PCI-8372+/8366+
cPCI-8312H

SSCNET Motion Control Card

Function Library

Manual Rev. 2.02

Revision Date: June 13, 2008

Part No: 50-1H002-1020

Recycled Paper

Advance Technologies; Automate the World.

The information in this document is subject to change without prior notice in order to improve reliability, design, and function and does not represent a commitment on the part of the manufacturer.

In no event will the manufacturer be liable for direct, indirect, special, incidental, or consequential damages arising out of the use or inability to use the product or documentation, even if advised of the possibility of such damages.

Trademarks

Product names mentioned herein are used for identification purposes only and may be trademarks and/or registered trademarks of their respective companies.

Getting Service from ADLINK

Customer Satisfaction is top priority for ADLINK Technology Inc. Please contact us should you require any service or assistance.

ADLINK TECHNOLOGY INC.

Web Site:	http://www.adlinktech.com
Sales & Service:	Service@adlinktech.com
TEL:	+886-2-82265877
FAX:	+886-2-82265717
Address:	9F, No. 166, Jian Yi Road, Chungho City, Taipei, 235 Taiwan

Please email or FAX this completed service form for prompt and satisfactory service.

Company Information	
Company/Organization	
Contact Person	
E-mail Address	
Address	
Country	
TEL	FAX:
Web Site	
Product Information	
Product Model	
Environment	OS: M/B: CPU: Chipset: Bios:

Please give a detailed description of the problem(s):

Table of Contents

List of Tables

1 SSCNET Function Library

This chapter describes the software functions in detail for the SSCNET Series cards. Users can use these functions to develop application programs in C, Visual Basic or C++ programming languages. If Delphi is used as the programming environment, it is necessary to transform the header file, MDSP.h, manually.

1.1 SSCNET Function List

Category	Section	Function
System Function	1.3	error_msg(ErrorNo, *ErrorMsg)
		MDSP_initial(CardIDType,CardNo)
		MDSP_close(CardID)
		MDSP_reset(CardID)
		get_base_addr(CardID, *BaseAddr)
		get_irq_number(CardID, *IrqNo)
		understand_mdsp(CardID, *Version, *Date)
		get_version_info(CardID, *Hardware, *Software_DLL, *Software_Driver

Table 1-1: SSCNET Function List

Category			Section	Function
Motion Command	Single motion	Single axis velocity move	1.4	tv_move(Axis, StrVel, MaxVel, Tacc)
				sv_move(Axis, StrVel, MaxVel, Tacc, Tlacc)
		Single axis P to P motion		start_tr_move(Axis, Dist, StrVel, MaxVel, FinVel, Tacc, Tdec)
				start_sr_move(Axis, Dist, StrVel, MaxVel, FinVel, Tacc, Tdec, Tlacc, Tldec)
				start_ta_move(Axis, Pos, StrVel, MaxVel, FinVel, Tacc, Tdec)
				start_sa_move(Axis, Pos, StrVel, MaxVel, FinVel, Tacc, Tdec, Tlacc, Tldec)
		Multi axes velocity move		tv_move_all(Length, *Axis, *StrVel, *MaxVel, *Tacc)
				sv_move_all(Length, *Axis, *StrVel, *MaxVel, *Tacc, *Tlacc)
		Multi axes P to P		start_tr_move_all(Length, *Axis, *Dist, *StrVel, *MaxVel, *FinVel, *Tacc, *Tdec)
				start_sr_move_all(Length, *Axis, *Dist, *StrVel, *MaxVel, *FinVel, *Tacc, *Tdec, *Tlacc, *Tldec)
				start_ta_move_all(Length, *Axis, *Pos, *StrVel, *MaxVel, *FinVel, *Tacc, *Tdec)
				start_sa_move_all(Length, *Axis, *Pos, *StrVel, *MaxVel, *FinVel, *Tacc, *Tdec, *Tlacc, *Tldec)
				tv_change(Axis, SpeedFactor, Tacc)
				sv_change(Axis, SpeedFactor, Tacc,)
				tv_stop(Axis, Tdec)
				sv_stop(Axis, Tdec)
				emg_stop(Axis)
		Multi axes linear interpolation		start_line_tr_move(Length, *AxisArray, *DistArray, StrVel, MaxVel, FinVel, Tacc, Tdec)
				start_line_sr_move(Length, *AxisArray, *DistArray, StrVel, MaxVel, FinVel, Tacc, Tdec, Tlacc, Tldec)
				start_line_ta_move(Length, *AxisArray, *PosArray, StrVel, MaxVel, FinVel, Tacc, Tdec)
				start_line_sa_move(Length, *AxisArray, *PosArray, StrVel, MaxVel, FinVel, Tacc, Tdec, Tlacc, Tldec)
		Circular interpolation		start_arc_tr_move(*AxisArray, *CenterArray, Angle, StrVel, MaxVel, FinVel, Tacc, Tdec)
				start_arc_sr_move(*AxisArray, *CenterArray, Angle, StrVel, MaxVel, FinVel, Tacc, Tdec, Tlacc, Tldec)
				start_arc_ta_move(*AxisArray, *CenterArray, Angle, StrVel, MaxVel, FinVel, Tacc, Tdec)
				start_arc_sa_move(*AxisArray, *CenterArray, Angle, StrVel, MaxVel, FinVel, Tacc, Tdec, Tlacc, Tldec)
	Home move			home_move(Axis, StrVel, MaxVel, Tacc)
				set_home_mode(Axis, HomeMode)
	Continuous motion	Start / End motion list		start_motion_list(Length, *AxisArray)
				end_motion_list(void)
				repeat_last_move(Axis)
		Add linear trajectory		add_line_tr_move(*DistArray, StrVel, MaxVel, FinVel, Tacc, Tdec)
				add_line_sr_move(*DistArray, StrVel, MaxVel, FinVel, Tacc, Tdec, Tlacc, Tldec)
				add_line_ta_move(*PosArray, StrVel, MaxVel, FinVel, Tacc, Tdec)
				add_line_sa_move(*PosArray, StrVel, MaxVel, FinVel, Tacc, Tdec, Tlacc, Tldec)
		Add arc trajectory		add_arc_tr_move(*CenterArray, Angle, StrVel, MaxVel, FinVel, Tacc, Tdec)
				add_arc_sr_move(*CenterArray, Angle, StrVel, MaxVel, FinVel, Tacc, Tdec, Tlacc, Tldec)
				add_arc_ta_move(*CenterArray, Angle, StrVel, MaxVel, FinVel, Tacc, Tdec)
				add_arc_sa_move(*CenterArray, Angle, StrVel, MaxVel, FinVel, Tacc, Tdec, Tlacc, Tldec)
				add_arc2_sa_move(*AxisArray, *CenterPosArray, Angle, StrVel, MaxVel, FinVel, Tacc, Tlacc, Tdec, Tldec)
				add_arc2_sr_move(*AxisArray, *CenterDistArray, Angle, StrVel, MaxVel, FinVel, Tacc, Tlacc, Tdec, Tldec);
				add_arc2_ta_move(*AxisArray ,*CenterPosArray, Angle, StrVel, MaxVel, FinVel, Tacc, Tdec);
				add_arc2_tr_move(*AxisArray, *CenterDistArray, Angle, StrVel, MaxVel, FinVel, Tacc, Tdec);
		Dwell		add_dwell(Sec)
		Smooth Trajectory		smooth_enable(Flag, R)
		Start/Sop command		start_cont_move(Void)

Table 1-1: SSCNET Function List

Category		Section	Function
Motion related I/O	Position control and feedback	1.5	get_position(Axis,*Pos_F, *Pos_C)
			set_position(Axis,Pos)
			get_fb_position(Axis,*Pos_F)
			get_target_pos(Axis,*TargetPos)
			get_move_ratio(Axis, *PulsePerMM)
			set_move_ratio(Axis, PulsePerMM)
			shift_position(AxisNo, shiftMM)
	Velocity feedback		get_velocity(Axis,*Vel_F, *Vel_C)
			get_cnt_speed(CardId,EncNo,*PPS)
	Motion DIO status		set_PEL_config(Axis, Logic, mode)
			set_MEL_config(Axis, Logic, mode)
			set_ORG_config(Axis, Logic)
			set_EMG_config(CardID, Logic)
			get_PEL_status(Axis, *status)
			get_MEL_status(Axis, *status)
			get_ORG_status(Axis, *status)
			get_EMG_status(CardID, *status)
	Software Limit		set_soft_limit(Axis, PLimit, Mlimit, ON_OFF)
			get_soft_limit(Axis, *PLimit, *Mlimit, *ON_OFF)
	Motion status		motion_status(Axis, *MotionStatus)
			axis_status(Axis,*AxisStatus)
			motion_done(Axis)
			frame_to_execute(Axis, *Len)
General Purpose IO	Encoder	1.6	set_cnt_iptmode(CardID, EncNo, IptMode)
			set_cnt_to_axis(CardID,EncNo,Axis,Resolution)
			set_cnt_value(CardID, EncNo, Value)
			get_cnt_value(CardID, EncNo, *Value)
			get_ring_counter(CardID,ExtCntNo,*RingValue)
			set_ring_counter(CardID,ExtCntNo, RingValue)
	DIO		get_di_status(CardID, ChNo, *Sts)
			set_do_value(CardID, ChNo, Value)
			set_di_mode(CardID, DI_Channel, DI_Mode, ActiveL)
			set_mio_mode(CardID, DI_Ch, Mode)
	DA		set_da_config(CardID, ChNo, Cfg)
			set_da_value(CardID,ChNo,Value)
	AD		set_ad_function(CardID, Enable, AD_Gain, AD_Last, AD2_SRC)
			get_ad_value(CardID, ChNo, *Value)
	Analog auto calibration		tune_ref_5v(CardID, Value)
			save_auto_k_value(CardId, Channel, Value)
			get_auto_k_value(CardId, Channel, *Value)
			tune_ad_offset_gain(CardId, Step, Value)
			tune_da_offset(CardId, Step, Value)
			reload_auto_k_setting(CardID)

Table 1-1: SSCNET Function List

Category		Section	Function
Driver Management	Driver parameters	1.7	get_servo_para(Axis, ParaNo,*Value)
			set_servo_para(Axis, ParaNo, Value)
			get_servo_para_all(Axis, *Value)
			set_servo_para_all(Axis, *Value)
			save_servo_para(Axis)
			set_servo_para_default(Axis)
	Data Monitoring		get_monitor_data(Axis, *Data)
			get_instant_monitor_data(Axis,*Data_0,*Data_1,*Data_2, *Data_3)
			start_monitor(Axis)
			check_monitor_ready(Axis, *status)
			set_monitor_channel(Axis, Channel_0, Channel_1, Channel_2, Channel_3)
			set_monitor_config(Axis, Trigger_Select, Trigger_Level, SamplePeriod, PreTriggerSampleNo, Sample-Number)
	Servo information		get_servo_info(Axis,*ServoInfo)
	Servo ON		set_servo_on(Axis,ON_OFF)
	Driver information		understand_driver(I16 Axis, U16 *Class_Code)
			understand_motor(Axis, *MotorType, *Capacity, *RateRPM, *RateCurrent, *MaxRPM, *MaxTorq, *PPR, *ENCInfo, *OptionalInfor)
	Alarm		Get_alarm_no(Axis,*AlarmNo)
			alarm_reset(Axis)
Control Gain Tuning		1.8	set_auto_tune(Axis, Mode, RSP, GD2)
			get_auto_tune(Axis, *Mode, *RSP, *GD2)
			set_control_gain(Axis, PG1, VG1, VIC,PG2, VG2, FFC)
			get_control_gain(Axis, *PG1, *VG1, *VIC,*PG2, *VG2, *FFC)
			set_notch_filter(Axis, Mode, NotchFrequency, NotchDepth)
			get_notch_filter(Axis, *Mode, *NotchFrequency, *NotchDepth)
			set_LP_filter(Axis,ON_OFF)
			get_LP_filter(Axis,*ON_OFF)
Interrupt Control		1.9	int_control(CardID, Flag)
			set_int_factor(CardID, Source, IntFactor)
			get_int_status(CardID, Source, *IntStatus)
			set_int_event(CardID,*HEvent)
			link_interrupt(CardID, *callbackAddr)
			clear_tlc_int(Axis)
			set_timer_interval(CardID, Sec)
Position Compare		1.10	set_compare(Axis,CMP1Pos,CMP1Dir,CMP2Pos,CMP2Dir)
			check_compare(Axis, *status)
			set_single_compare(Axis,Channel,CMP_Pos)
			map_dout_and_comparator(CardID,DOut_Ch,AxisNo, CompNo, DOut_mode)
			set_compare_table_dir(CardID, Table_ChNo, Dir)
			link_dout_and_compare_table(CardID, DO_ChNo, StartI, EndI, *TableData)
			set_ext_encoder_compare_method(CardID, EncNo, Mode)
			set_ext_encoder_compare_value(CardID, EncNo,CompareValue)
			set_compare_source
Inter Lock Function		1.11	set_interlock(CardID,Enable, Axis_X, Axis_Y, X1, X2, Y1, Y2)
			get_interlock(CardID, *Enable, *Axis_X, *Axis_Y, *X1, *X2, *Y1, *Y2)
Absolute Position Control		1.12	get_abs_position(Axis, *ABS_Pos)
			save_abs_position(Axis)
			clear_abs_data_on_flash(CardID)
Pulse Output Control		1.13	set_pulse_output_control(Axis, Enable, PulseCH, Mode)

Table 1-1: SSCNET Function List

Category	Section	Function
Sequence Motion Control	1.14	add_frame_ta_move(Axis, StartFrameNo, StartPos, EndPos, StartVel, MaxVel, FinVel, Tacc, Tdec);
		add_frame_dwell(Axis, StartFrameNo, DTime);
		get_pattern(CardID, PatternNo, *FirstFrame, *TotalFrame, *SyncAxes);
		insert_pattern_to_seq_buffer
		set_pattern(CardID, PatternNo, FirstFrame, TotalFrame, SyncAxes);
		insert_pattern_to_seq_buffer(CardID, SeqNo, PatternNo, StartCondition, SyncAxes);
		check_seq_buffer(CardID, SeqNo, *Begin, *End);
		start_seq_move(CardID, SeqNo);
		reset_seq_buffer(CardID, SeqNo);
		pause_seq_move(CardID, SeqNo, Dec_Time);
		resume_seq_move(CardID, SeqNo, Acc_Time);
		end_seq_move
		set_seq_sync_pause
		check_seq_buffer_index
		check_seq_buffer_empty_count
Auto Stop Function	1.15	set_pos_diff_stop(CardID, Enable, PosDiffCheckSetNo_0_To_5, Axis_X, Axis_Y, UpperLimit, LowerLimit)

Table 1-1: SSCNET Function List

1.2 C/C++ Programming Library

1.2.1 Data Type

This section gives details of all the functions. The function prototypes and some common data types are declared in MDSP.h. These data types are used by the PCI-83XX Series library. We suggest you to use these data types in your application programs. The following table shows the data type names and their range.

Type Name	Description	Range
U8	8-bit ASCII character	0 to 255
I16	16-bit signed integer	-32768 to 32767
U16	16-bit unsigned integer	0 to 65535
I32	32-bit signed long integer	-2147483648 to 2147483647
U32	32-bit unsigned long integer	0 to 4294967295
F32	32-bit single-precision floating-point	-3.402823E38 to 3.402823E38
F64	64-bit double-precision floating-point	-1.797683134862315E308 to 1.797683134862315E309
Boolean	Boolean logic value	TRUE, FALSE

Table 1-2: Data Types

1.2.2 Card Indexing

`MDSP_initial()` is used to initialize the SSCNET motion control card and register it to the system. Users must give this function a card type and card number parameters. The card type could be any of SSCNet model name. The card number is indexing from 0. This function will return a signed 16-bit integer. If the return value is negative, it means an error happens. Please lookup the error code table to understand it. If it is a non-negative value, it contents two pieces of information inside this value. The lower 8-bit is “CardID”, which represents this card and the upper 8-bit is total axes found on this card.

The range of “CardID” is from 0 to 31. Each physical card has its own “CardID”. Many functions need this “CardID” as the input parameter.

1.2.3 Axis Indexing

Axis index is according to the “CardID” and card type. Each card type has its maximum amount of axes to be connected. For example, PCI-8372+ can be connected up to 12 axes and PCI-8366+ can be connected up to 6 axes. If the PCI-8372+ gets CardID=0, the axis indexing range is from 0 to 11. If the PCI-8372+ gets CardID=1, the axis index range is from 12 to 23. No matter if there is an axis connected, the axis index will be occupied. The CardID’s indexing order depends on the calling order of MDSP_initial() in user’s program. For example, we have 3 PCI-8372+ and 2 PCI-8366+ on the system and users place an initialization sequence as followings.

```
MDSP_initial(PCI_8372+, 0);   // CardID=0
MDSP_initial(PCI_8366+, 0);   // CardID=1
MDSP_initial(PCI_8372+, 1);   // CardID=2
MDSP_initial(PCI_8372+, 2);   // CardID=3
MDSP_initial(PCI_8366+, 1);   // CardID=4
```

Then the card index and axis index would be:

Card Type	MDSP_initial() Card Number	MDSP_initial() Card ID Return	Maximum axes per card	Axis index in this system
PCI-8372+	0	0	12	0-11
PCI-8366+	0	1	6	12-17
PCI-8372+	1	2	12	18-29
PCI-8372+	2	3	12	30-41
PCI-8366+	1	4	6	42-47

Table 1-3: Axis Index

Note: The Card number is PCI slot dependent.

1.3 System Functions

@ Name

`error_msg(ErrorNo, *ErrorMsg)` – Copy error message to buffer

`MDSP_initial(CardType,CardNo)` – Card Initialization

`MDSP_close(CardID)` – Card Close

`MDSP_reset(CardType, CardNo)` – Software reset

`config_from_file(*Filename)` – Configure card according to configuration file

`get_base_addr(CardID, *BaseAddr)` – Get the base address of this card

`get_irq_number(CardID, *IrqNo)` – Get the IRQ number of this card

`understand_mdsp(CardID, *Version, *Date)` – Get on board DSP kernel code's version information

`get_version_info(CardID, *Hardware, *Software_DLL, *Software_Driver)`

@ Description

`error_msg` :

This function copy error message of specified error number into buffer.

`MDSP_initial`:

This function is used to initialize SSCNet series card. Every application program must call this before any other functions can be used. If the initialization is executed successfully, it returns a positive value. The lower 8-bit is the 'CardID' for this card, which will be used for thereafter function calls. The upper 8-bit is the total axis number found in this board.

`MDSP_close`:

This function is used to release all resource used by this card. This function is only to be used when users' program ends. After calling

this function. The LEDs on the panel will be turn off. If the card is not closed properly, the LEDs will flash continuously. If users try to re-initial this card by MDSP_initial(), the axes could not be found.

`MDSP_reset`:

By calling this function, the will be reset just as system is powered on.

`config_from_file`:

This function is used to load the configuration file created by Motion Creator utility. This file will be saved automatically into Windows' system directory and the file name is MDSP.INI.

`get_base_addr`:

This function is used to get the card's base address.

`get_irq_number`:

This function is used to get the card's IRQ number.

`understand_mdsp`:

This function is used to get the version number and the built date on board DSP kernel firmware.

`get_version_info`:

This function is used to get the board's version information

@ Syntax

C/C++ (Windows)

```
I16 error_msg(I16 ErrorNo, char *ErrorMsg)
I16 MDSP_initial(I16 CardType, CardNo)
I16 MDSP_close(I16 Card)
I16 MDSP_reset(I16 CardType, I16 CardNo)
I16 config_from_file(char *Filename)
I16 get_base_addr(I16 Card, U16 *BaseAddr)
I16 get_irq_number(I16 Card, U16 *IrqNo)
I16 understand_mdsp(I16 Card, U32 *Version, U32
     *Date)
I16 get_version_info(I16 CardID, I32 *hardware,
     I32 *Software_DLL, I32* Software_driver)
```

Visual Basic (Windows)

```
error_msg (ByVal ErrorNo As Integer, ErrorMsg As
     Byte) As Integer
MDSP_initial (ByVal CardType As Integer, ByVal
     CardNo As Integer) As Integer
MDSP_close (ByVal Card As Integer) As Integer
MDSP_reset (ByVal CardType As Integer, ByVal Card
     As Integer) As Integer
config_from_file(ByVal Filename as String) As
     Integer
get_base_addr (ByVal Card As Integer, BaseAddr As
     Integer) As Integer
get_irq_number (ByVal Card As Integer, IrqNo As
     Integer) As Integer
understand_mdsp (ByVal Card As Integer, Version
     As Long, BuiltDate As Long) As Integer
get_version_info(ByVal CardID As Integer,
     hardware As Long, Software_DLL As Long,
     Software_driver As Long) As Integer
```

@ Arguments

`AxisNo`: AxisNo of system

`ErrorNo`: Error number

`*ErrorMsg`: Error message string

`CardID`: The SSCNet series card index number.

`*DSP`: The data structure that carry environment information of PCI-83XX Series card.

`*IrqNo`: Irq number of specified PCI-83XX Series card.

`*BaseAddr`: base address of specified PCI-83XX Series card

`*Filename`: The specified filename recording the configuration of PCI-83XX Series. This file must be created by Motion Creator of PCI-83XX Series.

`*Version`: The version number of PCI-83XX Series firmware.

`*Date`: The built date of PCI-83XX Series firmware.

`*hardware`: Hardware version of this board

`*Software_DLL`: Running DLL version

***Software_Driver**: Runnning device driver version

@ Return Code

```
ERROR_NoError
ERROR_Dll_Opened_By_Other_Process
ERROR_Card_Not_Ready
ERROR_Card_Not_Initial
ERROR_PCIBios_Not_Exist
ERROR_Card_Reinitialized
ERROR_Card_Not_Exist
ERROR_Card_Not_Accessible
ERROR_Card_Not_Ready
ERROR_DSP_Not_Ready
ERROR_DSP_Initial_Time_Out
```

Note: The return code of MDSP_initial: If the return code is equal or greater than 0, it contents the information of total axes found while initializing. The total axis number is placed in upper 8-bit. Users can get this information from the upper 8 bits of the return code. The lower 8-bit is the CardID.

@ Example

<C/C++ >

error_msg

```
char buff[255];
I16 RetCode;
RetCode = MDSP_83XX_initial(0)
error_msg(RetCode , buff);
```

MDSP_initial

```
I16 CardID;
for(int j = 0 ; j<12; j++)
        CardID = MDSP_initial(PCI_83XX, j);
```

MDSP_close

```
I16 RetCode;
RetCode = MDSP_close(0);
```

MDSP_reset

```
I16 RetCode;
RetCode = MDSP_reset(0);
```

config_from_file

```
I16 RetCode;
RetCode =
     config_from_file("c:\windows\system32\mdsp.
     ini");
```

get_base_addr

```
I16 RetCode;
U16 Addr;
RetCode = get_base_addr(0, &Addr);
```

get_irq_number

```
I16 RetCode;
U16 Irq;
RetCode = get_irq_number(0, &Irq);
```

understand_mdsp

```
I16 RetCode;
U32 Version, Date;
RetCode = understand_mdsp(0, &Version , &Date);
```

<Visual Basic>

error_msg

```
Dim i As Integer
Dim ErrorMsg(0 To 255) As Byte
Dim MsgString As String
error_msg ReturnCode, ErrorMsg(0)       ' Get
     error message
For i = 0 To 255
MsgString = MsgString + Chr(ErrorMsg(i))
If ErrorMsg(i) = 0 Then Exit For
Next i
MsgBox MsgString
```

MDSP_initial

```
Dim RetCode As Integer
Dim i As Integer
For i = 0 To 11
RetCode = MDSP_initial(PCI_83XX, j)
If RetCode <> 0 Then Exit For
Next i
```

MDSP_close

```
Dim RetCode As Integer
```

```
RetCode = MDSP_close(0)
```

MDSP_reset

```
Dim RetCode As Integer
RetCode = MDSP_reset(0)
```

config_from_file

```
Dim RetCode As Integer
Dim FileString As String
FileString = "c:\windows\system32\mdsp.ini"
RetCode = config_from_file(FileString)
```

get_base_addr

```
Dim RetCode As Integer
Dim Addr As Integer
RetCode = get_base_addr(0, Addr)
```

get_irq_number

```
Dim RetCode As Integer
Dim IrqNo As Integer
RetCode = get_irq_number(0, IrqNo)
```

understand_mdsp

```
Dim RetCode As Integer
Dim Version As Long, Date1 As Long
RetCode = understand_mdsp(0, Version, Date1)
```

1.4 Motion Functions

1.4.1 Single Motion - Single axis velocity move

@ Name

`tv_move(Axis, StrVel, MaxVel, Tacc)` – Accelerate an axis to a constant velocity with trapezoidal profile

`sv_move(Axis, StrVel, MaxVel, Tacc, Tlacc)` –Accelerate an axis to a constant velocity with S-curve profile

@ Description

`tv_move` :

This function is to accelerate an axis to the specified constant velocity with trapezoidal profile. The axis will continue to travel at a constant velocity until the velocity is changed or the axis is commanded to stop. The sign of velocity parameter determines the direction.

`sv_move`:

This function is to accelerate an axis to the specified constant velocity with S-curve profile. The axis will continue to travel at a constant velocity until the velocity is changed or the axis is commanded to stop. The sign of velocity parameter determines the direction.

@ Syntax

C/C++ (Windows)

```
I16 tv_move(I16 Axis, F64 StrVel, F64 MaxVel, F64
     Tacc)
I16 sv_move(I16 Axis, F64 StrVel, F64 MaxVel, F64
     Tacc)
```

Visual Basic (Windows)

```
tv_move (ByVal Axis As Integer, ByVal StartVel As
     Double, ByVal MaxVel As Double, ByVal Tacc
     As Double) As Integer
sv_move (ByVal Axis As Integer, ByVal StartVel As
     Double, ByVal MaxVel As Double, ByVal Tacc
     As Double, ByVal Tlacc As Double) As Integer
```

@ Arguments

`Axis`: Axis index designated to move.

