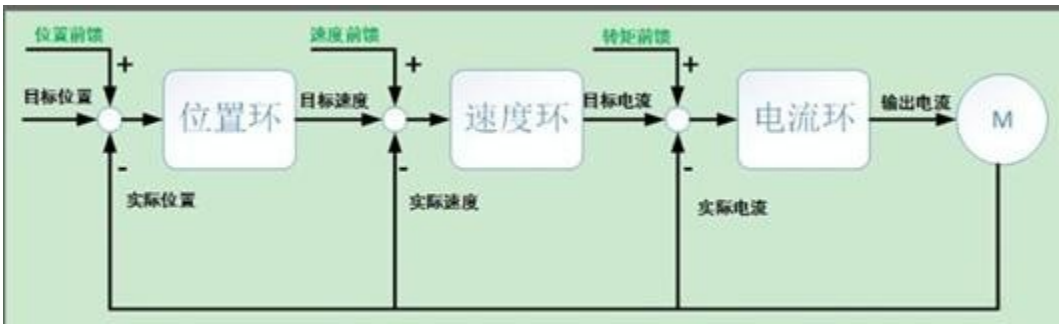
Instructions for use of hollow series motor control

catalogs

I. Motor Control Circuit



For permanent magnet brushless motors, under general conditions, the motor current is proportional to the motor torque, and the conversion formula is torque T = Kt*I

Where Kt is the motor torque constant, I is the current value, the motor torque constant can be obtained by consulting the motor parameter table, the following is not specified, the current is the torque.

Position loop using PD controller, PD adjustable, the host can specify the target position, specify the maximum speed and maximum current to control the motor movement, the actual speed and actual current of the motor operation and load-related, the specified maximum current needs to be greater than the maximum load of the motor movement process.

Speed loop using PI controller, PI adjustable, host can specify the target speed, maximum current control motor movement

The current loop uses a PI controller with an adjustable PI and a target current that can be specified by the host computer. The motor runs at a load-dependent speed and if the load is less than the torque, the motor will run at the maximum speed for that voltage and load condition.

II. ttl serial/rs485 control and configuration

1. The serial port is the motor configuration interface, which can configure the parameters related to motor control. The parameters configured by the serial port,

without any special description, are stored in the internal memory of the motor, which is still saved after power down, and will be operated according to the configuration after the next power up.

2. Serial port baud rate is fixed at 19200, use ascII character mode for interaction, host sends commands to end with \n\r, the host computer software provided by our company has automatically added the command suffix, if users use microcontroller or third-party serial port software, pay attention to add their own \n\r suffix, all the commands are lowercase characters, the following are not special instructions, do not include the \n\r terminator

III. Serial control of motors

X in the main menu, enter m/bXXX.X to control the motor according to the motor operating mode, e.g., m10.0 has different interpretations in different modes, the motor rotates positively up to 10.0 degrees in the position mode, the motor operates at 10 Hz in the speed mode, and the motor operates at 10 A positive torque in the torque mode.

If the input data is a floating point number, it needs to be entered with a decimal point.

Input r The motor stops and enters the free state, note that the serial control of the motor is only used to test the motor, the function is very limited, you need to use the can interface to use the motor in the product.

IV. Serial port configurable data

1. Motor operation mode, enter p to enter mode selection
   a) Torque Mode
   b) Speed Mode
   c) Position Mode

   The motor operation mode is mainly used to control the motor directly through serial commands, and the configured mode can be temporarily modified by the can/canfd command.

2. Motor parameters, enter s to enter parameter configuration
   a) Current loop bandwidth, i.e. current loop P value, 100-2000
   b) Velocity loop bandwidth, i.e., velocity loop P-value (not available at this time)
   c) Position Loop Gain, Position Loop P Value 0- 100
   d) Position ring damping, position ring D value 0-100
   e) For multi-turn motion setting, Pmax should be set to 1.0 for all hollow series motors.
   f) Canfd mode enable, select to use can mode or canfd mode, 1 for canfd mode, 0 for can mode
   g) Canid motor can id value, represents the motor receive filter id value, only data frames with can frame id = motor id will be received by the motor and generate an answer frame
   h) Can host id, the host id on the bus, the answer frame after the motor receives the data frame will be sent to the host corresponding to the host id
   i) CW config, forward and reverse direction control bits, not user modifiable, indicates the current phase sequence of the motor.

V. serial port to set the motor mechanical zero point or clear the mechanical zero point

1. Use the serial port to set the motor zero point, SetZeroPoint and ClrZeroPoint of the menu correspond to setting the zero point and clearing the zero point to the factory state, respectively.
2. The angle of the set zero point relative to the factory zero point will be displayed on

the serial port menu interface, the unit of display is degree, note that the zero point value needs to be

To range from 5-355 degrees, the user can read the displayed zero value to determine if the setting was successful.

3. The zero point set by Can&Canfd is not saved inside the motor, and after powering up again, the motor will perform an offset and move according to the zero point value displayed on the serial port.
4. The zero position is valid for a single turn, the zero point of multi-turn or with a deceleration mechanism needs to be offset by the user.

VI. Motor Can Mode Control Guide

1. Can Protocol Overview

Can rate 1M bit/s, the hardware includes CAN_H and CAN_L and GND three signal pins, general bus topology needs to be in the sender and receiver side each parallel connection of a 120-ohm terminating resistor, through the serial port to set up the communication rate of the motor module Can, without special instructions, all the motors do not support the adjustment of the communication baud rate

