<https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32e_esp32-wroom-32ue_datasheet_en.pdf>

For programming the ESP32-WROOM-32E/32UE, **you don’t need** an SPI or JTAG connection for standard firmware uploads. The module can be programmed directly using UART over USB with two main connections: **TX/RX pins** and **GPIO0** to initiate boot mode for programming. This allows you to flash firmware easily through tools like the **ESP-IDF** or **Arduino IDE** using a USB-to-UART adapter, so additional programming interfaces like JTAG are not necessary unless you need advanced debugging.

Yes, you can program the ESP32-WROOM-32E/32UE without mounting it on a PCB by using a **breadboard** or a similar setup. You would need to connect the necessary pins (TX, RX, GPIO0) to a USB-to-UART adapter. This way, you can upload code directly to the module without soldering it to a permanent PCB. Just ensure the connections are secure and that you manage the GPIO0 pin for booting into programming mode.

The **GPIO0 pin** on the ESP32 is crucial for entering programming mode. To manage it, follow these steps:

1. **Connecting GPIO0**: When powering up the ESP32, connect GPIO0 to ground (GND) to initiate the bootloader mode, allowing firmware uploads.
2. **Uploading Code**: After connecting GPIO0 to GND, reset the module (by cycling power or using a reset button), then start the programming tool.
3. **Releasing GPIO0**: Once programming is complete, disconnect GPIO0 from GND to allow the ESP32 to run the uploaded firmware normally.

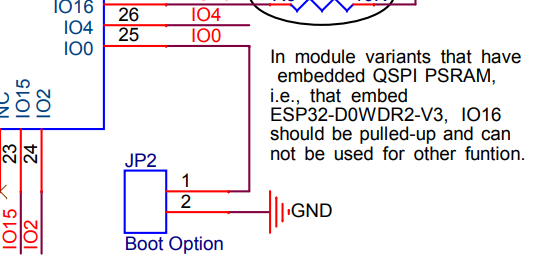
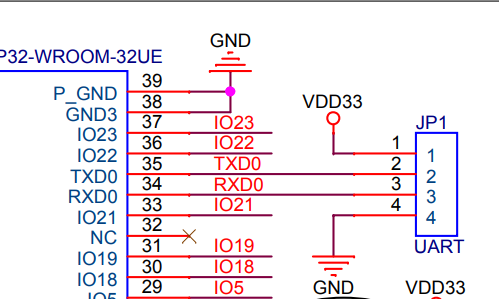
Make sure to check the specific datasheet for pin assignments and electrical characteristics.

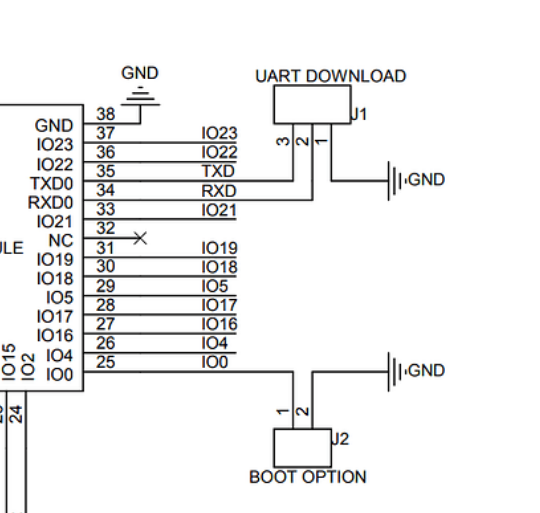
### **Steps to Program the ESP32-WROOM-32E Externally Using CP2102:**

