ESP32−C3 教程

尚硅谷

一　概述

ESP32-C3 SoC 芯片支持以下功能：

- 2.4 GHz Wi-Fi
- 低功耗蓝牙
- 高性能 32 位 RISC-V 单核处理器
- 多种外设
- 内置安全硬件

ESP32-C3 采用 40 nm 工艺制成，具有最佳的功耗性能、射频性能、稳定性、通用性和可靠性，适用于各种应用场景和不同功耗需求。

此芯片由乐鑫公司开发。

二　安装开发工具 ESP−IDF

ESP-IDF 需要安装一些必备工具，才能围绕 ESP32-C3 构建固件，包括 Python、Git、交叉编译器、CMake 和 Ninja 编译工具等。

在本入门指南中，我们通过 命令行 进行有关操作。

> ⚠ Warning
>
> 限定条件：
>
> - 请注意 ESP-IDF 和 ESP-IDF 工具的安装路径不能超过 90 个字符，安装路径过长可能会导致构建失败。
> - Python 或 ESP-IDF 的安装路径中一定不能包含空格或括号。
> - 除非操作系统配置为支持 Unicode UTF-8，否则 Python 或 ESP-IDF 的安装路径中也不能包括特殊字符（非 ASCII 码字符）
>
> 系统管理员可以通过如下方式将操作系统配置为支持 Unicode UTF-8：控制面板-更改日期、时间或数字格式-管理选项卡-更改系统地域-勾选选项 “Beta：使用 Unicode UTF-8 支持全球语言”-点击确定-重启电脑。

2.1　离线安装 ESP−IDF

点击链接下载离线安装包。

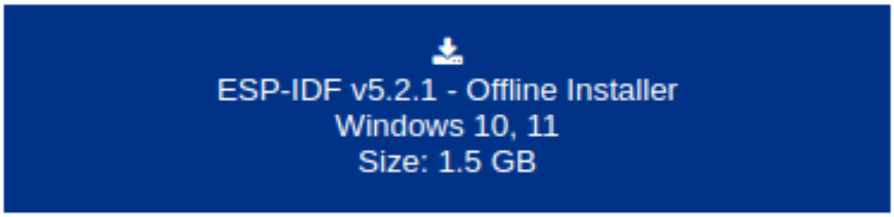

图 1　离线安装包示意图

2.2　安装内容

安装程序会安装以下组件：

- 内置的 Python
- 交叉编译器
- OpenOCD
- CMake 和 Ninja 编译工具
- ESP-IDF

安装程序允许将程序下载到现有的 ESP-IDF 目录。推荐将 ESP-IDF 下载到 `%userprofile%\Desktop\esp-idf` 目录下，其中 `%userprofile%` 代表家目录。

2.3　启动 ESP–IDF 环境

安装结束时，如果勾选了 `Run ESP-IDF PowerShell Environment` 或 `Run ESP-IDF Command Prompt (cmd.exe)`，安装程序会在选定的提示符窗口启动 `ESP-IDF`。

