LABORATORY VII – THE ESP32 DEV BOARD & WI-FI

1. Introduction

The purpose of this laboratory work is to introduce to you a new development board, based on the ESP32 microcontroller. This microcontroller supports the wireless connections Wi-Fi, Bluetooth, and Bluetooth Low Energy (BLE), making it suitable for IoT applications. The microcontroller has two 240 MHz cores, each one equipped with a Tensilica Xtensa LX6 32 bit microprocessor, and is able to work in industrial environments with temperatures ranging from -40 to 125 degrees Celsius. ESP32 has a low power consumption, between 160mA and 260mA.

Besides the active mode, the microcontroller also has four Low Power modes, including the Deep Sleep mode which requires less than 0.15 mA.

The ESP32 microcontroller is also equipped with the wired serial communication interfaces SPI, I2C and UART.

The development board must be powered by a voltage between 2.3 V and 3.6 V, and its pins will not accept a 5 V voltage. Usually, the power voltage is stabilized at 3.3 V.

The development board that we'll use in this lab is ESP32 Devkit V1, shown in Figure 1. The programmer can use the general purpose I/O pins for digital input and output (GPIO) or for reading analog signals (ADC), or for interfacing devices using the serial interfaces SPI (blue), I2C (pink) or UART (yellow).

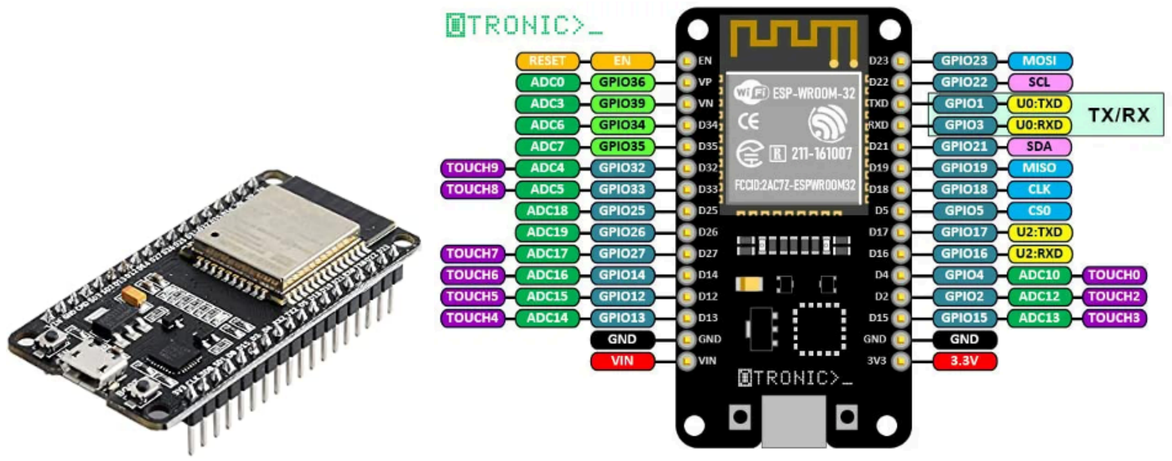


Figure 1. The ESP32 Devkit V1 development board and its pins.

2. The physical setup for the laboratory activities

For convenience and for preventing damage during the laboratory activities, the ESP32 boards are included into pre-built hardware setups, which include:

- The ESP-WROOM-32 V1 development board
- A SSD1306 OLED Display, connected to the ESP board via the I2C interface
- A YwRobot 545043 power supply, so that you can use external power supplies
- A 30 x (10 + 4) breadboard

The built setup is shown in Figure 2, and the schematic is shown in Figure 3.

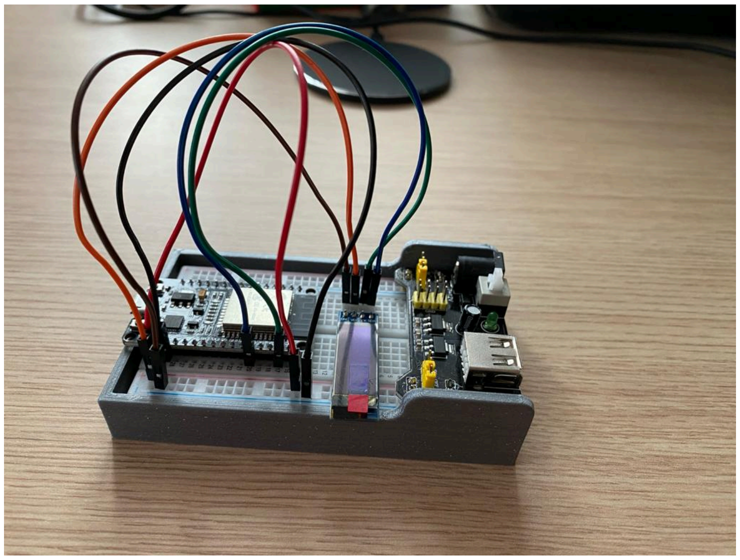


Figure 2. The pre-built hardware setup, based on ESP 32.

