



中华人民共和国国家标准

GB/T 19897.2—2005/IEC 62056-31:1999

自动抄表系统低层通信协议 第2部分： 基于双绞线载波信号的局域网使用

Automatic meter reading system lower layer communication protocol—
Part 2: Use of area networks on twisted pair with carrier signalling

(IEC 62056-31:1999, Electricity metering—

Data exchange for meter reading, tariff and load control—

Part 31: Use of local area networks on twisted pair with carrier signalling, IDT)

2005-09-09 发布

2006-04-01 实施

中华人民共和国国家质量监督检验检疫总局
中国国家标准化管理委员会 发布

目 次

前言	I
1 总论	1
1.1 范围	1
1.2 规范性引用文件	1
2 概述	1
2.1 基本术语	1
2.2 层和协议	1
2.3 特征语言	2
2.4 不带 DLMS 的本地数据交换的通信服务	2
2.5 带有 DLMS 功能的本地总线数据交换的通信服务	7
2.6 系统管理	8
3 不带 DLMS 的本地总线数据交换	8
3.1 物理层	8
3.2 数据链路层	22
3.3 应用层	29
4 带有 DLMS 的本地总线数据交换	33
4.1 物理层	33
4.2 数据链路层	33
4.3 应用层	43
5 本地总线数据交换——硬件	43
5.1 概述	43
5.2 通用技术条件	44
5.3 总线参数特征	47
5.4 磁插头	48
5.5 (50 kHz 信号的)主站发送器的功能特点	51
5.6 (50 kHz 信号的)主站接收器的功能特点	51
5.7 (50 kHz 信号的)从站发送器的功能特点	52
5.8 (50 kHz 信号的)从站接收器的功能特点	52
附录 A(规范性附录) 规范语言	54
附录 B(规范性附录) 时序类型和特征	56
附录 C(规范性附录) 致命错误清单	58
附录 D(规范性附录) 帧中命令码的编码	59
附录 E(规范性附录) CRC 原理	60
附录 F(规范性附录) 用于遗漏站点应答的随机整数的产生	61
附录 G(规范性附录) 用于电文辨认过程的随机数的产生(不带 DLMS 的结构)	62
附录 H(规范性附录) 系统管理的实现	63
附录 I(资料性附录) 交换的有关信息	64

前 言

GB/T 19897《自动抄表系统低层通信协议》分为 4 个部分：

- 第 1 部分：直接本地数据交换；
- 第 2 部分：基于双绞线载波信号的局域网使用；
- 第 3 部分：面向连接的异步数据交换的物理层服务进程；
- 第 4 部分：基于 HDLC 协议的数据链路层。

本部分为 GB/T 19897 的第 2 部分。

本部分等同采用 IEC 62056-31:1999。

《自动抄表系统》国家标准的预计结构及其对应的国际标准如下：

- a) 自动抄表系统 总则
- b) 自动抄表系统 抄表系统
 - 第 1 部分：低压电力线载波抄表系统
 - 第 2 部分：无线通信抄表系统
 - 第 3 部分：基于 IP 网络的抄表系统
- c) 自动抄表系统 应用层数据交换协议
 - 第 1 部分：对象标识系统
 - 第 2 部分：接口类
 - 第 3 部分：COSEM 应用层
- d) 自动抄表系统 低层通信协议
 - 第 1 部分：直接本地数据交换
 - 第 2 部分：基于双绞线载波信号的局域网使用
 - 第 3 部分：面向连接的异步数据交换的物理层服务进程
 - 第 4 部分：基于 HDLC 协议的数据链路层

本部分的附录 A、附录 B、附录 C、附录 D、附录 E、附录 F、附录 G 和附录 H 为规范性附录，附录 I 为资料性附录。

本部分由中国机械工业联合会提出。

本部分由全国电仪器仪表标准化技术委员会归口。

本部分起草单位：天津新巨升电子有限公司、山东电力研究院、河南思达高科技股份有限公司、华立集团、湖南威胜电子有限公司、深圳国电特瑞智能设备有限公司、上海金陵仪表有限公司、华北电力研究院、哈尔滨电仪器仪表研究所。

本部分主要起草人：王延波、徐民、吴建华、谭志强、商新民、左平、胡亚军、周新民、冯玉贵。

自动抄表系统低层通信协议 第2部分： 基于双绞线载波信号的局域网使用

1 总论

1.1 范围

GB/T 19897 的本部分阐述了用于对有源或无源的站点进行本地总线数据交换的两种新的结构。其中,对于无源站点来说,将由总线来提供用于数据交换的电源。

第一种结构用来完成于远程传输服务的基本协议 IEC 61142;第二种结构则用来解决如何使用同一物理介质和同一物理层进行 DLMS 服务操作。

这种完全的兼容性保证在同一总线的设备使用 IEC 61142 和本部分的可能性。

1.2 规范性引用文件

下列文件中的条款通过 GB/T 19897 的本部分的引用而成为本部分的条款。凡是注日期的引用文件,其随后所有的修改单(不包括勘误的内容)或修订版均不适用于本部分,然而,鼓励根据本部分达成协议的各方研究是否可使用这些文件的最新版本。凡是不注日期的引用文件,其最新版本适用于本部分。

ISO/IEC 8482:1993 信息技术 系统间远程通信和信息交换 双绞线多点互连

IEC 62056-51:1998 电气测量 抄表、费率及负荷控制的数据交换 第51部分:应用层协议

IEC 61334-4-41:1996 使用配电线载波系统的配电自动化 第4部分:数据通信协议 第41节应用协议 配电线消息描述

EIA 485 用于平衡数字多点系统的发送器和接收器的电气特性的标准

2 概述

2.1 基本术语

所有的通信请求在主站和从站间进行。主站是发起通信的一方,而从站是远端与其通信的一方。这种主从关系在一次通信的整个过程中将保持有效。

每次通信过程均可分解成若干传输事件,每个传输事件都是一次从发送端到接收端的传输。在一系列的传输事件中,主站和从站轮流担当发送和接收的任务。

对于 DLMS 服务的本地总线数据交换结构,客户机和服务器的概念有着和 DLMS 模式(参考 IEC 61334-4-41)相同的含义。服务器(从站)担当着 VDE(参考 IEC 61334-4-41)的角色,完成特殊的服务请求。客户机(主站)通过一个或多个服务请求,使用服务器来完成指定的功能。

2.2 层和协议

本地总线数据交换结构使用了解析后的三个网络层:物理层、数据链路层和应用层。对这两种本地总线数据交换结构,不管有没有 DLMS 服务,物理层都是相同的。允许各种站点安装在同一个总线上。

数据链路层和应用层协议定义见表1。

表1 结构

不带 DLMS 的结构	层	协议
	应用层	Application-62056-31
	数据链路层	Link-62056-31

表 1 (续)

	层	协议
带 DLMS 的结构	应用层	DLMS+
		Application+
	链路层	Link E/D

应用层的传输协议“Transport+”和应用子层协议“Application+”在 IEC 62056-51 中阐述。应用层的 DLMS 子层的“DLMS+”协议在 IEC 61334-4-41 中阐述。

2.3 特征语言

在此部分中,每层的协议都用状态转换表来描述。构造这些表的语法是由一种特征语言来定义的,见附录 A。

如果在文本描述和总线状态转换表的解释上有差异,以转换表内容为准。

2.4 不带 DLMS 的本地数据交换的通信服务

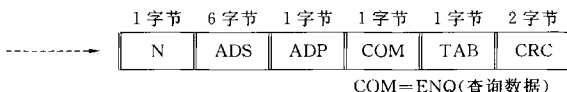
可用的服务有:

- 远程数据读取;
- 远程数据编程;
- 点对点远程传输,即一种简单化的远程编程服务;
- 广播式的远程传输;
- 总线初始化;
- 遗漏站点呼叫。

2.4.1 远程读取交换

ENQ 交换包括一组连续的两帧。

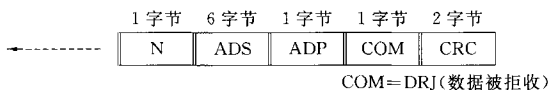
远程数据读取帧中包含有数据的类型,用 TAB 域加以选择:



肯定应答帧如下,在 DATA 域中包含有所选的数据:



否定应答帧如下(TAB 标识未知);



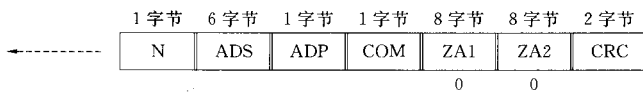
2.4.2 远程编程交换

REC 交换是由排列为两个序列的四帧组成。由于其中的一个序列用于验证目的,所以从应用角度来看,就像是只有一个序列的两帧一样。

远程编程帧中包含了数据类型域 TAB 和数据域 DATA:

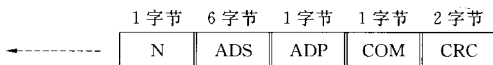


验证通过,肯定应答帧如下:



COM=EOS(对话结束)

验证通过,但远程编程数据无效时否定应答帧如下:



COM=DRJ(数据被拒收)

验证是通过交换随机数并使用每个从站专用的密钥对其进行加密的方法来实现的。随机数被定义为 8 字节长,使用一个主站和从站都共用的 8 字节密钥 K_i ,采用 DES 算法对其进行加解密运算。

首先,由主站产生一个随机数 $NA1$,将其送入远程编程帧的 ZA1 域中,而远程编程帧中的 ZA2 域被置为 0。

当远程编程帧到达从站后,ZA1 域中的数据将用密钥 K_i 对其进行 DES 运算,得出加密的随机数 $NA1K$,然后产生验证用的内部代码序列,它由两个帧组成。

第一帧(由从站到主站)中,在 ZA1 域中包含了随机数 $NA1K$,同时,在 ZA2 域中还包含了一个由从站产生的随机数 $NA2$ 。

主站接收到远程编程帧后,将 ZA1 域中的数据和其自身随机数用密钥 K_i 对 $NA1$ 进行 DES 运算的结果 $NA1'$ 进行比较,如果 $NA1'=NA1$,主站则认为被叫的从站通过了验证,否则,认为被叫的从站没有通过验证,本次通信对话失败。

主站通过了对从站的验证后,先用密钥 K_i 对随机数 $NA2$ 进行 DES 运算,得出结果 $NA2K'$ 并通过 ZA2 域将结果传出去,其中,ZA1 域被置为 0。

从站接收到应答的帧后,将 ZA2 域中的数据和其自身用密钥 K_i 对 $NA2$ 进行 DES 运算的结果 $NA2'$ 进行比较,如果 $NA2'=NA2$,从站则认为主站通过了验证,否则,认为主站没有通过验证并发送一个否定应答帧。

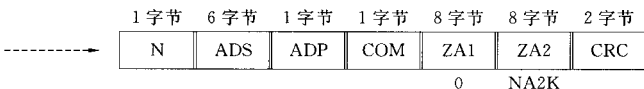
内部验证的交换如下所示:

内部验证帧中,ZA1 包含了加密运算的结果 $NA1K$,ZA2 中包含了随机数 $NA2$:



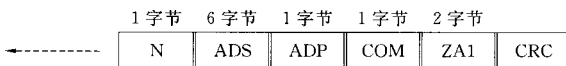
COM=ECH(回应)

如果从站是被认为通过了认证的,正确的响应在 ZA2 包含了加密运算的结果 $NA2K$:



COM=AUT(认证)

当主站没有通过验证时,将产生一个验证拒收帧,而不是正常的 EOS 和 DRJ 帧:



COM=ARJ(验证拒收)

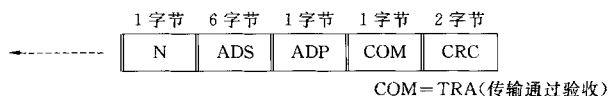
2.4.3 点对点远程传输交换

TRF 交换包括一组两帧。从应用的角度看,就像是不带验证的一组远地可编程交换。

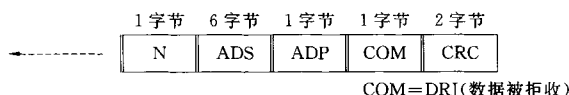
点对点远程传输帧中包含了数据类型域和数据域:



肯定应答帧如下:



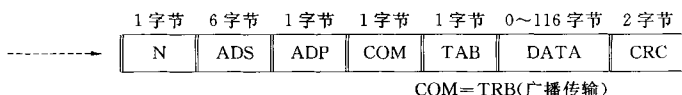
异常应答帧如下,远程传输数据无效:



2.4.4 广播式远程传输帧

TRB 帧不产生任何应答帧。从应用的角度看,就像是点对点的远程传输,但由于它是广播式的,因此没有应答。

广播式远程传输帧中包含了数据类型域和数据域:

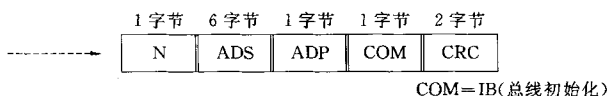


从站地址域(此地址将确定要接收数据的从站)应该为一个广播地址。

2.4.5 总线初始化帧

IB 帧不产生任何应答帧。从应用的角度看,就像是一个广播式远程传输,但由于它的目的仅仅是对所有用 ADP 地址编程的从站置一个特定标志(遗漏站点标志)“真”,因此没有任何数据。

总线初始化帧:



从站地址域(此地址将确定要接收数据的从站)应该为一个广播地址。

收到总线初始化帧之后,任何从站,只要是能收到一次正确的、包含已知 TAB 标志的 ENQ 帧,就不再认为是“遗漏站点”了。

2.4.6 遗漏站点呼叫交换

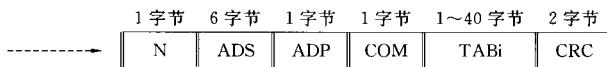
ASO 交换包含一组两帧。在一系列的远程数据读取操作的最后,主站可以寻找那些遗漏站点标志置为真的所有站点(每 100 个站点中最多允许 5 个)。

由于一次正确的远程读取交换会将相应站点的遗漏站点标志置为“假”,因此,ASO 交换一般是发生在一次总线初始化帧之后,当完成一系列(1 个或多个远程读取交换)的远程读取操作之后。

主站管控着一些间隙,当发现有帧冲突时,必须重试进行 ASO 交换。尽管每次接收了正确的从站应答,主站还是要通过对此从站进行一次正确的远程数据读取交换,将该站从遗漏站点的清单中删除掉。

为了保证选择约束(在 2.4.9 中有描述),那些无源站点将在第一个 ASO 交换的第一个时隙中应答,因此,只有遗漏站点会被选择,而且以后的 ASO 交换可以使用该通用原则。

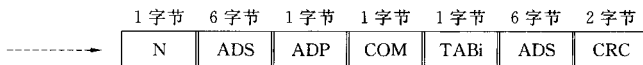
遗漏站点呼叫帧在 TABi(1 到 40 个 TAB 标志)域中包含有选择条件:



COM=ASO(遗漏站点呼叫)

从站地址(此地址将确定要接收数据的从站)应该为一个广播地址。

应答帧如下,它包含有第一个被识别出的 TAB 以及从站的地址 ADS。



COM=RSO(遗漏站点应答)

2.4.7 帧中的域

- N 帧中包含 N 在内的字节总数
- ADS 从站的绝对物理地址,48 位长。只有一个广播地址,即通用广播 ADG,以十六进制的“000000000000”[1]来表示。
- ADP 主站的物理地址,8 位长。“00H”保留,用于表示通用主站 APG[2]的物理地址。凡是物理地址为 APG 的主站的请求,从站在应答时须回传第一个曾对其进行编程的主物理地址。
- COM 根据交换类别和帧方向(见附录 D)而定的命令码。
- ZA1,ZA2 在远程编程交换中用于验证操作的码域。
- TAB 各种命令(ENQ、DAT、REC、TRF、TRB 域 RSO)中所选的数据的类型。“00h”保留,用于系统管理;“FFh”保留,用于报警管理。
- DATA 用户应用中的信息包,根据命令的不同,可以为空。
- CRC 循环冗余校验域,为 16 位长的循环冗余校验码(见附录 E)。

帧中的各个域按从前至后的顺序(从 N 到 CRC 的顺序)发送。当一个域中包含多个字节时,从最低位到最高位的顺序依次发送域中的字节,但 DATA 域的数据被当作 8 位字符串从前到后依次发送。

2.4.8 远程供电原理

数据交换的一般原理是对无源站点而言的。而所谓远程驱动,只是针对主站与一个或多个从站之间的通信供电而言。

为了开始一次通信对话,主站应发出一种“唤醒呼叫”的信号给总线上每个从站中的通信系统。这个呼叫是一种连续的载波信号,根据不同的远程驱动机制持续一小段特定的时间。

- 唤醒无源站点的“唤醒呼叫”的信号持续时间为 AGT。
- 唤醒有源站点的“唤醒呼叫”的信号持续时间为 AGN。

备注:从站可以被置为报警模式。在这种模式下,从站的通信部分始终处于远程供电状态,以便能随时传输报警信息给主站(见 2.4.11)。

那么,不管从站的驱动方式(有源或无源)如何,在以下几种情况下都要求主站方面发出“唤醒呼叫”信号:

- 在第一次 ENQ 或 TRF 交换之前;
- 对同一从站进行第 6 次连续且成功的 ENQ 或 TRF 交换之前;
- 对于从刚才所选的进行 ENQ 或 TRF 交换的从站切换到一个新的从站,要进行 ENQ 和 TRF 交换之前;
- 在任何 REC 交换之前;

- 在任何 TRB 帧之前；
- 在任何 IB 帧之前；
- 在任何 ASO 交换之前。

对于无源工作方式而言，意味着在不必要的情况下，主站能够避免唤醒所有的从站，以便节省电能。

使用一种特殊的调制解调器，主站可以保证同时完成供电驱动和调制解调的功能。通信的时间和无源从站的数量也是可选的，以便节省主站的电池能量。

还有一种可能，就是主站只是集中在调制和解调的功能上，这时，需要一个辅助的站点给总线持续供电。

一般来说，一个从站对于它的绝对物理地址 ADS 仅有一种逻辑上的应用。这时，从站可以有源的，也可以是无源的。

一个多用途的从站(包含与若干 ADS 相对应的若干应用)必须是无源的。这种特点在第 5 章有更多的描述。

2.4.9 无源站点的预选交换

为了降低总线的电能消耗，主站使用一次预选交换来选择一个无源从站。

预选交换发生在一个对总线上的所有无源站点发出的 AGT“唤醒呼叫”信号之后。为了限制总线的电能消耗，此时主站发出的第一个帧要尽可能短，并且被选址的从站必须在 TOPRE 触发唤醒之前应答。若没能及时收到帧，从站的调制解调器将返回低功耗模式。

在预选交换的过程中，所有的无源站点都将消耗能量。总线的电压和储能电容器的电能将不断下降，直到那些没被选址的从站都返回到低功耗工作状态。此后将对储能电容器连续充电，总线的电压逐渐升高。

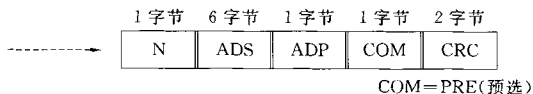
在第一次预选交换之前，主站的调制解调器必须储存好足够的能量，这一步由 TICB 唤醒机制控制的等待时间来保证。在预选交换之后，储能电容的能量已空，主站就必须等到总线的电压升高之后再进行下一次预选交换。

由于预选交换的帧长度不能超过 18 字节，它可以是：

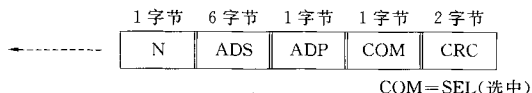
- 一个 ENQ 帧；
- 一个 TRB 或 TRF 帧，只有当数据域的长度不超过 6 字节时；
- 一个 IB 帧；
- 一个 ASO 帧，只有当 TABi 域不超过 7 个字节时。

由于第一个 REC 和 TRF 变换帧可能太长，所以为预选提供了一种附加的功能。这个完全透明的 PRE 交换包含一组两个帧：

无源站点预选交换帧



应答帧



为了减小主站的能量消耗，预选交换没有重试操作。如果一个被寻址的无源站点应答不正确，该站就不被选中，主站应当发出一个新的 AGT“唤醒呼叫”信号。

2.4.10 预选交换之后的通信交换

预选交换之后，无源站点的调制解调器可一直保持唤醒状态，以便继续通信，且对延时没有苛刻的

要求,因为连接设备的数量是有限的。主站向选中的站点供电,并且对未选中的站点的储能电容进行充电。

由于远程驱动的机制不同,正常的通信对话结束的时机也不同:

