STM32 开发实验指导书

更新日志

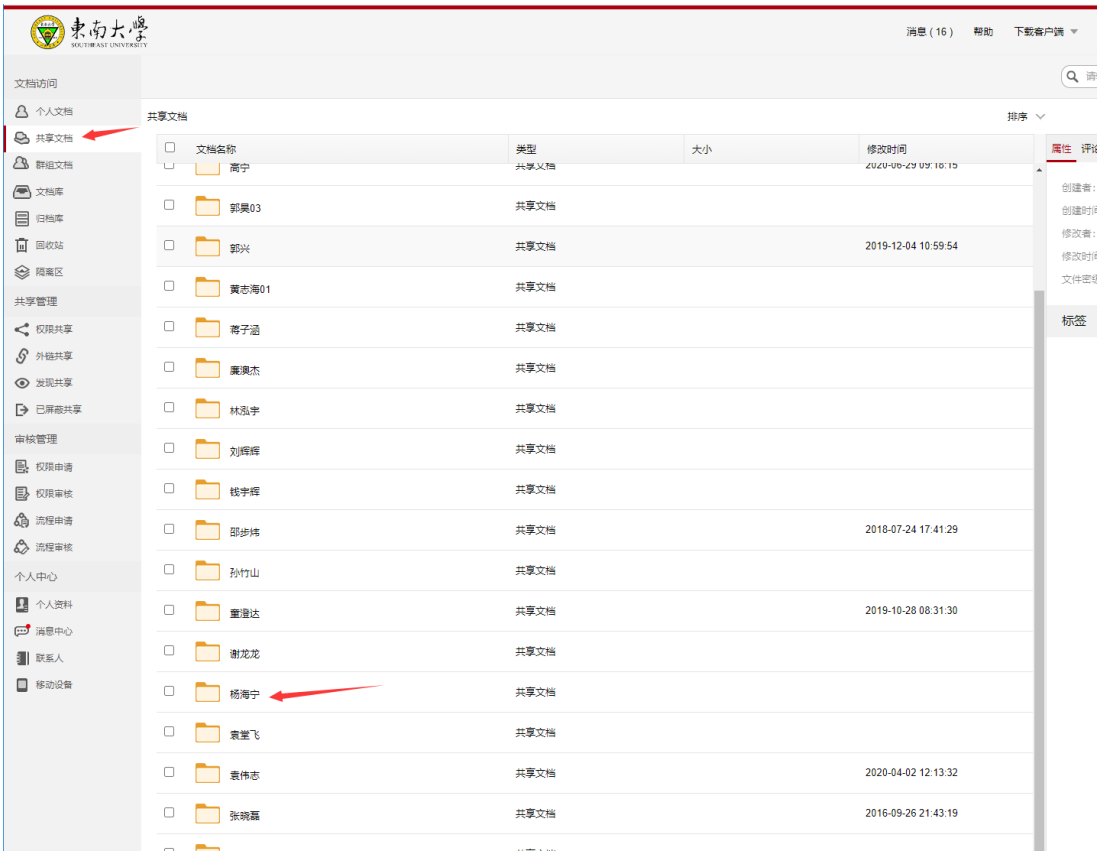
时间	更新内容	更新人
2020.09.08	完成初始版本	Disp301
2020.09.16	1. 补充第一章第三节 IKS01A3 拓展板支持包安装方法； 2. 补充第九章 PWM 输出实验； 3. 补充第十章 DAC 实验； 4. 补充第十一章 STTS751 温度获取实验；	Disp301
2020.10.10	1. 补充第十一章 ADC 实验 2. 补充第十三章 LSM6DSO 六轴加速度获取实验 3. 补充第十四章 FreeRTOS 实验 5. 修改第四章跑马灯实验中 LED 电平翻转部分代码 6. 修改第五章按键输入实验中按键读取部分代码 7. 修改第九章 PWM 输出实验中部分实验说明 8. 修改第十二章 STTS751 温度获取实验中拓展板初始化操作	Disp301
2021.08.05	部分文字修正与软件版本更新	Disp301

目录

第一章 软件安装

1.1 软件下载

1. 打开 https://pan.seu.edu.cn，用信息门户账号密码登录东南大学文档云，在**共享文档**中找到名为**杨海宁**的文件夹。



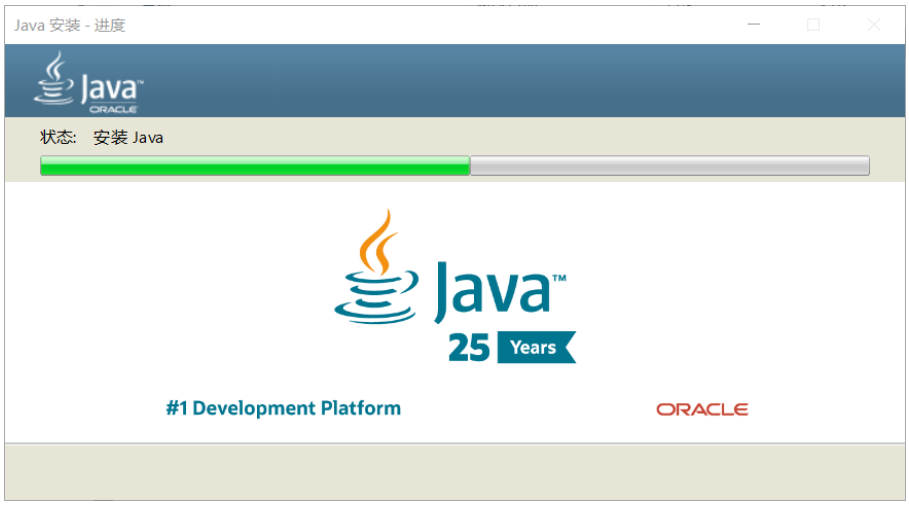
2. 进入 STM32 软件文件夹，下载固件包、软件内的文件到本地。如果出现无法下载的问题，可以下载东南大学文档云客户端，在客户端内下载文件。

1.2 安装 JAVA 开发环境

1. 运行 jre-8u261-windows-x64.exe

2. 直接点击安装

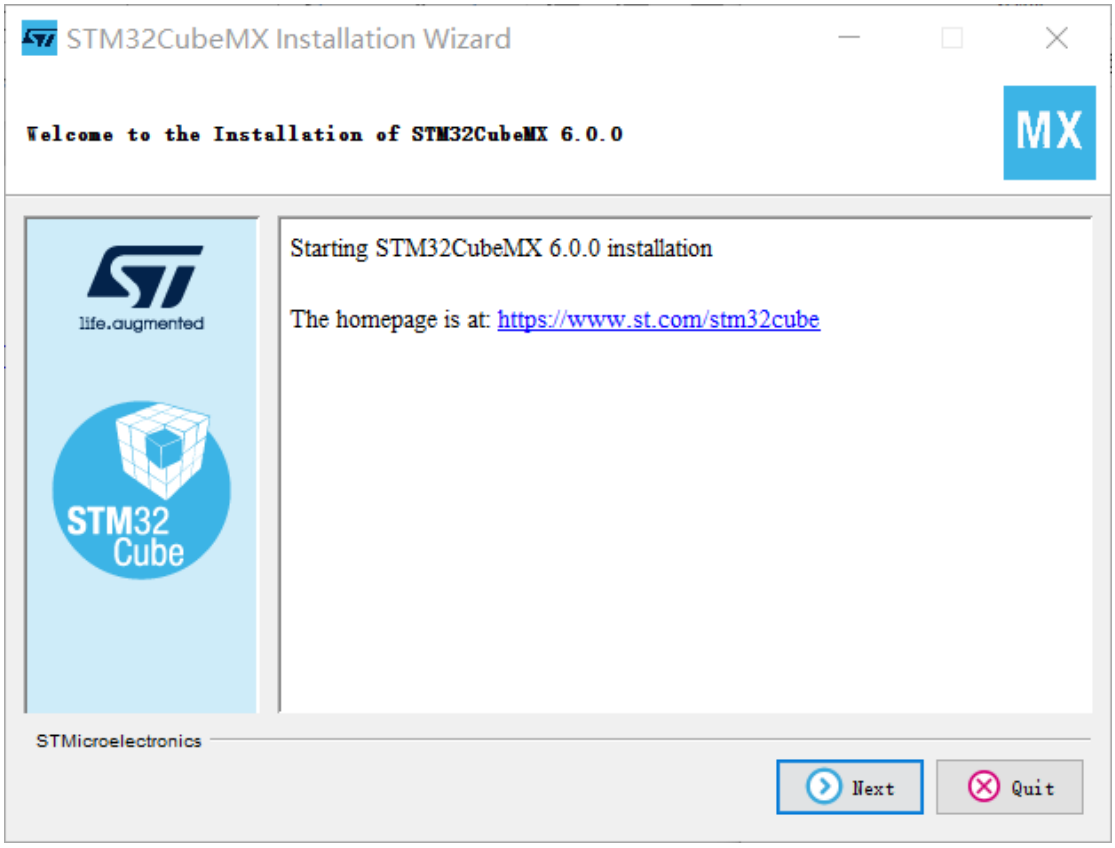
3. 出现以下界面，等待一会儿即可安装成功



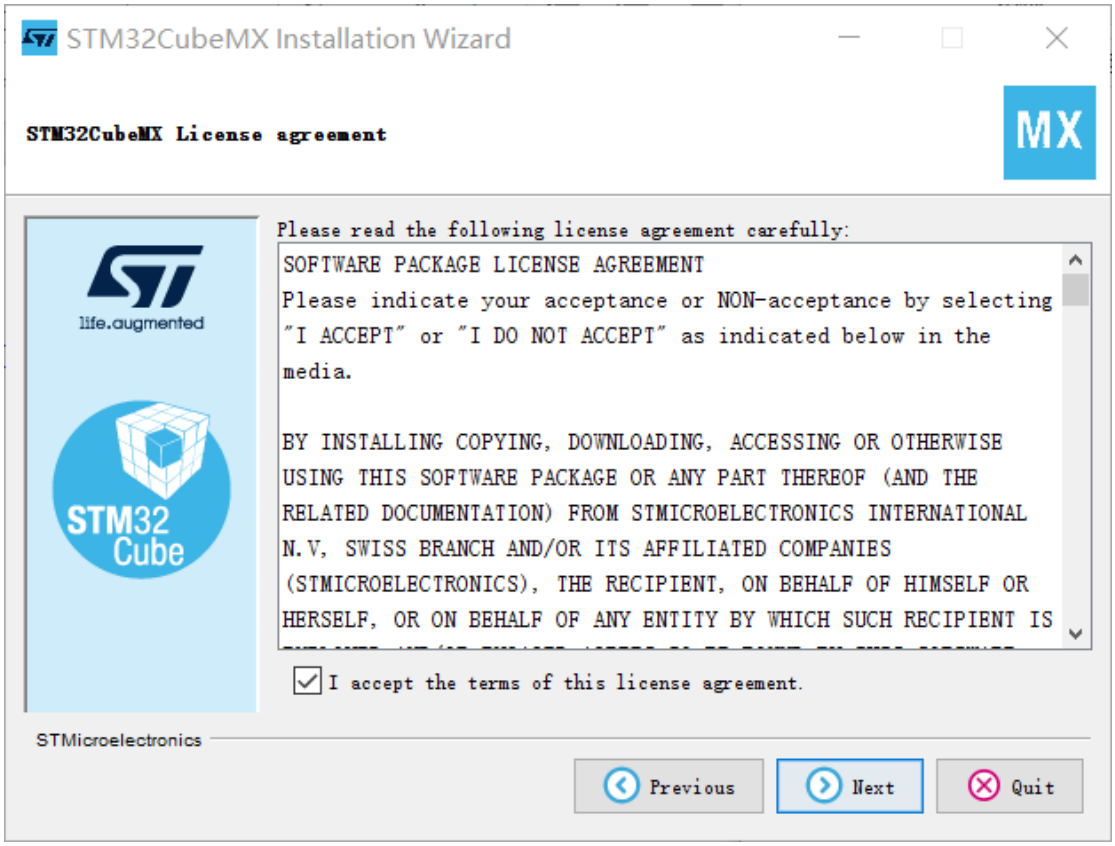
1.3 安装 STM32CubeMX

1. 运行 SetupSTM32CubeMX-6.0.0.exe

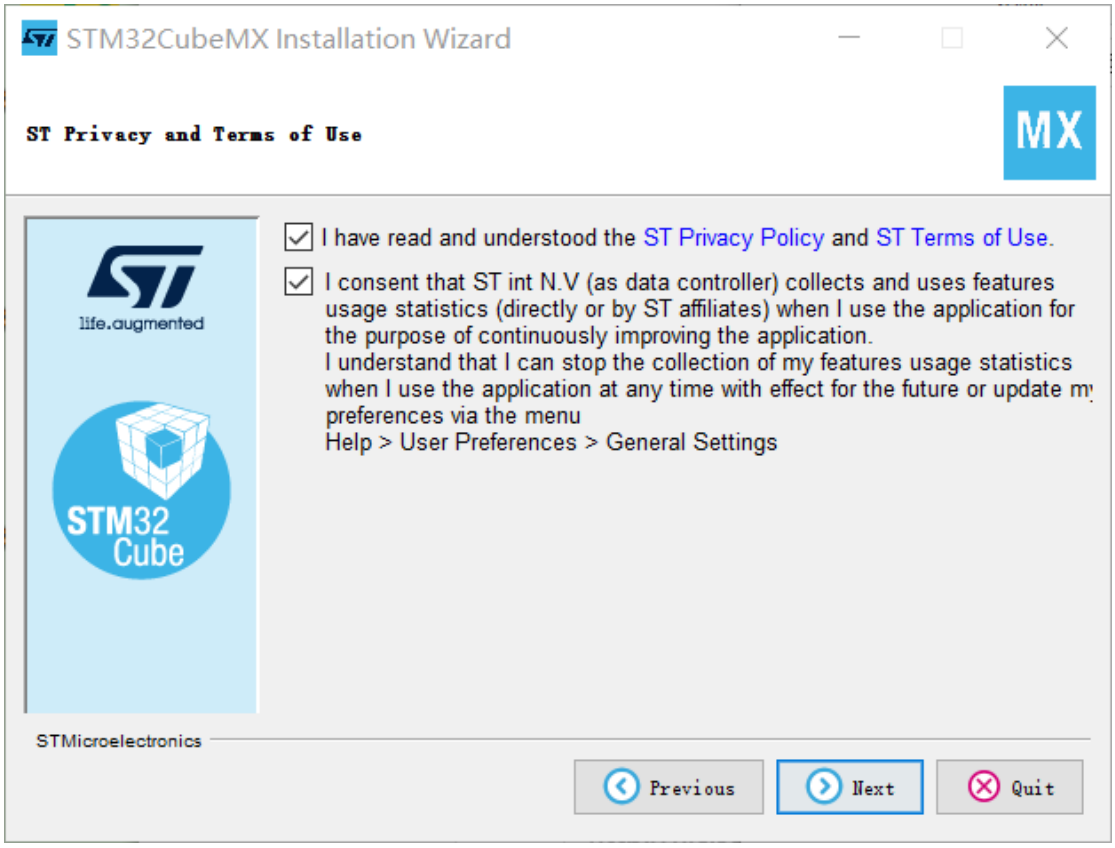
2. 点击 next->next



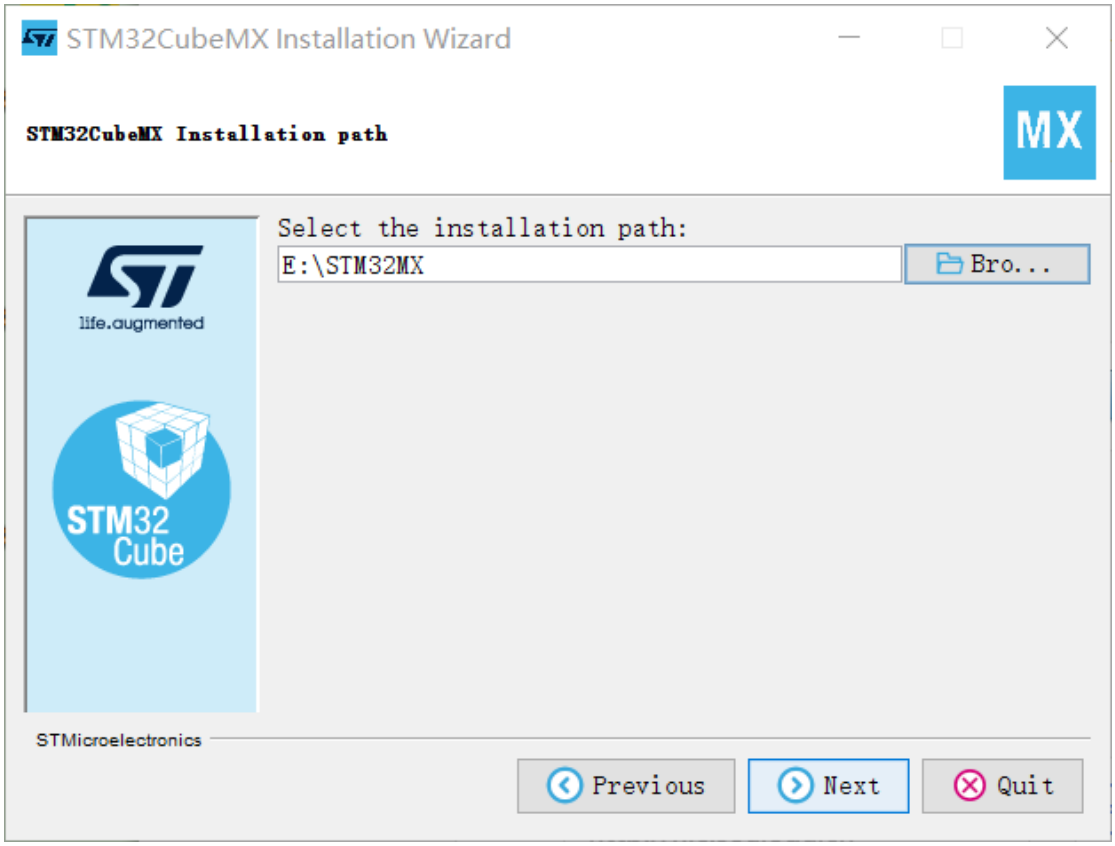
3. 勾选后点击 next



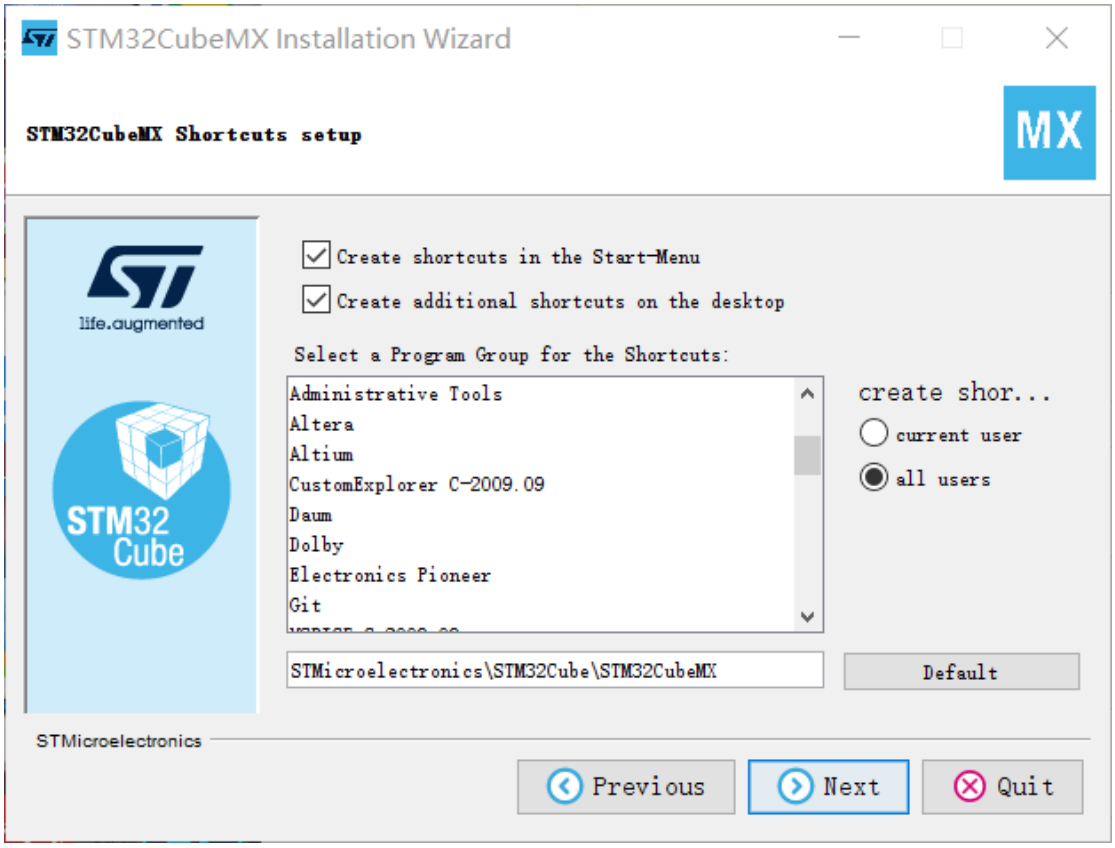
4. 勾选后点击 next



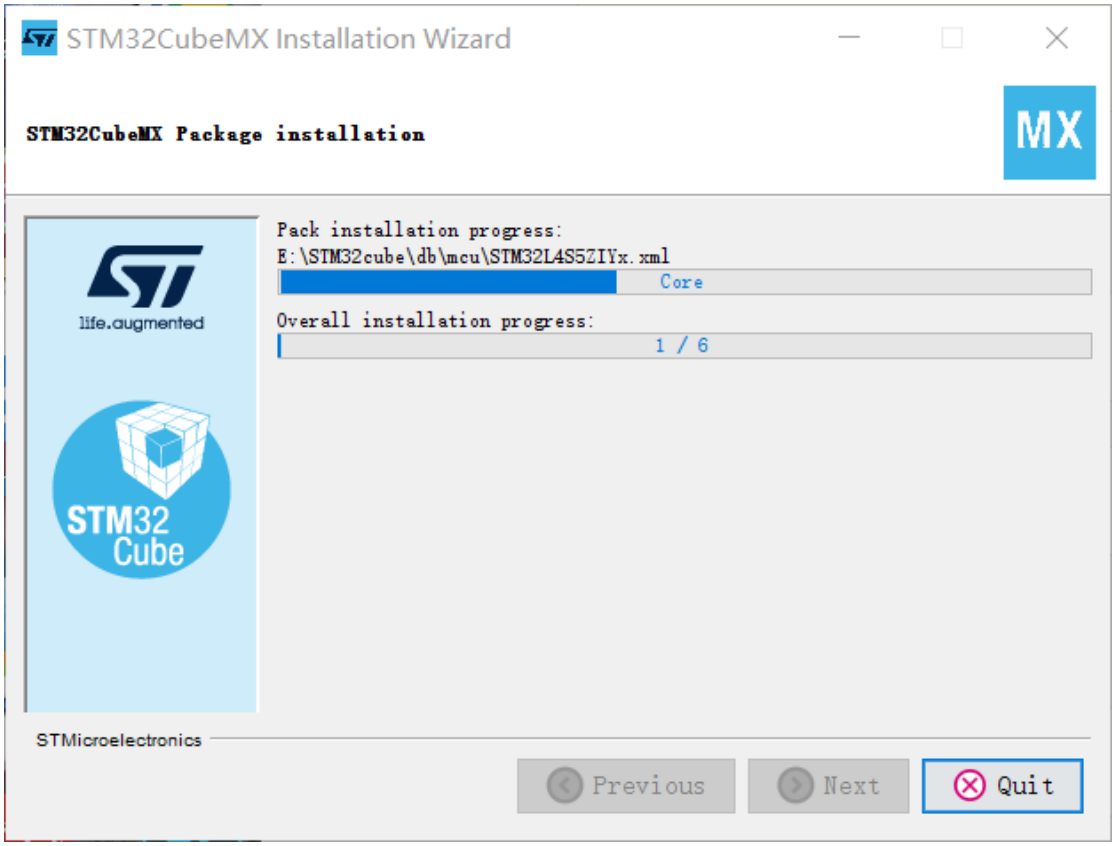
5. 建议不改变默认路径，如果需要可以修改路径，注意**安装路径不能包含中文**



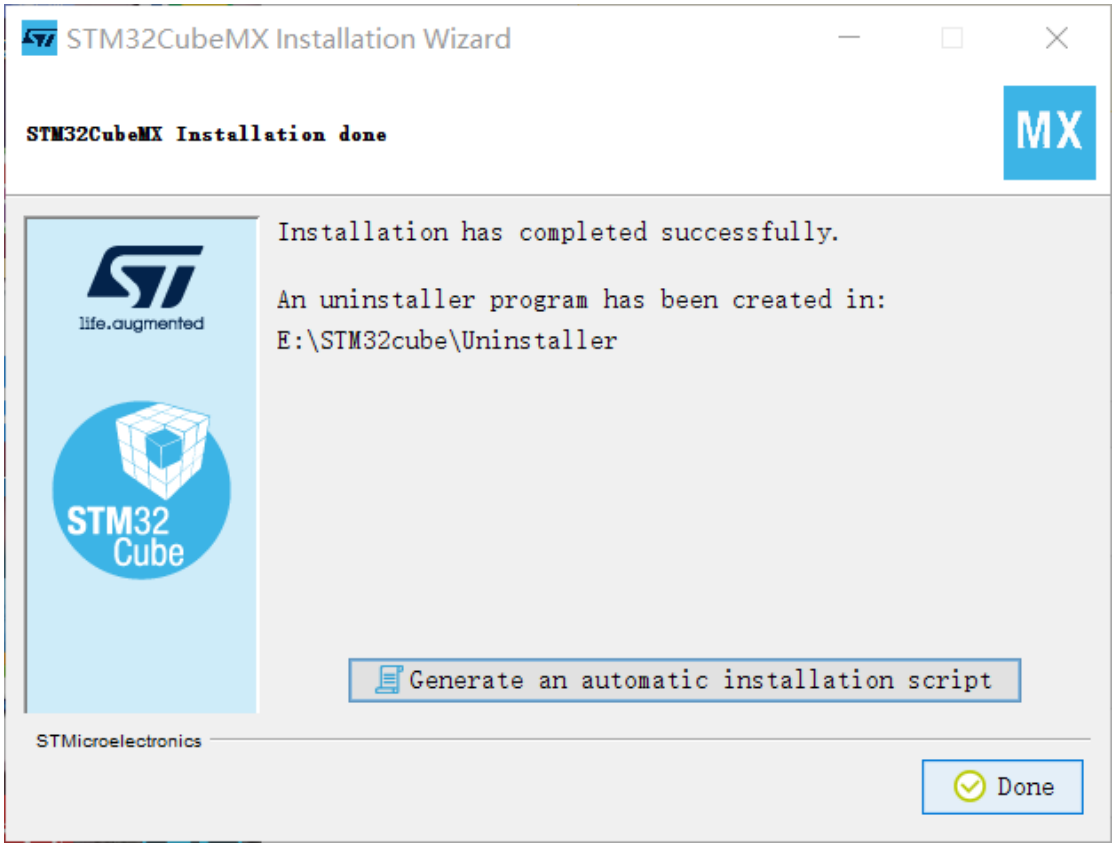
6. 点击 next



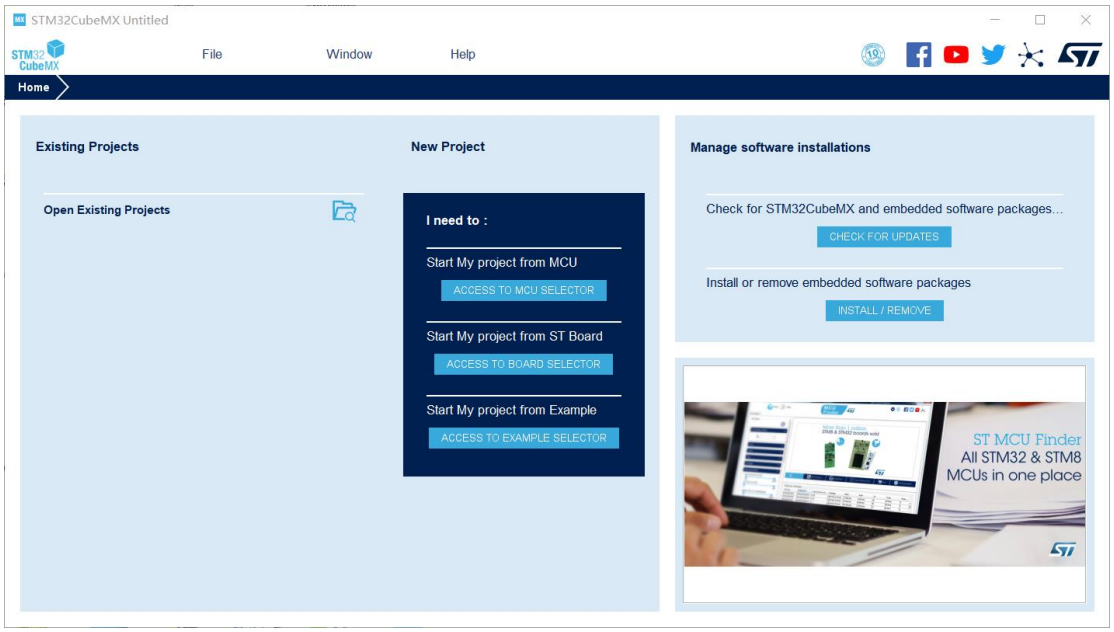
7. 等待安装结束即可



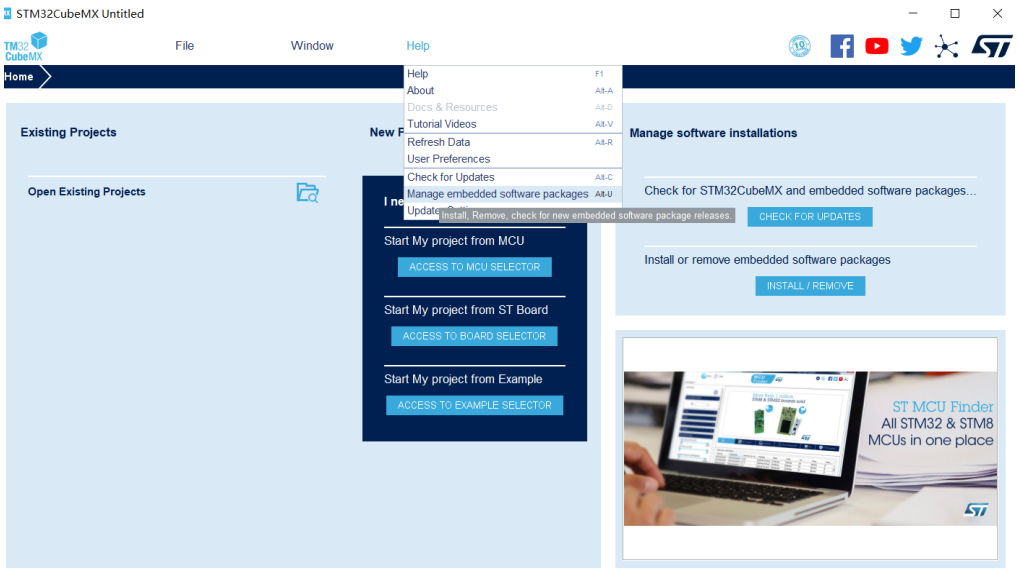
8. 安装成功



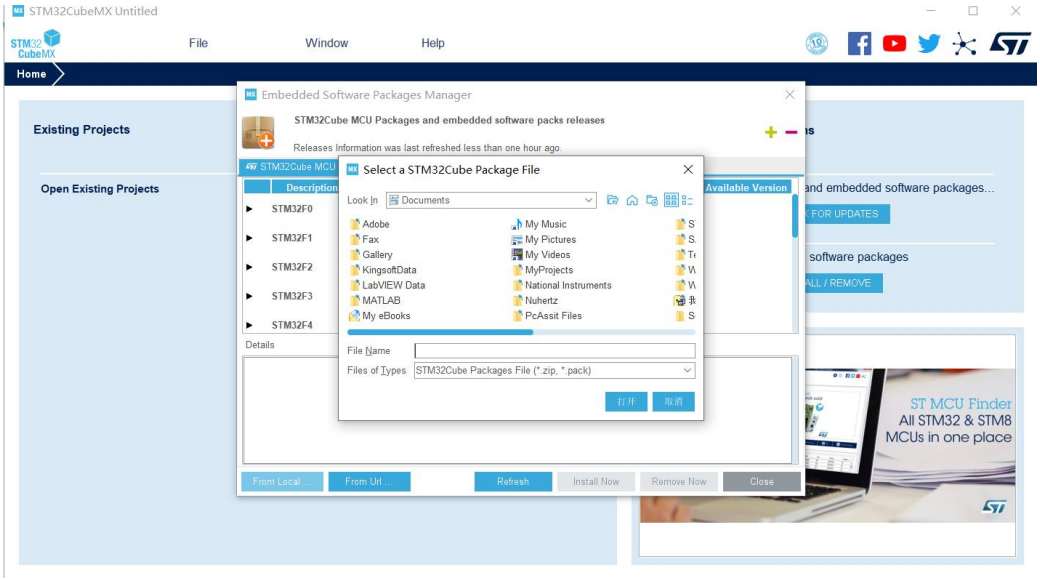
9. 下面需要添加固件包，打开软件



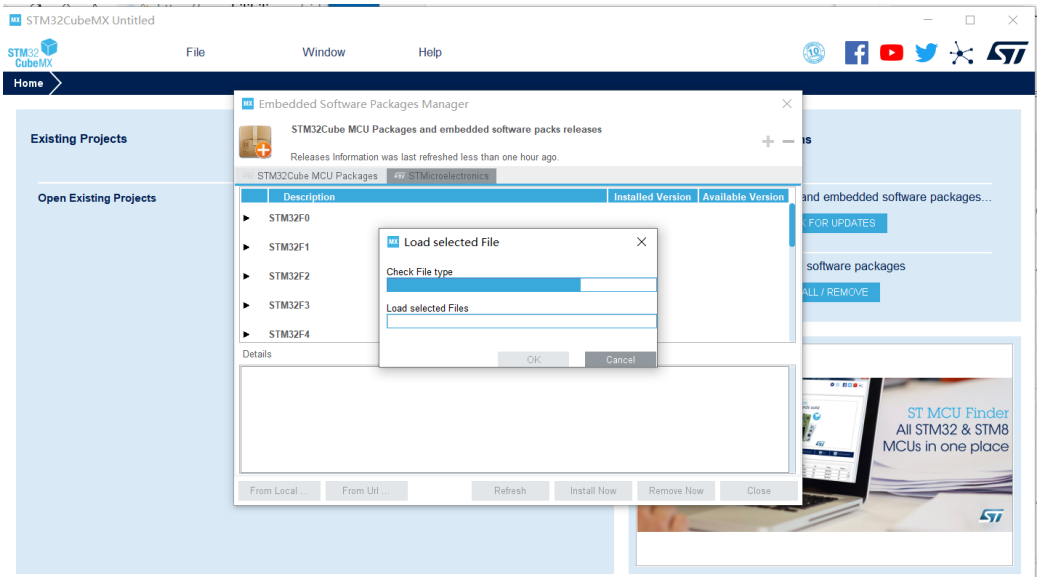
10. 点击 help->Manage embedded software packages



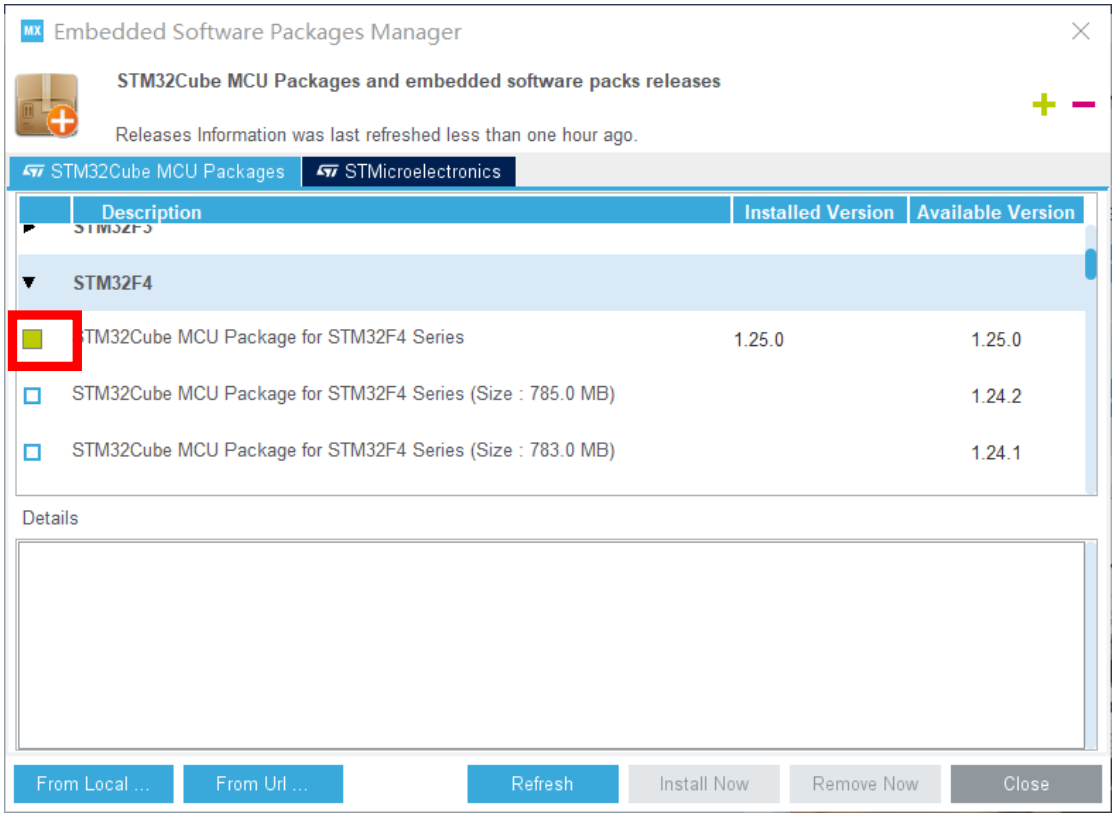
11.点击左下角 from local...找到 stm32cubef4_V1.25.0.zip，点击打开



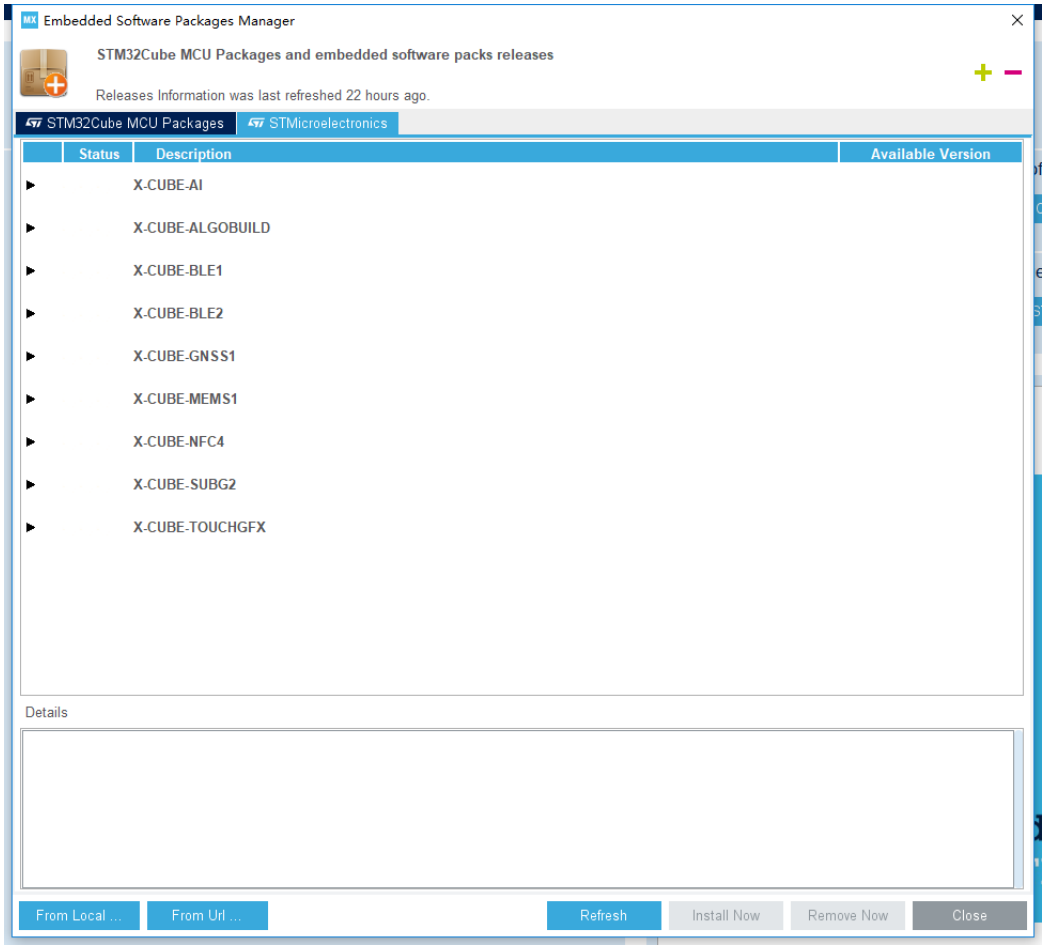
12.正在安装固件包



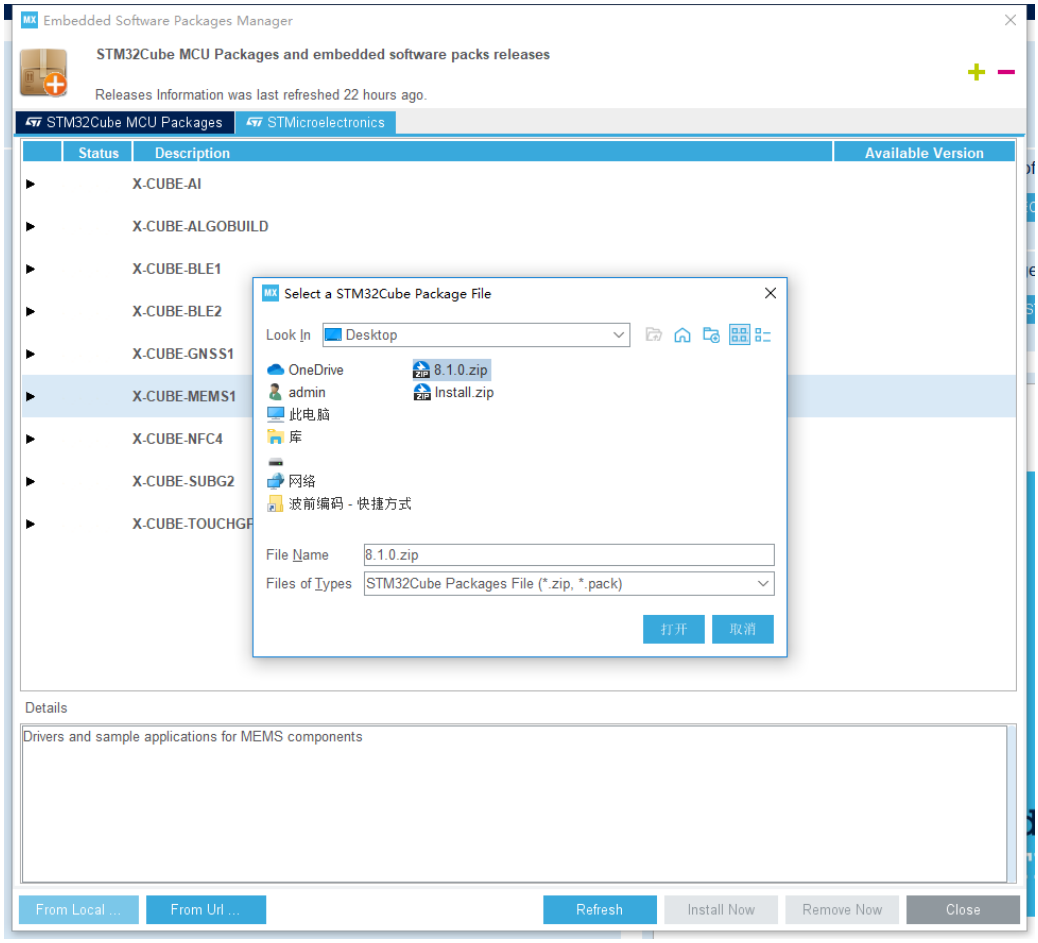
13.安装成功后，在 STM32F4 列表下 1.25.0 那一行的方框变绿



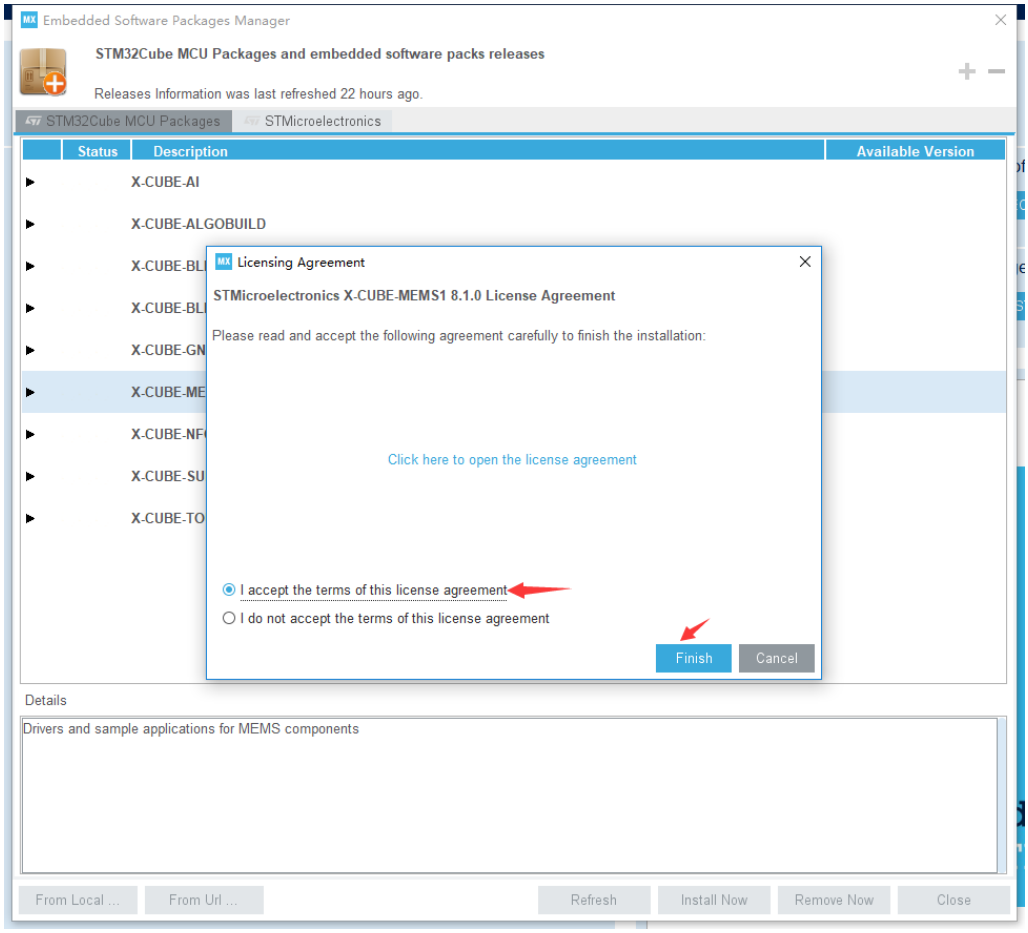
14.点击第二个选项卡 STMicroelectronics



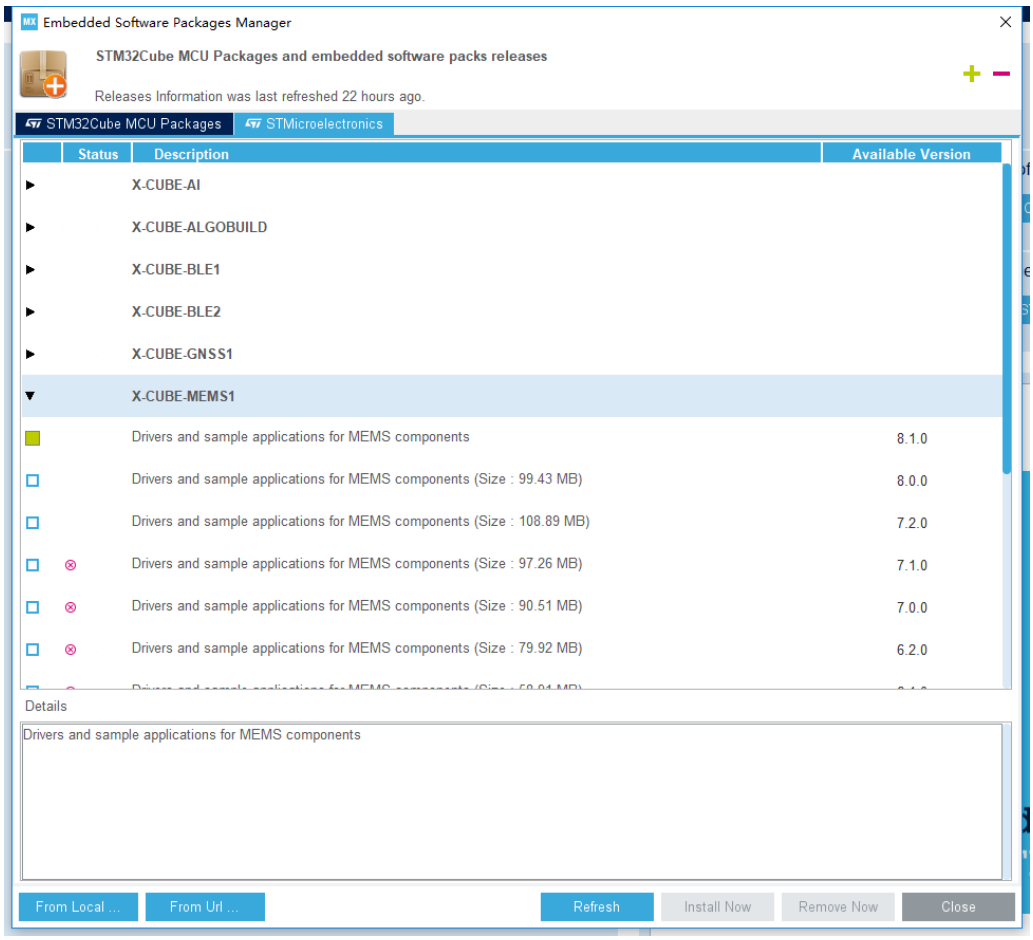
15.点击左下角 from local...找到 **8.1.0**.zip，点击打开



16. 点击接受，并点击结束



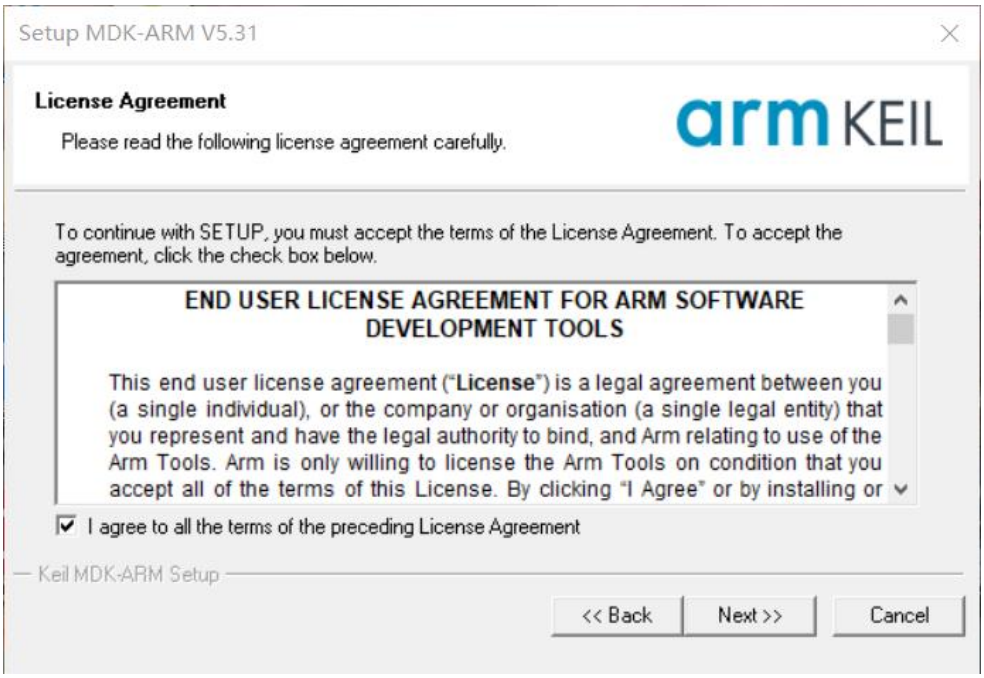
17.点击 X-CUBE-MEMS1，若 8.1.0 版本的拓展板支持变绿，则说明安装成功。



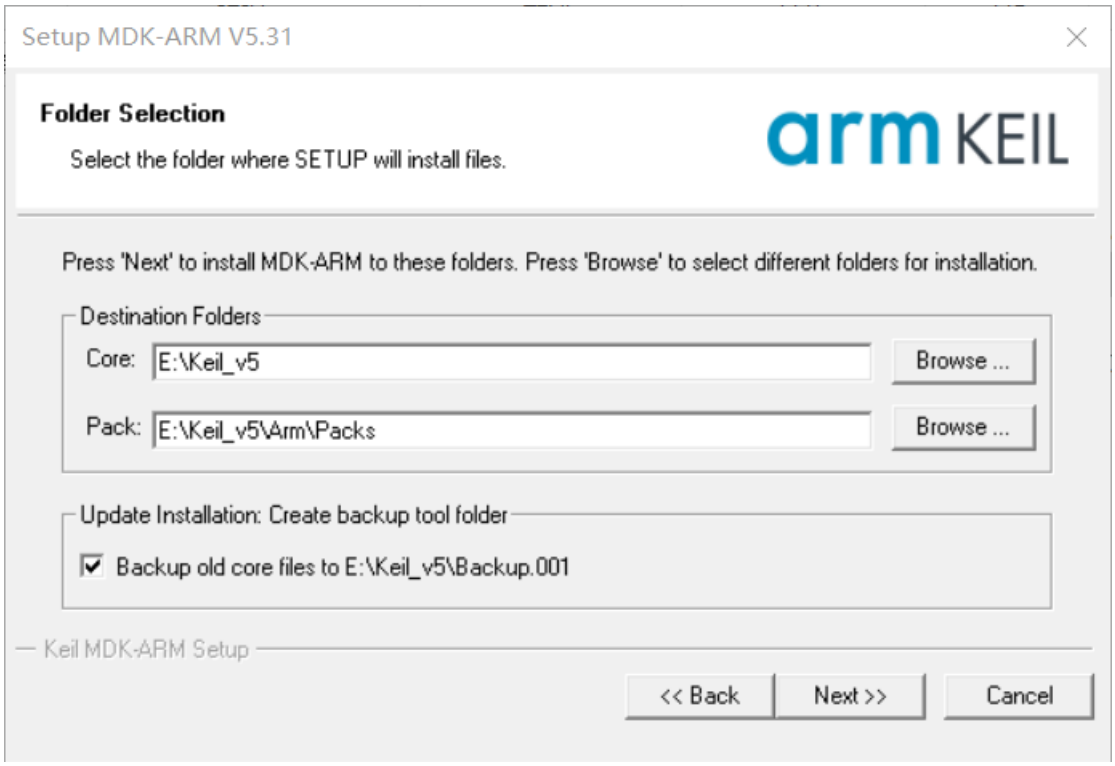
1.4 安装 MDK-ARM

1. 运行 MDK531.exe

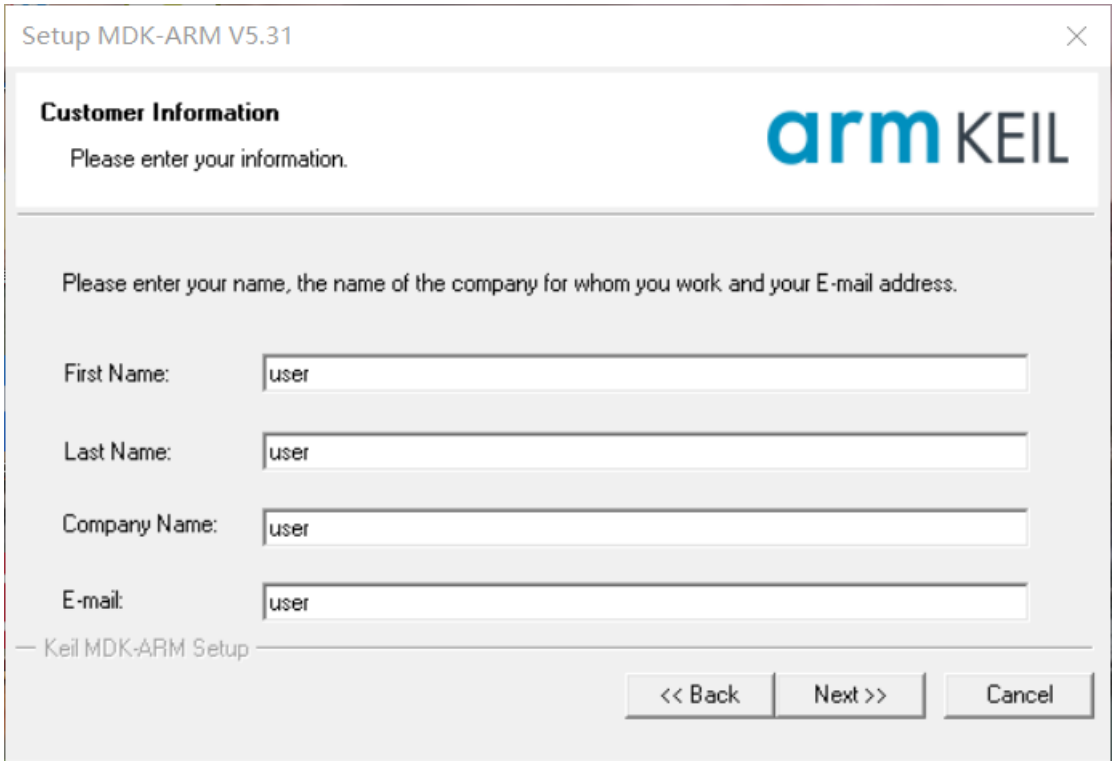
2. 点击 next->next



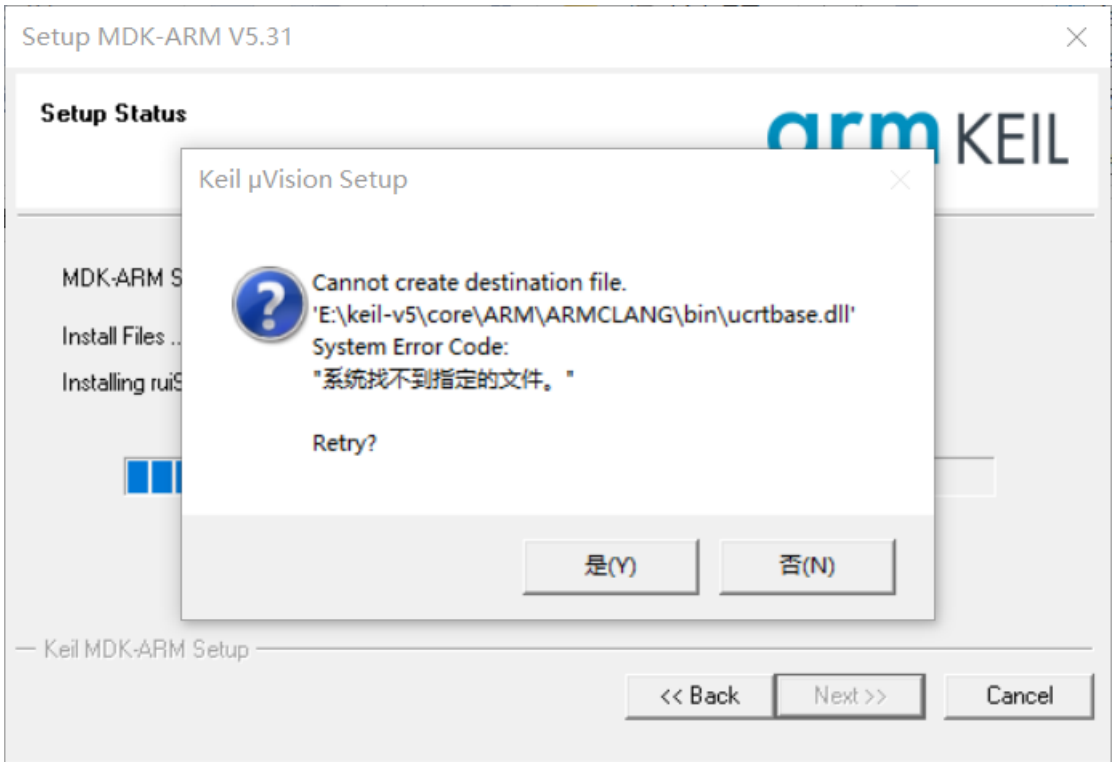
3. 推荐使用默认路径安装，也可以使用自己的路径，建议 packs 的文件位置包含在 core 的文件里面。



