Getting Started with Seeed Studio XIAO ESP32C3

Seeed Studio XIAO ESP32C3 is an IoT mini development board based on the Espressif **ESP32-C3** WiFi/Bluetooth dual-mode chip. ESP32-C3 is a **32-bit RISC-V CPU**, which includes an **FPU** (Floating Point Unit) for **32-bit single-precision arithmetic** with powerful computing power. It has excellent radio frequency performance, supporting **IEEE 802.11 b/g/n WiFi**, and **Bluetooth 5 (LE)** protocols. This board comes included with an external antenna to increase the signal strength for your wireless applications. It also has a **small and exquisite form-factor** combined with a **single-sided surface-mountable design**. It is equipped with rich interfaces and has **11 digital I/O** that can be used

as **PWM pins** and **4 analog I/O** that can be used as **ADC pins**. It supports four serial interfaces such as **UART, I2C, SPI and I2S**. There is also a small **reset button** and a **bootloader mode button** on the board. XIAO ESP32C3 is fully compatible with the Grove Shield for Seeeduino XIAO and Seeeduino XIAO Expansion board except for the Seeeduino XIAO Expansion board, the SWD spring contacts on the board will not be compatible.

With regard to the features highlighted above, XIAO ESP32C3 is positioned as a **high-performance, low-power, cost-effective IoT mini development board**, suitable for **low-power IoT applications and wireless wearable applications**.

This wiki will show you how you can quickly get started with XIAO ESP32C3!

Features¶

- Powerful CPU: ESP32-C3, 32bit RISC-V singlecore processor that operates at up to 160 MHz
- Complete WiFi subsystem: Complies with IEEE 802.11b/g/n protocol and supports Station mode, SoftAP mode, SoftAP + Station mode, and promiscuous mode

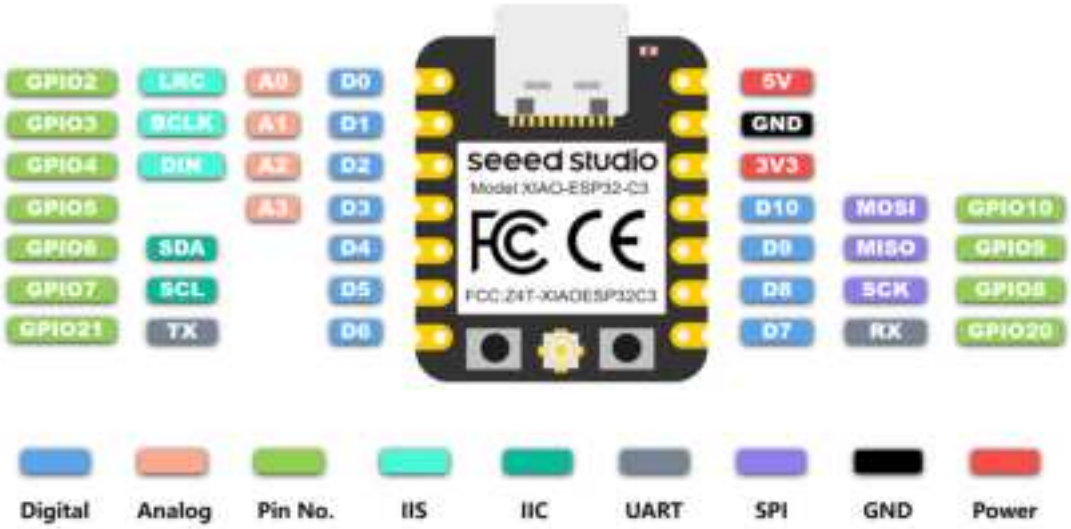
- Bluetooth LE subsystem: Supports features of Bluetooth 5 and Bluetooth mesh
- Ultra-Low Power: Deep sleep power consumption is about 43μA
- Better RF performance: External RF antenna included
- Battery charging chip: Supports lithium battery charge and discharge management
- Rich on-chip resources: 400KB of SRAM, and 4MB of on-board flash memory
- Ultra small size: As small as a thumb(20x17.5mm) XIAO series classic form-factor for wearable devices and small projects
- Reliable security features: Cryptographic hardware accelerators that support AES-128/256, Hash, RSA, HMAC, digital signature and secure boot
- Rich interfaces: 1xI2C, 1xSPI, 1xI2S, 2xUART, 11xGPIO(PWM), 4xADC, 1xJTAG bonding pad interface
- Single-sided components, surface mounting design

Specifications comparison¶

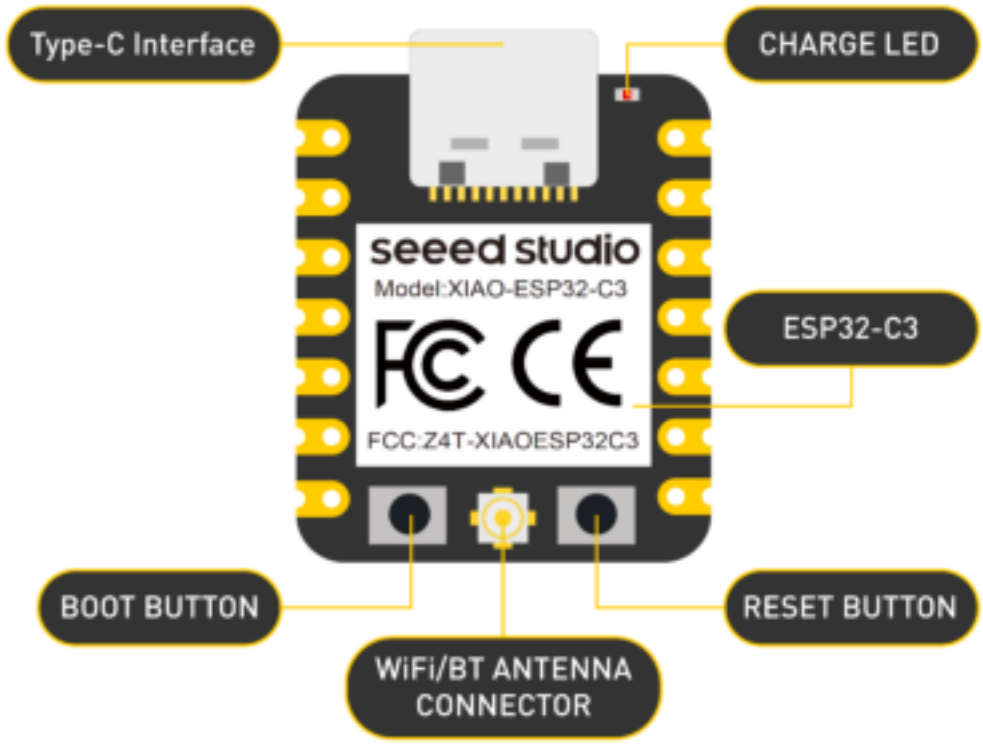
Processor	ESP32-C3 32-bit RISC-V @160MHz	SAMD21 M0+@48MHz	RP2040 Dual-core M0+@133Mhz	nRF52840 M4F@64MHz	nRF52840 M4F@64MHz
Wireless Connectivity	WiFi and Bluetooth 5 (LE)	N/A	N/A	Bluetooth 5.0/BLE/NFC	Bluetooth 5.0/BLE/NFC
Memory	400KB SRAM, 4MB onboard Flash	32KB SRAM 256KB FLASH	264KB SRAM 2MB onboard Flash	256KB RAM, 1MB Flash 2MB onboard Flash	256KB RAM,1MB Flash 2MB onboard Flash
Built-in Sensors	N/A	N/A	N/A	N/A	6 DOF IMU (LSM6DS3TR-C), PDM Microphone
Interfaces	I2C/UART/SPI/I2S	I2C/UART/SPI	I2C/UART/SPI	I2C/UART/SPI	I2C/UART/SPI
PWM/Analog Pins	11/4	11/11	11/4	11/6	11/6
Onboard Buttons	Reset/ Boot Button	N/A	Reset/ Boot Button	Reset Button	Reset Button
Onboard LEDs	Charge LED	N/A	Full-color RGB/ 3-in-one LED	3-in-one LED/ Charge LED	3-in-one LED/ Charge LED
Battery Charge Chip	Built-in	N/A	N/A	BQ25101	BQ25101
Programming Languages	Arduino	Arduino/ CircuitPython	Arduino/ MicroPython/ CircuitPython	Arduino/ MicroPython/ CircuitPython	Arduino/ MicroPython/ CircuitPython

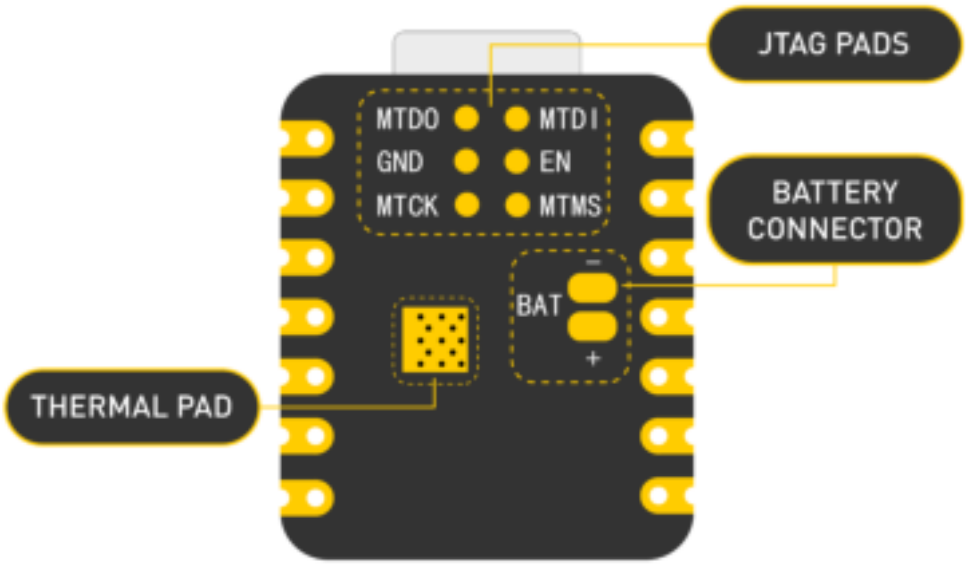
Hardware overview¶

Pinout diagram¶



Component overview¶





Power Pins¶

- 5V - This is 5v out from the USB port. You can also use this as a voltage input but you must have some sort of diode (schottky, signal, power) between your external power source and this pin with anode to battery, cathode to 5V pin.
- 3V3 - This is the regulated output from the onboard regulator. You can draw 700mA
- GND - Power/data/signal ground

Getting started¶

First, we are going to connect XIAO ESP32C3 to the computer, connect an LED to the board and upload a simple code from Arduino IDE to

check whether the board is functioning well by blinking the connected LED.

