Maestría en Ciberseguridad y Ciberdefensa

Internet de las cosas

Docente:

Ing. Jaider Ospina Navas

Integrantes:

Andrés Camilo Molano

Raúl Andrés Garay

Enrique Ramírez Flor

Oscar Vega Pulido

Escuela Superior de Guerra “General Rafael Reyes Prieto”

Bogotá D.C

2025

Tabla de Contenido

Tabla de figuras

1. Objetivo del proyecto

Diseñar, implementar y analizar un sistema de tipo domótica utilizando un ESP32 y/o Arduino UNO, integrando un control local y también remoto, mediante un sensor de temperatura y humedad (DTH11), pantalla LCD I2C, una matriz LED y un Buzzer. Este montaje esta conectado a la plataforma Arduino Cloud para poder habilitar un ambiente de automatización y revisión en tiempo real.

2. Montaje

2.1 Componentes usados en montaje físico

- Arduino UNO
- Sensor DHT11
- Pantalla LCD I2C
- Cables Dupont
- Matriz LED MAX7219
- Buzzer pasivo

2.2 Componentes usados en montaje simulado Arduino

- Arduino UNO
- Sensor DHT22
- Pantalla LCD I2C
- Cables Dupont
- Matriz LED MAX7219
- Buzzer pasivo

2.3 Componentes usados en montaje simulado ESP32

- ESP32
- Sensor DHT22
- Pantalla LCD I2C
- Cables Dupont
- Matriz LED MAX7219
- Buzzer pasivo

2.4 Diagramas de conexión

Los circuitos montados fueron ensamblados de manera física y virtual por medio de simulación en la plataforma Wokwi para poder hacer una simulación antes de las conexiones físicas y asegurar su funcionamiento para su puesta en marcha en el laboratorio con los componentes físicos.

2.4.1 Objetivo de la simulación

Esta simulación implementa un sistema embebido que mide en tiempo real la temperatura y humedad ambiente usando un sensor DHT22, y muestra el resultado en una pantalla LCD I2C, además de representar el estado ambiental mediante una matriz LED 32x8 (4 módulos) y un Buzzer, que se activa como alerta sonora cuando los valores están fuera del rango de confort definido.

2.4.2 Conexiones simulación Arduino

COMPONENTE	PIN DEL COMPONENTE	PIN EN ARDUINO UNO	COLOR DE CABLE SUGERIDO	OBSERVACIONES
DHT11	VCC	5V	Rojo	Alimentación
	GND	GND	Negro	Tierra común
	DATA	2	Verde	Señal del sensor
LCD I2C	VCC	5V	Rojo	Alimentación
	GND	GND	Negro	Tierra
	SDA	A4	Azul	Comunicación I2C
	SCL	A5	Azul	Comunicación I2C
MATRIZ LED MAX7219	V+	5V	Rojo	Alimentación
	GND	GND	Negro	Tierra
	DIN	11	Verde	Entrada de datos SPI
	CS (LOAD)	10	Amarillo	Chip Select
	CLK	13	Azul claro	Reloj SPI
BUZZER PASIVO	+	3	Verde	Señal de activación
	-	GND	Negro	Tierra

Tabla 1 Conexiones a Arduino UNO

2.4.3 Diagrama de Conexión

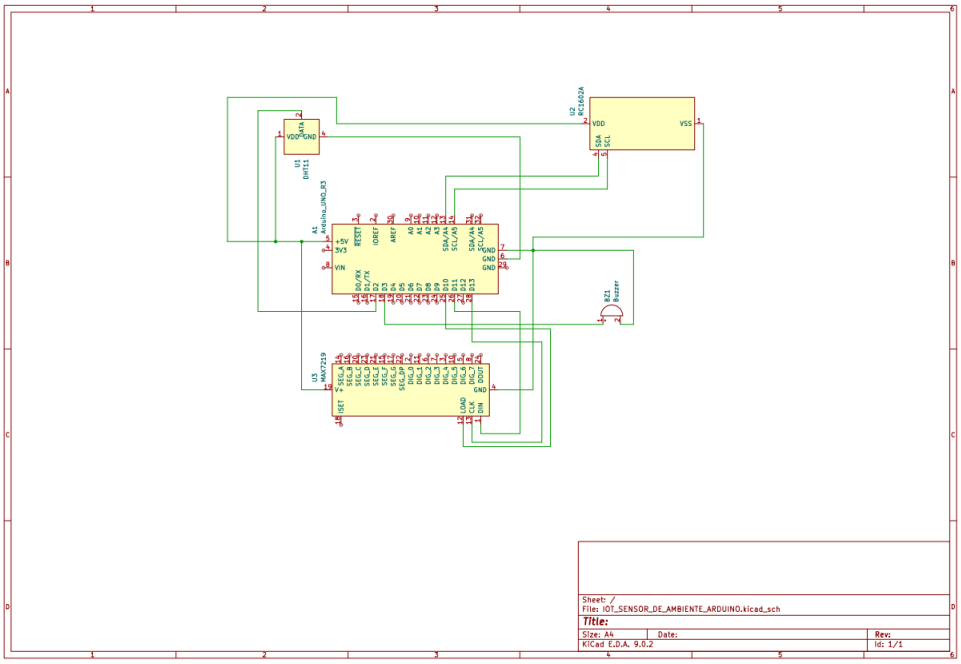

Figura 1 Diagrama de Conexión con Arduino UNO

2.4.4 Programación y pruebas

El código es adaptado para que funcione bajo el entorno simulado y para esto el componente DHT11 es cambiado a DHT22 dado que en la simulación no se cuenta con el componente DHT11, sin embargo, el cambio realizado no afecta el funcionamiento ya que son del mismo tipo de componente (sensor de temperatura y humedad).

