Chapter 10 Communications

The servo drive has Modbus (RS-232, RS-485) communication and CANopen communication functions. Together with the host computer communication software, it can realize multiple functions such as parameter modificatio n, parameter query and servo drive status monitoring.

10.1 MODBUS Communication

The RS-485 communication protocol uses a single-master-multiple-slave communication method and can support networking of multiple servo drives. The RS-232 communication protocol does not support networking of multipl e servo drives.

10.1.1 Hardware wiring and EMC considerations

1） **RS-232** connection diagram

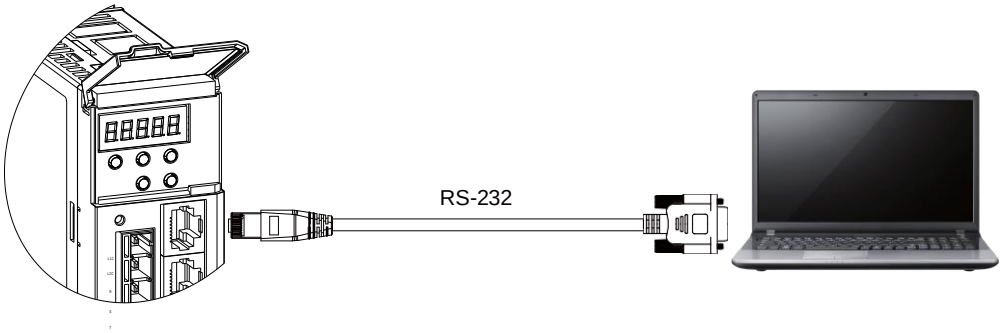

Figure 10-1 RS-232 connection diagram

2） **RS-485** connection diagram

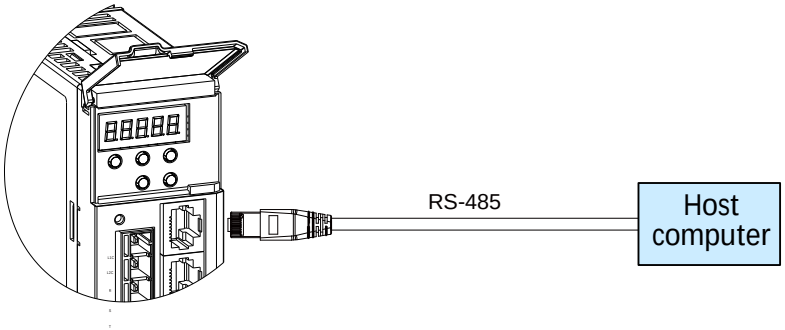

Figure 10-2 RS-485 connection diagram

3) When the number of nodes is large, the 485 bus is recommended to use a hand-in-hand bus structure.

If branch line connection is required, the branch length between the bus and the node should be as short as possible, and it is recommended not to exceed 3m.

Star connection is strictly prohibited. The following is a common bus structure diagram:

a) Recommended solution: Hand-in-hand connection structure

RS485 BUS

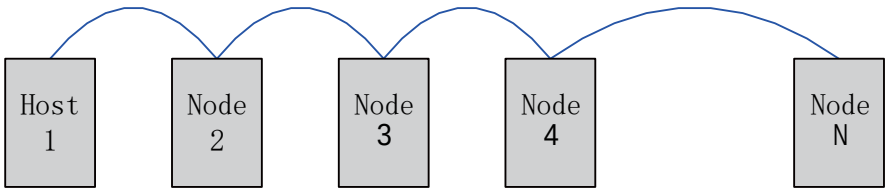

Figure 10-3 Schematic diagram of the recommended hand-in-hand connection structure

b) General solution: branch line connection structure

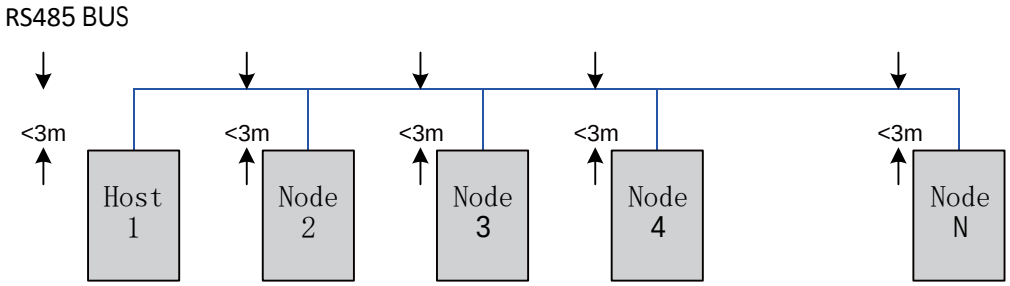

Figure 10-4 Schematic diagram of branch line connection structure

c) Wrong solution: star connection structure

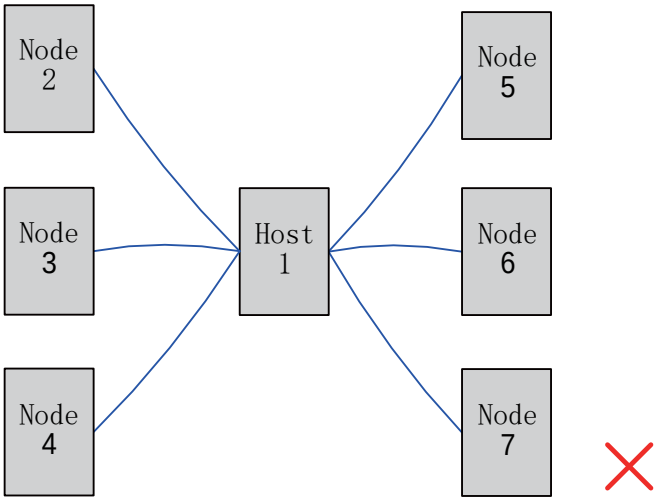

Figure 10-5 Incorrect star wiring method

NOTE

- Connect the correct bias and terminal resistors, refer to question 1 for details;
- The 485 communication line must use a twisted pair cable;
- Connect the 485 circuit reference ground GND of each node through the third cable, where the 485 ground refer ence ground of the 630P servo drive is GND;
- When shielded cables are used on site, it is recommended that both ends of the shielding layer be connected to P E at the same time. Do not connect one end to GND and the other end to PE, nor connect both ends to GND. Oth erwise, the port will be damaged.
- Use the hand-in-hand approach to arrange the bus, refer to question 3 for details;
- Use additional grounding wires to connect the PE of each node, refer to "10.1.2 EMC layout requirements".
- The 485 bus needs to be laid out separately from other interfering cables, refer to "10.1.2 EMC layout requirements".