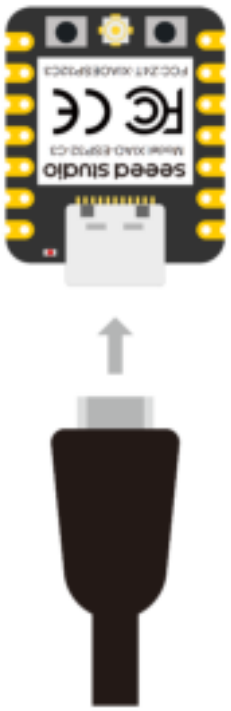
Hardware setup¶

You need to prepare the following:

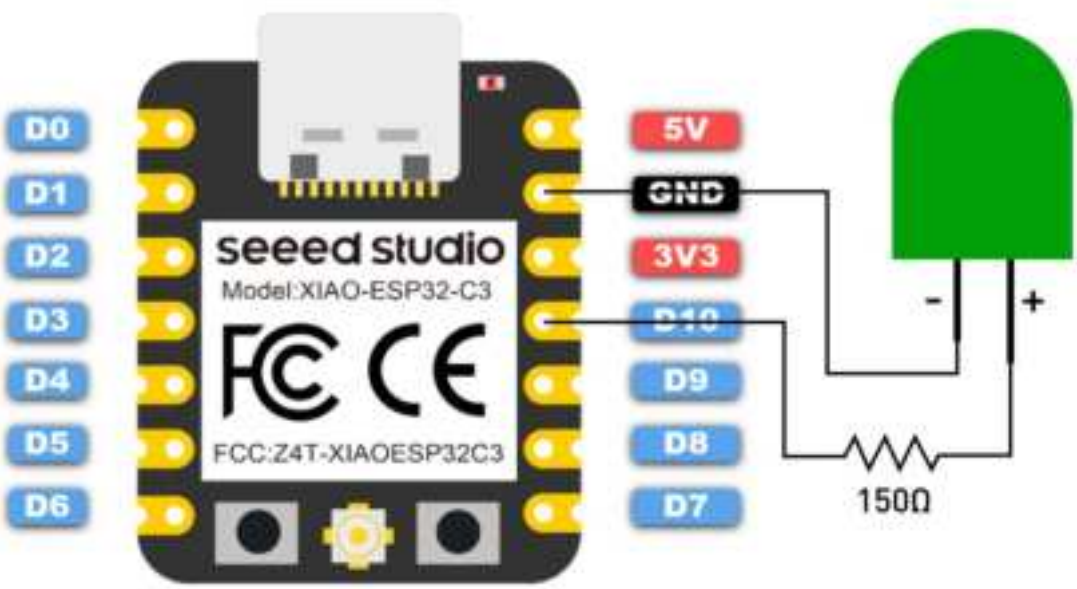
- 1 x Seeed Studio XIAO ESP32C3
- 1 x Computer
- 1 x USB Type-C cable

Tip
Some USB cables can only supply power and cannot transfer data. If you don't have a USB cable or don't know if your USB cable can transmit data, you can check Seeed USB Type-C support USB 3.1 .

- **Step 1.** Connect XIAO ESP32C3 to your computer via a USB Type-C cable.



- **Step 2.** Connect an LED to D10 pin as follows



Note: Make sure to connect a resistor (about 150Ω) in series to limit the current through the LED and to prevent excess current that can burn out the LED

Software setup¶

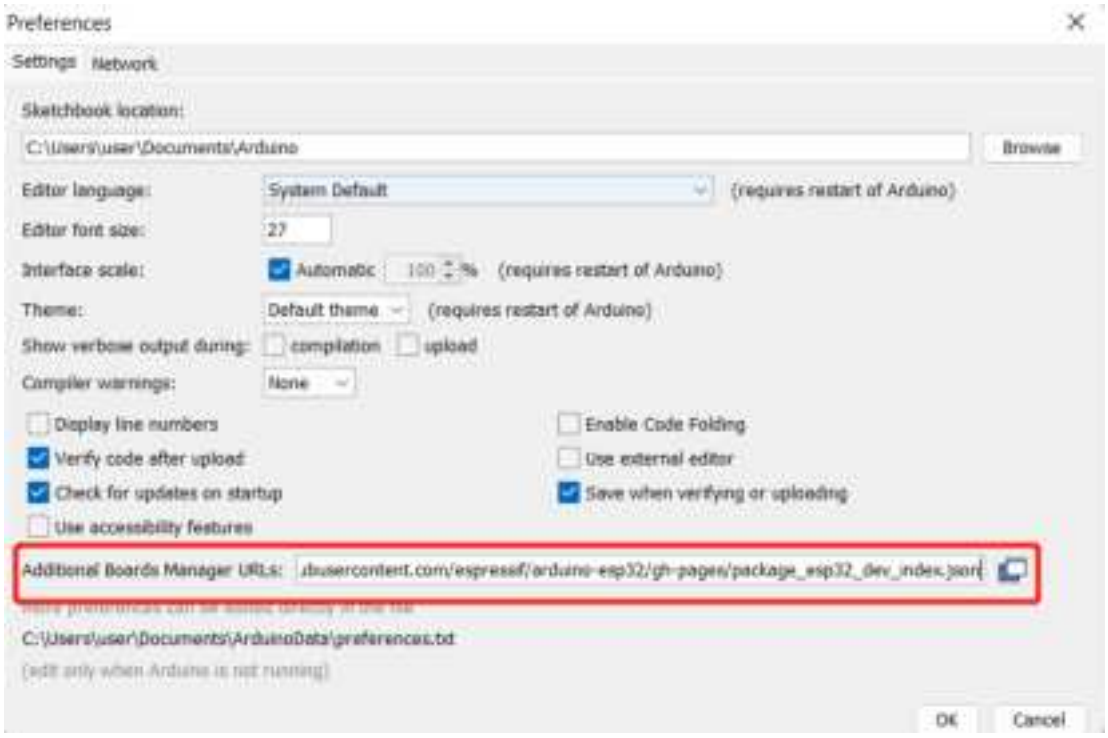
NOTE: Currently Seeed Studio XIAO ESP32C3 board is not available on Arduino Board Manager. The board has already been added to esp32 official board libraries but we need to wait until Arduino release the latest version of the board libraries. The current version is 2.0.3 and the version after this will have the XIAO ESP32C3 board library. Because of this, we will manually add the board library.

- **Step 1.** Download and Install the latest version of Arduino IDE according to your operating system

Download Arduino IDE

- **Step 2.** Launch the Arduino application
- **Step 3.** Add ESP32 board package to your Arduino IDE

Navigate to **File > Preferences**, and fill "**Additional Boards Manager URLs**" with the url below: *https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_dev_index.json*



Navigate to **Tools > Board > Boards Manager...**, type the keyword "**esp32**" in the search box, select the latest version of ****esp32****, and install it.

Bonus steps - add XIAO ESP32C3 manually (This step will be changed after XIAO ESP32C3 shows up on Arduino Board Manager)

Step 4: Navigate to `C:\Users\<username>\Documents\ArduinoData\packages\esp32\hardware\esp32\2.0.3\variants` and create a new folder named **XIAO_ESP32C3**

Step 5: Create a new file called **pins_arduino.h** and add the following content

```
#ifndef Pins_Arduino_h
#define Pins_Arduino_h
#include <stdint.h>
#define EXTERNAL_NUM_INTERRUPTS 22
#define NUM_DIGITAL_PINS 22
#define NUM_ANALOG_INPUTS 6
#define analogInputToDigitalPin(p)
(((p)<NUM_ANALOG_INPUTS)?(analogChannelToDigitalPin(p)):-1)
#define digitalPinToInterrupt(p) (((p)<NUM_DIGITAL_PINS)?(p):-1)
```

```
#define digitalPinHasPWM(p) (p < EXTERNAL_NUM_INTERRUPTS)
static const uint8_t TX = 21;
static const uint8_t RX = 20;
static const uint8_t SDA = 6;
static const uint8_t SCL = 7;
static const uint8_t SS = 20;
static const uint8_t MOSI = 10;
static const uint8_t MISO = 9;
static const uint8_t SCK = 8;
static const uint8_t A0 = 2;
static const uint8_t A1 = 3;
static const uint8_t A2 = 4;
static const uint8_t A3 = 5;
static const uint8_t D0 = 2;
static const uint8_t D1 = 3;
static const uint8_t D2 = 4;
static const uint8_t D3 = 5;
static const uint8_t D4 = 6;
static const uint8_t D5 = 7;
static const uint8_t D6 = 21;
static const uint8_t D7 = 20;
static const uint8_t D8 = 8;
static const uint8_t D9 = 9;
static const uint8_t D10 = 10;
#endif /* Pins_Arduino_h */
```

Step 6: Navigate

to `C:\Users\<username>\Documents\ArduinoData\packages\esp32\hardware\esp32\2.0.3` and open **boards.txt**