`Run ESP-IDF PowerShell Environment`：

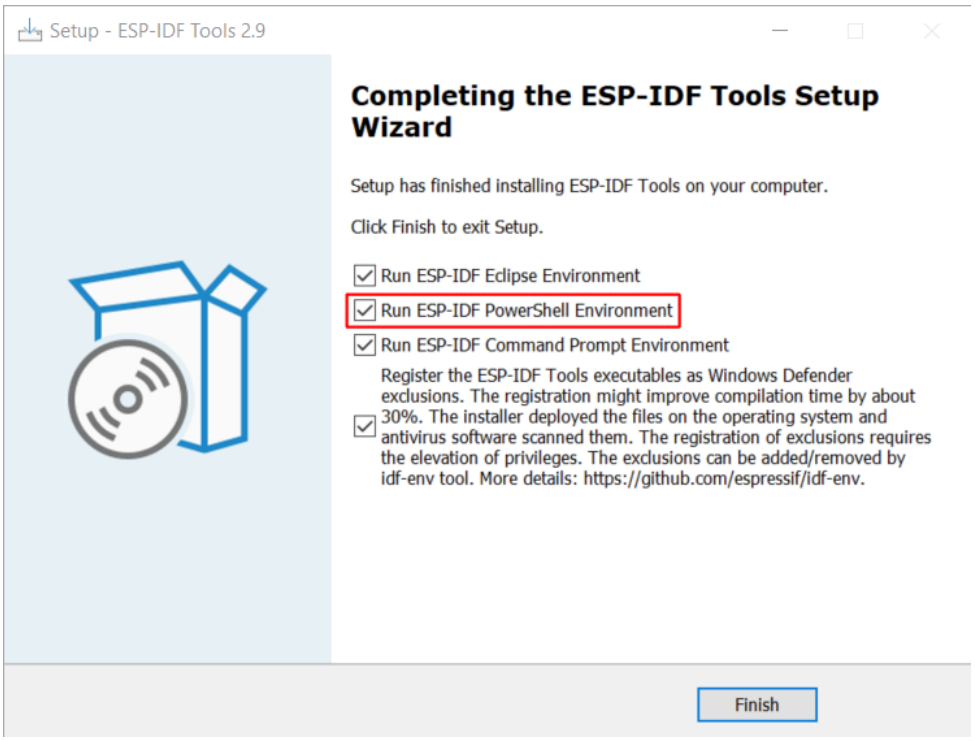

图 2　PowerShell

三　开始创建工程

现在，可以准备开发 ESP32 应用程序了。可以从 `ESP-IDF` 中 `examples` 目录下的 `get-started/hello_world` 工程开始。

> **Warning**
>
> ESP-IDF 编译系统不支持 ESP-IDF 路径或其工程路径中带有空格。

将 `get-started/hello_world` 工程复制至本地的 `~/esp` 目录下：

```
cd %userprofile%\esp
xcopy /e /i %IDF_PATH%\examples\get-started\hello_world hello_world
```

> **Info**
>
> ESP-IDF 的 examples 目录下有一系列示例工程，可以按照上述方法复制并运行其中的任何示例，也可以直接编译示例，无需进行复制。

3.1 连接设备

现在，请将 ESP32 开发板连接到 PC，并查看开发板使用的串口。

在 Windows 操作系统中，串口名称通常以 COM 开头。

3.2 配置工程

请进入 `hello_world` 目录，设置 ESP32-C3 为目标芯片，然后运行工程配置工具 `menuconfig` 。

```
cd %userprofile%\esp\hello_world
idf.py set-target esp32c3
idf.py menuconfig
```

打开一个新工程后，应首先使用 `idf.py set-target esp32c3` 设置“目标”芯片。注意，此操作将清除并初始化项目之前的编译和配置（如有）。也可以直接将“目标”配置为环境变量（此时可跳过该步骤）。

正确操作上述步骤后，系统将显示以下菜单：

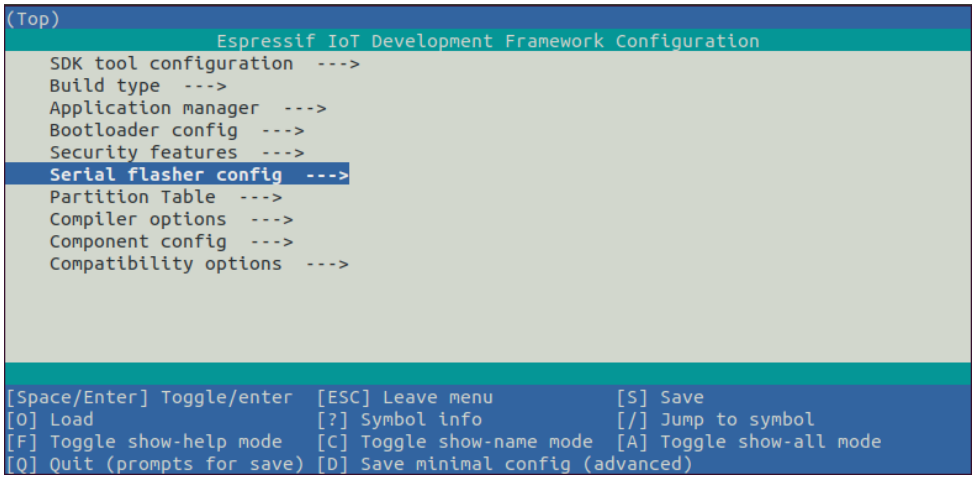

图 3　配置界面示意图

可以通过此菜单设置项目的具体变量，包括 Wi-Fi 网络名称、密码和处理器速度等。`hello_world` 示例项目会以默认配置运行，因此在这一项目中，可以跳过使用 `menuconfig` 进行项目配置这一步骤。

