ESP-IDF
Getting Started Guide

Version 1.0

About This Guide

This document is intended to help users set up a basic software and hardware development environment based on ESP32. Through a simple example, this document illustrates how to use ESP-IDF (Espressif IoT Development Framework), including the configuration, ESP-IDF compilation and firmware download.

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.

Release Notes

Date	Version	Release notes
2016.10	V1.0	Initial release.

Table of Contents

1. Introduction

1.1. ESP32

ESP32 provides a Wi-Fi plus Bluetooth 4.2 combo solution in the 2.4 GHz band as a single chip solution. Powered by 40 nm technology, ESP32 delivers a highly integrated Wi-Fi SoC solution to meet the continuous demands for efficient power usage, compact design, security, high performance, and reliability.

ESP32 series hardware and software is provided by Espressif to develop and program Wi-Fi, Bluetooth and wearable electronics applications and products in the Internet of Things (IoT) industry.

1.2. ESP-IDF

The Espressif IoT Development Framework (ESP-IDF for short) is a framework for developing ESP32 applications. Users can develop applications in Windows/Linux/MacOS based on ESP-IDF. It is recommended to use Linux distribution. We use ***Lubuntu*** 16.04 as an example to illustrate instructions in this document.

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

Run the following command in the terminal in ***Lubuntu***.

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

Install other dependent software packets.

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): to be provided;
- Linux (armhf): to be provided;
- MacOS: *https://dl.espressif.com/dl/xtensa-esp32-elf-osx-1.22.0-59.tar.gz*;
- Windows: to be provided.

> 📖 ***Note:***
>
> *This toolchain is built based on the crosstool-NG. Instructions on building the toolchain are not specified in this document.*

2.2.2. Installing the Toolchain

1. Run the following command in the terminal in ***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:

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

4. Open a new terminal and run the following command.

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

5. The screenshot below demonstrates the output when the toolchain has been successfully installed:

```
dev@espressif:~$ xtensa-esp32-elf-gcc -v
Using built-in specs.
COLLECT_GCC=xtensa-esp32-elf-gcc
COLLECT_LTO_WRAPPER=/opt/xtensa-esp32-elf/bin/../libexec/gcc/xtensa-esp32-elf/4.
8.5/lto-wrapper
Target: xtensa-esp32-elf
Configured with: /home/ivan/e/crosstool-NG/.build/src/gcc-4.8.5/configure --buil
d=x86_64-build_pc-linux-gnu --host=x86_64-build_pc-linux-gnu --target=xtensa-esp
32-elf --prefix=/home/ivan/e/crosstool-NG/builds/xtensa-esp32-elf --with-local-p
refix=/home/ivan/e/crosstool-NG/builds/xtensa-esp32-elf/xtensa-esp32-elf/sysroot
 --with-sysroot=/home/ivan/e/crosstool-NG/builds/xtensa-esp32-elf/xtensa-esp32-e
lf/sysroot --with-newlib --enable-threads=no --disable-shared --with-pkgversion=
'crosstool-NG 8d95cad' --disable-__cxa_atexit --enable-cxx-flags='-fno-exception
s -fno-rtti' --with-gmp=/home/ivan/e/crosstool-NG/.build/xtensa-esp32-elf/buildt
ools --with-mpfr=/home/ivan/e/crosstool-NG/.build/xtensa-esp32-elf/buildtools --
with-mpc=/home/ivan/e/crosstool-NG/.build/xtensa-esp32-elf/buildtools --with-isl
=/home/ivan/e/crosstool-NG/.build/xtensa-esp32-elf/buildtools --with-cloog=/home
/ivan/e/crosstool-NG/.build/xtensa-esp32-elf/buildtools --with-libelf=/home/ivan
/e/crosstool-NG/.build/xtensa-esp32-elf/buildtools --enable-lto --enable-target-
optspace --without-long-double-128 --disable-libgomp --disable-libmudflap --disa
ble-libssp --disable-libquadmath --disable-libquadmath-support --disable-nls --d
isable-multilib --enable-languages=c,c++
Thread model: single
gcc version 4.8.5 (crosstool-NG 8d95cad)
dev@espressif:~$
```

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 in ***Lubuntu***.

