Adafruit QT Py ESP32-C3 WiFi Dev Board

Created by lady ada

https://learn.adafruit.com/adafruit-qt-py-esp32-c3-wifi-dev-board

Last updated on 2022-03-19 12:20:06 PM EDT
Table of Contents

Overview 3

Pinouts 7
- Power 8
- ESP32-C3 Chip 9
- Logic Pins 10
- STEMMA QT Connector 11
- NeoPixel LED 11
- Buttons 12

Arduino IDE Setup 12

Arduino NeoPixel Blink 14
- Pre-Flight Check: Get Arduino IDE & Hardware Set Up 15
- Start up Arduino IDE and Select Board/Port 16
- Install NeoPixel Library 16
- New NeoPixel Blink Sketch 17
- Verify (Compile) Sketch 18
- Upload Sketch 20
- Native USB and manual bootloading 21
- Enter Manual Bootload Mode 22
- Finally, a Blink! 23

Arduino I2C Scan 23
- Common I2C Connectivity Issues 23
- Perform an I2C scan! 25
- Wiring the MCP9808 27

WiFi Test 28
- WiFi Connection Test 30
- Secure Connection Example 32
- JSON Parsing Demo 35

MicroPython Setup 39

Low Power Usage 42

Downloads 47
- Files: 47
- Schematic and Fab Print 47

©Adafruit Industries
What's life without a little RISC? This miniature dev board is perfect for small projects: it comes with our favorite connector - the STEMMA QT (https://adafruit.it/HMB), a chainable I2C port, WiFi, Bluetooth LE, and plenty of FLASH and RAM memory for many IoT projects. What a cutie pie! Or is it... a QT Py? This diminutive dev board comes with a RISC-V IoT microcontroller, the ESP32-C3!

ESP32-C3 is a low-cost microcontroller from Espressif that supports 2.4 GHz Wi-Fi and Bluetooth® Low Energy (Bluetooth LE). It has built-in USB-to-Serial, but not native USB - it cannot act as a keyboard or disk drive. The chip used here has 4MB of Flash memory, 400 KB of SRAM and can easily handle TLS connections.

The ESP32-C3 integrates a rich set of peripherals, ranging from UART, I2C, I2S, remote control peripheral, LED PWM controller, general DMA controller, TWAI controller, USB Serial/JTAG controller, temperature sensor, and ADC. It also includes SPI, Dual SPI, and Quad SPI interfaces. There is no DAC or native capacitive touch.

There's a minimum number of pins on this chip, it's specifically designed to be low cost and for simpler projects than ESP32-Sx or ESP32 classics with their large number of GPIO. Think of it more as an intended replacement to the ESP8266 than to the ESP32!
With its state-of-the-art power and RF performance, this SoC is an ideal choice for a wide variety of application scenarios relating to the Internet of Things (IoT) (https://adafru.it/Bwq), wearable electronics (https://adafru.it/Osb), and smart homes.

Please note: The C3 uses RISC V as a core, not Tensilica, and has Bluetooth LE (not classic!). The BLE core supports BT version 5 including Mesh

Pinout and shape are Seeed Xiao (https://adafru.it/NC3) compatible, with castellated pads. In addition to the QT connector, we also added an RGB NeoPixel, a reset button (great for restarting your program or entering the ROM bootloader), and a button on GPIO 9 for entering the ROM bootloader or for user input

Runs Arduino with Espressif’s ESP32 core (https://adafru.it/weF) and you can also run MicroPython on this chipset (https://adafru.it/Z1e).
OLEDs (https://adafru.it/NB)! Inertial Measurement Units (https://adafru.it/NC0)! Senso
rs a-plenty (https://adafru.it/NC1). All plug-and-play thanks to the innovative chainable
design: SparkFun Qwiic (https://adafru.it/Fpw)-compatible STEMMA QT (https://
adafru.it/Ft4) connectors for the I2C bus so you don’t even need to solder! Just plug
in a compatible cable and attach it to your MCU of choice, and you’re ready to load
up some software and measure some light. Seeed Grove I2C boards (https://adafru.it/
Ndk) will also work with this adapter cable.

- Same size, form-factor, and pin-out as Seeed Xiao
• USB Type C connector - If you have only Micro B cables, this adapter will come in handy (https://adafru.it/FQR)!
• ESP32-C3 32-bit RISC-V single core processor with 4MB of Flash memory, 400 KB of SRAM
• Built in USB-to-Serial inside the chip, which can also be used for JTAG programming. This peripheral is not native serial, so not for USB HID, MIDI or MSC: it does reduce cost since a separate converter isn't needed
• Can be used with Arduino IDE or MicroPython
• Built-in RGB NeoPixel LED note that due the small number of GPIO, we could not add a NeoPixel power pin
• Battery input pads on underside with diode protection for external battery packs up to 6V input
• 13 GPIO pins:
  ◦ 11 on breakout pads
  ◦ 5 x 12-bit analog inputs on A0 thru A3 and SDA pin
  ◦ PWM outputs on any pin
  ◦ I2C port, on the breakout pads shared with the STEMMA QT plug-n-play connector
  ◦ Hardware UART which is also the hardware serial debug port
  ◦ Hardware SPI
  ◦ Hardware I2S on any pins
• 3.3V regulator with 600mA peak output (https://adafru.it/NC4)
• Light sleep at 500uA, deep sleep at ~300uA. Lower power deep-sleep is not possible because we don't have an extra GPIO for disabling the NeoPixel power and we didn't want to share that with any of the IO pads. An engineering trade-off!
• Reset switch for starting your project code over, boot 9 button for entering bootloader mode
• Really really small
The QT Py ESP32-C3 is tiny but packed with features. Here's a detailed look.
Power

USB-C port - This is used for both powering and programming the board. You can power it with any USB C cable.

3.3V - These pins are the output from the 3.3V regulator, they can supply 500mA peak.

GND - This is the common ground for all power and logic.

5V - This is 5v out from the USB port. You can also use the 5V pin as a voltage input but you must have some sort of diode (schottky, signal, power, really anything) between your external power source and this pin with anode to battery, cathode to 5V pin. There are two pads on the back, with the protection diode in series, so you could use those if you wish.

