GOOGOL TECHNOLOGY

CPAC OtoStudio

Programming Manual of Motion Controller

GUC-ECAT Series

V1.0

2014.10

www.googoltech.com

Copyright Statement

 CAUTION	Machinery in motion can be dangerous! It is the user's responsibility to design effective error handling and safety protection methods as part of the machinery. Googol Technology shall not be liable or responsible for any incidental or consequential damages.

Contact Us

Googol Technology (Shenzhen) Ltd.

Address: 2nd Floor, West Wing, IER Building (PKU-HKUST Shenzhen Hong Kong Institution) High-tech Industrial Park, Nanshan, Shenzhen, PRC
Postal Code: 518057

Tel.: +(86) 755-26970817, 755-26970824, 755-26737236
Fax: +(86) 755- 26970821
E-mail: support@googoltech.com
URL: http://www.googoltech.com.cn

Googol Technology (HK) Ltd.

Address: Unit 1008-09, 10/F C-Bons International Center, 108 Wai Yip Street, Kwun Tong, Kowloon, Hong Kong

Tel.: +(852) 2358-1033
Fax: +(852)2719-8399
E-mail: info@googoltech.com
URL: http://www.googoltech.com

Document Version

Version	Date
1.0	2014-09-24

Foreword

Thank you for selecting Googol Technology motion controller

To repay user, we will help you establish your own control system, by providing our first-class motion controller, perfect after-sale services, and high-efficiency technical support.

More information about products of Googol Technology

Googol Technology's web site is http://www.googoltech.com. You can get more information about the company and products on our website, including company profile, product introduction, technical support, product recently released.

You can also get more information about the company and products through the phone: +(86) 755-26970817.

Technical support and after-sale services

To get our technical support and after-sale services:

E-mail: support@googoltech.com

Tel.: +(86) 755 2697-0843

Addr: Googol Technology (SZ) Ltd

2nd Floor, West Wing, IER Building (PKU-HKUST Shenzhen Hong Kong Institution) High-tech Industrial Park, Nanshan, Shenzhen, PRC.

Postal Code: 518057

Usage of this Programming Manual

By reading this manual, you will know the control functions of GUC-ECAT series motion controller, learn the usage of motion functions, and become familiar with programming of specific control function. Finally, you can program your application for controlling according to your specific control system.

User of this Programming Manual

This manual is applicable to those engineering developers who have the base knowledge of programming in C or other using Dynamic Link Library (DLL) in Windows environment, with certain work experience in motion control and understanding of the basic architecture of servo or step control.

Main Contents of this Programming Manual

This manual consists of twelve chapters, which introduced the control functions and programming of the GUC-ECAT series motion controller in detail.

Relevant Documents

For installing and debugging of GUC-ECAT series motion controller, please refer to *User's Guide of GUC-ECAT series Motion Controller* provided together with our product.

Contents

Chapter 1 Use of Motion Function Library in OtoStudio

1.1 Use of OtoStudio software library

For using the motion controller in CPAC software platform, directly run Setup to save the instruction function library of the motion controller under the default path. The library file name of GUC-X00-TPX controller is CPAC GUC_X00_TPX.lib. Since the motion controller needs to use EtherCAT bus, it is necessary to call EtherCAT private library, wherein the method is consistent with the method of using CPAC-X00-TPX.lib, and CPAC-X00-TPX.lib could be used at the same time, and the library file name is CPAC GUC-X00-TPX ECAT.lib.

1.1.1 Usage of the library in OtoStudio

1. Start the OtoStudio.exe, and create a new project;
2. Select the target platform: CPAC GUC-X00-TPX
3. The system adds CPAC GUC-X00-TPX.lib automatically;
4. Manually add CPAC GUC-X00-TPX ECAT.lib to the library file manager;

Now, users can call any commands in DLL and program their application programs in OtoStudio

Chapter 2 Return Values of Commands and Their Meanings

2.1 Return values of commands

CPAC controller works according to the motion controller commands sent by the host. These commands are encapsulated in DLL. User can call GUC-X00-TPX.lib in the library of CPAC controller to operate the motion controller when the user writes program to the host PC.

When receiving commands from the host, CPAC controller will give a feedback after checking and verifying the commands. The definitions of return values are listed in Tab 2-1.

Tab 2-1 Definition of Return Values of Motion Controller

Value	Meanings	Processing Methods
0	Command executed successful	
1	Command error	1. Check the execution condition of the current command
7	Command parameters error	1. Check the value of current command parameters
-1	Error in communication.	1. Check drive of motion controller; 2. Check connection between motion controller and host PC; 3. Change host PC; 4. Change motion controller
-6	Failure in opening the card.	1. Check drive of motion controller 2. Check if GT_Open() was called twice. 3. Check if the card was opened in another program
-7	No response of motion controller	1. Change another motion controller

It is suggested to check each command return value in user program to confirm if the execution of the command is successful, and establish necessary error treatment mechanism to assure the safety and reliability of the program.

Chapter 3 System Configuration

Before an operation in motion controller, user hs to configurate motion controller to ensure its statuses and work mode satisfy the application requirements. This process is called system configuration. With management software - Motion Controller Toolkit 2008(MCT2008 as short), which has a component for system configuration, user can configurate a motion control system. When the configuration operation is completed, MCT2008 will generate a configuration file with a cfg suffix name. When user programming a program, they can call correlative command and send configuration information to motion controller. In this way, controller can achieve system configuration. User can also use GT command to accomplish the operation of system configuration too.

3.1 Basic concepts of system configuration

Motion controller contains various software and hardware resources which can be combined freely to implement different applications.

3.1.1 Hardware resource

Digital Output resource (do): “do” represents digital output signals including servo enable output, alarm clear output and general digital output.

Digital Input resource (di): “di” represents digital input signals including limit signal input, home signal input, driver alarm input and general digital input.

Encoder resource (encoder): “encoder” will count the output pulses of encoder.

Pulse output resource (step): “step” represents pulse output channels which will send "pulse+direction" or "CW/CWW” controlling pulse.

Voltage output resource (dac): “dac” represents the voltage output channels which will send controlling voltage. The ranging in of “dac” is -10V to +10V.

3.1.2 Software resource

Profile manager resource (profile): “profile” can calculate the profile position and profile velocity depending on motion mode and motion parameters in real time, and generate velocity curve and output profile position.

Control resource (control): “control” can calculate the control output depending on control algorithm, control parameter and following error.

Axis resource (axis): “axis” combines all software and hardware resources as a unity to operate. It manages the output signal of drive alarm, limit signal, smooth stop signal and emergency stop signal. And it also has the functions like transform profile output into profile position, transform

counting position of encoder etc.

3.1.3 Resources combination

The process of combining software resources with hardware resources, and configurating basic property of all the resources is system configuration. The following two examples show the basic concept of resources combination.

Control mode of step motor can be configurated as Fig 3-1.

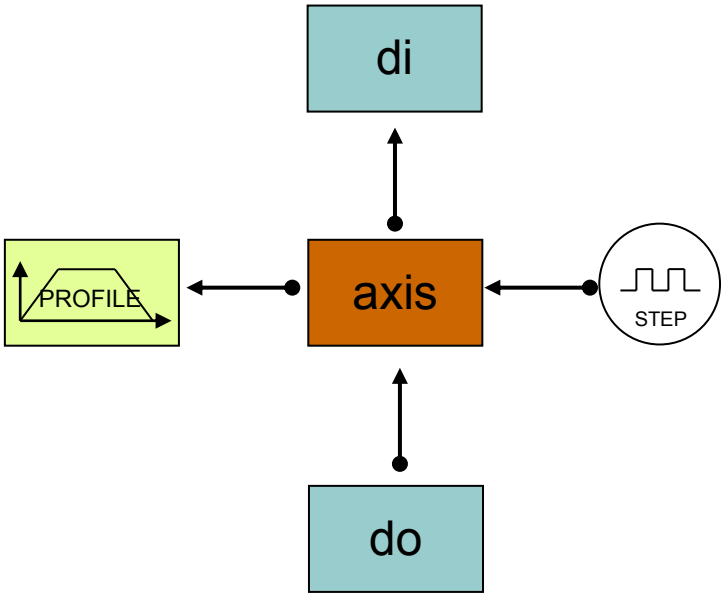

Fig 3-1 Step Control

In this example, profile manager outputs profile position to axis, axis will transform the profile position and output the data into step, which will generate the control pulses to drive motor. Axis needs to receive the digital input signals of alarm, limit switch, and smooth stop , emergency stop and so on to manage the motion. And at the same time, it also needs output digital signal of servo on to enable the motor.

Control mode of servo motor can be configurated as Fig 3-2.

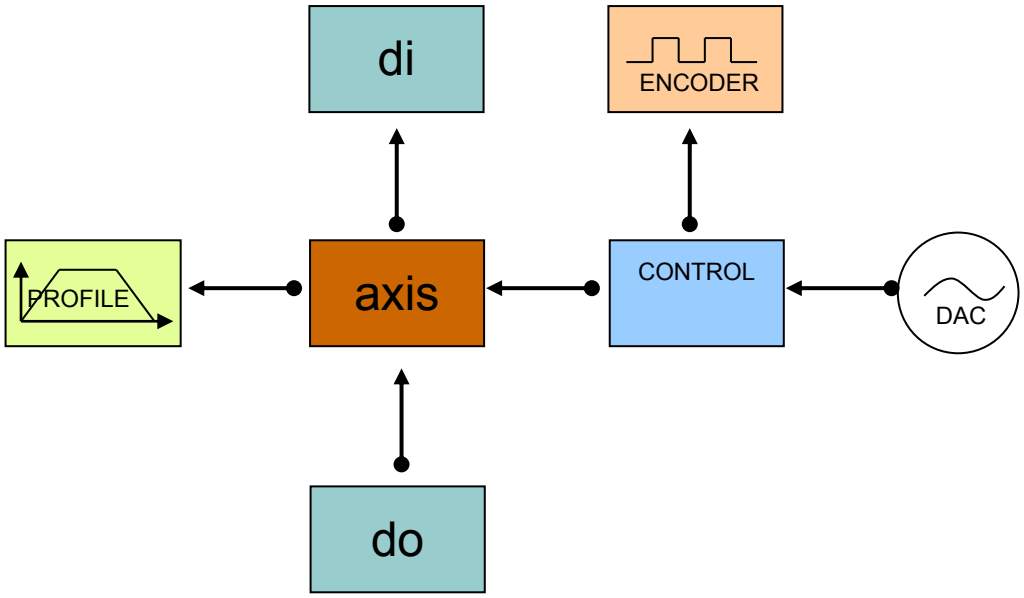

Fig 3-2 Servo Control

In this example, profile outputs profile position into axis, and then axis transforms the profile position and outputs the data into control. Control will compare the profile position with encoder position to get the following error. Then control can get real time controlled quantity calculated by specific servo control algorithm，and pass the control quantity to dac. Dac will control the motor motion by changing controlled quantity into control voltage. Axis needs to receive the digital input signals like alarm, limit, smooth stop, emergency stop and so on to manage the motion. And it also needs output digital signal of servo on to enable the motor.

3.2 System configuration tool

Googol Technology provides a motion controller management software, named Motion Controller Toolkit 2008(MCT2008), to configurate the system. The starting panel of MCT2008 is illustrated as Fig 3-3.

Fig 3-3 Motion Controller Toolkit 2008

User can start motion controller configuration panel by clicking “Tool”-> “Configuration”, in this panel user can set system configuration.

3.2.1 Axis configuration

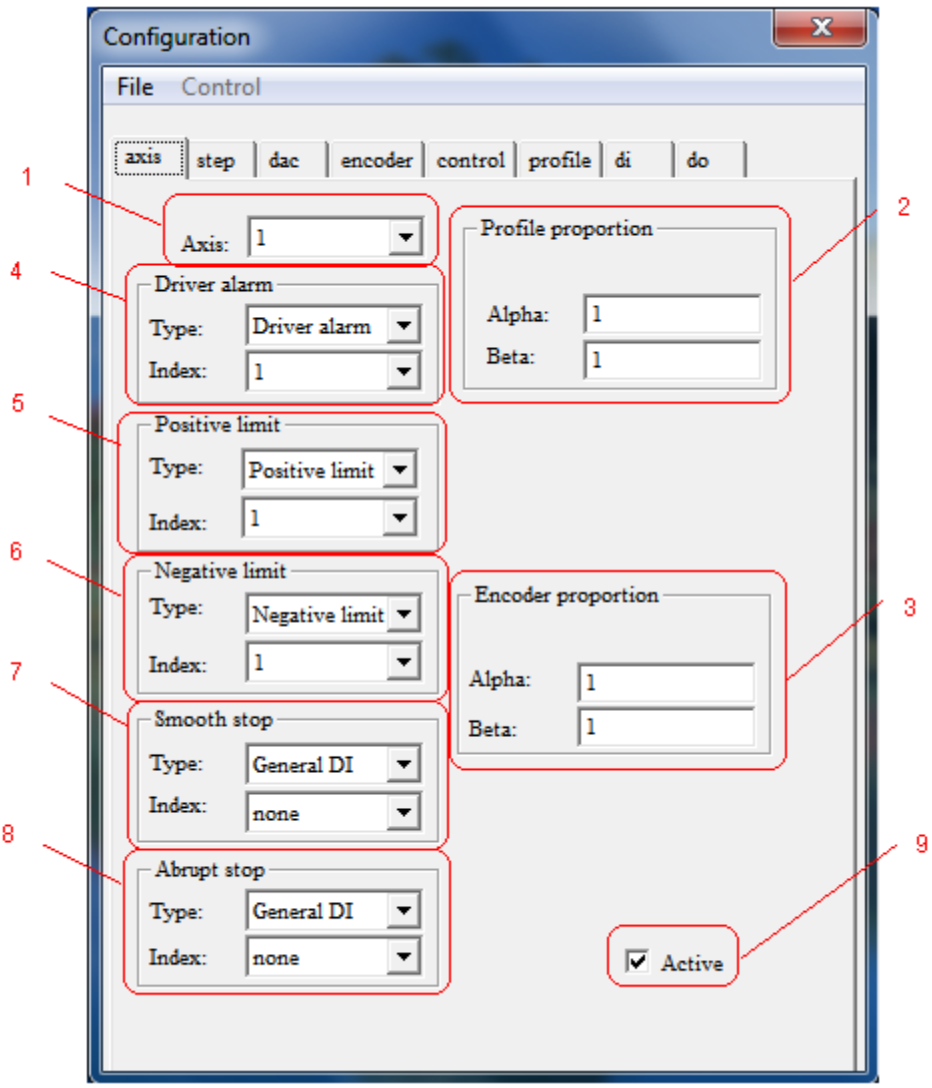

Fig 3-4 Axis configuration panel

1 Axis: select the index of axis which needs to be configurated.

2 Profile scale: If user needs to transform profile position of profile through axis, the parameters Alpha and Beta need to be set properly. The following equation represents the transformation relation:

$$\frac{\Delta P_{profile}}{\Delta P_{axis}} = \frac{Alpha}{Beta}$$

$\Delta P_{profile}$ represents the variable quantity of profile position in profile.

ΔP_{axis} represents the variable quantity of profile position in axis.

The default values of Alpha and Beta are 1, in another words, the profile position of profile will not change through axis. The ranging in of Alpha is (-32767, 0) and (0, 32767); the Beta is (-32767,0) and (0, 32767).This item can be set by GT_ProfileScale() command.

3 Encoder scale: If user needs to transform encoder position of encoder through axis, the parameters Alpha and Beta need to be set properly. The following equation represents the transformation relation:

$$\frac{\Delta E_{enc}}{\Delta E_{axis}} = \frac{Alpha}{Beta}$$

ΔE_{enc} is the variable quantity of encoder position in encoder;

ΔE_{axis} is the variable quantity of encoder position in axis.

The default values of Alpha and Beta are 1, in another words, the encoder position of encoder will not change through axis. The ranging in of Alpha is (-32767,0) and (0, 32767); the Beta is (-32767,0) and (0, 32767). This item can be set by GT_EncScale() command.

4 Drive alarm: User can choose different type of digital input as drive alarm signal. This motion controller support any digital input as alarm signals to increase the connection freedom of hardware. In the first pull-down menu, user can choose the type of digital input of alarm, by default, motion controller will choose the drive alarm as the digital input. In the second pull-down menu, user can choose the index of digital input. If user set the index of digital input as none, the alarm of corresponding axis is disabled. Drive alarm can be disabled by GT_AlarmOff() command and be enabled by GT_AlarmOn() command.

5 Positive limit: User can choose different type digital input as positive limit signal. This motion controller support any digital input as positive limit signals to increase the connection freedom of hardware. In the first pull-down menu, user can choose the type of digital input of positive limit. By default, motion controller will choose the positive limit as the digital input. In the second pull-down menu, user can choose the index of digital input. If user set the index of digital input as none, the positive limit of corresponding axis is disabled. Limit switch can be disabled by GT_LmtsOff() command and be enabled by GT_LmtsOn() command.

6 Negative limit: User can choose different type of digital input as negative limit signal. This motion controller support any digital input as negative limit signals to increase the connection freedom of hardware. By default, motion controller will choose the negative limit as the digital input. In the first pull-down menu, user can choose the type of digital input of negative limit. In the second pull-down menu, user can choose the index of digital input. If user set the index of digital input as none, the negative limit of corresponding axis is disabled. Negative limit switch can be disabled by GT_LmtsOff() command and be enabled by GT_LmtsOn() command.

7 Smooth stop: User can choose different type digital input as smooth stop signal. This motion controller support any digital input set as smooth stop signals to increase the connection freedom of hardware. In the first pull-down menu, user can choose the type of digital input of smooth stop. By default, motion controller disables the smooth stop of axes. In the second

pull-down menu, user can choose the index of smooth stop. If user set the index of digital input as none, the smooth stop of corresponding axis is disabled. The input type of smooth stop can be configurated by GT_SetStopIo() command.

8 Emergency stop: User can choose different types digital input of emergency stop signal. This motion controller support any digital input set as emergency stop signals to increase the freedom of hardware connection. In the first pull-down menu, user can choose the type of digital input of emergency stop. By default, motion controller disables the emergency stop of axes. In the second pull-down menu, user can choose the index of emergency stop. If user set the index of digital input as none, the emergency stop of corresponding axis is disabled. The input type of emergency stop can be configurated by GT_SetStopIo() command.

9 Active: If axis is not active, the calculating and management tasks of corresponding axis will be invalid. By default, all axes are active. If user does not need corresponding function of some axes, inactivate these axes will save resources of motion controller.

3.2.2 Step configuration

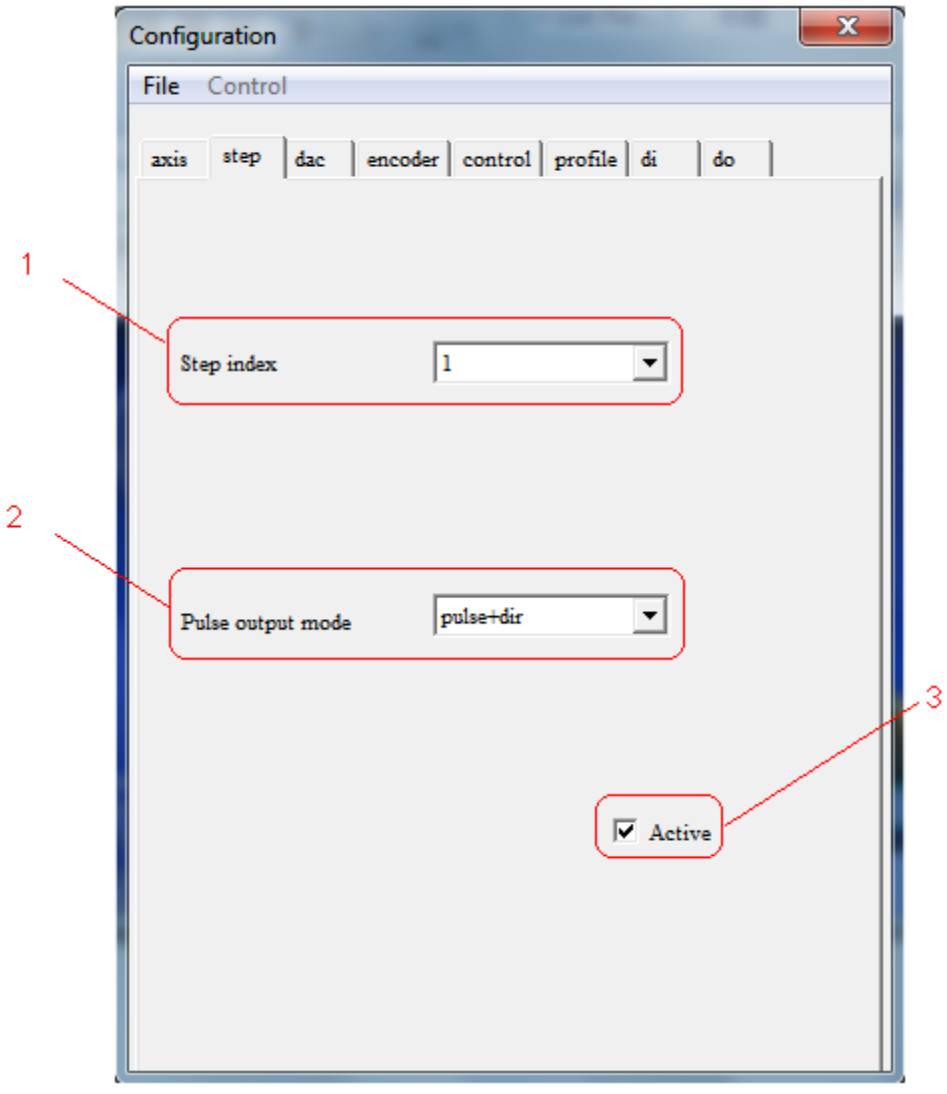

Fig 3-5 Step configuration panel

1 Step index: select index of the step which will be configurated.

2 Pulse output mode: step can realize two types of pulse output mode: "pulse+direction" and "CW/CCW". By default case, the pulse output mode is "pulse+direction". User can use GT_StepDir() command to set step to be "pulse+direction" mode. User can also use GT_StepPulse() command to set step to be "CW/CCW" mode.

