STM32 MCU Development

STM32 单片机开发

-- STM32 流水灯开发

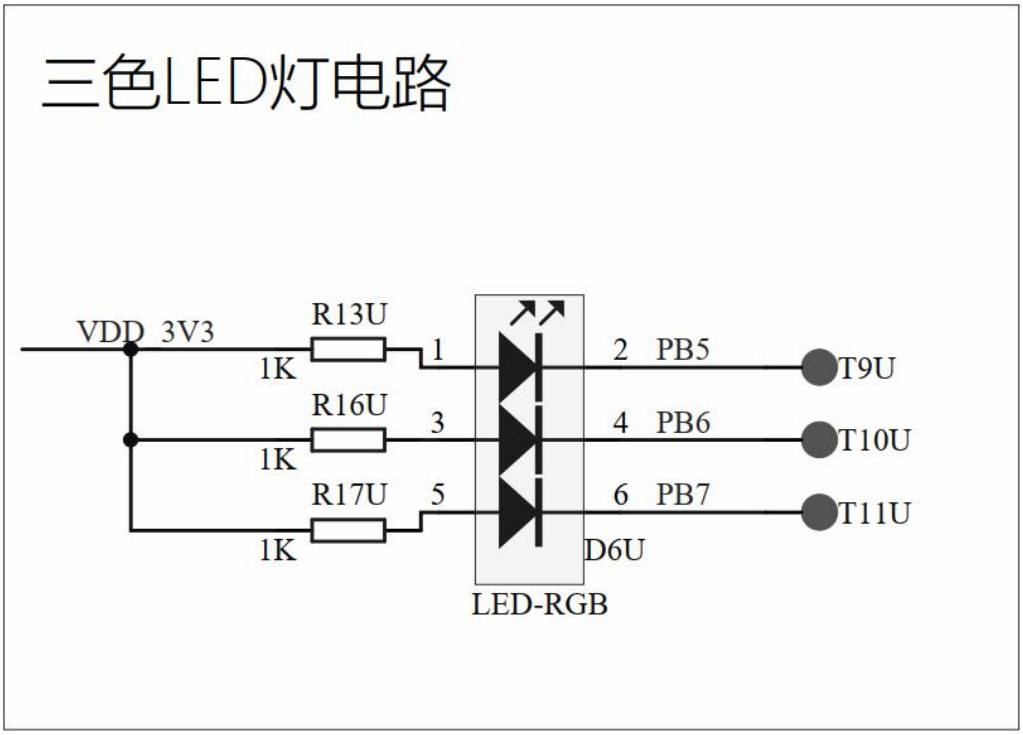
目录

1. STM32 流水灯开发

1.1. LED 灯工作原理分析

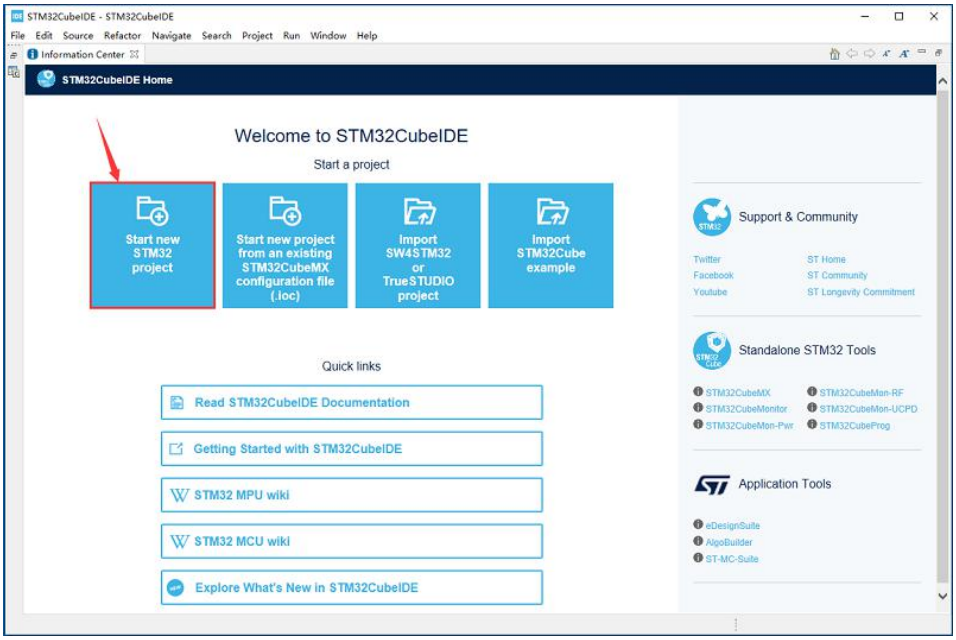
在 BearKE1 开发板上有一个 RGB 三色灯，他们分别连接到了 STM32 CPU 的 PB6(R)、PB7(G)、PB5(B)这三个管脚上，从下面的原理图上我们可以看出： CPU 的三个管脚如果输出低电平，那 Led 灯就亮了；而如果输出高电平，则三个 Led 就灭了。

