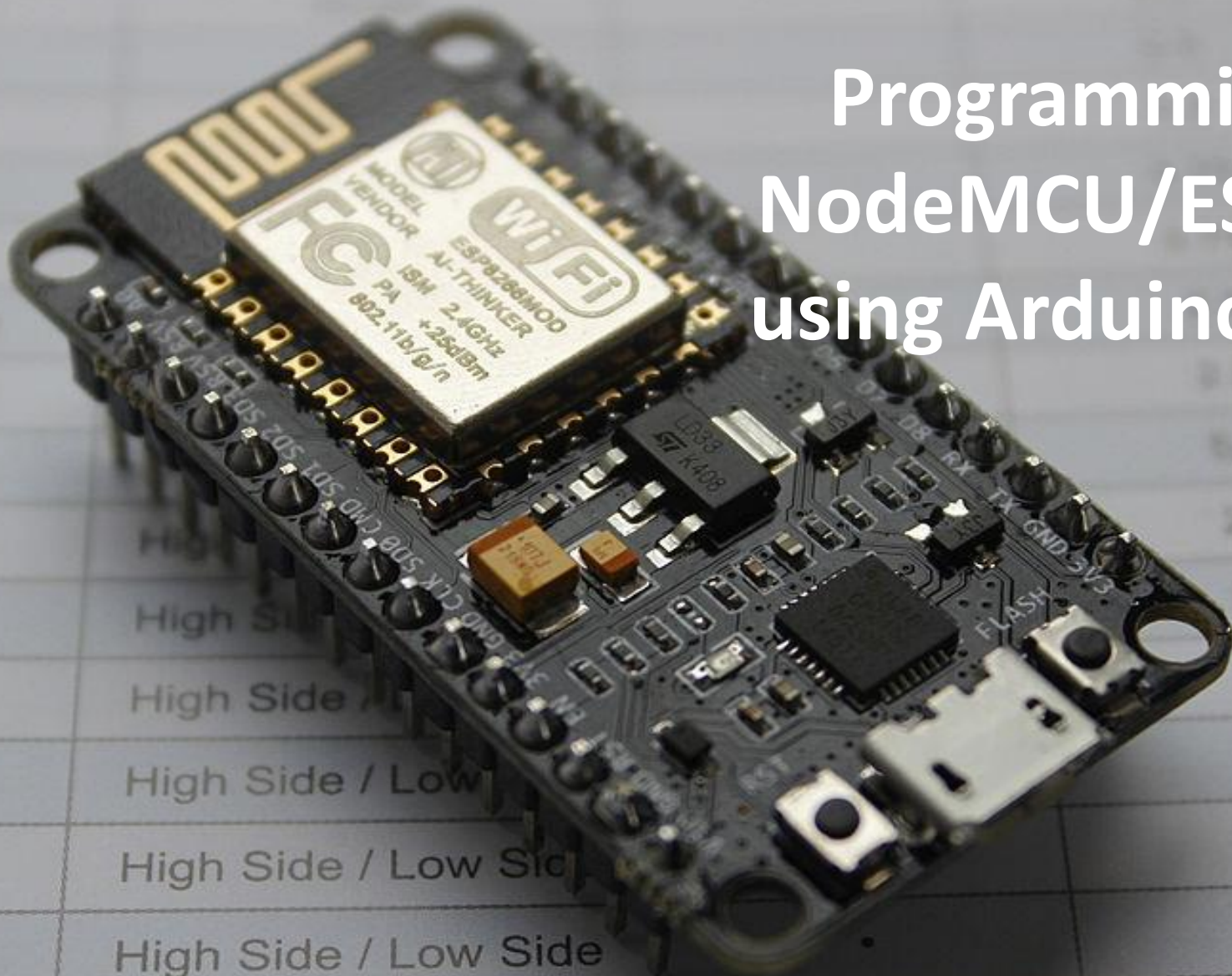


Programming NodeMCU/ESP32 using Arduino IDE



ESP32

- Open the Arduino IDE.
- Go to **Files** and click on the **Preference** in the Arduino IDE.
- Copy the URL below in the **Additional boards Manager**.

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

- Click **OK** to close the preference Tab.

Preferences



Settings Network

Sketchbook location:

C:\Users\z665059\OneDrive - ZF Friedrichshafen AG\Documents\Arduino

Browse

Editor language: System Default

(requires restart of Arduino)

Editor font size: 18

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

☐ Enable Code Folding

☒ Verify code after upload

☐ Use external editor

☐ Check for updates on startup

☒ Save when verifying or uploading

☐ Use accessibility features

Additional Boards Manager URLs: https://dl.espressif.com/dl/package_esp32_index.json



More preferences can be edited directly in the file

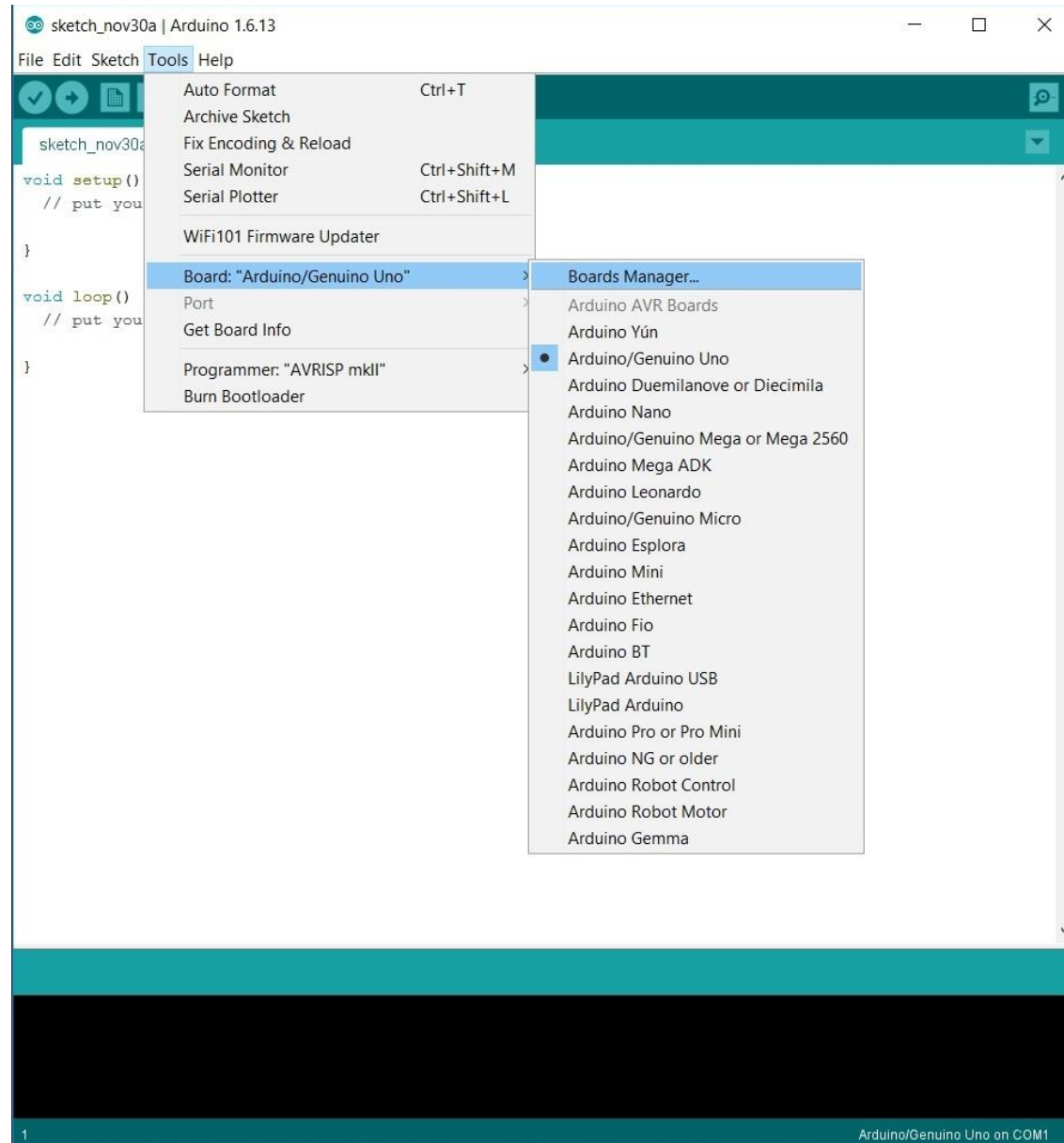
C:\Users\z665059\AppData\Local\Arduino15\preferences.txt

(edit only when Arduino is not running)

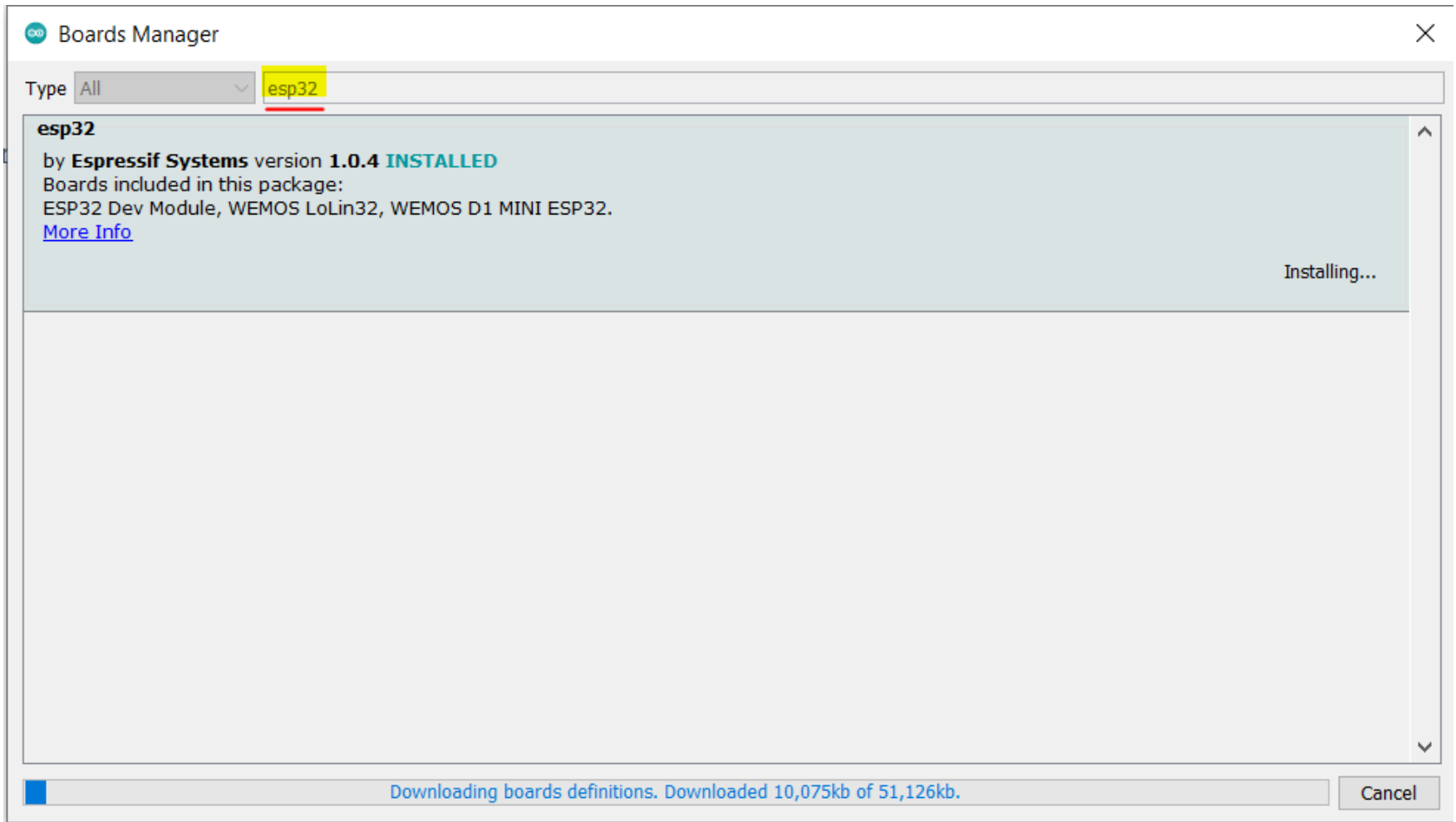
OK

Cancel

- Go to **Tools** and **Board**, and then select **Board Manager**.