`StrVel`: starting velocity in unit of mm per second

`MaxVel`: maximum velocity in unit of mm per second

`Tacc`: specified total acceleration time in unit of second

`Tlacc`: specified linear acceleration time in unit of second

@ Return Code

```
ERROR_NoError
ERROR_Dll_Opened_By_Other_Process
ERROR_Wrong_Axis_Number
ERROR_Card_Not_Ready
ERROR_Card_Not_Initial
ERROR_DSP_Not_Ready
ERROR_Axis_Not_In_Control
ERROR_Axis_Servo_Alarm
ERROR_Axis_Is_Not_Ready_ON
ERROR_Axis_Is_Not_Servo_ON
ERROR_Invalid_MaxVelocity
ERROR_Axis_Hand_Shake_Failed
ERROR_Axis_Not_Response
ERROR_Axis_Prepare_For_Motion
ERROR_Axis_Busy_For_Motion
ERROR_Axis_In_EMG_ON
```

@ Example

<C/C++ >

`tv_move`

```
I16 RetCode;
F64 StartVel=0.0;
F64 MaxVel=10.0;
F64 Tacc = 0.5;
RetCode  = tv_move(0, StartVel, MaxVel,  Tacc);
```

`sv_move`

```
I16 RetCode;
F64 StartVel=0.0;
F64 MaxVel=10.0;
```

```
F64 Tacc = 0.5;
F64 Tlacc = 0.1;
RetCode  = sv_move(0, StartVel, MaxVel,  Tacc,
     Tlacc);
```

<Visual Basic>

tv_move

```
Dim RetCode As Integer
Dim StartVel As Double, MaxVel As Double, Tacc As
     Double
StartVel = 0#
MaxVel = 10#
Tacc = 0.5
RetCode = tv_move(0, StartVel, MaxVel, Tacc)
```

sv_move

```
Dim RetCode As Integer
Dim StartVel As Double, MaxVel As Double, Tacc As
     Double
Dim Tacc As Double
StartVel = 0#
MaxVel = 10#
Tacc = 0.5
Tlacc = 0.1
RetCode = sv_move(0, StartVel, MaxVel, Tacc,
     Tlacc)
```

1.4.2 Single Motion - Single axis P to P motion

@ Name

`start_tr_move(Axis, Dist, StrVel, MaxVel, FinVel, Tacc, Tdec)` – Begin a relative trapezoidal profile move

`start_sr_move(Axis, Dist, StrVel, MaxVel, FinVel, Tacc, Tdec, Tlacc, Tldec)` – Begin a relative S-curve profile move

`start_ta_move(Axis, Pos, StrVel, MaxVel, FinVel, Tacc, Tdec)` – Begin an absolute trapezoidal profile move

`start_sa_move(Axis, Pos, StrVel, MaxVel, FinVel, Tacc, Tdec, Tlacc, Tldec)` – Begin an absolute S-curve profile move

@ Description

General: The sign of Pos or Dist parameter determines the moving direction. If the moving distance is too short to reach the specified maximum velocity, the controller will automatically lower the MaxVel, and the Tacc, Tdec, will also become shorter while the StrVel, FinVel, dV/dt (acceleration / deceleration) or d(dV/dt)/dt (jerk) keep unchanged.

`start_tr_move`:

This function causes the axis to accelerate from a starting velocity, slew at constant velocity, and decelerate to stop at the relative distance with trapezoidal profile. The acceleration and deceleration time is specified independently. It won't let the program wait for motion completion but immediately return control to the program.

`start_sr_move`:

This function causes the axis to accelerate from a starting velocity, slew at constant velocity, and decelerate to stop at the relative distance with S-curve profile. The acceleration and deceleration time is specified independently. It won't let the program wait for motion completion but immediately return control to the program.

`start_ta_move`:

This function causes the axis to accelerate from a starting velocity, slew at constant velocity, and decelerate to stop at the specified absolute position with trapezoidal profile. The acceleration and deceleration time is specified independently. It won't let the program wait for motion completion but immediately return control to the program.

`start_sa_move`:

This function causes the axis to accelerate from a starting velocity, slew at constant velocity, and decelerate to stop at the specified absolute position with S-curve profile. The acceleration and deceleration time is specified independently. It won't let the program wait for motion completion but immediately return control to the program.

@ Syntax

C/C++ (Windows)

```
I16 start_tr_move(I16 Axis, F64 Dist, F64 StrVel,
     F64 MaxVel, F64 FinVel, F64 Tacc, F64 Tdec)
I16 start_sr_move(I16 Axis, F64 Dist, F64 StrVel,
     F64 MaxVel, F64 FinVel, F64 Tacc, F64 Tdec,
     F64 Tlacc, F64 Tldec)
I16 start_ta_move(I16 Axis, F64 Pos, F64 StrVel,
     F64 MaxVel, F64 FinVel, F64 Tacc, F64 Tdec)
I16 start_sa_move(I16 Axis, F64 Pos, F64 StrVel,
     F64 MaxVel, F64 FinVel, F64 Tacc, F64 Tdec,
     F64 Tlacc, F64 Tldec)
```

Visual Basic (Windows)

```
start_tr_move (ByVal Axis As Integer, ByVal Dist
     As Double, ByVal StartVel As Double, ByVal
     MaxVel As Double, ByVal FinVel As Double,
     ByVal Tacc As Double, ByVal Tdec As Double)
     As Integer
start_sr_move (ByVal Axis As Integer, ByVal Dist
     As Double, ByVal StartVel As Double, ByVal
     MaxVel As Double, ByVal FinVel As Double,
     ByVal Tacc As Double, ByVal Tdec As Double,
     ByVal Tlacc As Double, ByVal Tldec As
     Double) As Integer
```

```
start_ta_move (ByVal Axis As Integer, ByVal pos
     As Double, ByVal StartVel As Double, ByVal
     MaxVel As Double, ByVal FinVel As Double,
     ByVal Tacc As Double, ByVal Tdec As Double)
     As Integer
start_sa_move (ByVal Axis As Integer, ByVal pos
     As Double, ByVal StartVel As Double, ByVal
     MaxVel As Double, ByVal FinVel As Double,
     ByVal Tacc As Double, ByVal Tdec As Double,
     ByVal Tlacc As Double, ByVal Tldec As
     Double) As Integer
```

@ Arguments

`Axis`: axis index designated to move or change position.

`Dist`: specified relative distance to move in unit of mm.

`Pos`: specified absolute position to move in unit of mm.

`StrVel`: starting velocity, in unit of mm per second.

`MaxVel`: maximum velocity, in unit of mm per second.

`FinVel`: final velocity, in unit of mm per second.

`Tacc`: specified total acceleration time in unit of second

`Tlacc`: specified linear acceleration time in unit of second

`Tdec`: specified deceleration time in unit of second.

`Tldec`: specified linear deceleration time in unit of second.

@ Return Code

```
ERROR_NoError
ERROR_Dll_Opened_By_Other_Process
ERROR_Wrong_Axis_Number
ERROR_Card_Not_Ready
ERROR_Card_Not_Initial
ERROR_DSP_Not_Ready
ERROR_Axis_Not_In_Control
ERROR_Axis_Servo_Alarm
ERROR_Axis_Is_Not_Ready_ON
ERROR_Axis_Is_Not_Servo_ON
ERROR_Invalid_MaxVelocity
ERROR_Axis_Hand_Shake_Failed
```

```
ERROR_Axis_Not_Response
ERROR_Axis_Prepare_For_Motion
ERROR_Axis_Busy_For_Motion
ERROR_Axis_In_EMG_ON
```

@ Example

<C/C++ >

start_tr_move

```
I16 RetCode;
F64 Dist = 20.0;
F64 StrVel=1.0;
F64 MaxVel=10.0;
F64 FinVel =0.0;
F64 Tacc = 0.5;
F64 Tdec = 0.1;
RetCode  = start_tr_move(0, Dist, StrVel, MaxVel,
     FinVel, Tacc, Tdec);
```

start_sr_move

```
I16 RetCode;
F64 Dist = 20.0;
F64 StrVel=1.0;
F64 MaxVel=10.0;
F64 FinVel =0.0;
F64 Tacc = 0.5;
F64 Tdec = 0.1;
F64 Tlacc = 0.2;
F64 Tldec = 0.0;
RetCode  = start_sr_move(0, Dist, StrVel, MaxVel,
     FinVel, Tacc, Tdec, Tlacc, Tldec);
```

start_ta_move

```
I16 RetCode;
F64 Pos = 10.0;
F64 StrVel=1.0;
F64 MaxVel=10.0;
F64 FinVel =0.0;
F64 Tacc = 0.5;
F64 Tdec = 0.1;
RetCode  = start_ta_move(0, Pos, StrVel, MaxVel,
     FinVel, Tacc, Tdec);
```

start_sa_move

```
I16 RetCode;
F64 Pos = 20.0;
F64 StrVel=1.0;
F64 MaxVel=10.0;
F64 FinVel =0.0;
F64 Tacc = 0.5;
F64 Tdec = 0.1;
F64 Tlacc = 0.2;
F64 Tldec = 0.0;
RetCode  = start_sa_move(0, Pos, StrVel, MaxVel,
     FinVel, Tacc, Tdec, Tlacc, Tldec);
```

<Visual Basic>

start_tr_move

```
Dim RetCode As Integer
Dim Dist As Double, StrVel As Double, MaxVel As
     Double
Dim FinVel As Double, Tacc As Double, Tdec As
     Double
Dist = 20#
StrVel = 1#
MaxVel = 10#
FinVel = 0#
Tacc = 0.5
Tdec = 0.1
RetCode = start_tr_move(0, Dist, StrVel, MaxVel,
     FinVel, Tacc, Tdec)
```

start_sr_move

```
Dim RetCode As Integer
Dim Dist As Double, StrVel As Double, MaxVel As
     Double
Dim FinVel As Double, Tacc As Double, Tdec As
     Double
Dim Tlacc As Double, Tldec As Double
Dist = 20#
StrVel = 1#
MaxVel = 10#
FinVel = 0#
Tacc = 0.5
Tdec = 0.1
```

```
Tlacc = 0.2
Tldec = 0#
RetCode = start_sr_move(0, Dist, StrVel, MaxVel,
     FinVel, Tacc, Tdec, Tlacc, Tldec)
```

start_ta_move

```
Dim RetCode As Integer
Dim Pos As Double, StrVel As Double, MaxVel As
     Double
Dim FinVel As Double, Tacc As Double, Tdec As
     Double
Pos = 20#
StrVel = 1#
MaxVel = 10#
FinVel = 0#
Tacc = 0.5
Tdec = 0.1
RetCode = start_ta_move(0, Pos, StrVel, MaxVel,
     FinVel, Tacc, Tdec)
```

start_sa_move

```
Dim RetCode As Integer
Dim Pos As Double, StrVel As Double, MaxVel As
     Double
Dim FinVel As Double, Tacc As Double, Tdec As
     Double
Dim Tlacc As Double, Tldec As Double
Pos = 20#
StrVel = 1#
MaxVel = 10#
FinVel = 0#
Tacc = 0.5
Tdec = 0.1
Tlacc = 0.2
Tldec = 0#
RetCode = start_sa_move(0, Pos, StrVel, MaxVel,
     FinVel, Tacc, Tdec, Tlacc, Tldec)
```

1.4.3 Single Motion - Multi axes velocity move

@ Name

`tv_move_all(Length, *Axis, *StrVel, *MaxVel, *Tacc)` – Accelerate all specified axes to an individually specified velocity with trapezoidal profile

`sv_move_all(Length, *Axis, *StrVel, *MaxVel, *Tacc, *Tlacc)` – Accelerate all specified axes to an individually specified velocity with S-curve profile

@ Description

`tv_move_all`:

The tv_move_all is a multi-axis-version tv_move. That is tv_move_all causes all specified axes to do tv_move. The only constrain is that all axes specified in the *Axis array must be of the same card. Each axis defines its velocity and acceleration time independently. Axes will keep moving until tv_change, sv_change, sv_stop, tv_stop, or emg_stop command is applied.

Note: tv_move_all only guarantees all specified axes start moving at the same time. These axes are not necessary to stop at the same time, in other words, user must apply tv_stop, sv_stop command to every axis.

`sv_move_all`:

The sv_move_all is a multi-axis-version sv_move. Refer to the description of tv_move_all above.

@ Syntax

C/C++ (Windows)

```
I16 tv_move_all(I16 Length, I16 *Axis, F64
     *StrVel, F64 *MaxVel, F64 *Tacc)
I16 sv_move_all(I16 Length, I16 *Axis, F64
     *StrVel, F64 *MaxVel, F64 *Tacc, F64 *Tlacc)
```

Visual Basic (Windows)

```
tv_move_all (ByVal length As Integer, AxisArray
     As Integer, StartVelArray As Double,
     MaxVelArray As Double, TaccArray As Double)
     As Integer
```

```
sv_move_all (ByVal length As Integer, AxisArray
     As Integer, StartVelArray As Double,
     MaxVelArray As Double, TaccArray As Double,
     TlaccArray As Double) As Integer
```

@ Arguments

Length: the total number of axis to apply move.

***Axis**: array of axis index designated to move.

***StrVel**: array of starting velocity in unit of mm per second

***MaxVel**: array of maximum velocity in unit of mm per second

***Tacc**: array of specified acceleration time in unit of second

***Tlacc**: array of specified linear acceleration time in unit of second

@ Return Code

```
ERROR_NoError
ERROR_Dll_Opened_By_Other_Process
ERROR_Wrong_Axis_Number
ERROR_Invalid_Length_Of_Axes
ERROR_Card_Not_Ready
ERROR_Card_Not_Initial
ERROR_DSP_Not_Ready
ERROR_Axis_Not_In_Control
ERROR_Axis_Servo_Alarm
ERROR_Axis_Is_Not_Ready_ON
ERROR_Axis_Is_Not_Servo_ON
ERROR_Invalid_MaxVelocity
ERROR_Axis_Hand_Shake_Failed
ERROR_Axis_Not_Response
ERROR_Axis_Prepare_For_Motion
ERROR_Axis_Busy_For_Motion
ERROR_Axis_In_EMG_ON
```

@ Example

<C/C++ >

tv_move_all

```
I16 RetCode;
```

```
I16 Length = 3;
I16 Axis[3] = {0, 1, 2};
F64 StrVel[3]= {0.0, 0.0, 0.0};
F64 MaxVel[3]= {10.0, 6.0, 5.0};
F64 Tacc[3] = {0.1 ,0.1, 0.2};
RetCode  = tv_move_all(Length, Axis, StrVel,
     MaxVel, Tacc);
```

sv_move_all

```
I16 RetCode;
I16 Length = 3;
I16 Axis[3] = {0, 1, 2};
F64 StrVel[3]= {0.0, 0.0, 0.0};
F64 MaxVel[3]= {10.0, 6.0, 5.0};
F64 Tacc[3] = {0.1 ,0.1, 0.2};
F64 Tlacc[3] = {0.1 ,0.05, 0.0};
RetCode  = sv_move_all(Length, Axis, StrVel,
     MaxVel, Tacc, Tlacc);
```

<Visual Basic>

tv_move_all

```
Dim RetCode As Integer
Dim Length As Integer
Dim Axis(0 To 2) As Integer, StrVel(0 To 2) As
     Double
Dim MaxVel(0 To 2) As Double, Tacc(0 To 2) As
     Double
For Length = 0 To 2
    Axis(Length) = Length
    StrVel(Length) = 0#
MaxVel(Length) = Length * 10
Tacc(Length) = Length * 0.1 + 0.1
Next Length
Length = 3
RetCode = tv_move_all(Length, Axis(0), StrVel(0),
     MaxVel(0), Tacc(0))
```

sv_move_all

```
Dim RetCode As Integer
Dim Length As Integer
Dim Axis(0 To 2) As Integer, StrVel(0 To 2) As
     Double
```

```
Dim MaxVel(0 To 2) As Double, Tacc(0 To 2) As
     Double
Dim Tlacc(0 To 2) As Double
For Length = 0 To 2
    Axis(Length) = Length
    StrVel(Length) = 0#
MaxVel(Length) = Length * 10
Tacc(Length) = Length * 0.1 + 0.1
Tlacc(Length) = Length * 0.1
Next Length
Length = 3
RetCode = sv_move_all(Length, Axis(0), StrVel(0),
     MaxVel(0), Tacc(0), Tlacc(0))
```

1.4.4 Single Motion - Multi axes P to P motion

@ Name

`start_tr_move_all(Length, *Axis, *Dist, *StrVel, *MaxVel, *FinVel, *Tacc, *Tdec)` – cause all specified axes to do relative trapezoidal profile move

`start_sr_move_all(Length, *Axis, *Dist, *StrVel, *MaxVel, *FinVel, *Tacc, *Tdec, *Tlacc, *Tldec)` – cause all specified axes to do relative S-curve profile move

`start_ta_move_all(Length, *Axis, *Pos, *StrVel, *MaxVel, *FinVel, *Tacc, *Tdec)` – cause all specified axes to do absolute trapezoidal profile move

`start_sa_move_all(Length, *Axis, *Pos, *StrVel, *MaxVel, *FinVel, *Tacc, *Tdec, *Tlacc, *Tldec)` – cause all specified axes to do absolute S-curve profile move

@ Description

`start_tr_move_all`:

This function causes all specified axes to perform srart_tr_move. Every axis must define its own parameters,ie, StrVel, MaxVel, FinVel, Tacc, Tdec. Note: start_tr_move_all only guarantees all specified axes start moving at the same time. These axes are not necessary to stop at the same time. The duration of motion of each axis is independently defined.

`start_sr_move_all`:

Refer to description of start_tr_move_all above.

`start_ta_move_all`:

Refer to description of start_tr_move_all above.

`start_sa_move_all`:

Refer to description of start_tr_move_all above.

@ Syntax

C/C++ (Windows)

```
I16 start_tr_move_all(I16 Length, I16 *Axis, F64
     *Dist, F64 *StrVel, F64 *MaxVel, F64
     *FinVel, F64 *Tacc, F64 *Tdec)
I16 start_sr_move_all(I16 Length, I16 *Axis, F64
     *Dist, F64 *StrVel, F64 *MaxVel, F64
     *FinVel, F64 *Tacc, F64 *Tdec, F64 *Tlacc,
     F64 *Tldec)
I16 start_ta_move_all(I16 Length, I16 *Axis, F64
     *Pos, F64 *StrVel, F64 *MaxVel, F64 *FinVel,
     F64 *Tacc, F64 *Tdec)
I16 start_sa_move_all(I16 Length,I16 *Axis, F64
     *Pos, F64 *StrVel, F64 *MaxVel,F64 *FinVel,
     F64 *Tacc, F64 *Tdec, F64 *Tlacc, F64
     *Tldec)
```

Visual Basic (Windows)

```
start_tr_move_all (ByVal length As Integer,
     AxisArray As Integer, DistArray As Double,
     StartVelArray As Double, MaxVelArray As
     Double, FinVelArray As Double, TaccArray As
     Double, TdecArray As Double) As Integer
start_sr_move_all (ByVal length As Integer,
     AxisArray As Integer, DistArray As Double,
     StartVelArray As Double, MaxVelArray As
     Double, FinVelArray As Double, TaccArray As
     Double, TdecArray As Double, TlaccArray As
     Double, TldecArray As Double) As Integer
start_ta_move_all (ByVal length As Integer,
     AxisArray As Integer, posarray As Double,
     StartVelArray As Double, MaxVelArray As
     Double, FinVelArray As Double, TaccArray As
     Double, TdecArray As Double) As Integer
start_sa_move_all (ByVal length As Integer,
     AxisArray As Integer, posarray As Double,
     StartVelArray As Double, MaxVelArray As
     Double, FinVelArray As Double, TaccArray As
     Double, TdecArray As Double, TlaccArray As
     Double, TldecArray As Double) As Integer
```

@ Arguments

Length: the total number of axis to apply move.

***Axis**: array of axis index designated to move.

***Dist**: array of specified relative distance to move in unit of mm.

***Pos**: array of specified absolute position to move in unit of mm.

***StrVel**: array of starting velocity, in unit of mm per second.

***MaxVel**: array of maximum velocity, in unit of mm per second.

***FinVel**: array of final velocity, in unit of mm per second.

***Tacc**: array of specified acceleration time in unit of second.

***Tdec**: array of specified deceleration time in unit of second.

***Tlacc**: array of specified linear acceleration time in unit of second.

***Tldec**: array of specified linear deceleration time in unit of second.

@ Return Code

```
ERROR_NoError
ERROR_Dll_Opened_By_Other_Process
ERROR_Wrong_Axis_Number
ERROR_Invalid_Length_Of_Axes
ERROR_Card_Not_Ready
ERROR_Card_Not_Initial
ERROR_DSP_Not_Ready
ERROR_Axis_Not_In_Control
ERROR_Axis_Servo_Alarm
ERROR_Axis_Is_Not_Ready_ON
ERROR_Axis_Is_Not_Servo_ON
ERROR_Invalid_MaxVelocity
ERROR_Axis_Hand_Shake_Failed
ERROR_Axis_Not_Response
ERROR_Axis_Prepare_For_Motion
ERROR_Axis_Busy_For_Motion
ERROR_Axis_In_EMG_ON
```

@ Example

<C/C++ >

start_tr_move_all

```
I16 RetCode;
I16 Length = 2;
I16 Axis[2] = {0, 2};
F64 Dist[2]= {20.0, 20.0};
F64 StrVel[2]= {0.0, 0.0};
F64 MaxVel[2]= {10.0, 5.0};
F64 FinVel[2]= {0.0, 0.0};
F64 Tacc[2] = {0.2 ,0.1};
F64 Tdec[2] = {0.1 ,0.2};
RetCode  = start_tr_move_all(Length, Axis, Dist,
     StrVel, MaxVel, FinVel, Tacc, Tdec);
```

start_sr_move_all

```
I16 RetCode;
I16 Length = 2;
I16 Axis[2] = {0, 2};
F64 Dist[2]= {20.0, 20.0};
F64 StrVel[2]= {0.0, 0.0};
F64 MaxVel[2]= {10.0, 5.0};
F64 FinVel[2]= {0.0, 0.0};
F64 Tacc[2] = {0.2 ,0.1};
F64 Tdec[2] = {0.1 ,0.2};
F64 Tlacc[2] = {0.1 ,0.0};
F64 Tldec[2] = {0.0 ,0.2};
RetCode  = start_sr_move_all(Length, Axis, Dist,
     StrVel, MaxVel, FinVel, Tacc, Tdec, Tlacc,
     Tldec);
```

start_ta_move_all

```
I16 RetCode;
I16 Length = 2;
I16 Axis[2] = {0, 2};
F64 Pos[2]= {50.0, 40.0};
F64 StrVel[2]= {0.0, 0.0};
F64 MaxVel[2]= {10.0, 5.0};
F64 FinVel[2]= {0.0, 0.0};
F64 Tacc[2] = {0.2 ,0.1};
F64 Tdec[2] = {0.1 ,0.2};
RetCode  = start_ta_move_all(Length, Axis, Pos,
     StrVel, MaxVel, FinVel, Tacc, Tdec);
```

start_sa_move_all

```
I16 RetCode;
I16 Length = 2;
I16 Axis[2] = {0, 2};
F64 Pos[2]= {50.0, 30.0};
F64 StrVel[2]= {0.0, 0.0};
F64 MaxVel[2]= {10.0, 5.0};
F64 FinVel[2]= {0.0, 0.0};
F64 Tacc[2] = {0.2 ,0.1};
F64 Tdec[2] = {0.1 ,0.2};
F64 Tlacc[2] = {0.1 ,0.0};
F64 Tldec[2] = {0.0 ,0.2};
RetCode  = start_sa_move_all(Length, Axis, Pos,
     StrVel, MaxVel, FinVel, Tacc, Tdec, Tlacc,
     Tldec);
```

<Visual Basic>

start_tr_move_all

```
Dim RetCode As Integer
Dim Length As Integer
Dim Axis(0 To 1) As Integer, Dist(0 To 1) As
     Double
Dim StrVel(0 To 1) As Double, MaxVel(0 To 1) As
     Double
Dim FinVel(0 To 1) As Double
Dim Tacc(0 To 1) As Double, Tdec(0 To 1) as
     Double
For Length = 0 To 1
    Axis(Length) = Length
Dist(Length) = Length*10
StrVel(Length) = 0#
MaxVel(Length) = Length * 10
FinVel(Length) = 0#
Tacc(Length) = Length * 0.1 + 0.1
Tdec(Length) = Length * 0.2 + 0.1
Next Length
Length = 2
RetCode = start_tr_move_all(Length, Axis(0),
     Dist(0), StrVel(0), MaxVel(0), FinVel(0),
     Tacc(0), Tdec(0))
```

start_sr_move_all

```
Dim RetCode As Integer
Dim Length As Integer
```

```
Dim Axis(0 To 1) As Integer, Dist(0 To 1) As
     Double
Dim StrVel(0 To 1) As Double, MaxVel(0 To 1) As
     Double
Dim FinVel(0 To 1) As Double
Dim Tacc(0 To 1) As Double, Tdec(0 To 1) as
     Double
Dim Tlacc(0 To 1) As Double, Tldec(0 To 1) as
     Double
For Length = 0 To 1
    Axis(Length) = Length
Dist(Length) = Length*10
StrVel(Length) = 0#
MaxVel(Length) = Length * 10
FinVel(Length) = 0#
Tacc(Length) = Length * 0.1 + 0.1
Tdec(Length) = Length * 0.2 + 0.1
Tlacc(Length) = 0#
Tldec(Length) = Length * 0.2 + 0.1
Next Length
Length = 2
RetCode = start_sr_move_all(Length, Axis(0),
     Dist(0), StrVel(0), MaxVel(0), FinVel(0),
     Tacc(0), Tdec(0), Tlacc(0), Tldec(0))
```

start_ta_move_all

```
Dim RetCode As Integer
Dim Length As Integer
Dim Axis(0 To 1) As Integer, Pos(0 To 1) As
     Double
Dim StrVel(0 To 1) As Double, MaxVel(0 To 1) As
     Double
Dim FinVel(0 To 1) As Double
Dim Tacc(0 To 1) As Double, Tdec(0 To 1) as
     Double
For Length = 0 To 1
    Axis(Length) = Length
Pos(Length) = Length*10
StrVel(Length) = 0#
MaxVel(Length) = Length * 10
FinVel(Length) = 0#
Tacc(Length) = Length * 0.1 + 0.1
Tdec(Length) = Length * 0.2 + 0.1
```

```
Next Length
Length = 2
RetCode = start_ta_move_all(Length, Axis(0),
     Pos(0), StrVel(0), MaxVel(0), FinVel(0),
     Tacc(0), Tdec(0))
```

start_sa_move_all

```
Dim RetCode As Integer
Dim Length As Integer
Dim Axis(0 To 1) As Integer, Pos(0 To 1) As
     Double
Dim StrVel(0 To 1) As Double, MaxVel(0 To 1) As
     Double
Dim FinVel(0 To 1) As Double
Dim Tacc(0 To 1) As Double, Tdec(0 To 1) as
     Double
Dim Tlacc(0 To 1) As Double, Tldec(0 To 1) as
     Double
For Length = 0 To 1
    Axis(Length) = Length
Pos(Length) = Length*10
StrVel(Length) = 0#
MaxVel(Length) = Length * 10
FinVel(Length) = 0#
Tacc(Length) = Length * 0.1 + 0.1
Tdec(Length) = Length * 0.2 + 0.1
Tlacc(Length) = 0#
Tldec(Length) = Length * 0.2 + 0.1
Next Length
Length = 2
RetCode = start_sa_move_all(Length, Axis(0),
     Pos(0), StrVel(0), MaxVel(0), FinVel(0),
     Tacc(0), Tdec(0), Tlacc(0), Tldec(0))
```

1.4.5 Single Motion - Speed Change on the fly

@ Name

`tv_change(Axis, SpeedFactor, Tacc)` – Change velocity on the fly with trapezoidal profile

`sv_change(Axis, SpeedFactor, Tacc)` – Change position on the fly with S-curve profile

`tv_stop(Axis, Tdec)` – Stop axis with trapezoidal profile

`sv_stop(Axis, Tdec)` – Stop axis with S-curve trapezoidal profile

`emg_stop(Axis)` – immediately stop axis.

@ Description

`tv_change`:

This function is used to change axis velocity on the fly with trapezoidal profile. Note: the new velocity is defined by giving a multiplier, “SpeedFactor”, to current maximum velocity.

For example:

A tv_move(0, 1.0, 20.0, 0.5) command causes axis 0 accelerate from speed 1.0 mm/sec to 20.0 mm/sec in 0.5 sec. When it slew at 20.0mm/sec, tv_change(0, 2.0, 1.0) is executed. It cause axis 0 start accelerating with trapezoidal profile to 40.0 mm/ser in 1.0 sec.

`sv_change`:

This function is used to change axis velocity on the fly with S-curve profile. Note: the new velocity is defined by giving a multiplier, “SpeedFactor”, to current maximum velocity.

