# UART、SPI和I2C详解

做单片机开发时UART,SPI和I2C都是我们最经常使用到的硬件接口，我收集了相关的具体材料对这三种接口进行了详细的解释。

## UART

UART是一种通用串行数据总线，用于异步通信。该总线双向通信，可以实现全双工传输和接收。在嵌入式设计中，UART用于主机与辅助设备通信，如汽车音响与外接AP之间的通信，与PC机通信包括与监控调试器和其它器件，如EEPROM通信。

将由计算机内部传送过来的并行数据转换为输出的串行数据流。将计算机外部来的串行数据转换为字节，供计算机内部并行数据的器件使用。在输出的串行数据流中加入奇偶校验位，并对从外部接收的数据流进行奇偶校验。在输出数据流中加入启停标记，并从接收数据流中删除启停标记。处理由键盘或鼠标发出的中断信号（键盘和鼠标也是串行设备）。可以处理计算机与外部串行设备的同步管理问题。有一些比较高档的UART还提供输入输出数据的缓冲区，比较新的UART是16550，它可以在计算机需要处理数据前在其缓冲区内存储16字节数据，而通常的UART是8250。

### 接口

在单片机中UART可以说是一种最基本的配置，很多与电脑进行通信的设备都采用到它，按计算机最常规的说法就是串行通信。

* TX - 数据发送接口
* RX - 数据接受接口

两个设备间将TX与RX相连，RX与TX相连即可正常工作。最常用到的就是我们电脑上的USB那就是个最典型的UART接口。

### 工作原理

#### 发送接收

发送逻辑对从发送FIFO 读取的数据执行“并→串”转换。控制逻辑输出起始位在先的串行位流，并且根据控制寄存器中已编程的配置，后面紧跟着数据位（注意：最低位 LSB 先输出）、奇偶校验位和停止位。  
在检测到一个有效的起始脉冲后，接收逻辑对接收到的位流执行“串→并”转换。此外还会对溢出错误、奇偶校验错误、帧错误和线中止（line-break）错误进行检测，并将检测到的状态附加到被写入接收FIFO 的数据中。

#### 波特率产生

波特率除数（baud-rate divisor）是一个22 位数，它由16 位整数和6 位小数组成。波特率发生器使用这两个值组成的数字来决定位周期。通过带有小数波特率的除法器，在足够高的系统时钟速率下，UART 可以产生所有标准的波特率，而误差很小。

#### 数据收发

发送时，数据被写入发送FIFO。如果UART 被使能，则会按照预先设置好的参数（波特率、数据位、停止位、校验位等）开始发送数据，一直到发送FIFO 中没有数据。一旦向发送FIFO 写数据（如果FIFO 未空），UART 的忙标志位BUSY 就有效，并且在发送数据期间一直保持有效。BUSY 位仅在发送FIFO 为空，且已从移位寄存器发送最后一个字符，包括停止位时才变无效。即 UART 不再使能，它也可以指示忙状态。BUSY 位的相关库函数是UARTBusy( )  
在UART 接收器空闲时，如果数据输入变成“低电平”，即接收到了起始位，则接收计数器开始运行，并且数据在Baud16 的第8 个周期被采样。如果Rx 在Baud16 的第8 周期仍然为低电平，则起始位有效，否则会被认为是错误的起始位并将其忽略。  
如果起始位有效，则根据数据字符被编程的长度，在 Baud16 的每第 16 个周期（即一个位周期之后）对连续的数据位进行采样。如果奇偶校验模式使能，则还会检测奇偶校验位。  
最后，如果Rx 为高电平，则有效的停止位被确认，否则发生帧错误。当接收到一个完整的字符时，将数据存放在接收FIFO 中。

#### 中断控制

出现以下情况时，可使UART 产生中断：

* FIFO 溢出错误
* 线中止错误（line-break，即Rx 信号一直为0 的状态，包括校验位和停止位在内）
* 奇偶校验错误
* 帧错误（停止位不为1）
* 接收超时（接收FIFO 已有数据但未满，而后续数据长时间不来）
* 发送
* 接收

由于所有中断事件在发送到中断控制器之前会一起进行“或运算”操作，所以任意时刻 UART 只能向中断产生一个中断请求。通过查询中断状态函数UARTIntStatus( )，软件可以在同一个中断服务函数里处理多个中断事件（多个并列的if 语句）。

#### FIFO 操作

FIFO 是“First-In First-Out”的缩写，意为“先进先出”，是一种常见的队列操作。 Stellaris 系列ARM 的UART 模块包含有2 个16 字节的FIFO：一个用于发送，另一个用于接收。可以将两个FIFO 分别配置为以不同深度触发中断。可供选择的配置包括：1/8、 1/4、1/2、3/4 和7/8 深度。例如，如果接收FIFO 选择1/4，则在UART 接收到4 个数据时产生接收中断。

**发送FIFO的基本工作过程**： 只要有数据填充到发送FIFO 里，就会立即启动发送过程。由于发送本身是个相对缓慢的过程，因此在发送的同时其它需要发送的数据还可以继续填充到发送 FIFO 里。当发送 FIFO 被填满时就不能再继续填充了，否则会造成数据丢失，此时只能等待。这个等待并不会很久，以9600 的波特率为例，等待出现一个空位的时间在1ms 上下。发送 FIFO 会按照填入数据的先后顺序把数据一个个发送出去，直到发送 FIFO 全空时为止。已发送完毕的数据会被自动清除，在发送FIFO 里同时会多出一个空位。

