一、GPIO

GPIO：General Purpose Input Output (Pins)

通用功能输入输出(引脚)

说白了，就是从芯片内部引出来的一根导线

该导线可以被CPU设置成不同的功能模式：

输入模式

输出模式

复用模式

模拟模式

...

`STM32F4xx` 共有 `144` 个 `GPIO` 口线

分为 `9组`，记为GPIOA、GPIOB、...、GPIOI

每组管理 `16个引脚`，编号从 `0 ~ 15`

GPIOA组有16个引脚，分别记为GPIOA0、GPIOA1、...、GPIOA15

简写为：PA0、PA1、...、PA15

其他分组同理

二、GPIO功能配置

1. 输出功能

（1）推挽输出：可以输出高电平或低电平

（2）开漏输出：只能输出低电平

2. 输入功能

（1）上拉输入：在电路中接了一个上拉电阻，引脚默认的电平状态为高电平

（2）下拉输入：在电路中接了一个下拉电阻，引脚默认的电平状态为低电平

（3）浮空输入：既不接入上拉也不介入下拉，引脚的电平状态完全由外部决定

三、GPIO的应用

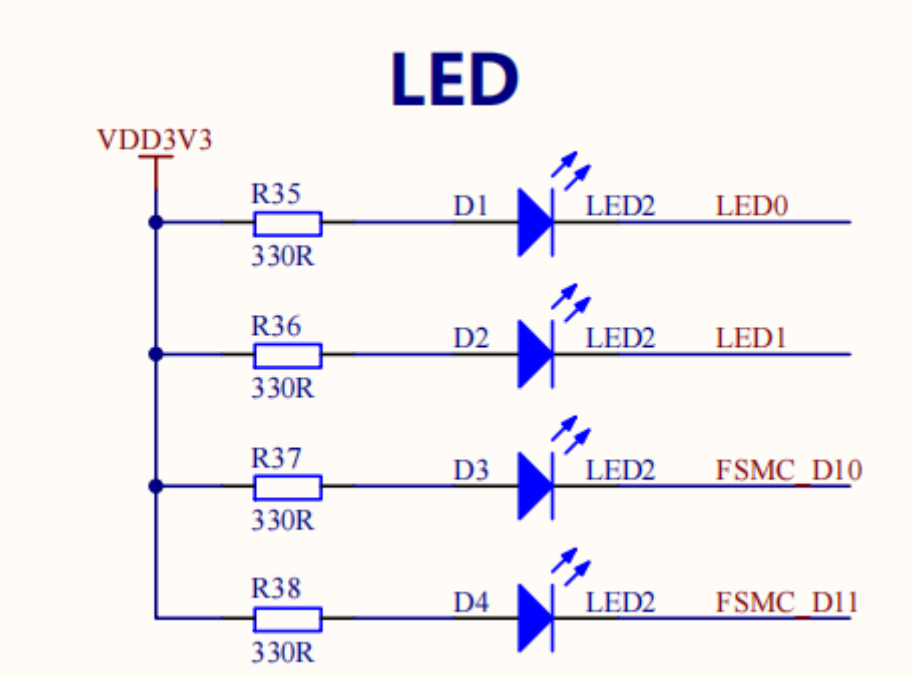
所有单片机开发都需要参考原理图

本次的开发板开发，要参考原理图：GEC-M4原理图.pdf

1. LED

以开发LED为例

step1：在硬件原理图中找到对应硬件所在的位置



step2：找到网络标签对应的CPU引脚

```
LED0 --- PF9
LED1 --- PF10
FSMC D10 --- PE13
FSMC_D11 ---- PE14
```