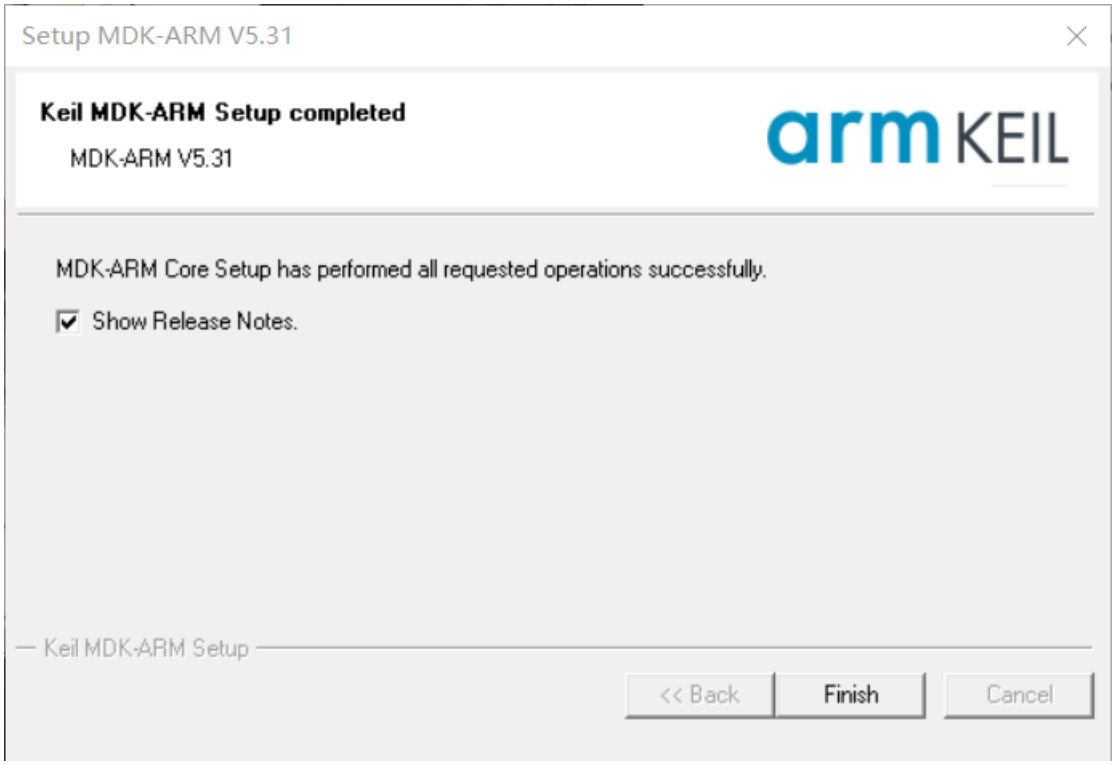
4. 随便填写用户信息，但要注意必须是英文



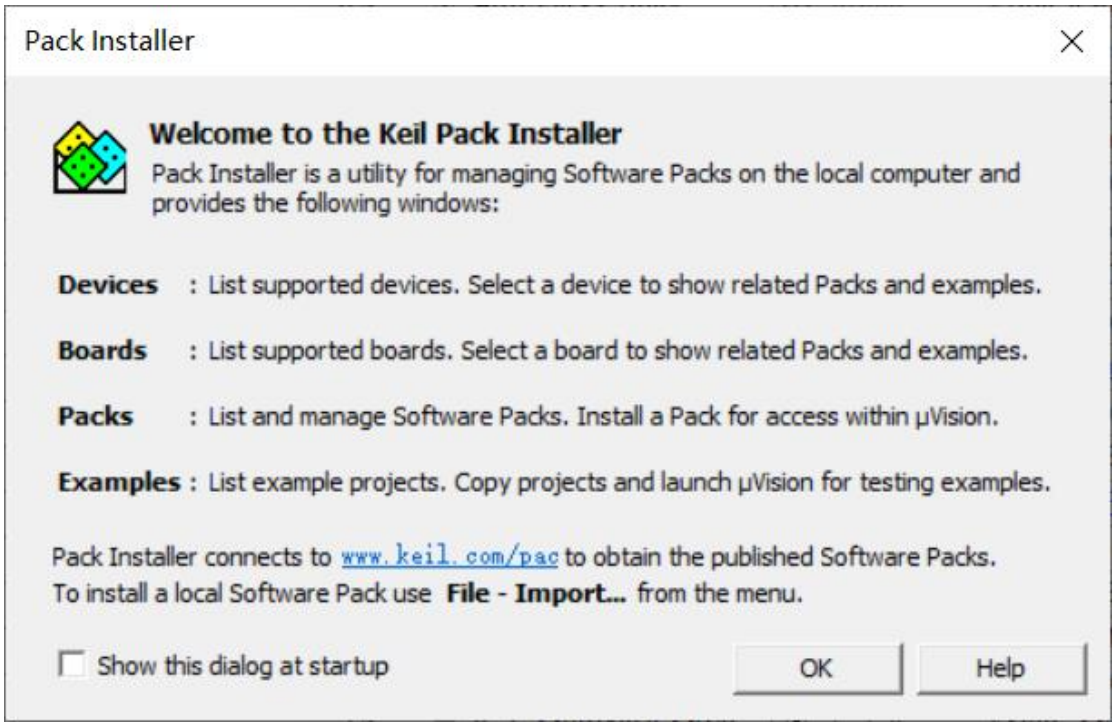
5. 注意：安装过程中如果防火墙拦截程序操作，一定要选“允许程序所有操作”；如果出现如下图所示的“系统找不到指定的文件”的错误提示，可以下载 ccleaner 清理注册表，之后重新下载.exe 文件或重新解压压缩包，用新的压缩包内的.exe 文件，回到步骤 1 开始操作。



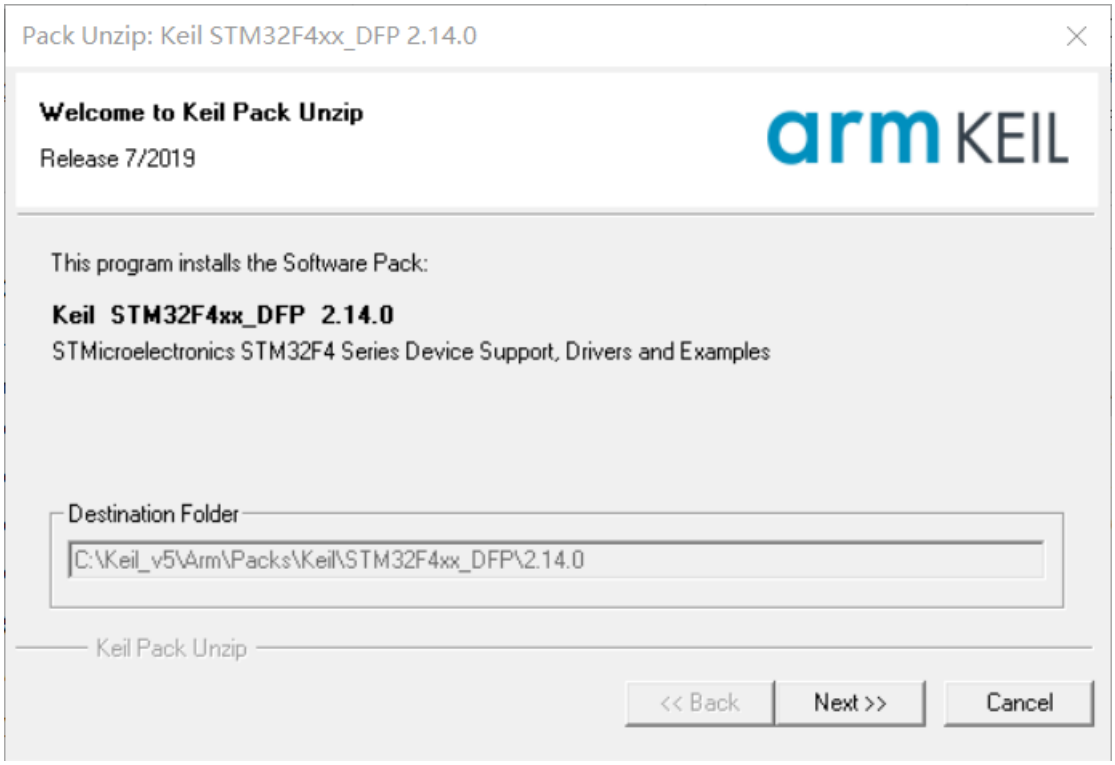
6. 安装成功，取消勾选，点击 finish



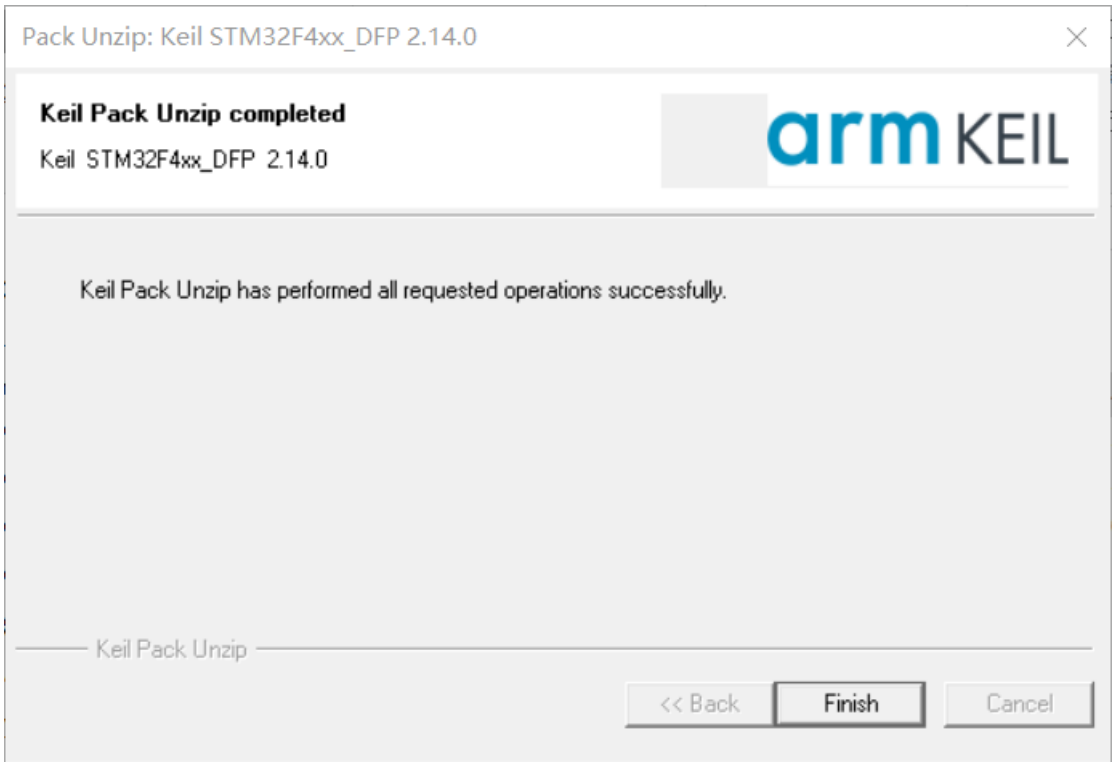
7. 自动跳出下载固件包的窗口，先取消勾选，然后依次关闭两个窗口。



8. 双击 **Keil.STM32F4xx_DFP.2.14.0.pack**，点击 next，安装固件包。



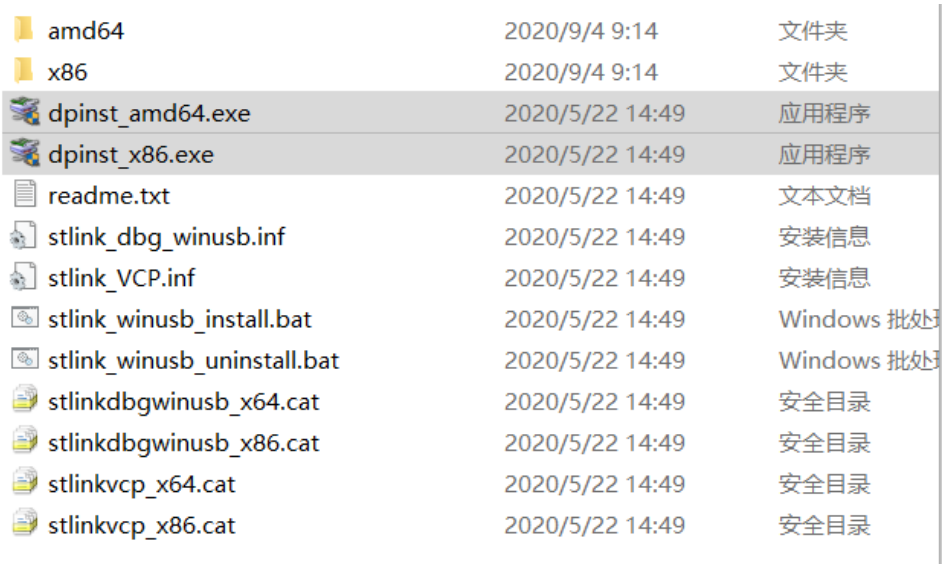
9. 安装成功



1.5 ST-Link 驱动安装及固件升级

1. 找到 MDK 软件安装路径，定位到 xxxx\Keil_v5\ARM\STLink\USBDriver

2. 根据自己的电脑操作系统选择 64 位或 32 位的驱动程序（一般都选用 64 位的），双击运行.exe 程序



amd64	2020/9/4 9:14	文件夹
x86	2020/9/4 9:14	文件夹
dpinst_amd64.exe	2020/5/22 14:49	应用程序
dpinst_x86.exe	2020/5/22 14:49	应用程序
readme.txt	2020/5/22 14:49	文本文档
stlink_dbg_winusb.inf	2020/5/22 14:49	安装信息
stlink_VCP.inf	2020/5/22 14:49	安装信息
stlink_winusb_install.bat	2020/5/22 14:49	Windows 批处
stlink_winusb_uninstall.bat	2020/5/22 14:49	Windows 批处
stlinkdbgwinusb_x64.cat	2020/5/22 14:49	安全目录
stlinkdbgwinusb_x86.cat	2020/5/22 14:49	安全目录
stlinkvcp_x64.cat	2020/5/22 14:49	安全目录
stlinkvcp_x86.cat	2020/5/22 14:49	安全目录

3. 确认安装成功与否：将 Nucleo 开发板与电脑连接（**必须确保 Micro USB 数据线可以传输数据，连接后红色 LD4 指示灯应当常亮，如果闪烁则应更换数据线**）->右击“我的电脑”->属性->设备管理器，可以看到通用串行总线设备中有 ST-Link 的驱动，端口中有 ST-Link 的虚拟端口，其中端口号随意。

4. 确保 ST-Link 仿真器与电脑相连后，定位到 xxxx\Keil_v5\ARM\STLink，双击运行 ST-LinkUpgrade.exe

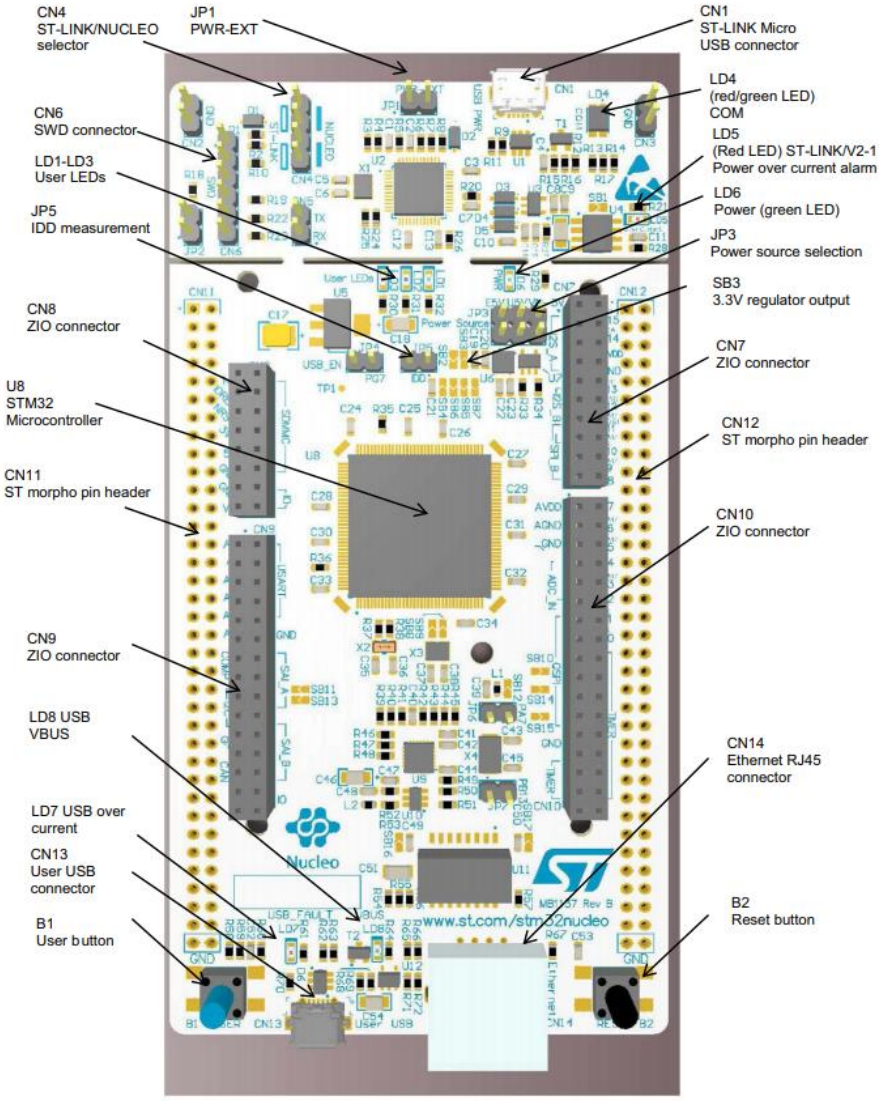
5. 点击 **Device Connect** ，连接成功，显示仿真器固件库版本和仿真器类型

6. 点击 **Yes>>>>**，升级成功

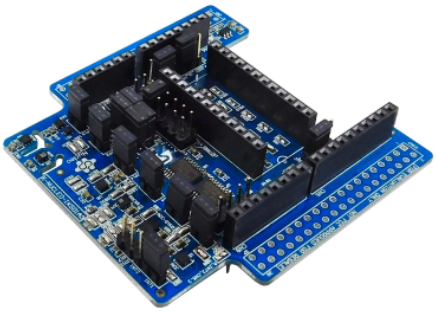
7. 最终检测板子是否正常工作：多次按下蓝色 B1 按键，用户指示灯变化

第二章 硬件速览

2. 1 Nucleo-144



2. 2 IKS01A3



Features

- LSM6DSO: MEMS 3D accelerometer (±2/±4/±8/±16 g) + 3D gyroscope (±125/±250/±500/±1000/±2000 dps)
- LIS2MDL: MEMS 3D magnetometer (±50 gauss)
- LIS2DW12: MEMS 3D accelerometer (±2/±4/±8/±16 g)
- LPS22HH: MEMS pressure sensor, 260-1260 hPa absolute digital output barometer
- HTS221: capacitive digital relative humidity and temperature
- STTS751: Temperature sensor (–40 °C to +125 °C)
- DIL 24-pin socket available for additional MEMS adapters and other sensors
- Free comprehensive development firmware library and example for all sensors compatible with STM32Cube firmware
- I²C sensor hub features on LSM6DSO available
- Compatible with STM32 Nucleo boards
- Equipped with Arduino UNO R3 connector
- RoHS compliant
- WEEE compliant

第三章 建立程序模板

3.1 实验目的

1. 学习 STM32CubeMX、Keil 基本使用
2. 建立 STM32 开发的程序模板

3.2 实验内容

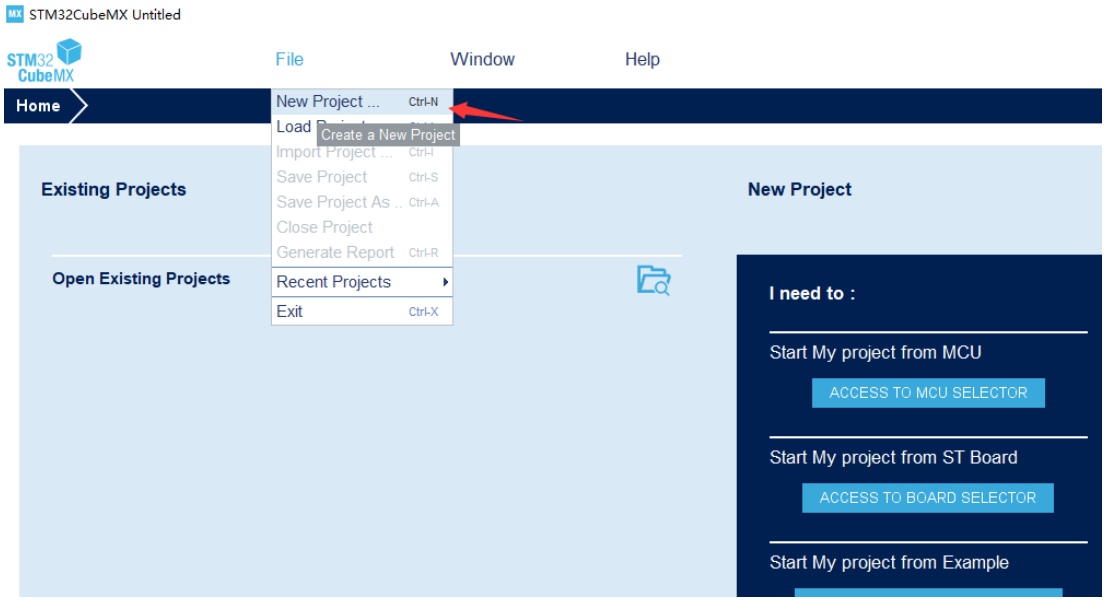
利用 STM32CubeMX 生成外设初始化代码

3.3 实验要求

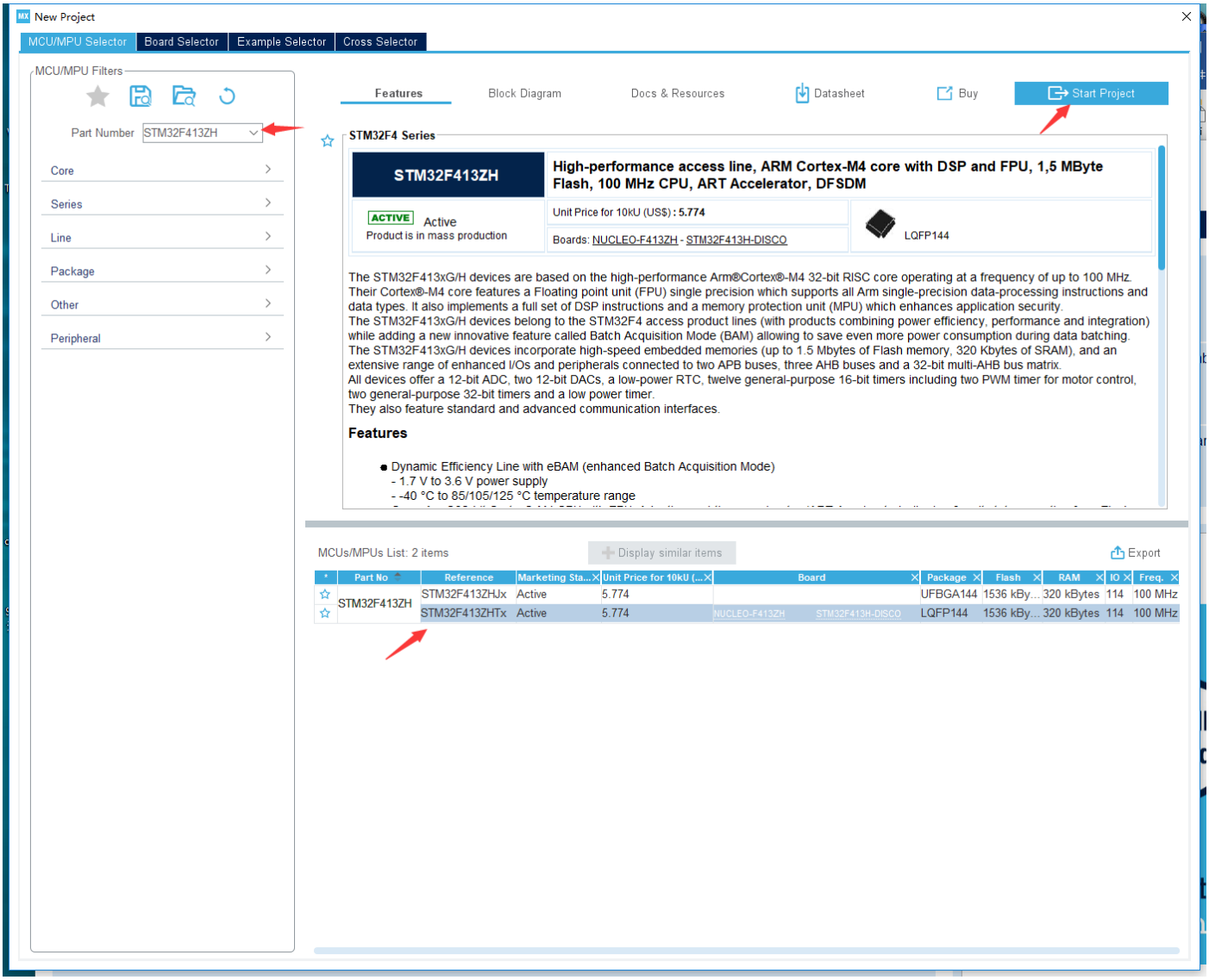
成功建立 Keil 工程，并编译通过

3.4 实验步骤

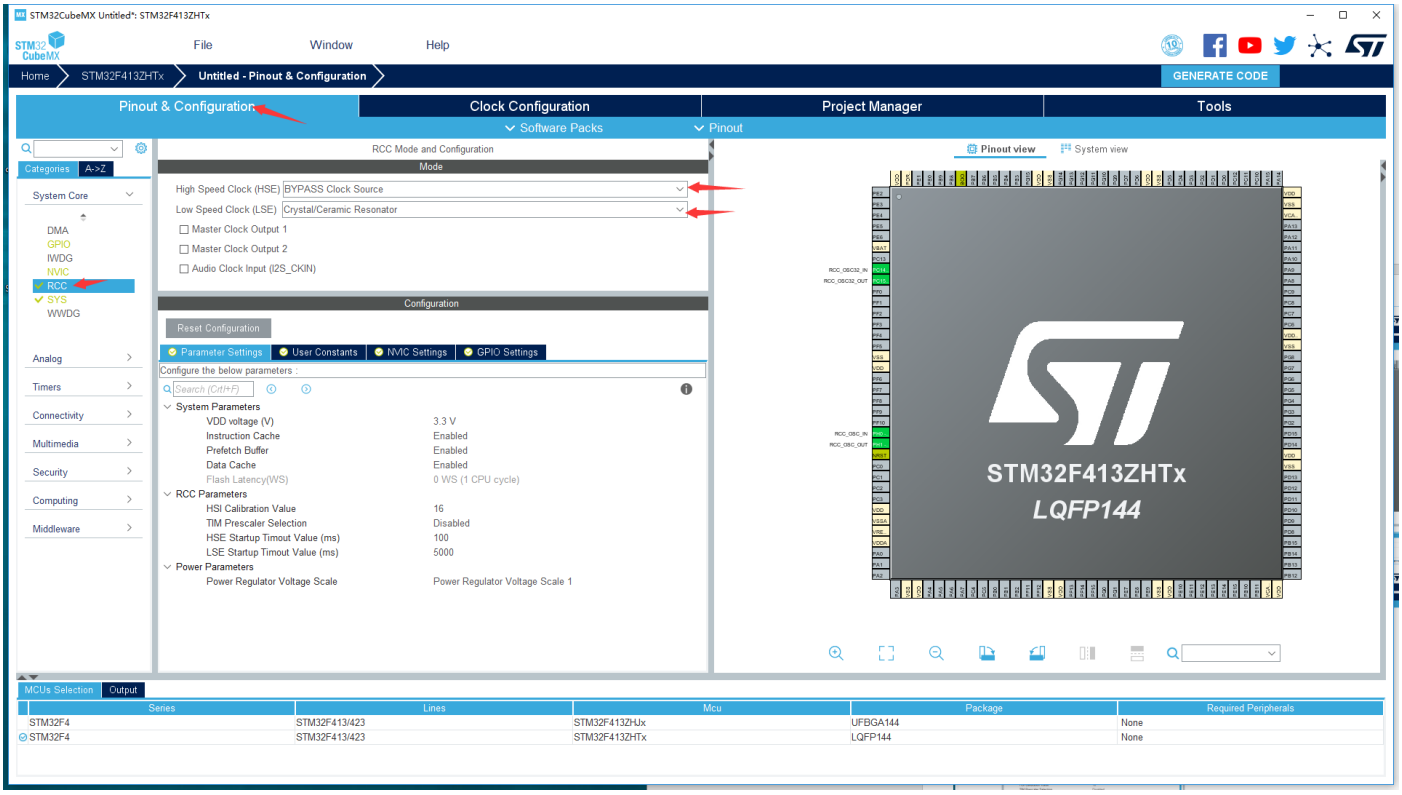
第一步，首先打开 STM32CubeMX 工具，点击如图所示的按钮新建工程。



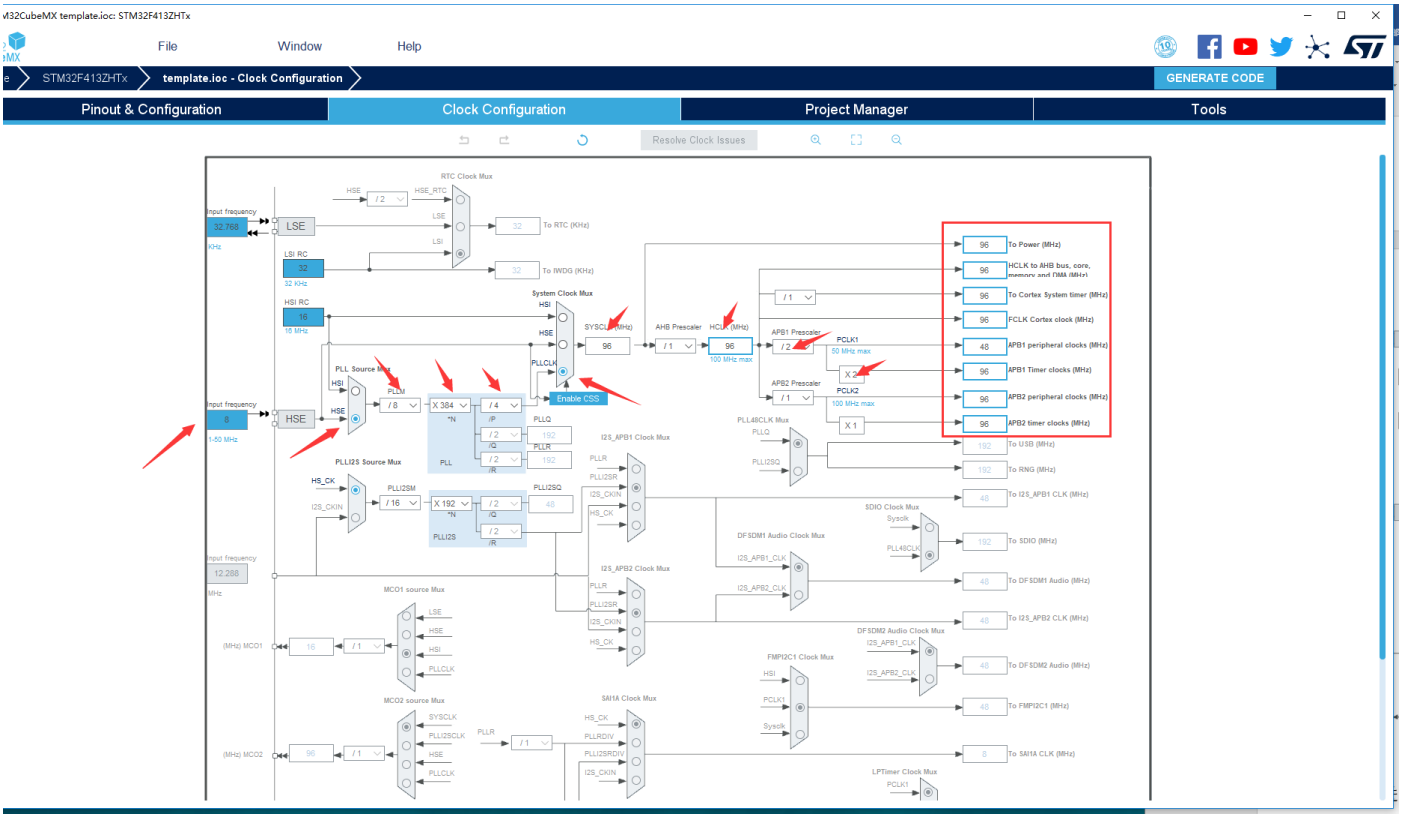
第二步，选择 Nucleo-144 的 MCU，型号为 STM32F413ZH，在搜索栏内可快速过滤。



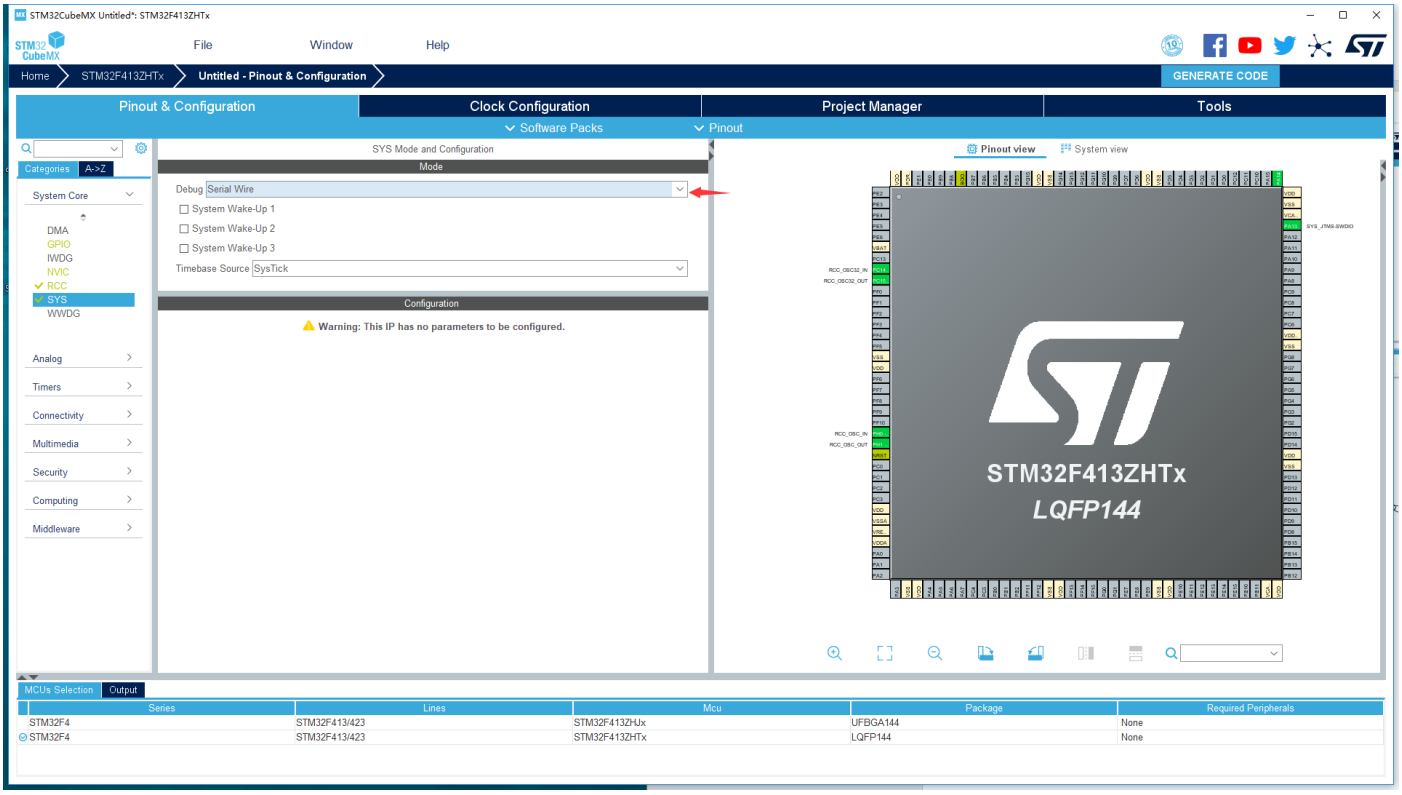
第三步，进入 RCC 界面对时钟源进行配置，这里高速时钟选择 BYPASS Clock Source，低速时钟选择 Crystal/Ceramic Resonator，其余保持默认



第四步，进入时钟树界面，对所标注部分分频系数进行修改。

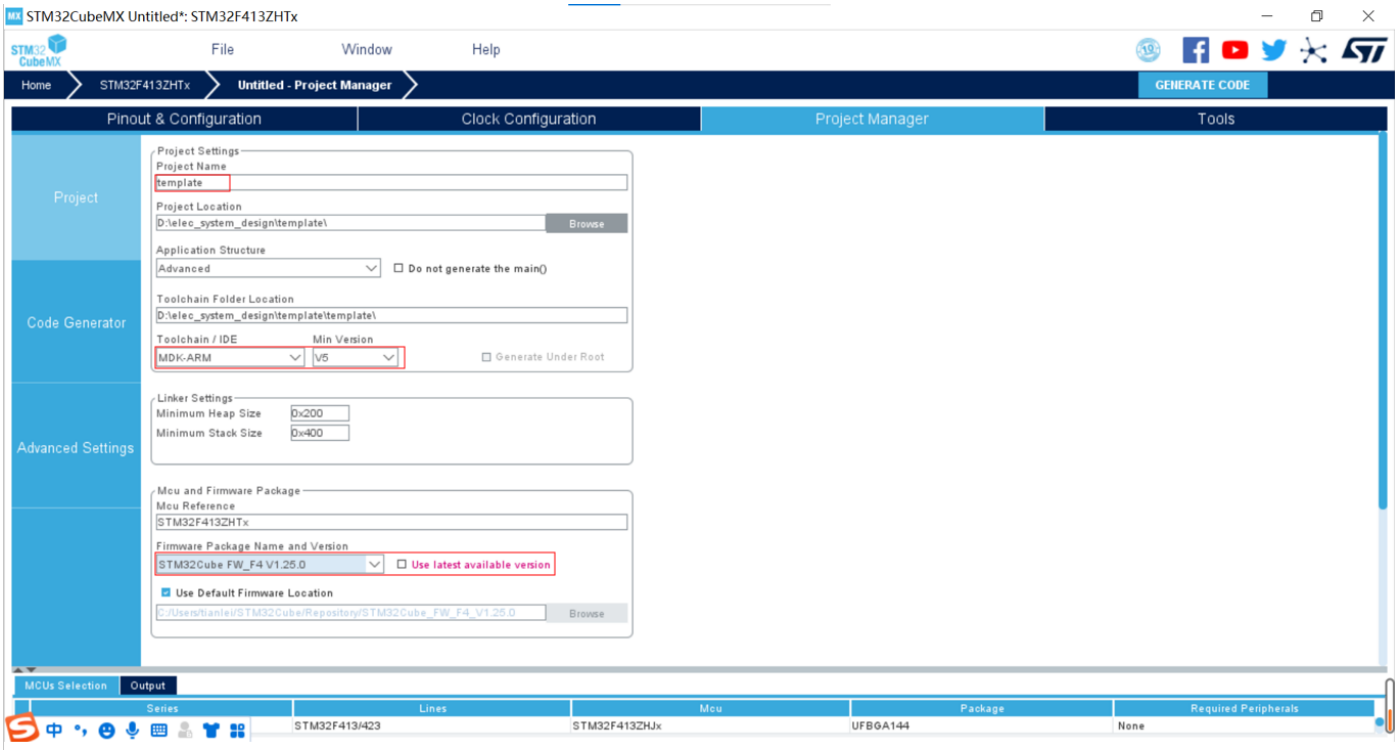


第五步，进入 SYS 界面对调试选项进行配置，这里在 Debug 内选择 Serial Wire 模式，其余保持默认。

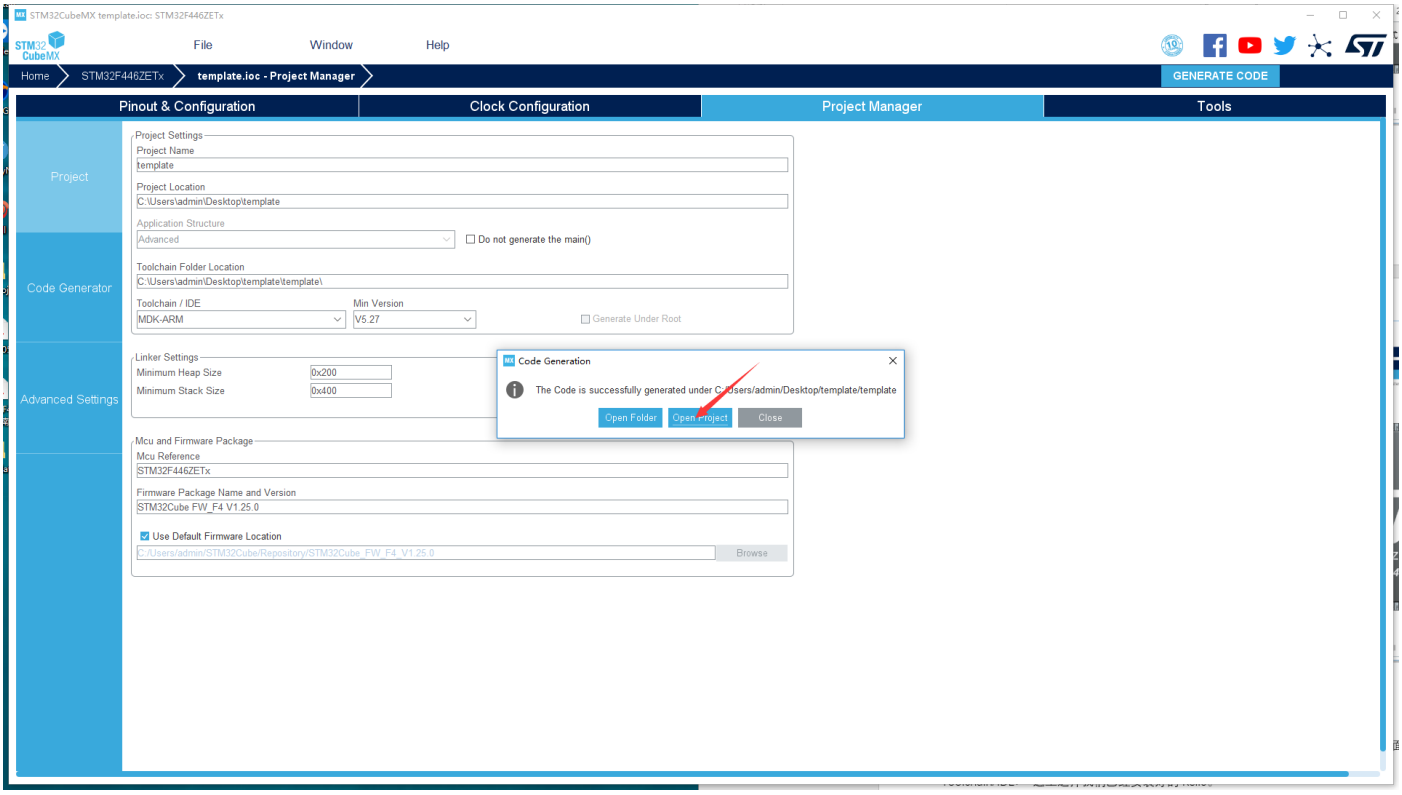


第六步，填写工程信息。

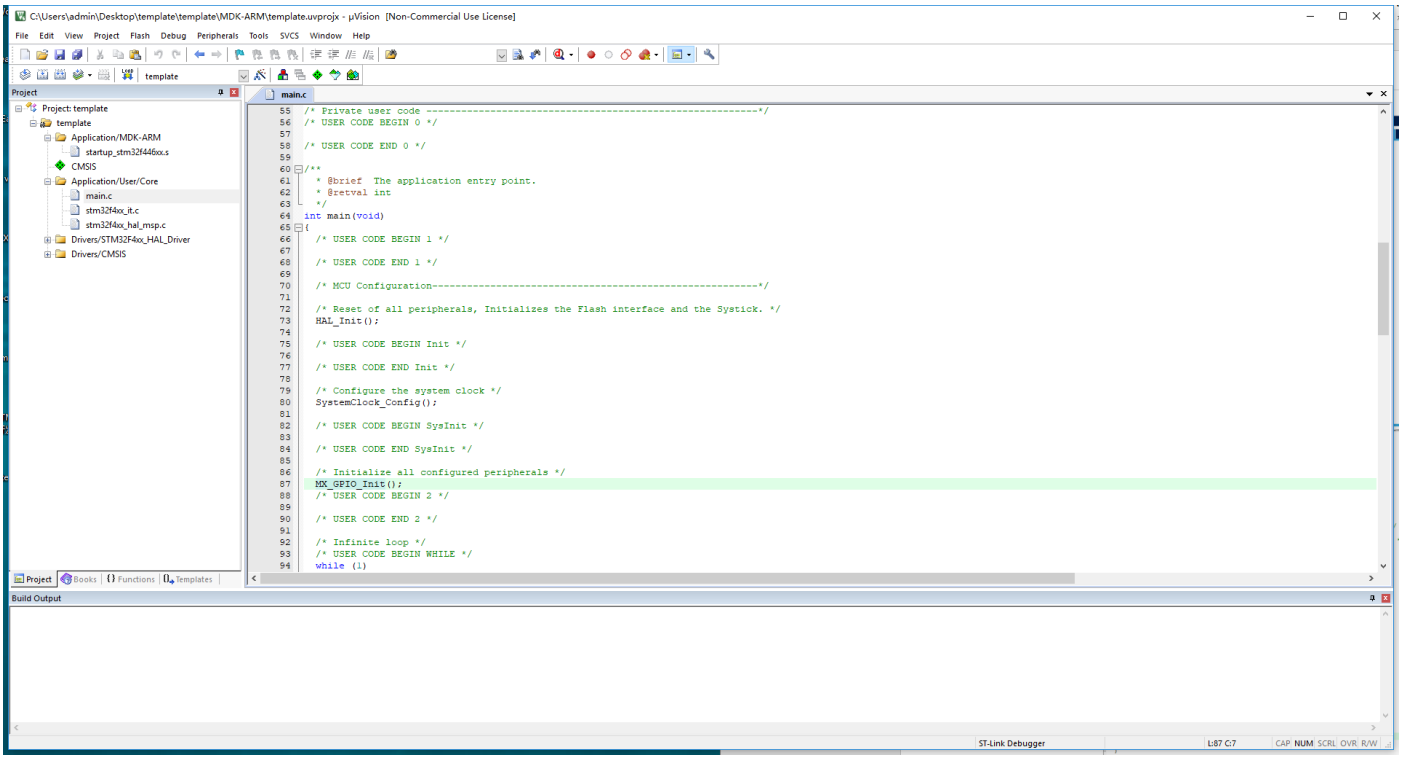
- Project Name： 工程名任意即可，这里填写 template。
- Project Location： 工程路径，建议新建空文件夹专门存放所有的工程文件，这里新建了 template 文件夹。注意路径中不要出现中文字符。
- Toolchain/IDE： 这里选择我们已经安装好的 Keil5，工具链（Toolchain）选择 MDK-ARM，版本号选择 V5
- 取消勾选 Use latest available version，选择 V1.25.0。
- 其余保持默认，点击右上角的 GENERATE CODE 生成 Keil 工程。



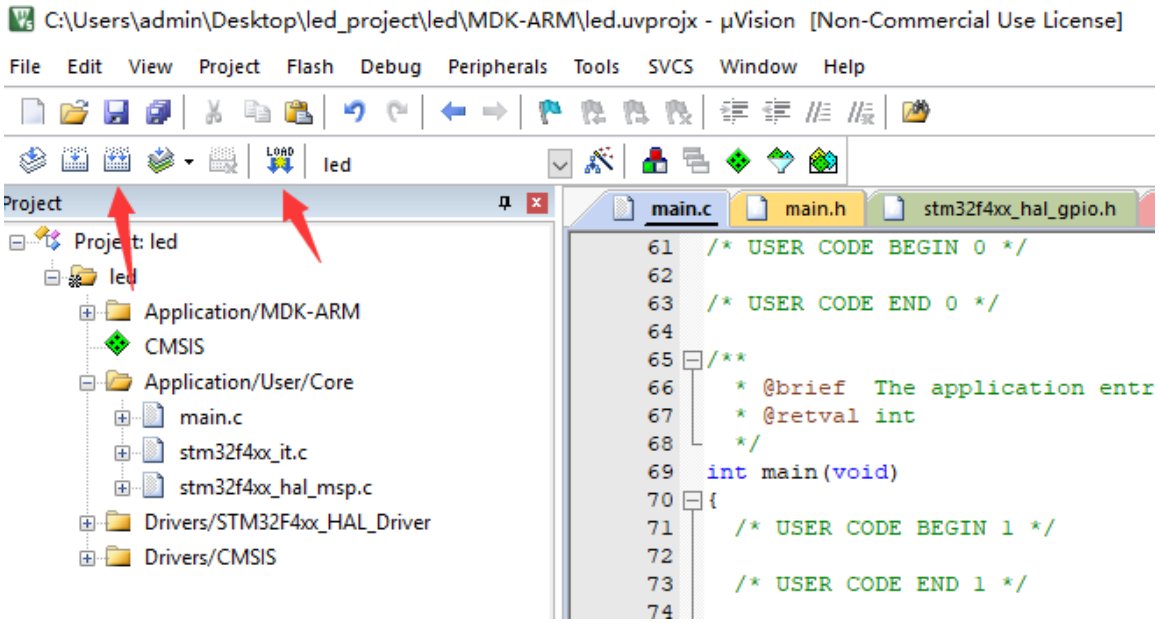
代码生成完成之后，点击打开 keil 工程。



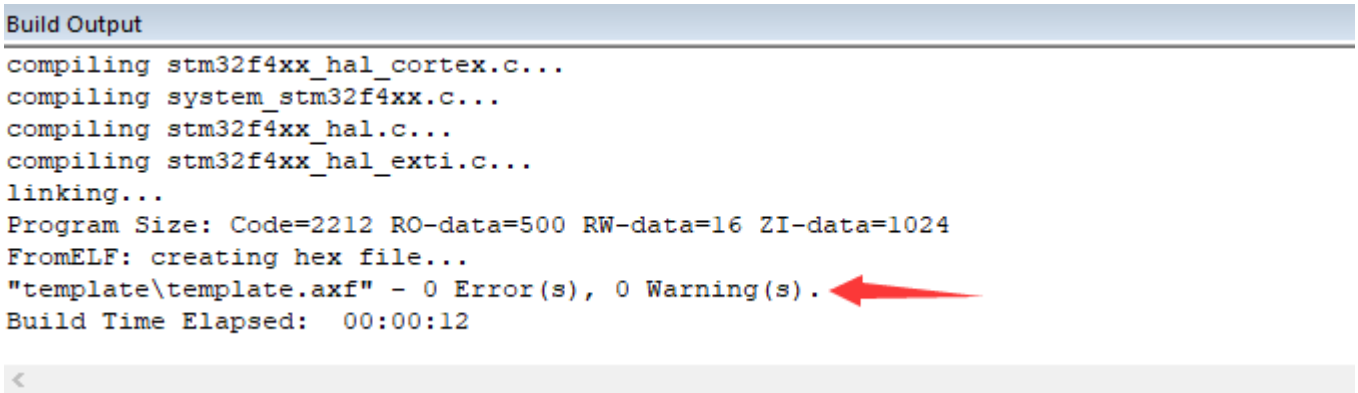
利用 Keil 打开工程以后，打开 main.c，这里是程序的入口。可以发现 STM32cube 已经帮我们完成了基本的初始化，并且在下方留出了空间让用户添加用户代码。



