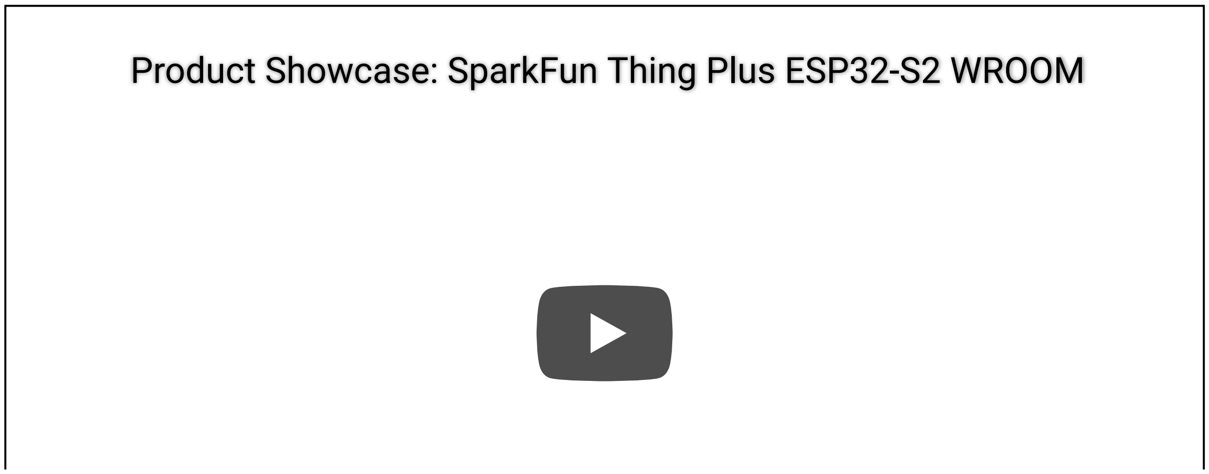
ESP32-S2 Thing Plus Hookup Guide

Introduction

The ESP32-S2 Thing Plus is a feather form-factor development board with the improved ESP32-S2 module, from Espressif. The new ESP32-S2 module addresses the security flaws in the original ESP32. While the ESP32-S2 does include improved security features, it lacks the Bluetooth capabilities of the original ESP32 module.

SparkFun Thing Plus - ESP32-S2 WROOM
WRL-17743



Required Materials

To get started, users will need a few items. Now some users may have a few of these items, feel free to modify your cart accordingly.

- ESP32-S2 Thing Plus
- USB 3.1 Cable A to C - 3 Foot - The USB interface serves two purposes: it powers the board and allows you to upload programs to it. (**If your computer doesn't provide a USB-A slot, then you will need to choose an appropriate cable or purchase an adapter as well.*)
- Computer with the an operating system (OS) that is compatible with all the software installation requirements.

SparkFun Thing Plus - ESP32-S2 WROOM
WRL-17743

USB 3.1 Cable A to C - 3 Foot
CAB-14743

HEADERS | BATTERIES | JUMPER MODIFICATION | JTAG FUNCTIONALITY

***Click the buttons** above to toggle the **additional materials** based on the options you wish to use. Feel free to modify the items in your cart to fit your needs.*

Suggested Reading

As a more professionally oriented product, we will skip over the more fundamental tutorials (i.e. **Ohm's Law** and **What is Electricity?**). However, below are a few tutorials that may help users familiarize themselves with various aspects of the board.

How to Solder: Through-Hole Soldering

Serial Communication

This tutorial covers everything you need to know about through-hole soldering.

Asynchronous serial communication concepts: packets, signal levels, baud rates, UARTs and more!

Serial Peripheral Interface (SPI)

SPI is commonly used to connect microcontrollers to peripherals such as sensors, shift registers, and SD cards.

Pulse Width Modulation

An introduction to the concept of Pulse Width Modulation.

Logic Levels

Learn the difference between 3.3V and 5V devices and logic levels.

I2C

An introduction to I2C, one of the main embedded communications protocols in use today.

Analog vs. Digital

This tutorial covers the concept of analog and digital signals, as they relate to electronics.

How to Work with Jumper Pads and PCB Traces

Handling PCB jumper pads and traces is an essential skill. Learn how to cut a PCB trace, add a solder jumper between pads to reroute connections, and repair a trace with the green wire method if a trace is damaged.

Installing Arduino IDE

A step-by-step guide to installing and testing the Arduino software on Windows, Mac, and Linux.

ESP32 Thing Plus Hookup Guide

Hookup guide for the ESP32 Thing Plus using the ESP32 WROOM's WiFi/Bluetooth system-on-chip in Arduino.

Installing Board Definitions in the Arduino IDE

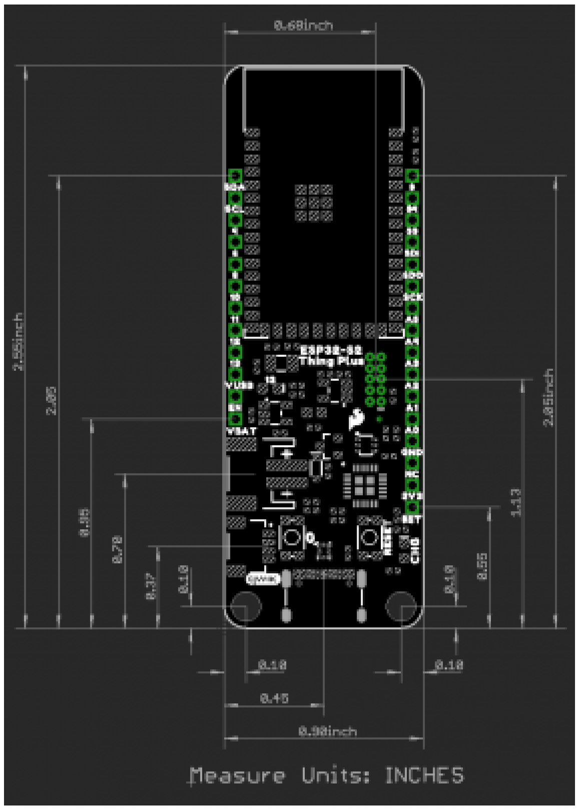
How do I install a custom Arduino board/core? It's easy! This tutorial will go over how to install an Arduino board definition using the Arduino Board Manager. We will also go over manually installing third-party cores, such as the board definitions required for many of the SparkFun development boards.

One of the new, advanced features of the board is that it takes advantage of the Qwiic connect system. We recommend familiarizing yourself with the **Logic Levels** and **I^{2}C** tutorials. Click on the banner above to learn more about Qwiic products.