3 Active: If step is not active, the output of step pulse is disabled. By default, all steps are active. If some steps are not used, inactivate these steps will save resources of motion controller.

3.2.3 Dac configuration

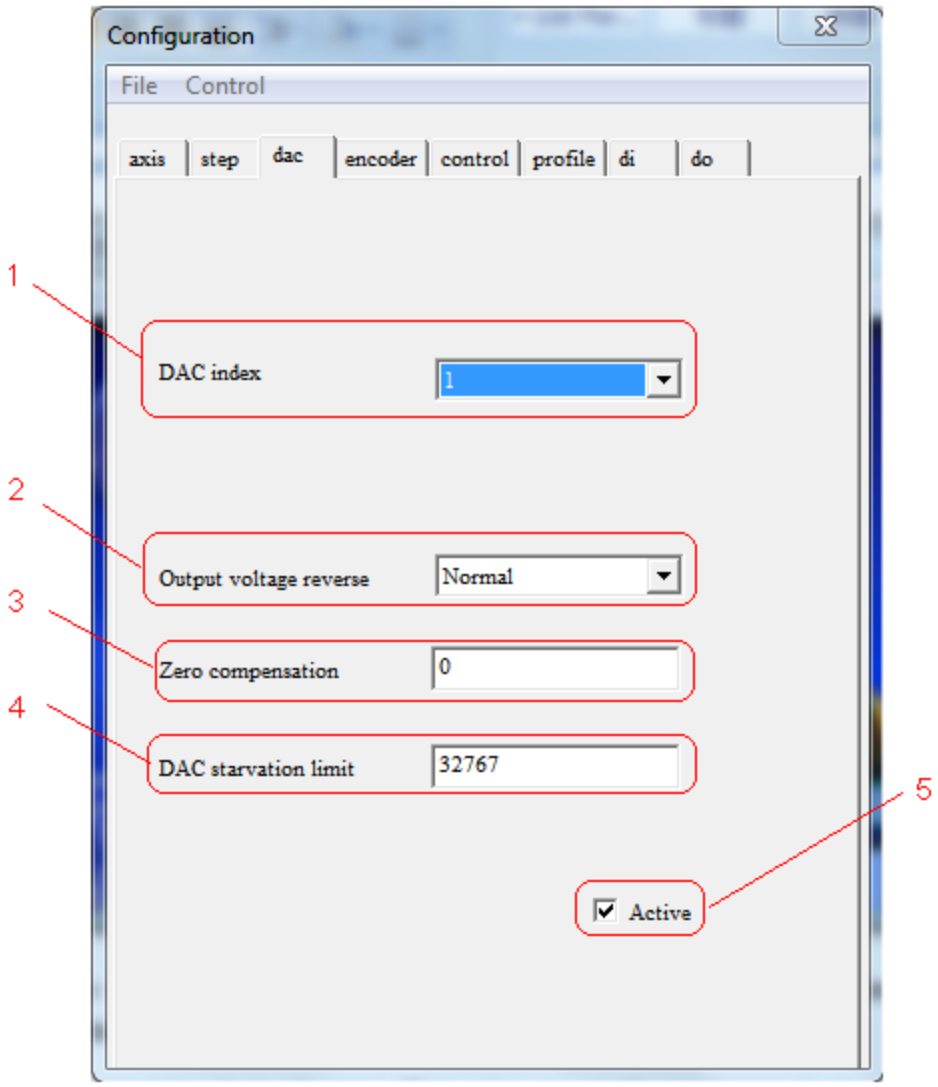

Fig 3-6 Dac configuration panel

1 Dac index: Select index of the Dac which will be configurated.

2 Output voltage reverse: This item is used to reverse Dac output voltage. If user set this option as “Normal”, dac will output positive voltage when the input voltage is positive, and it will output negative voltage when it is negative. If user set the item as “Reverse”, dac will output negative voltage if the input voltage is positive, and positive voltage if it is negative.

3 Zero compensation: User can set the value of zero compensation if needed. This item can be configurated by GT_SetMtrBias() command.

4 Dac saturation limit: This item is used to set the absolute value of maximum dac output voltage. If user set it as 32767, the allowed voltage value is -10V to +10V. If the value is 16384, the allowed voltage value is -5V to +5V. If the absolute value of control voltage output or the absolute value of voltage output set by GT_SetDac() is over this saturation limit, motion controller will output this value. This item can be set by GT_SetMtrLmt() command.

5 Active: If the dac is not active, the voltage output of dac is invalid. By default, all dacs are active. If some dacs are not used , inactivate these dacs will save resources of motion controller.

3.2.4 Encoder configuration

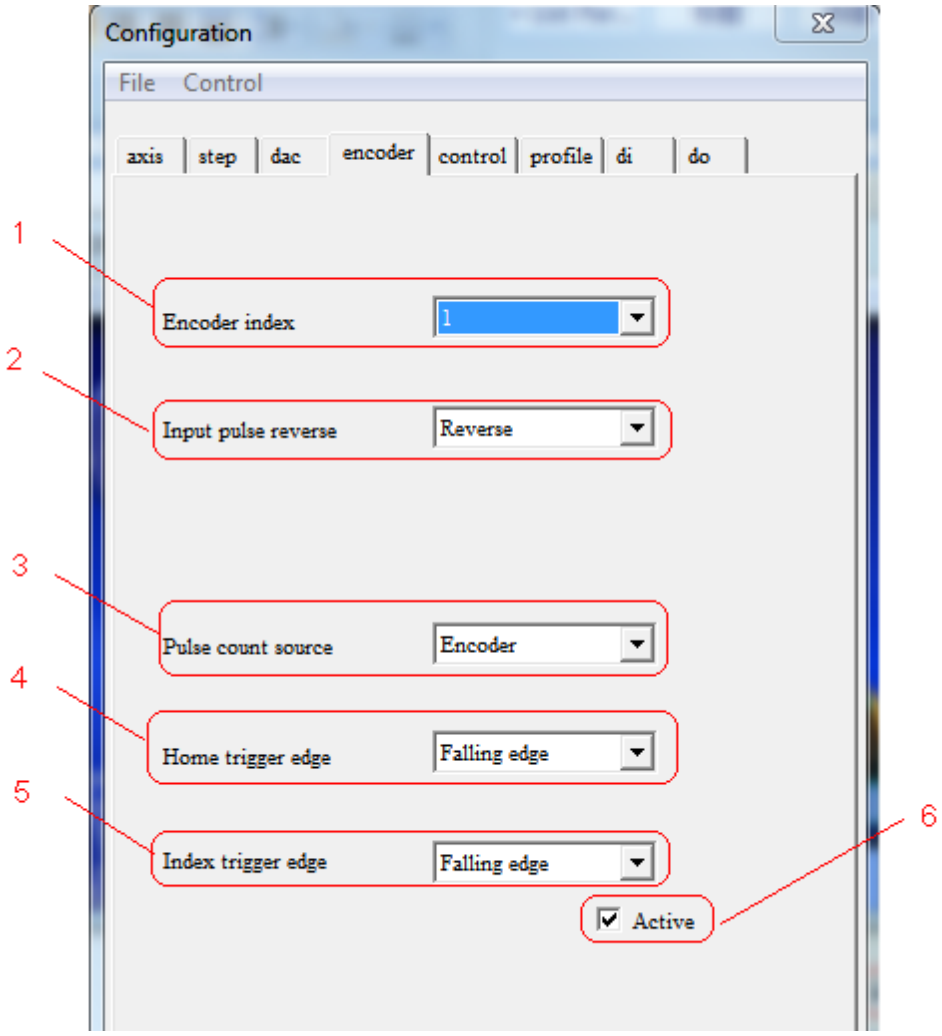

Fig 3-7 Encoder configuration panel

1 Encoder index: Select index of the encoder which will be configurated.

2 Input pulse reverse: Motion controller can receive orthogonal encoder signals. Tab 3-1 describes the relation of this item option, feedback pulse direction and encoder counting direction. This item can be set by calling GT_EncSns() command.

Tab 3-1 The relation of feedback pulse direction and encoder counting direction

	Normal		Reverse	
A Phase				
B Phase				
Encoder	Counting increasing	Counting decreasing	Counting decreasing	Counting increasing

3 Pulse count source: This item represents encoder count source. By default, the source is encoder. If there is no encoder, user can set as pulse counter. In this case, encoder will count the pulse output by step. User can call GT_EncOn() command to set the item as encoder; call GT_EncOff() command to set the item as pulse counter.

4 Home trigger edge: User can change this item to set the trigger edge of home capture. By default, it is triggered by falling edge. If user chooses a permanently closed switch, it should be configurated as rising edge. This item can be modified by calling GT_SetCaptureSense() command.

5 Index trigger edge: User can change this item to set the trigger edge of index capture. By default, it is triggered by falling edge. This item can be modified by calling GT_SetCaptureSense() command.

6 Active: If encoder is not active, the counting of input pulse is invalid. By default, all encoders are active. If some encoders are unused, inactivate these encoders will save resources of motion controller.

3.2.5 Control configuration

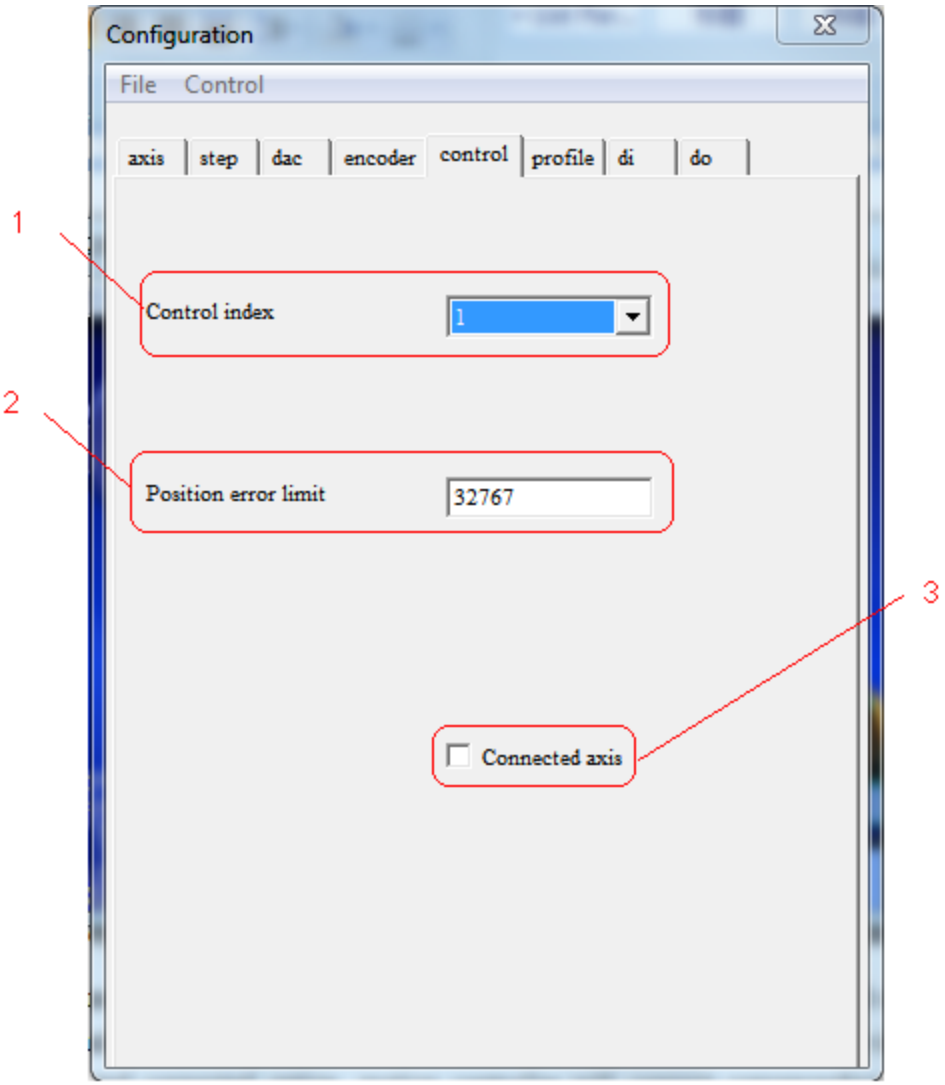

Fig 3-8 Control configuration panel

1 Control index：Select index of the control which will be configurated.

2 Following error limit: This parameter represents the limit error between profile position and encoder position. If actual following error limit exceed set value, motion controller will automatically close the corresponding axis. The default value is 32767, and its unit is pulse. This item can be set by calling GT_SetPosErr() command.

3 Related axis: If user uses servo motor in close-loop control, connected axis should be clicked. By default: the connected axis is unclicked, i.e. open-loop control (pulse control) mode. If “related axis” be clicked, motion controller will connect corresponding encoder, dac, axis and control together. As illustrated in Fig.3-2, dac cannot be set independently. And GT_SetDac() is invalid. GT_CtrlMode() command can switch between close-loop mode and open-loop mode.

3.2.6 Profile configuration

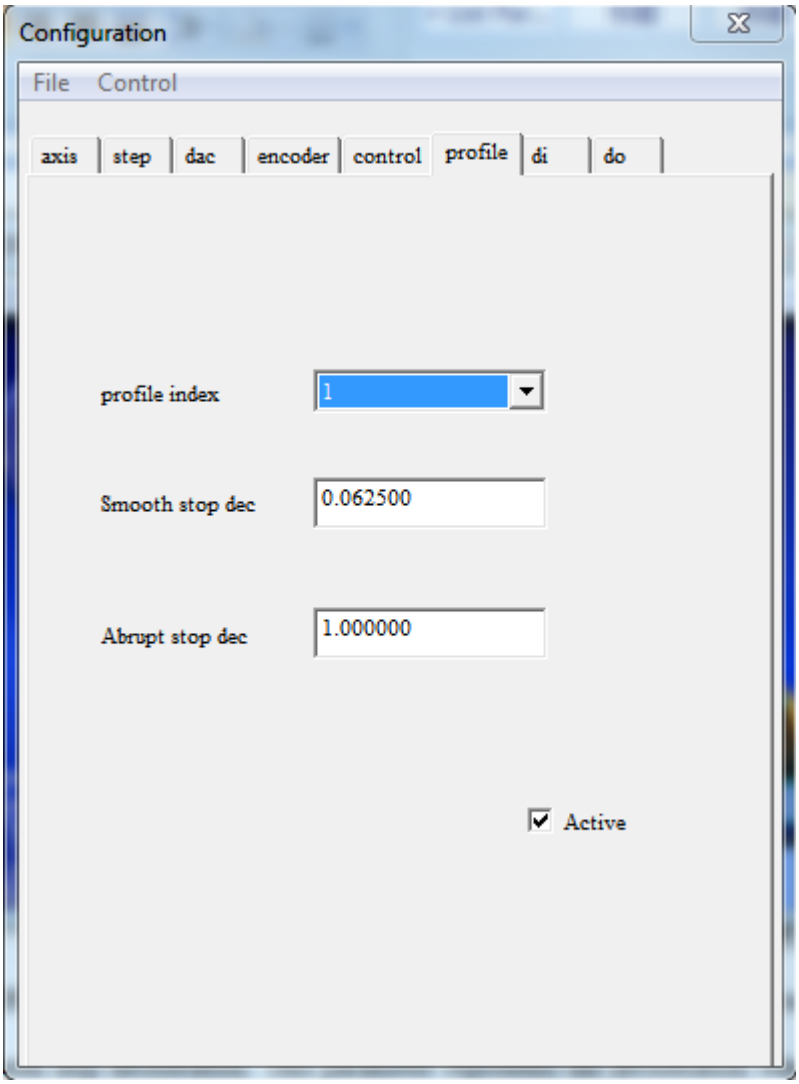

Fig 3-9 Profile configuration panel

1 Profile index: Select index of the profile which will be configurated.

2 Smooth stop dec: This parameter represents the deceleration when GT_Stop is called in smooth stop mode. The default value is 0.0625 pulse/ms2.This value can be modified by calling GT_SetStopDec() command.

3 Emergency stop dec: This parameter represents the deceleration when GT_Stop is called in emergency stop mode. The default value is 1 pulse/ms2. This value can be modified by calling GT_SetStopDec() command.

4 Active: If profile is not active, motion profile of motion controller is invalid. By default, all profiles are active. If some profile are unused, inactivate these profiles will save resources of motion controller.

3.2.7 Di configuration

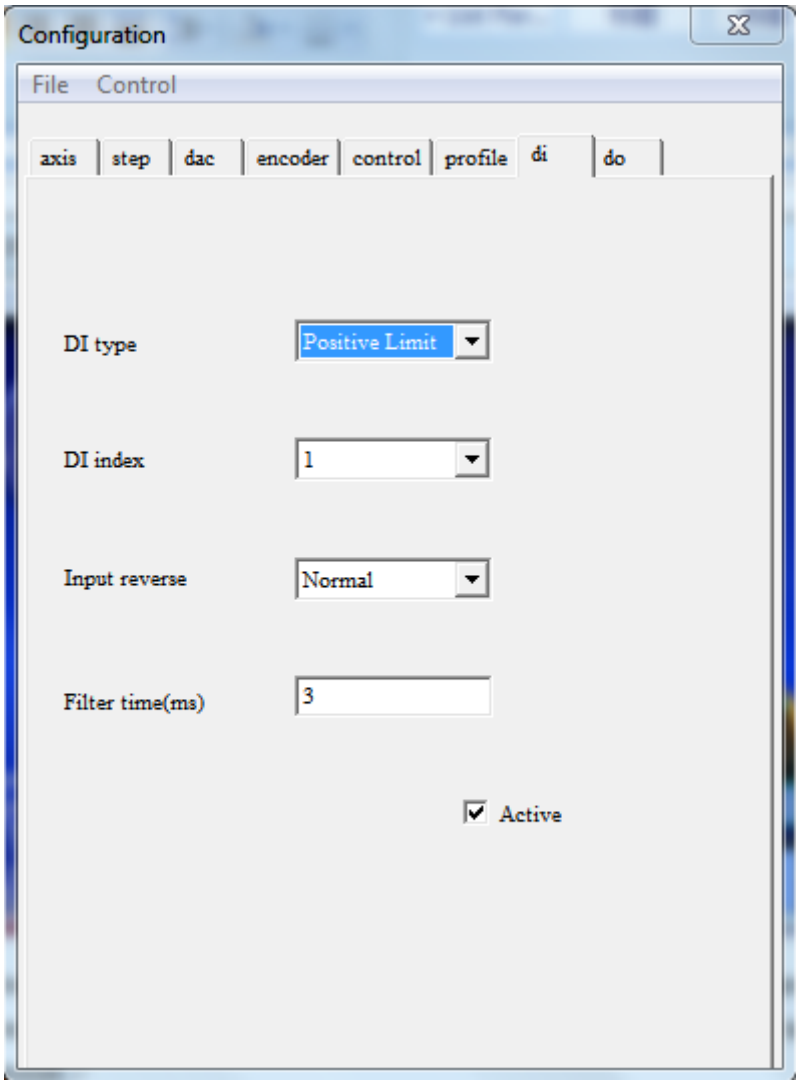

Fig 3-10 Di configuration panel

1 Di type: Select the type of di (digital input) which includes drive alarm, positive limit, negative limit, home and general inputs.

2 Di index: Select the index of di which will be configurated.

3 Input reverse: Used to reverse the input logic of di. By default, “0” represents input low level, and “1” represents input high level. If the item is “reverse”, the logic of di will been reversed: “1” represents input low level, and “0” represents input high level. It can be set by calling GT_GpiSns() command.