A continuación, se presenta el código generado en wokwi, código que como se indica no contiene el módulo DHT11 sino el DHT22:

2.4.4.1 Código de programación

```
// 0) Declaración de los componentes a usar

#include <DHT.h>
#include <Adafruit_Sensor.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

#include <SPI.h>
#include <MD_MAX72xx.h>
#include <MD_Parola.h>

// ____________________________________________________________________
_____

//  1) PINES Y CONSTANTES
```

```
// ------------------------------------------------------------------------------

#define PIN_DHT        2      // DATA del DHT11
#define DHTTYPE      DHT22  // Tipo de sensor DHT
#define PIN_BUZZER     3      // Buzzer pasivo

// Pines de la cadena de 4 módulos MAX7219:
#define PIN_MATRIX_DIN   11   // MOSI
#define PIN_MATRIX_CLK   13   // SCK
#define PIN_MATRIX_CS    10   // CS / LOAD

// HARDWARE_TYPE para MD_MAX72XX::FC16_HW (módulos 8×8 + MAX7219)

#define HARDWARE_TYPE  MD_MAX72XX::FC16_HW
#define MAX_DEVICES     4    // 4 módulos en cascada → 32×8 LEDs

// Rango “confort”: rango que indica cual es la zona donde la temperatura y humedad deben permanecer

const float T_MIN = 18.0;  // Temp mínima (°C)
const float T_MAX = 25.0;  // Temp máxima (°C)
const float H_MIN = 30.0;  // Humedad mínima (%)
const float H_MAX = 60.0;  // Humedad máxima (%)

// ------------------------------------------------------------------------------
//  2) OBJETOS DE BIBLIOTECAS
// ------------------------------------------------------------------------------

DHT             dht(PIN_DHT, DHTTYPE);
LiquidCrystal_I2C lcd(0x27, 16, 2);

// Para bitmaps estáticos:
MD_MAX72XX      mx = MD_MAX72XX(HARDWARE_TYPE, PIN_MATRIX_DIN, PIN_MATRIX_CLK, PIN_MATRIX_CS, MAX_DEVICES);

// También inicializamos Parola (no usamos animaciones aquí, pero conviene):
MD_Parola       parola = MD_Parola(HARDWARE_TYPE, PIN_MATRIX_CS, MAX_DEVICES);

// ------------------------------------------------------------------------------
//  3) BITMAPS 8×8 PARA CADA MÓDULO
```

```
// ------------------------------------------------------------------------------

static const uint8_t iconHappy[8] = {
  B00111100,
  B01000010,
  B10100101,
  B10000001,
  B10100101,
  B10011001,
  B01000010,
  B00111100
};

static const uint8_t iconWarn[8] = {
  B00011000,
  B00111100,
  B01111110,
  B01111110,
  B01111110,
  B00111100,
  B00011000,
  B00011000
};

// ------------------------------------------------------------------------------
//  4) SETUP
// ------------------------------------------------------------------------------

void setup() {
  // 4.1 Serial para debug (opcional)
  Serial.begin(9600);

  // 4.2 Inicializar sensor DHT11
  dht.begin();

  // 4.3 Inicializar LCD I2C
  lcd.init();
  lcd.backlight();

  // 4.4 Inicializar buzzer
  pinMode(PIN_BUZZER, OUTPUT);
  digitalWrite(PIN_BUZZER, LOW);  // Empieza apagado
```

```
  // 4.5 Inicializar matriz de LEDs
  mx.begin();                          // Inicia MD_MAX72XX
  mx.control(MD_MAX72XX::INTENSITY, 8);    // Brillo medio (0–15)
  mx.clear();                          // Limpia toda la matriz

  // Inicializar Parola (si en el futuro se agregan textos animados)
  parola.begin();
  parola.displayClear();

  // 4.6 Mensaje inicial en LCD
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("  Monitor Clima  ");
  lcd.setCursor(0, 1);
  lcd.print("   Iniciando...  ");
  delay(2000);
}

// ——————————————————————————————————————————————————————————————————————
