

## **IERG1000 Project 2 – Lab-2**

### **Arduino IDE with ESP32**

#### **Objectives**

- To download and install Arduino Integrated Development Environment (IDE);
- To download and install libraries for ESP32;
- To update USB-to-Serial driver (UART);
- To test the installed environment with an empty project;
- To flash the LED (digital output) on ESP32 board with official example;
- To use Arduino 'Serial Monitor' as a debug tool.
- To read a tactile switch (digital input) and use variables.

#### **Equipment / Component / Software**

- Arduino IDE (download from Internet);
- Libraries for ESP32 (download from Internet);
- Software driver for USB-to-Serial chip (from Internet / course blackboard)
- ESP32 microprocess module (x1);
- Micro USB cable for ESP32 (x1);
- Breadboard (x1);
- A bunch of wires for breadboard;
- 1K ohm resistor (x1);
- LED (x1);
- Tactile switch (x1);
- Thumb drive (x1, optional, if you don't have the right to write file to the computer)

## Introduction

As the story of Arduino, it is designed for someone who does not have engineering background but wants to use microprocessors. The original design of Arduino IDE is for ATmega CPU. You can get more information about Arduino from their page (<https://www.arduino.cc>).

Lately, many engineers have ported other microprocessors to the Arduino platform. ESP32 is one of the microprocessors embedded with WiFi and Bluetooth communication that can be used with Arduino. Then, you have to install Arduino IDE and ESP32 libraries to your computer.

You may not know any programming before. Thus, we start programming from some examples and simply make some modifications (copy / reuse).

We will do the following items in this Lab.

1. Flash a LED (on ESP32 board) and control it with a switch and output the status to a serial port, as debug tools.
2. Read environmental temperature and show it at the serial port.
3. Show the temperature (Centigrade) to OLED display module.

For students who have learnt programming, you can show more information on the display. (e.g. Fahrenheit, average, maximum and minimum temperature)

## Important notes:

The communication protocol between the ESP32 and the thermo-sensor modules is I2C. Use different color wires to indicate different signals.

RED= VIN and 3V3

BLACK = GND

WHITE = SCL

GREEN = SDA

Incorrect connections (e.g. SCL or SDA connected to 3V3 or GND) will damage the modules.

## Experiment 2.1: Install Arduino IDE

The Integrated Development Environment is Arduino, and special plug-in (libraries) for ESP32 are needed. The program is uploaded to the ESP32 through the serial port of your computer. A chip USB-to-Serial is on the ESP board as a bridge between your computer (USB) and the ESP32 (Serial). The driver of the USB chip may not be installed to your computer operation system (OS) automatically. You may have to install it manually.

### Note:

- A copy of the program and drivers are on the course blackboard (software).
- Arduino and ESP32 software update rapidly, the version may be changed.

### Procedures:

1. No need to connect ESP32 to your PC at this moment.
2. Download Arduino IDE; visit <https://www.arduino.cc/en/software> to download the latest Arduino IDE (ver 1.8.19) (Figure-1.1). Zip version for Windows is a green program and you do not need to install it.

### Downloads



**Arduino IDE 1.8.19**

The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. This software can be used with any Arduino board.

Refer to the [Getting Started](#) page for Installation instructions.

SOURCE CODE

Active development of the Arduino software is hosted by [GitHub](#). See the instructions for [building the code](#). Latest release source code archives are available [here](#). The archives are PGP-signed so they can be verified using [this](#) gpg key.

**DOWNLOAD OPTIONS**

**Windows** Win 7 and newer

**Windows** ZIP file 

**Windows app** Win 8.1 or 10 

**Linux** 32 bits

**Linux** 64 bits

**Linux** ARM 32 bits

**Linux** ARM 64 bits

**Mac OS X** 10.10 or newer

Release Notes

Checksums (sha512)

### Support the Arduino IDE

Since the release 1.x release in March 2015, the Arduino IDE has been downloaded **63,901,831** times — impressive! Help its development with a donation.

\$3

\$5

\$10

\$25

\$50

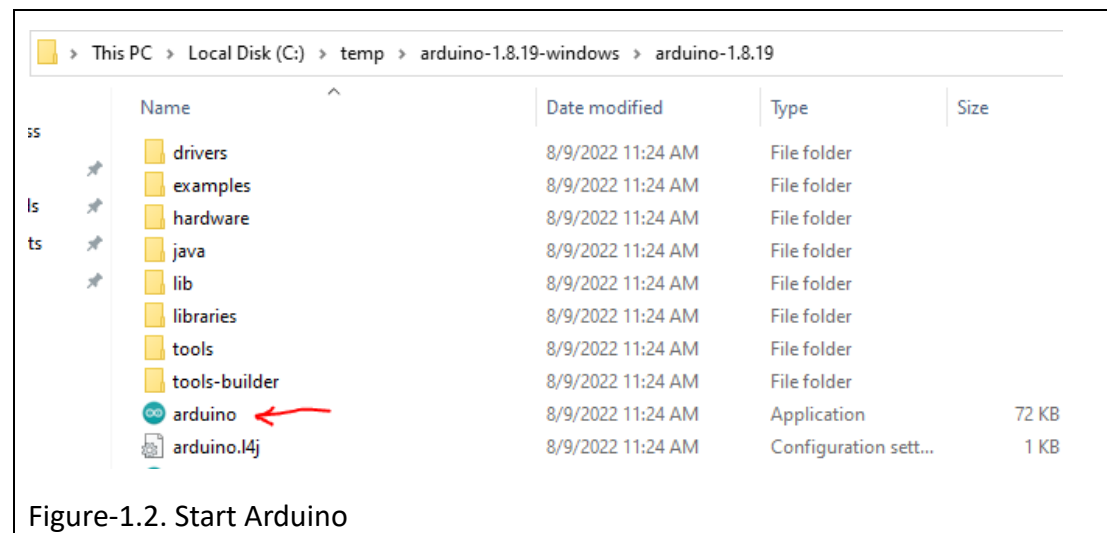
Other

JUST DOWNLOAD

CONTRIBUTE & DOWNLOAD

Figure-1.1 Download Arduino IDE

3. Unzipped the download file to a writable folder in your hard drive (e.g. desktop, c:\temp or thumb drive).
4. Start Arduino with 'arduino.exe' as in Figure-1.2. Figure-1.3 is an empty Arduino project. At the bottom line, it shows some information of your current settings of the IDE. **The setting information is very important for troubleshooting if one fails in compilation (sketch) or upload.**
5. The function calls named **setup()** and **loop()** at the upper of the program area (Figure-1.3) are different from traditional C-language program which starts at **main()**.
  - The program starts at **setup()**. It will run at once when being power up or RESET (EN button on ESP32 module is pressed). We will put configurations there.
  - **Loop()** is the main body of the program. When the program finishes the last line of the loop, it will go back to the top line of loop() and run again and loop infinitely.
  - Lines start with **//** are comments in C-language. They serve as the remarks for the programmers. Writing clear and sufficient comments in program is a good habit of being a professional software engineer.



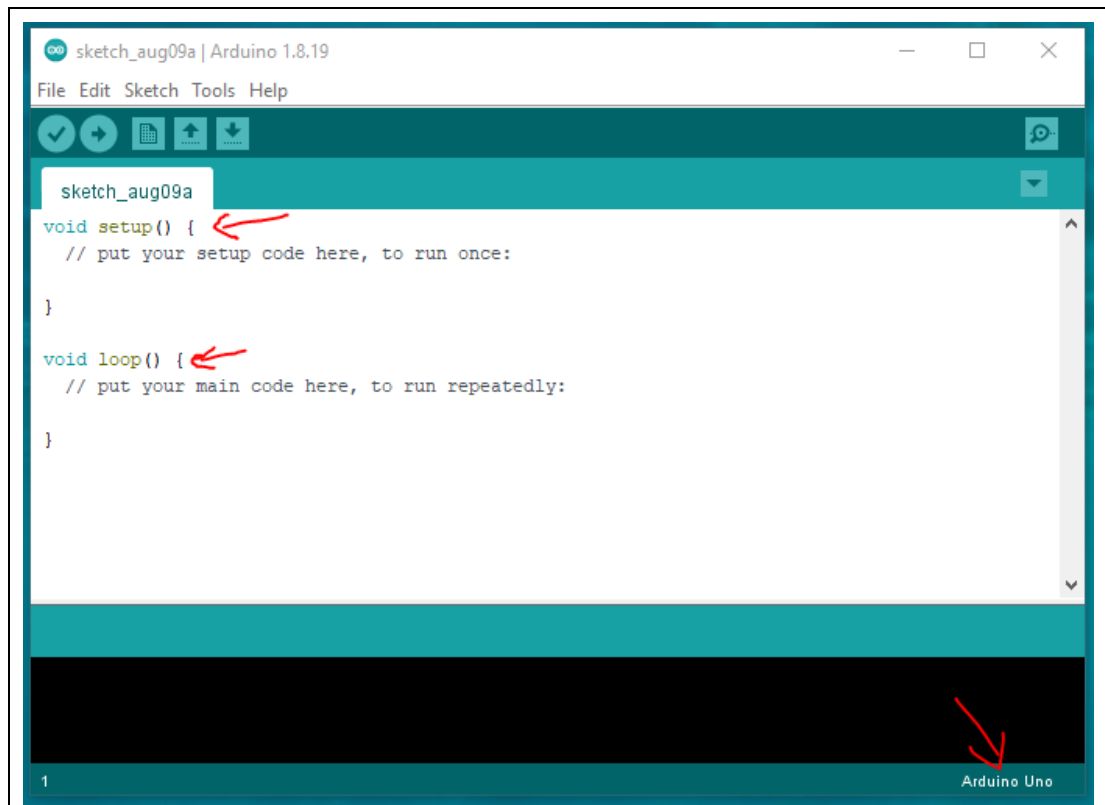


Figure-1.3. An Arduino empty project.

