



ESP32-S2-MINI-1 & ESP32-S2-MINI-1U

User Manual



Preliminary version 0.1
Espressif Systems
Copyright © 2020

About This Guide

This document is intended to help users set up the basic software development environment for developing applications using hardware based on ESP32-S2-MINI-1 and ESP32-S2-MINI-1U modules.

Release Notes

Date	Version	Release notes
Sep. 2020	V0.1	Preliminary release.

Documentation Change Notification

Espressif provides email notifications to keep customers updated on changes to technical documentation. Please subscribe at www.espressif.com/en/subscribe.

Certification

Download certificates for Espressif products from www.espressif.com/en/certificates.

Table of Contents

1. Introduction to ESP32-S2-MINI-1 & ESP32-S2-MINI-1U.....	1
1.1. ESP32-S2-MINI-1 & ESP32-S2-MINI-1U	1
1.2. Pin Description	1
2. Hardware Preparation.....	6
2.1. Hardware Preparation	6
2.2. Hardware Connection	6
3. Getting Started with ESP32-S2-MINI-1 & ESP32-S2-MINI-1U.....	8
3.1. ESP-IDF	8
3.2. Set up the Tools	8
3.2.1. Standard Setup of Toolchain for Windows	8
3.2.2. Standard Setup of Toolchain for Linux	9
3.2.3. Standard Setup of Toolchain for Mac OS.....	10
3.3. Get ESP-IDF	11
3.4. Add IDF_PATH to User Profile.....	11
3.4.1. Windows	11
3.4.2. Linux and MacOS	11
4. Establish Serial Connection with ESP32-S2-MINI-1 & ESP32-S2-MINI-1U	12
4.1. Connect ESP32-S2-MINI-1 and ESP32-S2-MINI-1U to PC.....	12
4.2. Check Port on Windows	12
4.3. Check Port on Linux and MacOS	13
4.4. Adding User to dialout on Linux.....	14
4.5. Verify Serial Connection	14
5. Configure	17
6. Build and Flash.....	18
7. Flash onto the Device	19
8. IDF Monitor	20
9. Examples	21



1. Introduction to ESP32-S2-MINI-1 & ESP32-S2-MINI-1U

1.1. ESP32-S2-MINI-1 & ESP32-S2-MINI-1U

ESP32-S2-MINI-1 and ESP32-S2-MINI-1U are two powerful, generic Wi-Fi MCU modules that target a wide variety of applications, ranging from low-power sensor networks to the most demanding tasks, such as voice encoding, music streaming and MP3 decoding.

Table 1-1. Specifications

Category	Parameters	Description
Wi-Fi	Wi-Fi protocols	802.11 b/g/n
	Operating frequency range	2412 MHz ~ 2484 MHz
Hardware	Peripherals	GPIO, SPI, LCD, UART, I2C, I2S, Camera interface, IR, pulse counter, LED PWM, USB OTG 1.1, ADC, DAC, touch sensor, temperature sensor
	Operating voltage	3.0 V ~ 3.6 V
	Operating current	TX: 120 ~ 190 mA
		RX: 63 ~ 68 mA
	Power supply	Minimum: 500 mA
	Operating temperature	-40 °C ~ 85 °C
	Storage temperature	-40 °C ~ 150 °C
	Dimensions	(18.00±0.10) mm x (31.00±0.10) mm x (3.30±0.10) mm (with shielding box)

