AlanTu

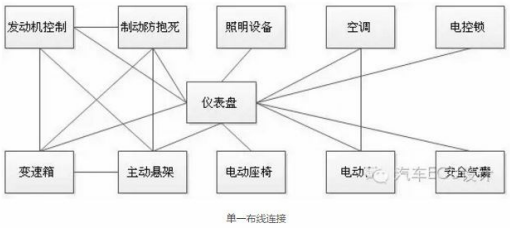
随笔 - 952, 文章 - 0, 评论 - 44, 阅读 - 188万

CAN总线基础

CAN总线基础（上）

概述

汽车电子设备的不断增多，对汽车上的线束分布以及信息共享与交流提出了更高的要求。传统的电气系统往往采用单一连接的方式通信，这必将带来线束的冗余以及维修的成本的提高。



单一布线连接

传统的单一通信的对接方式，已经不能满足现代汽车电子发展的需求，采用更为先进的总线技术势在必行。总线技术可以实现信息的实时共享、解决了传统布线方式中线束多、布线难、成本高等问题，从而提高整车通信的质量与品质。

CAN总线（Controller Area Network，控制器局域网络）由德国博世公司于上世纪80年代提出，近20年来，随着CAN总线在工业测控与汽车领域的普及，CAN网络技术不断优化，取得了长足发展。如今CAN总线已经成为了汽车上不可或缺的重要环节，ECU内部的CAN总线开发也占到了ECU开发中的很大分量。在汽车中为了满足车载系统的不同要求，主要采用高速CAN和低速CAN。这两者以不同的总线速率工作以获得最佳的性价比，在两条总线之间采用CAN网关进行连接。

（1）高速CAN（动力总线）

高速CAN总线的传输速率范围在125kbit/s - 1Mbit/s之间，主要用于传动系数传输的实时性要求（如发动机控制、自动变速箱控制、行驶稳定系统、组合仪表等）。

（2）低速CAN（舒适总线）

低速CAN总线的传输速率范围在5kbit/s - 125kbit/s之间。主要用于舒适系统和车身系统的数据传输的实时性要求（如空调控制、座椅调节、车窗升降等）。

整车CAN网络示意图

CAN总线特点

CAN总线是一种串行数据通讯协议，其中包含了CAN协议的物理层以及数据链路层。可以完成对数据的位填充，数据块编码，循环冗余效验，帧优先级的判别等工作。其主要特点如下：

（1）多主机方式工作，网络上任意一个节点（未脱离总线）均可以随时向总线网络上发布报文帧。

（2）节点发送的报文帧可以分为不同的优先级，满足不同实时要求。

（3）采用载波侦听多路访问/冲突检测（CSMA/CD）技术，当两个节点同时发布信息时，高优先级报文可不受影响地传输数据。

（4）节点总数实际可达110个。

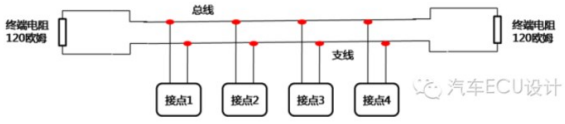
（5）采用短帧结构，每一帧最多有8个有效字节。

（6）当某个节点错误严重时，具有自动关闭功能，切断与总线的联系，致使总线上的其他操作不受影响。

CAN总线物理层

（1）总线结构

CAN总线采用双线传输，两根导线分别作为CAN_H、CAN_L，并在终端配备有120Ω的电阻。收到总线信号时，CAN收发器将信号电平转化为逻辑状态，即CAN_H与CAN_L电平相减后，得到一个插值电平。各种干扰（如点火系统）在两根导线上的作用相同，相减后得到的插值电平可以滤过这些干扰。

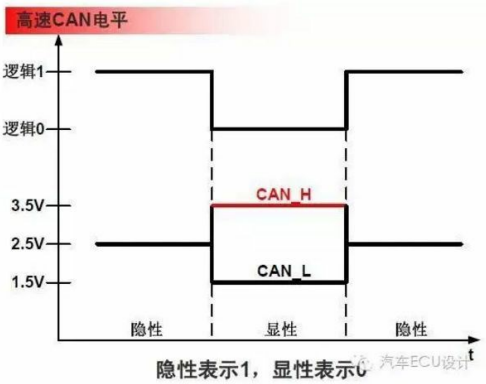


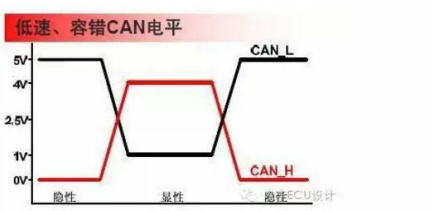
（2）总线电平

CAN总线有两种逻辑电平状态，即显性与隐性。显性电平代表“0”，隐性电平代表“1”。采用非归零码编码，即在两个相同电平之间并不强制插入一个零状态电平。

高速CAN在传输隐性位时，CAN_H与CAN_L上的电平位均为2.5V；在传输显性位时分别为3.5V与1.5V。

低速CAN在传输隐性状态位时，CAN_H上的电平为0V，CAN_L上的电平位5V。在传输显性状态位时，CAN_H上的电平位3.6V，CAN_L的位1.4V。





为了确保通讯的正确性，总线信号必须在一定时间内出现在总线上，并且保证被正确采样。总线信号传输有一定的时间延迟，最大的可靠的总线波特率与总线长度有关。ISO11898中对各种总线长度有着以下定义：

