ESP32: MANUAL BÁSICO PARA ESTUDIANTES

ALUMNI EDITORA 2025

PRIMERA EDICIÓN

ORTIZ ARCINIEGA – VALENCIA BARAHONA – BOSMEDIANO CÁRDENAS – BASTIDAS JÁCOME – AGUIRRE CHAGNA – JÁCOME AYALA

ESP32: Manual básico para estudiantes

Autores

José Luis Ortiz Arciniega

- Ingeniero en Electrónica y Redes de Comunicación
- Máster Universitario en Ingeniería Electrónica

https://orcid.org/0000-0003-3707-5252

jlortiz@ist17dejulio.edu.ec

Fernando Xavier Valencia Barahona

- Tecnólogo en Sistemas
- Ingeniero en Electrónica y Redes de Comunicación

https://orcid.org/0009-0006-5889-4918

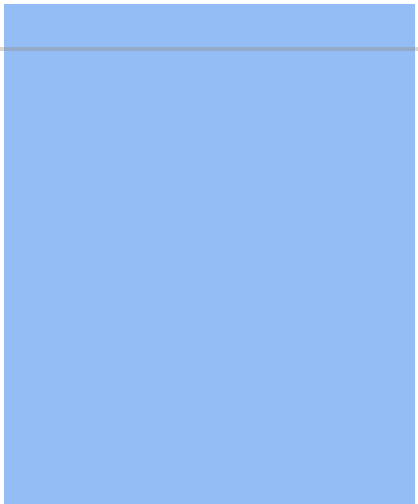
fvalencia@ ist17dejulio.edu.ec

Carlos Patricio Bosmediano Cárdenas

- Ingeniero en Electrónica y Redes de Comunicación
- Magister en Telecomunicaciones

https://orcid.org/0000-0003-3707-5252

cbosmediano@ist17dejulio.edu.ec



ESP32: Manual básico para estudiantes

Autores

Alex Danilo Bastidas Jácome

- Ingeniero en Mecatrónica
- Magister en Mecatrónica mención en Procesos Industriales

https://orcid.org/0009-0007-7601-7588

jtorres@ist17dejulio.edu.ec

Víctor Hugo Aguirre Chagna

- Ingeniero en Electrónica y Control
- Magister en Seguridad Industrial

https://orcid.org/0000-0002-0883-6780

vaguirre@ist17dejulio.edu.ec

Paulina Johanna Jácome Ayala

- Ingeniera en Sistemas Computacionales
- Master Universitario en Análisis y Visualización de Datos Masivos / Visual Analytics And Big Data

https://orcid.org/0009-0001-7046-7226

pjacome@ ist17dejulio.edu.ec

ESP32: Manual básico para estudiantes

Catalogación Bibliográfica

Autores	• José Luis Ortiz Arciniega • Fernando Xavier Valencia Barahona • Carlos Patricio Bosmediano Cárdenas • Alex Danilo Bastidas Jácome • Víctor Hugo Aguirre Chagna • Paulina Johanna Jácome Ayala
Título	ESP32: Manual básico para estudiantes
Descriptores	Microcontroladores, Sistemas embebidos, Programación, Electrónica, ESP32
Dewey	621.38
Thema	TJFD
Publicación	Febrero 2025
Edición	Primera
ISBN	978-9942-7307-6-3
DOI	https://doi.org/10.70625/alumned/9
Editorial	Alumni Editora
Pais - Ciudad	Ecuador - Atuntaqui
Formato	Adobe Acrobat Reader
Páginas	87

Cámara Ecuatoriana del Libro

ESP32: Manual básico para estudiantes

ALUMNI EDITORA

Peer Review Process

Revisión de Pares

Este libro ha sido evaluado mediante un proceso de revisión por pares externos bajo el formato de doble ciego. En consecuencia, la investigación presentada en esta obra cuenta con el respaldo de expertos en la materia, quienes han emitido un juicio imparcial basado en criterios científicos, garantizando así la solidez académica del contenido.

Peer Review

This book has undergone a peer review process by external academics using a double-blind system. Consequently, the research presented in this work has the endorsement of subject matter experts, who have provided an impartial assessment based on scientific criteria, ensuring the academic rigor of the content.

Declaración del Editor

Declaración del Editor

Alumni Editora declara para todos los efectos legales, que:

Esta publicación implica únicamente una cesión temporal de los derechos de autor y de publicación, sin que ello constituya responsabilidad solidaria en la creación de los manuscritos publicados en conformidad con la Ley de Propiedad Intelectual y las normativas legales aplicables.

Autoriza y fomenta que los autores firmen acuerdos con repositorios institucionales con el fin exclusivo de difundir la obra, siempre que se reconozca adecuadamente la autoría y la edición, y que no existan fines comerciales involucrados.

Todos los libros electrónicos publicados son de acceso abierto y, por lo tanto, no se venden en el sitio web de Alumni Editora, ni en plataformas asociadas, de comercio electrónico u otros medios virtuales o físicos, eximiéndose de la transferencia de derechos de autor a los autores.

Todos los miembros del consejo editorial cuentan con el grado académico de cuarto nivel y están vinculados a instituciones de educación superior, conforme a las recomendaciones de las entidades de evaluación académica nacionales e internacionales para la obtención de estándares de calidad editorial.

Alumni Editora no transfiere, comercializa, ni autoriza el uso de los nombres, correos electrónicos u otros datos personales de los autores para fines distintos a la difusión de esta obra.

Declaración del Autor

Declaración del Autor

El autor de la obra declara: 1. no poseer ningún interés comercial que pueda representar un conflicto de interés en relación con el presente documento publicado; 2. Asegura haber participado activamente en la elaboración del manuscrito, específicamente en la concepción del estudio, la obtención de datos y/o su análisis e interpretación; la redacción o revisión del documento para garantizar su relevancia intelectual y la aprobación final del manuscrito antes de su envío; 3. Certifica que el contenido publicado está libre de datos o resultados fraudulentos; 4. Confirma que todas las citas y referencias de datos e interpretaciones de investigaciones previas son correctas; 5. Reconoce haber declarado todas las fuentes de financiamiento recibidas para la investigación; 6. Autoriza la publicación de la obra, que incluye su inclusión en catálogos, asignación de ISBN, DOI, otros índices, diseño visual, portada, maquetación interior, y su posterior difusión según lo dispuesto por Alumni Editora.

Prólogo

En la era de la digitalización y la automatización, el conocimiento sobre hardware programable y sistemas embebidos se ha convertido en una herramienta fundamental para estudiantes, profesionales y entusiastas de la tecnología. La ESP32 es una plataforma versátil que permite la creación de proyectos innovadores en el campo del Internet de las Cosas (IoT), ofreciendo capacidades de comunicación inalámbrica y procesamiento eficiente en un solo dispositivo.

Este manual nace con el propósito de proporcionar una guía clara y estructurada para quienes desean explorar el potencial de la ESP32. A través de su contenido, los lectores encontrarán explicaciones detalladas sobre su configuración, programación y aplicación en proyectos prácticos. La metodología empleada busca facilitar la comprensión de conceptos complejos mediante ejemplos concretos y ejercicios aplicados.

Esperamos que este material sirva como una referencia valiosa en el aprendizaje y desarrollo de habilidades técnicas, permitiendo que los lectores amplíen sus conocimientos y fortalezcan su capacidad para diseñar soluciones tecnológicas eficientes y creativas.

Los Autores

Tabla de Contenido

Tabla de contenido

Tabla de Contenido

Tabla de Contenido

Tabla de Contenido

Introducción

En un mundo cada vez más conectado, el Internet de las Cosas (IoT) está transformando la manera en que interactuamos con los dispositivos y el entorno que nos rodea. La ESP32, una potente placa de desarrollo equipada con capacidades de WiFi y Bluetooth se ha convertido en una herramienta fundamental para aprender y desarrollar proyectos relacionados con esta tecnología emergente. Su versatilidad, bajo costo y facilidad de uso la hacen ideal tanto para principiantes como para profesionales que buscan explorar el mundo de la electrónica y la programación.

El libro "ESP32: Manual Básico para Estudiantes" está diseñado para servir como un recurso esencial para aquellos que desean iniciarse en el uso de placas de desarrollo hardware, específicamente la ESP32, y adquirir los conocimientos básicos necesarios para su programación y aplicación en proyectos de IoT. A lo largo de este manual, los estudiantes no solo aprenderán a utilizar esta herramienta tecnológica, sino que también desarrollarán competencias fundamentales para destacar en el campo del IoT y las telecomunicaciones, contribuyendo al desarrollo de soluciones innovadoras y prácticas en el mundo digital, teniendo como principales objetivos:

- Proporcionar una base sólida: Introducir a los estudiantes en conceptos básicos de programación, algoritmos y electrónica aplicada, sentando las bases para proyectos más avanzados.

Facilitar la transición al uso de la ESP32: Presentar las características principales de la placa, su configuración inicial y sus aplicaciones prácticas de manera clara y accesible.

- Enseñar mediante la práctica: Incluir ejercicios y proyectos básicos que permitan a los estudiantes aplicar lo aprendido de forma práctica y significativa.
- Fomentar el desarrollo de habilidades: Estimular el pensamiento lógico, la resolución de problemas y la creatividad en el diseño de proyectos.

El manual está organizado en ocho capítulos que guían al lector desde los conceptos más básicos hasta la implementación de proyectos integrados:

1. Introducción a la Programación y Construcción de Algoritmos: Familiariza al estudiante con los fundamentos de la programación y la lógica de algoritmos.

2. Arduino IDE: Herramienta de Programación para ESP32: Explica cómo instalar, configurar y utilizar el entorno de desarrollo Arduino IDE para programar la ESP32.

3. Introducción a las Placas de Desarrollo ESP32: Presenta las características técnicas de la ESP32 y su configuración inicial.

4. Introducción a Sensores y Actuadores: Explora los componentes electrónicos básicos y su integración con la ESP32.

5. Comunicación en Serie y Monitorización de Datos: Enseña cómo establecer comunicación entre la ESP32 y una computadora para depurar y monitorizar proyectos.

6. Uso de Pines ADC y PWM en la ESP32: Explica cómo utilizar los pines analógicos y de modulación por ancho de pulso para controlar dispositivos.

7. Proyectos Básicos con Sensores y Actuadores: Propone proyectos sencillos que combinan sensores y actuadores para aplicaciones prácticas.

8. Proyectos Básicos Integrados con la ESP32: Presenta proyectos más complejos que integran múltiples componentes y funcionalidades de la ESP32.

Este libro está dirigido a estudiantes de primeros niveles de carreras tecnológicas, como Redes y Telecomunicaciones, Electrónica, Informática y afines, que deseen adquirir una base sólida en programación y desarrollo de proyectos con la ESP32. No se requieren conocimientos previos en programación o electrónica, ya que el contenido está diseñado para ser accesible y progresivo.

Cada capítulo incluye explicaciones teóricas, ejemplos prácticos y actividades complementarias que permiten reforzar los conceptos aprendidos. Se recomienda seguir el orden de los capítulos, ya que cada uno construye sobre los conocimientos adquiridos en los anteriores. Se incluyen códigos de programación, diagramas de conexión y consejos prácticos para facilitar el aprendizaje.

Con este manual, los estudiantes no solo aprenderán a utilizar una herramienta tecnológica como la ESP32, sino que también desarrollarán habilidades que serán clave para cursos avanzados y para su futuro profesional en el ámbito de la tecnología.

CAPÍTULO I

Introducción y Antecedentes

ALUMNI EDITORA

Introducción a la Programación y Construcción de Algoritmos

El objetivo de este capítulo es familiarizar a los estudiantes con los fundamentos de la programación y la construcción de algoritmos, estableciendo una base sólida para trabajar con la placa ESP32 y proyectos de hardware. Los conceptos aquí revisados permitirán a los estudiantes entender cómo se estructuran los programas, cómo se toman decisiones en un código y cómo se crean soluciones mediante algoritmos.

Conceptos básicos de programación

¿Qué es un programa?

Un programa es una secuencia de instrucciones que una computadora o un microcontrolador, como la ESP32, ejecuta para realizar una tarea específica[1], permitiendo automatizar procesos, tomar decisiones basadas en datos e interactuar con el mundo físico mediante sensores y actuadores, utilizando lenguajes de programación como C++, empleado en el entorno Arduino IDE, que proporcionan una estructura comprensible para escribir estas instrucciones, las cuales pueden ir desde acciones simples, como encender y apagar un LED, hasta sistemas complejos, como un sistema operativo.

Lenguajes de programación

Existen dos tipos principales de lenguajes de programación:

- Lenguajes interpretados: Son aquellos en los que el código se ejecuta línea por línea mediante un intérprete[2]. Ejemplo: Python.
- Lenguajes compilados: El código se traduce completamente a un lenguaje de máquina antes de ejecutarse[3]. Ejemplo: C++ (utilizado en Arduino).

La programación para la ESP32 utiliza C++, un lenguaje compilado, en conjunto con el entorno de desarrollo Arduino IDE.

Estructura básica de un programa

Todo programa sigue una estructura básica dividida en tres partes:

1. Inicio: Declaraciones iniciales, como librerías y configuraciones.

2. Cuerpo: Bloques de instrucciones que realizan la tarea principal.
3. Fin: Puede incluir la finalización de procesos o simplemente el cierre del programa.

Un ejemplo básico en Arduino para encender un LED sería:

```
void setup() {
  pinMode(13, OUTPUT); // Configura el pin 13 como salida
}

void loop() {
  digitalWrite(13, HIGH); // Enciende el LED
  delay(1000);            // Espera 1 segundo
  digitalWrite(13, LOW);  // Apaga el LED
  delay(1000);            // Espera 1 segundo
}
```

Elementos fundamentales

Variables y constantes

- Variable: Es un espacio en la memoria que almacena un valor que puede cambiar durante la ejecución del programa.
- Constante: Es un valor que no cambia durante la ejecución del programa.

Ejemplo de declaración en Arduino:

```
int numero = 5;        // Variable
const float PI = 3.14; // Constante
```

Tipos de datos básicos

- Enteros (int): Números enteros, como 1, 2, -10
- Flotantes (float): Números decimales, como 3.14, -0.5
- Caracteres (char): Letras o símbolos, como 'a', '@'
- Booleanos (bool): Valores lógicos, true (verdadero) o false (falso)

Operadores aritméticos, lógicos y relacionales

- Aritméticos: `+`, `-`, `*`, `/`, `%` (módulo).

```
int suma = 5 + 3; // Resultado: 8
```

- Lógicos: && (AND), || (OR), ! (NOT).

```
if (a > 1 && b < 5) {
// Código a ejecutar si ambas condiciones son verdaderas
}
```

- Relacionales: `==`, `!=`, `<`, `>`, `<=`, `>=`.

```
if (x == 10) {
  // Código a ejecutar si x es igual a 10
}
```

Estructuras de control

Las estructuras de control permiten que los programas tomen decisiones y ejecuten tareas repetitivas de manera eficiente[3].

Condicionales

- if: Evalúa una condición y ejecuta el código si es verdadera.

```
if (x > 0) {
  // Código a ejecutar si x es mayor a 0
}
```

- if-else: Añade una alternativa si la condición es falsa.

```
if (x > 0) {
  // Código si x es mayor a 0
} else {
  // Código si x no es mayor a 0
}
```

- switch-case: Evalúa múltiples casos.

```
switch (valor) {
  case 1:
    // Código para el caso 1
    break;
  case 2:
    // Código para el caso 2
    break;
  default:
    // Código por defecto
}
```

Bucles

- for: Ejecuta un bloque de código un número específico de veces.

```
for (int i = 0; i < 10; i++) {
  // Código a ejecutar 10 veces
}
```

- while: Ejecuta el código mientras la condición sea verdadera.

```
while (x < 10) {
  // Código a ejecutar mientras x sea menor a 10
}
```

- do-while: Ejecuta el código al menos una vez y luego evalúa la condición.

```
do {
  // Código a ejecutar
} while (x < 10);
```

Introducción a funciones

¿Qué son las funciones y por qué usarlas?