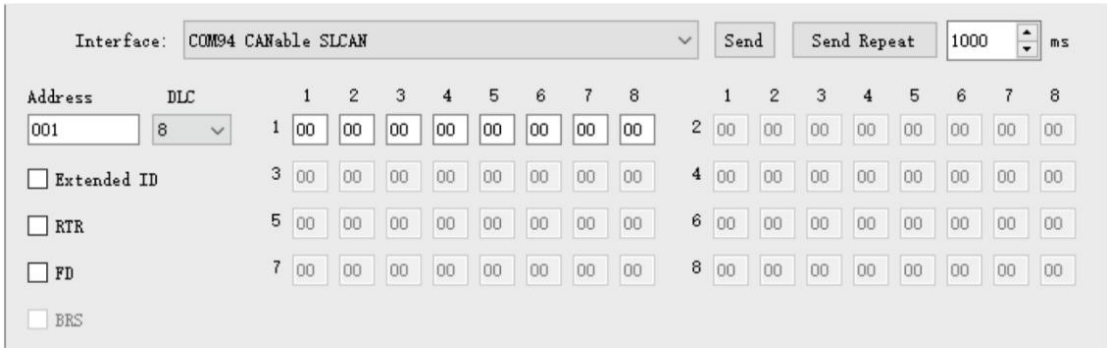
The motor module are all belong to the slave, the use of one should answer a communication mode, that is, the host sends commands to the slave to answer the command, if the host does not send the command corresponding to the ID, the slave is in a silent state will not take the initiative to send data to the bus, from the receipt of commands to the host to answer the motor module requires a delay time of 500us, so the system currently supports the maximum communication interval is 500us.

The ID of the motor module itself is configured via the serial port, any number between 0 and 100 can be selected, note that no duplicate IDs can appear on the same bus.

Standard CAN ID Frame Bit Definition, Address is the ID value for this description

The host ID is also written into the motor module through the serial port, so that when the motor module receives the Can data frame that meets the specification, it will send an answer data frame to the set host ID, and the default host ID is 100, i.e., the master controller can receive the answer from the motor module by setting its own ID to 100.

Standard can frame sending settings



Can uses standard frame and extended frame for communication, where standard frame is used for general communication and extended frame is used for some special settings, standard frame and extended frame use the same ID, for users who need more communication bits, they can use canfd protocol (16byte data frame length).

There are two types of Can control modes, normal mode is used by default for power-on, switching Can control mode is realized by sending specific extended frames to the motor, if you need to use the extended mode, you need to manually switch once every time the motor is powered on.

The extended frame bits are defined as follows

Total 8 bytes, use first byte second bit to set

extended mode and standard mode byte

definitions

Byte 1

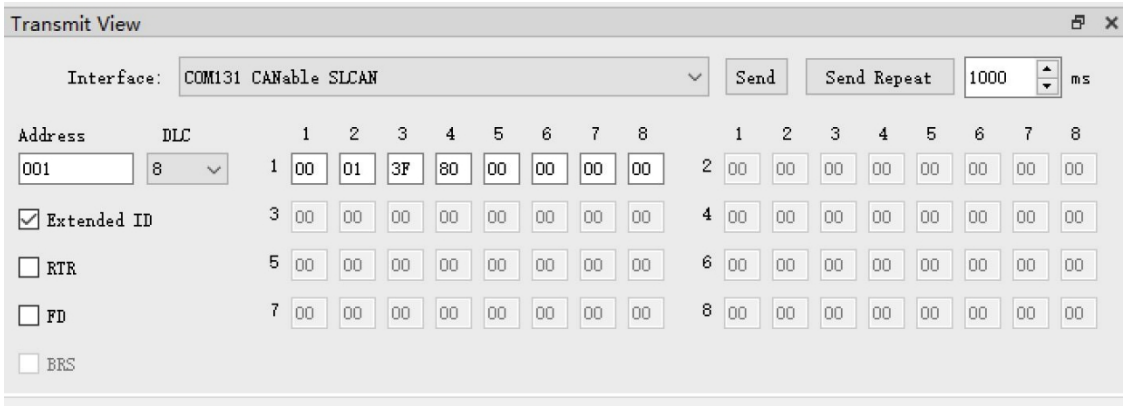
Bit8	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1

0 for unused	0 for unused	0 for unused	0 for unused	0 for unused	Zero setting required for unused	Extended Address Mode 1 = Normal mode 0 = Extended mode	Zero setting required for unused

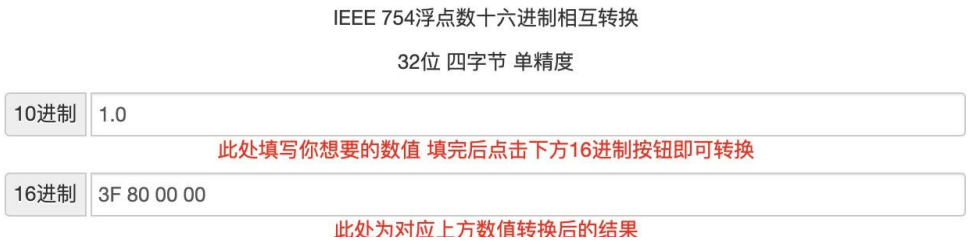
Byte 2-8 not used

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。

Extended frame to set the motor speed ring PI, current ring PI, PI setting value will not be stored after power down, and need to be reset every time power up.



The first byte is fixed at 0x00, the second byte is 0x01 - 0x04, which corresponds to speed_kp, speed_ki, current_kp, current_ki in that order.
The third to the sixth byte for the setting of the value, the value is a relative value, the actual pi value for the preset pi value * set value, such as the set value = 1.0, then the pi value remains unchanged, set the value of ieee754 32-bit floating point, the high position in the front, as shown in the figure 3F 80 00 00 = 1.0, the floating point conversion method can be accessed by their own



1. Can Normal Mode

(Absolute Position Mode) Special Instructions

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

The motor is started according to the set mode, position, speed and torque commands, and the setting method is described in the general commands.

If the motor does not receive any set commands before startup, the motor performs an unknown state of motion, so it is necessary to set the position speed torque command for the motor to run before startup

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

Stop the motor, the motor has no external power output.

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

Switching to torque control mode

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

Switch to speed torque control mode

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

Switch to position velocity torque triple closed loop mode

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

Setting the current motor position as the mechanical zero position and clearing the multi-turn count, which needs to be executed when the motor is not energized, and this position will not save the common instruction when power down.

a. Set the motor position, speed, torque, position ring Kp, Kd, where Kp affects the position ring response time, Kd affects the position ring motion damping, and there is no integral control term for the position ring. Speed ring Kp Ki Kd is not open, torque ring Kp can be set directly through serial port, torque ring Ki Kd is not open.

Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7	Byte8
Position Command High 8	Position Command Low 8	Speed command high 8	Speed command low 4+ Position gain high 4	Position gain low 8	Position damping high 8	Low positional damping 4+ Torque command high 4	Torque command low 8

Position command is 16 bits long, from 0x0000 to 0xFFFF, 0X8000 is the position command 0, so when the host computer sends the command, you need to offset the command 0X8000, the full scale of the position command is set in the serial port Pmax , Pmax = 1, the range of the position command is +-1 circle, i.e., 0XFFFF represents one circle of forward rotation, 0x0000 represents one circle of reverse rotation, when Pmax >1, the range of the position command becomes +- Pmax circle. When Pmax >1, the range of position command becomes +-Pmax turns.

For example, set the motor position to 180 degrees

Set position code value = (motor angle/Pmax*360)*0x8000 + 0x8000 (unitless)

Read motor position = ((code - 0X8000)/0X8000)*360*Pmax (unit degree) If the motor movement position is out of the range of +- one revolution, the uploaded motor position value is constant 0xffff or 0x0000, which can be used to judge whether the motor movement is out of range or not.

The speed command is 12 bits long, from 0x000 to 0xFFF, the same 0x800 is the speed command 0, so when the upper computer sends the command, you need to offset the command by 0X800, the full scale of the speed command is different for different types of motors, the speed command is different for different types of motors, the speed command is different for different types of motors.

For example, if the speed is set to +100 rad/s, and the full scale speed is 200 rad/s, the

speed code value is (100/200)*2048.
+ 2048 = 3072 = 0xC00

The speed is set to -100 rad/s, the full scale speed is 200 rad/s, and the speed code value is (-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）

The torque command is 12 bits long, from 0x000 to 0xFFF, the same 0x800 is the torque command 0, so when the upper computer sends the command, you need to offset the command by 0X800, the full range of the torque command is different for different types of motors.

For example, if the circuit is set to 2A and the full scale current is 4A, the current code value is 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）

Position loop gain and position loop damping are always positive, without offset, and need

增益数值 0 – 4095

阻尼数值 0 - 4095 无单位

to be adjusted according to different motor models and load conditions.

Definitions of speed and torque commands change for different motor modes.

In the position, speed and torque three closed-loop mode, the speed command represents the maximum speed that the motor can achieve under position control, and the torque command represents the maximum torque that the motor can achieve.

In the speed-torque loop, the speed command represents the speed at which the motor is running, and the torque (current) represents the maximum current that the motor can deliver if it is running at that speed

In the torque ring, the torque (current) represents the motor running at that torque Any mode motor upload values are real-time values

Command example, under position ring, move to 180 degrees at 100rad/s, max. current 2A, kp is set to 0x100, kd is set to 0x020 Send command as follows

c0 00 c0 01 00 02 0c 00

motor response frame

a. General mode

Byte1	Byte2	Byte3	Byte4	Byte5	Byte6
Motor module ID for host identification	Current position high 8 bits	Current position low 8 bits	Current speed high 8 bits	Current speed low 4 bits + Current torque high 4 positions	Current torque low 8 bits

The calculation of position, speed and torque is the same as that of the send command, but each of them needs to remove the offset.

Motor position read = ((code - 0X8000)/0X8000)*360*Pmax (in degrees) Speed value read = (code - 0x800)/0x800 * speed full scale (in rad/s) Current value read = (code - 0x800)/0x800 * Current full scale (in A)

2. Can Extended Mode (Incremental Position Mode)

Extended mode differs from normal mode in that the motor feedback position is extended from 2Byte to 4byte, the single turn is encoded in 20bits, and the maximum number of turns is max.

+-2048 laps, beyond this counting range the encoded value will be counted cyclically, the host computer can extend the number of laps read by itself. 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFC

Start the motor, the motor starts to run according to the set mode and position, speed and torque commands. In the extended mode, the position mode motor starts differently and the motor will be locked at the current position. Speed and torque modes are the same as normal mode.

Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7	Byte8

Position Command High 8	Position command low 8	Speed command high 8	Speed command low 4+ Position gain high 4	Position gain low 8	Position damping high 8	Low positional damping 4+ Torque command high 4	Torque command low 8

Position command is 16 bits long, from 0x0000 to 0xFFFF, 0X8000 is the position command 0, so when the host computer sends the command, you need to offset the command 0X8000, the full scale of the position command is set in the serial port Pmax, Pmax = 1, the position command range is +-1 circle, that is, 0XFFFF represents one circle of forward rotation, 0x0000 represents one circle of reverse rotation, when Pmax >1, the position command range becomes +-Pmax circle, the extended mode position command is an incremental value, that is, the motor moves forward or reverse angle relative to the current position. When Pmax >1, the range of position command becomes +-Pmax circle, the position command is incremental value in extended mode, i.e., the angle of the motor moving forward or backward relative to the current position.

For example, set the motor to move 180 degrees forward with respect to the current position, and the same for reverse movement.

Set position code value = (motor angle/Pmax*360)*0x8000 + 0x8000 (unitless) Extended mode motor answer frame

Byte1	Byte2	Byte3	Byte4	Byte5	Byte6	Byte7	Byte8
Motor Module ID, for host identification	Current Position 15-8 bits	Current position 7-0 bit	Current speed 8 bits higher	Current speed 4 digits lower + Current torque high 4 digits	Current torque 8 digits lower	Current Position 31-24	Current position 23-16 positions

The 32-bit position data is a signed number, 1048576 corresponds to a single revolution count of the motor. The extended mode position ring can be simply understood as a stepping mode, where the motor operates similar to a stepping motor, and the corresponding step angle is shifted by commands.

