Trabajo Práctico N° 8

Módulo 3

Visualizadores – Protocolos – Interfaces de E/S

Profesor

Jorge Elias Morales

Integrantes

Marcos Bordón Rios - Marcos-BR-03

Fernando Gimenez Coria - FerCbr

Karina Jazmin Barbero - karina-lolis

Nicolás Barrionuevo - NicolasBa27

Macarena Aylen Carballo - MacarenaAC

Raul Jara - r-j28

Diego Ezequiel Ares - diegote7

Juan Diego Gonzaléz Antoniazzi - JDGA1997

Fecha de entrega

Viernes 18 de Octubre 2024

Índice

Ejercicio Nº1

1. Implementen una simulación de un Sistema de Riego Inteligente en Wokwi o Proteus, utilizando ESP32 ó ARDUINO con los siguientes sensores y especificaciones:

Proyecto ABP: Sistema de riego automático

Descripción General

Este proyecto tiene como objetivo desarrollar un sistema de riego automático que permita tomar información de varios sensores y actuar sobre una bomba y una electroválvula.

El sistema está diseñado en torno a un microcontrolador **ESP32**, utiliza varios sensores para medir la humedad del suelo, detectar lluvia, medir el nivel de un tanque de agua. Además, cuenta con una interfaz de usuario a través de un **visualizador LCD** de 2x20 caracteres, que muestra el estado del sistema de forma cíclica.

Componentes:

1. Sensores:
    - **YL-69**: Sensor de humedad del suelo (resistivo).
    - **V1.2**: Sensor de humedad del suelo (capacitivo).
    - **YL-83**: Sensor de Lluvia.
    - **HC-SR04**: Sensor de ultrasonido, para medir el nivel del tanque de reserva.
    - **Display LCD de 2x20 I2C**: Visualización de los valores de los sensores y los actuadores.
2. Periféricos de entrada
    - **Touchpads capacitivos**: Para navegar por el menú del LCD.

3. Actuadores

   - **Salida a relé optoacoplada para bomba**: Acciona el encendido de una bomba aislando al controlador de los circuitos de potencia.
   - **Salida a relé optoacoplada para electroválvula**: Accionamiento de una electroválvula mediante relé, lo que permite manejar distintos tipos de válvulas y tensiones sin que el controlador tenga contacto directo con la etapa de potencia.

Estructura del Proyecto

```
lib/
 YL-69/
  ├── sensorYL-69.cpp
  ├── sensorYL-69.h
 YL-83/
  ├── sensorYL-83.cpp
  ├── sensorYL-83.h
 Hum_Cap/
  ├── sensor_hum_cap.cpp
  ├── sensor_hum_cap.h
 HC-SR04/
  ├── sensorHC-SR04.cpp
  ├── sensorHC-SR04.h
 LCDDisplay/
  ├── LCDDisplay.cpp
  ├── LCDDisplay.h
 Relay/
  ├── Salida_relay.cpp
  ├── Salida_relay.h
 Touch/
  ├── TouchController.cpp
  ├── TouchController.h

src/
  ├── main.cpp
```

Estructura del Código

1. Archivo `main.cpp`:

Este es el archivo principal que inicializa los sensores, el LCD, los touchpads y los actuadores. El bucle principal realiza lecturas de los sensores, actualiza el menú del LCD, y controla los actuadores en función de la humedad, la detección de lluvia y el nivel de agua de reserva de acuerdo a la siguiente lógica:

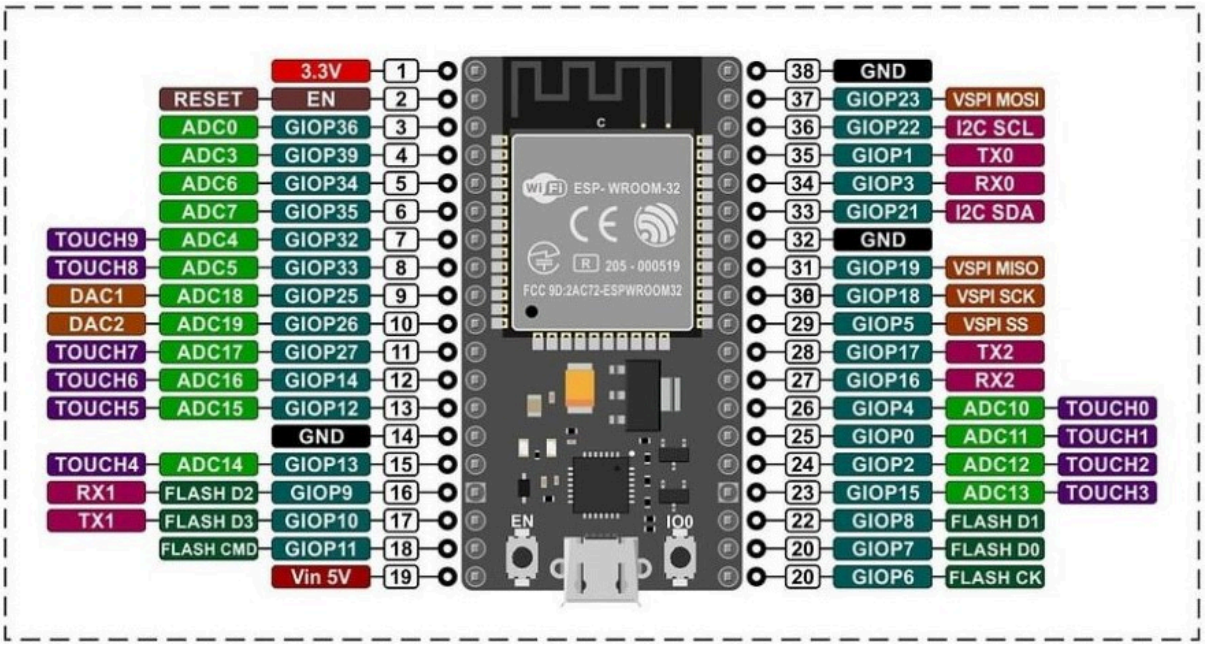
El controlador toma lectura de todos los sensores, dos de humedad de suelo que permite el manejo de dos zonas independientes, uno de lluvia, y uno de ultrasonido para medir el nivel de reserva de agua.

