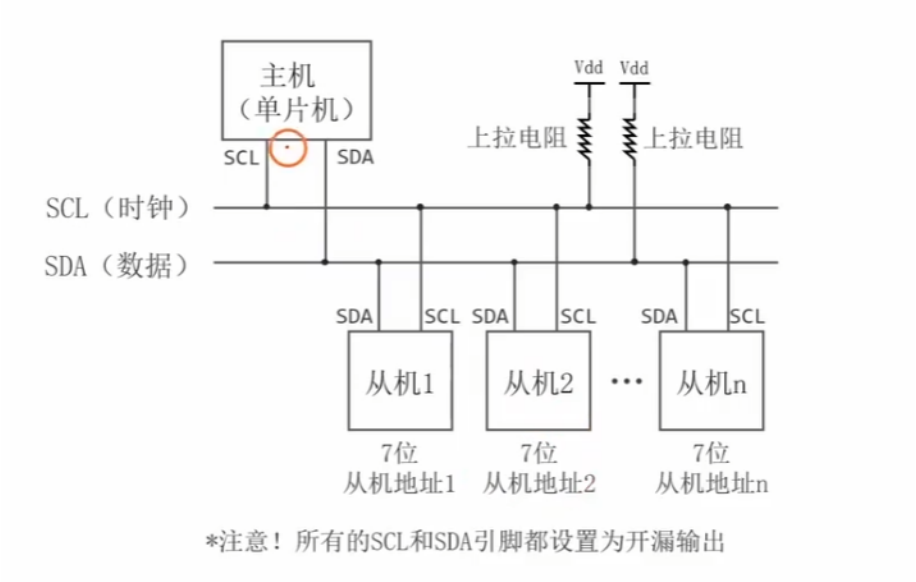
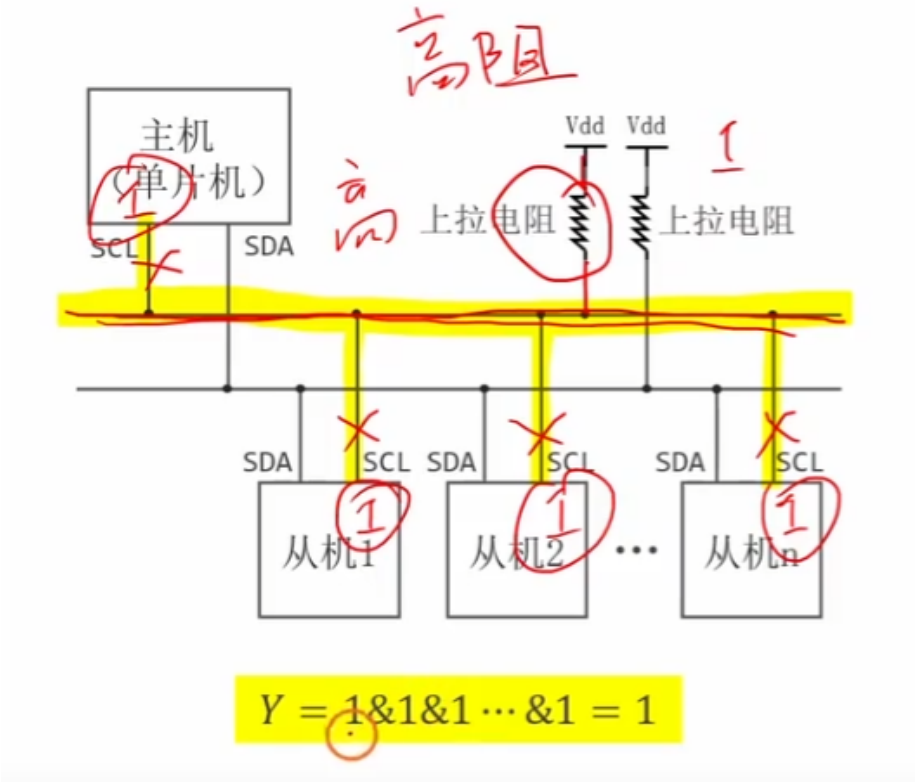
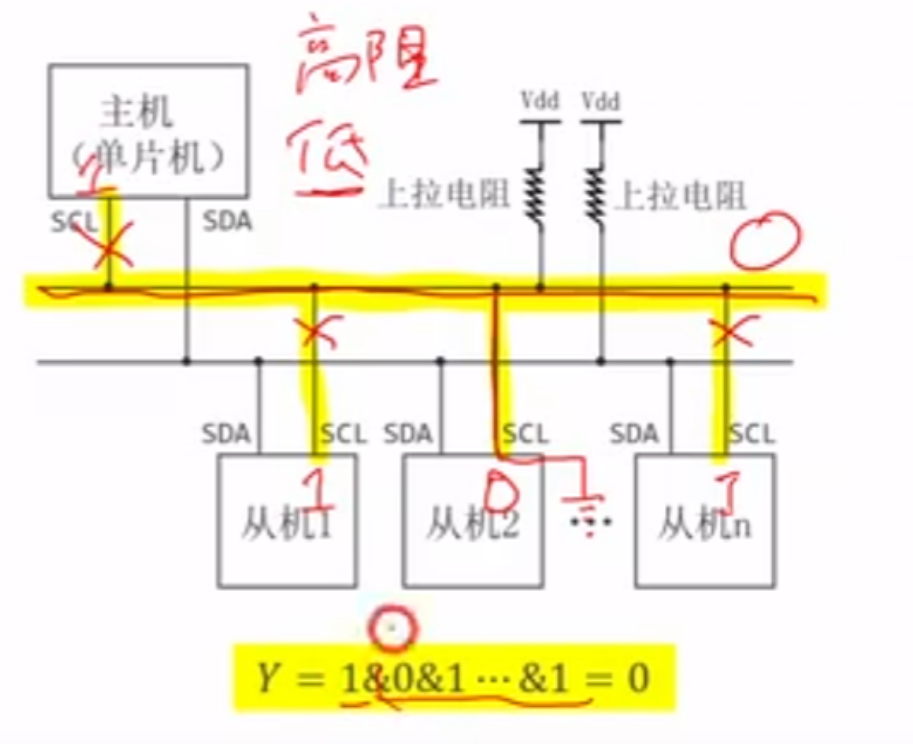
I2C系统

