

1.3.1 GPIO

引子

GPIO，即General Purpose Input Output 通用功能输入输出

其实就是我们看到的芯片的引脚。如果你已经打开过STM32CubeMX软件并且看到了芯片的引脚图，那么上面标有例如“PA1”这样的圆形就是一个引脚，或称“pin口”

每一个pin口包含不止一种功能，但是进行配置时，我们只能选用它的一种功能进行使用，这种特性被称为"GPIO的复用”。

而本章介绍的GPIO更聚焦于狭义的GPIO，即常用的数字量电平输入输出。

打开CubeMX，同时打开C板用户手册（不知道在哪的 看这个：[📖 1.3 嵌入式知识](#)）

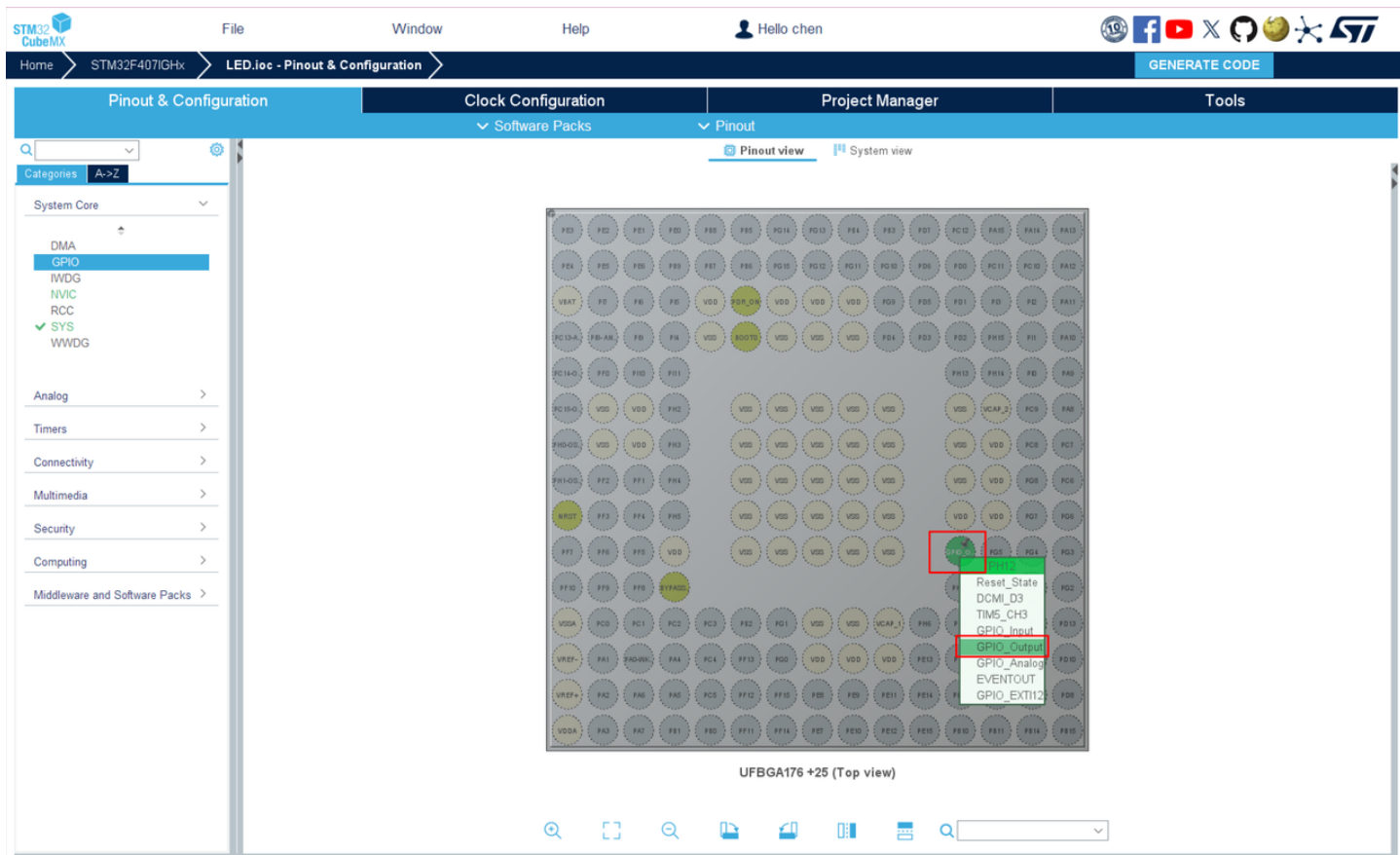
实践

下面我们先操作感受一下：

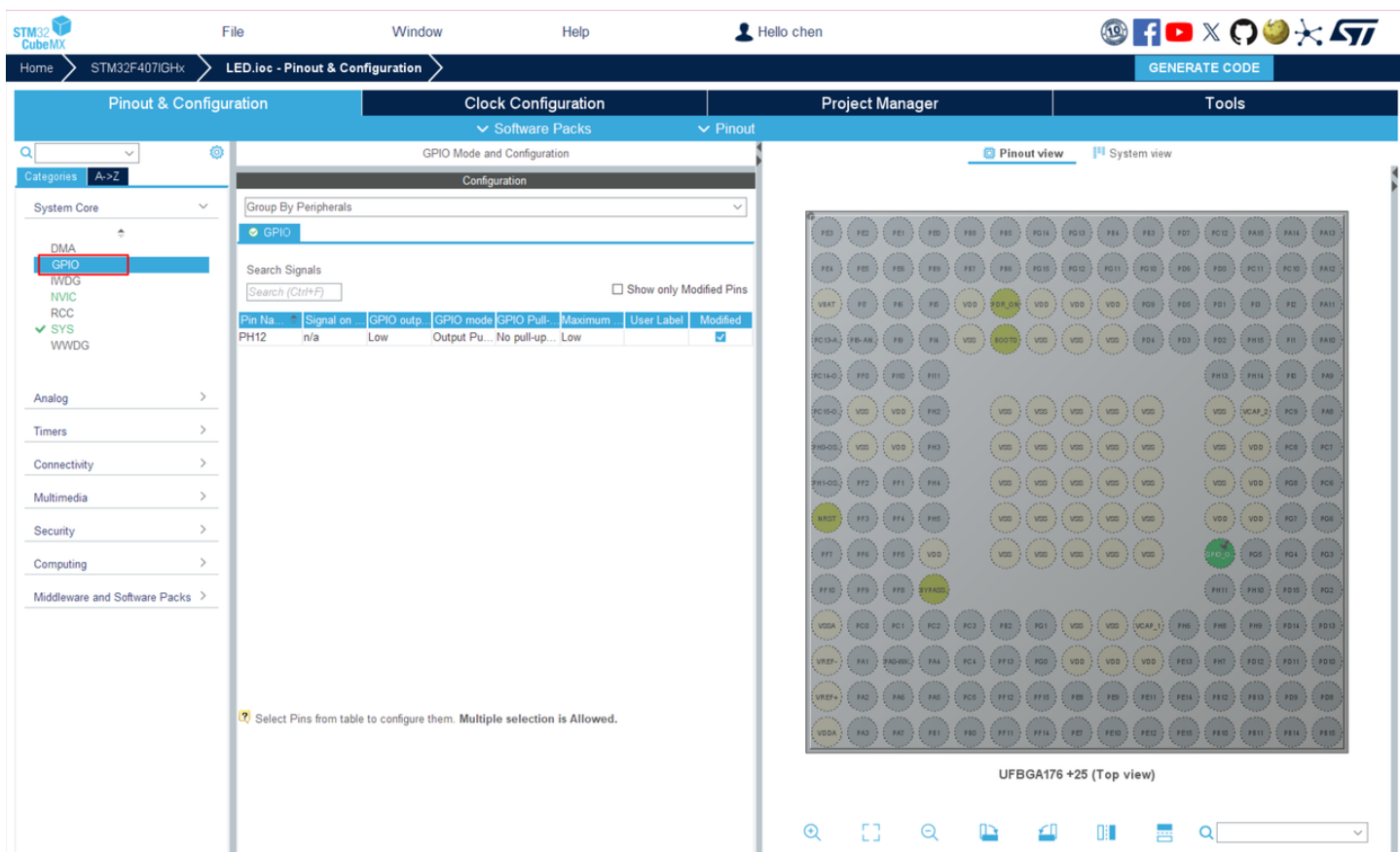
1. 首先根据用户手册第6页或者第17页可知该板子上已有三个LED灯，如果你已经看过了用户手册前面的部分，你应该知道这些LED灯具体在板子的什么位置。并且通过这两页，应该知道三个灯分别对应的pin口是什么。