接下来我们的主要工作就是通过编程控制 CPU，让这个三个管脚输出高低电平从而控制 LED 灯的亮灭。

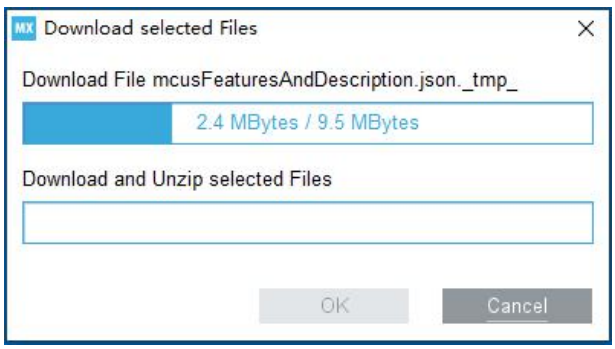


1.2. 创建 Led 流水灯项目

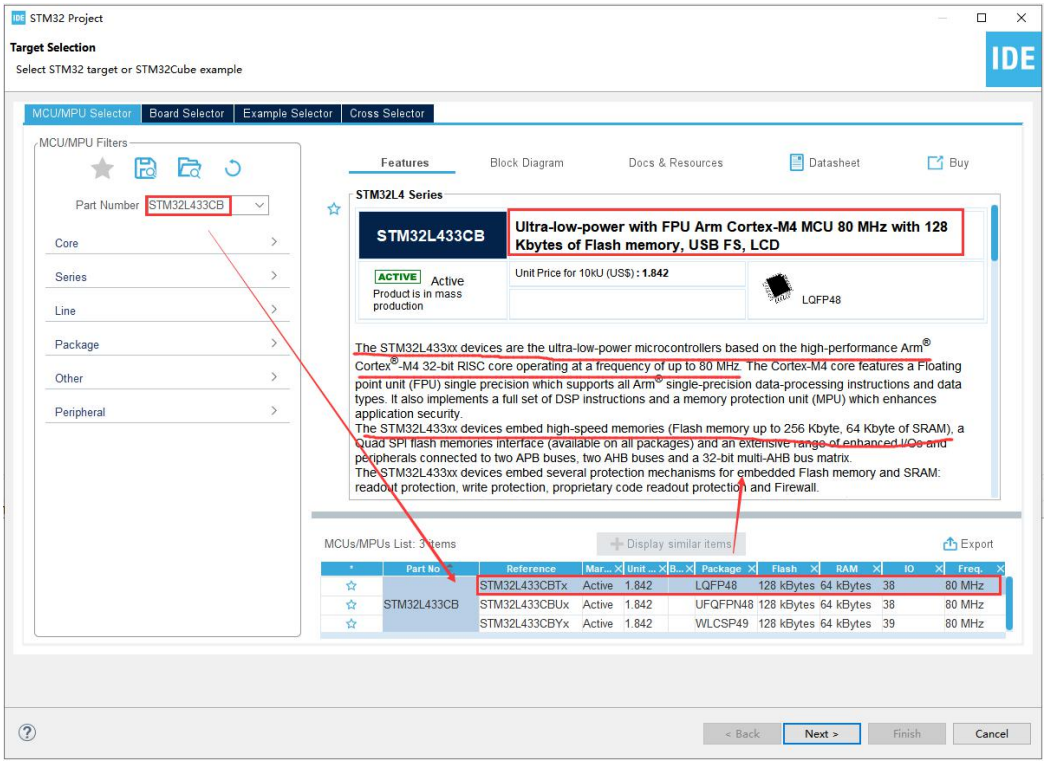
1. 使用 STM32CubeIDE 开始创建一个新的项目



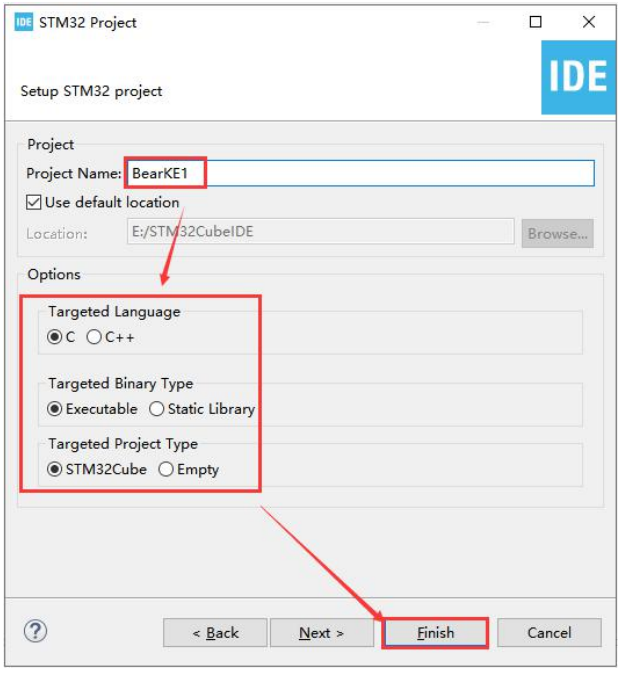
2. 如果首次使用或长久没使用则会提示软件更新，点击确认下载更新：



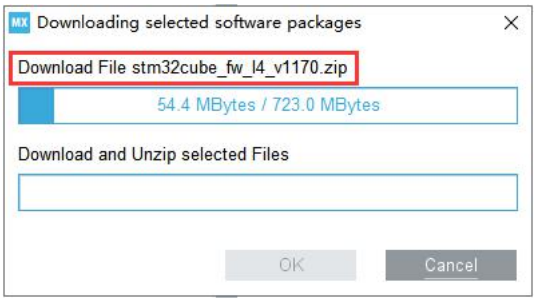
3. 更新完成后，在 Part Number 里输入开发板或产品所使用的 CPU 型号来查询相应 CPU，然后选中 CPU 的具体型号并点击 Next 开始创建项目。BearKE1 开发板使用的 CPU 型号为 STM32L433CBT6，这里我们输入 STM32L433CB 即可。