Las funciones son bloques de código independientes que realizan una tarea específica y pueden reutilizarse varias veces dentro de un programa[4], lo que mejora la organización del código y facilita su comprensión y mantenimiento, permitiendo en la ESP32 realizar operaciones como calcular un promedio, controlar un LED o manejar datos de sensores, donde, por ejemplo, una función que encienda un LED puede recibir como parámetro un valor booleano que indique si debe estar encendido o apagado, además de que el uso de parámetros y valores de retorno permite que las funciones sean flexibles y adaptables a diferentes situaciones.

Sintaxis de funciones en Arduino

```
tipo_de_retorno nombre_funcion(parametros) {
  // Código de la función
  return valor;
}
```

Ejemplo:

```
int sumar(int a, int b) {
  return a + b;
}
```

Introducción a algoritmos

¿Qué es un algoritmo?

Un algoritmo es una secuencia de pasos ordenados y finitos que permiten resolver un problema o realizar una tarea de manera eficiente[5], incluyendo, por ejemplo, un procedimiento para encender un LED dependiendo de la lectura de un sensor de luz, donde se leen los valores del sensor, se comparan con un umbral y, según el resultado, se enciende o apaga el LED, pudiendo representarse mediante diagramas de flujo, que visualizan gráficamente el proceso, o en pseudocódigo, que describe el algoritmo de manera textual utilizando un lenguaje

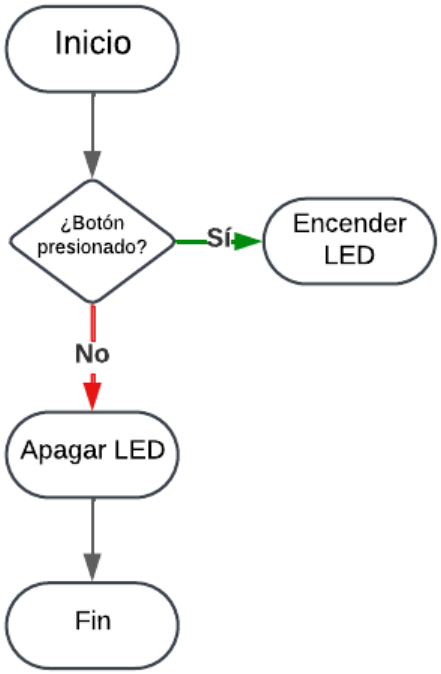
estructurado, siendo ambas herramientas valiosas para planificar y comunicar soluciones antes de implementarlas en código.

Representación de algoritmos

- Diagramas de flujo: representan gráficamente los pasos de un algoritmo.
- Pseudocódigo: representa los pasos de forma textual.

Ejemplo práctico: encender un LED dependiendo de una condición.

Diagrama de flujo:



Pseudocódigo:

```
Inicio
    Si botón está presionado
        Encender LED
    Si no
        Apagar LED
Fin
```

Código en Arduino:

```
void setup() {
  pinMode(13, OUTPUT); // Configura el pin 13 como salida
  pinMode(2, INPUT);   // Configura el pin 2 como entrada
}

void loop() {
```

```
  if (digitalRead(2) == HIGH) { // Si el botón está
presionado
    digitalWrite(13, HIGH);     // Enciende el LED
  } else {
    digitalWrite(13, LOW);      // Apaga el LED
  }
}
```

Prácticas propuestas

- **Práctica 1**: crear un algoritmo en pseudocódigo para calcular el área de un círculo basado en la entrada del radio.
- **Práctica 2**: desarrollar un diagrama de flujo para un sistema que encienda o apague un LED según un switch.
- **Práctica 3**: escribir un programa en Arduino que lea un valor de entrada de un potenciómetro y ajuste la intensidad de un LED.

Resumen del capítulo

En este capítulo, se introdujeron los conceptos fundamentales de programación y construcción de algoritmos, esenciales para trabajar con la ESP32, cubriendo temas como la estructura básica de un programa, los tipos de datos, operadores, estructuras de control (condicionales y bucles), y la creación de funciones.

Se exploró la importancia de los algoritmos y su representación mediante diagramas de flujo y pseudocódigo, con lo que se sentaron las bases para la programación de la ESP32, permitiendo a los estudiantes entender cómo diseñar soluciones eficientes y estructuradas

Autoevaluación

Preguntas teóricas:

- ¿Qué es un algoritmo y por qué es importante en la programación?
- Explica la diferencia entre un lenguaje interpretado y un lenguaje compilado.
- Describe las partes principales de un programa en Arduino (estructura básica).

Ejercicios prácticos:

- Escribe un pseudocódigo para un programa que encienda un LED si un botón está presionado y lo apague si no lo está.
- Crea un diagrama de flujo para un algoritmo que calcule el promedio de tres números ingresados por el usuario.

CAPÍTULO II

Arduino IDE: Herramienta de Programación para ESP32

ALUMNI EDITORA

Introducción al Arduino IDE Herramienta de Programación para ESP32

El objetivo de este capítulo es introducir a los estudiantes al entorno de desarrollo Arduino IDE, que se utiliza para programar y ejecutar códigos en la ESP32. Los estudiantes aprenderán a instalar, configurar y usar las herramientas básicas del IDE, además de practicar con ejemplos sencillos para interactuar con la placa ESP32.

¿Qué es el Arduino IDE?

El **Arduino IDE** (Integrated Development Environment) es una herramienta de desarrollo de código abierto utilizada para escribir, cargar y depurar programas en diversas placas de desarrollo, incluyendo la ESP32[6]. Es ampliamente usado por su facilidad de uso y compatibilidad con múltiples plataformas y placas.

El IDE permite escribir código en un lenguaje basado en C/C++, además de gestionar librerías y herramientas necesarias para programar microcontroladores[6]. Algunas de sus características principales son:

- Interfaz simple y amigable.
- Verificación de errores en el código antes de cargarlo.
- Comunicación con las placas mediante un puerto USB.
- Monitor serie para recibir y enviar datos durante la ejecución del programa.

Instalación en diferentes sistemas operativos

Para instalar el Arduino IDE, sigue estos pasos según tu sistema operativo:

1. **Windows:**
- Descarga el instalador desde la página oficial: https://www.arduino.cc.
- Ejecuta el instalador y sigue las instrucciones. Asegúrate de marcar la opción para instalar los drivers USB.
2. **macOS:**
- Descarga el archivo `.dmg` desde la página oficial.
- Arrastra el ícono de Arduino IDE a la carpeta `Aplicaciones`.

3. **Linux:**

- Descarga el archivo `.tar.xz` desde la página oficial.
- Extrae el contenido y ejecuta el script de instalación en una terminal: `sudo ./install.sh`

Configuración para trabajar con ESP32

Para utilizar una placa ESP32 en el **Arduino IDE**, se deben realizar varias configuraciones iniciales para asegurarse de que el entorno reconozca y permita programar esta placa correctamente, por lo que se explica el proceso a detalle:

Agregar el soporte para ESP32

El Arduino IDE no incluye soporte para ESP32 por defecto, por lo que es necesario agregarlo siguiendo estos pasos:

a) Configurar la URL del Gestor de Tableros

1. Abre el Arduino IDE.
2. Ve a **Archivo > Preferencias** (en macOS, está en **Arduino > Preferencias**).
3. En el campo **URL Adicionales para Gestor de Tableros**, agrega la siguiente URL: `https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json`
    - Si ya tienes otras URLs configuradas, sepáralas con una coma.
4. Haz clic en **OK**.

b) Instalar el paquete ESP32

1. Ve a **Herramientas > Placa > Gestor de Placas**.
2. En la barra de búsqueda, escribe `esp32`.
3. Selecciona el paquete que dice `esp32 by Espressif Systems` y haz clic en **Instalar**.
4. Espera a que termine la instalación.

Seleccionar la placa ESP32

1. Ve a **Herramientas > Placa > ESP32 Arduino.**

2. Selecciona el modelo específico de tu ESP32 (por ejemplo, `ESP32 Dev Module`, que es el más común).

Configurar el puerto serial

1. Conecta la ESP32 al ordenador usando un cable micro-USB o USB-C (dependiendo del modelo).
2. Asegúrate de que el cable sea **de datos**, no solo de carga.
3. Ve a **Herramientas > Puerto** y selecciona el puerto COM donde está conectado tu ESP32.
    - En sistemas Windows, el puerto suele aparecer como `COMx` (por ejemplo, COM3).
    - En macOS o Linux, aparece como `/dev/cu.SLAB_USBtoUART` o similar.

Instalar el controlador USB-UART (opcional)

Si el puerto no aparece:

1. Algunas placas ESP32 utilizan chips como el **CP2102** o el **CH340** para la comunicación serial.
2. Descarga e instala el controlador correspondiente:
    - **CP2102**: https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers
    - **CH340**: https://sparks.gogo.co.nz/ch340.html
3. Reinicia el Arduino IDE después de instalar el controlador.

Solución de problemas comunes

- **Error: "Failed to connect to ESP32: Timed out waiting for packet header"**
    - Mantén presionado el botón **BOOT** en la placa ESP32 mientras subes el código.
    - Suelta el botón una vez que veas que comienza la carga.
- **Puerto no detectado**
    - Verifica los controladores USB-UART y el cable de conexión.
- **No se enciende la placa**

- Asegúrate de que el cable USB sea funcional y proporcione alimentación.

Estructura básica de un programa en Arduino

La estructura básica de un programa en Arduino es muy sencilla y consta de dos funciones principales, `setup()` y `loop()`, donde cada una cumple un propósito específico y resulta fundamental para cualquier programa en Arduino, definiendo en `setup()` la configuración inicial del hardware y en `loop()` la ejecución repetitiva del código durante el funcionamiento del dispositivo[6].

```
// Declaración de variables y bibliotecas globales

void setup() {
// Configuración inicial: Se ejecuta solo una vez al iniciar el programa.
}

void loop() {
// Código principal: Se ejecuta de manera continua en un bucle infinito.
}
```

Partes de un programa de Arduino

Comentarios: son utilizados para documentar el código y explicar qué hace cada sección, estos pueden ser de una línea (con `//`) o de múltiples líneas (con `/* */`).

```
// Este es un comentario de una sola línea.
/*
   Este es un comentario
   de varias líneas.
*/
```

Declaración de bibliotecas (opcional): si el programa utiliza funcionalidades específicas (sensores, comunicación, etc.), se deben incluir bibliotecas con la palabra clave `#include`, ejemplo:

```
#include <Wire.h>            //Biblioteca para comunicación I2C
#include <Adafruit_Sensor.h> //Biblioteca para sensores Adafruit
```

Definición de constantes y variables globales: Aquí se definen las variables y constantes que estarán disponibles en todo el programa, por ejemplo:

```
const int ledPin = 13; // Constante para el pin del LED
int contador = 0;      // Variable global
```

Función `setup()`: se utiliza para inicializar el hardware y las configuraciones necesarias antes de que el programa entre en funcionamiento, ejecutándose solo una vez al inicio del programa o cuando el microcontrolador se reinicia.

```
void setup() {
  pinMode(13, OUTPUT); // Configura el pin 13 como salida
  Serial.begin(9600); // Inicia la comunicación serial a 9600 baudios
}
```

Función `loop()`: contiene el código que se ejecuta repetidamente mientras el programa esté en funcionamiento, por lo que se ejecuta en un bucle infinito.

```
void loop() {
  digitalWrite(13, HIGH); // Enciende el LED
  delay(1000);            // Espera 1 segundo
  digitalWrite(13, LOW);  // Apaga el LED
  delay(1000);            // Espera 1 segundo
}
```

Herramientas del Arduino IDE

El Arduino IDE es una plataforma de desarrollo que ofrece un conjunto de herramientas diseñadas para facilitar la programación y gestión de placas de desarrollo como Arduino, ESP32 y otras, permitiendo realizar configuraciones esenciales, como seleccionar la placa y el puerto de conexión, ajustar la velocidad de carga y gestionar bibliotecas y paquetes específicos, e incluyendo funciones avanzadas, como el uso del Monitor Serie para depuración, el Trazador Serie para visualización de datos en tiempo real y opciones para quemar el bootloader en caso de necesitar reconfigurar la placa[7]. Este conjunto de herramientas hace del Arduino IDE una plataforma flexible y eficiente para la creación de proyectos basados en microcontroladores.

Placa

La herramienta "Placa" permite seleccionar el modelo específico de la placa que se está utilizando, asegurando que el código se compile correctamente y se cargue en el hardware de acuerdo con sus especificaciones, con un menú que incluye opciones como Arduino Uno, Arduino Mega, Arduino Nano y ESP32[7], entre otras, donde, por ejemplo, si se trabaja con una ESP32, es necesario seleccionar "ESP32 Dev Module". Elegir correctamente la placa es crucial para evitar problemas de compatibilidad al cargar el programa.

Puerto

La opción "Puerto" se utiliza para seleccionar el puerto COM al que está conectada la placa, identificándose en sistemas Windows como COMx (por ejemplo, COM3) y en macOS y Linux como rutas similares a /dev/cu.SLAB_USBtoUART, siendo fundamental elegir el puerto correcto para garantizar una comunicación fluida entre la computadora y la placa[8]. Si el puerto no aparece, se recomienda verificar que los controladores USB correspondientes estén instalados correctamente.

Procesador

El procesador puede seleccionarse en ciertas placas que tienen diferentes variantes de hardware[8], como el Arduino Nano, donde, por ejemplo, es posible elegir entre procesadores como ATmega328P o ATmega328P (Old Bootloader), siendo esencial seleccionar el procesador correcto para garantizar que el código sea compatible con las especificaciones del hardware.

Programador

El "Programador" define el dispositivo o software utilizado para cargar el programa en la placa, con opciones comunes como AVRISP mkII, utilizado en muchas placas Arduino, y, en la mayoría de los casos, no es necesario cambiar esta configuración, ya que el IDE selecciona automáticamente el programador más adecuado para las placas estándar[6].

Velocidad de carga

La "Velocidad de carga" establece la velocidad en baudios para la comunicación entre el Arduino IDE y la placa al momento de cargar un programa, con valores comunes como 115200, utilizado como predeterminado en la mayoría de las placas, y 9600, que puede ser requerido por microcontroladores más básicos, por lo que, si se presentan errores al cargar un programa, reducir la velocidad de carga puede ser una solución eficaz.

Monitor Serie

El "Monitor Serie" es una consola que permite observar y enviar datos en tiempo real entre la computadora y la placa, siendo una herramienta muy útil

para depurar programas, especialmente cuando se trabaja con sensores y dispositivos que generan datos dinámicos, y permitiendo configurar la velocidad de baudios para que coincida con la utilizada en el programa de la placa, garantizando así una comunicación fluida.

Trazador Serie

El "Trazador Serie" permite graficar en tiempo real los datos enviados desde la placa a través de la comunicación serial, siendo especialmente útil para visualizar lecturas de sensores o variables de interés de manera gráfica, y requiriendo que los datos enviados estén formateados adecuadamente para su correcta interpretación.

Quemar Bootloader

La opción "Quemar Bootloader" se utiliza para instalar o reinstalar el bootloader en la placa, un pequeño programa esencial que permite cargar el código desde el Arduino IDE a través de la conexión USB[8], siendo útil cuando la placa no responde al intentar cargar programas o cuando se necesita reconfigurarla tras usar un programador externo.

Configuraciones avanzadas (en placas específicas)

Algunas placas, como la ESP32, ofrecen configuraciones avanzadas adicionales, incluyendo la selección de la frecuencia de la CPU (por ejemplo, 80 MHz, 160 MHz o 240 MHz), la definición del esquema de partición de la memoria (partition scheme) y los ajustes del modo o tamaño de la memoria Flash[7]. Estas opciones permiten optimizar el rendimiento y la capacidad de la placa para proyectos específicos.