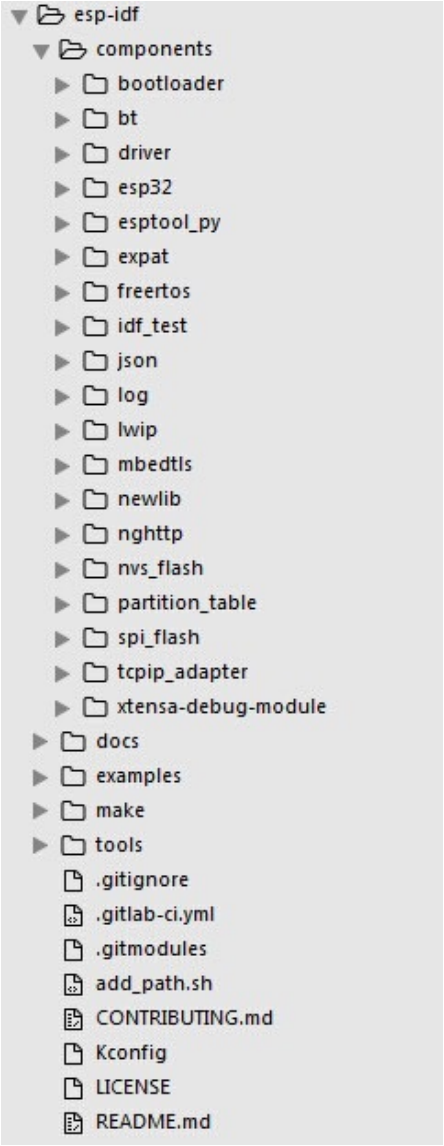
```
mkdir ~/workspace

cd ~/workspace

git clone 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** under the directory.*

2.4. *hello_world* Example

esp-idf/examples/01_hello_world directory contains the sample code to illustrate the overall procedure, including configuration, compilation, firmware download and running. 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***.

2.4.1. Configuration

Users can configure ESP-IDF through ***menuconfig***.

1. Run the following command in the terminal in ***Lubuntu***:

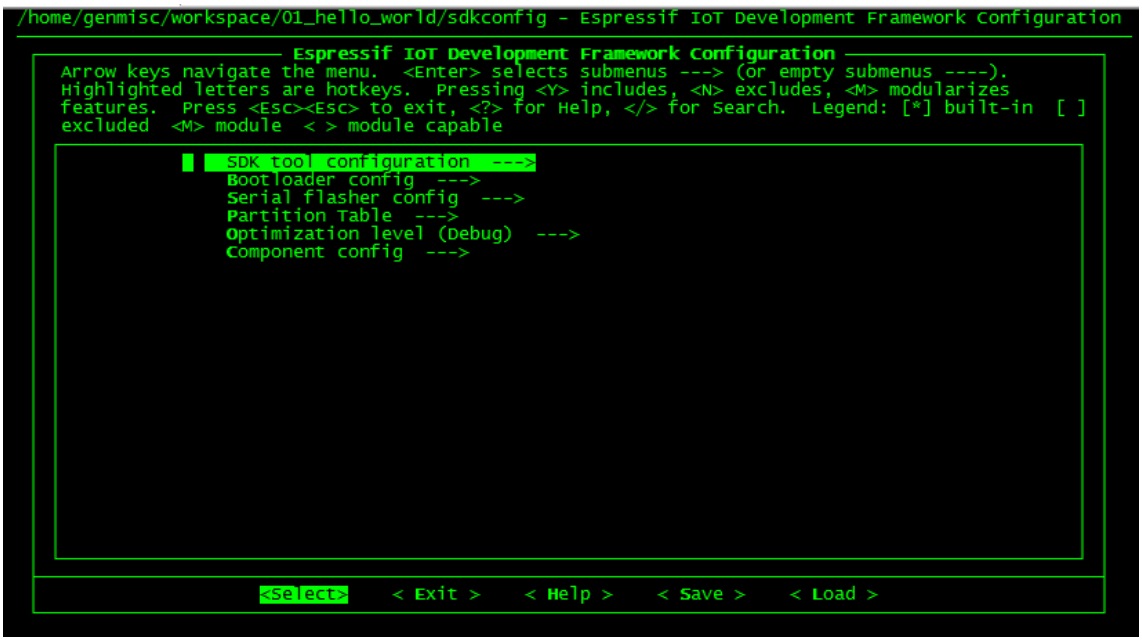
```
cp ~/workspace/esp-idf/examples/01_hello_world ~/workspace -r

cd ~/workspace/01_hello_world

export IDF_PATH=~/workspace/esp-idf

make menuconfig
```

2. 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.

