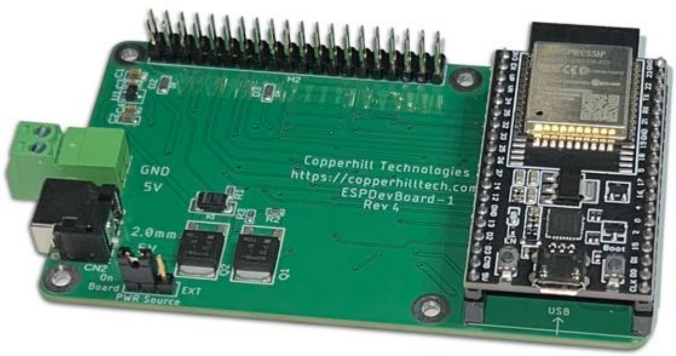
The espBerry Project
Extending ESP32 Functionality



espBerry - ESP32 Development Board with Raspberry Pi GPIO

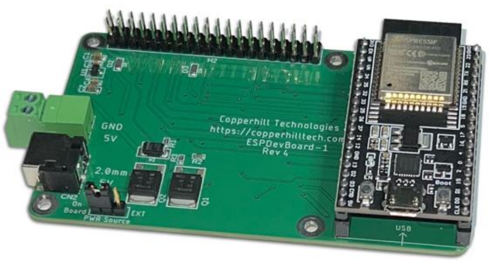
User Manual

Rev. 2023-05-24

https://espBerry.com – https://copperhilltech.com

Contents

Overview



The *espBerry* DevBoard combines the [ESP32-DevKitC development board] with any Raspberry Pi HAT by connecting to the onboard RPi-compatible 40-pin GPIO header.

The purpose of the *espBerry* should not be perceived as a Raspberry Pi alternative but as extending the ESP32's functionality by tapping into the vast offerings of RPi HATs in the market and taking advantage of the multiple and flexible hardware options.

The *espBerry* is the perfect solution for prototyping and Internet of Things (IoT) applications, especially those requiring wireless capabilities. All open-source code samples take advantage of the popular Arduino IDE with its excellent programming capabilities.

In the following, we will explain the hardware and software features, including all details you need to know to add the Raspberry HAT of your choice. In addition, we will provide a collection of hardware and software samples to demonstrates the espBerry's capabilities.

However, we will refrain from repeating information that is already available through other resources, i.e., online posts and articles. Wherever we deem that additional information is necessary, we will add references for you to study.

Note: *We are trying very hard to document every detail that may be important for our customers to know. However, documentation takes times, and we are not always perfect. If you need further information or have suggestions, please feel free to contact us.*

espBerry Features

- Processor: ESP32 DevKitC
    - 32-Bit Xtensa dual-core @240 MHz
    - WiFi IEEE 802.11 b/g/n 2.4 GHz
    - Bluetooth 4.2 BR/EDR and BLE
    - 520 kB SRAM (16 kB for cache)
    - 448 kB ROM
    - Programmable per USB A/micro–USB B cable

- Raspberry Pi Compatible 40-pin GPIO header
    - 20 GPIO
    - 2 x SPI
    - 1 x UART

- Input Power: 5 VDC
    - Reverse polarity protection
    - Overvoltage Protection
    - Power Barrel Connector Jack 2.00mm ID (0.079”), 5.50mm OD (0.217”)
    - 12/24 VDC options available

- Operating Range: -40°C ~ 85°C
  Note: Most RPi HATs operate at 0°C ~ 50°C

- Dimensions: 95 mm x 56 mm – 3.75” x 2.2”
  Complies to Standard Raspberry Pi HAT Mechanical Specifications…