1.2. Pin Description

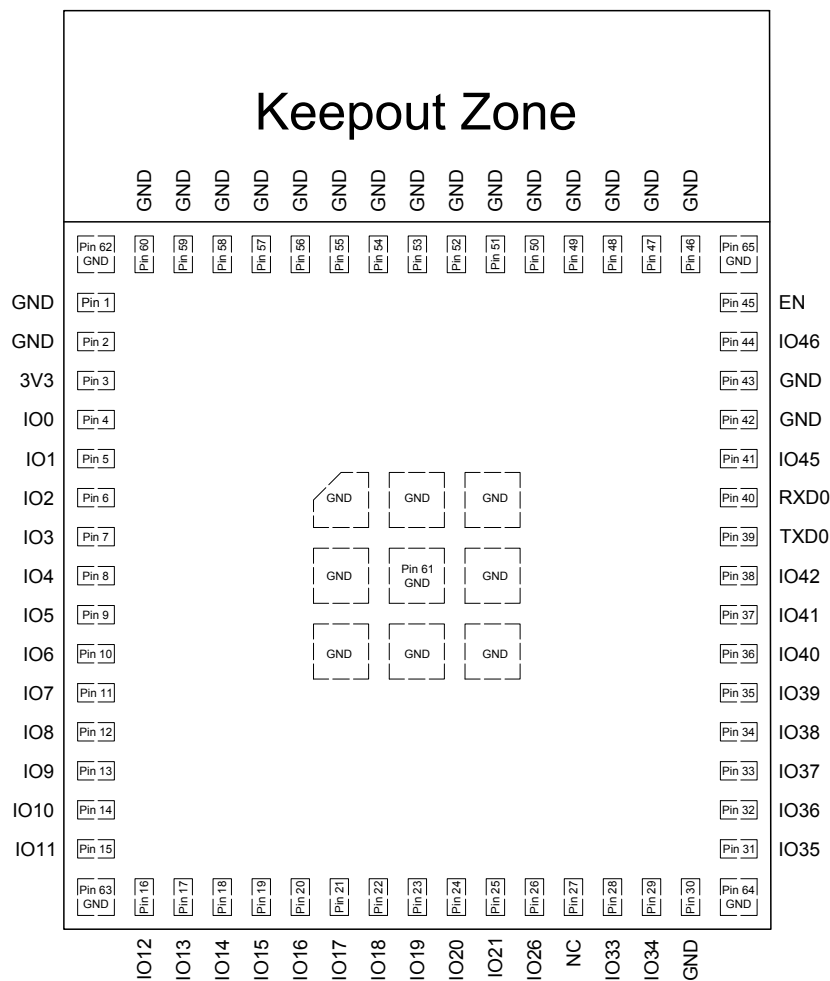


Figure 1-1. ESP32-S2-MINI-1 Pin Layout (Top View)

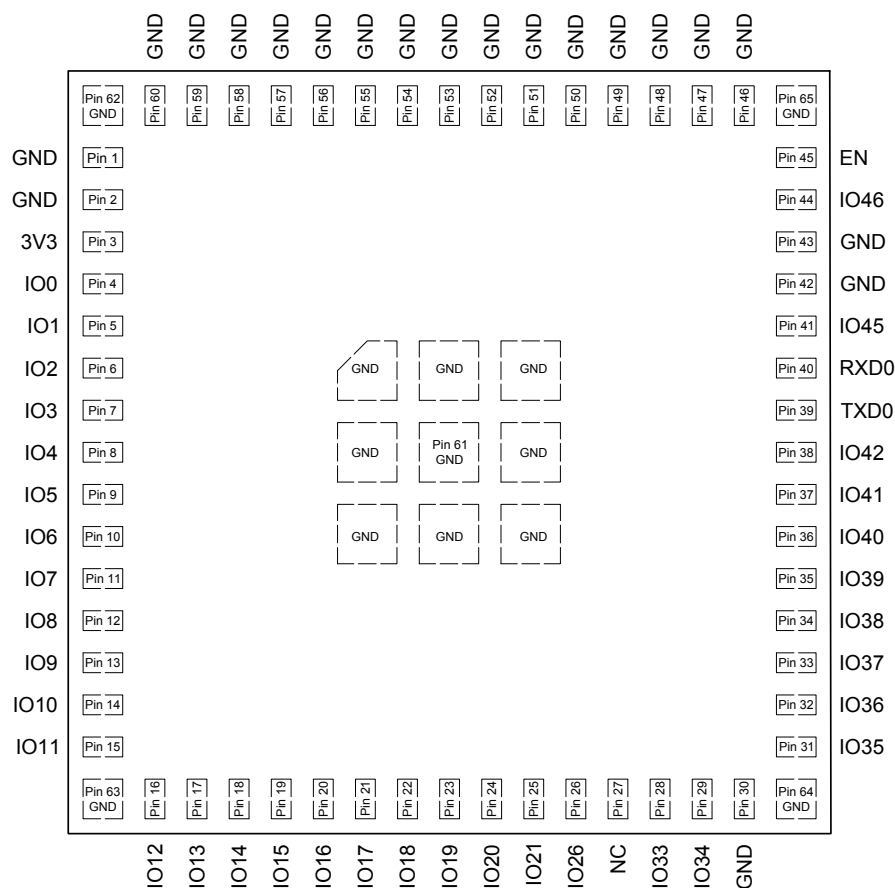


Figure 1-2. ESP32-S2-MINI-1U Pin Layout (Top View)

The modules have 65 pins. which are described in Table 1-2。

Table 1-2. Pin Description

Pin Name	No.	Type	Function Description
GND	1, 2, 30, 42, 43, 46-65	P	Ground
3V3	3	P	Power supply
IO0	4	I/O/T	RTC_GPIO0, GPIO0
IO1	5	I/O/T	RTC_GPIO1, GPIO1, TOUCH1, ADC1_CH0
IO2	6	I/O/T	RTC_GPIO2, GPIO2, TOUCH2, ADC1_CH1
IO3	7	I/O/T	RTC_GPIO3, GPIO3, TOUCH3, ADC1_CH2
IO4	8	I/O/T	RTC_GPIO4, GPIO4, TOUCH4, ADC1_CH3