★ 1Mbit/s 总线长度为40m（规范）。

★ 500kbit/s 总线长度最大值为100m（建议值）。

★ 250kbit/s 总线长度最大值为250m。

★ 125kbit/s 总线长度最大值为500m。

★ 40kbit/s 总线长度最大值为1000m。

CAN总线硬件设备

（1）CAN通信线缆，实现节点的互联，是传输数据的通道。主要有：普通双绞线，同轴电缆，光纤。

（2）CAN驱动/接收器，将信息封装为帧后发送，接收到的帧将其还原为信息、标定并报告节点状态。

（3）CAN控制器，专按协议要求设计制造，经简单总线连接即可实现CAN的全部功能。包括：SJA1000（Philips），82527（Intel）。

（4）CAN微控制器，嵌有部分或全部CAN控制模块及相关接口的通用型微控制器现如今很多芯片都配备CAN接口。

CAN总线基础（下）

CAN报文帧结构

在CAN总线上，报文是以“帧”来发送的，每一帧都包含以下几个部分：

（1）帧起始

在总线空闲时，总线为隐性状态。帧起始由单个显性位构成，标志着报文的开始，并在总线上起着同步作用。

（2）仲裁段

仲裁的主要是定义了报文的标识符，也俗称ID。在CAN2.0A规范中，标识符为11位，而在CAN2.0B中变为了29位。这意味着在2.0B中可以存在更多不同类型的报文，但是也降低了总线的利用率。

（3）控制段

主要定义了数据域字节的长度。通过数据长度码，接收节点可以判断报文数据是否完整。

（4）数据域

包含有0~8个字节数据。

（5）CRC域

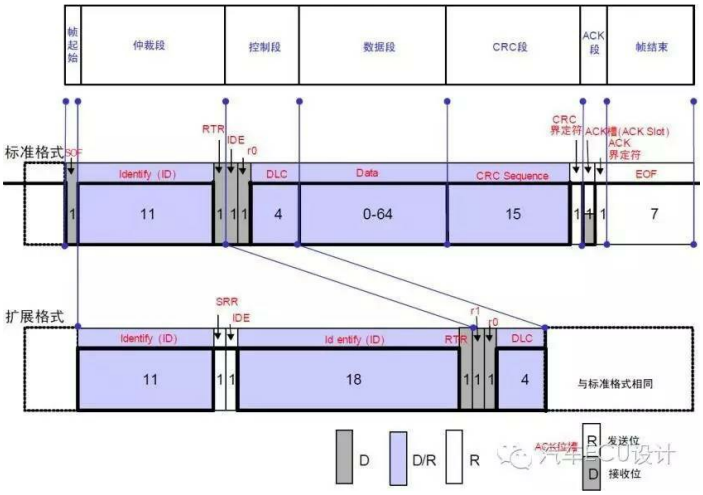
CRC又称循环冗余码效验（Cyclical Redundancy Check），是数据通信中常见的查错方法。

（6）ACK域

用于接收节点的反馈应答。

（7）帧结束

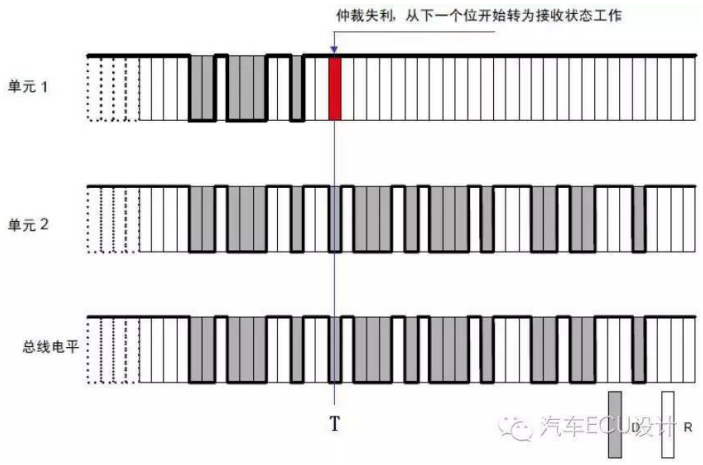
由一串7个隐性位组成，表示报文帧的结束。



Ps：在CAN总线的开发中，核心的关注点就是CAN报文ID以及其数据域。根据客户的要求，ECU接收自己感兴趣的ID报文的同时，也向外发送别的ECU所需要的ID报文。一般不同整车厂在开发自己的CAN协议规范的同时，也会有自己的checksum机制，不满足checksum的报文，数据将不会被ECU所接收。