For example:

A sv_move(0, 1.0, 20.0, 0.5) command causes axis 0 accelerate from speed 1.0 mm/sec to 20.0 mm/sec in 0.5 sec. When it slew at 20.0mm/sec, sv_change(0, 2.0, 1.0) is executed. It cause axis 0 start accelerating with S-curve profile to 40.0 mm/sec in 1.0 sec.

`tv_stop`, **`sv_stop`**:

These functions will stop the specified axis. The deceleration time is defined by Tdec.

emg_stop:

This function will stop specified axis immediately.

@ Syntax

C/C++ (DOS, Windows)

```
I16 tv_change(I16 Axis, F64 SpeedFactor, F64
     Tacc)
I16 sv_change(I16 Axis, F64 SpeedFactor, F64
     Tacc)
I16 tv_stop(I16 Axis, F64 Tdec)
I16 sv_stop(I16 Axis, F64 Tdec)
I16 emg_stop(I16 Axis)
```

Visual Basic (Windows)

```
tv_change (ByVal Axis As Integer, ByVal
     SpeedFactor As Single, ByVal Tdec As Double)
     As Integer
sv_change (ByVal Axis As Integer, ByVal
     SpeedFactor As Single, ByVal Tdec As Double)
     As Integer
tv_stop (ByVal Axis As Integer, ByVal Tdec As
     Double) As Integer
sv_stop (ByVal Axis As Integer, ByVal Tdec As
     Double) As Integer
emg_stop  (ByVal Axis As Integer) As Integer
```

@ Arguments

Axis: axis index designated to move

SpeedFactor: This parameter is used to define new max velocity. The new max velocity is equal to old max velocity multiply SpeedFactor.

Tacc: specified acceleration time in unit of second.

Tdec: specified deceleration time in unit of second.

@ Return Code

```
ERROR_NoError
ERROR_Dll_Opened_By_Other_Process
```

```
ERROR_Wrong_Axis_Number
ERROR_Card_Not_Ready
ERROR_Card_Not_Initial
ERROR_DSP_Not_Ready
ERROR_Axis_Not_In_Control
ERROR_Axis_Servo_Alarm
ERROR_Axis_Is_Not_Ready_ON
ERROR_Axis_Is_Not_Servo_ON
ERROR_Axis_Is_Not_In_Motion
ERROR_Axis_Is_In_V_Change
ERROR_Can_Not_Apply_P_Change
```

@ Example

<C/C++ >

tv_change

```
I16 RetCode;
F64 StartVel=0.0;
F64 MaxVel=10.0;
F64 Tacc = 0.5;
RetCode  = tv_move(0, StartVel, MaxVel,  Tacc);
Sleep(1000);
F64 NewVel = 15.0;
F64 SpeedFactor = NewVel / MaxVel;
Tacc = 0.1;
RetCode  = tv_change(0, SpeedFactor, Tacc);
```

sv_change

```
I16 RetCode;
F64 StartVel=0.0;
F64 MaxVel=10.0;
F64 Tacc = 0.5;
RetCode  = tv_move(0, StartVel, MaxVel,  Tacc);
Sleep(1000);
F64 NewVel = 15.0;
F64 SpeedFactor = NewVel / MaxVel;
Tacc = 0.1;
RetCode  = sv_change(0, SpeedFactor, Tacc);
```

tv_stop

```
I16 RetCode;
F64 Pos = 20.0;
F64 StrVel=1.0;
```

```
F64 MaxVel=10.0;
F64 FinVel =0.0;
F64 Tacc = 0.5;
F64 Tdec = 0.1;
F64 Tlacc = 0.2;
F64 Tldec = 0.0;
RetCode  = start_sa_move(0, Pos, StrVel, MaxVel,
     FinVel, Tacc, Tdec, Tlacc, Tldec);
Sleep(1000);
Tdec = 0.3;
RetCode  = tv_stop(0, Tdec);
```

sv_stop

```
I16 RetCode;
F64 StartVel=0.0;
F64 MaxVel=10.0;
F64 Tacc = 0.5;
RetCode  = tv_move(0, StartVel, MaxVel,  Tacc);
Sleep(1000);
F64 Tdec = 1.0;
RetCode  = sv_stop(0, Tdec);
```

emg_stop

```
I16 RetCode;
F64 Pos = 20.0;
F64 StrVel=1.0;
F64 MaxVel=10.0;
F64 FinVel =0.0;
F64 Tacc = 0.5;
F64 Tdec = 0.1;
F64 Tlacc = 0.2;
F64 Tldec = 0.0;
RetCode  = start_sa_move(0, Pos, StrVel, MaxVel,
     FinVel, Tacc, Tdec, Tlacc, Tldec);
Sleep(1000);
RetCode  = emg_stop(0);
```

<Visual Basic>

tv_change

```
Dim RetCode As Integer
Dim StartVel As Double, MaxVel As Double, Tacc As
     Double
StartVel = 0#
```

```
MaxVel = 10#
Tacc = 0.5
RetCode = tv_move(0, StartVel, MaxVel, Tacc)
Sleep (1000)
Dim NewVel As Double
NewVel = 15#
Dim SpeedFactor As Double
SpeedFactor = NewVel / MaxVel
Tacc = 0.1
RetCode = tv_change(0, SpeedFactor, Tacc)
```

sv_change

```
Dim RetCode As Integer
Dim StartVel As Double, MaxVel As Double, Tacc As
     Double
StartVel = 0#
MaxVel = 10#
Tacc = 0.5
RetCode = tv_move(0, StartVel, MaxVel, Tacc)
Sleep (1000)
Dim NewVel As Double
NewVel = 15#
Dim SpeedFactor As Double
SpeedFactor = NewVel / MaxVel
Tacc = 0.1
RetCode = sv_change(0, SpeedFactor, Tacc)
```

tv_stop

```
Dim RetCode As Integer
Dim Pos As Double, StrVel As Double, MaxVel As
     Double
Dim FinVel As Double, Tacc As Double, Tdec As
     Double
Dim Tlacc As Double, Tldec As Double
Pos = 20#
StrVel = 1#
MaxVel = 10#
FinVel = 0#
Tacc = 0.5
Tdec = 0.1
Tlacc = 0.2
Tldec = 0#
```

```
RetCode = start_sa_move(0, Pos, StrVel, MaxVel,
     FinVel, Tacc, Tdec, Tlacc, Tldec)
Sleep (1000)
Tdec = 0.3
RetCode = tv_stop(0, Tdec)
```

sv_stop

```
Dim RetCode As Integer
Dim StartVel As Double, MaxVel As Double, Tacc As
     Double
StartVel = 0#
MaxVel = 10#
Tacc = 0.5
RetCode = tv_move(0, StartVel, MaxVel, Tacc)
Sleep (1000)
F64 Tdec = 1#
RetCode = sv_stop(0, Tdec)
```

emg_stop

```
Dim RetCode As Integer
Dim Pos As Double, StrVel As Double, MaxVel As
     Double
Dim FinVel As Double, Tacc As Double, Tdec As
     Double
Dim Tlacc As Double, Tldec As Double
Pos = 20#
StrVel = 1#
MaxVel = 10#
FinVel = 0#
Tacc = 0.5
Tdec = 0.1
Tlacc = 0.2
Tldec = 0#
RetCode = start_sa_move(0, Pos, StrVel, MaxVel,
     FinVel, Tacc, Tdec, Tlacc, Tldec)
Sleep (1000)
RetCode = emg_stop(0)
```

1.4.6 Single Motion - Multi axes linear interpolation

@ Name

`start_line_tr_move(Length, *AxisArray, *DistArray, StrVel, MaxVel, FinVel, Tacc, Tdec)` – Begin a relative n-axis linear interpolation, with trapezoidal profile

`start_line_sr_move(Length, *AxisArray, *DistArray, StrVel, MaxVel, FinVel, Tacc, Tdec, Tlacc, Tldec)` – Begin a relative n-axis linear interpolation, with S-curve profile

`start_line_ta_move(Length, *AxisArray, *PosArray, StrVel, MaxVel, FinVel, Tacc, Tdec)` –Begin a absolute n-axis linear interpolation, with trapezoidal profile

`start_line_sa_move(Length, *AxisArray, *PosArray, StrVel, MaxVel, FinVel, Tacc, Tdec, Tlacc, Tldec)` – Begin a absolute n-axis linear interpolation, with S-curve profile

@ Description

`start_line_tr_move`:

This function performs a relative n-axis linear interpolation with trapezoidal profile. Interpolation means all axes start out at the same time and arrive destination at the same time. The specified start, max and final velocity are tangential velocity.

Note: All axes must be of the same card.

`start_line_sr_move`:

This function performs a relative n-axis linear interpolation with S-curve profile. Interpolation means all axes start out at the same time and arrive destination at the same time. The start, max and final velocity are the tangential velocity.

Note: All axes must be of the same card.

`start_line_ta_move`:

This function performs a absolute n-axis linear interpolation with trapezoidal profile. Interpolation means all axes start out at the

same time and arrive destination at the same time. The start, max and final velocity are the tangential velocity.

Note: All axes must be of the same card.

`start_line_sa_move`:

This function performs a absolute n-axis linear interpolation with S-curve profile. Interpolation means all axes start out at the same time and arrive destination at the same time. The start, max and final velocity are the tangential velocity.

Note: All axes must be of the same card.

@ Syntax

C/C++ (DOS, Windows)

```
I16 start_line_tr_move(I16 Length, I16
     *AxisArray, F64 *DistArray, F64 StrVel, F64
     MaxVel, F64 FinVel, F64 Tacc, F64 Tdec)
I16 start_line_sr_move(I16 Length, I16
     *AxisArray, F64 *DistArray, F64 StrVel, F64
     MaxVel, F64 FinVel, F64 Tacc, F64 Tdec, F64
     Tlacc, F64 Tldec)
I16 start_line_ta_move(I16 Length, I16
     *AxisArray, F64 *PosArray, F64 StrVel, F64
     MaxVel, F64 FinVel, F64 Tacc, F64 Tdec)
I16 start_line_sa_move(I16 Length, I16
     *AxisArray, F64 *PosArray, F64  StrVel, F64
     MaxVel, F64 FinVel, F64 Tacc, F64 Tdec, F64
     Tlacc, F64 Tldec)
```

Visual Basic (Windows)

```
start_line_tr_move (ByVal length As Integer,
     AxisArray As Integer, DistArray As Double,
     ByVal StartVel As Double, ByVal MaxVel As
     Double, ByVal FinVel As Double, ByVal Tacc
     As Double, ByVal Tdec As Double) As Integer
start_line_sr_move (ByVal length As Integer,
     AxisArray As Integer, DistArray As Double,
     ByVal StartVel As Double, ByVal MaxVel As
     Double, ByVal FinVel As Double, ByVal Tacc
     As Double, ByVal Tdec As Double, ByVal Tlacc
     As Double, ByVal Tldec As Double) As Integer
start_line_ta_move (ByVal length As Integer,
     AxisArray As Integer, PosArray As Double,
```

```
        ByVal StartVel As Double, ByVal MaxVel As
        Double, ByVal FinVel As Double, ByVal Tacc
        As Double, ByVal Tdec As Double) As Integer
start_line_sa_move (ByVal length As Integer,
        AxisArray As Integer, PosArray As Double,
        ByVal StartVel As Double, ByVal MaxVel As
        Double, ByVal FinVel As Double, ByVal Tacc
        As Double, ByVal Tdec As Double, ByVal Tlacc
        As Double, ByVal Tldec As Double) As Integer
```

@ Arguments

Length: the total number of axis to apply move.

***AxisArray**: array of axis number designated to move.

***DistArray**: array of specified relative distance to move in unit of mm.

***PosArray**: array of specified absolute position to move in unit of mm.

StrVel: starting tangential velocity, in unit of mm per second.

MaxVel: maximum tangential velocity, in unit of mm per second.

FinVel: final tangential velocity, in unit of mm per second.

Tacc: specified acceleration time in unit of second.

Tdec: specified deceleration time in unit of second.

Tlacc: specified linear acceleration time in unit of second.

Tldec: specified linear deceleration time in unit of second.

@ Return Code

```
ERROR_NoError
ERROR_Dll_Opened_By_Other_Process
ERROR_Wrong_Axis_Number
ERROR_Invalid_Length_Of_Axes
ERROR_Card_Not_Ready
ERROR_Card_Not_Initial
ERROR_DSP_Not_Ready
ERROR_Axis_Not_In_Control
ERROR_Axis_Servo_Alarm
ERROR_Axis_Is_Not_Ready_ON
ERROR_Axis_Is_Not_Servo_ON
```

```
ERROR_Invalid_MaxVelocity
ERROR_Axis_Hand_Shake_Failed
ERROR_Axis_Not_Response
ERROR_Axis_Prepare_For_Motion
ERROR_Axis_Busy_For_Motion
ERROR_Axis_In_EMG_ON
```

@ Example

<C/C++ >

start_line_tr_move

```
I16 RetCode;
I16 Length = 2;
I16 Axis[2] = {0, 2};
F64 Dist[2]= {10.0, 20.0};
F64 StrVel= 0.0;
F64 MaxVel= 15.0;
F64 FinVel= 0.0;
F64 Tacc = 0.1;
F64 Tdec = 0.2;
RetCode  = start_line_tr_move(Length, Axis, Dist,
     StrVel, MaxVel, FinVel, Tacc, Tdec);
```

start_line_sr_move

```
I16 RetCode;
I16 Length = 2;
I16 Axis[2] = {0, 2};
F64 Dist[2]= {20.0, 20.0};
F64 StrVel= 0.0;
F64 MaxVel= 5.0;
F64 FinVel= 0.0;
F64 Tacc = 0.3
F64 Tdec = 0.2
F64 Tlacc = 0.3;
F64 Tldec = 0.0;
RetCode  = start_line_sr_move(Length, Axis, Dist,
     StrVel, MaxVel, FinVel, Tacc, Tdec, Tlacc,
     Tldec);
```

start_line_ta_move

```
I16 RetCode;
I16 Length = 2;
I16 Axis[2] = {0, 2};
```

```
F64 Pos[2]= {50.0, 40.0};
F64 StrVel= 0.0;
F64 MaxVel= 25.0;
F64 FinVel= 0.0;
F64 Tacc = 0.2;
F64 Tdec = 0.1;
RetCode  = start_line_ta_move(Length, Axis, Pos,
     StrVel, MaxVel, FinVel, Tacc, Tdec);
```

start_line_sa_move

```
I16 RetCode;
I16 Length = 2;
I16 Axis[2] = {0, 2};
F64 Pos[2]= {50.0, 30.0};
F64 StrVel= 0.0;
F64 MaxVel= 25;
F64 FinVel= 0.0;
F64 Tacc = 0.2;
F64 Tdec = 0.2;
F64 Tlacc = 0.1;
F64 Tldec = 0.0;
RetCode  = start_line_sa_move(Length, Axis, Pos,
     StrVel, MaxVel, FinVel, Tacc, Tdec, Tlacc,
     Tldec);
```

<Visual Basic>

start_line_tr_move

```
Dim RetCode As Integer
Dim Length As Integer
Dim Axis(0 To 1) As Integer, Dist(0 To 1) As
     Double
Dim StrVel As Double, MaxVel As Double, FinVel As
     Double,
Dim Tacc As Double, Tdec as Double
For Length = 0 To 1
    Axis(Length) = Length
Dist(Length) = Length*10+5
Next Length
StrVel = 0#
MaxVel = 10
FinVel = 0#
Tacc = 0.1
Tdec = 0.2
```

```
Length = 2
RetCode = start_line_tr_move(Length, Axis(0),
     Dist(0), StrVel, MaxVel, FinVel, Tacc, Tdec)
```

start_line_sr_move

```
Dim RetCode As Integer
Dim Length As Integer
Dim Axis(0 To 1) As Integer, Dist(0 To 1) As
     Double
Dim StrVel As Double, MaxVel As Double, FinVel As
     Double,
Dim Tacc As Double, Tdec as Double
Dim Tlacc As Double, Tldec as Double
For Length = 0 To 1
    Axis(Length) = Length
Dist(Length) = Length*10+5
Next Length
StrVel = 0#
MaxVel = 10
FinVel = 0#
Tacc = 0.1
Tdec = 0.2
Length = 2
RetCode = start_line_sr_move(Length, Axis(0),
     Dist(0), StrVel, MaxVel, FinVel, Tacc, Tdec,
     Tlacc, Tldec)
```

start_line_ta_move

```
Dim RetCode As Integer
Dim Length As Integer
Dim Axis(0 To 1) As Integer, Pos(0 To 1) As
     Double
Dim StrVel As Double, MaxVel As Double, FinVel As
     Double,
Dim Tacc As Double, Tdec as Double
For Length = 0 To 1
    Axis(Length) = Length
Pos(Length) = Length*10+5
Next Length
StrVel = 0#
MaxVel = 10
FinVel = 0#
Tacc = 0.1
```

```
Tdec = 0.2
Length = 2
RetCode = start_line_ta_move(Length, Axis(0),
     Pos(0), StrVel, MaxVel, FinVel, Tacc, Tdec)
```

start_line_sa_move

```
Dim RetCode As Integer
Dim Length As Integer
Dim Axis(0 To 1) As Integer, Pos(0 To 1) As
     Double
Dim StrVel As Double, MaxVel As Double, FinVel As
     Double,
Dim Tacc As Double, Tdec as Double
Dim Tlacc As Double, Tldec as Double
For Length = 0 To 1
    Axis(Length) = Length
Pos(Length) = Length*10+5
Next Length
StrVel = 0#
MaxVel = 10
FinVel = 0#
Tacc = 0.1
Tdec = 0.2
Tlacc = 0.1
Tldec = 0.2
Length = 2
RetCode = start_line_sa_move(Length, Axis(0),
     Pos(0), StrVel, MaxVel, FinVel, Tacc, Tdec,
     Tlacc, Tldec)
```

1.4.7 Single Motion - Multi axes circular interpolation

@ Name

`start_arc_tr_move(*AxisArray, *CenterArray, Angle, StrVel, MaxVel, FinVel, Tacc, Tdec)` –Begin a relative 2-axis circular interpolation, with trapezoidal profile

`start_arc_sr_move(*AxisArray, *CenterArray, Angle, StrVel, MaxVel, FinVel, Tacc, Tdec, Tlacc, Tldec)` –Begin a relative 2-axis circular interpolation, with S-curve profile

`start_arc_ta_move(*AxisArray, *CenterArray, Angle, StrVel, MaxVel, FinVel, Tacc, Tdec)` –Begin a absolute 2-axis circular interpolation, with trapezoidal profile

`start_arc_sa_move(*AxisArray, *CenterArray, Angle, StrVel, MaxVel, FinVel, Tacc, Tdec, Tlacc, Tldec)` –Begin a absolute 2-axis circular interpolation, with S-curve profile

@ Description

`start_arc_tr_move`:

This function performs a relative 2-axis circular interpolation with trapezoidal angular velocity profile. The specified start, max and final velocity are tangential velocity.

Note: The 2 axes specified in AxisArray must be of the same card.

`start_arc_sr_move`:

This function performs a relative 2-axis circular interpolation with S-curve angular velocity profile. The specified start, max and final velocity is tangential velocity.

Note: The 2 axes specified in AxisArray must be of the same card.

`start_arc_ta_move`:

This function performs a absolute 2-axis circular interpolation with trapezoidal angular velocity profile. The specified start, max and final velocity are tangential velocity.

Note: The 2 axes specified in AxisArray must be of the same card.

`start_arc_sa_move`:

This function performs absolute 2-axis circular interpolation with S-curve angular velocity profile. The specified start, max and final velocity are tangential velocity.

Note: The 2 axes specified in AxisArray must be of the same card.

@ Syntax

C/C++ (DOS, Windows)

```
I16 start_arc_tr_move(I16 *AxisArray,F64
     *CenterArray, F64 Angle, F64 StrVel, F64
     MaxVel, F64 FinVel, F64 Tacc, F64 Tdec)
I16 start_arc_sr_move(I16 *AxisArray, F64
     *CenterArray, F64 Angle, F64 StrVel, F64
     MaxVel, F64 FinVel, F64 Tacc, F64 Tdec, F64
     Tlacc, F64 Tldec)
I16 start_arc_ta_move(I16 *AxisArray, F64
     *CenterArray, F64 Angle, F64 StrVel, F64
     MaxVel, F64 FinVel, F64 Tacc, F64 Tdec)
I16 start_arc_sa_move(I16 *AxisArray, F64
     *CenterArray, F64 Angle, F64 StrVel, F64
     MaxVel, F64 FinVel, F64 Tacc, F64 Tdec, F64
     Tlacc, F64 Tldec)
```

Visual Basic (Windows)

```
start_arc_tr_move (AxisArray As Integer,
     CenterArray As Double, ByVal Angle As
     Double, ByVal StartVel As Double, ByVal
     MaxVel As Double, ByVal FinVel As Double,
     ByVal Tacc As Double, ByVal Tdec As Double)
     As Integer
start_arc_sr_move (AxisArray As Integer,
     CenterArray As Double, ByVal Angle As
     Double, ByVal StartVel As Double, ByVal
     MaxVel As Double, ByVal FinVel As Double,
     ByVal Tacc As Double, ByVal Tdec As Double,
     ByVal Tlacc As Double, ByVal Tldec As
     Double) As Integer
start_arc_ta_move (AxisArray As Integer,
     CenterArray As Double, ByVal Angle As
     Double, ByVal StartVel As Double, ByVal
     MaxVel As Double, ByVal FinVel As Double,
```

```
        ByVal Tacc As Double, ByVal Tdec As Double)
        As Integer
start_arc_sa_move (AxisArray As Integer,
        CenterArray As Double, ByVal Angle As
        Double, ByVal StartVel As Double, ByVal
        MaxVel As Double, ByVal FinVel As Double,
        ByVal Tacc As Double, ByVal Tdec As Double,
        ByVal Tlacc As Double, ByVal Tldec As
        Double) As Integer
```

@ Arguments

***AxisArray**: the number of specified 2 axes.

***CenterArray**: the coordinate of arc center in unit of mm.

Angle: specified moving angle in unit of degree.

StrVel: starting tangential velocity, in unit of mm per second.

MaxVel: maximum tangential velocity, in unit of mm per second.

FinVel: final tangential velocity, in unit of mm per second.

Tacc: specified acceleration time in unit of second.

Tdec: specified deceleration time in unit of second.

Tlacc: specified linear acceleration time in unit of second.

Tldec: specified linear deceleration time in unit of second.

@ Return Code

```
ERROR_NoError
ERROR_Dll_Opened_By_Other_Process
ERROR_Wrong_Axis_Number
ERROR_Invalid_Length_Of_Axes
ERROR_Card_Not_Ready
ERROR_Card_Not_Initial
ERROR_DSP_Not_Ready
ERROR_Axis_Not_In_Control
ERROR_Axis_Servo_Alarm
ERROR_Axis_Is_Not_Ready_ON
ERROR_Axis_Is_Not_Servo_ON
ERROR_Invalid_MaxVelocity
ERROR_Axis_Hand_Shake_Failed
ERROR_Axis_Not_Response
```

```
ERROR_Axis_Prepare_For_Motion
ERROR_Axis_Busy_For_Motion
ERROR_Axis_In_EMG_ON
```

@ Example

<C/C++ >

start_arc_tr_move

```
I16 RetCode;
I16 Axis[2] = {0, 2};
F64 CenterArray [2]= {10.0, 20.0};
F64 Angle = 90;
F64 StrVel= 0.0;
F64 MaxVel= 15.0;
F64 FinVel= 0.0;
F64 Tacc = 0.1;
F64 Tdec = 0.2;
RetCode  = start_arc_tr_move(Axis, CenterArray,
     Angle, StrVel, MaxVel, FinVel, Tacc, Tdec);
```

start_arc_sr_move

```
I16 RetCode;
I16 Axis[2] = {0, 2};
F64 CenterArray [2]= {20.0, 20.0};
F64 Angle = 180;
F64 StrVel= 0.0;
F64 MaxVel= 5.0;
F64 FinVel= 0.0;
F64 Tacc = 0.3
F64 Tdec = 0.2
F64 Tlacc = 0.3;
F64 Tldec = 0.0;
RetCode  = start_arc_sr_move(Axis, CenterArray,
     Angle, StrVel, MaxVel, FinVel, Tacc, Tdec,
     Tlacc, Tldec);
```

start_arc_ta_move

```
I16 RetCode;
I16 Axis[2] = {0, 2};
F64 CenterArray [2]= {50.0, 40.0};
F64 Angle= 270;
F64 StrVel= 0.0;
F64 MaxVel= 25.0;
```

```
F64 FinVel= 0.0;
F64 Tacc = 0.2;
F64 Tdec = 0.1;
RetCode  = start_arc_ta_move(Axis, CenterArray,
     Angle, StrVel, MaxVel, FinVel, Tacc, Tdec);
```

start_arc_sa_move

```
I16 RetCode;
I16 Axis[2] = {0, 2};
F64 CenterArray[2]= {50.0, 30.0};
F64 Angle= 360;
F64 StrVel= 0.0;
F64 MaxVel= 25;
F64 FinVel= 0.0;
F64 Tacc = 0.2;
F64 Tdec = 0.2;
F64 Tlacc = 0.1;
F64 Tldec = 0.0;
RetCode  = start_arc_sa_move(Axis, CenterArray,
     Angle, StrVel, MaxVel, FinVel, Tacc, Tdec,
     Tlacc, Tldec);
```

<Visual Basic>

start_arc_tr_move

```
Dim RetCode As Integer
Dim i As Integer
Dim Axis(0 To 1) As Integer, CenterArray (0 To 1)
     As Double
Dim Angle As Integer
Dim StrVel As Double, MaxVel As Double, FinVel As
     Double,
Dim Tacc As Double, Tdec as Double
For i = 0 To 1
    Axis(i) = i
CenterArray (i) = i*10+5
Next I
Angle = 90
StrVel = 0#
MaxVel = 10
FinVel = 0#
Tacc = 0.1
Tdec = 0.2
```

```
RetCode = start_arc_tr_move(Axis(0), CenterArray
     (0), Angle, StrVel, MaxVel, FinVel, Tacc,
     Tdec)
```

start_arc_sr_move

```
Dim RetCode As Integer
Dim i As Integer
Dim Axis(0 To 1) As Integer, CenterArray (0 To 1)
     As Double
Dim Angle As Integer
Dim StrVel As Double, MaxVel As Double, FinVel As
     Double,
Dim Tacc As Double, Tdec as Double
Dim Tlacc As Double, Tldec as Double
For i = 0 To 1
    Axis(i) = i
CenterArray (i) = i*10+5
Next I
Angle = 180
StrVel = 0#
MaxVel = 10
FinVel = 0#
Tacc = 0.1
Tdec = 0.2
RetCode = start_arc_sr_move(Axis(0), CenterArray
     (0), Angle, StrVel, MaxVel, FinVel, Tacc,
     Tdec, Tlacc, Tldec)
```

start_arc_ta_move

```
Dim RetCode As Integer
Dim i As Integer
Dim Axis(0 To 1) As Integer, CenterArray (0 To 1)
     As Double
Dim Angle As Integer
Dim StrVel As Double, MaxVel As Double, FinVel As
     Double,
Dim Tacc As Double, Tdec as Double
For i = 0 To 1
    Axis(i) = i
CenterArray (i) = i*10+5
Next I
Angle = 270
StrVel = 0#
```

```
MaxVel = 10
FinVel = 0#
Tacc = 0.1
Tdec = 0.2
RetCode = start_arc_ta_move(Axis(0), CenterArray
     (0), Angle, StrVel, MaxVel, FinVel, Tacc,
     Tdec)
```

start_arc_sa_move

```
Dim RetCode As Integer
Dim i As Integer
Dim Axis(0 To 1) As Integer, CenterArray (0 To 1)
     As Double
Dim Angle As Integer
Dim StrVel As Double, MaxVel As Double, FinVel As
     Double,
Dim Tacc As Double, Tdec as Double
Dim Tlacc As Double, Tldec as Double
For i = 0 To 1
    Axis(i) = i
CenterArray (i) = i*10+5
Next I
Angle = 360
StrVel = 0#
MaxVel = 10
FinVel = 0#
Tacc = 0.1
Tdec = 0.2
Tlacc = 0.1
Tldec = 0.2
RetCode = start_arc_sa_move(Axis(0), CenterArray
     (0), Angle, StrVel, MaxVel, FinVel, Tacc,
     Tdec, Tlacc, Tldec)
```

1.4.8 Home Move

@ Name

`set_home_mode(Axis, HomeMode)` – Set home return mode.

