

GIM User Manual

Driver Instructions



Read Guide

😊 Welcome to use the GIM Joint Actuator Database! To help you quickly obtain the information you need, please refer to the following reading guide:

1. Product Selection Manual

For detailed parameters of GIM products, please refer to [GIM CATLOG Lite.pdf](#).

2. Motor Model File

Please contact us.

3. Motor Usage

3.1 Motor Test

Thank you for purchasing GIM motors. Please refer to the GIM GUI Software User Manual to connect and test the motors .

3.2 Secondary Development

- **Memory Table Definition**

Before integrating motor control into your own project, please first understand the definition of the internal memory table of the motor. [\[MD Series\] Memory Table.xlsx](#) or [\[QD&DD Series\] Memory Table & ModBus Address.xlsx](#). Controlling the motor essentially involves modifying the register values of the motor's memory table. Modification of the memory table register values can be achieved by sending data via the 485 or CAN interface.

- **Communication Protocol**

GIM motors have two communication interfaces, 485 and CAN. Please follow the format of the 485 Communication Protocol according to the specific motor model. “[485 Communication Protocol V2.0](#)”, the CAN Communication Protocol “[CAN Communication Control Protocol V2.0](#)” to define the format for sending and parsing interactive data.

- **SDK**

We provide 485 SDK interfaces “[485/TTL Communicate SDK](#)”, helping you quickly implement motor control. Currently, for CAN communication, there is a test Demo with the MIT Communication Protocol “[MIT Protocol Test Demo](#)”.

Table of Contents

GIM GUI Software User Manual	1
485 Communication Protocol V2.0	19
CAN Communication Control Protocol V2.0	36
MIT Protocol Test Demo	53
485/TTL Communicate SDK	66

GIM GUI Software User Manual

1. Software Overview

This software is a debugging and parameter setting software for GIM series motors, striving to provide users with a user-friendly and visual operating interface. The software mainly implements functions such as online search for motors, motor control, parameter setting, real-time data monitoring and curve plotting, firmware upgrade, etc. The software supports both RS485 and CAN communication methods, and also supports online control of multiple motors.

2. Device Wiring

Note: Before powering the device, please confirm the power supply voltage range.

GIM series motors provide two communication methods, RS485 and CAN, and RS485 or CAN communication can be set via a DIP switch.



Figure 1 Motor Communication Port Settings

The motor needs to be connected to the computer using a USB to RS485 module or a USB to CAN module. The following two types of modules are recommended.



Figure 2 USB to RS485 Module



Figure 3 USB to CAN Module

The communication functions of 485 and CAN are basically the same. Below, the software functions will be introduced using 485 communication as an example. The wiring of the 485 module and multiple motors is shown in the figure below.



Figure 4 Motor Wiring Diagram

After the USB to 485 module is connected to the computer, check the corresponding COM port in Device Manager.

3. Software Startup and Interface Introduction

This software is installation-free. Double-click the GIM_GUI_xxx.exe file to start the control software. The software connects to the GIM motor via a USB cable. After the software starts, the interface is shown in the figure below.

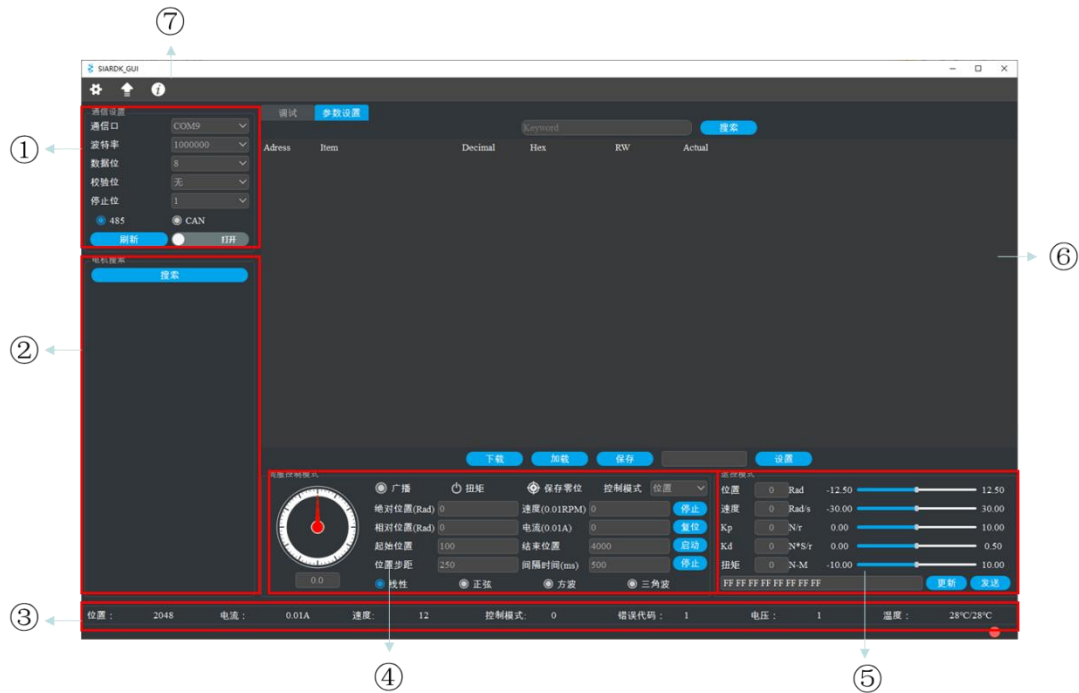


Figure 5 Main Interface Diagram

- ① Communication Settings Area
- ② Online Device List Area
- ③ Real-time Data Display Area
- ④ Servo Control Mode Area
- ⑤ Motion Mode (MIT) Area
- ⑥ Register Real-time Data Table and Real-time Curve Drawing Area
- ⑦ Menu Bar Area

4. Motor Commissioning Steps

4.1 Connect Device

First, switch the DIP switch of the motor communication interface to the 485(AB) side, use a USB to 485 module to connect the communication and power ports of the motor, and insert the 485 module into the computer's USB port at the same time. Open the GIM_GUI software on the computer, select the correct COM port, and choose the 485 mode to open serial communication.



Figure 6 Communication Settings

Successfully opened serial communication as follows



Figure 7 successfully opens serial communication

Power on the motors, click the "Search" button to start searching for all online devices and display them in the device list area. After clicking the "Search" button, the software attempts to connect to motors with IDs from 1 to 252. If the connected devices are already displayed in the device list area, you can click the "Stop" button to end the online search process. The first-level directory of the device list tree is the device model, and the second-level directory is the device ID. Motors of the same model are under the same first-level directory.

Note: The device ID number must be unique; otherwise, it will affect device communication.



Figure 8 Device Search Results

Left-click on the device ID row in the device list to select the currently controlled motor. At this time, the parameter setting area on the right side of the software and the status monitoring area at the bottom will display the real-time data of the selected motor, and the communication indicator light in the status bar at the bottom right will turn green to indicate normal communication. You can click on another device ID row again to switch the currently controlled motor.

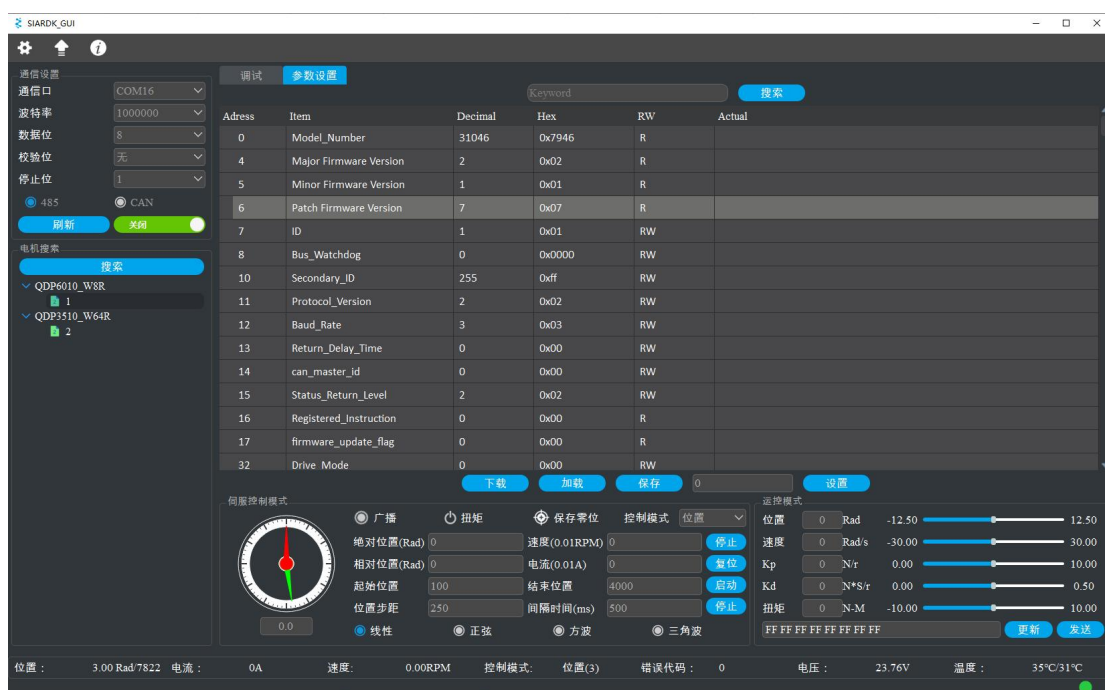


Figure 9 Select Control Device

4.2 Calibration Motor

Note: The motor has been calibrated before shipment, so this step can be skipped. If you need to calibrate the motor, please follow the steps below.

Calibration work mainly corrects the installation error of the sensor. During the process, the motor will rotate forward and backward. Please ensure that the motor can rotate freely, and it is best for the motor to run under no load; otherwise, calibration may fail.

Inside the GIM motor, registers are used to store motor operating parameters, and the motor operation can be controlled by modifying register values. The address of the motor calibration register is 403.

- Enter 403 in the "keyword" input box in the "Parameter Settings" area and press Enter. The parameter table only filters out row 403 of the motor calibration register, and the current value of the register is 48. The meaning of the value of register 403 will be introduced later;
- Click the mouse to select the 403 register row, and the background color of the selected row turns blue;
- Write a new value in the parameter setting input box, here write 51 (start internal and external encoder calibration).

4. Click the "Settings" button to start calibration.

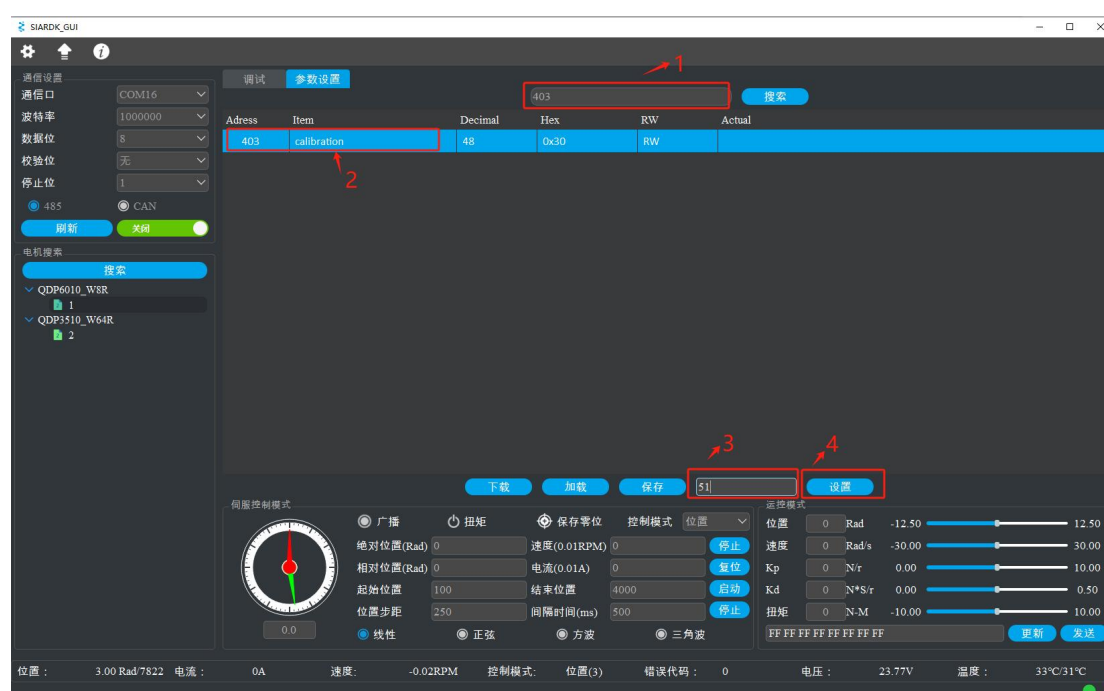


Figure 10 Setting Motor Calibration Register

At this time, the motor can be seen starting to rotate slowly, indicating that encoder calibration has begun. Please wait patiently. When the motor stops rotating and the value of register 403 changes to 60, it indicates that internal and external encoder calibration has been successful.

If the motor has dual encoders, the function of breakpoint position memory can be implemented. **The position range for breakpoint memory is (-PI, PI) .**

Calibration register 403, 1 ByteDance. Currently, 6 bit positions have been used. The meanings of each bit position are as follows:

Meaning of Table 1 Calibration Register (403)

Bit 7	Bit 6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
Retention	Retention	Is the external encoder connected (1: connected; 0: not connected)?	Is the internal encoder connected? (1: Connected; 0: Not connected)	Is the external encoder calibrated? (1: Calibrated; 0: Not Calibrated)	Is the internal encoder calibrated (1: calibrated; 0: uncalibrated)?	Whether to start external encoder calibration (1: start calibration; 0: no calibration)	Whether to start internal encoder calibration (1: start calibration; 0: no calibration)

After the motor is powered on, Bit4 and Bit5 for internal encoder connection status are automatically set based on the acquired encoder data; Bit3 and Bit2 are set by the calibration program results; Bit1 and Bit0 are set by the user to determine whether to start calibration, and will be restored to 0 after calibration is completed.

The meanings of common register 403 values are as follows. For other values, please analyze them according to the above definitions.

Table 2 Common Calibration Register 403 Values

48 (00110000)	Internal and external encoders are connected but not calibrated
60 (00111100)	Internal and external encoders are connected and calibrated
54 (00110110)	The internal and external encoders are connected, the internal encoder is calibrated, the external encoder is not calibrated, and the calibration of the external encoder is initiated
52 (00110100)	The internal and external encoders are connected, the internal encoder is calibrated, and the external encoder is uncalibrated

4.3 Servo Mode Debugging



Figure 11 Servo Control Debugging 1

The above area is the servo control mode area.

1. "Broadcast" button: When selected, it indicates that subsequent operations are effective for all online motors; otherwise, they are only effective for the currently selected motor;
2. "Torque" button: Select to turn on the motor torque switch; deselect to turn off the motor torque switch;
3. "Save Zero Position" button: Sets the current position as the zero position. This function takes effect only when the torque switch is off. After successful setting, the real-time position of the motor can be observed to display 0;
4. Control Modes: Currently supports four control modes: position, velocity, current, and MIT. Modes can be switched via the dropdown box.



Figure 12 Servo Control Debugging 2

1. Absolute Position: Enter the target position value and press Enter. The motor will run to the set position;
2. Relative Position: Enter the relative position value and press Enter. The motor will run to the entered relative position based on the current position;
3. Speed: Enter the target speed and press Enter. The motor will move at the set target speed;
4. "Stop" button: Sets the target speed to 0, and the motor stops moving;
5. Current: Enter the target current and press Enter. The motor will drive the motor to move with the torque corresponding to the set current;

6. "Reset" button: Sets the target current to 0, and the motor stops moving;



Figure 13 Servo Control Debugging 3

In the "Position" control mode, after turning on the torque switch, you can also control the motor movement by dragging the dashboard pointer with the mouse. The dashboard can control the motor to move within the range of $(0, 2\pi)$, with the zero position at the top and increasing clockwise.

The red pointer indicates the target position, and the green pointer indicates the current position. The value in the input box is synchronized with the red pointer on the dashboard, representing the target position of the motor. Therefore, you can also manually enter the target position value in the input box to control the motor to move to the specified position.



Figure 14 Servo Control Debugging 4



Figure 15 Servo Control Debugging 5

This area is the simulated curve control area. Target position commands can be continuously sent according to the set curve type and parameters. Currently, four types of simulated curves are supported: linear, sine, square wave, and triangular

wave. The "linear" curve requires setting 4 parameters, while the other three curves require setting 2 parameters.

1. "Linear"

Within the interval between the start position and the end position, the motor target position = start position + position step. The software sends the target position command at the set time interval.

