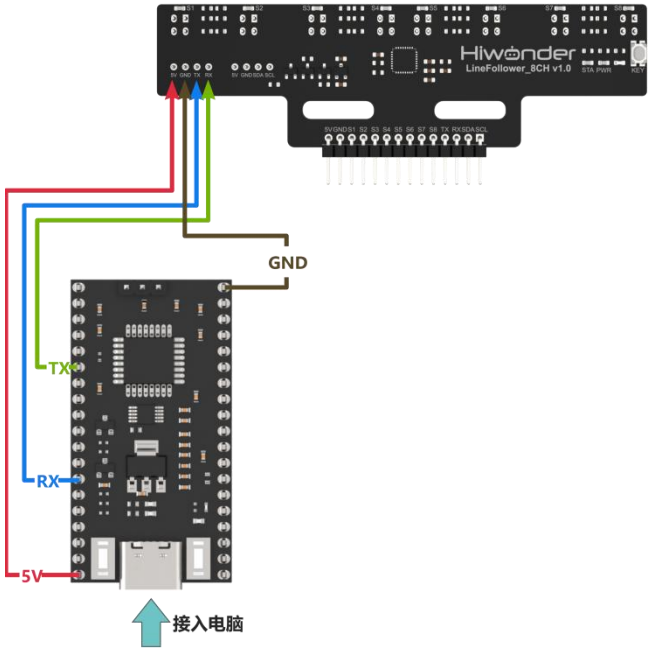
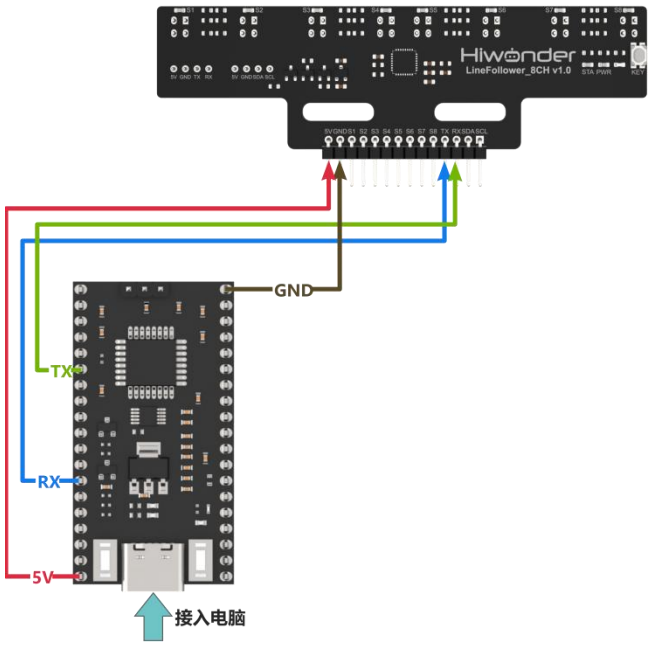
01 UART通信说明

1. 准备工作

1.1 接线说明

接线时，8路巡线传感器的5V、GND、TX和RX引脚需与STM32开发板进行连接，接线方式如下图所示：





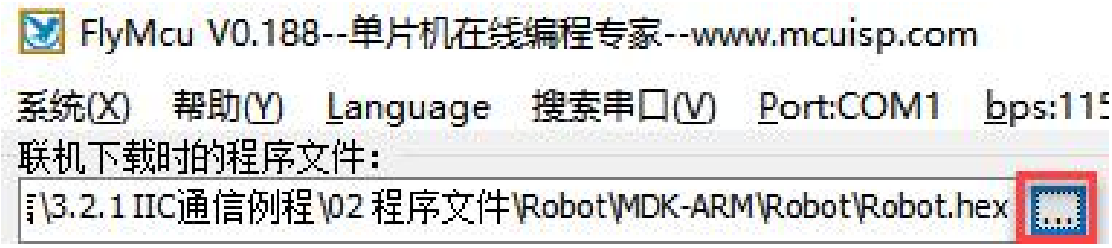
注意：通电前确保不要有金属物体接触到主板，否则可能会因主板底部的引脚导致电路短路从而烧毁主板。

