使用cmake+ninja+clangd工具链在windows平台开发STM32（Linux平台只会更简单，留作课后作业，完成了可以发评论区）

- cmake
- ninja
- clangd
- arm-gcc

安装`arm-none-eabi-gcc`，不要问哪个版本，无脑最新就行

- 放个锚点，下载`arm-none-eabi-gcc`
- 目前的最新版本为`14.2.rel1`（2025-03-16）
- 下载好之后，放到一个稳定的路径下，来到`bin`目录下复制路径，添加到环境变量，类似这样
  - C:\Toolchain\arm-gnu-toolchain-14.2.rel1-mingw-w64-i686-arm-none-eabi\bin

安装`Git Bash`

- gitbash下载传送门

安装`msys2`，官网直接搜索下载就行

- 安装位置随便，C盘够用就默认安装，但其实推荐装到其他盘，类似D盘
- msys2下载传送门
- 最好是在网络流畅的环境下进行

将安装目录下的这三个路径添加到环境变量

```
# 可能会有暂时用不到的路径被添加到了环境变量，防止你以后装了别的工具，忘了添加，所以一次性加上
D:\msys64\mingw64\bin
D:\msys64\ucrt64\bin
D:\msys64\usr\bin
```

打开`msys2 ucrt64`，安装以下工具，并进行检查

```
# 一次性安装
pacman -S cmake ninja mingw-w64-ucrt-x86_64-openocd mingw-w64-ucrt-x86_64-clang-tools-extra
# 或者分开安装
pacman -S cmake
pacman -S ninja
pacman -S mingw-w64-ucrt-x86_64-openocd
pacman -S mingw-w64-ucrt-x86_64-clang-tools-extra
```

```
# 验证你的工具是否安装正确，分开验证
cmake --version
ninja --version
clangd --version
openocd --version

# 一次性验证也行，这样写
cmake --version && ninja --version && openocd --version && clangd --version
# 终端里输出以下信息则表明正确安装了工具

--------------------------------
Admin@0xBB8 UCRT64 ~
# cmake --version && ninja --version && openocd --version && clangd --version
cmake version 3.31.5

CMake suite maintained and supported by Kitware (kitware.com/cmake).
1.12.1
Open On-Chip Debugger 0.12.0
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.org/doc/doxygen/bugs.html
clangd version 19.1.7
Features: windows
Platform: x86_64-w64-windows-gnu
--------------------------------
```

装vscode

- vscode下载传送门，选择`system installer x64`版本下载并进行安装，安装位置随意
- 打开`vscode`，在插件市场搜索以下关键词安装对应的插件
  - clangd
  - cortex debug

装STM32CubeMX

- `STM32CubeMX`下载传送门(https://www.st.com/en/development-tools/stm32cubemx.html)，下载最新版本就行（写教程的时候是6.14）

使用STM32CubeMX初始化一个工程led_blink

- 生成代码时，`Toolchain`下拉选择`cmake`

使用`vscode`打开工程`led_blink`

我们会使用一些命令行来完成手动编译，以及下载烧录，主要是熟悉一下编译过程，也会给出全自动的脚本工具，一键运行即可完成编译和烧录

- 手动配置部分
  - 编译部分

> 1. 使用vscode打开工程后，打开一个终端，这里推荐使用git bash
> 2. 在打开的终端里输入以下命令

```
mkdir build #存放编译的过程文件和elf文件以及之后会提到的bin和hex文件
cd build    #进入build目录
cmake -G Ninja ..      使用camke生成ninja可以执行的编译文件build.ninja（可以对比理解成之前教程里提到的makefile）
ninja -j 8  ninja根据build.ninja执行构建，最后生成elf文件
# 输入并且执行（输入完命令后按下回车：Enter键）完以上四条命令后，build目录下就会生成一个led_blink.elf文件（这个文件可以用来调试，也可以直接烧录，还可以使用工具生成bin和hex文件供烧录调试）
arm-none-eabi-objcopy -O binary led_blink.elf led_blink.bin #转换为bin文件
arm-none-eabi-objcopy -O ihex led_blink.elf led_blink.bin   #转换为hex文件
```

- 烧录部分

> 1. 在工程根目录下建一个文件，取名叫flash，叫什么名字，有没有后缀，都不重要（视频里以flash为例）
> 2. 在flash文件里粘贴以下命令

```
source [find interface/stlink.cfg]
# source [find interface/cmsis-dap.cfg] #使用daplink请删去本行第一个井号#，并且注释第一行（在第一行前面加井号#）
source [find target/stm32f4x.cfg]

program C:/Users/Admin/Desktop/led_blink/build/led_blink.elf 0x08000000 verify reset exit  #烧录bin文件也行

# program后面的这个elf文件，请更换为自己工程的绝对路径，相对路径会出错
```

