Electronic Note

# 元器件介绍

## 二极管

## 三极管

A picture containing diagram, line, plan, technical drawing

Description automatically generated

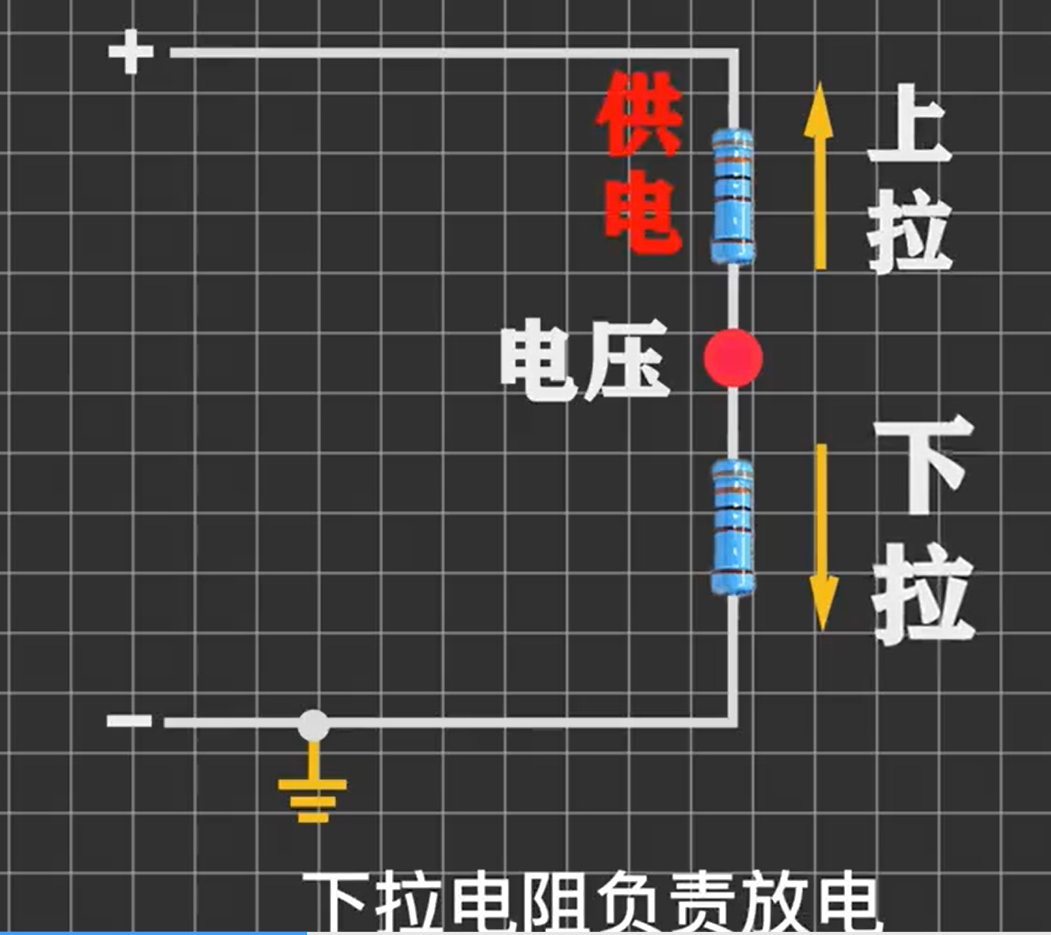
**NPN:** 当b极是高电平, c极电流就可以流向e极(发射极);

**PNP:** 当b极是**低电平**, e极电流就可以流向c极;

# 简单电路:

## 上拉/下拉电阻:

**概念:**



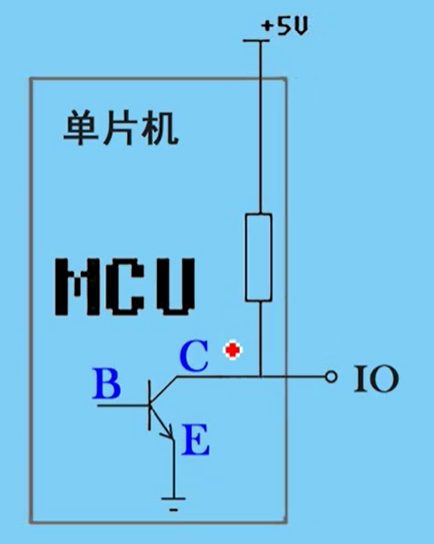
**放电**

**目的调节该点**

上拉电阻: 电阻一端接正极的叫上拉电阻;

下拉电阻: 电阻一端接负极的叫下拉电阻;

**单片机上的上拉电阻:**



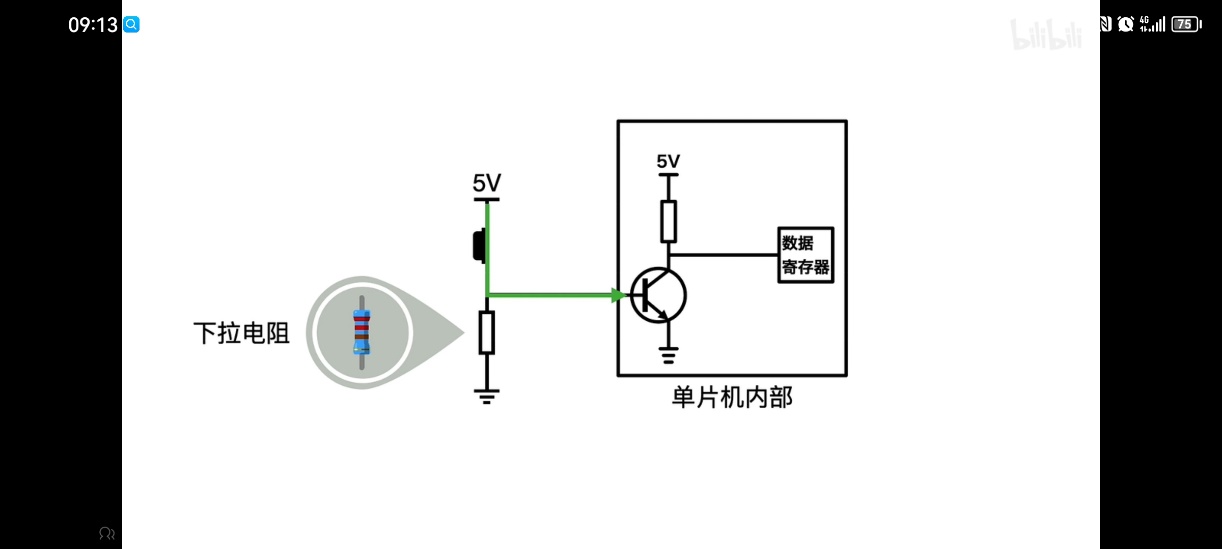
IO接上拉电阻才可以有电压输出,否则没有.

**单片机下拉电阻:**

**下拉电阻作用:** 将不确定的信号钳位在低电平;

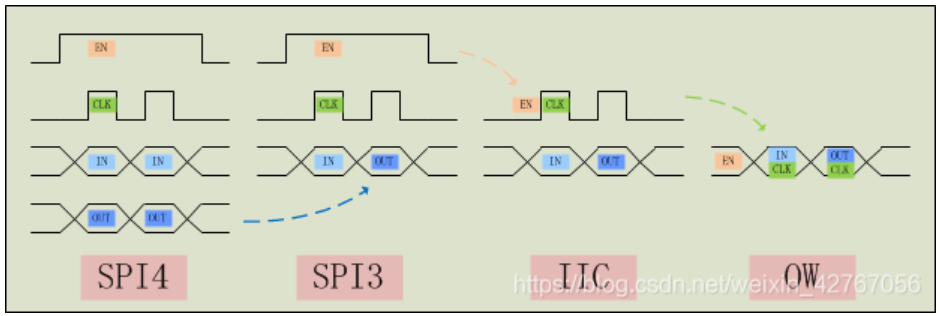
如希望A点有稳定的高电平,如果没有下拉电阻,由于三极管的特点会导致A点的电压在变化;

如果给三极管基极输入一个高电平, A点就可以保持在低电平;



**A**

# 总线演变历史



**1979, SPI**

美国 摩托罗拉

**1982, I2C**

荷兰 飞利浦

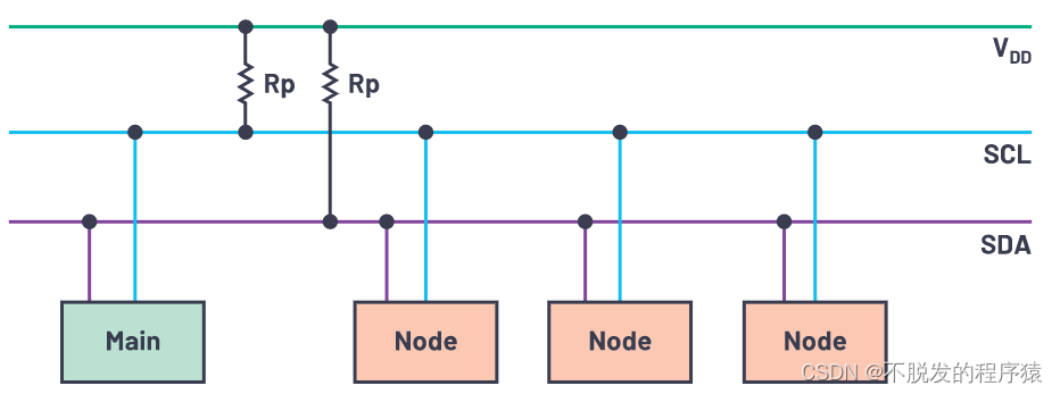
**,OW**

**美国DALLAS公司**

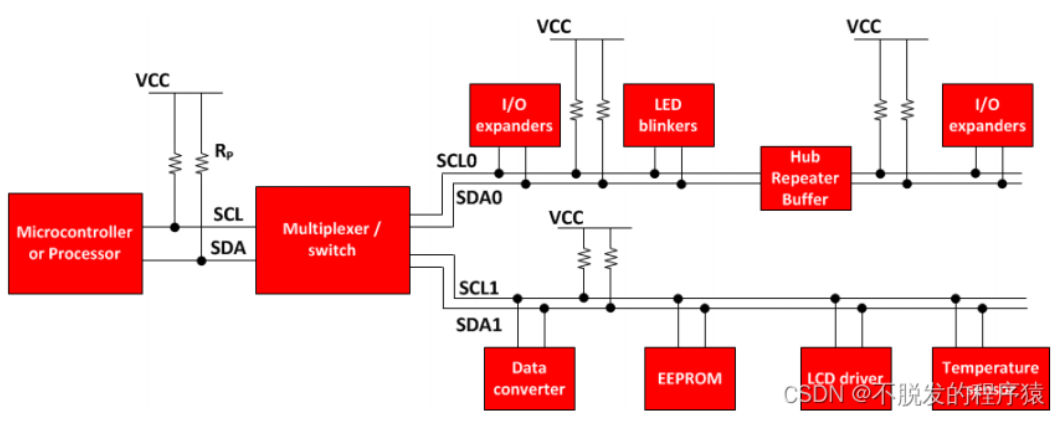
# I2C

## I2C简介

I2C（集成电路总线），由Philips公司（2006年迁移到NXP）在1980年代初开发的一种简单、双线双向的同步串行总线，它利用一根时钟线和一根数据线在连接总线的两个器件之间进行信息的传递，为设备之间数据交换提供了一种简单高效的方法。每个连接到总线上的器件都有唯一的地址，任何器件既可以作为主机也可以作为从机，但同一时刻只允许有一个主机。



