中空系列电机控制使用说明

目录

中空系列电机控制使用说明 1

一、 电机控制回路 2

二、 ttl 串口/rs485 控制和配置 2

三、 串口控制电机 2

四、 串口可配置的数据 3

五、 串口设置电机机械零点或清除机械零点 3

六、 电机 Can 模式控制指南 3

七、 电机 Canfd 模式控制指南 9

八、 电机配件使用指南（USB 转 CAN&CANFD 模块） 11

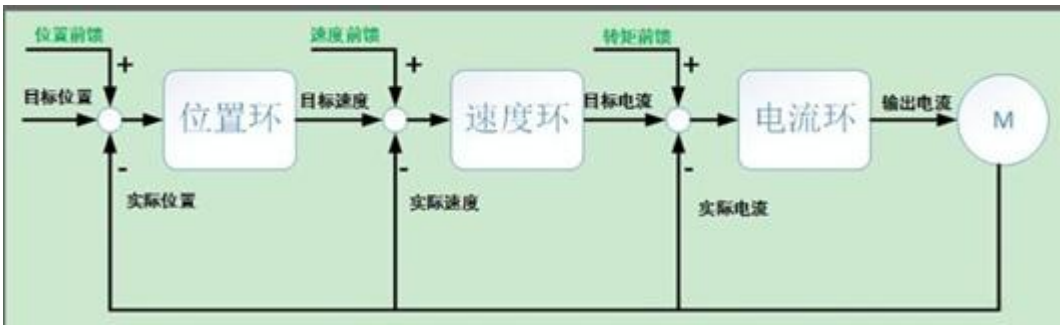
九、 电机配件使用指南（485&脉冲转 canfd 模块） 12

十、 电机 PWM 控制输入输出参数表 22

十一、 电机 Encoder 输出模式参数 23

十二、 中空编码器 23

一、电机控制回路



对于永磁无刷电机，一般条件下，电机电流正比于电机扭矩，换算公式为扭矩 T = Kt*I
其中 Kt 为电机扭矩常数，I 为电流值，电机扭矩常数可查阅电机参数表得到，以下未特殊说明，电流即为扭矩。

位置环使用 PD 控制器，PD 可调，主机可指定目标位置，指定最大速度和最大电流，控制电机运动，电机运行的实际速度和实际电流和负载相关，指定的最大电流需要大于电机运动过程中的最大负载，

速度环使用 PI 控制器，PI 可调，主机可指定目标速度，最大电流控制电机运动

电流环使用 PI 控制器，PI 可调，主机可指定目标电流，电机运行速度和负载有关，如果负载小于扭矩，电机将运行到该电压和负载条件下的最大速度

二、ttl 串口/rs485 控制和配置

1. 串口为电机配置界面，可配置电机控制相关参数，串口所配置的参数，无特殊说明，均存储在电机内部存储器中，掉电后依然保存，下次上电后，按配置运行。
2. 串口波特率为固定 19200，使用 ascII 字符模式进行交互，主机发送命令以\n\r 结束，本公司提供的上位机软件，已自动加上命令尾缀，如果用户使用单片机或第三方串口软件，需注意要自己加上\n\r 的尾缀，所有命令均为小写字符，以下无特殊说明，均不包含\n\r 结束符

三、串口控制电机

主菜单下输入 m/bXXX.X 根据电机工作模式控制电机，如 m10.0 在不同模式下有不同的解释，位置模式下电机正转到 10.0 度，速度模式下电机以 10Hz 转速运行，扭矩模式下为电流 10A 正扭矩运行，注

意输入数据为浮点数，需要带小数点输入

输入 r 电机停止，进入自由状态,注意串口控制电机只用于测试电机的好坏,功能非常有限,需要在产品里使用电机需要使用 can 接口

四、串口可配置的数据

1. 电机运行模式，输入 p 进入模式选择
   a) 扭矩模式
   b) 速度模式
   c) 位置模式

   电机运行模式主要用于通过串口命令直接控制电机，
   配置的模式，可通过 can/canfd 指令临时修改

2. 电机参数，输入 s 进入参数配置
   a) 电流环带宽，即电流环 P 值，100-2000
   b) 速度环带宽，即速度环 P 值（暂不开放）
   c) 位置环增益，位置环 P 值 0- 100
   d) 位置环阻尼，位置环 D 值 0-100
   e) 多圈运动设置，中空系列电机 Pmax 均需设置为 1.
   f) Canfd 模式使能，选择使用 can 模式或 canfd 模式，1 代表 canfd 模式，0 代表 can 模式
   g) Canid 电机 can id 值，代表电机接收滤波器 id 值，只有 can 帧 id =电机 id 的数据帧才会被电机接收并产生应答帧
   h) Can 主机 id，总线上的主机 id，电机接收到数据帧后的应答帧将发送给主机 id 对应的主机
   i) CW config，正反方向控制位，用户不可修改，指示电机当前相序

五、串口设置电机机械零点或清除机械零点

1. 使用串口设置电机零点，菜单的 SetZeroPoint 和 ClrZeroPoint 分别对应设置零点和清除零点到出厂状态。
2. 设置的零点相对出厂零点的角度会显示在串口菜单界面上，显示的单位为度，需注意零点数值需要在 5-355 度之间，用户可读取显示的零点数值确定是否设置成功。
3. Can&Canfd 设置的零点不保存在电机内部，再次上电后，电机将按串行口显示的零点数值执行偏移并运动
4. 零点位置为单圈有效，多圈或带减速机构的零点，需要用户自己做偏移

六、电机 Can 模式控制指南

1.Can 协议概述

Can 速率 1M bit/s,硬件包括 CAN_H 和 CAN_L 和 GND 三个信号脚,一般的总线拓扑需要在发送端和接收端各并联一个 120 欧终端电阻,通过串口设置电机模块 Can 的通讯速率，未特殊说明，所有电机均不支持调整通讯波特率

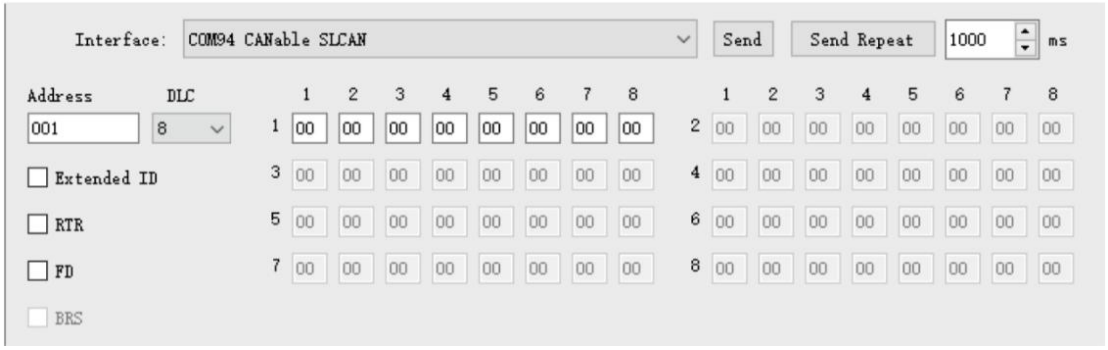
电机模块都属于从机,采用一应一答通讯模式,即主机发送命令从机应答命令,如果主机没有发送对应 ID 的命令,从机处于静默状态不会主动往总线上发送数据,从接收到指令到应答主机,电机模块需要 500us 的延迟时间,因此系统目前支持的最大通讯间隔是 500us

电机模块的自身的 ID 通过串口进行配置,可选择 0-100 之间的任意数,注意相同总线上不能出现重复的 ID

标准 CAN ID 帧 位定义, Address 即为本说明所指 ID 数值

主机 ID 也通过串口写入电机模块,这样当电机模块接收到符合规范的 Can 数据帧后,将向设定好的主机 ID 发送应答数据帧,默认的主机 ID 为 100,即主控制器设定自身 ID 为 100 就可以接收到电机模块的应答

标准 can 帧发送设定



Can 使用 标准帧和扩展帧用于通讯,其中标准帧用于常规通讯,扩展帧用于一些特殊的设置,标准帧和扩展帧使用同一个 ID，需要更多通讯位数的用户，可使用 canfd 协议（16byte 数据帧长度）