2. "Sine"

Amplitude: amplitude of the sine wave; Period: period of the sine wave. The software calculates the sine wave values at a fixed sampling frequency of 50 Hz and sets them as the position target values for transmission. The target position transmission time interval is: $1/\text{Period}$.

The total number of samples in a cycle, $\text{sample} = \text{cycle} * 0.05$;

Current simulation sampling index sampleIndex ;

Position target value = $\text{amplitude} * \sin(2 * \pi * \text{sampleIndex} / \text{sample})$;

3. " Fang Bo "

Amplitude: Square wave amplitude; Period: Square wave period. The software takes the set amplitude as the target value and sends it, sending the position target value twice within one period (once for the positive amplitude and once for the negative amplitude). The sending time interval is: $\text{Period}/2$.

4. "Triangle Wave":

Amplitude: Square wave amplitude; Period: Square wave period.

4.4 MIT Motion Control Mode Debugging



Figure 16 Motion Control Mode Debugging

The motion control mode, i.e., the MIT mode, requires setting five target values: position, target speed, Kp, Kd, and torque. After connecting the motor, the maximum and minimum values of the five MIT parameters can be obtained from the motor register and updated on the interface.

The registers corresponding to MIT-related parameters are as follows, with a unit of 0.01. The range can be modified by changing the values of these registers.

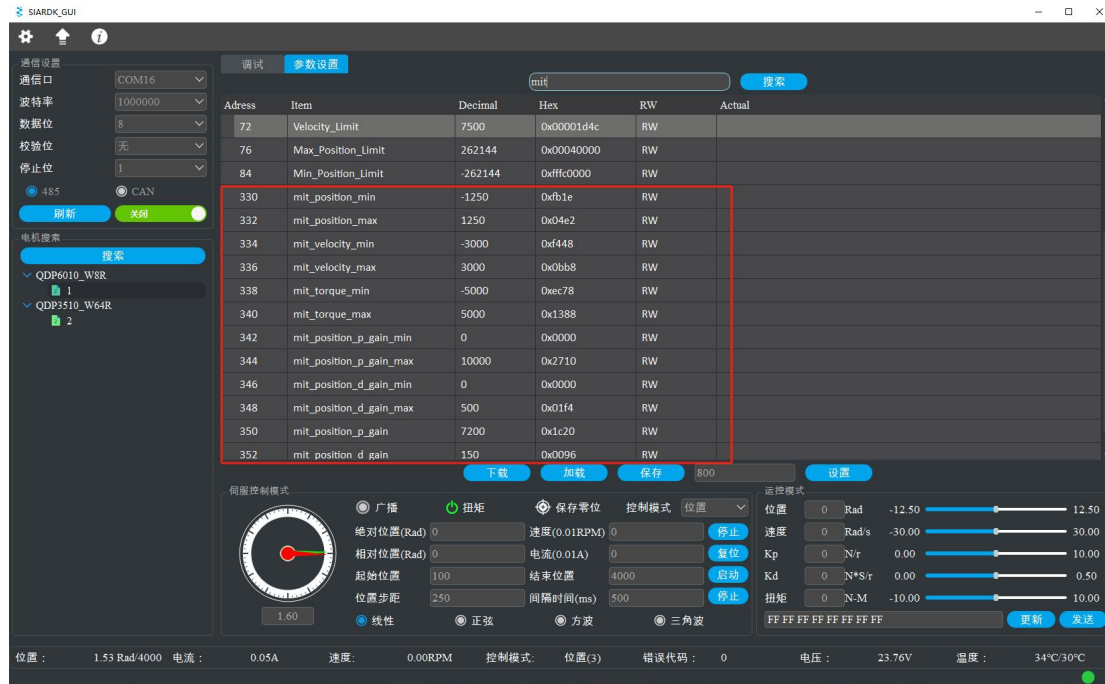


Figure 17 MIT Control-related Registers

First, the motor needs to be set to MIT mode, and then the torque switch should be turned on.

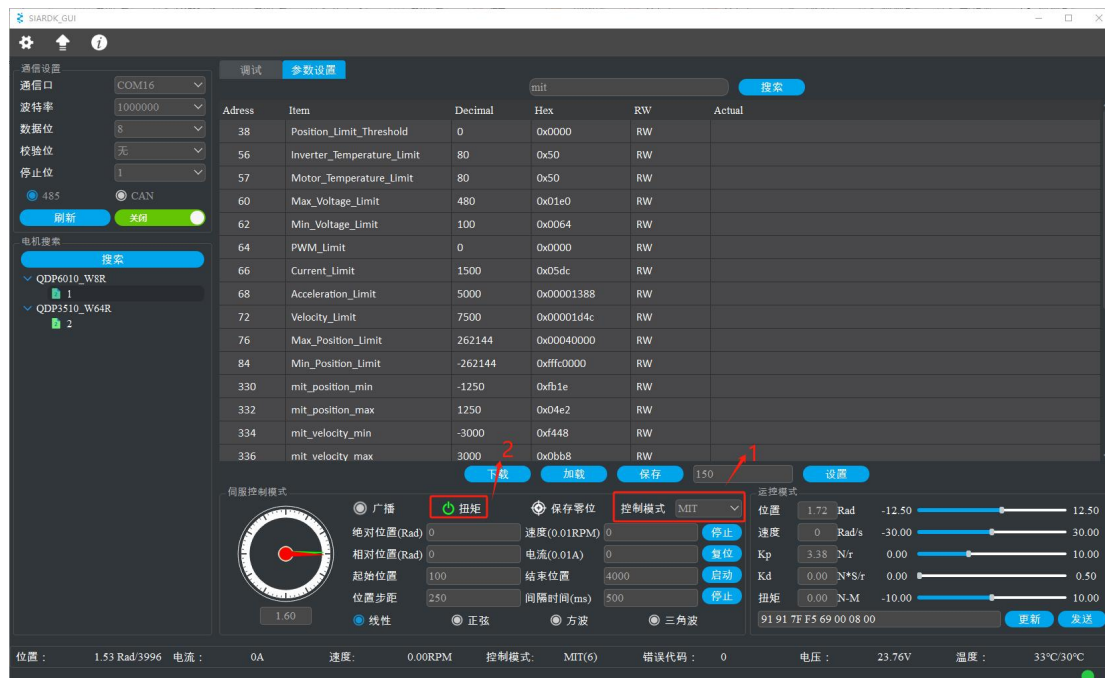


Figure 18 Set MIT Mode and Turn On Torque Switch

Drag the slider or modify the MIT target parameter value in the input box. After setting the target value, click the "Send" button to update the content of the hexadecimal display box and send the MIT command; click the "Update" button to only update the content of the hexadecimal display box without sending the

command.

The following details the three control methods under the MIT mode: speed, position, and torque. Why can the MIT mode implement these three control methods? This can be analyzed from the control block diagram of the MIT mode shown below.

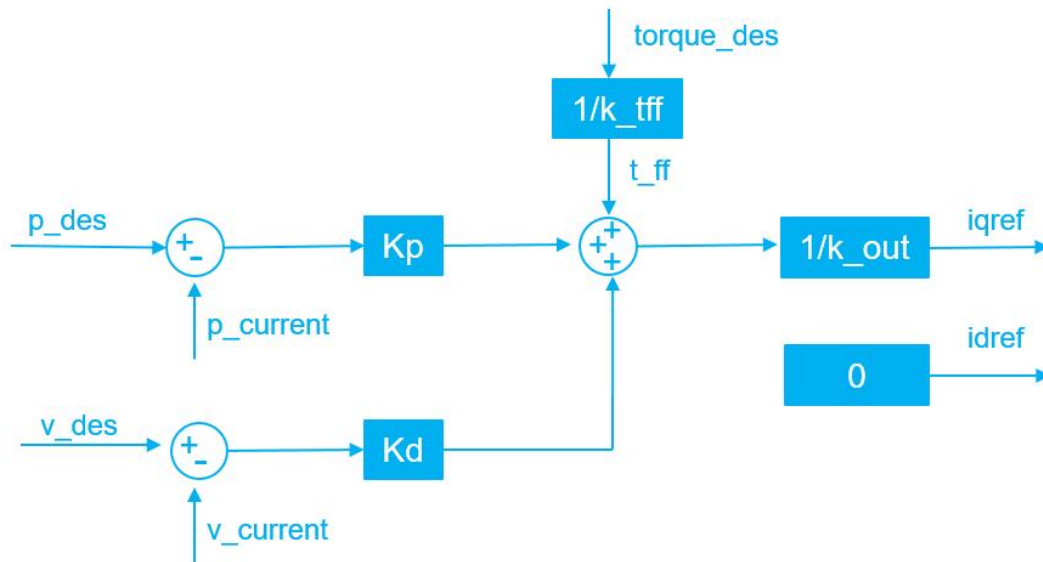


Figure 19 MIT Control Block Diagram

From the MIT operation block diagram, it can be seen that the calculation formula for the q-axis current i_{qref} of the motor is as follows:

$$i_{qref} = ((p_des - p_current) * Kp + (v_des - v_current) * Kd + (torque_des) * (1/k_tff)) * (1/k_out);$$

Among them, k_tff is the torque coefficient, and k_out is the current conversion coefficient, usually the motor constant, which does not need to be modified.

1. Position Mode

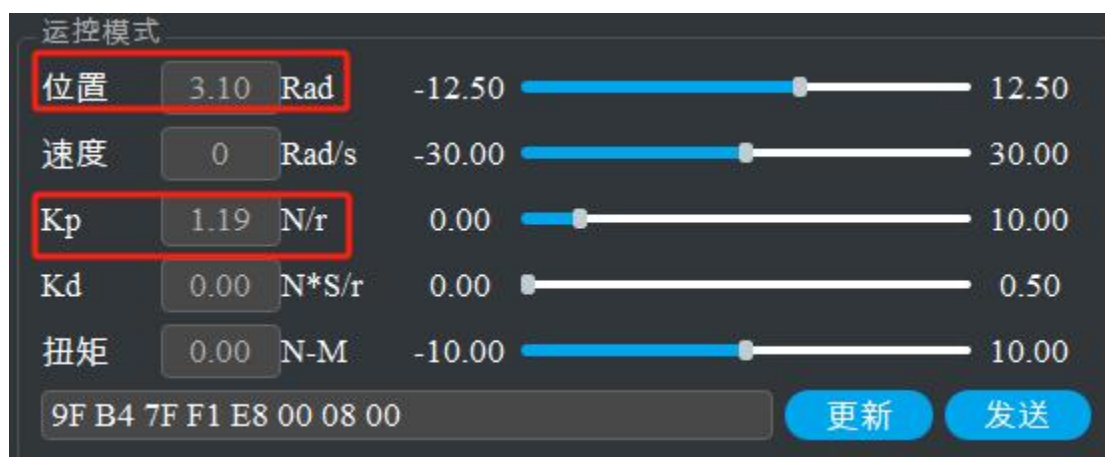


Figure 20 MIT - Position Mode

When the target position and Kp are given and other values are 0, i_{qref} can be simplified to the following formula. This is a position control with only a proportional loop, so there may be a certain steady-state error.

$$iqref = (p_des - p_current) * Kp ;$$

Note: At this time, attention should be paid to the initial position of the motor to avoid a large difference between the target position and the initial position, which may cause motor shock. You can set the current position of the motor as the zero point by clicking "Save Zero Point" to facilitate the setting of the "Position" parameter.

2. Speed Mode



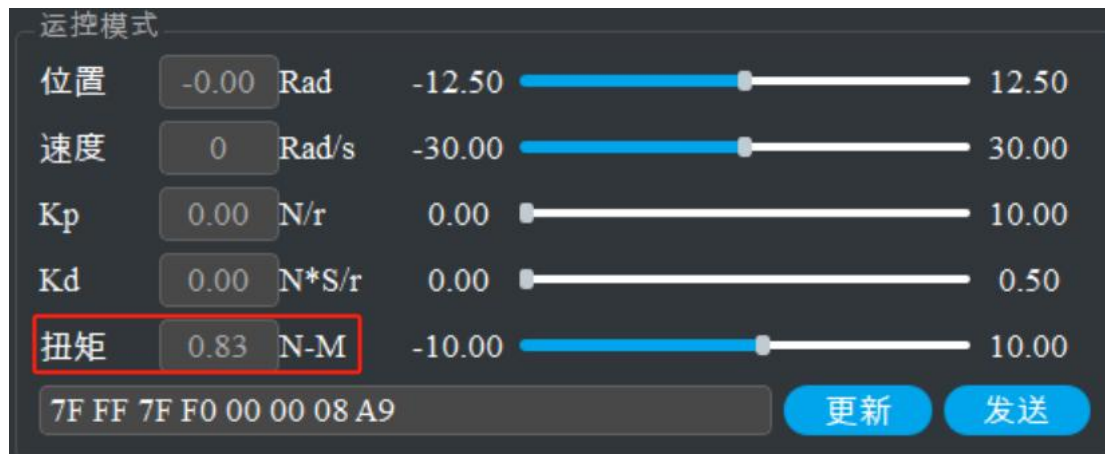
The image shows the MIT-Speed Mode control interface. It features a dark background with white text and blue sliders. The title '运控模式' (Control Mode) is at the top left. Below it, there are five rows of controls: '位置' (Position) with a value of 0 Rad, '速度' (Speed) with a value of 6.34 Rad/s, 'Kp' with a value of 0.00 N/r, 'Kd' with a value of 0.11 N*S/r, and '扭矩' (Torque) with a value of 0.00 N-M. Each row has a numerical input field, a unit, a range, and a slider. The '速度' and 'Kd' rows are highlighted with red boxes. At the bottom, there is a hexadecimal address field showing '7F FF 9B 00 00 22 18 00' and two buttons: '更新' (Update) and '发送' (Send).

Figure 21 MIT-Speed Mode

When the given target speed and Kd are provided, and other values are 0, Iqref can be simplified to the following formula. This is a speed control with only a proportional loop, so there may be a certain steady-state error.

$$iqref = (v_des - v_current) * Kd ;$$

3. Torque Mode



The image shows the MIT-Torque Mode control interface. It features a dark background with white text and blue sliders. The title '运控模式' (Control Mode) is at the top left. Below it, there are five rows of controls: '位置' (Position) with a value of -0.00 Rad, '速度' (Speed) with a value of 0 Rad/s, 'Kp' with a value of 0.00 N/r, 'Kd' with a value of 0.00 N*S/r, and '扭矩' (Torque) with a value of 0.83 N-M. Each row has a numerical input field, a unit, a range, and a slider. The '扭矩' row is highlighted with a red box. At the bottom, there is a hexadecimal address field showing '7F FF 7F F0 00 00 08 A9' and two buttons: '更新' (Update) and '发送' (Send).

Figure 22 MIT-Torque Mode

When the given target torque is set and other values are 0.

$$iqref = (torque_des) * (1/k_tff);$$

Note: Under no-load conditions, even when a very small torque is applied, the motor will accelerate to its maximum speed and rotate.

5. Other Functions

5.1 Set to Zero



Figure 23 Save Zero Position

Click the "Save Zero Position" button to set the current position as the zero position. After successful setting, you can observe that the position information in the bottom status bar changes to 0.

5.2 View real-time curve

Click the "Debug" tab to view real-time curves. Currently, only real-time position, speed, and current curve graphs are supported for viewing. Click the switch in the upper right corner to turn on the data you need to view, then click the ► button to start curve drawing. The display range of the vertical axis of the curve can be set by entering the maximum value in the input box on the right. If you need to view positive and negative values, you can click the "Negative" switch.

The curve drawing area supports zooming in and out using the mouse wheel; you can also select an area with the mouse to zoom in. The right-click menu option "Reset plot" can restore the original display scale of the curve. It is recommended to click the button to pause the curve playback before performing a selection to zoom in for viewing; otherwise, the curve will continuously refresh, making it impossible to view the data.