利用图示的两个按钮进行编译和代码烧录。



3.5 实验结果



可以看到编译结束后 0 错误 0 警告，代码生成成功。

第四章 跑马灯实验

4.1 实验目的

1. 学习 LED 灯的点亮原理
2. 实现对 Nucleo-144 开发板 LED 灯的控制

4.2 实验内容

通过对 GPIO 口拉高或拉低，实现对 LED 状态的控制

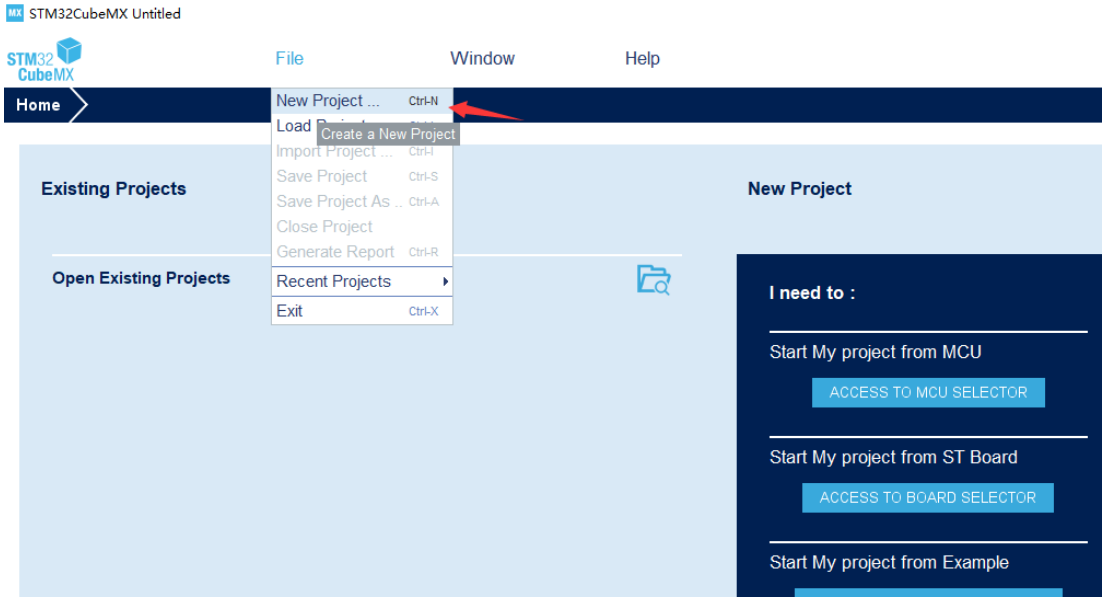
4.3 实验要求

Nucleo-144 上 LD1、LD2、LD3 实现闪烁效果

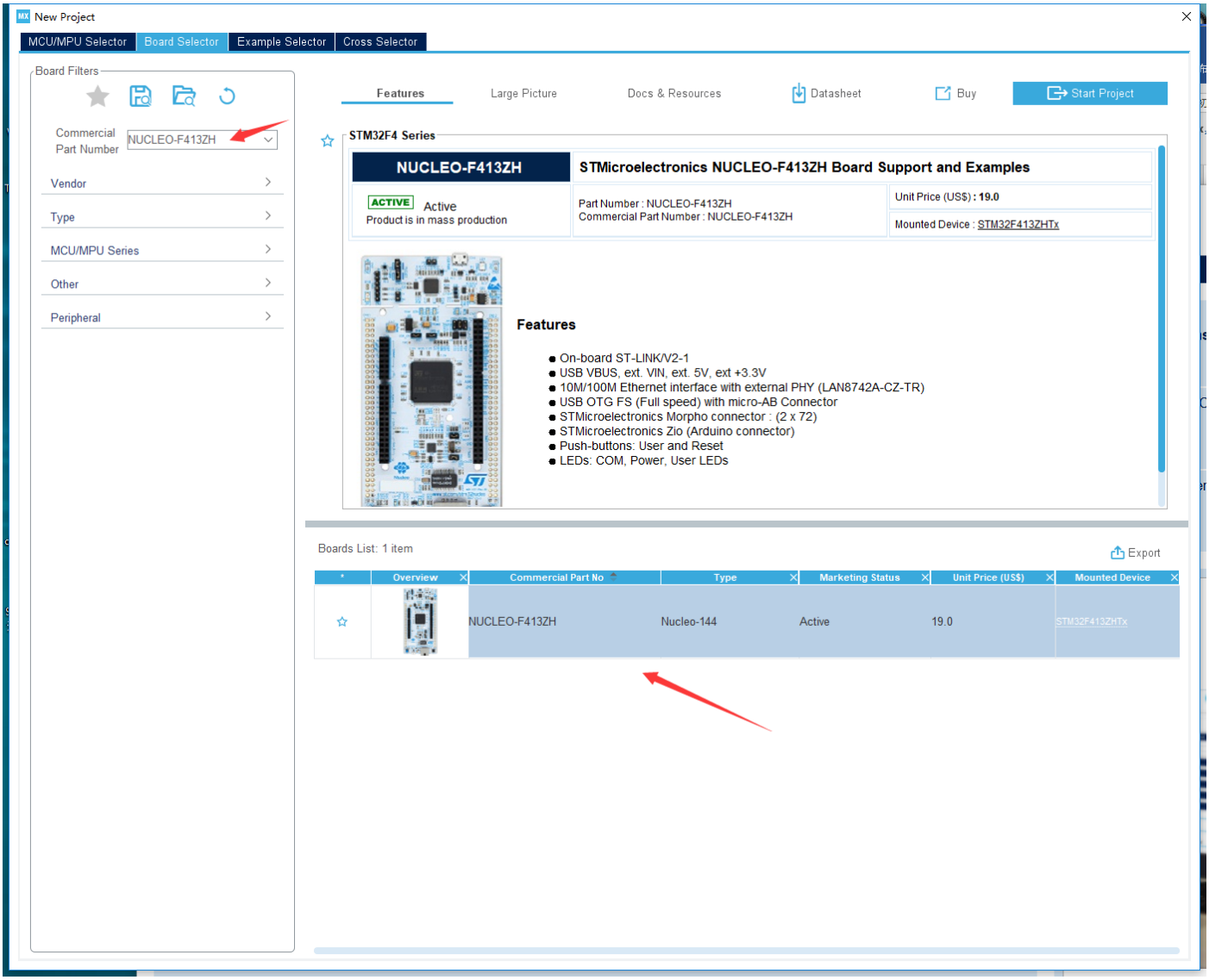
4.4 实验步骤

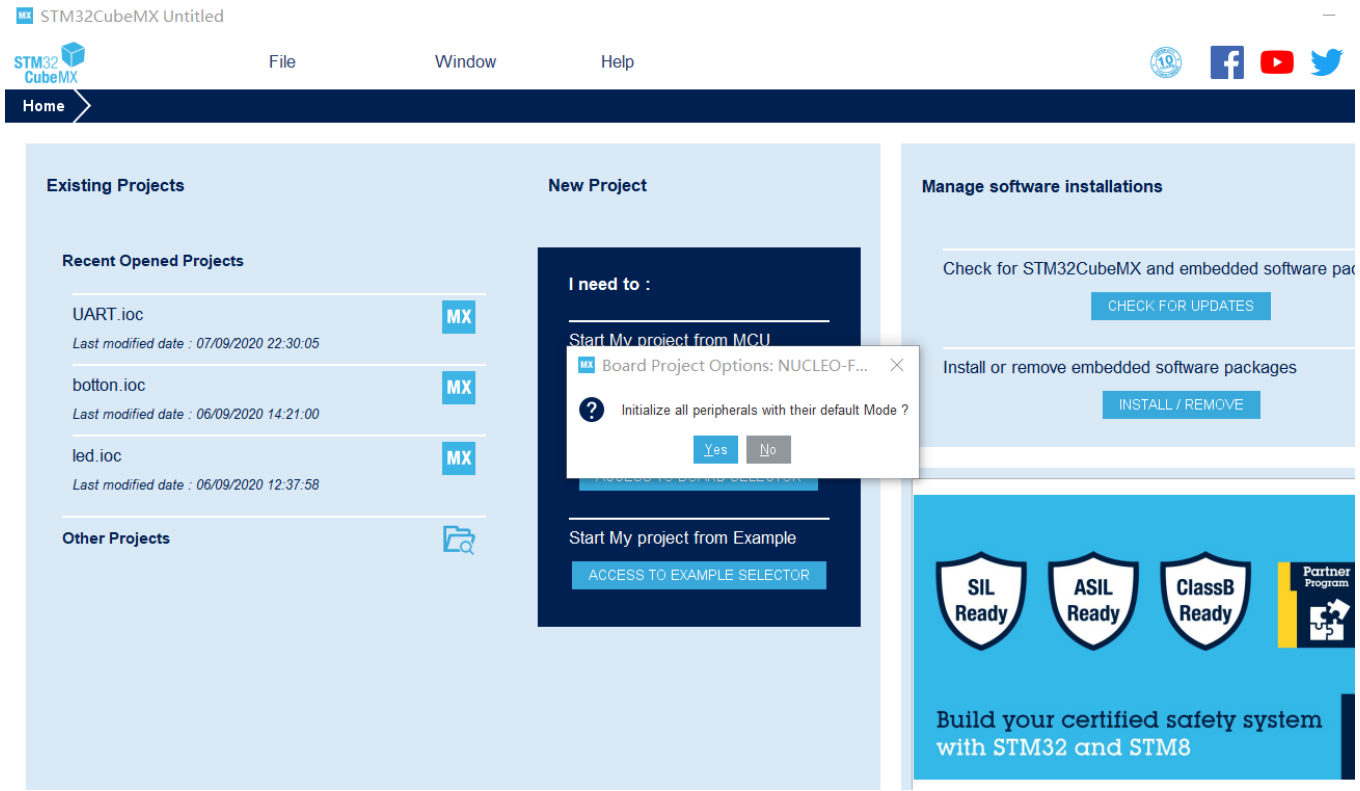
1. 利用 STM32CubeMX 生成模板代码

第一步，首先打开 STM32CubeMX 工具，点击如图所示的按钮新建工程。



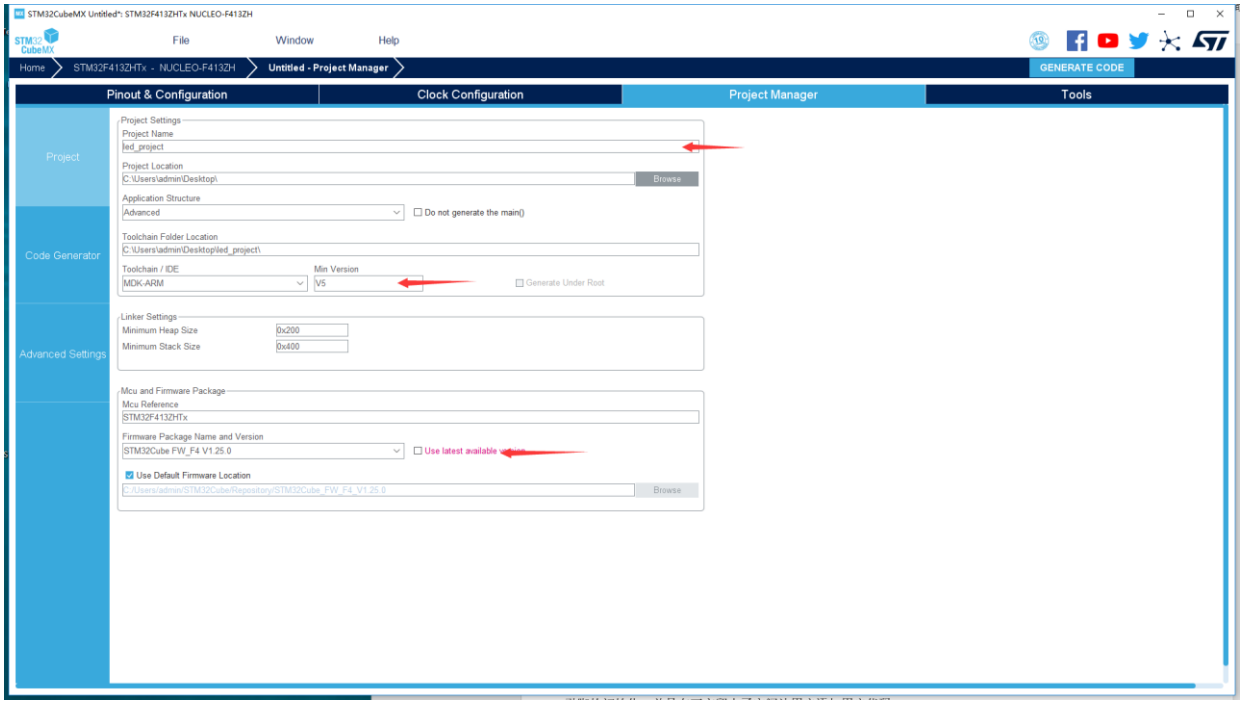
第二步，**本实验我们采用和第三章不同的方式，点击 Board Selector**，直接选择对应的 NUCLEO-144 开发板，**点击 YES**，选择完成基本配置，**打开 RCC 和 SYS 以及 LED 对应的 GPIO，可以看到这些都已经自动配置成功**。





第三步，创建工程后填写一下的工程信息。

- Project Name： 工程名任意即可，这里填写 led_project。
- Project Location： 工程路径，建议新建空文件夹专门存放所有的工程文件，这里新建了 led_project 文件夹。注意路径中不要出现中文字符。
- Toolchain/IDE： 这里选择我们已经安装好的 Keil5，工具链（Toolchain）选择 MDK-ARM，版本号选择 V5。
- 取消勾选 Use latest available version，选择 V1.25.0。
- 其余保持默认，点击右上角的 GENERATE CODE 生成 Keil 工程。



2. 利用 Keil 添加用户代码

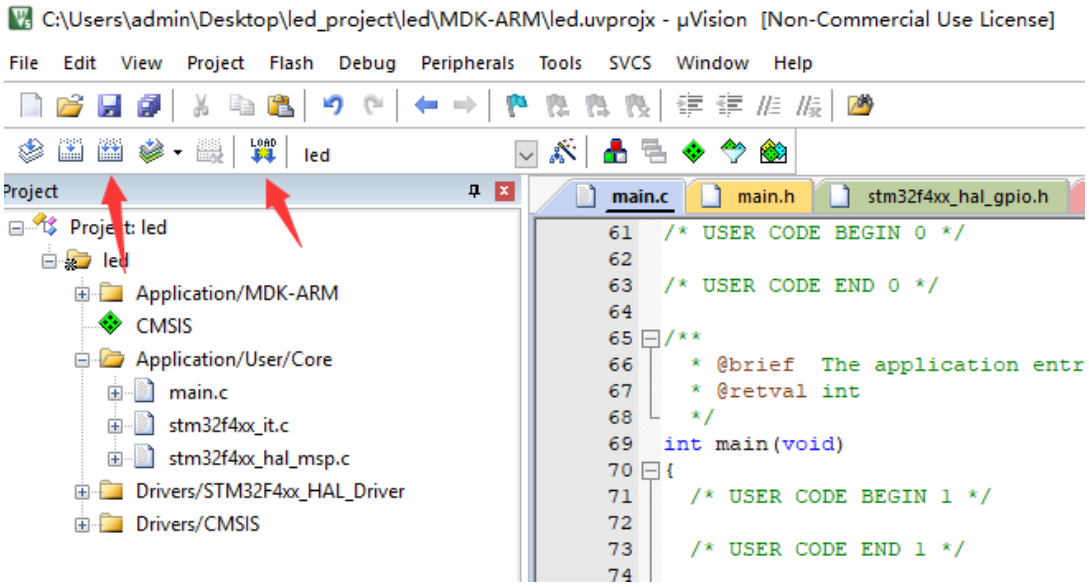
```
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART3_UART_Init();
  MX_USB_OTG_FS_PCD_Init();
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

利用 Keil 打开工程以后，打开 main.c，这里是程序的入口。可以发现 STM32cube 已经帮我们完成了关于 LED 引脚的初始化，并且在下方留出了空间让用户添加用户代码。

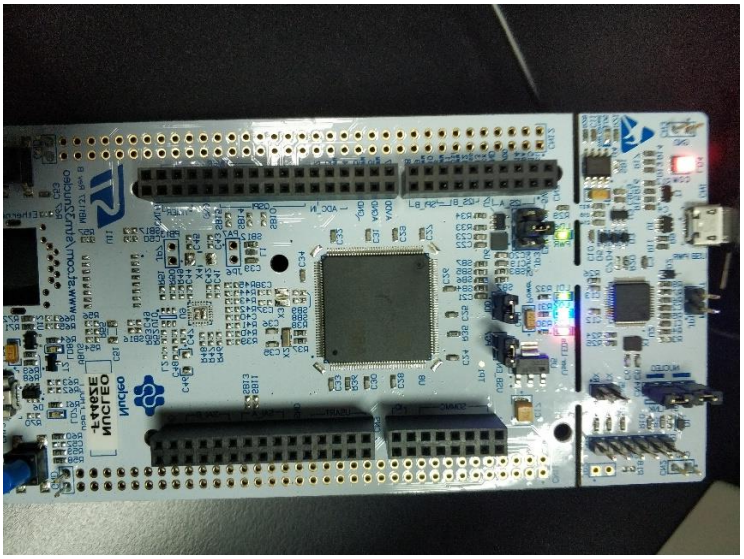
我们在 while 循环内添加如下语句，意为 LED 所在的口输出高电平，延时 500ms，之后输出低电平，再延时 500ms。

```
HAL_GPIO_WritePin(LD1_GPIO_Port,LD1_Pin,GPIO_PIN_SET);
HAL_GPIO_WritePin(LD2_GPIO_Port,LD2_Pin,GPIO_PIN_SET);
HAL_GPIO_WritePin(LD3_GPIO_Port,LD3_Pin,GPIO_PIN_SET);
HAL_Delay(500);

HAL_GPIO_WritePin(LD1_GPIO_Port,LD1_Pin,GPIO_PIN_RESET);
HAL_GPIO_WritePin(LD2_GPIO_Port,LD2_Pin,GPIO_PIN_RESET);
HAL_GPIO_WritePin(LD3_GPIO_Port,LD3_Pin,GPIO_PIN_RESET);
HAL_Delay(500);
```

利用图示的两个按钮进行编译和代码烧录。



4. 5 实验结果



代码烧录完成之后，按动开发板右下角的 RESET 按钮即可开始运行程序，可见 LED1、LED2、LED3 开始闪烁。

第五章 按键输入实验

5.1 实验目的

1. 学习按键的检测原理
2. 实现对 Nucleo-144 开发板按键的控制
3. 学会使用 STM32CubeMX 工具配置 GPIO

5.2 实验内容

检测用户按键是否按下并通过对 GPIO 口拉高或拉低，实现对 LED 状态的控制

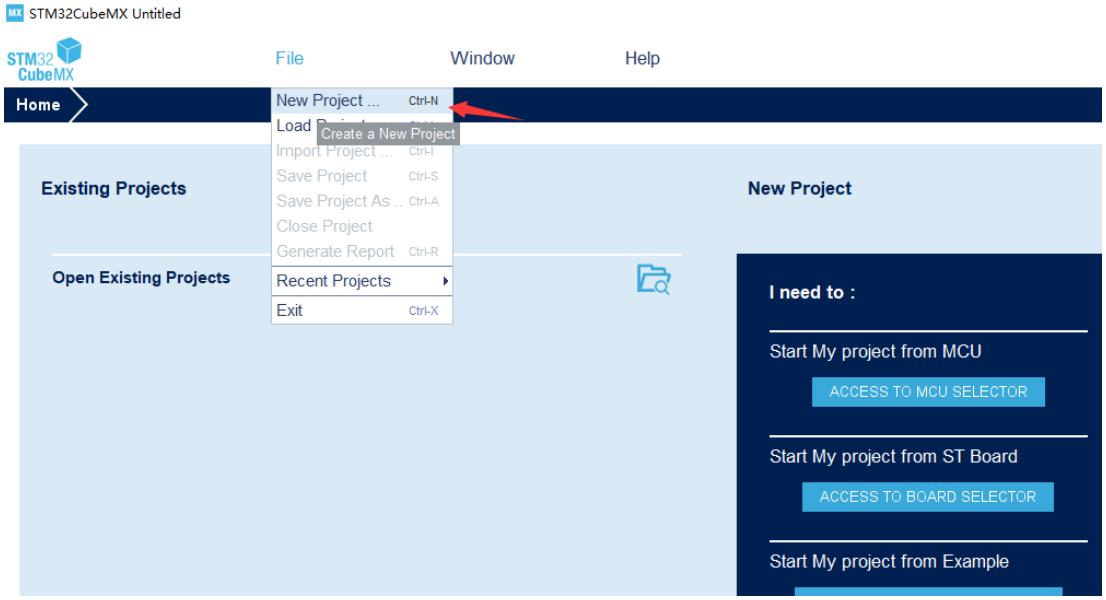
5.3 实验要求

Nucleo-144 上按下按键可以改变 LD1 的亮灭状态

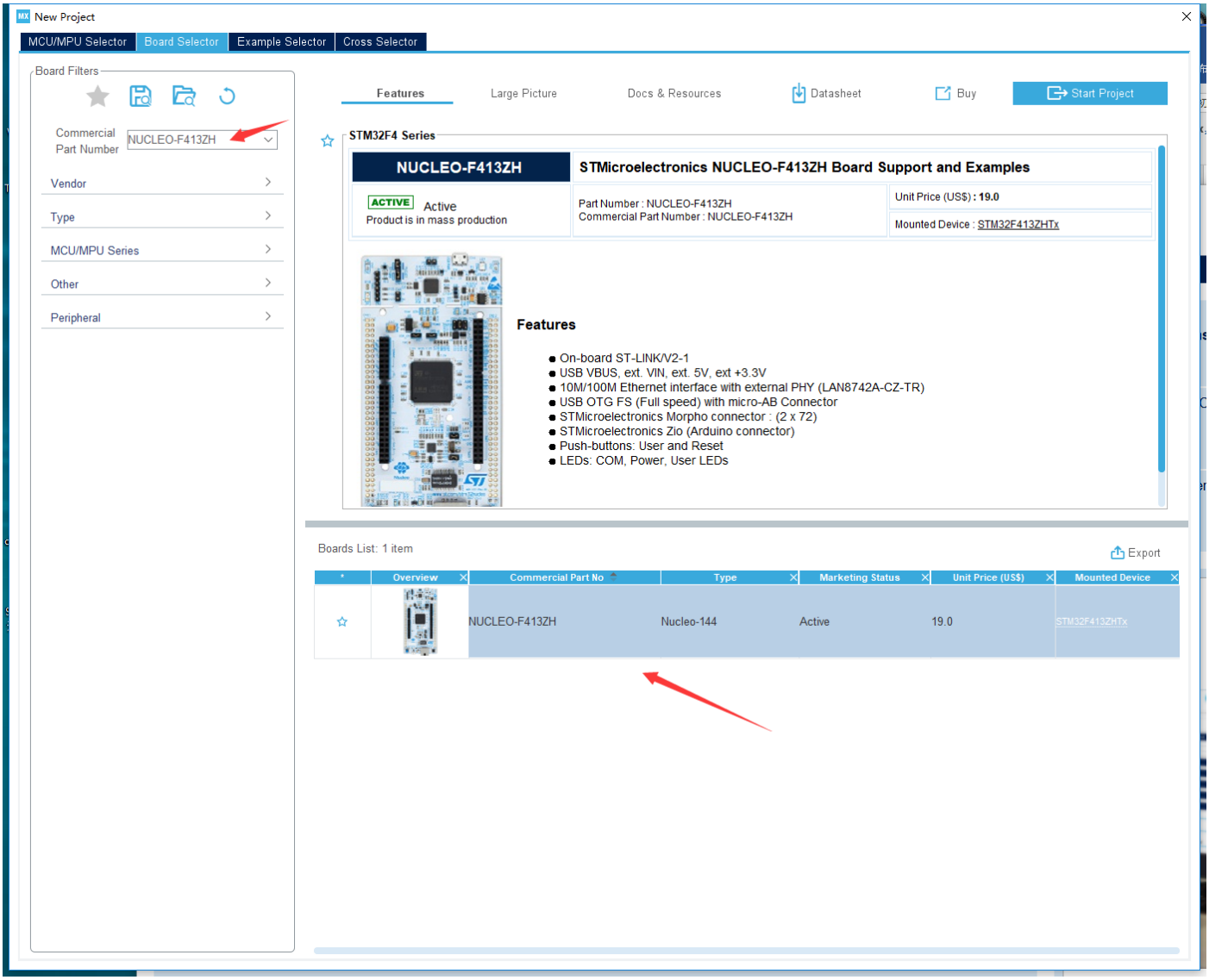
5.4 实验步骤

1. 利用 STM32CubeMX 生成模板代码

第一步，首先打开 STM32CubeMX 工具，点击如图所示的按钮新建工程。

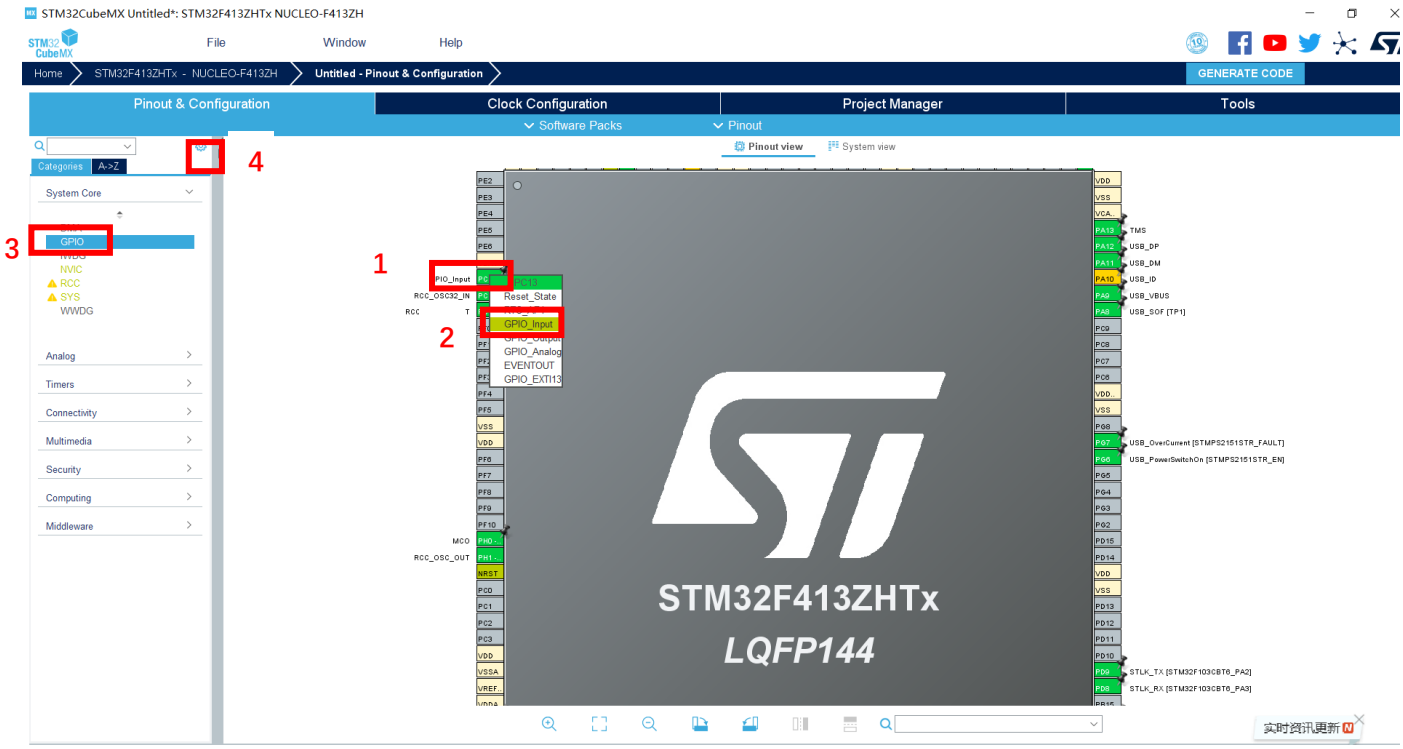


第二步，直接选择对应的 NUCLEO-144 开发板，完成基本配置。

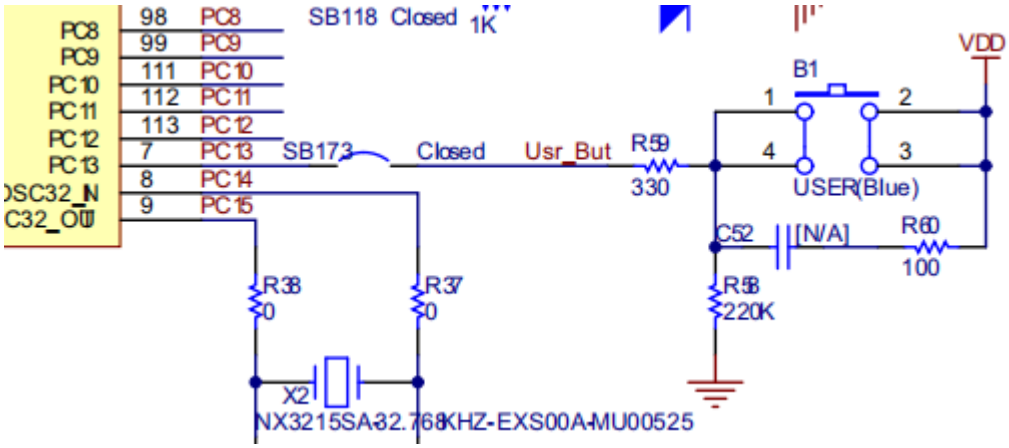


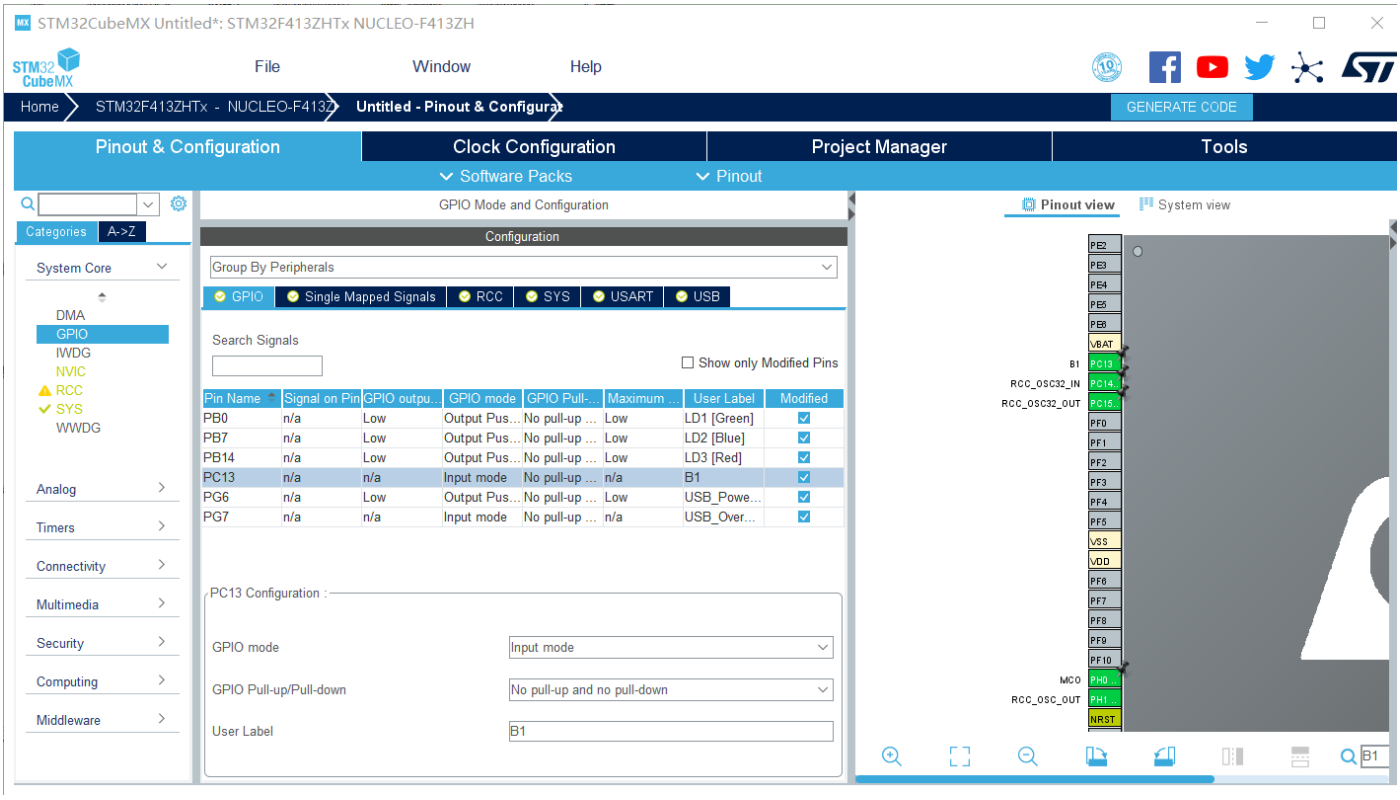
第三步，创建工程后填写一下的工程信息并对 GPIO 进行配置。

- Project Name： 工程名任意即可，这里填写 key。
- Project Location： 工程路径，建议新建空文件夹专门存放所有的工程文件，注意路径中不要出现中文字符。
- Toolchain/IDE： 这里选择我们已经安装好的 Keil5，工具链（Toolchain）选择 MDK-ARM，版本号选择 V5。
- 取消勾选 Use latest available version，选择 V1.25.0。
- 点击 Pinout&Configuration，在芯片引脚图中可以看到 PC13 已经被初始化为用户按键（蓝色按键 B1）的 IO 口，修改其为 GPIO_Input，



查阅 Nucleo-144 的数据手册可以知道 B1 的电平状态很明确，因此在 Configuration 中不需要设置上拉或下拉电阻，在这里可以设置 User Label 为 B1，方便后面的程序编写。

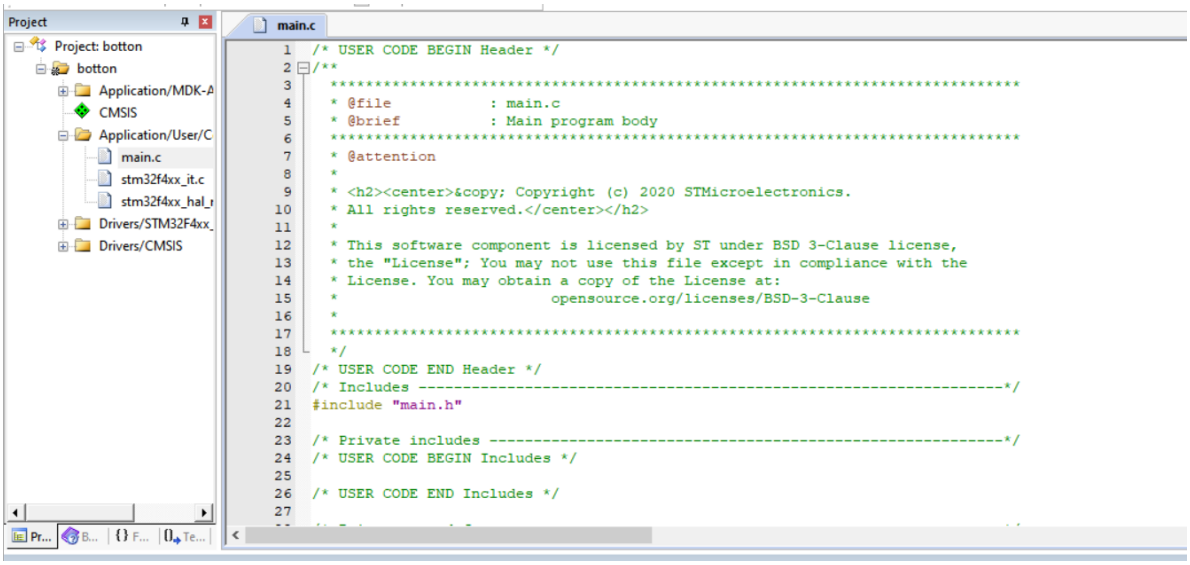




其余保持默认，点击右上角的 GENERATE CODE 生成 Keil 工程。

2. 利用 Keil 添加用户代码

工程建好后可以点击 open project，或者利用 Keil 打开工程以后，打开 main.c，这里是程序的入口。



可以发现 STM32cube 已经帮我们完成了关于按键引脚的初始化。

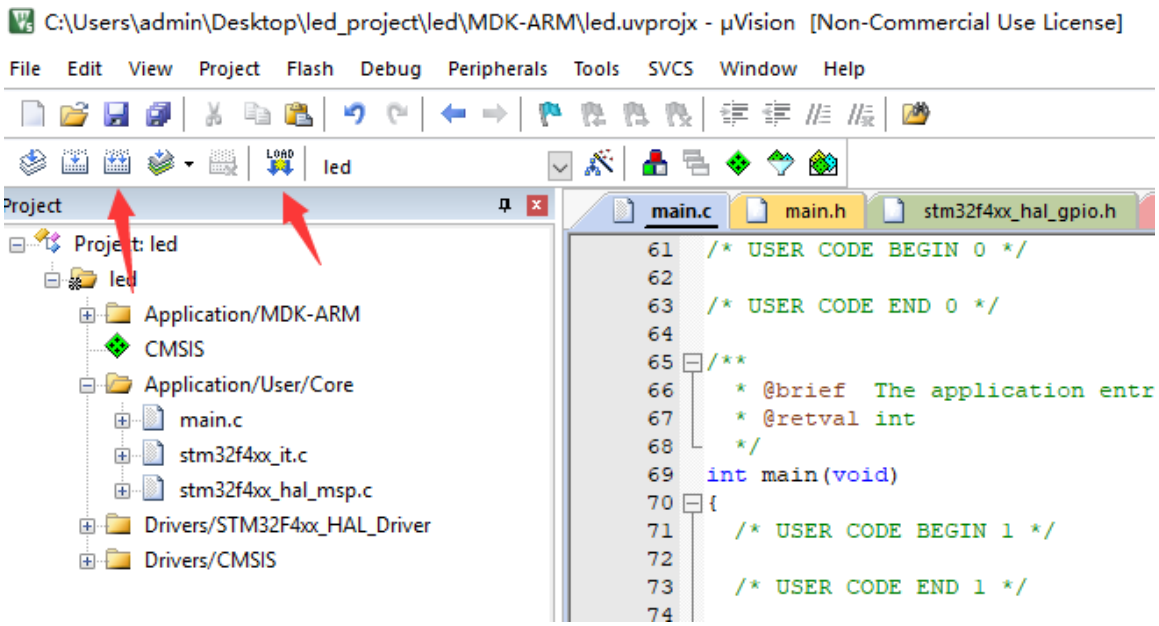
```
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOG_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, LD1_Pin|LD3_Pin|LD2_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(USB_PowerSwitchOn_GPIO_Port, USB_PowerSwitchOn_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin : B1_Pin */
  GPIO_InitStruct.Pin = B1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : LD1_Pin LD3_Pin LD2_Pin */
  GPIO_InitStruct.Pin = LD1_Pin|LD3_Pin|LD2_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
```

我们在 while 循环内添加如下语段，通过检测 IO 口的是否呈现高电平来检测按键是否被按下。

注： 由于开发板上已经做了硬件消抖，这里就不再做更多的处理。添加 1000ms 的延时为避免多次触发同一次按动。

```
if ((HAL_GPIO_ReadPin(B1_GPIO_Port,B1_Pin))==1)
{
    HAL_GPIO_TogglePin(LD1_GPIO_Port,LD1_Pin);
}
HAL_Delay(1000);
```

利用图示的两个按钮进行编译和代码烧录。



5.5 实验结果

代码烧录完成之后，按动开发板右下角的 RESET 按钮即可开始运行程序，可以观察到，按下按键，LD1 亮，再次按下按键，松开，LD1 灭，如此循环。

第六章 串口通信实验

6.1 实验目的

1. 学习串口与上位机的通信原理
2. 实现对 Nucleo-144 开发板串口的控制

6.2 实验内容

通过调用串口的发送功能，实现串口与上位机的通信

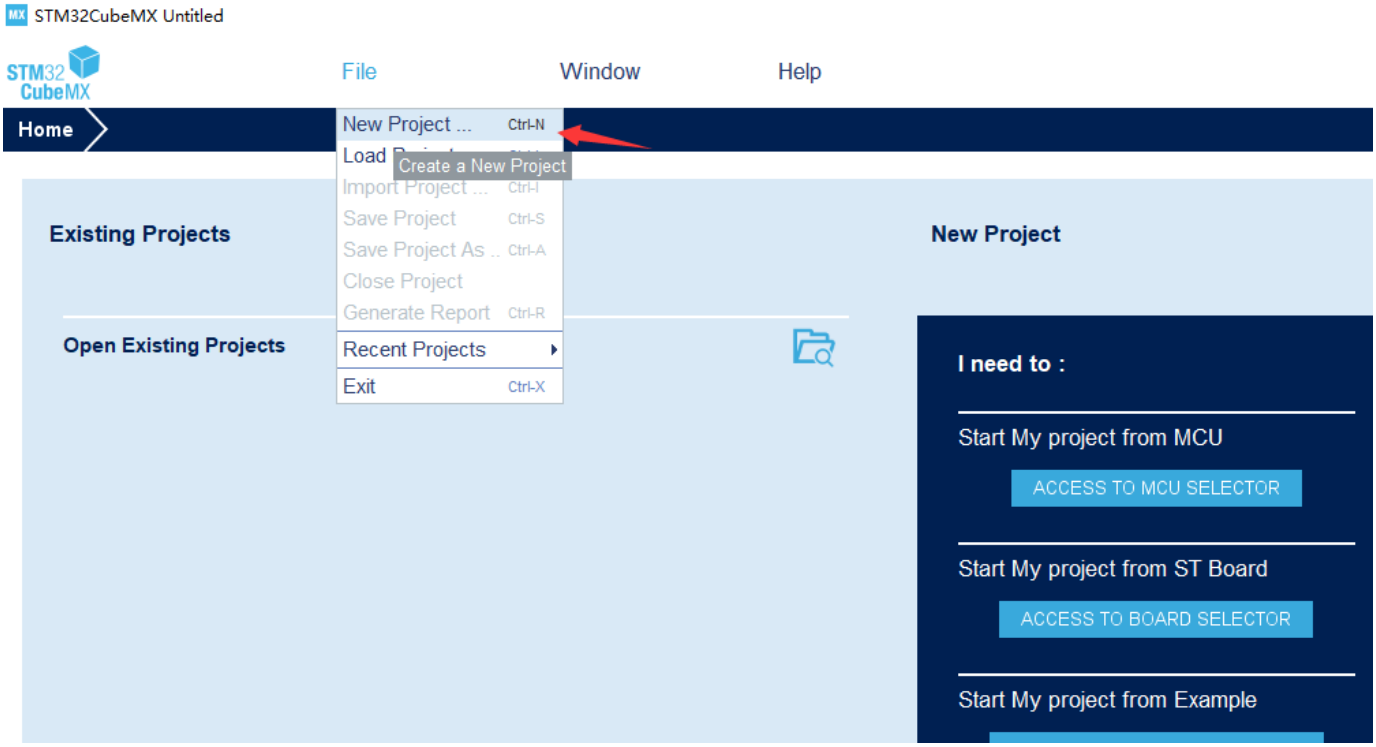
6.3 实验要求

Nucleo-144 上能够向上位机发送自定义的字符串；

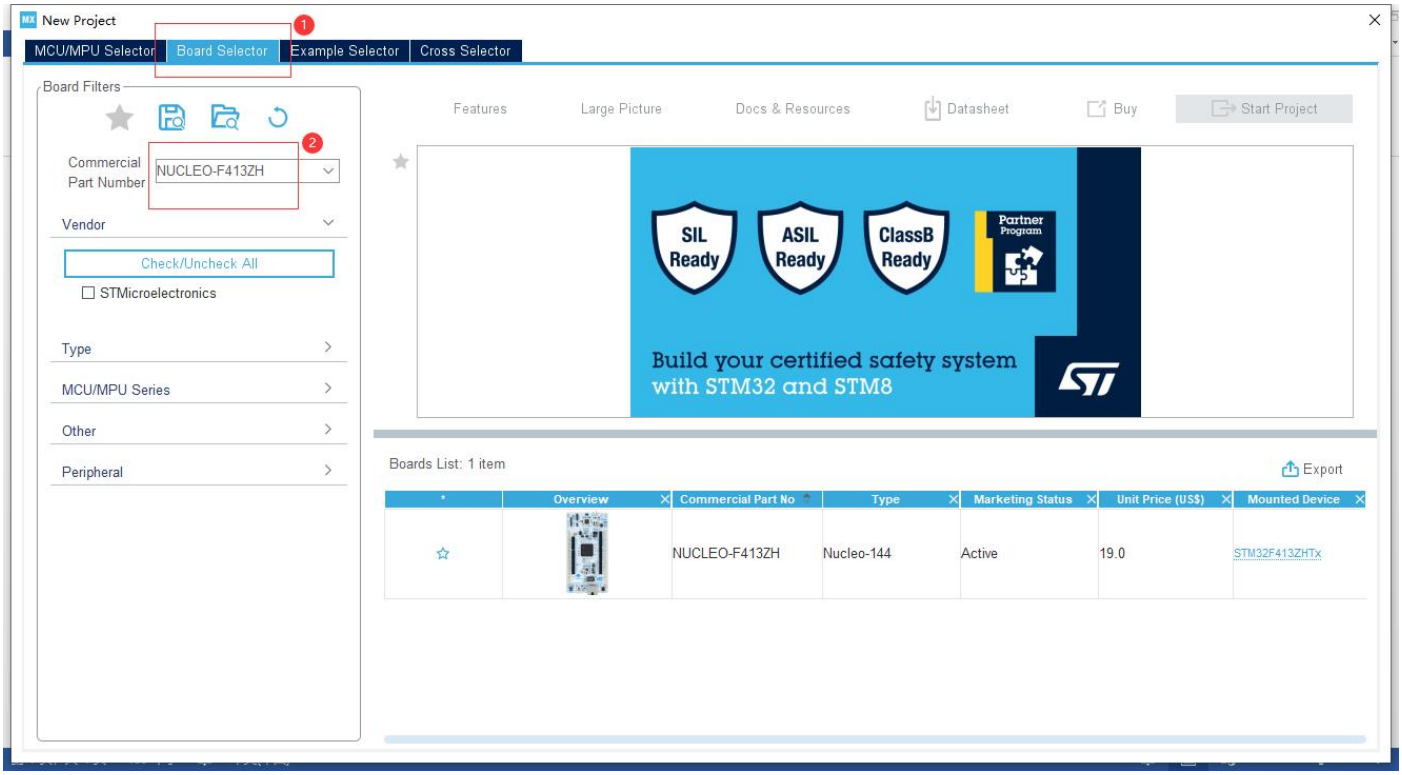
6.4 实验步骤

1. 利用 STM32CubeMX 生成模板代码

第一步，首先打开 STM32CubeMX 工具，点击如图所示的按钮新建工程。



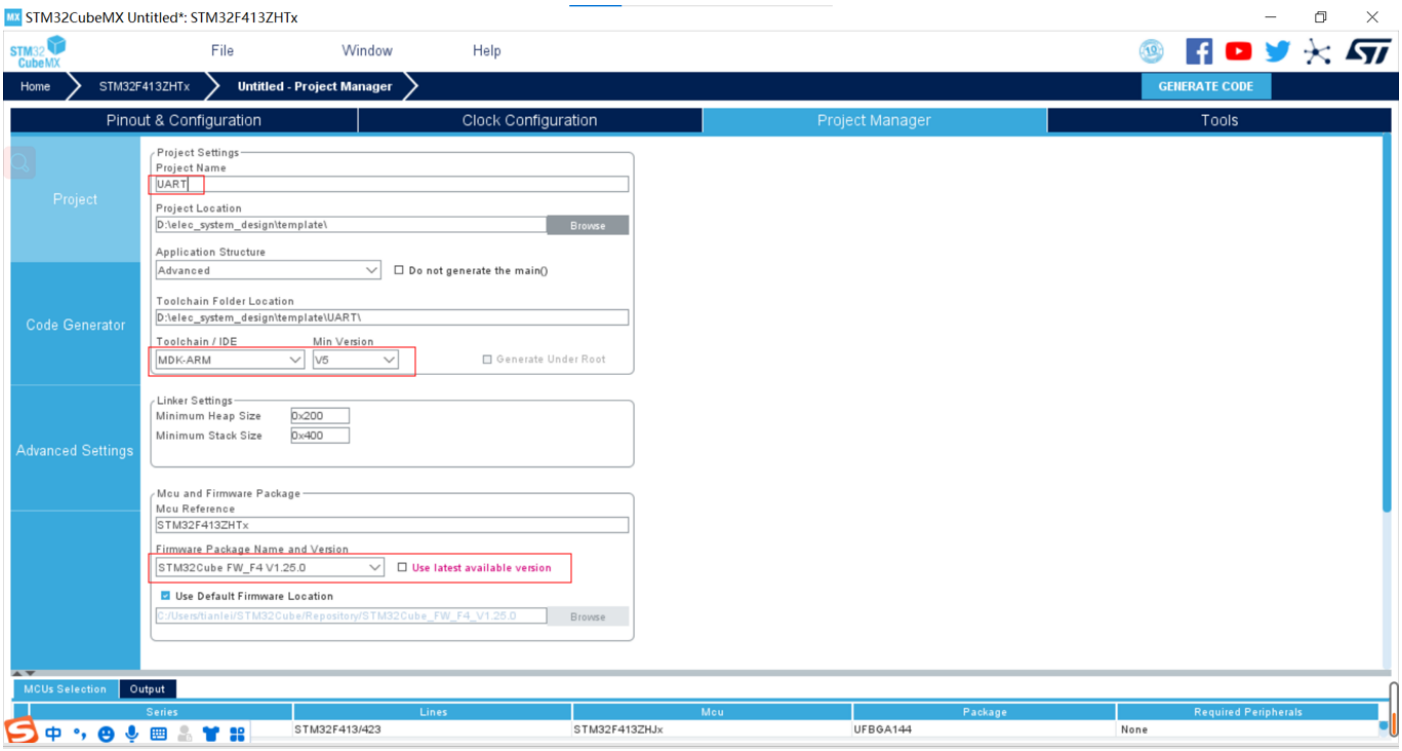
第二步，直接选择对应的 NUCLEO-144 开发板，完成基本配置。



第三步，创建工程后填写一下的工程信息。

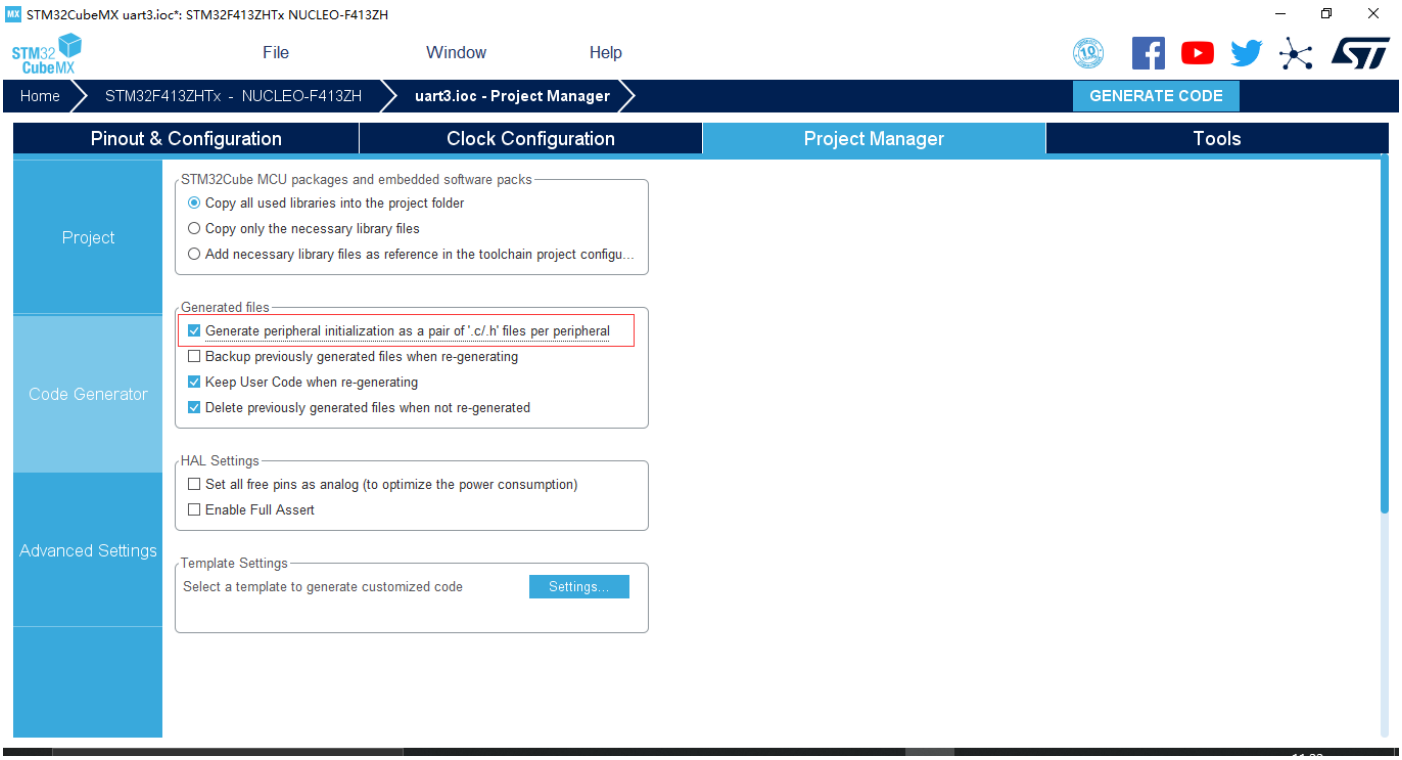
Project 栏目：

- Project Name： 工程名任意即可，这里填写 UART。
- Project Location： 工程路径，建议新建空文件夹专门存放所有的工程文件，注意路径中不要出现中文字符。
- Toolchain/IDE： 这里选择我们已经安装好的 Keil5，工具链（Toolchain）选择 MDK-ARM，版本号选择 V5。
- 取消勾选 Use latest available version，选择 V1.25.0。



Code Generator 栏目：

- 勾选第一项，可以将外设的初始化函数等生成单独的“.h”、”.c”文件，避免函数全部放到 main.c 里面。



2. 利用 Keil 添加用户代码并烧录

第一步，编写用户代码

```
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART3_UART_Init();
  MX_USB_OTG_FS_PCD_Init();
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

利用 Keil 打开工程以后，打开 main.c，这里是程序的入口。可以发现 STM32cube 已经帮我们完成了关于 UART 的初始化，并且在下方留出了空间让用户添加用户代码。

在 Include 处（21 行），添加如下头文件

```
#include "stdio.h"
```

在 main 函数上方，添加如下语段，完成对 printf 函数的重定向。

```
#ifdef __GNUC__
/* With GCC, small printf (option LD Linker->Libraries->Small printf
     set to 'Yes') calls __io_putchar() */
#define PUTCHAR_PROTOTYPE int __io_putchar(int ch)
#else
#define PUTCHAR_PROTOTYPE int fputc(int ch, FILE *f)
#endif /* __GNUC__ */

PUTCHAR_PROTOTYPE
{
   /* Place your implementation of fputc here */
   /* e.g. write a character to the USART3 and Loop until the end of transmission */
   HAL_UART_Transmit(&huart3, (uint8_t *)&ch, 1, 0xFFFF);

   return ch;
};
```

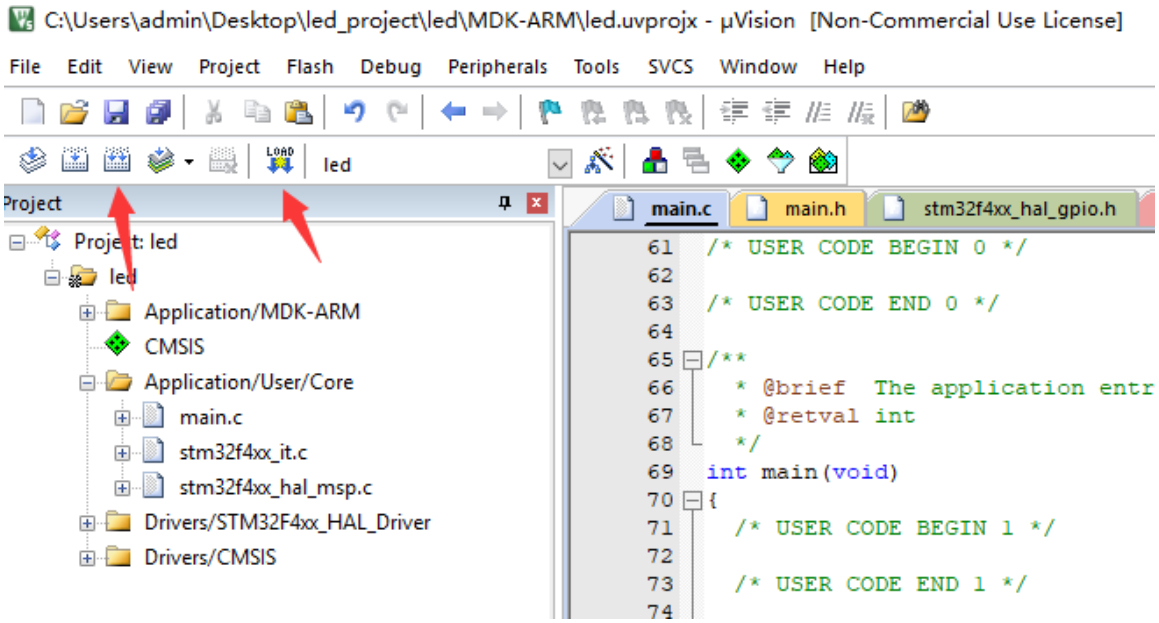
在 main 函数的 while 循环内，添加如下语段

```
printf("seu\r\n");
```

