WTR2019夏STM32培训

STM32概述

一.什么是STM32

1.ST——意法半导体，是一家半导体公司。可能大家经常听说ARM，ARM与STM32公司的关系可以简述为：STM32板子的内核由ARM设计，外设例如GPIO IIC FLASH等由ST公司生产。

2.M——Microelectronics的缩写，即微控制器。

3.32——32bit的意思，表示这是一个32bit的微控制器。

二、STM32芯片分类

表格 4-2 STM8 和 STM32 分类

CPU 位数	内核	系列	描述
32	Cortex-M0	STM32-F0	入门级
		STM32-L0	低功耗
	Cortex-M3	**STM32-F1**	**基础型，主频 72M**
		STM32-F2	高性能
		STM32-L1	低功耗
	Cortex-M4	STM32-F3	混和信号
		STM32-F4	**高性能，主频 180M**
		STM32-L4	低功耗
	Cortex-M7	STM32-F7	高性能
8	超级版 6502	**STM8S**	**标准系列**
		STM8AF	标准系列的汽车应用
		STM8AL	低功耗的汽车应用
		STM8L	低功耗

三、STM32F103C8T6

命名规则：

1.STM32：ST公司开发的32位微控制器。

2.F：代表flash，ST公司只开发了F系列的产品。

3.103：代表增强型系列。

4.C：代表48pin，R表示64，V代表100，Z表示144。

5.8：代表主闪存存储器（flash）容量，8表示64k字节。

6.T：表示QFP封装方式，这个是最常用的封装。

7.6：表示工作温度，在-40~85°C。

USB接口。

boot选项：选择板子的启动模式，boot0为0时则为竹山村存储器启动方式，即flash启动方式，一般情况下我们用的都是flash启动，所以正常情况下不要改boot。

复位按键。

主控板。

石英晶振。

RTC晶振。

电源指示灯。

LED灯。

SWD调试接口。

HAL库编程

STM32编程主要有三种，分别为寄存器、标准库和HAL库。我们目前只涉及HAL库编程，另外两种方式有时间的话也可以去研究。

寄存器开发可以通过直接操作寄存器来实现功能，STM32的寄存器非常多，如果要用寄存器开发方式，需要不断地翻阅芯片的数据手册，会很麻烦。但它的优点是更底层，可以更容易理解本质。

标准库，挺好用的，但有个现实，ST官方不再更新标准固件库。

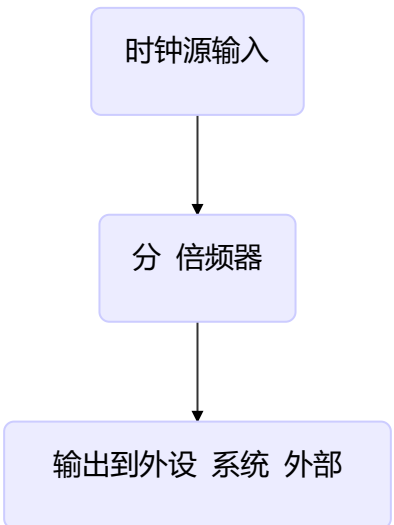
而HAL库，全称Hardware Abstraction Layer(硬件抽象层)是目前ST公司主推的开发方式。它比标准库更加抽象、简洁。它往下接底层硬件，网上给用户提供API。HAL库的使用大大节省了用户开发程序所花的时间。（~~虽然写程序还是很费时间~~）