I2C 标准是一个具有冲突检测机制和仲裁机制的真正意义上的多主机总线，它能在多个主机同时请求控制总线时利用仲裁机制避免数据冲突并保护数据。作为嵌入式开发者，使用I2C总线通信的场景有很多，例如驱动FRAM、E2PROM、传感器等。



**总结来说，I2C总线具有以下特点：**

* 只需要SDA、SCL两条总线；
* 没有严格的波特率要求；
* 所有组件之间都存在简单的主/从关系，连接到总线的每个设备均可通过唯一地址进行软件寻址；
* I2C是真正的多主设备总线，可提供仲裁和冲突检测；
* **传输速度分为四种模式：**
  1. 标准模式（Standard Mode）：100 Kbps
  2. 快速模式（Fast Mode）：400 Kbps
  3. 高速模式（High speed mode）：3.4 Mbps
  4. 超快速模式（Ultra fast mode）：5 Mbps
* 最大主设备数：无限制；
* 最大从机数：理论上，1008个从节点，寻址模式的最大节点数为2的7次方或2的10次方，但有16个地址保留用于特殊用途。

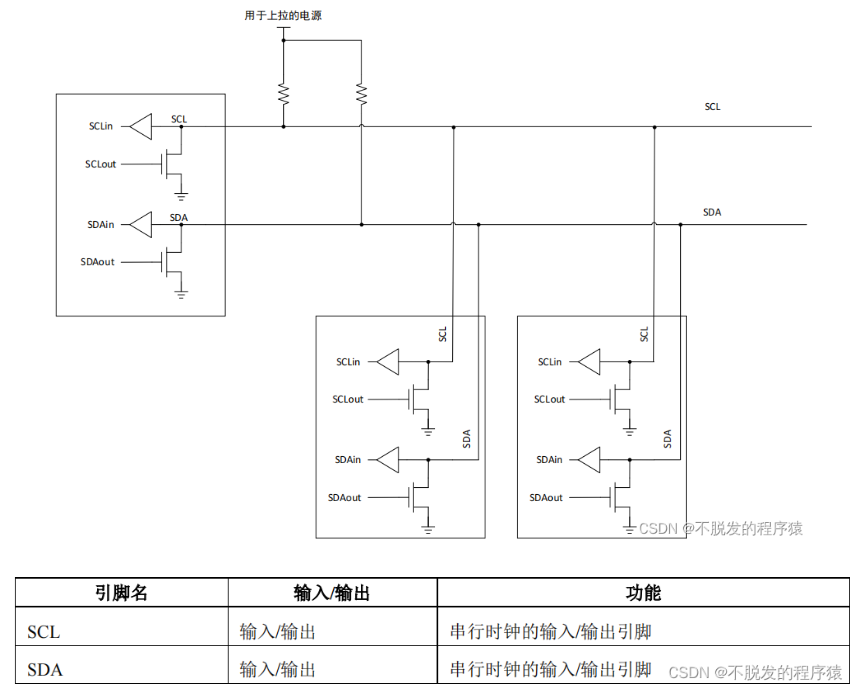
I2C有16个保留I2C地址。这些地址对应于以下两种模式之一：0000 XXX或1111 XXX。下表显示了为特殊目的而保留的I2C地址。



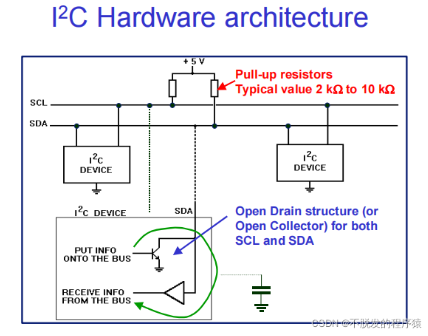
I2C还有两个变体，分别专注于系统和电源应用，称为系统管理总线（SMBus）和电源管理总线（PMBus）。

## 物理特性

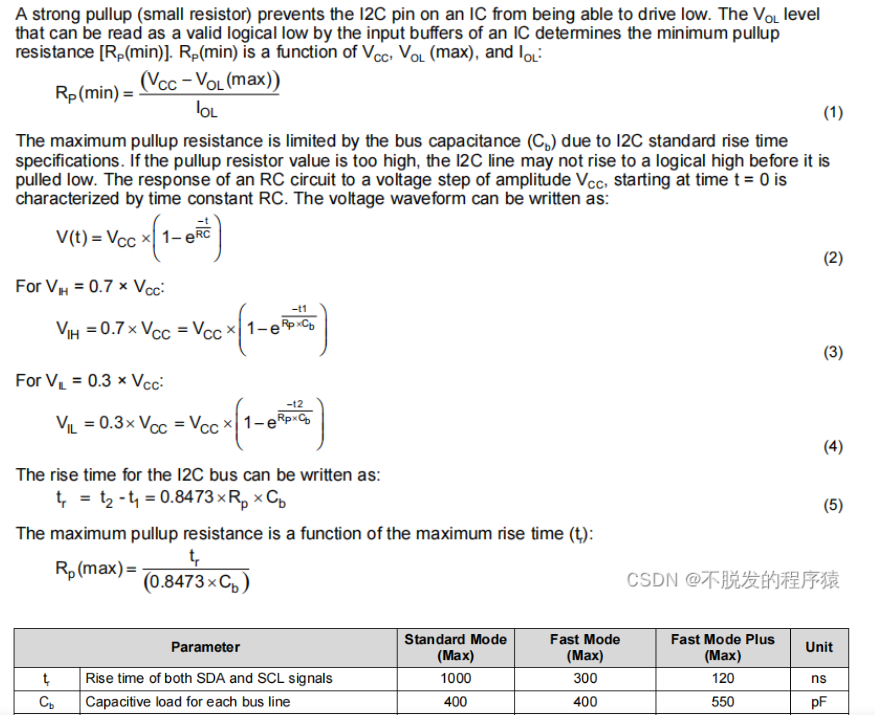
I2C 总线使用连接设备的 "SDA"（ 串行数据总线）和"SCL"（ 串行时钟总线 ） 来传送信息。



I2C 总线内部使用[漏极开路](https://blog.csdn.net/m0_38106923/article/details/103918647)输出驱动器，因此 SDA和 SCL **可以被拉低为低电平，但是不能被驱动为高电平**，所以每条线上都要使用一个上拉电阻，默认情况下将其保持在高电平。



I2C 总线上拉电阻阻值取决于系统应用，[TI 官方手册](https://www.ti.com/lit/an/slva689/slva689.pdf)推荐使用以下公式来计算上拉电阻值：



根据上表，这里不难发现需要在做电阻选择需要满足几个条件：

1. 灌电流最大值为3mA；
2. 低电平输出电压设置了最大值为0.4V。

所以根据上述公式可以计算，对于5V的电源，每个上拉电阻阻值至少1.53kΩ，而对于3.3V的电源，每个电阻阻值至少967Ω。

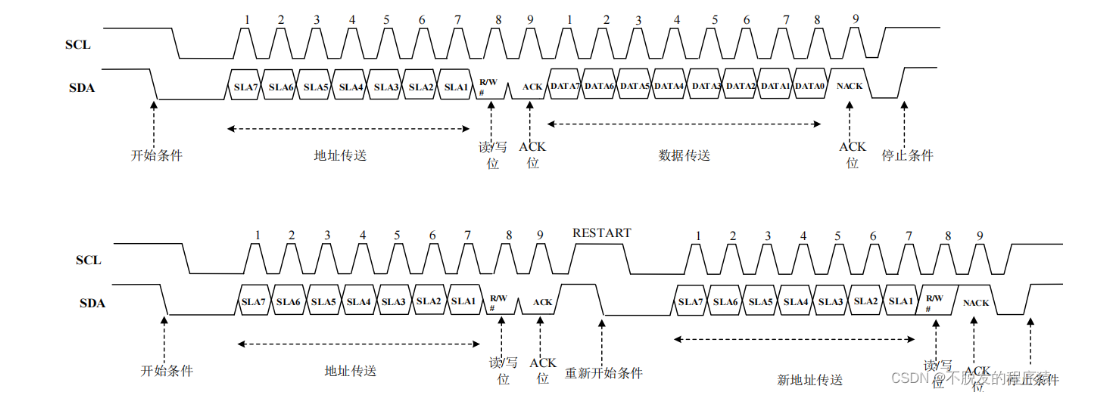
如果觉得计算电阻值比较麻烦，也可以使用**典型值 4.7kΩ**。若各位想了解更多可直接参见手册说明。