Note that you cannot power the USB port by supplying 5V to this pin: there is a protection diode that prevents the 5V from reaching the USB connector (unless you bridge the jumper on the back!). This is to protect host computer USB ports, etc. You can draw 1A peak through the diode, but we recommend keeping it lower than that, about 500mA.
BAT/GND pads - On the back of the board are two pads labeled BAT and GND. These are the battery input pads with diode protection for external battery packs up to 6V input.

ESP32-C3 Chip

The ESP32-C3 is a highly-integrated, low-power, 2.4 GHz Wi-Fi and BLE System-on-Chip (SoC) solution. With its state-of-the-art power and RF performance, this SoC is an ideal choice for a wide variety of application scenarios relating to the Internet of Things (IoT) (https://adafruit.it/Bwq), wearable electronics (https://adafruit.it/Osb), and smart homes.

Please note, this is a single-core 160 MHz chip so it won’t be as fast as ESP32’s with dual-core. This chip has WiFi and Bluetooth Low Energy, there is no Bluetooth classic support. This chip comes with 4 MB flash and 400 KB of SRAM. No PSRAM
Logic Pins

There are eleven GPIO pins broken out to pads. There is hardware I2C, UART, and SPI. The UART pins are also the hardware UART debug port pins.

Five pads are 12-bit analog inputs (A0 thru A3 and SDA).

You can do PWM output or hardware I2S on any pin.

That's the general concept. Now for the details!

There are five analog pins.

- A0 A1 A2 A3 are ADC1 inputs, and are also GPIO #4, #3, #1, #0 in that order
- SDA is an ADC2 input

The I2C pins. These are shared by the STEMMA QT connector!

- SCL - This is the I2C clock pin. There is no pull-up on this pin, so for I2C please add an external pull-up if the breakout doesn't have one already. GPIO #6
- SDA - This is the I2C data pin. There is no pull-up on this pin, so for I2C please add an external pull-up if the breakout doesn't have one already. GPIO #5

The UART interface:

- RX - This is the UART receive pin. Connect to TX (transmit) pin on your sensor or breakout. GPIO #20
- TX - This is the UART transmit pin. Connect to RX (receive) pin on your sensor or breakout. This is also the hardware UART output so debug information can be emitted from this pad. GPIO #21

The SPI interface:

- SCK - This is the SPI clock pin. GPIO #10
- MI - This is the SPI Microcontroller In / Sensor Out pin. GPIO #8
• MO - This is the SPI Microcontroller Out / Sensor In pin. GPIO #7

STEMMA QT Connector

This JST SH 4-pin STEMMA QT (https://adafru.it/Ft4) connector breaks out the I2C interface (SCL, SDA, 3.3V, GND). It allows you to connect to various breakouts and sensors with STEMMA QT connectors (https://adafru.it/Qgf) or to other things using assorted associated accessories (https://adafru.it/Ft6). It works great with any STEMMA QT or Qwiic sensor/device. You can also use it with Grove I2C devices thanks to this handy cable (https://adafru.it/Ndk).

The ESP32-C3 does not have enough pins for a secondary I2C port, so the STEMMA QT connector IO pins are the same as SDA and SCL!

NeoPixel LED

Near the SCK and MI on the silk, is the RGB NeoPixel LED. This addressable LED works both as a status LED (in CircuitPython), or can be controlled with code. It is available in CircuitPython as `board.NEOPIXEL`, and in Arduino as `PIN_NEOPIXEL`. It is on GPIO #2
Buttons

There are two buttons.

Reset button - This button restarts the board and helps enter the ROM bootloader. You can click it once to reset the board without unplugging the USB cable or battery.

Boot button - This button can be used to put the board into ROM bootloader mode. To enter ROM bootloader mode, hold down boot button while clicking reset button mentioned above. When in the ROM bootloader, you can upload code and query the chip using **esptool**. This button can also be read as GPIO 9 (set it to be an input-with-pullup)

---

Arduino IDE Setup

We primarily recommend using the ESP32 chipsets with Arduino. Don't forget you will also need to install the SiLabs CP2104 Driver if you are using an ESP32 board with USB-to-Serial converter! (There's no harm in doing it, so we recommend even if you aren't)

Install Arduino IDE

The first thing you will need to do is to download the latest release of the Arduino IDE. You will need to be using version 1.8 or higher for this guide

[Arduino IDE Download](https://adafru.it/f1P)

Install CP2104 / CP2102N USB Driver

Many ESP32 boards have a USB-to-Serial converter that talks to the chip itself, and will need a driver on your computer's operating system. The driver is available for Mac, Windows and Linux.
Install ESP32 Board Support Package from GitHub

For this board we recommend you don't use 'release' version of Espressif's board support package because the current release doesn't include board support.

Instead we will install the "very latest" by following these instructions (https://adafru.it/YYB) (scroll down for mac and linux as well)

Basically, install by git cloning the espressif esp32 board support to get the very latest version of the code

In the Tools → Board submenu you should see ESP32 Arduino (in sketchbook) and in that dropdown it should contain the ESP32 boards along with all the latest ESP32 boards.

Look for the board called Adafruit QT Py ESP32-C3 watch out you don't get confused by the other QT Py boards!
The upload speed can be changed: faster speed makes uploads take less time but sometimes can cause upload issues. 921600 should work fine, but if you're having issues, you can drop down lower.

Arduino NeoPixel Blink

The first and most basic program you can upload to your Arduino is the classic Blink sketch. This takes something on the board and makes it, well, blink! On and off. It's a great way to make sure everything is working and you're uploading your sketch to the right board and right configuration.

When all else fails, you can always come back to Blink!

Now traditionally you would use an onboard LED to make a blink occur. However, this board does not have an onboard single-color LED, so we will 'mimic' the same blink sketch but instead of using a digital output pin, we will use NeoPixel support to blink the onboard RGB LED!
Pre-Flight Check: Get Arduino IDE & Hardware Set Up

This lesson assumes you have Arduino IDE set up. This is a generalized checklist, some elements may not apply to your hardware. If you haven't yet, check the previous steps in the guide to make sure you:

- Install the very latest Arduino IDE for Desktop (not all boards are supported by the Web IDE so we don't recommend it)
- Install any board support packages (BSP) required for your hardware. Some boards are built in defaults on the IDE, but lots are not! You may need to install plug-in support which is called the BSP.
- Get a Data/ Sync USB cable for connecting your hardware. A significant amount of problems folks have stem from not having a USB cable with data pins. Yes, these cursed cables roam the land, making your life hard. If you find a USB cable that doesn't work for data/sync, throw it away immediately! There is no need to keep it around, cables are very inexpensive these days.
- Install any drivers required - If you have a board with a FTDI or CP210x chip, you may need to get separate drivers. If your board has native USB, it probably doesn't need anything. After installing, reboot to make sure the driver sinks in.
- Connect the board to your computer. If your board has a power LED, make sure its lit. Is there a power switch? Make sure its turned On!
The Adafruit QT Py ESP32-C3 has a single NeoPixel on GPIO #2. Unlike some other QT Py boards, there were not enough pins to add a NeoPixel power pin, so the NeoPixel will always be powered.

