sábado, 12 de abril de 2025

Sensor de Temperatura y Humedad DHT22

El sensor DHT22 utiliza tres pines principales: uno para VCC (voltaje de 3.3V a 5V), otro para GND, y uno para datos. Para alimentar el DHT22, conectaremos el pin VCC a la salida de 3.3V del kit de desarrollo ESP32 DevKit V1, y el pin GND del sensor al pin GND del kit. El pin de datos del DHT22 se conectará al GPIO 25, asegurando que no esté en uso por otra función por parte de ESP32 para evitar conflictos. Esto garantiza una correcta transmisión de los datos de temperatura y humedad a través de una comunicación digital de un solo cable, con un consumo aproximado de 4 mA.

Datasheet con informacion del DHT22:

https://cdn.sparkfun.com/assets/f/7/d/9/c/DHT22.pdf

ESP32, SoC desarrollado por Espressif, dentro del SoC de ESP32 podemos encontrar:

Cores: Doble núcleo Xtensa LX6, con frecuencias de hasta 240 MHz.

Conectividad: Wi-Fi y Bluetooth integrados.

Memoria: SRAM de 520 KB y Flash externa.

Periféricos: ADC, DAC, SPI, I2C, UART, PWM, entre otros.

Seguridad: Soporta encriptación y autenticación.

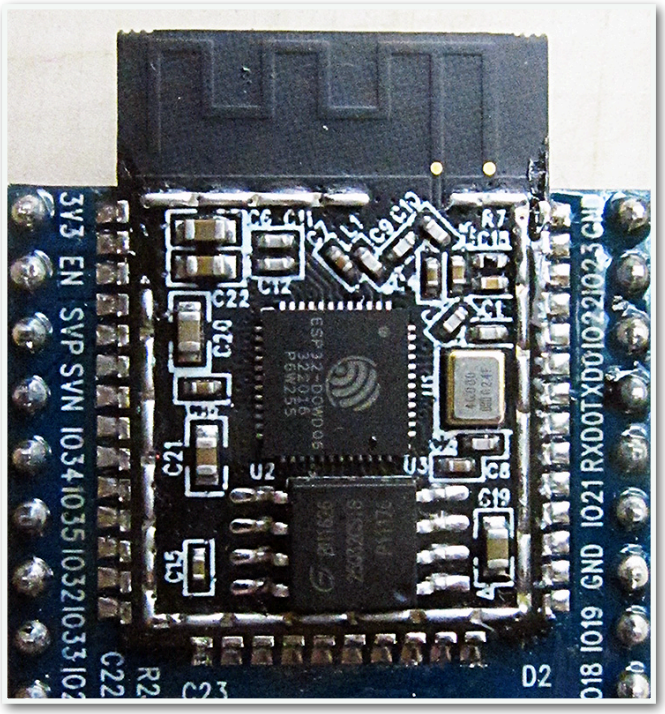
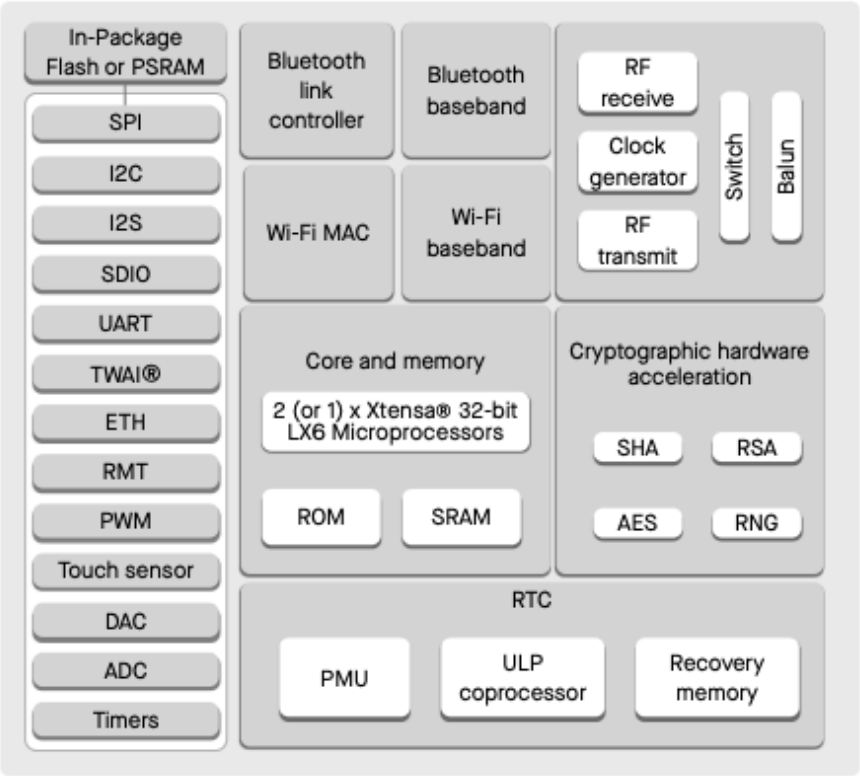