显然地，这里红灯对应的是**PH12**引脚，我们不妨先试试让红灯亮一亮。

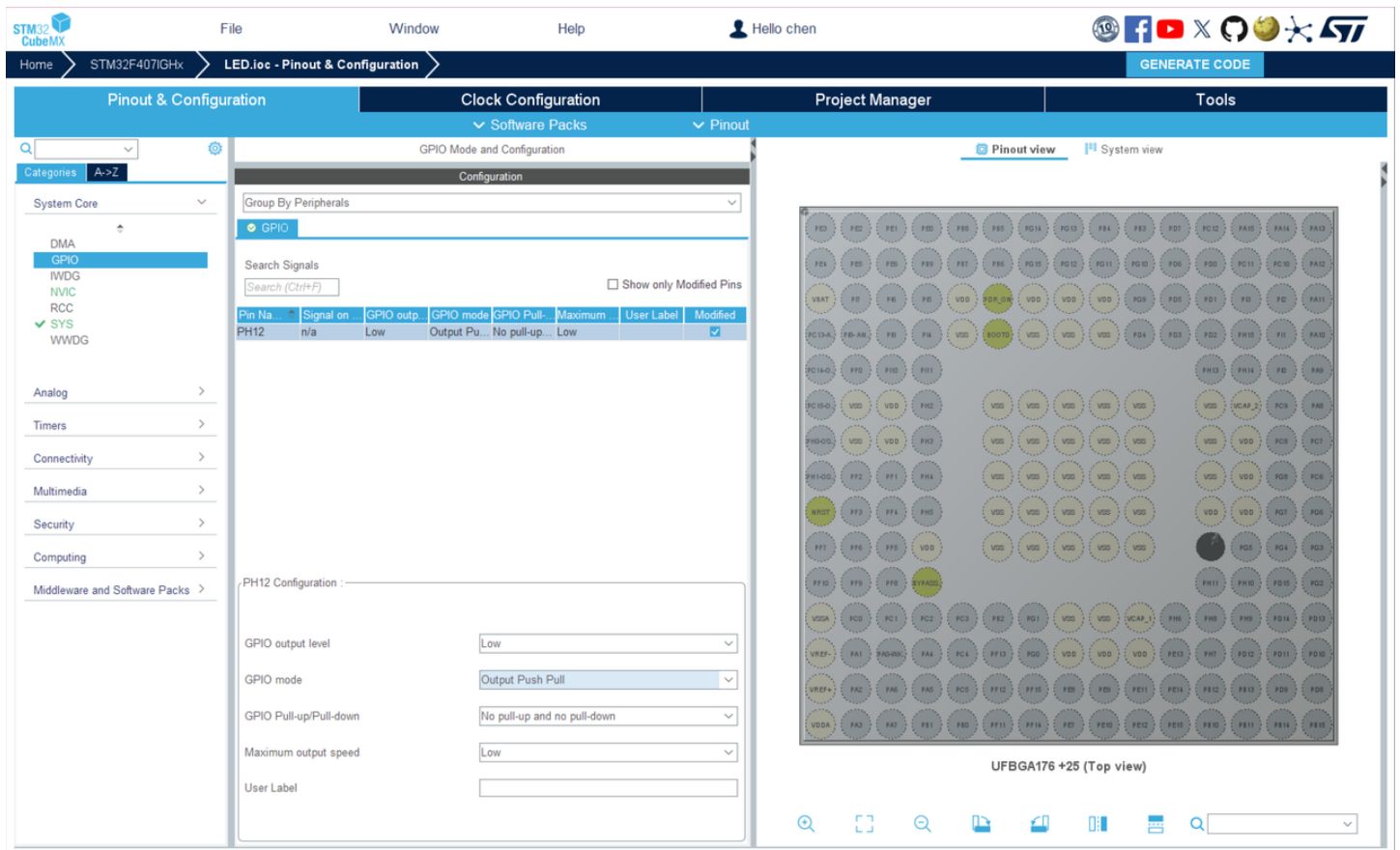
2. 打开CubeMX，新建工程，在芯片引脚处寻找PH12，并选取其中的GPIO_Output：