**接收FIFO的基本工作过程**： 当硬件逻辑接收到数据时，就会往接收FIFO 里填充接收到的数据。程序应当及时取走这些数据，数据被取走也是在接收FIFO 里被自动删除的过程，因此在接收 FIFO 里同时会多出一个空位。如果在接收 FIFO 里的数据未被及时取走而造成接收FIFO 已满，则以后再接收到数据时因无空位可以填充而造成数据丢失。

收发FIFO 主要是为了解决UART 收发中断过于频繁而导致CPU 效率不高的问题而引入的。在进行 UART 通信时，中断方式比轮询方式要简便且效率高。但是，如果没有收发 FIFO，则每收发一个数据都要中断处理一次，效率仍然不够高。如果有了收发FIFO，则可以在连续收发若干个数据（可多至14 个）后才产生一次中断然后一并处理，这就大大提高了收发效率。

完全不必要担心FIFO 机制可能带来的数据丢失或得不到及时处理的问题，因为它已经帮你想到了收发过程中存在的任何问题，只要在初始化配置UART 后，就可以放心收发了， FIFO 和中断例程会自动搞定一切。

#### 回环操作

UART 可以进入一个内部回环（Loopback）模式，用于诊断或调试。在回环模式下，从Tx 上发送的数据将被Rx 输入端接收。

#### 串行红外协议

在某些 Stellaris 系列 ARM 芯片里，UART 还包含一个 IrDA 串行红外（SIR）编码器/ 解码器模块。IrDA SIR 模块的作用是在异步UART数据流和半双工串行SIR 接口之间进行转换。片上不会执行任何模拟处理操作。SIR 模块的任务就是要给UART 提供一个数字编码输出和一个解码输入。UART 信号管脚可以和一个红外收发器连接以实现IrDA SIR物理层连接。

## I2C

I2C总线是由Philips公司开发的一种简单、双向二线制同步串行总线。它只需要两根线即可在连接于总线上的器件之间传送信息。  
主器件用于启动总线传送数据，并产生时钟以开放传送的器件，此时任何被寻址的器件均被认为是从器件．在总线上主和从、发和收的关系不是恒定的，而取决于此时数据传送方向。如果主机要发送数据给从器件，则主机首先寻址从器件，然后主动发送数据至从器件，最后由主机终止数据传送；如果主机要接收从器件的数据，首先由主器件寻址从器件．然后主机接收从器件发送的数据，最后由主机终止接收过程。在这种情况下．主机负责产生定时时钟和终止数据传送。

### 接口

1. SCL - 串行时钟线
2. SDA - 串行数据线接口

SDA（串行数据线）和SCL（串行时钟线）都是双向I/O线，接口电路为开漏输出．需通过上拉电阻接电源VCC．当总线空闲时．两根线都是高电平，连接总线的外同器件都是CMOS器件，输出级也是开漏电路．在总线上消耗的电流很小，因此，总线上扩展的器件数量主要由电容负载来决定，因为每个器件的总线接口都有一定的等效电容．而线路中电容会影响总线传输速度．当电容过大时，有可能造成传输错误．所以，其负载能力为**400pF**，因此可以估算出总线允许长度和所接器件数量。

主器件用于启动总线传送数据，并产生时钟以开放传送的器件，此时任何被寻址的器件均被认为是从器件．在总线上主和从、发和收的关系不是恒定的，而取决于此时数据传送方向。如果主机要发送数据给从器件，则主机首先寻址从器件，然后主动发送数据至从器件，最后由主机终止数据传送；如果主机要接收从器件的数据，首先由主器件寻址从器件．然后主机接收从器件发送的数据，最后由主机终止接收过程。在这种情况下．主机负责产生定时时钟和终止数据传送。

### 特点

I2C总线特点可以概括如下：

* 在硬件上，I2C总线只需要一根数据线和一根时钟线两根线，总线接口已经集成在芯片内部，不需要特殊的接口电路，而且片上接口电路的滤波器可以滤去总线数据上的毛刺．因此I2C总线简化了硬件电路PCB布线，降低了系统成本，提高了系统可靠性。因为I2C芯片除了这两根线和少量中断线，与系统再没有连接的线，用户常用IC可以很容易形成标准化和模块化，便于重复利用。
* I2C总线是一个真正的多主机总线，如果两个或多个主机同时初始化数据传输，可以通过冲突检测和仲裁防止数据破坏，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。数据传输和地址设定由软件设定，非常灵活。总线上的器件增加和删除不影响其他器件正常工作。
* I2C总线可以通过外部连线进行在线检测，便于系统故障诊断和调试，故障可以立即被寻址，软件也利于标准化和模块化，缩短开发时问。
* 连接到相同总线上的IC数量只受总线最大电容的限制，串行的8位双向数据传输位速率在标准模式下可达100Kbit/s，快速模式下可达400Kbit/s，高速模式下可达3．4Mbit/s。
* 总线具有极低的电流消耗．抗高噪声干扰，增加总线驱动器可以使总线电容扩大10倍，传输距离达到15m；兼容不同电压等级的器件，工作温度范围宽。

### 数据传输方式

#### 字节格式

发送到SDA 线上的每个字节必须为8 位，每次传输可以发送的字节数量不受限制。每个字节后必须跟一个响应位。首先传输的是数据的最高位（MSB），如果从机要完成一些其他功能后（例如一个内部中断服务程序）才能接收或发送下一个完整的数据字节，可以使时钟线SCL 保持低电平，迫使主机进入等待状态，当从机准备好接收下一个数据字节并释放时钟线SCL 后数据传输继续。

