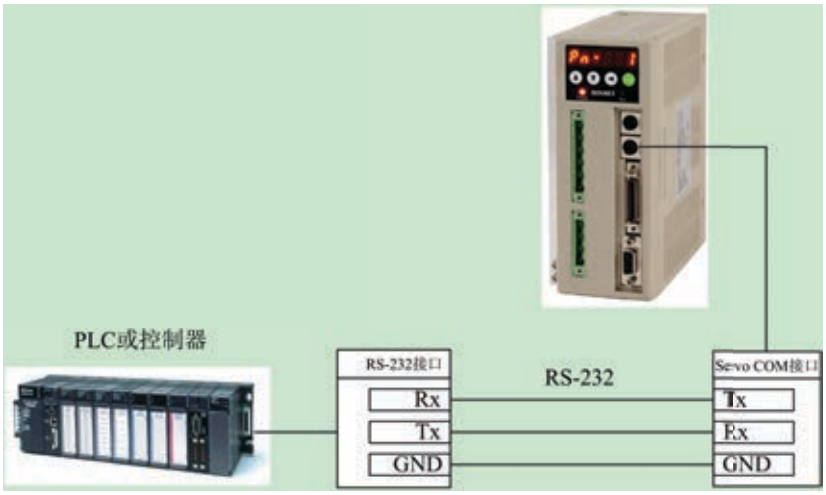
第七章 通讯功能

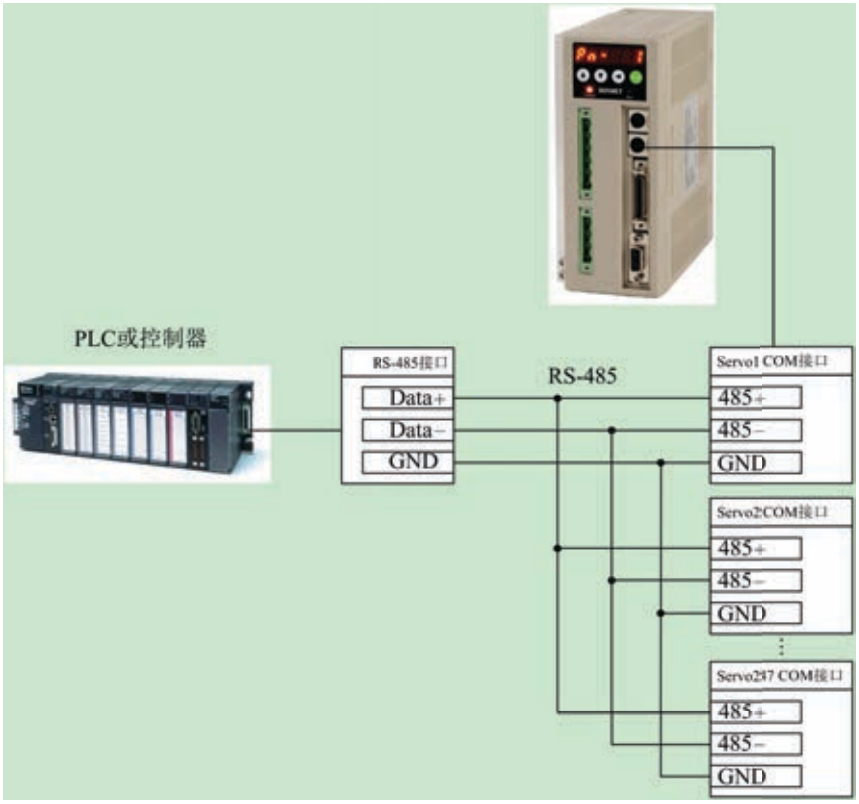
7.1 RS-232，RS-485,CANOpen 通讯硬件界面

BONMET 伺服驱动器具有 RS-232、RS-485 的串行通讯功能和 CANOpen 总线控制功能，使用此功能可驱动伺服系统、修改参数以及监视伺服系统状态等多项功能。其接线说明如下：