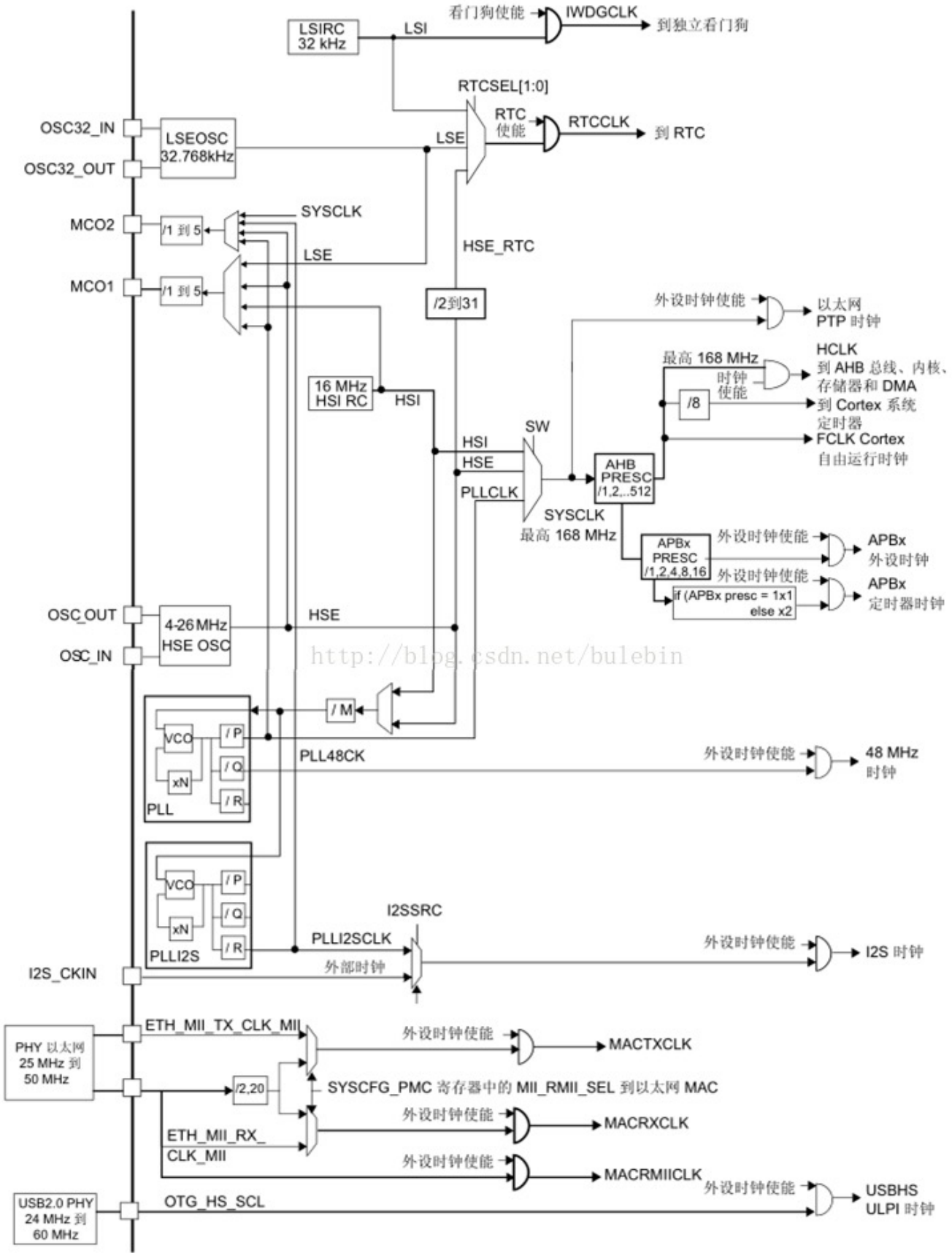
时钟树系统介绍

思考两个问题：

1.时钟信号从哪里来？

2.时钟怎么供给STM32系统工作：





STM32F4时钟树

问题一、时钟从哪里来：时钟源。

分别为LSI、HSI、LSE、HSE。

其中HSE来自于外部晶振，频率为4~26MHz，精度高，**一般就选用这个为时钟源。**

LSE来自于外部晶振，频率为32.768khz，一般用于RTC。

HSI由内部RC振荡器产生，精度差。

LSI位内部RC振荡器，用于独立看门狗时钟。

问题二、时钟怎么供给STM32的系统工作。

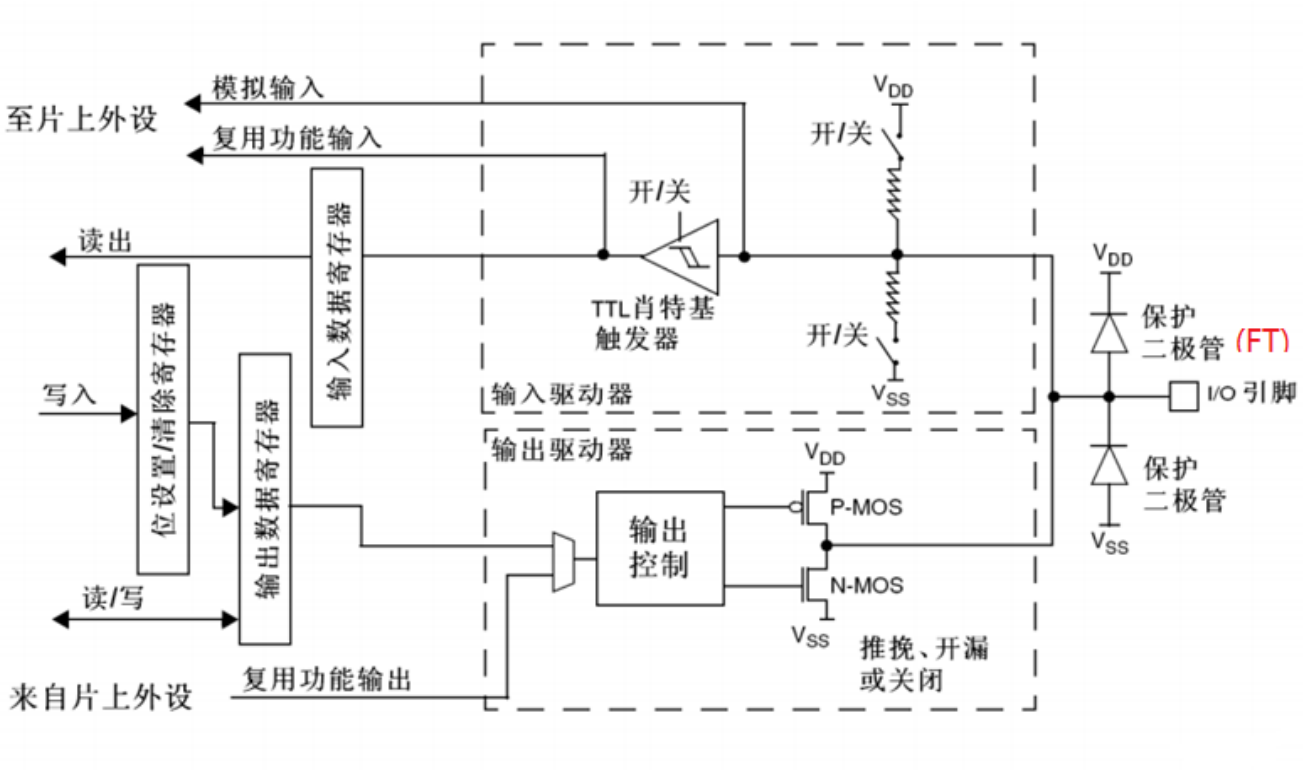
归纳为，看时钟信号流向，得到各个外设总线的时钟信号，使用外设时看外设挂载在哪一条总线上，使能这条总线时钟。

实验一 GPIO闪灯实验

STM32引脚说明

GPIO是通用输入/输出端口的简称，是STM32可控制的引脚。GPIO的引脚与外部硬件设备连接，可实现与外部通讯、控制外部硬件或者采集外部硬件数据的功能。我们用的STM32F103C8T6一共有3个通用输入输出口（GPIO）组，分别为GPIOA、GPIOB、GPIOC，每组有16个GPIO口，通常简称为PAx、PBx、PCx，x为0~15.