#### 应答响应

数据传输必须带响应，相关的响应时钟脉冲由主机产生。在响应的时钟脉冲期间发送器释放SDA 线（高）。

在响应的时钟脉冲期间，接收器必须将SDA 线拉低，使它在这个时钟脉冲的高电平期间保持稳定的低电平。  
通常被寻址的接收器在接收到的每个字节后，除了用CBUS 地址开头的数据，必须产生一个响应。当从机不能响应从机地址时（例如它正在执行一些实时函数不能接收或发送），从机必须使数据线保持高电平，主机然后产生一个停止条件终止传输或者产生重复起始条件开始新的传输。

如果从机接收器响应了从机地址，但是在传输了一段时间后不能接收更多数据字节，主机必须再一次终止传输。这个情况用从机在第一个字节后没有产生响应来表示。从机使数据线保持高电平，主机产生一个停止或重复起始条件。

如果传输中有主机接收器，它必须通过在从机发出的最后一个字节时产生一个响应，向从机发送器通知数据结束。从机发送器必须释放数据线，允许主机产生一个停止或重复起始条件。

#### 时钟同步

所有主机在SCL线上产生它们自己的时钟来传输I2C总线上的报文。数据只在时钟的高电平周期有效，因此需要一个确定的时钟进行逐位仲裁。

时钟同步通过线与连接I2C 接口到SCL 线来执行。这就是说SCL 线的高到低切换会使器件开始数它们的低电平周期，而且一旦器件的时钟变低电平，它会使SCL 线保持这种状态直到到达时钟的高电平。但是如果另一个时钟仍处于低电平周期，这个时钟的低到高切换不会改变SCL 线的状态。因此SCL 线被有最长低电平周期的器件保持低电平。此时低电平周期短的器件会进入高电平的等待状态。

当所有有关的器件数完了它们的低电平周期后，时钟线被释放并变成高电平。之后，器件时钟和SCL线的状态没有差别，而且所有器件会开始数它们的高电平周期。首先完成高电平周期的器件会再次将SCL线拉低。

这样产生的同步SCL 时钟的低电平周期由低电平时钟周期最长的器件决定，而高电平周期由高电平时钟周期最短的器件决定。

### 传输模式

#### 快速模式

快速模式器件可以在400kbit/s 下接收和发送。最小要求是：它们可以和400kbit/s 传输同步，可以延长SCL 信号的低电平周期来减慢传输。快速模式器件都向下兼容，可以和标准模式器件在0~100kbit/s 的I2C 总线系统通讯。但是，由于标准模式器件不向上兼容，所以不能在快速模式I2C 总线系统中工作。快速模式I2C 总线规范与标准模式相比有以下特征：

1. 最大位速率增加到400kbit/s；
2. 调整了串行数据（SDA） 和串行时钟（SCL ）信号的时序；
3. 快速模式器件的输入有抑制毛刺的功能，SDA 和SCL输入有施密特触发器；
4. 快速模式器件的输出缓冲器对SDA 和SCL 信号的下降沿有斜率控制功能；
5. 如果快速模式器件的电源电压被关断，SDA 和SCL 的I/O 管脚必须悬空，不能阻塞总线；
6. 连接到总线的外部上拉器件必须调整以适应快速模式I2C 总线更短的最大允许上升时间。对于负载最大是200pF 的总线，每条总线的上拉器件可以是一个电阻，对于负载在200pF~400pF 之间的总线，上拉器件可以是一个电流源（最大值3mA ）或者是一个开关电阻电路。

#### 高速模式

高速模式（Hs 模式）器件对I2C 总线的传输速度有巨大的突破。Hs 模式器件可以在高达3.4Mbit/s 的位速率下传输信息，而且保持完全向下兼容快速模式或标准模式（F/S 模式）器件，它们可以在一个速度混合的总线系统中双向通讯。  
Hs 模式传输除了不执行仲裁和时钟同步外，与F/S 模式系统有相同的串行总线协议和数据格式。

高速模式下I2C 总线规范如下：

1. Hs 模式主机器件有一个SDAH 信号的开漏输出缓冲器和一个在SCLH 输出的开漏极下拉和电流源上拉电路。这个电流源电路缩短了SCLH 信号的上升时间，任何时候在Hs 模式，只有一个主机的电流源有效；
2. 在多主机系统的Hs 模式中，不执行仲裁和时钟同步，以加速位处理能力。仲裁过程一般在前面用F/S 模式传输主机码后结束；
3. Hs 模式主机器件以高电平和低电平是1:2 的比率产生一个串行时钟信号。解除了建立和保持时间的时序要求；
4. 可以选择Hs 模式器件有内建的电桥。在Hs 模式传输中，Hs 模式器件的高速数据（SDAH）和高速串行时钟（SCLH ）线通过这个电桥与F/S 模式器件的SDA 和SCL 线分隔开来。减轻了SDAH 和SCLH 线的电容负载，使上升和下降时间更快；
5. Hs 模式从机器件与F/S 从机器件的唯一差别是它们工作的速度。Hs 模式从机在SCLH 和SDAH输出有开漏输出的缓冲器。SCLH 管脚可选的下拉晶体管可以用于拉长SCLH 信号的低电平，但只允许在Hs 模式传输的响应位后进行；
6. Hs 模式器件的输出可以抑制毛刺，而且SDAH 和SCLH 输出有一个施密特触发器；
7. Hs 模式器件的输出缓冲器对SDAH 和SCLH 信号的下降沿有斜率控制功能。