6. Arduino IDE is designed for another microprocessors (ATmega). You have to download some plug-ins (libraries) for ESP32. In Arduino IDE, select 'File' 'Preferences' (Figure-1.4).
7. In the text box **Additional Boards Manager URLs**, fill the path [https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package\\_esp32\\_index.json](https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json) of ESP32 json file as in Figure-1.5. Press OK to proceed to the next step.

Check out this page to know more information about installation

<https://docs.espressif.com/projects/arduino-esp32/en/latest/installing.html>

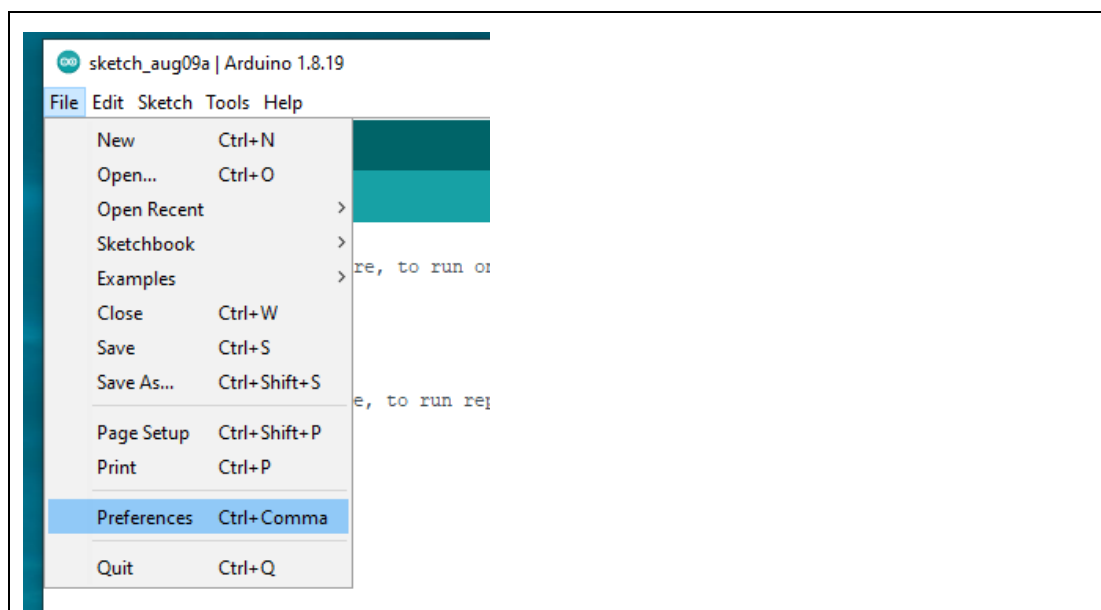


Figure-1.4. Select 'Preference'

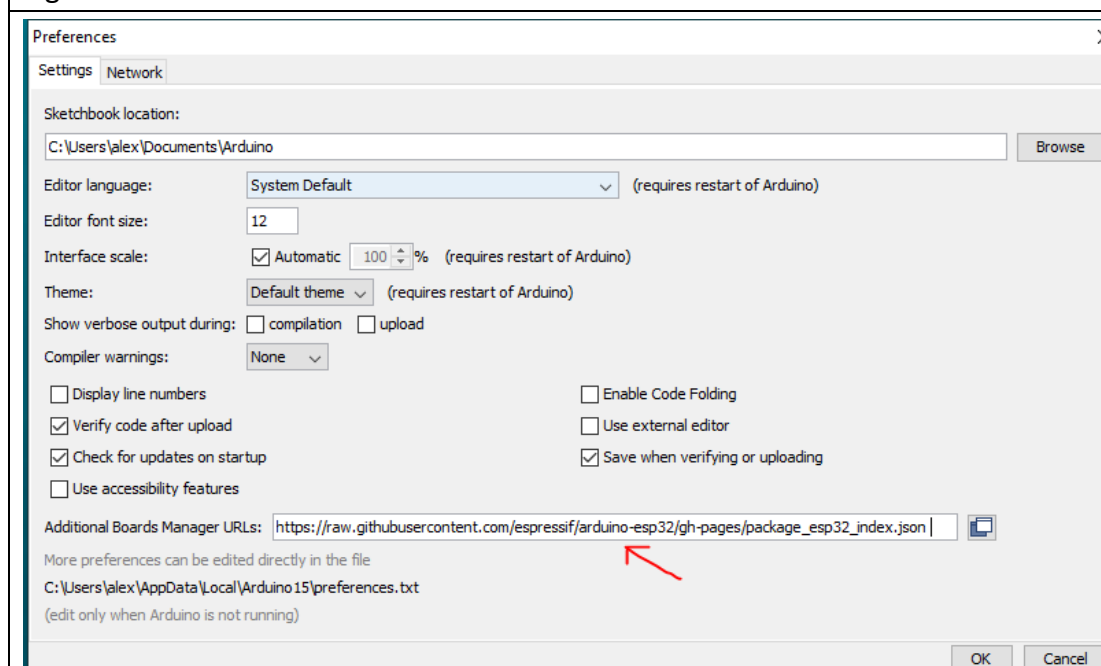


Figure-1.5. Fill the path of json file.

8. In Arduino IDE, select 'Tools' 'Board: xxx' 'BoardManager...' to change the board. (Figure-1.6)
9. In Board Manager, key in 'ESP32' to filter out other boards (top arrow on Figure-1.7). Press 'Install' to install the latest version drivers (bottom arrow on Figure-1.7). Wait for installation, it will take a few minutes. The percentage of the process will be shown at the bottom of Arduino IDE, as in Figure-1.8.
10. When the installation is completed, 'Installed' is shown, as in Figure-1.9. Then press the button 'Close' (at the bottom) to close board manager.

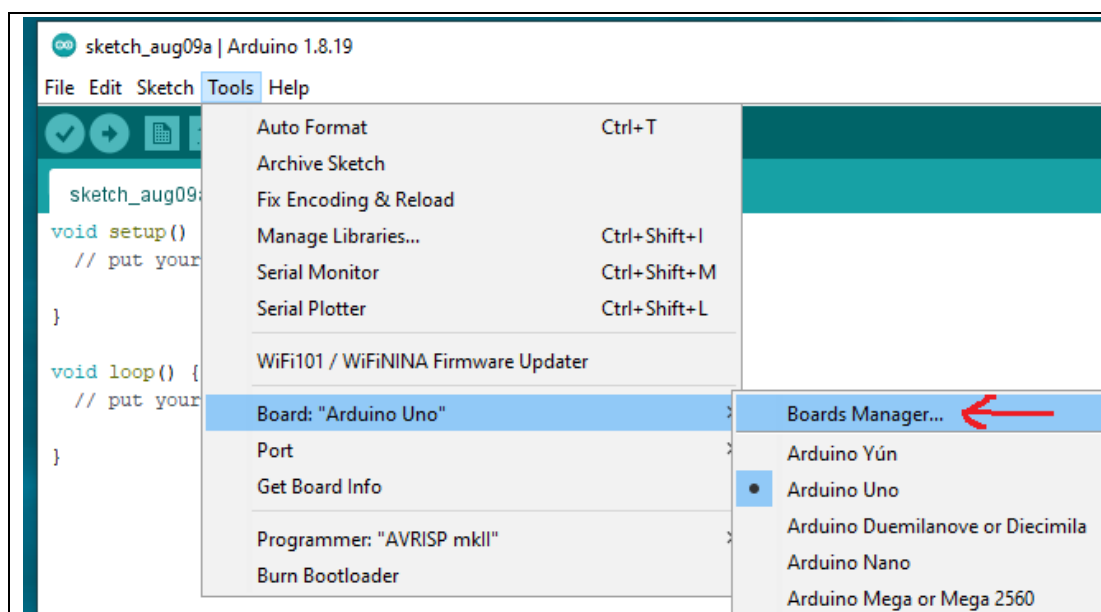


Figure-1.6. Select Boards Manager

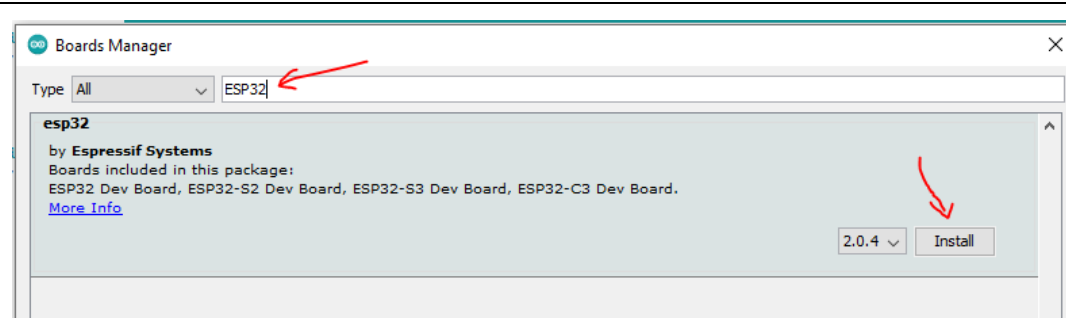


Figure-1.7. Filter Board information for ESP32

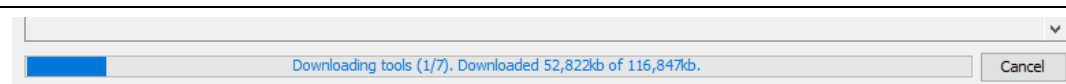


Figure-1.8. Process bar of board library download (At the bottom of Arduino IDE)

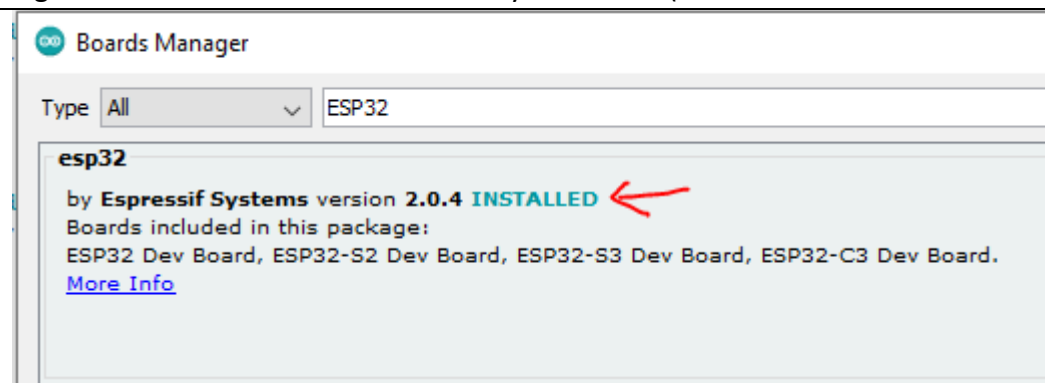


Figure-1.9. Board install completed.