4. 设置项目名称并确认项目源码保存路径：

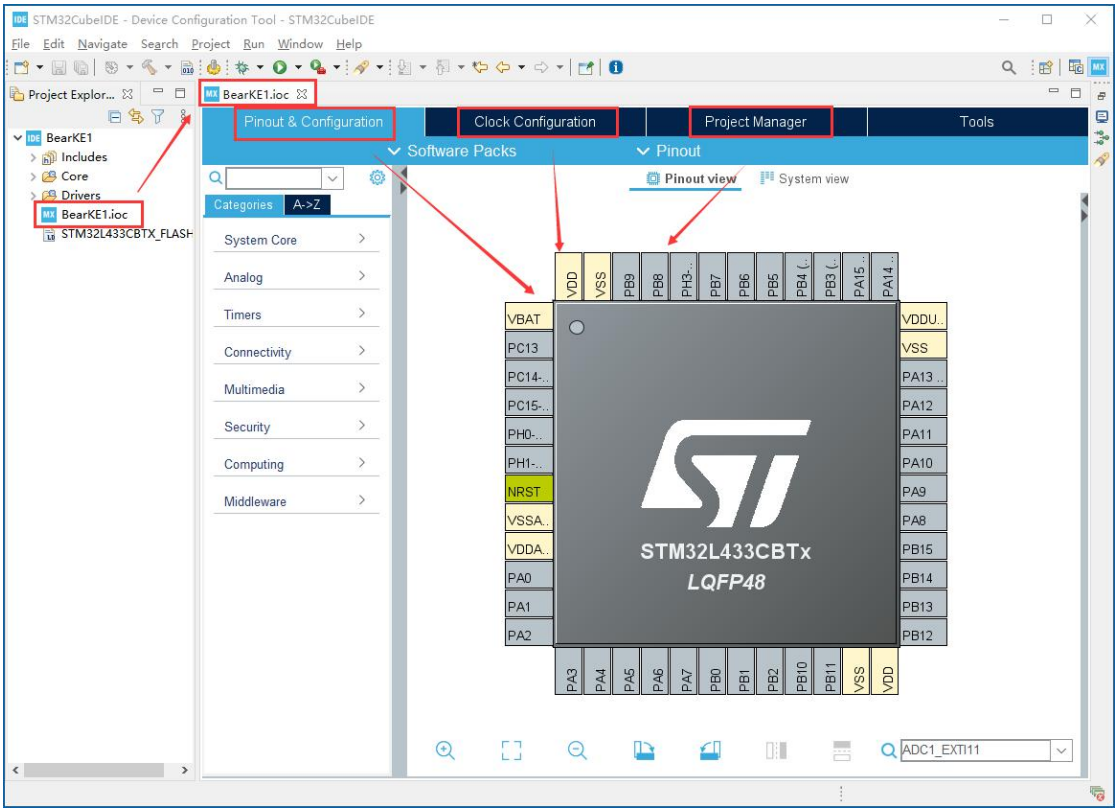


5. 如果首次使用或长时间未使用，则会自动更新 STM32 Hal 库源码：

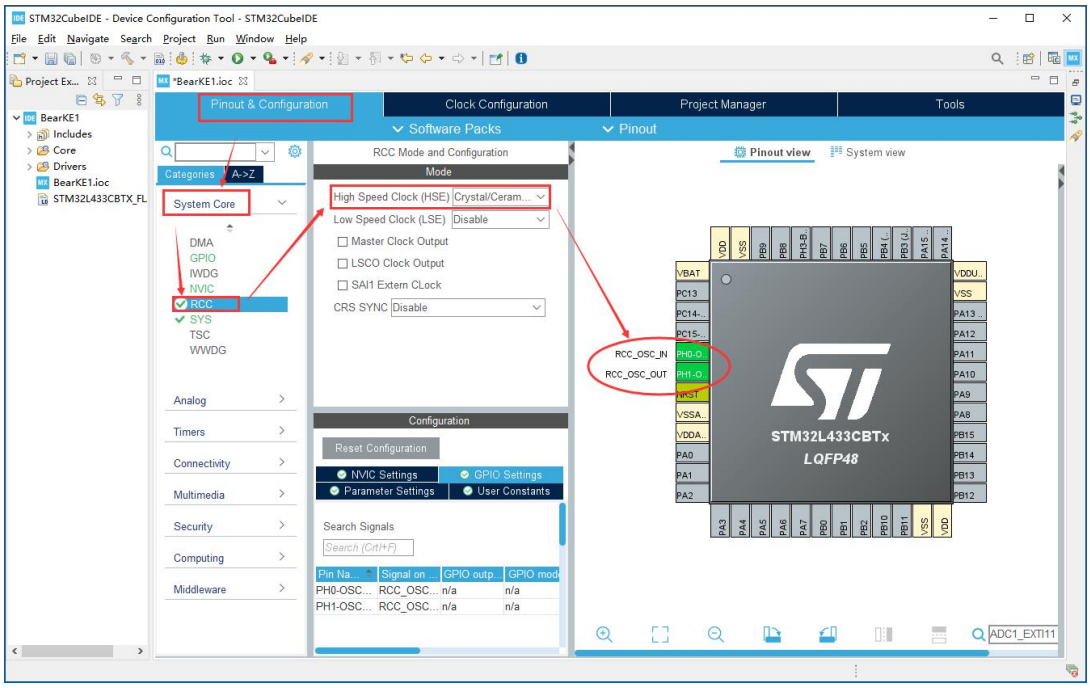


1.3. STM32CubeMX 配置

1. 项目创建好之后，默认将会进入到 STM32CubeMX 图形化配置界面，通过该界面可以对 CPU 进行配置：

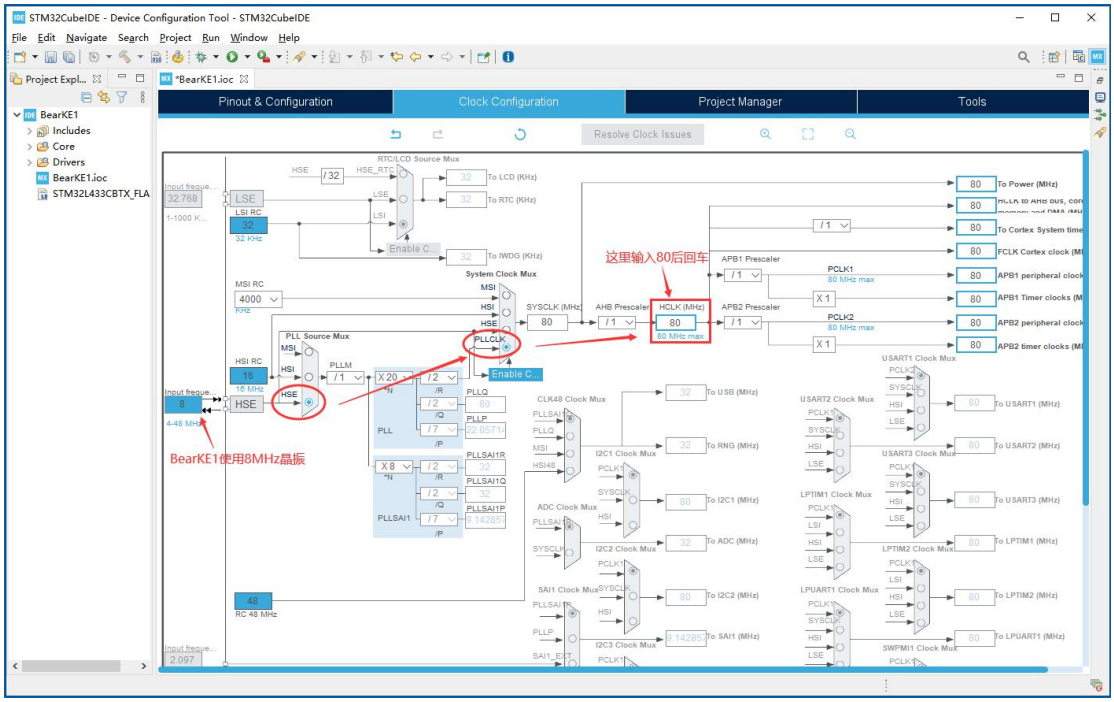


