

MicroMod ESP32 Processor Board Hookup Guide

Introduction

Introducing the SparkFun MicroMod ESP32 Processor Board! This bad boy pops an M.2 connector onto the ESP32 so you can take advantage of all that lovely ESP32 power with any of our MicroMod carrier boards. Grab yourself an ESP32 MicroMod Processor Board and let's dive in!



SparkFun MicroMod ESP32 Processor
© WRL-16781

Product Showcase: SparkFun MicroMod Ecosystem





Required Materials

In addition to your ESP32 Processor Board, you'll need a carrier board to get started. Here we use the Input and Display Carrier Board, but there are a number of others you can choose from.



SparkFun MicroMod Machine Learning Carrier Board

● DEV-16400



SparkFun MicroMod ATP Carrier Board

● DEV-16885



SparkFun MicroMod Input and Display Carrier Board

● DEV-16985

You'll also need a USB-C cable to connect the Carrier to your computer and if you want to add some Qwiic breakouts to your MicroMod project you'll want at least one Qwiic cable to connect it all together. Below are some options for both of those cables:



SparkFun Qwiic Cable Kit

● KIT-15081



Qwiic Cable - 100mm

● PRT-14427



Reversible USB A to C Cable - 2m

● CAB-15424



USB 3.1 Cable A to C - 3 Foot

● CAB-14743

Depending on which Carrier Board you choose, you may need a few extra peripherals to take full advantage of them. Refer to the Carrier Boards' respective Hookup Guides for specific peripheral recommendations.

Suggested Reading

The SparkFun MicroMod ecosystem is a unique way to allow users to customize their project to their needs. Do you want to send your weather data via a wireless signal (eg. Bluetooth or WiFi)? There's a MicroMod processor for that. Looking to instead maximize efficiency and processing power? You guessed it, there's a MicroMod processor for that. If you are not familiar with the MicroMod system, take a look here:

MicroMod

MicroMod Ecosystem

We recommend taking a look through the following tutorials if you are not familiar with the concepts covered in them:



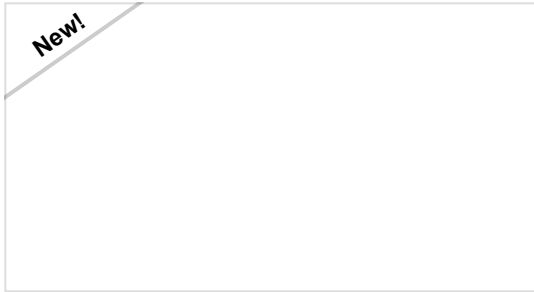
Getting Started with MicroMod

Dive into the world of MicroMod - a compact interface to connect a microcontroller to various peripherals via the M.2 Connector!



Designing with MicroMod

This tutorial will walk you through the specs of the MicroMod processor and carrier board as well as the basics of incorporating the MicroMod form factor into your own PCB designs!



SparkFun MicroMod Input and Display Carrier Board Hookup Guide

A short Hookup Guide to get started with the SparkFun MicroMod Input and Display Carrier Board

Hardware Overview

In this section we'll cover what's included on the MicroMod ESP32 Processor Board.

M.2 Connector

All of our MicroMod Processor boards come equipped with the **M.2 MicroMod Connector**, which leverages the M.2 standard and specification to allow you to install your MicroMod Processor board on your choice of carrier board.



M2 Connector from the Front



M2 Connector from the Back

Espressif ESP32

Ahhh, the Espressif ESP32. It's one of the most unique microcontrollers on the market. In its native form, it has a laundry list of features. On the MicroMod Processor Board, we include the following:

- Dual-core Tensilica LX6 microprocessor
- Up to 240MHz clock frequency

- 520kB internal SRAM
- Integrated 802.11 B/G/N WiFi transceiver
- Integrated dual-mode Bluetooth (classic and BLE)
- 2.7 to 3.6V operating range
- 500µA sleep current under hibernation
- 10-electrode capacitive touch support
- Hardware accelerated encryption (AES, SHA2, ECC, RSA-4096)
- 16MB Flash Storage



Stat LED



Wireless Antenna

Need wireless? The Espressif chip provides built-in BLE as well as a WiFi transceiver which sends and receives data through a 2.4GHz Antenna.



PinOut Notes

The ESP32 MicroMod has a few quirks. The ESP32's GPIO pins provide a lot of flexibility with what each pin can be used for. Whether it's I²C, I²S, SPI, UART, or PWM, the ESP32 MicroMod can do just about everything! However, with that flexibility and a fixed number of GPIO pins, the ESP32 isn't able to do it all at the same time. Below is a list of protocols the ESP32 supports, but pay close attention to the pins used, because some pins are assigned to two or possibly three functions.

Strapping Pins

One of the unique aspects of the ESP32 is the strapping pins. When the ESP32 comes out of reset, or as power is supplied, there are a few pins which control the behavior of the board. For a detailed description of these pins, check out the ESP32 Boot Mode Selection page on espressif's GitHub page. As a summary the strapping pins are:

GPIO 0

Having GPIO 0 pulled low as the ESP32 comes out of reset will enter the serial bootloader. Otherwise, the board will run the program stored in flash. On the MicroMod Processor, this pin is pulled high externally through a 10k resistor, and is connected to the boot button on the carrier boards, which can pull the pin low.

GPIO 2

Having GPIO 2 pulled high as the ESP32 comes out of reset will prevent the board from entering the serial bootloader. On the MicroMod Processor, this pin is connected to the status LED (active high) and does not interfere with the board from being able to enter the serial bootloader.

