一、通信

根据传输数据的方式，分为：并行通信、串行通信

`并行通信：`计算机与I/O设备之间，通过多条传输线，可以同时传输多个bit位的信号

`串行通信：`计算机与I/O设备之间，只有一根数据传输线，每次只能传输1bit数据

当多个数据传输时，只能按顺序依次1bit、1bit的传输

```
eg：
    串行通信的情况下，发送0x5c
        0101    1100
```

根据通信双方有无共同的时钟线，分为：同步通信、异步通信

`同步通信：`有时钟线，时钟线作为通信的同步信号

在时钟线为低跳变时，改变通信信号

在时钟线为高跳变时，采样通信信号

`异步通信：`没有时钟线

如果没有时钟线，通信双方怎么知道一个电平周期是多久？

即使不发送数据，数据线也会有一个电平状态，对方怎么知道是发送来的数据，还是默认电平呢？

协议，双方规定好传输方式

根据数据传输的方向，分为：单工通信、半双工通信、全双工通信

`单工通信：`只能发送数据或者接收数据

eg：收音机、广播

`半双工通信：`通信双方同一时间段内，只具有其中的一个功能，要么发送数据，要么接收数据

eg：对讲机

`全双工通信：`通信双方在同一时间段内，既可以发送数据，又可以接收数据

eg：手机

二、串口通信

利用串行接口进行通信

串口：是单片机中最常见、最简单的**串行数据**传输协议

串口通信只需要两根数据线就可以实现全双工通信

两根数据线：

```
Tx：发送数据端，用于给对方发送数据
Rx：接收数据端，用于接收对方发来的数据

设备a           设备b
Tx   --------     Rx
Rx   --------     Tx
```

单片机上使用到的串口：

```
UART：通用异步收发器
USART：通用同步/异步收发器
```

三、UART协议

作用：规定了串口发送和接收数据的方式，必须以**数据帧**为单位

一帧数据：

`1个起始位：`一个周期的低电平信号

串口数据线 Tx空闲时，永远保持高电平

`8~9个数据位：`通信的正文

具体时多少bit，需要双方协商

先传输 最低位（LSB）最后传输最高位（MSB）

`1个校验位：`表示是否需要校验

0 不需要校验

1 需要校验

校验方式：奇校验、偶校验

奇校验：用数据位最后1个bit来确保发送的数据中1的个数为奇数

偶校验：用数据位最后1个bit来确保发送的数据帧1的个数为偶数

`0.5~2个停止位：`持续的高电平

具体持续事件，有双方约定 0.5 / 1 / 1.5 / 2 个周期

Baudrate 波特率：UART的传输速率

决定了1 Frame 数据的传输周期

```
115200
38400
9600
...

bps：bits per second
```

综上：UART协议 => 帧格式 + 波特率

四、物理层

不同的电气标准串口，引脚个数不一样，但是数据线万变不离其宗

Rx Tx

我们本次使用的是`TTL Level UART`：电平串口

```
Tx        数据发送端
Rx        数据接收端
VCC       电源端口（+）
GND       电源端口（-）

通信双方的GND必须相同（连一起）          共地

设备a           设备b
Tx  --------    Rx
Rx  --------    Tx
```

