东风 8B 内燃机车 CAN 总线通信网络的实现

佴晓东

北方交通大学 人机工程实验室

摘要:

随着工控系统的发展机电系统也变得越来越复杂，因此其对通信系统的要求也越来越高。东风 8B 内燃机车就是一例。传统的 RS232/RS485 总线因为只能实现单主通信，使用一般通信速度双绞线时其通信速度只有 19.2k 位/秒且数据通信可靠性不高，无法适应东风 8B 内燃机车实时控制的需要。为满足新型内燃机车控制系统的通信需要，我们用控制器局域网（CAN）对其进行了改造。

控制器局域网是一种为实时控制而设计的串行通信总线。它可以以高达 1M 位/秒的波特率进行通信，且有很好的纠错能力。

CAN 最初是由德国的 Robert Bosch 公司为在汽车电器领域替代传统的昂贵且笨重的布线系统而开发的一种低成本通信总线。因为 CAN 的可靠性和鲁棒性，其不仅汽车工业领域应用广泛而且其他许多工业控制领域也采用 CAN 作为通信系统解决方案。

CAN 作为一种国际标准已经被收入了 ISO11898 [1]（高速通信应用）和 ISO11519（低速通信应用）。

CAN 是一种具有多主通信能力的串行通信总线系统。每个 CAN 节点都可以发送数据，多个 CAN 节点可以同时提出通信请求。ISO11898 国际标准的目标就是提供具有实时能力的串行总线系统，而 CAN 则包含了 ISO/OSI 参考模型的最低两层。在 CAN 网络中没有传统意义上的接受器或节点地址，取而代之的是具有不同优先级的信息被发送。一个发送节点向所有 CAN 节点发送信息（广播方式）。每个节点依据所接收信息的标识符来决定是否处理该信息。标识符同时也决定了此信息竞争网络存取的优先权。

CAN 协议的突出优点之一是他提供了很高的传输可靠性。CAN 控制器记录节点的错误并依据统计数字采取相应措施。因此产生错误的 CAN 节点若继续出错将会脱离 CAN 网络。

CAN 的最大传输速度定义为 1M 位/秒，在此速度下网络长度可达 40 米。在 125k 位/秒时为 500 米，在 50k 位/秒时可达 1 千米。

经过数月的实际运行足以证明东风 8B 内燃机车所采用的 CAN 网络具有高速、多主及鲁棒性强等优点，是一种成功的通信系统。

关键词：

控制器局域网，局域网，工业现场总线，串行通信，单片机，机车

第一章 绪论

1.1 DF8B 内燃机车通信系统

DF8B是为低速货运列车而设计的内燃机车，已经使用了很长时间。此内燃机车的通信系统中只有2个节点：主控计算机及显示屏。在物理层此通信系统使用的是RS232-RS485数据总线。因为RS232-RS485都是单主串行总线，因此本通信系统中只有一个主节点：主控计算机；系统中的显示屏是从节点。本通信系统框图如图1-1所示。

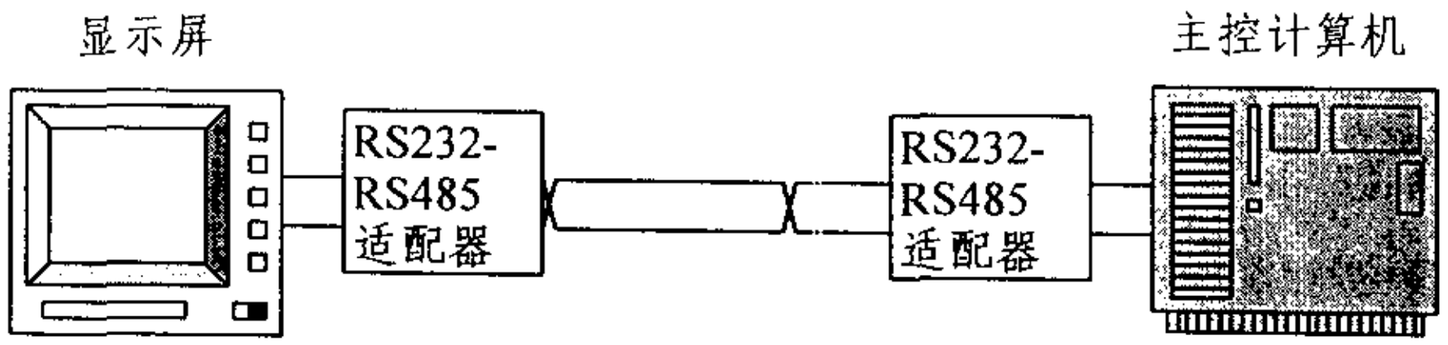


图 1-1 DF8B 内燃机车 RS232-RS485 通信系统框图

当开机上电后，显示屏被指定一个固定的网络地址。主控计算机就使用这个地址对显示屏进行读写控制。当主控计算机希望向显示屏发送数据时它将设置地址信息并发送数据，根据此地址信息显示屏就可得到数据。当主控计算机希望从显示屏接收数据时它首先发送地址信息和命令字，当显示屏接收到此命令后便发送响应信息。此过程称之为‘握手’。此后显示屏将向主控计算机发送数据。

此通信系统的主要缺陷为：

- 网络中只有一个节点可以作为主节点。
- 任何两个从节点不可直接相互发送数据。
- 当通信速度（波特率）较高时总线长度较短。
- 通信速度只有4800位/秒（bps）.
- 因此网络只定义了物理层（RS232-RS485）所以只能在字节宽度上检测通信错误而不能在帧宽度上检测通信错误（纠错能力差）.

1.2 通信系统网络软硬件改造的目标

本项目选择CAN（Controller Area Network）总线做为通信手段以改造此通信系统，其主要目标为：

- 长距离通信（大于400米）。
- 高通信速度（可达1000k bps）。
- 高数据可靠性。
- 多主网络体系结构。

在下面的各章中将分别介绍这些特点。

第二章 控制器局域网（CAN）简介

2.1 什么是 CAN（Controller Area Network）

控制器局域网(CAN)是一种串行通信总线，它特别适合于在一个局域系统或子系统中连接传感器、驱动器等智能设备。

CAN是一种具有多主通信能力的串行总线，即所有的CAN节点都可发送数据并且几个CAN节点可以同时向总线发出发送请求。这些特点正是ISO 11898 [1]标准所规定的具有实时通信能力串行总线的要求，而且CAN也包含了国际标准化组织/开放系统互联（ISO/OSI）的最低两层协议。在CAN网络中没有传统意义上的接收器或节点的地址，取而代之的是采用发送不同优先级的信息。一个发送节点向所有的CAN节点同时发送信息（广播方式）。每个节点依据信息的标识符来判断是否接受并处理此信息。信息标识符同时也决定了此信息在总线控制权竞争中的优先级。

欧洲客运轿车中决大多数都安装了CAN，货运及非公路运输车辆的生产商也大量采用CAN。CAN芯片已经大批量生产超过10年。在家用电器及工业控制领域CAN也越来越多的被采用。截止1999年春季全世界已安装了超过1亿5千万个CAN节点[(CIA) CAN In Automation]。

CAN协议最突出的特点之一是其具有很高的数据传输可靠性。CAN控制器记录一个节点的错误，统计并评估将采取何种相应对策。如此节点不断产生错误它将被与网络自动脱离。

CAN的最大传输速度被定义为1M bit/s。在此通信速度下网络长度可达40米。如加长网络长度通信速度必须降低：在500米长度可达125k bit/s，在1千米时可达50k bit/s。

2.2 CAN 的工作原理

2.2.1 数据通信规则

当数据在 CAN 网络上发送时没有节点被定以地址，相应的数据的内容（如发动机温度、转速等）被定义以此网络上唯一的标识符。此标识符不仅定义了此数据的内容，还定义了其网络优先级。此特性在几个节点同时竞争发送数据时分配网络控制权时非常重要。

当一个节点的 CPU 希望向网络中的一个或多个节点发送信息时它将把要发送的信息及标识符传给指定的 CAN 芯片（使准备就绪）。所有的 CPU 在发送数据时都必须做这一步——初始化。CAN 芯片将把此信息转化为 CAN 信息并发送（发送信息）。一旦此 CAN 节点得到了总线控制权 CAN 网络上其它所有节点便变为此信息的接收节点（接收信息）当 CAN 网络上每个节点正确地接收到此信息，他们便分别测试此信息是否与自己相关并决定接收与否（选择）。如果接收到的数据被认为相关它将被处理（接受），否则便被忽略。此发送、接收过程可用图 2-1 表示。

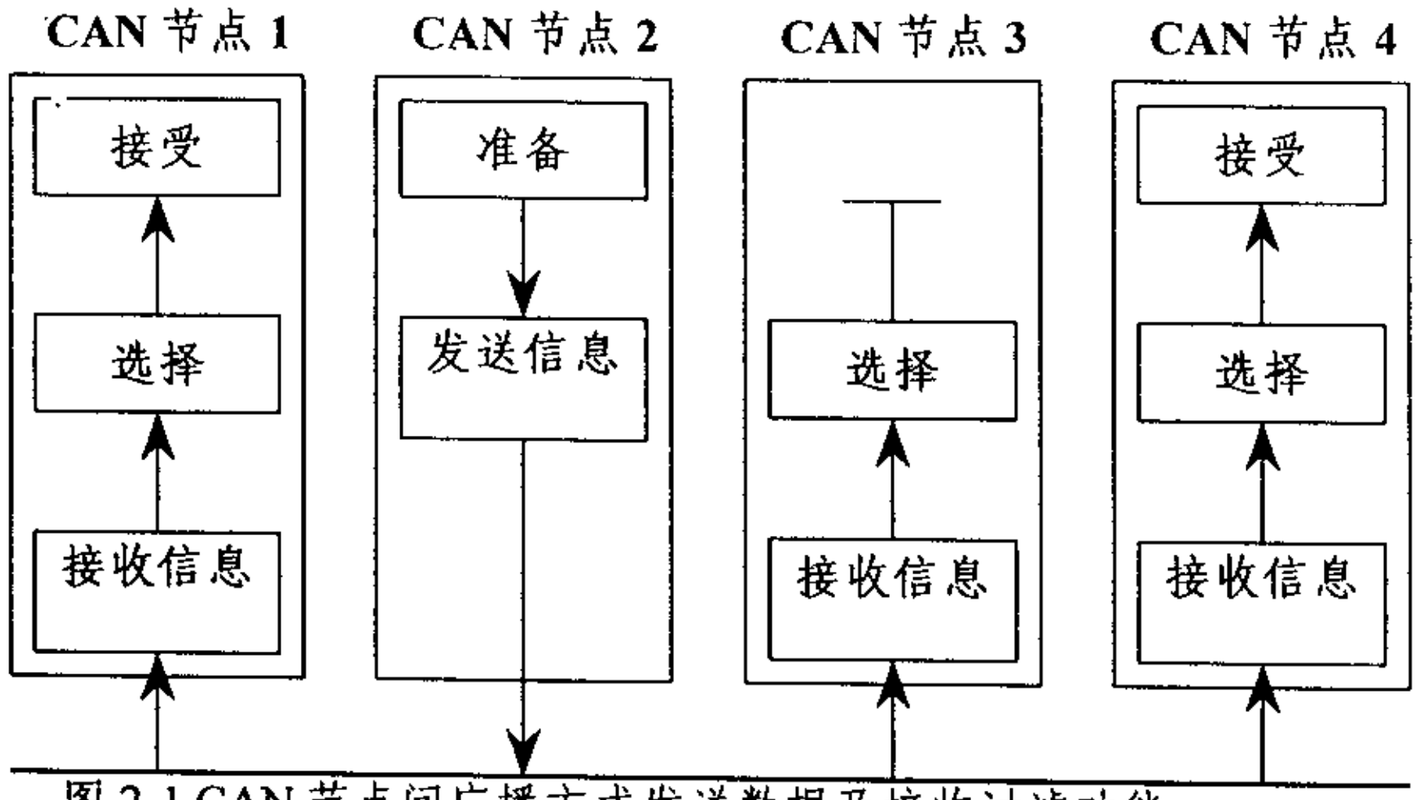


图 2-1 CAN 节点间广播方式发送数据及接收过滤功能。

CAN 网络及系统配置的高度灵活性是通过面向内容的寻址方式实现的。在已有的网络中若想加入新节点非常方便，不需要对已有节点的软硬件进行任何更新，只需将新节点设置为接收节点即可。因为 CAN 数据传输协议本身并不需要对各个节点在物理上定义定义其地址。CAN 支持模块式结构并允许多重接收（广播方式，多重发送方式）及分布式同步处理。例如多个控制器为进行某项测量所需要的数据可通过一个或多个传感器通过网络进行传输，这样就不需要为每个控制器配置各自的传感器。

2.2.2 非破坏性位宽度访问仲裁机制

数据进行实时处理的必须条件是它们能够在网络上快速的传输。这不仅意味着数据的物理传输速度应高达 1M bit/s，而且当几个节点希望同时发送数据时进行快速的总线分配。

在实时控制中通过网络所要发送数据信息的紧急程度可能差异很大：有些快速变化的变量（如柴油机载荷）必须被频繁的发送，而

其它变量（如柴油机温度）变化相对较慢，不须被频繁发送，也允许有较大的时间延迟。

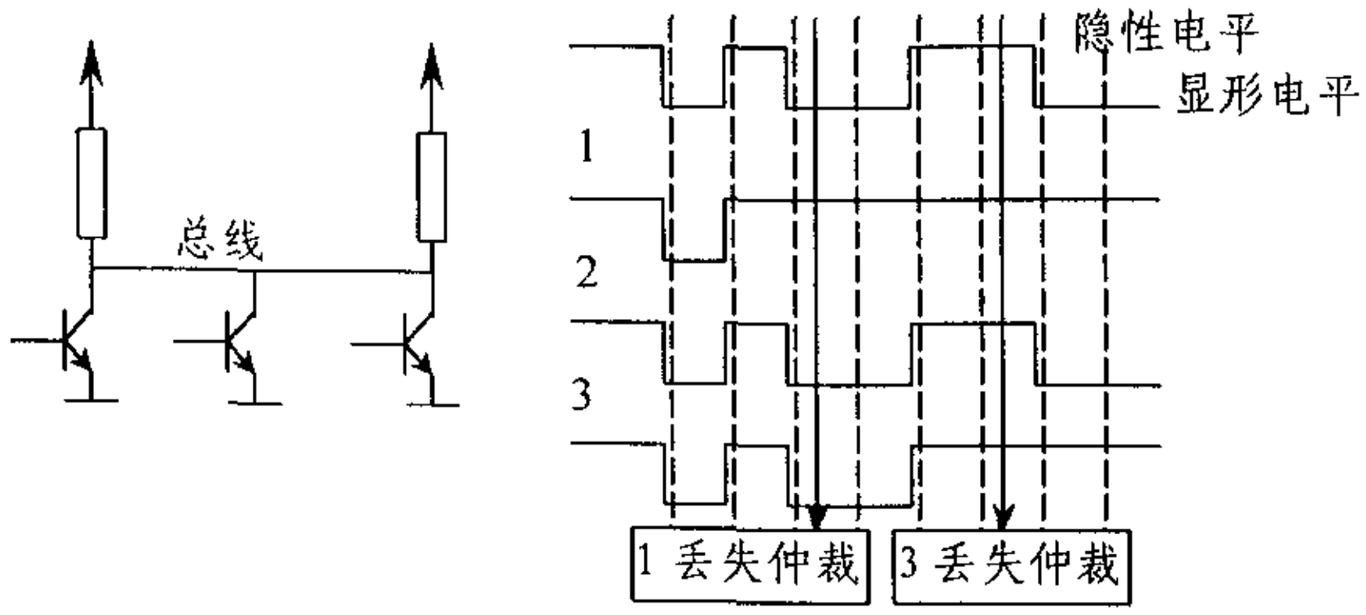


图 2-2 非破坏性位宽度仲裁机制

某需要紧急发送信息和其它信息相比所具有的相对优先级是由此信息的标识符所定义的。在进行系统设计时不同信息的优先级就被规定事先好并且不可以进行动态地改变。每个信息被分配一个二进制的数值（标识符），数值越小优先级越高。

对于总线控制权的竞争是通过位宽度的仲裁机制来解决的。每个希望取得总线控制权的节点依据其所发送信息的标识符一位一位的进行仲裁．根据线与机制显形电平（逻辑 0）将覆盖隐性电平（逻辑 1）．所有参与总线竞争的节点当发送隐性电平而接收到显形电平时将丢失仲裁．所有丢失仲裁的节点将自动变为此信息的接收节点，直到此信息发送结束总线空闲之前将不再竞争总线控制权。图 2-2 显示了此种竞争机制．

2.2.3 总线分配效率

总线分配效率主要是由此串行总线的应用系统所决定的。不同的应用系统适合于不同的总线。为简单起见，本文将总线分配过程用以下方法分为不同类别:

- 固定时间总线分配机制

  此机制将总线的控制权顺序地分配给每个参与总线的节点。不论此节点是否需要它都将得到在某个时间段内控制总线的权利．例如令牌跟踪（token slot）或令牌传递（token passing）机制。

- 依据需求分配总线

总线分配是基于当前最迫切的传输请求分配给某个参与总线分配的节点。例如总线只分配给希望传输数据的节点（例：CSMA，CSMA/CD，巡回分配（round robin）及位宽度仲裁）

对 CAN 总线的分配纯粹是基于等待发送的信息间的不同优先级而定。这就是说 CAN 可以归类为依据需求分配总线。

另一种判断仲裁系统效率的方法是判断总线访问的方式。可以分为两类：

- 非破坏性总线访问

  此种方法是将总线立刻或在一个总线访问周期中分配给某个节点（而且只分配给此节点）去访问一个或多个节点。这样就可以确保总线被一个节点所访问，不会产生总线分配的歧义性。例如令牌跟踪（token slot），令牌传递（token slot），巡回分配（round robin）及位宽度仲裁。

- 破坏性总线访问

  多个节点对总线的同时访问会造成所有的传输终止，因此没有一个节点能成功地分配到总线。对为取得总线的访问控制权也许要进行多次总线访问。例如 CSMA/CD，以太网（Ethernet）。

在 CAN 网络中对所有传输请求的处理必须遵从延迟时间的限制，即处理时间不得长于在一定的传输速度下的总线定时时间。CAN 协议必须实现在多个节点同时对总线提出访问时保证只有一个节点获得总线控制权（非歧义性）。因为 CAN 网络中每个信息所具有的标识符是唯一的，在位宽度总线访问仲裁机制中依据 11 位标识符（标准帧）或 29 位标识符（扩展帧）对总线进行分配不会产生歧义。与某些基于帧宽度（信息宽度）的总线访问控制仲裁（如 CSMA/CD）不同，CAN 的此种仲裁方式可以保证总线的所有容量都被用以传输有效的数据。

在 CAN 总线中对总线的访问权是和信息的标识符相联系的，与其他的仲裁方法相比此种方法也有其优越性。因为即使在总线超载的情况下，整个总线中最急需的信息（依优先级而定）也总是被优先发送。这样尽管总线容量不足，也能使系统的性能不致受到太大的影响。在此种情况下与其他系统相比此种方法总线的分配效率是最高的，而且每次总线分配之间的空隙也被压缩到最小程度。在其他系统中因总线超载所造成的整个系统立刻崩溃在 CAN 却中不会发生。因为采用了位宽度的非破坏性总线访问仲裁机制，我们可以依据信息的急需程度快速、高效地发送数据。

非破坏性总线访问也可以进一步分成两种:

- 集中式总线访问控制（centralized bus access control）
- 分散式总线访问控制（decentralized bus access control）

若整个总线中只有一个节点对总线的访问可以进行控制则称之为集中式总线访问控制，否则称之为分散式总线访问控制。如果网络采用集中式总线访问控制，当此节点（主节点）发生错误或失效时必须有某种机制以确保总线能继续运行。然而这种方法有其缺陷性，因为这种出错处理机制一般很难实现，而且其软硬件开销也很大。而且由一个节点进行进行网络访问控制，许多节点间进行相互通信时时间消耗非常大。基于以上原因 CAN 采用分散式总线访问控制，这样因只有一个主节点所造成的诸多困难变迎刃而解了。在 CAN 网络中总线访问控制、出错处理等机制都是由不同节点在多个时间分别完成的，这样可以实现通信系统很高的可利用性。

总之 CAN 实现了一种基于通信信息优先级的流量控制方法，即分散控制的非破坏性位宽度总线访问机制。在此种机制下用较低的通信速度就可以实现较高的有效数据传输量，而且对所有节点来说总线繁忙时间也大大缩短。对于那些有非常急需信息要发送的节点来说总线的通信效率大大提高。

对于实时系统来说这些特性是相当关键的，因为在实时系统中当发生总线超载时最高优先级的信息仍然可以被发送，这样硬实时系统不会因为超载而发生系统崩溃，软实时系统的性能不会因为超载而有很大降低。

2.2.4 信息帧格式

CAN 协议支持两种信息帧格式，其差别仅在于信息标识的长度有所不同。在标准帧中标识符长度为 11 位，在扩展帧中为 29 位。每个信息帧都包含七个数据场．图 2-3 所示为标准帧格式。

标准帧是由启始位（SOF）开始的．在启始位后面是仲裁场（arbitration field），它包括标识符（identifier）及远程请求位（remote transmission request）所组成，此位标志着一帧信息是数据帧还是远程帧．接下来是控制场（control field），它包括标识符扩展位（IDE）（标志此信息是否是扩展帧），一位保留位（R0）既数据长度域（DLG）（标志数据位的长度）。在控制场后面是数据场（data field）（包含 0 至 8 位数据），然后是校验场（CRC field）。校验场用来对一帧数据的可靠性进行检验。在此之后是应答场（ACK field），它包含一个应答间隙及一个分隔符（隐性位）．应答位被发送为隐性位，当所有接收节点正确接到数据时便发送一位显性位将其覆盖（肯定响应）．在一帧的最后是帧结束（end of frame）。

在两帧信息间是一个间歇空间（帧间空间）（intermission）。如果此时没有节点要求对总线进行访问则进入总线空闲状态（bus idle）。有关CAN信息帧格式更详细的信息请参阅第四章。

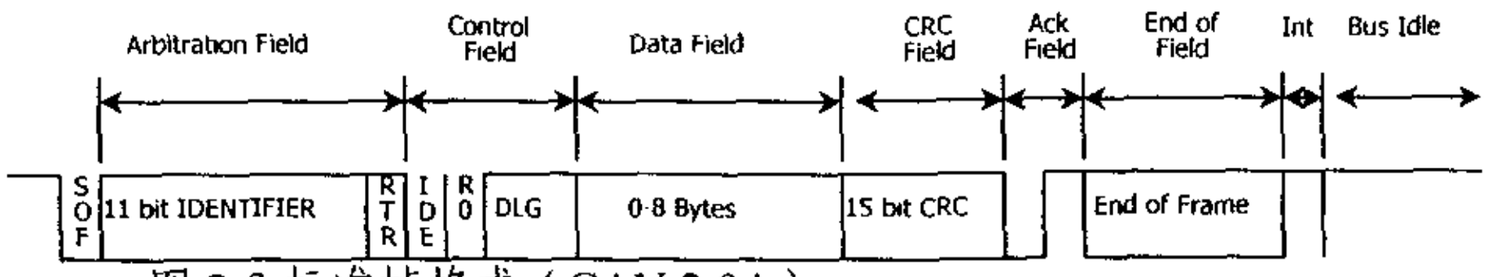


图 2-3 标准帧格式（CAN 2.0A）

2.2.5 出错检测机制

与其他总线系统不同 CAN 不是用响应信息；而是用信号来通告错误的发生。为进行出错检测 CAN 在信息帧的层次上实现了三种检测机制：

- 循环冗余校验（cyclic redundancy check—CRC）

  发送节点在传输一帧信息数据之后便发送循环冗余校验场，它负责对一帧数据进行校验。接收节点对这些校验位进行检验，如果不符合校验结果则产生CRC错误。

- 帧校验（frame check）

  此种机制对一帧信息的结构进行检查，即逐位检查所发送的数据．如果发现检查结果不符合 CAN 的格式则会产生格式错误。

- 响应错误（ACK errors）

  正如 2.2.4 节所述，当一帧数据被正确接收后所有的节点通过肯定响应对其进行应答．如果发送节点没有接收到对发送信息的响应则产生响应错误。这可能意味着接收节点检测出响应信息出错或没有接收节点。

在位层次上CAN也实现了两种出错检验机制：

- 监听机制（monitoring）

  发送节点对错误的检测是通过监听总线信号来实现的。即每个发送节点在发送数据时将同时接收数据并且检测所发送的数据与接收的数据是否一致。这样系统的整体错误就可以被检测到并且错误将被限制在发送节点内部。

- 位填充（bit stuffing）

  每一位数据的编码将在位层次上进行检测。CAN 所实现的是非归零码（NRZ—non return to zero）。此种方法使得在

位层次上的传输效率非常高。为实现信息的同步将进行位填充，即在连续的五位相同电平信号之后将在位数据流中填充一位相反位。在接收节点此位将被删除而只对其余为进行检验。

如果至少一个节点（可以是总线中的任何节点）应用以上机制发现了上述的任何错误，它将发送错误标志（error flag），因此当前的传输将被取消。这样其他节点就不会再接收此信息，所以总线上的数据可以保证其一致性。

如果某信息帧因出错而被取消，发送节点将自动重新发送此信息，这种机制称为自动重发请求（automatic repeat request）。因此它将重新参与总线竞争。按 CAN 协议规定出错信息将在错误被检测出以后的 23 位周期中进行重发；在某些情况下系统将在 31 位的周期中可以得到自动恢复。

尽管 CAN 实现了以上所述的诸多出错检验机制，当系统中某个节点发生严重错误时所有的信息（包括正确信息，出错信息等）都被阻塞。在这种情况下如果没有适当措施进行自我监听整个系统将瘫痪。因此 CAN 协议提供了一种区分偶然错误及永久错误的机制以使错误被限制在某个节点内部，即错误限制（fault confinement）机制。

此种机制对节点内部的错误进行统计，如果节点频繁出错则自动进入一种操作模式，因此其他节点将不会被此节点的错误所波及。如果此节点继续出错，最终它将使自己自动关闭，以保证系统的其它传输正常进行。

2.2.6 CAN 协议的数据可靠性

在汽车工业，运输行业等领域有些应用是与安全紧密相关的，因此需要有很高的数据传输可靠性。此类应用的目标是在其整个使用寿命中确保不会因为数据传输出错而使车辆发生故障，对驾驶员产生生命危险。

如果数据的可靠性非常高，或残余错误的可能性非常底，这个目标就可以实现。在总线系统中数据的可靠性可以用系统对正常数据被错误传输造成错误的识别能力来衡量。残余错误的可能性（残余错误概率）是对数据可靠性的严重损害，它被定义为应被抛弃的出错数据为被检测出而仍然存在的可能性。此概率应该很低以使得在整个产品的寿命周期中残余错误几乎不会发生。这样就可确保产品的安全可靠性。

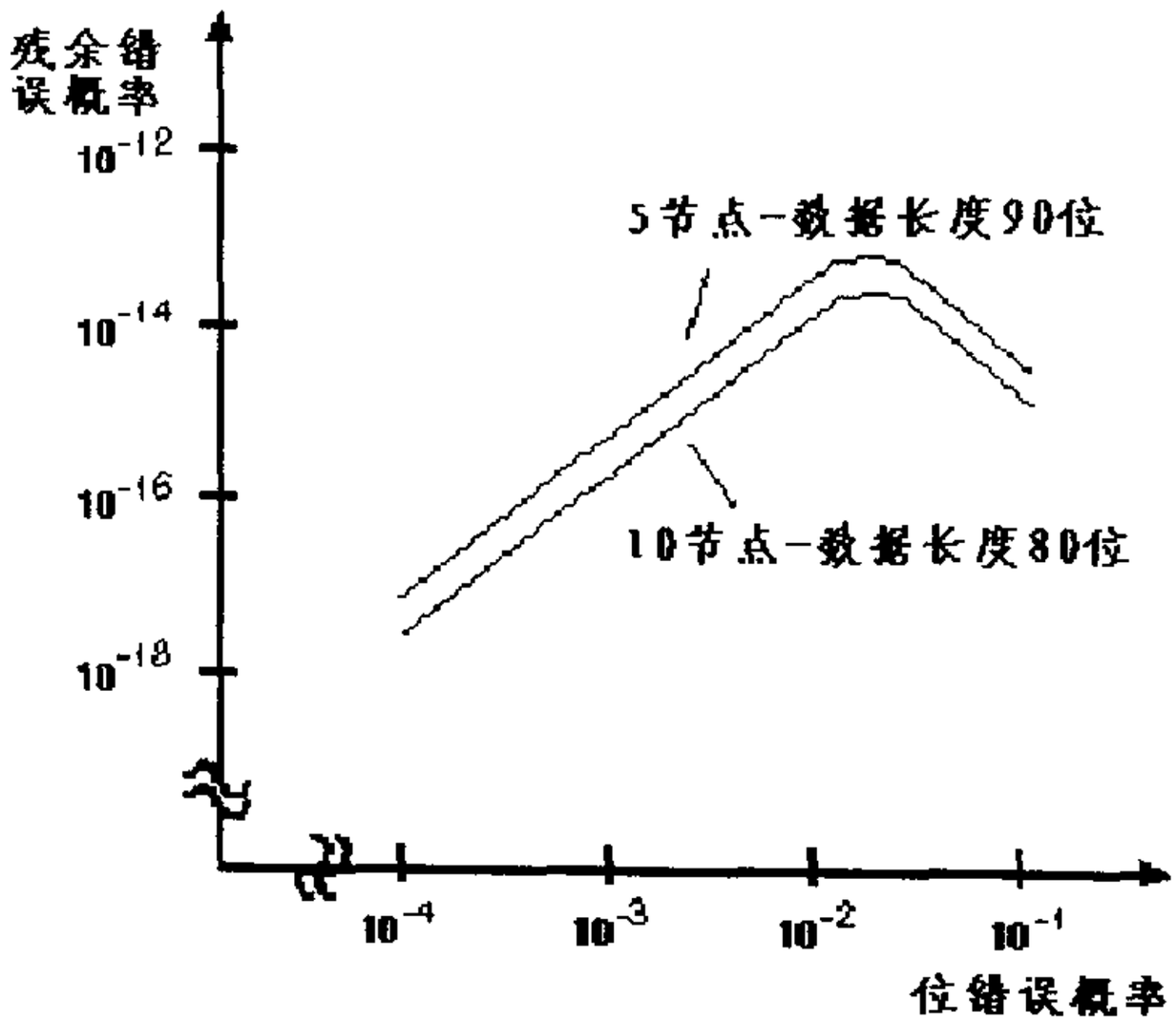


图 2-4 残余错误概率

我们可以将数据传输的整个通路视为一个模块，将所发生的错误进行分类，以计算残余错误概率．假设一帧数据的长度为 80 到 90 位，整个系统由 5 到 10 个节点所组成，其错误概率为 1/1000（每 1000 帧信息出错 1 次）．若位错误概率为 0.02，其平均残余错误概率为 10^{-13}。基于此种假设我们可以计算 CAN 网络的残余错误概率。

当 CAN 总线的波特率为 1M bit/s 时，平均总线利用率约为百分之五十．若其使用寿命总计为 4000 小时，则在其使用期限内总计发送数据帧为 $9x10^{10}$。因此在其使用寿命内未检测出错误的数目（残余错误概率）小于 10^{-2}．如果按每天工作 8 小时，每年 365 天计算，其错误率约 1000 年出错 1 次。图 2-4 为不同情况下残余错误概率[21]。

2.2.7 CAN 扩展帧格式

11 位的标识符对于绝大多数应用是足够的，它可以在一个 CAN 网络定义 2048 种不同的信息帧．但根据 SAE 卡车及公共汽车委员会信号及信息标准，为适应不同的波特率，采用更多的标识符位显然对于系统实现更为方便。

为适应这些需求，CAN 协议进行了扩展，增加为 29 为标识符。这 29 位标识符由已有的 11 位标识符（基本标识符）及 18 位扩展（扩展标识符）所组成。因此 CAN 协议允许使用两种信息格式：标准帧（CAN 2.0A）及扩展帧（CAN 2.0B）．此两种格式的信息帧可以在同

一总线上共存。当一个标准帧和一个扩展帧有相同的低 11 位标识符并同时竞争总线访问权时标准帧优先级较高，将会覆盖扩展帧而取得总线访问权。

如果一个 CAN 控制器支持扩展帧格式，它也能用标准帧进行数据收发。但如果一个 CAN 网络上的某些控制器只支持标准帧则在此网络上只能发送或接收标准帧格式的信息。近些年有的公司提供的产品虽然只支持标准帧格式，但可以识别扩展帧格式并对其进行忽略，这称为被动扩展型（Version 2.0B passive）。

区分标准帧和扩展帧的方法是比较标识符扩展位（IDE—identifier extension bit）。在标准帧中 IDE 位是显形位，而在扩展帧中此位是隐性位。

标准帧中远程请求位（RTR）可以是显形或隐性，用以表示此帧信息是否是远程请求帧。在扩展帧中此位被替代为替代远程请求位（SRR—substitute remote request），此位总是隐性位以保证当其和标准帧竞争总线访问权时标准帧具有更高的优先权而取得总线的访问控制权。