10.1.2 EMC layout requirements

1) Site layout requirements

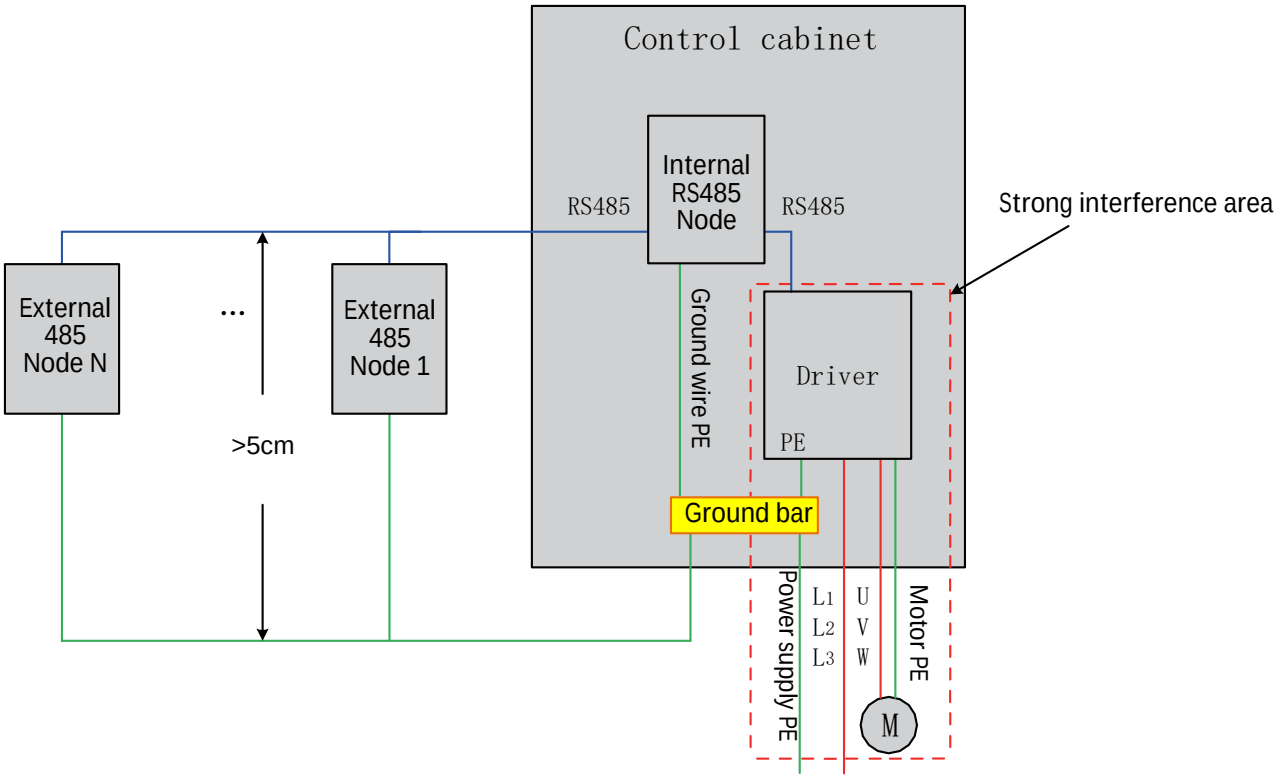

Figure 10-6 Schematic diagram of regional layout

NOTE

- Interference sources are isolated from sensitive equipment;
- The area occupied by interference equipment and cables is minimized, such as being arranged close to the outlet.

2) Ground wire PE connection requirements

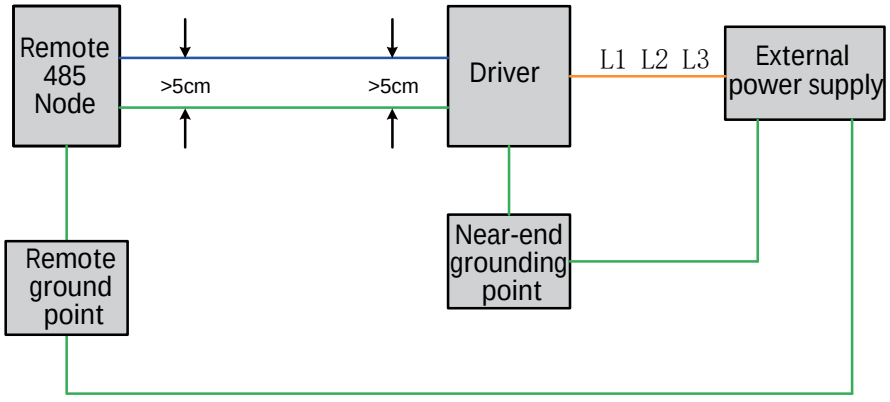

Figure 10-7 Schematic diagram of ground wire PE wiring

NOTE

- The ground wire PE must use a cable thicker than AWG12.
- The ground wire PE is connected to the ground terminal of the node or the ground bar of the cabinet where the node is located.

Note: The distance between the ground wire PE and the bus should be greater than 5cm.