- 对于有源从站来说,在没有中介 AGN“唤醒呼叫”信号请求、通信对话中出现一小段停止之后结束。这段时间由唤醒 TOL 来检查。

- 对于无源从站来说,在通信对话中出现一段稍长时间的停止之后结束。这段时间由唤醒 TOAG 来检查。

注意,对于无源站点来说,只要唤醒 TOAG 的时间还没有到,一个中介的 AGN“唤醒呼叫”信号就足以使现有的通信对话继续下去。

2.4.11 报警功能

集成在单一功能和多功能从站(见 5.2.3)之中的一个装置能够向主站发送报警信号,提供如下描述的接口的功能。

一个报警信号应在最多 10s 内从从站中获得。

接口配置可以编程,每个设备可以选择报警模式的两种状态:启动或禁止。

当报警模式被设置为启动时,设备允许从从站的内部产生报警信号。报警功能只有当总线能够且永久供电时方能生效。

设备在 TASB 期间发送报警信号。TASB 时间足够的长,在辅助总线上产生一个强制的“0”状态,这个状态可以被接口检测到,即使在通信过程中也能够检测到。

报警机制如图 1 所示。



图 1 报警机制

报警信号不是直接传向主站的。接口接收到报警信号后在 TAB 期间在总线上发一个“0”(50kHz 载波把它传出去),发送在以下的几种可能时进行:

- 总线上没有通信时。当接口在辅助总线上接收到报警信号时,接口将其传送到总线上。
- 在通信过程中和 AGN 或 AGT 同步传送。当总线上正在进行通信时,接口把接收到的报警信号现存下来,然后在以下之一情况结束后将其传送到总线上:

- TOALR 在接收到 AGN 或 AGT 之后;
- 当正常的通信会话结束后。

采用这种方式,接口可以过滤报警信号,以防总线上的通信冲突。

报警产生之后,从站被当作一个“遗漏站点”,其选择条件等于“FF”。

置为报警模式的主站当总线上没有通信进行时,在一个 AGN 或 AGT 传送之后监听总线,以便能检测到一个报警信号。当主站接收到一个报警信号之后,主站将执行遗漏站点呼叫过程,届时,在 TABi 域中的选择条件为“FF”(见 2.4.6)。

解释报警管理的时序图见 3.1.3。

2.5 带有 DLMS 功能的本地总线数据交换的通信服务

DLMS 不提供总线初始化和遗漏站点呼叫的服务,但不带 DLMS 的本地数据交换相同,支持 IB 和 ASO 交换,只是遗漏站点标志被当作一个全局变量,被应用程序接口共享使用。

DLMS 能直接预见远程数据读取和点对点远程传输,但不支持(冗余)远程数据编程。这是由于

把验证任务留给了应用层。

DLMS 管理着数据的构造,其帧格式非常简单,只须用到非标记帧。为了和不带 DLMS 的结构兼容,帧格式由以下 9 个域定义:

1 字节	6 字节	1 字节	3 字节	1 字节	2 字节	2 字节	0~117 字节	2 字节
Size	ADS	ADP	DATA+	Priority	Send	Confirm	Text	CRC
Size	帧中包含 Size 在内的字节总数。若其数值不是 11,接收方则得知其 Text 域中包含有数据。							
ADS	同不带 DLMS 的本地总线结构。							
ADP	同不带 DLMS 的本地总线结构。							
DATA+	置为“111”B。							
Priority	当前帧的传输优先级。应用层根据所需的服务设置其优先级。							
Send	上一传输帧的编号。							
Confirm	上一正确接收帧的编号。							
Text	高层的 DSDU(数据链路服务数据单元)。帧中可以不包含数据。若在发送帧时可以从应用层获得数据,Text 域中则包含有数据,否则便为空。这种机制为平衡双向数据传输提供了条件。为了避免 DATA + 帧和不带 DLMS 结构的帧发生混淆,DATA +、Priority、Send 和 Confirm 域组成一个特殊的命令码 COM,其值和先前保留的 COM 值(见附录 D)应该不同。							
CRC	同不带 DLMS 的本地总线结构。							

帧中的各个域按从前至后的顺序(从 N 到 CRC 的顺序)发送。当一个域中包含多个字节时,从最低位到最高位依次发送域中的字节,但 TEXT 域的数据被当作 8 位字符串从前到后依次发送。

2.6 系统管理

系统管理的目的是进行包括对总线上的从站识别的登记工作,它使用“发现”(Discover)服务来实现该目的。

登记工作包括一系列的从主站内部的初始化器发出的“发现”请求。每一个“发现”服务用于通知剩余的新站点将可以在下一个时隙获得一次应答的机会。

每个“发现”请求都包含一个特定的范围在 0~100 之间的应答概率码(整数),它以百分比的形式表示一个新站点应答的概率。当应答概率码为 100 时,总线上的所有站点都必须应答。

接收到“发现”请求后,所有的从站将测试它的“已经发现”标志的值。如果“已经发现”标志的值为真,那么就放弃这个请求,否则,从站产生一个 1~100 之间的随机数。若此随机数小于或等于应答概率码,这个新站点则发出一个“发现”应答,并随后将其“已经发现”标志置为真。

“已经发现”标志总是由 IB 帧来复位的。

为了确保最大可能的兼容性(对于包含或不包含 DLMS 的站点),在附录 H 中提出了实现系统管理的方案。

3 不带 DLMS 的本地总线数据交换

3.1 物理层

3.1.1 Physical-62056-31 协议

不带 DLMS 的本地总线数据交换结构的物理层的 Physical-62056-31 协议是不对称的结构。所以,主站的状态机制和从站的不一樣。

Physical-62056-31 协议既支持无源的也支持有源的从站。在概述中已经阐述过,远方的站点可以被 AGN 或 AGT 信号唤醒,通信对话在 TOL 或 TOAG 到期时结束。

唤醒呼叫信号之后,将在总线上以 1200bit/s、半双工的方式进行异步通信对话。

3.1.2 物理参数

一帧的最大长度,MaxIndex,为 128 字节。

执行一个“遗漏站点呼叫”的 RSO 时隙的最大值,MaxRSO,为 3。

AGN 唤醒呼叫信号的 AGN 时间在 50 ms~150 ms 的范围内,AGT 唤醒呼叫信号的 AGT 时间在 200 ms~300 ms 的范围内。

时序和特征在附录 B 中阐述。

主站的值定义如表 2。

表 2 主站的时序

	最小值 ms	典型值 ms	最大值 ms	类型	定 义
TAIO	—	—	120	T _{SLI}	接收帧第 1 个字节的最长等待时间
TAB	—	100	—	T _C	总线上报警信号的长度
TAGN	—	100	—	T _{PDF}	一个 AGN“唤醒呼叫”信号的长度
TAO	—	—	40	T _C	指示帧结束的接收字节的最长等待时间
TARSO	—	500	—	T _C	RSO 时隙的长度
TASB	—	1 200	—	T _C	从报警信号开始的等待时间
TEMPO	—	40	—	T _C	在唤醒信号或帧传输后的安全延时时间
TOE	—	—	1 100	T _L	排除硬件故障的安全延时时间
TOL	—	—	100	T _{SL2}	等待从上层来的请求的最长时间
无源站点的技术参数(供电状态下)					
TAGT	—	250	—	T _{PDF}	AGT 唤醒呼叫的时间
TICB	8 000	—	—	T _s	总线初始充电时间
TOAG	—	—	3 000	T _{PDF}	一个选中的无源站点识别出一个 AGN“唤醒呼叫”信号的最长等待延时

从站的值定义如表 3。

表 3 从站的时序

	最小值 ms	典型值 ms	最大值 ms	类 型	定 义
TAIO	30 ¹⁾	—	160	T _{SLI}	接收第 1 个字节的 longest 等待时间
TAB	—	100	—	T _C	总线上报警信号的长度
TAGN	50	100	150	T _{PDF}	一个 AGN“唤醒呼叫”信号的长度
TAO	—	—	40	T _C	指示帧结束的接收字节的最长等待时间
TARSO	—	—	500	T _C	RSO 时间片的长度
TOALR	20	—	—	T _C	AGN 或 AGT 接收之后等待发送一个 AGN 的时间
TOE	—	—	1 100	T _L	消除硬件的安全延时时间
TOL	—	—	100	T _{SL2}	等待从上层来的请求的最小时间

1) 在“唤醒呼叫”之后,必须最少有 30 ms 的延时期间。

表 3 (续)

	最小值 ms	典型值 ms	最大值 ms	类 型	定 义
无源站点技术参数(供电状态下)					
TAGT	200	250	300	T_{PDF}	AGT“唤醒呼叫”信号的时间
TASB	—	1 200	—	T_L	辅助总线报警信号的持续时间
TICB	8 000	—	—	T_c	总线初始充电时间
TOAG	—	—	3 000	T_{PFD}	一个选中的无源站点识别出一个 AGN“唤醒呼叫”信号的最长等待延时
TOAGN	—	—	300	T_c	为识别与有源站点结束通话所需的最大待机延时时间
TOAPPEL	—	—	180	T_{PFD}	等待预选帧的第 1 个字节的最长等待时间
TOBAVARD	—	—	260	T_{PFD}	确保预选帧长度的安全延时时间
TOPRE	—	—	130	T_{PFD}	等待预选应答的最长等待时间
TOSEUIL	—	150	—	T_c	唤醒无源站点的一个“唤醒呼叫”信号的持续时间
TVASB	40	—	—	T_L	在辅助总线上的报警信号的最小持续时间

3.1.3 时序图

图 2、图 3、图 4 用来说明无源从站所用协议的不同类型的对话。

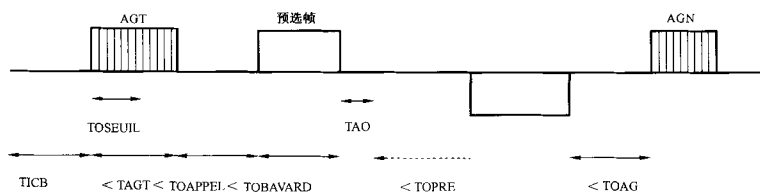


图 2 连续操作中的交换

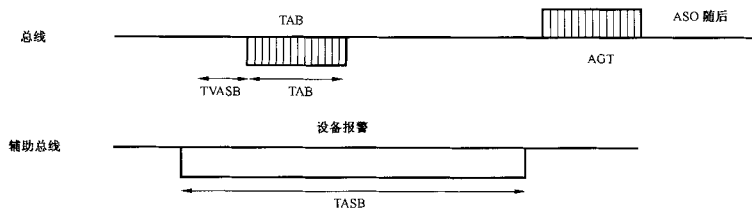


图 3 没有通信时的报警事件

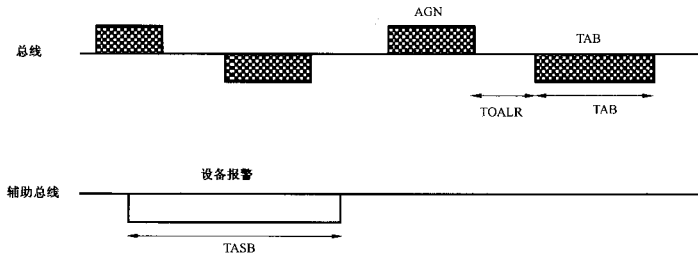


图 4 通信中的报警事件

3.1.4 物理服务和 service 基元

Physical-62056-31 协议的用户可以使用表 4 给出的服务和 service 基元。

表 4 物理服务和 service 基元

服 务	服 务 基 元
Phy_DATA	Phy_DATA.req(Frame)
	Phy_DATA.ind(Frame)
Phy_UNACK	Phy_UNACK.req(Frame)
Phy_APPG	Phy_APPG.req(TypeAG)
	Phy_APPG.ind()
Phy_ASO	Phy_ASO.req(Frame)
	Phy_ASO.ind(Frame)
Phy_RSO	Phy_RSO.req(Frame, Window)
Phy_COLL	Phy_COLL.ind()
Phy_ALARM	Phy_ALARM.req()
	Phy_ALARM.ind()
Phy_ABORT	Phy_ABORT.req()
	Phy_ABORT.ind(ErrorNb)

赋予每个基元的任务是：

- Phy_DATA.req(Frame)使数据链路层能够请求物理层发送一个 Frame 帧；
- Phy_DATA.ind(Frame)使物理层能够通知数据链路层一个 Frame 帧准备好；
- Phy_UNACK.req(Frame) 使数据链路层能够请求物理层不须等待确认应答就发送一个 Frame 帧；
- Phy_APPG.req(TypeAG) 使数据链路层能够请求物理层发送一个“唤醒呼叫”信号，这个信号的持续时间 TypeAG 可能是 AGN 或 AGT；
- Phy_APPG.ind()使物理层能够通知数据链路层一个“唤醒呼叫”信号发送结束；
- Phy_ASO.req(Frame) 使数据链路层能够请求物理层发送一个“遗漏站点呼叫”帧；
- Phy_ASO.ind(Frame) 使物理层能够通知数据链路层在遗漏站点的某个时隙中已经收到 Frame 帧；
- Phy_RSO.req(Frame, Window) 使数据链路层能够请求物理层在时隙 Window 中发送一个遗漏站点呼叫帧 Frame；
- Phy_COLL.ind()使物理层能够通知数据链路层在遗漏站点的某个时隙中已经检测到一个冲突；
- Phy_ALARM.req()使数据链路层能够请求物理层发送一个报警信号；

- Phy_ALARM.ind()使物理层能够通知数据链路层接收到一个报警信号；
- Phy_ABORT.req()使数据链路层能够请求物理层结束其激活状态；
- Phy_ABORT.ind(ErrorNb)使物理层能够通知数据链路层发生了一个出错标识号为 ErrorNb 的致命错误。

3.1.5 状态转换

表 5 Physical-62056-31 状态转换:主站

初始状态	触发条件	动作命令集	终结状态
Initial	\$ true ()	MaxRSO=3 MaxIndex=128 Collision=FALSE SessionAGT=FALSE wait_time (TICB)	Stopped
Stopped	Phy_APPG.req(AG)&AG=AGN	stop_timer(TOAG) FlagAbort=FLASE TypeAG=AGN send_AG(TypeAG)	W. AG
Stopped	Phy_APPG.req(AG)&AG=AGT	SessionAGT=TRUE FlagAbort=FLASE TypeAG=AGT send_AG(TypeAG)	W. AG
Stopped	time_out(TOAG)	Phy_ABORT.ind(EP-2) SessionAGT=FALSE	Stopped
Stopped	Phy_ABRT.req()	\$ none()	Stopped
Stopped	data-carrier_on	init_timer(TAB) init_timer(TASB)	W. ETABS
W. ETABS	data-carrier_off	stop_timer(TASB) stop_timer(TAB)	Stopped
W. ETABS	time_out(TAB)	Phy_ABORT.ind(EP-3) Phy_ALARM.ind()	W. TASB
W. AG	AG_sent_event	Phy_APPG.ind() init_timer(TEMPO)	W. TAG
W. AG	Phy_ABORT.req()	FlagAbort=TRUE	W. AG
W. TAB	data-carrier_on	Carrier=FALSE init_timer(TAB) init_timer(TASB)	W. TAB
W. TAB	data-carrier_off	Carrier=TURE stop_timer(TAB) stop_timer(TASB)	W. TAB
W. TAB	time_out(TEMPO)& not(FlagAbort)¬(Carrier)	init_timer(TOL)	M. Send
W. TAB	time_out (TEMPO) & FlagAbort & not(Carrier)	wait_time(TOL)	T. Session
W. TAB	time_out(TEMPO)&Carrier	init_timer(TOL)	W. ETAB

表 5 (续)

初始状态	触发条件	动作命令集	终结状态
W. TAB	Phy_ABORT. req()	FlagAbort=TRUE	W. TAB
W. ETAB	time_out(TAB)	Phy_ABORT. ind(EP-3) Phy_ALARM. ind() Stop_timer(TOL)	W. TASB
W. ETAB	data_carrier_off & not(FlagAbort)	Stop_timer(TAB) Stop_timer(TASB)	M. Send
W. ETAB	data_carrier_off & FlagAbort	Stop_timer(TAB) Stop_timer(TASB)	W. TOL
W. ETAB	Phy_ABORT. req()	FlagAbort=TRUE	W. ETAB
W. TASB	time_out(TASB)	\$ none()	Stopped
W. TOL	time_out(TOL)	\$ none()	T. Session
M. Send	Phy_DATA. req(Frame)	Service=NORMAL	SendFirst
M. Send	Phy_UNACK. req(Frame)	Service=UNACKNOWLEDGED	SendFirst
M. Send	Phy_ASO. req(Frame)	Service=ASO	SendFirst
M. Send	Phy_ABORT. req()	\$ none()	M. Send
M. Send	time_out(TOL)	\$ none()	T. Session
T. Session	SessionAGT=TRUE	init_timer(TOAG) Phy_ABORT. ind(EP-1) wait_time(TEMPO)	stopped
T. Session	SessionAGT=FALSE	Phy_ABORT. ind(EP-1) wait_time(TEMPO)	Stopped
SendFirst	\$ true()	stop_timer(TOL) Size=size(frame) Index=1 send_octet(Frame,index) Size=Size-1 init_timer(TOE)	Sending
Sending	octet_sent_event & Size>0	Index=index+1 send_octet(Frame,Index) Size=Size-1	Sending
Sending	octet_sent_event & Size=0	stop_timer(TOE) wait_time(TAO) Index=1 Frame=" "	Answer
Sending	Phy_ABORT. rep()	Stop_timer(TOE) wait_time(TAO) init_timer(TA10) FlagAbort=TRUE	M. Rec

表 5 (续)

初始状态	触发条件	动作命令集	终结状态
Sending	time_out(TOE)	Phy_ABORT.ind(EP-3F) wait_time(TAO) init_timer(TAIO) FlagAbort=TRUE	M. Rec
Answer	Service=NORMAL1 Service=UNACKNOWLEDGED	init_timer(TAIO)	M. Rec
Answer	Service=ASO	WinRSO=1 init_timer(TARSO) init_timer(TAIO)	M. Rec
M. Rec	octet_received_event	stop_timer(TAIO) Index=index+1 read_data(RecB) concat(Frame,RecB) init_timer(TAO)	Receiving
M. Rec	collision_detected_event	stop_timer(TAIO) Collision=TRUE init_timer(TAO)	Receiving
M. Rec	time_out(TAIO)	\$ none()	Received
M. Rec	Phy_ABORT.req()	FlagAbort=TRUE	M. Rec
Receiving	octet_received_event& Index<=MaxIndex	stop_timer(TAO) Index=Index+1 read_data(RecB) concat(Frame,RecB) init_timer(TAO)	Receiving
Receiving	octet_received_event& Index>MaxIndex	Phy_ABORT.ind(EP-4F) wait_time(TAO) FlagAbort=TRUE	Received
Receiving	Collision_detected_event	Stop_timer(TAO) Collision=TRUE Init_timer(TAO)	Receiving
Receiving	time_out(TAO)	\$ none()	Received
Receiving	time_out(TARSO)	Phy_ABORT.ind(EP-5F) Wait_time(TAO) FlagAbort=TRUE	Received
Receiving	Phy_ABORT.req()	FlagAbort=TRUE	receiving
Received	Service=NORMAL & not(Flagabort)	Phy_DATA.ind(Frame) Init_timer(TOL)	M. Send
Received	(Service=NORMAL & Service=UNACKNOWLEDGED)	Wait_time(TOL)	T. Session
Received	Service=ASO & Collision&. not(Flagabort)	Phy_COLL.ind() Collision=FALSE	T. RSO

表 5 (续)

