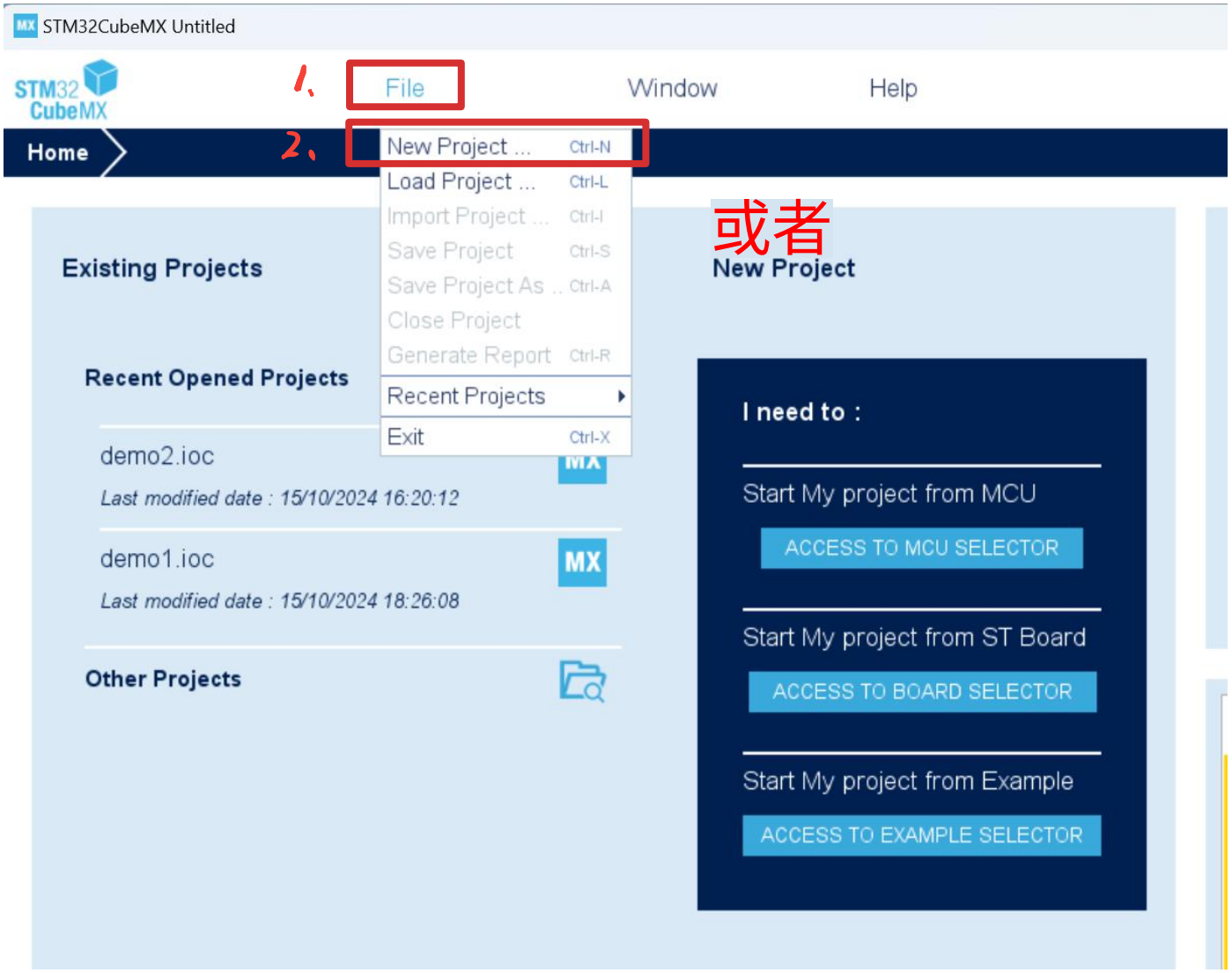
单片机入门

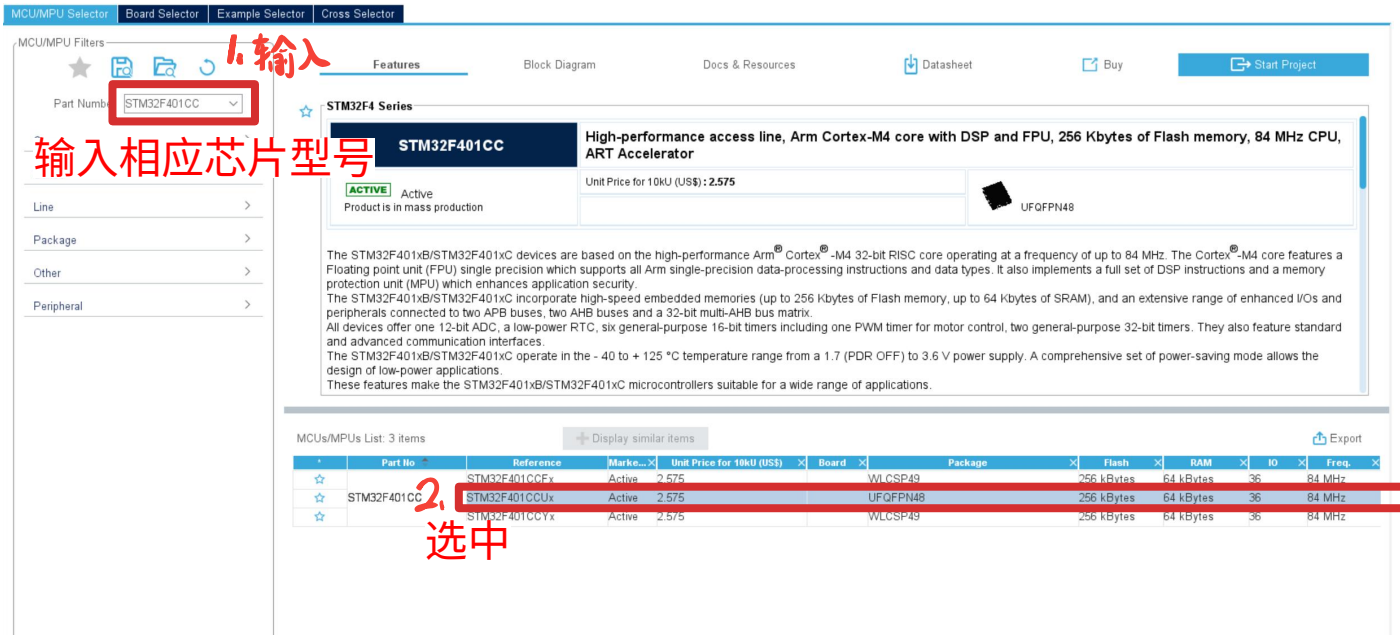
引脚图与CubeMX配置

首先，点击stm32CubeMX图标

接着，将鼠标移动至File上并点击New Project创建新项目



在红色框当中输入自己的stm32芯片型号，然后在右侧双击自己的芯片信号创建cubemx工程



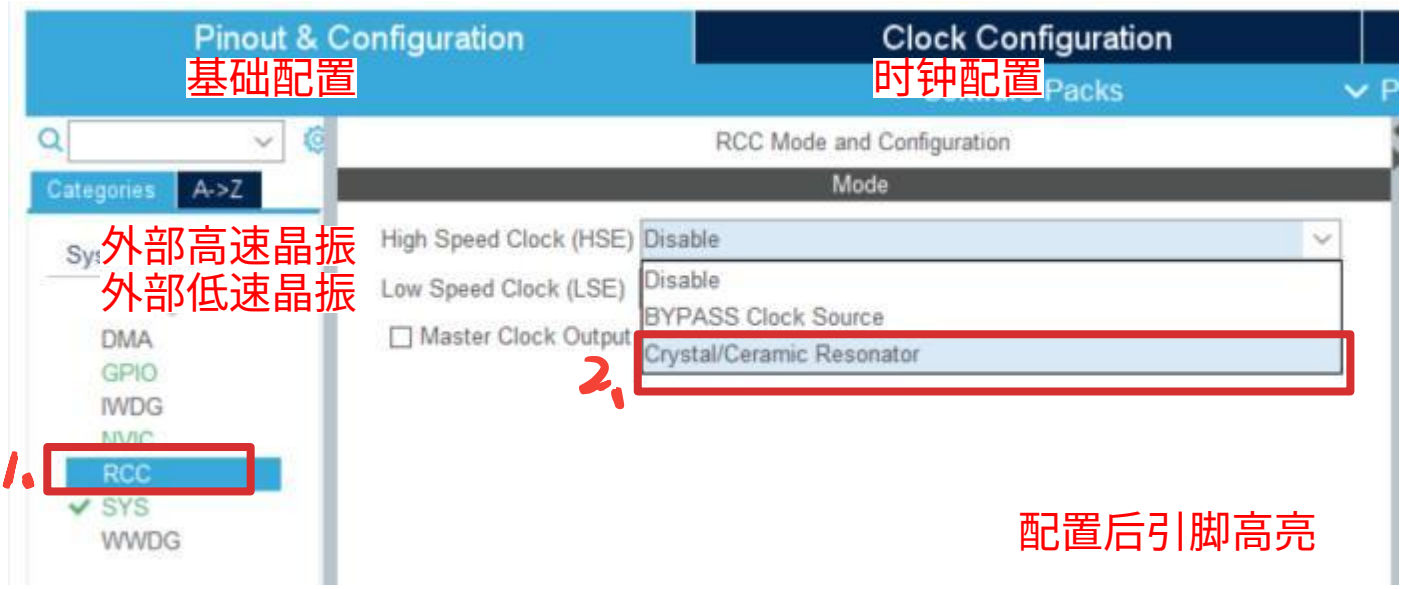
进入界面，看到最上面由四个大板块组成：

```
Pinout & Configuration：对引脚进行功能选择以及参数配置

Clock Configuartion：对时钟树进行配置

Project Manager：项目管理

Tools：对工具的管理
```

