LIN 入门

概要

本资料面向 LIN 总线初学者，对什么是 LIN，LIN 的特征，物理层、协议层及应用层相关规定进行说明。本资料主要是针对 LIN2.1 讲解。

使用注意事项

本资料对 LIN 协会所提出的 LIN 总线的概要及协议进行了归纳，可作为实际应用的参考资料，对于具有 LIN 功能的产品不负任何责任。

目录

1. LIN 是什么?

LIN 是 Local Interconnect Network 的缩写，是基于 UART/SCI(Universal Asynchronous Receiver-Transmitter / Serial Communication Interface，通用异步收发器/串行通信接口)的低成本串行通信协议。可用于汽车、家电、办公设备等多种领域。本文主要针对 LIN 在分布式的汽车电子网络系统中的应用。

1996 年，Volvo和Volcano通讯(VCT)为Volvo S80 系列开发了一种基于UART/SCI的协议，即Volcano Lite。1997 年，Motorola与Volvo和VCT合作，帮助它们改进Volcano Lite协议以满足各种不同需求(比如无需晶振的从机设备自动同步)，并制定可以支持各种半导体产品的开放标准。1998 年 12 月，Audi、BMW、Daimler Chrysler和Volkswagen也加入进来，由此形成了LIN协会(http://www.lin-subbus.org)。开发LIN标准的目的在于适应分层次车内网络在低端(速度和可靠性要求不高、低成本的场合)的需求。LIN经历了几个版本的发布和更新，如表 1.1 所示。

表 1.1 LIN 协议历史

发布时间	版本
1999/07	1.0 版发布。
2000/03	1.1 版发布。
2000/11	1.2 版发布。
2002/12	1.3 版发布，主要对物理层进行修改，提高了节点之间的兼容性。
2003/09	2.0 版发布，支持配置和诊断的标准化，规定了节点性能文件等。
2006/11	2.1 版发布，澄清了部分内容，修正了配置部分，将传输层和诊断部分独立成章。

图 1.1 为 LIN 在汽车中的应用，主要用于车身系统。

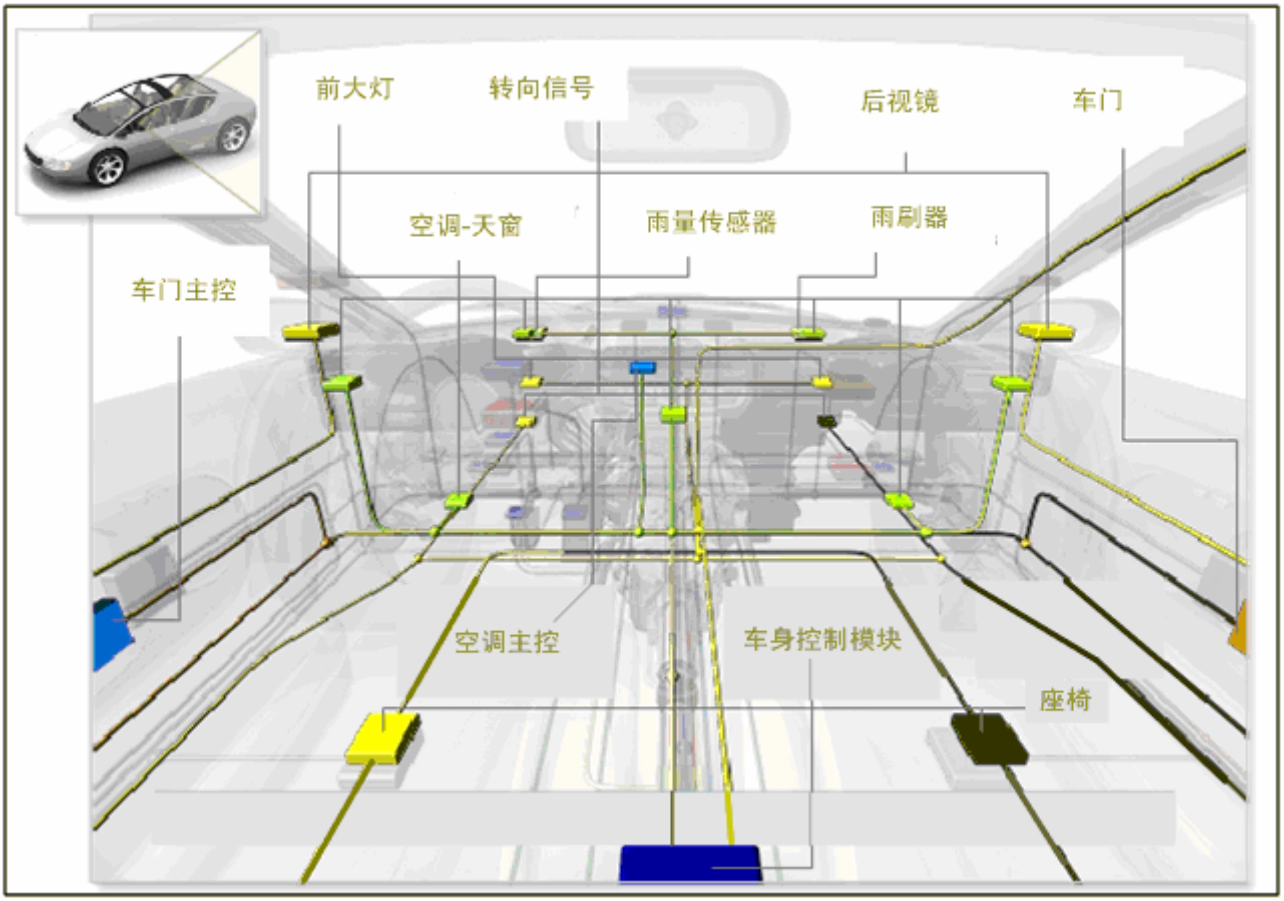

图 1.1 LIN 在汽车中的应用

1.1 LIN 子网(Cluster)与节点(Node)

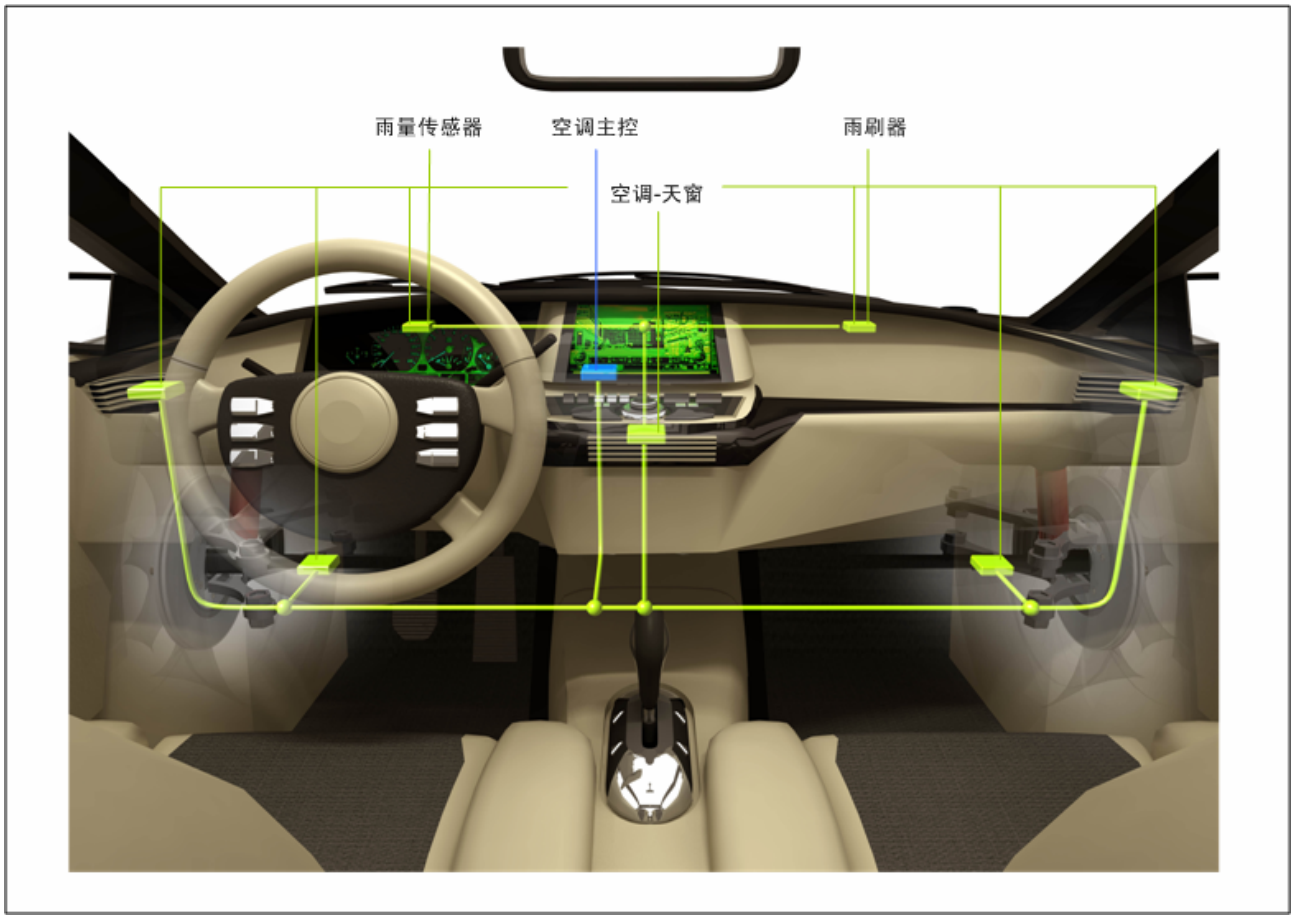

图 1.2 LIN 的典型应用示例

图 1.2 为一典型的车载 LIN 通信子网(注 1)，黄色方块为 LIN 的从机节点，蓝色方块为 LIN 的主机节点，一个节点即一个 LIN 接口(注 2)。LIN 网络与主干线 CAN(Controller Area Network，控制器局域网)总线相连时，需要加入 CAN-LIN 网关，一般由主机节点来充当。LIN 与上层网络相连时的示意图参照图 1.3。

注：1. 由于 LIN 网络在汽车中一般不独立存在，经常与上层网络(如 CAN)相连，因此子网的概念是相对于上层网络而言。在不强调与上层网络相连的情况下，后面也称作 LIN 网络。

2. 一个节点不一定对应一个 ECU(Electronic Control Unit，电子控制单元)，因为一个 ECU 可能提供多个 LIN 接口，并且这些接口可能连接到不同的 LIN 通信子网中。

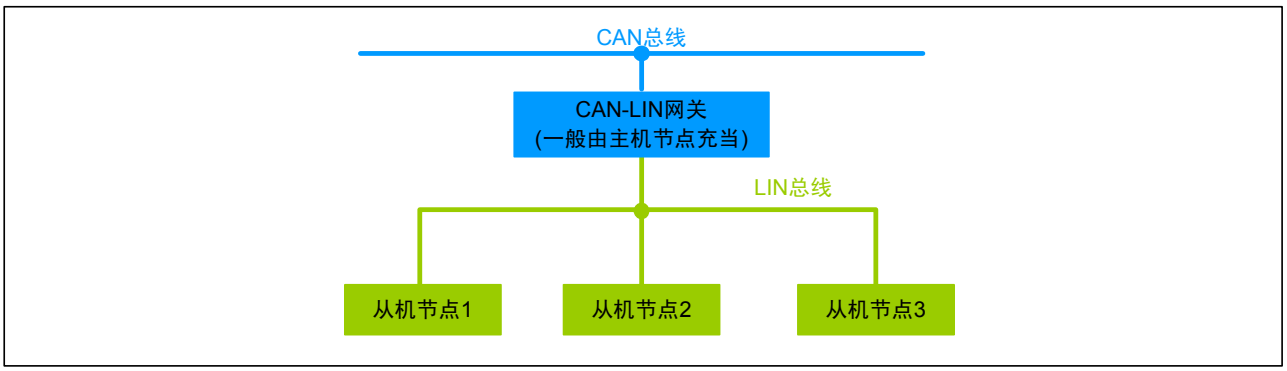

图 1.3 LIN 与上层网络连接示意图

节点可以抽象为如图 1.4 所示。

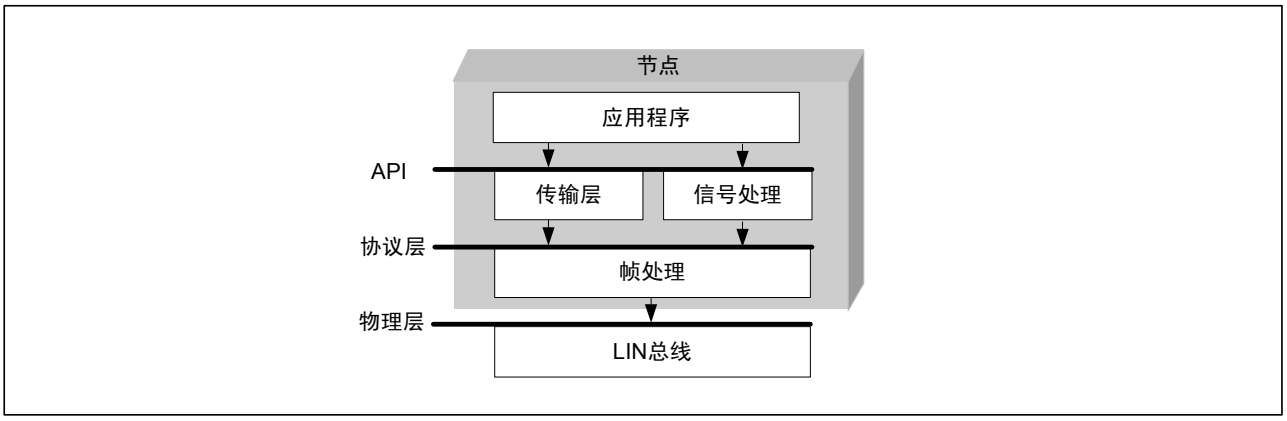

图 1.4 LIN 节点的构成

节点应用层向下层传输信号和消息。信号和消息位于帧中的数据段，是节点向其他节点传达的实质信息。它们之间的区别在于信号封装于信号携带帧 (帧 ID 范围在 0x00～0x3B 之间，参照 3.1.3 节表 3.1)中，用于在运行状态传递上层发生的事件，如温度传感器的测量结果等。消息封装于诊断帧(帧 ID 为 0x3C 或 0x3D，参照 3.1.3 节表 3.1)中，是有固定格式、最大长度不超过 4095 字节的信息，例如第 5 章介绍的服务请求。应用程序通过信号处理实现信号的传递，通过传输层实现消息的传递。

1.2 主/从机节点与主/从机任务

LIN 的拓扑结构为单线总线，应用了单一主机多从机的概念。总线电平为 12V，传输位速率(Bitrate)最高为 20kbps。由于物理层限制，一个 LIN 网络最多可以连接 16 个节点，典型应用一般都在 12 个节点以下，主机节点有且只有一个，从机节点有 1 到 15 个。

主机节点(Master Node)包含主机任务(Master Task)和从机任务(Slave Task)，从机节点(Slave Node)只包含从机任务，如图 1.5 所示。

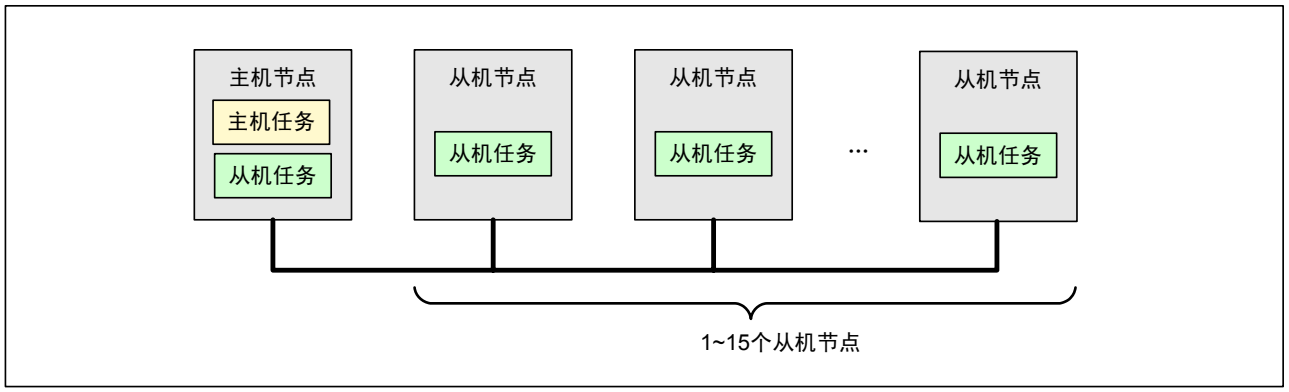

图 1.5 LIN 总线拓扑图

主机任务负责：

(1) 调度总线上帧的传输次序；

(2) 监测数据，处理错误；

(3) 作为标准时钟参考；

(4) 接收从机节点发出的总线唤醒命令。

从机任务不能够主动发送数据，需要接收主机发送的帧头(帧的起始部分，参照 3.1 节的图 3.1)，根据帧头所包含的信息(这里指帧 ID，详细内容参照 3.1.3 节)判断：

(1) 发送应答(帧中除帧头外剩下的部分，参照 3.1 节的图 3.1)；

(2) 接收应答；

(3) 既不接收也不发送应答。

2. LIN 的特点

LIN 具有以下特点：

(1) 网络由一个主机节点和多个从机节点构成。

(2) 使用 LIN 可以大幅度的削减成本，表现在以下方面：

- 开放型规范：规范可以免费从官方网站获得。
- 硬件成本削减：基于普通 UART/SCI 接口的低成本硬件实现，无需单独的硬件模块支持；从机节点无需高精度时钟就可以完成自同步；总线为一根单线电缆。
- 装配成本削减：LIN 采用了工作流(Work Flow)和现成节点(Off-the-shelf Node)的概念，将网络装配标准化，并可通过 LIN 传输层进行再配置。
- 缩短软件开发周期：LIN 协议将 API(Application Programming Interface，应用编程接口)标准化。

(3) 信号传输具有确定性，传播时间可以提前计算出，参照 3.1.6 节。

(4) LIN 具有可预测的 EMC(ElectroMagnetic Compatibility，电磁兼容性)性能，参照 4.7 节。为了限制 EMI(ElectroMagnetic Interference，电磁干扰)强度，LIN 协议规定最大位速率为 20kbps。

(5) LIN 提供信号处理、配置、识别和诊断四项功能，参照 5.2.1 节的图 5.2。

3. LIN 协议层

本章内容介绍了帧、进度表、主/从机的状态机实现、LIN 网络的休眠(Sleep)/唤醒(Wakeup)和状态管理等，对应着 LIN 规范的以下部分：

- LIN Protocol Specification

3.1 帧的结构

帧(Frame)包含帧头(Header)和应答(Response)两部分。主机任务负责发送帧头；从机任务接收帧头并对帧头所包含信息进行解析，然后决定是发送应答，还是接收应答，还是不作任何反应。帧在总线上的传输如图 3.1 所示。

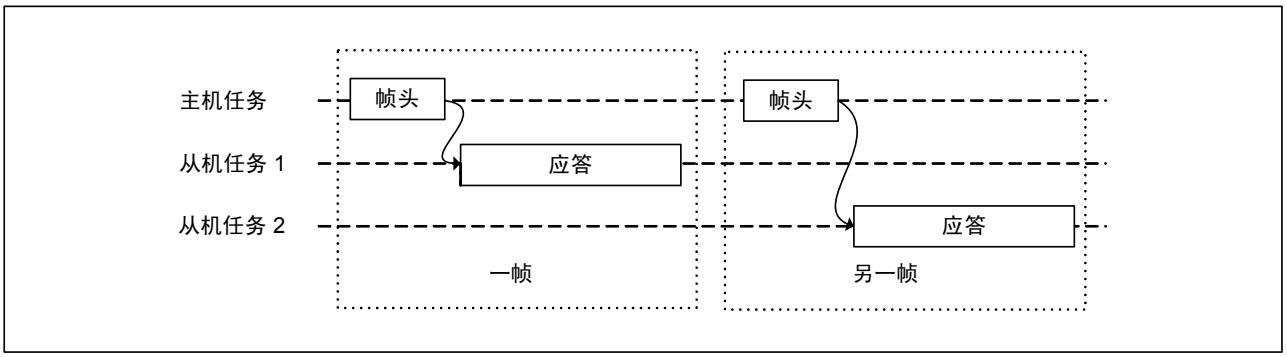

图 3.1 帧在总线上的传输

帧头包括同步间隔段、同步段以及PID(Protected Identifier，受保护ID)段，应答包括数据段和校验和段，如图 3.2 所示，其中值“0”为显性电平(Dominant)，值“1”为隐性电平(Recessive)，总线上实行“线-与”：当总线上有大于等于一个节点发送显性电平时，总线呈显性电平；所有的节点都发送隐性电平或不发送信息(不发送任何信息时总线默认呈隐性电平)时，总线才呈现隐性电平，即显性电平起主导作用。图中帧间隔为帧之间的间隔；应答间隔为帧头和应答之间的间隔；字节间间隔包括同步段和受保护ID段之间的间隔、数据段各字节间之间的间隔以及数据段最后一个字节和校验和段之间的间隔。下面对帧头和应答的各部分进行详细说明。

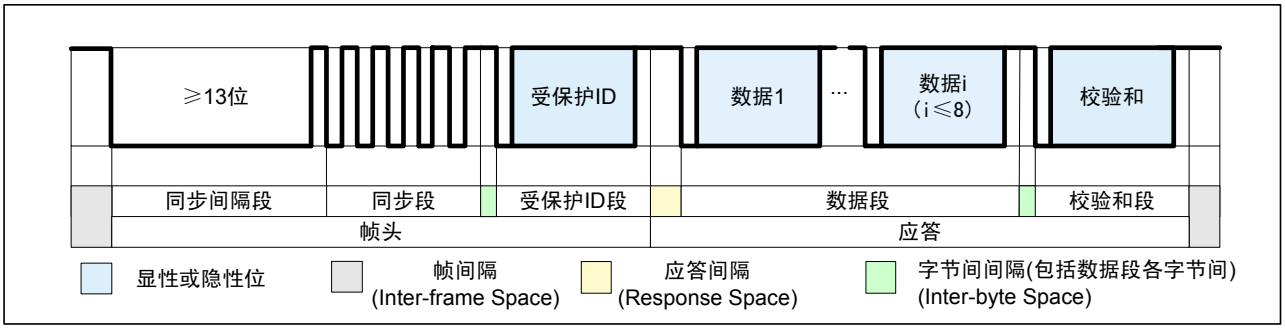

图 3.2 帧的结构

3.1.1 同步间隔段(Break Field)

同步间隔段由同步间隔(Break)和同步间隔段间隔符(Break Delimiter)构成，如图 3.3 所示。同步间隔是至少(注 1)持续 13 位(以主机节点的位速率为准)的显性电平，由于帧中的所有间隔或总线空闲时都应保持隐性电平，并且帧中的任何其它字段都不会发出大于 9 位的显性电平，因此同步间隔可以标志一个帧的开始。同步间隔段的间隔符是至少持续 1 位的隐性电平。

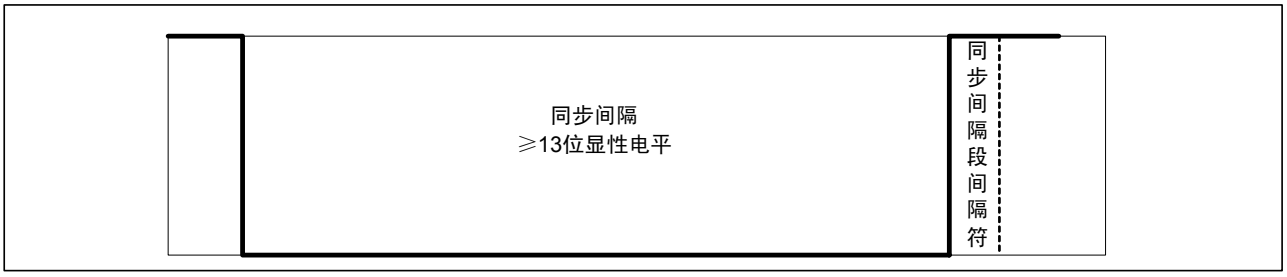

图 3.3 同步间隔段

从机任务接收帧头的同步间隔段时，以该从机任务所在节点的位速率为准，当检测总线上出现持续 11 位(注 2)的显性电平时，认为是帧的开始。当从机节点使用精度较高的时钟时，识别阈值可以选择 9.5 位(注 3)。

协议没有规定同步间隔段的发送和检测方法。

注：1. 发送显性电平的下限为 13 位，上限应保证帧的最大传输时间$T_{Header_Maximum}$(参照 3.1.6 节)在规定范围之内。

2. 参照 4.6 节的表 4.3，当从机节点选择的时钟(精度不高的时钟)在容限范围内(±14%)时，(13 - 11.18) / 13 = 14%，即是说当处于最差情况下(时钟相差 14%)时，从机任务按照自身时钟测量的主机节点发送的 13 位显性电平不会低于 11.18 位，若识别阈值高于 11.18 位，那么当选用 14%的时钟时，就会出现主机发送同步间隔，而从机检测不到的情形。由于在除同步间隔段以外，帧中任何其余部分都不会发送超过 9 位的显性电平(可以参照本章后面几节的内容)，(10.26 - 9) / 9 = 14%，即是说判断阈值必须大于 10.26 位，否则可能把帧中其余部分误判作为同步间隔段。综上，识别阈值为 11 位显性电平。