初始状态	触发条件	动作命令集	终结状态
<i>Received</i>	Service=ASO & not(Collision) & not(Flagabort)	Phy_AS0.ind(Frame)	<i>T. RSO</i>
<i>Received</i>	Service=ASO & Flagabort	\$ none()	<i>T. RSO</i>
<i>T. RSO</i>	(TypeAG=AGT) (WinRSO>=MaxRSO) & (TypeAG=AGN)	Stop_timer(TARSO)	<i>T. Session</i>
<i>T. RSO</i>	(WinRSO<MaxRSO) & (TypeAG=AGN)	Index=1 Frame=""	<i>W. RSO</i>
<i>W. RSO</i>	time_out(TARSO)	WinRSO=WinRSO+1 init_timer(TARSO) init_timer(TA1O)	<i>M. Rec</i>
<i>W. RSO</i>	Phy_ABORT.req()	Flagabort=TRUE	<i>W. RSO</i>

表 6 电源供电管理状态转换(只对无源从站点而言)

初始状态	触发条件	动作命令集	终结状态
<i>Initial</i>	alarm_detection()	Flagalarm=TRUE FlagSendAlarm=FALSE station_power(ON)	Stopped
<i>Initial</i>	not(alarm_detection())	Flagalarm=FALSE	Stopped
Stopped	occur(cpt_carrier_on) & Flagalarm	init_timer(TVASB)	<i>W. TVASB2</i>
Stopped	occur(data_carrier_on)	init_timer(TOSEUIL) init_timer(TAGT)	<i>W. TOSEUIL</i>
<i>W. TOSEUIL</i>	time_out(TOSEUIL) & not(Flagalarm)	station_power(ON)	<i>W. AGT</i>
<i>W. TOSEUIL</i>	occur(data_carrier_off) & not(Flagalarm)	stop_timer(TOSEUIL) stop_timer(TAGT)	Stopped
<i>W. TOSEUIL</i>	time_out(TOSEUIL) & Flagalarm	station_signal(ON) Tend=TOAG	<i>W. AGT</i>
<i>W. TOSEUIL</i>	occur(data_carrier_off) & Flagalarm	stop_timer(TOSEUIL) stop_timer(TAGT) Tend=TOAGN init_timer(Tend)	Hide
<i>W. TOSEUIL</i>	occur(cpt_carrier_on) & Flagalarm	init_timer(TVASB)	<i>W. TVASB1</i>
<i>W. AGT</i>	occur(data_carrier_off)	stop_timer(TAGT) init_timer(TOAPPEL)	<i>W. Sel</i>
<i>W. AGT</i>	time_out(TAGT) & not(Flagalarm)	station_power(OFF)	Stopped
<i>W. AGT</i>	time_out(TAGT) & Flagalarm	init_timer(Tend)	Hide
<i>W. Sel</i>	occur(octet_received_event)	stop_timer(TOAPPEL) init_timer(TOBAVARD) init_timer(TAO)	Select

表 6 (续)

初始状态	触发条件	动作命令集	终结状态
W. Sel	time_out(TOAPPEL) & not(Flagalarm)	station_power(OFF)	Stopped
W. Sel	Time_out(TOAPPEL) & Flagalarm	station_signal(OFF)	Stopped
W. Sel	occur(cpt_carrier_on) & Flagalarm & not(FlagSendalarm)	init_timer(TVASB)	W. TVASB1
Select	Occur(octer_received_event)	stop_timer(TAO) init_timer(TAO)	Select
Select	time_out(TAO)	stop_timer(TOBAVARD) init_timer(TOPRE)	W. Answer
Select	time-out(TOBAVARD) & not(Flagalarm)	stop_timer(TAO) station_power(OFF)	Stopped
Select	time_out(TOBAVARD) & Flagalarm	stop_timer(TAO) init_timer(Tend)	Hide
W. Answer	occur(octet_sent_event)	stop_timer(TOPRE) init_timer(Tend)	Hide
W. Answer	time_out(TOPRE) & not(Flagalarm)	station_power(OFF)	Stopped
W. Answer	time_out(TOPRE) & Flagalarm	init_timer(Tend)	Hide
W. Answer	occur(cpt_carrier_on) & Flagalarm & not(FlagSendalarm)	init_timer(TVASB)	W. TVASB1
Hide	occur(octet_received_event) occur(octet_sent_event) (occur(data_carrier_on) & not(FlagSendAlarm))	stop_timer(Tend) init_timer(Tend)	Hide
Hide	occur(data_carrier_on) & FlagSendAlarm	stop_timer(Tend)	W. AGend
Hide	time_out(Tend) & not(Flagalarm)	station_power(OFF)	Stopped
Hide	time_out(Tend) & Flagalarm & not(FlagSendAlarm)	station_signal(OFF)	Stopped
Hide	time_out(Tend) & Flagalarm & FlagSendAlarm	Send_AG(AGN)	W. AB
Hide	occur(cpt_carrier_on) & Flagalarm & not(FlagSendalarm)	init_timer(TVASB)	W. TVASB1
W. AGend	occur(data_carrier_off)	wait_time(TOALR) Send_AG(AGN)	W. AB

表 6 (续)

初始状态	触发条件	动作命令集	终结状态
W. TVASB1	occur(cpt_carrier_off)	stop_timer(TVASB) init_timer(Tend)	Hide
W. TVASB1	time_out(TVASB)	FlagSendAlarm=TRUE init_timer(Tend)	Hide
W. TVASB1	time_out(Tend)	\$ none()	W. TVASB2
W. TVASB2	occur(cpt_carrier_off)	Stop_timer(TVASB) Station_signal(OFF)	Stopped
W. TVASB2	occur(data_carrier_on)	\$ none()	W. TVASB1
W. TVASB2	time_out(TVASB)	Send_AG(AGN)	W. AB
W. AB	AG-sent_event	FlagSendAlarm=FALSE station_signal(OFF)	Stopped

表 7 Physical-62056-31 状态转换: 从站

初始状态	触发条件	动作命令集	终结状态
<i>Initial</i>	energized()	MaxIndex=128 FlagRSO=FALSE FirstWinRSO=FALSE	Stopped
<i>Initial</i>	not(energized())	MaxIndex=18 FlagRSO=FALSE FirstWinRSO=TRUE	Stopped
Stopped	AG_received_event	Stop_timer(TOAG) init_timer(TA1O)	M. Rec
Stopped	Phy_ALARM, req()	TypeAG=ASB Send_AG(TypeAG)	W. ASB
Stopped	time_out(TOAG)	MaxIndex=18 FirstWinRSO=TRUE	Stopped
M. Rec	octet_received_event	stop_timer(TA1O) Index=2 Frame="" Read_data(RecB) Concat(Frame, RecB) init_timer(TAO)	Receiving
M. Rec	time_out(TA1O)	Phy_ABORT(EP_1)	WTOAG
M. Rec	Phy_ABORT, req()	stop_timer(TA1O)	WTOAG
Receiving	octet_received_event & Index<=MaxIndex	Stop_timer(TAO) Index=Index+1 Read_data(RecB) Concat(Frame, RecB) init_timer(TAO)	Receiving
Receiving	octet_received_event & index>MaxIndex	Stop_timer(TAO) Phy_ABORT, ind(EP-4F)	WTOAG

表 7 (续)

初始状态	触发条件	动作命令集	终结状态
Receiving	time_out(TAO)	Phy_DATA.ind(Frame) init_timer(TOL)	M. Send
Receiving	Phy_ABORT.req()	Stop_timer(TAO)	WTOAG
M. Send	Phy_DATA.req(Frame)	Stop_timer(TOL) MaxIndex=128 Size=size(Frame) Index=1 Send_octet(Frame, Index) Size=Size-1 init_timer(TOE)	Sending
M. Send	Phy_RSO.req(Frame, Window)	Stop_timer(TOL) MaxIndex=128 Wait_window(FirstWinRSO, Window) FirstWinRSO=FALSE Size=size(Frame) Index=1 Send_octet(Frame, Index) Size=Size-1 FlagRSO=TRUE init_timer(TOE)	Sending
M. Send	Time_out(TOL)	init_timer(TAIO)	M. Rec
M. Send	Phy_ABORT.req()	Stop_timer(TOL)	WTOAG
Sending	Octet_sent_event & Size>0	Index=Index+1 Send_octet(Frame, Index) Size=Size-1	Sending
Sending	octet_sent_event & Size=0 & not(FlagRSO)	stop_timer(TOE) init_timer(TAIO)	M. Rec
Sending	octet_sent_event & Size=0 & FlagRSO	Stop_timer(TOE) Wait_time(TAO) FlagRSO=FALSE	WTOAG
Sending	Phy_ABORT.req()	Stop_timer(TOE)	WTOAG
Sending	time_out(TOE)	Phy_ABORT.ind(EP-3F)	WTOAG
W. ASB	time_out(TOAG)	MaxIndex=18 FirstWinRSO=TRUE	W. ASB
W. ASB	AG_sent_event	\$ none()	Stopped
WTOAG	not(energized)	init_timer(TOAG)	Stopped
WTOAG	Energized	\$ none()	Stopped

表 8 上述表中的状态含义

状 态	含 义
<i>Initial</i>	初始化该层的变量
Stopped	等待一个“唤醒呼叫”信号的状态
W. ETABS (Wait for end of “Alarm-Bus”)	等待一个在停止状态接收到的“总线报警”信号的结束
W. AG (Wait for end of “Wakeup Call”)	等待一个“唤醒呼叫”信号传输的结束
W. TAB (Wait “Alarm - Bus”)	在结束了“唤醒呼叫”信号之后的“安全延时期内等待一个“总线报警”信号
W. ETAB (Wait for end of “Alarm_Bus”)	在一个“唤醒呼叫”信号之后等待一个“总线报警”信号的结束
W. TASB	在接收到一个“总线报警”信号的开始等待唤醒 TSB 的触发
W. TOL	等待唤醒 TOL 的触发
M. Send (Must Send)	发送器等待发送一帧时的初始状态
<i>T. Session</i>	测试会话的类型(和无源或有源从站)
<i>SendFirst</i>	发送待传输帧的第 1 个字节
Sending	发送器每次传输 1 个字节时的状态
<i>Answer</i>	根据所请求的服务确定分支的去向
M. Rec (Must Receive)	接收器等待接收一帧的第 1 个字节时的初始化状态
Receiving	接收器每次接收 1 个字节时的循环状态
<i>Received</i>	根据请求的服务项目对接收帧进行处理
T. RSO (Text last RSO)	测试项目接收 RSO 帧的最后一个时隙的结束
W. RSO (Wait for end of an RSO time slot)	等待接收 RSO 帧的时隙的结束
W. ASB	等待“辅助总线报警”信号传输的结束
W. TOAG	如果需要的话,初始化“会话结束”TOAG 定时器
W. TOSEUIL	等待唤醒 TOSEUIL 的触发
W. AGT	等待一个 AGT“唤醒呼叫”信号
W. Sel (Wait for preSelection)	等待一个预选帧
Select	接收一个预选帧
W. Answer	等待被选站点的应答
Hide	等待选址的结束
W. Agend	等待 AG 接收的结束
W. TVASB1	在会话中,等待一个用来唤醒“辅助总线报警”信号的 TVASB 的触发
W. TVASB2	在会话结束时,等待一个用来唤醒“辅助总线报警”信号的唤醒 TVASB 的触发
W. AB	等待一个“总线报警”信号传输的结束

表 9 过程函数和事件分类(按字母排序)

过程、函数和事件	定 义
AG_received_event	从 MODEM 报告的一个 AGN“唤醒呼叫”信号已经正确接收的事件
AG_sent_event	从 MODEM 报告的一个“唤醒呼叫”信号结束的事件
alarm_detection()	检测站点报警模式状态是激活的
collision_detection()	从 MODEM 报告的检测出接收一个字节时出现帧结构错误的事件
concat(Frame, RecB)	将正在组建的帧 Frame 和字节 RecB 相关联
data_carrier_on, data_carrier_off	检测出总线上数据载波有和数据载波无
energized()	检测站点是供电的
init_timer(TOAPPEL), init_timer(TOSEUIL), init_timer(TAGT), init_timer(TOBAVARD), init_timer(TOPRE), init_timer(TOL), init_timer(TOE), init_timer(TAO), init_timer(TAIO), init_timer(TARSO), init_timer(TOAG), init_timer(TVASB) or init_timer(TAB)	设置唤醒 TOAPPEL, TOSEUIL, TAGT, TOBAVARD, TOPRE, TOL, TOE, TAO, TAIO, TARSO, TOAG, TVASB, TAB
occur(cpt_carrier_on), occur(cpt_carrier_off), occur(data_carrier_on), occur(data_carrier_off), occur(octet_received_event) or occur(octet_sent_event)	检测出辅助总线上数据载波有、数据载波无,总线上的数据载波有、数据载波无,接收到一个字节或发送出一个字节
octet_received_event	从 MODEM 报告的一个字节被接收的事件
octet_sent_event	从 MODEM 报告的一个字节被发送的事件
read_data(RecB)	通过读取接收到的 RecB 字节来处理事件 octet_received_event(按升序逐位发送)
send_AG(TypeAG)	请求 MODEM 在 TypeAG(AGN 或 AGT)期间发送一个“唤醒呼叫”信号
send_octet(Frame, Index)	在帧 Frame 中传送排序索引字节(按升序逐位传送)
size(Frame)	计算 Frame 帧的字节数
station_power(ON) or station_signal(OFF)	打开或关闭对设备的供电
station_signal(ON) or station_signal(OFF)	在辅助总线上打开或关闭对设备的信号传输

表 9 (续)

过程、函数和事件	定 义
stop_timer(TOAPPEL), stop_timer(TOSEUIL), stop_timer(TAGT), stop_timer(TOBAVARD), stop_timer(TOPRE), stop_timer(TOL), stop_timer(TOE), stop_timer(TAO), stop_timer(TAIO), stop_timer(TVASB), or stop_timer(TAB)	停止唤醒 TOAPPEL, TOSEUIL, TAGT, TOBAVARD, TOPRE, TOL, TOE, TAO, TAIO, TVASB, TAB
stop_timer(TOAG) or stop_timer(TARSO)	停止唤醒 TOAG 或 TARSO(若它们先前被设置过)
time_out(TOAPPEL), time_out(TOSEUIL), time_out(TAGT), time_out(TOBAVARD), time_out(TOPRE), time_out(TOL), time_out(TOE), time_out(TAO), time_out(TAIO), time_out(TARSO), time_out(TOAG), time_out(TVASB), or stop_timer(TAB)	触发唤醒 TOAPPEL, TOSEUIL, TAGT, TOBAVARD, TOPRE, TOL, TOE, TAO, TAIO, TARSO, TOAG, TVASB, TAB
wait_time(TAO), wait_time(TICB), wait_time(TOL) or wait_time(TOALR)	在 TAO, TICB, TOL 或 TOALR 时间内计算延时
wait_window(FirstWinRSO, Window)	计算等待时间如下: 当 FirstWinRSO=真 或 Window=0 时 $\Rightarrow >0\text{ms}$ 当 FirstWinRSO=假 或 Window>0 时 $\Rightarrow >40\text{ms} + (\text{TARSO} * \text{Window})\text{ms}$ (40ms 的延时确保传输是在该时隙内进行的)

3.1.6 出错清单和出错处理

错误用以下代码表示:

EP = 物理层的错误;

— = 分隔符号;

N = 出错编号;

F = 致命错误。

表 10 错误一览表

EP-1	在数据链路层请求传送帧之前(主站)TOL 唤醒时间到,或者是在从主站接收到任何数据之前(从站)的 TAlO 唤醒时间到
	这个错误可能导致在通知数据链路层之后产生一个“唤醒呼叫”信号
EP-2	在任何“唤醒呼叫”信号之前 TOAG 唤醒时间到
	这个错误可能导致在通知数据链路层之后产生一个“唤醒呼叫”信号
EP-3	收到一个报警
	这个错误会在通知数据链路层之后引起一次物理层的重新初始化
EP-3F	TOE 唤醒到期后检测到一个长度异常的传输帧
	这个错误会在通知数据链路层之后引起一次物理层的重新初始化
EP-4F	接收到的帧的字节数大于 MaxIndex(发送太多)
	这个错误会在通知数据链路层之后引起一次物理层的重新初始化
EP-5F	当接收一个 RSO 帧(只对于主站)时 TARSO 唤醒时间到
	这个错误会在通知数据链路层之后引起一次物理层的重新初始化

如果发生了表 10 中任何一种错误,它将在本地通过 Phy_ABORT.ind 服务基元上传。附录 C 中给出了全部的致命错误的清单。

3.2 数据链路层

3.2.1 Link-62056-31 协议

不带 DLMS 的本地总线数据交换结构的数据链路层的 Link-62056-31 协议是不对称结构的。所以,主站的状态机制和从站的不一样。

数据链路层将物理层使用的物理信道转换成一个能传输可靠信息的逻辑信道,其主要功能有:

- 执行数据的串行化操作和数据的恢复操作(设物理信道每次能进行 1 位串行化操作);
- 实现发送帧与接收帧的同步;
- 根据主站和从站的地址对帧进行过滤;
- 确保对传输中的错误实现有效的保护。

3.2.2 交换的管理

在主站,Link-62056-31 协议根据从站的类型管理者 AGN 或 AGT 唤醒呼叫信号的传输。当检测到从上层接收到的 DSDU 地址不兼容时,表明发生了致命错误,并终止 Link-62056-31 协议。

在从站,接收到一个错误帧之后不须进行任何处理,这是由于恢复工作将由主站进行。

对于无源站点,在 AGT“唤醒呼叫”信号之后检测到一个“遗漏站点呼叫”时将产生一个 RSO 应答,这个应答总是发生在第 1 个时隙中。于是,当主站在这样一个序列后检测到一个冲突时,将进行第 2 个“遗漏站点呼叫”操作,但这次是在 AGN“唤醒呼叫”信号之后。当然,在第 1 次呼叫后没有发生冲突时,是因为有 1 个遗漏站点或没有任何遗漏站点应答,此时,就不需要进行第 2 次呼叫了。

3.2.3 数据链路服务和服务基元

Link-62056-31 协议的用户可以使用的服务和基元在表 11 中列出。

表 11 数据链路服务和服务基元

服 务	服务基元
DL_DATA	DL_DATA.req(DSDU)
	DL_DATA.ind(DSDU)

表 11 (续)

服 务	服务基元
DL_ALARM	DL_ALARM.req() DL_ALARM.ind()
DL_ABORT	DL_ABORT.req() DL_ABORT.ind(ErrorNb)

赋予每个基元的任务是:

- DL_DATA.req(DSDU)使应用层能够请求数据链路层发送一个 DSDU 数据包;
- DL_DATA.ind(DSDU)使数据链路层通知应用层一个 DSDU 数据包到达;
- DL_ALARM.req()使应用层能够请求数据链路层发送一个报警;
- DL_ALARM.ind()使数据链路层通知应用层一个报警到达;
- DL_ABORT.req()使应用层能够请求数据链路层结束其激活状态;
- DL_ABORT.ind(ErrorNb)使数据链路层能够通知应用层发生了一个出错标识号为 ErrorNb 的致命错误。

3.2.4 数据链路参数

对于主站来说,MaxRetry 是在断开连接前传送给定帧的重试次数,MaxRetry 设为 2。

MaxChain——当没有唤醒呼叫信号而进行远程读取和远程传送时所链接的报文序列数,为了和使用原版本协议的从站保持兼容,此数设为 5。

MaxRSO 为处理“遗漏漏点呼叫”的 RSO 时隙数的最大值,该值在 AGN“唤醒呼叫”之后被置为 3,在 AGT“唤醒呼叫”之后被置为 1。

从站中有预置的主站地址清单和 TABi 的清单。

从站也对通用主地址 APG 进行应答,此时,将用其编程的第 1 个主地址进行应答。

3.2.5 状态转换

表 12 Link-62056-31 状态转换:主站

初始状态	触发条件	动作命令集	终结状态
Initial	\$ true ()	MaxRetry=2 MaxChain=5	Stopped
Stopped	DL_DATA.req(DSDU) &. check_req(DSDU)	NbChain=0 MaxRSO=3 PreSel=FALSE NoRetry=FALSE RepeatASO=FALSE EP-1=FALSE Context(ADS,ADP,TypeAG) Com=command(DSDU) init(Com,TypeAG) Phy_APPG.req(TypeAG)	W. AG
Stopped	DL_DATA.req(DSDU) &. not(check_req(DSDU))	DL_ABORT.ind(EL-1F)	Stopped
Stopped	Phy_ALARM.ind()	DL_ALARM.ind()	Stopped
W. AG	Phy_APPG.ind() &. not(PreSel) &. not(NoRetry) &. not(RepeatASO)	\$ none()	T. Req