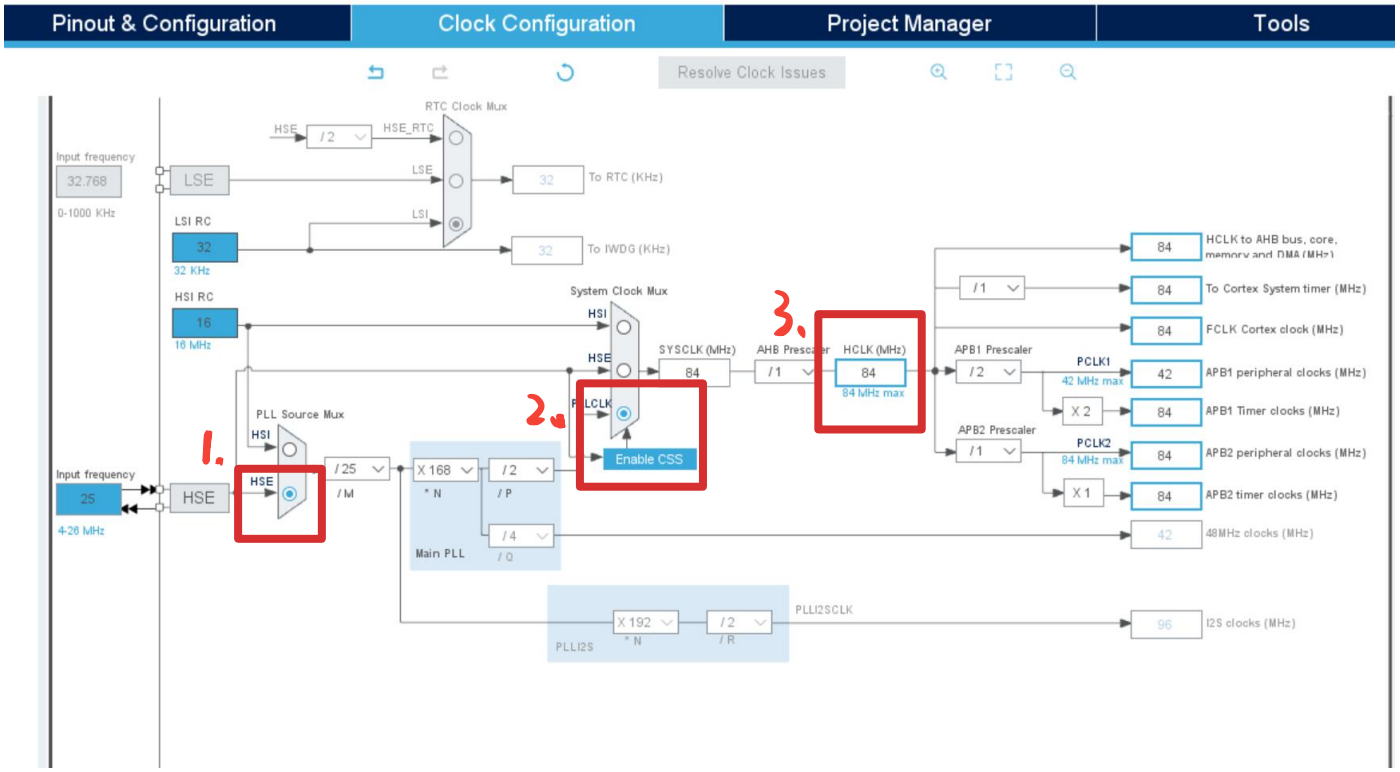
好，我们现在先进行工程项目最基本的配置：

1.配置烧录调试引脚：我们先点击System Core,再点击进SYS当中，在Debug选项中选择Serial Wire选项

2.配置时钟引脚：点击RCC，选择High Speed Clock(HSE)当中的Crystal/Ceramic Resontor



3.配置时钟树：点击最上面Clock Configuration。然后点击HSE和PLLCLK选项，将HCLK修改为84（即下方最大时钟频率）



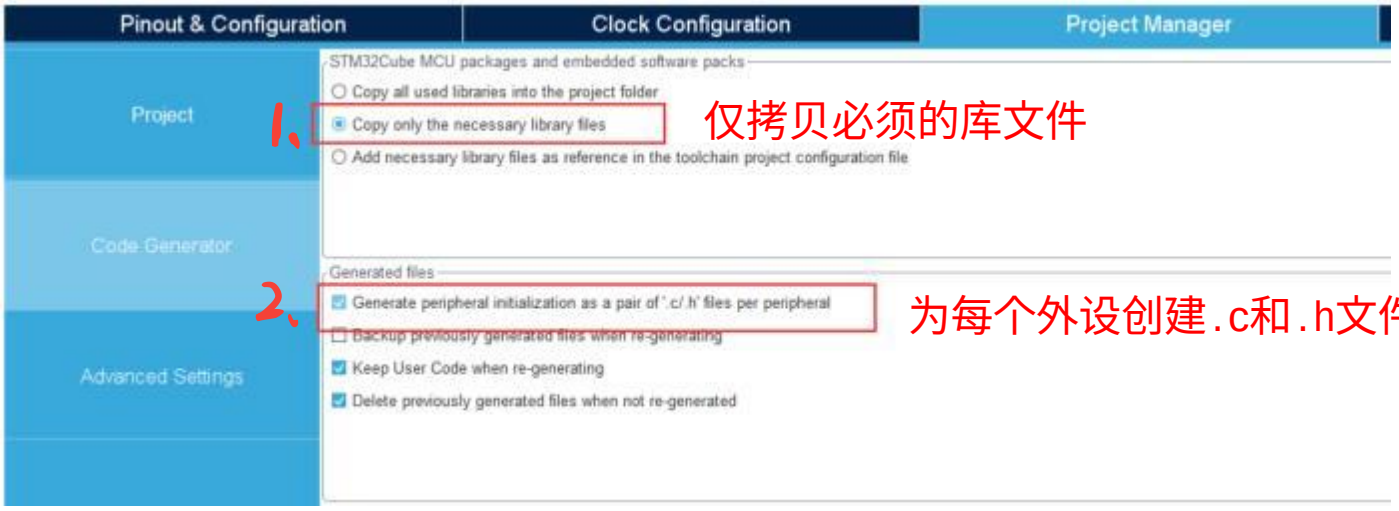
3.对项目进行配置：点击Project Manager，

Project Name当中输入自己的项目名称（由于我这个是测试样例，所以写了demo作为项目名称）。

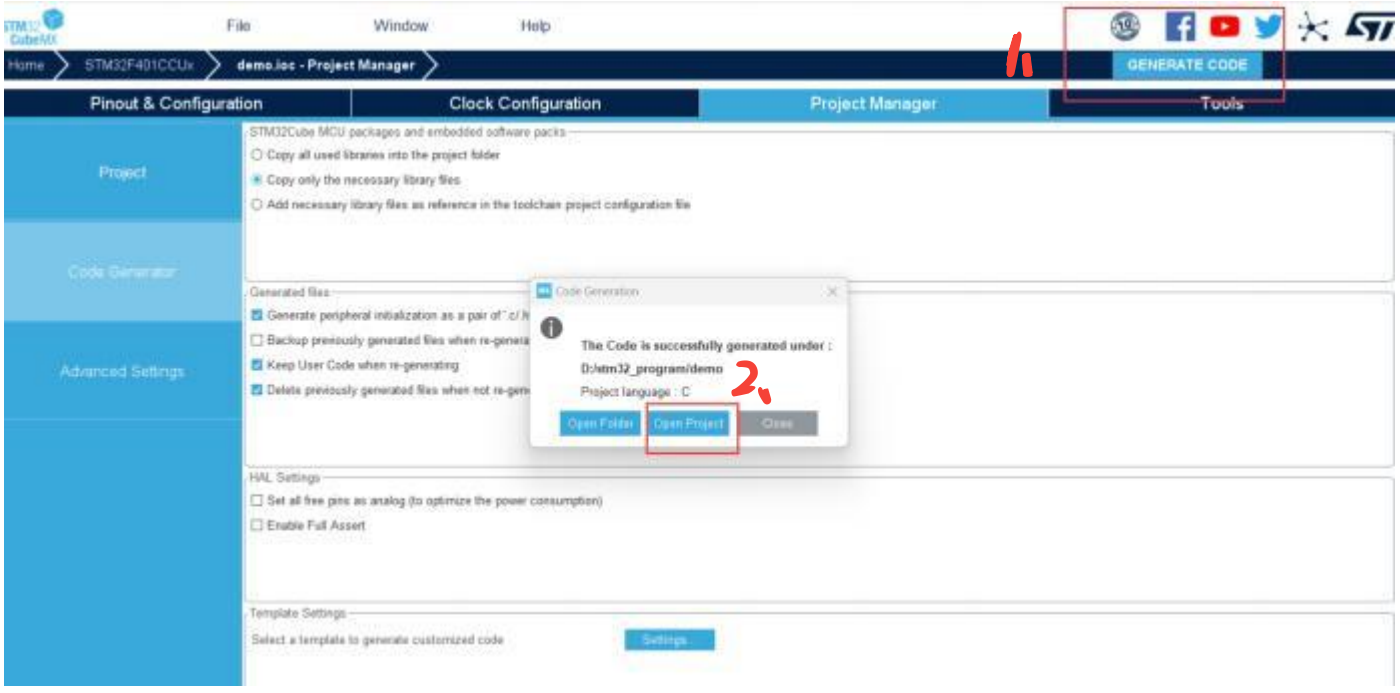
点击Project Location后面的Browse，进行项目的配置

Toolchain/IDE的下拉菜单，改为MDK-ARM

4.点击Code Generator，点击第一栏中的第二个选项（因为第一个选项会导致不必要的代码生成，加大存储的压力）；将第二栏中的第1、3、4选项打勾

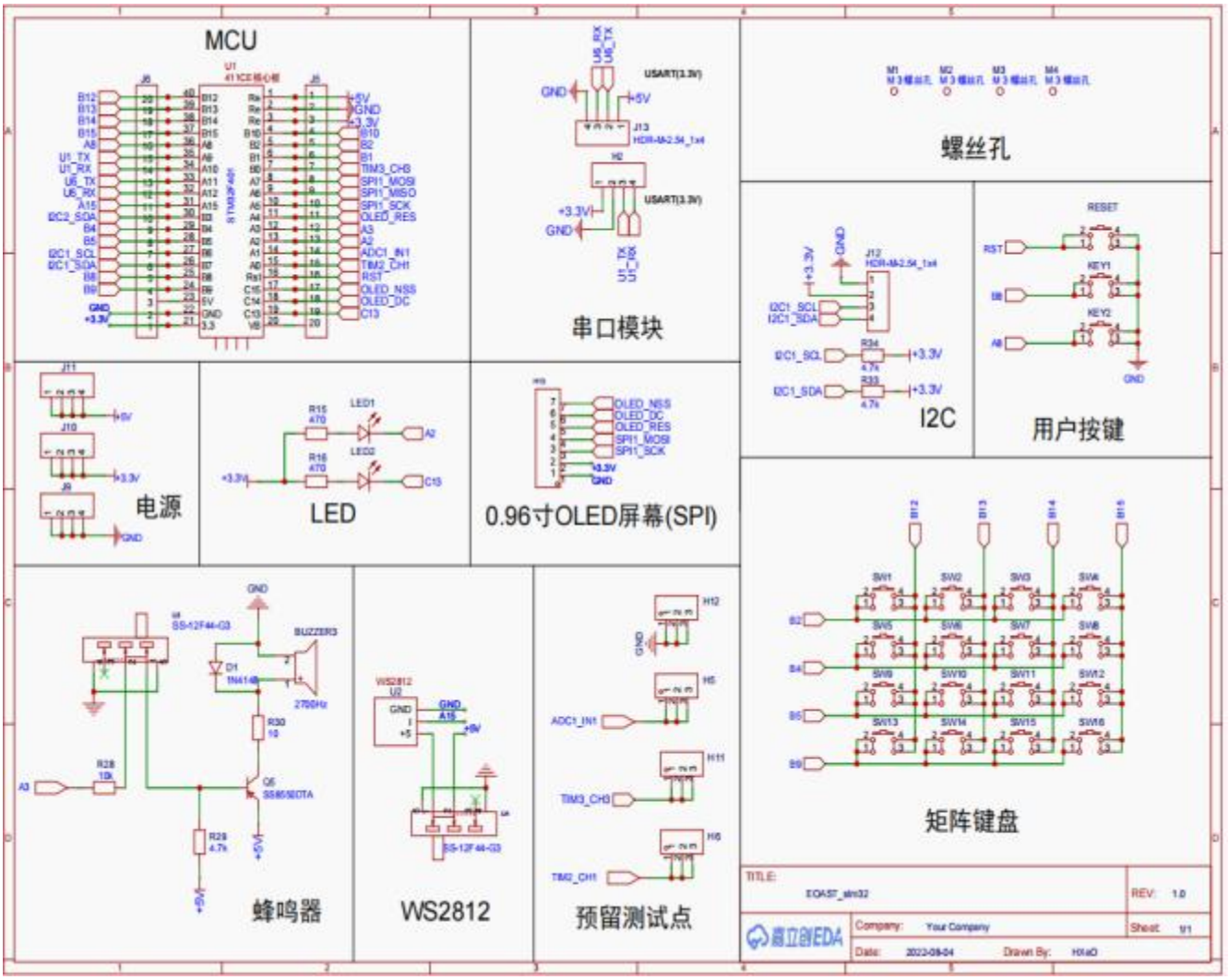


此时，这个项目算是基本配置完毕，可以点击右上角的GENERATE CODE来生成代码，然后会弹出一个小框，点击Open Project



下面，我们来学习如何通过观察引脚图配置Cubemx,我们以点亮学习板上面的LED1为例

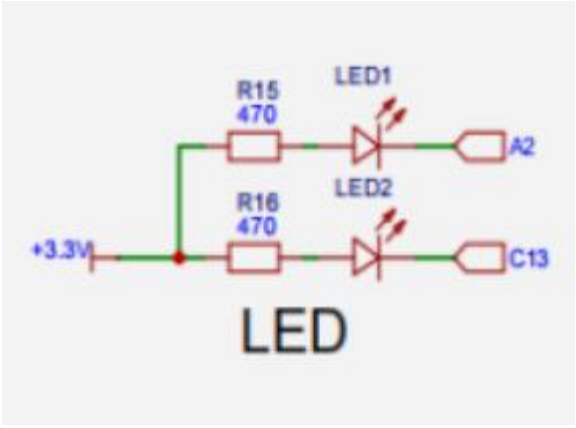
下图为开发板的原理图

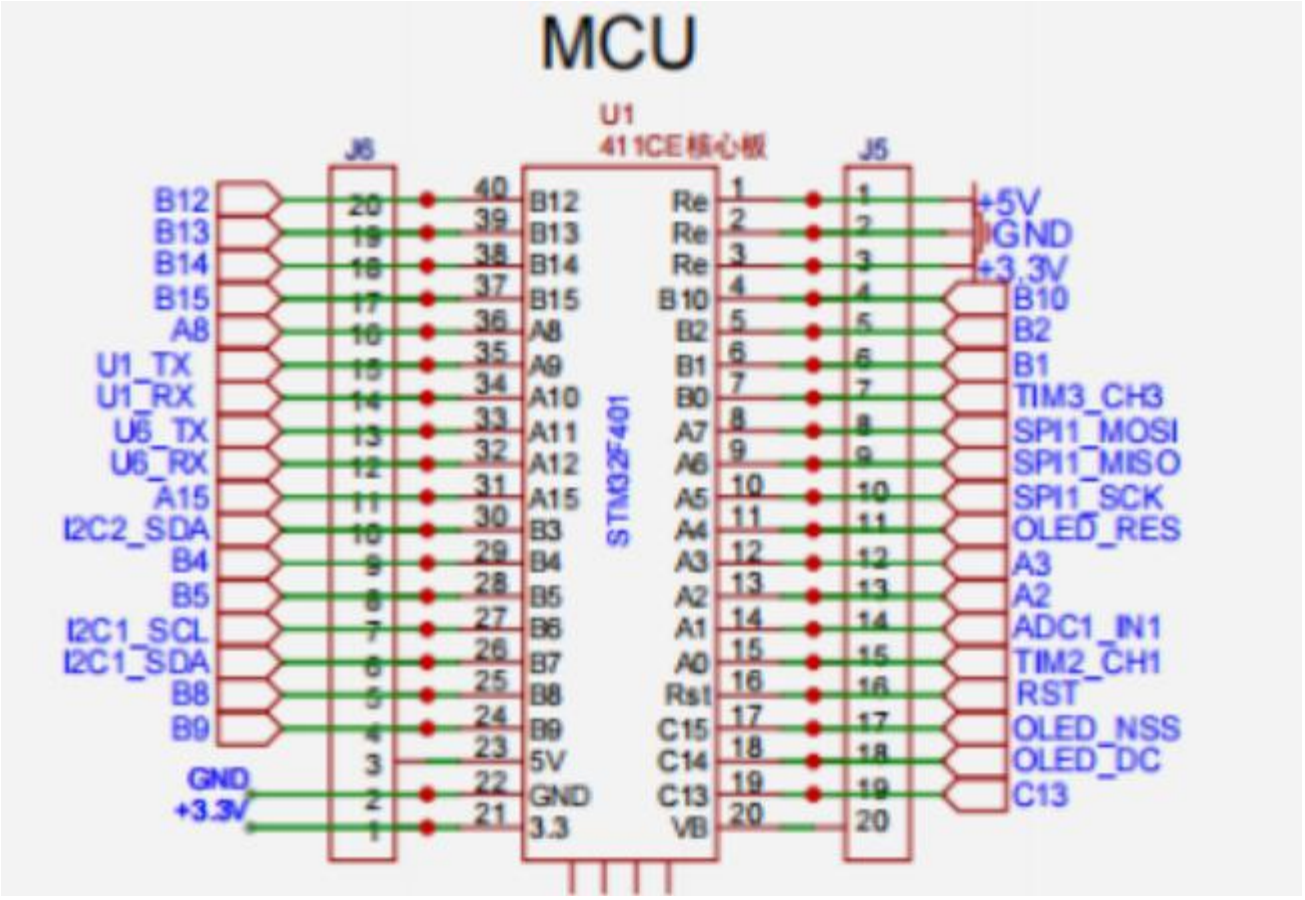