Hardware Overview

Board Dimensions

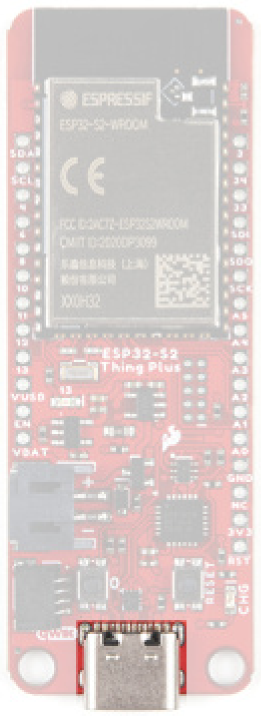
The board dimensions are illustrated in the drawing below. The listed measurements are in inches and the two mounting holes are compatible with 4-40 standoff screws.



[Board dimensions (PDF)](for the ESP32-S2 Thing Plus, in inches. (Click to enlarge)

USB-C Connector

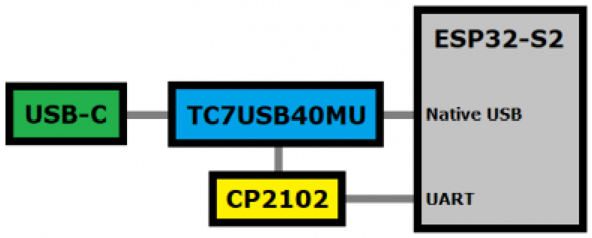
The USB connector is provided to power and program the board. For most users, it will be the primary programing interface for the ESP32-S2.



USB-C connector on the ESP32-S2 Thing Plus. (Click to enlarge)

TC7USB40MU USB Switch

The TC7USB40MU is used to switch the data lines for the USB-C connection, between the ESP32-S2 module's native USB pins or its UART pins, through a CP2102 Serial-to-UART bridge. There is a jumper on the back of the board to control the USB switch; by default, the data lines from the USB-C connector are tied to the CP2102 Serial-to-UART bridge.



Functional block diagram for the TC7USB40MU USB switch. (Click to enlarge)

CP2102 Serial-to-UART

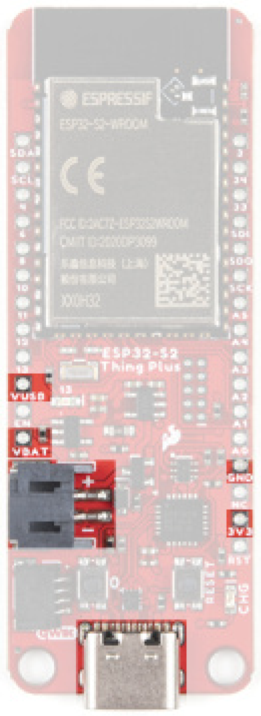
The CP2102 allows the ESP32-S2 to communicate with a computer/host device through its USB-C connection. This allows the board to show up as a device on the serial (or COM) port of the computer. Users will need to install the latest drivers for the computer to recognize the board *(see **Software Overview** section)*.

Power

The ESP32-S2 Thing Plus only requires 3.3V to power the board. However, the simplest method to power the board is with the USB-C connector. There are additional power pins available on the board:

- **3.3V** - A regulated 3.3V voltage source.
  - Regulated from the USB 5V power and/or battery connection.
  - Used to power the ESP32-S2 module and Qwiic I^2C bus.
- **USB** - The voltage from the USB-C connector, usually 5V.

- **VBAT** - The voltage from the JST battery connector; meant for single cell LiPo batteries.
- **GND** - The common ground or the 0V reference for the voltage supplies.



ESP32-S2 Thing Plus power connections. (Click to enlarge)

Charging Circuit

The charging circuit utilizes the MCP73831 linear charge management controller and is powered directly from the USB-C connector or `USB`. The controller is configured for a **500mA** charge rate by default; there is a jumper on the back of the boards for users to lower the charge rate to 100mA. Battery charging is indicated when the yellow, `CHG` LED. If the charge controller is shutdown or charging is complete, the `CHG` LED will turn off. For more information, please refer to the MCP73831 datasheet.

ESP32-S2 Module

The ESP32-S2 WROOM module from Espressif is the brains of the ESP32-S2 Thing Plus. While the new ESP32-S2 module does include improved security features to addresses the security flaws in the original ESP32, it lacks the Bluetooth capabilities of the original module. For more details check out the datasheets for the ESP32-S2 chipset and the ESP32-S2 WROOM module.

- Xtensa® Single-Core 32-bit LX7 Microprocessor *(up to 240MHz)*
    - RISC-V ULP Coprocessor
    - 128KB ROM and 320KB SRAM
    - 4MB of Embedded SPI Flash Storage
- Cryptographic Hardware Accelerators
    - AES, ECB/CBC/OFB/CFB/CTR, GCM, SHA, RSA, and ECC (Digital Signature)
- Physical Security Features
    - Transparent external flash and RAM encryption (AES-XTS)
    - Secure Boot feature ensures only signed firmware (with RSA-PSS signature) is booted
    - HMAC and Digital Signature modules use software inaccessible keys to generate SHA-MAC and MAC signatures

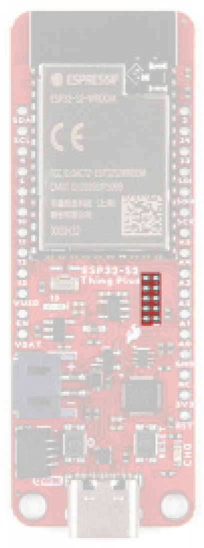
- Integrated 802.11 b/g/n WiFi 2.4GHz Transceiver *(up to 150Mbps)*
- Integrated Temperature Sensor *(-20°C to 110°C)*
- 34 GPIO *(excluding strapping pins)*
    - 1x USB OTG Interface
    - 1x DVP Camera Interface
    - 1x LCD Interface
- Operating Voltage: **3.0 to 3.6V**
    - WiFi: 310mA *(peak)*
    - Light-Sleep: 550µA
    - Deep-Sleep: 20-235µA

ESP32-S2 module on the ESP32-S2 Thing Plus. (Click to enlarge)

Note: Users should be aware of the following nuances and details of this board

- The ESP32-S2 is only compatible with **2.4GHz WiFi** networks; it will not work on the 5GHz bands.
- Users interested in the AES-XTS RAM encryption feature, will need to use the Espressif IDF. For more information, check out section **3.1.3 External Flash and RAM** of the ESP32-S2 chipset datasheet.
- For details on the boot mode configuration, please refer to section **3.3 Strapping Pins** of the ESP32-S2 WROOM module datasheet. By default, `IO46` uses an internal pull-down resistor.

Note: While most users will utilize the USB connection for serial programming, the ESP32-S2 can also be programmed through its JTAG or SWD pins. This might is useful for individuals developing and debugging firmware that would be flashed directly onto the module, such as in production for commercial applications.