表 12 (续)

初始状态	触发条件	动作命令集	终结状态
W. AG	Phy_APPG.ind() & PreSel	Index=MaxRetry+1 Fr=PRE Fr=concat(size_frame(Fr), ADS, ADP, Fr) Fr=concat(Fr, crc(Fr)) Phy_DATA.req(Fr)	M. Rec
W. AG	Phy_APPG.ind() & NoRetry	Index=MaxRetry+1 Fr=DSDU Fr=concat(size_frame(Fr), ADS, ADP, Fr) Fr=concat(Fr, crc(Fr)) NbChain=1 Phy_DATA.req(Fr) Noretry=FALSE	M. Rec
W. AG	Phy_APPG.ind() & RepeatASO	RepeatASO=FALSE Phy_ASQ.req(Fr)	M. RSO
W. AG	DL_ABORT.req()	Phy_ABORT.req()	W. EndS
W. Ends	Phy_ABORT.ind(ErrorNb)	\$ none()	T. Error
W. Ends	Phy_ALARM.ind()	DL_ALARM.ind()	Stopped
T. Error	((ErrorNb=EP-1 & TypeAG=AGN) (ErrorNb=EP-2 & TypeAG=AGT)) & (Com< >IB & Com< >TRB)	\$ none()	Stopped
T. Error	((ErrorNb=EP-1 & TypeAG=AGN) (ErrorNb=EP-2 & TypeAG=AGT)) & (Com=IB Com=TRB)	DL_ABORT.ind(ErrorNb)	Stopped
T. Error	ErrorNb<>EP-1 & ErrorNb< >EP-2	DL_ABORT.ind(ErrorNb)	W. EndS
T. Error	(ErrorNb=EP-1) & TypeAG= AGT	\$ none()	W. EndS
T. Req	Com=IB Com=TRB	Fr=DSDU Fr=concat(size_frame(Fr), ADS, ADP, Fr) Fr=concat(Fr, crc(Fr)) Phy_UNACK.req(Fr)	W. EndS
T. Req	Com=ASO & TypeAG=AGN	MaxRSO=3 NbRSO=1 ListRSO="" Collision=FALSE Fr=DSDU Fr=concat(size_frame(Fr), ADS, ADP, Fr) Fr=concat(Fr, crc(Fr)) Phy_ASQ.req(Fr)	M. RSO

表 12 (续)

初始状态	触发条件	动作命令集	终结状态
<i>T. Req</i>	Com=ASO & TypeAG=AGT	MaxRSO=1 NbRSO=1 ListRSO="" Collision=FALSE Fr=DSDU Fr=concat(size_frame(Fr), ADS, ADP, Fr) Fr=concat(Fr, crc(Fr)) Phy_ASO.req(Fr)	M. RSO
<i>T. Req</i>	((NbChain<MaxChain) & (Com=ENQ Com=TRF)) (NbChain=0 & Com=REC)	Fr=DSDU Fr=concat(size_frame(Fr), ADS, ADP, Fr) Fr=concat(Fr, crc(Fr)) Index=1 NbChain=NbChain+1 Phy_DATA.req(Fr)	M. Rec
<i>T. Req</i>	Com=AUT	Fr=DSDU Fr=concat(size_frame(Fr), ADS, ADP, Fr) Fr=concat(Fr, crc(Fr)) Index=1 NbChain=MaxChain Phy_DATA.req(Fr)	M. Rec
<i>T. Req</i>	(NbChain>=MaxChain) (NbChain<>0 & Com=REC)	NbChain=0 Phy_APPG.req(AGN)	W. AG
M. Rec	Phy_DATA.ind(Frame) & check_frame(Frame) & not(PreSel)	DSDU=extract_DSDU(Frame) DL_ DATA.ind(DSDU)	M. Send
M. Rec	Phy_DATA.ind(Frame) & check_frame(Frame) & command(Frame)=SEL & PreSel	PreSel=FALSE	<i>T. Req</i>
M. Rec	Phy_DATA.ind(Frame) & check_frame(Frame) & command(Frame)<>SEL&PreSel	Phy_ABORT.req() DL_ABORT.ind(EL-2F)	W. Ends
M. Rec	Phy_DATA.ind(Frame) & not(check_frame(Frame)) & Index<=MaxRetry	Index=Index+1 Phy_DATA.req(Fr)	M. Rec
M. Rec	Phy_DATA.ind(Frame) & not(check_frame(Frame)) & Index>MaxRetry	Phy_ABORT.req() DL_ABORT.ind(EL-2F)	W. EndS
M. Rec	DL_ABORT.req()	Phy_ABORT.req()	W. EndS
M. Rec	Phy_ABORT.ind(ErrorNb)	DL_ABORT.ind(ErrorNb)	W. EndS

表 12 (续)

初始状态	触发条件	动作命令集	终结状态
M. RSO	Phy_ASO.ind(Frame) & Size(Frame)=0	\$ none()	T. RSO
M. RSO	Phy_ASO.ind(Frame) & Size(Frame)<>0 & Check_frame(Frame) & Command(Frame)=RSO	Build_RSO(ListRSO,Frame)	T. RSO
M. RSO	Phy_ASO.ind(Frame) & size(Frame)<>0 & not(check_frame(Frame))	Collision=TRUE	T. RSO
M. RSO	Phy_COLL.ind()	Collision=TRUE	T. RSO
M. RSO	DL_ABORT.req()	Phy_ABORT.req()	W. EndS
M. RSO	Phy_ABORT.int(ErrorNb)	DL_ABORT.ind(ErrorNb)	W. EndS
T. RSO	MaxRSO=1 & Collision	MaxRSO=3 Collision=FALSE RepeatASO=TRUE Phy_APPG.req(AGN)	W. AG
T. RSO	(MaxRSO=1 & not(Collision)) (MaxRSO<>1 & NbRSO>=MaxRSO)	DSDU=rso(RSO, Collision, ListR- SO) DL_DATA.ind(DSDU)	W. EndS
T. RSO	NbRSO<MaxRSO	NbRSO=NbRSO+1	M. RSO
M. Send	DL_DATA.req(DSDU) & check_req((DSDU) & not(EP-1)	Com_command(DSDU)	T. Req
M. Send	DL_DATA.req(DSDU) & Check_req(DSDU) & EP-1	Com=command(DSDU) NbChain=0 EP-1=FALSE Phy_APPG.req(AGN)	W. AG
M. Send	DL_DATA.req(DSDU) & not(check_req((DSDU))	Phy_ABORT.req() DL_ABORT.ind(EL - 1F)	W. EndS
M. Send	Phy_ABORT.ind(EP-1)	EP-1=TRUE	M. Send
M. Send	Phy_ABORT.ind(EP-2)	\$ none()	Stopped
M. Send	DL_ABORT.req() & not(EP_1 & TypeAG=AGN	Phy_ABORT.req()	W. EndS
M. Send	DL_ABORT.req() & EP_1 & TypeAG=AGN	\$ none()	Stopped
M. Send	Phy_ABORT.ind(ErrorNb) & ErrorNb<>EP-1 & ErrorNb<>EP-2	DL_ABORT.ind(ErrorNb)	W. EndS

表 13 Link-62056-31 状态转换: 从站

初始状态	触发条件	动作命令集	终结状态
<i>Initial</i>	\$ true()	FlagDSO=TRUE Discovered=FALSE Flag_alarm=FALSE	Stopped
Stopped	Phy_DATA.ind(Frame) & check_frame(Frame) & check_address(Frame)	ADP=extract_ADP(Frame) Com=command(Frame)	<i>T. Com</i>
Stopped	Phy_DATA.ind(Frame) & check_frame(Frame) & not(check_address(Frame))	Phy_ABORT.req()	Stopped
Stopped	Phy_DATA.ind(Frame) & not(check_frame(Frame))	\$ none()	Stopped
Stopped	Phy_ABORT.ind(ErrorNb)	DL_ABORT.ind(ErrorNb)	Stopped
Stopped	DL_ALARM.req()	Phy_ALARM.req() Flag_alarm=TRUE	Stopped
<i>T. Com</i>	Com=IB	FlagDSO=TRUE Discovered=FALSE Phg_ABORT.req()	Stopped
<i>T. Com</i>	Com=TRB	DSDU=extract_DSDU (Frame)DL_DATA.ind (DSDU)Phy_ABORT.req()	Stopped
<i>T. Com</i>	Com=PRE	Fr=SElFr=concat (size_frame(Fr), ADS, DP, Fr)Fr= concat(Fr,crc(Fr)) Phy_DATA.req(Fr)	Stopped
<i>T. Com</i>	Com=ASO & test_TABi(Frame, TAB)	Fr=concat(RSO, TAB, ADS) Fr=concat(size_frame(Fr) ADS, ADP, Fr) Fr=concat(Fr, crc(Fr)) Phy_RSO.req (Fr, window_RSO())	Stopped
<i>T. Com</i>	Com=ASO & Not(test_TABi(Frame, TAB))	Phy_ABORT.req()	Stopped
<i>T. Com</i>	Com=ENQ Com=REC Com=TRF COM=AUT	DSDU=extract_DSDU (Frame) DL_DATA.ind(DSDU)	<i>M. Send</i>
<i>M. Send</i>	DL_DATA.req(DSDU)	Fr=DSDU Fr=concat(size_frame(Fr), ADS, ADP, Fr) Fr=concat(Fr, crc(Fr)) Phy_DATA.req(Fr) update_flag_ DSO(command(Fr))	Stopped
<i>M. Send</i>	Phy_ABORT.ind(ErrorNb)	DL_ABORT.ind(ErrorNb)	Stopped

表 14 前述表中状态的含义

状 态	含 义
<i>Initial</i>	初始化层中的变量
<i>Stopped</i>	等待从上层来的第一次请求或从下层来的第一次指示
<i>W. AG</i> (Wait for end of "Wakeup Call")	等待一个“唤醒呼叫”信号传输的结束
<i>W. EndS</i> (Wait for End of Session)	等待会话的结束
<i>T. Req</i> (Test Request)	测试从上层来的请求的属性
<i>M. Rec</i> (Must Receive)	等待从低层来的指示
<i>M. RSO</i> (Must receive RSO)	等待一个紧跟 RSO 帧之后发送的 ASO 帧
<i>T. RSO</i> (Test last RSO)	测试接收 RSO 帧的最后一个时隙的结束
<i>M. Send</i> (Must Send)	等待从上层来的请求的状态
<i>T. Com</i> (Test Command)	测试所收到帧的命令码

表 15 过程函数的定义(按字母排序)

过程或函数	定 义
<i>build_RSO(ListRSO, Frame)</i>	从接收到的 RSO 帧中解出 RSO 元素(TAB 域和 ADS 域),组成处理清单 ListRSO
<i>check_address(Frame)</i>	根据以下识别条件检查识别 ADP 和 ADS 地址: ——ADP 是 APG,或被编程为 ADP 地址的从站 ——若命令码是 ASO、IB、TRB,则 ADS 是 ADG ——若命令码不是 ASO、IB、TRB,则 ADS 是从站地址
<i>check_frame(Frame)</i>	检查接收到的帧 Frame 是正确的: ——帧的长度大于或等于 11,且小于 128 ——CRC 正确 ——字节数和域 N 相符 ——命令码能识别,且字节数和命令码相符
<i>check_req(DSDU)</i>	检查在 DSDU 中的请求命令码和通信中所定义的 ADP 及 ADS 地址是否相符
<i>Command(DSDU)</i> or <i>command(Frame)</i>	解出对接收帧或发送帧 Frame 的 DSDU 命令码的值
<i>concat(N, ADS, ADP, DSDU)</i> or <i>concat(Frame, CRC)</i>	将域 N、ADS、ADP 和 DSDU 相关联,或者将 CRC 和帧的尾部相关联
<i>context(ADS, ADP, TypeAG)</i>	从通信电文中解出相应的值
<i>crc(Frame)</i>	计算出发送帧 Frame 的 CRC 值
<i>extract_ADP(Frame)</i>	若帧中使用的 ADP 值不是 APG,或者是 APG 值但针对从站被编程的 ADP 清单是空的,那么,解出此值;否则解出和从站编程相关的第 1 个 ADP 的值
<i>extract_DSDU(Frame)</i>	从接收到的帧 Frame 中解出 DSDU(COM 和 DATA 域)

表 15 (续)

过程或函数	定 义
init(COM, TypeAG)	若 TypeAG 和 AGT 相等并且帧的长度大于 18 个字节, 将 PreSel 设为真; 若 TypeAG 和 AGT 相等, 并且帧的长度不大于 18 个字节, 将 NoRetry 设为真
rso(RSO, Collision, ListRSO)	将 RSO 命令码、冲突指示和 RSO 元素列表(TAB 域和 ADS 域)相关联
size(Frame)	计算接收到的帧 Frame 的长度
size_frame(DSDU)	计算将被发送的与 DSDU 相关的帧长度(DSDU 的长度+10)
test_TABi(Frame, TAB)	如果第 1 个包含在 ASO 帧 Frame 内的 TABi 为 00, 那么检查 Discovered=假, 在产生一个从 0~100 随机整数后, 检查此整数是否小于应答概率(第 2 个 TABi), 如小于, 则 TAB 变量被记录为 00; 如果第 1 个包含在 ASO 帧 Frame 内的 TABi 为 FF, 那么检查 Flag_alarm=真, 在这种情况下, Flag_alarm 设为假, 且 TAB 变量被记录为 FF; 如果第 1 个包含在 ASO 帧 Frame 内的 TABi 不是 00 或 FF, 那么检查 FlagDSO=真, 检查从站应被编程为接收到的 ASO 帧 Frame 所包含的 TABi 之一; 这种情况下, TAB 变量被记录为这些值的第 1 个值
update_flag_DSO(COM)	将 Flag_DSO 设为假, 如果 COM 等于 ENQ, 会发现 Discovered=真
window_RSO()	产生一个 0~MaxRSO-1 之间的随机整数, 用作 RSO 时隙数, 这些时隙用于从站应答(参考附录 F)

3.2.6 出错清单和出错处理

错误用以下代码表示:

EP = 物理层的错误;

— = 分隔符号;

N = 出错编号;

F = 致命错误。

表 16 错误一览表

EL-1F	(主站)在通信的过程中, 接收到一个和 ADP 和 ADS 地址不兼容的命令码
	这个错误在通知应用层之后会导致一个数据链路层的重新初始化
EL-2F	在无源站点预选帧之后或重复传输 MaxRetry 次请求之后, 从站接收到一个不正确的应答
	这个错误在通知应用层之后会导致一个数据链路层的重新初始化

如果表 16 中任何的错误发生了, 它将在本地通过 DL_ABORT.ind 服务基元上传。附录 C 中给出了全部的致命错误的清单。

3.3 应用层

3.3.1 Application-62056-31 协议

不带 DLMS 的本地总线数据交换结构的应用层的 Application-62056-31 协议是不对称结构的。所以, 主站的状态机制和从站的不一样。

不带 DLMS 的本地总线数据交换结构的应用层的 Application-62056-31 协议通过分析从用户应用所提供的命令码来控制 and 链接连续的消息。

3.3.2 应用服务和 service 基元

Application-62056-31 协议的用户可以使用的服务和 service 基元在表 17 中列出。

表 17 应用服务和 service 基元

服 务	服 务 基 元
A_DATA	A_DATA.req(COM,ASDU) A_DATA.ind(ASDU)
A_ALARM	A_ALARM.req() A_ALARM.ind()
A_ABORT	A_ABORT.req() A_ABORT.ind(ErrorNb)

赋予每个基元的任务是：

- A_DATA.req(COM,ASDU)使应用程序能够请求应用层发送一个和一个 ASDU 信息单元相关的 COM(对于主站来说,是 ENQ、REC、TRF、TRB、IB 或 ASO;对于从站来说,是 DAT、DRJ、EOS 或 TRA)命令码);
- A_DATA.ind(ASDU)使应用层通知应用程序一个 ASDU 数据包到达;
- A_ALARM.req()使应用程序能够请求应用层发送一个报警;
- A_ALARM.ind()使应用层通知应用程序一个报警到达;
- A_ABORT.req()使应用程序能够请求应用层结束其激活状态;
- A_ABORT.ind(ErrorNb) 使应用层能够通知应用程序发生了一个出错标识号为 ErrorNb 的致命错误。

3.3.3 应用参数

主站必须知道所有的从站的 DES 加密密钥,以便进行远程编程工作。

远程编程时,从站必须知道主站的 DES 加密密钥。

3.3.4 状态转换

表 18 Application-62056-31 状态转换:主站

初始状态	触发条件	动作命令集	终结状态
Stopped	A_DATA.req(Com, ASDU) & (Com=ASO Com=ENQ Com=TRF)	APDU=concat(Com,_,_,ASDU) DL_DATA.req(APDU)	M.Rec
Stopped	A_DATA.req(Com, ASDU) & (Com=IB Com=TRB)	APDU=concat(Com,_,_,ASDU) DL_DATA.req(APDU)	W.EndS
Stopped	A_DATA.req(Com, ASDU) & Com=REC	Na1=randomize() Zdt=zdt(ASDU) APDU=concat(REC,Na1,0,Zdt) DL_DATA.req(APDU) Na1k=cipher(Na1)	M.Rec
Stopped	DL_ALARM.ind()	A_ALARM.ind()	Stopped
Stopped	A_ABORT.req()	DL_ABORT.req()	Stopped
M.Rec	DL_DATA.ind(DSDU)	Resp=command(DSDU)	T.Resp
M.Rec	A_ABORT.req()	DL_ABORT.req()	Stopped
M.Rec	DL_ABORT.ind(ErrorNb) & Error_Nb<>EP_1 & Error_Nb<>EP_2	A_ABORT.ind(ErrorNb)	Stopped

表 18 (续)

初始状态	触发条件	动作命令集	终结状态
M. Rec	DL_ABORT. ind(ErrorNb) & (Error_Nb=EP_1 Error_Nb=EP_2)	\$ none()	M. Rec
W. EndS	DL_ABORT. ind(ErrorNb)	A_ABORT. ind(ErrorNb)	Stopped
W. Ends	A_ABORT. req()	DL_ABORT. req()	W. EndS
T. Resp	(Com=ASO & Resp=RSO) (Com=ENQ & Resp=DAT) (Com=ENQ & Resp=DRJ) (Com=TRF & Resp=TRA) (Com=TRF & Resp=DRJ) (Com=AUT & Resp=EOS) (Com=AUT & Resp=DRJ)	A_DATA. ind(DSDU)	Stopped
T. Resp	Com=REC & Resp=ECH & na1k(DSDU)=Na1k & Zdt= zdt (DSDU)	Na2=na2(DSDU) Na2k=cipher(Na2) ComAUT APDU=concat(AUT, 0, Na2k, "") DL_DATA. req(APDU)	M. Rec
T. Resp	(Com=ASO & Resp<>RSO) (Com=ENQ & Resp<>DAT & Resp<> >DRJ) (Com=TRF & Resp<>TRA & Resp<> >DRJ) (Com=REC & Resp<>ECH) (Com= AUT & Resp=ARJ & Resp<>DRJ & Resp<>EOS)	A_ABORT. ind(EA-1F) DL_A- BORT. req()	Stopped
T. Resp	Com=REC & (Resp<>ECH (na1k(DSDU)<>Na1k) (Zdt<>zdt(DSDU))	A_ABORT. ind(EA-2F) DL_A- BORT. req()	Stopped
T. Resp	Com=AUT & Resp=ARJ	A_ABORT. ind(EA-3F) DL_ABORT. req()	Stopped

表 19 Application-62056-31 状态转换: 从站

初始状态	触发条件	动作命令集	终结状态
Stopped	DL_DATA. ind(DSDU) & (command(DSDU)=ENQ command(DSDU)=TRF)	A_DATA. ind(DSDU) Req=command(DSDU)	M. Send
Stopped	DL_DATA. ind(DSDU) & command(DSDU) =TRB	A_DATA. ind(DSDU)	Stopped
Stopped	DL_DATA. ind(DSDU) & command(DSDU) =REC	Zdt=zdt(DSDU) Na1=na1(DSDU) Na1k=cipher(Na1) Na2=randomize() APDU=concat(ECH, Na1k, Na2, Zdt) DL_DATA. req(APDU) Na2k=cipher(Na2) Req=REC	W. AUT

表 19 (续)