我们来来看LED的细节

我们现在观察到LED1连接到A2，这就说明，LED1的一个引脚连接在了的A2口上

， 我们现在通过MCU这部分观察到A2，连接的是单片机的PA2口上





小练习：能否告诉我串口模块的U1_RX对应单片机的接口是哪里

（答案：PA10）

GPIO

常见函数介绍

1.输出电平

```
HAL_GPIO_WritePin(GPIO_TypeDef* GPIOX,uint16_t GPIO_Pin,GPIO_PinState pinstate);
```

其中：

GPIOX代表目标引脚的端口号，例如GPIOB

GPIO_Pin代表目标引脚，例如GPIO_Pin_5

pinstate代表当前引脚高低电平，高电平即为GPIO_PIN_SET，低电平即为GPIO_PIN_RESET

在代码中，GPIO_PIN_SET又可以替换成1，GPIO_PIN_RESET可以替换成0

2.翻转电平

```
HAL_GPIO_TogglePin(GPIO_TypeDef* GPIOX,uint16_t GPIO_Pin);
```

3.输入电平（检测当前引脚电平）

```
HAL_GPIO_ReadPin(GPIO_TypeDef* GPIOX,uint16_t GPIO_Pin);
```

返回值是0或1，即低电平或者高电平

点灯

任务：点亮LED1

配置引脚：**左击左边芯片的PA2 引脚(对于引脚的功能选择，我们大多都是这么做 的)**，选择其中GPIO_Output

我们现在先来看一下GPIO配置选项，

GPIO output level：

默认的输出电平，比如选择low就是低电平输出，同理，如果我们选择high，就是高电平输出。回到此任务，通过阅读引脚图，我们得知，led为共阳极（如果不懂，可以查阅附录），我们为了将其初始化的时候为灭的状态，所以将outputlevel选为High