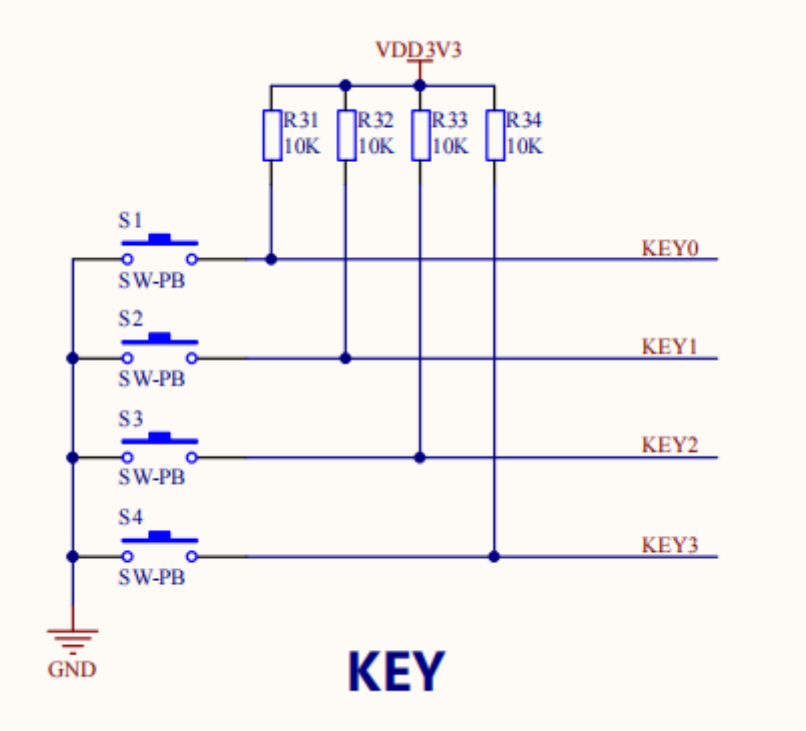
step3：查看原理图，分析电路具体的工作原理

如果说要让二极管导通（让LED工作）的话，就需要二极管的右端也就是连接到CPU的那一端电压比3.3V小

我们可以通过编写代码，控制CPU向特定的引脚输出一个较低的电压 或者 较高的电压

2. 独立按键

step1：在硬件原理图中找到对应硬件所在的位置



step2：找到网络标签对应的CPU引脚

```
S1      KEY0 --- PA0
S2      KEY1 --- PE2
S3      KEY2 --- PE3
S4      KEY3 --- PE4
```

step3：查看原理图，分析电路具体的工作原理

当按键按下时，读取引脚的电平状态为低电平

当按键松开时，读取引脚的电平状态为高电平

```
if(引脚的电平状态为低电平)
{
    按键被按下
}
```

四、使用固件库操作GPIO

步骤：

step1：使能GPIO分组的时钟

step2：GPIO引脚初始化

step3：操作GPIO

1. 使能GPIO分组的时钟

就是给时钟单元上电

`STM32F4_CODE_模板\FWLIB\src\stm32f1xx_rcc.c`

```
STM32F4xx时钟总线：
    AHB3
    AHB2
    AHB1
    APB2
    APB1
    ...

    // XXXX 代表GPIO对应的时钟总线
    // 查看方式 《STM32F4xx中文参考手册.pdf》 P53
void RCC_XXXXPeriphClockCmd(uint32_t RCC_XXXXPeriph, FunctionalState NewState)
// 功能：给指定的外设单元上电
    @RCC_XXXXPeriph          指定代操作的外设名
                                RCC_AHB1Periph_GPIOA    GPIOA clock
                                RCC_AHB1Periph_GPIOB    GPIOB clock
                                ....（stm32f4xx_rcc.c L2291）
    @NewState                指定新状态
                                ENABLE
                                DISABLE

eg：给PF9使能
    1. PF9属于GPIOF分组
    2. GPIOF属于AHB1总线
    3. 找到GPIOF的外设名称：RCC_AHB1Periph_GPIOF

    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOF, ENABLE);

练习：使能PE13、PE14使能
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOE, ENABLE);
```