Can 控制模式分为两种，上电默认使用普通模式，切换 Can 控制模式通过给电机发送特定的扩展帧来实现，如果需要使用扩展模式，每次电机上电，需要手动切换一次

扩展帧位定义如下

共 8 字节，使用第一字节第二位设定扩展模式和标准模式

字节定义

Byte 1

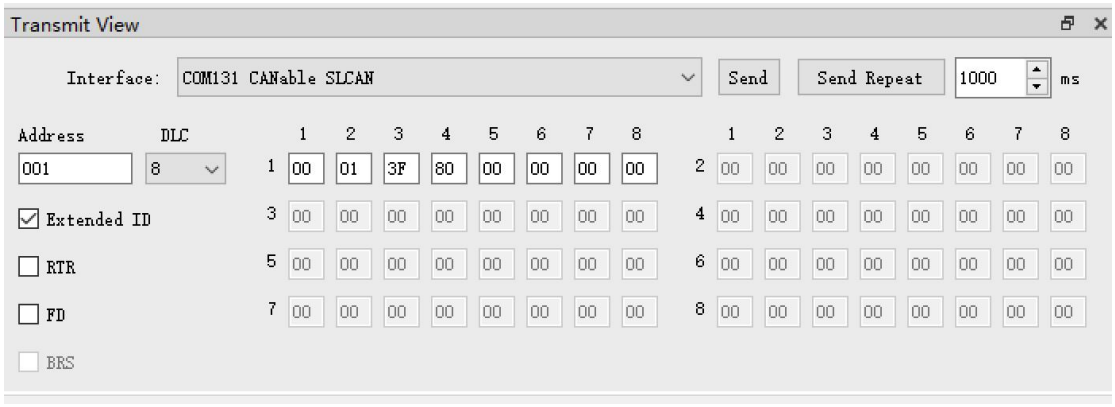
Bit8	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1
未使用 需置 0	未使用 需置 0	未使用 需置 0	未使用 需置 0	未使用 需置 0	未使用 需置零	扩展地址模式 1 = 普通模式 0 = 扩展模式	未使用 需置零

Byte 2-8

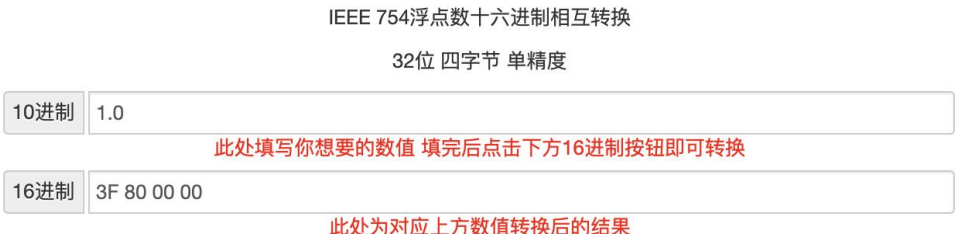
未使用

例如需要将电机设置为扩展 can 控制模式,发送 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 即可,电机应答帧长度，普通模式为 6Byte，扩展模式为 8Byte，可辅助判断切换模式是否成功。

扩展帧设置电机速度环 PI,电流环 PI, PI 设置值掉电不存储,每次上电需要重新设置



第一字节固定为 0x00,第二字节为 0x01 – 0x04,依次对应 speed_kp,speed_ki,current_kp,current_ki
第三到第六字节为设置的数值,数值是相对值, 实际 pi 值为预设 pi 数值*设置值,如设置值=1.0,那么 pi 值保持不变,设置的数值为 ieee754 32 位浮点,高位在前,如图所示 3F 80 00 00 = 1.0,浮点转换方法可自行查阅



1. Can 普通模式（绝对位置模式）

特殊指令

0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFC

启动电机,电机按设定好的模式和位置,速度,力矩指令启动运行,设置方式参考普通指令说明.

如果电机在启动前没有接收到任何设定的命令，电机执行的运动状态未知，因此在启动前需要先给电机设定需要运行的位置速度扭矩命令

0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFD

停止电机,电机无对外输出力

0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xF9

切换为力矩控制模式

0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFA

切换为速度力矩控制模式

0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFB

切换为位置速度力矩三闭环模式

0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFE

设定电机当前位置为机械零位,同时清除多圈计数,需要在电机未使能的状态下执行,此位置掉电不保存

普通指令

a. 设置电机位置,速度,力矩,位置环 Kp,Kd,其中 Kp 影响位置环响应时间,Kd 影响位置环运动阻尼,位置环无积分控制项. 速度环 Kp Ki Kd 暂不开放, 力矩环 Kp 可通过串口直接设置,力矩环 Ki Kd 暂不开放

Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7	Byte8

位置命令高 8	位置命令低 8	速度命令高 8	速度命令低 4+位置增益高 4	位置增益低 8	位置阻尼高 8	位置阻尼低 4+力矩命令高 4	力矩命令低 8

位置命令共 16 位长度,从 0x0000 到 0xFFFF,0X8000 为位置命令 0,因此上位机下发指令时,需要把指令偏移 0X8000, 位置命令满量程在串口中设置 Pmax , Pmax = 1 时,位置命令范围为+-1 圈,即 0XFFFF 代表正向旋转 1 圈,0x0000 代表反向旋转一圈,当 Pmax >1 时, 位置命令范围变为+-Pmax 圈

例如设置电机位置为 180 度

设置的位置码值 = （电机角度/Pmax*360）*0x8000 + 0x8000 （无单位）

读取的电机位置 = ((code – 0X8000)/0X8000)*360*Pmax（单位度）如果电机运动位置超出+-一圈的范围，上传的电机位置值恒定为 0xffff 或 0x0000，可根据此数值判断电机运动是否超出范围

速度命令共 12 位长度,从 0x000 到 0xFFF,同样 0x800 为速度命令 0,因此上位机下发指令时,需要把指令偏移 0X800,速度命令满量程针对不同型号电机不同 ，

例如速度设定为+100 rad/s ,速度满量程查表得到该电机为 200rad/s, 速度码值为 (100/200)*2048 + 2048 = 3072 = 0xC00

速度设定为-100 rad/s ,速度满量程查表得到该电机为 200rad/s, 速度码值为 (-100/200)*2048 + 2048 = 1024 = 0x400

设置的速度码 = （速度值/速度满量程）*0x800 + 0x800 (无单位)

读取的速度值 = （code – 0x800）/0x800 * 速度满量程（单位 rad/s）

力矩命令共 12 位长度,从 0x000 到 0xFFF,同样 0x800 为力矩命令 0,因此上位机下发指令时,需要把指令偏移 0X800,力矩命令满量程不同型号电机不同

例如电路设定为 2A ,电流满量程查表为 4A, 电流码值为 0xC00

设置的电流码 = （电流/电流满量程）*0x800 + 0x800(无单位)

读取的电流值 = (code – 0x800)/0x800 * 电流满量程（单位 A）

位置环增益和位置环阻尼恒为正数,不作偏移处理,需要根据不同的电机型号和负载情况调整数值

增益数值 0 – 4095

阻尼数值 0 - 4095 无单位

不同电机模式,速度,力矩命令的定义会发生变化

在位置速度力矩三闭环模式下,速度命令代表电机在位置控制下可达到的最大速度,力矩命令代表电机可达到的最大力矩

在速度力矩环下，速度命令代表电机运行速度，力矩（电流）代表电机在该速度下运行，能提供的最大电流

在力矩环下，力矩（电流）代表电机在该力矩下运行

任何模式电机上传数值均为实时数值

命令实例 ， 位置环下 ,以 100rad/s ,最大电流 2A,移动到 180 度, , kp 设定为 0x100, kd 设定为 0x020 发送命令为

C0 00 C0 01 00 02 0C 00

电机应答帧

a.普通模式

Byte1	Byte2	Byte3	Byte4	Byte5	Byte6
电机模块 ID,用于主机鉴别	当前位置高 8 位	当前位置低 8 位	当前速度高 8 位	当前速度低 4 位 + 当前力矩高 4 位	当前力矩低 8 位

位置,速度,力矩的计算同发送命令一致,需要各自去掉偏移

读取的电机位置 = ((code – 0X8000)/0X8000)*360*Pmax（单位度）

读取的速度值 = （code – 0x800）/0x800 * 速度满量程（单位 rad/s）

读取的电流值 = (code – 0x800)/0x800 * 电流满量程（单位 A）

2. Can 扩展模式（增量位置模式）

扩展模式与普通模式不同，电机反馈位置从 2Byte 扩展为 4byte，单圈编码为 20bits，最大圈数为最大 +-2048 圈，超出此计数范围编码数值将循环计数，主机可自行扩展读取的圈数。