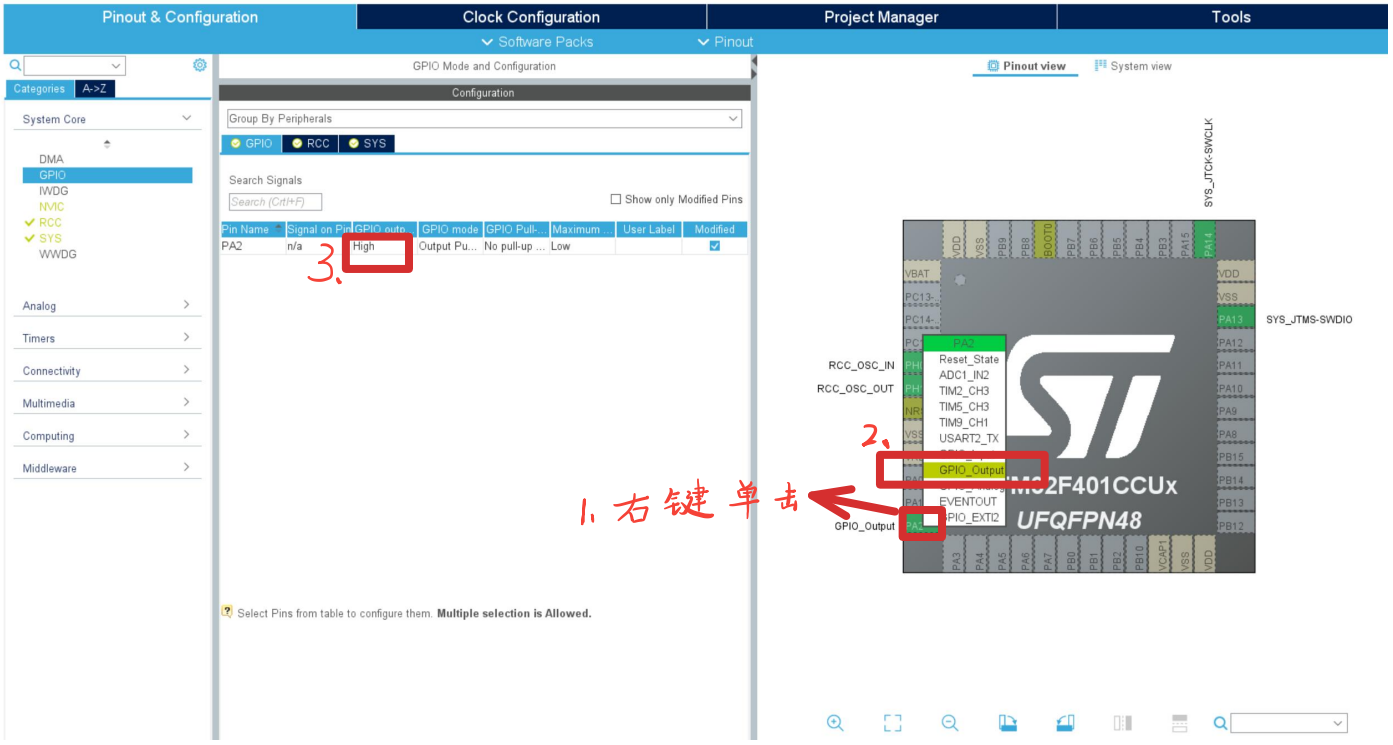
GPIO mode：Output Push Pull为推挽输出，Out Open Drain为开漏输出

GPIO Pull-up/Pull-down：这就是配置电阻的地方

当GPIO处于output模式的时候，我们一般选择no pull，引脚能够正确地输出目标值

当GPIO处于input模式的时候，需要根据默认的输入值来确定配置模式，如果默认输入的值为1的时候，最好配置为pull up，否则，最好配置为pull down

ps：开漏如果不连接外部上拉电阻，则只能输出低电平



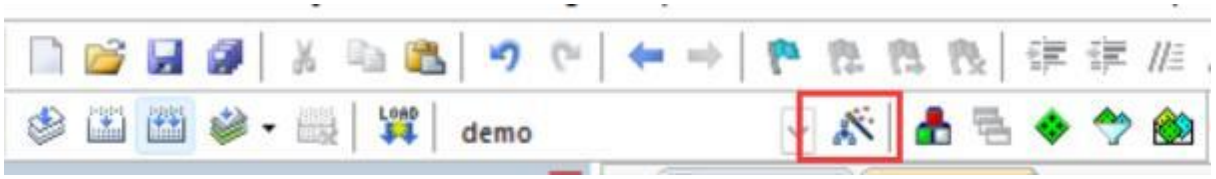
现在基本上小灯的程序已经配置完毕，我们产生一下代码，在while之前输入如下代码