1.2 程序下载

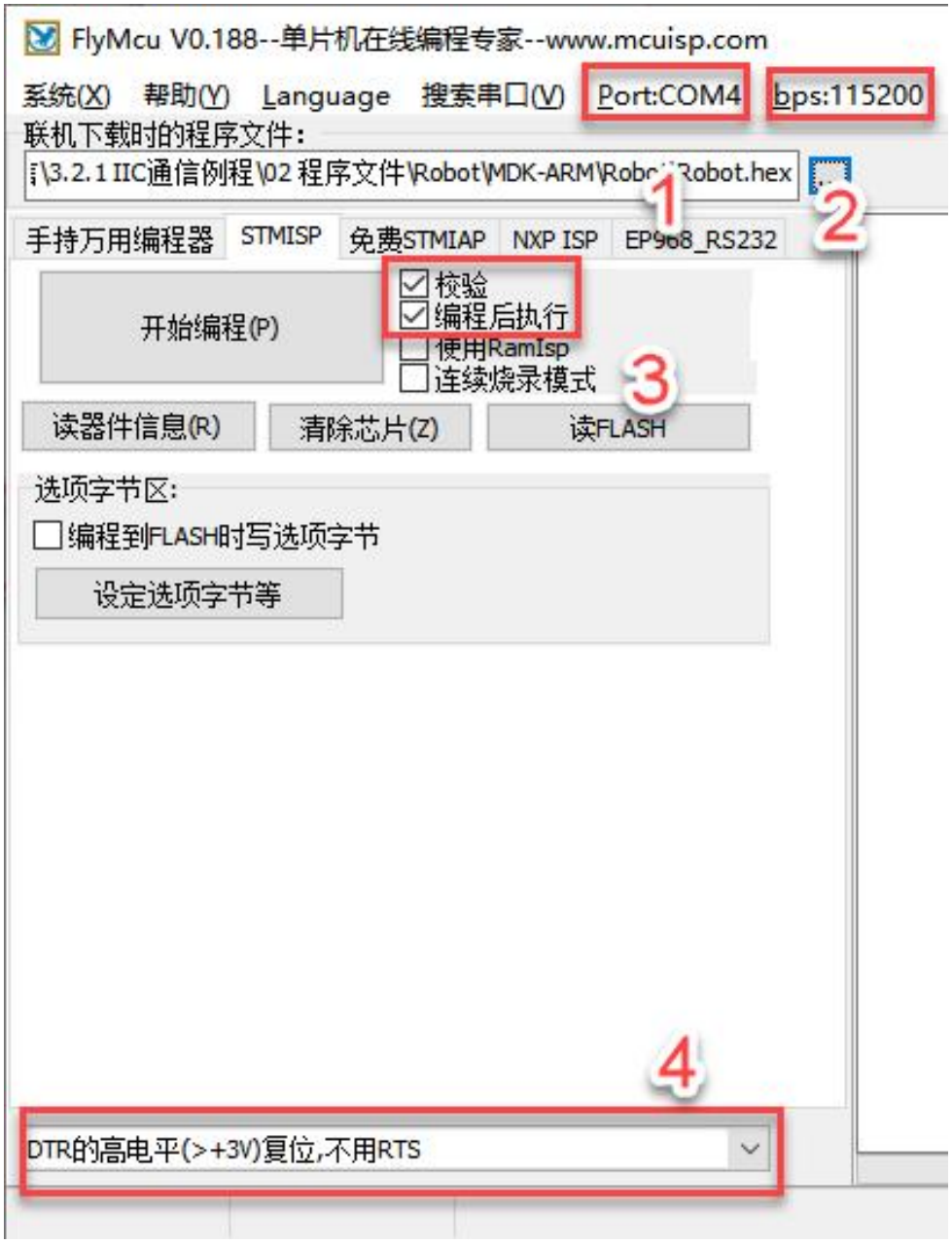
1） 将核心板通过 USB 线连接到电脑。

2） 双击打开 STM32 串口烧录软件“**FlyMcu**”  。