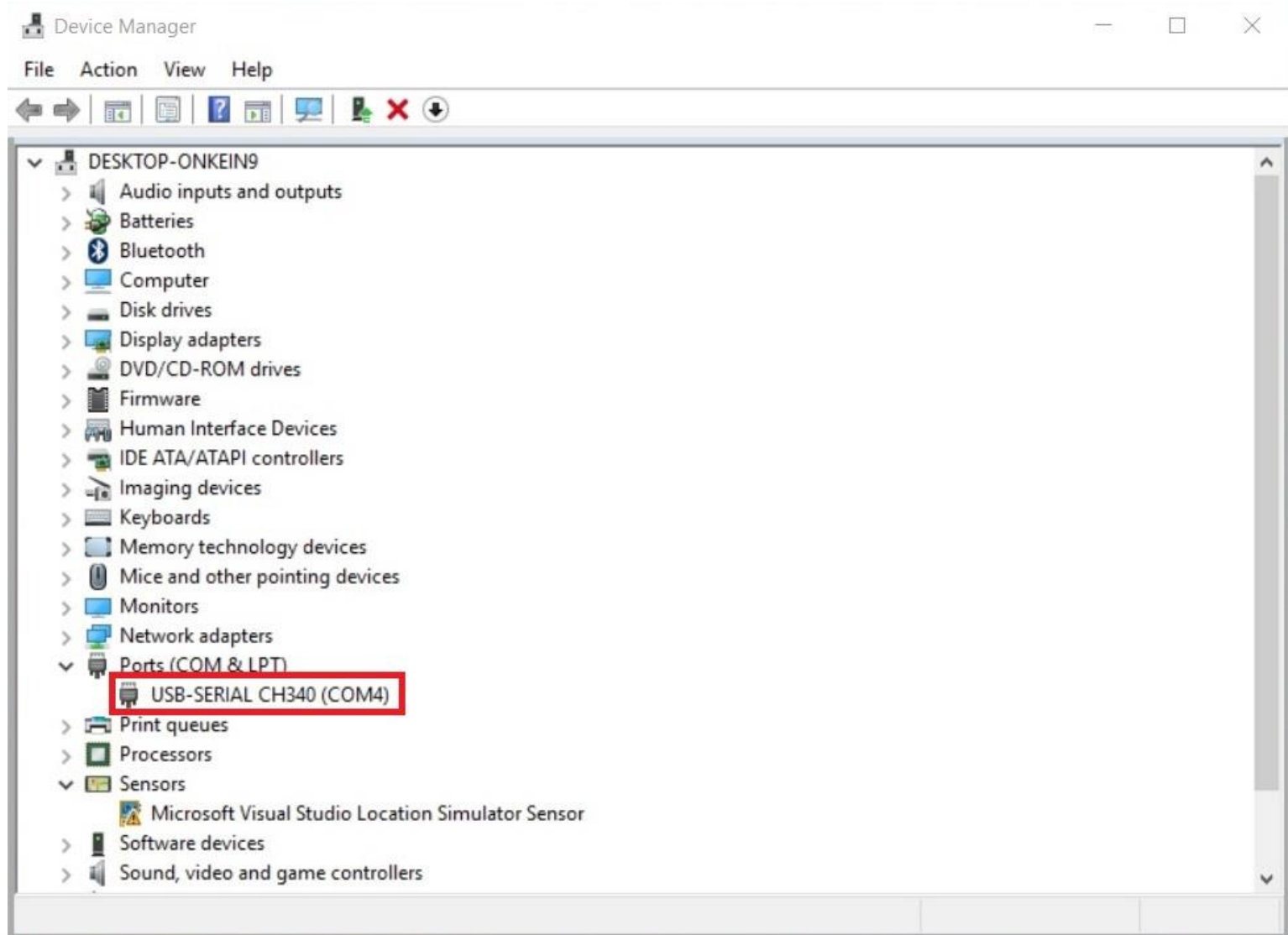


- Navigate to **esp32 by Espressif Systems** and install the software for Arduino.



- Once all the above process has been completed we are ready to program our **ESP32** with Arduino IDE.

- Connect NodeMCU to PC with USB to microUSB cable .
- Windows should automatically download necessary drivers for it.



Basics of Wi-Fi

- Devices that connect to Wi-Fi network are called stations.
- Connection to Wi-Fi is provided by an access point, that acts as a hub for one or more stations.
- Each access point is recognized by a SSID (**S**ervice **S**et **I**Dentifier).
- Clients can access services provided by servers in order to send, receive and process data.
- Server provide functionality to other programs or devices, called clients.
- We need to use ESP8266WiFi library to make Wi-Fi applications on NodeMCU.

WiFi library for ESP8266

- `WiFi.begin(ssid, password, channel, bssid, connect)`

Meaning of parameters is as follows:

ssid - a character string containing the SSID of Access Point we would like to connect to, may have up to 32 characters

password to the access point, a character string that should be minimum 8 characters long and not longer than 64 characters

channel of AP, if we like to operate using specific channel (optional)

bssid - mac address of AP (optional)

connect - a boolean parameter that if set to false, will instruct module just to save the other parameters without actually establishing connection to the access point

- `WiFi.localIP()`

Station Mode

- `WiFi.status()`

Function returns one of the following connection statuses:

`WL_CONNECTED` after successful connection is established

`WL_NO_SSID_AVAIL` in case configured SSID cannot be reached

`WL_CONNECT_FAILED` if password is incorrect

`WL_IDLE_STATUS` when Wi-Fi is in process of changing between statuses

`WL_DISCONNECTED` if module is not configured in station mode

- `WiFi.isConnected()`

Soft Access Point Mode

- `WiFi.softAP(ssid, password, channel, hidden)`
- The first parameter of this function is required, remaining three are optional.
- Meaning of parameters

channel - optional parameter to set Wi-Fi channel, from 1 -13. Default channel is 1.

hidden - optional parameter, if set to true will hide SSID

- Function will return true or false depending on result of setting the soft-AP
- `WiFi.softAPgetStationNum()`
- The maximum number of stations that may be connected to ESP8266 soft-AP is five.

Scanning

- `WiFi.scanNetworks()`
- `WiFi.scanNetworks(async, show_hidden)`
- Both function parameters are of boolean type.

async - if set to true then scanning will start in background and function will exit without waiting for result. To check for result use separate function.

show_hidden - set it to true to include in scan result networks with hidden SSID.

- `WiFi.scanComplete()`

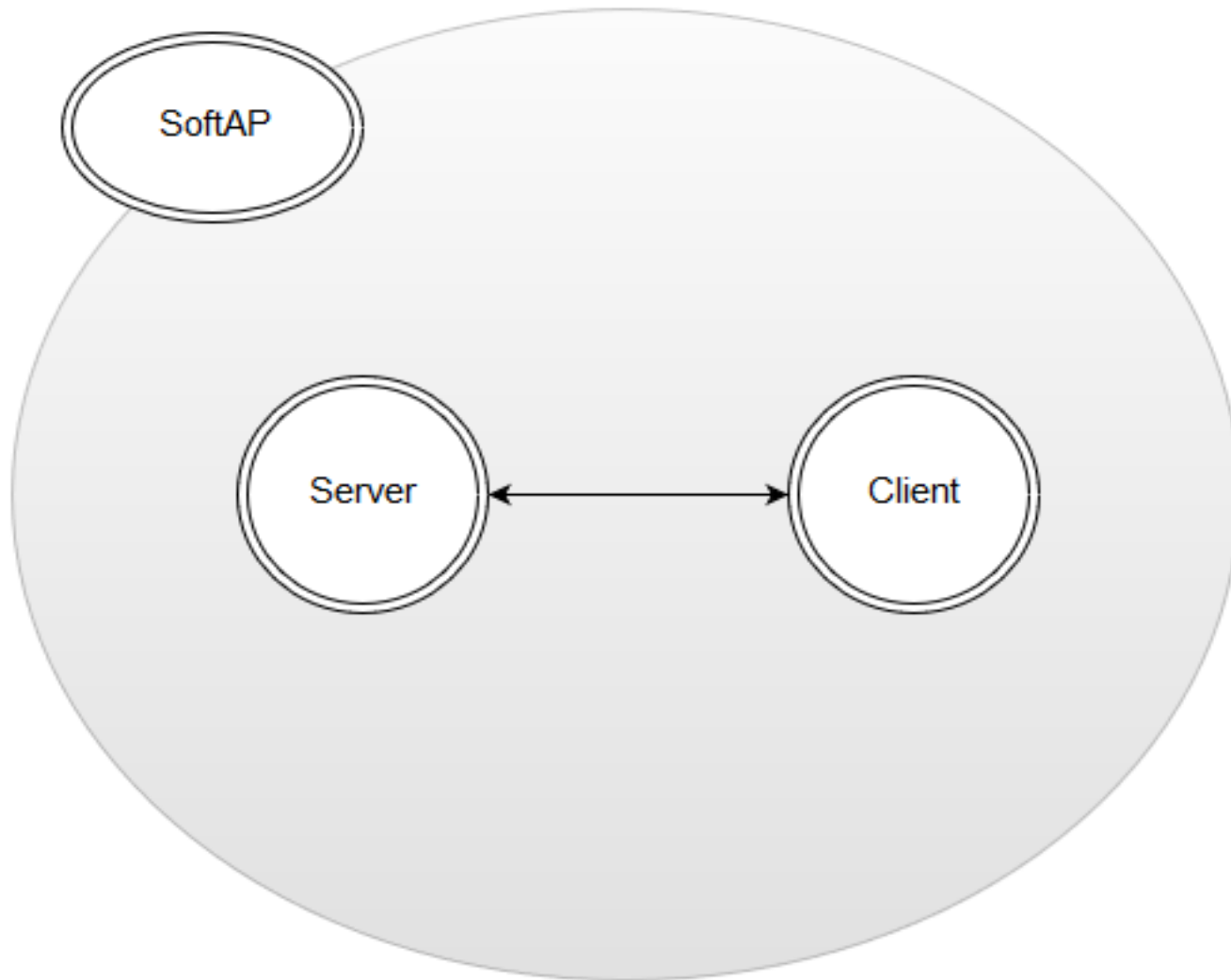
Client Mode

- Create a client object using
- `WiFiClient client;`
- Declare server address using
- `IPAddress server(74,125,115,105);`
- `client.connect(server, 80)`
- `client.stop()`
- `client.available()`
- Returns the number of bytes available for reading
- `client.write(data)`
- Write data to the server the client is connected to
- `client.read()`
- Read the next byte received from the server the client is connected to (after the last call to `read()`).

Server Mode

- `WiFiServer server(80);`
- Creates a server that listens for incoming connections on the specified port.
- `server.begin()`
- `server.available()`
- Returns a Client object, if no Client has data available for reading, this object will return false.
- `server.write(data)`
- Write data to all the clients connected to a server (one byte at a time).
- To read data from client use `available` to get client object and use `client.read()` to read data.

