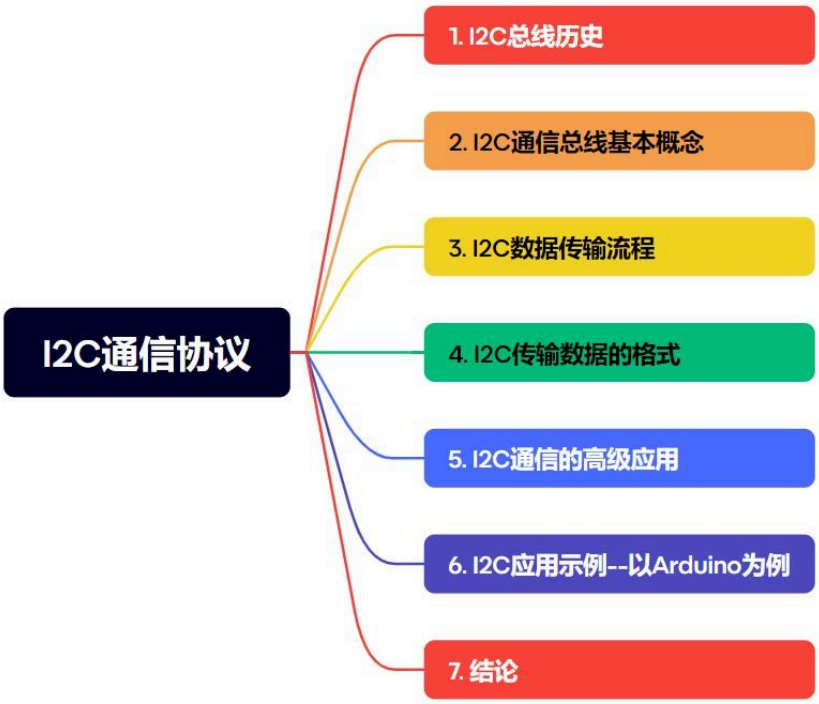
I2C总线和通信协议详解 (超详细配42张高清图+万字长文)



I2C（Inter-Integrated Circuit）通信总线，作为嵌入式系统设计中的一个关键组成部分，其灵活性和高效率使其在高级应用中备受青睐。本文旨在提供关于I2C通信总线的深度解析，包括其基本概念、特点、通信协议，以及在不同场景下的高级应用和最佳实践。I2C接口只有2根信号线，总线上可以连接多个设备，硬件实现简单，可扩展性强。I2C通信协议可以用普通GPIO引脚进行软件模拟。I2C接口主要用于通讯速率要求不高，以及多个器件之间通信的应用场景。

1. I2C总线历史

I2C（Inter-Integrated Circuit）总线是一种重要的串行通信协议，它的历史可以追溯到上世纪80年代初期。以下是对I2C总线历史的详细介绍：

1. **起源**：

- I2C总线技术由荷兰的飞利浦半导体（现在的恩智浦半导体）在1982年开发。最初，这项技术是为了在电视机内部实现简单、高效、低成本的通信而设计的。

1. **设计目标**：

- 设计I2C的初衷是减少电视机等复杂电子系统内部的布线数量，同时也降低制造成本。通过使用只有两根线的通信总线，它有效地减少了器件间连接的复杂性。

1. **技术发展**：

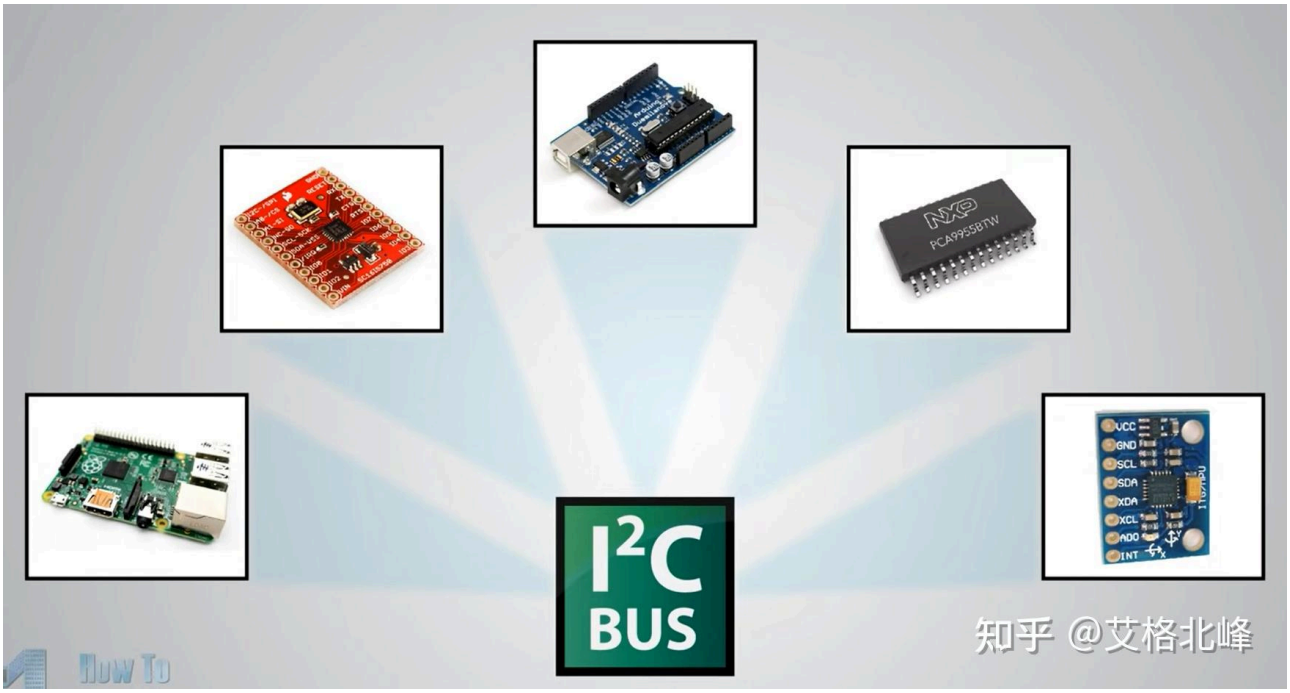
- 随着技术的成熟和普及，I2C协议得到了广泛的应用和扩展。从最初的标准模式（100kHz），发展到快速模式（400kHz）和高速模式（3.4MHz）。

1. **标准化和开放**：

- 虽然最初由飞利浦半导体开发，但I2C协议后来被标准化并广泛应用于多种设备中。飞利浦半导体放弃了对这项技术的专利权，使其成为开放标准。

1. **广泛应用**：

- I2C技术由于其简单性和有效性，已成为嵌入式系统设计中不可或缺的一部分。



2. I2C通信总线基本概念

I2C是一种多主机、两线制、低速串行通信总线，广泛用于微控制器和各种外围设备之间的通信。它使用两条线路：串行数据线（SDA）和串行时钟线（SCL）进行双向传输。

特点

1. **两线制总线**：I2C仅使用两条线——串行数据线（SDA）和串行时钟线（SCL）进行通信，有效降低了连接复杂性。
2. **多主多从设备支持**：I2C支持多个主设备和多个从设备连接到同一总线上。每个设备都有唯一的地址。
3. **可变的时钟速率**：I2C总线支持不同的速率模式，如标准模式（100kbps）、快速模式（400kbps）和高速模式（3.4Mbps）。
4. **同步通信**：I2C是一种同步通信协议，数据传输由时钟信号（SCL）来控制。
5. **简单的连接**：I2C通信对硬件的要求比较低，很容易在微控制器和外围设备间实现连接。
6. **地址分配**：每个I2C设备都通过一个7位或10位的地址来识别，这使得总线上可以连接多个设备。
7. **阻塞传输**：I2C支持阻塞传输机制，即主设备可以在传输过程中控制总线，防止其他设备发送数据。
8. **应用广泛**：由于其简单和灵活性，I2C被广泛应用于各种电子产品中，如传感器、LCD显示器、EEPROM等。
9. **总线仲裁和冲突检测**：在多主模式下，I2C能够处理多个主设备同时尝试控制总线的情况。
10. **低功耗**：I2C总线的设计使其成为低功耗的通信方式，适用于电池供电的设备。

基本特征

1. **总线结构**：

- **两线制**：使用两条线进行通信，分别是串行数据线（SDA）和串行时钟线（SCL）。
- **多主多从结构**：支持多个主设备和多个从设备连接到同一总线上。

1. **通信方式**：

- **同步串行**：数据传输同步于时钟信号。
- **字节格式**：每个字节由8位数据构成，加上开始和停止条件以及可选的应答位。

1. **时钟速率**：

- 支持多种速率，包括标准模式（100kbps）、快速模式（400kbps）和高速模式（3.4Mbps）。

工作原理

1. **总线控制**：

- **开始和停止条件**：通信由主设备通过在SDA线上生成特定的信号模式来开始和结束。
- **地址帧**：每次通信开始时，主设备发送一个地址帧来指定与之通信的从设备。

1. **数据传输**：

- **主从通信**：主设备控制时钟信号，向从设备发送或接收数据。
- **应答位**：每个字节后，接收方

发送一个应答位（ACK）或非应答位（NACK），以告知发送方数据是否被成功接收。

地址和仲裁

1. **设备地址**：

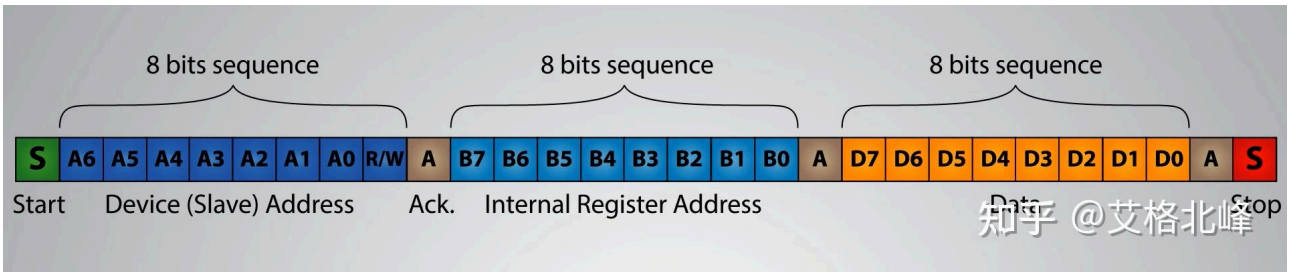
- **7位或10位地址**：每个I2C设备都有一个唯一的地址，允许在同一总线上连接多个设备。

1. **总线仲裁**：

- 在多主模式下，当两个主设备同时尝试控制总线时，I2C协议包含仲裁机制以决定哪个设备获得控制权。

3. I2C数据传输流程

数据信号以8位的序列传输。所以在特殊的开始条件发生后，就会出现第一个8位序列，它指示了数据被发送到哪个从设备的地址。每个8位序列之后都会跟随一个称为确认的位。

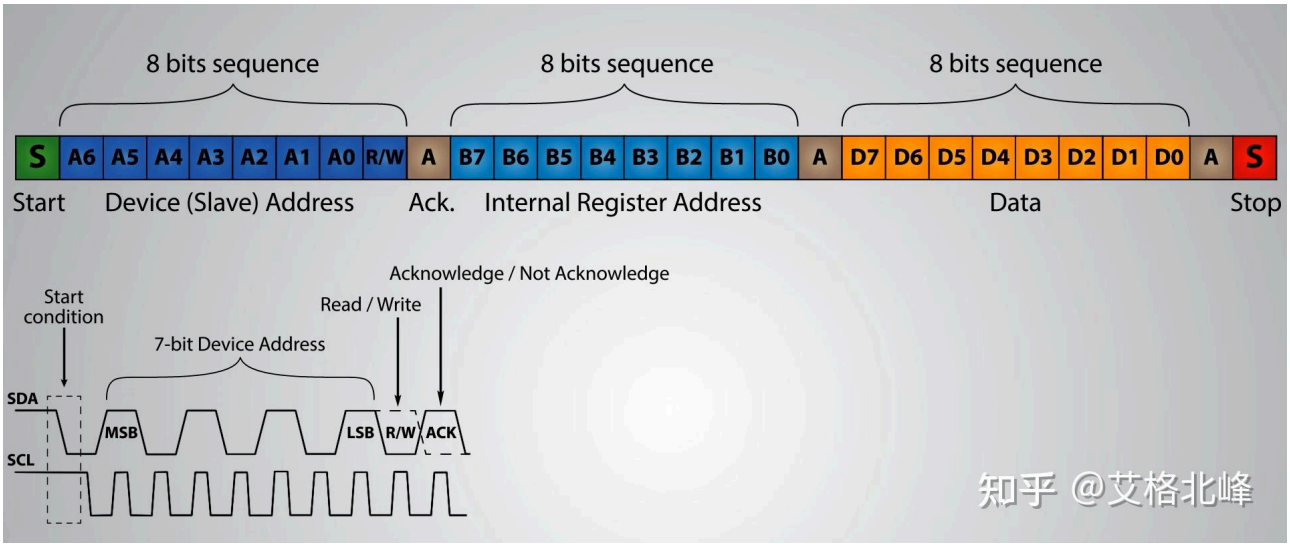


在大多数情况下，第一个确认位之后会跟着另一个寻址序列，但这次是针对从设备的内部寄存器。在寻址序列之后是数据序列，直到数据完全传输完毕，并以一个特殊的停止条件结束。