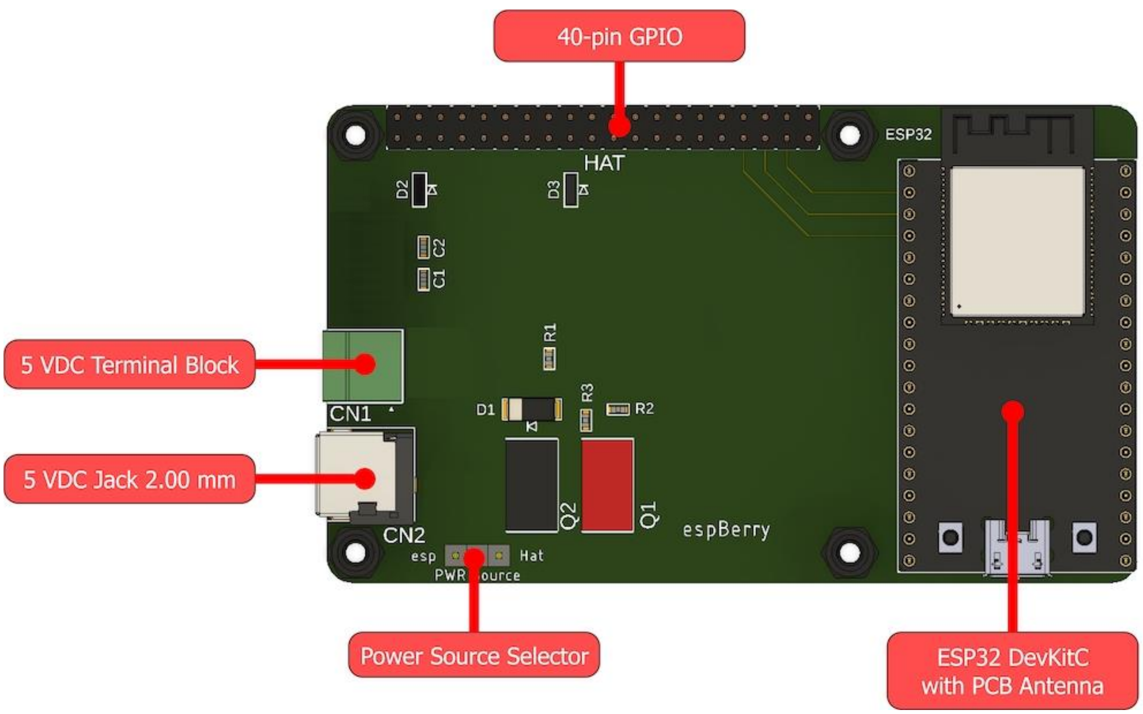
Hardware

In general, the espBerry development board combines the ESP32-DevKitC module with any Raspberry Pi HAT by connecting to the onboard RPi-compatible 40-pin GPIO header.

The most-used connections between the ESP32 and the RPi HAT are the SPI and the UART port as explained in the following chapters. We have also mapped several GPIO (General Purpose Input Output) signals. For more detailed information on the mapping, please refer to the schematic.

We are trying very hard to provide good documentation. However, please understand that we cannot explain all ESP32 details in this user manual. For more detailed information, please refer to the ESP32-DevKitC V4 Getting Started Guide.

espBerry Board Components



Power Source Connector

The espBerry can be powered through several sources:

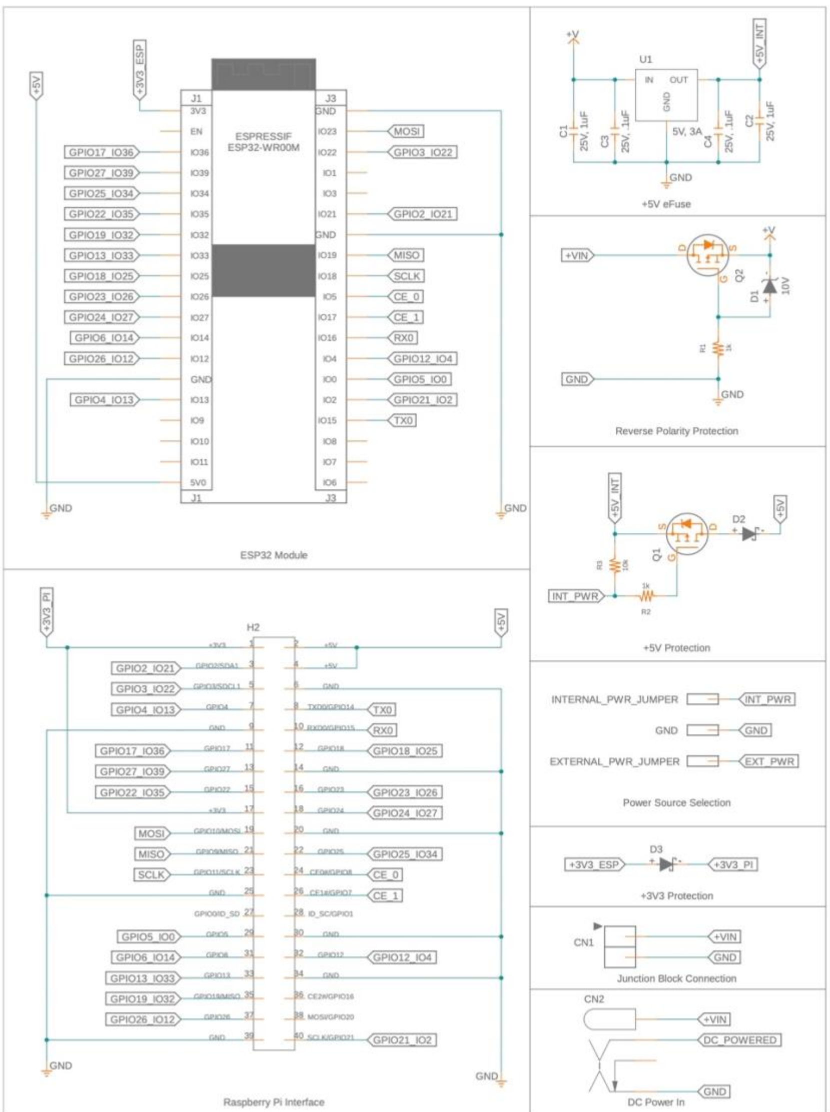
- The Micro-USB connector on the ESP32 DevKitC module
- The 5 VDC Jack 2.0 mm
- The 5 VDC Terminal Block
- External power supply connected to the RPi HAT

There are Raspberry Pi HATs that allow to supply external power (e.g., 12 VDC) directly to the HAT. When powering the espBerry through this external power supply, you need to set the jumper at the *Power Source Selector* to “EXT.” Otherwise, it must set to “On Board.”

