ESP32-WATG-32D

User Manual

About This Guide

This document is intended to help users set up the basic software development environment for developing applications using hardware based on the ESP32-WATG-32D module.

Release Notes

Date	Version	Release notes
2019.12	V0.1	Preliminary release.

Table of Contents

1. Introduction to ESP32-WATG-32D

1.1. ESP32-WATG-32D

ESP32-WATG-32D is a custom WiFi-BT-BLE MCU module for giving the "Connectivity Function" to customer's different products, including Water Heater and Comfort Heating Systems.

Table 1 provides the specifications of ESP32-WATG-32D.

Table 1: ESP32-WATG-32D Specifications

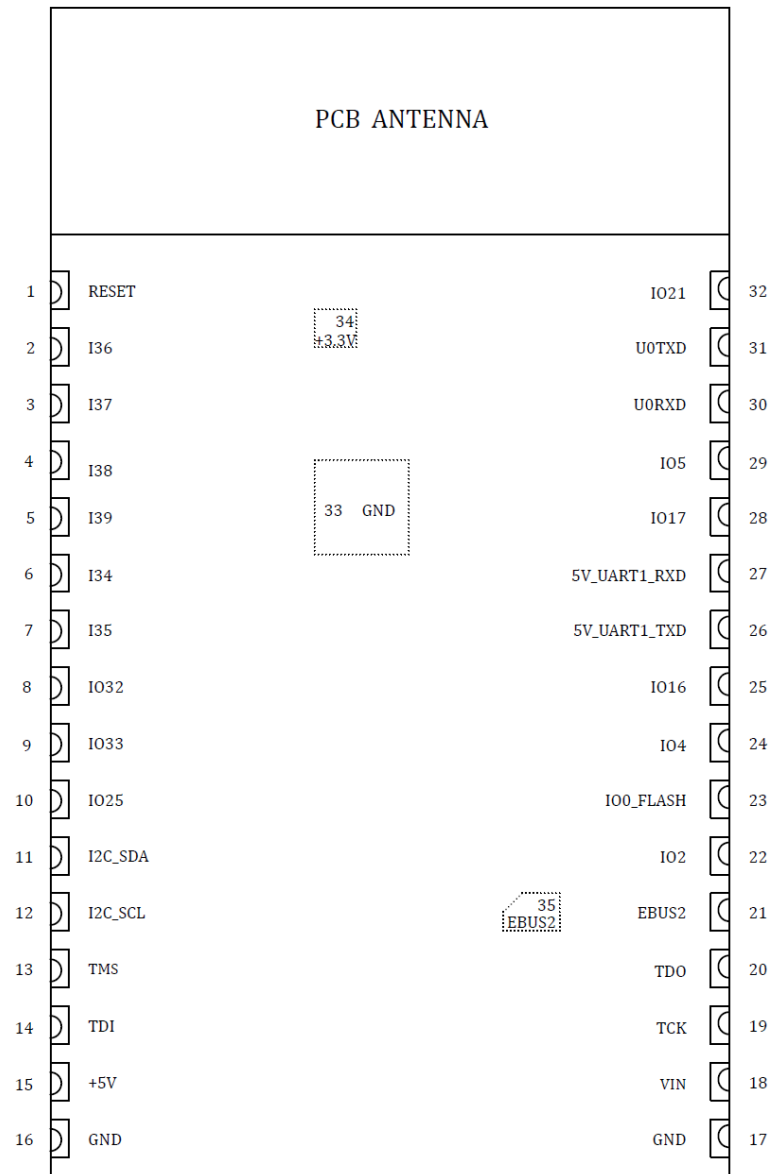| Categories	Items	Specifications
Wi-Fi	Protocols	802.11 b/g/n (802.11n up to 150 Mbps)
		A-MPDU and A-MSDU aggregation and 0.4 μs guard in- terval support
	Frequency range	2400 MHz ~ 2483.5 MHz
Bluetooth	Protocols	Bluetooth v4.2 BR/EDR and BLE specification
	Radio	NZIF receiver with –97 dBm sensitivity
		Class-1, class-2 and class-3 transmitter
		AFH
	Audio	CVSD and SBC
Hardware	Module interfaces	UART, I^{2}C, EBUS2, JTAG, GPIO
	On-chip sensor	Hall sensor
	Integrated crystal	40 MHz crystal
	Integrated SPI flash	8 MB
	Integrated DCDC Converter	3.3 V, 1.2 A
	Operating voltage/Power supply	12 V / 24 V
	Maximum current delivered by power supply	300 mA
	Recommended operating tem- perature range	–40 °C ~+ 85 °C
	Module Dimensions	(18.00±0.15) mm × (31.00±0.15) mm × (3.10±0.15) mm

ESP32-WATG-32D has 35 pins which are described in Table2.