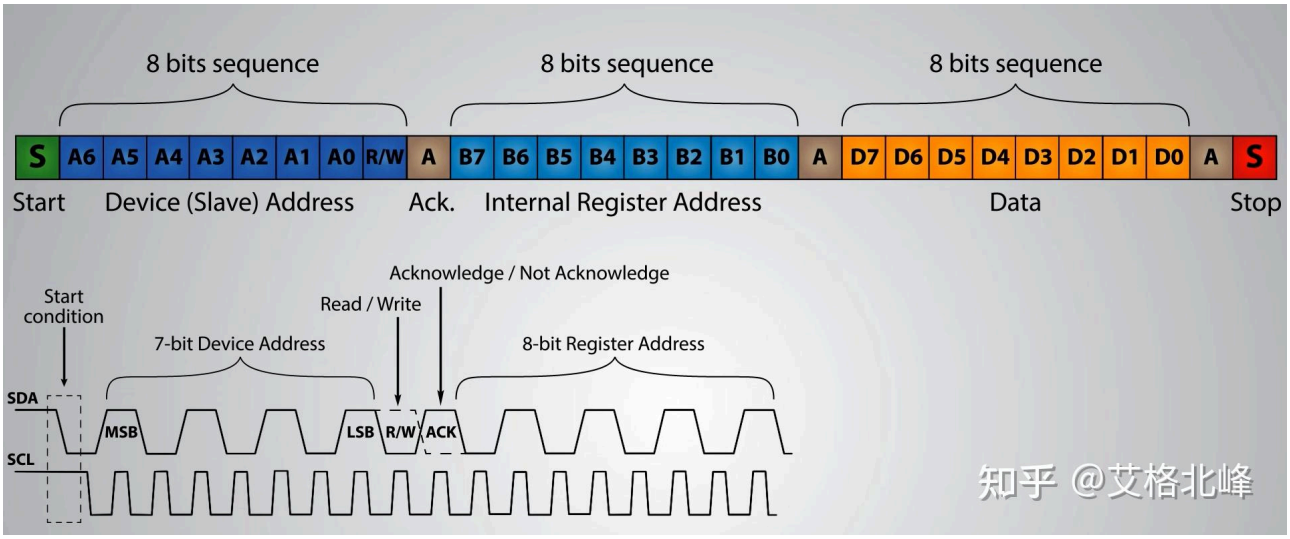
开始条件发生在数据线在时钟线仍然高电平的时候变低。之后，时钟开始，并且在每个时钟脉冲期间传输每一位数据。设备寻址序列从最重要的位开始，以最不重要的位结束，实际上是由7位组成的，因为第8位用于指示主设备是向从设备写入（逻辑低）还是从中读取（逻辑高）。

下一个确认位由从设备用来指示它是否成功接收了前一个位序列。所以这次主设备将SDA线的控制权交给从设备，如果从设备成功接收了前一个序列，它将把SDA线拉低到所谓的确认状态。

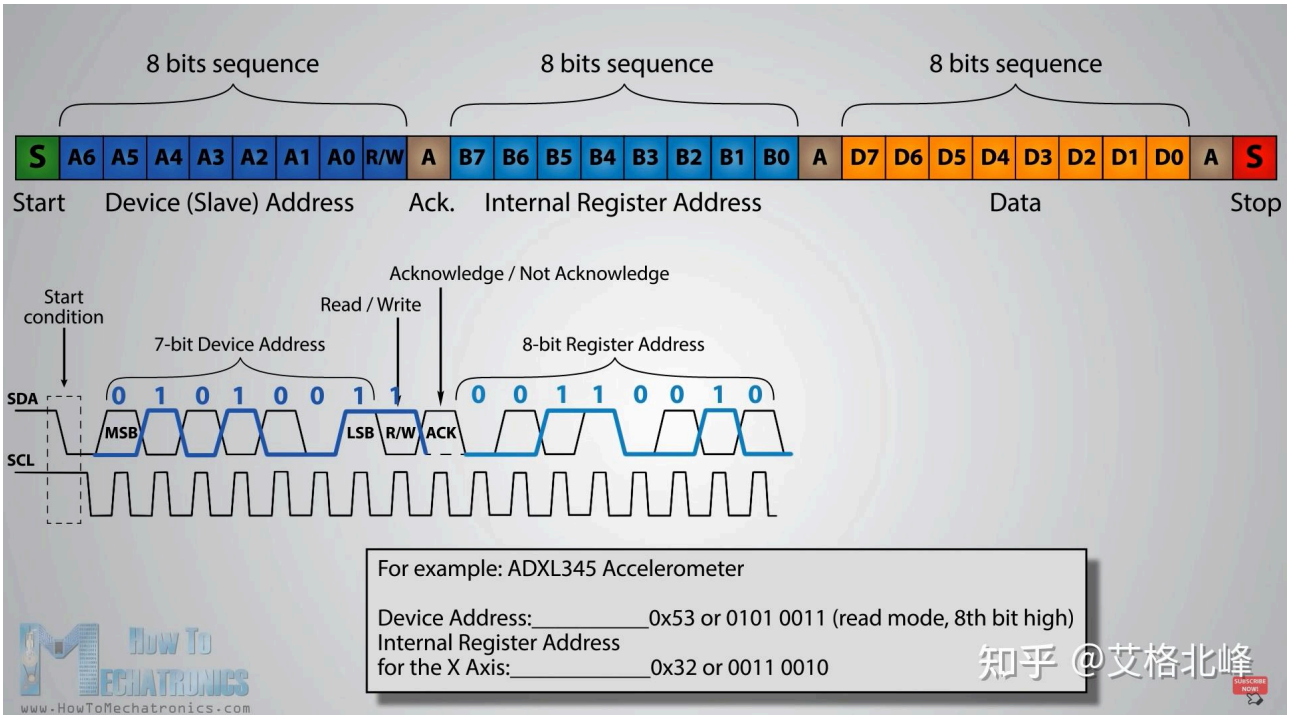
如果从设备没有把SDA线拉低，这种状态被称为不确认，意味着它没有成功接收前一个序列，这可能由多种原因造成。例如，从设备可能正忙，可能不理解接收到的数据，或者不能再接收任何数据等等。 在这种情况下，主设备决定如何继续操作。



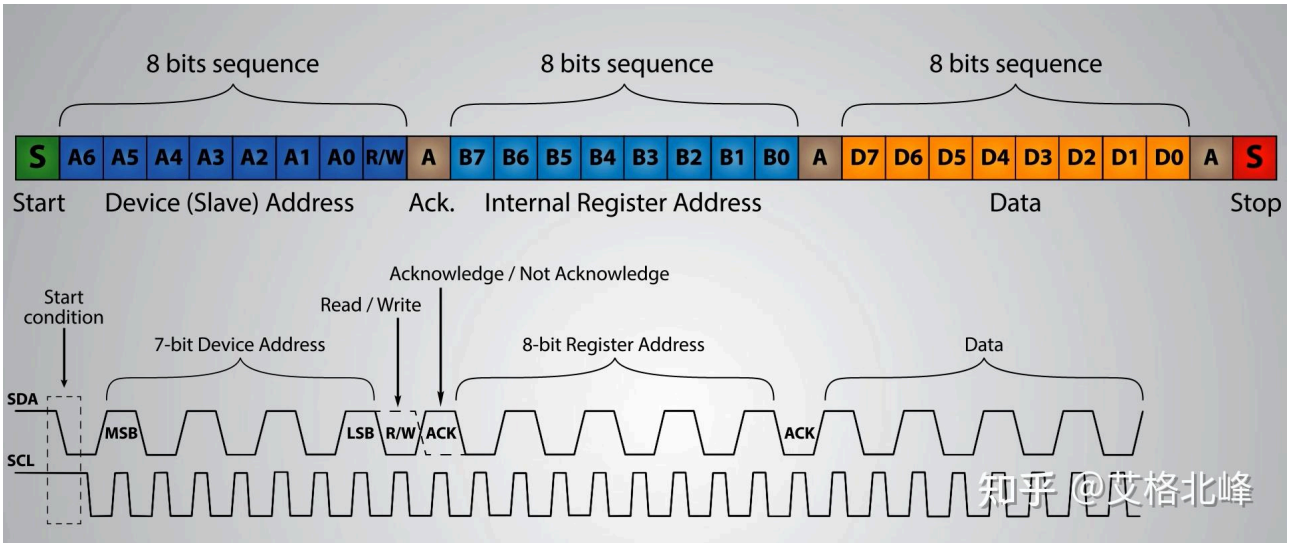
接下来是内部寄存器的寻址。内部寄存器是从设备内存中包含各种信息或数据的位置。



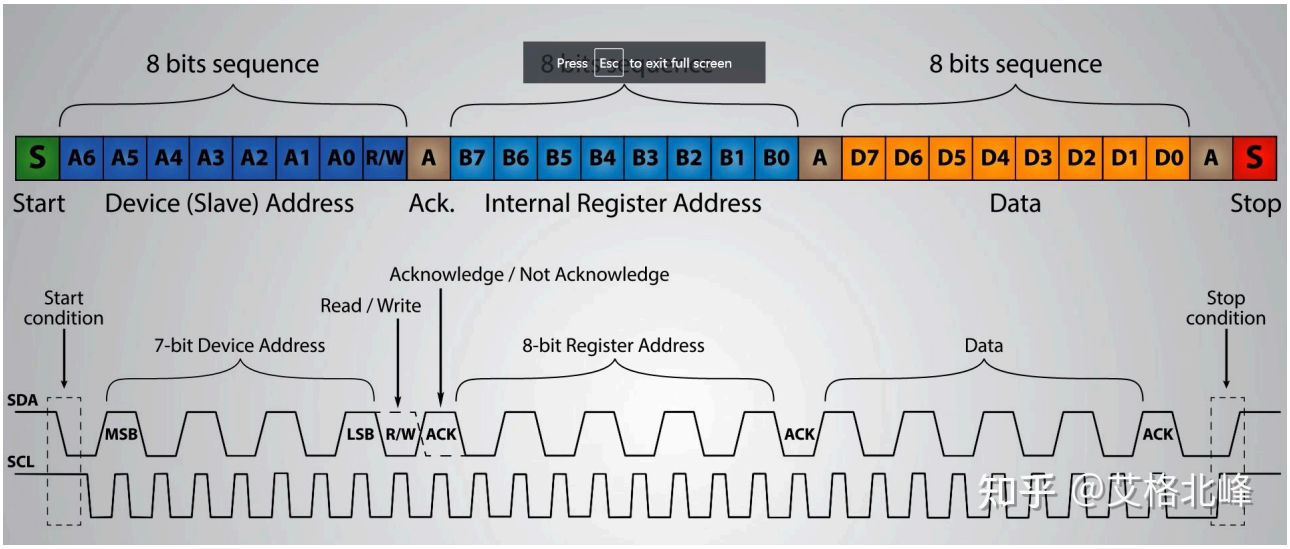
例如，ADXL345加速度计有一个独特的设备地址和额外的内部寄存器地址，用于X、Y和Z轴。 因此，如果我们首先想读取x轴的数据，我们需要发送设备地址，然后发送x轴的特定内部寄存器地址。这些地址可以从传感器的数据手册中找到。



在寻址之后，数据传输序列开始，要么来自主设备，要么来自从设备，这取决于在读/写位选择的模式。



在数据完全发送之后，传输将以停止条件结束，当SDA线在SCL线高电平时从低变高。这就是I2C通信协议的工作原理。

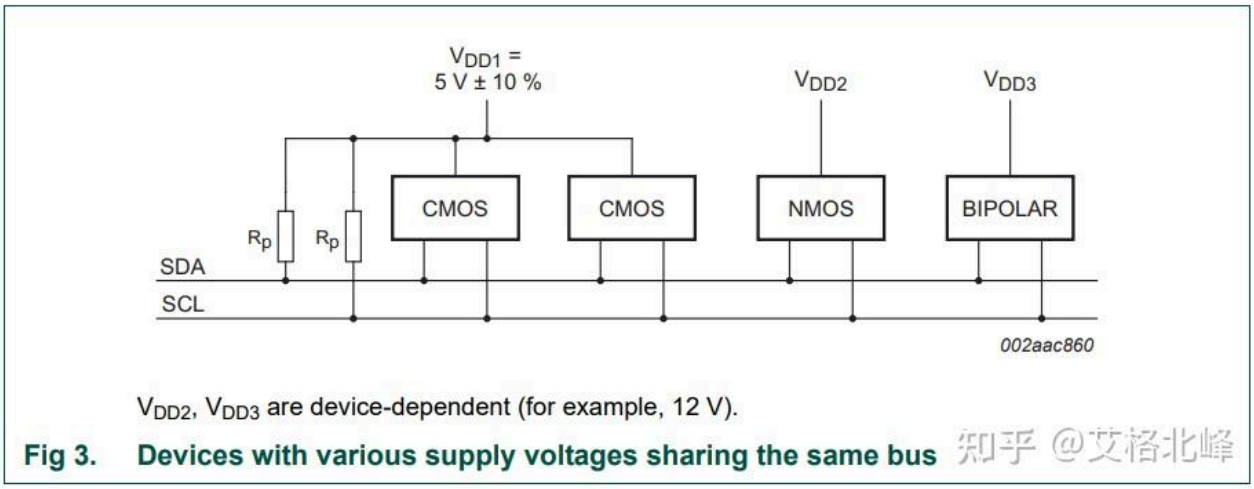


上述内容出现了很多特定概念，我们下面来分别解释他们：

1. SDA和SCL信号

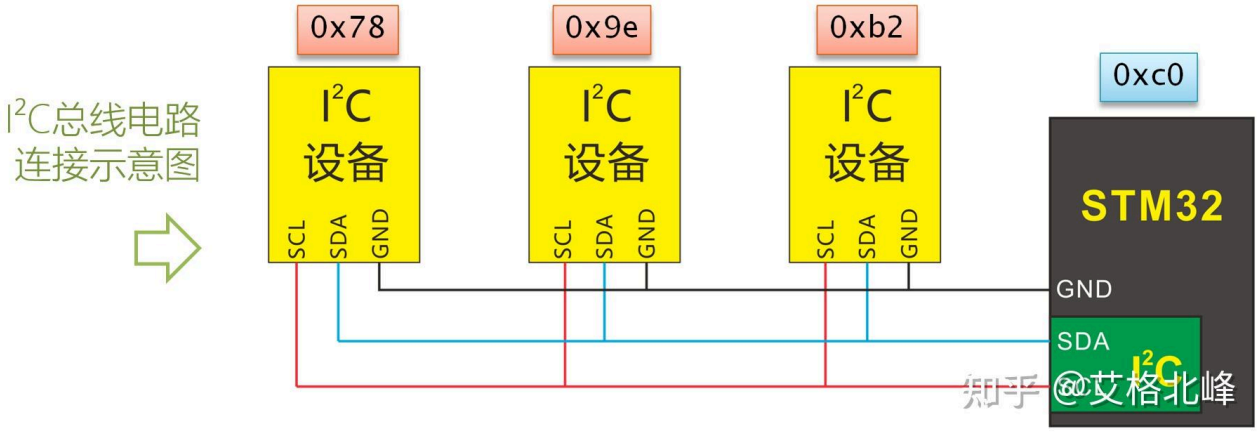
SDA和SCL都是双向线路，通过电流源或上拉电阻连接到正电源电压(见图3)。 当总线空闲时，两条线路都是HIGH。连接到总线的设备的输出级必须具有开漏极或开集电极来执行有线与功能。I2C总线上的数据可以在标准模式下以高达100 kbit/s的速度传输，在快速模式下可达400 kbit/s，在快速模式+中可达1 Mbit/s，或在高速模式下可达3.4 Mbit/s。总线

电容限制了连接到总线的接口数量。 对于单个主应用程序，如果总线上没有设备会拉伸时钟，主SCL输出可以是推挽驱动器设计。



V_{DD2}, V_{DD3} are device-dependent (for example, 12 V).