GPIO 12

If driven high, the flash voltage (VDD_SDIO) is set to **1.8V**. If unconnected or pulled low, VDD_SDIO is set to **3.3V**. The flash IC used on the MicroMod Processor has a minimum voltage of **2.7V**, which would create a brownout condition and might corrupt the data stored to the flash, or simply prevent the program from running. On the ESP32 MicroMod Processor, this pin is connected to PWM1.

GPIO 15

If driven low, the boot messages printed by the ROM bootloader (at 115200 baud) are silenced. If unconnected or driven high, the messages will be printed as they normally are. On the ESP32 MicroMod Processor, this pin is connected to G0.

I²C

We love us some I²C! We've broken out two I²C buses, which can be used with our Qwiic system. The main I²C bus has dedicated GPIO pins 21/22 connected to MicroMod pads 12/14, along with a dedicated interrupt pin connected to GPIO pin 4, which is connected to pad 16 of the MicroMod connector.

If you need a second I²C bus, the ESP32 uses GPIO pins 25/26 (pads 42/44 on the MicroMod) for SCL1 and SDA1.

Note: The secondary I²C bus is shared with G1 and G2, as well as the I²S bus pins for AUD_LRCLK and AUD_BCLK.

UART

The ESP32 Processor has two UARTs available. The primary UART has dedicated GPIO pins 1 and 3 which can be used for programming as well as printing debug messages to a terminal window. These GPIO pins aren't directly broken out, but instead are converted to USB which is connected to MicroMod pads 3 and 5.

The second UART is connected GPIO pins 16 and 17 (pads 19 and 17 on the MicroMod) for RX1 and TX1.

Note: The secondary UART is shared with G3 and G4, as well as the I²S bus pins for AUD_OUT, and AUD_IN.

GPIO/BUS

The MicroMod connector supports a total of 12 general purpose IO pins, 7 of which are used on the ESP32 Processor, on top of the 6 dedicated pins. The dedicated pins are just that, and are not shared with any other pin, unlike the general purpose pins which may be shared with other pins. The pins used are:

Dedicated Pins

- A0 - GPIO pin 34, pad 34 on the MicroMod (Input Only!)
- A1 - GPIO pin 35, pad 38 on the MicroMod (Input Only!)
- D0 - GPIO pin 14, pad 10 on the MicroMod
- D1 - GPIO pin 27, pad 18 on the MicroMod
- PWM0 - GPIO pin 13, pad 32 on the MicroMod
- PWM1 - GPIO pin 12, pad 47 on the MicroMod

General Purpose IO pins

- G0 - GPIO pin 15, pad 40 on the MicroMod
- G1 - GPIO pin 25, pad 42 on the MicroMod - Shared with the I²S bus, and secondary I²C bus.
- G2 - GPIO pin 26, pad 44 on the MicroMod - Shared with the I²S bus, and secondary I²C bus.
- G3 - GPIO pin 17, pad 46 on the MicroMod - Shared with the I²S bus, and secondary UART.
- G4 - GPIO pin 16, pad 48 on the MicroMod - Shared with the I²S bus, and secondary UART.
- G5 - GPIO pin 32, pad 73 on the MicroMod - Shared with the 32KHz RTC crystal.
- G6 - GPIO pin 33, pad 71 on the MicroMod - Shared with the 32KHz RTC crystal.

AUDIO

The ESP32 Processor supports audio using the I²S standard. The pins used are:

- AUD_OUT - GPIO pin 17, pad 56 on the MicroMod, this is the digital audio output.
- AUD_IN - GPIO pin 16, pad 54 on the MicroMod, this is the digital audio input.
- AUD_LRCLK - GPIO pin 25, pad 52 on the MicroMod. Officially called "word select", and also known as "frame sync".
- AUD_BCLK - GPIO pin 26, pad 50 on the MicroMod. Officially called "continuous serial clock, and also known as the "bit clock"

Note: The I²S bus is shared with the secondary UART, secondary I²C bus, and general purpose pins G1-G4.

SPI

The MicroMod standard supports two Serial Peripheral Interface (SPI) buses, but because of the limited GPIO pins here, only the primary SPI bus is used. This primary SPI bus is dedicated to the following pins:

- SCK - This is the clock pin, which is connected to GPIO 18, or MicroMod pad 57.
- SDO - This is the serial data output of the ESP32, which is connected to GPIO 23, or MicroMod pad 59.
- SDI - This is the serial data input of the ESP32, which is connected to GPIO 19, or MicroMod pad 61.
- #CS - This is the chip select pin, which is connected to GPIO 5, or MicroMod pad 55.

Note: You may not recognize the COPI/CIPO labels for SPI pins. SparkFun is working to move away from using MISO/MOSI to describe signals between the controller and the peripheral. Check out this page for more on our reasoning behind this change.