`home_move(Axis, StartVel, MaxVel, FinVel, Tacc)` – Perform a home return move.

@ Description

`set_home_mode`:

Configure the home return mode. Refer to chapter 4 for descriptions of different home return modes.

`home_move`:

This function will cause the axis to perform a home return move according to the setting of the set_home_mode() function. The direction of moving is determined by the sign of velocity parameter (StrVel, MaxVel). Since the stopping condition of this function is determined by home_mode setting, user should take care to select the initial moving direction. Or user should take care to handle the condition when limit switch is touched or other conditions that is possible causing the axis to stop. Executing tv_stop(), sv_stop() or emg_stop() function during home_move() can also cause the axis to stop.

@ Syntax

C/C++ (DOS, Windows)

```
I16 set_home_mode(I16 Axis, I16 HomeMode)
I16 home_move(I16 Axis, F64 StartVel, F64 MaxVel,
      F64 FinVel, F64 Tacc)
```

Visual Basic (Windows)

```
set_home_mode (ByVal Axis As Integer, ByVal
      HomeMode As Integer) As Integer
home_move (ByVal Axis As Integer, ByVal StartVel
      As Double, ByVal MaxVel As Double, ByVal
      FinVel As Double, ByVal Tacc As Double) As
      Integer
```

@ Arguments

Axis: axis index designated to configure and perform home returning

StrVel: starting velocity in unit of mm per second

MaxVel: maximum velocity in unit of mm per second

FinVel: final velocity in unit of mm per second

Tacc: specified acceleration time in unit of second

HomeMode: modes for home return

@ Return Code

```
ERROR_NoError
ERROR_Dll_Opened_By_Other_Process
ERROR_Wrong_Axis_Number
ERROR_Card_Not_Ready
ERROR_Card_Not_Initial
ERROR_DSP_Not_Ready
ERROR_Axis_Not_In_Control
ERROR_Axis_Servo_Alarm
ERROR_Axis_Is_Not_Ready_ON
ERROR_Axis_Is_Not_Servo_ON
ERROR_Invalid_MaxVelocity
ERROR_Axis_Hand_Shake_Failed
ERROR_Axis_Not_Response
ERROR_Axis_Prepare_For_Motion
ERROR_Axis_Busy_For_Motion
ERROR_Axis_In_EMG_ON
```

@ Example

<C/C++ >

set_home_mode

```
I16 RetCode;
I16 Axis = 2;
I16 HomeMode = 0;
RetCode  = set_home_mode(Axis, HomeMode);
```

home_move

```
I16 RetCode;
I16 Axis = 2;
```

```
F64 StrVel= 0.0;
F64 MaxVel= 5.0;
F64 FinVel= 0.2;
F64 Tacc = 0.3;
RetCode  = home_move(Axis, StrVel, MaxVel,
     FinVel, Tacc);
```

<Visual Basic>

set_home_mode

```
Dim RetCode As Integer, Axis As Integer, HomeMode
     As Integer
Axis = 2
HomeMode = 0
RetCode  = set_home_mode(Axis, HomeMode)
home_move –
Dim RetCode As Integer, Axis As Integer
Dim StrVel As Double, MaxVel As Double, FinVel As
     Double
Dim Tacc As Double
StrVel = 0
MaxVel= 5.0
FinVel= 0.2
Tacc = 0.3
RetCode  = home_move(Axis, StrVel, MaxVel,
     FinVel, Tacc)
```

1.4.9 Continuous Motion - Start/End motion list

@ Name

`start_motion_list(Length, *AxisArray)` – declare of begin of list of continuous motion trajectory

`end_motion_list(void)` – declare of end of list of continuous motion trajectory

`repeat_last_move(Axis)` – Repeat last motion list move

@ Description

`start_motion_list`:

This function is used to declare the starting of a trajectory list. User must call this before building his continuous motion trajectory. Also, this function declares the axes applying continuous motion. Once your program has called this function, remember to call end_motion_list() to exit trajectory list. Otherwise, not only the built trajectory list won't work, but also a second start_motion_list() will fail.

`end_motion_list`:

This function is used to declare the end of a trajectory list for continuous motion. User must call this function, so that start_cont_move() can be successfully executed.

`repeat_last_move`:

This function is used to restart the move created by previous motion list.

@ Syntax

C/C++ (DOS, Windows)

```
I16 start_motion_list(I16 Length, I16 *AxisArray)
I16 end_motion_list(void)
I16 repeat_last_move(I16 Axis)
```

Visual Basic (Windows)

```
start_motion_list (ByVal length As Integer,
     AxisArray As Integer) As Integer
end_motion_list () As Integer
```

```
repeat_last_move(ByVal Axis As Integer) As
     Integer
```

@ Arguments

Length: the total number of axis to apply move.

***AxisArray**: array of axis number designated to move.

Axis: The first axis of motion list

@ Return Code

```
ERROR_NoError
ERROR_Dll_Opened_By_Other_Process
ERROR_Wrong_Axis_Number
ERROR_Invalid_Length_Of_Axes
ERROR_Card_Not_Ready
ERROR_Card_Not_Initial
ERROR_DSP_Not_Ready
ERROR_Axis_Not_In_Control
ERROR_Axis_Servo_Alarm
ERROR_Axis_Is_Not_Ready_ON
ERROR_Axis_Is_Not_Servo_ON
ERROR_Axis_Prepare_For_Motion
ERROR_Axis_Busy_For_Motion
ERROR_Axis_In_EMG_ON
```

@ Example

<C/C++ >

start_motion_list

```
I16 RetCode;
I16 Length = 2;
I16 Axis[2] = {0, 3};
RetCode  = start_motion_list(Length, Axis);
```

end_motion_list

```
I16 RetCode;
RetCode = end_motion_list();
```

<Visual Basic>

start_motion_list

```
Dim RetCode As Integer, Length As Integer, Axis(0
     to 2) As Integer
Length = 2
Axis(0) = 0
Axis(1) = 3
RetCode  = start_motion_list(Length, Axis(0))
```

end_motion_list

```
Dim RetCode As Integer
RetCode  = end_motion_list()
```

1.4.10Continuous Motion - Add linear trajectory

@ Name

add_line_tr_move(*DistArray, StrVel, MaxVel, FinVel, Tacc, Tdec) – add a relative trapezoidal line into trajectory list

add_line_sr_move(*DistArray, StrVel, MaxVel, FinVel, Tacc, Tdec) – add a relative S-curve line into trajectory list

add_line_ta_move(*PosArray, StrVel, MaxVel, FinVel, Tacc, Tdec) – add a absolute trapezoidal line into trajectory list

add_line_sa_move(*PosArray, StrVel, MaxVel, FinVel, Tacc, Tdec) – add a absolute S-curve line into trajectory list

@ Description

add_line_tr_move:

This function will add a relative trapezoidal line into trajectory list. The StrVel, MaxVel, FinVel is for tangential velocity.

add_line_sr_move:

This function will add a relative S-curve line into trajectory list. The StrVel, MaxVel, FinVel is for tangential velocity.

add_line_ta_move:

This function will add a absolute trapezoidal line into trajectory list. The StrVel, MaxVel, FinVel is for tangential velocity.

add_line_sa_move:

This function will add a absolute S-curve line into trajectory list. The StrVel, MaxVel, FinVel is for tangential velocity.

@ Syntax

C/C++ (DOS, Windows)

```
I16 add_line_tr_move(I16 *DistArray, F64 StrVel,
     F64 MaxVel, F64 FinVel, F64 Tacc, F64 Tdec)
```

```
I16 add_line_sr_move(I16 *DistArray, F64 StrVel,
     F64 MaxVel, F64 FinVel, F64 Tacc, F64 Tdec)
I16 add_line_ta_move(I16 *PosArray, F64 StrVel,
     F64 MaxVel, F64 FinVel, F64 Tacc, F64 Tdec)
I16 add_line_sa_move(I16 *PosArray, F64 StrVel,
     F64 MaxVel, F64 FinVel, F64 Tacc, F64 Tdec)
```

Visual Basic (Windows)

```
add_line_tr_move (DistArray As Double, ByVal
     StrVel As Double, ByVal MaxVel As Double,
     ByVal FinVel As Double, ByVal Tacc As
     Double, ByVal Tdec As Double) As Integer
add_line_sr_move (DistArray As Double, ByVal
     StrVel As Double, ByVal MaxVel As Double,
     ByVal FinVel As Double, ByVal Tacc As
     Double, ByVal Tdec As Double, ByVal Tlacc As
     Double, ByVal Tldec As Double) As Integer
add_line_ta_move (posarray As Double, ByVal
     StrVel As Double, ByVal MaxVel As Double,
     ByVal FinVel As Double, ByVal Tacc As
     Double, ByVal Tdec As Double) As Integer
add_line_sa_move (posarray As Double, ByVal
     StrVel As Double, ByVal MaxVel As Double,
     ByVal FinVel As Double, ByVal Tacc As
     Double, ByVal Tdec As Double, ByVal Tlacc As
     Double, ByVal Tldec As Double) As Integer
```

@ Arguments

***DistArray**: array of specified relative distance to move in unit of mm.

***PosArray**: array of specified absolute position to move in unit of mm.

StrVel: starting tangential velocity, in unit of mm per second.

MaxVel: maximum tangential velocity, in unit of mm per second.

FinVel: final tangential velocity, in unit of mm per second.

Tacc: specified acceleration time in unit of second.

Tdec: specified deceleration time in unit of second.

@ Return Code

```
ERROR_NoError
ERROR_Dll_Opened_By_Other_Process
ERROR_Motion_List_Not_Started
ERROR_Invalid_Trajectory
```

@ Example

<C/C++ >

add_line_tr_move–

```
I16 RetCode;
F64 Dist[2]= {10.0, 20.0};
F64 StrVel= 0.0;
F64 MaxVel= 15.0;
F64 FinVel= 0.0;
F64 Tacc = 0.1;
F64 Tdec = 0.2;
RetCode  = add_line_tr_move(Dist, StrVel, MaxVel,
     FinVel, Tacc, Tdec);
```

add_line_sr_move

```
I16 RetCode;
F64 Dist[2]= {20.0, 20.0};
F64 StrVel= 0.0;
F64 MaxVel= 5.0;
F64 FinVel= 0.0;
F64 Tacc = 0.3
F64 Tdec = 0.2
F64 Tlacc = 0.3;
F64 Tldec = 0.0;
RetCode  = add_line_sr_move(Dist, StrVel, MaxVel,
     FinVel, Tacc, Tdec, Tlacc, Tldec);
```

add_line_ta_move

```
I16 RetCode;
F64 Pos[2]= {50.0, 40.0};
F64 StrVel= 0.0;
F64 MaxVel= 25.0;
F64 FinVel= 0.0;
F64 Tacc = 0.2;
F64 Tdec = 0.1;
```

```
RetCode  = add_line_ta_move(Pos, StrVel, MaxVel,
     FinVel, Tacc, Tdec);
```

add_line_sa_move

```
I16 RetCode;
F64 Pos[2]= {50.0, 30.0};
F64 StrVel= 0.0;
F64 MaxVel= 25;
F64 FinVel= 0.0;
F64 Tacc = 0.2;
F64 Tdec = 0.2;
F64 Tlacc = 0.1;
F64 Tldec = 0.0;
RetCode  = add_line_sa_move(Pos, StrVel, MaxVel,
     FinVel, Tacc, Tdec, Tlacc, Tldec);
```

<Visual Basic>

add_line_tr_move

```
Dim RetCode As Integer
Dim Dist(0 To 1) As Double
Dim StrVel As Double, MaxVel As Double, FinVel As
     Double,
Dim Tacc As Double, Tdec as Double
Dist(0) = 10
Dist(1) = 20
StrVel = 0#
MaxVel = 10
FinVel = 0#
Tacc = 0.1
Tdec = 0.2
Length = 2
RetCode = add_line_tr_move(Dist(0), StrVel,
     MaxVel, FinVel, Tacc, Tdec)
```

add_line_sr_move

```
Dim RetCode As Integer
Dim Dist(0 To 1) As Double
Dim StrVel As Double, MaxVel As Double, FinVel As
     Double,
Dim Tacc As Double, Tdec as Double
Dim Tlacc As Double, Tldec as Double
Dist(0) = 10
Dist(1) = 20
```

```
StrVel = 0#
MaxVel = 10
FinVel = 0#
Tacc = 0.1
Tdec = 0.2
Length = 2
RetCode = add_line_sr_move(Dist(0), StrVel,
     MaxVel, FinVel, Tacc, Tdec, Tlacc, Tldec)
```

add_line_ta_move

```
Dim RetCode As Integer
Dim Pos(0 To 1) As Double
Dim StrVel As Double, MaxVel As Double, FinVel As
     Double,
Dim Tacc As Double, Tdec as Double
Pos(0) = 10
Pos(1) = 20
StrVel = 0#
MaxVel = 10
FinVel = 0#
Tacc = 0.1
Tdec = 0.2
Length = 2
RetCode = add_line_ta_move(Pos(0), StrVel,
     MaxVel, FinVel, Tacc, Tdec)
```

add_line_sa_move

```
Dim RetCode As Integer
Dim Pos(0 To 1) As Double
Dim StrVel As Double, MaxVel As Double, FinVel As
     Double,
Dim Tacc As Double, Tdec as Double
Dim Tlacc As Double, Tldec as Double
Pos(0) = 10
Pos(1) = 20
StrVel = 0#
MaxVel = 10
FinVel = 0#
Tacc = 0.1
Tdec = 0.2
Tlacc = 0.1
Tldec = 0.2
Length = 2
```

```
RetCode = add_line_sa_move(Pos(0), StrVel,
      MaxVel, FinVel, Tacc, Tdec, Tlacc, Tldec)
```

1.4.11Continuous Motion - Add arc trajectory

@ Name

add_arc_tr_move(*CenterArray, Angle, StrVel, MaxVel, FinVel, Tacc, Tdec) – add a relative trapezoidal 2D arc into trajectory list

add_arc_sr_move(*CenterArray, Angle, StrVel, MaxVel, FinVel, Tacc, Tdec) – add a relative S-curve 2D arc into trajectory list

add_arc_ta_move(*CenterArray, Angle, StrVel, MaxVel, FinVel, Tacc, Tdec) – add a absolute trapezoidal 2D arc into trajectory list

add_arc_sa_move(*CenterArray, Angle, StrVel, MaxVel, FinVel, Tacc, Tdec) – add a absolute S-curve 2D arc into trajectory list

add_arc2_sa_move(*AxisArray, *CenterPosArray, Angle, StrVel, MaxVel, FinVel, Tacc, Tlacc, Tdec, Tldec)

add_arc2_sr_move(*AxisArray, *CenterDistArray, Angle, StrVel, MaxVel, FinVel, Tacc, Tlacc, Tdec, Tldec)

add_arc2_ta_move(*AxisArray ,*CenterPosArray, Angle, StrVel, MaxVel, FinVel, Tacc, Tdec)

add_arc2_tr_move(*AxisArray, *CenterDistArray, Angle, StrVel, MaxVel, FinVel, Tacc, Tdec)

@ Description

add_arc_tr_move:

This function will add a relative trapezoidal 2D arc into continuous motion trajectory. The StrVel, MaxVel, FinVel is for tangential velocity.

Note: This function is useless if Length of start_motion_list() is not equal to 2.

add_arc_sr_move:

This function will add a relative S-curve arc into continuous motion trajectory. The StrVel, MaxVel, FinVel is for tangential velocity.

Note: This function is useless if Length of start_motion_list() is not equal to 2.

`add_arc_ta_move`:

This function will add a absolute trapezoidal arc into continuous motion trajectory. The StrVel, MaxVel, FinVel is for tangential velocity.

Note: This function is useless if Length of start_motion_list() is not equal to 2.

`add_arc_sa_move`:

This function will add a absolute S-curve arc into continuous motion trajectory. The StrVel, MaxVel, FinVel is for tangential velocity.

Note: This function is useless if Length of start_motion_list() is not equal to 2.

`add_arc2_tr_move`:

This function will add a relative trapezoidal 2D arc into continuous motion trajectory. The StrVel, MaxVel, FinVel is for tangential velocity.

`add_arc2_sr_move`:

This function will add a relative S-curve arc into continuous motion trajectory. The StrVel, MaxVel, FinVel is for tangential velocity.

`add_arc2_ta_move`:

This function will add a absolute trapezoidal arc into continuous motion trajectory. The StrVel, MaxVel, FinVel is for tangential velocity.

`add_arc2_sa_move`:

This function will add a absolute S-curve arc into continuous motion trajectory. The StrVel, MaxVel, FinVel is for tangential velocity.

@ Syntax

C/C++ (DOS, Windows)

```
I16 add_arc_tr_move(F64 *CenterArray, F64 Angle,
      F64 StrVel, F64 MaxVel, F64 FinVel, F64
      Tacc, F64 Tdec)
I16 add_arc_sr_move(F64 *CenterArray, F64 Angle,
      F64 StrVel, F64 MaxVel, F64 FinVel, F64
      Tacc, F64 Tdec)
I16 add_arc_ta_move(F64 *CenterArray, F64 Angle,
      F64 StrVel, F64 MaxVel, F64 FinVel, F64
      Tacc, F64 Tdec)
I16 add_arc_sa_move(F64 *CenterArray, F64 Angle,
      F64 StrVel, F64 MaxVel, F64 FinVel, F64
      Tacc, F64 Tdec)
I16 add_arc2_sa_move(I16 *AxisArray, F64*
      CenterPosArray,F64 Angle,F64 StrVel,F64
      MaxVel,F64 FinVel,F64 Tacc,F64  Tlacc,F64
      Tdec,F64 Tldec)
I16 add_arc2_sr_move(I16 *AxisArray,F64*
      CenterDistArray, F64 Angle,F64 StrVel,F64
      MaxVel,F64 FinVel,F64 Tacc,F64 Tlacc,F64
      Tdec,F64 Tldec)
I16 add_arc2_ta_move(I16 *AxisArray, F64*
      CenterPosArray, F64 Angle,F64 StrVel,F64
      MaxVel,F64 FinVel,F64 Tacc,F64 Tdec)
I16 add_arc2_tr_move(I16 *AxisArray, F64*
      CenterDistArray, F64 Angle,F64 StrVel,F64
      MaxVel,F64 FinVel,F64 Tacc,F64 Tdec)
```

Visual Basic (Windows)

```
add_arc_tr_move (CenterDistArray As Double, ByVal
      Angle As Double, ByVal StrVel As Double,
      ByVal MaxVel As Double, ByVal FinVel As
      Double, ByVal Tacc As Double, ByVal Tdec As
      Double) As Integer
add_arc_sr_move (CenterDistArray As Double, ByVal
      Angle As Double, ByVal StrVel As Double,
      ByVal MaxVel As Double, ByVal FinVel As
      Double, ByVal Tacc As Double, ByVal Tdec As
      Double, ByVal Tlacc As Double, ByVal Tldec
      As Double) As Integer
add_arc_ta_move (CenterPosArray As Double, ByVal
      Angle As Double, ByVal StrVel As Double,
```

```
        ByVal MaxVel As Double, ByVal FinVel As
        Double, ByVal Tacc As Double, ByVal Tdec As
        Double) As Integer
    add_arc_sa_move (CenterPosArray As Double, ByVal
        Angle As Double, ByVal StrVel As Double,
        ByVal MaxVel As Double, ByVal FinVel As
        Double, ByVal Tacc As Double, ByVal Tdec As
        Double, ByVal Tlacc As Double, ByVal Tldec
        As Double) As Integer
    add_arc2_tr_move (AxisArray As Integer,
        CenterDistArray As Double, ByVal Angle As
        Double, ByVal StrVel As Double, ByVal MaxVel
        As Double, ByVal FinVel As Double, ByVal
        Tacc As Double, ByVal Tdec As Double) As
        Integer
    add_arc2_sr_move (AxisArray As Integer,
        CenterDistArray As Double, ByVal Angle As
        Double, ByVal StrVel As Double, ByVal MaxVel
        As Double, ByVal FinVel As Double, ByVal
        Tacc As Double, ByVal Tdec As Double, ByVal
        Tlacc As Double, ByVal Tldec As Double) As
        Integer
    add_arc2_ta_move (AxisArray As Integer,
        CenterPosArray As Double, ByVal Angle As
        Double, ByVal StrVel As Double, ByVal MaxVel
        As Double, ByVal FinVel As Double, ByVal
        Tacc As Double, ByVal Tdec As Double) As
        Integer
    add_arc2_sa_move (AxisArray As Integer,
        CenterPosArray As Double, ByVal Angle As
        Double, ByVal StrVel As Double, ByVal MaxVel
        As Double, ByVal FinVel As Double, ByVal
        Tacc As Double, ByVal Tdec As Double, ByVal
        Tlacc As Double, ByVal Tldec As Double) As
        Integer
```

@ Arguments

***AxisArray**: array of axis number designated to move.

***CenterArray**: the coordinate of arc center in unit of mm.

Angle: specified moving angle in unit of degree.

StrVel: starting tangential velocity, in unit of mm per second.

MaxVel: maximum tangential velocity, in unit of mm per second.

FinVel: final tangential velocity, in unit of mm per second.

***AxisArray**: array of axis number designated to move.

Tacc: specified total acceleration time in unit of second.

Tdec: specified total deceleration time in unit of second.

Tlacc: specified linerar acceleration time in unit of second.

Tldec: specified linerar deceleration time in unit of second.

@ Return Code

```
ERROR_NoError
ERROR_Dll_Opened_By_Other_Process
ERROR_Motion_List_Not_Started
ERROR_Invalid_Trajectory
```

@ Example

<C/C++ >

add_arc_tr_move

```
I16 RetCode;
F64 CenterArray [2]= {10.0, 20.0};
F64 Angle = 90;
F64 StrVel= 0.0;
F64 MaxVel= 15.0;
F64 FinVel= 0.0;
F64 Tacc = 0.1;
F64 Tdec = 0.2;
RetCode  = add_arc_tr_move(CenterArray, Angle,
     StrVel, MaxVel, FinVel, Tacc, Tdec);
```

add_arc_sr_move

```
I16 RetCode;
F64 CenterArray [2]= {20.0, 20.0};
F64 Angle = 180;
F64 StrVel= 0.0;
F64 MaxVel= 5.0;
F64 FinVel= 0.0;
F64 Tacc = 0.3
F64 Tdec = 0.2
F64 Tlacc = 0.3;
```

```
F64 Tldec = 0.0;
RetCode  = add_arc_sr_move(CenterArray, Angle,
     StrVel, MaxVel, FinVel, Tacc, Tdec, Tlacc,
     Tldec);
```

add_arc_ta_move

```
I16 RetCode;
F64 CenterArray [2]= {50.0, 40.0};
F64 Angle= 270;
F64 StrVel= 0.0;
F64 MaxVel= 25.0;
F64 FinVel= 0.0;
F64 Tacc = 0.2;
F64 Tdec = 0.1;
RetCode  = add_arc_ta_move(CenterArray, Angle,
     StrVel, MaxVel, FinVel, Tacc, Tdec);
```

add_arc_sa_move

```
I16 RetCode;
F64 CenterArray[2]= {50.0, 30.0};
F64 Angle= 360;
F64 StrVel= 0.0;
F64 MaxVel= 25;
F64 FinVel= 0.0;
F64 Tacc = 0.2;
F64 Tdec = 0.2;
F64 Tlacc = 0.1;
F64 Tldec = 0.0;
RetCode  = add_arc_sa_move(CenterArray, Angle,
     StrVel, MaxVel, FinVel, Tacc, Tdec, Tlacc,
     Tldec);
```

add_arc2_tr_move

```
I16 RetCode;
I16 Axis[2] = {0, 2};
F64 CenterArray [2]= {10.0, 20.0};
F64 Angle = 90;
F64 StrVel= 0.0;
F64 MaxVel= 15.0;
F64 FinVel= 0.0;
F64 Tacc = 0.1;
F64 Tdec = 0.2;
```

```
RetCode  = add_arc2_tr_move(Axis, CenterArray,
     Angle, StrVel, MaxVel, FinVel, Tacc, Tdec);
```

add_arc2_sr_move

```
I16 RetCode;
I16 Axis[2] = {0, 2};
F64 CenterArray [2]= {20.0, 20.0};
F64 Angle = 180;
F64 StrVel= 0.0;
F64 MaxVel= 5.0;
F64 FinVel= 0.0;
F64 Tacc = 0.3
F64 Tdec = 0.2
F64 Tlacc = 0.3;
F64 Tldec = 0.0;
RetCode  = add_arc2_sr_move(Axis, CenterArray,
     Angle, StrVel, MaxVel, FinVel, Tacc, Tdec,
     Tlacc, Tldec);
```

add_arc2_ta_move

```
I16 RetCode;
I16 Axis[2] = {0, 2};
F64 CenterArray [2]= {50.0, 40.0};
F64 Angle= 270;
F64 StrVel= 0.0;
F64 MaxVel= 25.0;
F64 FinVel= 0.0;
F64 Tacc = 0.2;
F64 Tdec = 0.1;
RetCode  = add_arc2_ta_move(Axis, CenterArray,
     Angle, StrVel, MaxVel, FinVel, Tacc, Tdec);
```

add_arc2_sa_move

```
I16 RetCode;
I16 Axis[2] = {0, 2};
F64 CenterArray[2]= {50.0, 30.0};
F64 Angle= 360;
F64 StrVel= 0.0;
F64 MaxVel= 25;
F64 FinVel= 0.0;
F64 Tacc = 0.2;
F64 Tdec = 0.2;
F64 Tlacc = 0.1;
```

```
F64 Tldec = 0.0;
RetCode  = add_arc2_sa_move(Axis, CenterArray,
     Angle, StrVel, MaxVel, FinVel, Tacc, Tdec,
     Tlacc, Tldec);
```

