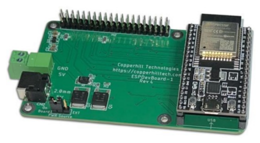
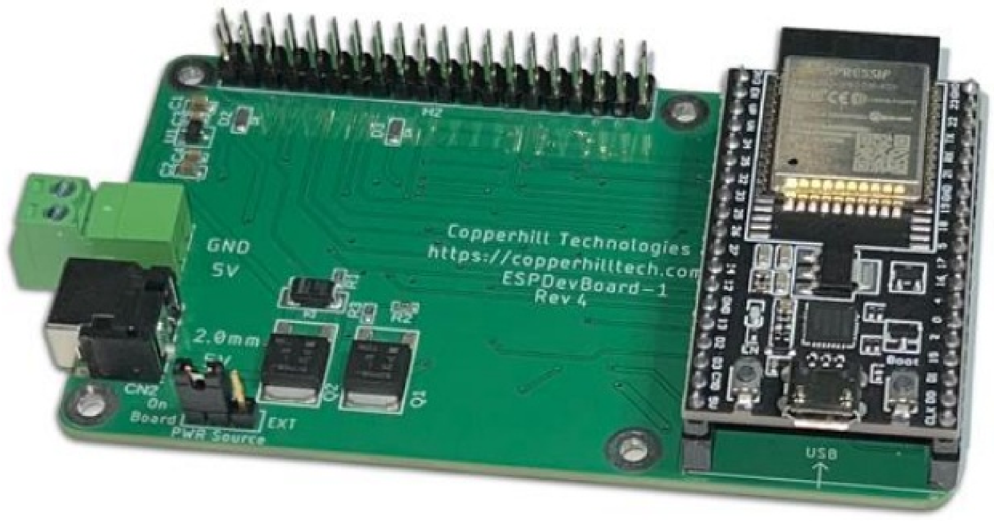


espBerry ESP32 Development Board with Raspberry Pi GPIO User Manual

Contents

The espBerry Project
Extending ESP32 Functionality

espBerry ESP32 Development Board with Raspberry Pi GPIO



PRODUCT INFORMATION

Specifications

- **Power Source:** Multiple sources
- **GPIO:** Compatible with Raspberry Pi 40-pin GPIO header
- **Wireless Capabilities:** Yes
- **Programming:** Arduino IDE

Overview

The espBerry DevBoard combines the ESP32DevKitC development board with any Raspberry Pi HAT by connecting to the onboard RPi compatible 40-pin GPIO header. It is not meant to be a Raspberry Pi alternative, but rather an extension of the ESP32's functionality by utilizing the wide range of RPi HATs available in the market.

Hardware

Power Source Connector
The espBerry can be powered through various sources. Please refer to the user manual for detailed information on the available power sources.

espBerry Schematics
The espBerry was designed to map as many signals (GPIO, SPI, UART, etc.) as possible. However, it may not cover all HATs available in the market. To adapt and develop your own HAT, refer to the espBerry's schematic. You can download the full espBerry schematics (PDF) **here**.

The ESP32 DevKit Pinout
The ESP32 DevKit pinout provides a visual representation of the board's pin configuration. For a full view of the pinout image, click **here**.

The Raspberry Pi 40-pin GPIO Header
The Raspberry Pi features a row of GPIO pins along the top edge of the board. The espBerry is compatible with the 40-pin GPIO header found on all current Raspberry Pi boards. Please note that the GPIO header is unpopulated on Raspberry Pi Zero, Raspberry Pi Zero W, and Raspberry Pi Zero 2 W. Prior to the Raspberry Pi 1 Model B+, boards had a shorter 26-pin header. The GPIO header has a 0.1 (2.54mm) pin pitch.

SPI Port Connection

The SPI port on the espBerry allows for serial full-duplex and synchronous communication. It utilizes a clock signal to transfer and receive data between a central control (master) and multiple peripheral devices (slaves). Unlike UART communication, which is asynchronous, the clock signal synchronizes data transfer.

FAQ

- **Can I use any Raspberry Pi HAT with the espBerry?**
  The espBerry is designed to be compatible with any Raspberry Pi HAT by connecting to the onboard 40-pin GPIO header. However, it may not cover all HATs available in the market. Please refer to the espBerry's schematic for more information.
- **What programming language can I use with the espBerry?**
  The espBerry supports programming using the popular Arduino IDE, which offers excellent programming capabilities.
- **Where can I find additional information and resources?**
  While this user manual provides detailed information, you can also explore online posts and articles for additional resources. If you need further information or have suggestions, feel free to contact us.