3.3 编译工程

请使用以下命令，编译烧录工程：

```
idf.py build
```

运行以上命令可以编译应用程序和所有 ESP-IDF 组件，接着生成引导加载程序、分区表和应用程序二进制文件。

```
$ 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'
```

如果一切正常，编译完成后将生成 `.bin` 文件。

3.4 烧录到设备

请运行以下命令，将刚刚生成的二进制文件烧录至 ESP32 开发板：

```
idf.py flash
```

> **Info**
>
> 勾选 `flash` 选项将自动编译并烧录工程，因此无需再运行 `idf.py build`。

3.5 常规操作

在烧录过程中，会看到类似如下的输出日志：

```
...
esptool.py --chip esp32 -p /dev/ttyUSB0 -b 460800 --
before=default_reset --after=hard_reset write_flash --flash_mode dio
--flash_freq 40m --flash_size 2MB 0x8000 partition_table/partition-
table.bin 0x1000 bootloader/bootloader.bin 0x10000 hello_world.bin
esptool.py v3.0-dev
Serial port /dev/ttyUSB0
Connecting........_
Chip is ESP32D0WDQ6 (revision 0)
Features: WiFi, BT, Dual Core, Coding Scheme None
Crystal is 40MHz
MAC: 24:0a:c4:05:b9:14
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 460800
Changed.
Configuring flash size...
Compressed 3072 bytes to 103...
Writing at 0x00008000... (100 %)
Wrote 3072 bytes (103 compressed) at 0x00008000 in 0.0 seconds
(effective 5962.8 kbit/s)...
Hash of data verified.
Compressed 26096 bytes to 15408...
Writing at 0x00001000... (100 %)
Wrote 26096 bytes (15408 compressed) at 0x00001000 in 0.4 seconds
(effective 546.7 kbit/s)...
Hash of data verified.
Compressed 147104 bytes to 77364...
Writing at 0x00010000... (20 %)
Writing at 0x00014000... (40 %)
Writing at 0x00018000... (60 %)
Writing at 0x0001c000... (80 %)
Writing at 0x00020000... (100 %)
Wrote 147104 bytes (77364 compressed) at 0x00010000 in 1.9 seconds
(effective 615.5 kbit/s)...
Hash of data verified.

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

如果一切顺利，烧录完成后，开发板将会复位，应用程序 “hello_world” 开始运行。

3.6 监视输出

使用 串口助手 监视输出和调试。

> ⚠ Warning
>
> 当要进行烧写时，请关闭串口助手！

四 基本 GPIO 操作

4.1 GPIO 配置

普通配置

```
gpio_config_t io_conf;
// 禁用中断
io_conf.intr_type = GPIO_INTR_DISABLE;
// 设置 GPIO 为输出模式
io_conf.mode = GPIO_MODE_OUTPUT;
// 设置 GPIO PIN 脚
io_conf.pin_bit_mask = ((1ULL << GPIO_NUM_1) | (1ULL <<
GPIO_NUM_2));
// 禁用下拉模式
io_conf.pull_down_en = 0;
// 开启上拉模式
io_conf.pull_up_en = 1;
// 使用以上配置来配置 GPIO
gpio_config(&io_conf);
```

有关中断的配置方法

```
// 上升沿触发中断
io_conf.intr_type = GPIO_INTR_POSEDGE;
// 设置为输入模式
io_conf.mode = GPIO_MODE_INPUT;
io_conf.pin_bit_mask = (1ULL << GPIO_NUM_0);
gpio_config(&io_conf);
```

操作 GPIO 的 API

```

```

```
// 将 GPIO 口设置为输入模式
gpio_set_direction(GPIO_NUM_2, GPIO_MODE_INPUT);
// 设置输出模式
gpio_set_direction(GPIO_NUM_2, GPIO_MODE_OUTPUT);
// 输出高低电平
gpio_set_level(GPIO_NUM_1, 1);
gpio_set_level(GPIO_NUM_1, 0);
// 获取 GPIO 的电平
gpio_get_level(GPIO_NUM_2);
```