1. **Connect the CP2102 to the ESP32-WROOM-32E:**
   * **CP2102 (USB to UART) Pin Connections:**
     + **TX (Transmitter)** on CP2102 to **RX (Receiver)** on the ESP32-WROOM-32E.
     + **RX (Receiver)** on CP2102 to **TX (Transmitter)** on the ESP32-WROOM-32E.
     + **GND** on CP2102 to **GND** on ESP32-WROOM-32E.
     + **VCC (3.3V)** on CP2102 to **3.3V** pin on the ESP32-WROOM-32E. The ESP32 cannot be powered by 5V directly; make sure to supply 3.3V to avoid damaging the chip.
2. **Reset/Boot Mode Configuration:**
   * The ESP32-WROOM-32E requires that you enter boot mode to upload the code.
   * **Connect GPIO0 to GND**: This puts the ESP32 in flashing mode (bootloader mode) and allows you to upload code via the CP2102.
   * **EN (Enable/RST) Pin**: You may need to briefly pull the EN (Enable) pin low (connect it to GND for a moment) to reset the ESP32 and initiate the programming process.
3. **Driver Installation for CP2102:**
   * If you haven’t installed the drivers for the CP2102, download and install them from Silicon Labs’ official website (especially for Windows).
4. **Upload Code via Arduino IDE or PlatformIO:**
   * Open your preferred IDE (e.g., Arduino IDE or PlatformIO).
   * Select **ESP32 Dev Module** as the board.
   * Select the **CP2102 COM port** under Tools > Port.
   * Ensure the correct baud rate (typically 115200) is selected.
   * Write or open the program you want to upload to the ESP32.
   * Put the ESP32 in boot mode by holding the **GPIO0 to GND**, and click **Upload** in the IDE.
   * After uploading the code, release GPIO0 from GND to exit boot mode.
5. **Test the Program:**
   * Once the code is uploaded successfully, you can test the functionality while the ESP32 is still connected to the CP2102.
6. **Soldering to PCB:**
   * After verifying the code works, you can proceed to solder the ESP32-WROOM-32E onto the PCB with the programmed code.

Make sure to consult the ESP32 datasheet for specific pin layouts and additional configurations.

HOW TO CONNECT THE UART:





<https://docs.espressif.com/projects/esp-idf/en/stable/esp32/get-started/establish-serial-connection.html>

1 ohm

10 ohm

100 ohm

1. Setup the ESP32 as a WiFi Access Point (AP) or Client

Option A: ESP32 as a WiFi Access Point (AP)

Configuration: Set up the ESP32 to act as a WiFi access point. This allows your PC to connect directly to the ESP32 without needing an external router.

Code Example:

cpp

Copy code

#include <WiFi.h>

const char\* ssid = "ESP32\_BattleBot";

const char\* password = "123456789";

void setup() {

Serial.begin(115200);

WiFi.softAP(ssid, password);

Serial.println("Access Point Started");

Serial.print("IP Address: ");

Serial.println(WiFi.softAPIP());

}

void loop() {

// Your code here

}

PC Connection: Connect your PC to the WiFi network named ESP32\_BattleBot with the password 123456789.

—----------------------------------------------------------------------------------------------------------------------------#include <WiFi.h>

#include <WebServer.h>

// WiFi credentials

const char\* ssid = "ESP32\_BattleBot";

const char\* password = "123456789";

// Motor driver pins

const int motor1Pin1 = 27; // IN1 on L298N for Motor 1

const int motor1Pin2 = 26; // IN2 on L298N for Motor 1

const int motor2Pin1 = 25; // IN3 on L298N for Motor 2

const int motor2Pin2 = 33; // IN4 on L298N for Motor 2

// Create web server on port 80

WebServer server(80);

void setup() {

Serial.begin(115200);

// Set motor pins as outputs

pinMode(motor1Pin1, OUTPUT);

pinMode(motor1Pin2, OUTPUT);

pinMode(motor2Pin1, OUTPUT);

pinMode(motor2Pin2, OUTPUT);

// Initialize WiFi in Access Point mode

WiFi.softAP(ssid, password);

Serial.println("Access Point Started");

Serial.print("IP Address: ");

Serial.println(WiFi.softAPIP());

// Define routes for the web server

server.on("/", handleRoot);

server.on("/forward", moveForward);

server.on("/backward", moveBackward);

server.on("/stop", stopMotors);