Pin Name	No.	Type	Function Description
IO5	9	I/O/T	RTC_GPIO5, GPIO5, TOUCH5, ADC1_CH4
IO6	10	I/O/T	RTC_GPIO6, GPIO6, TOUCH6, ADC1_CH5
IO7	11	I/O/T	RTC_GPIO7, GPIO7, TOUCH7, ADC1_CH6
IO8	12	I/O/T	RTC_GPIO8, GPIO8, TOUCH8, ADC1_CH7
IO9	13	I/O/T	RTC_GPIO9, GPIO9, TOUCH9, ADC1_CH8, FSPIHD
IO10	14	I/O/T	RTC_GPIO10, GPIO10, TOUCH10, ADC1_CH9, FSPICS0, FSPIIO4
IO11	15	I/O/T	RTC_GPIO11, GPIO11, TOUCH11, ADC2_CH0, FSPID, FSPIIO5
IO12	16	I/O/T	RTC_GPIO12, GPIO12, TOUCH12, ADC2_CH1, FSPICLK, FSPIIO6
IO13	17	I/O/T	RTC_GPIO13, GPIO13, TOUCH13, ADC2_CH2, FSPIQ, FSPIIO7
IO14	18	I/O/T	RTC_GPIO14, GPIO14, TOUCH14, ADC2_CH3, FSPIWP, FSPIDQS
IO15	19	I/O/T	RTC_GPIO15, GPIO15, U0RTS, ADC2_CH4, XTAL_32K_P
IO16	20	I/O/T	RTC_GPIO16, GPIO16, U0CTS, ADC2_CH5, XTAL_32K_N
IO17	21	I/O/T	RTC_GPIO17, GPIO17, U1TXD, ADC2_CH6, DAC_1
IO18	22	I/O/T	RTC_GPIO18, GPIO18, U1RXD, ADC2_CH7, DAC_2, CLK_OUT3
IO19	23	I/O/T	RTC_GPIO19, GPIO19, U1RTS, ADC2_CH8, CLK_OUT2, USB_D-
IO20	24	I/O/T	RTC_GPIO20, GPIO20, U1CTS, ADC2_CH9, CLK_OUT1, USB_D+
IO21	25	I/O/T	RTC_GPIO21, GPIO21
IO26	26	I/O/T	SPICS1, GPIO26
NC	27	-	NC
IO33	28	I/O/T	SPIIO4, GPIO33, FSPIHD
IO34	29	I/O/T	SPIIO5, GPIO34, FSPICS0
IO35	31	I/O/T	SPIIO6, GPIO35, FSPID
IO36	32	I/O/T	SPIIO7, GPIO36, FSPICLK
IO37	33	I/O/T	SPIDQS, GPIO37, FSPIQ
IO38	34	I/O/T	GPIO38, FSPIWP
IO39	35	I/O/T	MTCK, GPIO39, CLK_OUT3
IO40	36	I/O/T	MTDO, GPIO40, CLK_OUT2
IO41	37	I/O/T	MTDI, GPIO41, CLK_OUT1
IO42	38	I/O/T	MTMS, GPIO42
TXD0	39	I/O/T	U0TXD, GPIO43, CLK_OUT1
RXD0	40	I/O/T	U0RXD, GPIO44, CLK_OUT2
IO45	41	I/O/T	GPIO45



Pin Name	No.	Type	Function Description
IO46	44	I	GPIO46
EN	45	I	High: on, enables the chip. Low: off, the chip powers off. Note: Do not leave the EN pin floating



2. Hardware Preparation

2.1. Hardware Preparation

- ESP32-S2-MINI-1 and ESP32-S2-MINI-1U modules
- Espressif RF testing board
- One USB-TTL serial module
- PC, Windows 7 recommended
- Micro-USB cable

2.2. Hardware Connection

1. Connect ESP32-S2-MINI-1, ESP32-S2-MINI-1U and the RF testing board, as Figure 2-1 shows.

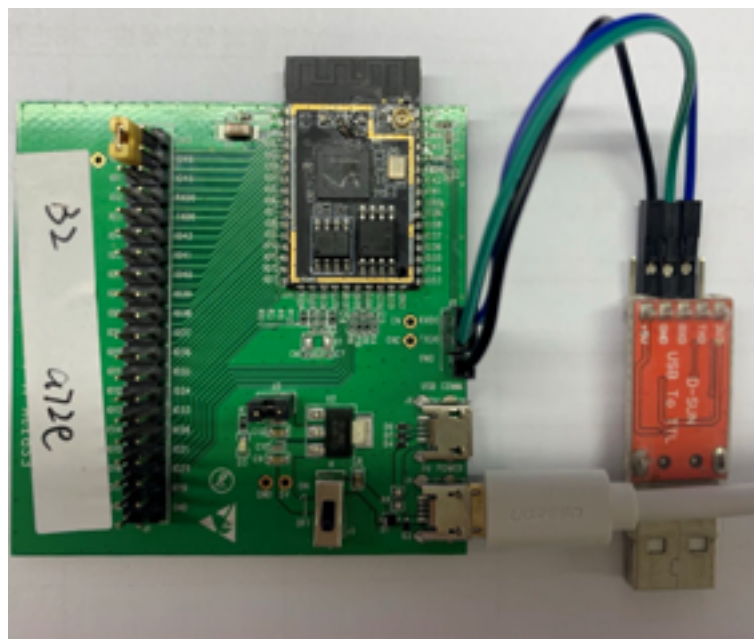


Figure 2-1. Testing Environment Setup

2. Connect USB -UART serial module to the RF testing board via TXD, RXD and GND.
3. Connect USB-UART module to the PC.
4. Connect the RF testing board to the PC or a power adaptor to enable 5 V power supply, via the Micro-USB cable.
5. During download, short IO0 to GND via a jumper. Then, turn "ON" the board.
6. Download firmware into flash using the download tool ESP32-S2 DOWNLOAD TOOL.
7. After download, remove the jumper on IO0 and GND.



