RWF01

OEM Installation Manual

Version 1.0

Copyright © 2019

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 RWF01. Through a simple example, this document illustrates how to use ESP-IDF (Espressif IoT Development Framework), including the menu based configuration wizard, compiling the ESP-IDF and firmware download to the ESP32 module.

Release Notes

Date	Version	Release notes
2019.05	V1.0	First release.

Table of Contents

1. Introduction to RWF01

1.1. RWF01

RWF01 is a powerful, generic Wi-Fi+BT+BLE MCU module that targets 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.

At the core of this module is the ESP32-S0WD chip. ESP32-S0WD is a member of the ESP32 family of chips, yet it features a single core and packs all the peripherals of its dual-core counterpart. Available in a 5x5 mm QFN, it offers great value for money, with its flawless performance when powering complex IoT applications.

The integration of Bluetooth, Bluetooth LE and Wi-Fi ensures that a wide range of applications can be targeted, and that the module is future proof: using Wi-Fi allows a large physical range and direct connection to the internet through a Wi-Fi router, while using Bluetooth allows the user to conveniently connect to the phone or broadcast low energy beacons for its detection. The sleep current of the ESP32 chip is less than 5 μA, making it suitable for battery powered and wearable electronics applications. ESP32 supports a data rate of up to 150 Mbps, and 20.5 dBm output power at the antenna to ensure the widest physical range. As such the chip does offer industry-leading specifications and the best performance for electronic integration, range, power consumption, and connectivity.

The operating system chosen for ESP32 is freeRTOS with LwIP; TLS 1.2 with hardware acceleration is built in as well. Secure (encrypted) over the air (OTA) upgrade is also supported, so that developers can continually upgrade their products even after their release.

1.2. 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 in Windows/Linux/MacOS based on ESP-IDF. It is recommended to use Linux distribution. *Lubuntu* 16.04 has been used as an example in this document for illustration purposes.

1.3. Preparation

To develop applications for RWF01 you need:

- PC loaded with either Windows, Linux or Mac operating system
- Toolchain to build the Application for ESP32
- ESP-IDF that contains API for ESP32 and scripts to operate the Toolchain
- A text editor to write programs (Projects) in C, e.g. Eclipse
- The ESP32 board itself and a USB cable to connect it to the PC

2. Get Started on RWF01

2.1. Standard Setup of Toolchain for Linux

The quickest way to start development with ESP32 is by installing a prebuilt toolchain. Pick up your OS below and follow provided instructions.

2.1.1. Install Prerequisites

To compile with ESP-IDF you need to get the following packages:

- CentOS 7:

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

- Ubuntu and Debian:

```
sudo apt-get install git wget make libncurses-dev flex bison gperf python python-serial
```

- Arch:

```
sudo pacman -S --needed gcc git make ncurses flex bison gperf python2-pyserial
```

2.1.2. Toolchain Setup

ESP32 toolchain for Linux is available for download from Espressif website:

- for 64-bit Linux:

https://dl.espressif.com/dl/xtensa-esp32-elf-linux64-1.22.0-61-gab8375a-5.2.0.tar.gz

- for 32-bit Linux:

https://dl.espressif.com/dl/xtensa-esp32-elf-linux32-1.22.0-61-gab8375a-5.2.0.tar.gz Download this file, then extract it in *~/esp* directory

```
mkdir -p ~/esp
cd ~/esp
tar -xzf ~/Downloads/xtensa-esp32-elf-linux64-1.22.0-61-gab8375a-5.2.0.tar.gz
```

The toolchain will be extracted into *~/esp/xtensa-esp32-elf/* directory.

To use it, you will need to update your ``*PATH*`` environment variable in *~/.bash profile* file. To make *xtensa-esp32-elf* available for all terminal sessions, add the following line to your *~/.bash profile* file:

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

Alternatively, you may create an alias for the above command. This way you can get the toolchain only when you need it. To do this, add different line to your *~/.bash_profile* file:

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

Then when you need the toolchain you can type *get_esp32* on the command line and the toolchain will be added to your *PATH*.

2.2. 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:

```
cd ~/esp
git clone --recursive https://github.com/espressif/esp-idf.git
```

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

Note:

- *While cloning submodules on Windows platform, the* git clone *command may print some output starting* ': not a valid identifier.... *This is a known issue but the git clone still succeeds without any problems.*
- *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
```

2.3. Set up Path to ESP-IDF

The toolchain programs access ESP-IDF using *IDF PATH* environment variable. This variable should be set up on your PC, otherwise projects will not build. Setting may be done manually, each time PC is restarted. Another option is to set up it permanently by defining *IDF PATH* in user profile. To do so, follow instructions specific to *:ref:`Windows <add-idf path-to-profile-windows>`*, *:ref:`Linux and MacOS <add-idf path-to-profile-linux-macos>`* in section *:doc:`add-idf path-to-profile`*.

3. Start a Project

Now you are ready to prepare your application for ESP32. To start off quickly, we will use *:example:`get-started/hello world` project* from *:idf:`examples`* directory in IDF.

Copy *:example:`get-started/hello world`* to *~/esp* directory:

```
cd ~/esp
cp -r $IDF PATH/examples/get-started/hello world .
```

You can also find a range of example projects under the *:idf:`examples`* directory in ESP-IDF. These example project directories can be copied in the same way as presented above, to begin your own projects.

> ***Note:***
> *The ESP-IDF build system does not support spaces in paths to ESP-IDF or to projects.*

4. Connect

You are almost there. To be able to proceed further, connect ESP32 board to PC, check under what serial port the board is visible and verify if serial communication works. If you are not sure how to do it, check instructions in section *:doc:`establish-serial-connection`*. Note the port number, as it will be required in the next step.

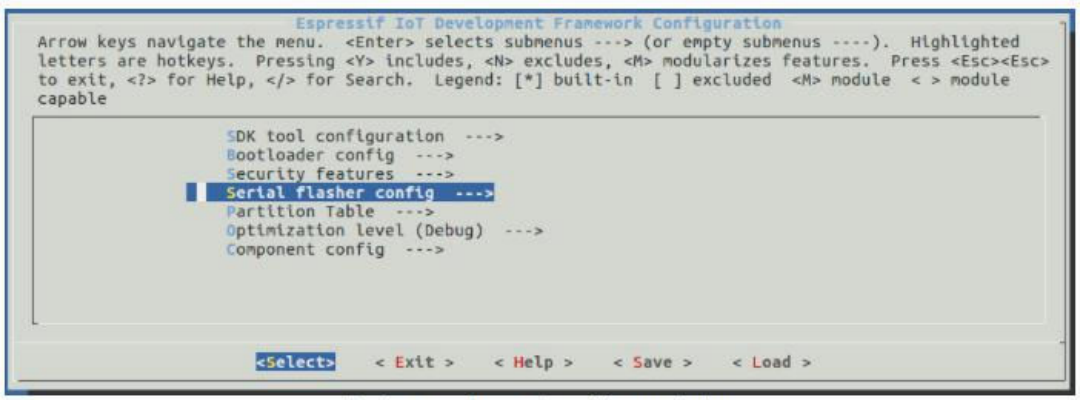
!

5. Configure

Being in terminal window, go to directory of *hello world* application by typing cd ~/esp/hello world. Then start project configuration utility *menuconfig*:

```
cd ~/esp/hello world
make menuconfig
```

If previous steps have been done correctly, the following menu will be displayed:



! *Project configuration - Home window*

In the menu, navigate to *Serial flasher config* > *Default serial port* to configure the serial port, where project will be loaded to. Confirm selection by pressing enter, save configuration by selecting < *Save* > and then exit application by selecting < *Exit* >. Here are couple of tips on navigation and use of *menuconfig*:

- Use up & down arrow keys to navigate the menu.
- Use Enter key to go into a submenu, Escape key to go out or to exit.
- Type *?* to see a help screen. Enter key exits the help screen.
- Use Space key, or *Y* and *N* keys to enable (Yes) and disable (No) configuration items with checkboxes "*[*]*".
- Pressing *?* while highlighting a configuration item displays help about that item.
- Type */* to search the configuration items.

Notes:

- *On Windows, serial ports have names like COM1. On MacOS, they start with /dev/cu.. On Linux, they start with /dev/tty. (See :doc:`establish-serial-connection` for full details.)*
- *If you are Arch Linux user, navigate to SDK tool configuration and change the name of Python 2 interpreter from python to python2.*
- *Most ESP32 development boards have a 40 MHz crystal installed. However, some boards use a 26 MHz crystal. If your board uses a 26MHz crystal, or you get garbage output from serial port after code upload, adjust the :ref:`CONFIG_ESP32_XTAL_FREQ_SEL` option in menuconfig.*

6. Build and Flash

6.1. Build and Flash

Now you can build and flash the application. Run:

```
make flash
```

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.

```
esptool.py v2.0-beta2

Flashing binaries to serial port /dev/ttyUSB0 (app at offset 0x10000)... esptool.py
v2.0-beta2

Connecting.....

Uploading stub...

Running stub...

Stub running...

Changing baud rate to 921600

Changed.

Attaching SPI flash...

Configuring flash size...

Auto-detected Flash size: 4MB

Flash params set to 0x0220

Compressed 11616 bytes to 6695...

Wrote 11616 bytes (6695 compressed) at 0x00001000 in 0.1 seconds (effective 920.5 kbit/s)... Hash of
data verified.