11. Select the corresponding board that we use. In Arduino IDE, select 'Tools' → 'Board: xxx' → 'ESP32 Arduino' → 'ESP32 Dev Module' (Figure-1.10). The Board Information (at the bottom of Arduino IDE) will be changed to ESP32 Dev Module (Figure-1.11).

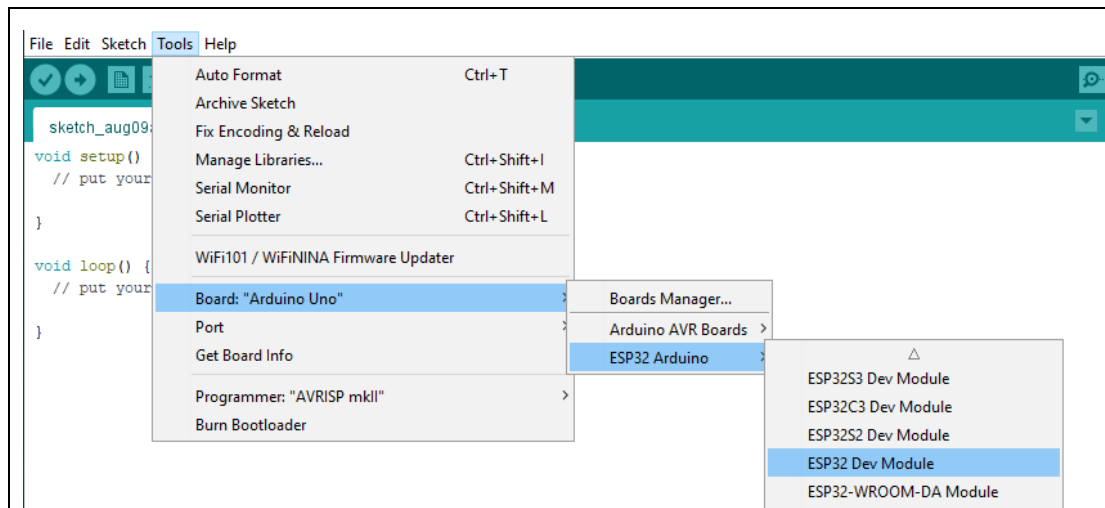


Figure-1.10



Figure-1.11 Board Information is changed.

12. Turn on Windows' device manager with Windows search box (Figure-1.12).
13. As in Figure-1.13, explore **Port (COM & LPT)** to check the communication interface attached.

**Note:** COM and LPT are notations for legacy communication interfaces.

- COM = communication port (serial),
- LPT= line printer terminal (parallel).



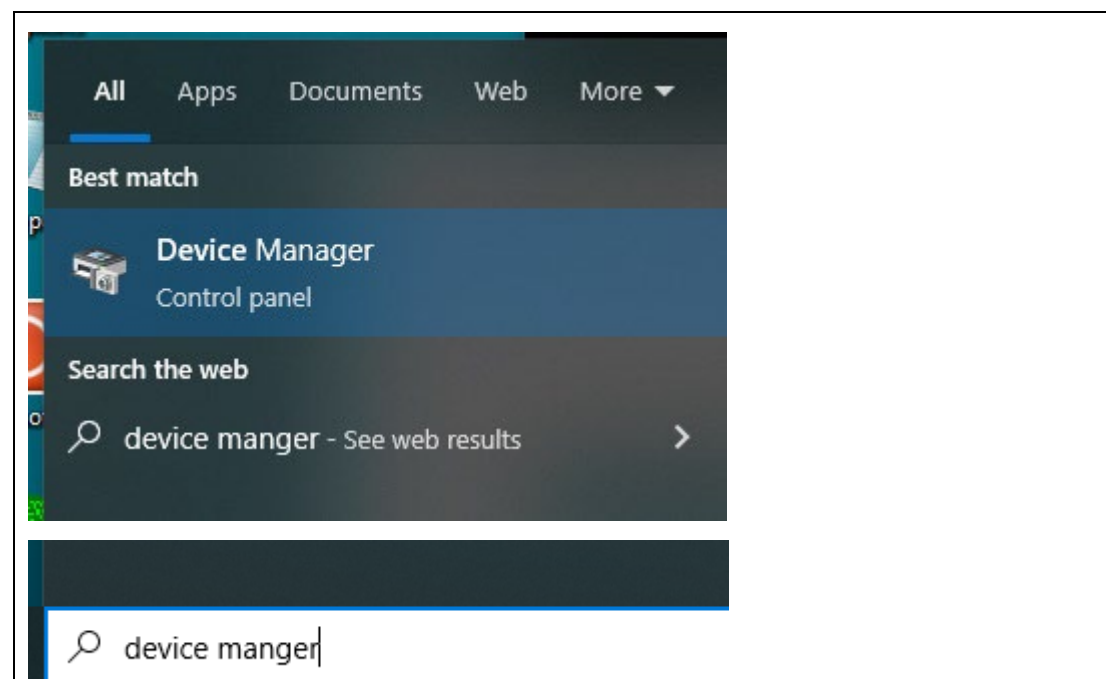


Figure-1.12

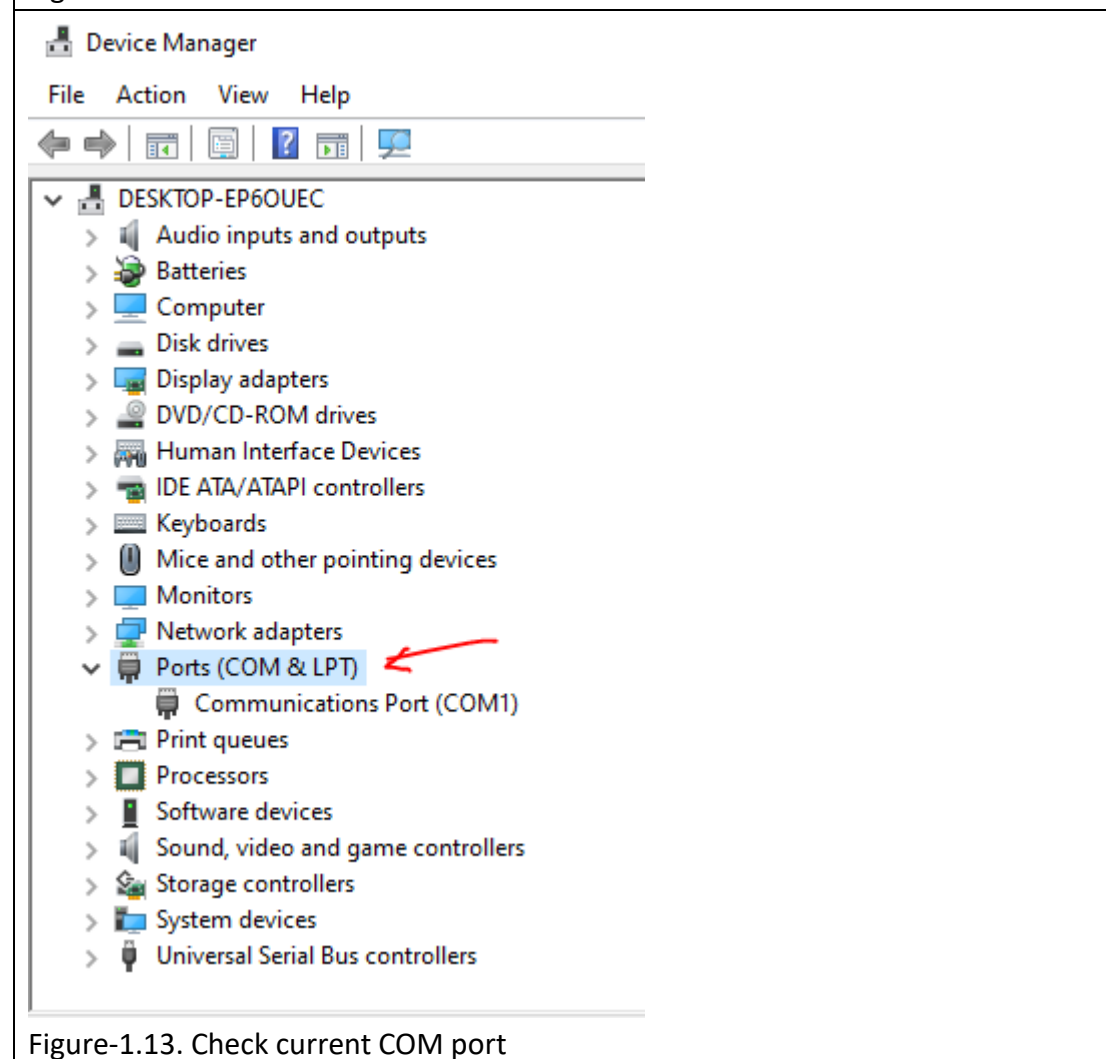
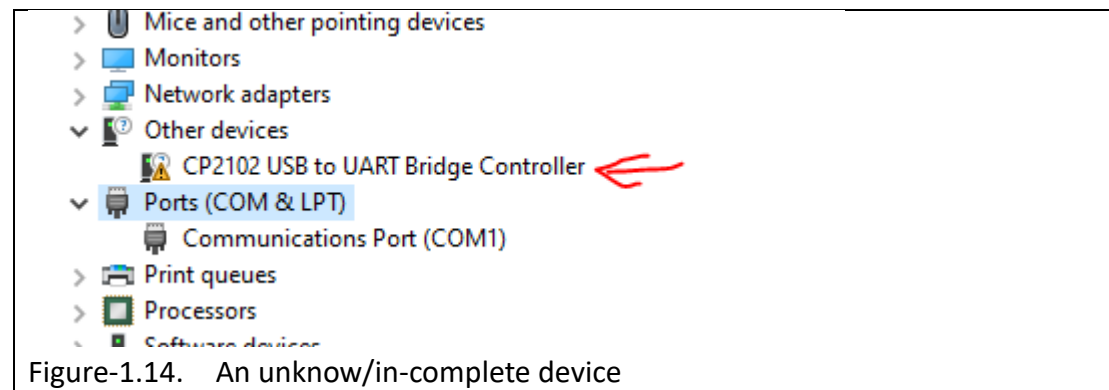