在 IDE 位之后扩展帧与标准帧不同，扩展帧中有 18 位的标识符扩展，之后是 RTR 位。

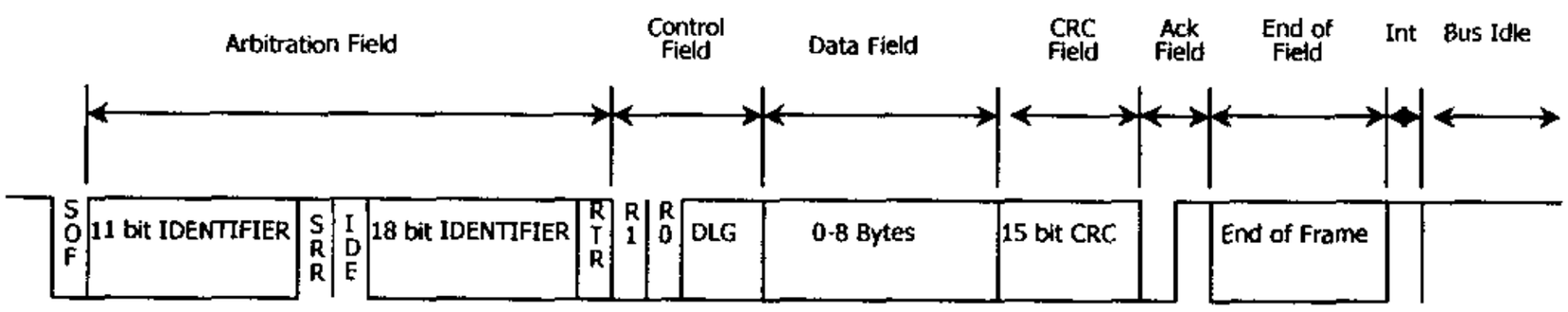


图 2-5 扩展帧格式（CAN 2.0B）

在其他数据场中标准帧和扩展帧完全相同，这样支持扩展帧格式的控制器就可以和支持标准帧格式的控制器相兼容而相互通信。

2.3 CAN 的物理连接

CAN 支持波特率高达 1M bit/s，因此需要有非常陡的边沿信号。为此许多厂家在生产 CAN 控制器同时还提供 CAN 收发器。CAN 收发器是一种功率器件，专门用来连接 CAN 总线和 CAN 控制器。图 2-6 一个标准的 CAN 总线应用系统框图。在第三章中其结构将会被详细介绍。

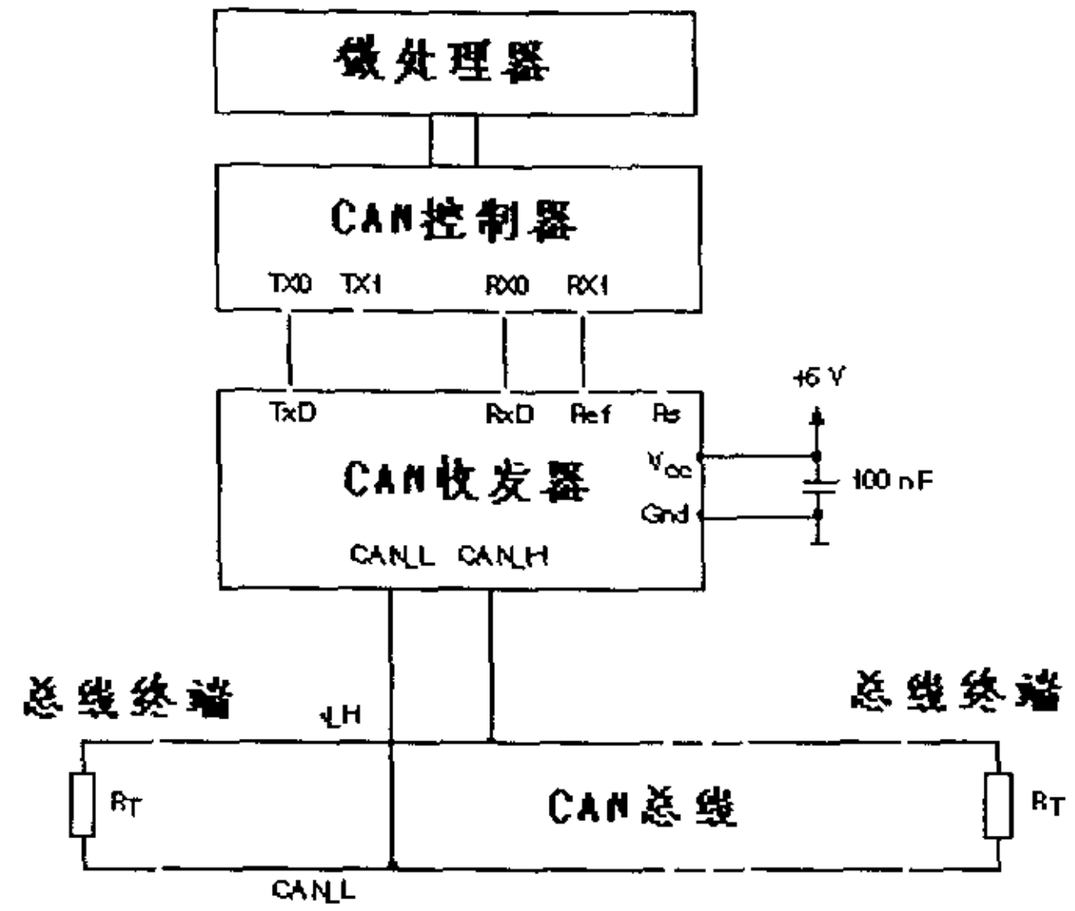


图 2-6 ISO 11898 CAN 物理连接

第三章 CAN网络硬件体系结构

3.1 概述

本章主要介绍依据ISO 11898标准所实现的CAN网络硬件体系结构. 根据ISO 11898标准CAN节点应包括一个模块控制器（主控制器）、一个CAN控制器（协议控制器）和一个CAN收发器。在总线物理连接层本项目使用的是双线差动总线（遵守ISO 11898标准），可以支持波特率为1M bit/s。

3.2 本项目的主要目标

正如本文第一章所述东风8B内燃机车中原有的集中式网络访问控制系统已经成为实现分布式计算机控制的严重障碍。如果能用分布式网络访问控制系统替代原有系统，计算机控制系统的设计实现将会更灵活，方便. 但是如果用全新的网络系统完全取代原有系统将会十分困难，因为原有系统中的控制软件，通信软件等十分庞大，不可能重新编写。因此可行的方法是对原有网络系统加以改造，只做最小的调整，并且兼容原有的与硬件独立的软件。

本项目的主要目标为:

- 实现分散式访问控制局域网通信系统
- 只改造网络硬件及与硬件相关软件系统
- 兼容原有与硬件独立的控制、通信软件系统

因此本项目最重要的任务是实现智能型网卡，由它来实现协议间的转换、出错检测及错误限制、任务调度等功能。智能型网卡的主要任务有以下几点:

- 初始化硬件系统，建立通信连接
- 在CAN协议与原有协议间进行数据翻译、转换功能
- 检测协议错误并采取相应的措施
- 调度任务
- 对发送及接收数据进行缓存

3.3 网络布线拓扑结构及特点

3.3.1 网络特点概要

为适应东风 8B 内燃机车通信系统的要求本网络由 2 个局域网所组成（CAN 总线）。在每个局域网中任何节点都可以和其他节点直接进行数据的收发。此网络主要特性如下:

- 总线长度（机车内部）：＜30m
- 总线长度（机车重联）：＜400m
- CAN 波特率：100k bit/s
- RS232 波特率：115.2k bit/s
- 网络节点数（机车内部）：7 个
- 通信协议：CAN 2.0A
- 网卡模块控制器：89C52 (22.1184MHz)
- 工控机 CPU（显示屏用）：80386SX (33MHz)

3.3.2 网络布线拓扑结构

在每个东风 8B 机车中装有 7 个 CAN 节点，两台机车用一个特殊的 CAN 节点相连接，此节点称之为网卡。局域网中其他的节点包括显示屏、主控计算机、逻辑单元等。图 3-1 显示了这些节点的连接结构图。

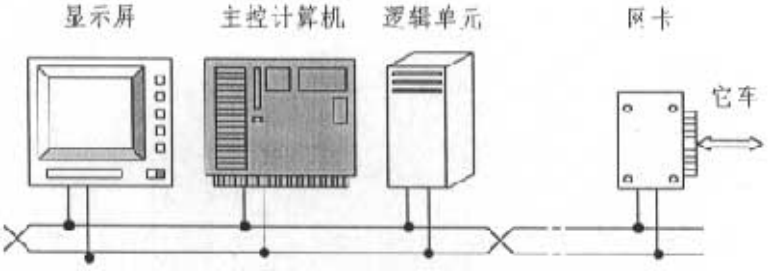


图 3-1 网络拓扑结构

如图 3-2 所示本项目中物理层中媒介连接子层是由 Philips 公司生产的 PCA82C250[7]或 PCA82C251[8]所实现的。物理信令子层及数据链路层是由 Philips 公司产协议控制器集成芯片 SJA1000 所实现。这些节点模块都通过保护电路与总线相连接。

以下将详细分析这些协议层的功能及实现。

3.5 物理层

3.5.1 芯片概要

82C250 是 Philips 公司生产的集成 CAN 收发器，它提供了对协议控制器和物理传输线的连接。本产品可与双线差动总线相连接（符合 ISO11898），适用于高达 1M bit/s 的波特率。其电气及物理特性见数据手册（参阅[7][8]）。表 3-1 为其管脚定义。

符号	管脚	描述
TXD	1	发送数据输入
GND	2	电源地
Vcc	3	电源
RXD	4	接收数据输出
Vref	5	参考电压输出
CANL	6	CAN低电平输入/输出
CANH	7	CAN高电平输入/输出
Rs	8	斜坡控制电阻输入

表3-1 82C250管脚定义

3.5.2 功能描述

在本项目中 82C250 实现了 CAN 协议的物理层中物理媒介连接子层的功能，它提供了对协议控制器和物理传输线的连接。本器件是为高速应用系统而设计的，其通信速度可达 1M bit/s。本器件对 CAN 总线提供了差动方式的数据发送能力，对 CAN 控制器提供了差动方式的数据接收能力，完全符合 ISO 11819 标准。

当芯片节温超过 160°C 时两个信号发送器输出管脚的电流都将下降，因为两个信号发送器输出管脚是电源的主要消耗部分，这时

其对电源的消耗将降低，因此导致芯片温度降低。此功能为热保护功能，当总线因某种原因而发生短路时此功能非常有效。在此情况下芯片的其他功能保持不变。

在汽车等实际应用场合可能发生电气击穿现象，CANH 和 CANL 两管脚也同时提供了击穿保护功能。

斜坡控制

管脚 8（Rs）提供了三种可供选择的工作模式：高速模式、斜坡控制模式及备用模式。

- 在高速模式中发送器的三极管以尽可能快的速度开启或关闭以实现陡峭的电平转换边沿。在此模式下对于上升沿或下降沿斜坡没有采取任何的限制措施。此模式只须将 8 管脚接地即可。
- 对于较低的通信速度或较短的总线长度，总线的物理传输媒介可以使用非屏蔽扁平电缆或或双绞线。在这种情况下为减少对外界的电磁辐射应当对上升沿和下降沿进行控制。这时可将管脚 8 用一电阻与地相连，则管脚 8 的电流与斜坡成比例地下降。
- 当在管脚 8 加一个高电平时，此芯片进入低电流消耗的备用模式。在此模式下发送器将被关闭，接收器转换为低电流状态。在这时如果检测到一个显性位（总线差动电压超过 0.9 伏）RXD 管脚将转换为低电平。在这种情况下微控制器应该作出反应，即通过 8 管脚将收发器转换为正常模式。因为此时收发器处于备用模式，所接收到的第一个信息将被丢失。

在本项目中收发器采用的是高速模式。

3.5.3 CAN 总线接口

为实现高速通信和保护节点电路不被外界强电压、电流所击穿可以采用各种方式实现收发器与总线的偶合。在本项目中 CAN 节点的最末端与总线相偶合的部分设计了一套保护电路。经过实验证实本保护电路可以防止高电压（<18 伏自恢复）及实现高速度数据通信（>100k bit/s）。

CAN 接口为 IDC10 针型插头，其信号定义见表 3-2。

管脚	1	2	3	4	5	6	7	8	9	10
信号	空	CANL	GND	空	空	GND	CANH	空	+5V	空

表 3-2 CAN 总线接口信号定义

3.6 数据链路层

3.6.1 芯片概要

SJA1000 是 Philips 公司生产的独立式 CAN 控制器，可支持 CAN 2.0A 及 CAN 2.0B 协议。

SJA1000 有两种操作模式:

BasicCAN 模式: 支持 CAN 2.0A 协议

PeliCAN 模式: 支持 CAN 2.0B 协议

当开机上电后系统的缺省模式为 BasicCAN 模式，这也是本项目的操作模式。然而，本项目也支持 PeliCAN 模式。

3. 6.2 结构框图

图 3-3 为 SJA1000 结构框图。

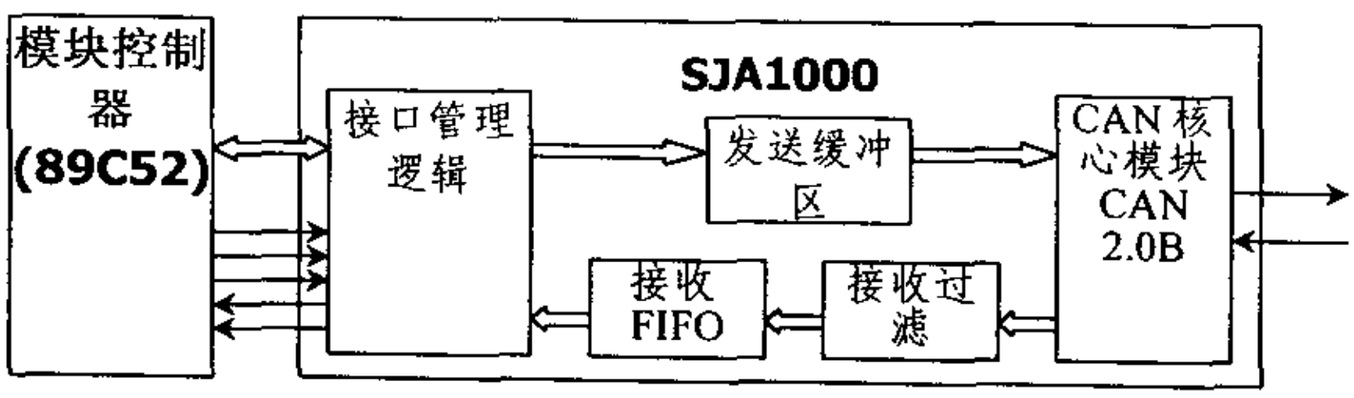


图 3-3 SJA1000 结构框图

从图中我们可以看到 *CAN 核心模块*根据 CAN 协议对于数据进行收发。

*接口管理逻辑*是本控制器与模块控制器（89C52）的桥梁。通过地址/数据复用总线对于 SJA1000 的寄存器进行的操作以及读写选通等任务都是由此部分完成的。在 PCA82C200 已有的 BasicCAN 基础上，本控制器有增加了 PeliCAN 功能。因此在本部分增加了与之相关的许多寄存器。

本控制器的*发送缓冲区*可存储一个完整的 CAN 信息帧（扩展帧格式）。一旦主控制器初始化了一个发送任务，接口管理模块将启动 CAN 核心模块将发送缓冲区的数据进行读取，并发送。

当接收信息时，CAN 核心模块将串行数据转换为并行数据，并提供给*接收过滤模块*，根据此可编程模块的设置 SJA1000 将决定是否将所接收的信息传给主控制器。

所有经过接收过滤模块的信息都存储在*接收 FIFO* 中。根据模式不同此 FIFO 可存储不同的数据，最多 32 帧信息可存储于此。这种功能使得用户可以更方便地处理当前任务，并且使数据超载的可能性大大降低。

为使得与模块控制器能方便地连接，SJA1000 提供了地址/数据复用总线及相关的读/写控制信号。这样对于模块控制器来说 SJA1000 就相当于地址映射在外部存储器上的 I/O 设备。

接收过滤机制:

SJA1000 有一个用户可编程的接收过滤模块。用户可通过它自动地对所接收到的信息标识符进行检查，并过滤掉很多无关的信息。用这种机制，无关信息不会存入接收 FIFO，更不会传给模块控制器，因此也减少了许多主控模块的工作。

此接收过滤模块是由接收代码和接收屏蔽寄存器所控制的[6]。所接收到的信息被一位位地与接收代码寄存器的值进行比较。而接收屏蔽寄存器则定义了标识符中那些位是需要比较的相关位（0=相关，1=不相关）。若一个信息被接收，其所有相关位必须与接收代码寄存器的值相对应。

BasicCAN 模式接收过滤机制:

接收过滤模块由两个 8 位的寄存器—接收代码寄存器（ACR）和接收屏蔽寄存器（AMR）—所控制。所接收到的所有信息其标识符中 8 位最关键位将与这些寄存器的值进行比较（参见下例）。比较后相关位相同的信息将被接收。

例:

接收代码寄存器（ACR）值	(msb)	11010010
接收屏蔽寄存器（AMR）值		00111000
具有这 11 位标识符的信息将被接收		11xxx010xxx

3.7 应用层

3.7.1 本层 的主要任务

本项目中的应用层功能是由 89C52 单片机所实现的．它控制 CAN 控制器并连接 PC（在本项目中 PC 作为司机显示屏，用来显示机车信息，并通过其显示屏键盘键盘发送司机命令）。

本层的主要任务有:

- 初始化 CAN 控制器
- 翻译 CAN 协议的信息
- 对 CAN 控制器的命令/请求或出错信息进行响应
- 管理 89C52 的数据/命令缓冲区
- 控制工作流程
- 通过标准 RS232 接口与 PC 通信

3.7.2 数据缓冲区框图

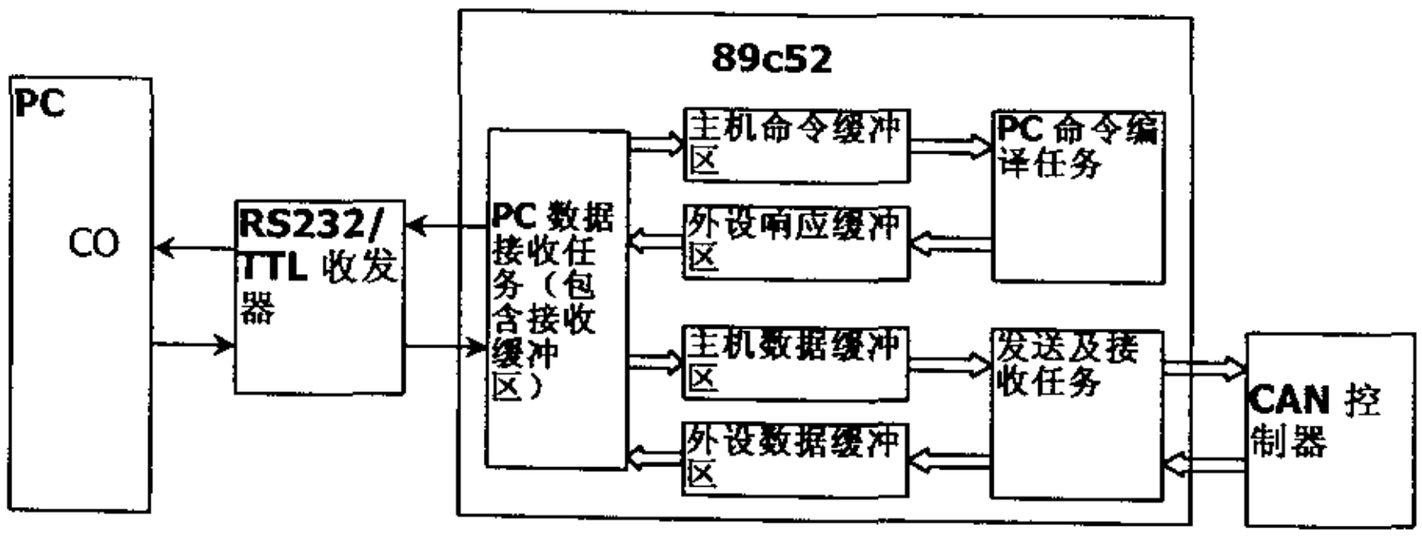


图 3-4 数据缓冲区框图

在开机上电后 89C52 将初始化寄存器，安装中断服务程序并生成数据/命令缓冲区。然后它将初始化 CAN 控制器（SJA1000）并进入正常运行阶段。

在本阶段 89C52 必须对 CAN 控制器及 PC 的数据/命令等信息作出响应，因此为这些信息设置不同的缓冲区是十分有效的。在本项目中所定义的数据/命令有:

- 主机命令: 从主控计算机（PC）所传来的命令
- 外设响应: 89C52 对主机命令的响应数据
- 主机数据: 从主控计算机（PC）所传来的并将通过 CAN 控制器所发送的数据
- 外设数据: 从 CAN 控制器接收并将传给主机的数据

根据这些定义缓冲区框图见图 3-4.

3.7.3 RS232 接口

本项目中 RS232 接口是 DB9 孔型插头，与 PC 的 COM 口相连，其信号定义见表 3-3.

管脚	1	2	3	4	5	6	7	8	9
信号	DCD	RXD	TXD	DTR	GND	DSR	RTS	CTX	RI

表 3-3 RS232 接口信号定义

第四章 网络配置

4.1 概述

只有优良的硬件设备，没有与之相配套的软件系统和正确的网络配置要想实现成功的网络应用无异于缘木求鱼。在第三章中介绍的保护电路及其他保护芯片对于实现系统的鲁棒性是非常重要的。在第五章中将会介绍软件系统及相关问题。本章则着重讨论网络配置及相关因素。

4.2 网络配置的主要项目

在网络配置方面当参数出现微小的错误时对系统也是很大的隐患。因为这些错误即使不会使网络立刻崩溃，也可能使得系统性能受到极大影响。影响网络运行的主要因素有:

1. 总线长度及电缆的类型
2. 总线分支长度
3. 总线上节点数目
4. 位定时参数
5. 通信速度
6. 电磁噪声及其他因素

在确定这些参数时既要进行理论计算，也要经过实验测量。在本章中这两种方法都将用来对以上因素中的前 5 项进行讨论。第 6 项因素不在本论文讨论之列。

4.3 最大总线长度及总线最大节点数

CAN 网络的最大允许总线长度从本质上讲是由以下这些物理因素所决定的。

1. 总线（导线等）及节点（CAN 控制器，CAN 收发器等）所造成的环路延迟。
2. 节点间因晶振公差引起的不同节点间位定时量的长度有所不同。

3. 因总线的串连电阻及节点并联电阻所造成的信号强度下降。

在本节将讨论由因素 3 所造成的影响，因素 1 及 2 所造成的影响将在 4.4 节中进行讨论。

在东风 8B 内燃机车内部总线长度只有 30 米，节点数只有 7 个，因此肯定满足总线长度的要求。在下面的讨论中将只讨论机车重联线的总线长度。

在 ISO 11898[1]中总线拓扑结构被定义为单一的环行连接，因为这种结构能使反射效应最小化。

在静态条件下某总线节点的输入电压是由此节点内电阻及流过此节点的电流所决定的。在一个显性位被发送期间，发送节点的输出三极管将被开启，产生电流。当一个隐性位被发送期间，发送节点的输出三极管将被关闭。

因此在某节点的输入端其差动电压取决于以下因素（参照图 4-1）：

- 发送节点的输出差动电压（$V_{diff.out}$）
- 总线电缆电阻
- 接收节点的输入电阻（R_{diff}）

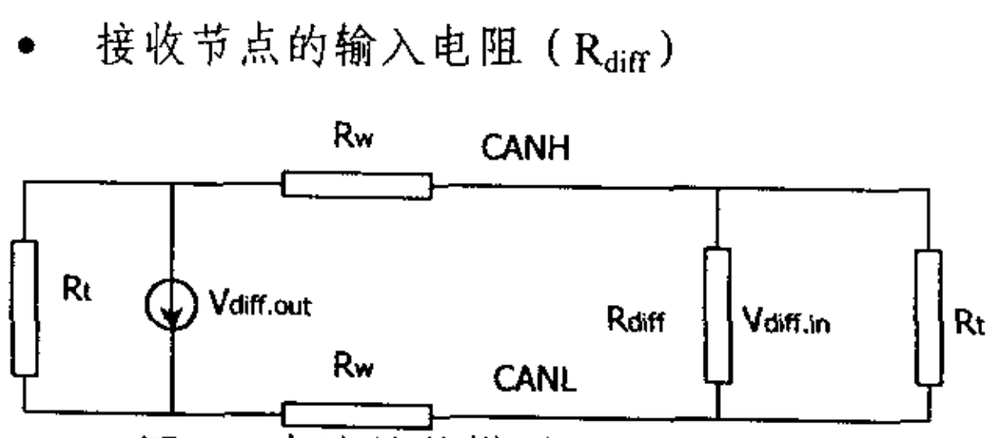


图 4-1 电流计算模型

R_t:终端电阻

R_w: 电缆电阻

根据图 4-1 接收节点的差动输入电压可由下面公式计算。

$$V_{diff.out} = V_{diff.in} + 2 * R_w * I_w \qquad (1)$$

$$Iw = \frac{Vdiff.out}{2Rw} + \frac{Rdiff * Rt}{Rdiff + Rt} \qquad (2)$$

($V_{diff.in}$>1.0V, $V_{diff.out}$=1.5V, R_{diff}>20kΩ, R_t=124Ω)

电缆: ISO 11898(automotive) 0.25mm² ρ=70Ω/km

根据（1）及（2）式可得总线长度为:

L<442m

在实际应用中总线长度为 400 米（在决大多数情况下小于 100 米），因此可以满足总线长度要求。

4.3 位定时参数

在很多情况下 CAN 的为同步功能可以对不适当的参数配置进行弥补，使得只在很偶然的情况下将产生一个出错的信息帧。

然而在仲裁期间两个或两个以上节点可能同时发送数据，对数据采样点的不适当选择可能使发送节点中的一个进入被动错误模式．对于这种偶然错误的分析及讨论需要对 CAN 位定时方法有比较细致的了解。

本节的目的并不是介绍计算 CAN 总线位定时的步骤（需要详细信息可参照[18]），而是介绍 CAN 位定时的方法、参数以及需要考虑的相关因素。

4.3.1 CAN 协议位定时

CAN所支持的波特率范围很广，从 5k bit/s 到 1000k bit/s。每个 CAN 节点都有其自己的时钟发生器（通常是一个石英晶体振荡器）．这些时钟发生器的震动周期（f_{osc}）也许有一定差异。每个 CAN 节点都可以分别对位定时参数进行设置，但必须符合一个共同的波特率。

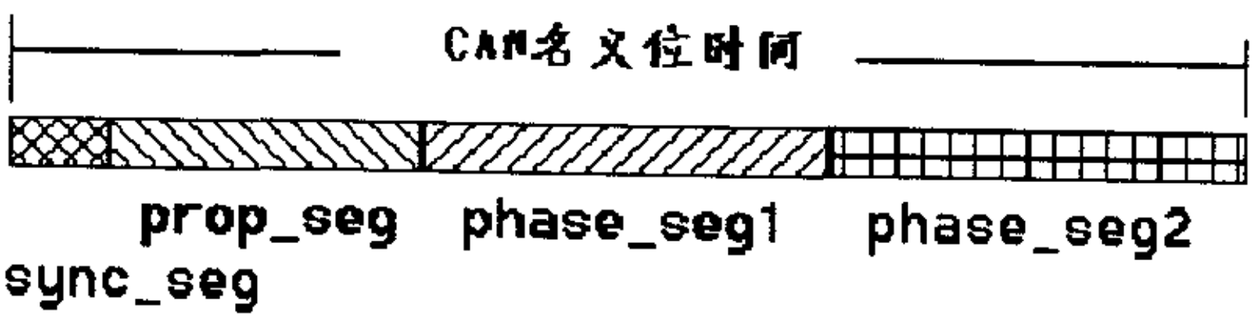


图 4-2 CAN位定时

这些石英晶体振荡器的频率也不总是一个常数，随着温度，电压及器件的老化，此频率会发生微小的变化．当这些变化所引起的差异在一定的范围之内时 CAN 节点可以通过从新同步机制（resynchronizing）来对位流进行弥补。

根据 CAN 协议，一位 CAN 信息可分成四个部分（segment）(参见图 4-2)，即同步段（synchronization segment）、传播时间段（propagation time segment）、缓冲阶段 1（phase buffer segment 1）和缓冲阶段 2（phase buffer segment 2）．这些段中每个都包含可编程的时间量（参见表 4-1）。这些时间量的最小单位就是位定时的时间单位（t_q）。此单位取决于 CAN 控制器的系统时钟（fsys）及分频器参数（BRP）：t_q=BRP/f_{sys}．通常 f_{sys}=f_{osc}。

参数 (t_q)	描述
BRP (<32)	定义t_q长度
Sync_Seg (1)	固定长度，将系统时钟与总线输入同步
Prop_Seg (<8)	弥补物理延迟时间
Phase_Seg1(<8)	在同步过程中可能被加长
Phase_Seg2(<8)	在同步过程中可能被缩短
SJW (<4)	同步跳转宽度

表4-1: CAN位定时的参数

同步段（Sync_Seg）是一位CAN信息中CAN总线上电平发生跳变的时间；如果在同步段以外发生CAN电平跳变则称此跳变为跳变阶段错误。

传播时间段（Prop_Seg）是用来弥补CAN网络上物理时间延迟的。这些延迟时间包括信号在网络上传输所造成的延迟，也包括在节点内部的延迟时间。

缓冲阶段1（Phase_Seg1）和缓冲阶段2（Phase_Seg2）及同步跳转宽度（SJW）是用来弥补石英晶体振荡器误差所造成时间差异。每当出现隐性电平向显性电平跳变的信号沿时将会发生同步。在同步过程中缓冲阶段可能延长或缩短。因此缓冲阶段的作用是来调整采样点与跳变边沿间的宽度。在每个采样周期（t_q）控制器都将对总线进行采样。如果在连续两次采样中出现先隐性位后显形位时将会发生同步。如果在同步段发生信号跳变边沿则发生同步化，否则跳变与同步段间的距离成为阶段错误。若跳变在同步段前发生称为负阶段错误，否则为正。

4.3.2 相关因素

振荡器误差范围(Oscillator Tolerance Range)

在CAN网络中每个节点都是通过各自的振荡器来获得各自的位定时值的。在实际的系统因公差、老化和温度变化等原因中振荡器的实际频率（f_{clk}）与其标称频率间会有一定误差。所有这些因素对频率所产生的影响统称为振荡器误差，可以用Δf来表示。

在不同的系统中可以用不同的方法产生参考时钟，例如石英晶体振荡器（Δf<0.1%）—本项目中就是采用此方法—和陶瓷振荡器（Δf<1.2%）。

传播延迟（Propagation delay）

在CAN总线系统中允许节点间为竞争总线访问权而进行非破坏性仲裁，也允许帧内应答，因此传播延迟非常重要。当多个节点在同时发送信息标识符时将发生仲裁。因为节点是靠位边沿进行同步的，过度的传播延迟将导致无效的仲裁。在一定的波特率下各种各样的延迟将限制CAN网络的长度。在两个CAN节点A与B间所有（由CAN控制器、CAN收发器及总线所造成的）延迟时间称为A到B的延迟，用符号$t_{porp(A,B)}$表示。

同样可以定义由CAN节点B到A的延迟$t_{porp(B,A)}$。因此系统中每个节点总回环传播延迟应为$t_{porp(A,B)}+t_{porp(B,A)}$。

在本系统中传播延迟主要产生于保护电路。此电路在串联及并联两方面对CAN节点进行了电气保护，以增强系统的抗干扰能力和安全性，但也带来了一些传播延迟。经实际测试此延迟时间可以满足波特率100k bit/s的通信速度要求。

4.4 通信速度

4.4.1 概述

人们总是希望在通信中实现尽可能高的通信速度，但高速通信的要求却常常与低成本、长距离、系统鲁棒性强等要求相矛盾。

控制器局域网（CAN）本身是一种高速度，低成本，多主通信的串行总线系统，具有很高的数据安全性，非常适合于实时控制。很多的高速通信网络就是用高达1M bit/s的CAN网络所实现的．但这并不是说一个CAN节点可以在1秒内向其他节点发送1M位（125K字节）数据，因为在CAN数据帧中还要发送启始位、仲裁场、控制场、校验场、应答场和截止位。在实际系统中向CAN控制器写发送数据及发送请求等的时间也必须考虑进去。本节将阐述怎样计算CAN总线的最大理论传输速度和测量实际传输速度。

为说明本文所做的计算和测量，特定义以下速率：

- 波特率（比特率）：传输1位数据所占时间的倒数，单位：位/秒。
- 最大理论有效数据传输速率：依据CAN传输协议，1秒内最多可传输的有效数据量，单位：字节/秒。
- 某理想软硬件系统单向最大有效数据传输速率：某理想系统1秒内单向数据传输的最大有效数据量，单位：字节/秒。

此理想系统定义是：

1. 系统有两个节点，A节点称为发送节点，B节点称为接收节点。
2. B节点以足够快的速度进行接收，即永不发生数据超载等错误。
3. A节点以最快速度进行发送，即除写入发送数据和设置发送请求命令外其余操作时间可忽略不记或与发送同步进行。

对发送节点，接收节点的定义为:

- 发送节点（TRANSMITTER）：一个信息的初始发送单元称之为该信息的发送节点。此节点在总线空闲或丢失仲裁前一直称为发送节点。
- 接收节点（RECEIVER）：当一个节点不是发送节点，此时总线也不空闲时此节点成为接收节点。

4.4.2 CAN 数据帧结构

标准帧

标准帧是由七个场所组成的：启始位（SOF）、仲裁场（arbitration field）、控制场（control field）、数据场（data field）（包含 0 至 8 位数据）、校验场（CRC field）、应答场（ACK field）和帧结束场（end of frame）.