3. 参照 4.6 节的表 4.3，当从机节点选择的时钟(精度较高的时钟)在容限范围内(±1. 5%)时，按照上面注 2 的计算，识别阈值应在 9.135 位(由(9.135 - 9) / 9 = 1.5%计算而来)到 12.805 位(由(13 – 12.805) / 13 = 1.5%计算而来)之间。具体设定阈值会随着所选时钟的精度，取值范围在 9.135 位到 12.805 位之间浮动。

3.1.2 同步段(Sync Byte Field)

在介绍同步段之前，首先介绍一下字节域(Byte Field)的概念，字节域包括 1 位起始位(Start Bit，显性) + 8 位数据位 + 1 位停止位(Stop Bit，隐性)，是一种标准 UART 数据传输格式，如图 3.4 所示。在 LIN 的一帧当中，除了上一节讲述的同步间隔段，后面的各段都是通过字节域的格式传输的。在 LIN 帧中，数据传输都是先发送 LSB(Least Significant Bit，最低有效位)，最后发送 MSB(Most Significant Bit，最高有效位)。

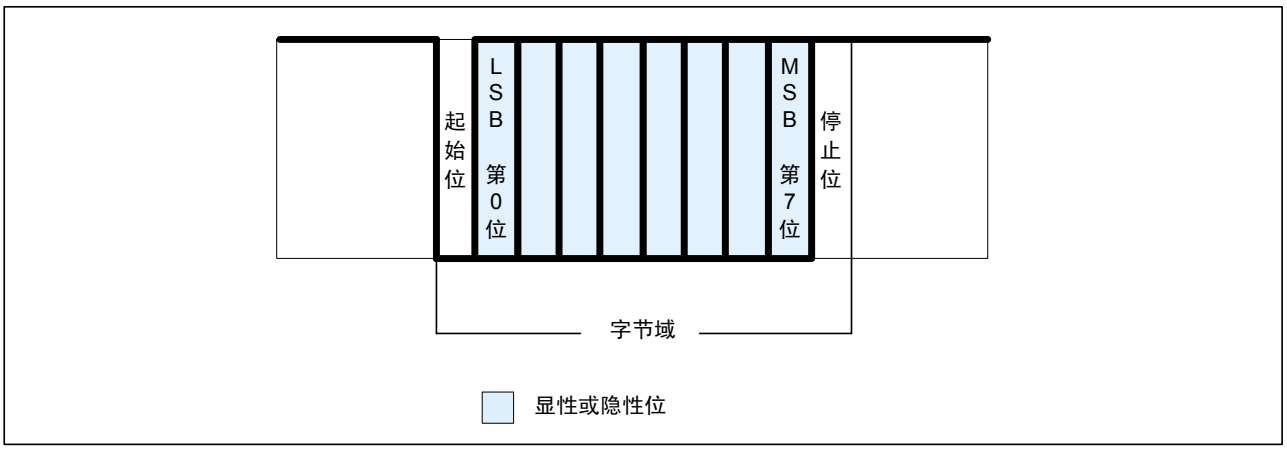

图 3.4 字节域

LIN 同步以下降沿为判断标志，采用字节 0x55(转换为二进制为 01010101b)。同步段的字节域如图 3.5 所示。

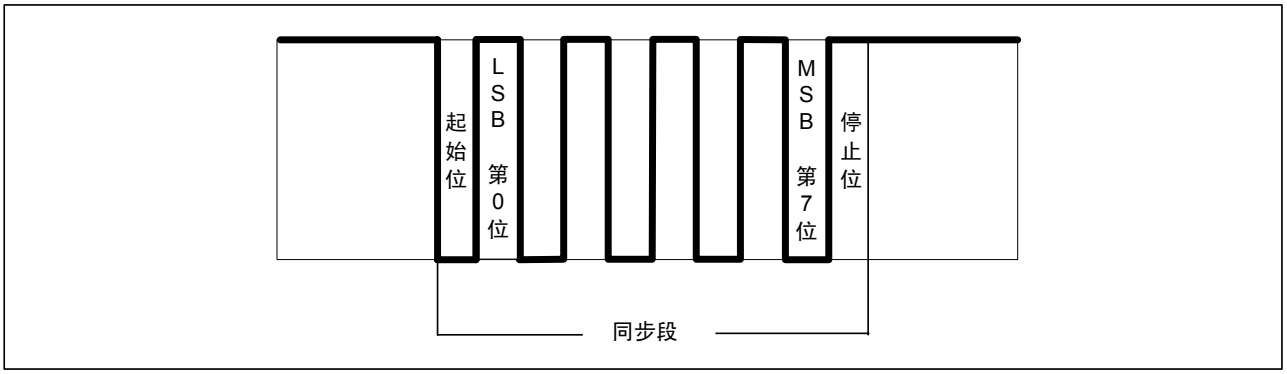

图 3.5 同步段

从机节点可以不采用精度高的时钟，而采用片上振荡器等精度和成本相对较低的时钟，由此带来的与主机节点时钟产生的偏差，需要通过同步段进行调整，调整的结果是使从机节点数据的位速率与主机节点一致。同步段用于同步的基准时钟为主机节点的时钟。从机节点通过接收主机节点发出的同步段，计算出主机节点位速率，根据计算结果对自身的位速率重新作调整。计算公式如下：

$$1\text{位时间} = \frac{\text{第7位的下降沿时刻} - \text{起始位的下降沿时刻}}{8}$$

通过计算，可以得到主机节点实际传输 1 位所用的时间，即位速率。

3.1.3 受保护 ID 段(Protected Identifier Field)

受保护 ID 段的前 6 位叫作帧 ID(Frame ID)，加上两个奇偶校验位后称作受保护 ID。如图 3.6 所示。

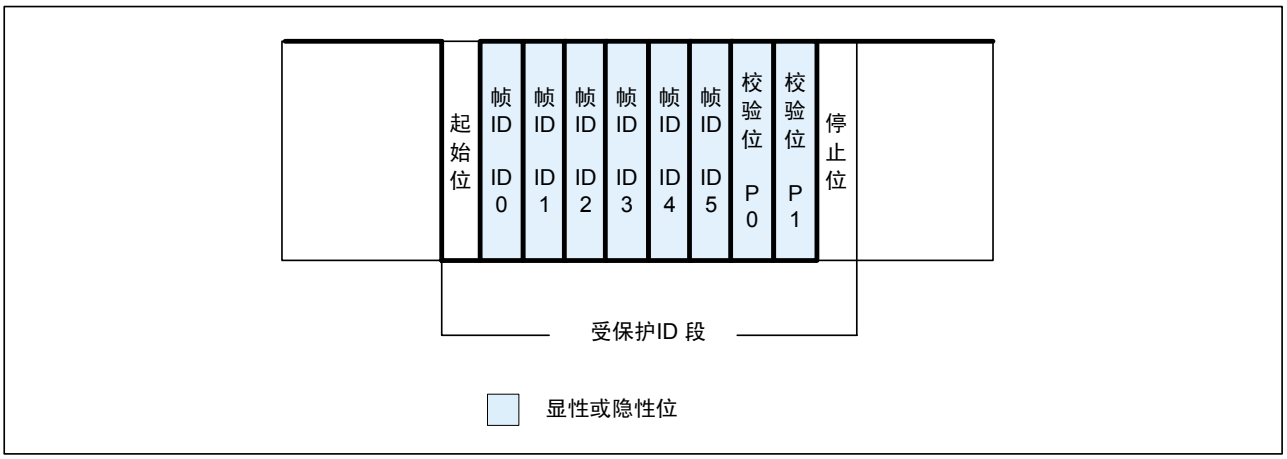

图 3.6 受保护 ID 段

帧 ID 的范围在 0x00～0x3F 之间，共 64 个。帧 ID 标识了帧的类别和目的地。从机任务对于帧头作出的反应(接收/发送/忽略应答部分)都是依据帧 ID 判断的。如果帧 ID 传输错误，将会导致信号无法正确到达目的地，因此引入奇偶校验位。校验公式如下，其中“⊕”代表“异或”运算，“¬”代表“取非”运算。

$$P0 = ID0 \oplus ID1 \oplus ID2 \oplus ID4$$

$$P1 = \neg (ID1 \oplus ID3 \oplus ID4 \oplus ID5)$$

由公式可以看出，PID 不会出现全 0 或全 1 的情况，因此，如果从机节点收到了“0xFF”或“0x00”，可判断为传输错误。

依据帧 ID 不同将帧进行分类，如表 3.1 所示，对各种类型的详细说明参照 3.2 节。

表 3.1 帧的类型

帧的类型		帧 ID
信号携带帧	无条件帧	0x00 ～ 0x3B
	事件触发帧	
	偶发帧	
诊断帧	主机请求帧	0x3C
	从机应答帧(注 1)	0x3D
保留帧		0x3E，0x3F

注：1. 从机应答帧是一个完整的帧，与帧结构中的“应答”(帧的一部分)不同，注意区别。

3.1.4 数据段(Data Field)

节点发送的数据位于数据段，包含 1 到 8 个字节(注 1)，先发送编号最低的字节 DATA1，编号依次增加，如图 3.7 所示。

数据段包含了两种数据类型，信号(Signal)和诊断消息(Diagnostic messages)。

信号(Signal)由信号携带帧传递，一个帧 ID 对应的数据段可能包含一个或多个信号。信号更新时要保证其完整性，不能只更新一部分。一个信号通常由一个固定的节点发出，此节点称为该信号的发布节点(Publisher)；其余的一个或多个节点接收，它们称为信号的收听节点(Subscriber)(注 2)。

诊断消息(Diagnostic message)由诊断帧传递，对消息内容的解析由数据自身和节点状态决定。

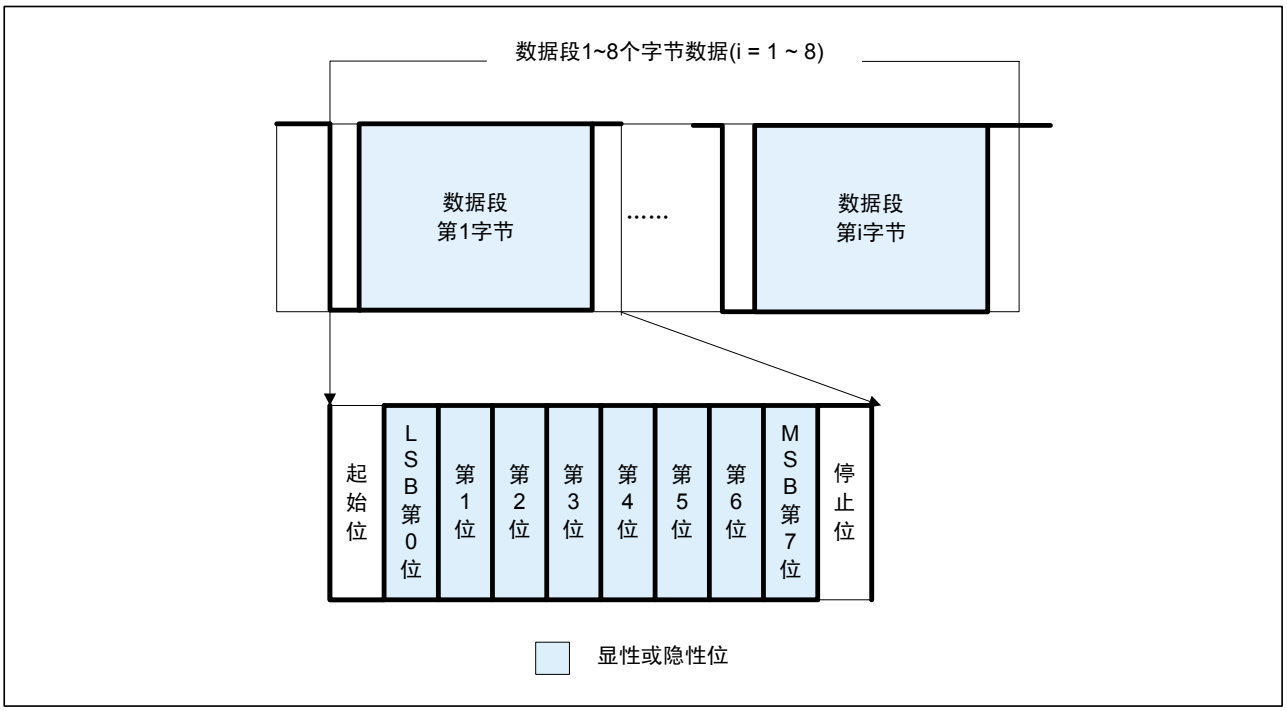

图 3.7 数据段

注：1. 协议没有规定帧中的哪一部分显示数据长度码的信息，数据的内容和长度是由系统设计者根据帧 ID 事先约定好的。

2. 总线上的数据是以广播形式被发送到总线上的，任何节点均能接收，但并非所有信号对每个节点都有用。收听节点接收帧的应答是因为该节点的应用层会使用这些信号，而对于其余节点，由于用不到这些信号，所以没有必要作接收处理，将忽略帧的应答部分。发布和收听由哪个节点进行完全根据应用层的需要由软件或配置工具实现。一般情况下，对于一个帧中的应答，总线上只存在一个发布节点，否则就会出现错误。事件触发帧例外，可能存在零个、一个或多个发布节点，参照 3.2.2 节。

3.1.5 校验和段(Checksum Field)

校验和段是对帧中所传输的内容进行校验，如图 3.8 所示。

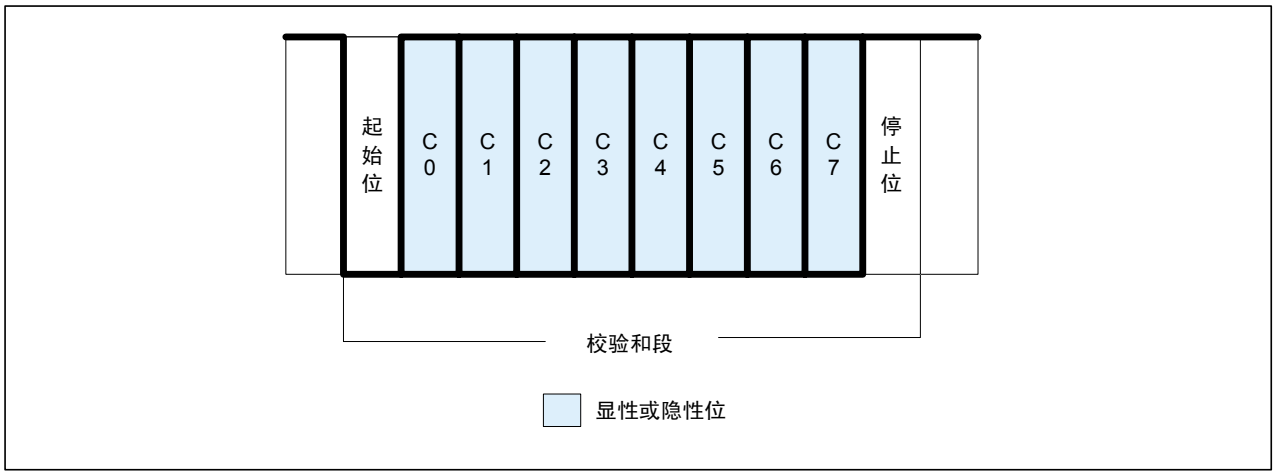

图 3.8 校验和段

校验和分为标准型校验和(Classic Checksum)及增强型校验和(Enhanced Checksum)，如表 3.2 所示。

表 3.2 校验和类型

校验和类型	校验对象	适用场合
标准型校验和	数据段各字节	诊断帧，与 LIN1.x 从机节点通信。
增强型校验和	数据段各字节以及受保护 ID	与 LIN2.x 从机节点通信(诊断帧除外)。

采用标准型校验和还是增强型校验和由主机节点管理，发布节点和各收听节点根据帧 ID 来判断采用哪种校验和。

校验方法为将校验对象的各字节作带进位二进制加法(每当结果大于等于 256 时就减去 255)，并将所得最终的和逐位取反，以该结果作为要发送的校验和。接收方根据校验和类型，对接收数据作相同的带进位二进制加法，最终的和不取反，并将该和与接收到的校验和作加法，如果结果为 0xFF，则校验和无误，这在一定程度上保证了数据传输的正确性。

例如：采用标准型校验和，Data1 = 0x4A，Data2 = 0x55，Data3 = 0x93，Data4 = 0xE5，计算方法如表 3.3 所示：

表 3.3 校验和计算举例

发送方/接收方										
	和	进位	D7	D6	D5	D4	D3	D2	D1	D0
0x4A	0x4A		0	1	0	0	1	0	1	0
+0x55 = 加上进位	0x9F 0x9F	0	1 1	0 0	0 0	1 1	1 1	1 1	1 1	1 1
+0x93 = 加上进位	0x132 0x33	1	0 0	0 0	1 1	1 1	0 0	0 0	1 1	0 1
+0xE5 = 加上进位	0x118 0x19	1	0 0	0 0	0 0	1 1	1 1	0 0	0 0	0 1
取反	0xE6		1	1	1	0	0	1	1	0
接收方										
0x19(重新计算的校验和，不取反) + 0x E6(接收到的校验和) =	0xFF		1	1	1	1	1	1	1	1

3.1.6 帧传输时间的计算

帧(有关帧的结构参照 3.1 节的图 3.2)在总线上传输的时间计算如表 3.4 所示。其中，$T_{Frame_Maximum}$为帧在总线上传输的最大时间；$T_{Header_Maximum}$为帧头在总线上传输的最大时间；$T_{Response_Maximum}$为应答在总线上传输的最大时间；$T_{Header_Nominal}$为帧头额定传输时间：同步间隔段(包含同步间隔和同步间隔段间隔符)的最小传输时间 + 同步段传输时间 + 受保护ID段传输时间；帧头的余量T_{Header_Rest}包含字节间间隔，规定为帧头额定传输时间的 0.4 倍；$T_{Response_Nominal}$为应答额定传输时间：数据段传输时间 + 校验和段传输时间；应答的余量$T_{Response_Rest}$包含应答间隔以及字节间间隔，规定为应答额定传输时间的 0.4 倍；N_{data}表示数据段包含N个字节。

表 3.4 帧在总线上传输时间的计算

$T_{Frame\_Maximum}$	$T_{Header\_Maximum}$					$T_{Response\_Maximum}$		
	$T_{Header\_Nominal}$				T_{Header_Rest}	$T_{Response\_Nominal}$		$T_{Response\_Rest}$
	同步间隔(最小值)	同步间隔段间隔符(最小值)	同步段	受保护 ID 段	40% × $T_{Header\_Nominal}$	数据段	校验和	40% × $T_{Response\_Nominal}$
时间(T_{bit})	13	1	10	10		10× N_{Data}	10	

3.1.7 帧在总线上的传输波形

帧在总线上的传输波形示例如图 3.9 所示。

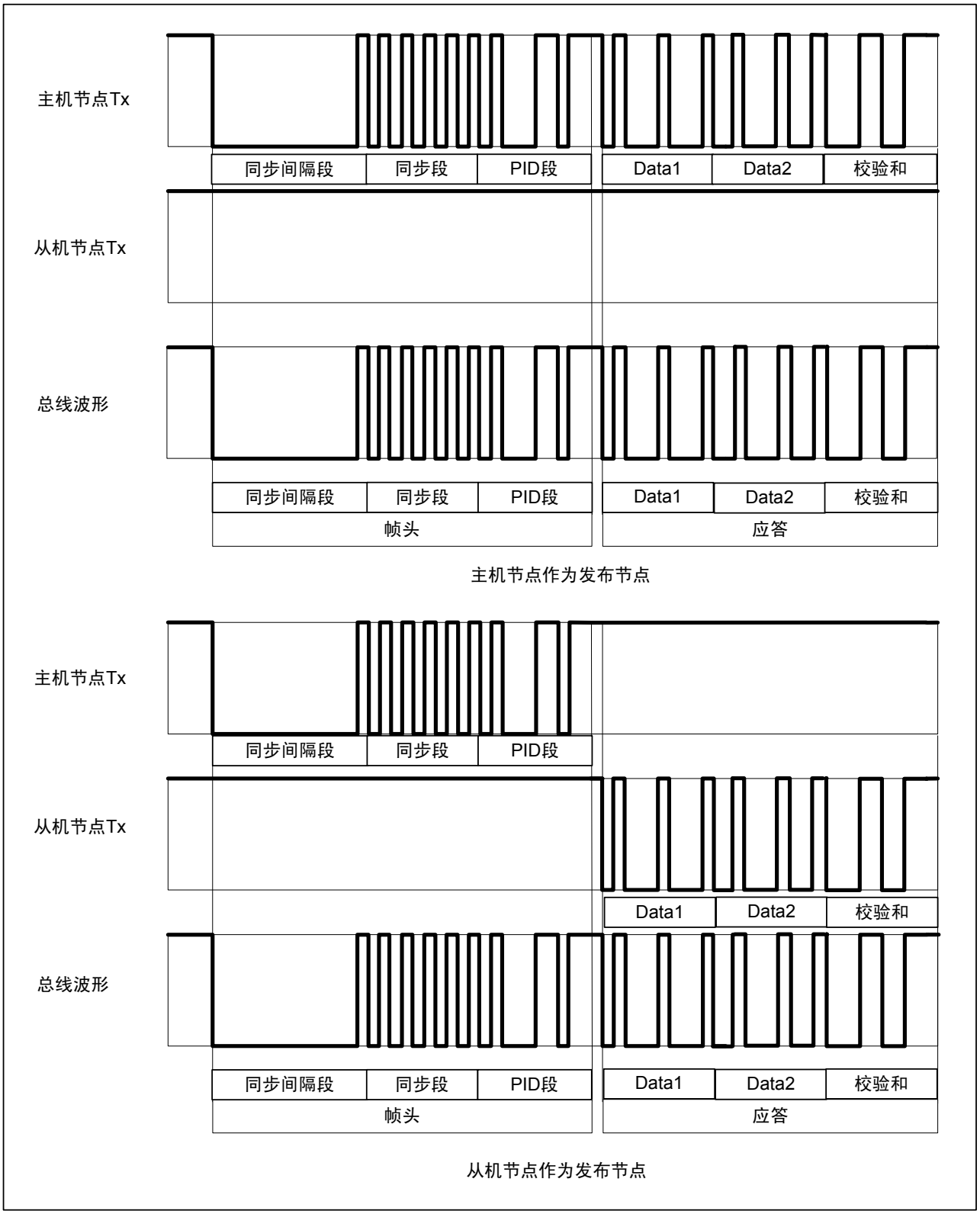

图 3.9 帧在总线上的传输波形

3.2 帧的类型

下面对 3.1.3 节表 3.1“帧的类型”中各种帧进行详细说明。

3.2.1 无条件帧(Unconditional Frame)

无条件帧是具有单一发布节点，无论信号是否发生变化，帧头都被无条件应答的帧。

无条件帧在主机任务分配给它的固定的帧时隙(参照 3.3 节)中传输。总线上一旦有帧头发送出去，必须有从机任务作应答(即无条件发送应答)，如图 3.10 所示，其中列出的帧 ID 的值只是为了举例说明，协议并未强制规定。

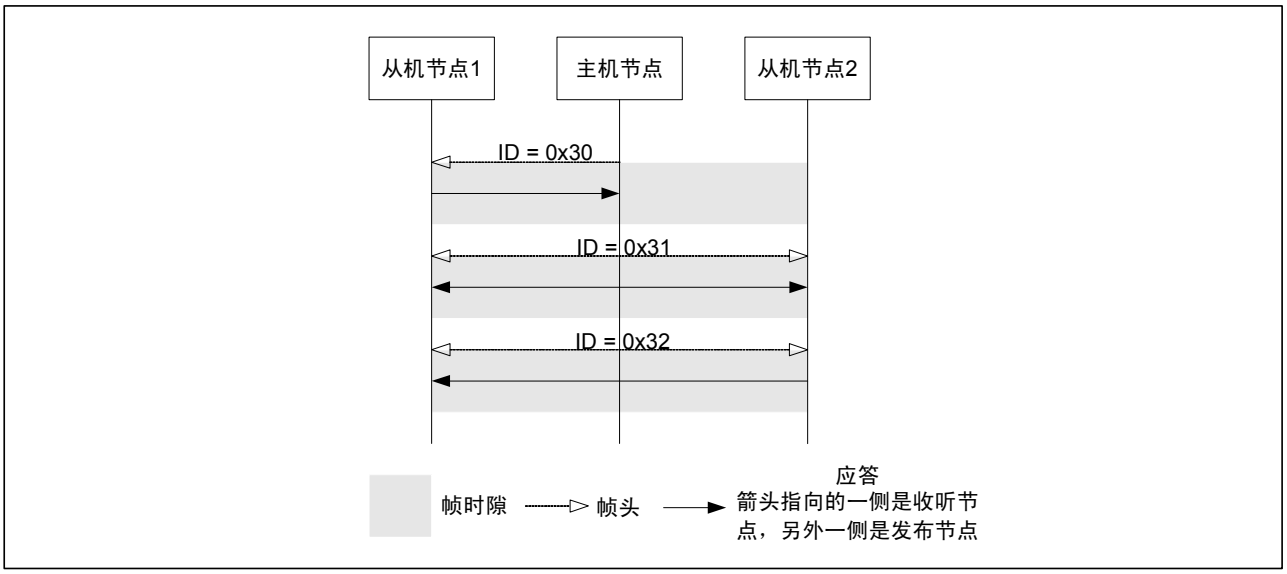

图 3.10 无条件帧

帧 ID = 0x30 应答部分的发布节点为从机节点 1，收听节点为主机节点。典型应用如从机节点 1 向主机节点报告自身某信号的状态。

帧 ID = 0x31 应答部分的发布节点为主机节点，收听节点为从机节点 1 和从机节点 2。典型应用如主机节点向从机节点发布信息。

帧 ID = 0x32 应答部分的发布节点为从机节点 2，收听节点为从机节点 1。典型应用如从机节点之间彼此通信。

3.2.2 事件触发帧(Event Triggered Frame)