3) Cable layout requirements

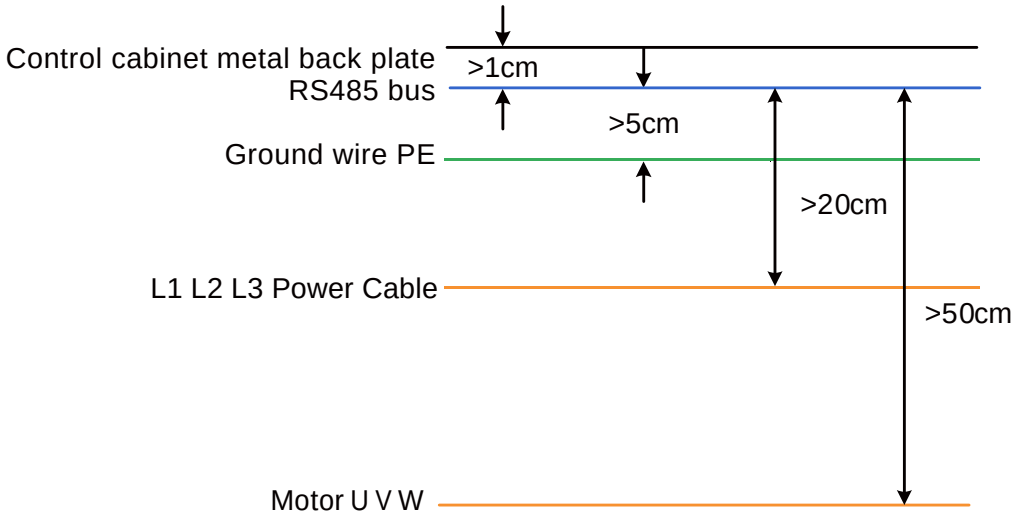

Figure 10-8 Cable layout diagram

NOTE

- Keep the 485 bus at least 20cm away from the high-voltage cable;
- Keep the 485 bus at least 1cm away from the metal cabinet back panel;
- Keep the 485 bus at least 50cm away from the motor UVW power line;
- Keep the 485 bus at least 5cm away from the field ground line;

10.1.3 Relationship between transmission distance, nodes and transmission rate in 485 interface field application

Number	rate	Transmission distance	Number of nodes	Wire diameter
1	57.6kpbs	100m	128	AWG26
2	19.2kbps	1000m	128	AWG26

Notice:

- RS485 can connect 32 servo drives at the same time. If you want to connect more servo drives, you must install an amplifier, which can be expanded to 247 servo drives at most.
- Using RS-485 communication, if the host computer only supports RS-232, it can be connected through an RS-232/RS-485 converter.

10.1.4 Communication parameter setting

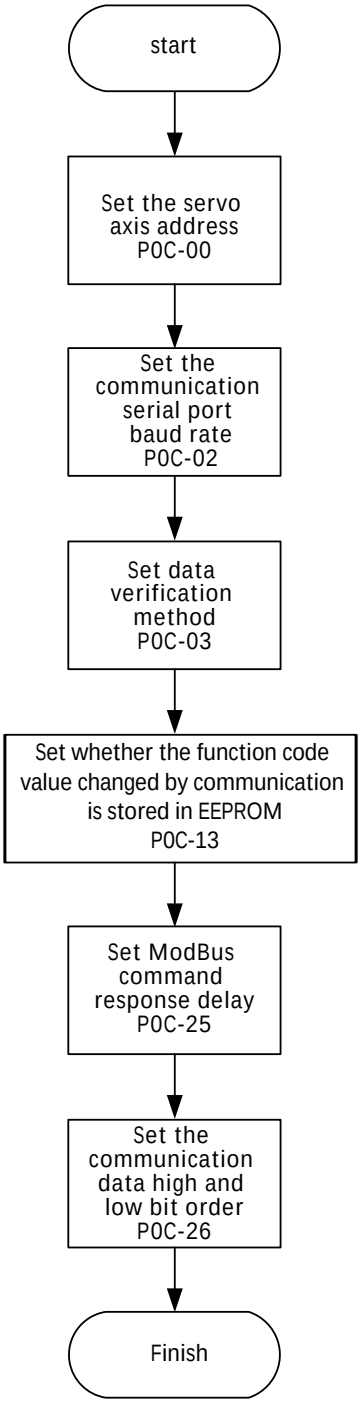

Figure 10-9 Communication parameter setting process

1) Set the drive axis address P0C-00:

When multiple servo drives are networked, each drive can only have a unique address, otherwise communication abnormalities will occur and communication will fail.

0: Broadcast address

1～247: Slave address

The host computer can write to all slave drivers through the broadcast address. The slave drivers receive the frames of the broadcast address and perform corresponding operations but do not respond.

2) Set the communication rate between the driver and the host computer P0C-02

The communication rate of the servo drive must be set consistent with the communication rate of the host computer, otherwise communication will fail.

When multiple servo drives are networked, if the communication baud rate of a drive is inconsistent with th at of the host, it will cause communication errors for the axis or affect the communication of other axes.

3) Set the data verification method when the drive communicates with the host computer P0C-03

LCDA630P provides 2 data check modes: even check (P0C-03=1) and odd check (P0C-03=2).
No check (P0C-03=0) is also available.

a) Even parity or odd parity

The actual number of bits transmitted per frame of data is 11 bits: 1 start bit, 8 data bits, 1 check bit, and 1 end bit.

10	9	8～1	0
End bit	Check bit	Data bit	Start bit

b) No checksum

If no checksum is selected, there are two data frame formats to choose from:

① The actual number of bits transmitted per frame of data is 11 bits, including 1 start bit, 8 data bits, and 2 end bits.

10～9	8～1	0
End bit	Data bit	Start bit

② The actual number of bits transmitted in one byte is 11, including 1 start bit, 8 data bits, and 1 end bit.

10	9	8～1	0
Invalid bit	End bit	Data bit	Start bit

The data bits are in hexadecimal.