> 3. 在终端里，切换到根目录（跟flash在一个目录），执行命令

```
openocd -f flash
```

> 4. 经过以上步骤，就完成了烧录
> 5. 但是此时打开`main.c`开始写代码，你会发现全屏爆红，解决办法如下

1. 在`build`目录下找到`compile_commands.json`文件，并且将文件里的类似`/c/`使用查找替换为`C:/`。
2. 如果工程不在桌面，则还需要将其他盘符，类似`/d/`替换为`D:/`
   整个环境搭建的部分，但这里就算结束了，但是为了易用性，以下给出一个通用脚本

- 自动编译并且烧录
  - 将以下脚本复制到工程的根目录，并且取名为`run.sh`，叫什么名字还是随意，这里只是作为示例

```bash
#!/bin/bash

# 创建并进入 build 目录，每次都会删除之前的文件
mkdir -p build
cd build
rm -rf *

# 运行 CMake 和 Ninja 构建
cmake -G Ninja ..
cmake --build . --target all --config Release -- -j 16

# 获取当前工程所在的盘符
drive_letter=$(pwd | cut -d'/' -f2 | tr '[:lower:]' '[:upper:]')

# 修改 compile_commands.json 文件中的路径
sed -i "s|/c/|C:/|g; s|/d/|D:/|g; s|/e/|E:/|g; s|/f/|F:/|g; s|/g/|G:/|g; s|/h/|H:/|g;
s|/i/|I:/|g; s|/j/|J:/|g; s|/k/|K:/|g; s|/l/|L:/|g; s|/m/|M:/|g; s|/n/|N:/|g; s|/o/|O:/|g;
s|/p/|P:/|g; s|/q/|Q:/|g; s|/r/|R:/|g; s|/s/|S:/|g; s|/t/|T:/|g; s|/u/|U:/|g; s|/v/|V:/|g;
s|/w/|W:/|g; s|/x/|X:/|g; s|/y/|Y:/|g; s|/z/|Z:/|g" compile_commands.json

# 获取 .ioc 文件的前缀
ioc_file=$(basename ../*.ioc .ioc)
elf_file="$ioc_file.elf"
bin_file="$ioc_file.bin"
hex_file="$ioc_file.hex"

# 检查 ELF 文件是否存在
if [ ! -f "$elf_file" ]; then
    echo "Error: ELF file $elf_file not found!"
    exit 1
fi

# 生成二进制文件和 HEX 文件
arm-none-eabi-objcopy -O binary "$elf_file" "$bin_file"
arm-none-eabi-objcopy -O ihex "$elf_file" "$hex_file"

# 检查二进制文件是否生成成功
if [ ! -f "$bin_file" ]; then
    echo "Error: Binary file $bin_file not generated!"
    exit 1
fi

# 检查 HEX 文件是否生成成功
if [ ! -f "$hex_file" ]; then
    echo "Error: HEX file $hex_file not generated!"
    exit 1
fi

# 打印文件大小信息
echo "ELF file size:"
arm-none-eabi-size "$elf_file"

echo "Binary file information:"
```

```
file "$bin_file"

echo "HEX file size:"
arm-none-eabi-size "$hex_file"

# 烧录
openocd -f interface/stlink.cfg -f target/stm32f4x.cfg -c "program $hex_file verify reset
exit"
```

如果需要支持串口浮点打印和去除掉一些不必要的warning，可以将以下命令，加到`cmake/gcc-arm-none-eabi.cmake`中

```
# 有中文注释的部分，是cubemx不会自动生成的，需要手动添加
set(CMAKE_C_LINK_FLAGS "${CMAKE_C_LINK_FLAGS} -u _printf_float")  # 支持 printf 函数打印浮点数
set(CMAKE_C_LINK_FLAGS "${CMAKE_C_LINK_FLAGS} -lm")  # 链接数学库 libm
set(CMAKE_EXE_LINKER_FLAGS "-Wl,--gc-sections,--no-warn-rwx-segments")  # 取消 rwx 段的警告
set(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} -Wno-unused-parameter")  # 忽略 C 代码中未使用参数的警告
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -Wno-unused-parameter")  # 忽略 C++ 代码中未使用参数的警告
```

写在最后

- 如果在使用脚本后，发现代码依旧爆红，请按下`ctrl+shift+p`，输入`clangd`，找到`clangd: Restart Language Server`，使用鼠标点击，或者使用键盘上的方向键选中后按下回车，此时你的代码应该就不会爆红了，而且代码提示和跳转也会恢复正常
- 如果跟完视频/看完教程，你还是无法复刻这个开发环境，不要犹豫，立即求助这个帅哥，发私信就好，传送门在这
- 或者你想vx求助，这是神秘代码：bin0ZZA
- 此教程的工程以及markdown文件会上传到github，这是github主页，传送门在这