ZHANGSHUO_CAR_V1 此工程用来记录无人小车的驱动部分(STM32开发)

所使用单片机和环境说明：

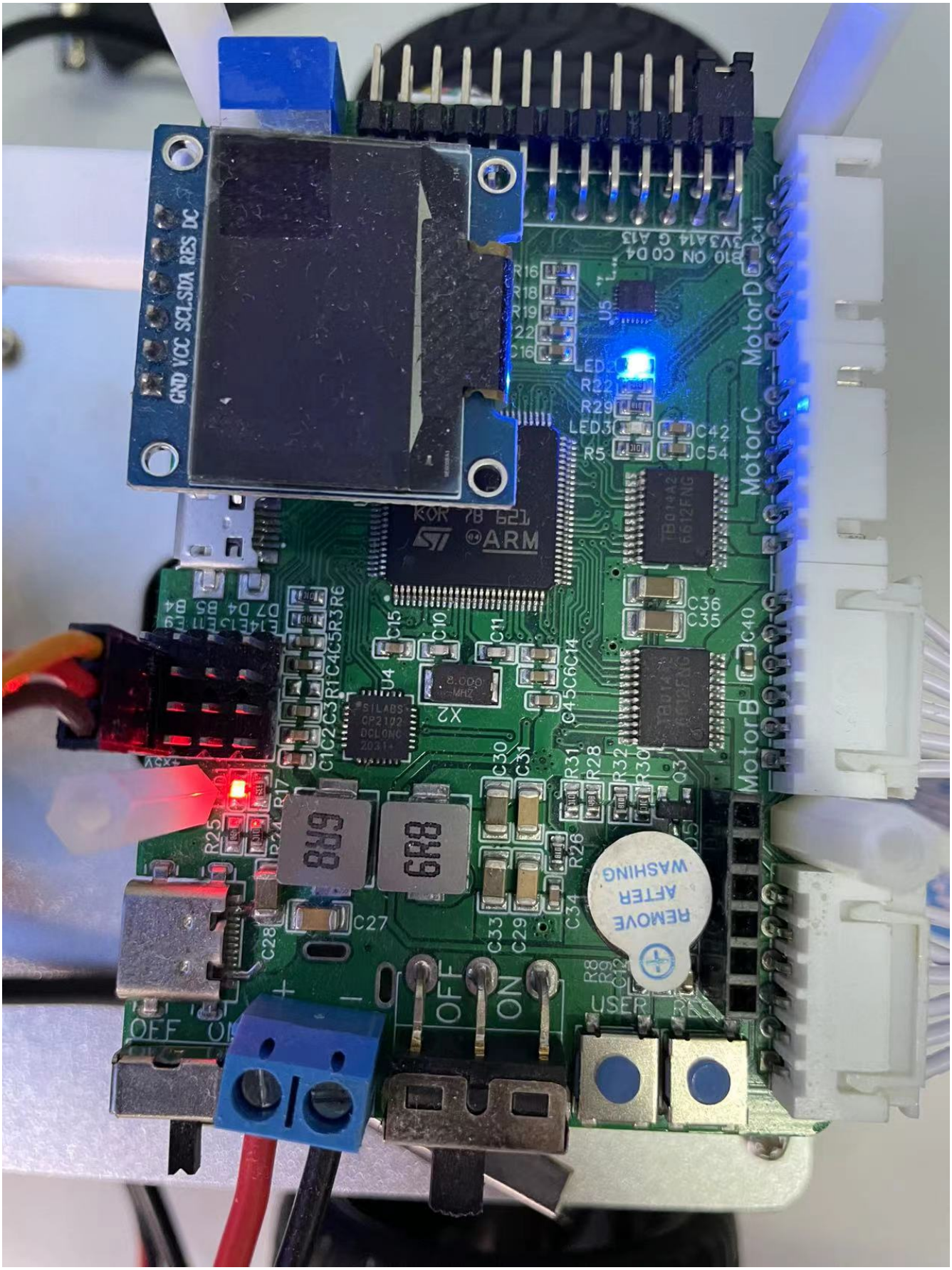
所使用的单片机：

本工程所使用的单片机为以STM32F103VET6：

有的童鞋可能对这个命名方式很感兴趣，在这里可以了解一下：

—	STM32	F	103	V	E	T	6
家族	STM32 表示 32bit 的 MCU						
产品类型	F 表示基础型						
具体特性	基础型						
引脚数目	V 表示 100pin，其他常用的为 C 表示 48，R 表示 64，Z 表示 144，Z 表示 144，B 表示 208，N 表示 216						
FLASH 大小	E 表示 512KB，其他常用的为 C 表示 256，E 表示 512，I 表示 2048						
封装	T 表示 QFP 封装，这个是最常用的封装						
温度	6 表示温度等级为 A：-40~85°						

本工程使用的下位机开发板如图所示



开发环境的搭建：

- *安装DK*:执行 jdk-8u181-windows-x64.exe,一路下一步,没有下一步就点击finish

- *STM32芯片引脚配置工具*：解压en.stm32cubemx_v5.4.0.zip并执行SetupSTM32CubeMX-5.4.0.exe，点击下一步下一步就可以完成安装
- *交叉编译工具*：解压arm的交叉编译工具：解压gcc-arm-none-eabi-9-2019-q4-major-win32.zip 将它的根目录和bin路径配置到系统的Path中
- *烧录工具*：解压FlyMcu.zip 可以直接使用.