The JTAG pins on the ESP32-S2 Thing Plus. (Click to enlarge)

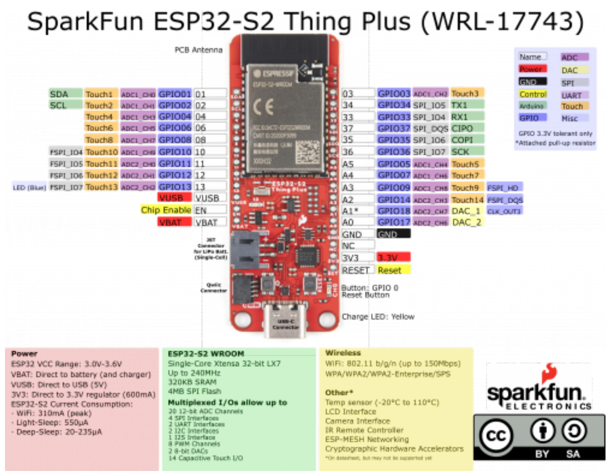
Peripherals and I/O

Note: Users should be aware of the following nuances of this board

- ⚡ All the GPIO on the ESP32-S2 Thing Plus are **3.3V** pins.
- ⚡ There are electrical limitations to the amount of current that the ESP32-S2 module can sink or source. For more details, check out the ESP32-S2 WROOM module datasheet.
- ⚡ There is a pullup resistor on `IO18` on pin `A1`. For more details, check out the **Note** following Fig. 6 in the **Peripheral Schematics** section of the ESP32-S2 WROOM module datasheet.
- There are some limitations to the ADC performance, see the **Note** from the **ADC Characteristics** section of the ESP32-S2 WROOM module datasheet.
- The `EN` pin is used to contol the ESP32-S2 chip.
    - **High:** On - enables the chip.
    - **Low:** Off - the chip powers off.

The pins from the ESP32-S2 module are broken out into a feather form factor layout. There are **21 I/O pins** broken out on this board, with 8 I/O pads on the back of the board. All of the ESP32-S2 Thing Plus pins are broken out with .1" pitch spacing for headers. With the ESP32's pin multiplexing, various pins will be capable of several functionalities. For more technical specifications on the **I/O** pins, you can refer to the ESP32-S2 datasheet.

- 16x 12-bit ADC Channels
- 2x 8-bit DAC
- 14x Capacitive Touch Sensing
- 4x SPI (only one is configured by default in the Arduino IDE)
- 1x I^{2}S
- 2x I^{2}C (only one is configured by default in the Arduino IDE)
- 2x UART (only two are configured by default in the Arduino IDE, one UART is used for bootloading/debug)
- 8x PWM Channels
- 1x DVP Camera Interface
- 1x LCD Interface



Graphical datasheet for the ESP32-S2 Thing Plus. (Click to enlarge)

Note: Users should be aware of the following limitations for the board in the Arduino IDE.

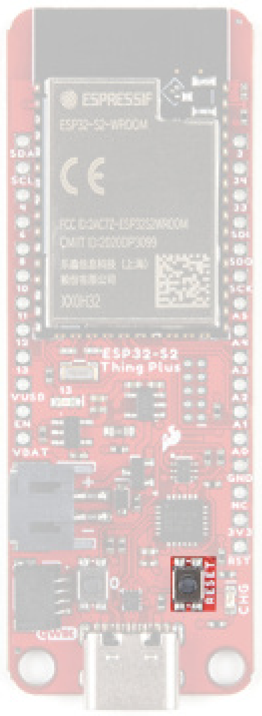
- Not all of the features, listed above, are available in the Arduino IDE. For the full capabilities of the ESP32-S2, the Espressif IDF should be utilized.
- Only one I^{2}C bus is defined.
- Only two UART interfaces are available.
  - **UART (USB):** `Serial`
  - **`RX1` / `TX1` Pins:** `Serial1`
- Only one SPI bus is defined.
- The bit-width for the ADC2 channels needs to be set when reading ADC values.
  - Setting the bit-width before taking any ADC readings doesn't work; the bit-width must be set, when the making ADC readings.

Buttons

There are two buttons on ESP32-S2 Thing Plus; a `Reset` and `IO0` button.

Reset Button

The reset (**`RST`**) button allows users to reset the program running on the ESP32-S2 module without unplugging the board.



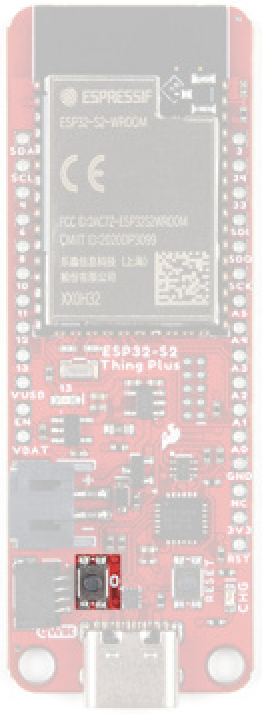
Reset button on the ESP32-S2 Thing Plus. (Click to enlarge)

User Button

Note: In order to utilize the user button, connected to `IO 0`, the pin mode will need to be configured as an input with an internal pullup (i.e. `INPUT_PULLUP`); see an example below.

```
pinMode(D0, INPUT_PULLUP);
```

The user (`0`) button allows users to short `IO 0` to ground (`GND`).



`IO 0` button on the ESP32-S2 Thing Plus. (Click to enlarge)

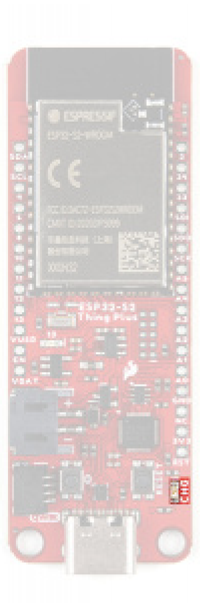
Indicator LEDs

There are two indicator LEDs on the ESP32-S2 Thing Plus:

- Status/Pin 13 (Blue)
- Battery Charging (Yellow)

Battery Charging LED

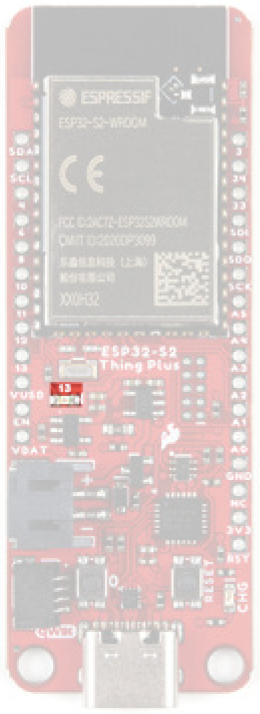
The yellow, **`CHG`** LED will light while a battery is being charged through the charging circuit. The LED will be off when no battery is present (**dimmed*), when the charge management controller is in standby (*after the battery charging has been completed*), or when the charge management controller is shutdown. The LED will be on when the charge management controller is in the process of charging the battery. For more information, please refer to the MCP73831 datasheet.


The battery charging (`CHG`) LED indicator on the ESP32-S2 Thing Plus. (Click to enlarge)

Charge Cycle State	STAT1
Shutdown • Thermal Shutdown • $V_{DD} < V_{BAT}$	**Off** (High Z)
No Battery Present	**Dimmed** (High Z)
Charge Complete – Standby	**Off** (H)
Preconditioning	**On** (L)
Constant-Current Fast Charge	**On** (L)
Constant Voltage	**On** (L)

STAT LED

The blue, status (**`13`**) LED is typically used as a test or status LED to make sure that a board is working or for basic debugging. This indicator is connected to `GPIO 18` .