8. Power up the RF testing board again. ESP32-S2-MINI-1 and ESP32-S2-MINI-1U will switch to working mode. The chip will read programs from flash upon initialization.

 Notes:

- *IO0 is internally logic high.*
- *For more information on ESP32-S2-MINI-1 and ESP32-S2-MINI-1U, please refer to ESP32-S2-MINI-1 and ESP32-S2-MINI-1U Datasheet.*



3. Getting Started with ESP32-S2-MINI-1 & ESP32-S2-MINI-1U

3.1. ESP-IDF

The Espressif IoT Development Framework (ESP-IDF for short) is a framework for developing applications based on the Espressif ESP32. Users can develop applications with ESP32-S2 in Windows/Linux/MacOS based on ESP-IDF.

3.2. Set up the Tools

Aside from the ESP-IDF, you also need to install the tools used by ESP-IDF, such as the compiler, debugger, Python packages, etc.

3.2.1. Standard Setup of Toolchain for Windows

The quickest way is to download the toolchain and MSYS2 zip from [dl.espressif.com](https://dl.espressif.com/dl/toolchains/preview/xtensa-esp32s2-elf-gcc8_2_0-esp32s2-dev-4-g3a626e-win32.zip):
https://dl.espressif.com/dl/toolchains/preview/xtensa-esp32s2-elf-gcc8_2_0-esp32s2-dev-4-g3a626e-win32.zip

Checking out

Run **C:\msys32\mingw32.exe** to open an MSYS2 terminal. Run:

```
mkdir -p ~/esp
```

Input `cd ~/esp` to enter the new directory.

Updating the Environment

When IDF is updated, sometimes new toolchains are required or new requirements are added to the Windows MSYS2 environment. To move any data from an old version of the precompiled environment to a new one:

Take the old MSYS2 environment (ie **C:\msys32**) and move/rename it to a different directory (ie **C:\msys32_old**).

Download the new precompiled environment using the steps above.

Unzip the new MSYS2 environment to **C:\msys32** (or another location).

Find the old **C:\msys32_old\home** directory and move this into **C:\msys32**.

You can now delete the **C:\msys32_old** directory if you no longer need it.

You can have independent different MSYS2 environments on your system, as long as they are in different directories.



3.2.2. Standard Setup of Toolchain for Linux

Install Prerequisites

CentOS 7:

```
sudo yum install gcc git wget make ncurses-devel flex bison gperf python pyserial python-pyelftools
```

Ubuntu 和 Debian:

```
sudo apt-get install gcc git wget make libncurses-dev flex bison gperf python python-pip python-setuptools python-serial python-cryptography python-future python-pyparsing python-pyelftools
```

Arch:

```
sudo pacman -S --needed gcc git make ncurses flex bison gperf python2-pyserial python2-cryptography python2-future python2-pyparsing python2-pyelftools
```

Set up The Toolchain

64-bit Linux:

https://dl.espressif.com/dl/toolchains/preview/xtensa-esp32s2-elf-gcc8_2_0-esp32s2-dev-4-g3a626e-linux-amd64.tar.gz

32-bit Linux:

https://dl.espressif.com/dl/toolchains/preview/xtensa-esp32s2-elf-gcc8_2_0-esp32s2-dev-4-g3a626e-linux-i686.tar.gz

1. Unzip the file to **~/esp** directory:

64-bit Linux:

```
mkdir -p ~/esp  
cd ~/esp  
tar -xzf ~/Downloads/xtensa-esp32s2-elf-gcc8_2_0-esp32s2-dev-4-g3a626e-linux-amd64.tar.gz
```

32-bit Linux:

```
mkdir -p ~/esp  
cd ~/esp  
tar -xzf ~/Downloads/xtensa-esp32s2-elf-gcc8_2_0-esp32s2-dev-4-g3a626e-linux-i686.tar.gz
```

2. The toolchain will be unzipped to **~/esp/xtensa-esp32s2-elf/** directory.

Add the following to **~/.profile**:

```
export PATH="$HOME/esp/xtensa-esp32s2-elf/bin:$PATH"
```

Optionally, add the following to **~/.profile**:

```
alias get_esp32s2='export PATH="$HOME/esp/xtensa-esp32s2-elf/bin:$PATH"'
```

3. Re-log in to validate **.profile**. Run the following to check PATH:

```
printenv PATH
```



```
$ printenv PATH
/home/user-name/esp/xtensa-esp32s2-elf/bin:/home/user-name/bin:/home/user-name/.local/
bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/
games:/snap/bin
```

Permission issues /dev/ttyUSB0

某些 Linux 版本可能在烧写 ESP32-S2 时会出现 Failed to open port /dev/ttyUSB0 错误消息。可以通过将当前用户添加到拨出组来解决。

With some Linux distributions you may get the Failed to open port /dev/ttyUSB0 error message when flashing the ESP32. This can be solved by adding the current user to the dialout group.