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

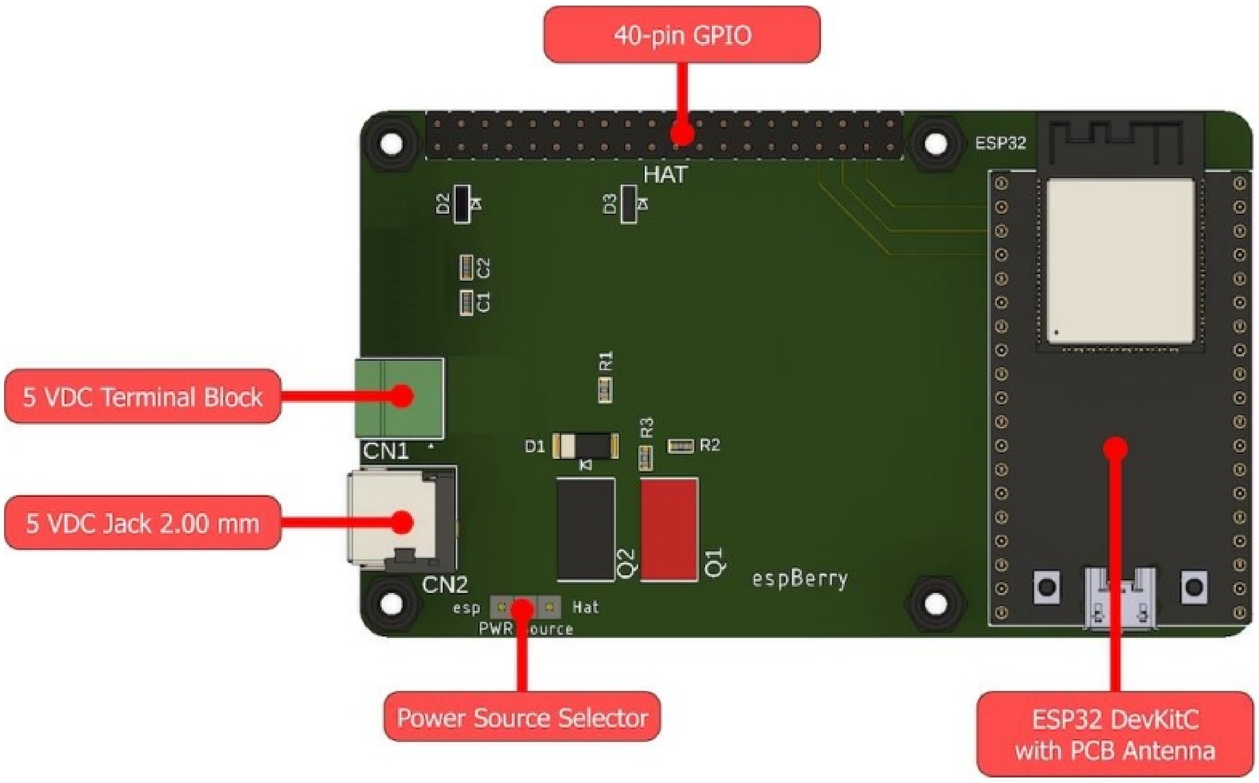
- **Processor: ESP32 DevKitC**
  - 32-Bit Xtensa dual-core @240 MHz
  - WiFi IEEE 802.11 b/g/n 2.4 GHz
  - Bluetooth 4.2 BR/EDR and BLE
  - 520 kB SRAM (16 kB for cache)
  - 448 kB ROM