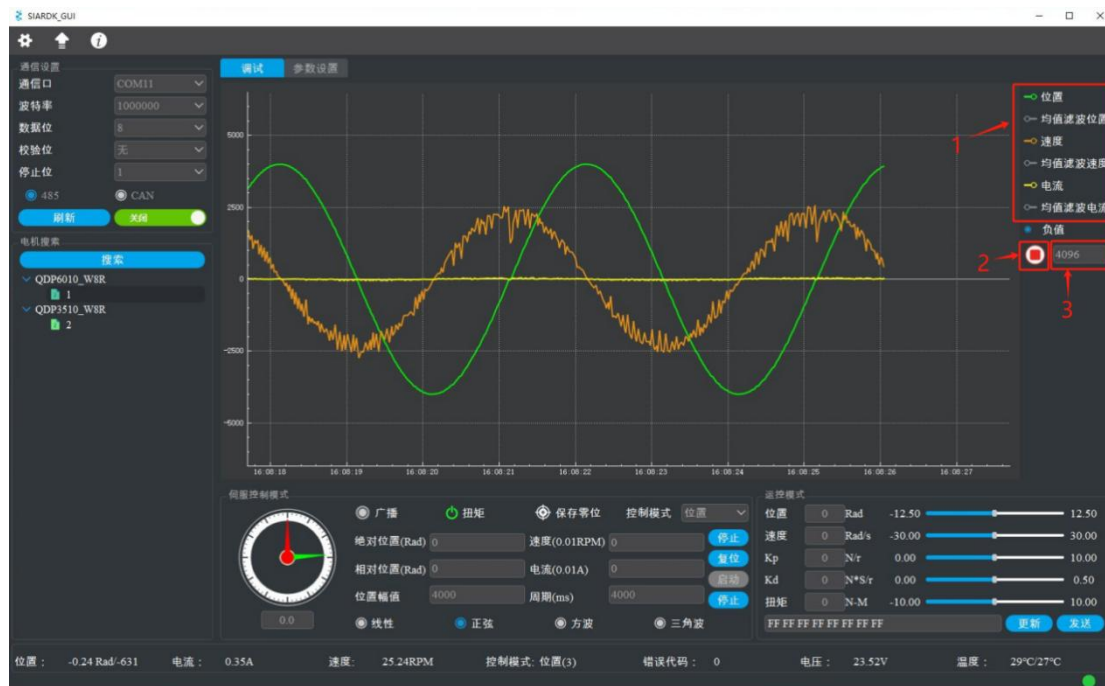


Figure 24 View Real-time Curve

5.3 Modify register value

1. Enter the search keyword in the "keyword" input field, press Enter, and the software will match the contents of the Address and Item columns in the table, filtering out the table row data that matches the keyword;
2. Select the row corresponding to the register that needs to be modified. The background of the selected row is blue;
3. Set a new value in the register value input box;
4. Click the "Settings" button. Send the settings command.

During the use of the motor, users can adjust the motor parameters to optimize its performance according to the specific usage scenario. For details on the definition of motor registers, please refer to the document "GIM Memory Table Analysis". The PID parameters of the motor's three-loop control have a significant impact on the motor's operating effect. It is recommended to have a basic understanding of motion control principles before modifying these parameters, and adjust the parameters in combination with the real-time curve.

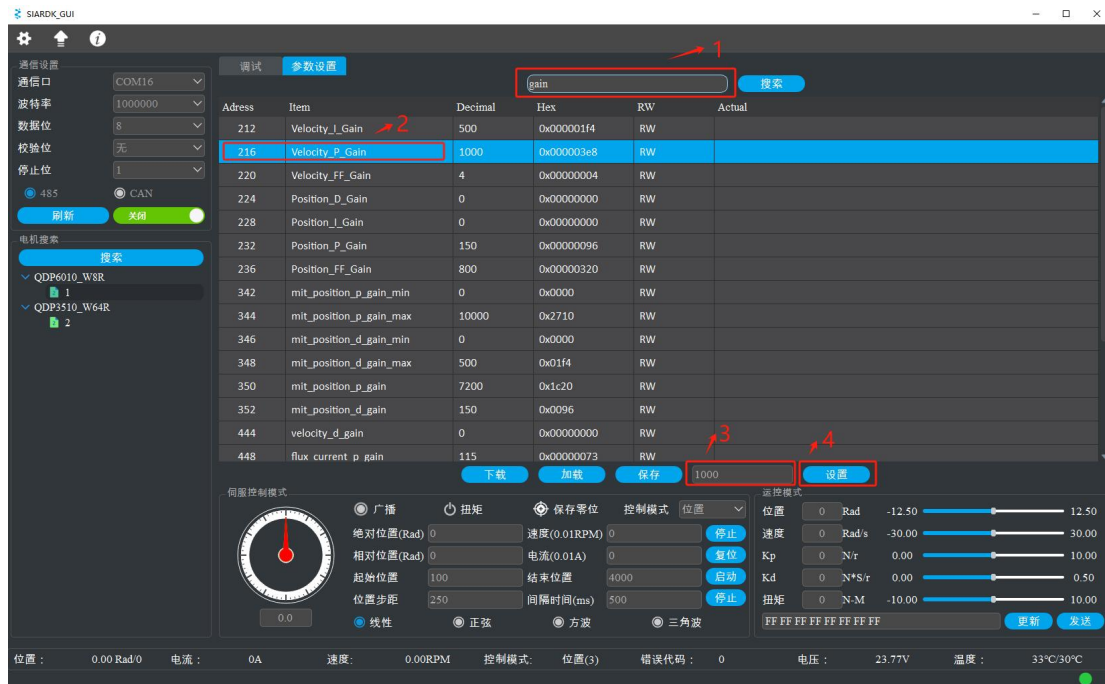


Figure 25 Modify Register

5.4 Register parameter saving, loading, and downloading

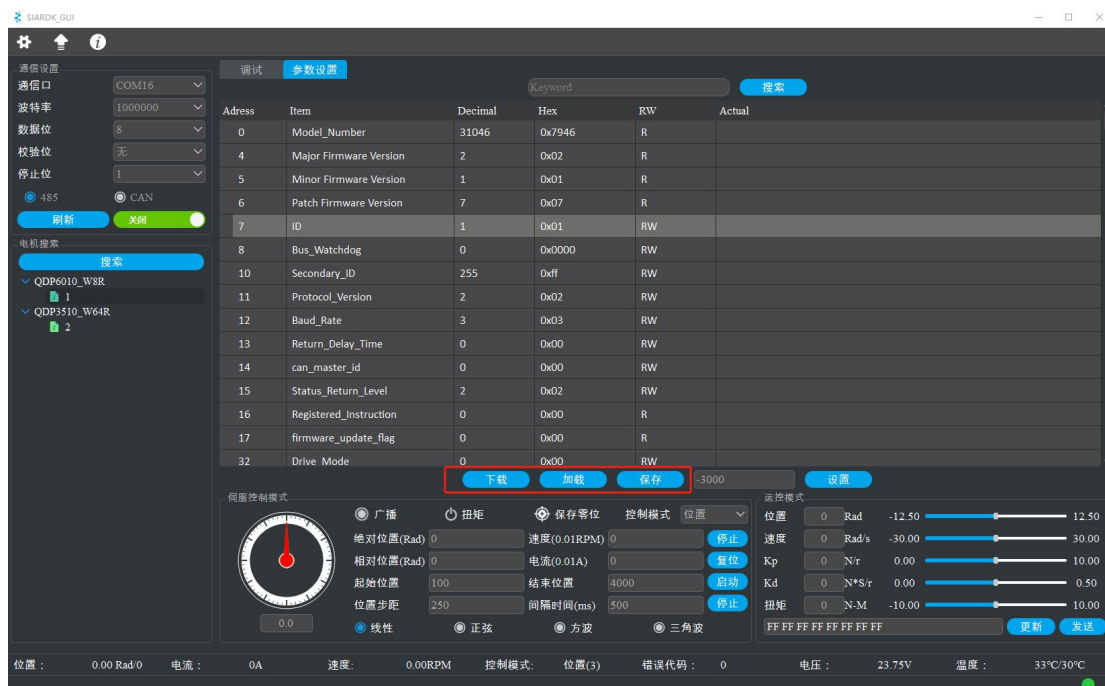


Figure 26 Register Parameter Saving, Loading, and Downloading

1. "Save" button: Saves the entire parameter table content as a local file with the file format.vdat;

2. "Load" button: can load the selected local parameter file (.vdat) into the parameter table area;
3. "Download" button: It can download the content of the entire parameter table to the motor. This operation is only supported in 485 communication mode and involves certain risks. Please operate with caution.

5.5 Firmware Upgrade

Click the firmware upgrade menu to pop up the firmware upgrade window. Select the firmware bin file, and finally click the "Upgrade" button to start the firmware upgrade.

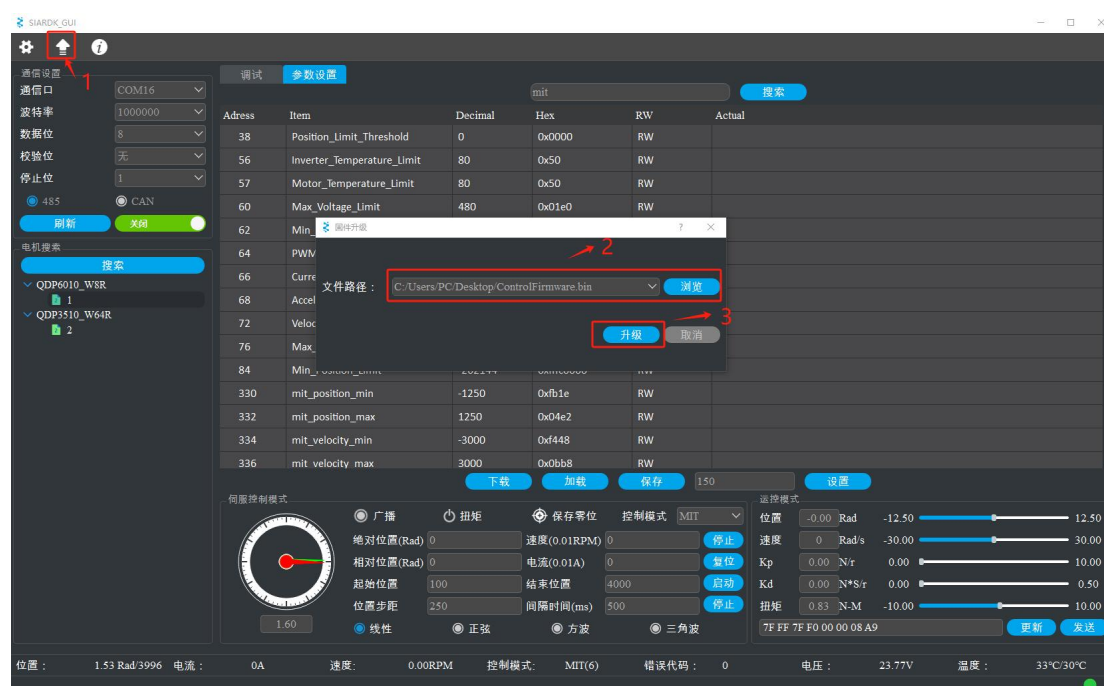


Figure 27 Initiate Firmware Upgrade

The firmware upgrade success prompt is shown in the figure below.

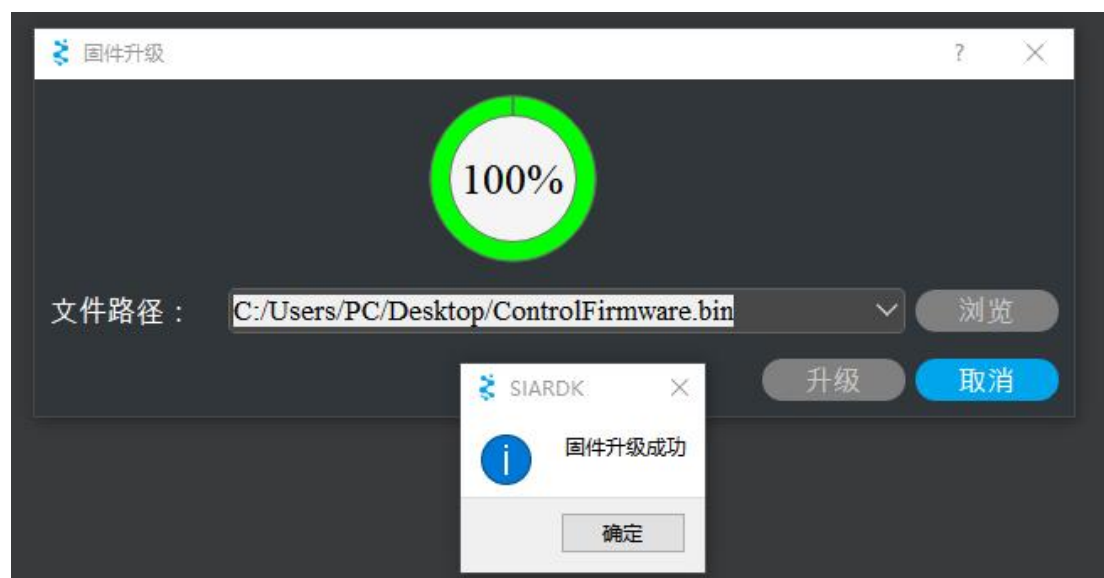


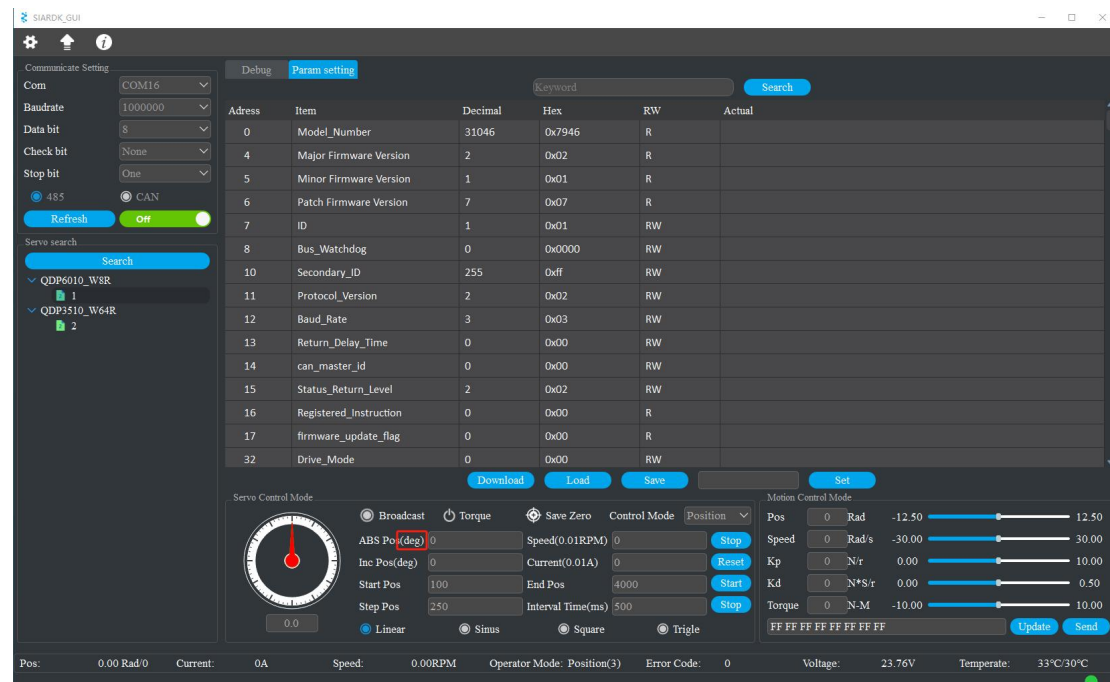
Figure 28 Firmware Upgrade Successful

5.6 Location Unit and Language Settings

Click the settings menu, pop up the settings window, set the position unit and software language, and finally click the "OK" button.



Figure 29 Unit and Language Settings



Results of Switching Units and Languages in Figure 30

485 Communication Protocol V2.0

The Communication Protocol requires specifying the starting address of the register. For the definition of register addresses, please refer to [Memory Table and Modbus Address Definition Description](#)

1 Introduction

This document defines the **GIM** series servo motor 485 communication control protocol format, with the protocol version being V2.0. Please strictly follow this protocol format specification to interact with GIM servo motors.

2 Data Frame Format

The communication of GIM series servo motors is asynchronous and in units of ByteDance. The communication data frame is an indeterminate serial data stream. The total length of the data frame is: (N + 10) Bytes.

2.1 Basic Format

2.1.1 Instruction Package

Message Headers H1-H3 (3 Bytes)			Reserved (RSRV)	Packet ID	Length (Low Byte)	Length (High Byte)	Instruction	Parameter	Parameter	Parameter	CR C 1	CR C 2
0x F F	0x F F	0 x F D	0x00	ID	Len_L	Len_H	Instruction	Parameter 1	...	Parameter N	CR C_L	CR C_H

The meanings of each ByteDance are as follows:

- **Message Header:** 3 Bytes, with fixed values of 0xFF, 0xFF, 0xFD;
- **Reserved:** Use 0X00 (Note that Reserved does not use 0XFD);
- **Packet ID :** This field indicates the ID of the device that should receive and process the instruction packet.