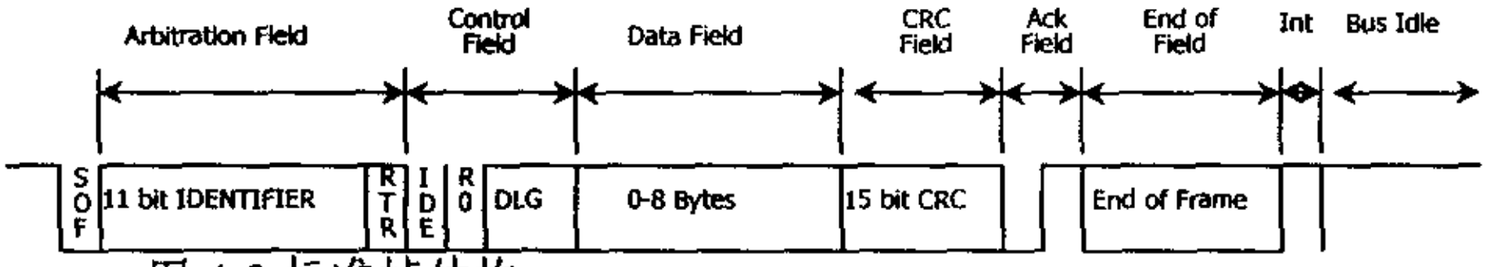


图 4-3 标准帧结构

- ***启始位（SOF）***：标志着一帧数据的开始，由一个显性位所组成.
- ***仲裁场（arbitration field）***:包括标识符（identifier）和远程请求位（RTR）.

  标识符（identifier）:由 11 位组成，先发送高位后低位（ID10—ID0）

  远程请求位（RTR）：在数据帧中此为为显性位，在远程帧中为隐性位.

- ***控制场（control field）***：由六位组成，包括数据长度码（data length code）和两位保留位.

- *数据场（data field）*:包含 0 到 8 字节所要发送的数据，高位在前低位在后。
- *校验场（CRC field）*:包含 CRC 循环冗余校验序列（15 位）和一个 CRC 分隔符（单一隐性位）。
- *应答场（ACK field）*：2 位，包括 1 位应答间隙和 1 位分隔符。
- *帧结束场（end of frame）*：由 7 个连续的隐性位所组成，标志着一个数据帧或远程帧的结束。

扩展帧

扩展帧是由七个场所组成的，它与标准帧只在以下几方面有所不同：

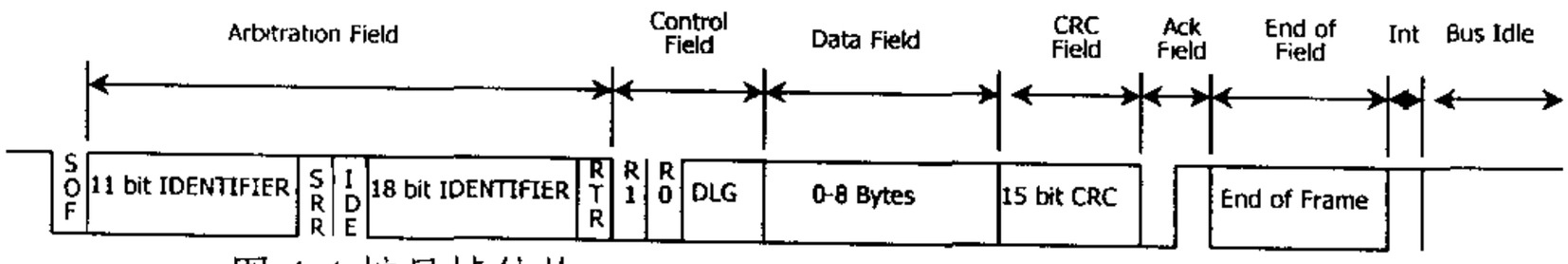


图 4-4 扩展帧结构

- *仲裁场（arbitration field）*：在标准帧中包括 11 位标识符（identifier）和远程请求位（RTR）。在扩展帧中包括 29 位标识符（identifier）、替代远程请求位（SRR）、IDE 位和远程请求位（RTR）。

帧间空间

数据帧和远程帧之间是用帧间空间来分隔的。帧间空间包含间歇场和总线空闲。

- *间歇场*包含三个隐性位。在此期间不允许发送数据帧或远程帧。只有当超载时可发超载信号
- *总线空闲*是在间歇场之后。在此期间为总线空闲时间，因此任何需要发送数据的节点都可以开始发送数据。如在此期间发生一个显性位的发送则标志着一帧信息的开始（帧启始—SOF）。

4.4.3 位流编码

在一帧数据中帧启始、仲裁场、控制场、数据场和校验场是用一种称为位填充的方法进行编码的。每当一个发送节点检测到 5 位相同的数值时它将在此位数据流中自动加入一位相反位。

在数据帧和远程帧的其他场中是固定格式，不进行位填充。

整个位流是用非归零码（Non-Return-to-Zero）进行编码的，即在整个位的长度内要么是显形位，要么是隐性位。因此 1 个标准帧的最多有效数据位是 64 位，总数据位是 81 位，加上帧间空间 3 位共 84 位。若 CAN 总线的波特率为 1M，1 秒内可传输总数位为 1M，有效数据位为 761.9K 位，即最大理论有效数据传输速率是 95238 字节/秒。

4.4.4 最大理论数据传输速度

标准帧:

根据标准帧的定义，在一帧数据中最多有效数据位是 64 位，总数据位是 81 位，加上帧间空间 3 位共 84 位。若 CAN 总线的波特率为 1M，1 秒内可传输总数位为 1M，有效数据位为 761.9K 位，即最大理论有效数据传输速率是 95238 字节/秒。

扩展帧:

CAN 扩展帧的仲裁场由 29 位数据标识码代替了 11 位数据标识码，并加入替代远程请求位和数据标识玛位，整个数据帧的长度为 99（加 3 位帧间空间），因此其最大理论有效数据传输速率是 76923 字节/秒。

位流编码:

根据 CAN 协议，当帧启始、仲裁场控制场数据场和校验场连续出现 5 个相同位时发送节点自动加入 1 个不同位。在此种情况下标准帧和扩展帧因位填充使数据长度有所增加，其最大理论有效数据传输速率可能降低为 81633 字节/秒和 65041 字节/秒。

4.4.5 理想 CAN 网络软硬件系统最大数据传输速度

在实际应用系统中此速度可能更为有用，因为在实际系统中理论传输速度往往达不到。

在以下部分将介绍对 4.1.1 节所提出的理想 CAN 总线软硬件系统进行测量，计算出其最大传输速度的方法。步骤如下:

1. 将 A 与 B 节点通过 CAN 进行连接，开机上电。
2. 对 A 与 B 节点进行初始化。
3. 在 A 节点停止除发送数据外的所有任务，以最大速度进行发送数据。
4. 用数字示波器对 CAN 总线上一帧数据进行捕捉。
5. 依据所捕捉到的数据计算传输速度。

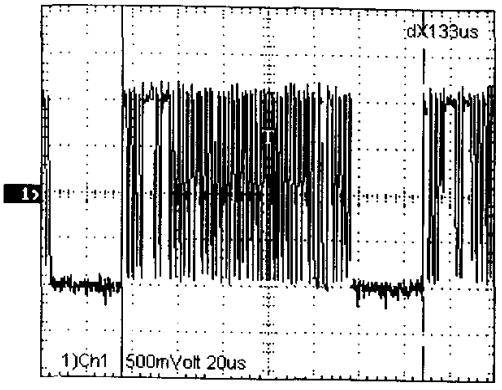


图 4-5 一帧标准帧的发送波形（波特率 1M）

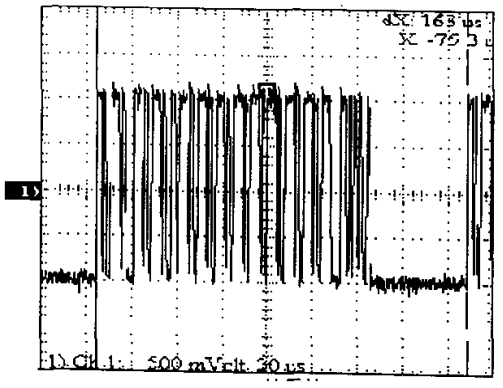


图 4-6 一帧扩展帧的发送波形（波特率 1M）

图 4-5 和图 4-6 显示了在 1M 波特率下一个标准帧和一个扩展帧发送的波形图。实际测得在 1M 波特率下标准帧和扩展帧的传输时间分别为 133μs 和 163μs，其相应的最大有效通信速率分别为 60150 字节/秒和 49080 字节/秒。

4.4.6 确定 CAN 总线最大数据传输速度

应如何确定实际的 CAN 总线网络中的最大有效通信速率呢？实际上本文所给出的两种最大速率分别有不同的用途。

最大理论有效数据传输速率是满足 CAN 传输要求的最大传输速率。在一个系统中若干节点同时发送数据时总有 1 个节点竞争成功而成为发送节点，而当此节点数据发送完后，其它节点的 CAN 控制器即可发送数据，而不再有发送数据的写入、写发送命令等时间。因此在

CAN 网络中最大理论有效数据传输速率是整个网络所能达到的传输速率的上限。

在一个网络系统中（尤其是实时控制系统）总有某些节点间数据传输要求较快，如一个闭环控制系统中从信号采集到控制器的信号传输和从控制器到执行器的数据传输，而其他节点间则要求不高。这些节点间的相互数据传输所能达到的最大有效速率应是本文所提出的单向最大传输速率。

表 4-2 给出了各个不同波特率下此两种最大速率的值和总线长度，可作为系统设计时的参考值。（总线长度参考 CIA(CAN in Automation)102 标准）

表4-2 最大传输速率和总线长度参考值

波特率（位/秒）	1000K	500K	250K	125K	100K
理论速率(CAN 2.0A)（字节/秒）	95238	47619	23809	11905	9524
单向速率(CAN 2.0A)（字节/秒）	60150	32786	17241	8850	7055
理论速率(CAN 2.0B)（字节/秒）	76923	38461	19231	9615	7692
单向速率(CAN 2.0B)（字节/秒）	49080	27397	14120	7229	5761
总线长度(米)	40	130	270	530	620

第五章 软件编程

5.1 概述

正如本文第三章所述本项目最重要的任务之一是实现一种智能型网卡。即通过它仅需要很少的几条命令就可实现通信功能。事实上仅仅三条基本命令便可实现通信功能，它们是初始化命令（initCanCmd），发送数据命令（sendDataCmd）和接收数据响应（receDataAck）。

本项目的软件可分为两大部分：单片机嵌入式软件及 PC 机设备驱动程序。单片机嵌入式软件是用 8051 汇编语言编写的，PC 机设备驱动程序是用 C 语言编写的。

在本章中将对这些软件的体系结构，数据流程及任务调度机制等进行介绍。

5.1.1 RS232 数据帧格式

PC 与网卡之间的通信是通过 RS232 接口进行的。通信数据是以帧为单位的。每一个帧都可以分为三个场：

- **帧启始**：标志一个数据帧的开始。为与原有的通信软件相兼容本场由四个有固定含义的字节所组成。在本系统的另一版本中提出了一种更新，更有效的帧启始，即应用 RS232 接口的 CTX 及 DTR 信号作为帧启始。
- **数据长度场**：在帧启始后面是 1 字节的数据长度场，标志着一帧数据的长度。每个数据帧的最大长度为 16 字节。
- **数据场**：数据场在数据长度场之后，由 0 到 15 个字节所组成。

在本项目中将由 PC 发往网卡的数据帧称为*命令帧*，而将由网卡发往 PC 的数据帧称为*响应帧*。

5.2 嵌入式软件系统

本网卡是由单片机及其他硬件和嵌入式软件系统所组成的。嵌入式软件是用 8051 汇编语言所编写并烧入 flash ROM 中。

5.2.1 本软件的源文件

本软件所用源文件及其说明列于表 5-1。

232CAN.ASM	本软件的主文件
ACK.ASM	响应任务定义文件
BASICCAN.ASM	BasicCAN 模式(CAN2.0 A) 任务定义文件
PELICAN.ASM	PeliCAN 模式(CAN2.0 B) 任务定义文件
RAM.INC	89C52 RAM 地址定义文件
CAN_REG.INC	SJA1000 寄存器地址定义文件
COMMAND.INC	232-CAN 命令及响应定义文件
CRYS16.INC	16M Hz 晶振位定时（BTR）定义文件
CRYS24.INC	24M Hz 晶振位定时（BTR）定义文件

5.2.2 软件编程定义

本软件软件编程中所用到的有特殊意义名词定义有*任务*、*工作*及*任务规划*，其定义如下。

在软件编程中一个很重要的目标是将软件组织成一个个既相对独立，又互相合作的程序。当这些程序共同运行时被称之为*任务*。在本项目中各个任务对于系统资源都可以直接访问，然而它们又分别拥有自己独立的运行空间和内存环境。

当 PC 向网卡发送一个命令后多个任务将被执行以完成此命令的功能。在此过程中所有参与实现此命令功能的任务组成一项*工作*。

为使多个任务共同完成一项工作需要有一种机制来调度这些任务，分配系统资源，这就是***任务规划***。在本项目中实行的是基于优先级的任务规划机制，这样系统就可以更迅速地对主机或网络节点的各种请求作出响应。

所有的这些任务及工作都必须依赖于硬件或软件事件来运行。例如当串行口输入中断事件发生时它将触发一个输入任务。中断是由一种特殊的程序—***中断服务程序（ISR）***来处理的。中断服务程序用来对所有的硬件事件作出响应，并触发响应的任务。因此中断

服务程序被称为硬件依赖型软件，而任务及工作被称为硬件独立型软件。

5.2.3 任务间通信

在不同的任务间信息的交换是必然的，为统一调度系统资源，又能使任务间相互协调工作，在本项目中采取了以下几种任务间通信方法：

- *共享内存*，简单数据共享使用
- *信息缓冲区*，任务间大量信息传递用。

正是采用这些方法，任务之间能相互沟通，执行出各种不同的工作。

5.2.4 主要任务一览

本项目中主要任务是：

1. **PC 数据接收任务**：接收从 PC 发送来的数据并将其存储。
2. **PC 命令编译任务**：编译从 PC 发送来的命令并通知相应的任务。
3. **PC 命令响应任务**：对 PC 发送来的命令进行响应。
4. **初始化任务**：对系统进行初始化。
5. **发送及接收任务**：将数据发送给 CAN 总线或从 CAN 总线接收数据。
6. **出错处理任务**：对 CAN 控制器的错误信息进行处理。
7. **任务规划任务（本项目的主任务）**：对不同任务进行调度，协调任务间工作。
8. **其他任务**：处理与本项目相关的其他工作。

图 5-1 显示了本项目中各个任务间的相互关系。

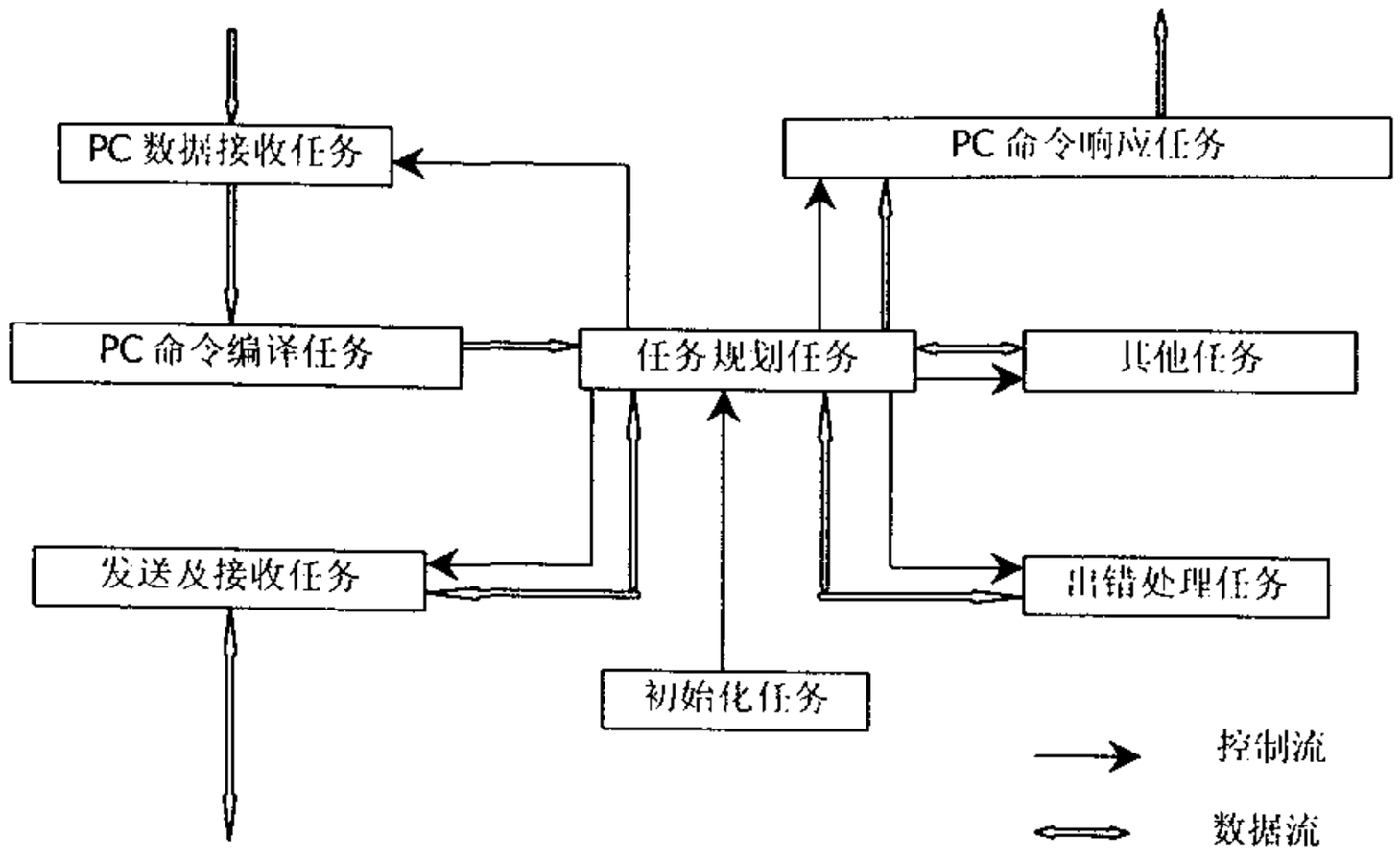


图 5-1 任务间相互关系及数据流

5.2.5 执行一项标准工作的任务流程

当 PC 向网卡发送了一帧命令后网卡将建立一向工作以执行这项命令。在本节将以一个很典型的命令—发送数据命令(sendDataCmd)—为例来介绍执行一向工作的任务流程。

当 PC 向网卡发送一帧命令时 PC 数据接收任务将对数据进行接收，判断帧的开始及结束，并将整个命令帧保存在命令缓冲区内。然后它将通知主任务命令接收完毕。

随后任务规划任务（主任务）将调度 PC 命令编译任务对此命令进行编译。

PC 命令编译任务随即对此命令进行编译，并检测出它是发送数据命令（sendDataCmd），因此它将此命令帧中的数据提取出，存入发送缓冲区.

在这时任务规划任务（主任务）将发现一帧数据已经被编译完毕，且发送缓冲区数据准备就绪. 因此它将发送缓冲区清空，并通知发送及接收任务启动发送程序。

发送及接收任务将启动发送程序，通过 CAN 控制器发送数据。如果数据被正确发送，它将返回一个 OK 状态。

当任务规划任务检测到 OK 状态后它将通知 PC 命令响应任务通知 PC 数据成功发送（如果 PC 允许接收响应信息）。

PC 命令响应任务将向 PC 发送一个 OK 消息。然后此工作结束。

5.2.6 中断服务程序

硬件中断处理程序（ISR）在本项目中扮演着非常重要的角色，因为外部的各种事件正是通过中断服务程序进行响应，并通知系统的。为快速地响应中断，本项目中的中断服务程序与各个任务相对独立，有自己独立的运行环境。

本项目的主要中断服务程序有：

- 定时器中断服务程序
- 串行口中断服务程序
- 外部中断 0 中断服务程序
- 外部中断 1 中断服务程序

5.2.7 初始化任务

在开机上电或正确的复位后单片机及 CAN 控制器进入复位状态。这时系统所执行的第一个任务是初始化任务。初始化任务将初始化单片机运行环境，设置中断服务程序所需资源，并对 CAN 控制器进行初始化。

本任务的流程图及主要功能列于表 5-2。

任务名	初始化任务
主要功能	初始化单片机及 CAN 控制器
相关中断	无

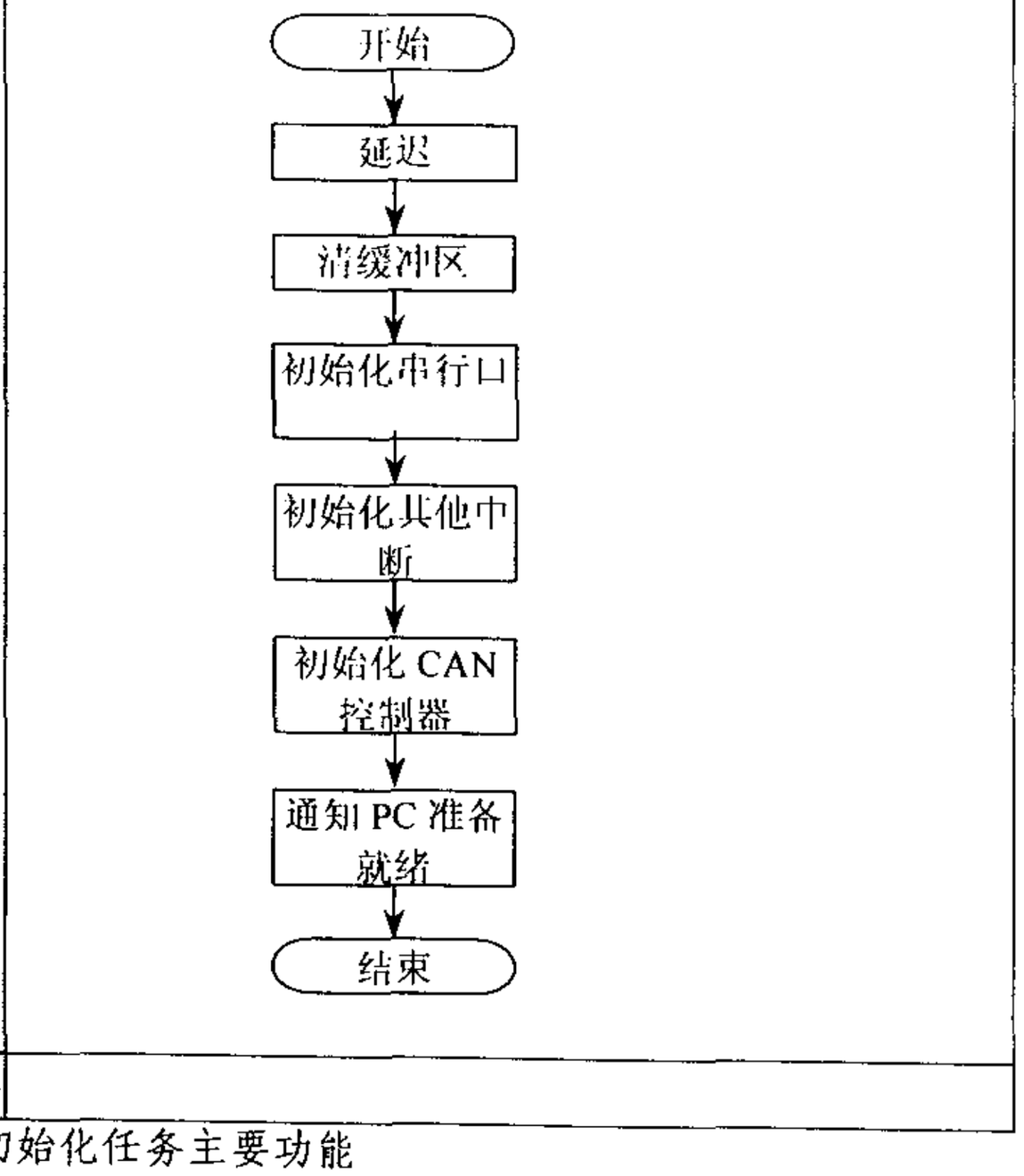
流程图	
注释	

表 5-2 初始化任务主要功能

5.2.8 任务规划任务

任务规划任务是本项目中最重要的任务。其主要职责为硬件维护、任务调度、错误处理及相关功能。本任务由初始化任务所初始化，并享有最高优先级。实际上在本软硬件系统中它起着操作系统的功能。

本任务的一个显著特点是自维护功能，即当某些严重错误发生时它可以调用初始化任务对整个系统进行初始化。在其他情况下当系统某些组件发生故障—例如 CAN 控制器进入被动错误状态—它将对此组件进行复位。

本任务的最重要功能是进行任务调度。为实现此功能本系统中所有的任务都在内存的一定区域建立起各自的信息，称之为信息表。任务规划任务将对此表进行查询，分析每个任务请求的优先级并将系统资源分配给优先级最高的任务。

本任务的流程图及主要功能列于表 5-3。

任务名	任务规划任务

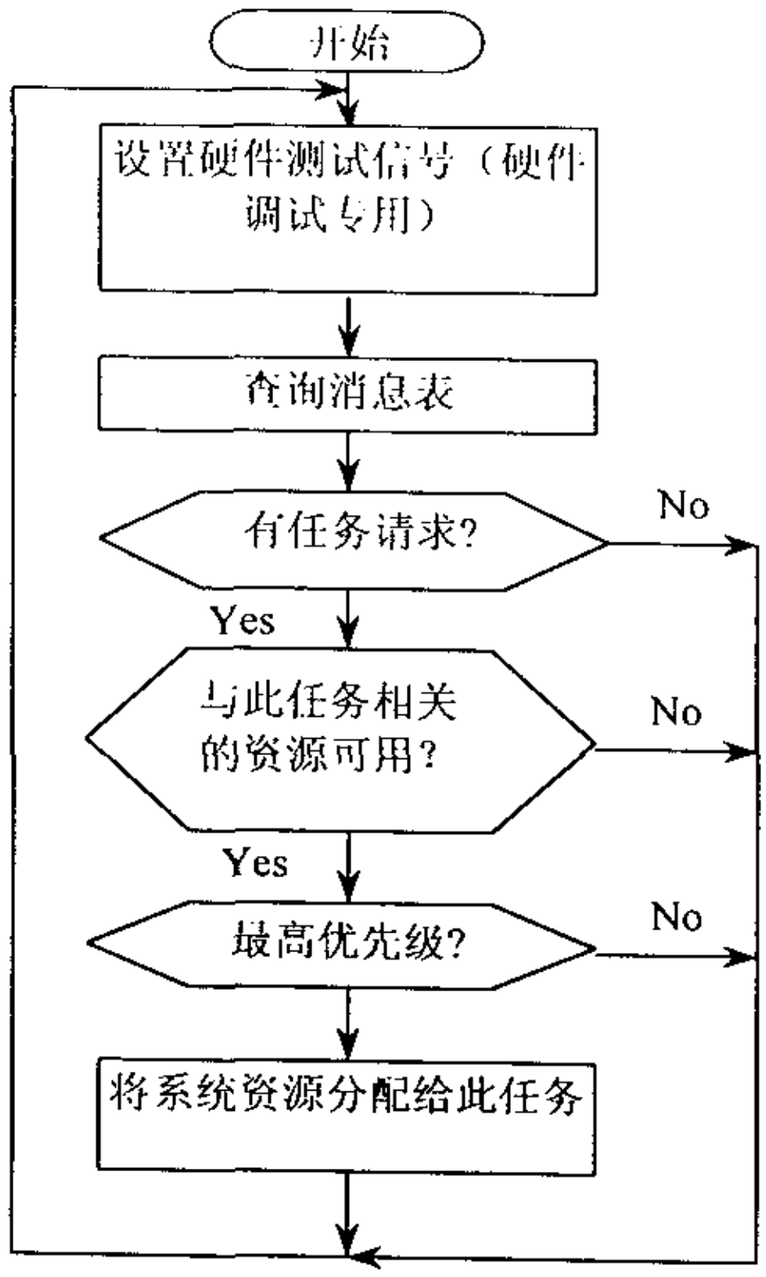
主要功能	维护硬件设备 调度任务 对系统错误进行处理 其他功能
相关中断	定时器中断 串行口中断（输入中断） 外部中断 0 外部中断 1
流程图	
注释	软件的自我维护功能在此表中没有介绍。此功能的实现依赖于在所有的软件程序最后部分加入一个特殊功能的宏，称之为 watch-dog 宏。

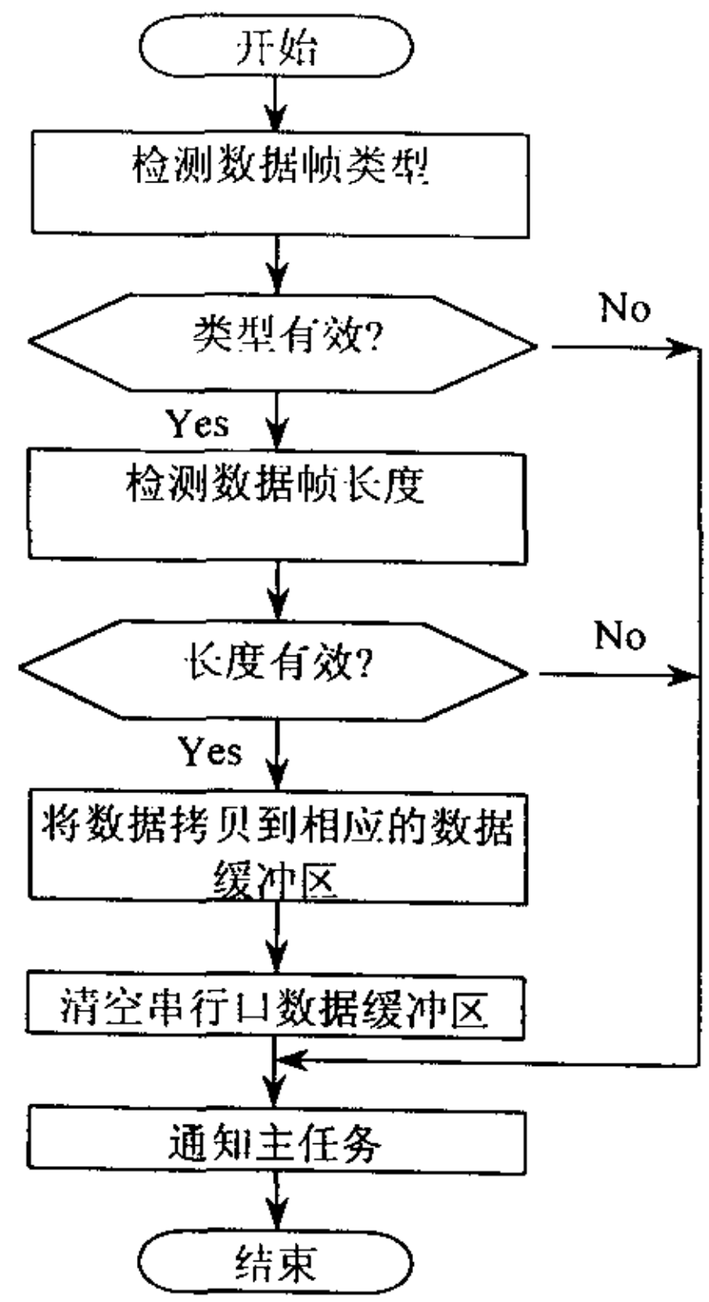
表 5-3 任务规划任务主要功能

5.2.9 PC 数据接收任务

当 PC 通过 RS232 总线想网卡发送了一字节数据后一个串行口输入中断请求将被生成．此时串行口中断服务程序将被调用，它将

把所接收到的数据存入一个特定的缓冲区—串行口数据缓冲区。当一个 RS232 数据帧的所有数据都被存入此缓冲区后 PC 数据接收任务将被调用。此任务将检查数据帧的类型，将其拷贝到相应的缓冲区，并清空串行口数据缓冲区。