4 Filter time (ms): the di signal is valid when it lasts more than Filter time. The default filter time is 3, and its unit is 250 ms.

5 Active: If the di is inactive, the digital inputs of motion controller are invalid. By default, all “di” are active. If user does not need some “di”, inactivate these dis will save the resources of motion controller.

3.2.8 Do configuration

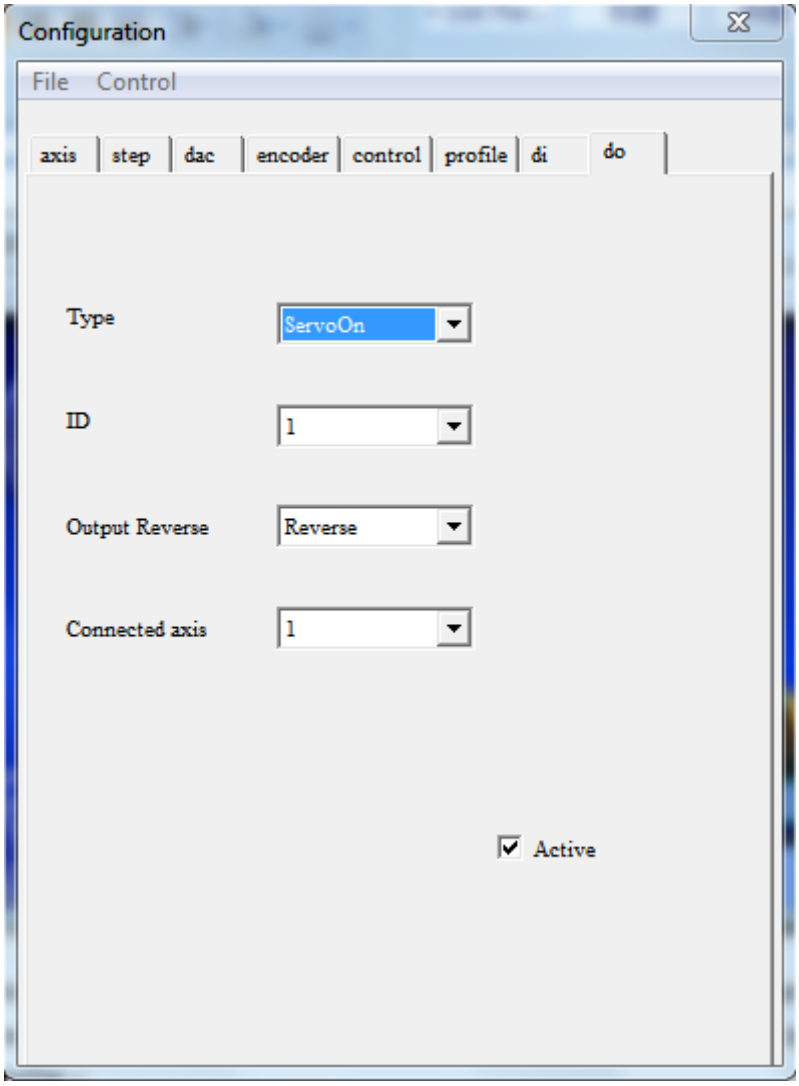

Fig 3-11 Do configuration panel

1 Do type: Select the type of do (digital output), including servo on, clear alarm output and general outputs.

2 Do index: Select the index of do which will be configurated.

3 Output reverse: Used to reverse the output logic of do. By default, “0” represents output low level, and “1” represents output high level. If the item is “reverse”, the logic of do will be reversed: “1” represents output low level, and “0” represents output high level.

4 Related axis: This item represents that the “do” be attached to servo on of specified axis. By default, each axis has its individual servo on output. If GT_AxisOn() is called, the value of do will been set as “1”. If the drive servo on is low level active, “output reverse” has to be set as reverse. Once do is attached to one axis, user can not call GT_SetDo() or GT_SetDoBit() to specify drive servo on output level directly. If user does not need drive servo on, the relation between do and axis should be cancelled. After the relation cancelled, the digital output of drive servo on can be a general digital output, and user can call GT_SetDo() or GT_SetDoBit() to

specify its output level directly.

5 Active: If the do is inactive, the digital outputs of motion controller are invalid. By default, all dos are active. If user does not need some dos, inactivate these dos will save the resources of motion controller.

3.3 Generate and download configuration file

Tab 3-2 Summary of download configuration file commands

Command	Description
GT_LoadConfig	Download the configuration files into motion controller

Tab 3-3 Definition of download configuration file commands

GT_LoadConfig(char *pFile)	
pFile	The name of configuration file.

After configuration process of motion control system has been done in the light of 3.2, as illustrated in Fig 3-12, user can save the configuration information. Click “file”->“Write to file”, then MCT2008 will generate a configuration file(*.cfg).

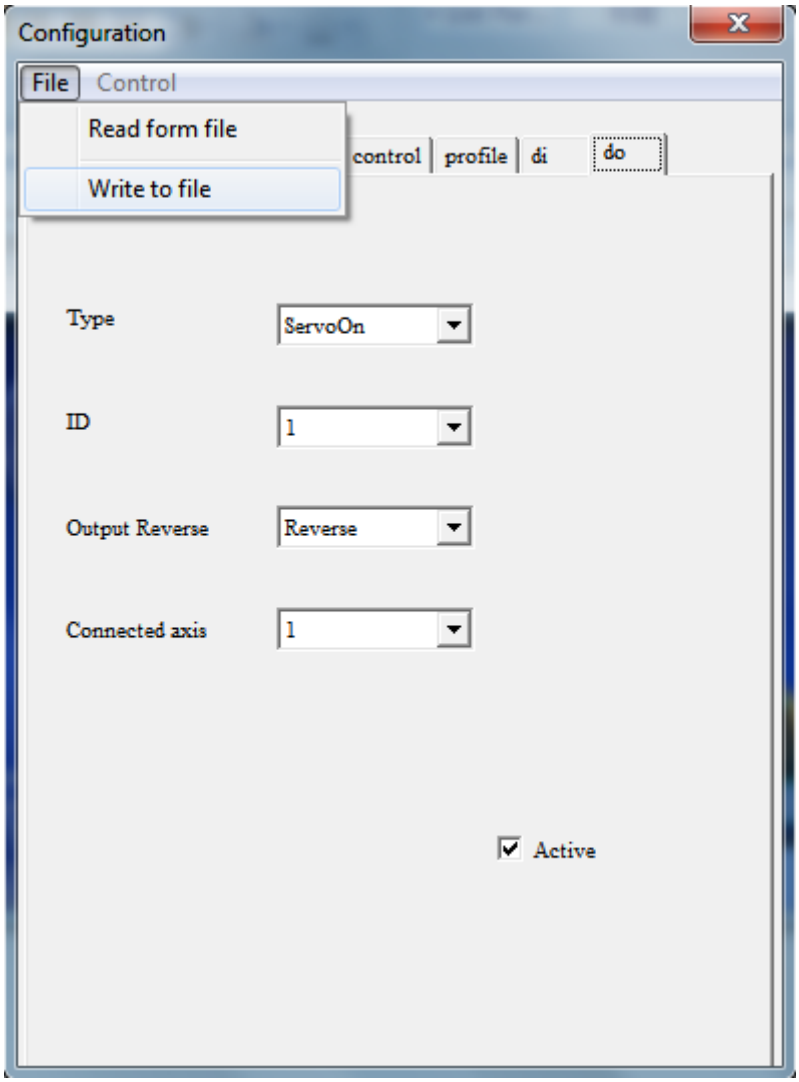

Fig 3-12 Generate configuration file panel

User can download the configuration files to motion controller by calling GT_LoadConfig(). Attention: if configuration files and executable files are not in the same catalogue, the parameter of this command should include the absolute path of configuration files when call GT_LoadConfig().

3.4 Command to modify configuration information

Except configuration file, User can use GT commands to initialization of control system.

3.4.1 Commands summary

Tab 3-4 Summary of configuration commands

Commands	Description
GT_AlarmOff	Disable the drive alarm
GT_AlarmOn	Enable the drive alarm
GT_LmtsOn	Enable the limit
GT_LmtsOff	Disable the limit

GT_ProfileScale	Set the profile scale of axis
GT_EncScale	Set the encoder scale of axis
GT_StepDir	Set step to be "pulse+direction" mode
GT_StepPulse	Set step to be "CW/CCW" mode
GT_SetMtrBias	Set the zero compensation value of dac
GT_GetMtrBias	Get the zero compensation value of dac
GT_SetMtrLmt	Set dac saturation limit
GT_GetMtrLmt	Get dac saturation limit
GT_EncSns	Set encoder count direction
GT_EncOn	Switch to "outside encoder" count mode
GT_EncOff	Switch to "pulse output of step" count mode
GT_SetPosErr	Set following error limit
GT_GetPosErr	Get following error limit
GT_SetStopDec	Set smooth stop decelerability and emergency stop decelerability
GT_GetStopDec	Get smooth stop decelerability and emergency stop decelerability
GT_LmtSns	Specify the effective electrical lever for limit switch
GT_CtrlMode	Set the output mode of specified axis as analog voltage output of pulse output
GT_SetStopIo	Set the input type of smooth stop and emergency stop
GT_GpiSns	Specify the effective electrical lever for digital input
GT_SetAdcFilter	Set the filter time parameter of adc input(for GTS-400-PX only)

Tab 3-5 Definition of configuration commands

GT_AlarmOff(short axis)	
axis	Axis NO.
GT_AlarmOn(short axis)	
axis	Axis NO.
GT_LmtsOn(short axis,short limitType=-1)	
axis	Axis NO.
limitType	Enable limit type MC_LIMIT_POSITIVE(this macro is defined 0): Enable the positive limit of axis MC_LIMIT_NEGATIVE(this macro is defined 1): Enable the negative limit of axis -1: Enable both of positive and negative limit of axis, default value
GT_LmtsOff(short axis,short limitType=-1)	
Axis	Axis NO.
limitType	Disable limit type MC_LIMIT_POSITIVE(this macro is defined 0): Disable the positive limit of axis MC_LIMIT_NEGATIVE(this macro is defined 1): Disable the negative limit of axis -1:Disable both of positive and negative limit of axis, default value
GT_ProfileScale(short axis,short alpha,short beta)	
axis	Axis NO.
alpha	Alpha value of profile scale,range:[-32768,32767], please refer to 3.2.1
beta	Beta value of profile scale,range:[-32768,32767], please refer to 3.2.1

GT_EncScale(short axis,short alpha,short beta)	
axis	Axis NO.
alpha	Alpha value of encoder scale,ranging in [-32768,32767], please refer to 3.2.1
beta	Beta value of encoder scale,ranging in[-32768,32767], please refer to 3.2.1
GT_StepDir(short step)	
step	Step No
GT_StepPulse(short step)	
step	Step No.
GT_SetMtrBias(short dac,short bias)	
dac	Dac No.
bias	Zero compensation value, its value ranging in[-32768,32767]
GT_GetMtrBias(short dac,short *pBias)	
dac	Dac No.
pBias	Zero compensation value
GT_SetMtrLmt(short dac,short limit)	
dac	Dac No.
limit	Set dac saturation limit. Its value ranges in (0,32767].
GT_GetMtrLmt(short dac,short *pLimit)	
dac	Dac No.
pLimit	Get dac saturation limit.
GT_EncSns(unsigned short sense)	
sense	Set encoder count direction by bit, bit0-bit7 corresponding to enconder 1-8, bit8 corresponding to auxiliary encoder. 0: not reverse this encoder count's direction 1: reverse this encoder count's direction please refer to 3.2.4
GT_EncOn(short encoder)	
encoder	Encoder No.
GT_EncOff(short encoder)	
encoder	Encoder No.
GT_SetPosErr(short control,long error)	
control	Control No.
error	Tracking error limit,range: (0,2147483648].
GT_GetPosErr(short control,long *pError)	
control	Control No.
pError	Returned following error limit.
GT_SetStopDec(short profile,double decSmoothStop,double decAbruptStop)	
profile	Profile No.
decSmoothStop	Smooth stop decelerability,range: (0,32767]
decAbruptStop	Emergency stop decelerability,range: (0,32767]
GT_GetStopDec(short profile,double *pDecSmoothStop,double *pDecAbruptStop)	
profile	Profile No.
pDecSmoothStop	Smooth stop decelerability.
pDecAbruptStop	Emergency stop decelerability.

GT_LmtSns(unsigned short sense)	
sense	Set effective electrical level ofr limit switch by bits, please refer to highlights for details.
GT_CtrlMode(short axis,short mode)	
axis	Axis NO.
mode	Output mode 0: analog voltage output mode 1: pulse output mode
GT_SetStoplo(short axis,short stopType,short inputType,short inputIndex)	
axis	Axis No, range of value: [1, 8].
stopType	Stop mode 0：emergency stop 1：smooth stop
inputType	Digital input MC_LIMIT_POSITIVE(this macro is defined 0) positive limit MC_LIMIT_NEGATIVE(this macro is defined 1) negative limit MC_ALARM(this macro is defined 2) drive alarm MC_HOME(this macro is defined 3) home MC_GPI(this macro is defined 4) general input MC_ARRIVE(this macro is defined 5) motor arrive(GTS-400-PX only)
inputIndex	Index of digital input, its range depends on inputType. If inputType= MC_LIMIT_POSITIVE its value ranging in [1, 8]; If inputType= MC_LIMIT_NEGATIVE its value ranging in [1, 8]; If inputType= MC_ALARM its value ranging in [1, 8]; If inputType= MC_HOME its value ranging in [1, 8]; If inputType= MC_GPI its value ranging in [1, 16] If inputType= MC_ARRIVE its value ranging in [1,8]
GT_GpiSns(unsigned short sense)	
sense	Set the digital input level by bit, bit0 to bit15 represents general input 1 to general input 16. 0: original input level, the return of GT_GetDi() is same as the input level, 0 means the input is low level, 1 means the input is high level. 1: reverse input level, the return of GT_GetDi() is same as the reversed value of input level, 0 means the input is high level, 1 means the input is low level.
GT_SetAdcFilter(short adc,short filterTime)	
adc	Adc No. its value ranging in [1,8]
filterTime	Time parameter of the digital input filter,its ranging in[1,50]

3.4.2 Highlights

(1) Set the direction of encoder

GT_EncSns command can modify the counting direction of encoder, if corresponding bit of parameter is set as 1, counting direction of the encoder of corresponding axis is reversed. The definition of status bit of the parameter as show in Tab 3-6.

Tab 3-6 Set direction of encoder

Status bit	8	7	6	5	4	3	2	1	0
encoder	AUX encoder	Enc8	Enc7	Enc6	Enc5	Enc4	Enc3	Enc2	Enc1

(2) Set effective electrical level for limit switch

The default limit switch is normally closed switch. In normal status, the signal of limit switch is at low level, and when a high level input, the switch will be triggered. If normally opened switch is used, user need to call GT_LmtSns() command to change effective electrical level for limit switch.

The parameter of GT_LmtSns() indicates the effective level of the positive/negative limit switch of each axis. When one status bit of the parameter is set as 0, it means that the trigger level of corresponding limit switch is high level active. Whereas, if it is set to 1, it means that the input signal of limit switch is low level active. The corresponding relationship between the status bit of parameter and limit switch is as show in Tab 3-7.

Tab 3-7 Set effective electrical level for limit switch

Status bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Limit	Axis 8		Axis 7		Axis 6		Axis 5		Axis 4		Axis 3		Axis 2		Axis 1	
	—	+	—	+	—	+	—	+	—	+	—	+	—	+	—	+

Chapter 4 New Instruction Descriptions of EtherCAT

This chapter focuses on the new libraries added for GUC-ECAT8/64-M23-L2-F4G controller using EtherCAT bus.The library name is CPAC GUC-X00-TPX ECAT.lib. Other library instruction please refer to previous descriptions. Additionally, EtherCAT instructions is also applicable without using libraries of other software platforms.

4.1 EtherCAT library

4.1.1 Commands summary

Tab 4-1 Summary of EtherCAT commands

Commands	Description
ecat_configure_done	Initialize communications return commands of EtherCAT.
GT_SetEcatGpioConfig	Set GPIO direction and effective level of EtherCAT GUC.
GT_StartEcatHoming	Start axes homing of EtherCAT.
GT_SetHomingMode	Switch axes homing mode of EtherCAT.
GT_GetEcatHomingStatus	Get axes homing status of EtherCAT.
GT_GetEcatProbeStatus	Get EtherCAT axes homing status of probe.

Tab 4-2 Description of EtherCAT commands

ecat_configure_done	
No operand, Return 1: Communication between EtherCAT and motion controller is normally. Others: Communication between EtherCAT and motion controller is not normally.	
GT_SetEcatGpioConfig (effectiveLevel,direction)	
effectiveLevel:INT	Set active level by bit. 0: active low; 1: active high.
direction:INT	Set GPIO to DI or DO by bit. 0: DO, 1: DI.
GT_StartEcatHoming(axis,method,speed1,speed2,acc,offset,probeFunction)	
axis:INT	Axis NO.
method:INT	Set homing mode.
speed1:LREAL	Search the switching speed. Unit: pulse/ms
speed2:LREAL	Search index speed.Unit: pulse/ms
acc:LREAL	Search acceleration。Unit: pulse/ms^2
offset:LREAL	Origin offset.
probefunction:UINT	Probe

GT_SetHomingMode(short axis,short mode)	
axis:INT	Axis NO.
mode:INT	Mode.
GT_GetEcatHomingStatus(axis, phomingStatus)	
axis:INT	Axis NO.
phomeStatus: POINTER TO INT	Return status of homing.
GT_GetEcatProbeStatus (axis, probeStatus,probe1PosValue, probe1NegValue, probe2PosValue, probe2NegValue)	
axis:INT	Axis NO.
probeStatus:POINTER TO INT	Probe status.
probe1PosValue:POINTER TO INT	Positive value of probe1.
probe1NegValue:POINTER TO INT	Negative value of probe1.
probe2PosValue:POINTER TO INT	Positive value of probe2.
probe2NegValue:POINTER TO INT	Negative value of probe2.

4.1.2 Highlights

After initialization of motion controller, the program must call ecat_configure_done firstly.While the return value is 1, the communication between EtherCAT and motion controller is normally, While the return value is 0, the communication between EtherCAT and motion controller is not normally.

The probefunction in GT_StartEcatHoming() is only used in homing mode 35 or 36.Similarly, GT_GetEcatProbeStatus() is also used in in homing mode 35 or 36.

4.1.3 Examples

Example of calling ecat_configure_done is as follows:

```
PROGRAM PLC_PRG
VAR
iConfRtn:INT;
END_VAR
----------------------------------------------------
iConfRtn := ecat_configure_done();
IF iConfRtn <> 1 THEN
RETURN;
END_IF
```

Example of homing mode 3:

```
PROGRAM PLC_PRG
VAR
  bAxisOff: BOOL;
   iHomeSts: INT;
  iCase: INT;
  iAxisNum: INT;
END_VAR
----------------------------------------------------------
```

```
    CASE  iCase THEN
1: (*Start homing*)
IF bAxisOff THEN (*Must be in the under servo status*)
GT_SetHomingMode(iAxisNum, 6); (*Switch to homing mode*)
GT_StartEcatHoming(iAxisNum, 3, 5000, 3000, 100000, 0, 0); (*Start homing *)
iCase := 2;
END_IF
2: (*Check homing status*)
GT_GetEcatHomingStatus(iAxisNum, ADR(iHomeSts));
IF iHomeSts = 3 THEN (*Complete homing *)
GT_SetHomingMode(iAxisNum, 8); (*Switch back to position control mode, and can perform
other control *)
END_IF
END_CASE
```

4.2 Other commands of EtherCAT

When using EtherCAT controller in the platform except CPAC, users can call the following commands and the commands in 0 except ecat_configure_done().

4.2.1 Commands summary

Tab 4-3 Summary of EtherCAT other commands

Commands	Description
GT_InitEcatComm	Initialize EtherCAT.
GT_TerminateEcatComm	Finish EtherCATcommunication.
GT_GetEcatEncPos	Get encoder position value。

Tab 4-4 Description of EtherCAT other commands

GT_InitEcatComm()	
No operand. EtherCAT initialization, scan slaves, and other operations.	
GT_TerminateEcatComm()	
No operand. Finish EtherCATcommunication.	
GT_GetEcatEncPos (axis, *pEncPos)	
axis:short	Axis NO.
phomeStatus: long	Encoder position.

Chapter 5 Motion Mode

Each axis of GUC-ECAT8/64-M23-L2-F4G controller can works independently in Point to Point motion mode, Jog motion mode, PT motion mode, electronic gear motion mode, or follow motion mode. About hardware configuration, please refer to documents about EtherCAT congfiguration file description and instructions for use of EtherCATconfiguration tools.

