ESP-IDF
Getting Started Guide

Version 1.1

Copyright © 2016

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 Espressif ESP32. 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 ESP32 boards.

The document is structured as follows.

Chapter	Title	Content
Chapter 1	Introduction	Introduction to the ESP32 and ESP-IDF.
Chapter 2	Getting Started	Introduction to the overall procedure of getting started with ESP-IDF.
Appendix	Learning Resources	Provision of ESP32-related must-read documents and must-have resources.

Release Notes

Date	Version	Release notes
2016.10	V1.0	Initial release.
2016.12	V1.1	Revised based on ESP-IDF 1.0.

Table of Contents

1. Introduction 1
   - 1.1. ESP32 1
   - 1.2. ESP-IDF 1
   - 1.3. Preparing the Hardware 1

2. Getting Started 2
   - 2.1. Installing Required Packages for ESP-IDF 2
   - 2.2. Cross-compiler Toolchain 2
     - 2.2.1. Downloading the Toolchain 2
     - 2.2.2. Installing the Toolchain 2
   - 2.3. ESP-IDF 3
     - 2.3.1. Downloading ESP-IDF 3
     - 2.3.2. Directory Structure 3
   - 2.4. hello_world Example 4
     - 2.4.1. Configuration 5
     - 2.4.2. Compilation 5
     - 2.4.3. Downloading the Binaries to Flash 8

A. Appendix - Learning Resources 9
   - A.1. Must-Read Documents 9
   - A.2. Must-Have Resources 10

1. Introduction

1.1. ESP32

ESP32 integrates Wi-Fi (2.4 GHz band) and Bluetooth 4.2 solutions on a single chip, along with dual high performance cores and many other versatile peripherals. Powered by 40 nm technology, ESP32 provides a robust, highly integrated platform to meet the continuous demands for efficient power usage, compact design, security, high performance, and reliability.

Espressif provides the basic hardware and software resources that empowers application developers to build their ideas around the ESP32 series hardware. The software development framework provided by Espressif is intended for rapidly developing Internet-of-Things (IoT) applications, with Wi-Fi, Bluetooth, flexible power management and other advanced system features.

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. Preparing the Hardware

- An ESP32 development board or ESP-WROOM-32 (a module built around the ESP32)
- A USB to TTL serial cable or a Micro-USB cable.

2. Getting Started

2.1. Installing Required Packages for ESP-IDF

Some packages are required for using ESP-IDF and related resources. Run the following command in the terminal on ***Lubuntu*** to download and install these pre-requisite components.

```
sudo apt-get install git make gcc libncurses5-dev flex bison gperf python-serial
```

If some pre-requisite packages for the above components are missing, you may be prompted to install them first. Install those components and then make sure that the above components are installed before you proceed. Please note that you may require administrator privileges to make these changes to your system.

2.2. Cross-compiler Toolchain

2.2.1. Downloading the Toolchain

Espressif provides pre-built cross-complier toolchains for the following operating systems. Users can download installation files at:

- Linux (x64): *https://dl.espressif.com/dl/xtensa-esp32-elf-linux64-1.22.0-59.tar.gz*
- Linux (x32): *https://dl.espressif.com/dl/xtensa-esp32-elf-linux32-1.22.0-59.tar.gz*
- Linux (armhf): To be provided.
- MacOS: *https://dl.espressif.com/dl/xtensa-esp32-elf-osx-1.22.0-59.tar.gz*
- Windows: *https://dl.espressif.com/dl/xtensa-esp32-elf-win32-1.22.0-59.zip*

> 📖 ***Note:***
>
> *This toolchain is built based on the crosstool-NG. Instructions on building the toolchain are not specified in this document. For Windows OS, please refer to: http://esp-idf.readthedocs.io/en/latest/windows-setup.html.*

2.2.2. Installing the Toolchain

1. Execute the following command in the command terminal on ***Lubuntu***.

```
wget https://dl.espressif.com/dl/xtensa-esp32-elf-linux64-1.22.0-59.tar.gz

sudo tar zxvf xtensa-esp32-elf-linux64-1.22.0-59.tar.gz -C /opt
```

2. Open ***~/.bashrc*** file with the following command.

```
vi ~/.bashrc
```

3. Add the following line to the end of the file to add the ESP32 toolchain path to the PATH environment variable. If you chose not to add this line, you will have to execute this line in the command terminal every time you boot into ***Lubuntu***.