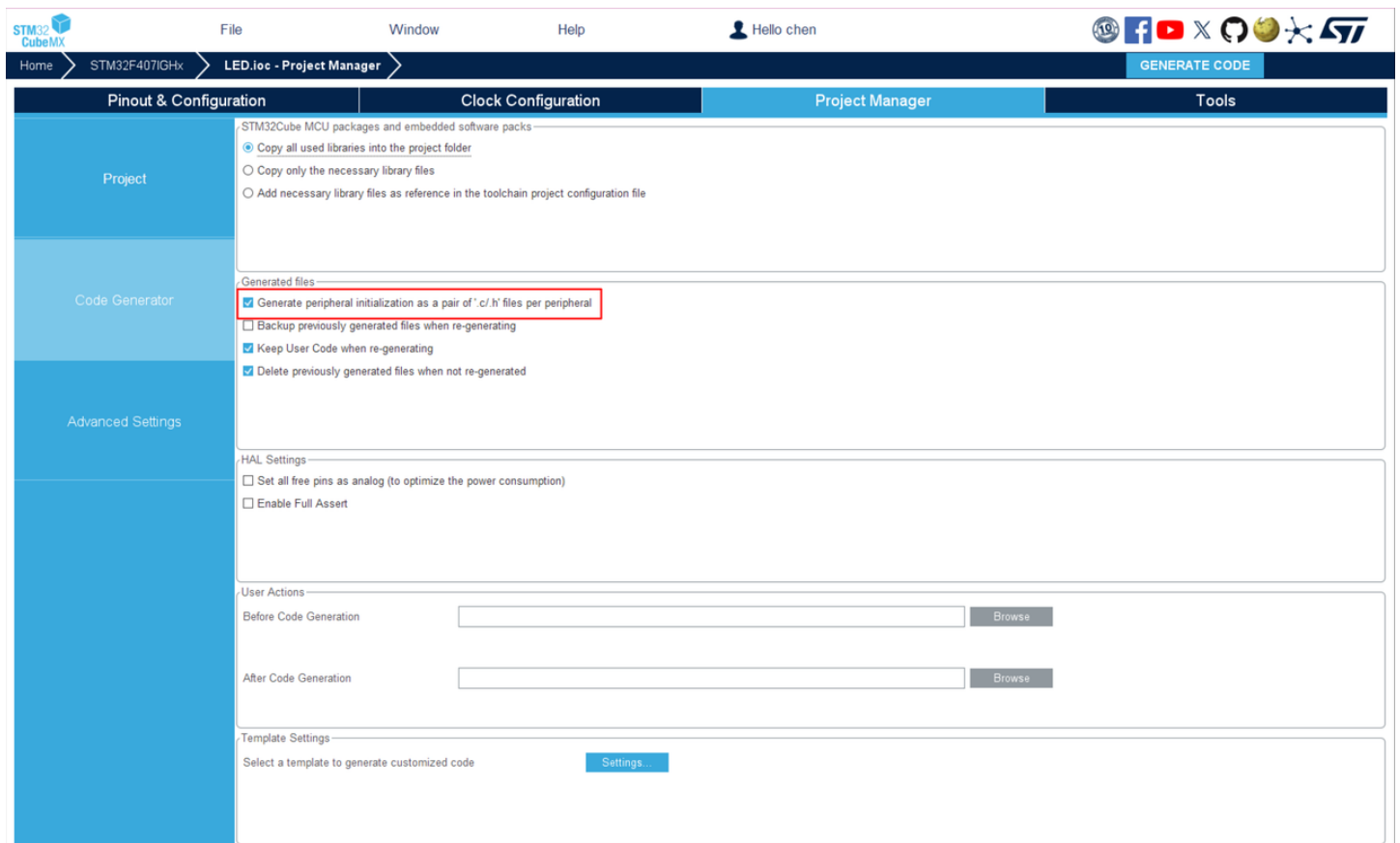
3. 则下面的引脚会变成绿色，左边点击System core选项，选其中的GPIO进行配置：