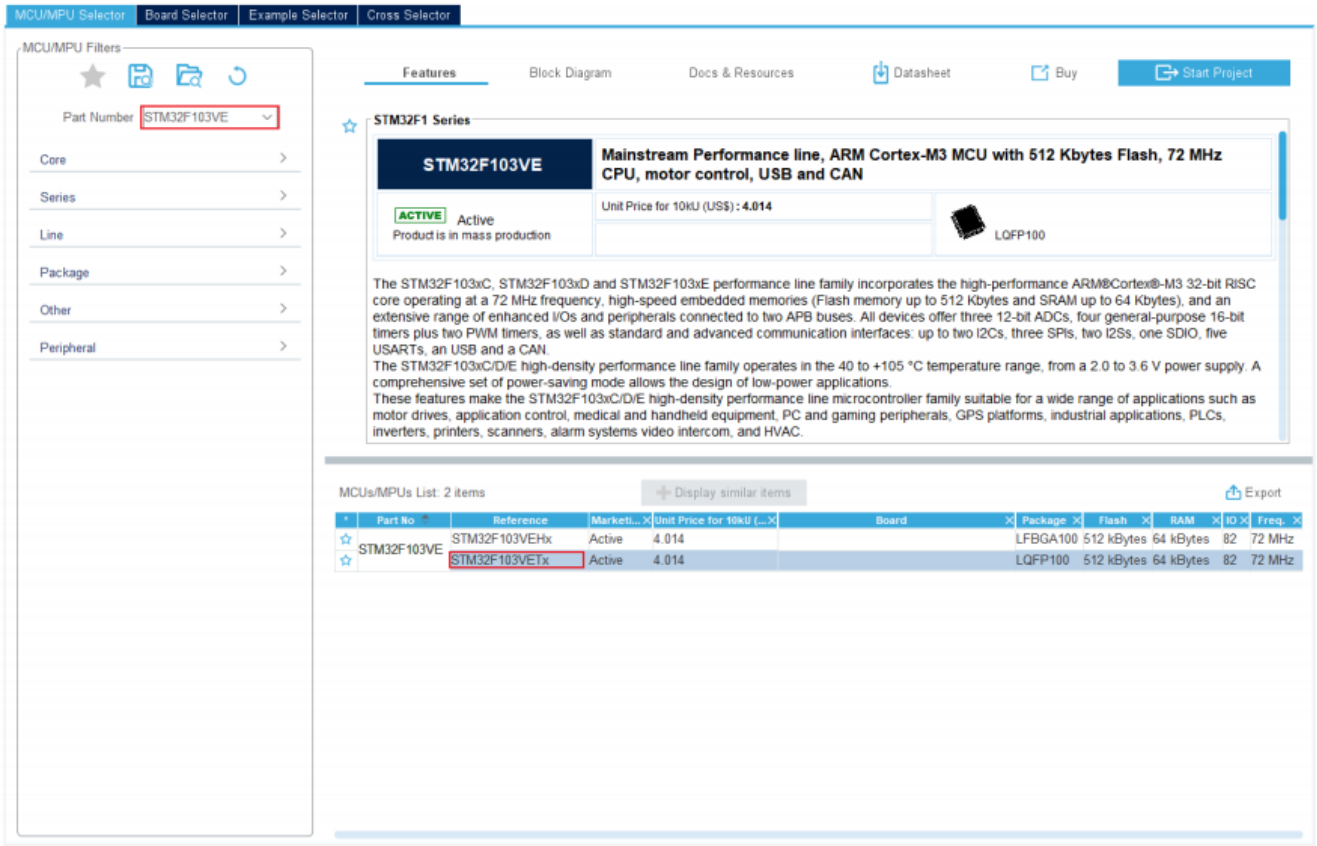
上面的安装工作完成之后，我们就可以进入STM32开发的工作中啦！

项目创建：

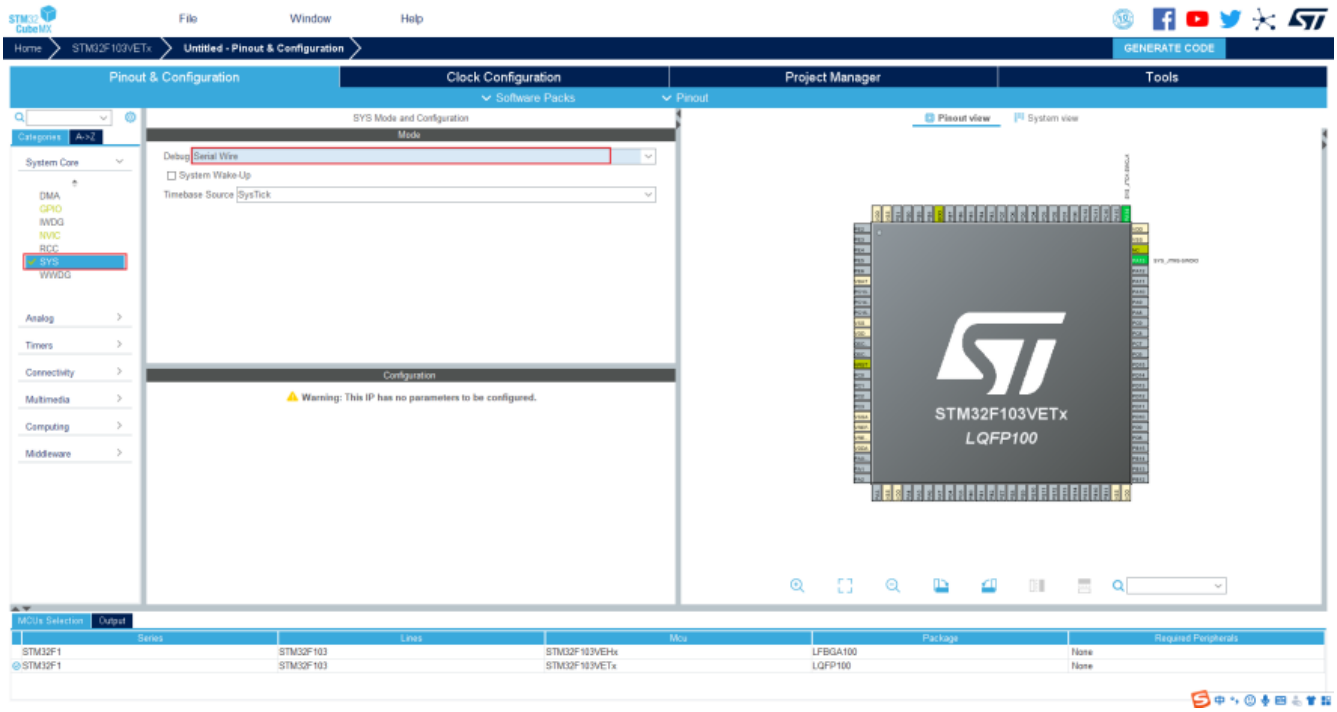
可以使用STM32CubeMX创建工程

选择芯片：

前面也说到了，该工程使用的单片机为STM32F103VET6,在搜索栏搜STM32F103VETx双击创建

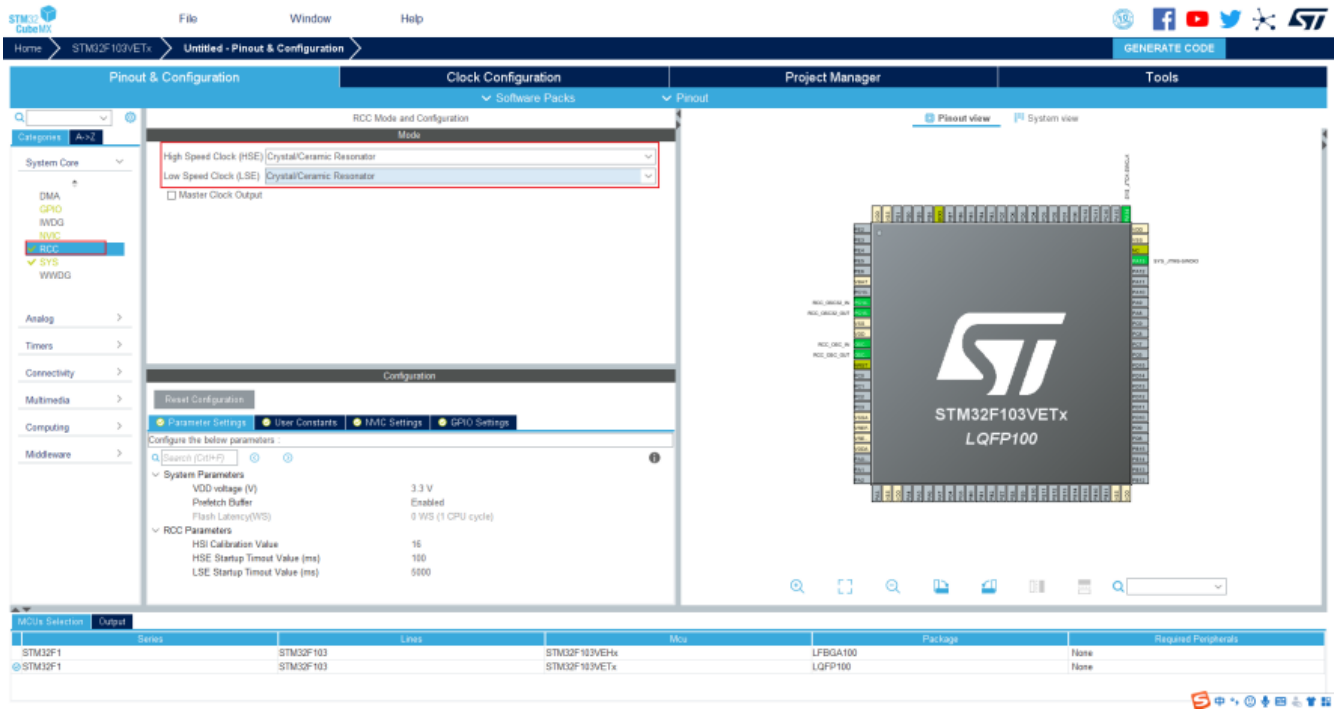


打开串口调试：

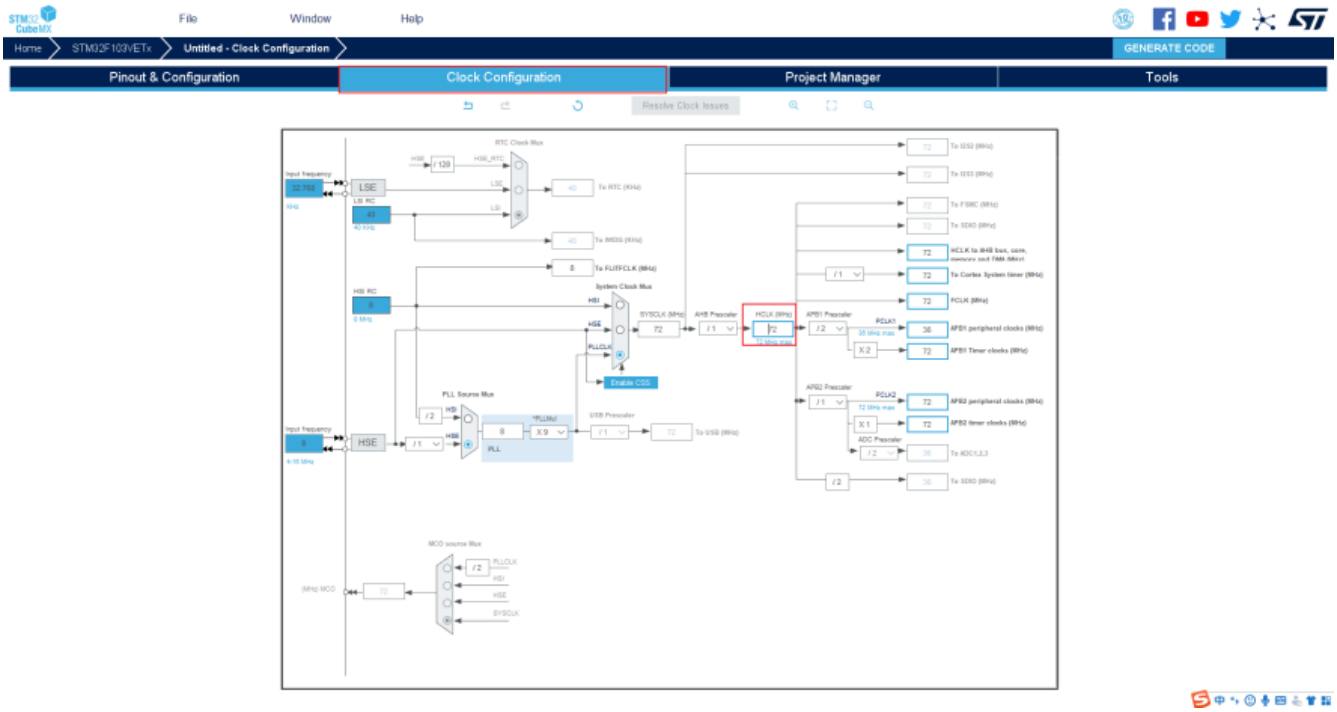


配置时钟：

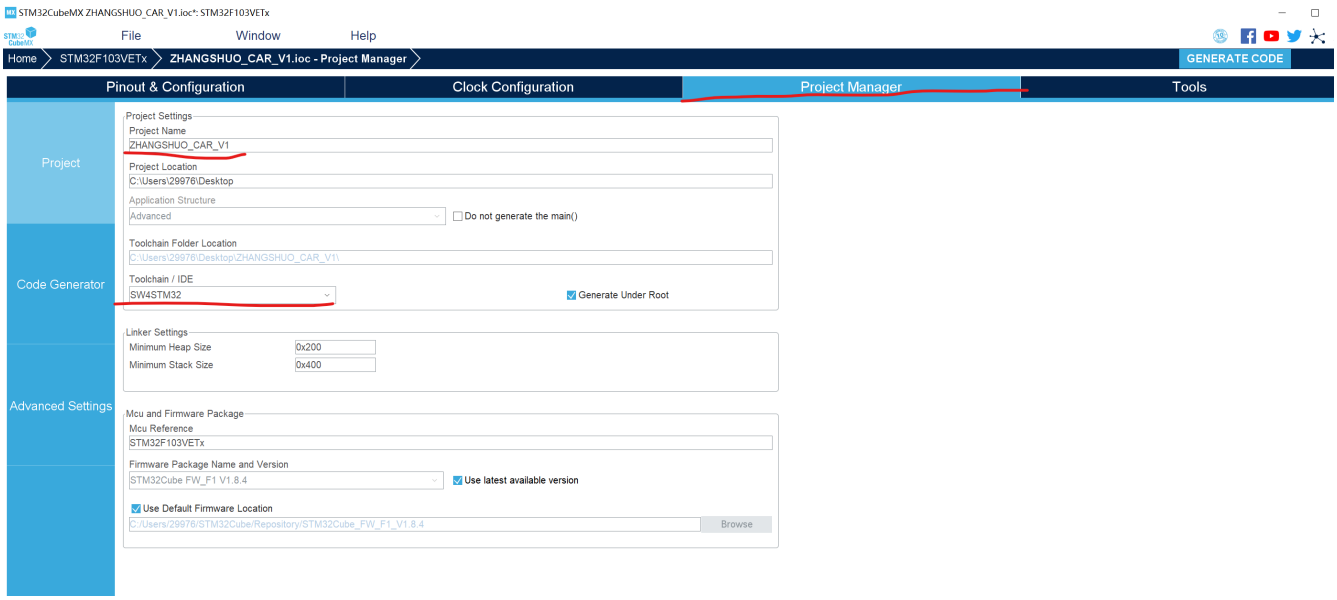
选择高速时钟

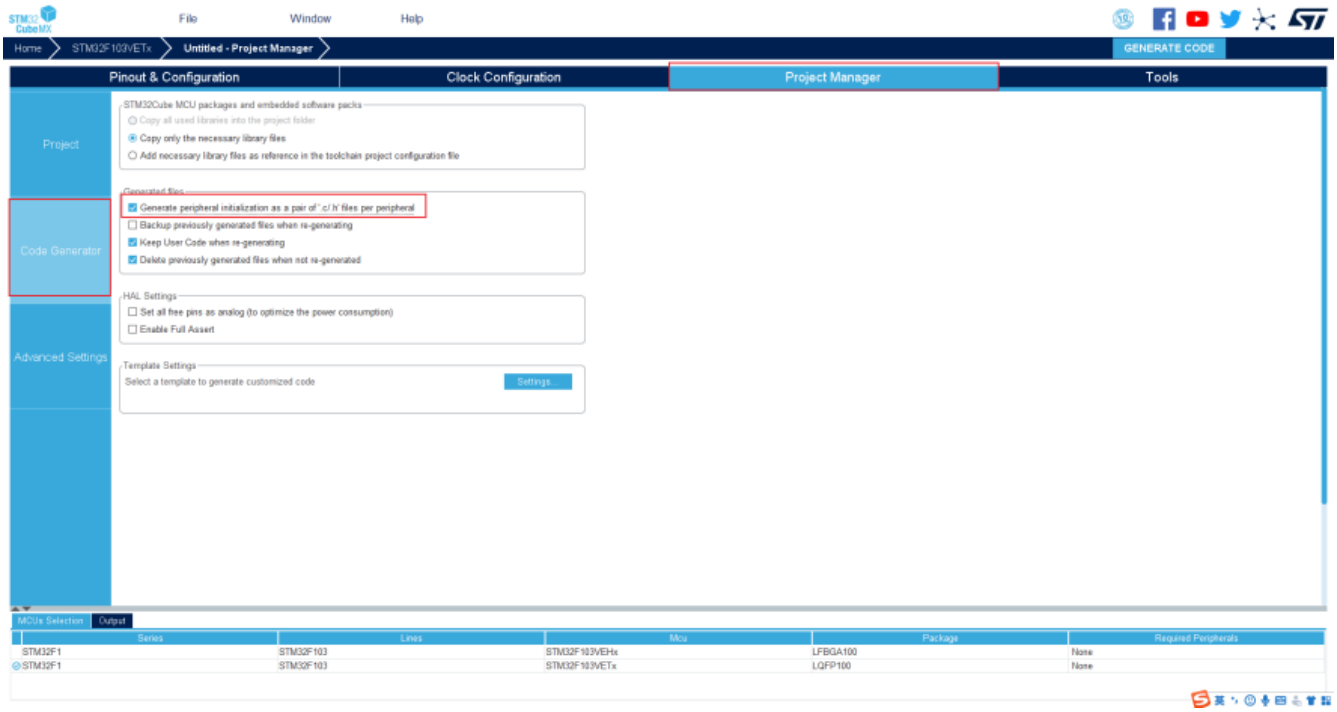


72Mhz指的是1秒钟执行72 000 000次

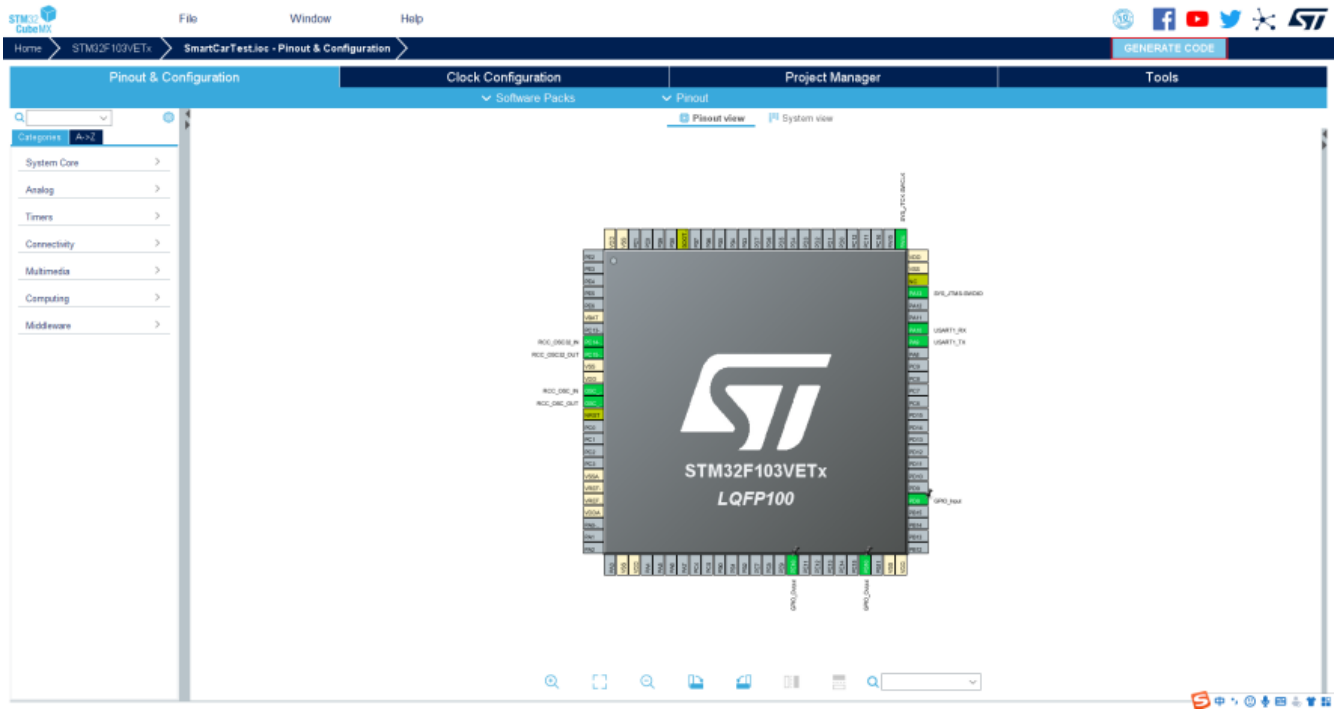

File
Window
Help
Home
STM32F103VETx
Untitled - Clock Configuration
GENERATE CODE
Pinout & Configuration
Clock Configuration
Project Manager
Tools
Resolve Clock Issues