NOTE

- The host computer data frame format must conform to the above format, otherwise it will not be able to communicate with the drive.

4) Set whether the function code changed by communication is stored in real time EEPRROMP0C-13

The servo drive provides a real-time saving function for function codes (P0C-13=1). After the corresponding funct ion code value is modified, it is stored in the EEPROM in real time, and has a power-off saving function. However, this function should be used with caution:

a) If the function code value only needs to be changed once and the value is to be used afterwards, the function code real-time saving function can be enabled (P0C-13=1);

b) If the function code value needs to be changed frequently, it is recommended to turn off the function code real-time saving function (P0C-13=0). Otherwise, the EEPROM service life will be reduced due to fre quent erasing of the EEPROM.

NOTE

- If the EEPROM is frequently erased within a certain period of time, the drive will generate warning FU.942 (servo parameters are stored in EEPROM very frequently)!
- If the EEPROM is damaged, the drive will have other irreversible faults!

5) Set MODBUS communication response delay P0C-25

Function code P0C-25 is used to add a delay to the servo response. After receiving the command, the servo will delay for the time set by P0C-25 before replying to the host.

6) Set the communication data high and low bit sequence P0C-26

Servo drive function code display format: HXX-YY

Where:

XX: function code group number, in hexadecimal format;

YY: Offset within the function code group, which is decimal data and must be converted into hexadecimal dat a in the communication data frame.

The servo drive function code communication address is a 16-bit address, consisting of the function code gro up number (high 8 bits) + group offset (low 8 bits).

If the function code data range is within -65536 to +65535, it is a 16-bit function code, which only occupies 1 function code group offset and 1 address, and does not involve the high and low bit sequence of communicat ion data. For example, P02-00, its communication address is 0x0200.

If the data range of the function code exceeds -65536 to +65535, it belongs to a 32-bit function code, occupies two consecutive offset numbers in the function code group, and occupies 2 consecutive addresses, but the communication address is only determined by the address with the lower offset number, and the high and low bit sequence of the communication data must be set correctly. Otherwise, data reading and writing errors will occur.

For example, P11-12 (the first shift) occupies two consecutive offset numbers in the function code group, namely P11-12 and P11-13. The communication address 0x110C with the lower offset number (P11-12) stores the lower 16 bits of the function code value, and the communication address 0x110D with the higher offset number (P11-13) stores the upper 16 bits of the function code value.

When the "1st stage movement displacement" is preset to 0x40000000 (decimal 1073741824), the value of P1 1-12 should be set to 0x0000 and the value of P11-13 should be set to 0x4000.

When writing the function code, you need to determine the sequence of "0x0000" and "0x4000" in the communicat ion frame according to the setting of P0C-26.

- The servo drive does not support separate operation of the upper 16 bits of the 32-bit function code!
- When using communication to modify the function code, you need to pay attention to the setting range, unit, effective time, setting category, positive and negative hexadecimal conversion, etc. of the function code. Plea se refer to the description of the function code for details.

Notice:

- The register address of some manufacturers' PLC/touch screen MODBUS instruction programming is not equa l to the actual register address, but equal to the actual register address plus 1. This is because the starting add ress of the standard MODBUS instruction register is 1, and the actual register address of many devices starts f rom 0 (such as this servo drive). Considering compatibility, the PLC/touch screen manufacturer has subtracted 1 from the programming register address during actual physical transmission. When this type of PLC/touch sc reen communicates with the servo drive through MODBUS, the programmer needs to clarify this point in orde r to correctly read and write the function code of the servo drive. For example, when programming, the read (write) register address is 0x0201, and the actual read (write) function code is P02-00, not P02-01.
- If you are not sure whether the register address during PLC/touch screen MODBUS instruction programming i s equal to the actual register address, you can select two adjacent function codes with different values, and u se the 0x03 (read) instruction to read the function code with the larger code. If the read function code value is equal to the function code value with the smaller code, it means that the register address during programmin g is equal to the actual register address plus 1.

☆Associated function code:

Function code	name	Setting range	unit	Function	setting method	Effective time	Factorys setting
P0C-00	Drive axis address	0～247	–	Set the drive axis address	Running settings	Effective immediately	1
P0C-02	Serial port baud rate setting	0：2400 1：4800 2：9600 3：19200 4：38400 5：57600	Kbp/s	Set the communication rate between the driver and the host computer	Running settings	Effective immediately	2
P0C-03	MODBUS Data Format	0: No parity, 2 end bits 1: Even parity, 1 end bit 2: Odd parity, 1 end bit 3: No parity, 1 end bit	–	Set the data verification method when the drive communicates with the host computer	Running settings	Effective immediately	3
P0C-13	Whether the MODBUS communication write function code is updated to EEPROM	0: Do not update EEPROM 1: Update EEPROM except P0B and P0D groups	–	Set whether the function code value changed by communication is stored in EEPROM	Running settings	Effective immediately	1

Function code	Name	setting range	unit	Function	Setting Method	Effective time	Factory Setting
P0C-25	MODBUS command response delay	0～5000	ms		Running settings	Effective immediately	1
P0C-26	MODBUS communication data high and low order	0-high 16 bits first, low 16 bits last 1-low 16 bits first, high 16 bits last	1	Set the transmission format for 32-bit data when using MODBUS communication	Running settings	Effective immediately	1

10.1.5 MODBUS communication protocol

The function codes of the servo drive are divided into 16 bits and 32 bits according to the data length. The fu nction codes can be read and written through the MODBUSRTU protocol. When writing function code data, the c ommand codes are different according to the data length.

operate	Command code
Read 16/32-bit function code	0x03
Write 16-bit function code	0x06
Write 32-bit function code	0x10

1) Read function code: 0x03

In the MODBUSRTU protocol, both 16-bit and 32-bit function codes are read using command code: 0x03

Request frame format:

START	Greater than or equal to 3.5 characters of idle time, indicating the beginning of a frame
ADDR	Servo axis address 1~247. ◆ Note: Here 1 to 247 are decimal numbers, which will be converted to hexadecimal numbers when filling in ADDR.
CMD	Command code: 0x03
DATA[0]	The starting function code group number, such as function code P06-11, 06 is the group number. ◆ Note: 06 is a hexadecimal number. No conversion is required when filling in DATA[0].
DATA[1]	The offset within the starting function code group, such as function code P06-11, 11 is the offset. ◆ Note: Here 11 is a decimal number and should be converted to hexadecimal 0x0B when filling in DATA[1].
DATA[2]	Read function code number (high 8 bits), hexadecimal
DATA[3]	Read function code number (lower 8 bits), hexadecimal
CRCL	CRC check valid byte (low 8 bits)
CRCH	CRC check valid byte (high 8 bits)
END	If the idle time is greater than or equal to 3.5 characters, a frame ends.