Diagrama de bloques de las funciones que puede realizar el SoC ESP32:



ESP32 Functional Block Diagram

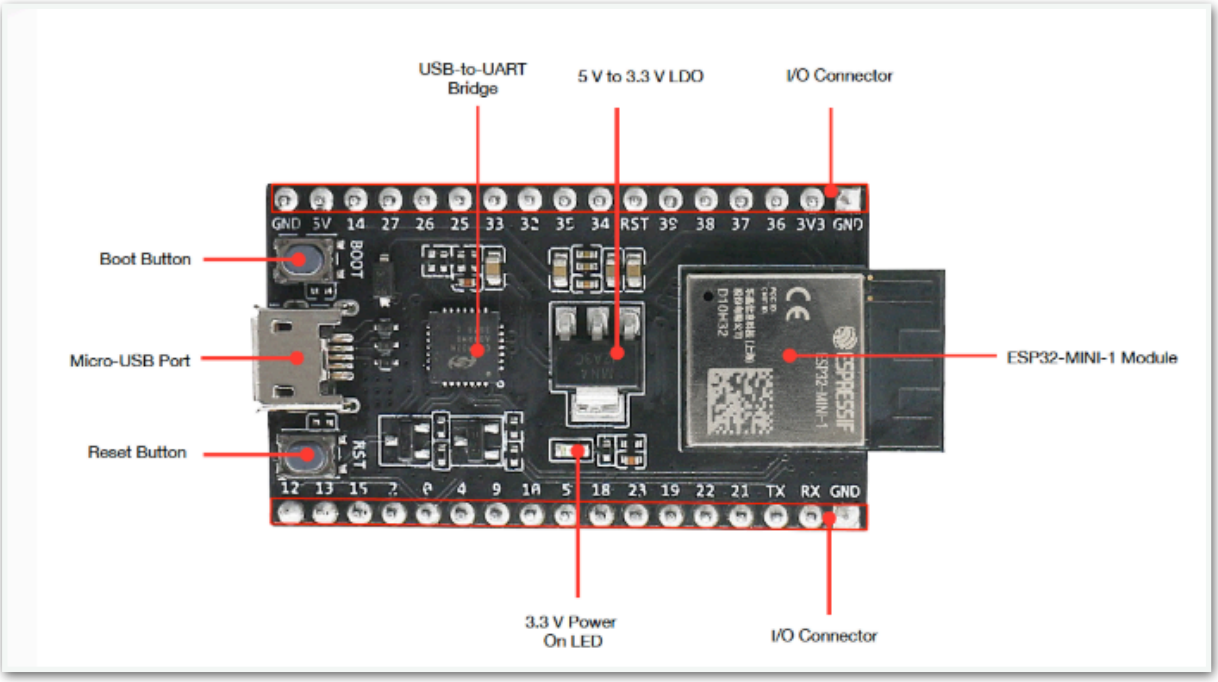
Para nuestro proyecto vamos a utilizar "Core" los procesadores de Tensilica, memoria SRAM para trabajar con el codigo, memoria ROM para las funciones principales de trabajo del chip. Memoria PSRAM externa al SoC ESP32 y util para la comunicacion entre los puertos o pines del KitDev V1 y el SoC ESP32, por ejemplo en nuestro proyecto la memoria PSRAM sera un intermediario entre la comunicacion por USB con el ordenador y el procesador de la ESP32 dentro del SoC ESP32.

Kit de desarrollo de Espressif , en concreto este Kit es : ESP32 DevKit V1.