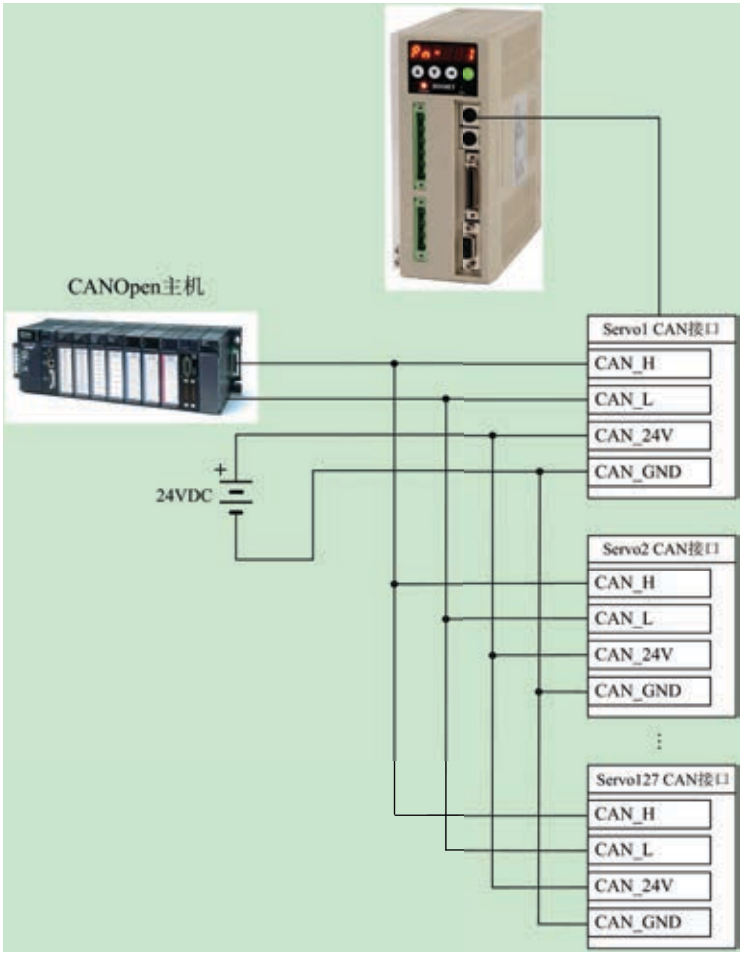
- **RS-232 接线图**



- **RS-485 接线图**



● CANOpen 接线图



7.2 通讯模式选择

通过定义 0x12F7 参数，选择 ModBus 通讯端口，伺服驱动器的通讯地址需藉由 0x12F8 参数来设定不同的伺服驱动器站号。设定 0x12F9 参数可以设置本驱动器的通讯速率，选择通讯速率需 PC 控制器与驱动器的通讯速度一致。

7.3 ModBus 通信协议简介

使用 RS-232/485 总线串联通讯时，每一台驱动器必须预先在参数 Pn185 上设定其伺服驱动器站号，计算机将根据站号对每台伺服驱动器实施控制。通讯方法是使用 ModBus 总线通讯，博美德伺服系统使用 RTU(Remote Terminal Unit)模式，以下为 ModBus 通讯说明。

7.3.1 编码意义

RTU(Remote Terminal Unit)模式下，报文中每个 8 位字节含有两个 4 位十六进制字符。这种模式的主要优点是较高的数据密度，在相同的波特率下比 ASCII 模式有更高的吞吐率。每个报文必须以连续的字符流传送。

RTU 模式每个字节 （ 11 位 ） 的格式为:

代码系统: 8 位二进制

报文中每个 8 位字节含有两个 4 位十六进制字符(0 - 9， A - F)

每个字节的位: 1 起始位

8 数据位， 首先发送最低有效位

1 位作为奇偶校验

1 停止位

偶校验是要求的， 其它模式（ 奇校验， 无校验 ） 也可以使用。 为了保证与其它产品的最大兼容性，同时支持无校验模式是建议的。默认校验模式模式 必须为偶校验。

注 ： 使用无校验要求 2 个停止位。

字符是如何串行传送的:

每个字符或字节均由此顺序发送(从左到右):

最低有效位 (LSB) . . . 最高有效位 (MSB)