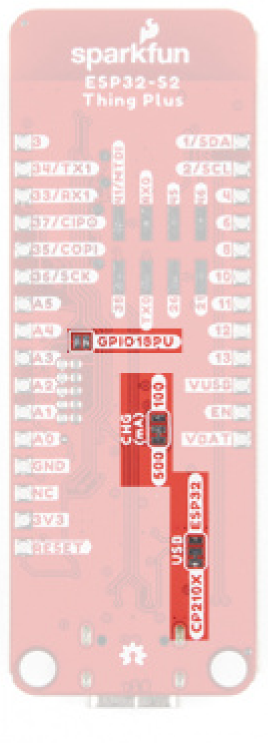


The status (`13`) LED indicator on the ESP32-S2 Thing Plus. (Click to enlarge)

Jumpers

There are three jumpers on the back of the board that can be used to easily modify the hardware connections on the board.

- **USB** - This jumper can be used to control the USB-C data line connections through the TC7USB40MU USB switch.
  - By default, the data lines are connected to the CP2102 Serial-to-UART bridge; and therefore, tied to the ESP32-S2's UART pins.
- **CHG** - This jumper can be used to modify the charging rate to the LiPo JST connector.
  - By default, the charge rate is configured to **500mA**.
- **GPIO18PU** - This jumper can be used to disconnect the pull-up resistor on `GPIO18` .



The jumpers on the back of the ESP32-S2 Thing Plus. (Click to enlarge)

Never modified a jumper before? Check out our Jumper Pads and PCB Traces tutorial for a quick introduction!

How to Work with Jumper Pads and PCB Traces

APRIL 2, 2018

Handling PCB jumper pads and traces is an essential skill. Learn how to cut a PCB trace, add a solder jumper between pads to reroute connections, and repair a trace with the green wire method if a trace is damaged.

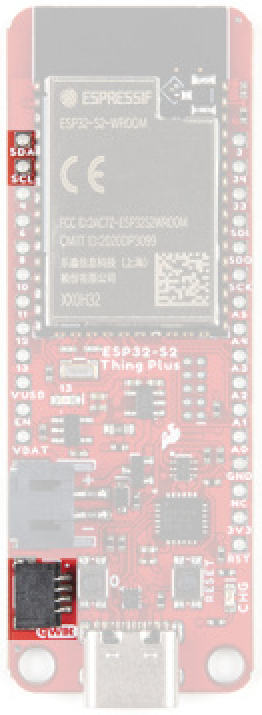
Primary I^{2}C Bus

The Qwiic connector is attached to the primary I^{2}C bus. The primary I^{2}C bus for this board utilizes the pin connections, detailed in the table below:

Connection	`VDD`	`GND`	`SCL`	`SDA`
Pad Number ***(ESP32-S2 module)***	**3.3V**	**GND**	`IO1`	`IO2`

Qwiic Connector

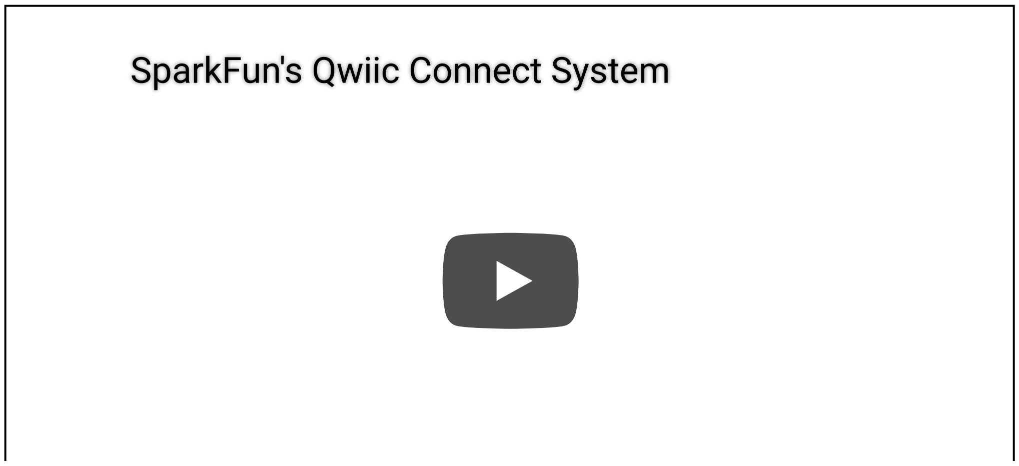
A Qwiic connector is provided for users to seamlessly integrate with SparkFun's Qwiic Ecosystem.



Qwiic connector and I^{2}C pins on the ESP32-S2 Thing Plus. (Click to enlarge)

What is Qwiic?

The Qwiic system is intended a quick, hassle-free cabling/connector system for I^{2}C devices. Qwiic is actually a play on words between "quick" and I^{2}C or "iic".

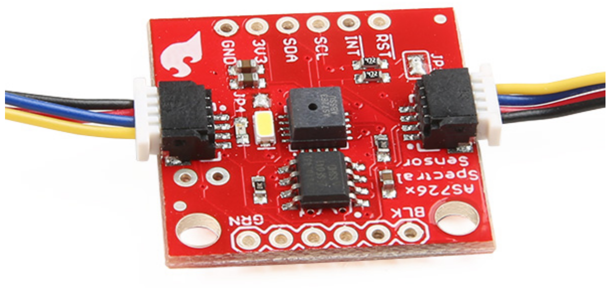


Features of the Qwiic System

NO SOLDERING | POLARIZED CONNECTOR | DAISY CHAIN

Keep your soldering iron at bay.

Cables plug easily between boards making quick work of setting up a new prototype. We currently offer three different lengths of Qwiic cables as well as a breadboard friendly cable to connect any Qwiic enabled board to anything else. Initially you may need to solder headers onto the shield to connect your platform to the Qwiic system but once that's done it's plug and go!



Qwiic cables connected to Spectral Sensor Breakout

Software Overview

Installing the CP2104 USB Driver

Note: Make sure to manually install the driver for the CP210X with the following instructions. The driver that Windows auto-installs will not work with the auto-reset circuit on the board and cause serial uploads to fail.

Users will need to install the SiLabs CP2104 Driver, which can be found here: USB to UART Bridge VCP Driver

DOWNLOAD WINDOWS VCP DRIVER (ZIP)

DOWNLOAD MAC OSX VCP DRIVER (ZIP)

Note: If applicable, make sure you are using the proper driver files for your CPU architecture. This is usually indicated by a folder or file name with "*x86*" for 32-bit processors or "*x64*" for 64-bit processors.