Tab 5-1 Summary of motion mode commands

Commands	Description
GT_PrfTrap	Set specified axis as Point to Point mode
GT_PrfJog	Set specified axis as Jog mode
GT_PrfPt	Set specified axis as PT mode
GT_PrfGear	Set specified axis as Electronic gear mode
GT_PrfFollow	Set specified axis as Following mode
GT_GetPrfMode	Get motion mode of specified axis

Make sure current axis in static when configurate or change the motion mode. If the axis is in motion , call GT_Stop() to stop one or more axes.

5.1 Point to Point motion mode

5.1.1 Commands summary

Tab 5-2 Summary of Point to Point mode commands

Commands	Description
GT_PrfTrap	Set specified axis as Point to Point mode
GT_SetTrapPrm	Set parameters of Point to Point mode
GT_GetTrapPrm	Get parameters of Point to Point mode
GT_SetPos	Set target position
GT_GetPos	Get target position
GT_SetVel	Set target velocity
GT_GetVel	Get target velocity
GT_Update	Start motion of Point to Point mode

Tab 5-3 Description of Point to Point mode commands

GT_PrfTrap(profile)	
Profile:INT	Profile No.
GT_SetTrapPrm(profile,pPrm)	
Profile:INT	Profile No.
PPrm:POINTER TO TTrapPrm	Parameters of Point to Point mode typedef struct TrapPrm STRUCT TTrapPrm

	Acc:LREAL; // acceleration, unit: pulse/ms^2"
	Dec:LREAL; // deceleration, unit: pulse/ms^2
	VelStart:LREAL; // start velocity, unit "pulse/ms"
	SmoothTime:INT; // smooth time, unit: ms

GT_GetTrapPrm(profile,pPrm)	
Profile:INT	Profile No.
PPrm: POINTER TO TTrapPrm	Get parameters of Point to Point mode.
GT_SetPos(profile,pos)	
Profile:INT	Profile No.
Pos:DINT	Target postion, unit: pulse
GT_GetPos(profile, pPos)	
Profile:INT	Profile No.
PPos: POINTER TO DINT	Get target postion, unit: pulse
GT_SetVel(profile, vel)	
Profile:INT	Profile No.
Vel:LREAL	Target velocity, unit: pulse/ms
GT_GetVel(profile,pVel)	
Profile:INT	Profile No.
PVel: POINTER TO LREAL	Get target velocity, unit: pulse/ms
GT_Update(mask)	
Mask:DINT	"Mask" represents specified axis No., which will be started by bit. Bit 0 represents axis1, bit 1 represents axis 2, and so on. When the bit X =1, the corresponding axis will be started.

5.1.2 Highlights

In Point to Point mode, user can set parameters of each axis independently, such as the target position, target velocity, acceleration, deceleration, start velocity and smooth time etc., and each axis can be started and stopped independently by user.

After calling GT_Update() to start up Point to Point motion, motion controller automatically generates relevant trapezoidal velocity profile according to motion parameters set by user. In Point to Point mode, the target velocity and position can be modified whenever the user needed.

Setting smooth time, motion controller can get a smooth velocity curve, so the acceleration and deceleration process will become smoothly, as illustrated in Fig 5-1.

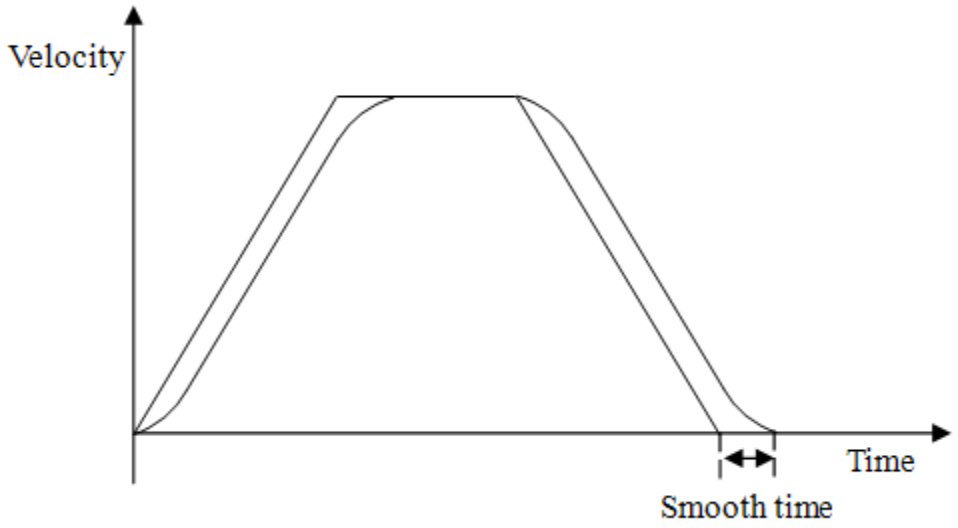

Fig 5-1 Velocity curve of Point to Point motion mode

“Smooth time” is adjustive time of acceleration and deceleration (unit: ms), its range is [0, 50].

5.1.3 Example

This example generates a trapezoidal velocity profile, illustrated in Fig 5-2.

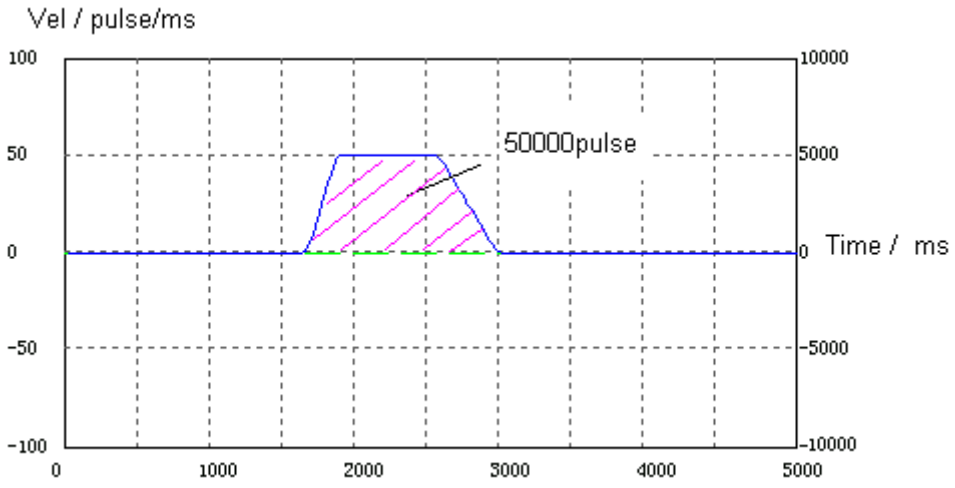

Fig 5-2 Profile velocity of Point to Point motion mode

```
0001 PROGRAM PLC_PRG
0002 VAR
0003     rtn: INT;
0004     sts: DWORD;
0005     i: SINT;
0006     Prm: TTrapPrm;
0007     SmoothTime: ARRAY[1..2] OF INT:= 0,100;
0008
0009     Start: BOOL := FALSE;
0010     VelX: LREAL;
0011     VelY: LREAL;
0012     Stop: BOOL := FALSE;
0013     Reset: BOOL := FALSE;
0014     done: BOOL:= FALSE;
0015
0016     startFlag: BOOL := FALSE;
0017     temp: LREAL;
0018     tempPos: DINT;
0019 END_VAR
```

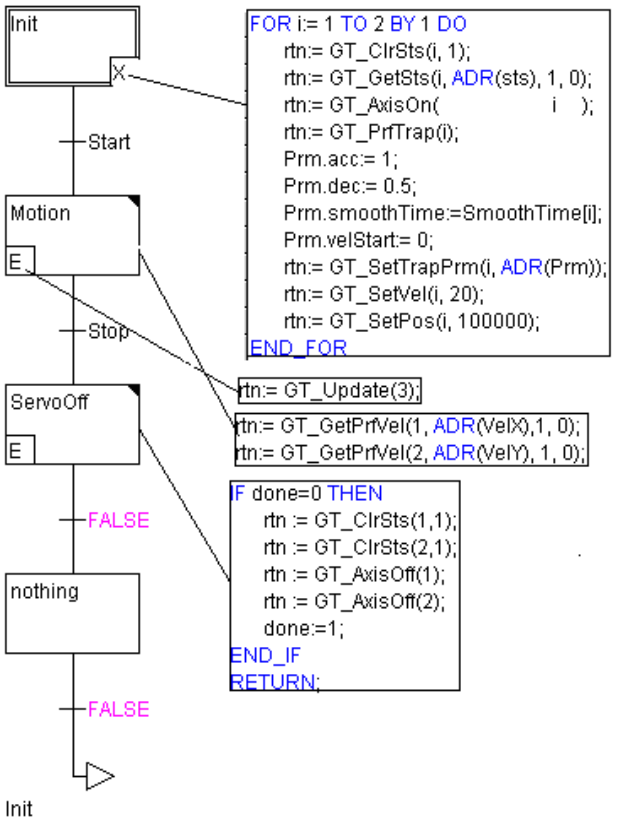

Fig 5-3 Example of point to point motion mode

5.2 Jog Motion Mode

5.2.1 Commands summary

Tab 5-4 Summary of Jog motion mode commands

Commands	Description
GT_PrfJog	Set specified axis as Jog mode
GT_SetJogPrm	Set parameters of Jog mode
GT_GetJogPrm	Get parameters of Jog mode
GT_SetVel	Set target velocity, unit: pulse/ms
GT_GetVel	Get target velocity, unit: pulse/ms
GT_Update	Start motion of Jog mode

Tab 5-5 Description of Jog mode commands

GT_PrfJog(profile)	
Profile:INT	Profile No.
GT_SetJogPrm(profile,TJogPrm *pPrm)	
Profile:INT	Profile No.

PPrm: POINTER TO TJogPrm	Parameters of Jog mode.
	STRUCT TJogPrm
	Acc:LREAL; // acceleration, unit: pulse/ms^2
	Dec:LREAL; // deceleration, unit: pulse/ms^2
	Smooth:LREAL; // smooth coefficient, Ranging in [0,1).

GT_GetJogPrm(profile, pPrm)	
Profile:INT	Profile No.
pPrm: POINTER TO TJogPrm	Get parameters of Jog mode.

5.2.2 Highlights

In Jog motion mode, user can set several parameters of each axis independently, such as the target velocity, acceleration, deceleration and smooth coefficient etc. At the same time, each axis can be started and stopped independently by calling related commands.

After calling GT_Update() to start Jog motion, the velocity increases to the target velocity according to the acceleration. Then the velocity will be kept until a new target velocity command received. The target velocity can be modified in motion as illustrated in Fig 5-4.

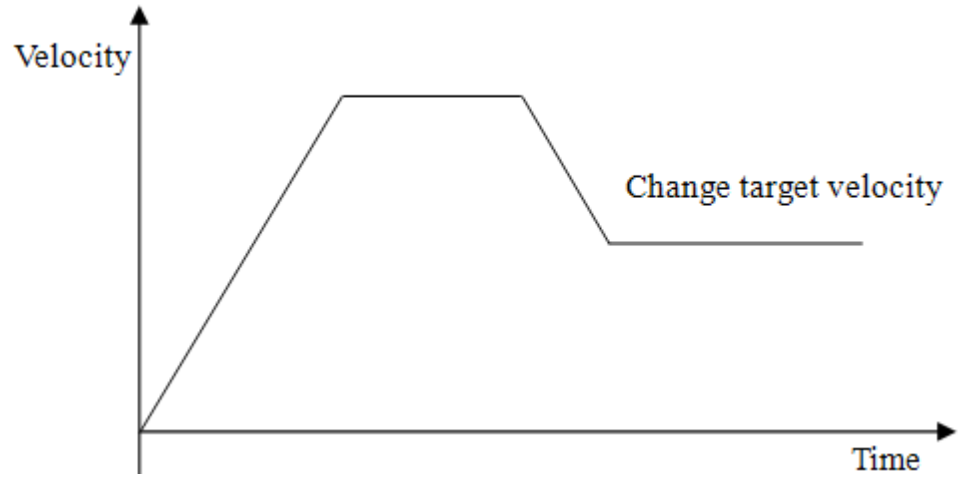

Fig 5-4 Velocity curve of Jog mode

User can set the smooth coefficient to get a smooth velocity curve. Consequently, the acceleration and deceleration process of specified axis will become smoother. The ranging in of smooth coefficient is [0，1). The larger the smooth coefficient is, the smoother the change of acceleration is.

5.2.3 Example

This example changes target velocity in motion. As illustrated in Fig 5-5.

```
0001 PROGRAM PLC_PRG
0002 VAR
0003     Enable: BOOL := TRUE;
0004     rtn: INT;
0005     AXIS_X: INT:= 1;
0006     Sts: DWORD;
0007     JogPrm: TJogPrm;
0008     PrfVel: LREAL;
0009     PrfPos: LREAL;
0010     trans: BOOL;
0011     STOP: BOOL := FALSE;
0012     Done: BOOL := TRUE;
0013 END_VAR
```

```
IF Enable THEN
    (* Servo on*)
    rtn:= GT_ClrSts(AXIS_X, 1);
    rtn:= GT_GetSts(AXIS_X, ADR(Sts), 1, 0);
    rtn:= GT_AxisOn(AXIS_X);
    (* Set X-axis as JOG mode*)
    rtn:= GT_PrfJog(AXIS_X);
    JogPrm.acc:= 0.5;
    JogPrm.dec:= 0.5;
    JogPrm.smooth:= 0.8;
    rtn:= GT_SetJogPrm(AXIS_X, ADR(JogPrm));
    (*Start Jog motion mode*)
    rtn:= GT_SetVel(AXIS_X, 50);
    rtn:= GT_Update(SHL(DWORD#1, AXIS_X-1));
    Enable:= FALSE;
    trans:= TRUE;
END_IF
rtn:= GT_GetSts(AXIS_X, ADR(Sts), 1, 0);
rtn:= GT_GetPrfVel(AXIS_X, ADR(PrfVel), 1, 0);
rtn:= GT_GetPrfPos(AXIS_X, ADR(PrfPos), 1, 0);
IF PrfPos>=100000 AND trans=TRUE THEN
    (*Change the target velocity*)
    rtn:= GT_SetVel(AXIS_X, 25);
    rtn:= GT_Update(SHL(DWORD#1, AXIS_X-1));
    trans:= FALSE;
END_IF

IF STOP=TRUE AND Done THEN
    rtn:= GT_ClrSts(AXIS_X, 1);
    rtn:= GT_GetSts(AXIS_X, ADR(Sts), 1, 0);
    rtn:= GT_AxisOff(AXIS_X);
    Done:= FALSE;
END_IF
```

Fig 5-5 Example of Jog motion mode

5.3 PT Motion Mode

5.3.1 Commands summary

Tab 5-6 Summary of PT Motion Mode commands

Commands	Description
GT_PrfPt	Set specified axis as PT mode
GT_PtSpace	Get the free space of FIFO
GT_PtData	Add data to FIFO
GT_PtClear	Clear FIFO data This command is invalid when the axis is motion or the FIFO memory is in dynamic mode.
GT_SetPtLoop	Set the cycle numbers. This command is invalid, when the FIFO memory is in dynamic mode.
GT_GetPtLoop	Get the cycle numbers This command is invalid, when the FIFO memory is in dynamic mode.
GT_PtStart	Start PT mode motion

Tab 5-7 Description of PT Motion Mode commands

GT_PrfPt(profile,mode)	
Profile:INT	Profile No.
Mode:INT	Set the FIFO mode of PT mode. “PT_MODE_STATIC” is the static mode(default mode) of FIFO; “PT_MODE_DYNAMIC” is the dynamic mode of FIFO
GT_PtSpace(profile,pSpace,fifo)	
Profile:INT	Profile No.
PSpace:POINTER TO INT	Free space of FIFO memory.
Fifo:INT	The FIFO which will be accessed. The default value is 0. In dynamic mode, this parameter is invalid.
GT_PtData(profile,pos,time,type,fifo)	
Profile:INT	Profile No.
Pos:LREAL	Ending position of segment, unit: pulse
Ttime:DINT	Ending time of segment, unti:pulse/ms
Type:INT	Segments type “PT_SEGMENT_NORMAL” represents the normal segment (default type); “PT_SEGMENT_EVEN” represents constant speed segments; “PT_SEGMENT_STOP” represents the segment with ending velocity as zero.
Fifo:INT	FIFO will store motion data. The default value is 0. In dynamic mode, this parameter is invalid.
GT_PtClear(profile,fifo)	
Profile:INT	Profile No.

Fifo:INT	The FIFO which will be cleared off. The default value is 0. In dynamic mode, this parameter is invalid.
GT_SetPtLoop(profile,loop)	
Profile:INT	Profile No.
Loop:DINT	Repeat times of PT mode. In dynamic mode, this parameter is invalid.
GT_GetPtLoop(profile,pLoop)	
Profile:INT	Profile No.
PLoop:POINTER TO DINT	Cycle numbers of PT mode. In dynamic mode, this parameter is invalid.
GT_PtStart(mask,option)	
Mask:DINT	"Mask" represents specified axis No., which will be started by bit. Bit 0 represents axis1, bit 1 represents axis 2, and so on. Start the specified axis when the bit X is 1.
Option:DINT	"Option" represents FIFO No. which will be used by axis in bit. The default value is 0. Bit 0 represents axis1, bit 1 represents axis 2, and so on. When bit X=0, it means the corresponding axis will use the FIFO1. When bit X=1, it means the corresponding axis will use the FIFO2. In dynamic mode, this parameter is invalid.

5.3.2 Highlights

The PT mode is very flexible in achieving any velocity profile. The motion controller will describe the movement as long as user provides the parameters of position and time.

PT mode uses a series of position and time to describe velocity profile. User need to divide the velocity curve into lots of segments, refer to Fig 5-6.

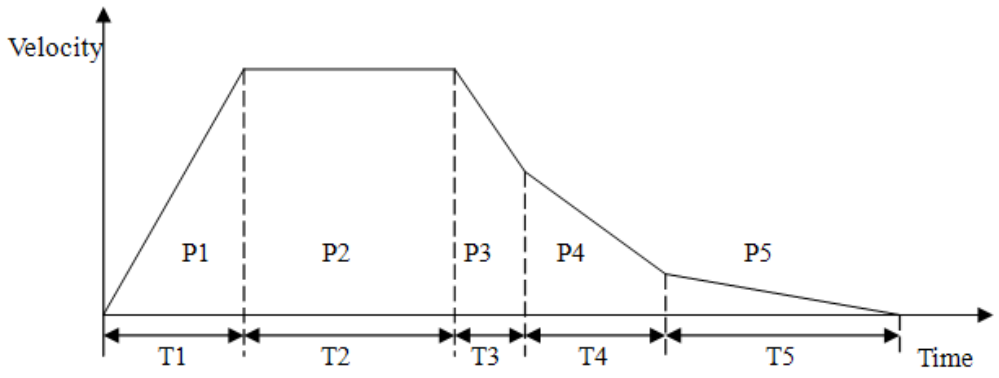

Fig 5-6 Velocity curve of PT mode

In Fig 5-6, the velocity curve has been divided into 5 segments. The starting-point velocity of first segment is 0, after time T1 the axis moved distance P1, so the ending-point velocity of first segment is $v1 = \frac{2P1}{T1}$. The starting-point velocity of second segment is $v1$, after time T2 the axis moved with

distance P2, so the ending-point velocity of second segment is $v2 = \frac{2P2}{T2} - v1$. Similarly, the third segment and the forth segment and all subsequent segments can be calculated.

The motion controller will calculate the velocity and position of each point in each segment and generate a sequential velocity curve, as long as user provides the motion time and motion distance of each segment. In order to get a smooth velocity curve, user can increase the number of segments.

During a whole PT motion, we assume position and time of starting-point in the first segment is 0, the position and time of ending-point of each segment are absolute value compared to starting-point of first segment. The unit of position is pulse and the unit of time is ms.

(1) Segmented data types

There are three types of segmented data of PT mode.

PT_SEGMENT_NORMAL represents normal segments. In FIFO memory, the velocity of starting-point of first segment is 0. And from second segment of FIFO memory, the starting-velocity of subsequent segments is equal to ending-velocity of last segment.

PT_SEGMENT_EVEN represents constant speed segments. In FIFO memory, the velocity of each segment keeps invariant. The velocity of segment is result of motion displacement of segment divide motion time of segment, refer to Fig 5-7.

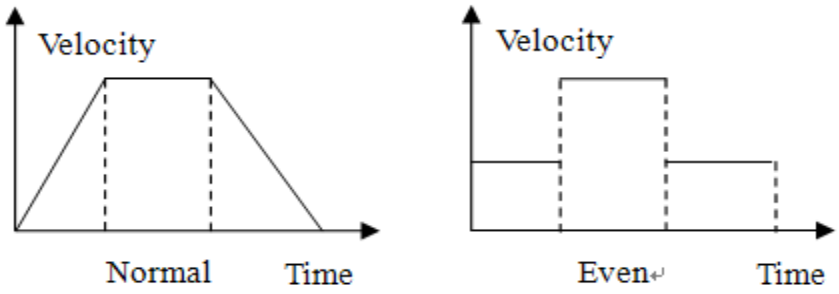

Fig 5-7 Constant speed segments of PT mode

PT_SEGMENT_STOP represents stop segments. The velocity of ending-point is 0. The starting-velocity can be calculated by motion displacement and time of segment and it is independent to last ending-velocity, refer to Fig 5-8.