仲裁机制

仲裁是总线应用中一个相当重要的概念，在CAN总线采用载波侦听多路访问/冲突检测(CSMA/CD)技术。如果总线空闲(隐性位)，有报文准备发送，那么每一个节点都可以开始发送报文。报文以显性位(报文帧开始位)开始，接着是标识符。如果多个节点同时开始发送报文，那么使用“线与”仲裁机制(仲裁用逻辑“与”)来解决总线冲突，确定优先级最高的报文，而不需要损失时间或数据(非破坏性仲裁)。仲裁机制使用标识符为判断依据，不仅代表报文帧的内容，还代表报文帧发送的优先级。二进制数越小的标识符，优先级越高；反之亦然。



如上图，ECU单元1和ECU单元2同时开始向总线发送数据，开始部分他们的数据格式是一样的，故无法区分优先级，直到T时刻，单元1输出隐性电平，而单元2输出显性电平，此时单元1仲裁失利，立刻转入接收状态工作，不再与单元2竞争，而单元2则顺利获得总线使用权，继续发送自己的数据。

CAN报文帧种类

CAN总线报文传输有以下4种不同的格式：

（1）数据帧：由发送节点发出，包含0 - 8个数据字节。

（2）远程帧：发送远程帧向网络节点请求发送某一标识符的数据帧。

（3）错误帧：总线节点发现错误时，以错误帧的方式通知网络上的其他节点。

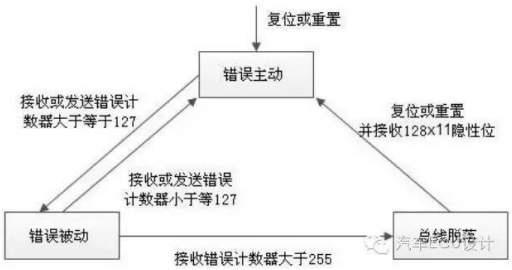
（4）过载帧：发送过载帧，表示当前节点不能处理后续的报文（如帧延迟等）。

Ps：为了保持总线的利用率，在车载总线上数据帧的报文一般均为8字节。

CAN总线错误

CAN总线将错误分为临时性错误和长期性错误。前者主要由外部因素引起，如总线上驱动电压波形不规整、有尖峰或毛刺时，其数据传输性能会受到一定程度的短期干扰。长期性错误则主要由网络组建非正常状况引起，比如接触不良、线路故障、发送器或接收器失效等。CAN中每个具有数据通信能力的网络单元内部都集成有一个发送错误计数器和接受错误计数器，当该单元在数据发送阶段出现一次错误时，其发送错误计数器自加8；在数据接收阶段出现一次错误时，其接收错误计数器自加1。在相应计数器内容非0的情况下，网络单元每成功发送一帧，发送错误计数器自减1；每成功接收一帧，接收错误计数内容原本小于127时自减1，大于127时被置为119 - 127之间任意值。这样，如果某个网络单元的错误计数在不断增长，就说明该单元的数据通信在频繁发生故障。当计数器内容超过一定阈值时，

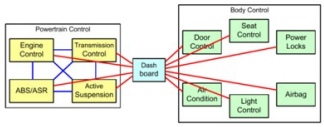
可以认为该故障是由长期性错误引起的。这种机制保证了当某一个节点出现故障的时候，不会造成总线长时间瘫痪。



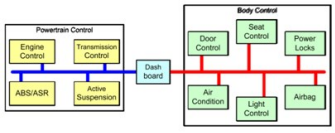
一口气从零读懂CAN总线 上

概要

上世纪八十年代以来，汽车ECU越来越多，如ABS，电控门窗，电子燃油喷射装置。如果仍然采用常规的点对点布线方式，即电线一段与开关相接，另一端与用电设备相通，将会导致车上电线数目的急剧增加，从而带来线束的冗余及维修成本的提高。这就对汽车的线束分布及信息通讯提出了更高的要求。总线技术可以实现信息的实时共享，解决了传统布线方式中线束多，布线难，成本高等问题，CAN（Controller Area Network）总线技术应运而生。

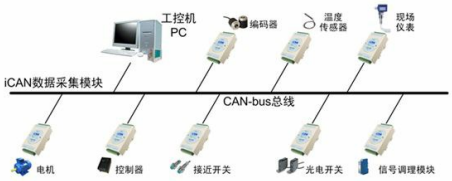


点对点通信



总线通信

CAN总线是由德国研发和生产汽车电子产品著称的BOSCH公司开发的，并最终成为国际标准（ISO11519），是国际上应用最广泛的现场总线之一。CAN总线是一种多主控（Multi-Master）的总线系统。传统总线系统如USB或以太网等是在总线控制器的协调下，实现从A节点到B节点大量数据的传输。CAN网络的消息是广播式的，即在同一时刻网络上所有节点侦测的数据是一致的，它是一种基于消息广播模式的串行通信总线。



CAN总线的很多优点，使得它得到了广泛的应用，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。

CAN总线标准

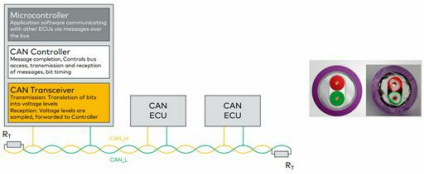
CAN总线标准之规定了物理层和数据链路层，至于应用层需要用户自定义。不同的CAN标准仅物理层不同。物理层和数据链路层：ISO11898；应用层：不同的应用领域使用不同的应用层标准。