ESP32 MicroMod Processor Pin Functionality

ESP32 PROCESSOR BOARD PINOUT TABLE

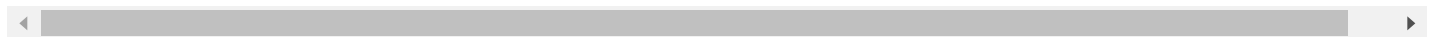
MICROMOD GENERAL PINOUT TABLE

MICROMOD GENERAL PIN DESCRIPTIONS

AUDIO	UART	GPIO/BUS	I ² C	SDIO	SPI0	Dedicated
-------	------	----------	------------------	------	------	-----------

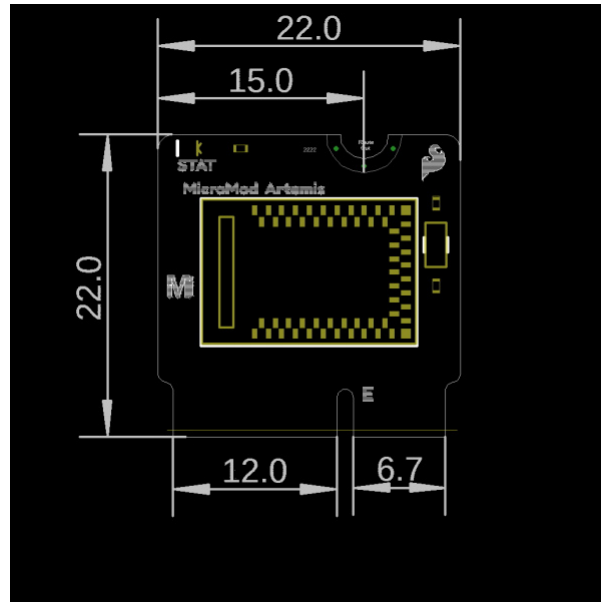
ESP32 Pin	Alternate Function	Primary Function	Bottom Pin	Top Pin	Primary Function	Alternate Function	ESP Pin
				73	G5	RTC	32
				71	G6	RTC	33
				61	SPI_CIPO		19
				59	SPI_COPI		23

					57	SPI_SCK			18
17	AUD_OUT	TX1	G3	56	55	SPI_CS#			5
16	AUD_IN	RX1	G4	54	53	SCL1	G1	AUD_LRCLK	25
25	AUD_LRCLK	SCL1	G1	52	51	SDA1	G2	AUD_BCLK	26
26	AUD_BCLK	SDA1	G2	50	49	BATT_VIN3			39
16	AUD_IN	RX1	G4	48	47	PWM1			12
17	AUD_OUT	TX1	G3	46					
26	AUD_BCLK	SDA1	G2	44					
25	AUD_LRCLK	SCL1	G1	42					
15			G0	40	39	GND			
35			A1	38					
34			A0	34					
13			PWM0	32					
					19	G4	RX1	AUD_IN	16
27		CAM_TRIG	D1	18	17	G3	TX1	AUD_OUT	17
4			I ² C Interrupt	16					
22			SCL	14					
21			SDA	12	11	BOOT			
14			D0	10	9	USB_VIN			
					7	GND			
			RESET	6	5	USB_D-			
					3	USB_D+			
			3.3V	2	1	GND			



Board Dimensions

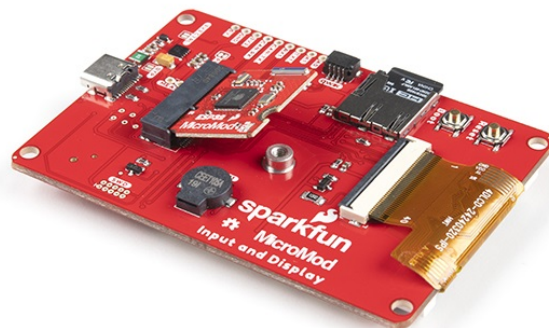
The board measures 22mm x 22mm, with 15mm to the top notch and 12mm to the E key. For more information regarding the processor board physical standards, head on over to the Getting Started with MicroMod tutorial and check out the Hardware Overview section.



Hardware Hookup

To get started with the ESP32 Processor Board, you'll need a carrier board. Here we are using the MicroMod Input and Display Carrier Board. Align the top key of the MicroMod ESP32 Processor Board to the screw terminal of the Input and Display Carrier Board and angle the board into the socket. Insert the board at an angle into the M.2 connector.

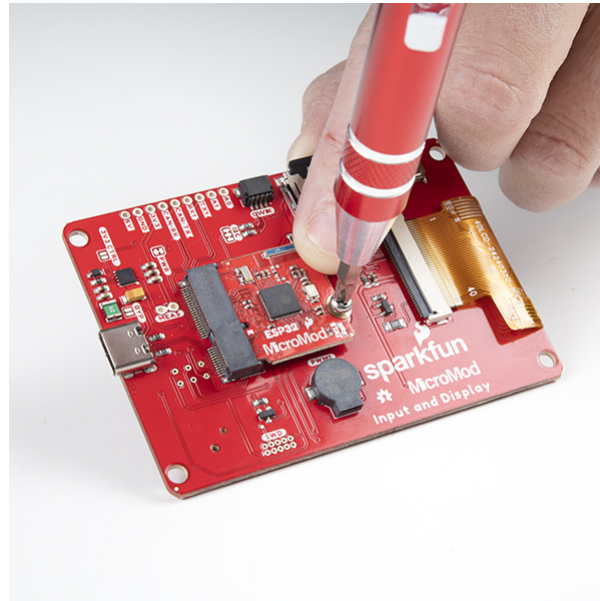
Note: There is no way to insert the processor backward since the key prevents it from mating with the M.2 connector and as an extra safeguard to prevent inserting a processor that matches the key, the mounting screw is offset so you will not be able to secure an improperly connected processor board.



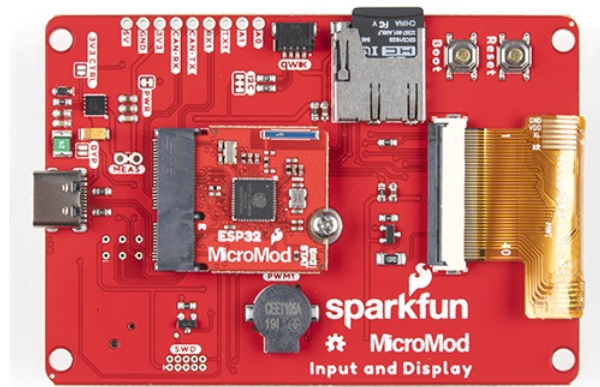
The Processor Board will stick up at an angle, as seen here:



Once the board is in the socket, gently hold the MicroMod Processor Board down and tighten the screw with a Phillip's head.