0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFC

启动电机,电机按设定好的模式和位置,速度,力矩指令启动运行,在扩展模式下，位置模式电机启动有所不同，电机将锁定在当前位置。速度和力矩模式与普通模式相同。

Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7	Byte8
位置命令高 8	位置命令低 8	速度命令高 8	速度命令低 4+ 位置增益高 4	位置增益低 8	位置阻尼高 8	位置阻尼低 4+ 力矩命令高 4	力矩命令低 8

位置命令共 16 位长度,从 0x0000 到 0xFFFF,0X8000 为位置命令 0,因此上位机下发指令时,需要把指令偏移 0X8000，位置命令满量程在串口中设置 Pmax，Pmax = 1 时,位置命令范围为+-1 圈,即 0XFFFF 代表正向旋转 1 圈,0x0000 代表反向旋转一圈,当 Pmax >1 时，位置命令范围变为+-Pmax 圈，扩展模式下位置命令为增量值，即电机相对当前位置正向或者反向移动的角度

例如设置电机相对当前位置正向移动 180 度，反向移动同理

设置的位置码值 = （电机角度/Pmax*360）*0x8000 + 0x8000 （无单位）

扩展模式电机应答帧

Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7	Byte8

电机模块ID,用于主机鉴别	当前位置15-8位	当前位置7-0位	当前速度高8位	当前速度低4位 + 当前力矩高4位	当前力矩低8位	当前位置31-24位	当前位置23-16位

32 位位置数据为有符号数，1048576 对应电机单圈计数，扩展模式位置环可简单理解为步进模式，电机运行类似于步进电机，通过命令移动对应步距角

3.程序实例 C

主机使用以下代码和电机模块通讯(位置模式)，发送数据时，因为存在 12 位和 16 位的数据，所以需要进行移位和字节的拼接，C 语言例程如下

定义发送接收的数据结构

```
uint8_t g_transmit_message.tx_data[8];
uint8_t g_receive_message.rx_data[8];

CanSendProcess()
{
        s_p_int       = 0x8000;//位置，0x8000 = 0 点
        s_v_int       = 0x900; //速度，速度数值 =  （0x900-0x800）/0x800*速度满量程
        s_Kp_int      = 0x800; //位置 Kp，  0x800 = 2048 = 50%
        s_Kd_int      = 0x800; //位置 Kd，  0x800 = 2048 = 50%
        s_c_int       = 0x900; //力矩，力矩数值=（0x900-0x800）/0x800*力矩满量程
        g_transmit_message.tx_data[0] = s_p_int>>8;
        g_transmit_message.tx_data[1] = s_p_int&0xFF;
        g_transmit_message.tx_data[2] = s_v_int>>4;;
        g_transmit_message.tx_data[3] = ((s_v_int&0xF)<<4) + (s_Kp_int >>8);
        g_transmit_message.tx_data[4] = s_Kp_int &0xFF;
        g_transmit_message.tx_data[5] = s_Kd_int>>4;
        g_transmit_message.tx_data[6] = ((s_Kd_int &0xF)<<4) + (s_c_int >>8);
        g_transmit_message.tx_data[7] = s_c_int&0xFF;;
        CanSend();
}
```

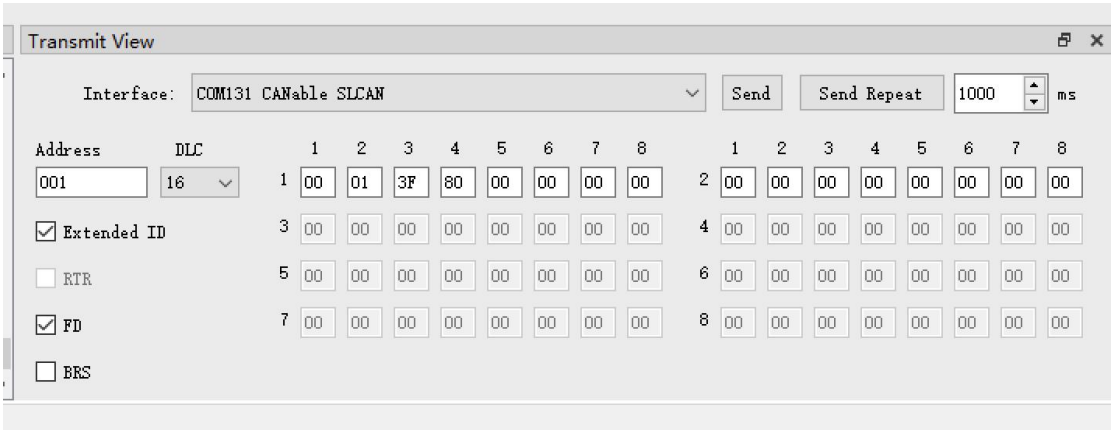
接收数据解析

```
CanReceiveData()
{
    motor_position=(g_receive_message.rx_data[1]<<8)+g_receive_message.rx_data[2];
    motor_velocity= (int16_t)(((g_receive_message.rx_data[3]&0xFF)<<4)
                    +((g_receive_message.rx_data[4]&0xF0)>>4))-2048;
    motor_current  = (int16_t)(((g_receive_message.rx_data [4]&0x0F)<<8)
                    + g_receive_message.rx_data [5])-2048;
}
```

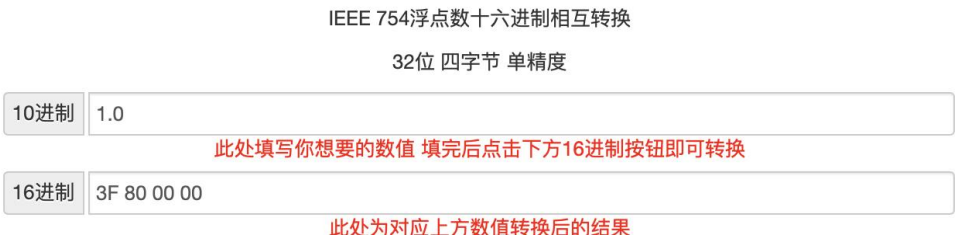
七、电机 Canfd 模式控制指南

在电机的串口中设置 CanFd En 打开,帧格式为 CanFd 标准帧
16byte 发送字节 16byte 接收字节
在 Canfd 工作模式下,单圈分辨率为 20bit,即单圈编码数为 1048576
位置数据长度为 5Byte
速度数据长度为 4Byte
力矩(电流)数据长度为 2Byte
变量定义
Pos 位置
Vel 速度
Cur 力矩(电流)

扩展帧设置电机速度环 PI,电流环 PI ,PI 设置值掉电不存储,每次上电需要重新设置



第一字节固定为 0x00,第二字节为 0x01 – 0x04,依次对应 speed_kp,speed_ki,current_kp,current_ki
第三到第六字节为设置的数值,数值是相对值, 实际 pi 值为预设 pi 数值*设置值,如设置值=1.0,那么 pi 值保持不变,设置的数值为 ieee754 32 位浮点,高位在前,如图所示 3F 80 00 00 = 1.0,浮点转换方法可自行查阅



普通 canfd 帧字节定义

发送帧

Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
Pos Bit 39 - 32	Pos Bit 31 -24	Pos Bit 23 - 16	Pos Bit 15 - 8	Pos Bit 7 - 0	Vel Bit 31 - 24	Vel Bit 23 - 16	Vel Bit 15 - 8

Byte8	Byte9	Byte10	Byte11	Byte12	Byte13	Byte14	Byte15
Vel Bit7 - 0	Cur Bit 15 - 8	Cur Bit 7 - 0	Mode Sel	Run Cmd	Kp	Kd	Zero

Mode Sel 运行模式选择，可在任意时刻切换

0 = 力矩

1= 速度

2 = 位置

RunCmd 1 = 电机启动

0 = 电机停止，并处于自由状态

Kp 位置环 Kp，0-255，数值越大电机位置环响应越快

Kd 位置环 Kd，0-255，数值越大电机位置环阻尼越大，运动过冲越小

Zero 清除多圈计数命令，每次往此字节写 1，设定电机当前位置为机械零位,同时清除多圈计数,需要在电机未使能的状态下执行,此位置掉电不保存，注意设置完成后此位置需要置零，否则电机将一直设置零点