Además, se cuenta con una salida para bomba y tres accionamientos de electroválvulas, una para el ingreso de agua al tanque, y una para cada zona de riego.

- **Si la humedad en una zona cae por debajo de un umbral establecido**, se enciende la bomba y se activa la electroválvula que corresponde a esa zona de riego.
- **Si se detecta un nivel de reserva de agua inferior al mínimo establecido**, se activa la electroválvula correspondiente al ingreso de agua al tanque desde la red.
- **Si se detecta lluvia**, se interrumpe cualquier ciclo de riego hasta que deje de llover.

Configuración de Pines:

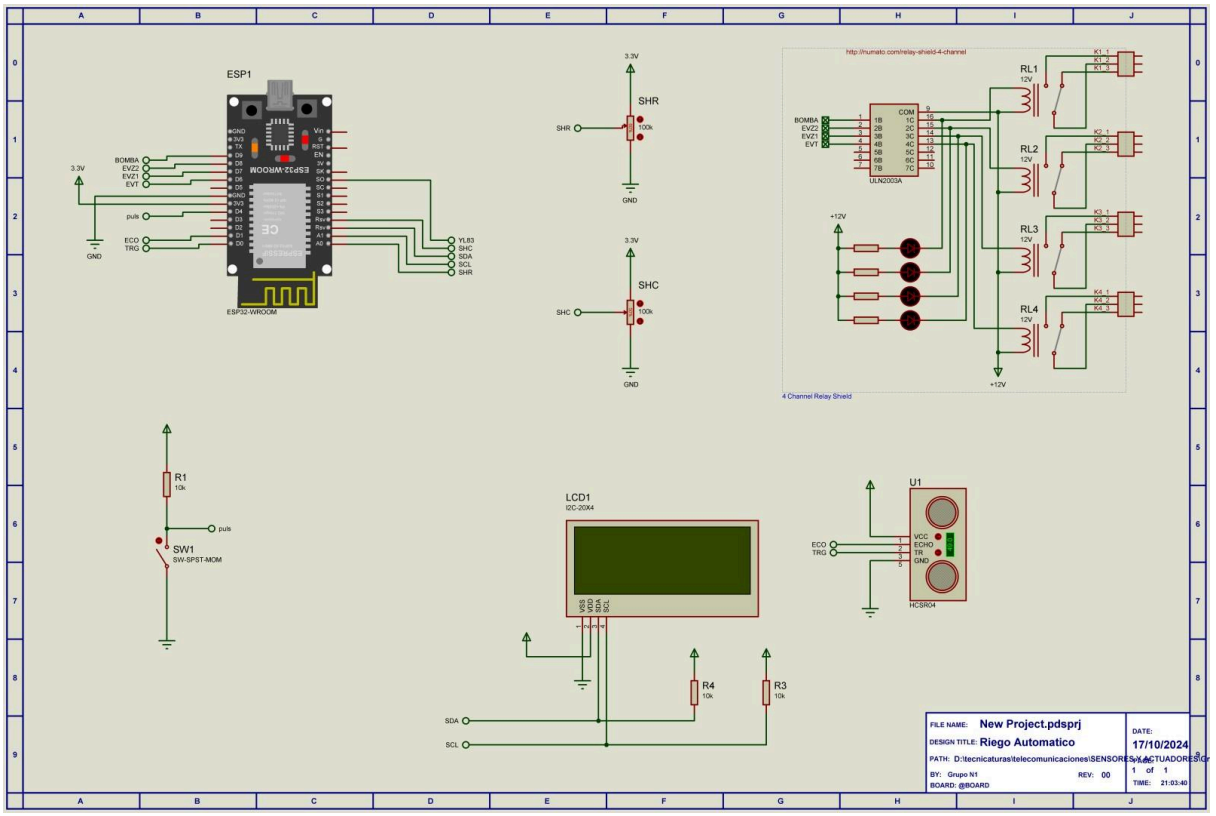
A continuación se muestra un diagrama con la distribución de pines en el modulo ESP32-Wroom de **Nodemcu**



Asignación

- **YL-69** (sensor analógico): Pin ADC0 (GIOP36 / pata 3)
- **Hum_Cap** (sensor analogico): Pin ADC5 (GIOP33 / pata 8)
- **YL-83** (entrada digital): Pin GIOP12 / pata 13
- **HC-SR04** (1 entrada digital, 1 salida digital): Pin GIOP39 / pata 4 (Trigg); Pin GIOP34 / pata 5 (Echo)
- **touchpad capacitivo**: Pin TOUCH0(GIOP4 / pata 26)

- **Salidas relay**: Pin GIOP25 / pata 9; Pin GIOP26 / pata 10; Pin GIOP27 / pata 11; Pin GIOP14 / pata 12
- **Dispositivos I2C**: Pin I2C SDA / pata 33 y Pin I2C SCL / pata 36



Inicialización de Sensores y Periféricos:

- En el `setup()`, se inicializan todos los sensores y periféricos (LCD, touchpads, RELAY).
  La primera pantalla del menú se muestra en el LCD.

Lógica del Menú:

- La función `navegarMenu()` controla la navegación por el menú del LCD mediante el touchpad capacitivo. El **touchpad TOUCH0** avanza al siguiente menú, cada vez que es pulsado.