5. **ID Range:** 0 ~ 252 (0x00 ~ 0xFC), with a total of 253 numbers available for use
2. **Broadcast ID:** 254 (0xFE), all connected devices execute the instruction data packet
3. **Data Length:** A field indicating the length of the data packet, which is divided into a low byte and a high byte in the instruction packet. The data packet length is 2 ByteDances, equal to the parameter N to be sent plus 3, i.e., "N+3".
4. **Instruction:** Packet operation function code, see 2.2 Instruction Types for details;
5. **Parameter:** As an auxiliary data field for instructions, each instruction has a different purpose;
2. **CRC:** 16-bit CRC field, used to check whether the data packet is corrupted during the communication process. In the command packet, it is divided into low ByteDance and high ByteDance. CRC calculation range: all parameters from the message header (FF FF FD) to the position before the CRC field in the command packet.

2.1.2 Response Packet

Message Headers H1-H3 (3 Bytes)			Res erv ed (RS RV)	Pa ck et ID	Len gth (Lo w Byte)	Len gth (Hig h Byte)	Instr ucti on	E R R	P ar a m et er	P ar a m et er	Pa ra m et er	CRC 1	CRC 2
0 x F F F	0 x F F F	0 x F D	0x0 0	ID	Len _L	Len_ H	Instr uctio n	Er ro r	P ar a m et er 1	...	Pa ra m et er N	CRC _L	CRC _H

The meanings of each ByteDance are as follows:

3. **Message Header:** 3 ByteDance, with fixed values of 0xFF, 0xFF, 0xFD;
4. **Reserved:** Use 0X00 (note that Reserved does not use 0XFD);
5. **Packet ID:** This field indicates the device ID of the device returning the response packet.
 6. **ID Range:** 0 ~ 252 (0x00 ~ 0xFC), with a total of 253 numbers available for use

7. **Broadcast ID:** 254 (0xFE), meaning all connected devices receive the command data packet;
2. **Data Length:** A field that indicates the length of the data packet, which is divided into a low byte and a high byte in the instruction packet. The data length is 2 ByteDance, equal to the parameter N to be sent plus 3, i.e., "N+3".
3. **Instruction:** Packet operation function code, see the following instruction types for details;
4. **Parameter:** As an auxiliary data field for Instruction, each Instruction has a different purpose;
5. **CRC:** 16-bit CRC field, used to check whether the data packet is corrupted during the communication process. In the command packet, it is divided into low ByteDance and high ByteDance. CRC calculation range: from the message header (FF FF FD) to the parameters before the CRC field in the command packet.

2.2 Instruction Type

Different instructions represent different operations, and the instructions supported by the GIM series motors are shown in the following table:

Instruction	Function	Value	Parameter Length
PING (Query)	Query device status	0x01	0
READ (Read)	Read data from the device control table	0x02	4
WRITE (Write)	Write data to the device control table	0x03	Greater than or equal to 2
FACTORY RESET (Restore Factory Settings)	Reset the device control table to its initial factory default settings	0x06	Greater than or equal to 3
REBOOT (Restart)	Restart the device	0x08	0
SYNC READ (Synchronous Read)	For multiple devices, read data of the same length from the same address at once	0x82	Not less than 4
SYNC WRITE	For multiple devices,	0x83	Greater than 4

(Synchronous Write)	write data of the same length from the same address at once		or greater than 5
FAST SYNC READ (Fast Synchronous Read)	Similar to the SYNC READ instruction in function, but enables faster communication	0x8A	Greater than 4
BULK READ (Batch Read)	For multiple devices, read data of different lengths from different addresses at once	0x92	Greater than 5
BULK WRITE (Batch Write)	For multiple devices, write data of different lengths from different addresses at once	0x93	Not less than 5
FAST BULK READ (Fast Bulk Read)	Similar to the Bulk Read instruction in function, but enables faster communication	0x9A	Not less than 5

2.2.1 Query Status PING

Function Query device status

Length 0X03

Instruction 0X01

Parameter None

When the PING command uses the broadcast address 0XFE, all online devices return response information.

Example: Query the status of device ID1

Command Frame: **FF FF FD 00 01 03 00 01 19 4E**(Sent in Hexadecimal)

H1	H2	H3	RSRV	ID	LEN1	LEN2	INST	CRC 1	CRC 2
0xFF	0xFF	0xFD	0x00	0x01	0x03	0x00	0x01	0x19	0x4E

Returned data frame: **FF FF FD 00 01 07 00 55 00 06 04 26 65 5D** (hexadecimal display)

H1	H2	H3	RS RV	ID	LE N1	LE N2	INS T	ER R	P1	P2	P3	CR C 1	CR C 2
0xF F	0xF F	0x F D	0x0 0	0x 01	0x0 7	0x0 0	0x5 5	0x0 0	0x0 6	0x 04	0x 26	0x6 5	0x5 D

2.2.2 Read Instruction READ

Function Read data from the device control table

Length 0X07

Instruction 0X02

Parameter 1 Low-order ByteDance of the starting address

Parameter 2 High-order Byte of the starting address

Parameter 3 Low-order ByteDance of Data Length (X)

Parameter 4 High-order ByteDance in Data Length (X)

Example: Read the current position of Motor ID1 (132, 0x0084, 4[ByteDance]), and the current position value is 166 (0x000000A6)

Command Frame:FF FF FD 00 01 07 00 02 84 00 04 00 1D 15(Sent in Hexadecimal)

H1	H2	H3	RS RV	ID	LE N1	LE N2	INS T	P1	P2	P3	P4	CR C 1	CR C 2
0x FF	0x FF	0x FD	0x0 0	0x 01	0x0 7	0x0 0	0x0 2	0x8 4	0x0 0	0x0 4	0x0 0	0x 1D	0x1 5

Returned Data Frame:FF FF FD 00 01 08 00 55 00 A6 00 00 00 8C C0
(Hexadecimal Display)

H1	H2	H3	RS RV	ID	LE N1	LE N2	INS T	ER R	P1	P2	P3	P4	CR C 1	CR C 2
0x FF	0x FF	0x F D	0x 00	0x 0 1	0x 08	0x 00	0x 55	0x 00	0x A6	0x 00	0x 00	0x 00	0x 8C	0x C0

2.2.3 Write Instruction WRITE

Function Write data to the device control table

Length 0X09

Instruction 0X03

Parameter 1 Low-order ByteDance of the starting address

Parameter 2 High-order Byte of the Starting Address

Parameter 3 First ByteDance

Parameter N+1 The Nth data

Example: Set the target position of Motor ID1 (116, 0x0074, 4[ByteDance]) to 512 (0x00000200)

Command Frame: **FF FF FD 00 01 09 00 03 74 00 00 02 00 00 CA 89** (Sent in Hexadecimal)

H 1	H 2	H 3	RS RV	ID	LE N 1	LE N 2	IN S T	P1	P2	P3	P4	P5	P6	CR C 1	CR C 2
0x FF	0x FF	0x F D	0x 00	0x 0 1	0x 09	0x 00	0x 03	0x 74	0x 00	0x 00	0x 02	0x 00	0x 00	0x CA	0x8 9

Returned Data Frame: **FF FF FD 00 01 04 00 55 00 A1 0C** (Hexadecimal Display)

H1	H2	H3	RSR V	ID	LEN 1	LEN 2	INS T	ERR	CRC 1	CRC 2
0xF F	0xF F	0xF D	0x00	0x0 1	0x04	0x00	0x55	0x00	0xA1	0x0C

2.2.4 Factory Reset RESET

Function Reset the Control Table to its initial factory default settings

Length 0X04

Instruction 0X06

Parameter 1

0xFF: Full Reset

0x01: Reset all content except for ID

0x02: Reset everything except ID and baud rate

Example: ID1 uses parameter 0x01 to reset all parameters except ID

Command Frame: **FF FF FD 00 01 04 00 06 01 A1 E6** (Sent in Hexadecimal)

H1	H2	H3	RSR	ID	LEN	LEN	INS	P1	CRC	CRC
----	----	----	-----	----	-----	-----	-----	----	-----	-----

			V		1	2	T		1	2
0xF F	0xF F	0xF D	0x00	0x01	0x04	0x00	0x06	0x01	0xA1	0xE6

Returned data frame: FF FF FD 00 01 04 00 55 00 A1 0C(hexadecimal display)

H1	H2	H3	RSR V	ID	LEN 1	LEN 2	INS T	ER R	CRC 1	CRC 2
0xF F	0xF F	0xF D	0x00	0x01	0x04	0x0 0	0x5 5	0x0 0	0xA1	0x0C

2.2.5 Reboot Device REBOOT

Function Restart device

Length 0X03

Instruction 0X08

Parameter None

Example: ID1 restarts the device

Command Frame: FF FF FD 00 01 03 00 08 2F 4E (Sent in Hexadecimal)

H1	H2	H3	RSR V	ID	LEN1	LEN 2	INST	CRC 1	CRC 2
0xFF	0xFF	0xFD	0x00	0x01	0x03	0x00	0x08	0x2F	0x4E

Returned Data Frame: FF FF FD 00 01 04 00 55 00 A1 0C (Hexadecimal Display)

H1	H2	H3	RSR V	ID	LEN 1	LEN 2	INST	ERR	CRC 1	CRC 2
0xF F	0xF F	0xF D	0x00	0x01	0x04	0x00	0x55	0x00	0xA1	0x0C

2.2.6 SYNC READ

Function For multiple devices, read data of the same length from the same address at once

ID 0XFE

Length 0X09

Instruction 0X82

Parameter 1 Low-order ByteDance of the starting address

Parameter 2 High-order Byte of the starting address

Parameter 3 Low-order ByteDance of Data Length (X)

Parameter 4 High Byte of Data Length (X)

...

Parameter N+2 ID number of the Nth servo

...

A SYNC READ command can query the control table contents of multiple motors at once. The SYNC READ command specifies the IDs of the motors to be queried, and the motors return response packets in the order of the IDs in the command packet. When using the SYNC READ command, the length of all queried data and the starting address of the data must be the same.

Example: Simultaneously read the current positions of ID1 - ID2 (132, 0x0084, 4[byte])

ID1's current position is 166 (0x000000A6)

ID2's current position is 2079 (0x0000081F)

Synchronous Read Instruction Packet: **FF FF FD 00 FE 09 00 82 84 00 04 00 01 02 CE FA** (Sent in Hexadecimal)

H 1	H 2	H 3	RS RV	ID	LE N 1	LE N 2	IN S T	P 1	P 2	P 3	P 4	P 5	P 6	CR C 1	CR C 2
0x FF	0x FF	0x FD	0x 00	0x FE	0x 09	0x 00	0x 82	0x 84	0x 00	0x 04	0x 00	0x 01	0x 02	0x CE	0x FA

Return Packet:

FF FF FD 00 01 08 00 55 00 A6 00 00 00 8C C0

FF FF FD 00 02 08 00 55 00 1F 08 00 00 8A BE

Return packet disassembly: ID1 return packet:**FF FF FD 00 01 08 00 55 00 A6 00 00 00 8C C0** (sent in hexadecimal)

H1	H2	H3	RS RV	ID	LE N1	LE N2	IN S T	E R R	P1	P2	P3	P4	CR C 1	CR C 2
0x	0x	0x	0x0	0x	0x	0x	0x	0x	0x	0x	0x	0x	0x8	0x

FF	FF	FD	0	01	08	00	55	00	A6	00	00	00	C	C0
----	----	----	---	----	----	----	----	----	----	----	----	----	---	----

ID2 Return Packet: FF FF FD 00 02 08 00 55 00 1F 08 00 00 8A BE (Sent in Hexadecimal)

H1	H2	H3	RS RV	ID	LE N1	LE N2	IN ST	E R R	P1	P2	P3	P4	CR C 1	CR C 2
0x FF	0x FF	0x FD	0x0 0	0x 02	0x 08	0x 00	0x 55	0x 00	0x 1F	0x 08	0x 00	0x 00	0x BA	0x BE

2.2.7 SYNC WRITE

Function For multiple devices, write data of the same length from the same address at once

ID 0XFE

Length 0X17

Instruction 0X83

Parameter 1 Low-order ByteDance of the starting address

Parameter 2 High-order Byte of the starting address

Parameter 3 Low-order ByteDance of Data Length (X)

Parameter 4 High Byte of Data Length (X)

Parameter 5[Device 1] ID

Parameter 5+1[Device 1] 1st ByteDance

...

Parameter 5+X [Device 1] Byte X

Parameter 6[Device 2] ID

Parameter 6+1[Device 2] Byte 1

...

Parameter 6+X [Device 2] Byte X

Example: Simultaneously set the target position of ID1 (116, 0x0074, 4[ByteDance]) to 150 (0x00000096)

Target position setting for ID2 (116, 0x0074, 4[ByteDance]) is 170 (0x000000AA)

Command Frame: FF FF FD 00 FE 11 00 83 74 00 04 00 01 96 00 00 00 02 AA 00 00 00 82 87(Sent in Hexadecimal)

H	H	H	R	I	L	L	I	P	P	P	P	P	P	P	P	P	P	P	P	C	C
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

1	2	3	S R V	D	E N 1	E N 2	N S T	1	2	3	4	5	6	7	8	9	1 0	1 1	1 2	1 3	1 4	R C 1	R C 2
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x
F	F	F	0	F	1	0	8	7	0	0	0	0	9	0	0	0	0	A	0	0	0	8	8
F	F	D	0	E	1	0	3	4	0	4	0	1	6	0	0	0	2	A	0	0	0	2	7

Since commands are sent using broadcast IDs, no data will be returned.

2.2.8 Fast Sync Read

This instruction enables faster communication than synchronous reading. Regardless of the number of series-connected servo motors, when using the fast synchronous reading instruction, all servo motors will only generate and return one status data packet, and its effect is equivalent to a single servo motor returning one status data packet. The data packet format of this instruction is exactly the same as that of the synchronous reading instruction.

Function Similar to the SYNC READ instruction function, but enables faster communication

Length 0X10

Instruction 0x8A

Parameter 1 Low-order ByteDance of the starting address

Parameter 2 High-order Byte of the starting address

Parameter 3 Low-order ByteDance of Data Length (X)

Parameter 4 High Byte of Data Length (X)

Parameter 4+1 1st Device ID

...

Parameter 4+N The Nth device ID

Explanation of Return Packet Parameters

Parameter 1[Device 1] ID

Parameter 2[Device 1] First Returned ByteDance Data

Parameter 3[Device 1] The 2nd returned ByteDance data

... ..

Parameter x+1[Device 1] The Xth returned ByteDance data

Parameter X+2 [Device 1] CRC Low ByteDance

Parameter X+3 [Device 1] CRC High ByteDance

Parameter X+4[Device 2] Error Code

Parameter X+5[Device 2] ID

Parameter X+5+1[Device 2] 1st Return ByteDance Data

Parameter X+5+2[Device 2] The 2nd returned ByteDance data

... ..

Parameter 2X+5[Device 2] The Xth returned ByteDance data

Parameter 2X+5+1 [Device 2] CRC Low ByteDance

Parameter 2X+5+ 2 [Device 2] CRC High ByteDance

... ..

Parameter nX+5[The Xth ByteDance return data of the nth device]

Note:

2. The CRC value of the intermediate device is used for internal motor operations to confirm the integrity of data packets between motor devices. During the data parsing process, it is only necessary to verify the CRC value at the end of the status data packet.
3. Each device will separately return a specific part of the status data packet to respond to the fast synchronization read command.

Example: Simultaneously read the current positions of ID3, ID4, and ID7 (132, 0x0084, 4[ByteDance])

ID3 current position is 166 (0x000000A6)

ID4's current position is 1023 (0x000003FF)

The current position of ID7 is 2,079 (0x0000081F).