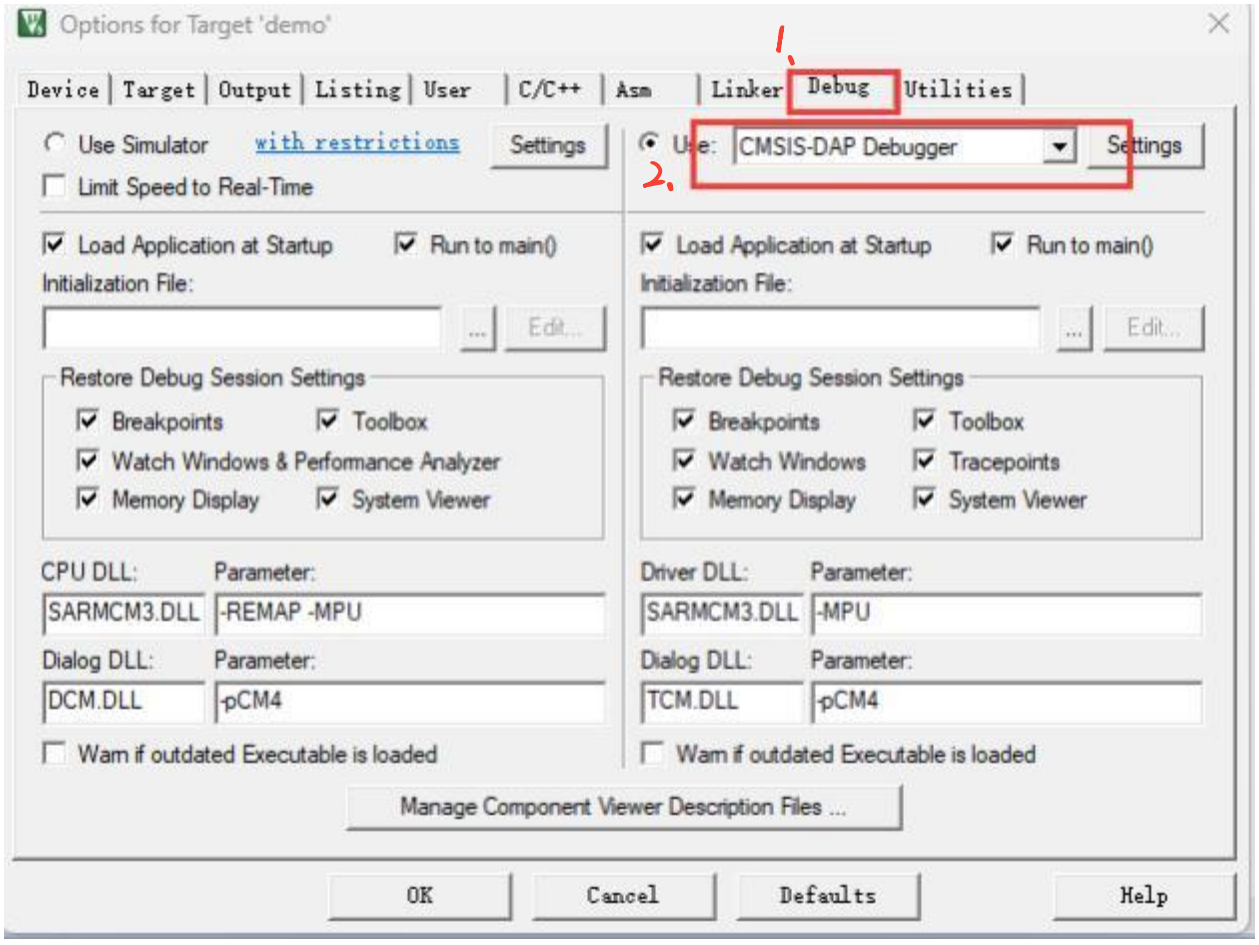
```
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  /* USER CODE BEGIN 2 */
HAL_GPIO_WritePin(GPIOA,GPIO_PIN_2,GPIO_PIN_RESET);
  /* USER CODE END 2 */

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

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

然后，我们需要对代码进行编译和烧录

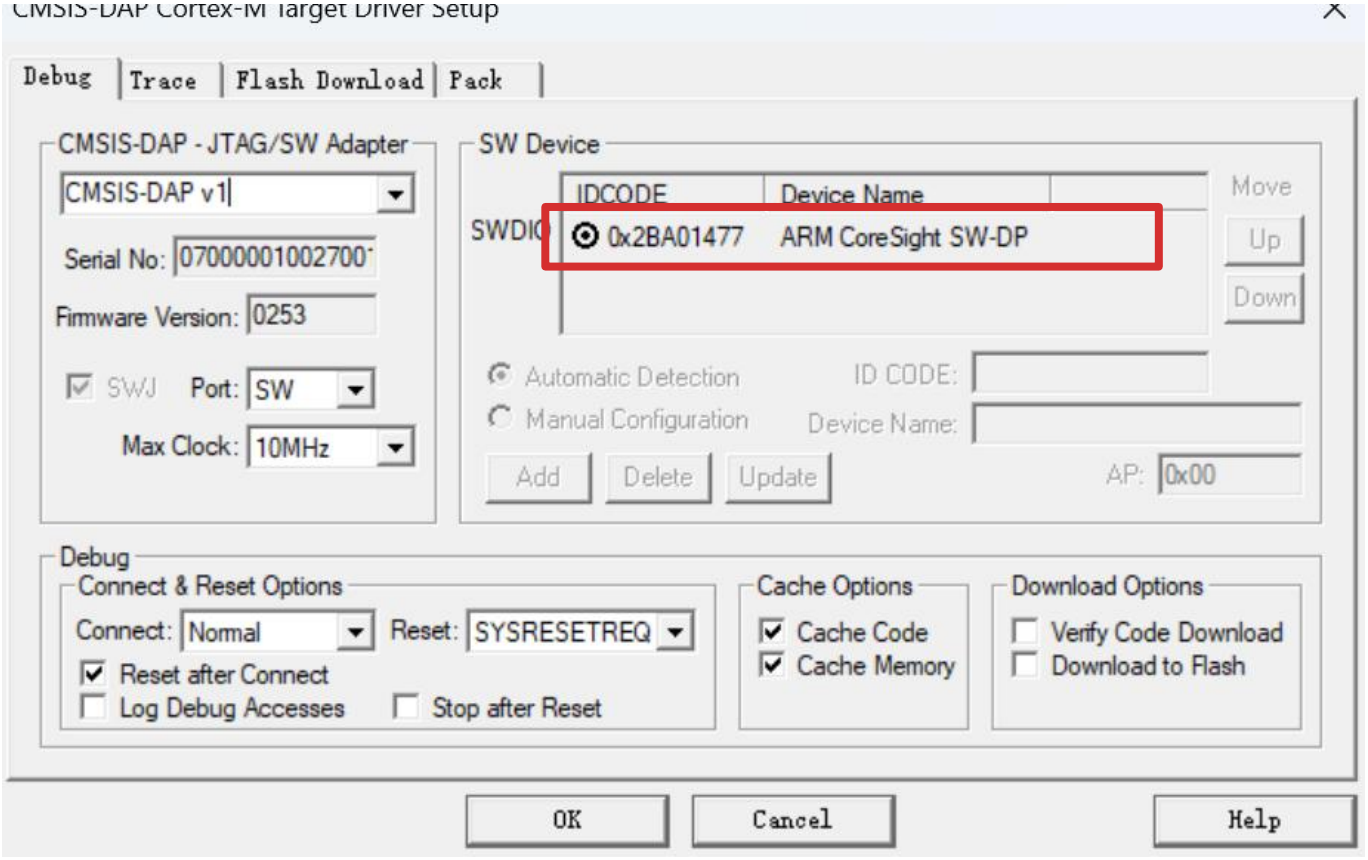
上图第二个为编译，最后一个为烧录，我们也习惯先按F7进行编译，再按F8进行烧录。

烧录前的准备：点开工具栏的魔法棒图标，并选择Debug。然后，选择自己的烧录器类型，我们在做测试的使用用的是DAP-Link，所以可以点击下拉菜单，选择其中的CMSIS-DAP Debugger选项



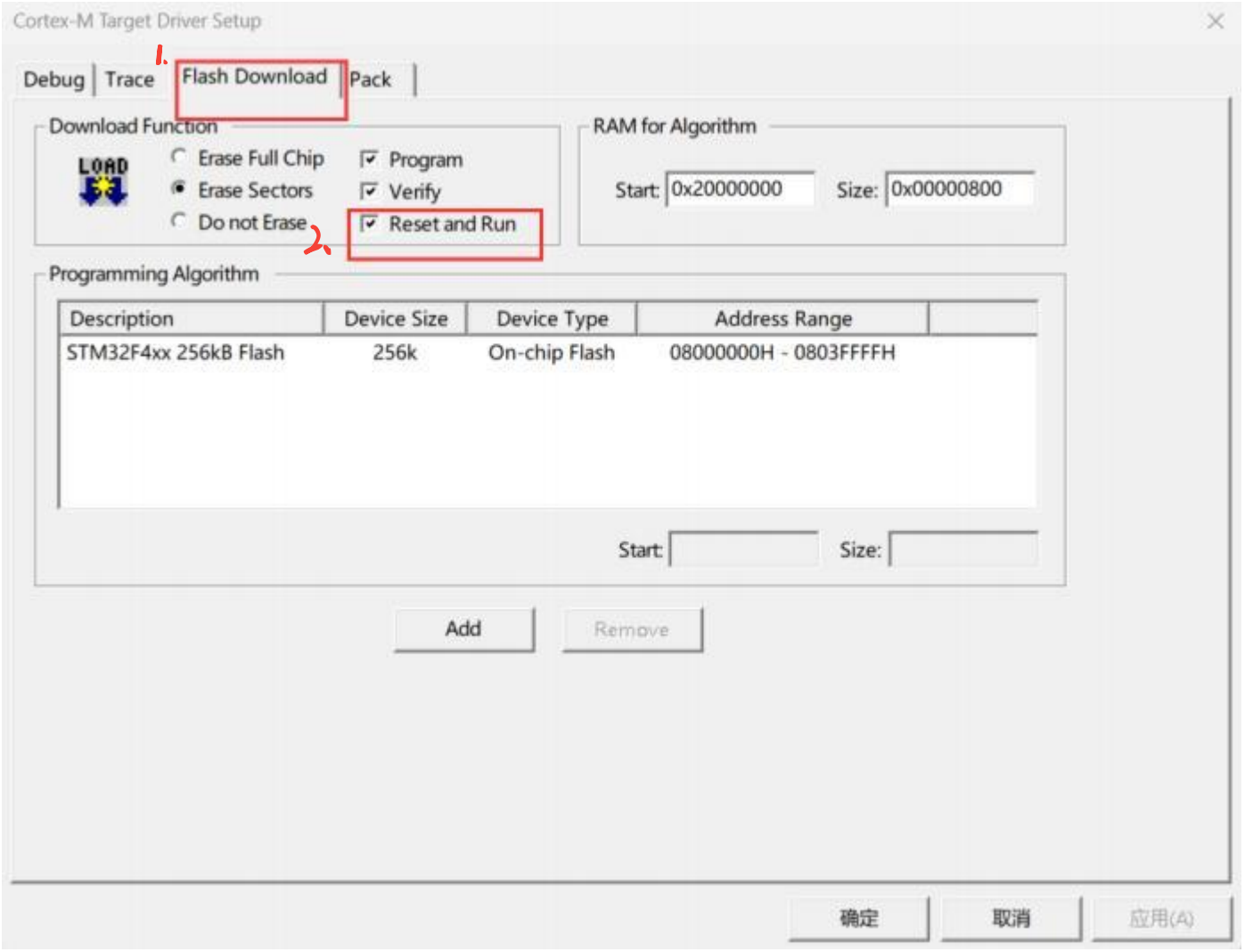


选好自己所用烧录器后，点开settings,看到如果你的界面像红框圈起来那样，那就说明，你能成功检测到烧录器（很多时候，对于烧录的debug也都是会检查这里的）

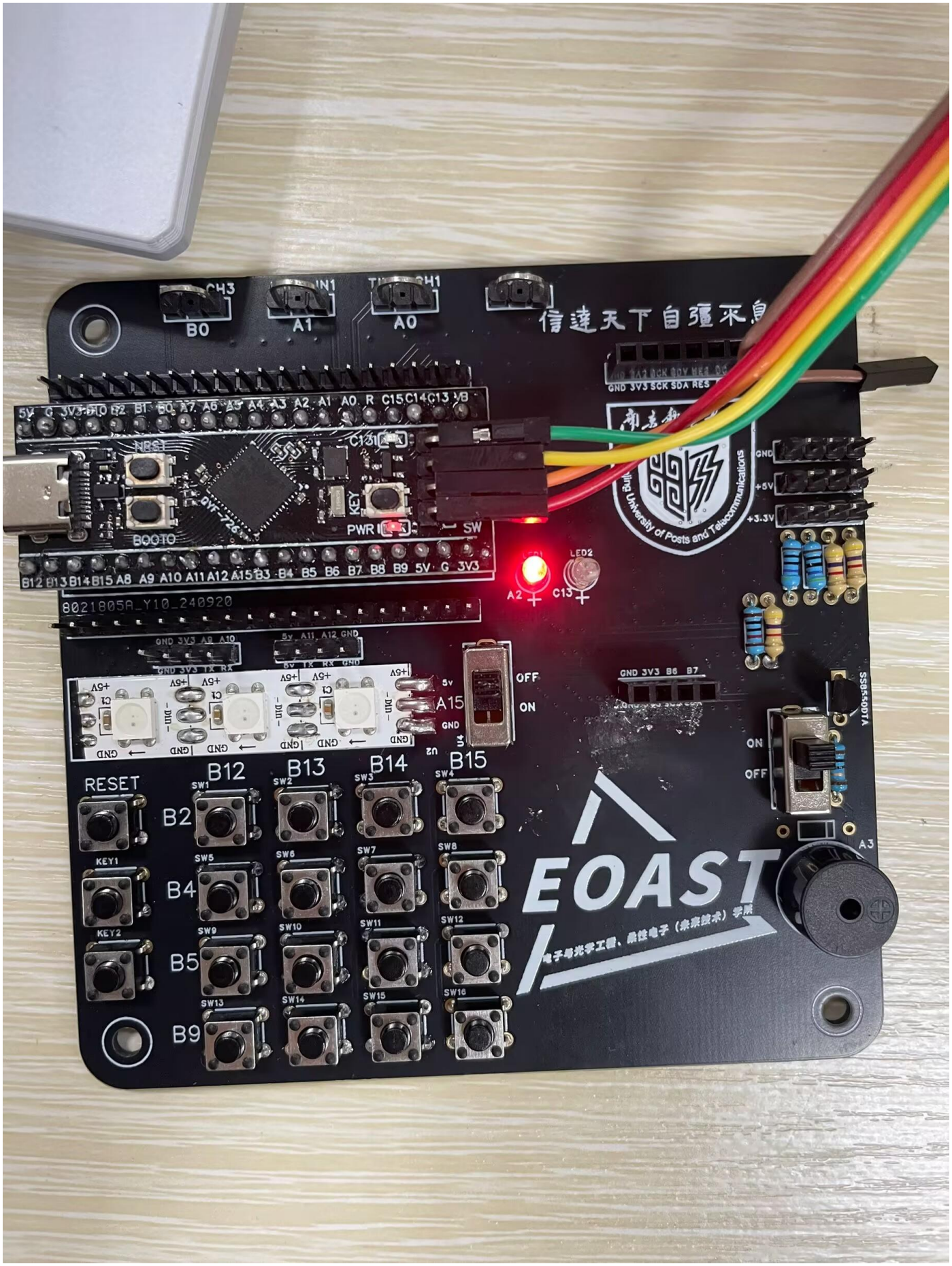


点击上面的Flash Download选项，再勾选其中的Reset and Run选项(这样烧录之后代码就会直接运行，如果不点击此选项，烧录之后必须要按一下单片机上的NRST才能开始

运行代码)，然后点击确定



烧录成功之后，可以看到如下场景（如果要上没有亮，按一下NRST，要是还不行，就debug一下，或许是你代码出现了一些小问题）


B0
A1
A0
RESET
BOOT0
PWR
KEY
LED1
LED2
A2
C13
OFF
ON
A15
B12
B13
B14
B15
B2
B4
B5
B9
KEY1
KEY2
SW1
SW2
SW3
SW4
SW5
SW6
SW7
SW8
SW9
SW10
SW11
SW12
SW13
SW14
SW15
SW16
EOAST
A3

小任务：将**LED2**以**1s**频率闪烁

提示：`HAL_Delay();`可以在括号中填入你想要的参数，他指定了**延时的时间**（单位是毫秒）。

ps：笔者这里留两个小问题

1.还有没有HAL库的其他方案可以延时的？这些方案之间有没有什么优劣？可以自行上互联网进行搜索（下文也会讲解另外一种方法）

2.这个HAL_Delay它的延时准确吗？为什么？

小任务答案：（答案不唯一）