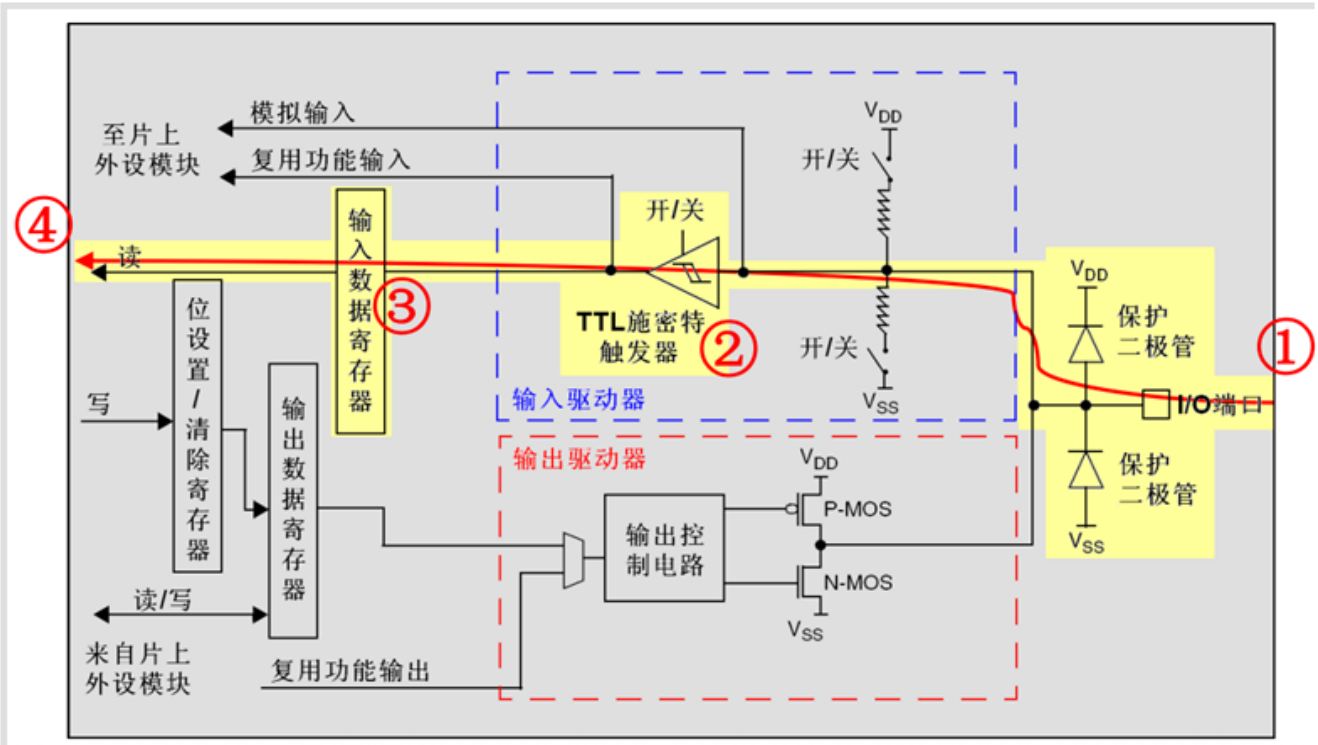
GPIO基本结构



GPIO支持4种输入模式（浮空输入、上拉输入、下拉输入、模拟输入）和4种输出模式（开漏输出、开漏复用输出、推挽输出、推挽复用输出）。同时，GPIO还支持三种最大翻转速度（2MHz、10MHz、50MHz）。

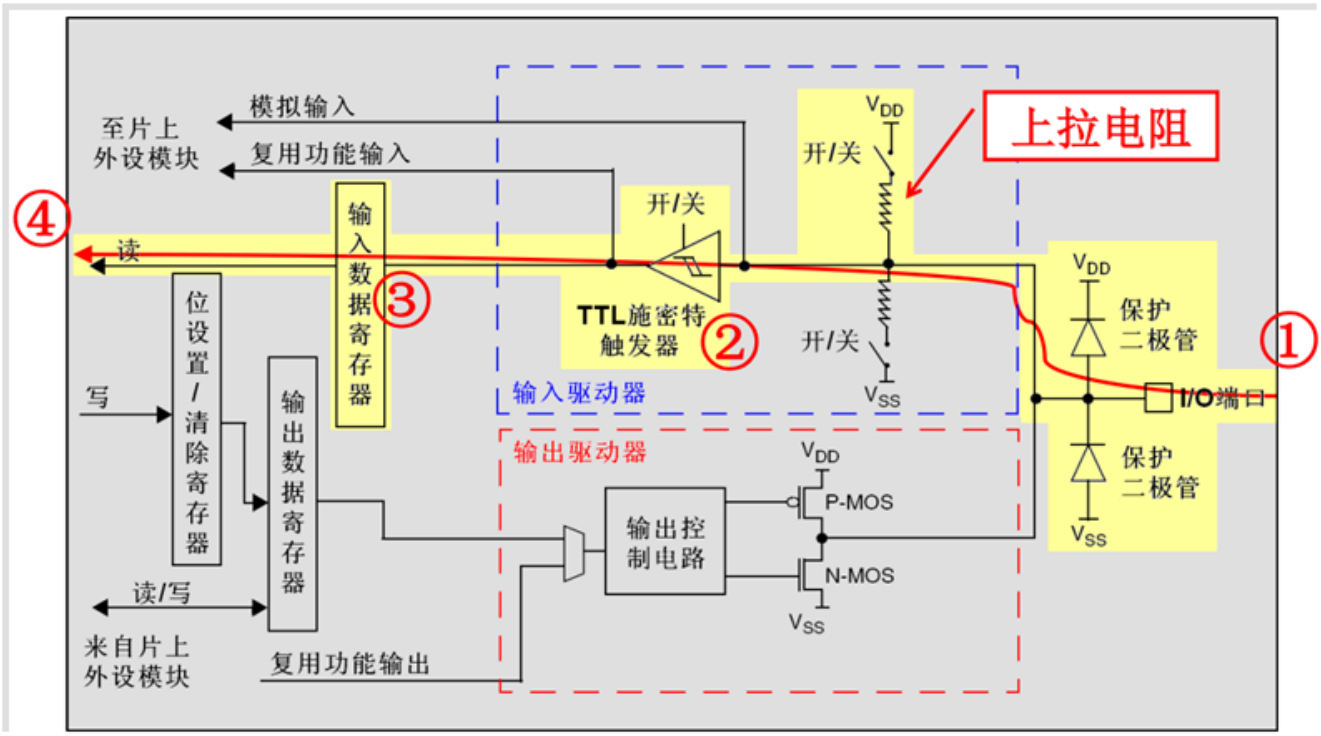
1. GPIO_Mode_AIN 模拟输入
2. GPIO_Mode_IN_FLOATING 浮空输入
3. GPIO_Mode_IPD 下拉输入
4. GPIO_Mode_IPU 上拉输入
5. GPIO_Mode_Out_OD 开漏输出
6. GPIO_Mode_Out_PP 推挽输出
7. GPIO_Mode_AF_OD 复用开漏输出
8. GPIO_Mode_AF_PP 复用推挽输出