    - 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

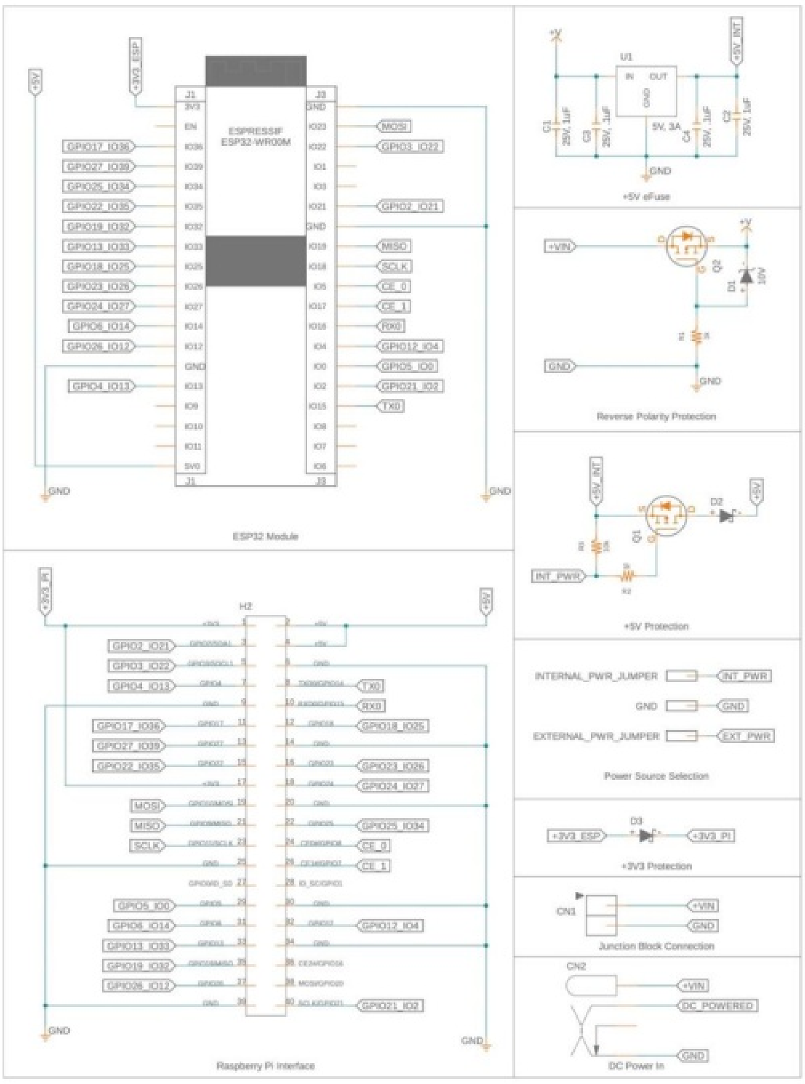


Power Source Connector

- 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

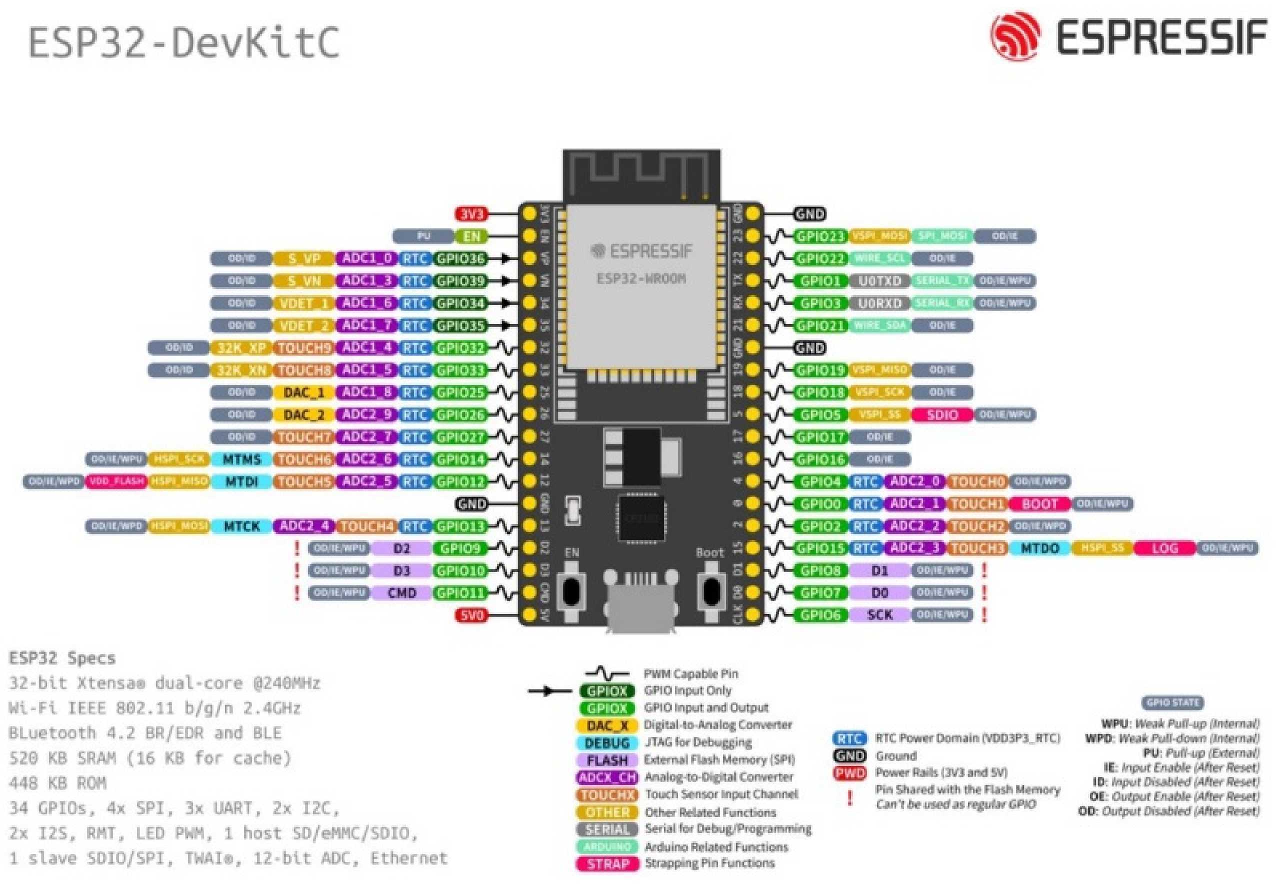
- The espBerry was designed to map as many signals (GPIO, SPI, UART, etc.) as possible. However, that does not necessarily mean that the espBerry covers all HATs available in the market. Your ultimate source for adaptations and developing your own HAT must be the espBerry’s schematic.



- 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

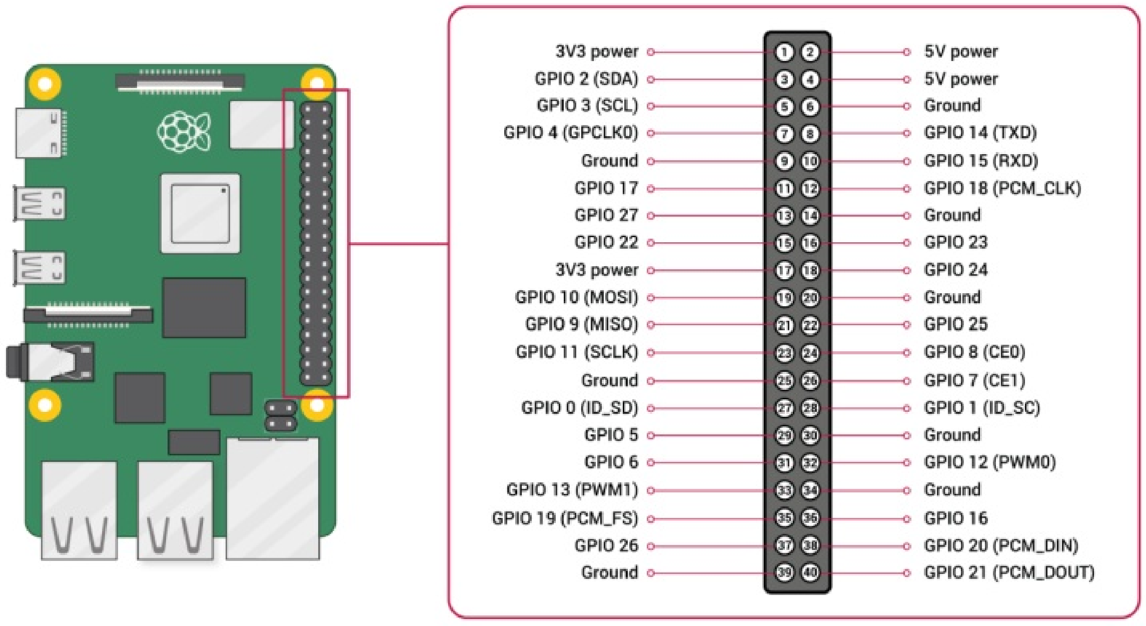
For a full view of the above image, click here.


ESP32-DevKitC
ESPRESSIF
ESP32-WROOM
3V3
EN
GPIO36
GPIO39
GPIO34
GPIO35
GPIO32
GPIO33
GPIO25
GPIO26
GPIO27
GPIO14
GPIO12
GND
GPIO13
GPIO9
GPIO10
GPIO11
5V0
GND
GPIO23
GPIO22
GPIO1
GPIO3
GPIO21
GND
GPIO19
GPIO18
GPIO5
GPIO17
GPIO16
GPIO4
GPIO0
GPIO2
GPIO15
GPIO8
GPIO7
GPIO6
EN
Boot
PWM Capable Pin
GPIOX GPIO Input Only
GPIOX GPIO Input and Output
DAC_X Digital-to-Analog Converter
DEBUG JTAG for Debugging
FLASH External Flash Memory (SPI)
ADCX_CH Analog-to-Digital Converter
TOUCHX Touch Sensor Input Channel
OTHER Other Related Functions
SERIAL Serial for Debug/Programming
ARDUINO Arduino Related Functions
STRAP Strapping Pin Functions
RTC RTC Power Domain (VDD3P3_RTC)
GND Ground
PWD Power Rails (3V3 and 5V)
! Pin Shared with the Flash Memory Can't be used as regular GPIO
GPIO STATE
WPU: Weak Pull-up (Internal)
WPD: Weak Pull-down (Internal)
PU: Pull-up (External)
IE: Input Enable (After Reset)
ID: Input Disabled (After Reset)
OE: Output Enable (After Reset)
OD: Output Disabled (After Reset)