图 8-X RTU 模式位序列

设备配置为奇校验、偶校验或无校验都可以接受。如果无奇偶校验，将传送一个附加的停止位以填充字符帧:

图 8-X RTU 模式位序列（无校验特殊情况）

帧检验域: 循环冗余校验 (CRC)

帧描述:

图 8-X RTU 报文帧

7.3.2ModBus 报文帧

由发送设备将 Modbus 报文构造为带有已知起始和结束标记的帧。这使设备可以在报文的开始接收新帧，并且知道何时报文结束。不完整的报文必须能够被检测到而错误标志必须作为结果被设置。在 RTU 模式，报文帧由时长至少为 3.5 个字符时间的空闲间隔区分。在后续的部分，这个时间区间被称作 t3.5。

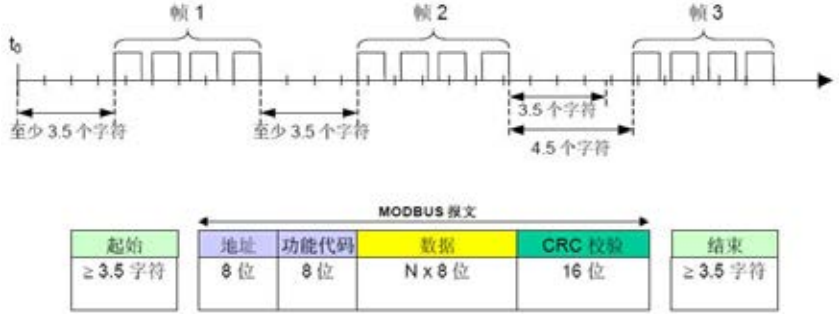
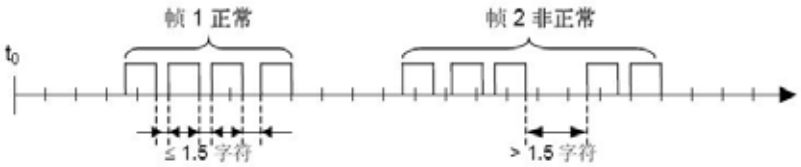


图 8-x RTU 报文帧

整个报文帧必须以连续的字符流发送。如果两个字符之间的空闲间隔大于 1.5 个字符时间，则报文帧被认为不完整应该被接收节点丢弃。



RTU 接收驱动程序的实现，由于 t1.5 和 t3.5 的定时，隐含着大量的对中断的管理。在高通信速率下，这导致 CPU 负担加重。因此，在通信速率等于或低于 19200 Bps 时，这两个定时必须严格遵守；对于波特率大于 19200 Bps 的情形，应该使用 2 个定时的固定值：建议的字符间超时时间(t1.5)为 750μs，帧间的超时时间 (t1.5) 为 1.750ms。

7.3.3 CRC 校验

在 RTU 模式包含一个对全部报文内容执行的，基于循环冗余校验 (CRC - Cyclical Redundancy Checking) 算法的错误检验域。CRC 域检验整个报文的内容。不管报文有无奇偶校验，均执行此检验。

循环冗余校验 (CRC) 域为两个字节，包含一个二进制 16 位值。附加在报文后面的 CRC 的值由发送设备计算。接收设备在接收报文时重新计算 CRC 的值，并将计算结果于实际接收到的 CRC 值相比较。如果两个值不相等，则为错误。

CRC 的计算, 开始对一个 16 位寄存器预装全 1. 然后将报文中的连续的 8 位子节对其进行后续的计算。只有字符中的 8 个数据位参与生成 CRC 的运算，起始位，停止位和校验位不参与 CRC 计算。

CRC 的生成过程中， 每个 8 - 位字符与寄存器中的值异或。然后结果向最低有效位 (LSB) 方向移动(Shift) 1 位，而最高有效位 (MSB) 位置充零。 然后提取并检查 LSB：如果 LSB 为 1， 则寄存器中的值与一个固定的预置值异或；如果 LSB 为 0， 则不进行异或操作。这个过程将重复直到执行完 8 次移位。完成最后一次（第 8 次）移位及相关操作后，下一个 8 位字节与寄存器的当前值异或，然后又同上面描述过的一样重复 8 次。当所有报文中子节都运算之后得到的寄存器中的最终值，就是 CRC.