Once the board is secure, your assembled MicroMod system should look similar to the image below!



Connecting Everything Up

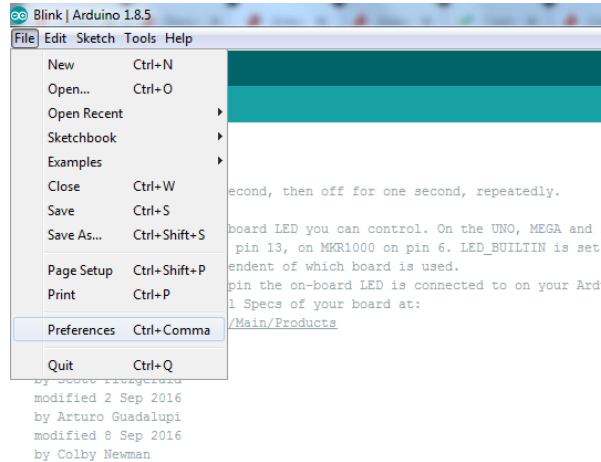
With your processor inserted and secured it's time to connect your carrier board to your computer using the USB-C connector on the Carrier. Depending on which carrier you choose and which drivers you already have installed, you may need to install drivers.

Note: If you've never connected a CP2104 device to your computer before, you may need to install drivers for the USB-to-serial converter. Check out our section on [How to Install CP2104 Drivers](#) for help with the installation.

Software Setup and Programming

Note: All of the examples in this tutorial assume you are using the latest version of the Arduino IDE on your desktop. If this is your first time using Arduino, please review our tutorial on installing the Arduino IDE. If you have not previously installed an Arduino library, please check out our installation guide.

To get started with the MicroMod ESP32 Processor Board, you'll need to install the ESP32 Board Definition. Open the Arduino IDE (must be v1.8.13 or later) and navigate to **File->Preferences**, like so:

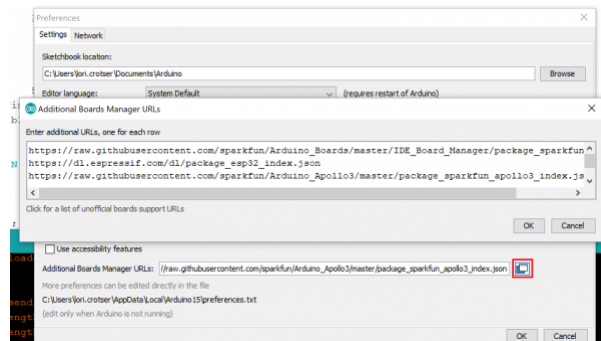


Having a hard time seeing? Click the image for a closer look.

In the "Additional Board Manager URL" box, make sure you have the following json file included. If you do not have it, add it to your preferences.

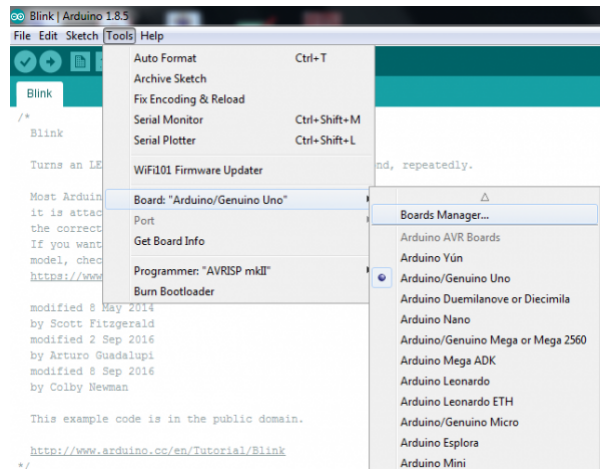
```
language:c
https://raw.githubusercontent.com/sparkfun/Arduino_Boards/master/IDE_Board_Manager/package_s
parkfun_index.json
```

If you have more than one json file, you can click on the button outlined in red and add the json link at the end. It'll look something like the following:



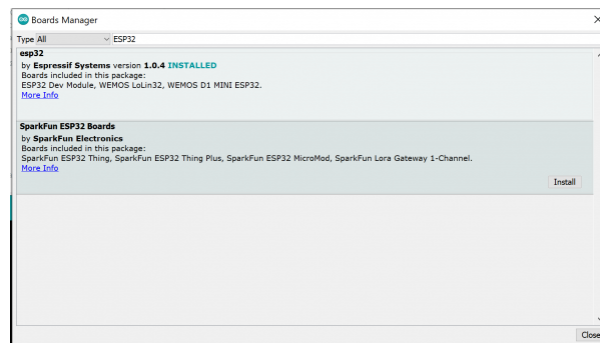
Having a hard time seeing? Click the image for a closer look.

Once you've got your preferences updated, go to **Tools -> Board** and select the **Boards Manager** like so:



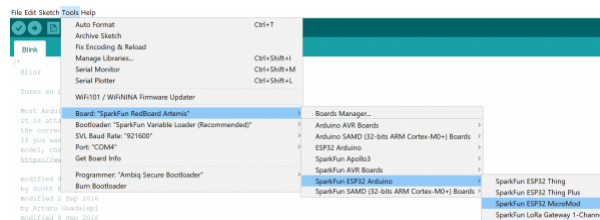
Having a hard time seeing? Click the image for a closer look.