Arduino IDE

Note: For first-time users, who have never programmed before and are looking to use the Arduino IDE, we recommend beginning with the SparkFun Inventor's Kit (SIK), which includes a simpler board like the Arduino Uno or SparkFun RedBoard and is designed to help users get started programming with the Arduino IDE.

Most users may already be familiar with the Arduino IDE and it's use. However, for those of you who have never heard the name *Arduino* before, feel free to check out the Arduino website. To get started with using the Arduino IDE, check out our tutorials below:

Installing an Arduino Library

How do I install a custom Arduino library? It's easy! This tutorial will go over how to install an Arduino library using the Arduino Library Manager. For libraries not linked with the Arduino IDE, we will also go over manually installing an Arduino library.

What is an Arduino?

What is this 'Arduino' thing anyway? This tutorials dives into what an Arduino is and along with Arduino projects and widgets.

Installing Arduino IDE

A step-by-step guide to installing and testing the Arduino software on Windows, Mac, and Linux.

Installing Board Definitions in the Arduino IDE

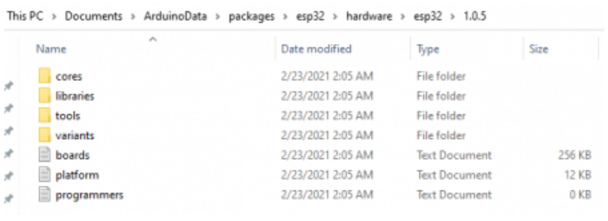
How do I install a custom Arduino board/core? It's easy! This tutorial will go over how to install an Arduino board definition using the Arduino Board Manager. We will also go over manually installing third-party cores, such as the board definitions required for many of the SparkFun development boards.

Install Board Definition

Installing *Development* Arduino Core:

Currently, the ESP32 Arduino core hasn't been updated to include support for the ESP32-S2. However, there is a branch in development that users can use to get a head start on their development in the Arduino IDE. For more information, please refer to the **Installation Instructions** from the GitHub repository. *(*User* *will need to have Git installed and be familiar with utilizing the command prompt.)*

1. Install the latest ESP32 Arduino core, through the Ardiuno IDE **Boards Manager**.
2. Locate the hardware folder for the Espressif ESP32 Arduino core. Depending on your operating system, IDE version, and IDE type *(i.e. store App, `.zip` file installation, or `.exe` installation)*, the folder location may vary. For our computers, the folder for the Arduino IDE was located in the `Documents` folder. From there, depending on how the Arduino IDE was installed, the folder for the ESP32 Arduino core was in one of the following locations:
    - `Arduino\hardware\espressif\esp32`
    - `ArduinoData\packages\esp32\hardware\esp32\<version_number>`
3. Delete the file contents.

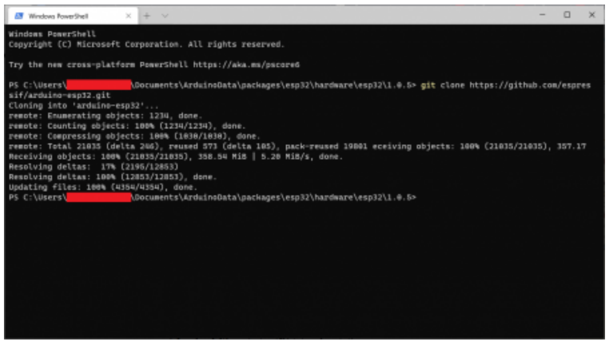


This PC › Documents › ArduinoData › packages › esp32 › hardware › esp32 › 1.0.5

Name	Date modified	Type	Size
cores	2/23/2021 2:05 AM	File folder	
libraries	2/23/2021 2:05 AM	File folder	
tools	2/23/2021 2:05 AM	File folder	
variants	2/23/2021 2:05 AM	File folder	
boards	2/23/2021 2:05 AM	Text Document	256 KB
platform	2/23/2021 2:05 AM	Text Document	12 KB
programmers	2/23/2021 2:05 AM	Text Document	0 KB

Example of file contents. (Click to enlarge)

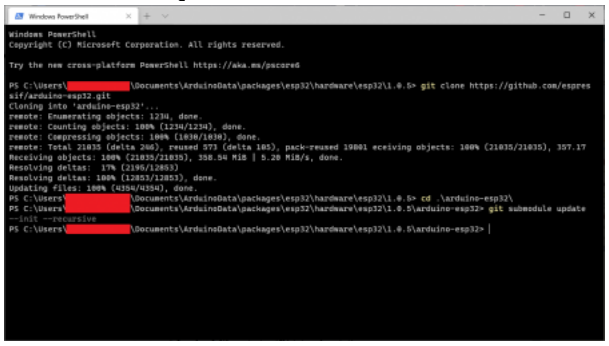
4. Using the command prompt (or terminal), clone the `master` branch of the ESP32 Arduino core from the GitHub repository with the following command: `git clone https://github.com/espressif/arduino-esp32.git`



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\[redacted]\Documents\ArduinoData\packages\esp32\hardware\esp32\1.0.5> git clone https://github.com/espres
sif/arduino-esp32.git
Cloning into 'arduino-esp32'...
remote: Enumerating objects: 1234, done.
remote: Counting objects: 100% (1234/1234), done.
remote: Compressing objects: 100% (1030/1030), done.
remote: Total 21035 (delta 246), reused 573 (delta 105), pack-reused 19801 eceiving objects: 100% (21035/21035), 357.17
Receiving objects: 100% (21035/21035), 358.54 MiB | 5.20 MiB/s, done.
Resolving deltas:  17% (2196/12853)
Resolving deltas: 100% (12853/12853), done.
Updating files: 100% (4354/4354), done.
PS C:\Users\[redacted]\Documents\ArduinoData\packages\esp32\hardware\esp32\1.0.5>
```

Cloning GiHub repository. (Click to enlarge)

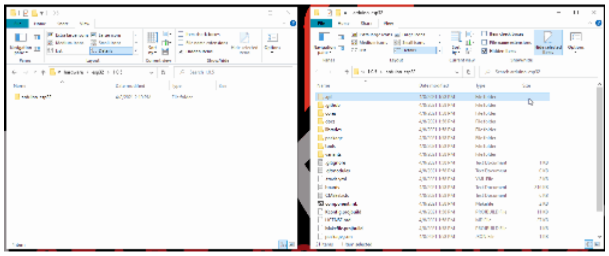
5. Using the command prompt (or terminal), change directories into the `arduino-esp32` folder. Then, update the submodules with the following command: `git submodule update --init --recursive`



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\[redacted]\Documents\ArduinoData\packages\esp32\hardware\esp32\1.0.5> git clone https://github.com/espres
sif/arduino-esp32.git
Cloning into 'arduino-esp32'...
remote: Enumerating objects: 1234, done.
remote: Counting objects: 100% (1234/1234), done.
remote: Compressing objects: 100% (1030/1030), done.
remote: Total 21035 (delta 246), reused 573 (delta 105), pack-reused 19801 eceiving objects: 100% (21035/21035), 357.17
Receiving objects: 100% (21035/21035), 358.54 MiB | 5.20 MiB/s, done.
Resolving deltas:  17% (2196/12853)
Resolving deltas: 100% (12853/12853), done.
Updating files: 100% (4354/4354), done.
PS C:\Users\[redacted]\Documents\ArduinoData\packages\esp32\hardware\esp32\1.0.5> cd .\arduino-esp32\
PS C:\Users\[redacted]\Documents\ArduinoData\packages\esp32\hardware\esp32\1.0.5\arduino-esp32> git submodule update
--init --recursive
PS C:\Users\[redacted]\Documents\ArduinoData\packages\esp32\hardware\esp32\1.0.5\arduino-esp32> |
```

Updating submodules. (Click to enlarge)

6. Move the contents from the `arduino-esp` folder up a directory *(i.e. into the folder the `arduino-esp` folder resides in)*.