应答帧

Byte0	Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7
Pos Bit 39 - 32	Pos Bit 31 -24	Pos Bit 23 - 16	Pos Bit 15 - 8	Pos Bit 7 - 0	Vel Bit 31 - 24	Vel Bit 23 - 16	Vel Bit 15 - 8

Byte8	Byte9	Byte10	Byte11	Byte12	Byte13	Byte14	Byte15
Vel Bit7 - 0	Cur Bit 15 - 8	Cur Bit 7 - 0	Mode	Run Status	Reserve	温度,单位度	从机 ID

Mode 电机当前运行模式

0 = 力矩

1= 速度

2 = 位置

RunStatus 电机当前运行状态

1 = 电机启动

0 = 电机停止，并处于自由状态

温度 -125-125 度，上传电机当前温度

从机 ID 用于多台从设备通讯时，标识出发送方地址

Position 计算方法,先把 40 位的数据做符号扩展到 64 位

```
Int64_t Position;
        if((g_receive_message.rx_data[0]>>7) == 0)
        {
            Position = (((int64_t)0x000000)<<40) |\
                       (((int64_t)g_receive_message.rx_data[0])<<32) | \
                       (((int64_t)g_receive_message.rx_data[1])<<24) | \
                       (((int64_t)g_receive_message.rx_data[2])<<16) | \
                       (((int64_t)g_receive_message.rx_data[3])<<8)  | \
                       (((int64_t)g_receive_message.rx_data[4])<<0);
        }
        else
        {
            Position = (((int64_t)0xFFFFFF)<<40) |\
                       (((int64_t)g_receive_message.rx_data[0])<<32) | \
                       (((int64_t)g_receive_message.rx_data[1])<<24) | \
                       (((int64_t)g_receive_message.rx_data[2])<<16) | \
                       (((int64_t)g_receive_message.rx_data[3])<<8)  | \
                       (((int64_t)g_receive_message.rx_data[4])<<0);
        }
```

电机单圈位置 = Position % 1048576

电机移动圈数 = Positon / 1048576

Velocity 计算方法

```
Int32_t Velocity =  (((int32_t)g_receive_message.rx_data[5])<<24) |\
                    (((int32_t)g_receive_message.rx_data[6])<<16) |\
                    (((int32_t)g_receive_message.rx_data[7])<<8)  |\
                    (((int32_t)g_receive_message.rx_data[8])<<0)  ;
```

电机转速 Freq = (Velocity / 8388608)*1000Hz

由于转速是以电机电频率表示,所以换算到 RPM 时

RPM = Freq * 60 / 14，其中 14 是 3507 电机极对数,当使用其他极对数电机时,这个参数需要做相应修改

Current 计算方法

```
Int16_t Current =  (((int16_t)g_receive_message.rx_data[9])<<8)  |\
                   (((int16_t)g_receive_message.rx_data[10])<<0)  ;
```

电机电流 Cur = Current / 32768 * 100 (A)

八、电机配件使用指南（USB 转 CAN&CANFD 模块）

可使用上位机连接 USB 转 CANFD,使用串口直接发送 CAN&CANFD 数据帧控制电机和读取,

a) 按示意图所示连接线路

b) 打开电源，查看上位机设备管理器，找到对应设备串口号

c) 打开串口助手，此指南使用共享串口助手软件说明，其他软件原理相同

d) 打开串口，波特率设置为 1000000

e) 打开设备，依次发送三个指令，注意发送接收均需要设置为 ASCII 模式

S8\r

Y5\r

O\r\n

f) CAN 模式通讯

发送两次 t00180000800000000800\r ，即可有电机应答数据显示，此命令中 t0018 为固定帧头，代表 can 标准帧，长度 8byte，后面 16 个 ascII 字符代表了 8byte 的 can 数据字节，ascII ‘00’ = hex 0x00

g) CANFD 模式通讯

发送两次 d001A00000000000000000000000200000000\r ，即可有电机应答数据显示，此命令中 d001A 为固定帧头，代表 canfd 帧 id001,长度 16byte，后面 32 个 ascII 字符代表了 16byte 的 canfd 数据字节，ascII ‘00’ = hex 0x00

使用前需要确定当前电机设置的通讯模式是 can&canfd 中哪一个

九、电机配件使用指南（485&脉冲转 canfd 模块）

a) 脉冲转 canfd

模块默认为脉冲转 canfd 模式运行，在这个模式下，必须一个模块对应一个电机。电机 ID 必须设置为 1，电机 master ID 必须设置为 100.

模块输入

1) En 使能.高电平有效，电压范围 3.3v-24V.

En 保持高电平，模块使用步进模式，在系统上电后，电机将锁定在当前位置。

En 输入脉冲，电机将回到机械位置零点并锁定

En 低电平，电机失能，处于自由状态。

2) Dir 方向，高电平有效，电压范围 3.3v-24V.

Dir 高电平，电机接收脉冲后顺时针旋转

Dir 低电平，电机接收脉冲后逆时针旋转

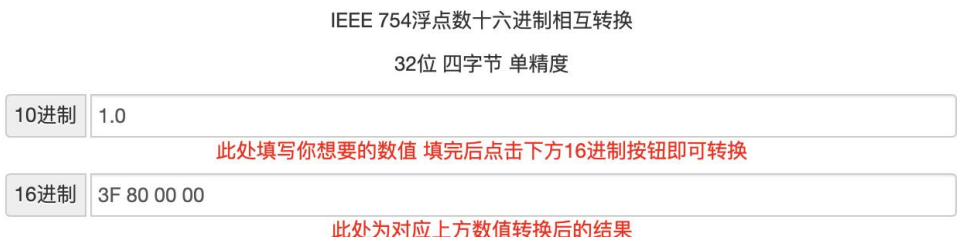
3) Pulse 脉冲输入 脉冲上升沿有效，电压范围 3.3v-24V.

接收一个脉冲，电机按设定的方向移动一个步距角。

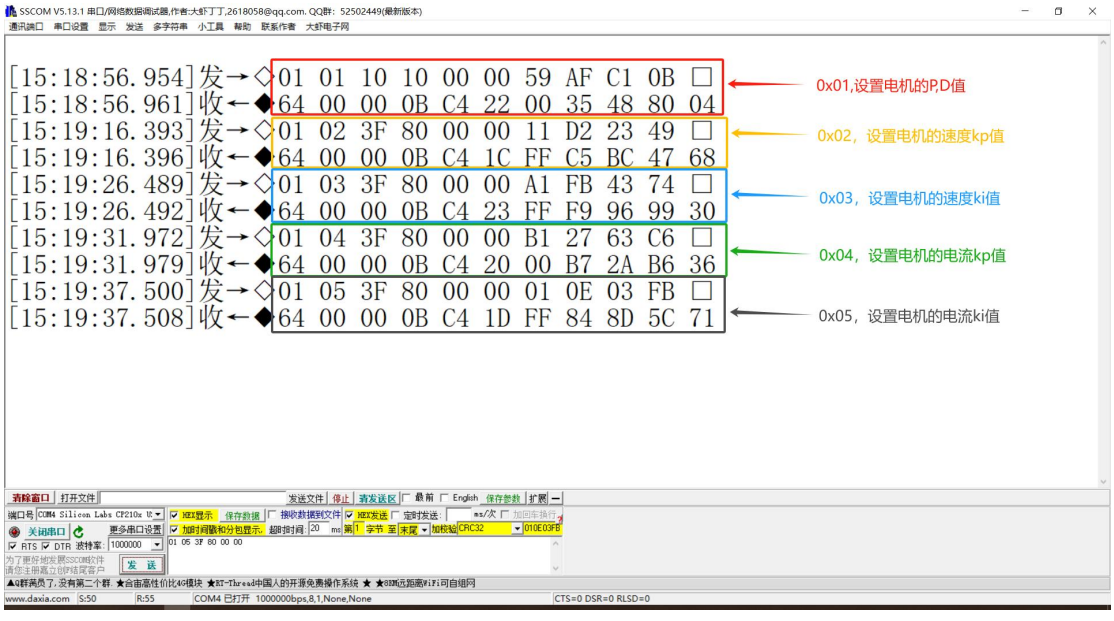
步距角的大小由拨码开关码值确定， 步距角 = 码值 * 0.125 度，脉冲转换模式下拨码开关必须至少有一位到 1

码值为计算方式：利用 sw1~6 的二进制数转成的 10 进制。On 为 1，off 为 0。例如 sw1=1 (on) 000001，则码值为 1，步距角 = 1 * 0125°，为 0.125°。若 sw1~6 都为 1（on）111111，则码值为 63，步距角 = 63 * 0.125°，为 7.875°。