CAN物理层

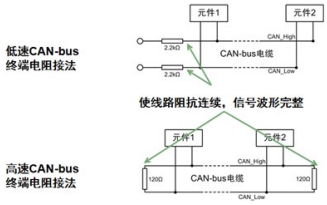
CAN拓扑网络

连接在CAN总线上的设备叫做节点设备（CAN Node），CAN网络的拓扑一般为线型。线束最常用的是双绞线，线上传输为对称的差分电平信号。下图为CAN总线网络示意图，节点主要包括Host、控制器和收发器。Host常集成有CAN控制器，CAN控制器负责处理协议相关功能，以减轻Host的负担。CAN收发器将控制器连接到传输媒介。通常控制器和总线收发器通过光耦或磁耦隔离，这样即使总线上过压损坏收发器，控制器和Host设备也可以得到保护。

在发送数据时，CAN控制器把要发送的二进制编码通过CAN_Tx线发送到CAN收发器，然后由收发器把这个普通的逻辑电平信号转化成差分信号，通过差分线CAN_High和CAN_Low输出到CAN总线网络。接收数据过程，相反。采用差分信号，可以取得更好的电磁兼容效果。因此，CAN总线物理传输媒介只需要两根线。



高速CAN总线最高信号传输速率为1Mbps，支持最长距离40m。ISO11898-2要求在高速CAN总线两段安装**端接电阻**RL（端接电阻一般为120Ω，因为电缆的特性阻抗为120 Ω，为了模拟无限远的传输线。）以消除反射。低速CAN最高速度只有125Kbps，所以ISO11898-3没有端接要求。

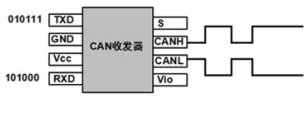


因为传输距离越大，信号时延也越大，为了保证消息的正确采样，总线上的信号速率相应也要下降。下图是推荐的信号速率与距离的关系。

Bus Length (m)	Signaling Rate (Mbps)
40	1
100	0.5
200	0.25
500	0.10
1000	0.05

CAN收发器

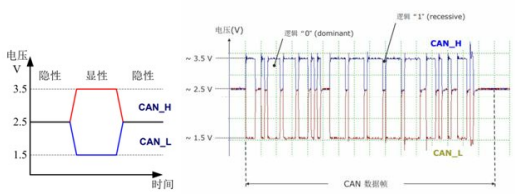
CAN总线分高速CAN和低速CAN，收发器也分为高速CAN收发器（1Mbps）和低速CAN收发器（125Kbps）。低速CAN也叫Fault
Tolerance CAN，指的是即使总线上一根线失效，总线依然可以通信。如同串口中的MAX3232用作电平转换，CAN收发器的作用则是把逻辑信号转换为差分信号。



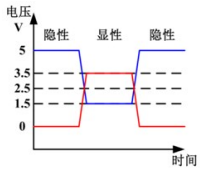
差分信号

CAN总线采用差分信号传输，通常情况下只需要两根信号线就可以进行正常的通信。在差分信号中，逻辑0和逻辑1是用两根差分信号线的电压差来表示。当处于逻辑1，CAN_High和CAN_Low的电压差小于0.5V时，称为隐性电平（Recessive）；当处于逻辑0，CAN_High和CAN_Low的电压差大于0.9V，称为显性电平（Dominant）。

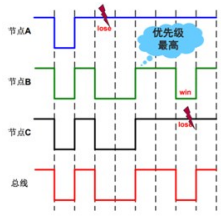
高速CAN



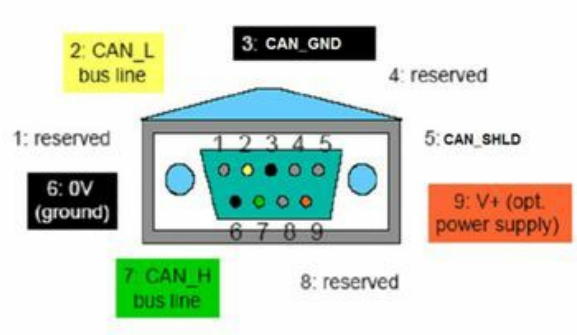
低速容错CAN（Fault Tolerance CAN）

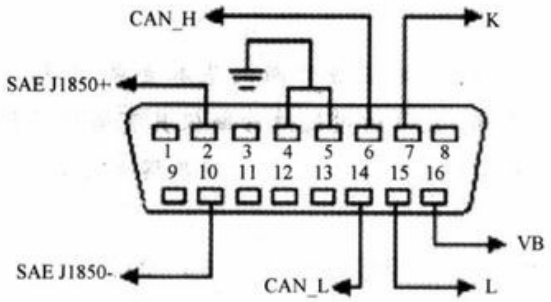


CAN总线遵从“线与”机制：“显性”位可以覆 盖“隐性”位；只有所有节点都发 送“隐性”位， 总线才处于“隐性” 状态。这种“线与”机制使CAN总线呈现显性优先的特性。