3.Program example C

The host computer uses the following code to communicate with the motor module

(position mode). When sending data, because there are 12-bit and 16-bit data, it is necessary to perform shifting and byte splicing, and the C language routine is as follows

Define the data structure for sending and receiving

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

CanSendProcess()
{
        s_p_int      = 0x8000;//position, 0x8000 = 0 points
                     s_v_int= 0x900; //speed, speed value = (0x900-
        0x800)/0x800*speed full scale s_Kp_int       = 0x800; //position Kp,
        0x800 = 2048 = 50%
        s_Kd_int    = 0x800; // location Kd, 0x800 = 2048 = 50%
                     s_c_int= 0x900; //torque, torque value = (0x900-
        0x800)/0x800*torque full scale 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 &
        gt;>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().
}
```

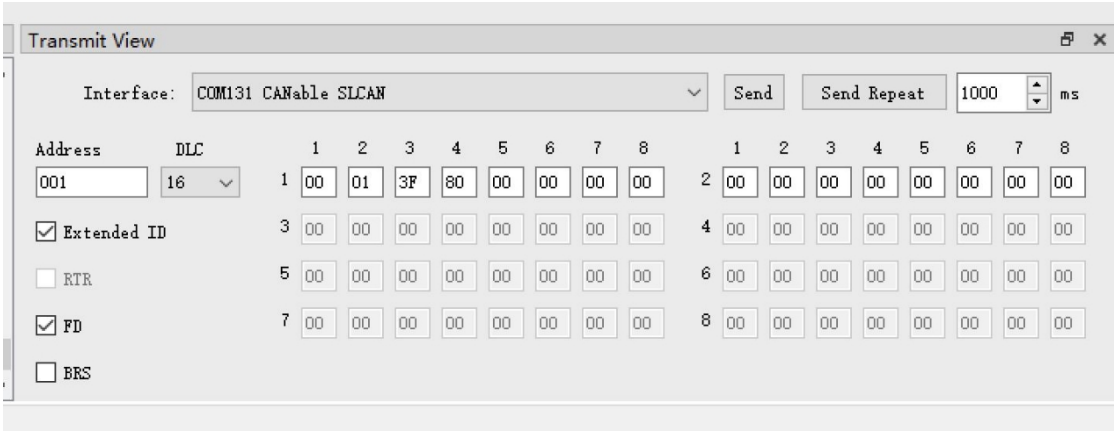
Received data parsing

```
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;
}
```

VII. Motor Canfd Mode Control Guide

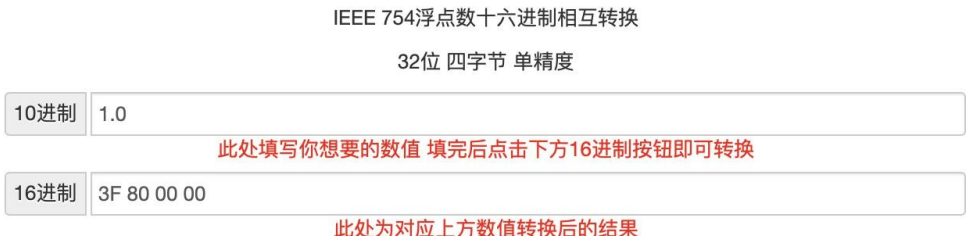
Set CanFd En on in the serial port of the motor, and
the frame format is CanFd standard frame 16byte
Send byte 16byte Receive byte
In Canfd operating mode, the resolution of a single turn is
20bit, i.e., the number of encoded single turns is
1048576 and the length of position data is 5Byte.
Speed data length is 4 Byte
Torque (current) data
length is 2Byte Variable
Definition
Pos
Positio
n Vel
Speed
Cur Torque (current)

Extended frame to set the motor speed ring PI, current ring PI, PI setting value will not be stored after power down, and need to be reset every time power up.



The first byte is fixed to 0x00, the second byte is 0x01 - 0x04, corresponding to speed_kp, speed_ki, current_kp, current_ki third to sixth byte for the setting of the value of the value, the value of the value is relative, the actual pi value for the preset pi value * setting value, such as setting value = 1.0, then the pi value remains unchanged, set the value of ieee754 32-bit floating point, the high first, as shown in the figure 3F 80 00 00 = 1.0, floating-point, floating-point, the high first, as shown in the figure 3F 80 00 00 = 1.0, floating-point Then the pi value remains unchanged, set the

value of ieee754 32-bit floating point, the high bit in front, as shown in the figure 3F 80 00 00 = 1.0, the floating point conversion method can be viewed on their own!



Normal canfd Frame Byte Definition

transmitter frame

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 operation mode selection, can be switched at any time

0 = Moment

1= Velocity

2 = Position RunCmd 1 = Motor start

0 = motor stopped and free

Kp Position loop Kp, 0-255, the higher the value the faster the motor position loop response

Kd Position ring Kd, 0-255, the higher the value the higher the damping of the motor position ring and the lower the motion overshoot

Zero clear multiturn count command, write 1 to this byte every time, set the current position of motor as mechanical zero position, and clear the multiturn c o u n t , need to be executed in the state of motor is not enabled, this position will not be saved when power off, note that this position needs to be set to zero after setting is completed, otherwise the motor will be set to zero point all the time.

response frame

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	Temperature in degrees	Slave ID

Mode Current operating mode of the motor

0 = Moment

1= Velocity

2 = pos

RunStatus Current motor

run status

1 = motor start

0 = motor stopped and free

Temperature -125-125 degrees, upload current motor temperature

Slave ID is used to identify the sender's address when communicating with multiple slave devices. Position is calculated by symbolically expanding the 40-bit data to 64 bits.