## SPI

SPI接口的全称是"Serial Peripheral Interface"，意为串行外围接口,是Motorola首先在其MC68HCXX系列处理器上定义的。SPI接口主要应用在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。

SPI接口是在CPU和外围低速器件之间进行同步串行数据传输，在主器件的移位脉冲下，数据按位传输，高位在前，低位在后，为全双工通信，数据传输速度总体来说比I2C总线要快，速度可达到几Mbps。

### 接口

1. MOSI – 主器件数据输出，从器件数据输入
2. MISO – 主器件数据输入，从器件数据输出
3. SCLK –时钟信号，由主器件产生,最大为fPCLK/2，从模式频率最大为fCPU/2
4. NSS – 从器件使能信号，由主器件控制,有的IC会标注为CS(Chip select)

在点对点的通信中，SPI接口不需要进行寻址操作，且为全双工通信，显得简单高效。**在多个从器件的系统中，每个从器件需要独立的使能信号，硬件上比I2C系统要稍微复杂一些。**

SPI接口在内部硬件实际上是两个简单的移位寄存器，传输的数据为8位，在主器件产生的从器件使能信号和移位脉冲下，按位传输，高位在前，低位在后。

### 优缺点

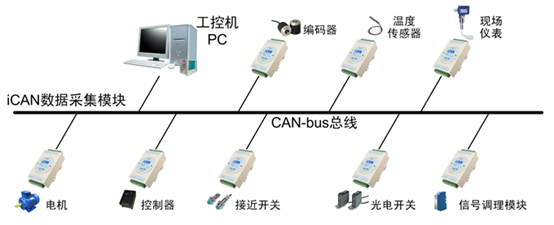
* 协议简单，相对数据速率高。
* 占用的Pin口较多
* 没有指定的流控制，没有应答机制确认是否接收到数据

## CAN总线

嵌入式的工程师一般都知道CAN总线广泛应用到汽车中，其实船舰电子设备通信也广泛使用CAN，随着国家对海防的越来越重视，对CAN的需求也会越来越大

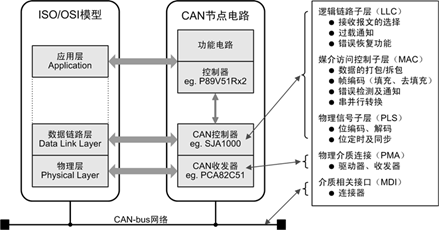
### 概述

CAN（Controller Area Network）即控制器局域网，是一种能够实现分布式实时控制的串行通信网络。想到CAN就要想到德国的Bosch公司，因为CAN就是这个公司开发的（和Intel）。CAN有很多优秀的特点，使得它能够被广泛的应用。比如：传输速度最高到1Mbps，通信距离最远到10km，无损位仲裁机制，多主结构。近些年来，CAN控制器价格越来越低，很多MCU也集成了CAN控制器。现在每一辆汽车上都装有CAN总线。一个典型的CAN应用场景

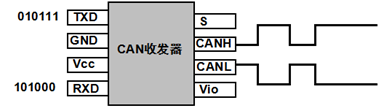


### CAN总线标准

CAN总线标准只规定了物理层和数据链路层，需要用户自定义应用层。不同的CAN标准仅物理层不同

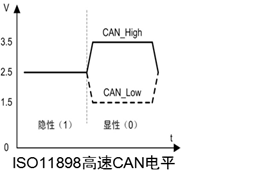


CAN收发器负责逻辑电平和物理信号之间的转换

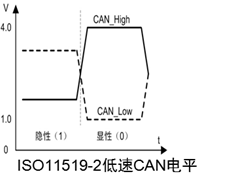


将逻辑信号转换成物理信号（差分电平），或者将物理信号转换成逻辑电平。

CAN标准有两个，即IOS11898和IOS11519，两者差分电平特性不同



高低电平幅度低，对应的传输速度快

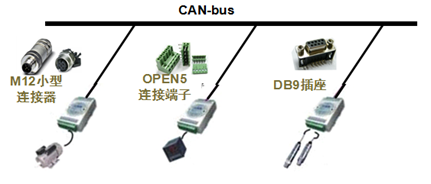


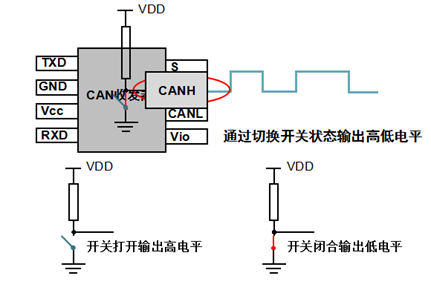


双绞线共模消除干扰，是因为电平同时变化，电压差不变。

### 物理层

CAN有三种接口器件

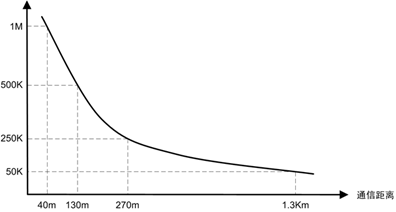




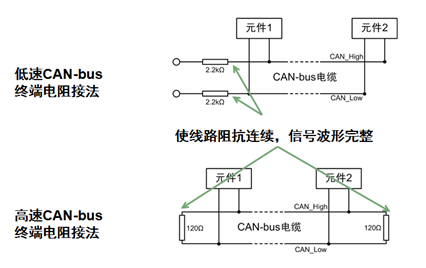
多个节点连接，只要有一个为低电平，总线就为低电平，只有所有节点输出高电平时，才为高电平。所谓"线与"。