CAN总线连接器





下一部分将介绍CAN总线数据链路层，和CAN总线同步机制。

一口气从零读懂CAN总线 下

上一篇文章讲了CAN总线的历史、标准、物理层，现在接着介绍CAN总线数据链路层，和CAN总线同步机制。

CAN数据链路层

在SPI通信中，片选、时钟信号、数据输入及数据输出这四个信号都有单独的信号线。而CAN使用的是两条差分信号线，只能表达一个信号。简洁的物理层决定了CAN必然要配上一套更为复杂的协议。如何用一个信号通道实现同样甚至更强大的功能，答案就是对数据或操作命令进行打包。

通信机制

多主机（Multi-Master）

安全敏感的应用（如汽车动力）对通信系统的可靠性要求很高。将总线能否正常工作归结到单一节点是非常危险的，比较合理的方案是对总线接入的去中心化，即每个节点都有接入总线的能力。这也是CAN总线采用多主控（Multi-Master）线性拓扑结构的原因。

在CAN总线上，每个节点都有往总线上发送消息的能力，而消息的发送不必遵从任何预先设定的时序，通信是事件驱动的。只有当有新的信息传递时，CAN总线才处于忙碌的状态，这使得节点接入总线速度非常快。CAN总线理论最高数据传输速率为1Mbps，对于异步事件反应迅速，基本对于ms级别的实时应用没有任何问题。

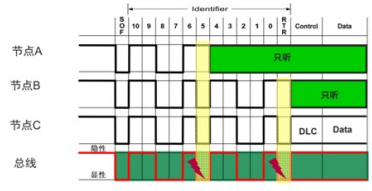
寻址机制

不同于其它类型的总线，CAN总线不设定节点的地址，而是通过消息的标识符（Identifier）来区别消息。这种机制虽然会增加消息的复杂度（增加标识符），但是节点在此情况下可以无需了解其他节点的状况，而相互间独立工作。在总线上增加节点时仅需关注消息类型，而非系统上其他节点的状况。这种以消息标识符寻址的方式，让总线上增加节点变得更加灵活。

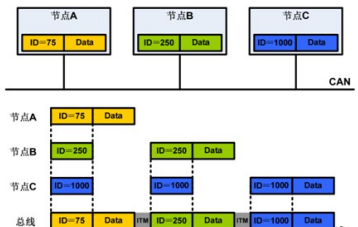
总线访问CSMA/CD+AMP

CAN总线通信原理可简单描述为多路载波侦听+基于消息优先级的冲突检测和非破坏性的仲裁机制（CSMA/CD+AMP）。CSMA（Carrie Sense Multiple Access）指的是所有节点必须都等到总线处于空闲状态时才能往总线上发送消息；CD+AMP（Collision Detection + Arbitration on Message Priority）指的是如果多个节点往总线上发送消息时，具备最高优先级的消息获得总线。

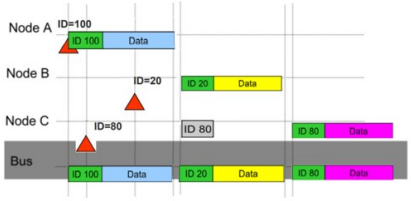
- 多路载波侦听：网络上所有节点以多点接入的方式连接在同一根总线上，且发送数据是广播式的。网络上各个节点在发送数据前都要检测总线上是否有数据传输：若网络上有数据，暂时不发送数据，等待网络空闲时再发；若网络上无数据，立即发送已经准备好的数据。
- 冲突检测：节点在发送数据时，要不停的检测发送的数据，确定是否与其他节点数据发送冲突，如果有冲突，则保证优先级高的报文先发送。
- 非破坏性仲裁机制：通过ID仲裁，ID数值越小，报文优先级越高。



发送低优先级报文的节点退出仲裁后，在下次总线空闲时自动重发报文。



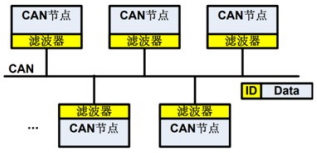
高优先级的报文不能中断低优先级报文的发送。



报文发送请求

报文接收过滤

CAN控制器大多具有根据ID过滤报文的功能，即只接收某些ID的报文。节点对接收到的报文进行过滤：比较消息ID与选择器（Accepter）中和接受过滤相关位是否相同。如果相同，接收；如果不相同，则过滤。



CAN的报文种类及结构

报文的种类

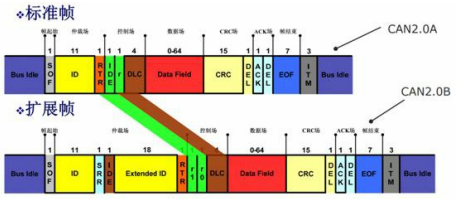
在原始数据段的前面加上传输起始标签、片选（识别）标签、控制标签，在数据的尾段加上CRC校验标签、应答标签和传输结束标签。把这些内容按特定的格式打包好，就可以用一个通道表达各种信号了。各种各样的标签，起到了协同传输的作用。当整个数据包被传输到其他设备时，只要这些设备按格式去解读，就能还原出原始数据。类似这样的数据包就被称为CAN的数据帧。