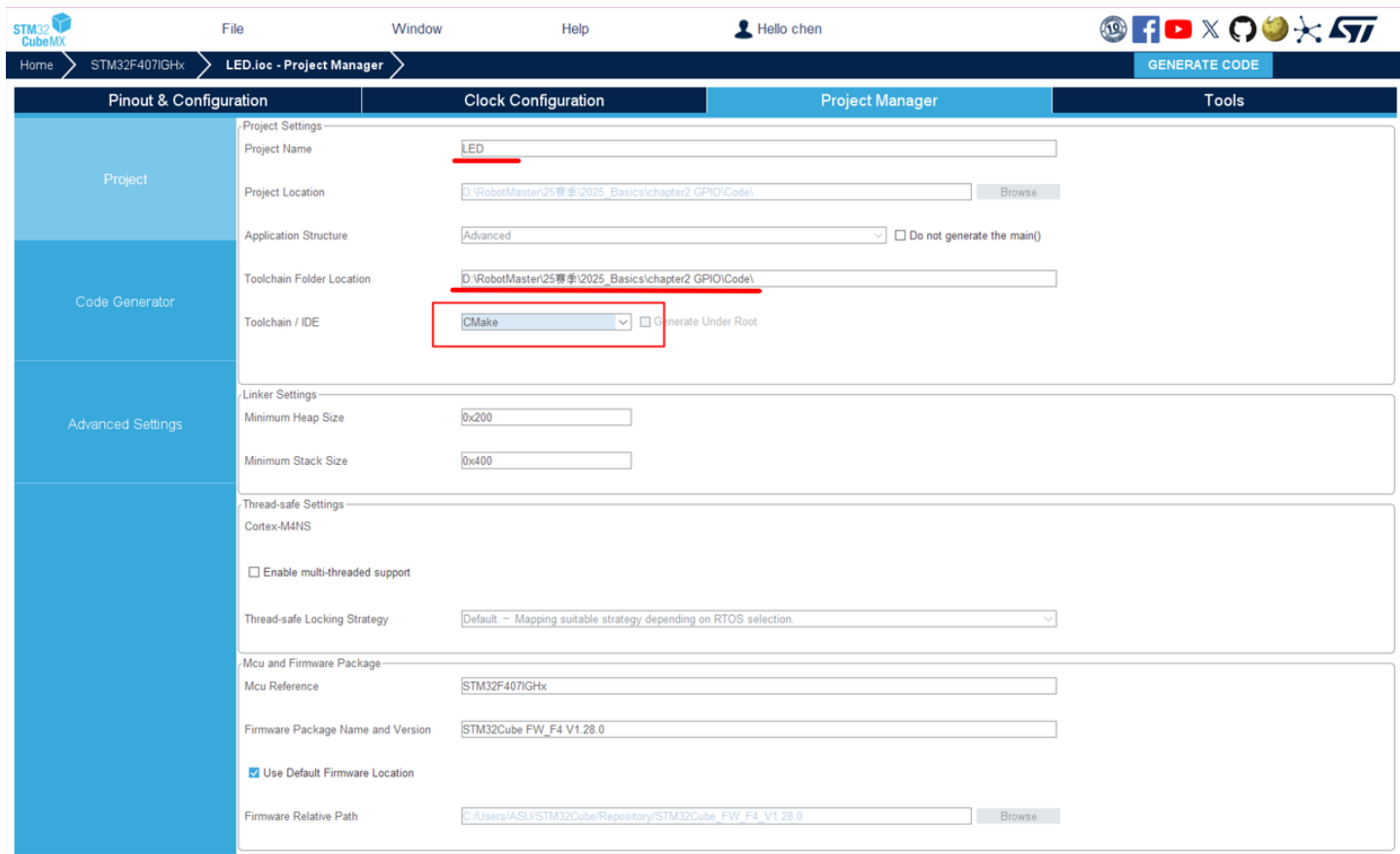
4. 点击PH12一栏，在下方选项卡中进行如下配置：



5. Project manager选项卡中，左侧的code generator如下配置：

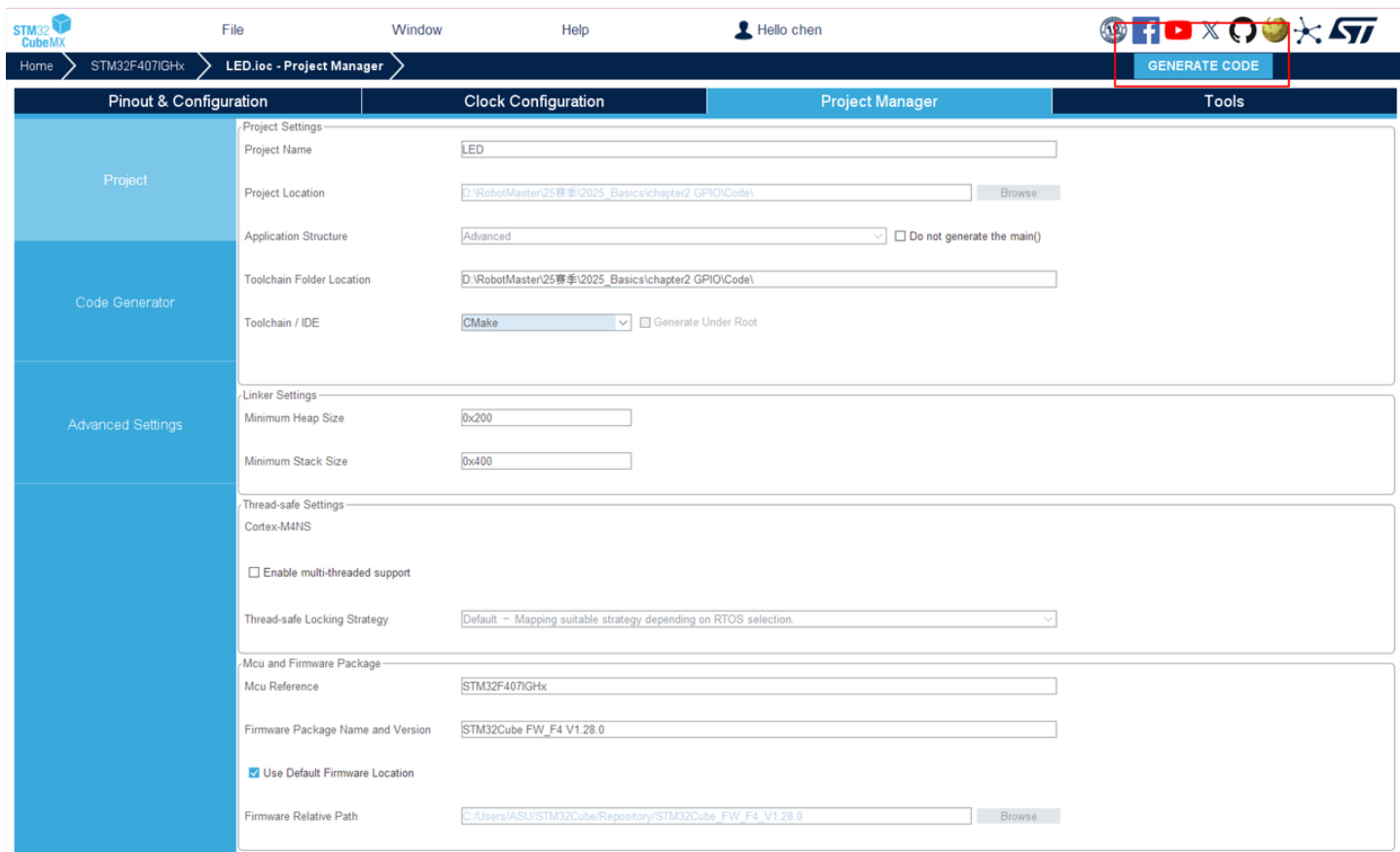


6. Project manager选项卡中，左侧的Project，自定义路径（最好为全英文路径），其余按照下图配置：



The screenshot shows the STM32CubeMX Project Manager interface. The 'Project Manager' tab is selected. The 'Project' section shows 'Project Name' as 'LED'. The 'Code Generator' section shows 'Toolchain / IDE' as 'CMake'. The 'Advanced Settings' section shows 'Linker Settings' with 'Minimum Heap Size' as '0x200' and 'Minimum Stack Size' as '0x400'. The 'Thread-safe Settings' section shows 'Thread-safe Locking Strategy' as 'Default'. The 'Mcu and Firmware Package' section shows 'Mcu Reference' as 'STM32F407IGHx' and 'Firmware Package Name and Version' as 'STM32Cube FW_F4 V1.28.0'.

7. 点击GENERATE CODE生成代码



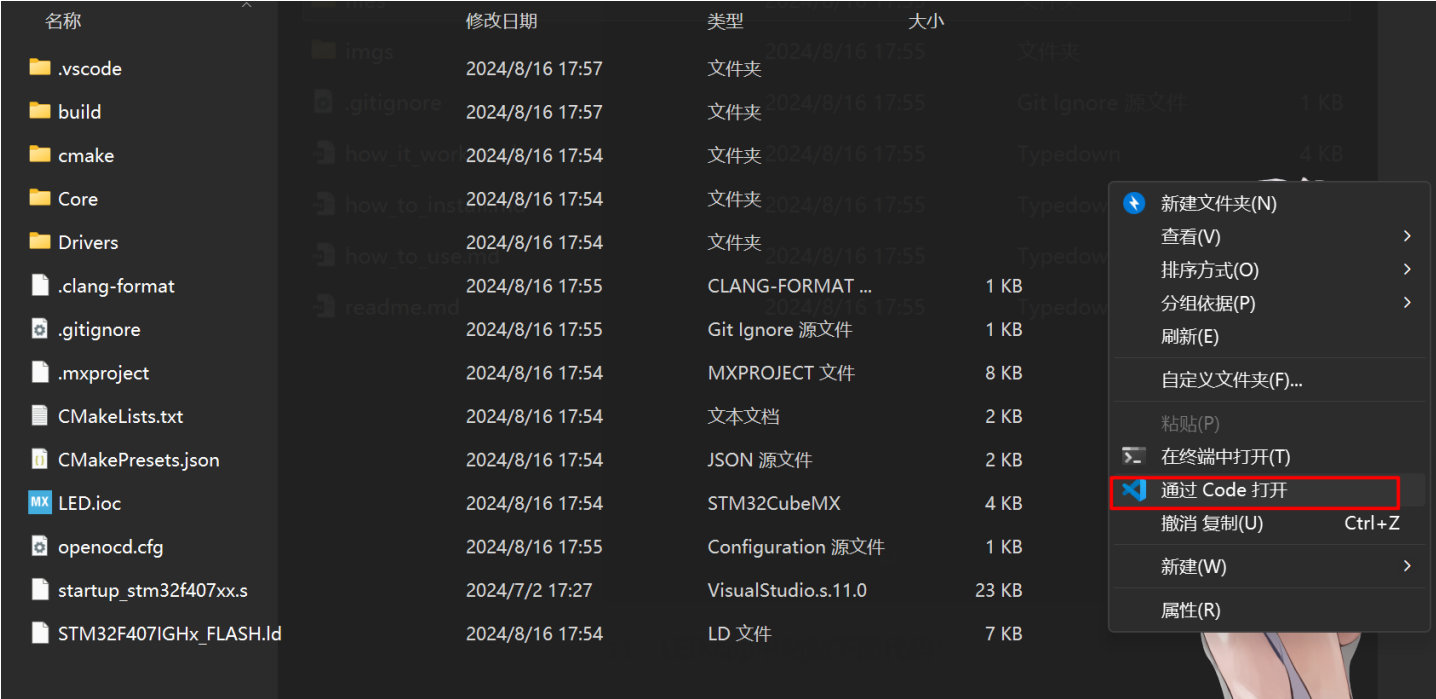
The screenshot shows the same STM32CubeMX Project Manager interface as above, but with the 'GENERATE CODE' button in the top right corner highlighted by a red box. The button is labeled 'GENERATE CODE'.