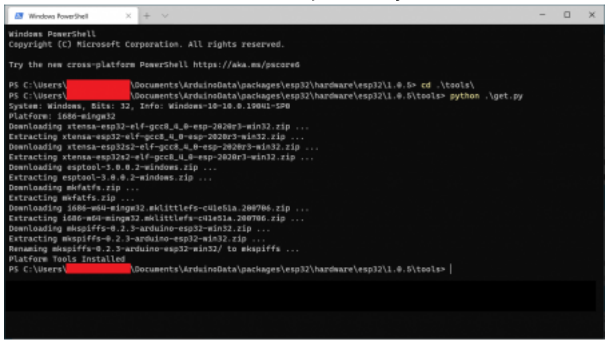


Moving the file contents back into the appropriate directory. (Click to enlarge)

**In step 4, users could also have modified the command to clone the GitHub repository into the current directory, but the command varies by system:*

- `git clone https://github.com/espressif/arduino-esp32.git .\`
- `git clone https://github.com/espressif/arduino-esp32.git && \`

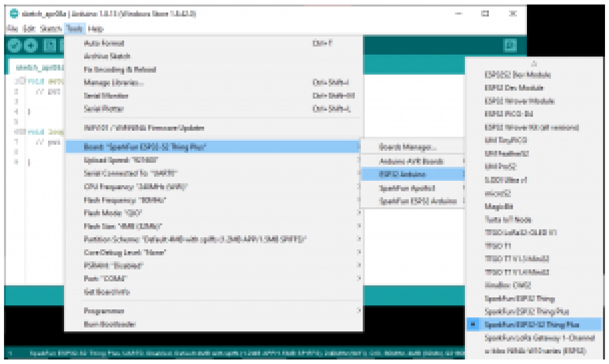
7. Using the command prompt (or terminal), change directories into the `tools` folder. Then, execute the `get.py` *(or `get.exe`)* file *(depending on your operating system)* to install the required tools. Using the command prompt, this is done with the command: `python get.py` . For more details, please refer to the **Installation Instructions** from the GitHub repository.



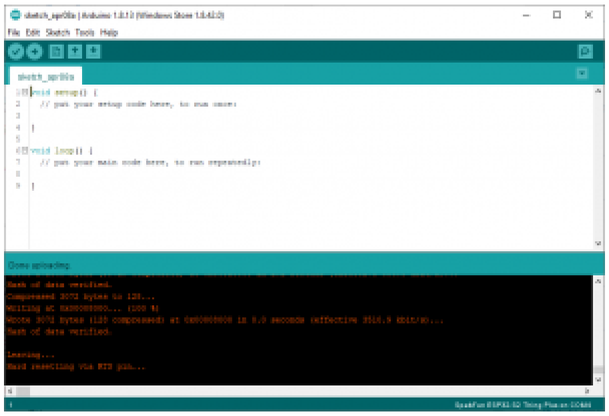
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\[redacted]\Documents\ArduinoData\packages\esp32\hardware\esp32\1.0.5> cd .\tools\
PS C:\Users\[redacted]\Documents\ArduinoData\packages\esp32\hardware\esp32\1.0.5\tools> python .\get.py
System: Windows, Bits: 32, Info: Windows-10-10.0.19041-SP0
Platform: i686-mingw32
Downloading xtensa-esp32-elf-gcc8_4_0-esp-2020r3-win32.zip ...
Extracting xtensa-esp32-elf-gcc8_4_0-esp-2020r3-win32.zip ...
Downloading xtensa-esp32s2-elf-gcc8_4_0-esp-2020r3-win32.zip ...
Extracting xtensa-esp32s2-elf-gcc8_4_0-esp-2020r3-win32.zip ...
Downloading esptool-3.0.0.2-windows.zip ...
Extracting esptool-3.0.0.2-windows.zip ...
Downloading mkfatfs.zip ...
Extracting mkfatfs.zip ...
Downloading i686-w64-mingw32.mklittlefs-c41e51a.200706.zip ...
Extracting i686-w64-mingw32.mklittlefs-c41e51a.200706.zip ...
Downloading mkspiffs-0.2.3-arduino-esp32-win32.zip ...
Extracting mkspiffs-0.2.3-arduino-esp32-win32.zip ...
Renaming mkspiffs-0.2.3-arduino-esp32-win32/ to mkspiffs ...
Platform Tools Installed
PS C:\Users\[redacted]\Documents\ArduinoData\packages\esp32\hardware\esp32\1.0.5\tools> |
```

Installing required tools. (Click to enlarge)

Users should now, be able to open the Arduino IDE and select the `ESP32-S2 Thing Plus` board and upload code.


File Edit Sketch Tools Help
Board: "SparkFun ESP32-S2 Thing Plus"
Boards Manager...
Arduino AVR Boards
ESP32 Arduino
SparkFun ESP32-S2 Thing Plus

`ESP32-S2 Thing Plus` is available in the board options. (Click to enlarge)