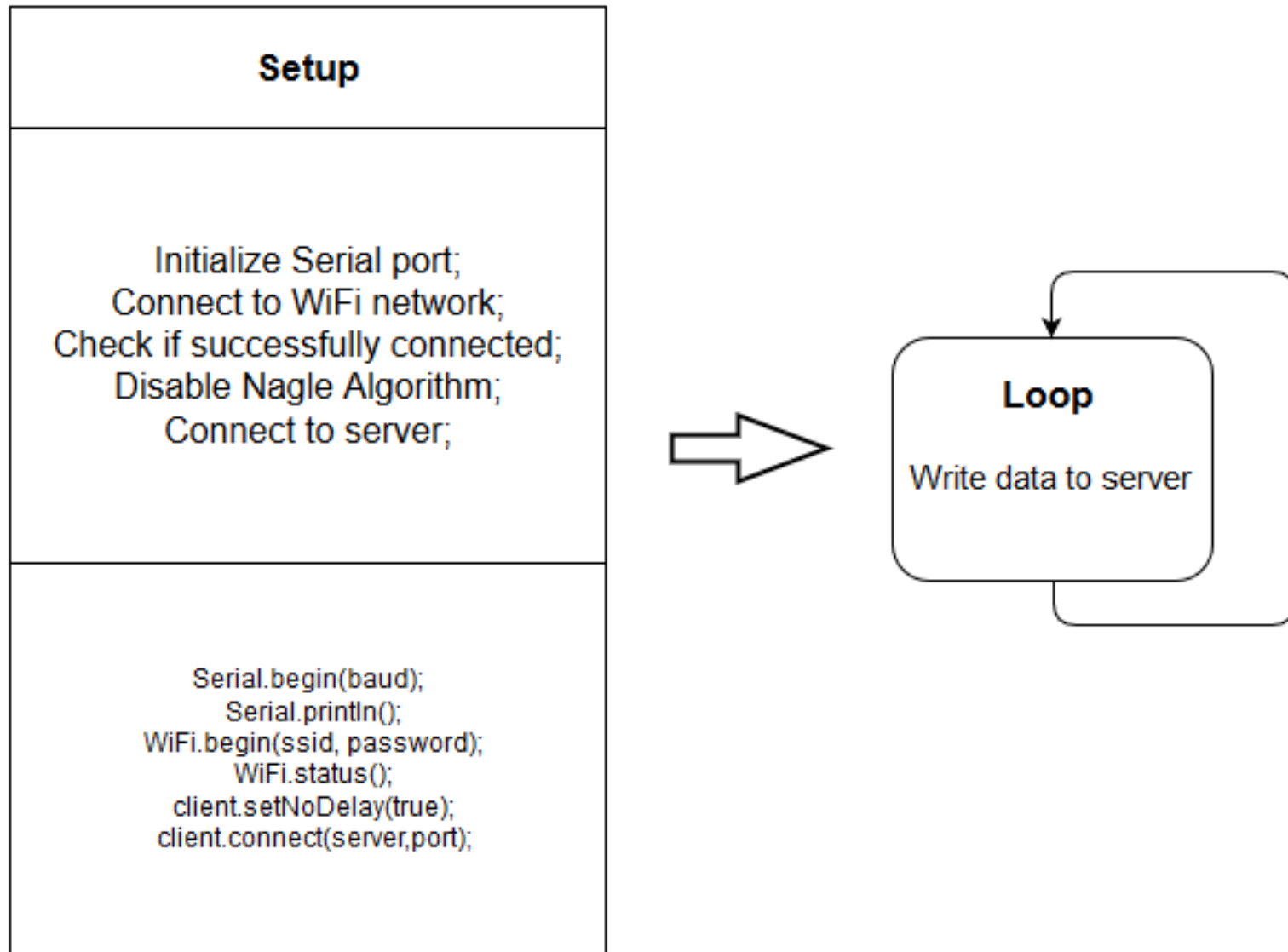
Basic Example



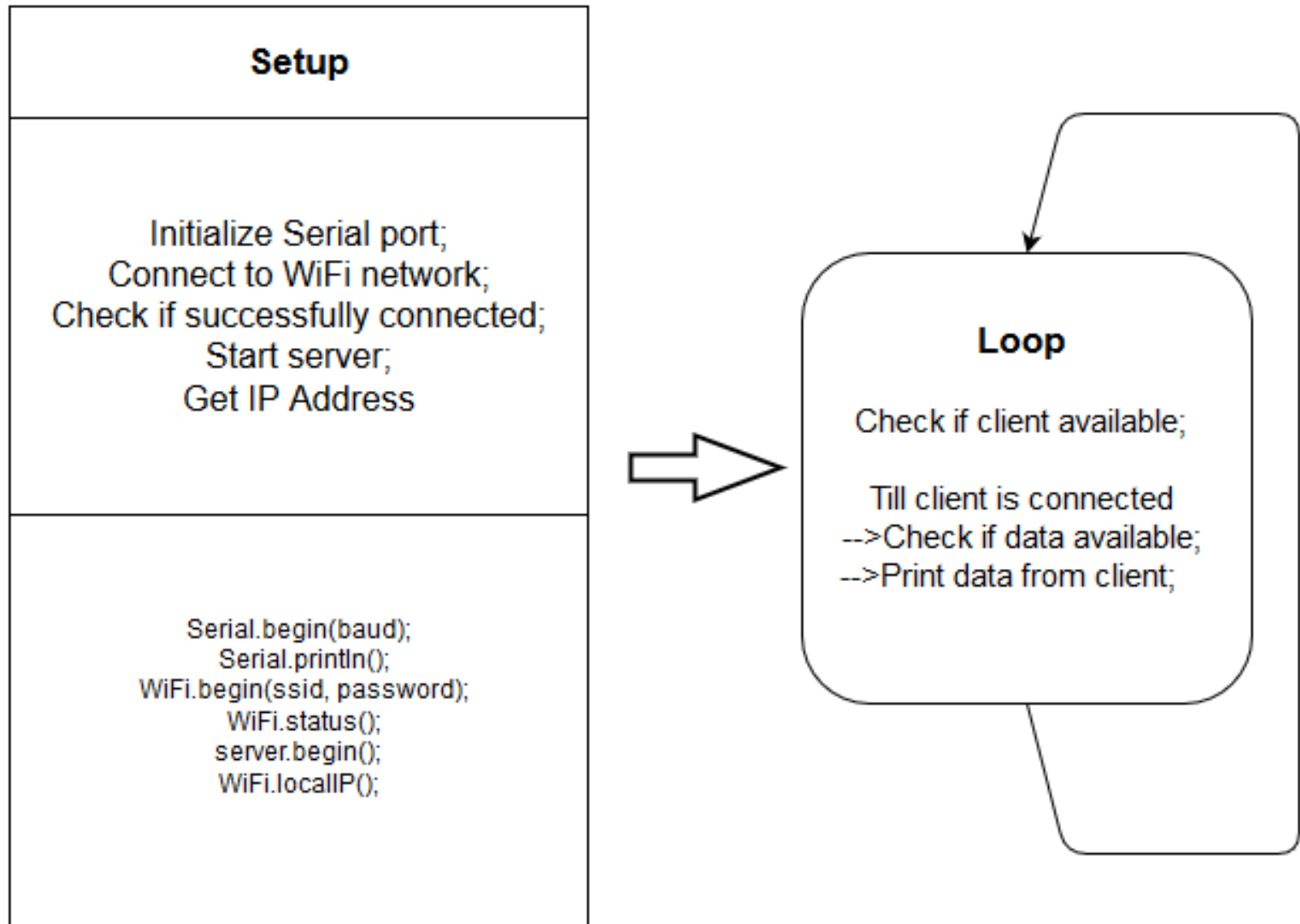
Writing your first program (sketch)

- The code always has at least 2 functions
 - `setup()`
 - `loop()`
- The `setup()` function is called when a sketch starts. Use it to initialize variables, pin modes, start using libraries, etc.
- The setup function will only run once, after each power up or reset.
- After creating a `setup()` function, which initializes and sets the initial values, the `loop()` function does precisely what its name suggests, and loops consecutively.
- Its very similar to C/C++.

Client



Server



PART2

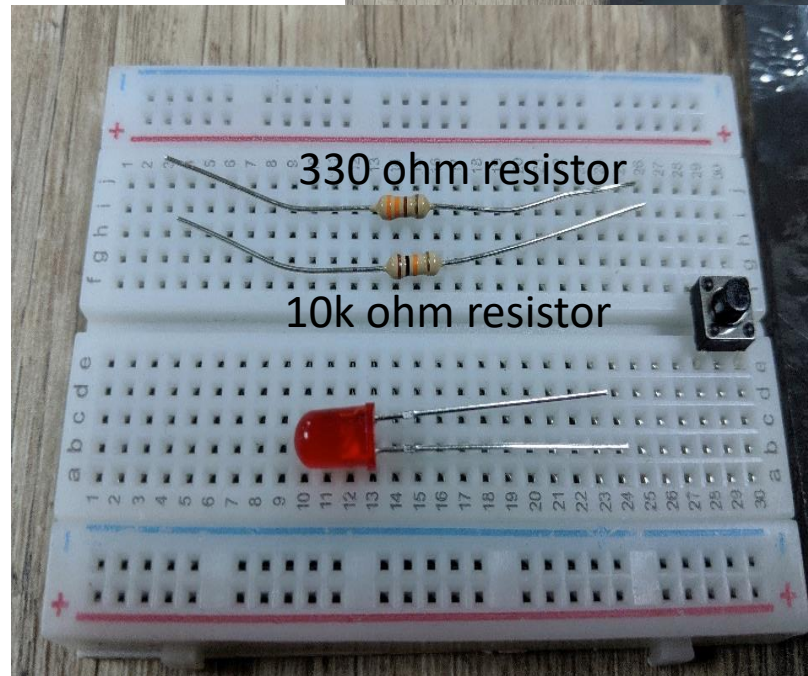
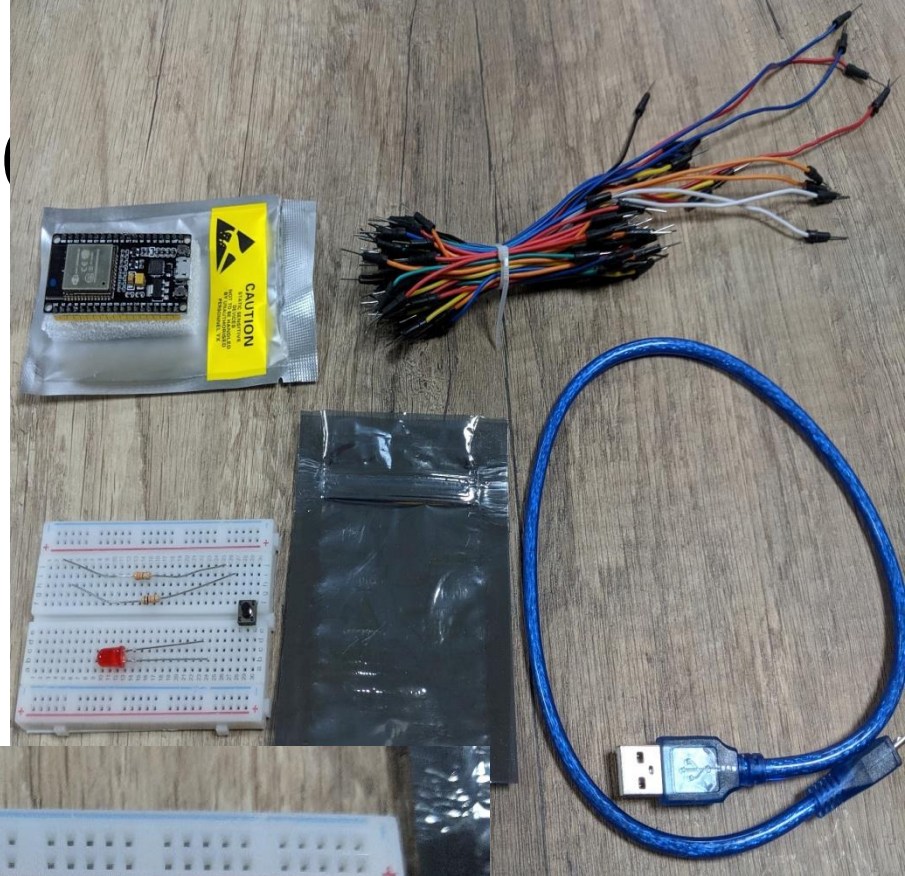
ESP32