2. 首先配置使能 CPU 外部晶振

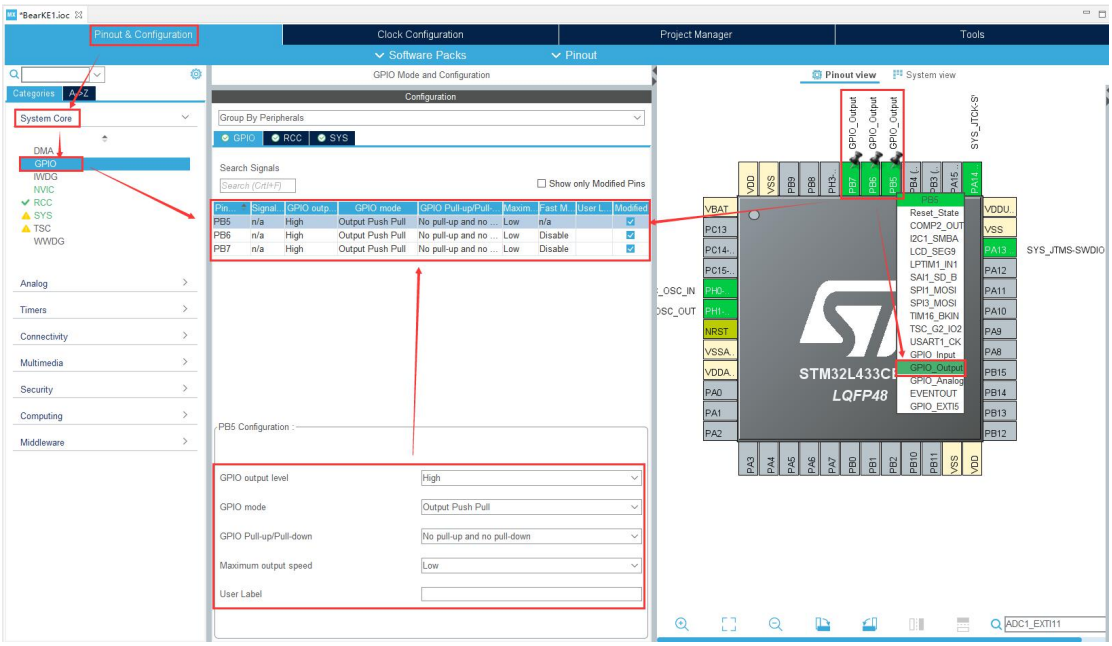


3. 配置 CPU 时钟树

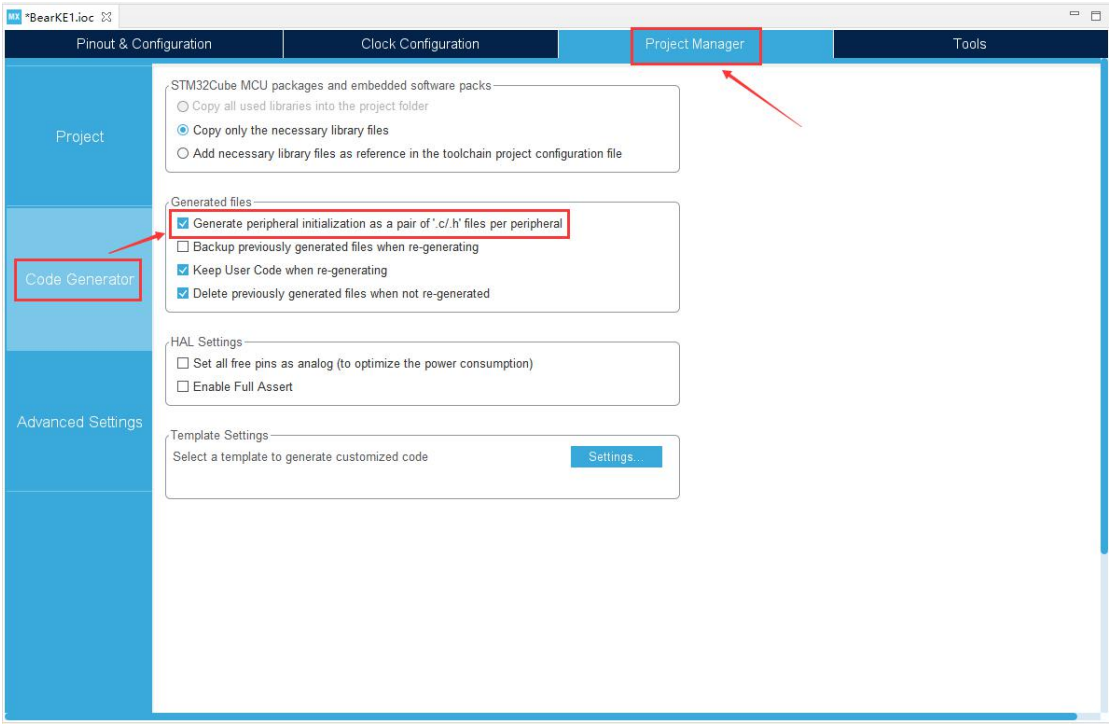
在 HCLK 配置里输入 80 后按回车，将会自动配置好 CPU 的时钟树：



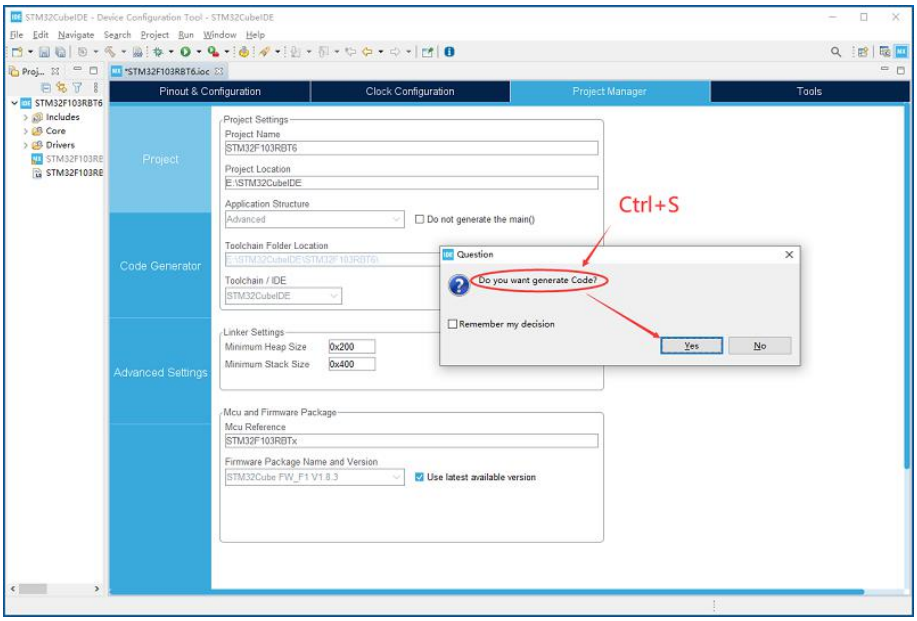
4. 配置 Led 连接的相应 CPU 管脚，设置默认电平为高电平（灯状态为灭）



