LA PROGRAMACIÓN DE ESP32 CON EJEMPLOS

Ph.D. Andrés Gallego

Doctor en Ingeniería Eléctrica

Profesor - Investigador TC

Ingeniería Electrónica

Universidad Cooperativa de Colombia

Sede Bogotá, Colombia

Correo: andres.gallego@campusucc.edu.co

CvLAC: https://scienti.minciencias.gov.co/cvlac/visualizador/generarCurriculoCv.do?cod_rh=0001539135

ORCID: https://orcid.org/0000-0002-5852-5788

Google Scholar: https://scholar.google.com.ec/citations?user=ca3u1iYAAAAJ&hl=lv

Contents

Resumen

La programación de microcontroladores implica una serie de etapas esenciales para su correcto funcionamiento. En primer lugar, se realiza el montaje, que incluye la conexión electrónica del circuito integrado a las fuentes de alimentación y otros componentes necesarios. Esta fase garantiza que el hardware esté preparado para recibir y ejecutar instrucciones. Posteriormente, se instala el entorno de programación donde se desarrollará el firmware, que es el software encargado de describir las tareas específicas que el microcontrolador debe llevar a cabo.

Una vez que se ha escrito el firmware, se procede a la compilación de este código, transformándolo en un formato que el microcontrolador pueda entender y ejecutar. Luego, se suben estas instrucciones compiladas al dispositivo. En esta fase final, es crucial conectar los sensores, actuadores o cualquier otro subsistema que sea parte del funcionamiento global del proyecto. Estas conexiones permiten verificar y probar el rendimiento del microcontrolador en un entorno real.

La tarjeta ESP32 simplifica considerablemente este proceso al ofrecer una placa con las conexiones necesarias, conexiones inalámbricas y un entorno de programación basado en Arduino IDE. Este entorno es tanto software como hardware abierto, lo que facilita su acceso y uso para una amplia gama de usuarios. A lo largo de este documento, exploraremos cómo utilizar estas plataformas para manejar el ESP32, programarlo y ponerlo a prueba con aplicaciones básicas de lectura de sensores y control de actuadores, demostrando su versatilidad y eficiencia en proyectos electrónicos.

Palabras clave

Arduino IDE, ESP32, Microcontroladores, WiFi, Bluetooth, sensores, actuadores, programación, Electrónica Digital, compilación.

Prefacio

Este documento es para las personas que tienen algún conocimiento básico en electrónica, o ningún tipo de conocimiento en esta materia, pero desean aprender una programación sencilla de Microcontroladores, siendo estudiantes, profesionales, o simplemente apasionados por este tema. La idea es ofrecerles una visión general de cómo se programan actualmente algunos dispositivos digitales para el control de sensores y de actuadores, lo que quiere decir que leeremos valores del mundo físico y los transformaremos a información digital que podamos procesar, para después tener la capacidad de decirle a un actuador (como una luz LED) cada cuántos segundos debe parpadear y con qué velocidad, por ejemplo.

Preparación de las herramientas

ESP32

El ESP32 utiliza un microprocesador Tensilica Xtensa LX6, disponible en versiones de uno o dos núcleos, y está equipado con interruptores de antena, balun de radiofrecuencia, amplificador de potencia, amplificador de recepción de bajo ruido, filtros y módulos de gestión de energía. Fue diseñado y desarrollado por Espressif Systems y es producido por TSMC utilizando un proceso de fabricación de 40 nm. Es el sucesor del SoC ESP8266 (Espressif, 2024).

La tarjeta de desarrollo que utilizaremos es la ESP 32 DEVKIT V1, que se muestra en la imagen a continuación y que tiene el microchip ESP32 con las conexiones necesarias para hacer desarrollos rápidos en electrónica.