事件触发帧是主机节点在一个帧时隙(参照 3.3 节)中查询各从机节点的信号是否发生变化时使用的帧，当存在多个发布节点时，通过冲突解决进度表(参照 3.3 节)来解决冲突。

当从机节点信号发生变化的频率较低时，主机任务一次次地轮询各个信号会占用一定的带宽。为了减小带宽的占用，引入了事件触发帧的概念。

事件触发帧的典型应用就是轮询四个车门的开关情况。与其利用无条件帧每个车门轮询一遍，不如同时对四个车门进行询问，如果其中一个车门打开了(事件发生)，该车门要对询问作应答，即事件触发的含义。这样做可以减小带宽，但同时会导致两种现象，其一就是没有车门被打开，即无节点应答——事件触发帧允许一帧中只有帧头无应答；另外一种情况就是冲突，即同时有大于等于两个车门被打开，对该问题同时作答——事件触发帧允许两个以上的节点对帧头作应答而不视为错误。当发生冲突时，主机节点需要重新作轮询，这样会增加一些响应时间，但由于事件触发帧本身就用来处理低概率事件，总的来说还是节省了带宽。

原先用作轮询的无条件帧，称为与该事件触发帧关联的无条件帧，即事件触发帧的应答部分是与其关联的无条件帧所提供的应答。当发生冲突时，需要立刻中断当前的进度表(参照 3.3 节)，启动冲突解决进度表(Collision Resolving Schedule)，重新调用这些关联的无条件帧。其中，冲突解决进度表要求包含所有的关联的无条件帧。图 3.11 示例描述了事件触发帧的传输状况。事件触发帧的帧 ID 为 0x10，与其关联的两个无条件帧的帧 ID 分别是 0x11 和 0x12，这些帧 ID 的值只是为了举例说明，协议并未强制规定。

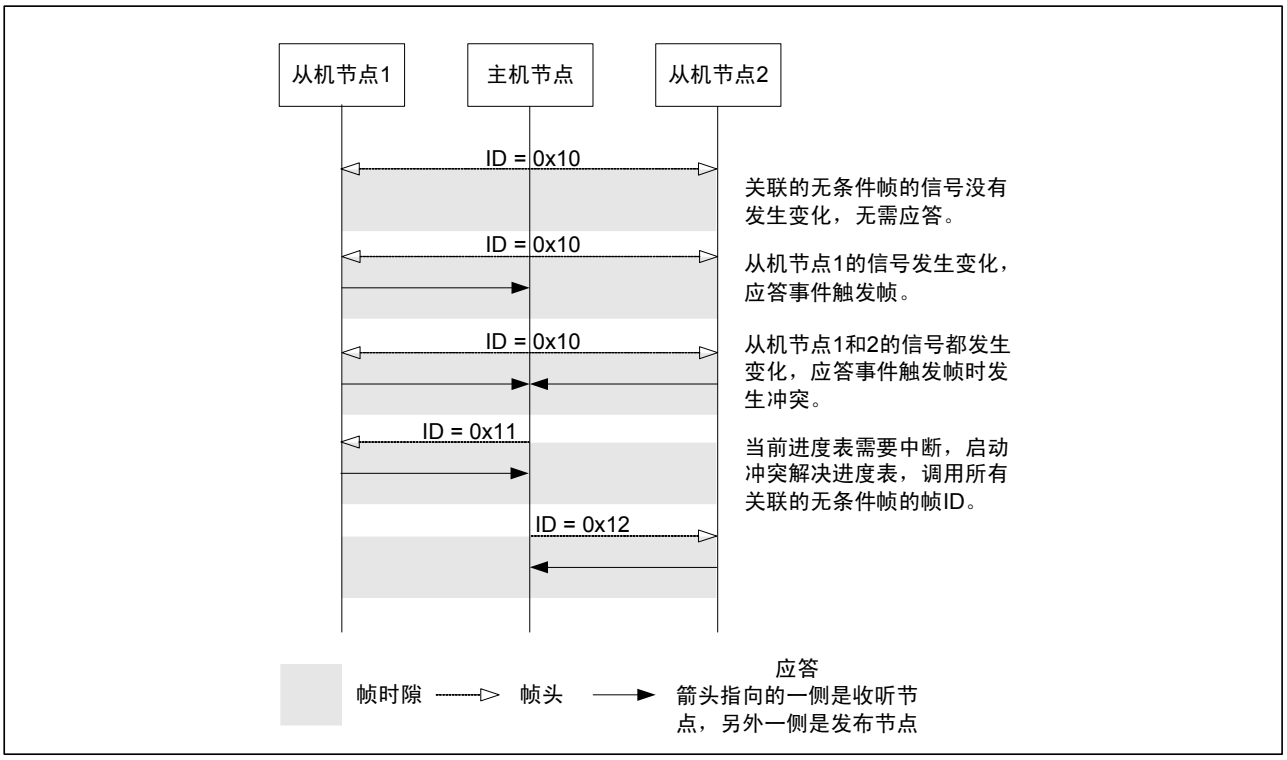

图 3.11 事件触发帧

与事件触发帧关联的多个无条件帧需要满足以下 5 个条件：

(1) 数据段包含的数据字节数等长；

(2) 使用相同的校验和类型；

(3) 数据段的第一个字节为该无条件帧的受保护 ID，这样才能够知道应答是哪个关联的无条件帧发送出来的；

(4) 由不同的从机节点发布；

(5) 不能与事件触发帧处于同一个进度表(参照 3.3 节)中。

3.2.3 偶发帧(Sporadic Frame)

偶发帧是主机节点在同一帧时隙(参照 3.3 节)中当自身信号发生变化时向总线启动发送的帧。当存在多个关联的应答信号变化时，通过事先设定的优先级来仲裁。

与事件触发帧一样，偶发帧的应答也关联了一组无条件帧。规定偶发帧只能由主机节点作为发布节点。偶发帧的传输可能出现三种状况：1)当关联的无条件帧没有信号发生变化时，该时隙(参照 3.3 节)保持沉默，如图 3.12 第一个帧时隙所示，主机节点连帧头都不需要发送；2)当其中一个关联的无条件帧包含的信号发生了变化，则发送该关联的无条件帧的应答部分；3)如果有两个或两个关联的无条件帧包含的信号发生了变化，则按照事先规定好的优先级，优先级较高的关联的无条件帧获得发送权，优先级较低的要等到下一个偶发帧的帧头到来时才能发送应答。由于主机节点是唯一的发布节点，所以主机节点事先就知道各个关联信号的优先级别，这样在传输时就不会产生冲突。

引入偶发帧的目的在于为进度表(参照 3.3 节)增加一些动态特性——当主机节点的信号发生变化时才有通信发生。事件触发帧和偶发帧反映了帧在不同时机(信号变化或未发生变化)的传输状况，引入它们的目的是为了增加通信的灵活性。

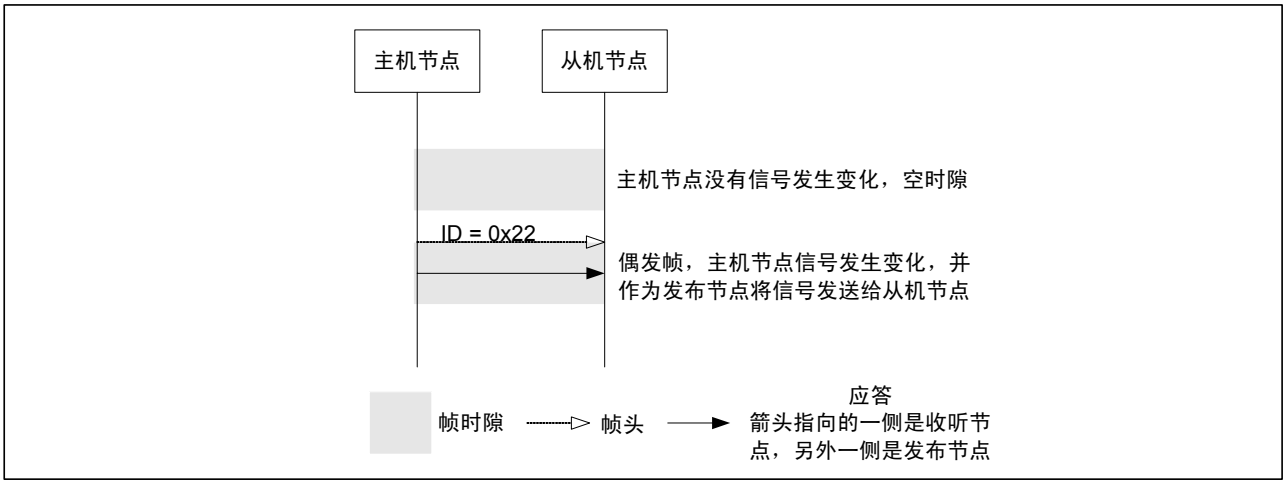

图 3.12 偶发帧

3.2.4 诊断帧(Diagnostic Frame)

诊断帧包括主机请求帧和从机应答帧，主要用于配置、识别和诊断用。主机请求帧(Master Request Frame，MRF)，帧 ID = 0x3C，应答部分的发布节点为主机节点；从机应答帧(Slave Response Frame，SRF)，帧 ID = 0x3D，应答部分的发布节点为从机节点。数据段规定为 8 个字节，一律采用标准型校验和。诊断帧的具体用法参照第 5 章。

3.2.5 保留帧(Reserved Frame)

保留帧的帧 ID 为 0x3E 和 0x3F，为将来扩展用。

3.3 进度表(Schedule)

进度表是帧的调度表，规定总线上帧的传输次序以及各帧在总线上的传输时间。进度表位于主机节点，主机任务根据应用层需要进行调度。进度表可以有多个，一般情况下，轮到某个进度表执行的时候，从该进度表规定的入口处开始顺序执行，到进度表的最后一个帧时，如果没有新的进度表启动，则返回到当前的进度表第一个帧循环执行；也有可能在执行某个进度表当中发生中断，执行另一个进度表后再返回，如事件触发帧的冲突解决过程就是一个典型的例子，如图 3.13 所示。

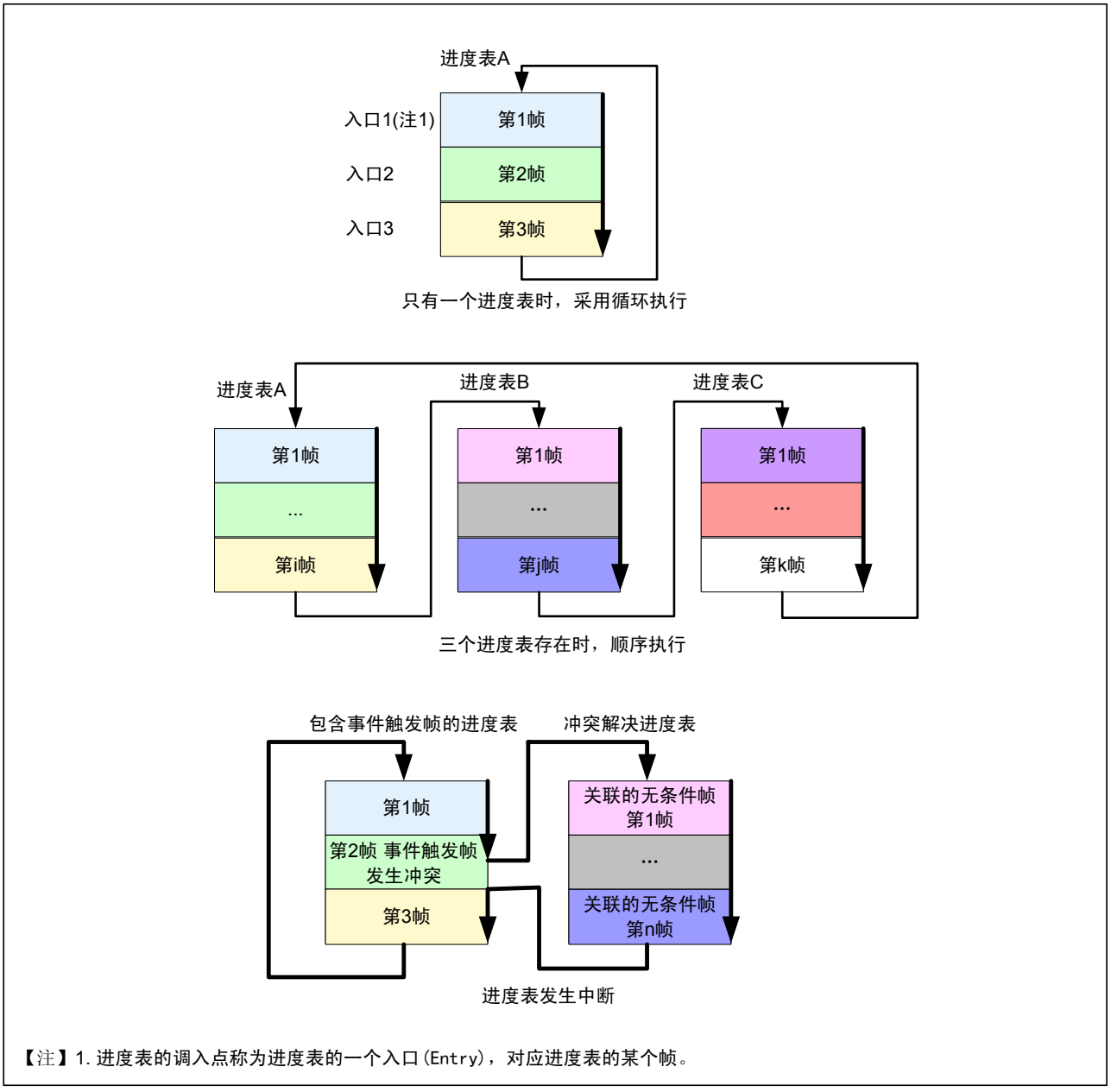

图 3.13 进度表

进度表除规定了帧 ID 的传输次序外，还规定了帧时隙(Frame Slot)的大小。帧时隙是进度表规定的一个帧的帧头起始到下一个的帧的帧头起始的时间。每个帧的帧时隙都可以不同，一个帧时隙对应了进度表的一个入口，如图 3.14 所示，其中 i = 1～8。

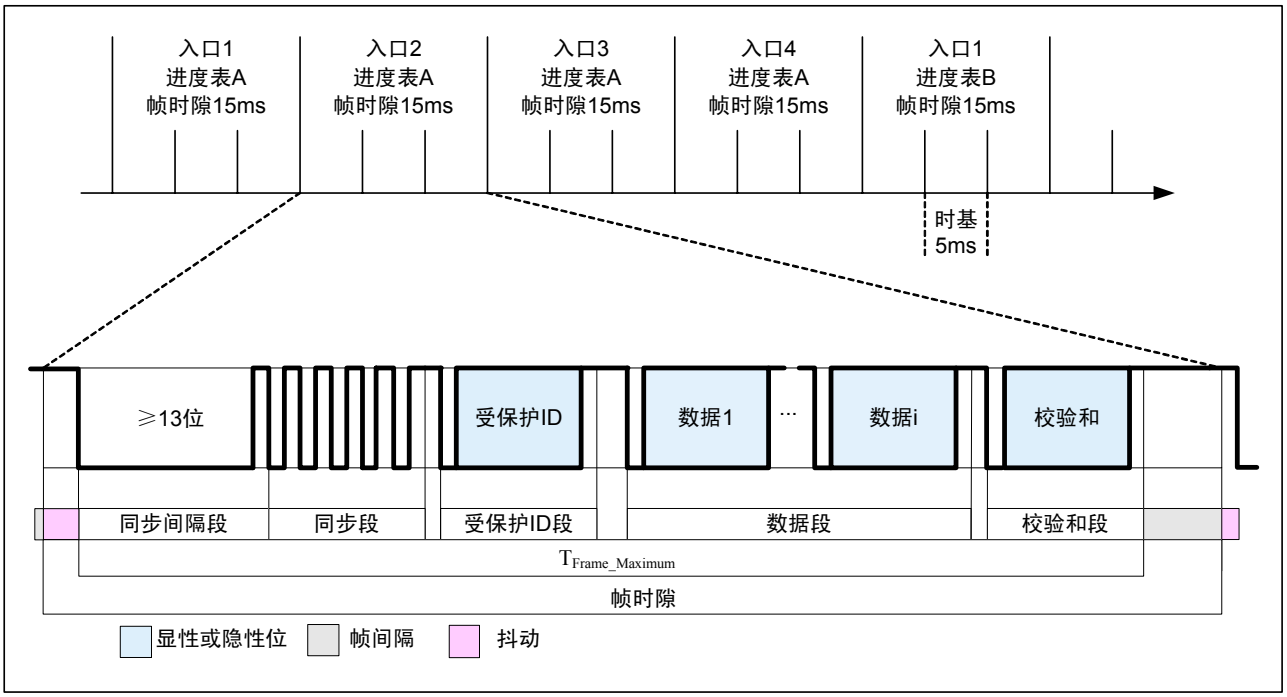

图 3.14 帧时隙

其中$T_{Frame_Maximum}$为帧在总线上传输的最大时间，参照 3.1.6 节的表 3.4。抖动(Jitter)为帧的同步间隔段的下降沿与帧时隙起始时刻相差的时间。时基(Time Base)为LIN子网的最小计时单位，通常设定为 5ms或 10ms。帧时隙必须为时基的整数倍，并且起始于时基的开始时刻(称为时基的节拍(Tick))，切换到另外一个进度表时一定要等到当前帧时隙的结束。

3.4 状态机(State Machine)实现

3.4.1 主机任务的状态机

当进度表启动后，主机任务依次发送同步间隔段、同步段和受保护 ID 段，如图 3.15 所示。

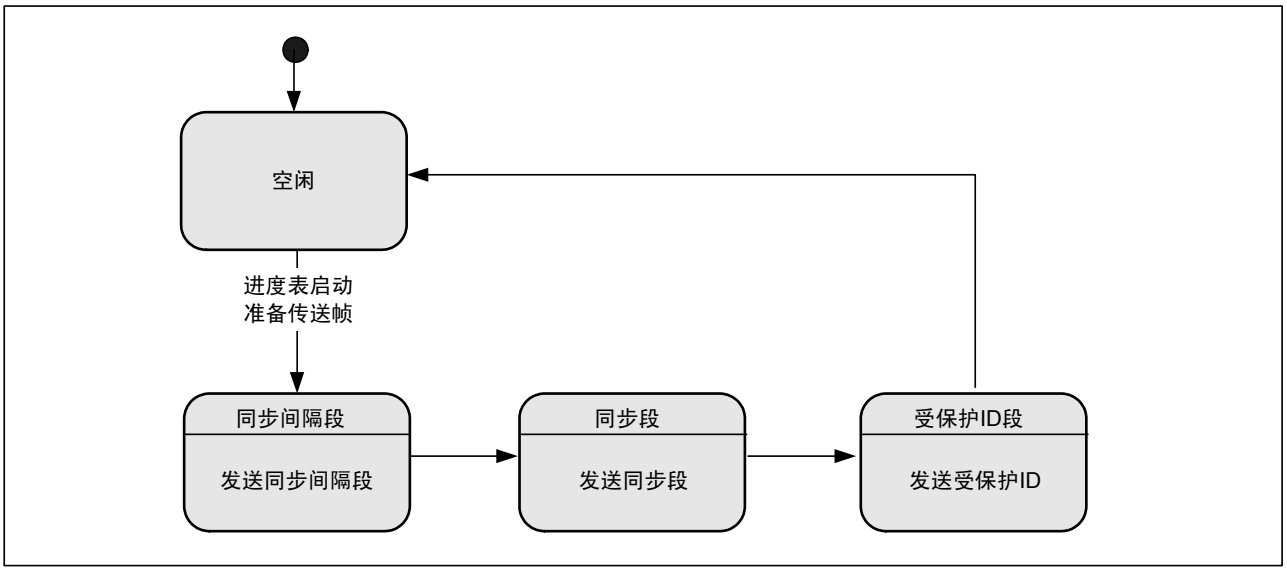

图 3.15 主机任务的状态机

3.4.2 从机任务的状态机

从机任务负责发布或者接听帧的应答。包括两个状态机：

1. 同步间隔段和同步段检查器
2. 帧处理器

从机任务状态机如表 3.5 所示，其中帧处理的状态机如图 3.16 所示。

表 3.5 从机任务状态机

从机任务状态机		说明
检测同步间隔段/同步段序列		要求节点处于任何状态下都能识别出该序列，包括已经检测到序列或进入帧处理的状态。
帧处理	接收并分析 PID	对接收到的受保护 ID 进行分析，按照事先的设计，选择是接收应答部分，还是发送应答部分，或者不接收也不发送。 在这五个子状态中，如果收到同步间隔段/同步段序列，将重新跳到“接收并分析 PID”的子状态，通信不停止，根据需要置位相应的错误标志。
	接收数据	
	接收校验和	
	发送数据	
	发送校验和	

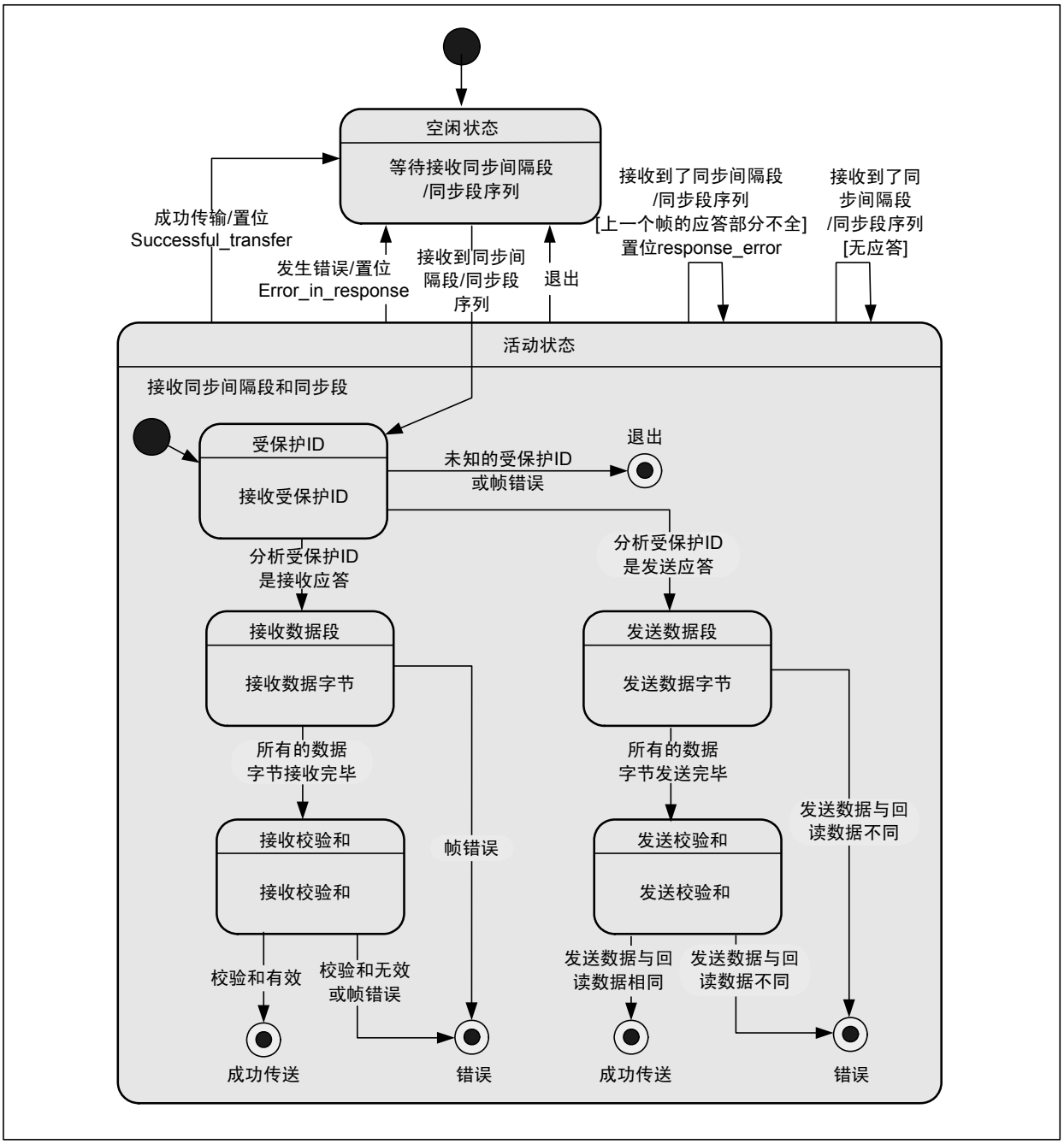

图 3.16 帧处理的状态机

3.5 网络管理

网络管理主要指的是网络的休眠和唤醒管理，如图 3.17 所示。

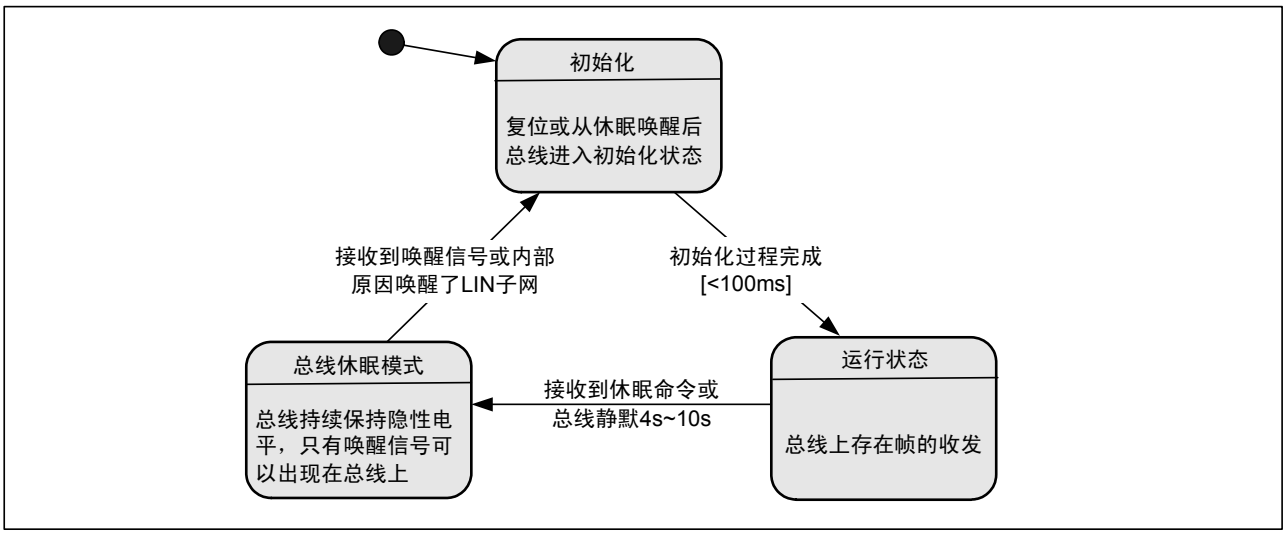

图 3.17 网络管理

3.5.1 唤醒

当总线处于休眠状态时，主/从机节点都可以向总线上发送唤醒信号，唤醒信号持续 250μs～5ms。其余节点(除发送唤醒信号以外的节点)以大于 150μs 为阈值判定唤醒信号。每个从机节点必须在唤醒信号显性脉冲的结束处算起 100ms 以内准备接收来自主机的命令(帧头)；主机节点也必须被唤醒，100ms 之内主机节点发送帧头开始通信。主机节点的同步间隔段也可以充当唤醒信号，由于从机节点需要作初始化处理，因此主机节点所发的这个帧有可能不会被正常接收。