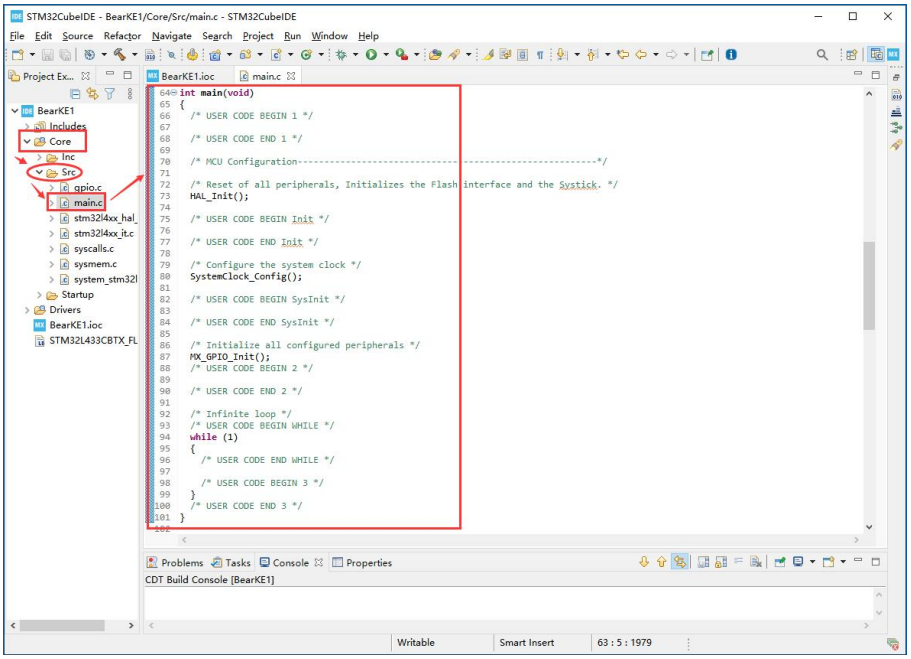
5. 配置代码生产的相关选项



6. 任意位置按 Ctrl+S 将开始自动生产代码



7. 下面是自动生成的代码



8. 下面是项目源码存储路径及目录结构

1.4. 程序修改及编译

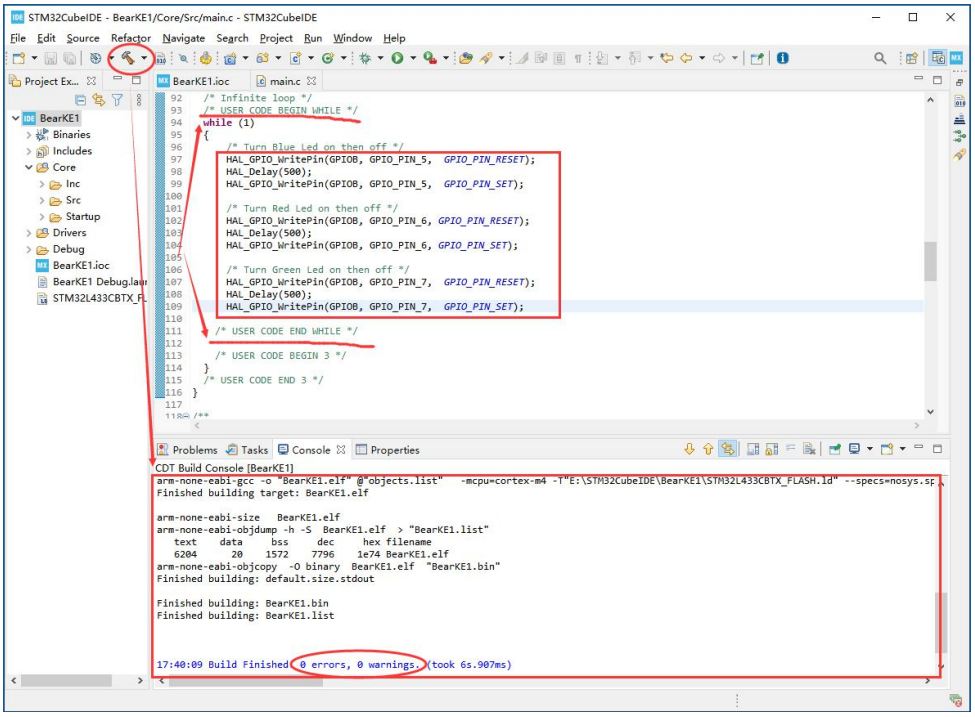
STM32CubeIDE 在添加代码时务必在 USER CODE BEGIN ... 和 USER CODE END ... 之间添加，否则 STM32CubeIDE 在重新生成代码时，会把我们添加的代码给删除。

1. 修改 main.c 中的 while 循环代码，添加 Led 流水灯的控制代码如下:

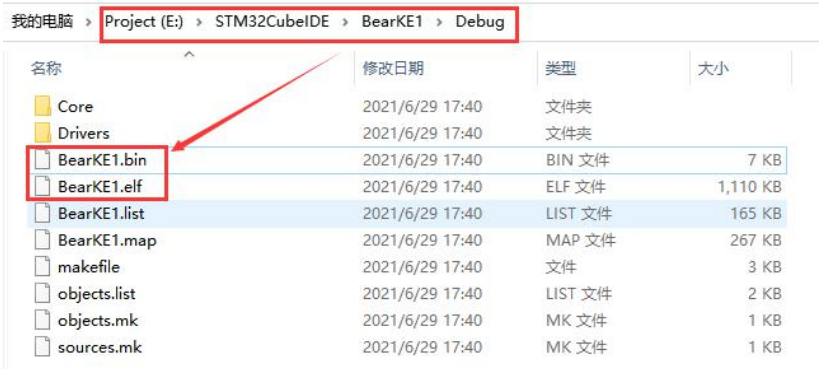
```
... ...
/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
    /* Turn Blue Led on then off */
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5,  GPIO_PIN_RESET);
    HAL_Delay(500);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5,  GPIO_PIN_SET);

    /* Turn Red Led on then off */
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_RESET);
    HAL_Delay(500);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_SET);

    /* Turn Green Led on then off */
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7,  GPIO_PIN_RESET);
    HAL_Delay(500);
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_7,  GPIO_PIN_SET);

  /* USER CODE END WHILE */

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

2. 源码编译

点击工具栏上的锤子按钮将开始编译代码：



编译生成的可执行文件将会存放到项目的 Debug 文件夹下：



这里的可执行文件格式有两个：.bin 文件和.elf 文件。

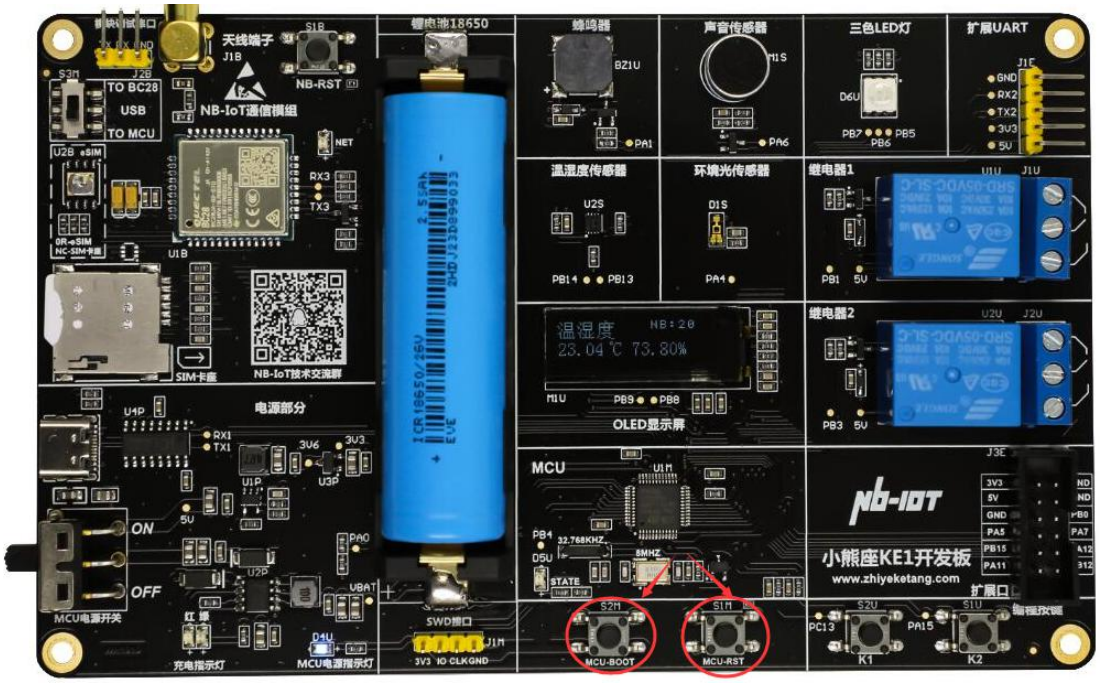
- STM32CubeIDE 使用的是 GCC 编译，它编译生成的可执行文件格式默认为 ELF(Executable and linking format)格式，该格式是 Linux 系统下的常用可执行文件格式；

- bin 文件就是直接的二进制文件，内部没有地址标记，一般用编程器直接烧写到目标开发板上运行；

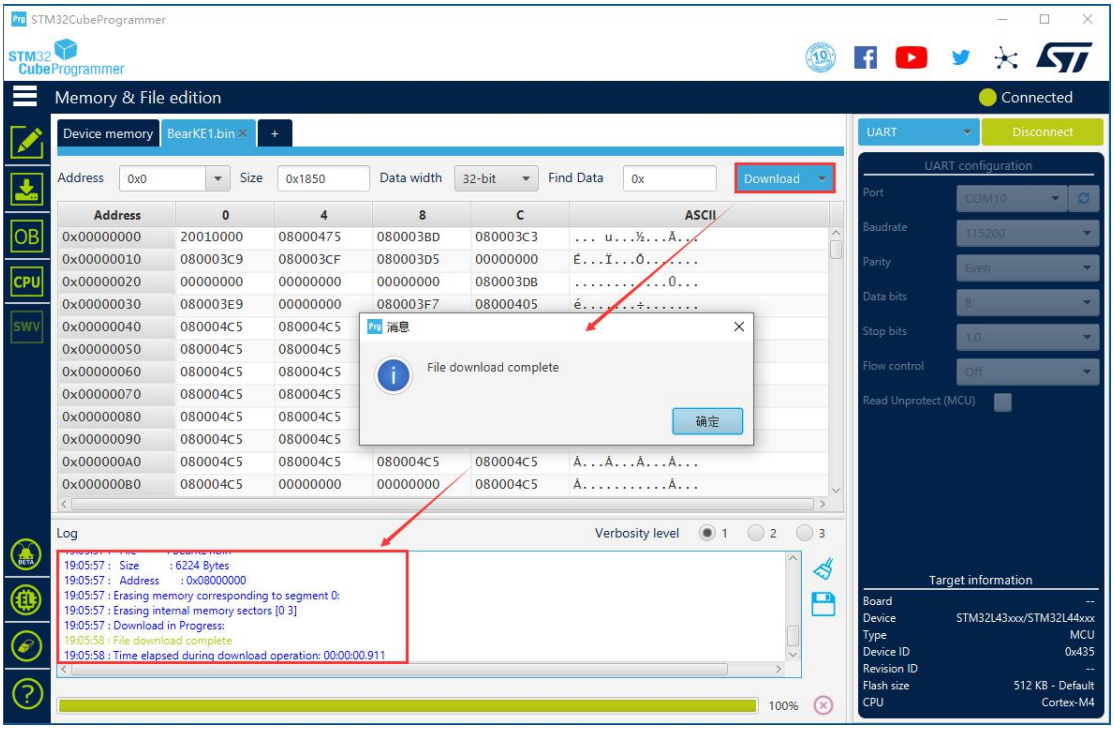
编译生成的可执行程序必须烧录到开发板上才能运行，STM32 CPU 支持两种烧录方式，一种是使用串口 ISP 烧录，另外一种是使用 ST-Link/J-link 等 ARM 调试器烧录，接下来我们将分别介绍这两种烧录方式。

1.5. 串口 ISP 烧录

接下来，我们按住开发板的 MCU_BOOT 按键，同时再按一下 MCU_RST 按键让系统复位，再松开 MCU_BOOT 按键此时开发板将进入到 ISP 串口烧录模式。



接下来我们运行 STM32CubeProgrammer 程序，选择相应的串口连接后开始烧录：



烧录成功之后，按 MCU_RST 复位按钮将会重启开发板，这时将会看到红绿蓝三个 Led 灯依次闪烁：

1.6. 代码缺陷

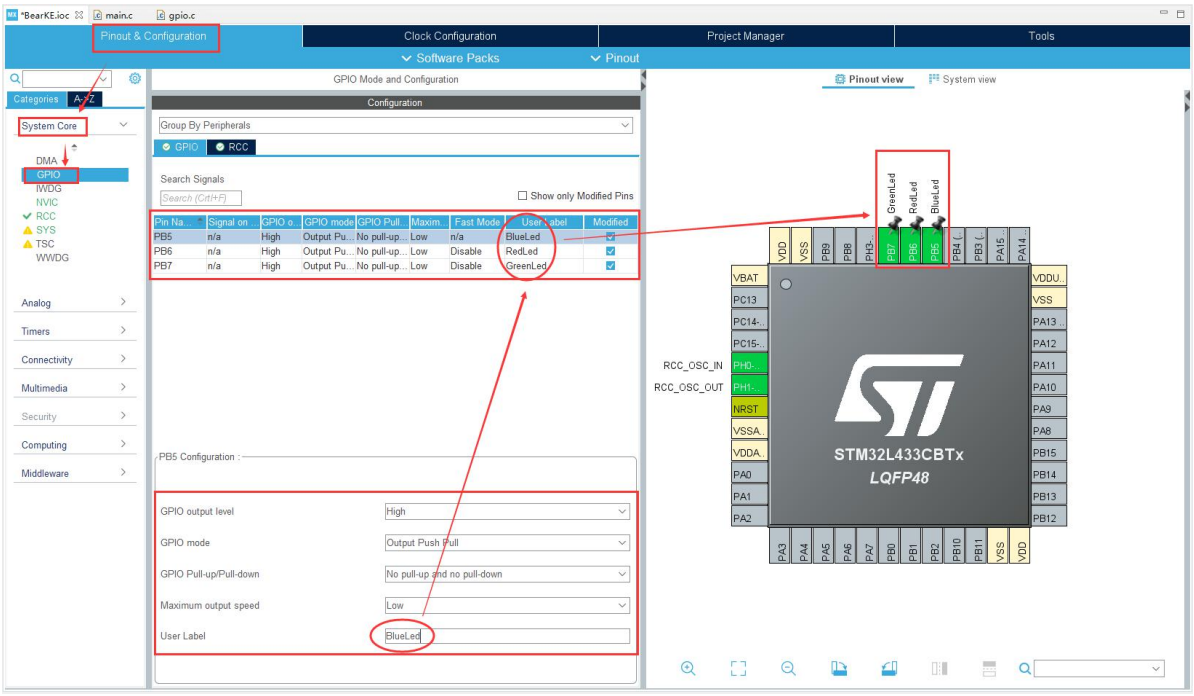
在上面的这段源码中，我们只能看出把 PC1、PC2、PC3 这三个管脚拉高/低电平，那这三个管脚是连的什么器件，高电平或低电平具体实现了什么功能却并不清晰。这时我们必须对照原理图才能了解代码的真实含义。

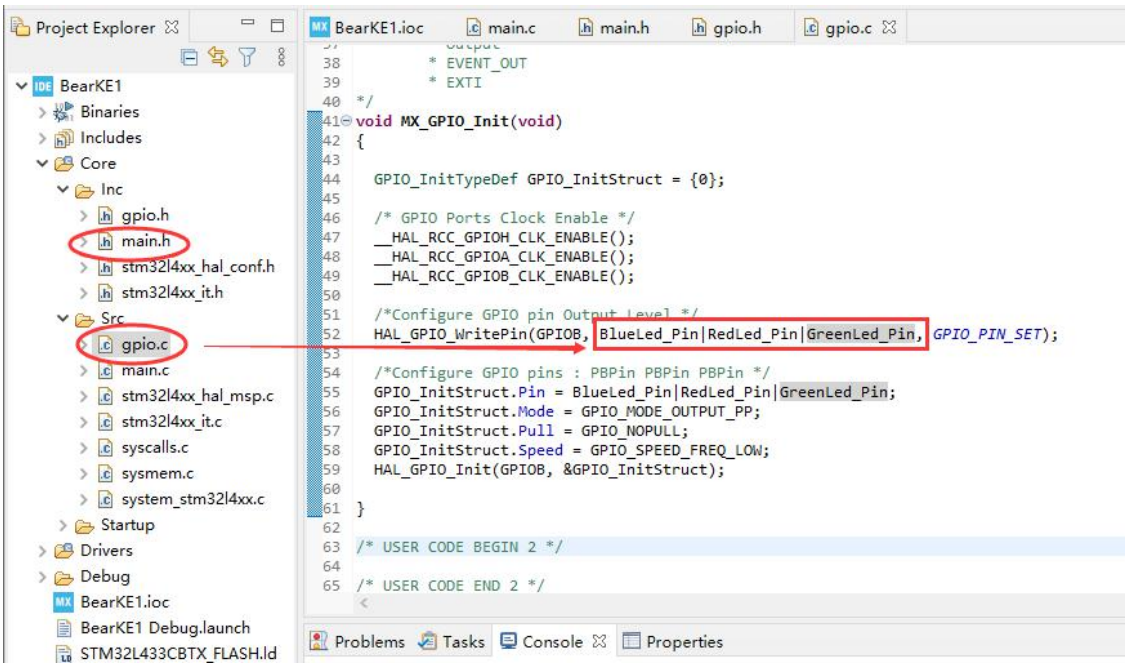
接下来我们将使用 C 语言编程的一些技巧，让我们的程序代码变得更容易理解。

2. 代码优化

2.1. STM32CubeIDE 对管脚重命名

在图形化界面里对连接 Led 的每个 GPIO 管脚重命名，并重新生成代码。这时候生成代码的三个 Led 的变量名分别为 BlueLed_Pin、RedLed_Pin 和 GreenLed_Pin 这些有意义的名称了。





2.2. Led 操作函数封装

源文件 gpio.c 添加上面两个 Led 控制函数的定义：