Step 7: Add the following content at the end of the file

```
XIAO_ESP32C3.name=XIAO_ESP32C3
XIAO_ESP32C3.vid.0=0x2886
XIAO_ESP32C3.pid.0=0x0047
XIAO_ESP32C3.bootloader.tool=esptool_py
XIAO_ESP32C3.bootloader.tool.default=esptool_py
XIAO_ESP32C3.upload.tool=esptool_py
XIAO_ESP32C3.upload.tool.default=esptool_py
XIAO_ESP32C3.upload.tool.network=esp_ota
XIAO_ESP32C3.upload.maximum_size=1310720
XIAO_ESP32C3.upload.maximum_data_size=327680
```

```
XIAO_ESP32C3.upload.flags=
XIAO_ESP32C3.upload.extra_flags=
XIAO_ESP32C3.upload.use_1200bps_touch=false
XIAO_ESP32C3.upload.wait_for_upload_port=false
XIAO_ESP32C3.serial.disableDTR=false
XIAO_ESP32C3.serial.disableRTS=false
XIAO_ESP32C3.build.tarch=riscv32
XIAO_ESP32C3.build.target=esp
XIAO_ESP32C3.build.mcu=esp32c3
XIAO_ESP32C3.build.core=esp32
XIAO_ESP32C3.build.variant=XIAO_ESP32C3
XIAO_ESP32C3.build.board=XIAO_ESP32C3
XIAO_ESP32C3.build.bootloader_addr=0x0
XIAO_ESP32C3.build.cdc_on_boot=1
XIAO_ESP32C3.build.f_cpu=160000000L
XIAO_ESP32C3.build.flash_size=4MB
XIAO_ESP32C3.build.flash_freq=80m
XIAO_ESP32C3.build.flash_mode=qio
XIAO_ESP32C3.build.boot=qio
XIAO_ESP32C3.build.partitions=default
XIAO_ESP32C3.build.defines=
XIAO_ESP32C3.menu.CDCOnBoot.default=Enabled
XIAO_ESP32C3.menu.CDCOnBoot.default.build.cdc_on_boot=1
XIAO_ESP32C3.menu.CDCOnBoot.cdc=Disabled
XIAO_ESP32C3.menu.CDCOnBoot.cdc.build.cdc_on_boot=0
XIAO_ESP32C3.menu.PartitionScheme.default=Default 4MB with spiffs
(1.2MB APP/1.5MB SPIFFS)
XIAO_ESP32C3.menu.PartitionScheme.default.build.partitions=default
XIAO_ESP32C3.menu.PartitionScheme.defaultffat=Default 4MB with ffat
(1.2MB APP/1.5MB FATFS)
XIAO_ESP32C3.menu.PartitionScheme.defaultffat.build.partitions=defaul
t_ffat
XIAO_ESP32C3.menu.PartitionScheme.default_8MB=8M Flash (3MB APP/1.5MB
FAT)
XIAO_ESP32C3.menu.PartitionScheme.default_8MB.build.partitions=defaul
t_8MB
XIAO_ESP32C3.menu.PartitionScheme.default_8MB.upload.maximum_size=334
2336
XIAO_ESP32C3.menu.PartitionScheme.minimal=Minimal (1.3MB APP/700KB
SPIFFS)
XIAO_ESP32C3.menu.PartitionScheme.minimal.build.partitions=minimal
XIAO_ESP32C3.menu.PartitionScheme.no_ota=No OTA (2MB APP/2MB SPIFFS)
XIAO_ESP32C3.menu.PartitionScheme.no_ota.build.partitions=no_ota
XIAO_ESP32C3.menu.PartitionScheme.no_ota.upload.maximum_size=2097152
```

```
XIAO_ESP32C3.menu.PartitionScheme.noota_3g=No OTA (1MB APP/3MB SPIFFS)
XIAO_ESP32C3.menu.PartitionScheme.noota_3g.build.partitions=noota_3g
XIAO_ESP32C3.menu.PartitionScheme.noota_3g.upload.maximum_size=1048576
XIAO_ESP32C3.menu.PartitionScheme.noota_ffat=No OTA (2MB APP/2MB FATFS)
XIAO_ESP32C3.menu.PartitionScheme.noota_ffat.build.partitions=noota_ffat
XIAO_ESP32C3.menu.PartitionScheme.noota_ffat.upload.maximum_size=2097152
XIAO_ESP32C3.menu.PartitionScheme.noota_3gffat=No OTA (1MB APP/3MB FATFS)
XIAO_ESP32C3.menu.PartitionScheme.noota_3gffat.build.partitions=noota_3gffat
XIAO_ESP32C3.menu.PartitionScheme.noota_3gffat.upload.maximum_size=1048576
XIAO_ESP32C3.menu.PartitionScheme.huge_app=Huge APP (3MB No OTA/1MB SPIFFS)
XIAO_ESP32C3.menu.PartitionScheme.huge_app.build.partitions=huge_app
XIAO_ESP32C3.menu.PartitionScheme.huge_app.upload.maximum_size=3145728
XIAO_ESP32C3.menu.PartitionScheme.min_spiffs=Minimal SPIFFS (1.9MB APP with OTA/190KB SPIFFS)
XIAO_ESP32C3.menu.PartitionScheme.min_spiffs.build.partitions=min_spiffs
XIAO_ESP32C3.menu.PartitionScheme.min_spiffs.upload.maximum_size=1966080
XIAO_ESP32C3.menu.PartitionScheme.fatflash=16M Flash (2MB APP/12.5MB FAT)
XIAO_ESP32C3.menu.PartitionScheme.fatflash.build.partitions=ffat
XIAO_ESP32C3.menu.PartitionScheme.fatflash.upload.maximum_size=2097152
XIAO_ESP32C3.menu.PartitionScheme.app3M_fat9M_16MB=16M Flash (3MB APP/9MB FATFS)
XIAO_ESP32C3.menu.PartitionScheme.app3M_fat9M_16MB.build.partitions=app3M_fat9M_16MB
XIAO_ESP32C3.menu.PartitionScheme.app3M_fat9M_16MB.upload.maximum_size=3145728
XIAO_ESP32C3.menu.PartitionScheme.rainmaker=RainMaker
XIAO_ESP32C3.menu.PartitionScheme.rainmaker.build.partitions=rainmaker
XIAO_ESP32C3.menu.PartitionScheme.rainmaker.upload.maximum_size=3145728
```

```
XIAO_ESP32C3.menu.CPUFreq.160=160MHz (WiFi)
XIAO_ESP32C3.menu.CPUFreq.160.build.f_cpu=160000000L
XIAO_ESP32C3.menu.CPUFreq.80=80MHz (WiFi)
XIAO_ESP32C3.menu.CPUFreq.80.build.f_cpu=80000000L
XIAO_ESP32C3.menu.CPUFreq.40=40MHz
XIAO_ESP32C3.menu.CPUFreq.40.build.f_cpu=40000000L
XIAO_ESP32C3.menu.CPUFreq.20=20MHz
XIAO_ESP32C3.menu.CPUFreq.20.build.f_cpu=20000000L
XIAO_ESP32C3.menu.CPUFreq.10=10MHz
XIAO_ESP32C3.menu.CPUFreq.10.build.f_cpu=10000000L
XIAO_ESP32C3.menu.FlashMode.qio=QIO
XIAO_ESP32C3.menu.FlashMode.qio.build.flash_mode=dio
XIAO_ESP32C3.menu.FlashMode.qio.build.boot=qio
XIAO_ESP32C3.menu.FlashMode.dio=DIO
XIAO_ESP32C3.menu.FlashMode.dio.build.flash_mode=dio
XIAO_ESP32C3.menu.FlashMode.dio.build.boot=dio
XIAO_ESP32C3.menu.FlashMode.qout=QOUT
XIAO_ESP32C3.menu.FlashMode.qout.build.flash_mode=dout
XIAO_ESP32C3.menu.FlashMode.qout.build.boot=qout
XIAO_ESP32C3.menu.FlashMode.dout=DOUT
XIAO_ESP32C3.menu.FlashMode.dout.build.flash_mode=dout
XIAO_ESP32C3.menu.FlashFreq.80=80MHz
XIAO_ESP32C3.menu.FlashFreq.80.build.flash_freq=80m
XIAO_ESP32C3.menu.FlashFreq.40=40MHz
XIAO_ESP32C3.menu.FlashFreq.40.build.flash_freq=40m
XIAO_ESP32C3.menu.FlashSize.4M=4MB (32Mb)
XIAO_ESP32C3.menu.FlashSize.4M.build.flash_size=4MB
XIAO_ESP32C3.menu.FlashSize.8M=8MB (64Mb)
XIAO_ESP32C3.menu.FlashSize.8M.build.flash_size=8MB
XIAO_ESP32C3.menu.FlashSize.8M.build.partitions=default_8MB
XIAO_ESP32C3.menu.FlashSize.2M=2MB (16Mb)
XIAO_ESP32C3.menu.FlashSize.2M.build.flash_size=2MB
XIAO_ESP32C3.menu.FlashSize.2M.build.partitions=minimal
XIAO_ESP32C3.menu.FlashSize.16M=16MB (128Mb)
XIAO_ESP32C3.menu.FlashSize.16M.build.flash_size=16MB
XIAO_ESP32C3.menu.UploadSpeed.921600=921600
XIAO_ESP32C3.menu.UploadSpeed.921600.upload.speed=921600
XIAO_ESP32C3.menu.UploadSpeed.115200=115200
XIAO_ESP32C3.menu.UploadSpeed.115200.upload.speed=115200
XIAO_ESP32C3.menu.UploadSpeed.256000.windows=256000
XIAO_ESP32C3.menu.UploadSpeed.256000.upload.speed=256000
XIAO_ESP32C3.menu.UploadSpeed.230400.windows.upload.speed=256000
XIAO_ESP32C3.menu.UploadSpeed.230400=230400
XIAO_ESP32C3.menu.UploadSpeed.230400.upload.speed=230400
```

```
XIAO_ESP32C3.menu.UploadSpeed.460800.linux=460800
XIAO_ESP32C3.menu.UploadSpeed.460800.macosx=460800
XIAO_ESP32C3.menu.UploadSpeed.460800.upload.speed=460800
XIAO_ESP32C3.menu.UploadSpeed.512000.windows=512000
XIAO_ESP32C3.menu.UploadSpeed.512000.upload.speed=512000
XIAO_ESP32C3.menu.DebugLevel.none=None
XIAO_ESP32C3.menu.DebugLevel.none.build.code_debug=0
XIAO_ESP32C3.menu.DebugLevel.error=Error
XIAO_ESP32C3.menu.DebugLevel.error.build.code_debug=1
XIAO_ESP32C3.menu.DebugLevel.warn=Warn
XIAO_ESP32C3.menu.DebugLevel.warn.build.code_debug=2
XIAO_ESP32C3.menu.DebugLevel.info=Info
XIAO_ESP32C3.menu.DebugLevel.info.build.code_debug=3
XIAO_ESP32C3.menu.DebugLevel.debug=Debug
XIAO_ESP32C3.menu.DebugLevel.debug.build.code_debug=4
XIAO_ESP32C3.menu.DebugLevel.verbose=Verbose
XIAO_ESP32C3.menu.DebugLevel.verbose.build.code_debug=5
```

Now we have manually added the XIAO ESP32C3 board.

- **Step 8.** Select your board and port

Board

Navigate to **Tools > Board > ESP32 Arduino** and select "**XIAO_ESP32C3**"

Port

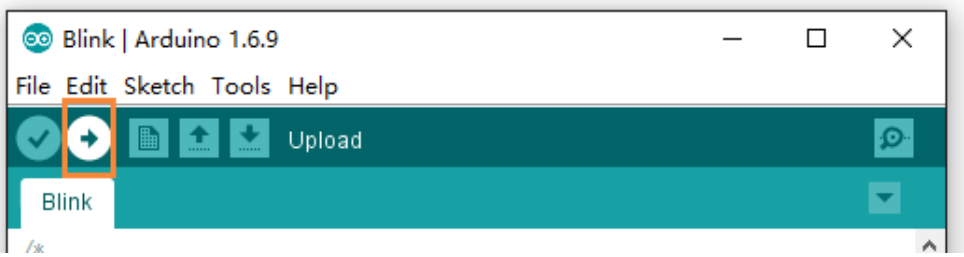
Navigate to **Tools > Port** and select the serial port name of the connected XIAO ESP32C3. This is likely to be COM3 or higher (**COM1** and **COM2** are usually reserved for hardware serial ports).

Blink the LED¶

- **Step 1.** Copy the below code to Arduino IDE

```
// define led according to pin diagram
int led = D10;
void setup() {
// initialize digital pin led as an output
pinMode(led, OUTPUT);
}
void loop() {
digitalWrite(led, HIGH); // turn the LED on
delay(1000); // wait for a second
digitalWrite(led, LOW); // turn the LED off
delay(1000); // wait for a second
}
```

- **Step 2.** Click the **Upload** button to upload the code to the board



Once uploaded, you will see the connected LED blinking with a 1-second delay between each blink. This means the connection is successful and now you can explore more projects with XIAO ESP32C3!

FAQ¶

Q1: My Arduino IDE is stuck when uploading code to the board¶

You can first try to reset the board by clicking the **RESET BUTTON** once while the board is connected to your PC. If that does not work, hold

the **BOOT BUTTON**, connect the board to your PC while holding the **BOOT** button, and then release it to enter **bootloader mode**.

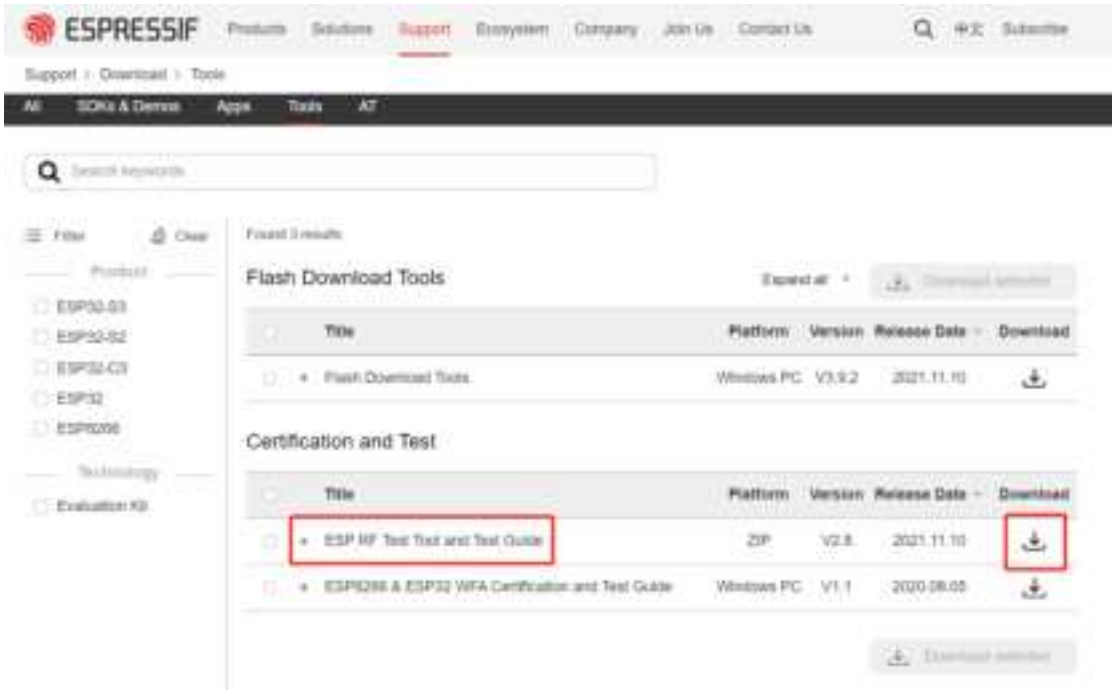
Q2: My board is not showing up as a serial device on Arduino IDE¶

Follow the same answer as for **Q1** above.