ESP32 Specs
32-bit Xtensa® dual-core @240MHz
Wi-Fi IEEE 802.11 b/g/n 2.4GHz
BLuetooth 4.2 BR/EDR and BLE
520 KB SRAM (16 KB for cache)
448 KB ROM
34 GPIOs, 4x SPI, 3x UART, 2x I2C,
2x I2S, RMT, LED PWM, 1 host SD/eMMC/SDIO,
1 slave SDIO/SPI, TWAI®, 12-bit ADC, Ethernet

The Raspberry Pi 40-pin GPIO Header

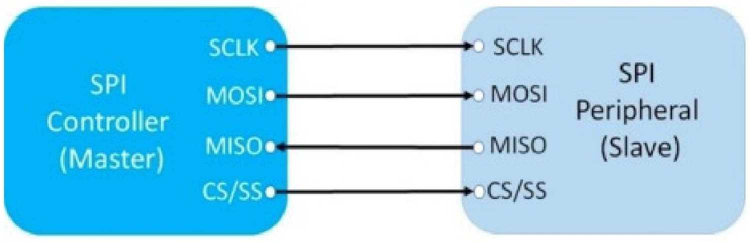
- A powerful feature of the Raspberry Pi is the row of GPIO (general-purpose input/output) pins along the top edge of the board. A 40-pin GPIO header is found on all current Raspberry Pi boards (unpopulated on Raspberry Pi Zero, Raspberry Pi Zero W and Raspberry Pi Zero 2 W). Prior to the Raspberry Pi 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.


3V3 power 1 2 5V power
GPIO 2 (SDA) 3 4 5V power
GPIO 3 (SCL) 5 6 Ground
GPIO 4 (GPCLK0) 7 8 GPIO 14 (TXD)
Ground 9 10 GPIO 15 (RXD)
GPIO 17 11 12 GPIO 18 (PCM_CLK)
GPIO 27 13 14 Ground
GPIO 22 15 16 GPIO 23
3V3 power 17 18 GPIO 24
GPIO 10 (MOSI) 19 20 Ground
GPIO 9 (MISO) 21 22 GPIO 25
GPIO 11 (SCLK) 23 24 GPIO 8 (CE0)
Ground 25 26 GPIO 7 (CE1)
GPIO 0 (ID_SD) 27 28 GPIO 1 (ID_SC)
GPIO 5 29 30 Ground
GPIO 6 31 32 GPIO 12 (PWM0)
GPIO 13 (PWM1) 33 34 Ground
GPIO 19 (PCM_FS) 35 36 GPIO 16
GPIO 26 37 38 GPIO 20 (PCM_DIN)
Ground 39 40 GPIO 21 (PCM_DOUT)

- 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

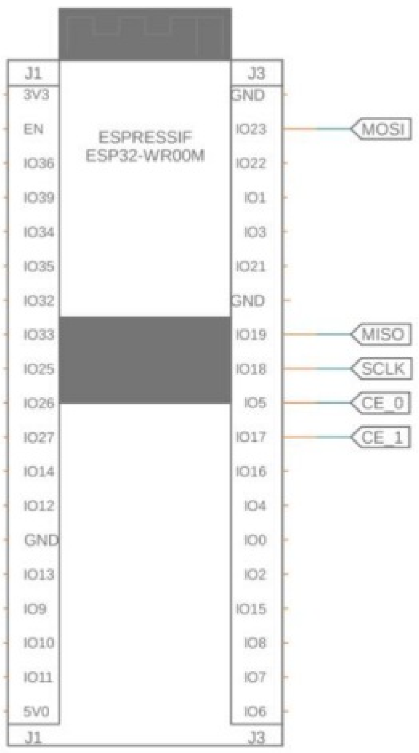
- SPI stands for Serial Peripheral Interface, a serial full-duplex and synchronous interface. The synchronous interface requires a clock signal to transfer and receive data. The clock signal is synchronized between one central control (“master”) and multiple peripheral devices (“slaves”). Unlike UART 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.

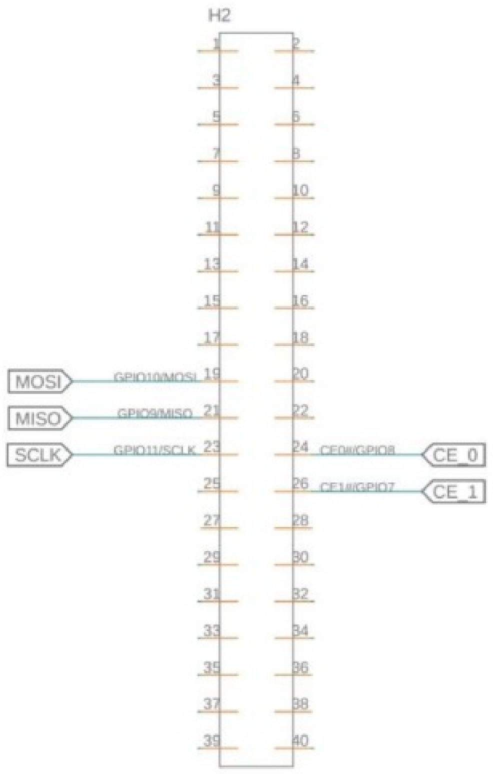


- On the espBerry, the signals assigned to the default IOs:

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.