修改完成后的效果如下所示

```
  #ifdef __GNUC__
/* With GCC, small printf (option LD Linker->Libraries->Small printf
   set to 'Yes') calls __io_putchar() */
#define PUTCHAR_PROTOTYPE int __io_putchar(int ch)
#else
#define PUTCHAR_PROTOTYPE int fputc(int ch, FILE *f)
#endif /* __GNUC__ */

PUTCHAR_PROTOTYPE
{
  /* Place your implementation of fputc here */
  /* e.g. write a character to the USART3 and Loop until the end of transmission */
  HAL_UART_Transmit(&huart3, (uint8_t *)&ch, 1, 0xFFFF);

  return ch;
};


int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART3_UART_Init();
  MX_USB_OTG_FS_PCD_Init();
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */
    printf("seu\r\n");

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

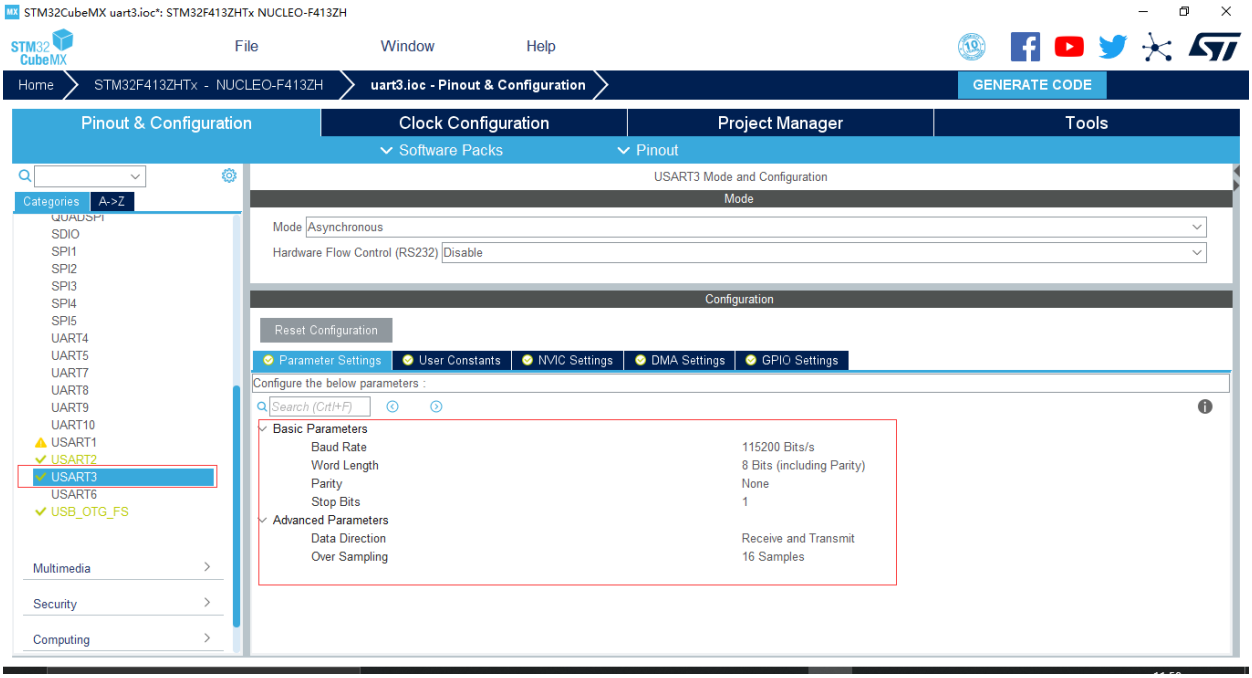
第二步，利用图示的两个按钮进行编译和代码烧录。

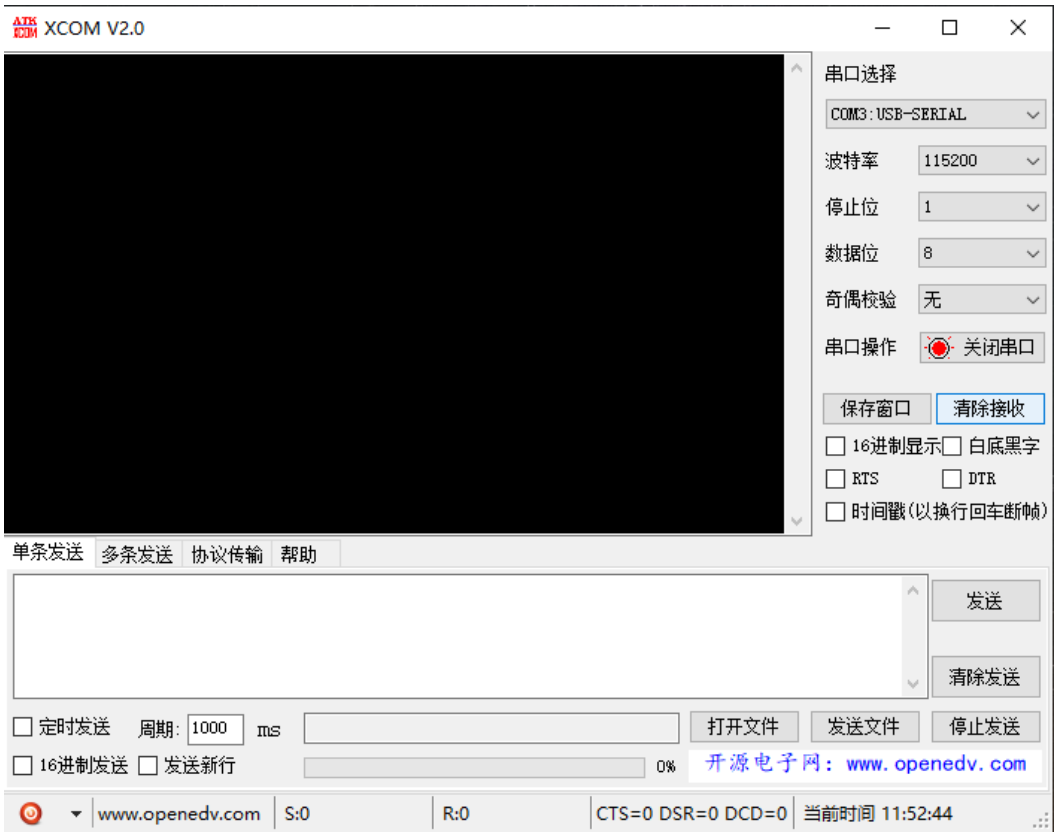


3. 进行串口配置

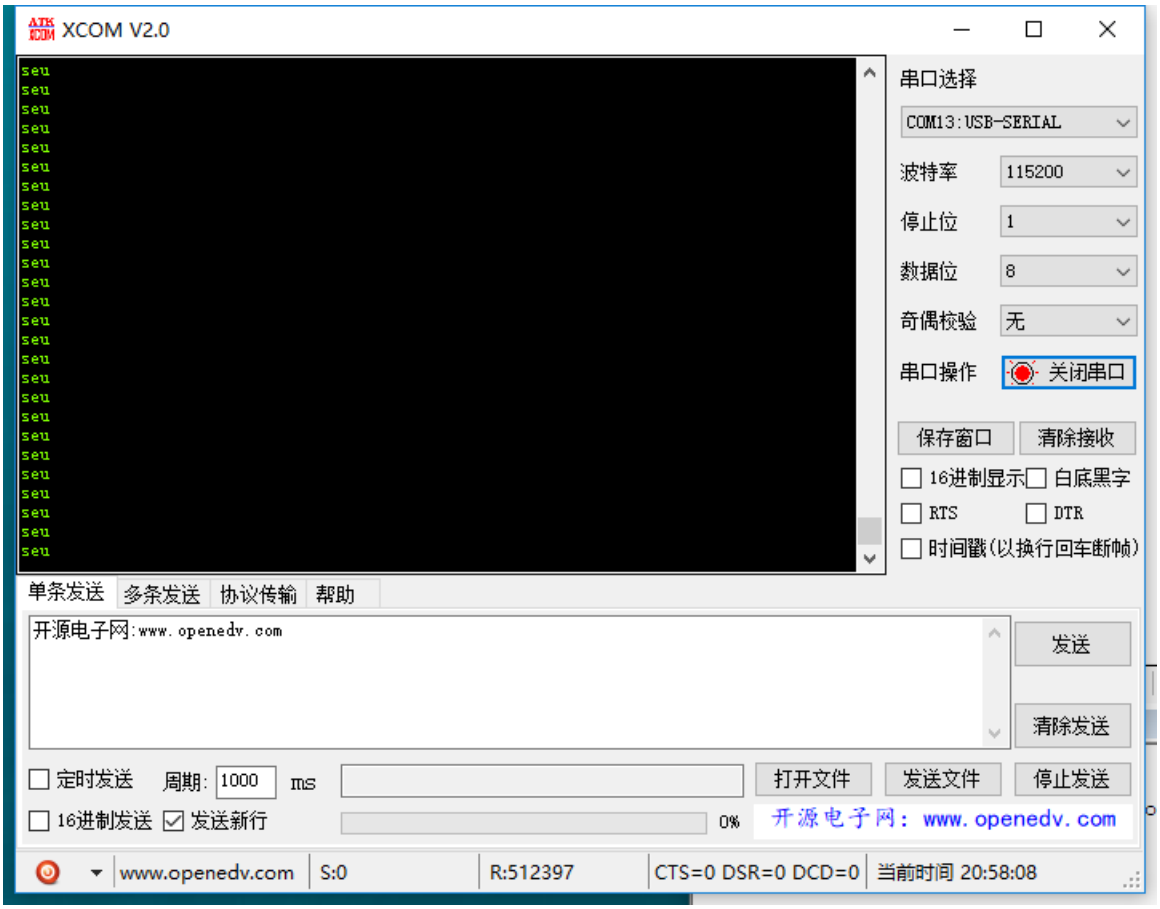
打开串口调试助手 XCOM V2.0，按照在 STM32CubeMX 里面查看的串口参数进行对应设置

- Baud Rate: 波特率，波特率表示每秒钟传送的码元符号的个数，是衡量数据传送速率的指标，它用单位时间内载波调制状态改变的次数来表示。这里选择 115200
- Word Length：数据长，这里选择 8bits
- Parity：奇偶校验，分为 无、奇校验、偶校验 三种，这里选择无
- Stop：停止位，分为 1、1.5、2 三种，这里选择 1





6.5 实验结果



可以看到 Nucleo-144 在不断向串口助手输出 seu 的字样。

第七章 外部中断实验

7.1 实验目的

1. 学习外部中断的原理
2. 利用外部中断实现 Nucelo-144 按键的控制

7.2 实验内容

按键按下时触发外部中断，以此控制 LED 闪烁状态

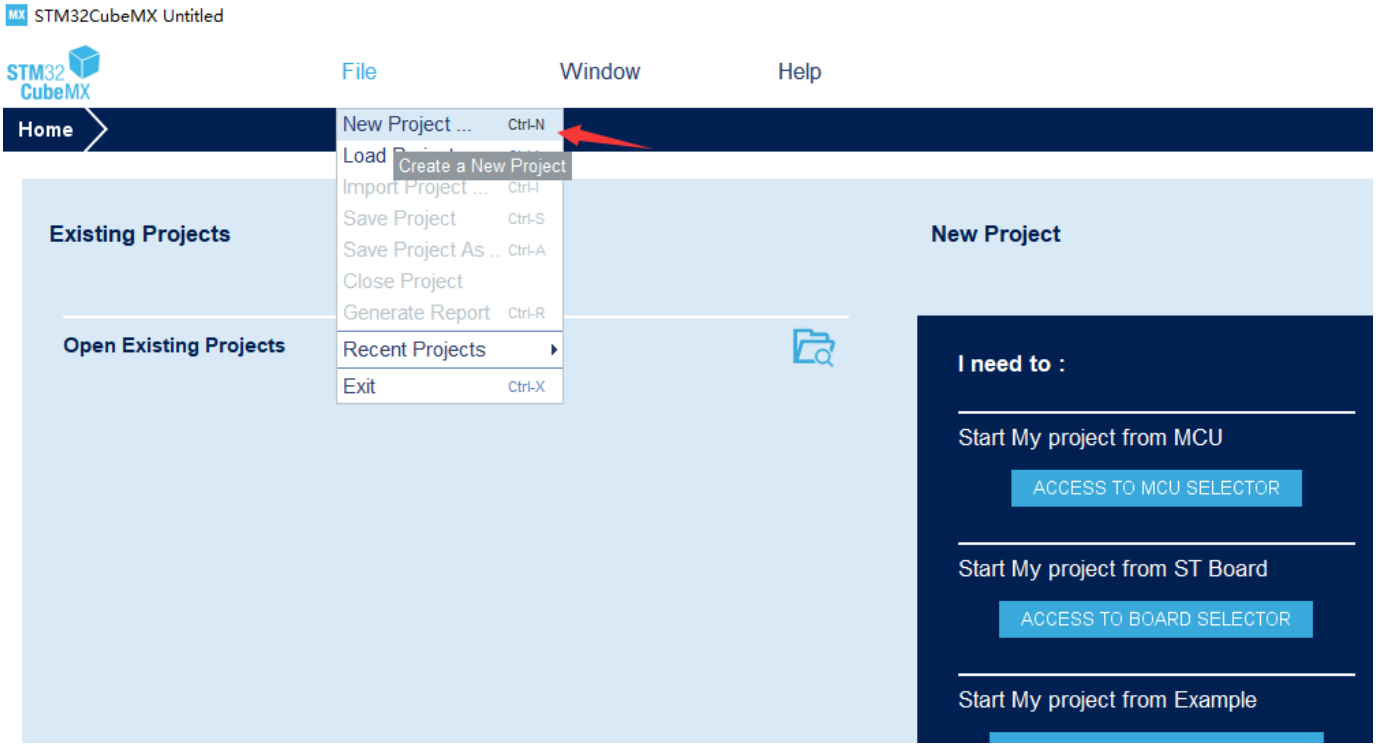
7.3 实验要求

LED 根据按键状态进行闪烁

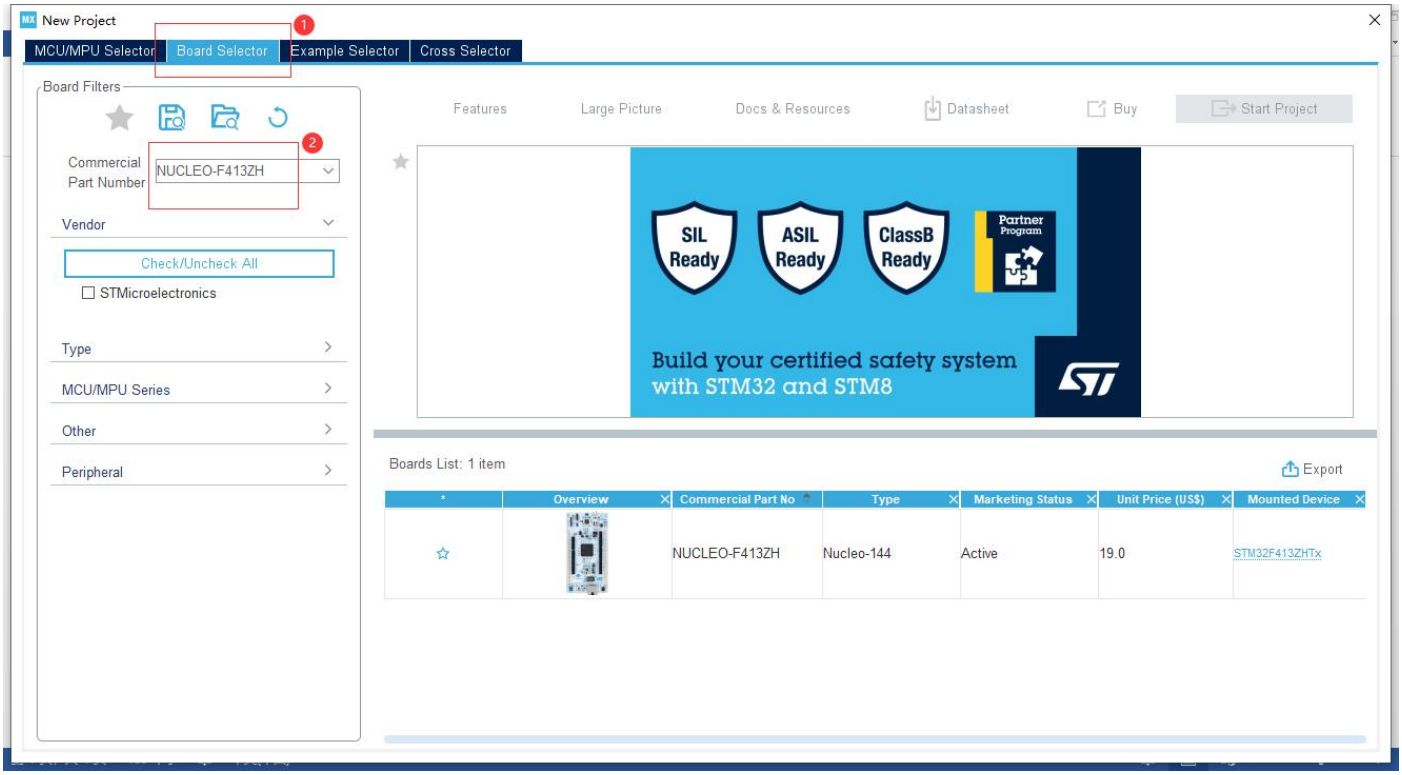
7.4 实验步骤

1. 利用 STM32CubeMX 生成模板代码

第一步，首先打开 STM32CubeMX 工具，点击如图所示的按钮新建工程。

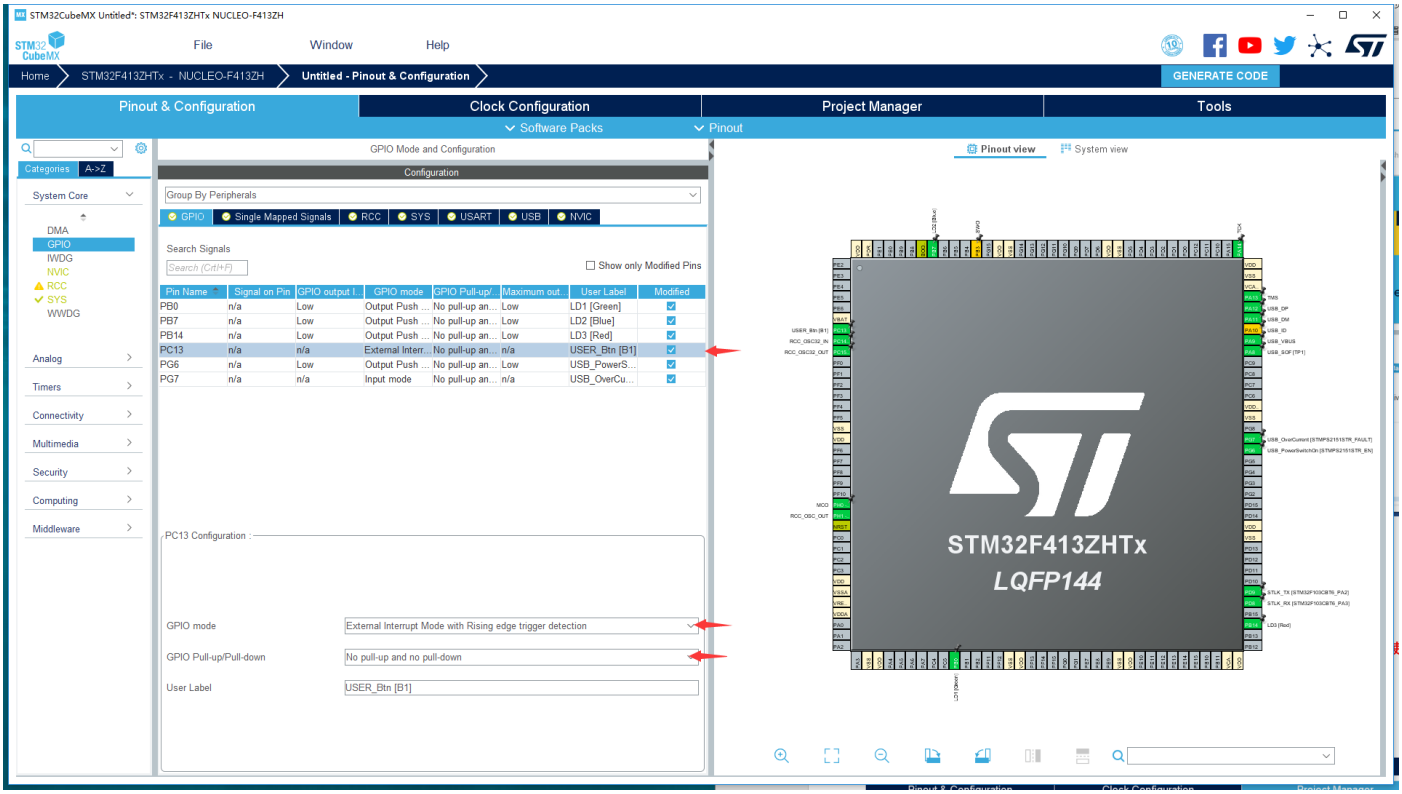


第二步，直接选择对应的 NUCLEO-144 开发板，完成基本配置。

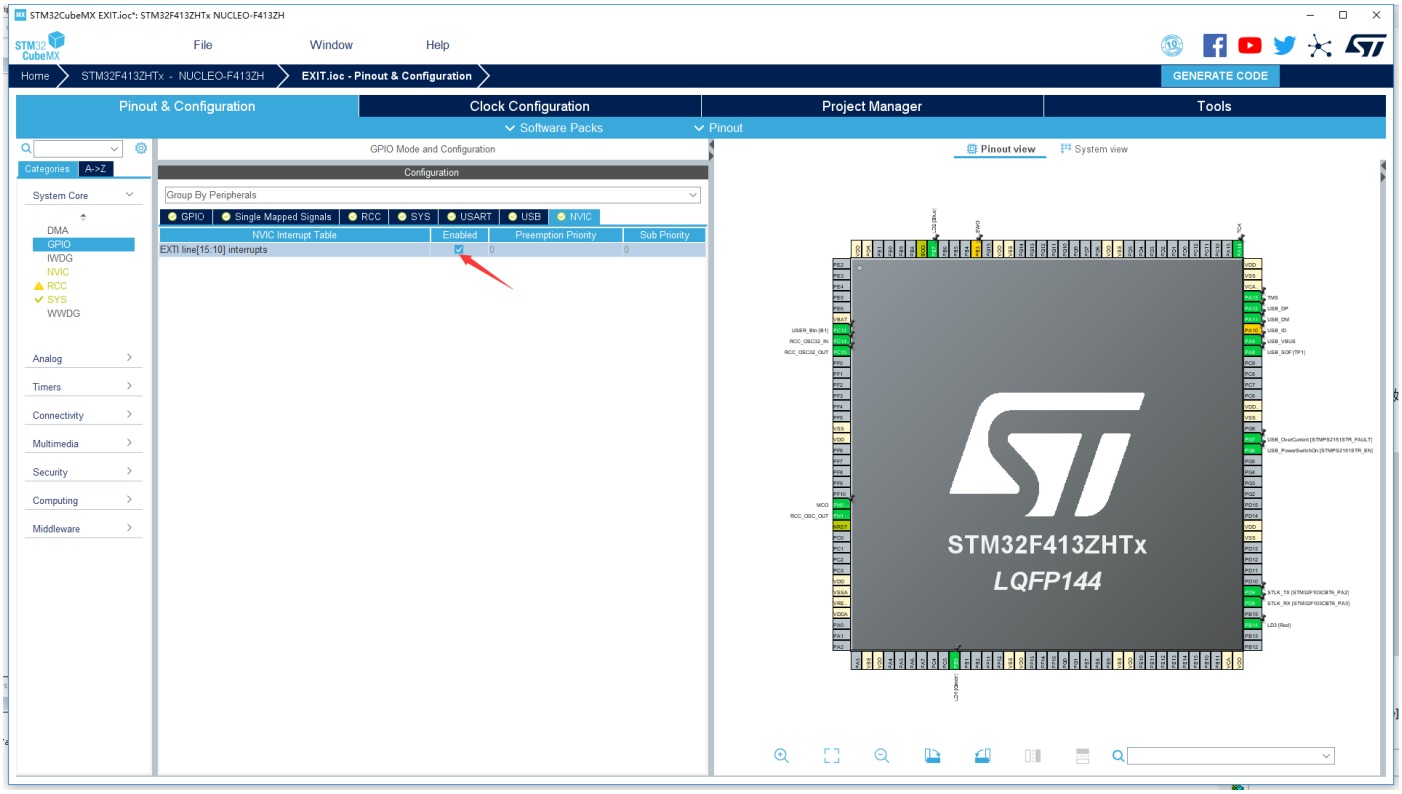


可以看到已经为我们将 PC13 初始化为外部中断模式，其中各参数含义如下：

- GPIO Mode：External Interrupt Mode with Rising edge trigger detection 外部中断模式，上升沿触发
- GPIO Pull-up/Pull-down：No pull-up and no pull-down 既不使能上拉，也不使能下拉



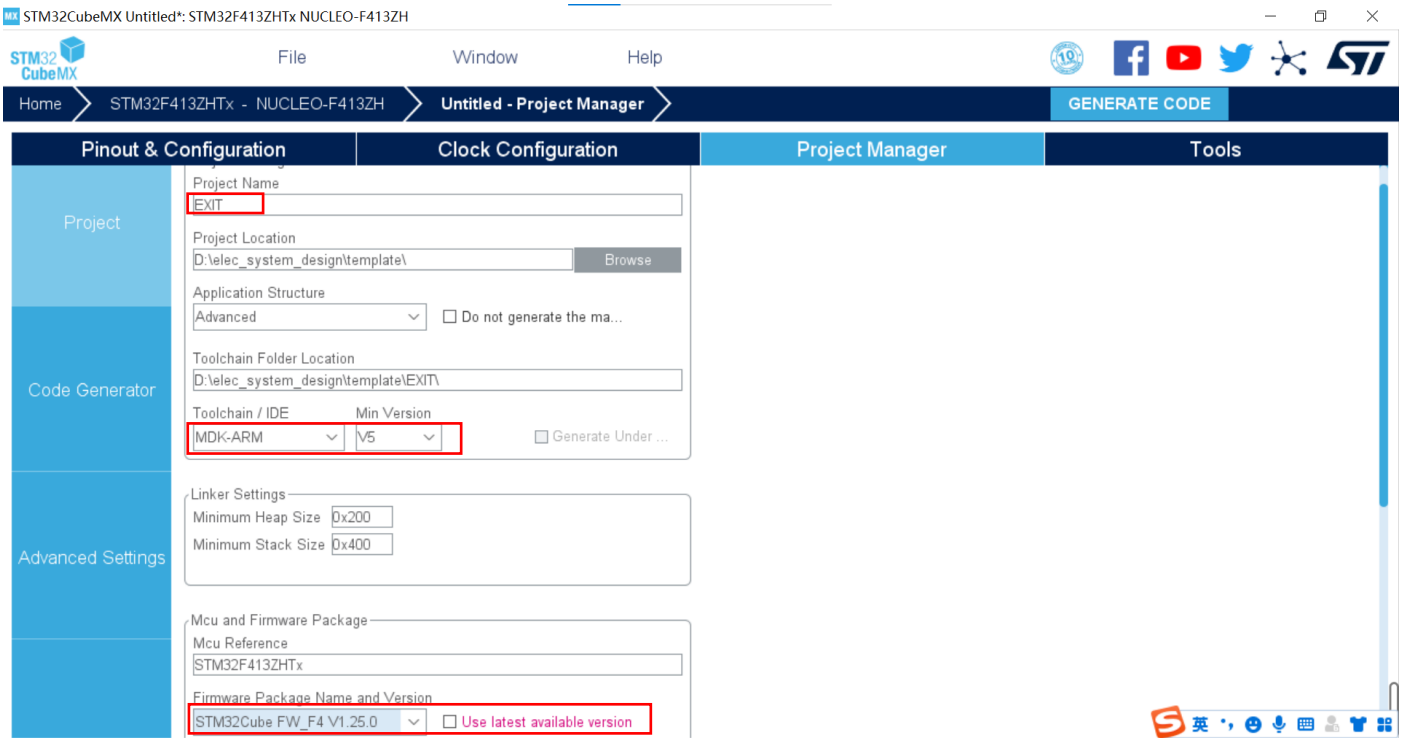
第三步，在 NVIC 界面使能中断



第四步，填写以下工程信息。

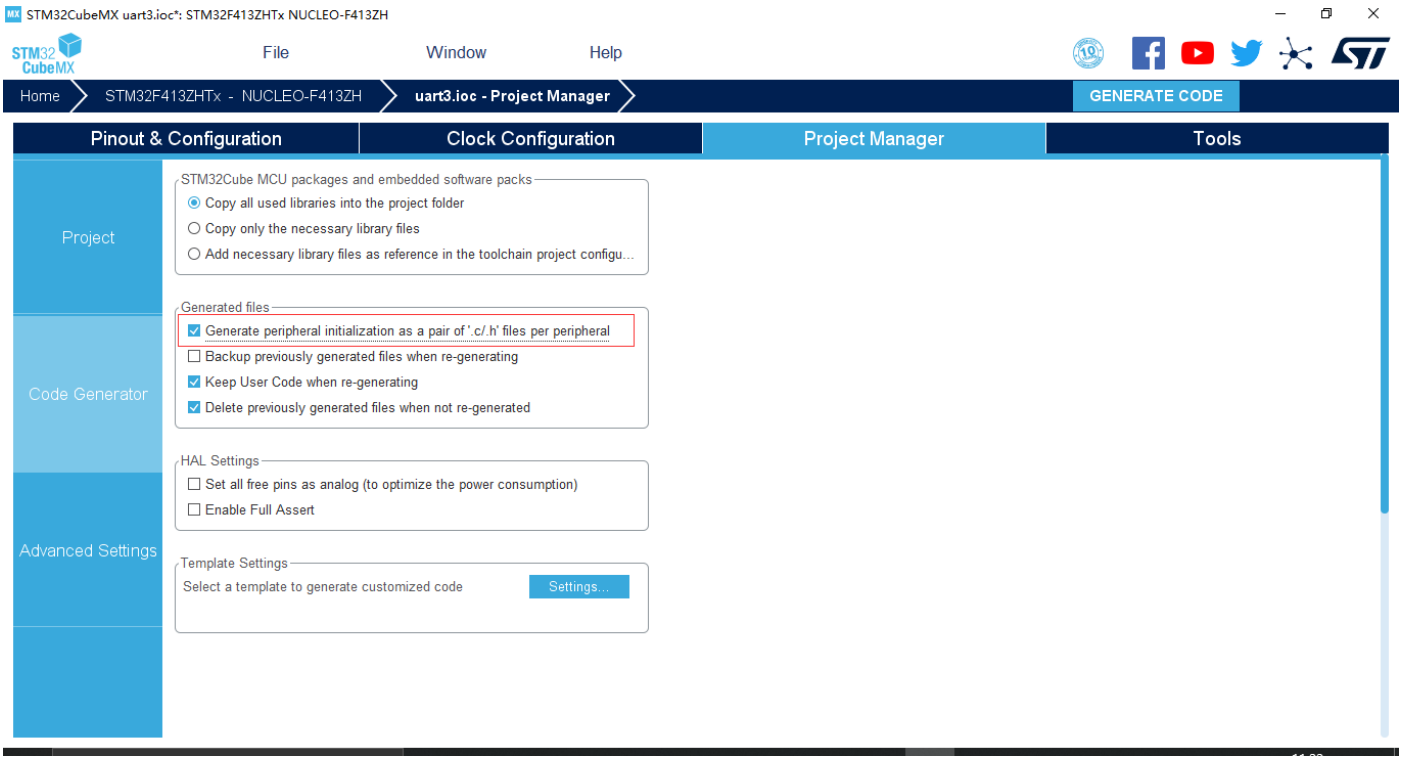
Project 栏目：

- Project Name： 工程名任意即可，这里填写 EXTI。
- Project Location： 工程路径，建议新建空文件夹专门存放所有的工程文件，这里新建了 EXTI 文件夹。注意路径中不要出现中文字符。
- Toolchain/IDE： 这里选择我们已经安装好的 Keil5，工具链（Toolchain）选择 MDK-ARM，版本号选择 V5。
- 取消勾选 Use latest available version，选择 V1.25.0。



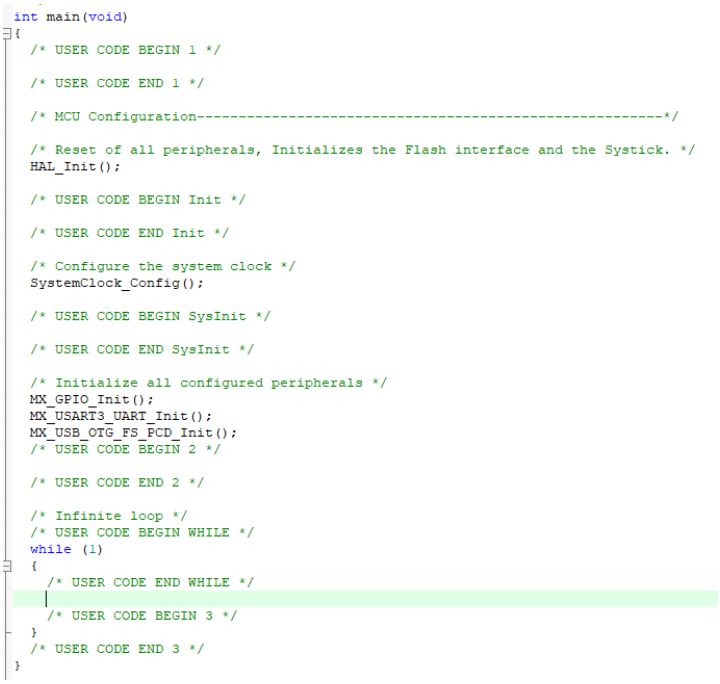
Code Generator 栏目：

- 勾选第一项，可以将外设的初始化函数等生成单独的“.h”、”.c”文件，避免函数全部放到 main.c 里面。



2. 利用 Keil 添加用户代码并烧录

第一步，编写用户代码



```
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART3_UART_Init();
  MX_USB_OTG_FS_PCD_Init();
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

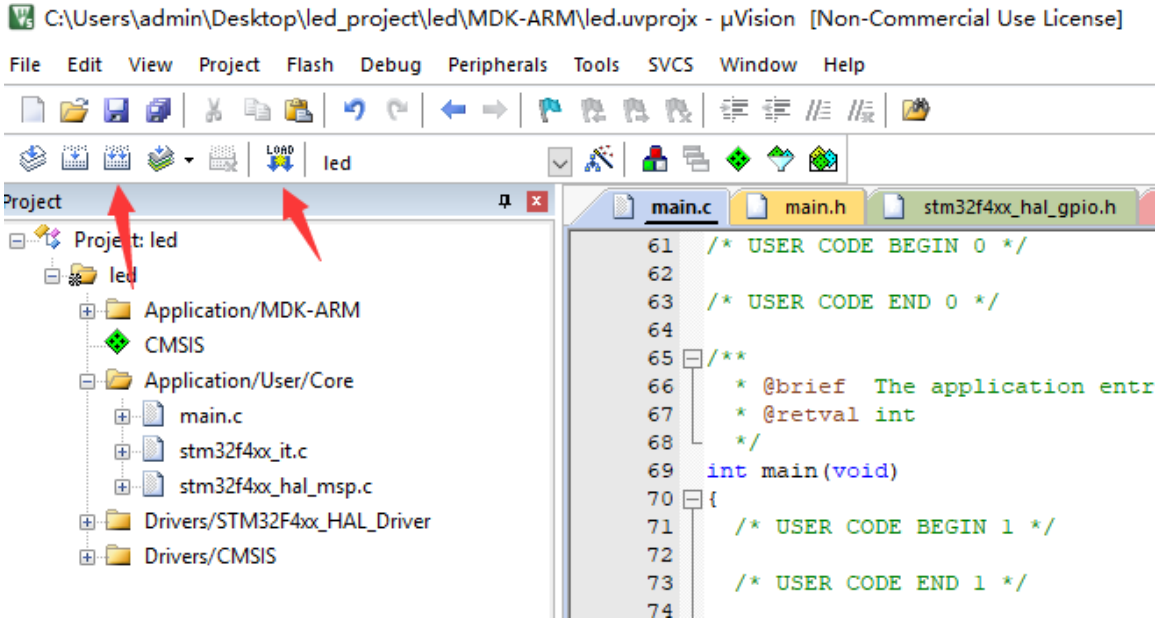
利用 Keil 打开工程以后，打开 main.c，这里是程序的入口。在 MX_GPIO_Init 函数内已经为我们初始化好了按键的外部中断。

在 USER CODE BEGIN 4 处（ 158 行），添加如下语句：

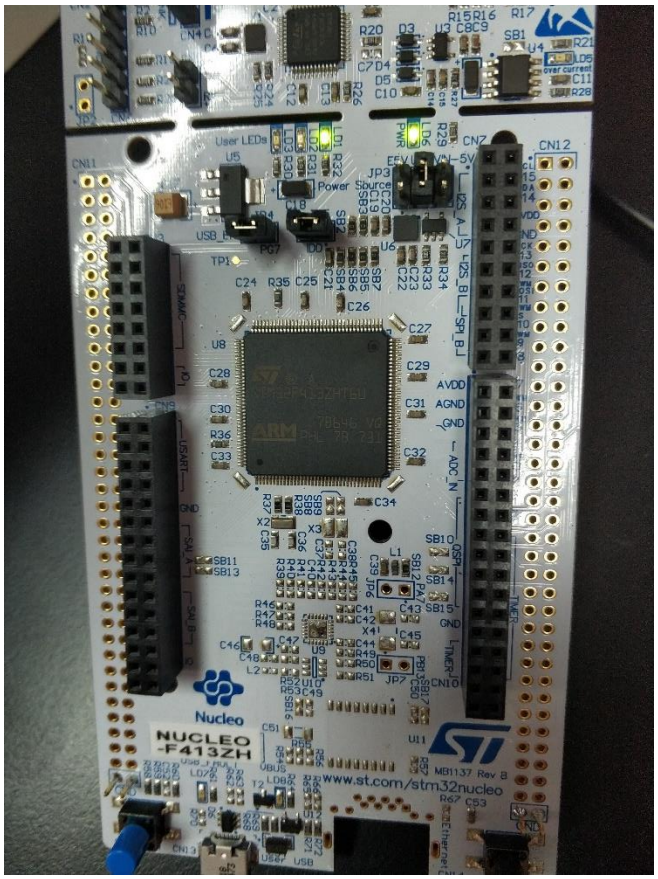
```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
        if (GPIO_Pin == GPIO_PIN_13)
  {
    /* Toggle LED1 */
```

```
    HAL_GPIO_TogglePin(LD1_GPIO_Port,LD1_Pin);
  }
}
```

第二步，利用图示的两个按钮进行编译和代码烧录。



7.5 实验结果



按动 B1，即可发现 LED1 状态改变。注意按下按键的时间不要太短，否则可能会因为存在硬件消抖而无法触发外部中断。

第八章 定时器中断实验

8.1 实验目的

1. 学习定时器中断的配置方法
2. 实现 LED 闪烁频率的设定

8.2 实验内容

通过对定时器预分频值、计时周期的设定，实现 LED 不同频率的闪烁

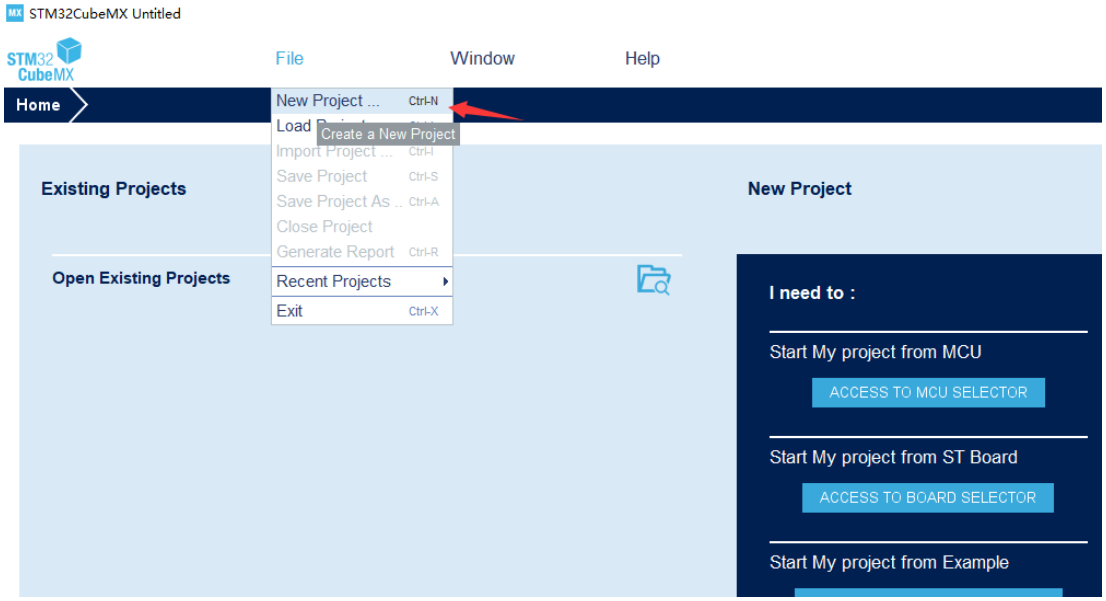
8.3 实验要求

Nucleo-144 上 LD1、LD2 分别以 500ms、1000ms 的时间间隔改变状态

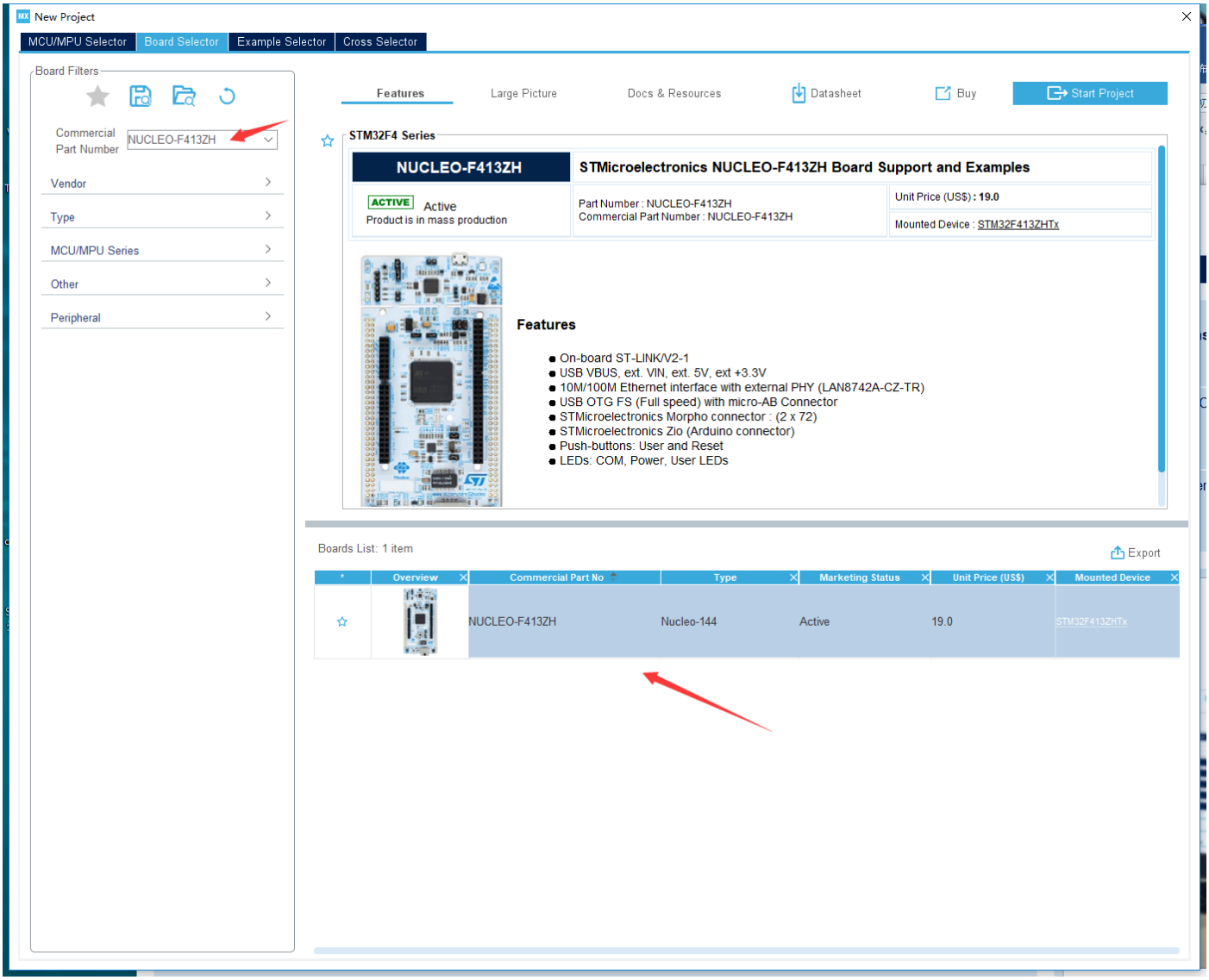
8.4 实验步骤

1. 利用 STM32CubeMX 生成模板代码

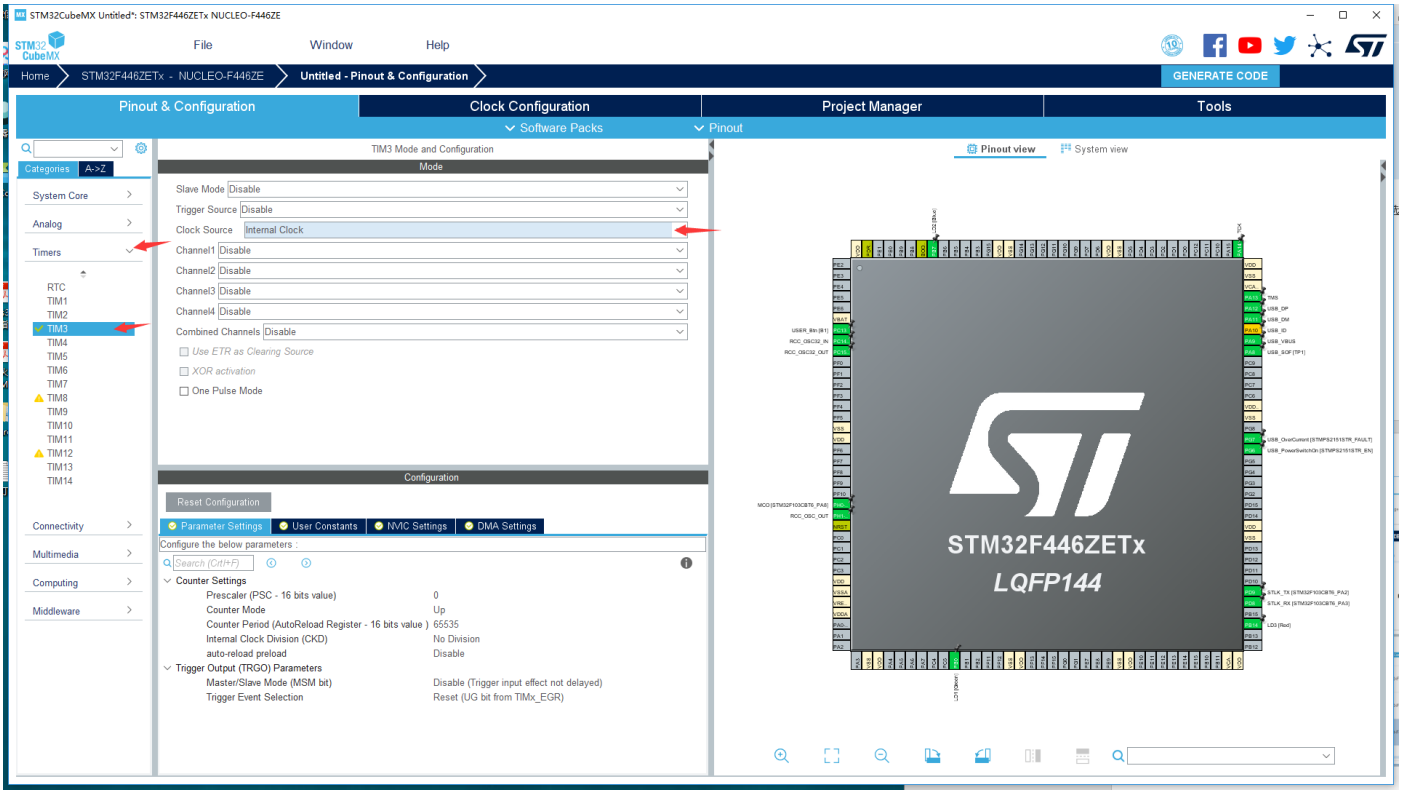
第一步，首先打开 STM32CubeMX 工具，点击如图所示的按钮新建工程。



第二步，直接选择对应的 NUCLEO-144 开发板，完成基本配置。

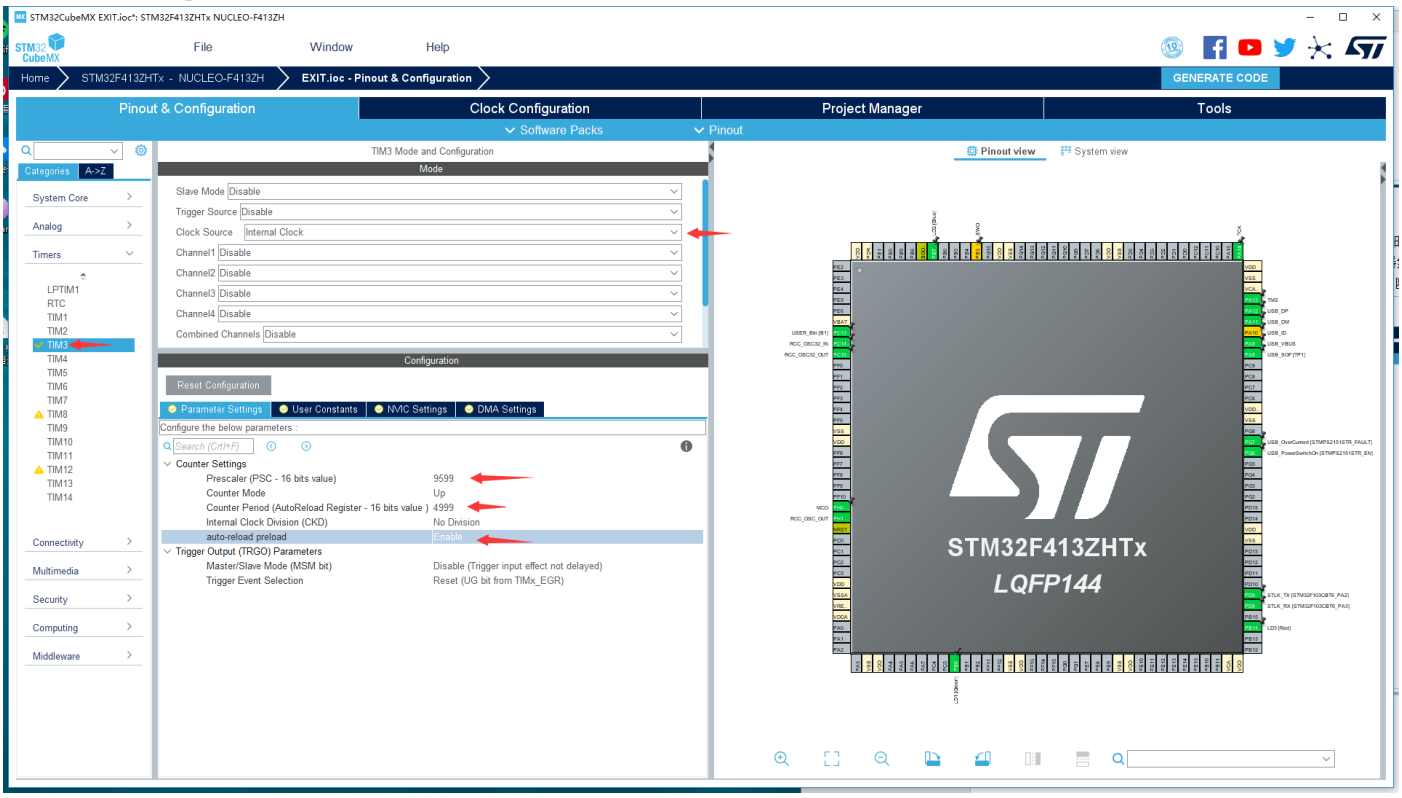


在定时器配置界面内配置 TIM3，并将时钟源修改为内部时钟。

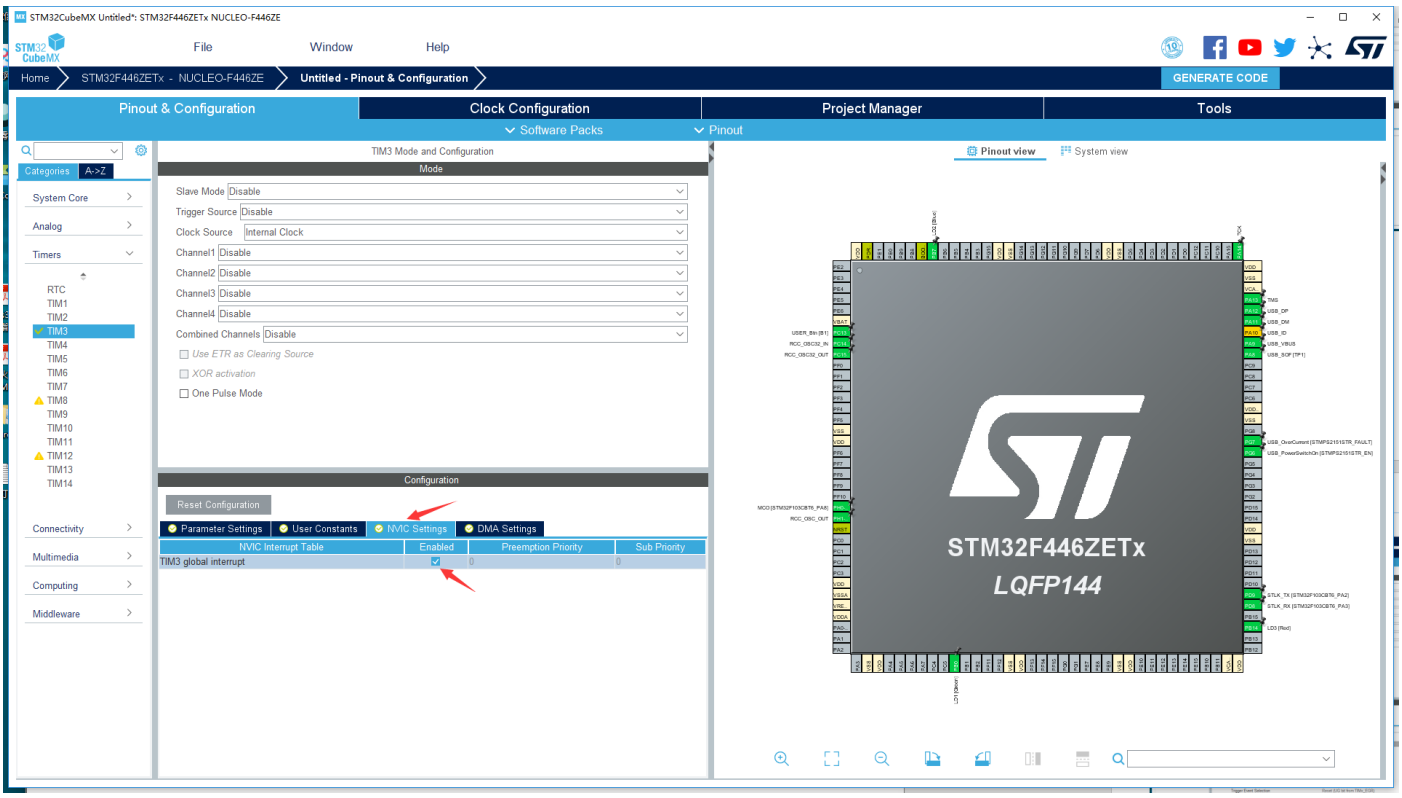


修改 TIM3 时钟参数：

- Prescaler：预分频系数，这里应填写 9599，将 96Mhz 的时钟分频为约 10000hz
- Counter Period：计数周期，这里填写 4999，使得定时器约 500ms 产生一次中断
- auto-reload period ：开启自动装载，定时器将在产生中断后继续重新计数



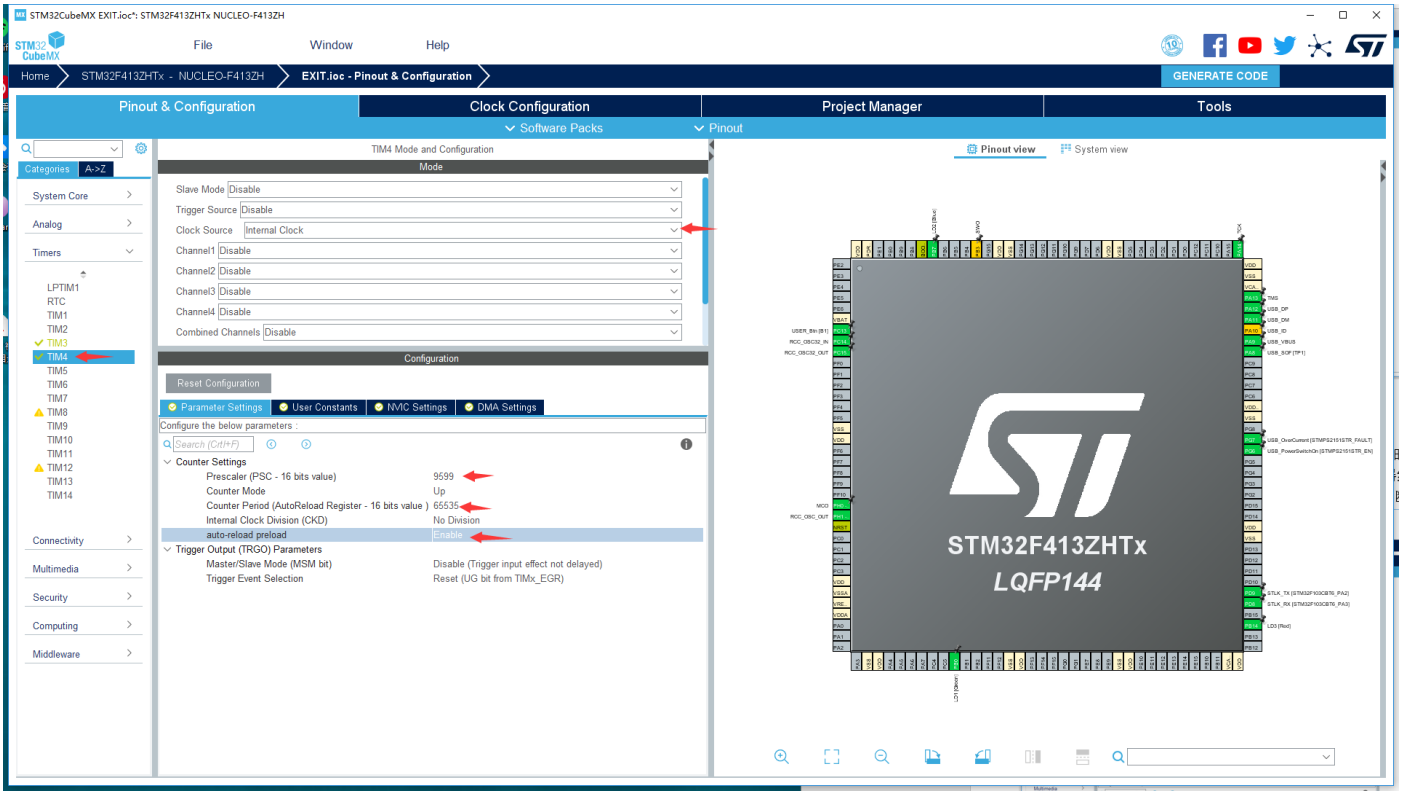
在中断配置界面将 TIM3 中断开启。



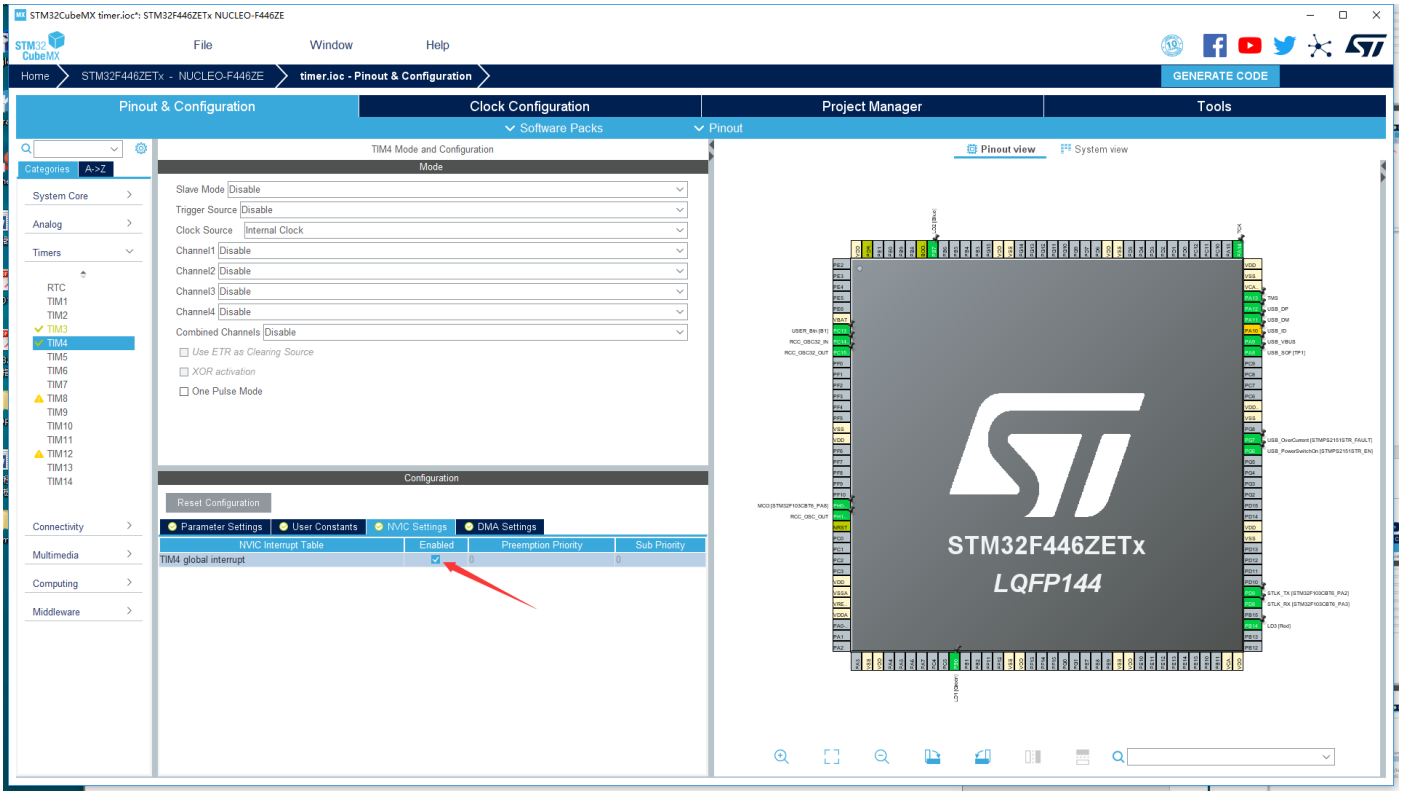
修改 TIM4 时钟参数：

- 时钟源修改为内部时钟
- Prescaler：预分频系数，这里应填写 9599，将 96Mhz 的时钟分频为约 10000hz

- Counter Period：计数周期，这里填写 9999，使得定时器约 1000ms 产生一次中断
- auto-reload period ：开启自动装载，定时器将在产生中断后继续重新计数