8. 将stm32_dev_env里file文件夹里的文件复制到CubeMX生成的文件夹中，并右键 通过Code打开

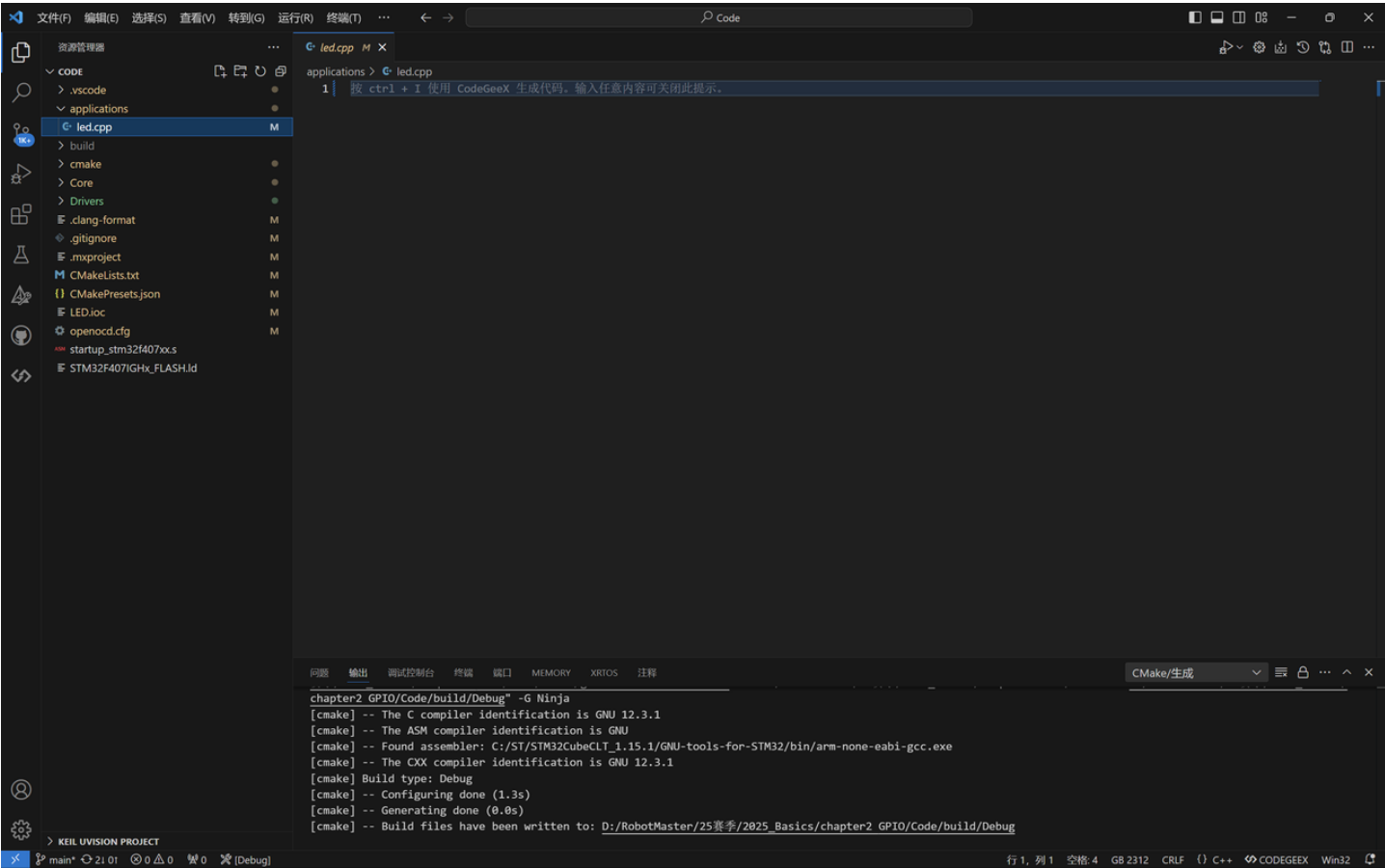
名称	修改日期	类型	大小
files	2024/8/16 17:55	文件夹	
imgs	2024/8/16 17:55	文件夹	
.gitignore	2024/8/16 17:55	Git Ignore 源文件	1 KB
how_it_works.md	2024/8/16 17:55	Typedown	4 KB
how_to_install.md	2024/8/16 17:55	Typedown	4 KB
how_to_use.md	2024/8/16 17:55	Typedown	2 KB
readme.md	2024/8/16 17:55	Typedown	1 KB