```
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])<<
        els            8)                                        \
        e              (((int64_t)g_receive_message.rx_data[4])<&
        {              lt;0);



                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[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.
                           int64_t)g_receive_message.rx_data[4])<<0).
```

Motor position per revolution = Position % 1048576 Number of motor revolutions = Positon / 1048576

Velocity Calculation Method

```
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.
                           message.rx_data[8])<<0) ;
```

Motor RPM Freq = (Velocity / 8388608)*1000Hz Since RPM is expressed as motor frequency, it is converted to RPM.

RPM = Freq * 60 / 14 , where 14 is the number of pole pairs of the 3507 motor, this parameter needs to be modified accordingly when using other pole pairs of motors.

Current Calculation Method

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

Motor Current Cur = Current / 32768 * 100 (A)

VIII. Motor Accessory User's Guide (USB to CAN & CANFD Modules)

USB to CANFD can be connected to the host computer, use the serial port to send CAN&CANFD data frames directly to control the motor and read.

a) Connect the wiring as shown in the schematic

b) Turn on the power, check the device manager of the host computer, find the corresponding device serial port number

c) Open the serial port assistant, this guide uses the shared serial port assistant software description, other software principle is the same

d) Open the serial port and set the baud rate to 1000000

e) Turn on the device, send three commands in turn, note that both send and receive need to be set to ASCII mode

S8\r

Y5\r

O\r\n

f) CAN mode communication

Send t00180000800000000000000800\r twice to have the motor answer data display, in this command t0018 is the fixed frame header, representing the can standard frame, the length is 8byte, the next 16 ascII characters represent the can data byte of 8byte, ascII '**OO**' = hex 0x00

g) CANFD mode communication

Send d001A000000000000000000000000000200000000\r twice, then the motor answer data will be displayed. d001A in this command is the fixed frame header, which represents the canfd frame id001, the length is 16byte, and the 32 ascII characters after it represent the 16byte bytes of canfd data. ascII '**OO**' = hex 0x00

Before use, you need to make sure which one of can&canfd is the communication mode set for the current motor.

IX. Motor Accessory User's Guide (485 & pulse to canfd module)

a) Pulse turn canfd

The module operates in pulse-to-canfd mode by default, in which one module must correspond to one motor. The motor ID must be set to 1 and the motor master ID must be set to 100.