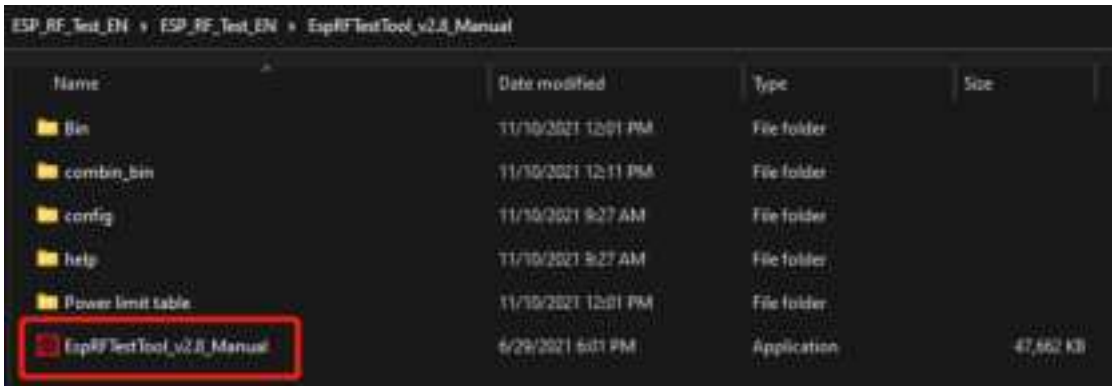
Q3: I want to reflash the bootloader with factory firmware¶

You can simply connect the board to a PC via **USB Type-C** and reflash the bootloader with factory firmware by using **ESP RF Test Tool**.

- **Step 1.** Hold on the **BOOT BUTTON** and connect XIAO ESP32C3 to the PC to enter **bootloader mode**
- **Step 2.** After it is connected, release the BOOT BUTTON
- **Step 3.** Visit this page and download **ESP RF Test Tool and Test Guide**

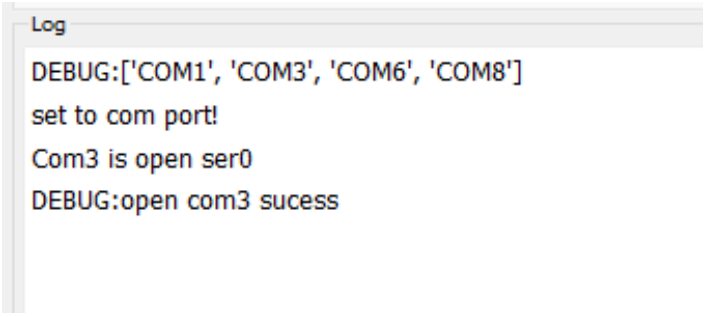


- **Step 4.** Extract the **.zip**, navigate to `ESP_RF_Test_EN\ESP_RF_Test_EN\EspRFTestTool_v2.8_Manual` and open **EspRFTestTool_v2.8_Manual.exe**

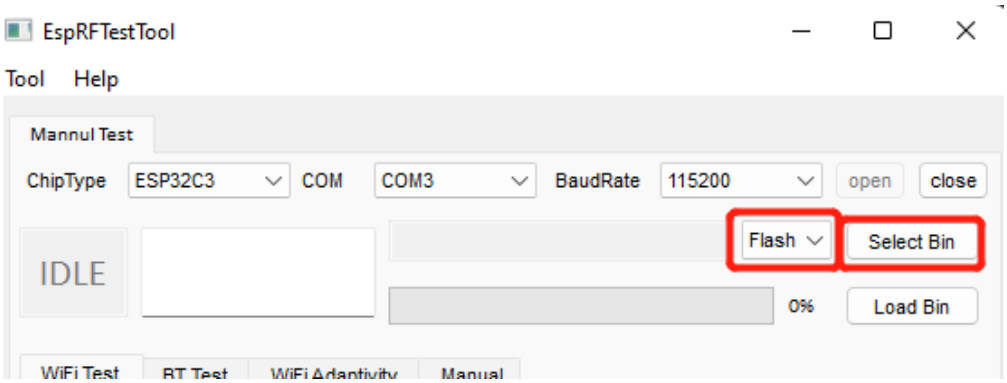


- **Step 5.** Select **ESP32C3** as the ChipType, your COM port, **115200** as the BaudRate and click **open**

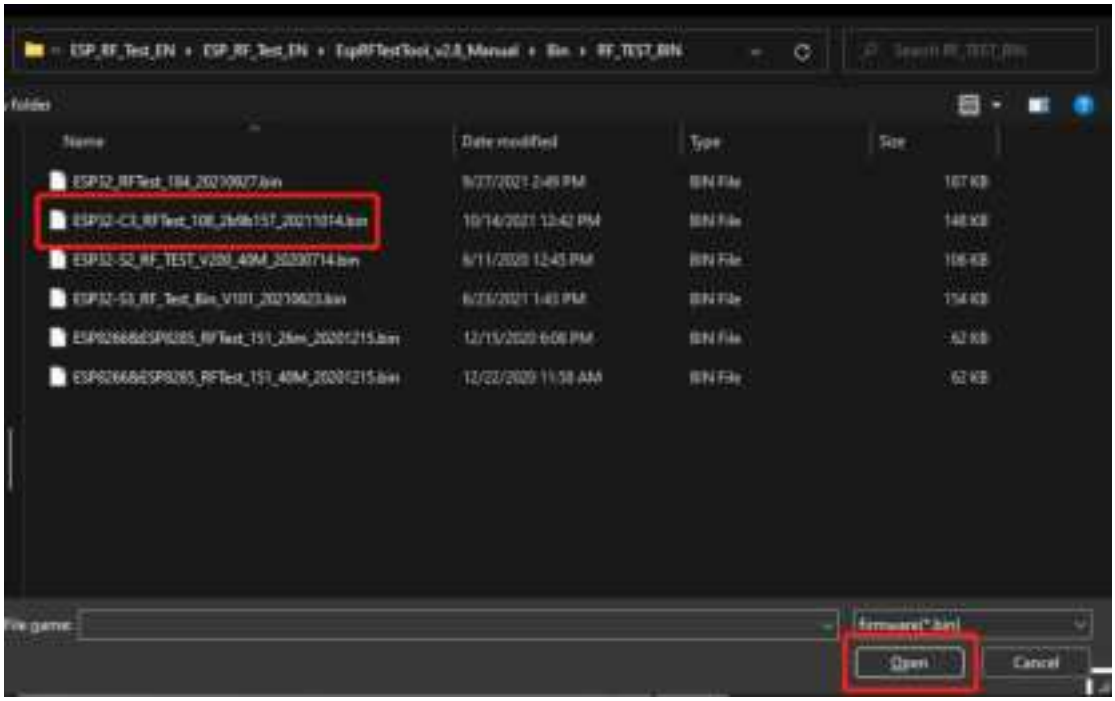
You will see the following output



- **Step 6.** Select **Flash** and click **Select Bin**

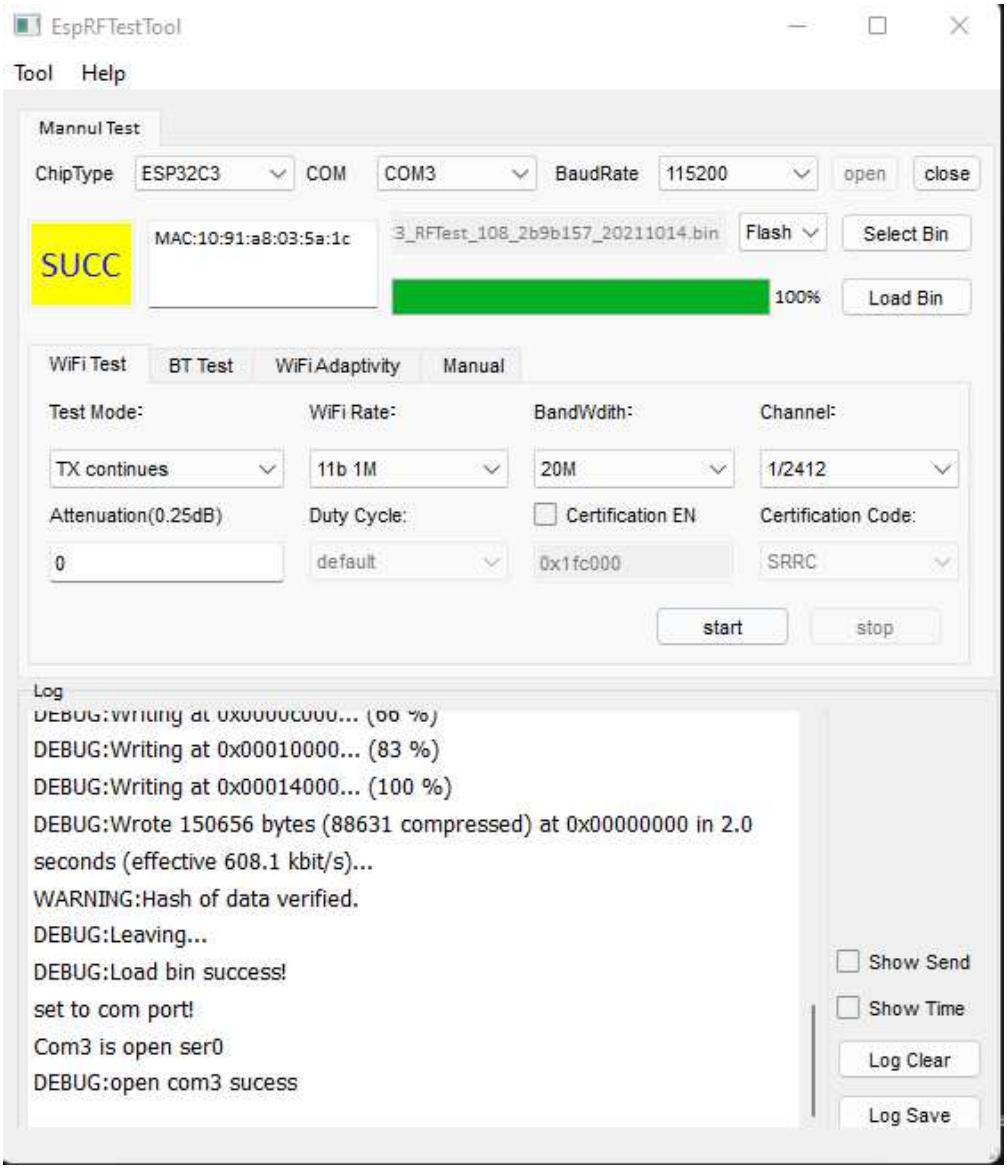


- **Step 7.** Select the file starting with **ESP32-C3** and click **Open**



- **Step 8.** Finally click **Load Bin**

You will see the following output when flashing is successful



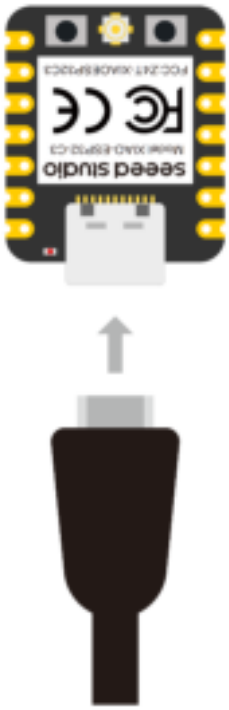
WiFi Usage

Seeed Studio XIAO ESP32C3 supports WiFi connectivity with IEEE 802.11b/g/n. This wiki will introduce the basics of WiFi usage on this board.