生成 CRC 的过程为:

1. 将一个 16 位寄存器装入十六进制 FFFF (全 1). 将之称作 CRC 寄存器.
2. 将报文的第一个 8 位字节与 16 位 CRC 寄存器的低字节异或，结果置于 CRC 寄存器.
3. 将 CRC 寄存器右移 1 位 (向 LSB 方向)， MSB 充零. 提取并检测 LSB.
4. (如果 LSB 为 0): 重复步骤 3 (另一次移位).
   (如果 LSB 为 1): 对 CRC 寄存器异或多项式值 0xA001 (1010 0000 0000 0001).
5. 重复步骤 3 和 4，直到完成 8 次移位。当做完此操作后，将完成对 8 位字节的完整操作。
6. 对报文中的下一个字节重复步骤 2 到 5，继续此操作直至所有报文被处理完毕。
7. CRC 寄存器中的最终内容为 CRC 值.
8. 当放置 CRC 值于报文时，如下面描述的那样，高低字节必须交换。

7.3.4 通讯参数的写入与读出

■ 03 (0x03)读保持寄存器

在一个远程设备中，使用该功能码读取保持寄存器连续块的内容。请求 PDU 说明了起始寄存器地址和寄存器数量。从零开始寻址寄存器。因此，寻址寄存器 1-16 为 0-15。

将响应报文中的寄存器数据分成每个寄存器有两字节，在每个字节中直接地调整二进制内容。

对于每个寄存器，第一个字节包括高位比特，并且第二个字节包括低位比特。

请求

功能码	起始地址	寄存器数量
1 个字节	2 个字节	2 个字节
0x03	0x0000 至 0xFFFF	1 至 125（0x7D）

响应

功能码	起始地址	寄存器数量
1 个字节	1 个字节	N*×2 个字节
0x03	2 ×N*	1 至 125（0x7D）

N*=寄存器的数量

这是一个请求读寄存器 108-110 的实例：

请求		响应	
域名	（十六进制）	域名	（十六进制）
功能	03	功能	03
		字节数	06
高起始地址	00	寄存器值Hi（108）	02
		寄存器值Lo（108）	2B
低起始地址	6B	寄存器值Hi（109）	00
高寄存器编号	00	寄存器值Lo（109）	00
		寄存器值Hi（110）	00
低寄存器编号	03	寄存器值Lo（110）	64

将寄存器 108 的内容表示为两个十六进制字节值 02 2B，或十进制 555。将寄存器 109-110 的内容分别表示为十六进制 00 00 和 00 64，或十进制 0 和 100。

■ 16（0x10）写多个寄存器

在一个远程设备中，使用该功能码写连续寄存器块(1 至约 120 个寄存器)。在请求数据域中说明了请求写入的值。每个寄存器将数据分成两字节。正常响应返回功能码、起始地址和被写入寄存器的数量。

请求 PDU

功能码	起始地址	寄存器数量	字节数	寄存器数量
1 个字节	2 个字节	2 个字节	1 个字节	N*×2 个字节
0x03	0x0000 至 0xFFFF	1 至 125（0x7D）	2 ×N*	值

N*=寄存器的数量

响应 PDU

功能码	起始地址	寄存器数量
1 个字节	2 个字节	2 个字节
0x10	0x0000 至 0xFFFF	1 至 123（0x7B）

这是一个请求将十六进制 00 0A 和 01 02 写入以 2 开始的两个寄存器的实例：

请求		响应	
域名	（十六进制）	域名	（十六进制）
功能	10	功能	10
起始地址 Hi	00		
起始地址 Lo	01	起始地址 Hi	00
寄存器数量 Hi	00		
寄存器数量 Lo	02	起始地址 Lo	01
字节数	04		
寄存器值 Hi	00	寄存器数量 Hi	00
寄存器值 Lo	0A		
寄存器值 Hi	01	寄存器数量 Lo	02
寄存器值 Lo	02		