Search for "ESP32", and you should find the **SparkFun ESP32 Boards** board package. Make sure the latest version is selected and click **Install**.



Having a hard time seeing? Click the image for a closer look.

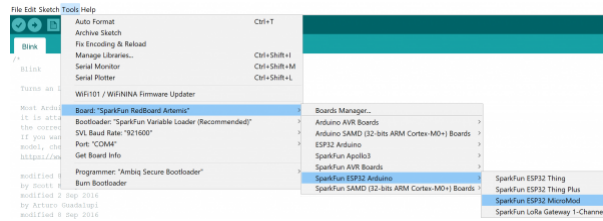
Once the board definitions have been installed, you should see the **SparkFun ESP32 MicroMod Board** under your **Tools -> Board -> SparkFun ESP32 Arduino** menu.



Voila! You're ready to rock your ESP32 MicroMod Processor Board!

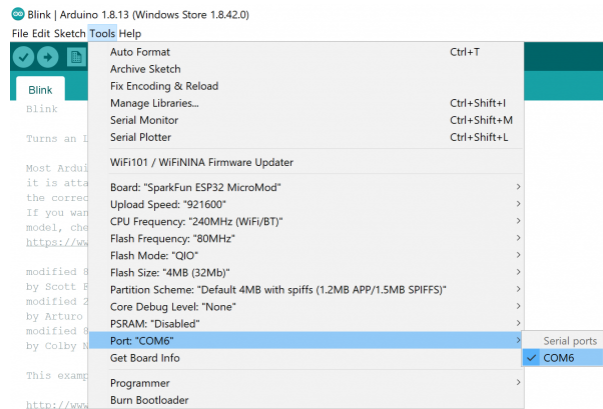
Arduino Example: Blink

With the SparkFun ESP32 Arduino core installed, you're ready to begin programming. Make sure you have the ESP32 MicroMod board definition selected under your **Tools > Board** menu.



Having a hard time seeing? Click the image for a closer look.

Then select your serial port under the **Tools > Port** menu.



Having a hard time seeing? Click the image for a closer look.

You can also select the **Upload Speed: "921600"** baud -- the fastest selectable rate -- will get the code loaded onto your ESP32 the fastest, but may fail to upload once-in-a-while. (It's still way worth it for the speed increase!)

Loading Blink

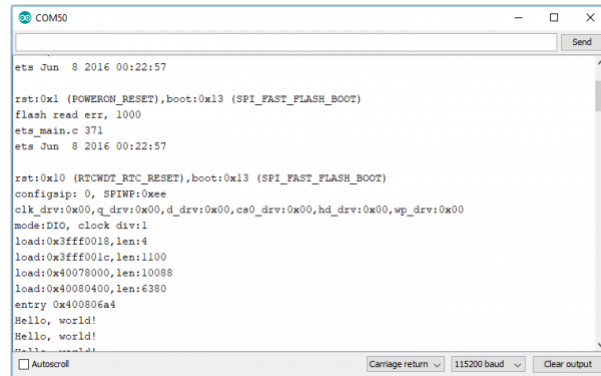
To make sure your toolchain and board are properly set up, we'll upload the simplest of sketches -- Blink! The STAT LED on the ESP32 Processor Board is perfect for this test. This is also a good time to test out serial communication. Copy and paste the example sketch below into a fresh Arduino sketch:

```
int ledPin = 2;

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**. You should see Hello, world! 's begin to fly by. You may also notice that 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**.



```
COM50
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 (RCWDT_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:10098
load:0x40080400,len:6380
entry 0x400806a4
Hello, world!
Hello, world!
```

Having a hard time seeing? Click the image for a closer look.

You should also see some blinking happening on the ESP32 Processor Board! Blink Blink Blink!



Having a hard time seeing? Click the image for a closer look.

If the blue LED remains off, it's probably still sitting in the bootloader. After uploading a sketch, you may need to **tap the reset button** to get your ESP32 MicroMod to run the sketch.

Arduino Example: 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. You 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.

⚠ Please note: Make sure you are connecting to the 2.4GHz band on 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 = LED_BUILTIN;

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 the Boot button 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 you fill in the `networkName` and `networkPswd` variables with the name (or SSID) and password of your WiFi network! Once you've done that and uploaded the code, open your **serial monitor**.

```

COM6
18:07:37.859 -> -----
18:07:37.859 -> Connecting to domain: example.com
18:07:37.894 -> Connected!
18:07:37.894 -> -----
18:07:37.894 ->
18:07:37.929 -> HTTP/1.1 200 OK
18:07:37.929 -> Accept-Ranges: bytes
18:07:37.929 -> Age: 552087
18:07:37.929 -> Cache-Control: max-age=604800
18:07:37.929 -> Content-Type: text/html; charset=UTF-8
18:07:37.929 -> Date: Tue, 20 Oct 2020 00:07:37 GMT
18:07:37.929 -> Etag: "3147526947"
18:07:37.929 -> Expires: Tue, 27 Oct 2020 00:07:37 GMT
18:07:37.929 -> Last-Modified: Thu, 17 Oct 2019 07:18:26 GMT
18:07:37.929 -> Server: ECS (dha/63AA)
18:07:37.929 -> Vary: Accept-Encoding
18:07:37.929 -> X-Cache: HIT
18:07:37.929 -> Content-Length: 1256
18:07:37.929 -> Connection: close
Autoscroll Show timestamp
Newline 115200 baud Clear output

```

Having a hard time seeing? Click the image for a closer look.

After your ESP32 connects to the WiFi network, it will wait for you to press the "Boot" button on your carrier board. 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.

Arduino Example: ESP32 BLE

Both the Board Manager install and the ESP32 arduino core install come with Bluetooth examples that range from serial to serial to acting as a simple BLE device to functioning as either a Bluetooth server or client. Here we will briefly go over the `BLE_write` example that can be found in **Files > Examples > ESP32 BLE Arduino**. This example allows you to write messages on your phone that can then be read in a serial monitor on your computer.

This example works with a **BLE scanner** on your phone. A good, basic app is the BLE Scanner for iPhone or Android. Make sure to install the app to follow along with this example.

Compile and upload the following code, or if you wish, open the **BLE_write** example from the **Files > Examples > ESP32 BLE Arduino** menu. Make sure you have **ESP32 MicroMod** as your board and the correct port has been selected.