Fig 3. Devices with various supply voltages sharing the same bus



2. 数据有效性

SDA线上的数据必须在时钟HIGH期间保持稳定。只有当SCL线上的时钟信号为LOW时，数据线的HIGH或LOW状态才能改变(见图4)。传输的每个数据位都会产生一个时钟脉冲。

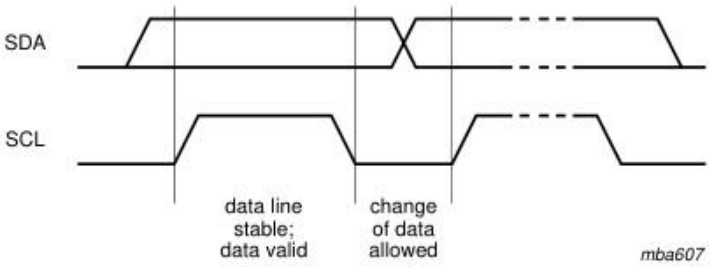


Fig 4. Bit transfer on the I²C-bus

3. START和STOP条件

所有事务都以START(S)开始，并由STOP(P)终止(参见图5)。 SDA线上SCL为HIGH时，HIGH到LOW的转换定义了一个START条件。 SDA线上SCL为HIGH时，LOW到HIGH的转换定义了一个STOP条件。

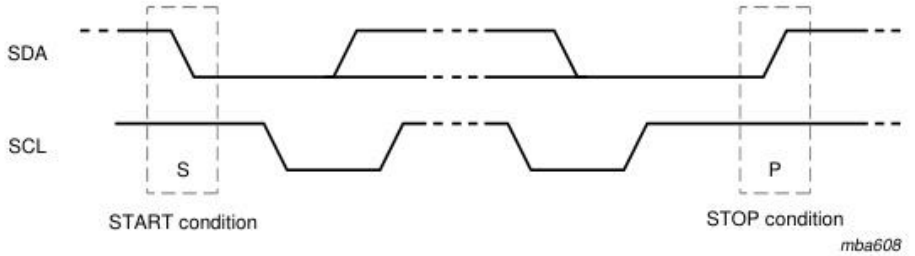


Fig 5. START and STOP conditions

START和STOP条件总是由主设备产生。在START条件之后，总线被认为是忙碌的。在STOP条件之后的某个时间，总线被认为是空闲的。

如果产生了重复的START(Sr)而不是STOP条件，总线保持忙碌。在这方面，START(S)和重复的START(Sr)条件在功能上是相同的。

因此，对于本文档的其余部分，S符号被用作代表START和重复的START条件的通用术语，除非Sr是特别相关的。

如果连接到总线的设备合并了必要的接口硬件，则检测START和STOP条件是容易的。然而，没有这种接口的微控制器必须在每个时钟周期内对SDA线采样至少两次，以感知转换。

4. 字节格式

每一个放在SDA线上的字节必须是8位长。每次传输可以传输的字节数是没有限制的。每一个字节后面必须跟一个确认位。数据以最有效位(MSB)为首进行传输(见图6)。如果一个从机在执行其他一些功能之前不能接收或传输另一个完整的字节数据，例如处理一个内部中断，它可以保持时钟线SCL LOW，迫使主机进入等待状态。当从机准备好接收另一个字节数据并释放时钟线SCL时，数据传输继续进行。

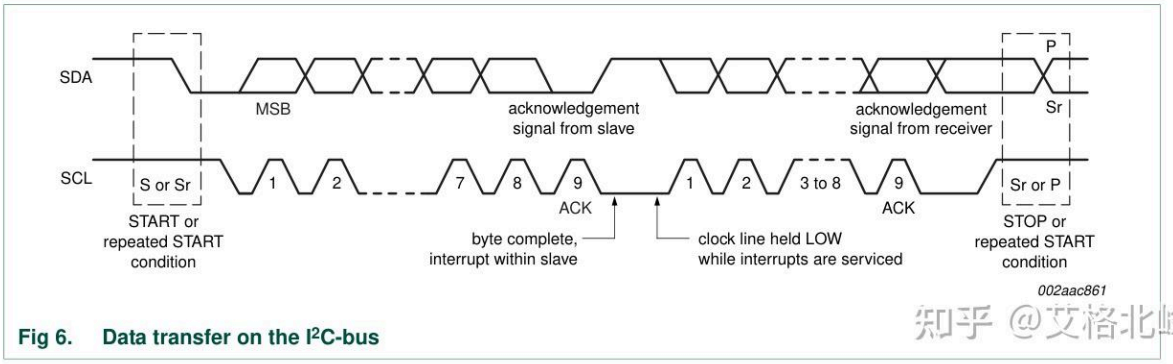


Fig 6. Data transfer on the I²C-bus

5. Acknowledge(ACK)和Not Acknowledge(NACK)

确认发生在每个字节之后。确认位允许接收端向发送端发出信号，表示字节被成功接收，可以发送另一个字节。主设备产生所有的时钟脉冲，包括确认的第九个时钟脉冲。

确认信号定义如下：发送端在确认时钟脉冲期间释放SDA线，这样接收端就可以拉SDA线LOW，并且在该时钟脉冲的HIGH期间保持稳定的LOW(参见图4)。设

当SDA在第九个时钟脉冲期间保持HIGH时，这被定义为不确认信号。主设备然后可以产生一个STOP条件来中止传输，或者重复的START条件来启动一个新的传输。有五个条件导致NACK的产生：

1. 没有接收器在总线上传输地址，所以没有设备响应确认。
2. 接收器无法接收或发送，因为它正在执行一些实时功能，并且还没有准备好与主服务器进行通信。
3. 在传输过程中，接收方收到了它无法理解的数据或命令。
4. 在传输过程中，接收方不能接收到任何更多的数据字节。
5. 主接收机必须向从发送机发出传输结束的信号。

6. 时钟同步

两个主控器可以同时在一个空闲总线上开始传输，必须有一种方法来决定哪个主控器控制总线并完成传输。这是通过时钟同步和仲裁来实现的。在单主控器系统中，时钟同步和仲裁是不需要的。

时钟同步是通过I2C接口到SCL线的有线与连接来实现的。这意味着SCL线上的HIGH到LOW转换会导致相关的主控器开始计数他们的LOW周期，一旦主控器时钟变为LOW，它会保持

SCL线处于该状态，直到时钟达到HIGH状态(见图7)。然而，如果另一个时钟仍然在它的LOW周期内，这个时钟的LOW到HIGH转换可能不会改变SCL线的状态。因此，SCL线被具有最长LOW周期的主控器保持为LOW。具有较短LOW周期的主控器在这段时间内进入HIGH等待状态。

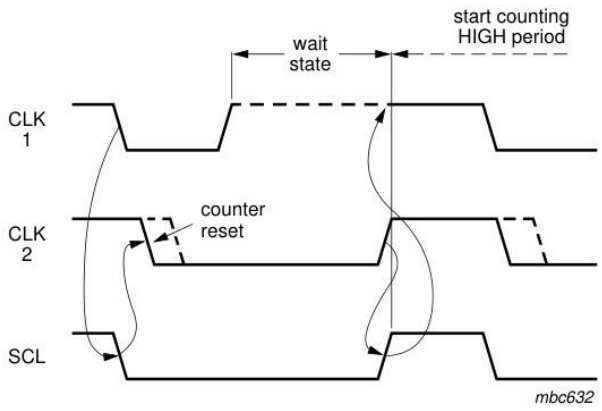


Fig 7. Clock synchronization during the arbitration procedure

当所有主控机都结束了低周期时，SCL线释放并变为高电平。此时主控机时钟与SCL线的状态没有区别，所有主控机开始计算它们的高周期。第一个完成高周期的主控机再次拉低SCL线。

这样，一个同步SCL时钟就产生了，它的低周期由低周期最长的主控机决定，而它的高周期由高周期最短的主控机决定。

7. 仲裁

仲裁，像同步一样，是指只有在系统中使用多个主设备时才需要的协议部分。从设备不参与仲裁过程。只有总线空闲时，主设备才可以开始传输。两个主设备可以在最小保持时间(tHD; STA)内产生一个START条件，这会导致总线上产生一个有效的START条件。然后需要仲裁来决定哪个主设备将完成它的传输。

仲裁逐位进行。在每个位期间，当SCL为HIGH时，每个主设备检查SDA电平是否与它所发送的相匹配。这个过程可能需要许多位。两个主设备实际上可以无误地完成整个事务，只要传输是相同的。第一次一个主设备试图发送HIGH，但检测到SDA电平为LOW，主设备知道它已经失去了仲裁并关闭SDA输出驱动器。另一个主设备继续完成它的事务。

在仲裁过程中没有信息丢失。一个失去仲裁的主设备可以产生时钟脉冲，直到它失去仲裁的字节结束，并继续产生时钟脉冲。必须在总线空闲时重新开始它的事务。

如果一个主设备也包含一个从设备功能，并且它在寻址阶段失去仲裁，有可能是获胜的主设备试图寻址它。因此，失败的主设备必须立即切换到它的从设备模式。

图8显示了两个主设备的仲裁过程。可能涉及更多内容，这取决于总线连接了多少主设备。当产生DATA1的主设备的内部数据电平与SDA线上的实际电平之间存在差异时，DATA1输出被关闭。这不影响由获胜的主设备发起的数据传输。

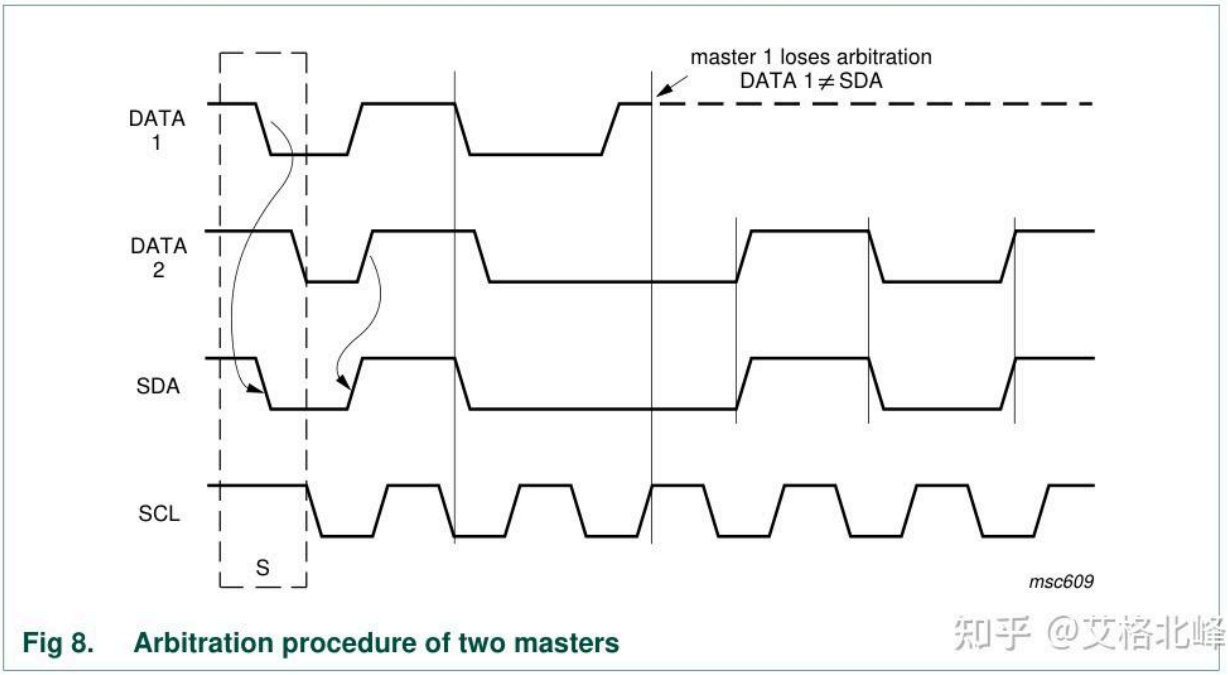


Fig 8. Arbitration procedure of two masters

由于I2C总线的控制完全由竞争主设备发送的地址和数据决定，所以没有中心主设备，总线上也没有任何优先顺序。 如果仲裁程序仍在进行，当一个主设备发送重复的START或STOP条件，而另一个主设备仍在发送数据时，则存在一个未定义的条件。换句话说，以下组合会导致一个未定义的条件：

- 主设备1发送重复的START条件，主设备2发送一个数据位。
- 主设备1发送STOP条件，主设备2发送一个数据位。
- 主设备1发送重复的START条件，主设备2发送一个STOP条件。

8. 时钟拉伸

时钟拉伸通过保持SCL线LOW暂停事务。事务无法继续，直到该线再次释放为HIGH。时钟拉伸是可选的，事实上，大多数从设备不包括SCL驱动程序，因此它们无法拉伸时钟。

在字节级，设备可能能够以较快的速度接收字节数据，但需要更多的时间来存储接收到的字节或准备传输另一个字节。从设备可以在接收和确认一个字节后保持SCL线LOW，以迫使主设备进入等待状态，直到从设备准备好在一种握手过程类型中进行下一个字节传输(见图7)。

在位级，设备如微控制器，具有或不具有I2C总线有限的硬件，可以通过延长每个时钟LOW周期来减慢总线时钟。任何主设备的速度都适应于该设备的内部运行速率。

9. 从地址和R/W位

数据传输遵循图9所示的格式。在START条件(S)之后，发送一个从地址。这个地址是7位长，后面跟着第八位，这是一个数据方向位(R/W)——“0”表示传输(WRITE)，“1”表示数据请求(READ)(参见图10)。数据传输总是由master生成的STOP条件(P)终止。然而，如果master仍然希望在总线上通信，它可以生成一个重复的START条件(Sr)并在没有首先生成STOP条件的情况下寻址另一个从设备。在这样的传输中，各种读/写格式的组合是可能的。

