环境搭建补充
和工程目录解析

淘宝店铺：

PC 端：

http://n-xytrt8gqu585po94mwj5atokcyd4.taobao.com/index.htm

手机端：

https://shop.m.taobao.com/shop/shop_index.htm?sellerId=755668508&shopId=104493595&inShopPageId=423890608&pathInfo=shop/index2

资料下载地址：

链接：https://pan.baidu.com/s/1kCjD8yktZECSGmHomx_veg?pwd=q8er
提取码：q8er

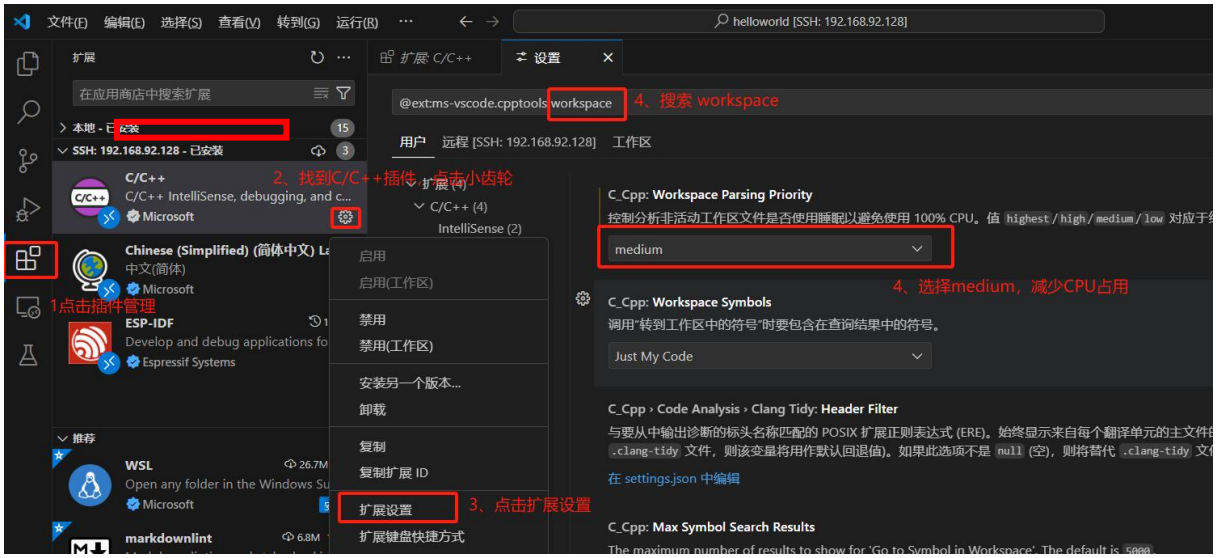
源码下载地址：

https://gitee.com/vi-iot/esp32-board.git

一、补充

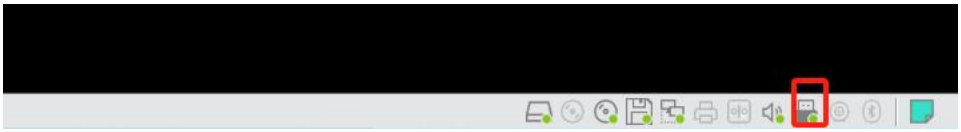
在上一课介绍了如何完整的搭建 esp-idf 开发环境以及所需要的工具，现在大家可以编译和方便的查看代码了。但如果想要做得更好，还有一些东西需要我们设置一下。

1）减少 C/C++插件的 CPU 占用，在使用 ssh 登录到 ubuntu 后，我们点击插件管理（注意不是本地上安装的插件，必须是 ubuntu 上安装的 C/C++插件，如下深红框所示），然后按照下图步骤设置即可。



2）完善 USB 设置

这里 USB 设置的意思是，我们的开发板通过 Type-c 转 USB 连接到我们的电脑进行下载和打印调试，为了方便操作，我们都是连接到虚拟机上进行下载和调试，我们启动虚拟机，并且插上开发板 Type-c 调试线后，会弹出一个提示框，提示是连接到主机还是虚拟机，我们选择虚拟机后，虚拟机右下角就会出现 USB 已连接的图标，大家也可以手动点这个图标进行连接和断开。