浮空输入模式



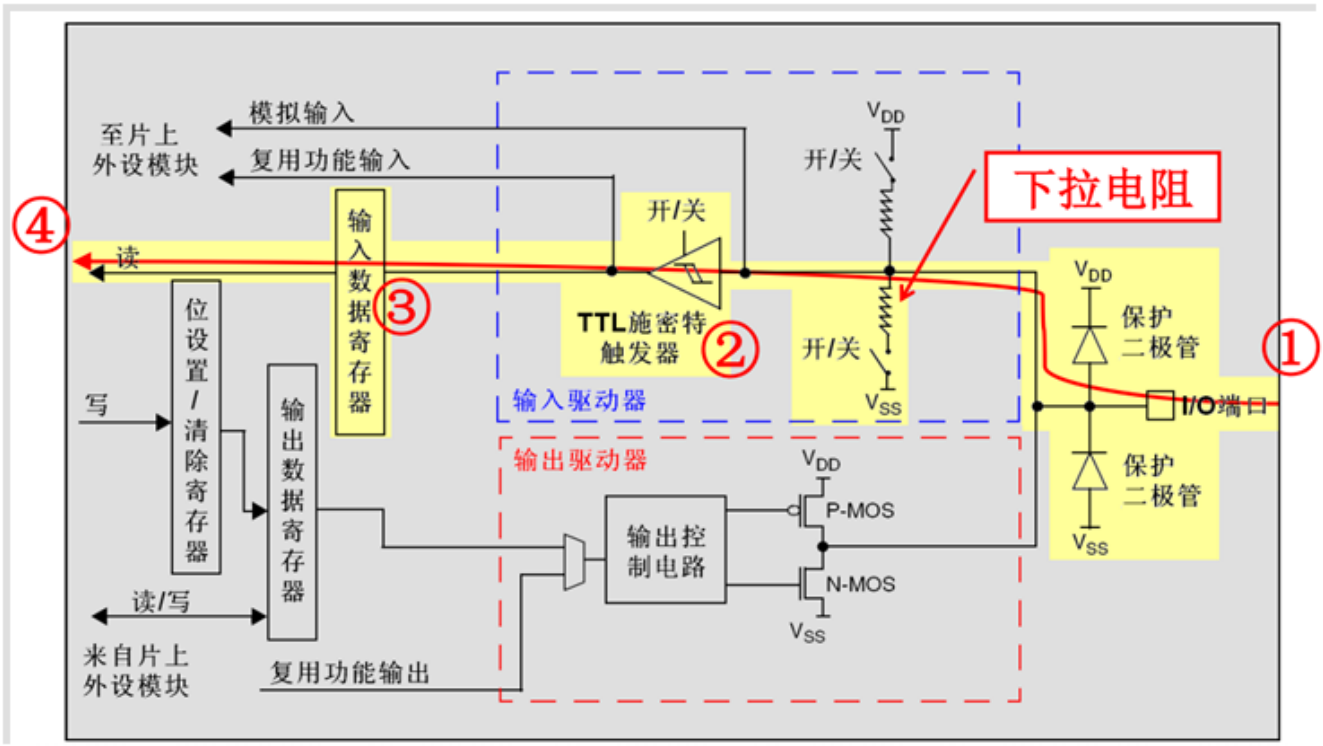
浮空输入模式下，I/O端口的电平信号直接进入输入数据寄存器。也就是说，I/O的电平状态是不确定的，完全由外部输入决定；如果在该引脚悬空（在无信号输入）的情况下，读取该端口的电平是不确定的。

上拉输入模式



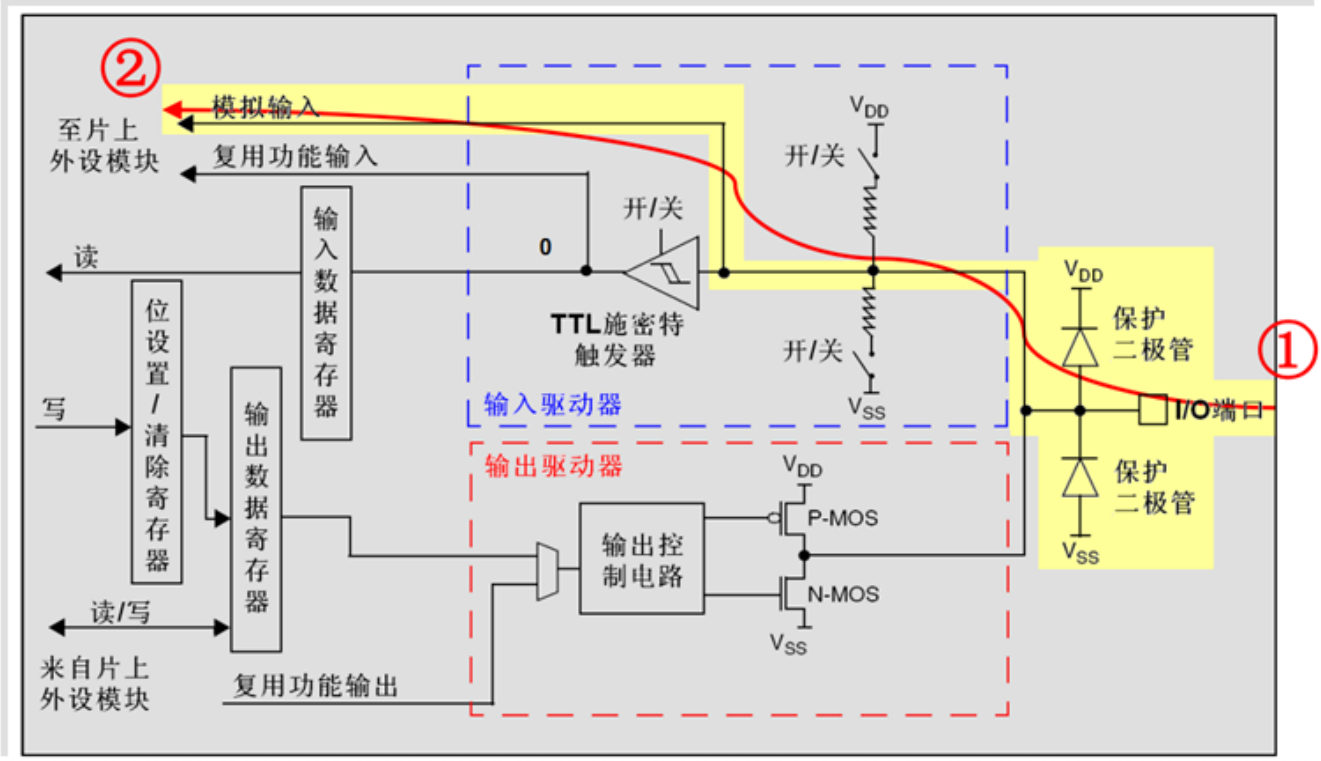
上拉输入模式下，I/O端口的电平信号直接进入输入数据寄存器。但是在I/O端口悬空（在无信号输入）的情况下，输入端的电平可以保持在高电平；并且在I/O端口输入为低电平的时候，输入端的电平也还是低电平。