// Start the server

server.begin();

Serial.println("Server started");

}

void loop() {

// Handle client requests

server.handleClient();

}

// Web server functions

void handleRoot() {

server.send(200, "text/html", "<h1>ESP32 BattleBot Controller</h1><p>Use /forward, /backward, and /stop to control the bot.</p>");

}

// Motor control functions

void moveForward() {

digitalWrite(motor1Pin1, HIGH);

digitalWrite(motor1Pin2, LOW);

digitalWrite(motor2Pin1, HIGH);

digitalWrite(motor2Pin2, LOW);

server.send(200, "text/plain", "Moving Forward");

}

void moveBackward() {

digitalWrite(motor1Pin1, LOW);

digitalWrite(motor1Pin2, HIGH);

digitalWrite(motor2Pin1, LOW);

digitalWrite(motor2Pin2, HIGH);

server.send(200, "text/plain", "Moving Backward");

}

void stopMotors() {

digitalWrite(motor1Pin1, LOW);

digitalWrite(motor1Pin2, LOW);

digitalWrite(motor2Pin1, LOW);

digitalWrite(motor2Pin2, LOW);

server.send(200, "text/plain", "Stopping Motors");

}

### **Explanation of the Code**

1. **WiFi Setup**: The ESP32 is set up as an access point with the SSID ESP32\_BattleBot.
2. **Motor Driver Pins**: Pins motor1Pin1, motor1Pin2, motor2Pin1, and motor2Pin2 control the L298N motor driver channels for forward and backward movement.
3. **Web Server**: A simple web server runs on the ESP32, listening for /forward, /backward, and /stop commands.
4. **Motor Control Functions**:
   * moveForward(): Sets the motor pins to move both motors forward.
   * moveBackward(): Sets the motor pins to move both motors backward.
   * stopMotors(): Stops all motors.

### **How to Control the Bot**

1. **Connect to the ESP32 WiFi network** (ESP32\_BattleBot).
2. **Send commands** by opening a browser and navigating to:
   * http://<ESP32\_IP>/forward to move forward.
   * http://<ESP32\_IP>/backward to move backward.
   * http://<ESP32\_IP>/stop to stop the motors.

—----------------------------------------------------------------------------------------------------------------------------<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>ESP32 BattleBot Controller</title>

</head>

<body>

<h1>BattleBot Controller</h1>

<p>Use the arrow keys to control the bot:</p>

<ul>

<li>Up Arrow: Move Forward</li>

<li>Down Arrow: Move Backward</li>

<li>Spacebar: Stop</li>

</ul>

<script>

// Define the ESP32 server IP address (replace this with your ESP32's actual IP)

const esp32\_ip = "http://192.168.4.1"; // Use the IP shown in Serial Monitor

// Function to send a command to the ESP32

function sendCommand(command) {

fetch(`${esp32\_ip}/${command}`)

.then(response => {

if (response.ok) {

console.log(`${command} command sent`);

} else {

console.error(`Failed to send ${command} command`);

}

})

.catch(error => console.error("Error:", error));

}

// Listen for key presses to control the bot

document.addEventListener("keydown", (event) => {

switch (event.key) {

case "ArrowUp":

sendCommand("forward");

break;

case "ArrowDown":

sendCommand("backward");

break;

case " ":

sendCommand("stop");

break;

default:

break;

}

});

</script>

</body>

</html>

### **Explanation of the Code**

1. **ESP32 IP Address**: Replace http://192.168.4.1 with the IP address assigned to the ESP32 Access Point. This IP address appears in the Serial Monitor when the ESP32 Access Point starts.
2. **JavaScript fetch Function**: The sendCommand function uses fetch() to make an HTTP request to the ESP32’s server based on the command (e.g., forward, backward, stop).
3. **Event Listener for Keydown Events**:
   * **Up Arrow**: Calls /forward to move the bot forward.
   * **Down Arrow**: Calls /backward to move the bot backward.
   * **Spacebar**: Calls /stop to stop the bot.