如果节点发送出唤醒信号后，在 150ms～250ms 之内没有接收到总线上的任何命令(帧头)，则可以重新发送一次唤醒信号。唤醒信号最多可以发送 3 次，3 次之后，必须等待至少 1.5s 之后才可以再次发送唤醒信号，如图 3.18 所示。

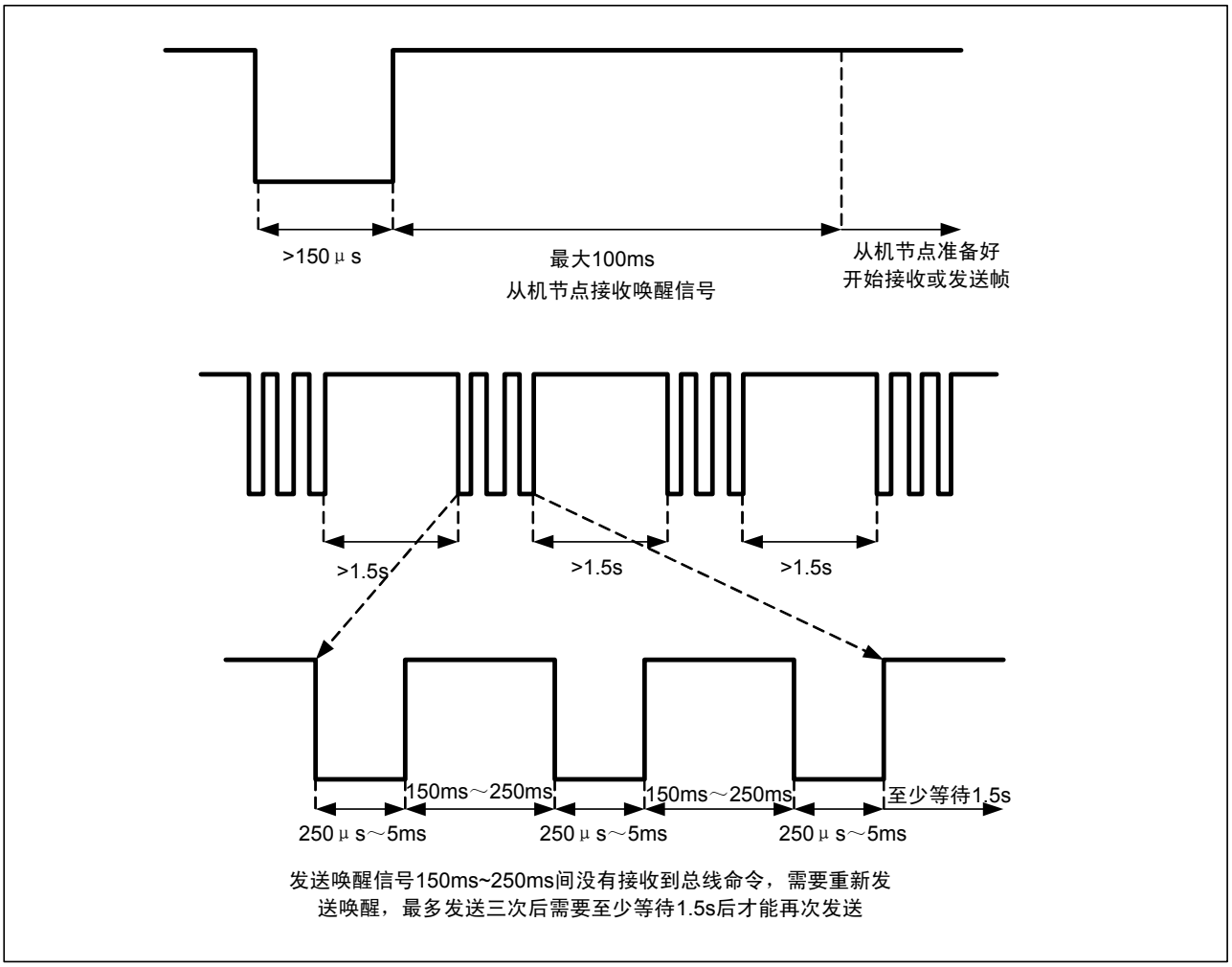

图 3.18 唤醒

3.5.2 休眠

总线可以在两种情况下进入休眠：

(1) 利用诊断帧中的主机请求帧 0x3C 作休眠命令，要求数据段的第一个字节为 0x00，其余字节为 0xFF。休眠命令由主机节点发出，总线上的从机节点只判断数据段的第一个字节，其余字节忽略。从机节点在接收到休眠命令后，不一定要进入低功耗模式，根据应用层需要设置，如图 3.19 所示。

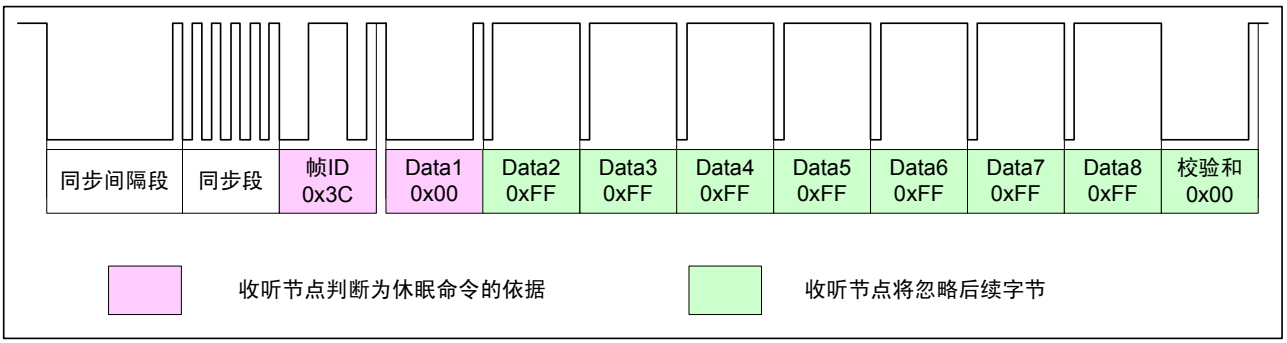

图 3.19 休眠命令

(2) 当总线静默(没有显性和隐性电平之间的切换)4s～10s 时，节点自动进入休眠状态。

3.6 状态管理

状态管理是为了检测运行中的错误。错误一旦被发现，根据设计需要采取不同的措施进行排除，一种方法是简单替换掉错误节点，另一种方法是让发生问题的节点进入到自我保护/安全模式(Limp Home Mode)。

3.6.1 网络报告

协议强制规定，每个从机节点都要在它发布的某个无条件帧中包含一个长度为一位的标量信号 response_error，向主机节点报告自身状态。主机节点负责接收这个信号并且执行分析，如表 3.6 所示。事件触发帧由于允许总线冲突，需特殊处理。

表 3.6 主机节点对 response_error 的解释

response_error	主机节点对信号的解释
FALSE	从机节点操作正确。
TRUE	从机节点有偶发错误。
从机节点无应答	从机节点、总线或主机存在严重错误。

LIN 协议并没有标准化错误类型，用户可根据需要自行制定。表 3.7 列出了可能出现的一些错误类型供参考。

表 3.7 错误类型举例

错误类型	解释
位错误	通常，在一个时刻，LIN 总线上只有一个节点在向外发送信息，发送的同时回读总线上的数据，当发送节点发送电平与回读电平不一致时，视为位错误。(事件触发帧的总线冲突除外，参照 3.2.2 节)。
同步段错误	根据接收的同步段重新计算的位速率超出了规定的容限(参照 4.6 节的表 4.3)，认为是同步段错误。
PID 错误	接收节点对帧 ID(PID 的前六位)按照校验规则重新计算校验位(P0 和 P1)，若与接收到的校验位不符，则接收节点认为是 PID 传输错误。
无应答错误	发送完帧头后，如果总线上没有节点应答，视为无应答错误(事件触发帧除外)。
应答不完整错误	收听节点接收的数据段不完整或没有接收到校验和段。
校验和错误	收听节点接收到的校验和与重新计算的校验和(不取反)加起来不等于 0xFF。
帧错误	字节域的停止位上出现了显性电平。
物理总线错误	总线短路或直接连到电源上导致总线无法通信，该错误由主机节点负责检测。

3.6.2 节点内部报告

节点自身需要设定两个状态位：Error_in_response 和 Successful_transfer。当发送或接收应答的时候发现错误，将置位 Error_in_response；成功传输则置位 Successful_transfer。节点需要将这两个状态位报告给应用层。

4. 帧收发的硬件实现

本章着重介绍与 LIN 帧收发相关的硬件的组成、特点以及应用设计时的注意事项。本章内容对应着 LIN 规范的以下部分：

- LIN Protocol Specification(部分内容)
- LIN Physical Layer Specification

4.1 组成

收发 LIN 帧需要的硬件包括协议控制器(Protocol Controller)、总线收发器(Bus Transceiver)和 LIN 总线三部分，如图 4.1 所示。

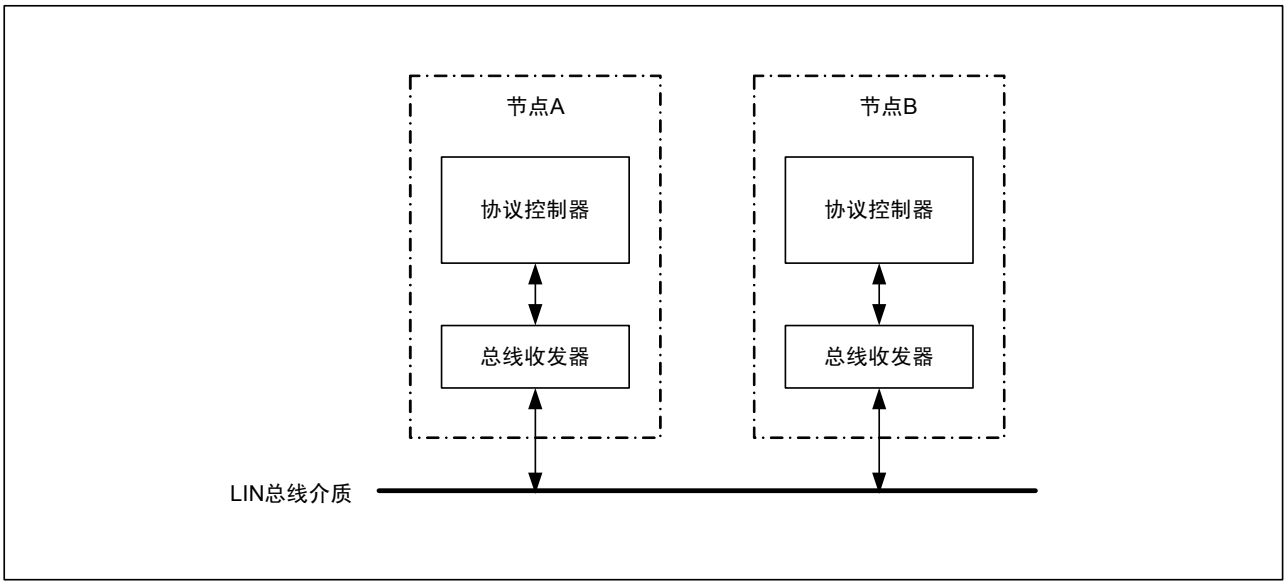

图 4.1 帧收发相关硬件

LIN 规范并未限定传输介质的类型和连接器的规格。目前 LIN 网络主要使用铜线作为传输介质，针对铜线的总线收发器也是市场主流。鉴于这种情况，下文如无特别说明，均是针对铜线介质展开。

4.2 LIN 的硬件特点

- 单线通信
- 从机节点无需高精度时钟源
- EMI 低而且可控
- 最高通信速率 20kbps

4.3 协议控制器

协议控制器的主体是一个基于 UART/SCI 的通信控制器，工作方式是半双工。协议控制器既可以使用专用模块实现，也可以用“UART/SCI+定时器”实现。发送时，协议控制器把二进制并行数据转变成高-低电平信号，并按照规定的串行格式(8 数据位，1 停止位，无校验位)送往总线收发器；接收时，协议控制器把来自总线收发器的高-低电平信号按照同样的串行格式储存下来，然后再将储存结果转换成二进制并行数据。

协议控制器要能产生和识别帧的同步间隔段。如第 3 章所述，同步间隔段包含一个低电平脉冲，长度至少为 13 位。发出和识别同步间隔段虽然增加了硬件设计的复杂度，但是从接收方的角度看，这样做能把同步间隔段与普通的数据字节区别开，确保了同步信息的特殊性。

协议控制器要能执行本地唤醒(Local Wakeup)。需要唤醒总线时，协议控制器通过总线收发器向 LIN 总线送出唤醒信号(参照 3.5.1 节)。

协议控制器要能识别总线唤醒(Bus Wakeup)。当收到来自 LIN 总线的唤醒信号时，协议控制器能够正确动作，进入规定的通信状态(注 1)。

注：1. 例如，主机节点延迟 100ms，然后查询唤醒来源。

4.3.1 实现方案

依据硬件资源不同可以分为 3 类：UART/SCI+定时器+外部中断、硬件 LIN(Hardware LIN)和 LIN 模块(LIN Module)，分别面向对成本和性能有不同侧重的应用。

4.4 总线收发器

总线收发器的主体是一个双向工作的电平转换器，完成协议控制器的高-低电平与 LIN 总线的隐性-显性电平(注 1)之间的转换。显性-隐性电平的定义如表 4.2 所示。

表 4.1 LIN 总线电平定义

	最小值	典型值	最大值	说明
显性	未规定	未规定	0.4Vsup	Vsup 指总线收发器的电源电压
隐性	0.6Vsup	未规定	未规定	

LIN 规范规定：LIN 总线的电平参考点是总线收发器的电源参考点。为了克服电源波动和参考点漂移的影响，LIN 规范要求总线收发器要能承受±11.5%的电源波动和参考点电平波动，并且能承受电源和参考点之间 8%的电位差波动。收发双方的电平鉴别门限也设置了较大的冗余度。参照参考资料[9]的表 6.6。

总线收发器还包括一些附加的功能，例如总线阻抗匹配、压摆率(Slew-rate)控制等。

此外，LIN 规范要求总线收发器具备这样一种特性：本地节点掉电或工作异常时，不能影响总线上其他节点工作。

注：1. “显性-隐性”此处有两个含义，一是突出 LIN 总线“线-与”的本质，二是与协议控制器的“高-低电平”相区别。

4.4.1 实现方案

在一些要求不高的场合，可以采用简单的收发器电路，如图 4.2 所示，参照参考资料[4]。

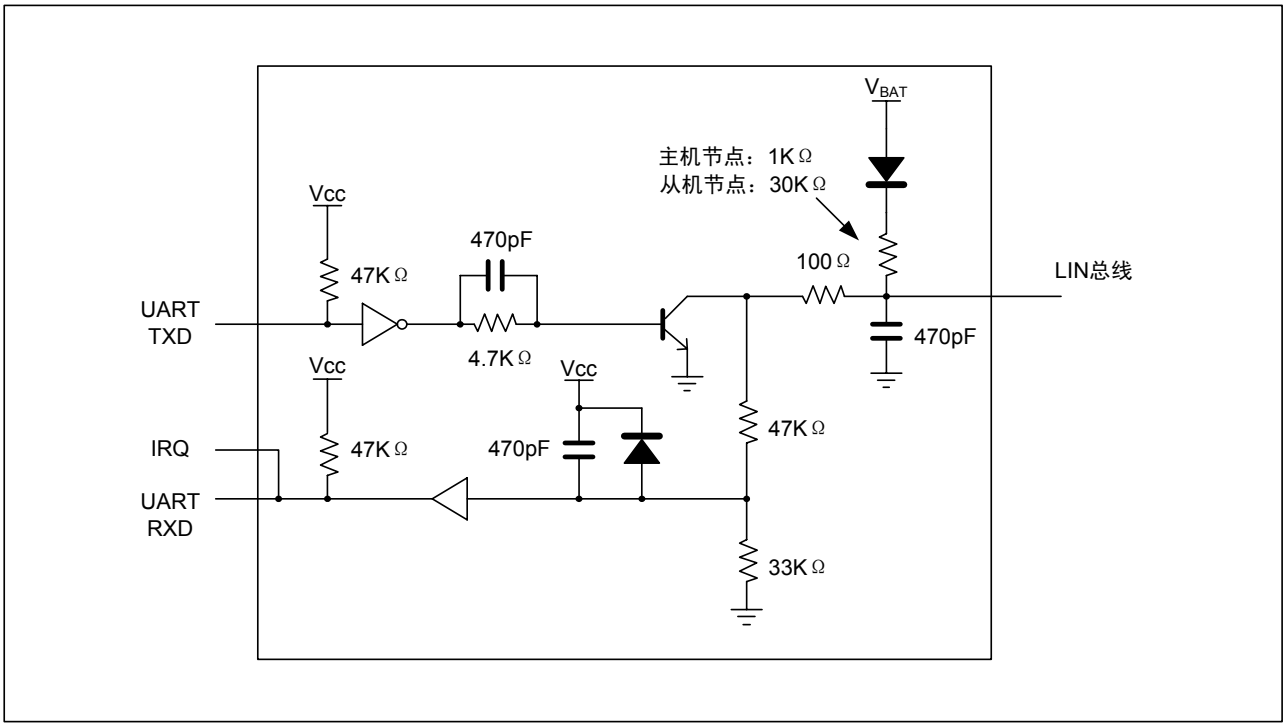

图 4.2 简易的 LIN 收发器电路

不少半导体厂商提供集成化的总线收发器，这些产品功能完善，环境适应能力强，设计产品时建议优先考虑。

4.5 LIN 总线

LIN 总线是衔接所有 LIN 节点的通信介质。

LIN 总线的特征阻抗——尤其是容抗——会影响信号的波形，在设计产品时应予以重视，参照 4.8.2 节。

为汽车电子产品增加 LIN 功能所花费的成本与获得的灵活性相比，往往后者更为显著。汽车上大多数传感器、执行器除至少要接 1 根电源线和 1 根地线外，此外还有一些模拟/数字信号线，这些接口往往存在兼容性的问题。如果采用 LIN 规范，仅用 3 根线(电源、地和 LIN)就可以实现标准化的数字接口。传感器、执行器通过总线连接，汽车结构设计可以更加灵活，线束的数量(重量)不但不会增加，还可能减少。

4.6 时钟源

LIN 网络的主机节点必须设置较高精度的时钟，而从机节点则不必。换句话说，主机节点是 LIN 网络的时间基准，这保证了位速率的准确性。LIN 规范规定一个 LIN 网络里只有一个主机节点，这保证了位速率的唯一性。LIN 规范规定所有通信都由主机节点发起，并在帧头中加入同步段，这就给从机节点提供了主机节点位速率的信息。只要所有从机节点都能在 LIN 通信时与主机节点采用同样的位速率，LIN 网络就能正常工作。这种做法虽然降低了传输效率，但是一方面减少了高精度时钟数量，降低了成本；另一方面不需要仲裁，降低了软硬件设计复杂度。

主机节点、从机节点位速率要满足表 4.3 的要求。

表 4.2 主机节点和从机节点位速率的允许误差

参数名称	允许误差	含义
Ftol_res_master	±0.5%	主机节点位速率相对于额定位速率的最大偏差。
Ftol_res_slave	±1.5%	对于自身位速率较准确的从机节点(例如使用了高精度时钟)，可以不必利用同步段修正自身的位速率。这个指标表示此类从机节点的位速率与额定位速率的最大偏差。
Ftol_unsync	±14%	对于自身位速率不准确的从机节点，需要利用同步段修正自身的位速率。这个指标表示此类节点在同步之前的位速率与额定位速率的最大偏差。
Ftol_sync	±2%	同步后，主机节点和从机节点位速率的最大偏差。
Ftol_sl_to_sl	±2%	收发应答段期间，互相通信的两个从机节点位速率的最大偏差。

对表 4.3 的参数说明如下：

- 主机节点、从机节点的位速率必须在使用环境要求的温度范围和电压范围内，优于规定的精度(主机节点是±0.5%，从机节点是±14%)。
- 同步之前，主机和从机节点的位速率与额定位速率的误差应符合 Ftol_res_master 和 Ftol_res_slave/Ftol_unsync 的要求。主机节点按主机位速率发出同步间隔段，从机节点应可按从机位速率将其解释为长度大于 9～11 位的显性电平(这个宽度必须是帧的其他部分不可能出现的)。这里除了要考虑从机节点对时间的测量误差(取决于时钟精度)，还要考虑从机节点可用位速率与额定位速率的误差，以及 LIN 总线电抗特性造成的传输延迟(固有误差，取决于硬件设计)。
- 同步之后，从 PID 段到校验和段，通信双方的位速率相对误差不大于±2%(即 Ftol_sync 和 Ftol_sl_to_sl)。如果是主机节点与从机节点通信，设 Ftol_res_master 为±0.5%，那么不论从机节点是否利用同步段修正位速率，其位速率相对于额定位速率的误差不能大于 Ftol_sync - Ftol_res_master，即±1.5%。如果是从机节点之间通信，对从机节点各自的位速率误差的要求将高于±1.5%。

4.7 EMI 及其控制

EMI 这里指电磁干扰。对于 LIN 而言，EMI 主要由位速率和压摆率共同决定。位速率决定单位时间内电平变化次数，压摆率决定电平跳变的快慢。单位时间内跳变次数越多，每次跳变持续时间越短，跳变过程包含的谐波成分就越丰富，EMI 也越大；相反，跳变持续时间越长，单位时间跳变次数越少，其谐波成分越少，EMI 也较低。

LIN 可以控制 EMI。这是因为协议控制器可以控制 LIN 总线位速率，总线收发器可以控制压摆率。另外，LIN 协会把 LIN 的最高位速率限制在 20kbps。值得一提的是，这个速度远非 LIN 物理层的极限，而是在数据速率与 EMI 之间权衡的结果。

4.8 设计电路时的注意事项

4.8.1 工作环境对时钟的影响

片上振荡器容易受到环境温度和电源电压的影响，石英晶体容易受到冲击振动的破坏。在选择时钟源时，一定要考虑使用环境的温度范围、电源电压范围和冲击振动情况。在电路板布局时，要让时钟器件尽量避开热源和易受外力冲击的部位。总之，要确保在最恶劣的情况下，也能保证时钟的精度和稳定性。