下拉输入模式



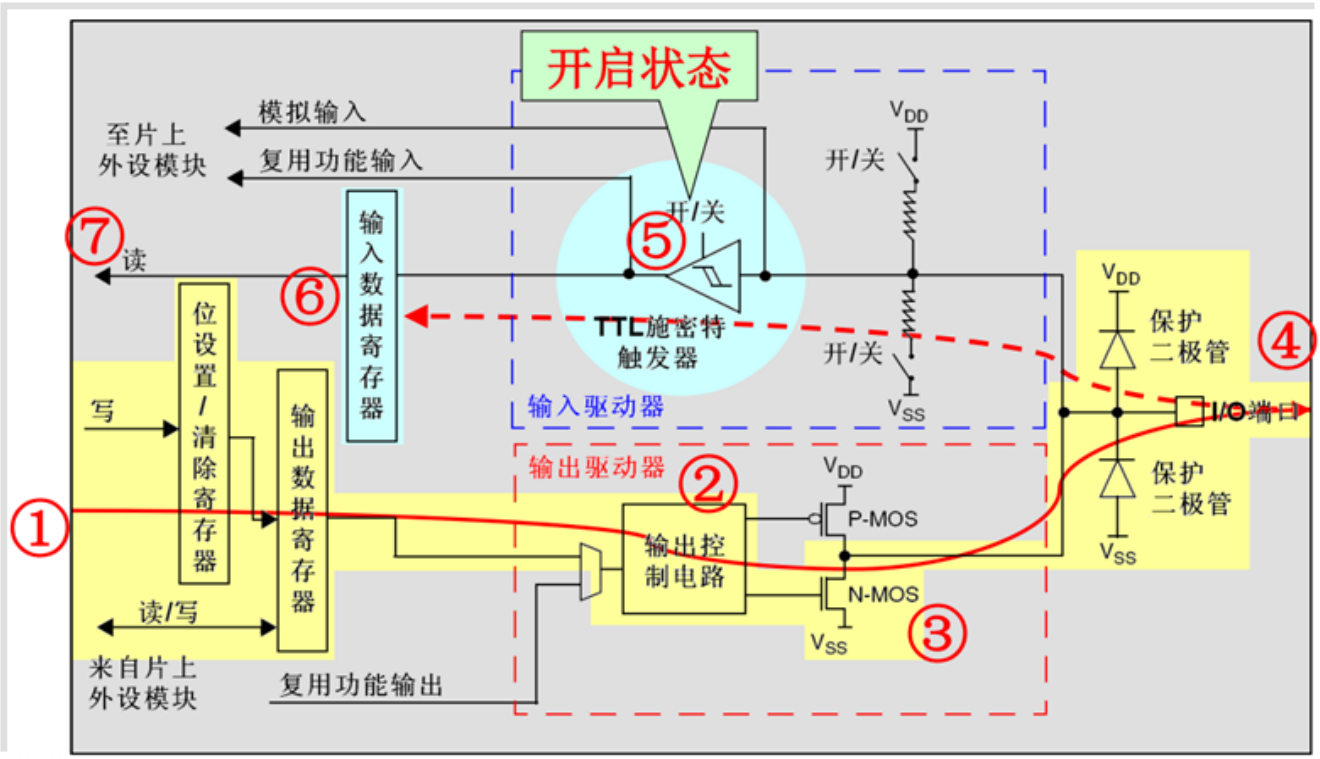
下拉输入模式下，I/O端口的电平信号直接进入输入数据寄存器。但是在I/O端口悬空（在无信号输入）的情况下，输入端的电平可以保持在低电平；并且在I/O端口输入为高电平的时候，输入端的电平也还是高电平。

模拟输入模式



模拟输入模式下，I/O端口的模拟信号（电压信号，而非电平信号）直接模拟输入到片上外设模块，比如ADC模块等等。

开漏输出模式

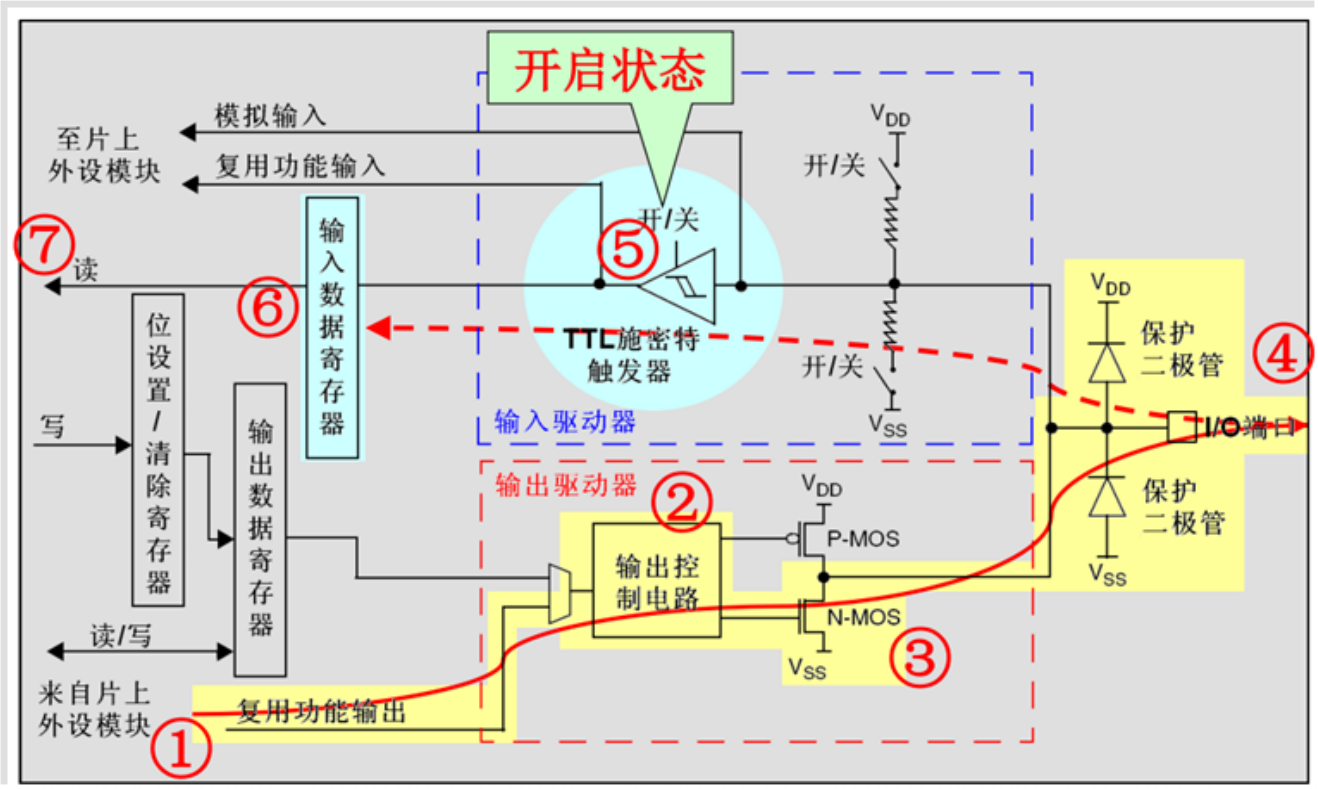


开漏输出模式下，通过设置位设置/清除寄存器或者输出数据寄存器的值，途经N-MOS管，最终输出到I/O端口。这里要**注意N-MOS管，当设置输出的值为高电平的时候，N-MOS管处于关闭状态，此时I/O端口的电平就不会由输出的高低电平决定，而是由I/O端口外部的上拉或者下拉决定；当设置输出的值为低电平的时候，N-MOS管处于开启状态，此时I/O端口的电平就是低电平。**同时，I/O端口的电平也可以通过输入电路进行读取；注意，I/O端口的电平不