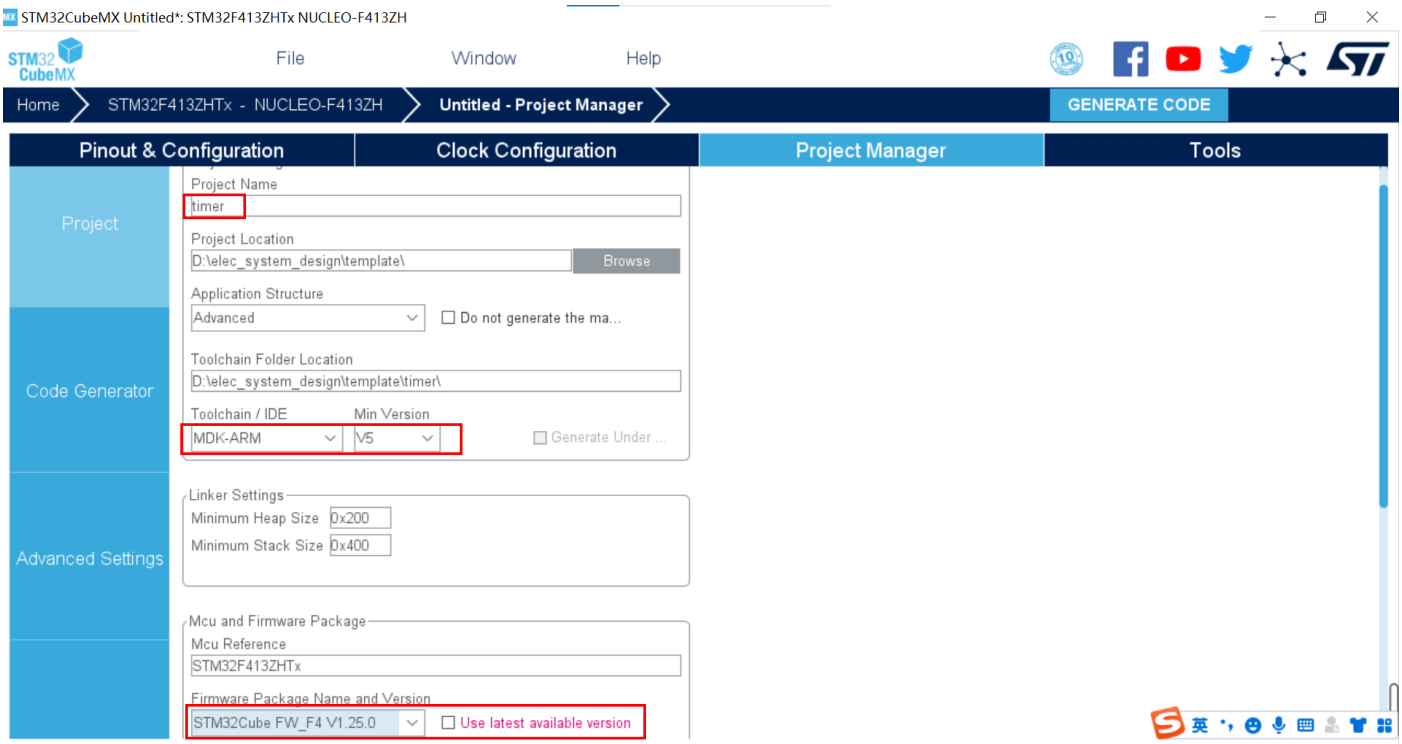


开启定时器 4 中断。



第三步，创建工程后填写一下的工程信息。

- Project Name： 工程名任意即可，这里填写 timer。
- Project Location： 工程路径，建议新建空文件夹专门存放所有的工程文件，这里在桌面新建了 timer 文件夹。注意路径中不要出现中文字符。
- Toolchain/IDE： 这里选择我们已经安装好的 Keil5，工具链（Toolchain）选择 MDK-ARM，版本号选择 V5。
- 取消勾选 Use latest available version，选择 V1.25.0。
- 其余保持默认，点击右上角的 GENERATE CODE 生成 Keil 工程。



2. 利用 Keil 添加用户代码

```
74  int main(void)
75  {
76    /* USER CODE BEGIN 1 */
77
78    /* USER CODE END 1 */
79
80    /* MCU Configuration--------------------------------------------------------*/
81
82    /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
83    HAL_Init();
84
85    /* USER CODE BEGIN Init */
86
87    /* USER CODE END Init */
88
89    /* Configure the system clock */
90    SystemClock_Config();
91
92    /* USER CODE BEGIN SysInit */
93
94    /* USER CODE END SysInit */
95
96    /* Initialize all configured peripherals */
97    MX_GPIO_Init();
98    MX_TIM3_Init();
99    MX_USART3_UART_Init();
100   MX_USB_OTG_FS_PCD_Init();
101   MX_TIM4_Init();
102   /* USER CODE BEGIN 2 */
103
104   /* USER CODE END 2 */
105
106   /* Infinite loop */
107   /* USER CODE BEGIN WHILE */
108   while (1)
109   {
110     /* USER CODE END WHILE */
111
112     /* USER CODE BEGIN 3 */
113   }
114   /* USER CODE END 3 */
115 }
```

利用 Keil 打开工程以后，打开 main.c，这里是程序的入口。可以发现 STM32cube 已经帮我们完成了关于 LED 引脚的初始化，并且在下方留出了空间让用户添加用户代码。

在 main 函数内 while 上方添加如下语句，开启定时器中断。

```
HAL_TIM_Base_Start_IT(&htim3);
HAL_TIM_Base_Start_IT(&htim4);
```

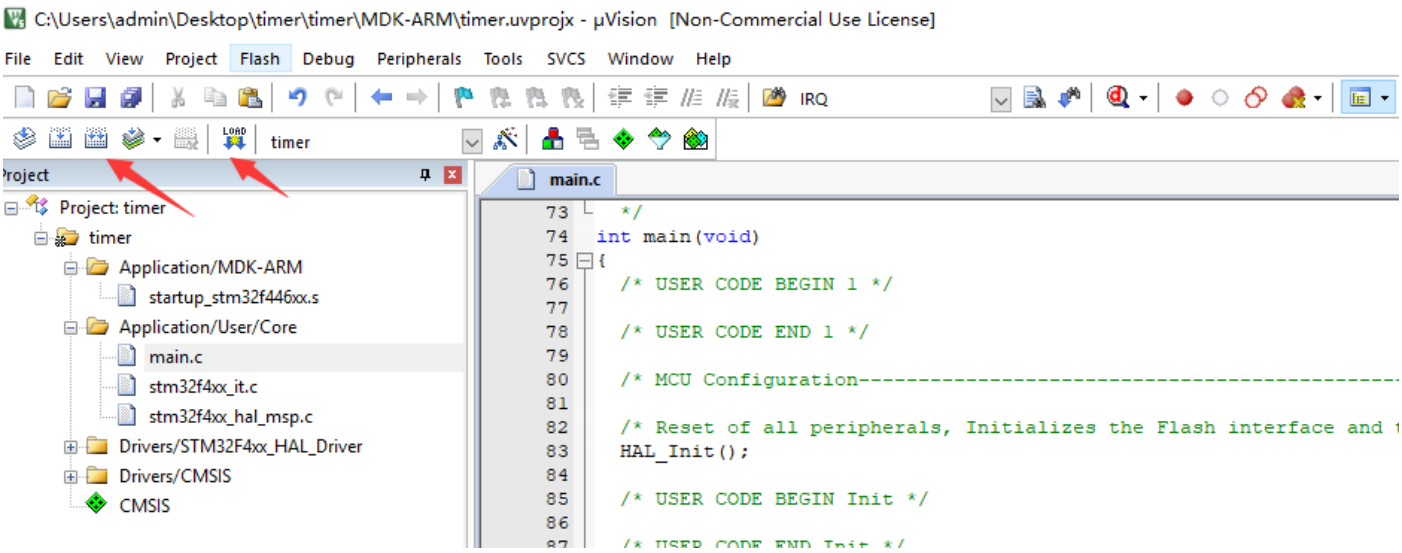
在 main 函数结束处下方添加如下语句。

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    if( htim == &htim3) HAL_GPIO_TogglePin(LD2_GPIO_Port,LD2_Pin);
    else if ( htim == &htim4) HAL_GPIO_TogglePin(LD3_GPIO_Port,LD3_Pin);
}
```

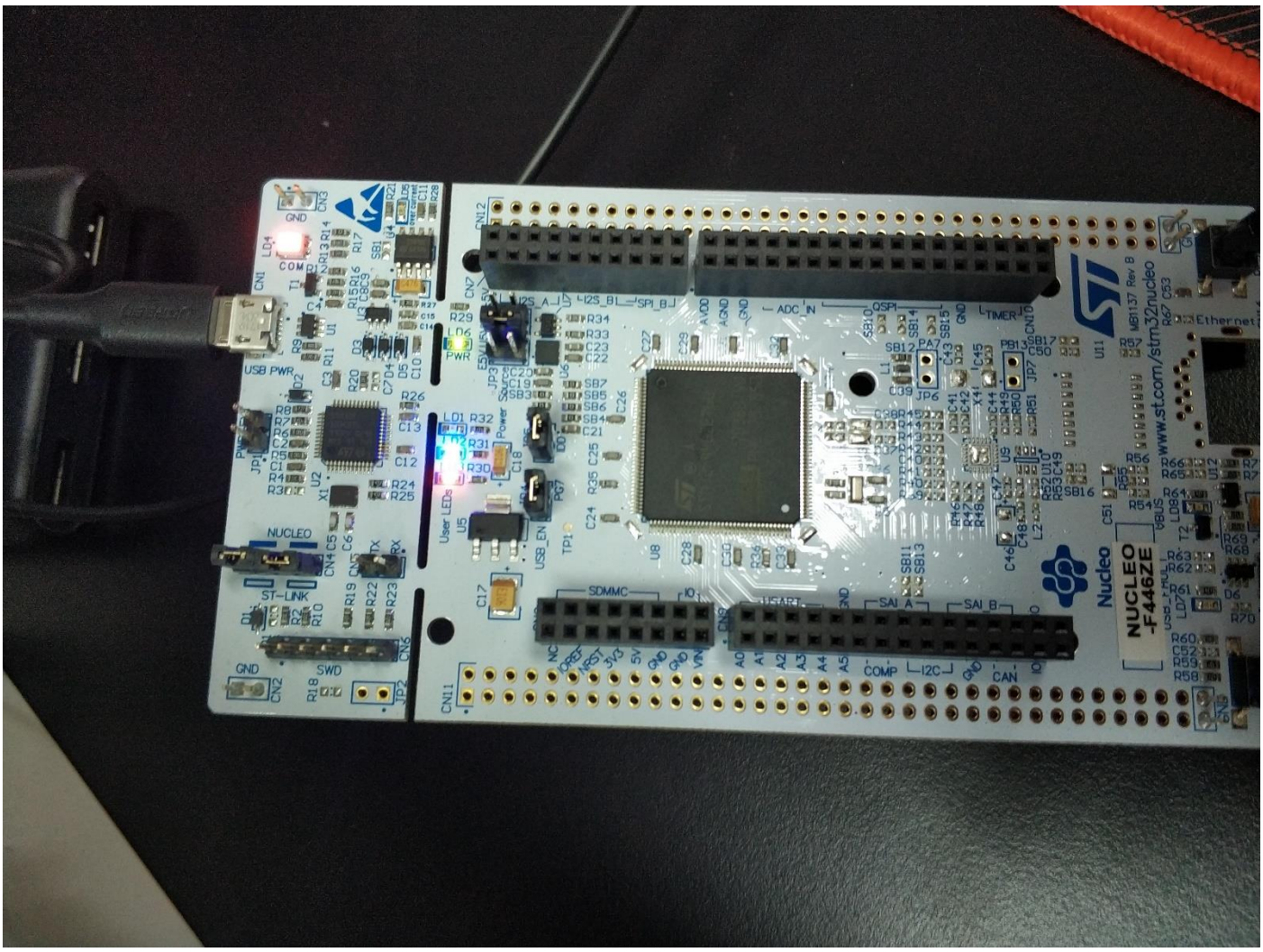
修改完的代码如图

```c
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_TIM3_Init();
  MX_USART3_UART_Init();
  MX_USB_OTG_FS_PCD_Init();
  MX_TIM4_Init();
  /* USER CODE BEGIN 2 */
  HAL_TIM_Base_Start_IT(&htim3);
  HAL_TIM_Base_Start_IT(&htim4);

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  if( htim == &htim3) HAL_GPIO_TogglePin(LD2_GPIO_Port,LD2_Pin);
  else if ( htim == &htim4) HAL_GPIO_TogglePin(LD3_GPIO_Port,LD3_Pin);
}
```

利用图示的两个按钮进行编译和代码烧录。



8.5 实验结果



代码烧录完成之后，按动开发板右下角的 RESET 按钮即可开始运行程序，可见 LED2、LED3 开始以不同频率闪烁，LED2 每 500ms 改变一次状态，LED3 每 1000ms 改变一次状态。

第九章 PWM 输出实验

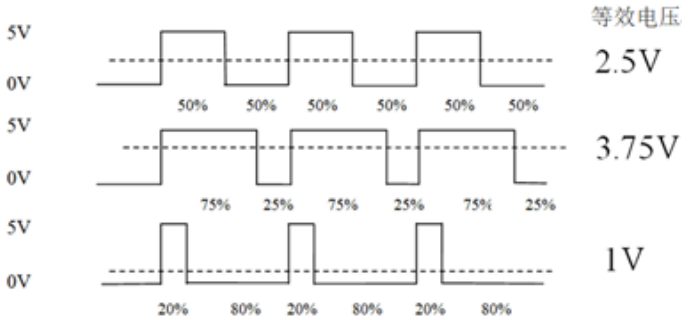
9.1 实验目的

1. 学习 PWM 原理与应用
2. 进一步学习定时器中断的配置方法
3. 利用程序改变寄存器 CCRx 的值，产生不同占空比的 PWM 波

9.2 实验原理

Pulse Width Modulation 脉冲宽度调制，简称 PWM。

PWM 对模拟信号电平进行数字编码的方法，计算机只能输出 0 或 5V 的数字电压值而不能输出模拟电压，而我们如果想获得一个模拟电压值(介于 0 - 5V 的电压值)，则需通过使用高分辨率计数器，改变方波的占空比来对一个模拟信号的电平进行编码。电压是以一种连接（1）或断开（0）的重复脉冲序列被夹到模拟负载上去的，连接即是直流供电输出，断开即是直流供电断开。通过对连接和断开时间的控制，只要带宽足够，可以输出任意不大于最大电压值的模拟电压。



我们假定定时器工作在向上计数 PWM 模式，且当 CNT<CCRx 时，输出 0，当 CNT>=CCRx 时输出 1。那么就可以得到如上的 PWM 示意图：当 CNT 值小于 CCRx 的时候， IO 输出低电平(0)，当 CNT 值大于等于 CCRx 的时候，IO 输出高电平(1)， 当 CNT 达到 ARR 值的时候，重新归零，然后重新向上计数，依次循环。改变 CCRx 的值，就可以改变 PWM 输出的占空比，改变 ARR 的值，就可以改变 PWM 输出的频率。

9.3 实验内容

本章通过对定时器内部寄存器 CCRx 数值的修改，生成占空比先递增后递减的 PWM，进而控制 LD1，达到呼吸灯的效果

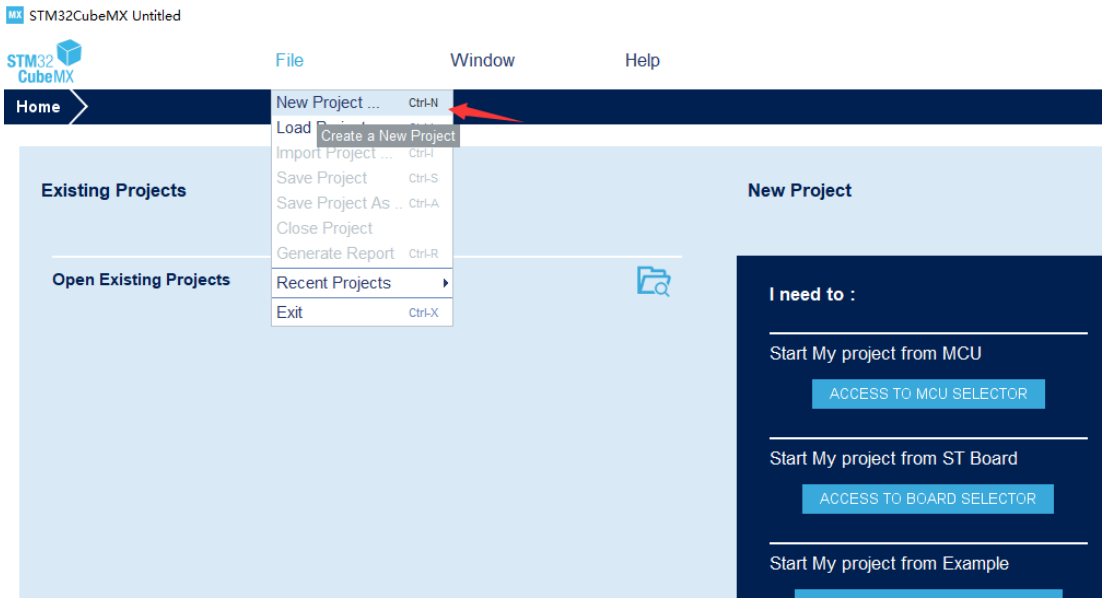
9.4 实验要求

Nucleo-144 上 LD1 能看到暗——亮——暗的循环呼吸效果

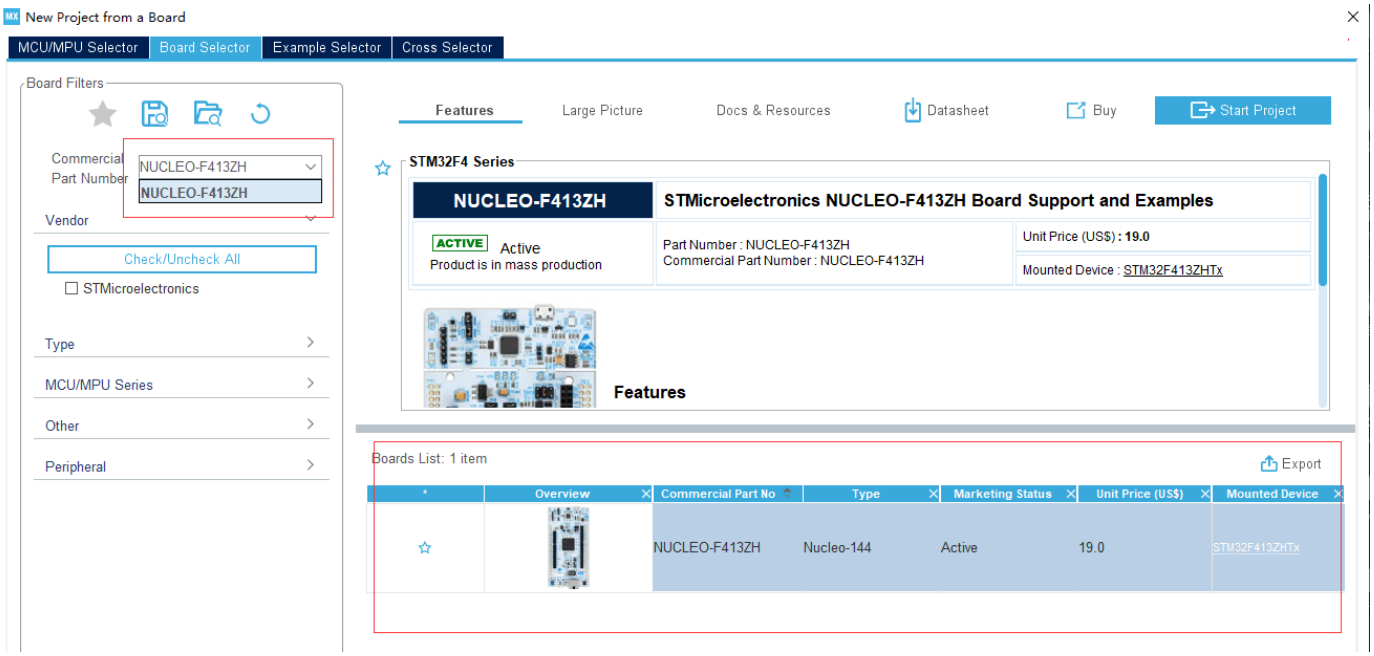
9.5 实验步骤

1. 利用 STM32CubeMX 生成模板代码

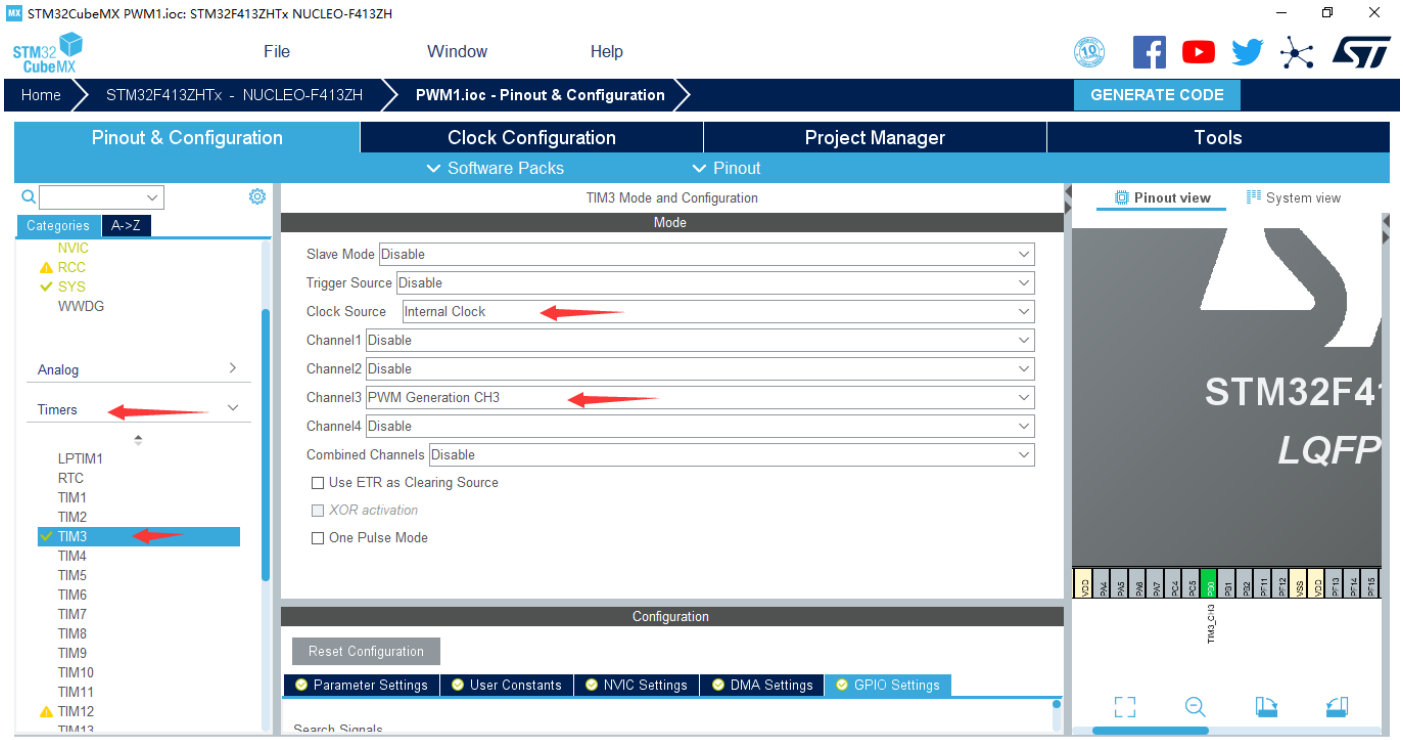
第一步，首先打开 STM32CubeMX 工具，点击如图所示的按钮新建工程。



第二步，直接选择对应的 NUCLEO-144 开发板，选择完成基本配置。



在定时器配置界面内配置 TIM3，并将时钟源修改为内部时钟，并选择通道 Channel3，以 PWM 形式输出。



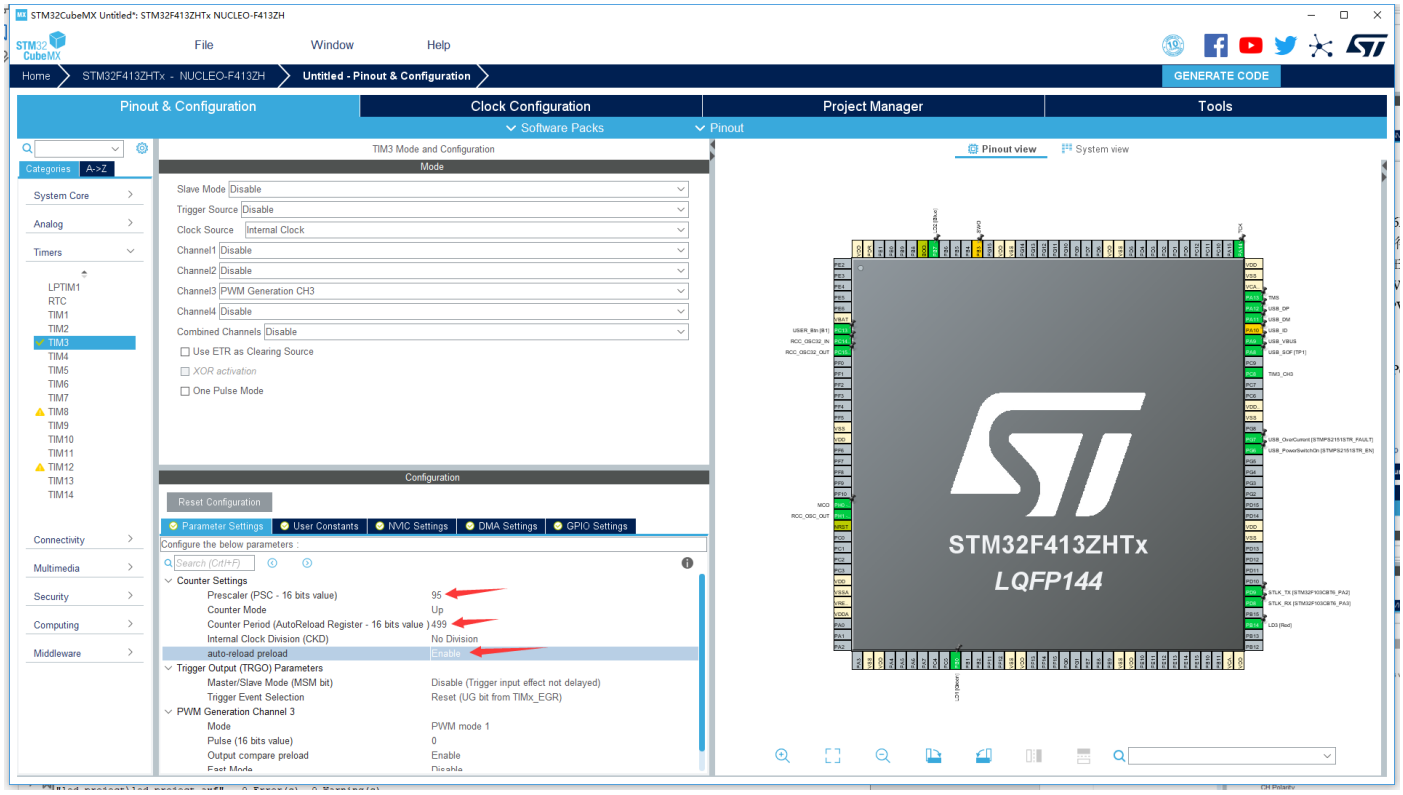
修改 TIM3 时钟参数：

- Prescaler：预分频系数，这里应填写 95，将 96Mhz 的时钟分频为 1000khz
- Counter Period：计数周期，这里填写 499，使得定时器产生中断的频率为 2 khz，该频率不可过小
- auto-reload period：开启自动装载，定时器将在产生中断后继续重新计数
- Mode：选择 PWM mode1，PWM mode1 与 PWM mode2 的输出电平的极性相反
- Pulse：通过在程序中修改此数据，改变输出 PWM 的占空比，这里保持默认。

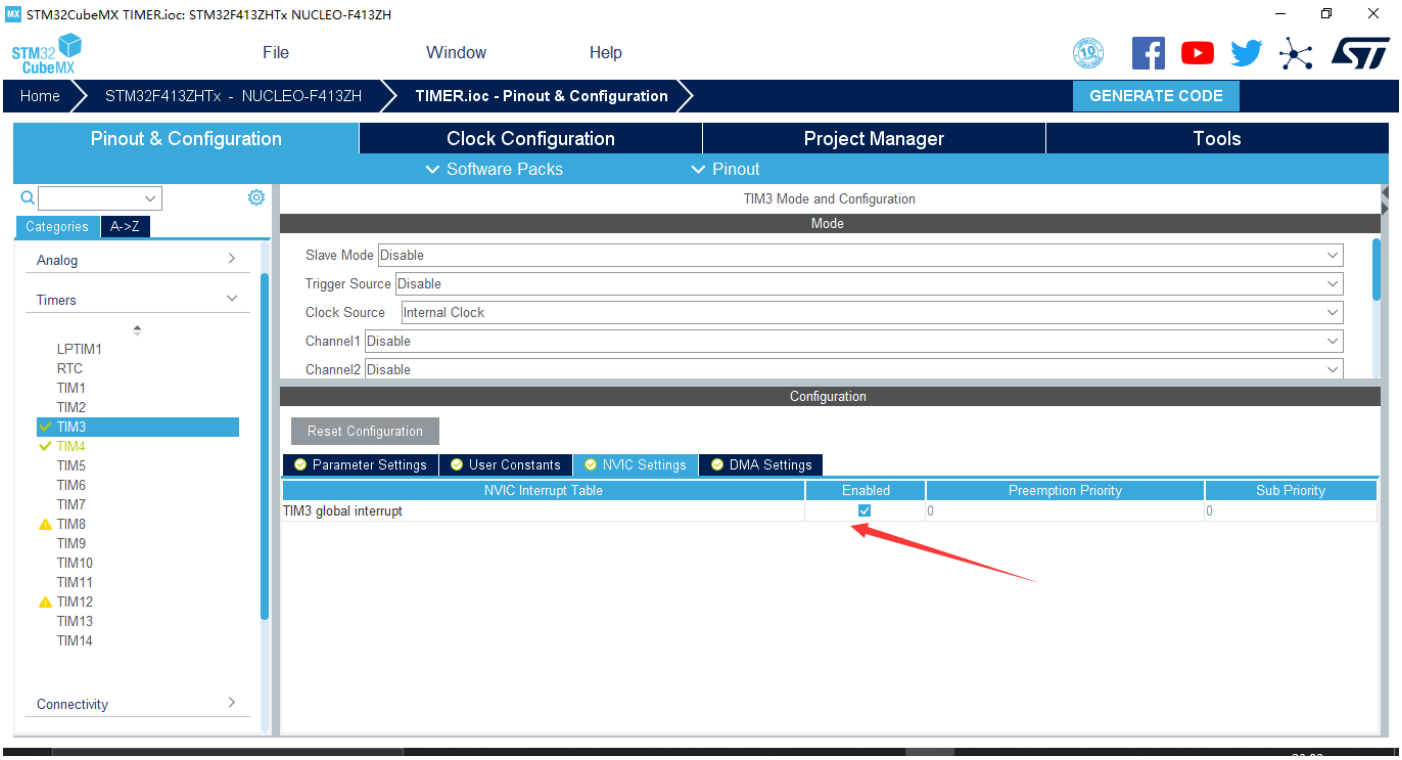
基本计算：

频率= 定时器时钟/ （Prescaler + 1）/ （Counter Period + 1）Hz

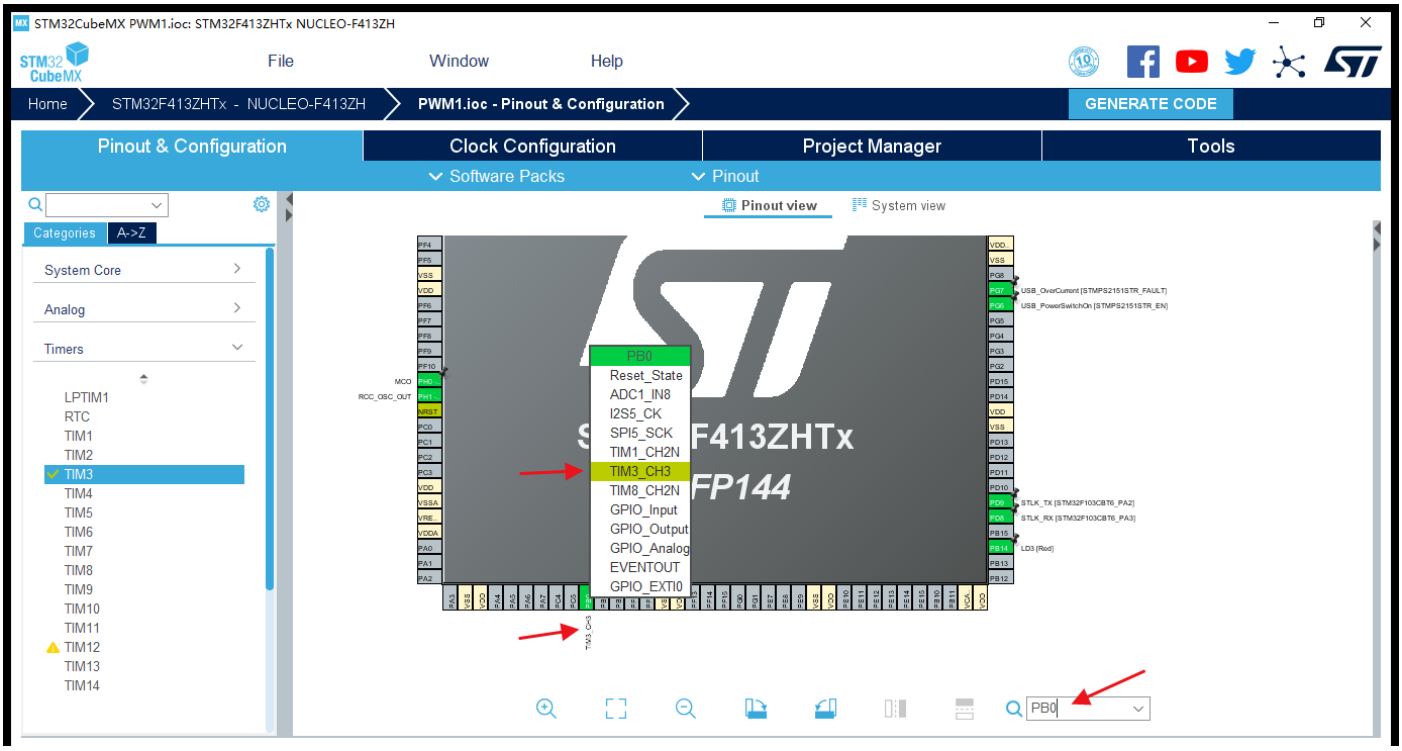
占空比= Pulse / Counter Period



在中断配置界面将 TIM3 中断开启。

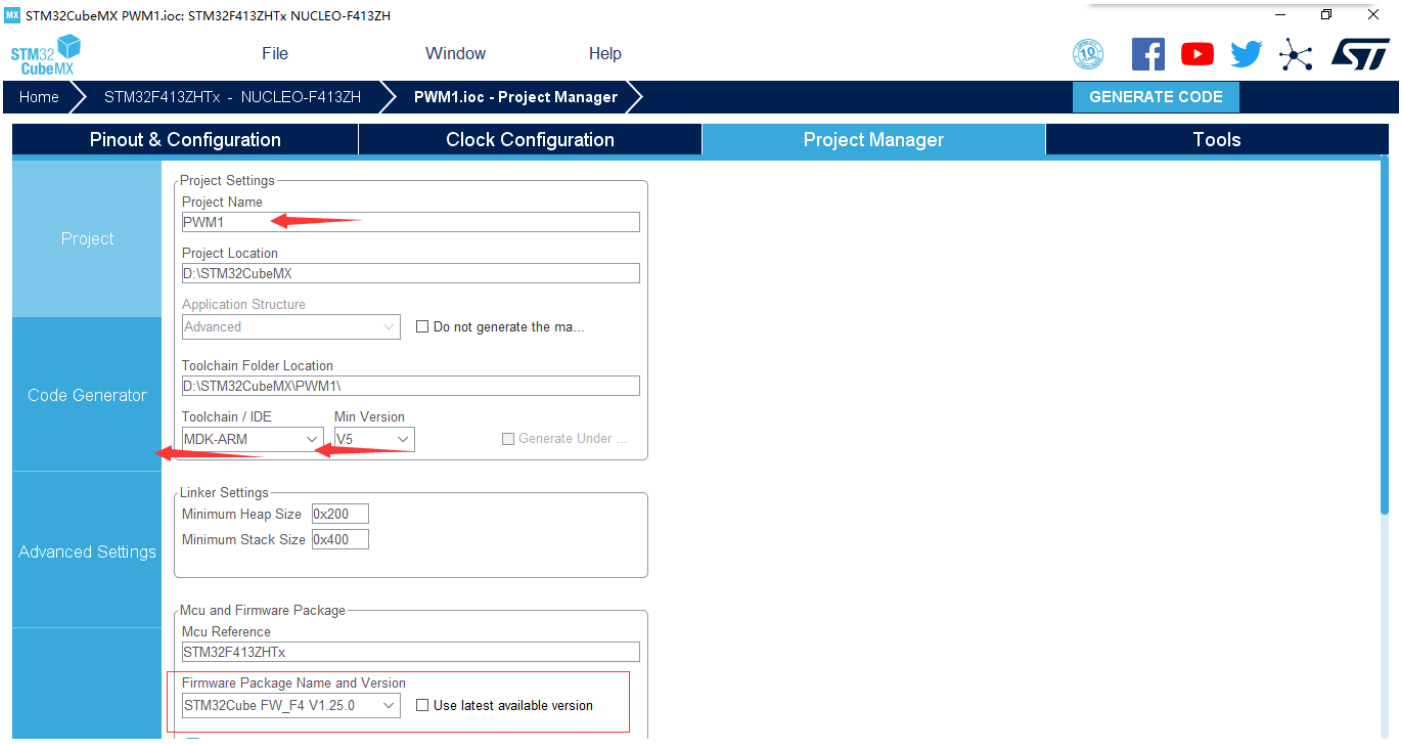


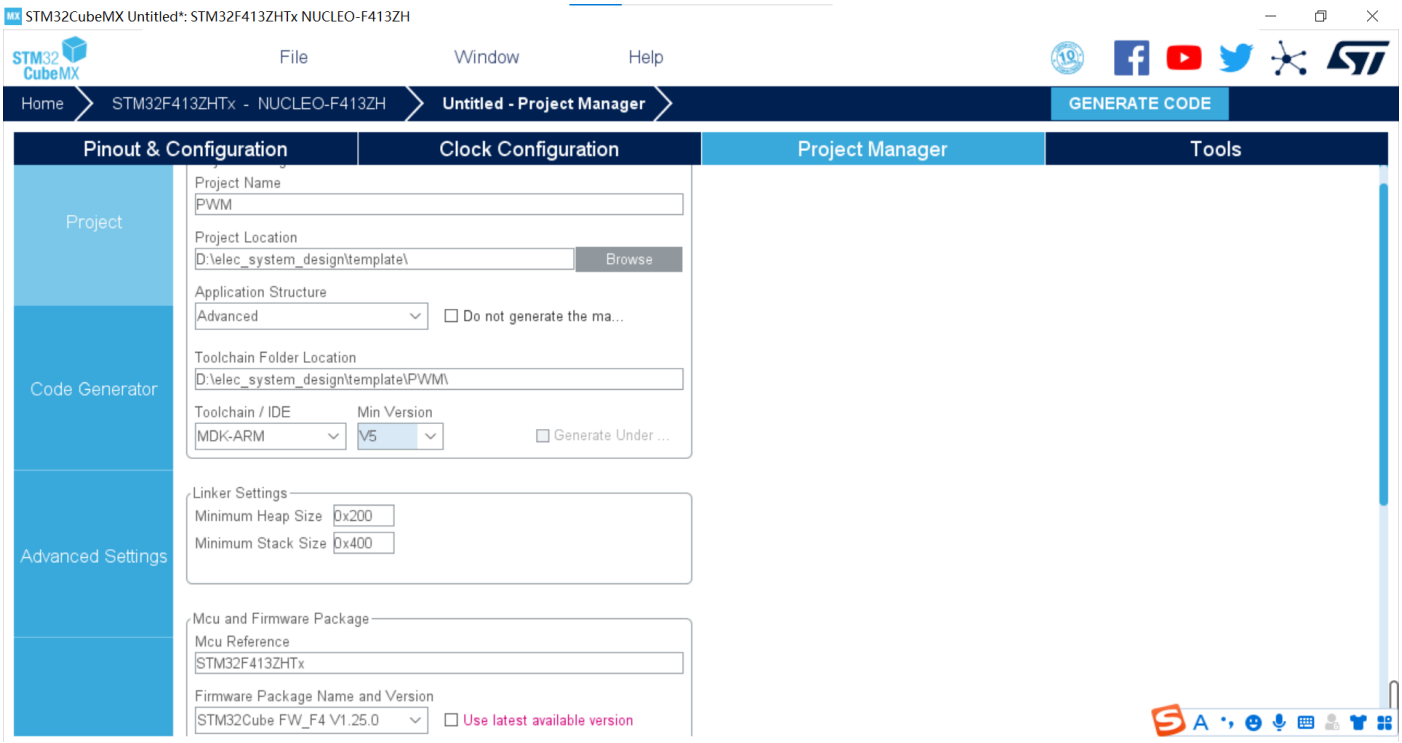
选择 GPIO 口 PB0 为 TIM3_CH3，同时 PB0 能够控制 LD1，这里为复用 PB0 端口



第三步，创建工程后填写一下的工程信息。

- Project Name： 工程名任意即可，这里填写 PWM。
- Project Location： 工程路径，建议新建空文件夹专门存放所有的工程文件，注意路径中不要出现中文字符。
- Toolchain/IDE： 这里选择我们已经安装好的 Keil5，工具链（Toolchain）选择 MDK-ARM，版本号选择 V5。
- 取消勾选 Use latest available version，选择 V1.25.0。
- 其余保持默认，点击右上角的 GENERATE CODE 生成 Keil 工程。





2. 利用 Keil 添加用户代码

```
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_TIM3_Init();
  MX_USART3_UART_Init();
  MX_USB_OTG_FS_PCD_Init();
  MX_TIM4_Init();
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

利用 Keil 打开工程以后，打开 main.c，这里是程序的入口。可以发现 STM32cube 已经帮我们完成了关于 LED 引脚的初始化，并且在下方留出了空间让用户添加用户代码。

在 USER CODE BEGIN 1 处，添加如下语句：

```
uint16_t  led0pwmval=0;
```

在 USER CODE BEGIN 2 处，添加如下语句：

```
HAL_TIM_PWM_Start(&htim3,TIM_CHANNEL_3);        /// 使能 tim3 的通道 3
```

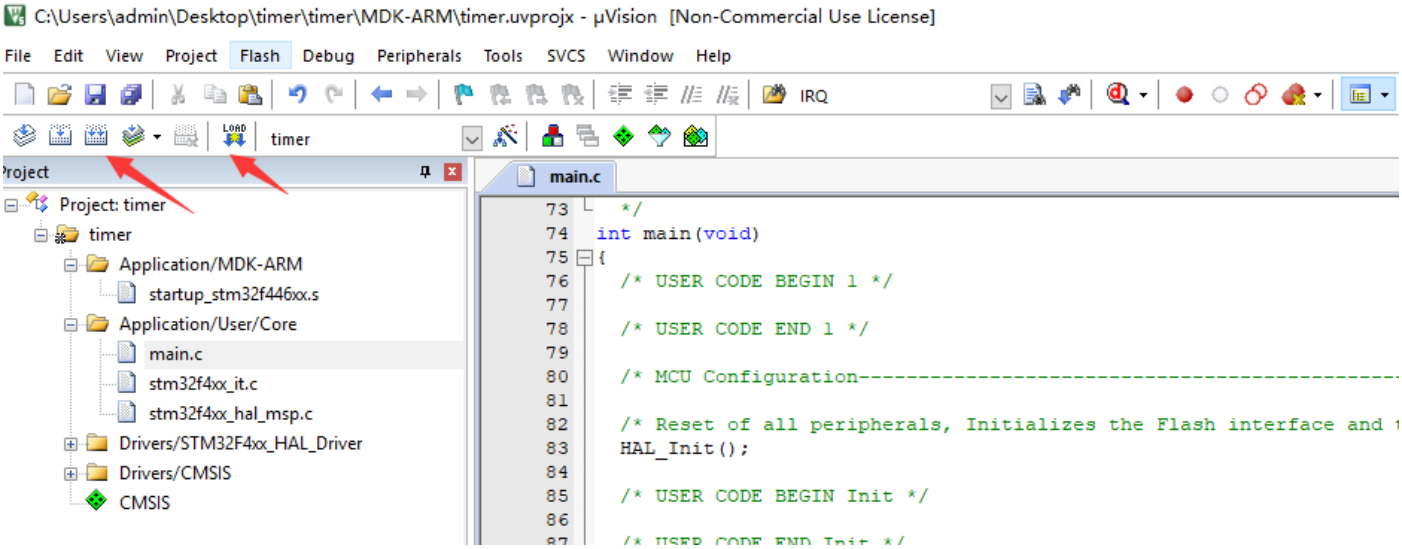
在 USER CODE BEGIN 3 处，添加如下语句：

```
while (led0pwmval< 300)                //递增到 300
  {
      led0pwmval++;
      __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_3, led0pwmval);      //改变比较值，改变占空比
//    TIM3->CCR3 = led0pwmval;      //与上一行代码作用相同，两者任选其一
      HAL_Delay(1);
  }
  while (led0pwmval)                   //再递减到 0
  {
      led0pwmval--;
      __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_3, led0pwmval);      //改变比较值，改变占空比
//    TIM3->CCR3 = led0pwmval;       //与上一行代码作用相同，两者任选其一
      HAL_Delay(1);
  }
```

修改完的代码如图

```c
int main(void)
{
  /* USER CODE BEGIN 1 */
    uint16_t  led0pwmval=0;

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_TIM3_Init();
  MX_USART3_UART_Init();
  MX_USB_OTG_FS_PCD_Init();
  /* USER CODE BEGIN 2 */

  HAL_TIM_PWM_Start(&htim3,TIM_CHANNEL_3);

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */

    while (led0pwmval< 300)
    {
      led0pwmval++;
      __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_3, led0pwmval);
//      TIM3->CCR1 = pwmVal;
      HAL_Delay(1);
    }
    while (led0pwmval)
    {
      led0pwmval--;
      __HAL_TIM_SetCompare(&htim3, TIM_CHANNEL_3, led0pwmval);
//      TIM3->CCR1 = pwmVal;
      HAL_Delay(1);
    }
  }
  /* USER CODE END 3 */
}
```

利用图示的两个按钮进行编译和代码烧录。



9.6 实验结果

代码烧录完成之后，按动开发板右下角的 RESET 按钮即可开始运行程序，可见 LD1 开始以呼吸灯的形式进行闪烁。

第十章 DAC 实验

10.1 实验目的

1. 学习 DAC 模块的工作原理
2. 实现对 Nucleo-144 开发板 DAC 模块的控制
3. 学会使用 STM32CubeMX 工具配置 DAC

10.2 实验原理

STM32F4 的 DAC 模块(数字/模拟转换模块)是 12 位数字输入，电压输出型的 DAC。DAC 可以配置为 8 位或 12 位模式，也可以与 DMA 控制器配合使用。DAC 工作在 12 位模式时，数据可以设置成左对齐或右对齐。DAC 模块有 2 个输出通道，每个通道都有单独的转换器。在双 DAC 模式下，2 个通道可以独立地进行转换，也可以同时进行转换并同步地更新 2 个通道的输出。DAC 可以通过引脚输入参考电压 Vref+（通 ADC 共用）以获得更精确的转换结果。

STM32F4 的 DAC 模块主要特点有：

1. 2 个 DAC 转换器：每个转换器对应 1 个输出通道，其输出通道 DAC_OUTx（x=1 或 2）分别对应引脚 PA4 和 PA5。
2. 8 位或者 12 位单调输出
3. 12 位模式下数据左对齐或者右对齐
4. 同步更新功能
5. 噪声波形生成
6. 三角波形生成
7. 双 DAC 通道同时或者分别转换
8. 每个通道都有 DMA 功能

DAC 输出是受 DORx 寄存器直接控制的，但是我们不能直接往 DORx 寄存器写入数据，而是通过 DHRx 间接的传给 DORx 寄存器，实现对 DAC 输出的控制。前面我们提到，STM32F4 的 DAC 支持 8/12 位模式，8 位模式的时候是固定的右对齐的，而 12 位模式又可以设置左对齐/右对齐。单 DAC 通道 x，总共有 3 种情况：

1. 8 位数据右对齐：用户将数据写入 DAC_DHR8Rx[7:0]位（实际存入 DHRx[11:4]位 ）。
2. 12 位数据左对齐：用户将数据写入 DAC_DHR12Lx[15:4]位（实际存入 DHRx[11:0]位 ）。
3. 12 位数据右对齐：用户将数据写入 DAC_DHR12Rx[11:0]位（实际存入 DHRx[11:0]位 ）。

我们本章使用的就是单 DAC 通道 1，采用 12 位右对齐格式，所以采用第③种情况，且引脚选用 PA4。当 DAC 的参考电压为 Vref+的时候，DAC 的输出电压是线性的从 0~Vref+，12 位模式下 DAC 输出电压与 Vref+以及 DORx 的计算公式如下：

DACx 输出电压=Vref*（DORx/4095）

因此寄存器输入值 DORx=输出电压*4095/Vref，本章 Vref=3.3V。

10.3 实验内容

在 STM32CubeMX 中配置 DAC 的输出通道，在 Keil 中通过代码控制输出的电压值。

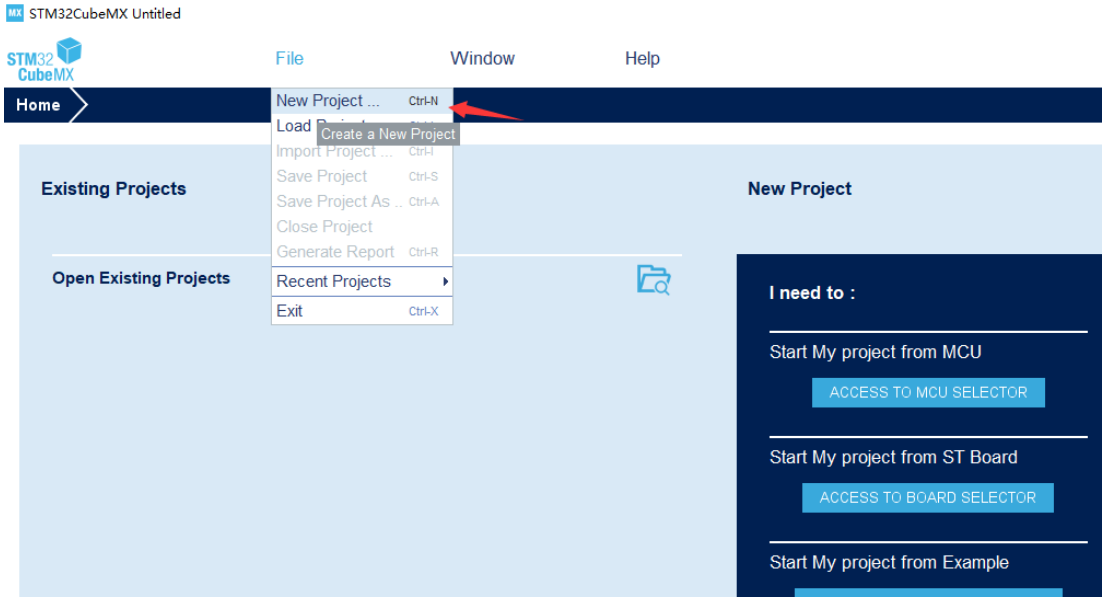
10.4 实验要求

控制 Nucleo-144 上 DAC 模块从 PA4 引脚输出固定的电压并测量验证。

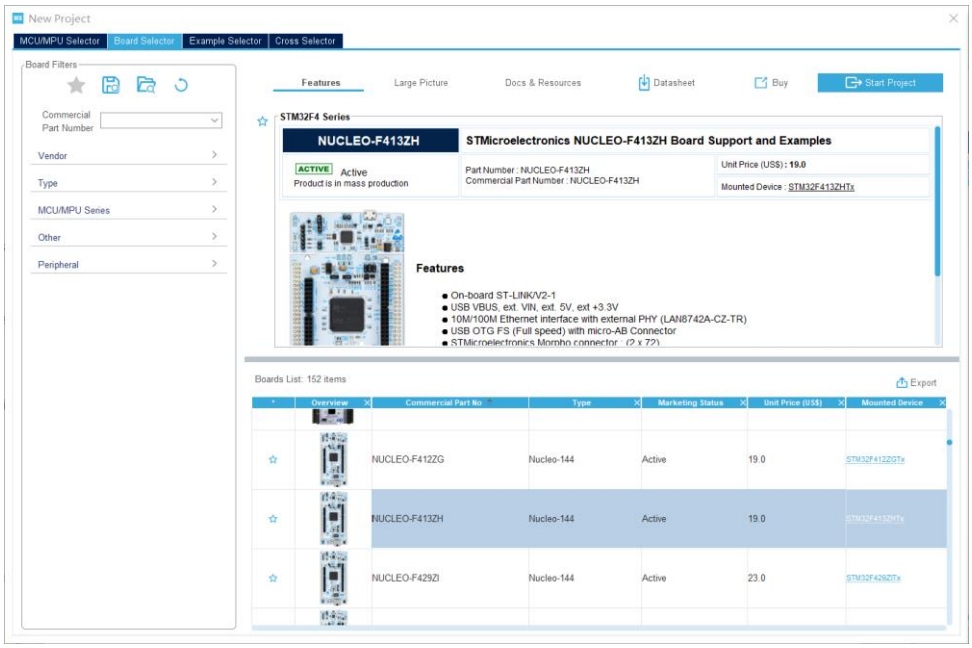
10.5 实验步骤

1. 利用 STM32CubeMX 生成模板代码

第一步，首先打开 STM32CubeMX 工具，点击如图所示的按钮新建工程。

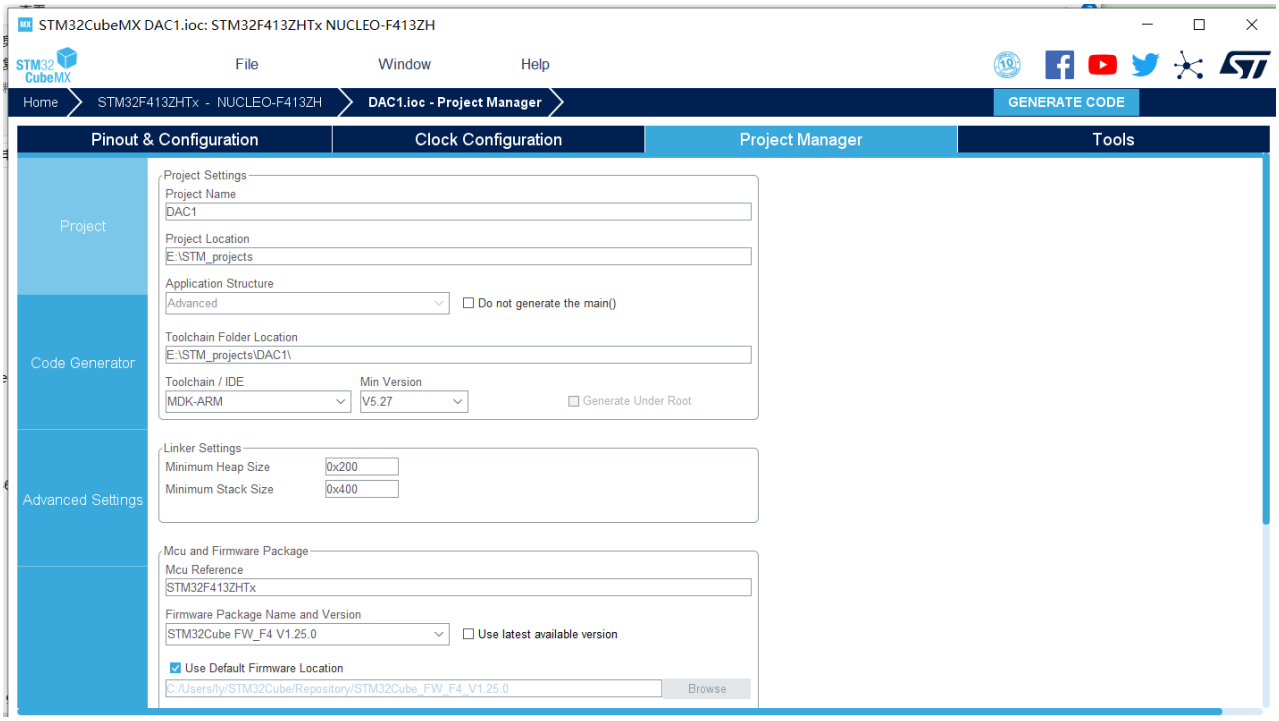


第二步，直接选择对应的 NUCLEO-144 开发板，省去对于时钟、中断等等的配置。

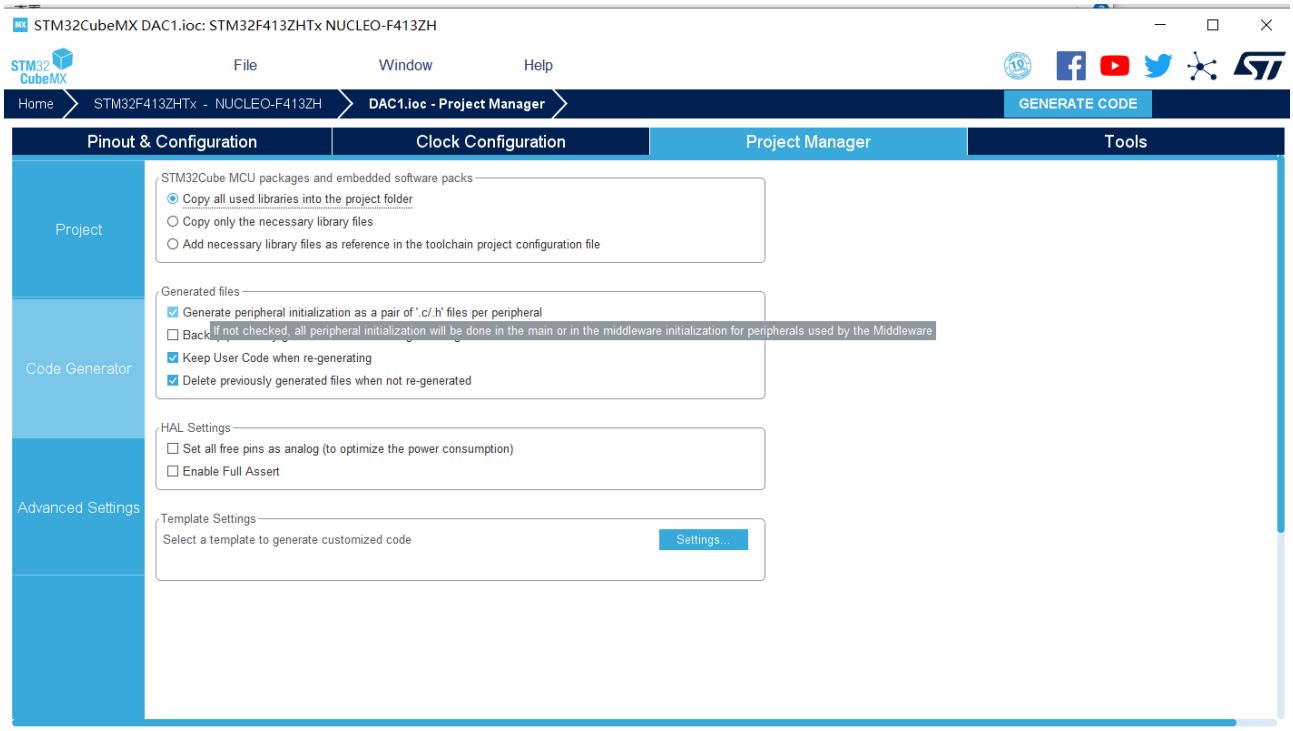


第三步，创建工程后填写一下的工程信息并对 GPIO 进行配置。

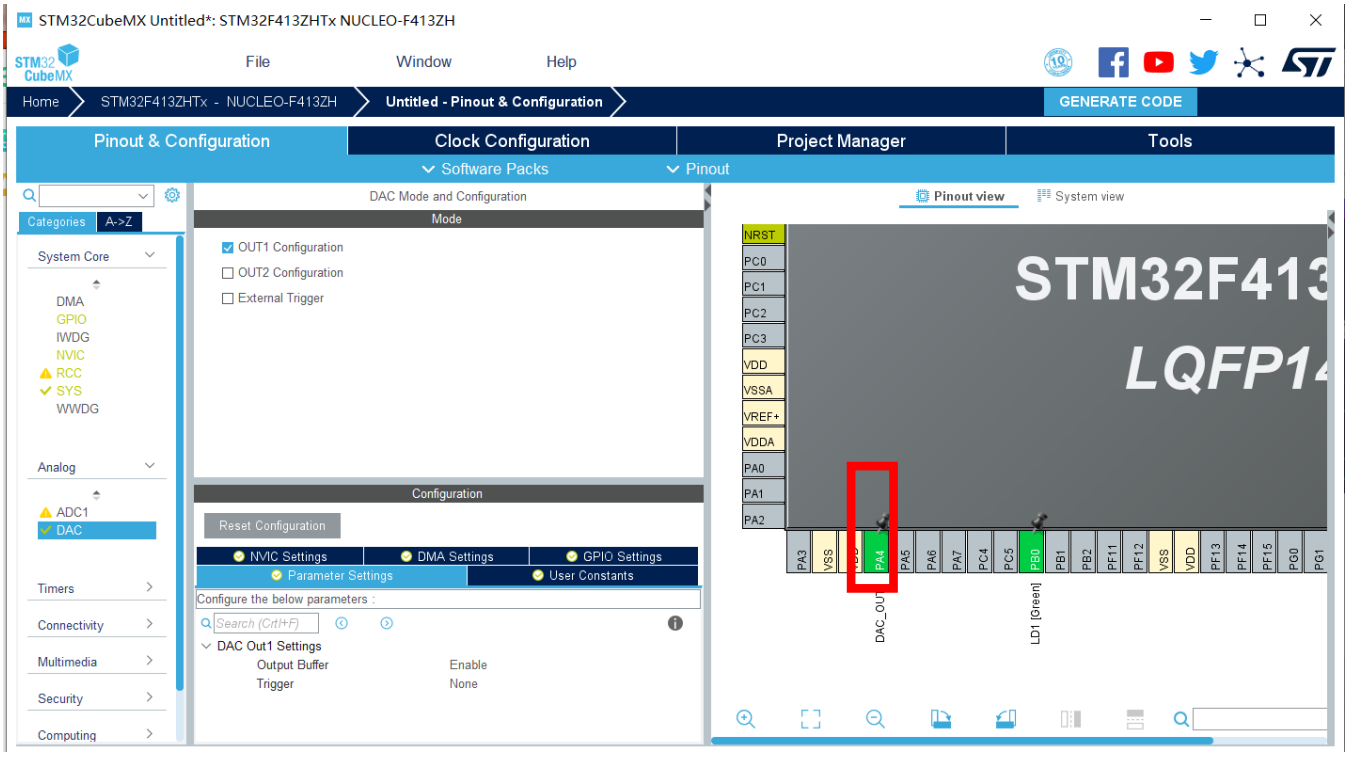
- Project Name： 工程名任意即可，这里填写 DAC1。
- Project Location： 工程路径，建议新建空文件夹专门存放所有的工程文件，注意路径中不要出现中文字符。
- Toolchain/IDE： 这里选择我们已经安装好的 Keil5，工具链（Toolchain）选择 MDK-ARM，版本号选择 V5。
- 取消勾选 Use latest available version，选择 V1.25.0。