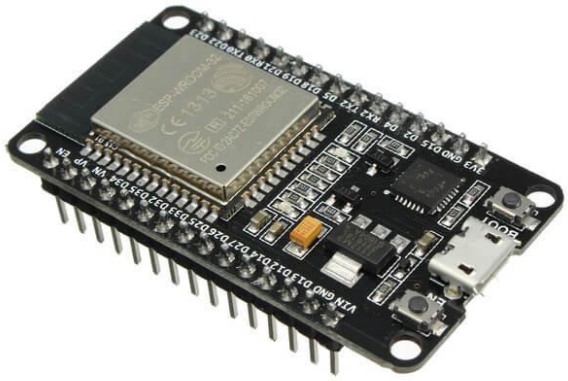


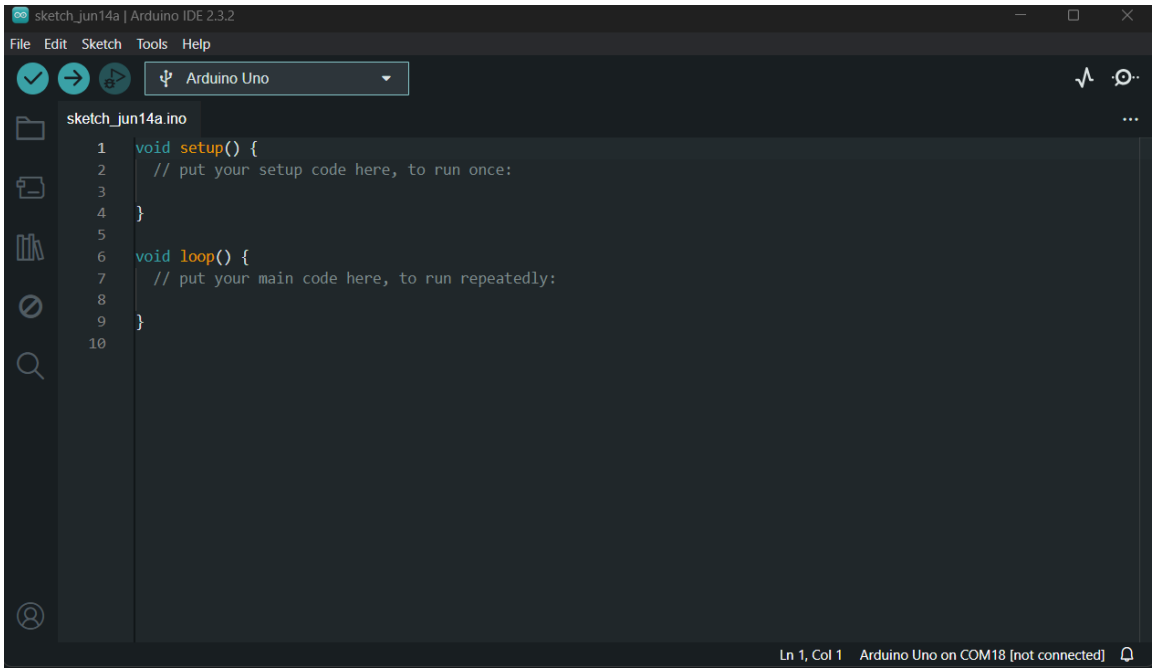
Figura 1. Imagen la tarjeta de desarrollo ESP32 (Electronilab, 2024)

Arduino IDE

Este es el entorno de desarrollo integrado (IDE) que permite programar la tarjeta, enfocándose en el microcontrolador. Aquí podemos escribir el código, compilarlo y transferirlo a la tarjeta para comprobar que las tareas se ejecuten correctamente. Una vez más, es necesario ir a la página de Arduino (https://www.arduino.cc/en/software) y descargarlo gratuitamente, como se muestra en la figura siguiente.

Figura 2. Página web para la descarga de Arduino IDE (Arduino, 2024).