```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Done uploading.

```
Hash of data verified.
Compressed 3072 bytes to 128...
Writing at 0x00008000... (100 %)
Wrote 3072 bytes (128 compressed) at 0x00008000 in 0.0 seconds (effective 3516.9 kbit/s)...
Hash of data verified.

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

Successfully uploading code to the ESP32-S2 Thing Plus. (Click to enlarge)

Note: Once Espressif releases a new Arduino core supporting the ESP32-S2, users will need to uninstall (and possibly manually delete the installed files), from the instructions above; before re-installing the latest board definitions.

Install the latest **ESP32** board definitions in the Arduino IDE.

Installing Board Definitions in the Arduino IDE

SEPTEMBER 9, 2020

How do I install a custom Arduino board/core? It's easy! This tutorial will go over how to install an Arduino board definition using the Arduino Board Manager. We will also go over manually installing third-party cores, such as the board definitions required for many of the SparkFun development boards.

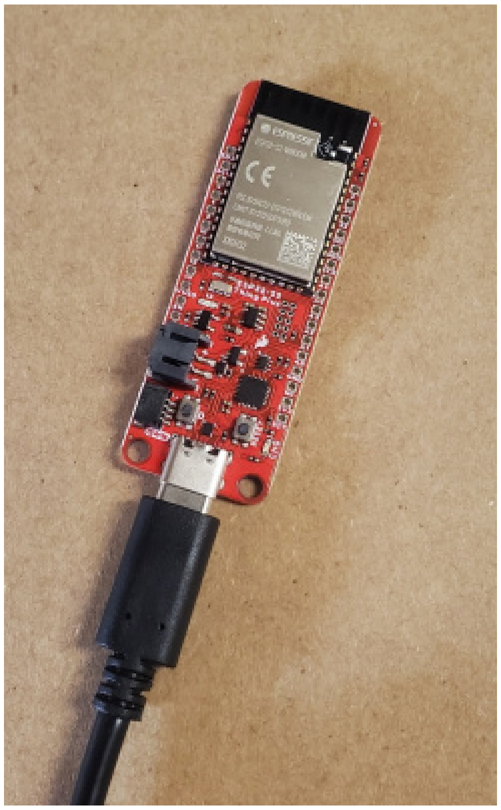
Note: For more instructions, users can follow this tutorial on Installing Additional Cores provided by Arduino. Users will also need the `.json` file for the Espressif Arduino Core:

```
https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json
```

Hardware Assembly

USB Programming

The USB connection is utilized for programming and serial communication. Users only need to plug their ESP32-S2 Thing Plus into a computer using a USB-C cable.


The ESP32-S2 Thing Plus with USB-C cable attached. (Click to enlarge)

Battery

For remote IoT applications, a Li-Po battery can be connected. Additionally, users may be interested in utilizing a solar panel and USB-C cable to recharge their battery.

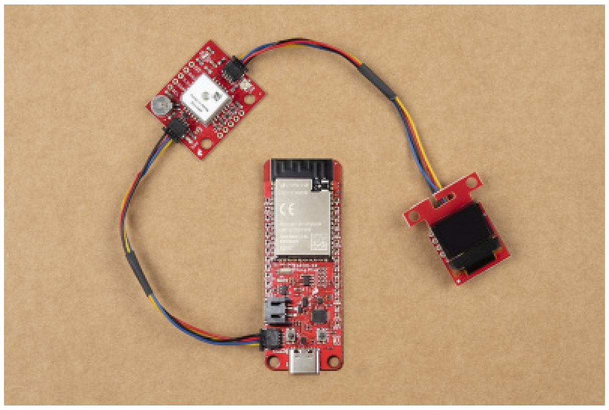
The ESP32-S2 Thing Plus with a battery connected. (Click to enlarge)

Solar Panel Charger - 10W
◉ PRT-16835

USB 3.1 Cable A to C - 3 Foot
◉ CAB-14743

Qwiic Devices

The Qwiic system allows users to effortlessly prototype with a Qwiic compatible I^{2}C device without soldering. Users can attach any Qwiic compatible sensor or board, with just a Qwiic cable. (**The example below, is for demonstration purposes and is not pertinent to the board functionality or this tutorial.*)


The Qwiic XA110 GPS breakout board and Micro OLED board connected to the ESP32-S2 Thing Plus.

Arduino Examples

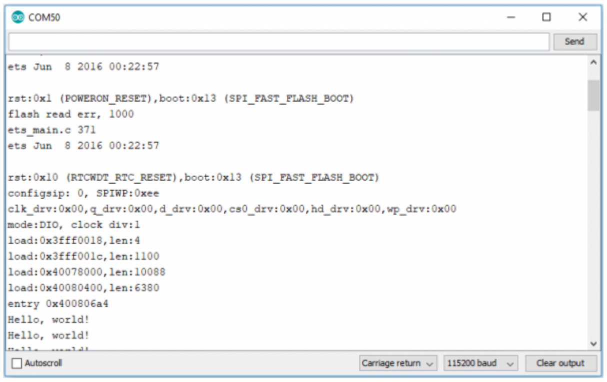
Blink

To verify the toolchain and board are properly set up on their computer, users can upload the simplest of sketches -- Blink! The LED attached to **GPIO 13** is perfect for this test. Plus, with the ESP32 attached to your computer, this is a good time to test out serial communication. Copy and paste the example sketch below into a fresh Arduino sketch:

```
int ledPin = 13;

void setup()
{
    pinMode(ledPin, OUTPUT);
    Serial.begin(115200);
}

void loop()
{
    Serial.println("Hello, world!");
    digitalWrite(ledPin, HIGH);
    delay(500);
    digitalWrite(ledPin, LOW);
    delay(500);
}
```

With everything setup correctly, upload the code! Once the code finishes transferring, **open the serial monitor** and set the baud rate to **115200**. Users should see `Hello, world!` 's begin to fly by.

If the blue LED remains dimly lit, it's probably still sitting in the bootloader. After uploading a sketch, users may need to **tap the `RESET` button** to get your ESP32-S2 Thing Plus to begin running the sketch.



```
ets Jun  8 2016 00:22:57

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
flash read err, 1000
ets_main.c 371
ets Jun  8 2016 00:22:57

rst:0x10 (RTCWDT_RTC_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0018,len:4
load:0x3fff001c,len:1100
load:0x40078000,len:10088
load:0x40080400,len:6380
entry 0x400806a4
Hello, world!
Hello, world!
```

When the ESP32 boots up, it prints out a long sequence of debug messages. These are emitted every time the chip resets -- always at **115200 baud**.

WiFi

The ESP32 Arduino core includes a handful of WiFi examples, which demonstrate everything from scanning for nearby networks to sending data to a client server. Users can find the examples under the **File** > **Examples** > **WiFi** menu.

Here's another example using the WiFi library, which demonstrates how to connect to a nearby WiFi network and poll a remote domain (http://example.com/) as a client.

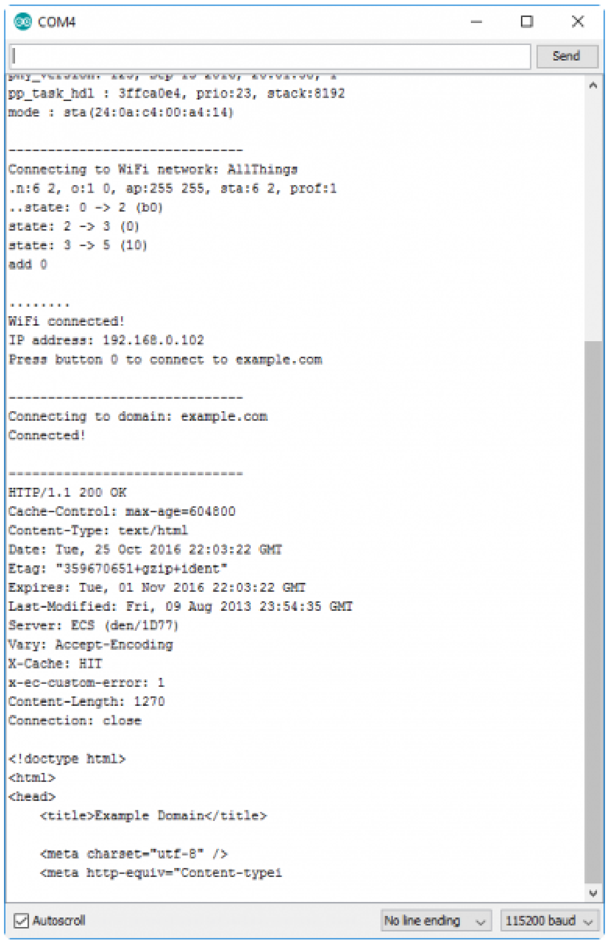
Note: Connect to the 2.4GHz band your wireless router; the ESP32 is not compatible with 5GHz signals.