LSE
HSI RC
HSE
PLL Source Mux
System Clock Mux
SYSCLK (MHz)
AHB Prescaler
HCLK (MHz)
APB1 Prescaler
APB2 Prescaler
PLLCLK
Enable CSS
USB Prescaler
To USB (MHz)
MCO source Mux

工程配置：


STM32CubeMX ZHANGSHUO_CAR_V1.ioc*: STM32F103VETx
File
Window
Help
Home
STM32F103VETx
ZHANGSHUO_CAR_V1.ioc - Project Manager
GENERATE CODE
Pinout & Configuration
Clock Configuration
Project Manager
Tools
Project
Code Generator
Advanced Settings
Project Settings
Project Name
ZHANGSHUO_CAR_V1
Project Location
C:\Users\29976\Desktop
Application Structure
Advanced
Do not generate the main()
Toolchain Folder Location
C:\Users\29976\Desktop\ZHANGSHUO_CAR_V1\
Toolchain / IDE
SW4STM32
Generate Under Root
Linker Settings
Minimum Heap Size
0x200
Minimum Stack Size
0x400
Mcu and Firmware Package
Mcu Reference
STM32F103VETx
Firmware Package Name and Version
STM32Cube FW_F1 V1.8.4
Use latest available version
Use Default Firmware Location
C:/Users/29976/STM32Cube/Repository/STM32Cube_FW_F1_V1.8.4
Browse



生成代码

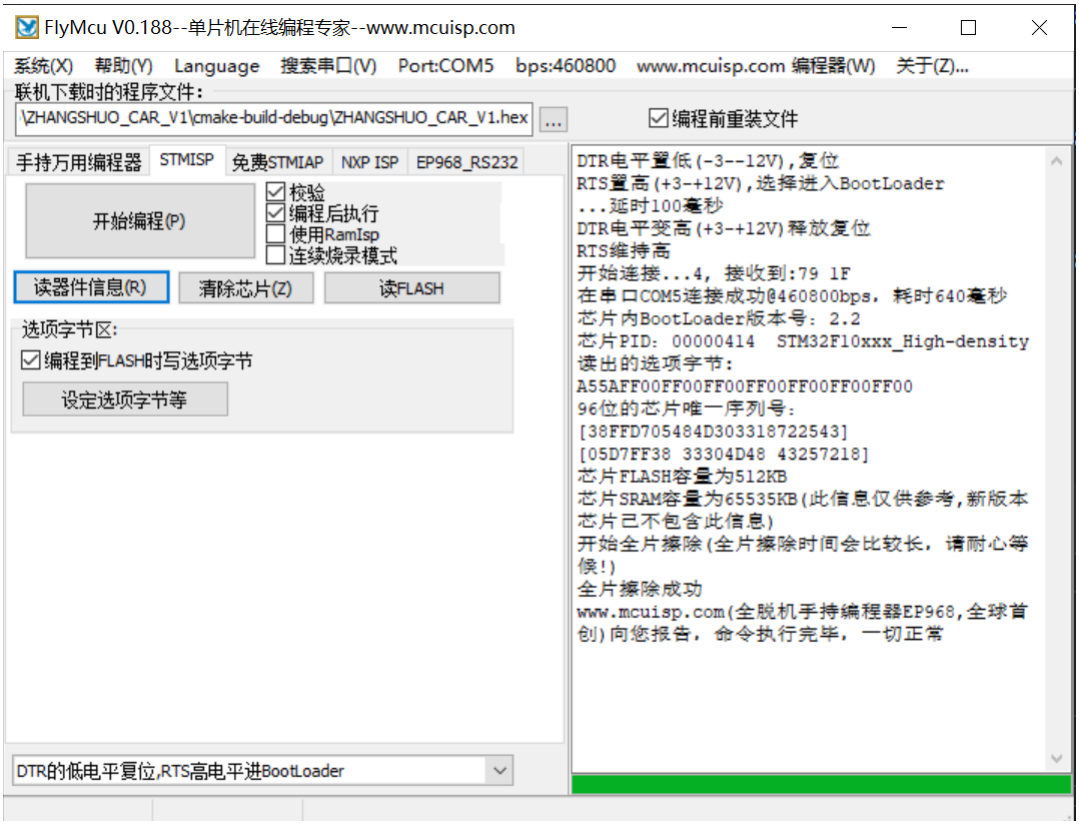


到此为止一个最基本的工程创建就完成了！

敲项目

用代码编辑工具（vs,clion,...都可）就可以愉快的玩耍了！

烧录程序：

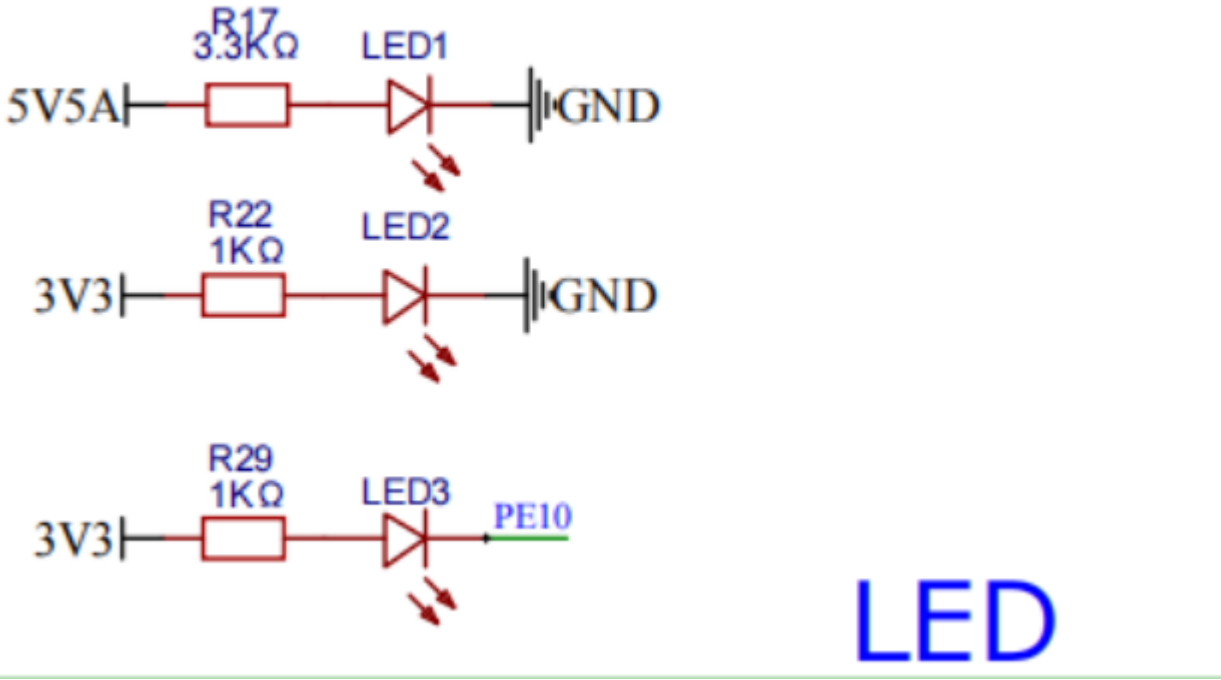


- port:选择连接STM32的串口，可在电脑设备管理器中查询
- bps:可以调最高，加快烧写速度
- 联机下载的程序文件：stm32工程生成的hex文件
- 最下面选择：DTR的电平复位，RTS高电平进BootLoader