Start up Arduino IDE and Select Board/Port

OK now you are prepared! Open the Arduino IDE on your computer. Now you have to tell the IDE what board you are using, and how you want to connect to it.

In the IDE find the Tools menu. You will use this to select the board. If you switch boards, you must switch the selection! So always double-check before you upload code in a new session.

Install NeoPixel Library

Despite their popularity, NeoPixel RGB LEDs are not supported 'out of the box' in Arduino IDE! You will need to add support by installing the library. Good news it is very easy to do it. Go to the Library Manager here

©Adafruit Industries
Search for and install the Adafruit NeoPixel library. It might not be first in the list so make sure you get the name matched up right!

New NeoPixel Blink Sketch

OK lets make a new blink sketch! From the File menu, select New

Then in the new window, copy and paste this text:

```cpp
#include <Adafruit_NeoPixel.h>

// How many internal neopixels do we have? some boards have more than one!
#define NUMPIXELS 1

Adafruit_NeoPixel pixels(NUMPIXELS, PIN_NEOPIXEL, NEO_GRB + NEO_KHZ800);

// the setup routine runs once when you press reset:
void setup() {
  Serial.begin(115200);
  pixels.begin(); // INITIALIZE NeoPixel strip object (REQUIRED)
  pixels.setBrightness(20); // not so bright
}

// the loop routine runs over and over again forever:
void loop() {
  // say hi
  Serial.println("Hello!");
```
// set color to red
pixels.fill(0xFF0000);
pixels.show();
delay(500); // wait half a second

// turn off
pixels.fill(0x000000);
pixels.show();
delay(500); // wait half a second

Note that in this example, we are not only blinking the NeoPixel LED but also printing to the Serial monitor, think of it as a little bonus to test the serial connection.

One note you'll see is that we reference the LED with the constant `PIN_NEOPIXEL` rather than a number. That's because each board could have the built in NeoPixels on a different pin and this makes the code a little more portable!

Verify (Compile) Sketch

OK now you can click the Verify button to convert the sketch into binary data to be uploaded to the board.

Note that Verifying a sketch is the same as Compiling a sketch - so we will use the words interchangeably.

During verification/compilation, the computer will do a bunch of work to collect all the libraries and code and the results will appear in the bottom window of the IDE.
If something went wrong with compilation, you will get red warning/error text in the bottom window letting you know what the error was. It will also highlight the line with an error.

For example, here I had the wrong board selected - and the selected board does not have a built in NeoPixel pin defined!

Here's another common error, in my haste I forgot to add a ; at the end of a line. The compiler warns me that it's looking for one - note that the error is actually a few lines up!

Turning on detailed compilation warnings and output can be very helpful sometimes - Its in Preferences under "Show Verbose Output During:" and check the Compilation button. If you ever need to get help from others, be sure to do...
On success you will see something like this white text output and the message Done compiling. in the message area.

![Screen capture of IDE showing compilation success]

### Upload Sketch

Once the code is verified/compiling cleanly you can upload it to your board. Click the Upload button.

The IDE will try to compile the sketch again for good measure, then it will try to connect to the board and upload a the file.

This is actually one of the hardest parts for beginners because it's where a lot of things can go wrong.

However, let's start with what it looks like on success! Here's what your board upload process looks like when it goes right:

![Screen capture of IDE showing successful upload]

Often times you will get a warning like this, which is kind of vague:
No device found on COM66 (or whatever port is selected)
An error occurred while uploading the sketch

This could be a few things.

First up, check again that you have the correct board selected! Many electronics boards have very similar names or look, and often times folks grab a board different from what they thought.

If you’re positive the right board is selected, we recommend the next step is to put the board into manual bootloading mode.

Native USB and manual bootloading

Historically, microcontroller boards contained two chips: the main micro chip (say, ATmega328 or ESP8266 or ESP32) and a separate chip for USB interface that would be used for bootloading (a CH430, FT232, CP210x, etc). With these older designs, the microcontroller is put into a bootloading state for uploading code by the separate chip. It allows for easier uploading but is more expensive as two chips are needed, and also the microcontroller can’t act like a keyboard or disk drive.

Modern chips often have 'native' USB - that means that there is no separate chip for USB interface. It's all in one! Great for cost savings, simplicity of design, reduced size and more control. However, it means the chip must be self-aware enough to be able to put itself into bootload/upload mode on its own. That's fine 99% of the time but is very likely you will at some point get the board into an odd state that makes it too confused to bootload.

A lot of beginners have a little freakout the first time this happens, they think the board is ruined or ‘bricked’ - it’s almost certainly not, it is just crashed and/or confused. You may need to perform a little trick to get the board back into a good state, at which point you won’t need to manually bootloader again.

Before continuing we really really suggest turning on Verbose Upload messages, it will help in this process because you will be able to see what the IDE is trying to do. It's a checkbox in the Preferences menu.
Enter Manual Bootload Mode

OK now you know it's probably time to try manual bootloading. No problem! Here is how you do that for this board:

1. Have the QT Py C3 plugged in
2. Press and hold down the BOOT button
3. Press and release the RESET button (while the BOOT button is held down)

Once you are in manual bootload mode, go to the Tools menu, and make sure you have selected the bootloader serial port. It is almost certain that the serial port has changed now that the bootloader is enabled.

Now you can try uploading again!

Did you remember to select the new Port in the Tools menu since the bootloader port has changed?
This time, you should have success!

After uploading this way, be sure to click the reset button - it sort of makes sure that the board got a good reset and will come back to life nicely.