Synchronous Read Command Frame: **FF FF FD 00 FE 0A 00 8A 84 00 04 00 03 07 04 20 F2** (Sent in Hexadecimal)

H 1	H 2	H 3	RS RV	ID	L E N 1	L E N 2	IN S T	P 1	P 2	P 3	P 4	P 5	P 6	P 7	C R C1	C R C2
0x F F F	0x F F D	0x F F D	0x0 0	0x F E	0x 0 A	0x 0 0	0x 8 A	0x 8 4	0x 0 0	0x 0 4	0x 0 0	0x 0 3	0x 0 7	0x 0 4	0x 20	0x F2

Return Frame:

FF FF FD 00 FE 19 00 55 00 03 A6 00 00 00 84 08

00 07 1F 08 00 00 16 CA

00 04 FF 03 00 00 D1 9E

Return packet disassembly: ID3 return packet: FF FF FD 00 FE 19 00 55 00 03 A6 00 00 00 84 08 (sent in hexadecimal)

H 1	H 2	H 3	RS RV	ID	L E N 1	L E N 2	IN S T	E R R	ID 1	D 1	D 2	D 3	D 4	CR C1	CR C2
0x F F F	0x F F F	0x F F D	0x0 0	0x F E	0x 19	0x 00	0x 55	0x 00	0x 03	0x A 6	0x 00	0x 00	0x 00	0x 84	0x 08

ID7 Return Packet:00 07 1F 08 00 00 16 CA (Sent in Hexadecimal)

ERR	ID2	D1	D2	D3	D4	CRC 1	CRC 2
0x00	0x07	0x1F	0x08	0x00	0x00	0x16	0xCA

ID4 Return Packet:00 04 FF 03 00 00 D1 9E (Sent in Hexadecimal)

ERR	ID3	D1	D2	D3	D4	CRC 1	CRC 2
0x00	0x04	0xFF	0x03	0x00	0x00	0xD1	0x9E

2.2.9 Bulk Read BULK READ

Function For multiple devices, read data of different lengths from different addresses at once

ID 0XFE

Length0X16

Instruction 0x92

Parameter 1[Device 1] ID

Parameter 2 [Device 1] Low-order ByteDance of the starting address

Parameter 3 [Device 1] High-order ByteDance of the starting address

Parameter 4 [Device 1] Low-order ByteDance of data length

Parameter 5 [Device 1] High-order ByteDance of data length

Parameter 6[Device 2] ID

Parameter 7 [Device 2] Low-order Byte of the starting address

Parameter 8 [Device 2] High-order Byte of the starting address

Parameter 9 [Device 2] Low-order ByteDance of data length

Parameter 10[Device 2] High-order ByteDance of data length

Example: Batch read the current voltage of ID1 (144, 0x0090, 2[ByteDance]), and the current voltage value is 119 (0x0077)

ID2 current temperature (146, 0x0092, 1[byte]), current position value is 36 (0x24)

Batch Write Command Frame:FF FF FD 00 FE 0D 00 92 01 90 00 02 00 02 92 00 01 00 1A 05 (Sent in Hexadecimal)

H1	H2	H3	RS	ID	LEN	LEN	INS	ERR	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	CRC1	CRC2
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0x	0x
x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	x	1	05
F	F	F	0	F	0	0	9	0	9	0	0	0	0	9	0	0	0	0	A	
F	F	D	0	E	D	0	2	1	0	0	2	0	2	2	0	1	0			

Return Frame:

FF FF FD 00 01 06 00 55 00 77 00 C3 69

FF FF FD 00 02 05 00 55 00 24 8B A9

Return packet disassembly: ID 1 status data packet:FF FF FD 00 01 06 00 55 00 77 00 C3 69 (sent in hexadecimal)

H1	H2	H3	RS	ID	LEN	LEN	INS	ERR	P1	P2	CRC1	CRC2
			RV		N1	N2	T	R			1	2
0x	0x	0x	0x0	0x0	0x0	0x0	0x5	0x0	0x7	0x0	0xC	0x69
FF	FF	FD	0	1	6	0	5	0	7	0	3	

Return packet disassembly: ID2 data packet:FF FF FD 00 02 05 00 55 00 24 8B A9 (sent in hexadecimal)

H1	H2	H3	RS	ID	LEN	LEN	INS	ERR	P1	CRC1	CRC2
			V		1	N2	T	R		1	2
0xF	0xF	0xF	0x00	0x02	0x05	0x0	0x5	0x0	0x2	0x8B	0xA9
F	F	D				0	5	0	4		

2.2.10 Bulk Write BULK WRITE

Function For multiple devices, write data of different lengths from different addresses at once

ID 0XFE

Length 0X13

Instruction 0x93

Parameter 1[Device 1] ID

Parameter 2 [Device 1] Low-order ByteDance of the starting address

Parameter 3 [Device 1] High-order ByteDance of the starting address

Parameter 4 [Device 1] Low-order ByteDance of Data Length (X)

Parameter 5 [Device 1] High-order ByteDance of Data Length (X)

Parameter 5+1[Device 1] 1st ByteDance

...

Parameter 5+X[Device 1] Byte X

Parameter 6+X[Device 2] ID

Parameter 7+X [Device 2] Low-order ByteDance of the starting address

Parameter 8+X [Device 2] High-order ByteDance of the starting address

Parameter 9+X[Device 2] Low-order ByteDance of Data Length (X)

Parameter 10+X [Device 2] High Byte of Data Length (X)

Parameter 10+X+1[Device 2] 1st ByteDance

Parameter 10+X+2[Device 2] Byte 2

...

Parameter 10+X+Y[Device 2] Byte Y

Example: Batch write ID1 maximum voltage (32, 0x0020, 2[ByteDance]) to 160 (0x00A0)

ID2 Temperature Limit (31, 0x001F, 1[ByteDance]) is 80 (0x50)

Batch Write Instruction Frame:FF FF FD 00 FE 10 00 93 01 20 00 02 00 A0 00 02 1F 00 01 00 50 B7 68 (Sent in Hexadecimal)

H 1	H 2	H 3	R S R V	I D	L E N 1	L E N 2	I N S T	P 1	P 2	P 3	P 4	P 5	P 6	P 7	P 8	P 9	P 10	P 11	P 12	P 13	P 14	P 15	P 16	P 17	P 18	P 19	P 20	P 21	P 22	P 23	P 24	P 25	P 26	P 27	P 28	P 29	P 30	P 31	P 32	P 33	P 34	P 35	P 36	P 37	P 38	P 39	P 40	P 41	P 42	P 43	P 44	P 45	P 46	P 47	P 48	P 49	P 50	P 51	P 52	P 53	P 54	P 55	P 56	P 57	P 58	P 59	P 60	P 61	P 62	P 63	P 64	P 65	P 66	P 67	P 68	P 69	P 70	P 71	P 72	P 73	P 74	P 75	P 76	P 77	P 78	P 79	P 80	P 81	P 82	P 83	P 84	P 85	P 86	P 87	P 88	P 89	P 90	P 91	P 92	P 93	P 94	P 95	P 96	P 97	P 98	P 99	P 100	P 101	P 102	P 103	P 104	P 105	P 106	P 107	P 108	P 109	P 110	P 111	P 112	P 113	P 114	P 115	P 116	P 117	P 118	P 119	P 120	P 121	P 122	P 123	P 124	P 125	P 126	P 127	P 128	P 129	P 130	P 131	P 132	P 133	P 134	P 135	P 136	P 137	P 138	P 139	P 140	P 141	P 142	P 143	P 144	P 145	P 146	P 147	P 148	P 149	P 150	P 151	P 152	P 153	P 154	P 155	P 156	P 157	P 158	P 159	P 160	P 161	P 162	P 163	P 164	P 165	P 166	P 167	P 168	P 169	P 170	P 171	P 172	P 173	P 174	P 175	P 176	P 177	P 178	P 179	P 180	P 181	P 182	P 183	P 184	P 185	P 186	P 187	P 188	P 189	P 190	P 191	P 192	P 193	P 194	P 195	P 196	P 197	P 198	P 199	P 200	P 201	P 202	P 203	P 204	P 205	P 206	P 207	P 208	P 209	P 210	P 211	P 212	P 213	P 214	P 215	P 216	P 217	P 218	P 219	P 220	P 221	P 222	P 223	P 224	P 225	P 226	P 227	P 228	P 229	P 230	P 231	P 232	P 233	P 234	P 235	P 236	P 237	P 238	P 239	P 240	P 241	P 242	P 243	P 244	P 245	P 246	P 247	P 248	P 249	P 250	P 251	P 252	P 253	P 254	P 255	P 256	P 257	P 258	P 259	P 260	P 261	P 262	P 263	P 264	P 265	P 266	P 267	P 268	P 269	P 270	P 271	P 272	P 273	P 274	P 275	P 276	P 277	P 278	P 279	P 280	P 281	P 282	P 283	P 284	P 285	P 286	P 287	P 288	P 289	P 290	P 291	P 292	P 293	P 294	P 295	P 296	P 297	P 298	P 299	P 300	P 301	P 302	P 303	P 304	P 305	P 306	P 307	P 308	P 309	P 310	P 311	P 312	P 313	P 314	P 315	P 316	P 317	P 318	P 319	P 320	P 321	P 322	P 323	P 324	P 325	P 326	P 327	P 328	P 329	P 330	P 331	P 332	P 333	P 334	P 335	P 336	P 337	P 338	P 339	P 340	P 341	P 342	P 343	P 344	P 345	P 346	P 347	P 348	P 349	P 350	P 351	P 352	P 353	P 354	P 355	P 356	P 357	P 358	P 359	P 360	P 361	P 362	P 363	P 364	P 365	P 366	P 367	P 368	P 369	P 370	P 371	P 372	P 373	P 374	P 375	P 376	P 377	P 378	P 379	P 380	P 381	P 382	P 383	P 384	P 385	P 386	P 387	P 388	P 389	P 390	P 391	P 392	P 393	P 394	P 395	P 396	P 397	P 398	P 399	P 400	P 401	P 402	P 403	P 404	P 405	P 406	P 407	P 408	P 409	P 410	P 411	P 412	P 413	P 414	P 415	P 416	P 417	P 418	P 419	P 420	P 421	P 422	P 423	P 424	P 425	P 426	P 427	P 428	P 429	P 430	P 431	P 432	P 433	P 434	P 435	P 436	P 437	P 438	P 439	P 440	P 441	P 442	P 443	P 444	P 445	P 446	P 447	P 448	P 449	P 450	P 451	P 452	P 453	P 454	P 455	P 456	P 457	P 458	P 459	P 460	P 461	P 462	P 463	P 464	P 465	P 466	P 467	P 468	P 469	P 470	P 471	P 472	P 473	P 474	P 475	P 476	P 477	P 478	P 479	P 480	P 481	P 482	P 483	P 484	P 485	P 486	P 487	P 488	P 489	P 490	P 491	P 492	P 493	P 494	P 495	P 496	P 497	P 498	P 499	P 500	P 501	P 502	P 503	P 504	P 505	P 506	P 507	P 508	P 509	P 510	P 511	P 512	P 513	P 514	P 515	P 516	P 517	P 518	P 519	P 520	P 521	P 522	P 523	P 524	P 525	P 526	P 527	P 528	P 529	P 530	P 531	P 532	P 533	P 534	P 535	P 536	P 537	P 538	P 539	P 540	P 541	P 542	P 543	P 544	P 545	P 546	P 547	P 548	P 549	P 550	P 551	P 552	P 553	P 554	P 555	P 556	P 557	P 558	P 559	P 560	P 561	P 562	P 563	P 564	P 565	P 566	P 567	P 568	P 569	P 570	P 571	P 572	P 573	P 574	P 575	P 576	P 577	P 578	P 579	P 580	P 581	P 582	P 583	P 584	P 585	P 586	P 587	P 588	P 589	P 590	P 591	P 592	P 593	P 594	P 595	P 596	P 597	P 598	P 599	P 600	P 601	P 602	P 603	P 604	P 605	P 606	P 607	P 608	P 609	P 610	P 611	P 612	P 613	P 614	P 615	P 616	P 617	P 618	P 619	P 620	P 621	P 622	P 623	P 624	P 625	P 626	P 627	P 628	P 629	P 630	P 631	P 632	P 633	P 634	P 635	P 636	P 637	P 638	P 639	P 640	P 641	P 642	P 643	P 644	P 645	P 646	P 647	P 648	P 649	P 650	P 651	P 652	P 653	P 654	P 655	P 656	P 657	P 658	P 659	P 660	P 661	P 662	P 663	P 664	P 665	P 666	P 667	P 668	P 669	P 670	P 671	P 672	P 673	P 674	P 675	P 676	P 677	P 678	P 679	P 680	P 681	P 682	P 683	P 684	P 685	P 686	P 687	P 688	P 689	P 690	P 691	P 692	P 693	P 694	P 695	P 696	P 697	P 698	P 699	P 700	P 701	P 702	P 703	P 704	P 705	P 706	P 707	P 708	P 709	P 710	P 711	P 712	P 713	P 714	P 715	P 716	P 717	P 718	P 719	P 720	P 721	P 722	P 723	P 724	P 725	P 726	P 727	P 728	P 729	P 730	P 731	P 732	P 733	P 734	P 735	P 736	P 737	P 738	P 739	P 740	P 741	P 742	P 743	P 744	P 745	P 746	P 747	P 748	P 749	P 750	P 751	P 752	P 753	P 754	P 755	P 756	P 757	P 758	P 759	P 760	P 761	P 762	P 763	P 764	P 765	P 766	P 767	P 768	P 769	P 770	P 771	P 772	P 773	P 774	P 775	P 776	P 777	P 778	P 779	P 780	P 781	P 782	P 783	P 784	P 785	P 786	P 787	P 788	P 789	P 790	P 791	P 792	P 793	P 794	P 795	P 796	P 797	P 798	P 799	P 800	P 801	P 802	P 803	P 804	P 805	P 806	P 807	P 808	P 809	P 810	P 811	P 812	P 813	P 814	P 815	P 816	P 817	P 818	P 819	P 820	P 821	P 822	P 823	P 824	P 825	P 826	P 827	P 828	P 829	P 830	P 831	P 832	P 833	P 834	P 835	P 836	P 837	P 838	P 839	P 840	P 841	P 842	P 843	P 844	P 845	P 846	P 847	P 848	P 849	P 850	P 851	P 852	P 853	P 854	P 855	P 856	P 857	P 858	P 859	P 860	P 861	P 862	P 863	P 864	P 865	P 866	P 867	P 868	P 869	P 870	P 871	P 872	P 873	P 874	P 875	P 876	P 877	P 878	P 879	P 880	P 881	P 882	P 883	P 884	P 885	P 886	P 887	P 888	P 889	P 890	P 891	P 892	P 893	P 894	P 895	P 896	P 897	P 898	P 899	P 900	P 901	P 902	P 903	P 904	P 905	P 906	P 907	P 908	P 909	P 910	P 911	P 912	P 913	P 914	P 915	P 916	P 917	P 918	P 919	P 920	P 921	P 922	P 923	P 924	P 925	P 926	P 927	P 928	P 929	P 930	P 931	P 932	P 933	P 934	P 935	P 936	P 937	P 938	P 939	P 940	P 941	P 942	P 943	P 944	P 945	P 946	P 947	P 948	P 949	P 950	P 951	P 952	P 953	P 954	P 955	P 956	P 957	P 958	P 959	P 960	P 961	P 962	P 963	P 964	P 965	P 966	P 967	P 968	P 969	P 970	P 971	P 972	P 973	P 974	P 975	P 976	P 977	P 978	P 979	P 980	P 981	P 982	P 983	P 984	P 985	P 986	P 987	P 988	P 989	P 990	P 991	P 992	P 993	P 994	P 995	P 996	P 997	P 998	P 999	P 1000	P 1001	P 1002	P 1003	P 1004	P 1005	P 1006	P 1007	P 1008	P 1009	P 1010	P 1011	P 1012	P 1013	P 1014	P 1015	P 1016	P 1017	P 1018	P 1019	P 1020	P 1021	P 1022	P 1023	P 1024	P 1025	P 1026	P 1027	P 1028	P 1029	P 1030	P 1031	P 1032	P 1033	P 1034	P 1035	P 1036	P 1037	P 1038	P 1039	P 1040	P 1041	P 1042	P 1043	P 1044	P 1045	P 1046	P 1047	P 1048	P 1049	P 1050	P 1051	P 1052	P 1053	P 1054	P 1055	P 1056	P 1057	P 1058	P 1059	P 1060	P 1061	P 1062	P 1063	P 1064	P 1065	P 1066	P 1067	P 1068	P 1069	P 1070	P 1071	P 1072	P 1073	P 1074	P 1075	P 1076	P 1077	P 1078	P 1079	P 1080	P 1081	P 1082	P 1083	P 1084	P 1085	P 1086	P 1087	P 1088	P 1089	P 1090	P 1091	P 1092	P 1093	P 1094	P 1095	P 1096	P 1097	P 1098	P 1099	P 1100	P 1101	P 1102	P 1103	P 1104	P 1105	P 1106	P 1107	P 1108	P 1109	P 1110	P 1111	P 1112	P 1113	P 1114	P 1115	P 1116	P 1117	P 1118	P 1119	P 1120	P 1121	P 1122	P 1123	P 1124	P 1125	P 1126	P 1127	P 1128	P 1129	P 1130	P 1131	P 1132	P 1133	P 1134	P 1135	P 1136	P 1137	P 1138	P 1139	P 1140	P 1141	P 1142	P 1143	P 1144	P 1145	P 1146	P 1147	P 1148	P 1149	P 1150	P 1151	P 1152	P 1153	P 1154	P 1155	P 1156	P 1157	P 1158	P 1159	P 1160	P 1161	P 1162	P 1163	P 1164	P 1165	P 1166	P 1167	P 1168	P 1169	P 1170	P 1171	P 1172	P 1173	P 1174	P 1175	P 1176	P 1177	P 1178	P 1179	P 1180	P 1181	P 1182	P 1183	P 1184	P 1185	P 1186	P 1187	P 1188	P 1189	P 1190	P 1191	P 1192	P 1193	P 1194	P 1195	P 1196	P 1197	P 1198	P 1199	P 1200	P 1201	P 1202	P 1203	P 1204	P 1205	P 1206	P 1207	P 1208	P 1209	P 1210	P 1211	P 1212	P 1213	P 1214	P 1215	P 1216	P 1217	P 1218	P 1219	P 1220	P 1221	P 1222	P 1223	P 1224	P 1225	P 1226	P 1227	P 1228	P 1229	P 1230	P 1231	P 1232	P 1233	P 1234	P 1235	P 1236	P 1237	P 1238	P 1239	P 1240	P 1241	P 1242	P 1243	P 1244	P 1245	P 1246	P 1247	P 1248	P 1249	P 1250	P 1251	P 1252	P 1253	P 1254	P 1255	P 1256	P 1257	P 1258	P 1259	P 1260	P 1261	P 1262	P 1263	P 1264	P 1265	P 1266	P 1267	P 1268	P 1269	P 1270	P 1271	P 1272	P 1273	P 1274	P 1275	P 1276	P 1277	P 1278	P 1279	P 1280	P 1281	P 1282	P 1283	P 1284	P 1285	P 1286	P 1287	P 1288	P 1289	P 1290	P 1291	P 1292	P 1293	P 1294	P 1295	P 1296	P 1297	P 1298	P 1299	P 1300	P 1301	P 1302	P 1303	P 1304	P 1305	P 1306	P 1307	P 1308	P 1309	P 1310	P 1311	P 1312	P 1313	P 1314	P 1315	P 1316	P 1317	P 1318	P 1319	P 1320	P 1321	P 1322	P 1323	P 1324	P 1325	P 1326	P 1327	P 1328	P 1329	P 1330	P 1331	P 1332	P 1333	P 1334	P 1335	P 1336	P 1337	P 1338	P 1339	P 1340	P 1341	P 1342	P 134
-----	-----	-----	---------	-----	---------	---------	---------	-----	-----	-----	-----	-----	-----	-----	-----	-----	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	--------	-------