## 通讯特性

通常情况下，一个完整的I2C通信过程包括以下 4 部分：

* 开始条件
* 地址传送
* 数据传送
* 停止条件

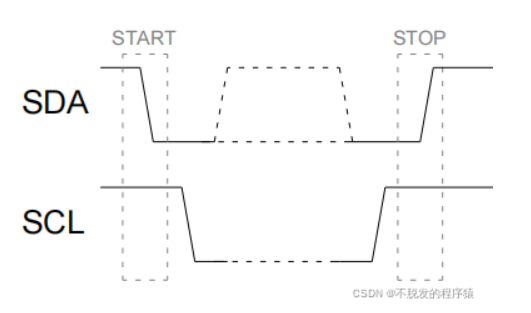
主机在 SCL 线上输出串行时钟信号，数据在 SDA 线上进行传输，每传输一个字节（**最高位 MSB 开始传输**）后面跟随一个应答位，一个 SCL 时钟脉冲传输一个数据位。



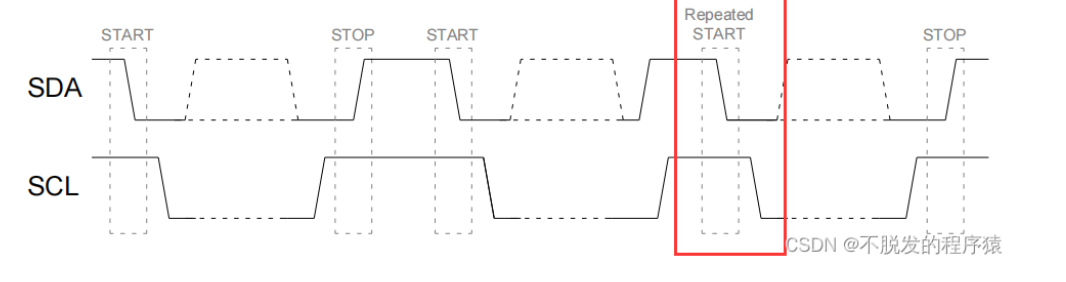
### 开始和停止条件

当总线上的主机都不驱动总线，总线进入空闲状态， SCL 和 SDA 都为高电平。总线空闲状态下总线上设备都可以通过发送开始条件启动通信。

**当 SCL 线为高时**，SDA 线上出现由高到低的信号，表明总线上产生了起始信号。 SDA 线上出现由低到高的信号，表明总线上产生了停止信号，如下图所示：



当两个起始信号之间没有停止信号时，即产生了重复起始信号。主机采用这种方法与另一个从机或相同的从机以不同传输方向进行通信（例如：从写入设备到从设备读出）而不释放总线。如下图所示：

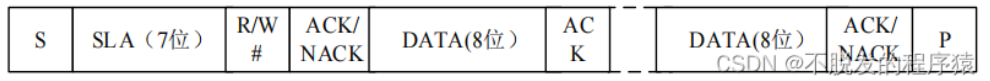


### 地址传送

**开始条件或者重新开始条件后面的帧是地址帧**（一个字节），用于指定主机通信的对象地址，在发送停止条件之前，指定的从机一直有效。

I2C通讯支持：7 位寻址和10 位寻址两种模式。

7 位寻址模式，地址帧（8bit）的高 7 位为从机地址，地址帧第 8 位来决定数据帧传送的方向：7 位从机地址 + 1位 读/写位，读/写位控制从机的数据传输方向（**0：写； 1：读**） 。帧格式如下所示：



0 位寻址模式，主机发送帧，第一帧 发送头序列（11110XX0，其中 XX 表示 10 位地址的高 两位），然后第二帧发送低八位从机地址。 主机接收帧 ，第一帧发送头序列（11110XX0，其中 XX 表示 10 位地址的高两位），然后第二帧发送低八位从机地址。接下来会发送一个重新开始条件，然后再发送一帧头序列（11110XX1 ，其中 XX 表示 10 位地址的高两位）帧格式如下所示：

A close-up of a white box

Description automatically generated

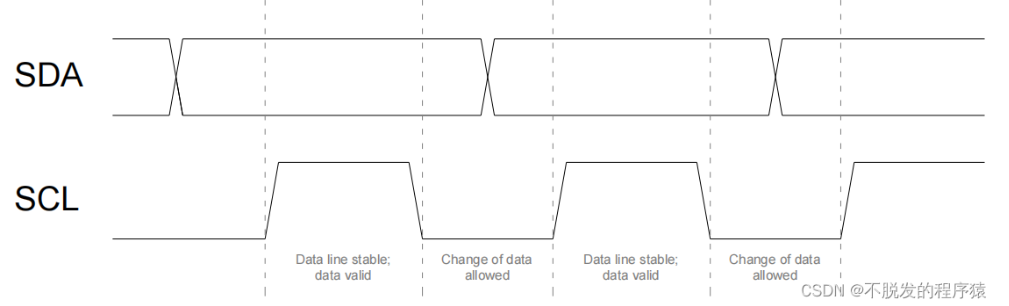
解析如下：

* S ：表示开始条件；
* SLA ：表示从机地址；
* R/W#：表示发送和接收的方向。当 R/W# 为“1” 时，将数据从从机发送到主机；当 R/W#为“0” 时，将数据从主机发送到从机；
* Sr ：表示重新开始条件；
* DATA ：表示发送和接收的数据；
* P ：表示停止条件。

### 数据传送

地址匹配一致后，总线上的主机根据 R/W 定义的方向一帧一帧的传送数据。 所有的地址帧后传送的数据都视为数据帧。即使是 10 位地址格式的低 8 位地址也视为数据帧。

数据帧的长度是 8 位。 SCL 的低电平 SDA 变化， SCL 的高电平 SDA 保持，每个时钟周期发送一位数据。数据帧后的第 9 个时钟是应答位，是接收方向发送方传送的握手信号。

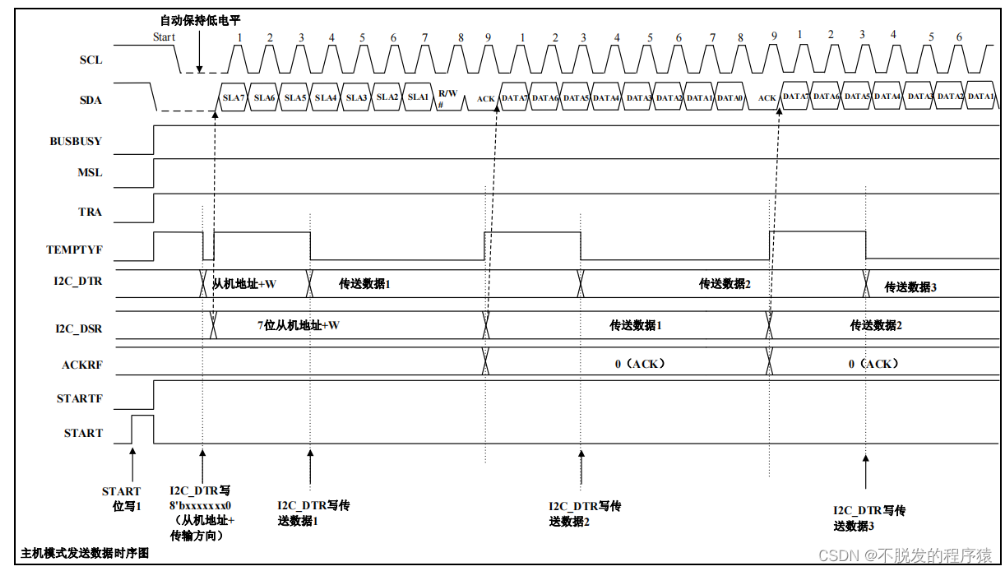


如果总线上从机接收数据，在第 9 个时钟周期不响应主机，从机必须发送 NACK。如果总线上主机接收数据，第 9 个周期发送 NACK，从机接收到 NACK，从机停止发送数据。