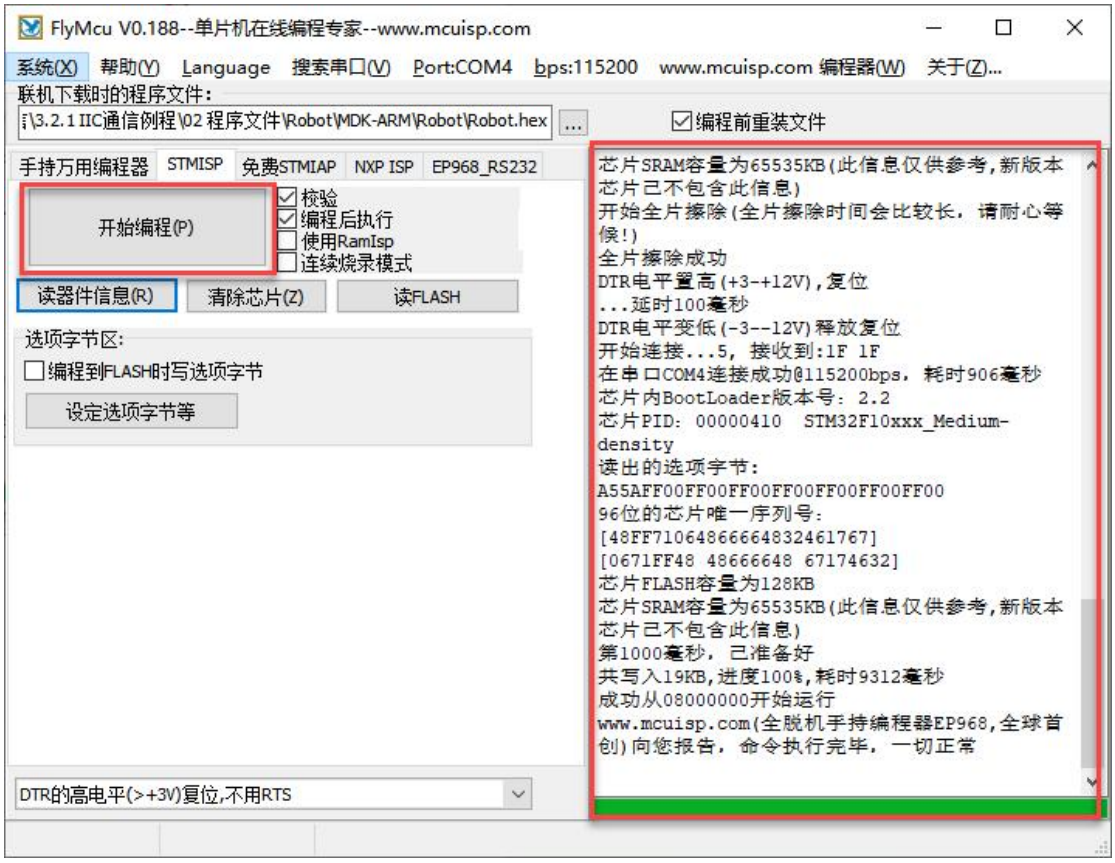
3） 点击“**...**”图标，在对应工程的“**Robot\MDK-ARM\Robot**”路径选择待烧录 HEX 文件。