Arch Linux Users¶

To run the precompiled gdb (xtensa-esp32-elf-gdb) in Arch Linux requires ncurses 5, but Arch uses ncurses 6.

Backwards compatibility libraries are available in AUR for native and lib32 configurations:

<https://aur.archlinux.org/packages/ncurses5-compat-libs/>

<https://aur.archlinux.org/packages/lib32-ncurses5-compat-libs/>

Before installing these packages you might need to add the author's public key to your keyring as described in the "Comments" section at the links above.

Alternatively, use crosstool-NG to compile a gdb that links against ncurses 6.

3.2.3. Standard Setup of Toolchain for Mac OS

Install pip:

```
sudo easy_install pip
```

Install Toolchain:

https://dl.espressif.com/dl/toolchains/preview/xtensa-esp32s2-elf-gcc8_2_0-esp32s2-dev-4-g3a626e-macos.tar.gz

Unzip the file into **~/esp** directory.

The toolchain will be unzipped into **~/esp/xtensa-esp32s2-elf/** path.

Add the following to **~/.profile**:

```
export PATH=$HOME/esp/xtensa-esp32s2-elf/bin:$PATH
```

Optionally, add the following to **~/.profile**:

```
alias get_esp32s2="export PATH=$HOME/esp/xtensa-esp32s2-elf/bin:$PATH"
```

Input **get_esp32s2** to add the toolchain to PATH.



3.3. Get ESP-IDF

Once you have the toolchain (that contains programs to compile and build the application) installed, you also need ESP32 specific API / libraries. They are provided by Espressif in [ESP-IDF repository](https://github.com/espressif/esp-idf). To get it, open terminal, navigate to the directory you want to put ESP-IDF, and clone it using `git clone` command:

```
git clone --recursive -b feature/esp32s2beta https://github.com/espressif/esp-idf.git
```

ESP-IDF will be downloaded into `~/esp/esp-idf`.

 Note:

Do not miss the --recursive option. If you have already cloned ESP-IDF without this option, run another command to get all the submodules:

```
cd ~/esp/esp-idf
git submodule update --init
```

3.4. Add IDF_PATH to User Profile

To preserve setting of IDF_PATH environment variable between system restarts, add it to the user profile, following instructions below.

3.4.1. Windows

Search for "Edit Environment Variables" on Windows 10.

Click New... and add a new system variable IDF_PATH. The configuration should include an ESP-IDF directory, such as **C:\Users\user-name\esp\esp-idf**.

Add `;%IDF_PATH%\tools` to the Path variable to run `idf.py` and other tools.

3.4.2. Linux and MacOS

Add the following to `~/.profile`:

```
export IDF_PATH=~/esp/esp-idf
export PATH="$IDF_PATH/tools:$PATH"
```

Run the following to check IDF_PATH:

```
printenv IDF_PATH
```

Run the following to check if `idf.py` is included in PATH:

```
which idf.py
```

It will print a path similar to **`${IDF_PATH}/tools/idf.py`**.

You can also enter the following if you do not want to modify IDF_PATH or PATH:

```
export IDF_PATH=~/esp/esp-idf
export PATH="$IDF_PATH/tools:$PATH"
```



4. Establish Serial Connection with ESP32-S2-MINI-1 & ESP32-S2-MINI-1U

This section provides guidance how to establish serial connection between ESP32-S2-MINI-1 and ESP32-S2-MINI-1U and PC.

4.1. Connect ESP32-S2-MINI-1 and ESP32-S2-MINI-1U to PC

Connect the ESP32 board to the PC using the USB cable. If device driver does not install automatically, identify USB to serial converter chip on your ESP32 board (or external converter dongle), search for drivers in internet and install them.

Below are the links to drivers for ESP32-S2-MINI-1 and ESP32-S2-MINI-1U boards produced by Espressif:

[CP210x USB to UART Bridge VCP Drivers](#)

[FTDI Virtual COM Port Drivers](#)

The drivers above are primarily for reference. Under normal circumstances, the drivers should be bundled with and operating system and automatically installed upon connecting one of the listed boards to the PC.

4.2. Check Port on Windows

Check the list of identified COM ports in the Windows Device Manager. Disconnect ESP32-S2 and connect it back, to verify which port disappears from the list and then shows back again.