1.2. Pin Description

PCB ANTENNA

1	RESET		IO21	32
2	I36	34 +3.3V	U0TXD	31
3	I37		U0RXD	30
4	I38		IO5	29
5	I39	33 GND	IO17	28
6	I34		5V_UART1_RXD	27
7	I35		5V_UART1_TXD	26
8	IO32		IO16	25
9	IO33		IO4	24
10	IO25		IO0_FLASH	23
11	I2C_SDA		IO2	22
12	I2C_SCL	35 EBUS2	EBUS2	21
13	TMS		TDO	20
14	TDI		TCK	19
15	+5V		VIN	18
16	GND		GND	17

Figure 1: Pin Layout

Table 2: Pin Definitions

Name	No.	Type	Function
RESET	1	I	Module enable signal(Internal pull-up by default). Active high.
I36	2	I	GPIO36, ADC1_CH0, RTC_GPIO0
I37	3	I	GPIO37, ADC1_CH1, RTC_GPIO1
I38	4	I	GPI38, ADC1_CH2, RTC_GPIO2

I39	5	I	GPIO39, ADC1_CH3, RTC_GPIO3
I34	6	I	GPIO34, ADC1_CH6, RTC_GPIO4
I35	7	I	GPIO35, ADC1_CH7, RTC_GPIO5
IO32	8	I/O	GPIO32, XTAL_32K_P (32.768 kHz crystal oscillator input), ADC1_CH4, TOUCH9, RTC_GPIO9
IO33	9	I/O	GPIO33, XTAL_32K_N (32.768 kHz crystal oscillator output), ADC1_CH5, TOUCH8, RTC_GPIO8
IO25	10	I/O	GPIO25, DAC_1, ADC2_CH8, RTC_GPIO6
I2C_SDA	11	I/O	GPIO26, I2C_SDA
I2C_SCL	12	I	GPIO27, I2C_SCL
TMS	13	I/O	GPIO14, MTMS
TDI	14	I/O	GPIO12, MTDI
+5V	15	PI	5 V power supply input
GND	16, 17	PI	Ground
VIN	18	I/O	12 V / 24 V power supply input
TCK	19	I/O	GPIO13, MTCK
TDO	20	I/O	GPIO15, MTDO
EBUS2	21, 35	I/O	GPIO19/GPIO22, EBUS2
IO2	22	I/O	GPIO2, ADC2_CH2, TOUCH2, RTC_GPIO12, HSPIWP, HS2_DATA0
IO0_FLASH	23	I/O	Download Boot: 0; SPI Boot: 1(Default).
IO4	24	I/O	GPIO4, ADC2_CH0, TOUCH0, RTC_GPIO10, HSPIHD, HS2_DATA1
IO16	25	I/O	GPIO16, HS1_DATA4
5V_UART1_TXD	26	O	GPIO23, 5V UART Data Transmit
5V_UART1_RXD	27	I	GPIO18, 5V UART Data Receive
IO17	28	-	GPIO17, HS1_DATA5
IO5	29	I/O	GPIO5, VSPICS0, HS1_DATA6
U0RXD	31	I/O	GPIO3, U0RXD
U0TXD	30	I/O	GPIO1, U0TXD
IO21	32	I/O	GPIO21, VSPIHD
GND	33	PI	EPAD, Ground
+3.3V	34	PO	3.3V Power supply output

2.　Hardware Preparation

2.1.　Hardware Preparation

- ESP32-WATG-32D module
- Espressif RF testing board (Carrier Board)
- One USB-to-UART dongle
- PC, Windows 7 recommended
- Micro-USB cable