Figure-1.13. Check current COM port

14. This step is **important**. Connect the ESP32 to your computer with the provided micro USB cable. A new device (USB-to-Serial chip) will be discovered. If your computer knows (has its driver) this new device, a new COM port will be added. In the other words, exclamation mark on the device means your computer doesn't know about this new device, thus a device driver is needed (Figure-1.14).

**Notes:** Two types of possible USB-to-Serial chips may be on ESP32 board are CP2102(CP210x) and CH9102.



**Notes:** There are two methods to install the device driver.

- Execute the driver installation pack (.exe) file.
- Search the relative driver files and update by Windows OS.

We prefer the method Search and Update by Windows OS.

15. A set of CP2102 driver files is on course blackboard (software, UART\_driver.zip), download and explode it to your hard drive (e.g. c:\temp).

16. Right click the mouse on the unknow device and select 'Update driver' (Figure-1.15).

17. Select 'Browse my computer for drivers', because we have got the relative files in procedure-15 (Figure-1.16).

18. Search the drivers in particular folder (e.g. C:\temp) (Figure 1.17) and click 'Next'. Windows will search and install the drivers automatically. After installation, a message will be shown as Figure-1.18. Close the pop-up message.

19. Check with Device Manager, the exclamation marked device will change to normal as Figure-1.19. **Important.** Take down the COM number of this device (e.g. **COM3**).

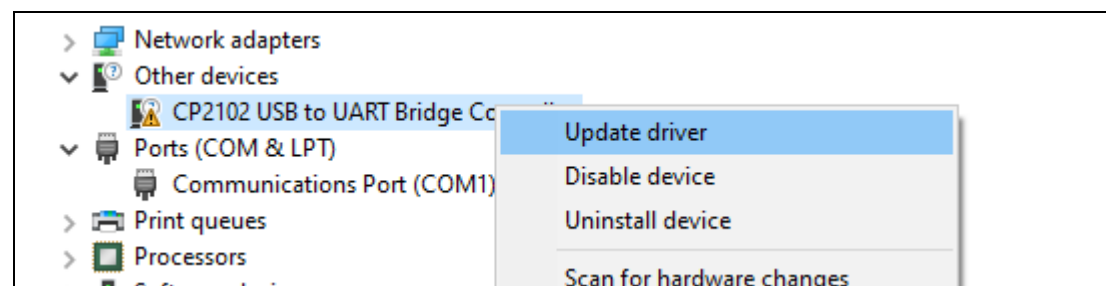


Figure-1.15. Update driver

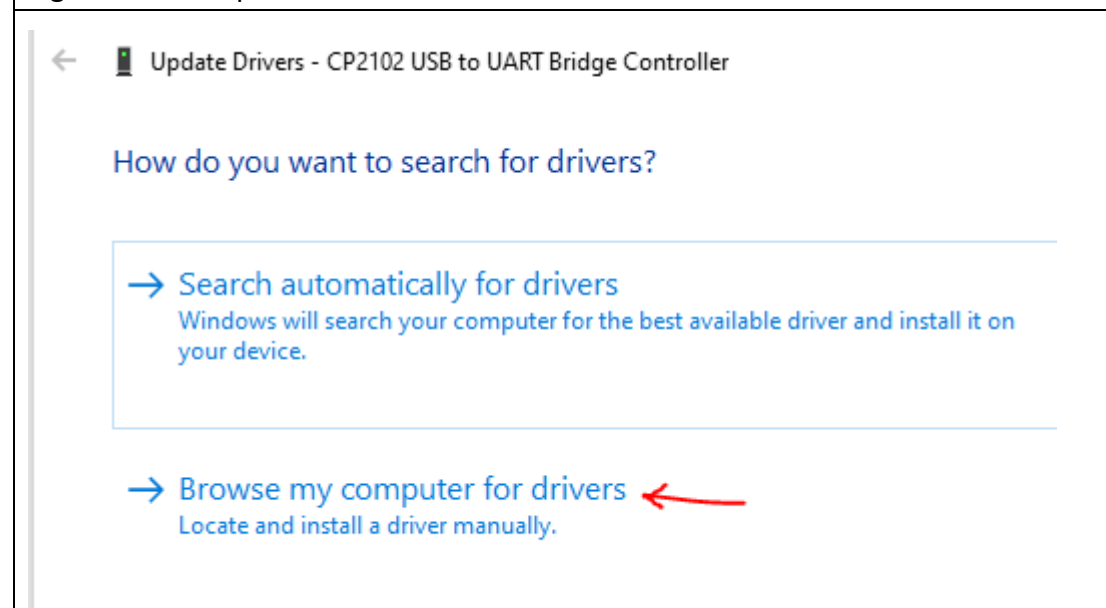


Figure-1.16. Search driver from your computer

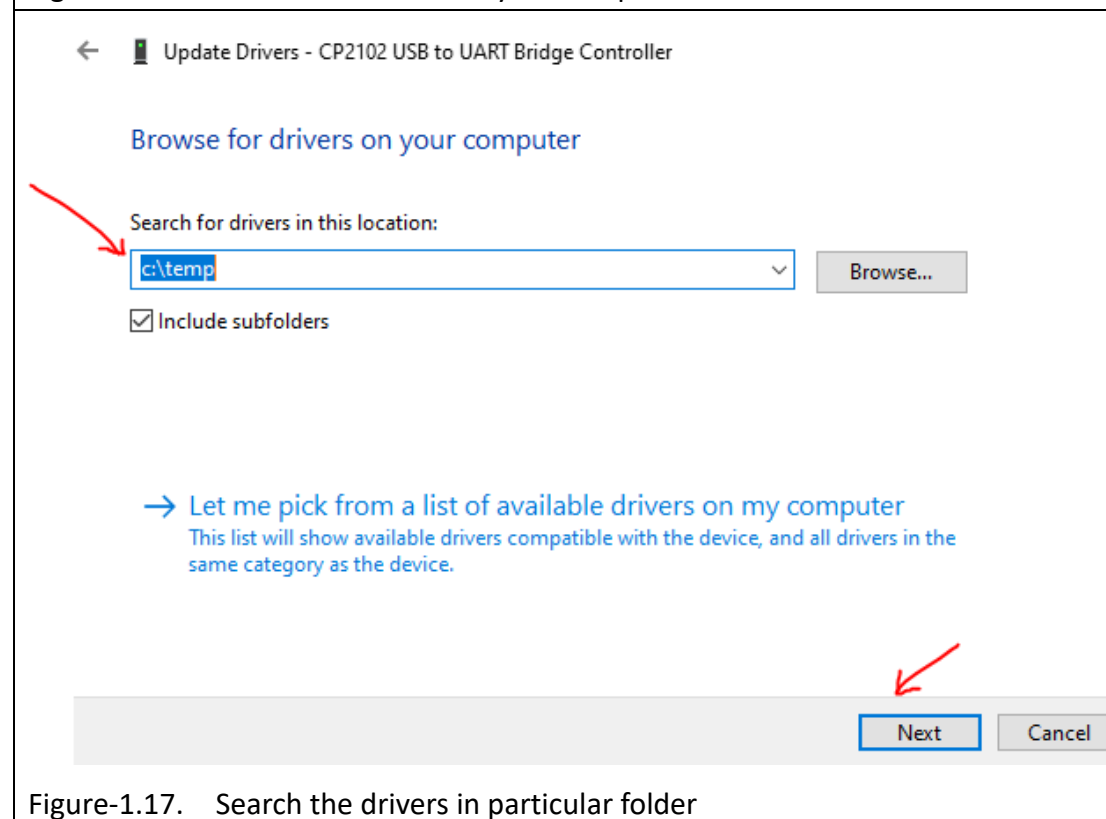


Figure-1.17. Search the drivers in particular folder

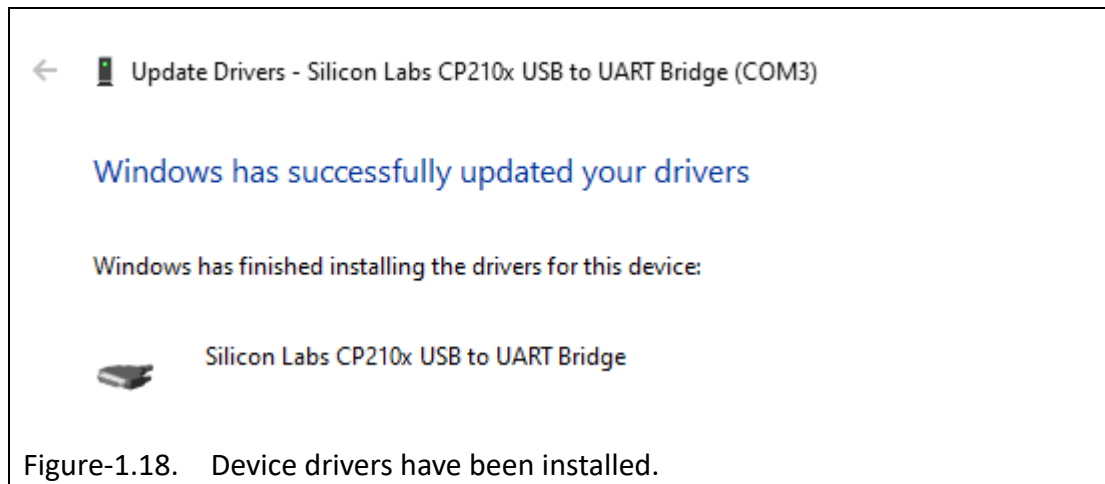


Figure-1.18. Device drivers have been installed.