Compressed 408096 bytes to 171625...

Wrote 408096 bytes (171625 compressed) at 0x00010000 in 3.9 seconds (effective 847.3 kbit/ s)...


Hash of data verified.

Compressed 3072 bytes to 82...

Wrote 3072 bytes (82 compressed) at 0x00008000 in 0.0 seconds (effective 8297.4 kbit/s)... Hash of
data verified.


Leaving...

Hard resetting...
```

If there are no issues, at the end of build process, you should see messages describing progress of loading process. Finally, the end module will be reset and “hello_world” application will start.
If you'd like to use the Eclipse IDE instead of running make, check out the *:doc:`Eclipse guide <eclipse-setup>`*.

6.2. Monitor

To see if “hello world” application is indeed running, type make monitor. This command is launching *:doc:`IDF Monitor <idf-monitor>`* application:

```
$ make monitor
MONITOR
---    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
...
```

Several lines below, after start up and diagnostic log, 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 monitor use shortcut *Ctrl+]*. To execute make flash and make monitor in one shoot type make flash monitor. Check section *:doc:`IDF Monitor <idf-monitor>`* for handy shortcuts and more details on using this application.
That's all what you need to get started with ESP32!
Now you are ready to try some other *:idf:`examples`*, or go right to developing your own applications.

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.

FCC Certification Requirements.

According to the definition of mobile and fixed device is described in Part 2.1091(b), this device is a mobile device.

And the following conditions must be met:

1. This Modular Approval is limited to OEM installation for mobile and fixed applications only. The antenna installation and operating configurations of this transmitter, including any applicable source-based time- averaging duty factor, antenna gain and cable loss must satisfy MPE categorical Exclusion Requirements of 2.1091.

2. The EUT is a mobile device; maintain at least a 20 cm separation between the EUT and the user’s body and must not transmit simultaneously with any other antenna or transmitter.

3.A label with the following statements must be attached to the host end product: This device contains FCC ID: A7M-RWF01.

4.To comply with FCC regulations limiting both maximum RF output power and human exposure to RF radiation, maximum antenna gain (including cable loss) must not exceed: ☒ WiFi: <3dBi

5. This module must not transmit simultaneously with any other antenna or transmitter

6. The host end product must include a user manual that clearly defines operating requirements and conditions that must be observed to ensure compliance with current FCC RF exposure guidelines.

For portable devices, in addition to the conditions 3 through 6 described above, a separate approval is required to satisfy the SAR requirements of FCC Part 2.1093 If the device is used for other equipment that separate approval is required for all other operating configurations, including portable configurations with respect to 2.1093 and different antenna configurations.

For this device, OEM integrators must be provided with labeling instructions of finished products. Please refer to KDB784748 D01 v07, section 8. Page 6/7 last two paragraphs:

A certified modular has the option to use a permanently affixed label, or an electronic label. For a permanently affixed label, the module must be labeled with an FCC ID - Section 2.926 (see 2.2 Certification (labeling requirements) above). The OEM manual must provide clear instructions explaining to the OEM the labeling requirements, options and OEM user manual instructions that are required (see next paragraph).

For a host using a certified modular with a standard fixed label, if (1) the module’s FCC ID is not visible when installed in the host, or (2) if the host is marketed so that end users do not have straightforward commonly used methods for access to remove the module so that the FCC ID of the module is visible; then an additional permanent label referring to the enclosed module:“Contains Transmitter Module FCC ID: A7M-RWF01” or “Contains FCC ID: A7M-RWF01” must be used. The host OEM user manual must also contain clear instructions on how end users can find and/or access the module and the FCC ID.

The final host / module combination may also need to be evaluated against the FCC Part 15B criteria for unintentional radiators in order to be properly authorized for operation as a Part 15 digital device.

The user’s manual or instruction manual for an intentional or unintentional radiator shall caution the user that changes or modifications not expressly approved by the party responsible for compliance could void the user's authority to operate the equipment. In cases where the manual is provided only in a form other than paper, such as on a computer disk or over the Internet, the information required by this section may be included in the manual in that alternative form, provided the user can reasonably be expected to have the capability to access information in that form.

This device complies with part 15 of the FCC Rules. Operation is subject to the following two conditions: (1) This device may not cause harmful interference, and (2) this device must accept any interference received, including interference that may cause undesired operation.

Changes or modifications not expressly approved by the manufacturer could void the user’s authority to operate the equipment.

To ensure compliance with all non-transmitter functions the host manufacturer is responsible for ensuring compliance with the module(s) installed and fully operational. For example, if a host was previously authorized as an unintentional radiator under the Declaration of Conformity procedure without a transmitter certified module and a module is added, the host manufacturer is responsible for ensuring that the after the module is installed and operational the host continues to be compliant with the Part 15B unintentional radiator requirements.