```
while (1)
{
      HAL_GPIO_TogglePin(GPIOA,GPIO_PIN_8);
      HAL_Delay(500);
      HAL_GPIO_TogglePin(GPIOA,GPIO_PIN_8);
      HAL_Delay(500);
  /* USER CODE END WHILE */
  /* USER CODE BEGIN 3 *
```

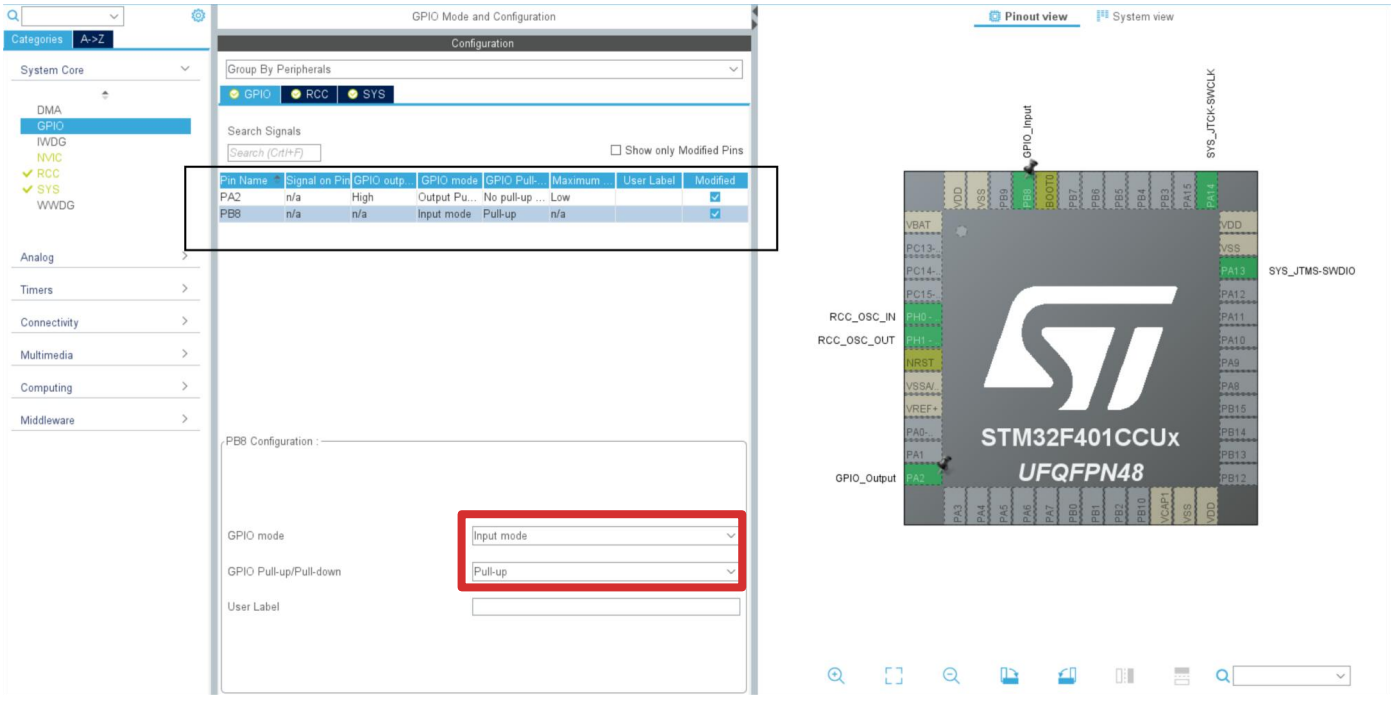
按键

单个按键（按键消抖）

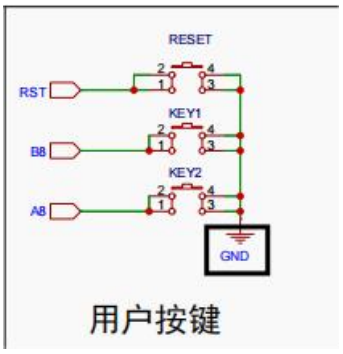
任务：做一个小作品，当我们按下按键的时候，LED的电平翻转（此讲义中，我们使用LED1+KEY1）

第一步：cubemx配置

GPIO引脚配置如下：led配置沿用之前的配置，按键的配置选择为输入模式，并且引脚上拉。（这里选择上拉是因为原理图画的时候按键接地了）



Pin Name	Signal on Pin	GPIO outp...	GPIO mode	GPIO Pull-...	Maximum ...	User Label	Modified
PA2	n/a	High	Output Pu...	No pull-up ...	Low		☑
PB8	n/a	n/a	Input mode	Pull-up	n/a		☑



第二部：keil代码编写，这个任务当中，我们只需要对while里面的代码进行修改（想知道为什么在while里面修改，具体main.c的代码框架请查看附录）

```
while (1)
{
                if(HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_8) == 0)
                        HAL_GPIO_TogglePin(GPIOA,GPIO_PIN_2);
  /* USER CODE END WHILE */

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

我们烧录好代码，进行测试的时候，会发现按键有时候起作用，有时候不起作用，这是因为按键的构造原因（具体可以查看附录），我们在这里可以对其进行**按键消抖**进行这种误差的消除

按键消抖

方法一：在这里，为了防止以后出现不必要的问题，我们采用 `HAL_GetTick()`函数进行延时

PS:`HAL_GetTick()`可以获取系统当前时间

这里我们在main函数外部定义全局变量tick，其中uint32_t意思为32位无符号整型

```
uint32_t tick;
int main(){
  .......
  while (1)
  {
        if(HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_8) == 0)
        {
                if(HAL_GetTick()-tick > 10)
                {
                        HAL_GPIO_TogglePin(GPIOA,GPIO_PIN_2);
                }
                tick = HAL_GetTick();
        }
  }
```

代码逻辑：我们首先判断是否读取低电平，因为按键由抖动情况，我们就认为，抖动的时候我们是没法读取到低电平的，或者说此时电平不是很稳定。这时候，if语句就无法被再次进入，直至抖动结束。然后翻转LED电平，并将tick更新。因为此后一直被读取到低电平，所以tick的值一直被更新，也就无法再次翻转LED电平。这也达到了按下一次翻转一次电平。

方法二：

```
uint32_t tick;
int main(){
  .......
  while (1)
  {
        if(HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_8) == 0)
        {
                tick = HAL_GetTick();
                while(HAL_GetTick()-tick <= 10);
                if(HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_8) == 0)
                {
                        HAL_GPIO_TogglePin(GPIOA,GPIO_PIN_2);
```

```
                }
            }
        }
    }
```

代码逻辑：我们先读取低电平。然后用HAL_GetTick()函数进行10ms的延时，之后再判断读取的是否位低电平，再决定是否翻转电平。

ps:方法二因为整个执行周期就只有10ms，加上其在while中，所以基本上你按了多少10ms也就执行了多少次。但是你也可以加上一些flag位，达到方法一的效果

综上，笔者在这里比较推荐方法一，虽然代码逻辑上可能稍微理解起来比方法二困难一些，但是，从执行的稳定度上，肯定是方法一要更优化一些

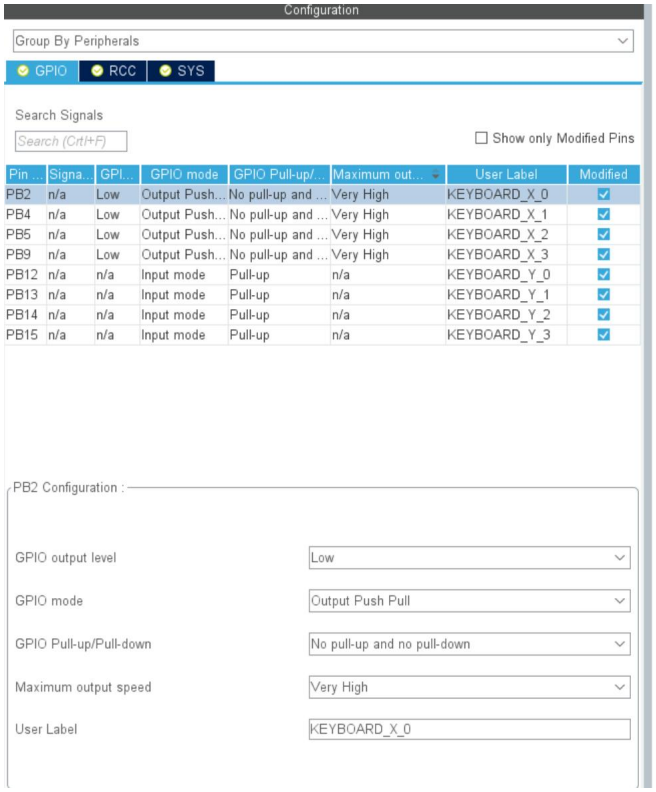
矩阵键盘

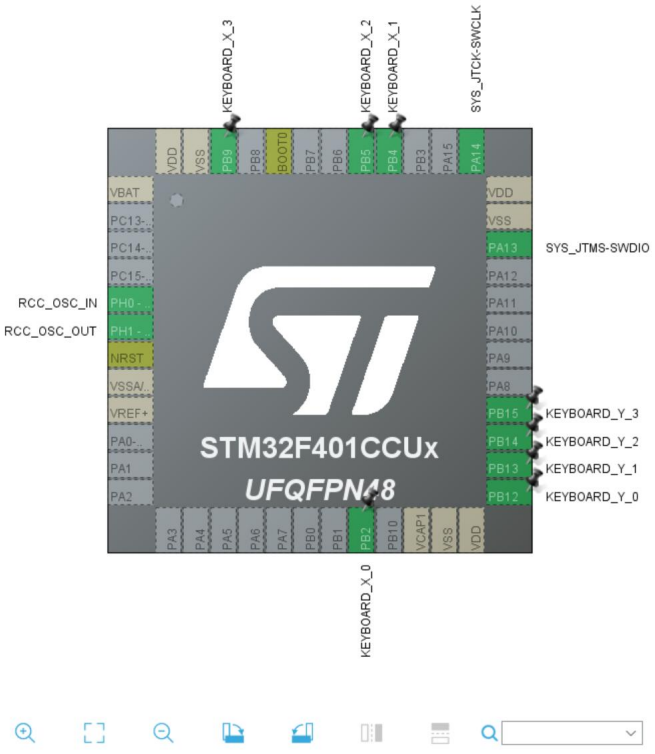
做一个小的矩阵键盘，可以使得两个LED灯以不同形式，不同速率点亮（矩阵键盘原理可以查看附录）