```
[genmisc@Ubuntu 01_hello_world]$make help
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.
```

2. Compiling all binaries

Run the following command:

```
make
```

OR,

```
make all
```

If successful, the system will output the following information:

```
To flash all build output, run 'make flash' or:
python /home/genmisc/workspace/esp-idf//components/esptool_py/esptool/esptool.py --chip esp
32 --port /dev/ttyUSB0 --baud 115200 write_flash -z --flash_mode dio --flash_freq 40m --fla
sh_size 2MB 0x1000 /home/genmisc/workspace/01_hello_world/build/bootloader/bootloader.bin 0
x10000 /home/genmisc/workspace/01_hello_world/build/hello-world.bin 0x4000 /home/genmisc/wo
rkspace/01_hello_world/build/partitions_singleapp.bin
```

bootloader, ***partition*** and ***app*** binaries are compiled by default. Users can follow the instructions to download the binaries.

3. Compiling bootloader

Run the following command:

```
make bootloader
```

If successful, the system will output:

```
[genmisc@Ubuntu 01_hello_world]$make bootloader
make[1]: Entering directory `/home/genmisc/workspace/esp-idf/components/bootloader/src'
GENCONFIG
#
# configuration written to /home/genmisc/workspace/01_hello_world/build/bootloader/sdkconfi
g
#
make[1]: Leaving directory `/home/genmisc/workspace/esp-idf/components/bootloader/src'
make[1]: Entering directory `/home/genmisc/workspace/esp-idf/components/bootloader/src'
make[2]: Entering directory `/home/genmisc/workspace/01_hello_world/build/bootloader/log'
CC log.o
AR liblog.a
make[2]: Leaving directory `/home/genmisc/workspace/01_hello_world/build/bootloader/log'
make[2]: Entering directory `/home/genmisc/workspace/01_hello_world/build/bootloader/spi_fl
ash'
CC spi_flash_rom_patch.o
AR libspi_flash.a
make[2]: Leaving directory `/home/genmisc/workspace/01_hello_world/build/bootloader/spi_fla
sh'
make[2]: Entering directory `/home/genmisc/workspace/01_hello_world/build/bootloader/main'
CC bootloader_start.o
CC flash_encrypt.o
CC secure_boot.o
AR libmain.a
make[2]: Leaving directory `/home/genmisc/workspace/01_hello_world/build/bootloader/main'
LD bootloader.elf
esptool.py v2.0-dev
make[1]: Leaving directory `/home/genmisc/workspace/esp-idf/components/bootloader/src'
Bootloader built. Default flash command is:
python /home/genmisc/workspace/esp-idf//components/esptool_py/esptool/esptool.py --chip esp
32 --port /dev/ttyUSB0 --baud 115200 write_flash -z --flash_mode dio --flash_freq 40m --fla
sh_size 2MB 0x1000 /home/genmisc/workspace/01_hello_world/build/bootloader/bootloader.bin
```

Then, follow the instructions to download the binary to flash.

4. Compiling the user program

Run the following command:

```
make app
```

If successful, the system will output:

```
App built. Default flash app command is:
python /home/genmisc/workspace/esp-idf//components/esptool_py/esptool/esptool.py --chip esp
32 --port /dev/ttyUSB0 --baud 115200 write_flash -z --flash_mode dio --flash_freq 40m --fla
sh_size 2MB 0x10000 /home/genmisc/workspace/01_hello_world/build/hello-world.bin
```

Then, follow the instructions to download the user program to the flash.

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 ***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 system will output:

```
[genmisc@Ubuntu 01_hello_world]$make partition_table
Partition table binary generated. Contents:
*******************************************************************************
# Espressif ESP32 Partition Table
# Name, Type, SubType, Offset, Size
factory,app,factory,0x10000,1M
rfdata,data,rf,0x110000,256K
wifidata,data,wifi,0x150000,256K
*******************************************************************************
Partition flashing command:
python /home/genmisc/workspace/esp-idf//components/esptool_py/esptool/esptool.py --chip esp
32 --port /dev/ttyUSB0 --baud 115200 write_flash 0x4000 /home/genmisc/workspace/01_hello_wo
rld/build/partitions_singleapp.bin
```

Then, follow the instructions to download the partition table to the flash.

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 instructions mentioned in ***Section 2.4.2***, or, with specific commands.

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. Downloading 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. Downloading user program to flash

Run the following command:

```
make app flash
```

3. Downloading partition table to flash

Run the following command:

```
make partition_table-flash
```

4. Downloading all binaries to flash

Run the following command:

```
make flash
```

Espressif IOT Team

www.espressif.com