After uploading with Manual Bootloader - don't forget to re-select the old Port again

It's also a good idea to try to re-upload the sketch again now that you've performed a manual bootloader to get the chip into a good state. It should perform an auto-reset the second time, so you don't have to manually bootloader again

Press the reset button, or unplug and replug USB to kickstart the blink sketch!

Finally, a Blink!

OK it was a journey but now we're here and you can enjoy your blinking LED. Next up, try to change the delay between blinks and re-upload. It's a good way to make sure your upload process is smooth and practiced.

Arduino I2C Scan

'A lot of sensors, displays, and devices can connect over I2C. I2C is a 2-wire 'bus' that allows multiple devices to all connect on one set of pins so it's very convenient for wiring!

When using your board, you'll probably want to connect up I2C devices, and it can be a little tricky the first time. The best way to debug I2C is go through a checklist and then perform an I2C scan

Common I2C Connectivity Issues

- Have you connected four wires (at a minimum) for each I2C device? Power the device with whatever is the logic level of your microcontroller board (probably 3.3V), then a ground wire, and a SCL clock wire, and a SDA data wire.
- If you're using a STEMMA QT board - check if the power LED is lit. It's usually a green LED to the left side of the board.
• Does the STEMMA QT/I2C port have switchable power or pullups? To reduce power, some boards have the ability to cut power to I2C devices or the pullup resistors. Check the documentation if you have to do something special to turn on the power or pullups.
• If you are using a DIY I2C device, do you have pullup resistors? Many boards do not have pullup resistors built in and they are required! We suggest any common 2.2K to 10K resistors. You'll need two: one each connects from SDA to positive power, and SCL to positive power. Again, positive power (a.k.a VCC, VDD or V+) is often 3.3V.
• Do you have an address collision? You can only have one board per address. So you cannot, say, connect two AHT20's to one I2C port because they have the same address and will interfere. Check the sensor or documentation for the address. Sometimes there are ways to adjust the address.
• Does your board have multiple I2C ports? Historically, boards only came with one. But nowadays you can have two or even three! This can help solve the "hey but what if I want two devices with the same address" problem: just put one on each bus.
• Are you hot-plugging devices? I2C does not support dynamic re-connection, you cannot connect and disconnect sensors as you please. They should all be connected on boot and not change. (Only exception is if you're using a hot-plug assistant but that'll cost ya (https://adafruit.it/XBY))
• Are you keeping the total bus length reasonable? I2C was designed for maybe 6" max length. We like to push that with plug-n-play cables but really please keep them as short as possible! (Only exception is if you're using an active bus extender (https://adafruit.it/XBZ)).

The QT Py ESP32-C3 does not have built-in pullup resistors. Also, unlike some other QT Py boards, the Stemma QT connector is the "same" I2C pins as the side breakout pads. So either way, use Wire to connect to I2C peripherals.
Perform an I2C scan!

Install TestBed Library

To scan I2C, the Adafruit TestBed library is used. This library and example just makes the scan a little easier to run because it takes care of some of the basics. You will need to add support by installing the library. Good news: it is very easy to do it. Go to the Arduino Library Manager.

Search for TestBed and install the Adafruit TestBed library
Now open up the I2C Scan example

```c
#include <Adafruit_TestBed.h>
extern Adafruit_TestBed TB;

#define DEFAULT_I2C_PORT &Wire

ifdef(ARDUINO_ADAFRUIT_KB2040_RP2040) 
|| defined(ARDUINO_ADAFRUIT_ITSYBITSY_RP2040) 
|| defined(ARDUINO_ADAFRUIT_QTPY_RP2040) 
|| defined(ARDUINO_ADAFRUIT_FEATHER_RP2040) 
|| defined(ARDUINO_ADAFRUIT_QTPY_ESP32S2) 
|| defined(ARDUINO_ADAFRUIT_QTPY_ESP32_PICO)
#define SECONDARY_I2C_PORT &Wire1
endif

void setup() {
  Serial.begin(115200);

  // Wait for Serial port to open
  while (!Serial) {
    delay(10);
  }
  delay(500);
  Serial.println("Adafruit I2C Scanner");

  #ifdef(ARDUINO_ADAFRUIT_QTPY_ESP32S2) || 
  #ifdef(ARDUINO_ADAFRUIT_QTPY_ESP32_PICO) // ESP32 is kinda odd in that secondary ports must be manually 
  // assigned their pins with setPins()!
    Wire1.setPins(SDA1, SCL1);
  #endif
  #endif

  #ifdef(ARDUINO_ADAFRUIT_FEATHER_ESP32S2) // turn on the I2C power by pulling pin to opposite of 'rest state'
    pinMode(PIN_I2C_POWER, INPUT);
    delay(1);
    bool polarity = digitalRead(PIN_I2C_POWER);
    pinMode(PIN_I2C_POWER, OUTPUT);
    digitalWrite(PIN_I2C_POWER, !polarity);
  #endif

  #ifdef(ARDUINO_ADAFRUIT_FEATHER_ESP32S2_TFT)
    pinMode(TFT_I2C_POWER, OUTPUT);
    digitalWrite(TFT_I2C_POWER, HIGH);
  #endif

  #ifdef(ADAARUINO_FEATHER_ESP32_V2)
    // Turn on the I2C power by pulling pin HIGH.
    pinMode(NEOPixel_I2C_POWER, OUTPUT);
    digitalWrite(NEOPixel_I2C_POWER, HIGH);
  #endif
}
```

©Adafruit Industries
void loop() {
    Serial.println("\n");
    Serial.println("\n");
    Serial.print("Default port ");
    TB.theWire = DEFAULT_I2C_PORT;
    TB.printI2CBusScan();
#if defined(SECONDARY_I2C_PORT)
    Serial.print("Secondary port ");
    TB.theWire = SECONDARY_I2C_PORT;
    TB.printI2CBusScan();
#endif
    delay(3000); // wait 3 seconds
}

Wire up I2C device

While the examples here will be using the Adafruit MCP9808 (https://adafru.it/Trb), a high accuracy temperature sensor, the overall process is the same for just about any I2C sensor or device.

The first thing you'll want to do is get the sensor connected so your board has I2C to talk to.