初始状态	触发条件	动作命令集	终结状态
Stopped	A_ALARM.req() & alarm_detection()	DL_ALARM.req()	Stopped
M. Send	A_DATA.req(COM, ASDU) & (((COM = DAT COM=DRJ) & Req=ENQ ((COM = TRA COM = DRJ) & Req = TRF) (COM=DRJ & Req=REC))	APDU=concat(COM,_,_,ASDU) DL_DATA.req(APDU)	Stopped
M. Send	A_DATA.req(COM, ASDU) & (COM=EOS & Req=REC)	APDU=concat(EOS, 0, 0, "") DL_DATA.req(APDU)	Stopped
M. Send	DL_ABORT.ind(ErrorNb)	A_ABORT.ind(ErrorNb)	Stopped
W. AUT	DL_DATA.ind(DSDU) & command(DSDU) = AUT& na2k(DSDU) = Na2k	ASDU=concat(REC,_,_,Zdt) A_DATA.ind(ASDU)	M. Send
W. AUT	DL_DATA.ind(DSDU) & Command(DSDU) = AUT& na2k(DSDU) < >Na2k	APDU=concat(ARJ,_,_, "") DL_DATA.req(APDU)	Stopped
W. AUT	DL_ABORT.ind(ErrorNb)	A_ABORT.ind(ErrorNb)	Stopped

表 20 前述表中列出的状态的含义

状 态	含 义
Stopped	等待从上层来的第 1 次请求,或从低层来的第 1 个指示
M. Rec (Must Receive)	准备响应发来的请求
T. Resp (Test Response)	处理接收到的应答
M. Send (Must Send)	等待收到请求的响应
W. AUT (Wait for AUT frame)	在发送一个 ECH 应答帧之后,等待一个 AUT 帧

表 21 过程函数的定义(按字母排序)

过程、函数	定 义
alarm_detection()	检查报警模式的状态是激活的
cipher(Na1) or cipher(Na2)	根据通信报文中包含的密钥,用 DES 算法给随机数 Na1 或 Na2 加密
command(DSDU)	解出收到的 DSDU 中的命令码值
concat(COM,_,_, ASDU), concat(COM, ZA1, ZA2, ZDT) or concat(COM, 0, 0, _)	将一个命令码 COM 和一个 ASDU,或将一个命令码和一个加密的值 ZA1、一个加密值 ZA2 和一个数据域 ZDT(TAB 和 DATA)相关联,或将一个命令码与 ZA1=0 和 ZA2=0 域相关联
na1(DSDU)	从收到的 REC 帧中的 ZA1 域中解出 Na1 的值
na1k(DSDU)	从收到的 ECH 帧中的 ZA1 域中解出 Na1k 的值
na2(DSDU)	从收到的 ECH 帧中的 ZA2 域中解出 Na2 的值

表 21 (续)

过程、函数	定 义
na2k(DSDU)	从收到的 AUT 帧中的 ZA2 域中解出 Na2k 的值
randomize()	根据附录 G 中描述的过程产生一个随机数
zdt(ASDU) or zdt(DSDU)	从一个 REC 请求、一个 REC 帧或一个 ECH 帧中解出数据(TAB 和 DATA)

3.3.5 出错清单和出错处理

错误用以下代码表示:

EP = 物理层的错误;

— = 分隔符号;

N = 出错编号;

F = 致命错误。

表 22 错误一览表

EA-1F	接收到的帧中的应答命令码和请求命令码(主站的)不符
	这个错误在通知应用层之后会导致一个应用层的重新初始化
EA-2F	从站没有通过(主站)的验证
	这个错误在通知应用层之后会导致一个应用层的重新初始化
EA-3F	主站没有通过验证(仅对主站)
	这个错误在通知应用层之后会导致一个应用层的重新初始化

如果发生了表 22 中任何一种错误,它将在本地通过 A_ABORT.ind 服务基元上传。附录 C 中给出了全部的致命错误的清单。

4 带有 DLMS 的本地总线数据交换

4.1 物理层

带有 DLMS 的本地总线数据交换结构的物理层的 Physical-62056-31 协议和不带 DLMS 的本地总线数据交换结构中的定义相同。

4.2 数据链路层

4.2.1 Link-E/D 协议

带有 DLMS 的本地总线数据交换结构的数据链路层的 Link-E/D 协议是不对称结构的。所以,主站的状态机制和从站的不一样。

数据链路层将物理层使用的物理信道转换成一个能传输可靠信息的逻辑信道,其主要功能有:

- 直接管理着总线初始化和遗漏站点呼叫服务;
- 执行数据的串行化操作和数据的恢复操作(若物理信道的串行处理能力是 1 位/次);
- 实现发送帧与接收帧的同步;
- 根据主站和从站的地址过滤帧;
- 确保对传输中的错误有效地检测。

4.2.2 交换的管理

总线初始化、报警和遗漏站点呼叫服务由带有 DLMS 的本地总线数据交换结构的数据链路层的 Link-E/D 协议提供,但是它们的操作是发生在应用层以外的。特定情况下,当进行远程读取交换时,遗漏站点标志可以被应用程序编程接口来修改。

除了通信会话和总线初始化的开始、报警报文或遗漏站点呼叫管理以外,协议是完全对称的,站点

之间轮流交替地担当着传送和接收的工作。

发送了一帧之后,传送方的数据链路层在再次传送之前总是等待一帧从接收方数据链路层发来的报文。这个等待时间由 T1 唤醒时间来控制,其延续时间为 10s。

当传送了一帧并且接收到其应答之后,当前帧才开始传送出去。传输重试次数由 MaxRetry 来限制,超过了这个次数,通信将在数据链路层上停止并通知给应用层。

系统中的一方每次收到一帧后,立即传送一个应答帧。这个应答帧可以包含从应用层来的数据。它总是包含一个根据先前发送和接收的报文而计算出来的发送码值和确认码值,以下是计算这些码值的算法:

- 若上一帧接收到的报文是无误的,并且其发送码值和先前传输的确认码值的 1 的补码相等,那么,此数据包将传给应用层,下一个传输帧将带有一个和接收到的发送码值相等的确认码值;否则,确认码值将不做修改,数据包也不会传给应用层。
- 若上一帧接收到的报文是无误的,并且其确认码值和先前传输中的发送码值完全相等,则下一帧传输的发送码值将加 1 代表新的一个数据包的传输。
- 若上一帧的接收出错,或者其确认码值和先前传输中的发送码值不相等,那么只要是在传输重试次数不大于 MaxRetry 的条件下,须将同一帧重新发送一次。

4.2.3 数据链路服务和服基元

Link-E/D 协议的用户可以使用的服务和服基元在表 23 中列出。

表 23 数据链路服务和服基元

服 务	服 务 基 元
DL_DATA	DL_DATA.req(Pr,DSDU) DL_DATA.ind(Pr,DSDU)
DL_IB	DL_IB.req()
DL_ASO	DL_ASO.req(DSDU) DL_ASO.ind(Collision,List)
DL_ALARM	DL_ALARM.req()
DL_ABORT	DL_ABORT.req(Strong) DL_ABORT.ind(ErrorNb)

赋予每个基元的任务是:

- DL_DATA.req(Pr,DSDU)使应用层能够请求数据链路层发送一个带有优先级 Pr²⁾ 的 DSDU 数据包;
- DL_DATA.ind(Pr,DSDU)使数据链路层通知应用层一个带有优先级 Pr 的 DSDU 数据包到达;
- DL_IB.req()使应用程序编程接口能请求数据链路层发送一个总线初始化帧;
- DL_ASO.req(DSDU)使应用程序编程接口能根据 DSDU 数据包请求数据链路层发送一个遗漏站点呼叫帧;
- DL_ASO.ind(Collision,List)使数据链路层能通知应用程序编程接口遗漏站点呼叫的结果;
- DL_ALARM.req()使应用程序编程接口能够请求数据链路层发送一个报警;
- DL_ABORT.req(Strong)使应用层能够请求数据链路层以 Strong 的优先级³⁾ 结束其激活状态;
- DL_ABORT.ind(ErrorNb) 使数据链路层能够通知应用层发生了一个出错标识号为 ErrorNb

2) 优先级 Pr 可以区分紧急服务的处理顺序,例如 InformationReport(Pr=1)优先于其他 DLMS 服务(Pr=0)

3) 参数 Strong 可使致命错误的处理(Strong=1)优先于由应用子层初始化的其他物理层的断开请求(Strong=0)

的致命错误。

4.2.4 数据链路参数

MaxRetry——在断开连接之前传输给定帧的重试次数, 设为 2。

MaxRSO——对于主站来说, 为处理“遗漏站点呼叫”所需的最多 RSO 时隙数, 设为 3。

从站中应当有预置的主站地址清单和 TABi 的清单。

从站也对通用主地址 APG 进行应答, 此时, 将用其编程的第 1 个主地址进行应答。

4.2.5 转换状态

表 24 Lin -E/D 状态转换: 主站

初始状态	触发条件	动作命令集	终结状态
Initial	\$ true ()	MaxRetry=2 MaxChain=5 init_incrChain()	Stopped
Stopped	exist_dl_req()	NbChain=0 RepeatASO=FALSE context(ADP, ADS, TypeAG) init(TypeAG) Phy_APPG_req(TypeAG)	W. AG
Stopped	Phy_ABORT.ind(ErrorNb)	DL_ABORT.ind(ErrorNb)	Stopped
Stopped	Phy_ALARM.ind()	creat_alarm(TPDU) DL_DATA.ind(Pr=1, TPDU)	Stopped
W. AG	Phy_APPG.ind() & not(RepeatASO) & NbChain=0	\$ none()	T. Req
W. AG	Phy_APPG.ind() & not(RepeatASO) & NbChain<>0	NbChain=0	M. Send
W. AG	Phy_APPG.ind() & RepeatASO	RepeatASO=FALSE Phy_ASO_req(Fr)	M. RSO
W. AG	DL_ABORT.req()	Phy_ABORT.req()	W. EndS
W. Ends	(Phy_ABORT.ind(EP-2) & TypeAG=AGT) (Phy_ABORT.ind(EP-1) & TypeAG=AGN)	\$ none()	Stopped
W. Ends	Phy_ALARM.ind	creat_alarm(TPDU) DL_DATA.ind(Pr=1, TPDU)	Stopped
W. Ends	Phy_ABORT.ind(ErrorNb) & ErrorNb<>EP-1 & ErrorNb<>EP-2	DL_ABORT.ind(ErrorNb)	W. EndS
T. Req	exist_dl_req(DL_IB. Req())	Fr="" Size=size_frame(Fr) Fr=concat(size_ADS, ADP, IB, Fr) Fr=concat(Fr, crc(Fr)) Phy_UNACK.req(Fr)	W. EndS

表 24 (续)

初始状态	触发条件	动作命令集	终结状态
<i>T. Req</i>	$\text{exist_dl_req}(\text{DL_ASO_req}(\text{DSDU})) \ \& \ \text{TypeAG}=\text{AGN}$	$\text{MaxRSO}=3$ $\text{NbRSO}=1$ $\text{ListRSO}=""$ $\text{Collision}=\text{FALSE}$ $\text{Fr}=\text{DSDU}$ $\text{Size}=\text{size_frame}(\text{Fr})$ $\text{Fr}=\text{concat}(\text{Size}, \text{ADS}, \text{ADP}, \text{ASO}, \text{Fr})$ $\text{Fr}=\text{concat}(\text{Fr}, \text{crc}(\text{Fr}))$ $\text{Phy_ASO_req}(\text{Fr})$	M. RSO
<i>T. Req</i>	$\text{exist_dl_req}(\text{DL_ASO_req}(\text{DSDU})) \ \& \ \text{TypeAG}=\text{AGT}$	$\text{MaxRSO}=1$ $\text{NbRSO}=1$ $\text{ListRSO}=""$ $\text{Collision}=\text{FALSE}$ $\text{Fr}=\text{DSDU}$ $\text{Size}=\text{size_frame}(\text{Fr})$ $\text{Fr}=\text{concat}(\text{Size}, \text{ADS}, \text{ADP}, \text{ASO}, \text{Fr})$ $\text{Fr}=\text{concat}(\text{Fr}, \text{crc}(\text{Fr}))$ $\text{Phy_ASO_req}(\text{Fr})$	M. RSO
<i>T. Req</i>	$\text{not}(\text{exist_dl_req}(\text{DL_IB_req}())) \mid \text{exist_dl_req}(\text{DL_ASO_req}())$	$\text{Pr}=0$ $\text{Send}="00"\text{B}$ $\text{Confirm}="11"\text{B}$ $\text{Fr}=""$ $\text{Index}=\text{Index}+1$ $\text{NbChain}=\text{NbChain}+\text{IncrChain}$ $\text{Size}=\text{size_frame}(\text{Fr})$ $\text{Com}=\text{com}(\text{DATA}+, \text{Pr}, \text{Send}, \text{Confirm})$ $\text{Fr}=\text{concat}(\text{Size}, \text{ADS}, \text{ADP}, \text{Com}, \text{Fr})$ $\text{Fr}=\text{concat}(\text{Fr}, \text{crc}(\text{Fr}))$ $\text{Phy_DATA_req}(\text{Fr})$	M. Rec
M. RSO	$\text{Phy_ASO_ind}(\text{Frame}) \ \& \ \text{size}(\text{Frame})=0$	$\$ \text{none}()$	<i>T. RSO</i>
M. RSO	$\text{Phy_ASO_ind}(\text{Frame}) \ \& \ \text{check_frame}(\text{Frame}) \ \& \ \text{command}(\text{Frame})\text{RSO}$	$\text{build_RSO}(\text{ListRSO}, \text{Frame})$	<i>T. RSO</i>
M. RSO	$\text{Phy_ASO_ind}(\text{Frame}) \ \& \ \text{not}(\text{check_frame}(\text{Frame})) \ \& \ \text{size}(\text{Frame}) < > 0$	$\text{Collision}=\text{TRUE}$	<i>T. RSO</i>
M. RSO	$\text{Phy_COLL_ind}()$	$\text{Collision}(\text{TRUE})$	<i>T. RSO</i>
M. RSO	$\text{DL_ABORT_req}()$	$\text{Phy_ABORT_req}()$	W. EndS
M. RSO	$\text{Phy_ABORT_ind}(\text{ErrorNb})$	$\text{DL_ABORT_ind}(\text{ErrorNb})$	W. EndS
<i>T. RSO</i>	$\text{MaxRSO}=1 \ \& \ \text{Collision}$	$\text{MaxRSO}=3$ $\text{Collision}=\text{FALSE}$ $\text{RepeatASO}=\text{TRUE}$ $\text{Phy_APPG_req}(\text{AGN})$	W. AG

表 24 (续)

初始状态	触发条件	动作命令集	终结状态
<i>T. RSO</i>	(MaxRSO=1 & not(Collision)) (MaxRSO<>1 & NbRSO>=MaxRSO)	DL_ASC.ind(Collision, ListRSO)	W. EndS
<i>T. RSO</i>	NbRSO<MaxRSO	NbRSO=NbRSO+1	M. RSO
<i>M. Send</i>	exist_dl_data_req(DL_DATA.req (Pr=1,DSDU)) & NbChain<MaxChain	Send=incr(Send) Ack_expected=TRUE Fr=DSDU Index=Index+1 NbChain=NbChain+IncrChain Size=size_frame(Fr) Com=com(DATA+,Pr,Send,Confirm) Fr=concat(Size,ADS,ADP, Com,Fr) Fr=concat(Fr,crc(Fr)) Phy_DATA.req(Fr)	M. Rec
<i>M. Send</i>	not(DL_DATA.req(Pr=1, _)) & exist_dl_data_req(DL_DATA. req(Pr=0,DSDU)) & NbChain<MaxChain	Send=incr(Send) Ack_expected=TRUE Fr=DSDU Index=Index+1 NbChain=NbChain+IncrChain Size=size_frame(Fr) Com=com(DATA+,Pr,Send,Con- firm) Fr=concat(Size,ADS,ADP,Com,Fr) Fr=concat(Fr,crc(Fr)) Phy_DATA.req(Fr)	M. Rec
<i>M. Send</i>	not(DL_DATA.req(_, _)) & NbChain<MaxChain	Pr=0 Fr="" Index=Index+1 NbChain=NbChain+IncrChain Size=size_frame(Fr) Com=com(DATA+,Pr,Send,Com- firm) Fr=concat(Size,ADS,ADP,Com,Fr) Fr=concat(Fr,crc(Fr)) Phy_DATA.req(Fr)	M. Rec
<i>M. Send</i>	NbChain>=MaxChain	Phy_APPG.req(AGN)	W. AG
<i>M. Rec</i>	Phy_DATA.ind(Frame) & (check_frame(Frame) & check_address(Frame) & is_data+(Frame) & is_ack(Frame)) & is_text(Frame)	DL_DATA.ind(extract_prty(Frame),ex- tract_text(Frame)) Confirm=incr(Confirm) Ack_expected=FALSE Index=0	<i>M. Send</i>

表 24 (续)

初始状态	触发条件	动作命令集	终结状态
M. Rec	Phy_DATA.ind(Frame) & (check_frame(Frame) & check_address(Frame) & is_data+(Frame) & is_ack(Frame)) & not(is_text(Frame))	Ack_expected=FALSE Index=0	M. Send
M. Rec	Phy_DATA.ind(Frame) & not(check_frame(Frame) & check_address(Frame) & is_data+(Frame) & is_ack(Frame)) & Index≤MaxRetry	Phy_DATA.req(Fr) Index=Index+1	M. Rec
M. Rec	Phy_DATA.ind(Frame) & not(check_frame(Frame) & check_address(Frame) & is_data+(Frame) & is_ack(Frame)) & Index>MaxRetry	DL_ABORT.ind(EL-2F) Phy_ABORT.req()	W. Ends
M. Rec	DL_ABORT.req(Strong=0) & not(DL_DATA.req(.,.,.)) & Ack_expected=FALSE	Phy_ABORT.req()	W. EndS
M. Rec	DL_ABORT.req(Strong=1)	Phy_ABORT.req()	W. EndS
M. Rec	Phy_ABORT.ind(ErrorNb)	DL_ABORT.ind(ErrorNb)	W. EndS

表 25 Link-E/D 状态转换:从站

初始状态	触发条件	命令集	终结状态
Initial	\$ true()	MaxRetry=2 FlagDSO=TRUE Discovered=FALSE Flag_alarm=FALSE	Stopped
Stopped	Phy_DATA.ind(Frame) & check_frame(Frame) & check_address(Frame)	ADP=extract_ADP(Frame) Com=command(Frame)	T. Com
Stopped	Phy_DATA.ind(Frame) & check_frame(Frame) & not(check_address(Frame))	Phy_ABORT.req()	Stopped
Stopped	Phy_DATA.ind(Frame) & not(check_frame(Frame))	\$ none()	Stopped
Stopped	DL_ALARM.req() & alarm_detection()	Phy_ABORT.req() Flag_alarm=TRUE Phy_ALARM.req()	Stopped
T. Com	Com=IB	FlagDSO=TRUE Discovered=FALSE Phy_ABORT.req()	Stopped

表 25 (续)