It is possible to power the espBerry internally (“On Board”) while still having power applied to the HAT.

espBerry Schematics

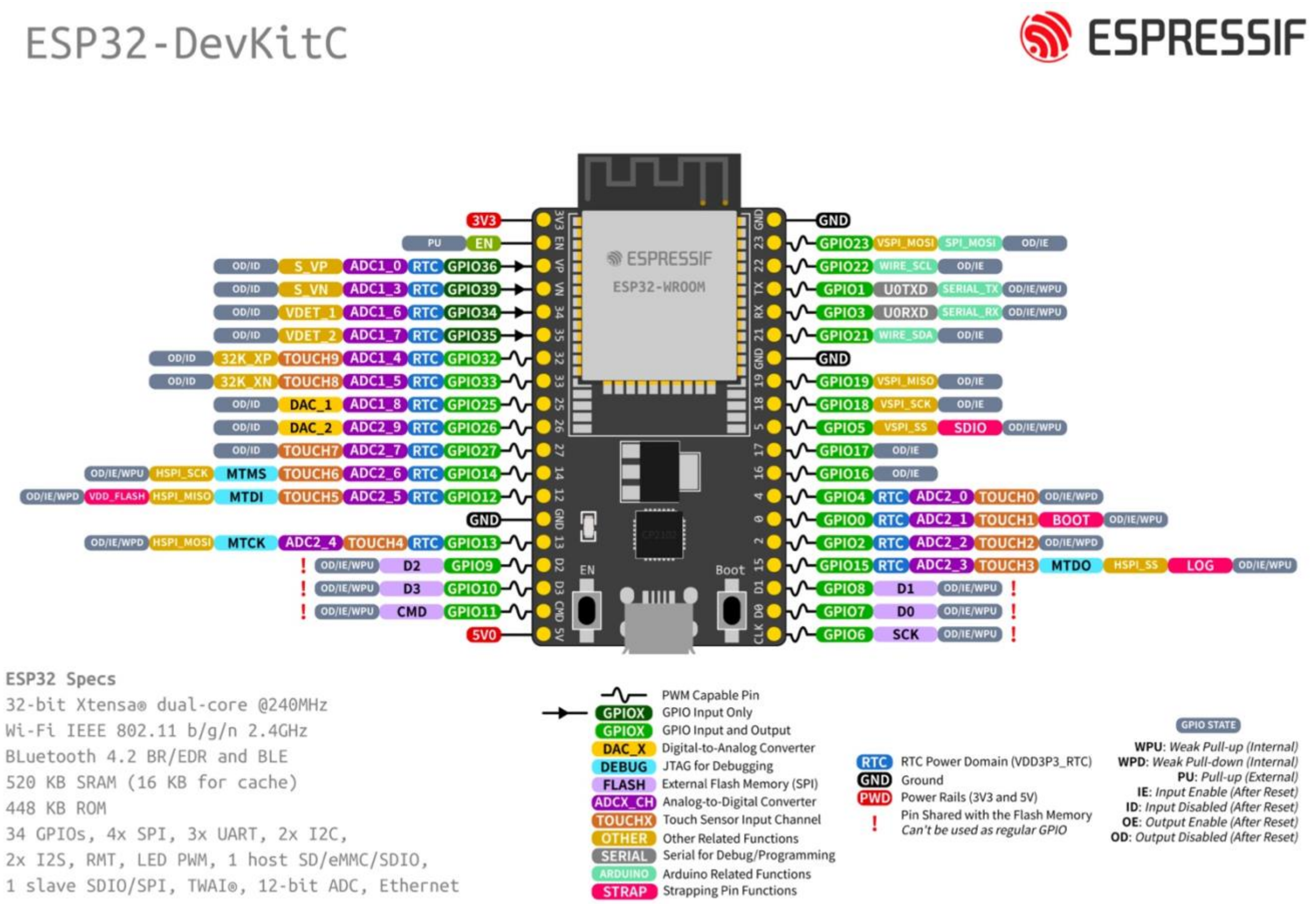
The espBerry was designed to map as many signals (GPIO, SPI, UART, etc.) as possible. However, that does not necessarily mean that the espBerry covers all HATs available in the market. Your ultimate source for adaptations and developing your own HAT must be the espBerry's schematic.



Click here to download the full espBerry schematics (PDF).

In addition, we have added the ESP32 DevKitC and the Raspberry Pi 40-pin GPIO header pinout in the following chapters.