Response frame format:

START	Greater than or equal to 3.5 characters of idle time, indicating the beginning of a frame
ADDR	Servo axis address, hexadecimal
CMD	Command code, 0x03
DATALENGTH	The number of function code bytes is equal to the number of read function codes N*2
DATA[0]	Starting function code value, high 8 bits
DATA[1]	Starting function code value, lower 8 bits
DATA[…]	–
DATA[N*2-1]	The last function code value, lower 8 bits
CRCL	CRC check low effective byte
CRCH	CRC check high significant byte
END	If the idle time is greater than or equal to 3.5 characters, a frame ends.

In the MODBUSRTU protocol, the command code for writing a 16-bit function code is 0x06; the command code for writing a 32-bit function code is 0x10.

2) Write 16-bit function code (0x06)

 Notice:

- It is forbidden to use 0x06 to write the 32-bit function code, otherwise unpredictable errors will occur!

Request frame format:

START	Greater than or equal to 3.5 characters of idle time, indicating the beginning of a frame
ADDR	Servo axis address 1~247. ◆ Note: Here 1 to 247 are decimal numbers, which will be converted to hexadecimal numbers when filling in ADDR.
CMD	Command code: 0x06
DATA[0]	The starting function code group number, such as function code P06-11, 06 is the group number. ◆ Note: 06 is a hexadecimal number. No conversion is required when filling in DATA[0].
DATA[1]	The offset within the starting function code group, such as function code P06-11, 11 is the offset. ◆ Note: Here 11 is a decimal number and should be converted to hexadecimal 0x0B when filling in DATA[1].
DATA[2]	Write data high byte, hexadecimal
DATA[3]	Write data low byte, hexadecimal
CRCL	CRC check low effective byte
CRCH	CRC check high significant byte
END	If the idle time is greater than or equal to 3.5 characters, a frame ends.

Response frame format:

START	Greater than or equal to 3.5 characters of idle time, indicating the beginning of a frame
ADDR	Servo axis address, hexadecimal.
CMD	Command code: 0x06
DATA[0]	The group number of the function code being written. For example, if the function code is H06-11, it is 0x06
DATA[1]	The written function code offset, such as writing function code H06-11, is 0x0B
DATA[2]	Write data high byte, hexadecimal
DATA[3]	Write data low byte, hexadecimal
CRCL	CRC check low effective byte
CRCH	CRC check high significant byte
END	If the idle time is greater than or equal to 3.5 characters, a frame ends.

3) Write 32-bit function code (0x10)

 Notice:

- It is forbidden to use 0x10 to write the 16-bit function code, otherwise unpredictable errors will occur!

Request frame format:

START	Greater than or equal to 3.5 characters of idle time, indicating the beginning of a frame
ADDR	Servo axis address 1~247. ◆ Note: Here 1 to 247 are decimal numbers, which will be converted to hexadecimal numbers when filling in ADDR.
CMD	Command code: 0x10
DATA[0]	The starting function code group number to be written, such as writing function code H11-12, 11 is the function code group. ◆ Note: Here 11 is a hexadecimal number, and no conversion is required when filling in DATA[0]
DATA[1]	The internal offset of the written starting function code group, such as writing function code H11-12, 12 is the internal offset of the group. ◆ Note: Here 12 is a decimal number, which should be converted to hexadecimal 0x0C when filling in DATA[1].
DATA[2]	The high 8 bits of the function code are M(H), and the length of a 32-bit function code is 2.
DATA[3]	Function code number lower 8 digits M(L)
DATA[4]	The number of function codes corresponds to the number of bytes M*2. For example, if P05-07 is written alone, DATA[4] is P04.

DATA[5]	Write the high 8 bits of the start function code, hexadecimal
DATA[6]	Write the lower 8 bits of the start function code, hexadecimal
DATA[7]	Write the high 8 bits of the start function code group offset +1, hexadecimal
DATA[8]	Write the lower 8 bits of the start function code group offset +1, hexadecimal
CRCL	CRC check low effective byte
CRCH	CRC check high significant byte
END	If the idle time is greater than or equal to 3.5 characters, a frame ends.

Response frame format:

START	Idle time greater than or equal to 3.5 characters indicates the start of a frame
ADDR	Servo axis address, hexadecimal.
CMD	Command code: 0x10
DATA[0]	Written function code group number, such as writing function code H11-12, then 0x11
DATA[1]	Written function code offset, such as writing function code H11-12, then 0x0C
DATA[2]	Written function code number high 8 bits
DATA[3]	Written function code number low 8 bits
CRCL	CRC check low effective byte
CRCH	CRC check high effective byte
END	Idle time greater than or equal to 3.5 characters, a frame ends

4) Error response frame

Error frame response format:

START	Idle time greater than or equal to 3.5 characters indicates the start of a frame
ADDR	Servo axis address, hexadecimal.
CMD	Command code + 0x80
DATA[0]～[3]	DATA|errorcode
CRCL	CRC check low effective byte
CRCH	CRC check high effective byte
END	Greater than or equal to 3.5 characters idle time, a frame ends

Error code:

Error Code	Coding description
0x0001	Illegal command code
0x0002	Illegal data address
0x0003	Illegal data
0x0004	Slave device failure

5) Communication example (P0C-26=0)

a) The host sends a request frame

01	03	02	02	00	02	CRCL	CRCH

This request frame indicates: read 0x0002 words of data from the register starting with the function code P02-02 of the drive with the axis address 01.