2. GPIO引脚初始化

设置GPIO相关配置

```
void GPIO_Init(GPIO_TypeDef* GPIOx, GPIO_InitTypeDef* GPIO_InitStruct)
    @GPIOx               指定要设置的GPIO所属的分组
                            GPIOA
                            GPIOB
                            ...
                            GPIOI
    @GPIO_InitStruct     结构体指针
                         指向的空间，保存了要设置的配置信息
```

```
GPIO_InitTypeDeftypedef struct
{
  uint32_t GPIO_Pin;// 指定具体的引脚编号
        GPIO_Pin0 ~ GPIO_Pin_15

  GPIOMode_TypeDef GPIO_Mode;// 指定工作模式
        GPIO_Mode_IN// 输入模式
        GPIO_Mode_OUT// 输出模式
        GPIO_Mode_AF// 复用模式
        GPIO_Mode_AN// 模拟模式

  GPIOSpeed_TypeDef GPIO_Speed;// 指定引脚的传输速率
        GPIO_Speed_50MHz <= 一般用这个

  GPIOOType_TypeDef GPIO_OType;// 指定具体的输出模式
        GPIO_OType_PP// 推挽输出
        GPIO_OType_OD// 开漏输出

  GPIOPuPd_TypeDef GPIO_PuPd;// 指定具体的输入模式
        GPIO_PuPd_NOPULL// 浮空输入
        GPIO_PuPd_UP// 上拉输入
        GPIO_PuPd_DOWN// 下拉输入
}GPIO_InitTypeDef;

eg：配置PF9为推挽输出模式
    1. 定义结构体变量
    2. 给结构体变量的成员变量赋值
    3. 调用函数完成配置

GPIO_InitTypeDef g;
g.GPIO_Pin = GPIO_Pin_9;
g.GPIO_Mode = GPIO_Mode_OUT;
g.GPIO_Speed = GPIO_Speed_50MHz;
g.GPIO_OType = GPIO_OType_PP;
GPIO_Init(GPIOF, &g);

练习：配置PE13为推挽输出模式
GPIO_InitTypeDef g;
g.GPIO_Pin = GPIO_Pin_13;
g.GPIO_Mode = GPIO_Mode_OUT;
g.GPIO_Speed = GPIO_Speed_50MHz;
g.GPIO_OType = GPIO_OType_PP;
GPIO_Init(GPIOE, &g);
```

3. 操作GPIO

```
void GPIO_SetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
// 将指定的GPIO置位（置为高电平）

void GPIO_ResetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
// 将指定的GPIO复位（值为低电平）
```

```
@GPIOx      指定GPIO分组
                GPIOA
                GPIOB
                ...
                GPIOI
@GPIO_Pin   指定具体的GPIO编号
                GPIO_Pin_1 ~ GPIO_Pin_15

eg：设置PF9为高电平
GPIO_SetBits(GPIOF, GPIO_Pin_9);


uint8_t GPIO_ReadInputDataBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
// 读取指定引脚的电平状态
@GPIOx      指定GPIO分组
@GPIO_Pin   指定具体的GPIO编号
@return     1   表示引脚的电平状态为高电平
            0   表示引脚的电平状态为低电平

eg：判断PA0是否为低电平
if(GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_0) == 0)
{
    // S1 被按下
    // 操作

    while(GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_0) == 0);// 等待按键松开
}
```

练习：编写代码点亮D1

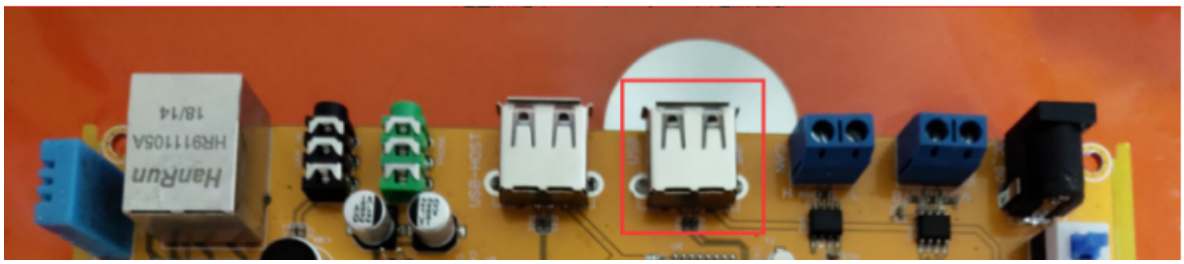
```
// D1 -- LED0 -- PF9

int main()
{
  // step1：使能PF9所属分组的时钟
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOF, ENABLE);

    // step2：配置PF9为推挽输出模式
    GPIO_InitTypeDef g;
    g.GPIO_Pin = GPIO_Pin_9;
    g.GPIO_Mode = GPIO_Mode_OUT;
    g.GPIO_Speed = GPIO_Speed_50MHz;
    g.GPIO_OType = GPIO_OType_PP;
    GPIO_Init(GPIOF, &g);

    // step3：操作PF9：设置PF9为低电平
    GPIO_ResetBits(GPIOF, GPIO_Pin_9);

    while(1);// 卡住程序
}
```