Bucle Principal (`loop()`):

- El bucle principal realiza las siguientes tareas:
  1. **Navegación en el Menú:** Comprueba si se ha tocado el touchpad y actualiza el menú en consecuencia.
  2. **Lectura de Sensores:** Lee los valores de los sensores y actualiza el estado del sistema.

3. **Actualización del LCD:** Muestra los valores en el LCD, dependiendo de la pantalla seleccionada en el menú.
4. **Control de los actuadores:** Ejecuta la lógica descrita arriba actuando sobre los relé en función del estado del sistema.

2. Clase `sensorYL-69` (Sensor de humedad de suelo resistivo):

Descripción:

Esta clase maneja el sensor de humedad de suelo **YL-69**, que mide la humedad del suelo mediante un fenómeno resistivo.

Métodos:

- **begin():** Inicializa el sensor YL-69.
- **leerHumedad_Res():** Lee y devuelve un valor de humedad en porcentaje.

Uso:

El sensor YL-69 se inicializa en el `setup()` y se llama a `leerHumedad_Res()` en el bucle principal para obtener la humedad del suelo y mostrarla en el menú correspondiente del LCD.

3. Clase `sensorYL-83` (Sensor de Lluvia):

Descripción:

Esta clase gestiona el sensor de detección de lluvia **YL-83**, que básicamente detecta si está lloviendo.

Métodos:

- **begin():** No requiere inicialización especial, pero se incluye para mantener la consistencia en la estructura.
- **detectarLluvia():** Lee la entrada digital correspondiente al sensor, y devuelve un valor bool.

Uso:

El sensor ML8511 se inicializa en el `setup()` y `detectarLluvia()` se llama en el bucle principal. Se muestra en el LCD en la sección correspondiente del menú.

4. Clase `sensor_hum_cap` (Sensor de humedad de suelo capacitivo)

Descripción:

Esta clase maneja el sensor de humedad de suelo capacitivo **V 1.2**, que mide la humedad del suelo mediante un fenómeno capacitivo.

Métodos:

- **begin():** Inicializa el sensor.
- **leerHumedad_Cap():** Lee y devuelve un valor de humedad en porcentaje.

Uso:

El sensor se inicializa en el `setup()` y se llama a `leerHumedad_Cap()` en el bucle principal para obtener la humedad del suelo y mostrarla en el menú correspondiente del LCD.

5. Clase `sensorHC-SR04` (Sensor ultrasónico por eco):

Descripción:

Esta clase maneja el sensor de humedad de suelo **HC-SR04**, que mide el nivel de reserva de agua en un tanque por medio de eco.

Métodos:

- **begin():** Inicializa el sensor HC-SR04.
- **leerNivel():** Lee y devuelve un nivel de reserva de agua porcentaje.

Uso:

El sensor HC-SR04 se inicializa en el `setup()` y se llama a `leerNivel()` en el bucle principal para obtener la humedad del suelo y mostrarla en el menú correspondiente del LCD.

6. Clase `LCDDisplay` (Pantalla LCD de 2x16):

Descripción:

Esta clase maneja la pantalla LCD I2C de 2x20 caracteres. Proporciona la capacidad de mostrar diferentes menús con los valores de los sensores y el estado de los actuadores.

Métodos:

- **begin():** Inicializa la pantalla LCD y enciende la retroiluminación.
- **displayMenu(int menu):** Muestra el nombre del menú actual en la primera línea del LCD.
- **updateMenu(int menu, ...):** Actualiza el contenido de la segunda línea del LCD con los valores de los sensores correspondientes al menú seleccionado.

Menús Disponibles:

1. **Menú 1: Humedad en el suelo**
2. **Menú 2: Riego**

3. **Menú 3: Reserva de agua**

7. Clase `Salida_relay` (Controlador de relés):

Descripción:

Esta clase gestiona las cuatro salidas optoacopladas para el manejo de la bomba y las tres electroválvulas.

Métodos:

- **begin():** Inicializa las salidas.
- **actualizar_salidas():** se ejecuta esta función para cambiar el estado de las cuatro salidas que posee el controlador, basado en el valor de una variable global en de tipo Array que contiene cuatro valor booleanos correspondientes a: Salidas[0]->BOMBA
  Salidas[1]->EV_zona_1
  Salidas[2]->EV_zona_2
  Salidas[3]->EV_ent_tanque

Uso:

Las salidas se inicializan en el `setup()`, luego en el bucle principal se actualiza los valores del Array `Salidas` y se llama a `actualizar_salidas()` para cambiar el estado de las mismas. Además este mismo Array se consulta para mostrar en el menú correspondiente del LCD el estado de cada salida.

8. Clase `TouchController` (Controlador de Touchpads):

Descripción:

Esta clase gestiona las tres superficies metálicas conectadas a las entradas capacitivas TOUCH0, TOUCH1 y TOUCH2 del ESP32, que permiten la navegación por el menú del LCD.

Métodos:

- **begin():** Inicializa las entradas capacitivas (no requiere configuración adicional).
- **checkTouch():** Comprueba si alguna de las superficies táctiles ha sido activada y devuelve un código para avanzar o retroceder en el menú.

Navegación:

- **TOUCH0 (Next):** Avanza al siguiente menú.

CONFIGURACIONES

Librerías necesarias

Para que el programa funcione es necesario instalar las siguientes librerías en el IDE:

- **LiquidCrystal_I2C.h**

Para el caso de PlatformIO la mismas pueden instalarse desde el gestor de librerías buscando el nombre de cada una, y haciendo click en `install`.