Gestor de Placas

El "Gestor de Placas" es una herramienta que permite instalar y administrar los paquetes necesarios para trabajar con diferentes placas en el Arduino IDE, desde la cual se pueden buscar e instalar controladores para nuevas placas, como ESP32 o ESP8266, siendo importante contar con conexión a Internet para descargar los paquetes requeridos.

Gestor de Bibliotecas

El "Gestor de Bibliotecas" facilita la instalación y gestión de bibliotecas que amplían las capacidades del Arduino IDE[6], conteniendo funciones y ejemplos predefinidos para trabajar con sensores, actuadores, módulos de comunicación y otros dispositivos, permitiendo buscar bibliotecas específicas, como "Adafruit_Sensor", e instalarlas directamente desde esta herramienta.

Preferencias

El menú "Preferencias" permite personalizar el entorno del Arduino IDE, donde se puede definir la ubicación para guardar los programas, activar la salida detallada durante la compilación y carga, o agregar URLs adicionales para instalar nuevas placas mediante el gestor de placas, lo que permite ajustar el IDE a las necesidades específicas del usuario y del proyecto.

Actividades complementarias

- **Actividad 1**: investigación sobre placas compatibles:
    - Elaborar un informe sobre las principales placas compatibles con el Arduino IDE incluyendo una tabla comparativa entre diferentes placas.
- **Actividad 2**: exploración de bibliotecas:
    - Buscar e identificar al menos cinco de las bibliotecas más utilizadas y elaborar una descripción breve de cada una, indicando su utilidad, los tipos de proyectos en los que se puede emplear y las funciones principales que ofrece.
- **Actividad 3**: guía ilustrada del Arduino IDE:
    - Diseñar una guía visual que explique cada herramienta del menú "Herramientas" del Arduino IDE. La guía debe incluir capturas de pantalla, una breve descripción de cada opción y ejemplos prácticos de cómo y cuándo utilizarlas.

Resumen del capítulo

Este capítulo se centró en el entorno de desarrollo Arduino IDE, explicando cómo instalarlo, configurarlo para trabajar con la ESP32, y utilizar sus herramientas principales, detallando la estructura básica de un programa en Arduino, incluyendo las funciones setup() y loop(), y se exploraron herramientas como el monitor serie, el trazador serie y el gestor de bibliotecas.

Los estudiantes aprendieron a cargar programas en la ESP32 y a utilizar el IDE para depurar y monitorizar sus proyectos, lo que les permitió familiarizarse con el entorno de desarrollo que utilizarán a lo largo del manual.

Autoevaluación

Preguntas teóricas:

- ¿Qué es el Arduino IDE y cuáles son sus principales características?
- Explica cómo configurar el Arduino IDE para trabajar con la ESP32.
- ¿Para qué se utilizan las funciones setup() y loop() en un programa de Arduino?

Desafío:

- Investiga cómo instalar una biblioteca en el Arduino IDE y úsala para controlar un sensor DHT11.

CAPÍTULO III

Introducción a las Placas de Desarrollo ESP32

ALUMNI EDITORA

Introducción a las Placas de Desarrollo ESP32

El objetivo de este capítulo es presentar a los estudiantes la placa de desarrollo ESP32, sus características principales, su configuración inicial y las ventajas que ofrece como herramienta fundamental para proyectos de Internet de las Cosas (IoT). Este capítulo servirá como punto de partida para que los estudiantes comprendan el hardware físico y aprendan a interactuar con la ESP32 desde el entorno de desarrollo.

¿Qué es la ESP32?

Breve historia y desarrollo

La ESP32 es una placa de desarrollo basada en un microcontrolador de bajo costo y alto rendimiento, desarrollado por Espressif Systems, lanzada como sucesora de la popular ESP8266, ofreciendo mayores capacidades de procesamiento, conectividad y flexibilidad, lo que la convierte en una opción ideal para proyectos IoT, donde se requiere conectividad inalámbrica y un consumo eficiente de energía[9].

La ESP32 ha revolucionado el mundo de las placas de desarrollo gracias a su bajo costo, facilidad de uso y capacidades avanzadas, lo que ha impulsado su popularidad tanto en entornos educativos como en aplicaciones comerciales, como se puede observar en la tabla 1, donde se detallan las diferencias con otras placas de desarrollo.

Tabla 1
Diferencias entre ESP32 y otras placas de desarrollo.

Característica	ESP32	ESP8266	Arduino UNO
Procesador	Dual-core Xtensa	Single-core Xtensa	AVR de 8 bits
Velocidad de reloj	Hasta 240 MHz	Hasta 160 MHz	16 MHz
Memoria RAM	~520 KB	~160 KB	2 KB
Conectividad	WiFi y Bluetooth	WiFi	No incluye
Pines GPIO	30-36	17	14
Periféricos	UART, SPI, I2C, ADC	UART, SPI, I2C	UART, SPI, I2C

Fuente: adaptado de elosciloscopio.com [10].

Como se observa, la ESP32 ofrece un rendimiento ampliamente superior a sus predecesores y otras placas, siendo ideal para proyectos que requieren conectividad inalámbrica y procesamiento en tiempo real.

Características técnicas

La documentación oficial de la placa ESP32[9] detalla todas sus características técnicas, siendo las principales las siguientes:

Procesador y memoria

- Procesador dual-core Xtensa LX6: permite ejecutar múltiples tareas simultáneamente.
- Velocidad de reloj: hasta 240 MHz, mucho mayor que la mayoría de las placas tradicionales.
- Memoria RAM: 520 KB, suficiente para manejar tareas complejas en proyectos IoT.

Conectividad

- WiFi: compatible con redes 802.11 b/g/n, permitiendo conectarse a redes locales e internet.
- Bluetooth: incluye Bluetooth 4.2 y BLE (Bluetooth Low Energy), útil para dispositivos de baja energía y comunicación directa entre dispositivos.

Pines de entrada/salida (GPIO)

- La ESP32 tiene entre 30 y 36 pines GPIO, dependiendo del modelo, estos pines permiten conectar sensores, actuadores y otros dispositivos.
- ADC (Analog to Digital Converter): convierte señales analógicas (como del sensor de temperatura) a digitales.
- PWM (Pulse Width Modulation): genera señales de pulsos para controlar dispositivos como motores o LEDs con brillo variable.

Periféricos integrados

- UART: comunicación serie (como con el monitor serie del Arduino IDE).
- SPI: comunicación con dispositivos como pantallas o memorias externas.
- I2C: protocolo de comunicación para sensores y módulos externos.

Configuración inicial de la ESP32

Para comenzar a trabajar con la ESP32, es necesario realizar una configuración inicial que garantice su correcta operación, teniendo como antecedentes la instalación del Arduino IDE, los driver y librerías necesarias (proceso detallado en el Capítulo II).

Materiales necesarios

- **Placa ESP32**: cualquier modelo funcional (por ejemplo, ESP32 DevKit V1).
- **Cable USB**: preferentemente un cable Micro-USB o USB-C, dependiendo de la placa.
- **Computadora**: con sistema operativo Windows, macOS o Linux.
- **Arduino IDE**: el entorno de desarrollo donde programaremos la ESP32.

Subir un programa básico

El primer programa que subiremos será un clásico: hacer parpadear el LED integrado de la ESP32, para lo cual escribiremos el siguiente código:

```
void setup() {
  pinMode(2, OUTPUT); // Configura el pin 2 como salida
(LED integrado)
}

void loop() {
  digitalWrite(2, HIGH); // Enciende el LED
  delay(1000);           // Espera 1 segundo
  digitalWrite(2, LOW);  // Apaga el LED
  delay(1000);           // Espera 1 segundo
}
```

1. Copia este programa en el Arduino IDE.
2. Haz clic en el botón Subir (flecha hacia la derecha) para cargarlo en la ESP32.
3. Observa cómo el LED integrado parpadea.

Prácticas propuestas

- **Práctica 1**: identificar las características de la ESP32.
  - Investiga y documenta las especificaciones técnicas de tu placa ESP32.
  - Identifica visualmente los siguientes componentes:
    - Pines GPIO.
    - LED integrado.
    - Chip principal.
    - Conector USB.
  - Completa una tabla con las características de tu placa.
- **Práctica 2**: instalar el entorno de desarrollo y subir el primer programa.
  - Instala los drivers y el soporte para ESP32 en el Arduino IDE.
  - Escribe y sube el programa de parpadeo del LED.
  - Describe el proceso realizado, incluyendo problemas encontrados y cómo los solucionaste.
- **Práctica 3**: configurar la comunicación serie.
  - Escribe un programa que envíe mensajes desde la ESP32 al monitor serie.

```
void setup() {
  Serial.begin(115200); // Inicia la comunicación serie a 115200
bps
  Serial.println("¡Hola, ESP32!"); // Envía un mensaje inicial
}

void loop() {
  Serial.println("La ESP32 está funcionando correctamente.");
  delay(2000); // Espera 2 segundos
}
```

  - Sube el programa y abre el monitor serie en el Arduino IDE.

- Observa los mensajes enviados por la ESP32.

Resumen del capítulo

En este capítulo, se presentó la placa de desarrollo ESP32, sus características técnicas y su configuración inicial, explicando aspectos clave como el procesador dual-core, la conectividad WiFi y Bluetooth, los pines GPIO, y los periféricos integrados.

Los estudiantes aprendieron a realizar la configuración inicial de la ESP32, incluyendo la instalación de los controladores necesarios y la carga de un programa básico para hacer parpadear un LED, sentando las bases para entender el hardware de la ESP32 y cómo interactuar con él.

Autoevaluación

Preguntas teóricas:

- Describe las principales características técnicas de la ESP32 (procesador, memoria, conectividad).
- ¿Cuál es la diferencia entre la ESP32 y otras placas como el Arduino UNO o la ESP8266?
- Explica cómo se configura la comunicación serial en la ESP32.

Desafío:

- Investiga cómo cambiar la frecuencia del procesador de la ESP32 y escribe un programa que utilice una frecuencia de 160 MHz.

CAPÍTULO IV

Introducción a Sensores y Actuadores

ALUMNI EDITORA

Introducción a sensores y actuadores

El objetivo de este capítulo es familiarizar a los estudiantes con los principios básicos del funcionamiento de sensores y actuadores, así como su conexión e integración con la placa ESP32, este conocimiento les permitirá comprender cómo interactuar con el entorno físico y crear proyectos prácticos utilizando elementos básicos de entrada y salida.

¿Qué son los sensores y actuadores?

Los sensores y actuadores son componentes fundamentales en cualquier sistema de hardware automatizado.

Un sensor es un dispositivo que detecta cambios en el entorno físico y los convierte en señales eléctricas que pueden ser interpretadas por un microcontrolador[11], como la ESP32; por ejemplo, un sensor de temperatura mide la temperatura ambiente y la traduce en un valor digital que la ESP32 puede procesar, permitiendo que los sistemas electrónicos perciban el entorno.

Por otro lado, un actuador es un dispositivo que realiza acciones físicas en respuesta a una señal eléctrica o comando de un microcontrolador[11]; por ejemplo, un motor puede girar en respuesta a una señal, o un relé puede encender o apagar un electrodoméstico, permitiendo que los sistemas electrónicos interactúen con el entorno y generen movimiento, luz, sonido u otras acciones.

En aplicaciones cotidianas, los sensores y actuadores trabajan juntos; por ejemplo, en un aire acondicionado, un sensor de temperatura detecta si el ambiente está caliente o frío, y un actuador, como el compresor, ajusta la temperatura en consecuencia. Otro ejemplo es una puerta automática, que utiliza un sensor de movimiento para detectar personas y un motor para abrir o cerrar la puerta.

Tipos de sensores comunes

Sensor de temperatura y humedad DHT11/22

Uno de los sensores más utilizados en proyectos básicos es el sensor de temperatura y humedad, como el DHT11 o el DHT22[12]. Estos sensores son económicos y fáciles de usar, ofreciendo datos precisos sobre la temperatura y la humedad relativa del entorno.

Mientras que el DHT11 es adecuado para mediciones generales, el DHT22 proporciona mayor precisión y un rango más amplio de medición como lo muestran sus hojas de datos.

- **DHT11**: es más económico, pero tiene una menor precisión (±2 °C para temperatura y ±5 % para humedad) y un rango más limitado de medición.
- **DHT22**: es más preciso (±0.5 °C para temperatura y ±2 % para humedad) y tiene un rango de medición más amplio.

La conexión con la ESP32 tanto para el DHT11 como para el DHT22 es similar, ya que los dos sensores tienen un pin de datos (DATA) que debe conectarse a un GPIO de la placa (por ejemplo, el pin GPO25), la alimentación de VCC y GND puede ser a una batería externa o a los pines específicos de la ESP32.

Sensor de luz (LDR - Light Dependent Resistor)

Un **LDR** es un sensor que mide la intensidad de la luz, su resistencia disminuye cuando la luz aumenta, lo que permite obtener valores proporcionales a la cantidad de luz que incide sobre él[13], haciendo que este tipo de sensor sea ideal para proyectos como lámparas automáticas que se encienden cuando oscurece o sistemas de monitoreo de luz ambiental.

El LDR se conecta a un pin analógico, en el caso de la ESP32 debe ser un pin ADC como el GPIO 27, para leer los cambios en la intensidad de luz y se utiliza una resistencia pull-down o pull-up para completar el circuito, generalmente de 10kΩ.

Sensor ultrasónico (HC-SR04)

Los sensores ultrasónicos utilizan ondas de sonido para medir distancias, en este caso, el módulo HC-SR04 emite un pulso ultrasónico y mide el tiempo que tarda en regresar tras reflejarse en un objeto, lo cual lo hace ideal para sistemas robóticos y de detección de obstáculos[14].

El sensor tiene un pin de "trigger" (para emitir el pulso) y un pin de "echo" (para detectar el retorno), ambos se conectan a pines digitales de la ESP32.

Sensor de gas (MQ series)

Los sensores de gas, como los modelos **MQ-2**, MQ-3**, MQ-5** y **MQ-135**, detectan la presencia de gases específicos en el ambiente, como gas butano, metano, dióxido de carbono o monóxido de carbono, por lo que son esenciales en sistemas de seguridad y monitoreo ambiental, ya que permiten detectar fugas de gas en el hogar, monitorear la calidad del aire y brindar alarmas de humo y fuego[15].

Estos sensores pueden proporcionar una salida analógica o digital dependiendo del modelo, por lo que pueden conectarse a pines digitales o analógicos (ADC).

Sensor de movimiento (PIR - Passive Infrared)

El sensor PIR detecta el movimiento de objetos que emiten radiación infrarroja, como los humanos o animales, que activa su salida cuando detecta un cambio en la radiación infrarroja de su entorno, es comúnmente utilizado en sistemas de seguridad y alarmas, iluminación automática y seguimiento de actividad en el hogar[16].

Su conexión con las placas de desarrollo ESP32 es sencilla ya que pin de salida del sensor se conecta a un pin digital de la ESP32, así cuando se detecta movimiento, se envía un pulso HIGH para generar la alerta o activar un actuador.