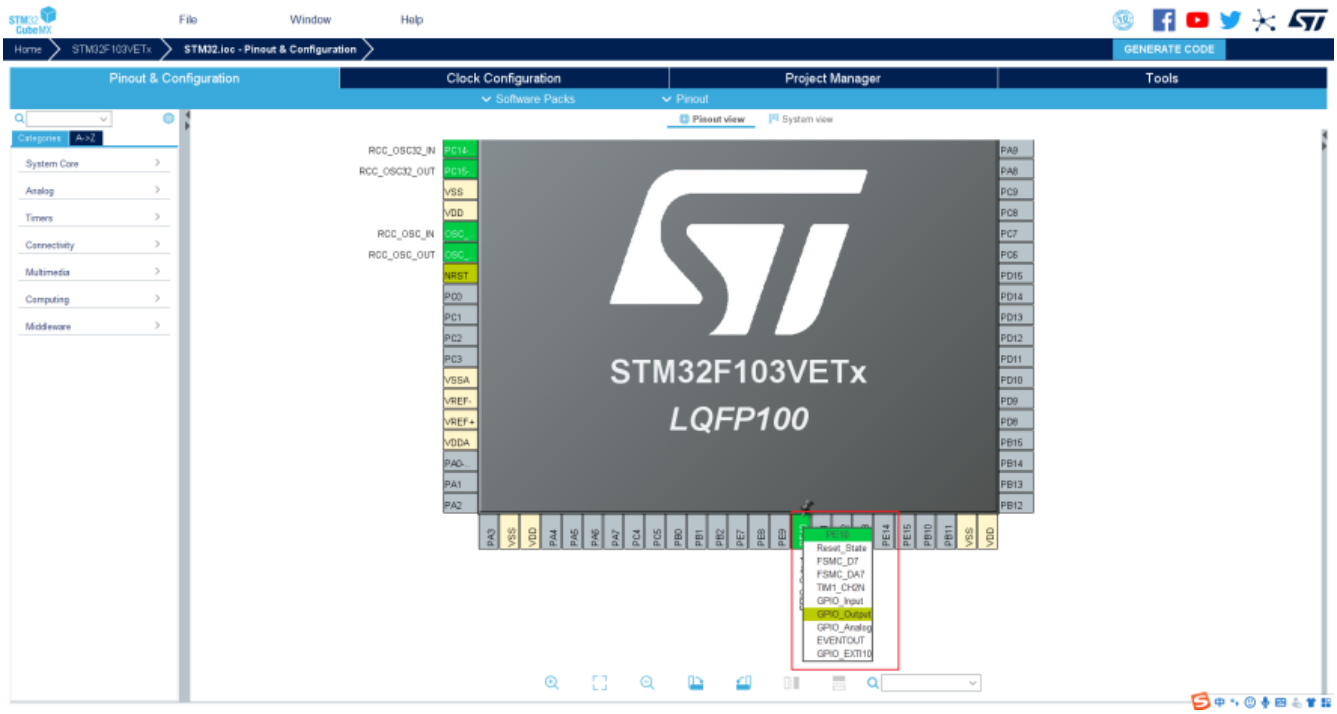
以上就是开发下位机的一个大概流程，后面的各个模块驱动由下面给出：

功能编写

LED小灯的控制



根据原理图可以知道LED小灯由PE10引脚控制，在STM32CubeMx中将PE10引脚设置为输出



按照电路图,LED的一端已经连接了高电平, 那么PE10这一端我们需要给它低电平,才能让灯亮起来

```
HAL_GPIO_WritePin(GPIOA,GPIO_PIN_1,GPIO_PIN_RESET);
```

如果想让灯灭的话, 我们需要给PE10这一端设定为高电平

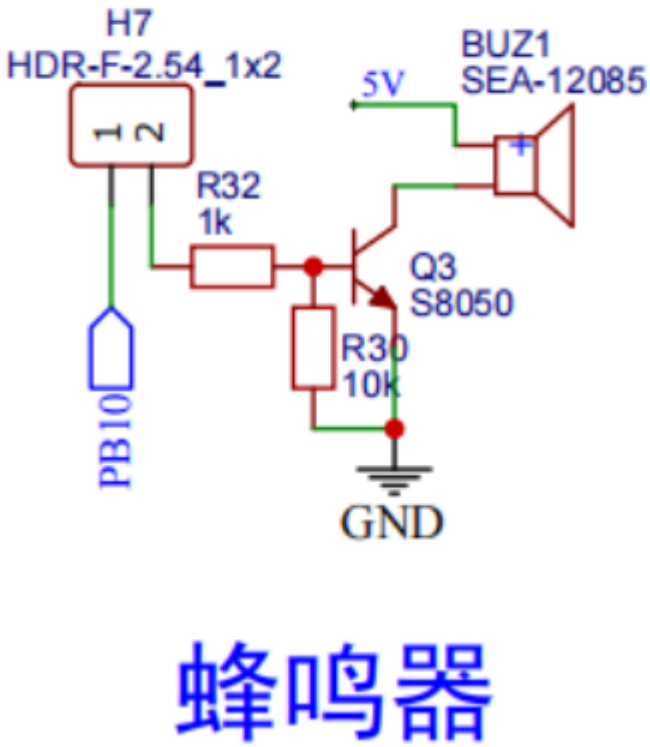
```
HAL_GPIO_WritePin(GPIOA,GPIO_PIN_1,GPIO_PIN_SET);
```

如果想让LED间隔一定时间闪烁的话,我们可以采用如下的方式:

```
    // 闪烁led灯
    HAL_GPIO_WritePin(GPIOA,GPIO_PIN_1,GPIO_PIN_SET);
    HAL_Delay(500);
    HAL_GPIO_WritePin(GPIOA,GPIO_PIN_1,GPIO_PIN_RESET);
    HAL_Delay(500);
```

这段程序可以在main函数中查看到

蜂鸣器的控制



蜂鸣器

蜂鸣器的控制和LED一致

```
            /*****************************响_蜂鸣器*********************************/
            HAL_GPIO_WritePin(GPIOB,GPIO_PIN_10,GPIO_PIN_SET);
            //延迟两秒钟
            HAL_Delay(1000);
            /*停_蜂鸣器*/
            HAL_GPIO_WritePin(GPIOB,GPIO_PIN_10,GPIO_PIN_RESET);
            //延迟两秒钟
            HAL_Delay(1000);
```

