.

Configuración de la librería:

```
//**************************************************************************
**********************
//Librerías
```

```
//**************************************************************************
************************
#include "display7.h"

uint8_t pinA, pinB, pinC, pinD, pinE, pinF, pinG, pindP;

//**************************************************************************
************************
//Funciones
//**************************************************************************
************************
//Función para configurar el display de 7 segmentos
void configdisplay7(uint8_t pA, uint8_t pB, uint8_t pC, uint8_t pD, uint8_t
pE, uint8_t pF, uint8_t pG, uint8_t pdP){
    //Dar valor a la variable global
    pinA = pA;
    pinB = pB;
    pinC = pC;
    pinD = pD;
    pinE = pE;
    pinF = pF;
    pinG = pG;
    pindP = pdP;

    //Configuración de todos los pines como salidas
    pinMode(pinA, OUTPUT);
    pinMode(pinB, OUTPUT);
    pinMode(pinC, OUTPUT);
    pinMode(pinD, OUTPUT);
    pinMode(pinE, OUTPUT);
    pinMode(pinF, OUTPUT);
    pinMode(pinG, OUTPUT);
    pinMode(pindP, OUTPUT);

    //Configuración para apagar todos los pines
    digitalWrite(pinA, LOW);
    digitalWrite(pinB, LOW);
    digitalWrite(pinC, LOW);
    digitalWrite(pinD, LOW);
    digitalWrite(pinE, LOW);
    digitalWrite(pinF, LOW);
    digitalWrite(pinG, LOW);
    digitalWrite(pindP, LOW);
}
```

```
//Función para desplegar el valor al 7 segmentos
void valor(uint8_t valor){
    switch (valor) {
    case 0:
        digitalWrite(pinA, HIGH);
        digitalWrite(pinB, HIGH);
        digitalWrite(pinC, HIGH);
        digitalWrite(pinD, HIGH);
        digitalWrite(pinE, HIGH);
        digitalWrite(pinF, HIGH);
        digitalWrite(pinG, LOW);
        break;
    case 1:
        digitalWrite(pinA, LOW);
        digitalWrite(pinB, HIGH);
        digitalWrite(pinC, HIGH);
        digitalWrite(pinD, LOW);
        digitalWrite(pinE, LOW);
        digitalWrite(pinF, LOW);
        digitalWrite(pinG, LOW);
        break;
    case 2:
        digitalWrite(pinA, HIGH);
        digitalWrite(pinB, HIGH);
        digitalWrite(pinC, LOW);
        digitalWrite(pinD, HIGH);
        digitalWrite(pinE, HIGH);
        digitalWrite(pinF, LOW);
        digitalWrite(pinG, HIGH);
        break;
    case 3:
        digitalWrite(pinA, HIGH);
        digitalWrite(pinB, HIGH);
        digitalWrite(pinC, HIGH);
        digitalWrite(pinD, HIGH);
        digitalWrite(pinE, LOW);
        digitalWrite(pinF, LOW);
        digitalWrite(pinG, HIGH);
        break;
    case 4:
        digitalWrite(pinA, LOW);
        digitalWrite(pinB, HIGH);
        digitalWrite(pinC, HIGH);
```

```
        digitalWrite(pinD, LOW);
        digitalWrite(pinE, LOW);
        digitalWrite(pinF, HIGH);
        digitalWrite(pinG, HIGH);
        break;
    case 5:
        digitalWrite(pinA, HIGH);
        digitalWrite(pinB, LOW);
        digitalWrite(pinC, HIGH);
        digitalWrite(pinD, HIGH);
        digitalWrite(pinE, LOW);
        digitalWrite(pinF, HIGH);
        digitalWrite(pinG, HIGH);
        break;
    case 6:
        digitalWrite(pinA, HIGH);
        digitalWrite(pinB, LOW);
        digitalWrite(pinC, HIGH);
        digitalWrite(pinD, HIGH);
        digitalWrite(pinE, HIGH);
        digitalWrite(pinF, HIGH);
        digitalWrite(pinG, HIGH);
        break;
    case 7:
        digitalWrite(pinA, HIGH);
        digitalWrite(pinB, HIGH);
        digitalWrite(pinC, HIGH);
        digitalWrite(pinD, LOW);
        digitalWrite(pinE, LOW);
        digitalWrite(pinF, LOW);
        digitalWrite(pinG, LOW);
        break;
    case 8:
        digitalWrite(pinA, HIGH);
        digitalWrite(pinB, HIGH);
        digitalWrite(pinC, HIGH);
        digitalWrite(pinD, HIGH);
        digitalWrite(pinE, HIGH);
        digitalWrite(pinF, HIGH);
        digitalWrite(pinG, HIGH);
        break;
    case 9:
        digitalWrite(pinA, HIGH);
        digitalWrite(pinB, HIGH);
```

```
        digitalWrite(pinC, HIGH);
        digitalWrite(pinD, HIGH);
        digitalWrite(pinE, LOW);
        digitalWrite(pinF, HIGH);
        digitalWrite(pinG, HIGH);
        break;

    default:
    Serial.print("Ese valor no se puede mostrar \n");
        break;
    }
}

void verpunto(boolean punto){
    if(punto){
        digitalWrite(pindP, HIGH);
    }else {
        digitalWrite(pindP,LOW);
    }
}
```