为了更有效的控制通信，CAN一共规定了5中类型的帧，帧也称为报文。

帧	帧用途
数据帧（Data Frame）	用于发送单元向接收单元传输数据的帧
远程帧（Remote Frame）	用于接收单元向具有相同 ID 的发送单元请求数据的帧
错误帧（Error Frame）	用于当检测出错误时，向其他单元通知错误的帧
超载帧（Overload Frame）	用于接收单元通知其尚未做好接收准备的帧
帧间隔（Inter Frame Space）	用于将数据帧及遥控帧与前面的帧分离开来的帧

数据帧

数据帧在CAN通信中最主要，也最复杂。数据帧以一个显性位（逻辑0）开始，以7个连续的隐性位（逻辑1）结束。CAN总线的数据帧有标准格式（Standard Format）和扩展格式（Extended Format）的区分。



数据帧可以分为七段：

- 帧起始（SOF）

标识一个数据帧的开始，固定一个显性位。

用于同步， 总线空闲期间的任何隐性到显性的跳变都将引起节点进行 硬同步。只有总线在空闲期间节点才能够发送SOF。

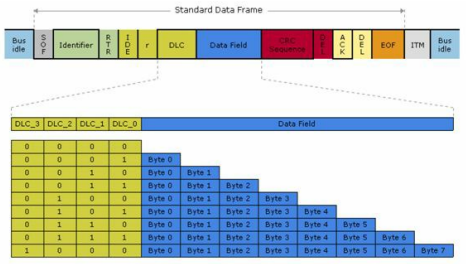
- 仲裁段（Arbitration Field）

仲裁段的内容主要为本数据帧的ID信息。数据帧分为标准格式和扩展格式两种，区别就在于ID信息的长度：标准格式的ID为11位；扩展格式为29位。在CAN协议中，ID决定着数据帧发送的优先级，也决定着其他设备是否会接收这个数据帧。

仲裁段除了报文ID外，还有RTR, IDE, SRR位。

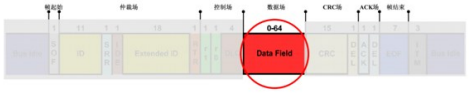
- 控制段

在控制段，r1（reserved1）和r0（reserved0）为保留位，默认设置为显性位。最主要的是DLC（Data Length Code）段，它是用二进制编码表示本报文中的数据段包含多少个字节。DLC段由4位组成，DLC3−DLC0，表示的数字为0-8.



- 数据段

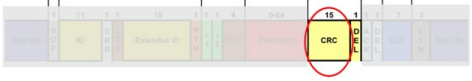
数据帧的核心内容，有0-8个字节长度，由DLC确定。

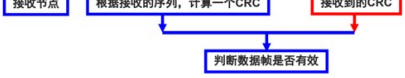


- CRC段

为了保证报文的正确传输，CAN的报文包含了一段15位的CRC校验码，一旦接收端计算出的CRC码跟接收到的CRC码不同，就会向发送端反馈出错信息以及重新发送。CRC部分的计算和出错处理一般由CAN控制器硬件完成，或由软件控制最大重发数。

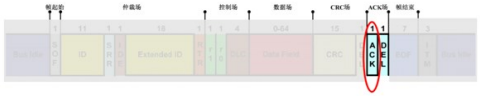
在CRC校验码之后，有一个CRC界定符，它为隐性位，主要作用是把CRC校验码与后面的ACK段隔开。

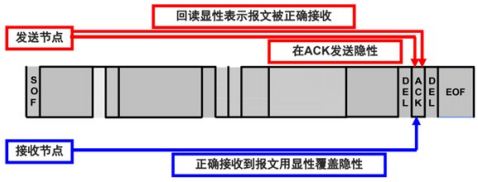




- ACK段

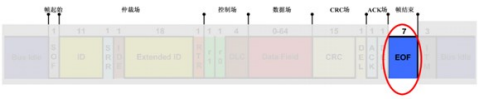
包含确认位（ACK slot）和界定符（Delimiter,
DEL）。ACK在发送节点发送时，为隐性位。当接收节点正确接收到报文时，对其用显性位覆盖。DEL界定符同样为隐性位，用于隔开。





- 帧结束段（End-of-Frame, EOF）

帧结束段由发送端发送7个隐性位表示结束。



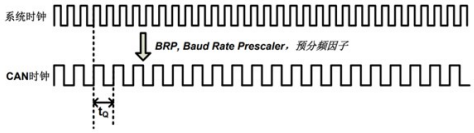
同步

CAN总线使用位同步的方式来确保通信时序，以及对总线的电平进行正确采样。

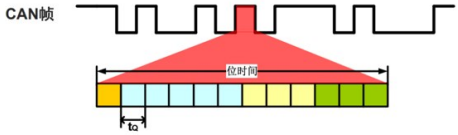
位时序

在讲位时序之前，先介绍几个基本概念。

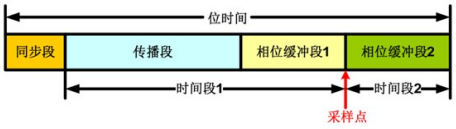
Time Quantum 时间份额tQ ：CAN控制器工作的最小时间单位，通常对系统时钟分频得到。



波特率：单位时间内（1s）传输的数据位，公式：1/位时间。举个栗子，系统时钟频率36MHz，预分频因子为4，则CAN时钟频率9MHz，则Tq=1/9M。假设一个CAN位包含10个Tq，则一个位周期T=10Tq，从而波特率为1/T=0.9MHz.