SDK installation

- <https://github.com/yeokm1/iot-esp32-mcu-workshop>
- Install Serial Virtual Com Port drivers for your operating system if needed
 - <https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers>
- Install Arduino IDE:
 - <https://www.arduino.cc/en/main/software>
- Install ESP32 SDK addon to Arduino IDE
 - Add the following path to Additional Boards URL configuration
 - https://dl.espressif.com/dl/package_esp32_index.json

Components

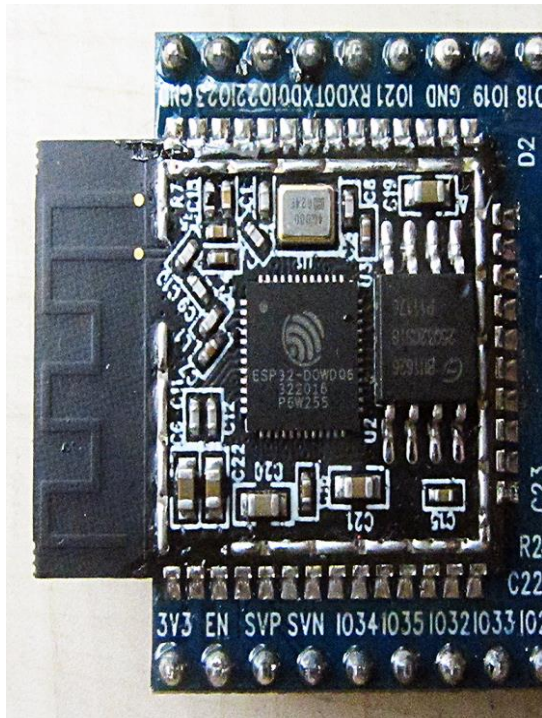
- ESP32 breakout board
- 50cm micro-USB cable
- Half-size breadboard
- Bunch of wires
- In ESD bag
 - Red LED
 - Push button
 - 330 ohm resistor
 - 10k ohm resistor



Agenda

- Basics of Wifi
 1. Setup ESP32 as a Station
 2. Setup ESP32 as a AP
- Basics of electronics
 1. Serial - Setting up of software SDK and testing
 2. Blink - Connecting and blinking LED
 3. Button - Connecting button and try to detect press
 4. Debounce - Concept of input debouncing
- The “Internet” portion of IoT
 5. Post - Posting Request to a server on button press
 6. Get - Repeatedly polling the server for on or off instruction

breakout



ESP32 Specs

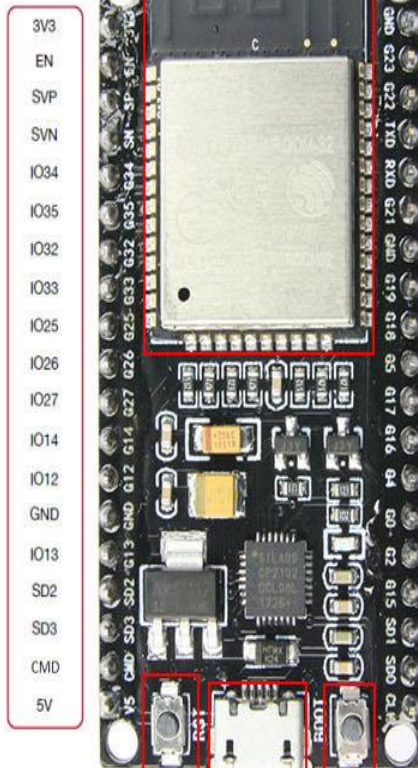


| Specs | Value |
|-----------------------|--|
| CPU | Xtensa dual/single-core 32-bit LX6 microcontroller, operating at 160/240 MHz |
| RAM | 520 KiB |
| Flash memory | 4 - 16 MiB |
| Network Connectivity | Wifi 802.11n 2.4Ghz + BT 4.2 |
| Peripheral Interfaces | 3x UART, 2x SPI, 2x I2C, ADC, SD touch sensors, etc |

JTAG



ESP-WROOM-32



Reset

Micro-B
USB

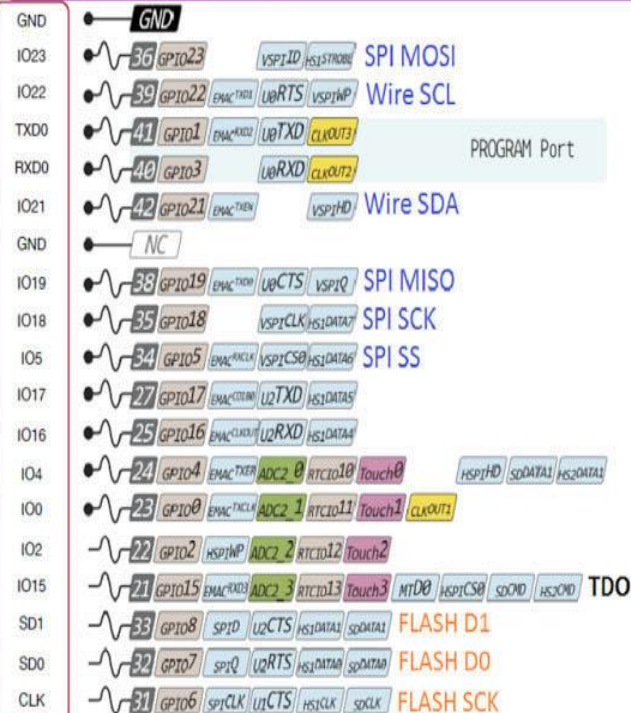
Boot

JTAG 20-pin

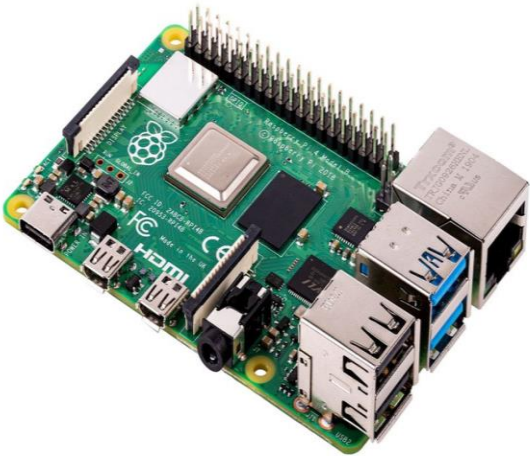
| | | | | |
|----------|-----------|-------|-----------|-----------|
| Red | 3V3 | VTref | 1 ● ● 2 | --- |
| Gray | RSTnTRST | | 3 ● ● 4 | GND GND |
| Orange | IO12 | TDI | 5 ● ● 6 | GND Black |
| Yellow | IO14 | TMS | 7 ● ● 8 | GND |
| Blue | IO13 | TCK | 9 ● ● 10 | GND Green |
| (to GND) | RTCK | | 11 ● ● 12 | GND White |
| Purple | IO15 | TDO | 13 ● ● 14 | GND |
| | RESET | | 15 ● ● 16 | GND |
| | DBGREQ | | 17 ● ● 18 | GND |
| | 5V-Supply | | 19 ● ● 20 | GND |

Pins 14, 16, 18, 20:

On some models like the high-end model J-Link PRO, these pins may not be connected to GND but are reserved for future use/extension. In case of doubt, leave open on target hardware.



Why not single-board computers like



| Specs | Arduino-compatible ESP32 | Raspberry Pi 4 B |
|---------------------------|------------------------------|--|
| CPU type | Microcontroller | Microprocessor |
| Speed | Dual Core 160/240Mhz | Quad core 1.5 Ghz |
| RAM | 520KB | 1 - 4 GB |
| GPU/Display | None | VideoCore VI GPU, 4K |
| Disk | 4 MiB | Depends on microSD card |
| Other connectivity | Wifi 802.11n 2.4Ghz + BT 4.2 | USB, Ethernet, HDMI, audio, Wifi 2.4/5Ghz 802.11ac, BT 5.0 |
| Operating System | None | Linux (usually Raspbian) |
| Real Time Operation | Better | Poorer |
| Typical Power consumption | 0.8W | 8W |

Arduino SDK

- Easy to use cross-platform IDE for microcontroller programming
- Support for many microcontrollers
- C/C++ language
- Numerous libraries



The image shows the Arduino IDE interface during the upload of a program to an ESP32 DevKit V1. The top window displays the source code for a file named '01-serial'. The code includes a setup function that initializes the serial port at 115200 baud and prints 'Setup started'. It also sets log modes for test, warning, info, and debug. The loop function prints 'Testing serial output' every 1000ms. The bottom window shows the progress of the upload, indicating that 3072 bytes were written to the flash memory in 1.8 seconds. The status bar at the bottom shows '2 DOIT ESP32 DEVKIT V1 on /dev/cu.SLAB_USBtoUART'.

```

01-serial
1 // Runs once on bootup
2 void setup() {
3   Serial.begin(115200);
4   Serial.println("Setup started");
5
6   //ESP32 specific log modes
7   log_e("test - error");
8   log_w("test - warning");
9   log_i("test - info");
10  log_d("test - debug");
11
12  //No verbose option in Arduino IDE
13  log_v("test - verbose");
14
15
16 // Runs as infinite loop like "while(true)" after setup() completes
17 void loop() {
18   Serial.println("Testing serial output");
19
20   //Pause here for 1000ms
21   delay(1000);
22 }
  
```

Done uploading.

Writing at 0x00000000... (0 %)
Writing at 0x00001c00... (57 %)
Writing at 0x00002000... (71 %)
Writing at 0x00002400... (85 %)
Writing at 0x00002800... (100 %)
Wrote 202304 bytes (101732 compressed) at 0x000010000 in 1.8 seconds (effective 894.5 Kbytes/s)
Hash of data verified.
Compressed 3072 bytes to 144...

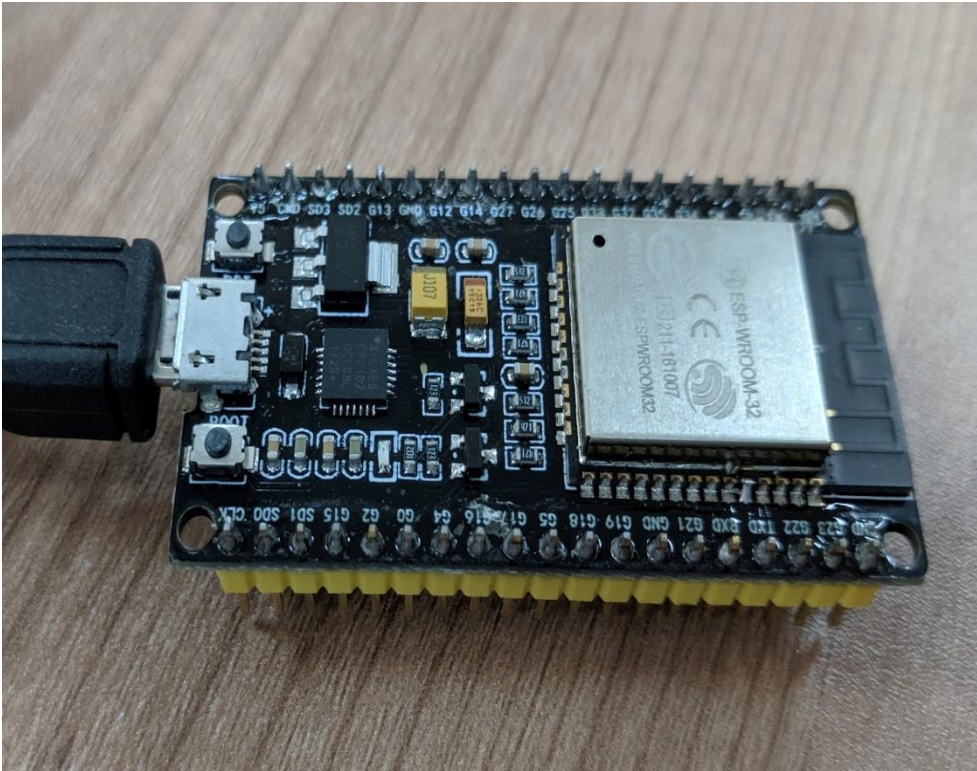
Writing at 0x00000000... (100 %)
Wrote 3072 bytes (144 compressed) at 0x00000000 in 0.0 seconds (effective 3000.3 Kbytes/s)
Hash of data verified.
Leaving...
Hard resetting via RTS pin...