但按照默认设置，使用起来有点问题，ubuntu 里面会经常连接不上串口设备提示。解决方法如下图

设置完成后，重启一下虚拟机系统

3）USB 串口权限问题

根据上节教程，我们布置好了开发环境，并且用户登录后自动执行 esp-idf 环境变量的设置，就可以使用 idf.py 这个工具了，在 helloworld 工程目录下，idf.py build 是执行编译生成 bin 文件，然后执行 idf.py flash 则是将程序烧录进开发板。烧录的前提是我们的开发板通过 type-c 线连接到电脑上的虚拟机。但我们输入 idf.py flash 后，会提示权限不足：

Permission denied: '/dev/ttyUSB0'

临时的解决方法是，输入 sudo chmod 777 /dev/ttyUSB0，然后输入用户密码，这样就把 ttyUSB0 串口设备文件开到最大了。但这有个问题，当我们重新插拔后，还需要重复上述步骤。而一劳永逸的方法是输入如下指令

sudo usermod -aG dialout usrname

这句命令最后的 usrname 需要换成你自己 ubuntu 的用户名。然后重启系统即可

4）使用证书 SSH 登录到 ubuntu

我们在使用 VSCode，远程登录 ubuntu，打开某个工程文件夹，每次都需要输入至少两次密码，有没有办法不用密码直接登录呢？这当然是可以的。需要用到加密证书登录，操作步骤如下：

1.Windows 系统下，打开 powershell，输入如下命令

ssh-keygen -t ed25519 -C youremail@xxx.com

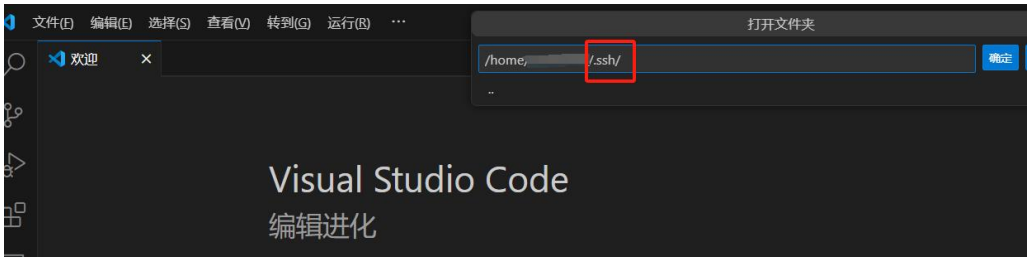
上述 youremail@xxx.com 要换成你自己的邮箱，其实不用太在乎邮箱，随便输一个，没什么用。然后一路回车

2.C:\Users\${当前用户}\.ssh 目录下找到 id_ed25519.pub

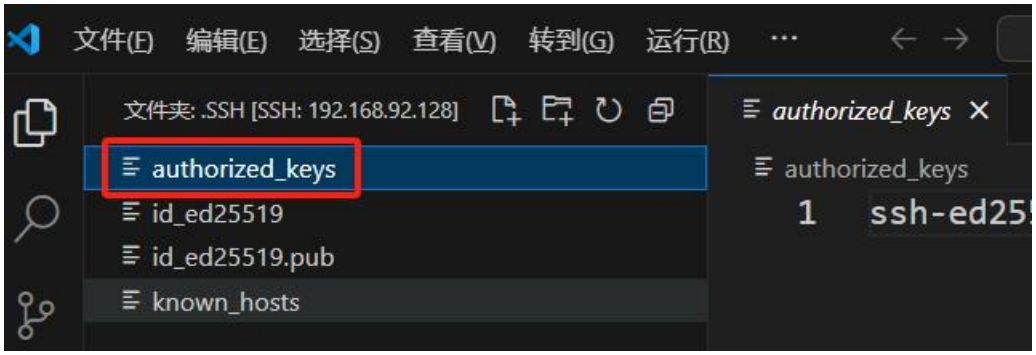
上述${当前用户}是你当前登录 windows 的用户名
用文本打开这个文件，然后复制里面的内容

```
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAI[redacted]
```

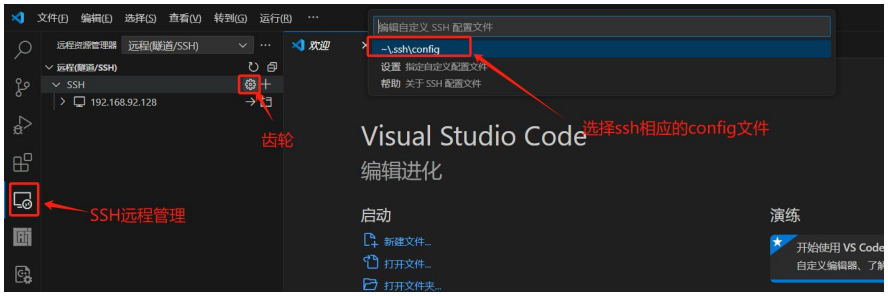
3.使用 VSCode 远程登录到 ubuntu，打开.ssh 文件夹



找到 authorized_keys 文件，将刚才 windows 下生成的 id_ed25519.pub 里面的内容复制到这个 authorized_keys 文件里面。