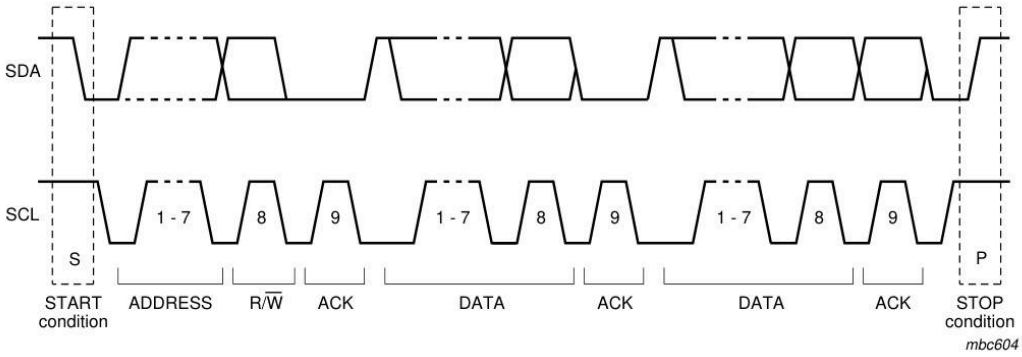


Fig 9. A complete data transfer

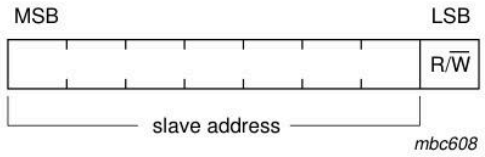


Fig 10. The first byte after the START procedure

10. 10位寻址

10位寻址扩展了可能的地址数。具有7位和10位地址的设备可以连接到同一个I2C总线，并且7位和10位寻址都可以在所有总线速度模式下使用。目前，10位寻址还没有被广泛使用。10位从属地址是由一个START条件(S)或重复的START条件(Sr)之后的前两个字节组成的。第一个字节的前7位是组合1111 0XX，其中最后两个位(XX)是10位地址的两个最有效位(MSB)；第一个字节的第八位是R/W位，它决定了消息的方向。 虽然有8个可能的保留地址位1111 XXX的组合， 但只有四个组合1111 0XX用于10位寻址。其余四个组合1111 1XX被保留用于未来的I2C总线增强。

所有先前描述的7位寻址的读/写格式组合都可能用10位寻址。这里详细介绍两种格式：

- 主发送器用一个10位从属地址向从属接收器发送。 传输方向不变(见图14)。当一个10位地址跟随一个START条件时，每个从属比较从属地址第一个字节的前7位(1111 0XX)与自己的地址，并测试第八位(R/W方向位)是否为0。可能有多个设备找到一个匹配并产生一个确认(A1)。所有找到匹配的从属比较从属地址第二个字节的八位(XXXX XXXX)与自己的地址，但只有一个从属找到一个匹配并产生一个确认(A2)。匹配的从属仍然由主寻址，直到它接收到一个STOP条件(P)或重复的START条件(Sr)，后面跟着一个不同的从属地址。
- 主接收器用一个10位从属地址读取从属发送器。 传输方向在第二个R/W位之后改变(图15)。直到并包括确认位A2，过程与前面描述的用于一个从属发送器的程序相同。主发送器寻址从接收器。在重复的START条件(Sr)之后，匹配的从设备记住它之前被寻址过。这个从设备然后检查Sr之后的从地址的第一个字节的前7位是否与它们在START条件(S)之后是相同的，并测试第八位(R/W)是否为1。 如果有匹配，从设备认为它被作为一个发送器寻址，并产生确认A3。从发送器保持寻址状态，直到它接收到一个STOP条件(P)或接收到另一个重复的START条件(Sr)后跟随一个不同的从地址。在重复的START条件(Sr)之后，所有其他从设备也将比较从地址(1111 0XX)的第一个字节的前7位与它们自己的地址，并测试第八位(R/W)。 然而，它们中没有一个被寻址，因为R/W=1(10位设备)，或者1111 0XX从地址(7位设备)不匹配。

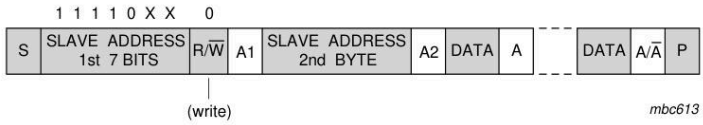


Fig 14. A master-transmitter addresses a slave-receiver with a 10-bit address

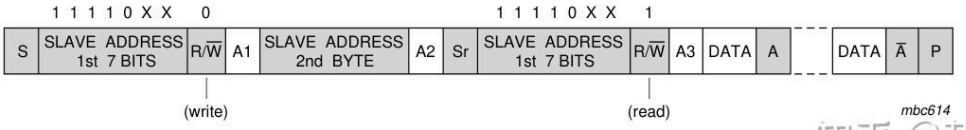


Fig 15. A master-receiver addresses a slave-transmitter with a 10-bit address

具有10位寻址的从设备对“通用调用”的反应与具有7位寻址的从设备相同。硬件主设备可以在“通用调用”后传输其10位地址。在这种情况下，“通用调用”地址字节后面跟着两个连续的字节，其中包含主发送器的10位地址。格式如图15所示，第一个数据字节包含主地址的最低有效位8位。

开始字节0000 0001 (01h)可以以与7位地址相同的方式出现在10位地址之前。

11. 通用调用地址

通用调用地址用于同时寻址连接到I2C总线的每个设备。然而，如果一个设备不需要通用调用结构中提供的任何数据，它可以通过不发出确认来忽略这个地址。如果一个设备确实需要来自通用调用地址的数据，它会确认这个地址并表现为从接收器。如果一个或多个设备响应，主设备实际上不知道有多少设备确认。第二个字节和后续字节被每一个能够处理此数据的从接收器确认。一个不能处理这些字节之一的从设备必须通过不确认来忽略它。同样，如

果一个或多个从设备确认，主设备将不会看到不确认。通用调用地址的含义总是在第二个字节中指定(见图16)。

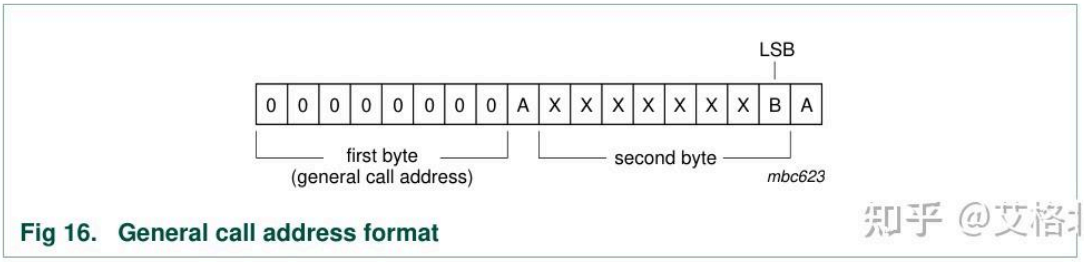


Fig 16. General call address format

有两种情况需要考虑：

- 当最低有效位B为“0”时。
- 当最低有效位B为“1”时。

当位B为“0”时，第二个字节有以下定义：

- 0000 0110 (06h)：硬件复位并写入从地址的可编程部分。在接收到这个2字节序列时，所有设计用于响应通用调用地址的设备都复位并接收其地址的可编程部分。 必须采取预防措施，以确保设备在施加电源电压后没有拉下SDA或SCL线，因为这些低电平会阻塞总线。
- 0000 0100 (04h)：硬件写入从地址的可编程部分。 行为与上述相同，但设备不复位。
- 0000 0000 (00h)：此代码不允许用作第二个字节。 编程过程的序列在适当的设备数据表中公布。其余代码尚未固定，设备必须忽略它们。 当位B为“1”时，2字节序列是“硬件通用调用”。这意味着该序列由硬件主设备传输，例如键盘扫描器， 它可以被编程来传输所需的从地址。由于硬件主设备事先并不知道消息必须被传输到哪个设备，它只能生成这个硬件通用调用和它自己的地址 — 向系统标识它自己(参见图 17)。

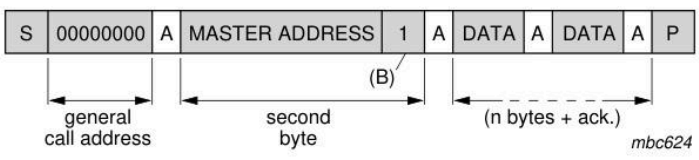


Fig 17. Data transfer from a hardware master-transmitter

第二个字节中剩下的七位包含硬件主机的地址。 这个地址被连接到总线的智能设备(例如，微控制器)识别，然后接受来自硬件主机的信息。如果硬件主机也可以充当从机，从机地址与主机地址相同。

在某些系统中，另一种方法是将硬件主发射机在系统复位后设置为从接收机模式。这样，系统配置主可以告诉硬件主发射机(现在处于从接收机模式)必须发送数据到哪个地址(见图18)。在编程程序之后，硬件主保持在主发射机模式。

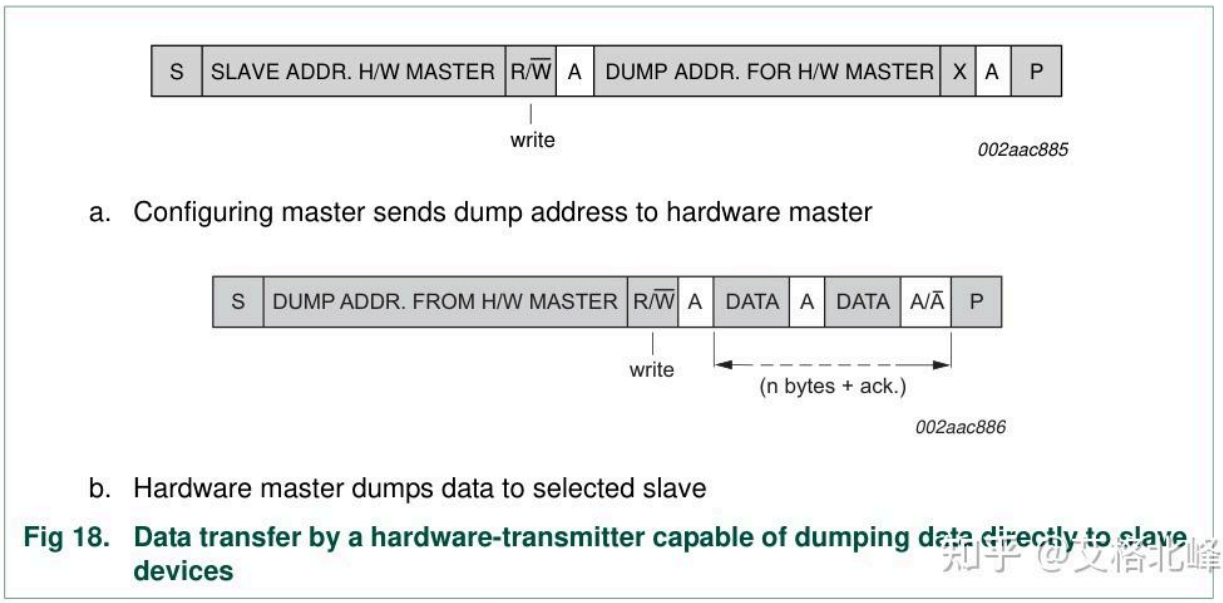


Fig 18. Data transfer by a hardware-transmitter capable of dumping data directly to slave devices

12. 开始字节

微控制器可以以两种方式连接到I2C总线。带有片上硬件I2C总线接口的微控制器可以被编程为只被总线请求中断。当设备没有这样的接口时，它必须通过软件不断地监视总线。显然，微控制器监视或轮询总线的次数越多，它执行预定功能的时间就越少。 因此，在快速的硬件设备和相对较慢的依赖于软件轮询的微控制器之间存在速度差异。 在这种情况下，数据传输可以先由一个比正常情况长得多的启动过程(见图19)。启动过程包括：

- 一个开始条件(S)
- 一个开始字节(0000 0001)
- 一个确认时钟脉冲(ACK)
- 一个重复的开始条件(Sr)

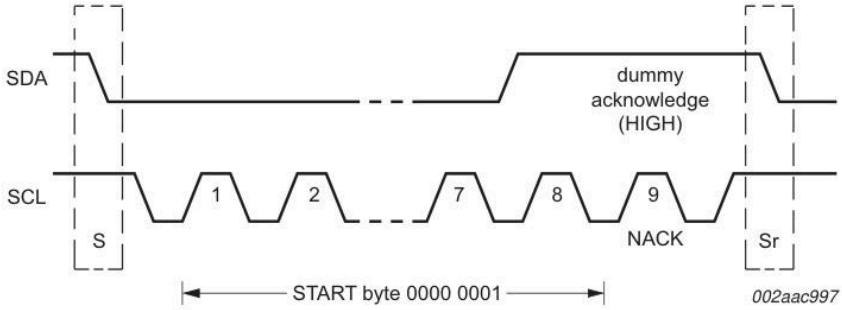


Fig 19. START byte procedure

在需要总线访问的主机传输了START条件S之后， 传输START字节(0000 0001)。另一个微控制器因此可以以低采样率对SDA线进行采样，直到检测到START字节中的7个零之一。在检测到SDA线上的LOW电平后，微控制器可以切换到更高的采样率，以找到重复的START条件Sr，然后用于同步。

硬件接收器在接收到重复的START条件Sr后重置，因此忽略START字节。

在START字节后生成一个与确认相关的时钟脉冲。这只是为了符合总线上使用的字节处理格式。不允许任何设备确认START字节。

13. 设备ID

设备ID字段(见图20)是一个可选的3字节只读(24位)字，提供以下信息：

- 12位制造商名称，每个制造商(例如NXP)都是唯一的
- 9位部件标识，由制造商指定(例如PCA9698)
- 3位模具修订，由制造商指定(例如RevX)