### **Step 2: Connect and Test**

1. **Connect to the ESP32 WiFi Network**: On your PC, connect to the network hosted by the ESP32, such as "ESP32\_BattleBot."
2. **Open the HTML File**: Open this HTML file in a web browser while connected to the ESP32 network.
3. **Control the Bot**: Use the arrow keys to control the bot—up for forward, down for backward, and space to stop.

### **Here’s a breakdown of where each part goes:**

1. **Arduino IDE** (for ESP32 code):
   * Use the Arduino IDE for writing and uploading code to the ESP32. This code sets up the WiFi Access Point, starts the web server, and defines routes like /forward, /backward, and /stop.
   * The ESP32 code listens for HTTP requests and triggers motor movements based on the received commands.
2. **HTML + JavaScript (Client-Side Code)**:
   * Save the HTML and JavaScript code as an .html file (e.g., controller.html).
   * Open this file in a web browser on any device connected to the ESP32 WiFi network (like a laptop or phone).
   * This HTML page acts as a control interface by listening for key presses and sending HTTP requests to the ESP32.

### **Steps to Set Up and Use:**

1. **Upload the ESP32 code** to the microcontroller using the Arduino IDE.
2. **Save the HTML file** (e.g., controller.html) and open it in a web browser.
3. **Connect to the ESP32 network** from your PC or device.
4. **Open the HTML file in your browser** and use the arrow keys to send commands to the ESP32.

This setup separates the control interface (HTML + JavaScript on your PC) from the ESP32 code (Arduino IDE), allowing them to communicate over WiFi.

—----------------------------------------------------------------------------------------------------------------------------

**UPDATED CODE WITH RIGHTS AND LEFT TURNS**

#include <WiFi.h>

#include <WebServer.h>

// WiFi credentials

const char\* ssid = "ESP32\_BattleBot";

const char\* password = "123456789";

// Motor driver pins

const int motor1Pin1 = 27; // IN1 on L298N for Motor 1

const int motor1Pin2 = 26; // IN2 on L298N for Motor 1

const int motor2Pin1 = 25; // IN3 on L298N for Motor 2

const int motor2Pin2 = 33; // IN4 on L298N for Motor 2

// Create a web server on port 80

WebServer server(80);

void setup() {

Serial.begin(115200);

// Set motor pins as outputs

pinMode(motor1Pin1, OUTPUT);

pinMode(motor1Pin2, OUTPUT);

pinMode(motor2Pin1, OUTPUT);

pinMode(motor2Pin2, OUTPUT);

// Initialize WiFi in Access Point mode

WiFi.softAP(ssid, password);

Serial.println("Access Point Started");

Serial.print("IP Address: ");

Serial.println(WiFi.softAPIP());

// Define routes for the web server

server.on("/", handleRoot);

server.on("/forward", moveForward);

server.on("/backward", moveBackward);

server.on("/stop", stopMotors);

server.on("/left", turnLeft);

server.on("/right", turnRight);

// Start the server

server.begin();

Serial.println("Server started");

}

void loop() {

// Handle client requests

server.handleClient();

}

// Web server functions

void handleRoot() {

server.send(200, "text/html", "<h1>ESP32 BattleBot Controller</h1><p>Use /forward, /backward, /left, /right, and /stop to control the bot.</p>");

}

// Motor control functions

void moveForward() {

digitalWrite(motor1Pin1, HIGH);

digitalWrite(motor1Pin2, LOW);

digitalWrite(motor2Pin1, HIGH);

digitalWrite(motor2Pin2, LOW);

server.send(200, "text/plain", "Moving Forward");

}

void moveBackward() {

digitalWrite(motor1Pin1, LOW);

digitalWrite(motor1Pin2, HIGH);

digitalWrite(motor2Pin1, LOW);

digitalWrite(motor2Pin2, HIGH);

server.send(200, "text/plain", "Moving Backward");