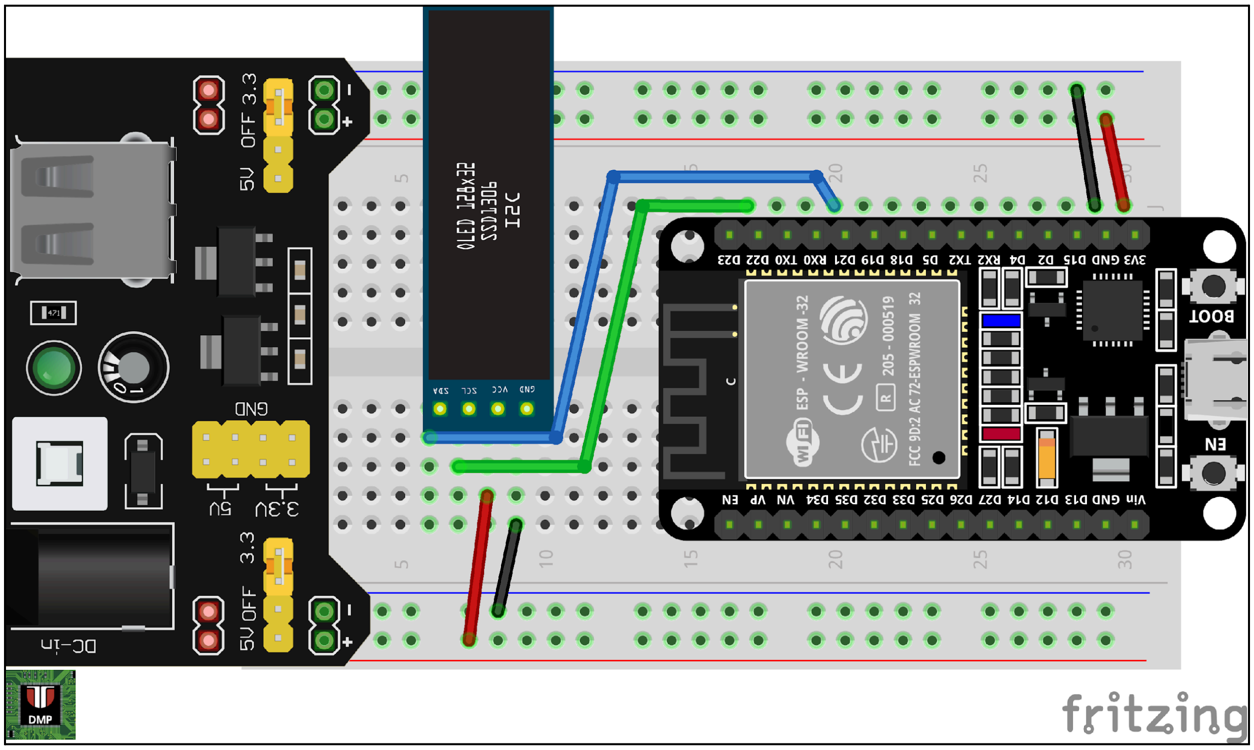

OLED 128x32
SSD1306
I2C
SDA SCL VCC GND
ESP - WROOM-32
BOOT
EN
3V3 GND D15 D2 D4 RX2 TX2 D5 D18 D19 D21 RX0 TX0 D22 D23
Vin GND D13 D12 D14 D27 D26 D25 D33 D32 D35 D34 VN VP EN
5V OFF 3.3
3.3V 5V
GND
DC-in
fritzing

Figure 3. The diagram of the pre-built hardware setup.

Display

The SSD1306 is the model of the controller for the Organic Light Emitting Diode (OLED) display panel on the module. It encompasses the signals necessary to drive the display, includes contrast control, an internal RAM and 256 levels adjustable brightness.

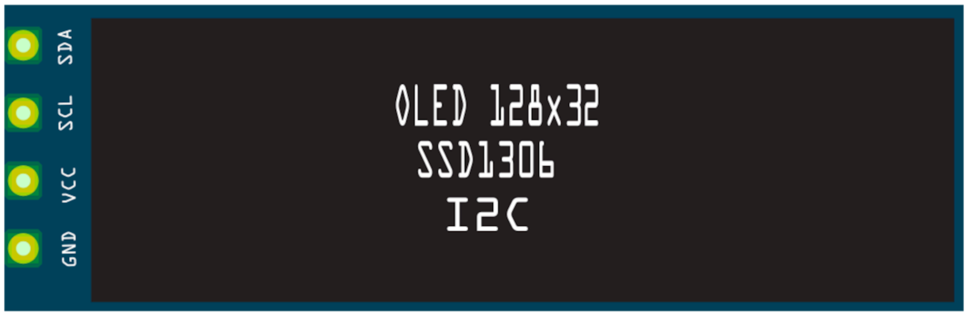


Figure 4. The OLED display.

The OLED display is a dot-matrix display that supports 128 columns along with 32 rows to display any manner of graphics, and has strong community support for creating fonts and for transforming images to the compatible format for the display.

The display communicates with the ESP32 development board via I^2C, and supports both 3.3 and 5 V for the V_{CC} power pin.

Power Supply

The YwRobot 545043 power supply is based on the MB V2 design and is meant to be a power supply specifically for breadboards.

It can be powered via a standard barrel (5.5mm outer (total) diameter x 2.1mm inner diameter) plug connector with 6.5 - 12 V (DC) and is capable of supplying either 3.3 V or 5 V (DC) with a maximum of 700 mA, for a maximum power output of 3.5 W on two different rails.

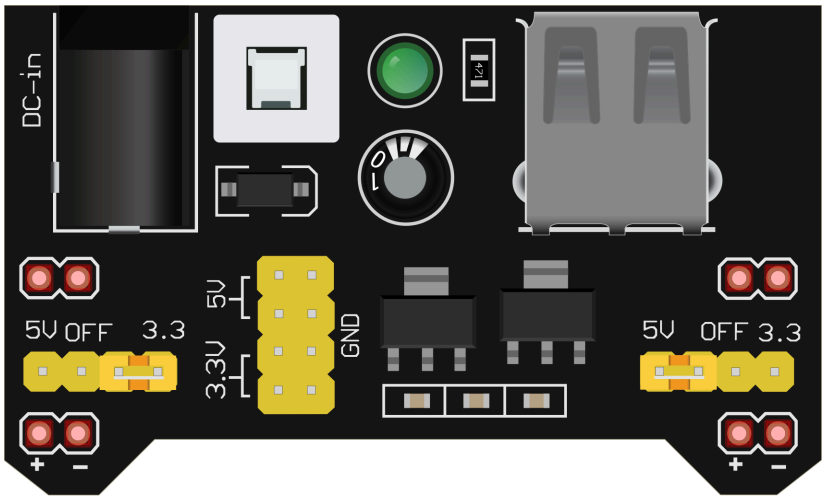


Figure 5. The breadboard power adapter.

Both rails can be configured independently with the help of jumpers (the jumpers select the voltage to be delivered to the rails, 3.3 V, 5 V, or OFF. **Please do not move the jumpers, as they are configured 3.3V, suitable for ESP32**). There is an additional USB type A port for powering other devices. **This port is output only, it cannot be used as power input.**

The power supply has a push button for turning it ON / OFF, and a LED status indicator that shows as to whether there's any current flowing through the system.

Breadboard

The breadboard used in the setup is often referred to as a *"half-size breadboard"*, since it "only" has thirty rows (1 – 30), along with fourteen (a – e + f – j + 2 x (+,-)) columns for connecting pins.

Contrary to the previously used breadboard here at the labs, this one has dedicated V_{CC} and GND columns on the two edges, where the pins are connected vertically in columns, instead of rows. The remaining pins in between are connected as usual, meaning horizontally, split into two parts.

The picture below highlights the individual connections, meaning the two vertically connected sets of pins – blue for the *minus (-)* and red for the *plus (+)* – and the horizontally connected ones, – green for columns a – e and the other half, yellow for columns f – j.