El ESP32 DevKit V1 incluye diversos componentes para facilitar su uso:

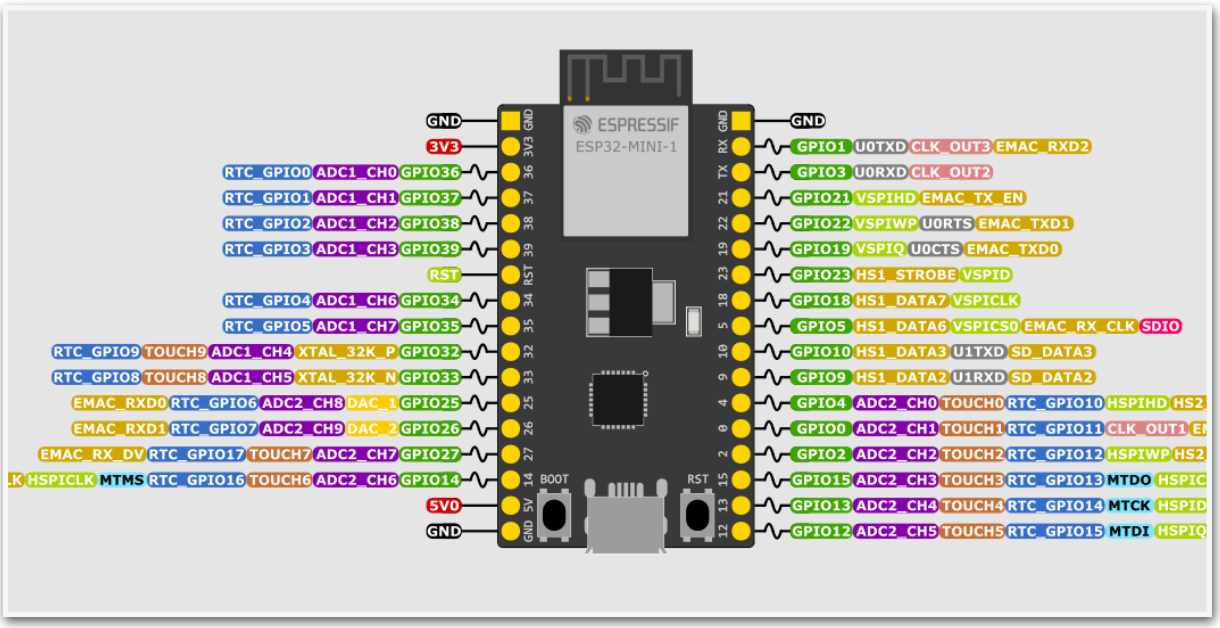
1. **Pines GPIO**: Permiten la comunicación con el SoC del ESP32.
2. **Puente UART**: Un chip que facilita la comunicación con el puerto micro USB.
3. **Regulador de voltaje**: Proporciona 5V y protege la placa.
4. **LED de encendido**: Indica cuando la placa está en funcionamiento.
5. **Botones**: Uno para reset y otro para entrar en modo bootloader para cargar programas.

En nuestro proyecto utilizaremos la comunicacion por USB con el ordenador, esta comunicacion sera Serial, utilizando el protocolo UART que utiliza RX para recibir informacion y TX para transmitir informacion. Todo esto esta integrado en el cable USB y lo gestiona normalmente el chip CP2102, chip UART del KitDev V1.



https://docs.espressif.com/projects/esp-dev-kits/en/latest/esp32/esp32-devkitm-1/user_guide.html#getting-started

Pines del kit de desarrollo de la ESP32 llamado DevKit V1.



En nuestro proyecto utilizaremos el pin de datos GPIO 25, como podemos ver en la imagen el pin GPIO 25 lo podemos utilizar para entradas y salidas de señal digitales. Tenemos que tener en cuenta la disposicion de los pines del KitDev V1 ESP32 y las funciones que realiza cada pin para no tener problemas al utilizar pines reservados en alguna funcion que realice nuestro codigo, esto es asi, porque las funciones de la ESP32 son mas numerosas que los pines que dispone el KitDev V1, por este motivo por ejemplo si utilizamos el protocolo SPI o el I2C para comunicarnos con algun sensor, entonces, en ese caso, para el pin de datos del DHT22 deberemos tener cuidado en no elegir un pin que utilice estos protocolos, para no crear conflicto de señales.