五、程序烧录

1. 检查CH340驱动是否安装成功

step1：使用数据线连接开发板和PC

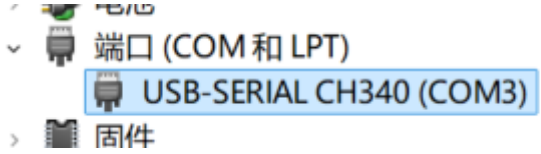
开发板上边有两个USB口，连接右边那个USB口



step2：打开电脑的`设备管理器`

step3：查看`端口`选项

如果出现了`USB-Serial CH340(COMx)`项目，则证明驱动安装成功

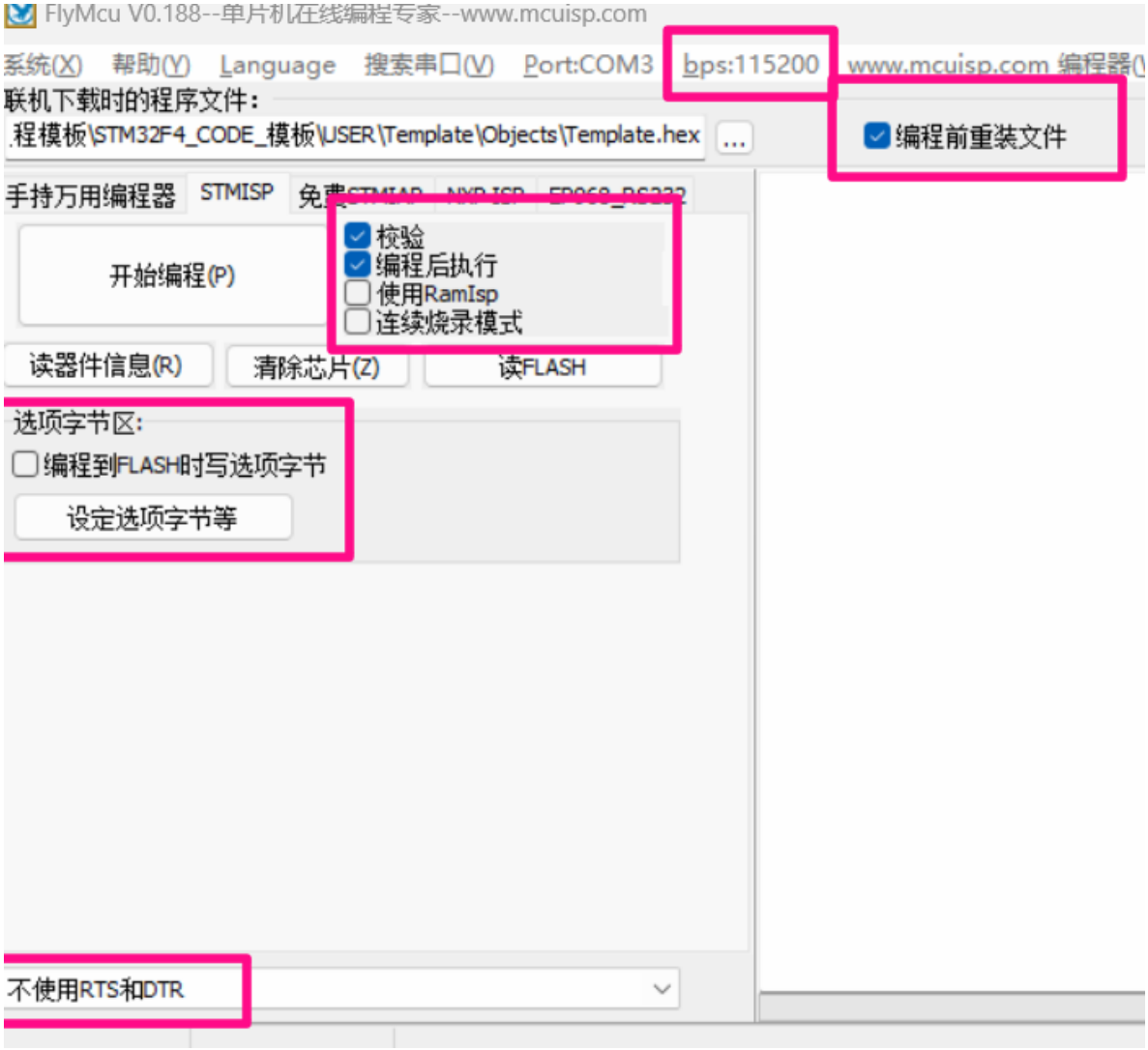


2. 串口烧录软件的使用

step1：打开串口烧录软件`FlyMCU.exe`

step2：点击`搜索串口`，在`Port`选线的下拉栏中选择与`设备管理器中`相同的端口

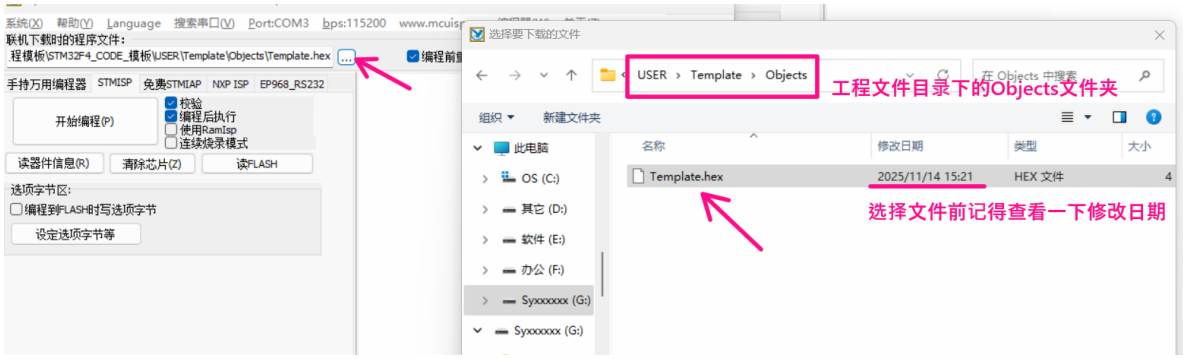
step3：按照图示将软件配好



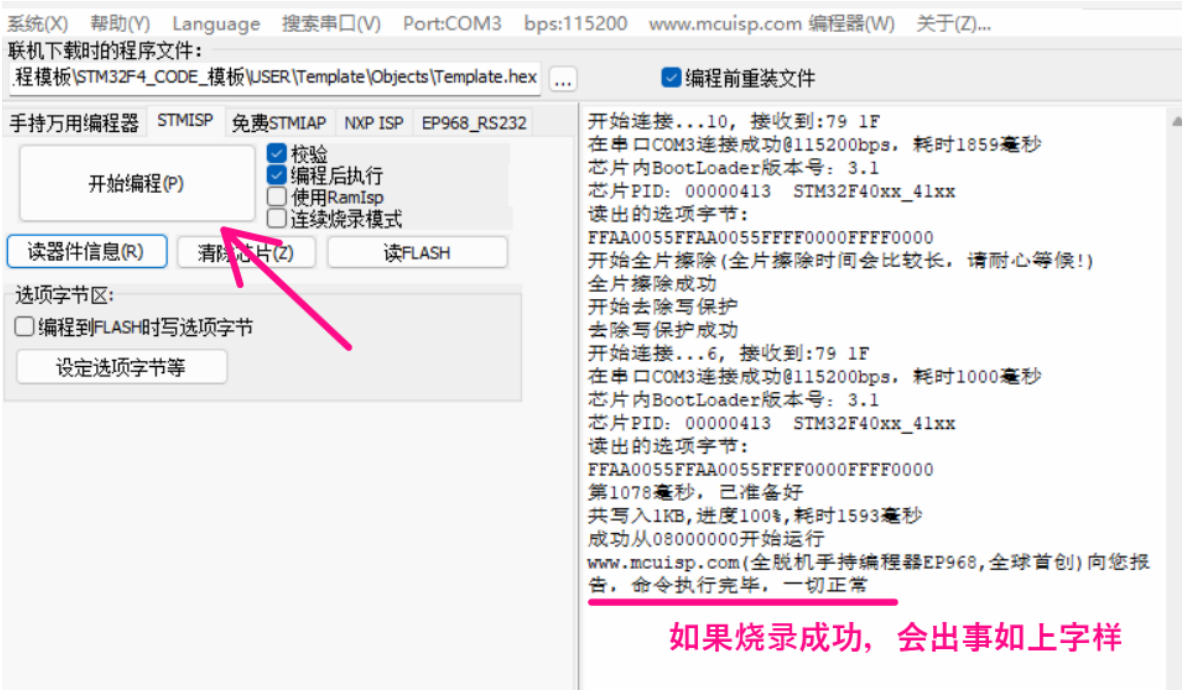
step4：选择 联机下载时的程序文件

即在Keil工程中编译所生成的 hex文件

hex文件 一般保存在工程目录下的 Object文件夹 中