2.2.　Hardware Connection

1. Solder ESP32-WATG-32D to the Carrier Board, as Figure 2 shows.
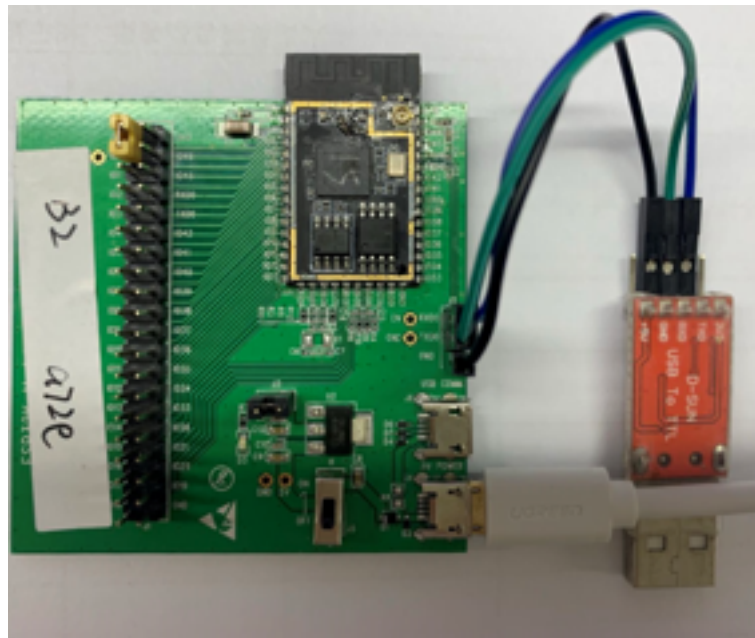

Figure 2: Testing Environment Setup(Needs Update)

2. Connect USB-to-UART dongle to the carrier board via TXD, RXD and GND.

3. Connect USB-to-UART dongle to the PC via the Micro-USB cable.

4. Connect the carrier board to 24 V adapter for power supply.

5. During download, short IO0 to GND via a jumper. Then, turn "ON" the board.

6. Download firmware into flash using the ESP32 DOWNLOAD TOOL.

7. After download, remove the jumper on IO0 and GND.

8. Power up the carrier board again. ESP32-WATG-32D 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-WATG-32D, please refer to ESP32-WATG-32D Datasheet.*

3. Getting Started with ESP32-WATG-32D

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 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/
esp32_win32_msys2_environment_and_toolchain-20181001.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/xtensa-esp32-elf-linux64-esp32-2019r1-8.2.0.tar.gz

32-bit Linux :
https://dl.espressif.com/dl/xtensa-esp32-elf-linux32-esp32-2019r1-8.2.0.tar.gz

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

64-bit Linux :

```
mkdir -p ~/esp
cd ~/esp
tar -xzf ~/Downloads/xtensa-esp32-elf-linux64-esp32-2019r1-8.2.0.tar.gz
```

32-bit Linux :

```
mkdir -p ~/esp
cd ~/esp
tar -xzf ~/Downloads/xtensa-esp32-elf-linux32-esp32-2019r1-8.2.0.tar.gz
```

2. The toolchain will be unzipped to **~/esp/xtensa-esp32-elf/** directory.
   Add the following to **~/.profile**:

```
export PATH="$HOME/esp/xtensa-esp32-elf/bin:$PATH"
```

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

```
alias get_esp32='export PATH="$HOME/esp/xtensa-esp32-elf/bin:$PATH"'
```

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

```
    printenv PATH


    $ printenv PATH
    /home/user-name/esp/xtensa-esp32-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

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://github.com/espressif/esp-idf/blob/master/docs/en/get-started/macos-setup.rst#id1

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

The toolchain will be unzipped into **~/esp/xtensa-esp32-elf/** path.

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

```
export PATH=$HOME/esp/xtensa-esp32-elf/bin:$PATH
```

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

```
alias get_esp32="export PATH=$HOME/esp/xtensa-esp32-elf/bin:$PATH"
```

Input `get_esp322` 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**. 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 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 PAT:

```
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-WATG-32D

This section provides guidance how to establish serial connection between ESP32-WATG-32D and PC.

4.1. Connect ESP32-WATG-32D to PC

Solder ESP32-WATG-32D module to the carrier board and connect carrier board to the PC using the USB-to-UART dongle. If device driver does not install automatically, identify USB to serial converter chip on your external USB-to-UART dongle, search for drivers in internet and install them.

Below are the links to drivers that can be used.

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 USB-to-UART dongle to the PC.