<Visual Basic>

add_arc_tr_move

```
Dim RetCode As Integer
Dim i As Integer
Dim CenterArray (0 To 1) As Double
Dim Angle As Integer
Dim StrVel As Double, MaxVel As Double, FinVel As
     Double,
Dim Tacc As Double, Tdec as Double
For i = 0 To 1
CenterArray (i) = i*10+5
Next I
Angle = 90
StrVel = 0#
MaxVel = 10
FinVel = 0#
Tacc = 0.1
Tdec = 0.2
RetCode = add_arc_tr_move(CenterArray (0), Angle,
     StrVel, MaxVel, FinVel, Tacc, Tdec)
```

add_arc_sr_move

```
Dim RetCode As Integer
Dim i As Integer
Dim CenterArray (0 To 1) As Double
Dim Angle As Integer
Dim StrVel As Double, MaxVel As Double, FinVel As
     Double,
Dim Tacc As Double, Tdec as Double
Dim Tlacc As Double, Tldec as Double
For i = 0 To 1
CenterArray (i) = i*10+5
Next I
Angle = 180
StrVel = 0#
MaxVel = 10
FinVel = 0#
Tacc = 0.1
```

```
Tdec = 0.2
RetCode = add_arc_sr_move(CenterArray (0), Angle,
     StrVel, MaxVel, FinVel, Tacc, Tdec, Tlacc,
     Tldec)
```

add_arc_ta_move

```
Dim RetCode As Integer
Dim i As Integer
Dim CenterArray (0 To 1) As Double
Dim Angle As Integer
Dim StrVel As Double, MaxVel As Double, FinVel As
     Double,
Dim Tacc As Double, Tdec as Double
For i = 0 To 1
CenterArray (i) = i*10+5
Next I
Angle = 270
StrVel = 0#
MaxVel = 10
FinVel = 0#
Tacc = 0.1
Tdec = 0.2
RetCode = add_arc_ta_move(CenterArray (0), Angle,
     StrVel, MaxVel, FinVel, Tacc, Tdec)
```

add_arc_sa_move

```
Dim RetCode As Integer
Dim i As Integer
Dim CenterArray (0 To 1) As Double
Dim Angle As Integer
Dim StrVel As Double, MaxVel As Double, FinVel As
     Double,
Dim Tacc As Double, Tdec as Double
Dim Tlacc As Double, Tldec as Double
For i = 0 To 1
CenterArray (i) = i*10+5
Next I
Angle = 360
StrVel = 0#
MaxVel = 10
FinVel = 0#
Tacc = 0.1
Tdec = 0.2
```

```
Tlacc = 0.1
Tldec = 0.2
RetCode = add_arc_sa_move(CenterArray (0), Angle,
     StrVel, MaxVel, FinVel, Tacc, Tdec, Tlacc,
     Tldec)
```

add_arc2_tr_move

```
Dim RetCode As Integer
Dim i As Integer
Dim Axis(0 To 1) As Integer, CenterArray (0 To 1)
     As Double
Dim Angle As Integer
Dim StrVel As Double, MaxVel As Double, FinVel As
     Double,
Dim Tacc As Double, Tdec as Double
For i = 0 To 1
    Axis(i) = i
CenterArray (i) = i*10+5
Next I
Angle = 90
StrVel = 0#
MaxVel = 10
FinVel = 0#
Tacc = 0.1
Tdec = 0.2
RetCode = add_arc2_tr_move(Axis(0), CenterArray
     (0), Angle, StrVel, MaxVel, FinVel, Tacc,
     Tdec)
```

add_arc2_sr_move

```
Dim RetCode As Integer
Dim i As Integer
Dim Axis(0 To 1) As Integer, CenterArray (0 To 1)
     As Double
Dim Angle As Integer
Dim StrVel As Double, MaxVel As Double, FinVel As
     Double,
Dim Tacc As Double, Tdec as Double
Dim Tlacc As Double, Tldec as Double
For i = 0 To 1
    Axis(i) = i
CenterArray (i) = i*10+5
Next I
```

```
Angle = 180
StrVel = 0#
MaxVel = 10
FinVel = 0#
Tacc = 0.1
Tdec = 0.2
RetCode = add_arc2_sr_move(Axis(0), CenterArray
     (0), Angle, StrVel, MaxVel, FinVel, Tacc,
     Tdec, Tlacc, Tldec)
```

add_arc2_ta_move

```
Dim RetCode As Integer
Dim i As Integer
Dim Axis(0 To 1) As Integer, CenterArray (0 To 1)
     As Double
Dim Angle As Integer
Dim StrVel As Double, MaxVel As Double, FinVel As
     Double,
Dim Tacc As Double, Tdec as Double
For i = 0 To 1
    Axis(i) = i
CenterArray (i) = i*10+5
Next I
Angle = 270
StrVel = 0#
MaxVel = 10
FinVel = 0#
Tacc = 0.1
Tdec = 0.2
RetCode = add_arc2_ta_move(Axis(0), CenterArray
     (0), Angle, StrVel, MaxVel, FinVel, Tacc,
     Tdec)
```

add_arc2_sa_move

```
Dim RetCode As Integer
Dim i As Integer
Dim Axis(0 To 1) As Integer, CenterArray (0 To 1)
     As Double
Dim Angle As Integer
Dim StrVel As Double, MaxVel As Double, FinVel As
     Double,
Dim Tacc As Double, Tdec as Double
Dim Tlacc As Double, Tldec as Double
```

```
For i = 0 To 1
    Axis(i) = i
CenterArray (i) = i*10+5
Next I
Angle = 360
StrVel = 0#
MaxVel = 10
FinVel = 0#
Tacc = 0.1
Tdec = 0.2
Tlacc = 0.1
Tldec = 0.2
RetCode = add_arc2_sa_move(Axis(0), CenterArray
     (0), Angle, StrVel, MaxVel, FinVel, Tacc,
     Tdec, Tlacc, Tldec)
```

1.4.12Continuous Motion - Add dwell

@ Name

`add_dwell(Sec)` – add a waiting time into trajectory list

@ Description

`add_dwell`:

This function will add a pause period into motion trajectory. The waiting time is in unit of second.

@ Syntax

C/C++ (DOS, Windows)

```
I16 add_dwell (F64 Sec)
```

Visual Basic (Windows)

```
add_dwell (ByVal Sec As Double) As Integer
```

@ Arguments

Sec: dwell period, in unit of second (Min. resolution is 1 ms)

@ Return Code

```
ERROR_NoError
ERROR_Dll_Opened_By_Other_Process
ERROR_Motion_List_Not_Started
```

@ Example

<C/C++ >

`add_dwell`

```
I16 RetCode;
F64 TimeSec = 1;    // 1 second
RetCode  = add_dwell(TimeSec);
```

<Visual Basic>

`add_dwell`

```
Dim RetCode As Integer
Dim TimeSec As Double
TimeSec = 1 '1 Second
```

```
RetCode = add_dwell(TimeSec)
```

1.4.13Continuous Motion - Smooth trajectory

@ Name

`smooth_enable(Flag, R)` – enable/disable trajectory auto smoothing function

@ Description

smooth_enable:

This function is used to enable/disable the smoothing facility. When enabled, the intersection of 2 trajectory will be rounded automatically. Please refer to chapter 4 for more detail explanation.

@ Syntax

C/C++ (DOS, Windows)

```
I16 smooth_enable(I16 Flag, F64 R)
```

Visual Basic (Windows)

```
smooth_enable(ByVal Flag As Integer, ByVal R As
     Double) As Integer
```

@ Arguments

`Flag`: enable/disable of smoothing facility

`disable`: 0, (Default)

`enable`: 1,

`R`: The distance between intersection and round-starting point.

@ Return Code

```
ERROR_NoError
ERROR_Dll_Opened_By_Other_Process
ERROR_Motion_List_Not_Started
```

@ Example

<C/C++ >

`smooth_enable`

```
I16 RetCode;
I16 Flag = 1;    // smooth enabled
F64 Radius = 3; // 3mm
RetCode  = smooth_enable(Flag, Radius);
```

<Visual Basic>

smooth_enable

```
Dim RetCode As Integer
Dim Flag As Integer, Radius As Double
Flag = 1   ‘ Enabled
Radius = 3# ‘3mm
RetCode = smooth_enable(Flag, Radius)
```

1.4.14Continuous Motion - Start/Stop command

@ Name

```
start_cont_move(Void) – Start continuous motion
stop_cont_move(Void) – Stop continuous motion
```

@ Description

`start_cont_move`:

After building a trajectory by start/stop motion list commands or loading it from file, this function can be executed to realize it.

`stop_cont_move`:

This function is used to immediately stop the continuous motion. All axes specified in AxisArray in start_motion_list() will stop. If user want to stop only one axis, he can use tv_stop(), sv_stop or emg_stop(), and, in that case , other axes will keep going.

@ Syntax

C/C++ (DOS, Windows)

```
I16 start_cont_move(Void)
I16 stop_cont_move(Void)
```

Visual Basic (Windows)

```
start_cont_move () As Integer
stop_cont_move () As Integer
```

@ Arguments

@ Return Code

```
ERROR_NoError
ERROR_Dll_Opened_By_Other_Process
ERROR_Motion_List_Not_Defined
ERROR_Card_Not_Ready
ERROR_Card_Not_Initial
ERROR_DSP_Not_Ready
ERROR_Axis_Not_In_Control
ERROR_Axis_Servo_Alarm
ERROR_Axis_Is_Not_Ready_ON
ERROR_Axis_Is_Not_Servo_ON
ERROR_Axis_Hand_Shake_Failed
```

```
ERROR_Axis_Not_Response
```

@ Example

<C/C++ >

start_cont_move

```
I16 RetCode;
RetCode  = start_cont_move();
```

stop_cont_move

```
I16 RetCode;
RetCode  = stop_cont_move ();
```

<Visual Basic>

start_cont_move

```
Dim RetCode As Integer
RetCode  = start_cont_move()
```

stop_cont_move

```
Dim RetCode As Integer
RetCode  = stop_cont_move ()
```

1.5 Motion Related I/O Function

1.5.1 Position control and feedback

@ Name

`get_position(Axis,*Pos_F, *Pos_C)` – get current position of specified axis

`set_position(Axis,Pos)` – set current position of specified axis

`get_target_pos(Axis,* TargetPos)` – get current target position of specified axis

`get_move_ratio(Axis, *PulsePerMM)` – get motion ration of specified axis

`set_move_ratio(Axis, PulsePerMM)` –set motion ration of specified axis

`get_fb_position(Axis, *Pos_F)` – get feedback position only

`shift_position(Axis, Pos)` – Offset both feedback and command position synchronized with SSCNET cycle

@ Description

`get_position`:

This function is used to read the value of current position of specified axis. Both command position and feedback position can be retrieved. The position information is in unit of mm.

`set_position`:

This function is used to set the value of current position of specified axis. The Pos carry the position information in unit of mm.

`get_target_pos`:

This function is used to get current target position of specified axis, which is maintained by software driver. It records the position to settle down for current running motion.

`get_move_ratio`:

This function is used to read the resolution setting of specified axis in unit of pulse/mm .

`set_move_ratio`:

This function is used to set the resolution of axis. If the encoder resolution is 10000 pulses per revolution, and the displacement in one revolution is 5 mm, then the value of PulsePerMM is 10000/5 = 2000.0 (pulse / mm)

`get_fb_position`:

This function is used to get the feedback position only. Not like get_position, it saves get command position time. The position information is in unit of mm.

`shift_position`:

This function is used to shift the feedback and command position inside the SSCNET control board. It is synchronized with the SSC-NET cycle. The purpose is to prevent the 32-bit counters overflow. Especially in endless one direction move application. The position information is in unit of mm.

@ Syntax

C/C++ (DOS, Windows)

```
I16 get_position(I16 Axis, F64 *Pos_F, F64
     *Pos_C)
I16 set_position(I16 Axis, F64 Pos)
I16 get_target_pos(I16 Axis, F64 *TargetPos)
I16 get_move_ratio(I16 Axis, F32 *PulsePerMM)
I16 set_move_ratio(I16 Axis, F32 PulsePerMM)
I16 get_fb_position(I16 Axis, F64 *Pos_F)
I16 shift_position(I16 Axis, F64 shiftMM)
```

Visual Basic (Windows)

```
get_position (ByVal Axis As Integer, Pos_F As
     Double, Pos_C As Double) As Integer
set_position (ByVal Axis As Integer, ByVal
     Position As Double) As Integer
get_target_pos (ByVal Axis As Integer, Position
     As Double) As Integer
get_move_ratio (ByVal Axis As Integer, PulsePerMM
     As Single) As Integer
```

```
set_move_ratio (ByVal Axis As Integer, ByVal
     PulsePerMM As Single) As Integer
get_fb_position (ByVal Axis As Integer, Pos_F As
     Double) As Integer
shift_position (ByVal Axis As Integer, ByVal
     shiftMM As Double) As Integer
```

@ Arguments

Axis: designated axis index.

Pos_F: current feedback position value in unit of mm.

Pos_C: current command position value in unit of mm.

TargetPos: target position value in unit of mm.

PulsePerMM: specified resolution in unit of pulse/mm.

shiftMM: the shift value in mm

@ Return Code

```
ERROR_NoError
ERROR_Dll_Opened_By_Other_Process
ERROR_Wrong_Axis_Number
ERROR_Card_Not_Ready
ERROR_Card_Not_Initial
ERROR_DSP_Not_Ready
ERROR_Axis_Not_In_Control
ERROR_Axis_Busy_For_Motion
ERROR_Invalid_MoveRatio
```

@ Example

<C/C++ >

get_position

```
I16 RetCode;
I16 Axis = 1;
F64 Pos_F;
F64 Pos_C;
RetCode  = get_position(Axis, &Pos_F, &Pos_C);
```

set_position

```
I16 RetCode;
I16 Axis = 1;
```

```
F64 Pos = 0.0;
RetCode  = set_position(Axis, Pos)
```

get_target_position

```
I16 RetCode;
I16 Axis = 0;
F64 TargetPos;
RetCode  = get_target_pos(Axis, &TargetPos)
```

get_move_ratio

```
I16 RetCode;
I16 Axis = 0;
F32 PulsePerMM;
RetCode  = get_move_ratio(Axis, &PulsePerMM);
```

set_move_ratio

```
I16 RetCode;
I16 Axis = 0;
F32 PulsePerMM = 1000.0;    // 1000.0 (pulse/mm)
RetCode  = set_move_ratio(Axis, PulsePerMM);
```

<Visual Basic>

get_position

```
Dim RetCode As Integer
Dim Axis As Integer
Dim Pos_F As Double, Pos_C As Double
Axis = 1
RetCode  = get_position(Axis, Pos_F, Pos_C)
```

set_position

```
Dim RetCode As Integer
Dim Axis As Integer
Dim Pos As Double
Axis = 1
Pos = 0#
RetCode  = set_position(Axis, Pos)
```

get_target_position

```
Dim RetCode As Integer
Dim Axis As Integer
Dim TargetPos As Double
Axis = 1
RetCode  = get_target_pos(Axis, TargetPos)
```

get_move_ratio

```
Dim RetCode As Integer
Dim Axis As Integer
Dim PulsePerMM As Double
Axis = 0
RetCode  = get_move_ratio(Axis, PulsePerMM)
```

set_move_ratio

```
Dim RetCode As Integer
Dim Axis As Integer
Dim PulsePerMM As Double
Axis = 0
PulsePerMM = 1000#           ‘ 1000.0 (pulse/mm)
RetCode  = set_move_ratio(Axis, PulsePerMM)
```

1.5.2 Velocity Feedback

@ Name

`get_velocity(Axis,*Vel_F, *Vel_C)` –Get current speed of specified axis

`get_cnt_speed(CardID, EncNo, *PPS)` – Get external encoder input counter speed

@ Description

`get_velocity`:

This function is used to read current speed of specified axis. Both command velocity and feedback velocity can be retrieved.

`get_cnt_speed`:

This function is used to read current speed of specified external encoder counter in unit of PPS. Notice that the value is calculated by SSCNET cycle. It means that the minimum value is I pulse per SSCNET cycle.

@ Syntax

C/C++ (DOS, Windows)

```
I16 get_velocity(I16 Axis,F64 *Vel_F, F64 *Vel_C)
I16 get_cnt_speed(I16 CardID, I16 EncNo, F32
     *PPS)
```

Visual Basic (Windows)

```
get_velocity (ByVal Axis As Integer, Vel_F As
     Double, Vel_C As Double) As Integer
get_cnt_speed (ByVal CardID As Integer, ByVal
     EncNo As Integer, PPS As float) As Integer
```

@ Arguments

`Axis`: designated axis index.

`*Vel_F`: current feedback speed value in unit of mm/sec

`*Vel_C`: current command speed value in unit of mm/sec

`CardID`: The CardID from MDSP_Initial()

EncNo: The encoder number

PPS: encoder counter speed in PPS

@ Return Code

```
ERROR_NoError
ERROR_Dll_Opened_By_Other_Process
ERROR_Wrong_Axis_Number
ERROR_Card_Not_Ready
ERROR_Card_Not_Initial
ERROR_DSP_Not_Ready
ERROR_Axis_Not_In_Control
```

@ Example

<C/C++ >

get_velocity

```
I16 RetCode;
I16 Axis = 1;
F64 Vel_F;
F64 Vel_C;
RetCode  = get_velocity(Axis, & Vel_F, & Vel_C);
```

<Visual Basic>

get_velocity

```
Dim RetCode As Integer
Dim Axis As Integer
Dim Vel_F As Double, Vel_C As Double
Axis = 1
RetCode  = get_velocity (Axis, Vel_F, Vel_C)
```

1.5.3 Motion DIO Status

@ Name

set_PEL_config(Axis, Logic, Mode) – Set configure of PEL signal

set_MEL_config(Axis, Logic, Mode) – Set configure of MEL signal

set_ORG_config(Axis, Logic) – Set configure of ORG signal

set_EMG_config(CardID, Logic) – Set configure of EMG signal

get_PEL_status(Axis, *Status) – get status of PEL signal

get_MEL_status(Axis, *Status) – get status of MEL signal

get_ORG_status(Axis, *Status) – get status of ORG signal

get_EMG_status(CardID, *Status) – get status of EMG signal

@ Description

set_PEL_config, **set_MEL_config**, **set_ORG_config**, **set_EMG_config**:

These functions are used to set the configuration of motion-related IO, ie PEL, MEL, ORG, EMG. The default configuration is “active low”, which means if PEL is giving a 0V, it will cause axis stop to move in positive direction.

get_PEL_status, **get_MEL_status**, **get_ORG_status**, **get_EMG_status**:

These functions are used to get the status of motion IO

@ Syntax

C/C++ (DOS, Windows)

```
I16 set_PEL_config(I16 Axis, I16 Logic, I16 Mode)
I16 set_MEL_config(I16 Axis, I16 Logic, I16 Mode)
I16 set_ORG_config(I16 Axis, I16 Logic)
```

```
I16 set_EMG_config(I16 Card, I16 Logic)
I16 get_PEL_status(I16 Axis, I16* Status)
I16 get_MEL_status(I16 Axis, I16* Status)
I16 get_ORG_status(I16 Axis, I16* Status)
I16 get_EMG_status(I16 Card, I16* Status)
```

Visual Basic (Windows)

```
get_PEL_status (ByVal Axis As Integer, Status As
     Integer) As Integer
get_MEL_status (ByVal Axis As Integer, Status As
     Integer) As Integer
get_ORG_status (ByVal Axis As Integer, Status As
     Integer) As Integer
get_EMG_status (ByVal Card As Integer, Status As
     Integer) As Integer
set_PEL_config (ByVal Axis As Integer, ByVal
     Logic As Integer, ByVal Mode As Integer) As
     Integer
set_MEL_config (ByVal Axis As Integer, ByVal
     Logic As Integer, ByVal Mode As Integer) As
     Integer
set_ORG_config (ByVal Axis As Integer, ByVal
     Logic As Integer) As Integer
set_EMG_config (ByVal Card As Integer, ByVal
     Logic As Integer) As Integer
```

@ Arguments

`Axis`: designated axis index.

`CardID`: The SSCNet series card index number.

`Logic`: to assign the logic of IO

- value = 0, active low
- value = 1, active high

`Mode`: to assign the response mode when IO becomes actived

- value = 0, stop immediately
- value = 1, slow down then stop

`Status`: IO status, 1 for active, 0 for not active

@ Return Code

```
ERROR_NoError
```

```
ERROR_Dll_Opened_By_Other_Process
ERROR_Wrong_Axis_Number
ERROR_Card_Not_Ready
ERROR_Card_Not_Initial
ERROR_DSP_Not_Ready
```

@ Example

<C/C++ >

set_PEL_config

```
I16 RetCode;
I16 Axis = 1;
I16 Logic = 1;
I16 Mode = 0;
RetCode  = set_PEL_config(Axis, Logic, Mode);
```

set_MEL_config

```
I16 RetCode;
I16 Axis = 1;
I16 Logic = 1;
I16 Mode = 0;
RetCode  = set_MEL_config(Axis, Logic, Mode);
```

set_ORG_config

```
I16 RetCode;
I16 Axis = 1;
I16 Logic = 1;
RetCode  = set_ORG_config(Axis, Logic);
```

set_EMG_config

```
I16 RetCode;
I16 Card = 1;
I16 Logic = 1;
RetCode  = set_EMG_config(CardID, Logic);
```

get_PEL_status

```
I16 RetCode;
I16 Axis = 1;
I16 Status;
RetCode  = get_PEL_status(Axis, &Status);
```

get_MEL_status

```
I16 RetCode;
```

```
I16 Axis = 1;
I16 Status;
RetCode  = get_MEL_status(Axis, &Status);
```

get_ORG_status

```
I16 RetCode;
I16 Axis = 1;
I16 Status;
RetCode  = get_ORG_status(Axis, &Status);
```

get_EMG_status

```
I16 RetCode;
I16 Card = 0;
I16 Status;
RetCode  = get_EMG_status(CardID, &Status);
```

<Visual Basic>

set_PEL_config

```
Dim RetCode As Integer
Dim Axis As Integer, Logic As Integer, Mode As
     Integer
Axis = 1
Logic = 1
Mode = 0
RetCode  = set_PEL_config(Axis, Logic, Mode)
```

set_MEL_config

```
Dim RetCode As Integer
Dim Axis As Integer, Logic As Integer, Mode As
     Integer
Axis = 1
Logic = 1
Mode = 0
RetCode  = set_MEL_config(Axis, Logic, Mode)
```

set_ORG_config

```
Dim RetCode As Integer
Dim Axis As Integer, Logic As Integer
Axis = 1;
Logic = 1;
RetCode  = set_ORG_config(Axis, Logic)
```

set_EMG_config

```
Dim RetCode As Integer
Dim Card As Integer, Logic As Integer
Card = 1;
Logic = 1;
RetCode  = set_EMG_config(CardID, Logic)
```

get_PEL_status

```
Dim RetCode As Integer
Dim Axis As Integer, Status As Integer
Axis = 1
RetCode  = get_PEL_status(Axis, Status)
```

get_MEL_status

```
Dim RetCode As Integer
Dim Axis As Integer, Status As Integer
Axis = 1
RetCode  = get_MEL_status(Axis, Status)
```

get_ORG_status

```
Dim RetCode As Integer
Dim Axis As Integer, Status As Integer
Axis = 1
RetCode  = get_ORG_status(Axis, Status)
```

get_EMG_status

```
Dim RetCode As Integer
Dim Card As Integer, Status As Integer
Card = 0
RetCode  = get_EMG_status(CardID, Status)
```

1.5.4 Software Limit

@ Name

`set_soft_limit(Axis, PLimit, MLimit, ON_OFF)` – Set and enable/disable soft limit

`get_soft_limit(Axis, *PLimit, *MLimit, *ON_OFF)` – get current soft limit value

@ Description

`set_soft_limit`:

This function is used to set the value of software limit and enable/ disable the software limit function.

`get_soft_limit`:

This function is used to get current setting value of software limit.

@ Syntax

C/C++ (DOS, Windows)

```
I16 set_soft_limit(I16 Axis, F64 PLimit, F64
     MLimit, I16 ON_OFF)
I16 get_soft_limit(I16 Axis, F64 *PLimit, F64
     *MLimit, I16 *ON_OFF)
```

Visual Basic (Windows)

```
set_soft_limit(ByVal Axis As Integer, ByVal
     Plimit As Double, ByVal Mlimit As Double,
     ByVal ON_OFF As Integer) As Integer
get_soft_limit(ByVal Axis As Integer, Plimit As
     Double, Mlimit As Double, ON_OFF As Integer)
     As Integer
```

@ Arguments

`Axis`: designated axis index.

`PLimit`: the positive software limit value in unit of mm.

`MLimit`: the minus software limit value in unit of mm.

ON_OFF: turn ON/OFF the software limit function.

- 0: OFF
- 1: ON

@ Return Code

```
ERROR_NoError
ERROR_Dll_Opened_By_Other_Process
ERROR_Wrong_Axis_Number
ERROR_Card_Not_Ready
ERROR_Card_Not_Initial
ERROR_DSP_Not_Ready
ERROR_Axis_Not_In_Control
```

@ Example

<C/C++ >

set_soft_limit

```
I16 RetCode;
I16 Axis = 1;
F64 Plimit = 100.0;
F64 Mlimit =  -10.0;
I16 ON_OFF = 1;
RetCode  = set_soft_limit(Axis, PLimit, MLimit,
     ON_OFF);
```

get_soft_limit

```
I16 RetCode;
I16 Axis = 1;
F64 Plimit , Mlimit;
I16 ON_OFF;
RetCode  = get_soft_limit(Axis, &PLimit, &MLimit,
     &ON_OFF);
```

<Visual Basic>

set_soft_limit

```
Dim RetCode As Integer
Dim Axis As Integer, ON_OFF As Integer
Dim Plimit As Double, Mlimit As Double
Axis = 1
Plimit = 100.0
Mlimit =  -10.0
```

```
ON_OFF = 1
RetCode  = set_soft_limit(Axis, PLimit, MLimit,
     ON_OFF)
```

get_soft_limit

```
Dim RetCode As Integer
Dim Axis As Integer, ON_OFF As Integer
Dim Plimit As Double, Mlimit As Double
I16 Axis = 1
RetCode  = get_soft_limit(Axis, PLimit, MLimit,
     ON_OFF)
```

1.5.5 Motion Status

@ Name

`motion_status(Axis, *MotionStatus)` – return motion status of specified axis

`axis_status(Axis,*AxisStatus)` – return the servo status of axis.

`motion_done(Axis)` – to check if the axis has finished previous motion command

@ Description

`motion_status`:

This function is used to get current motion status of specified axis:

`axis_status`:

This function is used to get current servo status of specified axis:

`motion_done`:

This function is used to check if the axis has finished previous motion command

- return '1' – Axis is ready for next motion command
- return '0' – Axis is still in motion

@ Syntax

C/C++ (DOS, Windows)

```
I16 motion_status(I16 Axis, U16* MotionStatus)
I16 axis_status(I16 Axis, U16* AxisStatus)
I16 motion_done(I16 Axis)
```

Visual Basic (Windows)

```
motion_status (ByVal Axis As Integer,
     MotionStatus As Integer) As Integer
axis_status (ByVal Axis As Integer, AxisStatus As
     Integer) As Integer
motion_done (ByVal Axis As Integer) As Integer
```

@ Arguments

`Axis`: designated axis index.