Hardware set up¶

- **Step 1.** Connect the included **WiFi/ Bluetooth antenna** to the **IPEX connector** on the board

- **Step 2.** Connect XIAO ESP32C3 to your computer via a USB Type-C cable



Scan WiFi networks (Station Mode)¶

In this example, we are going to use XIAO ESP32C3 to scan available WiFi networks around it. Here the board will be configured in Station (STA) Mode.

- **Step 1.** Copy and paste the code below into Arduino IDE

```
#include "WiFi.h"
void setup()
{
Serial.begin(115200);
// Set WiFi to station mode and disconnect from an AP if it was previously connected
WiFi.mode(WIFI_STA);
WiFi.disconnect();
delay(100);
Serial.println("Setup done");
}
void loop()
{
Serial.println("scan start");
```

```
// WiFi.scanNetworks will return the number of networks found
int n = WiFi.scanNetworks();
Serial.println("scan done");
if (n == 0) {
Serial.println("no networks found");
} else {
Serial.print(n);
Serial.println(" networks found");
for (int i = 0; i < n; ++i) {
// Print SSID and RSSI for each network found
Serial.print(i + 1);
Serial.print(": ");
Serial.print(WiFi.SSID(i));
Serial.print(" (");
Serial.print(WiFi.RSSI(i));
Serial.print(")");
Serial.println((WiFi.encryptionType(i) == WIFI_AUTH_OPEN)?" ":"*");
delay(10);
}
}
Serial.println("");
// Wait a bit before scanning again
delay(5000);
}
```

Step 2. Upload the codes and open the Serial Monitor to start scanning for WiFi networks

Connect to a WiFi network¶

In this example, we are going to use XIAO ESP32C3 to connect to a WiFI network.

- **Step 1.** Copy and paste the code below into Arduino IDE

```
#include <WiFi.h>
const char* ssid = "your-ssid";
const char* password = "your-password";
void setup()
{
Serial.begin(115200);
delay(10);
// We start by connecting to a WiFi network
Serial.println();
Serial.println();
Serial.print("Connecting to ");
Serial.println(ssid);
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
delay(500);
```

```
Serial.print(".");
}
Serial.println("");
Serial.println("WiFi connected");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());
}
void loop()
{
}
```

Step 2. Upload the codes and open the Serial Monitor to check that the board is connected to the WiFI network

Use as an Access Point¶

In this example, we are going to use XIAO ESP32C3 as a WiFi access point where other devices can be connected to it. This is similar to WiFi hotspot feature on mobile phones.

- **Step 1.** Copy and paste the code below into Arduino IDE

```
#include "WiFi.h"
void setup()
{
Serial.begin(115200);
WiFi.softAP("ESP_AP", "123456789");
```

```
}
void loop()
{
Serial.print("Host Name:");
Serial.println(WiFi.softAPgetHostname());
Serial.print("Host IP:");
Serial.println(WiFi.softAPIP());
Serial.print("Host IPV6:");
Serial.println(WiFi.softAPIPv6());
Serial.print("Host SSID:");
Serial.println(WiFi.SSID());
Serial.print("Host Broadcast IP:");
Serial.println(WiFi.softAPBroadcastIP());
Serial.print("Host mac Address:");
Serial.println(WiFi.softAPmacAddress());
Serial.print("Number of Host Connections:");
Serial.println(WiFi.softAPgetStationNum());
Serial.print("Host Network ID:");
Serial.println(WiFi.softAPNetworkID());
Serial.print("Host Status:");
Serial.println(WiFi.status());
delay(1000);
}
```

Step 2. Upload the codes and open the Serial Monitor to check more details about the WiFI access point

Step 3. Scan for WiFi networks on a PC or mobile phone and you will be able to connect to this newly created network using the password we specified in the code

Now you will see that the **Number of Host Connections** on serial monitor has been updated to **1**

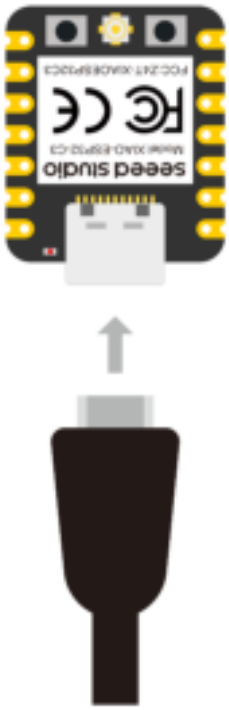
Bluetooth Usage

Seeed Studio XIAO ESP32C3 supports Bluetooth 5 (LE) connectivity. This wiki will introduce the basics of Bluetooth usage on this board.

Hardware set up¶

- **Step 1.** Connect the included **WiFi/ Bluetooth antenna** to the **IPEX connector** on the board

- **Step 2.** Connect XIAO ESP32C3 to your computer via a USB Type-C cable



Scan Bluetooth devices¶

In this example, we are going to use XIAO ESP32C3 to scan available Bluetooth devices around it.

- **Step 1.** Copy and paste the code below into Arduino IDE

```
#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEScan.h>
#include <BLEAdvertisedDevice.h>
int scanTime = 5; //In seconds
BLEScan* pBLEScan;
class MyAdvertisedDeviceCallbacks: public
BLEAdvertisedDeviceCallbacks {
void onResult(BLEAdvertisedDevice advertisedDevice) {
Serial.printf("Advertised Device: %s \n",
advertisedDevice.toString().c_str());
}
};
void setup() {
Serial.begin(115200);
Serial.println("Scanning...");
```

```
BLEDevice::init("");
pBLEScan = BLEDevice::getScan(); //create new scan
pBLEScan->setAdvertisedDeviceCallbacks(new
MyAdvertisedDeviceCallbacks());
pBLEScan->setActiveScan(true); //active scan uses more power, but get
results faster
pBLEScan->setInterval(100);
pBLEScan->setWindow(99); // less or equal setInterval value
}
void loop() {
// put your main code here, to run repeatedly:
BLEScanResults foundDevices = pBLEScan->start(scanTime, false);
Serial.print("Devices found: ");
Serial.println(foundDevices.getCount());
Serial.println("Scan done!");
pBLEScan->clearResults(); // delete results fromBLEScan buffer to
release memory
delay(2000);
}
```

Step 2. Upload the codes and open the Serial Monitor to start scanning for Bluetooth devices

XIAO ESP32C3 as Bluetooth server¶

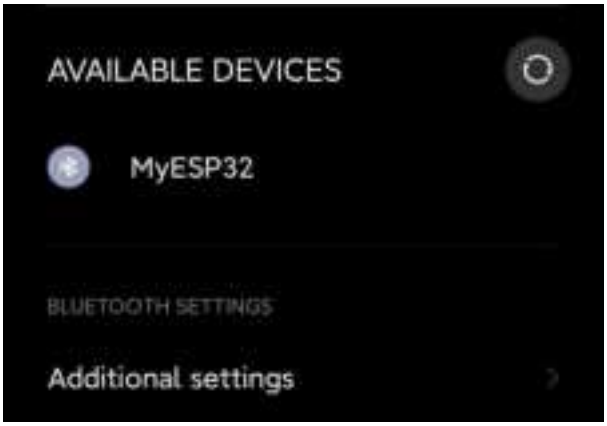
In this example, we are going to use XIAO ESP32C3 as a Bluetooth server. Here we will search for XIAO ESP32C3 board using a smartphone and send out strings to display on the serial monitor

- **Step 1.** Copy and paste the code below into Arduino IDE

```
#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEServer.h>
// See the following for generating UUIDs:
// https://www.uuidgenerator.net/
#define SERVICE_UUID "4fafc201-1fb5-459e-8fcc-c5c9c331914b"
#define CHARACTERISTIC_UUID "beb5483e-36e1-4688-b7f5-ea07361b26a8"
class MyCallbacks: public BLECharacteristicCallbacks {
void onWrite(BLECharacteristic *pCharacteristic) {
std::string value = pCharacteristic->getValue();
if (value.length() > 0) {
Serial.println("*********");
Serial.print("New value: ");
for (int i = 0; i < value.length(); i++)
Serial.print(value[i]);
Serial.println();
Serial.println("*********");
}
}
};
void setup() {
Serial.begin(115200);
BLEDevice::init("MyESP32");
BLEServer *pServer = BLEDevice::createServer();
BLEService *pService = pServer->createService(SERVICE_UUID);
BLECharacteristic *pCharacteristic = pService->createCharacteristic(
CHARACTERISTIC_UUID,
BLECharacteristic::PROPERTY_READ |
BLECharacteristic::PROPERTY_WRITE
);
pCharacteristic->setCallbacks(new MyCallbacks());
pCharacteristic->setValue("Hello World");
pService->start();
BLEAdvertising *pAdvertising = pServer->getAdvertising();
pAdvertising->start();
}
void loop() {
// put your main code here, to run repeatedly:
delay(2000);
}
```

- **Step 2.** Upload the codes and open the Serial Monitor

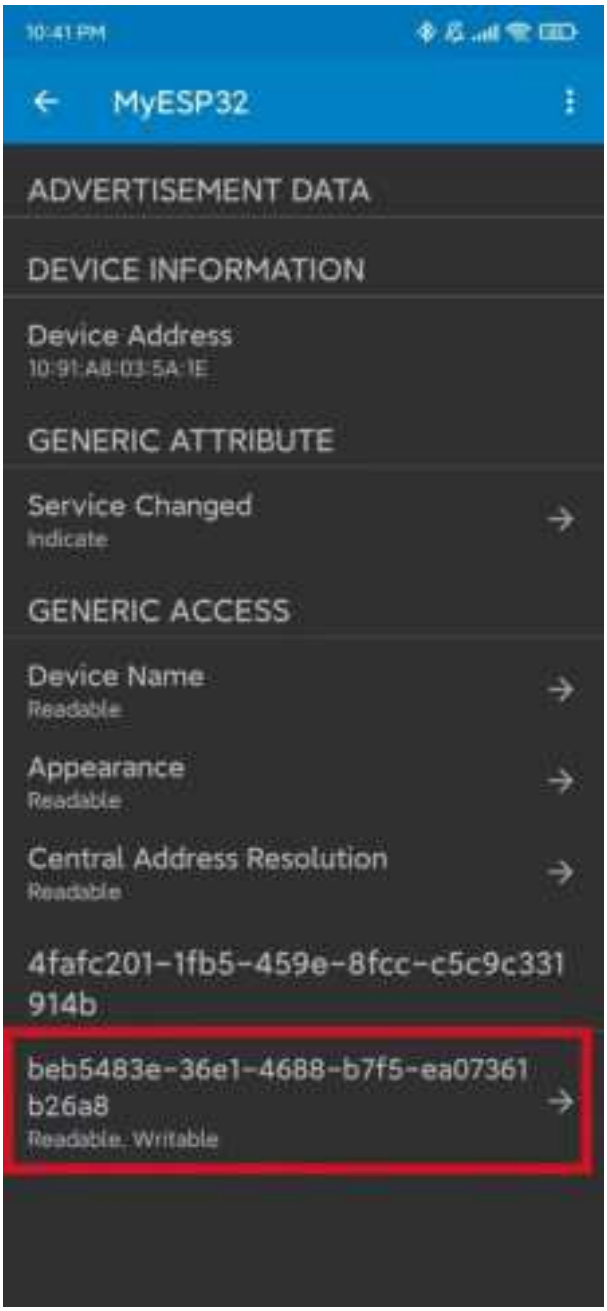
- **Step 3.** Download and install LightBlue App on your smartphone
- LightBlue App (Android)
- LightBlue App (Apple)
- **Step 4.** Open Bluetooth on your phone, bring the phone close to XIAO ESP32C3, scan for devices and connect with **MyESP32** device