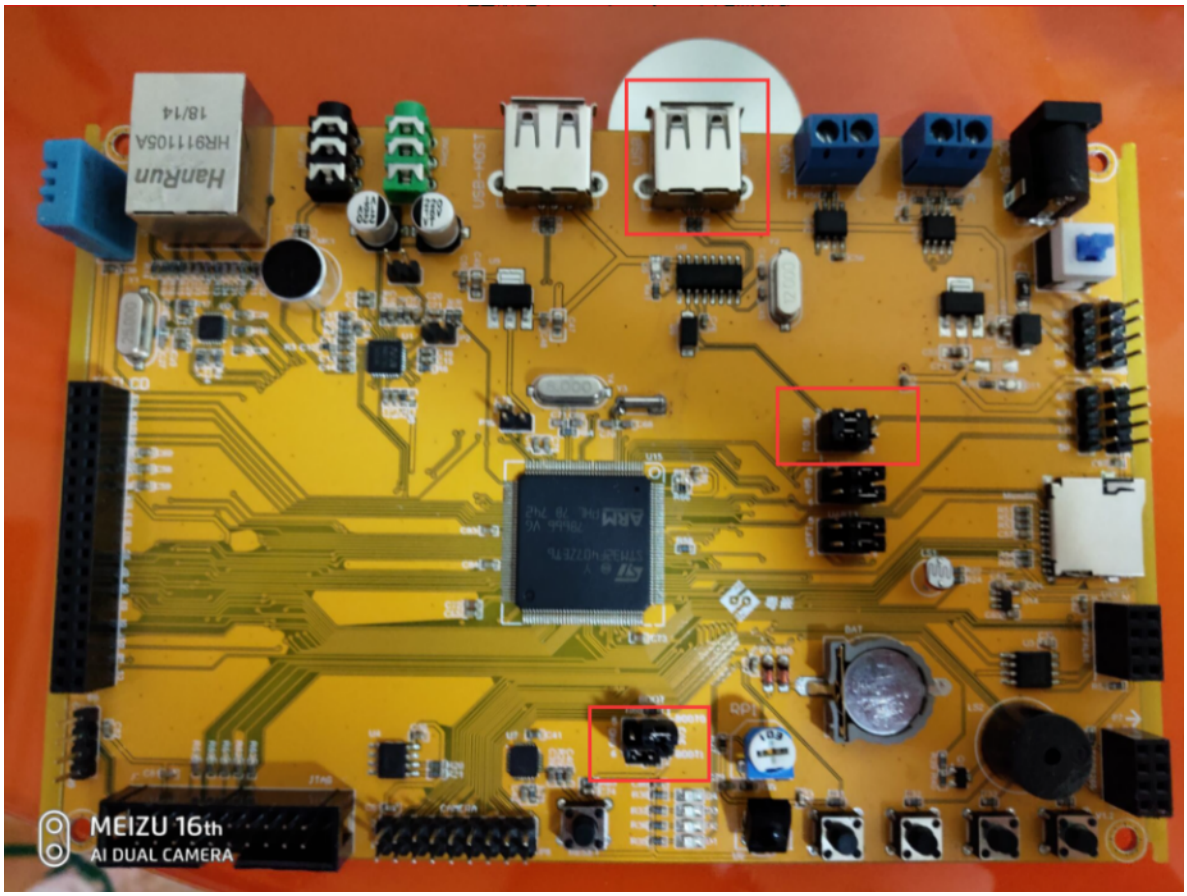


step5：点击 开始编程



烧录失败的情况:

(1) 检查跳线帽是否设置好了，USB口是否连接正确



(2) 点击`开始编程`按钮后，如果一直卡在`开始连接...`字样，按下`RESET`按键

(3) 二次点击`开始编程`按钮，取消 开始编程，然后按下`RESET`按键不松手，再按下`开始编程`按钮，然后松开`RESET`按键

练习：复制点亮D1的代码，然后在Keil5中进行编译生成hex文件，并使用FlyMcu完成烧录，最终需要在板子上看见D1亮起！！！

练习：程序实现，点亮所有LED灯

```
// LED0 --- PF9
// LED1 --- PF10
// FSMC D10 --- PE13
// FSMC_D11 ---- PE14

int main()
{
    // 1. 使能PF9、PF10、PE13、PE14对应分组的时钟

    // 2. 配置PF9、PF10、PE13、PE14为推挽输出模式