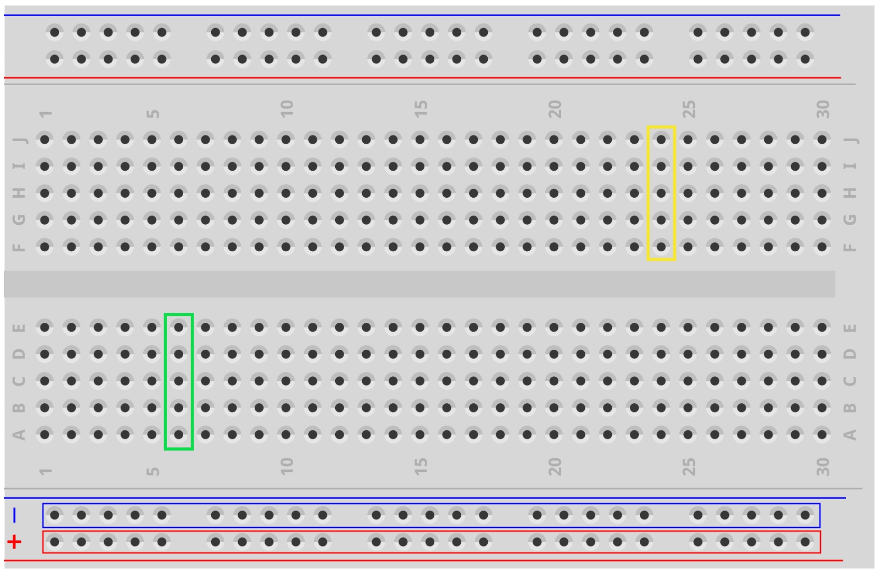


Figure 6. The breadboard with power rails.

Please note, that the split in the middle is not just meant to serve as a visual aid, but it denotes a split in the connections as well. E.g. in case a certain signal is connected by a wire to one of the E pins, then the rest of the pins in that row, meaning A – D, will have the same signal, but the ones on the other side (F - J) will not.

3. Setting up the Arduino environment for ESP32

Driver for Windows:
https://www.silabs.com/documents/public/software/CP210x_Windows_Drivers.zip

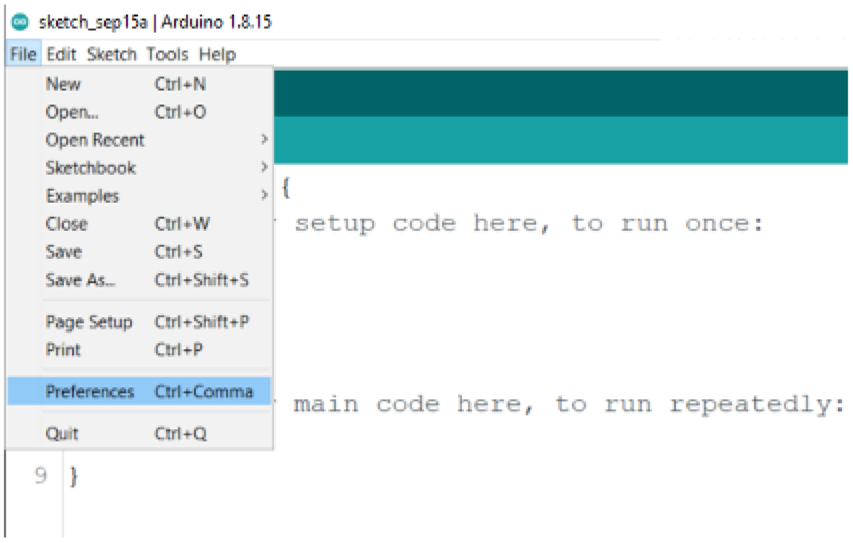
Arduino IDE v1.x

The Arduino IDE is able to work with the new development board, but it needs to be configured. For installing the ESP32 board family, open the **File / Preferences** menu, and open the **Additional Boards Manager URLs** list, and type the following link:

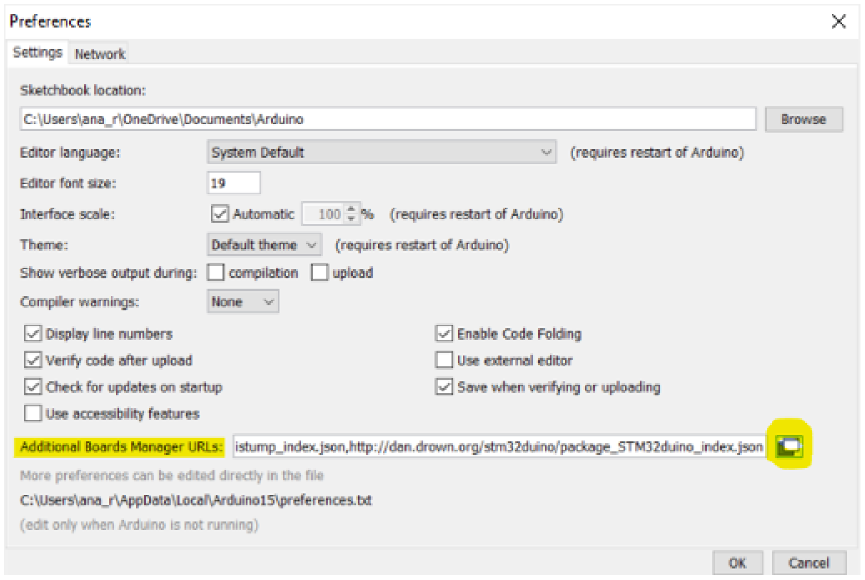
https://dl.espressif.com/dl/package_esp32_index.json

The next step is to open the **Tools / Board/Board Manager** menu, to search for the ESP32 board family by Espressif Systems, and to install the latest version of the software.