11. LED.cpp中粘贴下面代码：

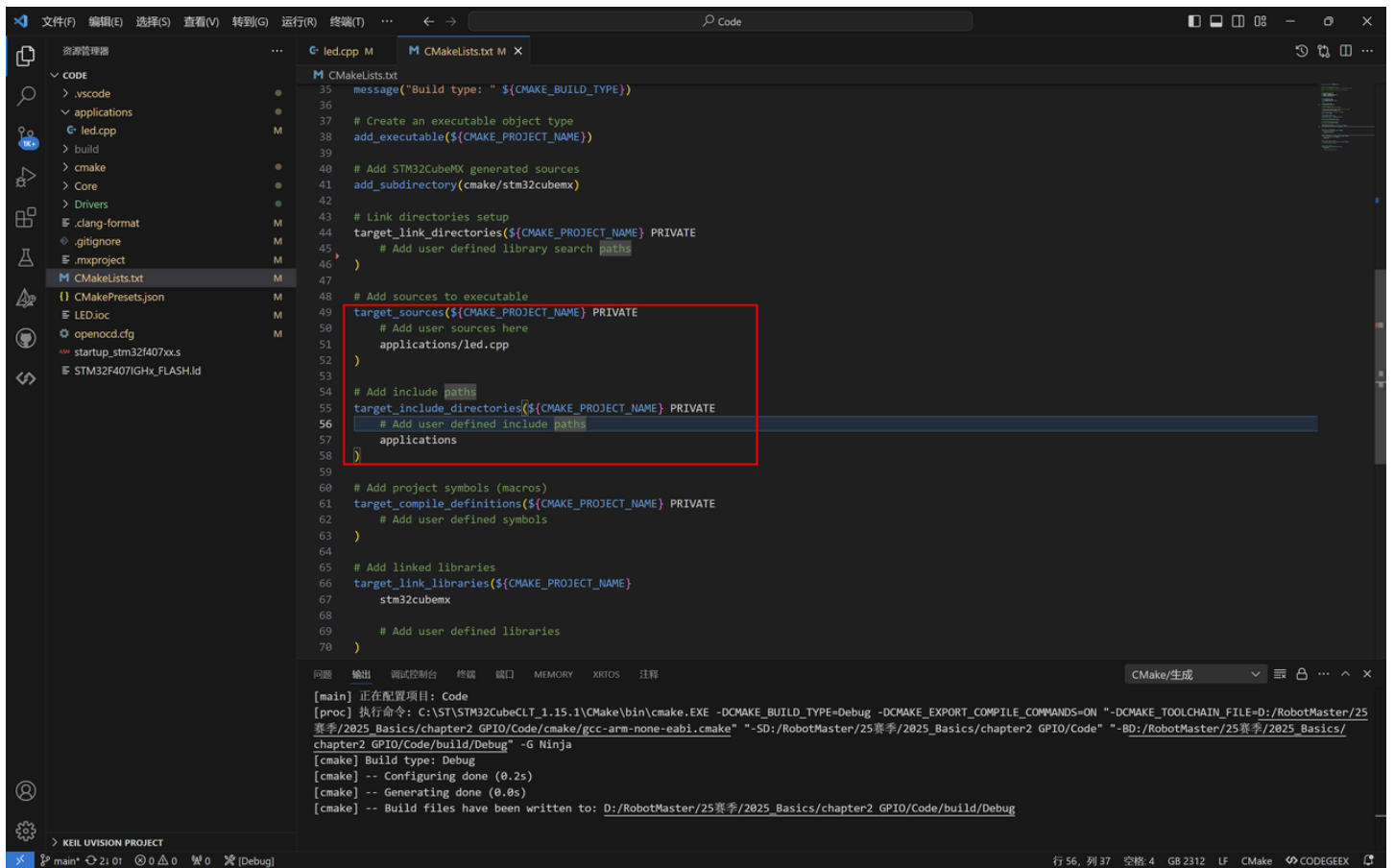
名称	修改日期	类型	大小
.vscode	2024/8/16 17:57	文件夹	
build	2024/8/16 17:57	文件夹	
cmake	2024/8/16 17:54	文件夹	
Core	2024/8/16 17:54	文件夹	
Drivers	2024/8/16 17:54	文件夹	
.clang-format	2024/8/16 17:55	CLANG-FORMAT ...	1 KB
.gitignore	2024/8/16 17:55	Git Ignore 源文件	1 KB
.mxproject	2024/8/16 17:54	MXPROJECT 文件	8 KB
CMakeLists.txt	2024/8/16 17:54	文本文档	2 KB
CMakePresets.json	2024/8/16 17:54	JSON 源文件	2 KB
LED.ioc	2024/8/16 17:54	STM32CubeMX	4 KB
openocd.cfg	2024/8/16 17:55	Configuration 源文件	1 KB
startup_stm32f407xx.s	2024/7/2 17:27	VisualStudio.s.11.0	23 KB
STM32F407IGHx_FLASH.ld	2024/8/16 17:54	LD 文件	7 KB