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

4. Open a new terminal and input the following command to find out the installed toolchain version.

```
xtensa-esp32-elf-gcc -v
```

5. The screenshot below demonstrates the output when the toolchain has been successfully installed. If you get a “command not found” or similar error, it indicates that your toolchain was not successfully installed or configured.

2.3. ESP-IDF

2.3.1. Downloading ESP-IDF

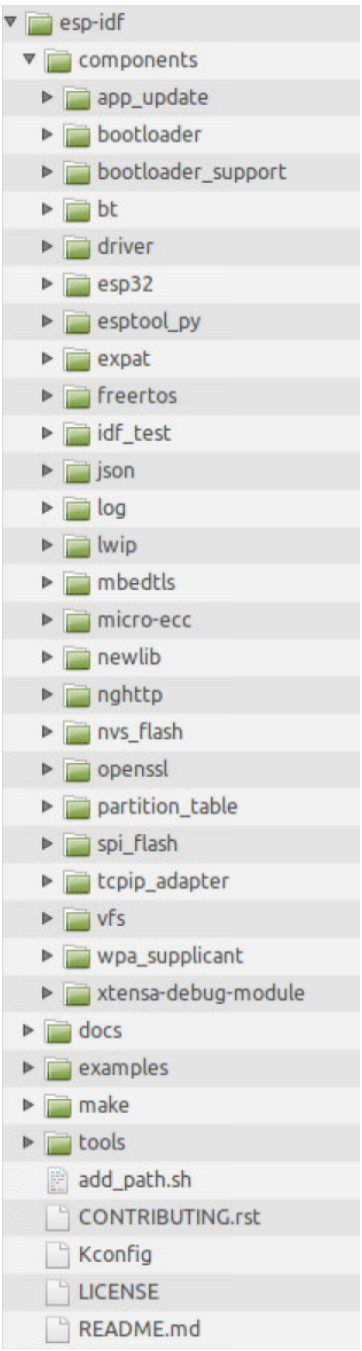
Please download ESP-IDF from: *https://github.com/espressif/esp-idf.git*.

Run the following command in the terminal.

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

2.3.2. Directory Structure

The following figure shows the directory structure of ESP-IDF, including ***components***, ***examples***, ***make***, ***tools*** and ***docs***. ***components*** folder contains the core components of ESP-IDF; ***examples*** folder contains the program examples of ESP-IDF; ***make*** folder contains makefiles for ESP-IDF; ***tools*** folder is the toolkit; ***docs*** contains ESP-IDF-relevant documentation.



> 📖 ***Note:***
> *For more information, please see **README.md**.*

2.4. *hello_world* Example

esp-idf/examples/01_hello_world directory contains sample code that can be run on the ESP32. Using this example project, the following sections illustrate the overall compilation and deployment process, including configuration, compilation, firmware download and

execution. If the procedure is successfully completed, ESP32 will connect to the specified router. Users can copy the ***esp-idf/examples/01_hello_world*** directory to ***~/workspace***.

For better project directory management, it is recommended that users' project directory is stored in an individual directory that is separate from ***esp-idf***. This will ensure that ***esp-idf*** directory contents or structure are not altered by mistake.