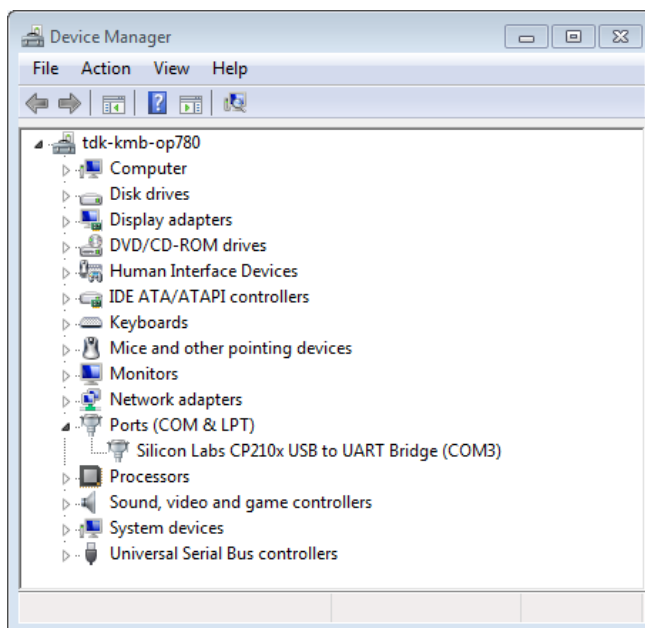


Figure 4-1. USB to UART bridge of ESP32-S2 Board in Windows Device Manager

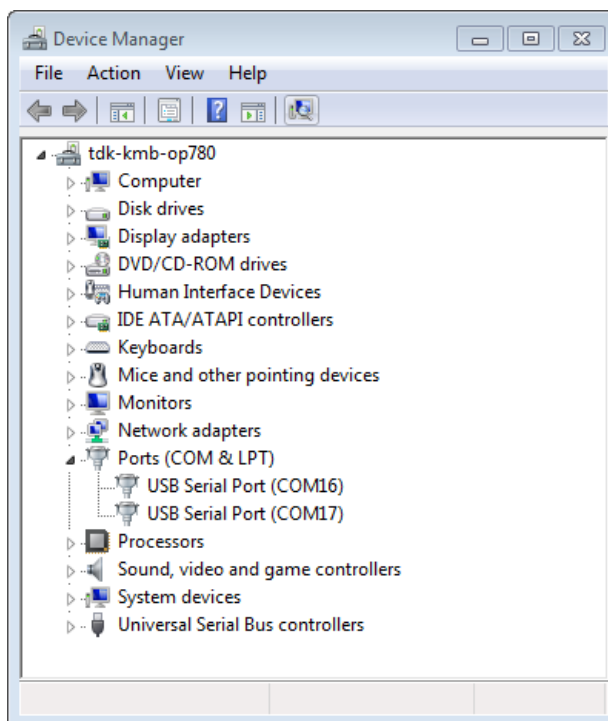


Figure 4-2. Two USB Serial Ports of ESP32-S2 Board in Windows Device Manager

4.3. Check Port on Linux and MacOS

To check the device name for the serial port of your ESP32-S2 board (or external converter dongle), run this command two times, first with the board / dongle unplugged, then with plugged in. The port which appears the second time is the one you need:

Linux



```
ls /dev/tty*
```

MacOS

```
ls /dev/cu.*
```

4.4. Adding User to dialout on Linux

The currently logged user should have read and write access the serial port over USB. On most Linux distributions, this is done by adding the user to dialout group with the following command:

```
sudo usermod -a -G dialout $USER
```

on Arch Linux this is done by adding the user to uucp group with the following command:

```
sudo usermod -a -G uucp $USER
```

Make sure you re-login to enable read and write permissions for the serial port.

4.5. Verify Serial Connection

Now verify that the serial connection is operational. You can do this using a serial terminal program. In this example we will use PuTTY SSH Client that is available for both Windows and Linux. You can use other serial program and set communication parameters like below.

Run terminal, set identified serial port, baud rate = 115200, data bits = 8, stop bits = 1, and parity = N. Below are example screen shots of setting the port and such transmission parameters (in short described as 115200-8-1-N) on Windows and Linux. Remember to select exactly the same serial port you have identified in steps above.