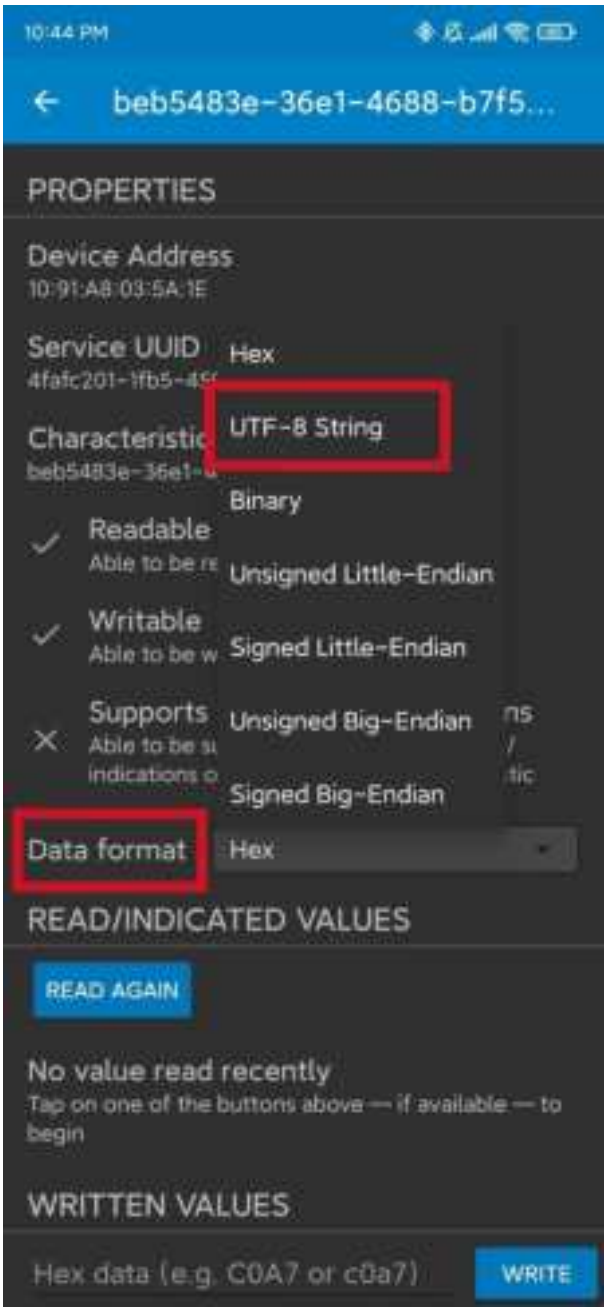
- **Step 5.** Open the LightBlue app and click **Bonded** tab

- **Step 6.** Click **CONNECT** next to **MyESP32**

- **Step 7.** Click the section at the very bottom which says **Readable, Writable**



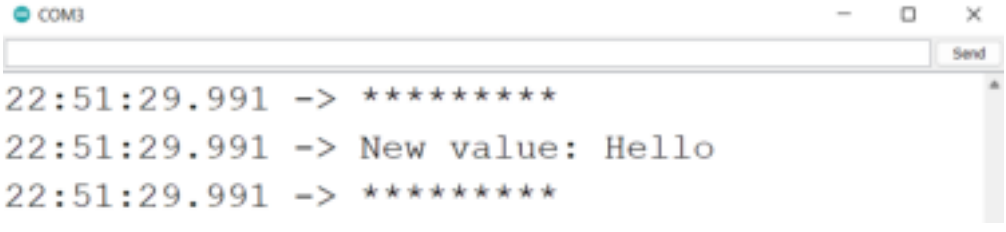
- **Step 8.** Under **Data format** drop-down menu, select **UTF-8 String**



- **Step 9.** Type "Hello" under **WRITTEN VALUES** and click **WRITE**

You will see the text string "Hello" output on the serial monitor of

Arduino IDE



Pin Multiplexing

Seeed Studio XIAO ESP32C3 has rich interfaces. There are **11 digital I/O** that can be used as **PWM pins** and **4 analog inputs** that can be used as **ADC pins**. It supports four serial communication interfaces such as **UART, I2C, SPI and I2S**. This wiki will be helpful to learn about these interfaces and implement them in your next projects!

Digital¶

Connect a pushbutton to Pin D6 and an LED to Pin D10. Then upload the following code to control the ON/OFF of LED using the pushbutton.

```
const int buttonPin = D6; // pushbutton connected to digital pin 6
const int ledPin = D10; // LED connected to digital pin 10
int buttonState = 0; // variable for reading the pushbutton status
void setup() {
// initialize the LED pin as an output:
pinMode(ledPin, OUTPUT);
// initialize the pushbutton pin as an input:
pinMode(buttonPin, INPUT);
}
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);
} else {
// turn LED off:
digitalWrite(ledPin, LOW);
}
}
```

Digital as PWM¶

Connect an LED to Pin D10. Then upload the following code to see the LED gradually fading.

```
int ledPin = D10; // LED connected to digital pin 10
void setup() {
// declaring LED pin as output
pinMode(led_pin, OUTPUT);
}
void loop() {
// fade in from min to max in increments of 5 points:
for (int fadeValue = 0 ; fadeValue <= 255; fadeValue += 5) {
// sets the value (range from 0 to 255):
analogWrite(ledPin, fadeValue);
// wait for 30 milliseconds to see the dimming effect
delay(30);
}
// fade out from max to min in increments of 5 points:
for (int fadeValue = 255 ; fadeValue >= 0; fadeValue -= 5) {
// sets the value (range from 0 to 255):
analogWrite(ledPin, fadeValue);
// wait for 30 milliseconds to see the dimming effect
delay(30);
}
}
```

Analog¶

Connect a potentiometer to Pin A0 and an LED to Pin D10. Then upload the following code to control the blinking interval of the LED by rotating the potentiometer knob.

```
const int sensorPin = A0;
const int ledPin = D10;
void setup() {
pinMode(sensorPin, INPUT); // declare the sensorPin as an INPUT
pinMode(ledPin, OUTPUT); // declare the ledPin as an OUTPUT
```

```
}
void loop() {
// read the value from the sensor:
int sensorValue = analogRead(sensorPin);
// turn the ledPin on
digitalWrite(ledPin, HIGH);
// stop the program for <sensorValue> milliseconds:
delay(sensorValue);
// turn the ledPin off:
digitalWrite(ledPin, LOW);
// stop the program for for <sensorValue> milliseconds:
delay(sensorValue);
}
```

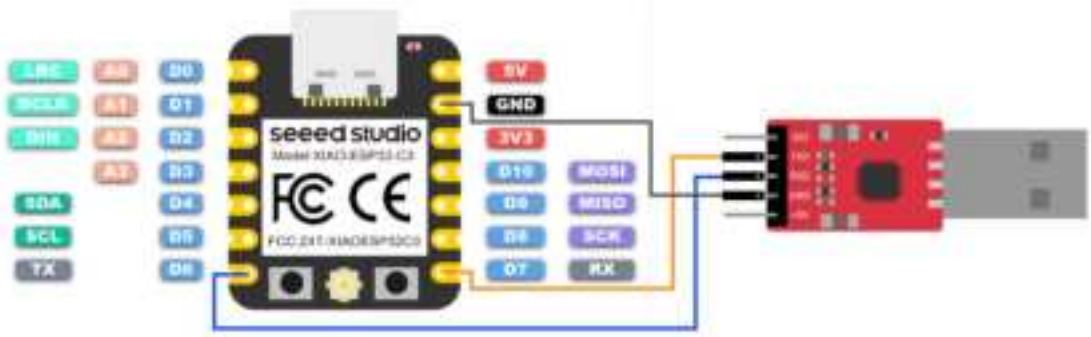
Serial¶

There are 2 serial interfaces on this board:

- USB Serial
- UART0 Serial

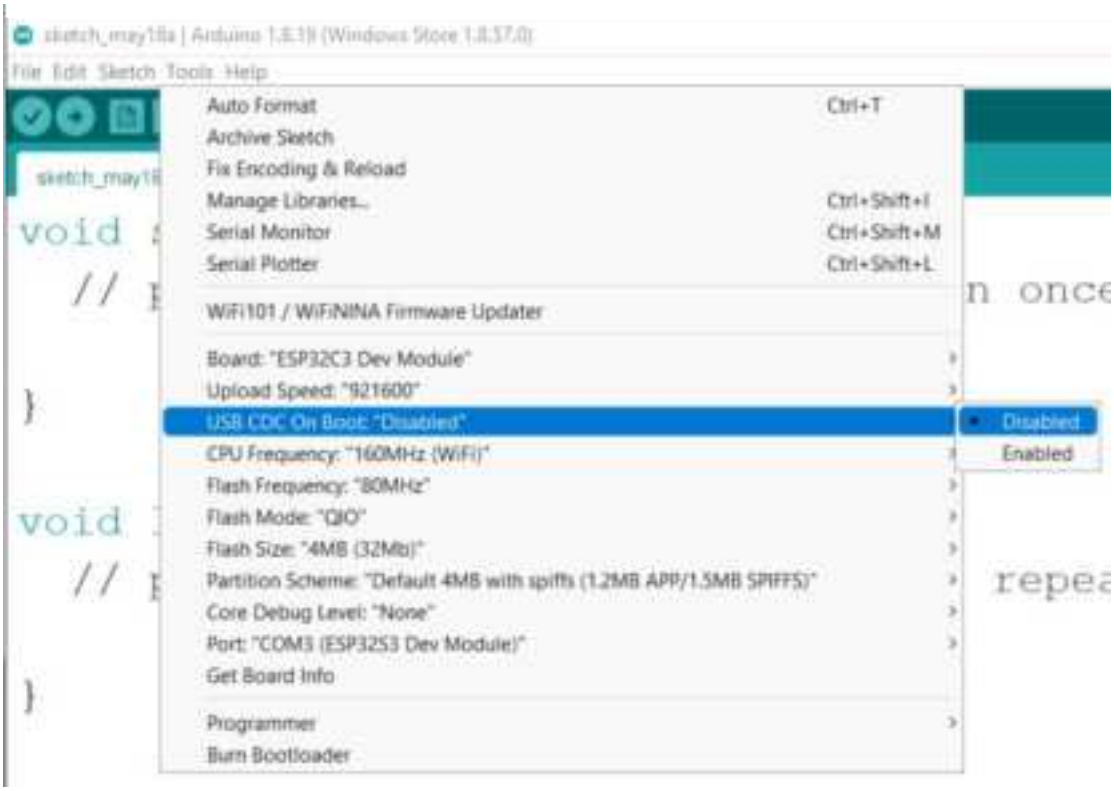
By default, USB serial is enabled, which means you can connect the board to a PC via USB Type-C and open serial monitor on Arduino IDE to view data sent via serial.