F	F	F	0	F	1	0	9	0	2	0	0	0	A	0	0	1	0	0	0	5	B7	8
F	F	D	0	E	0	0	3	1	0	0	2	0	0	0	2	F	0	1	0	0		

2.2.11 Fast Bulk Read

This instruction is an enhanced high-speed communication instruction compared to batch reading. Regardless of the number of cascaded devices, when using the fast batch reading instruction, all devices will aggregate, generate, and return a single status data packet, which is equivalent to a single device returning a single status data packet. The data packet format of this instruction is exactly the same as that of the batch reading instruction.

Function Compared with Bulk Read, Enhanced Instruction enables faster communication

ID 0XFE

Length 0X18

Instruction 0x9A

Parameter 1 [Device 1] ID

Parameter 2 [Device 1] Low-order ByteDance of the starting address

Parameter 3 [Device 1] High-order ByteDance of the starting address

Parameter 4 [Device 1] Low-order ByteDance (X1) from data

Parameter 5 [Device 1] Higher-order ByteDance (X1) from data

Parameter 6 [Device 2] ID

Parameter 7 [Device 2] Low-order Byte of the starting address

Parameter 8 [Device 2] High-order Byte of the starting address

Parameter 9 [Device 2] Low-order ByteDance (X2) from data

Parameter 10 [Device 2] Higher-order ByteDance (X2) from data

Explanation of Return Packet Parameters

Parameter 1 [Device 1] ID

Parameter 2 [Device 1] 1st ByteDance data

Parameter 3 [Device 1] 2nd ByteDance Data

... ..

Parameter x1+1 [Device 1] ByteDance X1 data

Parameter X1+2 [Device 1] CRC Low ByteDance

Parameter X1+3 [Device 1] CRC High ByteDance

Parameter X1+4 [Device 2] Error Code

Parameter X1+5[Device 2] ID

Parameter X1+5+1[Device 2] 1st ByteDance data

Parameter X1+5+2[Device 2] 2nd ByteDance data

... ..

Parameter X1+5 +X2 [Device 2] ByteDance data of the X2-th byte

Parameter X1+5 +X2 +1 [Device 2] CRC Low ByteDance

Parameter X1 + 5 + X2 + 2[Device 2] CRC High ByteDance

... ..

Parameter X1+5+X2+... 5+Xn [Device n] Returned data of the Xn-th ByteDance

Example: Simultaneously set the current position of ID3 (132, 0x0084, 4[ByteDance]) to 166 (0x000000A6)

ID4 Current Temperature (146, 0x0092, 1[byte]) is 31 (0x1F)

ID7 Current PWM (124, 0x007C, 2[byte]) is 421 (0x01A5)

Fast Bulk Read Instruction Frame:**FF FF FD 00 FE 12 00 9A 03 84 00 04 00 07 7C 00 02 00 04 92 00 01 00 20 F2** (Send in Hexadecimal)

H 1	H 2	H 3	R S R V	I D	L E N 1	L E N 2	I N S T	P 1	P 2	P 3	P 4	P 5	P 6	P 7	P 8	P 9	P 10	P 11	P 12	P 13	P 14	P 15	C R C 1	C R C 2
0 x F F	0 x F F	0 x F D	0 x 0 0	0 x F E	0 x 1 2	0 x 0 0	0 x 9 A	0 x 0 3	0 x 8 4	0 x 0 0	0 x 0 4	0 x 0 0	0 x 0 7	0 x 7 C	0 x 0 0	0 x 0 2	0 x 0 0	0 x 0 4	0 x 9 2	0 x 0 0	0 x 0 1	0 x 0 0	0 x 2 0	0 x F 2

Return Frame:

FF FF FD 00 FE 14 00 55 00 03 A6 00 00 00 67 A4

00 07 A5 01 24 74

00 04 1F D9 C1

Return packet disassembly: ID 3 status data packet return packet:**FF FF FD 00 FE 14 00 55 00 03 A6 00 00 00 67 A4** (sent in hexadecimal)

H	H	H	RS	ID	L	L	IN	E	ID	D	D	D	D	CR	C
1	2	3	RV		E	E	S	R	1	1	2	3	4	C1	R
					N	N	T	R							C2
					1	2									

0x	0x	0x	0x	0x	0x	0x	0x	0x	0x	0x	0x	0x	0x	0x	0x
F	F	F	00	FE	14	00	55	00	03	A	00	00	00	67	A4
F	F	D								6					

ID7 Return Packet:00 07 A5 01 24 74(Sent in Hexadecimal)

ERR	ID2	D1	D2	CRC1	CRC2
0x00	0x07	0xA5	0x01	0x24	0x74

ID4 Return Packet:00 04 1F D9 C1 00 04 1F (Sent in Hexadecimal)

ERR	ID2	D1	CRC1	CRC2
0x00	0x04	0x1F	0xD9	0xC1

CAN Communication Control Protocol V2.0

CAN communication currently supports multiple protocols: proprietary protocol and MIT protocol. The default communication protocol is the MIT protocol. The communication protocol type can be switched by modifying the register value. After switching the protocol type, the device needs to be restarted to take effect.

The CAN identity of SIAR series motors is jointly defined by registers: ID(7) and CAN_ID(14), where ID(7) is used for the motor to receive CAN data message ID, and CAN_ID(14) is used for the motor to feedback CAN data message ID. During the CAN communication process of the motor, using different values for the received and sent data message IDs reduces data transmission conflicts during the CAN communication process and improves the reliability of network communication. It is recommended to set different values for the received data ID(7) and the feedback data CAN_ID(14), with ID(7) < CAN_ID(14), and the priority of received data is higher than that of sent data, which can achieve better control effects.

Unless otherwise specified in this document, all multi-byte register content transmissions use **little-endian byte order**, i.e., the low-order byte first and the high-order byte last.

1. Private Protocol Data Frame Format

CAN Private Communication Protocol, with a default baud rate of 1Mbps, uses the extended frame format. The definitions of each ByteDance segment are as follows.

Data Domain	29-bit ID			8 Bytes Data
Size	Bit28~Bit24	Bit23~Bit8	Bit7~Bit0	Byte0~Byte7
Description	Message Type	Data Area 2	Target Motor ID	Data Area 1

1.1 Device Search (0x00)

Data Domain	29-bit ID	8 Bytes Data
-------------	-----------	--------------

Size	Bit28~Bit24	Bit23~Bit8	Bit7~Bit0	Byte0~Byte7
Description	0x00	0x00	Target Motor ID	None

Response Frame:

Data Domain	29-bit ID			8 Bytes Data
Size	Bit28~Bit24	Bit23~Bit8	Bit7~Bit0	Byte0~Byte7
Description	0x00	Bit16~23: 0x00 Bit15~8: Target Motor ID	Target Motor CAN_ID	Byte0~Byte1: Motor Model Byte2~Byte4: Motor firmware version number Byte5~Byte7: None

1.2 Motor feedback data (0x01)

Data Domain	29-bit ID			8 Bytes Data
Size	Bit28~Bit24	Bit23~Bit8	Bit7~Bit0	Byte0~Byte7
Description	0x01	Bit16~23: Motor Fault Code (495) Bit16: Power supply voltage anomaly Bit17: Reserved Bit18: Reserved Bit19: Reserved Bit20: FOC timeout Bit21: Overload Bit22: Motor temperature overheat	Target Motor CAN_ID	Byte0 ~ Byte1: Current position [0~65535] corresponds to the maximum and minimum values of the MIT position (registers 330, 332) Byte2~Byte3: Current speed [0~65535] corresponds to the maximum and minimum values of MIT speed (registers

		Bit23: Temperature sensing anomaly		334, 336)
		Bit15~Bit8: Target Motor ID		Byte4~Byte5: Current torque [0~65535] corresponding to the maximum and minimum values of MIT torque; Byte6: MOS temperature (570)
				Byte7: Motor Temperature (571)
				The above data has the low byte first and the high byte last

1.3 Set Motor Control Mode (0x02)

Data Domain	29-bit ID			8 Bytes Data
Size	Bit28~Bit24	Bit23~Bit8	Bit7~Bit0	Byte0~Byte7
Description	0x02	0x00	Target Motor ID	Byte0: 0: Current mode; 1: Speed mode; 3: Position mode; 6: MIT mode. Byte1~Byte7: None

Response Frame: Motor Feedback Data (0x01)

1.4 Motor Enable (0x03)

Data Domain	29-bit ID			8 Bytes Data
Size	Bit28~Bit24	Bit23~Bit8	Bit7~Bit0	Byte0~Byte7
Description	0x03	0x00	Target Motor ID	None

Response Frame: Motor Feedback Data (0x01)

1.5 Motor Disabled (0x04)

Data Domain	29-bit ID			8 Bytes Data
Size	Bit28~Bit24	Bit23~Bit8	Bit7~Bit0	Byte0~Byte7
Description	0x04	0x00	Target Motor ID	None

Response Frame: Motor Feedback Data (0x01)

1.6 Set the current position as the mechanical zero position (0x05)

Data Domain	29-bit ID			8 Bytes Data
Size	Bit28~Bit24	Bit23~Bit8	Bit7~Bit0	Byte0~Byte7
Description	0x05	0x00	Target Motor ID	None

Response Frame: Motor Feedback Data (0x01)

1.7 Motion Control Mode (MIT) Motor Command (0x06)

This command takes effect when the control mode is MIT mode and the torque switch is turned on.

Data Domain	29-bit ID			8 Bytes Data
Size	Bit28~Bit24	Bit23~Bit8	Bit7~Bit0	Byte0~Byte7
Description	0x06	Bit23~Bit8: Target torque [0~65535] corresponds to the maximum and	Target Motor ID	Byte0~Byte1: Target position [0~65535] corresponds to the maximum and minimum values of MIT position (registers 330, 332) Byte2~Byte3: Target speed

		minimum values of MIT torque		[0~65535] corresponds to the maximum and minimum values of MIT speed (registers 334, 336) Byte4~Byte5: Kp [0~65535] corresponds to the maximum and minimum values of the MIT position proportional gain (registers 342, 344) Byte6~Byte7: Kd [0~65535] corresponds to the maximum and minimum values of the MIT position differential gain (Registers 346, 348)
--	--	------------------------------	--	--

Response Frame: Motor Feedback Data (0x01)

Note: The maximum and minimum values of the proportional conversion of position, velocity, torque, KP, KD, etc. in MIT control are stored in the MIT-related registers. For details, please refer to the content in Section 3, "Conversion between Floating-Point Numbers and Integers in CAN Communication MIT Commands".

1.8 Position Control Mode Motor Command (0x07)

This command takes effect when the control mode is: Position Mode and the torque switch is turned on.

Data Domain	29-bit ID			8 Bytes Data
Size	Bit28~Bit24	Bit23~Bit8	Bit7~Bit0	Byte0~Byte7
Description	0x07	0x00	Target Motor ID	Byte0~Byte3: Target position. Low ByteDance first, high ByteDance last

Response Frame: Motor Feedback Data (0x01)

1.9 Motor Command for Speed Control Mode (0x08)

This command takes effect when the control mode is: speed mode and the torque switch is turned on.

Data Domain	29-bit ID			8 Bytes Data
Size	Bit28~Bit24	Bit23~Bit8	Bit7~Bit0	Byte0~Byte7
Description	0x08	0x00	Target Motor ID	Byte0~Byte3: Target speed. Low ByteDance first, high ByteDance last

Response Frame: Motor Feedback Data (0x01)

1.10 Current Control Mode Motor Command (0x09)

This command takes effect when the control mode is: speed mode and the torque switch is turned on.

Data Domain	29-bit ID			8 Bytes Data
Size	Bit28~Bit24	Bit23~Bit8	Bit7~Bit0	Byte0~Byte7
Description	0x09	0x00	Target Motor ID	Byte0~Byte1: Target current. Low ByteDance first, high ByteDance last

Response Frame: Motor Feedback Data (0x01)

1.11 Write Register (0x0A)

This command is used to modify the register values starting from the specified address, with a maximum of 8 ByteDance written at a time.

Data Domain	29-bit ID			8 Bytes Data
Size	Bit28~Bit24	Bit23~Bit8	Bit7~Bit0	Byte0~Byte7
Description	0x0A	Bit23~Bit8: Register start address	Target Motor ID	Byte0~Byte7: Data to be written to the register Note: Fill in as many

				ByteDance data as needed. Maximum 8 ByteDance.
--	--	--	--	--

Response Frame: Motor Feedback Data (0x01)

1.12 Read Register (0x0B)

This command reads the register values starting from the specified address, with a maximum of 8 ByteDance read at a time.

Data Domain	29-bit ID			8 Bytes Data
Size	Bit28~Bit24	Bit23~Bit8	Bit7~Bit0	Byte0~Byte7
Description	0x0B	0x00	Target Motor ID	Byte0~Byte1: Register start address Byte2: Read Length

Response Frame:

Data Domain	29-bit ID			8 Bytes Data
Size	Bit28~Bit24	Bit23~Bit8	Bit7~Bit0	Byte0~Byte7
Description	0x0B	0x00	Target Motor CAN_ID	Byte0~Byte7: Data of the specified register

1.13 Read Motor Status 1 (0x0C)

Data Domain	29-bit ID			8 Bytes Data
Size	Bit28~Bit24	Bit23~Bit8	Bit7~Bit0	Byte0~Byte7
Description	0x0C	0x00	Target Motor ID	None

Response Frame: Motor Feedback Data (0x01)

1.14 Read Motor Status 2 (0x0D)

Data Domain	29-bit ID			8 Bytes Data
Size	Bit28~Bit24	Bit23~Bit8	Bit7~Bit0	Byte0~Byte7
Description	0x0D	0x00	Target Motor ID	None

This command can read multiple status data at once: Operator_mode, ExencoderBits, Torque, InputVoltage, ErrorCode.

Response Frame

Data Domain	29-bit ID			8 Bytes Data
Size	Bit28~Bit 24	Bit23~Bit8	Bit7~Bit0	Byte0~Byte7
Description	0x0D	0x00	Target Motor CAN_ID	Byte0: OperatorMode Byte1: ExencoderBits Byte2: Torque Byte3: INPUT_VOLTAGE(L) Byte4: INPUT_VOLTAGE(H) Byte5: ErrorCode Byte6: T_MOS Byte7: T_Rotor

- OperatorMode: Control mode (speed mode, current mode, position mode, MIT mode, etc.). See the memory table definition document for details
 - ExencoderBits: External Encoder Bit Count
 - Torque: Motor Torque Switch
- 6. INPUT_VOLTAGE: Current Input Voltage
- 3. ErrorCode: Error code. See the memory table definition document for details
- 4. T_MOS: Represents the average temperature of the driving upper MOS, with the unit of °C
- 5. T_Rotor: Represents the average temperature of the internal coils of the motor, with the unit of °C

2. MIT Protocol Data Frame Format

The default baud rate for CAN MIT protocol communication is 1Mbps, using the standard frame format. The definitions of each ByteDance segment are as follows.