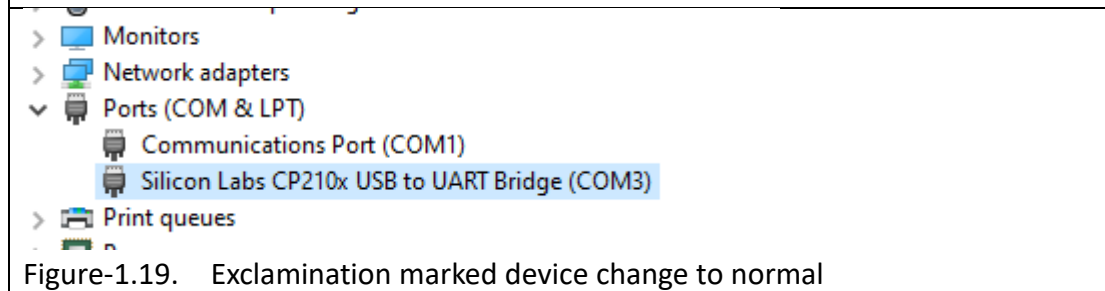


Figure-1.19. Exclamation mark device change to normal

20. After the Arduino IDE, ESP32 plug-in and USB-to-Serial driver installed. Select correct settings and test the environment. In Arduino IDE, select 'Tools' 'Port' 'COMx' (the port for the new UART bridge) (Figure-1.19 and Figure-1.20)
21. **Important.** Check the board information and COM information again at the bottom line of Arduino IDE. (Figure-1.21).
22. If everything is fine, we can test the environment. In Arduino IDE, click the arrow (Figure-1.22) to start sketch (compile) and upload the executable code to ESP32 module. At the bottom of Arduino IDE, it shows the process;
  - 'Compiling sketch' (Figure-1.23).
  - 'Uploading' (Figure-1.24)
  - 'Done Uploading' (Figure-1.25)

**Important:** If 'Done Uploading' is shown, it means that the environment is fine. This is an empty project, so you cannot find any different on ESP32 module.

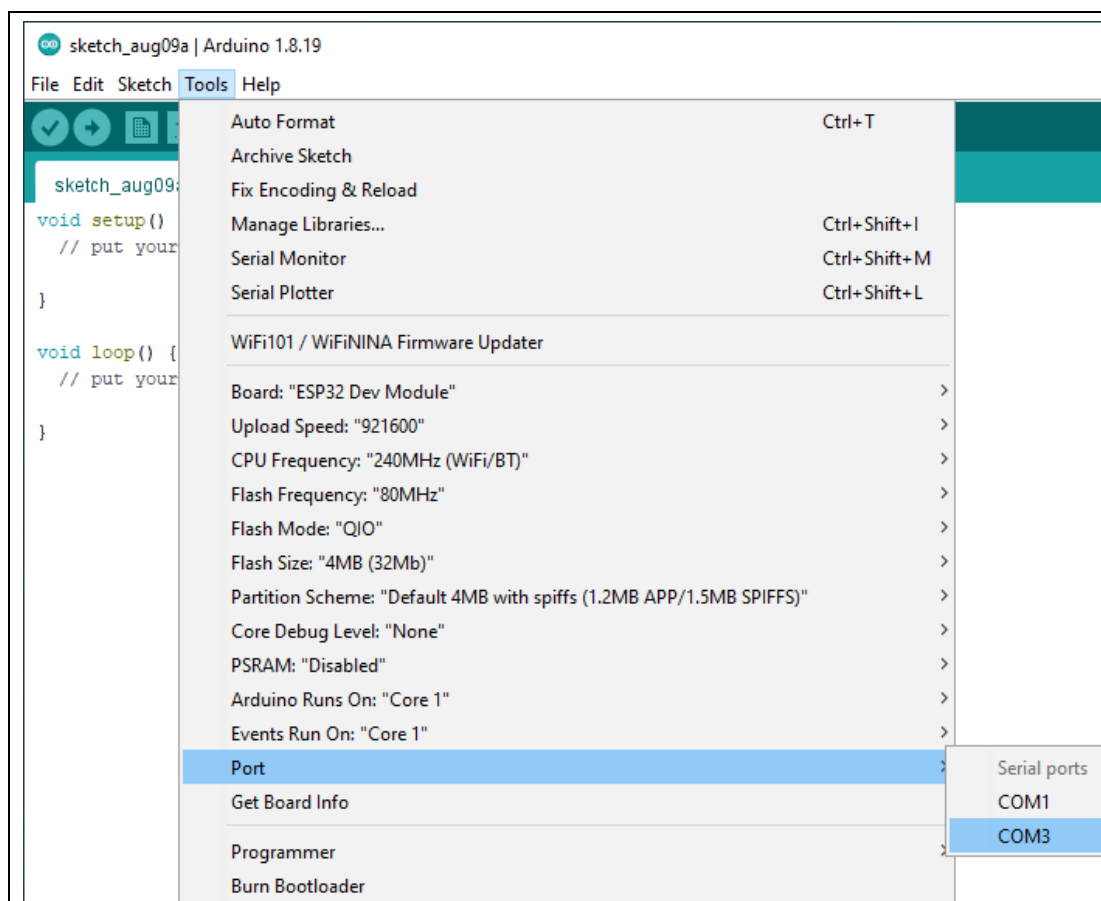


Figure-1.20. Select correct COM port



Figure-1.21. Check board and COM information

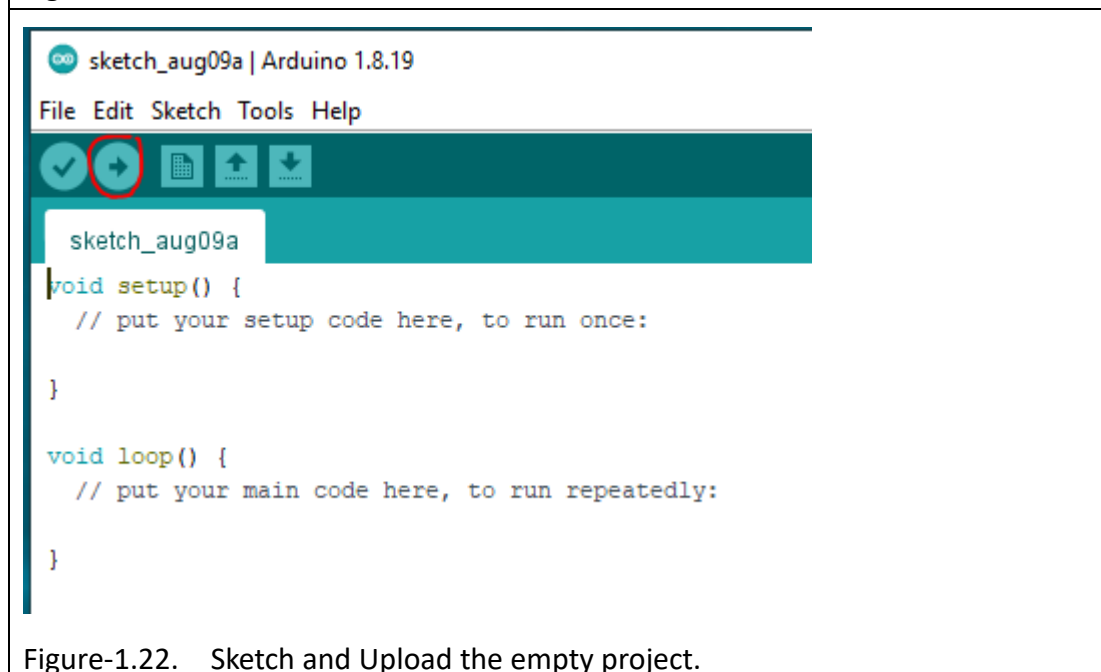


Figure-1.22. Sketch and Upload the empty project.