//  5) LOOP PRINCIPAL
// ——————————————————————————————————————————————————————————————————————

void loop() {
  // 5.1 Leer temperatura y humedad del DHT11
  float temp = dht.readTemperature();
  float hum  = dht.readHumidity();

  // 5.2 Verificar lectura válida
  if (isnan(temp) || isnan(hum)) {
    // Error: mostrar en LCD y limpiar matriz
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("  Error DHT11   ");
    mx.clear();
    // Apagar buzzer por seguridad
    noTone(PIN_BUZZER);
    delay(2000);
    return;
  }

  // 5.3 Mostrar lecturas en el LCD
  lcd.clear();
```

```
  lcd.setCursor(0, 0);
  lcd.print("Temp: ");
  lcd.print(temp, 1);
  lcd.print((char)223);  // Símbolo “°”
  lcd.print("C");

  lcd.setCursor(0, 1);
  lcd.print("Hum:  ");
  lcd.print(hum, 1);
  lcd.print("%");

  // 5.4 Determinar si está dentro de rango “confort”
  bool dentroConfort = (temp >= T_MIN && temp <= T_MAX && hum >= H_MIN &&
hum <= H_MAX);

  // 5.5 Dibujar icono en la matriz (4 módulos)
  mx.clear();
  for (uint8_t module = 0; module < MAX_DEVICES; module++) {
    for (uint8_t row = 0; row < 8; row++) {
      uint8_t value = dentroConfort ? iconHappy[row] : iconWarn[row];
      mx.setRow(module, row, value);
    }
  }

  // 5.6 Activar o desactivar buzzer según estado
  if (dentroConfort) {
    // Ambiente cómodo → apagar buzzer
    noTone(PIN_BUZZER);
  } else {
    // Fuera de rango → emitir tono de 1 kHz
    tone(PIN_BUZZER, 1000);
  }

  // 5.7 Debug por Serial (opcional)
  Serial.print("Temp: ");
  Serial.print(temp);
  Serial.print(" C  |  Hum: ");
  Serial.print(hum);
  Serial.print(" %  | Estado: ");
  Serial.println(dentroConfort ? "CONFORT" : "FUERA");

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

2.4.4.2 Evidencia de simulación

La simulación a continuación muestra el modo apagado y el modo en funcionamiento luego de hacer el respectivo montaje, conexiones y programación de este sensor de temperatura y humedad.

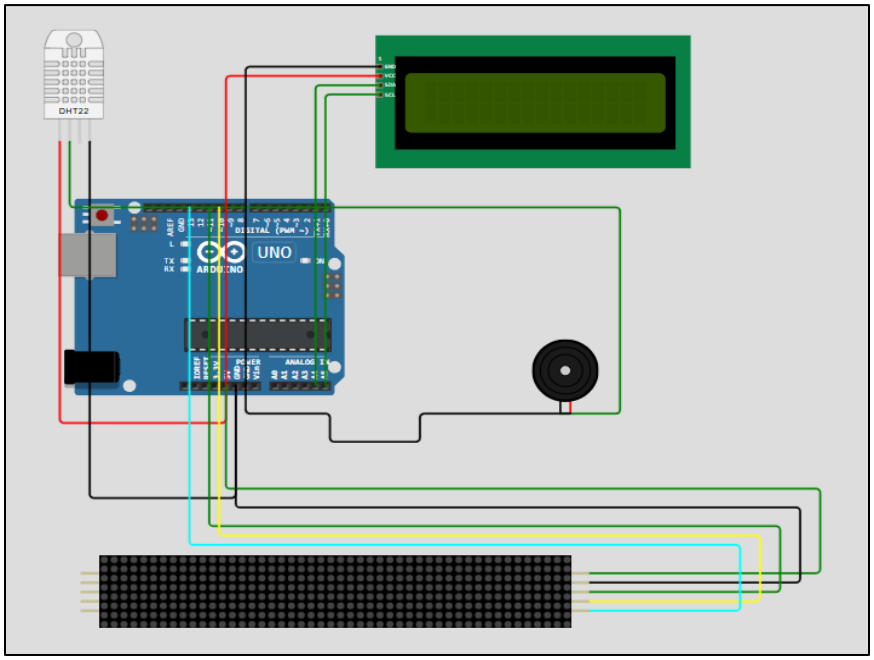

Figura 2 Simulación con Arduino Apagad.

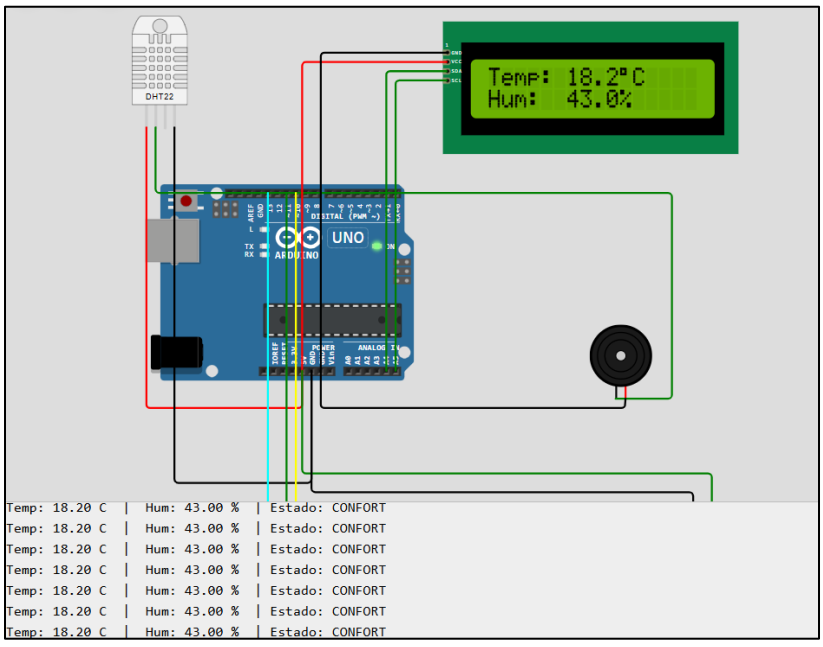

Figura 3 Simulación con Arduino en funcionamiento.

2.4.4.3 Pruebas de funcionamiento

Lectura de sensores:

- Se toma una muestra de temperatura y humedad.
- Se verifica si la lectura es válida (para evitar errores de hardware o conexión).

Visualización en LCD:

- Se muestran los valores actuales en la pantalla LCD.
- Temperatura en la primera línea, humedad en la segunda.

Evaluación del ambiente:

- Se verifica si los valores están dentro del rango "confort".
- Según esta evaluación, se elige uno de dos íconos:
- iconHappy: rostro sonriente.
- iconWarn: signo de advertencia.

Visualización en matriz LED:

- Se limpia la matriz y se dibuja el icono correspondiente en cada uno de los 4 módulos.

Alerta sonora:

- Si los valores están fuera del rango → el buzzer suena a 1000 Hz.
- Si están dentro del rango → se apaga el buzzer.

Salida serial (debug):

- Se imprime por consola la temperatura, la humedad y el estado de confort.

Pausa:

- El ciclo espera 2 segundos antes de realizar una nueva lectura.

2.4.4.4 Vista de conexiones en Wokwi

En la plataforma wokwi se muestra como se dan las conexiones. Estas conexiones se detallan a través de un archivo .json y allí se muestra uno a uno como están las conexiones que se presentaron en el diagrama de conexiones anteriormente mencionado.

{

"version": 1,

"author": "wokwi",

"editor": "wokwi",