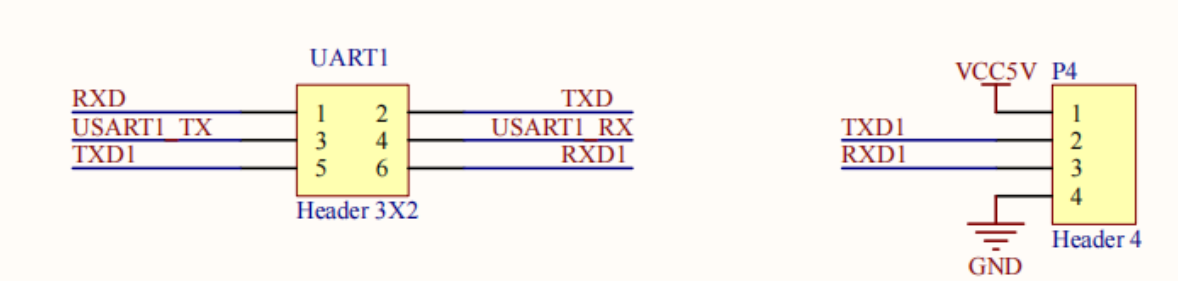
五、STM32F4xx 固件库控制串口

1. 串口配置步骤

根据参考手册可知 USART的 Tx/Rx引脚都是通过GPIO复用得到的

由原理图可知，GEC-M4 引出了3个串口

USART1：



1-3短接、2-4短接：作为烧录串口（串口经过CH340转USB和PC相连）

3-5短接、4-6短接：作为外部串口使用（串口可以通过杜邦线连接到模块）

所以在使用串口的时候，先根据用途，对跳线帽进行短接，再通过代码配置

2. 代码配置

step1：配置GPIO

```
（1）使能GPIO对应分组的时钟
（2）配置GPIO引脚的工作模式
    AF模式      => 复用模式
    Tx引脚对应的GPIO配置为 推挽输出
    Rx引脚对应的GPIO配置为 浮空输入
```

step2：配置GPIO复用功能

```
void GPIO_PinAFConfig(GPIO_TypeDef* GPIOx,
                      uint16_t GPIO_PinSource,
                      uint8_t GPIO_AF)
        @GPIOx              指定GPIO分组
        @GPIO_PinSource     指定GPIO引脚编号
        @GPIO_AF            指定复用成什么功能
                                GPIO_AF_USART1

eg:
    GPIO_PinAFConfig(GPIOA, GPIO_Pin_9, GPIO_AF_USART1);
    GPIO_PinAFConfig(GPIOA, GPIO_Pin_10, GPIO_AF_USART1);
```

step3：配置USART控制器

```
（1）使能时钟
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);
（2）配置USART控制器
    void USART_Init(USART_TypeDef* USARTx, USART_InitTypeDef* USART_InitStruct)
        @USARTx             指定具体的串口
                                USART1
                                ...
        @USART_InitStruct   结构体指针
                            指向的空间保存了串口的配置信息

typedef struct
```

```
{
  uint32_t USART_BaudRate;// 指定通信的波特率，要求通信双方的波特率一致

  uint16_t USART_WordLength;// 指定数据帧中数据位的位数
    USART_WordLength_8b
    USART_WordLength_9b

  uint16_t USART_StopBits;// 指定数据帧中停止位的位数
    USART_StopBits_1
    USART_StopBits_0_5
    USART_StopBits_2
    USART_StopBits_1_5

  uint16_t USART_Parity;// 指定有无校验位/校验的方式
    USART_Parity_No // 无校验
    USART_Parity_Even// 偶校验
    USART_Parity_Odd// 奇校验

  uint16_t USART_Mode;// 指定串口模式
    USART_Mode_Rx
    USART_Mode_Tx
        一般收发都需要，所以赋值时会写成如下：USART_Mode_Rx|USART_Mode_Tx

  uint16_t USART_HardwareFlowControl;// 指定流控
    一般不会需要硬件流控，直接复制成：USART_HardwareFlowControl_None
} USART_InitTypeDef;
```

step4：NVIC配置（如果使用中断）

```
USART1_IRQn

参考 《5.中断.pdf》
```

step5：使能串口

```
void USART_Cmd(USART_TypeDef* USARTx, FunctionalState NewState)
    @USARTx      指定串口编号
                     USART1
                     ...
    @NewState    ENABLE
                 DISABLE
```

3. 接收数据和发送数据

```
void USART_SendData(USART_TypeDef* USARTx, uint16_t Data);
// 功能：从指定的串口发送数据
    @USARTx      指定串口编号
    @Data        指定要发送的数据，1个字节

uint16_t USART_ReceiveData(USART_TypeDef* USARTx);
// 功能：从指定的串口接收1个字节数据
    @USARTx      指定串口编号
                     USART1
                     ...
    @return      返回从串口中读取到的数据

tips：串口接收数据一般在串口中断中完成，需要配置串口中断
```