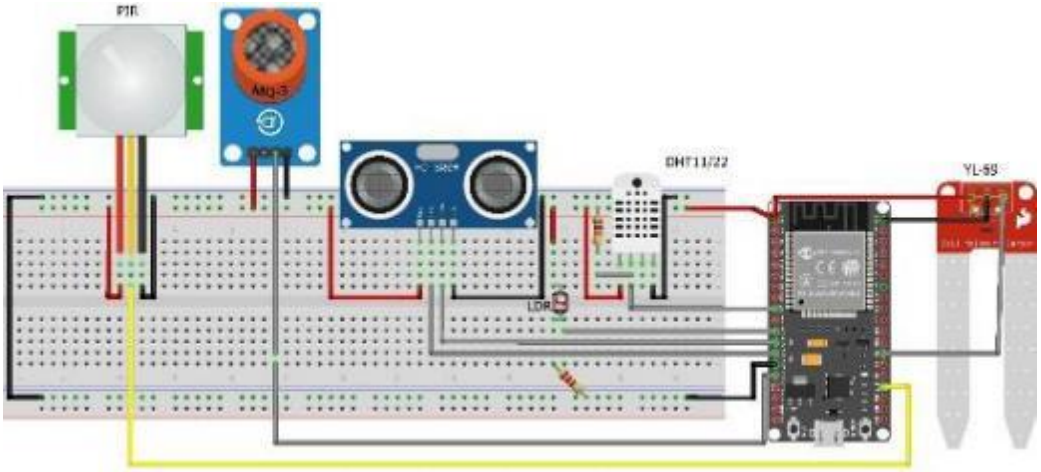
Sensor de humedad del suelo (YL-69)

Este sensor mide el nivel de humedad en el suelo, lo que lo hace ideal para proyectos de jardinería y agricultura inteligente, se conforma de dos sondas que miden la conductividad eléctrica del suelo y se utiliza en sistemas de riego automático, monitoreo de plantas en interiores y agricultura de precisión[17].

Este sensor puede ofrecer salidas analógicas o digitales para medir el nivel exacto de humedad.

En la siguiente figura se muestra la conexión típica de estos sensores a la placa ESP32:

Figura 1
Conexión de sensores en la ESP32.



Tipos de actuadores comunes

Pantallas LCD y OLED

Las pantallas son actuadores que permiten mostrar datos o información visual al usuario, siendo las **LCD (Liquid Crystal Display)** y **OLED (Organic Light Emitting Diode)** dos de las más utilizadas en proyectos con la ESP32.

La conexión puede variar dependiendo del tipo de pantalla, pero es recomendable utilizar módulos I2C para facilitar la comunicación.

- **LCD (16x2 o 20x4):** Pantallas básicas de texto.
- **OLED:** Pantallas más avanzadas que permiten gráficos y texto con mayor resolución.

Zumbadores (Buzzers)

Los zumbadores son actuadores que generan sonido cuando reciben una señal eléctrica, siendo comúnmente utilizados para emitir alertas o tonos en alarmas de seguridad, indicadores sonoros en dispositivos y generación de tonos musicales. Su conexión es sencilla ya que necesitan únicamente un pin digital o PWM.

Relés

Un **relé** es un interruptor eléctrico[11] controlado mediante una señal de bajo voltaje, lo que permite encender o apagar dispositivos de alto voltaje, como electrodomésticos o luces, esto los hace imprescindibles en la automatización del

hogar, control de dispositivos de alto voltaje y sistemas de seguridad. Para su activación o desactivación necesita conectarse a un pin digital de la ESP32.

LEDs y tiras LED

Los LEDs (diodos emisores de luz) son actuadores simples, pero muy útiles, que generan luz cuando se les aplica corriente, se pueden usar para indicar estados del sistema o como salida visual en proyectos, mientras que las tiras LED permiten efectos más avanzados, como iluminación RGB controlada dinámicamente. Ambos se conectan a pines digitales de la placa de desarrollo.

Motores (servo, DC, paso a paso)

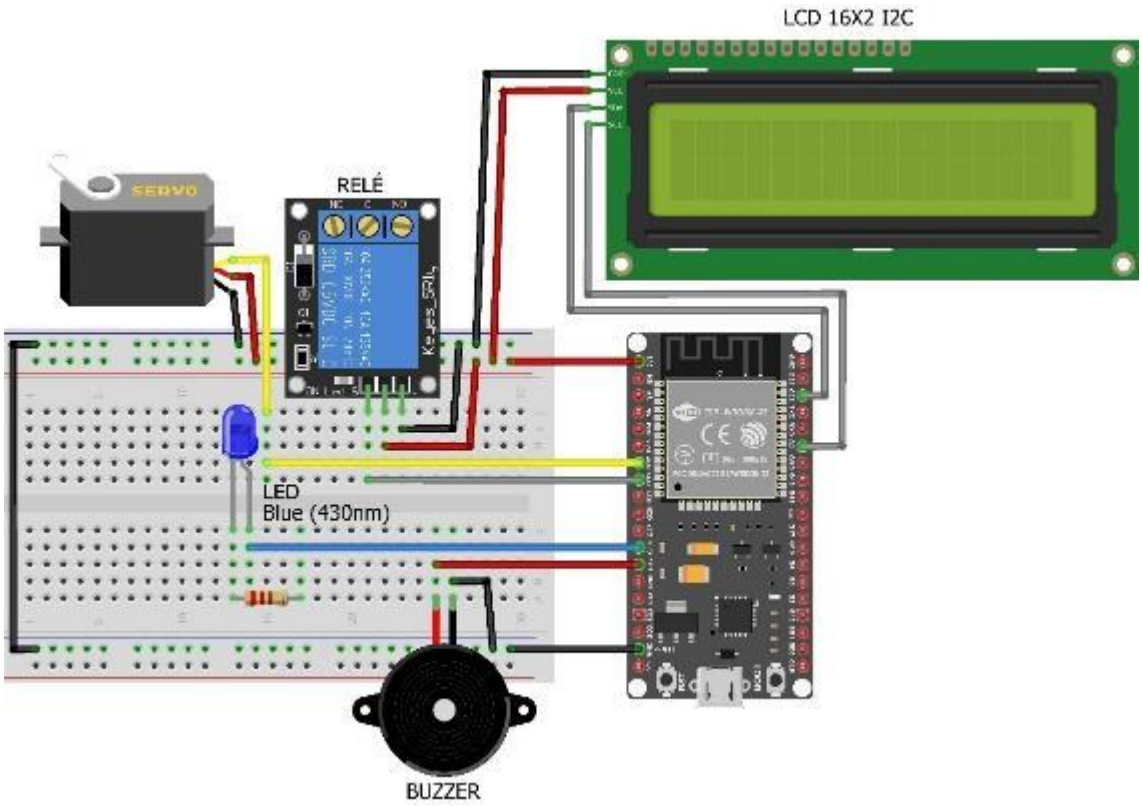
Servomotores: Son motores que permiten controlar el ángulo de rotación con precisión, generalmente entre 0° y 180°, siendo útiles en aplicaciones donde se necesita posicionamiento, como brazos robóticos y control de dirección en vehículos.

Motores DC: Son motores de corriente continua que giran a gran velocidad y son fáciles de controlar, por lo que se utilizan en ventiladores, carros o bombas.

Motores paso a paso: Permiten un control más preciso del movimiento, avanzando en pasos pequeños, lo que los hace ideales para impresoras 3D y sistemas de posicionamiento.

En la figura 2 se muestra la conexión típica de estos actuadores a la placa ESP32:

Figura 2
Conexión de actuadores en la ESP32



Importancia de Sensores y Actuadores en Proyectos IoT

Los sensores y actuadores son los componentes clave para que los dispositivos IoT interactúen con el entorno, esto debido a que los sensores recopilan datos del mundo físico, como temperatura, luz o movimiento, mientras que los actuadores permiten que el sistema tome acciones, como encender una luz, mover un motor o activar una alarma.

La combinación de ambos en un proyecto permite crear soluciones automatizadas, inteligentes y eficientes, por ejemplo: en un sistema de riego automatizado, un sensor de humedad detecta si el suelo está seco y una electroválvula permite el riego; en un sistema de seguridad, un sensor PIR detecta movimiento y un zumbador activa una alarma.

Con la **ESP32**, la integración de sensores y actuadores es sencilla gracias a sus múltiples pines GPIO[9], compatibilidad con protocolos de comunicación y soporte para bibliotecas que facilitan la programación, esto permite que los

estudiantes y desarrolladores creen sistemas interactivos y complejos con facilidad.

Lectura y escritura de datos con la ESP32

La ESP32 es una placa de desarrollo que facilita la interacción con el entorno físico gracias a su capacidad de leer señales provenientes de sensores (entrada) y enviar comandos a actuadores (salida)[9]. La lectura y escritura de datos son procesos esenciales para establecer esta comunicación, y se realizan utilizando los pines digitales y analógicos de la placa

Lectura de datos digitales

Los sensores digitales generan señales binarias, es decir, valores de 0 o 1, que representan estados como encendido/apagado o activado/desactivado, para leer datos digitales, se conectan los sensores a los pines GPIO configurados como entrada; por ejemplo, un sensor de movimiento PIR emite una señal alta (1) cuando detecta movimiento y una señal baja (0) cuando no lo hace, el método digitalRead(pin) se utiliza para capturar estos valores en el código.

Lectura de datos analógicos

Los sensores analógicos generan valores continuos dentro de un rango, como 0-3.3 V en el caso de la ESP32, estos valores representan magnitudes físicas como temperatura, luz o presión.

La ESP32 cuenta con varios pines ADC[9] (Convertidores Analógico-Digital) que convierten estas señales en valores digitales que el microcontrolador puede procesar, para leer datos de un sensor analógico, como un potenciómetro o un sensor de luz LDR, se utiliza el método analogRead(pin), este valor leído depende de la resolución del ADC, que en la ESP32 es de 12 bits por defecto, lo que permite obtener valores entre 0 y 4095.

Escritura en pines digitales

Los actuadores digitales, como relés o LEDs, reciben señales binarias para operar, por lo que un LED puede encenderse o apagarse enviando un valor alto o

bajo al pin correspondiente mediante el método digitalWrite(pin, valor), lo que permite controlar dispositivos de manera simple y eficiente.

Escritura de señales PWM (Pulse Width Modulation)

La modulación por ancho de pulso (PWM) se utiliza para controlar dispositivos que requieren señales de salida variables, como motores y tiras LED.

La ESP32 incluye pines capaces de generar señales PWM, que permiten variar la intensidad de luz de un LED o ajustar la posición de un servomotor, para ello, se utiliza la función ledcWrite(channel, dutyCycle)[18], donde se configura un canal específico y un ciclo de trabajo que define la proporción de tiempo en que la señal está activa.

Comunicación con dispositivos avanzados

Algunos actuadores, como pantallas LCD, requieren protocolos de comunicación específicos, como I2C o SPI[19].

La ESP32 admite estos protocolos, lo que permite enviar comandos complejos y manejar dispositivos más sofisticados, por lo que se pueden escribir datos en un display OLED utilizando funciones provistas por bibliotecas específicas como Adafruit_SSD1306.

Uso de bibliotecas

El ecosistema de bibliotecas disponibles para la ESP32 en el Arduino IDE facilita enormemente el proceso de lectura y escritura, ya que estas bibliotecas incluyen funciones predefinidas que abstraen los detalles técnicos, permitiendo trabajar con sensores y actuadores de manera más eficiente, algunos ejemplos son:

- Biblioteca DHT.h: permite leer temperatura y humedad de sensores DHT11 y DHT22 con funciones simples como readTemperature() y readHumidity().
- Biblioteca Servo.h: facilita el control de servomotores mediante funciones como write() para definir el ángulo de rotación.
- Biblioteca AdafruitSensor.h: proporciona un marco estándar para trabajar con diversos sensores, unificando el manejo de datos.

Prácticas propuestas

- **Práctica 1**: leer temperatura y humedad con el sensor DHT11.
  - Conecta el DHT11 a la ESP32.
  - Usa la biblioteca **DHT.h** para leer y mostrar los valores en el monitor serie.
  - Utiliza el siguiente código base:

```
#include <DHT.h> // Incluye la biblioteca DHT para trabajar con
sensores de temperatura y humedad
#define DHTPIN 15   // Define el pin al que está conectado el sensor
DHT
#define DHTTYPE DHT11 // Define el tipo de sensor (DHT11 en este caso)
// Crea un objeto DHT asociado al pin y al tipo de sensor
especificados
DHT dht(DHTPIN, DHTTYPE);
void setup() {
  Serial.begin(115200); // Inicializa la comunicación serial con una
velocidad de 115200 bps
  dht.begin();          // Inicializa el sensor DHT
}
void loop() {
  // Lee la temperatura desde el sensor
  float temp = dht.readTemperature();
  // Lee la humedad desde el sensor
  float hum = dht.readHumidity();
  // Imprime la temperatura en el monitor serial
  Serial.print("Temperatura: ");
  Serial.print(temp);
  Serial.print(" °C, Humedad: ");
  // Imprime la humedad en el monitor serial
  Serial.print(hum);
  Serial.println(" %");
  // Espera 2 segundos antes de realizar otra lectura
  delay(2000);
}
```

- **Práctica 2:** controlar un servomotor con un potenciómetro
  - Conecta el potenciómetro al pin ADC y el servomotor a un pin GPIO.
  - Usa la biblioteca Servo.h para mover el servomotor según la posición del potenciómetro.
  - Utiliza el siguiente código base:

```
#include <Servo.h> // Incluye la biblioteca Servo.h para controlar el
servomotor
Servo servoMotor; // Crea un objeto Servo para controlar el servomotor
#define POT_PIN 34 // Define el pin ADC donde está conectado el
potenciómetro
```

```
#define SERVO_PIN 26 // Define el pin GPIO donde está conectado el
servomotor
void setup() {
  servoMotor.attach(SERVO_PIN); // Conecta el objeto Servo al pin
definido para el servomotor
  Serial.begin(115200); // Inicializa la comunicación serial para
monitoreo
}
void loop() {
  // Lee el valor analógico del potenciómetro (rango de 0 a 4095 en
ESP32)
  int potValue = analogRead(POT_PIN);
  // Mapea el valor del potenciómetro (0-4095) al rango de movimiento
del servomotor (0-180 grados)
  int angle = map(potValue, 0, 4095, 0, 180);
  // Mueve el servomotor al ángulo calculado
  servoMotor.write(angle);
  // Imprime los valores en el monitor serial para verificar
  Serial.print("Valor del potenciómetro: ");
  Serial.print(potValue);
  Serial.print(" | Ángulo del servomotor: ");
  Serial.println(angle);
  // Espera 100 ms antes de la siguiente lectura para evitar
sobrecargar el monitor serial
  delay(100);
}
```

- **Práctica 3**: encender y apagar un LED con un sensor de luz (LDR)
  - Conecta un LDR con una resistencia pull-down al pin ADC.
  - Programa la ESP32 para encender un LED si el valor de luz cae por debajo de un umbral.
  - Utiliza el siguiente código base:

```
#define LDR_PIN 34   // Define el pin ADC donde está conectado el LDR
#define LED_PIN 25   // Define el pin GPIO donde está conectado el LED
#define THRESHOLD 2000 // Define el umbral de luz para encender el LED
(valor ajustable)
void setup() {
  pinMode(LED_PIN, OUTPUT); // Configura el pin del LED como salida
  Serial.begin(115200);     // Inicializa la comunicación serial para
monitoreo
}
void loop() {
  // Lee el valor analógico del LDR (rango de 0 a 4095 en ESP32)
  int ldrValue = analogRead(LDR_PIN);
  // Imprime el valor del LDR en el monitor serial
  Serial.print("Valor del LDR: ");
  Serial.println(ldrValue);
  // Comprueba si el valor del LDR está por debajo del umbral
  if (ldrValue < THRESHOLD) {
    digitalWrite(LED_PIN, HIGH); // Enciende el LED si hay poca luz
    Serial.println("LED ENCENDIDO (Luz baja)");
  } else {
    digitalWrite(LED_PIN, LOW);  // Apaga el LED si hay suficiente luz
    Serial.println("LED APAGADO (Luz suficiente)");
  }
  // Espera 500 ms antes de realizar otra lectura
```

```
  delay(500);
}
```

Resumen del capítulo

Este capítulo introdujo los conceptos de sensores y actuadores, componentes esenciales en cualquier proyecto de hardware, explorando varios tipos de sensores comunes, como el DHT11 (temperatura y humedad), el LDR (luz), el HC-SR04 (ultrasónico), y el PIR (movimiento), así como actuadores como LEDs, motores, relés y pantallas.