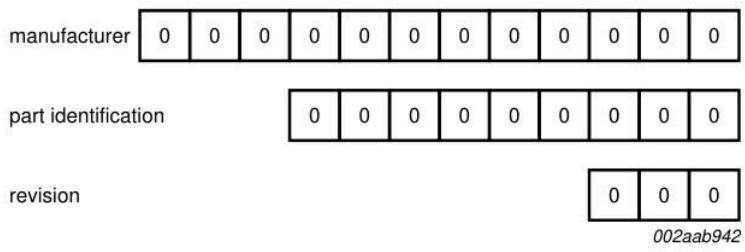


Fig 20. Device ID field

设备ID是只读的，硬连接在设备中，可以按如下方式访问：

1. START 条件

2. 主控器发送保留设备ID I2C总线地址，后面跟着设置为‘0’的R/W位(写入)：“1111 1000”。
3. 主设备发送它必须识别的从设备的I2C总线从地址。LSB是一个“不关心”的值。只有一个设备必须确认这个字节(具有I2C总线从地址的设备)。
4. 主设备发送一个Re-START条件。 备注：一个STOP条件跟随一个START条件重置从设备的状态机，设备ID读取无法执行。同样，一个STOP条件或一个Re-START条件跟随访问另一个从设备重置从设备的状态机，设备ID读取无法执行。
5. 主控器发送保留设备ID I2C总线地址，后面跟着设置为‘1’的R/W位：‘1111 1001’。
6. 设备ID读取可以完成，从12个制造商位(第一个字节+第二个字节的四个MSB)开始，接下来是9个部件识别位(第二个字节的四个LSB+第三个字节的五个MSB)，然后是三个模具修正位(第三个字节的三个LSB)。
7. 主设备通过ACK最后一个字节结束读取序列，从而重置从设备的状态机并允许主设备发送STOP条件。

备注：设备ID的读取可以通过发送一个ACK在任何时候停止。

如果主设备继续ACK第三个字节后的字节，从设备回滚到第一个字节并继续发送设备ID序列，直到检测到一个ACK。

Table 4. Assigned manufacturer IDs

Manufacturer bits												Company
11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	NXP Semiconductors
0	0	0	0	0	0	0	0	0	0	0	1	NXP Semiconductors (reserved)
0	0	0	0	0	0	0	0	0	0	1	0	NXP Semiconductors (reserved)
0	0	0	0	0	0	0	0	0	0	1	1	NXP Semiconductors (reserved)
0	0	0	0	0	0	0	0	0	1	0	0	Ramtron International
0	0	0	0	0	0	0	0	0	1	0	1	Analog Devices
0	0	0	0	0	0	0	0	0	1	1	0	STMicroelectronics
0	0	0	0	0	0	0	0	0	1	1	1	ON Semiconductor
0	0	0	0	0	0	0	0	1	0	0	0	Sprintek Corporation
0	0	0	0	0	0	0	0	1	0	0	1	ESPROS Photonics AG
0	0	0	0	0	0	0	0	1	0	1	0	Fujitsu Semiconductor
0	0	0	0	0	0	0	0	1	0	1	1	Flir
0	0	0	0	0	0	0	0	1	1	0	0	O_2Micro
0	0	0	0	0	0	0	0	1	1	0	1	Atmel

Designers of new I^2C devices who want to implement the device ID feature should contact NXP at **i2c.support@nxp.com** to have a unique manufacturer ID assigned.

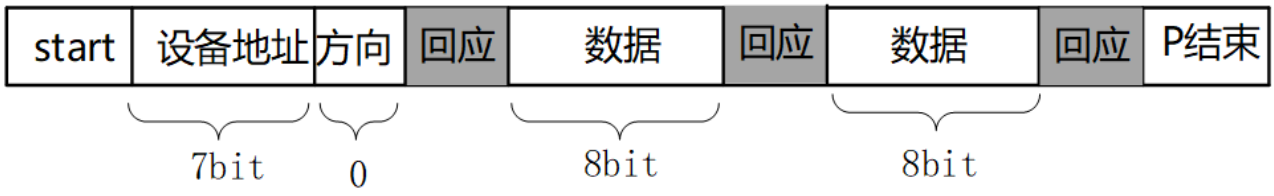
4. I2C传输数据的格式

4.1 写操作

流程如下：

- 主芯片要发出一个start信号
- 然后发出一个设备地址(用来确定是往哪一个芯片写数据)，方向(读/写，0表示写，1表示读)
- 从设备回应(用来确定这个设备是否存在)，然后就可以传输数据
- 主设备发送一个字节数据给从设备，并等待回应
- 每传输一字节数据，接收方要有一个回应信号（确定数据是否接受完成)，然后再传输下一个数据。
- 数据发送完之后，主芯片就会发送一个停止信号。

下图：白色背景表示"主→从"，灰色背景表示"从→主"

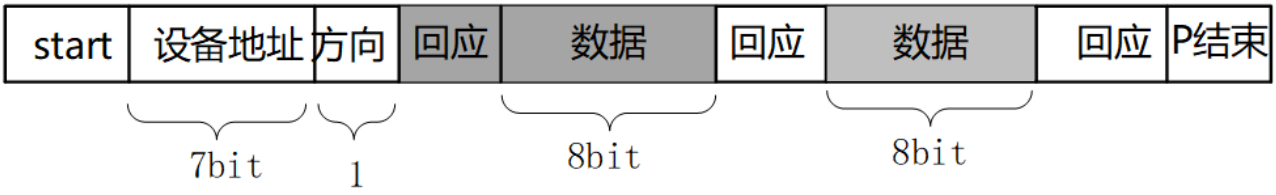


4.2 读操作

流程如下：

- 主芯片要发出一个start信号
- 然后发出一个设备地址(用来确定是往哪一个芯片写数据)，方向(读/写，0表示写，1表示读)
- 从设备回应(用来确定这个设备是否存在)，然后就可以传输数据
- 从设备发送一个字节数据给主设备，并等待回应
- 每传输一字节数据，接收方要有一个回应信号（确定数据是否接受完成)，然后再传输下一个数据。
- 数据发送完之后，主芯片就会发送一个停止信号。

下图：白色背景表示"主→从"，灰色背景表示"从→主"

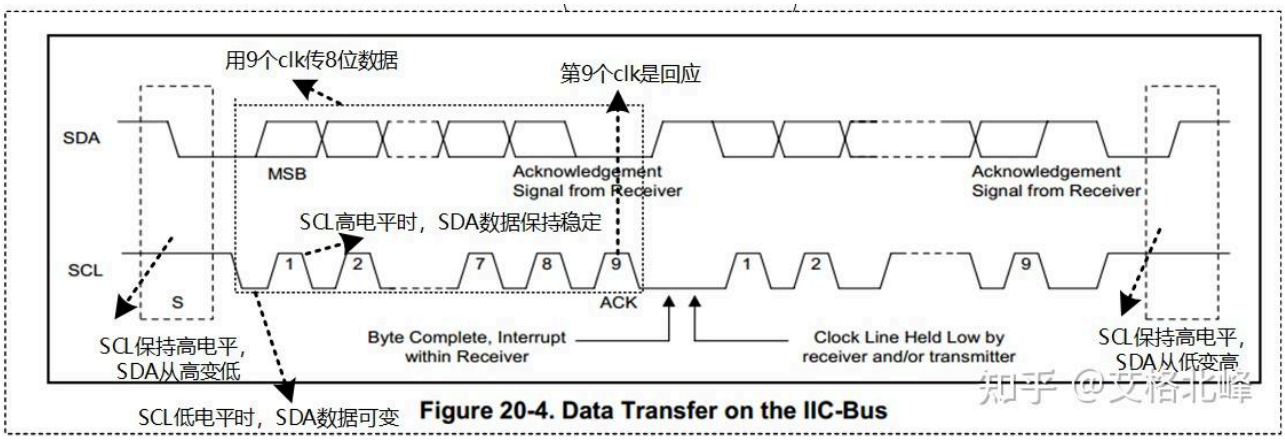


4.3 I2C信号

I2C协议中数据传输的单位是字节，也就是8位。但是要用到9个时钟：前面8个时钟用来传输8数据，第9个时钟用来传输回应信号。传输时，先传输最高位(MSB)。

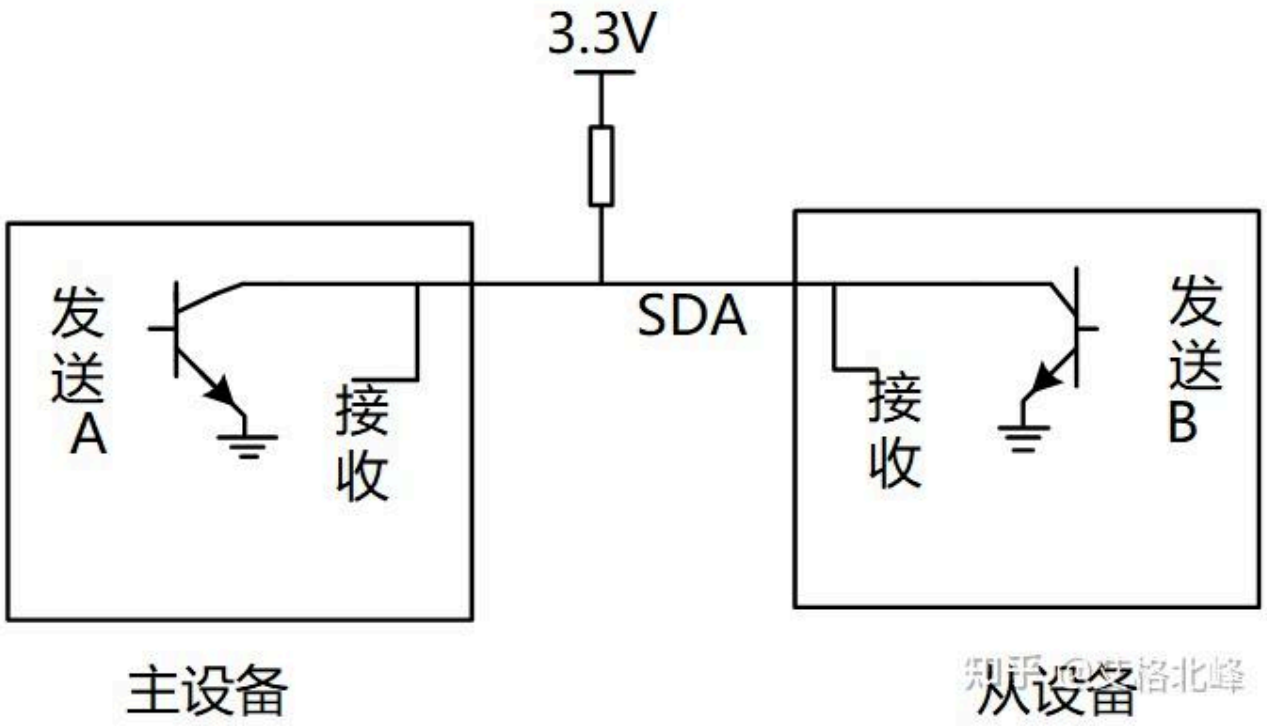
- 开始信号（S）：SCL为高电平时，SDA山高电平向低电平跳变，开始传送数据。
- 结束信号（P）：SCL为高电平时，SDA由低电平向高电平跳变，结束传送数据。
- 响应信号(ACK)：接收器在接收到8位数据后，在第9个时钟周期，拉低SDA
- SDA上传输的数据必须在SCL为高电平期间保持稳定，SDA上的数据只能在SCL为低电平期间变化

I2C协议信号如下：



4.4 协议细节

- 如何在SDA上实现双向传输？ 主芯片通过一根SDA线既可以把数据发给从设备，也可以从SDA上读取数据，连接SDA线的引脚里面必然有两个引脚（发送引脚/接受引脚）。
- 主、从设备都可以通过SDA发送数据，肯定不能同时发送数据，怎么错开时间？ 在9个时钟里， 前8个时钟由主设备发送数据的话，第9个时钟就由从设备发送数据； 前8个时钟由从设备发送数据的话，第9个时钟就由主设备发送数据。
- 双方设备中，某个设备发送数据时，另一方怎样才能不影响SDA上的数据？ 设备的SDA中有一个三极管，使用开极/开漏电路(三极管是开极，CMOS管是开漏，作用一样)，如下图：



真值表如下：

从真值表和电路图我们可以知道：

- 当某一个芯片不想影响SDA线时，那就不驱动这个三极管
- 想让SDA输出高电平，双方都不驱动三极管(SDA通过上拉电阻变为高电平)
- 想让SDA输出低电平，就驱动三极管

4.5 示例：主设备发送（8bit）给从设备

从下面的例子可以看看数据是怎么传的（实现双向传输）。 举例：主设备发送（8bit）给从设备

- 前8个clk

    - 从设备不要影响SDA，从设备不驱动三极管
    - 主设备决定数据，主设备要发送1时不驱动三极管，要发送0时驱动三极管

- 第9个clk，由从设备决定数据
    - 主设备不驱动三极管
    - 从设备决定数据，要发出回应信号的话，就驱动三极管让SDA变为0
    - 从这里也可以知道ACK信号是低电平

从上面的例子，就可以知道怎样在一条线上实现双向传输，这就是SDA上要使用上拉电阻的原因。

4.6 为何SCL也要使用上拉电阻?