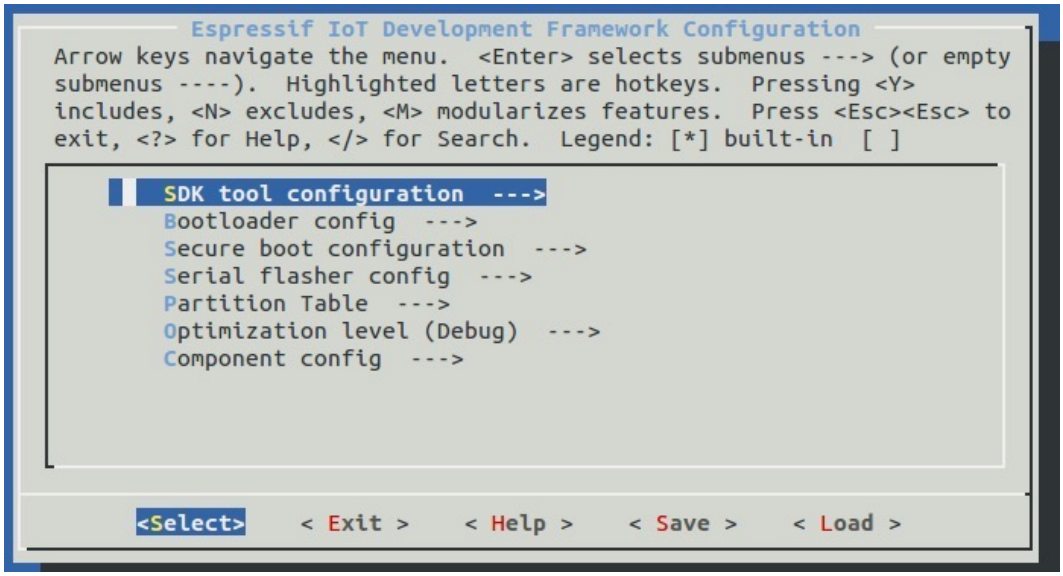
2.4.1. Configuration

Users can configure ESP-IDF through `menuconfig` utility.

Run the following command in the terminal:

```
cp ~/workspace/esp-idf/examples/01_hello_world ~/workspace -r
cd ~/workspace/01_hello_world
export IDF_PATH=~/workspace/esp-idf
make menuconfig
```

Then, the following interface is displayed:



📖 ***Notes:***

- *For specifics of each configuration option, Please see* ***Help****.*
- ***IDF_PATH*** *needs to be exported to the project directory so that the IDF path would be known.*

2.4.2. Compilation

1. Compiling the help command

Run the following command:

```
make help
```

If successful, the following information will be printed. Users can choose which binary to compile.

```
Welcome to Espressif IDF build system. Some useful make targets:

make menuconfig - Configure IDF project
make defconfig - Set defaults for all new configuration options

make all - Build app, bootloader, partition table
make flash - Flash all components to a fresh chip
make clean - Remove all build output

make app - Build just the app
make app-flash - Flash just the app
make app-clean - Clean just the app

See also 'make bootloader', 'make bootloader-flash', 'make bootloader-clean',
'make partition_table', etc, etc.
lvxinyue@ubuntu:~/workspace/01_hello_world$
```

2. Compiling all binaries

Run the following command:

```
make
```

OR,

```
make all
```

If successful, the following information will be printed:

```
To flash all build output, run 'make flash' or:
python /home/lvxinyue/workspace/esp-idf/components/esptool_py/esptool/esptool.py
 --chip esp32 --port /dev/ttyUSB0 --baud 115200 write_flash -z --flash_mode dio
--flash_freq 40m --flash_size 2MB 0x1000 /home/lvxinyue/workspace/01_hello_world
/build/bootloader/bootloader.bin 0x10000 /home/lvxinyue/workspace/01_hello_world
/build/hello-world.bin 0x8000 /home/lvxinyue/workspace/01_hello_world/build/part
itions_singleapp.bin
lvxinyue@ubuntu:~/workspace/01_hello_world$
```

bootloader, ***partition*** and ***app*** binaries are compiled by default. Users can see the `flash command` to download the binaries.

3. Compiling bootloader

Run the following command:

```
make bootloader
```

If successful, the following information will be printed:

```
Bootloader built. Default flash command is:
python /home/lvxinyue/workspace/esp-idf/components/esptool_py/esptool/esptool.py
 --chip esp32 --port /dev/ttyUSB0 --baud 115200 write_flash -z --flash_mode dio
--flash_freq 40m --flash_size 2MB 0x1000 /home/lvxinyue/workspace/01_hello_world
/build/bootloader/bootloader.bin
lvxinyue@ubuntu:~/workspace/01_hello_world$
```

Users can see the "`Default flash command`" to download the binary.