```
... ...
/* USER CODE BEGIN 2 */
void turn_led(int which, int status)
{
    GPIO_PinState       level;

    level = (OFF==status) ? GPIO_PIN_SET : GPIO_PIN_RESET;

    switch( which )
    {
        case RedLed:
            HAL_GPIO_WritePin(RedLed_GPIO_Port, RedLed_Pin, level);
            break;

        case GreenLed:
            HAL_GPIO_WritePin(GreenLed_GPIO_Port, GreenLed_Pin, level);
            break;

        case BlueLed:
            HAL_GPIO_WritePin(BlueLed_GPIO_Port, BlueLed_Pin, level);
            break;

        default:
            break;
    }
}

void blink_led(int which, uint32_t interval)
{
    turn_led(which, ON);
    HAL_Delay(interval);

    turn_led(which, OFF);
    HAL_Delay(interval);
}
/* USER CODE END 2 */
... ...
```

头文件 gpio.h 里添加 Led 控制函数 turn_led()和 blink_led()的声明及相关枚举、宏定义：

```
... ...
/* USER CODE BEGIN Prototypes */
enum
{
    RedLed,
    GreenLed,
    BlueLed,
    LedMax,
};

#define OFF        0
#define ON         1

/*  函数说明：  控制 LED 亮/灭的功能函数
  * 参数说明：  which 指定要控制哪个灯，取值为枚举里的 RedLed、GreenLed 或 BlueLed
  *            status 要控制灯亮还是灭，取值为宏定义 ON 或 OFF
  *   返回值：  无
  */
extern void turn_led(int which, int status);

/*  函数说明：  控制 LED 闪烁的功能函数
  * 参数说明：  which 指定要控制哪个灯，取值为枚举里的 RedLed、GreenLed 或 BlueLed
  *            interval 指定闪烁的间隔时间，其单位为毫秒
  *   返回值：  无
  */
extern void blink_led(int which, uint32_t interval);
/* USER CODE END Prototypes */
... ...
```

2.3. main()函数代码优化

修改 main.c 中的 while 循环代码，直接调用 blink_led()实现跑马灯功能：

```
  while (1)
  {
     blink_led(BlueLed,  500);
     blink_led(RedLed,   500);
     blink_led(GreenLed, 500);
    /* USER CODE END WHILE */
    /* USER CODE BEGIN 3 */
  }
```

2.4. 编译烧录运行测试

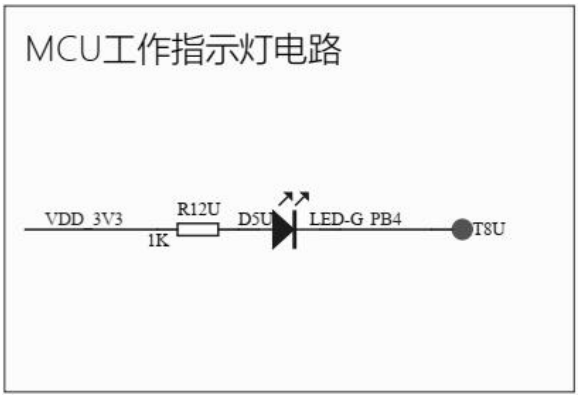
2.5. 代码进一步优化

修改源文件 gpio.c 中的 Led 操作函数 turn_led()代码，使用结构体大大简化代码：