Slave response frame:

01	03	04	00	01	00	00	CRCL	CRCH

This response frame indicates that the slave returns 2 words (4 bytes) of data, and the data content is 0x0001, 0x0000.

If the slave response frame is:

01	83	02	CRCL	CRCH

This response frame indicates: a communication error occurred, the error code is 0x02; 0x83 indicates an error.

b) The host sends a request frame:

01	06	02	02	00	01	CRCL	CRCH

This request frame indicates: write 0x0001 to the function code P02-02 of the drive with axis address 01.

Slave response frame:

01	06	02	02	00	01	CRCL	CRCH

This response frame indicates that the host successfully writes the function code.

If the slave response frame is:

01	86	02	CRCL	CRCH

This response frame indicates: a communication error occurred, the error code is 0x02; 0x86 indicates an error.

c) Read 32-bit function code P05-07:

Host request frame:

01	03	05	07	00	02	CRCL	CRCH

Slave response frame:

01	03	04	00	01	00	00	CRCL	CRCH

This response frame indicates that the value of the P05-07 function code is 0x00000001.

6) 32-bit function code addressing

When using MODBUS instructions to read and write 32-bit function codes, the communication address is determined by the address with the lower offset number in the function code group, and operations are performed on the offset numbers in 2 function code groups at a time.

For example, the MODBUS command to read "1st segment movement displacement" P11-12 is:

Servo axis address	03	11	0C	00	02	CRCL	CRCH

If the "1st segment shift" is known to be 0x40000000 (decimal 1073741824):

If P0C-26=1 (lower 16 bits first, higher 16 bits last), the response frame is:

Servo axis address	03	04	00	00	40	00	CRCL	CRCH

If P0C-26=0 (high 16 bits first, low 16 bits last), the response frame is:

Servo axis address	03	04	40	00	00	00	CRCL	CRCH

For example, the MODBUS command to write "0x12345678" to "Movement Displacement 1":

If P0C-26=1 (lower 16 bits first, higher 16 bits last)

Servo axis address	10	11	0C	00	02	04	56	78	12	34	CRCL	CRCH

If P0C-26=0 (high 16 bits first, low 16 bits last)

Servo axis address	10	11	0C	00	02	04	12	34	56	78	CRCL	CRCH

For example, to write the 32-bit function code P05-07 data to 0x00100000 (decimal is 1048576):

If P0C-26=0 (high 16 bits first, low 16 bits last), the response frame is:

01	10	05	07	00	02	04	00	00	00	10	CRCL	CRCH

7) CRC check

The host computer and servo drive must use the same CRC check algorithm to communicate, otherwise a CRC check error will occur. The servo drive uses a 16-bit CRC, with the low byte first and the high byte last. The CRC function is as follows:

```
Uint16COMM_CrcValueCalc(constUint16*data,Uint16length)
{
    Uint16crcValue=0xffff;
    int16i;
    while(length--)
    {
        crcValue^=*data++;
        for(i=0;i<8;i++)
        {
            if(crcValue&0x0001)
            {
                crcValue=(crcValue>>1)^0xA001;
            }
            else
            {
                crcValue=crcValue>>1;
          }
      }
    }
    return(crcValue);
}
```

8) Hexadecimal representation of signed numbers

When writing a signed function code (including 16-bit and 32-bit), the pre-written data needs to be converted into a hexadecimal complement.

a) 16-bit function code

● Data is positive or 0: complement code = original code

● If the data is a negative number: the complement code = 0xFFFF - the complement code of the absolute value of the data + 0x0001

For example:

The original code of the 16-bit signed positive number +100 is 0x0064, so the complement code is also: 0x0064;

The hexadecimal complement of the 16-bit signed negative number -100 is: 0xFFFF－0x0064＋0x0001=FF9C

b) 32-bit function code

● Data greater than or equal to 0: complement code = original code

● If the data is a negative number: the complement code = 0xFFFFFFFF - the complement code of the absolute value of the data + 0x00000001

For example:

The original code of the 32-bit number 100 is 0x00000064, so the complement code is also: 0x00000064;

The 32-bit number -100, its hexadecimal complement is: 0xFFFFFFFF–0x00000064+0x00000001=FFFFFF9C

10.1.6 Common problems and solutions for RS485 communication on site

1) Question 1: Correct terminal resistor connection method

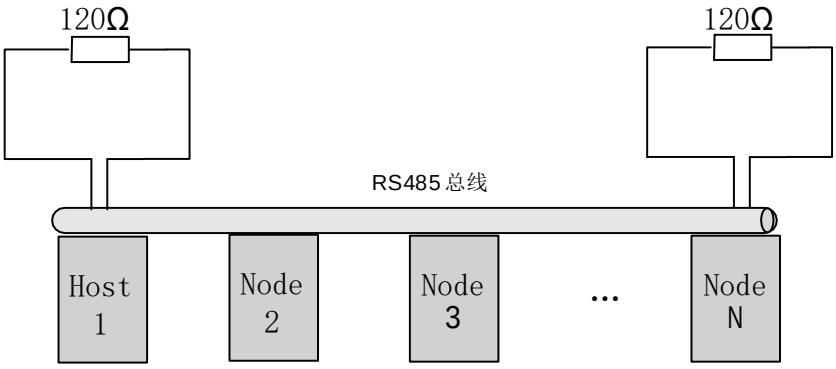

Figure 10-10 Schematic diagram of terminal resistance access method

- It can only be terminated and matched at both ends;
- The master station is recommended to be placed at one end of the bus;
- Use the ohmmeter of the multimeter to measure the resistance between the 485 buses (the device needs to be powered off during measurement). If the measured value is around 60Ω, it is normal. If it is less than 50Ω, please check whether there are other nodes with matching resistors in addition to the two ends of the bus, a nd disconnect them. If it is 0Ω, please check whether there is a short circuit or node damage.