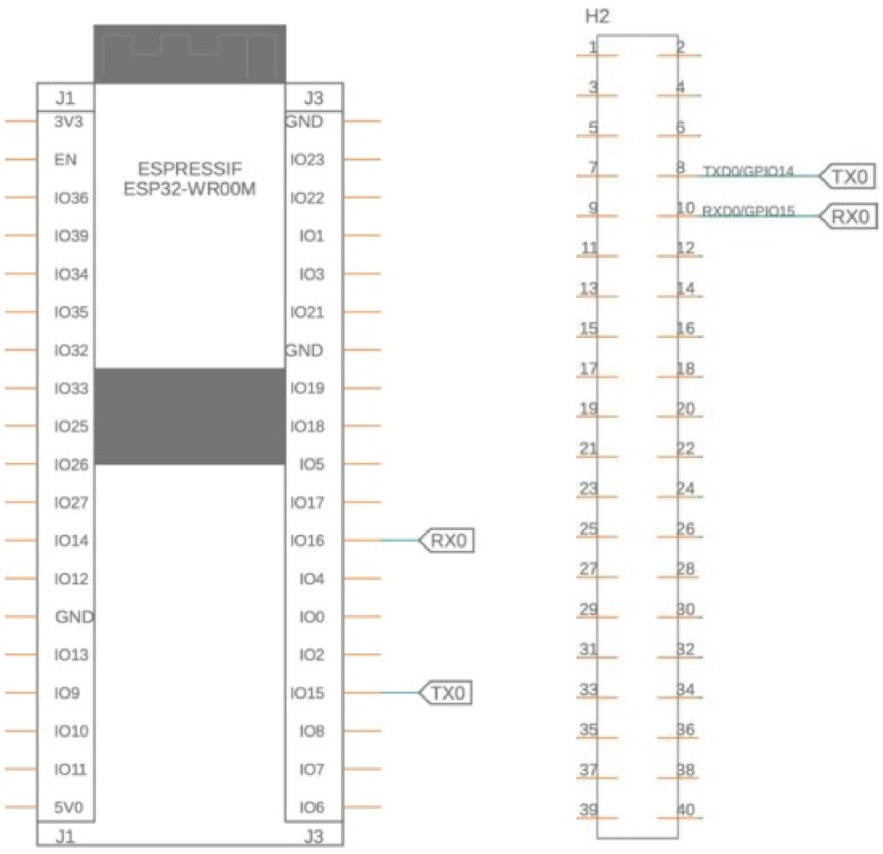




- There are many types of ESP32 boards available. Boards other than the espBerry may have different default SPI pins, but you can find information about default pins from their datasheet. But if default pins are not mentioned, you can find them by using an Arduino sketch (use first link below).
- For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

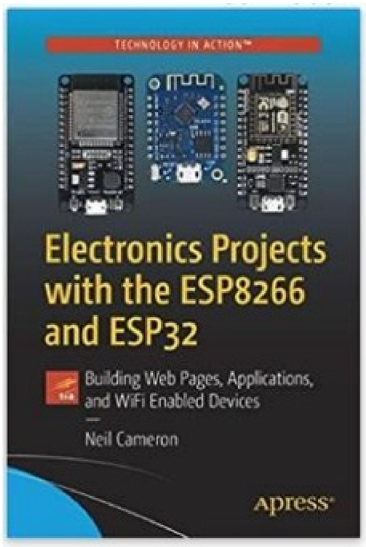
- Besides the onboard USB port, the ESP32 development module has three UART interfaces, i.e., UART0, UART1, and UART2, which provide asynchronous communication at a speed of up to 5 Mbps. These serial ports can be mapped to almost any pin. On the espBerry, we assigned IO15 as Rx and IO16 as Tx, which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

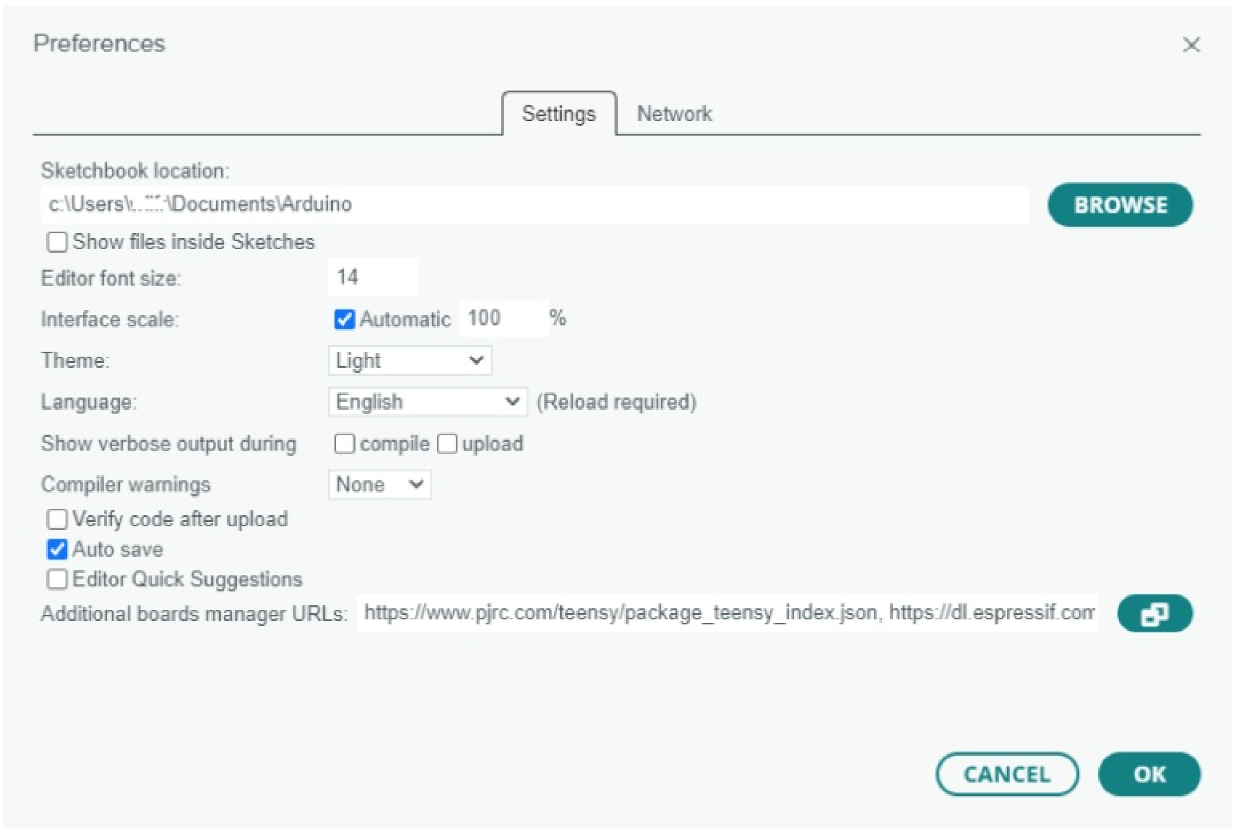
Software

- 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.