```
/* USER CODE BEGIN 2 */
typedef struct gpio_s
{
     GPIO_TypeDef        *group;
     uint16_t             pin;
} gpio_t;

gpio_t     leds[LedMax] =
{
     { RedLed_GPIO_Port,   RedLed_Pin},
     { GreenLed_GPIO_Port, GreenLed_Pin},
     { BlueLed_GPIO_Port,  BlueLed_Pin},
};

void turn_led(int which, int status)
{
     GPIO_PinState        level;

     if( which >= LedMax )
          return ;

     level = status==OFF ? GPIO_PIN_SET : GPIO_PIN_RESET;

     HAL_GPIO_WritePin(leds[which].group, leds[which].pin, level);
}
```

3. 系统状态指示灯

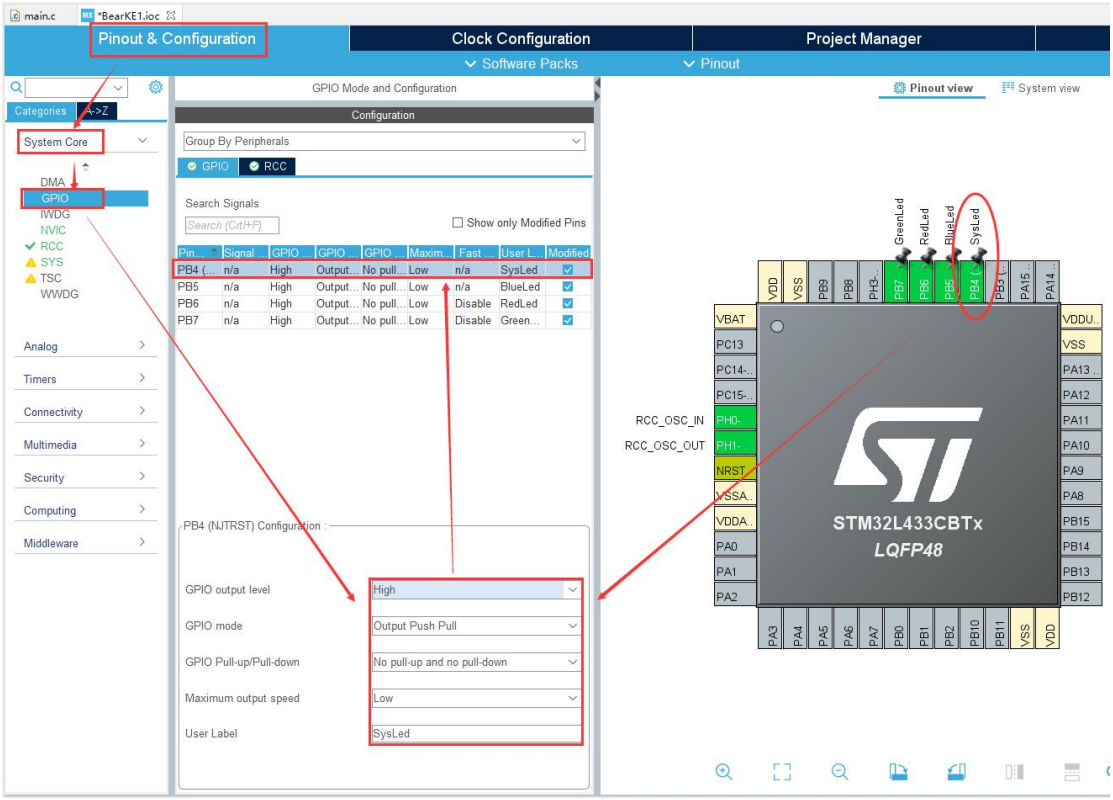
小熊座 NB-IoT 开发板除了上面的 RGB 三色 Led 灯以外，还有一个通过 PB4 管脚控制的绿色 Led 灯，通常该灯用作系统运行的状态指示灯。



接下来我们以该灯为例，来深入理解对上述代码进行抽象会有什么作用。

3.1. STM32CubeIDE 配置

首先配置 PB4 管脚为 GPIO 输出模式，由原理图可知默认低电平点亮，高电平熄灭。这里我们设置灯的状态默认为灭，并重命名管脚为 SysLed。修改好后 Ctrl+S 重新生成代码。



3.2. 源码修改

头文件 gpio.h 里添加 SysLed 灯的 enum 定义，如下所示：

```
... ...
/* USER CODE BEGIN Prototypes */
enum
{
    SysLed,  /* 如果想添加 Led 灯，到相应位置添加即可 */
    RedLed,
    GreenLed,
    BlueLed,
    LedMax,
};

#define OFF        0
#define ON         1
```

c 文件 gpio.c 里添加 SysLed 灯管脚定义，如下所示：

```
... ...
/* USER CODE BEGIN 2 */

typedef struct gpio_s
{
    GPIO_TypeDef       *group;
    uint16_t            pin;
} gpio_t;

gpio_t    leds[LedMax] =
{
    { SysLed_GPIO_Port,   SysLed_Pin},  /* 只需要在 enum 对应位置添加 Led 灯管脚定义即可 */
    { RedLed_GPIO_Port,   RedLed_Pin},
    { GreenLed_GPIO_Port, GreenLed_Pin},
    { BlueLed_GPIO_Port,  BlueLed_Pin},
};
```

接下来不用作任何其他修改，我们就可以调用之前定义好的 turn_led()、blink_led()函数了。

3.3. Led 心跳灯

很多设备在系统开始运行时，都会让系统灯快闪两下，然后再常亮一下表示系统还在运行。这里我们也可以在 gpio.c 文件中实现该函数功能，如下所示：

```
... ...
/* USER CODE BEGIN 2 */


... ...

void sysled_hearbeat(void)
{
    blink_led(SysLed, 100);
    blink_led(SysLed, 100);
    blink_led(SysLed, 800);
}

/* USER CODE END 2 */
```

修改 main.c 中的代码，while(1)循环之前添加心跳灯的函数调用，让系统上电的时候运行一下：

```
  ... ...
  sysled_hearbeat();
  while (1)
  {
     blink_led(BlueLed,  500);
     blink_led(RedLed,   500);
     blink_led(GreenLed, 500);
    /* USER CODE END WHILE */
    /* USER CODE BEGIN 3 */
  }
```

3.4. 运行测试

编译、烧录并重新运行程序，系统重启时就会看到 MCU-BOOT 按钮旁的系统状态灯会闪烁一下。