A continuacion esta el codigo comentado y descrito linea por linea:

```
#include <Arduino.h>   // Incluye la biblioteca principal de Arduino

#include <DHT.h>       // Incluye la biblioteca para manejar sensores DHT


// ------------------------------

// Configuración del sensor DHT

// ------------------------------


// Define el pin GPIO 33 del ESP32 al que está conectado el pin de datos del sensor

#define DHTPIN 25


// Define el tipo de sensor DHT utilizado (puede ser DHT11 o DHT22)

#define DHTTYPE DHT11


// Crea una instancia del sensor DHT con el pin y tipo definidos

DHT dht(DHTPIN, DHTTYPE);


// ------------------------------

// Función de inicialización

// ------------------------------

void setup() {
```

```
  Serial.begin(9600);       // Inicializa la comunicación por el puerto serie a 9600 baudios


  dht.begin();              // Inicializa el sensor DHT
  Serial.println("Sensor DHT inicializado.");  // Mensaje de confirmación en el monitor serie
}


// ----------------------------
// Bucle principal
// ----------------------------
void loop() {
  // Lee la temperatura (en grados Celsius) y la guarda en la variable 'temp'
  float temp = dht.readTemperature();


  // Lee la humedad relativa (en porcentaje) y la guarda en la variable 'hum'
  float hum = dht.readHumidity();


  // Verifica si alguna de las lecturas ha fallado
  if (isnan(temp) || isnan(hum)) {
    Serial.println("Error al leer el sensor DHT");  // Muestra un mensaje de error si la lectura falla
  } else {
```

```
    // Si las lecturas son válidas, muestra los valores en el monitor
serie
    Serial.print("Temperatura: ");
    Serial.print(temp);
    Serial.print(" °C\tHumedad: ");
    Serial.print(hum);
    Serial.println(" %");
  }

  delay(2000);  // Espera 2 segundos antes de realizar otra lectura
}
```

La siguiente parte del proyecto es crear un script en codigo python que establezca una comunicacion por el puerto USB con la placa ESP32, tendremos en cuenta que el sistema operativo es Windows.

En Windows para conocer el puerto que el sistema operativo le ha asignado a la plca ESP32 deberemos movernos a "administrador de dispositivos" y clickear en puertos COM, una vez conectada nuestra placa ESP32 por USB deberiamos poder ver que puerto COM tiene asignado, esto nos sera util para nuestro script python.

Porque elegimos el codigo C o C++ para programar el microprocesador de la ESP32 y elegimos codigo python para la inteligencia artificial y comunicacion por USB?

¿POR QUÉ USAMOS C++ PARA PROGRAMAR LA ESP32?

Ventajas clave en este contexto:

1. **Rendimiento en tiempo real**: La ESP32 necesita ejecutar tareas con precisión temporal (ej. leer sensores, activar pines, manejar interrupciones). C++ permite eso.
2. **Control total del hardware**: Puedes manipular registros, usar pines GPIO, configurar periféricos como UART, SPI, I2C, PWM, etc.
3. **Librerías específicas**: El entorno Arduino y ESP-IDF están escritos en C/C++ y brindan un ecosistema completo para trabajar con el hardware.
4. **Uso eficiente de recursos**: C++ permite trabajar con bajo uso de memoria y procesamiento, lo cual es crítico en microcontroladores.

Desventajas menores:

- La programación puede ser más compleja (punteros, gestión de memoria).
- Menor productividad para tareas de alto nivel (como lógica de inteligencia artificial o visualización de datos).

¿POR QUÉ USAMOS PYTHON EN EL ORDENADOR PARA COMUNICARNOS CON LA ESP32 O HACER IA?

Ventajas clave en este contexto:

1. **Facilidad de desarrollo**: Crear scripts en Python es mucho más rápido, ideal para depuración, prototipos y automatización.
2. **Serial por USB muy fácil**: Con bibliotecas como `pyserial` puedes abrir un puerto COM en 2 líneas y comunicarte con la ESP32.
3. **Excelentes herramientas para IA**: Python tiene frameworks muy potentes como TensorFlow, PyTorch, Scikit-learn.

4. **Entornos de trabajo modernos**: Funciona muy bien con Jupyter, VSCode, entornos virtuales, y se integra con librerías de gráficos, bases de datos, etc.

Desventajas menores:

- No tiene acceso directo al hardware del sistema embebido.
- Más lento si se usara para tareas de tiempo real.

¿CUÁNDO USAR UNO U OTRO?

Escenario	¿C++ o Python?
Leer sensores y controlar actuadores con ESP32	**C++**
Enviar comandos desde el ordenador a la ESP32	**Python**
Procesar los datos de sensores en la nube	**Python**
Crear interfaz gráfica para monitorear la ESP32	**Python**
Hacer una red neuronal que decida una acción	**Python**
Ejecutar esa acción en la ESP32	**C++**

CONCLUSIÓN PRÁCTICA

- Usamos **C++ en la ESP32** porque necesitamos precisión, rendimiento y acceso directo al hardware.
- Usamos **Python en el PC** porque nos permite interactuar de forma sencilla con la placa, analizar datos, hacer IA, y automatizar tareas con menos esfuerzo.