However, if you want to use UART0 as the serial, you need to connect pin D6 as the TX pin and pin D7 as RX pin with a USB-Serial adapter.



Also, you need to set **USB CDC On Boot** to **Disabled** from Arduino IDE.

NOTE: Change photo when board shows up on Arduino Board Manager



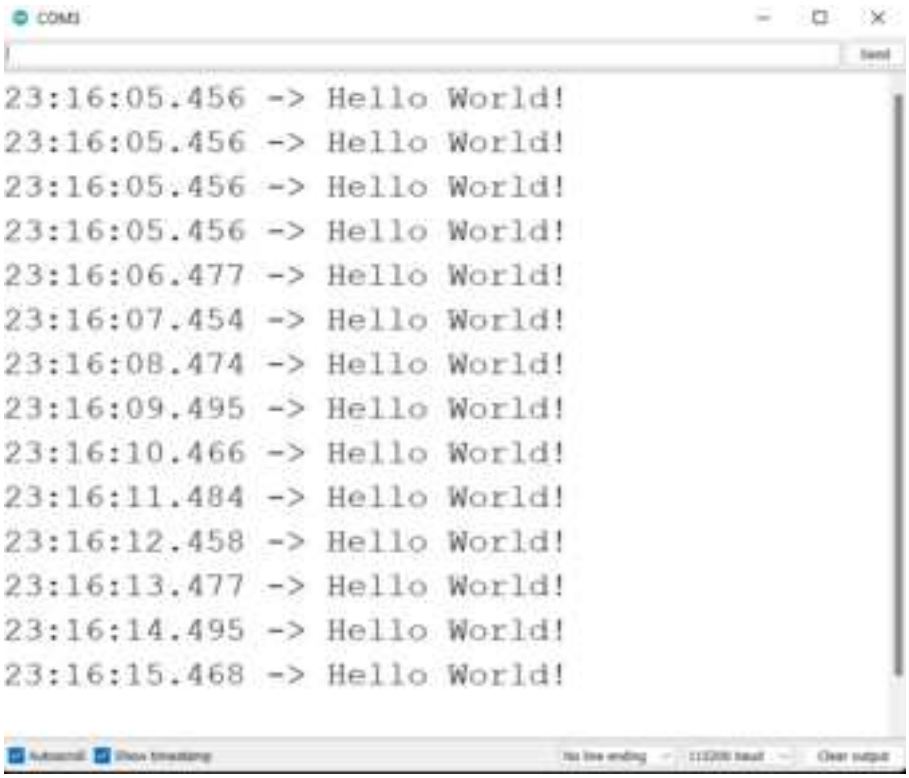
Upload the following code to Arduino IDE to send the string "Hello

World!" via serial

```
void setup() {
Serial.begin(115200);
while (!Serial);
}
```

```
void loop() {
Serial.println("Hello World!");
delay(1000);
}
```

The output will be as follows on Arduino Serial Monitor

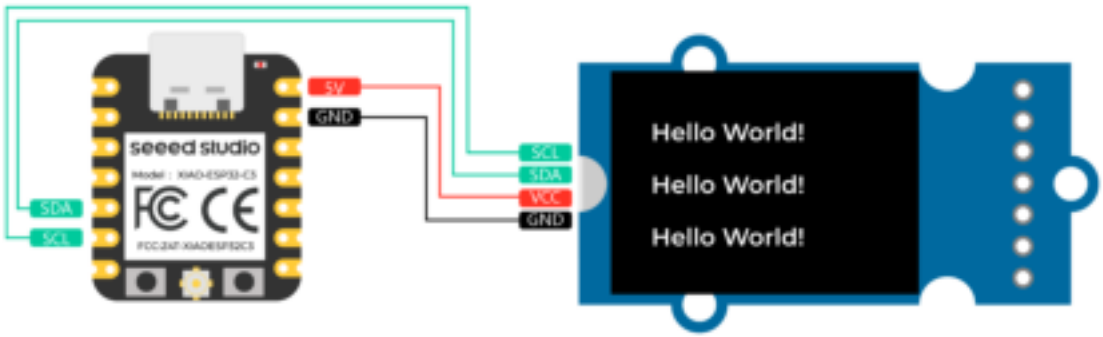


I2C¶

Hardware connection¶

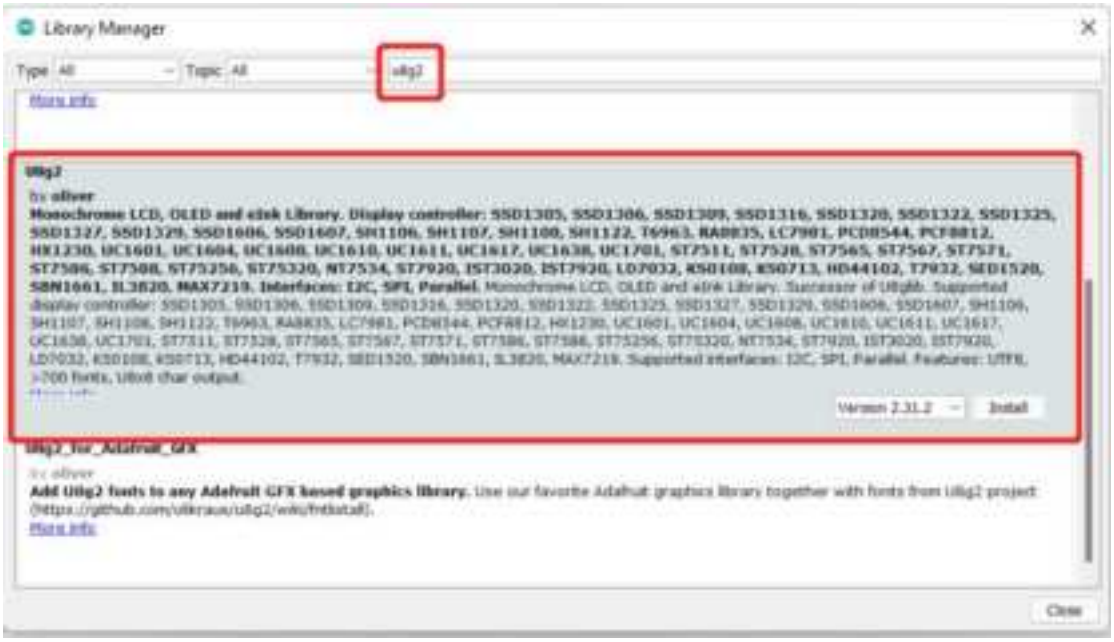
Connect a Grove - OLED Yellow&Blue Display 0.96 (SSD1315) to XIAO ESP32C3 by following the hardware connection as follows.

SCL	SCL
SDA	SDA
VCC	5V
GND	GND



Software setup¶

- **Step 1.** Open Arduino IDE, navigate to `Sketch > Include Library > Manage Libraries...`
- **Step 2.** Search for **u8g2** and install it



- **Step 3.** Upload the following code to display text strings on the OLED Display

```
//#include <Arduino.h>
#include <U8g2lib.h>
#ifdef U8X8_HAVE_HW_SPI
#include <SPI.h>
```

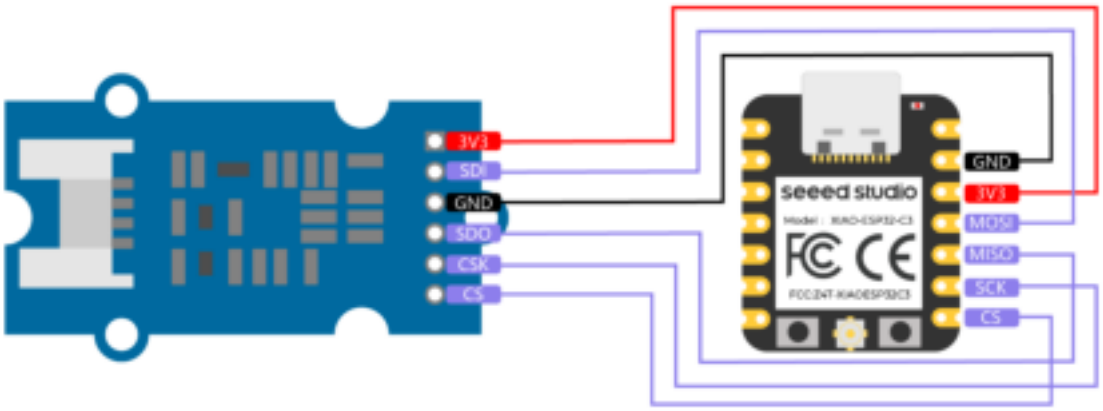
```
#endif
#ifdef U8X8_HAVE_HW_I2C
#include <Wire.h>
#endif
U8G2_SSD1306_128X64_NONAME_F_SW_I2C u8g2(U8G2_R0, /* clock=*/ SCL, /*
data=*/ SDA, /* reset=*/ U8X8_PIN_NONE); //Low spped I2C
void setup(void) {
u8g2.begin();
// u8x8.setFlipMode(1); // set number from 1 to 3, the screen word
will rotary 180
}
void loop(void) {
u8g2.clearBuffer(); // clear the internal memory
u8g2.setFont(u8g2_font_ncenB08_tr); // choose a suitable font
u8g2.drawStr(0,15,"Hello World!"); // write something to the internal
memory
u8g2.drawStr(0,30,"Hello World!");
u8g2.drawStr(0,40,"Hello World!");
u8g2.sendBuffer(); // transfer internal memory to the display
// delay(1000);
}
```

SPI¶

Hardware connection¶

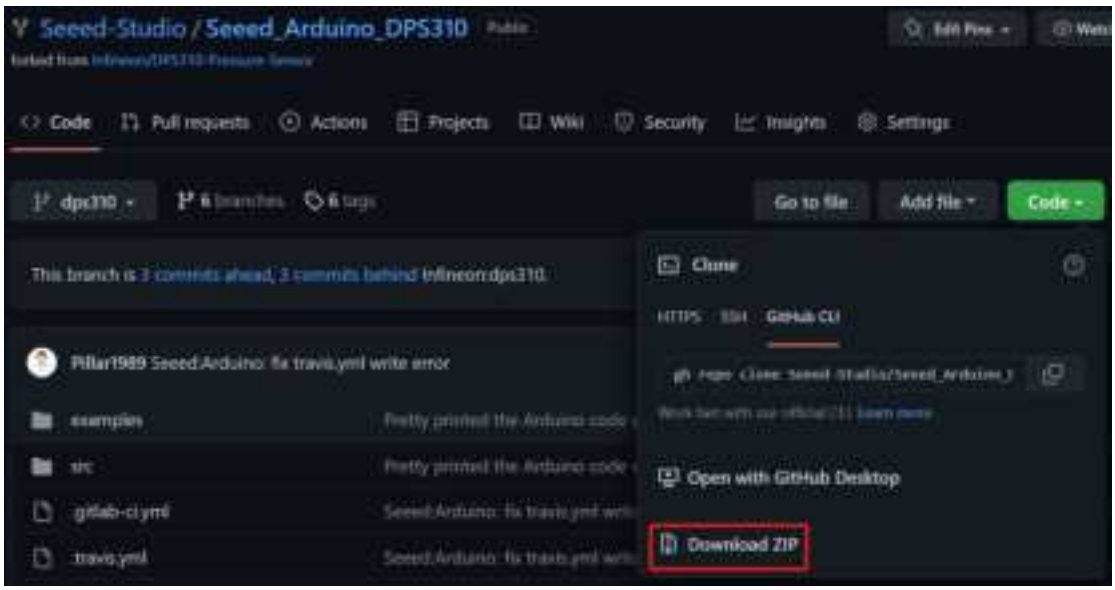
Connect a Grove - High Precision Barometric Pressure Sensor (DPS310) to XIAO ESP32C3 by following the hardware connection as follows.