为了实现位同步，CAN协议把每一位的时序分解成下图所示的四段。这四段的长度加起来即为一个CAN数据位的长度。一个完整的位由8-25个Tq组成。



- 同步端（SS，Synchronization Segment）

一个位的输出从同步段开始。若总线的跳变沿被包含在SS段的范围之内，则表示节点与总线的时序同步。节点与总线同步时，采样点采集到的总线电平即可被确定为该电平的电位。SS段的大小为1Tq.

- 传播段（PTS，Propagation Time Segment）

用于补偿信号在网络和节点传播的物理延时时间，是总线上输入比较器延时和输出驱动器延时总和的两倍。通常1-8Tq

- 相位缓冲段1（PBS1， Phase Buffer Segment 1）

主要用于补偿边沿阶段的误差，其时间长度在重新同步时可以加长。初始大小1-8Tq.

- 相位缓冲段2（PBS2，Phase Buffer Segment 2）

也是用于补偿边沿阶段的误差，其时间长度在重新同步时可以缩短。初始大小2-8Tq.

同步

CAN同步分为硬同步和重新同步。

同步规则：

- 一个位时间内只允许一种同步方式
- 任何一个“隐性”到“显性”的跳变都可用于同步
- 硬同步发生在SOF阶段，所有接收节点调整各自当前位的同步段，使其位于发送的SOF位内。
- 重新同步发生在一个帧的其他阶段，即当跳变沿落在同步段之外。

硬同步

当总线上出现帧起始信号（SOF，即隐性到显性的边沿）时，其他节点的控制器根据总线上的这个下降沿对自己的位时序进行调整，把该下降沿包含到SS段内。这样根据起始帧来进行的同步称为硬同步。

可以看到在总线出现帧起始信号时，该节点原来的位时序与总线时序不同步，因而这个状态的采样点采集到的数据是不正确的。所以节点以硬同步的方式调整，把自己的位时序中的SS段平移至总线出现下降沿的部分，获得同步，这时采样点采集到的数据是正确数据。

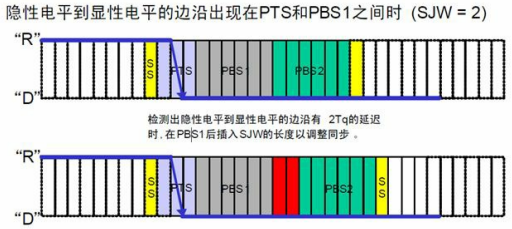
同步前

同步后

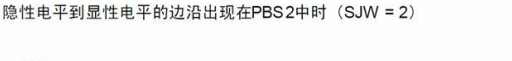
重新同步

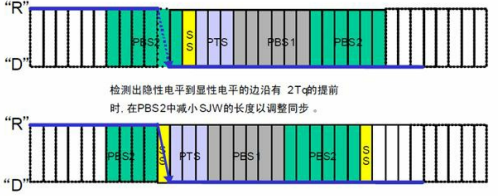
因为硬同步时只是在有帧起始信号时起作用，无法确保后续一连串的位时序都是同步的，所以CAN引入了重新同步的方式。在检测到总线上的时序与节点使用的时序有相位差时（即总线上的跳变沿不在节点时序的SS段范围），通过延长PBS1段或缩短PBS2段来获得同步，这样的方式称为重新同步。

分两种情况：第一种， 节点从总线的边沿跳变中，检测到它的时序比总线的时序相对滞后2个Tq，这是控制器在下一个时序中的PBS1段增加2Tq的时间长度，使得节点与总线时序重新同步。



第二种，节点从总线的边沿跳变中，检测到它的时序相对超前2Tq，这时控制器在前一个位时序中的PBS2段减少2Tq的时间长度，获得同步。





在重新同步的时候，PBS1和PBS2段的允许加长或缩短的时间长度定义为，重新同步补偿宽度（SJW，reSynchronization Jump Width）。这里设置的PBS1和PBS2能够增减的最大时间长度SJW=2Tq，若SJW设置的太小则重新同步的调整速度慢，若太大，则影响传输速率。