4.2. Check Port on Windows

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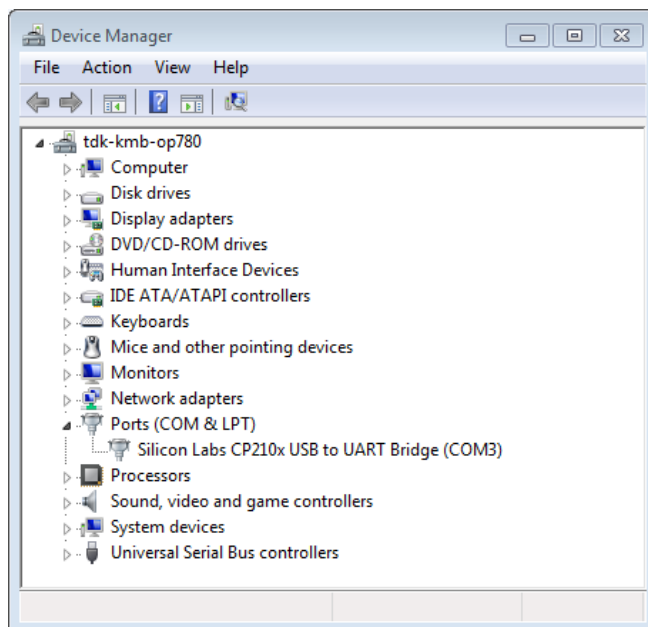

Figure 4-1. USB to UART bridge of USB-to-UART dongle in Windows Device Manager
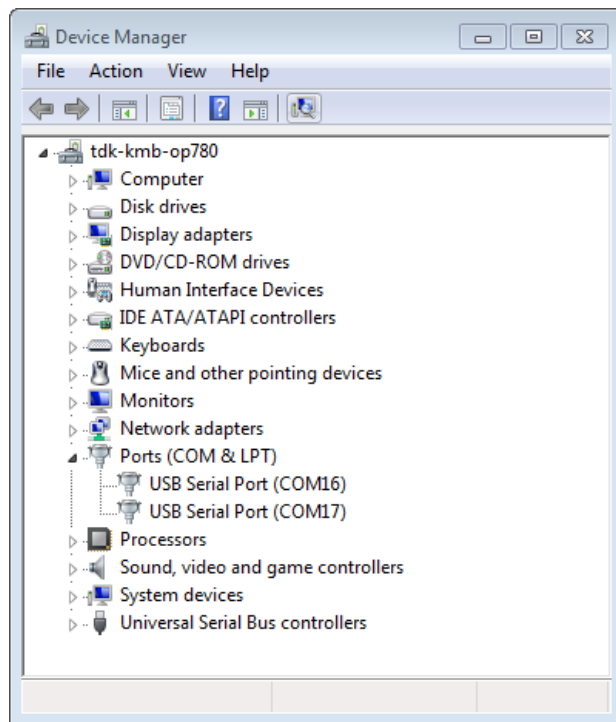
Figure 4-2. Two USB Serial Ports of USB-to-UART dongle in Windows Device Manager