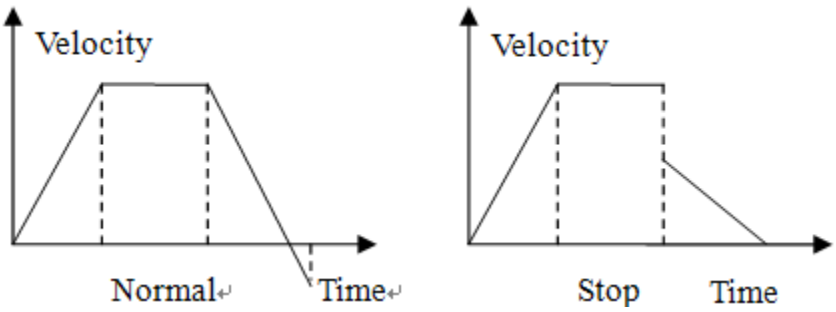

Fig 5-8 Stop segment of PT mode

(2) PT motion mode

PT mode can use FIFO in two types of mode: Static mode and dynamic mode.

There are two FIFO to store the data in PT mode: FIFO1, FIFO2.

In static mode, controller starts one of FIFO and stop it after motion profile is stop. Controller will not clear the data of FIFO, so user can use the data of this FIFO repeatedly. In static mode, user can clear the data of specified FIFO by calling GT_PtClear(). When the axis is moving, user can not clear the data of FIFO in use, but user can clear the data of FIFO which is not in use.

In dynamic mode, when the data in one of the FIFO has been finished, controller will automatically clear off all the data in this FIFO and switch to another FIFO. At the same time, user can send new data to the controller. When all the data of two FIFO have been finished, the motion will stop. In order to avoid abnormal stop, user need to send new data to FIFO before all the data in FIFO have been executed. User can call GT_PtSpace() to check how many empty space the FIFO memory have.

User can set the FIFO in static mode or dynamic mode when the controller switches to PT mode. In PT motion mode, user can not modify the using mode of FIFO.

5.3.3 Example

(1) Static FIFO in PT mode

This example generate a velocity profile of T-curve, refer to Fig 5-9.

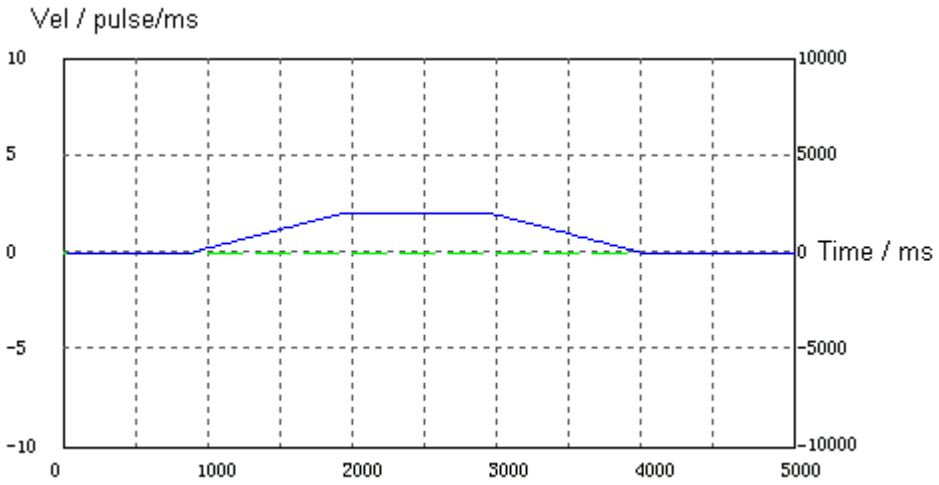

Fig 5-9 Trapezoidal velocity profile of PT mode

```
PROGRAM PLC_PRG

VAR

  Start: BOOL := FALSE;

  Enable: BOOL := TRUE;

  rtn: INT;
```

```
    AXIS_X: INT := 1;
    space: DINT;
    pos: LREAL;
    gtime: DINT;
    Sts: DWORD;
    PrfVel: LREAL;
    PrfPos: LREAL;
    Stop: BOOL := FALSE;
    Done: BOOL := TRUE;
END_VAR
-----------------------------------------------------
IF Start AND Enable THEN
    (*Servo on*)
    rtn:= GT_ClrSts(AXIS_X, 1);
    rtn:= GT_AxisOn(AXIS_X);
    (*Specify the motion mode of axis as PT mode*)
    rtn:= GT_PrfPt(AXIS_X, PT_MODE_STATIC);
    rtn:= GT_PtClear(AXIS_X, 0);
    (*Push data to buffer*)
    rtn:= GT_PtSpace(AXIS_X, ADR(space), 0);
    pos:=1024;
    gtime:= 1024;
    IF space>0 THEN
        rtn:= GT_PtData(AXIS_X, pos, gtime, PT_SEGMENT_NORMAL, 0);
    END_IF

    rtn:= GT_PtSpace(AXIS_X, ADR(space), 0);
    pos:=pos+2048;
    gtime:= gtime+1024;
    IF space>0 THEN
        rtn:= GT_PtData(AXIS_X, pos, gtime, PT_SEGMENT_NORMAL, 0);
```

```
        END_IF

        rtn:= GT_PtSpace(AXIS_X, ADR(space), 0);
        pos:=pos+1024;
        gtime:= gtime+1024;
        IF space>0 THEN
            rtn:= GT_PtData(AXIS_X, pos, gtime, PT_SEGMENT_NORMAL, 0);
        END_IF
        (*Start up PT mode*)
        rtn:= GT_PtStart(SHL(INT#1, AXIS_X-1), 16#00);

        Enable:= FALSE;
    END_IF
    (*Get the status, profile velocity and profile position*)
    rtn:= GT_GetSts(AXIS_X, ADR(Sts), 1, 0);
    rtn:= GT_GetPrfVel(AXIS_X, ADR(PrfVel), 1, 0);
    rtn:= GT_GetPrfPos(AXIS_X, ADR(PrfPos), 1, 0);

    IF Stop AND Done THEN
        (*servo off *)
        rtn:= GT_ClrSts(AXIS_X, 1);
        rtn:= GT_AxisOff(AXIS_X);
        Done:= FALSE;
    END_IF
```

(2) Dynamic type FIFO in PT mode

This example generate a velocity profile of sin-curve, as illustrated in Fig 5-10.

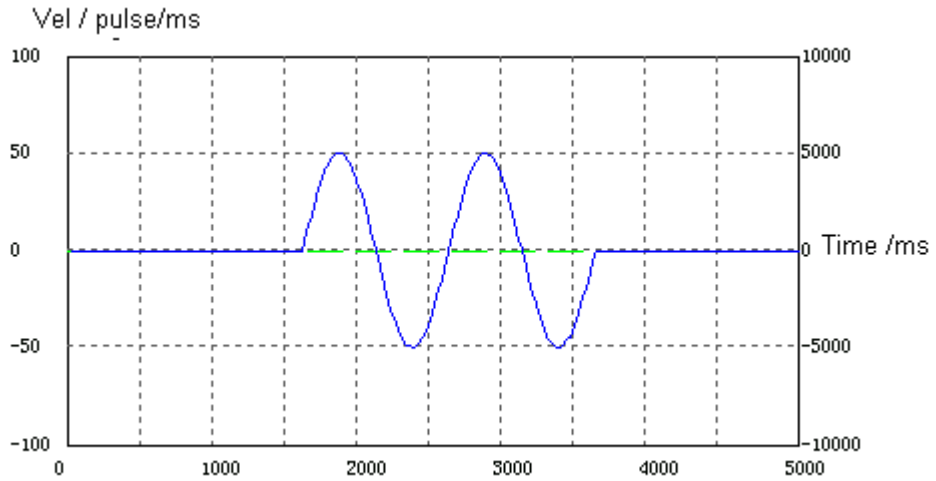

Fig 5-10 Sinusoidal velocity profile of PT mode

```
PROGRAM PLC_PRG

VAR CONSTANT

   T: DINT := 2000;

   DeltaT: DINT:= 2;

   PI: LREAL:= 3.1415926;

END_VAR

VAR

   InitDone: BOOL := FALSE;

   Start: BOOL := FALSE;

   rtn: INT;

   AXIS_X: INT := 4;

   PrfPos: LREAL;

   PrfVel: LREAL;

   Sts: DWORD;

   Pos: LREAL := 0;

   gtime: DINT := 0;

   Vel: LREAL;

   PreVel: LREAL;

   Space: DINT;

   i: INT;

   PTStart: BOOL := FALSE;
```

```
    Stop: BOOL := FALSE;
    StopDone: BOOL := FALSE;
    EncPos: LREAL;
END_VAR
-----------------------------------------------------
IF NOT InitDone AND Start THEN
    (*Servo on *)
    rtn:= GT_ClrSts(AXIS_X, 1);
    rtn:= GT_AxisOn(AXIS_X);
    (*Specify the motion mode of axis as dynamic PT mode *)
    rtn:= GT_PrfPt(AXIS_X, PT_MODE_DYNAMIC);
    rtn:= GT_PtClear(AXIS_X, 0);

    Pos:= 0;
    Vel:= 0;
    gtime:= 0;
    PreVel:= 0;
    InitDone:= TRUE;
END_IF

(*Get the status, profile velocity and profile position *)
rtn:= GT_GetSts(AXIS_X, ADR(Sts), 1, 0);
rtn:= GT_GetPrfVel(AXIS_X, ADR(PrfVel), 1, 0);
rtn:= GT_GetPrfPos(AXIS_X, ADR(PrfPos), 1, 0);

(*Push data to buffer *)
FOR i:=1 TO 5 BY 1 DO
    rtn:= GT_PtSpace(AXIS_X, ADR(Space), 0);
    IF Space>0 THEN
        gtime:= gtime+DeltaT;
        Vel:= 30*SIN(2*PI*gtime/T);
```

```
            Pos:= Pos+ (Vel+PreVel)*DeltaT/2;
            PreVel:= Vel;
            rtn:= GT_PtData(AXIS_X, Pos, gtime, PT_SEGMENT_NORMAL, 0);
        END_IF
    END_FOR

    IF NOT PTStart AND InitDone THEN
        rtn:= GT_PtStart(SHL(INT#1, AXIS_X-1), 0);
        PTStart:= TRUE;
    END_IF

    IF Stop    AND NOT StopDone THEN
        rtn:= GT_Stop(SHL(DINT#1, AXIS_X-1), 0);
        rtn:= GT_ClrSts(AXIS_X, 1);
        rtn:= GT_AxisOff(AXIS_X);
        StopDone:= TRUE;
    END_IF
```

5.4 Electronic gear motion mode

5.4.1 Commands summary

Tab 5-8 Summary of Electronic gear mode commands

Commands	Description
GT_PrfGear	Set specified axis as Electronic gear mode
GT_SetGearMaster	Set master axis
GT_GetGearMaster	Get the information about the master axis
GT_SetGearRatio	Set the electronic gear ratio
GT_GetGearRatio	Get the electronic gear ratio
GT_GearStart	Start Electronic gear mode motion

Tab 5-9 Description of Electronic gear mode commands

GT_PrfGear (profile ,dir)	
Profile:INT	Profile No.
Dir:INT	Following type of slave axis 0 represents the following type is bidirectional follow; 1 represents the following type is positive follow; -1 represents the following type is negative follow.
GT_SetGearMaster(profile , masterIndex, masterType,masterItem)	
Profile:INT	Profile No.
MasterIndex:INT	The index of master axis
MasterType:INT	Master axis type GEAR_MASTER_ENCODER means follow encoder. GEAR_MASTER_PROFILE means follow profile axis. GEAR_MASTER_AXIS means follow synthetic axis.
MasterItem:INT	Synthetic axis type 0: the profile position of synthetic axis. 1: the encoder position of synthetic axis.
GT_GetGearMaster(profile, pMasterIndex, pMasterType,pMasterItem)	
Profile:INT	Profile No.
PMasterIndex:POINTER TO INT	The index of master axis
PMasterType: POINTER TO INT	Master axis type
PMasterItem: POINTER TO INT	Synthetic axis type
GT_SetGearRatio(profile,masterEven,slaveEven,masterSlope)	
Profile:INT	Profile No.
MasterEven:DINT	Electronic gear ratio, the displacement of master axis
SlaveEven:DINT	Electronic gear ratio, the displacement of slave axis.
MasterSlope:DINT	The displacement of clutch zone of master.
GT_GetGearRatio(profile,pMasterEven,pSlaveEven,pMasterSlope)	
Profile:INT	Profile No.

PMasterEven: POINTER TO DINT	The displacement of master axis.
PSlaveEven: POINTER TO DINT	The displacement of slave axis.
PMasterSlope: POINTER TO DINT	The displacement of clutch zone of master.
GT_GearStart(mask)	
Mask:DINT	"Mask" represents specified axis No., which will be started by bit. Bit 0 represents axis1, bit 1 represents axis 2, and so on. When the bit X =1, the corresponding axis will be started.

5.4.2 Highlights

Electronic gear mode can associate two or more axes together to implement the synchronous motion precisely, so this mode can replace the traditional mechanical gear. In electronic gear mode user can set electronic gear ratio flexible, thus saving installation time of mechanical system.

In Electronic gear mode, one master axis can drive more than one slave axes, and slave axis can follow the profile position or encoder position of master axis.

In order to reduce the following delay of slave axis, the NO. of slave axis should be larger than the NO. of master axis.

In Electronic gear mode, controller can modify the electronic gear ratio in motion. User can set clutch zone to change velocity smoothly when the electronic gear ratio is changed.As illustrated in Fig 5-11.

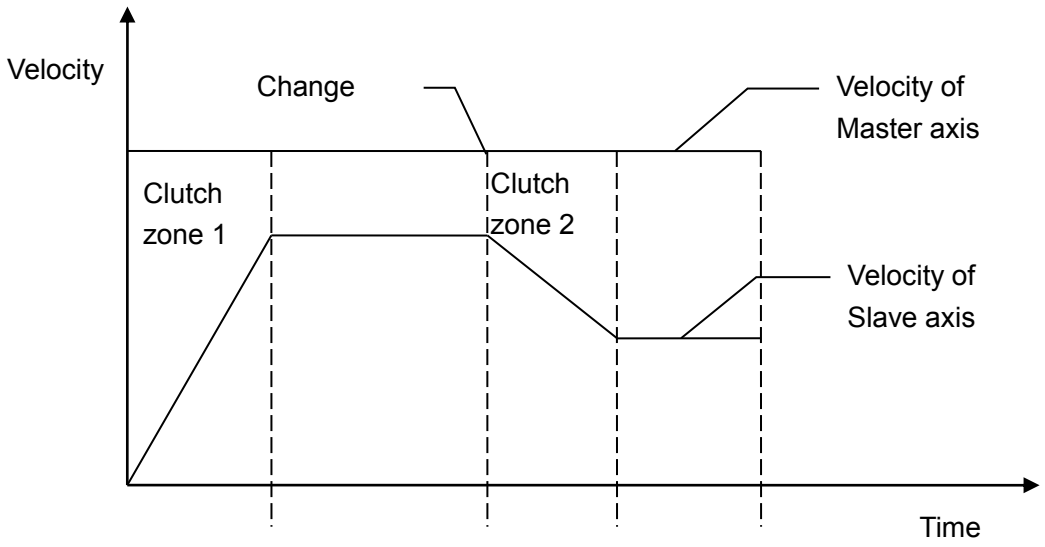

Fig 5-11 Velocity curve of electronic gear mode

When master axis motion at a constant speed, slave axis is in Electronic gear mode. In clutch zone 1, the velocity of slave axis will increase from 0 , until the set electronic gear ratio. When the given electronic gear ratio be modified, the velocity of slave axis will gradually change to achieve the new ratio in clutch zone 2. The lager the clutch zone is, the more smooth changing process of slave axis is.

When the distance of master axis is alpha , the distance of slave axis is beta. Slave axis reach the given ratio after master axis move slope. User should call GT_SetGearRatio(slave, alpha, beta, slope)

5.4.3 Example

This example specify master axis as Jog mode and slave axis as electronic gear mode, and the ratio is 2:1. When the master axis completes clutch zone distance, slave axis reach the ratio . Refer to Fig 5-12, Fig 5-13.

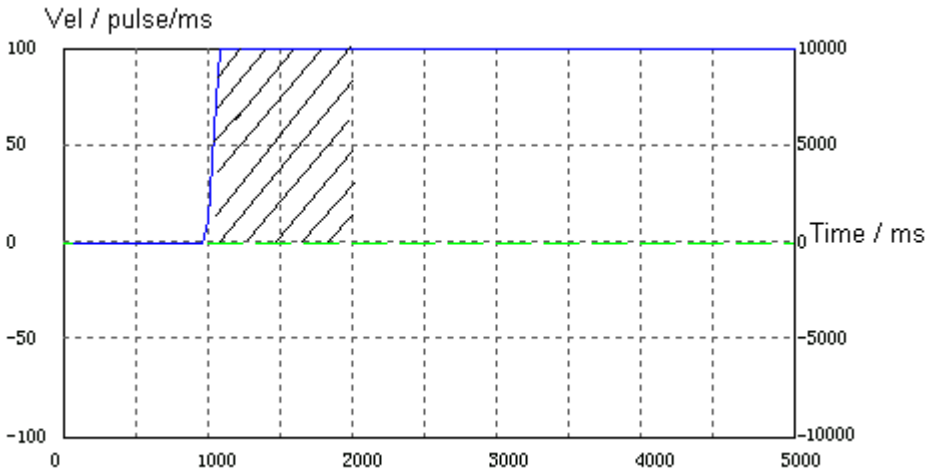

Fig 5-12 Velocity profile of master axis

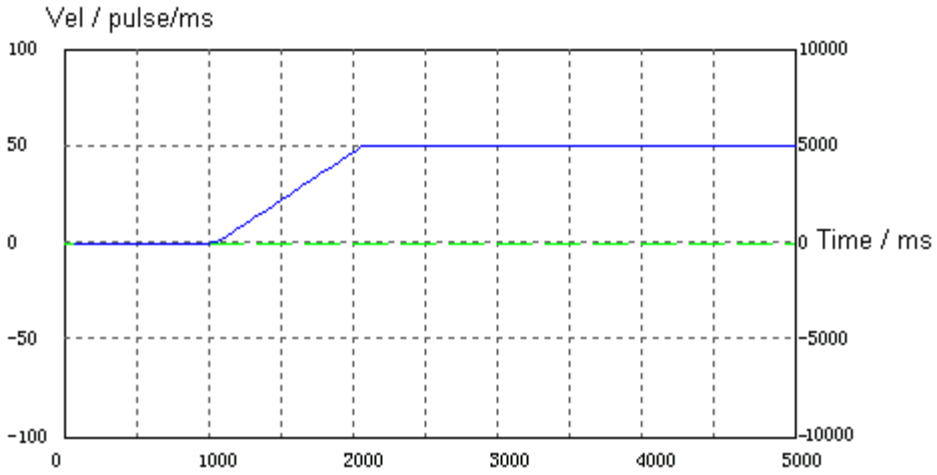

Fig 5-13 Velocity profile of slave axis

```
PROGRAM PLC_PRG

VAR

   Start: BOOL := FALSE;

   Enable: BOOL := TRUE;

   rtn: INT;

   MASTER: INT := 3;

   SLAVE: INT := 4;

   JogPrm:TJogPrm;
```

```
    MasterPrfVel: LREAL;
    SlavePrfVel: LREAL;
    MasterPrfPos: LREAL;
    SlavePrfPos: LREAL;
    Stop: BOOL := FALSE;
    StopDone: BOOL := FALSE;
    ModifyGearRation: BOOL := FALSE;
    ModifyGearRationDone: BOOL := FALSE;
END_VAR
-----------------------------------------------------
IF Start AND Enable THEN
    rtn:= GT_ClrSts(MASTER, 1);
    rtn:= GT_ClrSts(SLAVE, 1);
    rtn:= GT_AxisOn(MASTER);
    rtn:= GT_AxisOn(SLAVE);
    (*Set the master axis as Jog mode *)
    rtn:= GT_PrfJog(MASTER);
    JogPrm.acc:= 0.5;
    JogPrm.dec:= 0.5;
    JogPrm.smooth:= 0.8;
    rtn:= GT_SetJogPrm(MASTER, ADR(JogPrm));
    rtn:= GT_SetVel(MASTER, 40);
    rtn:= GT_Update(SHL(DINT#1, MASTER-1));
    (*Set the slave axis as Gear mode *)
    rtn:= GT_PrfGear(SLAVE, 0);
    rtn:= GT_SetGearMaster(SLAVE, MASTER, GEAR_MASTER_PROFILE, 0);
    rtn:= GT_SetGearRatio(SLAVE, 2, 1, 8000);
    rtn:= GT_GearStart(SHL(DINT#1, SLAVE-1));

    Enable:= FALSE;
END_IF
```

```
rtn:= GT_GetPrfVel(MASTER, ADR(MasterPrfVel), 1, 0);
rtn:= GT_GetPrfVel(SLAVE, ADR(SlavePrfVel), 1, 0);
rtn:= GT_GetPrfPos(MASTER, ADR(MasterPrfPos), 1, 0);
rtn:= GT_GetPrfPos(SLAVE, ADR(SlavePrfPos), 1, 0);

IF ModifyGearRation AND (NOT ModifyGearRationDone) THEN
   rtn:= GT_SetGearRatio(SLAVE, 4, 1, 10000);
   ModifyGearRationDone:= TRUE;
END_IF

IF Stop AND (NOT StopDone) THEN
   rtn:= GT_Stop(SHL(DINT#1, SLAVE-1), 0);
   rtn:= GT_Stop(SHL(DINT#1, MASTER-1), 0);
   rtn:= GT_ClrSts(MASTER, 1);
   rtn:= GT_ClrSts(SLAVE, 1);
   rtn:= GT_AxisOff(MASTER);
   rtn:= GT_AxisOff(SLAVE);
   StopDone:= TRUE;
END_IF
```