3V3	3V3
SDI	MOSI
GND	GND
SDO	MISO
CSK	SCK
CS	CS



Software setup¶

- **Step 1**. Download Seeed_Arduino_DPS310 Library as a zip file



- **Step 2**. Open Arduino IDE, navigate to `Sketch > Include Library > Add .ZIP Library...` and open the downloaded zip file

- **Step 3.** Navigate to `File > Examples > DigitalPressureSensor > spi_background` to open the **spi_background** example

Alternatively you can copy the code from below as well

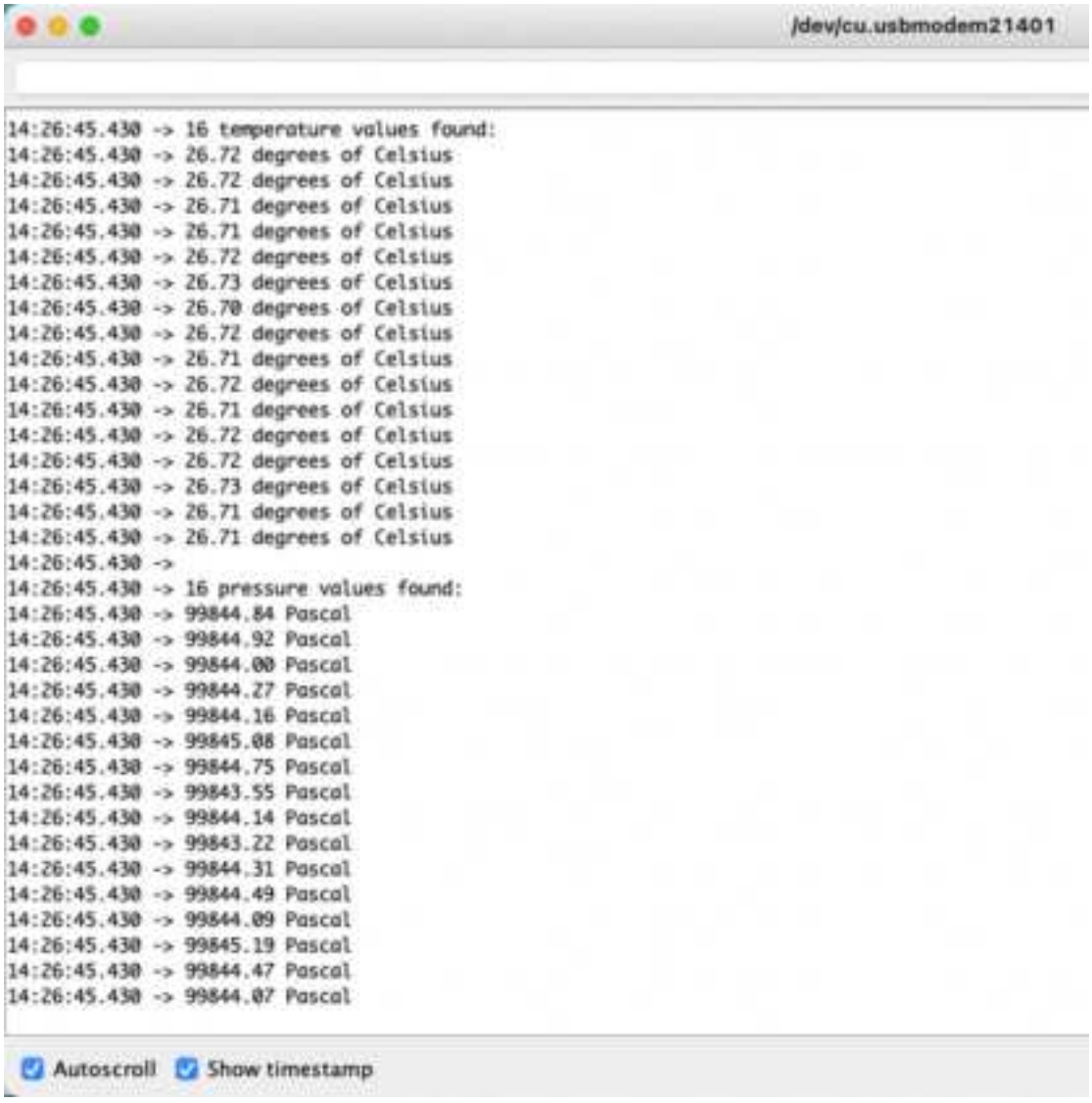
```
#include <Dps310.h>
// Dps310 Opject
Dps310 Dps310PressureSensor = Dps310();
```

```
void setup() {
//pin number of your slave select line
//XMC2GO
int16_t pin_cs = SS;
//for XMC 1100 Bootkit & XMC4700 Relax Kit uncomment the following
line
//int16_t pin_cs = 10;
Serial.begin(9600);
while (!Serial);
//Call begin to initialize Dps310
//The parameter pin_nr is the number of the CS pin on your
Microcontroller
Dps310PressureSensor.begin(SPI, pin_cs);
//temperature measure rate (value from 0 to 7)
//2^temp_mr temperature measurement results per second
int16_t temp_mr = 2;
//temperature oversampling rate (value from 0 to 7)
//2^temp_osr internal temperature measurements per result
//A higher value increases precision
int16_t temp_osr = 2;
//pressure measure rate (value from 0 to 7)
//2^prs_mr pressure measurement results per second
int16_t prs_mr = 2;
//pressure oversampling rate (value from 0 to 7)
//2^prs_osr internal pressure measurements per result
//A higher value increases precision
int16_t prs_osr = 2;
//startMeasureBothCont enables background mode
//temperature and pressure ar measured automatically
//High precision and hgh measure rates at the same time are not
available.
//Consult Datasheet (or trial and error) for more information
int16_t ret = Dps310PressureSensor.startMeasureBothCont(temp_mr,
temp_osr, prs_mr, prs_osr);
//Use one of the commented lines below instead to measure only
temperature or pressure
//int16_t ret = Dps310PressureSensor.startMeasureTempCont(temp_mr,
temp_osr);
//int16_t ret = Dps310PressureSensor.startMeasurePressureCont(prs_mr,
prs_osr);
if (ret != 0) {
Serial.print("Init FAILED! ret = ");
Serial.println(ret);
} else {
```

```
Serial.println("Init complete!");
}
}
void loop() {
uint8_t pressureCount = 20;
float pressure[pressureCount];
uint8_t temperatureCount = 20;
float temperature[temperatureCount];
//This function writes the results of continuous measurements to the
arrays given as parameters
//The parameters temperatureCount and pressureCount should hold the
sizes of the arrays temperature and pressure when the function is
called
//After the end of the function, temperatureCount and pressureCount
hold the numbers of values written to the arrays
//Note: The Dps310 cannot save more than 32 results. When its result
buffer is full, it won't save any new measurement results
int16_t ret = Dps310PressureSensor.getContResults(temperature,
temperatureCount, pressure, pressureCount);
if (ret != 0) {
Serial.println();
Serial.println();
Serial.print("FAIL! ret = ");
Serial.println(ret);
} else {
Serial.println();
Serial.println();
Serial.print(temperatureCount);
Serial.println(" temperature values found: ");
for (int16_t i = 0; i < temperatureCount; i++) {
Serial.print(temperature[i]);
Serial.println(" degrees of Celsius");
}
Serial.println();
Serial.print(pressureCount);
Serial.println(" pressure values found: ");
for (int16_t i = 0; i < pressureCount; i++) {
Serial.print(pressure[i]);
Serial.println(" Pascal");
}
}
//Wait some time, so that the Dps310 can refill its buffer
delay(10000);
}
```

- **Step 4.** Upload the codes and open the **Serial Monitor**

Note: Once you upload the codes, it will not be executed automatically until you click **Serial Monitor** on the upper right corner of the Arduino window.



Now you will see the temperature and pressure data displayed one after the other on the serial monitor as above!

Federal Communication Commission (FCC) Radiation Exposure Statement

This equipment complies with FCC RF radiation exposure limits set forth for an uncontrolled environment. When using the product, maintain a distance of 20cm from the body to ensure compliance with RF exposure requirements.

This device complies with part 15 of the FCC rules. Operation is subject to the following two conditions: (1) this device may not cause harmful interference, and (2) this device must accept any interference received, including interference that may cause undesired operation.

NOTE: The manufacturer is not responsible for any radio or TV interference caused by unauthorized modifications or changes to this equipment. Such modifications or changes could void the user's authority to operate the equipment.

NOTE: This equipment has been tested and found to comply with the limits for a Class B digital device, pursuant to part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference in a residential installation. This equipment generates uses and can radiate radio frequency energy and, if not installed and used in accordance with the instructions, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

- Reorient or relocate the receiving antenna.
- Increase the separation between the equipment and receiver.

-Connect the equipment into an outlet on a circuit different from that to which the receiver is connected.

-Consult the dealer or an experienced radio/TV technician for help.

FCC Caution: Any changes or modifications not expressly approved by the party responsible for compliance could void the user's authority to operate this equipment.

ORIGINAL EQUIPMENT MANUFACTURER (OEM) NOTES

The OEM must certify the final end product to comply with unintentional radiators (FCC Sections 15.107 and 15.109) before declaring compliance of the final product to Part 15 of the FCC rules and regulations. Integration into devices that are directly or indirectly connected to AC lines must add with Class II Permissive Change.

The OEM must comply with the FCC labeling requirements. If the module's label is not visible when installed, then an additional permanent label must be applied on the outside of the finished product which states: "Contains transmitter module FCC ID: Z4T-XIAOESP32C3.

Additionally, the following statement should be included on the label and in the final product's user manual: "This device complies with Part 15 of the FCC Rules. Operation is subject to the following two conditions: (1) This device may not cause harmful interferences, and (2) this device must accept any interference received, including interference that may cause undesired operation."

The module is allowed to be installed in mobile and portable applications

A module or modules can only be used without additional authorizations if they have been tested and granted under the same intended end - use operational conditions, including simultaneous transmission operations. When they have not been tested and granted in this manner, additional testing and/or FCC application filing may be required. The most straightforward approach to address additional testing conditions is to have the grantee responsible for the certification of at least one of the modules submit a permissive change application. When having a module grantee file a permissive change is not practical or feasible, the following guidance provides some additional options for host manufacturers. Integrations using modules where additional testing and/or FCC application filing(s) may be required are: (A) a module used in devices requiring additional RF exposure compliance information (e.g., MPE evaluation or SAR testing); (B) limited and/or split modules not meeting all of the module requirements; and (C) simultaneous transmissions for independent collocated transmitters not previously granted together.

This Module is full modular approval, it is limited to OEM installation ONLY.

Integration into devices that are directly or indirectly connected to AC lines must add with Class II Permissive Change. (OEM) Integrator has to assure compliance of the entire end product include the integrated Module. Additional measurements (15B) and/or equipment authorizations (e.g. Verification) may need to be addressed depending on co-location or simultaneous transmission issues if applicable. (OEM) Integrator is reminded to assure that these installation instructions will not be made available to the end user