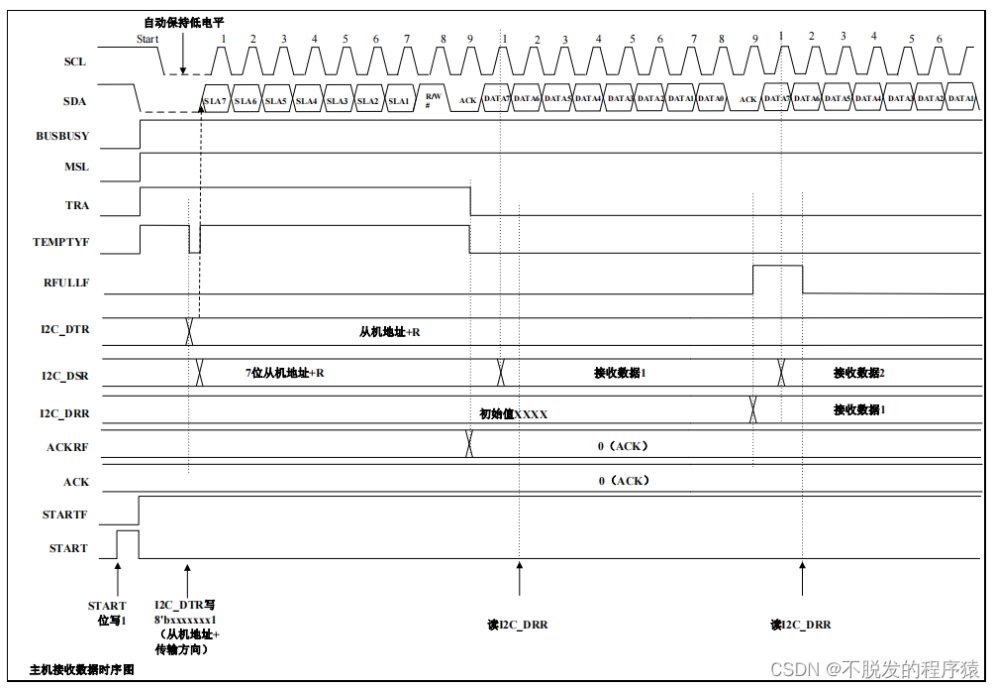
无论主机还是从机发送了 NACK，数据传送终止。主机可以做下列任一动作：

* 发送停止条件释放总线 ；
* 发送重新开始条件开始一个新的通信。

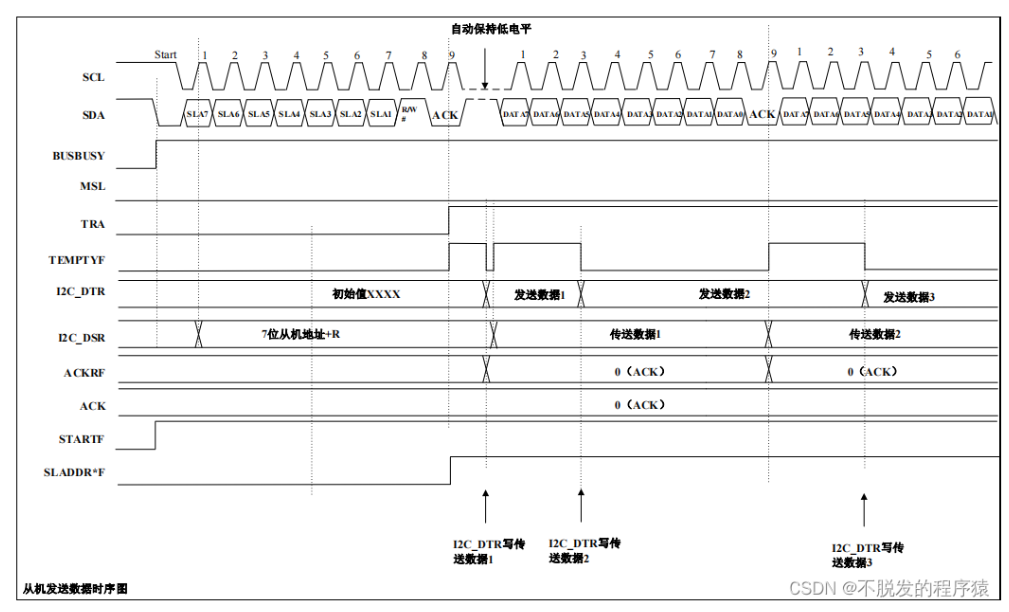
以华大MCU（HC3F4A0系列）为例，在主机接收模式中，主机输出 SCL 时钟，接收从机数据并返回应答。主机接收数据的运行时序例如下图所示：



在主机接收模式中，主机输出 SCL 时钟，接收从机数据并返回应答。主机接收数据的运行时序例如下图所示：



在从机发送模式中，接收来自主机的 SCL 时钟，本产品为从机发送数据，并且接收主机返回应答。从机发送数据的运行时序例如下图所示：



在从机接收模式中，接收来自主机的 SCL 时钟和数据，接收完数据后返回应答。从机接收数据的运行时序例如下图所示：

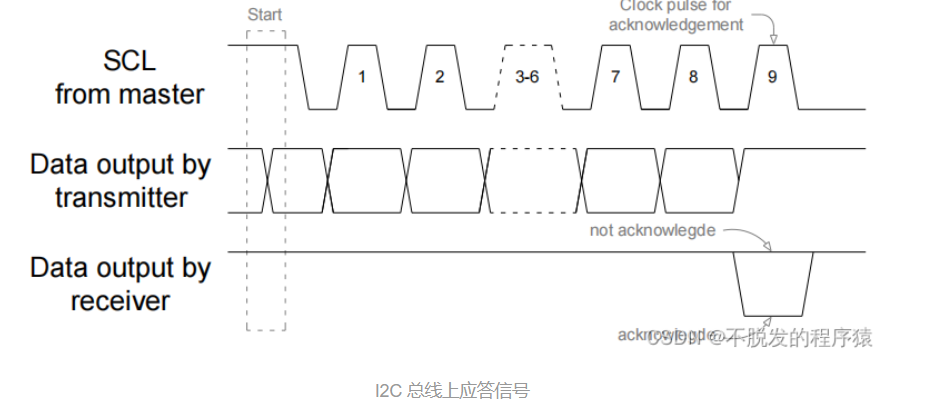
A diagram of a machine

Description automatically generated

### 总线应答

每传输一个字节，后面跟随一个应答位。通过将 SDA 线拉低，来允许接收端回应发送端。ACK 为 一个**低电平信号**，当时**钟信号为高时**， **SDA 保持低电平**则表明接收端已成功接收到发送端的数据。

当主机作为发送器件时，如果从机上产生无响应信号（NACK） ，主机可以产生停止信号来退出数据传输，或者产生重复起始信号开始新一轮的数据传输。当主机作为接收器件时，发生无响应信号（NACK） ，从机释放 SDA 线，使主机产生停止信号或重复起始信号。



### 总线仲裁

I2C 总线上的仲裁分为两个部分： SCL 线上的同步和 SDA 线上的仲裁。

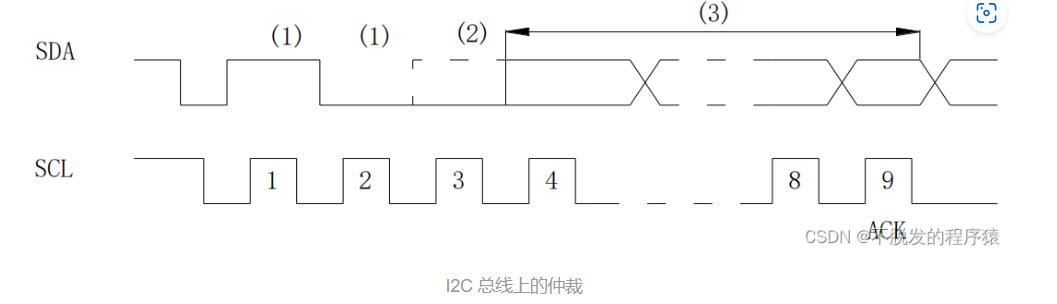
* SCL 线上的同步（时钟同步）

由于 I2C 总线具有线“与”的逻辑功能， SCL 线上只要有一个节点发送低电平，总线上就表现低电平。当所有的节点都发送高电平时，总线才能表现为高电平。所以，时钟低电平的时间由时钟电平期最长的器件决定，而时钟的高电平时间由时钟高电平期最短的器件决定。

由于 I2C 这种特性，当多个主机同时发送时钟信号时，在总线上表示的是统一的时钟信号。如果从机希望主机降低传送速度可以通过将 SCL 主动拉低延长其低电平时间来通知主机，当主机在准备下一次传送时发现 SCL 的电平被拉低时进行等待，直到从机完成操作并释放 SCL 线的控制权。

* SDA 线上的仲裁

SDA 线上的仲裁也是由于 I2C 总线具有线“与”的逻辑功能。主机在发送数据后，通过比较总线上的数据来决定是否退出竞争。丢失仲裁的主机立即切换到未被寻址的从机状态，以确保自身能被仲裁胜利的主机寻址到。仲裁失败的主机继续输出时钟脉冲（在 SCL 上），直到发送完当前的串行字节。通过这种原理可以保证 I2C 总线在多个主机企图控制总线时保证数据的不丢失。



**解析如下：**

（1）另一器件发送串行数据；

（2）另一器件通过拉低 SDA 先撤消了该 I2C 主机发送的一个逻辑 1 （虚线）。仲裁丢失，I2C 进入从接收模式；

（3）此时 I2C 处于从接收模式，但仍产生时钟脉冲，直至发送完当前字节。 I2C 将不为下个字节的传输产生时钟脉冲。一旦赢得仲裁，SDA 上的数据传输由新的主机来启动。

## 时钟同步/时钟延展

I2C规范没有为时钟同步规定任何超时条件，也就是说，任何器件都可以根据需要保持SCL。

在I2C通信协议中，时钟速度和信号始终由主器件产生。I2C主器件产生的信号提供主器件和节点连接之间的同步。

在某些情况下，节点或子节点不是以全状态工作，在接收主器件生成的时钟之前，需要减慢速度。这是通过一种称为"时钟同步/时钟延展"的机制来实现的。

在时钟同步/时钟延展期间，为了降低总线速度，允许节点压低时钟。而在主器件方面，在其变为高电平状态后，必须回读时钟信号。然后，它必须等待，直至线路达到高电平状态。

通过时钟同步/时钟延展，I2C节点器件可以强制主器件进入等待状态。当节点器件需要更多时间来管理数据时，例如存储接收到的数据或准备发送另一字节的数据时，它可能会执行时钟同步/时钟延展。这通常发生在节点器件接收并确认收到一个字节的数据之后。

是否需要时钟延展取决于节点器件的功能。这里有两个例子：

* 处理器件（如微处理器或微控制器）可能需要额外的时间来处理中断，接收和管理数据，以及执行适当的功能；
* 较简单的器件（如EEPROM）不在内部处理数据，因此不需要时钟延展来执行任何功能。

## 通信时序和协议

说起I2C通信协议必然离不开通信时序，主器件和从节点必须遵守I2C时序规格才能正确传输数据。



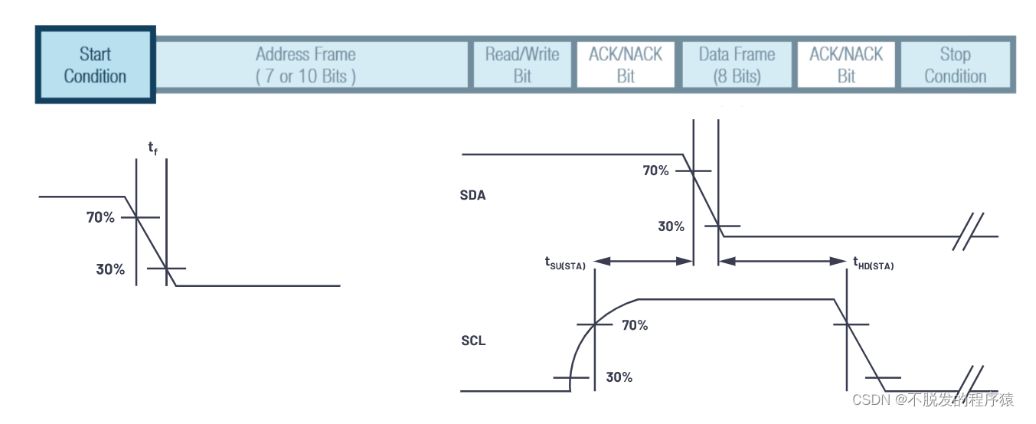
下表显示了时序规格表上给出的符号和参数。

A screenshot of a computer

Description automatically generated

### 起始条件

起始条件总是在传输开始时出现，并由主器件发起。这样做是为了唤醒总线上的空闲节点器件。SDA线从高电平切换到低电平，然后SCL线从高电平切换到低电平。时序和协议如下图所示：