进一步说明：

按键	LED1	LED2	LED状态
SW1	亮	-	LED1以0.5s速率闪烁
SW2	-	亮	LED2以0.5s速率闪烁
SW3	亮	亮	LED1，2以0.5s速率闪烁
SW5	亮	-	LED1以1s速率闪烁
SW6	-	亮	LED2以1s速率闪烁
SW7	亮	亮	LED1，2以1s速率闪烁
SW9	亮	-	LED1常亮
SW10	-	亮	LED2常亮
SW11	灭	灭	LED1，2都灭

步骤一：cubemx配置





步骤二：keil5代码编写

```c
uint32_t key_scan();
/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
uint32_t tick,tick1;

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */
        int key_code;
  /* 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();
/* USER CODE BEGIN 2 */
/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
		key_code = key_scan();

		switch(key_code){
			case 1:
				HAL_GPIO_TogglePin(GPIOB,GPIO_PIN_8);
				tick1= HAL_GetTick();
				while(HAL_GetTick()-tick1 <= 500);
				HAL_GPIO_TogglePin(GPIOB,GPIO_PIN_8);
				tick1= HAL_GetTick();
				while(HAL_GetTick()-tick1 <= 500);
				break;
			case 2:
				HAL_GPIO_TogglePin(GPIOA,GPIO_PIN_8);
				tick1= HAL_GetTick();
				while(HAL_GetTick()-tick1 <= 500);
				HAL_GPIO_TogglePin(GPIOA,GPIO_PIN_8);
				tick1= HAL_GetTick();
				while(HAL_GetTick()-tick1 <= 500);
				break;
			case 3:
				HAL_GPIO_TogglePin(GPIOA,GPIO_PIN_8);
				HAL_GPIO_TogglePin(GPIOB,GPIO_PIN_8);
				tick1= HAL_GetTick();
				while(HAL_GetTick()-tick1 <= 500);
				HAL_GPIO_TogglePin(GPIOA,GPIO_PIN_8);
				HAL_GPIO_TogglePin(GPIOB,GPIO_PIN_8);
				tick1= HAL_GetTick();
				while(HAL_GetTick()-tick1 <= 500);
				break;
			case 4:
				HAL_GPIO_TogglePin(GPIOB,GPIO_PIN_8);
				tick1= HAL_GetTick();
				while(HAL_GetTick()-tick1 <= 1000);
				HAL_GPIO_TogglePin(GPIOB,GPIO_PIN_8);
				tick1= HAL_GetTick();
				while(HAL_GetTick()-tick1 <= 1000);
				break;
			case 6:
				HAL_GPIO_TogglePin(GPIOA,GPIO_PIN_8);
				tick1= HAL_GetTick();
				while(HAL_GetTick()-tick1 <= 1000);
				HAL_GPIO_TogglePin(GPIOA,GPIO_PIN_8);
				tick1= HAL_GetTick();
				while(HAL_GetTick()-tick1 <= 1000);
				break;
			case 7:
				HAL_GPIO_TogglePin(GPIOA,GPIO_PIN_8);
```

```
                                HAL_GPIO_TogglePin(GPIOB,GPIO_PIN_8);
                                tick1= HAL_GetTick();
                                while(HAL_GetTick()-tick1 <= 1000);
                                HAL_GPIO_TogglePin(GPIOA,GPIO_PIN_8);
                                HAL_GPIO_TogglePin(GPIOB,GPIO_PIN_8);
                                tick1= HAL_GetTick();
                                while(HAL_GetTick()-tick1 <= 1000);
                                break;
                        case 9:
                                HAL_GPIO_WritePin(GPIOB,GPIO_PIN_8,0);
                                break;
                        case 10:
                                HAL_GPIO_WritePin(GPIOA,GPIO_PIN_8,0);
                                break;
                        case 11:
                                HAL_GPIO_WritePin(GPIOB,GPIO_PIN_8,1);
                                HAL_GPIO_WritePin(GPIOA,GPIO_PIN_8,1);
                                break;
                        default:
                                break;
                }
    /* USER CODE END WHILE */
    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE2);

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 25;
  RCC_OscInitStruct.PLL.PLLN = 168;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 4;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
```

```
  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
  {
    Error_Handler();
  }
}

/* USER CODE BEGIN 4 */
uint32_t key_scan()
{
        int key_code;
                        /*====对第一行进行检测=====*/
                HAL_GPIO_WritePin(GPIOB,GPIO_PIN_2,0);
                HAL_GPIO_WritePin(GPIOB,GPIO_PIN_5,1);
                HAL_GPIO_WritePin(GPIOB,GPIO_PIN_4,1);

                if(HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_12) == 0)
                {
                        if(HAL_GetTick()-tick > 10)
                        {
                                key_code = 1;
                        }
                        tick = HAL_GetTick();
                }
                if(HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_13) == 0)
                {
                        if(HAL_GetTick()-tick > 10)
                        {
                                key_code = 2;
                        }
                        tick = HAL_GetTick();
                }
                if(HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_14) == 0)
                {
                        if(HAL_GetTick()-tick > 10)
                        {
                                key_code = 3;
                        }
                        tick = HAL_GetTick();
                }

                /*====对第二行进行检测=====*/
                HAL_GPIO_WritePin(GPIOB,GPIO_PIN_2,1);
                HAL_GPIO_WritePin(GPIOB,GPIO_PIN_5,1);
                HAL_GPIO_WritePin(GPIOB,GPIO_PIN_4,0);

                if(HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_12) == 0)
                {
                        if(HAL_GetTick()-tick > 10)
```

```
                {
                        key_code = 5;
                }
                tick = HAL_GetTick();
        }
        if(HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_13) == 0)
        {
                if(HAL_GetTick()-tick > 10)
                {
                        key_code = 6;
                }
                tick = HAL_GetTick();
        }
        if(HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_14) == 0)
        {
                if(HAL_GetTick()-tick > 10)
                {
                        key_code = 7;
                }
                tick = HAL_GetTick();
        }

        /*====对第三行进行检测=====*/
        HAL_GPIO_WritePin(GPIOB,GPIO_PIN_2,1);
        HAL_GPIO_WritePin(GPIOB,GPIO_PIN_5,0);
        HAL_GPIO_WritePin(GPIOB,GPIO_PIN_4,1);

        if(HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_12) == 0)
        {
                if(HAL_GetTick()-tick > 10)
                {
                        key_code = 9;
                }
                tick = HAL_GetTick();
        }
        if(HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_13) == 0)
        {
                if(HAL_GetTick()-tick > 10)
                {
                        key_code = 10;
                }
                tick = HAL_GetTick();
        }
        if(HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_14) == 0)
        {
                if(HAL_GetTick()-tick > 10)
                {
                        key_code = 11;
                }
                tick = HAL_GetTick();
        }
        return key_code;
}
```

PS:大家刚开始看这些代码有些复杂或者很长，但是，渐渐的会发现，一般一个程序的行数上3位数是很轻松的事情

但是我们也可以优化一部分对于按键扫描的代码（这部分我们不做讲解，日后大家会慢慢理解的），虽然说这些代码阅读起来复杂一些，但是，维护起来会轻松很多

```
typedef struct
{
        GPIO_TypeDef *GPIOx;
        uint16_t GPIO_Pin;
}GPIO_Pin_TypeDef;

GPIO_Pin_TypeDef keyboardRow[]={
        {GPIOB,GPIO_PIN_2},
        {GPIOB,GPIO_PIN_4},
        {GPIOB,GPIO_PIN_5}
};

GPIO_Pin_TypeDef keyboardCol[]=
{
        {GPIOB,GPIO_PIN_12},
        {GPIOB,GPIO_PIN_13},
        {GPIOB,GPIO_PIN_14}
};

uint32_t tick,tick1;
int key_code;

uint32_t key_scan()
{
        uint32_t result = 0;
        int row,col,tiick;
        for(row = 0;row < sizeof(keyboardRow)/sizeof(GPIO_Pin_TypeDef); row++)
        {

HAL_GPIO_WritePin(keyboardRow[row].GPIOx,keyboardRow[row].GPIO_Pin,GPIO_PIN_RESET);
                for(col = 0;col < sizeof(keyboardCol)/sizeof(GPIO_Pin_TypeDef);
col++)
                {

if(HAL_GPIO_ReadPin(keyboardCol[col].GPIOx,keyboardCol[col].GPIO_Pin) == 0)
                        {
                                if(HAL_GetTick()-tick>10)
                                {
                                        result = row *
sizeof(keyboardRow)/sizeof(GPIO_Pin_TypeDef) + col;
                                }
                                tick = HAL_GetTick();
                        }
                }

HAL_GPIO_WritePin(keyboardRow[row].GPIOx,keyboardRow[row].GPIO_Pin,GPIO_PIN_SET);
        }
```

```
        return result;
}
```

在这里，如果细心的小伙伴会发现有个小bug，也就是对于前两行，如果我短按，我必须每次必须等待他灯闪烁一次结束，而且这个时刻我恰巧按下了按键，才能识别我目前按下的按键。这时候，可以尝试使用外部中断来进行按键扫描，这就留给课后作业吧（后面上课也会涉及到中断这个概念，到那时候再回头修复这个小bug也来得及）

小任务

对键盘上的所有按键进行编程，使其每个按键完成不同的任务。

附录

GPIO原理小简介

GPIO全名为General Purpose Input Output，即通用输入输出。有时候简称为“IO口”。通用，说明它是常见的。输入输出，就是说既能当输入口使用，又能当输出口使用。端口，就是元器件上的一个引脚。