LIN 的硬件部分应该在-40℃~+125℃的温度范围内保证满足 LIN 规范规定的指标。在电源适应性方面，要能承受±11.5%(注 1)的电压波动。对于汽车电子产品，还要承受汽车电源系统常见的总线短路、负载突降(Load Dump)、电源反接、蓄电池串联(Jump-start)等现象。

注：1. LIN 规范 2.1 版将这个指标提高到±11.5%，LIN 规范 1.x 和 2.0 版中，这个指标是±10%。

4.8.2 端接阻抗和总线负载

为了实现“线-与”特性，LIN 规范规定了主机节点和从机节点的端接电阻，如表 4.4 所示。端接电阻一端连接 LIN 总线，另一端经串联二极管连接收发器电源，如图 4.3 所示。图中的串联二极管是必须的，当 ECU 的“电池断路”的时候，它可以防止 LIN 总线向 ECU 供电。

表 4.3 端接电阻

	最小值	典型值	最大值	单位
主机节点	900	1000	1100	Ω
从机节点	20	30	60	KΩ

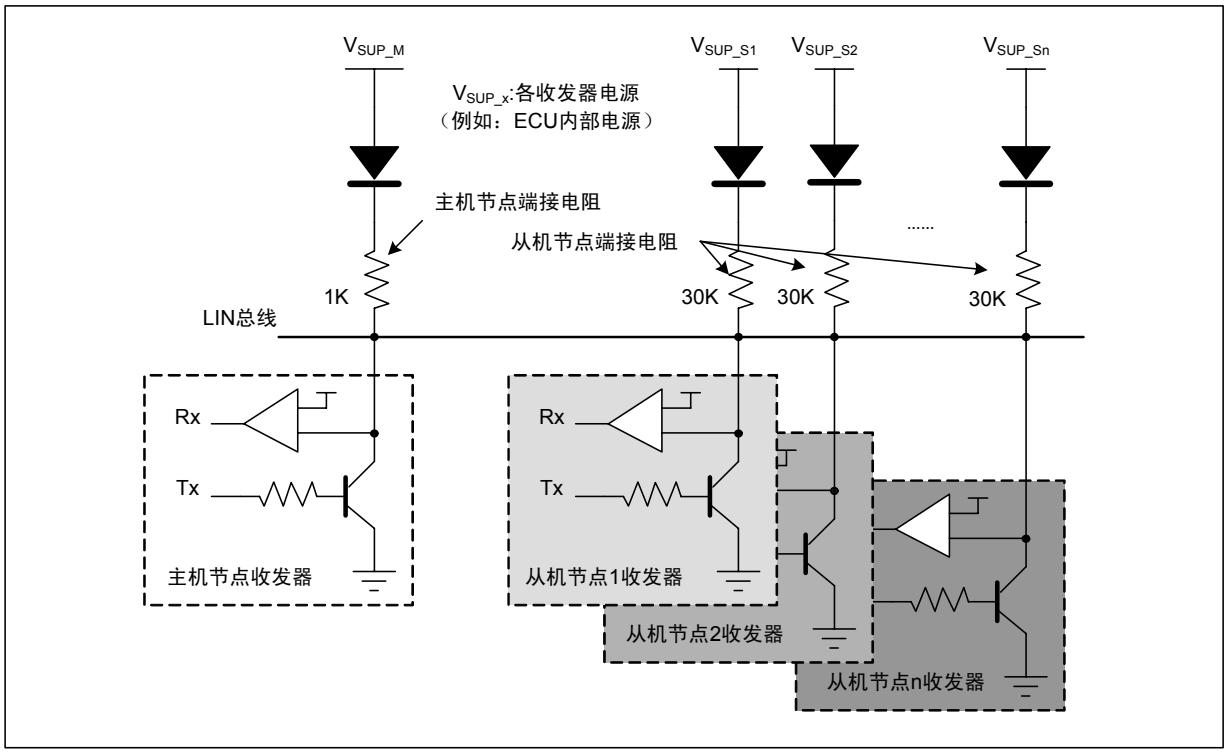

图 4.3 端接电阻

LIN 子网各节点并联在一起，构成如图 4.4 所示的等效电路。其中，总线负载电阻等于各节点端接电阻的并联等效电阻(总线的电阻通常很小，可以忽略)，总线负载电容等于各节点输入电容和总线分布电容的并联等效电容。总线电阻决定了总线收发器驱动级的功率和通信期间的功耗；总线电容可以很好地吸收周围环境的噪声干扰。总线电阻和总线电容构成的 RC 滤波器还有助于控制压摆率。

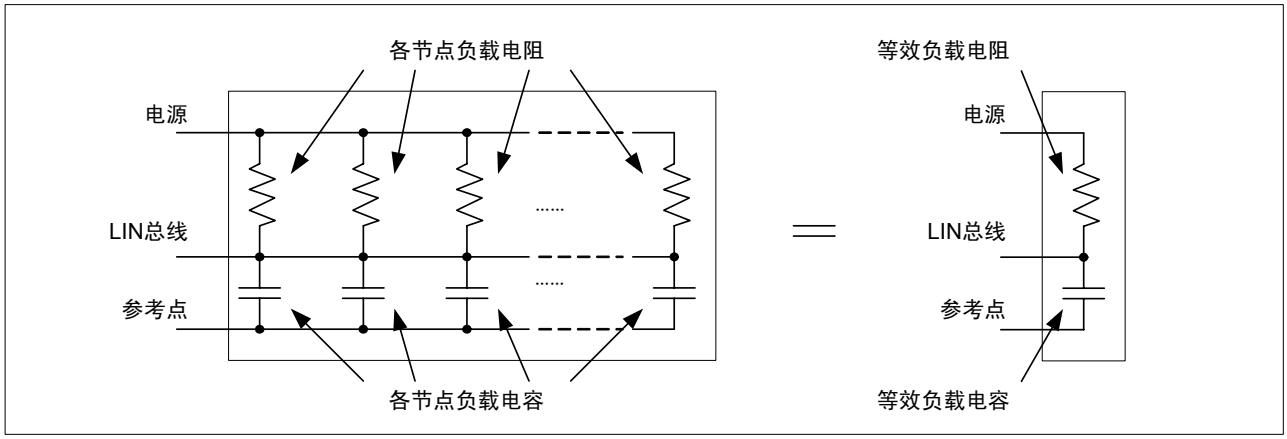

图 4.4 总线等效电路

为确保最恶劣情况下正常通信的需要，LIN 规范除了限制节点的端接电阻、电容和时间常数，还规定 LIN 总线长度不超过 40 米，一个 LIN 网络的最大节点数目不超过 16。

应注意，由于端接电阻连接着电源和 LIN 总线，当出现 LIN 总线对地短路时，如果不采取保护措施，会有较大的电流流过端接电阻，产生显著的功耗。

4.8.3 ESD 防护

ESD(Electrostatic Discharge)指静电危害，表现为短暂而幅度迅速衰减的高压、大电流放电，由于其瞬间电压可达数千伏，对于工作在干燥的环境或者接地不良的电子产品威胁最严重。

总线收发器最容易受到 ESD 的冲击。LIN 规范要求总线收发器的电源和地应直接连接到 ECU 的接口处，这就使电源线成为 ESD 侵入的窗口。同样，LIN 总线也可能窜入 ESD。如果处理不当，容易因为放电过程中的高压、大电流烧毁内部器件。

设计时可以采取以下措施：

- 在电源线和地线之间串联电阻和电容；
- 地线加粗，并与其它大面积接地导体就近，链接低阻抗；
- 在 LIN 信号线与地线之间并接 ESD 保护器件，例如瞬态电压抑制器件(Transient Voltage Suppressor，TVS)、RC 滤波器(注 1)等。

有关此问题的详细分析参照参考资料[5]、[6]。

注：1. RC 滤波器的截止频率应远高于 LIN 的位速率，否则会影响 LIN 通信波形。

4.8.4 兼容性

协议控制器可以用广泛使用的 UART/SCI 来实现。

总线收发器的主要指标符合 ISO 9141 的要求，参照参考资料[10]。

4.9 参考资料

[1] R8C/Tiny Series R8C/11 Group LIN(Local Interconnect Network) Application Note: Slave Volume, Renesas Technology, 2006
[2] R8C/Tiny Series R8C/22 Group R8C23 Group Hardware Manual, Renesas Technology, 2006
[3] R32C/100 Series R32C/121 Group Hardware Manual (preliminary), Renesas Technology, 2006
[4] H8/300H Tiny Series LIN(Local Interconnect Network) Application Note: Master, Renesas Technology, 2003
[5] TJA1020 LIN transceiver application note, NXP Semiconductors, 2005
[6] TLE7259G LIN transceiver application note, Infineon Technologies, 2005
[7] LIN Protocol Specification Revision 1.3, LIN Consortium, 2002
[8] LIN Physical Specification Revision 2.0, LIN Consortium, 2005
[9] LIN Physical Specification Revision 2.1, LIN Consortium, 2006
[10] ISO9141:1989 Road vehicles - Diagnostic systems - Requirements for interchange of digital information

5. 信号处理、配置、识别和诊断

本章从应用需求入手，介绍了信号处理、配置、识别(Identification)和诊断的概念、功能和用途。本章内容对应于 LIN 规范的以下部分：

- LIN Transport Layer Specification
- LIN Node configuration and Identification Specification
- LIN Diagnostic Specification

从使用的角度来看，LIN 提供四项功能——信号处理、配置、识别和诊断，这四项功能共同构成了 LIN 的应用层。传输层是配置、识别和诊断这三项功能的通信载体，实现应用层消息与帧之间的格式转换和传输。为了规范使用，LIN 为应用层和传输层定义了 API 接口，参照第 6 章。

5.1 传输层

传输层的任务单一，就是充当一个“翻译官”，把来自诊断服务的消息(Message)“翻译”成协议层可以处理的 PDU (Packet Data Unit，分组数据单元)，或者反过来，把协议层收到的 PDU“翻译”成诊断服务需要的消息。消息到 PDU 的转换过程称为拆分(Packing)，PDU 到消息的转换过程称为重组(Unpacking)。PDU 对应着帧结构的数据段，并通过诊断帧发送或接收。

5.1.1 PDU 结构

为满足汽车行业的要求，LIN 传输层 PDU 的格式与 ISO 制定的基于 CAN 网络的诊断标准(参照参考资料[9])非常相似(是 ISO 标准的子集)。这种兼容性大大减少了在 CAN 和 LIN 之间转换数据格式的工作量，降低了对节点计算能力的要求。

从发送格式上，PDU 单元可分为单帧(Single Frame，SF)、首帧(First Frame，FF)和续帧(Consecutive Frames，CF)三种。从发送源上，主机发送请求 PDU，从机发送应答 PDU。

如图 5.1 所示，为 PDU 格式，包括节点地址(NAD),协议控制信息(PCI),LEN,服务 ID(SID),应答服务 ID(RSID),消息字节段(D1~D6)。首字节 NAD 首先发送，末字节 D4,D5,D6 最后发送。

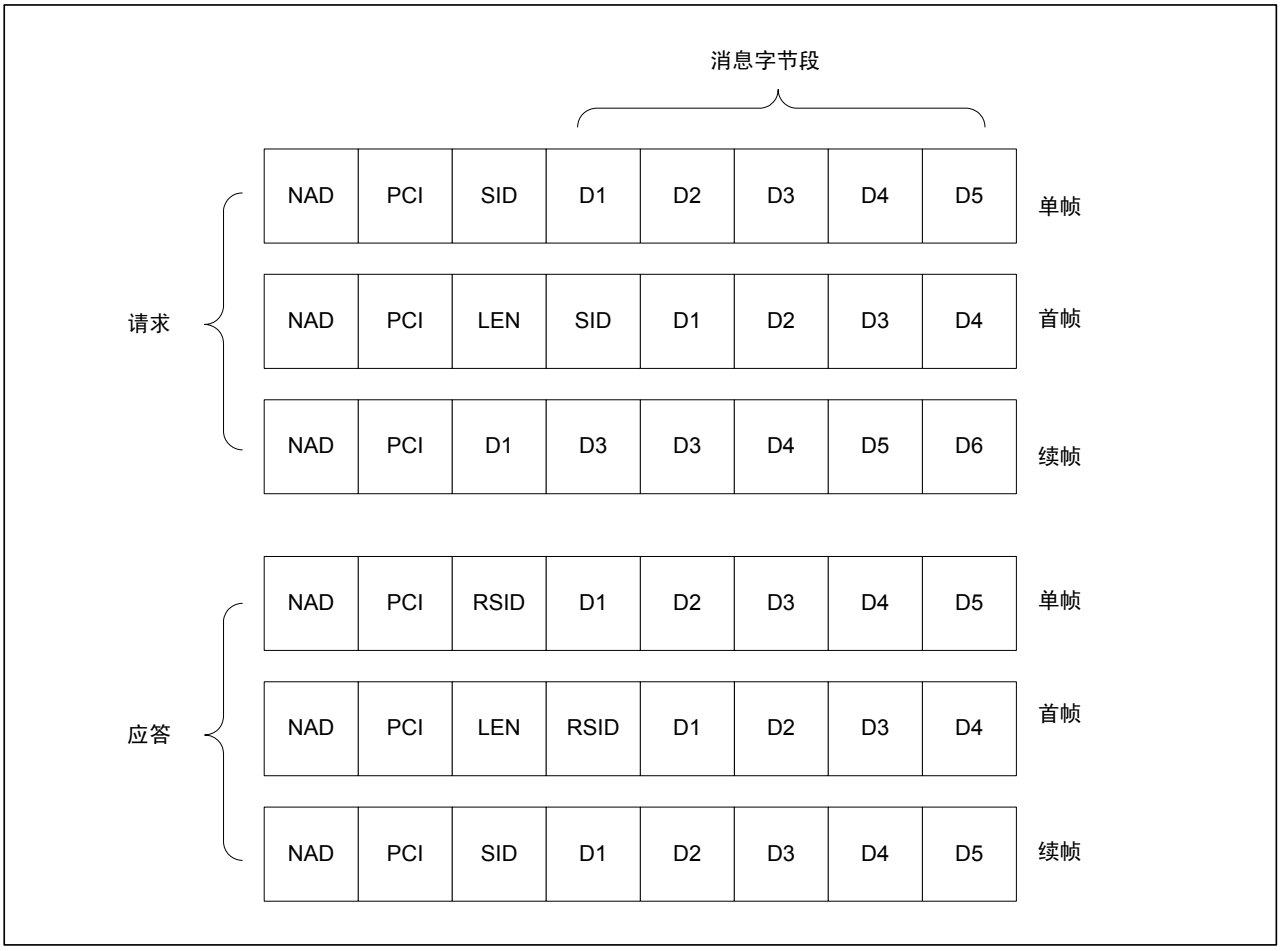

图 5.1 LIN 传输层支持的 PDU 格式

5.1.1.1 NAD

PDU 单元的第一个字节是 NAD（node address），用于区分不同从机节点的地址。

如表 5.1 所示，列出了 NAD 的取值范围

表 5.1 NAD 取值范围

0x00	用于休眠命令，参见 3.5.2。
0x01～0x7D	从机节点地址，即 NAD
0x7E	功能节点地址（功能 NAD）
0x7F	广播节点地址（广播 NAD）
0x80～0xFF	用户自定义

5.1.1.2 PCI

PDU 单元的第二个字节是 PCI（Protocol Control Information）信息，包含了 PDU 单元类型和消息字节长度的信息。如表 5.2：

表 5.2：PCI 数据结构

类型	PCI 类型				附加信息			
	B7	B6	B5	B4	B3	B2	B1	B0
SF（单帧）	0	0	0	0	Length			
FF（首帧）	0	0	0	1	Length 高 4 位			
CF（续帧）	0	0	1	0	帧计数器			

单帧中，附加信息 Length 表示消息字节数加 1。首帧中，附加信息只表示 Length 的高 4 位，低 8 位在 LEN 中表示。因此在消息长度为 12 位数据，最大长度为 4095（0xFFF）。

续帧中的附加信息表示首帧后，跟随的续帧的编号，第一个续帧编号为 1，之后累加 1。如果续帧数多于 15 个，那么帧计数器在第 16 个续帧时从 0 重新计数。

5.1.1.3 SID 与 RSID

SID（Service Identifier）表示了从机节点应完成的服务请求。节点配置服务的 SID 区间为 0xB0~0xB7,诊断服务的 SID 区间为 0x00~0xAF,0xB8~0xFE。

RSID（Response Service Identifier）表示从机节点应答的内容，它的值是 SID+0x40。

5.1.1.4 消息字节段

消息字节段的内容取决于服务的种类。在单帧中，消息字段最多 6 个字节。在首帧和续帧中，所有 PDU 的消息字段，经过“重组”组成一个完成的消息。

5.1.2 传输层通信

应用层发出的消息如果长度不超过单帧的容量，传输层会按单帧的格式交给协议层发送。传输层收到的单帧也会直接作为消息送往应用层；如果消息长度超过单帧的容量，传输层先要把消息拆分成首帧和续帧并排好次序，然后再交给协议层依次发送。反过来，协议层收到的首帧和续帧，传输层先要按照接收次序将其重组为消息，最后交给应用层处理。

LIN 传输层只能按顺序接收续帧。

LIN 传输层具备出错重传功能。

传输层由传输层 API 完成，参照 6.4 节。表 5.3 列出了传输层与 API 的对应关系，API 的内容参照第 6 章。

表 5.3 传输层与 API 的关联

操作	传输层 API
收/发一个 PDU	ld_put_raw ld_get_raw
收/发一个消息	ld_send_message ld_receive_message

5.2 LIN 应用层

5.2.1 概述

LIN 应用层提供信号处理、配置、识别和诊断四项功能。配置、识别和诊断功能又包含若干项目，称为服务(Service)。为了区别，每项服务都有固定、唯一的服务代号(Service ID，SID)。图 5.2 描述了 LIN 应用层及其关联。

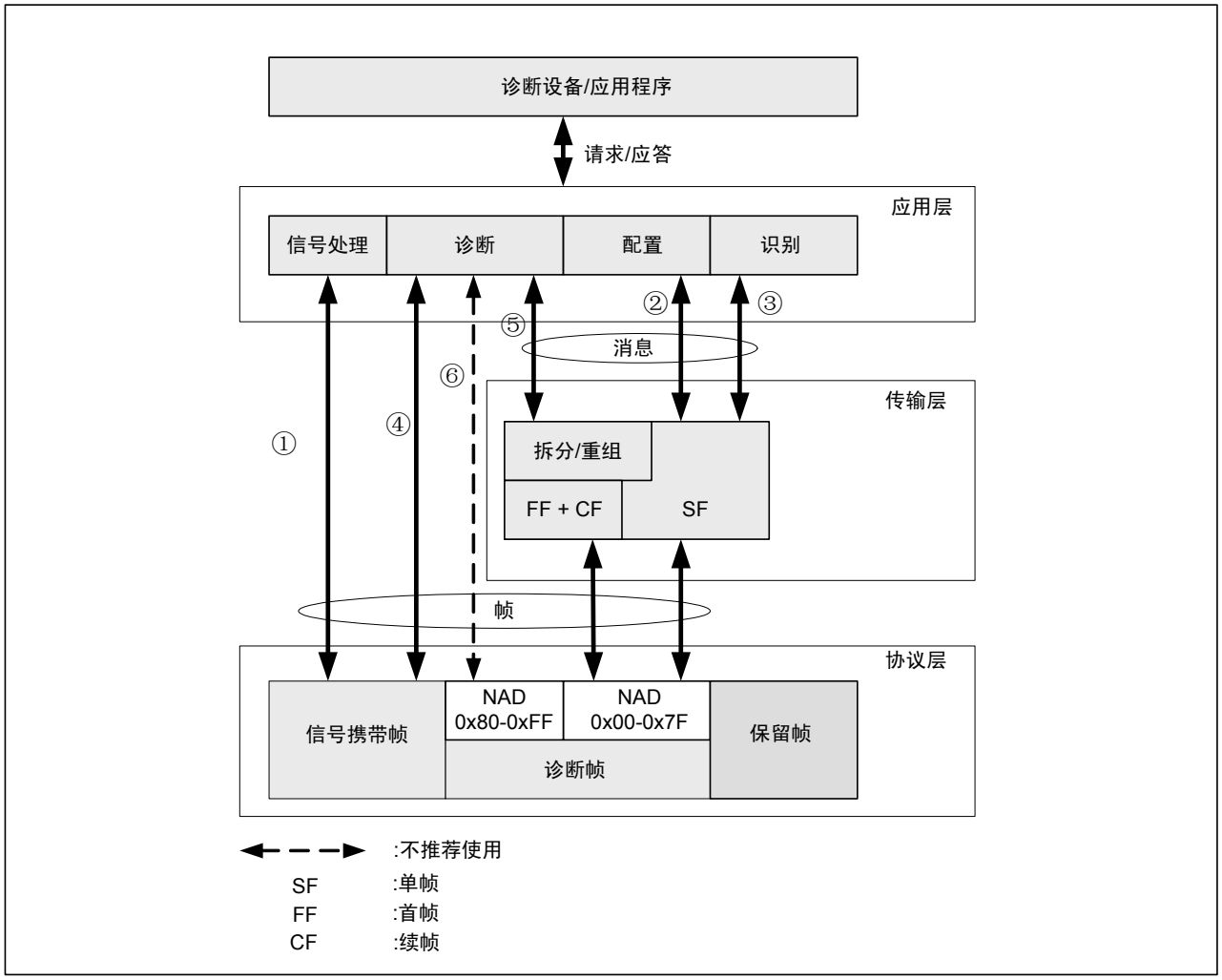

图 5.2 LIN 应用层及其关联

① 信号通过信号携带帧通信
② 配置服务通过传输层，以单帧的形式通信
③ 识别服务通过传输层，以单帧的形式通信
④ 基于信号的诊断服务
⑤ 诊断传输层，通过传输层通信，需要使用复帧的形式通信
⑥ 用户自定义的诊断

为便于理解本图，后文对每项功能都分别进行了详细描述并提出了工作模型的概念。

LIN 应用层的配置、识别和诊断都是针对逻辑节点(Logical Node)的。逻辑节点是能够对来自主机节点和/或诊断设备的服务请求作出响应的功能实体。为了区别不同的逻辑节点，LIN 定义了 NAD(Node Address for Diagnose，诊断地址)。第 1 章介绍了物理节点(Physical Node)、从机任务和接口(Interface)的概念。对于一个物理节点来说，从机任务和接口对应着实现帧收发的软件和硬件实体，而逻辑节点则代表了配置、识别和诊断方面的能力。物理节点、从机任务以及接口是一一对应的，但是物理节点可以包括 1 个或者多个逻辑节点。

为了规范地使用应用层的功能，LIN 规范定义了一套 API。下文会提到各项功能与 API 的关联，API 的内容参照第 6 章。

5.2.2 信号处理功能

信号处理功能是指应用层可以不经过传输层，直接从协议层获取或修改网络中的信号。这些信号由NCF(Node Capability File，节点性能文件)定义，既可以是工作参数(例如温度、压力的测量值、继电器的开合状态等)，也可以是状态标志(例如某信号携带帧的收发状态)。

信号处理功能的工作模型如图 5.3 所示。信号携带帧在 LIN 网络的节点之间传递，每个节点既可以是信号的发布者，也可以是信号的收听者。

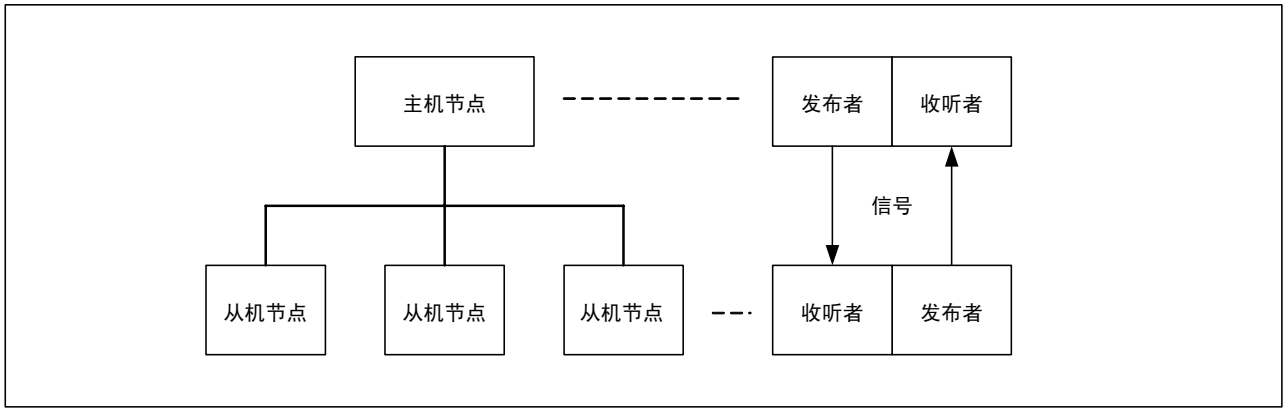

图 5.3 信号处理模型

信号处理功能由核心 API 完成，参照 6.3 节。表 5.4 列出了信号处理功能与 API 的对应关系，API 的内容参照第 6 章。

表 5.4 信号处理功能与 API 的关联

操作	核心 API
信号读写	l_bool_rd/l_bool_wr l_u8_rd/l_u8_wr l_u16_rd/l_u16_wr l_bytes_rd/l_bytes_wr
读标志，清除标志	l_flg_tst l_flg_clr