Se incluye el archivo header que se creó para la librería de los displays. Aquí se codifican los prototipos de función declarados en el archivo .h. Se retoman las variables globales para los pines de los segmentos de los displays. En la función de configuración se aclara que variables se van a utilizar y que los pines son una salida. También se coloca que al trabajar con displays cátodos, todos van a empezar en estado LOW. Luego está la función de valor, allí se coloca un switch case para así dependiendo del caso/número que se quiera representar se encienden los segmentos correspondientes. Aquí se coloca HIGH en los segmentos que se quieren encender y LOW en los que se mantendrán apagados.

```
//**********************************************************************
//Función para mostrar valor en los displays
//**********************************************************************
//Función para obtener el valor que se desplegará en cada display
void displaysvalor(void){
  //La temperatura se multiplica por 100 para no tener decimales
  int temperatura = LM35_TempC_Sensor1 * 100;
  //Se obtiene cada número por separado
  int unidad = (temperatura/1) %10;
  int decena = (temperatura/10) %10;
  int decimal = (temperatura/100) % 10;
  int centena = (temperatura/1000) %10;

  //Se enciende el display correspondiente y con la función valor se indica
que segmentos encender
```

```
  digitalWrite(display1, HIGH);
  digitalWrite(display2, LOW);
  digitalWrite(display3, LOW);
  digitalWrite(display4, LOW);
  valor(centena);
  verpunto(0); //Para no prender el punto en este display
  delay(5);

  digitalWrite(display1, LOW);
  digitalWrite(display2, HIGH);
  digitalWrite(display3, LOW);
  digitalWrite(display4, LOW);
  valor(decimal);
  verpunto(1); //Se prende el punto en este display
  delay(5);

  digitalWrite(display1, LOW);
  digitalWrite(display2, LOW);
  digitalWrite(display3, HIGH);
  digitalWrite(display4, LOW);
  valor(decena);
  verpunto(0);
  delay(5);

  digitalWrite(display1, LOW);
  digitalWrite(display2, LOW);
  digitalWrite(display3, LOW);
  digitalWrite(display4, HIGH);
  valor(unidad);
  verpunto(0);
  delay(5);
}
```

En esta función que se encuentra en el main.cpp, el código principal, se hace uso de la librería de displays creada anteriormente. Aquí primero se colocan las ecuaciones utilizadas para obtener cada dígito de la temperatura por separado y poder colocarlo en un display. Al tener los valores por separado, ya se puede encender el display correspondiente e imprimir el valor con los segmentos correctos. Además, se utiliza la función de ver punto, para solo activar el punto decimal correspondiente.

Parte 5 Dashboard Adafruit IO.

Diseñe una interfaz en los servidores de Adafruit IO, en donde pueda mostrar el resultado de los valores obtenidos del sensor de temperatura.

Para utilizar Adafruit se crea un archivo de configuración, para que sepa el microcontrolador a que se tiene que conectar y que internet utilizar.

```
/************************ Adafruit IO Config
*******************************/

// visit io.adafruit.com if you need to create an account, or if you need
your Adafruit IO key.
#define IO_USERNAME  "aler21620"
#define IO_KEY       "aio_PAbV49Ob7GrE6NCFXWtBZ6VrqEZc"

/******************************* WIFI
*************************************/
#define WIFI_SSID "iPhone de Alejandra"
#define WIFI_PASS "digital2"

// comment out the following lines if you are using fona or ethernet
#include "AdafruitIO_WiFi.h"

AdafruitIO_WiFi io(IO_USERNAME, IO_KEY, WIFI_SSID, WIFI_PASS);
```

```
//*****************************************************************************
//Función para LED en Adafruit
//*****************************************************************************
//Esta función ayuda a manejar los datos que adafruit envía al
microcontrolador
void handleMessage(AdafruitIO_Data *data) {
  //Se corre Adafruit
  io.run();
  //Recibe el dato
  Serial.print("recieved <-");
  Serial.println(data->value());
  //Se crean condicionales para saber que hacer con el dato recibido
  if(*data->value() == '1'){
    estadoR = true;
  } else{
    estadoR = false;
  }
  if(*data->value() == '2'){
```

```
    estadoB = true;
  } else{
    estadoB = false;
  }
  if(*data->value() == '4'){
    estadoG = true;
  } else{
    estadoG = false;
  }
}
```

Esta función se comunica con Adafruit para recibir los datos que manda para encender el led RGB.

Parte 6 ESP32 WIFI.
Implemente la comunicación entre el ESP32 y los servidores de Adafruit IO, utilizando WIFI. El ESP32 deberá ser capaz de mandar los valores obtenidos del sensor de temperatura y el reloj de temperatura.