目的: 学习I2C工作原理, 利用库函数编写I2C代码, 最后利用汇编语言编写I2C代码



逻辑线与:当所有SCL引脚写1的时候,线上是高电压表示1; 当有一个SCL引脚写0的时候,线上是低电压表示0(对SDA也一样)





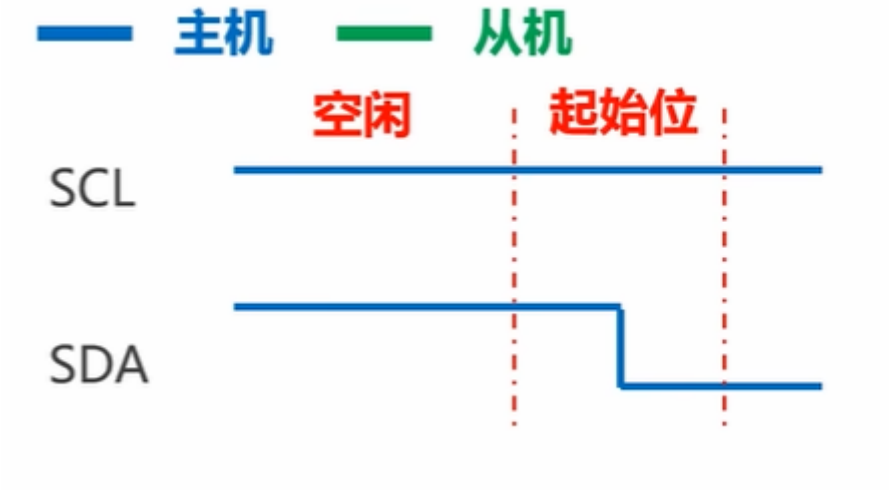
获得写0/1权利的主机/从机, 就是当前操作SCL线的单片机, 其他没有获得操作权的就写1断开. 有操作权的单片机输出的0/1经过SCL线, 全线上的机器都有资格接收到这个信息.

知识准备:I2C工作原理

- I2C用于主机和大量设备的链接
- I2C以一主多从的形式进行连接
- I2C由两条线组成:
    - SCL 串行时钟线, 负责传输时钟信号
    - SDA 串行数据线, 负责传输数据
    - 主机和从机都由SCL和SDA引脚, 连接在SCL和SDA
    - SCL和SDA上各连接一个上拉电阻
    - SCL和SDA的引脚都应该使用开漏输出
- I2C常见作用过程主要分为:
    - 起始位
    - 寻址阶段
    - 数据传输阶段
    - 停止位

起始位:

平时SCL, SDA高电压, 表示空闲状态, 在SCL高电压时向SDA发送下降沿表示起始位



寻址：

主机向总线发送从机地址，此处以7位为例，发送了起始位后所以从机都会有反应，而从机地址的发送旨在让我们锁定特定的从机.