Adafruit MCP9808 High Accuracy I2C Temperature Sensor Breakout
The MCP9808 digital temperature sensor is one of the more accurate/precise we've ever seen, with a typical accuracy of ±0.25°C over the sensor's -40°C to...
https://www.adafruit.com/product/5027

Wiring the MCP9808

The MCP9808 comes with a STEMMA QT connector, which makes wiring it up quite simple and solder-free.
Now upload the scanning sketch to your microcontroller and open the serial port to see the output. You should see something like this:

```
<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>21:57:52.436 -&gt; Adafruit I2C Scanner</td>
</tr>
<tr>
<td>21:57:52.436 -&gt;</td>
</tr>
<tr>
<td>21:57:52.436 -&gt;</td>
</tr>
<tr>
<td>21:57:52.436 -&gt; Default port I2C scan: 0x37,</td>
</tr>
<tr>
<td>21:57:52.436 -&gt;</td>
</tr>
<tr>
<td>21:57:52.436 -&gt;</td>
</tr>
<tr>
<td>21:57:52.436 -&gt; Default port I2C scan: 0x37,</td>
</tr>
</tbody>
</table>

WiFi Test

Thanksfully if you have ESP32 sketches, they'll 'just work' with variations of ESP32. You can find a wide range of examples in the File->Examples->Examples for Adafruit Metro ESP32-S2 subheading (the name of the board may vary so it could be "Examples for Adafruit Feather ESP32 V2" etc)

Let's start by scanning the local networks.
Load up the WiFiScan example under Examples->Examples for YOUR BOARDNAME->WiFi->WiFiScan

And upload this example to your board. The ESP32 should scan and find WiFi networks around you.

For ESP32, open the serial monitor, to see the scan begin.

For ESP32-S2 and -C3, don't forget you have to click Reset after uploading through the ROM bootloader. Then select the new USB Serial port created by the ESP32-S2. It will take a few seconds for the board to complete the scan.

If you can not scan any networks, check your power supply. You need a solid power supply in order for the ESP32 to not brown out. A skinny USB cable or drained battery can cause issues.
WiFi Connection Test

Now that you can scan networks around you, it’s time to connect to the Internet!

Copy the example below and paste it into the Arduino IDE:

```
#include <WiFi.h>

char ssid[] = "YOUR_SSID"; // your network SSID (name)
char pass[] = "YOUR_SSID_PASSWORD"; // your network password (use for WPA, or use as key for WEP)
int keyIndex = 0; // your network key Index number (needed only for WEP)
int status = WL_IDLE_STATUS;

void setup() {
    // Initialize serial and wait for port to open:
    Serial.begin(115200);
    while (!Serial) {
        ; // wait for serial port to connect. Needed for native USB port only
    }

    // attempt to connect to WiFi network:
    Serial.print("Attempting to connect to SSID: ");
    Serial.println(ssid);
    WiFi.begin(ssid, pass);
```
while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print('.');
}
Serial.println('');
Serial.println("Connected to WiFi");
printWifiStatus();
Serial.println("Starting connection to server...");
// if you get a connection, report back via serial:
if (client.connect(server, 80)) {
    Serial.println("connected to server");
    // Make a HTTP request:
    client.print("GET "); client.print(path); client.println(" HTTP/1.1");
    client.print("Host: "); client.println(server);
    client.println("Connection: close");
    client.println();
}
}

void loop() {
  // if there are incoming bytes available
  // from the server, read them and print them:
  while (client.available()) {
    char c = client.read();
    Serial.write(c);
  }

  // if the server's disconnected, stop the client:
  if (!client.connected()) {
    Serial.println();
    Serial.println("disconnecting from server.");
    client.stop();
    // do nothing forevermore:
    while (true) {
      delay(100);
    }
  }
}

void printWifiStatus() {
  // print the SSID of the network you're attached to:
  Serial.print("SSID: ");
  Serial.println(WiFi.SSID());

  // print your board's IP address:
  IPAddress ip = WiFi.localIP();
  Serial.print("IP Address: ");
  Serial.println(ip);

  // print the received signal strength:
  long rssi = WiFi.RSSI();
  Serial.print("signal strength (RSSI): ");
  Serial.print(rssi);
  Serial.println(" dBm");
}

NOTE: You must change the **SECRET_SSID** and **SECRET_PASS** in the example code to your WiFi SSID and password before uploading this to your board.
After you've set it correctly, upload and check the serial monitor. You should see the following. If not, go back, check wiring, power and your SSID/password

```
Attempting to connect to SSID: Transit
........
Connected to WiFi
SSID: Transit
IP Address: 192.168.1.182
signal strength (RSSI):-57 dBm

Starting connection to server...
connected to server
HTTP/1.1 200 OK
Server: nginx/1.10.3 (Ubuntu)
Date: Wed, 11 Nov 2020 20:51:30 GMT
Content-Type: text/html
Content-Length: 70
Last-Modified: Thu, 16 May 2019 18:21:16 GMT
Connection: close
ETag: "5cdda1c-46"
Accept-Ranges: bytes

This is a test of Adafruit WiFi!
If you can read this, its working :)

disconnecting from server.
```

Secure Connection Example

Many servers today do not allow non-SSL connectivity. Lucky for you the ESP32 has a great TLS/SSL stack so you can have that all taken care of for you. Here's an example of using a secure WiFi connection to connect to the Twitter API.

Copy and paste it into the Arduino IDE:

```
// SPDX-FileCopyrightText: 2015 Arturo Guadalupi
// SPDX-FileCopyrightText: 2020 Brent Rubell for Adafruit Industries
//
// SPDX-License-Identifier: MIT
/*
This example creates a client object that connects and transfers data using always SSL.
```
It is compatible with the methods normally related to plain connections, like client.connect(host, port).

Written by Arturo Guadalupi
last revision November 2015

*/

#include <WiFiClientSecure.h>

// Enter your WiFi SSID and password
char ssid[] = "YOUR SSID";  // your network SSID (name)
char pass[] = "YOUR SSID_PASSWORD";  // your network password (use for WPA, or use as key for WEP)
int keyIndex = 0;  // your network key index number (needed only for WEP)

int status = WL_IDLE_STATUS;  // if you don't want to use DNS (and reduce your sketch size)
// use the numeric IP instead of the name for the server:
// IPAddress server(74,125,232,128);  // numeric IP for Google (no DNS)
#define SERVER "cdn.syndication.twimg.com"
#define PATH   "/widgets/followbutton/info.json?screen_names=adafruit"

// Initialize the SSL client library
// with the IP address and port of the server
// that you want to connect to (port 443 is default for HTTPS):
WiFiClientSecure client;

void setup() {
  // Initialize serial and wait for port to open:
  Serial.begin(115200);
  while (!Serial) {
    // wait for serial port to connect. Needed for native USB port only
  }

  // attempt to connect to WiFi network:
  Serial.print("Attempting to connect to SSID: ");
  Serial.println(ssid);
  WiFi.begin(ssid, pass);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.println("\nConnected to WiFi");
  printWifiStatus();

  client.setInsecure();  // don't use a root cert

  Serial.println("\nStarting connection to server...");
  if (client.connect(SERVER, 443)) {
    Serial.println("connected to server");
    // Make a HTTP request:
    client.println("GET " PATH " HTTP/1.1");
    client.println("Host: " SERVER);
    client.println("Connection: close");
    client.println();
  }
}

uint32_t bytes = 0;

void loop() {
// if there are incoming bytes available
// from the server, read them and print them:
while (client.available()) {
    char c = client.read();
    Serial.write(c);
    bytes++;
}

// if the server's disconnected, stop the client:
if (!client.connected()) {
    Serial.println();
    Serial.println("disconnecting from server.");
    client.stop();
    Serial.print("Read "); Serial.print(bytes); Serial.println(" bytes");
    // do nothing forevermore:
    while (true);
}

void printWiFiStatus() {
    // print the SSID of the network you're attached to:
    Serial.print("SSID: ");
    Serial.println(WiFi.SSID());

    // print your board's IP address:
    IPAddress ip = WiFi.localIP();
    Serial.print("IP Address: ");
    Serial.println(ip);

    // print the received signal strength:
    long rssi = WiFi.RSSI();
    Serial.print("signal strength (RSSI): ");
    Serial.print(rssi);
    Serial.println(" dBm");
}

As before, update the ssid and password first, then upload the example to your board.

Note we use WiFiClientSecure client instead of WiFiClient client; to require a SSL connection! This example will connect to a twitter server to download a JSON snippet that says how many followers adafruit has
This example is a little more advanced - many sites will have API's that give you JSON data. We will build on the previous SSL example to connect to twitter and get that JSON data chunk

Then we'll use ArduinoJSON (https://adafruit.it/Evn) to convert that to a format we can use and then display that data on the serial port (which can then be re-directed to a display of some sort)

First up, use the Library manager to install ArduinoJSON (https://adafruit.it/Evo).

Then load the example JSONdemo by copying the code below and pasting it into your Arduino IDE.

```cpp
// SPDX-FileCopyrightText: 2014 Benoit Blanchon
// SPDX-FileCopyrightText: 2014 Arturo Guadalupi
// SPDX-FileCopyrightText: 2020 Brent Rubell for Adafruit Industries
//
// SPDX-License-Identifier: MIT

/*
This example creates a client object that connects and transfers data using always SSL, then shows how to parse a JSON document in an HTTP response.

It is compatible with the methods normally related to plain connections, like client.connect(host, port).
*/
```

©Adafruit Industries
```cpp
#include <WiFiClientSecure.h>
#include <ArduinoJson.h>

// uncomment the next line if you have a 128x32 OLED on the I2C pins
//#define USE_OLED
// uncomment the next line to deep sleep between requests
//#define USE_DEEPSLEEP

#if defined(USE_OLED)
  // Some boards have TWO I2C ports, how nifty. We should use the second one sometimes
  #if defined(ARDUINO_ADAFRUIT_QTPY_ESP32S2) \
    #| defined(ARDUINO_ADAFRUIT_QTPY_ESP32_PICO)
      #define OLED_I2C_PORT &Wire1
  #else
    #define OLED_I2C_PORT &Wire
  #endif
#endif

#include <Adafruit_SSD1306.h>
Adafruit_SSD1306 display = Adafruit_SSD1306(128, 32, OLED_I2C_PORT);
#endif

// Enter your WiFi SSID and password
char ssid[] = "YOUR_SSID"; // your network SSID (name)
char pass[] = "YOUR_SSID_PASSWORD"; // your network password (use for WPA, or
// use as key for WEP)
int keyIndex = 0; // your network key Index number (needed only for WEP)

int status = WL_IDLE_STATUS;
// if you don't want to use DNS (and reduce your sketch size)
// use the numeric IP instead of the name for the server:
// IPAddress server(74,125,232,128);  // numeric IP for Google (no DNS)
#define SERVER "cdn.syndication.twimg.com"
#define PATH "/widgets/followbutton/info.json?screen_names=adafruit"

void setup() {
  //Initialize serial and wait for port to open:
  Serial.begin(115200);

  // Connect to WPA/WPA2 network
  WiFi.begin(ssid, pass);

  #if defined(USE_OLED)
    setupI2C();
    delay(200); // wait for OLED to reset
    if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { // Address 0x3C for 128x32
      Serial.println(F("SSD1306 allocation failed"));
      for(;;); // Don't proceed, loop forever
    }
    display.display();
    display.setTextSize(1);
    display.setTextColor(WHITE);
    display.clearDisplay();
    display.setCursor(0,0);
  #else
    // Don't wait for serial if we have an OLED
    while (!Serial) {
      // wait for serial port to connect. Needed for native USB port only
      delay(10);
    }
```
// attempt to connect to Wifi network:
Serial.print("Attempting to connect to SSID: ");
Serial.println(ssid);
#if defined(USE_OLED)
  display.clearDisplay(); display.setCursor(0,0);
  display.print("Connecting to SSID\n"); display.println(ssid);
  display.display();
#endif
while (WiFi.status() != WL_CONNECTED) {
  delay(500);
  Serial.print(".");
}
Serial.println("\n");
Serial.println("Connected to WiFi");
#if defined(USE_OLED)
  display.print("...OK!\n");
  display.display();
#endif
printWifiStatus();
}

uint32_t bytes = 0;

void loop() {
  WiFiClientSecure client;
  client.setInsecure(); // don't use a root cert

  Serial.println("\nStarting connection to server...\n");
  #if defined(USE_OLED)
    display.clearDisplay(); display.setCursor(0,0);
    display.print("Connecting to "); display.println(SERVER);
    display.display();
  #endif

  // if you get a connection, report back via serial:
  if (client.connect(SERVER, 443)) {
    Serial.println("connected to server");
    // Make a HTTP request:
    client.print("GET \" PATH \" HTTP/1.1\"\n");
    client.println("Host: " SERVER);
    client.println("Connection: close\"\n");
  }

  // Check HTTP status
  char status[32] = {0};
  client.readBytesUntil(‘\r’, status, sizeof(status));
  if (strcmp(status, "HTTP/1.1 200 OK") != 0) {
    Serial.print(F("Unexpected response: "));
    Serial.println(status);
    #if defined(USE_OLED)
      display.print("Connection failed, code: "); display.println(status);
      display.display();
    #endif
    return;
  }

  // wait until we get a double blank line
  client.find("\n\n\n\n", 4);

  // Allocate the JSON document
Use arduinojson.org/v6/assistant to compute the capacity.