5.5 Follow Motion Mode

5.5.1 Commands summary

Tab 5-10 Summary of Following mode commands

Commands	Description
GT_PrfFollow	Set specified axis as following mode
GT_SetFollowMaster	Set master axis
GT_GetFollowMaster	Get the information of master axis
GT_SetFollowLoop	Set cycle numbers
GT_GetFollowLoop	Get the cycle numbers
GT_SetFollowEvent	Set start condition
GT_GetFollowEvent	Get the start condition
GT_FollowSpace	Get the free space of FIFO
GT_FollowData	Add data to FIFO
GT_FollowClear	Clear up FIFO data When the axis is in motion, this command is invalid
GT_FollowStart	Start following mode motion
GT_FollowSwitch	Switch FIFO from one to another

Tab 5-11 Description of Following mode commands

GT_PrfFollow (profile ,dir)	
Profile:INT	Profile No.
Dir:INT	Following type of slave axis. 0 represents the following type is bidirectional follow; 1 represents the following type is positive follow; -1 represents the following type is negative follow
GT_SetFollowMaster(profile, masterIndex, masterType,masterItem)	
Profile:INT	Profile No.
MasterIndex:INT	The index of master axis.
MasterType:INT	The type of master axis. FOLLOW_MASTER_ENCODER means follow encoder; FOLLOW_MASTER_PROFILE means follow profile axis; FOLLOW_MASTER_AXIS means follow synthetic axis.
MasterItem:INT	The type of synthetic axis 0: the profile position of synthetic axis. 1: the encoder position of synthetic axis.
GT_GetFollowMaster(profile, pMasterIndex, pMasterType, pMasterItem)	
Profile:INT	Profile No.
PMasterIndex: POINTER TO INT	The index of master axis.
PMasterType: POINTER TO INT	The index of master axis.
PMasterItem: POINTER TO INT	The type of synthetic axis.

GT_SetFollowLoop(profile,loop)	
Profile:INT	Profile No.
Loop:INT	Set cycle numbers of axis executed the Following mode.
GT_GetFollowLoop(profile,pLoop)	
Profile:INT	Profile No.
PLoop: POINTER TO DINT	Get cycle numbers of axis executed the Following mode.
GT_SetFollowEvent(profile,event,masterDir,pos)	
Profile:INT	Profile No.
Event:INT	Start condition of following action. FOLLOW_EVENT_STAR: start immediately after calling GT_FollowStart. FOLLOW_EVENT_PAS: start when master axis pass “pPos”
MasterDir:INT	Motion action of master axis. 1: master axis motion forward; -1: slave axis motion backward.
Pos:DINT	Pass position. Only when the “pEvent” is “FOLLOW_EVENT_PASS”, the “pPos” is effective.
GT_GetFollowEvent(profile,pEvent,pMasterDir,pPos)	
Profile:INT	Profile No.
PEvent: POINTER TO INT	Start condition of following action.
PMasterDir: POINTER TO INT	Motion action of master axis.
PPos: POINTER TO DINT	Pass position. Only when the “pEvent” is “FOLLOW_EVENT_PASS”, the “pPos” is effective.
GT_FollowSpace(profile,pSpace,fifo)	
Profile:INT	Profile No.
PSpace: POINTER TO INT	The returned free space of FIFO memory.
Fifo:INT	Specify which FIFO memory will inquire. The default value of “fifo” is 0.
GT_FollowData(profile,masterSegment,slaveSegment,type,fifo)	
Profile:INT	Profile No.
MasterSegment:DINT	Displacement of master axis.
SlaveSegment:DINT	Displacement of slave axis.
Type:INT	Type of segments. FOLLOW_SEGMENT_NORMAL: “normal” type segment, default value FOLLOW_SEGMENT_EVEN: “even” type segment. FOLLOW_SEGMENT_STOP: “stop” type segment. FOLLOW_SEGMENT_CONTINUE: the velocity between segments will be continuous.
Fifo:INT	Specify the FIFO which will store data. The default value is 0.
GT_FollowClear(profile, fifo)	
Profile:INT	Profile No.
Fifo:INT	FIFO which will be cleared. The default value is 0.
GT_FollowStart(mask, option)	
Mask:DINT	“Mask” represents specified axis No., which will be started by bit.

	Bit 0 represents axis1, bit 1 represents axis 2, and so on. When the bit X =1, the corresponding axis will be started.
Option:DINT	The FIFO memory which will be used by bit, default value is 0. Bit 0 represents axis1, bit 1 represents axis 2, and so on. When bit X=0, it means that the corresponding axis use FIFO1. When bit X=1, it means that the corresponding axis use FIFO2.
GT_FollowSwitch(mask)	
Mask:DINT	The FIFO memory which will be switched into Following mode by bit. Bit 0 represents axis1, bit 1 represents axis 2, and so on. When bitX=1, it means that controller will switch corresponding FIFO.

5.5.2 Highlights

In many applications, two or more axes need to keep synchronization of position and velocity. As illustrated in Fig 5-14.

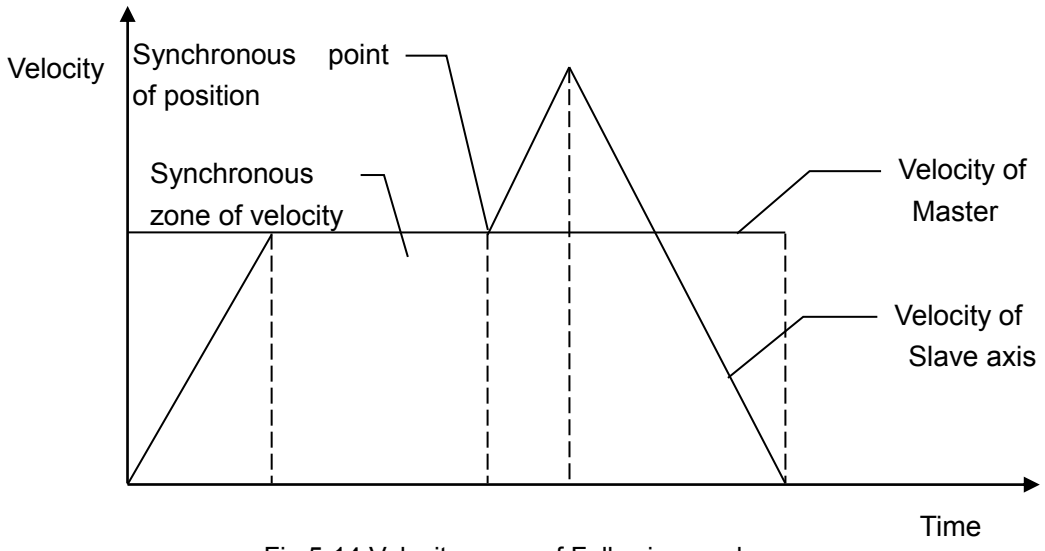

Fig 5-14 Velocity curve of Following mode

Position synchronous point means the position of master axis and slave axis reach at the same time.

Velocity synchronous area means the velocity ratio of master axis and slave axis reach at the same time are constant.

The first is acceleration area. In this area, the slave axis accelerates to synchronous velocity.

The second is velocity synchronous area.In this area, the slave axis and master axis always keep a set ratio until to the synchronous point of position. The velocity synchronous area is finished.

The third is acceleration area. In this area, after passing through the synchronous point of position, the slave axis accelerates rapidly and detaches from velocity synchronous area.

The fourth is deceleration area.In this area, the slave axis decelerates to 0 gradualy.

For less the following lag, it is recommended that the number of slave axis is greater than that of master axis.

(1) Types of segments

Following mode have 4 types of segments.

FOLLOW_SEGMENT_NORMAL represents normal segment, the starting-velocity ratio of subsequent segments is equal to 0. From the second segment, the velocity ratio of subsequent segments is equal to ending-velocity of last segment.

FOLLOW_SEGMENT_EVEN represents constant speed segments. In FIFO memory, the velocity ratio of each segment keeps invariant.

FOLLOW_SEGMENT_STOP represents stop segment, the velocity of ending-point is 0. The starting-velocity can be calculated by motion displacement and time of segment and it is independent with the last ending-velocity.

FOLLOW_SEGMENT_CONTINUE represents continuous segments; in the FIFO memory, the starting-velocity of subsequent segments is equal to ending-velocity of last segment. From the second segment on, the velocity of subsequent segments is equal to ending-velocity of last segment.

(2) Exchange FIFO

In following mode, there are 2 independent FIFO to save the data. These two FIFO can be exchanged with each other in motion.

As Fig 5-15 illustrated, motion rules of slave axis need to exchange from velocity curve 1 to velocity curve 3. To realize smooth velocity conversion of slave axis, velocity curve 2 is added. The starting-up velocity of velocity curve 2 is equal to velocity curve 1. The ending-up velocity of velocity curve 2 is equal to velocity curve 3.

To realize the continue velocity between two FIFO, user should set data type as FOLLOW_SEGMENT_CONTINUE when calling GT_FollowData.

Firstly, add “velocity curve 2 data” to FIFO2, then call GT_FollowSwitch() to exchange FIFO.

When all the data in FIFO1 have been executed, controller would exchange to FIFO2, and clear up all the data in FIFO1 automatically.

The controller will send data of velocity curve 3 to FIFO1 immediately after exchange to FIFO2, and then call GT_FollowSwitch() to exchange FIFO.

When all the data in FIFO2 have been executed, controller would exchange to FIFO1, and clear up all the data in FIFO2 automatically.

So far, the movement of slave axis is from “velocity curve 1” through “velocity curve 2” to “velocity curve 3”.

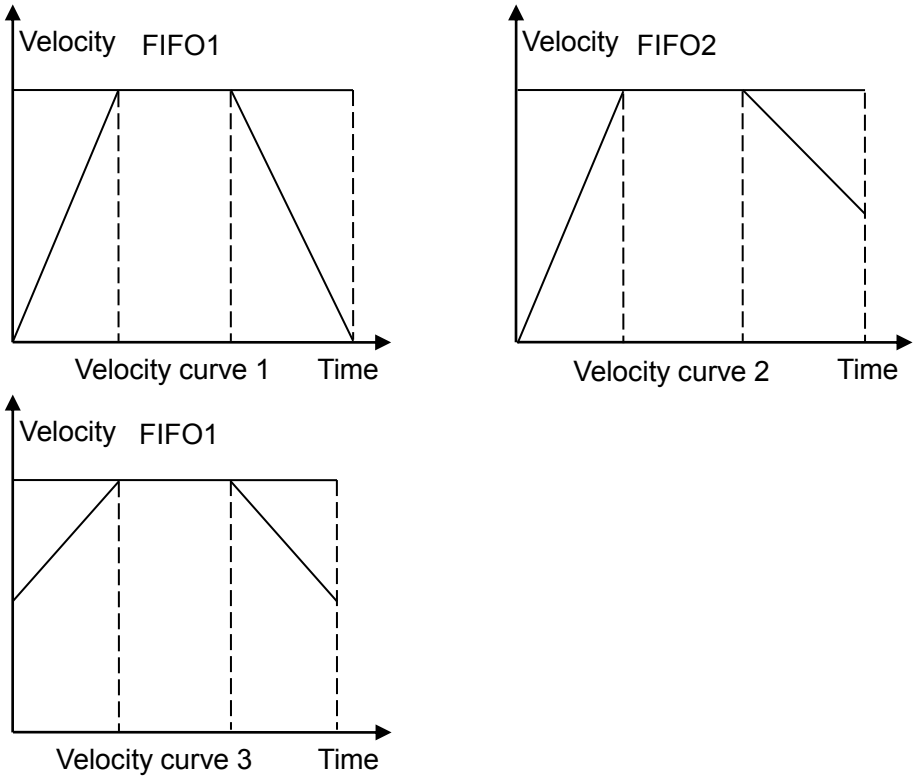

Fig 5-15 Exchange FIFO in Following mode

5.5.3 Example

(1) Follow single FIFO

In this program, master axis is in Jog motion mode and slave axis is in following motion mode. Slave axis motion principle includes 3 segments: acceleration segment, constant velocity segment, and deceleration segment, which is illustrated as below.

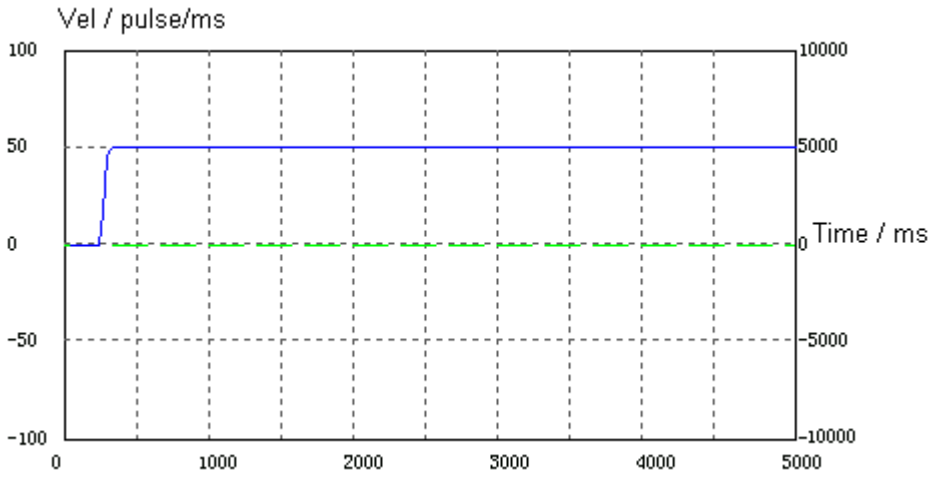

Fig 5-16 Velocity curve of master axis in Following single FIFO mode

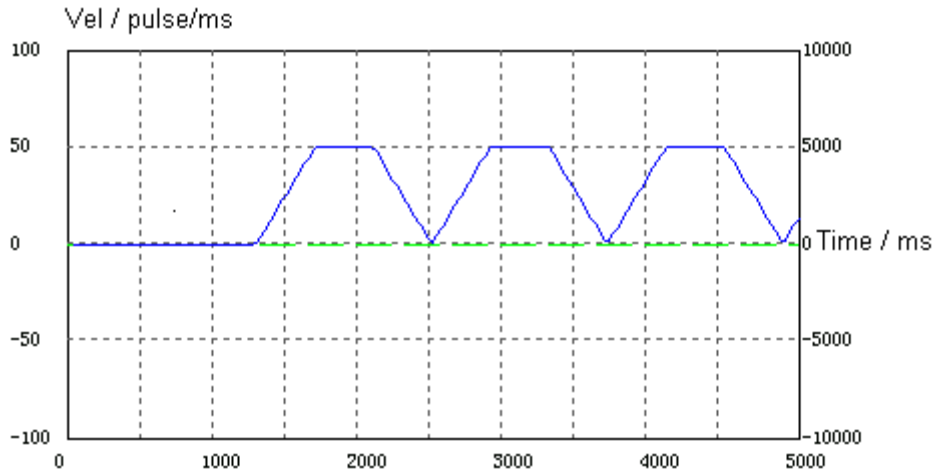

Fig 5-17 Velocity curve of slave axis in Following single FIFO mode

```
PROGRAM PLC_PRG

VAR

   Start: BOOL := FALSE;

   Enable: BOOL := TRUE;

   rtn: INT;

   MASTER: INT := 1;

   SLAVE: INT := 2;

   JogPrm: TJogPrm;

   MasterPos: DINT;

   SlavePos: DINT;

   MasterPrfVel: LREAL;

   SlavePrfPos: LREAL;

   SlavePrfVel: LREAL;

   MasterPrfPos: LREAL;

   Stop: BOOL := FALSE;

   StopDone: BOOL := FALSE;

END_VAR

------------------------------------------------------------

IF Start AND Enable THEN

   (*Servo on*)

   rtn:= GT_ClrSts(MASTER, 1);
```

```
rtn:= GT_ClrSts(SLAVE, 1);
rtn:= GT_AxisOn(MASTER);
rtn:= GT_AxisOn(SLAVE);

(*Set the motion mode of master axis as Jog mode *)
rtn:= GT_PrfJog(MASTER);
JogPrm.acc:= 0.2;
JogPrm.dec:= 0.2;
JogPrm.smooth:= 0.8;
rtn:= GT_SetJogPrm(MASTER, ADR(JogPrm));
rtn:= GT_SetVel(MASTER, 40);
rtn:= GT_Update(SHL(DINT#1, MASTER-1));

(*Set the motion mode of slave axis as Following mode *)
rtn:= GT_PrfFollow(SLAVE, 0);
rtn:= GT_SetFollowMaster(SLAVE, MASTER, FOLLOW_MASTER_PROFILE, 0);
rtn:= GT_FollowClear(SLAVE, 0);
(*Push data to buffer*)
MasterPos:= 40000;          (*acceleration segment *)
SlavePos:=10000;
rtn:= GT_FollowData(SLAVE, MasterPos, SlavePos, FOLLOW_SEGMENT_NORMAL, 0);
MasterPos:= MasterPos+80000;          (*constant velocity segment *)
SlavePos:= SlavePos+40000;
rtn:= GT_FollowData(SLAVE, MasterPos, SlavePos, FOLLOW_SEGMENT_NORMAL, 0);
MasterPos:=   MasterPos+40000;          (*deceleration segment *)
SlavePos:= SlavePos+10000;
rtn:= GT_FollowData(SLAVE, MasterPos, SlavePos, FOLLOW_SEGMENT_NORMAL, 0);

(*Set the cylcle mode as infinite, set the start condition of following motion is 40000pulse*)
rtn:= GT_SetFollowLoop(SLAVE, 0);
rtn:= GT_SetFollowEvent(SLAVE, FOLLOW_EVENT_PASS, 1, 40000);
```

```
    rtn:= GT_FollowStart(SHL(DINT#1, SLAVE-1), 0);


    Enable:= FALSE;
  END_IF
  (*Check the profile velocity and position*)
  rtn:= GT_GetPrfVel(MASTER, ADR(MasterPrfVel), 1, 0);
  rtn:= GT_GetPrfVel(SLAVE, ADR(SlavePrfVel), 1, 0);
  rtn:= GT_GetPrfPos(MASTER, ADR(MasterPrfPos), 1, 0);
  rtn:= GT_GetPrfPos(SLAVE, ADR(SlavePrfPos), 1, 0);


  IF Stop AND (NOT StopDone) THEN
    (*Servo off*)
    rtn:= GT_Stop(SHL(DINT#1, SLAVE-1), 0);
    rtn:= GT_Stop(SHL(DINT#1, MASTER-1), 0);
    rtn:= GT_ClrSts(MASTER, 1);
    rtn:= GT_ClrSts(SLAVE, 1);
    rtn:= GT_AxisOff(MASTER);
    rtn:= GT_AxisOff(SLAVE);
    StopDone:= TRUE;
  END_IF
```

(2) Exchange of two FIFO in following mode

The example set the master axis as Jog mode and set the slave axis as following mode. Slave axis can change following mode in motion, which is illustrated in Fig 5-18, Fig 5-19, Fig 5-20 and Fig 5-21

```
0001 PROGRAM PLC_PRG
0002 VAR
0003     rtn: INT;
0004     MASTER: INT := 2;
0005     SLAVE: INT := 4;
0006     JogPrm: TJogPrm;
0007     Start: BOOL := FALSE;
0008     masterPrfVel: LREAL;
0009     slavePrfVel: LREAL;
0010     Stop: BOOL := FALSE;
0011     StepOneDone: BOOL := FALSE;
0012     Space: INT;
0013     masterPos: DINT;
0014     slavePos: DINT;
0015     StepTwoDone: BOOL := FALSE;
0016     StepThreeDone: BOOL := FALSE;
0017     ServoOffDone: BOOL := FALSE;
0018     Switch: BOOL := FALSE;
0019 END_VAR
```