```
#include <WiFi.h>

// WiFi network name and password:
const char * networkName = "YOUR_NETWORK_HERE";
const char * networkPswd = "YOUR_PASSWORD_HERE";

// Internet domain to request from:
const char * hostDomain = "example.com";
const int hostPort = 80;

const int BUTTON_PIN = 0;
const int LED_PIN = 13;

void setup()
{
  // Initilize hardware:
  Serial.begin(115200);
  pinMode(BUTTON_PIN, INPUT_PULLUP);
  pinMode(LED_PIN, OUTPUT);

  // Connect to the WiFi network (see function below loop)
  connectToWiFi(networkName, networkPswd);

  digitalWrite(LED_PIN, LOW); // LED off
  Serial.print("Press button 0 to connect to ");
  Serial.println(hostDomain);
}

void loop()
{
  if (digitalRead(BUTTON_PIN) == LOW)
  { // Check if button has been pressed
    while (digitalRead(BUTTON_PIN) == LOW)
      ; // Wait for button to be released

    digitalWrite(LED_PIN, HIGH); // Turn on LED
    requestURL(hostDomain, hostPort); // Connect to server
    digitalWrite(LED_PIN, LOW); // Turn off LED
  }
}

void connectToWiFi(const char * ssid, const char * pwd)
{
  int ledState = 0;

  printLine();
  Serial.println("Connecting to WiFi network: " + String(ssid));

  WiFi.begin(ssid, pwd);

  while (WiFi.status() != WL_CONNECTED)
  {
    // Blink LED while we're connecting:
```

```
    digitalWrite(LED_PIN, ledState);
    ledState = (ledState + 1) % 2; // Flip ledState
    delay(500);
    Serial.print(".");
  }

  Serial.println();
  Serial.println("WiFi connected!");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());
}

void requestURL(const char * host, uint8_t port)
{
  printLine();
  Serial.println("Connecting to domain: " + String(host));

  // Use WiFiClient class to create TCP connections
  WiFiClient client;
  if (!client.connect(host, port))
  {
    Serial.println("connection failed");
    return;
  }
  Serial.println("Connected!");
  printLine();

  // This will send the request to the server
  client.print((String)"GET / HTTP/1.1\r\n" +
               "Host: " + String(host) + "\r\n" +
               "Connection: close\r\n\r\n");
  unsigned long timeout = millis();
  while (client.available() == 0)
  {
    if (millis() - timeout > 5000)
    {
      Serial.println(">>> Client Timeout !");
      client.stop();
      return;
    }
  }

  // Read all the lines of the reply from server and print them to Serial
  while (client.available())
  {
    String line = client.readStringUntil('\r');
    Serial.print(line);
  }

  Serial.println();
  Serial.println("closing connection");
  client.stop();
}
```

```
void printLine()
{
  Serial.println();
  for (int i=0; i<30; i++)
    Serial.print("-");
  Serial.println();
}
```

Make sure to fill in the `networkName` and `networkPswd` variables with the name (or SSID) and password of your WiFi network! Then upload the code, open the **Serial Monitor**.



```
pp_task_hdl : 3ffca0e4, prio:23, stack:8192
mode : sta(24:0a:c4:00:a4:14)

------------------------------
Connecting to WiFi network: AllThings
.n:6 2, o:1 0, ap:255 255, sta:6 2, prof:1
..state: 0 -> 2 (b0)
state: 2 -> 3 (0)
state: 3 -> 5 (10)
add 0

........
WiFi connected!
IP address: 192.168.0.102
Press button 0 to connect to example.com

------------------------------
Connecting to domain: example.com
Connected!

------------------------------
HTTP/1.1 200 OK
Cache-Control: max-age=604800
Content-Type: text/html
Date: Tue, 25 Oct 2016 22:03:22 GMT
Etag: "359670651+gzip+ident"
Expires: Tue, 01 Nov 2016 22:03:22 GMT
Last-Modified: Fri, 09 Aug 2013 23:54:35 GMT
Server: ECS (den/1D77)
Vary: Accept-Encoding
X-Cache: HIT
x-ec-custom-error: 1
Content-Length: 1270
Connection: close

<!doctype html>
<html>
<head>
    <title>Example Domain</title>

    <meta charset="utf-8" />
    <meta http-equiv="Content-typei
```

After the ESP32-S2 connects to the WiFi network, it will wait for users to press the `0` button. Tapping that will cause the ESP32 to make an HTTP request to example.com. You should see a string of HTTP headers and HTML similar to the screenshot above.

Troubleshooting

Not working as expected and need help?

If you need technical assistance and more information on a product that is not working as you expected, we recommend heading on over to the SparkFun Technical Assistance page for some initial troubleshooting.

SPARKFUN TECHNICAL ASSISTANCE PAGE

If you can't find what you need there, you'll need a Forum Account to search product forums and post questions.

Resources and Going Further

- Schematic (PDF)
- Eagle Files (ZIP)
- Board Dimensions (PDF)
- Graphical Datasheet (PDF)
- Hardware Component Information:
  - ESP32-S2 Datasheet (PDF)
  - ESP32-S2 WROOM Module Datasheet (PDF)
  - ESP32-S2 Hardware Design Guidelines (PDF)
  - SparkFun Qwiic Connect System
- GitHub Hardware Repository
- Development Platforms for Artemis Module:
  - Espressif's ESP32 Arduino Core
    - `.json` file needed for Epressif's ESP32 Arduino Core: `https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json`
- SFE Product Showcase

For more inspiration, check out these other ESP32 tutorials:

ESP32 Environment Sensor Shield Hookup Guide

SparkFun's ESP32 Environment Sensor Shield provides sensors and hookups for monitoring environmental conditions. This tutorial will show you how to connect your sensor suite to the Internet and post weather data online.

ESP32 Thing Power Control Shield Hookup Guide

This tutorial shows you how to get started with the ESP32 Thing Power Control Shield.

ESP32 Thing Motion Shield Hookup Guide

SparkFun ESP32 DMX to LED Shield

Getting started with the ESP32 Thing Motion Shield to detect movements using the on-board LSM9DS1 IMU and adding a GPS receiver. Data can be easily logged by adding an microSD card to the slot.

Learn how to utilize your DMX to LED Shield in a variety of different ways.