本任务的流程图及主要功能列于表 5-4。

任务名	PC 数据接收任务
主要功能	检测数据帧的类型 拷贝数据到相应的缓冲区 清空串行口数据缓冲区
相关中断	串行口中断（输入中断） 外部中断 0
流程图	
注释	如果所接收到的数据帧的类型或长度无效它将不会被拷贝到相应的缓冲区．在这种情况下 PC 数据接收任务将通知主任务一个出错信息并返回。 如果所接收到的数据帧的长度及类型是合法的它将被拷贝

	到相应的缓冲区。此后 PC 数据接收任务将通知主任务一个 OK 信息并返回。

表 5-4 PC 数据接收任务主要功能

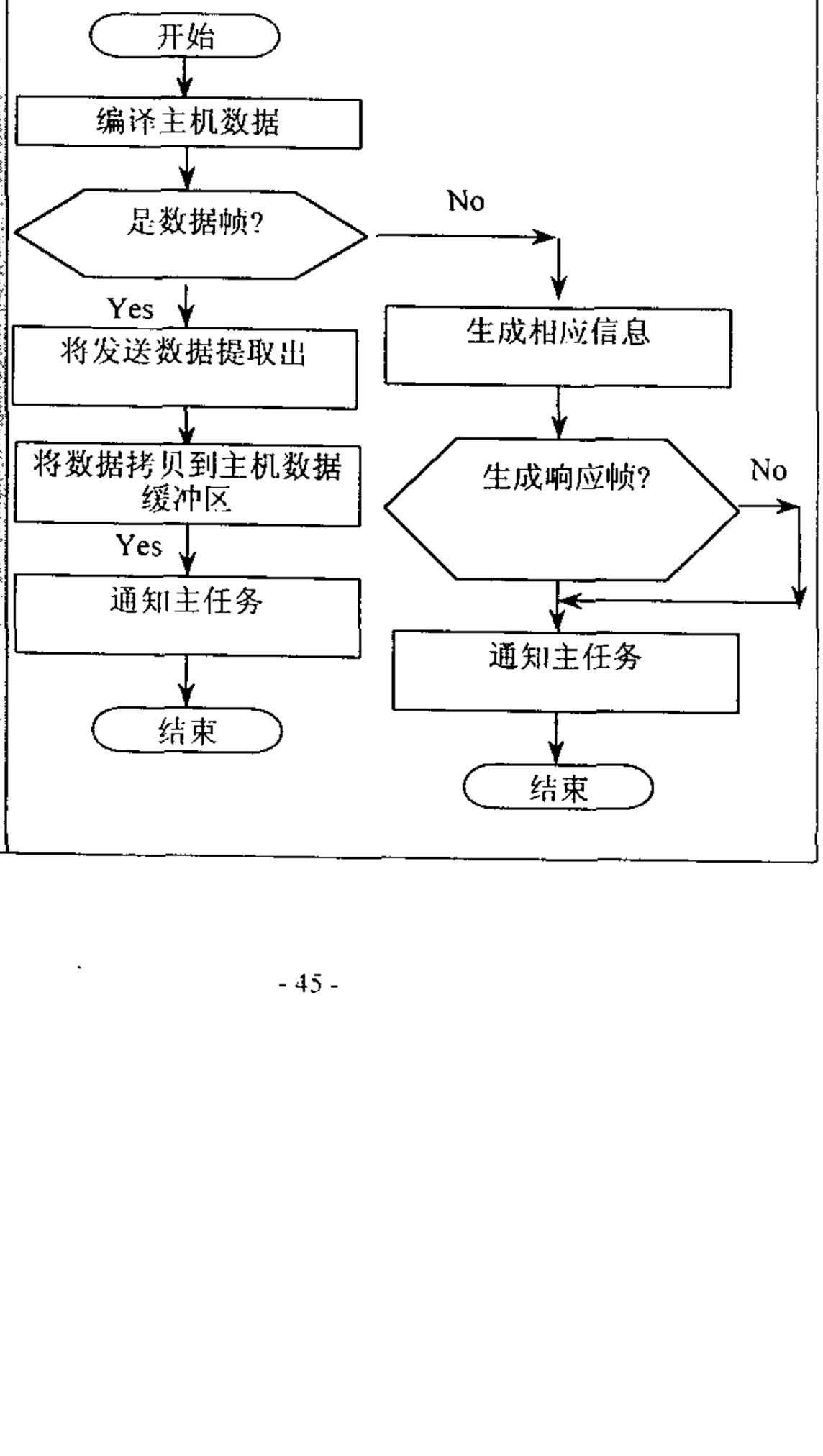
5.2.10 PC 命令编译任务

总的来讲从 PC 传来的数据可以分为两种类型：数据帧和其他类型帧。

数据帧是所有命令帧中最常用的，它将被编译且其中的数据将被传送到主机数据缓冲区中。

其他类型帧在系统中实现着初始化，系统维护等功能。这些帧将被编译，并根据系统设定和主机需要生成响应帧。

本任务的流程图及主要功能列于表 5-5。

任务名	PC 命令编译任务
主要功能	编译数据帧 生成响应信息（若需要） 通知主任务返回信息
相关中断	无
流程图	

注释	

表 5-5 PC 命令编译任务主要功能

5.2.11 PC 命令响应任务

当一帧响应信息已经生成或已经从 CAN 总线接收到数据，PC 命令响应任务将被调用，以相 PC 发送数据。本任务的另一功能是对所发送的信息进行测试，使之符合 RS232 数据帧格式。

本任务的流程图及主要功能列于表 5-6。

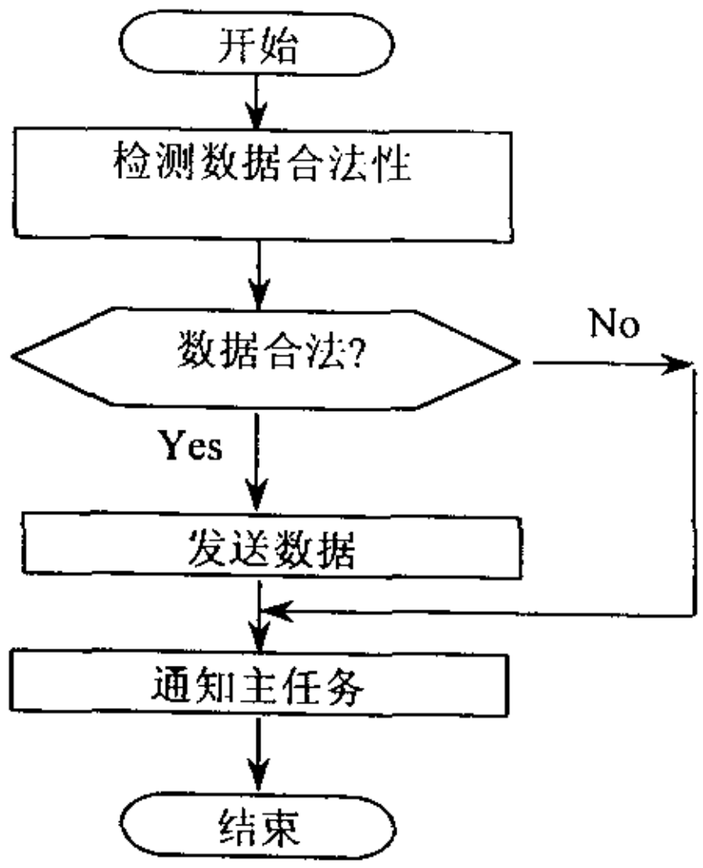
任务名	PC 命令响应任务
主要功能	检测响应帧 向 PC 发送数据
相关中断	无
流程图	
注释	

表 5-6 PC 命令响应任务主要功能

5.2.12 发送及接收任务

本任务实现了单片机与 CAN 控制器间双向通信功能，是由多个子任务及中断服务程序所组成的。这些子程序及中断服务程序主要有:

- 数据发送子任务
- CAN 控制器状态查询子任务
- CAN 控制器数据输入中断服务程序

这些子任务及中断服务程序都有各自的运行环境，并通过相应的消息与主任务通信，因此本任务可视为一个任务集合。

本任务的流程图及主要功能列于表 5-7。

任务名	发送及接收任务
主要功能	向 CAN 控制器发送数据 查询 CAN 控制器状态 处理 CAN 数据输入中断
相关中断	外部中断 1
流程图	子任务及中断服务程序都有各自的运行环境及流程图
注释	

表 5-7 发送及接收任务主要功能

5.2.13 出错处理任务

本智能型网卡的一个重要特点是其具有自我维护及出错处理功能，用此功能单片机可自动维护其系统资源，对 CAN 控制器的出错信息进行响应并根据需要向 PC 发送相应信息。

自我维护的功能是由主任务（任务规划任务）所完成的，出错处理功能主要是由本任务所完成的。

出错处理任务对 CAN 控制器的错误信息进行检测，并依据此信息作出相应的响应。

CAN 控制器可运行于两种模式：BasicCAN（CAN 2.0A）及 PeliCAN(CAN 2.0B)。在 BasicCAN 模式当 CAN 控制器的总线状态位或出错状态位发生变化时将会有出错中断发生。在 PeliCAN 模式出错中断必须处理以下这些中断：

- 总线错误中断（Bus error interrupt—BEI）
- 被动错误中断（Error passive interrupt—EPI）
- 数据超载中断（Data overrun interrupt—DOI）
- 错误警告中断（Error warning interrupt—EI）

本任务的流程图及主要功能列于表 5-8。

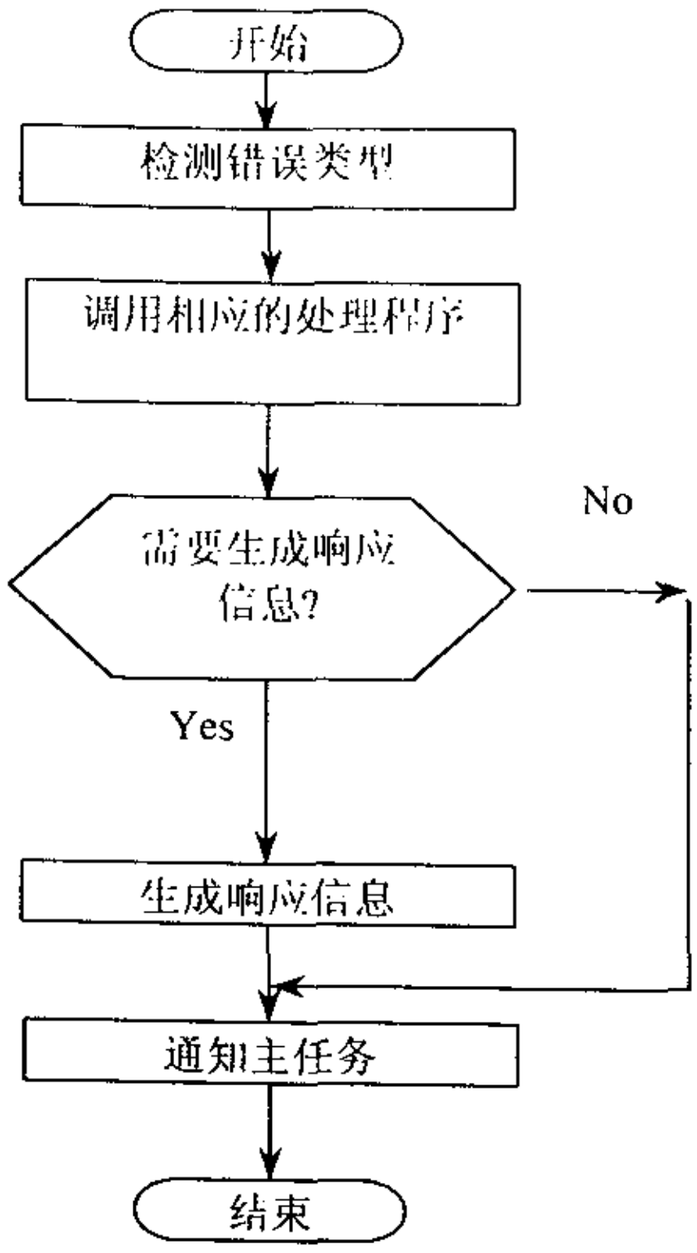
任务名	出错处理任务
主要功能	处理 CAN 控制器出错中断及相关信息
相关中断	外部中断 1
流程图	
注释	

表 5-8 出错处理任务主要功能

5.2.14 其它任务

本系统中的其他任务负责对输入/输出及单片机本身的事件作出响应。其主要功能为:

- 初始化 RS232: 如果 PC 希望对其重新初始化
- 对某些命令进行响应: 获取版本号命令、读寄存器命令、写寄存器命令、进入睡眠模式命令、取消发送命令、接收过滤命令、自测试命令、进入监听模式命令等。
- 对某些响应作出回答: 读寄存器响应、仲裁丢失响应等。

5.3 PC 设备驱动程序

5.3.1 命令及响应一览

本节着重讨论 PC 设备驱动程序的编程。本项目中所用的命令及响应列于表 5-9。

序号	命令		响应	
系统命令				
01h	init232Cmd	初始化 RS232		
03h	getVersionCmd	获取版本号	getVersionAck	获取版本号
04h	readRegCmd	读寄存器	readRegAck	读寄存器
05h	writeRegCmd	写寄存器		
CAN2.0A 命令				
11h	initCanCmd_a	初始化 CAN		
12h	sendDataCmd_a	发送数据	sendDataAck_a	发送数据
13h	aboutSendCmd_a	取消发送		
14h	goToSleepCmd_a	进入睡眠状态		
18h			receDataAck_a	接收数据
19h			OverrunAck_a	数据超载
1ah			errorAck_a	出错响应
CAN2.0B 命令				

21h	initCanCmd_b	初始化 CAN		
22h	sendDataCmd_b	发送数据	sendDataAck_b	发送数据
23h	aboutSendCmd_b	取消发送		
24h	goToSleepCmd_b	进入睡眠模式		
25h	acceFilterCmd_b	接收滤波器		
26h	selfTestCmd_b	自我检测模式		
27h	listenOnlyCmd_b	进入监听模式		
28h			receDataAck_b	接收数据
29h			errorAck_b	出错响应
2ah			overrunAck_b	数据超载
2bh			passiveAck_b	错误被动状态
2ch			arbiLostAck_b	仲裁丢失
2dh			busErrorAck_b	总线错误

表 5-9 命令及响应列表

5.3.2 系统命令

序号	01h

命令	初始化 RS232			
格式	03h 01h b			
	b: 波特率	0-115200 3-19200 6-2400	1-57600 4-9600 7-1200	2-38400 5-4800
响应	无			
注释	系统上电后默认波特率为 9600bps			

序号	03h
命令	获取版本号
格式	02h 03h
响应	（16 进制）16 03h 4eh 4ah 55h 32h 30h 30h 20h 56h 65h 72h 31h 2eh 32h 30h （ASC II 格式）▸♥NJU200 Ver1.20

序号	04h
命令	读寄存器
格式	??h 04h r...r
	r:寄存器号（最多 7 个）
响应	??h 04h r v ... r v
	r:寄存器号　　v:寄存器值

序号	05h
命令	写寄存器
格式	??h 04h r v ... r v

	r:寄存器号（最多 7 个）　　v:寄存器值
响应	无

5.3.3 CAN 2.0A 命令

序号	11h
命令	初始化 CAN
格式	06h 11h a m btr0 btr1
	a:　　接收代码寄存器值 m:　　接收屏蔽寄存器值 btr0:　位定时寄存器 0 值 btr1:　位定时寄存器 1 值
响应	无

序号	12h
命令	发送数据
格式	??h 12h d ... d
	d:　　要发送的数据
响应	03h 12h ack
	ack:　发送结果（00h-发送缓冲区满　　0ffh-发送成功）

序号	13h
命令	取消发送
格式	02h 13h

响应	无

序号	14h
命令	进入睡眠状态
格式	02h 14h
响应	无

序号	18h
命令	无
响应	接收数据
格式	??h 18h d ... d
	D: 接收的数据

序号	19h
命令	无
响应	数据超载
格式	02h 19h
备注	转换卡在发送此响应前自动清除数据超载位

序号	1ah
命令	无
响应	出错响应
格式	03h 1ah s

	s: 状态寄存器值

5.3.4 CAN 2.0B 命令

序号	21h
命令	初始化 CAN
格式	12 21h a0 a1 a2 a3 m0 m1 m2 m3 btr0 btr1
	a0—a3: 接收代码寄存器 0—3 值 m0—m3: 接收屏蔽寄存器 0—3 值 btr0: 位定时寄存器 0 值 btr1: 位定时寄存器 1 值
响应	无

序号	22h
命令	发送数据
格式	??h 22h d ... d
	d: 要发送的数据
响应	03h 22h ack
	ack: 发送结果（00h-发送缓冲区满 0ffh-发送成功）

序号	23h
命令	取消发送
格式	02h 23h
响应	无

序号	24h
命令	进入睡眠模式
格式	02h 24h
响应	无

序号	25h
命令	进入接收滤波器模式
格式	02h 25h
响应	无

序号	26h
命令	进入自我检测模式并发送测试数据
格式	??h 26h d ... d
	d: 要发送的数据
响应	无

序号	27h
命令	进入监听模式
格式	02h 27h
响应	无

序号	28h

命令	无
响应	接收数据
格式	??h 28h d ... d
	d: 接收的数据

序号	29h
命令	无
响应	出错响应
格式	06h 29h s e r t
	s: 状态寄存器值 e: 出错捕捉寄存器值 r: 接收出错寄存器值 t: 发送出错寄存器值

序号	2ah
命令	无
响应	数据超载
格式	02h 2ah
备注	转换卡在发送此响应前自动清除数据超载位

序号	2bh
命令	无
响应	错误被动状态

格式	06h 2bh s e r t
	s: 状态寄存器值 e: 出错捕捉寄存器值 r: 接收出错寄存器值 t: 发送出错寄存器值

序号	2ch
命令	无
响应	仲裁丢失
格式	03h 29h a
	a: 仲裁丢失捕捉寄存器值

序号	2dh
命令	无
响应	总线错误
格式	06h 2dh s e r t
	s: 状态寄存器值 e: 出错捕捉寄存器值 r: 接收出错寄存器值 t: 发送出错寄存器值

5.3.5 通信软件

本节所列软件为 PC 与网卡间通信软件，用 C 语言编写。虽然这些软件可以单独编译、运行，在实际应用中这些软件是作为司机显示屏软件的一部分（通信接口）而存在的。

- 例 1：基本通信功能

```
#include <dos.h>
#include <conio.h>
#include <stdio.h>

void sendCommand(unsigned char[]);          /*send command function */
void main(void)
{
 unsigned int receData[16];                 /* RX data buffer          */
 unsigned char i=0,j=0;
 unsigned char msr;
 unsigned char getVersion[]={2,0x03};       /* get version command     */
 for(j=0;j<16;j++) receData[j]=0;

 /* initialize COM 1 */
 clrscr();
 outportb(0x3fb,0x80);                      /* set baud rate           */
 outportb(0x3f8,0x0c);                      /* 9600 bps                */
 outportb(0x3f9,0x00);
 outportb(0x3fb,0x03);                      /* set mode                */
 outportb(0x3f9,0x0);                       /* disable interrupt       */
 outportb(0x3fc,0x00);                      /* initialize RTS pin      */
 receData[i]=inportb(0x3f8);                /* clear RX buffer (COM1) */
 receData[i]=inportb(0x3f8);
 sendCommand(getVersion);                   /* send the command        */

 /* receive data */
 while(!(kbhit()))
 {
        msr=inportb(0x3fe);                 /* detect start of frame   */
        if((msr&0x01!=0)&&(msr&0x10)==0)
        {
                i=0;
                printf("\n");
        }

        if((inportb(0x3fd)&0x01)==1)
        {
                receData[i]=inportb(0x3f8);/* receive data             */
                cprintf("%c",receData[i]);
                i++;
                 if(i>=16)          i=0;
        }
```

```
  }
}
void sendCommand(unsigned char* command)
{
 unsigned char i;
 outportb(0x3fc,0x03);                    /* send start of frame      */
 outportb(0x3fc,0x00);
 outportb(0x3fc,0x03);

 for(i=0;i<command[0];i++)                /* send data                */
 {
 while((inportb(0x3fd)&0x20)==0) {}
 outportb(0x3f8,command[i]);
 }
}
```

本程序实现了获取版本号的功能，其命令帧的定义为：

```
unsigned char getVersion[]={2,0x03};
```

当运行此程序后，网卡的响应为：

♥NJU200 Ver1.20

如要实现读取寄存器的值，可用如下命令帧：

```
unsigned char readReg[]={3,0x04,2};
```

本命令可实现读取寄存器 2 的值。

- 例 2 CAN 节点自测试程序

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <io.h>
#include <dos.h>

void int_ok(void);
void int_end(void);
void interrupt (*old_int4)(void);          /* old interrupt vector     */
void interrupt new_int4(void);             /* new interrupt vector     */
void init_232(void);
void x_sound(int xx,int yy);
void sendCommand(unsigned char[]);         /* send command function */

unsigned char data_get[300];               /* receive  data buffer     */
unsigned int count = 0;                    /* receive buffer pointer   */

unsigned int PORT = 0X3F8;          unsigned char IRQ4 = 0x0c;
unsigned char LSB = 0x30;           unsigned char MSB = 0x00;
unsigned char rs232baudrate[]={3,0x01,4}; /* initialize RS232          */

void main(void)
```

```
        {
         char ch;   int i,j;
         unsigned char initCan_b[]={12,0x21,0x00,0x00,0x00,0x00,0xff,
        0xff,0xff,0xff,0x00,0x1c};                    /* initialize CAN controller     */
         unsigned char selfTest_b[]={7,0x26,2,5,0,1,1}; /* self test command     */

         clrscr();
         gotoxy(7,18); cprintf("1 - COM1:3F8H, 2 - COM2:2F8H press<1,2> to
select");
         ch = getch();
         switch(ch)
         {
          case '1' :  PORT = 0x3f8;IRQ4 = 0x0c; break;
          case '2' :  PORT = 0x2f8;IRQ4 = 0x0b; break;
          default  : break;
         }

         clrscr();
         gotoxy(7,18); cprintf("0 - 115200   1 - 57600   2 - 38400   3 -19200");
         gotoxy(7,19); cprintf("4 - 9600     5 - 4800    6 - 2400    7 -1200");
         gotoxy(7,21); cprintf("Please select. Default: 9600");
         ch = getch();
         switch(ch)
         {
          case '0' : LSB=0x01; MSB=0x00; rs232baudrate[2]=0; break;
          case '1' : LSB=0x02; MSB=0x00; rs232baudrate[2]=1; break;
          case '2' : LSB=0x03; MSB=0x00; rs232baudrate[2]=2; break;
          case '3' : LSB=0x06; MSB=0x00; rs232baudrate[2]=3; break;
          case '4' : LSB=0x0c; MSB=0x00; rs232baudrate[2]=4; break;
          case '5' : LSB=0x18; MSB=0x00; rs232baudrate[2]=5; break;
          case '6' : LSB=0x30; MSB=0x00; rs232baudrate[2]=6; break;
          case '7' : LSB=0x60; MSB=0x00; rs232baudrate[2]=7; break;
          default  :break;
         }
         disable();
         clrscr();
         delay(100);
         init_232();                          /* initialize RS232              */
         old_int4 = getvect(IRQ4);            /* save old interrupt vector     */
         setvect(IRQ4, new_int4);             /* set new interrupt vector      */
         int_ok();                            /* enable interrupt              */
         sendCommand(initCan_b);              /* initialize CAN controller     */
         delay(100);
         sendCommand(selfTest_b);             /* go to self-test mode          */
         while(ch!=27)                        /* press ESC to return to DOS    */
         {
          delay(2000);
          if(count!=0)
          {
                printf("\n Receve:");
```

```
            for(j=1;j<count;j++)
                cprintf("%3x",data_get[j]);
            count=0;
    }
    if(kbhit())
            ch=getch();
    else ch='k';
  }
  int_end();
  setvect(IRQ4, old_int4);
}
void int_ok(void)
{
  int da;
  outportb(0x20,0x20);
  da=inportb(0x21);
  da &= 0x80;
  outportb(0x21,da);
  enable();
}
void int_end(void)
{
  int dat;
  dat=inportb(0x21);
  dat |= 0x10;
  outportb(0x21,dat);
  outportb(0x20,0x20);
  disable();
}
void x_sound(int xx,int yy)
{
  sound(xx);
  delay(yy);
  nosound();
}
void init_232(void)
{
  outportb(PORT+3,0x80);      /* set baud rate 9600              */
  outportb(PORT+0,0x0c);      /* LSB                             */
  outportb(PORT+1,0x00);      /* MSB                             */
  outportb(PORT+3,0x03);      /* 8-data,1-stop,no-p              */
  outportb(PORT+4,0x0f);      /* -RTS = 1                        */
  outportb(PORT+1,0x09);      /* enable receive,modem interrupt */
  sendCommand(rs232baudrate);
  delay(60);

  outportb(PORT+3,0x80);      /* set baud rate                   */
  outportb(PORT+0,LSB);           /* LSB  0c is 9600,30 is 2400 L.T.B */
  outportb(PORT+1,MSB);           /* MSB                             */
  outportb(PORT+3,0x03);          /* 8-data,1-stop,no-p              */
```

```
 outportb(PORT+4,0x0f);              /* -RTS = 1                          */
 outportb(PORT+1,0x09);              /* enable receive,modem interrupt    */
}

void interrupt new_int4(void)
{
 unsigned char iir,msr;
 disable();
 msr=inportb(PORT+6);                /* frame header information          */

 if((msr&0x01!=0)&&(msr&0x10)==0)
 {
 count++;                            /* detect start of frame             */
 data_get[count]=0;
 }
 count++;
 iir = inportb(PORT+2) & 0x07;       /* interrupt ID. register            */
 switch(iir)
 {
 case 0x04:                          /* receive data                      */
 if((inportb(PORT+5) & 0x1e) == 0x00)
 {
        data_get[count]=inportb(PORT);
 }                                   /* no error                          */
 else
 {
        inportb(PORT);
        x_sound(100,10); x_sound(1000,10);
        data_get[count] = 0xff;
 }
 break;
  }
  outportb(0x20,0x20);
  enable();
 }
 void sendCommand(unsigned char* command)
 {
  unsigned char i;
  outportb(PORT+4,0x0f);             /* send start of frame               */
  outportb(PORT+4,0x0c);
  outportb(PORT+4,0x0f);

  for(i=0;i<command[0];i++)
  {
  while((inportb(0x3fd)&0x20)==0) {}
  outportb(0x3f8,command[i]);
  }
 }
```

本程序将 CAN 控制器初始化为 CAN 2.0B 模式并发送测试命令。按 ESC 键可返回 DOS。自测试命令的定义为：