`*MotionStatus`: axis motion status

Bit	Name	Description
0	Ready_for_Motion	Axis is not moving and it is available for another move command
1	In_Motion	Axis is moving and can't accept another move command
2	In_Home_Move	Axis is in moving in home procedure and can't accept another move command
3	In_V_Change	After lauching velocity change command, this bit will be ON till the change is done
4	In_P_Change	After lauching position change command, this bit will be ON till the change is done
5	MEL_ON	Axis touches the positive limit switch
6	PEL_ON	Axis touches the negative limit switch
7	ORG_ON	Axis touched the origin switch
8	EMG_ON	Emergency input pin is ON
9	P_Soft_ON	Axis is reached the positive software limit
10	M_Soft_ON	Axis is reached the negative software limit
11	EZ_ON	Axis touched the external Index switch
12	Stop_cmd_end	After v_stop() command ends, this bit will be ON
13	Stop_cmd_running	This bit will be ON if users lauched a v_stop() command
14	Interlock_Pause	Once the axis is paused be interlock procedure, this bit will be ON
15	Waiting Other Axis	Command is pending because other axis' condition has not been met

Table 1-4: Motion Status

`*AxisStatus`: axis servo status

Bit	Name	Value & Description
0	Not_In_Control	1: Axis not in control
		0: Axis is in control
1	In_Servo_Alarm	1: Axis is in servo alarm
		0: Axis is not in servo alarm

Table 1-5: Axis Status

Bit	Name	Value & Description
2	Not_Ready_ON	1: Axis not Ready ON
		0: Axis is Ready ON
3	Not_Servo_ON	1: Axis not Servo ON
		0: Axis is Servo ON

Table 1-5: Axis Status

@ Return Code

```
ERROR_NoError
ERROR_Dll_Opened_By_Other_Process
ERROR_Wrong_Axis_Number
ERROR_Card_Not_Ready
ERROR_Card_Not_Initial
ERROR_DSP_Not_Ready
ERROR_Axis_Not_In_Control
ERROR_Axis_Servo_Alarm
ERROR_Axis_Is_Not_Ready_ON
ERROR_Axis_Is_Not_Servo_ON
```

@ Example

<C/C++ >

motion_status

```
I16 RetCode;
I16 Axis = 1;
U16 MotionStatus;
RetCode  = motion_status(Axis, &MotionStatus);
```

axis_status

```
I16 RetCode;
I16 Axis = 1;
U16 AxisStatus;
RetCode  = axis_status(Axis, &AxisStatus);
```

Motion_done

```
I16 Axis = 1;
while (motion_done(Axis) == 0) { ….};
```

<Visual Basic>

motion_status

```
Dim RetCode As Integer
Dim Axis As Integer, MotionStatus As Integer
Axis = 1
RetCode  = motion_status(Axis, MotionStatus)
```

axis_status

```
Dim RetCode As Integer
Dim Axis As Integer, AxisStatus As Integer
Axis = 1
RetCode  = axis_status(Axis, AxisStatus)
```

Motion_done

```
Dim Axis As Integer
Axis = 0
Do while motion_done(Axis) = 0
...
loop
```

1.5.6 Frame Management

@ Name

`frame_to_execute(Axis, *Len)` – check the number of remaining frames to be executed

@ Description

`frame_to_execute`:

This function is used to get the number of remaining frame to be executed.

@ Syntax

C/C++ (DOS, Windows)

```
U16 frame_to_execute(I16 Axis, I16 *Len)
```

Visual Basic (Windows)

```
frame_to_execute (ByVal Axis As Integer, Len As
     Integer) As Integer
```

@ Arguments

`Axis`: designated axis index.

`*Len`: number of frame

@ Return Code

```
ERROR_NoError
ERROR_Dll_Opened_By_Other_Process
ERROR_Wrong_Axis_Number
ERROR_Card_Not_Ready
ERROR_Card_Not_Initial
ERROR_DSP_Not_Ready
ERROR_Axis_Not_In_Control
ERROR_Axis_Servo_Alarm
ERROR_Axis_Is_Not_Ready_ON
ERROR_Axis_Is_Not_Servo_ON
```

@ Example

<C/C++ >

frame_to_execute

```
I16 RetCode;
I16 Axis = 1;
I16 Len;
RetCode  = frame_to_execute(Axis, &Len);
```

<Visual Basic>

frame_to_execute

```
Dim RetCode As Integer
Dim Axis As Integer, Len As Integer
Axis = 1
RetCode  = frame_to_execute (Axis, Len)
```

1.6 General-purposed IO Function

1.6.1 Encoder

@ Name

`set_cnt _iptmode(CardID, EncNo, IptMode)` – set the configuration of encoder counter

`set_cnt_to_axis(CardID, EncNo, Axis, Resolution)` – To assign a counter to some axis as the position feedback source, and declare the resolution of that counter.

`set_cnt_value(CardID, EncNo, Value)` – set current counter value

`get_cnt_value(CardID, EncNo, *Value)` – get current counter value

`set_ring_counter(CardID, EncNo, RingValue)` – set current ring counter value

`get_ring_counter(CardID, EncNo, *RingValue)` – get current ring counter value

@ Description

`set_cnt_iptmode`:

Configure the input modes of encoder counter. There are 3 sets of encoder counter in PCI-83XX series; 2 set in cPCI series.

`set_cnt_to_axis`:

This function has two functionalities. One is to assign a counter to work as position feedback source of certain axis; the other is to declare the resolution of this counter. For example: set_cnt_to_axis(0, 0, 0, 1000) let the counter “0” of SSCNet board as the position feedback source of axis ‘0’. This counter counts 1000 pulses when axis 0 rotates one revolution.

`set_cnt_value`:

Set current counter value.

`get_cnt_value`:

Get current counter value.

`set_ring_counter`:

The external encoder counter could be set as ring counter for rotary application. The ring counter value represents a maximum value of the external encoder counter. Once the counter reaches this value, it will automatically reset by hardware. This value is positive but the counter will be reseted both on positive and negative counter value.

If the value is zero, it means disable ring counter.

`get_ring_counter`:

This function is for users to check the current ring counter value.

@ Syntax

C/C++ (DOS, Windows)

```
I16 set_cnt_iptmode(I16 Card, I16 EncNo, I16
     IptMode)
I16 set_cnt_to_axis(I16 Card, I16 EncNo, I16
     Axis, F32 Resolution)
I16 set_cnt_value(I16 Card, I16 EncNo, I32 Value)
I16 get_cnt_value(I16 Card, I16 EncNo, I32
     *Value)
I16 set_ring_counter(I16 Card, I16 EncNo, I32
     RingValue)
I16 get_ring_counter(I16 Card, I16 EncNo, I32
     *RingValue)
```

Visual Basic (Windows)

```
set_cnt_iptmode (ByVal Card As Integer, ByVal
     EncNo As Integer, ByVal IPTMode As Integer)
     As Integer
set_cnt_to_axis (ByVal Card As Integer, ByVal
     EncNo As Integer, ByVal Axis As Integer,
     ByVal Resolution as Single) As Integer
set_cnt_value (ByVal Card As Integer, ByVal EncNo
     As Integer, ByVal Value As Long) As Integer
get_cnt_value (ByVal Card As Integer, ByVal EncNo
     As Integer, Value As Long) As Integer
set_ring_counter(ByVal Card As Integer, ByVal
     EncNo As Integer, ByVal RingValue As Long)
     As Integer
```

```
get_ring_counter(ByVal Card As Integer, ByVal
     EncNo As Integer,  RingValue As Long) As
     Integer
```

@ Arguments

CardID: The SSCNet series card index number.

EncNo: designated encoder counter channel number.

Axis: designated axis index.

Resolution: number of pulses counted by counter per revolution.

IptMode: setting of encoder feedback pulse input mode

Value	Meaning
0	Not Used
1	OUT/DIR
2	CW/CCW
3	4X A/B
4	2X A/B
5	1X A/B

Table 1-6: IPT Modes

Axis: designated index.

Value: counter value.

RingValue: a 32-bit value for ring counter. 0 means disable

@ Return Code

```
ERROR_NoError
ERROR_Dll_Opened_By_Other_Process
ERROR_Card_Not_Exist
ERROR_Card_Not_Ready
ERROR_Card_Not_Initial
ERROR_DSP_Not_Ready
ERROR_Wrong_Encoder_Channel_Number
```

@ Example

<C/C++ >

set_cnt_iptmode

```
I16 RetCode;
I16 Card = 0;
I16 EncNo = 1;
I16 IptMode = 1;
RetCode  = set_cnt_iptmode(CardID, EncNo,
     IptMode);
```

set_cnt_to_axis

```
I16 RetCode;
I16 Card = 0;
I16 EncNo = 0;
I16 Axis = 1
F32 Resolution = 1000.0;
RetCode  = set_cnt_to_axis(CardID, EncNo, Axis,
     Resolution);
```

set_cnt_value

```
I16 RetCode;
I16 Card = 0;
I16 EncNo = 1;
I32 Value = 0;
RetCode  = set_cnt_value(CardID, EncNo, Value);
```

get_cnt_value

```
I16 RetCode;
I16 Card = 0;
I16 EncNo = 1;
I32 Value;
RetCode  = get_cnt_value(CardID, EncNo, &Value);
```

<Visual Basic>

set_cnt_iptmode

```
Dim RetCode As Integer
Dim Card As Integer, EncNo As Integer, IptMode As
     Integer
Card = 0
EncNo = 1
IptMode = 1
RetCode  = set_cnt _iptmode(CardID, EncNo,
     IptMode)
```

set_cnt_to_axis

```
Dim RetCode As Integer
Dim Card As Integer, EncNo As Integer, Axis As
     Integer
Dim Resolution As Single
Card = 0
EncNo = 1
IptMode = 1
Resolution = 1000.0
RetCode  = set_cnt_to_axis(CardID, EncNo, Axis,
     Resolution)
```

set_cnt_value

```
Dim RetCode As Integer
Dim Card As Integer, EncNo As Integer, Value As
     Long
Card = 0
EncNo = 1
Value = 0
RetCode  = set_cnt_value(CardID, EncNo, Value)
```

get_cnt_value

```
Dim RetCode As Integer
Dim Card As Integer, EncNo As Integer, Value As
     Long
Card = 0
EncNo = 1
RetCode  = get_cnt_value(CardID, EncNo, Value)
```

1.6.2 DIO

@ Name

`get_di_status(CardID, ChNo ,*Sts)` – get DO status

`set_do_value(CardID, ChNo, Value)` – set DO value

`set_di_mode(CardID, DI_Channel, DI_Mode, ActiveL)` – Set DI mode

`set_mio_mode(CardID, MDI_Ch, Mode)`

@ Description

`get_di_status`:

This function is used to get current DI status.

`set_do_value`:

This function is used to set DO value.

`set_di_mode`:

This function is used to set DI Channel mode.

`set_mio_mode`:

This function is used to switch the motion DI (PEL, MEL, EMG) to general purpose DI.

@ Syntax

C/C++ (DOS, Windows)

```
I16 get_di_status(I16 Card, I16 ChNo, I16 *Sts)
I16 set_do_value(I16 Card, I16 ChNo, I16 Value)
I16 set_di_mode(I16 Card, I16 DI_Channel, I16
     DI_Mode, I16 ActiveL)
I16 set_mio_mode(I16 Card, I16 MDI_Ch, I16 Mode)
```

Visual Basic (Windows)

```
get_di_status (ByVal Card As Integer, ByVal ChNo
     As Integer, Sts As Integer) As Integer
set_do_value (ByVal Card As Integer, ByVal ChNo
     As Integer, ByVal Value As Integer) As
     Integer
```

```
set_di_mode(ByVal Card As Integer, ByVal
     DI_Channel As Integer, ByVal DI_Mode As
     Integer, ByVal ActiveL As Integer)
set_mio_mode(ByVal Card As Integer, ByVal MDI_Ch
     As Integer, ByVal Mode As Integer)
```

@ Arguments

CardID: The SSCNet series card index number.

***Sts**: DI status :

ChNo: specified channel number

- For get_di_status : ChNo = 0 , 1
- For set_do_value : ChNo = 0 , 1 (DO on CN5) ; 2 ~7 (TTL output on CN3)

Value: DO value , 0: Low, 1: High

DI_Channel: 0=EMG, 1=DI0, 2=DI1

DI_Mode: 0=original function, 1=slow down and stop axis motion

ActiveL: DI active level, 0=active low, 1=active high

MDI_Ch:

Axis	DI	MDI_Ch
Axis 0	PEL	0
	MEL	1
	ORG	2
Axis 1	PEL	3
	MEL	4
	ORG	5
. . .	. . .	. . .
Axis 11	PEL	33
	MEL	34
	ORG	35

```
Mode: 0 = Default, 1= general DI
```

@ Return Code

```
ERROR_NoError
ERROR_Dll_Opened_By_Other_Process
ERROR_Card_Not_Exist
ERROR_Card_Not_Ready
ERROR_Card_Not_Initial
ERROR_DSP_Not_Ready
ERROR_Wrong_DIO_Channel_Number
```

@ Example

<C/C++ >

get_di_status

```
I16 RetCode;
I16 Card = 0;
I16 ChNo = 1;
I16 Sts;
RetCode  = get_di_status(CardID, ChNo, &Sts);
```

set_do_value

```
I16 RetCode;
I16 Card = 0;
I16 ChNo = 1;
I16 Value = 0;
RetCode  = set_do_value(CardID, ChNo, Value);
```

<Visual Basic>

get_di_status

```
Dim RetCode As Integer
Dim Card As Integer, EncNo As Integer, Sts As
     Long
Card = 0
EncNo = 1
RetCode  = get_di_status(CardID, ChNo, Sts)
```

set_do_value

```
Dim RetCode As Integer
Dim Card As Integer, EncNo As Integer, Value As
     Long
Card = 0
EncNo = 1
Value =0
```

```
RetCode  = set_do_value(CardID, ChNo, Value)
```

1.6.3 DA

@ Name

`set_da_config(CardID, ChNo, Cfg)` – set configuration of specified DA channel

`set_da_value(CardID,ChNo,Value)` – set output value of specified DA channel

@ Description

`set_da_config`:

Set DA output configuration. There are 2 options. One is for direct DA output and the other is for velocity profile output. The default is DA direct output. By direct DA output, user can control DA value by using set_da_value() function. While if velocity profile output is selected, the DA output value will be proportional to current command velocity.

`set_da_value`:

Set output value/voltage to specified DA channel. There are 2 12-bit DA channels in PCI-83XX Series card.

@ Syntax

C/C++ (DOS, Windows)

```
I16 set_da_config(I16 Card, I16 ChNo, U16 Cfg)
I16 set_da_value(I16 Card, I16 ChNo, F64 Value)
```

Visual Basic (Windows)

```
set_da_config (ByVal Card As Integer, ByVal ChNo
     As Integer, ByVal Cfg As Integer) As Integer
set_da_value (ByVal Card As Integer, ByVal ChNo
     As Integer, ByVal Value As Double) As
     Integer
```

@ Arguments

`CardID`: The SSCNet series card index number.

`ChNo`: specified channel number

Cfg: Setting of configuration

- Cfg = 0, Direct DA output (default)
- Cfg = 1, Velocity profile of Axis 0
- Cfg = 2, Velocity profile of Axis 1
- :
- Cfg = 12, Velocity profile of Axis 11
- Cfg = 13, DA closed loop output of Axis 0
- Cfg = 14, DA closed loop output of Axis 1
- :
- Cfg = 24, DA closed loop output of Axis 11

Value: output value, -10.0 ~ 10.0 V

@ Return Code

```
ERROR_NoError
ERROR_Dll_Opened_By_Other_Process
ERROR_Card_Not_Exist
ERROR_Card_Not_Ready
ERROR_Card_Not_Initial
ERROR_DSP_Not_Ready
ERROR_Wrong_DA_Channel_Number
```

@ Example

<C/C++ >

set_da_config

```
I16 RetCode;
I16 Card = 0;
I16 ChNo = 1;
U16 Cfg = 0;  // Direct DA output
RetCode  = set_da_config(CardID, ChNo, Cfg);
```

set_da_value

```
I16 RetCode;
I16 Card = 0;
I16 ChNo = 1;
I16 Value = 1.0; // DA output 1.0 Volt
RetCode  = set_da_value(CardID, ChNo, Value);
```

<Visual Basic>

set_da_config

```
Dim RetCode As Integer
Dim Card As Integer, ChNo As Integer, Cfg As
     integer
Card = 0
ChNo = 1
Cfg =1     ‘ DA output proportional to Axis_0’s
     velocity
RetCode  = set_da_config(CardID, ChNo, Cfg)
```

set_da_value

```
Dim RetCode As Integer
Dim Card As Integer, ChNo As Integer, Value As
     double
Card = 0
ChNo = 1
Value = 1.0 ‘ DA output 1.0 Volt
RetCode  = set_da_value(CardID, ChNo, Value)
```

1.6.4 AD

@ Name

`set_ad_function(CardID, Enable, AD_Gain, AD_Last, AD2_SRC)` – set configuration of specified AD channel

`get_ad_value(CardID, ChNo, *Value)`– get input value of specified AD channel

@ Description

`set_ad_function` :

Set AD input configuration. There are 3 AD channel available on cPCI Series SSCNET motion cards. AD0 and AD1's sources are directly from external pin. AD2's sources are selectable. Users can set the AD2 channel input source from internal +5V , GND and one of external AD pins. Each channel's gain is also adjustable from 4x, 2x, and 1x for voltage range +/-2.5V, +/-5V, +/-10V.

`get_ad_value` :

Get input voltage from specified AD channel. There are 2 16-bit AD channels in cPCI Series SSCNET motion cards.

@ Syntax

C/C++ (DOS, Windows)

```
I16   set_ad_function(I16 CardID, U16 Enable, U16
      AD_Gain, U16 AD_Last, U16 AD2_SRC)
I16   get_ad_value(I16 CardID, I16 ChNo, F64
      *Value)
```

Visual Basic (Windows)

```
set_ad_function (ByVal CardID As Integer, ByVal
      Enable As Integer, ByVal AD_Gain As Integer,
      ByVal AD_Last As Integer, ByVal AD2_SRC As
      Integer) As Integer
get_ad_value (ByVal Card As Integer, ByVal ChNo
      As Integer, Value As Double) As Integer
```

@ Argument

`CardID`: The SSCNet series card index number.

ChNo: specified channel number, 0=AD0, 1=AD1, 2=AD2

AD_Gain: 0 = 1x, 1 = 2x, 2 = 4x

AD_Last: 0 = Ch0 1 = Ch1, 2 = Ch2 (Last AD scan channel)

AD2_SRC: 0 = Ref+5V, 1 = AGND, 2 = DA0, 3 = DA1

Enable: 0=disable, 1=enable

***Value**: input value, -10.0 ~ 10.0 V

@ Return Code

```
ERROR_NoError
ERROR_Dll_Opened_By_Other_Process
ERROR_Card_Not_Exist
ERROR_Card_Not_Ready
ERROR_Card_Not_Initial
ERROR_DSP_Not_Ready
ERROR_Wrong_DA_Channel_Number
```

1.6.5 Analog Auto Calibration

@ Name

`tune_ref_5v(CardID, Value)` – tunning reference 5V

`save_auto_k_value(CardID, Channel, Value)` – save tunning value to ROM

`get_auto_k_value(CardID, Channel, *Value)` – get tunning value from ROM

`tune_ad_offset_gain(CardID, Step, Value)` – tune AD offset and gain

`tune_da_offset(CardID, Step, Value)` – tune DA offset

`reload_auto_k_setting(CardID)` – get tunning data from ROM and set to analog control register

@ Description

`tune_ref_5v`:

This function is used for tuning on board reference +5V source. This source is very important because the whole auto calibration procedures need this accurate voltage source. Users must use a voltage meter to measure the on board reference 5V from JP1 connector.

`save_auto_k_value`:

This function is used for saving the auto-k value channel by channel to EEPROM.

`get_auto_k_value`:

This function is used for getting the auto-k value channel by channel from EEPROM.

`tune_ad_offset_gain`:

This function is used for tunning ADC's 0v offset and its gain.

`tune_da_offset`:

This function is used for tunning each DA channel's 0 voltage output offset.

`reload_auto_k_setting`:

Get auto-k setting from EEPROM and set them to analog control register. This function will be called automatically when MDSP_initial() is called.

@ Syntax

C/C++ (DOS, Windows)

```
I16 tune_ref_5v(I16 CardID, U8 Value);
I16 tune_ad_offset_gain(I16 CardID, I16 Step, U8
     Value);
```

Visual Basic (Windows)

```
tune_ref_5v (ByVal CardID As Integer, ByVal Value
     As Byte) As Integer
tune_ad_offset_gain (ByVal CardID As Integer,
     ByVal Step As Integer, ByVal Value Byte) As
     Integer
```

@ Argument

`CardID`: The SSCNet series card index number.

`Channel`: specified auto-K channel number, 0-8 for cPCI system. 0-2 for PCI system.

- Channel0=No use.
- Channel1= DA0 Fine Offset
- Channel2= DA1 Fine Offset
- Channel3= AD Fine Offset
- Channel4= AD Fine Gain, not available now
- Channel5= DA0 Coarse Offset
- Channel6= DA1 Coarse Offset
- Channel7= AD Coarse Offset
- Channel8= AD Coarse Gain

`Step`: tunning step number 0~3

[AD Function]

- Step=0 Tune AD Coarse Offset, Channel7
- Step=1 Tune AD Fine Offset, Channel3
- Step=2 Tune AD Coarse Gain, Channel8
- Step=3 Tune AD Fine Gain, Channel4, not available now

[DA Function]

- Step=0 Tune DA0 Coarse Offset, Channel5
- Step=1 Tune DA0 Fine Offset, Channel1
- Step=2 Tune DA1 Coarse Offset, Channel6
- Step=3 Tune DA1 Fine Offset, Channel2

`Value`: tunnig value, 0 ~ 255

@ Return Code

```
ERROR_NoError
ERROR_Dll_Opened_By_Other_Process
ERROR_Card_Not_Exist
ERROR_Card_Not_Ready
ERROR_Card_Not_Initial
ERROR_DSP_Not_Ready
ERROR_Wrong_DA_Channel_Number
ERROR_DPBUSY_TIMEOUT
```

@ Example

```
tune_ref_5v( 0, 128); // use this function until
      voltage meter read 5.000V from JP1

tune_ad_offset_gain(0,1,128); // reset to middle
      point
tune_ad_offset_gain(0,3,128); // reset to middle
      point

tune_ad_offset_gain(0,0,K7); // start tunning AD
      offset
  get_ad_value(0, 2 , &AD2_Value); // check
      AD2_Value = 0.0
  save_auto_k_value(0, 7, K7); // save to channel
      7
```

```
tune_ad_offset_gain(0,1,K3); // start tunning AD
      offset
  get_ad_value(0, 2 , &AD2_Value); // check
      AD2_Value = 0.0
  save_auto_k_value(0, 3, K3); // save to channel
      3
tune_ad_offset_gain(0,2,K8); // start tunning AD
      offset
  get_ad_value(0, 2 , &AD2_Value); // check
      AD2_Value = 5.0
  save_auto_k_value(0, 8, K8); // save to channel
      8

tune_da_offset_gain(0,1,128); // reset to middle
      point
tune_da_offset_gain(0,3,128); // reset to middle
      point

tune_da_offset_gain(0,0,K5); // start tunning AD
      offset
  get_ad_value(0, 2 , &AD2_Value); // check
      AD2_Value = 0.0
  save_auto_k_value(0, 5, K5); // save to channel
      5

tune_ad_offset_gain(0,1,K1); // start tunning AD
      offset
  get_ad_value(0, 2 , &AD2_Value); // check
      AD2_Value = 0.0
  save_auto_k_value(0, 1, K1); // save to channel
      1

tune_ad_offset_gain(0,2,K6); // start tunning AD
      offset
  get_ad_value(0, 2 , &AD2_Value); // check
      AD2_Value = 0.0
  save_auto_k_value(0, 6, K6); // save to channel
      6

tune_ad_offset_gain(0,3,K2); // start tunning AD
      offset
  get_ad_value(0, 2 , &AD2_Value); // check
      AD2_Value = 0.0
```

```
  save_auto_k_value(0, 2, K2); // save to channel
      2

// Finally Save Result to ROM
save_auto_k_value(0, 0~8, K0~K8);
```

1.7 Driver Management Function

1.7.1 Driver Parameter

@ Name

`get_servo_para(Axis, ParaNo,*Value)` – Read current servo parameter value

`set_servo_para(Axis, ParaNo, Value)` – Set servo parameter

`get_servo_para_all(Axis, *Value)` – Read all current servo parameter value

`set_servo_para_all(Axis, *Value)` – Set all servo parameters

`save_servo_para(Axis)` – Write current servo parameter value into Flash ROM on PCI-83XX Series.

`set_servo_para_default(Axis)` – Set all servo parameter to default value.

@ Description

`get_servo_para`:

This function is used to read current servo parameter value of specified parameter number.

`set_servo_para`:

This function is used to set a new servo parameter value for specified parameter number.

`get_servo_para_all`:

This function is used to read all current servo parameter value.

`set_servo_para_all`:

This function is used to set new servo value for all servo parameters.

`save_servo_para`:

This function is used to set write current servo parameter value into Flash ROM on PCI-83XX Series. So that, next time when booting, all the parameter values will be the same as saved.

`set_servo_para_default`:

This function is used to set all servo parameter values to default.

@ Syntax

C/C++ (DOS, Windows)

```
I16 get_servo_para(I16 Axis,I16 ParaNo,U16
     *Value)
I16 set_servo_para(I16 Axis,I16 ParaNo,U16 Value)
I16 get_servo_para_all(I16 Axis,U16 *Value)
I16 set_servo_para_all(I16 Axis,U16 *Value)
I16 save_servo_para(I16 Axis)
I16 set_servo_para_default(I16 Axis)
```

Visual Basic (Windows)

```
get_servo_para (ByVal Axis As Integer, ByVal
     ParaNo As Integer, Value As Integer) As
     Integer
set_servo_para (ByVal Axis As Integer, ByVal
     ParaNo As Integer, ByVal Value As Integer)
     As Integer
get_servo_para_all (ByVal Axis As Integer, Value
     As Integer) As Integer
set_servo_para_all (ByVal Axis As Integer, Value
     As Integer) As Integer
save_servo_para (ByVal Axis As Integer) As
     Integer
set_servo_para_default (ByVal Axis As Integer) As
     Integer
```

@ Arguments

`Axis`: specified Axis index

`ParaNo`: specified parameter number. Range: 1 ~ 56

`Value`: servo parameter value.

Note: Please refer to SSCNet servo driver manual for definition of individual servo parameter.