}

void turnLeft() {

digitalWrite(motor1Pin1, LOW);

digitalWrite(motor1Pin2, HIGH);

digitalWrite(motor2Pin1, HIGH);

digitalWrite(motor2Pin2, LOW);

server.send(200, "text/plain", "Turning Left");

}

void turnRight() {

digitalWrite(motor1Pin1, HIGH);

digitalWrite(motor1Pin2, LOW);

digitalWrite(motor2Pin1, LOW);

digitalWrite(motor2Pin2, HIGH);

server.send(200, "text/plain", "Turning Right");

}

void stopMotors() {

digitalWrite(motor1Pin1, LOW);

digitalWrite(motor1Pin2, LOW);

digitalWrite(motor2Pin1, LOW);

digitalWrite(motor2Pin2, LOW);

server.send(200, "text/plain", "Stopping Motors");

}

—-----------------------------------

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>ESP32 BattleBot Controller</title>

<style>

button {

width: 100px;

height: 50px;

margin: 5px;

font-size: 16px;

}

</style>

</head>

<body>

<h1>ESP32 BattleBot Controller</h1>

<div>

<button onclick="sendCommand('forward')">Forward</button>

</div>

<div>

<button onclick="sendCommand('left')">Left</button>

<button onclick="sendCommand('stop')">Stop</button>

<button onclick="sendCommand('right')">Right</button>

</div>

<div>

<button onclick="sendCommand('backward')">Backward</button>

</div>

<script>

// Replace with your ESP32's IP address

const ESP32\_IP = "http://192.168.4.1"; // Update to your ESP32’s IP after connecting to its network

// Function to send commands to the ESP32

function sendCommand(command) {

fetch(`${ESP32\_IP}/${command}`)

.then(response => response.text())

.then(data => console.log(data))

.catch(error => console.error('Error:', error));

}

// Keyboard controls for the bot

document.addEventListener("keydown", (event) => {

switch (event.key) {

case "ArrowUp":

sendCommand("forward");

break;

case "ArrowDown":

sendCommand("backward");

break;

case "ArrowLeft":

sendCommand("left");

break;

case "ArrowRight":

sendCommand("right");

break;

case " ":

sendCommand("stop");

break;

default:

break;

}

});

</script>

</body>

</html>

### **Explanation of the HTML and JavaScript**

1. **Buttons**: Each button (Forward, Left, Right, Backward, and Stop) is tied to a function (sendCommand) that sends a specific command to the ESP32 when clicked.
2. **JavaScript sendCommand Function**: This function uses JavaScript's fetch API to send HTTP requests to the ESP32’s IP address and the corresponding command endpoint (/forward, /backward, /left, /right, or /stop).
3. **Keyboard Controls**: The keydown event listener checks which arrow key is pressed:
   * ArrowUp for forward
   * ArrowDown for backward
   * ArrowLeft for left
   * ArrowRight for right
   * Spacebar (" ") for stop
4. **Replace the ESP32 IP**: Update ESP32\_IP with the actual IP of the ESP32 once it’s connected.

**SETUP**

1. **Create the HTML File: Copy the HTML and JavaScript code from the previous message, paste it into a text editor, and save it as a file (e.g., controller.html).**
2. **Open the HTML File:**
   * **Connect your computer to the WiFi network created by the ESP32 (e.g., "ESP32\_BattleBot").**
   * **Open the saved controller.html file in a web browser (such as Chrome or Firefox).**
   * **This will bring up a control interface with buttons and keyboard support to send commands to your ESP32.**
3. **Update the IP Address:**
   * **The JavaScript const ESP32\_IP = "http://192.168.4.1"; should point to the IP address of the ESP32 (displayed when the Access Point starts).**
   * **If the ESP32's IP address is different, update the IP in the JavaScript code before saving and opening the file.**

**With this setup:**

* **The HTML file acts as your client-side control panel.**
* **The ESP32 code you uploaded from the Arduino IDE handles the commands sent from the HTML file.**

**This way, the HTML interface communicates with the ESP32 through WiFi, allowing you to control your bot remotely from the web page.**