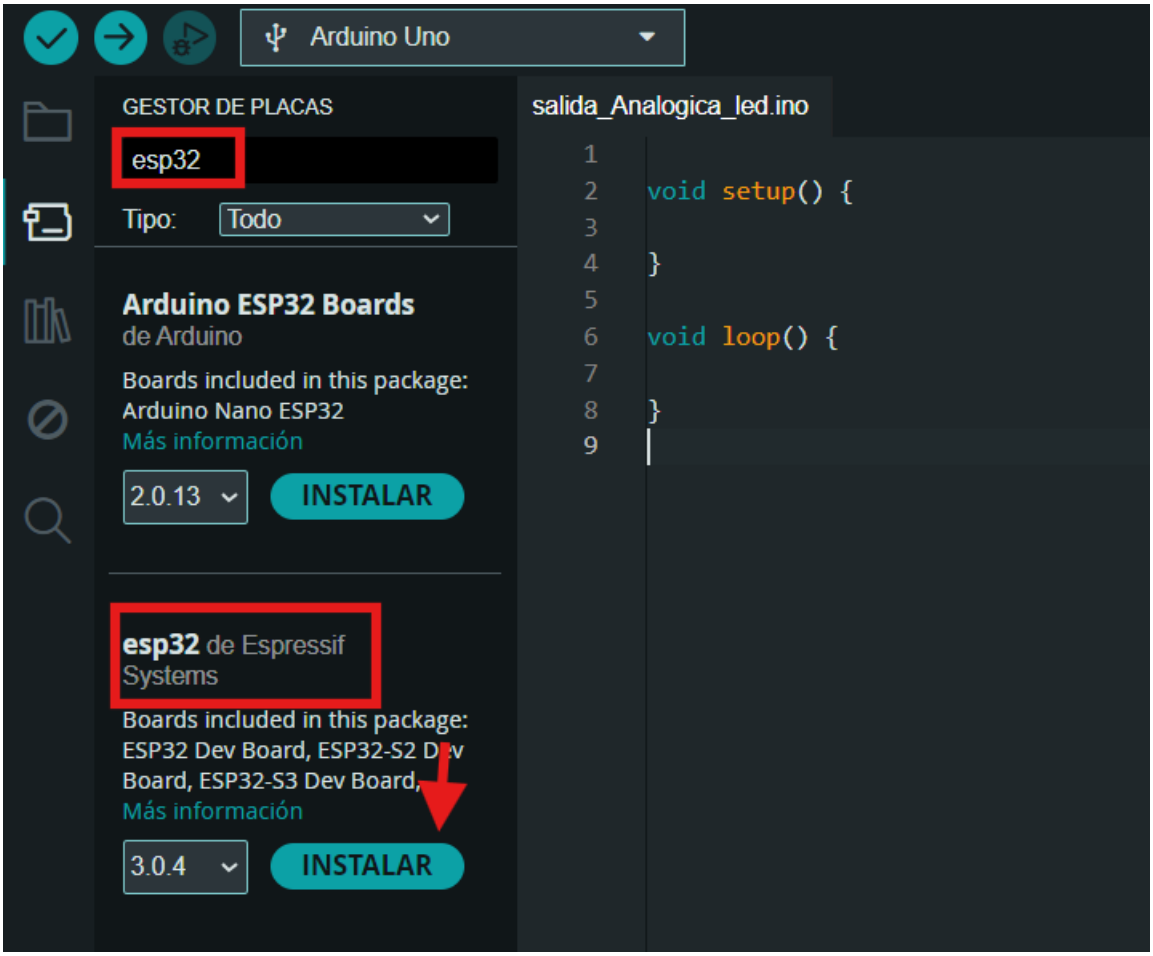
Una vez descargado, entonces procedemos con su instalación, que usualmente no tiene ningún inconveniente. Una vez instalado, podemos abrir el programa y se verá de la siguiente manera:



Debemos ahora hacer la instalación de las herramientas que permiten la programación de la ESP32. Empezamos por poner una URL adicional para poder descargar de repositorios diferentes a los de Arduino. En *Archivo->Preferencias* debemos poner el siguiente enlace:

https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json

Le damos click en aceptar, y procedemos a la instalación de herramientas automáticamente. Esto se hace desde *Herramientas->Placa->Gestor de placas*, buscando ESP32 y seleccionando *esp32 by Espressif*. En este caso se instala la versión 3.0.4 como se muestra en la siguiente figura:



Cuando ya esté instalado (por lo general toma algunos minutos), reiniciamos la aplicación de Arduino IDE.

Ya con las herramientas instaladas en la IDE, debemos asegurarnos de instalar los controladores USB a UART del chip CP2102 (o el que corresponda) que trae la tarjeta para poder comunicarse con el computador. Para hacer esto, vamos a descargar de https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers?tab=downloads el software llamado CP210x Windows Drivers. Lo descomprimimos y ejecutamos el instalador (en nuestro caso el de 64 bits) como lo muestra la figura a continuación:

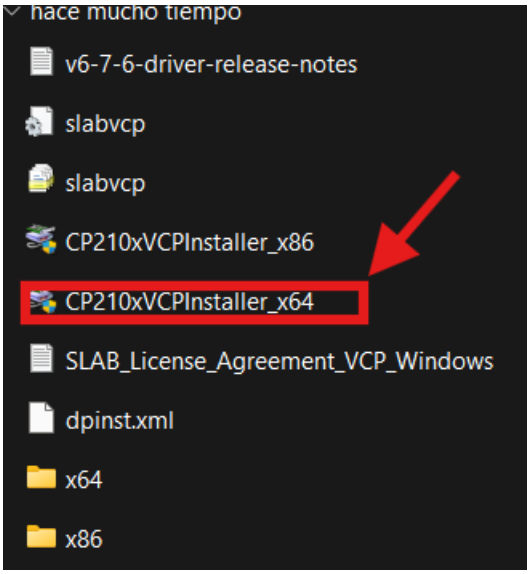


Figura 3. Instalador de la comunicación USB-UART en el computador.

Ya con este controlador instalado, tenemos todas las herramientas para poder programar la tarjeta sin problemas. En la siguiente sección se hará un ejemplo de prueba para verificar que las herramientas necesarias estén funcionando.

Prueba inicial de funcionamiento de las herramientas

Vamos a cargar un ejemplo que tiene la función de identificar las redes WiFi disponibles y mostrarlas en la consola serial del IDE. Esto nos permitirá verificar el funcionamiento tanto de las herramientas como de la tarjeta. Para esto, en el IDE vamos a *Archivo->Ejemplos->WiFi->WiFiScan* y lo seleccionamos. Se cargará el código automáticamente. Debemos asegurarnos de tener un **cable** de buena calidad que permita la **transferencia de datos** y no solamente la carga de dispositivos. Podemos también copiar y pegar el siguiente código:

```
/*
 *  This sketch demonstrates how to scan WiFi networks.
 *  The API is based on the Arduino WiFi Shield library, but has significant
changes as newer WiFi functions are supported.
 *  E.g. the return value of `encryptionType()` different because more
modern encryption is supported.
 */
#include "WiFi.h"

void setup() {
  Serial.begin(115200);
```

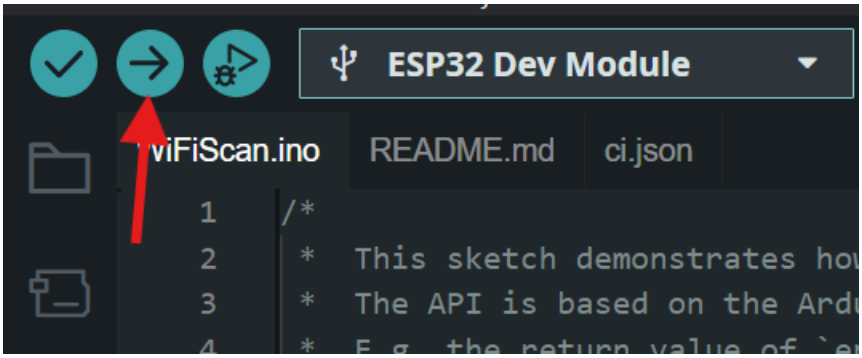
```
  // Set WiFi to station mode and disconnect from an AP if it was previously
connected.
  WiFi.mode(WIFI_STA);
  WiFi.disconnect();
  delay(100);

  Serial.println("Setup done");
}

void loop() {
  Serial.println("Scan start");

  // WiFi.scanNetworks will return the number of networks found.
  int n = WiFi.scanNetworks();
  Serial.println("Scan done");
  if (n == 0) {
    Serial.println("no networks found");
  } else {
    Serial.print(n);
    Serial.println(" networks found");
    Serial.println("Nr | SSID                             | RSSI | CH |
Encryption");
    for (int i = 0; i < n; ++i) {
      // Print SSID and RSSI for each network found
      Serial.printf("%2d", i + 1);
      Serial.print(" | ");
      Serial.printf("%-32.32s", WiFi.SSID(i).c_str());
      Serial.print(" | ");
      Serial.printf("%4ld", WiFi.RSSI(i));
      Serial.print(" | ");
      Serial.printf("%2ld", WiFi.channel(i));
      Serial.print(" | ");
      switch (WiFi.encryptionType(i)) {
        case WIFI_AUTH_OPEN:            Serial.print("open"); break;
        case WIFI_AUTH_WEP:             Serial.print("WEP"); break;
        case WIFI_AUTH_WPA_PSK:         Serial.print("WPA"); break;
        case WIFI_AUTH_WPA2_PSK:        Serial.print("WPA2"); break;
        case WIFI_AUTH_WPA_WPA2_PSK:    Serial.print("WPA+WPA2"); break;
        case WIFI_AUTH_WPA2_ENTERPRISE: Serial.print("WPA2-EAP"); break;
        case WIFI_AUTH_WPA3_PSK:        Serial.print("WPA3"); break;
        case WIFI_AUTH_WPA2_WPA3_PSK:   Serial.print("WPA2+WPA3"); break;
        case WIFI_AUTH_WAPI_PSK:        Serial.print("WAPI"); break;
        default:                        Serial.print("unknown");
      }
      Serial.println();
```

```
      delay(10);
    }
  }
  Serial.println("");

  // Delete the scan result to free memory for code below.
  WiFi.scanDelete();

  // Wait a bit before scanning again.
  delay(5000);
}
```