@ Return Code

```
ERROR_NoError
ERROR_Dll_Opened_By_Other_Process
ERROR_Wrong_Axis_Number
ERROR_Card_Not_Ready
ERROR_Card_Not_Initial
ERROR_DSP_Not_Ready
ERROR_Axis_Not_In_Control
ERROR_Axis_Servo_Alarm
ERROR_Axis_Is_In_Servo_ON
ERROR_Servo_Parameter_Set_Failed
ERROR_Servo_Parameter_Get_Failed
```

@ Example

<C/C++ >

get_servo_para

```
I16 RetCode;
I16 Axis = 0;
I16 ParaNo = 9;
U16 Value;
RetCode  = get_servo_para(Axis, ParaNo, &Value);
```

set_servo_para

```
I16 RetCode;
I16 Axis = 0;
I16 ParaNo = 9;
U16 Value = 6;
RetCode  = set_servo_para(Axis, ParaNo, Value);
```

get_servo_para_all

```
I16 RetCode;
I16 Axis = 0;
U16 Value[56];
RetCode  = get_servo_para_all(Axis, Value);
```

set_servo_para_all

```
I16 RetCode;
I16 Axis = 0;
U16  Value[56];
RetCode  = get_servo_para(Axis, Value);
Value[8] = 6;
```

```
RetCode  = set_servo_para(Axis, Value);
```

save_servo_para

```
I16 RetCode;
I16 Axis = 0;
RetCode  = save_servo_para (Axis);
```

set_servo_para_default

```
I16 RetCode;
I16 Axis = 0;
RetCode  = set_servo_para_default(Axis);
```

<Visual Basic>

get_servo_para

```
Dim RetCode As Integer
Dim Axis As Integer, ParaNo As Integer, Value As
     Integer
Axis = 0
ParaNo = 9
RetCode  = get_servo_para(Axis, ParaNo, Value)
```

set_servo_para

```
Dim RetCode As Integer
Dim Axis As Integer, ParaNo As Integer, Value As
     Integer
Axis = 0
ParaNo = 9
Value = 6
RetCode  = set_servo_para(Axis, ParaNo, Value)
```

get_servo_para_all

```
Dim RetCode As Integer
Dim Axis As Integer, Value(0 to 55) As Integer
Axis = 0
RetCode  = get_servo_para_all(Axis, Value(0))
```

set_servo_para_all

```
Dim RetCode As Integer
Dim Axis As Integer, Value(0 to 55) As Integer
Axis = 0
RetCode  = get_servo_para(Axis, Value)
Value(8) = 6
RetCode  = set_servo_para(Axis, Value(0))
```

save_servo_para

```
Dim RetCode As Integer
Dim Axis As Integer
Axis = 0
RetCode  = save_servo_para (Axis)
```

set_servo_para_default–

```
Dim RetCode As Integer
Dim Axis As Integer
Axis = 0
RetCode  = set_servo_para_default(Axis)
```

1.7.2 Data Monitoring

@ Name

set_monitor_channel(Axis, Channel_0, Channel_1,Channel_2, Channel_3) –Set axis data monitoring channel

set_monitor_config(Axis, Trigger_Select, Trigger_Level, SamplePeriod, PreTriggerSampleNo, SampleNumber) – Set axis data monitoring configuration

start_monitor(Axis) – Start monitor process.

check_monitor_ready(Axis, *Status) – check if monitor process finished.

get_monitor_data(Axis, *Data) – Get axis monitoring data.

get_instant_monitor_data(Axis, *Data_0, *Data_1, *Data_2, *Data_3) – Get instant monitor data

@ Description

`set_monitor_channel`:

This function is used to set axis data monitoring Channels. Each axis has 4 monitor channels. Each cannel can be set independently by parameter 'Channel_#'. The definition of value of 'Channel_#' is listed below:

Value	Description	Unit
FF	Not Used	
00	Feedback pulse accumulation	Pulse
01	(Reserved)	
02	Motor revolution speed	0.1rpm
03	(Reserved)	
04	Accumulated pulse	Pulse
05	(Reserved)	
06	Regenerative load factor	%
07	Execution load factor	%
08	Peak load factor	%
09	Bus voltage	
0A	Load inertia ratio	
0B	ABS counter	Rev
0C	Position within one revolution	Pulse
0D	(Reserved)	
0E	F/B present value	Pulse
0F	(Reserved)	
10	Position droop	Pulse
11	(Reserved)	
12	Speed command	0.1rpm
13	(Reserved)	0.1rpm
14	Speed feedback	0.1rpm
15	(Reserved)	0.1rpm
16	Current command	0.1%
17	Current feedback	0.1%

Table 1-7: Channel_# Definitions

Value	Description	Unit
18	ZCT (Bottom)	Pulse
19	(Reserved)	
1A	Present revolution counts	Rev
1B	Origin revolution counts	Rev
1C	Origin position within one revolution	Pulse
1D	(Reserved)	
1E	(Reserved)	
1F	(Reserved)	
20	Alarm status AL-1	
21	Alarm status AL-2	
22	Alarm status AL-3	
23	Alarm status AL-4	
24	Alarm status AL-5	
25	Alarm status AL-6	
26	Alarm status AL-7	
27	Alarm status AL-8	
28	Alarm status AL-9	
29	Alarm status AL-E	
2A	(Reserved)	
2B	(Reserved)	
2C	(Reserved)	
2D	(Reserved)	
2E	(Reserved)	
2F	(Reserved)	
30	Alarm history #1,#2	
31	Alarm history #3,#4	
32	Alarm history #5,#6	
33	Alarm history #7,#8	
34	Alarm history #9,#10	
35	(Reserved)	
36	(Reserved)	
37	(Reserved)	

Table 1-7: Channel_# Definitions

Value	Description	Unit
38	Parameter error NO.Pr01 to Pr16	
39	Parameter error NO.Pr17 to Pr32	
3A	Parameter error NO.Pr33 to Pr40	
3B	(Reserved)	
3C	(Reserved)	
3D	(Reserved)	
3E	(Reserved)	
3F	(Reserved)	
A5	INP (in position)	Active: 1, Inactive: 0
B0	Velocity Command	Pulse/sec
B2	DA1 Value	
B3	DA2 Value	
B4	Speed Feedback	
B6	External Encoder Feedback	
B8	Command Pulse	

Table 1-7: Channel_# Definitions

`set_monitor_config`:

This function is used to set configuration for axis data monitoring.

Parameter Name	Type	Description
Trigger_Select	I16	This variable is used to define the trigger source. Trigger_Select: ▶ Value = 0: No trigger ▶ Value = 1: CH0 as trigger source, going high ▶ Value = 2: CH1 as trigger source, going high ▶ Value = 3: CH2 as trigger source, going high ▶ Value = 4: CH3 as trigger source, going high ▶ Value = -1: CH0 as trigger source, going low ▶ Value = -2: CH1 as trigger source, going low ▶ Value = -3: CH2 as trigger source, going low ▶ Value = -4: CH3 as trigger source, going low
TriggerLevel	I32	Define the trigger level

Table 1-8: set_monitor_config Settings

Parameter Name	Type	Description
SamplePeriod	U32	This variable is used to define the sample period. ▶ Value = 1: 0.888 ms ▶ Value = 2: 2 * 0.888 ms ▶ Value = 3: 3 * 0.888 ms ▶ Value = 4: 4 * 0.888 ms
PreTriggerSam-pleNo	I16	Define the Number of samples before Trigger Value = 1 ~ 1023
SampleNumber	I16	Define the Total Number of samples Value = 1 ~ 1023

Table 1-8: set_monitor_config Settings

`start_monitor`:

After setting monitor channel and monitor configuration, user can call this function to inform 83XX to start monitor process.

`check_monitor_ready`:

This function is used to check if axis monitoring process in 83XX finished.

`get_monitor_data`:

After monitoring process finished, this function can be applied to retrieve axis monitoring data from 83XX. The parameter '*Data' is a 'I32' array with size = (4*SampleNumber).

`get_instant_monitor_data`:

This function is used to get axis monitor data at this instant. This function is always available regardless monitor configuration is set or not.

@ Syntax

C/C++ (DOS, Windows)

```
I16 set_monitor_channel(I16 Axis, U8 Channel_0,
     U8 Channel_1, U8 Channel_2, U8 Channel_3)
I16 set_monitor_config(I16 Axis, I16
     Trigger_Select, I32 Trigger_Level, U32
     SamplePeriod, I16 PreTriggerSampleNo,I16
     SampleNumber)
I16 start_monitor(I16 Axis)
```

```
I16 check_monitor_ready(I16 Axis, U8 *Status)
I16 get_monitor_data(I16 Axis, I32 *Data)
I16 get_instant_monitor_data(I16 Axis, I32
     *Data_0, I32 *Data_1, I32 *Data_2, I32
     *Data_3)
```

Visual Basic (Windows)

```
set_monitor_channel (ByVal Axis As Integer, ByVal
     Channel_0 As Byte, ByVal Channel_1 As Byte,
     ByVal Channel_2 As Byte, ByVal Channel_3 As
     Byte) As Integer
set_monitor_config (ByVal Axis As Integer, ByVal
     Trigger_Select As Integer, ByVal
     Trigger_Level As Long, ByVal SamplePeriod As
     Long, ByVal PreTriggerSampleNo As Integer,
     ByVal SampleNumber As Integer) As Integer
start_monitor (ByVal Axis As Integer) As Integer
check_monitor_ready (ByVal Axis As Integer,
     Status As Byte) As Integer
get_monitor_data (ByVal Axis As Integer, Data As
     Long) As Integer
get_instant_monitor_data (ByVal Axis As Integer,
     Data_0 As Long, Data_1 As Long, Data_2 As
     Long, Data_3 As Long) As Integer
```

@ Arguments

`Axis`: specified Axis index

`Channel_#`: to specify selection of monitor channel #.

`Trigger_Select`: Define the trigger source& direction.

`Trigger_Level`: Define the trigger level

`SamplePeriod`: Define the sample period

`PreTriggerSampleNo`: Define the number of samples before trigger

`SampleNumber`: Define number of samples

`Status`: status of monitoring process.

- Value = 1 (true), monitoring process is finished.
- Value = 0 (false), monitoring process is not finished yet.

`*Data`: servo monitor data.

***Data_#**: The servo monitor data in the instant when get_instant_monitor_data() was applied.

@ Return Code

```
ERROR_NoError
ERROR_Dll_Opened_By_Other_Process
ERROR_Wrong_Axis_Number
ERROR_Card_Not_Ready
ERROR_Card_Not_Initial
ERROR_DSP_Not_Ready
ERROR_Axis_Not_In_Control
ERROR_Monitor_Configuration_Not_Set
ERROR_Monitor_Process_Not_start
ERROR_Monitor_Process_Not_Finish
ERROR_Axis_Hand_Shake_Failed
```

@ Example

<C/C++ >

set_monitor_channel

```
I16 RetCode;
I16 Axis= 0;
U8 Channel_0 = 0xA5;
U8 Channel_1 = 0xB1;
U8 Channel_2 = 0x14;
U8 Channel_3 = 0x10;
RetCode  = set_monitor_channel(Axis, Channel_0,
     Channel_1, Channel_2, Channel_3);
```

set_monitor_config

```
I16 RetCode;
I16 Axis =0;
I16 Trigger_Select = 1;  // channel 0, going up
I32 Trigger_Level = 1;   // trigger level 1
U32 SamplePeriod = 1; // sample period = 1 *
     0.888 ms
I16 PreTriggerSampleNo = 100;   // pre trigger
     sample number = 100
I16 SampleNumber = 1023; // total sample number =
     1023
RetCode  = set_monitor_config(Axis,
     Trigger_Select, Trigger_Level,
```

```
        SamplePeriod, PreTriggerSampleNo,
        SampleNumber);
```

start_monitor

```
I16 RetCode;
I16 Axis = 0;
RetCode  = start_monitor(Axis);
```

check_monitor_ready

```
I16 RetCode;
I16 Axis = 0;
U8 Status;
RetCode  = check_monitor_ready(Axis, &Status);
```

get_monitor_data

```
I16 RetCode;
I16 Axis = 0;
I32 Data[1023];
RetCode  = get_monitor_data(I16 Axis, Data);
```

get_instant_monitor_data

```
I16 RetCode;
I16 Axis= 0;
I32 Data_0, Data_1, Data_2, Data_3;
RetCode  = get_instant_monitor_data(I16 Axis,
     &Data_0, &Data_1, &Data_2, &Data_3);
```

<Visual Basic>

set_monitor_channel

```
Dim RetCode As Integer
Dim Axis As Integer
Dim Channel_0 As Byte, Channel_1 As Byte
Dim Channel_2 As Byte, Channel_3 As Byte
Axis= 0;
Channel_0 = &hA5
Channel_1 = &hB1
Channel_2 = &h14
Channel_3 = &h10
RetCode  = set_monitor_channel(Axis, Channel_0,
     Channel_1, Channel_2, Channel_3)
```

set_monitor_config

```
Dim RetCode As Integer
```

```
Dim Axis As Integer
Dim Trigger_Select As Integer, Trigger_Level As
     Long
Dim SamplePeriod As Long, PreTriggerSampleNo As
     Integer
Dim SampleNumber As Integer
Axis= 0;
I16 Trigger_Select = 1 ‘channel 0, going up
I32 Trigger_Level = 1   ‘ trigger level 1
U32 SamplePeriod = 1 ‘ sample period = 1 * 0.888
     ms
I16 PreTriggerSampleNo = 100   ‘ pre trigger
     sample number = 100
I16 SampleNumber = 1023 ‘ total sample number =
     1023
RetCode  = set_monitor_config(Axis,
     Trigger_Select, Trigger_Level,
     SamplePeriod, PreTriggerSampleNo,
     SampleNumber)
```

start_monitor

```
Dim RetCode As Integer
Dim Axis As Integer
Axis= 0;
RetCode  = start_monitor(Axis)
```

check_monitor_ready

```
Dim RetCode As Integer
Dim Axis As Integer
Dim Status As Byte
Axis = 0
RetCode  = check_monitor_ready(Axis, Status);
```

get_monitor_data

```
Dim RetCode As Integer
Dim Axis As Integer
Dim Data(0 to 1022) As Long
Axis = 0;
RetCode  = get_monitor_data(I16 Axis, Data(0));
```

get_instant_monitor_data

```
Dim RetCode As Integer
Dim Axis As Integer
```

```
Dim Data_0 As Long, Data_1 As Long, Data_2 As
     Long
Dim Data_3 As Long
Axis= 0
RetCode  = get_instant_monitor_data(I16 Axis,
     Data_0, Data_1, Data_2, Data_3);
```

1.7.3 Servo information

@ Name

`get_servo_info(Axis,*ServoInfo)` – retrieve servo information

@ Description

`get_servo_info`:

This function is used to get current servo information. The meaning of individual bit of retrieved *ServoInfo is described at argument description.

@ Syntax

C/C++ (DOS, Windows)

```
I16 get_servo_info(I16 Axis, U32 * ServoInfo)
```

Visual Basic (Windows)

```
get_servo_info (ByVal Axis As Integer, ServoSts
     As Long) As Integer
```

@ Arguments

`Axis`: specified axis index

`*ServoInfo`: variable to carry out servo status information

- Bit 0: in course of Ready-ON
- Bit 1: in course of Servo-ON
- Bit 2: in course of in-Position
- Bit 3: in course of zero speed
- Bit 4: Pass through Z phase
- Bit 5: in course of Torque limit
- Bit 6: in course of Alarm
- Bit 7: in course of warning
- Bit 8: Absolute position reference point data set

- Bit 9: Absolute position reference point data set wrong
- Bit 10 ~ 14: Reserved
- Bit 15: in course of Speed limit
- Bit 16 ~ 31 : Reserved

@ Return Code

```
ERROR_NoError
ERROR_Dll_Opened_By_Other_Process
ERROR_Wrong_Axis_Number
ERROR_Card_Not_Ready
ERROR_Card_Not_Initial
ERROR_DSP_Not_Ready
ERROR_Axis_Not_In_Control
```

@ Example

<C/C++ >

`get_servo_info`

```
I16 RetCode;
I16 Axis = 0;
U32 ServoInfo;
RetCode  = get_servo_info(Axis, &ServoInfo);
```

<Visual Basic>

`get_servo_info` –

```
Dim RetCode As Integer
Dim Axis As Integer, ServoInfo As Long
Axis = 0
RetCode  = get_servo_info(Axis, ServoInfo)
```

1.7.4 Servo ON

@ Name

`set_servo_on(Axis, ON_OFF)` – set servo on

@ Description

`set_servo_on`:

This function is used to command servo driver of specified axis to starts controlling its servomotor. Then motion function could be applied on this axis.

@ Syntax

C/C++ (DOS, Windows)

```
I16 set_servo_on(I16 Axis, I16 ON_OFF)
```

Visual Basic (Windows)

```
set_servo_on (ByVal Axis As Integer, ByVal ON_OFF
     As Integer) As Integer
```

@ Arguments

`Axis`: specified axis index

`ON_OFF`: variable to control servo on or off

- ON_OFF = 0, servo off
- ON_OFF = 1, servo on

@ Return Code

```
ERROR_NoError
ERROR_Dll_Opened_By_Other_Process
ERROR_Wrong_Axis_Number
ERROR_Card_Not_Ready
ERROR_Card_Not_Initial
ERROR_DSP_Not_Ready
ERROR_Axis_Not_In_Control
ERROR_Axis_Servo_Alarm
```

@ Example

<C/C++ >

set_servo_info

```
I16 RetCode;
I16 Axis = 0;
I16 ON_OFF = 1;
RetCode  = set_servo_on(Axis, ON_OFF);
```

<Visual Basic>

set_servo_info

```
Dim RetCode As Integer
Dim Axis As Integer, ON_OFF As Long
Axis = 0
ON_OFF = 1
RetCode  = set_servo_on(Axis, ON_OFF)
```

1.7.5 Driver information

@ Name

`understand_driver()` – retrieve servo driver classification code

`understand_motor()` – retrieve servo motor information

@ Description

`understand_driver`:

This function is used to get servo driver's classification code

`understand_motor`:

This function is used to get servo motor's information, including motor type, motor capacity, rated revolution speed… etc. please refer to description of function call parameters.

@ Syntax

C/C++ (DOS, Windows)

```
I16 understand_driver(I16 Axis, U16 *Class_Code,
      U16 *SoftwareNo)
I16 understand_motor(I16 Axis, U16* MotorType,
      U16* Capacity, U16* RateRPM, U16*
      RateCurrent, U16* MaxRPM, U16* MaxTorq, U32*
      PPR, U16* ENCInfo, U16* OptionalInfor)
```

Visual Basic (Windows)

```
understand_driver (ByVal Axis As Integer,
      Class_Code As Integer, SoftwareNo As
      Integer) As Integer
understand_motor (ByVal Axis As Integer,
      MotorType As Integer, Capacity As Integer,
      RateRPM As Integer, RateCurrent As Integer,
      MaxRPM As Integer, MaxTorq As Integer, PPR
      As Long, ENCInfo As Integer, OptionalInfor
      As Integer) As Integer
```

@ Arguments

`Axis`: specified axis index

***Class_Code**: variable to carry out servo driver's classification code

***SoftwareNo**: Array to carry out servo driver's Software number

***MotorType**: variable to carry out servo motor's type information

***Capacity**: variable to carry out servo motor's capacity information

***RateRPM**: variable to carry out servo motor's Rated Revolution Speed information

***RateCurrent**: variable to carry out servo motor's Rated Crrent information

***MaxRPM**: variable to carry out servo motor's Maximum Revolution number

***MaxTorq**: variable to carry out servo motor's Maximum Torque information

***PPR**: variable to carry out servo motor's Encoder Pulse Per revolution

***ENCInfo**: variable to carry out motor's Encoder type information, 0 for incremental , 1 for absolute

***OptionalInfor**: variable to carry out motor's Optional information

@ Return Code

```
ERROR_NoError
ERROR_Dll_Opened_By_Other_Process
ERROR_Wrong_Axis_Number
ERROR_Card_Not_Ready
ERROR_Card_Not_Initial
ERROR_DSP_Not_Ready
ERROR_Axis_Not_In_Control
```

@ Example

<C/C++ >

understand_driver

```
I16 RetCode;
I16 Axis = 0;
```

```
U16 Class_Code;
U16 SoftwareNo[6];
RetCode  = understand_driver(Axis, &Class_Code,
     SoftwareNo)
```

understand_motor

```
I16 RetCode;
I16 Axis= 0;
U16 MotorType, Capacity, RateRPM, RateCurrent,
     MaxRPM, MaxTorq
U32 PPR;
U16 ENCInfo, OptionalInfor;
RetCode  = understand_motor (Axis, &MotorType,
     &Capacity, &RateRPM, &RateCurrent, &MaxRPM,
     &MaxTorq, &PPR, &ENCInfo, &OptionalInfor)
```

<Visual Basic>

understand_driver

```
Dim RetCode As Integer
Dim Axis As Integer, Class_Code As Integer
Dim SoftwareNo(0 to 5) As Integer
Axis = 0
RetCode  = understand_driver(Axis, Class_Code,
     SoftwareNo(0))
```

understand_motor

```
Dim RetCode As Integer
Dim Axis As Integer
Dim MotorType As Integer, Capacity As Integer,
     RateRPM As Integer,
Dim RateCurrent As Integer, MaxRPM As Integer,
     MaxTorq As Integer
Dim PPR As Long
Dim ENCInfo As Integer, OptionalInfor As Integer
Axis = 0
RetCode  = understand_motor (Axis, MotorType,
     Capacity, RateRPM, RateCurrent, MaxRPM,
     MaxTorq, PPR, ENCInfo, OptionalInfor)
```

1.7.6 Alarm reset

@ Name

`get_alarm_no(Axis, *AlarmNo)` – get alarm number.

`alarm_reset(Axis)` – servo alarm reset

@ Description

`get_alarm_no`:

When servo alarm occurs, this function can help to get alarm number.

`alarm_reset`:

When servo alarm occurs, servo motor will stop moving. After the alarm condition passed, this function can help to clear alarm and reset servo.

@ Syntax

C/C++ (DOS, Windows)

```
I16 get_alarm_no(I16 Axis, U8* AlarmNo)
I16 alarm_reset(I16 Axis)
```

Visual Basic (Windows)

```
alarm_reset (ByVal Axis As Integer) As Integer
get_alarm_no (ByVal Axis As Integer, AlarmNo As
     Byte) As Integer
```

@ Arguments

`Axis`: specified axis index

`*AlarmNo`: variable to carry out alarm number

@ Return Code

```
ERROR_NoError
ERROR_Dll_Opened_By_Other_Process
ERROR_Wrong_Axis_Number
ERROR_Card_Not_Ready
ERROR_Card_Not_Initial
ERROR_DSP_Not_Ready
ERROR_Axis_Not_In_Control
```

@ Example

<C/C++ >

get_alarm_no

```
I16 RetCode;
I16 Axis = 0;
U8 AlarmNo;
RetCode  = get_alarm_no(Axis, &AlarmNo);
```

alarm_reset–

```
I16 RetCode;
I16 Axis = 0;
RetCode  = alarm_reset(I16 Axis);
```

<Visual Basic>

get_alarm_no

```
Dim RetCode As Integer
Dim Axis As Integer, AlarmNo As Byte
Axis = 0
RetCode  = get_alarm_no(Axis, AlarmNo)
```

alarm_reset

```
Dim RetCode As Integer
Dim Axis As Integer
Axis = 0
RetCode  = alarm_reset(Axis)
```

1.8 Control Gain Tuning

@ Name

set_auto_tune(Axis, Mode, RSP, GD2) – Set auto tuning mode and parameters

get_auto_tune(Axis, *Mode, *RSP, *GD2) – Read auto tuning mode and parameters

set_control_gain(Axis, PG1, VG1, VIC,PG2, VG2, FFC) – Set control gain value

get_control_gain(Axis, *PG1, *VG1, *VIC,*PG2, *VG2, *FFC) – Set control gain value

set_notch_filter(Axis, Mode, NotchFrequency, NotchDepth) – Set machine resonance suppression filter

get_notch_filter(Axis, *Mode, *NotchFrequency, *NotchDepth) –Get current machine resonance suppression filter value

set_LP_filter(Axis,ON_OFF) – Enable/Disable low pass filter

get_LP_filter(Axis,*ON_OFF) – get current low pass filter mode

@ Description

set_auto_tune:

This function is used to select auto-tuning mode and set RS, GD2 parameter. There are 5 auto-tuning modes available, and it is specified by the 2nd parameter. When mode 1 or 3 was selected, the servo driver automatically determines all control gains. If mode 2,4, or 0 was selected, control gains is of user's choice by using

set_control_gain(). Please refer tothe MR-J2S-B servo driver instruction manual for more information.

Mode Value	Description	RSP	GD2	PG1	VG1	VIC	PG2	VG2	FFC
1	Auto-Tuning mode 1	M	A	A	A	A	A	A	A
2	Manual mode 2	--	M	M	M	M	M	M	M
3	Auto-Tuning mode 2	M	M	A	A	A	A	A	A
4	Manual mode 1	--	M	M	A	M	A	M	A
0*	Interpolation mode	--	A	M	M	A	A	A	A

Table 1-9: Mode Values

- A: Set automatically
- M: Allow set by function calls

Note: Mode 0 is normally not used.

`get_auto_tune`:

This function is used to read current auto-tuning mode selection, and RSP, GD2 parameter setting. Please refer tothe MR-J2S-B servo driver instruction manual for more information.

`set_control_gain`:

This function is to set control gains. This function is valid only when 'Mode' is set_auto_tune() is set to 2,4, or 0. Please refer to section 4.9 for control gains introduction. Please refer tothe MR-J2S-B servo driver instruction manual for more information.

`get_control_gain`:

This function is used to read current control gains setting. Please refer tothe MR-J2S-B servo driver instruction manual for more information.

`set_notch_filter`:

This function is to set match resonance suppression filter (Notch Filter). There are 6 modes available, and it is specified by the 2nd parameter; there are 32 notch frequency selection, and it is specified by the 3rd parameter; there are 4 notch depth selection, and it is specified by the 4th parameter. Please refer tothe MR-J2S-B servo driver instruction manual for more information.

`get_notch_filter`:

This function is used to read current notch filter configuration. Please refer tothe MR-J2S-B servo driver instruction manual for more information.

`set_LP_filter`:

This function is to enable/disable the low pass filter function for torque command. Please refer tothe MR-J2S-B servo driver instruction manual for more information.

`get_LP_filter`:

This function is to read current setting for low pass filter function. Please refer tothe MR-J2S-B servo driver instruction manual for more information.

@ Syntax

C/C++ (DOS, Windows)

```
I16 set_auto_tune(I16 Axis, U16 Mode, U16 RSP,
     U16 GD2)
I16 get_auto_tune(I16 Axis, U16 *Mode, U16 *RSP,
     U16 *GD2)
I16 set_control_gain(I16 Axis, U16 PG1, U16 VG1,
     U16 VIC, U16 PG2, U16 VG2, U16 FFC)
I16 get_control_gain(I16 Axis, U16 *PG1, U16
     *VG1, U16 *VIC, U16 *PG2,U16 *VG2, U16 *FFC)
I16 set_notch_filter(I16 Axis, U16 Mode, U16
     NotchFrequency, U16 NotchDepth)
I16 get_notch_filter(I16 Axis, U16 *Mode, U16
     *NotchFrequency, U16 *NotchDepth)
I16 set_LP_filter(I16 Axis,I16 ON_OFF)
I16 get_LP_filter(I16 Axis,I16 *ON_OFF)
```

Visual Basic (Windows)

```
set_auto_tune (ByVal Axis As Integer, ByVal Mode
     As Integer, ByVal RSP As Integer, ByVal GD2
     As Integer) As Integer
get_auto_tune (ByVal Axis As Integer, Mode As
     Integer, RSP As Integer, GD2 As Integer) As
     Integer
set_control_gain (ByVal Axis As Integer, ByVal
     PG1 As Integer, ByVal VG1 As Integer, ByVal
     VIC As Integer, ByVal PG2 As Integer, ByVal
```

```
      VG2 As Integer, ByVal FFC As Integer) As
      Integer
get_control_gain (ByVal Axis As Integer, PG1 As
      Integer, VG1 As Integer, VIC As Integer, PG2
      As Integer, VG2 As Integer, FFC As Integer)
      As Integer
set_notch_filter (ByVal Axis As Integer, ByVal
      Mode As Integer, ByVal NotchFrequency As
      Integer, ByVal NotchDepth As Integer) As
      Integer
get_notch_filter (ByVal Axis As Integer, Mode As
      Integer, NotchFrequency As Integer,
      NotchDepth As Integer) As Integer
set_LP_filter (ByVal Axis As Integer, ByVal
      ON_OFF As Integer) As Integer
get_LP_filter (ByVal Axis As Integer, ON_OFF As
      Integer) As Integer
```

@ Arguments

Axis: specified axis index

Mode: specified Auto-Tuning mode

RSP: specified auto tuning response level setting

RSP value (in Hex)	1	2	3	4	5 *	6	7	8	9	A	B	C	D	E	F
Machine response frequency guideline	15Hz	20Hz	25Hz	30Hz	35Hz*	45Hz	55Hz	70Hz	85Hz	105Hz	130Hz	160Hz	200Hz	240Hz	300Hz

Table 1-10: RSP Tuning Settings

Note: *Default setting value

GD2: specified ratio of load inertia moment to servo motor inertia moment.