```
unsigned char selfTest_b[]={7,0x26,2,5,0,1,1};      /* self test command */
```

网卡对此命令的响应为：

7 28 2 5 0 1 1

响应中 28 是接收数据响应序号，‘2 5 0 1 1’是返回的测试数据。

参考文献

[1] Road vehicles - Interchange of digital information - Controller area network(CAN) for high-speed communication. ISO 11898. International Standardization Organization. 1993.

[2] CAN specification Version 2.0, Parts A and B. Robert Bosch GmbH. Postfach 30 02 40, D-70442 Stuttgart . 1991.

[3] Farsi M,Ratcliff K, Barbosa Manuel. Overview of controller area network. Computing and Control Engineering Journal v 10 n 3 1999 IEE p 113-120 0956-3385.

[4] Livani MA, Kaiser J, Jia WJ. Scheduling hard and soft real-time communication in a controller area network. Control Engineering Practice 7: (12) 1515-1523 1999/12.

[5] Tindell K, Burns A, Wellings AJ. Calculating Controller Area Network(CAN) Message Response-times. Control Engineering Practice 3: (8) 1163-1169 1995/08.

[6] Data Sheet SJA1000. Philips Semiconductors. 2000 May 11.

[7] Data Sheet PCA82C250 CAN controller interface. Philips Semiconductors. 2000 Jan. 13.

[8] Data Sheet PCA82C251 CAN controller interface. Philips Semiconductors. 2000 Jan. 13.

[9] Navet N. (LORIA-INPL), Song Y.Q. Reliability improvement of the dual-priority protocol under unreliable transmission. Control Engineering Practice 7 8 1999 Elsevier Science Ltd p 975-981 0967-0661.

[10] Travis A.R.L. (Cambridge Univ.), Collier, M. Software interface for an industrial control network using the CAN protocol. IEEE AFRICON Conference v 2 Sep 25-27 1996, IEEE p 1081-1082.

[11] Cena G. (Politecnico di Torino), Valenzano A. Overclocking of controller area networks. Electronics Letters 35 22 1999 IEE p 1923-1925 0013-5194.

[12] Rodrigues Luis, Guimaraes Mario , Rufino Jose. Fault-tolerant clock synchronization in CAN. Proceedings - Real-Time Systems Symposium Dec 2-4 1998 IEEE p 420-429.

[13] DeviceNet Specification, Volume I, Release 1.3. Open DeviceNet Vendor Association Inc. December 1995.

[14] How to Handle Data Overrun Conditions of the 82C200, 8xC592 and 8xCE598. Philips Semiconductors. AN95092.

[15] Dietmayer K. CAN Bit Timing. Philips Semiconductors. Technical Report HAI/TR9708, 1997.

[16] Dietmayer K. Overberg K. W. CAN Bit Timing Requirements, SAE Paper 970295, 1997

[17] Hank P. PeliCAN: A New CAN Controller Supporting Diagnosis and System Optimization. 4th International CAN Conference, Berlin, Germany, October 1997.

[18] Florian Hartwich, Armin Bassemir. The Configuration of the CAN Bit Timing. Robert Bosch GmbH, Abt. K8/EIS,Tübinger Straße 123, 72762 Reutlingen.6th International CAN Conference 2nd to 4th November, Turin (Italy).

[19] CAN Physical Layer for Industrial Applications. CIA Draft Standard 102. 20 April 1994.

[20] CANopen. CAN in automation. www.cia.org.

[21] M J Schofield. The Controller Area Network (CAN). http://www.omegas.co.uk

[22] Ghosh Indradeep (Princeton Univ), Raghunathan Anand , Jha Niraj K. Design for testability technique for RTL circuits using control/data flow extraction. IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers Nov 10-14 1996 1996 Sponsored by: IEEE IEEE p 329-336.

[23] Livani MA, Kaiser J, Jia WJ. Scheduling hard and soft real-time communication in a controller area network. CONTROL ENGINEERING PRACTICE 7: (12) 1515-1523 1999 Dec.

[24] Anon . CANbus brings new thinking to the control of an injection moulding machine. British Plastics and Rubber Nov 1998 MCM Publishing p 13-16 0307-6164.

致谢

98 年夏天我刚刚毕业进入人机工程实验室时绝对不会想到这两年多的经历会给我带来如此大的收获，其间的风风雨雨也是我从未料想到的，如果没有戴老师和谭老师的帮助与鼓励我可能早已退学了。当时的我根本不知道 CAN 为何物，当然不会想到有一天在东风 8B 内燃机车及北京局内燃动车组上会有我设计的产品。希望本产品能作为献给两位老师最好的礼物。

对于自己亲手设计、制造、调试的产品能应用于国家的铁路建设当然感到很高兴，但我认为我的收获并不仅限于此。对于一个硕士研究生要想对科学理论做出什么巨大的贡献恐怕不现实，硕士研究生论文的要求也不在于此。但经过两年多的学习、研究，当他取得了硕士文凭时是否已经具备了独立地进行科研，实践的能力则是最重要的。我想我的收获正在于此。

因此在这里我向这些年来关心、帮助过我的老师，同学及朋友道一声："谢谢"。在此课题进行期间苏师兄树强、焦师兄凤川及同实验室的同学于印泉、查卫翔都直接给予过帮助。而对于徐女士趁宵、王小姐兆娟我真不知是应该致谢还是将她们列为第二、第三作者，在课题进行初期她们都亲身参与过很多设计、实验工作。另外丁同学群及其战友石小姐丽莉在我最困难的时刻能雪中送炭，非常感谢。

写完论文才发现，这世界其实挺好的。

佴晓东

2001 年 1 月

Realization of the CAN Bus Communications System on DF8B Locomotives

NI Xiaodong

Ergonomics Laboratory

Northern Jiaotong University

Abstract:

As the development of industrial control system, the mechatronics system becomes more and more complex. So the demand for the communications system is stricter. The DF8B diesel locomotive is a typical example, in which the traditional single-master, low speed RS232/RS485 serial bus becomes unsuitable for real-time control, because it can not operates at data rates of over 19.2k bps via twisted-pair wire and the data security is not enough. To meet the demand of the new locomotive system the Controller Area Network(CAN) is adopted to reform the communications system of it.

The Controller Area Network (the CAN bus) is a serial communications bus for real-time control applications. CAN operates at data rates of up to 1 Megabits per second and has excellent error detection and confinement capabilities.

CAN was originally developed by the German company Robert Bosch for use in the car industry to provide a cost-effective communications bus for in-car electronics and as alternative to expensive and cumbersome wiring looms. The car industry continues to use CAN for an increasing number of applications, but because of its proven reliability and robustness, CAN is now also being used in many other industrial control applications.

CAN is an international standard and is documented in ISO 11898 (for high-speed applications) and ISO 11519 (for lower-speed applications).

CAN is a serial bus system with multi-master capabilities, that is, all CAN nodes are able to transmit data and several CAN nodes can request the bus simultaneously. The serial bus system with real-time capabilities is the subject of the ISO 11898 international standard and covers the lowest two layers of the ISO/OSI reference model. In CAN networks there is no addressing of subscribers or stations in the conventional sense, but instead, prioritized messages are transmitted. A transmitter sends a message to all CAN nodes (broadcasting). Each node decides on the basis of the identifier received whether it should process the message or not. The identifier also determines the priority that the message enjoys in competition for bus access.

One of the outstanding features of the CAN protocol is its high transmission reliability. The CAN controller registers a station's error and evaluates it statistically in order to take appropriate measures. These may extend to disconnecting the CAN node producing the errors.

The maximum transmission rate is specified as 1 M bit/s. This value applies to networks up to 40 m. For longer distances the data rate must be reduced: for distances up to 500 m a speed of 125 kbit/s is possible, and for transmissions up to 1 km a data rate of 50 kbit/s is permitted.

Adopting CAN as the communications method the communications system of DF8B locomotive becomes a robust, versatile, multi-master and high speed channel for data inter-change, and it has been running successfully for months.

Keywords:

Controller Area Network(CAN), Local Area Network, Industrial Field Bus, Serial Communications, Single-chip Microcomputer, Locomotive

Chapter 1 Introduction

1.1 The Communications System of DF8B Diesel Locomotives

DF8B locomotives have been used on low speed freight trains for a long time. There are only two nodes in the communications system of DF8B, which are a major computer and a display screen. The physical layer of this system is RS232-RS485 data bus. The RS232-RS485 are all single-master serial bus so there is only one master node in this bus—the major computer and the display screen is the slave node. The block diagram of this communications system is shown in figure 1-1.

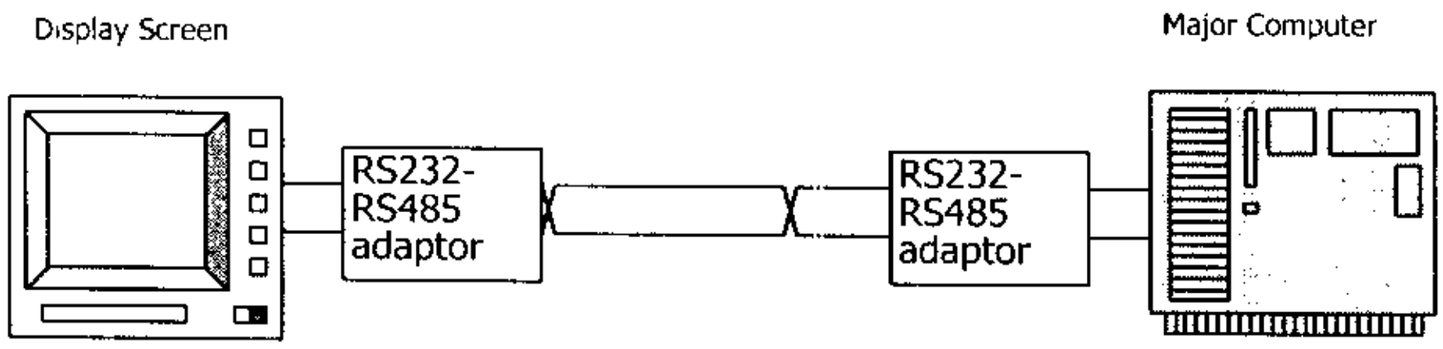


Fig. 1-1 block diagram of RS232-RS485 communications system

When powered on the display screen is assigned an address, with which the master node—major computer—can access the display screen. When the major computer want to send some data to the display screen it will set the address information and data, so the display can receive the data. When the major computer want to receive some data it will send the address and a command, then the display screen will send the acknowledge. This sequence is called hand shaking. After this the display screen will send some data to the major computer.

The main restrictions of this system are:

- Only one node in the network can be the master node.
- The communications speed is low (4800bps).
- The bus length is short when operating at high bit rate.
- Any slave nodes cannot send data to another one directly.
- Only the physical layer of the network is defined （RS232-RS485）, so the network can only detect error at byte level, not frame level.

1.2 Reform the Network Hardware and Software

CAN bus is selected to reform the traditional RS232-RS485 bus. The main objects of this CAN bus system are:

- Long distance network communication (over 400m).
- High bit rate (up to 100k bps).
- High level of data security.
- Multi-master network architecture.

In the following chapters these characters will be introduced respectively.

Chapter 2 Introduction of the Controller Area Network

2.1 What's CAN?

The Controller Area Network (CAN) is a serial bus system especially suited for networking "intelligent" devices as well as sensors and actuators within a system or sub-system.

CAN is a serial bus system with multi-master capabilities, that is, all CAN nodes are able to transmit data and several CAN nodes can request the bus simultaneously. The serial bus system with real-time capabilities is the subject of the ISO 11898 international standard and covers the lowest two layers of the ISO/OSI reference model. In CAN networks there is no addressing of subscribers or stations in the conventional sense, but instead, prioritized messages are transmitted. A transmitter sends a message to all CAN nodes (broadcasting). Each node decides on the basis of the identifier received whether it should process the message or not. The identifier also determines the priority that the message enjoys in competition for bus access.

The use of CAN in most of European passenger cars and the decision by truck and off-road vehicle manufacturers for CAN guarantees the availability of CAN chips for more than 10 years. Other high volume markets, like domestic appliances and industrial control, also increase the CAN sales figures. Up to spring 1999 there are more than 150 million CAN nodes installed[CAN In Automation].

One of the outstanding features of the CAN protocol is its high transmission reliability. The CAN controller registers a stations error and evaluates it statistically in order to take appropriate measures. These may extend to disconnecting the CAN node producing the errors.

The maximum transmission rate is specified as 1 Mbit/s. This value applies to networks up to 40 m. For longer distances the data rate must be reduced: for distances up to 500 m a speed of 125 kbit/s is possible, and for transmissions up to 1 km a data rate of 50 kbit/s is permitted.

2.2 How Does CAN Function

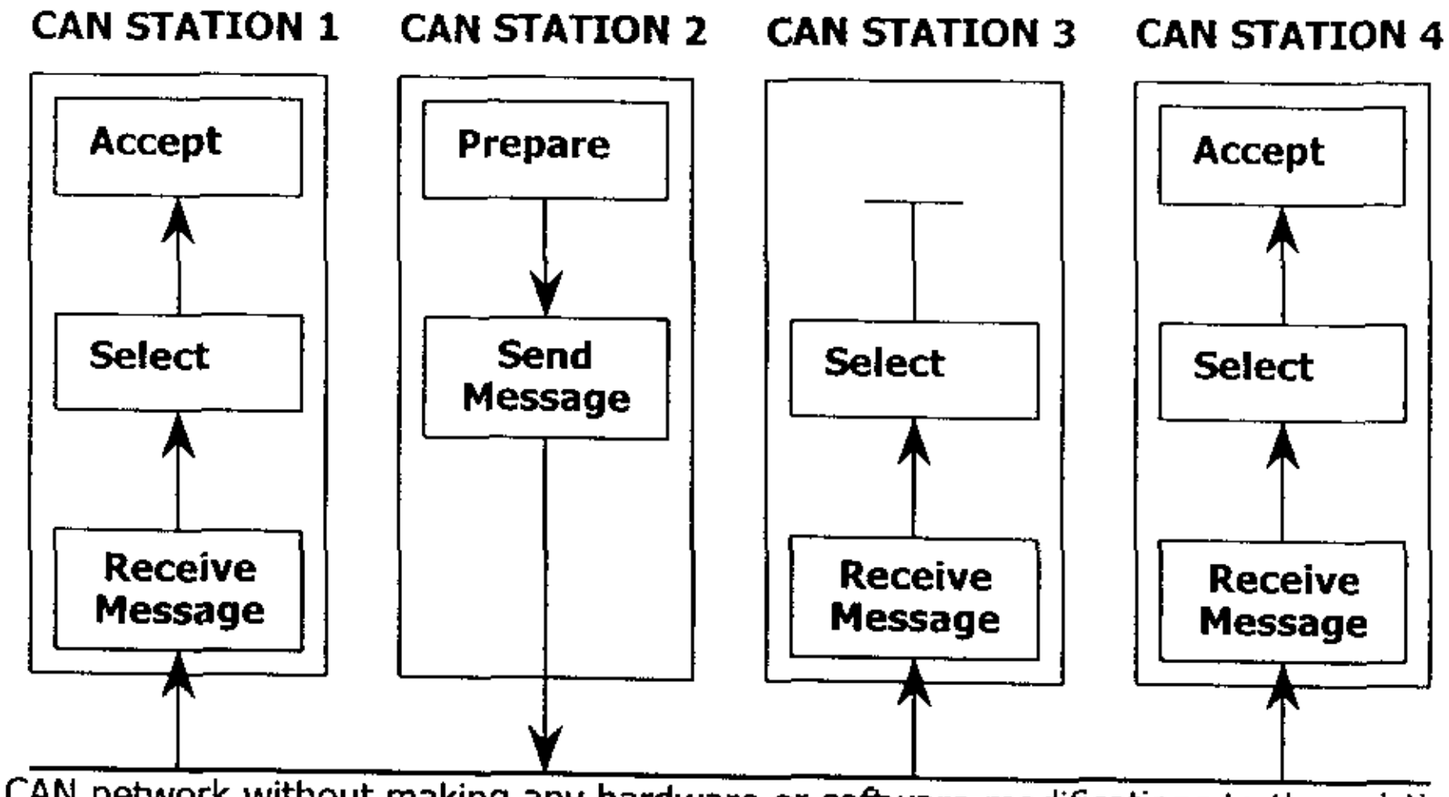
2.2.1 Principles of Data Exchange

When data are transmitted by CAN, no stations are addressed, but instead, the content of the message (e.g. rpm or engine temperature) is designated by an identifier that is unique throughout the network. The identifier defines not only the content but also the priority of the message. This is important for bus allocation when several stations are competing for bus access.

If the CPU of a given station wishes to send a message to one or more stations, it passes the data to be transmitted and their identifiers to the assigned CAN chip ("Make ready"). This is all the CPU has to do: To initiate data exchange. The message is constructed and transmitted by the CAN chip. As soon as the CAN chip receives the bus allocation ("Send Message") all other stations on the CAN network become receivers of this message ("Receive Message"). Each station in the CAN network, having received the message correctly, performs an acceptance test to determine whether the data received are relevant for that station ("Select"). If the data are of significance for the station concerned they are processed ("Accept"), otherwise they are ignored. Figure 2-1 shows the transmission and acceptance diagram.

Fig. 2-1 Broadcast transmissions and acceptance filtering by CAN nodes

A high degree of system and configuration flexibility is achieved as a result of the content-oriented addressing scheme. It is very easy to add stations to the existing



CAN network without making any hardware or software modifications to the existing stations, provided that the new stations are purely receivers. Because the data transmission protocol does not require physical destination addresses for the individual components, it supports the concept of modular electronics and also permits multiple reception (broadcast, multicast) and the synchronization of distributed processes: measurements needed as information by several controllers can be transmitted via the network, in such a way that it is unnecessary for each controller to have its own sensor.

2.2.2 Non-destructive Bitwise Arbitration

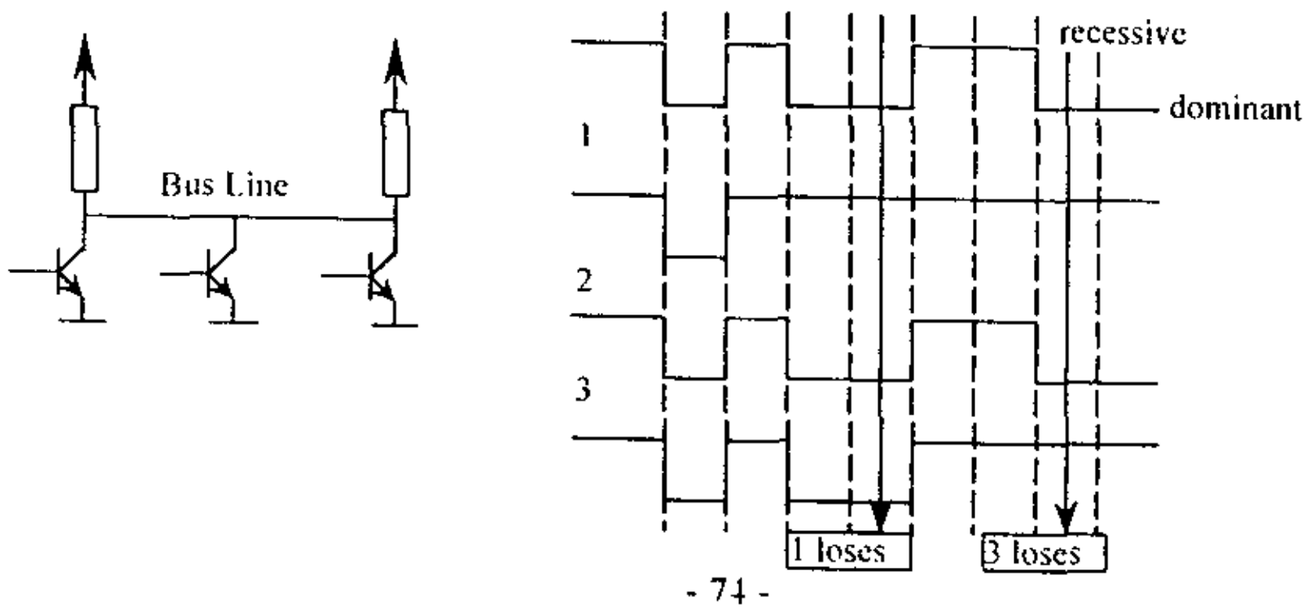


Fig. 2-2 Principle of non-destructive bitwise arbitration

For the data to be processed in real time they must be transmitted rapidly. This not only requires a physical data transfer path with up to 1 Mbit/s but also calls for rapid bus allocation when several stations wish to send messages simultaneously.

In real-time processing the urgency of messages to be exchanged over the network can differ greatly: a rapidly changing dimension (e.g. engine load) has to be transmitted more frequently and therefore with less delays than other dimensions (e.g. engine temperature) which change relatively slowly.

The priority at which a message is transmitted compared with another less urgent message is specified by the identifier of the message concerned. The priorities are laid down during system design in the form of corresponding binary values and cannot be changed dynamically. The identifier with the lowest binary number has the highest priority.

Bus access conflicts are resolved by bitwise arbitration on the identifiers involved by each station observing the bus level bit for bit. In accordance with the "wired and" mechanism, by which the dominant state (logical 0) overwrites the recessive state (logical 1), the competition for bus allocation is lost by all those stations with recessive transmission and dominant observation. All "losers" automatically become receivers of the message with the highest priority and do not re-attempt transmission until the bus is available again. In figure 2-2 this principle is showed.

2.2.3 Efficiency of Bus Allocation

The efficiency of the bus allocation system is determined mainly by the possible applications for a serial bus system. In order to judge as simply as possibly which bus systems are suitable for which applications the literature includes a method of classifying bus allocation procedures. Generally we distinguish between the following classes:

- Bus allocation on a fixed time schedule

  Allocation is made sequentially to each participant for a maximum duration regardless of whether this participant needs the bus at this moment or not (examples: token slot or token passing).

- Bus allocation on the basis of need

  The bus is allocated to one participant on the basis of transmission requests outstanding, i.e. the allocation system only considers participants wishing to transmit (examples: CSMA, CSMA/CD, round robin or bitwise arbitration).

For CAN, bus allocation is negotiated purely among the messages waiting to be transmitted. This means that the procedure specified by CAN is classified as allocation on the basis of need.

Another means of assessing the efficiency of bus arbitration systems is the bus access method:

- Non-destructive bus access

With methods of this type the bus is allocated to one and only one station either immediately or within a specified time following a single bus access (by one or more stations). This ensures that each bus access by one or more stations leads to an unambiguous bus allocation (examples: token slot, token passing, round robin, bitwise arbitration).

- Destructive bus allocation

Simultaneous bus access by more than one station causes all transmission attempts to be aborted and therefore there is no successful bus allocation. More than one bus access may be necessary in order to allocate the bus at all, the number of attempts before bus allocation is successful being a purely statistical quantity (examples: CSMA/CD, Ethernet).

In order to process all transmission requests of a CAN network while complying with latency constraints at as low a data transfer rate as possible, the CAN protocol must implement a bus allocation method that guarantees that there is always unambiguous bus allocation even when there are simultaneous bus accesses from different stations. The method of bitwise arbitration using the identifier of the messages to be transmitted uniquely resolves any collision between a number of stations wanting to transmit, and it does this at the latest within 13 (standard format) or 33 (extended format) bit periods for any bus access period. Unlike the message-wise arbitration employed by the CSMA/CD method this non-destructive method of conflict resolution ensures that no bus capacity is used without transmitting useful information.

Even in situations where the bus is overloaded the linkage of the bus access priority to the content of the message proves to be a beneficial system attribute compared with existing CSMA/CD or token protocols: In spite of the insufficient bus transport capacity, all outstanding transmission requests are processed in order of their importance to the overall system (as determined by the message priority). The available transmission capacity is utilized efficiently for the transmission of useful data since "gaps" in bus allocation are kept very small. The collapse of the whole transmission system due to overload, as can occur with the CSMA/CD protocol, is not possible with CAN. Thus, CAN permits implementation of fast, traffic-dependent bus access which is non-destructive because of bitwise arbitration based on the message priority employed.

Non-destructive bus access can be further classified into

- centralized bus access control
- decentralized bus access control

depending on whether the control mechanisms are present in the system only once (centralized) or more than once (decentralized). A communication system with a designated station must provide a strategy to take effect in the event of a failure of the master station. This concept has the disadvantage that the strategy for failure management is difficult and costly to implement and also that the takeover of the central station by a redundant station can be very time-consuming. For these reasons and to circumvent the problem of the reliability of the master station (and thus of the whole communication system), the CAN protocol implements decentralized bus control. All major communication mechanisms, including bus access control, are implemented several times in the system, because this is the only way to fulfill the high requirements for the availability of the communication system.

In summary it can be said that CAN implements a traffic-dependent bus allocation system that permits, by means of a non-destructive bus access with decentralized bus access control, a high useful data rate at the lowest possible bus data rate in terms of the bus busy rate for all stations. The efficiency of the bus arbitration procedure is increased by the fact that the bus is utilized only by those stations with pending transmission requests.

These requests are handled in the order of the importance of the messages for the system as a whole. This proves especially advantageous in overload situations. Since bus access is prioritized on the basis of the messages, it is possible to guarantee low individual latency times in real-time systems.

2.2.4 Message Frame Formats

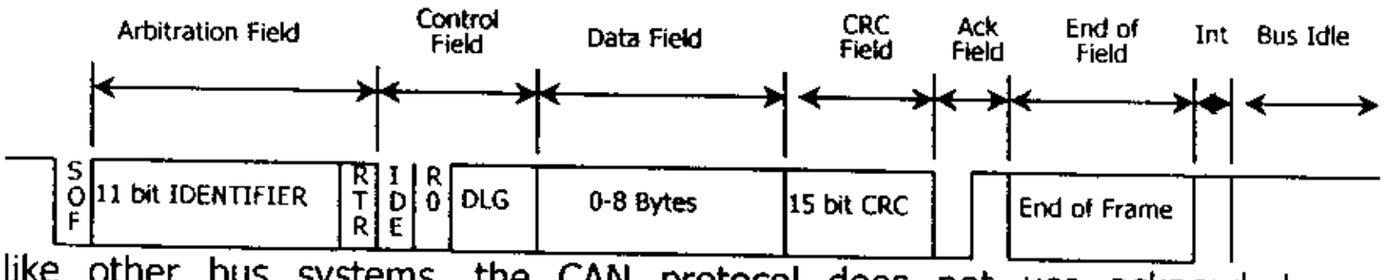
The CAN protocol supports two message frame formats, the only essential difference being in the length of the identifier (ID). In the standard format the length of the ID is 11 bits and in the extended format the length is 29 bits. The message frame for transmitting messages on the bus comprises seven main fields. Figure 2-3 shows the standard format message.

A message in the standard format begins with the start bit "start of frame", this is followed by the "arbitration field", which contains the identifier and the "RTR" (remote transmission request) bit, which indicates whether it is a data frame or a request frame without any data bytes (remote frame). The "control field" contains the IDE (identifier extension) bit, which indicates either standard format or extended format, a bit reserved for future extensions and - in the last 4 bits - a count of the data bytes in the data field.

The "data field" ranges from 0 to 8 bytes in length and is followed by the "CRC field", which is used as a frame security check for detecting bit errors. The "ACK field" comprises the ACK slot (1 bit) and the ACK delimiter (1 recessive bit). The bit in the ACK slot is sent as a recessive bit and is overwritten as a dominant bit by those receivers which have at this time received the data correctly (positive acknowledgement). The receivers regardless of the result of the acceptance test acknowledge correct messages. The end of the message is indicated by "end of frame". "Intermission" is the minimum number of bit periods separating consecutive messages. If there is no following bus access by any station, the bus remains idle ("bus idle"). For more detailed information about CAN message formats see Chapter 4.

Fig 2-3 Message frame for standard format (CAN Specification 2.0A)

2.2.5 Detecting and Signaling Errors



Unlike other bus systems, the CAN protocol does not use acknowledgement messages but instead signals any errors that occur. For error detection the CAN protocol implements three mechanisms at the message level:

- Cyclic Redundancy Check (CRC)

The CRC safeguards the information in the frame by adding redundant check bits at the transmission end. At the receiver end these bits are re-computed and tested against the received bits. If they do not agree there has been a CRC error.

- Frame check

This mechanism verifies the structure of the transmitted frame by checking the bit fields against the fixed format and the frame size. Errors detected by frame checks are designated "format errors".

- ACK errors

As mentioned above, frames received are acknowledged by all recipients through positive acknowledgement. If no acknowledgement is received by the transmitter of the message (ACK error) this may mean that there is a transmission error which has been detected only by the recipients, that the ACK field has been corrupted or that there are no receivers.

The CAN protocol also implements two mechanisms for error detection at the bit level:

- Monitoring

The ability of the transmitter to detect errors is based on the monitoring of bus signals: each node which transmits also observes the bus level and thus detects differences between the bit sent and the bit received. This permits reliable detection of all global errors and errors local to the transmitter.

- Bit stuffing

The coding of the individual bits is tested at bit level. The bit representation used by CAN is NRZ (non-return-to-zero) coding, which guarantees maximum efficiency in bit coding. The synchronization edges are generated by means of bit stuffing, i.e. after five consecutive equal bits the sender inserts into the bit stream a stuff bit with the complementary value, which is removed by the receivers. The code check is limited to checking adherence to the stuffing rule.

If one or more errors are discovered by at least one station (any station) using the above mechanisms, the current transmission is aborted by sending an "error flag". This prevents other stations accepting the message and thus ensures the consistency of data throughout the network.

After transmission of an erroneous message has been aborted, the sender automatically re-attempts transmission (automatic repeat request). There may again be competition for bus allocation. As a rule, retransmission will be begun within 23 bit periods after error detection; in special cases the system recovery time is 31 bit periods.

However effective and efficient the method described may be, in the event of a defective station it might lead to all messages (including correct ones) being aborted, thus blocking the bus system if no measures for self-monitoring were taken. The CAN protocol therefore provides a mechanism for distinguishing sporadic errors from permanent errors and localizing station failures (fault confinement).

This is done by statistical assessment of station error situations with the aim of recognizing a station's own defects and possibly entering an operating mode where the rest of the CAN network is not negatively affected. This may go as far as the

station switching itself off to prevent messages erroneously recognized as incorrect from being aborted.

2.2.6 Data Reliability of the CAN Protocol

The introduction of safety-related systems in automobiles brought with it high requirements for the reliability of data transmission. The objective is frequently formulated as not permitting any dangerous situations for the driver to occur as a result of data exchange throughout the whole life of a vehicle.

This goal is achieved if the reliability of the data is sufficiently high or the residual error probability is sufficiently low. In the context of bus systems data reliability is understood as the capability to identify data corrupted by transmission faults. The residual error probability is a statistical measure of the impairment of data reliability: It specifies the probability that data will be corrupted and that this corruption will remain undetected. The residual error probability should be so small that on average no corrupted data will go undetected throughout the whole life of a system.

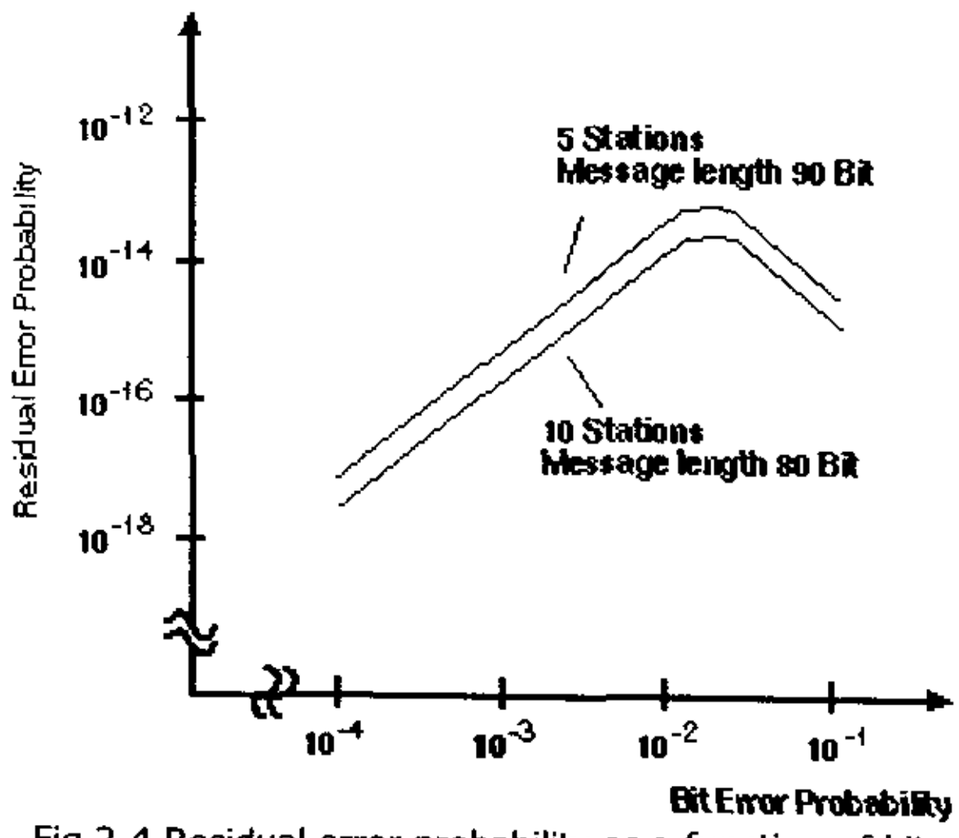


Fig 2-4 Residual error probability as a function of bit error probability

Calculation of the residual error probability requires that the errors which occur be classified and that the whole transmission path be described by a model. If we determine the residual error probability of CAN as a function of the bit error probability for message lengths of 80 to 90 bits, for system configurations of, for instance, five or ten nodes and with an error rate of 1/1000 (an error in one message in every thousand), then maximum bit error probability is approximately 0.02 - on the order of 10^{-13}. Based on this it is possible to calculate the maximum number of undetectable errors for a given CAN network.

For example, if a CAN network operates at a data rate of 1 Mbit/s, at an average bus capacity utilization of 50 percent, for a total operating life of 4000 hours and with an average message length of 80 bits, then the total number of messages transmitted is 9×10^{10}. The statistical number of undetected transmission errors during the operating life is thus on the order of less than 10^{-2}. Or to put it another way, with an operating time of eight hours per day on 365 days per year and an error rate of 0.7 s, one undetected error occurs every thousand years (statistical average). Figure 2-4 shows the residual error probability[21].

2.2.7 Extended Format CAN Message

The SAE "Truck and Bus" subcommittee standardized signals and messages as well as data transmission protocols for various data rates. It became apparent that standardization of this kind is easier to implement when a longer identification field is available.

To support these efforts, the CAN protocol was extended by the introduction of a 29-bit identifier. This identifier is made up of the existing 11-bit identifier (base ID) and an 18-bit extension (ID extension). Thus the CAN protocol allows the use of two message formats: StandardCAN (Version 2.0A) and ExtendedCAN (Version 2.0B). As the two formats have to coexist on one bus it is laid down which message has higher priority on the bus in the case of bus access collisions with differing formats and the same base identifier: The message in standard always has priority over the message in extended format.

CAN controllers which support the messages in extended format can also send and receive messages in standard format. When CAN controllers which only cover the standard format (Version 2.0A) are used on one network, then only messages in standard format can be transmitted on the entire network. Messages in extended format would be misunderstood. However there are CAN controllers which only support standard format but recognize messages in extended format and ignore them (Version 2.0B passive).