- Code Generator：勾选第一项。



- 点击 Pinout&Configuration，在芯片引脚图中将 PA4 设置为 DAC_OUT1，在 Analog->DAC 中可以看到其他配置，这里保持默认。



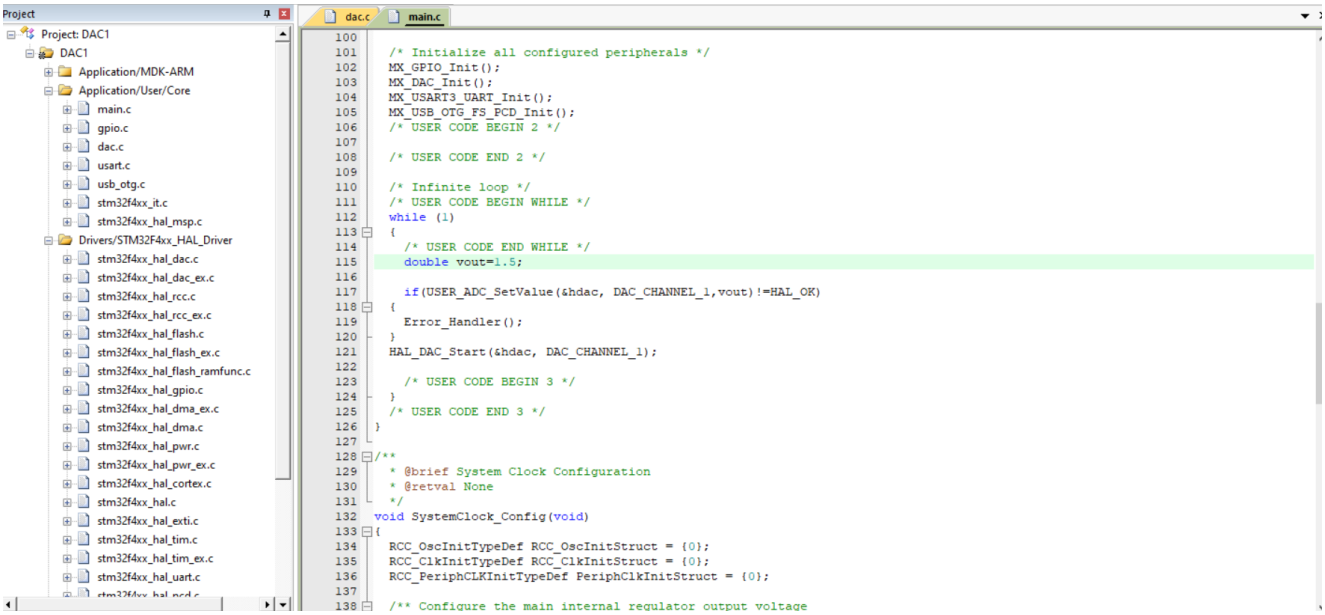
- 其余保持默认，点击右上角的 GENERATE CODE 生成 Keil 工程。

2. 利用 Keil 添加用户代码

工程建好后可以点击 open project，或者利用 Keil 打开工程以后，打开 dac.c，可以看到 DAC 的相关初始化代码已完成。

```
52  void HAL_DAC_MspInit(DAC_HandleTypeDef* dacHandle)
53  {
54
55    GPIO_InitTypeDef GPIO_InitStruct = {0};
56    if(dacHandle->Instance==DAC)
57    {
58    /* USER CODE BEGIN DAC_MspInit 0 */
59
60    /* USER CODE END DAC_MspInit 0 */
61      /* DAC clock enable */
62      __HAL_RCC_DAC_CLK_ENABLE();
63
64      __HAL_RCC_GPIOA_CLK_ENABLE();
65      /**DAC GPIO Configuration
66      PA4     ------> DAC_OUT1
67      */
68      GPIO_InitStruct.Pin = GPIO_PIN_4;
69      GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
70      GPIO_InitStruct.Pull = GPIO_NOPULL;
71      HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
72
73    /* USER CODE BEGIN DAC_MspInit 1 */
74
75    /* USER CODE END DAC_MspInit 1 */
76    }
77  }
78
79  void HAL_DAC_MspDeInit(DAC_HandleTypeDef* dacHandle)
80  {
81
82    if(dacHandle->Instance==DAC)
83    {
84    /* USER CODE BEGIN DAC_MspDeInit 0 */
85
86    /* USER CODE END DAC_MspDeInit 0 */
87      /* Peripheral clock disable */
88      __HAL_RCC_DAC_CLK_DISABLE();
89
90      /**DAC GPIO Configuration
```

打开 main.c，这里是程序的入口。



我们在 USER CODE BEGIN 0（59 行）处，也就是 main 函数前添加如下代码，对设置的电压值进行合理性判断（是否超出最大输出电压 3.3V）：

```
HAL_StatusTypeDef USER_ADC_SetValue(DAC_HandleTypeDef* hdac, uint32_t Channel, double CValue)
{
	uint32_t Data=0;

	if(CValue>3.3)
		return HAL_ERROR;

	Data=CValue*4095/3.3;

	return HAL_DAC_SetValue(hdac, Channel, DAC_ALIGN_12B_R, Data);	//设置 DAC 输出电压的库函数
}
```

在 while 循环前添加如下代码，设置输出电压：

```
double vout=1.5;
```

在 while 循环中添加如下代码：

```
		if(USER_ADC_SetValue(&hdac, DAC_CHANNEL_1,vout)!=HAL_OK)
		{
			Error_Handler();
		}
		HAL_DAC_Start(&hdac, DAC_CHANNEL_1);	//如果电压值合理，则使能 DAC 通道
```

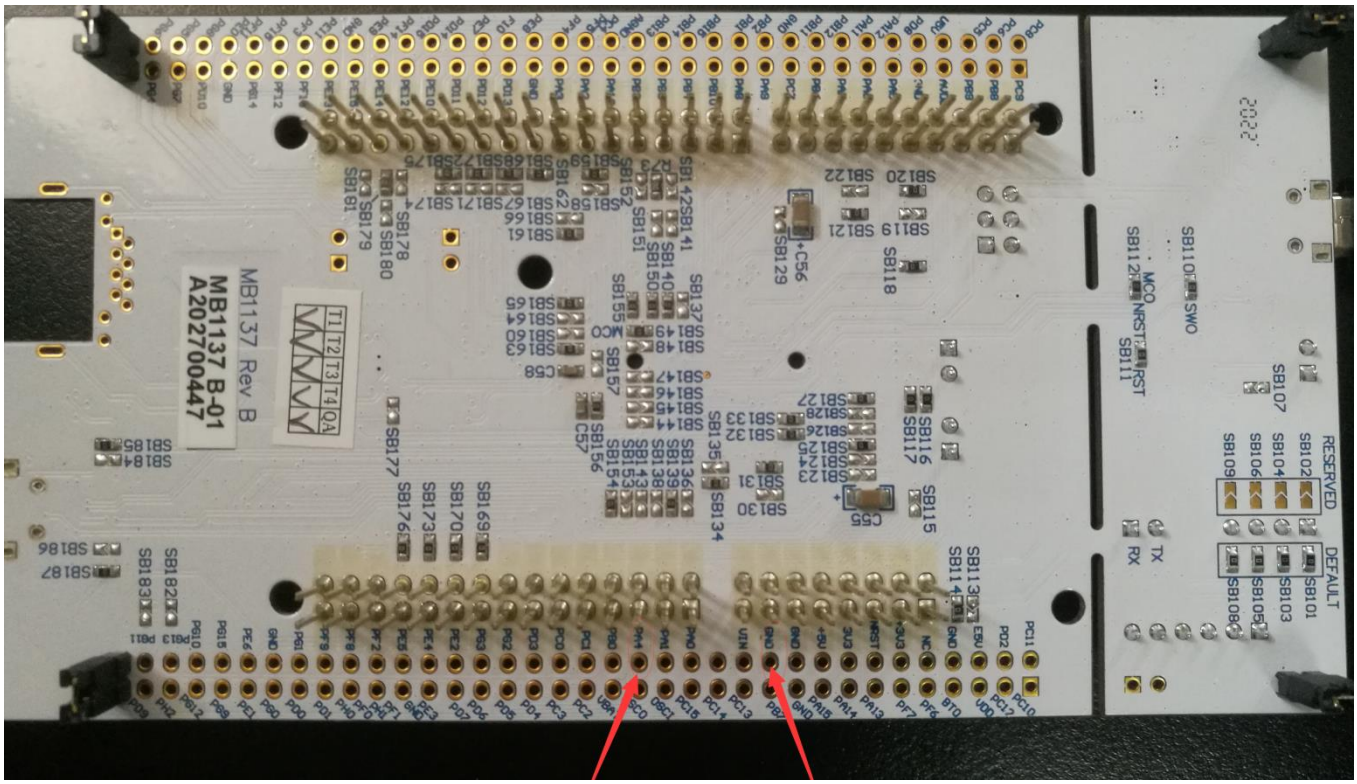
修改后的代码如下图所示：

```c
/* USER CODE BEGIN 0 */
HAL_StatusTypeDef USER_ADC_SetValue(DAC_HandleTypeDef* hdac, uint32_t Channel, double CValue)
{
  uint32_t Data=0;

  if(CValue>3.3)
    return HAL_ERROR;

//  CValue=CValue/3.3;
  Data=CValue*4095/3.3;

  return HAL_DAC_SetValue(hdac, Channel, DAC_ALIGN_12B_R, Data);
}

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_DAC_Init();
  MX_USART3_UART_Init();
  MX_USB_OTG_FS_PCD_Init();
  /* USER CODE BEGIN 2 */
  double vout=1.5;

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    if(USER_ADC_SetValue(&hdac, DAC_CHANNEL_1,vout)!=HAL_OK)
  {
    Error_Handler();
  }
  HAL_DAC_Start(&hdac, DAC_CHANNEL_1);

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

进行编译和代码烧录。

10.6 实验结果

代码烧录完成之后，可以用万用表测得 PA4 和 GND 之间的电压值为 1.5V（将板子反过来，可以找到 PA4 和 GND 对应的焊盘，表笔与之接触即可）。



第十一章 ADC 实验

11.1 实验目的

1. 学习 ADC 模块的工作原理
2. 在第十章的基础上实现对 Nucleo-144 开发板 ADC 模块的控制
3. 学会使用 STM32CubeMX 工具配置 ADC

11.2 实验原理

STM32F4 系列有 3 个 ADC，这些 ADC 可以独立使用，也可以使用双重/三重模式（提高采样率）。STM32F4 的 ADC 是 12 位逐次逼近型的模拟数字转换器。它有 19 个通道，可测量 16 个外部源、2 个内部源和 Vbat 通道的信号。这些通道的 A/D 转换可以单次、连续、扫描或间断模式执行。ADC 的结果可以左对齐或右对齐方式存储在 16 位数据寄存器中。模拟看门狗特性允许应用程序检测输入电压是否超出用户定义的高/低阀值。

STM32F4 的 ADC 最大的转换速率为 2.4Mhz，也就是转换时间为 0.41us（在 ADCCLK=36M,采样周期为 3 个 ADC 时钟下得到），不要让 ADC 的时钟超过 36M，否则将导致结果准确度下降。

STM32F4 将 ADC 的转换分为 2 个通道组：规则通道组和注入通道组。规则通道相当于你正常运行的程序，而注入通道呢，就相当于中断。在你程序正常执行的时候，中断是可以打断你的执行的。同这个类似，注入通道的转换可以打断规则通道的转换，在注入通道被转换完成之后，规则通道才得以继续转换。

STM32F4 的 ADC 可以进行很多种不同的转换模式，我们本章仅介绍如何使用规则通道的单次转换模式。与 DAC 相对应的是，ADC 所读取的数据其实是寄存器里的相对电压值，在参考电压为 Vref+的时候，12 位模式下真实电压为：

真实电压=读取数值*（Vref/4095）

控制 STM32F4 的 ADC 需要对多个寄存器进行赋值配置，非常复杂，利用 STM32CubeMX 工具我们可以更直观更快速地配置 ADC。

11.3 实验内容

在 STM32CubeMX 中配置 DAC 的输出通道与 ADC 的输入通道，在 Keil 中通过代码控制输出的电压值、ADC 采样该电压值并在串口打印测量结果。

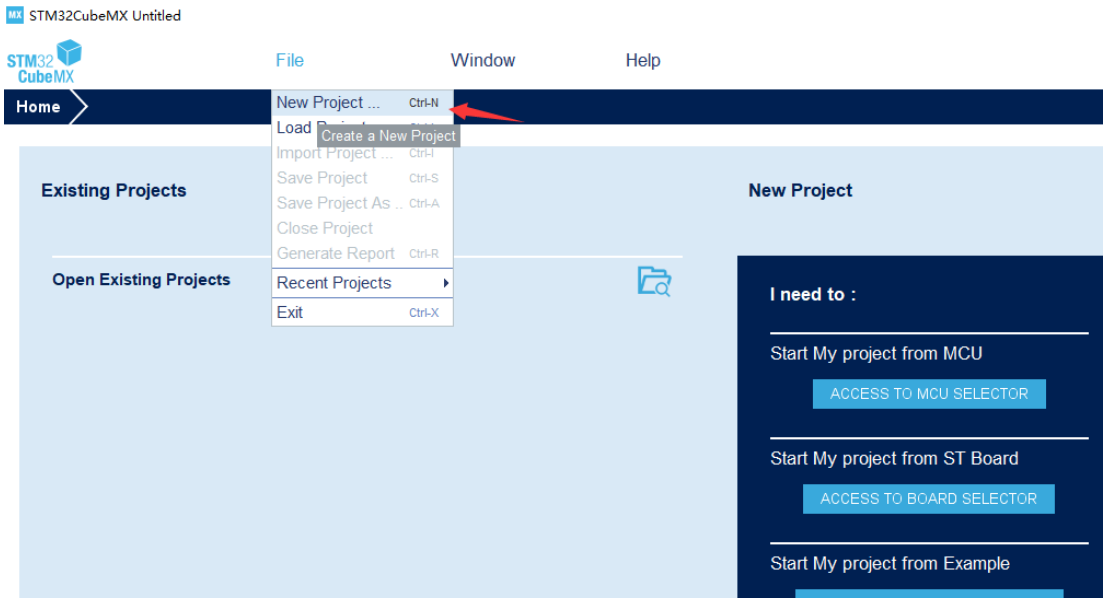
11.4 实验要求

控制 Nucleo-144 上 DAC 模块从 PA4 引脚输出固定的电压并复用 PA4 引脚为 ADC 输入通道，ADC 采样 DAC 的输出电压值并在串口打印出相应的测量结果。

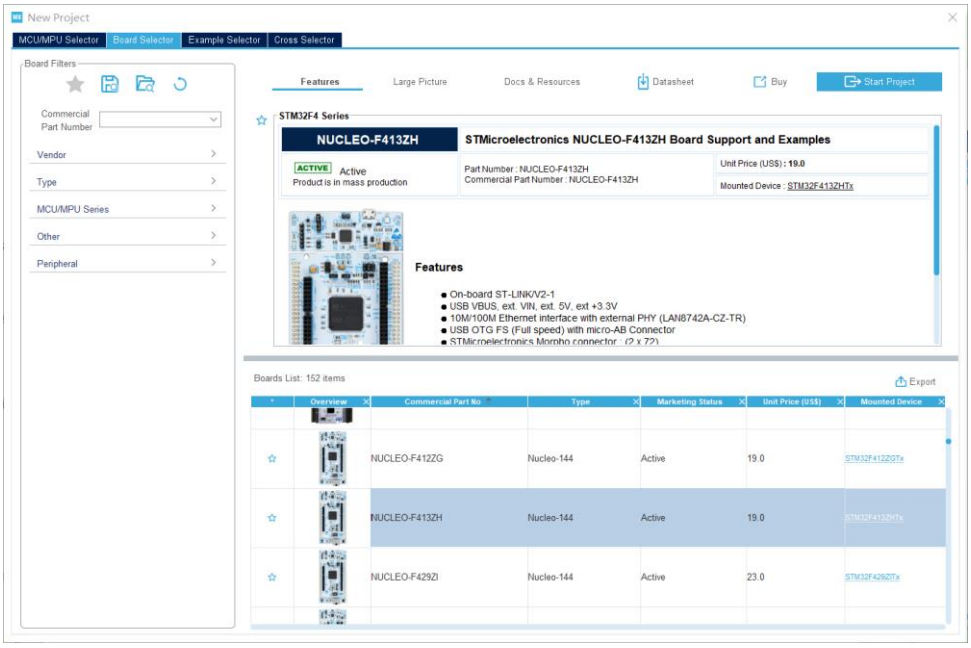
11.5 实验步骤

1. 利用 STM32CubeMX 生成模板代码

第一步，首先打开 STM32CubeMX 工具，点击如图所示的按钮新建工程。

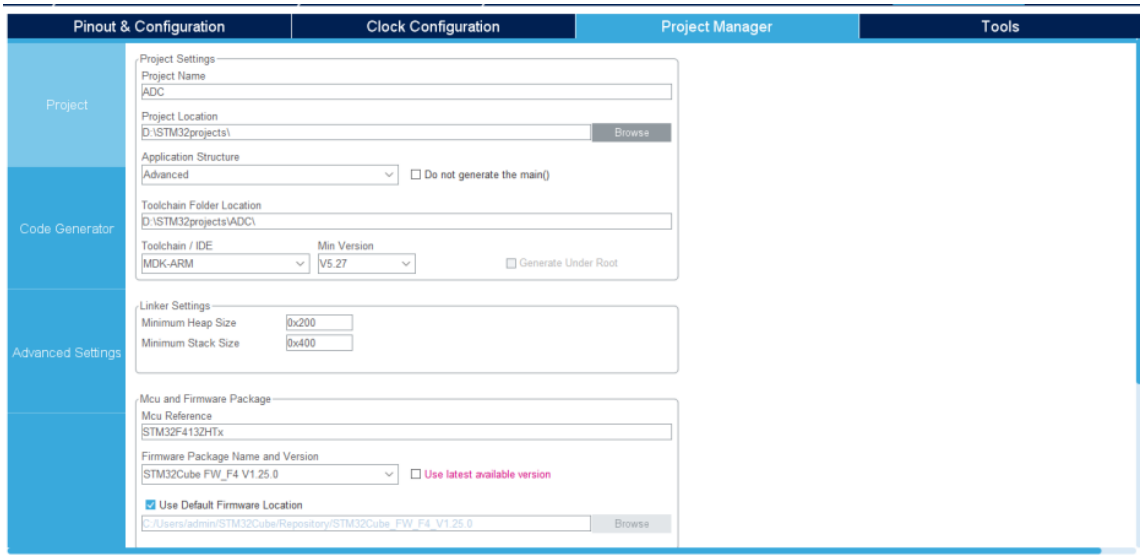


第二步，直接选择对应的 NUCLEO-144 开发板，省去对于时钟、中断等等的配置。

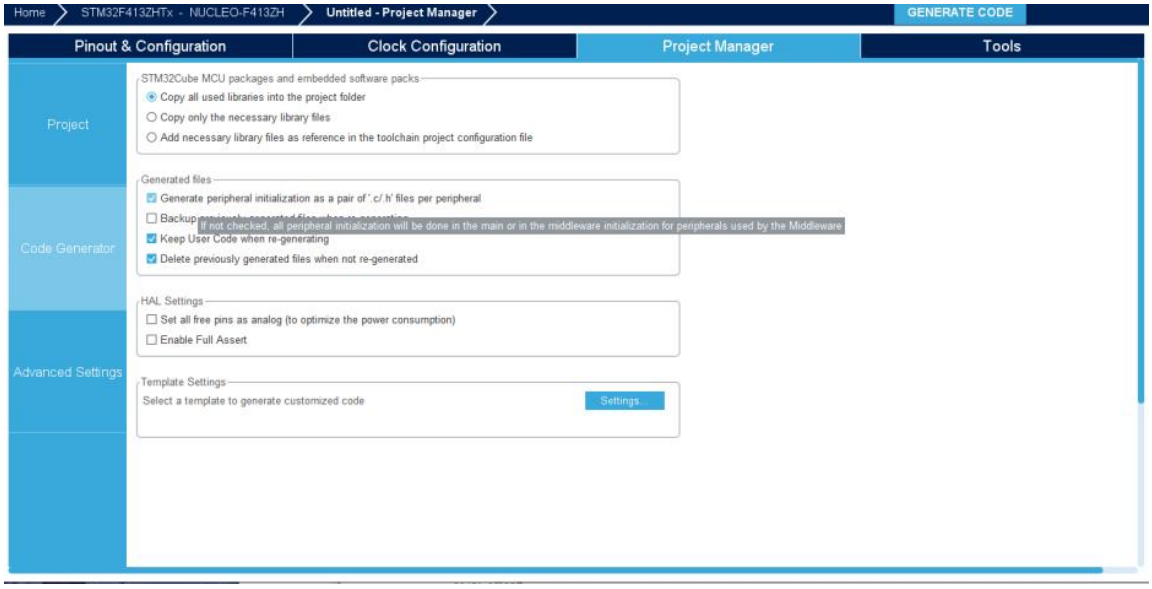


第三步，创建工程后填写一下的工程信息并对 GPIO 进行配置。

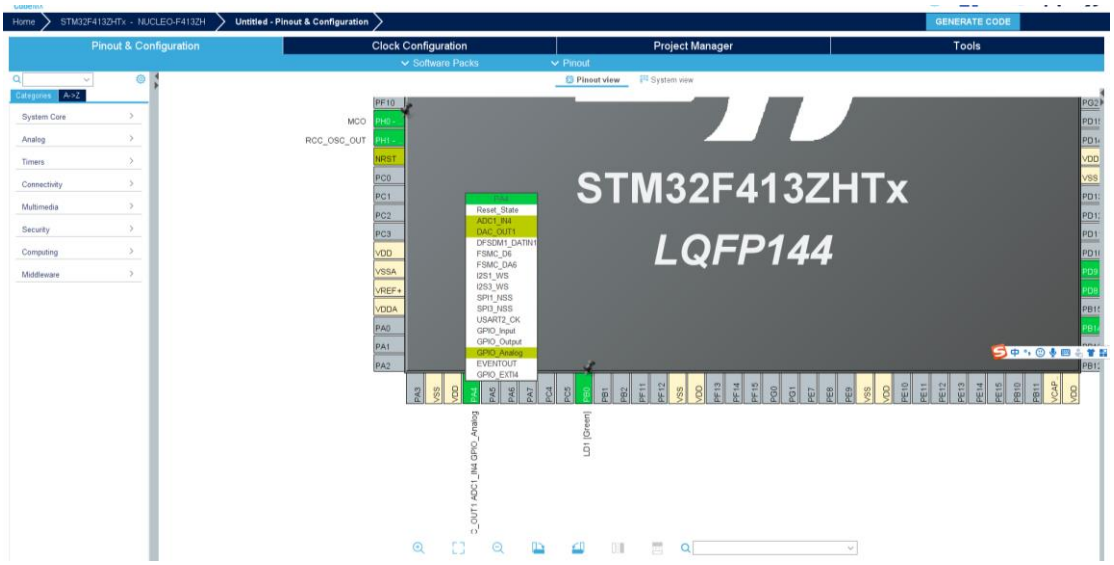
- Project Name： 工程名任意即可，这里填写 ADC。
- Project Location： 工程路径，建议新建空文件夹专门存放所有的工程文件，注意路径中不要出现中文字符。
- Toolchain/IDE： 这里选择我们已经安装好的 Keil5。
- 取消勾选 Use latest available version，选择 V1.25.0。



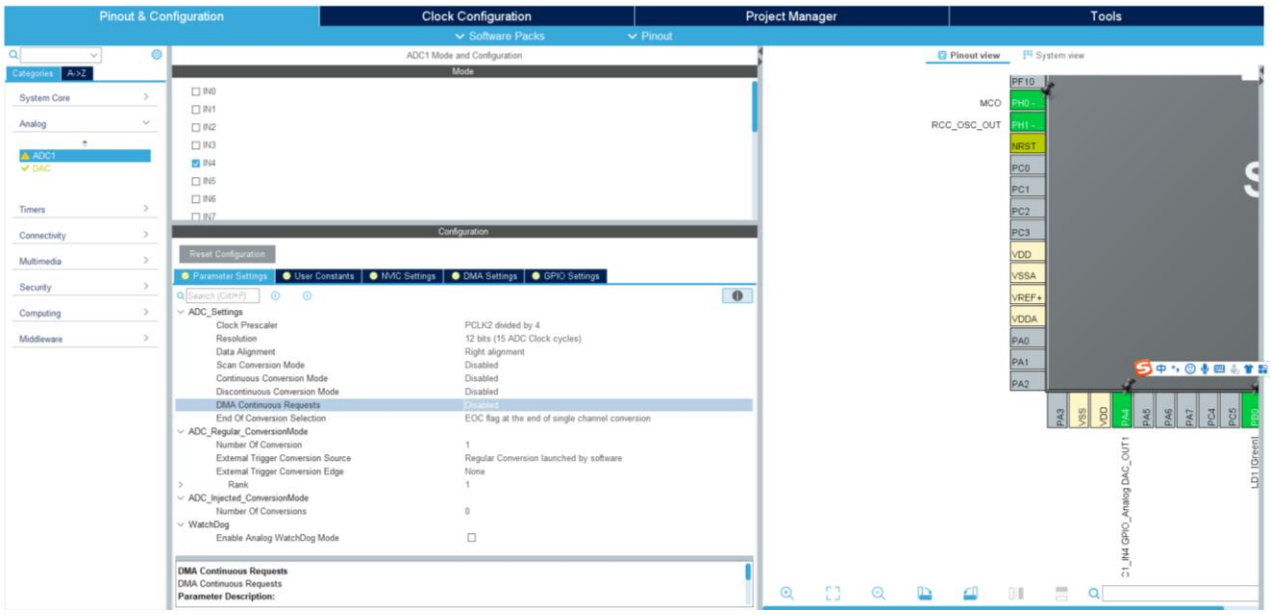
- Code Generator：勾选第一项。



- 点击 Pinout&Configuration，在芯片引脚图中将 PA4 设置为 DAC_OUT1，复用为 ADC1_IN4，并且设置为 GPIO_Analog（模拟复用）。设置为模拟复用的原因是：芯片上电默认 GPIO 是浮空输入的，在干扰和噪声环境下，内部的施密特触发器输出随机电平状态，造成功耗上升；而设置为模拟复用，施密特触发器的输出强制为 0，降低功耗和噪声。



- 在 Analog->ADC1 中可以看到 ADC1 其他配置，保持默认。



这里从上到下依次为：

ADC 设置>>>

-分频系数：可选 2/4/6/8 分频，本章选用 4 分频，系统时钟频率约为 96MHz，因此 ADC 的工作频率 24MHz；

-分辨率：可选 12/10/8/6 位，本章选用 12 位；

-对齐方式：左对齐还是右对齐，本章选用右对齐；

-扫描模式：DISABLE，不开启；

-开启连续转换模式：DISABLE；

-开启单次转换模式：DISABLE；

-开启 DMA 请求连续模式：DISABLE；

-转换结束标志选择：单个通道转换结束的 EOC 标志；

ADC 规则通道模式>>>

-规则序列中有多少个转换：1；

-外部触发方式：ADC_SOFTWARE_START，软件触发；

-外部触发边沿；

Rank>>>

-用来设置要配置的通道是规则序列中的第几个转换；

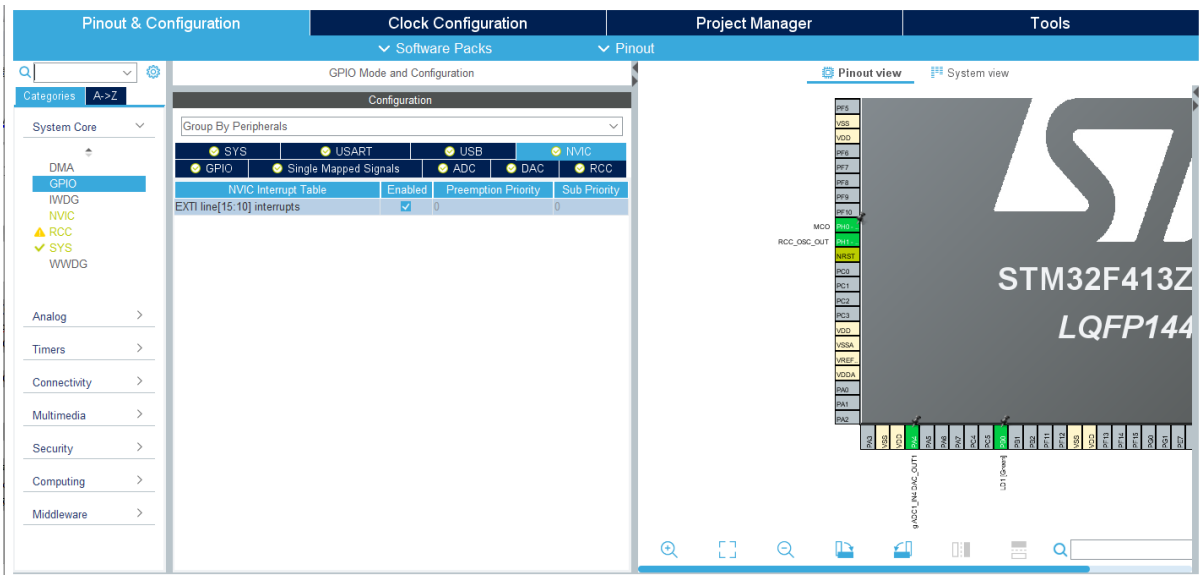
ADC 注入通道模式>>>

-转换通道的数量：0；

看门狗使能>>>

-开启后允许应用程序检测输入电压是否超出用户定义的高/低阀值。

- 在 System Core->GPIO->NVIC 中开启按键的中断

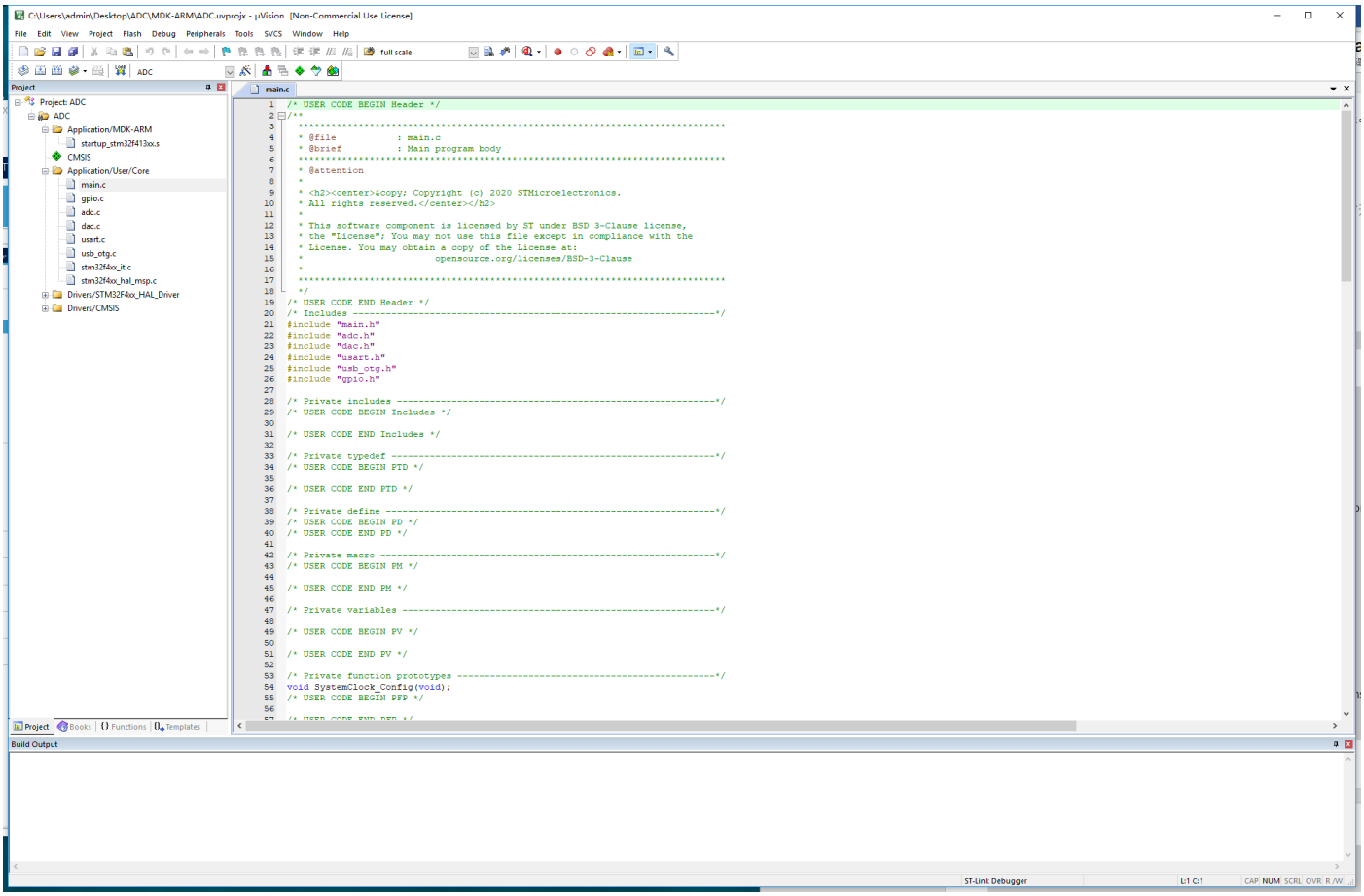


- 其余保持默认，点击右上角的 GENERATE CODE 生成 Keil 工程。

2. 利用 Keil 添加用户代码

工程建好后可以点击 open project，或者利用 Keil 打开工程以后，打开 dac.c 和 adc.c，可以看到 DAC 和 ADC 的相关初始化代码已完成。

打开 main.c，这里是程序的入口。



在 USER CODE BEGIN Includes 处，添加如下头文件

```
#include "stdio.h"
```

在 USER CODE BEGIN 0 处，也就是 main 函数前添加如下代码：

```
float temp;             //最终测量的电压值
uint16_t adcx;          //ADC 直接读取到的数值
extern uint16_t adcx;     //声明为全局变量

//对 printf 函数的重定向
#ifdef __GNUC__           /* With GCC, small printf (option LD Linker->Libraries->Small printf
    set to 'Yes') calls __io_putchar() */
#define PUTCHAR_PROTOTYPE int __io_putchar(int ch)
#else
#define PUTCHAR_PROTOTYPE int fputc(int ch, FILE *f)
#endif /* __GNUC__ */

PUTCHAR_PROTOTYPE
{
  /* Place your implementation of fputc here */
  /* e.g. write a character to the USART3 and Loop until the end of transmission */
  HAL_UART_Transmit(&huart3, (uint8_t *)&ch, 1, 0xFFFF);

  return ch;
};

//DAC 输出电压范围预判及电压输出函数
HAL_StatusTypeDef USER_ADC_SetValue(DAC_HandleTypeDef* hdac, uint32_t Channel, double CValue)
{
      uint32_t Data=0;

      if(CValue>3.3)
            return HAL_ERROR;

      Data=CValue*4095/3.3;

      return HAL_DAC_SetValue(hdac, Channel, DAC_ALIGN_12B_R, Data);   //设置 DAC 输出电压的库函数
}
```

在 USER CODE BEGIN 2 处添加如下代码，设置输出电压：

```
      double vout=1.630;

            if(USER_ADC_SetValue(&hdac, DAC_CHANNEL_1,vout)!=HAL_OK)
            {
                  Error_Handler();
            }
            HAL_DAC_Start(&hdac, DAC_CHANNEL_1);      //如果电压值合理，则使能 DAC 通道
```

在 while 循环中 USER CODE BEGIN 3 处添加如下代码：

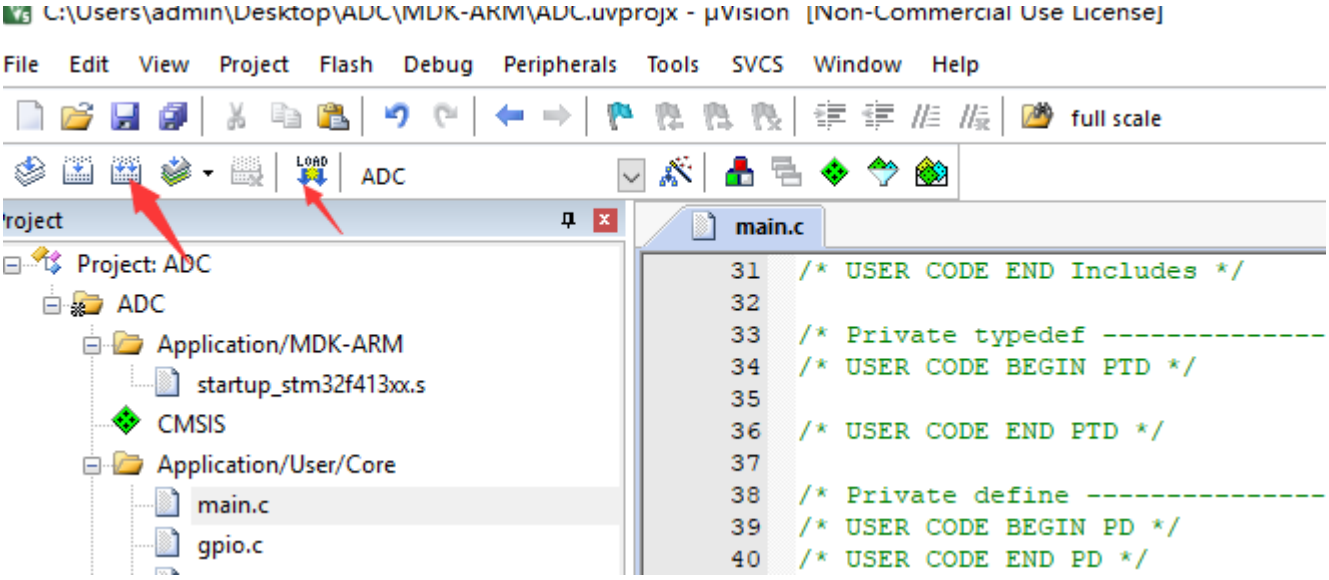
```
            adcx=HAL_DAC_GetValue(&hdac,DAC_CHANNEL_1);   //适用于 ADC 直接读取 DAC 数
```

```
值
//          HAL_ADC_Start(&hadc1);   //使能 ADC 通道
//          HAL_ADC_PollForConversion(&hadc1,10);    //采用查询方式读取，需等待上一次转换结束，此处等待 10ms
//          adcx=HAL_ADC_GetValue(&hadc1);
//这三句注释掉的语句适用于 ADC 读取外部电压，不限于 DAC，适用范围更广，但在本章中使用的话精确度较低
            temp=(float)adcx*(3.3/4095);   //转化成真实电压值
            HAL_Delay(20);
```

在 USER CODE BEGIN 4 处添加中断回调函数，使得按键按下一次才在串口打印测量结果，注意中断回调函数内不能使用延时函数，并且尽量进行简单的操作：

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
        if (GPIO_Pin == GPIO_PIN_13)
  {
            printf("%d",adcx);   //整型输出直接读取的数值
            printf("\r\n");
            printf("%f",temp);    //以小数形式输出真实电压值，注意浮点型不能限制精度，默认保留小数点后六位
            printf("V\r\n");
  }
}
```

进行编译和代码烧录。



12.6 实验结果

打开 XCOM V2.0，选择相应的串口和波特率，打开串口，按下蓝色的用户按键一次，串口打印出一次测量结果，与我们预先设置 DAC 输出的电压值基本一致：


XCOM V2.0
2022
1.629451V
2022
1.629451V
2022
1.629451V
串口选择
COM3:USB-SERIAL
波特率
115200
停止位
1
数据位
8
奇偶校验
无
串口操作
关闭串口
保存窗口
清除接收
单条发送
多条发送
协议传输
帮助
开源电子网:www.openedv.com
发送
清除发送
定时发送
周期:
1000
ms
打开文件
发送文件
停止发送
16进制发送
发送新行
0%
开源电子网：www.openedv.com
www.openedv.com
S:0
R:51
CTS=0 DSR=0 DCD=0
当前时间 15:36:46

第十二章 STTS751 温度获取实验

12.1 实验目的

1. 学习 IKS01A3 拓展板的使用
2. 学会使用 STM32CubeMX 工具配置 IKS01A3 所需端口

12.2 实验内容

在 STM32CubeMX 中配置 STTS751 所需端口，通过串口输出传感器值

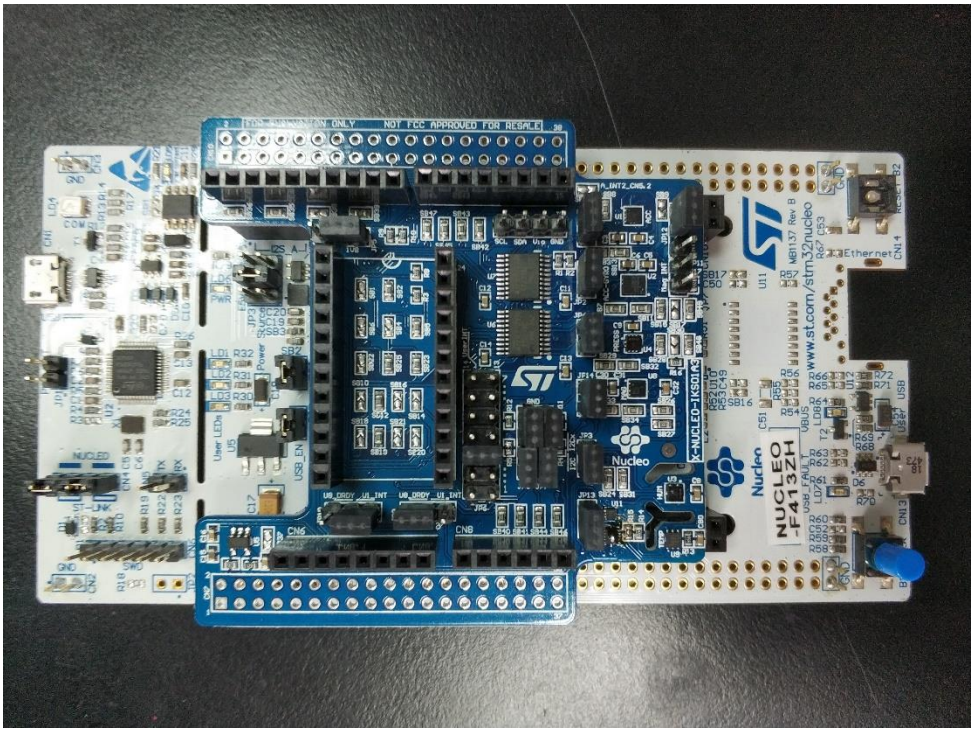
12.3 实验要求

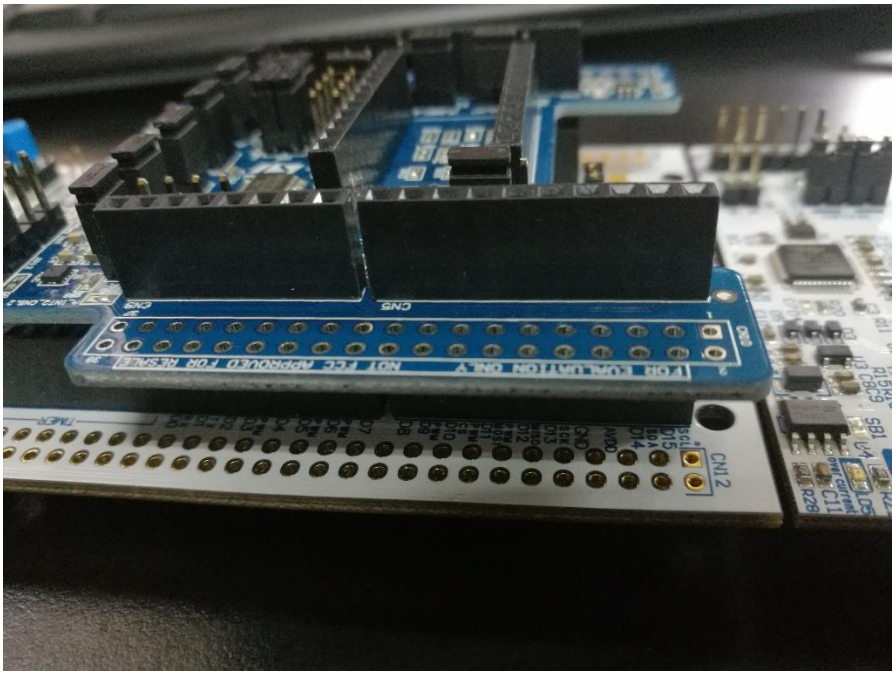
能够在串口上实时获取当前室温

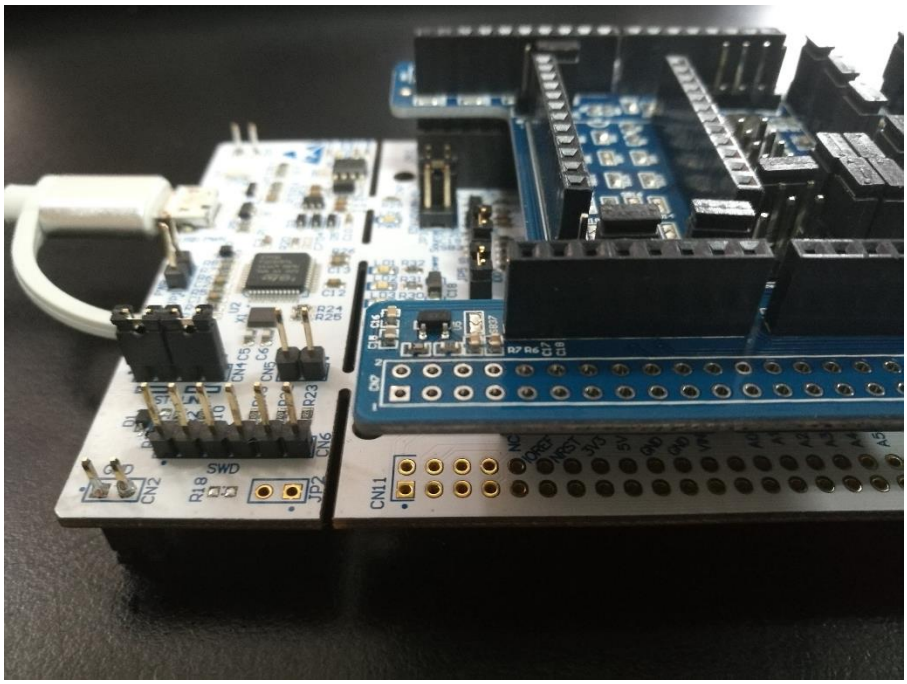
12.4 实验步骤

1. 将 KS01A3 拓展板安装到 NUCLEO-144 开发板上

仔细对照正视图和侧视图，注意扩展版凹口向上，排针与母座的缺口需要对应，两边第一个排针都与第一个外侧母座相对应。

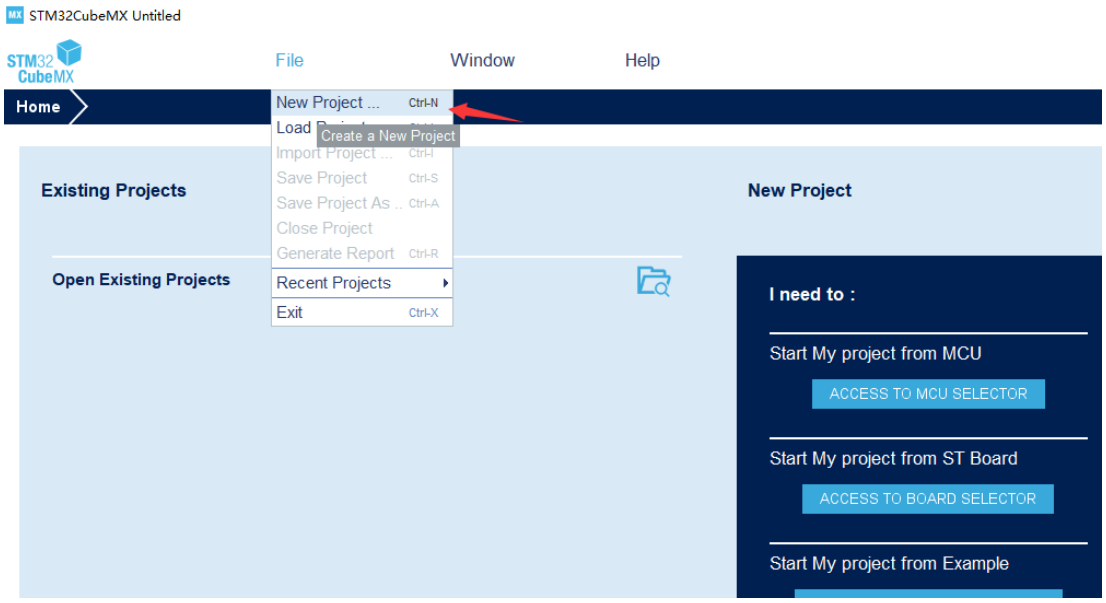




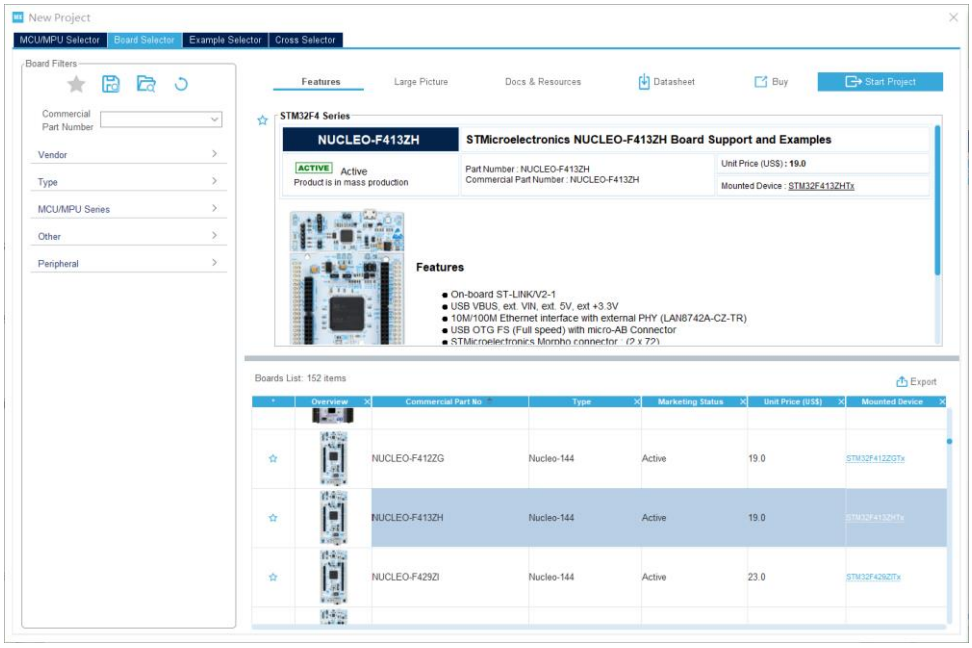


2. 利用 STM32CubeMX 生成模板代码

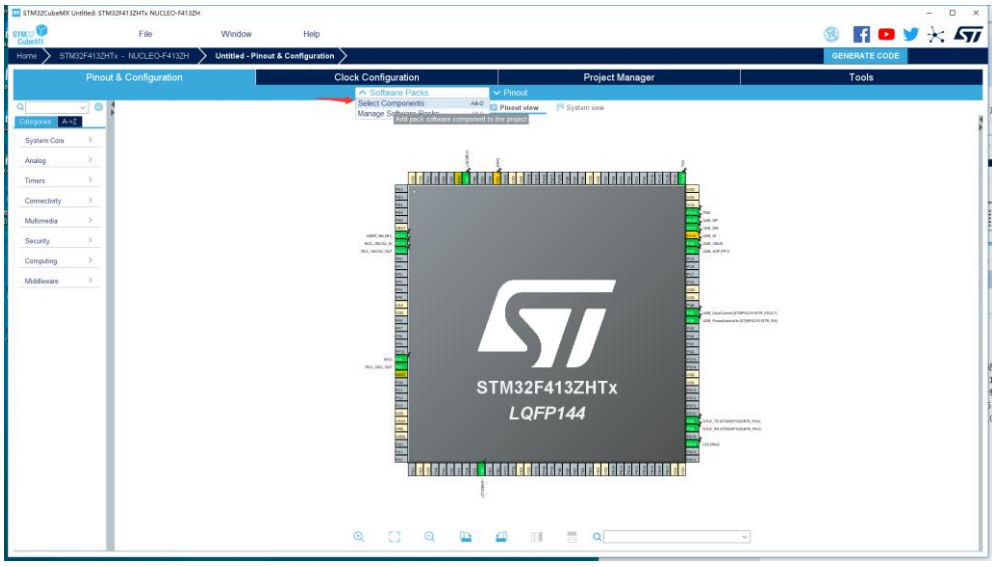
第一步，首先打开 STM32CubeMX 工具，点击如图所示的按钮新建工程。



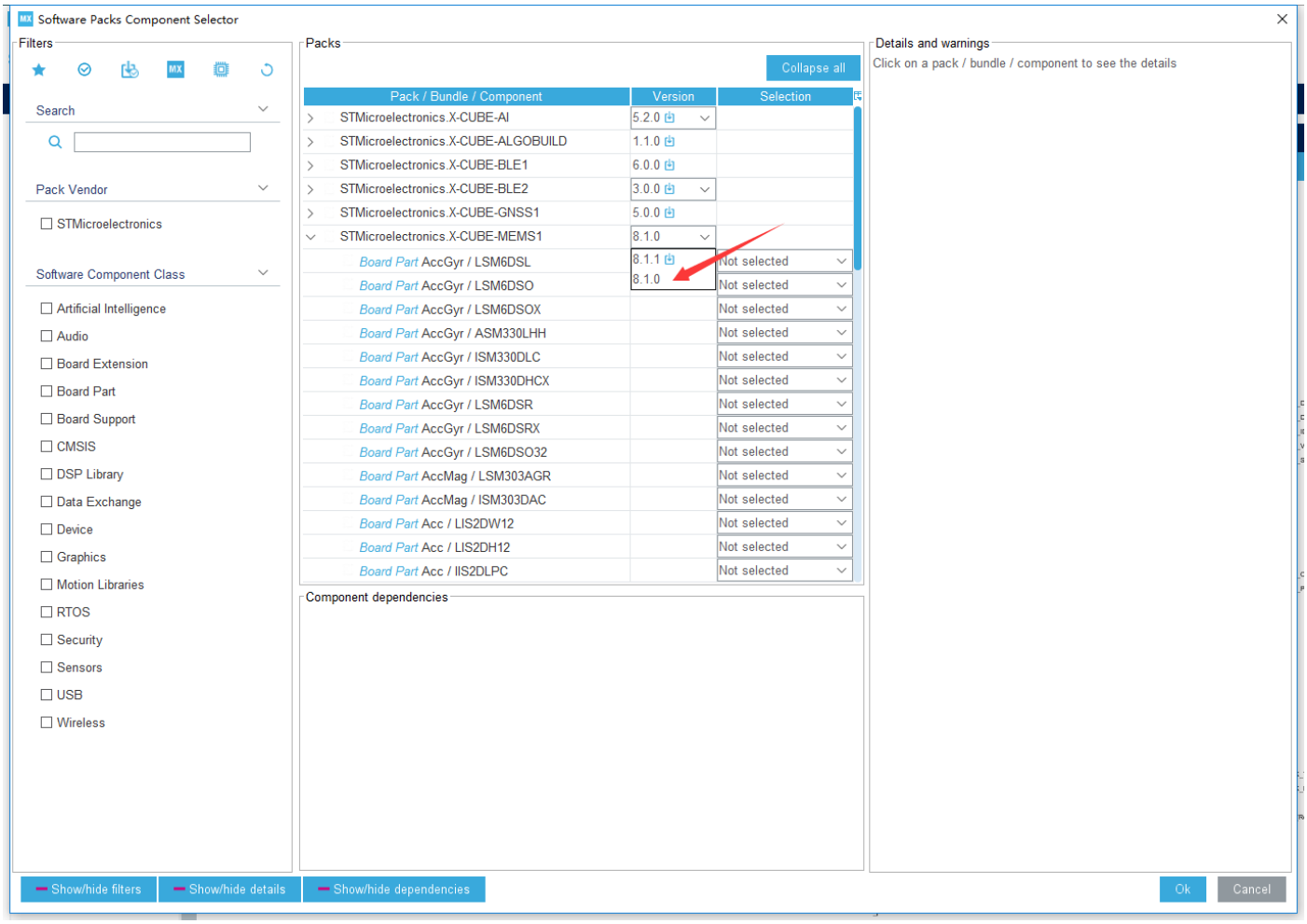
第二步，直接选择对应的 NUCLEO-144 开发板，省去对于时钟、中断等等的配置。