### 地址帧

地址帧包含7位或10位序列，具体取决于可用性（参见数据手册）。如下图所示：



不像SPI协议，I2C没有节点选择线路，因此它需要另一种方法来让节点知道数据正向其发送，而不是向另一个节点发送。这是通过寻址来实现的。地址帧始终是新消息中起始位之后的第一帧。

主器件将其想要与之通信的节点地址发送到其所连接的每个节点。然后，每个节点将主器件所发送的地址与其自己的地址进行比较。如果地址匹配，它便向主器件发送一个低电压ACK位。如果地址不匹配，则节点什么也不做，SDA线保持高电平。

### 读/写位

地址帧的最后一位告知节点，主器件是想要将数据写入其中还是从中接收数据。如果主器件希望将数据发送到节点，则读⁄写位处于低电平。如果主器件请求从节点得到数据，则该位处于高电平。如下图所示：



### ACK/NACK位

消息中的每一帧后面都跟随一个应答⁄不应答位。如果成功接收到一个地址帧或数据帧，则从机会向主机返回一个ACK位。如下图所示：



### 数据帧

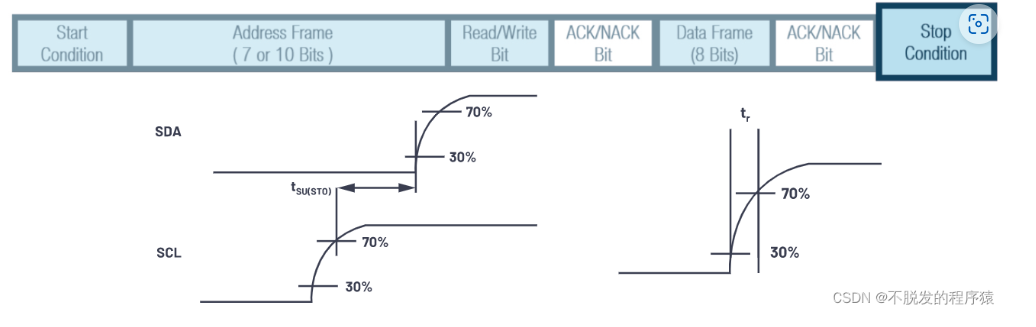
主器件检测到来自从节点的ACK位之后，就准备发送第一数据帧。数据帧总是8位长，并以MSB优先方式发送。每个数据帧之后紧接着一个ACK⁄NACK位，以验证该帧是否已成功接收。主器件或节点（取决于谁发送数据）必须收到ACK位，然后才能发送下一数据帧。时序和协议如下图所示：

A diagram of a diagram

Description automatically generated

### 停止条件

发送完所有数据帧之后，主器件可以向节点发送停止条件以停止传输。停止条件是指SCL线上的电压从低电平变为高电平，然后在SCL线保持高电平的情况下，SDA线上的电压从低电平变为高电平。时序和协议如下图所示：

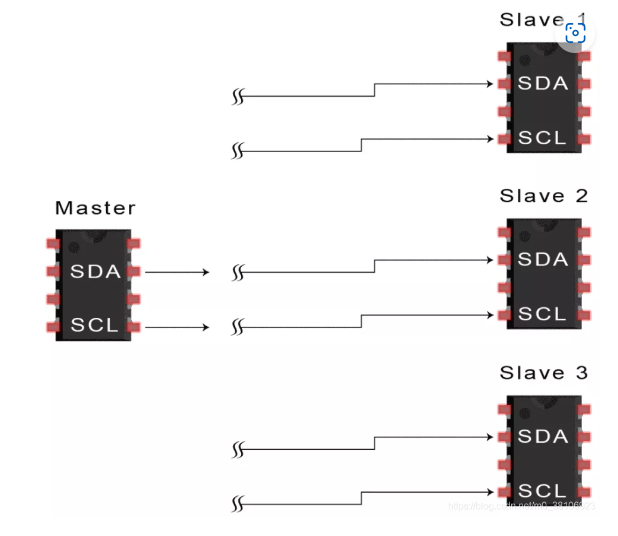


## 工作过程

最后整体叙述一下I2C通讯过程，本小节内容整理来源于：微信公众号：小麦大叔，作者菜刀和小麦。

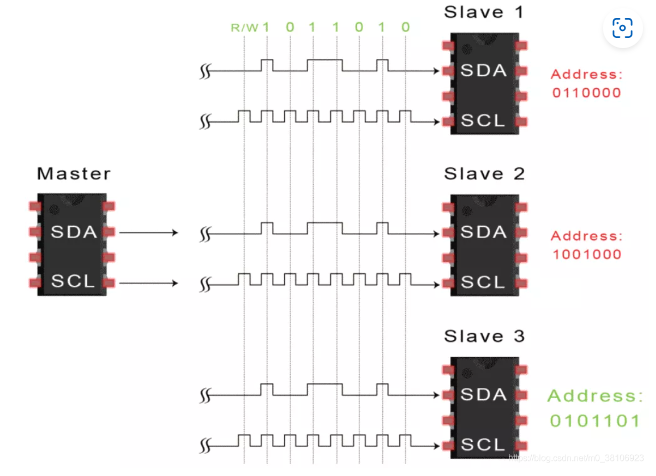
**第1步:** 起始条件

主设备通过将SDA线从高电平切换到低电平，再将SCL线从高电平切换到低电平，来向每个连接的从机发送启动条件，如下图所示：



**第2步:** 发送从设备地址

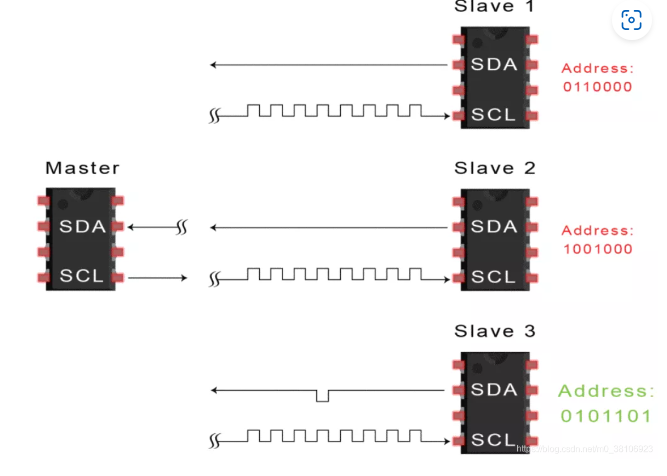
主设备向每个从机发送要与之通信的从机的7位或10位地址，以及相应的读/写位，如下图所示：



**第3步:** 接收应答

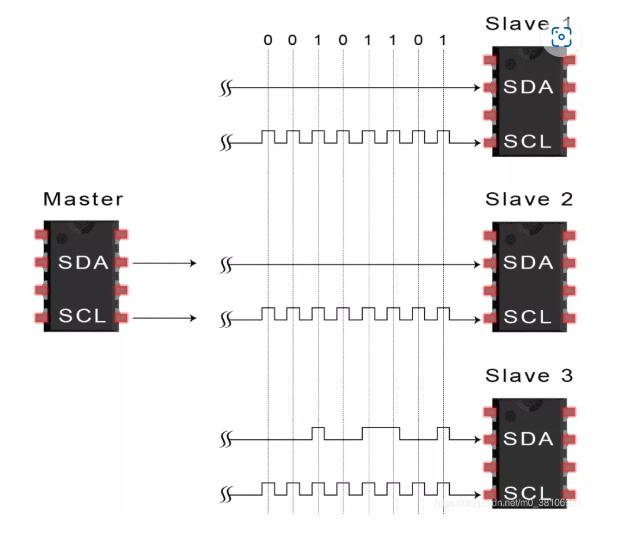
每个从设备将主设备发送的地址与其自己的地址进行比较。如果地址匹配，则从设备通过**将SDA线拉低一位以表示返回一个ACK位。**