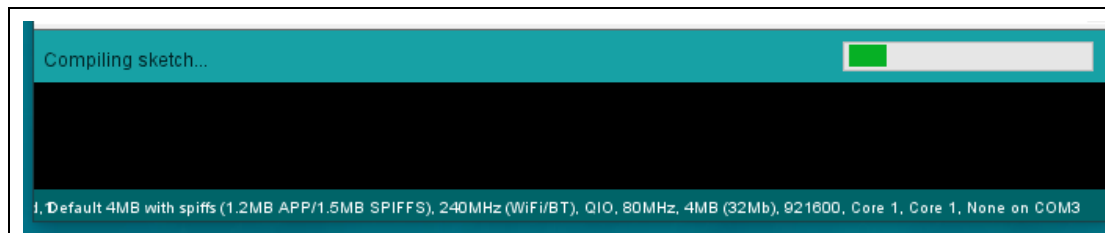


Figure-1.23 Compiling

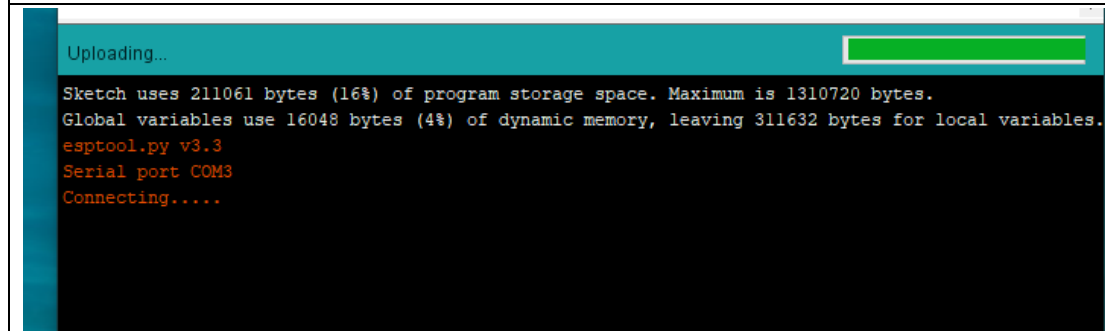


Figure-1.24. Uploading

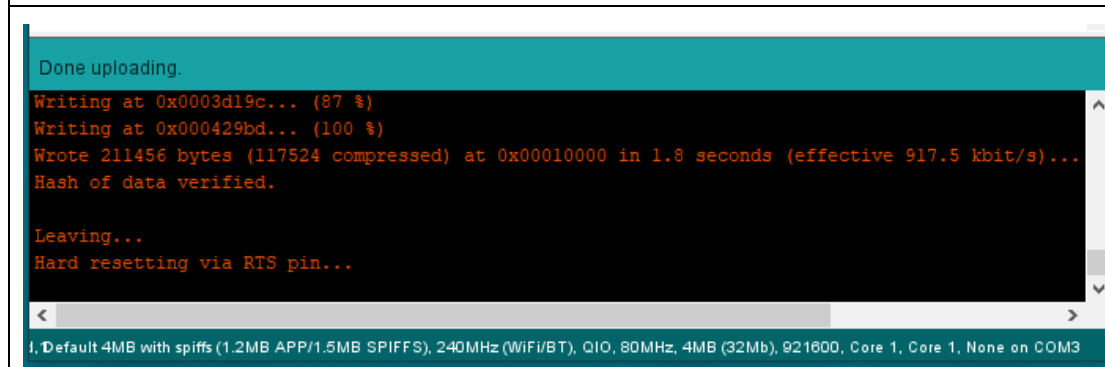


Figure-1.25. Done uploading

## Experiment 2.2: Flash the on-board LED

When the Arduino environment is ready, we can start programming. As a newbie we start from Arduino official examples and just do some modifications.

### Procedures:

1. In Arduino IDE, select 'File' 'Examples' '01.Basics' 'Blink' to start an official example. (Figure-2.1)
2. If you want to modify the official example, you have to save it as your own project. Select 'File' 'Save AS...' to save the project to a writable folder (e.g. C:\temp) (Figure-2.2). You can save it with the same or other name.

**Important:** A new folder will be generated as the project name (Figure-2.4). Do not modify the folder name, the Arduino project (\*.ino) must under the folder with the same name.

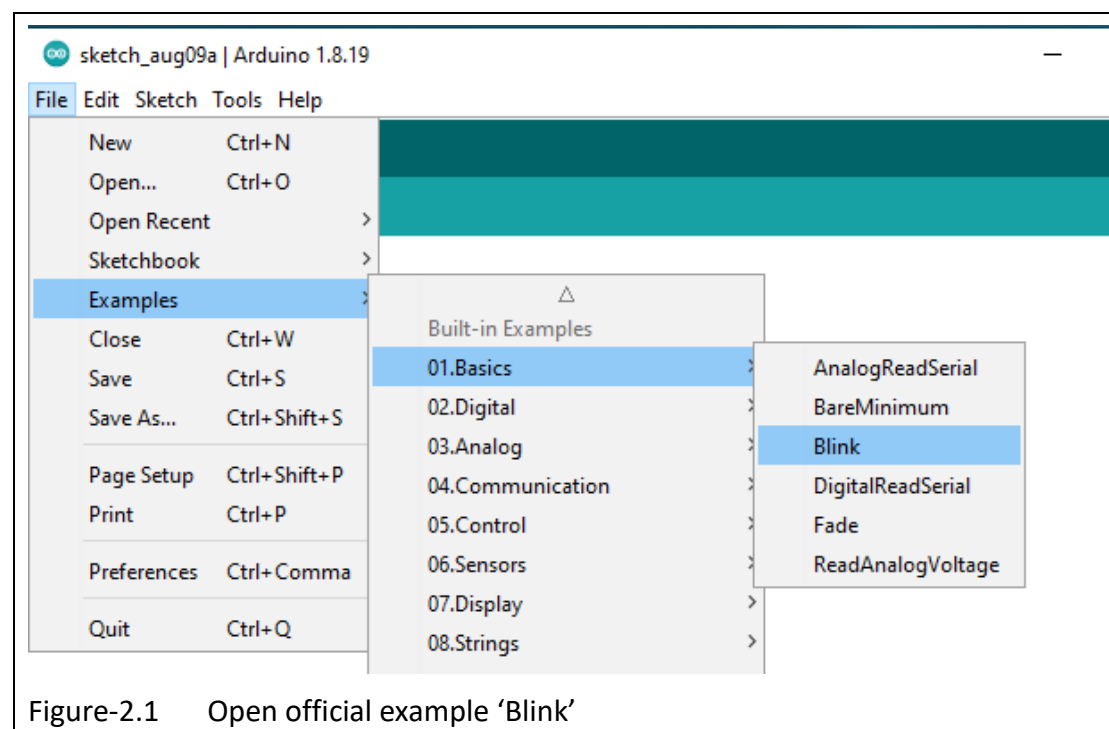


Figure-2.1 Open official example 'Blink'

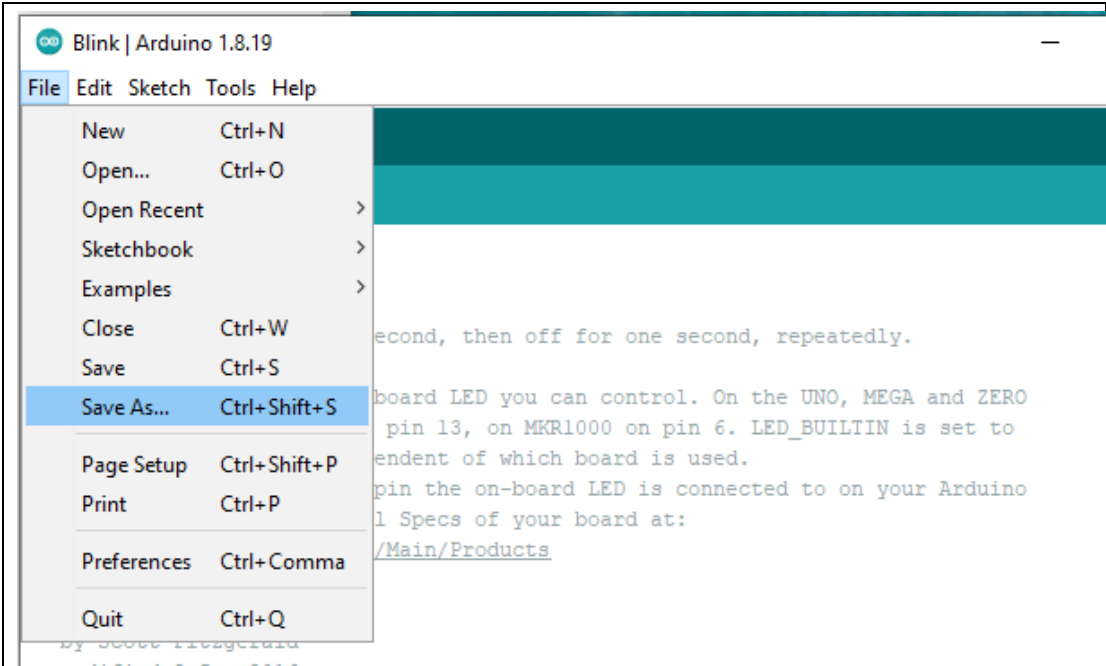


Figure-2.2 Save as a new project

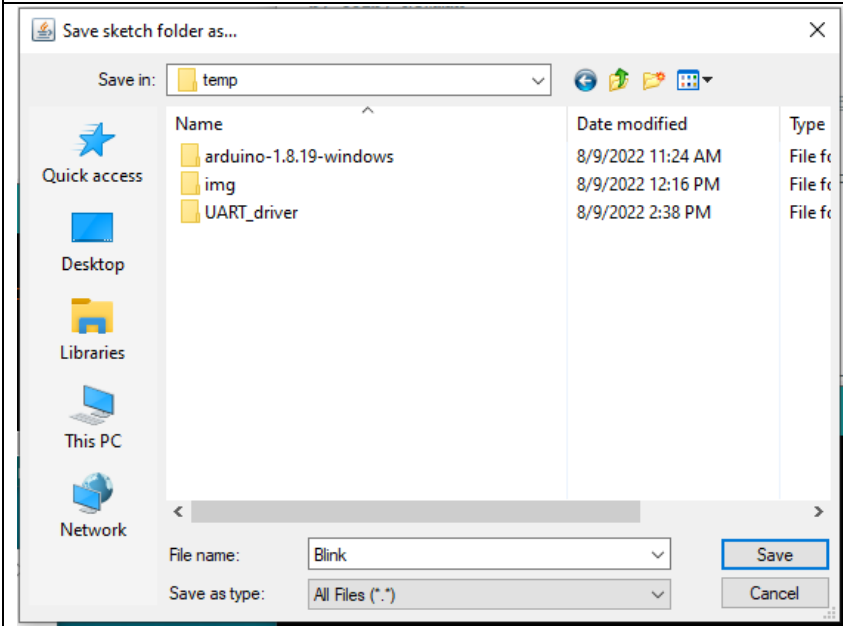


Figure-2.3 Save the project to a writable folder.