上面的小灯和蜂鸣器的实验都是拿引脚作为输出来使用，接下来来实现一下引脚作为输入的功能：

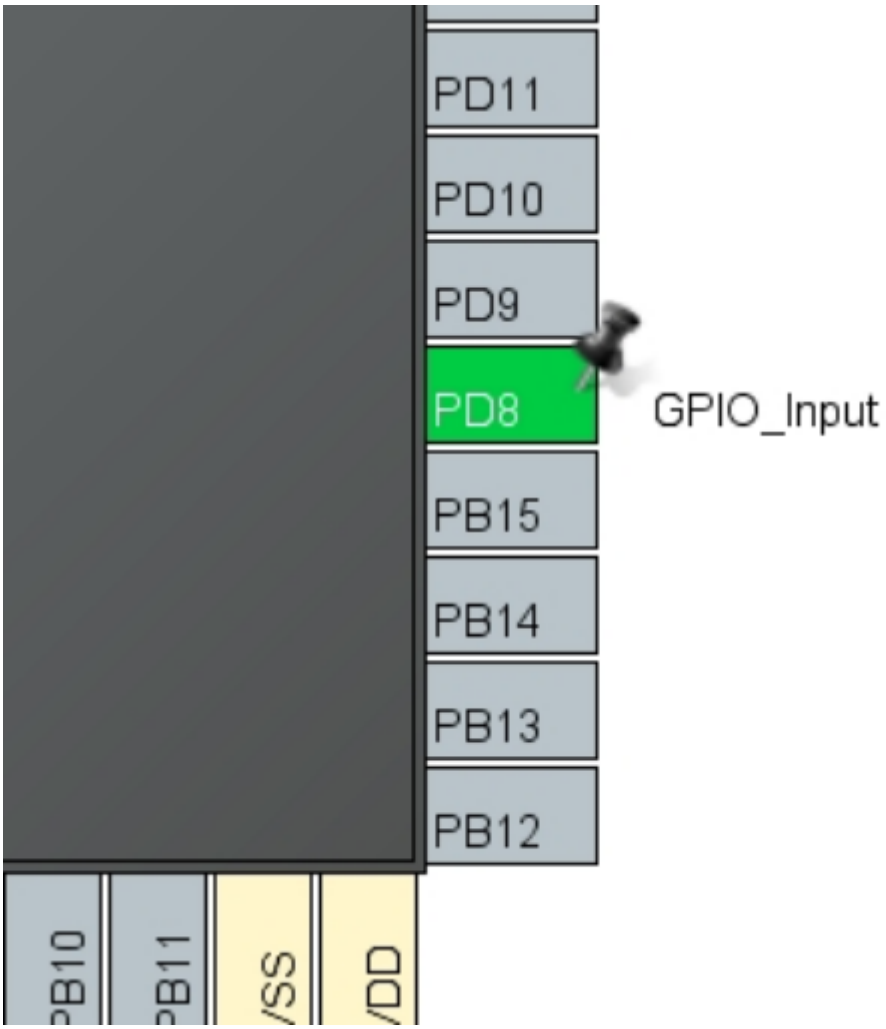
开关的使用

我们使用GPIO来读取开关状态，其实就是要读取IO口的电平变化

1.查看原理图：

2.配置STM32的IO口：

我们可以看到，我们将要使用8号引脚作为输入



3.在main中编写逻辑：

```
        /*************************开关按钮
*****************************************/
        //读取开关按钮状态
        GPIO_PinState state = HAL_GPIO_ReadPin(GPIOD, GPIO_PIN_8);

        if (state==GPIO_PIN_RESET){
            // 如果低电平，亮LED;
            HAL_GPIO_WritePin(GPIOE, GPIO_PIN_10, GPIO_PIN_RESET);
        } else if (state==GPIO_PIN_SET){
            // 如果高电平，灭LED;
            HAL_GPIO_WritePin(GPIOE, GPIO_PIN_10, GPIO_PIN_SET);
```

使得按下开关按钮，灯灭；松开开关按钮，灯亮

知识补充：通讯的基本概念

在计算机设备与设备之间或集成电路之间常常需要进行数据传输

串行通讯与并行通讯

按数据传送的方式，通讯可以分为：

- 串行通讯：串行通讯是指设备之间通过少量数据信号线（一般8根以下），地线以及控制信号线，按数据位形式一位一位地传输数据的通讯方式。
- 并行通讯：而并行通讯一般是指使用 8、16、32及 64 根或更多的数据线进行传输的通讯方式。

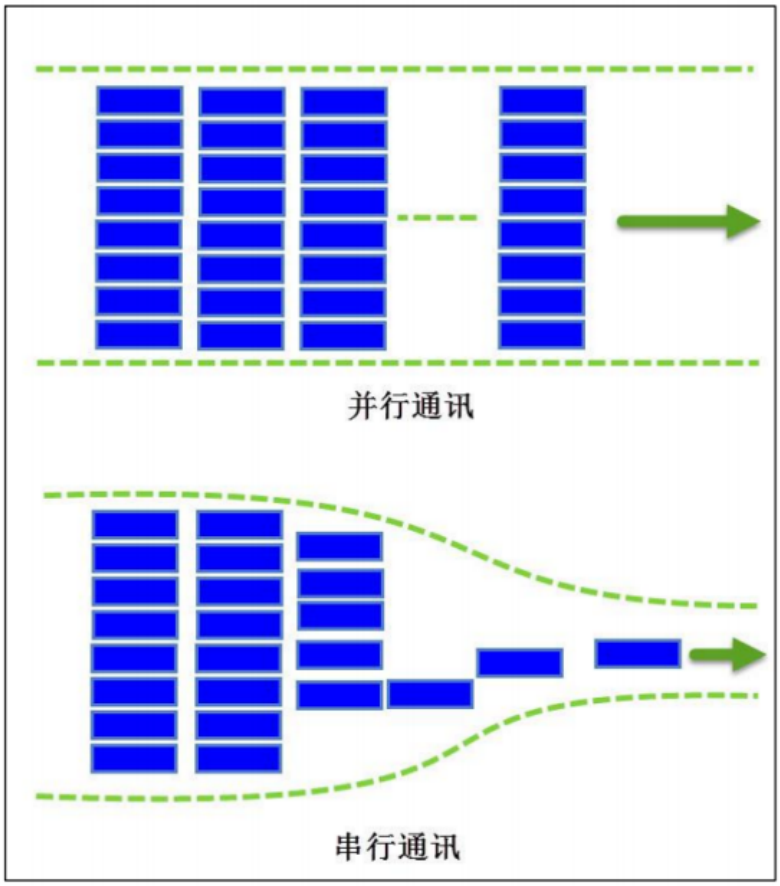


图 20-1 并行通讯与串行通讯的对比图

串行通讯和并行通讯的特性对比

特性	串行通讯	并行通讯
通讯距离	较远	较近
抗干扰能力	较强	较弱
传输速度	较慢	较快
成本	较低	较高

全双工、半双工及单工通讯

根据数据通讯的方向，通讯又分为全双工、半双工及单工通讯，它们主要 以信道的方向来区分

通讯方式	说明
全双工	同一时刻，两个设备之间可以同时收发数据
半双工	两个设备之间可以收发数据，但是不能在同一时刻进行
单工	在任何时刻都只能进行一个方向的通讯，即以恶搞固定为发送设备，另一个固定为接收设备（遥控器这种）