一定是输出的电平。

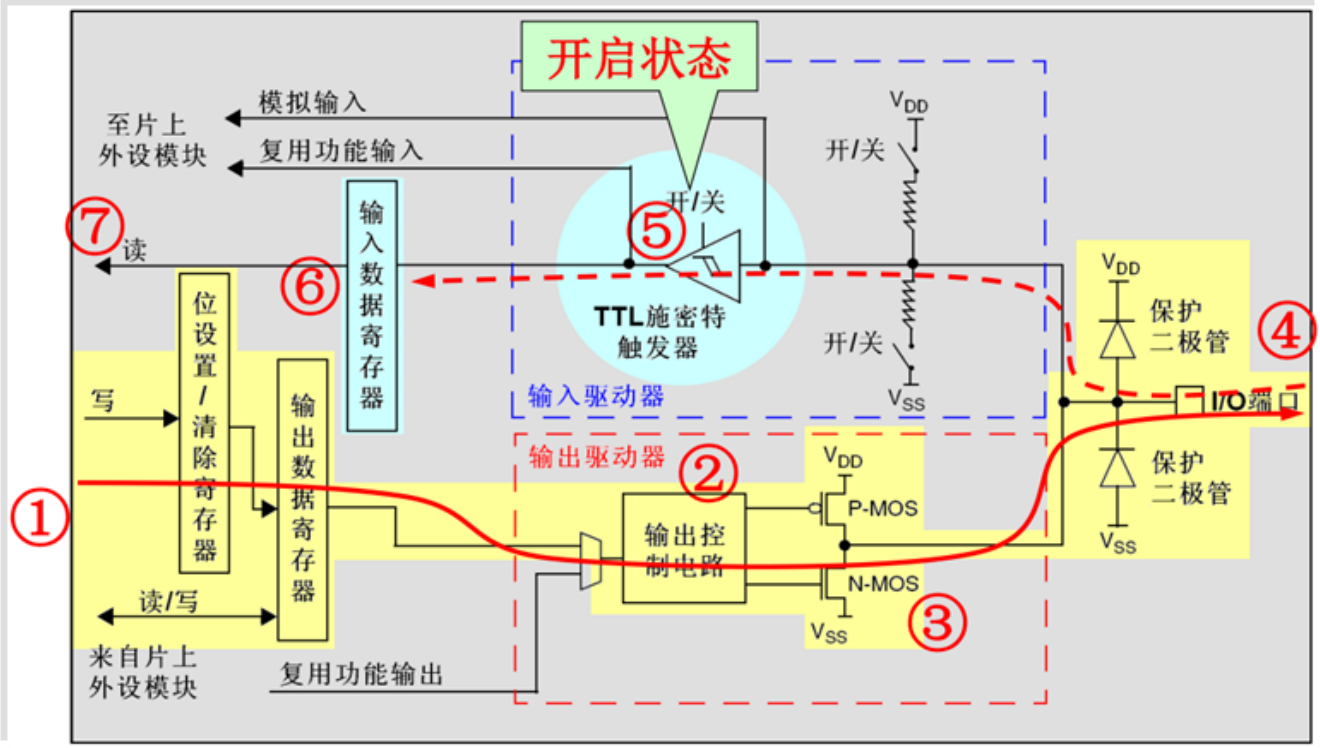
开漏输出的特点：输出高电平由外接上拉下拉来决定。

开漏复用输出模式



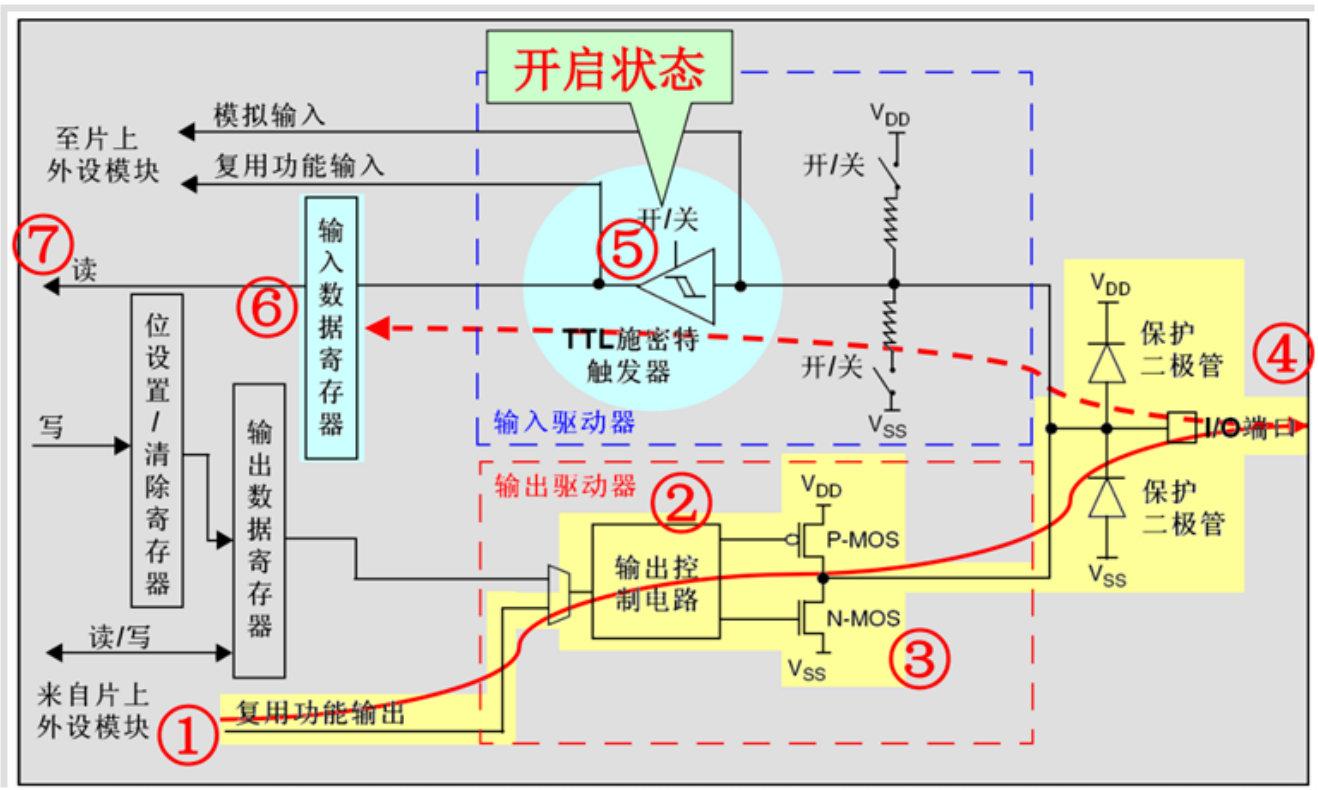
开漏复用输出模式，与开漏输出模式很是类似。只是输出的高低电平的来源，不是让CPU直接写输出数据寄存器，取而代之利用片上外设模块的复用功能输出来决定的。

推挽输出模式



推挽输出模式下，通过设置位设置/清除寄存器或者输出数据寄存器的值，途经P-MOS管和N-MOS管，最终输出到I/O端口。这里要**注意P-MOS管和N-MOS管，当设置输出的值为高电平的时候，P-MOS管处于开启状态，N-MOS管处于关闭状态，此时I/O端口的电平就由P-MOS管决定：高电平；当设置输出的值为低电平的时候，P-MOS管处于关闭状态，N-MOS管处于开启状态，此时I/O端口的电平就由N-MOS管决定：低电平。**同时，I/O端口的电平也可以通过输入电路进行读取；注意，此时I/O端口的电平一定是输出的电平。

复用推挽输出模式



推挽复用输出模式，与推挽输出模式很是类似。只是输出的高低电平的来源，不是让CPU直接写输出数据寄存器，取而代之利用片上外设模块的复用功能输出来决定的。

总结与分析

1、什么是推挽结构和推挽电路?