    GPIO_InitTypeDef g;
    g.GPIO_Pin = GPIO_Pin_9;
    g.GPIO_Mode = GPIO_Mode_OUT;
    g.GPIO_Speed = GPIO_Speed_50MHz;
    g.GPIO_OType = GPIO_OType_PP;
    GPIO_Init(GPIOF, &g);

    g.GPIO_Pin = GPIO_Pin_10;
    GPIO_Init(GPIOF, &g);

    g.GPIO_Pin = GPIO_Pin_13;
    GPIO_Init(GPIOE, &g);

    g.GPIO_Pin = GPIO_Pin_14;
    GPIO_Init(GPIOE, &g);

    // 3. 设置对应的GPIO引脚输出低电平（点亮）

    while(1);// 卡住程序

}
```

六、流水灯

```
D1亮 => 延时 => D1灭 => D2亮 => 延时 => D2灭 .... D4亮 => 延时 => D4灭
```

`#include "delay.h"` 中存在延时函数

```
void Mdelay_Lib(int nms);// 毫秒级延时函数
void Udelay_Lib(int nms);// 微秒级延时函数


eg：我想延时1s
    Mdelay_Lib(1000);
```

```c
int main()
{
    // 1. 使能LED对应GPIO所属的时钟

    // 2. 配置LED对应GPIO为推挽输出

    // 3. 设置LED默认状态为熄灭（LED对应GPIO引脚为高电平）

    // 开始流水灯
    while(1)
    {
        // D4亮

        // 延时

        // D4灭

        // D3亮

        // 延时

        // D3灭

        ...
    }
}
```

七、按键控灯

练习1：代码实现，使用S1 控制D1的亮灭

```
按下S1：
    D1是灭的 => 点亮D1
    D1是亮的 => 熄灭D1
```