此模块还可以调整电机刚度（PID）。第一字节固定为 0x00,第二字节为 0x01 – 0x05,依次对应设置 P-D 值，speed_kp,speed_ki,current_kp,current_ki。除 0x01 外，其余第三到第六字节为设置的数值,数值是相对值,设置值=1.0,那么 pi 值保持不变,设置的数值为 ieee754 32 位浮点,高位在前,如图所示 3F 80 00 00 = 1.0,浮点转换方法可自行查阅。



设置 P,D 值。发送帧为 01 01 10 10 00 00，第二字节为 0x01 设置电机第三字节和第四字节 P 值和 D 值。后两位字节填充 0x00，范围位 0~255。



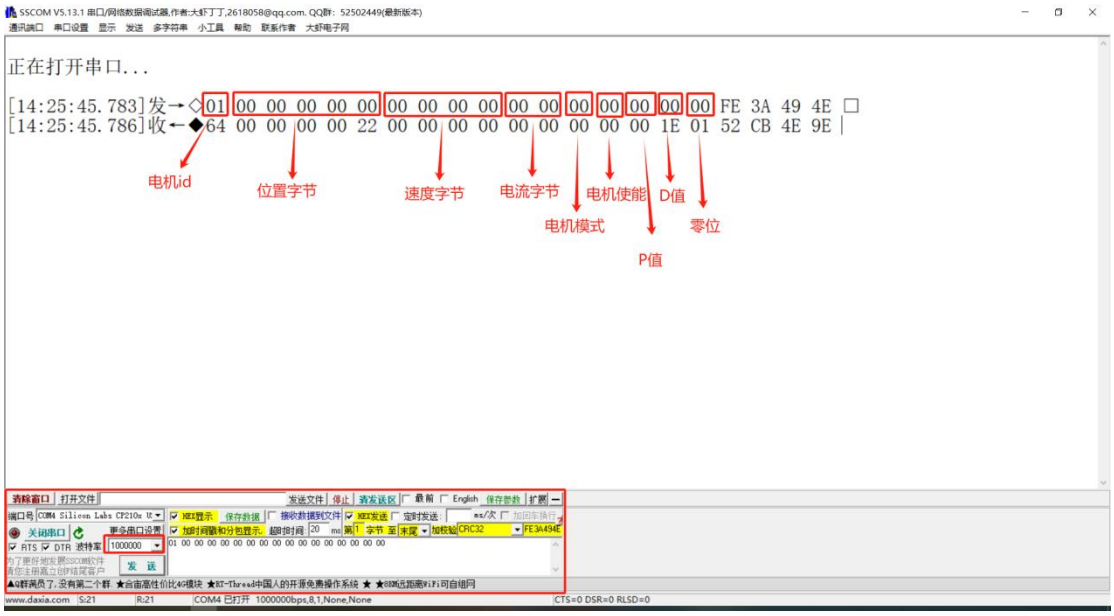
b) 485 转 canfd

转换模块接收到有效 485 数据时，切换为 485 到 canfd 转换模式，根据拨码开关状态确定 485

转换模式，485 转换模式可实现多机通讯，不要求电机从机 ID，模块自动转发 485 数据到 canfd 总线，同时把总线数据传输到 485 接口。当拨码开关全 off 状态时，切换到 modbus’ RTU 模式，任意一位处于 on 状态，切换到透明传输模式。

1) 透明传输模式

透明传输波特率固定为 1M，数据由 ID + 16Byte canfd 帧 + CRC32 组成，id 为电机 canid，canfd 数据帧解析参考 canfd 说明部分，crc32 为 ieee 标准 crc32 算法，注意高低位顺序。



发送帧：01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FE 3A 49 4E

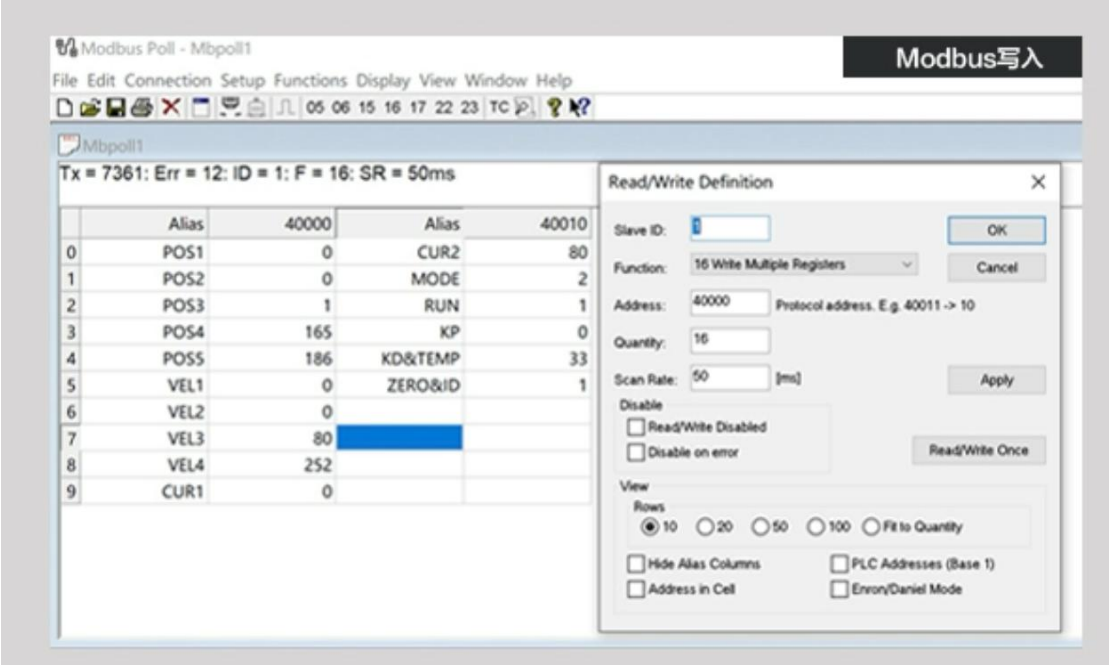
0x01	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0xFE	0x3A	0x49	0x4E
电机id	位置字节						速度字节				电流字节		模式	使能	P值	D值	零位	CRC32校验			

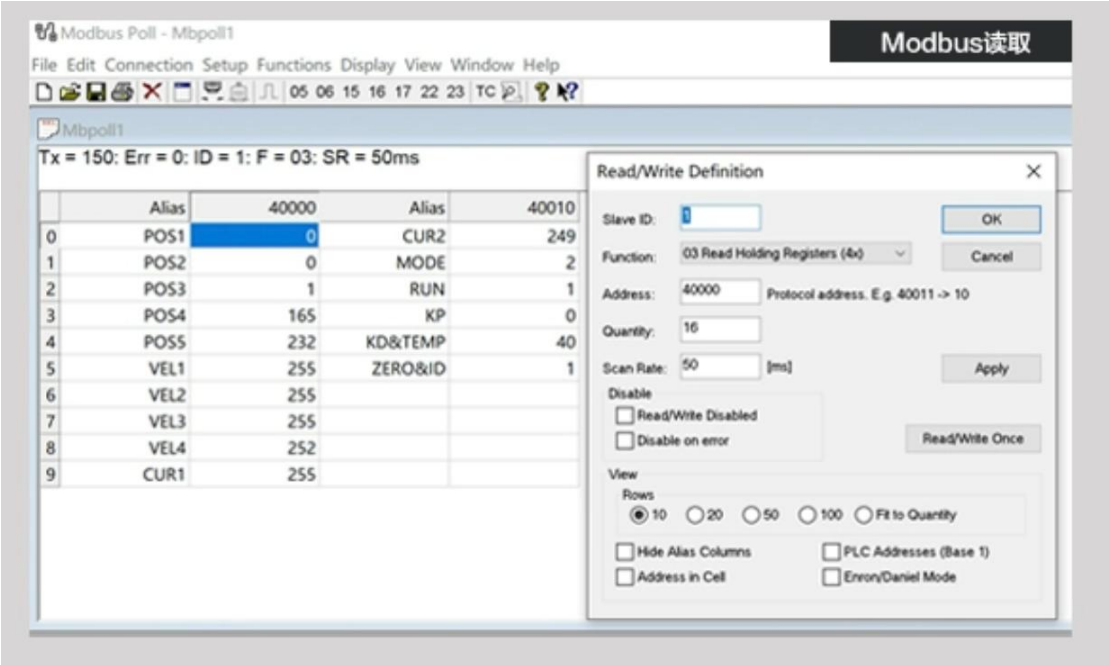
返回帧：64 00 00 00 00 22 00 00 00 00 00 00 00 00 00 1E 01 52 CB 4E 9E