The distinction between standard format and extended format is made using the IDE bit (Identifier Extension Bit) which is transmitted as dominant in the case of a frame in standard format. For frames in extended format it is recessive.

The RTR bit is transmitted dominant or recessive depending on whether data are being transmitted or whether a specific message is being requested from a station. In place of the RTR bit in standard format the SRR (substitute remote request) bit is transmitted for frames with extended ID. The SRR bit is always transmitted as recessive, to ensure that in the case of arbitration the standard frame always has priority bus allocation over an extended frame when both messages have the same base identifier.

Unlike the standard format, in the extended format the IDE bit is followed by the 18-bit ID extension, the RTR bit and a reserved bit (r1).

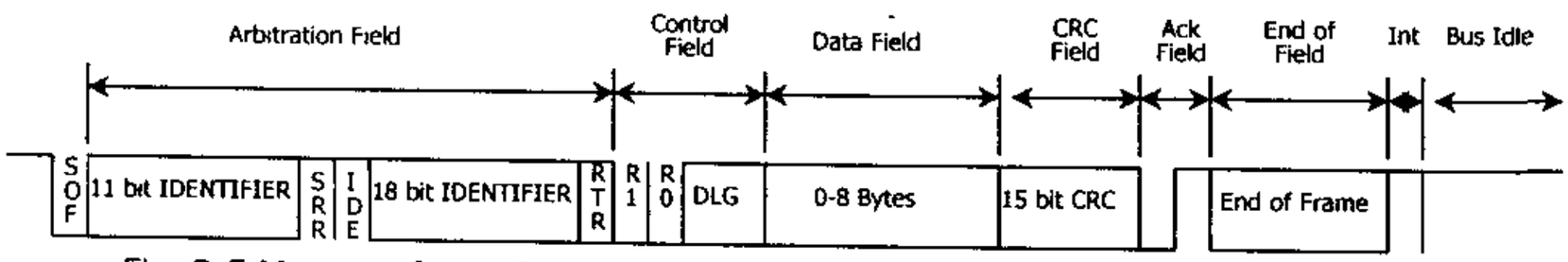


Fig. 2-5 Message frame for extended format (CAN specification 2.0B)

All the following fields are identical with standard format. Conformity between the two formats is ensured by the fact that the CAN controllers which support the extended format can also communicate in standard format.

2.3 Physical CAN Connection

The data rates (up to 1 Mbit/s) necessitate a sufficiently steep pulse slope, which can be implemented only by using power elements. A number of physical

connections are basically possible. Figure 2-6 shows the block diagram of a standard CAN bus application system.

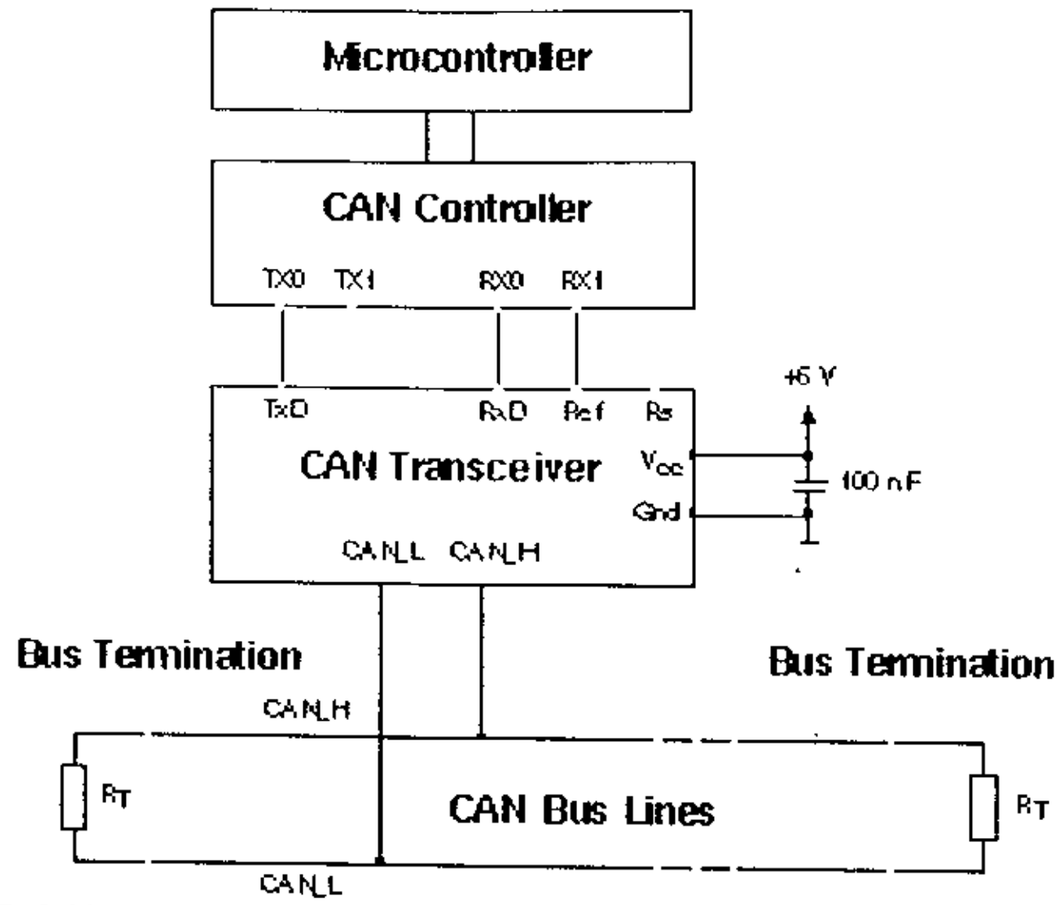


Fig 2-6 Physical CAN Connection according to ISO 11898

Chapter 3 Hardware Architecture of the CAN Network

3.1 Overview

This chapter is intended to provide basic technical information for the implementation of the Physical Medium Attachment in a CAN network according to the ISO 11898 standard, using a package of a **Module controller (host controller)**, a **CAN controller (protocol controller)** and a **transceiver** product. This application supports bit rates up to 1 Mbit/s over a two-wire differential bus line, which is the transmission medium being specified by the ISO 11898 standard.

3.2 Objectives of This Application

As what is shown in Chapter 1, the traditional centralized bus access control network system of DF8B locomotive becomes an obstacle for implementing distributed computer control. If a decentralized network system can replace this one the control system will be flexible. But it is hard to delete the entire traditional network and replace a new one, because most of the software to implement the control mechanism and control the communications method cannot be replaced. So the possible way is to reform the traditional network, make minimum changes and suit the hardware independent software.

The objectives of this application is:

- Install a decentralized local network communications system
- Only reform the network interlink hardware and hardware dependent software
- Suit the demand of the traditional hardware independent software.

So the most important task of this application is to build an "intelligent network card" that implement most work of protocol transferring, error detecting and confinement, task scheduling etc. The major tasks of the intelligent network card are:

- Initializing hardware and build communications link
- Transferring CAN protocol with the traditional protocol
- Detecting protocol error and make corresponding action
- Scheduling tasks
- Buffering TX and RX data

3.3 Network Characteristics and Wiring Topology

3.3.1 Network Characteristics Outline

To suit the demand of the DF8B locomotive communications system, the network are made up 2 Local Networks (CAN bus). In each one every node can send/receive message with any other node directly. The basic characteristics are listed below:

- Bus length (In the locomotive): <30m
- Bus length (Locomotive interconnection): <400m
- CAN bus bit rate: 100k bit per second
- RS232 bit rate: 115.2k bit per second
- Number of CAN node (In the locomotive): 7
- Protocol: CAN 2.0A
- Network card module controller: 89c52(22.1184MHz)
- CPU of Industrial PC (Display Screen): 80386sx(33MHz)

3.3.2 Network Wiring Topology

7 CAN nodes are installed on every locomotive and make up of the CAN network. The two locomotives are interlinked with a special CAN node called "Network Card". Other nodes are Main Control Computer, Logical Unit, and Driver's Display Screen etc. These components are showed in figure 3-1.

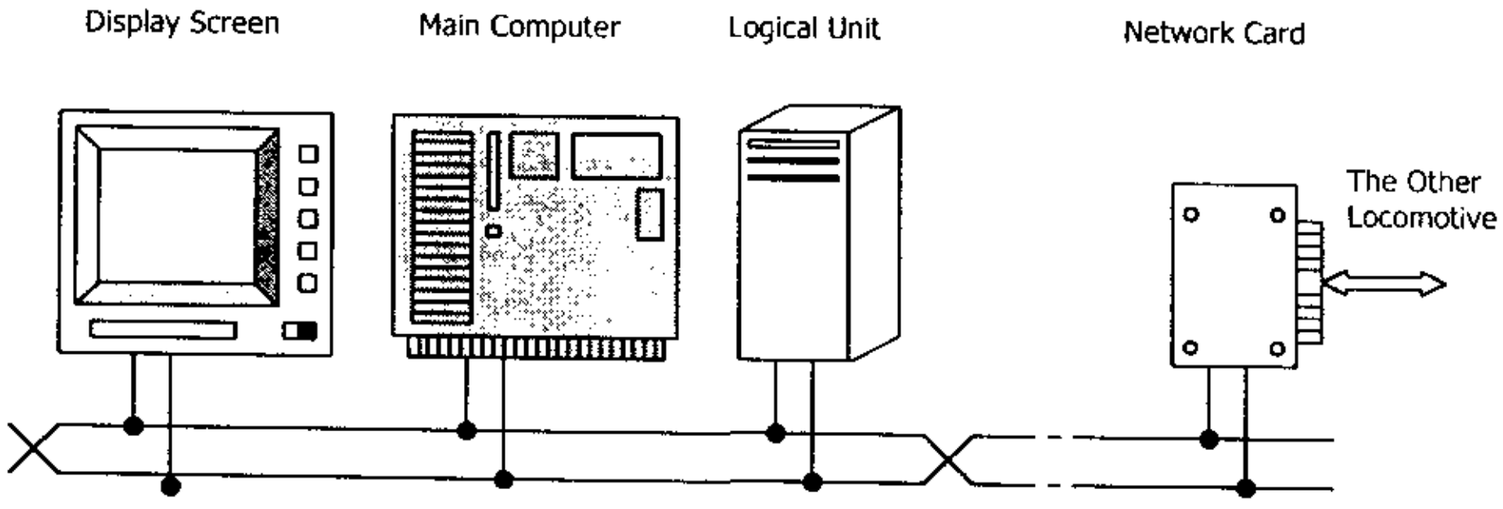


Fig. 3-1 Network Topology

3.4 CAN Node Architecture

In this application each CAN module can be divided into different functional blocks. The following part focuses on illustrating the hardware architecture of the Driver's Display Screen node.

The connection to the CAN bus lines of the node is usually built with a **CAN Transceiver** optimized for the applications [6], [7]. The transceiver controls the logic level signals from the CAN controller into the physical levels on the bus.

The next upper level is a **CAN Controller** which implements the complete CAN protocol defined in the CAN Specification [2]. It also covers message buffering and acceptance filtering.

All these CAN functions are controlled by a **Module Controller** which performs the functionality of the application. For example, it controls actuators, reads sensors and handles the man-machine interface (MMI). The module controller of this application is 89c52 single-chip micro-controller.

This architecture complies with the ISO 11898 [1] standard.

ISO 11898 [1] is the international standard for in-vehicle high-speed communication using the Controller Area Network (CAN) bus protocol. The scope of this standard essentially is to specify the so-called data link layer and physical layer of the communication link. The physical layer is subdivided into three sublayers as shown in Fig. 3-2.

These are

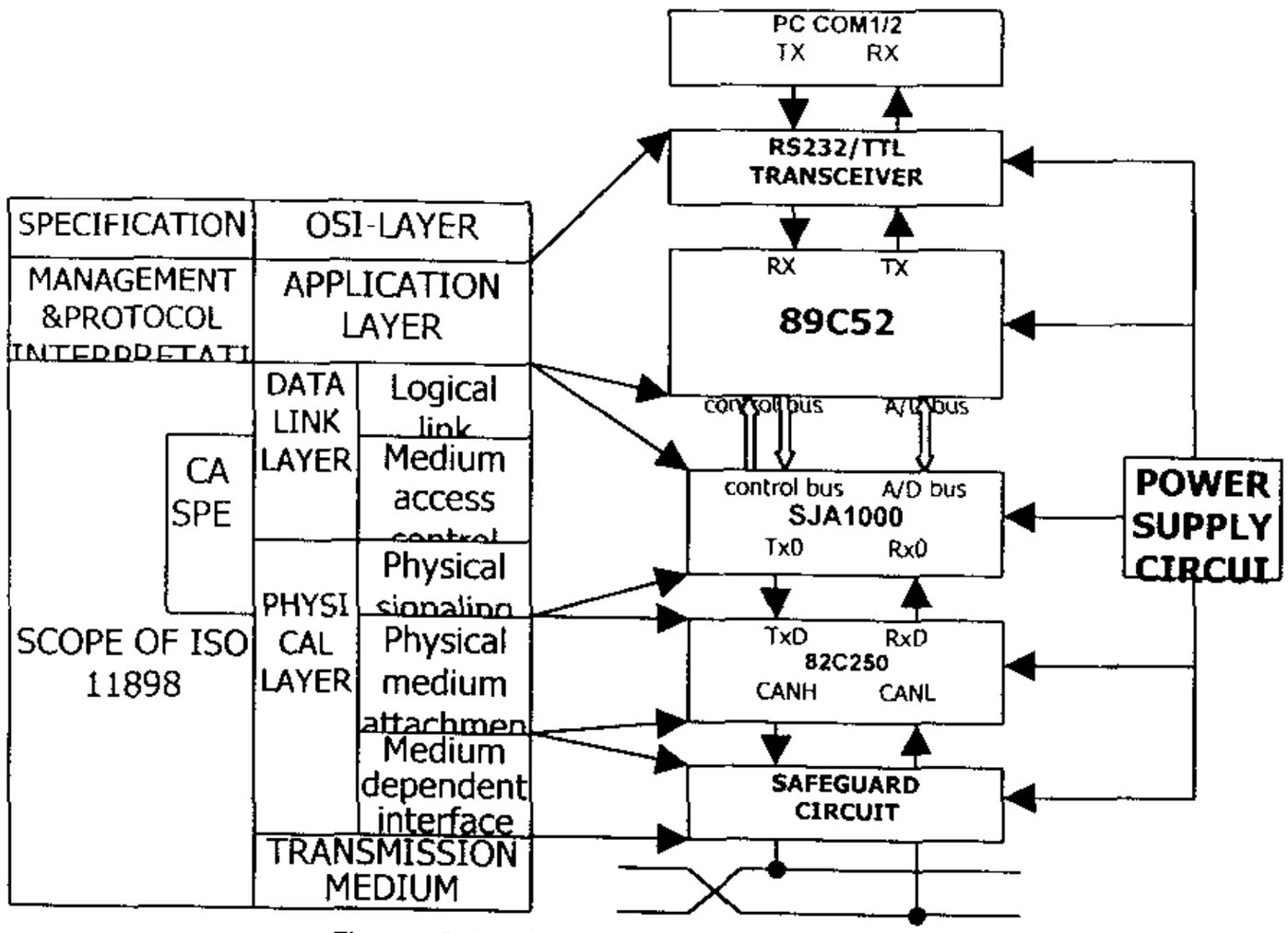


Figure 3-2 CAN node architecture

- Physical Signaling bit coding, timing and synchronization
- Physical Medium Attachment driver and receiver characteristics
- Medium Dependent Interface bus connector

As shown in Fig. 3-2 the Physical Medium Attachment sublayer of this CAN node is implemented with the transceivers PCA82C250 [7] or PCA82C251 [8] from Philips Semiconductors. The implementation of the Physical Signaling sublayer and the Data Link Layer is performed by integrated protocol controller products, SJA1000 from Philips Semiconductors. Connection to the transmission medium is provided via the Medium Dependent Interface i.e. a connector used to attach bus nodes to the bus line.

3.5 Physical Layer

3.5.1 Chip Outline

The 82C250 transceiver products basically provide interfacing between a protocol controller and a physical transmission line. They are designed to transmit data with a bit rate of up to 1 Mbit/s over a two-wire differential voltage bus line as described in the ISO 11898 standard. Their general features are listed in the data sheets (see [7] and [8]). The pining description is shown in table 3-1.

SYMBOL	PIN	DESCRIPTION

TXD	1	Transmit data input
GND	2	Ground
Vcc	3	Supply voltage
RXD	4	Receive data output
Vref	5	Reference voltage output
CANL	6	LOW-level CAN voltage input/output
CANH	7	HIGH-level CAN voltage input/output
Rs	8	Slope resistor input

TABLE 3-1 PINING OF 82C250

3.5.2 Functional Description

The PCA82C250, which implements the physical layer, is the interface between the CAN protocol controller and the physical bus. It is primarily intended for high-speed applications (up to 1 M baud) in cars. The device provides differential transmit capability to the bus and differential receive capability to the CAN controller. It is fully compatible with the "ISO 11898" standard.

If the junction temperature exceeds a value of approximately 160 °C, the limiting current of both transmitter outputs is decreased. Because the transmitter is responsible for the major part of the power dissipation, this will result in a reduced power dissipation and hence a lower chip temperature. All other parts of the IC will remain in operation. The thermal protection is particularly needed when a bus line is short-circuited.

The CANH and CANL lines are also protected against electrical transients, which may occur in an automotive environment.

Slope control

Pin 8 (Rs) allows three different modes of operation to be selected: high-speed, slope control or standby.

- For high-speed operation, the transmitter output transistors are simply switched on and off as fast as possible. In this mode, no measures are taken to limit the rise and fall slope. The high-speed mode is selected by connecting pin 8 to ground.
- For lower speeds or shorter bus length, an unshielded twisted pair or a parallel pair of wires can be used for the bus. To reduce RFI, the rise and fall slope should be limited. The rise and fall slope can be programmed with a resistor connected from pin 8 to ground. The slope is proportional to the current output at pin 8.
- If a HIGH level is applied to pin 8, the circuit enters a low current standby mode. In this mode, the transmitter is switched off and the receiver is switched to a low current. If dominant bits are detected (differential bus voltage >0.9 V), RXD will be switched to a LOW level. The microcontroller should react to this condition by switching the transceiver back to normal operation (via pin 8). Because the receiver is slow in standby mode, the first message will be lost.

In this application the high-speed mode is adopt.

3.5.3 CAN Bus Interface

Various couple methods can be selected to protect each CAN nodes from electrical destroying and meet the demand of high bus speed. In this application a set of safeguard circuit is occupied at the end of CAN node. After the practical/lab tests, it can be concluded that this circuit can prevent high voltage damage (<18V) and suit the demand of high speed bus transmission (>100K bps).

The CAN interface is an IDC10 male plug. The signals of it are defined in table 3-2.

PIN	1	2	3	4	5	6	7	8	9	10
Signa	NC	CANL	GND	NC	NC	GND	CANH	NC	+5V	NC

Table 3-2 CAN Bus interface

3.6 Data Link Layer

3.6.1 Chip Outline

The SJA1000 is a stand-alone CAN Controller product which fulfill the CAN protocol 2.0A and 2.0B.

The stand-alone CAN controller SJA1000 has two different Modes of Operation:

BasicCAN Mode: implements the CAN protocol 2.0A

PeliCAN Mode: implements the CAN protocol 2.0B

Upon Power-up the BasicCan Mode is the default mode of operation, and it is also the operating mode of this application. The PeliCAN mode is also supported in this firmware.

3.6.2 Block Diagram

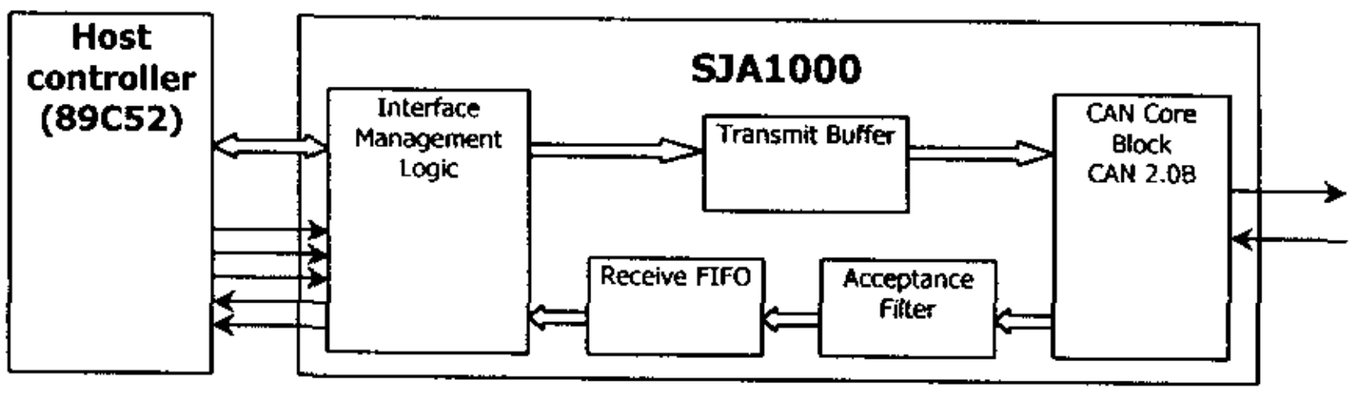


Fig. 3-3 Block Diagram of SJA1000

The figure 3-3 shows the block diagram of the SJA1000.

The **CAN Core Block** controls the transmission and reception of CAN frames according to the CAN specification.

The **Interface Management Logic** block performs a link to the external host controller (89c52). Every register access via the SJA1000 multiplexed address/data bus and controlling of the read/write strobes is handled in this unit. Additionally to the BasicCAN functions known from the PCA82C200, new PeliCAN features have been added. As a consequence of this, additional registers and logic have been implemented mainly in this block.

The **Transmit Buffer** of the SJA1000 is able to store one complete message (Extended or Standard). Whenever a transmission is initiated by the host controller

the Interface Management Logic forces the CAN Core Block to read the CAN message from the Transmit Buffer.

When receiving a message, the CAN Core Block converts the serial bit stream into parallel data for the **Acceptance Filte**r. With this programmable filter the SJA1000 decides which messages the host controller actually receives.

All received messages accepted by the acceptance filter are stored within a **Receive FIFO**. Depending on the mode of operation and the data length up to 32 messages can be stored. This enables the user to be more flexible when specifying interrupt services and interrupt priorities for the system because the probability of data overrun conditions is reduced extremely.

For connection to the host controller, the SJA1000 provides a multiplexed address/data bus and additional read/write control signals. The SJA1000 could be seen as a peripheral memory mapped I/O device for the host controller.

Acceptance Filter

The stand-alone CAN controller SJA1000 is equipped with a versatile acceptance filter, which allows an automatic check of the identifier and data bytes. Using these effective filtering methods, messages or a group of messages not valid for a certain node can be prevented from being stored in the Receive Buffer. Thus it is possible to reduce the processing load of the host controller.

The filter is controlled by the acceptance code and mask registers according to the algorithms given in the data sheet [6]. The received data is compared bitwise with the value contained in the Acceptance Code register. The Acceptance Mask Register defines the bit positions, which are relevant for the comparison (0 = relevant, 1 = not relevant). For accepting a message all **relevant** received bits have to match the respective bits in the Acceptance Code Register.

Acceptance Filtering in BasicCAN Mode

The filter is controlled by two 8-bit wide registers – Acceptance Code Register (ACR) and Acceptance Mask Register (AMR). The 8 most significant bits of the identifier of the CAN message are compared to the values contained in these registers, see also the example. Thus always groups of eight identifiers can be defined to be accepted for any node.

Example:

The Acceptance Code register (ACR) contains: (msb) 11010010
The Acceptance Mask register (AMR) contains: 00111000
Messages with the following 11-bit identifiers are accepted: 11xxx010xxx

With this facility each node can group messages sent to different ones on the locomotive.

3.7 Application Layer

3.7.1 Main Tasks of This Layer

The application layer in this application is implemented with 89C52 single-chip microcontroller. It connects the CAN controller and the PC which is utilized to display the working status of the locomotive and send drivers command via keyboard on it.

The main tasks of this layer are:

- Initialization of CAN controller
- Interpretation of CAN protocol message
- Response to the command/request or error message of CAN controller
- Managing the data/command buffer of 89c52
- Controlling the process sequence
- Connecting to the standard RS232 interface

3.7.2 Data Buffer Diagram

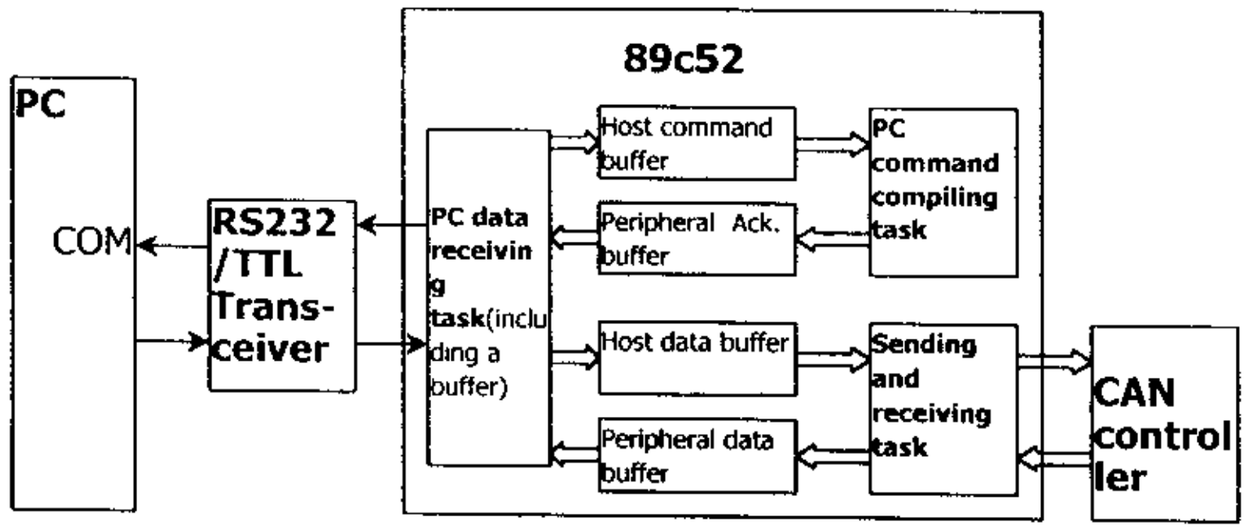


Fig. 3-4 Buffer blocks Diagram

When bootin g up the 89c52 initializes its registers, installs interrupts, and creates data/command buffers. Then it initializes the CAN controller (SJA1000), and enters the normal running phase.

In this phase the microcontroller must response to the data/command request of the CAN controller as well as the PC, so it is useful to create different buffers for all of these data. The definition of these data/command is:

- Host command: command from the host PC
- Peripheral Acknowledgement: 89c52's response data to the host command
- Host data: data from the host PC and will be transmitted to the CAN controller
- Peripheral data: data from the CAN controller and will be transmitted to the host PC

According to this definition the buffer blocks is shown in figure 3-4.

3.7.3 RS232 Interface

The RS232 interface connected to the PC serial port is a DB9 female plug. The signals of it are defined in table 3-3.

PIN	1	2	3	4	5	6	7	8	9
Signa	DCD	RXD	TXD	DTR	GND	DSR	RTS	CTX	RI

Table 3-3 RS232 interface

Chapter 4 Network Configuration

4.1 Introduction

A successful network depends on not only excellent hardware components but also suitable software program and correct network configuration. The hardware safeguard circuit as well as other protection circuit/chips illustrated in Chapter 3 can ensure hardware robustness. In Chapter 5 suitable software algorisms will be introduced. This Chapter will discusses network configuration and related issues.

4.2 Main Subjects

Even if minor errors in the configuration of the CAN bus parameters do not result in immediate failure, the performance of a CAN network can be reduced significantly. The parameters and factors that affect the running of the network are:

1. Bus length and type of cable
2. Bus branch length
3. Node number on the bus
4. Bit timing parameters
5. Communications speed
6. Electro magnetic noise and other factors

Practical tests and calculation of parameters cannot replace with each other. In this Chapter the first 5 factors will be discussed with calculation method and/or test method. The last factor is beyond the scope of this dissertation.

4.3 Maximum Bus Line Length and Maximum Number of Nodes

The maximum achievable bus line length in a CAN bus network is determined essentially by the following physical effects:

1. The loop delays of the connected bus nodes (CAN controller, transceiver etc.) and the delay of the bus line
2. The differences in bit time quantum length due to the relative oscillator tolerance between nodes
3. The signal amplitude drop due to the series resistance of the bus cable and the input resistance of bus nodes

The effect 3 is discussed below. The effects 1 and 2 are discussed in 4.4.

The CAN bus cable length within the locomotive is just 30m and only 7 nodes are connected to the CAN bus, so it must meet the demand. In the following part only the interconnection bus between locomotives are discussed.

The ISO 11898 Standard [1] assumes the network wiring topology to be close to a single line structure in order to minimize reflection effects on the bus line.

At static conditions the differential input voltage at a bus node is determined by the current flowing through the differential input resistance of that node. In case of a dominant bit the output transistors of the transmitting node are switched on, causing a current flow, whereas the transistors are switched off for a recessive bit.

Thus the generated differential voltage at the input of a node ($V_{diff.in}$) depends on (see Figure 4-1)

- The differential output voltage of the transmitting nodes ($V_{diff.out}$)
- The resistance of the bus cable
- The differential input resistance of receiving nodes (R_{diff})

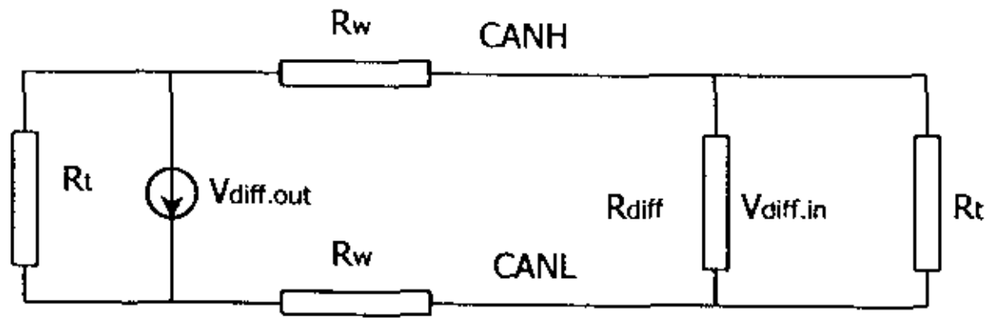


Fig. 4-1 calculation circuit module

R_t: termination resisters

R_w: cable resisters

For this case the differential input voltage at the receiving node is calculated using Fig. 4-1

$$V_{diff.out} = V_{diff.in} + 2 * R_w * I_w \qquad (1)$$

$$Iw = \frac{Vdiff.out}{2Rw} + \frac{Rdiff * Rt}{Rdiff + Rt} \qquad (2)$$

($V_{diff.in}$>1.0V, $V_{diff.out}$=1.5V, R_{diff}>20kΩ, R_t=124Ω)

Cable: ISO 11898(automotive) 0.25mm^2 ρ=70Ω/km

Based on the equation (1), (2) the length of the cable can be calculate as:

L<442m

The actual cable length is 400m(In most cases the length less than 100m), so the cable length requirement can be satisfied.

4.3 Bit Timing Parameters

In many cases, the CAN bit synchronization will amend a faulty configuration of the CAN bit timing to such a degree that only occasionally an error frame is generated.

In the case of arbitration however, when two or more CAN nodes simultaneously try to transmit a frame, a misplaced sample point may cause one of the transmitters to become error passive. The analysis of such sporadic errors requires a detailed knowledge of the CAN bit synchronization inside a CAN node and of the CAN nodes' interaction on the CAN bus.

The purpose of this part is not to introduce the detailed steps to calculate the CAN bus bit timing parameters (Refer to [18] for more information), but describe the CAN bit synchronization algorithm and the parameters which have to be considered for the proper calculation of the CAN bit time.

4.3.1 CAN Protocol Bit Time

CAN supports bit rates in the range of lower than 5 kBit/s up to 1000 kBit/s. Each member of the CAN network has its own clock generator, usually a quartz oscillator. The timing parameter of the bit time (i.e. the reciprocal of the bit rate) can be configured individually for each CAN node, creating a common bit rate even though the CAN nodes' oscillator periods (f_{osc}) may be different.

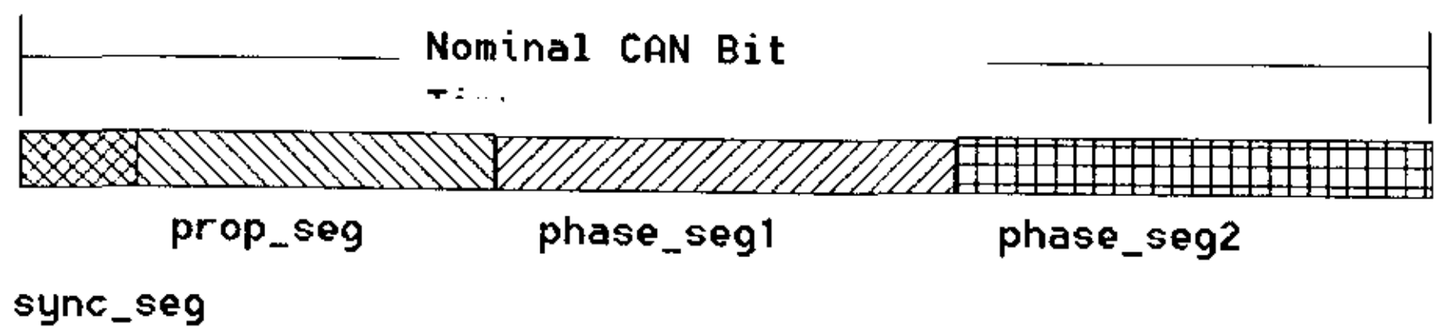


Fig. 4-2 Bit Timing

The frequencies of these oscillators are not absolutely stable, small variations are caused by changes in temperature or voltage and by deteriorating components. As long as the variations remain inside a specific oscillator tolerance range, the CAN nodes are able to compensate for the different bit rates by resynchronizing to the bit stream.

According to the CAN specification, the bit time is divided into four segments (see figure 4-2). The Synchronization Segment, the Propagation Time Segment, the Phase Buffer Segment 1, and the Phase Buffer Segment 2. Each segment consists of a specific, programmable number of time quanta (see Table 4-1). The length of the time quantum (t_q), which is the basic time unit of the bit time, is defined by the CAN controller's system clock f_{sys} and the Baud Rate Prescaler (BRP): t_q=BRP/f_{sys} . Typical system clocks are: f_{sys}= f_{osc}.

Parameter (t_q)		***Description***
BRP	(<32)	Defines the length of the time quantum t_q
Sync_Seg	(1)	Fixed length, synchronization of bus input to system clock
Prop_Seg	(<8)	Compensates for the physical delay times
Phase_Seg1	(<8)	May be lengthened temporarily by synchronization
Phase_Seg2	(<8)	May be shortened temporarily by synchronization
SJW	(<4)	May not be longer than either Phase Buffer Segment

Table 4-1: Parameters of the CAN Bit Time

The Synchronization Segment Sync_Seg is that part of the bit time where edges of the CAN bus level are expected to occur; the distance between an edge that occurs outside of Sync_Seg and the Sync_Seg is called the phase error of that edge.

The Propagation Time Segment Prop_Seg is intended to compensate for the physical delay times within the CAN network. This part of the bit time is used to compensate physical delay times within the network. These delay times consist of the signal propagation time on the bus and the internal delay time of the CAN nodes.

The Phase Buffer Segments (Phase_Seg1 and Phase_Seg2) and the Synchronization Jump Width (SJW) are used to compensate for the oscillator tolerance. The Phase Buffer Segments may be lengthened or shortened by synchronization. Synchronizations occur on edges from recessive to dominant, their purpose is to control the distance between edges and Sample Points. Edges are detected by sampling the actual bus level in each time quantum and comparing it with the bus level at the previous Sample Point. Synchronization may be done only if a recessive bit was sampled at the previous Sample Point and if the actual time quantum's bus level is dominant. An edge is synchronous if it occurs inside of Sync_Seg; otherwise the distance between edge and the end of Sync_Seg is the edge phase error, measured in time quanta. If the edge occurs before Sync_Seg, the phase error is negative, else it is positive.

4.3.2 Related Factors

Oscillator Tolerance Range

Each node in a CAN network derives its bit timing from its own oscillator reference. In real systems, the oscillator reference frequency, f_{CLK}, will deviate from its nominal value due to initial tolerance offset, aging and ambient temperature variations. The sum of these deviations result in a total oscillator tolerance, defined as Δf. This oscillator tolerance is a relative tolerance, which represents the deviation of the oscillator reference frequency, normalized to the nominal frequency:

Clock references typically used in real systems, such as crystal oscillators (Δf <0.1%)-which are occupied in this application- and ceramic resonators (Δf <1.2%) satisfy the demand of this application.

Propagation Delay

The significance of propagation delay in a CAN system stems from the fact that CAN allows for non-destructive arbitration between nodes contending for access to the network, as well as in-frame acknowledgement. Arbitration occurs during the identifier field and implies that multiple nodes simultaneously drive their identifier bits onto the bus. As nodes synchronize on bit edges, excessive propagation delay in the system results in invalid arbitration. Ultimately the various delays in the CAN system limit the maximum network bus length at a given bit rate. The one-way propagation delay between two nodes A and B, defined as $t_{prop\ (A,\ B)}$. This delay is the sum of all the device delays in the signal path, including the transceiver, the CAN controller and the bus medium.

As nodes must receive each other's waveforms, synchronize to them and then transmit back during arbitration, the total propagation delay in the system is the sum of the two nodes' delays. Assuming that each node in a given network has comparable delays, the total round-trip delay, defined as t_{prop}, is the sum of $t_{prop\ (A,\ B)}$ and $t_{prop\ (B,\ A)}$.

Most part of the propagation delay in this application is generated in the safeguard circuit. When the bit rate is 100k bps, this delay can be tolerated.

4.4 Communications Speed

4.4.1 Introduction

High-speed communication is appreciated in many cases, however the requirement for high bit rate always conflict with those of low cost, long distance, system robustness etc.

The Controller Area Network (CAN) is a high speed, low cost, multimaster serial communications system which efficiently supports distributed realtime control with a very high level of security. Some high speed networks are connected using CAN with bitrates up to 1 Mbit/s. But this does not mean that any CAN node can transmit 1 M bits (125 K bytes) data to another within 1 second, for some extra information must be send in a CAN data frame such as start bit, arbitration field, control field, CRC field, ACK field and end of frame field. In a practical CAN bus system the time to write data into the CAN controller and set transmission request must also be calculated. This part illustrates how to calculate the theoretical maximum transmission speed and measure the practical speed in a CAN bus.

The definitions in this paper of various data transmission speed are listed below:

- Baud rate (bit rate): the number of bit transmitted per second (bit/second)
- The maximum theoretical data transmission speed: the maximum available number of data byte per second transmitted based on CAN protocol. (byte/second)
- The maximum data transmission speed of an ideal CAN firmware system: the highest actual data transmission speed per second in an ideal CAN firmware system (byte/second)

The ideal CAN firmware system refers to:

- A CAN firmware system which is made up of only 2 nodes. The node A is called transmitter node, the other one, node B, is called receiver node.
- The speed of data receiving of node B is high enough to ensure that no case of data overrun will occur.
- The node A always transmits data with its highest speed—it is only occupied in writing data to the transmission buffer and setting sending command.

Definition of TRANSMITTER / RECEIVER

- TRANSMITTER: A unit originating a message is called "TRANSMITTER" of that message. The unit stays TRANSMITTER until the bus is idle or the unit loses ARBITRATION.
- RECEIVER: A unit is called "RECEIVER" of a message, if it is not TRANSMITTER of that message and the bus is not idle.

4.4.2 The Structure of the CAN Data Frame

Standard Data Frame

A DATA FRAME is composed of seven different bit fields: START OF FRAME, ARBITRATION FIELD, CONTROL FIELD, DATA FIELD, CRC FIELD, ACK FIELD, END OF FRAME. The DATA FIELD can be of length zero.

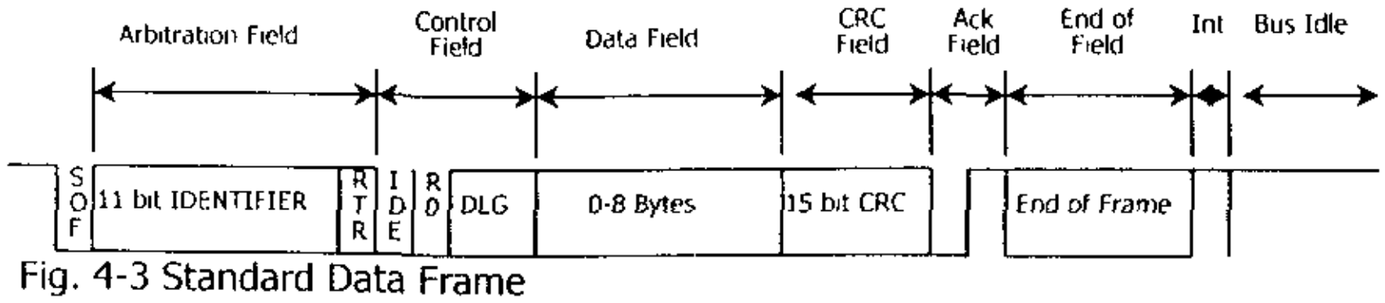


Fig. 4-3 Standard Data Frame

- **START OF FRAME**: marks the beginning of DATA FRAMEs and REMOTE FRAMEs. It consists of a single 'dominant' bit.
- **ARBITRATION FIELD**: The ARBITRATION FIELD consists of the IDENTIFIER and the RTR-BIT.
- IDENTIFIER: The IDENTIFIER's length is 11 bits. These bits are transmitted in the order from ID-10 to ID-0. The least significant bit is ID-0. The 7 most significant bits (ID-10 - ID-4) must not be all 'recessive'.

  RTR BIT: Remote Transmission Request BIT. In DATA FRAMEs the RTR BIT has to be 'dominant'. Within a REMOTE FRAME the RTR BIT has to be 'recessive'.

- **CONTROL FIELD**: The CONTROL FIELD consists of six bits. It includes the DATA LENGTH CODE and two bits reserved for future expansion. The reserved bits have to be sent 'dominant'. Receivers accept 'dominant' and 'recessive' bits in all combinations.

DATA LENGTH CODE: The number of bytes in the DATA FIELD is indicated by the DATA LENGTH CODE. This DATA LENGTH CODE is 4 bits wide and is transmitted within the CONTROL FIELD.

- **DATA FIELD**: The DATA FIELD consists of the data to be transferred within a *DATA FRAME*. It can contain from 0 to 8 bytes, which each contain 8 bits which are transferred MSB first.