如果来自主设备的地址与从机自身的地址不匹配，则**从设备将SDA线拉高，表示返回一个NACK位**。



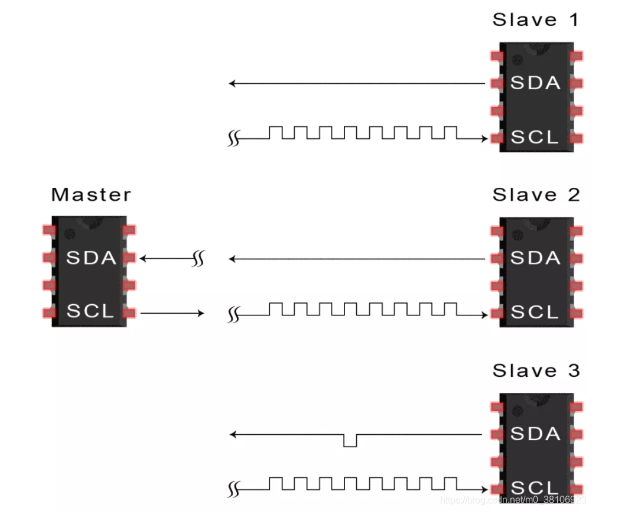
**第4步:** 收发数据

主设备发送或接收数据到从设备，如下图所示：



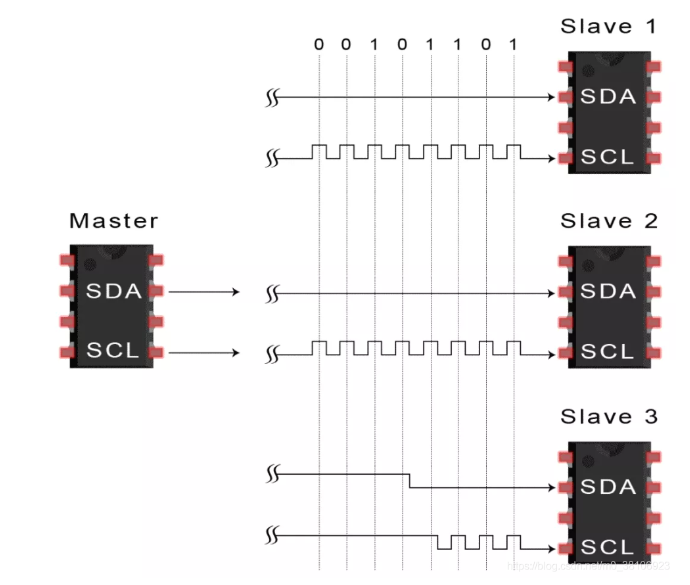
第5步: 接收应答

在传输完每个数据帧后，接收设备将另一个ACK位返回给发送方，以确认已成功接收到该帧，如下图所示：



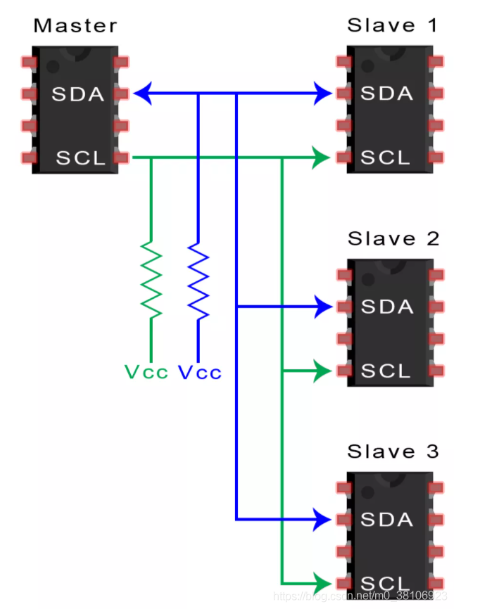
第6步: 停止通信

为了停止数据传输，主设备将SCL切换为高电平，然后再将SDA切换为高电平，从而向从机发送停止条件，如下图所示：



### 单个主设备连接多个从机

I2C单个主设备连接多个从机。使用7位地址可提供128 (2的7次方)个唯一地址。使用10位地址很罕见，但可提供1024 (2的10次方)个唯一地址。要将多个节点连接到单个主器件，请使用4.7 kΩ上拉电阻连接这些节点，并将SDA和SCL线连接到VCC，如下图所示：



### 多个主设备连接多个从机

多个主设备可以连接到一个或多个从机。

当两个主设备试图通过SDA线路同时发送或接收数据时，同一系统中的多个主设备就会出现问题。

为了解决这个问题，每个主设备都需要在发送消息之前检测SDA线是低电平还是高电平；

* 如果SDA线为低电平，则意味着另一个主设备可以控制总线，并且主设备应等待发送消息；
* 如果SDA线为高电平，则可以安全地发送消息。



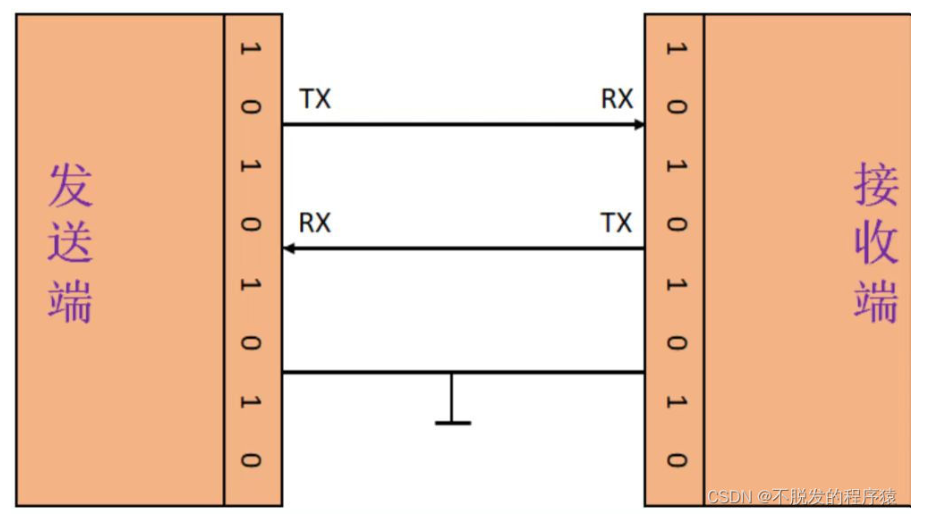
[(89条消息) 一文搞懂I2C通信总线\_i2c通信的详细讲解\_不脱发的程序猿的博客-CSDN博客](https://blog.csdn.net/m0_38106923/article/details/123673285)

# UART

## URART介绍

UART （Universal Asynchronous Receiver/Transmitter，即通用异步收发器）串行通信是单片机最常 用的一种通信技术，通常用于单片机和电脑之间以及单片机和单片机之间的通信。

UART是一种双向、串行、异步的通信总线，仅用一根数据接收线和一根数据发送线就能实现全双工通信。典型的串口通信使用3根线完成，分别是：发送线（TX）、接收线（RX）和地线（GND），通信时必须将双方的TX和RX交叉连接并且GND相连才可正常通信，如下图所示：

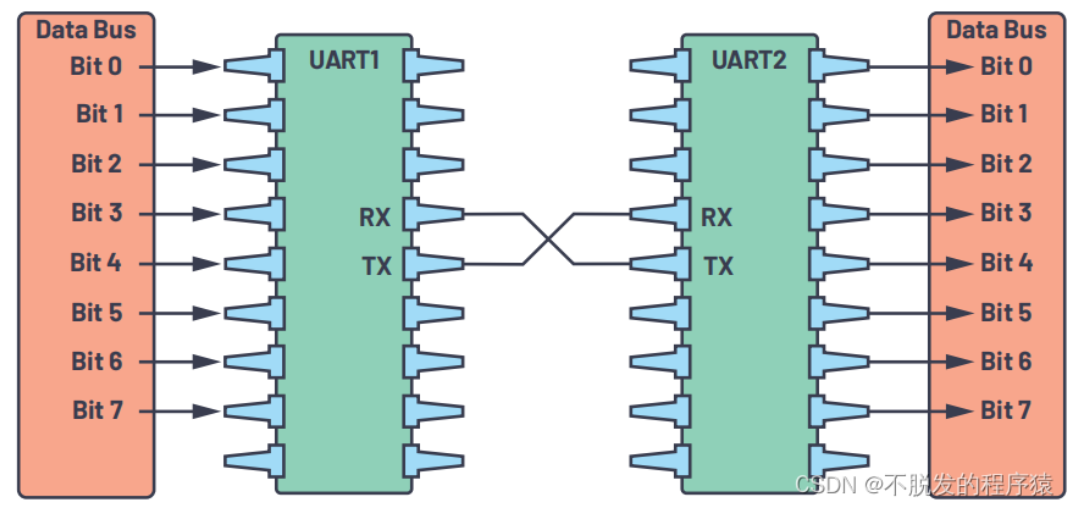


## UART特性

UART 接口不使用时钟信号来同步发送器和接收器设备，而是以异步方式传输数据。发送器**根据其时钟信号**生成的位流取代了时钟信号，接收器使用其内部时钟信号对输入数据进行采样。

同步点是通过两个设备的相同波特率（UART和大多数串行通信一样，发送和接收设备需要将波特率设置为相同的值。对于串行端口，设定的波特率将用作每秒传输的最大位数）来管理的。

如果波特率不同，发送和接收数据的时序可能会受影响，导致数据处理过程出现不一致。允许的波特率差异最大值为10%，超过此值，位的时序就会脱节。



**下总结了关于UART必须了解的几点：**

|  |  |
| --- | --- |
| 导线数量 | 3根（TX、RX和GND） |
| 速度 | 1200、2400、4800、9600、19200、38400、57600、115200等 |
| 传输方式 | 全双工异步 |
| 最大主机数量 | 1 |
| 最大从机数量 | 1 |

## UART协议帧

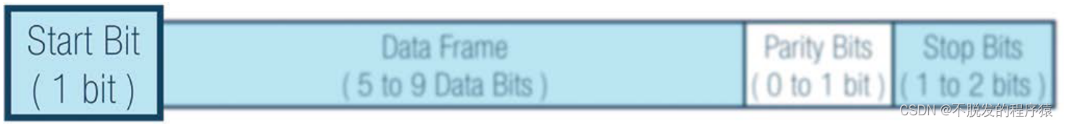
在 UART中，传输模式为数据包形式。数据包由起始位、数据帧、奇偶校验位和停止位组成。



### 起始位

当不传输数据时， UART 数据传输线通常保持高电压电平。若要开始数据传输，发送UART 会将传输线从高电平拉到低电平并保持1 个时钟周期。

当接收 UART 检测到高到低电压跃迁时，便开始以波特率对应的频率读取数据帧中的位。



### 数据位

数据帧包含所传输的实际数据。如果使用奇偶校验位，数据帧长度可以是5 位到 8 位。如果不使用奇偶校验位，数据帧长度可以是9 位。

在大多数情况下，数据以最低有效位优先方式发送。



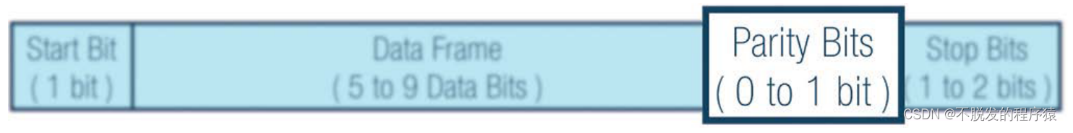
### 奇偶校验位

奇偶性描述数字是偶数还是奇数。通过奇偶校验位，接收 UART判断传输期间是否有数据发生改变。电磁辐射、不一致的波特率或长距离数据传输都可能改变数据位。

校验位可以配置成 1 位偶校验或 1 位奇校验或无校验位。

接收UART 读取数据帧后，将统计数值为 1 的位，检查总数是偶数还是奇数。如果奇偶校验位为0 （偶数奇偶校验），则数据帧中的1或逻辑高位总计应为偶数。如果奇偶校验位为 1 （奇数奇偶校验），则数据帧中的1 或逻辑高位总计应为奇数。

当奇偶校验位与数据匹配时，UART 认为传输未出错。但是，如果奇偶校验位为0 ，而总和为奇数，或者奇偶校验位为 1 ，而总和为偶数，则UART 认为数据帧中的位已改变。