```

/*
   Based on Neil Kolban example for IDF: https://github.com/nkolban/esp32-snippets/blob/master/cpp\_utils/tests/BLE%20Tests/SampleWrite.cpp
   Ported to Arduino ESP32 by Evandro Copercini
*/

#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);

    Serial.println("1- Download and install an BLE scanner app in your phone");
    Serial.println("2- Scan for BLE devices in the app");
    Serial.println("3- Connect to MyESP32");
    Serial.println("4- Go to CUSTOM CHARACTERISTIC in CUSTOM SERVICE and write something");
    Serial.println("5- See the magic =)");

    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());

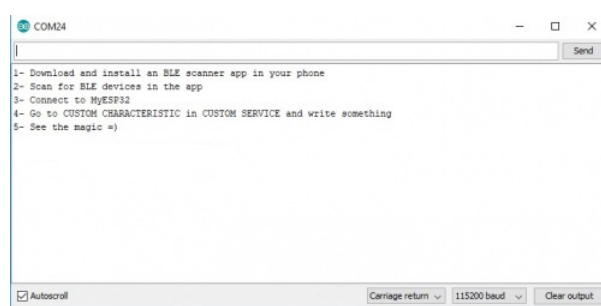
pCharacteristic->setValue("Hello World");
pService->start();

BLEAdvertising *pAdvertising = pServer->getAdvertising();
pAdvertising->start();
}