2. Configuración de Hardware

Los sensores se conectan al ESP32-Wroom según las siguientes especificaciones:

- YL-69(Analógico):
  - **VCC:** 3.3V
  - **GND:** GND
  - **Salida de señal:** GPIO36 (ADC1_CH0)
- Hum_Cap(Analógico):
  - **VCC:** 3.3V
  - **GND:** GND
  - **Salida de señal:** GPIO33 (ADC5_CH0)
- YL-83(digital):
  - **VCC:** 3.3V
  - **GND:** GND
  - **Salida de señal:** GPIO12
- HC-SR04(digital):
  - **VCC:** 3.3V
  - **GND:** GND
  - **Entrada de señal Trigger:** GPIO39
  - **Salida de señal Echo:** GPIO34
- Display LCD (I2C):
  - **VCC:** 3.3V
  - **GND:** GND
  - **SDA:** GPIO21
  - **SCL:** GPIO22

Funcionamiento del Menú

Menú Navegable

El sistema tiene un menú navegable que muestra que consta de tres pantallas:

1. **Menú 1: Humedad en el suelo**
    - Muestra la humedad medida en las dos zonas Z1 y Z2.
    - Si llueve se muestra el mensaje: Lluvia, lectura interrumpida.
2. **Menú 2: Riego**
    - Muestra el estado de la Bomba de riego y las Electroválvulas de cada zona EVZ1 y EVZ2.
    - Si llueve se muestra el mensaje: Lluvia, riego interrumpido.
3. **Menú 3: Reserva de agua**
    - Muestra el nivel de reserva de agua en porcentaje y el estado de la electroválvula EV_ent_tanque.

El usuario puede navegar entre estos menús tocando la superficie conectada a **TOUCH0** (siguiente menú).

3. Compilación y Carga del Código

Seguir los siguientes pasos:

1. Abrir el proyecto en el IDE PlatformIO o en VScode con Platformio integrado.
2. Seleccionar la placa **nodemcu-32s** en el menú **Tools > Board**.
3. Seleccionar el puerto correcto donde está conectado el ESP32.
4. Compilar y cargar el código en el ESP32.

4. Monitoreo de Datos

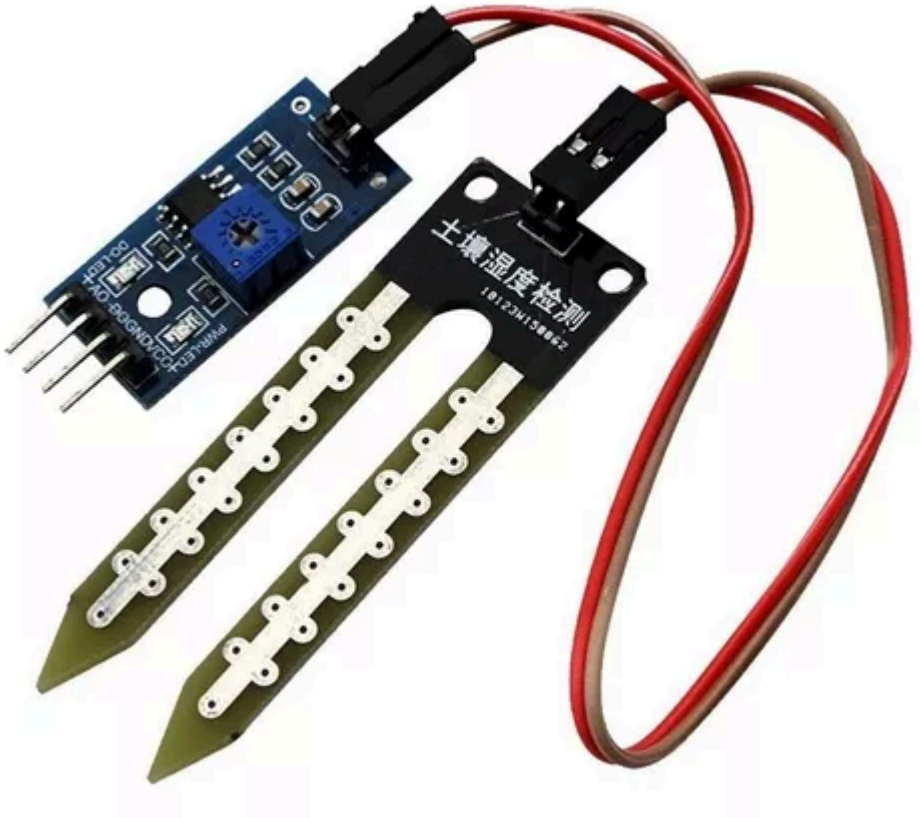
Abrir el **Monitor Serie** en el IDE y configurarlo a **115200 baudios** para observar las lecturas de los sensores. Los datos se mostrarán en tiempo real y se indicará si alguna lectura es inválida debido a valores fuera de los rangos definidos.

Futuras Expansiones

Este proyecto está diseñado para ser fácilmente escalable.
Se pueden agregar más sensores o funcionalidades, como enviar los datos a una plataforma en la nube o a un servidor remoto.
El código modular facilita la incorporación de nuevas características sin afectar la estructura existente.

Licencia

A) Sensor de humedad del suelo Resistivo. YL-69, HL-69 ó FC-28



Estructura del Sensor de Humedad Resistivo

Archivo de cabecera "sensor_humedad.h"

```
#ifndef SENSOR_HUMEDAD_H
#define SENSOR_HUMEDAD_H

#include <Arduino.h>

class Sensor_YL69{
	private:
	int sensorPin; // se defino el pin que voy a usar para la entrada del sensor

	public:
	// construtor recibe el numero de pin
	Sensor_YL69(int pin_yl69);

	// metodo de inicializacion
	void begin();

	// lectura de la humedad de suelo
	float leerHumedad();
};

#endif
```

Métodos públicos:

- **`Sensor_YL69(int pin_yl69)`:** Constructor de la clase. Recibe como parámetro el número de pin al que está conectado el sensor.
- **`void begin()`:** Método para inicializar el sensor. En este caso, no requiere configuración adicional de pines.
- **`float leerHumedad()`:** Método para leer el valor de humedad del suelo. Devuelve un valor flotante que representa el porcentaje de humedad.