4） 在下图①处，选择核心板对应的串口号；在②处，选择波特率为 115200；在③④处按照图中选项进行配置。



5） 点击“**开始编程**”后，按下**核心板 RST 按键**，程序自动开始下载，当右侧日志输出栏输出如下内容时，表示程序下载完毕。



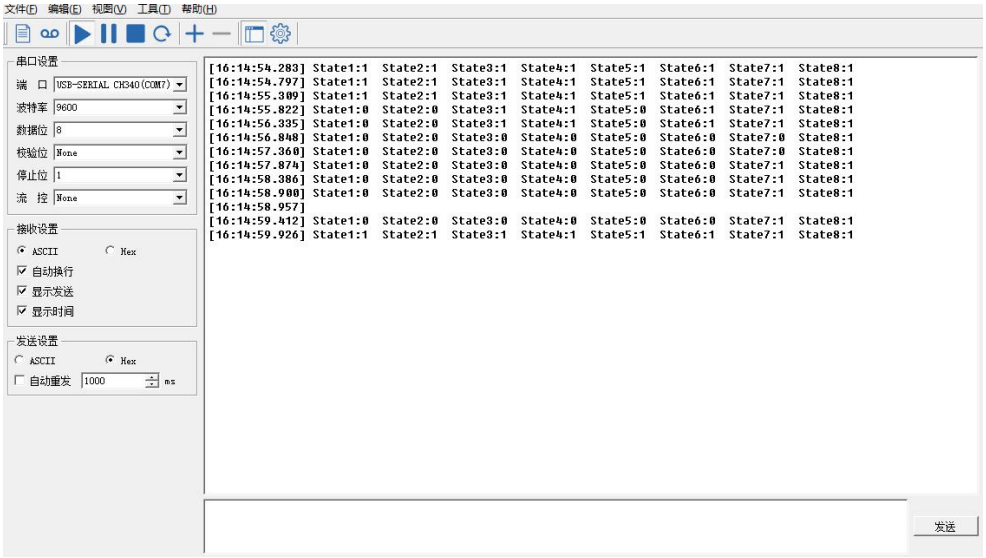
2. 测试案例

本例程使用 STM32 开发板获取 8 路巡线模块的识别结果，并通过串口打印出来。

2.1 实现效果

注意：模块在识别前，需要先对模块进行学习校准，才能进行识别。

当 8 路巡线模块识别到对应颜色的巡线目标后，会串口将会打印每一路传感器的状态，可在程序中切换打印模拟值或阈值数据。



2.2 程序简要分析

1) 首先需要导入程序相关库文件，以及巡线传感器库文件，里面包含巡线传感器的功能接口。

```
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "adc.h"
#include "dma.h"
#include "i2c.h"
#include "spi.h"
#include "tim.h"
#include "usart.h"
#include "gpio.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

#include "global.h"
#include "stdio.h"
#include "stdlib.h"
#include "led.h"
#include "buzzer.h"
#include "adc_sample.h"
#include "LineFollow.h"
/* USER CODE END Includes */
```

2) 在 main 函数中，对相关硬件进行初始化，同时调用 LineFollowUART_init()函数，初始化巡线传感器的功能模式为手动发送模式，此状态下需要通过主控向传感器发送读取指令，才能读取到相应的数据。

```
HAL_Init();

/* USER CODE BEGIN Init */

/* USER CODE END Init */

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

/* USER CODE BEGIN SysInit */
__HAL_RCC_I2C1_CLK_ENABLE();
__HAL_RCC_DMA1_CLK_ENABLE();
/* USER CODE END SysInit */

/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_DMA_Init();
MX_TIM3_Init();
MX_USART3_UART_Init();
MX_SPI1_Init();
MX_USART1_UART_Init();
MX_ADC1_Init();
MX_I2C1_Init();
MX_TIM4_Init();
MX_USART2_UART_Init();

/* Initialize interrupts */
MX_NVIC_Init();
/* USER CODE BEGIN 2 */
led_init();
LineFollowUART_init(LINEFOLLOW_MODE_MANUAL);
```