分类: 个人总结

posted on 2018-06-21 16:49 AlanTu 阅读(3032) 评论(0) 编辑 收藏 举报

高质量博客链接

最新评论

- 1. Re:c++拷贝构造函数详解
- 如果定义了拷贝构造函数，那也必须重载赋值操作符.这句话怎么理解？上面的例子没有重载赋值运算符呀
- --出手无招
- 2. Re:linux select函数详解
- select 必须搭配轮询吗?
- --明明1109
- 3. Re:Linux深入理解Socket异常
- 写的真好
- --喜欢兰花山丘
- 4. Re:Ext4文件系统架构分析(一)
- @miaoqingcoding 换个浏览器， Firefox可以看到图， chrome看不到...
- --Tudou_Blog
- 5. Re:Ext4文件系统架构分析(一)
- 大佬你这怎么没图啊
- --miaoqingcoding
- 6. Re:Linux下逻辑地址、线性地址、物理地址详细总结
- 第一小节讲的很棒
- --小北觅
- 7. Re:AOE网与关键路径简介
- 文章中有一处错误。应该是你截图的书的勘误。 “可以发现，在计算ltv时，其实是把拓扑序列倒过来进行而已，因此可以得到计算顶点vk最晚发生时间即求ltv[k] 的公式是：” 这句话下面的图中，公式里的第...
- --lulipro
- 8. Re:深入理解Linux内存分配
- 两脸懵逼
- --不知也
- 9. Re:prim算法
- 图清晰
- --pubby
- 10. Re:c++拷贝构造函数详解
- 讲的正好
- --杀马特彪少
- 11. Re:c++拷贝构造函数详解
- 最后一句：“如果定义了拷贝构造函数，那也必须重载赋值操作符。”是指赋值操作符也要自己重新定义？
- --noone3001
- 12. Re:二维码的生成细节和原理
- 大佬，数据和数据纠错码填充是从右下角开始。
- --amisarex
- 13. Re:利用recv和readn函数实现readline函数
- 学习了，但是有一个问题：recv_peek 之后，如果找到了 \r\n，就需要清空缓存区。但是清空又得重复 Copy 缓存区，有没有只清空指定长度的缓存区，不重新 Copy 的方式？
- --陈鑫伟
- 14. Re:【转】对 Rust 语言的分析
- 看到lifetime，我就感觉这门语言有点复杂
- --sabiameno
- 15. Re:c++拷贝构造函数详解
- @jarven_0728编译器优化的问题，你用2019版的就应该没问题...
- --流霜抚墨
- 16. Re:c++拷贝构造函数详解
- 这个界面是怎么回事.....
- --NoneType
- 17. Re:c++拷贝构造函数详解
- 顶顶顶!
- --小小程序梦
- 18. Re:Linux内存初始化(一)
- “假设我们分配4个page分别保存Level 0到level 3的translation table，那么可以建立的最大的地址映射范围是512（level 3中有512个entry）X 4k = 2...
- --飘零剑客
- 19. Re:c++拷贝构造函数详解
- 菜鸡在线感激
- --无趣趣无
- 20. Re:二维码的生成细节和原理
- 您好，您的pdf连接好像打不开了，能发一份给我吗？百度网盘链接也可以，谢谢。
- --ruxia_TJY
- 21. Re:【转】对 Rust 语言的分析
- （），在Rust就是元组，
- --m12
- 22. Re:P问题、NP问题和NPC问题
- 请问，一个图中是否不存在Hamilton回路这样的问题是不是就属于NP-Hard问题了？
- --破碎中的麦粒
- 23. Re:prim算法
- 很棒!
- --菜鸟qiz
- 24. Re:Linux文件系统详解
- 博主内容很好，希望排版能改善
- --瘪丁
- 25. Re:C语言二维数组作为函数的参数
- mark！
- --shiyideliutang
- 26. Re:深入理解Linux内存分配
- 一脸懵逼

- --折花刀
- 27. Re:Linux文件系统详解
- 博主您好，您的博文您自己看了阅读版面效果吗。 能稍微改进一下，对阅读友好点吗。
- --Tudou_Blog
- 28. Re:c++拷贝构造函数详解
- 顶
- --你的雷哥
- 29. Re:linux下判断文件和目录是否存在
- 使用opendir函数时需要包含头文件<dirent.h>
- --红城客栈蓝精灵
- 30. Re:windows下LIB和DLL的区别与使用
- 谢谢 理解了
- --EasyWorld
- 31. Re:深入理解Linux内存分配
- MARK
- --小星星的大猩猩
- 32. Re:malloc的内存分配原理
- 这个图片都挂了
- --WOTGL
- 33. Re:贪心算法 - 0/1背包问题
- @ 笑里藏道对，这是动态规划算法，不是贪心算法...
- --FyuNaru
- 34. Re:c++拷贝构造函数详解
- 大赞博主
- --ssif
- 35. Re:Linux内存初始化(一)
- --xiashaohua2010
- 36. Re:贪心算法 - 0/1背包问题
- 这个不是贪心，是记忆化搜索（或者备忘录，别名很多）吧，跟动态规划类似，你的证明过程其实也没有证明贪心选择性质
- --笑里藏道
- 37. Re:理解矩阵乘法
- 优秀
- --Happy_lei
- 38. Re:AOV网与拓扑排序
- 写得非常清晰易懂！赞一个~
- --坚持的彼岸是星辰大海
- 39. Re:c++拷贝构造函数详解
- 函数的返回值是类的对象，这个时候，为什么我运行的结果没有调用赋值构造函数？
- --jarven_0728
- 40. Re:linux内存碎片防治技术
- 你好，请教一个问题就是fallbacks这个数组的构建，这里定义的是一个5*4的数组，但是只初始化了部分元素，没有初始化的元素变成了0，而0刚好对应的是UNMOVABLE，也就是说所有的迁移类型最后都...
- --DoOrDie

阅读排行榜

评论排行榜

推荐排行榜