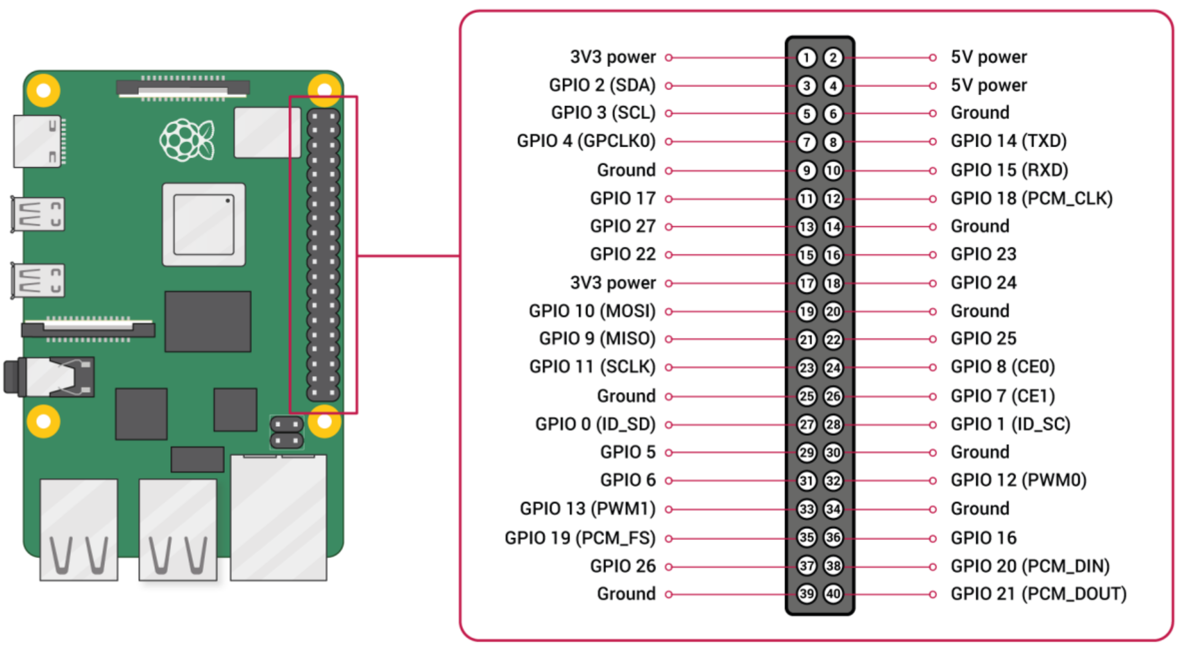
The ESP32 DevKit pinout



For a full view of the above image, click here.

The Raspberry Pi 40-pin GPIO Header

A powerful feature of the Raspberry Pi is the row of GPIO (general-purpose input/output) pins along the top edge of the board. A 40-pin GPIO header is found on all current Raspberry Pi boards (unpopulated on Raspberry Pi Zero, Raspberry Pi Zero W and Raspberry Pi Zero 2 W). Prior to the Raspberry Pi 1 Model B+ (2014), boards comprised a shorter 26-pin header. The GPIO header on all boards (including the Raspberry Pi 400) have a 0.1" (2.54mm) pin pitch.



For more information, refer to [Raspberry Pi Hardware – GPIO and the 40-pin Header].

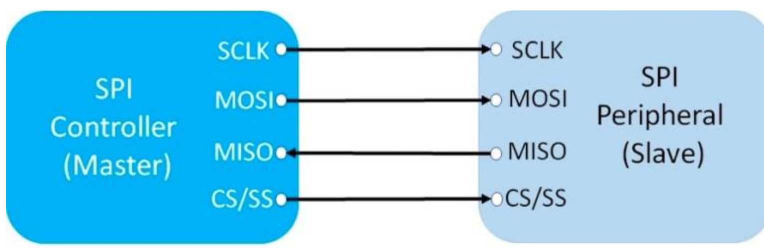
For more information on Raspberry Pi HATs, please refer to [Add-On Boards and HATs].

SPI Port Connection

SPI stands for Serial Peripheral Interface, a serial full-duplex and synchronous interface. The synchronous interface requires a clock signal to transfer and receive data. The clock signal is synchronized between one central control ("master") and multiple peripheral devices ("slaves"). Unlike UART communication, which is asynchronous, the clock signal controls when data is to be sent and when it should be ready to read.

Only a master device can control the clock and provide a clock signal to all slave devices. Data cannot be transferred without a clock signal. Both master and slave can exchange data with each other. No address decoding is required.