The installation steps are shown in Figure 7.



```
{
 setup code here, to run once:

 main code here, to run repeatedly:
}
```



Preferences

Settings | Network

Sketchbook location: C:\Users\ana_r\OneDrive\Documents\Arduino

Editor language: System Default (requires restart of Arduino)

Editor font size: 19

Interface scale: ☑ Automatic 100 % (requires restart of Arduino)

Theme: Default theme (requires restart of Arduino)

Show verbose output during: ☐ compilation ☐ upload

Compiler warnings: None

☑ Display line numbers
☑ Verify code after upload
☑ Check for updates on startup
☐ Use accessibility features
☑ Enable Code Folding
☐ Use external editor
☑ Save when verifying or uploading

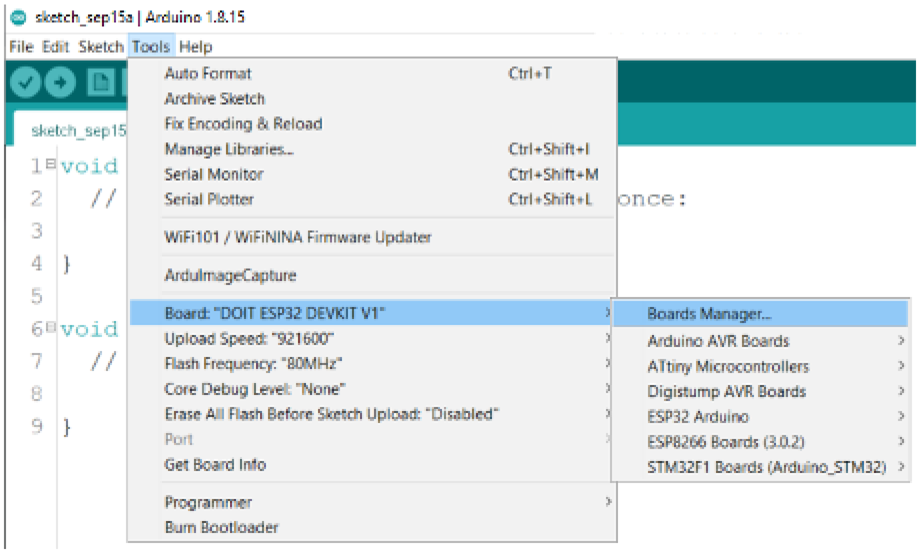
Additional Boards Manager URLs: istump_index.json,http://dan.drown.org/stm32duino/package_STM32duino_index.json

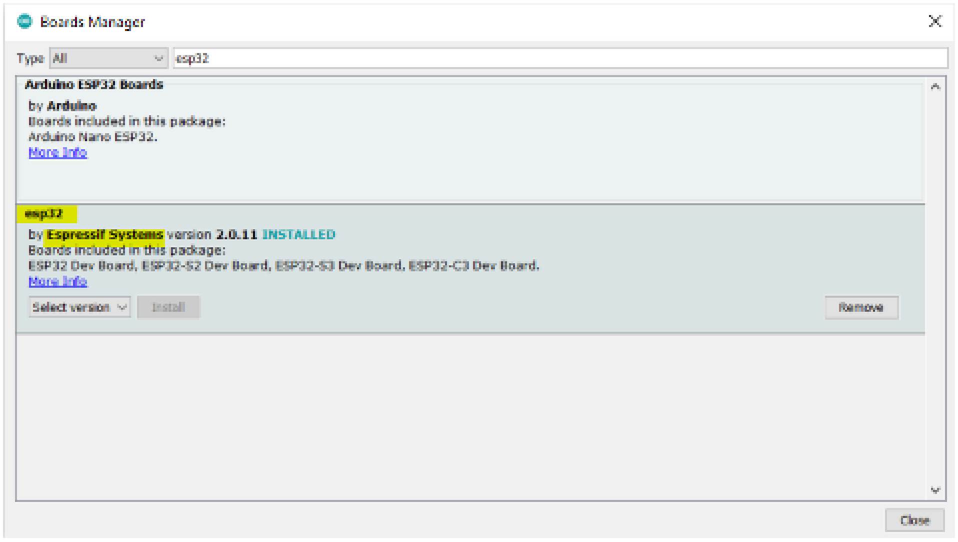
More preferences can be edited directly in the file

C:\Users\ana_r\AppData\Local\Arduino15\preferences.txt

(edit only when Arduino is not running)

OK | Cancel





Arduino ESP32 Boards
by **Arduino**
Boards included in this package:
Arduino Nano ESP32.
More Info

esp32
by **Espressif Systems** version **2.0.11** INSTALLED
Boards included in this package:
ESP32 Dev Board, ESP32-S2 Dev Board, ESP32-S3 Dev Board, ESP32-C3 Dev Board.
More Info

Figure 7. Setting the ESP32 boards with Arduino IDE v1.x.

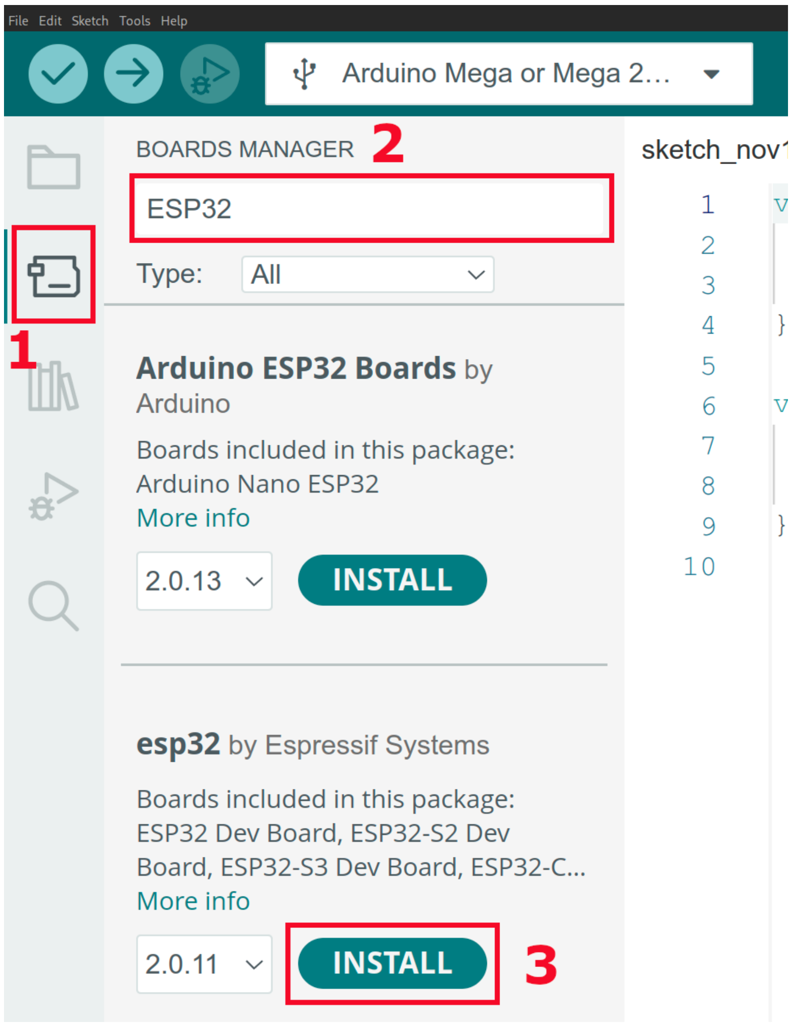
After the installation is complete, you have to restart the Arduino IDE, and a new family of boards will become available in the menu **Tools / Board / ESP 32 Arduino**. From the list, you must select the board **DOIT ESP32 DEVKIT V1**.