Los estudiantes aprendieron a conectar estos componentes a la ESP32 y a leer y escribir datos utilizando pines digitales y analógicos, siendo que este capítulo proporcionó las herramientas necesarias para interactuar con el entorno físico mediante la ESP32.

Autoevaluación

Preguntas teóricas:

- ¿Qué es un sensor y cuál es su función en un sistema electrónico?
- Describe cómo funciona un sensor de temperatura DHT11 y cómo se conecta a la ESP32.
- Explica la diferencia entre un actuador digital y uno analógico.

Ejercicios prácticos:

- Conecta un sensor LDR a la ESP32 y escribe un programa que lea el valor de luz y lo muestre en el monitor serie.
- Controla un servomotor con pulsadores conectados a la ESP32.

Desafío:

- Diseña un sistema que utilice un sensor de movimiento PIR para encender un LED cuando se detecte movimiento.

CAPÍTULO V

Comunicación en Serie y Monitorización de Datos

ALUMNI EDITORA

Introducción a la Comunicación en Serie y Monitorización de Datos

El objetivo de este capítulo es enseñar a los estudiantes cómo establecer una comunicación en serie entre la ESP32 y una computadora, utilizando el Arduino IDE para monitorizar datos, depurar programas y enviar comandos en tiempo real, permitiendo comprender la importancia de este tipo de comunicación en proyectos de IoT, donde el monitoreo y control remoto son esenciales.

¿Qué es la comunicación en serie?

La comunicación en serie es un método de transmisión de datos en el que la información se envía bit a bit a través de un canal, como un cable o un puerto, gracias a su sencillez es ampliamente utilizada en dispositivos electrónicos para intercambiar datos de manera eficiente.

El protocolo UART[20] (Universal Asynchronous Receiver-Transmitter) es una forma común de implementar esta comunicación, donde no se requiere un reloj compartido entre el emisor y el receptor, ya que la comunicación se sincroniza mediante configuraciones de velocidad de transmisión, conocida como baud rate, que es la cantidad de bits transmitidos por segundo, una configuración incorrecta del baud rate puede llevar a datos corruptos, por lo que es crucial establecer la misma velocidad en ambos extremos.

Uso del monitor serie en Arduino IDE

El monitor serie en el Arduino IDE es una herramienta que permite interactuar con la ESP32 en tiempo real y para usarlo, es necesario seleccionar el puerto COM al que está conectada la placa y configurar el baud rate adecuado siendo de 115200 para la ESP32.

Se pueden enviar datos desde la ESP32 utilizando funciones como Serial.print() y recibir información desde la computadora mediante el comando Serial.read(), lo que permite depurar programas, verificar el comportamiento de sensores y actuadores, y controlar dispositivos de manera interactiva.

Una de las ventajas del monitor serie es que soporta la representación formateada de datos, como cadenas de texto y valores numéricos, lo que facilita la visualización y análisis de la información enviada por la ESP32.

Enviar datos desde la ESP32

La ESP32 puede enviar datos al monitor serie usando las funciones Serial.print() y Serial.println(), estas funciones permiten mostrar información en tiempo real, como los valores leídos por sensores o mensajes de estado de un programa.

Por ejemplo: se usa Serial.print() y Serial.println() para enviar datos al monitor serie incrementando un contador que se muestra en el monitor cada segundo:

```
void setup() {
  // Inicializar la comunicación en serie con un baud rate de 115200
  Serial.begin(115200);
  // Mensaje inicial para confirmar que la ESP32 está lista
  Serial.println("Iniciando comunicación en serie...");
}
void loop() {
  // Variable para almacenar el contador
  static int contador = 0;
  // Enviar el valor del contador al monitor serie
  Serial.print("Contador: ");
  Serial.println(contador);
  // Incrementar el contador en 1
  contador++;
  // Pausa de 1 segundo antes de enviar el siguiente valor
  delay(1000);
}
```

Recepción de datos en la ESP32

La ESP32 puede recibir datos enviados desde el monitor serie mediante la función Serial.read() y sus variantes, esto permite controlar dispositivos conectados a la ESP32 mediante comandos enviados desde la computadora.

Por ejemplo: se utiliza Serial.available() para comprobar si hay datos disponibles para leer desde el monitor serie y Serial.readStringUntil('\n') para capturar comandos enviados por el usuario, lo que permite controlar un LED en función del comando recibido, proporcionando retroalimentación en el monitor serie:

```
// Definir el pin del LED (puede ser cualquier pin GPIO)
```

```
#define LED_PIN 2
void setup() {
  // Configurar el pin del LED como salida
  pinMode(LED_PIN, OUTPUT);
  // Inicializar la comunicación en serie con un baud rate de 115200
  Serial.begin(115200);
  // Mensaje inicial para indicar que el programa está listo
  Serial.println("Escribe 'on' para encender el LED o 'off' para
apagarlo.");
}
void loop() {
  // Comprobar si hay datos disponibles para leer desde el monitor
serie
  if (Serial.available() > 0) {
    // Leer la cadena enviada desde el monitor serie
    String comando = Serial.readStringUntil('\n');
    // Eliminar espacios en blanco y convertir el comando a minúsculas
    comando.trim();
    comando.toLowerCase();
    // Verificar si el comando es "on"
    if (comando == "on") {
      // Encender el LED
      digitalWrite(LED_PIN, HIGH);
      Serial.println("LED encendido.");
    }
    // Verificar si el comando es "off"
    else if (comando == "off") {
      // Apagar el LED
      digitalWrite(LED_PIN, LOW);
      Serial.println("LED apagado.");
    }
    // Mensaje de error si el comando no es válido
    else {
      Serial.println("Comando no reconocido. Usa 'on' o 'off'.");
    }
  }
}
```

Errores comunes en la comunicación en serie

La comunicación en serie puede presentar problemas si no se configuran correctamente los parámetros, por lo que algunos errores comunes incluyen:

- Baud rate incorrecto: si la velocidad de transmisión configurada en el monitor serie no coincide con la de la ESP32, los datos no se interpretarán correctamente.
- Conflictos de puerto: ocurren cuando otro programa utiliza el mismo puerto COM, impidiendo que el Arduino IDE se comunique con la ESP32, incluso si se utilizan puertos virtuales.
- Errores de conexión: conexiones físicas inadecuadas o cables defectuosos pueden causar pérdida de datos.

Para evitar estos problemas, es importante revisar las configuraciones y utilizar herramientas de depuración, como mensajes de estado en el monitor serie.

Prácticas propuestas

- **Práctica 1**: crear un programa para leer datos de temperatura de un sensor y enviarlos al monitor serie.
- **Práctica 2:** diseñar un programa que controle la frecuencia de parpadeo de un LED según valores ingresados desde el monitor serie.
- **Práctica 3**: enviar un comando desde el monitor serie para activar o desactivar un relé conectado a la ESP32.

Resumen del capítulo

En este capítulo, se abordó la comunicación en serie entre la ESP32 y una computadora, utilizando el monitor serie del Arduino IDE.

Los estudiantes aprendieron a enviar y recibir datos, lo que es fundamental para depurar programas y monitorizar proyectos en tiempo real, cubriendo funciones como Serial.print(), Serial.read(), y Serial.available(), y se explicaron errores comunes en la comunicación en serie, lo que permitió a los estudiantes entender cómo establecer una comunicación efectiva entre la ESP32 y otros dispositivos.

Autoevaluación

Preguntas teóricas:

- ¿Qué es la comunicación en serie y por qué es importante en proyectos con microcontroladores?
- Explica cómo se utiliza el monitor serie en el Arduino IDE.
- ¿Qué función se utiliza para enviar datos desde la ESP32 al monitor serie?

Ejercicios prácticos:

- Escribe un programa que envíe un mensaje desde la ESP32 al monitor serie cada vez que se presione un botón.

- Crea un programa que reciba un comando desde el monitor serie para encender o apagar un LED.

Desafío:

- Implementa un sistema que envíe datos de temperatura y humedad desde la ESP32 al monitor serie en tiempo real.

CAPÍTULO VI

Uso de Pines ADC y PWM en la ESP32

ALUMNI EDITORA

Introducción al Uso de Pines ADC y PWM en la ESP32

Este capítulo tiene como finalidad enseñar a los estudiantes el manejo práctico de los pines ADC (para lectura analógica) y PWM (para generación de señales de modulación por ancho de pulso) de la ESP32, mediante la realización de proyectos básicos que demuestren su aplicación en el control de dispositivos como LEDs, motores y sensores.

¿Qué es un ADC?

Un ADC convierte señales analógicas (tensiones continuas) en valores digitales que la ESP32 puede procesar, por lo que se usan para leer sensores analógicos como potenciómetros, sensores de luz, de temperatura, entre otros.

La ESP32 cuenta con 18 pines ADC divididos en dos controladores (ADC1 y ADC2)[9], cada uno con una resolución por defecto es de 12 bits, lo que da un rango de lectura de 0 a 4095 y un rango de voltaje de 0 a 3.3V.

Lectura de valores analógicos

Función analogRead():

La función analogRead(pin) permite leer valores analógicos de un pin ADC y devuelve un valor proporcional al voltaje de entrada (0-4095 para 0-3.3V).

Ejemplo Práctico:

Este ejemplo lee el valor de un potenciómetro conectado a un pin ADC y lo muestra en el monitor serie, para esto se utiliza el siguiente código:

```
#define POT_PIN 34 // Pin ADC conectado al potenciómetro
void setup() {
  Serial.begin(115200); // Iniciar la comunicación serie y el sensor
DHT
}
void loop() {
  int potValue = analogRead(POT_PIN); // Leer el valor del
potenciómetro
  Serial.print("Valor del potenciómetro: ");
  Serial.println(potValue);          // Mostrar el valor en el monitor
serie
  delay(500);                        // Esperar medio segundo
}
```

¿Qué es el PWM?

La modulación por ancho de pulso (PWM) es una técnica que permite generar una señal digital con un ancho de pulso variable, simulando un voltaje analógico.

Aplicaciones Comunes:

- Controlar el brillo de LEDs.
- Ajustar la velocidad de motores.
- Controlar la posición de servos.

Generación de Señales PWM con la ESP32:

La ESP32 tiene hasta 16 canales PWM por hardware y se puede configurar la frecuencia, resolución y ancho de pulso.

Configuración de Señales PWM

Pasos Básicos para Configurar PWM:

1. Asignar un canal PWM a un pin con ledcAttachPin().
2. Configurar la frecuencia y la resolución del canal con ledcSetup().
3. Ajustar el ancho de pulso con ledcWrite().

Ejemplo Práctico:

Este programa ajusta el brillo de un LED según la posición de un potenciómetro, para lo cual se utiliza el siguiente código:

```
#define POT_PIN 34  // Pin ADC conectado al potenciómetro
#define LED_PIN 25  // Pin GPIO conectado al LED
#define PWM_CHANNEL 0 // Canal PWM
#define PWM_FREQ 5000 // Frecuencia en Hz
#define PWM_RES 8     // Resolución en bits (0-255)
void setup() {
  // Configurar el canal PWM
  ledcSetup(PWM_CHANNEL, PWM_FREQ, PWM_RES);
  ledcAttachPin(LED_PIN, PWM_CHANNEL); // Asignar el canal PWM al pin
del LED
}
void loop() {
  int potValue = analogRead(POT_PIN); // Leer el valor del
potenciómetro (0-4095)

  // Escalar el valor del potenciómetro al rango del PWM (0-255)
  int pwmValue = map(potValue, 0, 4095, 0, 255);
```

```
  ledcWrite(PWM_CHANNEL, pwmValue); // Ajustar el brillo del LED

  delay(100);
}
```

Prácticas propuestas

- **Práctica 1**: leer la salida analógica de un sensor de luz (LDR).
  - Leer el valor analógico de un sensor LDR y mostrarlo en el monitor serie.
  - Utilizar el siguiente código:

```
#define LDR_PIN 34 // Pin ADC conectado al LDR
void setup() {
  Serial.begin(115200);
}
void loop() {
  int ldrValue = analogRead(LDR_PIN);
  Serial.print("Intensidad de luz: ");
  Serial.println(ldrValue);
  delay(500);
}
```

- **Práctica 2:** usar un potenciómetro para controlar el brillo de un LED.
  - Ajustar el brillo de un LED según la posición de un potenciómetro.
  - Utilizar el siguiente código:

```
#define POT_PIN 34  // Pin ADC conectado al potenciómetro
#define LED_PIN 25  // Pin GPIO conectado al LED
void setup() {
  pinMode(LED_PIN, OUTPUT);
}
void loop() {
  int potValue = analogRead(POT_PIN);
  int brightness = map(potValue, 0, 4095, 0, 255);
  analogWrite(LED_PIN, brightness); // Ajustar el brillo del
LED
  delay(100);
}
```

- **Práctica 3**: ajustar la velocidad de un motor de corriente continua usando PWM.
  - Controlar la velocidad de un motor conectado a un módulo MOSFET usando PWM.
  - Utilizar el siguiente código:

```
#define POT_PIN 34  // Pin ADC conectado al potenciómetro
#define MOTOR_PIN 26 // Pin GPIO conectado al módulo MOSFET
#define PWM_CHANNEL 0
#define PWM_FREQ 5000
```

```
#define PWM_RES 8
void setup() {
  ledcSetup(PWM_CHANNEL, PWM_FREQ, PWM_RES);
  ledcAttachPin(MOTOR_PIN, PWM_CHANNEL);
}
void loop() {
  int potValue = analogRead(POT_PIN);
  int pwmValue = map(potValue, 0, 4095, 0, 255);
  ledcWrite(PWM_CHANNEL, pwmValue); // Ajustar la velocidad
del motor
  delay(100);
}
```

Resumen del capítulo

Este capítulo se centró en el uso de los pines ADC (Convertidor Analógico-Digital) y PWM (Modulación por Ancho de Pulso) de la ESP32, para que los estudiantes aprendan a leer valores analógicos de sensores como potenciómetros y LDRs, y a generar señales PWM para controlar dispositivos como LEDs y motores.

Se explicaron conceptos como la resolución de los ADC, la configuración de canales PWM, y la generación de señales con diferentes ciclos de trabajo, proporcionando las herramientas necesarias para trabajar con señales analógicas y controlar dispositivos de manera precisa.

Autoevaluación

Preguntas teóricas:

- ¿Qué es un ADC y cómo se utiliza en la ESP32?
- Explica cómo funciona la modulación por ancho de pulso (PWM) y cuáles son sus aplicaciones.
- ¿Cuál es la resolución por defecto de los pines ADC en la ESP32?

Ejercicios prácticos:

- Conecta un potenciómetro a un pin ADC de la ESP32 y escribe un programa que lea su valor y lo muestre en el monitor serie.
- Controla el brillo de un LED utilizando un pin PWM y un potenciómetro.

CAPÍTULO VII

Proyectos Básicos con Sensores y Actuadores

ALUMNI EDITORA

Introducción a los Proyectos Básicos Integrados con la ESP32

Este capítulo está diseñado para consolidar todos los conocimientos adquiridos a lo largo del libro mediante la realización de proyectos integradores, por lo que cada proyecto combina el uso de sensores, actuadores y funciones avanzadas de la ESP32, permitiendo a los estudiantes experimentar con aplicaciones prácticas y reales.

Proyecto 1: Control de luz automatizado

Este proyecto consiste en encender un LED automáticamente cuando un sensor LDR detecte niveles bajos de luz en el ambiente, lo que es un ejemplo básico de automatización que tiene aplicaciones prácticas como sistemas de iluminación nocturna.