Module Input

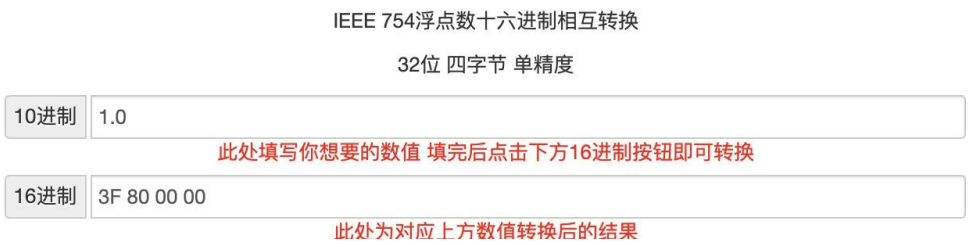
1) En Enable. Active high, voltage range 3.3v-24v.
   En held high, the module uses stepping mode and the motor will lock at the current position after the system is powered up. En Input pulse, the motor will return to the mechanical position zero point and lock
   En Low, the motor is de-energized and in the free state.
2) Dir direction, active high, voltage range 3.3v-24v. Dir high, motor receives pulse and rotates clockwise
   Dir low, motor receives pulse and rotates counterclockwise
3) Pulse Pulse Input Rising edge of pulse is active, voltage range 3.3v-24v. Receiving a pulse, the motor moves one step angle in the set direction.

The step angle is determined by the dipswitch code value, step angle = code value * 0.125 degrees, the dipswitch must have at least one digit to 1 in pulse transition mode.

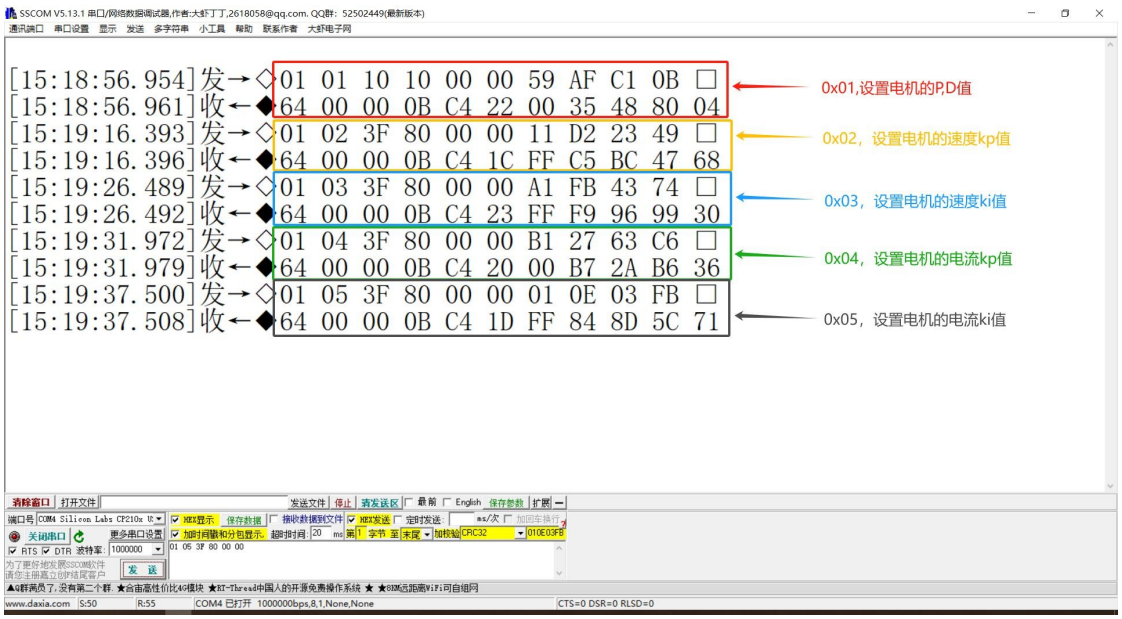
The code value is calculated by using the binary number of sw1~6 converted to decimal.on is 1, off is 0.Forexample,ifsw1=1(on)000001, then the code value is 1, and the step angle = 1 * 0125°, is 0.125°. If sw1~6 are all 1(on)111111, the code value is 1 and the step angle = 1 * 0125°, which is 0.125°.

The code value is 63 and the step angle = 63 * 0.125°, which is 7.875°.

This module also adjusts the motor stiffness (PID). The first byte is fixed to 0x00, the second byte is 0x01 - 0x05, corresponding to the setting of the P-D value, speed_kp, speed_ki, current_kp, current_ki. in addition to 0x01, the rest of the third to the sixth byte for the setting of the value, the value of the value is relative, set the value of = 1.0, then the pi value remains unchanged, set the value of ieee754 32-bit floating point, the high bit in the first, as shown in the figure. pi value remains unchanged, set the value of ieee754 32-bit floating point, the high bit in front, as shown in the figure 3F 80 00 00 = 1.0, floating-point conversion method can be consulted.



Set the P,D value. Send frame is 01 01 10 10 00 00, second byte is 0x01 Set motor third and fourth byte P value and D value. The last two bytes are padded with 0x00, range bits 0 to 255.



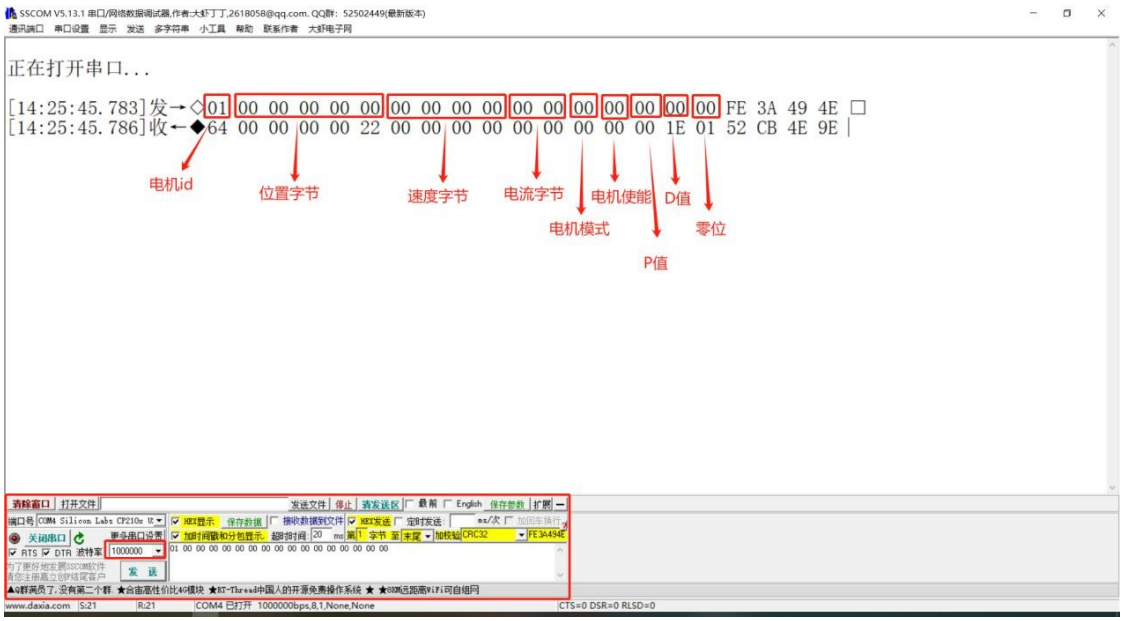
b) 485 to canfd

When the converter module receives valid 485 data, it switches to 485 to canfd conversion mode and determines the 485 conversion mode according to the status of the dipswitch.

Conversion mode, 485 conversion mode can realize multi-machine communication without requiring the ID of the motor slave, the module automatically forwards 485 data to the canfd bus, and at the same time transmits the bus data to the 485 interface. The module automatically forwards the 485 data to the canfd bus and transmits the bus data to the 485 interface at the same time. When the dip switch is all off, it switches to the modbus' RTU mode, and any bit is on, it switches to the transparent transmission mode.