推挽结构一般是指**两个参数相同的三极管或MOS管分别受两互补信号的控制，总是在一个三极管或MOS管导通的时候另一个截止。高低电平由输出电平决定。**

推挽电路是两个参数相同的三极管或MOSFET，以推挽方式存在于电路中，各负责正负半周的波形放大任务。电路工作时，两只对称的功率开关管每次只有一个导通，所以导通损耗小、效率高。输出既可以向负载灌电流，也可以从负载抽取电流。推拉式输出级既提高电路的负载能力，又提高开关速度。

2、开漏输出和推挽输出的区别?

- **开漏输出：只可以输出强低电平，高电平得靠外部电阻拉高。输出端相当于三极管的集电极。适合于做电流型的驱动，其吸收电流的能力相对强(一般20ma以内)；**
- **推挽输出:可以输出强高、低电平，连接数字器件。**

几个常用的关于GPIO口的HAL库函数：

```
void HAL_GPIO_WritePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState);//写入电平
void HAL_GPIO_TogglePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin); //反转电平
GPIO_PinState HAL_GPIO_ReadPin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin); //读取电平，返回值为GPIO_PIN_STATE
```

前面两个用于推挽输出；第三个用于读取IO口电平。

实验二 串口通信

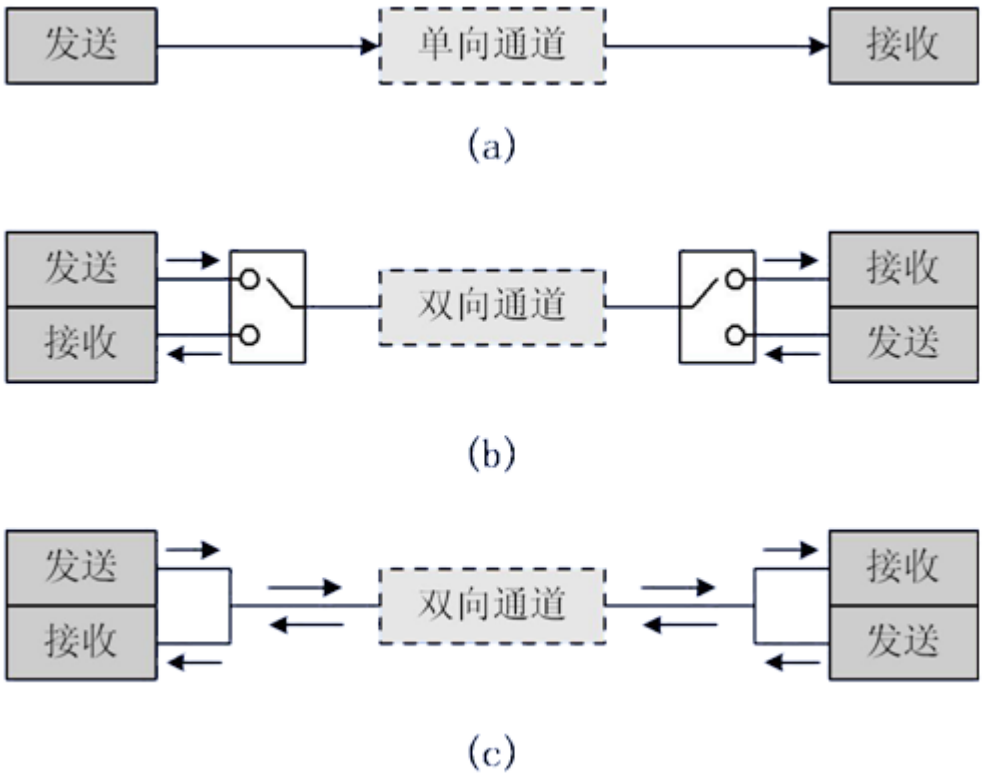
一般设备之间的通信方式可以分为串行通信和并行通信，二者各有特点：

	并行通信	串行通信
传输原理	数据多个位同时传输	数据一位一位传
优点	速度快	占用引脚资源少
缺点	占用引脚资源多	速度相对较慢

串行通信分类

1、按照数据传送方向分：

- 单工：**数据传输只支持在一个方向单向传输。**
- 半双工：允许数据在两个方向传输，但在某一时刻，只允许数据单向传输。它实际上是一种切换方向的单工通信；**它不需要独立的接收端和发送端，两者可以合并一起使用一个端口。**
- **全双工：允许数据同时在两个方向上传输。**因此，全双工通信是两个单工通信方式的结合，**需要独立的接收端和发送端。**



2、按照通信方式：

- **同步通信：带时钟同步信号传输。比如：SPI，IIC通信接口。**
- **异步通信：不带时钟同步信号。比如：UART(通用异步收发器)。**

在同步通讯中，收发设备上方会使用一根信号线传输信号，在时钟信号的驱动下双方进行协调，同步数据。例如，通讯中通常双方会统一规定在时钟信号的上升沿或者下降沿对数据线进行采样。

在异步通讯中不使用时钟信号进行数据同步，它们直接在数据信号中穿插一些用于同步的信号位，或者将主题数据进行打包，以数据帧的格式传输数据。通讯中还需要双方规约好数据的传输速率（也就是波特率）等，以便更好地同步。常用的波特率有4800bps、9600bps、115200bps等。

在同步通讯中，数据信号所传输的内容绝大部分是有效数据，而异步通讯中会则会包含数据帧的各种标识符，所以同步通讯效率高，但是同步通讯双方的时钟允许误差小，稍稍时钟出错就可能导致数据错乱，异步通讯双方的时钟允许误差较大。