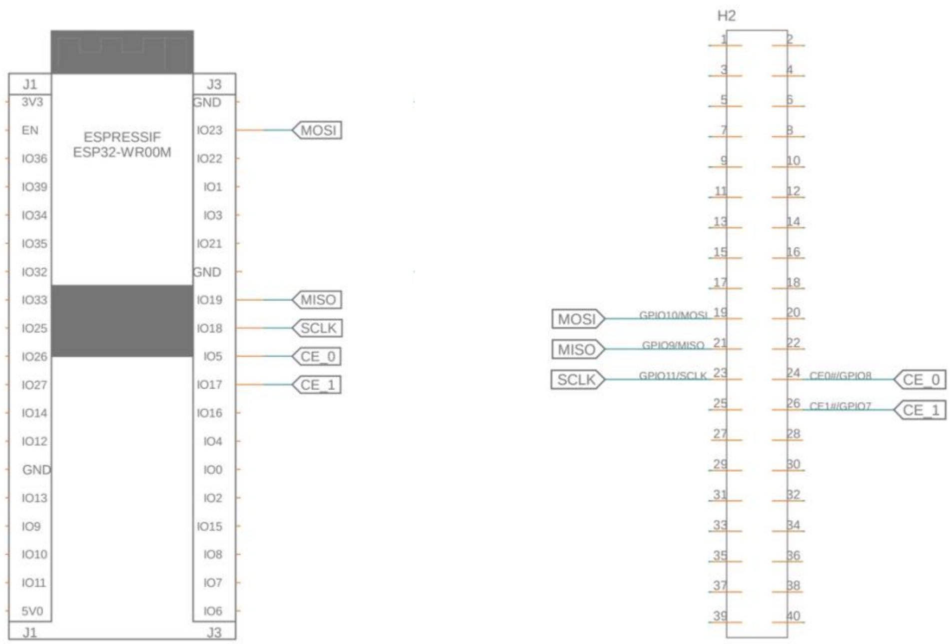
The ESP32 has four SPI buses, but only two are available for usage, and they are known as HSPI and VSPI. As mentioned earlier, in SPI communication, there is always one controller (also known as a master) that controls other peripheral devices (also known as slaves). You can configure the ESP32 either as a master or slave.



On the espBerry, the signals assigned to the default IOs:

SCK	->	IO18	a.k.a. SLCK
MISO	->	IO19	
MOSI	->	IO23	
CS_0	->	IO5	a.k.a. SS
CS_1	->	IO17	

Below image shows the SPI signals from the ESP32 module to the RPi GPIO header as an excerpt from the schematic.



There are many types of ESP32 boards available. Boards other than the espBerry may have different default SPI pins, but you can find information about default pins from their datasheet. But if default pins are not mentioned, you can find them by using an Arduino sketch (use first link below).

For more information, see:

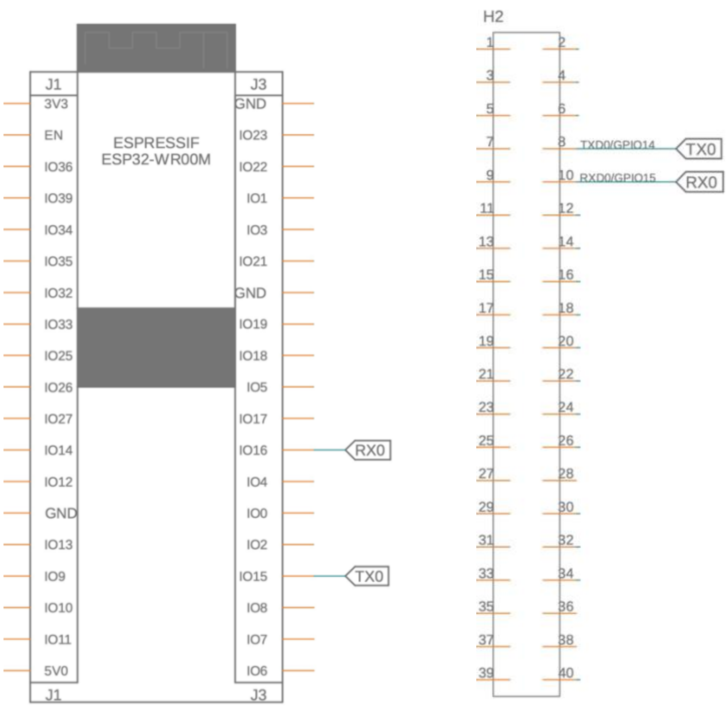
- ESP32 SPI Tutorial Master Slave Communication Example...
- Espressif GPIO Matrix and IO_MUX...

The espBerry uses the VSPI connection as a default, meaning if you go with the default signals, you should not run into problems. There are ways to change the pin assignment and switch to HSPI (as explained in the above references), but we haven't explored these scenarios for the espBerry.

See also our section on SPI Port Programming.

Serial (UART) Port Connection

Besides the onboard USB port, the ESP32 development module has three UART interfaces, i.e., UART0, UART1, and UART2, which provide asynchronous communication at a speed of up to 5 Mbps. These serial ports can be mapped to almost any pin. On the espBerry, we assigned IO15 as Rx and IO16 as Tx, which are connected to GPIO16 and GPIO20 on the 40-pin header as shown here:



We have chosen not to use the standard RX/TX (GPIO3/GPIO1) signals on the ESP32 DevKit, since they are often used for test prints through the Serial Monitor of the Arduino IDE. This may interfere with the communication between the ESP32 and the RPi HAT. Instead, you must map IO16 as Rx and IO15 as Tx per software as explained in the *Software* section of this manual.