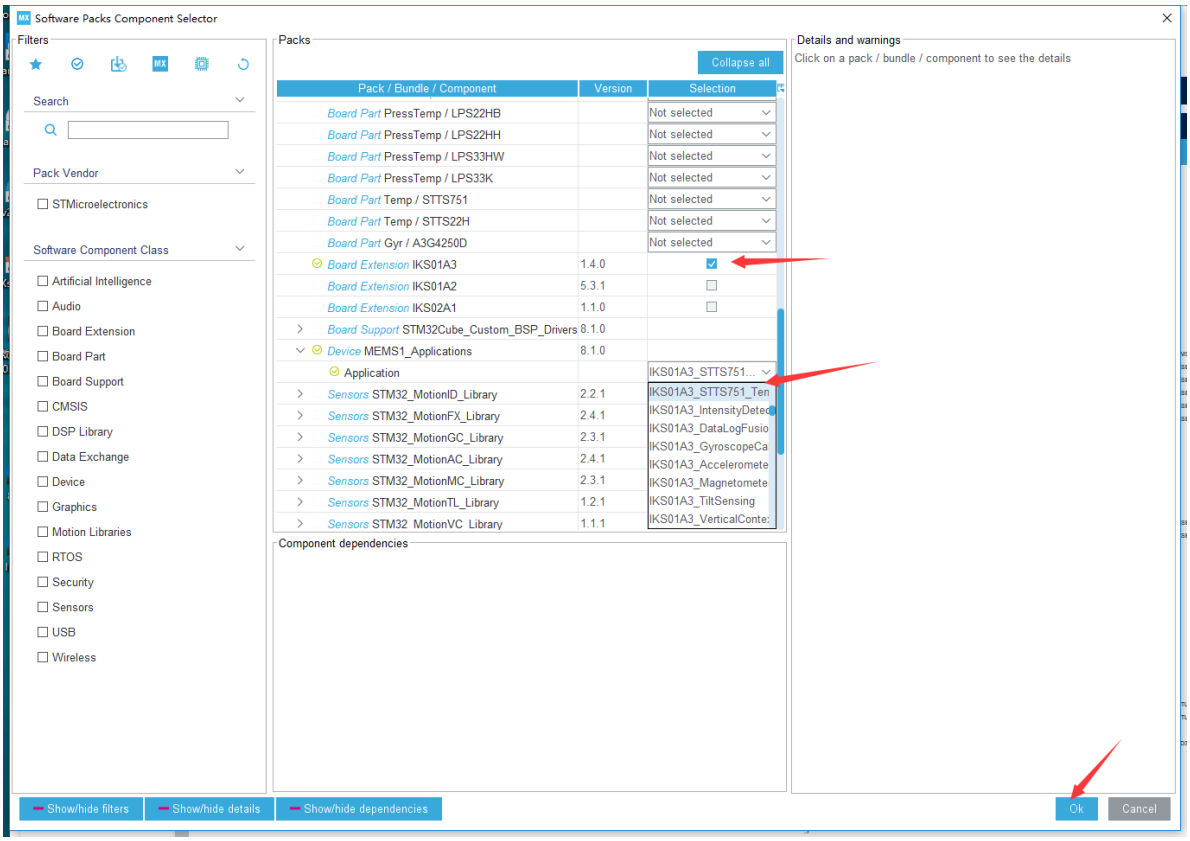
在 Pinout&Configure 界面点击 Select Components



选择 STMmicroelectronics X-CUBE-MEMS1，并将版本切换到 8.1.0



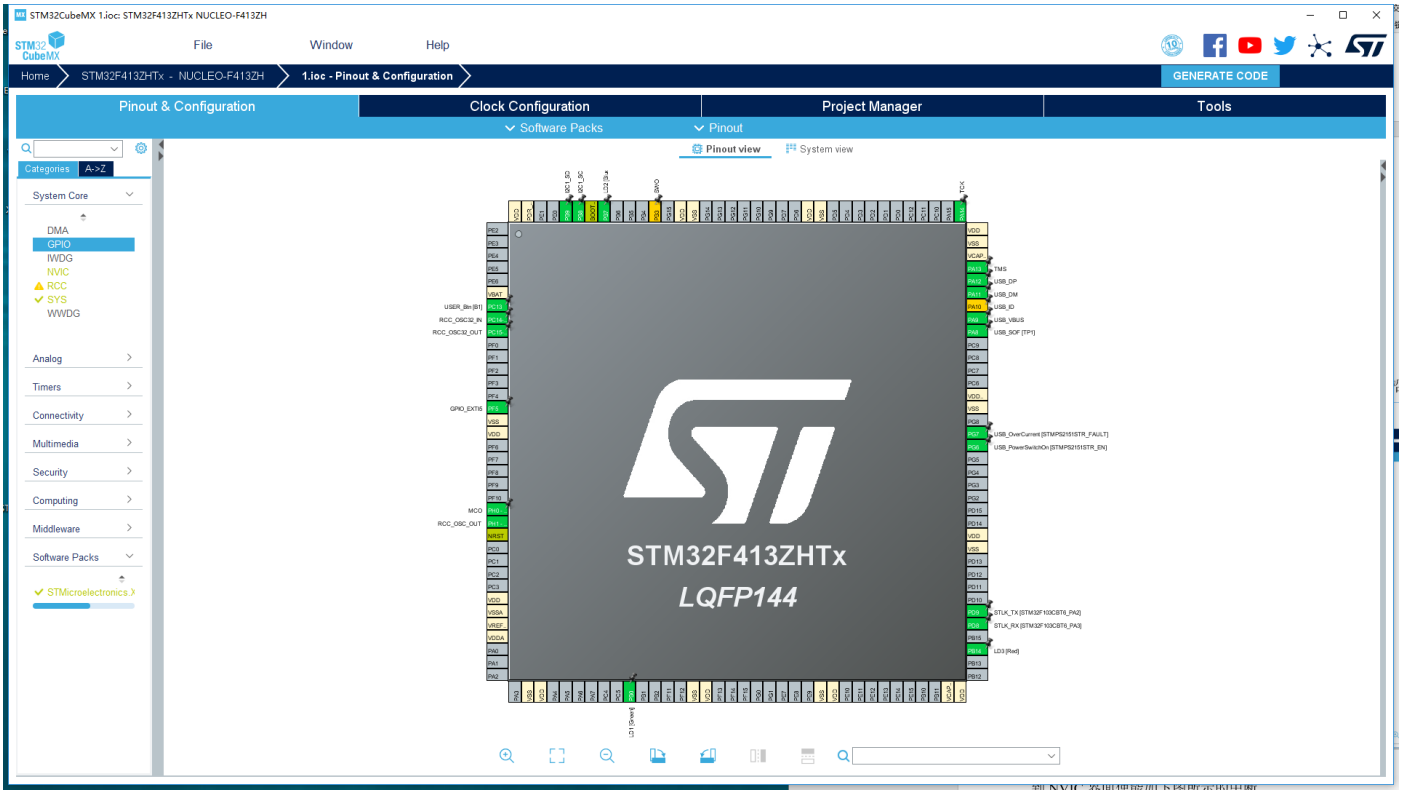
选中 IKS01A3，并在应用中选择 STTS751 温度读取，点击 OK



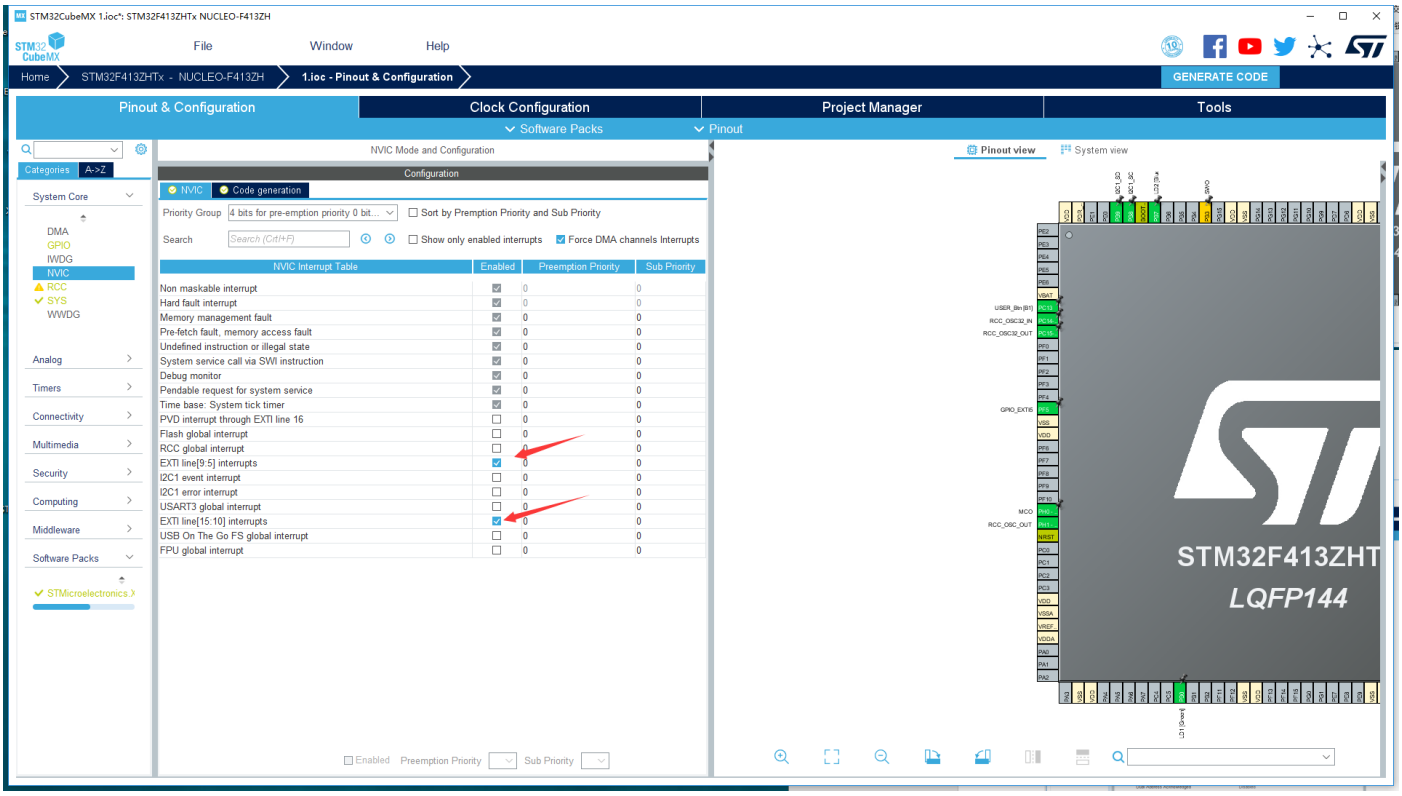
回到引脚配置界面，对以下引脚进行配置

- PB8：　I2C1_SCL
- PB9：　I2C1_SDA
- PF5：　GPIO_EXITI5

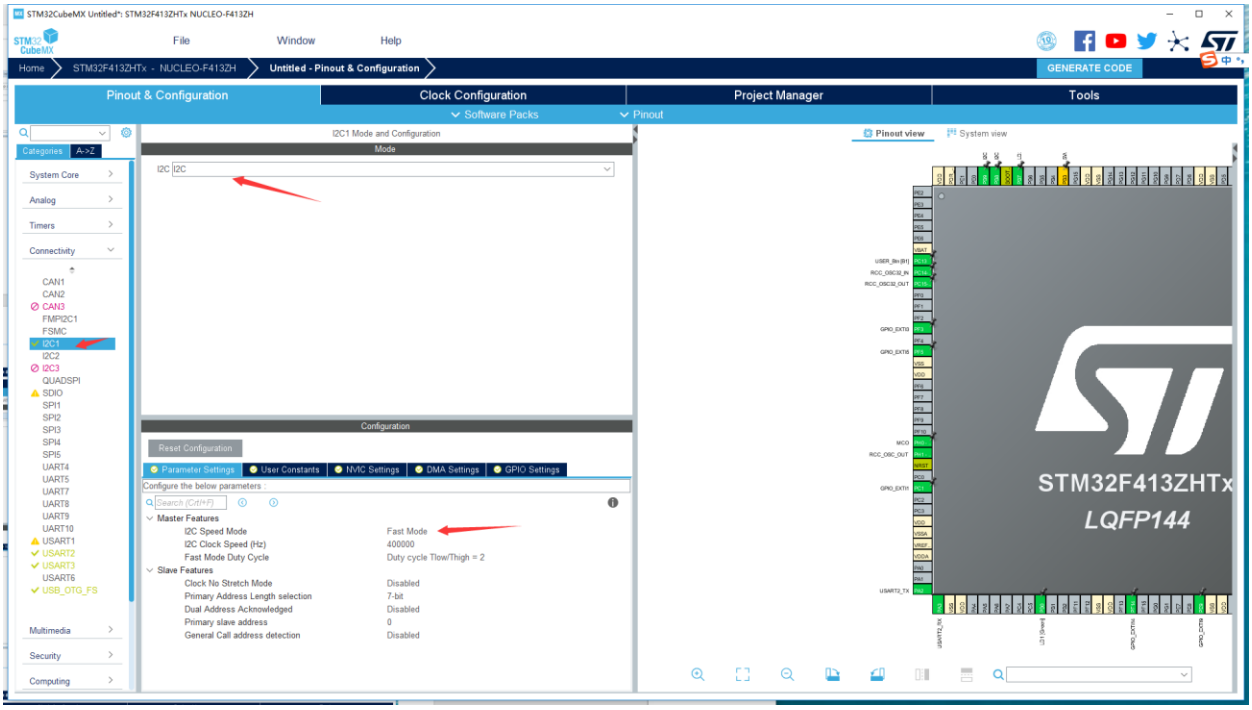
引脚配置完成后如下图所示



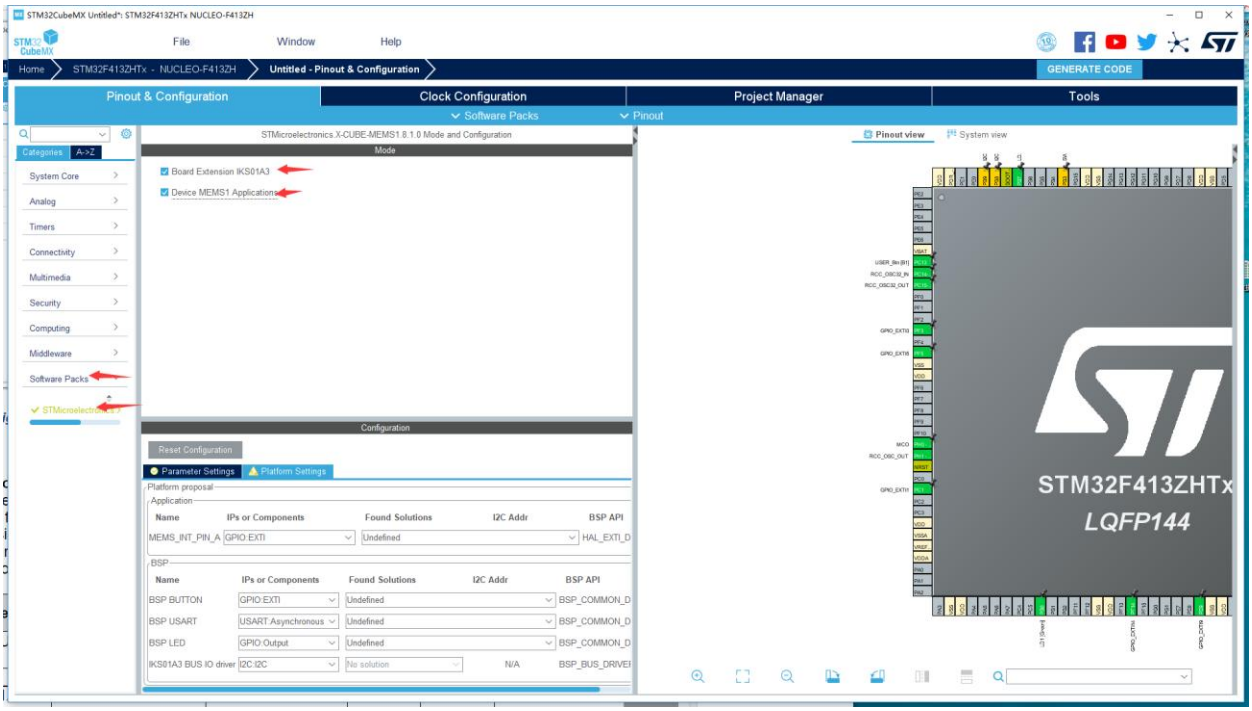
到 NVIC 界面使能如下图所示的中断



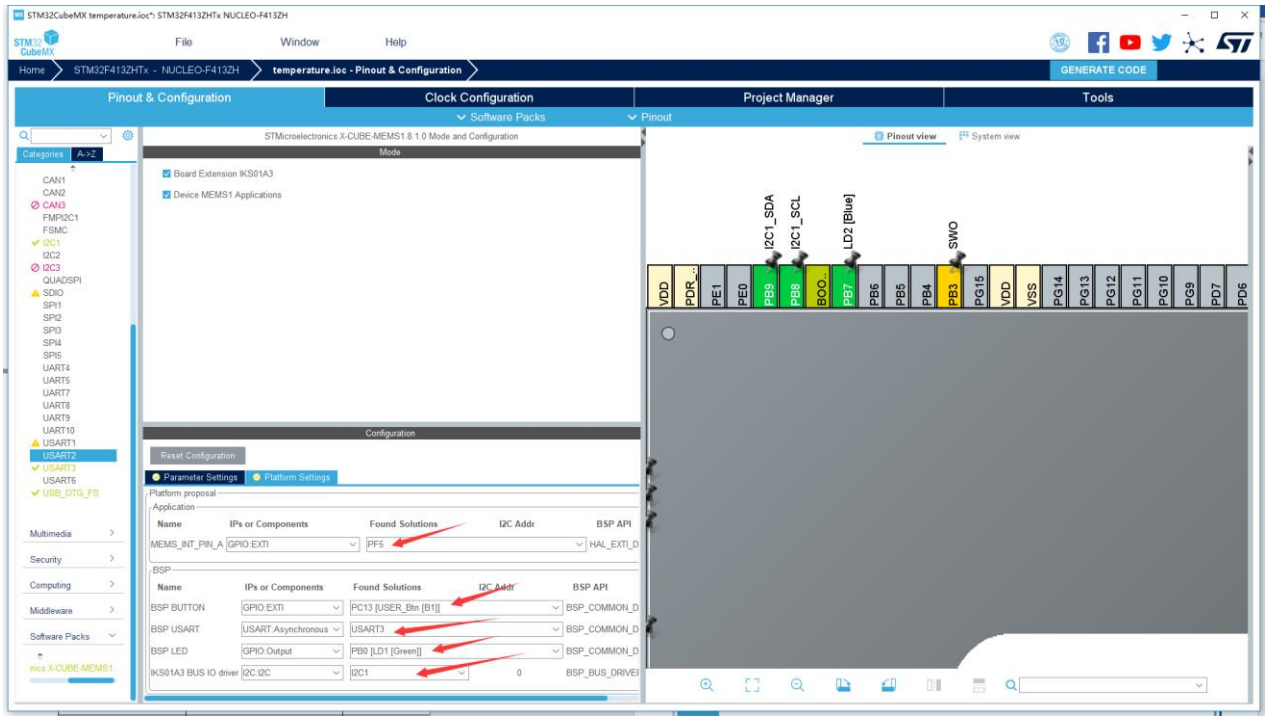
开启 I2C1，并将模式改成 Fast Mode，其余保持默认



在 SoftWare Packs 对拓展板进行配置，这里我们使能了 IKS01A3。

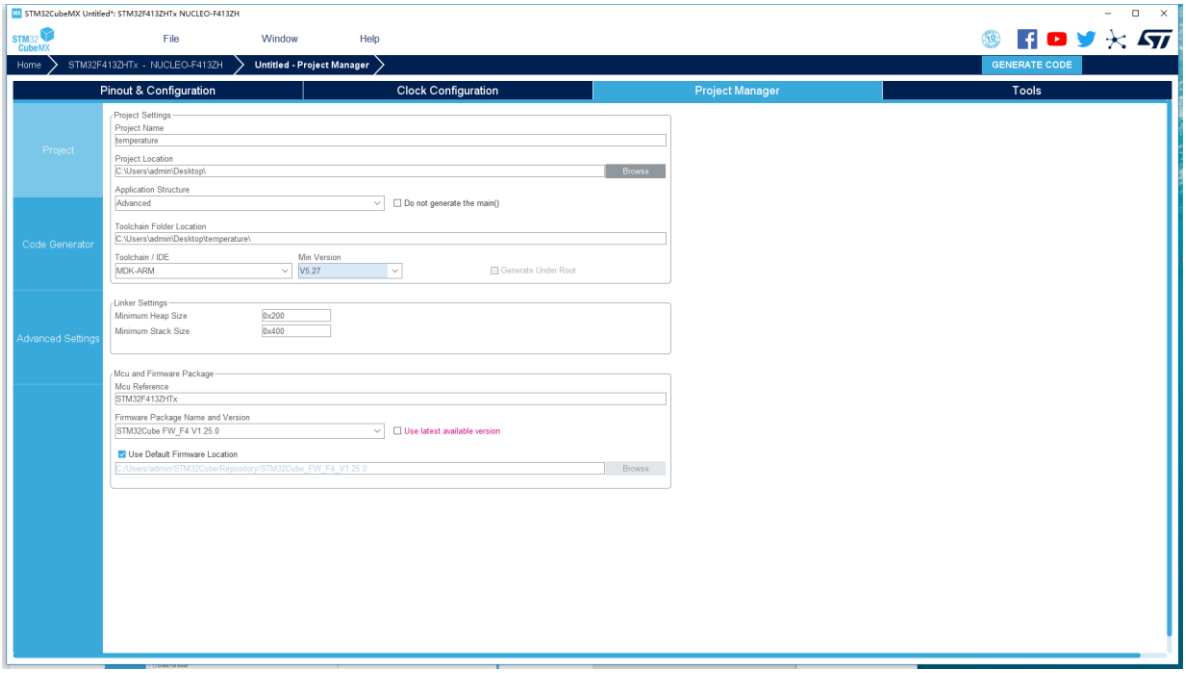


在 Platform Setting 中配置连接方式，按照下图所示的选项连接即可。

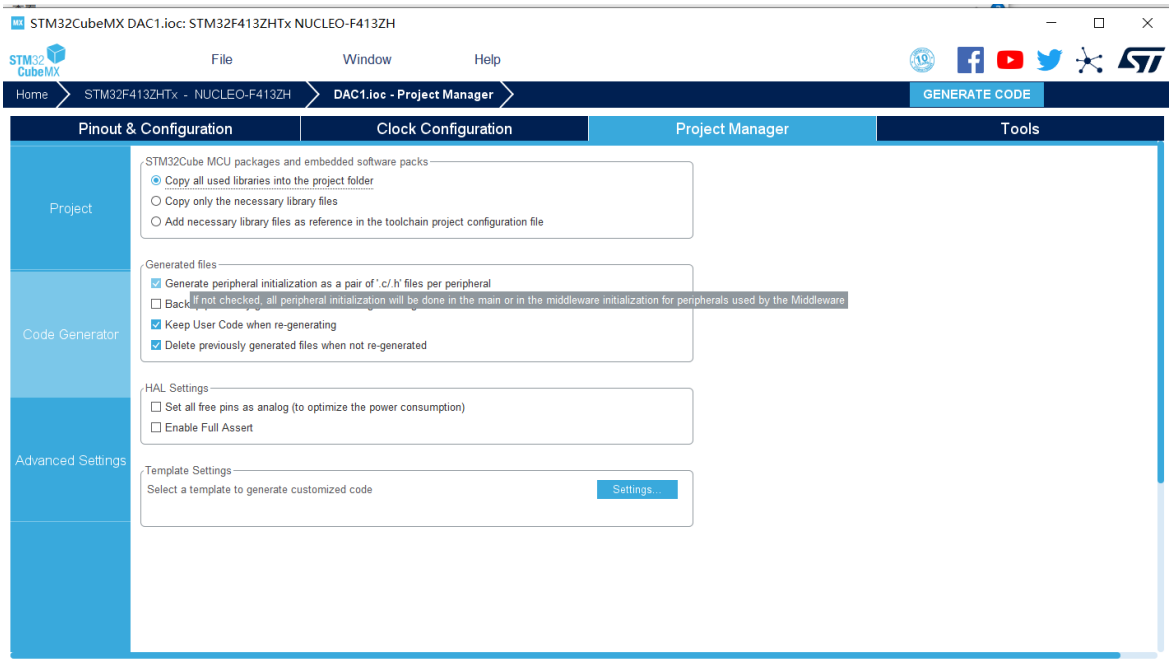


创建工程后填写以下的工程信息。

- Project Name： 工程名任意即可，这里填写 temperature。
- Project Location： 工程路径，建议新建空文件夹专门存放所有的工程文件，注意路径中不要出现中文字符。
- Toolchain/IDE： 这里选择我们已经安装好的 Keil5。
- 取消勾选 Use latest available version，选择 V1.25.0。



- Code Generator：勾选第一项。



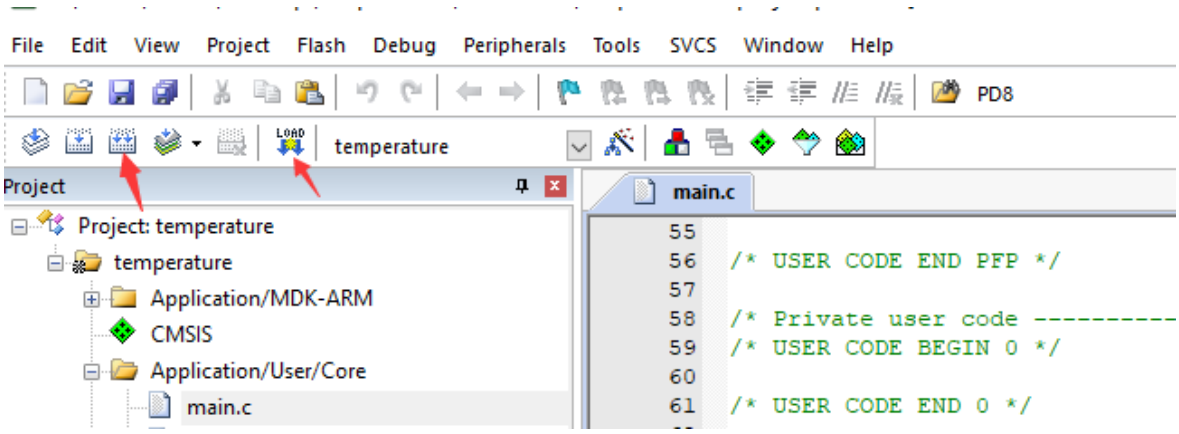
其余保持默认，点击右上角的 GENERATE CODE 生成 Keil 工程。

3. 利用 Keil 添加用户代码

打开 main.c，这里是程序的入口。可以看到 CubeMX 已经为我们做好关于 IKS01A3 的初始化，并且完成了用户代码。

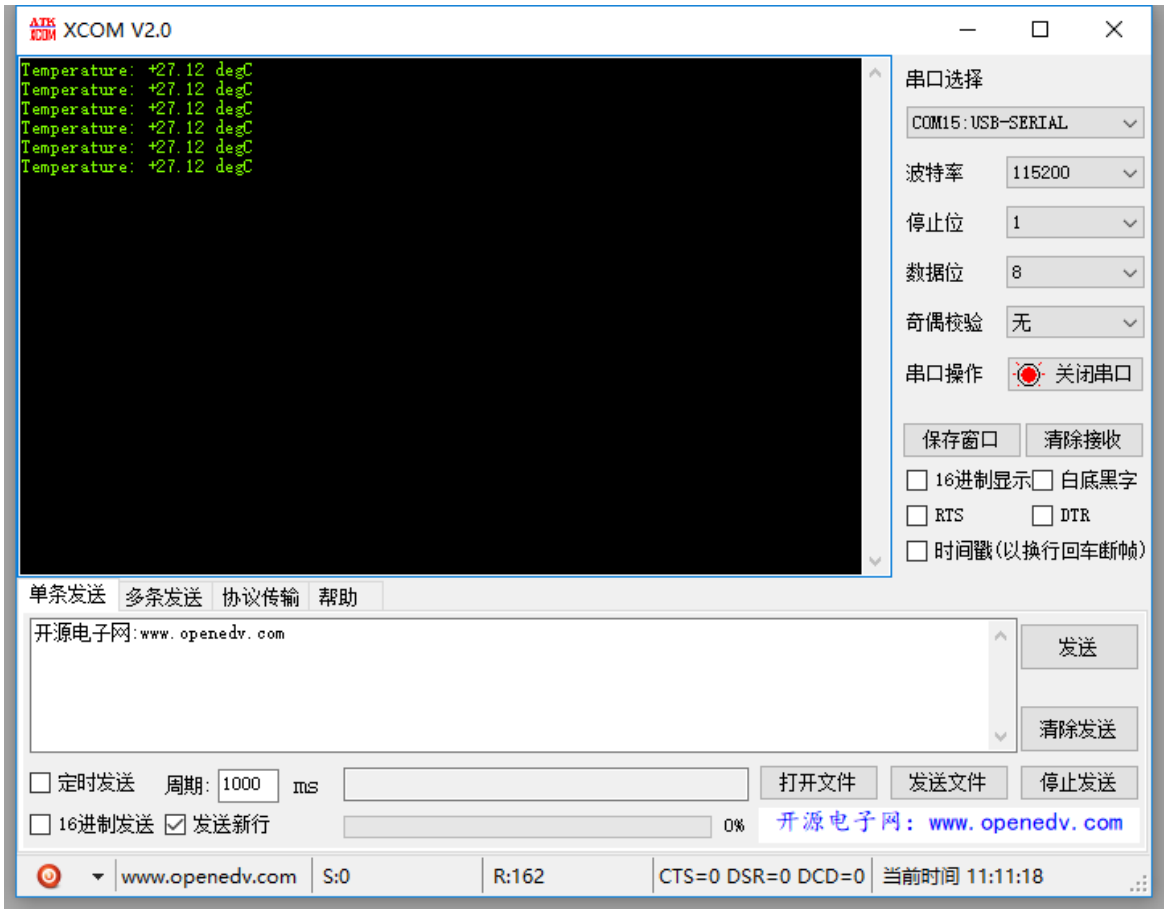
```c
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART3_UART_Init();
  MX_USB_OTG_FS_PCD_Init();
  MX_MEMS_Init();
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

  MX_MEMS_Process();
    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
```

进行编译和代码烧录。



12.5 实验结果

按动开发板的 RESET 按钮，打开串口助手，选择对应的串口、波特率。开启串口后，即可看到对应点温度在不断打印。



第十三章 LSM6DSO 六轴加速度获取实验

13.1 实验目的

1. 学习 IKS01A3 拓展板的使用
2. 学会使用 STM32CubeMX 工具配置 IKS01A3 所需端口

13.2 实验内容

在 STM32CubeMX 中配置 LSM6DSO 所需端口，通过串口输出图形，反映板子的面朝方向

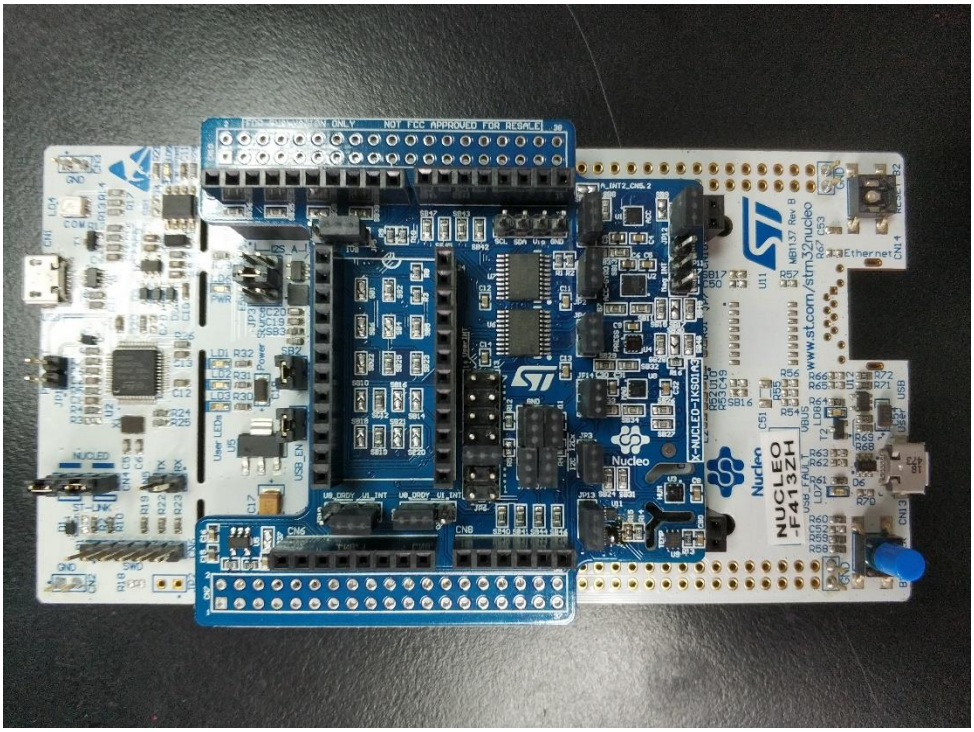
13.3 实验要求

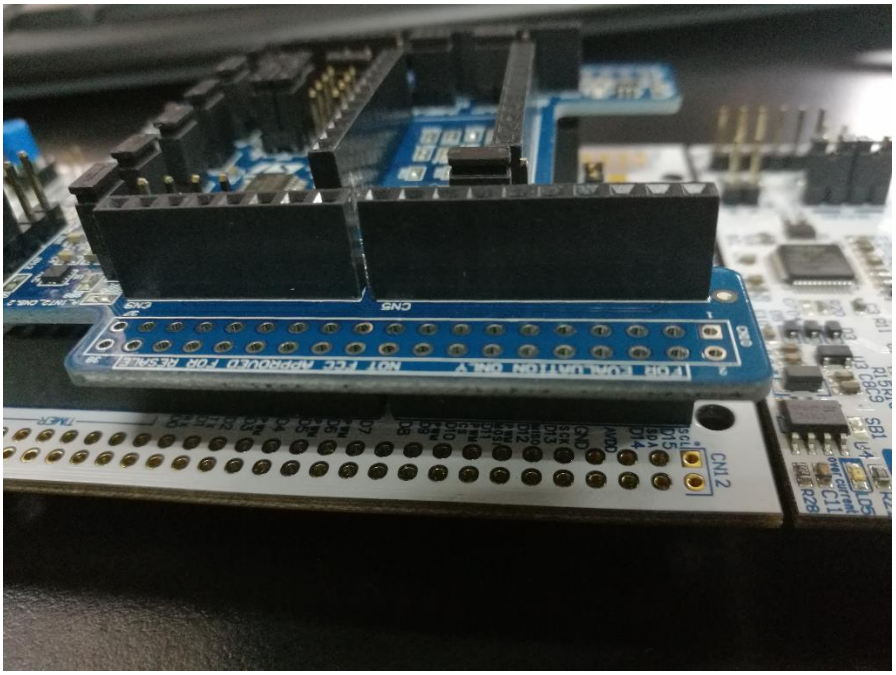
能够在串口上实时获取当前板子的偏转角度以及转动加速度。

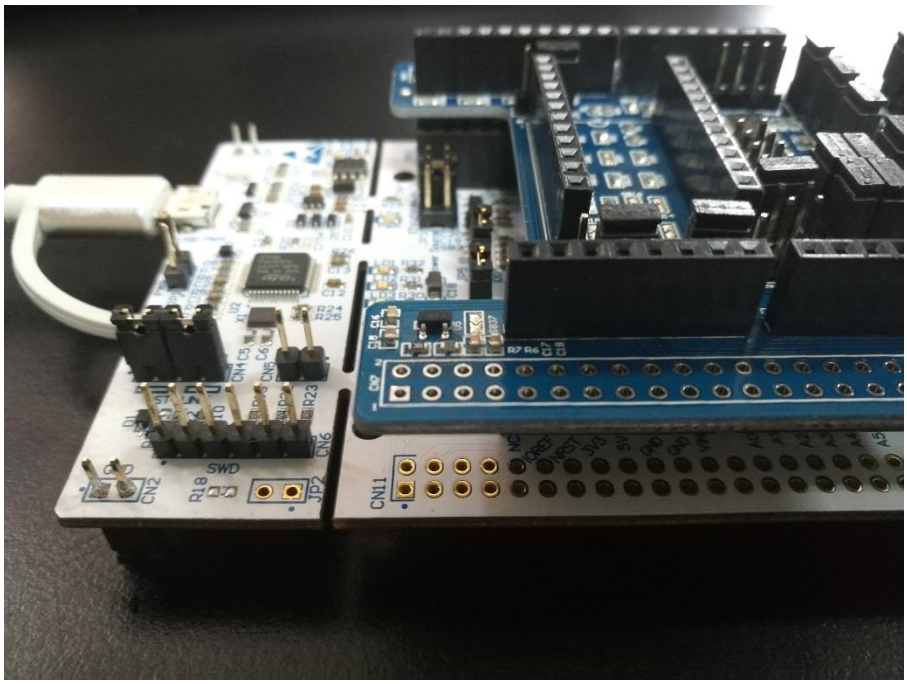
13.4 实验步骤

1. 将 KS01A3 拓展板安装到 NUCLEO-144 开发板上

仔细对照正视图和侧视图，注意扩展版凹口向上，排针与母座的缺口需要对应，两边第一个排针都与第一个外侧母座相对应。

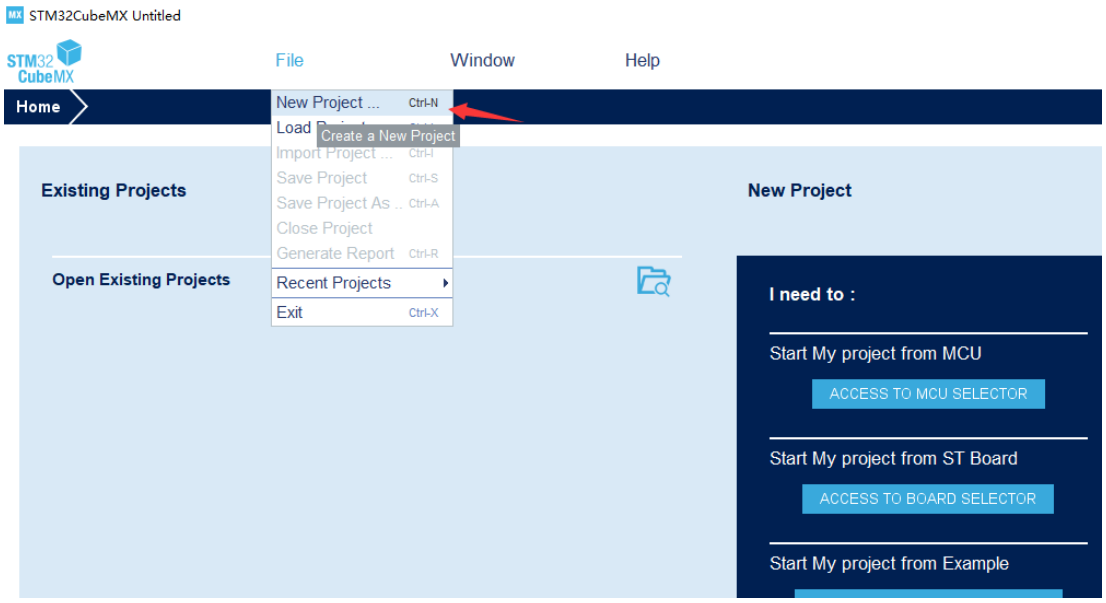




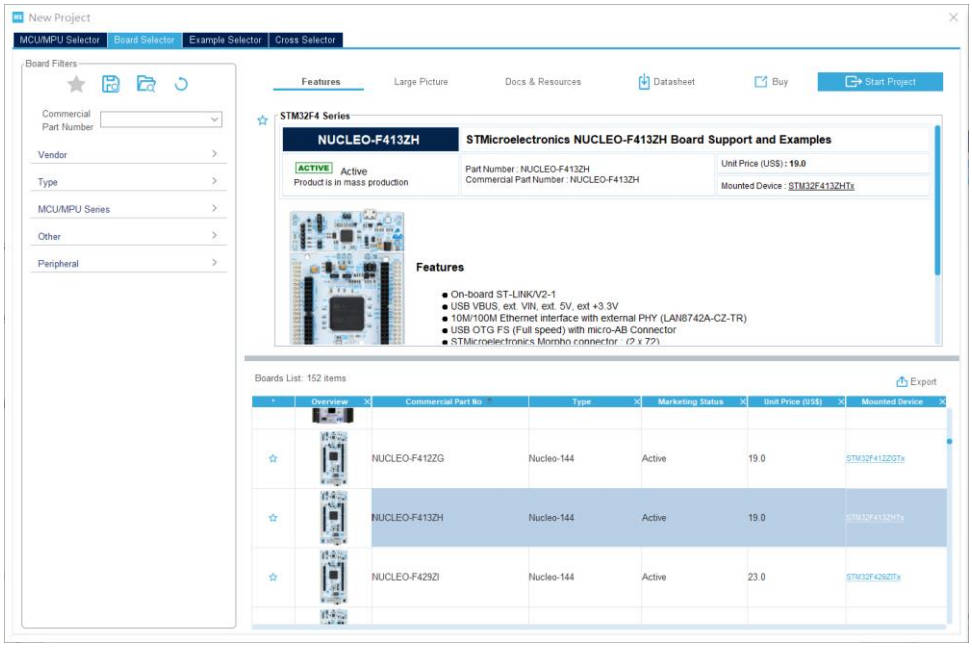


2. 利用 STM32CubeMX 生成模板代码

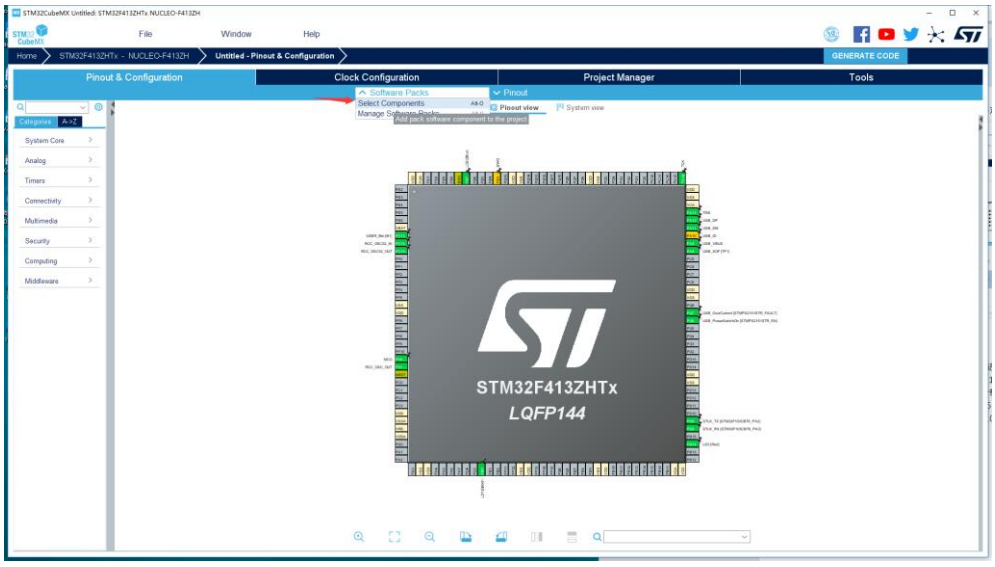
第一步，首先打开 STM32CubeMX 工具，点击如图所示的按钮新建工程。



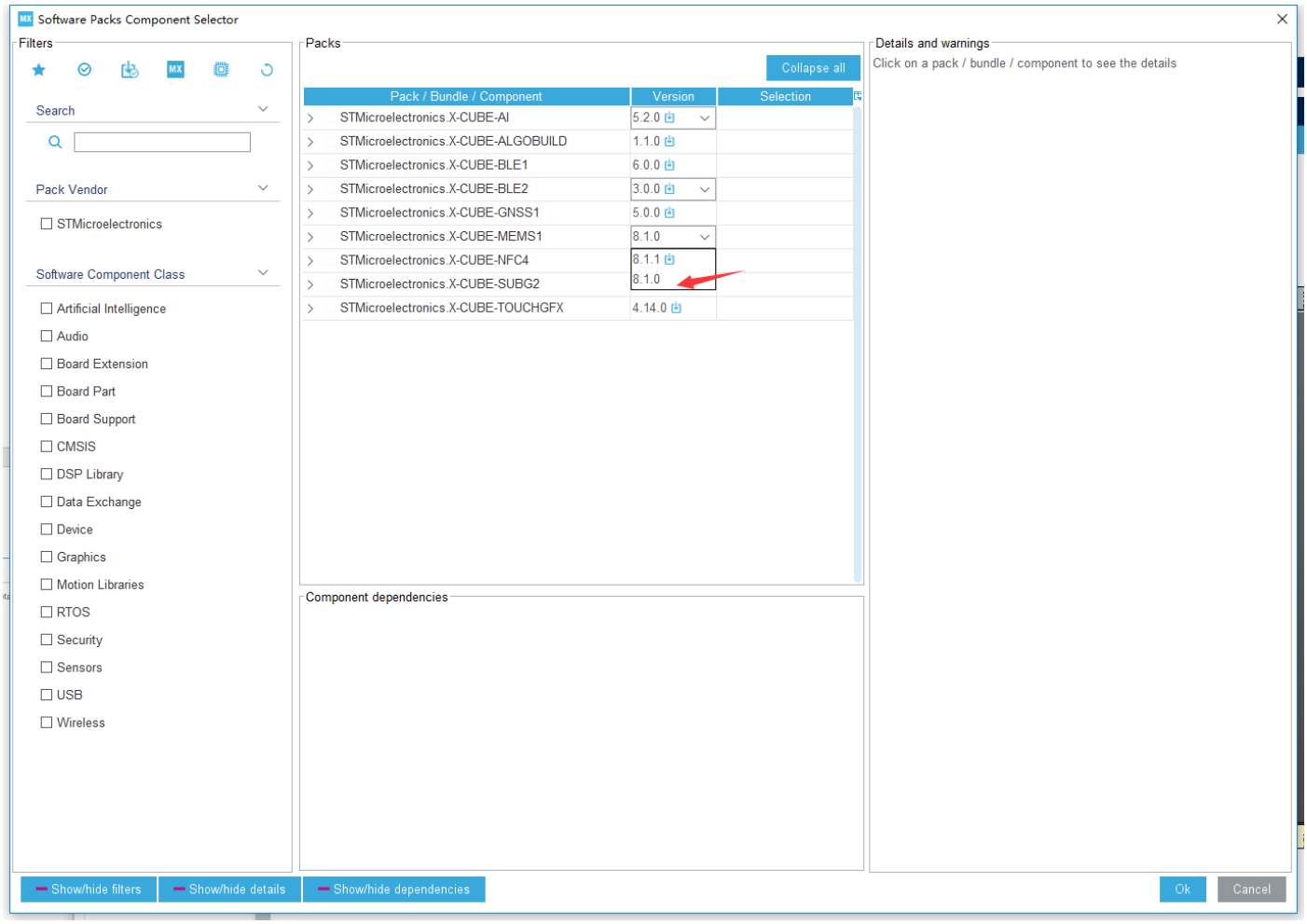
第二步，直接选择对应的 NUCLEO-144 开发板，省去对于时钟、中断等等的配置。

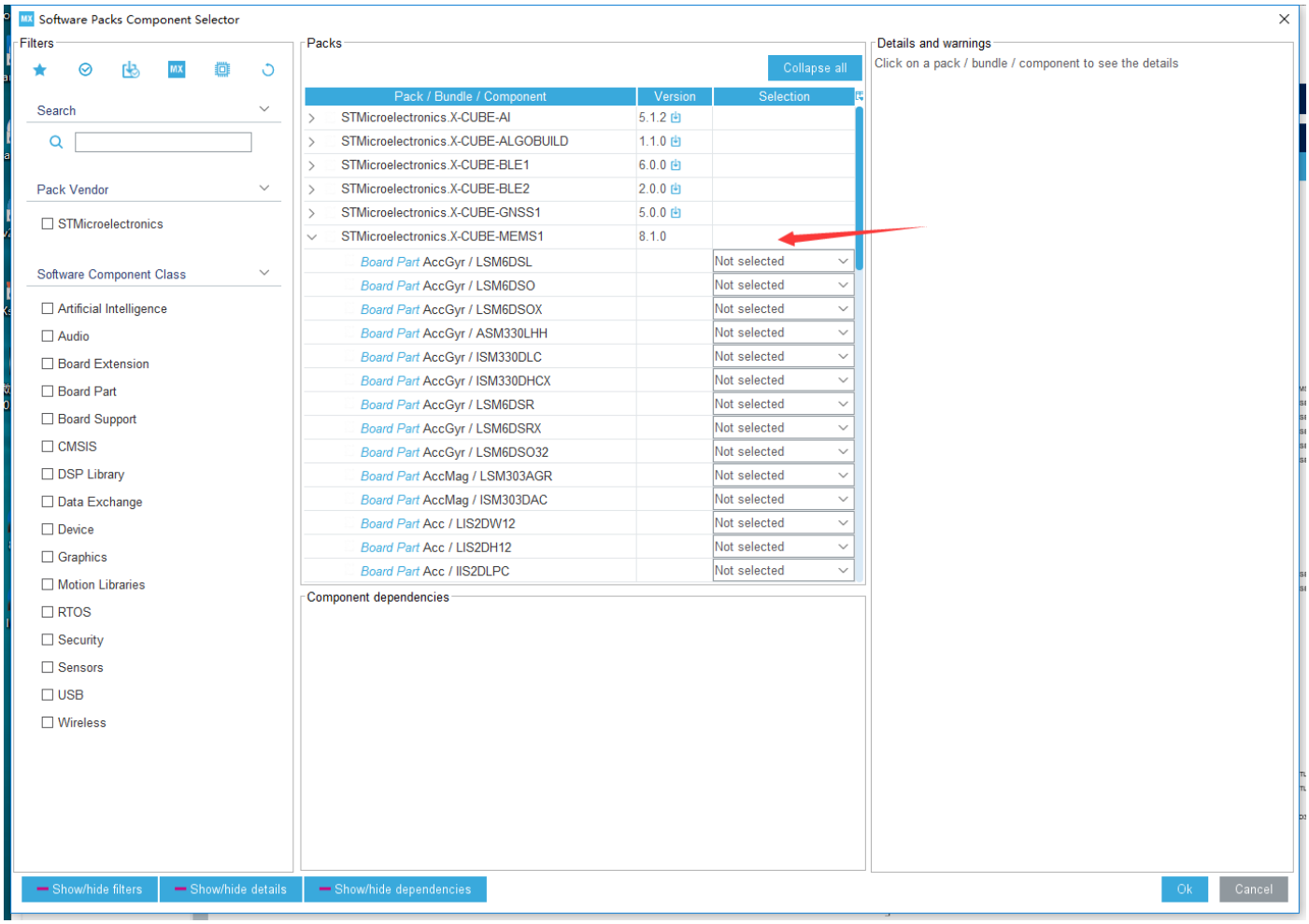


在 Pinout&Configure 界面点击 Select Components

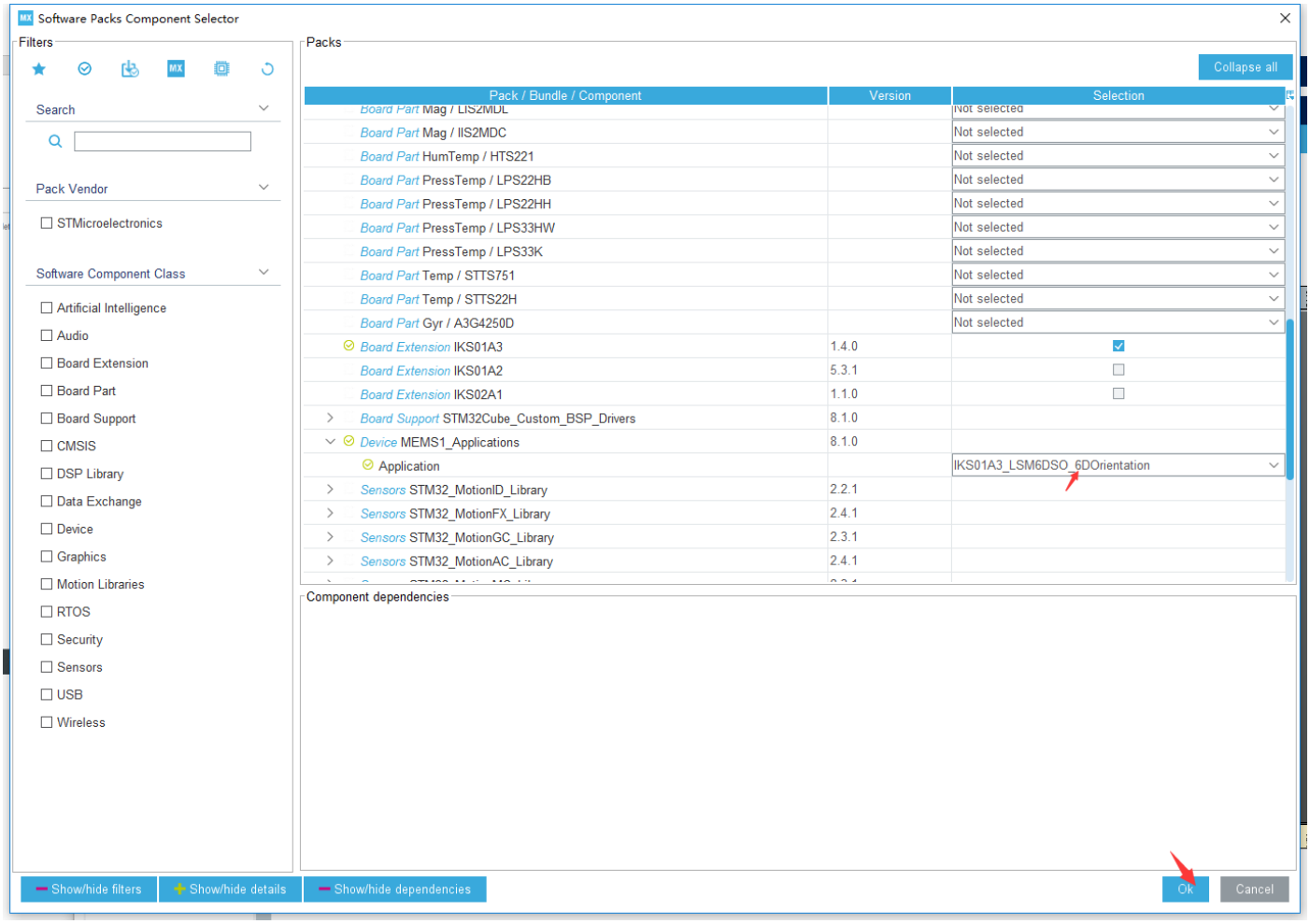


选择 STMmicroelectronics X-CUBE-MEMS1，并使用 8.1.0 版本





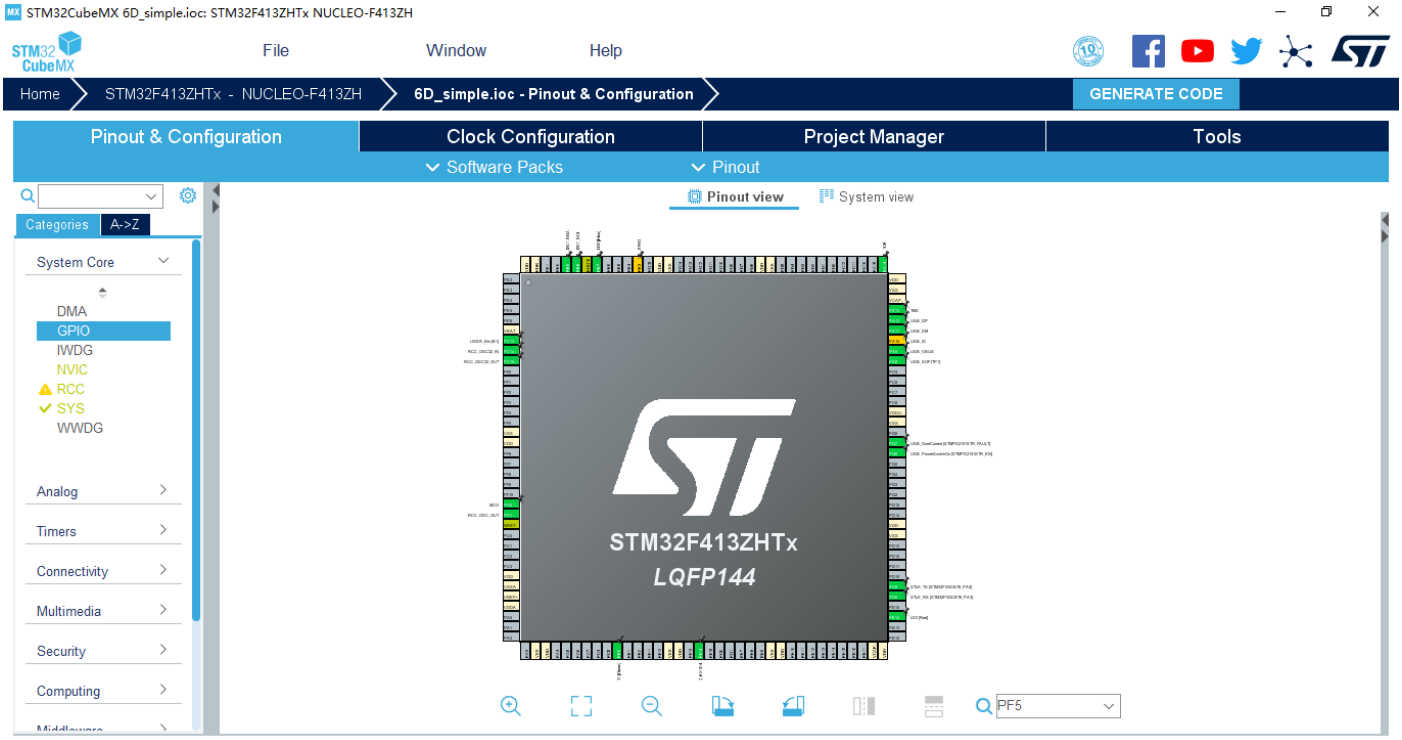
选中 IKS01A3，并在应用中选择 LSM6DSO_6DOrientation，点击 OK