void loop() {
  // put your main code here, to run repeatedly:
  delay(2000);
}

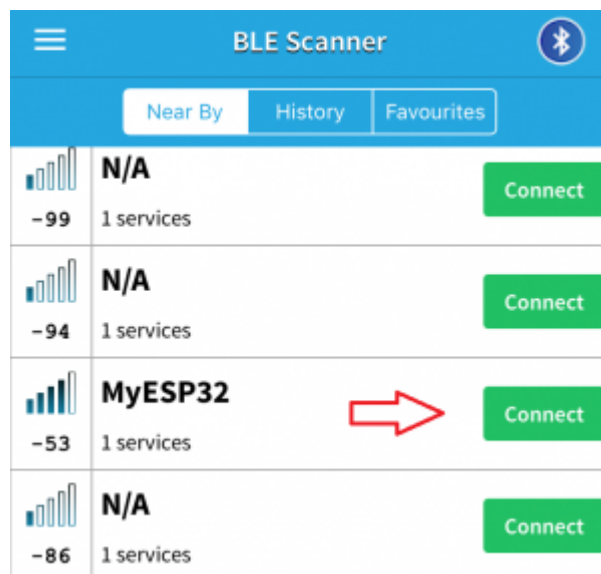
```

Once you have uploaded your code, open the **serial monitor** set at **115200 baud** so you can see the message that we will write.

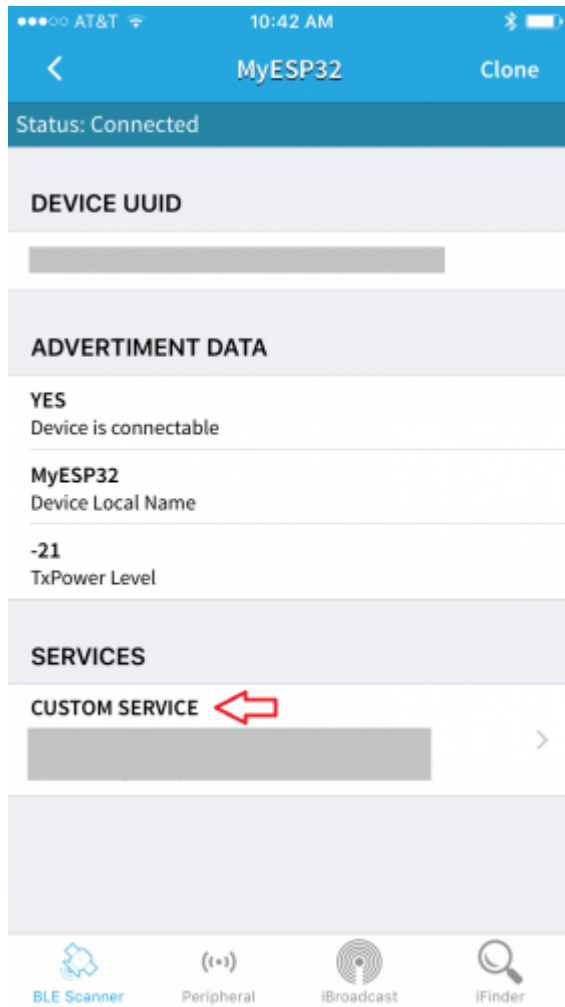


Having a hard time seeing? Click the image for a closer look.

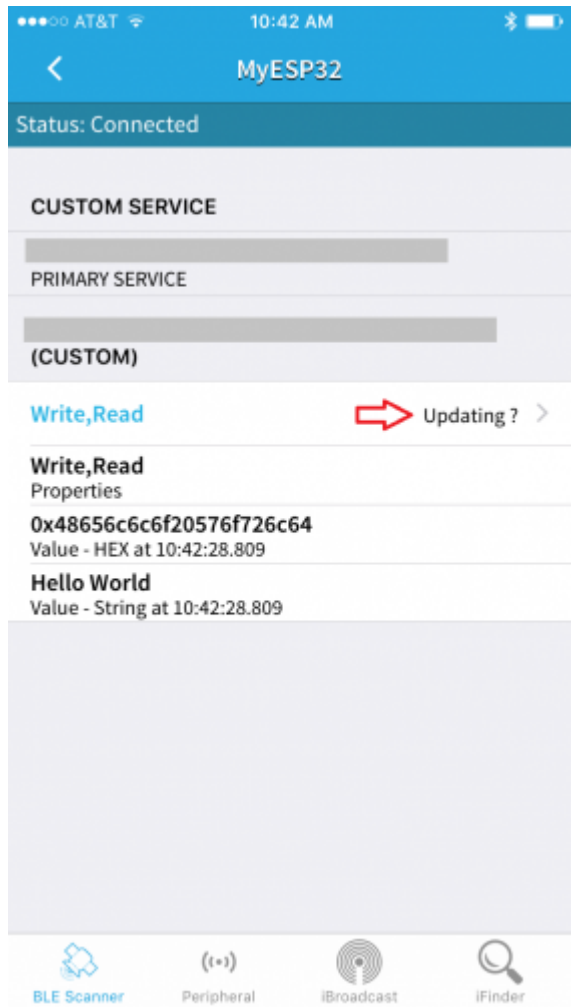
Then open your **BLE Scanner** app on your phone. You will see multiple options; scroll through these and connect to **MyESP32**.



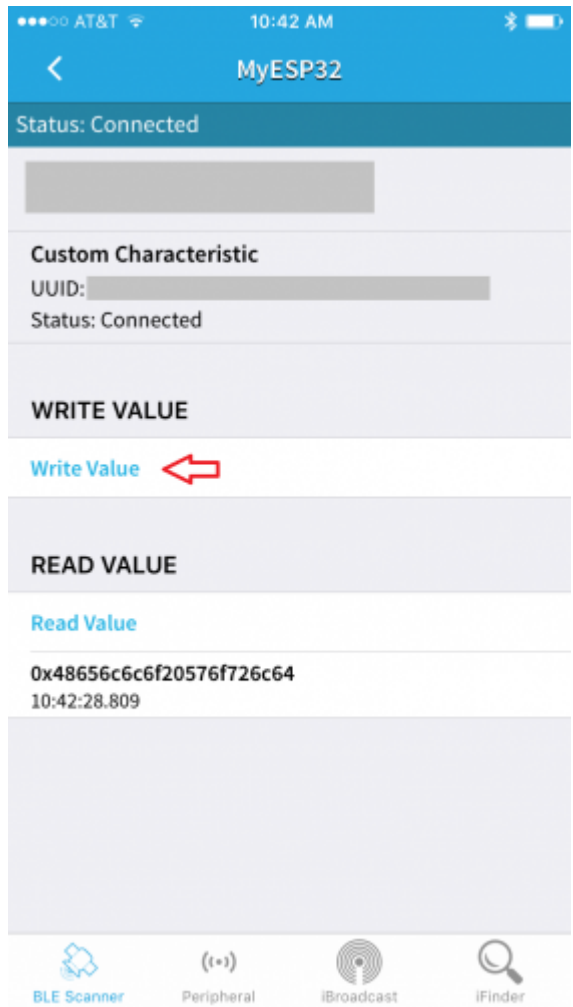
Now we need to drill down to the communication capability we want. Once you are connected to **MyESP32**, you will be taken to the following page. Select **CUSTOM SERVICE**.



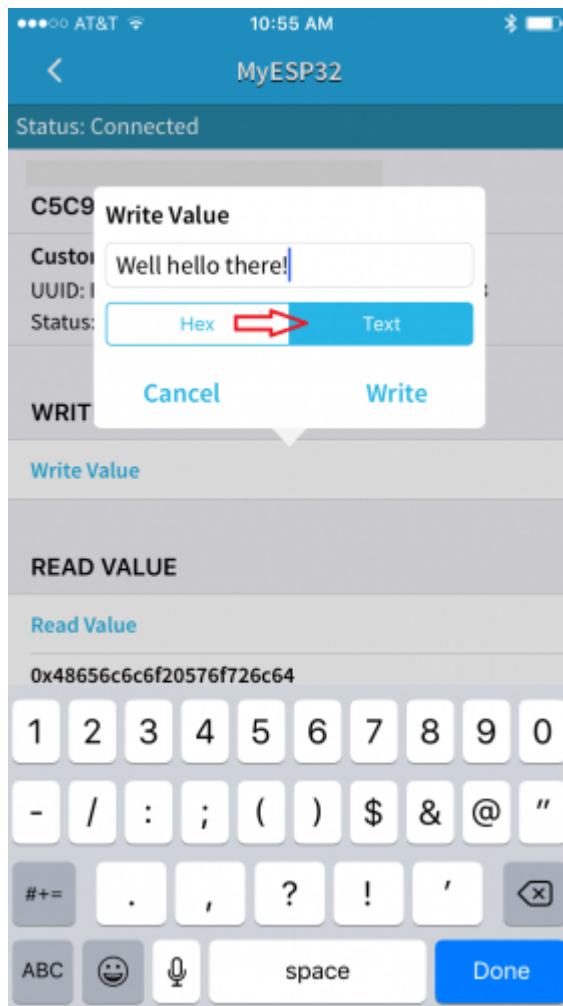
The next page will show you communications and options for doing so. Select **Write,Read**.



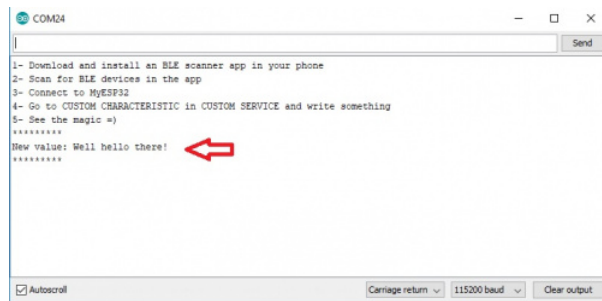
Finally, we can choose the option that allows us to write a message. Select **Write Value**.



Now we can write our message. Make sure you choose **Text**, write yourself a message, and click the **Write** button.



So now what? Go have a look at your serial monitor. You should see **"New value:"** with your message.



Having a hard time seeing? Click the image for a closer look.

This is just a quick walk through of one of the provided examples. We recommend looking through the rest of the provided samples and playing with the code to see what may work for your application. For more information on Bluetooth technology and how it works, check out our Bluetooth Basics Tutorial.

Further Examples

With the MicroMod system, the possibilities for examples with all the processor/carrier board are endless, and we just can't cover them all. You'll notice that in this tutorial, we've selected the Input and Display Carrier Board, but have focused our examples on the Esp32 Processor Board. If you're interested in examples specifically for our carrier board, head on over to our SparkFun MicroMod Input and Display Carrier Board Hookup Guide.

Troubleshooting

With the MicroMod Processors, you can change out the processors with little to no changes in the code. But because each processor board's architecture is different, the way communication protocols are initialized might be a little bit different. For the ESP32 Processor, the two main protocols are the Universal Asynchronous Receiver Transmitter (UART), aka Serial, and I²C, aka Wire.

Secondary Serial/UART Initialization Tips

The UART is initialized with the begin function as:

```
void begin(unsigned long baud, uint32_t config, int8_t rxPin, int8_t txPin, bool invert, unsigned long timeout_ms)
```

The primary UART works like any other Arduino board being able to initialize and send messages back to your computer over the USB cable using `Serial.begin(115200)` for a baud rate of 115200 as an example. If you wanted to communicate at the same 115200 baud rate on the secondary UART you would initialize `Serial1` as:

```
Serial1.begin(115200, SERIAL_8N1, RX1, TX1);
```

Or if you want to use the GPIO pin numbers instead, it would be:

```
Serial1.begin(115200, SERIAL_8N1, 16, 17);
```

Secondary I²C Initialization Tips

The Wire bus is initialized with the begin function as:

```
bool begin(int sda, int scl, uint32_t frequency); // returns true, if successful init of i2c bus
```

With the primary Wire bus, these pins use the default SCL and SDA pins connected to GPIO pins 22 and 21 and can be initialized by simply calling `wire.begin()`. If you plan on using the secondary Wire bus, you need to provide the pins, and possibly the desired frequency if the default 400kHz is too fast. For most applications though you can use:

```
Wire1.begin(SDA1, SCL1);
```

Or if you wanted to use the GPIO pin numbers, it would be:

```
Wire1.begin(26, 25);
```

Need help?

If your product is not working as you expected or you need technical assistance or information, head on over to the [SparkFun Technical Assistance page](#) for some initial troubleshooting.

If you don't find what you need there, the [SparkFun Forums](#) are a great place to find and ask for help. If this is your first visit, you'll need to create a [Forum Account](#) to search product forums and post questions.

Resources and Going Further

For more information about the MicroMod ESP32 Processor Board, check out the following links:

- Schematic (PDF)
- Eagle Files (ZIP)
- Board Dimensions (PNG)
- ESP32 Datasheet (PDF)
- GitHub Hardware Repo

For more information about the SparkFun MicroMod Ecosystem, take a look at the links below:

- Getting Started with MicroMod
- Designing with MicroMod
- MicroMod Info Page
- MicroMod Forums

Looking for some project inspiration using your ESP32 Processor Board? The tutorials below can help you get started!



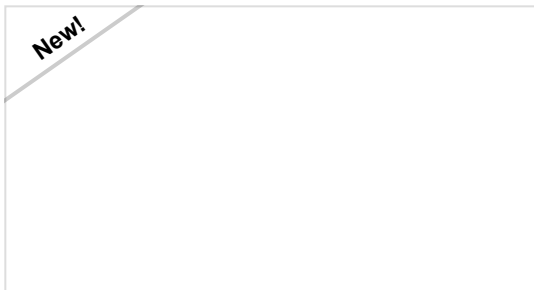
Getting Started with MicroMod

Dive into the world of MicroMod - a compact interface to connect a microcontroller to various peripherals via the M.2 Connector!



MicroMod Artemis Processor Board Hookup Guide

Get started with the Artemis MicroMod Processor Board in this tutorial!



MicroMod All The Pins (ATP) Carrier Board

Access All The Pins (ATP) of the MicroMod Processor Board with the Carrier Board!



SparkFun MicroMod Input and Display Carrier Board Hookup Guide

A short Hookup Guide to get started with the SparkFun MicroMod Input and Display Carrier Board