Archivo fuente "sensor_humedad.cpp"

```
#include "sensor_humedad.h"

// contructor para inicializar el pin de entrada
Sensor_YL69::Sensor_YL69(int pin_yl69){
	sensorPin = pin_yl69;
}
```

```
// Implementación de la función para inicializar el sensor
void Sensor_YL69::begin(){
// no se requuiere configuracion adicional de pines
}

// Implementación de la función para leer el valor de humedad
float Sensor_YL69::leerHumedad(){
        int valorLectura = analogRead(sensorPin); // leer el valor analogico
        // convertir el valor analogico en un parcentaje de humedad (0 - 100%)
        float porcentajedeHumedad = (float)map(valorLectura, 0, 4095, 0, 100);
        return porcentajedeHumedad;
}
```

Archivo principal "main.cpp"

```
#include <Arduino.h>
#include "sensor_humedad.h"  // Incluir el archivo de encabezado del sensor

// Definir el pin analógico donde está conectado el sensor YL-69
#define PIN_SENSOR_HUMEDAD 34

// Crear una instancia del sensor
Sensor_YL69 sensorHumedad(PIN_SENSOR_HUMEDAD);

void setup() {
        // Iniciar la comunicación serie para monitorear los valores en el monitor serie
        Serial.begin(115200);

        // Inicializar el sensor
        sensorHumedad.begin();
}

void loop() {
        // Leer la humedad del suelo
        float humedad = sensorHumedad.leerHumedad();

        // Mostrar el valor de humedad en el monitor serie
        Serial.print("Humedad del suelo: ");
        Serial.print(humedad);
        Serial.println("%");

        // Esperar 2 segundos antes de la siguiente lectura
        delay(2000);
```

}

El programa está estructurado en:

- **Archivo de cabecera:** Define la clase `Sensor_YL69` con sus métodos públicos para interactuar con el sensor.
- **Archivo fuente:** Implementa los métodos de la clase `Sensor_YL69` para leer el valor del sensor y convertirlo a porcentaje de humedad.
- **Archivo principal:** Crea una instancia del sensor, lo inicializa y en el bucle principal lee la humedad del suelo y la muestra en el monitor serial.

B) Sensor de humedad del suelo Capacitivo. V1.2

Estructura del Sensor de Humedad del Suelo

1. Archivo de cabecera "sensor_humedad.h"

#ifndef SENSOR_HUMEDAD_H

```
#define SENSOR_HUMEDAD_H

#include <Arduino.h>

class SensorHumedad {
  private:
        int pin;  // Pin analógico donde está conectado el sensor

  public:
        // Constructor que recibe el pin donde está conectado el sensor
        SensorHumedad(int analogPin);

        // Método de inicialización
        void begin();

        // Método para leer la humedad del suelo (retorna un valor en porcentaje)
        float leerHumedad();
};

#endif
```

Métodos públicos:

- **`SensorHumedad(int analogPin)`:** Constructor que recibe el pin analógico donde está conectado el sensor.
- **`void begin()`:** Método de inicialización.
- **`float leerHumedad()`:** Método para leer la humedad del suelo y retorna un valor en porcentaje.

2. Archivo fuente "sensor_humedad.cpp"

```
#include "sensor_humedad.h"

// Constructor que inicializa el pin del sensor
SensorHumedad::SensorHumedad(int analogPin) {
  pin = analogPin;
}

// Método begin: no se requiere configuración especial, pero lo incluimos por consistencia
void SensorHumedad::begin() {
  // No se requiere configuración especial para pines analógicos
}
```

```
// Método para leer la humedad del suelo y convertirla en un porcentaje
float SensorHumedad::leerHumedad() {
  int valorLectura = analogRead(pin);  // Leer el valor analógico
  // Convertir el valor analógico a un porcentaje de humedad (0 - 100%)
  float porcentajeHumedad = map(valorLectura, 0, 4095, 0, 100);
  return porcentajeHumedad;
}
```

- **Definición del constructor `SensorHumedad(int analogPin)`:** Inicializa el pin del sensor.
- **Definición del método `begin()`:** No requiere configuración especial para pines analógicos.
- **Definición del método `leerHumedad()`:** Lee el valor analógico del pin y lo convierte a un porcentaje de humedad utilizando la función `map()`.

3. Archivo principal "main.cpp"

```
#include <Arduino.h>
#include "sensor_humedad.h"   // Incluir la cabecera del sensor de humedad del suelo

// Crear una instancia del sensor de humedad en el pin GPIO 34 (pin analógico)
SensorHumedad sensorSuelo(34);

void setup() {
  Serial.begin(115200);  // Inicializamos la comunicación serie
  sensorSuelo.begin();   // Inicializamos el sensor
}

void loop() {
  // Leer el porcentaje de humedad del suelo
  float humedad = sensorSuelo.leerHumedad();

  // Mostrar el valor de humedad en el monitor serie
  Serial.print("Humedad del suelo: ");
  Serial.print(humedad);
  Serial.println("%");

  // Esperar 1 segundo antes de la siguiente lectura
  delay(1000);
}
```

El programa está estructurado en:

- **`void setup()`:**

- Inicializa la comunicación serie.
- Crea una instancia del sensor de humedad en el pin GPIO 34 (pin analógico).
- Inicializa el sensor.

- `void loop()`:
  - Lee el porcentaje de humedad del suelo utilizando el método `leerHumedad()`.
  - Muestra el valor de humedad en el monitor serie.
  - Espera 1 segundo antes de la siguiente lectura utilizando la función `delay()`.

C) Sensor de lluvia Resistivo YL-83

Propósito de la práctica

En esta práctica, se utilizará un sensor de lluvia **YL-83** para detectar la presencia de agua (lluvia) mediante un **ESP32**. El objetivo es leer el estado del sensor (lluvia/no lluvia) y mostrar el resultado en el monitor serie.