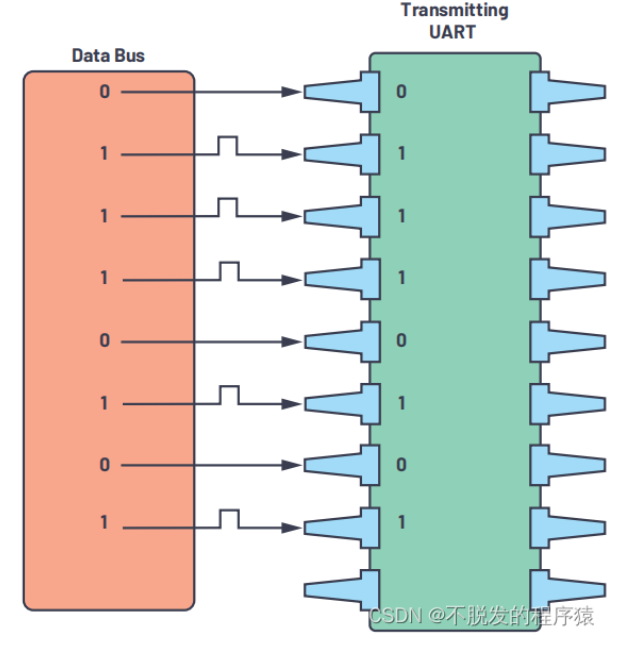
### 停止位

为了表示数据包结束，发送 UART 将数据传输线从低电压驱动到高电压并保持1 到 2 位时间。

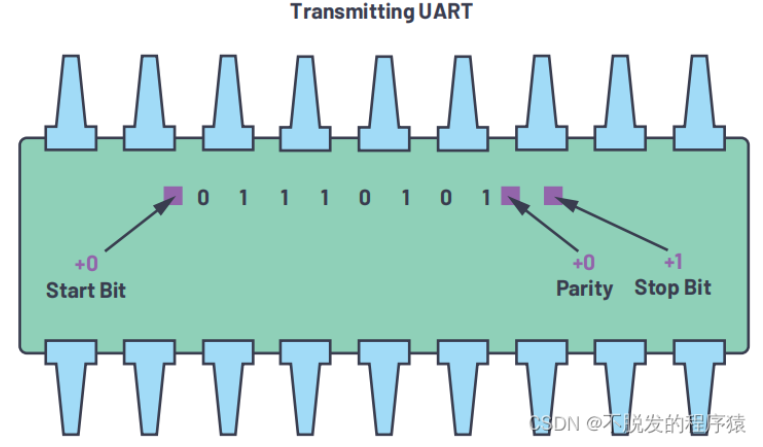


## UART通信步骤

第1步：数据从数据总线到发送器。



第2步：发送 UART 将起始位、奇偶校验位和停止位添加到数据帧。



第3步：从起始位到结束位，整个数据包以串行方式从发送器送至接收 器 。

接收 UART 以预配置的波特率对数据线进行采样。

A line with numbers and lines

Description automatically generated

第4步：接收 器 丢弃数据帧中的起始位、奇偶校验位和停止位。

A diagram of data frame

Description automatically generated

第5步：接收 器 将串行数据转换回并行数据，并将其传输到接收端的数据总线。

A diagram of a data bus

Description automatically generated

# SPI

## 什么是SPI

SPI，是一种高速的，全双工，同步的通信总线，并且在芯片的管脚上只占用四根线，节约了芯片的管脚，同时为PCB的布局上节省空间，提供方便，主要应用在 EEPROM，FLASH，实时时钟，AD转换器，还有数字信号处理器和数字信号解码器之间。

## 主从模式:

SPI分为主、从两种模式，一个SPI通讯系统需要包含一个（且只能是一个）主设备，一个或多个从设备。提供时钟的为主设备（Master），接收时钟的设备为从设备（Slave），SPI接口的读写操作，都是由主设备发起。当存在多个从设备时，通过各自的片选信号进行管理。

**SPI是全双工且SPI没有定义速度限制，一般的实现通常能达到甚至超过10 Mbps**

## SPI信号线

SPI接口一般使用四条信号线通信：

SDI（数据输入），SDO（数据输出），SCK（时钟），CS（片选）

MISO： 主设备输入/从设备输出引脚。该引脚在从模式下发送数据，在主模式下接收数据。

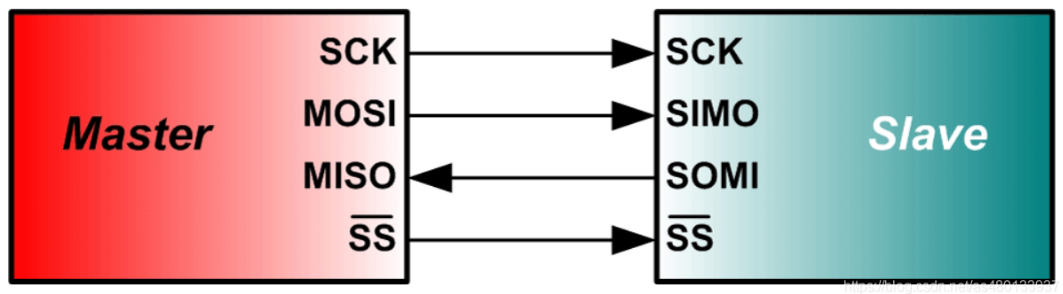
MOSI： 主设备输出/从设备输入引脚。该引脚在主模式下发送数据，在从模式下接收数据。

SCLK：串行时钟信号，由主设备产生。

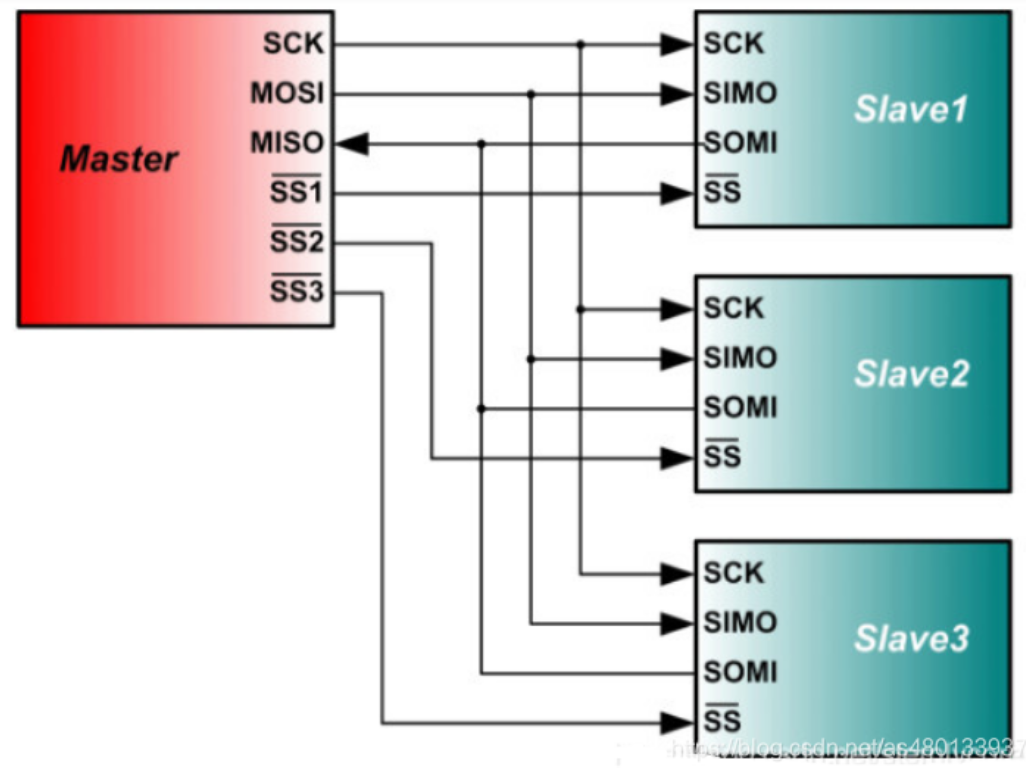
CS/SS：从设备片选信号，由主设备控制。它的功能是用来作为“片选引脚”，也就是选择指定的从设备，让主设备可以单独地与特定从设备通讯，避免数据线上的冲突。

硬件上为4根线。

## SPI 1v1:

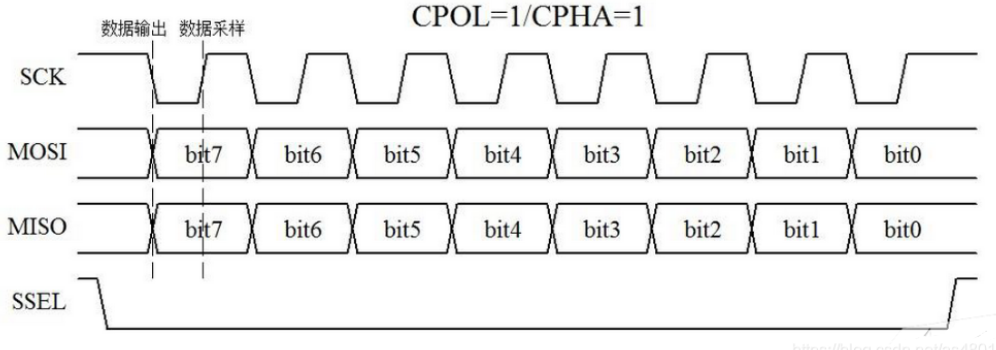


## SPI 1 v XX



## SPI设备选择

SPI是［单主设备（ single-master ）］通信协议，这意味着总线中的只有一支中心设备能发起通信。当SPI主设备想读/写［从设备］时，它首先拉低［从设备］对应的SS线（SS是低电平有效），接着开始发送工作脉冲到时钟线上，在相应的脉冲时间上，［主设备］把信号发到MOSI实现“写”，同时可对MISO采样而实现“读”，如下图：