TECHNOLOGY IN ACTION™
Electronics Projects with the ESP8266 and ESP32
Building Web Pages, Applications, and WiFi Enabled Devices
Neil Cameron
Apress

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:



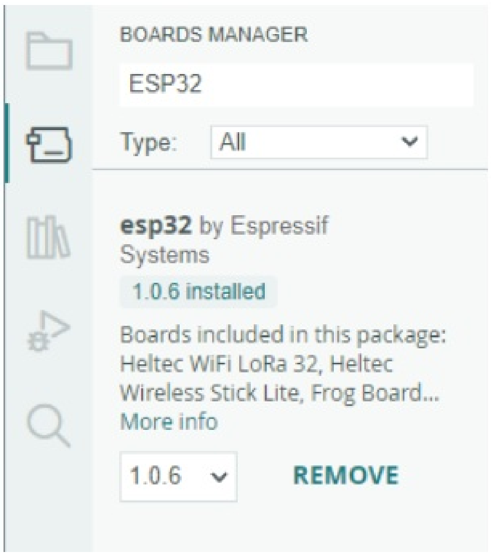
Sketchbook location:
c:\Users\[redacted]\Documents\Arduino
☐ Show files inside Sketches
Editor font size: 14
Interface scale: ☑ Automatic 100 %
Theme: Light
Language: English (Reload required)
Show verbose output during ☐ compile ☐ upload
Compiler warnings None
☐ Verify code after upload
☑ Auto save
☐ Editor Quick Suggestions
Additional boards manager URLs: https://www.pjrc.com/teensy/package_teensy_index.json, https://dl.espressif.com

  - 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.



BOARDS MANAGER

ESP32

Type: All

esp32 by Espressif Systems

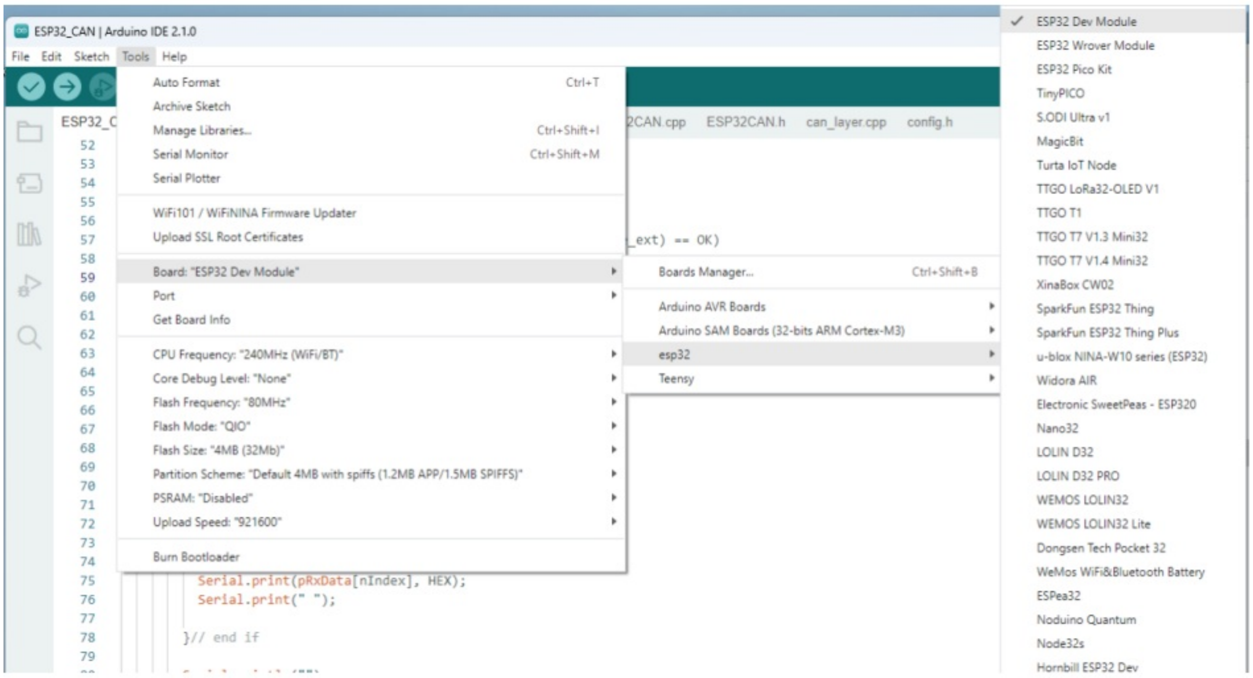
1.0.6 installed

Boards included in this package:
Heltec WiFi LoRa 32, Heltec Wireless Stick Lite, Frog Board...
More info

1.0.6 REMOVE

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:



ESP32_CAN | Arduino IDE 2.1.0

File Edit Sketch Tools Help

ESP32_C 2CAN.cpp ESP32CAN.h can_layer.cpp config.h

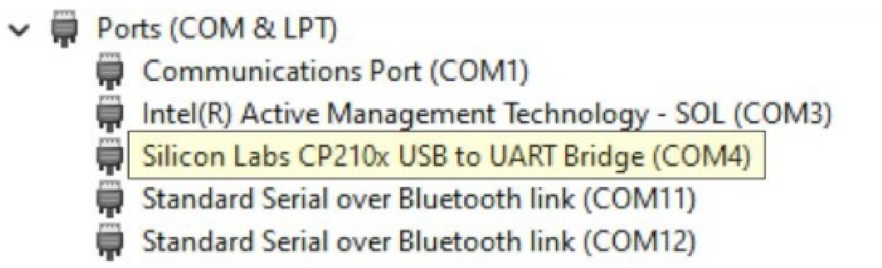
Auto Format Ctrl+T
Archive Sketch
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter
WiFi101 / WiFiNINA Firmware Updater
Upload SSL Root Certificates
Board: "ESP32 Dev Module"
Port
Get Board Info
CPU Frequency: "240MHz (WiFi/BT)"
Core Debug Level: "None"
Flash Frequency: "80MHz"
Flash Mode: "QIO"
Flash Size: "4MB (32Mb)"
Partition Scheme: "Default 4MB with spiffs (1.2MB APP/1.5MB SPIFFS)"
PSRAM: "Disabled"
Upload Speed: "921600"
Burn Bootloader

Boards Manager... Ctrl+Shift+B
Arduino AVR Boards
Arduino SAM Boards (32-bits ARM Cortex-M3)
esp32
Teensy