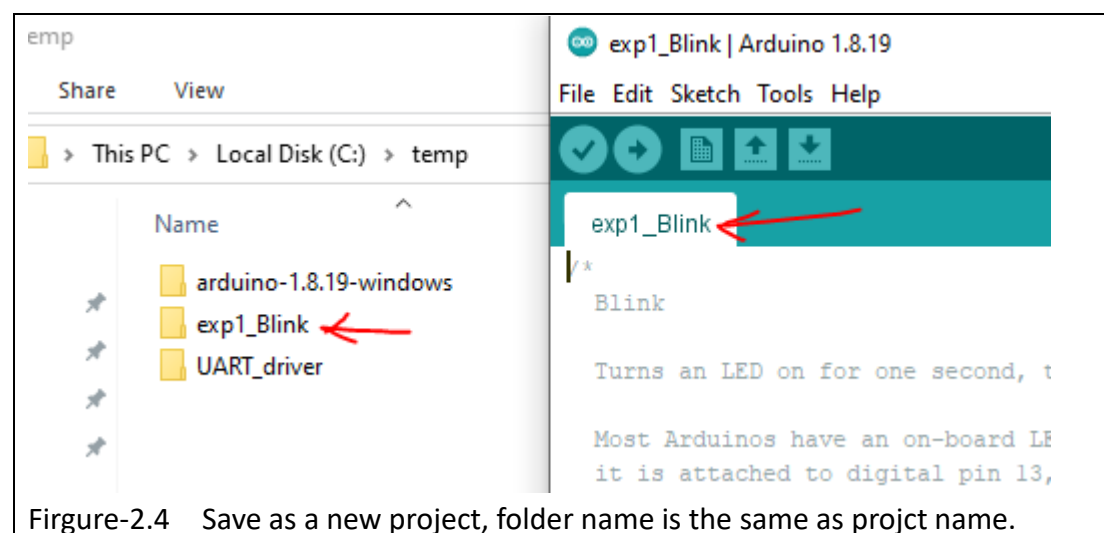


Figure-2.4 Save as a new project, folder name is the same as project name.

3. Try to understand the program of 'Blink' (Figure-2.5). Find the portion `setup()` and `loop()`.

In `setup()`,

- `pinMode( )`, declare a particular pin named 'LED\_BUILTIN' as an output pin.

In `loop()`,

- `digitalWrite( )`, set the particular pin (LED\_BUILTIN) to HIGH or LOW. (Digital = binary format = HIGH/LOW)
- `delay(1000)`, wait for 1000ms (1 second).
- after the end of loop (4<sup>th</sup> lines), the program will float back to the start of the loop as the arrow (Figure-2.5), so the project will run forever.
- **Important:** The semi-colons at the end of command lines (official called statement) mean the command lines end.

4. What is the particular pin 'LED\_BUILTIN'? This project 'Blink' is not designed for ESP32, so you have to declare what 'LED\_BUILTIN' is.

Add a line '#define' before `setup( )` to declare an alias (as Figure-2.6). The syntax of #define is '**#define A B**', all names 'A' will be replaced by 'B'. The build-in LED is connected to pin-2. So the line '`pinMode (LED_BUILTIN, OUTPUT)`' in `setup( )` is equal to '`pinMode(2, OUTPUT)`'.

**Notes:** No semi-colon at the end of #define.

5. This program is modified. You can compile it as procedure-22 of experiment-2.1, press the arrow. (Figure-1.22). After uploading, a LED (should be BLUE) on the ESP32 module will flash one-second ON and one-second OFF because `delay(1000)`.

6. Modify the lines `delay(1000)` to **`delay(100)`**, re-compile and upload again. Will the LED flash faster?

```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH); // turn the LED on (HIGH is the voltage level)
  delay(1000);                     // wait for a second
  digitalWrite(LED_BUILTIN, LOW);  // turn the LED off by making the voltage LOW
  delay(1000);                     // wait for a second
}
```




Figure-2.5 Program in 'Blink'

```
#define LED_BUILTIN 2 ←
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}
```




Figure-2.6 Add #define to declare LED\_BUILTIN is pin-2

**Experiment 2.3: Digital Input *digitalRead()* and variable**

The port to drive/control a LED is in OUTPUT mode. In this experiment, we try to port INPUT mode and store data to a variable.

In C-language, you have to declare the type of variables (the size of the container). Type **integer** (Int) can store value from -32768 to +32767 (16-bit). This page contains more information about Arduino's variables (<https://www.arduino.cc/referene/en>)

**Procedures:**

1. Open another official example 'Button'. Select 'File' 'Examples' '02.Ditigal' 'Button'. (Figure-3.1).
2. Store this official example to a writable folder (as procedure-2 of Experiment-2.2).
3. Try to understand the program.
  - Modify the pin assignment from Figure-3.2 to Figure-3.3.
    - Comment (use `//`) out the line with 'const .....'.
    - Add lines that use `#define` the same way as in Experiment-2.2.
    - The LED is connected to pin-2.
    - We will add a button that connects to pin-32.
  - A variable 'buttonState' is used and its type is **int**. (Figure-3.4)
  - Modify the pinMode of buttonPin from INPUT to **INPUT\_PULLUP** (Figure-3.5). We will build the circuit similar to that in this page (<https://www.arduino.cc/en/Tutorial/DigitalInputPullup>), the differences are
    - We use ESP32. (Figure-3.6 is Arduino UNO)
    - We use 2-pin tactile switch. (Figure-3.6 is 4-pin tactile swtich)
  - Now, let's go through the program.
    - `#define` declare the pin numbers of buttonPin and ledPin.
    - `Setup()` declare buttonPin is INPUT and ledPin is OUTPUT.
    - `Loop()`, **`digitalRead()`** reads the status of buttonPin and stores it to variable **buttonState**.
    - `If()` statements, check the buttonState, if it is HIGH then controls ledPin to HIGH and vice versa. About if-else statements, please read this page (<https://created.arduino.cc/projecthub/innovech/lesson-4-if-and-else-statements-79e43a>).
    - When it reaches the end of the program, it will go back to the start of `loop()`. So the program will run forever.

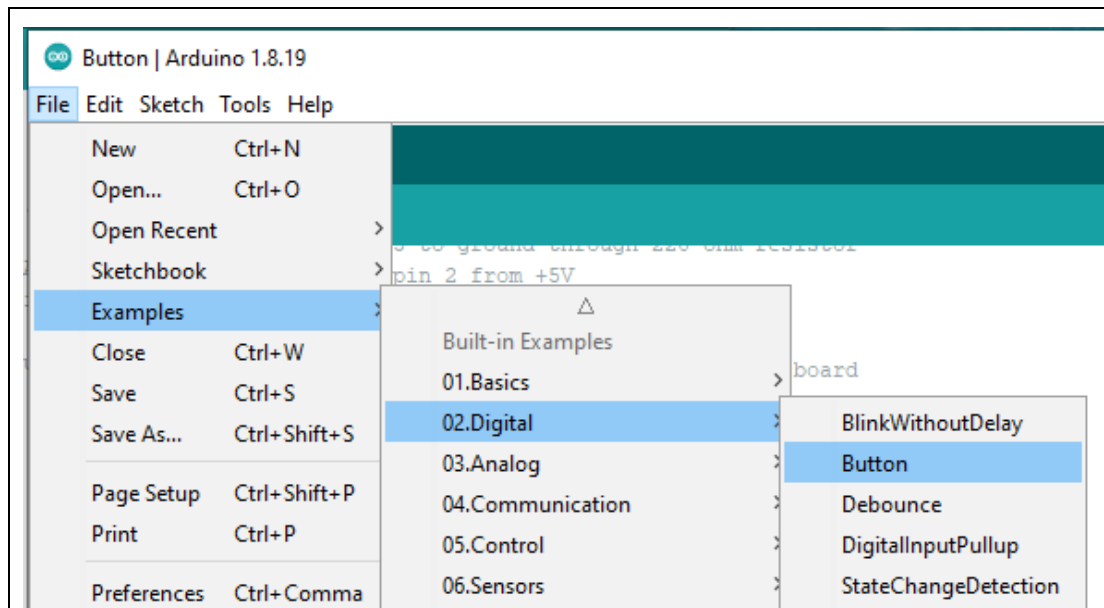


Figure-3.1. Open official example 'Button'

```
// constants won't change. They're used here to set pin numbers:
const int buttonPin = 2;    // the number of the pushbutton pin
const int ledPin = 13;     // the number of the LED pin
```

Figure-3.2 Original pin assignment

```
// constants won't change. They're used here to set pin numbers:
//const int buttonPin = 2;    // the number of the pushbutton pin
//const int ledPin = 13;     // the number of the LED pin
#define buttonPin 32  //-- use Experiment-2.2 #define
#define ledPin 2      //-- use Experiment-2.2 #define
```