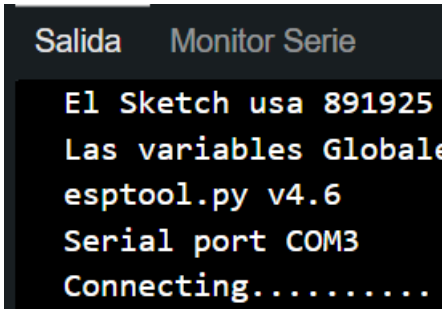
En la parte superior de la interfaz, seleccionamos “Seleccione otra placa y puerto…” para decirle al programa que lo cargue a la tarjeta ESP32 como se muestra a continuación:

Una vez allí, buscamos y seleccionamos la placa y el puerto en la cual está conectado:

Después de los pasos anteriores, es necesario cargar el código a la placa. Presionamos en la IDE el botón para compilar y cargar:



Aparecerán algunos mensajes en la parte inferior del software, y cuando aparezca en pantalla el mensaje de “Connecting”, **presionamos el botón llamado “Boot”** en la placa ESP32 durante unos segundos. Esto es necesario para poder cargar el programa a la tarjeta.



Cuando la tarjeta esté programada correctamente, saldrá en la parte inferior de la IDE algo como lo siguiente, que quiere decir que se está subiendo el código a la tarjeta:

```
Writing at 0x000d5c57... (91 %)
Writing at 0x000dadf9... (94 %)
Writing at 0x000e0bca... (97 %)
Writing at 0x000e60e0... (100 %)
Wrote 898496 bytes (588783 compressed) at 0x00010000 in 9.7 seconds (effective 744.2 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
```

Luego, abrimos la consola serial de la IDE y ajustamos la velocidad de baudios a 115200. Debería mostrar algo como lo de la figura a continuación, lo que significa que está identificando las redes WiFi disponibles y todo funciona correctamente:

```
Scan start
Scan done
62 networks found
Nr | SSID                             | RSSI | CH | Encryptio
 1 | FYL.R_2.4GHZ                     |  -54 |  6 | WPA+WPA2
 2 | FYL_2.4GHZ                       |  -79 |  1 | WPA+WPA2
 3 | NATALIA                          |  -81 |  1 | WPA+WPA2
 4 | Renata                           |  -81 |  8 | WPA2
 5 | IMDC2022                         |  -81 |  8 | WPA2
 6 | VIRUS                            |  -82 |  1 | WPA+WPA2
 7 | ALAN                             |  -83 |  1 | WPA+WPA2
 8 | FRANCISCO                        |  -83 |  6 | WPA+WPA2
```

Algunas veces, cuando tenemos el Bluetooth activado en el computador, el puerto COM que se usa interfiere con la comunicación de la ESP32. Se soluciona verificando la instalación el controlador USB-UART y desactivando el Bluetooth del computador.