✓ ESP32 Dev Module
ESP32 Wrover Module
ESP32 Pico Kit
TinyPICO
S.ODI Ultra v1
MagicBit
Turta IoT Node
TTGO LoRa32-OLED V1
TTGO T1
TTGO T7 V1.3 Mini32
TTGO T7 V1.4 Mini32
XinaBox CW02
SparkFun ESP32 Thing
SparkFun ESP32 Thing Plus
u-blox NINA-W10 series (ESP32)
Widora AIR
Electronic SweetPeas - ESP320
Nano32
LOLIN D32
LOLIN D32 PRO
WEMOS LOLIN32
WEMOS LOLIN32 Lite
Dongsen Tech Pocket 32
WeMos WiFi&Bluetooth Battery
ESPea32
Noduino Quantum
Node32s
Hornbill ESP32 Dev

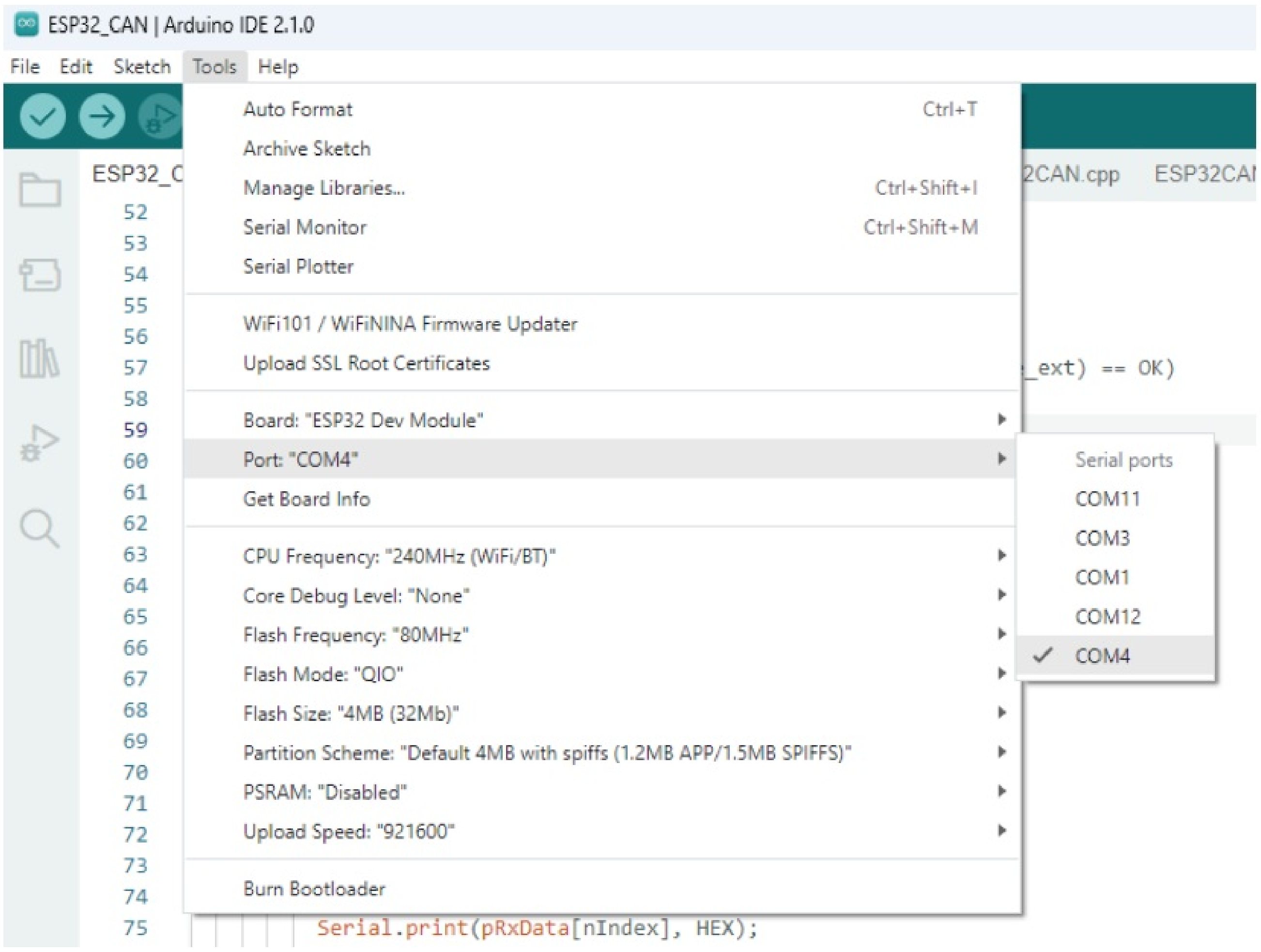
```
_ext) == OK)
        Serial.print(pRxData[nIndex], HEX);
        Serial.print(" ");

      }// end if
```

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



Ports (COM & LPT)
- Communications Port (COM1)
- Intel(R) Active Management Technology - SOL (COM3)
- Silicon Labs CP210x USB to UART Bridge (COM4)
- Standard Serial over Bluetooth link (COM11)
- Standard Serial over Bluetooth link (COM12)

- 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.



ESP32_CAN | Arduino IDE 2.1.0

File Edit Sketch Tools Help

Auto Format	Ctrl+T
Archive Sketch	
Manage Libraries...	Ctrl+Shift+I
Serial Monitor	Ctrl+Shift+M
Serial Plotter	
WiFi101 / WiFiNINA Firmware Updater	
Upload SSL Root Certificates	
Board: "ESP32 Dev Module"	▸
Port: "COM4"	▸
Get Board Info	
CPU Frequency: "240MHz (WiFi/BT)"	▸
Core Debug Level: "None"	▸
Flash Frequency: "80MHz"	▸
Flash Mode: "QIO"	▸
Flash Size: "4MB (32Mb)"	▸
Partition Scheme: "Default 4MB with spiffs (1.2MB APP/1.5MB SPIFFS)"	▸
PSRAM: "Disabled"	▸
Upload Speed: "921600"	▸
Burn Bootloader	