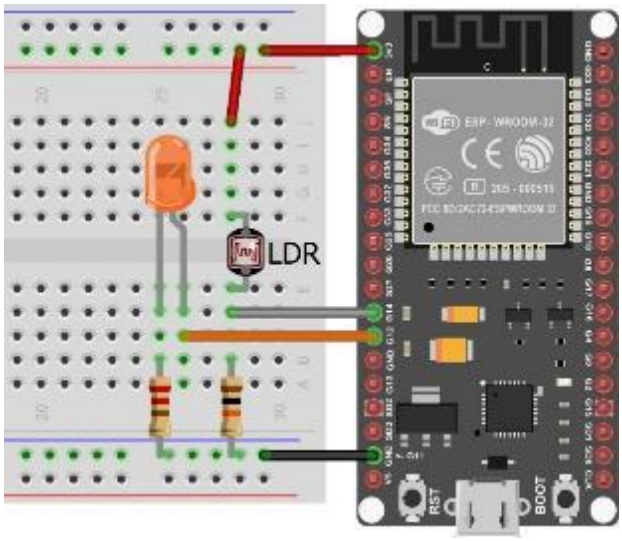
Componentes necesarios:

- Sensor LDR.
- Resistencia de 10 kΩ (como pull-down para el LDR).
- ESP32.
- LED.
- Resistencia de 220 Ω (para el LED).
- Cables de conexión
- Protoboard

Conexión del circuito:

El LDR se conecta en un divisor de voltaje con una resistencia pull-down, y la salida de este divisor se conecta a un pin analógico (ADC-GPIO 14) de la ESP32 y el LED se conecta a un pin digital (GPIO 12) mediante una resistencia para limitar la corriente, en la figura 3 se muestra la conexión de este proyecto.

Figura 3
Conexión del circuito del Proyecto 1.



Código de programación:

El programa lee el valor del LDR a través del pin ADC de la ESP32, compara este valor con un umbral predeterminado y enciende o apaga el LED según corresponda, el umbral puede ajustarse según las condiciones de luz del entorno, ya sea para interiores o exteriores, se utiliza el siguiente código para el correcto funcionamiento:

```
// Definir los pines para el LDR y el LED
#define LDR_PIN 14   // Pin ADC donde se conecta el LDR
#define LED_PIN 12   // Pin GPIO donde se conecta el LED
// Variable para almacenar el valor leído del LDR
int ldrValor = 0;
// Umbral para encender el LED (ajustable según las condiciones de
luz)
const int LDR_UMBRAL = 1000;
void setup() {
  // Configurar el pin del LED como salida
  pinMode(LED_PIN, OUTPUT);
  // Iniciar la comunicación serie para monitoreo
  Serial.begin(115200);
}
void loop() {
  // Leer el valor del LDR desde el pin ADC
  ldrValor = analogRead(LDR_PIN);
  // Imprimir el valor leído en el monitor serie
  Serial.print("Valor del LDR: ");
  Serial.println(ldrValor);
  // Comparar el valor del LDR con el umbral
  if (ldrValor < LDR_UMBRAL) {
    // Si el nivel de luz es bajo, encender el LED
    digitalWrite(LED_PIN, HIGH);
    Serial.println("LED encendido.");
  } else {
```

```
    // Si el nivel de luz es suficiente, apagar el LED
    digitalWrite(LED_PIN, LOW);
    Serial.println("LED apagado.");
  }
  // Esperar 2 segundos antes de la siguiente lectura
  delay(2000);
}
```

Proyecto 2: Control de un ventilador basado en temperatura

Este proyecto usa un sensor de temperatura para encender un ventilador automáticamente cuando la temperatura ambiente supere un umbral definido, lo que hace a este tipo de sistema utilizarse comúnmente en refrigeración automatizada y control de climatización.

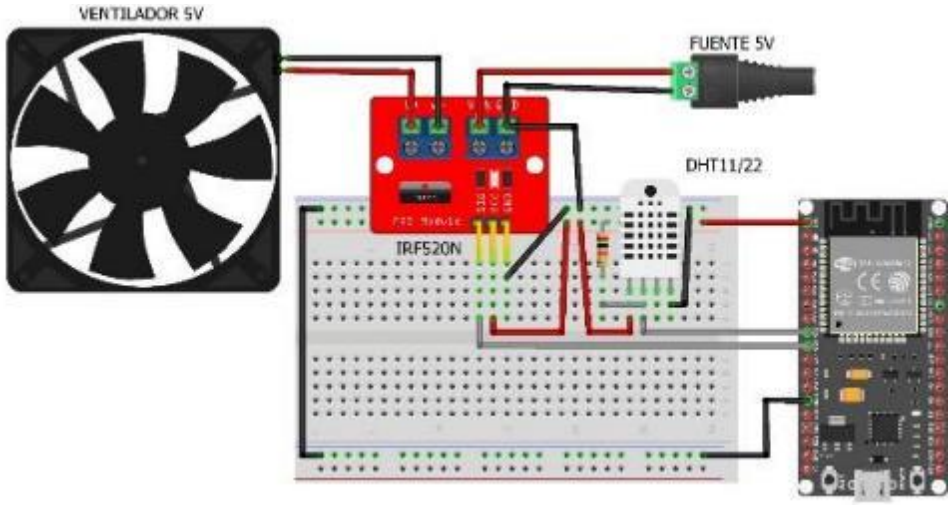
Componentes necesarios:

- Sensor DHT11 o DHT22 (para medir la temperatura).
- Resistencia de 10 kΩ (como pull-up para el DHT11/22).
- Módulo IRF520N comercial (MOSFET para el control del ventilador).
- Ventilador de 5V o 12V.
- ESP32.
- Fuente de alimentación externa (según el ventilador).
- Cables de conexión
- Protoboard

Conexión del circuito:

Las conexiones se detallan en la figura 4:

Figura 4
Conexión del circuito del Proyecto 2.



Código de programación:

El sensor DHT11 mide la temperatura y envía los datos a la ESP32, cuando la temperatura excede el umbral, la ESP32 activa el módulo MOSFET que enciende el ventilador.

El programa incluye mensajes en el monitor serie para mostrar la temperatura en tiempo real y el estado del ventilador, lo que permite a los estudiantes comprender cómo integrar sensores, actuadores y comunicación en serie, se utiliza el siguiente código para el correcto funcionamiento:

```
// Incluye la biblioteca DHT para trabajar con sensores de temperatura
y humedad
#include <DHT.h>
// Configuración del sensor DHT11
#define DHTPIN 25         // Pin GPIO conectado al DHT11
#define DHTTYPE DHT11     // Tipo de sensor DHT11
// Crea un objeto DHT asociado al pin y al tipo de sensor
especificados
DHT dht(DHTPIN, DHTTYPE);
// Configuración del módulo MOSFET
#define MOSFET_PIN 26     // Pin GPIO conectado al SIG del módulo
MOSFET
// Umbral de temperatura (ajustable)
const float TEMP_UMBRAL = 30.0; // Temperatura en grados Celsius
void setup() {
  // Configurar el pin del MOSFET como salida
  pinMode(MOSFET_PIN, OUTPUT);
  // Asegurarse de que el ventilador esté apagado al inicio
  digitalWrite(MOSFET_PIN, LOW);
  // Iniciar la comunicación serie y el sensor DHT
  Serial.begin(115200);
  dht.begin();
}
void loop() {
```

```
  // Leer la temperatura del sensor DHT
  float temperatura = dht.readTemperature();
  // Verificar si la lectura es válida
  if (isnan(temperatura)) {
    Serial.println("Error al leer el sensor DHT.");
    return;
  }
  // Imprimir la temperatura en el monitor serie
  Serial.print("Temperatura: ");
  Serial.print(temperatura);
  Serial.println(" °C");
  // Comparar la temperatura con el umbral
  if (temperature > TEMP_UMBRAL) {
    // Encender el ventilador si la temperatura supera el umbral
    digitalWrite(MOSFET_PIN, HIGH);
    Serial.println("Ventilador encendido.");
  } else {
    // Apagar el ventilador si la temperatura es baja
    digitalWrite(MOSFET_PIN, LOW);
    Serial.println("Ventilador apagado.");
  }
  // Esperar 2 segundos antes de la siguiente lectura
  delay(2000);
}
```

Proyecto 3: Servo controlado por potenciómetro

Este proyecto controla el movimiento de un servomotor mediante la lectura de un potenciómetro, por lo que el ángulo del servomotor se ajusta en tiempo real dependiendo de la posición del potenciómetro, lo que ilustra el control proporcional.

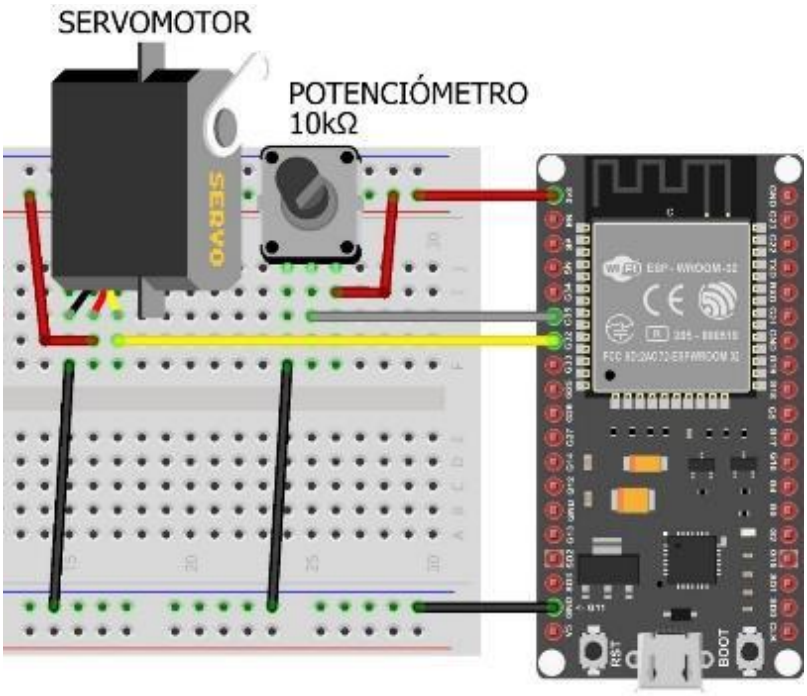
Componentes necesarios:

- Servomotor (como SG90 o MG996R).
- Potenciómetro.
- ESP32.
- Fuente de alimentación externa (si el servomotor requiere más corriente).
- Cables de conexión
- Protoboard

Conexión del circuito:

Las conexiones se detallan en la figura 5:

Figura 5
Conexión del circuito del Proyecto 3.



Código de programación:

El potenciómetro actúa como un divisor de voltaje, generando una señal analógica que la ESP32 lee a través de un pin ADC, así la posición del potenciómetro se traduce en un valor digital que se convierte en el ángulo de rotación del servomotor (generalmente de 0 a 180 grados).

El programa utiliza la biblioteca Servo.h para controlar el servomotor, lo que facilita el envío de comandos de posición y con esta actividad los estudiantes comprenden cómo transformar señales analógicas en acciones mecánicas precisas y controladas, se utiliza el siguiente código para el correcto funcionamiento:

```
// Incluye la biblioteca Servo.h para controlar el servomotor
#include <Servo.h>
Servo myServo;        // Crear un objeto servo
#define POT_PIN 35    // Pin ADC conectado al potenciómetro
#define SERVO_PIN 32 // Pin GPIO conectado al servomotor
void setup() {
  // Adjuntar el servomotor al pin correspondiente
  myServo.attach(SERVO_PIN);
  // Iniciar la comunicación serie
  Serial.begin(115200);
}
void loop() {
  // Leer el valor del potenciómetro (0-4095 para la ESP32)
  int potValor = analogRead(POT_PIN);
```

```
  // Mapear el valor del potenciómetro al rango del servomotor (0-180
grados)
  int servoAngulo = map(potValor, 0, 4095, 0, 180);
  // Mover el servomotor al ángulo correspondiente
  myServo.write(servoAngulo);
  // Imprimir los valores en el monitor serie para depuración
  Serial.print("Valor del potenciómetro: ");
  Serial.print(potValor);
  Serial.print(", Ángulo del servomotor: ");
  Serial.println(servoAngulo);
  // Esperar 100 ms antes de actualizar nuevamente
  delay(100);
}
```

Prácticas propuestas

- **Práctica 1**: diseñar un sistema que detecte movimiento y encienda un LED o emita una señal sonora en base a las lecturas de un sensor PIR.
- **Práctica 2:** implementar un programa para encender y apagar luces mediante un botón pulsador, simulando un interruptor básico.
- **Práctica 3**: diseñar un contador digital que incremente un valor cada vez que se presiona un botón y lo muestre en el monitor serie.

Resumen del capítulo

En este capítulo, los estudiantes aplicaron los conocimientos adquiridos en capítulos anteriores mediante la realización de proyectos prácticos sencillos como un control de luz automatizado con un LDR, un ventilador controlado por temperatura, y un servomotor controlado por un potenciómetro.

Estos proyectos permitieron a los estudiantes integrar sensores y actuadores con la ESP32, reforzando su comprensión de cómo interactuar con el entorno físico y cómo diseñar soluciones automatizadas.

Autoevaluación

Preguntas teóricas:

- Describe cómo funciona un sistema de control de luz automatizado con un sensor LDR.
- ¿Qué componentes se necesitan para controlar un ventilador basado en la temperatura ambiente?
- Explica cómo se controla un servomotor con un potenciómetro.

CAPÍTULO VIII

Proyectos Básicos Integrados con la ESP32

ALUMNI EDITORA

Introducción a los Proyectos Básicos Integrados con la ESP32

Este capítulo está diseñado para consolidar todos los conocimientos adquiridos a lo largo del libro mediante la realización de proyectos integradores, por lo que cada proyecto combina el uso de sensores, actuadores y funciones avanzadas de la ESP32, permitiendo a los estudiantes experimentar con aplicaciones prácticas y reales.

Proyecto 1: Estación Meteorológica Básica

El primer proyecto consiste en diseñar una estación meteorológica básica que permita medir y monitorear en tiempo real parámetros como la temperatura, la humedad y la luz ambiental, dicha información obtenida será mostrada directamente en el monitor serie de la ESP32, lo que proporciona un enfoque inicial para trabajar con sensores combinados y analizar datos.

Para su construcción se requiere un sensor DHT11 para medir la temperatura y la humedad, un fotorresistor (LDR) para captar la luz ambiental, resistencias para el circuito y la ESP32 como unidad central de procesamiento, la conexión se puede observar en la figura 6.