Data Domain	11-digit ID	8 Bytes Data
Size	Bit10~Bit0	Byte0~Byte7
Description	Target Motor ID	Data Area

In the MIT protocol, message types are distinguished by CAN data length. The CAN data message ID sent to the motor is filled with the value of motor register ID (7), and the motor feedback data message ID is filled with the value of motor register CAN_ID (14). The following explains the content of the CAN data area.

2.1 Device Search (PING)

(1) Send data

Data Length: 1 ByteDance

Data Content: 0x01

(2) Return Data

Data Length: 8 ByteDance

Data Content:

DATA0	DATA1	DATA2	DATA3
485_ID	CAN_ID	MODEL_NUMBER (L)	MODEL_NUMB ER(H)
DATA4	DATA5	DATA6	DATA7
MAJOR_FIRMWA RE_VERSION	MINOR_FIRMWA RE_VERSION	PATCH_FIRMWA RE_VERSION	DRIVE_MODE

6. 485_ID: 485 Communication ID
3. CAN_ID: CAN Communication ID
4. MODEL_NUMBER: Motor Model (2 ByteDance)
5. MAJOR_FIRMWARE_VERSION: Major Firmware Version Number
6. MINOR_FIRMWARE_VERSION: Minor Firmware Version Number

7. PATCH_FIRMWARE_VERSION: Firmware Additional Version Number
8. DRIVE_MODE: Drive Mode. See the Memory Table Definition Document for details

2.2 Read device status

This command can read multiple status data at once: Operator_mode, ExencoderBits, Torque, InputVoltage, ErrorCode.

(1) Send data

Data Length: 1 ByteDance

Data Content: 0x02

(2) Return Data

Data Length: 8 ByteDance

Data Content:

DATA0	DATA1	DATA2	DATA3
OperatorMode	ExencoderBits	Torque	INPUT_VOLTAGE (L)
DATA4	DATA5	DATA6	DATA7
INPUT_VOLTAGE (H)	ErrorCode	T_MOS	T_Rotor

3. OperatorMode: Control mode (speed mode, current mode, position mode, MIT mode, etc.). See the memory table definition document for details
4. ExencoderBits: External Encoder Bit Count
5. Torque: Motor Torque Switch
6. INPUT_VOLTAGE: Current Input Voltage
3. ErrorCode: Error code. See the memory table definition document for details
4. T_MOS: Represents the average temperature of the driving upper MOS, with the unit of °C
4. T_Rotor: Represents the average temperature of the internal coils of the motor, with the unit of °C

2.3 Read Register

This command reads the register values starting from the specified address, with a maximum of 8 ByteDance read at a time.

(1) Send data

Data Length: 3 ByteDance

Data Content:

DATA0	DATA1	DATA2
Address (L)	Address (H)	Len

2. Address: Register start address (2 ByteDance). See the memory table definition document for details

3. Len: Read ByteDance length (less than or equal to 8)

(2) Return Data

Read successful, return register content; read failed, no return.

Data Length: n ByteDance

Data Content: Specify the value of the register to be read

2.4 Write Register

This command is used to modify the register value starting from the specified address, writing up to 4 ByteDance at a time.

(1) Send data

Data Length: 7 ByteDance

Data Content:

DATA0	DATA1	DATA2	DATA3-DATA6
Address (L)	Address (L)	Len	Value

4. Address: Register start address (2 ByteDance). See the memory table definition document for details

5. Len: Write register ByteDance length (less than or equal to 4)

2. Value: The value to be written to the register. (If it is less than 4 ByteDance, it needs to be padded with FF.)

(2) Return Data

Data Length: 1 ByteDance

Data Content: 0x00: Failure; 0x01: Success

2.5 Turn on the torque switch

(1) Send data

Data Length: 8 ByteDance

Data content: FF FF FF FF FF FF FC

(2) Return Data

Data Length: 8 ByteDance

DATA0	DATA1	DATA2	DATA3	DATA4	DATA5	DATA6	DATA7
ID ERR <<4	POS[15 :8]	POS[7: 0]	VEL[11: 4]	VEL[3:0] T[11:8]	T[7:0]	T_MOS	T_Rotor

3. ID: The ID of the controller, taking the lower 8 bits of CAN_ID
4. ERR: Indicates motor failure
- POS: Indicates the current position of the motor
- VEL: Represents the current speed of the motor
- T: Represents the torque status signal of the motor
- T_MOS: Represents the average temperature of the driving upper MOS, with the unit of °C
- T_Rotor: Represents the average temperature of the internal coils of the motor, with the unit of °C

Note: Position, velocity, and torque use a linear mapping relationship to convert floating-point data into signed fixed-point data, where position uses 16-bit data, and both velocity and torque use 12-bit data.

2.6 Turn off the torque switch

(1) Send data

Data Length: 8 ByteDance

Data Content: FF FF FF FF FF FF FD

No return data

Data Length: 0 ByteDance

2.7 Set the current position to the zero position

(1) Send data

Data Length: 8 ByteDance

Data Content: FF FF FF FF FF FF FE

No return data

Data Length: 0 ByteDance

2.8 MIT Control Command

(1) Send data

Data Length: 8 ByteDance

Data Content:

DATA0	DATA1	DATA2	DATA3	DATA4	DATA5	DATA6	DATA7
p_des [15:8]	p_des [7:0]	v_des [11:4]	v_des[3:0] Kp[11:8]	Kp [7:0]	Kd [11:4]	Kd[3:0] t_ff[11:8]	t_ff[7:0]

- P_des: Position Setpoint
- V_des: Speed Setpoint
- Kp: Position Proportional Coefficient
- Kd: Position Differential Coefficient
- T_ff: Torque Setpoint

Each parameter conforms to the previous mapping relationship, where the ranges of p_des, v_des, t_ff, Kp, and Kd can be set through the host computer software. A standard CAN data frame has only 8 ByteDance, and the MIT control command format combines the five parameters of Position, Velocity, Kp, Kd, and Torque bitwise into 8 ByteDance. Among them: Position occupies 2 ByteDance (16 bits), Velocity occupies 12 bits, Kp occupies 12 bits, and Kd occupies 12 bits.

(2) Return Data

Data Length: 8 ByteDance

DATA0	DATA1	DATA2	DATA3	DATA4	DATA5	DATA6	DATA7
ID ERR <<4	POS[15:8]	POS[7:0]	VEL[11:4]	VEL[3:0] T[11:8]	T[7:0]	T_MOS	T_Rotor

- ID: The ID of the controller, taking the lower 8 bits of CAN_ID
- ERR: Motor failure
- POS: Current position of the motor
- VEL: Current speed of the motor
- T: Torque status signal of the motor
- T_MOS: Average temperature of the upper MOS driver, unit: °C
- T_Rotor: Average temperature of the internal coils of the motor, unit: °C

3. Conversion between Floating-Point Numbers and Integers in CAN Communication MIT Commands

The MIT command converts floating-point data proportionally into integers and sends them to the drive, which then converts the received integers proportionally back into floating-point data.

This part of the conversion uses two conversion functions, `uint_to_float` and `float_to_uint`, for conversion. The specific implementation of this part is as follows:

```
SQL
float uint_to_float(int x_int, float x_min, float x_max, int bits)
{
    /// converts unsigned int to float, given range and number of
    bits ///
    float span = x_max - x_min;
    float offset = x_min;
    return ((float)x_int)*span/((float)((1<<bits)-1)) + offset;
}

int float_to_uint(float x, float x_min, float x_max, int bits)
{
    /// Converts a float to an unsigned int, given range and
    number of bits ///
    float span = x_max - x_min;
    float offset = x_min;
    return (int) ((x-offset)*((float)((1<<bits)-1))/span);
}
```

These two conversion functions need to first determine the maximum and minimum values of the two equal-proportion conversions. **The maximum and minimum values of the equal-proportion conversions for position, velocity, torque, KP, and KD in MIT control are stored in registers, and the maximum and minimum ranges can be modified by changing the register values.** These parameters can be adjusted according to the actual parameters of the motor. **However, when sending control commands, they must be consistent with the set values; otherwise, the control commands will be scaled proportionally.**

The definitions of MIT-related registers are as follows.

Name	Register Address	ByteDance	Unit	Default Value
MIT Maximum Position	330	2	0.01 [rad]	1250
MIT Minimum Position	332	2	0.01 [rad]	-1250
MIT Maximum Speed	334	2	0.01 [rad/s]	3000
MIT Minimum Speed	336	2	0.01 [rad/s]	-3000
MIT Maximum Torque	338	2	0.002 [N*M]	5000
MIT Minimum Torque	340	2	0.002 [N*M]	-5000
MIT's largest KP	342	2	0.01	10000
MIT Minimum KP	344	2	0.01	0
MIT Maximum KD	346	2	0.01	500
MIT Minimum KD	348	2	0.01	0
MIT Control KP	350	2	0.01	7200
MIT Controls KD	352	2	0.01	150

The encapsulation of the MIT control command sending function is as follows.

Markdown

```

/*****
*****
* @brief:      mit_ctrl: MIT 模式下的电机控制函数
* @param[in]:  hcan:      指向 CAN_HandleTypeDef 结构的指针，用于指定
CAN 总线
* @param[in]:  motor_id: 机 ID，指定目标电机
* @param[in]:  pos:      位置给定值
* @param[in]:  vel:      速度给定值
* @param[in]:  kp:      位置比例系数
* @param[in]:  kd:      速度微分系数
* @param[in]:  torq:     转矩给定值
* @retval:     void
* @details:    通过 CAN 总线向电机发送 MIT 模式下的控制帧。
*****
*****/
void mit_ctrl(hcan_t* hcan, uint16_t motor_id, float pos, float
vel,float kp, float kd, float torq)
{
    uint8_t data[8];
    uint16_t pos_tmp,vel_tmp,kp_tmp,kd_tmp,tor_tmp;
    uint16_t id = motor_id + MIT_MODE;

    pos_tmp = float_to_uint(pos,  P_MIN,  P_MAX,  16);
    vel_tmp = float_to_uint(vel,  V_MIN,  V_MAX,  12);
    kp_tmp  = float_to_uint(kp,   KP_MIN, KP_MAX,  12);
    kd_tmp  = float_to_uint(kd,   KD_MIN, KD_MAX,  12);
    tor_tmp = float_to_uint(torq, T_MIN,  T_MAX,  12);

    data[0] = (pos_tmp >> 8);
    data[1] = pos_tmp;
    data[2] = (vel_tmp >> 4);
    data[3] = ((vel_tmp&0xF)<<4)|(kp_tmp>>8);
    data[4] = kp_tmp&0xFF;
    data[5] = (kd_tmp >> 4);
    data[6] = ((kd_tmp&0xF)<<4)|(tor_tmp>>8);
    data[7] = tor_tmp&0xFF;

    canx_send_data(hcan, id, data, 8);
}

```

The encapsulation of the parsing function for the return data of MIT control commands is as follows

Markdown

```

/*****
*****
* @brief:      获取电机反馈数据函数
* @param[in]:  motor:    指向 motor_t 结构的指针，包含电机相关信息和
反馈数据
* @param[in]:  rx_data:  指向包含反馈数据的数组指针
* @param[in]:  data_len: 数据长度
* @retval:     void
* @details:    从接收到的数据中提取电机的反馈信息，包括电机 ID、 状态、
位置、速度、扭矩相关温度参数、寄存器数据等
*****
*****/
void can_fbdata(Joint_Motor_t *motor, uint8_t *rx_data, uint32_t
data_len)
{
    if(data_len==FDCAN_DLC_BYTES_8)
    { //返回的数据有 8 个字节
        motor->para.id = (rx_data[0])&0x0F;
        motor->para.state = (rx_data[0])>>4;
        motor->para.p_int=(rx_data[1]<<8)|rx_data[2];
        motor->para.v_int=(rx_data[3]<<4)|(rx_data[4]>>4);
        motor->para.t_int=((rx_data[4]&0xF)<<8)|rx_data[5];
        motor->para.pos = uint_to_float(motor->para.p_int, P_MIN,
P_MAX, 16); // (-12.5,12.5)
        motor->para.vel = uint_to_float(motor->para.v_int, V_MIN,
V_MAX, 12); // (-30.0,30.0)
        motor->para.tor = uint_to_float(motor->para.t_int, T_MIN,
T_MAX, 12); // (-10.0,10.0)
        motor->para.Tmos = (float)(rx_data[6]);
        motor->para.Tcoil = (float)(rx_data[7]);
    }
}

```

MIT Protocol Test Demo

To control the motor using this Demo program, it is necessary to use the standard CAN communication module (SLCAN protocol) in conjunction, and at the same time, a Python 3.0 or higher environment is required, with the can module installed.

Plain Text

```
#运行命令安装 can 模块  
pip install python-can
```

The Demo program implements the following functions: scanning and controlling multiple motors. Variables such as COM port, baud rate, target position range, target position change step, and control frequency can be modified according to actual requirements.

- Modify COM Port

Plain Text

```
SLAN_PORT = 'COM29'
```

2. Modify the motor ID

Plain Text

```
MOTOR_CAN_IDS = [ 0x01, 0x02,0x3, 0x04, 0x05,0x06]
```

3. Modify the control frequency

Plain Text

```
LOOP_FREQUENCY = 1000
```

4. Modify the target position range and step size

Plain Text

```
MAX_GOAL_POS_RAD = 1.0 * math.pi # 180 度, 可根据需要调整  
MIN_GOAL_POS_RAD = -1.0 * math.pi # -180 度  
GOAL_POS_STEP_RAD = 0.001 * math.pi # 每次更新的步长
```

The complete code is as follows:

Python

```
import can
```

```

import time
import struct
import math

RAD_TO_SIGNAL = 2607.5946

# 定义电机命令
CMD_PING = 0x01
CMD_READ_STATUS = 0x02 # “02 查询”

# 寄存器地址常量
PRESENT_POSITION = 0x228 # 当前位置寄存器地址 552 = 0x228
GOAL_POSITION = 0x214 # 目标位置寄存器地址 532 = 0x214
TORQUE_ON_OFF_REGISTER = 0x200 # 扭矩开关寄存器地址 512 = 0x200
OPENTING_MODEL_REGISTER = 0x21 # Operating Mode 寄存器地址 33 = 0x21
SET_ZERO_POSITION_REGISTER = 0x34 # 设置零位寄存器 52 = 0x34
SET_ZERO_POSITION_VALUE = 0x01 # 设置零位的值

TORQUE_ON_VALUE = 0x01 # 打开扭矩的值
TORQUE_OFF_VALUE = 0x00 # 关闭扭矩的值
# 设置 OPENTING_MODEL 的值 (0: 电流模式; 1: 速度模式; 3: 位置模式; 6: MIT 模式)
OPENTING_MODEL_VALUE = 0x03

# MIT 控制参数的映射范围 (示例, 请根据实际文档调整)
P_DES_MIN = -12.5
P_DES_MAX = 12.5
V_DES_MIN = -65.0
V_DES_MAX = 65.0
T_FF_MIN = -18.0
T_FF_MAX = 18.0
KP_MIN = 0.0
KP_MAX = 500.0
KD_MIN = 0.0
KD_MAX = 5.0

# 位置/速度/扭矩的实际浮点数范围 (用于 uint_to_float 转换)
# 这些需要根据电机的实际规格来定义
POS_MIN_RANGE = -12.5 * math.pi # 假设是弧度, 根据电机文档调整
POS_MAX_RANGE = 12.5 * math.pi
VEL_MIN_RANGE = -65.0 * math.pi # 假设是弧度/秒