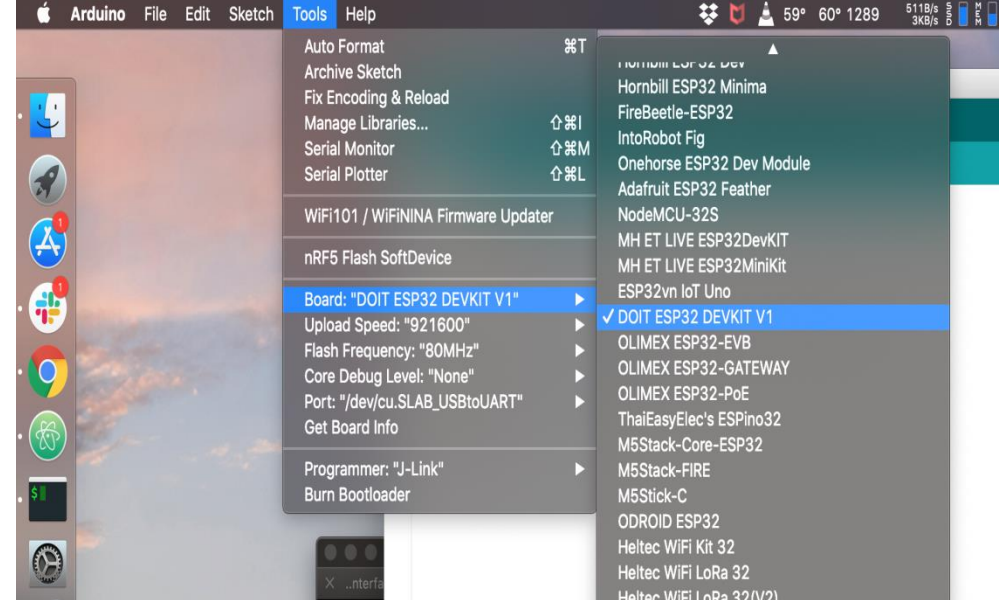
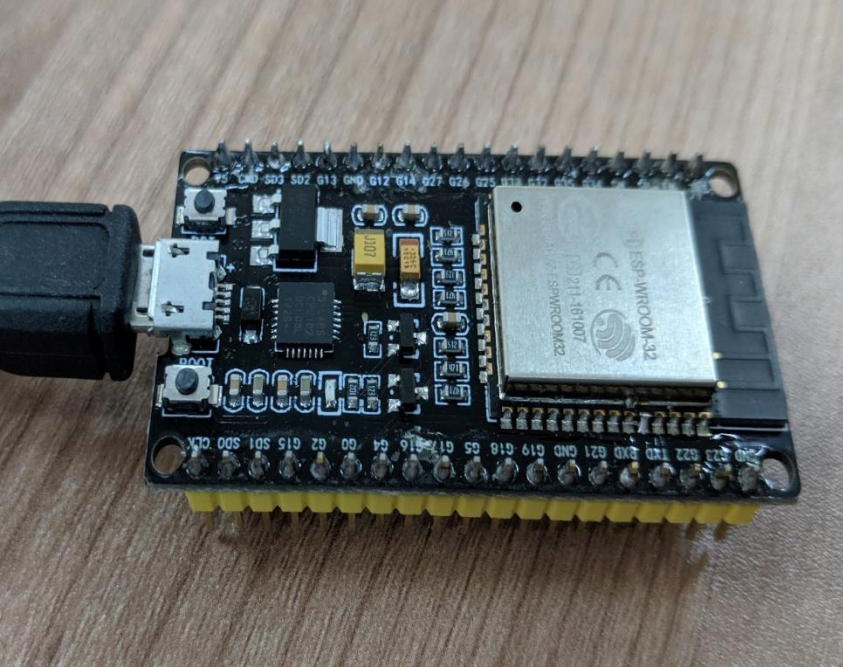
2 DOIT ESP32 DEVKIT V1 on /dev/cu.SLAB_USBtoUART

01-Serial

1. Verify Arduino SDK installed correctly as in the parent README.md
2. How to flash an Arduino program?

and ESP32 log





```

01-serial | Arduino 1.8.9
1 // Runs once on bootstrap
2 void setup(){
3   Serial.begin(115200);
4   Serial.println("Setup started");
5 }
6
7 // Runs as infinite loop like "while(true)" after setup() completes
8 void loop(){
9
10  //ESP32 specific log modes
11  log_e("test - error");
12  log_w("test - warning");
13  log_i("test - info");
14  log_d("test - debug");
15
16  //No verbose option in Arduino IDE
17  log_v("test - verbose");
18
19  //Serial.println does not show up when set to info/debug"
20  Serial.println("Testing serial output");
21  Serial.println();
22
23  //Pause here for 1000ms
24  delay(1000);
25 }

```

Done uploading.
Leaving...
Hard resetting via RTS pin...

```

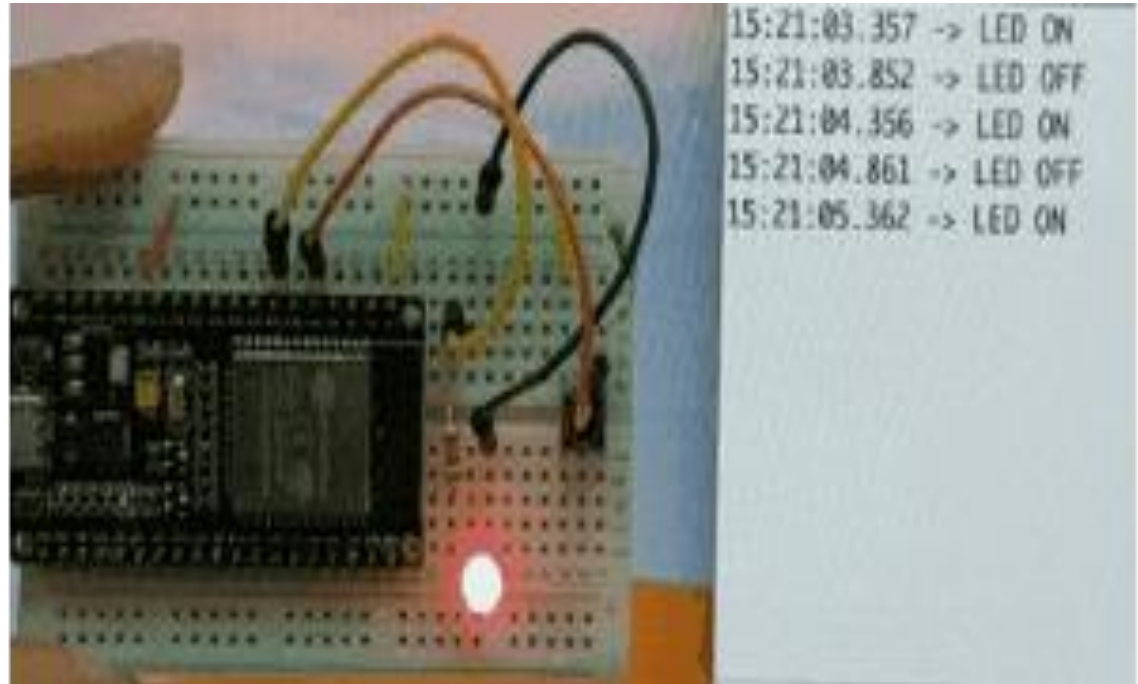
15:25:23.772 -> [W] [01-serial.ino:12] loop(): test - warning
15:25:23.772 -> [I] [01-serial.ino:13] loop(): test - info
15:25:23.772 -> [D] [01-serial.ino:14] loop(): test - debug
15:25:24.776 -> [E] [01-serial.ino:11] loop(): test - error
15:25:24.776 -> [W] [01-serial.ino:12] loop(): test - warning
15:25:24.776 -> [I] [01-serial.ino:13] loop(): test - info
15:25:24.776 -> [D] [01-serial.ino:14] loop(): test - debug
15:25:25.790 -> [E] [01-serial.ino:11] loop(): test - error
15:25:25.790 -> [W] [01-serial.ino:12] loop(): test - warning
15:25:25.790 -> [I] [01-serial.ino:13] loop(): test - info
15:25:25.790 -> [D] [01-serial.ino:14] loop(): test - debug
15:25:26.789 -> [E] [01-serial.ino:11] loop(): test - error
15:25:26.789 -> [W] [01-serial.ino:12] loop(): test - warning
15:25:26.789 -> [I] [01-serial.ino:13] loop(): test - info
15:25:26.789 -> [D] [01-serial.ino:14] loop(): test - debug
15:25:27.792 -> [E] [01-serial.ino:11] loop(): test - error
15:25:27.792 -> [W] [01-serial.ino:12] loop(): test - warning
15:25:27.792 -> [I] [01-serial.ino:13] loop(): test - info
15:25:27.792 -> [D] [01-serial.ino:14] loop(): test - debug
15:25:28.793 -> [E] [01-serial.ino:11] loop(): test - error
15:25:28.793 -> [W] [01-serial.ino:12] loop(): test - warning
15:25:28.793 -> [I] [01-serial.ino:13] loop(): test - info
15:25:28.793 -> [D] [01-serial.ino:14] loop(): test - debug
15:25:29.784 -> [E] [01-serial.ino:11] loop(): test - error
15:25:29.784 -> [W] [01-serial.ino:12] loop(): test - warning
15:25:29.784 -> [I] [01-serial.ino:13] loop(): test - info
15:25:29.822 -> [D] [01-serial.ino:14] loop(): test - debug
15:25:30.806 -> [E] [01-serial.ino:11] loop(): test - error
15:25:30.806 -> [W] [01-serial.ino:12] loop(): test - warning
15:25:30.806 -> [I] [01-serial.ino:13] loop(): test - info
15:25:30.806 -> [D] [01-serial.ino:14] loop(): test - debug

```

- Magnifying glass opens serial console
- Show Timestamps
- Newline
- 115200 baud

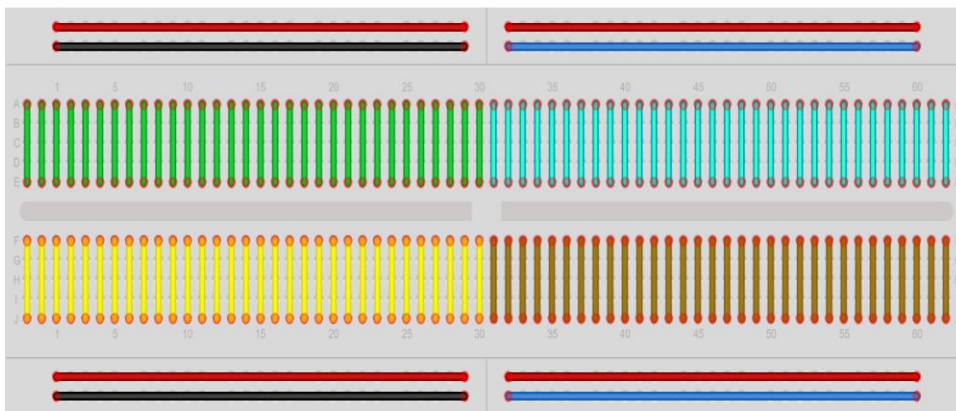
02-Blinky

1. Connect an LED
2. Blink from the LED



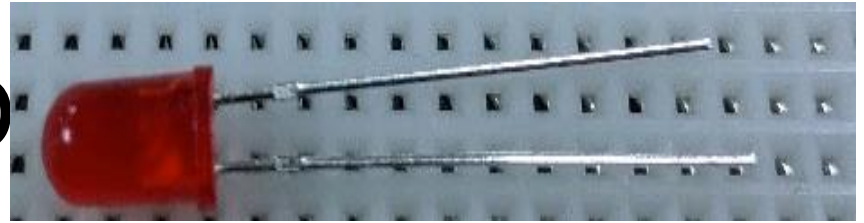
Half/Full Breadboard

- Base Prototyping component
- Continuous lines indicate those holes are

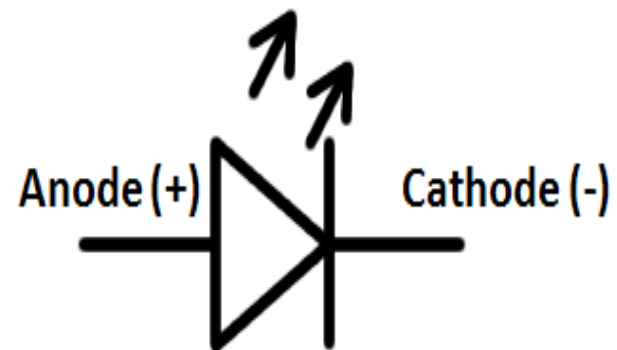


- Horizontal usually for power
- Convention: **Red** for +, Black/**Blue** for -
- Vertical for your components