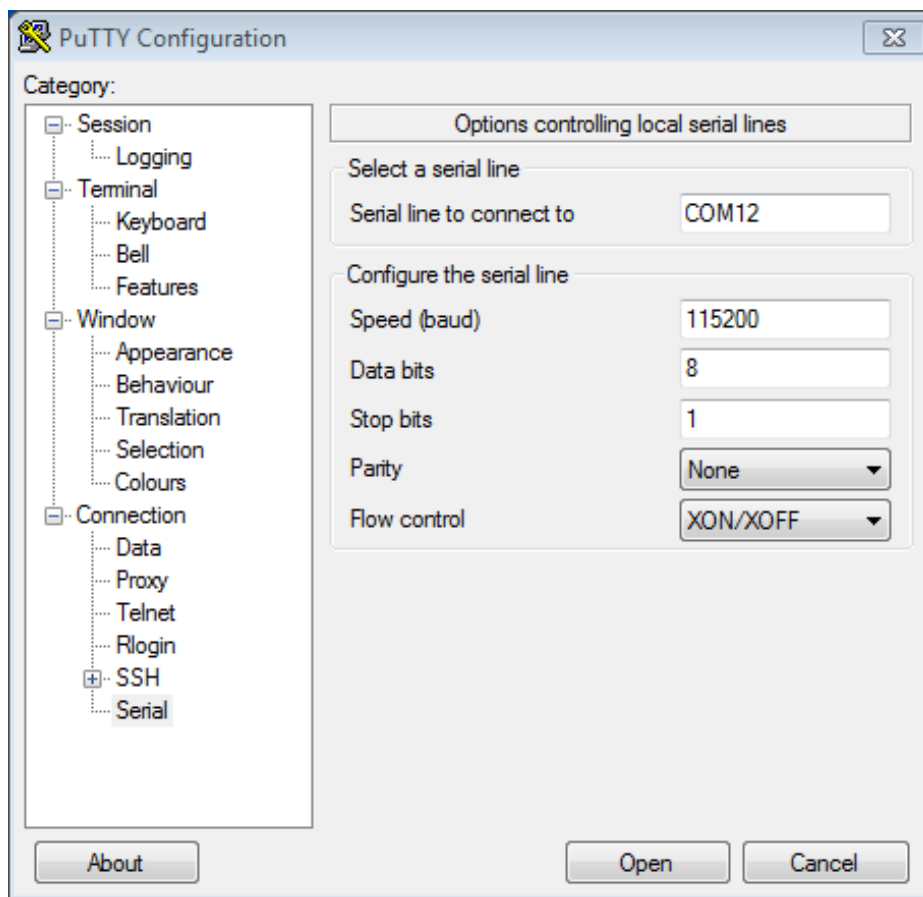


Figure 4-3. Setting Serial Communication in PuTTY on Windows

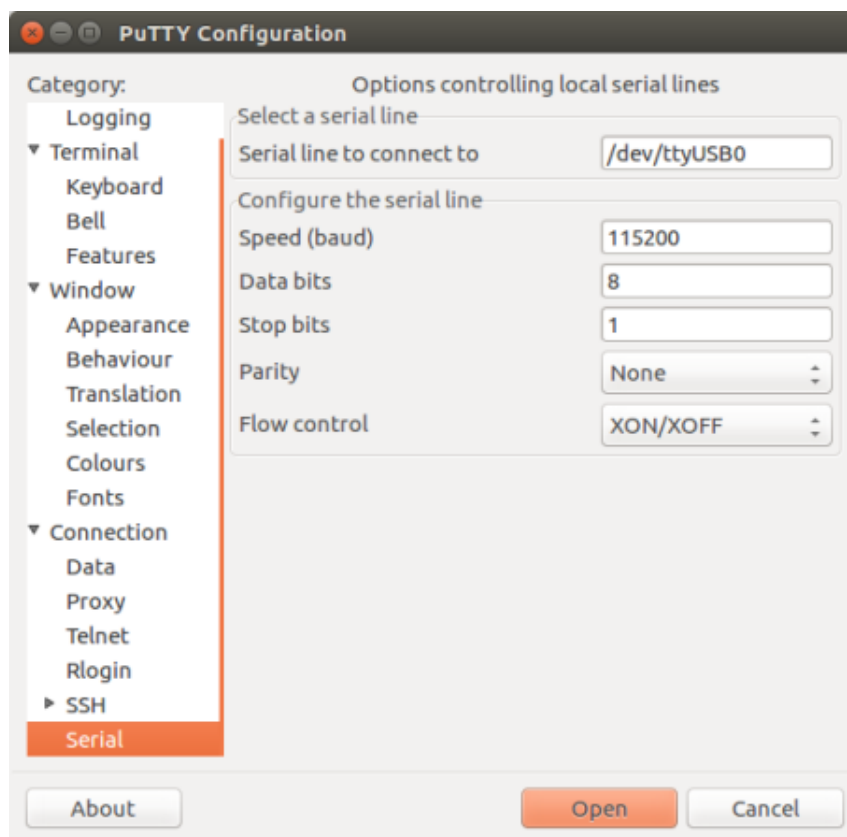


Figure 4-4. Setting Serial Communication in PuTTY on Linux

Then open serial port in terminal and check, if you see any log printed out by ESP32-S2. The log contents will depend on application loaded to ESP32-S2.

Notes:

- For some serial port wiring configurations, the serial RTS & DTR pins need to be disabled in the terminal program before the ESP32-S2 will boot and produce serial output. This depends on the hardware itself, most development boards (including all Espressif boards) do not have this issue. The issue is present if RTS & DTR are wired directly to the EN & GPIO0 pins. See the esptool documentation for more details.
- Close serial terminal after verification that communication is working. In the next step we are going to use a different application to upload a new firmware to ESP32-S2. This application will not be able to access serial port while it is open in terminal.



5.

Configure

Enter *hello_world* directory and run menuconfig.

Linux and MacOS

```
cd ~/esp/hello_world  
idf.py -DIDF_TARGET=esp32s2beta menuconfig
```

You may need to run python2 idf.py on Python 3.0.

Windows

```
cd %userprofile%\esp\hello_world  
idf.py -DIDF_TARGET=esp32s2beta menuconfig
```

The Python 2.7 installer will attempt to configure Windows to associate a .py file with Python 2. If other programs (such as Visual Studio Python tools) have been associated with other versions of Python, idf.py may not work properly (the file will open in Visual Studio). In this case, you can choose to run C:\Python27\python idf.py every time, or change the Windows .py associated file settings.