Componentes necesarios:

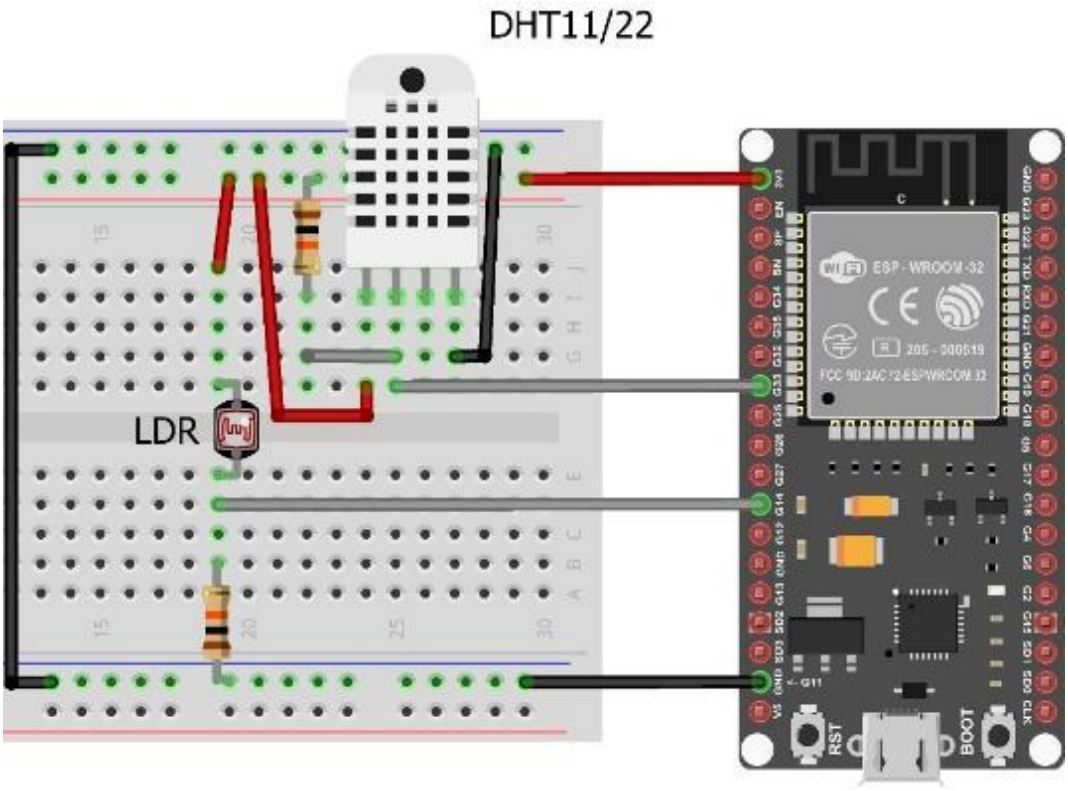
- Sensor DHT11 (temperatura y humedad).
- LDR (fotorresistor) para medir luz ambiental.
- Resistencia de 10kΩ para el LDR.
- Resistencia de 10kΩ para el DHT11.
- ESP32.
- Cables de conexión
- Protoboard

Conexión del circuito:

El LDR se conecta en un divisor de voltaje con una resistencia pull-down, y la salida de este divisor se conecta a un pin analógico (ADC-GPIO 14) de la ESP32 y el DHT11/22 se conecta a un pin digital (GPIO 33) mediante una

resistencia para limitar la corriente, en la figura 3 se muestra la conexión de este proyecto.

Figura 6
Conexión del circuito del Proyecto Integrado 1.



Código de programación:

Los datos obtenidos de los sensores se leen y se imprimen en el monitor serie de la ESP32, utilizando el siguiente código para el correcto funcionamiento:

```
#include <DHT.h>  // Biblioteca para manejar el sensor DHT
#define DHTPIN 4        // Pin donde está conectado el DHT11
#define DHTTYPE DHT11   // Tipo de sensor DHT usado (DHT11)
#define LDRPIN 34       // Pin ADC donde está conectado el LDR
DHT dht(DHTPIN, DHTTYPE);  // Inicializamos el sensor DHT
void setup() {
  Serial.begin(115200);  // Inicializamos la comunicación serie
  dht.begin();           // Activamos el sensor DHT
}
void loop() {
  float temperatura = dht.readTemperature();  // Leemos la temperatura
  float humedad = dht.readHumidity();         // Leemos la humedad
  // Verificamos si las lecturas son válidas
  if (isnan(temperatura) || isnan(humedad)) {
    Serial.println("Error al leer el sensor DHT");
    return;
  }
```

```
  int valorLuz = analogRead(LDRPIN);  // Leemos el valor analógico del
LDR
  // Mostramos los datos en el monitor serie
  Serial.print("Temperatura: ");
  Serial.print(temperatura);
  Serial.println(" °C");
  Serial.print("Humedad: ");
  Serial.print(humedad);
  Serial.println(" %");
  Serial.print("Nivel de luz (LDR): ");
  Serial.println(valorLuz);
  delay(2000);  // Pausa de 2 segundos antes de la siguiente lectura
}
```

Proyecto 2: Sistema de control de luces automatizado

En este proyecto se implementará un sistema que permite encender o apagar una luz de manera automática al detectar movimiento, utilizando un sensor PIR y un módulo de relé, lo que hace a este sistema ideal para automatizar espacios interiores como habitaciones, pasillos o baños, optimizando el consumo de energía.

Los componentes necesarios incluyen un sensor PIR que detecta la presencia de personas, un módulo de relé para controlar el encendido o apagado de la luz, una lámpara o un LED que actúe como actuador, y la ESP32 para coordinar todo el proceso.

Componentes necesarios:

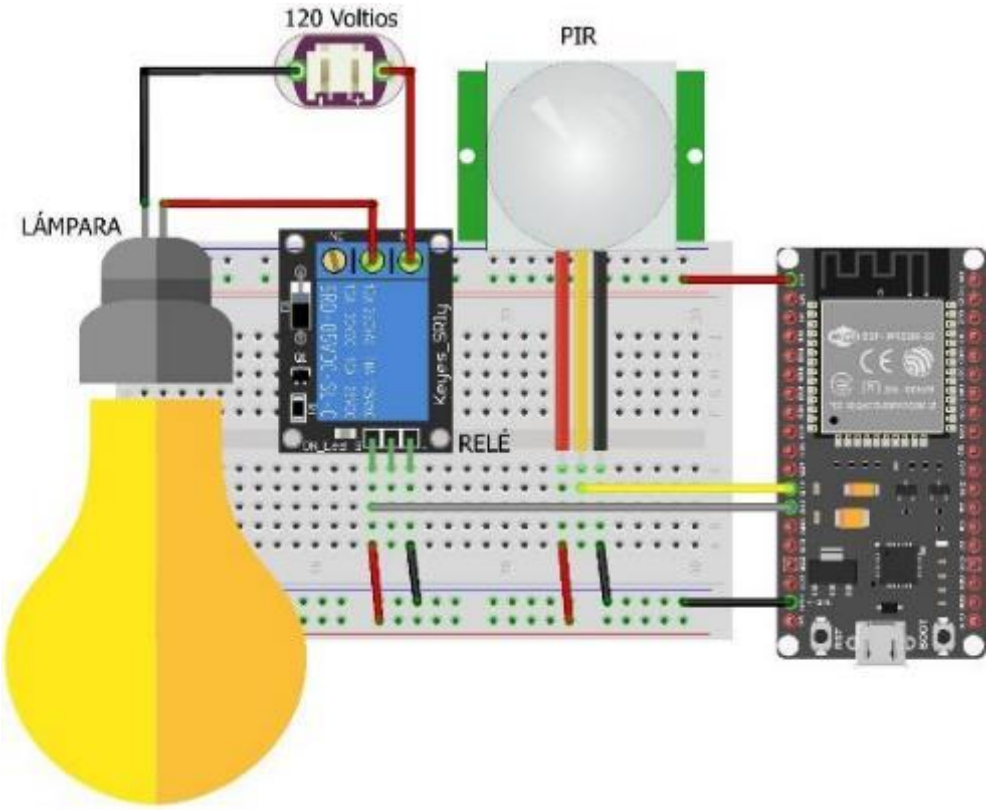
- Sensor PIR.
- Módulo de relé.
- Lámpara o LED como actuador.
- ESP32.
- Fuente de alimentación externa (si se utiliza lámpara).
- Cables de conexión
- Protoboard

Conexión del circuito:

El circuito es simple pero efectivo: el sensor PIR se conecta a un pin de la ESP32 (GPIO 14) para enviar señales de detección, mientras que el módulo de

relé es activado desde otro pin (GPIO12), permitiendo controlar la lámpara conectada, las conexiones se detallan en la figura 7:

Figura 7
Conexión del circuito del Proyecto Integrado 2.



Código de programación:

El código desarrollado para este proyecto incluye un esquema lógico que verifica continuamente la señal del sensor PIR y enciende la luz al detectar movimiento, apagándola automáticamente cuando no se detecta actividad después de un intervalo de tiempo, se utiliza el siguiente código para el correcto funcionamiento:

```
#define PIRPIN 14  // Pin donde está conectado el sensor PIR
#define RELEPIN 12  // Pin donde está conectado el módulo relé
void setup() {
  pinMode(PIRPIN, INPUT);   // Configuramos el pin del PIR como
entrada
  pinMode(RELEPIN, OUTPUT); // Configuramos el pin del relé como
salida
  digitalWrite(RELEPIN, LOW);  // Inicialmente apagamos la luz
}
void loop() {
  int estadoPIR = digitalRead(PIRPIN);  // Leemos el estado del sensor
PIR
```

```
  if (estadoPIR == HIGH) {
    // Si se detecta movimiento, encendemos la luz
    digitalWrite(RELEPIN, HIGH);
    Serial.println("Movimiento detectado: luz encendida");
  } else {
    // Si no hay movimiento, apagamos la luz
    digitalWrite(RELEPIN, LOW);
    Serial.println("Sin movimiento: luz apagada");
  }
  delay(500);  // Pausa para evitar lecturas erráticas
}
```

Proyecto 3: Control de velocidad de un ventilador

Este proyecto tiene como objetivo controlar la velocidad de un ventilador de manera dinámica, combinando un ajuste automático basado en la temperatura ambiente con un control manual mediante un potenciómetro.

La implementación requiere un sensor de temperatura, un potenciómetro, un módulo MOSFET para manejar la corriente del ventilador, la ESP32 como controlador principal y un ventilador de baja potencia como actuador.

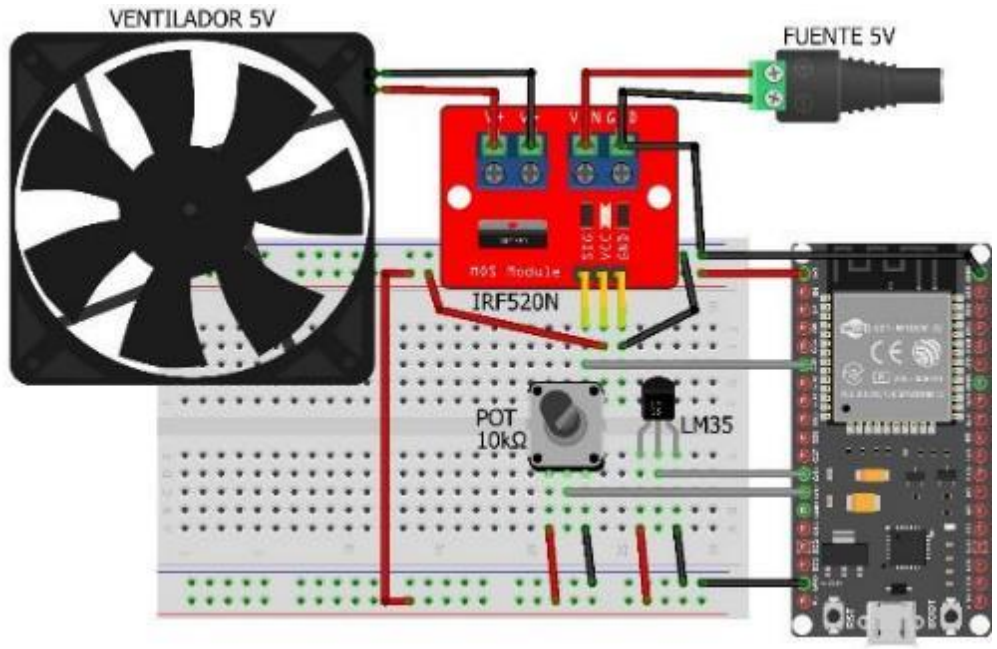
Componentes necesarios:

- Sensor de temperatura analógico LM35.
- Potenciómetro.
- Módulo MOSFET IRF520N.
- Ventilador de 5V.
- ESP32.
- Fuente de alimentación externa (para el ventilador).
- Cables de conexión
- Protoboard

Conexión del circuito:

El diseño del circuito incluye la conexión del sensor de temperatura y el potenciómetro a pines ADC, GPIO14 y GPIO12 respectivamente, de la ESP32 para realizar lecturas analógicas, mientras que el módulo MOSFET se controla desde un pin PWM (GPIO35), lo que permite variar la velocidad del ventilador de forma precisa, las conexiones se detallan en la figura 8:

Figura 8
Conexión del circuito del Proyecto Integrado 3.



Código de programación:

En el código proporcionado, se realiza la lectura de las señales analógicas del sensor de temperatura y del potenciómetro, mapeándolas a un rango de valores PWM, así el ventilador ajusta su velocidad en función del mayor valor entre el control automático y el manual, asegurando una experiencia interactiva y funcional, se utiliza el siguiente código para el correcto funcionamiento:

```
// Definición de los pines y configuraciones para el sistema
#define TEMP_PIN 14      // Pin ADC conectado al sensor de temperatura
#define POT_PIN 12       // Pin ADC conectado al potenciómetro
#define FAN_PIN 35       // Pin GPIO utilizado para el control del
ventilador mediante PWM
#define PWM_CHANNEL 0    // Canal PWM asignado al ventilador
#define PWM_FREQ 5000    // Frecuencia del PWM configurada en 5000 Hz
#define PWM_RES 8        // Resolución del PWM configurada en 8 bits
(valores de 0 a 255)
// Configuración inicial del sistema
void setup() {
  // Configuración del canal PWM con la frecuencia y resolución
establecidas
  ledcSetup(PWM_CHANNEL, PWM_FREQ, PWM_RES);
  // Asignación del pin del ventilador al canal PWM definido
  ledcAttachPin(FAN_PIN, PWM_CHANNEL);
}
// Bucle principal del programa
void loop() {
  // Leer el valor analógico del sensor de temperatura
  int tempValue = analogRead(TEMP_PIN);
```

```
  // Leer el valor analógico del potenciómetro
  int potValue = analogRead(POT_PIN);
  // Mapear el valor del sensor de temperatura al rango del PWM (0 a
255)
  int tempSpeed = map(tempValue, 0, 4095, 0, 255);
  // Mapear el valor del potenciómetro al rango del PWM (0 a 255)
  int manualSpeed = map(potValue, 0, 4095, 0, 255);
  // Seleccionar la mayor velocidad entre el control automático
(temperatura) y el manual (potenciómetro)
  int finalSpeed = max(tempSpeed, manualSpeed);
  // Aplicar el valor final del PWM al ventilador
  ledcWrite(PWM_CHANNEL, finalSpeed);
  // Esperar 100 ms antes de repetir el bucle
  delay(100);
}
```

Proyecto 4: Semáforo controlado por botón

En este proyecto se implementa un semáforo básico que cambia de estado entre las luces verde, amarilla y roja al presionar un botón, simulando el funcionamiento de un semáforo real y permitiendo al estudiante comprender conceptos básicos de lógica secuencial y control de LEDs.

Los componentes necesarios incluyen tres LEDs de colores rojo, amarillo y verde, un pulsador, resistencias para los LEDs, y la ESP32 como unidad de control.

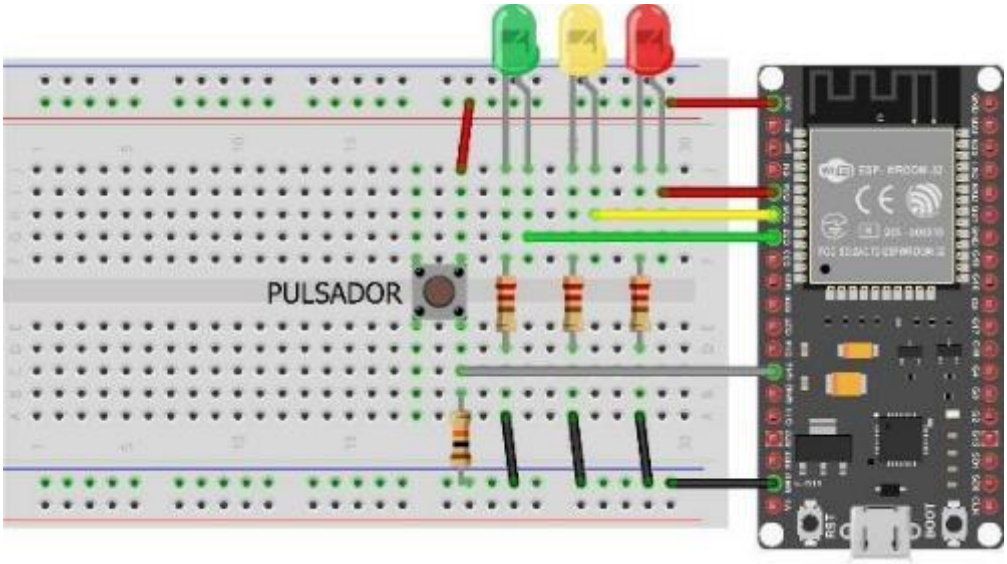
Componentes necesarios:

- LEDs (rojo, amarillo, verde).
- Resistencias de 220Ω por LED.
- Pulsador.
- Resistencia de 10kΩ
- ESP32.
- Cables de conexión
- Protoboard

Conexión del circuito:

Los LEDs se conectan a pines GPIO de la ESP32 con sus respectivas resistencias para limitar la corriente GPIO32 para el LED verde, GPIO35 para el LED amarillo y GPIO34 para el LED rojo, mientras que el pulsador se conecta al GPIO12 con una resistencia pull-down de 10kΩ para evitar falsos positivos, las conexiones se detallan en la figura 9:

Figura 9
Conexión del circuito del Proyecto Integrado 4.