5.2.3 配置功能

LIN 规范规定，每个逻辑节点都应该有 NAD。在网络运行期间，任意两个逻辑节点的 NAD 都必须不同，否则就会产生冲突。此外，每个逻辑节点都要能处理带有某些 PID 的帧。由此可见，NAD 和 PID 分别与逻辑节点建立了一种映射关系，LIN 规范把 NAD 和 PID 的这样一种组合称为逻辑节点的配置项(Configuration)。一个逻辑节点可以有一个以上的配置项，但在网络运行期间，每个逻辑节点只能有一个配置项有效。

配置功能是指 LIN 的主机节点能自动地给所有逻辑节点选择配置项，消除 NAD 和 PID 分配中存在的冲突，使网络正常工作。配置功能是确保各节点协调运作的内部功能，包含分配 NAD、分配 PID 等服务。配置功能通过传输层完成配置服务。

为了适应汽车行业的需要，LIN 规范定义配置功能的服务时，参照了 ISO 制定的 UDS(Unified Diagnostic Services，车辆统一诊断服务)标准(参照参考资料[7])和 OBD(On-board Diagnostic，车载自动诊断)标准(参照参考资料[9])。配置功能各项服务及其 SID 都是 ISO 标准的子集。

配置功能的工作模型与计算机局域网的“客户机-服务器”模型很相似，如图 5.4 所示。主机节点可以被视为客户机，逻辑节点被视为服务器。客户机首先向服务器发出服务请求，服务器依照请求执行操作，然后向客户机返回应答。

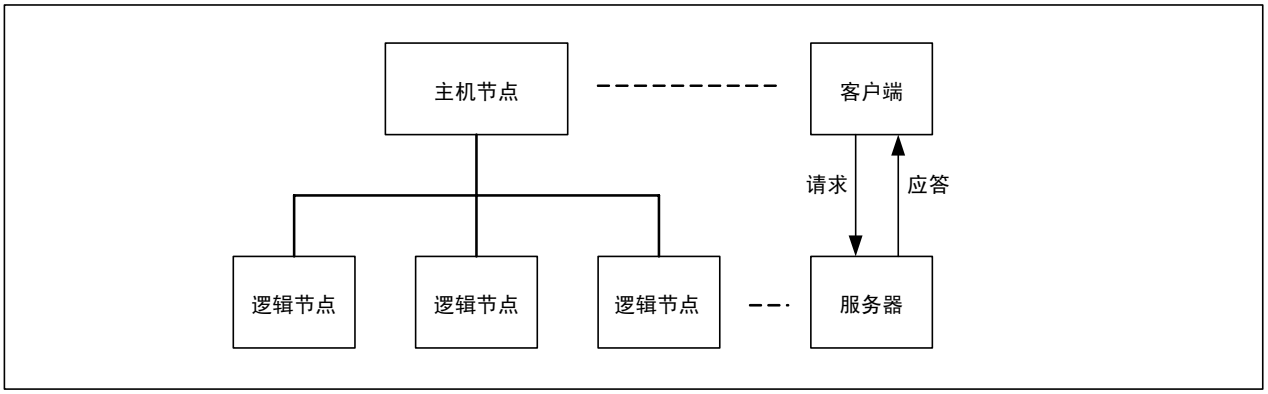

图 5.4 配置/识别模型

5.2.3.1 节点存储模型

如图 5.5，节点存储模型

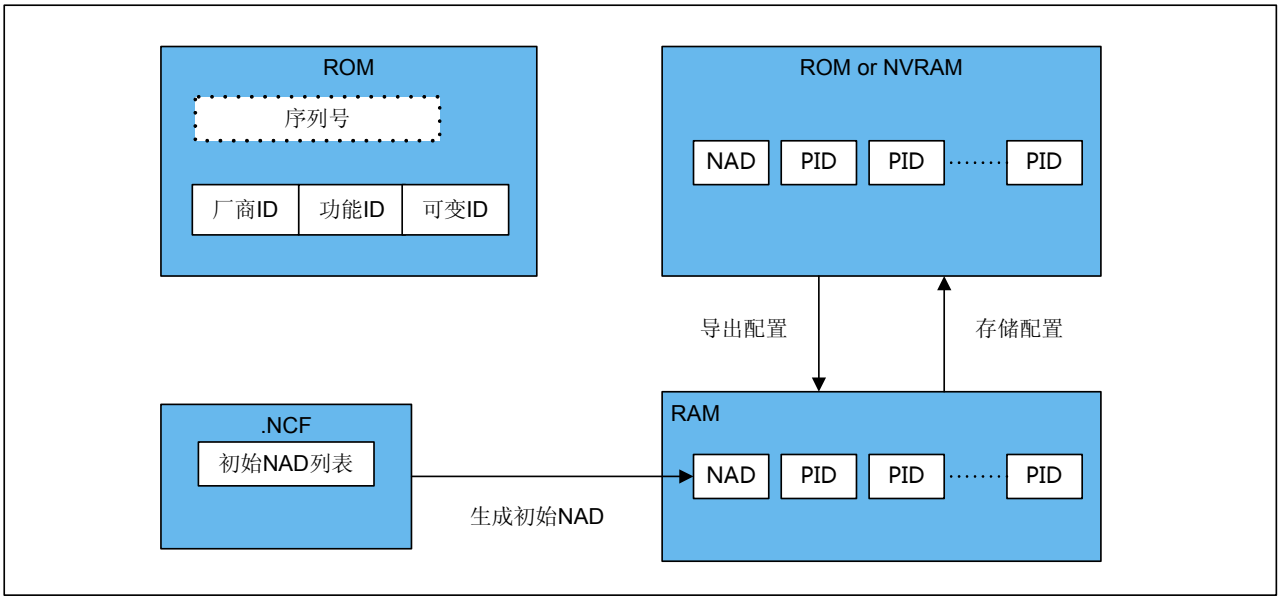

图 5.5 节点存储模型

如同商品包装上的条形码，每个物理节点都有一个固定的编码，叫做 LIN 产品代号(LIN Product Identification)。产品代号是出厂时赋予的，除非修改产品，否则其内容不变。产品代号保存在不需要电源就能维持记录的地方，例如 ROM 或者非易失性存储器(Non-volatile Random Accessible Memory，NVRAM)。在进行配置服务时，从主机接收的产品代号必须和从机节点保存的产品代号一致，才能正常进行配置服务。

表 5.5 LIN 产品代号存储格式

D1	D2	D3	D4	D5
厂商 ID LSB	厂商 ID MSB	功能 ID LSB	功能 ID MSB	可变 ID

另外，从机节点还可以有一个序列号，用于识别特殊的节点。序列号大小为 4 字节。

从机节点可以将配置信息保存起来，重启后调用保存的配置信息，而无需主机节点再次分配。

针对配置项的存储类型，LIN 规范定义了三种从机节点配置模型：

第一种，无配置节点，这种从机节点在重启后，自身没有配置项，每次重启都需要主机进行配置。

第二种，预配置节点，这种从机节点在重启后，调用预先设置的配置项。但是在主机重新对其进行配置后，不能存储新配置项。

第三种，全功能配置节点，这种从机节点可以保存主机对其的配置，并在重启后调用此配置。

表 5.6 从机节点配置模型类型

从机节点配置	重启后配置	存储配置
无配置节点	无	无
预配置节点	有	无
全功能节点	有	有

5.2.3.2 从机节点 NAD 配置

有三种方法生成配置 NAD，如果初始 NAD 等于配置 NAD，那么不需要进行其他配置操作。如果配置 NAD 需要从从机节点存储的保留配置中提取，需要调用 ld_set_configuration 进行配置，如果 NAD 需要变更，则需要主机发送配置 NAD 请求。

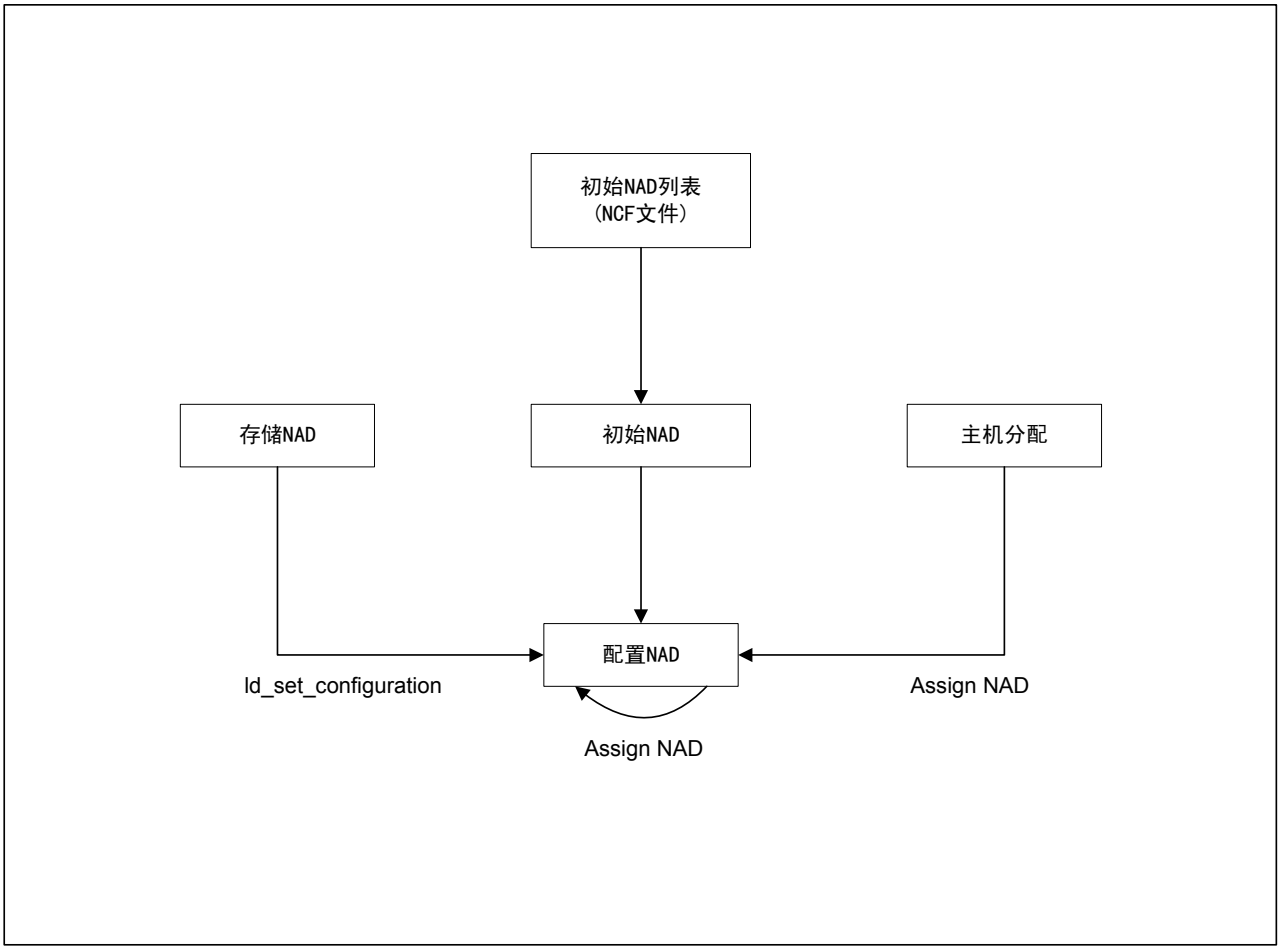

图 5.6 配置 NAD

主机节点给从机节点分配 NAD 是通过 Assign NAD 服务完成的。首先主机节点向从机节点发送配置 NAD 请求，如果从机节点配置成功，从机节点会应答。

分配 NAD 服务的 PDU 结构如表 5.7 所示

表 5.7 分配 NAD 请求与应答

主机请求	NAD	PCI	SID	D1	D2	D3	D4	D5
	初始 NAD	0x06	0xB0	厂商 ID LSB	厂商 ID MSB	功能 ID LSB	功能 ID MSB	新分配 NAD
从机应答	NAD	PCI	RSID	未定义				
	初始 NAD	0x01	0xF0	0xFF	0xFF	0xFF	0xFF	0xFF

注意，应答时，仍然使用初始 NAD。

每个从机节点有一个初始 NAD，初始 NAD 是从一个初始 NAD 列表中选择的。初始 NAD 列表是在编写节点性能文件（NCF）时设置的。LIN 协议没有对生成初始 NAD 的具体方法进行限制。

5.2.3.3 从机节点 PID 配置

从机节点各个帧的 PID，是主机进行分配的。通过分配 PID 列服务，主机一次最多可给从机节点分配 4 个帧的 PID。

分配 PID 列服务的 PDU 结构如表 5.8 所示：

表 5.8 从机节点 PID 配置请求与应答

主机请求	NAD	PCI	SID	D1	D2	D3	D4	D5
	NAD	0x06	0xB7	开始 index	PID (index)	PID (index+1)	PID (index+2)	PID (index+3)
从机应答	NAD	PCI	RSID	未定义				
	NAD	0x06	0xF7	0xFF	0xFF	0xFF	0xFF	0xFF

其中，消息字节段的第一字节是开始帧索引，表示分配第一个帧的排列号。从机节点中各帧的排列顺序是按照节点性能文件（NCF）和 LIN 描述文件（LDF）中定义的顺序定义的。第一帧的索引编号是 0。

后续四个字节是给从机节点分配的 PID。如果分配的 PID 值为 0，表示对应的信号携带帧无效。如果分配的 PID 值为 0xFF，表示保持对应帧的 PID 不变。

5.2.3.4 其它服务

除了对从机节点 NAD 和 PID 的配置，LIN 规范还定义了其他配置服务，如条件变更 NAD，数据导入，保存配置。

配置功能由配置与识别 API 完成，参照 6.5 节。表 5.9 列出了配置服务与 API 的对应关系，API 的内容参照第 6 章。

表 5.9 配置服务及与 API 的关联

服务名称	用途	识别与配置 API
Assign NAD	为逻辑节点分配新的 NAD。	ld_assign_NAD
Conditional change NAD	为 NAD 有冲突的逻辑节点分配新的 NAD。	ld_conditional_change_NAD
Assign frame ID range	为逻辑节点可以处理的帧分配新的 PID。	ld_assign_frame_id_range
Save configuration	请求逻辑节点保存当前的配置项。	ld_save_configuration ld_read_configuration

5.2.4 识别功能

识别功能是指主机节点能够获取逻辑节点的信息，例如产品代号等。借助识别功能，主机节点和逻辑节点还可以实现一些自定义的操作。

识别功能与上面介绍的配置功能使用同样的工作模型，如图 5.4 所示。

识别服务中，主机发送的请求 PDU 单元结构如表 5.10：

表 5.10 识别功能的请求与应答

主机请求	NAD	PCI	SID	D1	D2	D3	D4	D5
	NAD	0x06	0xB2	识别 ID	厂商 ID LSB	厂商 ID MSB	功能 ID LSB	功能 ID MSB
从机应答	NAD	PCI	RSID	D1	D2	D3	D4	D5
识别 ID=0	NAD	0x06	0xF2	厂商 ID LSB	厂商 ID MSB	功能 ID LSB	功能 ID MSB	可变 ID
识别 ID=1	NAD	0x05	0xF2	序列号 0 LSB	序列号 1	序列号 2	序列号 3 MSB	0xFF
识别 ID=32~63	NAD	0x05	0xF2	用户定义	用户定义	用户定义	用户定义	用户定义
失败应答	NAD	0x03	0x7F	请求 SID（=0xB2）	错误代码（=0x12）	0xFF	0xFF	0xFF

其中，从机根据目标 ID 的值来回应相应的信息。目标 ID 指定的相关信息如表 5.11：

表 5.11 目标 ID

目标 ID	指定读出的内容	应答消息长度
0	LIN 产品 ID	6 = 5+RSID
1	序列号	5 = 4+RSID
2-31	保留	-
32-63	用户自定义	用户自定义
64-255	保留	-

识别功能由识别 API 完成，参照 6.5 节。表 5.12 列出了识别服务与 API 的对应关系，API 的内容参照第 6 章。

表 5.12 识别服务及与 API 的关联

服务名称	用途	识别与配置 API
Read by identifier	读取逻辑节点的信息，或者实现自定义的操作。	ld_read_by_id ld_read_by_id_callout

5.2.5 诊断功能

诊断功能是指 LIN 网络之外的诊断设备可以直接连接 LIN 的主机节点，或者通过外部的其它网络(例如 ISO11898 定义的 CAN 网络，参照参考资料[8])连接主机节点，连接后，诊断设备可以按规定的诊断协议(例如 ISO15765 规范，参照参考资料[9])与 LIN 的逻辑节点通讯。与配置功能相比，诊断功能是 LIN 网络作为一个整体对外呈现的可配置、可访问的属性。

为了适应汽车行业的需要，LIN 规范定义诊断服务时，参照了 ISO 制定的 UDS 标准(参照参考资料[7])和 OBD 标准(参照参考资料[9])。LIN 诊断功能是以上两个标准的子集，相同服务的 SID 也相同。

诊断功能的工作模型如图 5.7 所示，它是配置功能工作模型的扩展。主机节点在此扮演了一个“网关”的角色，在诊断设备和 LIN 网络之间传递服务请求和应答。

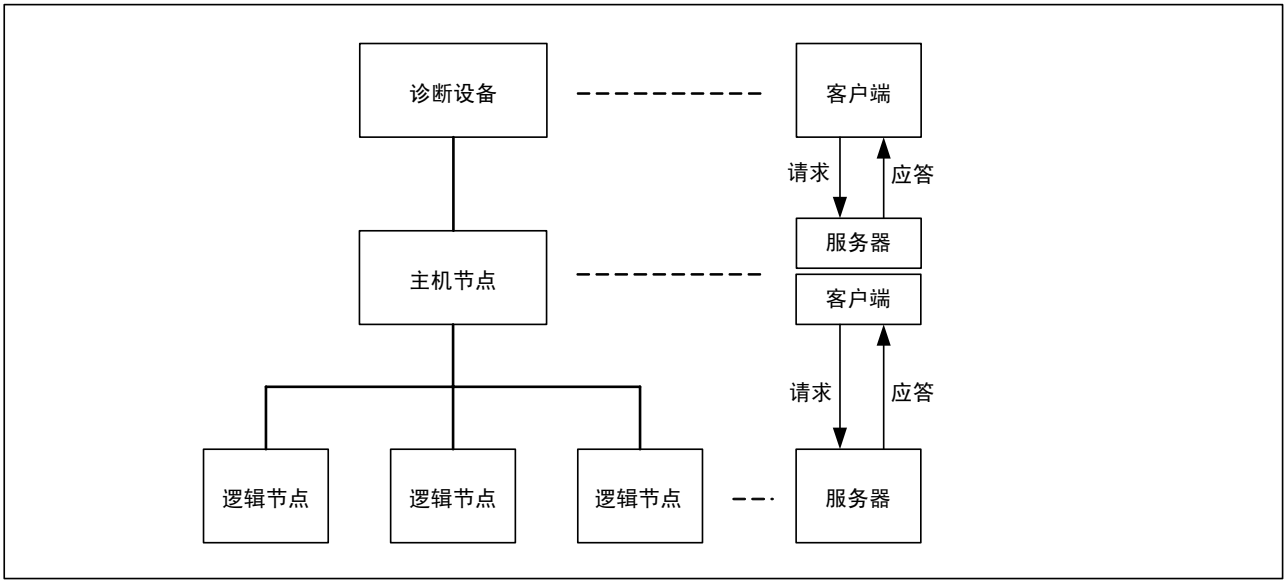

图 5.7 诊断模型

诊断功能的实用意义可以用一个例子来说明：假设采用 LIN 子网的车门连接在底盘 CAN 网络上，诊断车门故障时，只需要把 OBD 设备连接于车载计算机的 CAN 接口即可，而不需要拆下车门；确定车门故障点后，

拆下车门维修完毕，只需把 OBD 设备连接于车门的 CAN 接口，就能确认门是否已被修好，而不需要事先把车门装回。

一般而言，节点的计算能力与成本成正比。应用层的四项功能中，只有诊断功能可以根据具体产品而灵活裁减，其余的功能都是固定而且必须的。

诊断功能的可裁减性体现在两方面：实现方式以及支持的服务种类，这些都是直接影响着节点计算负荷的因素。

5.2.5.1 诊断方式

LIN 网络有三种方式来实现诊断功能，它们的差别在于传输层的复杂度，因为拆分/重组需要一定的计算量，这些方式如下表 5.13 所示，另外可参考图 5.2。

表 5.13 诊断功能的实现方式

	方式 A	方式 B	方式 C
传输层		支持 SF/FF/CF	视设计而定
协议层	信号携带帧	诊断帧	诊断帧(使用自定义 NAD)
计算量	最小	大	视设计而定
可移植性	好	好	差

5.2.5.2 诊断类型(Diagnostic Class)

逻辑节点功能越复杂，支持的服务越多，对逻辑节点的计算能力要求就越高。

依据诊断服务的数量，LIN 规范划分出三种不同的诊断类型——I 类、II 类和 III 类，适用于不同条件的逻辑节点。I 类最低，III 类最高，较高类型完全包含较低类型的功能。

I 类是所有诊断类型的公共部分，提供信号处理、识别、配置功能，诊断功能采用表 5.12 中的方式 A，这也是每个逻辑节点必备的服务。II 类节点增加了 UDS 定义的识别服务(注 1)，诊断方式一般采用表 5.12 中的方式 B。III 类节点相比 II 类节点，又增加了 UDS 定义的部分其它服务，此外，还增加了通过 LIN 总线在线升级的功能。

注：1. 请注意区别 UDS 定义的识别服务与 5.2.4 描述的识别服务。

诊断类型的关联如图 5.8 所示。

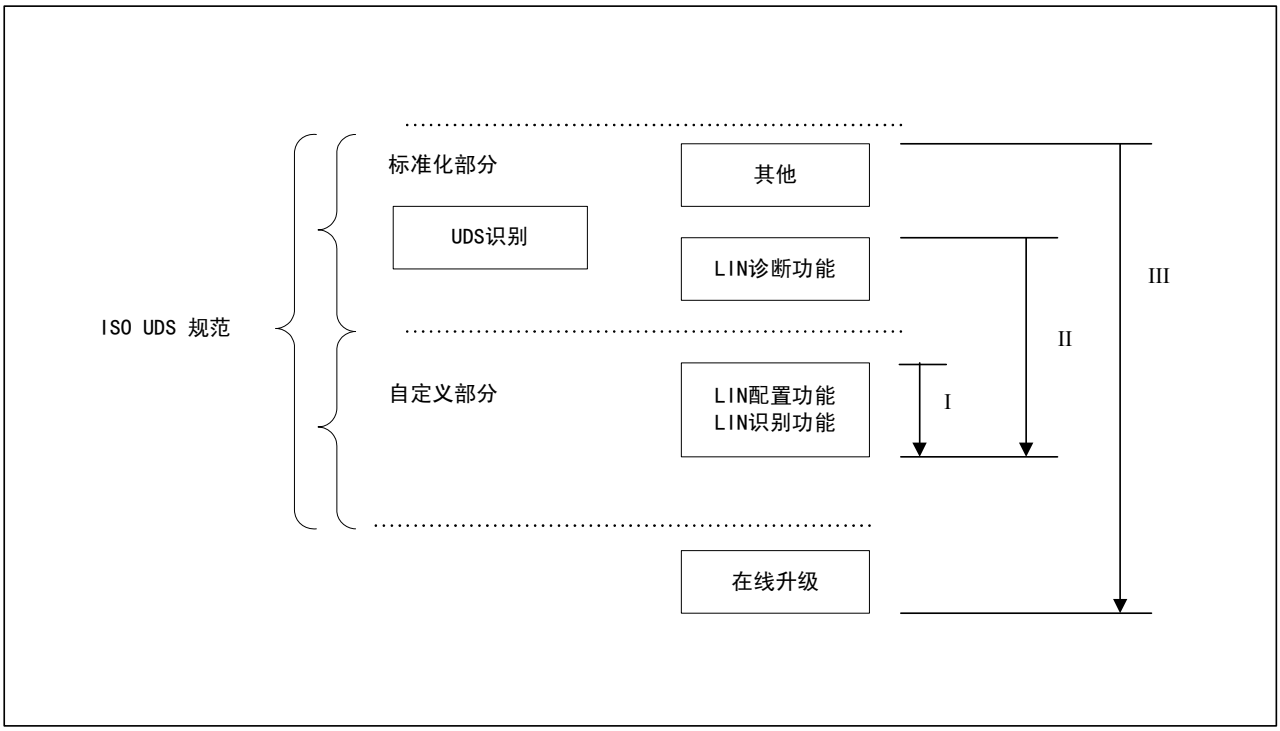
ISO UDS 规范
标准化部分
UDS识别
自定义部分
其他
LIN诊断功能
LIN配置功能
LIN识别功能
在线升级
I
II
III