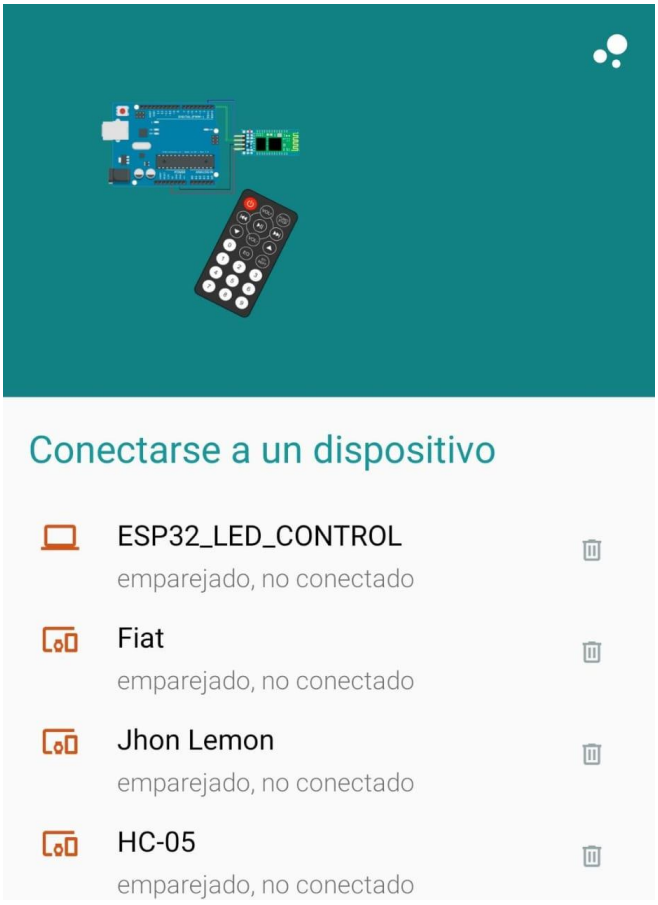
Manejo de sensores y actuadores básicos

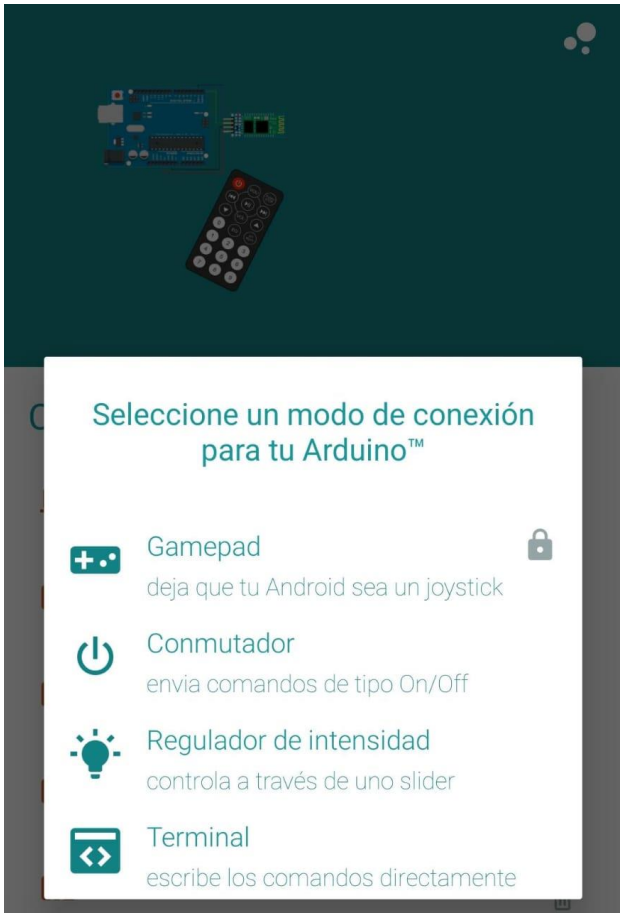
Lo que debemos hacer ahora es montajes electrónicos y códigos que permitan manejar sensores, actuadores, y las capacidades Bluetooth y WiFi del microcontrolador ESP32 a través de las herramientas que acabamos de instalar. Después de hacer este ejemplo, entenderás cómo podemos manejar otros tipos de sensores de una manera similar. Vamos a encender y apagar un LED usando Bluetooth modificando un código existente de manejo de Bluetooth (Carrasco, 2024).