4. Compiling the user program

Run the following command:

```
make app
```

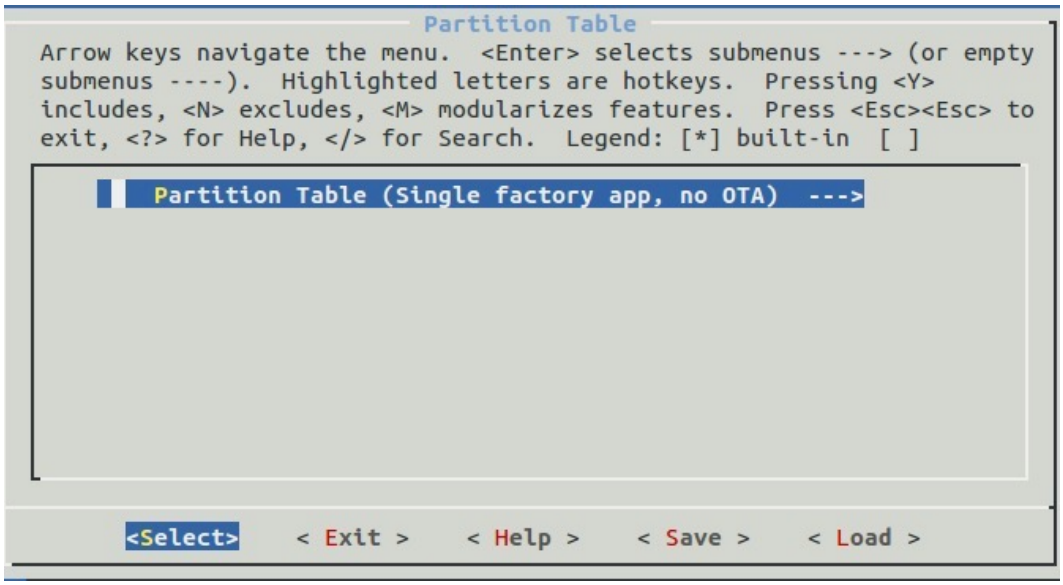
If successful, the following information will be printed:

```
App built. Default flash app command is:
python /home/lvxinyue/workspace/esp-idf/components/esptool_py/esptool/esptool.py
 --chip esp32 --port /dev/ttyUSB0 --baud 115200 write_flash -z --flash_mode dio
--flash_freq 40m --flash_size 2MB 0x10000 /home/lvxinyue/workspace/01_hello_worl
d/build/hello-world.bin
lvxinyue@ubuntu:~/workspace/01_hello_world$
```

Users can see the `flash command` to download the binary.

5. Compiling the partition table

For better SPI flash management, ESP-IDF introduces partition table. For details on the configuration of the partition table, please see ***partition-tables.rst*** under the ***docs*** directory. Users can select the partition table through `make menuconfig`. If not, ***Single factory app, no OTA*** Partition Table will be used by default. The Partition Table is shown below:



Run the following command:

```
make partition_table
```

If successful, the following information will be printed:

```
lvxinyue@ubuntu:~/workspace/01_hello_world$ make partition_table
Partition table binary generated. Contents:
*******************************************************************************
# Espressif ESP32 Partition Table
# Name, Type, SubType, Offset, Size
nvs,data,nvs,0x9000,24K
phy_init,data,phy,0xf000,4K
factory,app,factory,0x10000,1M
*******************************************************************************
Partition flashing command:
python /home/lvxinyue/workspace/esp-idf/components/esptool_py/esptool/esptool.py
 --chip esp32 --port /dev/ttyUSB0 --baud 115200 write_flash 0x8000 /home/lvxinyu
e/workspace/01_hello_world/build/partitions_singleapp.bin
lvxinyue@ubuntu:~/workspace/01_hello_world$
```

Users can see the `flash command` to download the binary.