在第9个时钟之后，如果有某一方需要更多的时间来处理数据，它可以一直驱动三极管把SCL拉低。 当SCL为低电平时候，大家都不应该使用IIC总线，只有当SCL从低电平变为高电平的时候，IIC总线才能被使用。 当它就绪后，就可以不再驱动三极管，这是上拉电阻把SCL变为高电平，其他设备就可以继续使用I2C总线了。

对于IIC协议它只能规定怎么传输数据，数据是什么含义由从设备决定。

5. I2C通信的高级应用

在嵌入式系统设计中，I2C应用广泛，如：

5.1 传感器网络

在多传感器系统中，I2C用于读取各种环境参数，如温度、湿度、光照强度等。这些数据可以被用于自动化控制系统或数据监测。

5.2 多设备控制

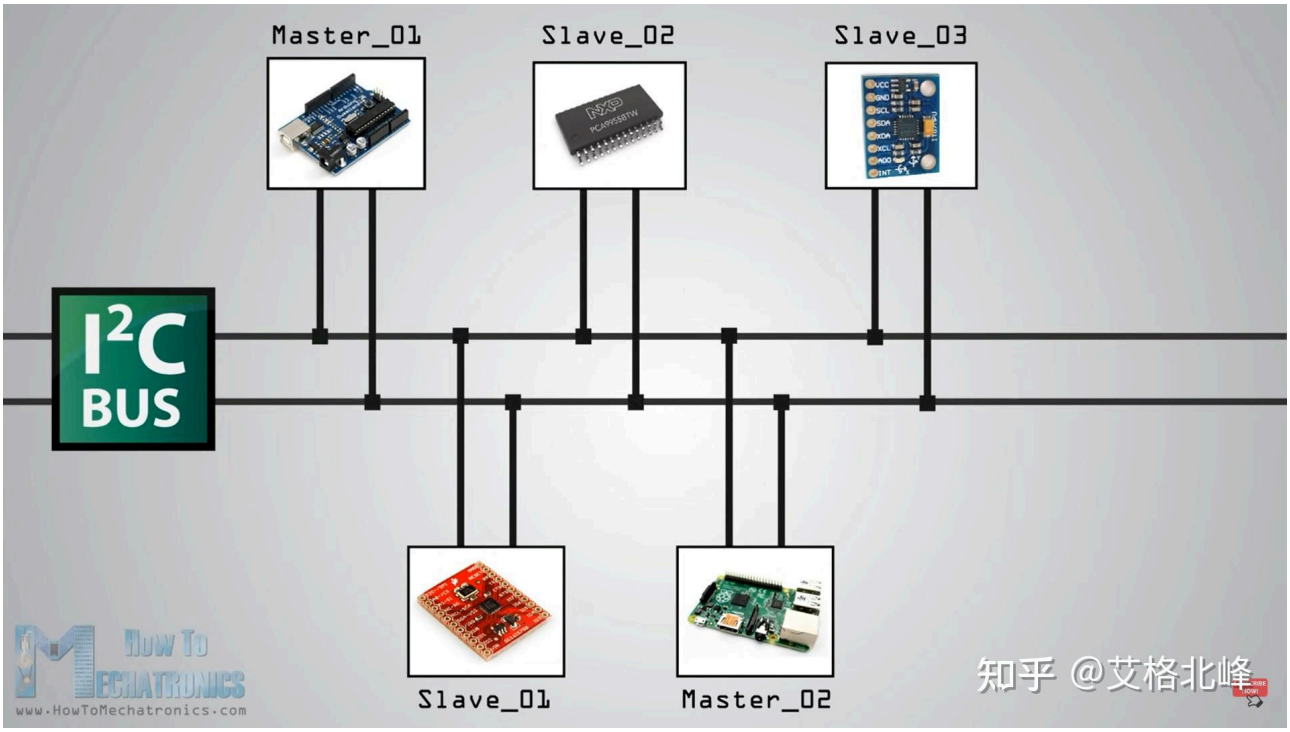
在复杂的嵌入式系统中，如机器人或无人机，I2C用于控制和监测多个执行器和传感器，实现精确的运动控制和环境反馈。

5.3 嵌入式通信网络

I2C也常用于建立微控制器和各种外围设备（如显示屏、存储设备等）之间的通信网络。

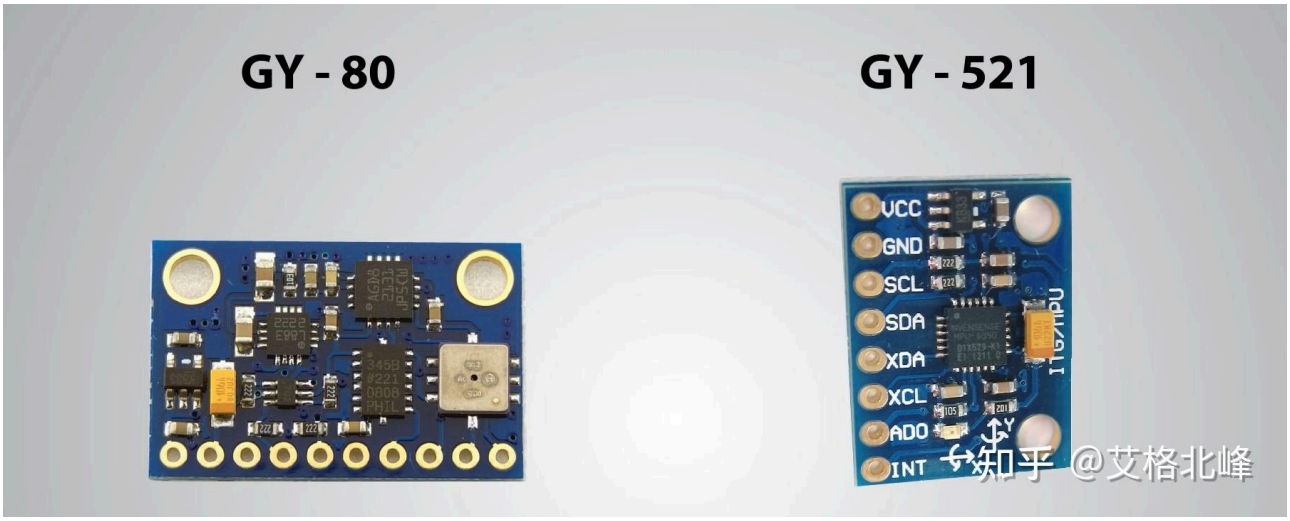
6. I2C应用示例--以Arduino为例

要用GPIO模拟I2C（通常被称为"bit-banging"），你需要手动控制GPIO引脚来模拟I2C协议的SDA（数据线）和SCL（时钟线）。下面是使用Arduino系列单片机进行I2C的一个基本示例。



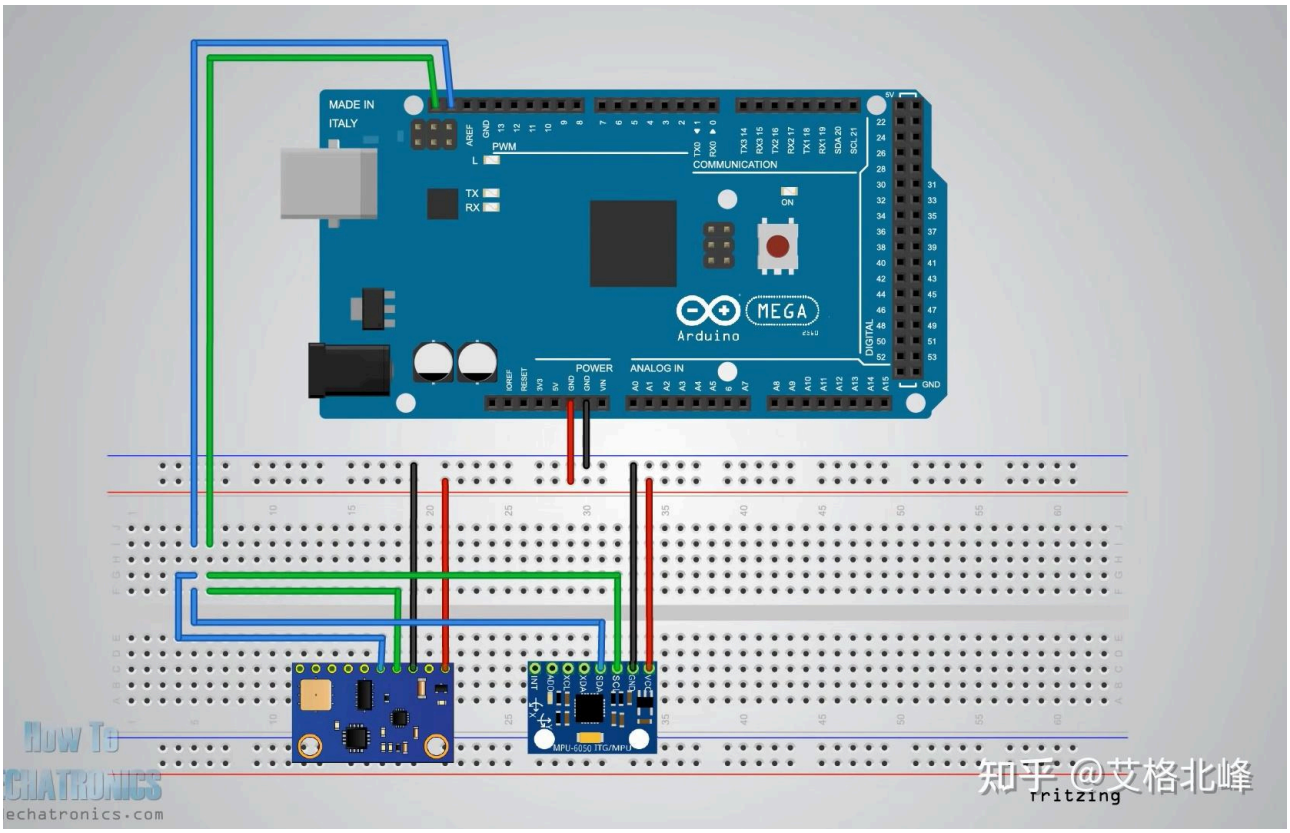
6.1 应用I2C传输数据的传感器

现在让我们用Arduino板和一些传感器来做一个例子并演示它。作为一个例子，我将使用GY-80电路板，它包含五个不同的传感器，以及GY-521电路板，它包含三个不同的传感器。



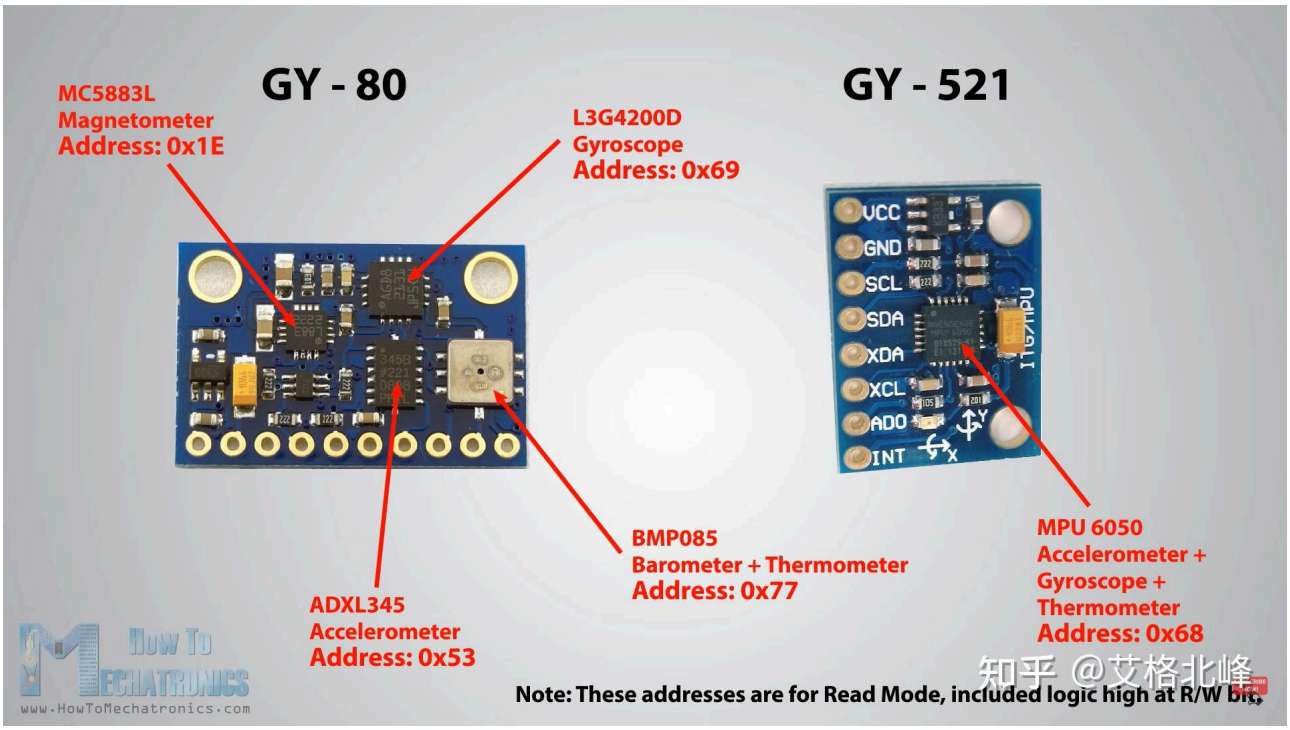
6.2 电路板连接

因此，我们可以只用两根线通过I2C总线从八个不同的传感器获取数据。这里是我们将如何连接这些板。Arduino板的串行时钟引脚将连接到两个断路板的串行时钟引脚，对于串行数据引脚也是如此，我们还将使用Arduino板的地线和5V引脚为板提供电源。