4.最后一步需要在 VSCode 中添加验证文件路径（这一步应该不做也可以，大家如果还是要输密码，就把这步也加上）

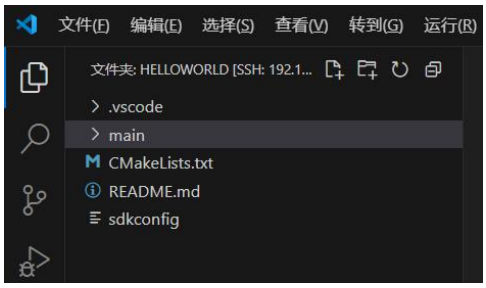


IdentityFile "C:\Users\${当前用户}\.ssh\id_ed25519.pub",${当前用户}要替换成你 windows 下登录的用户名

完成后重启 VSCode

二、工程目录解析

上一章，我们补充了环境搭建的细节，以后我们可以更加快速方便的开发我们的项目，这一章我们看下 helloworld 工程有哪些文件，分别有什么用。

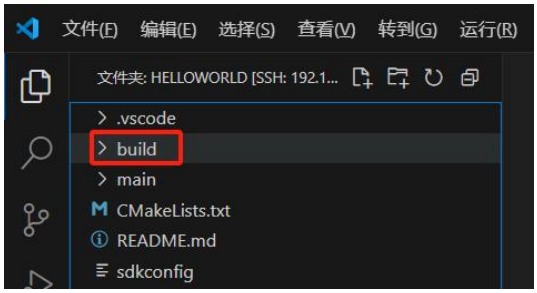


这个是大家 git clone 下来的 helloworld 目录文件，现在我们用 mobaxterm 登录上去后，进行编译看看

cd esp32/esp32-board/helloworld

idf.py build

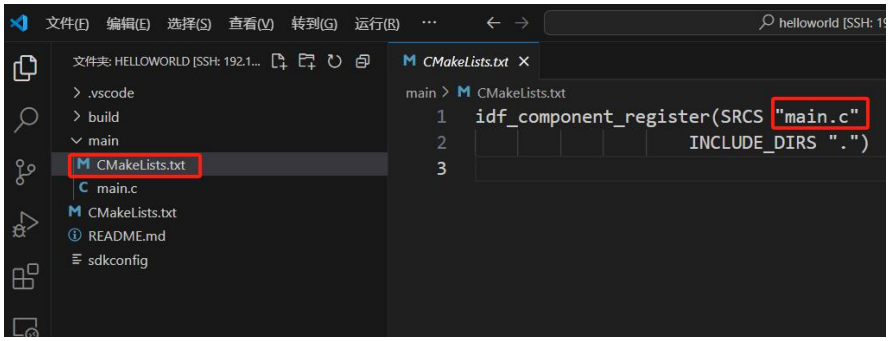
完成编译后会多一个 build 文件夹，里面包含 bin 文件和中间编译文件



现在介绍一下每个文件有什么作用

main/目录，项目工程的主要源码目录，这个是必须要存在的，不然编译不过

main/目录下有 main.c 里面包含我们应用代码的入口函数 app_main(), 这个 CMakeLists.txt 用于指示 CMake 构建系统，对 main/目录进行构建，大家增加源码文件，需要在这个 CMakeLists.txt 中增加源码文件。