初始状态	触发条件	命令集	终结状态
<i>T. Com</i>	$\text{Com} = \text{ASO} \ \& \ \text{test_TABi}(\text{Frame}, \text{TAB})$	$\text{Fr} = \text{concat}(\text{RSO}, \text{TAB}, \text{ADS})$ $\text{Fr} = \text{concat}(\text{size_frame}(\text{Fr}),$ $\text{ADS}, \text{ADP}, \text{Fr})$ $\text{Fr} = \text{concat}(\text{Fr}, \text{crc}(\text{Fr}))$ $\text{Phy_RSO.req}(\text{Fr}, \text{window_RSO}())$	Stopped
<i>T. Com</i>	$\text{Com} = \text{ASO} \ \& \ \text{not}(\text{test_TABi}(\text{Frame}, \text{TAB}))$	$\text{Phy_ABORT.req}()$	Stopped
<i>T. Com</i>	$\text{is_data}+(\text{Frame}) \ \& \ \text{is_text}(\text{Frame})$	$\text{Ack_expected} = \text{FALSE}$ $\text{Send} = "11"\text{B}$ $\text{Confirm} = "00"\text{B}$ $\text{DL_DATA.ind}(\text{extract_prty},$ $\text{extract_text}(\text{Frame}))$	<i>M. Send</i>
<i>T. Com</i>	$\text{is_data}+(\text{Frame}) \ \& \ \text{not}(\text{is_text}(\text{Frame}))$	$\text{Ack_expected} = \text{FALSE}$ $\text{Send} = "11"\text{B}$ $\text{Confirm} = "00"\text{B}$	<i>M. Send</i>
<i>M. Send</i>	$\text{exist_dl_data_req}(\text{DL_DATA.req}(\text{Pr} = 1, \text{DSDU}))$	$\text{Discovered} = \text{TRUE}$ $\text{Send} = \text{incr}(\text{Send})$ $\text{Ack_expected} = \text{TRUE}$ $\text{Fr} = \text{DSDU}$ $\text{Index} = 1$ $\text{Size} = \text{size_frame}(\text{Fr})$ $\text{Com} = \text{com}(\text{DATA} +, \text{Pr}, \text{Send},$ $\text{Confirm})$ $\text{Fr} = \text{concat}(\text{Size}, \text{ADS}, \text{ADP}, \text{Com},$ $\text{Fr})$ $\text{Fr} = \text{concat}(\text{Fr}, \text{crc}(\text{Fr}))$ $\text{Phy_DATA.req}(\text{Fr})$	<i>M. Rec</i>
<i>M. Send</i>	$\text{not}(\text{DL_DATA.req}(\text{Pr} = 1,)) \ \& \ \text{exist_dl_data_req}(\text{DL_DATA.req}(\text{Pr} = 0, \text{DSDU}))$	$\text{Discovered} = \text{TRUE}$ $\text{Send} = \text{incr}(\text{Send})$ $\text{Ack_expected} = \text{TRUE}$ $\text{Fr} = \text{DSDU}$ $\text{Index} = 1$ $\text{Size} = \text{size_frame}(\text{Fr})$ $\text{Com} = \text{com}(\text{DATA} +, \text{Pr}, \text{Send},$ $\text{Confirm})$ $\text{Fr} = \text{concat}(\text{Size}, \text{ADS}, \text{ADP}, \text{Com},$ $\text{Fr})$ $\text{Fr} = \text{concat}(\text{Fr}, \text{crc}(\text{Fr}))$ $\text{Phy_DATA.req}(\text{Fr})$	<i>M. Rec</i>

表 25 (续)

初始状态	触发条件	命令集	终止状态
M. Send	not(DL_DATA.req(,))	Pr=0 Fr="" Index=1 Size=size_frame(Fr) Com=com(DATA+, Pr, Send, Confirm) Fr=concat(Size, ADS, ADP, Com, Fr) Fr=concat(Fr, crc(Fr)) Phy_DATA.req(Fr)	M. Rec
M. Rec	Phy_DATA.ind(Frame) & (check_frame(Frame) & check_address(Frame) & is_data+(Frame) & is_ack(Frame)) & is_text(Frame)	stop_timer(T1) Confirm=incr(Confirm) Ack_expected=FALSE DL_DATA.ind(extract_prt, extract_text(Frame))	M. Send
M. Rec	Phy_DATA.ind(Frame) & (check_frame(Frame) & check_address(Frame) & is_data+(Frame) & is_ack(Frame)) & not(is_text(Frame))	stop_time(T1) Ack_expected=FALSE	M. Send
M. Rec	Phy_DATA.ind(Frame) & not(check_frame(Frame) & check_address(Frame) & is_data+(Frame) & is_ack(Frame)) & index<=MaxRetry	stop_timer(T1) Phy_DATA.req(Fr) Index=Index+1	M. Rec
M. Rec	Phy_DATA.ind(Frame) & not(check_frame(Frame) & check_address(Frame) & is_data+(Frame) & is_ack(Frame)) & index>MaxRetry	stop_timer(T1) DL_ABORT.ind(EL-2F) Phy_ABORT.req(Fr)	Stopped
M. Rec	DL_ABORT.req(Strong=0) & not((DL_DATA.req(,)) & Ack_expected=FALSE	Stop_timer(T1) Phy_ABORT.req()	Stopped
M. Rec	DL_ALARM.req() & alarm_detection()	Stop_timer(T1) DL_ABORT.ind(EL-1F) Phy_ABORT.req() Flag_alarm=TRUE Phy_ALARM.req()	Stopped

表 25 (续)

初始状态	触发条件	命令集	终结状态
M. Rec	DL_ABORT.req(Strong=1)	Stop_timer(T1) Phy_ABORT.req()	Stopped
M. Rec	Phy_ABORT.ind(EP-1)	init_timer(T1)	M. Rec
M. Rec	Phy_ABORT.ind(ErrorNb) & ErrorNb< >EP-1	Stop_timer(T1) DL_ABORT.ind(ErrorNb)	Stopped
M. Rec	time_out(T1)	DL_ABORT.ind(EL_3F)	Stopped

表 26 前述表中的状态的含义

状 态	含 义
<i>Initial</i>	初始化层中的变量
Stopped	等待从上层来的第 1 次请求或从低层来的第 1 个指示的状态
W. AG (Wait for end of “Wakeup Call”)	等待请求的“唤醒呼叫”信号的结束
W. EndS (Wait for End of Session)	等待会话的结束
T. Req (Test Request)	测试从上层发来的请求的属性
M. RSO (Must receive RSO)	在 ASO 发送之后,等待一个 RSO 帧
T. RSO (Test last RSO)	测试用于 RSO 接收的最后一个时隙的结束
M. Send (Must Send)	一帧将被发送(可能文本域为空)
M. Rec (Must Receive)	等待从低层来的指示信号的状态
T. Com (Test Command)	测试接收到的帧中的命令码

表 27 过程和函数的定义(按字母排序)

过程和函数	定 义
Alarm_detection()	检查报警模式的激活状态
build_RSO(ListRSO,Frame)	从接收到的 RSO 帧 Frame 中解出 RSO 元素(TAB 和 ADS 域),并和前述的 ListRSO 相关联
check_address(Frame)	检查 ADP 和 ADS 地址,根据以下标准识别: ——ADP 是 APG 或站点已经被编程为 ADP 地址 ——若命令码是 ASO 或 IB,则 ADS 是 ADG ——若命令码不是 ASO,也不是 IB,则 ADS 是从站地址
check_frame(Frame)	检查接收到的帧 Frame 是正确的: ——字节数不小于 11,而且不大于 128 ——CRC 是正确的 ——字节数和域大小相符 ——命令码可识别
com(DATA+, Pr, Send, Confirm)	组织各位形成一个命令码
command(Frame)	从接收到的帧 Frame 中解出命令码的值

表 27 (续)

过程和函数	定 义
concat (Size, ADS, ADP, COM, Text) or concat (Frame, CRC)	将域 Size, ADS, ADP, COM 和文本相关联, 将 CRC 和 Frame 的帧尾相关联
context (ADS, ADP, TypeAG)	从通信的报文中解出相应的值
crc (Frame)	计算将要发送的帧 Frame 的 CRC 值
create_alarm (TPDU)	计算 TPDU, 这里 STSAP=0, DTSAP=0, 并且计算 Unsolicited(主动提供的)RegPDU, 其条件是: client-type=FFFF, serveridentifier=0, object-name=FFFF, variable type=boolean, value=true
exist_dl_data_req (DL_DATA.req (Pr, DSDU))	需用到一个 DL_DATA.req (Pr, DSDU) 事件
exist_dl_req ()	检查 DL_IB.req(), DL_ASO.req(DSDU) 或 DL_DATA.req (Pr, DSDU) 事件的存在, 并检查是否是通信报文中定义的 ADP 和 ADS 地址相符
exist_dl_req (DL_IB.req()) or exist_dl_req (DL_ASO.req(DSDU))	需用到一个 DL_IB.req() 或 DL_ASO.req(DSDU) 事件
extract_ADp(Frame)	若帧中使用的 ADP 值不是 APG, 或是 APG 但从站被编程的 ADP 值的清单为空, 则解开此值, 否则按照从站被编程的第一个 ADP 值解开
extract_prtY(Frame)	从接收到的帧 Frame 中解出优先级的域
extract_text(Frame)	从接收到的帧 Frame 中解出文本的域
init (TypeAG)	若 TypeAG 等于 AGT, 将 Index 设为 MaxRetry, 否则设为 0
init_incrChain()	若支持报警功能, 将 IncrChain 设为 0, 否则设为 1
init_timer(T1)	设置唤醒时间 T1
is_ack(Frame)	检查接收到的帧 Frame 中是否包含一个等于上一发送帧中发送域的确认域
is_data+(Frame)	检查接收到的帧 Frame 中是否包含一个正确的 DATA+域("111"B)
is_text(Frame)	检查接收到的帧 Frame 中是否包含一个非空的文本域, 并且发送域和上一帧的确认域相同
size(Frame)	计算接收到的帧 Frame 的长度
size_frame(DSDU)	计算把 DSDU 数据单元算在一起的帧的长度 (size(DSDU)+11)
stop_timer(T1)	停止唤醒时间 T1
test_TABirame, TAB)	若接收到的 ASO 帧 Frame 中的第一个 TABi 等于 00, 检查 Discovered=假, 在生成一个 0~100 之间的随机数之后, 检查这个随机数是否小于应答概率(第二个 TABi), 如小于应答概率, 则 TAB 变量被存储为 00; 如果接收到的 ASO 帧 Frame 的第一个 TABi 等于 FF, 检查 Flag_alarm=真, 这种情况下, Flag_alarm 设置为假, TAB 变量被存储为 FF; 若接收到的 ASO 帧 Frame 的第一个 TABi 不等于 00 或 FF, 检查 Flag_DSO=真, 检查从站被编程为接收到的 ASO 帧 Frame 中的一个 TABi, 这种情况下, TAB 变量被存储为这些值的第一个值
time_out(T1)	唤醒时间 T1 的触发
window_RSO()	产生从 0 到 MaxRSO-1 之间的一个随机整数, 用作从站应答的 RSO 时隙数(参考附录 F)

4.2.6 出错清单和出错处理

错误用以下代码表示：

EP = 数据链路层的错误；

— = 分隔符号；

N = 出错编号；

F = 致命错误。

表 28 错误一览表

EL-1F	在一次关联中接收到 DL_ALARM.req()
	这个请求在请求物理层发出一次报警之后导致一个数据链路层的重新初始化
EL-2F	在传输重试次数 MaxRetry 溢出之后接收到从站的不正确应答
	这个错误在通知应用层之后会导致一次数据链路层的重新初始化,并且导致物理层中止
EL-3F	由于 T1 延时的到期而停止通信
	这个错误在通知应用层之后会导致一个数据链路层的重新初始化,并且导致物理层中止

如果表 28 中任何的错误发生了,它将在本地通过 DL_ABORT.ind 服务基元上传。附录 C 中给出了全部的致命错误的清单。

4.3 应用层

应用层将在 IEC 62056-51 中进行描述,这里只是简要地介绍一下带有 DLMS 的本地总线数据交换结构的操作情况。

4.3.1 传输子层

最大的数据包的长度,MaxPktSize(见 IEC 62056-51),为 114。

4.3.2 应用子层

像 IEC 62056-51 中提到的那样,必须根据使用的不同通信介质,对客户机连接(Client_connect)和服务器连接(Server_connect)函数加以说明。带 DLMS 的本地总线数据交换结构不接受服务器连接函数,这是由于其不支持未请求的服务管理。客户机连接函数定义如表 29 所示。

表 29 客户机连接函数定义

函数过程	定 义
Client_connect(Adp, Ads)	如果没有激活的应用程序相连,此函数将设置低层使用的参数 ADS、ADP 和 TypeAG; 如果已经有激活的应用程序相连,对于相同的(ADP, ADS)地址此函数将是透明的; 如果已经有激活的应用程序相连,但(ADP, ADS)地址不同,此函数将导致失败

5 本地总线数据交换——硬件

5.1 概述

此协议阐述了并联在一个硬件总线上的主站和从站之间的数据交换。主站通过一个无源磁插头连接在总线上。

本文将阐述如下几个项目：

- 信号参数；
- 总线参数；
- 磁插头；
- 主站；

- e) 从站;
- f) 供电参数。

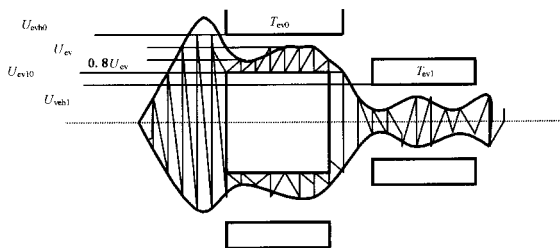
5.2 通用技术条件

5.2.1 信号以 50 kHz 传输

传输包括:

- a) 二进制数据传输;
- b) 双向、半双工;
- c) 传输速率: $1\,200(1 \pm 1\%) \text{ bit/s}$;
- d) 0 和 1 的占空比相同;
- e) $(50 \pm 1.5) \text{ kHz}$ 载波的信号幅值调制(ASM);
- f) 极性:
 - 0——检测到载波;
 - 1——没有检测到载波。

信号的特征参数用图 5 的载波包络线来定义。



特征值:

U_{evh1} 是传输为“1”时的电平最大值。

U_{ev0} 是传输为“0”时的电平最小值。

U_{evh0} 是传输为“0”时的电平最大值。

T_{ev1} 是信号包络线保持低于 U_{evh1} 的最小保证时间。

T_{ev0} 是信号包络线保持在 U_{ev0} 和 U_{evh0} 之间的最小保证时间。

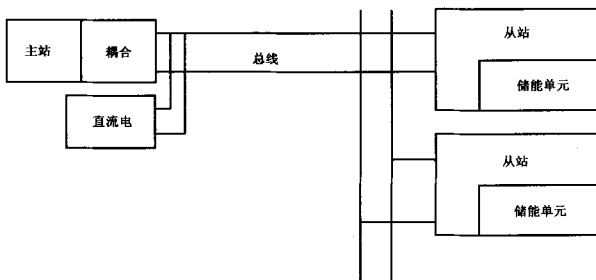
图 5 总线上的信号波形

- a) U_{ev0} 和 U_{evh0} 并不是波形包络线的极限值,只是为了确保正确操作而规定的值。
- b) 在 T_{ev0} 期间,波形包络电平的变化不应超过 20%。
- c) 在 T_{ev0} 和 T_{ev1} 之间的空隙内,波形包络线上升和下降是呈指数态或阻尼正弦态变化,频率也会发生突变。
- d) 用 $100 \, \Omega$ 的电阻和 $31.8 \, \text{nF}$ 的电容代替总线时,波形连续传输时,信号的总谐波失真低于 15%。
- e) 所有的电压值都指的是峰值。
- f) 在定义位的时序(“1”和“0”的时间)时,必须考虑以下参数:
 - 确保传输“0”的最大时间信号 $\geq U_{ev0}$;
 - 不确保传输“0”的最大时间信号 $\geq U_{evh1}$;
 - 确保传输“1”的最大时间信号 $\leq U_{evh1}$;
 - 不确保传输“1”的最大时间信号 $\leq U_{ev0}$ 。

5.2.2 信号传输的供电

5.2.2.1 远程供电参数

图 6 表示了总线上的远程供电方式。



注：直流电可以选择外置；
直流电可以集成在主站中。

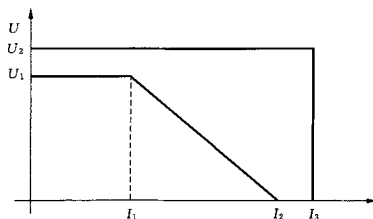
图 6 总线示意图

远程供电的 3 个主要因素是：

- 一个直流供电电源；
- 从站中的储能元件；
- 从站的功耗。

5.2.2.2 供电电源

供电电源按照图 7 的模式向总线供给电压和电流。



特征值：

$$U_1 = 22 \text{ V}, U_2 = 35 \text{ V}$$

$$I_1 = 80 \text{ mA}, I_2 = 250 \text{ mA}, 350 \text{ mA} \leq I_3 \leq 1\,000 \text{ mA}$$

图 7 供电电源特性参数

1 kHz~1 MHz 范围内的纹波噪声峰值小于 10 mV，低于 1 kHz 的纹波噪声峰值小于 100 mV。供电电源本身的特性不在此处描述。

5.2.2.3 从站的储能元件

为了在交换过程中提供峰值电流和优化远程供电电源，每一个从站都配有一个相应的储能元件（最大值为 470 μF ，相对误差为 +20%）来积累电能。

5.2.2.4 从站的功耗

图 8 和图 9 表示出一次交换会话的相关状态。

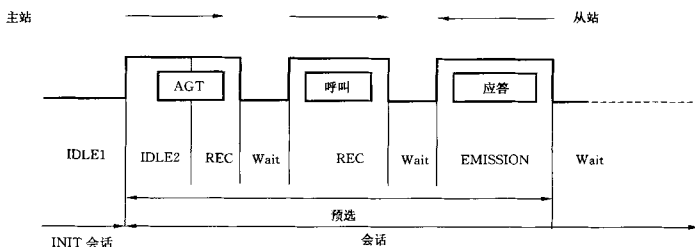


图 8 和一次会话相关的状态:对于被选址的从站

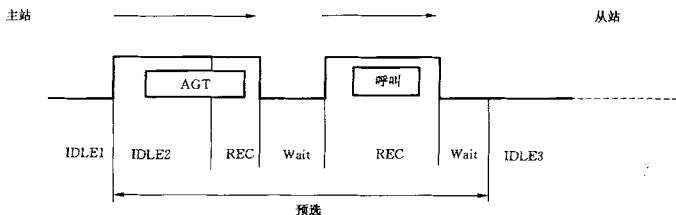


图 9 和会话相关的状态:对于未被选址的从站

各状态下的功耗如下,这些值代表了最大的平均功耗:

- Idle 模式 1 最大 15 mW;
- Idle 模式 2 最大 15 mW;
- Idle 模式 3 最大 15 mW;
- 接收模式 (最大) $40^{(1)} \text{ mW} + n \times \text{最大 } 10^{(1)} \text{ mW}$ (n =设备个数);
- 等待模式 (最大) $40 \text{ mW} + n \times \text{最大 } 10 \text{ mW}$ (n =设备个数);
- 发送模式 (最大) $140^{(1)} \text{ mW} + n \times \text{最大 } 10 \text{ mW}$ (n =设备个数)。

最坏的情况下,传输中 90% 为“0”;

- 接收模式 最大 $50^{(1)} \text{ mW} + n \times \text{最大 } 14^{(1)} \text{ mW}$ (n =设备个数);
- 发送状态 最大 $180^{(1)} \text{ mW} + n \times \text{最大 } 14^{(1)} \text{ mW}$ (n =设备个数)。

对这些不同模式的详细描述(包含相关时序)在状态转换表中给出。

5.2.3 简单从站和复杂从站

一个简单从站只对应一个逻辑地址 ADS。

一个复杂从站对应多个逻辑地址 ADS。

复杂从站的作用是允许对一个辅助总线上的从站中的不同设备分别选址访问。

4) 接收和发送的功耗取决于报文中“0”的个数。这些值是假设报文中的“0”和“1”的个数相同时的平均功耗。

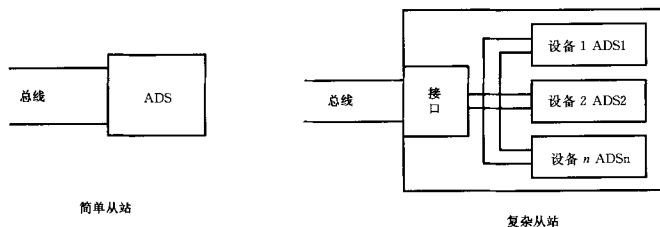


图 10 简单从站和复杂从站

辅助总线结构的基本原理如下：

- 辅助总线的长度不包含在主总线之内。
- 辅助总线有 4 根线，其中 2 根是信号线，另外 2 根是供电线。
- 一个无源从站，不管是简单的还是复杂的，对于物理通信参数来说，它最多只能等效于 2 个从站。
- 除了在调制方面的特征，主总线和辅助总线的协议是相同的：辅助总线上的信号是基带 (1 200 bit/s) 信号线，符合 EIA485 和 ISO/IEC 8482 标准。
- 辅助总线的最大长度为 50 m。
- 辅助总线上的设备数量最多为 6 个。
- 辅助总线的电缆和主总线的一样。
- 如果辅助总线是一直供电的，则可以通过接口从一个设备向主站发出报警信号。
- 这个报文由载波传输 TAB 产生，同时主站的接口和设备必须是处于报警激活模式的。