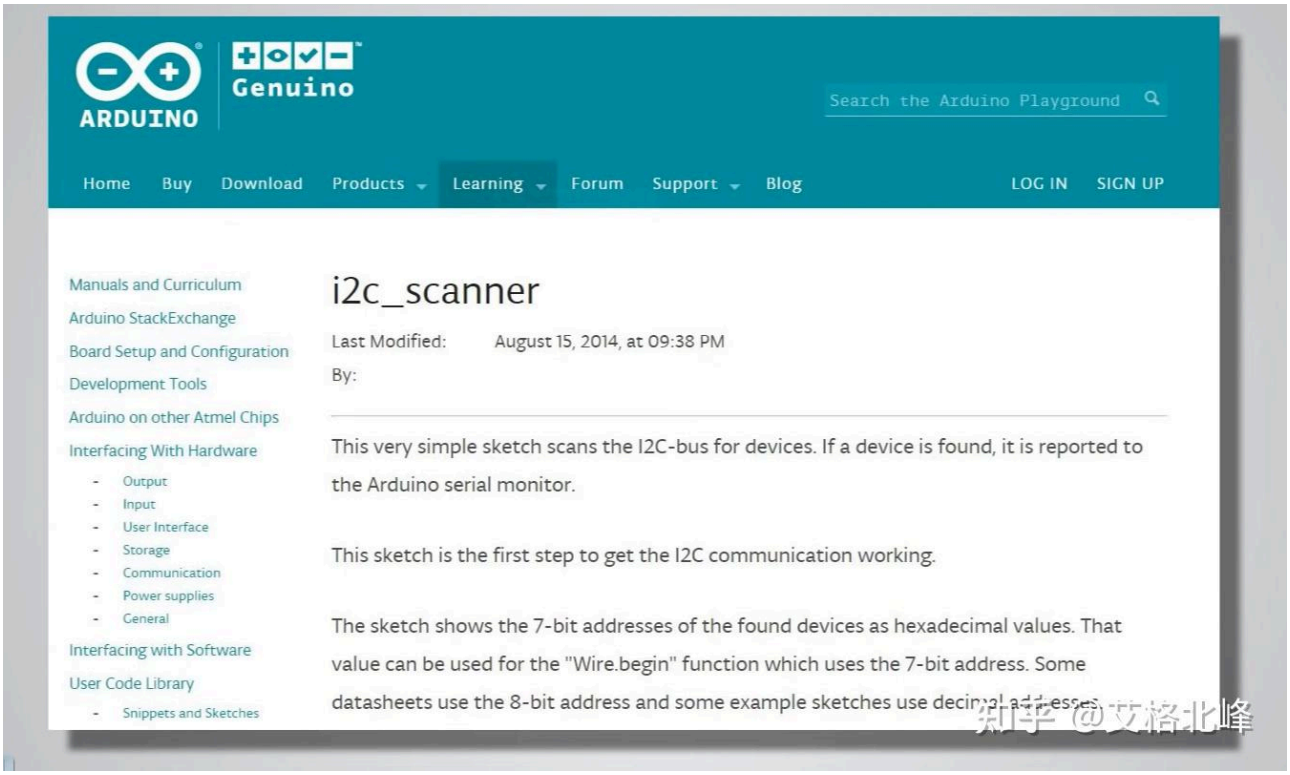


6.3 Address地址查找

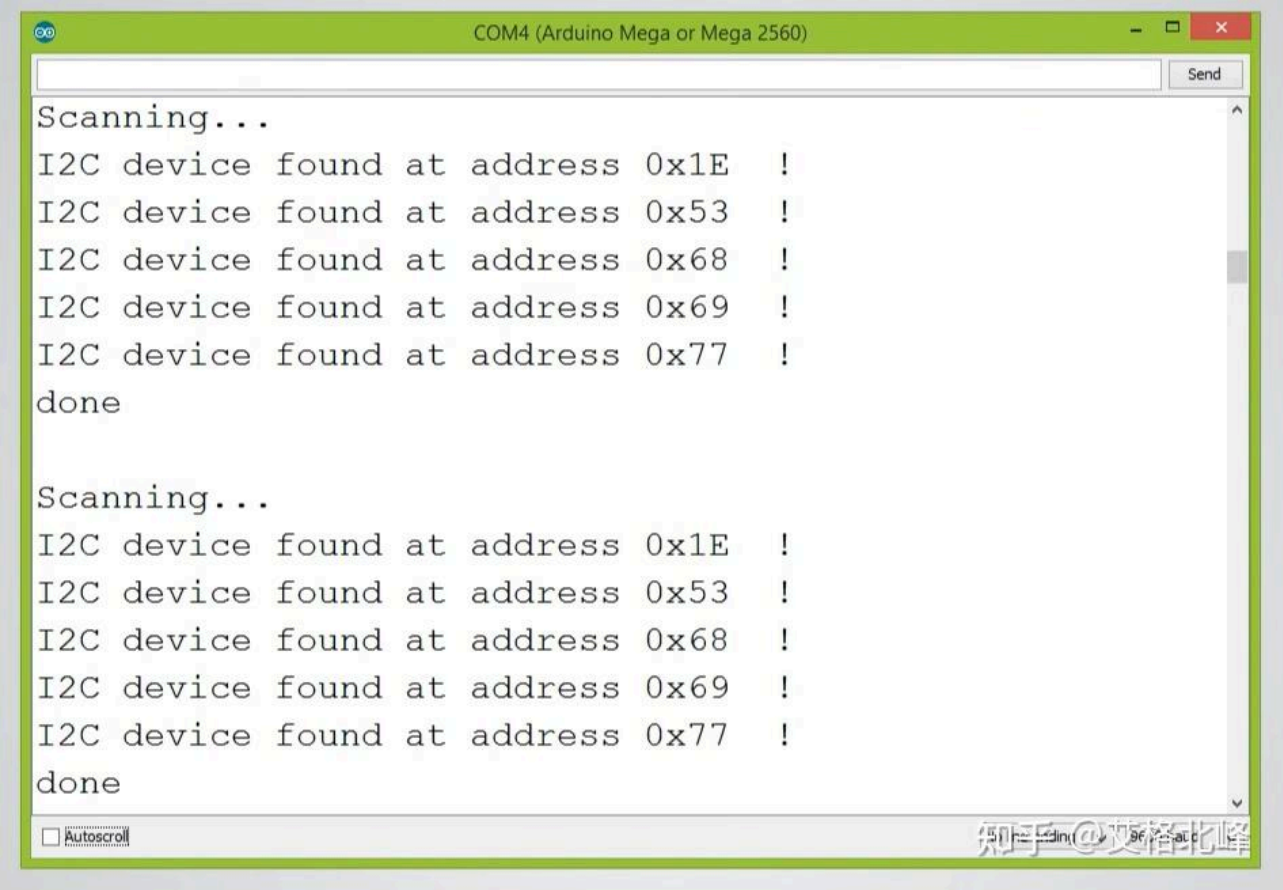
注意，我们不使用上拉电阻，因为电路板已经有了。现在为了与这些芯片或传感器通信，我们需要知道它们的独特地址。我们可以从传感器的数据手册中找到它们。对于GY-80断路板，我们有以下四个地址：0x53是十六进制的三轴加速度计，0x69是十六进制的三轴陀螺仪，0x1E是十六进制的三轴磁力计，0x77是十六进制的气压计和温度传感器。对于GY-521断路板，我们只有一个地址，那就是十六进制的0x68。



我们也可以使用I2C扫描器草图来获取或检查地址，这可以从Arduino官方网站找到。所以如果我们上传并运行那个草图，我们将得到I2C总线上连接设备的地址。



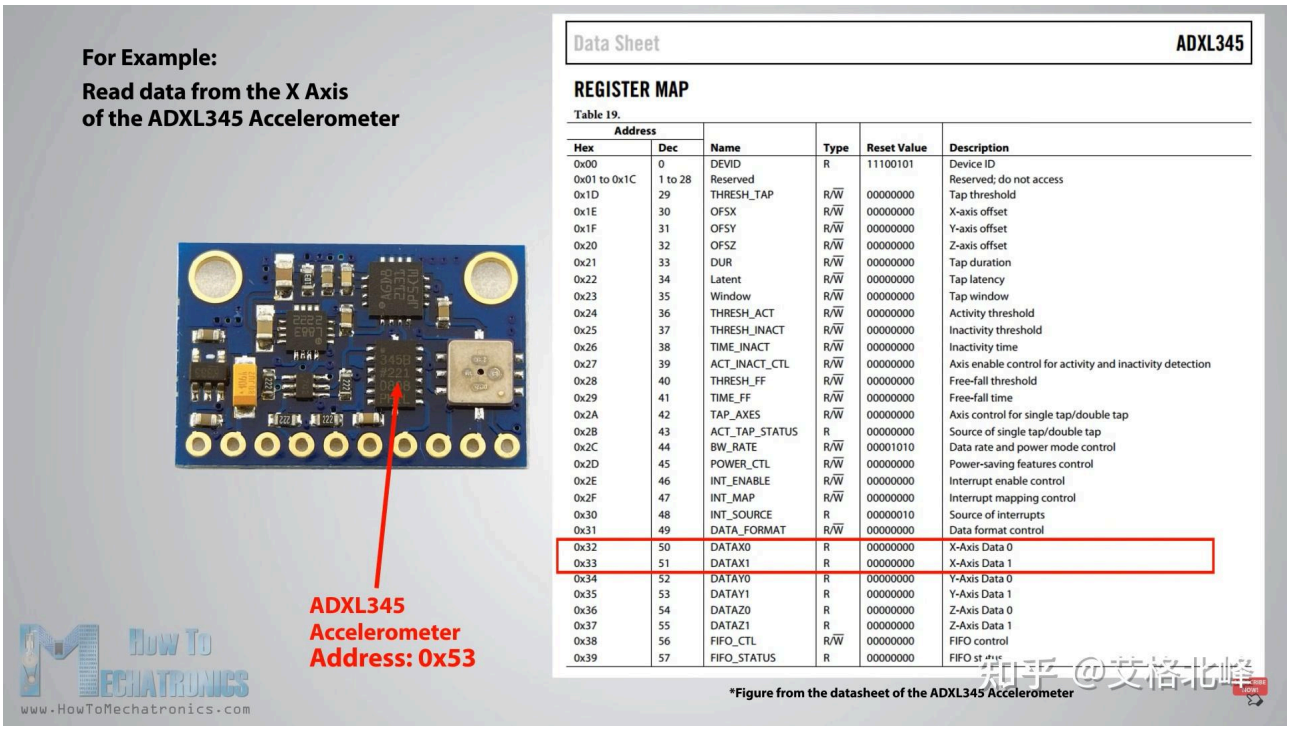
所以如果我们上传并运行那个草图，我们将得到I2C总线上连接设备的地址。所以在我们找到设备的地址之后，我们还需要找到它们的内部寄存器地址，以便从中读取数据。



6.4 I2C读取数据示例

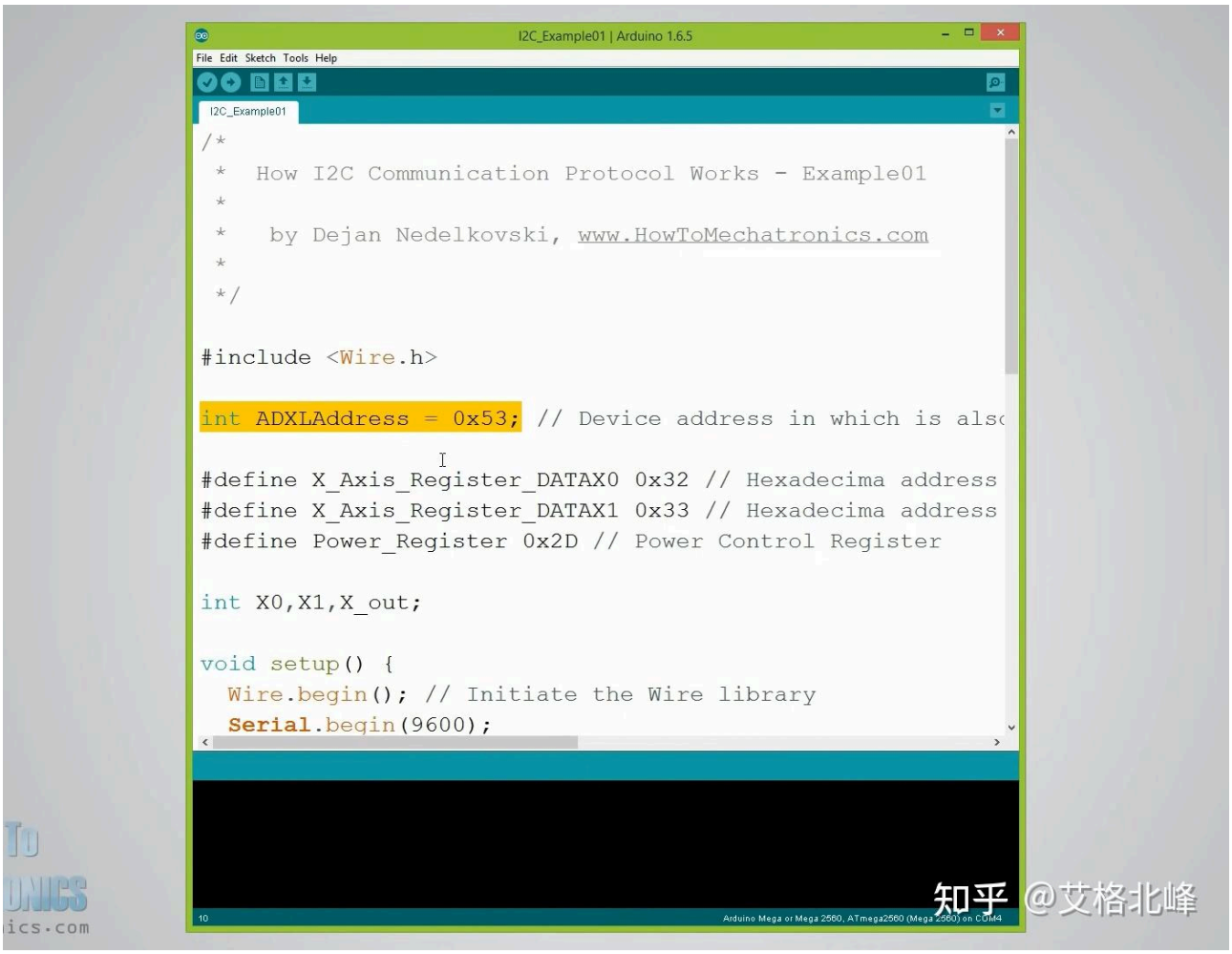
例如，如果我们想从GY-80断路板的三轴加速度计传感器中读取X轴的数据，我们需要找到存储X轴数据的内部寄存器地址。

从传感器的数据手册中我们可以看到，X轴的数据实际上存储在两个寄存器中，DATAX0的十六进制地址是0x32，DATAX1的十六进制地址是0x33。现在让我们编写代码来获取X轴的数据。