Arduino IDE v2.x

From the left vertical menu, click on the board manager icon, which will expand an additional panel for you. Afterwards, type "ESP32" into the search box, then install the "esp32 by Espressif Systems" library, as shown below in Figure 8.

Once the installation is complete, click on the board selector at the top, then on "select other board and port".

At the board selection dialog, enter the search term "**doit**", and select the **DOIT ESP32 DEVKIT V1.**



BOARDS MANAGER

ESP32

Type: All

Arduino ESP32 Boards by Arduino

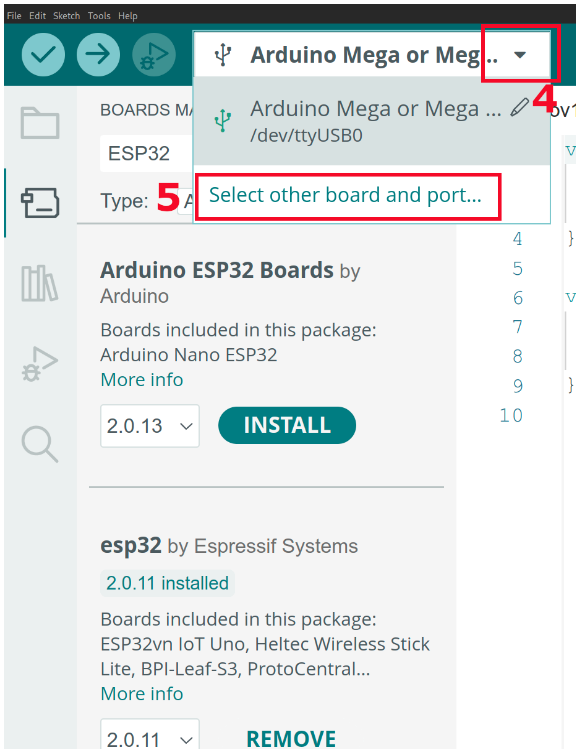
Boards included in this package:
Arduino Nano ESP32
More info

2.0.13 INSTALL

esp32 by Espressif Systems

Boards included in this package:
ESP32 Dev Board, ESP32-S2 Dev Board, ESP32-S3 Dev Board, ESP32-C...
More info

2.0.11 INSTALL


File Edit Sketch Tools Help
Arduino Mega or Meg...
4
Arduino Mega or Mega ...
/dev/ttyUSB0
5
Select other board and port...
BOARDS MA
ESP32
Type:
Arduino ESP32 Boards by Arduino
Boards included in this package:
Arduino Nano ESP32
More info
2.0.13
INSTALL
esp32 by Espressif Systems
2.0.11 installed
Boards included in this package:
ESP32vn IoT Uno, Heltec Wireless Stick Lite, BPI-Leaf-S3, ProtoCentral...
More info
2.0.11
REMOVE

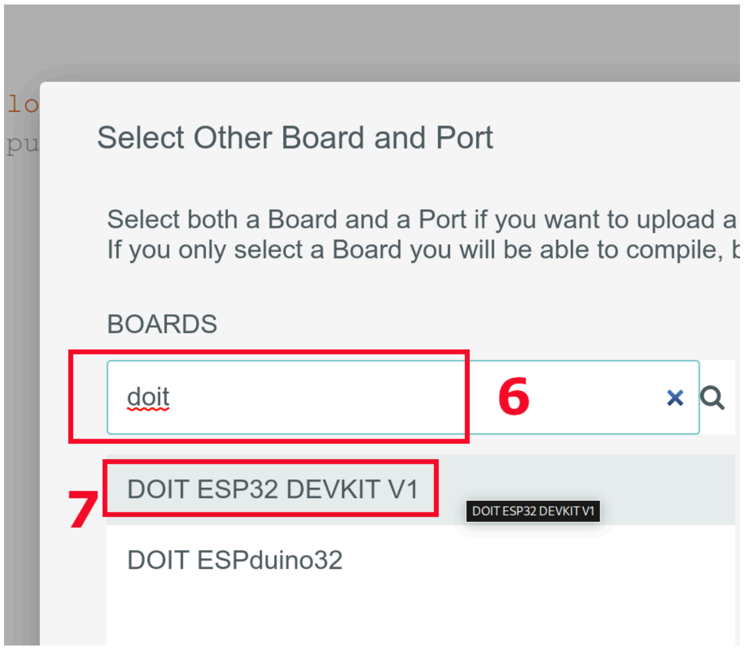

Select Other Board and Port
Select both a Board and a Port if you want to upload a
If you only select a Board you will be able to compile, b
BOARDS
doit
6
7
DOIT ESP32 DEVKIT V1
DOIT ESP32 DEVKIT V1
DOIT ESPduino32

Figure 8. Setting the ESP32 boards with Arduino IDE v2.x.

4. Programming the ESP32 board

Connect the board to the PC using the **Micro USB cable**. Select the proper serial port from the Tools menu of the Arduino IDE, and the proper board type (see previous chapter).

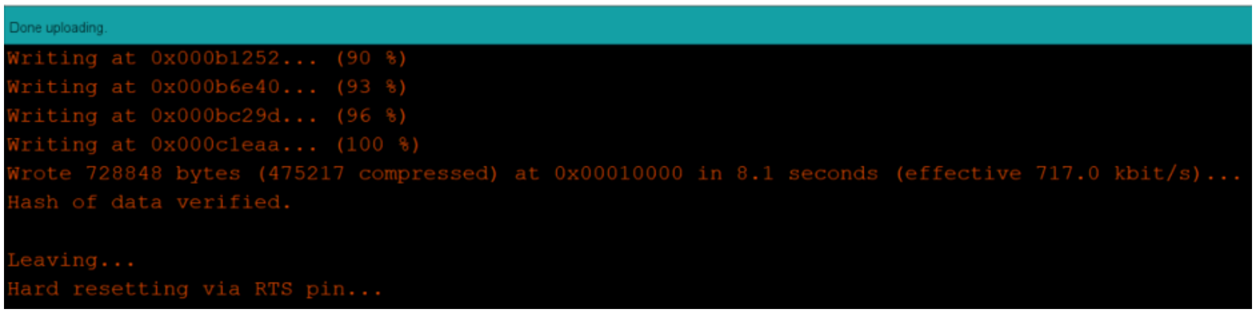
Type or paste the program in the Arduino IDE, and then use the same “Upload” command that you have used before to compile the program and write it to the board.