- **CRC FIELD**: contains the CRC SEQUENCE followed by a CRC DELIMITER.

CRC DELIMITER: The CRC SEQUENCE is followed by the CRC DELIMITER which consists of a single 'recessive' bit.

- **ACK FIELD**: The ACK FIELD is two bits long and contains the ACK SLOT and the ACK DELIMITER. In the ACK FIELD the transmitting station sends two 'recessive' bits. A RECEIVER which has received a valid message correctly, reports this to the TRANSMITTER by sending a 'dominant' bit during the ACK SLOT (it sends 'ACK').

ACK SLOT: All stations having received the matching CRC SEQUENCE report this within the ACK SLOT by superscribing the 'recessive' bit of the TRANSMITTER by a 'dominant' bit.

ACK DELIMITER: The ACK DELIMITER is the second bit of the ACK FIELD and has to be a 'recessive' bit. As a consequence, the ACK SLOT is surrounded by two 'recessive' bits (CRC DELIMITER, ACK DELIMITER).

- **END OF FRAME**: Each DATA FRAME and REMOTE FRAME is delimited by a flag sequence consisting of seven 'recessive' bits.

Extended Data Frame

A EXTANDED DATA FRAME is composed of seven different bit fields and only differs from A STANDARD DATA FRAME in the following ways:

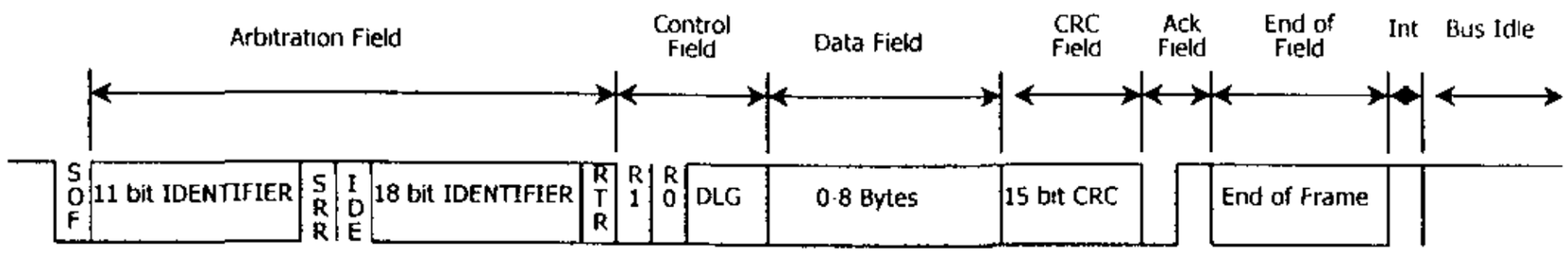


Fig. 4-4 Extended Data Frame

- **ARBITRATION FIELD**: The format of the ARBITRATION FIELD is different for Standard Format and Extended Format Frames.

  - In Standard Format the ARBITRATION FIELD consists of the 11 bit IDENTIFIER and the RTR-BIT. The IDENTIFIER bits are denoted ID-28 ... ID-18.

  - In Extended Format the ARBITRATION FIELD consists of the 29 bit IDENTIFIER, the SRR-Bit, the IDE-Bit, and the RTR-BIT. The IDENTIFIER bits are denoted ID-28 ... ID-0.

  In order to distinguish between Standard Format and Extended Format the reserved bit r1 in previous CAN specifications version 1.0-1.2 now is denoted as IDE Bit.

- **IDENTIFIER**:

  IDENTIFIER - Standard Format

  The IDENTIFIER's length is 11 bits and corresponds to the Base ID in Extended Format. These bits are transmitted in the order from ID-28 to ID-18. The least significant bit is ID-18. The 7 most significant bits (ID-28 - ID-22) must not be all 'recessive'.

  IDENTIFIER - Extended Format

  In contrast to the Standard Format the Extended Format consists of 29 bits. The format comprises two sections:

  Base ID with 11 bits and the Extended ID with 18 bits

Interframe Spacing

DATA FRAMEs and REMOTE FRAMEs are separated from preceding frames by a bit field called INTERFRAME SPACE. INTERFRAME SPACE contains the bit fields INTERMISSION and BUS IDLE.

- **INTERMISSION** consists of three 'recessive' bits.

  During INTERMISSION no station is allowed to start transmission of a DATA FRAME or REMOTE FRAME. The only action to be taken is signaling an OVERLOAD condition.

- **BUS IDLE**: The period of BUS IDLE may be of arbitrary length. The bus is recognized to be free and any station having something to transmit can access the bus. A message, which is pending for transmission during the

transmission of another message, is started in the first bit following INTERMISSION.

The detection of a 'dominant' bit on the bus is interpreted as a START OF FRAME.

4.4.3 Bit Stream Coding

The frame segments START OF FRAME, ARBITRATION FIELD, CONTROL FIELD, DATA FIELD and CRC SEQUENCE are coded by the method of bit stuffing. Whenever a transmitter detects five consecutive bits of identical value in the bit stream to be transmitted it automatically inserts a complementary bit in the actual transmitted bit stream.

The remaining bit fields of the DATA FRAME or REMOTE FRAME (CRC DELIMITER, ACK FIELD, and END OF FRAME) are of fixed form and not stuffed.

The bit stream in a message is coded according to the Non-Return-to-Zero (NRZ) method. This means that during the total bit time the generated bit level is either 'dominant' or 'recessive'.

4.4.4 The Maximum Theoretical Data Transmission Speed

Standard Data Frame

According to the definition of the standard data frame, the maximum number of data bits in a data frame is 64 and the total numbers of bits is 81 plus inter frame 3 bits. When the bit rate of a CAN bus is 1M bps, 1 M bits of signals can be transmitted within 1 second, so the number of data bits is 761.9 k that is to say the maximum theoretical data transmission speed is 95238 bytes per second.

Extended Data Frame

In an extended data frame the number of data bits is 64 and the total number of bits is 99 plus inter frame 3 bits. It can be calculated that the maximum theoretical data transmission speed is 76923 byte per second.

Bit Stream Coding

According to the bit stream coding method, when a transmitter detects five consecutive bits of identical value in the bit stream to be transmitted it automatically inserts a complementary bit in the actual transmitted bit stream. In this case the length of a CAN data frame will be increased and the maximum theoretical data transmission speed of standard data frame and extended data frame are 81633 bytes per second and 65041 bytes per second respectively.

4.4.5 The Maximum Data Transmission Speed of an Ideal CAN Firmware System

This speed is more useful for practical applications, for the maximum theoretical data transmission speed cannot be achieved in a network.

The following steps showed below introduce the method to gauge the maximum data transmission speed of an ideal CAN firmware system, which is illustrated in chapter 4.4.1.

1. Connect the CAN network and turn on the power
2. Initialize CAN notes A and B

3. Start data transmission with the full speed of node A (stop all the other tasks of this node and only engaged it in transmitting data)
4. Capture a whole data frame and gauge the time consuming of it with a digital oscillograph.
5. Calculate the data transmission speed based on the time value.

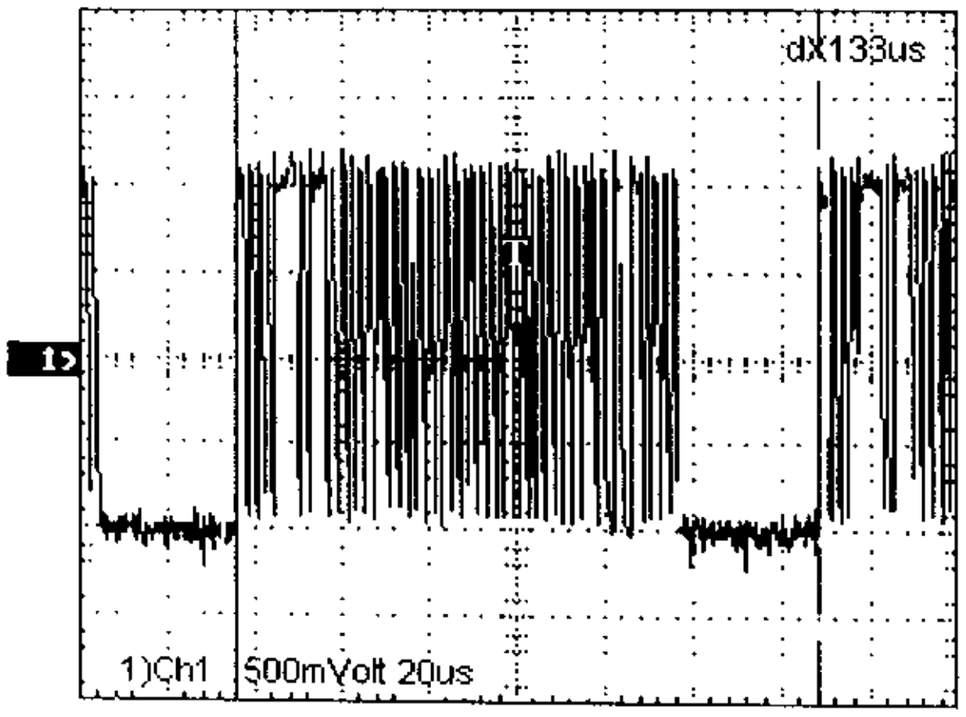


Fig. 4-5 Timing of Standard Data Frame (1M bit rate)

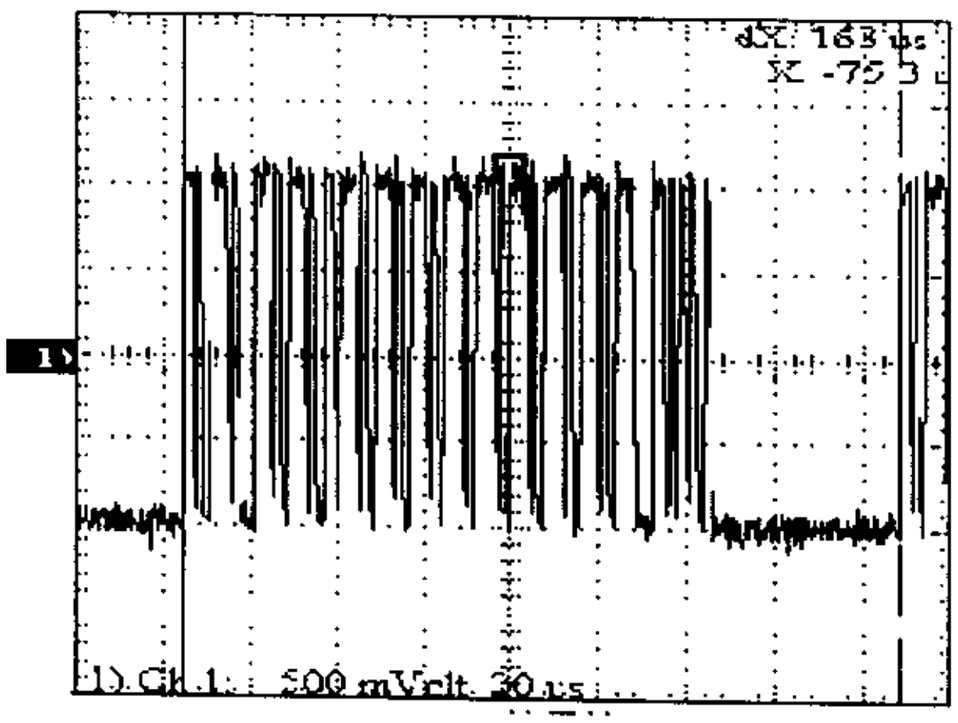


Fig.4-6 Timing of Extended Data Frame (1M bit rate)

Fig 4-5 and 4-6 show the data frame timing at 1M bit rate of the standard and extended CAN frame. The transmission time width of a standard and extended data frame is 133 μs and 163 μs, respectively.

Based on these values it can be calculated that the maximum transmission speed of this firmware system is 61650 bytes per second (standard data frame) and 48080 bytes per second (extended data frame) respectively.

4.4.6 Determine The Maximum Speed of the CAN Network

The maximum theoretical data transmission speed and the maximum transmission speed of an ideal CAN firmware network have different meaning.

The first one determines the maximum burst speed of a CAN bus network. When some nodes of a CAN bus network try to send data at the same time, one of them can get the bus access right and start transmission based on the priority arbitration mechanism. When this node completes its transmission some other node will get the bus access right and start transmission without any delay. In this case the maximum theoretical data transmission speed can be achieved.

However in a practical network system, the transmission speed between two special nodes is a more important factor and this speed cannot meet the maximum theoretical data transmission speed. The Maximum Data Transmission Speed of an Ideal CAN Firmware System determines the highest data transmission speed from one node to another.

Table 4-2 shows the two kinds of maximum speed at various bit rates.

Bit rate (bits/second)		1M	500k	250k	125k	100k
Maximum theoretical speed (bytes/second)	***Standard frame***	95238	47619	23809	11905	9524
	Extended frame	76923	38461	19231	9615	7692
Maximum trans speed of an ideal system (bytes/second)	***Standard frame***	60150	32786	17241	8850	7055
	Extended frame	49080	27397	14120	7229	5761

Table 4-2 The Maximum data transmission speed

Chapter 5 Software Implementation

5.1 Introduction

As what is illustrated in chapter 3, the most important opportunity of this application is to realize an intelligent network card, with which only a few commands can fulfill the communications functions. In fact it is very simple and easy to operate this device that with only three basic commands of this network device, which are initCanCmd, sendDataCmd and receDataAck, communications jobs can be built up.

The software of this application can be classified into two parts, which are single-chip microcomputer software and the PC device driver. The single-chip microcomputer software is coded with assembler language and the PC device driver with C language.

In this chapter the software architecture, data flow sequence and task scheduling mechanism will be introduced.

5.1.1 RS232 Data Frame

The PC communicates with the network card via RS232 interface and some data compose a data frame. Every data frame can be divided into three fields:

- **START OF FRAME**: marks the beginning of a DATA FRAME. To be compatible with the traditional data transmission method this field is made up of 4 bytes that have exactly meaning. However a new method is also utilized in this application to make the transmission more efficiently. This new method makes use of CTX and DTR pin of the RS232 interface to indicate the beginning of a DATA FRAME.
- **DATA LENGTH FIELD**: consists of only 1 byte and indicate the length of the DATA FRAME. The maximum number of bytes within a DATA FRAME is 16.
- **DATA FIELD**: can be made up of 0 to 15 bytes of data.

The data frames transferred from the PC are called ***command frames*** and the data frames from network card are called ***acknowledgement frames***.

5.2 Embedded Software

The single-chip microcomputer and its embedded software make up of the network card firmware system. The software is programmed with assembler language and down loaded in the flash ROM.

5.2.1 Source Files of This Software

The source files of this software are showed in table 5-1.

232CAN.ASM	Main source file of this software.

ACK.ASM	Acknowledgement tasks definitions
BASICCAN.ASM	Tasks for SJA1000 in BasicCAN mode (CAN2.0 A)
PELICAN.ASM	Tasks for SJA1000 in PeliCAN mode (CAN2.0 B)
RAM.INC	RAM address definitions for 89C52
CAN_REG.INC	Register definitions for SJA1000
COMMAND.INC	232-CAN command and acknowledgement definitions
CRYS16.INC	BTR definition with 16M oscillator frequency for SJA1000
CRYS24.INC	BTR definition with 24M oscillator frequency for SJA1000

5.2.2 Definitions of This Software

Some names used in this application, which have special meanings, are: ***task***, ***job*** and ***task scheduling***.

It is often essential to organize applications into independent, though cooperating, programs. Each of these programs, while executing, is called a ***task***. In this application, tasks have immediate, shared access to most system resources, while also maintaining enough separate contexts to maintain individual threads of control.

When a command is transferred to the network card, several tasks will perform various functions to implement the function of this command. All of the tasks, which are involved in this process, make up a ***job***.

Multitasking requires a ***scheduling algorithm (task scheduling)*** to allocate the CPU to ready tasks. Priority-based preemptive scheduling is the scheduling algorithm in this application, with which the system can response to various requests of the host computer and the network nodes more quickly.

All of the tasks and jobs depend on the hardware event and software event to run. For example when a serial port input interrupt event occurs, it will trigger an input task. The interrupt is handled by a special program called interrupt service routine ***(ISR)***. The ISRs handles all the hardware event and trigger related tasks and jobs. So ISRs is called hardware dependent software and tasks and jobs are called hardware independent software.

5.2.3 Intertask Communications

The complement to the multitasking routines described in the previous part is the intertask communication facilities. These facilities permit independent tasks to coordinate their actions.

This application supplies a set of intertask communication mechanisms, including:

- ***Shared memory***, for simple sharing of data.
- ***Message buffer***s, for intertask message passing within a CPU.

With these facilities the tasks can cooperate with each other and carry out various jobs of the intelligent network card.

5.2.4 Major Tasks Outline

The major tasks of this application is:

1. PC data receiving task: to receive and store the data frames from the host PC.
2. PC command compiling task: to compile the command from the PC and inform the corresponding tasks.
3. Pc command acknowledgement task: to acknowledge the commands from the PC.
4. Initialization task: to initialize the system.
5. Sending and receiving task: to send or receive data via CAN bus network.
6. Error handling task: to handle the error message of the CAN controller.
7. Task scheduling task (main task of this application): to scheduling the various tasks of this application.
8. Other tasks: tasks that perform all the other functions, which are related to the correct operating of this application.

Figure 5-1 illustrates the relations of these tasks.

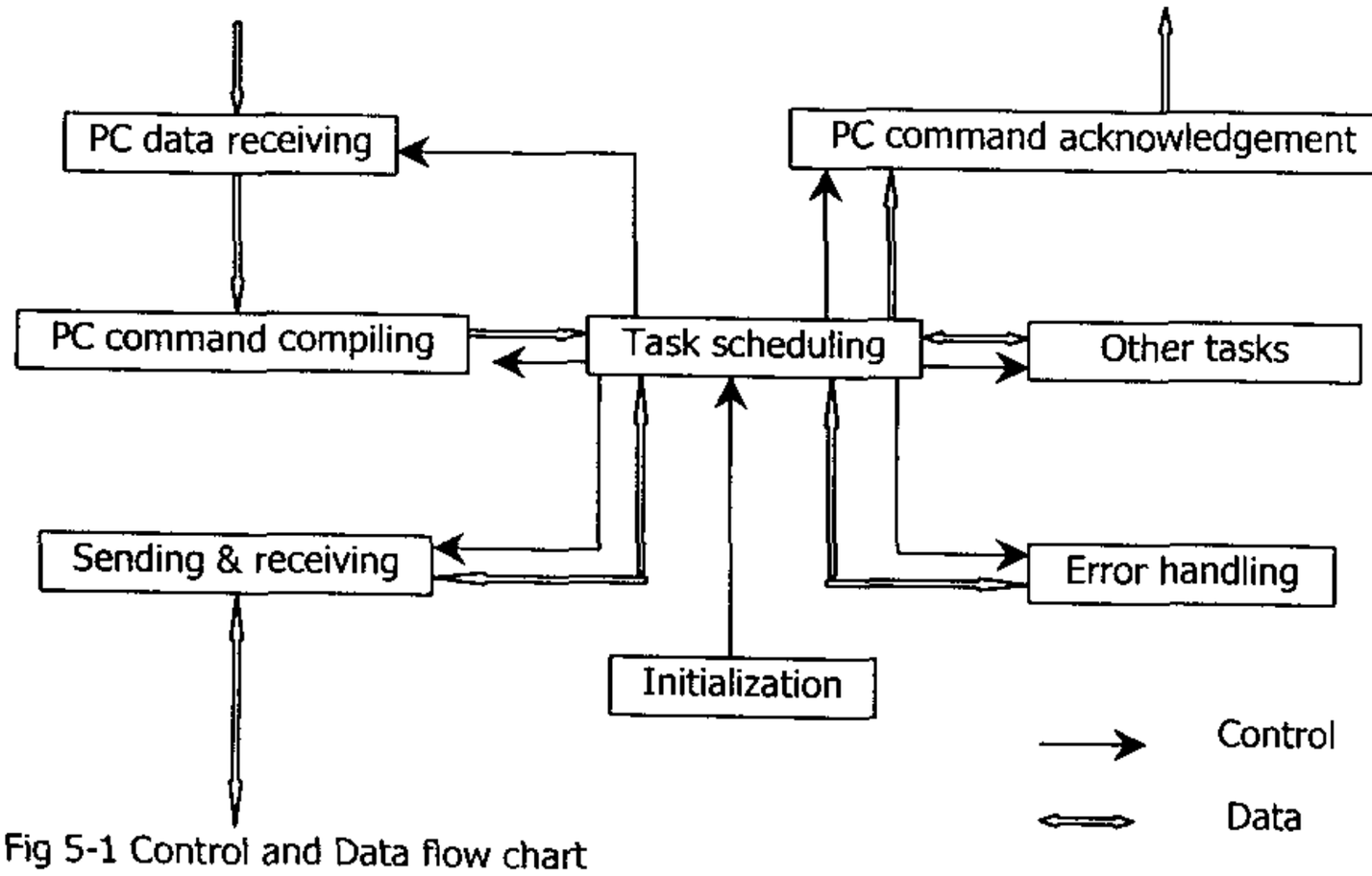


Fig 5-1 Control and Data flow chart

5.2.5 Task Sequence of a Typical Job

When the PC sends a command to the network card, the network card software will start a job to carry out this command. In the following part the task sequence of a typical command — sendDataCmd — will be introduced.

When the PC sends a command to the network card, the PC data receiving task will receive the data, detect the start and end of the command frame and transfer the whole command frame into the command buffer.

At this time the task scheduling task finds that a command is ready in the command buffer, so it inform the PC command compiling task to compile the command.

Then the command compiling task compiles the command and detects that it is a sendDataCmd (to send a data frame), so this task will transfer the data bytes in this command into the transmit buffer.

After this the task scheduling task finds that a command has been compiled and some data in the transmit buffer is ready. So it flush the command buffer and inform the sending and receiving task to start transmission.

The sending and receiving task will start sending the data frame via the CAN network. If this data frame is transmitted successfully this task will return an OK status.

When the task scheduling task detects the OK status it will inform the PC command acknowledgement task to acknowledge the PC that the data has been transmitted successfully (if the PC permits to receive acknowledge).

The PC command acknowledgement task will send an OK status to the PC and the job finishes.

5.2.6 Interrupt Service Routine

Hardware interrupt handling is of key significance in this system, because it is usually through interrupts that the system is informed of external events. For the fastest possible response to interrupts, interrupt service routines (ISRs) in this firmware system run in a special context outside of any task's context. Thus, interrupt handling involves no task context switch.

The main ISRs of this system are:

- Timer interrupt service routine
- Serial port interrupt service routine (input interrupt)
- External interrupt service routine 0
- External interrupt service routine 1

5.2.7 Initialization Task

After powered on or correct reset, the single-chip microcomputer (89C52) and the CAN controller (SJA1000) enter the default status and are ready to run. The first task when boot up is the initialization task, which sets up the single-chip microcomputer environment, locates the interrupt service routine handler address and initializes the CAN controller.

The main function and flow chart of this task are listed in table 5-2

Task name	Initialization
Main function	Initialize single-chip microcomputer and CAN controller
Related interrupt	None

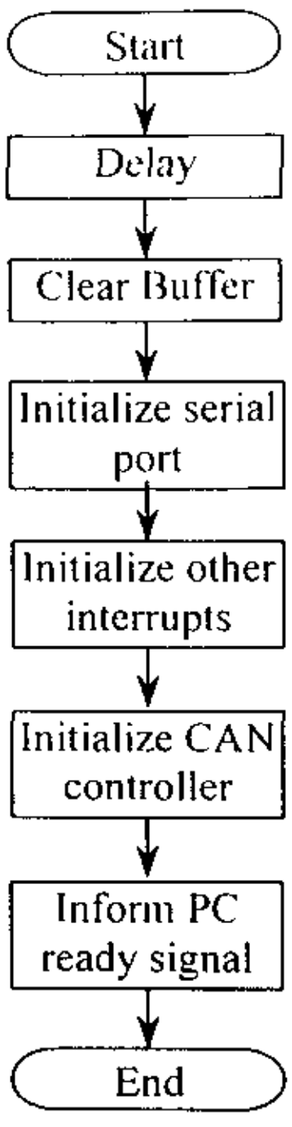
Flow chart	
Note	

Table 5-2 Main function of the initialization task

5.2.8 Task scheduling Task

This task is the most important task of the network firmware system and performs hardware maintenance, task scheduling, error handling and other related functions. This task is initialized by the initialization task and has the highest priority of all the tasks. In fact it serve as the operating system of this firmware system.

One important character of this task is self-maintenance function, which can reset itself by call the initialization task when fatal error occurs. In some other cases when it detects error occurs in some component of the system, e.g. the CAN controller has enter passive error mode, it will reset this component.

The most important function of this task is to schedule tasks. To implement this function all the tasks of this firmware system setup the respective task message stored in a special RAM area called message table. The task scheduling task will polling the message table, diagnostic the task priority of each pending task and locate the resources of the firmware environments to the highest priority task

The main function and flow chart of this task are listed in table 5-3

Task name	Task scheduling Task
Main function	Maintain hardware devices Schedule tasks Handle errors of the firmware system

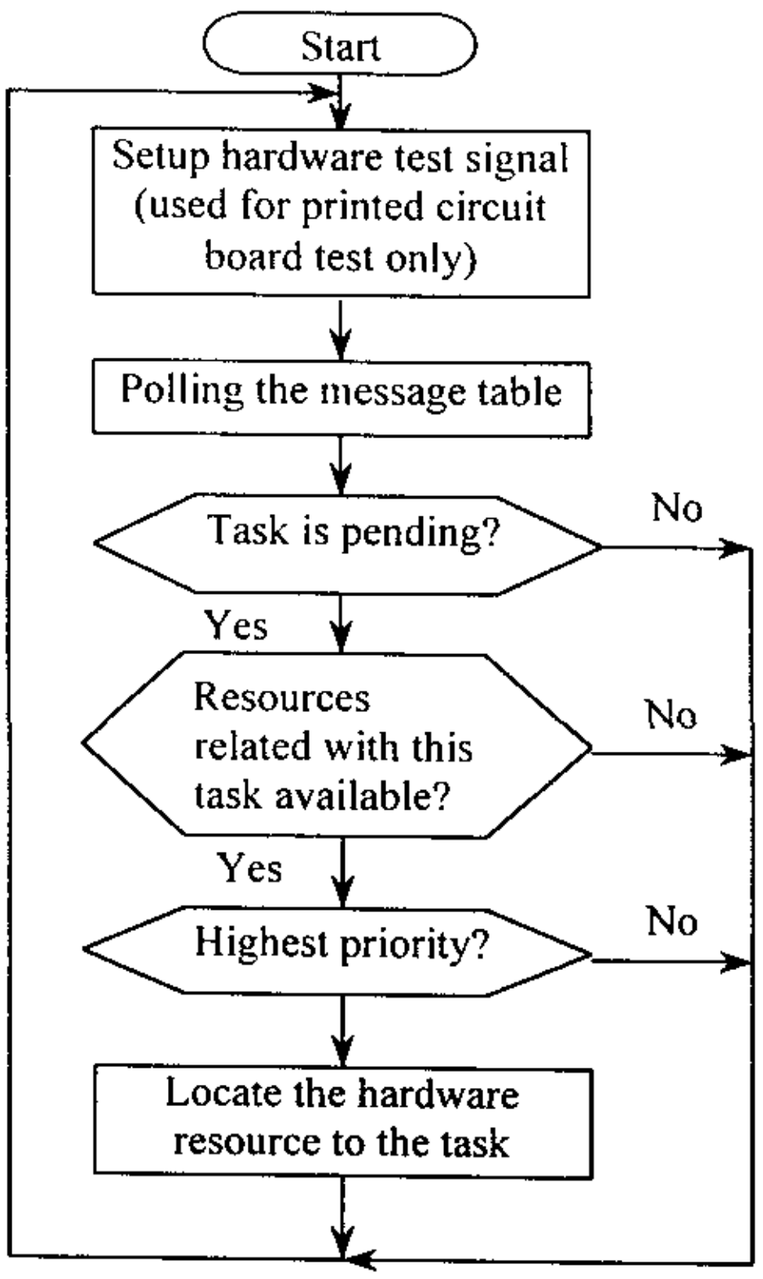
	Other functions
Related interrupt	Timer interrupt Serial port interrupt (input interrupt) External interrupt 0 External interrupt 1
Flow chart	
Note	The self-maintenance function is not introduced in this table. To implement this function a special macro program called watch-dog macro are programmed in the last part of all the software source codes.

Table 5-3 Main function of the task scheduling task

5.2.9 PC Data Receiving Task

When the PC sends one byte data via RS232 bus to the network card, a serial port input interrupt request will be generated and the serial port input ISR will fetch the data to a special buffer called serial data buffer. When all of the data of a frame have been received and stored in the serial data buffer, the PC data receiving task will be called. At this time this task will detects the type of the frame, copy it to the correspondent buffer and clear the serial data buffer.

The main function and flow chart is listed in table 5-4

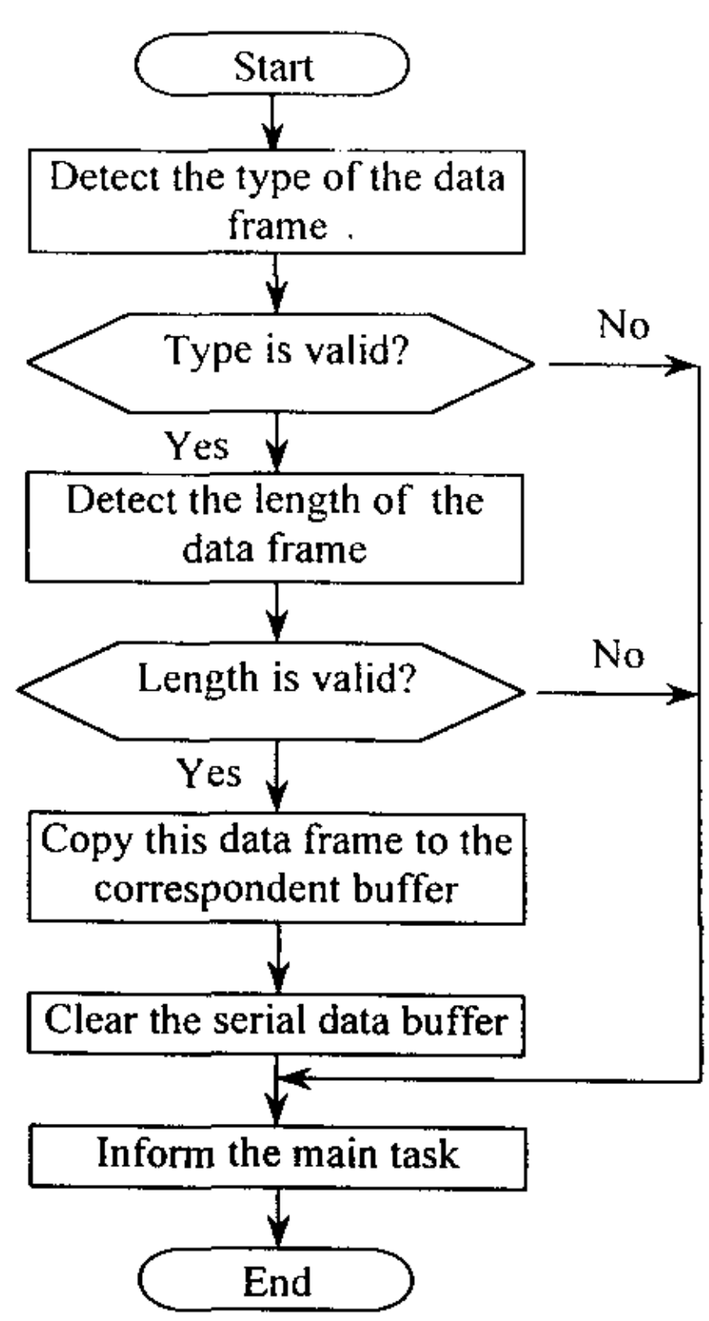
Task name	PC data receiving task
Main function	Detect the type of the data frame Copy the data frame to the correspondent buffer Clear serial port data buffer
Related interrupt	Serial port interrupt (input interrupt) External interrupt 0
Flow chart	
Note	If the type or length of the data frame is invalid, it will not be copied. In this case the PC data receiving task will inform the task scheduling task an error message number and return. If the type and length of the data frame is valid, it will be copied to the correspondent buffer and the PC data receiving task will inform the task scheduling task an OK message and return.

Table 5-4 Main function of the PC data receiving task

5.2.10 PC Command Compiling Task

In general the data frame from PC can be classified into two groups: Data transfer frame and other frames.

The data transfer frame, which is most important command, will be compiled and the transfer data in this frame will be moved to the host data buffer.

The other frames are all command frames, which perform initialization, maintenances and other functions. These commands will be compiled and, if necessary, an acknowledgement frame will be generated based on the command and status of the network card firmware and network.

The main function and flow chart is listed in table 5-5

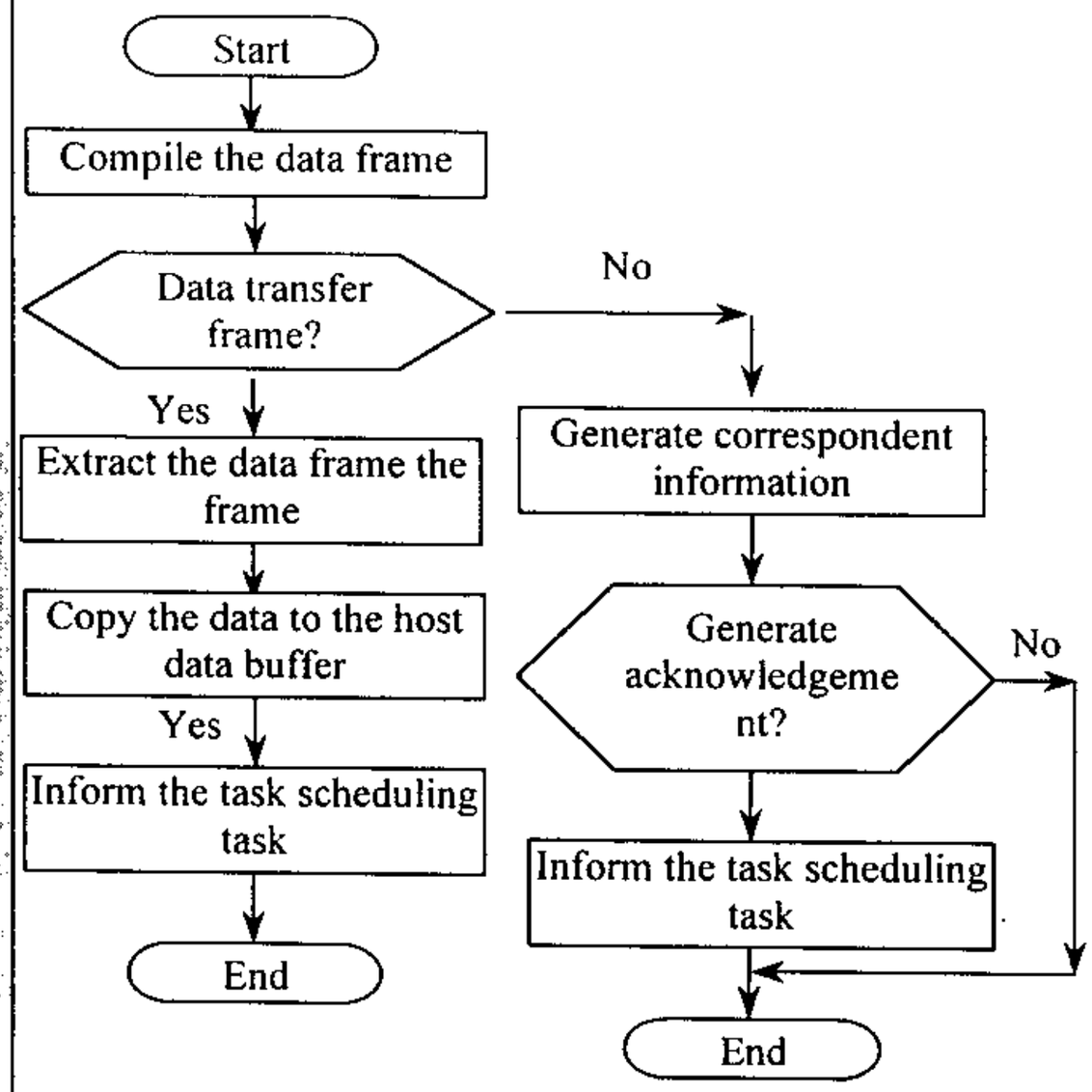
Task name	PC command compiling task
Main function	Compile the data frame Generate the acknowledgement frame Inform the task scheduling task a message
Related interrupt	None
Flow chart	
Note	

Table 5-5 Main function of the PC command compiling task

5.2.11 PC Command Acknowledgement Task

When an acknowledgement frame is generated or some data from the CAN bus arrive, this task will be called (by the task scheduling task) to send data to the PC. In this task the acknowledgement frame will be tested and then sent.

The main function and flow chart is listed in table 5-6

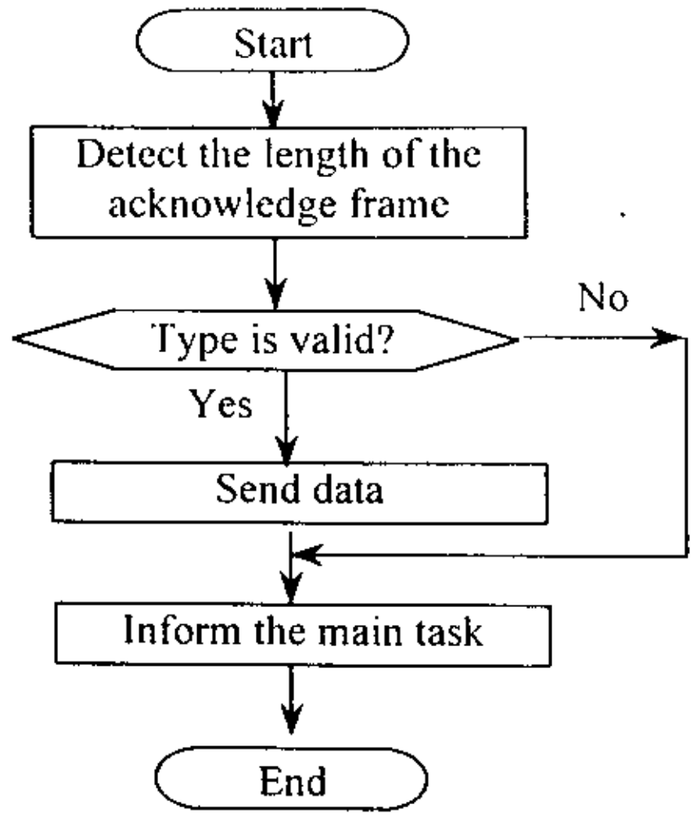
Task name	PC command acknowledgement task
Main function	Detect the length of the acknowledgement frame Send this frame to the PC
Related interrupt	None
Flow chart	
Note	

Table 5-6 Main function of the PC command acknowledgement task

5.2.12 Sending and Receiving Task

This task implements the communications function between the single-chip microcomputer and the CAN controller. It is made up of several sub-tasks and ISR, which are:

- Data sending sub-task
- CAN controller status polling sub-task
- CAN data input interrupt service routine

Each of these sub-tasks and ISR maintains its own context and communicates with the task scheduling task with correspondent messages. So this task can be called a task group.

The main function and flow chart is listed in table 5-7

Task name	Sending and receiving task
Main function	Send data to the CAN controller Polling the status of the CAN controller Handle the CAN data input interrupt

Related interrupt	External interrupt 1
Flow chart	Every sub-task and ISR has its own flow chart respectively.
Note	

Table 5-7 Main function of the sending and receiving task

5.2.13 Error Handling Task

A key characteristic of this intelligent network card is self-maintenance and error handling mechanism, by which the single-chip microcomputer can maintain its resources, react to the error message of the CAN controller, and (if necessary) send some information to the PC.

The self-maintenance mechanism is implemented by the main task (task scheduling task) and the error handling by the error handling task.

The error handling task detects the error status of the CAN controller and makes correspondent reaction based on the error status.

The CAN controller can run in two modes, which are: BASIC CAN (CAN 2.0A) mode and Peli CAN (CAN 2.0B) mode. In BASIC CAN mode when a change of either the error status or bus status bits occurs the EI (error interrupt) bit of the interrupt register is set and a CAN error interrupt is generated (if enabled). In Peli CAN mode several sources can generate the error interrupt which are:

- Bus error interrupt (BEI): this error interrupt is generated when the CAN controller detects an error on the CAN bus and the BEIE bit is set within the interrupt enable register.
- Error passive interrupt (EPI): this error interrupt is generated when the CAN controller has reached the error passive status (at least one error counter exceeds the protocol-defined level of 127) or if the CAN controller is in the error passive status and enters the error active status again and the EPIE bit is set within the interrupt enable register.
- Data overrun interrupt (DOI): this interrupt is generated on a '0-to-1' transition of the data overrun status bit and the DOIE bit is set within the interrupt enable register.
- Error warning interrupt (EI): this interrupt is generated on every change (set and clear) of either the error status or bus status bits and the EIE bit is set within the interrupt enable register.

The main function and flow chart is listed in table 5-8

Task name	Error handling task
Main function	Handle CAN controller error interrupt and other messages
Related interrupt	External interrupt 1

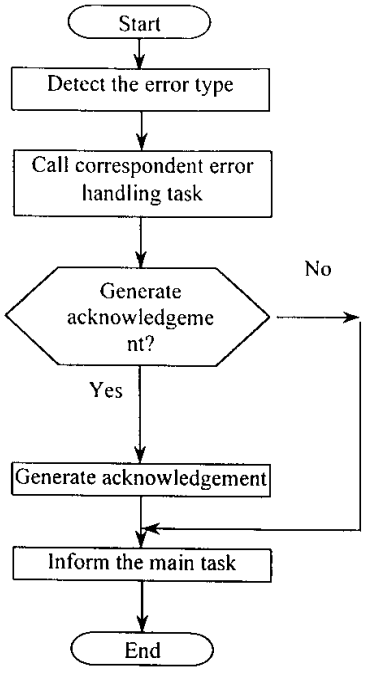
Flow chart	
Note	

Table 5-8 Main function of the error handling task

5.2.14 Other Tasks

The other tasks in this firmware system response to the various event of the input/output and single-chip microcomputer itself. The main functions of these tasks are:

- Initialize RS232: if the PC want to re-initialize it.
- Response to some commands: get version command, read registers command, write registers command go to sleep command, about send command, acceptance filter command, self test command, go to listen only mode command etc.
- Response to some acknowledgements: read registers acknowledgement, arbitration lost acknowledgement etc.

5.3 Device Driver of PC Software

5.3.1 Commands and Acknowledgements Outline

The commands and acknowledges used in this application are listed in table 5-9.

No.	Command		Acknowledgement	
System commands				
01h	init232Cmd	Initialize RS232		
03h	getVersionCmd	Get version	getVersionAck	Get version
04h	readRegCmd	Read registers	readRegAck	Read registers
05h	writeRegCmd	Write registers		
CAN2.0A commands				
11h	initCanCmd_a	Initialize CAN controller		
12h	sendDataCmd_a	Send data	sendDataAck_a	Send data
13h	aboutSendCmd_a	About sending data		
14h	goToSleepCmd_a	Go to sleep mode		
18h			receDataAck_a	Receive data
19h			OverrunAck_a	Data overrun
1ah			errorAck_a	Error acknowledgement
CAN2.0B commands				
21h	initCanCmd_b	Initialize CAN controller		
22h	sendDataCmd_b	Send data	sendDataAck_b	Send data
23h	aboutSendCmd_b	About sending data		
24h	goToSleepCmd_b	Go to sleep mode		
25h	acceFilterCmd_b	Acceptance filter		
26h	selfTestCmd_b	Self test		
27h	listenOnlyCmd_b	Go to listen only mode		
28h			receDataAck_b	Receive data
29h			errorAck_b	Error acknowledgeme

No.	01h
Command	Initialize RS232
Format	03h 01h B
	B: baud rate 0-115200 1-57600 2-38400 3-19200 4-9600 5-4800 6-2400 7-1200
Acknowledgement	None
Note	The default baud rate after power on is 9600bps.

No.	03h
Command	Get version
Format	02h 03h
Acknowledgement	(hex) 10h 03h 4eh 4eh 55h 32h 30h 30h 20h 56h 65h 72h 31h 2eh 32h 30h (ASCII) ♦♥NJU200 Ver1.20
Note	

No.	04h
Command	Read registers
Format	??h 04h R…R
	R: register address
Acknowledgement	??h 04h R V…R V
	R: register address

	V: value
Note	The number or registers to be read must no more than 7

No.	05h
Command	Write registers
Format	??h 05h R V...R V
	R: register address
	R: register address V: value
Acknowledgement	None
Note	The number or registers to be read must no more than 7

5.3.3 CAN 2.0A Commands

No.	11h
Command	Initialize CAN controller
Format	06h 11h A M BTR0 BTR1
	A: value of the acceptance filtering register M: value of the acceptance mask register BTR0: value of BTR0 register BTR1: value of BTR1 register
Acknowledgement	None
Note	

No.	12h
Command	Send data
Format	??h 12h D...D
	D: data to be send
Acknowledgement	03h 12h ACK
	ACK: 00h — TX buffer full

	0ffh— send OK
Note	

No.	13h
Command	About sending data
Format	02h 13h
Acknowledgement	None
Note	

No.	14h
Command	Go to sleep mode
Format	02h 14h
Acknowledgement	None
Note	

No.	18h
Command	None
Acknowledgement	Receive data
Format	??h 18h D...D
	D: data received
Note	

No.	19h
Command	None
Acknowledgement	Data overrun
Format	02h 19h
Note	Data overrun bit is cleared before this acknowledgement is sent.

No.	1ah
Command	None
Acknowledgement	Error acknowledgement
Format	03h 1ah S
	S: value of the status register
Note	

5.3.4 CAN 2.0B Commands

No.	21h
Command	Initialize CAN controller
Format	0eh 21h A0 A1 A2 A3 M0 M1 M2 M3 BTR0 BTR1
	A0—A3: value of the acceptance filtering register M0—M3: value of the acceptance mask register BTR0: value of BTR0 register BTR1: value of BTR1 register
Acknowledgement	None
Note	

No.	22h
Command	Send data
Format	??h 22h D...D
	D: data to be send
Acknowledgement	03h 22h ACK
	ACK: 00h — TX buffer full 0ffh— send OK
Note	

No.	23h
Command	About sencing data
Format	02h 23h

Acknowledgement	None
Note	

No.	24h
Command	Go to sleep mode
Format	02h 24h
Acknowledgement	None
Note	

No.	25h
Command	Go to acceptance mode
Format	02h 25h
Acknowledgement	None
Note	

No.	26h
Command	Go to self-test mode and send test data
Format	??h 26h D…D
	D: data to be sent
Acknowledgement	None
Note	

No.	27h
Command	Go to listen only mode
Format	02h 27h
Acknowledgement	None
Note	

No.	28h

Command	None
Acknowledgement	Receive data
Format	??h 28h D.. D
	D: data received
Note	

No.	29h
Command	None
Acknowledgement	Error acknowledgement
Format	06h 29h S E R T
	S: value of the status register E: value of the error capture register R: value of the receive error counter T: value of the transmit error counter
Note	

No.	2ah
Command	None
Acknowledgement	Data overrun
Format	02h 2ah
Note	Data overrun bit is cleared before this acknowledgement is sent.

No.	2bh
Command	None
Acknowledgement	Error passive acknowledgement
Format	06h 2bh S E R T
	S: value of the status register E: value of the error capture register R: value of the receive error counter T: value of the transmit error counter

Note	

No.	2ch
Command	None
Acknowledgement	Arbitration lost
Format	06h 2ch A
	A: value of the arbitration lost capture register
Note	

No.	2dh
Command	None
Acknowledgement	Bus error acknowledgement
Format	06h 2dh S E R T
	S: value of the status register E: value of the error capture register R: value of the receive error counter T: value of the transmit error counter
Note	

5.3.5 Communication Software

Sample programs are provided in this part. These programs implement the communications function between the PC and the network card. All these programs are coded by C language.

- Sample program 1: Basic communications