6.

Build and Flash

Now you can build and flash the application. Run:

```
idf.py build
```

This will compile the application and all the ESP-IDF components, generate bootloader, partition table, and application binaries, and flash these binaries to your ESP32-S2 board.

```
$ idf.py build
Running cmake in directory /path/to/hello_world/build
Executing "cmake -G Ninja --warn-uninitialized /path/to/hello_world"...
Warn about uninitialized values.
-- Found Git: /usr/bin/git (found version "2.17.0")
-- Building empty aws_iot component due to configuration
-- Component names: ...
-- Component paths: ...

... (more lines of build system output)

[527/527] Generating hello-world.bin
esptool.py v2.3.1

Project build complete. To flash, run this command:
../../components/esptool_py/esptool/esptool.py -p (PORT) -b 921600 write_flash --
flash_mode dio --flash_size detect --flash_freq 40m 0x10000 build/hello-world.bin build
0x1000 build/bootloader/bootloader.bin 0x8000 build/partition_table/partition-table.bin
or run 'idf.py -p PORT flash'
```

If there are no issues, at the end of build process, you should see generated .bin files.



7. Flash onto the Device

Flash the binaries that you just built onto your ESP32-S2 board by running:

```
idf.py -p PORT [-b BAUD] flash
```

Replace PORT with your ESP32-S2 board's serial port name. You can also change the flasher baud rate by replacing BAUD with the baud rate you need. The default baud rate is 460800.

```
Running esptool.py in directory [...]/esp/hello_world
Executing "python [...]/esp-idf/components/esptool_py/esptool/esptool.py -b 460800
write_flash @flash_project_args"...
esptool.py -b 460800 write_flash --flash_mode dio --flash_size detect --flash_freq 40m
0x1000 bootloader/bootloader.bin 0x8000 partition_table/partition-table.bin 0x10000 hello-
world.bin
esptool.py v2.3.1
Connecting...
Detecting chip type... ESP32
Chip is ESP32D0WDQ6 (revision 1)
Features: WiFi, BT, Dual Core
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 460800
Changed.
Configuring flash size...
Auto-detected Flash size: 4MB
Flash params set to 0x0220
Compressed 22992 bytes to 13019...
Wrote 22992 bytes (13019 compressed) at 0x00001000 in 0.3 seconds (effective 558.9 kbit/
s)...
Hash of data verified.
Compressed 3072 bytes to 82...
Wrote 3072 bytes (82 compressed) at 0x00008000 in 0.0 seconds (effective 5789.3 kbit/s)...
Hash of data verified.
Compressed 136672 bytes to 67544...
Wrote 136672 bytes (67544 compressed) at 0x00010000 in 1.9 seconds (effective 567.5 kbit/
s)...
Hash of data verified.

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

If there are no issues by the end of the flash process, the module will be reset and the “hello_world” application will be running.



8.

IDF Monitor

To check if "hello_world" is indeed running, type `idf.py -p PORT monitor` (Do not forget to replace PORT with your serial port name).

This command launches the monitor application:

```
$ idf.py -p /dev/ttyUSB0 monitor

Running idf_monitor in directory [...]/esp/hello_world/build

Executing "python [...]/esp-idf/tools/idf_monitor.py -b 115200 [...]/esp/hello_world/build/hello-world.elf"...

--- idf_monitor on /dev/ttyUSB0 115200 ---

--- Quit: Ctrl+] | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---

ets Jun  8 2016 00:22:57

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)

ets Jun  8 2016 00:22:57

...
```

After startup and diagnostic logs scroll up, you should see "Hello world!" printed out by the application.

```
...

Hello world!

Restarting in 10 seconds...

I (211) cpu_start: Starting scheduler on APP CPU.

Restarting in 9 seconds...

Restarting in 8 seconds...

Restarting in 7 seconds...
```

To exit IDF monitor use the shortcut Ctrl+].

If IDF monitor fails shortly after the upload, or, if instead of the messages above, you see random garbage similar to what is given below, your board is likely using a 26MHz crystal. Most development board designs use 40MHz, so ESP-IDF uses this frequency as a default value.



9.

Examples

For ESP-IDF examples, please go to [ESP-IDF GitHub](#).



Espressif IoT Team
www.espressif.com

Disclaimer and Copyright Notice

Information in this document, including URL references, is subject to change without notice.

THIS DOCUMENT IS PROVIDED AS IS WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE.

All liability, including liability for infringement of any proprietary rights, relating to use of information in this document is disclaimed. No licenses express or implied, by estoppel or otherwise, to any intellectual property rights are granted herein.

The Wi-Fi Alliance Member logo is a trademark of the Wi-Fi Alliance. The Bluetooth logo is a registered trademark of Bluetooth SIG.

All trade names, trademarks and registered trademarks mentioned in this document are property of their respective owners, and are hereby acknowledged.

Copyright © 2020 Espressif Inc. All rights reserved.