To test that the board and the upload mechanism is working, you can use a simple Blink program, which will turn on and off an on-board LED connected to digital pin 2.

```
void setup() {

  pinMode(2, OUTPUT);

}

void loop() {

  digitalWrite(2, 1);

  delay (500);

  digitalWrite(2,0);

  delay(500);

}
```

The upload process for the ESP32 boards takes more time than the similar process on the AVR Arduino boards. If the error **“Wrong boot mode detected (0x13)! The chip needs to be in download mode.”** appears during the upload process, you must press the BOOT button on the ESP32 board after the “Connecting…” message is displayed, and keep it pressed until the “Writing…” lines are displayed. After the “Writing… “ messages are shown, you can release the BOOT button, and wait until the process is completed.

A successful upload will have the console look like the one in Figure 9.



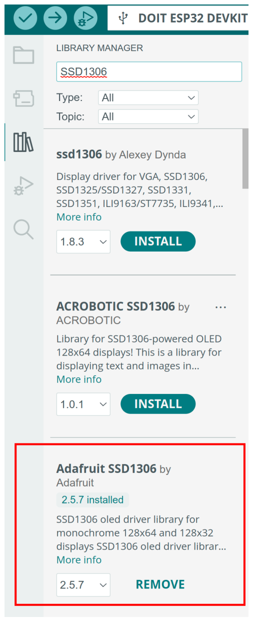
```
Done uploading.
Writing at 0x000b1252... (90 %)
Writing at 0x000b6e40... (93 %)
Writing at 0x000bc29d... (96 %)
Writing at 0x000c1eaa... (100 %)
Wrote 728848 bytes (475217 compressed) at 0x00010000 in 8.1 seconds (effective 717.0 kbit/s)...
Hash of data verified.

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

Figure 9. Console messages after a successful program upload to ESP32.

If your program uses **Serial** for input/output, and after you open the Serial Monitor no message appears, you may need to reset the board by pressing the Reset button.

5. Demo program for the OLED display

In order to use the SSD1306 OLED display, you will have to first install a corresponding library that supports it. For this, the Adafruit SSD1306 library is recommended, since it has good support for the controller.



DOIT ESP32 DEVKIT
LIBRARY MANAGER
SSD1306
Type: All
Topic: All
ssd1306 by Alexey Dynda
Display driver for VGA, SSD1306, SSD1325/SSD1327, SSD1331, SSD1351, ILI9163/ST7735, ILI9341,...
More info
1.8.3
INSTALL
ACROBOTIC SSD1306 by ACROBOTIC
Library for SSD1306-powered OLED 128x64 displays! This is a library for displaying text and images in...
More info
1.0.1
INSTALL
Adafruit SSD1306 by Adafruit
2.5.7 installed
SSD1306 oled driver library for monochrome 128x64 and 128x32 displays SSD1306 oled driver librar...
More info
2.5.7
REMOVE

Figure 10. Install the “Adafruit SSD1306” library from the IDE’s library manager.

The following demo code is also available online at
https://github.com/UTCN-AC-CS-DMP/Lab-7-ESP32-Part-1/blob/main/Lab_7_OLED_only.ino

```
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128  // OLED display width, in pixels
#define SCREEN_HEIGHT 32  // OLED display height, in pixels

// Declaration for an SSD1306 display connected to I2C (SDA, SCL pins)
// The pins for I2C are defined by the Wire-library.
// On Arduino UNO:       A4(SDA), A5(SCL)
// On Arduino MEGA:     20(SDA), 21(SCL)
// On ESP 32:      21 (SDA), 22 (SCL)
#define OLED_RESET -1
// Reset pin # (or -1 if sharing Arduino reset pin)
#define SCREEN_ADDRESS 0x3C
// The I2C address for OLED, 0x3D for 128x64, 0x3C for 128x32
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire,
                        OLED_RESET);