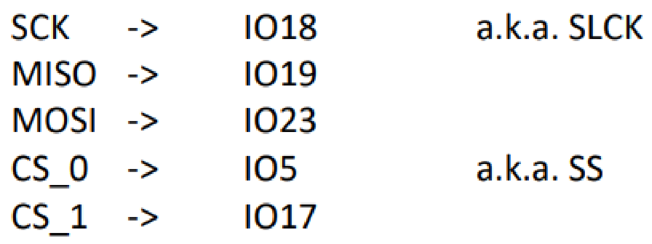
Serial ports
- COM11
- COM3
- COM1
- COM12
- ✓ COM4

```
_ext) == OK)
Serial.print(pRxData[nIndex], HEX);
```

- 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:

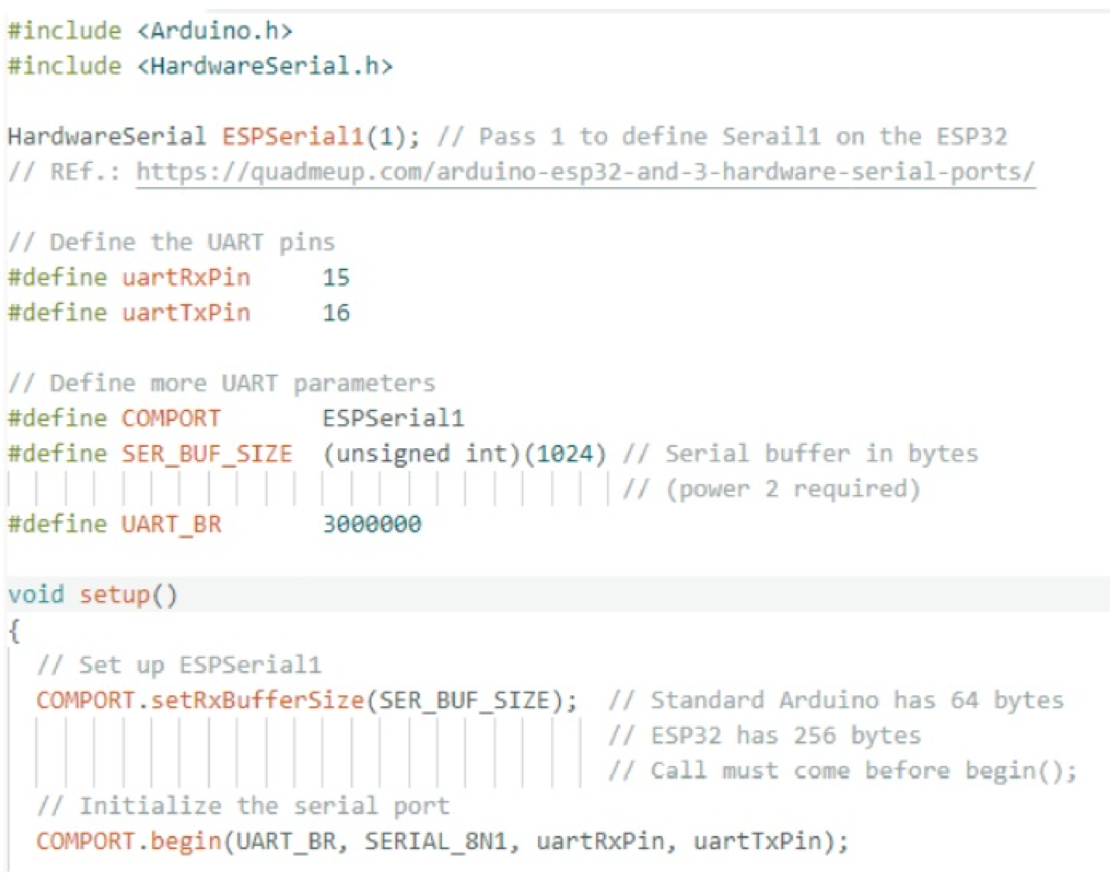


SCK	->	IO18	a.k.a. SLCK
MISO	->	IO19	
MOSI	->	IO23	
CS_0	->	IO5	a.k.a. SS
CS_1	->	IO17	

- 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 .

ABOUT COMPANY

- https://espBerry.com
- https://copperhilltech.com

Documents / Resources

The espBerry Project espBerry - ESP32 Development Board with Raspberry Pi GPIO User Manual	espBerry ESP32 Development Board with Raspberry Pi GPIO [pdf] User Manual ESP32 Development Board with Raspberry Pi GPIO, ESP32, Development Board with Raspberry Pi GPIO, Board with Raspberry Pi GPIO, Raspberry Pi GPIO

References

- GitHub: Let's build from here · GitHub
- Electronics Projects with the ESP8266 and ESP32: Building Web Pages, Applications, and WiFi Enabled Devices: Cameron, Neil: 9781484263358: Books
- Copperhill Technologies - SAE J1939, CAN Bus, NMEA 2000, IoT
- Copperhill Technologies - SAE J1939, CAN Bus, NMEA 2000, IoT
- espBerry Project: ESP32 with CH9102F USB-UART Chip for Serial Speed up to 3Mbit/s - Copperhill
- Contact Us
- dl.espressif.com/dl/package_esp32_index.json
- Using the Arduino Software (IDE) | Arduino Documentation
- docs.espressif.com/projects/esp-idf/en/latest/esp32/_images/esp32-devkitC-v4-pinout.png
- SPI Master Driver - ESP32 - — ESP-IDF Programming Guide latest documentation
- The espBerry Project - The espBerry Project - Extending ESP32 Functionality
- The espBerry Project - The espBerry Project - Extending ESP32 Functionality
- ESP32 Literature Archives - The espBerry Project - Extending ESP32 Functionality
- ESP32 Programming Tips - The espBerry Project - Extending ESP32 Functionality
- GitHub - coryjfowler/MCP_CAN_lib: MCP_CAN Library
- GitHub - raspberrypi/hats
- ESP32 SPI Tutorial Master Slave Communication Example
- ESP32, Arduino and 3 hardware serial ports | QuadMeUp
- ESP32 SPI Communication: Pins, Multiple SPI, Peripherals (Arduino) | Random Nerd Tutorials
- How to use SPI with ESP32 and Arduino - Stack Overflow
- Software | Arduino

- Raspberry Pi Documentation - Raspberry Pi hardware
- User Manual