主总线可以支持如下站点：

- 简单有源从站；
- 简单无源从站；
- 复杂无源从站。

5.3 总线参数特征

5.3.1 特征概述

- 特别支持远程读取和远程编程。总线总是有一个磁接口和至少一个从站。
- 总线的拓扑结构不重要，可以是线型、星型或树型等不闭合的结构，线缆的总长度不超过 500 m (包括供电线缆)。从站内部的辅助总线，不在此长度之内。
- 总线可以为从站供电。总线因此可支持有源和无源站点。
- 总线与接收器和发射器的所有电子设备之间保持电路隔离，其隔离电压的值参考从站所需的标准。
- 总线上的供电可以是永久性的，这样就可以进行报警操作。
- 总线上可以并联 1~100 个从站。总线上的无源站点 (简单或复杂功能的) 的数量最多为 50 个。总线上有源站点的数量最多为 100 个。总线上和从站 (简单或复杂功能的) 相关的设备的数量最多为 50 个。如果一个总线上的从站类型是混合型的，既有无源站点又有有源站点，确定每种类型数量的原则是：若 N_1 是总线上无源复杂从站的数量， N_2 是总线上无源简单从站的数量， N_3 是总线上有源从站的数量，那么必须符合：当电源是集成在主站之内时，则 $2(N_1 + N_2) + N_3 \leq 100$ ；当电源是外置的，直接连接在总线上时，则 $2(N_1 + N_2) + N_3 \leq 98$ 。
- 这些从站中的一个从站偶尔可以处于一个低阻状态 (不发送“0”的传输模式)，这样不会中断通信的进行。
- 和主站的通信经过一个磁插头 (只有 1 个)。

- i) 该总线应能承受与 230 V 电源的偶然连接。试验控制过程将连续 5 次加上 250 V 交流电,每次持续 5 min,每次间隔 5 s。

5.3.2 电缆参数特征

室内电话线类型:

- 双绞线,有铝屏蔽层;
- 导体:直径在(0.5~0.6)mm 的镀锡铜线;
- 绝缘的 PVC。

电气特征:

20℃ 时的直流回路阻抗:(117~192) Ω /km

-15℃~+45℃,交流 50 kHz:

- a) 线性回路阻抗:(154~220) Ω /km
- b) 线性回路电感:(500~800) μ H /km
- c) 线性电容:(80~130) nF /km
- d) 容性损失因数:最大 5%
- e) 芯线对屏蔽线间电容的不平衡度:最大 5%
- f) 综合特征阻抗:(74~115) Ω
- g) 线性相位移(50 kHz):最大 150°/km

以上特性是对带屏蔽隔离的对称源而言,其等效阻抗 Z 和 Z' 在 50 Hz 条件下大于 1 000 Ω 。

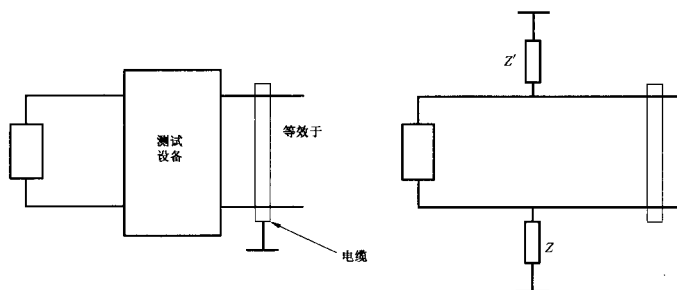


图 11 测试设备等效电路图

5.3.3 连线

- a) 连接从站时应确保屏蔽层的连通性(例如,采用 3 端分线盒)。
- b) 屏蔽层应一点接地,或在某种情况下接等效参考电位。
- c) 除本电缆外,没有小于 1 000 Ω (50 kHz)的阻抗连接在总线和屏蔽层或地之间。

注:对于使用与上述要求稍有不同线缆,应注意:

- 容性较大、阻抗较高的电缆,应减小其长度。长度应与阻抗呈反比。
- 容性较小、阻抗较小的电缆,在总线较空时可能引起对接收器输入端过压。这个问题可以采用在磁插头的对端附近,在总线之间连接一个阻尼电阻((330~1 000) Ω , 0.25 W, 根据过压率来选择)来解决。为了确保能承受 230 V 电压,可采用一个耐压合适的 47 nF 的电容,把它和这个电阻串联在一起。

5.4 磁插头

5.4.1 功能

5.4.1.1 简单磁插头

磁插头包含一个移动的插头(主站)和一个固定的插座(从站)。

当两部分连接之后,磁插头在 HHU(连接在插头上)和总线(连接在插座上)之间双向地传输信号。

每个部分各是铁氧体变压器的一半,在磁回路中留有气隙。

为了对这个变压器进行高串联电感和低并联电感的补偿,需要在每侧连 1 个谐振电容和 1 个阻尼电阻,将其转变成一个 4 阶的带通滤波器,中心频率为 50 kHz,品质因数 $Q < 3$ 。

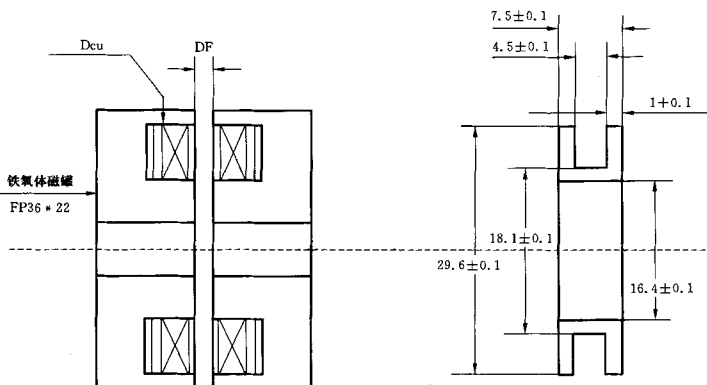
这样可以采用简单的方波传输,并可以消除频率的瞬变。

5.4.1.2 带电源的磁插头

除上述特点以外,带电源的磁插头还可以传输(400~600) kHz 的信号。默认值定在 500 kHz,这个信号在对总线注入直流信号之前先进行整流和滤波。

5.4.2 一般机械特征

单位:mm



FP——铁氧体;

DF——铁氧体之间的空隙;

Dcu——磁芯的直径。

图 12 铁氧体和线圈

磁插头的每一半都包含半个铁氧体的半个磁罐和里面放的线圈架组成,外面包着坚固的塑料盒。当插合时,这一对线圈与磁心基本是同心的,对于中界面基本是对称的。而且:

- 磁气隙 $DF = (4.25 \pm 0.25) \text{ mm}$;
- 同心度: 0.25 mm。

组装连接时,成对的磁心和线圈必须固定好并压紧。比如,考虑到公差或尺寸关系,在线圈架的背部,线圈架与磁心间应留有轴向间隙。

铁氧体是标准型号的,使用频率低于 100 kHz:

- 初始的磁导率超过 1 800 Gs;
- 当频率为 100 kHz 时,其介质损耗系数 $\tan \delta$ 最大值约为 2%;
- 由于漏磁较大,因此不易饱和(实际上,0.4 T 饱和)。

5.4.3 简单插头的电气图

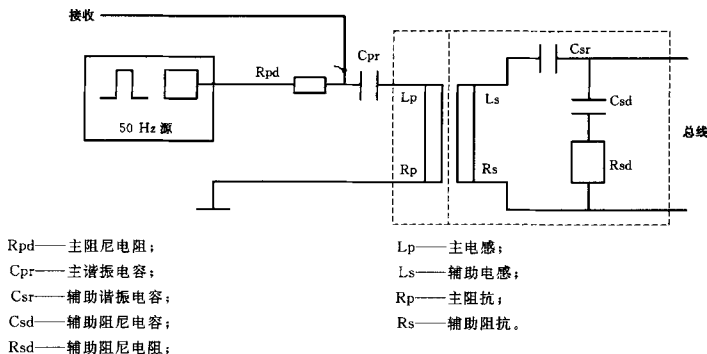


图 13 磁插头的相关器件

5.4.3.1 总线侧插座的相关器件

- 和 L_s 谐振的串联电容 C_{sr} ;
- 并联阻尼电阻 R_{sd} ;
- 并联电容 C_{sd} 和 R_{sd} 串联, 确保能耐受对 230 V 电源的意外联接。

5.4.3.2 插头侧的相关器件

- 和 L_p 谐振的串联电容 C_{pr} 。
- 串联阻尼电阻 R_{pd} (取决于 50 kHz 的源, 包含所有的 50 kHz 的串联电阻, 如软线的、连接器的等等)。
- 50 kHz 的电源, 如方波源。
- 接收电路、解调和方波形成电路, 与 C_{pr} 和 R_{pd} 间的节点 (如全波整流和阈值电路等) 连接。

处于接收模式时, 50 kHz 源的输出阻抗, 即使是低电压时, 也应当保持在几欧姆以上, 以确保主电路的阻尼。

5.4.4 带电源的插头的电气图

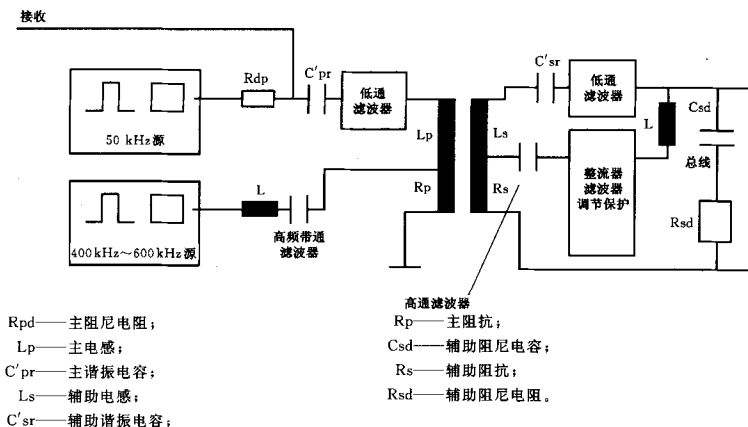


图 14 供电插头的相关器件

除以下情况外,50 kHz 部分的工作与简单插头电路类似:

- 由于两个低通滤波器的存在,使两路均产生附加衰减。采用 50 kHz 电源的接收设备,必须考虑这个衰减。
- 由于当 50 kHz 时,低通滤波器阻抗的影响,在 C'_{pr} 、 C'_{sr} 与 C_{pr} 、 C_{sr} 之间会稍有差别。

5.5 (50 kHz 信号的)主站发送器的功能特点

主站发送器由主站发送源和磁插头组成。

在整个温度范围内,总线上输出的信号应符合如下限制(见图 5):

- $T_{ev1}=750\ \mu\text{s}$, $T_{ev0}=750\ \mu\text{s}$ 。
- $U_{evh1}=0.25\ \text{V}$ 。

所指信号的频率在 1 kHz~1 MHz 之间。

总线输出开路时:

- $U_{evl0}=5.8\ \text{V}$ 。
- $U_{evh0}=7.5\ \text{V}$ 。

以 $100\ \Omega$ 电阻代替总线时:

- $U_{evl0}=4\ \text{V}$ 。
- $U_{evh0}=4.8\ \text{V}$ 。

以 $31.8\ \text{nF}$ 电容代替总线时:

- $U_{evl0}=5.2\ \text{V}$ 。
- $U_{evh0}=6.5\ \text{V}$ 。

此外,输出端应该开路,或匹配一个 $100\ \Omega$ 的电阻,

或一个 $31.8\ \mu\text{F}$ 的电容。

- 总线上传输的噪声,在各种状态下,各种不超过 1 MHz 的频率下,在瞬态消失后,不应超过峰值 $10\ \text{mV}$ (1 kHz~1 MHz 之间)。
- 由于从发送模式转成接收模式(反之亦然)的开关造成的过压尖峰,在各种条件下不超过峰值 $0.25\ \text{V}$ 。

以上这些值是在磁间隙 $DF=(4.25\ \text{mm}\pm 0.25\ \text{mm})^{+0.15}_{-0}\ \text{mm}$ 的条件下得到的。

5.6 (50 kHz 信号的)主站接收器的功能特点

主站接收器是由主站接收电路(解调和方波产生)和磁插头组成的。

接收器应在整个温度范围内能对正弦输入信号(其特征上文已经阐述过)正确接收(正确接收指帧的重复率 $<10^{-5}$)。

信号应通过一个串联阻抗连接到磁插头的总线端子上,阻抗值如下:

- $T_{ev0}=700\ \mu\text{s}$, $T_{ev1}=700\ \mu\text{s}$ 。

当通过一个 $100\ \Omega$ 的电阻时:

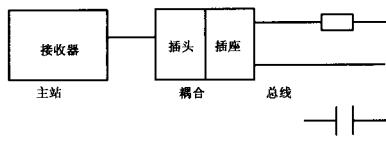
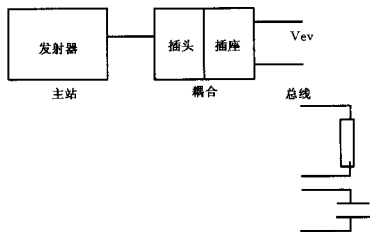
- $U_{evh1}=0.25\ \text{V}$ 。
- $U_{evl0}=0.7\ \text{V}$ 。
- $U_{evh0}=3.2\ \text{V}$ 。

当通过一个 $31.8\ \text{nF}$ 的电容时:

- $U_{evh1}=0.20\ \text{V}$ 。
- $U_{evl0}=0.55\ \text{V}$ 。
- $U_{evh0}=2.5\ \text{V}$ 。

另外,接收器在条件 b) 的情况下(串联一个 $100\ \Omega$ 的电阻)能进行正确接收(正确接收指帧重复率 $<10^{-5}$)的条件是:

- 频率 1 kHz~1 MHz,峰值为 $0.1\ \text{V}$ 的连续波形。



- i) 幅度为 20 V,持续时间为 5 μ s 的方波脉冲。
- j) 幅度为 3.5 V,持续时间为 200 μ s 的方波脉冲。

5.7 (50 kHz 信号的)从站发送器的功能特点

在整个温度范围内,传输到总线上的信号必须满足以下的要求:

- a) $T_{ev0}=750\ \mu\text{s}$, $T_{ev1}=750\ \mu\text{s}$ 。
- b) $U_{evh1}=0.1\ \text{V}$ 。

这只是为了频率为 1 kHz~1 MHz 的信号而言的。

当总线上放置 100 Ω 的电阻时:

- c) $U_{evl0}=1.2\ \text{V}$ 。
- d) $U_{evl0}=1.8\ \text{V}$ 。

当总线上放置 31.8 nF 的电容,测量和这个电容串联在一起的 1 Ω 电阻上的输出信号,并将结果乘以 100:

- e) $U_{evl0}=1.5\ \text{V}$ 。
- f) $U_{evh0}=2.5\ \text{V}$ 。

另外,当总线端子两侧连接一个 100 Ω 的电阻或 31.8 nF 的电容时:

- g) 由于从发送模式转成接收模式(反之亦然)的开关造成的过压尖峰,在各种条件下不超过峰值 0.75 V。
- h) 总线上传输的噪声,在各种条件下,各种不超过 1 MHz 的频率下,在瞬变消失后,不应超过峰值 10 mV(1 kHz~1 MHz 之间)。

另外:

- i) 最大的短路电流:峰值 26 mA,50 kHz。
- j) 发射器应能承受长时间的短路和连接到 230 V 上的情况。
- k) 在总线输入和其他设备(包括从站)的输入之间的最大的共模电容固定在 15 pF。

5.8 (50 kHz 信号的)从站接收器的功能特点

接收器应在整个温度范围内对正弦输入信号(其特征上文已经阐述过)正确接收(正确接收的含义是指误码率 $<10^{-5}$):

- a) $T_{ev0}=700\ \mu\text{s}$, $T_{ev1}=700\ \mu\text{s}$ 。

带有一个和接收器的输入阻抗相比,其内阻可以忽略的电压产生器时:

- b) $U_{evh1}=0.3\ \text{V}$ 。
- c) $U_{evl0}=2\ \text{V}$ 。
- d) $U_{evh0}=8\ \text{V}$ 。

另外,接收器对以下信号应不敏感:

- e) 1 kHz~1 MHz 之间的峰值为 0.25 V 的持续的信号。
- f) 宽度为 5 μ s 的 20 V 的脉冲信号。

50 kHz 的输入阻抗:

- g) 当电压高达 5 V 时,不论接收器是工作着的还是没有工作着的,都应有一个电阻与一个电容相并联。其中,有源和无源站点也必须考虑进去:

- 对于一个有源站点来说:

- 1) 电阻 $>20\ \text{k}\Omega$;
- 2) 感性时感抗 $>20\ \text{k}\Omega(60\ \text{mH})$;
- 3) 容性时容抗 $>100\ \text{k}\Omega(30\ \text{pF})$ 。

- 对于一个无源站点来说:

- 1) 电阻 $>10\ \text{k}\Omega$;

- 2) 感性时感抗 $>10\text{ k}\Omega$ (30 mH);
- 3) 容性时容抗 $>50\text{ k}\Omega$ (60 pF)。
- h) 在峰值超过 5 V 时内部可以箝位,当电压高于箝位电压时,动态输入阻抗 $>200\text{ }\Omega$ (50 kHz)。
- i) 在 50 kHz 时,传输逻辑 1(总线上没有信号)的最小的输入阻抗:200 Ω 。
- j) 接收器能承受在总线端子上长时间加 230 V 电压。
- k) 在总线输入和其他输入之间的最大的共模电容,对于一个有源站点:15 pF;对于一个无源站点:100 pF。

注意:所有的用于测量的阻抗的精度为 1%。

附录 A

(规范性附录)

规范语言

A.1 词汇和操作规则

为了清楚地阐述出本地总线数据交换结构中的各层所担当的角色,技术规范中采用了一种表格的形式来规定出一个状态数量有限的控制器的行为。

每一个控制器对应一个唯一的逻辑表格,如果这个逻辑表格很大的话,它可以分解成若干个物理表格。

每一个控制器的事件对应一个参考控制器逻辑表格的实例特征活动的拷贝。

每一个物理表格包含若干状态行,每个状态行描述了设备在触发条件(第2列)下,执行一系列行动(列3),由一个初始状态(列1)转换到一个终结状态(列4)。

第一个初始状态是控制器的启动状态。这个状态是唯一的,用粗体字的形式来特别地表示。

控制器的停止状态是一个终结状态,因此不能作为任何状态行的初始状态。若一个控制器没有停止状态,则它是无休止运行的;一个有休止的控制器有一个或多个停止状态。这些状态也用粗体字来表示。这种约定表示,包含在物理表格的状态的出现顺序是无关紧要的。

当若干个状态行对应同一个初始状态时采用相同的约定,这是由于它们的触发条件是相互紧密关联的。这样一来,物理表格中的状态行的顺序只是出于表现形式的考虑而排列的。虽然如此,逻辑上,往往从初始状态的传送的描述开始。

在一个状态行里的行动集必须严加考虑(也就是说是不可打断的行动序列),这里描述的行动必须按照所列出的顺序一一执行。一个行动用一种命名的过程函数的形式来定义,括号中带有有一个或多个参数,或没有参数。所有这些被命名的过程函数应该有独立的描述。另外,有两个预先定义的过程: `assignment` 和 `空过程 none()` (无任何行动)。

和某个状态行相关的触发条件可能包含若干个子条件。对一个复合型的触发条件进行判断时总是要涉及到对其包含的所有子条件的判断。这样一来,子条件书写的顺序就无关紧要了。

用于表示复合条件的运算符是逻辑运算符 `&` (逻辑与)、`|` (逻辑或)、`not()` (逻辑非) 和比较运算符 `<`、`>`、`<=`、`>=`、`<>`。

共有两种类型的触发条件:

简单条件根据定义,是立即就判断的。如果它是复合型的,其所有的子条件都必须是简单类型的。一个布尔函数就是一个简单条件的例子。所有这些被命名的布尔函数应该有独立的描述。