0x64	0x00	0x00	0x00	0x00	0x22	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x1E	0x01	0x52	0xCB	0x4E	0x9E
master id	位置字节						速度字节				电流字节		模式	使能	暂未开放	温度	电机id	CRC32校验			

2) ModbusRTU 模式

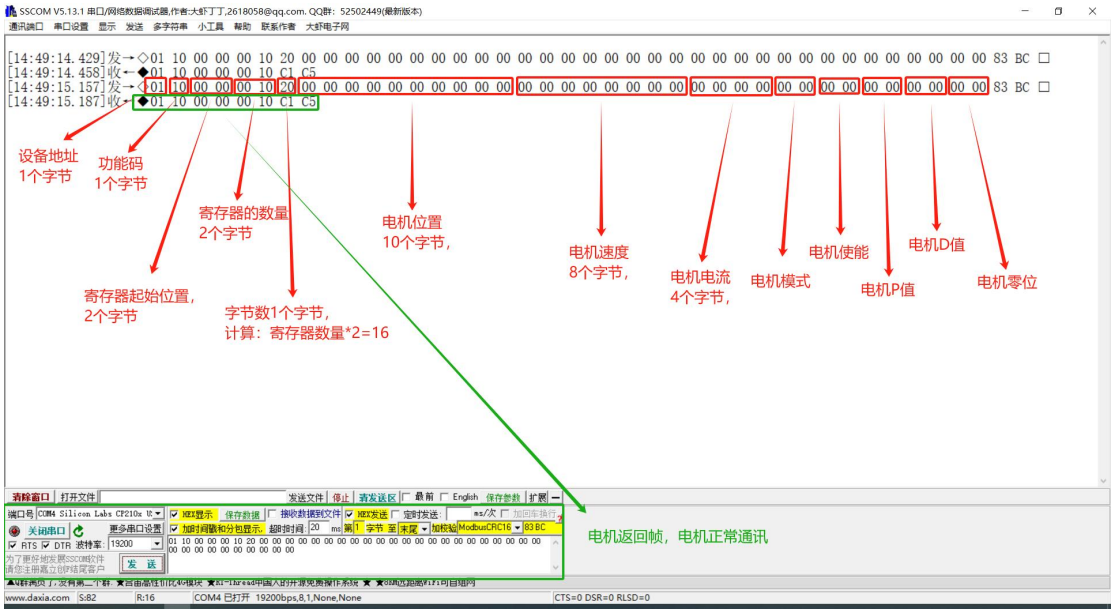
ModbusRTU 模式波特率固定为 19200，Modbus 协议中，Slaveaddr 对应电机 canid，使用 modbus 40001 到 40016 寄存器代表 canfd 16byte 数据，读写兼容 modbusRTU 协议，如果用户不使用 PLC，需计算 modbus CRC16 尾缀。注意 modbus 协议寄存器为 16 位，canfd 通讯协议字节为 8 位，因此转换到 modbus 寄存器是，高 8 位默认用 0x00 填充，低 8 位使用 canfd 协议字节。





3) ModbusRTU 模式(此模式是本司上位机软件下 ModeBusRTU 转 canfd，目前仅支持操作功能码为 0x10 和 0x03)

ModbusRTU 模式波特率固定为 19200，设备地址对应电机 canid，寄存器的起始地址为 0x0000，用 16 个寄存器代表 canfd 16byte 数据，读写兼容 modbusRTU 协议。注意 modbus 协议寄存器为 16 位，canfd 通讯协议字节为 8 位，modbus 寄存器是高 8 位默认用 0x00 填充，低 8 位使用 canfd 协议字节。



发送帧（功能码位 0x10，即写数据）

01 10 00 00 00 10 20 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 83 BC

0x01	0x10	0x00 00	0x00 10	0x20	0x00 00	0x00 00	0x00 00	0x00 00	0x00 00	0x00 00	0x00 00	0x00 00	0x00 00
设备地址	功能码	起始地址	寄存器数量	字节数	pos1	pos2	pos3	pos4	pos5	vel1	vel2	vel3	vel4
					0x00 00	0x00 00	0x00 00	0x00 00	0x00 00	0x00 00	0x00 00	0x83 BC	
					cur1	cur2	mode	run	KP	KD	zero	crc16	

发送给电机的实际数据是 16 位只取 pos1~zero 的低地址。

应答帧 01 10 00 00 00 10 C1 C5

0x01	0x10	0x00 00	0x00 10	0x C1C5
设备地址	功能码	起始地址	寄存器数量	crc16

发送帧（功能码为 0x03，即读数据）

01 03 00 00 00 10 44 06

0x01	0x03	0x00 00	0x00 10	0x44 06
设备地址	功能码	起始地址	寄存器数量	crc16

应答帧：

01 03 20 00 FF 00 FB 00 27 00 F9 00 F5 00 00 00 00 00 00 00 00 00 00 00 08 00 00 00 00 00 00 00 1F 00 01 74 DB

0x01	0x03	0x20	0x00 FF	0x00 FB	0x00 27	0x00 F9	0x00 00	0x00 00	0x00 00	0x00 00	0x00 00
设备地址	功能码	字节数	pos1	pos2	pos3	pos4	pos5	vel1	vel2	vel3	vel4
			0x00 00	0x00 08	0x00 00	0x00 00	0x00 00	0x00 1F	0x00 01	0x74 DB	
			cur1	cur2	mode	run	暂未开放	温度	从机地址	crc16	

同理，实际数据是 16 位只取 pos1~zero 的低地址。

Crc32 计算代码：

1.查表法

```
static const unsigned int crc32tab[] = {
  0x00000000L, 0x77073096L, 0xee0e612cL, 0x990951baL,
  0x076dc419L, 0x706af48fL, 0xe963a535L, 0x9e6495a3L,
  0x0edb8832L, 0x79dcb8a4L, 0xe0d5e91eL, 0x97d2d988L,
  0x09b64c2bL, 0x7eb17cbdL, 0xe7b82d07L, 0x90bf1d91L,
  0x1db71064L, 0x6ab020f2L, 0xf3b97148L, 0x84be41deL,
  0x1adad47dL, 0x6ddde4ebL, 0xf4d4b551L, 0x83d385c7L,
  0x136c9856L, 0x646ba8c0L, 0xfd62f97aL, 0x8a65c9ecL,
  0x14015c4fL, 0x63066cd9L, 0xfa0f3d63L, 0x8d080df5L,
  0x3b6e20c8L, 0x4c69105eL, 0xd56041e4L, 0xa2677172L,
  0x3c03e4d1L, 0x4b04d447L, 0xd20d85fdL, 0xa50ab56bL,
  0x35b5a8faL, 0x42b2986cL, 0xdbbbc9d6L, 0xacbcf940L,
  0x32d86ce3L, 0x45df5c75L, 0xdcd60dcfL, 0xabd13d59L,
  0x26d930acL, 0x51de003aL, 0xc8d75180L, 0xbfd06116L,
  0x21b4f4b5L, 0x56b3c423L, 0xcfba9599L, 0xb8bda50fL,
  0x2802b89eL, 0x5f058808L, 0xc60cd9b2L, 0xb10be924L,
  0x2f6f7c87L, 0x58684c11L, 0xc1611dabL, 0xb6662d3dL,
```