CAN总线有5个连续相同位后，就插入一个相反位，产生跳变沿，用于同步。从而消除累积误差。

和485、232一样，CAN的传输速度与距离成反比。



CAN总线，终端电阻的接法：



为什么是120Ω，因为电缆的特性阻抗为120Ω，为了模拟无限远的传输线

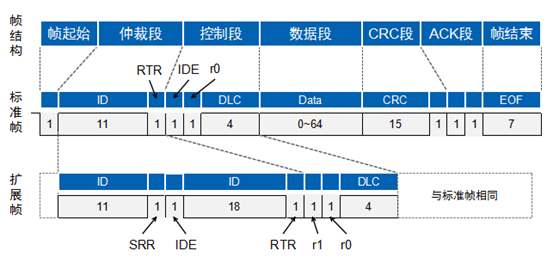
### 数据链路层

CAN总线传输的是CAN帧，CAN的通信帧分成五种，分别为数据帧、远程帧、错误帧、过载帧和帧间隔。

数据帧用来节点之间收发数据，是使用最多的帧类型；远程帧用来接收节点向发送节点接收数据；错误帧是某节点发现帧错误时用来向其他节点通知的帧；过载帧是接收节点用来向发送节点告知自身接收能力的帧；用于将数据帧、远程帧与前面帧隔离的帧。

数据帧根据仲裁段长度不同分为标准帧（2.0A）和扩展帧（2.0B）

#### 帧起始



帧起始由一个显性位（低电平）组成，发送节点发送帧起始，其他节点同步于帧起始；

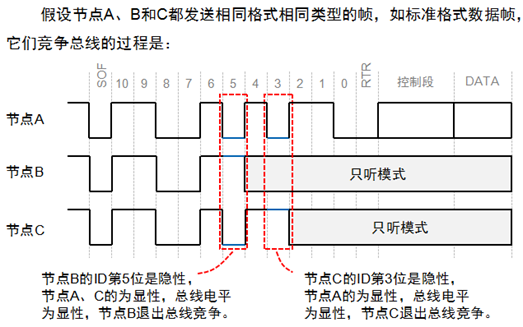
帧结束由7个隐形位（高电平）组成。

#### 仲裁段

CAN总线是如何解决多点竞争的问题？

由仲裁段给出答案。

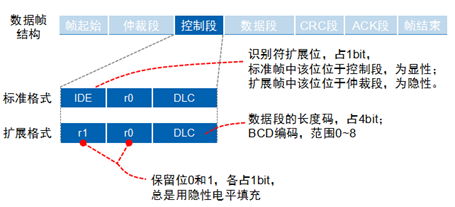
CAN总线控制器在发送数据的同时监控总线电平，如果电平不同，则停止发送并做其他处理。如果该位位于仲裁段，则退出总线竞争；如果位于其他段，则产生错误事件。



帧ID越小，优先级越高。由于数据帧的RTR位为显性电平，远程帧为隐性电平，所以帧格式和帧ID相同的情况下，数据帧优先于远程帧；由于标准帧的IDE位为显性电平，扩展帧的IDE位为隐形电平，对于前11位ID相同的标准帧和扩展帧，标准帧优先级比扩展帧高。