Codigo python para establecer una comunicacion por USB desde el ordeandor hasta la ESP32, y crear automaticamente un archivo de texto llamado "datos.txt" con la informacion leida por el detector DHT22:

```
# Importa la biblioteca 'serial' para comunicarse con la ESP32 por el puerto serie
import serial

# Importa la biblioteca 'time' para gestionar pausas y obtener la hora actual
import time

# ------------------------------------------------
# Configuración del puerto serie
# ------------------------------------------------

# Define el puerto serie al que está conectada la ESP32 (ajusta según tu equipo)
PUERTO_SERIAL = 'COM5'  # Por ejemplo, en Windows puede ser COM5, COM3, etc.

# Define la velocidad de transmisión de datos (debe coincidir con la definida en la ESP32)
BAUDRATE = 9600

# ------------------------------------------------
# Apertura del puerto serie
```

```
# ---------------------------------------------

try:
    # Intenta abrir el puerto serie con la configuración indicada
    esp32 = serial.Serial(PUERTO_SERIAL, BAUDRATE, timeout=1)

    # Si la conexión fue exitosa, se muestra un mensaje en consola
    print(f"Conectado a {PUERTO_SERIAL} a {BAUDRATE} baudios.")
except serial.SerialException as e:
    # Si hubo un error al abrir el puerto (por ejemplo, no existe o está ocupado), se muestra un mensaje y se termina el programa
    print(f"Error al conectar con el puerto serial: {e}")
    exit()

# ---------------------------------------------
# Apertura del archivo y lectura continua
# ---------------------------------------------

# Abre el archivo 'datos.txt' en modo adjuntar ('a') para no sobrescribir datos previos
# Se usa codificación UTF-8 para soportar caracteres especiales
with open("datos.txt", "a", encoding="utf-8") as archivo:
    try:
        # Bucle principal: se ejecuta indefinidamente hasta que el usuario lo detenga
```

```
        while True:
            # Verifica si hay datos disponibles en el buffer de entrada del puerto serie
            if esp32.in_waiting > 0:
                # Lee una línea del puerto serie, decodifica a UTF-8 y elimina espacios y saltos de línea
                dato = esp32.readline().decode('utf-8').strip()

                # Obtiene la hora actual con formato "YYYY-MM-DD HH:MM:SS"
                timestamp = time.strftime('%Y-%m-%d %H:%M:%S', time.localtime())

                # Escribe la línea de datos con el timestamp en el archivo
                archivo.write(f"{timestamp} - {dato}\n")

                # Fuerza a escribir los datos en el disco sin esperar a cerrar el archivo
                archivo.flush()

                # Muestra por pantalla que se ha guardado un dato
                print(f"Guardado: {dato}")

            # Espera 20 segundos antes de leer de nuevo
            time.sleep(20)
```

```
    except KeyboardInterrupt:
        # Si el usuario presiona Ctrl+C, se interrumpe el bucle y se muestra un mensaje
        print("Lectura detenida por usuario.")

    finally:
        # Al finalizar el programa (ya sea por error o por interrupción del usuario), se cierra el puerto serie
        esp32.close()
```

Guía para ejecutar el script de Python desde la consola de comandos en Windows

Para ejecutar el script de Python que se comunica con la placa ESP32, sigue estos pasos:

1. **Abrir la Consola de Comandos (CMD)**:
   - Presiona `Win + R`, escribe `cmd` y pulsa `Enter`.
2. **Navegar al Directorio del Archivo**:
   - Usa el comando `cd` (change directory) para ir al directorio donde está guardado el archivo Python. Por ejemplo, si el archivo está en `C:\Proyectos\ESP32`, escribe:
     ```
     cd C:\Proyectos\ESP32
     ```
3. **Ejecutar el Script de Python**:
   - Una vez en el directorio, ejecuta el script usando el comando `python` seguido del nombre del archivo. Si el archivo se llama `comunicacion_esp32.py`, escribe:
     ```
     python comunicacion_esp32.py
     ```

- Luego presiona `Enter` para ejecutar el script.

4. **Visualizar el Resultado**:
    - El script se conectará a la placa ESP32 a través del puerto USB y comenzará a leer y guardar los datos en el archivo `datos.txt`.

Esta secuencia te permitirá ejecutar tu script de Python de manera sencilla desde la consola de comandos.

Importante no tener mas de un programa accediendo al mismo puerto COM, es posible que si tenemos VSCode abierto con una ccomunicacion establecida con la ESP32, entonces podriamos tener problemas para comunicarnos con nuestro script de python por el mismo puerto COM.