项目顶层也有一个 CMakeLists.txt 文件，用于指示 CMake 构建系统，对 helloworld 工程进行构建，如果我们想要修改工程名称，需要修改这个文件中 project(main)中的 main，改成我们想要的名称即可。

sdkconfig 文件，这个是工程配置文件，如果不存在，idf 会为我们生成一个默认的，这个文件也很关键，以后我们用 idf.py menuconfig 命令配置工程，就会自动生成这个文件

build/目录执行编译后，生成的 bin 和中间文件存放，自动生成

接下来我们看一下生成的 bin 放在 build 的什么地方，其实我们这个工程生成了几个有用的 bin，大家看下 cat build/flasher_args.json 这个文件


```
{
    "write_flash_args" : [ "--flash_mode", "dio",
                           "--flash_size", "2MB",
                           "--flash_freq", "40m" ],
    "flash_settings" : {
        "flash_mode": "dio",
        "flash_size": "2MB",
        "flash_freq": "40m"
    },
    "flash_files" : {
        "0x1000" : "bootloader/bootloader.bin",
        "0x10000" : "main.bin",
        "0x8000" : "partition_table/partition-table.bin"
    },
    "bootloader" : { "offset" : "0x1000", "file" : "bootloader/bootloader.bin", "encrypted" : "false" },
    "app" : { "offset" : "0x10000", "file" : "main.bin", "encrypted" : "false" },
    "partition-table" : { "offset" : "0x8000", "file" : "partition_table/partition-table.bin", "encrypted" : "false" },
    "extra_esptool_args" : {
        "after"  : "hard_reset",
        "before" : "default_reset",
        "stub"   : true,
        "chip"   : "esp32"
    }
}
```


里面清楚的可以看到，我们生成了这三个文件，并且分别对应烧录的位置。

bootloader，应用启动前的一段代码。

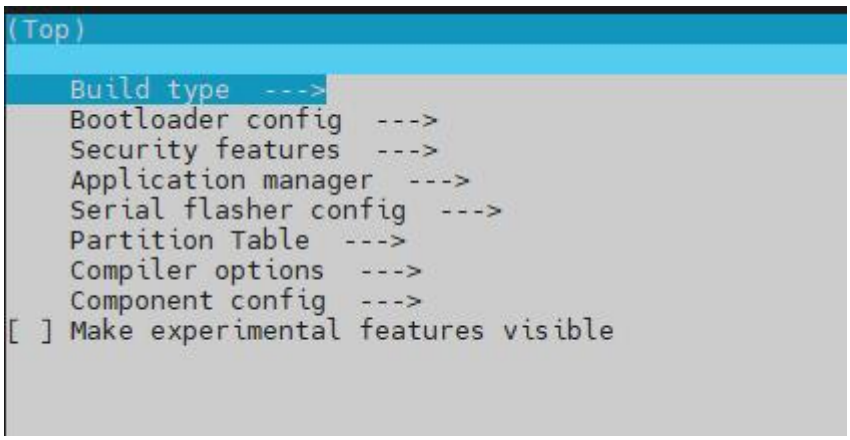
main.bin，我们的应用代码

partition-table.bin 分区表，用于表示我们的几个区分别存放了什么，这个在后续的教程中会介绍

大家以后要新建项目，建议从现有的工程拷贝一份出来再修改，这样效率非常高。

三、工程配置

esp-idf 如此庞大，但其中很多功能不是我们项目中要用到的，或者说一些参数我们要调整，因此我们需要对我们的工程项目进行配置，在我们使用 mobaxterm 工具登录到我们的虚拟机系统后，在 esp32-board/helloworld 目录下，我们输入 idf.py menuconfig，弹出如下界面，我们使用键盘的上下左右和回车进行操作