Estructura de archivos

El proyecto se organizará en tres archivos principales:

- **`sensor_yl83.h`**: Archivo de cabecera que define la clase `SensorYL83`.

- **`sensor_yl83.cpp`**: Archivo que implementa los métodos de la clase `SensorYL83`.
- **`main.cpp`**: Archivo principal que inicializa el sensor y gestiona el bucle principal del programa.

Estructura del proyecto:

```
├── include
│   └── sensor_yl83.h # Archivo de cabecera para el sensor de lluvia YL-83
├── src
│   ├── main.cpp # Código principal donde se gestiona la lógica del programa
│   └── sensor_yl83.cpp # Implementación de la clase del sensor de lluvia YL-83
└── platformio.ini # Archivo de configuración de PlatformIO
```

Descripción de cada archivo y su función

`sensor_yl83.h` (Archivo de cabecera)

Este archivo contiene la definición de la clase `SensorYL83`. Su propósito es declarar los atributos y métodos necesarios para interactuar con el sensor de lluvia.

```
#ifndef SENSOR_YL83_H
#define SENSOR_YL83_H

#include <Arduino.h>

class SensorYL83 {
  private:
    int pin;  // Pin digital donde está conectado el sensor

  public:
    // Constructor que recibe el pin donde está conectado el sensor
    SensorYL83(int digitalPin);

    // Método de inicialización (aunque no requiere configuración especial)
    void begin();

    // Método para detectar lluvia (retorna true si está lloviendo)
```

```
    bool detectarLluvia();
};

#endif
```

Explicación:

`#ifndef, #define y #endif:` Estas directivas evitan la inclusión múltiple del archivo (es decir, que se cargue el archivo más de una vez).

`Clase SensorYL83:` Se define una clase que encapsula el comportamiento del sensor YL-83.

Atributos y métodos:

int pin: Almacena el pin GPIO donde está conectado el sensor.

SensorYL83(int digitalPin): Constructor que inicializa el sensor con el pin asignado.

begin(): Configura el pin como entrada para que pueda recibir el estado del sensor.

detectarLluvia(): Lee el valor del pin digital y devuelve true si el sensor detecta lluvia.

`sensor_yl83.cpp` (Implementación de la clase)

Este archivo contiene el código que define cómo funcionan los métodos declarados en el archivo de cabecera.

```
#include "sensor_yl83.h"


// Constructor que inicializa el pin del sensor
SensorYL83::SensorYL83(int digitalPin) {
  pin = digitalPin;
}


// Método begin: configura el pin como entrada
void SensorYL83::begin() {
```

```
  pinMode(pin, INPUT);
}

// Método para detectar si está lloviendo
bool SensorYL83::detectarLluvia() {
  // Leer el valor digital del sensor (HIGH = no lluvia, LOW = lluvia)
  return digitalRead(pin) == LOW;  // LOW indica que está lloviendo
}
```

Explicación:

Constructor

`SensorYL83::SensorYL83(int digitalPin)`: Almacena el pin donde está conectado el sensor.

`begin()`: Configura el pin como entrada digital usando la función pinMode().

`detectarLluvia()`: Usa digitalRead() para leer el estado del pin digital. Si el pin está en LOW, significa que el sensor detecta lluvia, por lo tanto, devuelve true.

`main.cpp` (Código principal)

Este archivo es el punto de entrada del programa. Inicializa el sensor y contiene el bucle principal que lee continuamente el estado del sensor y muestra los resultados en el monitor serie.

```
#include <Arduino.h>
#include "sensor_yl83.h"   // Incluimos la cabecera del sensor de lluvia

// Instanciamos el sensor de lluvia en el pin GPIO 12
SensorYL83 rainSensor(12);

void setup() {
  Serial.begin(115200);  // Inicializamos la comunicación serie
  rainSensor.begin();    // Inicializamos el sensor de lluvia
}
```

```
void loop() {
  // Detectar si está lloviendo
  bool estaLloviendo = rainSensor.detectarLluvia();

  if (estaLloviendo) {
    Serial.println("Está lloviendo");
  } else {
    Serial.println("No está lloviendo");
  }

  delay(1000);  // Pausa de 1 segundo
}
```

Explicación:

`SensorYL83 rainSensor(12);` Creamos una instancia de la clase SensorYL83 y especificamos el pin GPIO 12 como el pin donde está conectado el sensor.

`Serial.begin(115200);` Inicializa la comunicación serie a 115200 baudios para poder mostrar los mensajes en el monitor serie.

`rainSensor.begin();` Inicializa el sensor, configurando el pin como entrada digital.

`loop();` Lee continuamente el estado del sensor cada segundo y muestra en el monitor serie si está lloviendo o no.

Conclusión

Esta práctica introduce a los sensores digitales en IoT, en este caso utilizando el sensor de lluvia YL-83. El código está organizado de manera modular para que sea fácil de entender, mantener y escalar. Si en el futuro se quiere agregar más sensores, simplemente se puede replicar la estructura de archivos.

2. Salida a una pantalla LCD de 16x2 ó 20x4.

3. Comando automático de una Bomba que encienda o apague el riego, y si es posible agregarle una Electroválvula. Usar salidas por relé u optoacopladas.

Archivo de cabecera "bomba_riego.h"

```
#ifndef BOMBA_RIEGO_H
#define BOMBA_RIEGO_H

#include <Arduino.h>

class BombaRiego {
  private:
        int pinRelé;   // Pin donde está conectado el relé

  public:
        // Constructor que recibe el pin del relé
        BombaRiego(int pin);

        // Método de inicialización
        void begin();

        // Método para encender la bomba/electroválvula
        void encender();

        // Método para apagar la bomba/electroválvula
        void apagar();
};

#endif
```

- `#include <Arduino.h>`: Incluye la biblioteca principal de Arduino para acceder a las funciones básicas.