```
"parts": [
  { "type": "wokwi-arduino-uno", "id": "uno", "top": 39, "left": 47.4, "attrs": {} },
  {
    "type": "wokwi-dht22",
    "id": "dht",
    "top": -134.1,
    "left": 33,
    "attrs": { "temperature": "18.2" }
  },
  {
    "type": "wokwi-max7219-matrix",
    "id": "matrix",
    "top": 403.8,
    "left": 67.46,
    "attrs": { "chain": "4" }
  },
  { "type": "wokwi-buzzer", "id": "buzz", "top": 175.2, "left": 501, "attrs": {} },
  {
    "type": "wokwi-lcd1602",
    "id": "lcd1",
    "top": -128,
    "left": 351.2,
    "attrs": { "pins": "i2c" }
  }
],
"connections": [
  [ "dht:VCC", "uno:5V", "red", [ "v288", "h159.4" ] ],
```

```
    [ "dht:GND", "uno:GND.2", "black", [ "v364.8", "h140.1" ] ],
    [ "dht:SDA", "uno:2", "green", [ "v0" ] ],
    [ "lcd1:VCC", "uno:5V", "red", [ "h0" ] ],
    [ "lcd1:GND", "uno:GND.3", "black", [ "h0" ] ],
    [ "lcd1:SDA", "uno:A4", "green", [ "h0" ] ],
    [ "lcd1:SCL", "uno:A5", "green", [ "h0" ] ],
    [ "buzz:1", "uno:3", "green", [ "h57.6", "v-211.2" ] ],
    [ "buzz:2", "uno:GND.3", "black", [ "h-0.4", "v9.6", "h-311.2" ] ],
    [ "matrix:V+", "uno:5V", "green", [ "h220.8", "v-86.4", "h-570.2" ] ],
    [ "matrix:GND", "uno:GND.2", "black", [ "h201.6", "v-76.8", "h-541.5" ] ],
    [ "matrix:DIN", "uno:11", "green", [ "h182.4", "v-67.2", "h-547.8" ] ],
    [ "matrix:CS", "uno:10", "yellow", [ "h163.2", "v-67.2", "h-519.1" ] ],
    [ "matrix:CLK", "uno:13", "cyan", [ "h144", "v-67.2", "h-528.4" ] ]
  ],
  "dependencies": {}
}
```


2.4.4.5 Reflexión

Este código representa un proyecto completo y funcional de monitoreo ambiental basado en Arduino. Integra múltiples formas de visualización (LCD, matriz LED) y alerta (Buzzer), lo que permite una respuesta inmediata tanto para el usuario como para futuras aplicaciones domóticas o industriales.

Puntos fuertes del código

- Modular y limpio, fácilmente adaptable.
- Incluye validación de datos del sensor.
- Usa salidas visuales y sonoras de forma coordinada.
- Permite fácil expansión con WiFi, almacenamiento o comunicación remota.

Posibles mejoras

- Agregar RTC (reloj de tiempo real) para registrar hora/fecha de las lecturas.

- Guardar históricos de temperatura/humedad en tarjeta SD.
- Agregar conectividad WiFi (ESP8266/ESP32) para monitoreo remoto.
- Configurar umbrales variables desde una interfaz externa.

2.4.5 Objetivo de la simulación

Este código tiene como finalidad construir un sistema de monitoreo ambiental que mida temperatura y humedad en tiempo real con un sensor DHT22, muestre los datos en una pantalla LCD I2C, visualice un icono en una matriz LED (indicando si las condiciones son cómodas o no) y active un Buzzer como alerta sonora si los valores están fuera de un rango de confort predeterminado.