图 5.8 诊断类型

5.3 参考资料

[1] LIN Transport Layer Specification Revision 2.1, LIN Consortium, 2006
[2] LIN Node configuration and Identification Specification Revision 2.1, LIN Consortium, 2006
[3] LIN Diagnostic Specification Revision 2.1, LIN Consortium, 2006
[4] LIN Application Program Interface Specification Revision 2.1, LIN Consortium, 2006
[5] LIN Diagnostic and Configuration Specification Revision 2.0, LIN Consortium, 2003
[6] LIN Application Program Interface Specification Revision 2.0, LIN Consortium, 2003
[7] ISO 14229:2006 Road vehicles – Unified diagnostic services (UDS) - Specification and requirements
[8] ISO 11898:2003 Road vehicles – Controller area network (CAN) -- Part 1: Data link layer and physical signaling
[9] ISO 15765:2004 Road vehicles – Diagnostics on Controller area network (CAN) -- Part 3: Network layer services

6. LIN 的 API

本章介绍 LIN 的 API 的概念、功能和一般用法，并以例子的形式介绍了调用 API 的一般流程。本章内容对应 LIN 规范的以下部分：

- LIN Application Program Interface Specification

6.1 什么是 API？

API 是一组“规约”，用来定义软件模块的使用方法。API 既可以是数据结构，也可以是若干个函数，还可以是它们的混合。

软件开发者可以把 API 看作是与软件模块的会话方式。应用程序和程序员既可以使用该模块的功能，又无需访问其源代码，或者理解其内部工作机制的细节。

API 对软件开发意义重大。软件规模日益庞大，常常需要把复杂系统划分成小的组成部分，或者重复使用代码，这时都会涉及到 API。

6.2 LIN 的 API

LIN 规范用 C 语言定义了 LIN 的 API，但未定义 API 的内部实现。

LIN 协会规定：对于采用 LIN 规范 2.x 版的 LIN 节点，如果用 C 语言开发应用程序，那么就必须使用 API，对采用 LIN 规范 1.x 版的 LIN 节点，可以不使用标准规定的 API。

按照用途，可以把 LIN 的 API 分为 3 类——核心 API、传输层 API 和配置与识别 API。三类 API 相对独立，彼此关联，如图 6.1 所示。

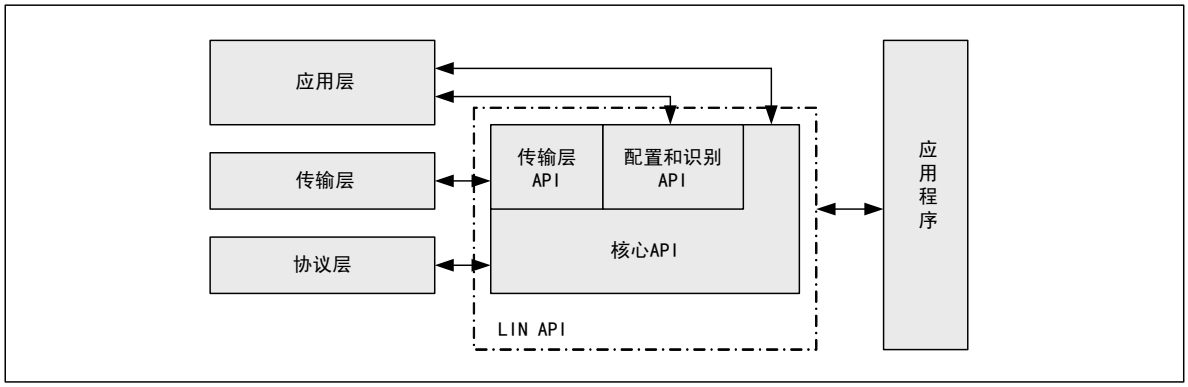

图 6.1 LIN API 及其关联

6.3 核心 API

核心 API 是 API 的基础，除了完成协议层的帧收发，LIN 应用层各项功能都要用到核心 API。

核心 API 包含多个函数，其中，l_sch_tick()(时基节拍管理)和 l_sch_set()(进度表管理)是与进度表相关的两个函数。其他的函数负责控制各种硬件协调工作，完成初始化、中断响应、比特流收发、字节缓冲、休眠、唤醒以及物理层的差错报告等功能。

LIN 规范把核心 API 分成 6 个组，如表 6.1 所示。

表 6.1 核心 API 的分类

类别	名称	说明
驱动与网络管理	l_sys_init	初始化整个 LIN 节点，包含应用层、传输层和协议层。在调用其它 API 函数前，必须首先调用此函数，例如在应用程序初始化阶段调用。
信号处理	l_bool_rd/l_bool_wr l_u8_rd/l_u8_wr l_u16_rd/l_u16_wr l_bytes_rd/l_bytes_wr	读写信号，用于信号处理功能。所有信号长度之和不超过一帧的容量。每个信号的长度可以是 1 比特(bool)、2~8 比特(u8)、9~16 比特(u16)和 2~8 字节(bytes)。
状态操作	l_flg_tst l_flg_clr	查询/清除状态标志，用于基本的诊断功能。为加强应用层与核心 API 的联系，允许自定义一些标志，由核心 API 置位，应用层可以查询和清除。
进度表管理(注 1)	l_sch_tick l_sch_set	l_sch_tick 是一个时间触发的函数，应该在每个时基节拍(参照 3.3 节)处调用。如果调用时恰逢帧时隙的开始，此函数将启动下一个入口；其他情况将完成诸如信号更新、读取通信状态报告(Status Word)、指示下一个进度表入口等功能。请参照参考资料[4]的 2.2.4 节和参考资料[3]的 6.2.5.8 节。 l_sch_set 用于选择当前有效的进度表(l_sch_tick 就是根据当前有效的进度表工作的)。l_sch_set 不但可以选择进度表，而且可以精确地选择某个进度表入口。
接口管理	l_ifc_init l_ifc_goto_sleep(注 1) l_ifc_wake_up l_ifc_ioctl l_ifc_rx l_ifc_tx l_ifc_aux l_ifc_read_status	l_ifc_init 用于物理层初始化。在执行“接口管理”的其它 API 函数之前必须首先调用。 l_ifc_goto_sleep 和 l_ifc_wake_up 分别实现休眠和唤醒操作。 l_ifc_ioctl 执行一些底层的自定义操作。 l_ifc_rx 和 l_ifc_tx 控制帧的收发。 l_ifc_aux 用于捕获帧头，这是个可选的函数，其功能可以包含在 l_ifc_rx 中。 l_ifc_read_status 用来读写状态报告。
逆向调用	l_sys_irq_disable l_sys_irq_restore	可选择的功能。某些物理层驱动可能不允许被中断打断，这两个函数用来屏蔽和恢复系统的中断。

注：1. 仅适用于主机节点。

6.4 传输层 API

从 LIN 规范 2.0 版开始，增加了传输层 API。

传输层 API 是为配置、识别和诊断这三项服务设置的，是应用层与协议层的接口。传输层 API 的功能包括：建立并管理 PDU 队列、收发 PDU 以及检查 PDU 的通信状态。传输层 API 接收应用层消息，调用核心 API 发送主机请求帧；收到从机应答帧时，传输层剥离协议层的帧头信息获得 PDU，送往应用层处理。

对于识别或配置服务，因为用于识别和配置的诊断帧已经预先安排在进度表内，所以传输层 API 只是在有关帧时隙到来时才工作，不影响核心 API 的进度表调度，不影响 LIN 的确定性。对于诊断服务，因为诊断请求通常来自诊断仪表或者上级网络(例如 CAN)，发生时机和频次不可预测，所以传输层 API 要能动态地产生诊断帧，并将诊断帧插入到当前的进度表里，这会影响 LIN 的确定性。

LIN 规范定义了两种传输层 API——Raw API 和 Cooked API，二者功能一致，区别在于对应用层消息的处理方式。两种 API 不建议混用，用户宜根据需要选用一种。如果节点需要监视通信细节，那么应该用 Raw API，它允许节点以 PDU 为单位处理信息。如果节点只需要转发消息而不需要关心消息内容，那么适合使用 Cooked API，它允许节点以消息为单位处理信息。

传输层 API 如表 6.2 所示。

表 6.2 传输层 API

类别	名称	说明
初始化	ld_init	ld_init 用于传输层初始化。
Raw	ld_put_raw ld_get_raw	完成 PDU 到帧的相互转换，并且在 MRF/SRF 的帧时隙到来时传输。
	ld_raw_tx_status ld_raw_rx_status	查询传输层的工作状态。
Cooked	ld_send_message ld_receive_message	完成消息到帧的相互转换，并且在 MRF/SRF 的帧时隙到来时传输。
	ld_tx_status ld_rx_status	查询传输层的工作状态。

6.5 配置与识别 API

从 LIN 规范 2.0 版开始，增加了配置与识别 API。用于支持应用层的配置功能和识别功能。

配置与识别 API 如表 6.3 所示。

表 6.3 配置与识别 API

类别	名称	说明
配置	ld_is_ready(注 1) ld_check_response(注 1)	ld_is_ready 用来检查上一次的服务请求的执行状况。 调用 ld_check_response 读取服务的执行情况。
	ld_assign_NAD(注 1) ld_conditional_change_NAD(注 1) ld_assign_frame_id_range(注 1)	给指定的从机节点分配 NAD/PID。
	ld_set_configuration(注 2) ld_save_configuration(注 1) ld_read_configuration(注 2)	从机节点调用 ld_set_configuration 设置初始配置项(NAD 和 PID)。 主机节点调用 ld_save_configuration，请求从机节点保存当前的配置项。 从机节点调用 ld_read_configuration，整理当前配置项并保存。
识别	ld_read_by_id(注 1)	读取从机节点的产品代号或者其他参数。
	ld_read_by_id_callout(注 2)	向主机节点发送自定义的数据。

注：1. 仅适用于主机节点。

2. 仅适用于从机节点。

配置 API 除了实现具体的服务项目，还可以向应用程序报告服务的执行情况。

识别 API 包含两个函数：ld_read_by_id 和 ld_read_by_id_callout。ld_read_by_id 仅供主机节点使用，主机节点只能通过它获得从机节点的硬件信息，例如产品代号等等；ld_read_by_id_callout 仅供从机节点使用，这是一个从 API 向节点应用程序的逆向调用，它并不是必须的，专门用于实现用户自定义的服务。

6.6 注意事项

6.6.1 兼容性

API 的兼容性体现在两个方面，一是不同版本 API 之间的兼容性，二是 API 对帧收发硬件的兼容性。

API 版本之间的兼容性如表 6.4 所示。

表 6.4 不兼容的 API 列表

服务	名称	LIN 1.X	LIN 2.0	LIN 2.1
接口管理	l_ifc_connect l_ifc_disconnect	YES	YES	N/A
	l_ifc_goto_sleep l_ifc_wake_up l_ifc_read_status	N/A	YES	YES
识别	ld_read_by_id	N/A	YES	YES
	ld_read_by_id_callout	N/A	N/A	YES
通信管理	ld_is_ready ld_check_response	N/A	YES	YES
传输层	ld_put_raw ld_get_raw ld_raw_tx_status ld_raw_rx_status ld_send_message ld_receive_message ld_tx_status ld_rx_status	N/A	YES	YES
给指定的从机节点分配 NAD	ld_assign_NAD ld_conditional_change_NAD	N/A	YES	YES
给指定的从机节点分配 PID	ld_assign_frame_id	N/A	YES	N/A
配置项相关操作	ld_assign_frame_id_range	N/A	N/A	YES
	ld_save_configuration ld_read_configuration ld_set_configuration	N/A	N/A	YES

不同的硬件需要使用不同的 API。LIN API 的实现通常都是与帧收发硬件密切相关的，不能简单挪用。

6.6.2 开发工具

目前已经有一些商品化的 LIN 开发工具。要开发 LIN 的应用，商品化的开发工具并不是必须的，不过，此类工具确实能提高开发效率，尤其是处理那些同时容纳不同 LIN 规范版本的节点的网络。

图 6.2 显示了此类开发工具的工作原理。API 的实现可以是独立的若干个库文件，也可能包含在某种开发工具之中。API 通常不能直接被调用，需要配合若干外部附属模块(例如驱动函数)以及映射文件(例如用宏定义实现节点端口与 API 库文件之间的衔接)。库文件与附属模块、映射文件一起，在编译阶段添加到用户代码中，请参照参考资料[11]、[12]。

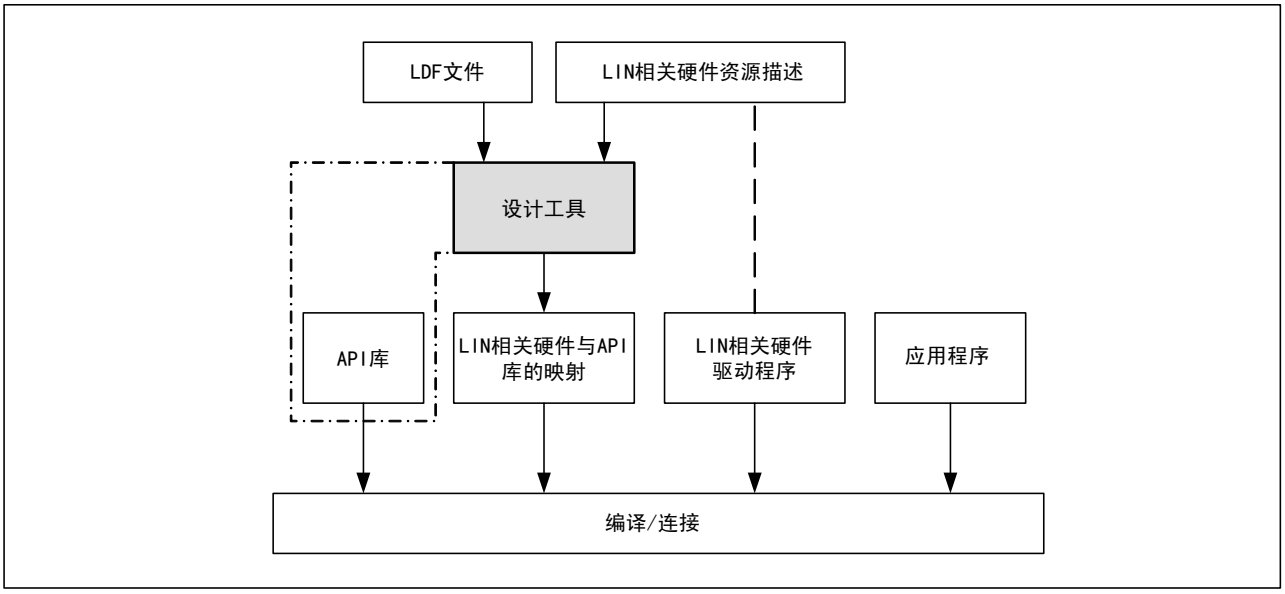

图 6.2 LIN 设计工具示意图

6.7 API 使用示例

参考资料[11]给出了一个 LIN 2.0 版的 API 的示例，从中可以看出 API 的调用次序。该示例包括两部分：在从机节点初始化阶段需要执行的 API，以及在从机节点应用程序中调用 API 的方法。

6.7.1 从机节点初始化

```
extern unsigned char lin_SomeCotrol_init( void );
void PowerON_Reset(void)
{
HardwareSetup();/*  系统初始化  */
if( l_sys_init() ) {
/* LIN API 初始化失败  */
sleep();
}
else {
    if( lin_SomeCotrol_init() ) {
    /* LIN 相关的模块初始化失败，例如传感器、执行器  */
        sleep();
    }
    }
    /*  其他系统要求的功能  */
main();
return;
}

/*  帧收发硬件的驱动程序入口  */
const T_Lib_Slave_Handle Slave_handle = {
Lin_Drv_Init,
Lin_Drv_HeaderIn,
Lin_Drv_Pid_RecvReq,
Lin_Drv_SendData,
Lin_Drv_RecvData,
Lin_Drv_SendRecvFinish,
Lin_Drv_LinBus_Enable,
Lin_Drv_LinBus_Disable,
Lin_Drv_WakeUp
};

/* LIN 网络初始化  */
unsigned char lin_SomeCotrol_init( void )
```

```
{
unsigned char rtn;

rtn = 0;
if( l_ifc_ioctl( 0, LIN_ENTRY_SLAVE_DRV, &Slave_handle ) ) {
    /*  帧收发硬件的驱动程序初始化失败  */
    rtn = 1u;
}
else {
    l_ifc_init(0);                  /* LIN 端口初始化  */
    if( l_ifc_connect(0) ) {
        /* LIN 端口初始化失败  */
        rtn = 1u;
        }
    else {
        /*  其他必要的操作  */
        }
    }
return rtn;
}
```

6.7.2 从机节点主程序

```
#include "sfr_r825.h"
#include "Lin_DrvR8C.h"
#include "lin20.h"
void lin_application( void );
/*****************************/
/* Main Function                */
/*****************************/
void main(void)
{
while( 1 ) {
    /* ......Something to do */

        lin_application();

    /* ......Something to do */
    }
}
```

```
/*******************************/
/* LIN Application Function        */
/*******************************/
extern l_flg Lin_Frm_FrameMst0_flg;
extern l_flg Lin_Frm_FrameU1_flg;
extern l_flg Lin_Frm_FrameU2_flg;
extern l_flg Lin_Frm_FrameU3_flg;
extern l_flg Lin_Frm_FrameEve0_flg;
extern l_flg Lin_Frm_FrameSlv0_flg;
extern l_flg Lin_Sig_Command_flg;
extern T_Signal Lin_Sig_Status_Slv0;
extern T_Signal Lin_Sig_Status_Slv1;
extern T_Signal Lin_Sig_Command;
void lin_application( void )
{
l_u8       data[8];
l_u16      status;

/*  判断：是否收到了新的帧？  */
if( 0 != l_flg_tst(&Lin_Frm_FrameU1_flg) ) {
      l_flg_clr( &Lin_Frm_FrameU1_flg );
            /*  根据收到的帧执行相应的操作  */
      }
else if( 0 != l_flg_tst(&Lin_Frm_FrameMst0_flg) ) {
      l_flg_clr( &Lin_Frm_FrameMst0_flg );
      /*  根据收到的帧执行相应的操作  */
}

/*  判断：帧是否已经发出？  */
if( 0 != l_flg_tst(&Lin_Frm_FrameU2_flg) ) {
      l_flg_clr( &Lin_Frm_FrameU2_flg );
      /*  执行发送结束之后的操作  */
}
else if( 0 != l_flg_tst(&Lin_Frm_FrameU3_flg) ) {
            l_flg_clr( &Lin_Frm_FrameU3_flg );
      /*  执行发送结束之后的操作  */
}
```

```
else if( 0 != l_flg_tst(&Lin_Frm_FrameEve0_flg) ) {
    l_flg_clr( &Lin_Frm_FrameEve0_flg );
    /* 执行发送结束之后的操作 */
}
else if( 0 != l_flg_tst(&Lin_Frm_FrameSlv0_flg) ) {
    l_flg_clr( &Lin_Frm_FrameSlv0_flg );
    /* 执行发送结束之后的操作 */
}

status = l_ifc_read_status( 0 );

/* 处理可能出现的应答错误 */
if( status & 0x0001u ) {
    /* 出现应答错误，执行相关操作 */
}
if( LD_DATA_AVAILABLE == ld_raw_rx_status(0) ) {
    ld_get_raw( 0, data );
}
/* 判断：来自主节点的信号被更新了吗？ */
if( 0 != l_flg_tst(&Lin_Sig_Command_flg) ) {
    l_flg_clr( &Lin_Sig_Command_flg );
    if( 0x1234u == l_u16_rd(&Lin_Sig_Command) ) {
        l_u16_wr( &Lin_Sig_Status_Slv0, 0x0101u );
        l_u16_wr( &Lin_Sig_Status_Slv1, 0x0201u );
        /* 其他相关操作 */
    }
    else if( 0x5678u == l_u16_rd(&Lin_Sig_Command) ) {
        l_u16_wr( &Lin_Sig_Status_Slv0, 0x0100u );
        l_u16_wr( &Lin_Sig_Status_Slv1, 0x0200u );
        /* 其他相关操作 */
    }
}
/* 监测休眠命令 */
if( status & 0x0008u ) {
    /* LIN 端口休眠的相关操作 */
}
return;
}
```

6.8 参考资料

[1] LIN API Recommended Practice Revision 1.3, LIN Consortium, 2002
[2] LIN Application Program Interface Specification Revision 2.0, LIN Consortium, 2003
[3] LIN Application Program Interface Specification Revision 2.1, LIN Consortium, 2006
[4] LIN Protocol Specification Revision 2.1, LIN Consortium, 2006
[5] H8/300H Tiny Series H8/36049 Group LIN(Local Interconnect Network) : Master Volume, Renesas Technology, 2003
[6] H8/300H Tiny Series LIN(Local Interconnect Network) Application Note: Master, Renesas Technology, 2003
[7] H8/300H Tiny Series H8/36014 Group LIN(Local Interconnect Network) : Slave Volume, Renesas Technology, 2003
[8] H8/3664F/3694F/36014F Series LIN(Local Interconnect Network) Application Note: Slave, Renesas Technology, 2003
[9] H8/3687F Series LIN(Local Interconnect Network) Application Note: Slave, Renesas Technology, 2003
[10] R8C/Tiny Series R8C/11 Group LIN(Local Interconnect Network) Application Note: Slave Volume, Renesas Technology, 2006
[11] R8C/Tiny Series R8C/25 Group LIN(Local Interconnect Network) Application Note: Slave Volume, Renesas Technology, 2006
[12] LINkits LIN Evaluation Boards, Freescale Semiconductors, 2007

7. 工作流

本章介绍了 LIN 工作流的概念，以及节点性能文件和 LIN 描述文件的内容，对应着 LIN 规范的以下部分：

- LIN Node Capability Language Specification
- LIN Configuration Language Specification

为了实现从机节点入网的“即插即用”，LIN 规范标准化了 LIN 网络从设计到生成的工作流程，如图 7.1 所示。

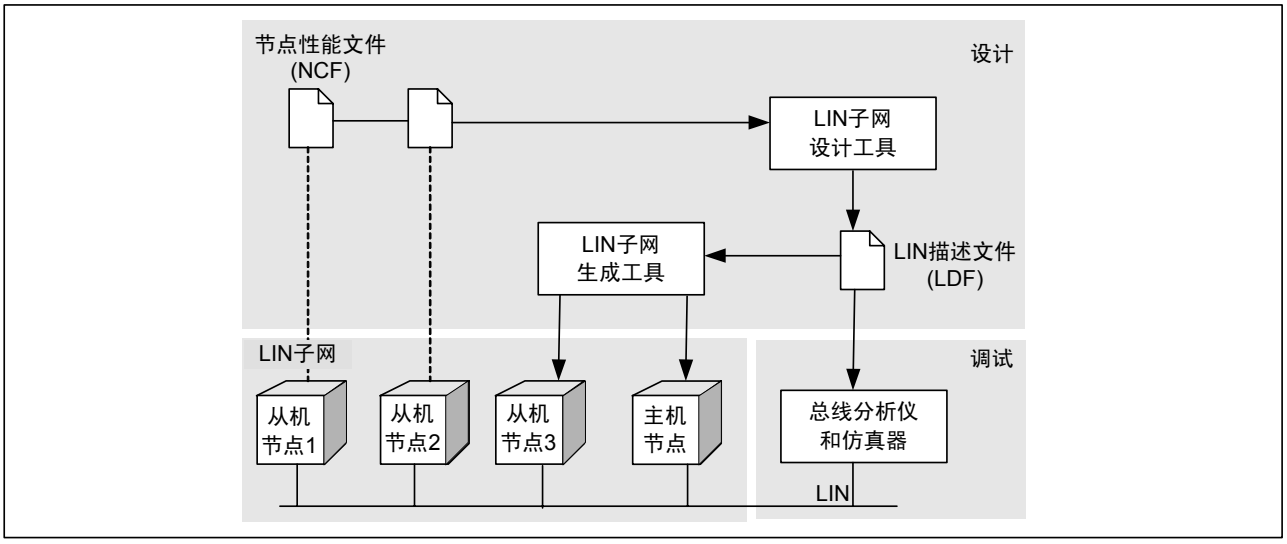

图 7.1 工作流

其中节点性能文件(NCF)定义了节点名称和节点的属性值，包括产品代号、位速率、帧的定义等信息。LIN 子网设计工具收集到节点性能文件的信息，自动生成 LIN 描述文件(LDF)。LDF 包含了整个子网的信息，包括所有的信号和帧的声明，以及进度表等信息。LDF 文件还可以作为调试时总线分析仪和仿真器的输入。LIN 子网生成工具根据 LDF 生成各种通信驱动，可以建立起通信子网，也可以将具备节点性能文件的现成节点加入到已经建立好的通信子网中，并在网络进入运行前排除掉可能产生的冲突。