#define LOGO_WIDTH 128
#define LOGO_HEIGHT 32
// 'logo_utcn', 128x32px
static const unsigned char PROGMEM logo_bmp[] = {
 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xc0, 0x00, 0x3f,
 0x80, 0x00, 0x7f, 0x00, 0x00, 0xfe, 0x00, 0x01, 0xfc, 0x00, 0x03, 0xff,
 0xff, 0xfe, 0x07, 0xff, 0xfc, 0x0f, 0xff, 0xf8, 0x1f, 0xff, 0xf0, 0x3f,
 0xff, 0xe0, 0x7f, 0xff, 0xff, 0xc2, 0x04, 0x3f, 0x84, 0x08, 0x7f, 0x08,
 0x10, 0xfe, 0x10, 0x21, 0xfc, 0x20, 0x43, 0xff, 0xff, 0xc2, 0x04, 0x3f,
 0x84, 0x08, 0x7f, 0x08, 0x10, 0xfe, 0x10, 0x21, 0xfc, 0x20, 0x43, 0xff,
 0xff, 0xc2, 0x04, 0x3f, 0x84, 0x08, 0x7f, 0x08, 0x10, 0xfe, 0x10, 0x21,
 0xfc, 0x20, 0x43, 0xff, 0xff, 0xc2, 0x04, 0x3f, 0x84, 0x08, 0x7f, 0x08,
 0x10, 0xfe, 0x10, 0x21, 0xfc, 0x20, 0x43, 0xff, 0xff, 0xc2, 0x04, 0x3f,
 0x84, 0x08, 0x7f, 0x08, 0x10, 0xfe, 0x10, 0x21, 0xfc, 0x20, 0x43, 0xff,
 0xff, 0xe2, 0x04, 0x7f, 0xc4, 0x08, 0xff, 0x88, 0x11, 0xff, 0x10, 0x23,
 0xfe, 0x20, 0x47, 0xff, 0xff, 0xe2, 0x04, 0x7f, 0xc4, 0x08, 0xff, 0x88,
 0x11, 0xff, 0x10, 0x23, 0xfe, 0x20, 0x47, 0xff, 0xff, 0xf2, 0x04, 0xff,
 0xe4, 0x09, 0xff, 0xc8, 0x13, 0xff, 0x90, 0x27, 0xff, 0x20, 0x4f, 0xff,
 0xff, 0xfa, 0x05, 0xff, 0xf4, 0x0b, 0xff, 0xe8, 0x17, 0xff, 0xd0, 0x2f,
 0xff, 0xa0, 0x5f, 0xff, 0xff, 0xfe, 0x07, 0xff, 0xfc, 0x0f, 0xff, 0xf8,
 0x1f, 0xff, 0xf0, 0x3f, 0xff, 0xe0, 0x7f, 0xff, 0xff, 0xff, 0xff, 0xff,
 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xe1, 0xf1,
 0xff, 0xf8, 0x00, 0x7f, 0xff, 0xc0, 0xff, 0xff, 0x0f, 0x0f, 0xff, 0xff,
 0xff, 0xff, 0xe1, 0xf1, 0xff, 0xf8, 0x00, 0x7f, 0xff, 0x00, 0x7f, 0xff,
 0x07, 0x0f, 0xff, 0xff, 0xff, 0xff, 0xe1, 0xf1, 0xff, 0xff, 0x87, 0xff,
 0xfe, 0x1e, 0x7f, 0xff, 0x07, 0x0f, 0xff, 0xff, 0xff, 0xff, 0xe1, 0xf1,
 0xff, 0xff, 0x87, 0xff, 0xfe, 0x3f, 0x7f, 0xff, 0x03, 0x0f, 0xff, 0xff,
 0xff, 0xff, 0xe1, 0xf1, 0xff, 0xff, 0x87, 0xff, 0xfc, 0x3f, 0xff, 0xff,
 0x01, 0x0f, 0xff, 0xff, 0xff, 0xff, 0xe1, 0xf1, 0xff, 0xff, 0x87, 0xff,
 0xfc, 0x7f, 0xff, 0xff, 0x11, 0x0f, 0xff, 0xff, 0xff, 0xff, 0xe1, 0xf1,
 0xff, 0xff, 0x87, 0xff, 0xfc, 0x7f, 0xff, 0xff, 0x10, 0x0f, 0xff, 0xff,
 0xff, 0xff, 0xe1, 0xf1, 0xff, 0xff, 0x87, 0xff, 0xfc, 0x7f, 0xff, 0xff,
 0x18, 0x0f, 0xff, 0xff, 0xff, 0xff, 0xf1, 0xf1, 0xff, 0xff, 0x87, 0xff,
 0xfc, 0x3f, 0xff, 0xff, 0x1c, 0x0f, 0xff, 0xff, 0xff, 0xff, 0xf1, 0xf1,
 0xff, 0xff, 0x87, 0xff, 0xfc, 0x3f, 0x7f, 0xff, 0x1c, 0x0f, 0xff, 0xff,
 0xff, 0xff, 0xf0, 0xe1, 0xff, 0xff, 0x87, 0xff, 0xfe, 0x1e, 0x7f, 0xff,
 0x1e, 0x0f, 0xff, 0xff, 0xff, 0xff, 0xf8, 0x03, 0xff, 0xff, 0x87, 0xff,
 0xff, 0x00, 0x7f, 0xff, 0x1e, 0x0f, 0xff, 0xff, 0xff, 0xff, 0xfc, 0x0f,
 0xff, 0xff, 0x87, 0xff, 0xff, 0xc0, 0xff, 0xff, 0x1f, 0x0f, 0xff, 0xff,
 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff
};


void setup() {
 Serial.begin(9600);
 // SSD1306_SWITCHCAPVCC = generate display voltage from 3.3V internally
 if (!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
   Serial.println(F("SSD1306 allocation failed"));
   for (;;)
     ;  // Don't proceed, loop forever
 }


 // Clear the buffer
 display.clearDisplay();


 // Draw demo logo
 testDrawUTCNLogo();


 // Write sample string to different rows
 writeTextSSD1306("First line", 0, 1, true);
 writeTextSSD1306("Another string below", 8, 1, false);
 // 3 second delay
 delay(3000);
 // Larger font size with automatic line break
 writeTextSSD1306("Auto line break", 0, 2, true);

 delay(3000);

 display.setCursor(0, 0);
 display.clearDisplay();

 display.write("Waiting");
}

int c = 0;

void loop() {

 display.write(".");
 display.display();

 delay(1000);

 c++;

 if (c > 13) {
   display.clearDisplay();
   display.setCursor(0, 0);
   c = 0;
 }
}


void writeTextSSD1306(const String &message, const uint8_t rowNum, const uint8_t textSize,
const bool clearDisp)
{
 if (clearDisp) {
   display.clearDisplay();
 }

 display.setTextSize(textSize);
 display.setTextColor(SSD1306_WHITE);  // Draw white text
 // Specify location to display message
 display.setCursor(0, rowNum);
 // Use full 256 char 'Code Page 437' font
 display.cp437(true);
 display.write(message.c_str());
 display.display();
}

void testDrawUTCNLogo(void) {
 display.clearDisplay();


 display.drawBitmap(0, 0, logo_bmp, LOGO_WIDTH, LOGO_HEIGHT, WHITE);
 display.display();
 delay(3000);
}
```

Highlights: the program is centered around the **display** object, of the **Adafruit_SSD1306** class. The constructor of the class takes as parameters the width and height of the display and the pointer to the Wire object used for I2C communication. The initialization of the display object receives as parameter the I2C address of the display, which is currently 0x3C.

The display object can be cleared by the method **clearDisplay**, can set the cursor for text output using **setCursor**, can configure the text size by **setTextSize**, and can write a message using **write**. A graphical image can be written using the method **drawBitmap**.

6. Demo program for WiFi (ESP32 as access point)

The code is also available online at
https://github.com/UTCN-AC-CS-DMP/Lab-7-ESP32-Part-1/blob/main/Lab_7_WiFi_only.ino

```
#include <WiFi.h>
#include <WiFiAP.h>
```

```
#include <WiFiClient.h>

// MESSAGE STRINGS
const String SETUP_INIT = "SETUP: Initializing ESP32 dev board";
const String SETUP_ERROR = "!!ERROR!! SETUP: Unable to start SoftAP mode";
const String SETUP_SERVER_START = "SETUP: HTTP server started --> IP addr: ";
const String SETUP_SERVER_PORT = " on port: ";
const String INFO_NEW_CLIENT = "New client connected";
const String INFO_DISCONNECT_CLIENT = "Client disconnected";

// HTTP headers always start with a response code (e.g. HTTP/1.1 200 OK)
// and a content-type so the client knows what's coming, then a blank line:
const String HTTP_HEADER = "HTTP/1.1 200 OK\r\nContent-type:text/html\r\n\r\n";
const String HTML_WELCOME = "<h1>Welcome to your ESP32 Web Server!</h1>";

// BASIC WIFI CONFIGURATION CONSTANTS
// The SSID (Service Set IDentifier), in other words, the network's name
const char *SSID = "<your_unique_SSID>";
// Password for the network
// By default the ESP32 uses WPA / WPA2-Personal security, therefore the
// the password MUST be between 8 and 63 ASCII characters
const char *PASS = "<your_password_here>";
// The default port (both TCP & UDP) of a WWW HTTP server number according to
// RFC1340 is 80
const int HTTP_PORT_NO = 80;