2.4.6 Conexiones simulación ESP32

COMPONENTE	PIN DEL COMPONENTE	PIN EN ESP32	COLOR DE CABLE SUGERIDO	OBSERVACIONES
DHT11	VCC	3V	Rojo	Alimentación
	GND	GND	Negro	Tierra común
	DATA	15	Verde	Señal del sensor
LCD I2C	VCC	3V	Rojo	Alimentación
	GND	GND	Negro	Tierra
	SDA	21	Azul	Comunicación I2C
	SCL	22	Azul	Comunicación I2C
MATRIZ LED MAX7219	V+	VIN	Rojo	Alimentación
	GND	GND	Negro	Tierra
	DIN	23	Verde	Entrada de datos SPI
	CS (LOAD)	5	Amarillo	Chip Select
	CLK	18	Azul claro	Reloj SPI
BUZZER PASIVO	+	25	Verde	Señal de activación
	-	GND	Verde	Tierra

Tabla 2 Conexiones a ESP32

2.4.7 Diagrama de Conexión

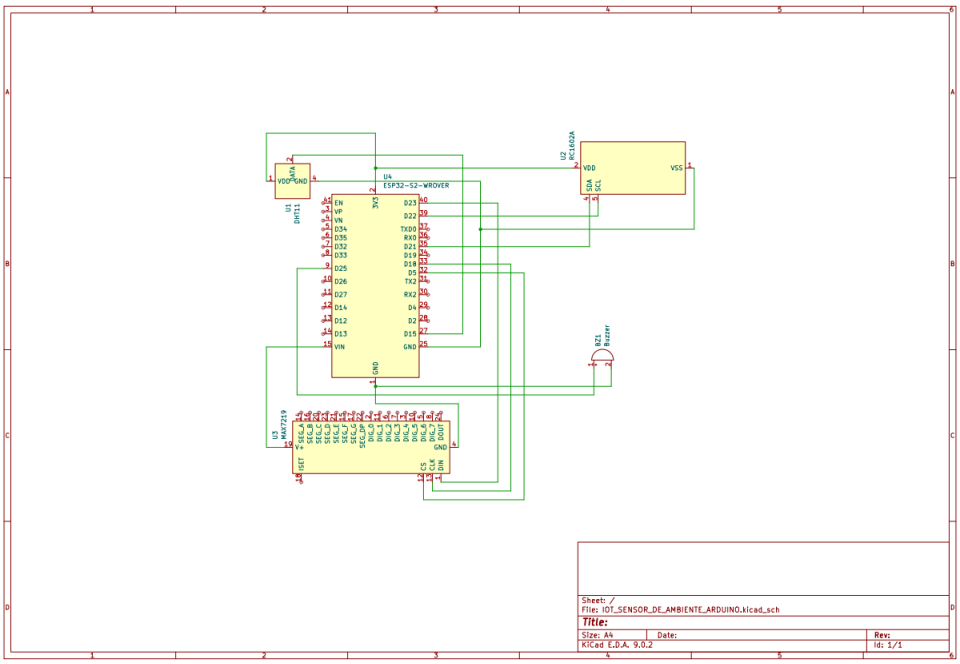

Figura 4 Diagrama de Conexión con ESP32

2.4.8 Programación y pruebas

El código es adaptado para que funcione bajo el entorno simulado y para esto el componente DHT11 es cambiado a DHT22 dado que en la simulación no se cuenta con el componente DHT11, sin embargo, el cambio realizado no afecta el funcionamiento ya que son del mismo tipo de componente (sensor de temperatura y humedad).

A continuación, se presenta el código generado en wokwi, código que como se indica no contiene el módulo DHT11 sino el DHT22:

2.4.8.1 Código de programación

```
// 0) Declaración de los componentes a usar

#include <DHT.h>

#include <Adafruit_Sensor.h>

#include <Wire.h>

#include <LiquidCrystal_I2C.h>


#include <SPI.h>

#include <MD_MAX72xx.h>
```

```
#include <MD_Parola.h>

// ——————————————————————————————————————————————
//  1) PINES Y CONSTANTES
// ——————————————————————————————————————————————

#define PIN_DHT        15      // DATA del DHT11
#define DHTTYPE        DHT22  // Tipo de sensor DHT
#define PIN_BUZZER     25      // Buzzer pasivo

// Pines de la cadena de 4 módulos MAX7219:
#define PIN_MATRIX_DIN   23   // MOSI
#define PIN_MATRIX_CLK   18   // SCK
#define PIN_MATRIX_CS    5   // CS / LOAD

// HARDWARE_TYPE para MD_MAX72XX::FC16_HW (módulos 8×8 + MAX7219)
#define HARDWARE_TYPE  MD_MAX72XX::FC16_HW
#define MAX_DEVICES     6    // 4 módulos en cascada → 32×8 LEDs

// Rango “confort”:
const float T_MIN = 18.0;  // Temp mínima (°C)
const float T_MAX = 25.0;  // Temp máxima (°C)
const float H_MIN = 30.0;  // Humedad mínima (%)
const float H_MAX = 60.0;  // Humedad máxima (%)
```

```
// ------------------------------------------------------------------------
//  2) OBJETOS DE BIBLIOTECAS
// ------------------------------------------------------------------------

DHT             dht(PIN_DHT, DHTTYPE);
LiquidCrystal_I2C lcd(0x27, 16, 2);

// Para bitmaps estáticos:
MD_MAX72XX      mx = MD_MAX72XX(HARDWARE_TYPE, PIN_MATRIX_DIN,
PIN_MATRIX_CLK, PIN_MATRIX_CS, MAX_DEVICES);