2) Question 2: Correct wiring method (for some nodes without GND wiring points)

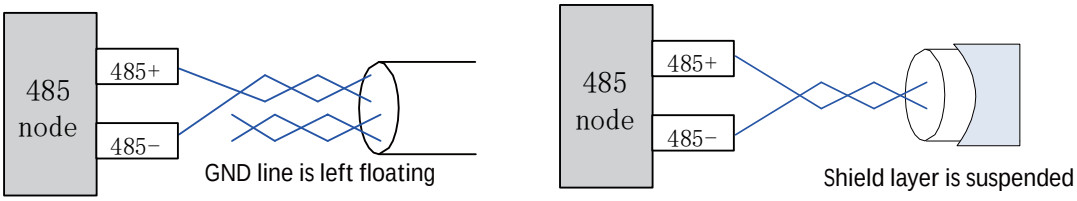

Figure 10-11 Wiring diagram when there is no GND

● Solution 1: Check if there is a reference ground shared with the 485 circuit on other ports of this node. If so, GND is connected to this reference ground. Pay special attention that the shielding layer cannot be connected to the reference ground, otherwise the 485 port will be damaged.

● Solution 2: Check if there is a reference ground shared with the 485 circuit on this node board. If so, GND is connected to this reference ground. Pay special attention that the shielding layer cannot be connected to the reference ground, otherwise the 485 port will be damaged.

● Solution 3: If the reference ground of the 485 circuit cannot be found, please suspend the GND line as shown in the figure above, and ensure that the ground wire PE is reliably connected.

● Solution 4: When the number of nodes is small, add a filter capacitor between 485+ and 485-, refer to question 6.

3) Question 3: Correct multi-node connection method

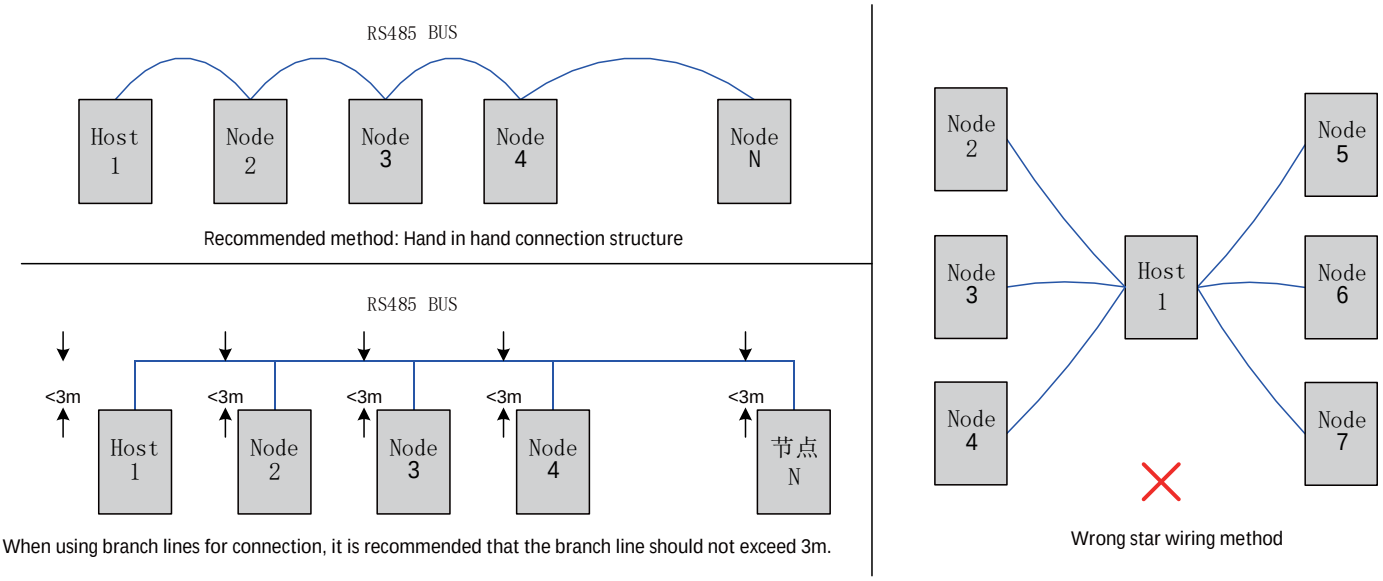

Figure 10-12 Schematic diagram of three multi-node connection methods

4) Question 4: Measures to suppress external interference in the system

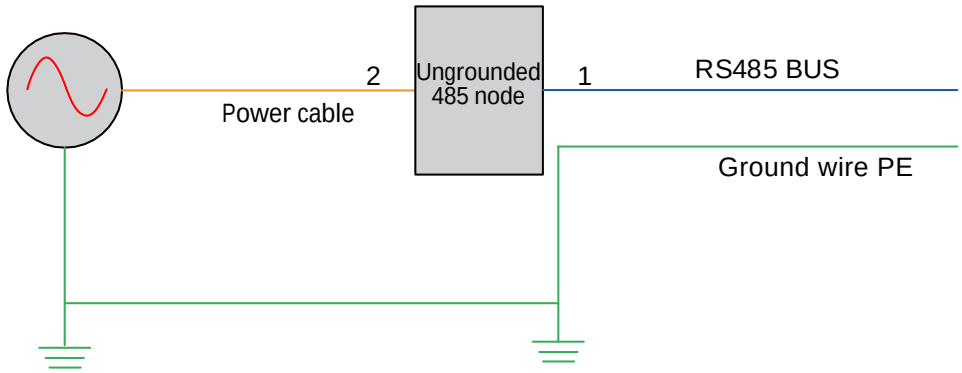

Figure 10-13 Schematic diagram of suppressing external interference

● Solution 1: Wrapping a magnetic ring at position 1 can effectively suppress external interference of the system. This method is recommended.
● Solution 2: Wrapping a magnetic ring at position 2 can also suppress external interference of the system.