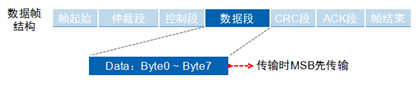
#### 控制段

共6位，标准帧的控制段由扩展帧标志位IDE、保留位r0和数据长度代码DLC组成；扩展帧控制段则由IDE、r1、r0和DLC组成。



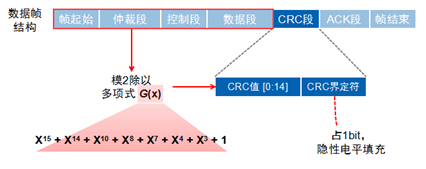
#### 数据段

为0-8字节，短帧结构，实时性好，适合汽车和工控领域；



#### CRC段

CRC校验段由15位CRC值和CRC界定符组成。



#### ACK段

当接收节点接收到的帧起始到CRC段都没错误时，它将在ACK段发送一个显性电平，发送节点发送隐性电平，线与结果为显性电平。

### 远程帧

远程帧分为6个段，也分为标准帧和扩展帧，且RTR位为1（隐性电平）



CAN是可靠性很高的总线，但是它也有五种错误。

CRC错误：发送与接收的CRC值不同发生该错误；

格式错误：帧格式不合法发生该错误；

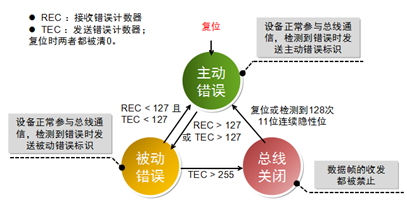
应答错误：发送节点在ACK阶段没有收到应答信息发生该错误；

位发送错误：发送节点在发送信息时发现总线电平与发送电平不符发生该错误；

位填充错误：通信线缆上违反通信规则时发生该错误。

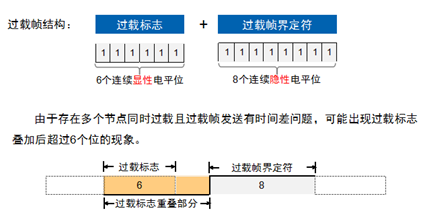
当发生这五种错误之一时，发送节点或接受节点将发送错误帧

为防止某些节点自身出错而一直发送错误帧，干扰其他节点通信，CAN协议规定了节点的3种状态及行为



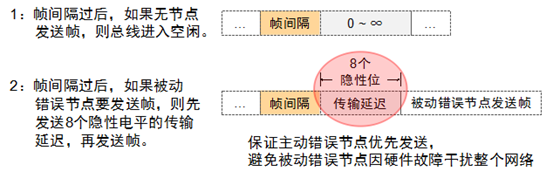
### 过载帧

当某节点没有做好接收的"准备"时，将发送过载帧，以通知发送节点。



### 帧间隔

用来隔离数据帧、远程帧与他们前面的帧，错误帧和过载帧前面不加帧间隔。

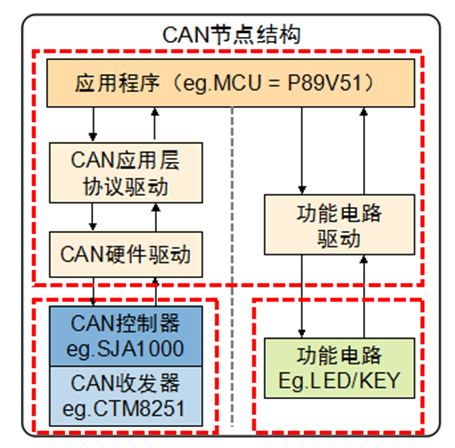


//好好理解1.6最后一张ppt

### 构建CAN节点

构建节点，实现相应控制，由底向上分为四个部分：CAN节点电路、CAN控制器驱动、CAN应用层协议、CAN节点应用程序。

虽然不同节点完成的功能不同，但是都有相同的硬件和软件结构。

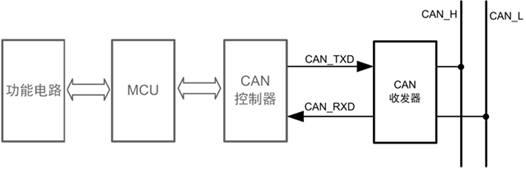


CAN收发器和控制器分别对应CAN的物理层和数据链路层，完成CAN报文的收发；功能电路，完成特定的功能，如信号采集或控制外设等；主控制器与应用软件按照CAN报文格式解析报文，完成相应控制。

CAN硬件驱动是运行在主控制器（如P89V51）上的程序，它主要完成以下工作：基于寄存器的操作，初始化CAN控制器、发送CAN报文、接收CAN报文；

如果直接使用CAN硬件驱动，当更换控制器时，需要修改上层应用程序，移植性差。在应用层和硬件驱动层加入虚拟驱动层，能够屏蔽不同CAN控制器的差异。

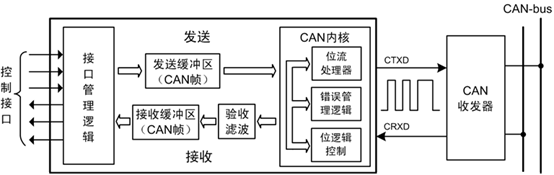
一个CAN节点除了完成通信的功能，还包括一些特定的硬件功能电路，功能电路驱动向下直接控制功能电路，向上为应用层提供控制功能电路函数接口。特定功能包括信号采集、人机显示等。



CAN收发器是实现CAN控制器逻辑电平与CAN总线上差分电平的互换。实现CAN收发器的方案有两种，一是使用CAN收发IC（需要加电源隔离和电气隔离），另一种是使用CAN隔离收发模块。推荐使用第二种。

CAN控制器是CAN的核心元件，它实现了CAN协议中数据链路层的全部功能，能够自动完成CAN协议的解析。CAN控制器一般有两种，一种是控制器IC（SJA1000），另一种是集成CAN控制器的MCU（LPC11C00）。

MCU负责实现对功能电路和CAN控制器的控制：在节点启动时，初始化CAN控制器参数；通过CAN控制器读取和发送CAN帧；在CAN控制器发生中断时，处理CAN控制器的中断异常；根据接收到的数据输出控制信号；



接口管理逻辑：解释MCU指令，寻址CAN控制器中的各功能模块的寄存器单元，向主控制器提供中断信息和状态信息。

发送缓冲区和接收缓冲区能够存储CAN总线网络上的完整信息。

验收滤波是将存储的验证码与CAN报文识别码进行比较，跟验证码匹配的CAN帧才会存储到接收缓冲区。

CAN内核实现了数据链路的全部协议。

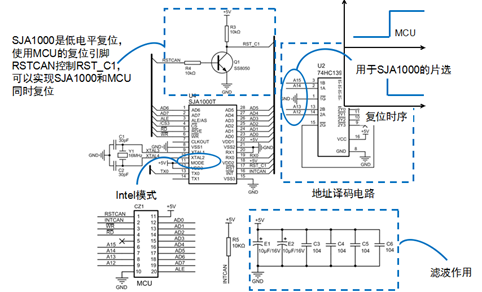
### CAN协议应用层概述

CAN总线只提供可靠的传输服务，所以节点接收报文时，要通过应用层协议来判断是谁发来的数据、数据代表了什么含义。常见的CAN应用层协议有： CANOpen、DeviceNet、J1939、iCAN等。