3） 在传感器的初始化函数中，调用该函数会通过 LineFollow_write_and_read()，向传感器发送一次工作模式的指令。这里需要注意的是，传感器在上电后只会接收一次设置工作模式的指令，若想切换工作模式需要将模块重新上电再发送指令。

```
void LineFollowUART_init(uint8_t Mode)
{
    memset(&LineFollow, 0, sizeof(LineFollow));
    __HAL_UART_CLEAR_FLAG(&huart2, UART_FLAG_TXE);
    __HAL_UART_CLEAR_FLAG(&huart2, UART_FLAG_TC);
    __HAL_UART_CLEAR_FLAG(&huart2, UART_FLAG_RXNE);
    LineFollow.work_mode = Mode;
    LineFollow_write_and_read(&LineFollow, LineFollow.work_mode, true);
    LineFollow.it_state = LINEFOLLOW_WRITE_DATA_READY;
}
```

4） 在 while 循环中，以获取传感器电平值为例，通过 LineFollowUART_State()函数获取传感器的电平状态，并存储在 LineFollowLearn 对象中，最后通过串口打印出来。由于串口模式下工作模式需要重新上电才能改变，所以这里无法同时打印电平值、模拟值以及阈值。

```
  while (1)
  {
    /* USER CODE END WHILE */
    /* USER CODE BEGIN 3 */

    /*  Read State */
    LineFollowUART_State(&LineFollowLearn);
    printf("State1:%d  State2:%d  State3:%d  State4:%d  State5:%d  State6:%d  State7:%d  State8:%d\r\n",
     LineFollowLearn.data[0], LineFollowLearn.data[1], LineFollowLearn.data[2], LineFollowLearn.data[3],
     LineFollowLearn.data[4], LineFollowLearn.data[5], LineFollowLearn.data[6], LineFollowLearn.data[7]);

    /*  Read Analog */
//    LineFollowUART_Analog(&LineFollowLearn);
//    printf("Analog1:%d  Analog2:%d  Analog3:%d  Analog4:%d  Analog5:%d  Analog6:%d  Analog7:%d  Analog8:%d\r\n",
//     LineFollowLearn.data[0], LineFollowLearn.data[1], LineFollowLearn.data[2], LineFollowLearn.data[3],
//     LineFollowLearn.data[4], LineFollowLearn.data[5], LineFollowLearn.data[6], LineFollowLearn.data[7]);

    /*  Read Threshold */
//    LineFollowUART_Threshold(&LineFollowLearn);
//    printf("Threshold1:%d  Threshold2:%d  Threshold3:%d  Threshold4:%d  Threshold5:%d  Threshold6:%d  Threshold7:%d  Threshold8:%d\r\n",
//     LineFollowLearn.data[0], LineFollowLearn.data[1], LineFollowLearn.data[2], LineFollowLearn.data[3],
//     LineFollowLearn.data[4], LineFollowLearn.data[5], LineFollowLearn.data[6], LineFollowLearn.data[7]);

    HAL_Delay(400);
  }
  /* USER CODE END 3 */
}
```

5） 在 LineFollowUART_State()的实现中，可以看到首先会对工作模式进行判断，如果是手动发送模式，那么调用 LineFollow_write_and_read()将 LINEFOLLOW_MODE_MANUAL_STATE 读取电平指令，发送给传感器。根据协议可知接收到的数据为 1 个字节，这里通过 for 循环对数据进行解析，最后就能得到传感器每个探头的电平数据。

```
bool LineFollowUART_State(LineFollowHandleTypeDef* State)
{
  if(LineFollow.work_mode == LINEFOLLOW_MODE_MANUAL)
  {
    LineFollow.manual_mode = LINEFOLLOW_MODE_MANUAL_STATE;
    if (0 == LineFollow_write_and_read(&LineFollow, LINEFOLLOW_MODE_MANUAL_STATE, false))
    {
      for(int i=0; i<sizeof(LineFollow.data);i++){
        State->data[i] = (LineFollow.results[0] >> i) & 0x01;
      }
      LineFollow.it_state = LINEFOLLOW_WRITE_DATA_READY;
      return true;
    }
  }
```