LED

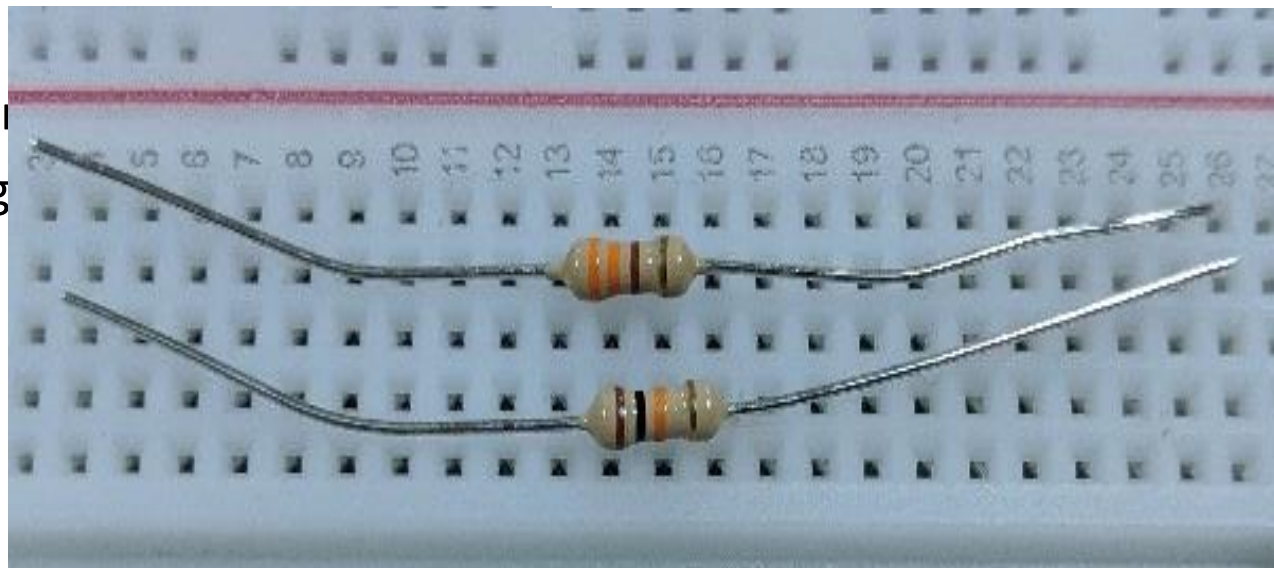
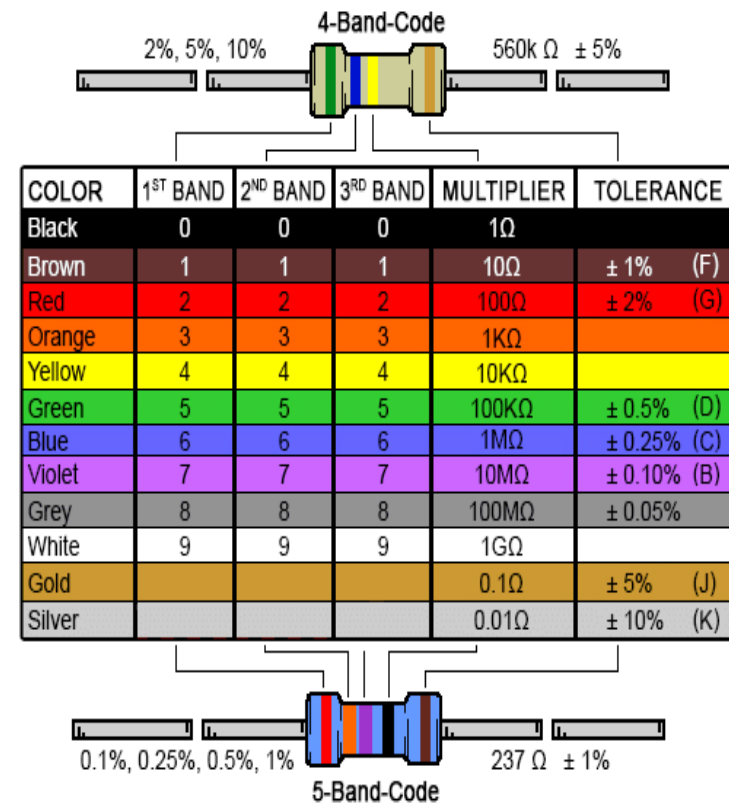


- “A **light-emitting diode (LED)** is a semiconductor light source that emits light when current flows through it. ”
- Diode: Component allowing current flow
- Anode +: Longer leg
- Cathode -: Shorter Leg



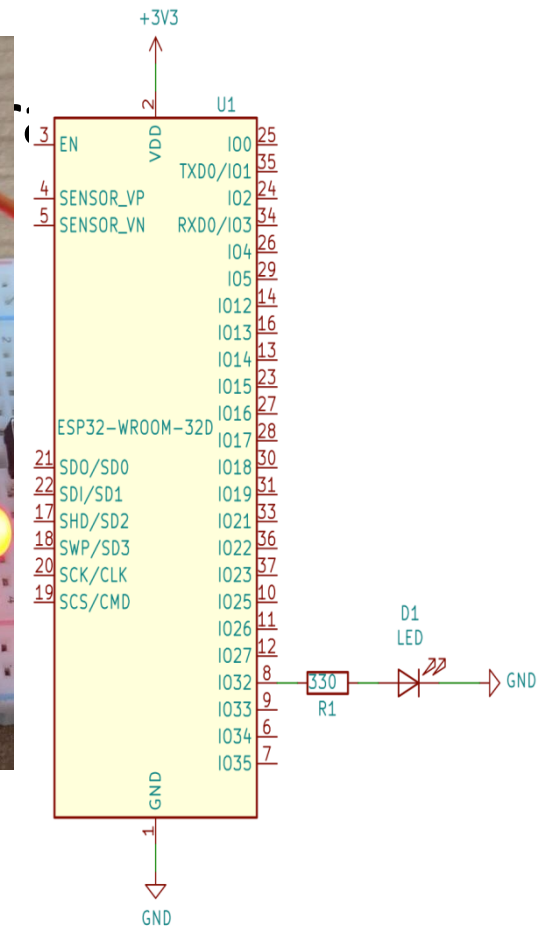
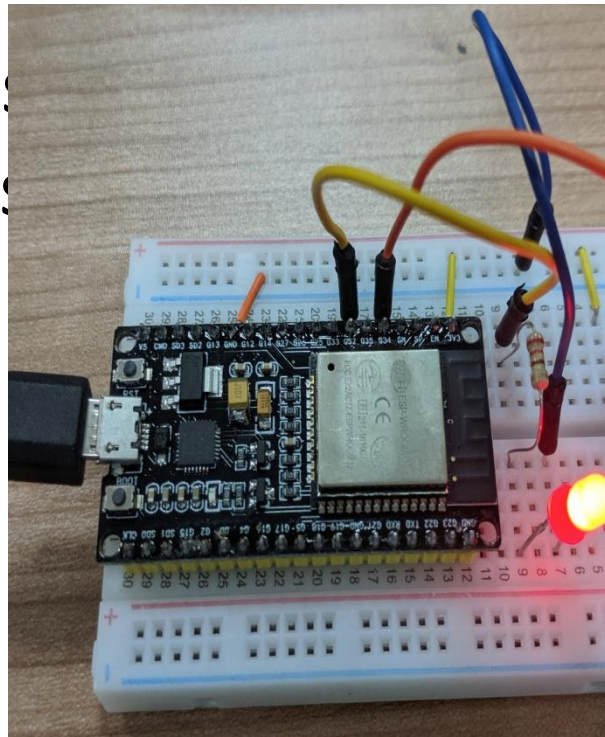
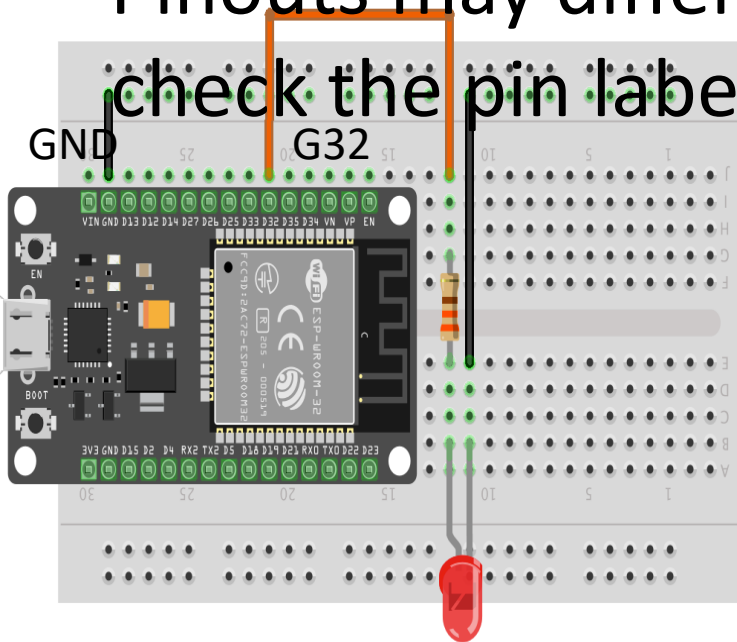
Resistor

- Resists the flow of current
- Resistor Colour bands
- 4-band 330 ohm
 - Orange, Orange, Brown, Gold
 - 3, 3, x10, 5% tolerance
- 4-band 10K ohm
 - 1, 0, x1000, 5% tolerance
 - Brown, Black, Orange