- `class BombaRiego`: Define la clase "BombaRiego" que encapsula la funcionalidad de la bomba.

Métodos públicos:

- `BombaRiego(int pin)`: Constructor que recibe el pin donde está conectado el relé y lo almacena en la variable privada `pinRelé`.
- `void begin()`: Inicializa el pin del relé como salida y asegura que la bomba esté apagada al inicio.
- `void encender()`: Enciende la bomba/electroválvula estableciendo el pin del relé en `LOW` (el comportamiento puede variar según el tipo de relé).
- `void apagar()`: Apaga la bomba/electroválvula estableciendo el pin del relé en `HIGH`.

Archivo fuente "bomba_riego.cpp"

```
#include "bomba_riego.h"

// Constructor que inicializa el pin del relé
BombaRiego::BombaRiego(int pin) {
  pinRelé = pin;
}

// Método begin: configura el pin del relé como salida
void BombaRiego::begin() {
  pinMode(pinRelé, OUTPUT);
  apagar();  // Asegurar que la bomba esté apagada al inicio
}

// Método para encender la bomba/electroválvula
void BombaRiego::encender() {
  digitalWrite(pinRelé, LOW);  // LOW activa el relé (depende del tipo de relé)
}

// Método para apagar la bomba/electroválvula
void BombaRiego::apagar() {
  digitalWrite(pinRelé, HIGH);  // HIGH desactiva el relé
}
```

- `#include "bomba_riego.h"`: Incluye el archivo de cabecera "bomba_riego.h" para poder acceder a la definición de la clase `BombaRiego`.
- Implementación de los métodos de la clase `BombaRiego`:
  - `BombaRiego(int pin)`

- `void begin()`
- `void encender()`
- `void apagar()`

Archivo principal "main.cpp"

```
#include <Arduino.h>
#include "sensor_humedad.h"
#include "bomba_riego.h"

// Crear una instancia del sensor de humedad en el pin GPIO 34
SensorHumedad sensorSuelo(34);

// Crear una instancia de la bomba/electroválvula controlada por el relé en el pin GPIO 25
BombaRiego bombaRiego(25);

// Umbral de humedad para activar o desactivar el riego
const float umbralHumedad = 40.0;

void setup() {
  Serial.begin(115200);  // Inicializamos la comunicación serie
  sensorSuelo.begin();   // Inicializamos el sensor de humedad
  bombaRiego.begin();          // Inicializamos la bomba/electroválvula
}

void loop() {
  // Leer el porcentaje de humedad del suelo
  float humedad = sensorSuelo.leerHumedad();

  // Mostrar el valor de humedad en el monitor serie
  Serial.print("Humedad del suelo: ");
  Serial.print(humedad);
  Serial.println("%");

  // Lógica para encender o apagar la bomba según la humedad
  if (humedad < umbralHumedad) {
        Serial.println("Humedad baja, activando riego...");
        bombaRiego.encender();
  } else {
        Serial.println("Humedad suficiente, desactivando riego...");
        bombaRiego.apagar();
  }

  delay(1000);  // Pausa de 1 segundo entre lecturas
```

}

- `#include <Arduino.h>`: Incluye la biblioteca principal de Arduino.
- `#include "sensor_humedad.h"`: Incluye el archivo de cabecera para el sensor de humedad (no está incluido en el código proporcionado).
- `#include "bomba_riego.h"`: Incluye el archivo de cabecera para la clase `BombaRiego`.
- `SensorHumedad sensorSuelo(34)`: Crea una instancia del sensor de humedad en el pin GPIO 34 (se asume que la clase `SensorHumedad` está definida en "sensor_humedad.h").
- `BombaRiego bombaRiego(25)`: Crea una instancia de la bomba/electroválvula en el pin GPIO 25.
- `const float umbralHumedad = 40.0`: Define el umbral de humedad para activar o desactivar el riego.

El programa está estructurado en:

- **`void setup()`**:
  - `Serial.begin(115200)`: Inicializa la comunicación serie para mostrar información en el monitor serie.
  - `sensorSuelo.begin()`: Inicializa el sensor de humedad.
  - `bombaRiego.begin()`: Inicializa la bomba/electroválvula.
- **`void loop()`**:
  - `float humedad = sensorSuelo.leerHumedad()`: Lee el porcentaje de humedad del suelo usando el sensor.
  - `Serial.print("Humedad del suelo: "); Serial.print(humedad); Serial.println("%");`: Muestra el valor de humedad en el monitor serie.
  - Lógica de control:
    - Si la humedad es menor que el umbral (`humedad < umbralHumedad`):
      - Muestra un mensaje indicando que se activa el riego.
      - Enciende la bomba/electroválvula (`bombaRiego.encender()`).
    - Si no:
      - Muestra un mensaje indicando que se desactiva el riego.
      - Apaga la bomba/electroválvula (`bombaRiego.apagar()`).
  - `delay(1000)`: Pausa de 1 segundo entre lecturas.

4. Se considerará el Control de Nivel de un Tanque de almacenamiento de agua. Si llueve, el sistema de riego no debería activarse, en caso de no haber lluvia, el sistema debe activarse o no de acuerdo a la humedad del suelo.

Sensor Ultrasónico HC-Sr04.

El **sensor HC-SR04** mide distancias mediante el envío de un pulso ultrasónico que se refleja en un objeto (como el agua en un tanque) y se devuelve al sensor. El programa mide el tiempo que tarda el pulso en regresar y lo convierte en una distancia.

1. Archivo de cabecera `sensor_hcsr04.h`

Este archivo contiene la definición de la clase que utilizamos para interactuar con el sensor HC-SR04.