发送了一串地址，地址后有一个读写位，发送后主机不动，等待从机应答，从机把ACK拉低表示应答，比如图中就是向0111100地址的从机发送了写的请求，从机应答ACK

此后的各种操作的总流程：

指定地址写操作：

- 开始
- 寻址
- 从机应答流程
    - 主机输出完一个字节后释放SDA，此时从机就可以操作SDA让主机注意到
    - 从机将SDA下拉表示给主机ACK的信号(置0)
    - 从机释放SDA，将SDA置1

- 主机指定从机的寄存器
- 重复第三步, 等待从机应答
- 主机发送数据给从机
- 重复第三步, 等待从机应答
- 结束

当前地址读操作:

这里的读是按照顺序直接读, 主机没有选择的权利

- 开始
- 寻址
- 从机应答
- 主机读入
    - 从机释放SDA
    - 每当SCL=1时, 读取数据一位数据
- 主机应答
    - 每当从机发送满一字节后, 从机释放SDA, 等待主机应答
    - 主机应答, SDA置0
    - 主机释放SDA
- 若此时信息未发完: 重复第四步, 主机继续读入
- 若信息已发完, 主机SDA置1, 停止读入
- 停止

指定地址读操作:

指定地址读操作可以使主机选择读的区域

- 开始
- 寻址, 但是发送的是从机地址+W(写)
- 从机应答
- 主机指定从机寄存器(目前和指定地址写操作一致)
- 重复3, 等待从机应答
- 重复起始条件

> *Sr意为重复起始条件，相当于另 起一个时序，因为指定读写标志位只能是跟着起始条件后的第一个字节*

- 再次发送寻址代码, 但是这次是从机地址 + R
- 从机应答
- 重复当前地址的第四步执行到第六步, 读取数据

多个字节的读写

指定地址写：最后一部分多重复几次即可，此时存储的数据地址是连续的

当前地址读/指定地址读：最后一部分多重复几次即可，连续读出一片区域的寄存器

注意：仅读一个字节就停止的话，一定要让主机发送非应答，让从机释放SDA，同理，多个字节读的话 在最后一个字节发送非应答即可

具体操作1, 调库实现I2C

我们先思考一下调用I2C的大致步骤:

- 为了多体验stm32的功能, 我们在P8, P9上使用I2C, 这样加深对重映射的了解, 所以第一步我们打开APB2时钟, 并使能I2C1的重映射
- 初始化P8P9引脚
-

  while(I2C_GetFlagStatus(I2Cx, I2C_FLAG_BUSY) == SET);
  I2C_GenerateSTART(I2Cx, ENABLE);
  while(I2C_GetFlagStatus(I2Cx, I2C_FLAG_SB) == RESET);

```
I2C_ClearFlag(I2Cx, I2C_FLAG_AF);
I2C_SendData(I2Cx, Addr & 0xfe);
while(1){
    if(I2C_GetITStatus(I2Cx, I2C_FLAG_ADDR) ==
SET) break;
    if(I2C_GetFlagStatus(I2Cx, I2C_FLAG_AF) ==
SET){
        I2C_GenerateSTOP(I2Cx, ENABLE);
        return -1;
    }
}

I2C_ReadRegister(I2Cx, I2C_Register_SR1);
I2C_ReadRegister(I2Cx, I2C_Register_SR2);
uint16_t i;
for(i=0; i < Size; i++)
{
    while(1)
    {
    if(I2C_GetFlagStatus(I2Cx, I2C_FLAG_AF) ==
SET){
            I2C_GenerateSTOP(I2Cx, ENABLE);
            return -2;
        }
    if(I2C_GetFlagStatus(I2Cx, I2C_FLAG_TXE) ==
SET) break;
    }
        I2C_SendData(I2Cx, pData[i]);
}
```

```
    while(1)
{
    if(I2C_GetFlagStatus(I2Cx, I2C_FLAG_AF) ==
SET)
    {
            I2C_GenerateSTOP(I2Cx, ENABLE);
            return -2;
    }

    if(I2C_GetFlagStatus(I2Cx, I2C_FLAG_BTF) ==
SET) break;

}
```

```
    I2C_GenerateSTOP(I2Cx, ENABLE);
    return 0;
```