```cpp
const size_t capacity = JSON_ARRAY_SIZE(1) + JSON_OBJECT_SIZE(8) + 200;
DynamicJsonDocument doc(capacity);
```

Parse JSON object

```cpp
DeserializationError error = deserializeJson(doc, client);
if (error) {
    Serial.print(F("deserializeJson() failed: "));
    Serial.println(error.c_str());
    return;
}
```

Extract values

```cpp
JsonObject root_0 = doc[0];
Serial.println(F("Response:"));
const char* root_0_screen_name = root_0["screen_name"];
long root_0_followers_count = root_0["followers_count"];
Serial.print("Twitter username: "); Serial.println(root_0_screen_name);
Serial.print("Twitter followers: "); Serial.println(root_0_followers_count);
```

#endif

Disconnect

```cpp
client.stop();
delay(1000);
```

#if defined(USE_OLED)
    display.clearDisplay(); display.setCursor(0,0);
    display.setTextSize(2);
    display.println(root_0_screen_name);
    display.println(root_0_followers_count);
    display.display();
#endif

#else
    delay(10 * 1000);
#endif

void setupI2C()
{
    #if defined(ARDUINO_ADAFRUIT_QTPY_ESP32S2) ||
    defined(ARDUINO_ADAFRUIT_QTPY_ESP32_PICO)
        // ESP32 is kinda odd in that secondary ports must be manually
        // assigned their pins with setPins()!
        Wire1.setPins(SDA1, SCL1);
    #endif

    #if defined(NEOPIXEL_POWER)
        digitalWrite(NEOPIXEL_POWER, LOW); // off
    #elif defined(NEOPIXEL_I2C_POWER)
        digitalWrite(NEOPIXEL_I2C_POWER, LOW); // off
    #endif

    // turn on the I2C power by setting pin to opposite of 'rest state'
    pinMode(PIN_I2C_POWER, OUTPUT);
    digitalWrite(NEOPIXEL_I2C_POWER, HIGH); // on
    pinMode(NEOPIXEL_I2C_POWER, OUTPUT);
    digitalWrite(NEOPIXEL_I2C_POWER, HIGH); // on
```

```cpp
bool polarity = digitalRead(PIN_I2C_POWER);
```
pinMode(PIN_I2C_POWER, OUTPUT);
digitalWrite(PIN_I2C_POWER, !polarity);
#endif

void printWifiStatus()
{
// print the SSID of the network you're attached to:
Serial.print("SSID: ");
Serial.println(WiFi.SSID());

// print your board's IP address:
IPAddress ip = WiFi.localIP();
Serial.print("IP Address: ");
Serial.println(ip);

// print the received signal strength:
long rssi = WiFi.RSSI();
Serial.print("signal strength (RSSI): ");
Serial.print(rssi);
Serial.println(" dBm");
}

By default it will connect to the Twitter banner image API, parse the username and followers, and display them.

<table>
<thead>
<tr>
<th>Attempting to connect to SSID: Transit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connected to WiFi</td>
</tr>
<tr>
<td>SSID: Transit</td>
</tr>
<tr>
<td>IP Address: 192.168.1.182</td>
</tr>
<tr>
<td>signal strength (RSSI):-54 dBm</td>
</tr>
</tbody>
</table>

Starting connection to server...
connected to server
Response:
Twitter username: adafruit
Twitter followers: 176400

MicroPython Setup

You can also use this board with MicroPython and it's quite a nice experience.

This page has information and download for the most recent release of MicroPython for ESP32-C3 (https://adafru.it/Z1e) - just make sure you select 'USB' type since we are taking advantage of the USB port built into the C3!

You'll need esptool 3.1 or later to program the C3. Enter the ROM bootloader by holding down the BOOT button while clicking RESET button. Then upload to the serial port or COM port created.
We used a command like this to upload successfully:

```bash
esptool --chip esp32c3 --port COM3 --baud 115200 --before default_reset --after hard_reset --no-stub write_flash --flash_mode dio --flash_freq 80m 0x0 firmware.bin
```

To manage and run code, we used Thonny - setup to be a MicroPython (ESP32) board, then pick the matching serial port.

Install the `urequests` library via the Package Manager.
Download the following code and save it to your ESP32-C3 as `main.py`.

Don't forget to change `MY_SSID` and `MY_PASSWORD` to your WiFi access point.

```python
import time
import network
import urequests

station = network.WLAN(network.STA_IF)
station.active(True)

# Network settings
wifi_ssid = "MY_SSID"
wifi_password = "MY_PASSWORD"
url = "http://wifitest.adafruit.com/testwifi/index.html"

print("Scanning for WiFi networks, please wait...")
authmodes = ['Open', 'WEP', 'WPA-PSK', 'WPA2-PSK', 'WPA/WPA2-PSK']
for (ssid, bssid, channel, RSSI, authmode, hidden) in station.scan():
    print("* {:s}").format(ssid)
    print("   - Channel: {}".format(channel))
    print("   - RSSI: {}".format(RSSI))
    print("   - BSSID: {:02x}:{:02x}:{:02x}:{:02x}:{:02x}:{:02x}".ormat(*bssid))
print()

# Continually try to connect to WiFi access point
while not station.isconnected():
    # Try to connect to WiFi access point
    print("Connecting...")
    station.connect(wifi_ssid, wifi_password)
    time.sleep(10)

# Display connection details
print("Connected!")
print("My IP Address:", station.ifconfig()[0])

# Perform HTTP GET request on a non-SSL web
response = urequests.get(url)

# Display the contents of the page
print(response.text)
```
Now click Stop and Run to load the main.py and run it, you should get the following indicating it could connect to WiFi and read the data from our webserver.