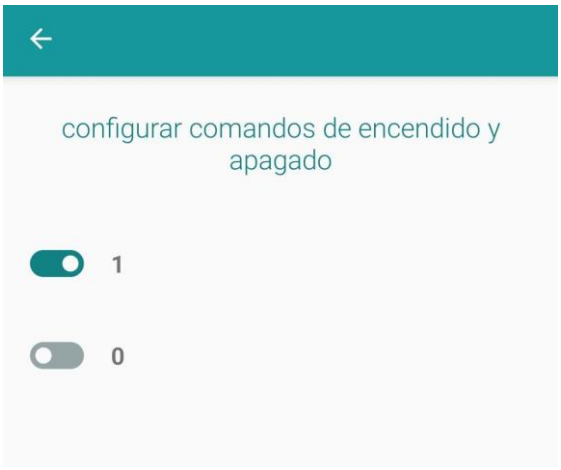
Empezamos por conectar al pin D23 de la ESP32 una resistencia, luego un LED en serie que después va conectado a la tierra (GND) de la tarjeta. Usaremos la librería BluetoothSerial de Arduino (Arduino, BluetoothSerial, 2024).Con este montaje básico procedemos a escribir el código de la siguiente manera:

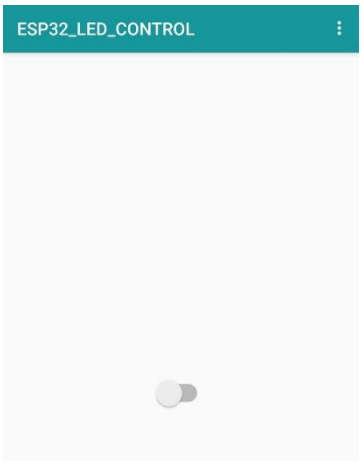
```
#include "BluetoothSerial.h"


int LED = 23;
BluetoothSerial BT;
void setup() {
  Serial.begin(9600);
  BT.begin("ESP32_LED_CONTROL");
  Serial.println("El BT esta listo para enparejarse");
  pinMode(LED, OUTPUT);
}

void loop() {
  if (BT.available()){
    int incoming = BT.read();
    Serial.print("Recibido: ");
    Serial.println(incoming);

    if (incoming == 49){
      digitalWrite(LED, HIGH);
      BT.println("LED encendido");
    }
    if (incoming == 48){
      digitalWrite(LED, LOW);
      BT.println("LED apagado");
    }
  }
  delay(20);
}
```

Luego descargamos en el celular o dispositivo móvil la aplicación de Bluetooth de Arduino, conectamos el dispositivo Bluetooth, seleccionamos ESP32_LED_CONTROL para conectarnos, elegimos la opción de Conmutador, lo configuramos para enviar 1 o 0, y estamos listos para probar que nuestro montaje funciona:









Apartado final

Acabamos de ver cómo se puede hacer la instalación de las herramientas necesarias para manejar la tarjeta de ESP32 usando el entorno de desarrollo de Arduino. La programación de entradas y salidas tanto digitales como analógicas se hace de manera similar a las de la tarjeta Arduino. Se declaran funciones WiFi o Bluetooth adicionalmente para un sinfín de aplicaciones.

Se pueden usar estos conocimientos y adaptarlos para la gran mayoría de aplicaciones que usen un microcontrolador ESP32.

Referencias

Arduino. (01 de 04 de 2024). *Arduino Uno Rev3*. Obtenido de Arduino: https://store.arduino.cc/products/arduino-uno-rev3?_gl=1*1sqm5nb*_gcl_au*MTcyNDIwOTk5NS4xNzExMTI0NjI2*FPAU*MTcyNDIwOTk5NS4xNzExMTI0NjI2*_ga*MzQ3MTA1NDk2LjE3MTExMjQ2MjQ.*_ga_NEXN8H46L5*MTcxODQwNjY0NC4xNC4xLjE3MTg0MDY3ODkuMC4wLjc3ODI3MTcyNw..*_fplc*QmNrUCUy

Arduino. (27 de 08 de 2024). *BluetoothSerial*. Obtenido de BluetoothSerial: https://www.arduino.cc/reference/en/libraries/bluetoothserial/

Carrasco, D. (27 de 08 de 2024). *ESP32: Empezando a usar el Bluetooth (SPP)*. Obtenido de ESP32: Empezando a usar el Bluetooth (SPP): https://www.electrosoftcloud.com/esp32-empezando-a-usar-el-bluetooth-spp/

Electronilab. (14 de 08 de 2024). *Electronilab*. Obtenido de Board de desarrollo con ESP32 WiFi Bluetooth BLE CH9102F: https://electronilab.co/tienda/board-de-desarrollo-con-modulo-esp32-wifibluetooth-ble/

Espressif. (14 de 08 de 2024). *esp32*. Obtenido de espressif: https://www.espressif.com/en/products/socs/esp32