// También inicializamos Parola (no usamos animaciones aquí, pero conviene):
//MD_Parola       parola = MD_Parola(HARDWARE_TYPE, PIN_MATRIX_CS,
MAX_DEVICES);

// ------------------------------------------------------------------------
//  3) BITMAPS 8×8 PARA CADA MÓDULO
// ------------------------------------------------------------------------

static const uint8_t iconHappy[8] = {
  B00111100,
  B01000010,
  B10100101,
  B10000001,
  B10100101,
```

```
  B10011001,
  B01000010,
  B00111100
};

static const uint8_t iconWarn[8] = {
  B00011000,
  B00111100,
  B01111110,
  B01111110,
  B01111110,
  B00111100,
  B00011000,
  B00011000
};

// ———————————————————————————————————————————————————————
//  4) SETUP
// ———————————————————————————————————————————————————————

void setup() {
  // 4.1 Serial para debug (opcional)
  Serial.begin(9600);

  // 4.2 Inicializar sensor DHT11
  dht.begin();
```

```
  // 4.3 Inicializar LCD I2C
  lcd.init();
  lcd.backlight();

  // 4.4 Inicializar buzzer
  pinMode(PIN_BUZZER, OUTPUT);
  digitalWrite(PIN_BUZZER, LOW);  // Empieza apagado

  // 4.5 Inicializar matriz de LEDs
  mx.begin();                          // Inicia MD_MAX72XX
  mx.control(MD_MAX72XX::INTENSITY, 8);    // Brillo medio (0–15)
  mx.clear();                          // Limpia toda la matriz

  // Inicializar Parola (si en el futuro se agregan textos animados)
  //parola.begin();
  //parola.displayClear();

  // 4.6 Mensaje inicial en LCD
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("  Monitor Clima  ");
  lcd.setCursor(0, 1);
  lcd.print("   Iniciando...  ");
  delay(2000);
}
```

```
// ------------------------------------------------------------------------
//  5) LOOP PRINCIPAL
// ------------------------------------------------------------------------

void loop() {
  // 5.1 Leer temperatura y humedad del DHT11
  float temp = dht.readTemperature();
  float hum  = dht.readHumidity();

  // 5.2 Verificar lectura válida
  if (isnan(temp) || isnan(hum)) {
    // Error: mostrar en LCD y limpiar matriz
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("  Error DHT11   ");
    mx.clear();
    // Apagar buzzer por seguridad
    noTone(PIN_BUZZER);
    delay(2000);
    return;
  }

  // 5.3 Mostrar lecturas en el LCD
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Temp: ");
```

```
  lcd.print(temp, 1);
  lcd.print((char)223);  // Símbolo “°”
  lcd.print("C");

  lcd.setCursor(0, 1);
  lcd.print("Hum:  ");
  lcd.print(hum, 1);
  lcd.print("%");

  // 5.4 Determinar si está dentro de rango “confort”
  bool dentroConfort = (temp >= T_MIN && temp <= T_MAX && hum >= H_MIN &&
hum <= H_MAX);

  // 5.5 Dibujar icono en la matriz (4 módulos)
  mx.clear();
  for (uint8_t module = 0; module < MAX_DEVICES; module++) {
    for (uint8_t row = 0; row < 8; row++) {
      uint8_t value = dentroConfort ? iconHappy[row] : iconWarn[row];
      mx.setRow(module, row, value);
    }
  }

  // 5.6 Activar o desactivar buzzer según estado
  if (dentroConfort) {
    // Ambiente cómodo → apagar buzzer
    noTone(PIN_BUZZER);
  } else {
```

```
  // Fuera de rango → emitir tono de 1 kHz
  tone(PIN_BUZZER, 1000);
 }
 // 5.7 Debug por Serial (opcional)
 Serial.print("Temp: ");
 Serial.print(temp);
 Serial.print(" C  |  Hum: ");
 Serial.print(hum);
 Serial.print(" %  | Estado: ");
 Serial.println(dentroConfort ? "CONFORT" : "FUERA");


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

2.4.8.2 Evidencia de simulación

La simulación a continuación muestra el modo apagado y el modo en funcionamiento luego de hacer el respectivo montaje, conexiones y programación de este sensor de temperatura y humedad.

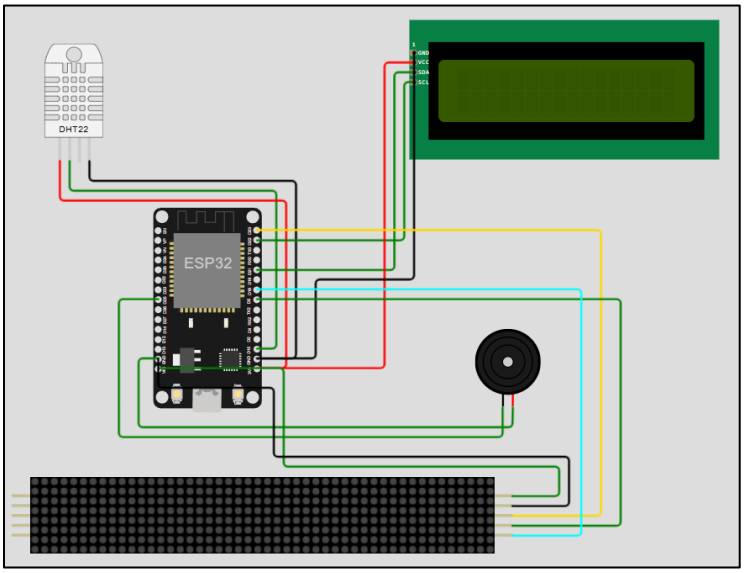

Figura 5 Simulación con ESP32 Apagado.