4.3. Check Port on Linux and MacOS

To check the device name for the serial port of your USB-to-UART dongle, run this command two times, first with the 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.
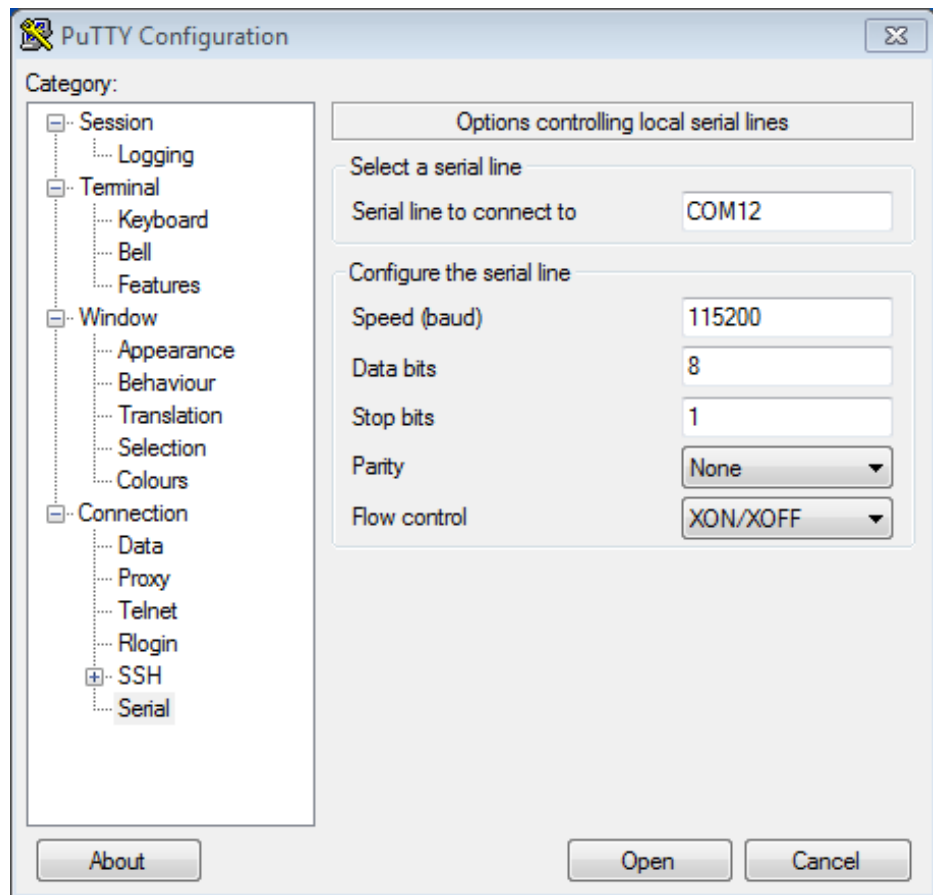

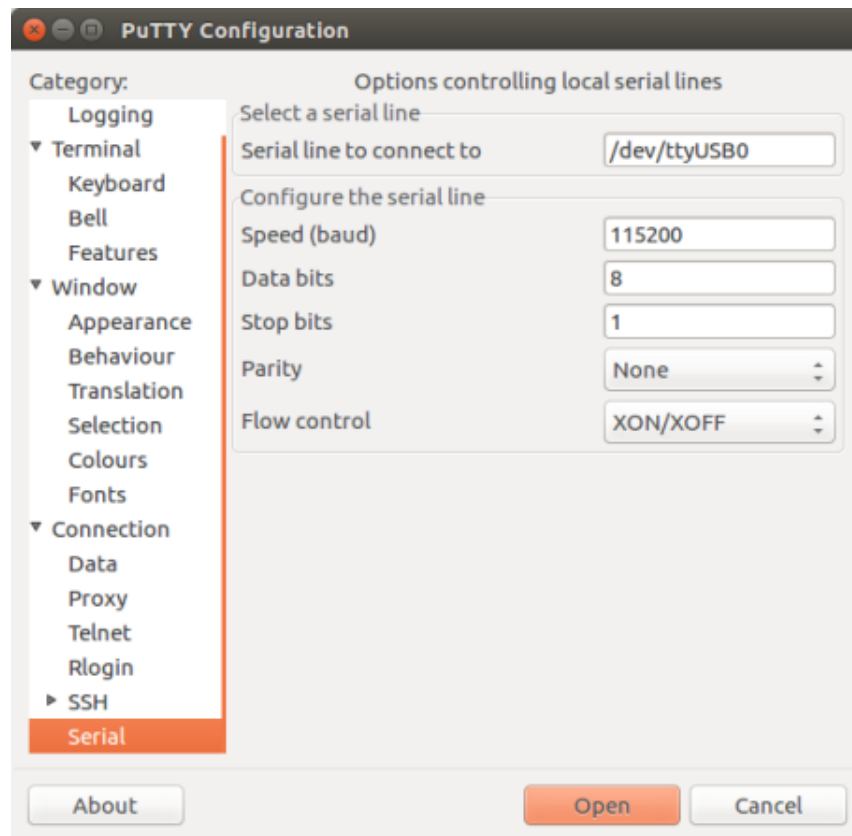Figure 4-3. Setting Serial Communication in PuTTY on Windows

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. The log contents will depend on application loaded to ESP32.

Notes:

- For some serial port wiring configurations, the serial RTS & DTR pins need to be disabled in the terminal program before the ESP32 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. 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=esp32 menuconfig
```

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

Windows

```
cd %userprofile%\esp\hello_world
idf.py -DIDF_TARGET=esp32 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 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 board by running:

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

Replace PORT with your ESP32 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*.