// ADDITIONAL GLOBALS
// Initialize the HTTP server on the ESP32 board
WiFiServer HttpServer(HTTP_PORT_NO);

void setup() {
 Serial.begin(9600);

 if (!WiFi.softAP(SSID)) {
       // replace with if (!WiFi.softAP(SSID, PASS)) to use the password
   Serial.println(SETUP_ERROR);
   // Lock system in infinite loop in order to prevent further execution
   while (1)
     ;
 }

 // Get AP's IP address for info message
 const IPAddress accessPointIP = WiFi.softAPIP();
 const String webServerInfoMessage = SETUP_SERVER_START + accessPointIP.toString()
                                     + SETUP_SERVER_PORT + HTTP_PORT_NO;

 // Start the HTTP server
 HttpServer.begin();
 Serial.println(webServerInfoMessage);
}

void loop() {
 WiFiClient client = HttpServer.available();  // listen for incoming clients

 if (client) {                             // if you get a client,
   Serial.println(INFO_NEW_CLIENT);        // print a message out the serial port
   String currentLine = "";               // make a String to hold incoming data from the client
   while (client.connected()) {            // loop while the client's connected
     if (client.available()) {             // if there's bytes to read from the client,
       const char c = client.read();       // read a byte, then
       Serial.write(c);                    // print it out the serial monitor
```

```
        if (c == '\n') {  // if the byte is a newline character
          // if the current line is blank, you got two newline characters in a row.
          // that's the end of the client HTTP request, so send a response:
          if (currentLine.length() == 0) {
            // Send welcome page to the client
            printWelcomePage(client);
            break;
          } else currentLine = "";
        } else if (c != '\r') {  // if you got anything else but a carriage return character,
          currentLine += c;      // add it to the end of the currentLine
        }
      }
    }
    // close the connection:
    client.stop();
    Serial.println(INFO_DISCONNECT_CLIENT);
    Serial.println();
  }
}

void printWelcomePage(WiFiClient client) {
  // Always start the response to the client with the proper headers
  client.println(HTTP_HEADER);

  // Send the relevant HTML
  client.print(HTML_WELCOME);

  // The HTTP response ends with another blank line
  client.println();
}
```

Code highlights and warnings: any WiFi access point has a unique identifier, its **SSID**, and can be secured with a password. Please ensure that you will set a SSID that is unique to you, and easily recognizable, as duplicate SSIDs will cause errors in the WiFi network. Therefore, please change the following line of code to ensure a unique SSID:

```
const char *SSID = "<your_unique_SSID>";
```

The connections are handled by the object **HttpServer**, of the class **WiFiServer**. The server listens to the specified **port** (port 80 for the HTTP protocol) and returns a **client** object, of the class **WiFiClient**, when a device such as a laptop or a mobile phone connects to it (by opening a browser and typing the address of the server). The client object can read data from the connected device, using methods similar to the Serial interface (**available**, **read**), and can also send data using similar methods (**write**, **print**, **println**). The communication between the ESP32 board and the client device is a simple text communication. The programmer must format the output text to form a valid html page, and must also parse the input text to detect the requests from the client device.

In order to test the program, you have to upload it to the ESP32 board, and then you have to set up your laptop or mobile phone to connect to the WiFi access point that the ESP32 board is acting as. You will identify the board by your unique SSID. After the connection is set up, you will open the browser and type the IP address of the board, which will be 192.168.4.1. The server on the ESP32 board should reply with a simple welcome message, as shown in Figure 11:

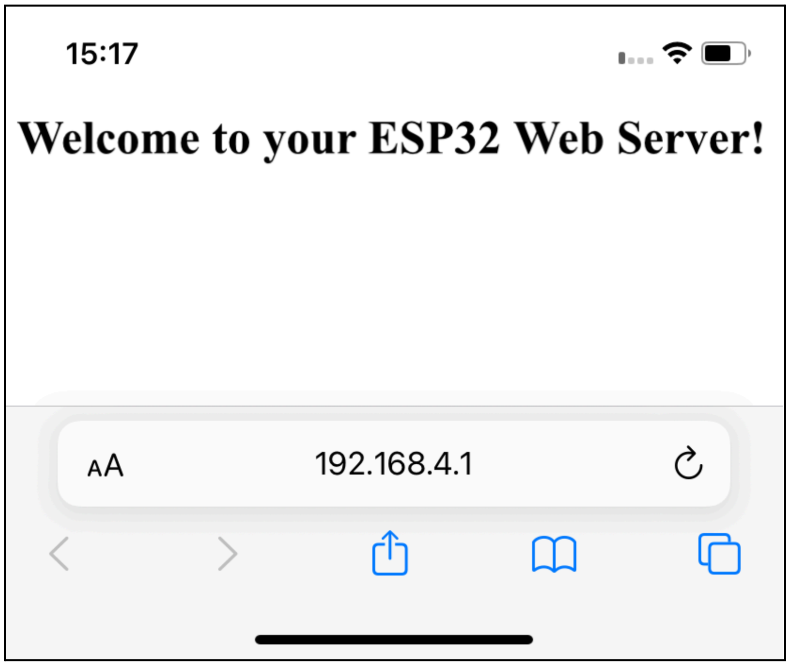


Figure 11. The resulting webpage from the ESP32's built-in webserver.

NOTE: For Apple iPhones using the Safari browser, you need to specify the protocol as well, meaning it's not enough to write only the IP address, since it will attempt to use HTTPS by default. Therefore, you will have to write http://192.168.4.1 in Safari's address bar instead of just 192.168.4.1

7. Individual work

1. Run the examples, analyze the code and the explanations.
2. Display the status of the connection and the GET request strings received by the WiFi web server using the OLED display.
3. Based on the GET request, turn on or off the board's built in LED (*hint: you can add text after the IP address, e.g. 192.168.4.1/LedOn*)
4. Modify the web server so that the html page sent to the user will contain links for turning the LED on or off.

Bibliography

1. Description of the ESP32 microcontroller https://www.espressif.com/en/products/socs/esp32
2. The ESP32 WiFi API https://docs.espressif.com/projects/arduino-esp32/en/latest/api/wifi.html
3. SSD1306 Datasheet https://cdn-shop.adafruit.com/datasheets/SSD1306.pdf
4. YwRobot Documentation https://static.rapidonline.com/pdf/73-4538_v1.pdf
5. Breadboard Info https://www.adafruit.com/product/64