Veamos cada parte:

```
#ifndef SENSOR_HCSR04_H
#define SENSOR_HCSR04_H
```

```
#include <Arduino.h>  // Incluimos la librería base de Arduino

class SensorHCSR04 {
  private:
    int pinTrig;  // Pin que controla la señal de Trigg
    int pinEcho;  // Pin que lee la señal de Echo

  public:
    // Constructor que inicializa los pines Trigg y Echo
    SensorHCSR04(int trigPin, int echoPin);

    // Método de inicialización
    void begin();

    // Método para medir la distancia en cm
    float medirDistancia();
};

#endif
```

`#ifndef, #define, #endif`: Son directivas de preprocesador que garantizan que el archivo de cabecera no se incluya más de una vez en el programa, evitando errores de duplicación.

`#include <Arduino.h>`: Esto permite usar las funciones y tipos estándar de Arduino (como pinMode, digitalWrite, pulseIn, etc.).

`Clase SensorHCSR04`: Define la estructura y los métodos que vamos a utilizar para interactuar con el sensor.

`Atributos privados (pinTrig, pinEcho)`: Aquí almacenamos los pines que conectan el Trigg y el Echo del sensor.

Métodos públicos:

`SensorHCSR04(int trigPin, int echoPin)`: El constructor que recibe los pines y los asigna a los atributos de la clase.

`void begin()`: Configura los pines como entrada o salida. Es importante para definir el comportamiento correcto del sensor.

`float medirDistancia()`: Este método mide el tiempo de respuesta del pulso y lo convierte en una distancia en centímetros.

2. Archivo fuente `sensor_hcsr04.cpp`

Este archivo contiene la implementación de los métodos definidos en el archivo .h
Aquí es donde ocurre la "magia":

```
#include "sensor_hcsr04.h"

SensorHCSR04::SensorHCSR04(int trigPin, int echoPin) {
  pinTrig = trigPin;
  pinEcho = echoPin;
}

void SensorHCSR04::begin() {
  pinMode(pinTrig, OUTPUT);  // Configuramos el Trigg como salida
  pinMode(pinEcho, INPUT);   // Configuramos el Echo como entrada
}

float SensorHCSR04::medirDistancia() {
  // Enviamos un pulso de 10 microsegundos al Trigg
  digitalWrite(pinTrig, LOW);
  delayMicroseconds(2);
  digitalWrite(pinTrig, HIGH);
  delayMicroseconds(10);
  digitalWrite(pinTrig, LOW);

  // Medimos el tiempo que tarda en regresar el pulso (en microsegundos)
  long duracion = pulseIn(pinEcho, HIGH);

  // Convertimos el tiempo en distancia (velocidad del sonido: 0.034 cm/us)
  float distancia = duracion * 0.034 / 2;

  return distancia;
}
```

`SensorHCSR04::SensorHCSR04(int trigPin, int echoPin)`: Este es el constructor. Cuando se crea un objeto de la clase SensorHCSR04, este constructor asigna los pines Trigg y Echo a las variables internas pinTrig y pinEcho. Es la primera función que se ejecuta cuando el objeto se instancia.

`void begin()`: Aquí se configuran los pines.
`pinMode(pinTrig, OUTPUT)` configura el pin de Trigg como una salida digital (enviará señales), mientras que `pinMode(pinEcho, INPUT)` configura el pin de Echo como una entrada (recibirá señales).

`float medirDistancia()`: Esta es la función que mide la distancia usando el sensor:

Generar pulso en el Trigg: El sensor necesita un pulso en el pin Trigg para comenzar a medir. Esto se hace con digitalWrite. Se envía un pulso de 10 microsegundos (configurado por delayMicroseconds(10)).

Medir el tiempo del pulso de retorno: Después de enviar el pulso, el sensor espera a que el pulso de sonido rebote y regrese al Echo. La función `pulseIn(pinEcho, HIGH)` mide cuánto tiempo (en microsegundos) el pin Echo se mantiene en alto mientras recibe el pulso de retorno.

Calcular la distancia: La distancia se calcula basándonos en el tiempo que tardó el pulso en regresar. Sabemos que la velocidad del sonido es aproximadamente 0.034 cm por microsegundo, y como el pulso tiene que viajar de ida y vuelta, dividimos el resultado por 2:

```
distancia = (duración × 0.034) / 2
```

3. Archivo principal `main.cpp`

Este archivo es donde instancias la clase y la utilizas en tu programa principal:

```
#include "sensor_hcsr04.h"

SensorHCSR04 sensorNivel(39, 34);  // Pines configurados: Trigg en GIOP39 (pata 4), Echo en GIOP34 (pata 5)

void setup() {
  Serial.begin(9600);    // Inicializamos comunicación serial
  sensorNivel.begin();   // Inicializamos el sensor HC-SR04
}

void loop() {
  float nivelAgua = sensorNivel.medirDistancia();  // Medimos la distancia

  Serial.print("Nivel de agua: ");
  Serial.print(nivelAgua);
  Serial.println(" cm");  // Mostramos la distancia en cm

  delay(2000);  // Esperamos 2 segundos entre lecturas
}
```

`SensorHCSR04 sensorNivel(39, 34);`: Aquí creamos un objeto sensorNivel de la clase SensorHCSR04, pasándole los pines correspondientes al Trigg (39) y al Echo (34).

`setup()`: En esta función inicializamos el monitor serial con Serial.begin(9600) para ver la distancia en la consola, y llamamos a sensorNivel.begin() para configurar los pines del sensor.

`loop()`: Este es el ciclo principal del programa. Cada vez que se ejecuta, llama al método medirDistancia() para medir la distancia actual del agua en el tanque. Luego muestra el resultado en el monitor serial.

El programa está estructurado en:

- Un archivo .h que define la clase del sensor.
- Un archivo .cpp que implementa los métodos del sensor.
- Un archivo main.cpp donde usas la clase y sus métodos.