1) transparent transfer mode

Transparent transmission baud rate is fixed at 1M, the data consists of ID + 16Byte canfd frame + CRC32, id is the canid of the motor, refer to the canfd description for canfd frame parsing, crc32 is the ieee standard crc32 algorithm, pay attention to the order of the high and low bits.



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

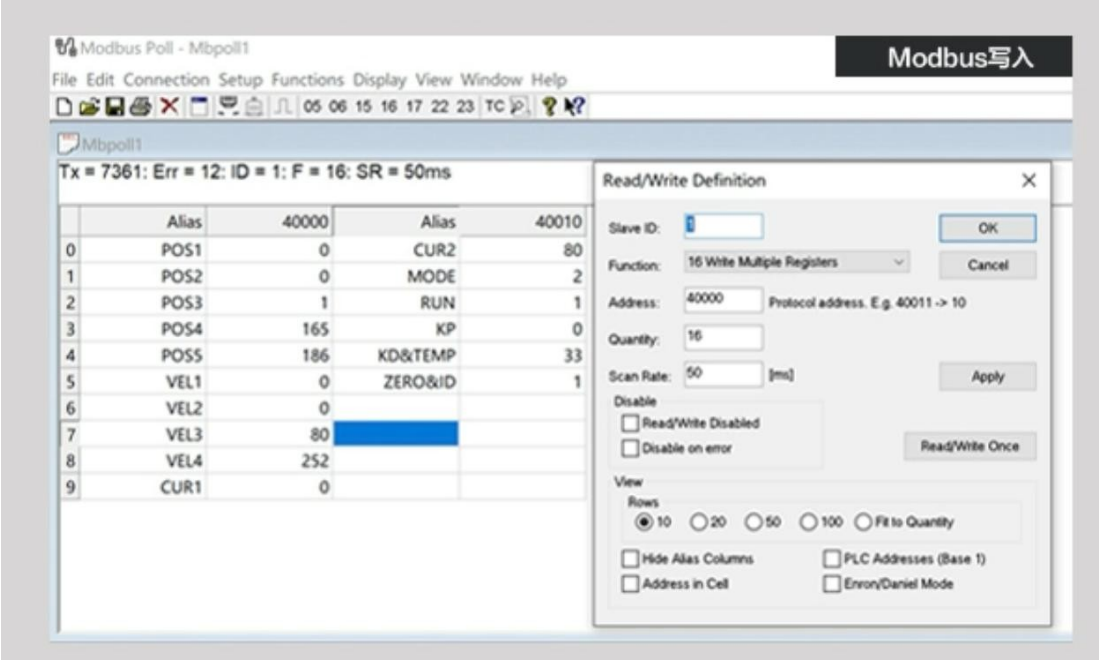
0x01	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00	0xFE	0x3A	0x49	0x4E
电机id	位置字节					速度字节				电流字节		模式	使能	P值	D值	零位	CRC32校验			

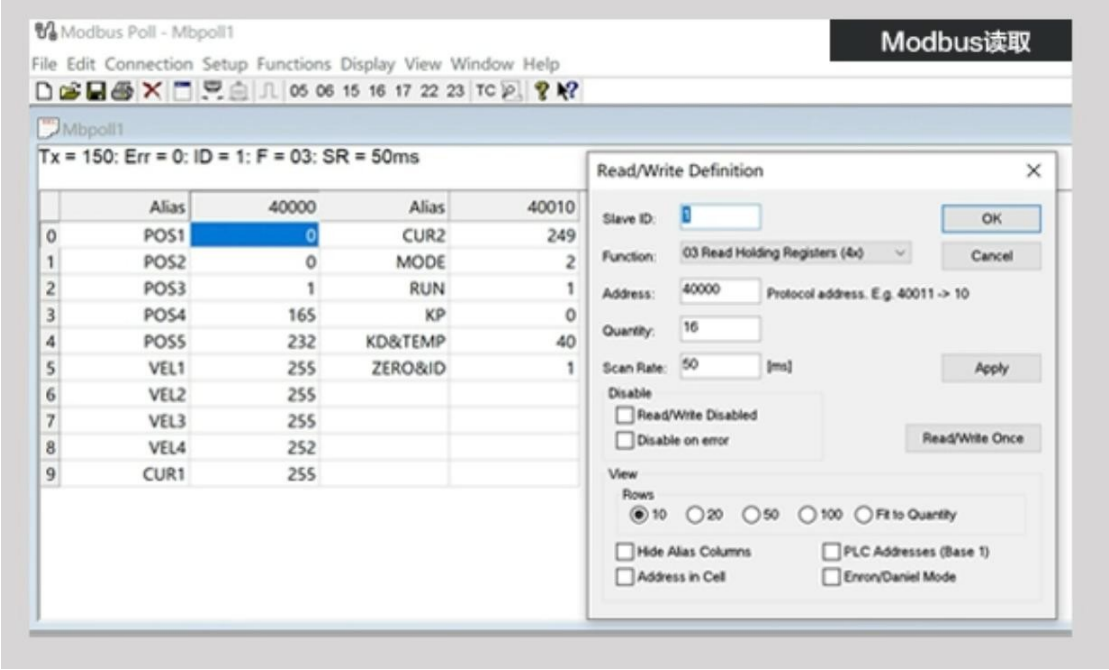
返回帧：64 00 00 00 00 22 00 00 00 00 00 00 00 00 00 1E 01 52 CB 4E 9E

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 mode

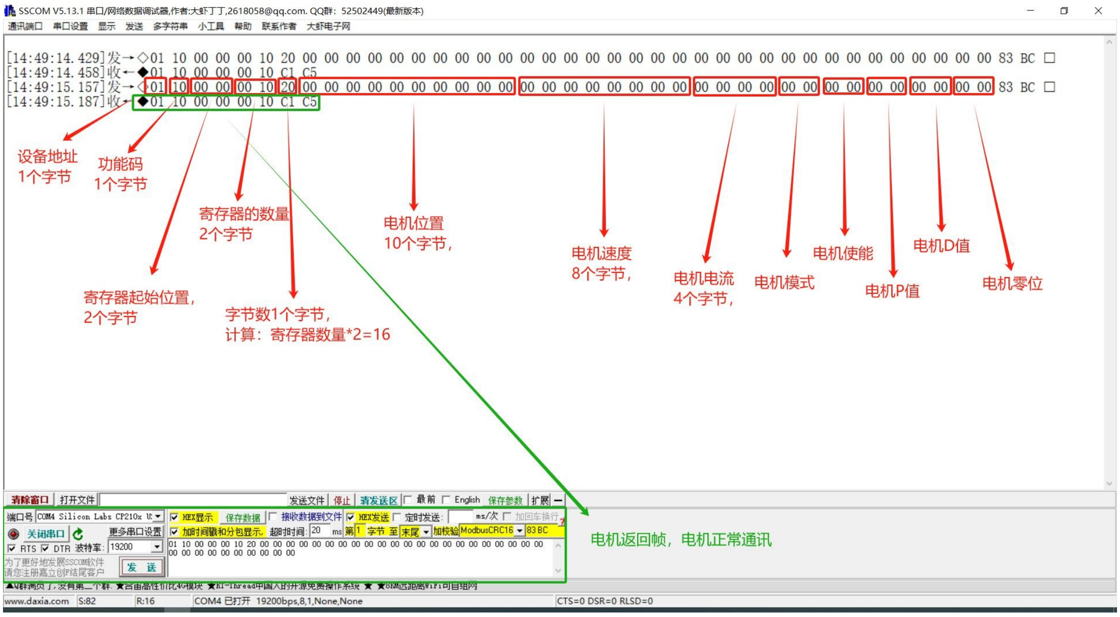
ModbusRTU mode baud rate is fixed at 19200, in Modbus protocol, Slaveaddr corresponds to motor canid, use modbus 40001 to 40016 registers to represent canfd 16byte data, read/write is compatible with modbusRTU protocol, if users don't use PLC, they need to calculate the modbus If user does not use PLC, need to calculate modbus CRC16 suffix. Note that modbus protocol register is 16 bits, canfd communication protocol byte is 8 bits, so when converting to modbus register, the high 8 bits will be filled with 0x00 by default, and the low 8 bits will use canfd protocol byte.





3) ModbusRTU mode (this mode is ModeBusRTU to canfd under our host computer software, and currently only supports the operation function codes 0x10 and 0x03).

ModbusRTU mode baud rate is fixed at 19200, the device address corresponds to motor canid, the start address of register is 0x0000, with 16 registers representing canfd 16byte data, read and write compatible with modbusRTU protocol. Note that modbus protocol registers are 16 bits and canfd protocol bytes are 8 bits, modbus registers are filled with 0x00 by default for high 8 bits and canfd protocol bytes for low 8 bits.



Send frame (function code bit 0x10, i.e. write data)

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 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	

The actual data sent to the motor is 16 bits taking only the low address of pos1~zero. Answer frame

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

Send frame (function code 0x03,

i.e. read data) 01 03 00 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

Answer frame:
01 03 20 00 ff 00 fb 00 27 00 f9 00 f5 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 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 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	

Similarly, the actual data is 16-bit only

taking the low address of pos1~zero. Crc32

calculation code:

1. look-up table method