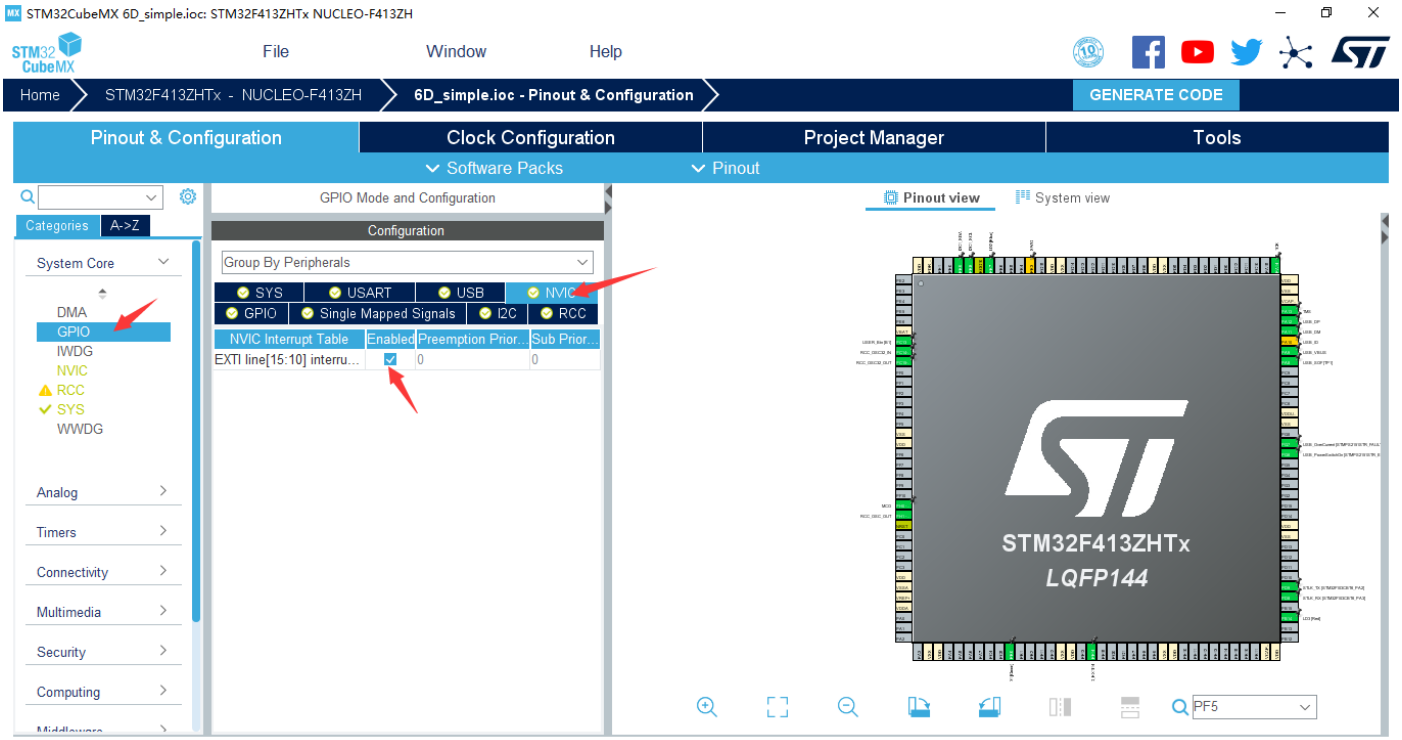
回到引脚配置界面，对以下引脚进行配置

- PB8：　I2C1_SCL
- PB9：　I2C1_SDA
- PF14：　GPIO_EXITI14

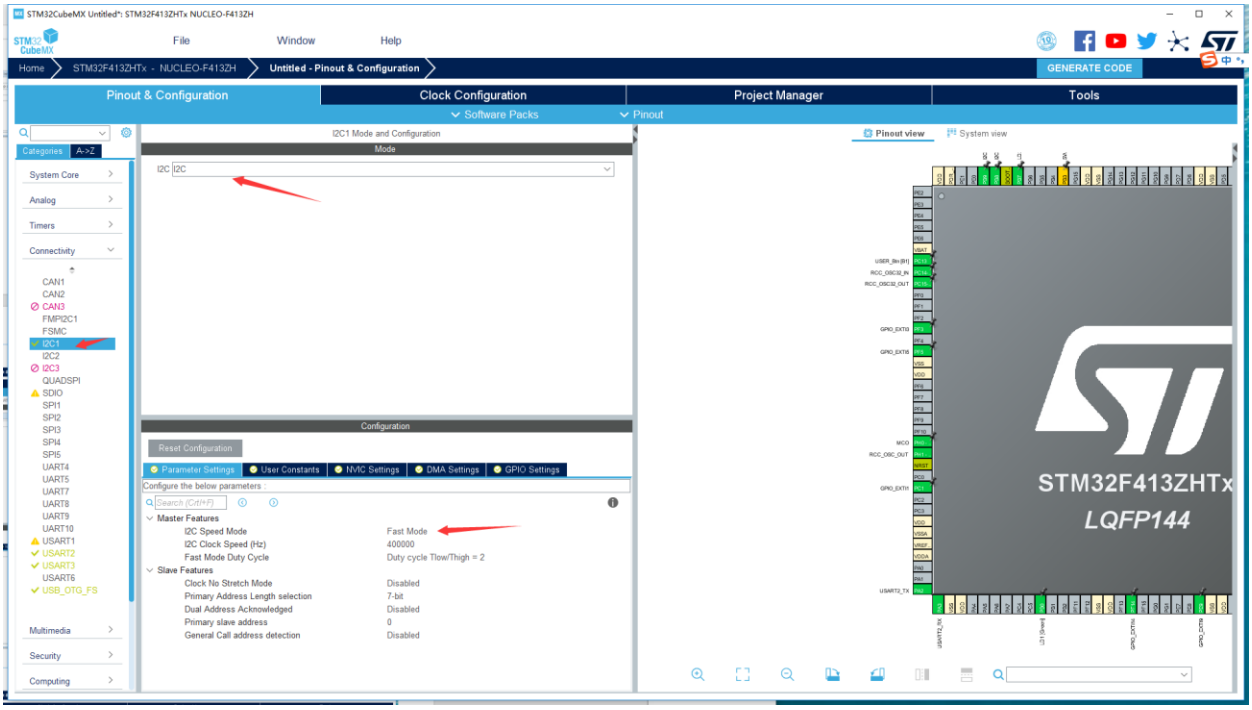
引脚配置完成后如下图所示



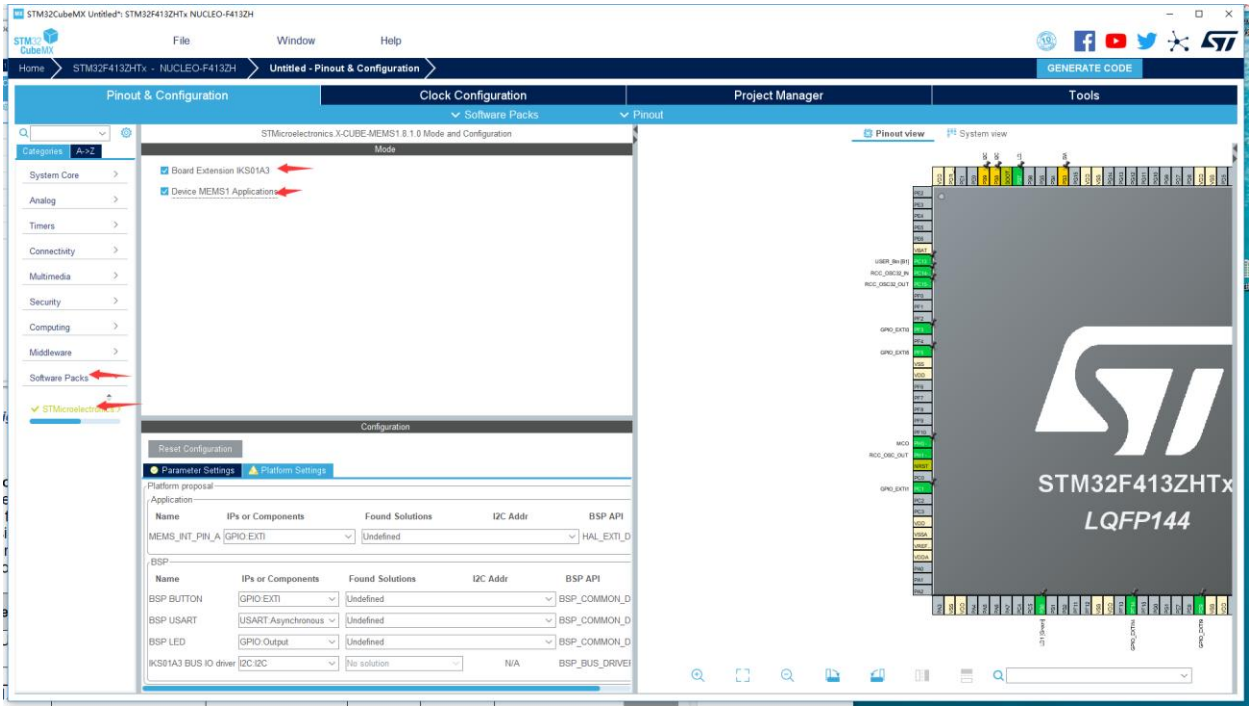
到 NVIC 界面使能如下图所示的中断



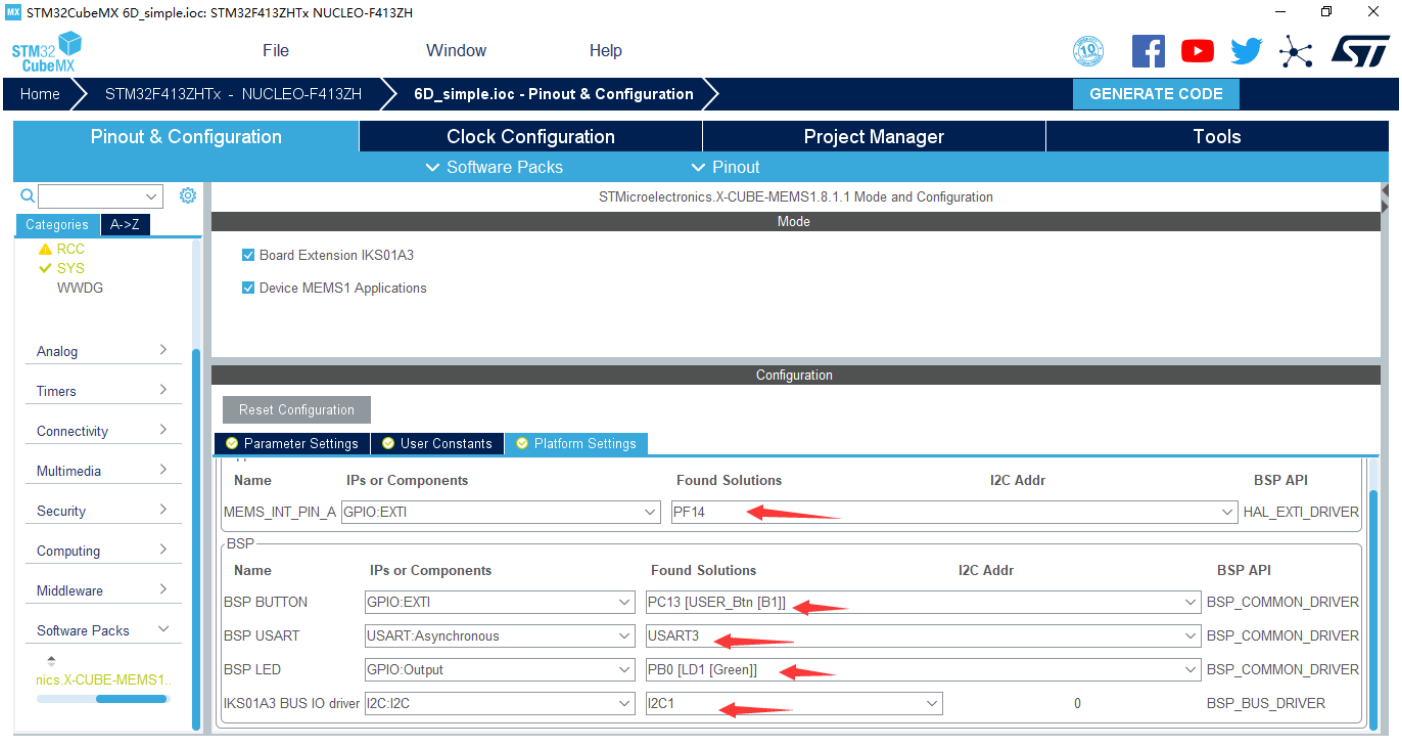
开启 I2C1，并将模式改成 Fast Mode，其余保持默认



在 SoftWare Packs 对拓展版进行配置，这里我们使能了 IKS01A3。

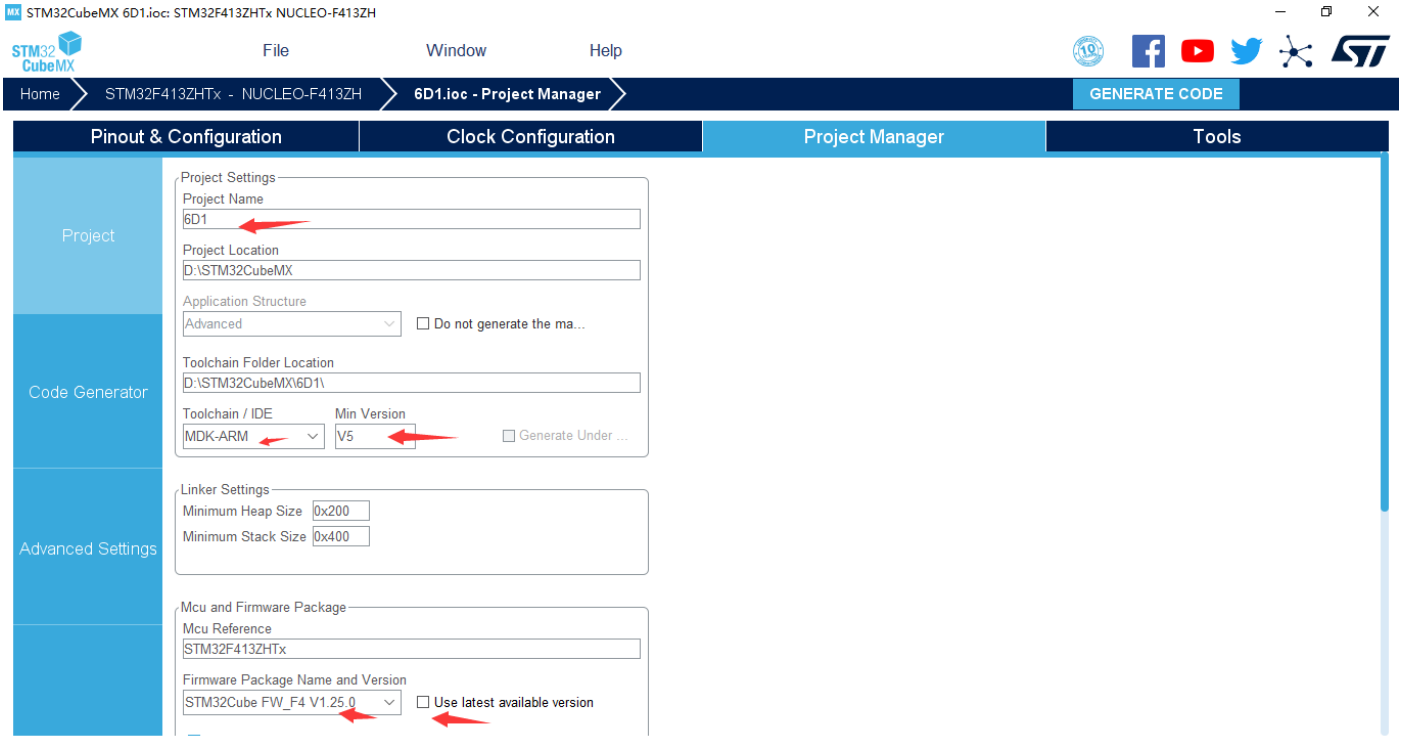


在 Platform Setting 中配置连接方式，按照下图所示的选项连接即可。

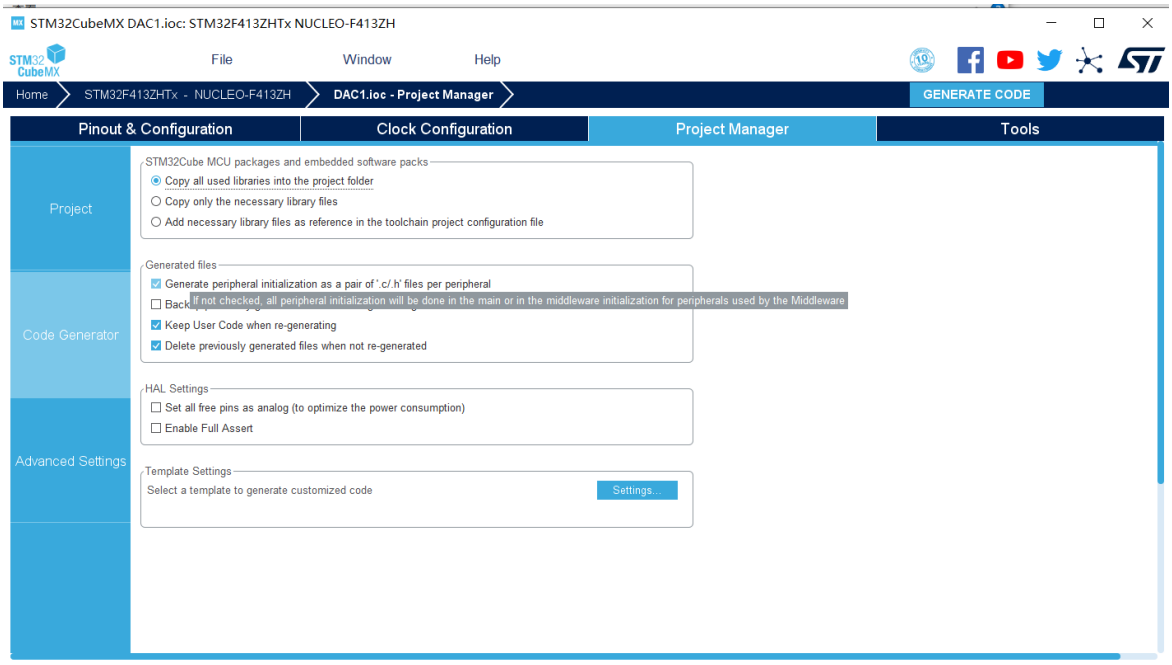


创建工程后填写以下的工程信息。

- Project Name： 工程名任意即可，这里填写 6D。
- Project Location： 工程路径，建议新建空文件夹专门存放所有的工程文件，注意路径中不要出现中文字符。
- Toolchain/IDE： 这里选择我们已经安装好的 Keil5。
- 取消勾选 Use latest available version，选择 V1.25.0。



- Code Generator：勾选第一项。



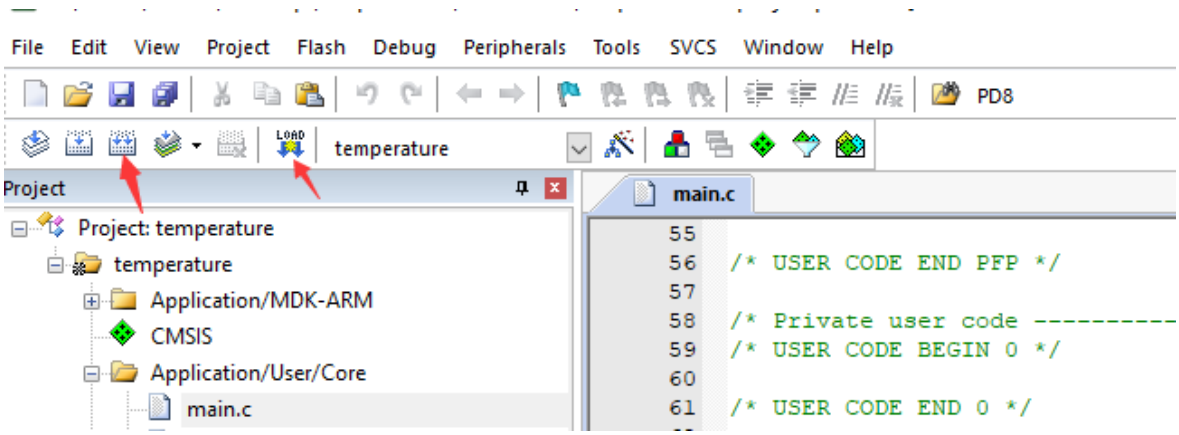
其余保持默认，点击右上角的 GENERATE CODE 生成 Keil 工程。

3. 利用 Keil 添加用户代码

打开 main.c，这里是程序的入口。可以看到 CubeMX 已经为我们做好关于 IKS01A3 的初始化，并且完成了用户代码。

```c
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART3_UART_Init();
  MX_USB_OTG_FS_PCD_Init();
  MX_MEMS_Init();
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

  MX_MEMS_Process();
    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
```

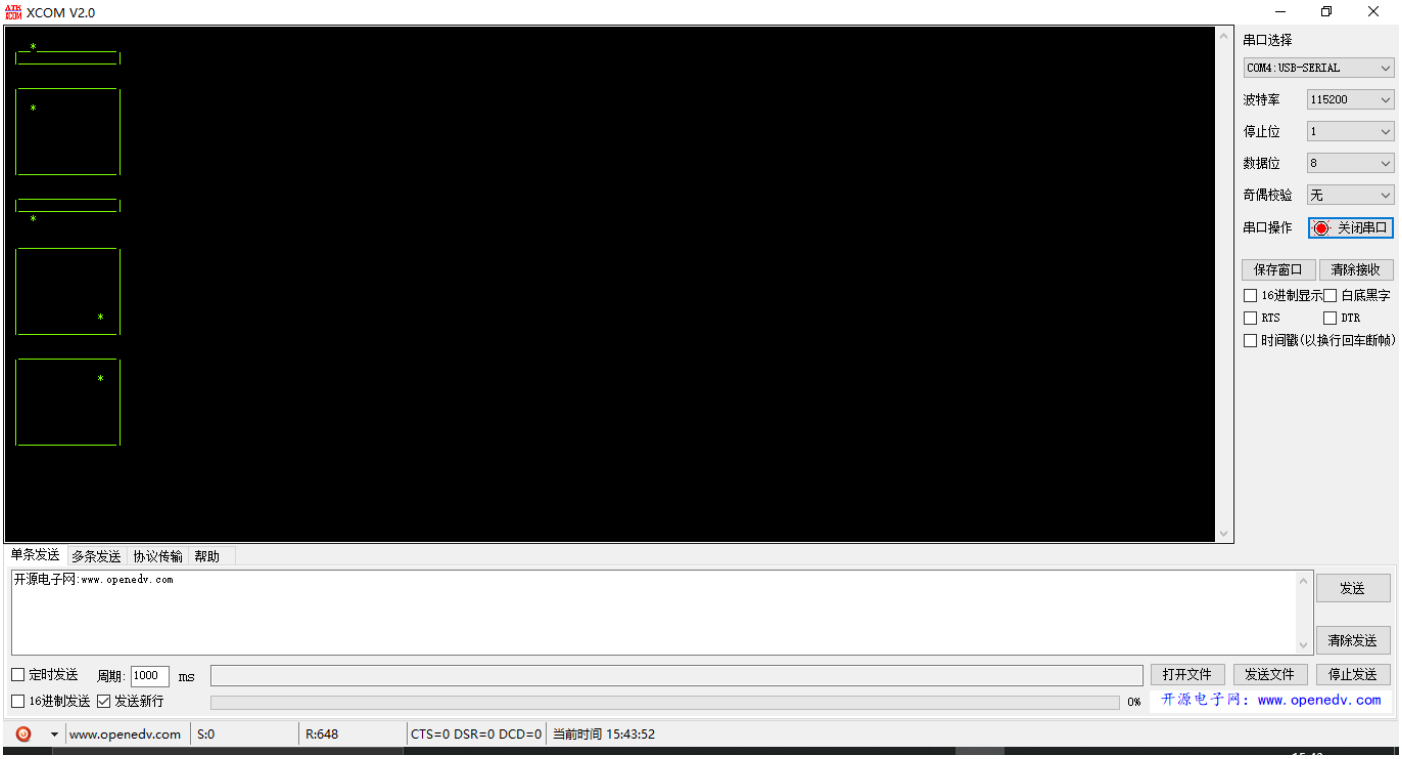
进行编译和代码烧录。



13.5 实验结果

按动开发板的 RESET 按钮，打开串口助手，选择对应的串口、波特率。

开启串口后，旋转开发板，即可看到对应开发板不同位置的形貌打印。



第十四章 FreeRTOS 实验

14.1 实验目的

1. 学习 FreeRTOS 的使用
2. 学习利用线程操作 NUCLEO-144 外设

14.2 实验内容

利用线程分别控制 LED1、LED2、LED3

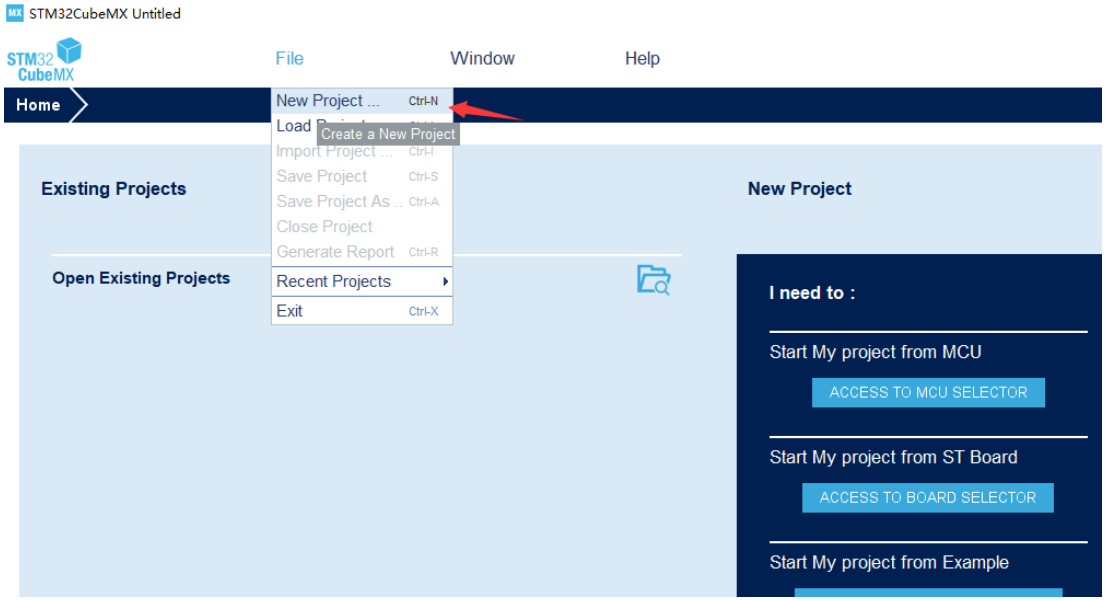
14.3 实验要求

Nucleo-144 上 LD1、LD2、LD3 分别以 100ms、200ms、500ms 的时间间隔改变状态

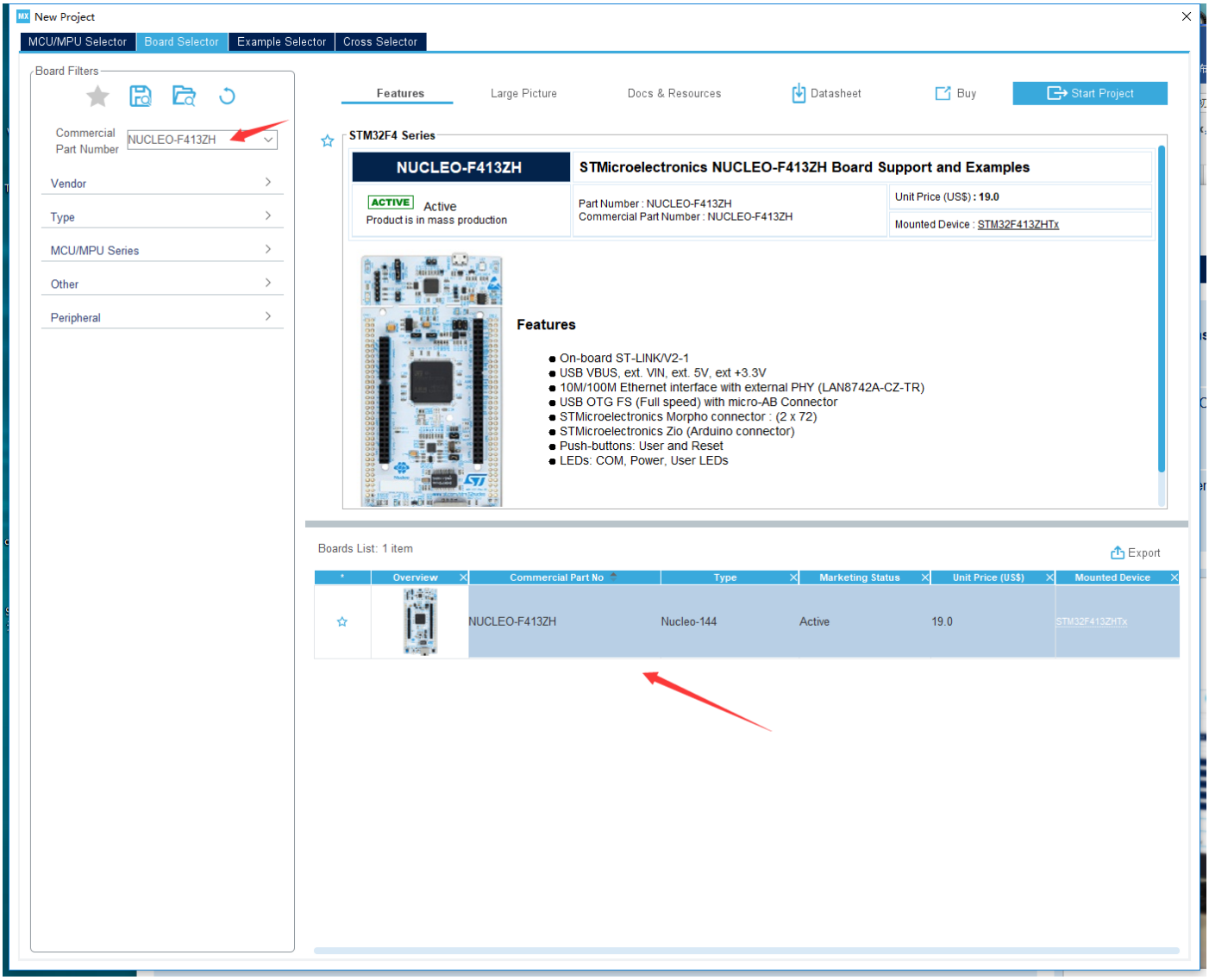
14.4 实验步骤

3. 利用 STM32CubeMX 生成模板代码

第一步，首先打开 STM32CubeMX 工具，点击如图所示的按钮新建工程。

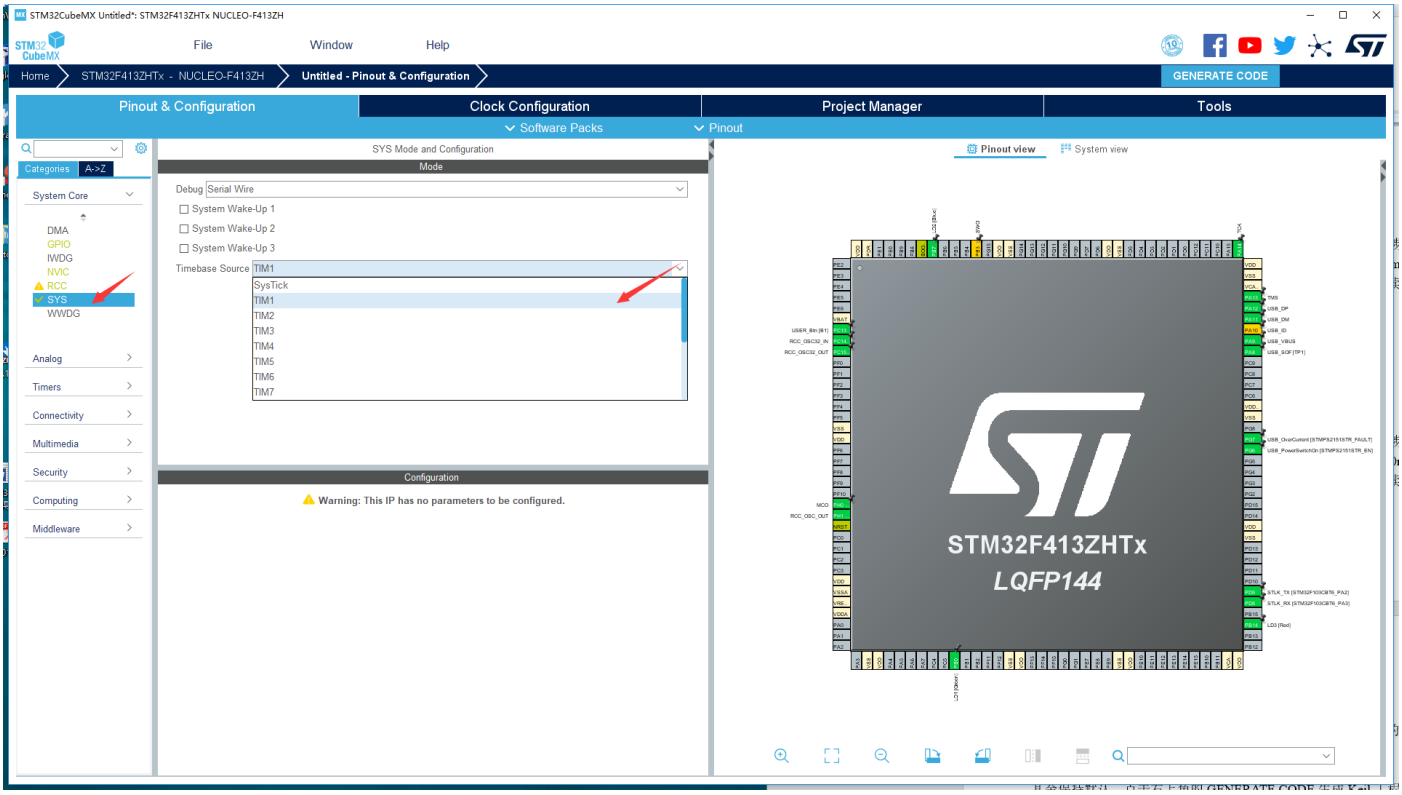


第二步，直接选择对应的 NUCLEO-144 开发板，完成基本配置。

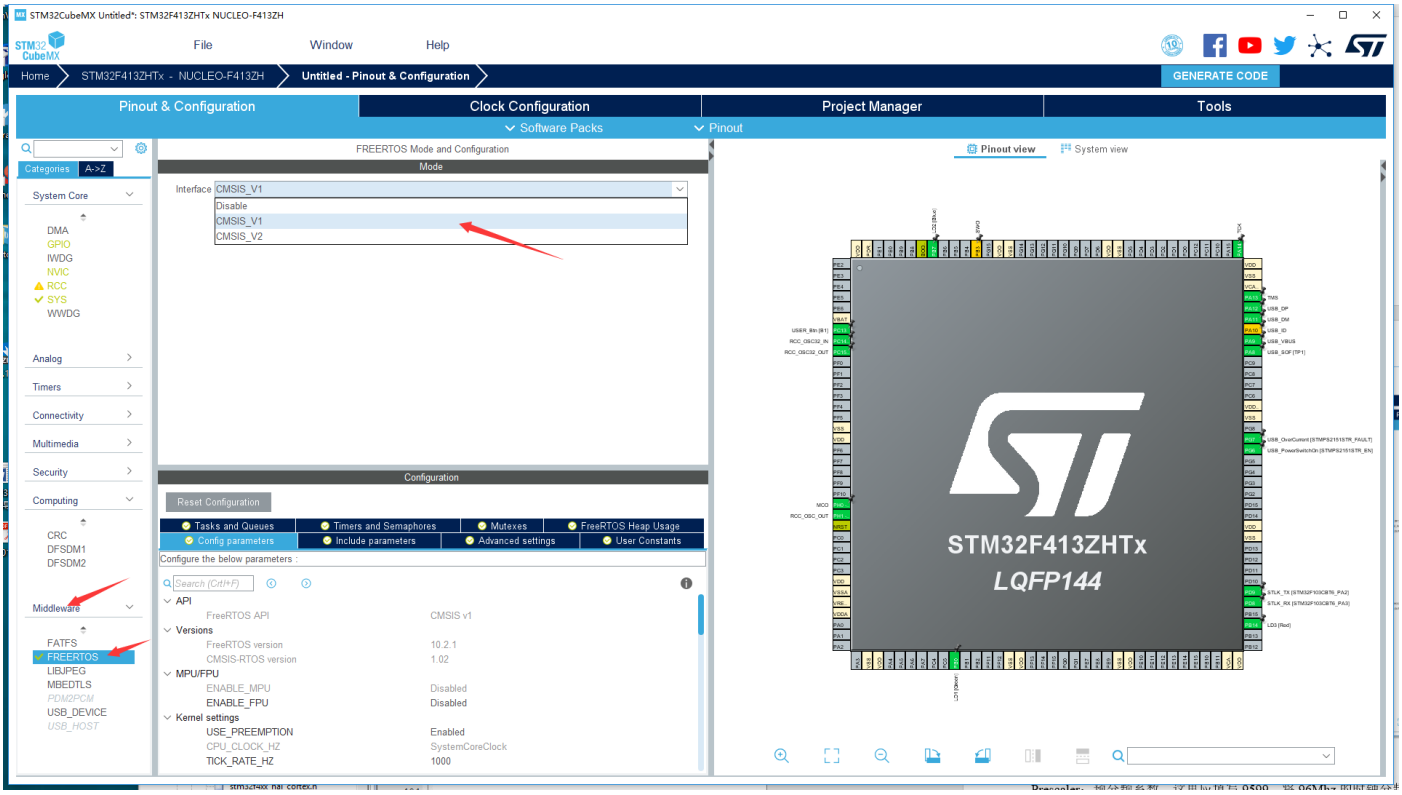


第三步，进入 SYS 界面，将时基源由 SysTick 改为 TIM1。

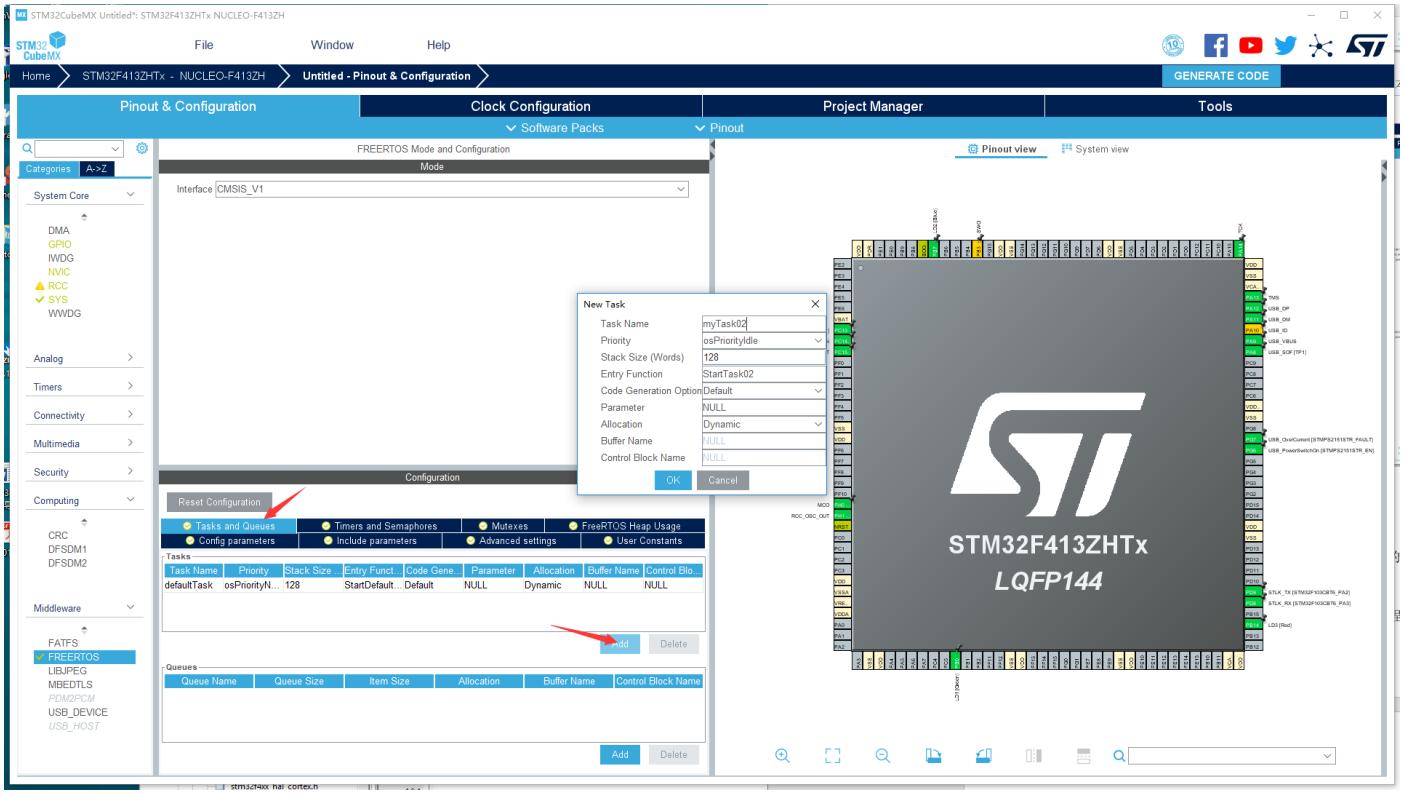
时基源表示 HAL 库所使用的时钟来源。FreeRTOS 的接口默认使用 SysTick，倘若 HAL 库同样使用 SysTick 则可能会产生错误（例如 HAL 库中的 HAL_Delay 与 FreeRTOS 中的 osDelay 会冲突），因此将 HAL 库的时钟源改为定时器以避免类似情况。



第四步，在 Middleware 界面开启 FreeRTOS，并使用 CMSIS_V1 版本接口。



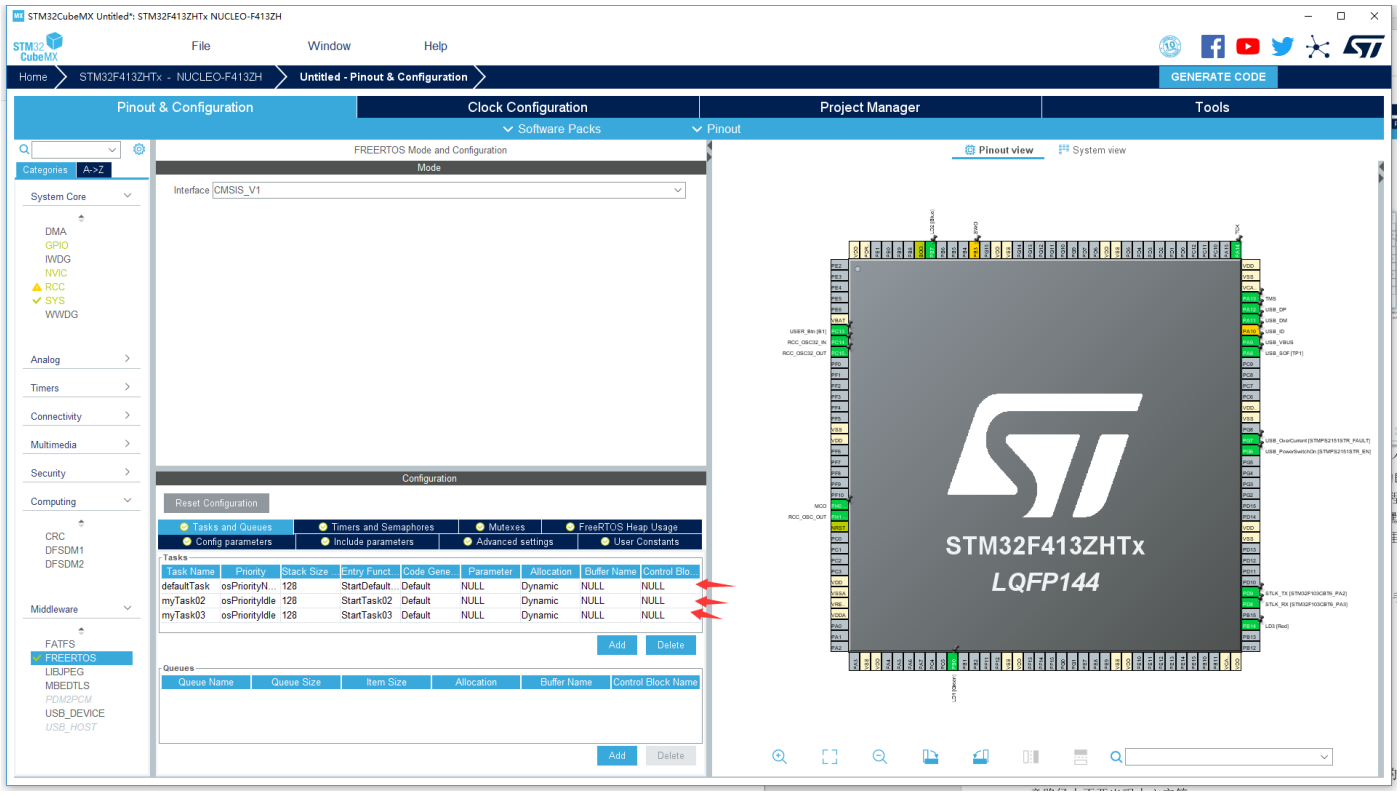
第五步，在 Tasks and Queues 界面创建线程。

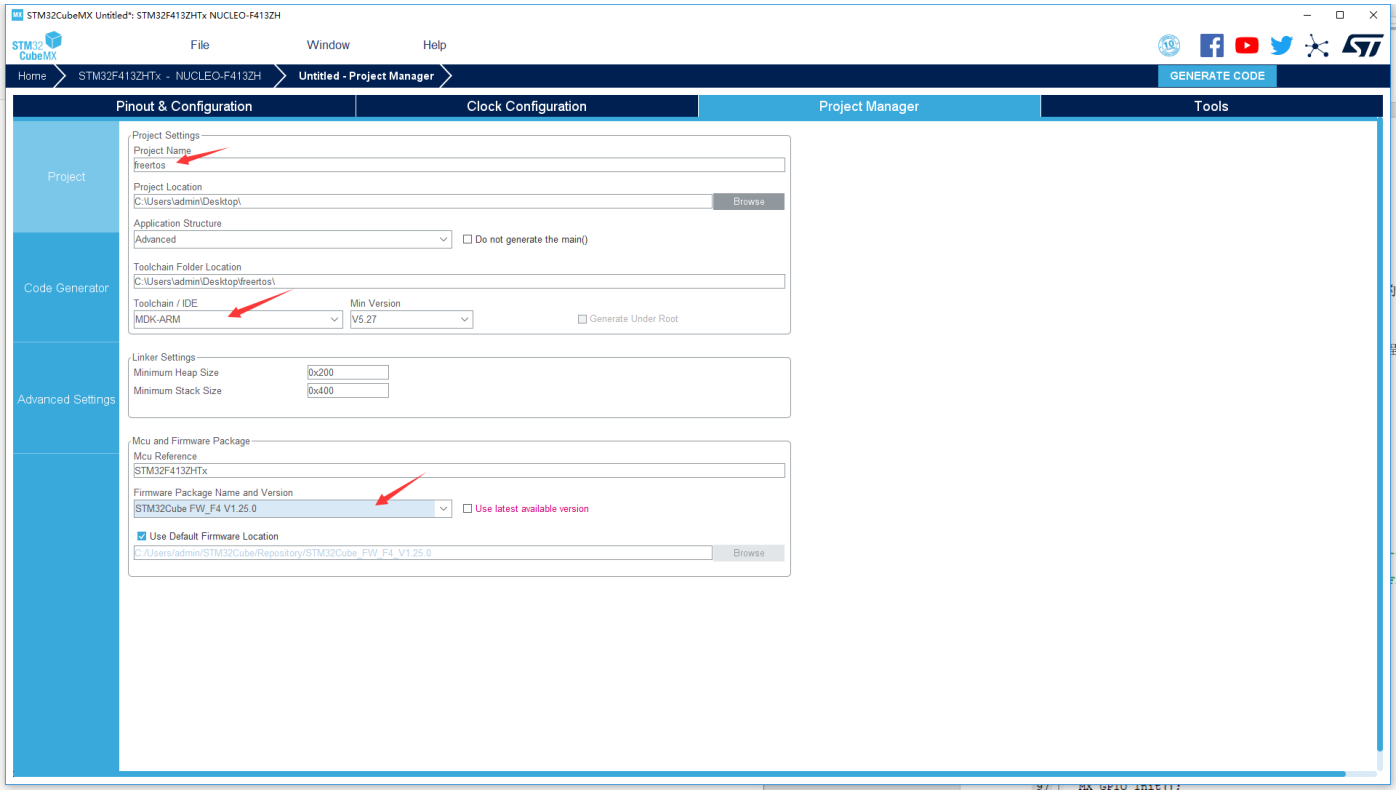


界面中已经为用户创建了默认线程，我们仍需要创建两个线程，各个参数含义如下

- Task Name：任务名，用于区分不同线程。这里使用默认生成的即可
- Priority：优先级。这里选择最低级即可。注意：低优先级的线程将无法抢占高优先级使用的时间片
- Stack Size：堆栈空间。为该线程分配能使用的内存大小，这里默认 128 即可
- Entry Function：线程函数名。表示线程的执行函数的名字。这里使用默认生成的即可
- Code Generation Option： 代码生成选项，使用默认即可
- Parameter：传入参数，这里选择 NULL
- Allocation：分配方式，这里选择 Dynamic 动态分配，否则需要手动修改缓冲区和控制块的相关配置。

线程创建结束后，界面内将有默认线程、myTask02、myTask03 共三个线程。





第六步，创建工程后填写一下的工程信息。

- Project Name： 工程名任意即可，这里填写 freertos。
- Project Location： 工程路径，建议新建空文件夹专门存放所有的工程文件，这里在桌面新建了 freertos 文件夹。注意路径中不要出现中文字符。
- Toolchain/IDE： 这里选择我们已经安装好的 Keil5。
- Firmware Package Name and Version： 取消勾选使用最新版本，并切换至 1.25.0 版本。
- 其余保持默认，点击右上角的 GENERATE CODE 生成 Keil 工程。

4. 利用 Keil 添加用户代码

```c
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART3_UART_Init();
  MX_USB_OTG_FS_PCD_Init();
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */

  /* USER CODE BEGIN RTOS_MUTEX */
  /* add mutexes, ... */
  /* USER CODE END RTOS_MUTEX */

  /* USER CODE BEGIN RTOS_SEMAPHORES */
  /* add semaphores, ... */
  /* USER CODE END RTOS_SEMAPHORES */

  /* USER CODE BEGIN RTOS_TIMERS */
  /* start timers, add new ones, ... */
  /* USER CODE END RTOS_TIMERS */

  /* USER CODE BEGIN RTOS_QUEUES */
  /* add queues, ... */
  /* USER CODE END RTOS_QUEUES */

  /* Create the thread(s) */
  /* definition and creation of defaultTask */
  osThreadDef(defaultTask, StartDefaultTask, osPriorityNormal, 0, 128);
  defaultTaskHandle = osThreadCreate(osThread(defaultTask), NULL);

  /* definition and creation of myTask02 */
  osThreadDef(myTask02, StartTask02, osPriorityIdle, 0, 128);
  myTask02Handle = osThreadCreate(osThread(myTask02), NULL);

  /* definition and creation of myTask03 */
  osThreadDef(myTask03, StartTask03, osPriorityIdle, 0, 128);
  myTask03Handle = osThreadCreate(osThread(myTask03), NULL);

  /* USER CODE BEGIN RTOS_THREADS */
  /* add threads, ... */
```

利用 Keil 打开工程以后，打开 main.c，这里是程序的入口。可以发现 STM32cube 已经帮我们完成了关于 FreeRTOS 的初始化，并创建、启动了我们在 cubeMX 中配置的三个线程。

```
336  /* USER CODE END Header_StartDefaultTask */
337  void StartDefaultTask(void const * argument)
338  {
339    /* USER CODE BEGIN 5 */
340    /* Infinite loop */
341    for(;;)
342    {
343      osDelay(1);
344    }
345    /* USER CODE END 5 */
346  }
347
348  /* USER CODE BEGIN Header_StartTask02 */
349  /**
350  * @brief Function implementing the myTask02 thread.
351  * @param argument: Not used
352  * @retval None
353  */
354  /* USER CODE END Header_StartTask02 */
355  void StartTask02(void const * argument)
356  {
357    /* USER CODE BEGIN StartTask02 */
358    /* Infinite loop */
359    for(;;)
360    {
361      osDelay(1);
362    }
363    /* USER CODE END StartTask02 */
364  }
365
366  /* USER CODE BEGIN Header_StartTask03 */
367  /**
368  * @brief Function implementing the myTask03 thread.
369  * @param argument: Not used
370  * @retval None
371  */
372  /* USER CODE END Header_StartTask03 */
373  void StartTask03(void const * argument)
374  {
375    /* USER CODE BEGIN StartTask03 */
376    /* Infinite loop */
377    for(;;)
378    {
379      osDelay(1);
380    }
381    /* USER CODE END StartTask03 */
382  }
```

在下方 337 行以下，已经帮我们构建好了三个线程对应的执行函数 StartDefaultTask、StartTask02、与 StartTask03。

将 StartDefaultTask 函数内的 for 循环改成如下所示语句。

```
for(;;)
{
  HAL_GPIO_TogglePin(LD1_GPIO_Port,LD1_Pin);
  osDelay(100);
}
```

将 StartTask02 函数内的 for 循环改成如下所示语句。

```
for(;;)
{
    HAL_GPIO_TogglePin(LD2_GPIO_Port,LD2_Pin);
    osDelay(200);
}
```

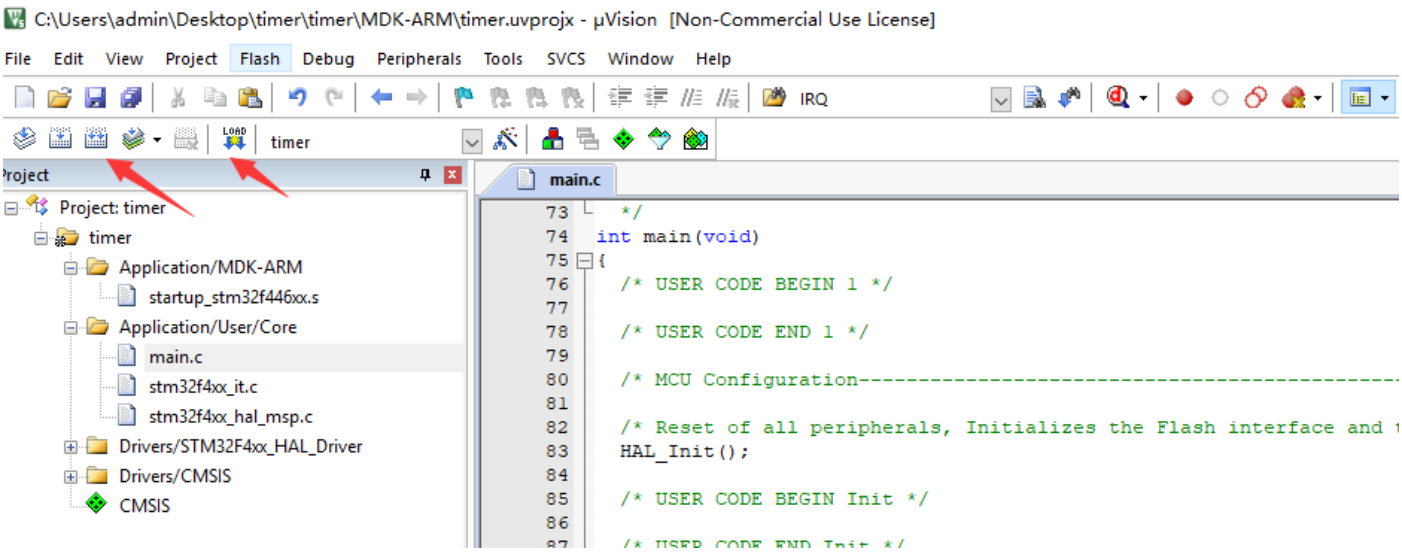
将 StartTask03 函数内的 for 循环改成如下所示语句。

```
for(;;)
{
    HAL_GPIO_TogglePin(LD3_GPIO_Port,LD3_Pin);
    osDelay(500);
}
```

修改完的代码如图

```c
/* USER CODE END Header_StartDefaultTask */
void StartDefaultTask(void const * argument)
{
  /* USER CODE BEGIN 5 */
  /* Infinite loop */
  for(;;)
  {
    HAL_GPIO_TogglePin(LD1_GPIO_Port,LD1_Pin);
    osDelay(100);
  }
  /* USER CODE END 5 */
}

/* USER CODE BEGIN Header_StartTask02 */
/**
* @brief Function implementing the myTask02 thread.
* @param argument: Not used
* @retval None
*/
/* USER CODE END Header_StartTask02 */
void StartTask02(void const * argument)
{
  /* USER CODE BEGIN StartTask02 */
  /* Infinite loop */
  for(;;)
  {
    HAL_GPIO_TogglePin(LD2_GPIO_Port,LD2_Pin);
    osDelay(200);
  }
  /* USER CODE END StartTask02 */
}

/* USER CODE BEGIN Header_StartTask03 */
/**
* @brief Function implementing the myTask03 thread.
* @param argument: Not used
* @retval None
*/
/* USER CODE END Header_StartTask03 */
void StartTask03(void const * argument)
{
  /* USER CODE BEGIN StartTask03 */
  /* Infinite loop */
  for(;;)
  {
    HAL_GPIO_TogglePin(LD3_GPIO_Port,LD3_Pin);
    osDelay(500);
  }
  /* USER CODE END StartTask03 */
}
```

利用图示的两个按钮进行编译和代码烧录。



14.5 实验结果

代码烧录完成之后，按动开发板右下角的 RESET 按钮即可开始运行程序，可见 LED1、LED2、LED3 开始以不同频率闪烁，LED1 每 100ms 改变一次状态，LED2 每 200ms 改变一次状态，LED3 每 500ms 改变一次状态。