CAN应用层协议驱动是运行在主控制器（如P89V51）上的程序，它按照应用层协议来对CAN报文进行定义、完成CAN报文的解析与拼装。例如，我们将帧ID用来表示节点地址，当接收到的帧ID与自身节点ID不通过时，就直接丢弃，否则交给上层处理；发送时，将帧ID设置为接收节点的地址。

### CAN收发器

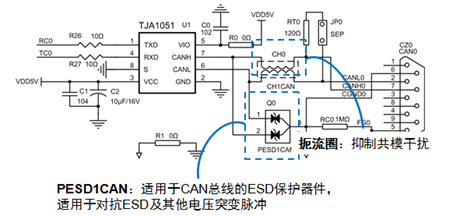
SJA1000的输出模式有很多，使用最多的是正常输出模式，输入模式通常不选择比较器模式，可以增大通信距离，并且减少休眠下的电流。



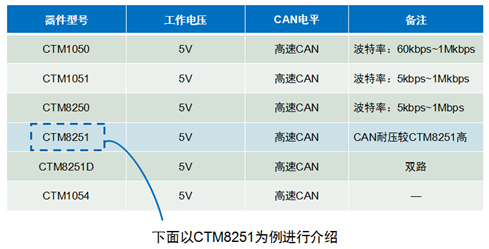
收发器按照通信速度分为高速CAN收发器和容错CAN收发器。

同一网络中要使用相同的CAN收发器。

CAN连接线上会有很多干扰信号，需要在硬件上添加滤波器和抗干扰电路

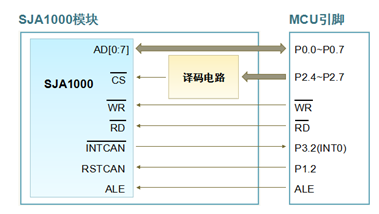


也可以使用CAN隔离收发器（集成滤波器和抗干扰电路）。

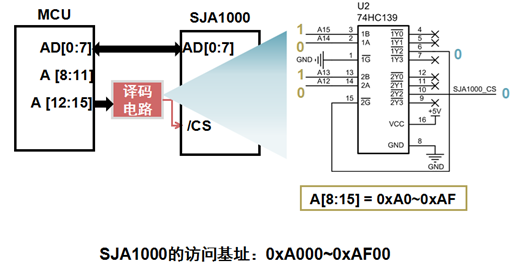




CAN控制器与MCU的连接方式



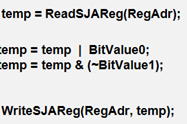
SJA1000可被视为外扩RAM，地址宽度8位，最多支持256个寄存器





|  |
| --- |
| #define REG\_BASE\_ADDR 0xA000 // 寄存器基址  unsigned char \*SJA\_CS\_Point = (unsigned char \*) REG\_BASE\_ADDR ;    // 写SJA1000寄存器  void WriteSJAReg(unsigned char RegAddr, unsigned char Value) {  \*(SJA\_CS\_Point + RegAddr) = Value;  return;  }    // 读SJA1000寄存器  unsigned char ReadSJAReg(unsigned char RegAddr) {   return (\*(SJA\_CS\_Point + RegAddr));  } |



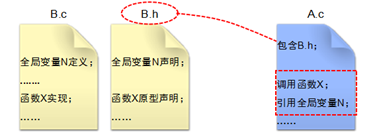


将缓存区的数据连续写入寄存器

…… for (i=0;i<len;i++) { WriteSJAReg(RegAdr+i,ValueBuf[i]); } ……

将连续多个寄存器连续读入缓存区

…… for (i=0;i<len;i++) { ReadSJAReg(RegAdr+i,ValueBuf[i]); } ……



头文件包含方案：

1. 每个程序包含用到的头文件
2. 每个程序包含一个公用头文件，公用头文件包含所有其他头文件  
   1. #ifndef \_\_CONFIG\_H\_\_ // 防止头文件被重复包含
   2. #define \_\_CONFIG\_H\_\_
   3. #include <8051.h>         // 包含80C51寄存器定义头文件
   4. #include "SJA1000REG.h"         // 包含SJA1000寄存器定义头文件
   6. // 定义取字节运算
   7. #define LOW\_BYTE(x)  (unsigned char)(x)
   8. #define HIGH\_BYTE(x)  (unsigned char)((unsigned int)(x) >> 8)
   10. // 定义振荡器时钟和处理器时钟频率（用户可以根据实际情况作出调整）
   11. #define OSCCLK 11059200UL
   12. // 宏定义MCU的时钟频率
   13. #define CPUCLK (OSCCLK / 12)
   14. #endif // \_\_CONFIG\_H\_\_