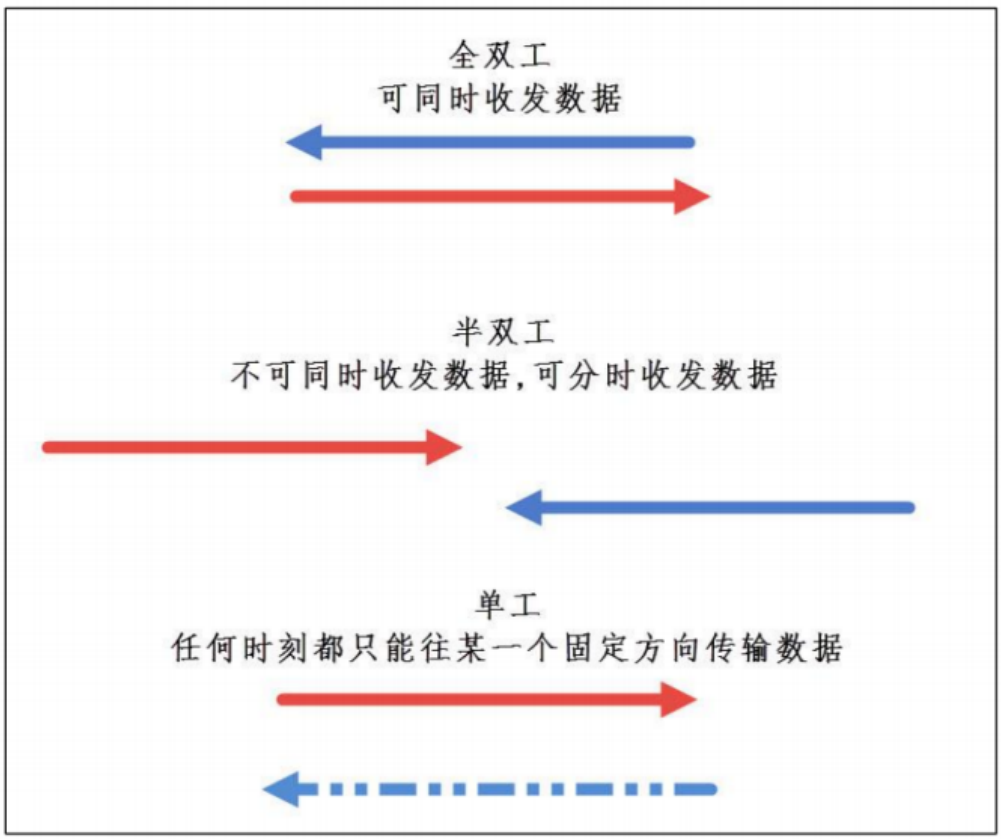


图 20-2 全双工、半双工及单工通讯

同步通信和异步通信

根据通讯的数据同步方式，又分为同步和异步两种，可以根据通讯过程中是否有使用到时钟信号进行简单的区分。

简单的说就是主机在相互通信时发送数据的频率是否一样。

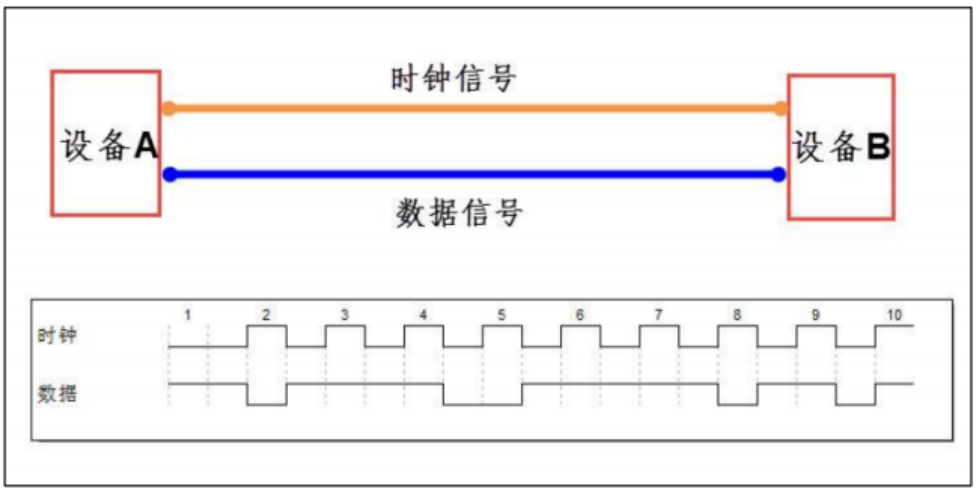


图 20-3 同步通讯

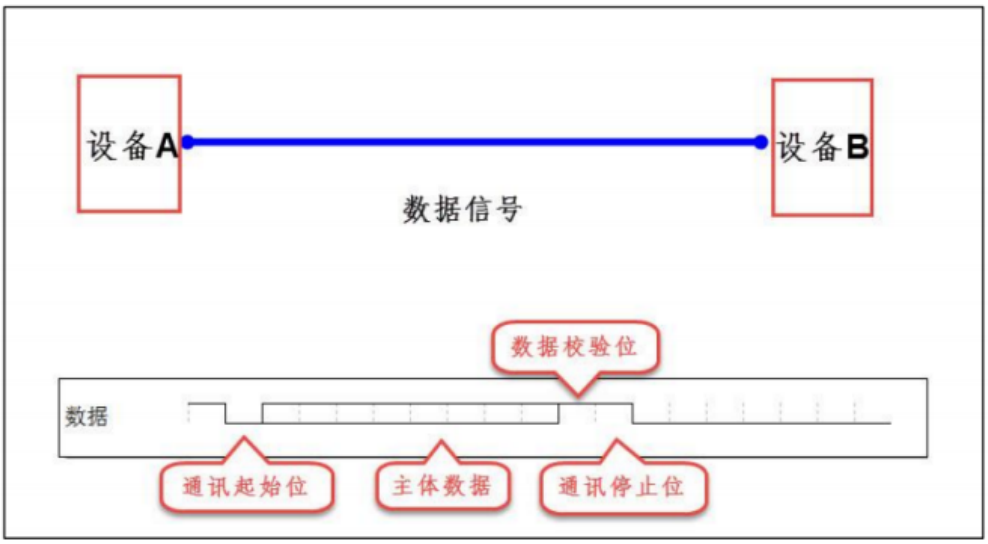


图 20-4 某种异步通讯

通讯速率

衡量通讯性能的一个非常重要的参数就是通讯速率，通常以比特率(Bitrate) 来表示，即每秒钟传输的二进制位数，单位为比特每秒(bit/s)。

举个例子：

例如比特率为115200，也就是一秒钟可以传递115200bit。

1byte = 8 bit

$115200bit = \frac{115200}{8}byte = 14400byte$

1kb = 1024byte

$115200bit = 115200/8/1024kb = 14.0625kb$

容易与比特率混淆的概念是“波特率”(Baudrate)，它表示每秒钟传输了多少个码元。

啥是码元?

信息单位:

- 是1、否0 可以用1bit表示（如果码元只有两种情况的时候，也就是1个字节的时候，他和比特率是一样的概念）
- 比如：4，5，6，7 需要用00，01，10，11表示

而码元是通讯信号调制的概念，通讯中常用时间间隔相同的符号来表示一个二进制数字，这样的信号称为码元。如常见的通讯传输中，用 0V 表示数字 0，5V 表示数字 1，那么一个码元可以表示两种状态 0 和 1，所以一个码元等于一个二进制比特位，此时波特率的大小与比特率一致；如果在通讯传输中，有 0V、2V、4V 以及 6V分别表示二进制数 00、01、10、11，那么每个码元可以表示四种状态，即两个二进制比特位，所以码元数是二进制比特位数的一半，这个时候的波特率为比特率的一半。因为很多常见的通讯中一个码元都是表示两种状态，人们常常直接以波特率来表示比特率。

常见的波特率为4800、9600、115200 等。

USART串口通信

串口通信

串口通讯(Serial Communication)是一种设备间非常常用的串行通讯方式，因为它简单便捷，因此大部分电子设备 都支持该通讯方式，电子工程师在调试设备时也经常使用该通讯方式输出调试信息。对于通讯协议，可以分为物理层和协议层。

- **物理层**规定通讯系统中具有机械、电子功能部分的特性，确保原始数据在物理媒体的传输。就是硬件部分
  - 物理层规定我们用嘴巴还是用肢体来交流
- **协议层**主要规定通讯逻辑，统一收发双方的数据打包、解包标准。就是软件部分
  - 协议层则规定我们用中文还是英文来交流。

物理层

物理层常见的标准RS232（全双工）、RS485（半双工）、USB转串口(TTL)以及原生的串口到串口(TTL-TTL)

协议层

数据包组成

图 21-6 串口数据包的基本组成

举个例子（随便写的，只是示意），比如传两个数据:

hsdia shidoaidh这两个数据没有标识位我们不知道一个数据结束没有，下一个数据开始没有。

wwhsdiarr: 看见ww是开始，看见rr结束。

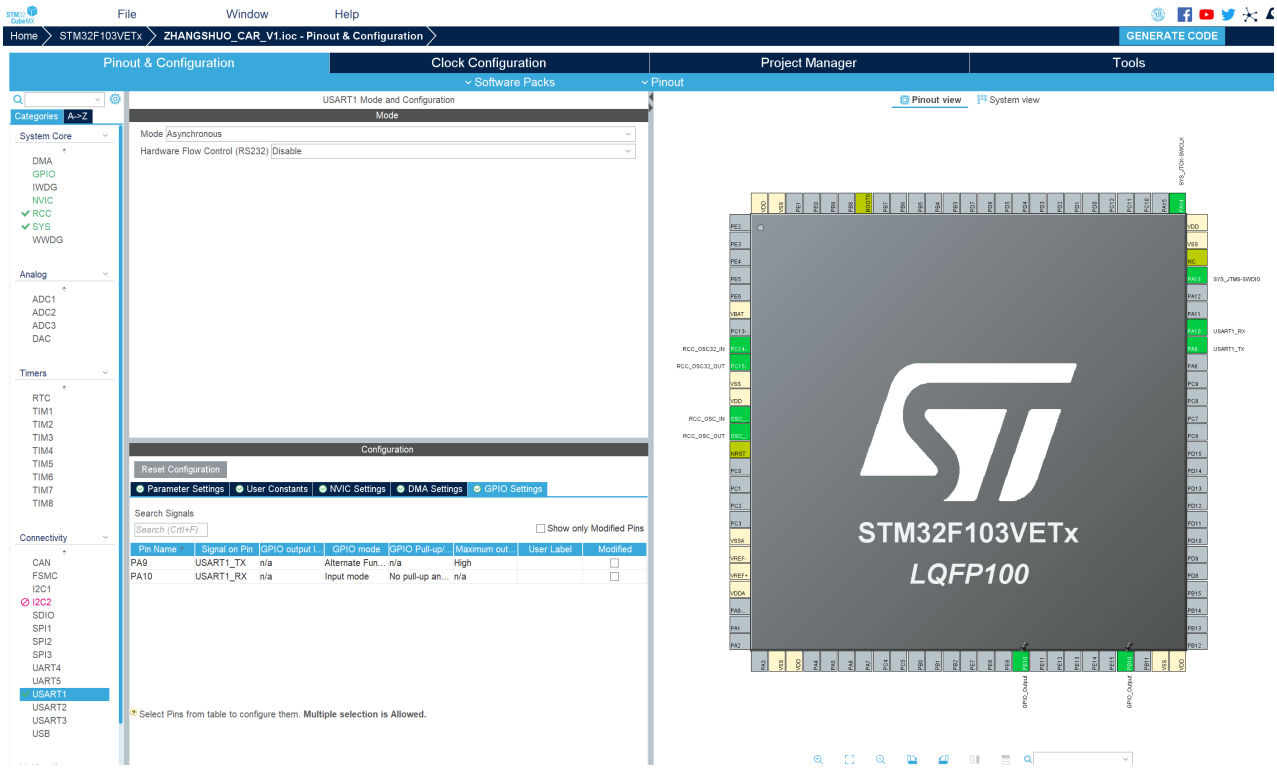
USART 简介

通用同步异步收发器(Universal Synchronous Asynchronous Receiver and Transmitter)是一个串行通信设备， 可以灵活地与外部设备进行全双工数据交换。有别于 USART 还有一个 UART(Universal Asynchronous Receiver and Transmitter)，它是在 USART 基础上裁剪掉了同步通信功能，只有异步通信。

USART 在 STM32 应用最多莫过于“打印”程序信息，一般在硬件设计时都会预留一个USART通信接口连接电脑，用于在调试程序是可以把一些调试信息“打印”在电脑端的串口调试助手工具上，从而了解程序运行是否正确、如果出错哪具体哪里出错等等。

USART通信配置

1. 连接CP2102串口
2. 在STM32CubeMX中打开USART通信：



3. **发送数据**

```
    char data[] = "hello world!\r\n";
    HAL_UART_Transmit(&huart1,data,sizeof(data),3000);
```

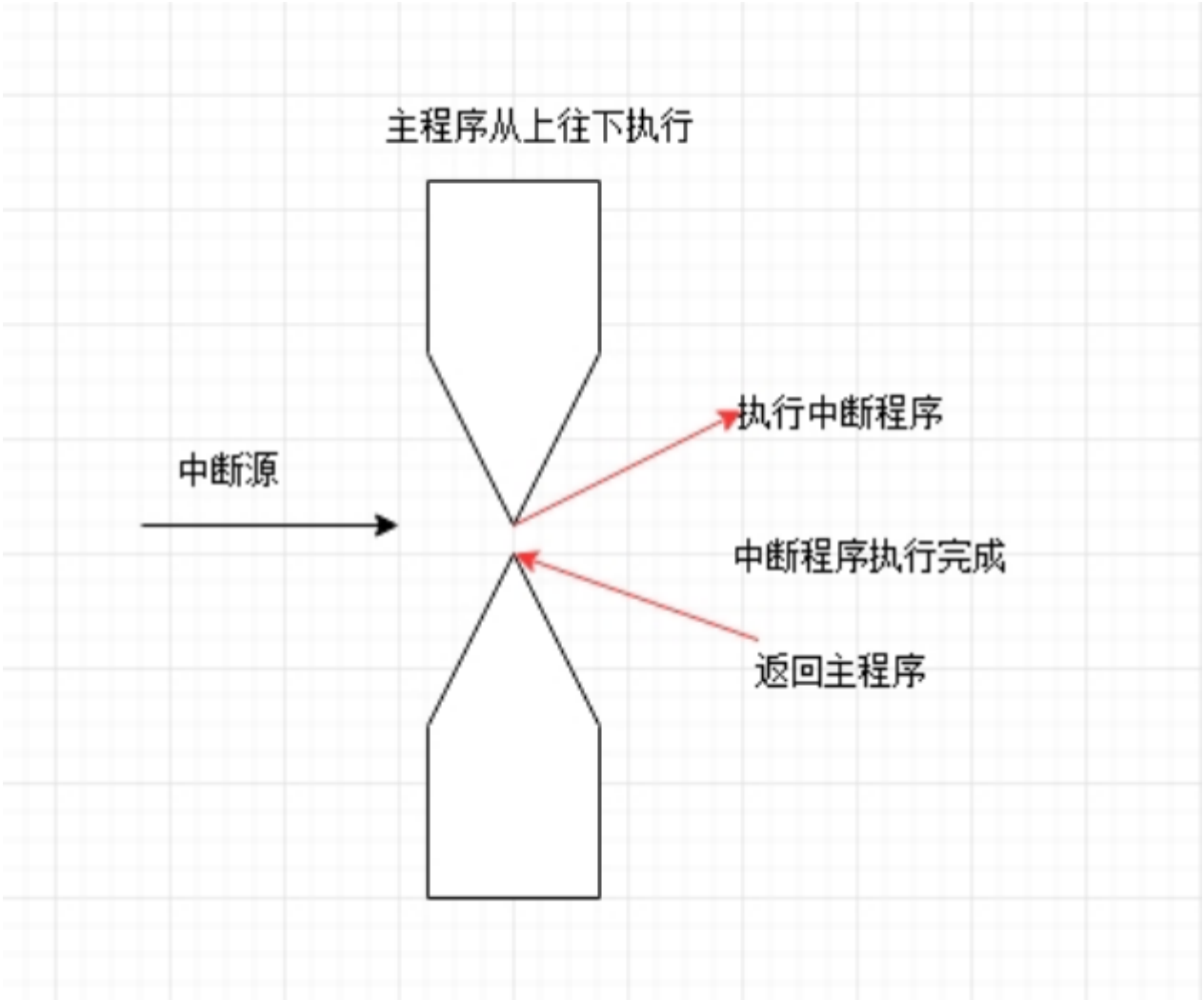
4. 接收数据

```
    char data[20] = {0};
    HAL_UART_Receive(&huart1,data,20,3000);
```

中断方式接收信息

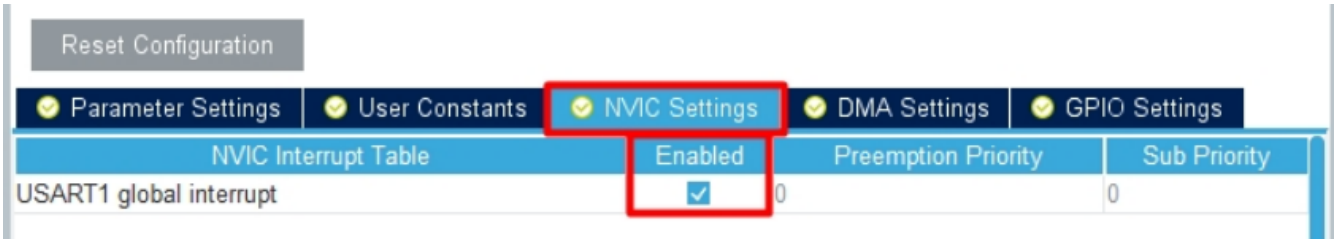
中断

中断，在单片机中占有非常重要的地位。代码默认地从上向执行，遇到条件或者其他语句，会按照指定的地方跳转。而在单片机执行代码的过程中，难免会有一些突发的情况需要处理， 这样就会打断当前的代码，待处理完突发情况之后，程序会回到被打断的地方继续执行。



中断接收串口消息

1. 在cubemx中开启中断（开启的是USART1的中断）



2. 编写代码：

   在主函数里，while外开启一个中断

```
1       HAL_UART_Receive_IT(&huart1,data,1);
```

   在主函数外定义回调函数：接收usart1串口数据

```c
    uint8_t data[20]={0};
    //回调函数：接收usart1串口数据
    void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart){
        HAL_UART_Transmit(&huart1,data,1,3000);
        /*再开启中断*/
        HAL_UART_Receive_IT(&huart1,data,1);
    }
```

NOTE:

- 在stm32f1xx_it.c中 `USART1_IRQHandler` 函数是处理回调的,进入实现 `HAL_UART_RxCpltCallback` 是回调
- `HAL_UART_TxHalfCpltCallback();` 一半数据发送完成时调用
- `HAL_UART_TxCpltCallback();` 数据完全发送完成后调用
- `HAL_UART_RxHalfCpltCallback();` 一般数据接收完成时调用
- `HAL_UART_RxCpltCallback();` 数据完全接受完成后调用
- `HAL_UART_ErrorCallback();` 传输出现错误时调用