Fig 5-18 Example of two FIFO exchange (a)

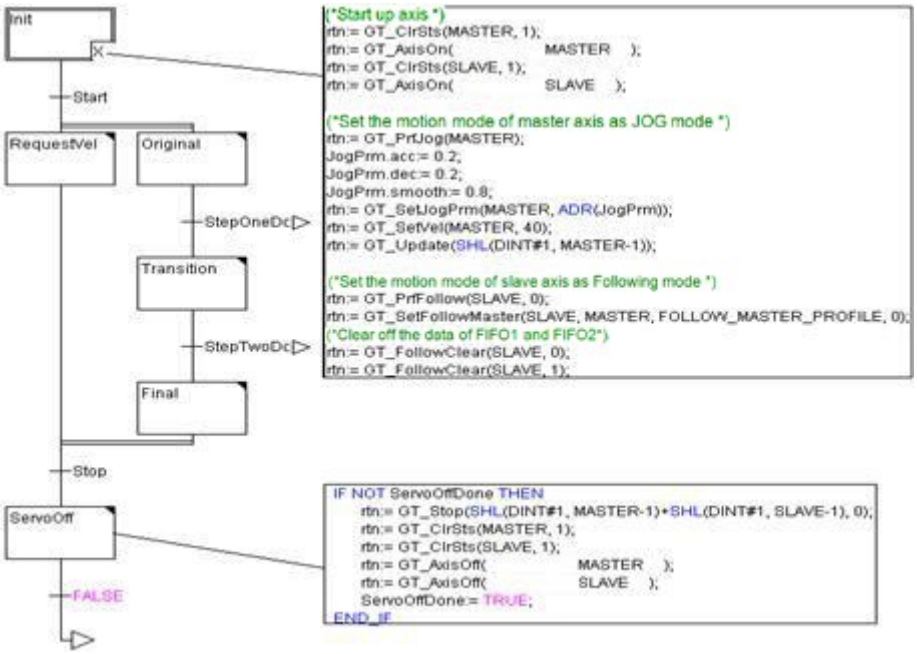

Fig 5-19 Example of two FIFO exchange (b)

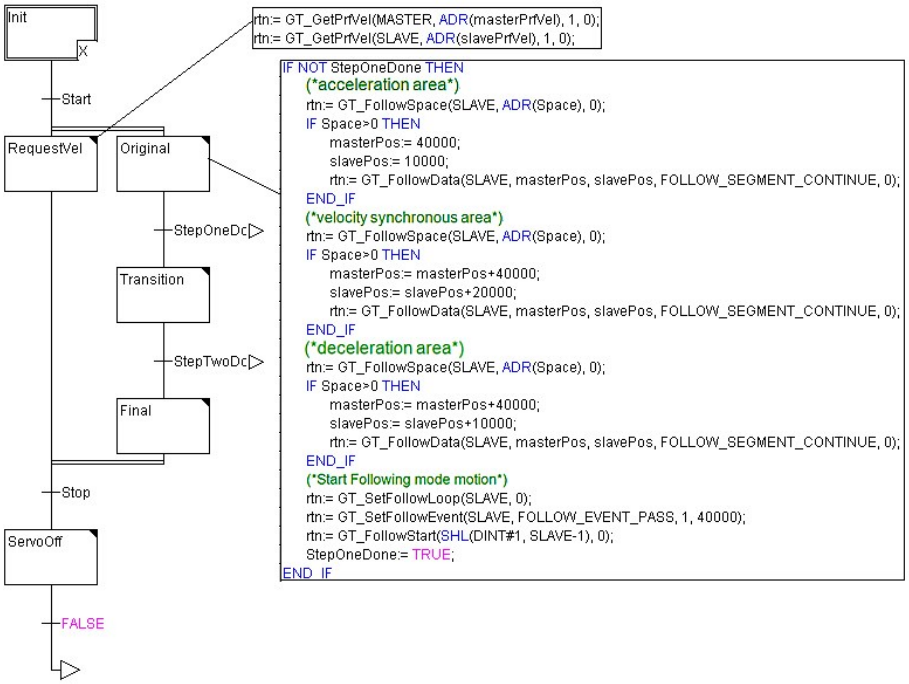

Fig 5-20 Example of two FIFO exchange (c)

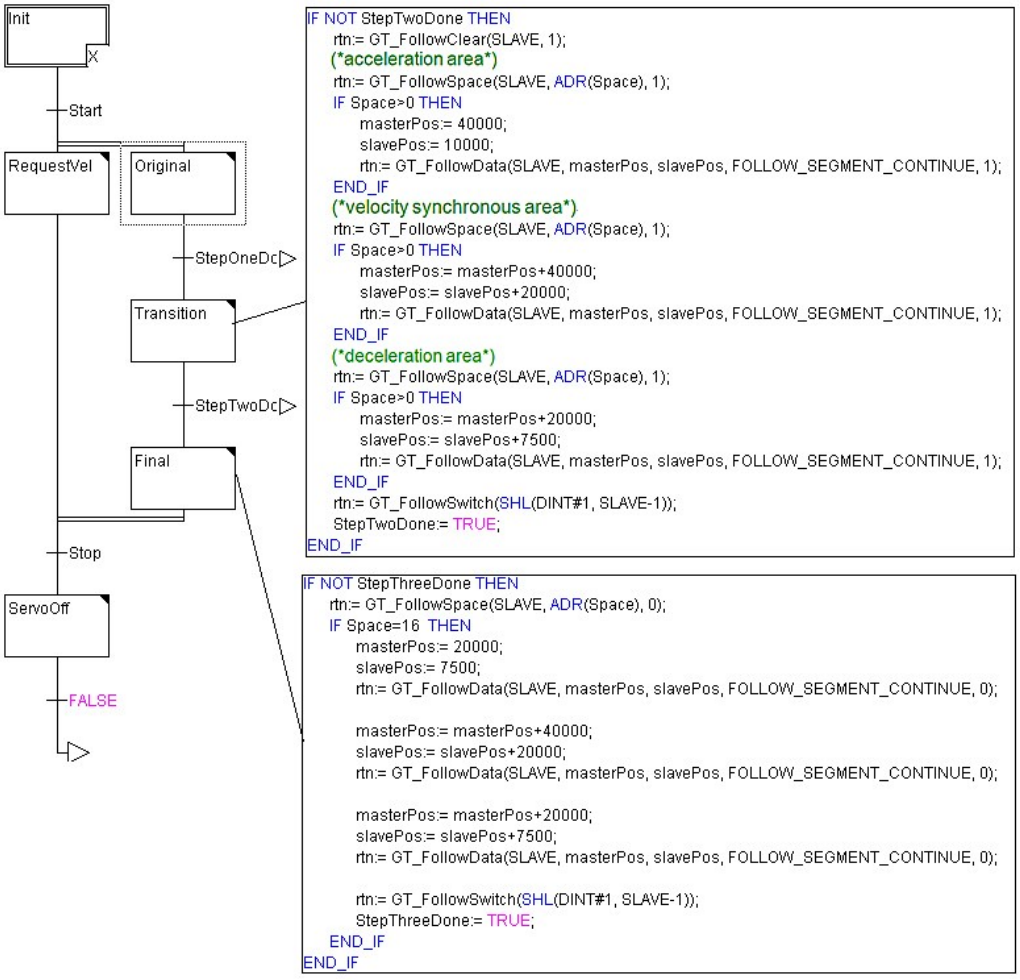

Fig 5-21 Example of two FIFO exchange (d)

Attention: Some commands of the of the several motion mode is time-consuming. Please do not invoke repeatedly when program. These commands are GT_Update(), GT_PTStart(), GT_GearStart(), GT_FollowStart().

Chapter 6 Access Hardware Resource

6.1 Access digital IO

6.1.1 Commands summary

Tab 6-1 Summary of access digital IO commands

Commands	Description
GT_GetDi	Get digital input status
GT_SetDo	Get digital input status
GT_SetDoBit	Set digital output status by bit
GT_GetDo	Get digital output status

Tab 6-2 Description of access digital IO commands

GT_GetDi(diType,pValue)	
DiType:INT	Type of digital IO input. MC_LIMIT_POSITIVE positive limit MC_LIMIT_NEGATIVE negative limit MC_ALARM drive alarm MC_HOME home MC_GPI general input
PValue:POINTER TO DINT	Status of digital input. Represent input level by bit. By default, “1” represents high level and “0” represents low level.
GT_SetDo(doType,value)	
DoType:INT	Type of digital output. MC_ENABLE servo on MC_CLEAR clear alarm output MC_GPO general output
Value:DINT	Represent input level by bit. By default, “1” represents high level and “0” represents low level.
GT_SetDoBit(doType,doIndex,value)	
DoType:INT	Type of digital output. MC_ENABLE servo on MC_CLEAR clear alarm output MC_GPO general output
DoIndex:INT	Index of digital output.
Value:INT	Set input level by bit. By default, “1” represents high level and “0” represents low level.
GT_GetDo(doType,pValue)	
DoType:INT	Type of digital output. MC_ENABLE servo on

	MC_CLEAR MC_GPO	clear alarm output general output
PValue:POINTER TO DINT	Status of digital output. Represent input level by bit. By default, “1” represents high level and “0” represents low level.	

6.1.2 Highlights

GT_GetDi() can get the input level status of limit switch, drive alarm, home, and general input.

GT_SetDo can set the output level status of drive enable, clear alarm output, and general output. By default, user can not specify output level of Servo on derectly because of the connection with axis.About how to set or cancel the connection of “do “with the “axis”, please refer to ”System Configuration”.

6.1.3 Example

```
Alarm，LimitPositive,limitNegative,home,gpi:WORD;
----------------------------------------------
(*Get the level of drive alarm*)
GT_GetDi(MC_ALARM,ADR(alarm));
(*Get the level of positive limit *)
GT_GetDi(MC_LIMIT_POSITIVE,ADR(limitPositive));
(*Get the level of negative limit *)
GT_GetDi(MC_LIMIT_NEGATIVE,ADR(limitNegative));
(*Get the level of home*)
GT_GetDi(MC_HOME,ADR(home));
(*Get the level of general input *)
GT_GetDi(MC_GPI,ADR(gpi));
```

6.2 Access encoder

6.2.1 Commands summary

Tab 6-3 Summary of access encoder commands

Commands	Description
GT_GetEncPos	Get encoder position.
GT_GetEncVel	Get encoder velocity.
GT_SetEncPos	Set the position of encoder.

Tab 6-4 Description of access encoder commands

GT_GetEncPos(encoder,pValue,count, pClock)	
Encoder:INT	Encoder No.
PValue:POINTER TO LREAL	Position of encoder.
Count:INT	“Count” represents the number of encoders to be read. The default value is 1, and maximum value is 8.
PClock:POINTER TO DWORD	Get the clock of motion controller.
GT_GetEncVel(encoder, pValue, count, pClock)	
Encoder:INT	Encoder No.
PValue:POINTER TO LREAL	Velocity of encoder.
Count:INT	“Count” represents the number of encoders to be read. The default value is 1, and maximum value is 8.
PClock: POINTER TO DWORD	Get the clock of motion controller.
GT_SetEncPos(encoder, encPos)	
Encoder:INT	Encoder No.
EncPos:DINT	Position of encoder.

6.2.2 Example

```
(*Get the position of 8 encoders *)
Enc,vel:array[0..7] of LREAL;
-------------------------------------------
GT_GetEncPos(1,ADR(enc[0]),8,0);
GT_GetEncVel(1,ADR(vel[0]),8,0);
GT_SetEncPos(1, enc[0]);
```

6.3 Access DAC

6.3.1 Commands summary

Tab 6-5 Summary of Access dac commands

Commands	Description
GT_SetDac	Set output voltage of dac.
GT_GetDac	Get output voltage of dac.

Tab 6-6 Description of Access dac commands

GT_SetDac(dac,pValue, count)	
Dac:INT	Start number of dac.
PValue: POINTER TO INT	Output voltage of dac

	The value of “-32768” represents -10V, and “32767” represents +10V.
Count:INT	“Count” represents how many dacs will be read. The default value is 1, and maximum value is 8.
GT_GetDac(dac, pValue, count, pClock)	
dac:INT	Start number of dac.
PValue: POINTER TO INT	Output voltage of dac
Count:INT	The number of dacs to be read. The default value is 1, and maximum value is 8.
PClock: POINTER TO DOWRD	Get the clock of motion controller.

Chapter 7 Safety Mechanism

7.1 Limit

Motion controller can specify the motion bound of axes by installing limit switch or setting soft limit. Refer to Fig 7-1.

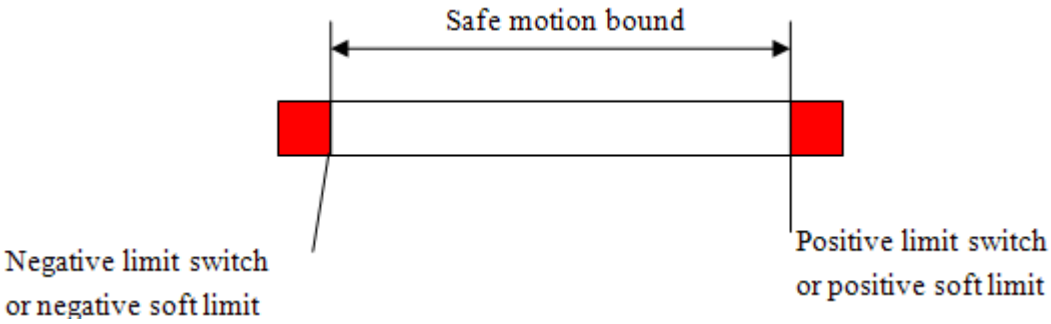

Fig 7-1 Motion bound of axes

When the worktable triggered the limit switch or the profile position was over the soft limit, the controller would stop the motion of worktable abruptly. When the limit has been trigger, the motion controller would not allow the axis to move towards the corresponding direction and limit trigger status of corresponding axis would be set as 1. After the controlled axis returns back to the safe motion bound, the user must call the command GT_ClrSts() to clear the relevant status bit, so as to restore the status of controlled axis from the status of over bound to normal status.

7.1.1 Commands summary

Tab 7-1 Summary of soft limit commands

Commands	Description
GT_SetSoftLimit	Set soft limit
GT_GetSoftLimit	Get soft limit

Tab 7-2 Description of soft limit commands

GT_SetSoftLimit(axis,positive,negative)	
Axis:INT	Axis No.
Positive:DINT	Positive soft limit. When profile position is greater than this value, the positive limit will be triggered. The default value: 0x7fffffff. Which means the positive soft limit is invalid.
Negative:DINT	Negaitive soft limit. When profile position is less than this value, the negative limit will be triggered.

	The default value: 0x80000000. Which means the negative soft limit is invalid.
GT_GetSoftLimit(axis,pPositive,pNegative)	
Axis:INT	Axis No.
PPositive: POINTER TO DINT	Get position of positive soft limit.
PNegative: POINTER TO DINT	Get position of negative soft limit.

7.1.1 Highlights

User should set the soft limit after the axis back to origin. The position value of soft limit must be greater than negative soft limit. The soft limit and limit switch can be used at the same time, and the limit status will also be set as 1 when the soft limit was triggered.

Motion controller will stop an axis abruptly once its limit was triggered. The default value of abrupt deceleration: 1 pulse/mm^2. Please refer to "***Profile configuration***" which shows how to set the abrupt deceleration value.

7.1.2 Example

```
PROGRAM PLC_PRG
VAR
    Start : BOOL;
    AXIS:INT;
    Trap: TTrapPrm;
    Sts : WORD;
    prfPos : LREAL;
END_VAR
  ----------------------------------------------------------------
  IF Start THEN
  rtn = GT_ClrSts(1,8);
    rtn = GT_SetSoftLimit(AXIS,20000,-20000);
    rtn = GT_PrfTrap(AXIS);
    rtn = GT_GetTrapPrm(AXIS,ADR(trap));
    trap.acc = 0.125;
    trap.dec = 0.125;
    rtn = GT_SetTrapPrm(AXIS,ADR(trap));
    rtn = GT_SetVel(AXIS,50);
```

```
    rtn = GT_SetPos(AXIS,1000000);

    rtn = GT_Update(SHL(1,(AXIS-1)));

    Start:=0;
END_IF

rtn = GT_GetSts(AXIS,ADR(sts),1,0);
rtn = GT_GetPrfPos(AXIS,ADR(prfPos),1,0);
```

7.2 Drive Alarm

Motion controller provides this specific drive alarm signal input interface. When drive alarm signal was detected, motion controller will servo off corresponding axis and stop motion profile abruptly, and set the alarm bit as 1.

When alarm signal was detected, the following operation should be executed,

1 Confirm the reason for drive alarm and correct it.

2 Reset the drive.

3 Call GT_ClrSts to clear the alarm bit, and redo the home capture to make worktable back to origin.

7.3 Smooth stop and emergency stop

For each axis in motion controller, user can define the smooth stop IO and emergency stop IO.

When the input value of smooth stop IO equals trigger sense value (the trigger sense can be set by user), the motion controller will automatically smooth stop the corresponding axes and set the bit7 of axis status as 1.

When the input value of emergency stop IO equals trigger sense value (the trigger sense can be set by user), the motion controller will automatically abruptly stop the corresponding axes and set the bit8 of axis status as 1.

After the smooth stop and emergency stop motion completed, user must call GT_ClrSts to clear the stop status bit (bit7 and bit8), otherwise the axis cannot move.

7.4 Error position limit

For servo control system, In some exceptional circumstances, the encoder position of motor may be far away from the profile position when some abnormal dangerous motion happens, such as motor error, encoder wire mistake, greater mechanic friction, and so on. In order to detect this situation quickly and improve the security of system, motion controller provides the security protection mechanism which uses error position limit to stop motion automatically.

The motion controller will check the error between encoder position and profile position per-sample period. If the error grows greater than the value of tracking error limit which was set by user, the motion controller will stop the axis abrupt automatically, and the error position limit status bit of this axis will be set as 1.

Chapter 8 Motion Status Detection

8.1 Commands summary

Tab 8-1 Summary of motion status detection commands

Commands	Description
GT_GetSts	Get the status of specified axis
GT_ClrSts	Clear the bit of drive alarm, following error limit, and position limit alarm ✓ Clear the bit of drive alarm, only when the alarm of drive is not been triggered. ✓ Clear the bit of following error limit only when following limit is in normal status. ✓ Clear the bit of position limit alarm in axis status register only when the axis has left the limit switch or profile position of the axis is in soft limit bound
GT_GetPrfMode	Get motion mode of specified profile
GT_GetPrfPos	Get profile position of specified profile
GT_GetPrfVel	Get profile velocity of specified profile
GT_GetPrfAcc	Get profile acceleration of specified profile
GT_SetAxisBand	Set arrival error band When the error between profile position and encoder position less than allowed scope, and the profile will be stopped, the arrival bit of motor will be set as 1.
GT_GetAxisBand	Get arrival error band.
GT_Stop	Stop one or more axe's profile motion.

Tab 8-2 Description of motion status detection commands

GT_GetSts(axis, pSts,count,pClock)	
Axis:INT	Axis No.
PSts:POINTER TO DINT	32 bits axis status characters. Please refer to highlights for detail info about of these status characters
Count:INT	The number of profiles to be read. The default value is 1. The maximum value is 8.
PClock: POINTER TO DWORD	Get the clock of motion controller.
GT_ClrSts(axis,count)	
Axis:INT	Axis No.
Count:INT	Number of axes need to be clear. The default value is 1. The maximum value is 8.
GT_GetPrfMode(profile,pValue,count, pClock)	
Profile:INT	Profile No.
PValue: POINTER TO DINT	The motion mode of profile 0: T-curve, default value 1: Jog mode 2: PT mode 3: Electronic gear mode

	4: Following mode
Count:INT	The number of profiles to be read. The default value is 1. The maximum value is 8.
PClock: POINTER TO DWORD	Get the clock of motion controller.
GT_GetPrfPos(profile, pValue, count, pClock)	
Profile:INT	Profile No.
PValue: POINTER TO LREAL	Profile position.
Count:INT	The number of profiles to be read. The default value is 1. The maximum value is 8.
PClock: POINTER TO DWORD	Get the clock of motion controller.
GT_GetPrfVel(profile, pValue, count, pClock)	
Profile:INT	Profile No.
PValue: POINTER TO LREAL	Profile velocity.
Count:INT	The number of profiles to be read. The default value is 1. The maximum value is 8.
PClock: POINTER TO DWORD	Get the clock of motion controller.
GT_GetPrfAcc(profile, pValue, count, pClock)	
Profile:INT	Profile No.
PValue: POINTER TO LREAL	Profile acceleration.
Count:INT	The number of profiles to be read. The default value is 1. The maximum value is 8.
PClock: POINTER TO DWORD	Get the clock of motion controller.
GT_SetAxisBand(axis, band, time)	
Axis:INT	Axis No.
Band:DINT	Allowed scope, unit: pulse.
Time:DINT	Error band holding time, unit: ms.
GT_GetAxisBand(axis, pBand, pTime)	
Axis:INT	Axis No.
PBand: POINTER TO DINT	Get the allowed scope.
PTime: POINTER TO DINT	Get the error band holding time.
GT_Stop(mask, option)	
Mask:DINT	“Mask” represents specified axis No. or coordinate No., which will be stopped by bit. Bit 0 represents axis1, bit 1 represents axis 2, and so on. When the bit X =1, the corresponding axis will be stopped.
Option:DINT	“Option” represents specified axis No. or coordinate No., which will be stopped in smooth or sudden mode by bit. Bit 0 represents axis1, bit 1 represents axis 2, and so on. When the bit X =0, the corresponding axis will be stopped in smooth mode. When the bit X =1, the corresponding axis will be stopped in sudden mode.