（1）置位模式寄存器Bit0位进入复位模式；SJA1000上电后处于复位状态，必须初始化后才能工作。

（2）设置时钟分频寄存器选择时钟频率、CAN模式；

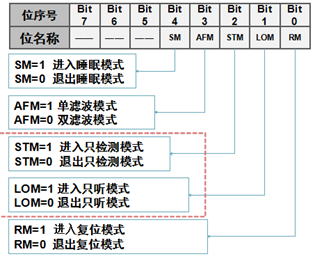
（3）设置验收滤波，设定验证码和屏蔽码；

（4）设置总线定时器寄存器0、1设定CAN波特率；

（5）设置输出模式；

（6）清零模式寄存器Bit0位退出复位模式；

模式寄存器



只检测模式：SJA1000发送CAN帧时不检查应答位；

只听模式：此模式下SJA1000不会发送错误帧，用于自动检测波特率；SJA1000以不同的波特率接收CAN帧，当收到CAN帧时，表明当前波特率与总线波特率相同。

### 波特率设置

CAN总线无时钟，使用异步串行传输；波特率是1秒发送的数据位；

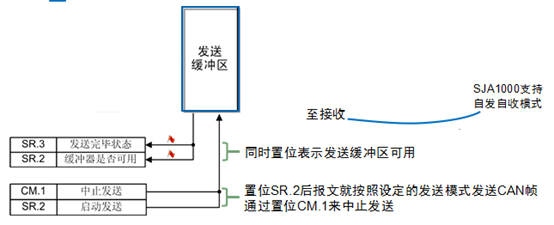


### CAN帧发送：

发送CAN帧的步骤：1.检测状态寄存器，等待发送缓冲区可用；

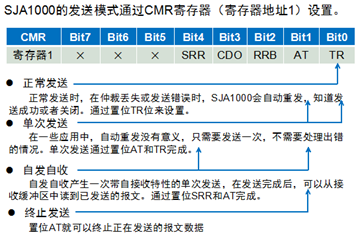
2.填充报文到发送缓冲区；

3.启动发送。



SJA1000具有一个12字节的缓冲区，要发送的报文可以通过寄存器16-28写入，也可通过寄存器96-108写入或读出





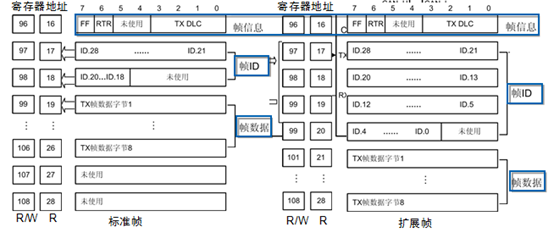
设置发送模式

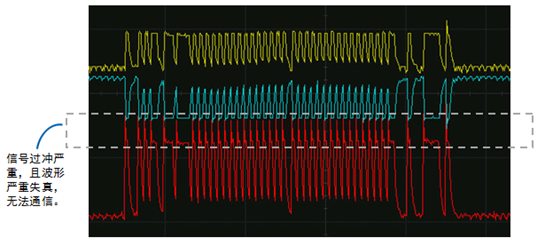
1. char SetSJASendCmd(unsigned char cmd) {
2. unsigned char ret;
3. switch (cmd) {
4. default:
5. case 0:
6. ret = SetBitMask(REG\_CAN\_CMR, TR\_BIT); //正常发送
7. break;
8. case 1:
9. ret = SetBitMask(REG\_CAN\_CMR, TR\_BIT|AT\_BIT); //单次发送
10. break;
11. case 2:
12. ret = SetBitMask(REG\_CAN\_CMR, TR\_BIT|SRR\_BIT);//自收自发
13. break;
14. case 0xff:
15. ret = SetBitMask(REG\_CAN\_CMR, AT\_BIT);//终止发送
16. break;
17. }
18. return ret;
19. }

发送函数

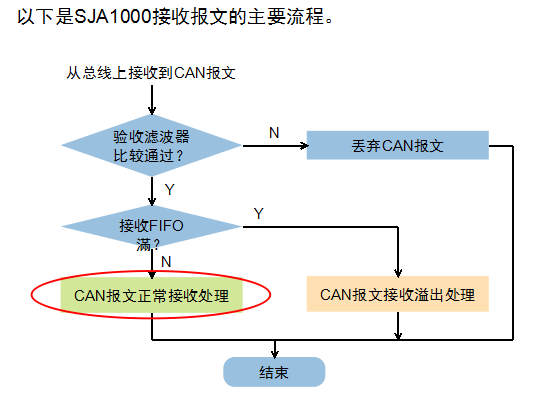
1. unsigned char SJA\_CAN\_Filter[8] = { // 定义验收滤波器的参数，接收所有帧
2. 0x00, 0x00, 0x00, 0x00,
3. // ACR0~ACR3
4. 0xff, 0xff, 0xff, 0xff
5. // AMR0~AMR3
6. };
7. unsigned char STD\_SEND\_BUFFER[11] = { // CAN 发送报文缓冲区
8. 0x08, // 帧信息，标准数据帧，数据长度 = 8
9. 0xEA, 0x60, // 帧ID = 0x753
10. 0x55, 0x55, 0x55, 0x55, 0xaa, 0xaa, 0xaa, 0xaa // 帧数据
11. };
12. void main(void) // 主函数，程序入口
13. {
14. timerInit();// 初始化
15. D1 = 0;
16. SJA1000\_RST = 1; // 硬件复位SJA1000
17. timerDelay(50); // 延时500ms
18. SJA1000\_RST = 0;
19. SJA1000\_Init(0x00, 0x14, SJA\_CAN\_Filter); // 初始化SJA1000，设置波特率为1Mbps
20. // 无限循环，main()函数不允许返回
21. for(;;) {
22. SJASendData(STD\_SEND\_BUFFER, 0x0);
23. timerDelay(100); // 延时1000ms
24. }
25. }

为什么帧ID是0x753，这与CAN帧在缓冲区的存储格式有关。





终端电阻非常重要，当波特率较高而且没加终端电阻时，信号过冲非常严重。



SJA1000有64个字节的接收缓冲区（FIFO），这可以降低对MCU的要求。MCU可以通过查询或中断的方式确定SJA1000接收到报文后读取报文。

参考资料：《项目驱动-CAN-bus现场总线基础教程》 广州周立功单片机科技有限公司

# 参考

https://www.jianshu.com/p/df2ed057b4a0