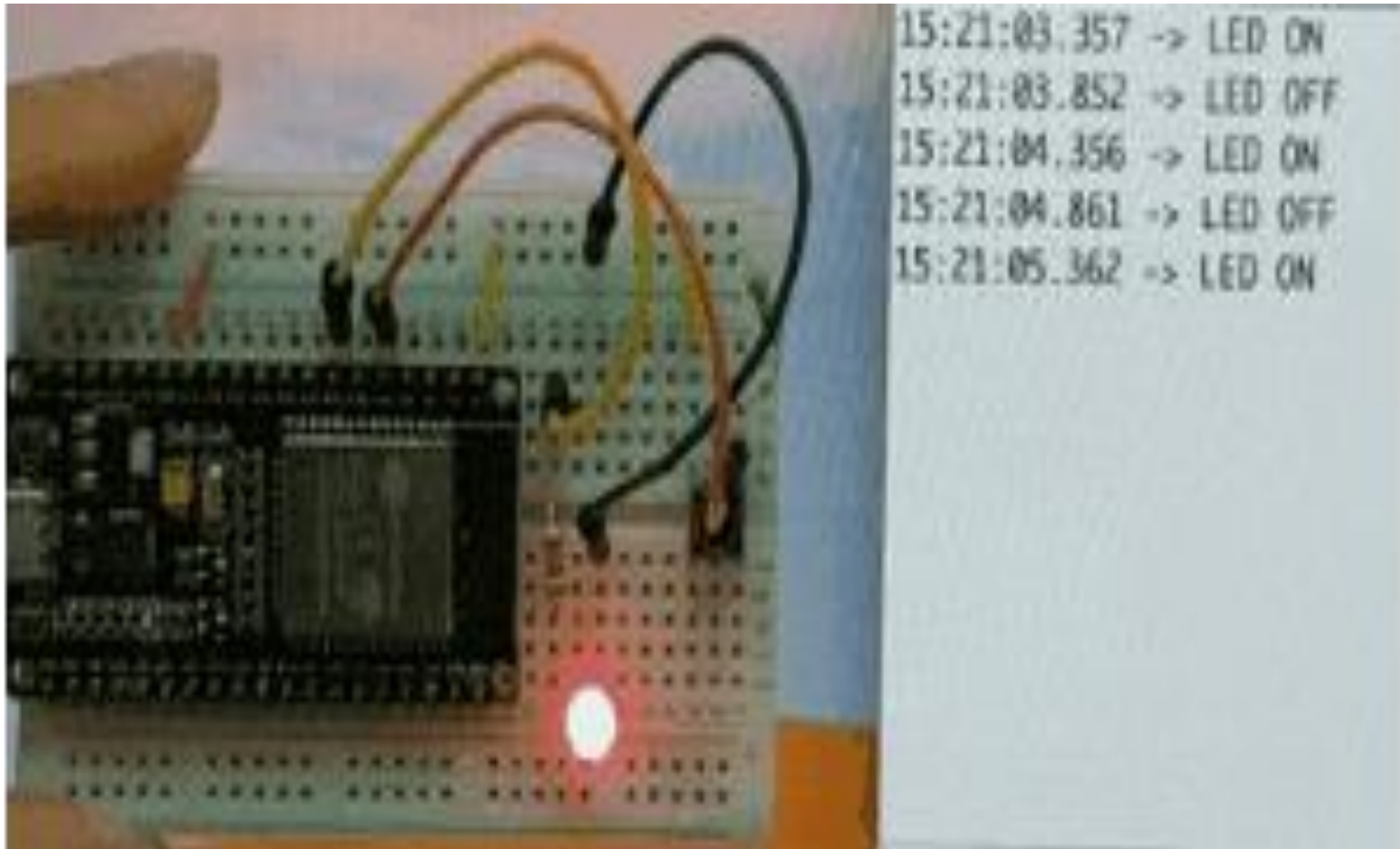


Connecting the LED setup

- Unplug USB cable before working on electronics
- Breadboard + ESP32 board (G32, GND) + 330 ohm resistor + LED + wires
- Pinouts may differ so check the pin labels

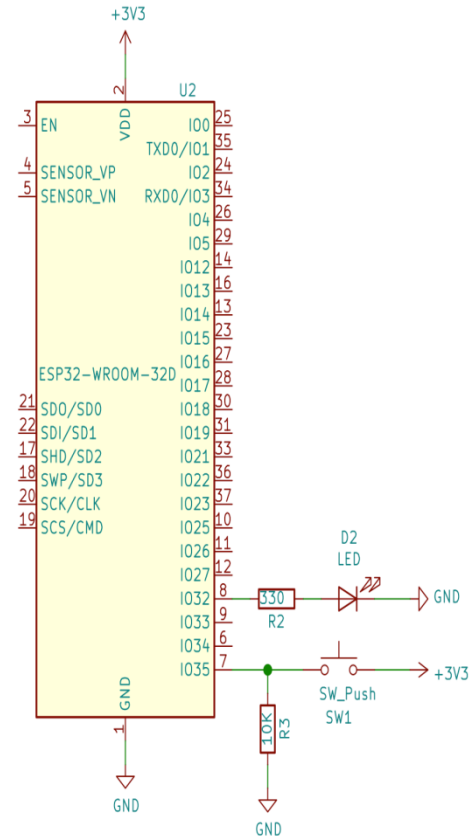
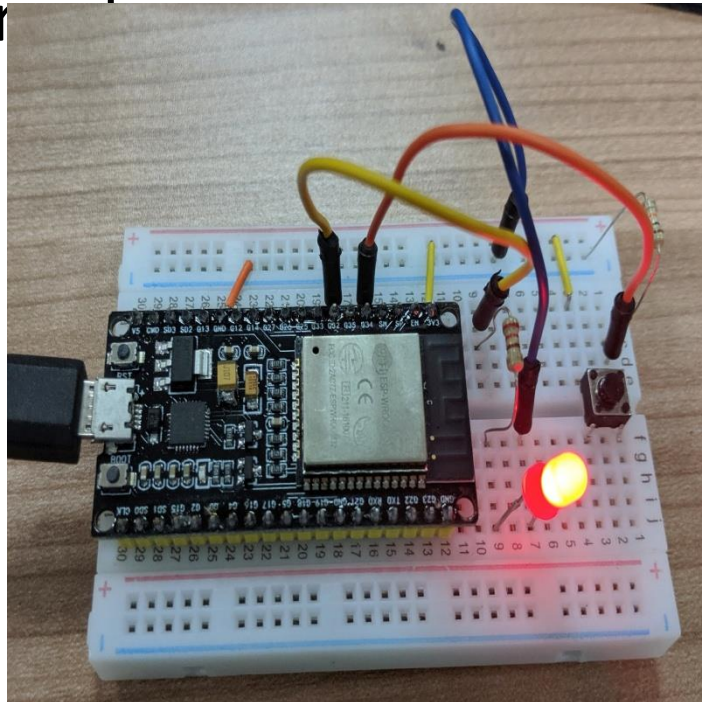
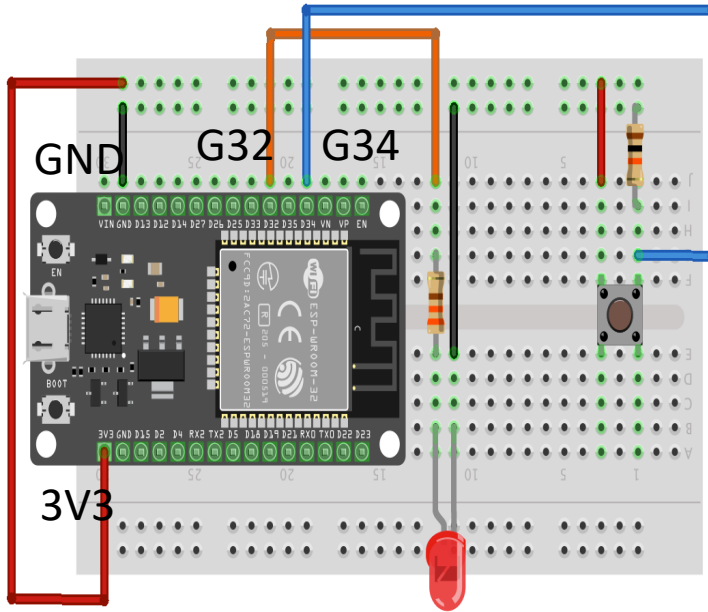


Program the board with 02-blinky.ino



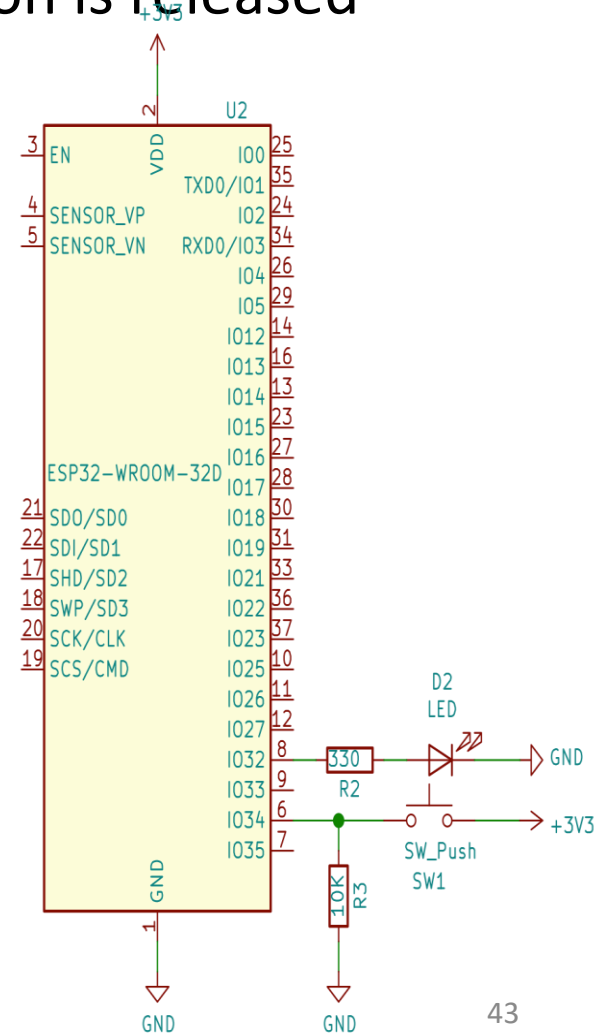
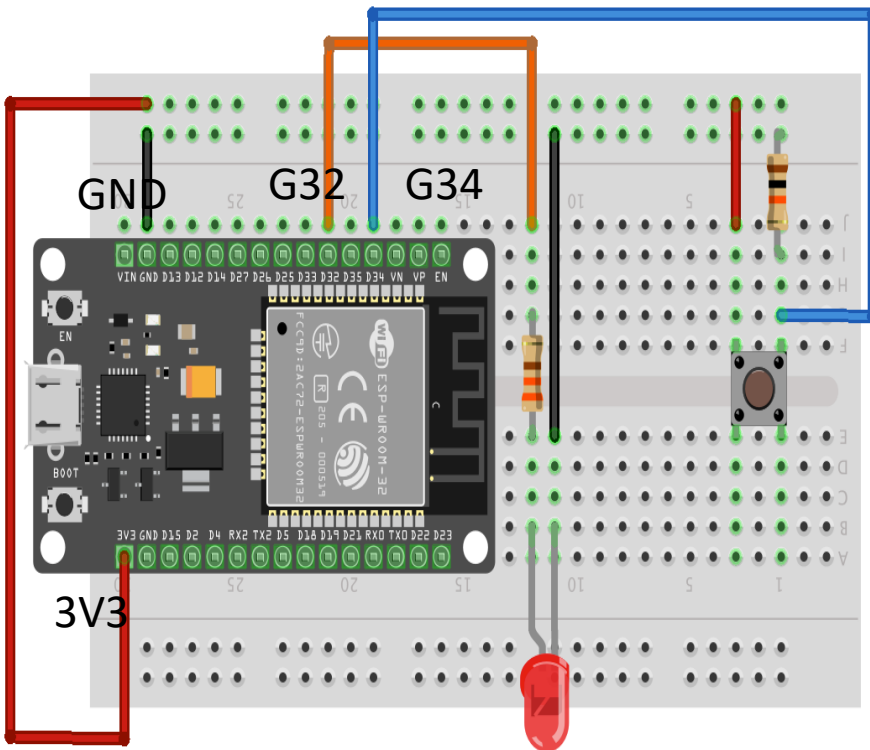
03-button button setup

- 10k ohm resistor + button + wires -> G34
- Actual breakout 3V3 pin position **differs from picture**
- “Pull down” resistor