## SPI数据发送接收

SPI主机和从机都有一个**串行移位寄存器**，主机通过向它的SPI串行寄存器写入一个字节来发起一次传输。

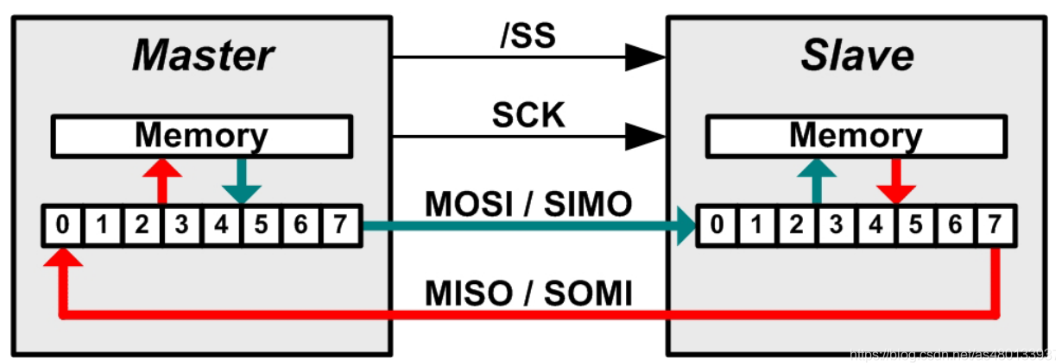
首先**拉低对应SS信号线**，表示与该设备进行通信

主机通过发送SCLK时钟信号，来告诉从机写数据或者读数据

**注意**，SCLK时钟信号可能是低电平有效，也可能是高电平有效，因为SPI有四种模式，这个我们在下面会介绍

主机(Master)将要发送的数据写到发送数据缓存区(Menory)，缓存区经过移位寄存器(0~7)，串行移位寄存器通过MOSI信号线将字节一位一位的移出去传送给从机，，同时MISO接口接收到的数据经过移位寄存器一位一位的移到接收缓存区。

从机(Slave)也将自己的串行移位寄存器(0~7)中的内容通过MISO信号线返回给主机。同时通过MOSI信号线接收主机发送的数据，这样，两个移位寄存器中的内容就被交换。



SPI只有主模式和从模式之分，没有读和写的说法，外设的写操作和读操作是同步完成的。如果只进行写操作，主机只需忽略接收到的字节；反之，若主机要读取从机的一个字节，就必须发送一个空字节来引发从机的传输。也就是说，你发一个数据必然会收到一个数据；你要收一个数据必须也要先发一个数据。

## SPI通信的四种模式

**SPI的四种模式，简单地讲就是设置SCLK时钟信号线的那种信号为有效信号**

**SPI通信有4种不同的操作模式**，不同的从设备可能在出厂是就是配置为某种模式，这是不能改变的；但我们的通信双方必须是工作在同一模式下，所以我们可以对我们的主设备的SPI模式进行配置，通过CPOL（时钟极性）和CPHA（时钟相位）来控制我们主设备的通信模式，具体如下：

**时钟极性(CPOL)定义了时钟空闲状态电平：**

CPOL=0，表示当SCLK=0时处于空闲态，所以有效状态就是SCLK处于高电平时

CPOL=1，表示当SCLK=1时处于空闲态，所以有效状态就是SCLK处于低电平时

**时钟相位(CPHA)定义数据的采集时间。**

CPHA=0，在时钟的第一个跳变沿（上升沿或下降沿）进行数据采样。，在第2个边沿发送数据

CPHA=1，在时钟的第二个跳变沿（上升沿或下降沿）进行数据采样。，在第1个边沿发送数据.

**例如：**

Mode0：CPOL=0，CPHA=0：此时空闲态时，SCLK处于低电平，数据采样是在第1个边沿，也就是SCLK由低电平到高电平的跳变，所以数据采样是在上升沿(准备数据），（发送数据）数据发送是在下降沿。

Mode1：CPOL=0，CPHA=1：此时空闲态时，SCLK处于低电平，数据发送是在第1个边沿，也就是SCLK由低电平到高电平的跳变，所以数据采样是在下降沿，数据发送是在上升沿。

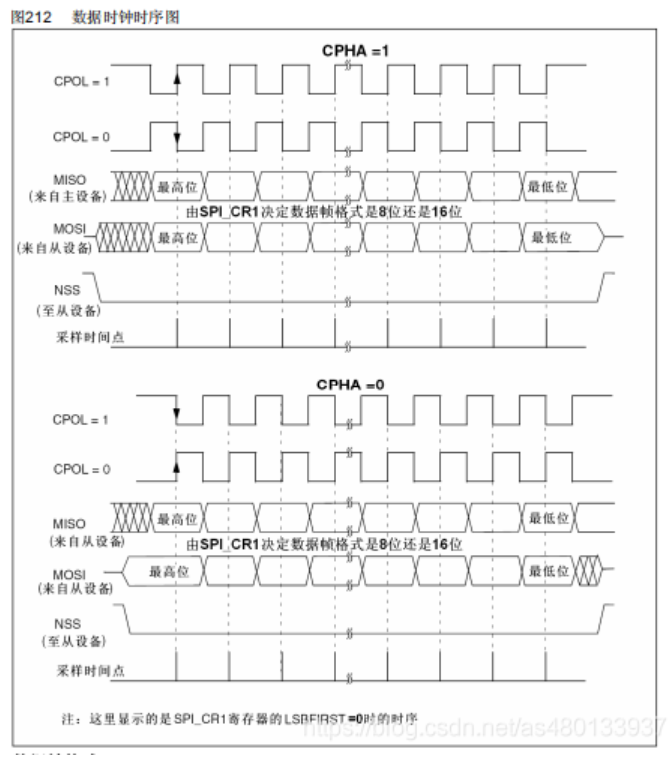
Mode2：CPOL=1，CPHA=0：此时空闲态时，SCLK处于高电平，数据采集是在第1个边沿，也就是SCLK由高电平到低电平的跳变，所以数据采集是在下降沿，数据发送是在上升沿。

Mode3：CPOL=1，CPHA=1：此时空闲态时，SCLK处于高电平，数据发送是在第1个边沿，也就是SCLK由高电平到低电平的跳变，所以数据采集是在上升沿，数据发送是在下降沿。



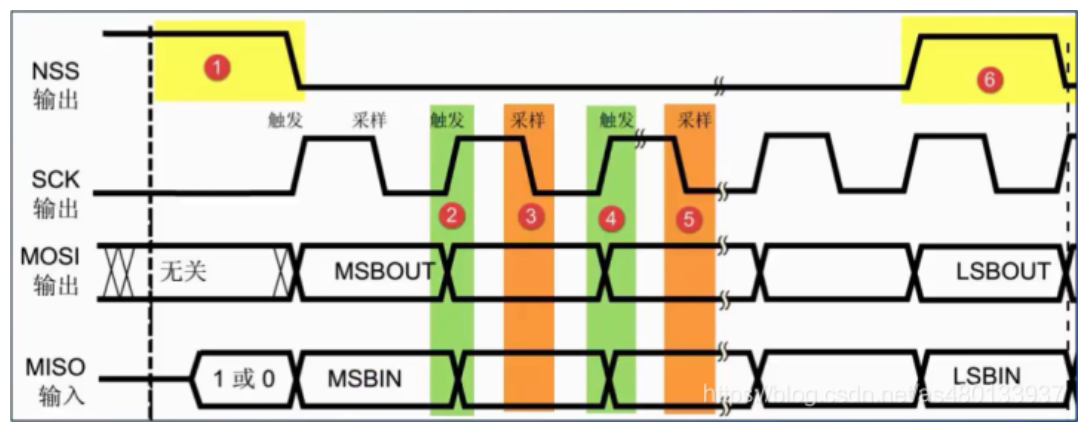
A diagram of a graph

Description automatically generated



**它们的区别是定义了在时钟脉冲的哪条边沿转换（toggles）输出信号，哪条边沿采样输入信号，还有时钟脉冲的稳定电平值（就是时钟信号无效时是高还是低）。每种模式由一对参数刻画，它们称为时钟极（clock polarity）CPOL与时钟期（clock phase）CPHA。**

## SPI通讯协议



从设备必须使用**相同的工作模式**——SCLK、CPOL 和 CPHA，才能正常工作。如果有多个从设备，并且它们使用了不同的工作模式，那么主设备必须在读写不同从设备时需要重新修改对应从设备的模式。以上SPI总线协议的主要内容。

是不是感觉，这就完了？ SPI就是如此，他没有规定最大传输速率，没有地址方案，也没规定通信应答机制，没有规定流控制规则。

只要四根信号线连接正确，SPI模式相同，将CS/SS信号线拉低，即可以直接通信，一次一个字节的传输，读写数据同时操作，这就是SPI

些通信控制都得通过SPI设备自行实现，SPI并不关心物理接口的电气特性，例如信号的标准电压。

PS:

这也是SPI接口的一个缺点：没有指定的流控制，没有应答机制确认是否接收到数据。

## SPI的三种模式

**SPI工作在3中模式下，分别是运行、等待和停止。**

**运行模式（Run Mode）**

这是基本的操作模式

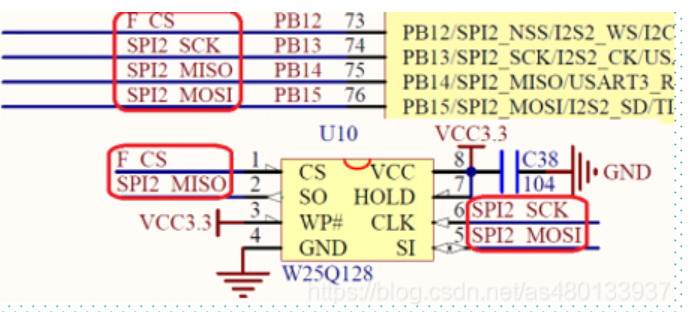
**等待模式（Wait Mode）**

SPI工作在等待模式是一种可配置的低功耗模式，可以通过SPICR2寄存器的SPISWAI位进行控制。在等待模式下，如果SPISWAI位清0，SPI操作类似于运行模式。如果SPISWAI位置1，SPI进入低功耗状态，并且SPI时钟将关闭。如果SPI配置为主机，所有的传输将停止，但是会在CPU进入运行模式后重新开始。如果SPI配置为从机，会继续接收和传输一个字节，这样就保证从机与主机同步。

**停止模式（Stop Mode）**

为了降低功耗，SPI在停止模式是不活跃的。如果SPI配置为主机，正在进行的传输会停止，但是在CPU进入运行模式后会重新开始。如果SPI配置为从机，会继续接受和发送一个字节，这样就保证了从机与主机同步。

## SPI原理图连接



## SPI: ESP32配置

见ESP32\_Note.