8.2 Highlights

Tab 8-3 Definition of axis status word

Bit	Definition
0	Reserved
1	Alarm bit of motor servo drive. If the drive of control axis alarms, this bit = 1
2	Reserved
3	Reserved
4	Motion error bit. If the position error exceeds the allowed scope, the controller will set this bit as 1. If only when the controller is not in motion error status, this bit can be cleared
5	Triggering bit of positive limit switch If the switch is triggered, this bit = 1 If the profile position exceeds positive soft limit, this bit=1
6	Triggering bit of negative limit switch If the switch is triggered, this bit=1 If the profile position exceeds negative soft limit, this bit=1
7	Triggering bit of smooth stop IO If the axis set the smooth stop IO, when the input of this IO has been triggered, this bit =1 and the axis will be smooth stopped
8	Triggering bit of emergency stop IO If the axis set the emergency stop IO, when the input of this IO has been triggered, this bit=1 and the axis will be emergency stopped
9	Motor activation status (1 means activated.)
10	Status bit of profile motion. When the profile manager is motion, this bit=1
11	Arrival bit of motor motion If profile manager is in static status, the error between profile position and encoder position is less the allowed scope (the allowed scope refer to GT_SetAxisBand), and the axis has stayed in allowed scope more than a setting period, this bit will be set as 1
12~31	Reserved

When these status bits of axis have been triggered, such as alarm bit of motor drive, triggering bit of limit switch and so on, the corresponding bit will not be automatically clear as 0. When the abnormal events of axis are eliminated, then user can call GT_ClrSts to clear the corresponding bit.

Status bit of profile motion (bit10) just represents the motion status in theory. Bit 10 is 1 means profile manager is in motion status, while bit is 0 means profile manager is in static status. Usually, because of motion following delay or mechanical system concussion, the mechanical system may not stop when the profile manager has stopped.

The arrival bit of motor motion (bit11) represents the actual arrival status of motor. Bit 10 is 1 means the profile manager is in static status, and the error between profile position and encoder position is less than the allowed scope, and the axis has stayed in allowed scope more than a setting period. When the error between profile position and encoder position is more than the allowed scope, this bit will be set as 0. Checking the arrival bit of motor motion can ensure system's positioning accuracy. User should set an appropriate allowed scope and the holding time. If the allowed scope is too narrow or the holding time is too long, the arrival time of motor motion will increase and the machining efficiency will decrease.

Tips of using arrival bit(bit11):

1. The axis should connect with the encoder correctly, and the direction of encoder is the same with the direction of profile.

2. Set the error band correctly. The error band is invalid by default.

3. After calling GT_SetPrfPos () or GT_SetEncPos(), should call GT_SynchAxisPos().

8.3 Example

This example illustrates the usage of the arrival bit (bit11). After the arrival of one axis, application starts the motion of another axis.

```
PROGRAM Main
VAR CONSTANT
    AXIS_X:INT:=1;
    AXIS_Y:INT:=2;
END_VAR
VAR
    Enable:BOOL:=TRUE;
    Rtn:INT;
    Trap: TTrapPrm;
    Sts:DINT;
    posX,posY:DINT;
    prfPos,prfVel:LREAL;
    Current_axis:INT:=1;
    X_DONE,Y_DONE:BOOL;
END_VAR
    ----------------------------------------------------------------------------
IF Enable THEN
    (*Activate the axis X and Enables the drive*)
    Rtn:= GT_AxisOn(AXIS_X);

    (*Activate the axis Y and Enables the drive *)
    Rtn:= GT_AxisOn(AXIS_Y);
```

```
(*Clear the profile postion of axis X *)
Rtn:= GT_SetPrfPos(AXIS_X,0);

(*Clear encoder position of axis X *)
Rtn:= GT_SetEncPos(AXIS_X,0);

(*Recalculate the axis position according to profile position*)
(*Recalculate the actual position according to the encoder position*)
Rtn:= GT_SynchAxisPos(SHL(DINT#1, AXIS_X-1));

(*Set the allowed band of axis X *)
Rtn:= GT_SetAxisBand(AXIS_X,20,5);

(*Clear profile position of axis Y*)
Rtn:= GT_SetPrfPos(AXIS_Y,0);

(*Clear encoder position of axis Y*)
Rtn:= GT_SetEncPos(AXIS_Y,0);

(*Recalculate the axis position according to profile position*)
(*Recalculate the actual position according to the encoder position*)
Rtn:= GT_SynchAxisPos(SHL(1,AXIS_Y-1));

(*Set the allowed band of axis Y *)
Rtn:= GT_SetAxisBand(AXIS_Y,20,5);

(*Set the motion mode of axis X as Point to Point *)
Rtn:= GT_PrfTrap(AXIS_X);

(*Get the motion parameter of axis X in Point to Point mode *)
```

```
        Rtn:= GT_GetTrapPrm(AXIS_X,ADR(trap));
        trap.acc:= 1;
        trap.dec:= 0.5;
        (*Set the motion parameter of axis X in Point to Point mode *)
        Rtn:= GT_SetTrapPrm(AXIS_X,ADR(trap));

        (*Set target velocity of X axis *)
        Rtn:= GT_SetVel(AXIS_X,10);

        (*Y Set the motion mode of axis Y as Point to Point *)
        Rtn:= GT_PrfTrap(AXIS_Y);
        (*Get the motion parameter of axis Y in Point to Point mode *)
        Rtn:= GT_GetTrapPrm(AXIS_Y,ADR(trap));
        trap.acc:= 1;
        trap.dec:= 0.5;
        (*Set the motion parameter of axis Y in Point to Point mode *)
        Rtn:= GT_SetTrapPrm(AXIS_Y,ADR(trap));
        (*Set target velocity of Y axis *)
        Rtn:= GT_SetVel(AXIS_Y,10);
        posX:= 10000;
        posY:= 20000;
        (*Set target position of X axis *)
        Rtn:= GT_SetPos(AXIS_X,posX);
    (*Start up the motion of axis X*)
        Rtn:= GT_Update(SHL(DINT#1, AXIS_X-1));
        Enable:=FALSE;
    END_IF
    (*Wait axis X enter into allowed band *)
    GT_GetSts(Current_axis,ADR(sts),1,0);
    GT_GetPrfPos(Current_axis,ADR(prfPos)1,0);
    GT_GetPrfVel(Current_axis,ADR(prfVel)1,0);
```

```
    IF 16#800 = ( sts AND 16#800 ) ) THEN
        IF NOT(X_DONE) THEN
            (*Set target position of Y axis *)
        Rtn:= GT_SetPos(AXIS_Y,posY);
          Rtn:= GT_Update(2);
          Current_axis:=AXIS_Y;
            X_Done:=TRUE;
        ELSE
            Y_Done:=TRUE;
        END_IF
    END_IF
```

Chapter 9 Motion Program

Reserved.

Chapter 10 Other Commands

10.1 Reset motion controller

Tab 10-1 Summary of open/close motion controller commands

Command	Description
GT_Reset	Reset motion controller

Before using motion controller, user must call GT_Open() to establish communication with motion controller. When user exit the application program, using GT_Close() to close motion controller.

User can call GT_Reset() to reset all registers of motion controller. Googoltech strongly recommended user call this command after open motion controller.

GT_SetCardNo() is used to switch one card as specified using motion controller. When more than one motion controllers have been set up into a PC, user can call this command to specify the specified using motion controller. After GT_SetCardNo() has been called successfully, the following GT commands called by user will only operates on this motion controller. In a multi-motion controller system, each motion controller will be assigned a card number (0 to 15) in order to distinguish from different motion controllers. The principle of assigning card number follows the PNP rule, in another words, motion controller which is firstly identified by the system will be set as number 0. So the system will assign the same card number to motion controllers if hardware does not change.

10.2 Get the firmware version

Tab 10-2 Description of get firmware version command

GT_GetVersion(pVersion)	
pVersion:POINTER TO STRING	Get the firmware version of motion controller.

In order to help user check the firmware version of motion controller, motion controller provides GT_GetVersion() to read version information. The firmware version contains 18 characters: aaa bbbbbb ccc dddddd as character string. The definition of this character string, please refer to Tab 10-3.

Tab 10-3 Definiton of firmware version

aaa	“aaa” represents the version number of firmware 1. For example, “100” represent version 1.00.
bbbbbb	“bbbbbb” represents the finish time of firmware 1. For example, “090908” reprsents the firmware 1 was finished in 2009-09-08.
ccc	“aaa” represents the version number of firmware 2.
dddddd	“bbbbbb” represents the finish time of firmware 2.

Example is as follows：

```
PROGRAM PLC_PRG

VAR
```

```
    version:LREAL;
    firmwareVersion:STRING(18);
    pVersion:POINTER TO STRING(18);
    rtn:INT;
    i:INT;
END_VAR
-----------------------------------------------------
rtn:=GetVersion(ADR(version));
rtn:=GT_GetVersion(ADR(pVersion));
firmwareVersion:=pVersion^;
```

10.3 Get the system clock

Tab 10-4 Description of get clock commands

GT_GetClock(pClock, pLoop)	
pClock:POINTER TO DWORD	Clock of motion controller, unit: ms.
Ploop:POINTER TO DWORD	an interior parameter which is not valid to user. The default value is 0.

After initialization of motion controller, the internal clock counting from 0 and adding 1 in 1 ms. User can get the value of this clock by using GT_GetClock(),and clear this clock by calling GT_Reset().

10.4 Enable/Disable servo

Tab 10-5 Description of enable/disable servo command

GT_AxisOn(axis)	
Axis:INT	Axis NO. of enabled servo motor.
GT_AxisOff(axis)	
Axis:INT	Axis NO. of disabled servo motor.

User can call GT_AxisOn() to enable the servo motor which is connected with related control axis, and make this control axis ready to be controlled. If user do not configurate do/di which connect to this axis in system configuration stage, GT_AxisOn() will invalid when user call this command(refer to 3.2.8 configurate do).

10.5 Position profile modification

Tab 10-6 Summary of position profile modification commands

Command	Description
GT_SetPrfPos	Position profile modification

GT_SynchAxisPos	Synchronize synthetic profile position with related profile manager Synchronize synthetic encoder position with related encoder manager
GT_ZeroPos	Set profile position and encoder position as zero, and zero drift compensation.

Tab 10-7 Description of position profile modification commands

GT_SetPrfPos(profile, prfPos)	
Profile:INT	Profile NO.
PrfPos:DINT	Set profile position.
GT_SynchAxisPos(mask)	
Mask:DINT	Axis No.which will be synchronized by bit. Bit 0 represents axis1, bit 1 represents axis 2, and so on. If the value of bit X is 1, the corresponding axis needs synchronize its position If the value of bit X is 0, the corresponding axis needs not synchronize its position
GT_ZeroPos(axis, count)	
Axis:INT	Axis No.
Count:INT	“Count” represents how many axes’ position will be cleared.

Axis can transform the unit of position output of profile and encoder, which discussed in chapter three. If user needs to synchronize the output of axis with profile or encoder, GT_SynchAxisPos() should be called, when the output of profile and encoder has been changed after user called GT_SetPrfPos() or GT_SetEncPos() .

10.6 Arrival detection

Tab 10-8 Summary of arrival detection commands

Commands	Description
GT_SetAxisBand	Set arrival error band When the error between profile position and encoder position less than allowed scope, and the profile will be stopped, the arrival bit of motor will be set as 1.
GT_GetAxisBand	Get arrival error band

Tab 10-9 Description of arrival detection commands

GT_SetAxisBand(axis, band, time)	
Axis:INT	Axis No.
Band:DINT	Allowed scope, unit: pulse.
Time:DINT	Error band holding time, unit: 250us.
GT_GetAxisBand(axis, pBand, pTime)	
Axis:INT	Axis No.
PBand:POINTER TO DINT	Get the allowed scope.
PTime:POINTER TO DINT	Get the error band holding time.

Servo motor may have motion delay when it is moving, which means the motor may not arrive the profile position after profile has stopped. User can use motor arrival detection function to check whether the motion has arrived the target position. By default, this function is invalid. This function becomes validly when user set the corresponding allowed band and it’s holding time by calling GT_SetAxisBand().

When this function is valid, if profile is in static status, i.e. the value of axis status bit (bit11) is 0(refer to 5.2.1), and the error between profile position and encoder position is less than the allowed band and the axis has stayed in allowed band more than the holding time, the value of axis status bit (bit11) will be set as 1(refer to 5.2.1). When the error between profile position and encoder position is larger than the allowed band, this bit will be set as 0. Checking the arrival bit of motor motion can ensure system's positioning accuracy. User should set an appropriate allowed band and holding time. If the allowed band is too narrow or the holding time is too long, the arrival time of motor motion will increase and the machining efficiency will decrease.

Tips of using arrival bit:

1. The axis should connect with the encoder correctly, and the direction of encoder is the same with the direction of profile.

2. Set the error band correctly. The error band is invalid by default.

3. After calling GT_SetPrfPos () or GT_SetEncPos(), should call GT_SynchAxisPos().

Chapter 11 Command List

Tab 11-1 Command summary GUC-ECATXX-M23-L2-F4G

System initialization	
GT_Reset	Reset motion controller。
GT_GetVersion	Get the firmware version of motion controller.
EtherCAT libray	
ecat_configure_done	Initialize communications return commands of EtherCAT.
GT_SetEcatGpioConfig	Set GPIO direction and effective level of EtherCAT GUC.
GT_StartEcatHoming	Start axes homing of EtherCAT.
GT_SetHomingMode	Switch axes homing mode of EtherCAT.
GT_GetEcatHomingStatus	Get axes homing status of EtherCAT.
GT_GetEcatProbeStatus	Get EtherCAT axes homing status of probe.
Access hardware resource	
GT_SetDo	Set digital output status.
GT_SetDoBit	Set digital output status by bit.
GT_GetDo	Get digital output status.
GT_GetDi	Get digital input status.
GT_SetDac	Set output voltage of dac.
GT_GetDac	Get output voltage of dac.
GT_SetEncPos	Set the position of encoder.
GT_GetEncPos	Get encoder position.
GT_GetEncVel	Get encoder velocity.
Logical management	
GT_ClrSts	Clear the bit of drive alarm, following limit alarm, and position limit alarm
GT_AxisOn	Enable the servo motor.
GT_AxisOff	Disable the servo motor.
GT_Stop	Stop type of specified axis is smooth stop or emergency stop.
GT_SetPrfPos	Modify the profile position
GT_SynchAxisPos	Synchronize synthetic profile position with related profile manager. Synchronize synthetic encoder position with related encoder manager.
GT_SetSoftLimit	Set soft limit
GT_GetSoftLimit	Get soft limit
GT_SetAxisBand	Set arrival error band.
GT_GetAxisBand	Get arrival error band.
Motion status of detection	
GT_GetSts	Get the status of specified axis
GT_GetPrfPos	Get profile position of specified profile.
GT_GetPrfVel	Get profile velocity of specified profile.
GT_GetPrfAcc	Get profile acceleration of specified profile.
GT_GetPrfMode	Get motion mode of specified profile.
GT_GetAxisPrfPos	Get transformed profile position of specified axis.

GT_GetAxisPrfVel	Get transformed profile velocity of specified axis.
GT_GetAxisPrfAcc	Get transformed profile acceleration of specified axis.
GT_GetAxisEncPos	Get transformed encoder position of specified axis.
GT_GetAxisEncVel	Get transformed encoder velocity of specified axis.
GT_GetAxisEncAcc	Get transformed encoder acceleration of specified axis.
GT_GetAxisError	Get the difference between transformed profile position and transformed encoder position.
GT_GetClock	Get the internal clock of motion controller.
Point to Point mode	
GT_PrfTrap	Set specified axis as Point to Point motion mode.
GT_SetTrapPrm	Set parameters of Point to Point motion mode.
GT_GetTrapPrm	Get parameters of Point to Point motion mode.
GT_SetPos	Set target position.
GT_GetPos	Get target position.
GT_SetVel	Set target velocity.
GT_GetVel	Get target velocity.
GT_Update	Start motion of Point to Point mode.
Jog mode	
GT_PrfJog	Set specified axis as Jog mode.
GT_SetJogPrm	Set parameters of Jog mode.
GT_GetJogPrm	Get parameters of Jog mode.
GT_SetVel	Set target velocity, unit: pulse/ms.
GT_GetVel	Get target velocity, unit: pulse/ms.
GT_Update	Start motion of Jog mode.
PT mode	
GT_PrfPt	Set specified axis as PT mode.
GT_SetPtLoop	Set the cycle numbers.
GT_GetPtLoop	Get the cycle numbers.
GT_PtSpace	Get the free space of FIFO.
GT_PtData	Add data to FIFO.
GT_PtClear	Empty FIFO data.
GT_PtStart	Start PT mode motion.
Electronic gear mode	
GT_PrfGear	Set specified axis as Electronic gear mode.
GT_SetGearMaster	Set master axis.
GT_GetGearMaster	Get the information about the master axis.
GT_SetGearRatio	Set the electronic gear ratio.
GT_GetGearRatio	Get the electronic gear ratio.
GT_GearStart	Start Electronic gear mode motion.
Following mode	
GT_PrfFollow	Set specified axis as following mode.
GT_SetFollowMaster	Set master axis.
GT_GetFollowMaster	Get the information of master axis.

GT_SetFollowLoop	Set cycle numbers.
GT_GetFollowLoop	Get the cycle numbers.
GT_SetFollowEvent	Set start condition.
GT_GetFollowEvent	Get the start condition.
GT_FollowSpace	Get the free space of FIFO.
GT_FollowData	Add data to FIFO.
GT_FollowClear	Empty FIFO data. When the axis is in motion, this command is invalid.
GT_FollowStart	Start Following mode motion.
GT_FollowSwitch	Switch FIFO from one to another.
Motion program	
GT_Download	Download motion program to controller core.
GT_GetFunId	Read function ID of motion program.
GT_Bind	Binding thread, function and data page
GT_RunThread	Start a thread
GT_StopThread	Stop the running thread
GT_PauseThread	Pause the running thread
GT_GetThreadSts	Read thread status
GT_GetVarId	Read variable ID of motion program.
GT_SetVarValue	Set the variable value of motion program
GT_GetVarValue	Get the variable value of motion program

Chapter 12 Encryption Mechanism

1. At present, two types of encryption modes are supported:

   (1) Software encryption: the application development developers set password protection program during the application development process under OtoStudio environment. Such protection is divided into two types:

   1) Protect program code: under the catalog of OtoStudio->option->password: set password and protection password.

   2) Protect program run: for example, in OtoStudio program development, a password comparison sentence is given out in the program as the condition of running or not running this program.

   (2) Hardware encryption: Googoltech could provide two types of hardware encryption modes:

   1) When running GRT.exe, automatically check the hardware version No.; in case of incorrect version number, this program could not be normally started. (The version number is provided by Googoltech, and the same series of our controllers have the same version number.)

   2) The function used for binding the adapter address is provided, so the application program developer could encrypt the application program through reading the adapter address. (Each set of hardware platform has unique adapter address and could not be modified.)

2. For payment encryption: Googoltech could provide the following reference scheme:

Set timer in the application program and read CPU clock. For example, client is wished to pay the money within three months, or else the application program could not normally run. Through such mode, firstly wait for three months' system clock and then verify the software encryption [2].

Tab 12-1 Specification of Encryption Function

GetMacAddress (ulAdapterNumber, pMacAddress , ulAddressSize)	
ulAdapterNumber: UINT	Adapter number of present controller, default value is 0
pMacAddress: POINTER TO BYTE	Output adapter address, namely the first array address
ulAddressSize: UINT	Array length of adapter; unit: Byte