7.1 节点性能文件

节点性能文件包含节点的物理特性、帧和信号的定义等内容，如表 7.1 所示。

表 7.1 节点性能文件

<table>
<tr><td colspan="6">节点性能文件</td></tr>
<tr><td colspan="2">全局定义</td><td colspan="4">LIN 语言版本</td></tr>
<tr><td colspan="6">节点定义(注 1)</td></tr>
<tr><td rowspan="20">节点名称</td><td rowspan="6">概要定义</td><td colspan="4">LIN 协议版本</td></tr>
<tr><td colspan="4">厂商 ID(Suppliler ID)</td></tr>
<tr><td colspan="4">功能 ID(Function ID)</td></tr>
<tr><td colspan="4">可变 ID(Variant ID)</td></tr>
<tr><td colspan="4">位速率</td></tr>
<tr><td colspan="4">发送唤醒使能</td></tr>
<tr><td rowspan="8">诊断定义</td><td colspan="4">初始 NAD</td></tr>
<tr><td colspan="4">诊断类型</td></tr>
<tr><td colspan="4">P2_min(注 2)</td></tr>
<tr><td colspan="4">ST_min(注 3)</td></tr>
<tr><td colspan="4">N_As_timeout(注 4)</td></tr>
<tr><td colspan="4">N_Cr_timeout(注 5)</td></tr>
<tr><td colspan="4">支持的 SID</td></tr>
<tr><td colspan="4">诊断传输层最大消息长度(单位:字节)</td></tr>
<tr><td rowspan="7">帧定义</td><td colspan="4">帧类别(发布或收听)</td></tr>
<tr><td colspan="4">帧名称</td></tr>
<tr><td rowspan="4">帧属性</td><td colspan="3">长度</td></tr>
<tr><td colspan="3">最小时间</td></tr>
<tr><td colspan="3">最大时间</td></tr>
<tr><td colspan="3">事件触发帧</td></tr>
<tr><td rowspan="3">信号定义</td><td colspan="3">信号名称</td></tr>
<tr><td rowspan="4">信号属性</td><td>初始值</td></tr>
<tr><td>保留位数</td></tr>
<tr><td>偏移量(注 6)</td></tr>
<tr><td>编码类型名称</td></tr>
<tr><td rowspan="4">信号编码类型定义</td><td colspan="4">编码类型名称</td></tr>
<tr><td rowspan="3">逻辑值</td><td colspan="3">逻辑值</td></tr>
<tr><td colspan="3">信号值</td></tr>
<tr><td colspan="3">文本信息</td></tr>
<tr><td>物理值</td><td colspan="3">物理值</td></tr>
</table>

			最小值
			最大值
			缩放倍数(注 7)
			偏移量(注 7)
			文本信息
		BCD 值	
		ASCII 值	
	状态管理	应答错误名称	
		错误状态信号	
		发布节点	
	自由文本定义		

(蓝色笔标出的部分为可选项，红色笔标出的部分为选择其中的一项或几项。)

注：1. 一个节点描述文件可以定义多个节点，在同一个文件中的节点不能重名。
2. P2_min: 从 LIN 子网接收到主机请求帧到 LIN 的从机节点准备好数据发送应答之间的最小时间间隔。
3. ST_min: 从机节点准备接收下一个帧(主机请求帧)或准备发送下一个帧(从机应答帧)的应答部分所需要的最小准备时间。
4. N_As_timeout: 从传输层向 LIN 子网请求一个帧开始到传输层确认请求的帧传输结束之间的超时间隔，即从发送方看，发送 LIN 帧(MRF 或 SRF)的超时间隔。
5. N_Cr_timeout: 在复合帧(首帧+续帧的形式，参照 5.3 节)的传输中，传输层接收到的首帧和续帧、续帧和续帧之间的超时间隔。
6. 信号在字节中的偏移量如表 7.2 举例。

表 7.2 信号在字节中的偏移量示例

10 位信号 S(LSB)在 4 字节中的偏移量为 16。																															
第 0 字节								第 1 字节								第 2 字节								第 3 字节							
																S	S	S	S	S	S	S	S	S	S						
0							7	8							15	16							23	24							31

7. 物理值 = 缩放倍数 × 原始值 + 偏移量

7.1.1 节点性能文件举例说明

以下为一个节点性能文件的例子和说明。

```
node_capability_file;                                  //节点性能文件
LIN_language_version = “2.1”;                          //LIN 语言版本 2.1
node step_motor {                                      //节点定义。节点名称 step_motor
general {                                              //摘要定义
LIN_protocol_version = “2.1”;                          //LIN 协议版本 2.1
supplier = 0x0005; function = 0x0020; variant = 1;     //厂商 ID，功能 ID，可变 ID
bitrate = automatic min 10 kbps max 20 kbps;           //比特率自动选择范围[10kbps-20kbps]
sends_wake_up_signal = “yes”;                          //该节点可以发送 wakeup 信号
}
diagnostic {                                           //诊断定义
NAD = 1 to 3;                                          //初始 NAD 可以选择的值：1~3
diagnostic_class = 2;                                  //支持的诊断类别为第 2 类
P2_min = 100 ms; ST_min = 40 ms;                       //参照表 7.1 注释
support_sid { 0xB0, 0xB2, 0xB7 };                      //节点支持的全部 SID
```

```
}
frames {                                                    //帧定义
publish node_status {                                       //帧名称为 node_status，作为发布节点
length = 4; min_period = 10 ms; max_period = 100 ms;

                                                            //帧长度 4 字节，传输最小、最大时间
signals {                                                   //信号定义(该帧中包含的信号)
state {size = 8; init_value = 0; offset = 0;}               //state 信号：长度 8bit，
                                                            //初始值为 0，偏移量为 0(即 bit[7：0])
fault_state {size = 2; init_value = 0; offset = 9; fault_enc;}

                                                            //fault_state 信号：长度 2bit，初始值为 0，
                                                            //偏移量为 9(即 bit9、bit10)，
                                                            //编码类型名称为 fault_enc
error_bit {size = 1; init_value = 0; offset = 8;}

                                                            //error_bit 信号：长度 1bit，
                                                            //初始值为 0，偏移量为 8
angle {size = 16; init_value = {0x22, 0x11}; offset = 16;}

                                                            //angle 信号：长度 16bit，
                                                            //初始值为 0x22，0x11，偏移量 16
}
}
subscribe control {                                         //帧名称为 control，作为收听节点
length = 1; max_period = 100 ms;                            //帧长度 1 字节，传输最大时间 100ms
signals {                                                   //信号定义(该帧中包含的信号)
command {size = 8; init_value = 0; offset = 0; position;}

                                                            //command 信号：长度 8bit，初始值为 0，
                                                            //偏移量为 0，编码类型名称为 position
}
}
}
encoding {                                                  //信号编码类型
position {physical_value 0, 199, 1.8, 0, “deg”;}            //编码类型名称为 position：
                                                            //物理值，最小值 0，最大值 199，
                                                            //缩放倍数 1.8，偏移量为 0，
                                                            //文本信息为“deg”
fault_enc {logical_value, 0, “no result”;

                                                            //编码类型名称为 fault_enc：
                                                            //逻辑值，信号值 0，文本信息为“no result”
logical_value, 1, “failed”;                                 //逻辑值，信号值 1，文本信息为“failed”
logical_value, 2, “passed”;}                                //逻辑值，信号值 2，文本信息为“passed”
}
status_management { response_error = error_bit;             //状态管理：表示 response_error 的信号名称
fault_state_signals = fault_state; }                        //表示 fault_state_signals 的信号名称
free_text { “step_motor signal values outside 0 - 199 are ignored” }
                                                            //自由文本定义
}
```

7.2 LIN 描述文件

LIN 描述文件对整个 LIN 网络进行了描述，也包含了要监测 LIN 网络所必需的信息，包含 LIN 网络内所有节点、帧和信号的信息以及进度表等内容，如表 7.3 所示。

表 7.3 LIN 描述文件

LIN 描述文件			
全局定义	LIN 协议版本		
	LIN 语言版本		
	LIN 位速率定义		
	通道后缀名称定义(注 1)		
节点定义	参与节点	主机节点	节点名称
			时基
			抖动
		从机节点	节点名称列表
	节点属性	节点名称	
		LIN 协议版本	
		配置 NAD(从机节点地址)	
		初始 NAD	
		属性定义	产品 ID
			应答错误名称
			错误状态信号
			P2_min
			ST_min
			N_As_timeout
			N_Cr_timeout
			符合 LIN2.0 的可配置的帧
			符合 LIN2.1 的可配置的帧
	节点组合定义(注 2)	配置名称	组合节点名称(物理节点)
			逻辑节点名称列表 (包含于该物理节点中)
信号定义	标准信号	信号名称	信号大小
			初始值
			该信号的发布节点
			该信号的收听节点列表
	诊断信号	信号名称	
		信号大小	
		信号在帧中的偏移量	
	信号组(注 3)	信号组名称列表	组大小
			信号名称及在组中的偏移量列表

帧定义	无条件帧定义	帧名称	
		帧 id	
		该帧的发布节点	
		帧大小	
		信号名称及在帧中的偏移量列表	
	偶发帧定义	偶发帧名称	
		关联的无条件帧名称列表	
	事件触发帧定义	事件触发帧名称	
		冲突解决进度表	
		事件触发帧的帧 ID	
		关联的无条件帧名称列表	
	诊断帧定义	主机请求帧	主机请求帧包含信号及偏移量
		从机应答帧	从机应答帧包含信号及偏移量
进度表定义	进度表名称		
	命令	帧名称	
		主机请求帧	
		从机应答帧	
		分配 NAD	
		条件改变 NAD	
		数据 DUMP	
		保留配置	
		分配 PID 范围	
		自由格式	
		分配 PID	
	命令延迟时间(即帧时隙)		
附加信息	信号编码类型定义	信号编码类型名称	
		逻辑值	
		物理值	
		BCD 值	
		ASCII 值	
	信号表示定义	信号编码类型名称	
		该编码类型对应的信号名称列表	

(蓝色笔标出的部分为可选项，红色笔标出的部分为选择其中的一项或几项。)

注：1. 当一个主机 ECU 与多个 LIN 通信子网相连时，会包含多个 LDF 文件，为了避免命名冲突，规定 LDF 文件中的所有对象必须使用通道后缀名称。
2. 当存在从机物理节点对应多个逻辑节点时，节点组合定义可以让主机节点用同一软件无需作任何改变就可处理多个从机节点的配置。
3. 只应用于依据 LIN1.3 版本设计的节点。

7.2.1 LIN 描述文件举例说明

以下为一个 LIN 描述文件的例子和说明。

```
LIN_description_file;                                   //LIN 描述文件
LIN_protocol_version = “2.1”;                           //LIN 协议版本
LIN_language_version = “2.1”;                           //LIN 语言版本
LIN_speed = 19.2 kbps;                                  //LIN 通信速度
Channel_name = “DB”;                                    //通道后缀名称“DB”

Nodes {                                                 //节点定义
Master: CEM, 5 ms, 0.1 ms;                              //主机节点，名称：CEM，时基：5ms，抖动：0.1ms
Slaves: LSM, RSM;                                       //从机节点，LSM，RSM
}

Node_attributes {                                       //节点属性定义
LSM {                                                   //节点名称为 LSM 的节点的定义
LIN_protocol = “2.1”;                                   //该节点依据 LIN 协议 2.1 设计
configured_NAD = 0x20;                                  //配置 NAD 为 0x20
initial_NAD = 0x01;                                     //初始 NAD 为 0x01
product_id = 0x4A4F, 0x4841;                            //产品 ID：厂商 ID 为 0x4A4F，功能 ID 为 0x4841
response_error = LSMerror;                              //应答错误名称为：LSMerror
fault_state_signals = IntTest;                          //错误状态信号为 IntTest
P2_min = 150 ms;                                        //参照节点性能文件注释
ST_min = 50 ms;                                         //参照节点性能文件注释
configurable_frames { CEM_Frm1; LSM_Frm1; LSM_Frm2;}
                                                        //可配置的帧列表
}
RSM {                                                   //节点名称为 RSM 的节点的定义
LIN_protocol = “2.0”;                                   //该节点依据 LIN 协议 2.0 设计
configured_NAD = 0x20;                                  //配置 NAD 为 0x20
product_id = 0x4E4E, 0x4553, 1;                         //产品 ID：厂商 ID 为 0x4E4E，
                                                        //功能 ID 为 0x4553，可变 ID 为 1
response_error = RSMerror;                              //应答错误名称：RSMerror
P2_min = 150 ms;                                        //参照节点性能文件注释
ST_min = 50 ms;                                         //参照节点性能文件注释
configurable_frames {CEM_Frm1 = 0x0001; LSM_Frm1 = 0x0002; LSM_Frm2 = 0x0003;}
                                                        //可配置的帧列表
}
}
Signals {                                               //信号定义
IntLightsReq: 2, 0, CEM, LSM, RSM;                      //信号名称：IntLightsReq，
                                                        //长度：2，初始值：0，
                                                        //发布节点：CEM，收听节点：LSM、RSM
```

```
    RightIntLightsSwitch: 8, 0, RSM, CEM;          //信号名称：RightIntLightsSwitch，
                                                   //长度：8，初始值：0，
                                                   //发布节点：RSM，收听节点：CEM
    LeftIntlLightsSwitch: 8, 0, LSM, CEM;          //信号名称：LeftIntlLightsSwitch，
                                                   //长度：8，初始值：0，
                                                   //发布节点：LSM，收听节点：CEM
    LSMerror, 1, 0, LSM, CEM;                      //信号名称：LSMerror，长度：1，初始值：0，
                                                   //发布节点：LSM，收听节点：CEM
    RSMerror, 1, 0, LSM, CEM;                      //信号名称：RSMerror，长度：1，初始值：0，
                                                   //发布节点：LSM，收听节点：CEM
    IntTest, 2, 0, LSM, CEM;                       //信号名称：IntTest，长度：2，初始值：0，
                                                   //发布节点：LSM，收听节点：CEM
    }
    Frames {                                       //帧定义
    CEM_Frm1: 0x01, CEM, 1 {                       //帧名称：CEM_Frm1，帧 ID：0x01，
                                                   //该帧的发布节点：CEM，数据段为 1 个字节
    InternalLightsRequest, 0;                      //包含的信号名称
                                                   //为 InternalLightsRequest，
                                                   //在帧中的偏移量为 0
    }
    LSM_Frm1: 0x02, LSM, 2 {                       //帧名称：LSM_Frm1，帧 ID：0x02，
                                                   //该帧的发布节点：LSM，数据段为 2 个字节
    LeftIntLightsSwitch, 0;                        //包含的信号名称为 LeftIntLightsSwitch，
                                                   //在帧中的偏移量为 0
    }
    LSM_Frm2: 0x03, LSM, 1 {                       //帧名称：LSM_Frm2，帧 ID：0x03，
                                                   //该帧的发布节点：LSM，数据段为 1 个字节
    LSMerror, 0;                                   //包含的信号名称为 LSMerror，帧中偏移量为 0
    IntError, 1;                                   //包含的信号名称为 IntError，帧中偏移量为 1
    }
    RSM_Frm1: 0x04, RSM, 2 {                       //帧名称：RSM_Frm1，帧 ID：0x04，
                                                   //该帧的发布节点：RSM，数据段为 2 个字节
    RightIntLightsSwitch, 0;                       //包含的信号名称为 RightIntLightsSwitch，
                                                   //在帧中的偏移量为 0
    }
    RSM_Frm2: 0x05, RSM, 1 {                       //帧名称：RSM_Frm2，帧 ID：0x05，
                                                   //该帧的发布节点：RSM，数据段为 1 个字节
    RSMerror, 0;                                   //包含的信号名称为 RSMerror，帧中偏移量为 0
    }
    }
    Event_triggered_frames {                       //事件触发帧定义
    Node_Status_Event : Collision_resolver, 0x06, RSM_Frm1, LSM_Frm1;
                                                   //事件触发帧名称：Node_Status_Event，
                                                   //冲突解决进度表：Collision_resolver，
                                                   //事件触发帧的帧 ID：0x06
                                                   //关联的无条件帧为 RSM_Frm1，LSM_Frm1
    }
    Schedule_tables {                              //进度表定义
    Configuration_Schedule {                       //进度表名称为：Configuration_Schedule
```

```
AssignNAD {LSM} delay 15 ms;                    //给节点 LSM 分配 NAD，帧时隙为 15ms
AssignFrameIdRange {LSM, 0} delay 15 ms;
                                                //给节点 LSM 从第 0 帧开始分配 PID，
                                                //帧时隙为 15ms
AssignFrameId {RSM, CEM_Frm1} delay 15 ms;
                                                //给节点 RSM 的帧 CEM_Frm1 分配 PID，
                                                //帧时隙为 15ms
AssignFrameId {RSM, RSM_Frm1} delay 15 ms;
                                                //给节点 RSM 的帧 RSM_Frm1 分配 PID，
                                                //帧时隙为 15ms
AssignFrameId {RSM, RSM_Frm2} delay 15 ms;
                                                //给节点 RSM 的帧 RSM_Frm2 分配 PID，
                                                //帧时隙为 15ms
}
Normal_Schedule {                               //进度表名称为：Normal_Schedule
CEM_Frm1 delay 15 ms;                           //帧 CEM_Frm1，帧时隙 15ms
LSM_Frm2 delay 15 ms;                           //帧 LSM_Frm2，帧时隙 15ms
RSM_Frm2 delay 15 ms;                           //帧 RSM_Frm2，帧时隙 15ms
Node_Status_Event delay 10 ms;                  //事件触发帧 Node_Status_Event，
                                                //帧时隙 10ms
}
MRF_schedule {                                  /进度表名称为：MRF_schedule
MasterReq delay 10 ms;                          //主机请求帧，帧时隙 10ms
}
SRF_schedule {                                  //进度表名称为：SRF_schedule
SlaveResp delay 10 ms;                          //从机应答帧，帧时隙 10ms
}
Collision_resolver {                            //发生冲突时需保证非事件触发帧的传输时序
                                                //进度表名称：Collision_resolver
CEM_Frm1 delay 15 ms;                           //帧 CEM_Frm1，帧时隙 15ms
LSM_Frm2 delay 15 ms;                           //帧 LSM_Frm2，帧时隙 15ms
RSM_Frm2 delay 15 ms;                           //帧 RSM_Frm2，帧时隙 15ms
RSM_Frm1 delay 10 ms;                           //轮询 RSM 节点
                                                //帧 RSM_Frm1，帧时隙 10ms
CEM_Frm1 delay 15 ms;                           //帧 CEM_Frm1，帧时隙 15ms
LSM_Frm2 delay 15 ms;                           //帧 LSM_Frm2，帧时隙 15ms
RSM_Frm2 delay 15 ms;                           //帧 RSM_Frm2，帧时隙 15ms
LSM_Frm1 delay 10 ms;                           //轮询 LSM 节点
                                                //帧 LSM_Frm1，帧时隙 10ms
}
}
Signal_encoding_types {                         //信号编码类型：Signal_encoding_types
Dig2Bit {                                       //信号编码类型名称：Dig2Bit
logical_value, 0, “off”;                        //逻辑值 0，代表“off”
logical_value, 1, “on”;                         //逻辑值 1，代表“on”
logical_value, 2, “error”;                      //逻辑值 2，代表“error”
logical_value, 3, “void”;                       //逻辑值 3，代表“void”
}
ErrorEncoding {                                 //信号编码类型名称：ErrorEncoding
```

```
    logical_value, 0, “OK”;                                  //逻辑值 0，代表“OK”
    logical_value, 1, “error”;                               //逻辑值 1，代表“error”
    }
    FaultStateEncoding {                                     //信号编码类型名称：FaultStateEncoding
    logical_value, 0, “No test result”;                      //逻辑值 0，代表“No test result”
    logical_value, 1, “failed”;                              //逻辑值 1，代表“failed”
    logical_value, 2, “passed”;                              //逻辑值 2，代表“passed”
    logical_value, 3, “not used”;                            //逻辑值 3，代表“not used”
    }
    LightEncoding {                                          //信号编码类型名称：LightEncoding
    logical_value, 0, “Off”;                                 //逻辑值：0，代表：“Off”
      hysical_value, 1, 254, 1, 100, “lux”;                  //物理值：1，最小值：1，
                                                             //最大值：254，缩放倍数：1，
                                                             //偏移量：100，文字信息：“lux”
    logical_value, 255, “error”;                             //逻辑值：255，代表：“error”
    }
    }
    Signal_representation {                                  //信号表示定义
    Dig2Bit: InternalLightsRequest;                          //应用信号编码类型为 Dig2Bit 的信号
                                                             //InternalLightsRequest
    ErrorEncoding: RSMerror, LSMerror;                       //应用信号编码类型为 ErrorEncoding 的
                                                             //信号 RSMerror，LSMerror
    FaultStateEncoding: IntError;                            //应用信号编码类型为
                                                             //FaultStateEncoding 的信号 IntError
    LightEncoding: RightIntLightsSwitch, LefttIntLightsSwitch;
                                                             //应用信号编码类型为 LightEncoding 的
                                                             //信号 RightIntLightsSwitch，
                                                             //LefttIntLightsSwitch
    }
```

关键词对照表

英文	中文
API (Application Program Interface)	应用编程接口
BRG (Baud Rate Generator)	波特率发生器
Bitrate	位速率
Break	同步间隔
Break Delimiter	同步间隔段间隔符
Break Field	同步间隔段
Bus Transceiver	总线收发器
Bus Wakeup	总线唤醒
Byte Field	字节域
Checksum	校验和
Classic Checksum	标准型校验和
Cluster	子网，网络
Cluster Design	子网设计，参照图 7.1 的 LIN 子网设计工具
Cluster Generation	子网生成，参照图 7.1 的 LIN 子网生成工具
Collision Resolving Schedule	冲突解决进度表
Configuration	配置项
CF (Consecutive Frames)	续帧
CAN (Controller Area Network)	控制器局域网
Data	数据
Diagnostic Class	诊断类型
Diagnostic Frame	诊断帧
Dominant	显性电平
EMC (Electromagnetic Compatibility)	电磁兼容性
EMI (Electromagnetic Interference)	电磁干扰
ECU (Electronic Control Unit)	电子控制单元
ESD (Electrostatic Discharge)	静电放电
Enhanced Checksum	增强型校验和
Entry	入口
Event Triggered Frame	事件触发帧
FF (First Frame)	首帧
Frame ID	帧 ID
Frame Slot	帧间隔
Function ID	功能 ID
Hardware LIN	硬件 LIN
Header	帧头

Identification	识别
Inter-byte Space	字节间间隔
Inter-frame Space	帧间隔
Interface	接口
Jitter	抖动
Jump-start	蓄电池串联
LSB (Least Significant Bit)	最低有效位
Limp Home Mode	自我保护/安全模式
LDF (LIN Description File)	LIN 描述文件
LIN Module	LIN 模块
LIN Product Identification	LIN 产品代号
Load Dump	负载突降
Logical Node	逻辑节点
LIN (Local Interconnect Network)	本地互连网
Local Wakeup	本地唤醒
Master Node	主机节点
Message	消息
MRF (Master Request Frame)	主机请求帧
Master Task	主机任务
MSB (Most Significant Bit)	最高有效位
Node	节点
NAD (Node Address for Diagnose)	诊断地址
NCF (Node Capability File)	节点性能文件
NVRAM (Non-volatile Random Accessible Memory)	非易失性随机存储器
Off-the-shelf Node	现成节点
OBD (On-board Diagnose)	车载自动诊断
Operational	运行状态
Packing	拆分
PCI(Protocol Control Information)	协议控制信息
PDU (Packet Data Unit)	分组数据单元
Physical node	物理节点
Product ID	产品 ID
PID (Protected identifier)	受保护 ID
Protocol Controller	协议控制器
Publisher	发布节点
Recessive	隐性电平

Reserved Frame	保留帧
Response	应答
Response Space	应答间隔
RSID(Response Service ID)	应答服务 ID
Schedule	进度表
Service	服务
SID (Service ID)	服务代号
Signal	信号
Signal Carrying Frame	信号携带帧
SF (Single Frame)	单帧
Slave Node	从机节点
SRF (Slave Response Frame)	从机应答帧
Slave Task	从机任务
Sleep	休眠
Slew-rate	压摆率
Sporadic Frame	偶发帧
Start Bit	起始位
State Machine	状态机
Status Word	状态报告
Stop Bit	停止位
Subscriber	收听节点
Supplier ID	厂商 ID
Sync Byte Field	同步段
Tick	节拍
Time Base	时基
TVS (Transient Voltage Suppressor)	瞬态电压抑制器件
Transport Layer	传输层
Unconditional Frame	无条件帧
Unconditional frames associated with the event triggered frame	事件触发帧关联的无条件帧
Unconditional frames associated with the sporadic frame	偶发帧关联的无条件帧
Unpacking	重组
UDS (Unified Diagnostic Services)	车辆统一诊断服务
UART/SCI (Universal Asynchronous Receiver-Transmitter / Serial Communication Interface)	通用异步收发器/串行通信接口

Variant ID	可变 ID
Wakeup	唤醒
Work Flow	工作流

公司主页和咨询窗口

有关本入门书的技术方面的咨询请发邮件到下面的邮箱。

瑞萨科技公司主页　　http://cn.renesas.com

亚洲地区技术支持中心　E-Mail：support.asia@renesas.com

修订记录

修订记录	LIN 入门书

Rev.	发行日	修订内容	
		页码	要点
1.00	2010.10.25	—	初版发行

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.
   - "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.
   - "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.
   - "Specific": Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

RENESAS

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-585-100, Fax: +44-1628-585-900

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China
Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-9318, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
7F, No. 363 Fu Shing North Road Taipei, Taiwan, R.O.C.
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
1 harbourFront Avenue, #06-10, keppel Bay Tower, Singapore 098632
Tel: +65-6213-0200, Fax: +65-6278-8001

Renesas Electronics Malaysia Sdn.Bhd.
Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics Korea Co., Ltd.
11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141