2.4.3. Downloading the Binaries to Flash

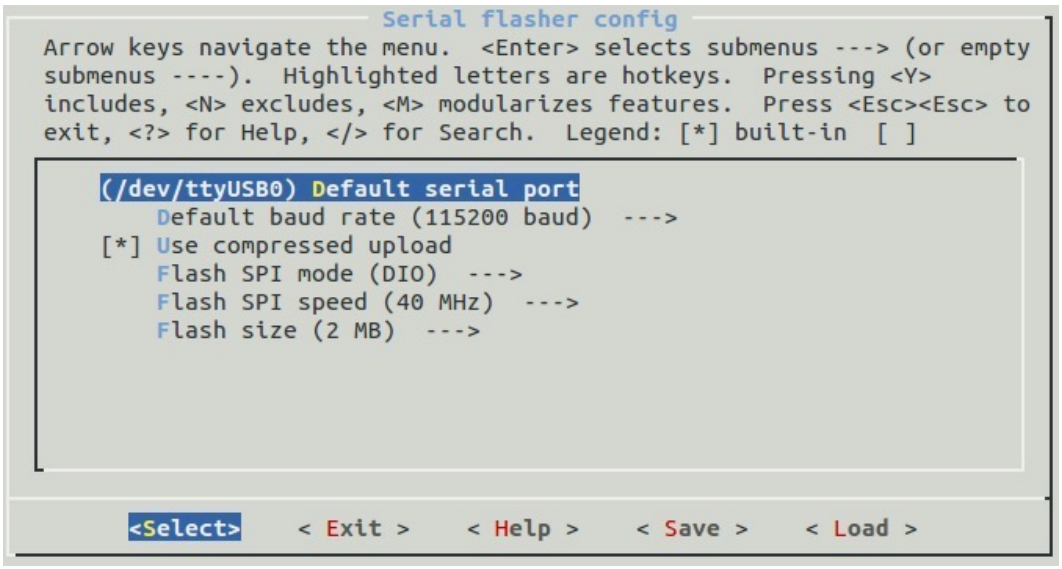
The USB device should be authorized to read and write in ***Lubuntu*** with the following command:

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

Make sure you re-login to enable a read-and-write permission for the USB device.

The binaries can be downloaded according to the `flash command` mentioned in ***Section 2.4.2***, or, with specific commands specified as follows.

The serial port parameters can be configured when executing `make menuconfig`. In ***Serial flasher config*** interface as shown below, users can configure the serial port number and baud rate, as well as decide whether to use compressed upload or not.



1. Download bootloader to flash.

This secondary boot loader is located in flash at 0x1000, and is responsible for booting the user program. The boot loader only needs to be downloaded once, unless there are updates to it.

Run the following command:

```
make bootloader-flash
```

2. Download user program to flash. Run the following command:

```
make app flash
```

3. Download partition table to flash. Run the following command:

```
make partition_table-flash
```

4. Download all binaries to flash. Run the following command:

```
make flash
```

A. Appendix - Learning Resources

A.1. Must-Read Documents

- *ESP32 Datasheet*

  Description: This document presents the detailed specifications for the ESP32, including an overview of the features, protocols, electrical characteristics, design parameters and application specific information. It also provides detailed information on the pinout and dimensions of the chip package. The major functional modules integrated on the ESP32 are also described. This document summarizes all design parameters required to develop hardware based on the ESP32.

- *ESP32 Pin List*

  Description: This document lists the type and functions of each pin on ESP32. Please consult this guide for effective use and programming of the ESP32 pins.

- *ESP32 Pinout*

  Description: This document provides the pinout of ESP32, can be used for quick reference.

- *ESP32 Technical Reference Manual*

  Description: This manual targets application developers. The manual provides detailed and complete information on how to use the ESP32 memory, peripherals and other integrated hardware resources. Please refer to the ESP32 datasheet for information on related electrical characteristics.

- *ESP32 Hardware Design Guidelines*

  Description: The guidelines outline recommended design practices when developing standalone or add-on systems based on the ESP32 series of products, including ESP32, the ESP-WROOM-32 module, and ESP32-DevKitC — the development board. This document is highly recommended for hardware development and system design engineering.

- *ESP-IDF Programming Guide*

  Description: This Github webpage provides resources for getting started with ESP32 in general, using Espressif ESP-IDF. Basic toolchain setup procedure, API descriptions and listing, examples and some advanced information is included. This guide is useful for application developers looking for information on a specific section or peripheral APIs.

- *ESP-IDF API Reference*

Description: This Github webpage lists and describes system APIs, TCP/UDP APIs, mesh APIs, application specific APIs, related definitions and data structures, as well as APIs for peripheral interfacing.

A.2. Must-Have Resources

- *ESP-IDF*

  Description: This website page provides the latest version of ESP-IDF and updated information.

- *ESP32 Tools*

  Description: This website page provides links to the ESP32 flash download tools and ESP32 certification and test tools.

- *ESP32 BBS*

  Description: Link to the official ESP32 forum. Have a question? Need to discuss something? Drop a line!

- *ESP32 Resouces*

  Description: This website page links in software resources and latest documentation released by Espressif. Please check back often to stay updated with latest releases.

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 © 2016 Espressif Inc. All rights reserved.