0x76dc4190L, 0x01db7106L, 0x98d220bcL, 0xefd5102aL,
0x71b18589L, 0x06b6b51fL, 0x9fbfe4a5L, 0xe8b8d433L,
0x7807c9a2L, 0x0f00f934L, 0x9609a88eL, 0xe10e9818L,
0x7f6a0dbbL, 0x086d3d2dL, 0x91646c97L, 0xe6635c01L,
0x6b6b51f4L, 0x1c6c6162L, 0x856530d8L, 0xf262004eL,
0x6c0695edL, 0x1b01a57bL, 0x8208f4c1L, 0xf50fc457L,
0x65b0d9c6L, 0x12b7e950L, 0x8bbeb8eaL, 0xfcb9887cL,
0x62dd1ddfL, 0x15da2d49L, 0x8cd37cf3L, 0xfbd44c65L,
0x4db26158L, 0x3ab551ceL, 0xa3bc0074L, 0xd4bb30e2L,
0x4adfa541L, 0x3dd895d7L, 0xa4d1c46dL, 0xd3d6f4fbL,
0x4369e96aL, 0x346ed9fcL, 0xad678846L, 0xda60b8d0L,
0x44042d73L, 0x33031de5L, 0xaa0a4c5fL, 0xdd0d7cc9L,
0x5005713cL, 0x270241aaL, 0xbe0b1010L, 0xc90c2086L,
0x5768b525L, 0x206f85b3L, 0xb966d409L, 0xce61e49fL,
0x5edef90eL, 0x29d9c998L, 0xb0d09822L, 0xc7d7a8b4L,
0x59b33d17L, 0x2eb40d81L, 0xb7bd5c3bL, 0xc0ba6cadL,
0xedb88320L, 0x9abfb3b6L, 0x03b6e20cL, 0x74b1d29aL,
0xead54739L, 0x9dd277afL, 0x04db2615L, 0x73dc1683L,
0xe3630b12L, 0x94643b84L, 0x0d6d6a3eL, 0x7a6a5aa8L,
0xe40ecf0bL, 0x9309ff9dL, 0x0a00ae27L, 0x7d079eb1L,
0xf00f9344L, 0x8708a3d2L, 0x1e01f268L, 0x6906c2feL,
0xf762575dL, 0x806567cbL, 0x196c3671L, 0x6e6b06e7L,
0xfed41b76L, 0x89d32be0L, 0x10da7a5aL, 0x67dd4accL,
0xf9b9df6fL, 0x8ebeeff9L, 0x17b7be43L, 0x60b08ed5L,
0xd6d6a3e8L, 0xa1d1937eL, 0x38d8c2c4L, 0x4fdff252L,
0xd1bb67f1L, 0xa6bc5767L, 0x3fb506ddL, 0x48b2364bL,
0xd80d2bdaL, 0xaf0a1b4cL, 0x36034af6L, 0x41047a60L,
0xdf60efc3L, 0xa867df55L, 0x316e8eefL, 0x4669be79L,
0xcb61b38cL, 0xbc66831aL, 0x256fd2a0L, 0x5268e236L,
0xcc0c7795L, 0xbb0b4703L, 0x220216b9L, 0x5505262fL,
0xc5ba3bbeL, 0xb2bd0b28L, 0x2bb45a92L, 0x5cb36a04L,
0xc2d7ffa7L, 0xb5d0cf31L, 0x2cd99e8bL, 0x5bdeae1dL,
0x9b64c2b0L, 0xec63f226L, 0x756aa39cL, 0x026d930aL,
0x9c0906a9L, 0xeb0e363fL, 0x72076785L, 0x05005713L,
0x95bf4a82L, 0xe2b87a14L, 0x7bb12baeL, 0x0cb61b38L,
0x92d28e9bL, 0xe5d5be0dL, 0x7cdcefb7L, 0x0bdbdf21L,
0x86d3d2d4L, 0xf1d4e242L, 0x68ddb3f8L, 0x1fda836eL,
0x81be16cdL, 0xf6b9265bL, 0x6fb077e1L, 0x18b74777L,
0x88085ae6L, 0xff0f6a70L, 0x66063bcaL, 0x11010b5cL,
0x8f659effL, 0xf862ae69L, 0x616bffd3L, 0x166ccf45L,
0xa00ae278L, 0xd70dd2eeL, 0x4e048354L, 0x3903b3c2L,
0xa7672661L, 0xd06016f7L, 0x4969474dL, 0x3e6e77dbL,
0xaed16a4aL, 0xd9d65adcL, 0x40df0b66L, 0x37d83bf0L,
0xa9bcae53L, 0xdebb9ec5L, 0x47b2cf7fL, 0x30b5ffe9L,

```
 0xbdbdf21cL, 0xcabac28aL, 0x53b39330L, 0x24b4a3a6L,
 0xbad03605L, 0xcdd70693L, 0x54de5729L, 0x23d967bfL,
 0xb3667a2eL, 0xc4614ab8L, 0x5d681b02L, 0x2a6f2b94L,
 0xb40bbe37L, 0xc30c8ea1L, 0x5a05df1bL, 0x2d02ef8dL
};


unsigned int crc32( const unsigned char *buf, unsigned int size)
{
        unsigned int i, crc;
        crc = 0xFFFFFFFF;


        for (i = 0; i < size; i++)
          crc = crc32tab[(crc ^ buf[i]) & 0xff] ^ (crc >> 8);

        return crc^0xFFFFFFFF;
}
```

直接计算法

```
static unsigned int crc32_for_byte(unsigned int r)
{
   for(int j = 0; j < 8; ++j)
      r = (r & 1? 0: (unsigned int)0xEDB88320L) ^ r >> 1;
   return r ^ (unsigned int)0xFF000000L;
}


unsigned int crc32( const unsigned char *buf, unsigned int size)
{
   static unsigned int table[0x100];
   unsigned int crc = 0;
   unsigned short i;
   if(!*table)
   for(i = 0; i < 0x100; ++i)
         table[i] = crc32_for_byte(i);
   for(i = 0; i < size; ++i)
      crc = table[(unsigned char)crc ^ ((unsigned char*)buf)[i]] ^ crc >> 8;
   return crc;
}
```

Crc16 计算代码：

1.直接算法：

```
unsigned int ModBusCRC16(unsigned char *data, unsigned int len)
```

```
{
    unsigned int i, j, tmp, CRC16;

    CRC16 = 0xFFFF;                    //CRC 寄存器初始值
    for (i = 0; i < len; i++)
    {
        CRC16 ^= data[i];
        for (j = 0; j < 8; j++)
        {
            tmp = (unsigned int)(CRC16 & 0x0001);
            CRC16 >>= 1;
            if (tmp == 1)
            {
                CRC16 ^= 0xA001;      //异或多项式
            }
        }
    }
    return CRC16;
}
```

2.半字节查表法：

```
const unsigned int CRC_16_Tab[16] =
{
  0x0000, 0xCC01, 0xD801, 0x1400, 0xF001, 0x3C00, 0x2800, 0xE401,
  0xA001, 0x6C00, 0x7800, 0xB401, 0x5000, 0x9C01, 0x8801, 0x4400
};

unsigned int ModBusCRC16(unsigned char *data, unsigned char len)
{
  unsigned char i,temp;
  unsigned int crc_16 = 0xffff;
  for(i=0;i<len;i++)
  {
    temp = ((unsigned char)(crc_16&0x000F))^(*data&0x0F);
    crc_16 >>= 4;
    crc_16 ^= CRC_16_Tab[temp];
    temp = ((unsigned char)(crc_16&0x000F))^(*data>>4);
    crc_16 >>= 4;
    crc_16 ^= CRC_16_Tab[temp];
  }
  return crc_16;
}
```