6） 如果是自动发送模式，就不需要手动发送传感器数据，在发送的参数中填入 LINEFOLLOW_CMD_NULL，表示不需要发送数据给传感器，接收到数据后通过循环对数据进行解析。

```
  else
  {
    if (0 == LineFollow_write_and_read(&LineFollow, LINEFOLLOW_CMD_NULL, false))
    {
      for(int i=0; i<sizeof(LineFollow.data);i++){
        State->data[i] = (LineFollow.results[0] >> i) & 0x01;
      }
      LineFollow.it_state = LINEFOLLOW_WRITE_DATA_READY;
      return true;
    }
  }

  return false;
}
```

7） 在 LineFollowUART_Analog()的实现中，传感器发送给主控的数据是以数据帧的格式发送的，根据传感器的串口协议可知每一帧数据内都包含了帧头、指令、数据长度、数

据、校验值。获取到数据帧后，需要在 LineFollow_rx_handler()函数内对数据进行校验处理。最后再将数据存储到 LineFollow.results 内，通过 for 循环将数据的高 8 位和低 8 位进行解析后，得到完整的模拟值数据。

```
bool LineFollowUART_Analog(LineFollowHandleTypeDef* Analog)
{
  uint8_t count = 0;
  if(LineFollow.work_mode == LINEFOLLOW_MODE_MANUAL)
  {
    LineFollow.manual_mode = LINEFOLLOW_MODE_MANUAL_ANALOG;
    if (0 == LineFollow_write_and_read(&LineFollow, LINEFOLLOW_MODE_MANUAL_ANALOG, false))
    {
      for(int i=0; i<sizeof(LineFollow.data);i++){
        Analog->data[i] = LineFollow.results[count] | (LineFollow.results[count+1] << 8);
        count += 2;
      }
      LineFollow.it_state = LINEFOLLOW_WRITE_DATA_READY;
      return true;
    }
  }
  else
  {
    if (0 == LineFollow_write_and_read(&LineFollow, LINEFOLLOW_CMD_NULL, false))
    {
      for(int i=0; i<sizeof(LineFollow.data);i++){
        Analog->data[i] = LineFollow.results[count] | (LineFollow.results[count+1] << 8);
        count += 2;
      }
      LineFollow.it_state = LINEFOLLOW_WRITE_DATA_READY;
      return true;
    }
  }

  return false;
}
```

8） 在 LineFollowUART_Threshold()的实现中，处理与接收逻辑均与 LineFollowUART_Analog()函数一致，只是传感器发送的数据变成了阈值。

```
bool LineFollowUART_Threshold(LineFollowHandleTypeDef* Threshold)
{
  uint8_t count = 0;
  if(LineFollow.work_mode == LINEFOLLOW_MODE_MANUAL)
  {
    LineFollow.manual_mode = LINEFOLLOW_MODE_MANUAL_THRESHOLD;
    if (0 == LineFollow_write_and_read(&LineFollow, LINEFOLLOW_MODE_MANUAL_THRESHOLD, false))
    {
      for(int i=0; i<sizeof(LineFollow.data);i++){
        Threshold->data[i] = LineFollow.results[count] | (LineFollow.results[count+1] << 8);
        count += 2;
      }
      LineFollow.it_state = LINEFOLLOW_WRITE_DATA_READY;
      return true;
    }
  }
  else
  {
    if (0 == LineFollow_write_and_read(&LineFollow, LINEFOLLOW_CMD_NULL, false))
    {
      for(int i=0; i<sizeof(LineFollow.data);i++){
        Threshold->data[i] = LineFollow.results[count] | (LineFollow.results[count+1] << 8);
        count += 2;
      }
      LineFollow.it_state = LINEFOLLOW_WRITE_DATA_READY;
      return true;
    }
  }

  return false;
}
```