5) Question 5: Driver interference suppression measures

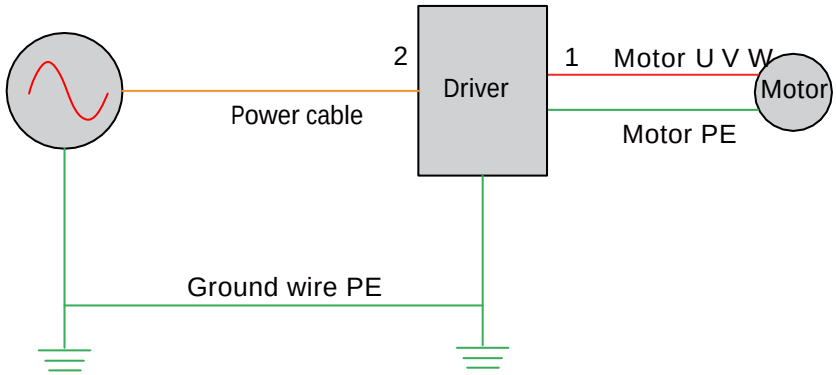

Figure 10-14 Driver interference suppression diagram

● Solution 1: Add a filter magnetic ring at position 1, pass the three lines UVW (excluding ground wire PE) through the magnetic ring at the same time, and it is recommended to wind three turns.
Measure 1 is the preferred solution with the best effect.

● Solution 2: Add a filter magnetic ring at position 2, pass the three lines UVW (excluding ground wire PE) through the magnetic ring at the same time, and it is recommended to wind three turns.

On-site problem location flow chart:

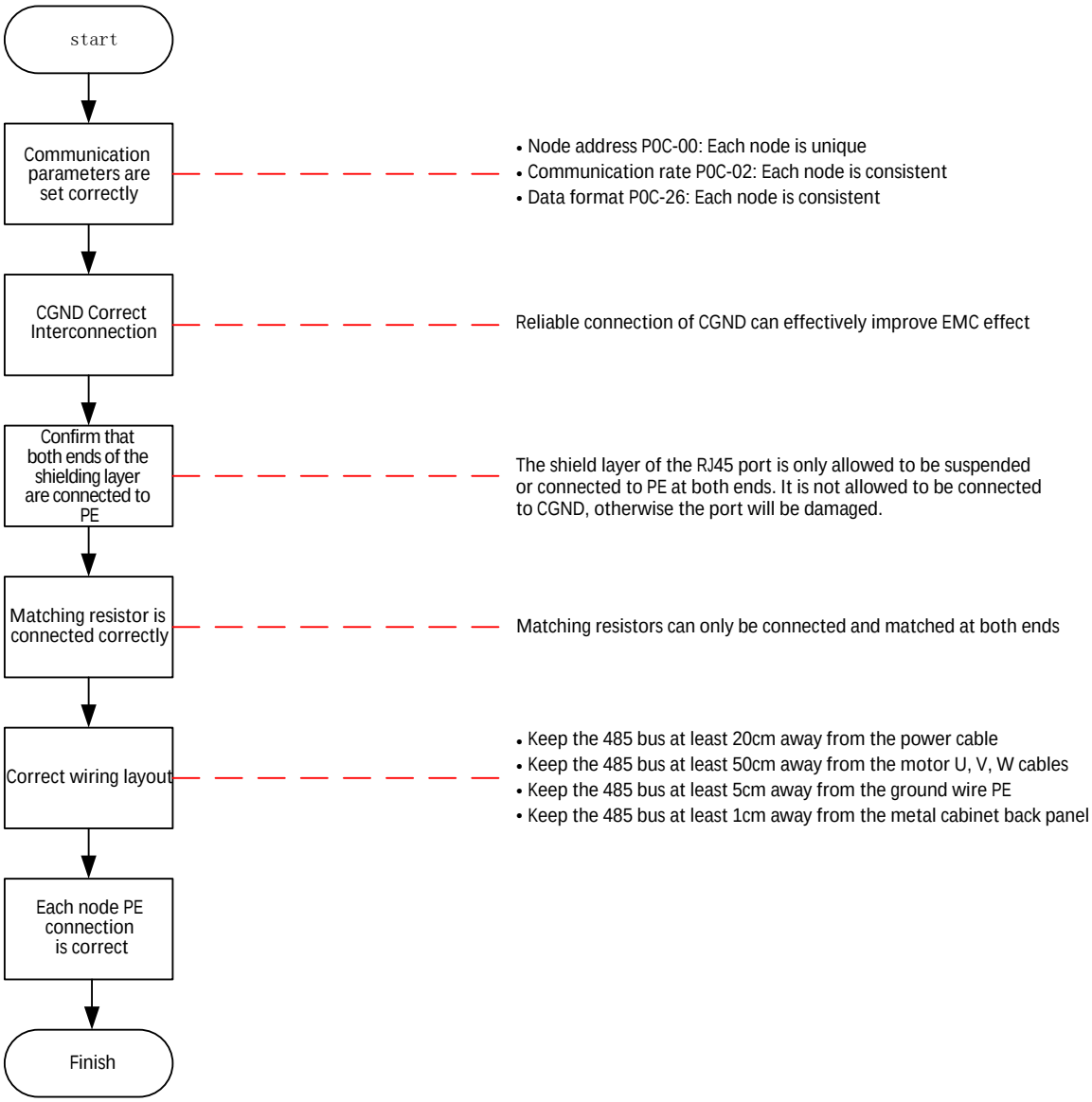

Figure 10-15 485 communication on-site problem location flowchart