3.全字节查表法：

```
/* Table of CRC values for high–order byte */
const unsigned char auchCRCHi[] = {
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01,
0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81,
0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01,
0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01,
0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01,
0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
0x40
} ;

/* Table of CRC values for low–order byte */
const unsigned char auchCRCLo[] = {
0x00, 0xC0, 0xC1, 0x01, 0xC3, 0x03, 0x02, 0xC2, 0xC6, 0x06, 0x07, 0xC7, 0x05, 0xC5, 0xC4,
0x04, 0xCC, 0x0C, 0x0D, 0xCD, 0x0F, 0xCF, 0xCE, 0x0E, 0x0A, 0xCA, 0xCB, 0x0B, 0xC9, 0x09,
0x08, 0xC8, 0xD8, 0x18, 0x19, 0xD9, 0x1B, 0xDB, 0xDA, 0x1A, 0x1E, 0xDE, 0xDF, 0x1F, 0xDD,
0x1D, 0x1C, 0xDC, 0x14, 0xD4, 0xD5, 0x15, 0xD7, 0x17, 0x16, 0xD6, 0xD2, 0x12, 0x13, 0xD3,
0x11, 0xD1, 0xD0, 0x10, 0xF0, 0x30, 0x31, 0xF1, 0x33, 0xF3, 0xF2, 0x32, 0x36, 0xF6, 0xF7,
0x37, 0xF5, 0x35, 0x34, 0xF4, 0x3C, 0xFC, 0xFD, 0x3D, 0xFF, 0x3F, 0x3E, 0xFE, 0xFA, 0x3A,
0x3B, 0xFB, 0x39, 0xF9, 0xF8, 0x38, 0x28, 0xE8, 0xE9, 0x29, 0xEB, 0x2B, 0x2A, 0xEA, 0xEE,
0x2E, 0x2F, 0xEF, 0x2D, 0xED, 0xEC, 0x2C, 0xE4, 0x24, 0x25, 0xE5, 0x27, 0xE7, 0xE6, 0x26,
0x22, 0xE2, 0xE3, 0x23, 0xE1, 0x21, 0x20, 0xE0, 0xA0, 0x60, 0x61, 0xA1, 0x63, 0xA3, 0xA2,
0x62, 0x66, 0xA6, 0xA7, 0x67, 0xA5, 0x65, 0x64, 0xA4, 0x6C, 0xAC, 0xAD, 0x6D, 0xAF, 0x6F,
0x6E, 0xAE, 0xAA, 0x6A, 0x6B, 0xAB, 0x69, 0xA9, 0xA8, 0x68, 0x78, 0xB8, 0xB9, 0x79, 0xBB,
0x7B, 0x7A, 0xBA, 0xBE, 0x7E, 0x7F, 0xBF, 0x7D, 0xBD, 0xBC, 0x7C, 0xB4, 0x74, 0x75, 0xB5,
0x77, 0xB7, 0xB6, 0x76, 0x72, 0xB2, 0xB3, 0x73, 0xB1, 0x71, 0x70, 0xB0, 0x50, 0x90, 0x91,
0x51, 0x93, 0x53, 0x52, 0x92, 0x96, 0x56, 0x57, 0x97, 0x55, 0x95, 0x94, 0x54, 0x9C, 0x5C,
0x5D, 0x9D, 0x5F, 0x9F, 0x9E, 0x5E, 0x5A, 0x9A, 0x9B, 0x5B, 0x99, 0x59, 0x58, 0x98, 0x88,
0x48, 0x49, 0x89, 0x4B, 0x8B, 0x8A, 0x4A, 0x4E, 0x8E, 0x8F, 0x4F, 0x8D, 0x4D, 0x4C, 0x8C,
0x44, 0x84, 0x85, 0x45, 0x87, 0x47, 0x46, 0x86, 0x82, 0x42, 0x43, 0x83, 0x41, 0x81, 0x80,
0x40
} ;
```

```
unsigned short ModBusCRC16(unsigned char *puchMsg,unsigned short usDataLen)
{
    unsigned char uchCRCHi = 0xFF ; /* high byte of CRC initialized */
    unsigned char uchCRCLo = 0xFF ; /* low byte of CRC initialized */
    unsigned uIndex ;                /* will index into CRC lookup table */
    while (usDataLen--)              /* pass through message buffer */
    {
        uIndex = uchCRCHi ^ *puchMsg++ ; /* calculate the CRC */
        uchCRCHi = uchCRCLo ^ auchCRCHi[uIndex] ;
        uchCRCLo = auchCRCLo[uIndex] ;
    }
    return (uchCRCHi << 8 | uchCRCLo) ;
}
```

十、电机 PWM 控制输入输出参数表

PWM 控制表	位置最大	位置最小	正向旋转速度	反向旋转速度	正向力矩	反向力矩
PWM 输入脉宽	2ms	1ms	1.55ms - 2ms	1ms - 1.45ms	1.55ms - 2ms	1ms - 1.45ms
HO4307	正方向一周	负方向一周	0.1Hz---585 Hz	-585Hz -0.1Hz	0A ----0.74A	-0.74A---0A
HO5515	正方向一周	负方向一周	0.1Hz---585 Hz	-585Hz -0.1Hz	0A ----0.74A	-0.74A---0A
HO6213	正方向一周	负方向一周	10Hz----90 Hz	-90Hz -10Hz	0A-----3.71 A	-3.71A---0A
HO7213 标准	正方向一周	负方向一周	10Hz----90 Hz	-90Hz -10Hz	0A-----3.71 A	-3.71A---0A
HO7213 高速	正方向一周	负方向一周	20Hz----495 Hz	-495Hz -20Hz	0A-----3.71 A	-3.71A---0A
HO8110	正方向一周	负方向一周	10Hz----450 Hz	-450Hz -10Hz	0A-----3.71 A	-3.71A---0A
HO10010	正方向一周	负方向一周	10Hz----450 Hz	-450Hz -10Hz	0A-----14.8 5A	-14.85A----- 0A
HO10025	正方向一周	负方向一周	10Hz----450 Hz	-450Hz -10Hz	0A-----14.8 5A	-14.85A----- 0A

十一、 电机 Encoder 输出模式参数

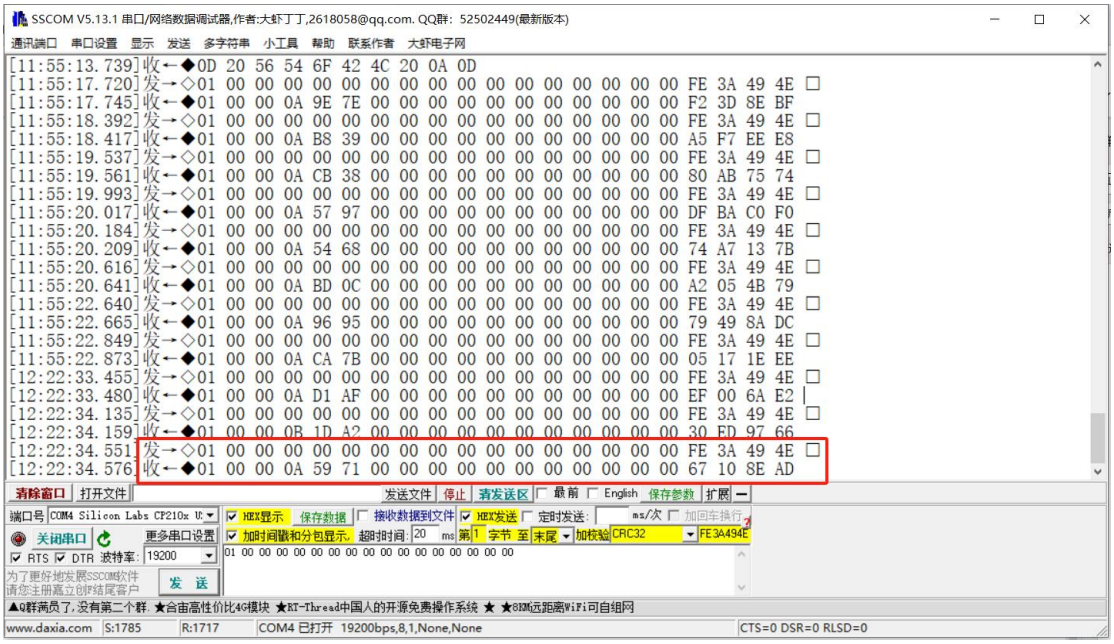
电机 Encoder2 作为输出口使用时,可输出脉冲信号,通过配置电机参数中的 STEP_CONFIG 实现

当 STEP_CONFIG 设置为 0 时,Encoder2 为输入口,和减速器末端编码器组成中空关节模块,减速器末端编码器需使用我公司提供的编码器.

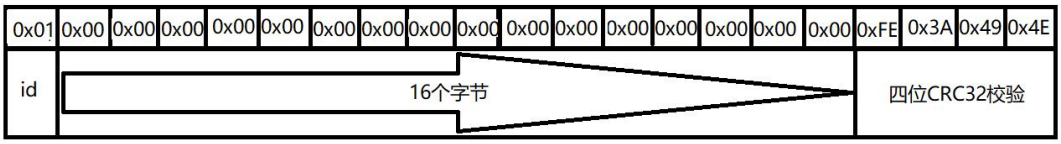
当 STEP_CONFIG 设置为大于 0 的数时(整数),令此数据为 n,电机将间隔 n 度在 encoder1 引脚输出 1ms 的脉冲,同时每次当电机运动到单圈零点时在 encoder2 引脚输出 1ms 的脉冲

十二、 中空编码器

使用串口调试助手。



发送帧为 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FE 3A 49 4E；发送数据为 16 进制数，



电机应答帧为 01 00 00 0A 59 71 00 00 00 00 00 00 00 00 00 00 67 10 8E AD；电机回馈的字节帧。