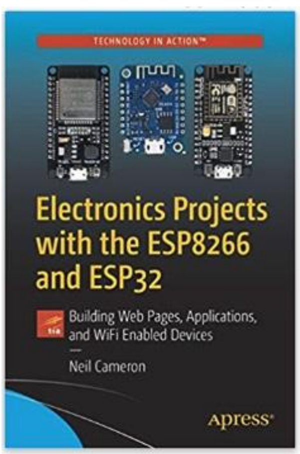
See also our section on [Serial (UART) Programming](#).

Software

In the following, we will briefly explain the most important programming aspects for the espBerry. As mentioned previously in this user manual, we will add online references where we deem that additional information is necessary.

For more, hands-on project samples, see also our ESP32 Programming Tips.

In addition, there are many examples of ESP32 programming literature, which are worth the investment.



However, we highly recommend using Electronic Projects with the ESP8266 and ESP32, especially for your wireless application projects. Yes, many good books and free online resources are available these days, but this is the book we are using. It made our approach to Bluetooth, BLE, and WIFI a breeze. Programming wireless applications without hassles was fun, and we share them on our web site.

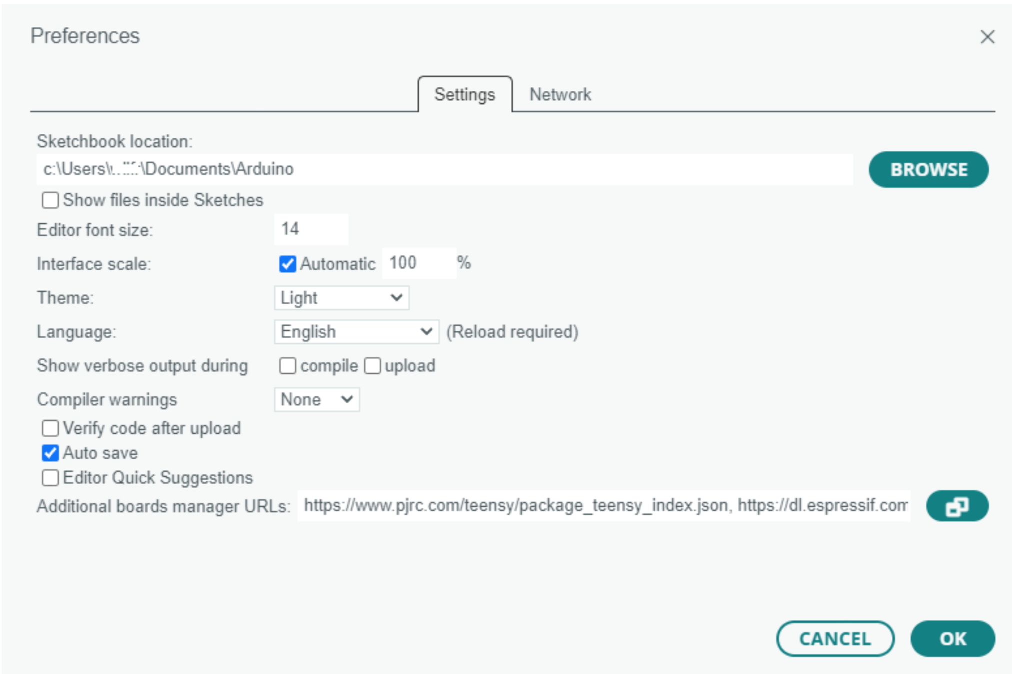
Installing and Preparing the Arduino IDE

All our programming samples have been developed using the Arduino IDE (Integrated Development Environment) due to its ease of installation and usage. Furthermore, there are a myriad of Arduino sketches available online for the ESP32.

For the installation, follow these steps:

Step 1: The first step would be to download and install the Arduino IDE. This can be done easily by following the link https://www.arduino.cc/en/Main/Software and downloading the IDE for free. If you already have one, make sure you have the latest version.

Step 2: Once installed, open the Arduino IDE, and go to *Files -> Preferences* to open the preferences window and locate the "Additional Boards Manager URLs:" as shown below:



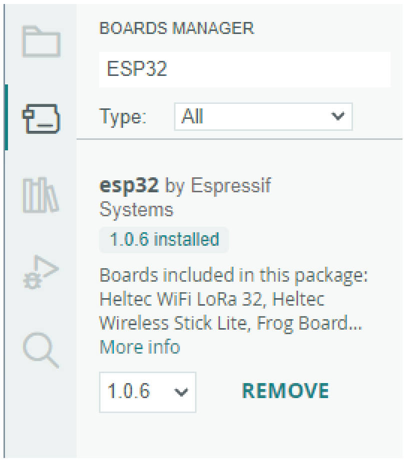
The text box may be empty or already contain some other URL if you have used it previously for another board. If it is empty, simply paste the below URL into the text box.

https://dl.espressif.com/dl/package_esp32_index.json

If the text box already contains some other URL just add this URL to it, separate both with a comma (,). Ours already had the Teensy URL. We just entered the URL and added the comma.