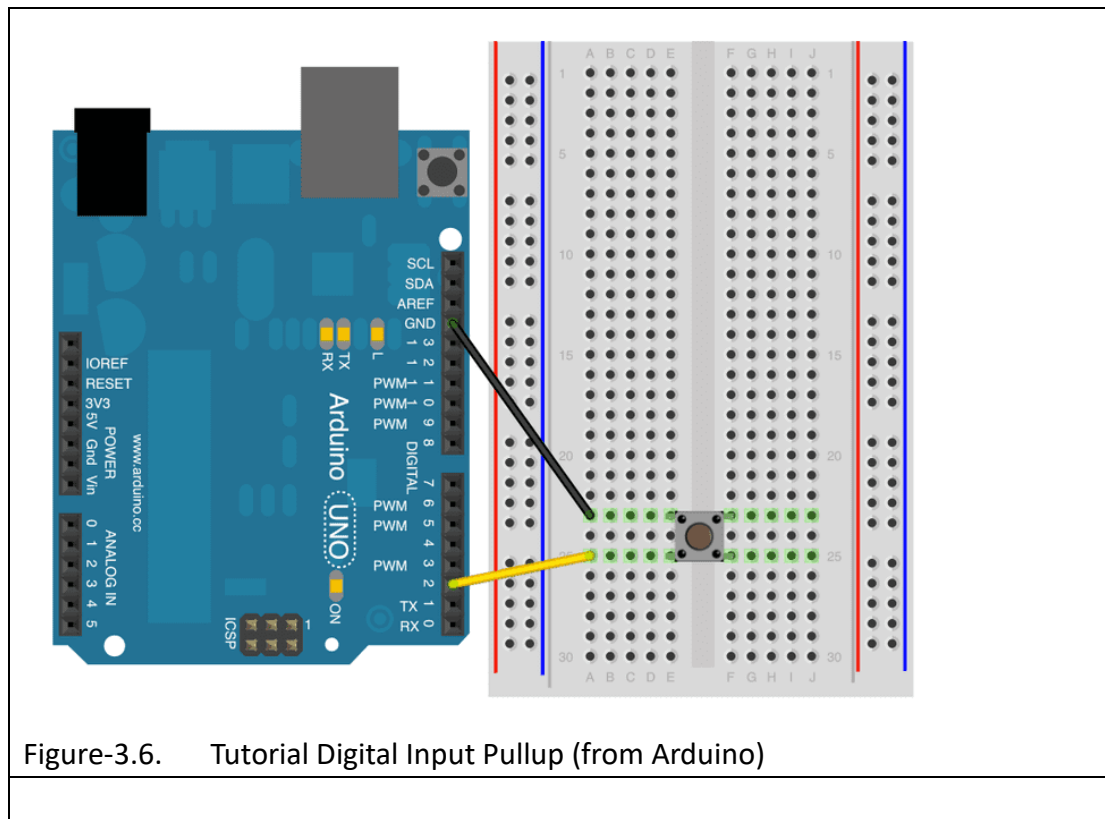
Figure-3.3 Modified pin assignment (use #define)

```
// variables will change:
int buttonState = 0;        // variable for reading the pushbutton status
```

Figure-3.4 Variable 'buttonState' is used and its type is int

```
void setup() {
  // initialize the LED pin as an output:
  pinMode(ledPin, OUTPUT);
  // initialize the pushbutton pin as an input:
  //pinMode(buttonPin, INPUT);
  pinMode(buttonPin, INPUT_PULLUP);
}
```

Figure-3.5 Modify 'buttonPin' pinMode from INPUT to INPUT\_PULLUP



4. Modify the hardware, add a 2-pin tactile switch(button) to ESP32-D32 and GND. Two terminals of tactile switch will be shorted together when you press the button, so two terminals are isolated normally, we called it the NORMAL-OPEN (NO). If the button is pressed, it means that ESP32-D32 will be shorted to GND (logic LOW at ESP32-D32). Use a jumper wire to connect ESP32-D32 (should be C10b-e) to the hole D10f. Insert the button to D10j and D8j. Add a wire from D8f to GND (refer Lab-1 using breadboard). The wiring should be similar to Figure-3.6 (Figure-3.6 shows Arduino UNO, do similar with ESP32).
5. Compile and upload the program to ESP32 (refer procedure-22 of Experiment-2.1).
6. Press the button, the LED ON/OFF depends on the button state (up/down). Take down the relationship of the LED and Button, present both (button and LED) with logic HIGH/LOW.

**Note:** Will not provide images and steps again if similar procedures are introduced before.

### Experiment 2.4: Debug tool (Serial Monitor)

It is very hard to know the value of variable and the situation of the program. **Serial Monitor** is a useful tool of Arduino IDE to show/print something on screen; show variables, milestone, etc. Serial Monitor communicates to your computer through ESP32 download port (UART-1), we will use 115200bps (default 9600bps).

#### Procedures:

1. Select 'Tools' 'Serial Monitor' to turn on Serial Monitor (Figure-4.1).
2. When the Serial Monitor pop-up, change the baud rate from 9600 to 115200. (Figure-4.2)
3. Modify Experiment-2.3 program with Serial communication.
  - I. Add `Serial.begin(115200)` into `setup( )` to initialize the serial port.
  - II. Add lines `Serial.print('H')` and `Serial.print('L')` into 'if' statements to report the status. For printing single character, use Single quote. For string (e.g. "HELLO") use double quote.
  - III. Add `delay(50)` at the end of the loop. Otherwise your computer will become slow motion due to the fact that serial port is too busy. `Delay(50)` will let the program do sampling per 50ms (1 second do 20 times).
4. Compile and upload your program. 'H' and 'L' will be shown on Serial Monitor continuously. Pressing the button and observe the result on Serial Monitor.

**Note:** Up to now, you should know and do how to

- Output HIGH/LOW to a pin;
- Control the on-board LED;
- Read digital state from a pin;
- Use variable to store data;
- Slow down the process time of the microprocessor;
- Print Character, String, variable to Serial Monitor.

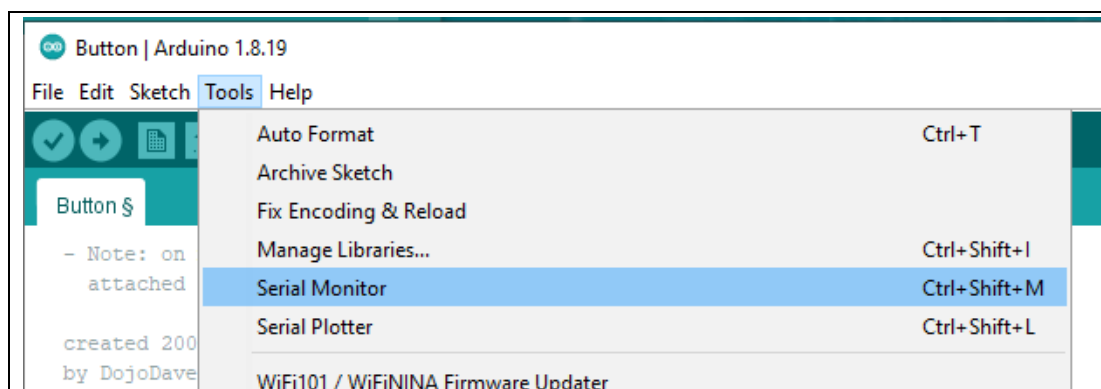


Figure-4.1 Use Serial Monitor

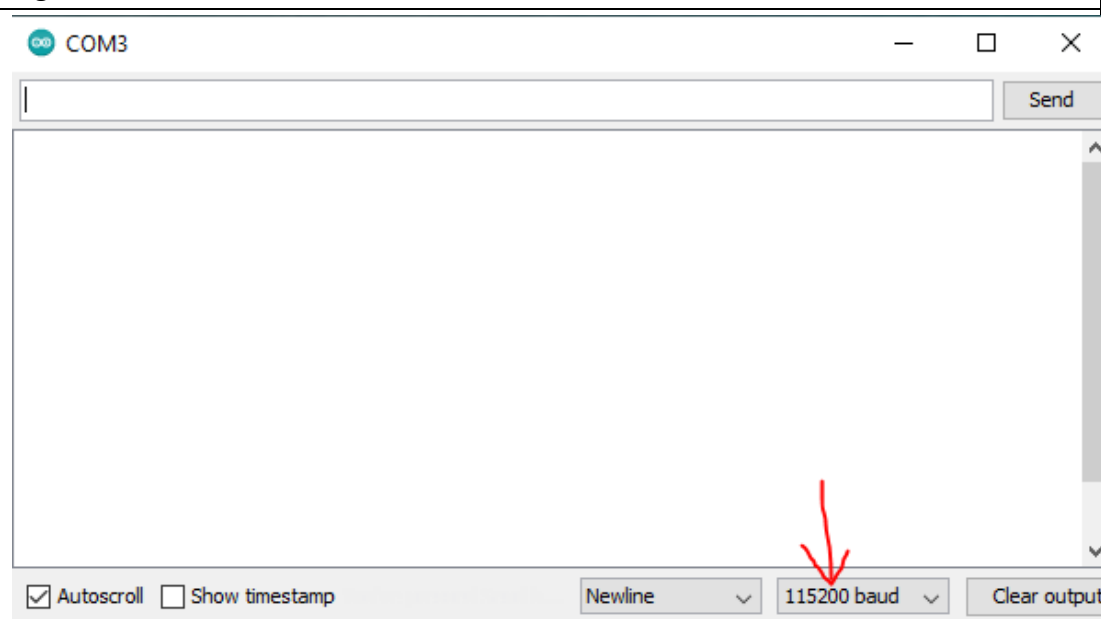


Figure-4.2 Change baudrate to 115200

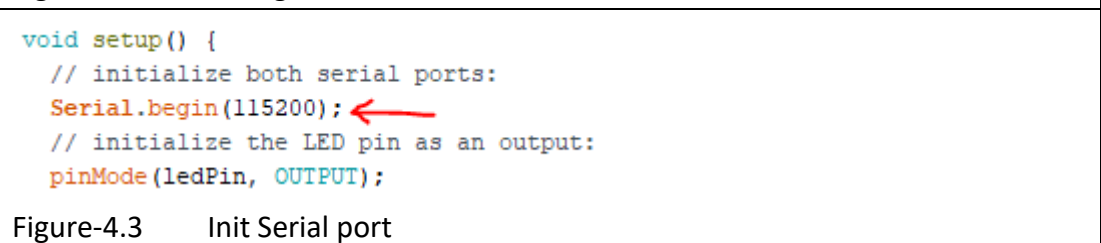


Figure-4.3 Init Serial port

```
void loop() {  
  // read the state of the pushbutton value:  
  buttonState = digitalRead(buttonPin);  
  
  // check if the pushbutton is pressed. If it is, the buttonState is HIGH:  
  if (buttonState == HIGH) {  
    // turn LED on:  
    digitalWrite(ledPin, HIGH);  
    Serial.print('H'); ←  
  } else {  
    // turn LED off:  
    digitalWrite(ledPin, LOW);  
    Serial.print('L'); ←  
  }  
  delay(50); ←  
}
```

Figure-4.4 Add Serial.print( ) into the program

~ END ~