Low Power Usage

This microcontroller board can be used for low power usage thanks to the ESP32's multiple sleep modes.

There are three basic operating states to Espressif chips: normal, light sleep and deep sleep.

Normal power usage is as you expect: you can use the chip and run code as you like - connecting to WiFi, reading sensors, etc.

Light sleep is sort of a 'hibernation' - power usage is minimal and WiFi is disconnected, but the internal clock and memory is kept. That means you can wake up where you left off, in the middle of the code as desired. You'll still need to re-
initialize any external hardware that got disconnected, and WiFi, but it's often faster than waking from a deep sleep

Deep sleep is the lowest power but the tradeoff is that all memory and state is lost - the only thing that's running is the real time clock that can wake the chip up. When woken up, the chip starts as if it was physically reset - from the beginning of the code. This can be beneficial if you want to have a fresh start each time

A rough guideline is:

- Normal power: 100mA+ can be as much power as need and spike during WiFi connection
- Light sleep: 2mA assuming all external hardware is de-powered
- Deep sleep: 100uA assuming all external hardware is de-powered

The ESP32-C3 does not have any spare pins so we could not add a NeoPixel power pin. The deep sleep current suffers for it but we think having a NeoPixel is so useful, its worth the current draw

Here's a generic sketch we use for all our boards that has a macro-defined section for enabling and disabling all external powered elements. For example, if there's a power pin for NeoPixels, I2C port, TFT, etc...we turn that off before going into light or deep sleep! This will minimize power usage

```cpp
// SPDX-FileCopyrightText: 2022 Limor Fried for Adafruit Industries
// // SPDX-License-Identifier: MIT

#include <Adafruit_NeoPixel.h>

// While we wait for Feather ESP32 V2 to get added to the Espressif BSP,
// we have to select PICO D4 and UNCOMMENT this line!
//#define ADAFRUIT_FEATHER_ESP32_V2

// then these pins will be defined for us
#if defined(ADAFRUIT_FEATHER_ESP32_V2)
#define PIN_NEOPIXEL 0
#define NEOPIXEL_I2C_POWER 2
#endif

#if defined(PIN_NEOPIXEL)
   Adafruit_NeoPixel pixel(1, PIN_NEOPIXEL, NEO_GRB + NEO_KHZ800);
#endif

void setup() {
   Serial.begin(115200);

   // Turn on any internal power switches for TFT, NeoPixels, I2C, etc!
   enableInternalPower();
}

void loop() {
```
LEDon();
delay(1000);

disableInternalPower();
LEDoff();
esp_sleep_enable_timer_wakeup(1000000); // 1 sec
esp_light_sleep_start();
// we'll wake from light sleep here

// wake up 1 second later and then go into deep sleep
esp_sleep_enable_timer_wakeup(1000000); // 1 sec
esp_deep_sleep_start();
// we never reach here
}

void LEDon() {
#if defined(PIN_NEOPIXEL)
    pixel.begin(); // INITIALIZE NeoPixel
    pixel.setBrightness(20); // not so bright
    pixel.setPixelColor(0, 0xFFFFFF);
    pixel.show();
#endif
}

void LEDoff() {
#if defined(PIN_NEOPIXEL)
    pixel.setPixelColor(0, 0x0);
    pixel.show();
#endif
}

void enableInternalPower() {
#if defined(NEOPIXEL_POWER)
    pinMode(NEOPIXEL_POWER, OUTPUT);
    digitalWrite(NEOPIXEL_POWER, HIGH);
#endif
#if defined(NEOPIXEL_I2C_POWER)
    pinMode(NEOPIXEL_I2C_POWER, OUTPUT);
    digitalWrite(NEOPIXEL_I2C_POWER, HIGH);
#endif
#if defined(ARDUINO_ADAFRUIT_FEATHER_ESP32S2)
    // turn on the I2C power by setting pin to opposite of 'rest state'
    pinMode(PIN_I2C_POWER, INPUT);
    delay(1);
    bool polarity = digitalRead(PIN_I2C_POWER);
    pinMode(PIN_I2C_POWER, OUTPUT);
    digitalWrite(PIN_I2C_POWER, !polarity);
    pinMode(NEOPIXEL_POWER, OUTPUT);
    digitalWrite(NEOPIXEL_POWER, HIGH);
#endif
}

void disableInternalPower() {
#if defined(NEOPIXEL_POWER)
    pinMode(NEOPIXEL_POWER, OUTPUT);
    digitalWrite(NEOPIXEL_POWER, LOW);
#endif
#if defined(NEOPIXEL_I2C_POWER)
    pinMode(NEOPIXEL_I2C_POWER, OUTPUT);
    digitalWrite(NEOPIXEL_I2C_POWER, LOW);
#endif
#if defined(ARDUINO_ADAFRUIT_FEATHER_ESP32S2)
    // turn on the I2C power by setting pin to rest state (off)
    pinMode(PIN_I2C_POWER, INPUT);
    pinMode(NEOPIXEL_POWER, OUTPUT);
#endif
The best way to really test power draw is with a specialty power meter such as the Nordic PPK 2.

The Power Profiler Kit II is a standalone unit, which can measure and optionally supply currents all the way from sub-uA and as high as 1A on all Nordic DKs, in... [https://www.adafruit.com/product/5048](https://www.adafruit.com/product/5048)

When running the above code and monitoring with a PPK, you'll get a graph like this:

The big pulse is normal mode, you can see the ESP32 booting up, loading code, and then pausing 1 second. Then there's a big drop for one sec to light sleep, and finally one more 1 second pause at deep sleep.
Normal Power Mode

The power draw when running normally (no WiFi) is 20mA.

Light Sleep Mode

The power draw in light sleep is 500uA.

Deep Sleep Mode
The power draw in deep sleep is 290uA.

Downloads

Files:

- ESP32-C3 datasheet (https://adafru.it/Zad)
- EagleCAD PCB files on GitHub (https://adafru.it/Zae)
- Fritzing object in the Adafruit Fritzing Library (https://adafru.it/Zaf)

Schematic and Fab Print