9. 在该项目中建立一个新的文件夹 application（当然也可以不建立，现在先听我的，学会了请自由发挥），并在其中新建一个led.cpp，如下图所示：

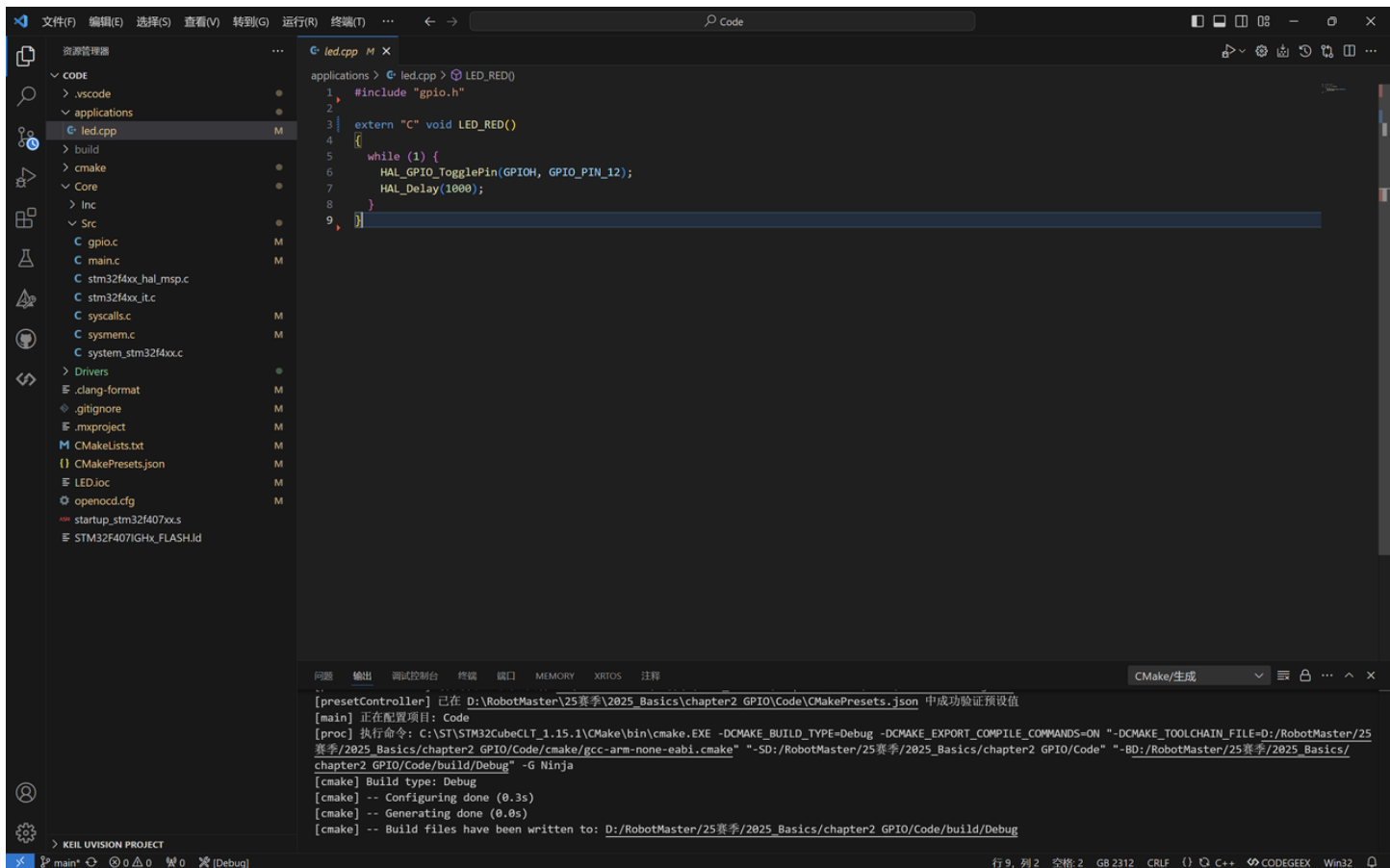


10. 在cmakelists.txt中添加两句，如下图所示：

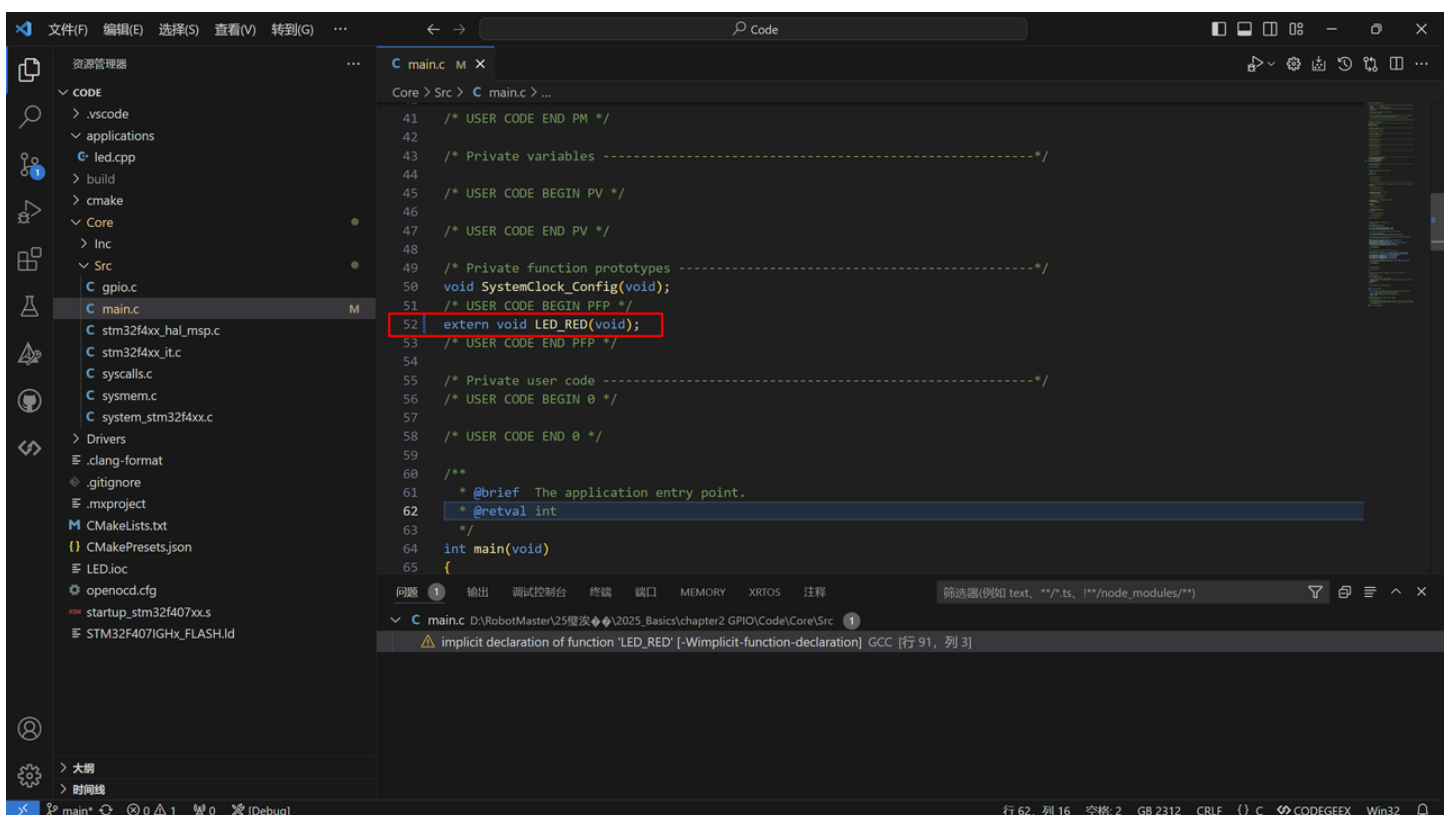


11. LED.cpp中粘贴下面代码：

```
1 #include "gpio.h"
2
3 extern "C" void LED_RED()
4 {
5     while (1) {
6         HAL_GPIO_TogglePin(GPIOH, GPIO_PIN_12);
7         HAL_Delay(1000);
8     }
9 }
```

12. main.c中添加函数声明



13. main.c中添加函数，如下图所示：


```
Core > Src > C main.c > main(void)
64 int main(void)
65 {
66     /* MCU Configuration ----- */
67
68     /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
69     HAL_Init();
70
71     /* USER CODE BEGIN Init */
72
73     /* USER CODE END Init */
74
75     /* Configure the system clock */
76     SystemClock_Config();
77
78     /* USER CODE BEGIN SysInit */
79
80     /* USER CODE END SysInit */
81
82     /* Initialize all configured peripherals */
83     MX_GPIO_Init();
84
85     /* USER CODE BEGIN 2 */
86     LED_RED();
87     /* USER CODE END 2 */
88
89     /* Infinite loop */
90     /* USER CODE BEGIN WHILE */
91     while (1)
92     {
93         /* USER CODE END WHILE */
94
95         /* USER CODE BEGIN 3 */
96
97         /* USER CODE END 3 */
98     }
99 }
100
```