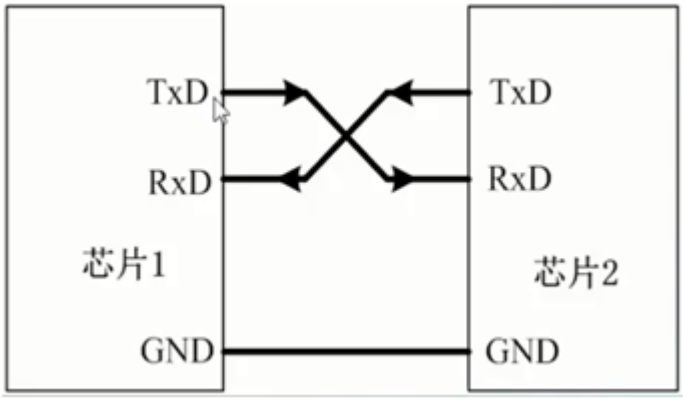
常见的串行通信接口

通信标准	通信方式	通信方向
UART	异步	全双工
SPI	同步	全双工
I2C	同步	半双工

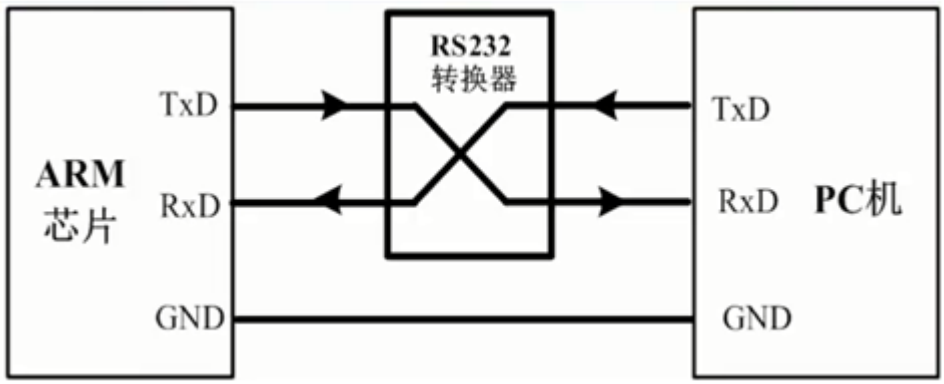
STM32串口通信

STM32的串口通信接口有两种，分别是：**UART（通用异步收发器）、USART（通用同步异步收发器）。而对于大容量STM32F10x系列芯片，分别有3个USART和2个UART。**一般我们接触不到同步通信，都会选择异步，即UART。

UART引脚连接方法



对于两个芯片之间的连接，两个芯片GND共地，同时TXD和RXD交叉连接。这里的交叉连接的意思就是，芯片1的RxD连接芯片2的TXD，芯片2的RXD连接芯片1的TXD。这样，**两个芯片之间就可以进行TTL电平通信了。**



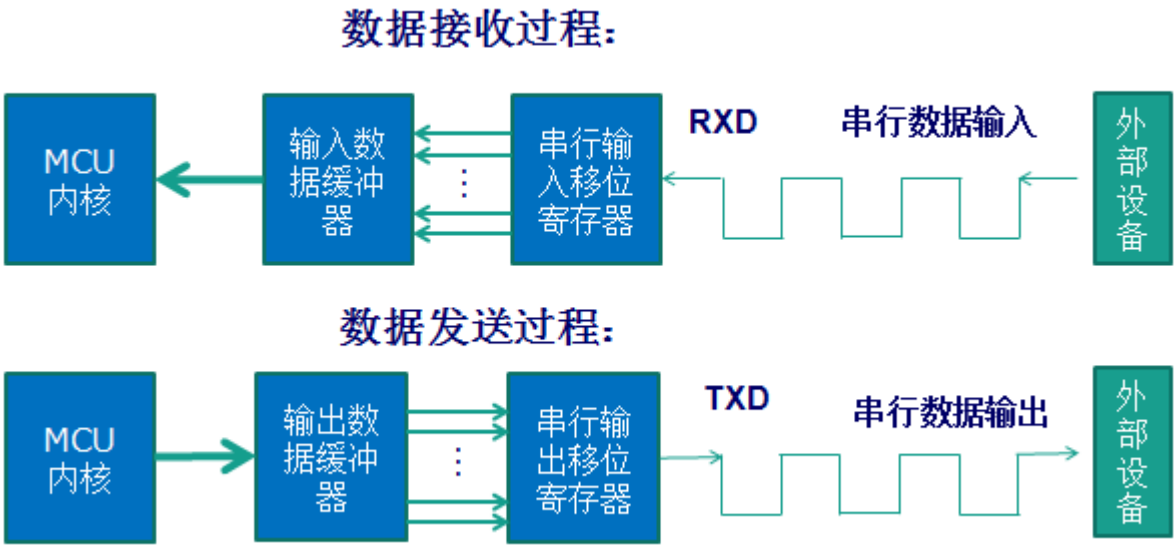
若是芯片与PC机（或上位机）相连，除了共地之外，就不能这样直接交叉连接了。尽管PC机和芯片都有TXD和RXD引脚，但是通常PC机（或上位机）通常使用的都是RS232接口（通常为DB9封装），因此不能直接交叉连接。RS232接口是9针（或引脚），通常是TxD和RxD经过电平转换得到的。故，**要想使得芯片与PC机的RS232接口直接通信，需要也将芯片的输入输出端口也电平转换成RS232类型，再交叉连接。**

经过电平转换后，芯片串口和rs232的电平标准是不一样的：

- **单片机的电平标准（TTL电平）：+5V表示1，0V表示0；**
- **Rs232的电平标准：+15/+13 V表示0，-15/-13表示1。**

STM32的UART特点

全双工异步通信； 分数波特率发生器系统，提供精确的波特率。发送和接受共用的可编程波特率，最高可达4.5Mbits/s； 可编程的数据字长度（8位或者9位）； 可配置的停止位（支持1或者2位停止位）； 可配置的使用DMA多缓冲器通信； 单独的发送器和接收器使能位； 检测标志：① 接受缓冲器 ②发送缓冲器空 ③传输结束标志； 多个带标志的中断源，触发中断； 其他：校验控制，四个错误检测标志。



UART串口通信的数据包以帧为单位，常用的帧结构为：1位起始位+8位数据位+1位奇偶校验位（可选）+1位停止位。

奇偶校验位分为奇校验和偶校验两种，是一种简单的数据误码校验方法。**奇校验是指每帧数据中，包括数据位和奇偶校验位的全部9个位中1的个数必须为奇数；偶校验是指每帧数据中，包括数据位和奇偶校验位的全部9个位中1的个数必须为偶数。**

串口通信实验

串口数据发送函数：

```
HAL_StatusTypeDef HAL_UART_Transmit(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size, uint32_t Timeout);
```

串口数据接收函数：

```
HAL_StatusTypeDef HAL_UART_Receive(UART_HandleTypeDef *huart, uint8_t *pData, uint16_t Size, uint32_t Timeout);
```

还有两个常用函数：

```
__weak void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart);//用户可以重定义该函数，为串口发送完成回调函数。顾名思义，只要串口发送完成，便执行该函数。
```

```
__weak void HAL_UART_RxCpltCpltCallback(UART_HandleTypeDef *huart);//该函数为串口接收完成回调函数。
```