```
static const unsigned int crc32tab[] = {
 0x00000000L, 0x77073096L, 0xee0e612cL, 0x990951baL.
 0x076dc419l, 0x706af48fl, 0xe963a535l, 0x9e6495a3l.
 0x0edb8832L, 0x79dcb8a4L, 0xe0d5e91eL, 0x97d2d988L,
 0x09b64c2bL, 0x7eb17cbdL, 0xe7b82d07L, 0x90bf1d91L,
 0x1db71064L, 0x6ab020f2L, 0x3b97148L, 0x84be41deL,
 0x1adad47dL, 0x6ddde4ebL, 0x4d4b551L, 0x83adad47dL,
 0x6ddde4ebL, 0x4d4b551L 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, 0x6bc5767L, 0x3fb506ddL, 0x6bc5767L
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,
0xf6b9265e1L, 0x6fb077e1L, 0x81be16cdL,

0x6fb077e1L 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        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;
}

direct method of calculation
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(! *i;
  if(!)
  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
Calculation
Code: 1. Direct
Algorithm:
unsigned int ModBusCRC16(unsigned char *data, unsigned int len)

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

    CRC16 = 0xFFFF;                 /CRCregister
    initial value 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.      // heteroscedastic polynomials
            }
        }
    }
    return CRC16.
}
```

2. Half-byte lookup table method:

```
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. Full byte lookup table method:

```
/* 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) ;)
}
```

X. Motor PWM Control Input/Output Parameter Table

PWM Control a meter (measuring sth)	Position Max	Position Minimum	Forward rotation speed	Reverse rotation speed degree (angles, temperature etc)	positive moment	reversing torque
PWM Input pulse width	2ms	1ms	1.55ms - 2.5ms 2ms	1ms - 1.45ms	1.55ms - 2.5ms 2ms	1ms - 1.45ms
HO4307	orientation 1 weekly	Negative direction one weekly	0.1Hz---585 Hz	-585Hz -0.1Hz	0A ---- 0.74A	-0.74A - 0A
HO5515	orientation 1 weekly	Negative direction one weekly	0.1Hz---585 Hz	-585Hz -0.1Hz	0A ---- 0.74A	-0.74A - 0A
HO6213	orientation 1 weekly	Negative direction one weekly	10Hz --- 90 Hz	-90Hz -10Hz	0A ---- 3.71 A	-3.71A---0A

HO7213 Label quasi-	orientation 1 weekly	Negative direction one weekly	10Hz --- 90 Hz	-90Hz -10Hz	0A ---- 3.71 A	-3.71A---0A
HO7213 High velocity	orientation 1 weekly	Negative direction one weekly	20Hz --- 495 Hz	-495Hz -20Hz	0A ---- 3.71 A	-3.71A---0A
HO8110	orientation 1 weekly	Negative direction one weekly	10Hz --- 450 Hz	-450Hz -10Hz	0A ---- 3.71 A	-3.71A---0A
HO10010	orientation 1 weekly	Negative direction one weekly	10Hz --- 450 Hz	-450Hz -10Hz	0A ---- 14.8 5A	-14.85A ----- 0A
HO10025	orientation 1 weekly	Negative direction one weekly	10Hz --- 450 Hz	-450Hz -10Hz	0A ---- 14.8 5A	-14.85A ----- 0A

XI, Motor Encoder Output Mode Parameters

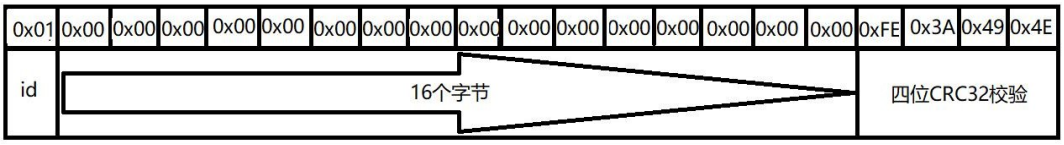
When the motor Encoder2 is used as an input port, it can be used to input a pulse signal, which is realized by configuring STEP_CONFIG in the motor parameters.
When STEP_CONFIG is set to 0, Encoder2 is the input port, and it forms a hollow joint module with the reducer end encoder, and the reducer end encoder has to use the encoder provided by our company.
When STEP_CONFIG is set to a number greater than 0 (integer), this data is n, and the motor will transmit 🐶1ms pulses to the encoder1 pin at n-degree intervals, and at the same time, each time when the motor reaches the zero point of a single revolution, it will transmit 🐶1ms pulses to the encoder2 pin.

Twelve, Hollow Encoder

Use the serial port debugging assistant.

发送帧为 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FE 3A 49 4E；发送数据

is a hexadecimal number.



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
of the byte frame.