```

```

VEL_MAX_RANGE = 65.0 * math.pi
TORQUE_MIN_RANGE = -18.0
TORQUE_MAX_RANGE = 18.0

# 将浮点数映射到定点数的函数
def float_to_uint(value, min_val, max_val, num_bits):
    if value < min_val:
        value = min_val
    if value > max_val:
        value = max_val
    scaled_value = (value - min_val) / (max_val - min_val)
    return int(scaled_value * ((1 << num_bits) - 1))

# 将定点数映射回浮点数的函数
def uint_to_float(value, min_val, max_val, num_bits):
    scaled_value = value / ((1 << num_bits) - 1)
    return scaled_value * (max_val - min_val) + min_val

# 对于带符号的定点数，需要做额外的处理
def signed_uint_to_float(value, min_val, max_val, num_bits):
    if value & (1 << (num_bits - 1)):
        value = value - (1 << num_bits)
    range_span = max_val - min_val
    integer_span = (1 << num_bits) - 1 # 注意：对于带符号数，范围可能不是 (1<<num_bits)-1，而是 (1<<(num_bits-1))-1 到 -(1<<(num_bits-1))
    return min_val + (value / integer_span) * range_span

class MotorSLCan:
    def __init__(self, port, baudrate=1000000):
        self.port = port
        self.baudrate = baudrate
        self.bus = None

    def connect(self):
        try:
            self.bus = can.interface.Bus(interface='slcan',
                                          channel=self.port,
                                          bitrate=self.baudrate)
            print(f"Connected to SLCan on {self.port} with bitrate

```

```

{self.baudrate}")
        return True
    except Exception as e:
        print(f"Error connecting to SLCan: {e}")
        return False

    def disconnect(self):
        if self.bus:
            self.bus.shutdown()
            print("Disconnected from SLCan.")

    def send_command(self, arbitration_id, data,
is_extended_id=False):
        try:
            message = can.Message(
                arbitration_id=arbitration_id,
                is_extended_id=is_extended_id,
                data=data
            )
            self.bus.send(message)
            return True
        except Exception as e:
            print(f"Error sending CAN message: {e}")
            return False

    def receive_response(self, timeout=1.0):
        try:
            message = self.bus.recv(timeout)
            if message:
                return message
            else:
                return None
        except Exception as e:
            print(f"Error receiving CAN message: {e}")
            return None

    def ping_device(self, can_id):
        self.send_command(can_id, bytearray([CMD_PING]))
        response = self.receive_response(timeout=0.2) # PING 响应
可能需要稍长一些
        if response and len(response.data) == 8:
            _485_id = response.data[0]
            can_id_response = response.data[1]
            model_number = struct.unpack('<H',

```

```

response.data[2:4])[0]
    major_fw = response.data[4]
    minor_fw = response.data[5]
    patch_fw = response.data[6]
    drive_mode = response.data[7]
    print(f"--- PING Response (ID: {hex(can_id)}) ---")
    print(
        f" 485 ID: {hex(_485_id)}, CAN ID:
{hex(can_id_response)}, Model: {model_number}, FW:
{major_fw}.{minor_fw}.{patch_fw}")
    return {
        "485_ID": _485_id, "CAN_ID": can_id_response,
"MODEL_NUMBER": model_number,
        "MAJOR_FIRMWARE_VERSION": major_fw,
"MINOR_FIRMWARE_VERSION": minor_fw,
        "PATCH_FIRMWARE_VERSION": patch_fw, "DRIVE_MODE":
drive_mode
    }
    else:
        print(f"PING failed or invalid response for CAN ID
{hex(can_id)}.")
        return None

def read_device_status(self, can_id):
    self.send_command(can_id, bytearray([CMD_READ_STATUS]))
    response = self.receive_response(timeout=0.05) # 状态查询
响应时间可以短一些
    if response and len(response.data) == 8:
        operator_mode = response.data[0]
        exencoder_bits = response.data[1]
        torque = response.data[2]
        input_voltage = struct.unpack('<H',
response.data[3:5])[0]
        error_code = response.data[5]
        t_mos = response.data[6]
        t_rotor = response.data[7]
        # print(f"--- Device Status (ID: {hex(can_id)}) ---")
        # print(f"  Mode: {operator_mode}, Voltage:
{input_voltage / 100.0:.2f}V, Error: {hex(error_code)}, T_MOS:
{t_mos}, T_Rotor: {t_rotor}")
        return {
            "OperatorMode": operator_mode, "ExencoderBits":
exencoder_bits, "Torque": torque,
            "INPUT_VOLTAGE": input_voltage, "INPUT_VOLTAGE_V":

```

```

input_voltage / 100.0,
        "ErrorCode": error_code, "T_MOS": t_mos,
    "T_Rotor": t_rotor
    }
    else:
        # print(f"Read device status failed or invalid
response for CAN ID {hex(can_id)}.")
        return None

    def read_register(self, can_id, address, length):
        if not (1 <= length <= 8):
            print(f"Error: Register read length must be between 1
and 8. (CAN ID: {hex(can_id)})")
            return None

        data = bytearray([
            (address & 0xFF),
            ((address >> 8) & 0xFF),
            length
        ])
        self.send_command(can_id, data)
        response = self.receive_response(timeout=0.05) # 寄存器读
取响应时间可以短一些
        if response and len(response.data) == length:
            # print(f"--- Read Register (ID: {hex(can_id)}, Addr:
{hex(address)}, Len: {length}) --- Data: {response.data.hex()}")
            return response.data
        else:
            # print(f"Read register failed or invalid response for
CAN ID {hex(can_id)}, Address {hex(address)}.")
            # if response:
            #     print(f"  Received data length:
{len(response.data)}, Expected: {length}")
            #     print(f"  Received data: {response.data.hex()}")
            return None

    def write_register(self, can_id, address, value_bytes):
        if not (1 <= len(value_bytes) <= 4):
            print(
                f"Error: Register write value length must be
between 1 and 4 bytes. Got {len(value_bytes)} bytes. (CAN ID:
{hex(can_id)})")
            return False

```

```

        padded_value_bytes = bytearray(value_bytes)
        while len(padded_value_bytes) < 4:
            padded_value_bytes.append(0xFF) # 文档说是 FF 填充, 但
实际中 00 更常见, 请核实

        data = bytearray([
            (address & 0xFF),
            ((address >> 8) & 0xFF),
            len(value_bytes)
        ])
        data.extend(padded_value_bytes)

        self.send_command(can_id, data)
        response = self.receive_response(timeout=0.05) # 写寄存器
响应时间可以短一些
        if response and len(response.data) >= 1:
            status = response.data[0]
            if status == 0x01:
                # print(f"Write register (ID: {hex(can_id)}, Addr:
{hex(address)}, Val: {value_bytes.hex()}) successful (Status:
{hex(status)}).")
                return True
            else:
                print(
                    f"Write register (ID: {hex(can_id)}, Addr:
{hex(address)}, Val: {value_bytes.hex()}) failed (Status:
{hex(status)}).")
                return False
        else:
            print(f"Write register failed or invalid response for
CAN ID {hex(can_id)}, Address {hex(address)}.")
            if response:
                print(f" Received response data:
{response.data.hex()}")
            return False

    def open_torque(self, can_id):
        return self.write_register(can_id, TORQUE_ON_OFF_REGISTER,
bytearray([TORQUE_ON_VALUE]))

    def close_torque(self, can_id):
        return self.write_register(can_id, TORQUE_ON_OFF_REGISTER,
bytearray([TORQUE_OFF_VALUE]))

```

```

def set_zero_position(self, can_id):
    return self.write_register(can_id,
SET_ZERO_POSITION_REGISTER, bytearray([SET_ZERO_POSITION_VALUE]))

def set_openting_model(self, can_id):
    return self.write_register(can_id,
OPENING_MODEL_REGISTER, bytearray([OPENING_MODEL_VALUE]))

def send_mit_control(self, can_id, p_des, v_des, kp, kd,
t_ff):
    # 暂时不修改这部分，因为它不属于你当前的主要需求，但保留
    p_des_int = float_to_uint(p_des, P_DES_MIN, P_DES_MAX, 16)
    v_des_int = float_to_uint(v_des, V_DES_MIN, V_DES_MAX, 12)
    kp_int = float_to_uint(kp, KP_MIN, KP_MAX, 12)
    kd_int = float_to_uint(kd, KD_MIN, KD_MAX, 12)
    t_ff_int = float_to_uint(t_ff, T_FF_MIN, T_FF_MAX, 12)

    data = bytearray(8)
    data[0] = (p_des_int >> 8) & 0xFF
    data[1] = p_des_int & 0xFF
    data[2] = (v_des_int >> 4) & 0xFF
    data[3] = ((v_des_int & 0x0F) << 4) | ((kp_int >> 8) &
0x0F)
    data[4] = kp_int & 0xFF
    data[5] = (kd_int >> 4) & 0xFF
    data[6] = ((kd_int & 0x0F) << 4) | ((t_ff_int >> 8) &
0x0F)
    data[7] = t_ff_int & 0xFF

    self.send_command(can_id, data)
    response = self.receive_response(timeout=0.05)
    if response and len(response.data) == 8:
        return self._parse_mit_response(response.data)
    else:
        print(f"MIT control failed or invalid response for CAN
ID {hex(can_id)}.")
        return None

def _parse_mit_response(self, data):
    # 暂时不修改这部分，因为它不属于你当前的主要需求，但保留
    motor_id = data[0] & 0x0F
    err_code = (data[0] >> 4) & 0x0F
    pos_raw = (data[1] << 8) | data[2]
    vel_raw = ((data[3] << 4) | ((data[4] >> 4) & 0x0F))

```

```

        torque_raw = (((data[4] & 0x0F) << 8) | data[5])
        t_mos = data[6]
        t_rotor = data[7]
        # print(f"--- MIT/Torque Response (ID: {hex(motor_id)}) --
- Err: {err_code}, Pos: {pos_raw}, Vel: {vel_raw}, T:
{torque_raw}")
        return {
            "ID": motor_id, "ERR": err_code, "POS_RAW": pos_raw,
            "VEL_RAW": vel_raw, "T_RAW": torque_raw, "T_MOS":
t_mos, "T_Rotor": t_rotor
        }

if __name__ == "__main__":
    SLAN_PORT = 'COM29'
    # 四个驱动器的 CAN ID
    MOTOR_CAN_IDS = [ 0x01, 0x02, 0x03, 0x04, 0x05, 0x06]

    motor_comm = MotorSLCan(SLAN_PORT)

    if motor_comm.connect():
        try:
            # 存储每个电机的最后操作时间，用于调度
            last_status_query_time = {can_id: 0 for can_id in
MOTOR_CAN_IDS}
            last_pos_query_time = {can_id: 0 for can_id in
MOTOR_CAN_IDS}
            last_goal_update_time = {can_id: 0 for can_id in
MOTOR_CAN_IDS}

            # 目标位置生成器的状态
            current_goal_position_rad = {can_id: 0.0 for can_id in
MOTOR_CAN_IDS}
            goal_position_direction = {can_id: 1 for can_id in
MOTOR_CAN_IDS} # 1 for increasing, -1 for decreasing
            MAX_GOAL_POS_RAD = 1.0 * math.pi # 180 度，可根据需要
调整
            MIN_GOAL_POS_RAD = -1.0 * math.pi # -180 度
            GOAL_POS_STEP_RAD = 0.001 * math.pi # 每次更新的步长

            print("\n--- INITIALIZING MOTORS ---")
            for motor_id in MOTOR_CAN_IDS:
                print(f"\nInitializing Motor {hex(motor_id)}:")

```

```

        # 1. PING
        print(f"  Pinging...")
        motor_info = motor_comm.ping_device(motor_id)
        if not motor_info:
            print(f"  Failed to PING motor
{hex(motor_id)}. Skipping initialization.")
            continue # 如果 PING 失败，跳过这个电机的初始化

        # 2. 设置 OPENTING_MODEL
        print(f"  Setting OPENTING_MODEL...")
        if not motor_comm.set_openting_model(motor_id):
            print(f"  Failed to set OPENTING_MODEL for
motor {hex(motor_id)}.")
            continue
        #time.sleep(0.1)
        # 3. 打开扭矩开关
        print(f"  Opening torque...")
        if not motor_comm.open_torque(motor_id):
            print(f"  Failed to open torque for motor
{hex(motor_id)}.")
            continue

    print("\n--- STARTING CONTROL LOOP ---")

    # 主循环频率控制
    LOOP_FREQUENCY = 1000 # Hz
    LOOP_PERIOD = 1.0 / LOOP_FREQUENCY # seconds

    # 指令发送频率
    STATUS_QUERY_PERIOD = 0.100 # 100 ms
    POS_QUERY_PERIOD = 0.010 # 10 ms
    GOAL_UPDATE_PERIOD = 0.010 # 10 ms

    start_time = time.monotonic()
    loop_count = 0

    while True:
        loop_start_time = time.monotonic()

        for motor_id in MOTOR_CAN_IDS:
            current_time = time.monotonic()

```

```

        # ① 0x02 查询 (100ms)
        if current_time -
last_status_query_time[motor_id] >= STATUS_QUERY_PERIOD:
            status =
motor_comm.read_device_status(motor_id)
            #if status:
                # print(
                #     f"Loop {loop_count}: Motor
{hex(motor_id)} Status: Mode={status['OperatorMode']},
Volt={status['INPUT_VOLTAGE_V']:.2f}V,
Temp={status['T_MOS']}C/{status['T_Rotor']}C")
                last_status_query_time[motor_id] =
current_time

        # ② 位置查询 (10ms)
        if current_time -
last_pos_query_time[motor_id] >= POS_QUERY_PERIOD:
            # 假设 PRESENT_POSITION 寄存器返回 4 字节
            # (32 位) 表示位置
            pos_raw_bytes =
motor_comm.read_register(motor_id, PRESENT_POSITION, 4)
            if pos_raw_bytes and len(pos_raw_bytes) ==
4:
                pos_raw = struct.unpack('<i',
pos_raw_bytes)[0] # 假设无符号 16 位小端
                # 将原始整数位置转换为浮点数 (需要根据你的
                # 电机文档提供正确的映射范围和位数)
                # 这里我假设 PRESENT_POSITION 也是 16
                # 位, 范围与 MIT 的 P_DES 类似
                #actual_pos_rad =
                uint_to_float(pos_raw, POS_MIN_RANGE, POS_MAX_RANGE, 16)
                #print(
                #     f"Loop {loop_count}: Motor
{hex(motor_id)} Pos: {pos_raw} ")
                last_pos_query_time[motor_id] =
current_time

        # ③ 更新目标位置 (10ms)
        if current_time -
last_goal_update_time[motor_id] >= GOAL_UPDATE_PERIOD:
            # 简单的目标位置生成: 在 MIN_GOAL_POS_RAD 和
            # MAX_GOAL_POS_RAD 之间往复运动
            if goal_position_direction[motor_id] == 1:

```

```

        current_goal_position_rad[motor_id] +=
GOAL_POS_STEP_RAD
        if
current_goal_position_rad[motor_id] >= MAX_GOAL_POS_RAD:

current_goal_position_rad[motor_id] = MAX_GOAL_POS_RAD
        goal_position_direction[motor_id]
= -1
        else:
        current_goal_position_rad[motor_id] -=
GOAL_POS_STEP_RAD
        if current_goal_position_rad[motor_id]
<= MIN_GOAL_POS_RAD:

current_goal_position_rad[motor_id] = MIN_GOAL_POS_RAD
        goal_position_direction[motor_id]
= 1

        # 将浮点目标位置转换为电机期望的整数格式（假设
是 16 位，与 P_DES_MIN/MAX 类似）
        #goal_pos_int =
float_to_uint(current_goal_position_rad[motor_id], POS_MIN_RANGE,
POS_MAX_RANGE, 16)
        goal_pos_int = int(RAD_TO_SIGNAL *
current_goal_position_rad[motor_id])
        goal_bytes = struct.pack('<i',
goal_pos_int) # 假设有符号的 4 字节小端

        # 写入 GOAL_POSITION 寄存器
        motor_comm.write_register(motor_id,
GOAL_POSITION, goal_bytes)
        #print(f"Loop {loop_count}: Motor
{hex(motor_id)} Set Goal:
{current_goal_position_rad[motor_id]:.4f} rad (raw:
{goal_pos_int})")
        last_goal_update_time[motor_id] =
current_time

        loop_count += 1

        # 确保主循环以 3000Hz 运行
        loop_end_time = time.monotonic()
        time_elapsed = loop_end_time - loop_start_time
        time_to_sleep = LOOP_PERIOD - time_elapsed

```

```

        if time_to_sleep > 0:
            time.sleep(time_to_sleep)
        else:
            # 如果循环执行时间超过了周期, 打印警告
            #print(f"Warning: Loop {loop_count} took
{time_elapsed*1000:.2f} ms, exceeding {LOOP_PERIOD*1000:.2f} ms.")
            pass

    except KeyboardInterrupt:
        print("\nExiting program, closing torque for all
motors.")
        for motor_id in MOTOR_CAN_IDS:
            motor_comm.close_torque(motor_id)
            time.sleep(0.1) # 间隔一下, 确保命令发出
        finally:
            motor_comm.disconnect()
    else:
        print("Failed to connect to SLCAN, please check port and
device.")

```

485/TTL Communicate SDK

We provide cross-platform, multi-language C, C++, Python, and ROS interface SDKs to help you quickly use GIM products.