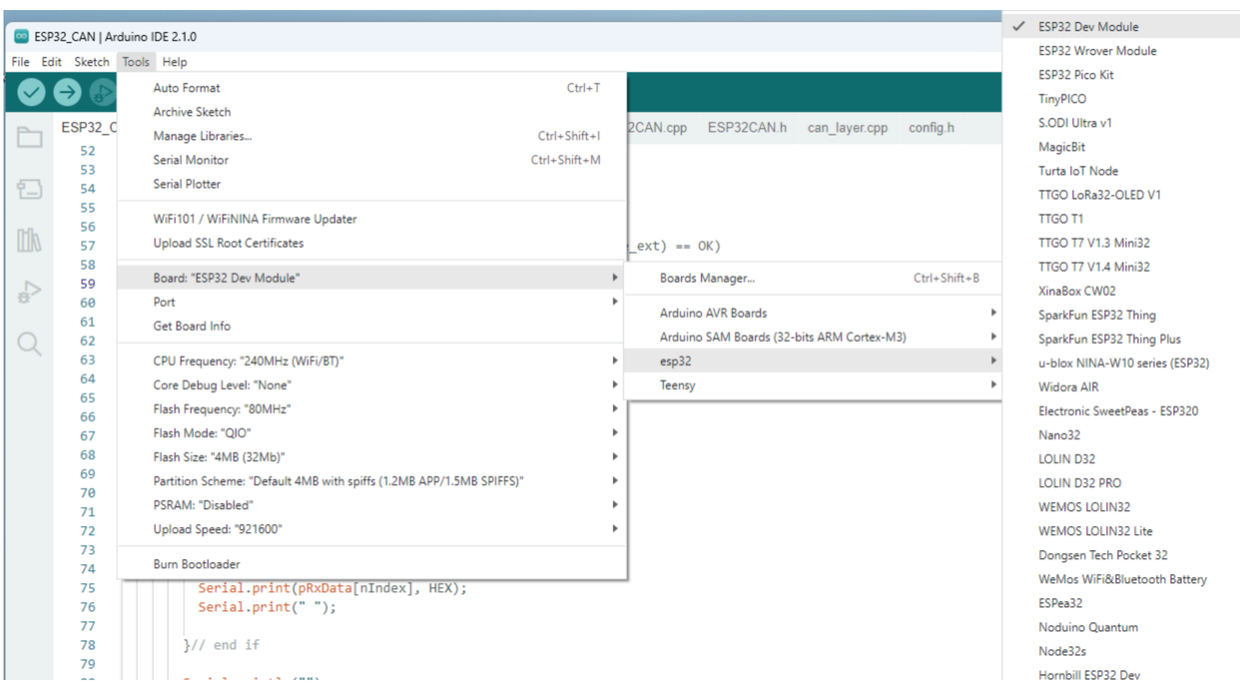
Once done, click on OK and the window will disappear.

Step 3: Go to *Tools -> Boards -> Board Managers* to open the Board manager window and search for ESP32. If the URL was pasted correctly your window should find the below screen with Install button, just click on the Install button and your board should get installed.



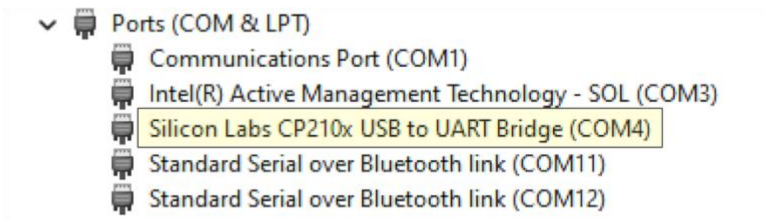
The above screen shot shows the ESP32 after it was installed.

Step 4: Before you start programming, you must set the select the appropriate ESP32 hardware (there are multiple options). Navigate to *Tools -> Boards* and select *ESP32 Dev Module* as shown here:

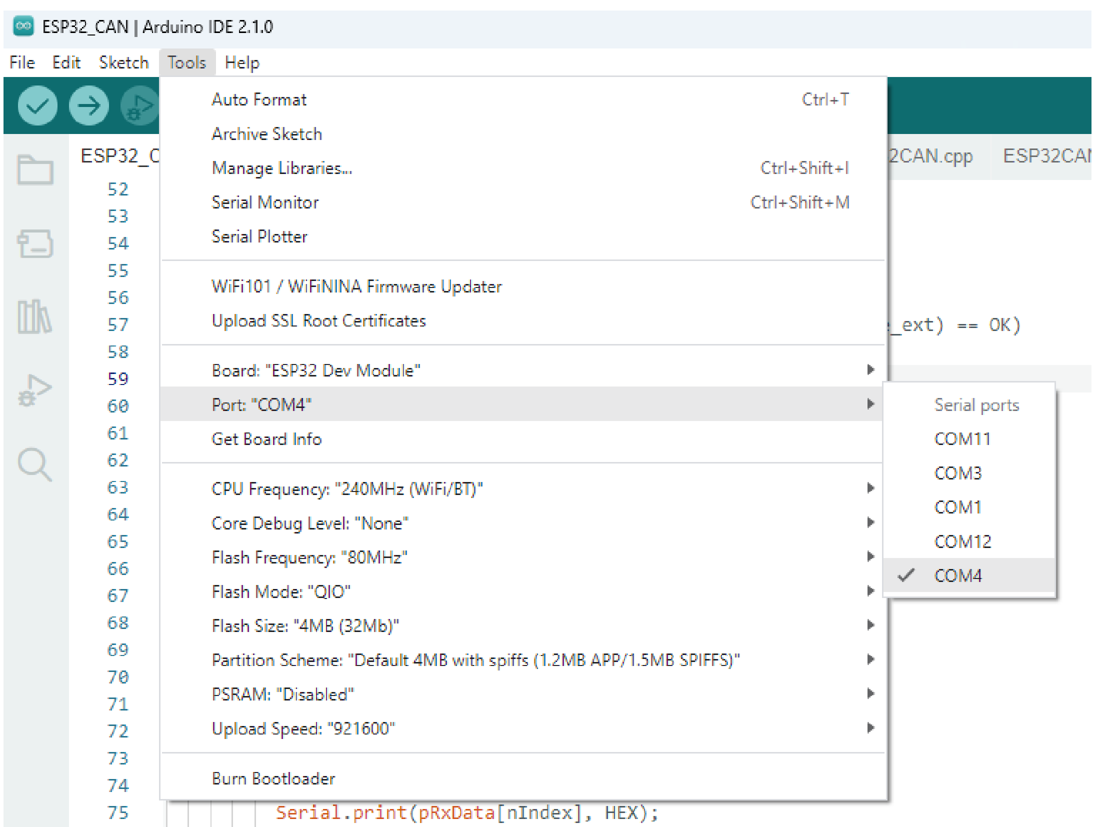


The espBerry Project
Extending ESP32 Functionality

Step 5: Open the device manager and check to which COM port your ESP32 is connected.



When using the espBerry, look for the *Silicon Labs CP210x USB to UART Bridge*. In our setup it shows COM4. Go back to Arduino IDE and under *Tools -> Port*, select the Port to which your ESP is connected.



If you are a beginner with the Arduino IDE, please refer to [Using the Arduino Software (IDE).]

SPI Port Programming

The following represents only a brief overview of SPI programming. SPI programming is not easy, but whenever we start a new project, we look for code online (e.g., github.com).

For instance, to program the MCP2515 CAN controller, we are using a modified version of the MCP_CAN Library for Arduino by Cory Fowler, i.e., we are utilizing his knowledge and effort for our project.

Nevertheless, it is worth spending time to understand SPI programming on a basic level. For instance, the espBerry has the SPI signals mapped as shown here:

SCK	->	IO18	a.k.a. SLCK
MISO	->	IO19	
MOSI	->	IO23	
CS_0	->	IO5	a.k.a. SS
CS_1	->	IO17	

These settings must be applied in the application's code. Please refer to the following resources to learn more about SPI programming with the ESP32:

- ESP32 SPI Communication: Set Pins, Multiple Bus Interfaces, and Peripherals…
- How to use SPI with ESP32 and Arduino…

Serial Port (UART) Programming

On the espBerry, we assigned IO15 as Rx and IO16 as Tx, which are connected to GPIO16 and GPIO20 on the 40-pin header.

We have chosen not to use the standard RX/TX (GPIO3/GPIO1) signals on the ESP32 DevKit, since they are often used for test prints through the Serial Monitor of the Arduino IDE. This may interfere with the communication between the ESP32 and the RPi HAT. Instead, you must map IO16 as Rx and IO15 as Tx per software.

```
#include <Arduino.h>
#include <HardwareSerial.h>

HardwareSerial ESPSerial1(1); // Pass 1 to define Serail1 on the ESP32
// REf.: https://quadmeup.com/arduino-esp32-and-3-hardware-serial-ports/

// Define the UART pins
#define uartRxPin     15
#define uartTxPin     16

// Define more UART parameters
#define COMPORT       ESPSerial1
#define SER_BUF_SIZE  (unsigned int)(1024) // Serial buffer in bytes
                                           // (power 2 required)
#define UART_BR       3000000

void setup()
{
  // Set up ESPSerial1
  COMPORT.setRxBufferSize(SER_BUF_SIZE);  // Standard Arduino has 64 bytes
                                          // ESP32 has 256 bytes
                                          // Call must come before begin();
  // Initialize the serial port
  COMPORT.begin(UART_BR, SERIAL_8N1, uartRxPin, uartTxPin);
```

The above code represents an application example using *Serial1*.

When working with the ESP32 under the Arduino IDE, you will notice that the *Serial* command works just fine but *Serial1* and *Serial2* do not. The ESP32 has three hardware serial ports that can be mapped to almost any pin. To get *Serial1* and Serial2 to work, you need to involve the *HardwareSerial* class. As a reference, see ESP32, Arduino and 3 Hardware Serial Ports.

See also our post espBerry Project: ESP32 with CH9102F USB-UART Chip for Serial Speed up to 3Mbit/s.