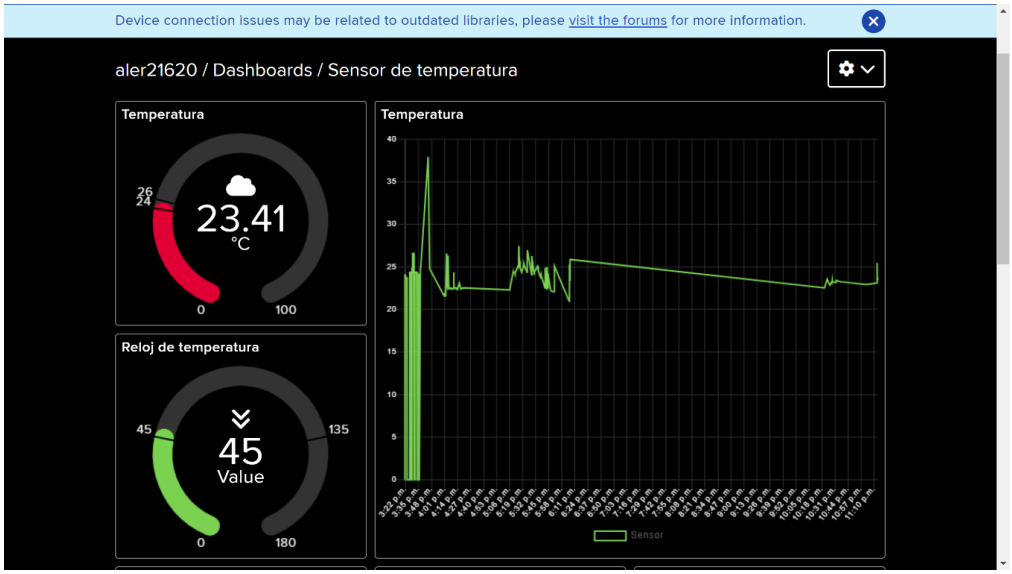

Figura 2. Dashboard de Adafruit

- **Link a dashboard de Adafruit:** https://io.adafruit.com/aler21620/dashboards/sensor-de-temperatura
- **Link al canal que recibe los datos de temperatura:** https://io.adafruit.com/aler21620/feeds/sensor-de-temperatura
- **Link al canal que recibe los datos de la posición del servo:** https://io.adafruit.com/aler21620/feeds/reloj
- **Link a canal que envía los datos para las leds:** https://io.adafruit.com/aler21620/feeds/canalled

Circuito armado:

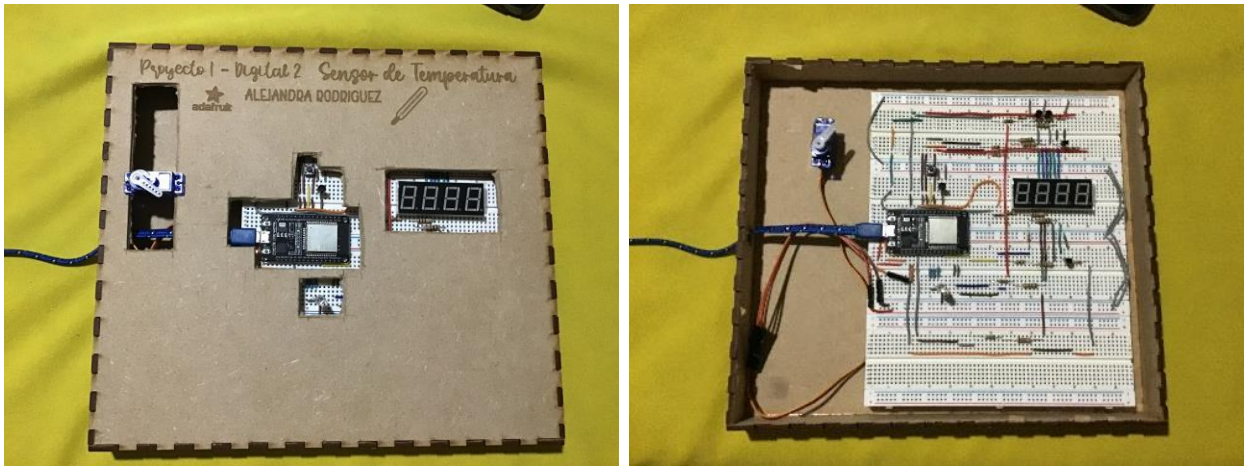

Figura 3. Presentación y circuito armado

Entregables:

- **Link a vídeo de YouTube con el funcionamiento y explicación:** https://www.youtube.com/watch?v=ZGvQB_fnlaQ&feature=youtu.be
- **Link a GitHub, para ver la carpeta de Visual Code completa y los commits realizados a lo largo del proyecto:** https://github.com/aler21620/Proyecto1-Digital2-21620