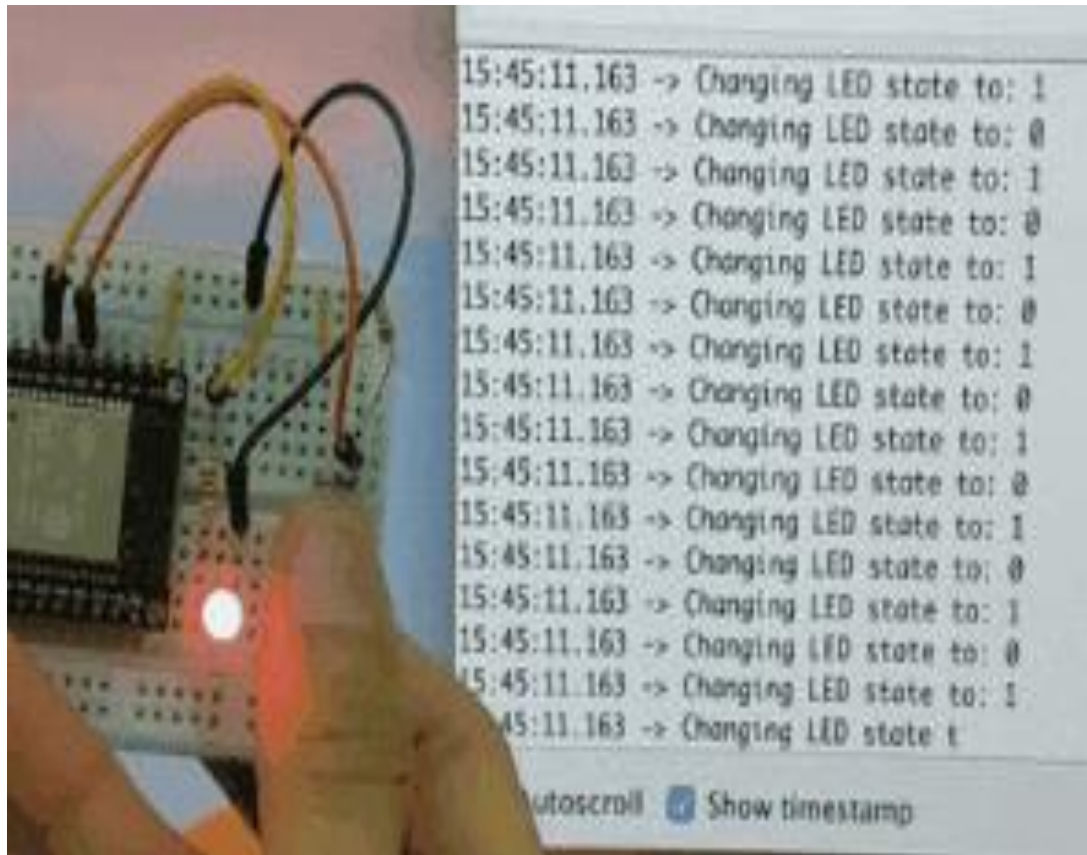
Pull down resistor for button

- Resistor “pulls” IO34 to ground when button is released
- Guarantees that IO34 pin goes to 0



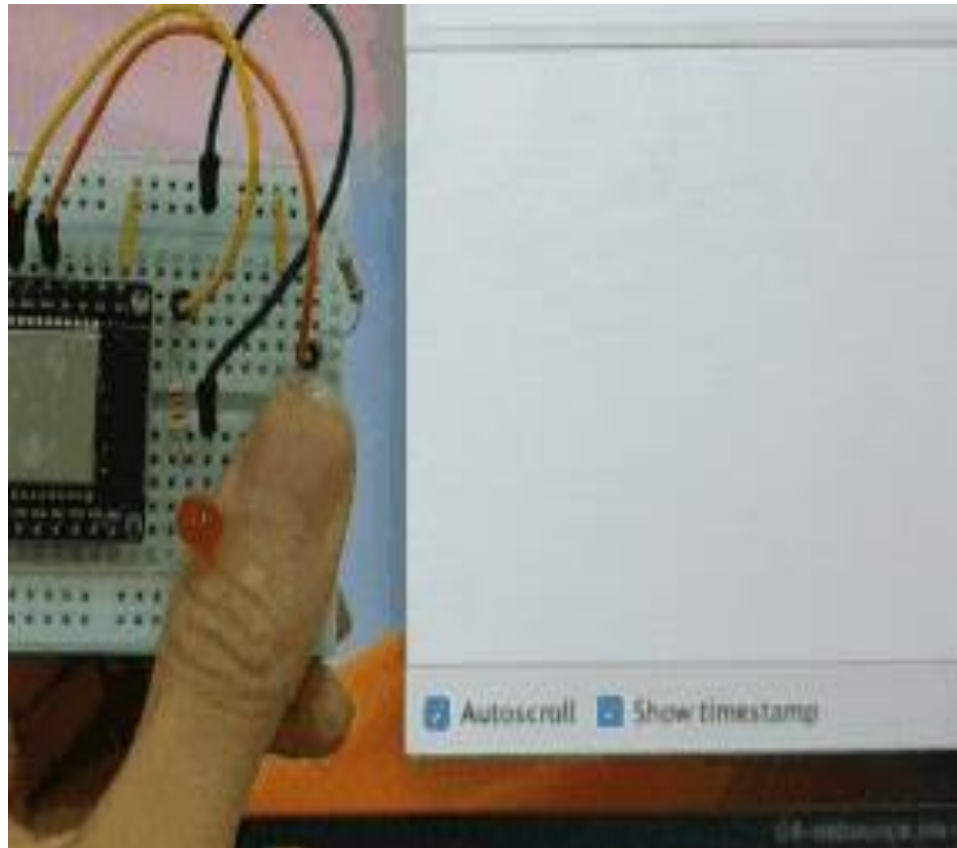
Program the board with 03-button.ino

- We want to toggle LED on each button press
- Observe looping speed of button press detection is too fast to be useful



04-debounce

- Process one input within an interval



05-wifi-post

1. Making a POST request to the server via a button press
 2. Able to receive a POST request using a Go program
- WIFI_SSID: iot-workshop

The image shows two terminal windows. The left window is an SSH session to a machine named 'yeokm1@YKM-testing'. It shows the execution of 'uname -a' and 'sudo ./server'. The 'server' program outputs a series of messages indicating it has received POST requests from an ESP32 with values 1234, 1235, 1236, 1237, and 1238. The right window is a serial terminal window titled '/dev/cu.SLAB_USBtoUART'. It shows a log of events: 'Setup complete', 'Button pressed, sending: ESP32 value 1234', 'Received a POST request', 'Button pressed, sending: ESP32 value 1235', and so on, corresponding to the values seen in the left window. The serial terminal window has a 'Send' button and checkboxes for 'Autoscroll' and 'Show timestamp', with a baud rate of 115200.

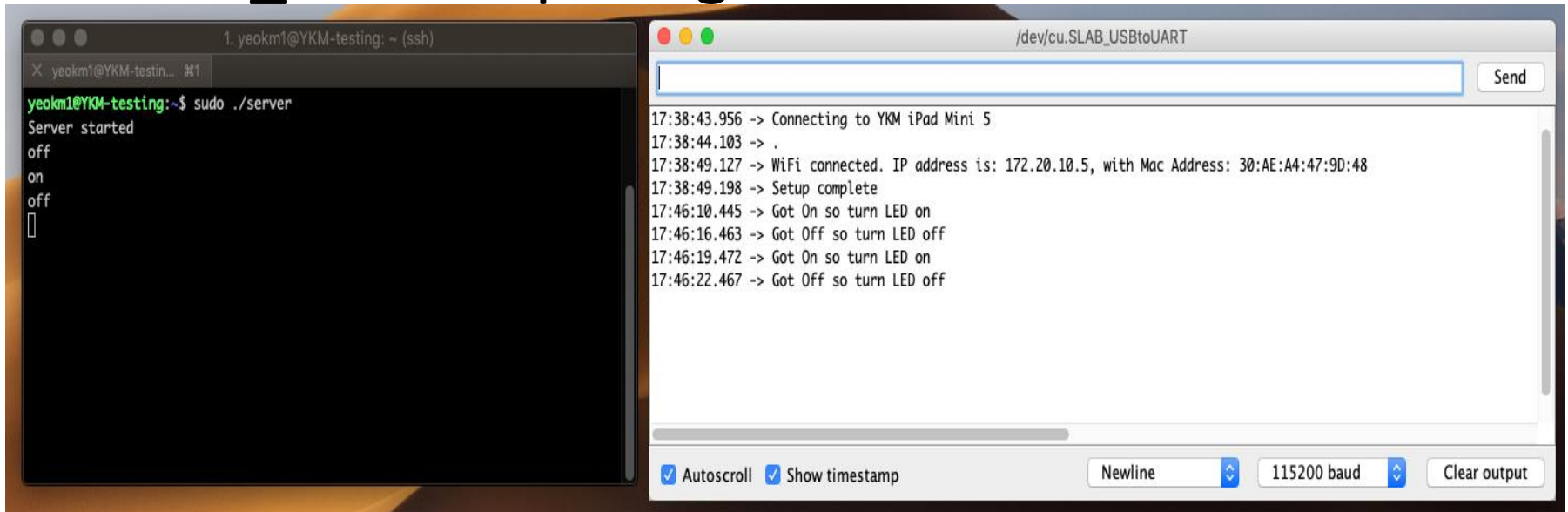
```
yeokm1@YKM-testing:~$ uname -a
Linux YKM-testing 4.18.0-1023-azure #24-18.04.1-Ubuntu SMP Tue Jun 25 15:14:42 UTC 2019 x86_64 x86_64 x86_64 GNU/Linux
yeokm1@YKM-testing:~$ sudo ./server
Server started

Received a POST request from: <ESP32          value 1234>
Received a POST request from: <ESP32          value 1235>
Received a POST request from: <ESP32          value 1236>
Received a POST request from: <ESP32          value 1237>
Received a POST request from: <ESP32          value 1238>
```

```
/dev/cu.SLAB_USBtoUART
17:34:00.341 -> Setup complete
17:34:03.192 -> Button pressed, sending: ESP32          value 1234
17:34:03.573 -> 200
17:34:03.573 -> Received a POST request
17:34:04.357 -> Button pressed, sending: ESP32          value 1235
17:34:04.749 -> 200
17:34:04.749 -> Received a POST request
17:34:05.421 -> Button pressed, sending: ESP32          value 1236
17:34:05.525 -> 200
17:34:05.525 -> Received a POST request
17:34:40.952 -> Button pressed, sending: ESP32          value 1237
17:34:41.343 -> 200
17:34:41.343 -> Received a POST request
17:35:10.637 -> Button pressed, sending: ESP32          value 1238
17:35:11.127 -> 200
17:35:11.127 -> Received a POST request
```

06-wifi-get

1. Making a Get Request to a server to get instruction
2. Serve a GET request using a Go program
 - WIFI_SSID: iot-workshop
 - WIFI_PASS: esp32isgreat



The image shows two terminal windows. The left window, titled '1. yeokm1@YKM-testing: ~ (ssh)', shows the execution of a Go program named 'server'. The user runs 'sudo ./server', and the program outputs 'Server started', followed by a series of 'off' and 'on' commands. The right window, titled '/dev/cu.SLAB_USBtoUART', shows the serial output of the program. It displays the connection process to a YKM iPad Mini 5, including the IP address (172.20.10.5) and Mac Address (30:AE:A4:47:9D:48). The program then sends a series of 'Got On so turn LED on' and 'Got Off so turn LED off' commands to the device. The right window also features a 'Send' button and a status bar with 'Autoscroll' and 'Show timestamp' checked, a 'Newline' button, a baud rate of '115200 baud', and a 'Clear output' button.

```
1. yeokm1@YKM-testing: ~ (ssh)
X yeokm1@YKM-testin... %1
yeokm1@YKM-testing:~$ sudo ./server
Server started
off
on
off
[]

/dev/cu.SLAB_USBtoUART
17:38:43.956 -> Connecting to YKM iPad Mini 5
17:38:44.103 -> .
17:38:49.127 -> WiFi connected. IP address is: 172.20.10.5, with Mac Address: 30:AE:A4:47:9D:48
17:38:49.198 -> Setup complete
17:46:10.445 -> Got On so turn LED on
17:46:16.463 -> Got Off so turn LED off
17:46:19.472 -> Got On so turn LED on
17:46:22.467 -> Got Off so turn LED off
```


Useful websites

- <http://frightanic.com/iot/comparison-of-esp8266-nodemcu-development-boards/>
- <https://www.gitbook.com/book/krzychb/esp8266wifi-library/details>
- <https://github.com/esp8266/Arduino/blob/master/doc/reference.md>
- <http://www.esp8266.com/wiki/doku.php>
- <https://github.com/esp8266/Arduino/blob/master/doc/libraries.md>
- <http://esp8266.github.io/Arduino/versions/2.0.0/doc/libraries.html>

Thank You