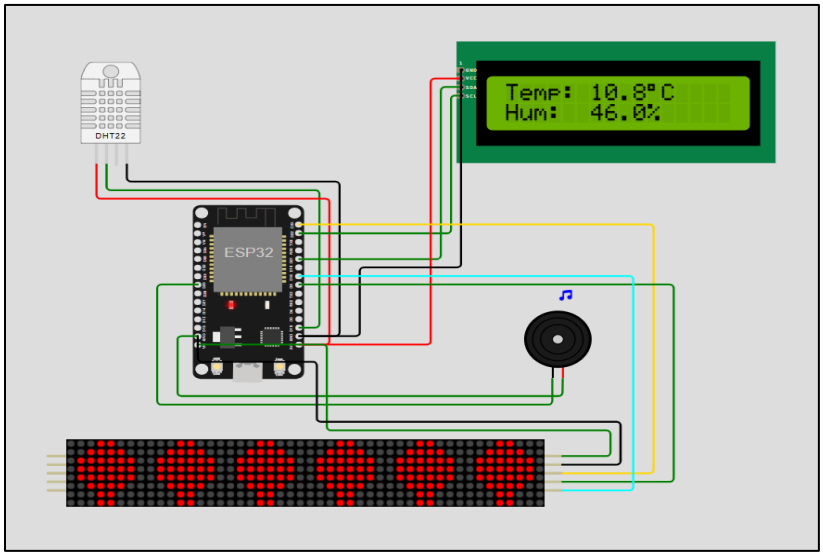

Figura 6 Simulación con ESP32 en funcionamiento.

Cabe mencionar que durante la programación con ESP32 se comentaron las líneas MD_PAROLA parola =, parola.begin y parola.display, esto para dar funcionamiento a los leds de la Matriz.

2.4.8.3 Pruebas de funcionamiento

Lectura de sensores:

- Se toma una muestra de temperatura y humedad.
- Se verifica si la lectura es válida (para evitar errores de hardware o conexión).

Visualización en LCD:

- Se muestran los valores actuales en la pantalla LCD.
- Temperatura en la primera línea, humedad en la segunda.

Evaluación del ambiente:

- Se verifica si los valores están dentro del rango “confort”.
- Según esta evaluación, se elige uno de dos íconos:

- iconHappy: rostro sonriente.
- iconWarn: signo de advertencia.

Visualización en matriz LED:

- Se limpia la matriz y se dibuja el icono correspondiente en cada uno de los 4 módulos.

Alerta sonora:

- Si los valores están fuera del rango → el buzzer suena a 1000 Hz.
- Si están dentro del rango → se apaga el buzzer.

Salida serial (debug):

- Se imprime por consola la temperatura, la humedad y el estado de confort.

Pausa:

- El ciclo espera 2 segundos antes de realizar una nueva lectura.

2.4.8.4 Vista de conexiones en Wokwi

En la plataforma wokwi se muestra como se dan las conexiones. Estas conexiones se detallan a través de un archivo .json y allí se muestra uno a uno como están las conexiones que se presentaron en el diagrama de conexiones anteriormente mencionado.


```
{
 "version": 1,
 "author": "wokwi",
 "editor": "wokwi",
 "parts": [
  { "type": "wokwi-esp32-devkit-v1", "id": "esp", "top": 81.5, "left": 167.8, "attrs": {} },
  {
   "type": "wokwi-dht22",
   "id": "dht",
   "top": -76.5,
   "left": 61.8,
   "attrs": { "temperature": "18.2" }
  },
  {
   "type": "wokwi-max7219-matrix",
   "id": "matrix",
   "top": 346.2,
```

```
    "left": 29.06,
    "attrs": { "chain": "6" }
   },
   { "type": "wokwi-buzzer", "id": "buzz", "top": 194.4, "left": 481.8, "attrs": {} },
   {
    "type": "wokwi-lcd1602",
    "id": "lcd1",
    "top": -99.2,
    "left": 418.4,
    "attrs": { "pins": "i2c" }
   }
  ],
  "connections": [
   [ "dht:SDA", "esp:D15", "green", [ "v28.8", "h201.7", "v153.8" ] ],
   [ "dht:VCC", "esp:3V3", "red", [ "v38.4", "h220.8", "v163.2" ] ],
   [ "dht:GND", "esp:GND.1", "black", [ "v19.2", "h201.6", "v172.9" ] ],
   [ "lcd1:SDA", "esp:D21", "green", [ "h-19.2", "v192.6" ] ],
   [ "lcd1:SCL", "esp:D22", "green", [ "h-9.6", "v154.2" ] ],
   [ "lcd1:VCC", "esp:3V3", "red", [ "h-28.8", "v297.7" ] ],
   [ "lcd1:GND", "esp:GND.1", "black", [ "v220.8", "h-96", "v76.9" ] ],
   [ "matrix:DIN", "esp:D23", "gold", [ "h86.4", "v-278.5" ] ],
   [ "matrix:CS", "esp:D5", "green", [ "h105.6", "v-220.8" ] ],
   [ "matrix:CLK", "esp:D18", "cyan", [ "h67.2", "v-240" ] ],
   [ "buzz:1", "esp:D25", "green", [ "v28.8", "h-374.4", "v-134.4" ] ],
   [ "buzz:2", "esp:GND.2", "green", [ "h-0.4", "v19.2", "h-364.8", "v-67.1" ] ],
   [ "matrix:V+", "esp:VIN", "green", [ "h45.58", "v-28.8", "h-268.8", "v-96" ] ],
```