```
#include <dos.h>
#include <conio.h>
#include <stdio.h>

void sendCommand(unsigned char[]);          /*send command function */

void main(void)
{
```

```
unsigned int receData[16];                    /* RX data buffer          */
unsigned char i=0,j=0;
unsigned char msr;
unsigned char getVersion[]={2,0x03};          /* get version command     */
for(j=0;j<16;j++) receData[j]=0;

/* initialize COM 1 */
clrscr();
outportb(0x3fb,0x80);                         /* set baud rate           */
outportb(0x3f8,0x0c);                         /* 9600 bps                */
outportb(0x3f9,0x00);
outportb(0x3fb,0x03);                         /* set mode                */
outportb(0x3f9,0x0);                          /* disable interrupt       */
outportb(0x3fc,0x00);                         /* initialize RTS pin      */
receData[i]=inportb(0x3f8);                   /* clear RX buffer (COM1) */
receData[i]=inportb(0x3f8);

sendCommand(getVersion);                      /* send the command        */

/* receive data */
while(!(kbhit()))
{
        msr=inportb(0x3fe);                   /* detect start of frame   */
        if((msr&0x01!=0)&&(msr&0x10)==0)
        {
                i=0;
                printf("\n");
        }

        if((inportb(0x3fd)&0x01)==1)
        {
                receData[i]=inportb(0x3f8);/* receive data               */
                cprintf("%c",receData[i]);
                i++;
                 if(i>=16)          i=0;
        }
  }
}
void sendCommand(unsigned char* command)
{
 unsigned char i;
 outportb(0x3fc,0x03);                        /* send start of frame     */
 outportb(0x3fc,0x00);
 outportb(0x3fc,0x03);

 for(i=0;i<command[0];i++)                    /* send data               */
 {
 while((inportb(0x3fd)&0x20)==0) {}
 outportb(0x3f8,command[i]);
 }
```

}

This program can get the version of the network card. The command frame definition is:

unsigned char getVersion[]={2,0x03};

and the acknowledgement of the network card is:

♥NJU200 Ver1.20

Another example of the command frame definition is read registers command:

unsigned char readReg[]={3,0x04,2};

This command can read the value of the register 2 (status register).

- Sample program 2: CAN node self-test function

```
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <io.h>
#include <dos.h>

void int_ok(void);
void int_end(void);
void interrupt (*old_int4)(void);              /* old interrupt vector      */
void interrupt new_int4(void);                 /* new interrupt vector      */
void init_232(void);
void x_sound(int xx,int yy);
void sendCommand(unsigned char[]);             /* send command function */

unsigned char data_get[300];                   /* receive  data buffer      */
unsigned int count = 0;                        /* receive buffer pointer    */
unsigned int PORT = 0X3F8;          unsigned char IRQ4 = 0x0c;
unsigned char LSB = 0x30;           unsigned char MSB = 0x00;
unsigned char rs232baudrate[]={3,0x01,4}; /* initialize RS232            */

void main(void)
{
 char ch;   int i,j;
 unsigned char initCan_b[]={12,0x21,0x00,0x00,0x00,0x00,0xff,
0xff,0xff,0xff,0x00,0x1c};                     /* initialize CAN controller     */
 unsigned char selfTest_b[]={7,0x26,2,5,0,1,1}; /* self test command     */

 clrscr();
 gotoxy(7,18); cprintf("1 - COM1:3F8H, 2 - COM2:2F8H press<1,2> to
select");
 ch = getch();
 switch(ch)
 {
  case '1' :  PORT = 0x3f8;IRQ4 = 0x0c; break;
  case '2' :  PORT = 0x2f8;IRQ4 = 0x0b; break;
```

```
 default  : break;
}

clrscr();
gotoxy(7,18); cprintf("0 - 115200   1 - 57600   2 - 38400   3 -19200");
gotoxy(7,19); cprintf("4 - 9600     5 - 4800    6 - 2400    7 -1200");
gotoxy(7,21); cprintf("Please select. Default: 9600");
ch = getch();
switch(ch)
{
 case '0' : LSB=0x01; MSB=0x00; rs232baudrate[2]=0; break;
 case '1' : LSB=0x02; MSB=0x00; rs232baudrate[2]=1; break;
 case '2' : LSB=0x03; MSB=0x00; rs232baudrate[2]=2; break;
 case '3' : LSB=0x06; MSB=0x00; rs232baudrate[2]=3; break;
 case '4' : LSB=0x0c; MSB=0x00; rs232baudrate[2]=4; break;
 case '5' : LSB=0x18; MSB=0x00; rs232baudrate[2]=5; break;
 case '6' : LSB=0x30; MSB=0x00; rs232baudrate[2]=6; break;
 case '7' : LSB=0x60; MSB=0x00; rs232baudrate[2]=7; break;
 default  :break;
}
 disable();
 clrscr();
 delay(100);
 init_232();                    /* initialize RS232                */
 old_int4 = getvect(IRQ4);      /* save old interrupt vector       */
 setvect(IRQ4, new_int4);       /* set new interrupt vector        */
 int_ok();                      /* enable interrupt                */
 sendCommand(initCan_b);        /* initialize CAN controller       */
 delay(100);
 sendCommand(selfTest_b);       /* go to self-test mode            */
 while(ch!=27)                  /* press ESC to return to DOS      */
   {
    delay(2000);
    if(count!=0)
    {
            printf("\n Receve:");
            for(j=1;j<count;j++)
                    cprintf("%3x",data_get[j]);
            count=0;
    }
    if(kbhit())
            ch=getch();
    else ch='k';
   }
  int_end();
  setvect(IRQ4, old_int4);
}
void int_ok(void)
{
 int da;
```

```
 outportb(0x20,0x20);
 da=inportb(0x21);
 da &= 0x80;
 outportb(0x21,da);
 enable();
}
void int_end(void)
{
 int dat;
 dat=inportb(0x21);
 dat |= 0x10;
 outportb(0x21,dat);
 outportb(0x20,0x20);
 disable();
}
void x_sound(int xx,int yy)
{
 sound(xx);
 delay(yy);
 nosound();
}
void init_232(void)
{
 outportb(PORT+3,0x80);      /* set baud rate 9600                    */
 outportb(PORT+0,0x0c);      /* LSB                                   */
 outportb(PORT+1,0x00);      /* MSB                                   */
 outportb(PORT+3,0x03);      /* 8-data,1-stop,no-p                    */
 outportb(PORT+4,0x0f);      /* -RTS = 1                              */
 outportb(PORT+1,0x09);      /* enable receive,modem interrupt */
 sendCommand(rs232baudrate);
 delay(60);
 outportb(PORT+3,0x80);      /* set baud rate                         */
 outportb(PORT+0,LSB);       /* LSB  0c is 9600,30 is 2400 L.T.B */
 outportb(PORT+1,MSB);       /* MSB                                   */
 outportb(PORT+3,0x03);      /* 8-data,1-stop,no-p                    */
 outportb(PORT+4,0x0f);      /* -RTS = 1                              */
 outportb(PORT+1,0x09);      /* enable receive,modem interrupt  */
}

void interrupt new_int4(void)
{
 unsigned char iir,msr;
 disable();
 msr=inportb(PORT+6);        /* frame header information              */

 if((msr&0x01!=0)&&(msr&0x10)==0)
 {
 count++;                    /* detect start of frame                 */
 data_get[count]=0;
```

```
}
count++;
iir = inportb(PORT+2) & 0x07;        /* interrupt ID. register          */
switch(iir)
{
case 0x04:                           /* receive data                    */
if((inportb(PORT+5) & 0x1e) == 0x00)
{
        data_get[count]=inportb(PORT);
}                                    /* no error                        */
else
{
        inportb(PORT);
        x_sound(100,10); x_sound(1000,10);
        data_get[count] = 0xff;
 }
 break;
 }
 outportb(0x20,0x20);
 enable();
}
void sendCommand(unsigned char* command)
{
 unsigned char i;
 outportb(PORT+4,0x0f);              /* send start of frame             */
 outportb(PORT+4,0x0c);
 outportb(PORT+4,0x0f);

 for(i=0;i<command[0];i++)
 {
 while((inportb(0x3fd)&0x20)==0) {}
  outportb(0x3f8,command[i]);
  }
 }
```

This program initialize the CAN controller to CAN 2.0B mode and send the self test command. Press ESC key can return to DOS. The command definition is:

```
unsigned char selfTest_b[]={7,0x26,2,5,0,1,1};      /* self test command */
```

The acknowledgement of the network card is:

7 28 2 5 0 1 1

The '28' is the acknowledgement number of receive data. '2 5 0 1 1' are test data.

References

[1] Road vehicles - Interchange of digital information - Controller area network(CAN) for high-speed communication. ISO 11898. International Standardization Organization. 1993.

[2] CAN specification Version 2.0, Parts A and B. Robert Bosch GmbH. Postfach 30 02 40, D-70442 Stuttgart . 1991.

[3] Farsi M,Ratcliff K, Barbosa Manuel. Overview of controller area network. Computing and Control Engineering Journal v 10 n 3 1999 IEE p 113-120 0956-3385.

[4] Livani MA, Kaiser J, Jia WJ. Scheduling hard arid soft real-time communication in a controller area network. Control Engineering Practice 7: (12) 1515-1523 1999/12.

[5] Tindell K, Burns A, Wellings AJ. Calculating Controller Area Network(CAN) Message Response-times. Control Engineering Practice 3: (8) 1163-1169 1995/08.

[6] Data Sheet SJA1000. Philips Semiconductors. 2000 May 11.

[7] Data Sheet PCA82C250 CAN controller interface. Philips Semiconductors. 2000 Jan. 13.

[8] Data Sheet PCA82C251 CAN controller interface. Philips Semiconductors. 2000 Jan. 13.

[9] Navet N. (LORIA-INPL), Song Y.Q. Reliability improvement of the dual-priority protocol under unreliable transmission. Control Engineering Practice 7 8 1999 Elsevier Science Ltd p 975-981 0967-0661.

[10] Travis A.R.L. (Cambridge Univ.), Collier, M. Software interface for an industrial control network using the CAN protocol. IEEE AFRICON Conference v 2 Sep 25-27 1996, IEEE p 1081-1082.

[11] Cena G. (Politecnico di Torino), Valenzano A. Overclocking of controller area networks. Electronics Letters 35 22 1999 IEE p 1923-1925 0013-5194.

[12] Rodrigues Luis, Guimaraes Mario , Rufino Jose. Fault-tolerant clock synchronization in CAN. Proceedings - Real-Time Systems Symposium Dec 2-4 1998 IEEE p 420-429.

[13] DeviceNet Specification, Volume I, Release 1.3. Open DeviceNet Vendor Association Inc. December 1995.

[14] How to Handle Data Overrun Conditions of the 82C200, 8xC592 and 8xCE598. Philips Semiconductors. AN95092.

[15] Dietmayer K. CAN Bit Timing. Philips Semiconductors. Technical Report HAI/TR9708, 1997.

[16] Dietmayer K. Overberg K. W. CAN Bit Timing Requirements, SAE Paper 970295, 1997

[17] Hank P. PeliCAN: A New CAN Controller Supporting Diagnosis and System Optimization. 4th International CAN Conference, Berlin, Germany, October 1997.

[18] Florian Hartwich, Armin Bassemir. The Configuration of the CAN Bit Timing. Robert Bosch GmbH, Abt. K8/EIS,Tübinger Straße 123, 72762 Reutlingen.6th International CAN Conference 2nd to 4th November, Turin (Italy).

[19] CAN Physical Layer for Industrial Applications. CIA Draft Standard 102. 20 April 1994.

[20] CANopen. CAN in automation. www.cia.org.

[21] M J Schofield. The Controller Area Network (CAN). http://www.omegas.co.uk

[22] Ghosh Indradeep (Princeton Univ), Raghunathan Anand , Jha Niraj K. Design for testability technique for RTL circuits using control/data flow extraction. IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers Nov 10-14 1996 1996 Sponsored by: IEEE IEEE p 329-336.

[23] Livani MA, Kaiser J, Jia WJ. Scheduling hard and soft real-time communication in a controller area network. CONTROL ENGINEERING PRACTICE 7: (12) 1515-1523 1999 Dec.

[24] Anon . CANbus brings new thinking to the control of an injection moulding machine. British Plastics and Rubber Nov 1998 MCM Publishing p 13-16 0307-6164.

Acknowledgments

I am greatly indebted to many teachers, classmates and helpers in all phases of the task of preparing this edition:

To:

Professor DAI Minsen

Dr. TAN Nanlin

A.P. SU Shuqiang

A.P. JIAO Fengchuan

And all the stuff members of the Ergonomics Laboratory:

Ms. XU Chenxiao

Miss WANG Zhaojuan

Mr. YU Yinquan

Mr. ZHA Weixiang

And my friends:

Mr. DING Qun

Miss SHI Lili

Who have gave me a lot of support in these two years.

NI Xiaodong

January, 2001