模式汇总

输入模式

浮空输入（GPIO_Mode_IN_FLOATING）： 引脚电平是**真实的**外部连接器件**电压**，电平有不确定性

上拉输入（GPIO_Mode_IPU）: 默认通过电阻上拉到Vcc，不接外部器件时可以读出**高电平**

下拉输入（GPIO_Mode_AF_OD）: 默认通过电阻下拉到GND，不接外部器件时可以读出**低电压**

模拟输入（GPIO_Mode_AIN0）: 将外部信号直接传输到数模转换通道上

输出模式

开漏输出（GPIO_Mode_out_OD）: 只能输出**低电平**，高电平由电阻上拉决定

开漏复用功能（GPIO_Mode_AF_OD）: 用于外设功能使用

推挽式输出（GPIO_Mode_out_PP）: 可以输出强高和强低电压，多用于控制LED

推挽式复用功能（GPIO_Mode_AF_PP）: 用于外设功能使用

Keil main.c代码框架讲解

```
19  /* Includes ------------------------------------------------------------------*/
20  #include "main.h"
21  #include "gpio.h"
22
23  /* Private includes ----------------------------------------------------------*/
24  /* USER CODE BEGIN Includes */
25
26  /* USER CODE END Includes */
27
28  /* Private typedef -----------------------------------------------------------*/
29  /* USER CODE BEGIN PTD */
30
31  /* USER CODE END PTD */
32
33  /* Private define ------------------------------------------------------------*/
34  /* USER CODE BEGIN PD */
35  /* USER CODE END PD */
36
37  /* Private macro -------------------------------------------------------------*/
38  /* USER CODE BEGIN PM */
39
40  /* USER CODE END PM */
41
42  /* Private variables ---------------------------------------------------------*/
43
44  /* USER CODE BEGIN PV */
45
46  /* USER CODE END PV */
47
48  /* Private function prototypes -----------------------------------------------*/
49  void SystemClock_Config(void);
50  /* USER CODE BEGIN PFP */
51
52  /* USER CODE END PFP */
53
54  /* Private user code ---------------------------------------------------------*/
55  /* USER CODE BEGIN 0 */
56
57  /* USER CODE END 0 */
58
59  /**
60    * @brief  The application entry point.
61    * @retval int
62    */
```

如上图，该部分为main函数之前的部分，主要作用：头文件包含，全局变量的定义，函数声明与定义

代码最好写在BEGIN和END之间，因为每一次cubemx更新的时候，会检查这里面的代码，会将不在BEGIN和END之间的代码给删除

```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();
  /* 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 */
}
```

如上图为main函数内部，其中，while(1)属于循环语句，单片机会一直**重复执行**其中的语句

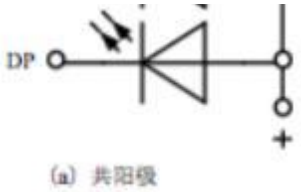
while上面的语句均为**引脚和时钟初始化语句**，我们可以在这里进行一些初始化和变量的定义

main函数下面有一堆代码，我们可以不去探究其含义，这些代码就是不动就好了

LED小知识

led分为共阳极和共阴极两种类型:

共阳极即为led一端接在高电平上，所以，我们如果想点亮共阳极led，就在其另一端输入低电平。



(a) 共阳极

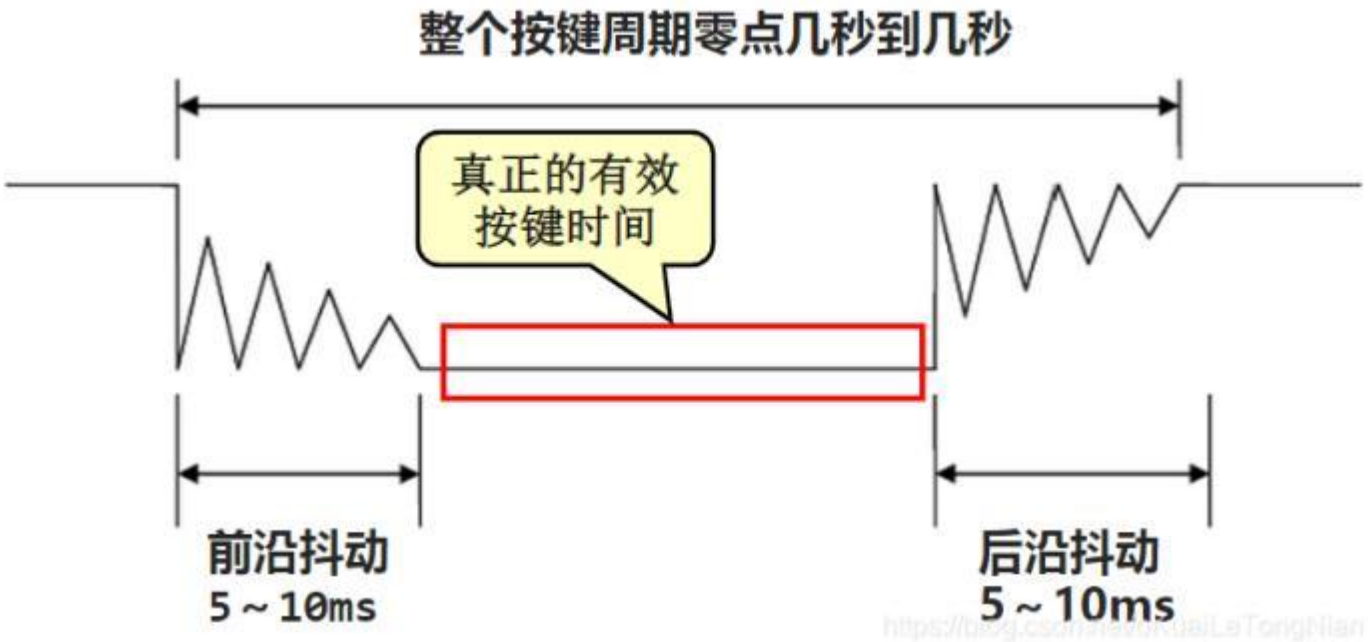
共阴极即为led一端接在低电平上，所以，如果我们想点亮共阴极led，就在另一端输入高电平。

(b) 共阴极

按键消抖原理

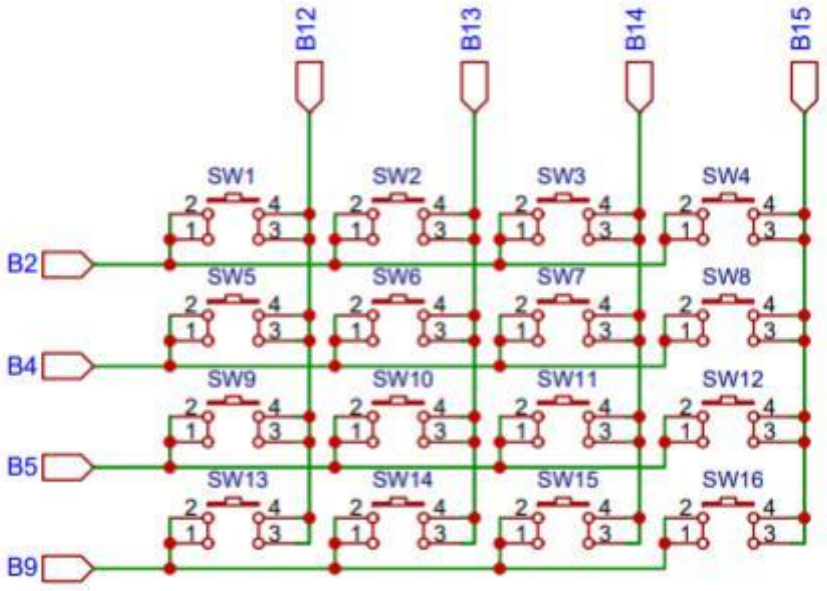
通常的按键所用开关为机械弹性开关，当机械触点断开、闭合的时候，由于机械触点的弹性作用，会导致信号的抖动，即一次按键动作会被当成多次按键，为了确保MCU对按键的一次闭合仅作一次处理，必须消除按键的抖动行为。

按键抖动的时长也由按键的机械特性决定，一般为5ms-10ms。



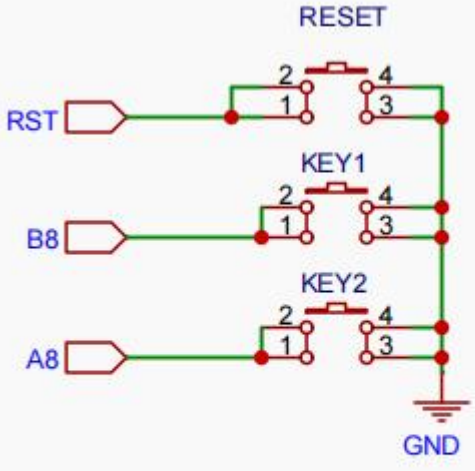
矩阵键盘原理

我们学完一个按键的处理之后，直接上手3×3的矩阵键盘是有一些小小的难度的，我们不妨先降低要求，我们先搞清楚1×3的按键是怎么被检测的，再开始去思考3×3的时候就会容易不少，



矩阵键盘

我们首先先看SW1-3这个1×3的按键，按键的本质就是开关，虽然说按键连接了两个端口，但是我们读取按键电平的时候，永远都是只读取其中一个引脚，那另外一个做什么作用呢?



用户按键

这时候，我们回到我们上文对一个按键的分析，在原理图当中，KEY1的左边引脚连接的是单片机，右边引脚接地。我们也可以认为，按键有一个引脚是要有一个确切的电平的，无论是高电平还是低电平。

回到我们对1×3的按键分析，也就是说，我们一个引脚可以提供一个高/低电平以便另一个引脚判断其是否被按下。

代码实现逻辑：我们可以让PB2输出低电平，对PB12-14读取是否为低电平从而判断按键是否被按下。现在我们可以去尝试挑战一下3×3的矩阵键盘编写了。但是这里有个小问题，因为我可以认为是，行作为输出，列作为输入。我一次只能检测3个按键，这时候我只能用**轮询**的方式来进行按键的检测，即我先把PB2拉低，PB4，PB5拉高，然后检测SW1-3，然后把PB4拉低，PB2,PB5拉高，检测SW5-7.....