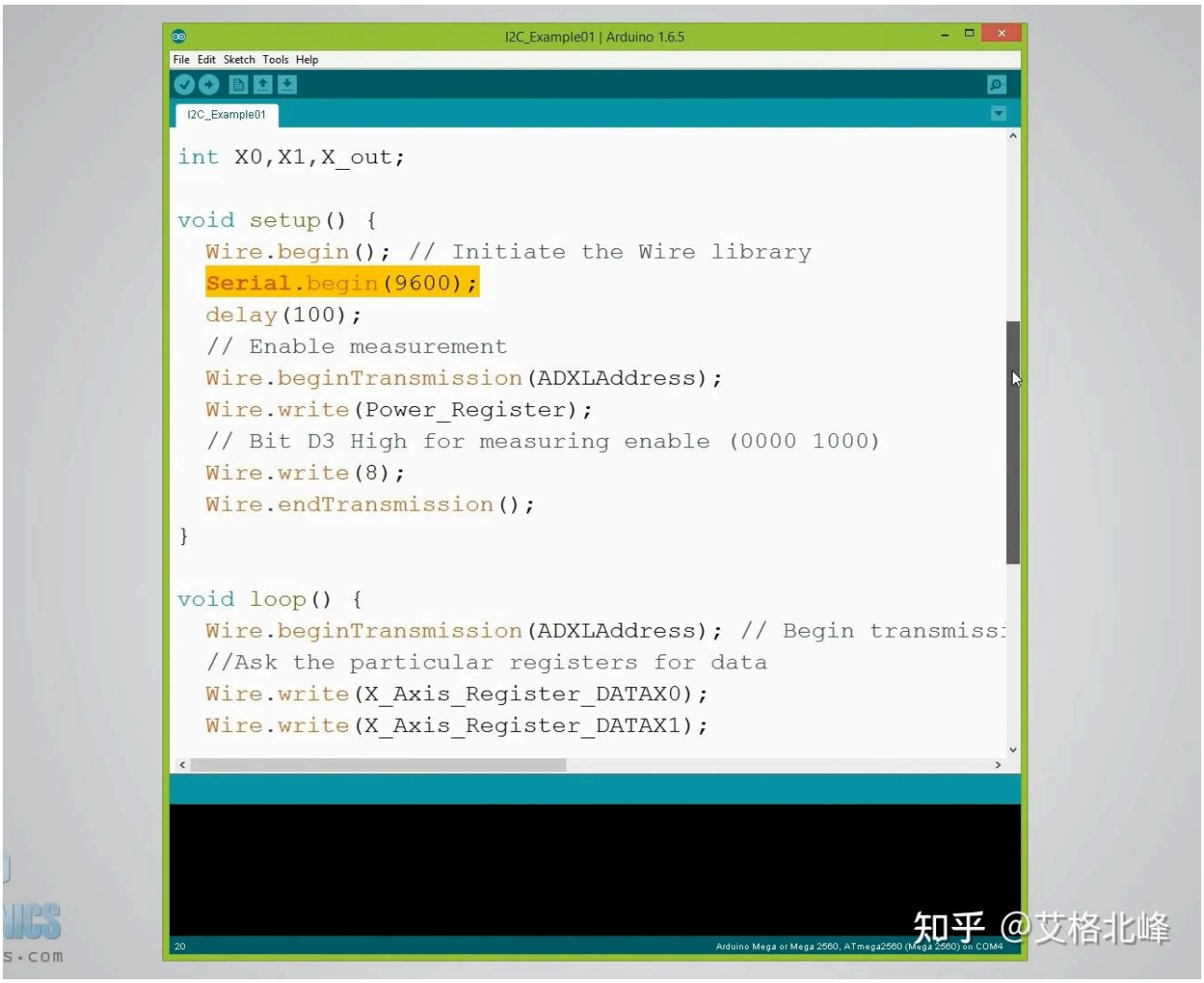


6.5 程序编写

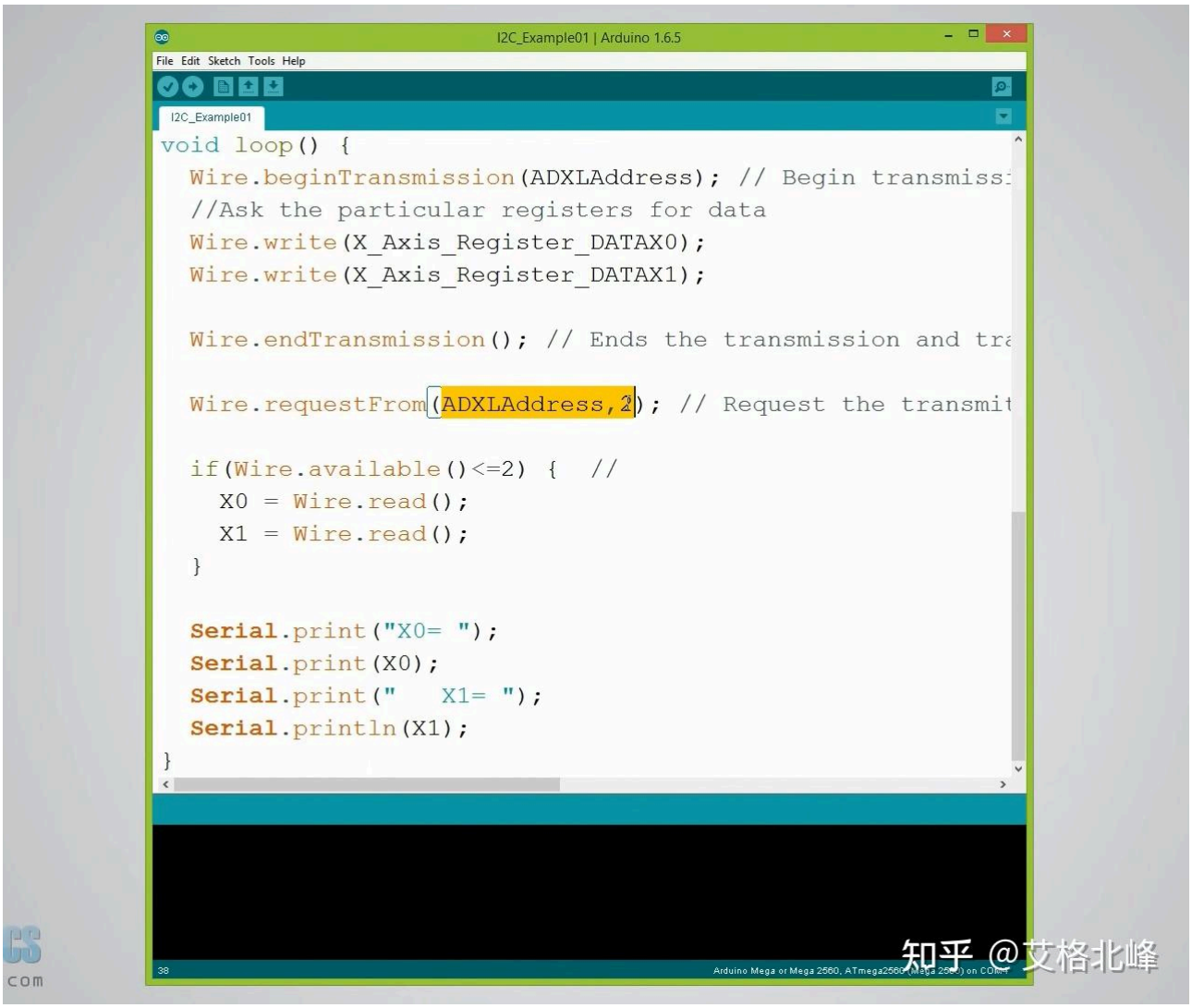
所以我们将使用Arduino Wire库，这必须包含在草图中。这里首先我们必须定义我们之前找到的传感器地址和两个内部寄存器地址。



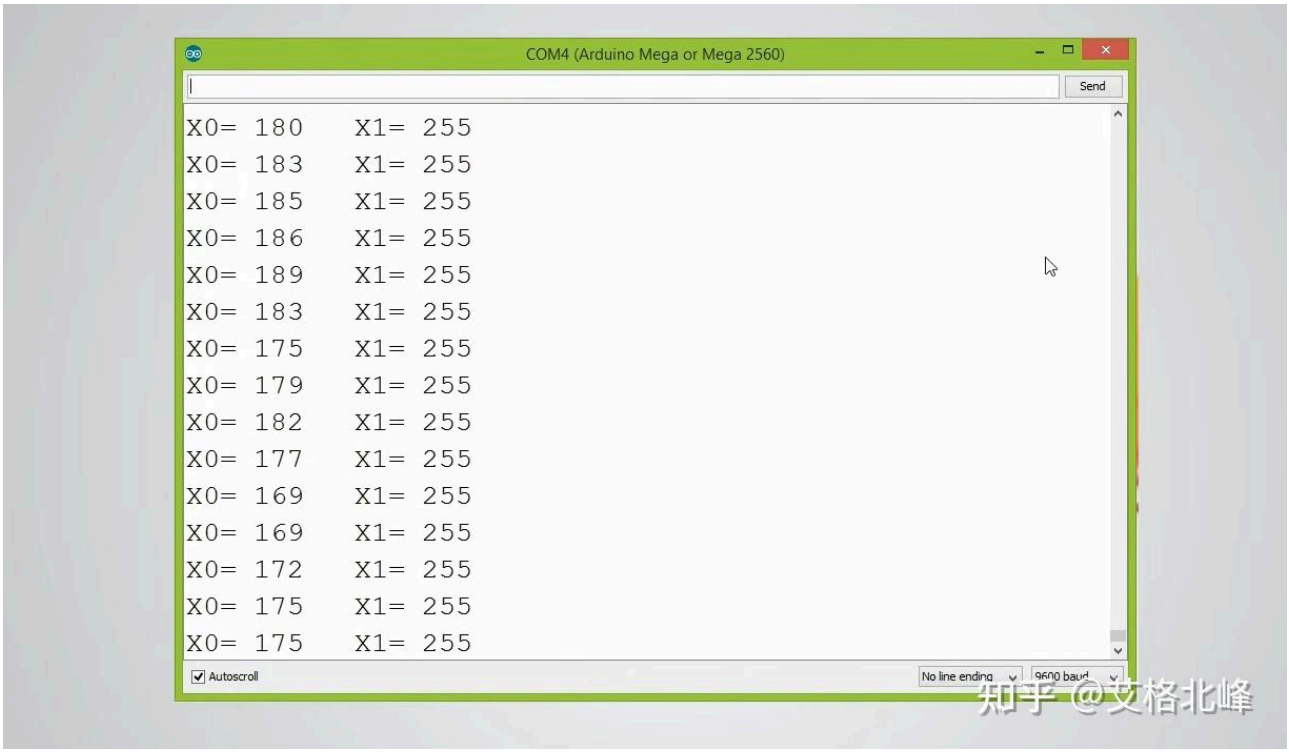
Wire.begin()函数将启动Wire库，我们还需要启动串行通信，因为我们将使用串行监视器显示传感器的数据。

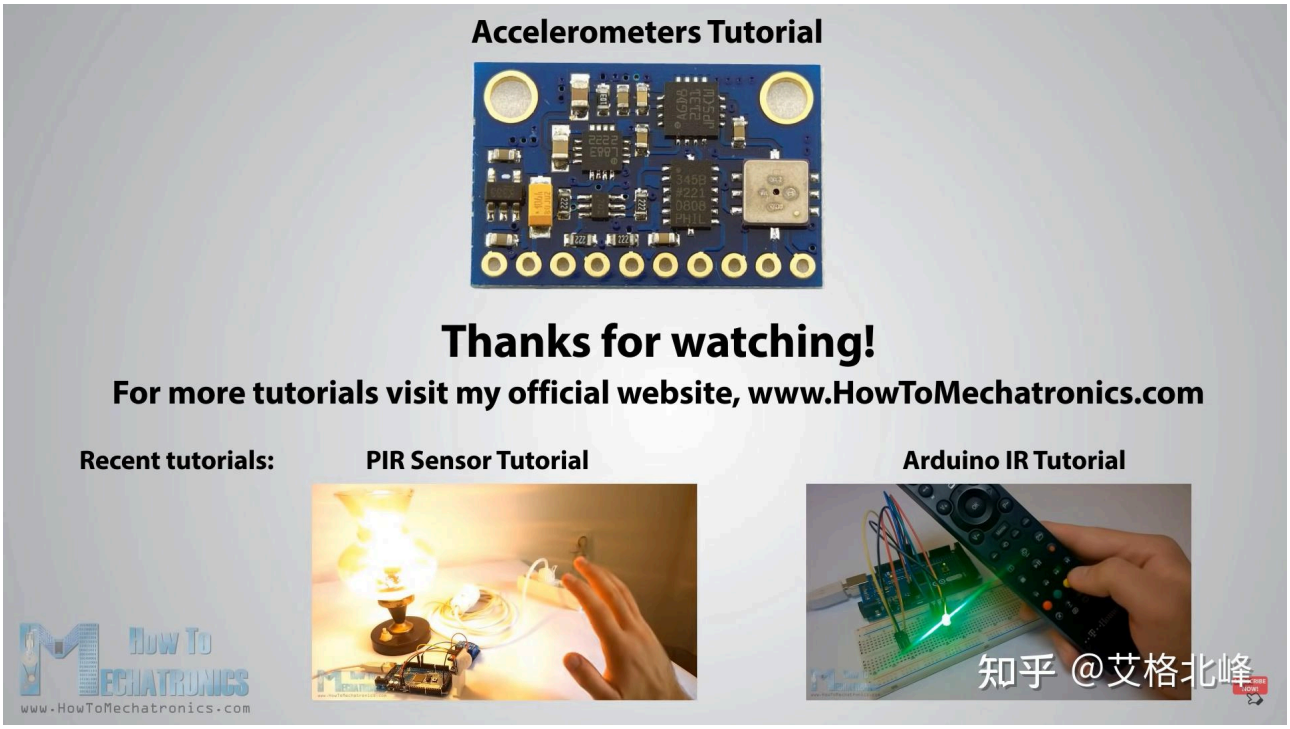


在循环中，我们将从Wire开始。beginTransmission()函数将开始向特定传感器传输，我们的案例中是三轴加速度计。然后使用Wire.write()函数，我们将请求来自X轴两个寄存器的特定数据。Wire.endTransmission()函数将结束传输并传输来自寄存器的数据。现在通过Wire.requestFrom()函数，我们将要求传输的数据或寄存器的两个字节。Wire.available()函数将返回可检索的字节数，如果该数字与我们请求的字节数相匹配，在我们的案例中是两个字节，使用Wire.read()函数，我们将从X轴的寄存器中读取字节。最后我们将在串行监视器中打印数据。



这是那些数据，但请记住，这些是原始数据，需要做一些数学运算才能得到X轴的正确值。你可以在我的下一个教程中找到更多细节，那个教程是关于如何使用Arduino板上的加速度计，因为我不想让这个教程过于复杂，它的主要目标是解释I2C通信是如何工作的。





结论

I2C通信总线以其简单、高效和灵活的特性，在嵌入式系统设计中占据了举足轻重的地位。对于经验丰富的技术专家而言，深入理解I2C的工作原理和应用场景，不仅可以提高系统设计的效率和可靠性，还可以为解决复杂的设计挑战提供更多的可能性。随着嵌入式技术的不

断进步，I2C的应用领域和技术深度也将持续扩展，为嵌入式系统设计带来更多创新和突破。