注意！此处代码需要填写在 `/* USER CODE BEGIN 2 */` 与 `/* USER CODE END 2 */` 两句中间。

这中间的代码在修改cubemx配置并且重新生成后不会被修改，否则重新生成会被删除。

14. 编译并烧录（`F8` 或者没有添加快捷键的话按 `Ctrl+Shift+B` 选择 `OpenOCD: FLASH`）

有C板现象如下视频：



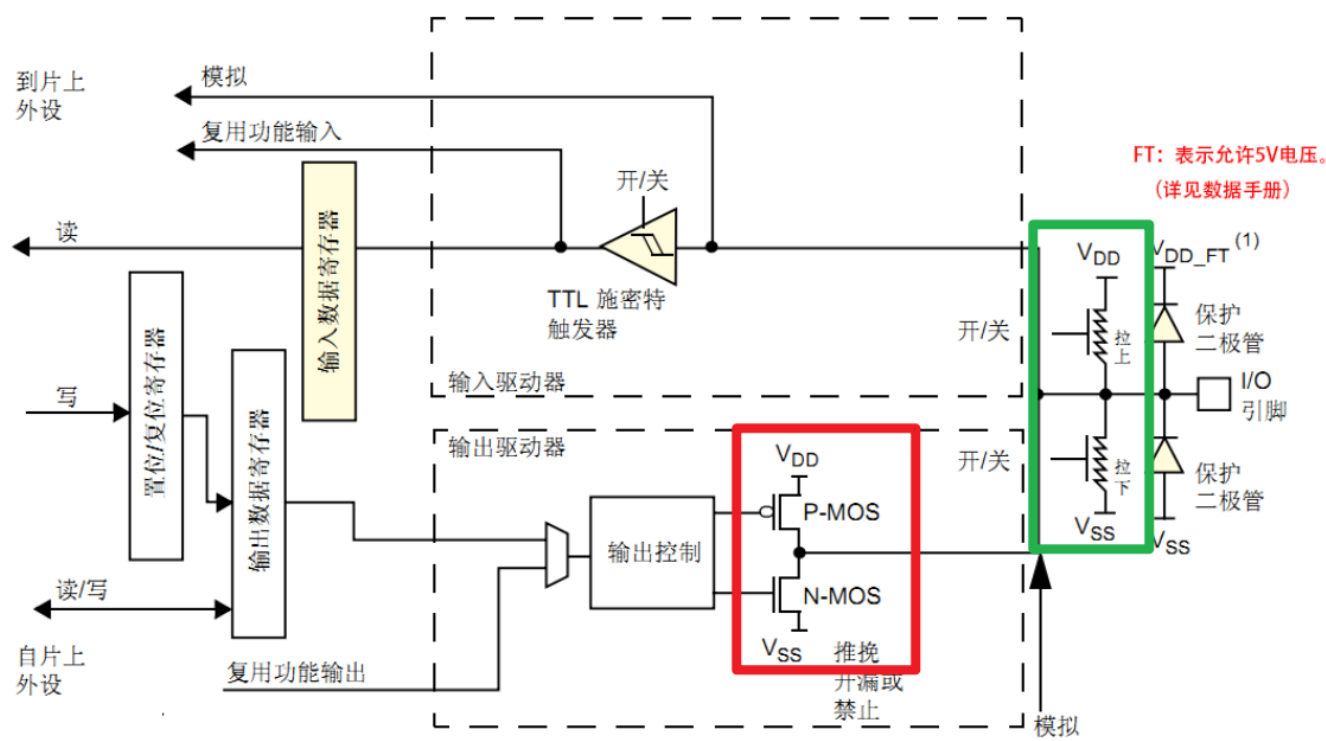
则成功。

详解

那么在上面的实验中究竟发生了什么使得红灯闪烁？下面来具体讲解。

下面是GPIO的原理图：

首先说明，原理如果不会也没关系，会配置就行了。



- 1. 上面步骤2中，复用的选项中有很多，其中与狭义GPIO相关的只有input、output以及Analog。
 - 2. 进入步骤4后，
 - a. 第一个选框中选择low，含义是在GPIO初始化时会将该GPIO的输出值拉到低电平，默认LED灯是不会亮的，该初始化函数位于gpio.h中。
 - b. 第二个选框中选择push pull，这个意思是GPIO的一种输出模式，叫做推挽输出，这种输出模式的特点是CPU写入一个高电平，引脚就输出一个高电平，同理写一个低电平就输出一个低电平。
- 另外还有一种输出模式叫做开漏输出（open drain），这种输出模式的特点是在CPU写入0时，则红色框处直接连接VSS，也就是接地，则输出会直接被拉到低电平，输出一定是低电平，然而在CPU写入1时，输出值取决于绿色框中的上拉与下拉电阻的配置情况。
- c. 第三个意为输出模式为上拉输出，也就是配置绿色框中的上拉以及下拉电阻，根据上面所说，当推挽输出时，输出值和电阻配置情况无关，因此这里就直接选没有上拉下拉即可。

但是如果上面选择了开漏输出，则这里如果不配置电阻会导致输出的电平浮动。因此需要配置上拉或者下拉电阻。

- d. 输出速度包括低速（2 MHz）、中速（25 MHz）、高速（50 MHz）和非常高速（100 MHz），低速都完全够用了。
 - e. 下面是用户label值，如果填了，会产生一个define宏定义，将pin口名字定义为你规定的那个。
3. 同理，在这个实验中我们配置的是GPIO的输出模式，如果我们想要使用一些例如光电传感器等外部的传感器，获取它们的返回值，那么需要用到GPIO的input模式，在这个模式下可以将外界输入读进CPU。

GPIO的input模式有四种，分别为上拉、下拉、浮空以及模拟。

其中上拉时，很显然，外界没有输入时，板子会从绿色框的VDD处获得一个高电平，输入高电平时自然为高电平，输入低电平时为低电平。

下拉时同理，外界没有输入会读到VSS，也就是低电平，其余输入高读高、输入低读低。

浮空时，则读到的值完全取决于外界的输出，但是在某些光电传感器的使用说明中明确要求有电阻存在，此时应当合理配置上拉或者下拉电阻，而不是用浮空以避免电平浮动。

模拟时，则通过ADC转化为数字量读入。

4. 不妨看一下gpio.c里面的这个初始化函数：

```
void MX_GPIO_Init(void)
{
    GPIO_InitTypeDef GPIO_InitStruct = {0};

    /* GPIO Ports Clock Enable */
    __HAL_RCC_GPIOH_CLK_ENABLE();

    /*Configure GPIO pin Output Level */
    HAL_GPIO_WritePin(GPIOH, GPIO_PIN_12, GPIO_PIN_SET);

    /*Configure GPIO pin : PH12 */
    GPIO_InitStruct.Pin = GPIO_PIN_12;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_NOPULL;
    GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
    HAL_GPIO_Init(GPIOH, &GPIO_InitStruct);
}
```

这个函数在main.c中也进行了调用，在这个函数中进行了一些配置，根据其意思很容易发现其实就是在cubeMX里进行的配置。同时最下面这个init函数很重要，没有它GPIO就不能使能（术语，开始工作的意思）。

另外，在LED.cpp中有一个反转函数，是在stm32HAL库函数中预先定义的，能够使电平输出反转，加上延时后就每一秒转换一次。