这里面有十分多的内容可以配置，一时半会我们也学习不完，而且大部分配置我们是可以用默认值的。在日后学习的过程中，我们用到哪个就去配哪个，现在我们先修改如下几个值：

工作主频->默认 160Mhz，ESP32 最高支持 240Mhz

Flash 大小->默认 2M，开发板上的模组是 4M Flash

PSRAM->默认没有，开发板上的模组是 2M PSRAM

工作主频：

Component config ---> ESP System Settings --->CPU frequency (160 MHz) 改为 240MHz

Flash 大小：

Serial flasher config --->Flash size (2 MB) 改为 4MB

PSRAM 大小：

Component config --->ESP PSRAM --->启用 Support for external, SPI-connected RAM

修改完成后，按 s，回车保持，q 退出，输入 idf.py build 重新编译

四、烧录和下载

对于程序的烧录，调试时我们用最简单方式，我们板子通过 Type-c 线插到电脑上，选择连接虚拟机，板子上已设计好一键下载电路，我们无需按任何按键就可以下载。在 idf.py build 编译完成后，我们输入 idf.py flash 命令直接将编译的 bin 文件烧录到板子中，烧录正确的话如下图打印所示：

```
vi@vi:~/esp32/esp32-board/helloworld$ idf.py build
Executing action: all (aliases: build)
Running ninja in directory /home/vi/esp32/esp32-board/helloworld/build
Executing "ninja all"...
[1/4] cd /home/vi/esp32/esp32-board/helloworld/build/esp-idf/esptool_py && /home
main.bin binary size 0x37110 bytes. Smallest app partition is 0x100000 bytes. 0x
[1/1] cd /home/vi/esp32/esp32-board/helloworld/build/bootloader/esp-idf/esptool_
Bootloader binary size 0x6850 bytes. 0x7b0 bytes (7%) free.

Project build complete. To flash, run:
 idf.py flash
or
 idf.py -p PORT flash
or
 python -m esptool --chip esp32 -b 460800 --before default_reset --after hard_re
on_table/partition-table.bin 0x10000 build/main.bin
or from the "/home/vi/esp32/esp32-board/helloworld/build" directory
 python -m esptool --chip esp32 -b 460800 --before default_reset --after hard_re
vi@vi:~/esp32/esp32-board/helloworld$ idf.py flash
Executing action: flash
Serial port /dev/ttyUSB0
Connecting.....
Detecting chip type... Unsupported detection protocol, switching and trying agai
Connecting....
Detecting chip type... ESP32
Running ninja in directory /home/vi/esp32/esp32-board/helloworld/build
Executing "ninja flash"...
[1/5] cd /home/vi/esp32/esp32-board/helloworld/build/esp-idf/esptool_py && /home
main.bin binary size 0x37110 bytes. Smallest app partition is 0x100000 bytes. 0x
[1/1] cd /home/vi/esp32/esp32-board/helloworld/build/bootloader/esp-idf/esptool_
Bootloader binary size 0x6850 bytes. 0x7b0 bytes (7%) free.
[2/3] cd /home/vi/esp32/esp-idf/components/esptool_py && /usr/bin/cmake -D IDF_P
esptool.py --chip esp32 -p /dev/ttyUSB0 -b 460800 --before=default_reset --after
x8000 partition_table/partition-table.bin
esptool.py v4.7.0
Serial port /dev/ttyUSB0
Connecting....
Chip is ESP32-D0WDR2-V3-V3 (revision v3.0)
Features: WiFi, BT, Dual Core, 240MHz, VRef calibration in efuse, Coding Scheme
Crystal is 40MHz
MAC: 80:64:6f:dd:6a:d0
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 460800
Changed.
Configuring flash size...
Flash will be erased from 0x00001000 to 0x00007fff...
Flash will be erased from 0x00010000 to 0x00047fff...
Flash will be erased from 0x00008000 to 0x00008fff...
Compressed 26704 bytes to 16351...
Writing at 0x00001000... (100 %)
Wrote 26704 bytes (16351 compressed) at 0x00001000 in 0.9 seconds (effective 238
Hash of data verified.
Compressed 225552 bytes to 112078...
Writing at 0x0002563a... (42 %)
```

那我们如何查看串口打印呢？

程序烧录完成后，输入 idf.py monitor 命令，就可以启用监控串口打印了。

```
vi@vi:~/esp32/esp32-board/helloworld$ idf.py monitor
Executing action: monitor
Serial port /dev/ttyUSB0
Connecting.....
Detecting chip type... Unsupported detection protocol, switching and trying again...
Connecting.....
Detecting chip type... ESP32
Running idf_monitor in directory /home/vi/esp32/esp32-board/helloworld
Executing "/home/vi/.espressif/python_env/idf5.2_py3.8_env/bin/python /home/vi/esp32
home/vi/esp32/esp32-board/helloworld/build/main.elf -m '/home/vi/.espressif/python_e
--- esp-idf-monitor 1.4.0 on /dev/ttyUSB0 115200 ---
--- Quit: Ctrl+] | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---
ets Jul 29 2019 12:21:46

rst:0x1 (POWERON_RESET),boot:0x17 (SPI_FAST_FLASH_BOOT)
configsip: 153911750, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:2
load:0x3fff0030,len:7172
load:0x40078000,len:15532
load:0x40080400,len:4
0x40080400: _init at ??:?

load:0x40080404,len:3904
entry 0x40080640
I (30) boot: ESP-IDF v5.2 2nd stage bootloader
I (30) boot: compile time Apr 19 2024 14:51:28
I (30) boot: Multicore bootloader
I (34) boot: chip revision: v3.0
I (38) boot.esp32: SPI Speed      : 40MHz
```

注意！这个串口监控工具退出方法是 ctrl+]