["matrix:GND", "esp:GND.2", "black", ["h55.18", "v-48", "h-288", "v-67.2", "h38.4"]
]

],

"dependencies": {}

}

2.4.8.5 Reflexión

Este código representa un sistema eficaz y didáctico para la monitorización ambiental en tiempo real, combinando entrada (sensor), procesamiento lógico, y salidas múltiples (pantalla, matriz LED, Buzzer). Utiliza buenas prácticas como la separación de funciones por bloques, verificación de errores y uso eficiente de librerías.

Ventajas:

- Fácil de adaptar a diferentes umbrales o sensores.
- Permite monitoreo visual, sonoro y digital (Serial).
- Interfaz amigable gracias a LCD y matriz LED.

Sugerencias de mejora:

- Agregar historial o almacenamiento de datos (con SD o en la nube).
- Implementar interfaz web o Bluetooth para consulta remota.
- Añadir niveles de alerta con distintos tonos o animaciones.

2.4.9 Visualización de simulaciones en plataforma Wokwi

A continuación, se comparte los enlaces públicos donde se puede acceder a correr dichas simulaciones. Las simulaciones están protegidas y por tanto no pueden modificarse, sin embargo, en caso de querer hacerlo, se puede realizar una copia de estos proyectos y allí hacer las modificaciones que se gusten hacer para agregar y/o eliminar componentes o módulos.

a. La simulación con Arduino puede ser consultada en: https://wokwi.com/projects/432673766990872577

b. La simulación con ESP32 puede ser consultada en: https://wokwi.com/projects/432684060098027521

2.5 ¿Dónde puede aplicarse esta solución o mantaje?

2.5.1 Estación meteorológica doméstica o educativa

Uso

- Medición de temperatura y humedad ambiente.
- Mostrar datos en la pantalla LCD en tiempo real.
- Usar la matriz LED para mostrar símbolos (sol, nube, gotas) según condiciones.
- El buffer (buzzer) puede alertar sobre temperaturas extremas.

Aplicación

- Hogares inteligentes.
- Salones de clase para educación STEM.
- Estaciones de monitoreo en invernaderos o jardines urbanos.

2.5.2 Sistema de monitoreo ambiental para invernaderos o cultivos indoor

Uso

- El sensor controla el microclima del invernadero.
- La pantalla LCD muestra el estado de temperatura y humedad.
- La matriz LED actúa como semáforo ambiental (verde, amarillo, rojo).
- El buffer suena si se superan límites críticos.

Aplicación

- Agricultura urbana.
- Cultivos automatizados.
- Sistemas de domótica para producción vegetal.

2.5.3 Monitor ambiental en espacios sensibles (laboratorios, hospitales, museos)

Uso

- Registro constante de condiciones ambientales.
- Alarmas mediante buzzer ante desvíos peligrosos.
- Indicadores visuales claros para el personal (matriz LED).
- Datos desplegados en LCD de forma local.

Aplicación

- Control de humedad en salas de archivo o conservación.
- Quirófanos o salas limpias.

2.5.4 Sistema de alerta temprana en espacios cerrados

Uso

- Activación del buzzer si la temperatura supera cierto umbral.
- Matriz LED muestra mensajes como "ALERTA", "CALOR", "HUMEDAD".

- LCD entrega detalles como "Temp: 34°C - Riesgo de calor".

Aplicación

- Seguridad ocupacional en fábricas.
- Talleres, bodegas o cuartos eléctricos.

2.5.5 Proyecto educativo de IoT con Arduino Cloud o Blynk

Uso

- Recoger datos del sensor y mostrarlos en tiempo real localmente y en la nube.
- Interactuar con usuarios mediante botones o comandos remotos.
- Notificaciones visuales y sonoras.

Aplicación

- Tesis de ingeniería electrónica o mecatrónica.
- Talleres de formación en programación y electrónica.

Demostración de sistemas embebidos.

Ejemplo de uso industrial:

Proyecto: Sistema de Monitoreo Ambiental Industrial con Alerta Visual y Sonora

3. Conclusiones

Se logra la implementación de un sistema de IoT funcional basado en Arduino UNO, con múltiples métodos de control, visualización y alertas. La plataforma Arduino IoT Cloud demostró ser una solución efectiva para desarrollar y monitorear soluciones IoT.

Este proyecto evidencia la viabilidad de aplicar estas tecnologías en hogares, industrias y entornos educativos, con un enfoque en seguridad y automatización.

Las simulaciones realizadas arrojan resultados satisfactorios para poder llevar a la vida real la implementación de la solución y así asegurar su funcionamiento sin el riesgo de daño sobre alguno de los módulos o componentes.