事件条件表示在等待一个事件的发生。它可能会包含若干个事件或简单子条件。

当一个触发条件的判断结果为真时,条件就满足了,触发条件的满足就导致状态进行转换。

一个事件可以定义为为了满足一个事件型触发条件的元素。

当一个事件,其包含的事件型的触发条件满足之后,它就自动完成而结束了。一个事件只能结束一次。

控制器要完成处于无法处理的状态的事件时,这些事件将被逐一地存放在一个称为交互控制器队列的区域中。

这样一来,每个控制器有一个和控制器本身事件共享的简单队列,队列的空间大小假设是无穷大的,其组织和管理方式在这里不做描述。应当注意的是,对于任何一个从启动状态开始的转换,会对队列进行局部的整理工作(只是针对于和当前控制器本身事件相关事件的队列)。

另外,有一个自整理机制,用于自动删除那些明显的无法完成的外来事件。而且,有一个预定义的

名称为 \$purge() 的过程, 对应于对当前的交互控制器队列进行全面整理。相应控制器的所有的事件则返回启动状态。

事件是一些描述过的行动产生的, 用一套行动集在相应的状态行中表示。一个内部的事件, 只能被产生它的控制器来完成, 而一个外部事件只能是由产生它之外的控制器来完成。

应当注意, 一个不发生的事件(用逻辑运算符 NOT() 来表示)总是一个简单的子条件。

对于初始化状态, 当有一个状态行其触发条件是某种类型的(简单或事件), 那么, 所有的具有相同初始化状态的状态行也具有相同的触发条件。

当这个类型是简单类型时, 初始化状态称为子状态。子状态用斜体字形式来特别写明。它是瞬时的, 而且总是可以移动的。它在物理表格中的出现只能被表示的进一步澄清所证明。对于特殊的例子, 启动子状态, 一个特殊的条件被预定义了: \$true(), 总是真。

针对每一个控制器事件, 物理表格中的触发条件和行动中的变量保持为局部变量。还有一个预定义的变量(无关联变量)用于代替在任何函数和过程中的任何没有使用的参数。

A.2 实体和实体事件

在这里描述的规范语言和 OSI(开放系统交互连接)中的概念是并行的, 这一点非常有趣。

例如, 对于每一层, 实体对应控制器, 实体事件对应控制器事件。

附录 B

(规范性附录)

时序类型和特征

B.1 时序类型定义

时序分类为以下几种类型：

- 逻辑时序：TL 类型

定义从停止位到起始位。

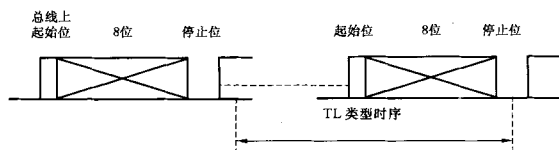


图 B.1 逻辑时序类型

- 物理时序：TPFD 或 TPDF 类型

定义为从载波的结束到载波的开始 (TPFD)，从载波的开始到载波的结束 (TPDF)。

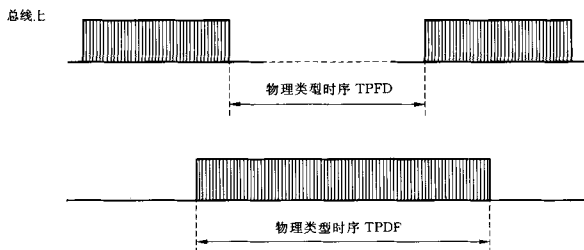


图 B.2 物理时序类型

- 半逻辑时序：TSL1 类型

从载波或事件的停止到停止位。

- 半逻辑时序：TSL2 类型

从事件的停止到起始位。

- 时间时序：Tc 类型

由事件触发，根据编程的延时时间自动停止。

- 特定时序：Ta 类型

依靠总线的供电；TICB 是 Ta 类型。

B.2 时序的测量和特征

时序的精度为 $\pm 1\%$ 。最小值不能小于 10 ms。

在数组中的每个时序限定值应考虑到这点。结果用以下方式处理：

- a) 高限－允许偏差 $>M>$ 低限＋允许偏差, 结果是正确的。
- b) $M<$ 低限－允许偏差, 结果是不正确的, M 应为测量值。
- c) $M>$ 低限＋允许偏差, 结果是不正确的。
- d) (低限－允许偏差) $<M<$ (低限＋允许偏差), 结果不确定, OK 或 NOK。
- e) (高限－允许偏差) $<M<$ (高限＋允许偏差), 结果不确定, OK 或 NOK。

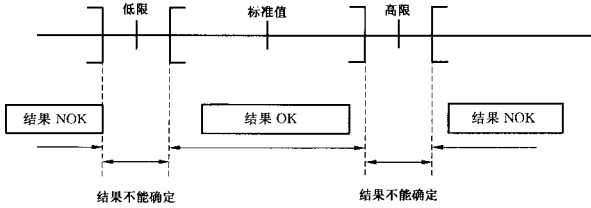


图 B.3 定义有低限和高限的时序的结果处理

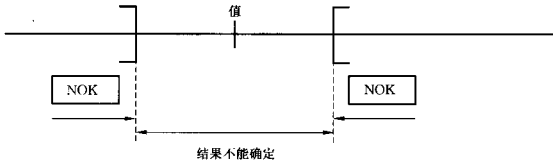


图 B.4 定义为标准值的时序的结果处理

附 录 C
(规范性附录)
致命错误清单

表 C.1 中列出的任何一个致命错误的发生将在本地上传给应用层。

表 C.1 致命错误的错误号

编号	1	2	3	4	5	6	7	8
错误	EP-3F	EP-4F	EP-5F	EL-1F	EL-2F	EA-1F	EA-2F	EA-3F

不论涉及到哪一层,发生致命错误会导致:

- 适当的话,通过服务基元 abort.req 停止下一个低层;
- 适当的话,通过服务基元 abort.ind 连带出错号一并通知下一个高层;
- 对相应处理器事件进行完整的重新初始化操作。

附 录 D
(规范性附录)
帧中命令码的编码

D.1 不带 DLMS 的本地总线数据交换的命令码**表 D.1 不带 DLMS 的命令码**

缩写	16 进制值	2 进制值	作 用
ENQ	01	0000 0001	请求远程读取
DAT	02	0000 0010	远程读取应答
REC	03	0000 0011	请求远程编程
ECH	04	0000 0100	在远程编程中的数据回应
AUT	05	0000 0101	认证命令
EOS	06	0000 0110	结束远程编程
ASO	07	0000 0111	遗漏站点呼叫
RSO	08	0000 1000	遗漏站点应答
IB	09	0000 1001	总线初始化
DRJ	0A	0000 1010	拒绝远程编程数据
ARJ	0B	0000 1011	拒绝远程编程认证
TRF	0C	0000 1100	点对点远程传输
TRB	0D	0000 1101	广播式远程传输(无应答)
TRA	0E	0000 1110	点对点远程传输应答
PRE	10	0001 0000	选址无源站点
SEL	11	0001 0001	无源站点选址应答

D.2 带 DLMS 的本地总线数据交换的命令码

为了避免不带 DLMS 结构的 DATA+帧发生混淆,DATA+,优先级,发送和确认域组成一个特殊的命令码 COM,其取值从保留的 COM 值之外选择。

表 D.2 带 DLMS 的命令码

缩写	16 进制值	2 进制值	作 用
ND1	E0	1110 0000	正常数据(优先级=0),序列号=0000
ND2	E3	1110 0011	正常数据(优先级=0),序列号=0011
ND3	EC	1110 1100	正常数据(优先级=0),序列号=1100
ND4	EF	1110 1111	正常数据(优先级=0),序列号=1111
UD1	F0	1111 0000	正常数据(优先级=1),序列号=0000
UD2	F3	1111 0011	正常数据(优先级=1),序列号=0011
UD3	FC	1111 1100	正常数据(优先级=1),序列号=1100
UD4	FF	1111 1111	正常数据(优先级=1),序列号=1111

附 录 E

(规范性附录)

CRC 原理

对传输位的正确性的检验是通过组织一个用字节来表示的信息块(参考数据链路层)来实现的。检验关键字是一组被称之为检验域的位串信息。其计算方法是基于循环编码的原理,即使用多项式的除法和多项式除法余数的代数运算。

E.1 多项式的运算

若 $A = (a_1, a_2, \dots, a_n)$ 是要进行计算的位串,这个位串可以看作是一个 $n-1$ 阶的多项式:

$$A(X) = a_1 X^{n-1} + \dots + a_{n-1} X + a_n$$

找一个 m 阶的多项式 $D(X)$ 为除数,则多项式的除法 $A(X) \times X^m / D(X)$ 等效如下:

$$A(X) \times X^m = D(X) \times Q(X) + R(X)$$

这里 $R(X)$ 是一个阶小于或等于 $m-1$ 的多项式,等效于位串 $R = (r_1, r_2, \dots, r_m)$

进行布尔逻辑运算,等效于:

$$A(X) \times X^m - R(X) = A(X) \times X^m + R(X) = D(X) \times Q(X)$$

代表位串 $(a_1, a_2, \dots, a_n, r_1, r_2, \dots, r_m)$

E.2 检验过程

CRC(循环冗余校验)的校验域用位串 $R = (r_1, r_2, \dots, r_m)$ 来表示。其计算程序是基于移位寄存器和累加寄存器的,从而在数据到达时能计算出校验码。其计算算法这里不再描述。

根据其原理,数据发送方须将位串 $R = (r_1, r_2, \dots, r_m)$ 和要进行保护的位串 $A = (a_1, a_2, \dots, a_n)$ 放在一起,发送 $(a_1, a_2, \dots, a_n, r_1, r_2, \dots, r_m)$ 到接收方。

当接收方收到位串是一个能被 $D(X)$ 整除的 $m+n-1$ 阶多项式时,认定无传输错误发生。

E.3 计算参数

具体实现该算法时,选择的参数是:

m	16
$D(X)$	$X^{16} + X^{15} + X^2 + 1$

附录 F

(规范性附录)

用于遗漏站点应答的随机整数的产生

遗漏站点呼叫交换过程中,为了管理多个时间片,需要产生和处理范围在 $[0, \text{MaxRSO}]$ 之间的随机整数。

F.1 随机整数的标准

下述标准的选择只是针对于随机数,而不是说明一个具体的解决方案和算法。

“对于一个在 $[0, n]$ 范围内的一个整数 I ,当其出现的概率总是在 $[100/n - D, 100/n + D]$ 的范围时,在相对短的 T 时间里提供有效数字 N 。这个整数被认为是随机的。”

F.2 操作参数

对于处理“遗漏站点呼叫”的时间片 RSO ,其最大值 MaxRSO 设置为 3。在这个条件下, N 、 T 、 D 参数的选择参照下表:

N	T	D
≤ 100	$\leq 10 \text{ min}$	≤ 7

附 录 G

(规范性附录)

用于电文辨认过程的随机数的产生(不带 DLMS 的结构)

根据 DES(数据加密标准)标准算法,认证交换过程需要产生和处理 64 位的随机数。

以下两个准则是描述实现控制的,而不是具体解决方案:

准则 1:海明间距

随机数 $N_{Ai}(k)$,由主站($i=1$)或从站($i=2$)产生,应该有一个和先前的随机数之间大于 4 的海明间距 D_h ,这里在总线上观测的次数 k 小于 400。

它由如下公式产生:

对于第 k 次观测, $D_h[N_{Ai}(n+k), N_{Ai}(n+k-i)] > 4$, 其中, $0 < i < k$ 。

最大的连续观测次数 k 为 400,观测从 $k=0$ 开始,第一个随机数值为 $N_{Ai}(n)$ 。

观测的发生至少有 1 s 的延时。

准则 2:位分布的概率

对于上述 400 个随机数中的行 i 的全部的位的范围,0 或 1 出现的概率在 0.35~0.65 之间。

它由如下公式产生:

$$0.35 < [\Pr(\text{BitVal } 2^i) = 0] < 0.65 \text{ 对于 } 0 \leq i \leq 63$$

$$0.35 < [\Pr(\text{BitVal } 2^i) = 1] < 0.65 \text{ 对于 } 0 \leq i \leq 63$$

附 录 H
(规范性附录)
系统管理的实现

为确保最大限度的兼容性(对于带或不带 DLMS 的站点),建议采用遗漏站点呼叫机制来实现系统管理。

发现请求	发现请求应答
在 TABi 域中有 2 个字节的 ASO 帧(第 1 个字节对应于保留的 TAB0, 第 2 个字节对应应答概率值)	包含系统标题(ADS)的标准 RSO 帧
过程或函数(主站)	定 义
Discover. request (energized, adp, response-probability)	energized: 发现新的有源站点为真, 发现新的无源站点为假 ADP: 主站的物理地址 Response-probability: 应答的请求概率
Discover. confirm(collision, discover-list)	Collision: 若发生冲突为真, 否则为假 Discover-list: 新发现站点的 ADS 清单

在从站, 被用来应答发现服务请求的随机整数范围在 $[0, 100]$ 之间。随机函数应有一个 $\pm 10\%$ 的随机波动特性, 在该条件下, 对于系统的一个有效数字 $N(N > 10)$, 报告系统数字应该为: $(\text{应答概率} \times N)/100$ 。

附 录 I
(资料性附录)
交换的有关信息

1.1 无源站点的会话

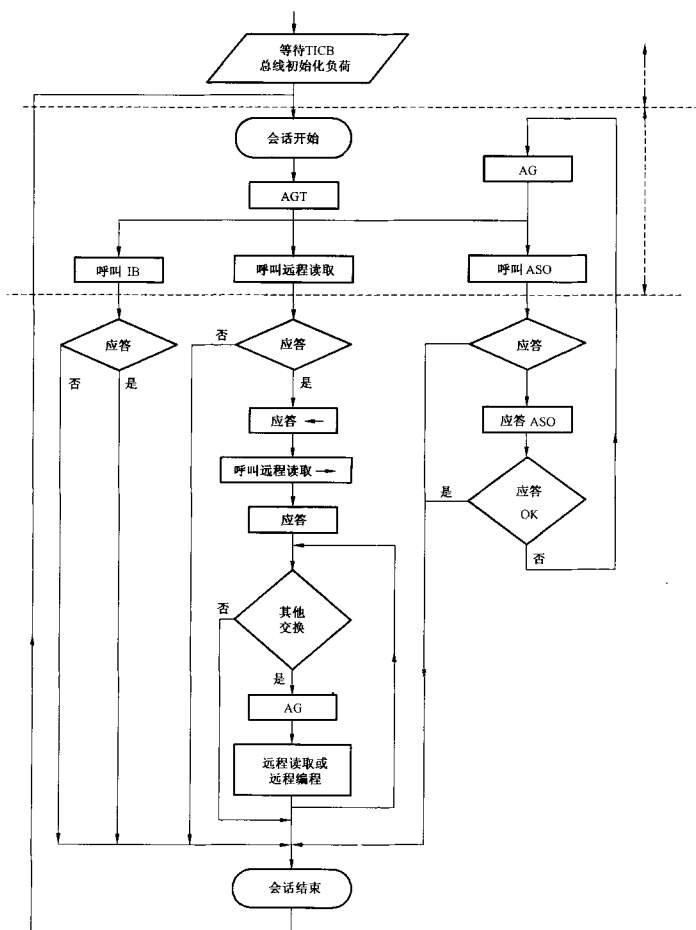


图 I.1 无源站点会话

I.2 远程读取和编程交换

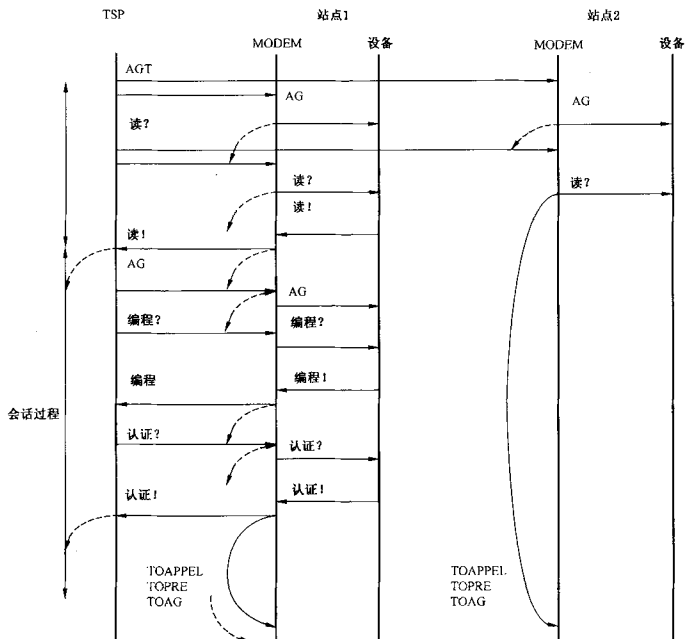


图 I.2 远程读取和编程交换

主站发送一个 AGT“唤醒呼叫”信号给无源站点。经过从站的 MODEM 过滤后,从站接收到一个 AGN 信号,初始化 TOAPPEL 计时器(选址帧的最大等待时间)。

主站发送一个远程读取帧给站 1。从站 2 不应答,并且在 TOPRE 时间后返回到低功耗状态。从站 1 应答:它被选址。这个例子是一个远程读取和一个远程编程交换的一系列的动作。会话由唤醒 TOAG 来检查。

I.3 总线初始化帧

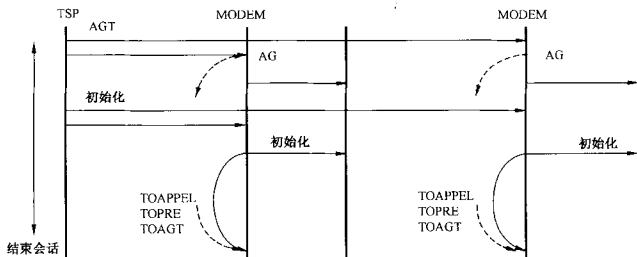


图 I.3 总线初始化

主站发送一个 AGT“唤醒呼叫”信号和一个总线初始化帧。所有的无源从站被唤醒。IB 帧不会引起任何的应答,所以,之后所有的站点会返回低功耗状态。

I.4 遗忘站呼叫交换

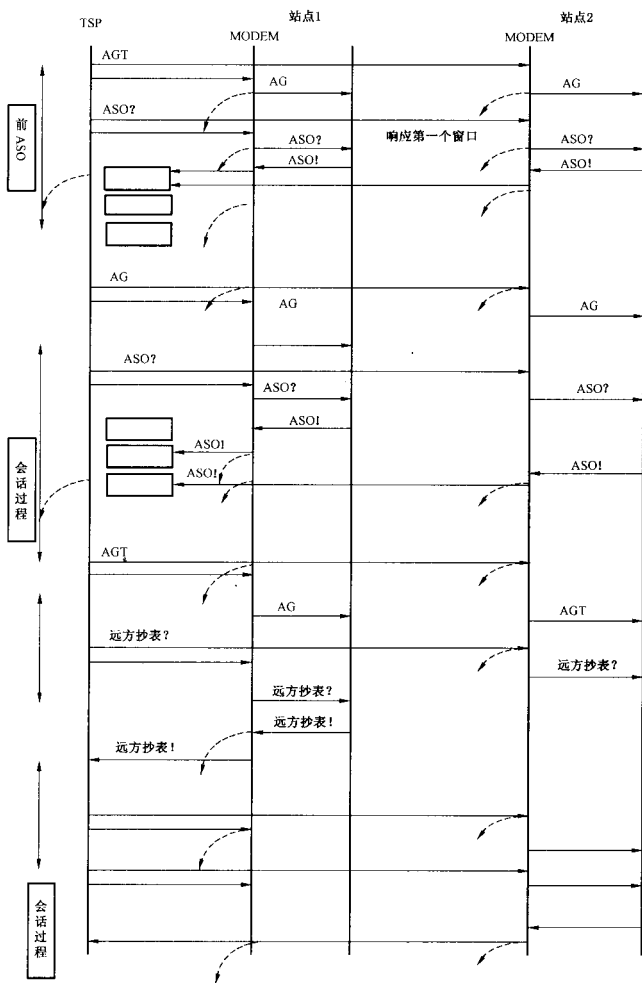


图 I.4 遗忘站呼叫交换

在这种情况下,两个从站成为“遗忘站”。在最初的遗忘站呼叫请求(预选请求),第一个时间片内两个遗忘站都回答。在接下的遗忘站呼叫交换中,1号站在第二个时间片回答,而2号站选择在第三个时间片回答。