Código de programación:

El código guía al estudiante a programar la lógica de cambio de estado del semáforo, asegurando que cada luz se encienda de manera secuencial al detectar una pulsación del botón, lo que hace a este proyecto especialmente útil para reforzar la comprensión de estructuras condicionales y bucles en la programación de la ESP32, se utiliza el siguiente código para el correcto funcionamiento:

```
#define LED_ROJO 34     // Pin del LED rojo
#define LED_AMARILLO 35 // Pin del LED amarillo
#define LED_VERDE 32    // Pin del LED verde
#define PIN_BOTON 12    // Pin del pulsador
int estadoSemaforo = 0; // Estado actual del semáforo
void setup() {
  // Configuramos los LEDs como salida
  pinMode(LED_ROJO, OUTPUT);
  pinMode(LED_AMARILLO, OUTPUT);
  pinMode(LED_VERDE, OUTPUT);
  // Configuramos el pulsador como entrada con resistencia pull-up
  pinMode(PIN_BOTON, INPUT);
}
void loop() {
  if (digitalRead(PIN_BOTON) == HIGH) {
    estadoSemaforo = (estadoSemaforo + 1) % 3; // Cambiar de estado
    delay(200); // Antirrebote
  }
  // Control de LEDs según el estado
  digitalWrite(LED_ROJO, estadoSemaforo == 0);
  digitalWrite(LED_AMARILLO, estadoSemaforo == 1);
  digitalWrite(LED_VERDE, estadoSemaforo == 2);
}
```

Prácticas propuestas

- **Práctica 1**: sistema de riego automático con un sensor de humedad del suelo.
    - Utilizar un sensor de humedad del suelo para detectar si la tierra está seca o húmeda.
    - Si el nivel de humedad es bajo, se activa una bomba de agua o una electroválvula para regar la planta automáticamente.
    - La ESP32 lee los valores del sensor y, dependiendo del umbral configurado, enciende o apaga el sistema de riego.

- **Práctica 2:** termostato que encienda/apague un calentador eléctrico.
    - Utilizar un sensor de temperatura (como el DHT11 o DS18B20) para medir la temperatura ambiente.
    - Si la temperatura cae por debajo de un valor establecido, se activa un calentador eléctrico mediante un relé o un módulo MOSFET.
    - Cuando la temperatura alcanza el nivel deseado, el calentador se apaga lo que permite mantener una temperatura constante de forma automatizada.

- **Práctica 3**: alarma para puertas basada en sensores magnéticos.
    - Consiste en un sistema de seguridad que detecta si una puerta ha sido abierta.
    - Se usa un sensor magnético de efecto Hall o un interruptor reed para detectar la apertura de la puerta.
    - Si el sensor detecta que la puerta se ha abierto, la ESP32 puede activar una alarma sonora (buzzer), encender una luz de advertencia o incluso enviar una notificación a través de una conexión WiFi.

Resumen del capítulo

Este capítulo consolidó todos los conocimientos previos mediante la realización de proyectos integrados más complejos, en los que los estudiantes desarrollaron una estación meteorológica básica, un sistema de control de luces

automatizado, un control de velocidad de ventilador, y un semáforo controlado por botón.

Estos proyectos combinaron múltiples sensores, actuadores y funcionalidades avanzadas de la ESP32, permitiendo a los estudiantes experimentar con aplicaciones prácticas y reales, lo que sirvió como cierre del manual, demostrando cómo los conceptos aprendidos pueden aplicarse en proyectos completos.

Autoevaluación

Preguntas teóricas:

- ¿Qué componentes se necesitan para construir una estación meteorológica básica con la ESP32?
- Describe cómo funciona un sistema de control de luces automatizado con un sensor PIR.
- Explica cómo se controla la velocidad de un ventilador utilizando un sensor de temperatura y un potenciómetro.

Glosario

Este glosario reúne los términos técnicos y conceptos clave utilizados a lo largo del libro para facilitar la comprensión del lector:

1. **ADC (Analog to Digital Converter):** Componente que convierte señales analógicas en datos digitales interpretables por la ESP32.
2. **Actuador:** Dispositivo que realiza una acción física (movimiento, encendido, etc.) en respuesta a una señal eléctrica.
3. **Algoritmo:** Secuencia de pasos ordenados y finitos que permiten resolver un problema o realizar una tarea. Es la base de cualquier programa.
4. **Antirrebote (Debouncing):** Técnica utilizada para evitar lecturas erróneas en botones o interruptores debido a vibraciones mecánicas.
5. **Arduino IDE:** Entorno de desarrollo integrado usado para programar microcontroladores como la ESP32.
6. **Baud Rate:** Velocidad de transmisión de datos en comunicación serial, medida en bits por segundo (bps). Debe coincidir en ambos extremos de la comunicación.
7. **Bluetooth:** Tecnología inalámbrica utilizada para transferir datos en distancias cortas.
8. **Bootloader:** Pequeño programa almacenado en la memoria del microcontrolador que permite cargar nuevos programas desde una computadora.
9. **Ciclo de Trabajo (Duty Cycle):** En PWM, es el porcentaje de tiempo que una señal está en estado alto (encendido) durante un ciclo completo. Controla la intensidad de salida en dispositivos como LEDs y motores.
10. **Frecuencia de CPU:** Velocidad a la que opera el procesador de la ESP32. Puede configurarse en 80 MHz, 160 MHz o 240 MHz.
11. **GPIO (General Purpose Input/Output):** Pines de propósito general en la ESP32 que pueden configurarse como entrada o salida.
12. **IoT (Internet of Things):** Red de dispositivos interconectados capaces de recopilar y compartir datos a través de internet. La ESP32 es una herramienta clave para desarrollar dispositivos IoT.
13. **LDR (Light Dependent Resistor):** Sensor que mide la intensidad de la luz. Su resistencia varía según la cantidad de luz que incide sobre él.

14. **MQ Series:** Familia de sensores de gas que detectan la presencia de gases como metano, butano y monóxido de carbono. Son comunes en sistemas de seguridad y monitoreo ambiental.
15. **Node-RED:** Herramienta de programación visual basada en flujos, utilizada para conectar dispositivos IoT y crear aplicaciones de automatización.
16. **PWM (Pulse Width Modulation):** Técnica para generar señales analógicas utilizando salidas digitales.
17. **Pull-up/Pull-down:** Resistencias utilizadas para estabilizar el estado de un pin digital cuando no está activo. Una resistencia pull-up mantiene el pin en alto (HIGH), mientras que una pull-down lo mantiene en bajo (LOW).
18. **Resolución ADC:** Número de bits que utiliza un ADC para representar una señal analógica. En la ESP32, la resolución por defecto es de 12 bits (0-4095).
19. **Sensor:** Dispositivo que detecta cambios en el entorno y los convierte en señales eléctricas.
20. **UART (Universal Asynchronous Receiver-Transmitter):** Protocolo de comunicación serie usado por la ESP32.
21. **WiFi:** Tecnología inalámbrica que permite la conexión a redes locales e internet.

Referencias

[1] J. L. Andrango Quilumba, “Implementación de un prototipo de monitoreo de espacios en un parqueadero basado en ESP32 y página web.,” Oct. 2024, Accessed: Jan. 29, 2025. [Online]. Available: http://bibdigital.epn.edu.ec/handle/15000/25833

[2] “Iniciando a Programar con Python.: Guía básica de programación - Lina María Cuervo Díaz, Nathalia Andrea Cuervo Díaz, Javier Cuervo Álvarez - Google Libros.” Accessed: Jan. 29, 2025. [Online]. Available: https://books.google.com.ec/books?hl=es&lr=&id=GpYkEQAAQBAJ&oi=fnd&pg=PA17&dq=%E2%80%A2%09Lenguajes+interpretados:+Son+aquellos+en+los+que+el+c%C3%B3digo+se+ejecuta+l%C3%ADnea+por+l%C3%ADnea+mediante+un+int%C3%A9rprete.+Ejemplo:+Python&ots=w8pMFeRzno&sig=8iS0em32MhlUpX91qxVNuHe2hYI&redir_esc=y#v=onepage&q&f=false

[3] “Lenguajes de programación - MARTÍN VILLALBA Carla , URQUÍA MORALEDA Alfonso , RUBIO GONZÁLEZ Miguel Ángel - Google Libros.” Accessed: Jan. 29, 2025. [Online]. Available: https://books.google.com.ec/books?hl=es&lr=&id=qms4EAAAQBAJ&oi=fnd&pg=PA1&dq=Lenguajes+compilados:+El+c%C3%B3digo+se+traduce+completamente+a+un+lenguaje+de+m%C3%A1quina+antes+de+ejecutarse&ots=pQFYayyM92&sig=0m5Ys0G9KeWW8qzYHRH7i0Vr_wo&redir_esc=y#v=onepage&q&f=false

[4] S. Florida, S. Roy Clinton Daniel, M. Agrawal, P. Brescia, C. Olivia Ocampo, and C. Olivia, “Capítulo 13 Lenguajes informáticos, aplicaciones y tecnologías Capítulo 13 Lenguajes informáticos, aplicaciones y tecnologías emergentes emergentes Scholar Commons Citation Scholar Commons Citation”, Accessed: Jan. 29, 2025. [Online]. Available: https://digitalcommons.usf.edu/dit_tb_spa/13

[5] J. E. V. Sampedro, J. E. V. Sampedro, S. P. C. Machado, N. M. M. Ramírez, and F. E. C. Estrella, “Fundamentos de programación: una aproximación al desarrollo del pensamiento,” *Polo del Conocimiento*, vol. 9, no. 8, pp. 3292–3309, Aug. 2024, doi: 10.23857/pc.v9i8.7881.

[6] “Overview of the Arduino IDE 1 | Arduino Documentation.” Accessed: Jan. 30, 2025. [Online]. Available: https://docs.arduino.cc/software/ide-v1/tutorials/Environment/?_gl=1*q5ya5h*_up*MQ..*_ga*MTkzNjc5Nzc1OC4x

NzM2ODYzMDM4*_ga_NEXN8H46L5*MTczNjg2MzAzNy4xLjAuMTczNjg2MzAzNy4wLjAuNTkyNjE0OTEy

[7] Asim. Zulfiquar, “HANDS-ON ESP32 WITH ARDUINO IDE unleash the power of IoT with ESP32 and build exciting projects with this practical guide,” 2024.

[8] H. K. Kondaveeti, N. K. Kumaravelu, S. D. Vanambathina, S. E. Mathe, and S. Vappangi, “A systematic literature review on prototyping with Arduino: Applications, challenges, advantages, and limitations,” *Comput Sci Rev*, vol. 40, p. 100364, May 2021, doi: 10.1016/J.COSREV.2021.100364.

[9] “ESP32 Wi-Fi & Bluetooth SoC | Espressif Systems.” Accessed: Jan. 30, 2025. [Online]. Available: https://www.espressif.com/en/products/socs/esp32

[10] “▷ Comparación de Arduino vs ESP8266 vs ESP32 (2025).” Accessed: Jan. 30, 2025. [Online]. Available: https://elosciloscopio.com/comparacion-arduino-vs-esp8266-vs-esp32/

[11] “Sensors and Actuators | part of Nanonetworks: The Future of Communication and Computation | Wiley-IEEE Press books | IEEE Xplore.” Accessed: Jan. 30, 2025. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10623922

[12] P. Docs and D. • Simulator, “Connecting to a DHTxx Sensor Using a DHTxx Sensor with Arduino DHT CircuitPython Code • Adafruit CircuitPython Module Install • Wiring • Usage • Example Code”, Accessed: Jan. 30, 2025. [Online]. Available: http://adafru.it/

[13] I. A. Ayoade, O. A. Adeyemi, O. A. Adeaga, R. O. Rufai, and S. B. Olalere, “Development of Smart (Light Dependent Resistor, LDR) Automatic Solar Tracker,” *Proceedings of the 5th International Conference on Information Technology for Education and Development: Changing the Narratives Through Building a Secure Society with Disruptive Technologies, ITED 2022*, 2022, doi: 10.1109/ITED56637.2022.10051239.

[14] A. Sudarmanto, M. A. Khalif, A. K. Huda, and A. Khoirul Huda, “Detection of building slope and land subsidence using ultrasonic HC-SR04 sensors based Arduino Uno R3 and Blynk,” vol. 2540, p. 100004, 2023, doi: 10.1063/5.0125207.

[15] “Winsen MQ Sensor,MQ Series Gas Sensor.” Accessed: Jan. 30, 2025. [Online]. Available: https://www.winsen-sensor.com/mq-sensor.html?campaignid=10463189402&adgroupid=106436716769&feeditemid

=&targetid=kwd-355576591810&device=c&creative=483241564758&keyword=mq%20gas%20sensors&gad_source=1&gclid=Cj0KCQiAhvK8BhDfARIsABsPy4hCEjRlxipZeuBTi8Is8vy8FM3mLR5ljqvxtBCtdO0E_vn6pUfkl5MaAqY0EALw_wcB

[16] B. Yang *et al.*, "Arduino Based Security System using Passive Infrared (PIR) Motion Sensor," *IOP Conf Ser Earth Environ Sci*, vol. 655, no. 1, p. 012039, Feb. 2021, doi: 10.1088/1755-1315/655/1/012039.

[17] "Sensor de humedad del suelo YL38 y YL69 — Talos Electronics." Accessed: Jan. 30, 2025. [Online]. Available: https://www.taloselectronics.com/products/sensor-de-humedad-del-suelo-yl38-y-yl69

[18] V. Ranka, D. Shah, D. Shah, and P. Saval, "Self-Adaptive Smart Lighting System using Yolov8 and ESP32," *10th International Conference on Electrical Energy Systems, ICEES 2024*, 2024, doi: 10.1109/ICEES61253.2024.10776837.

[19] H. Kareem and D. Dunaev, "The Working Principles of ESP32 and Analytical Comparison of using Low-Cost Microcontroller Modules in Embedded Systems Design," *2021 4th International Conference on Circuits, Systems and Simulation, ICCSS 2021*, pp. 130–135, May 2021, doi: 10.1109/ICCSS51193.2021.9464217.

[20] "UART: A Hardware Communication Protocol Understanding Universal Asynchronous Receiver/Transmitter | Analog Devices." Accessed: Jan. 30, 2025. [Online]. Available: https://www.analog.com/en/resources/analog-dialogue/articles/uart-a-hardware-communication-protocol.html

ESP32: MANUAL BÁSICO PARA ESTUDIANTES

ALUMNI EDITORA 2025

PRIMERA EDICIÓN

ISBN: 978-9942-7307-6-3

ORTIZ ARCINIEGA – VALENCIA BARAHONA – BOSMEDIANO CÁRDENAS – BASTIDAS JÁCOME – AGUIRRE CHAGNA – JÁCOME AYALA