- Value range: 0 ~ 3000 (unit 0.1 times)
- Default setting: 70

PG1: position loop gain 1.

- Value range: 4 ~ 2000 (rad/s)
- Default setting: 35

VG1: velocity loop gain 1.

- Value range: 20 ~ 8000 (rad/s)
- Default setting: 177

VIC: velocity integral compensation.

- Value range: 1 ~ 1000 (ms)
- Default setting: 48

PG2: position loop gain 2.

- Value range: 1 ~ 500 (rad/s)
- Default setting: 35

VG2: velocity loop gain 2.

- Value range: 20 ~ 20000 (rad/s)
- Default setting: 817

FFC: Velocity feed foreword gain.

- Value range: 0 ~ 100 (%)
- Default setting: 0

Mode: specified notch filter modes

NotchFrequency: specified notch frequency

NotchDepth: specified notch depth

Setting	Depth(Gain)
0	-40db
1	-14db
2	-8db
3	-4db

Table 1-11: Notch Depth

ON_OFF: Enable/Disable selection

- ON_OFF = 1, Enabled
- ON_OFF = 0, Disabled

@ Return Code

```
ERROR_NoError
```

```
ERROR_Dll_Opened_By_Other_Process
ERROR_Wrong_Axis_Number
ERROR_Card_Not_Ready
ERROR_Card_Not_Initial
ERROR_DSP_Not_Ready
ERROR_Axis_Not_In_Control
ERROR_Axis_Servo_Alarm
ERROR_Axis_Is_In_Servo_ON
ERROR_Servo_Parameter_Set_Failed
ERROR_Servo_Parameter_Get_Failed
ERROR_Invaild_Auto_Tune_Mode
```

@ Example

<C/C++ >

set_auto_tune

```
I16 RetCode;
I16 Axis =0;
U16 Mode = 1;
U16 RSP = 6;   // 45 Hz
U16 GD2 = 70;  // 7 times
RetCode  = set_auto_tune(Axis, Mode, RSP, GD2);
```

get_auto_tune

```
I16 RetCode;
I16 Axis = 0;
U16 Mode, RSP, GD2;
RetCode  = get_auto_tune(Axis, &Mode, &RSP,
     &GD2);
```

set_control_gain

```
I16 RetCode;
I16 Axis = 0;
U16 PG1 = 45;
U16 VG1 = 200;
U16 VIC = 48 ;
U16 PG2 = 35;
U16 VG2 = 817;
U16 FFC = 0;
RetCode  = set_control_gain(Axis, PG1, VG1, VIC,
     PG2, VG2, FFC);
```

get_control_gain

```
I16 RetCode;
I16 Axis = 0;
U16 PG1, VG1, VIC, PG2, VG2, FFC;
RetCode  = get_control_gain(Axis, &PG1, &VG1,
     &VIC, &PG2, &VG2, &FFC);
```

set_notch_filter

```
I16 RetCode;
I16 Axis =0 ;
U16 Mode = 1;
U16 NotchFrequency = 5;
U16 NotchDepth = 1;
RetCode  = set_notch_filter(Axis, Mode,
     NotchFrequency, NotchDepth);
```

get_notch_filter

```
I16 RetCode;
I16 Axis = 0;
U16 Mode, NotchFrequency, NotchDepth;
RetCode  = get_notch_filter(Axis, &Mode,
     &NotchFrequency, &NotchDepth);
```

set_LP_filter

```
I16 RetCode;
I16 Axis = 0;
I16 ON_OFF = 0;
RetCode  = set_LP_filter(Axis, ON_OFF);
```

get_LP_filter–

```
I16 RetCode;
I16 Axis = 0;
I16 ON_OFF;
RetCode  = get_LP_filter(Axis,&ON_OFF);
```

<Visual Basic>

set_auto_tune –

```
Dim RetCode As Integer
Dim Axis As Integer
Dim Mode As Integer, RSP As Integer, GD2 As
     Integer
Axis = 0;
Mode = 1;
RSP = 6;   // 45 Hz
```

```
GD2 = 70;  // 7 times
RetCode  = set_auto_tune(Axis, Mode, RSP, GD2)
get_auto_tune –
Dim RetCode As Integer
Dim Axis As Integer, Mode As Integer, RSP As
     Integer, GD2 As Integer
Axis = 0
RetCode  = get_auto_tune(Axis, Mode, RSP, GD2)
```

set_control_gain

```
Dim RetCode As Integer
Dim Axis As Integer, PG1 As Integer, VG1 As
     Integer, VIC As Integer,
Dim PG2 As Integer, VG2 As Integer, FFC As
     Integer
Axis = 0
PG1 = 45
VG1 = 200
VIC = 48
PG2 = 35
VG2 = 817
FFC = 0
RetCode  = set_control_gain(Axis, PG1, VG1, VIC,
     PG2, VG2, FFC)
```

get_control_gain

```
Dim RetCode As Integer
Dim Axis As Integer, PG1 As Integer, VG1 As
     Integer, VIC As Integer,
Dim PG2 As Integer, VG2 As Integer, FFC As
     Integer
Axis = 0
RetCode  = get_control_gain(Axis, PG1, VG1, VIC,
     PG2, VG2, FFC)
```

set_notch_filter

```
Dim RetCode As Integer
Dim Axis As Integer, Mode As Integer,
     NotchFrequency As Integer,
Dim NotchDepth As Integer
Axis = 0
Mode = 1
NotchFrequency = 5
```

```
NotchDepth = 1
RetCode  = set_notch_filter(Axis, Mode,
     NotchFrequency, NotchDepth)
```

get_notch_filter

```
Dim RetCode As Integer
Dim Axis As Integer, Mode As Integer,
     NotchFrequency As Integer,
Dim NotchDepth As Integer
RetCode  = get_notch_filter(Axis, Mode,
     NotchFrequency, NotchDepth)
```

set_LP_filter

```
Dim RetCode As Integer
Dim Axis As Integer, ON_OFF As Integer
Axis = 0
ON_OFF = 0
RetCode  = set_LP_filter(Axis, ON_OFF);
```

get_LP_filter

```
Dim RetCode As Integer
Dim Axis As Integer, ON_OFF As Integer
Axis = 0
RetCode  = get_LP_filter(Axis,ON_OFF)
```

1.9 Interrupt Control Function

@ Name

`int_control(CardID, Flag)` – Enable/Disable INT service

`set_int_factor(CardID, Source, IntFactor)` – Set INT factor

`get_int_status(CardID, Source, *IntStatus)` – Get INT Status

`set_int_event(CardID,*HEvent)` – Enable event

`link_interrupt(CardID, *callbackAddr)` – setup call back unction

`set_int_event2(CardID,*TotalEvent)` – Enable all Interrupt factor events

`clear_tlc_int(Axis)` – clear TLC interrupt status

`set_timer_int_interval(CardID, Sec)` – set timer interrupt interval in seconds

@ Description

`int_control`:

This function is used to enable interrupt generating to host PC.

`set_int_factor`:

This function allows users to select factors to initiate the interrupt. The first parameter defines the applied cardID, the second define the source, and the third defines the interrupt factor.

- Source = 0 - 11 for Axis 0 - Axis 11 respectively.

Bit of 'IntFactor'	Name	Description
0	PEL	Positive Limit Switch
1	MEL	Negative Limit Switch
2	ORG	Home Switch
3	RDY	Servo Ready
4	INP	In Position
5	EZ	Index signal passed
6	ZSPD	Zero Speed
7	TLC	Torque Limit reached
8	ALM	Alarm signal on
9	WRN	Servo Warning on
10	HOME	Home Move completed
11	MTC	Motion Completed
12	CPBF	Curve Parameter Buffer Full
13	EPD	Position deviation is too large
14	CMP1	Position compare1 is true
15	CMP2	Position compare2 is true

Table 1-12: IntFactor Bits for Source 0-11

- Source = 12 for system interrupt

Bit of 'IntFactor'	Description
0	System Error
1	Emergency Stop
2	Cyclic Timer Interrupt

Table 1-13: IntFactor Bits for Source 12

- Source = 13 for GPIO interrupt

Bit of 'IntFactor'	Description
0	General purposed DI, Channel 0
1	General purposed DI, Channel 1
8	Compare_Counter_CH0
9	Compare_Counter_CH1
10	Compare_Counter_CH2

Table 1-14: IntFactor Bits for Source 13

`get_int_status`:

This function allows user to identify what cause the interrupt signal. After interrupt, user can call this function to check what causes interrupt. The returned interrupt status information has exactly the same definition as described in previous function.

`set_int_event`:

This function is used to assign the window INT event.

`int_disable`:

This function is used to disable the window INT event.

`set_int_event2`:

This function is used for creating an event mapping array for all interrupt factors of this board. There are 14 interrupt sources in one SSCNET board. Each source has 16 interrupt factors. Therefore, total events on one board are 224. Users must declare an event array which has 224 events. Each event represent one interrupt factor. Users can use WaitForSingleOobject WINAPI function to wait one specific event without using get_int_status() function to check. This method shorten the time on an event-driven based control system.

`clear_tlc_int`:

This function is used to clear TLC interrupt status. TLC means torque limit reached.

`set_timer_int_interval`:

This function is used to set the timer interrupt interval in sec. The minimum value is 0.000888sec. In Windows system, please don't use the timer inverval less than 1ms because it is not a real time OS.

@ Syntax

C/C++ (DOS, Windows)

```
I16 int_control(I16 Card, I16 Flag)
I16 set_int_factor(I16 Card, I16 Source, U16
     IntFactor)
I16 get_int_status(I16 Card, I16 Source, U16
     *IntStatus)
I16 set_int_event(I16 Card, HANDLE *HEvent)
I16 link_interrupt(I16 Card,void ( __stdcall
     *callbackAddr)( I16 IntAxisInCard))
I16 set_int_event2(I16 CardID, HANDLE
     *TotalEvent)
I16 clear_tlc_int(I16 Axis);
I16 set_timer_int_interval(I16 CardID, F64 Sec);
```

Visual Basic (Windows)

```
int_control (ByVal Card As Integer, ByVal Flag As
     Integer) As Integer
set_int_factor (ByVal Card As Integer, ByVal
     Source As Integer, ByVal IntFactor As
     Integer) As Integer
get_int_status (ByVal Card As Integer, ByVal
     Source As Integer, IntStatus As Integer) As
     Integer
set_int_event (ByVal Card As Integer, hEvent As
     Long) As Integer
link_interrupt (ByVal Card As Integer, ByVal
     lpCallBackProc As Long) As Integer
set_int_event2 (ByVal CardID As Integer,
     TotalEvent As Long) As Integer
clear_tlc_int (ByVal Axis As Integer) As Integer
set_timer_int_interval (ByVal CardID As Integer,
     ByVal Sec As Double) As Integer
```

@ Arguments

`CardID`: The SSCNet series card index number.

Axis: specified axis index 0,1,2,3,4…

Flag: interrupt flag, 0 or 1 (0: Disable, 1:Enable)

Source: interrupt source,

- 0 - 11: for axis 0 - 11 respectively
- 12: for DSP system
- 13: for GPIO

IntFactor: interrupt factor, refer to previous table

***IntStatus**: interrupt status, refer to previous table

***HEvent**: event handler (for Windows)

***callbackAddr**: Call back function address.

TotalEvent: Event[224] pointer

Sec: timer interrupt interval in second, it is a 0.888ms 16bit counter.

@ Return Code

```
ERROR_NoError
ERROR_Dll_Opened_By_Other_Process
ERROR_Card_Not_Exist
ERROR_PCIBios_Not_Exist
ERROR_Card_Reinitialized
ERROR_Card_Not_Exist
ERROR_Card_Not_Accessible
ERROR_Card_Not_Ready
ERROR_DSP_Not_Ready
ERROR_DSP_Initial_Time_Out
ERROR_Unknow_Card_Type
```

@ Example

<C/C++ >

```
int_control –
I16 RetCode;
I16 Card = 0;
I16 Flag = 1;    //Enable interrupt
RetCode  = int_control(CardID, Flag);
```

set_int_factor–

```
I16 RetCode;
I16 Card = 0;
I16 Source = 1; // Axis 1
U16 IntFactor= 0x800;// Motion complete
RetCode  = set_int_factor(CardID, Source,
     IntFactor);
```

get_int_status

```
I16 RetCode;
I16 Card = 0;
I16 Source = 1; // Axis 1
U16 IntStatus;
RetCode  = get_int_status(CardID, Source,
     &IntStatus);
```

set_int_event

```
I16 RetCode;
I16 Card = 0;
HANDLE HEvent[14];
RetCode  = set_int_event (CardID, HEvent);
```

link_interrupt

```
I16 RetCode;
I16 Card = 0;
RetCode  = link_interrupt (CardID, CallBackProc);
...
__stdcall CallBackProc ( I16 Source)
...
}
```

set_int_event2

```
HANDLE hTotalEvent[224];
I16 RetCode;
set_int_event2(CARD0, hTotalEvent);
int_control(CARD0,1);
set_int_factor(CARD0, Source_Axis3, 0x10); //
     Axis3 INP
start_tr_move(Axis3, 50, 0, 100, 0, 0.01, 0.01)
WaitForSingleObject(hTotalEvent[16*Axis3 + 4],
     INFINITE ); // INP
ResetEvent(hTotalEvent[16*Axis3 + 4]);
```

<Visual Basic>

int_control

```
Dim RetCode As Integer
Dim Card As Integer, Flag As Integer
Card = 0;
Flag = 1;    ‘Enable interrupt
RetCode  = int_control(CardID, Flag)
```

set_int_factor

```
Dim RetCode As Integer
Dim Card As Integer, Source As Integer, IntFactor
     As Integer
Card = 0
Source = 1 ‘ Axis 1
IntFactor= 0x800‘ Motion complete
RetCode  = set_int_factor(CardID, Source,
     IntFactor);
```

get_int_status

```
Dim RetCode As Integer
Dim Card As Integer, Source As Integer, IntStatus
     As Integer
Card = 0
Source = 1 ‘ Axis 1
RetCode  = get_int_status(CardID, Source,
     IntStatus)
```

set_int_event

```
Dim RetCode As Integer
Dim Card As Integer, Hevent(0 to 13) As Long
Card = 0
RetCode  = set_int_event (CardID, Hevent(0));
```

link_interrupt

```
Dim RetCode As Integer
Dim Card As Integer, Hevent(0 to 13) As Long
Card = 0
RetCode  = link_interrupt (CardID, AddressOf
     CallBackProc);
...
Public Function CallBackProc (ByVal Source As
     Integer)
...
End Function
```

1.10 Position Compare Function

@ Name

set_compare(Axis, CMP1Pos, CMP1Dir, CMP2Pos, CMP2Dir) – set position compare

check_compare(Axis, *status) – check position comparing status

set_compare_source – set position comparator source

set_compare_channel(Axis, Channel, CMPPos, CMPDir) – set compare config by channel

set_single_compare(Axis, Channel, CMP_Pos) – set compare position by channel

map_dout_and_comparator(CardID, DOut_Ch, AxisNo, CompNo, DOut_mode) – mapping digital output channel as trigger output

set_compare_table_dir(CardID, Table_ChNo, Dir) – change compare data table's direction

link_dout_and_compare_table(CardID, DO_ChNo, StartI, EndI, *Table_Data) – select trigger output's compare data table

set_ext_encoder_compare_method(CardID, EncNo, Mode) – set external encoder's compare method

set_ext_encoder_compare_value(CardID, EncNo, CompareValue) – set external encoder's compare value

@ Description

set_compare:

This function is used to set up position comparing for specified axis. Every motion axis of PCI-83XX Series has 2 position compare channel. It compares feedback position with desired position.

check_compare:

This function is used to check position-comparing status. If the comparing has already come into existence, the relative bit of sta-

tus becomes ‘1’, otherwise ‘0’. All bit of status will be set to ‘0’ every time when set_compare() function is executed.

set_compare_source:

This function is used to select the source of position compare function. There are two sources of compare function: motor’s feedback position or controller’s command position.

set_compare_channel:

This function is used to set compare position and direction by channel. This function has fewer parameters than set_compare() function. It needs less time than set_compare() function.

set_single_compare:

This function is used to set compare position by channel. This function has fewer parameters than set_compare() function. It needs less time than set_compare() function.

map_dout_and_comparator:

This function is used to mapping digital output pin to one of comparator in a specific axis. Each axis has two position comparators and each card has only two digital output pin. Users must tell the card mapping relation.

set_compare_table_dir:

This function is used to set compare table’s reloading direction. The compare data is static. When running continuous compare trigger function, on board DSP must know reloading direction of the data table. This function tells what reloading direction must be run. It reduce compare data size for two directions. Users need only build one compare data series plus a direction parameter instead of two compare data series table by difference directions.

link_dout_and_compare_table:

This function is used to link a static compare table data and attach it to one digital output channel. Also you can assign the starting index and ending index from the static compare table. After calling this function, the data from start index to end index will be transferred to SSCNET board. The talbe points for SSCNET Board must be less than 100.

set_ext_encoder_compare_method:

This function is used to set external encoder's compare method.

set_ext_encoder_compare_value:

This function is used to set external encoder's compare value.

@ Syntax

C/C++ (DOS, Windows)

```
I16 set_compare(I16 Axis, F64 CMP1Pos, I16
     CMP1Dir, F64 CMP2Pos, I16 CMP2Dir)
I16 check_compare (I16 Axis, I16* status)
I16 set_compare_source(I16 Axis, I16 Source)
I16   set_compare_channel(I16 Axis, I16 Channel,
     F64 CMPPos, I16 CMPDir);
I16   set_single_compare(I16 Axis, I16 Channel,
     F64 CMPPos);
I16   map_dout_and_comparator(I16 CardID, I16
     DOut_Ch, I16 Axis, I16 Channel,I16
     DOut_mode);
I16   set_compare_table_dir(I16 CardID, I16
     Table_ChNo, I16 CMPDir );
I16 link_dout_and_compare_table(I16 CardID, I16
     DO_ChNo, I16 StartI,I16 EndI, F32
     *Table_Data);
I16 set_ext_encoder_compare_method(I16 CardID,
     I16 EncNo, U16 Mode);
I16 set_ext_encoder_compare_value(I16 CardID, I16
     EncNo, I32 CMPPos);
```

Visual Basic (Windows)

```
set_compare (ByVal Axis As Integer, ByVal CMP1Pos
     As Double, ByVal CMP1Dir As Integer, ByVal
     CMP2Pos As Double, ByVal CMP2Dir As Integer)
     As Integer
check_compare (ByVal Axis As Integer, status As
     Integer) As Integer
set_compare_source (ByVal Axis As Integer, ByVal
     Source As Integer) As Integer
set_compare_channel(ByVal Axis As Integer, ByVal
     Channel As Integer, ByVal CMPPos As Double,
     ByVal CMPDir As Integer) As Integer
```

```
set_single_compare(ByVal Axis As Integer, ByVal
      Channel As Integer, ByVal CMPPos As Double)
      As Integer
map_dout_and_comparator(ByVal CardID As Integer,
      ByVal DOut_Ch As Integer, ByVal Axis As
      Integer, ByVal Channel As Integer,ByVal
      DOut_mode As Integer) As Integer
set_compare_table_dir(ByVal CardID As Integer,
      ByVal Table_ChNo As Integer, ByVal CMPDir As
      Integer) As Integer
link_dout_and_compare_table(ByVal CardID As
      Integer, ByVal DO_ChNo As Integer, ByVal
      StartI As Integer, ByVal EndI As Integer,
      Table_Data As Single) As Integer
set_ext_encoder_compare_method(ByVal CardID As
      Integer, ByVal EncNo As Integer, ByVal Mode
      As Integer) As Integer
set_ext_encoder_compare_value(ByVal CardID As
      Integer, ByVal EncNo As Integer, ByVal
      CMPPos As Long) As Integer
```

@ Arguments

`Axis`: specified axis index

`CMP1Pos`: Desired position for compare channel1 in unit of mm.

`CMP2Pos`: Desired position for compare channel1 in unit of mm.

`CMP1Dir`: select comparing direction for compare channel 1

`CMP2Dir`: select comparing direction for compare channel 1

- CMP#Dir = 0, disable
- CMP#Dir = 1, feedback > ComparePos
- CMP#Dir = 2: feedback >= ComparePos
- CMP#Dir = 3, feedback < ComparePos
- CMP#Dir = 4: feedback <= ComparePos
- CMP#Dir = 5, feedback == ComparePos

`*status`: return of compare startus

- bit 0 , for compare channel 1
- bit 1 , for compare channel 2

Source: 0 means feedback, 1 means command

CardID: The SSCNet series card index number.

Channel: compare channel inside axis, 0=CMP1, 1=CMP2

CMPPos: compare position

CMPDir: compare table direction, 0=increasing, 1=decreasing

DOut_Mode: digital output mode, 0=normal low, 1=normal high

DOut_Ch: digital output channel selection, 0=DO1, 1=DO2

Table_ChNo: Table Attached DO number

StartI: start index of compare table

EndI: end index of compare table, (EndI-StartI)<100

***Table_Data**: static table data pointer created by users

EncNo: external encoder number

Mode: compare method= 1, 2:greater or equal, 3,4: less or equal

@ Return Code

```
ERROR_NoError
ERROR_Dll_Opened_By_Other_Process
ERROR_Wrong_Axis_Number
ERROR_Card_Not_Ready
ERROR_Card_Not_Initial
ERROR_DSP_Not_Ready
ERROR_Axis_Not_In_Control
ERROR_Wrong_CMP_Channel
ERROR_Wrong_DIO_Channel_Number
ERROR_Wrong_Compare_Talbe_Count
```

@ Example

<C/C++ >

set_compare

```
I16 RetCode;
I16 Axis = 1;
F64 CMP1Pos = 10.0;// 10.0 mm
F64 CMP2Pos = -10.0;// -10.0 mm
I16 CMP1Dir = 1;// feedback > ComparePos
I16 CMP2Dir = 3;// feedback < ComparePos
```

```
RetCode  = set_compare(Axis, CMP1Pos, CMP1Dir,
     CMP1Pos, CMP2Dir);
```

check_compare

```
I16 RetCode;
I16 Axis = 1;
I16 status;
RetCode  = check_compare (Axis, &status);
```

<Visual Basic>

set_compare

```
Dim RetCode As Integer
Dim Axis As Integer, CMP1Pos As Double, CMP1Dir
     As Integer
Dim CMP2Pos As Double, CMP2Dir As Integer
Axis = 1
CMP1Pos = 10.0                  ‘ 10.0 mm
CMP2Pos = -10.0‘ -10.0 mm
CMP1Dir = 1‘ feedback > ComparePos
CMP2Dir = 3‘ feedback < ComparePos
RetCode  = set_compare(Axis, CMP1Pos, CMP1Dir,
     CMP2Pos, CMP2Dir)
```

check_compare

```
Dim RetCode As Integer
Dim Axis As Integer, status As Integer
Axis = 1
RetCode  = check_compare (Axis, status)
```

1.11 Interlock Function

@ Name

`set_interlock(CardID, Flag, Axis_X, Axis_Y, X1, X2, Y1, Y2,Time)` – setup interlock function

`get_interlock(CardID, *Flag, *Axis_X, *Axis_Y, *X1, *X2, *Y1, *Y2, *Time)` – get interlock parameters from board.

@ Description

`set_interlock`:

This function is used to set up interlock facility. Every PCI-83XX Series has a interlock protect facility.

`get_interlock`:

This function is used to get interlock parameters on SSCNET Board.

@ Syntax

C/C++ (DOS, Windows)

```
I16 set_interlock(I16 Card, I16 Flag, I16 Axis_X,
     I16 Axis_Y,F64 X1, F64 X2, F64 Y1, F64 Y2)
I16   get_interlock(I16 CardID, I16 *Flag, I16
     *Axis_X,I16 *Axis_Y, F32 *X1, F32 *X2, F32
     *Y1, F32 *Y2, F32 *Time);
```

Visual Basic (Windows)

```
set_interlock(ByVal Card As Integer, ByVal Flag
     As Integer, ByVal Axis_X As Integer, ByVal
     Axis_Y As Integer, ByVal X1 As Double, ByVal
     X2 As Double, ByVal Y1 As Double, ByVal Y2
     As Double) As Integer
get_interlock(ByVal Card As Integer, Flag As
     Integer, Axis_X As Integer,  Axis_Y As
     Integer,  X1 As Double, X2 As Double,  Y1 As
     Double,  Y2 As Double) As Integer@ Arguments
```

`CardID`: specified card index

Flag: Enable/Disable interlock facility

- Flag = 0, Disable
- Flag = 1, Enable

Axis_X, **Axis_Y**: specified axis number

X1, **X2**, **Y1**, **Y2**: specified interlock region

@ Return Code

```
ERROR_NoError
ERROR_Dll_Opened_By_Other_Process
ERROR_Wrong_Axis_Number
ERROR_Card_Not_Ready
ERROR_Card_Not_Initial
ERROR_DSP_Not_Ready
ERROR_Axis_Not_In_Control
```

@ Example

<C/C++ >

set_interlock

```
I16 RetCode;
I16 Card = 0;
I16 Flag = 1; //Enable interlock
I16 Axis_X = 1;// Axis 1
I16 Axis_Y = 6;//Axis 6
F64 X1 = -10.0;
F64 X2 = 15.0;
F64 Y1 = -15.0;
F64 Y2 = 10.0;
F64 Time = 0.01 ;
RetCode  = set_interlock(CardID, Flag, Axis_X,
     Axis_Y, X1, X2, Y1, Y2, Time);
```

<Visual Basic>

set_interlock

```
Dim RetCode As Integer
Dim Card As Integer, Flag As Integer
Dim Axis_X As Integer, Axis_Y As Integer
Dim X1 As Double, X2 As Double, Y1 As Double, Y2
     As Double
Card = 0
```

```
Flag = 1 //Enable interlock
Axis_X = 1// Axis 1
Axis_Y = 6//Axis 6
X1 = -10.0
X2 = 15.0
Y1 = -15.0
Y2 = 10.0
FTime = 0.01
RetCode  = set_interlock(CardID, Flag, Axis_X,
     Axis_Y, X1, X2, Y1, Y2, FTime)
```

1.12 Absolute Position System

@ Name

`get_abs_position(Axis , *ABS_Pos)` – get ABS position value from driver

`save_abs_position(Axis)` – save current position as absolute zero to ROM

`clear_abs_data_on_flash(CardID)` – clear ABS data in ROM

@ Description

`get_abs_position`:

This function will get the ABS data from servo driver and calculate ABS position. It changes the monitor channels to 0x0b, 0x0c, 0x1c and 0x1a. Then it obtains RevOffset, CurData, OrgData and AbsR. It also obtains the servo parameter07 about the rotation direction setting and the PPR of the axis. Finally, it will calculate the ABS_Pos according to servo parameter07. You can see that the procedures are more than the get_position() function and it takes a longer time. Normally, this function is not needed at run time; use get_position() instead.

If servo parameter07 is set to '1', this function must be used to obtain ABS_Pos after MDSP_initial() and the set_position() function must be used by this ABS_Pos. If this is not performed, the initial position of get_position() function will feedback an incorrect value. If servo parameter07 is set to '0', nothing need to be done after MDSP_initial(). The get_position() value will be the correct ABS_Pos.

`save_abs_position`:

This function is used to save current zero point into the onboard flash. Next time the servo drivers are connected, the ABS position will be calculated according to this zero point. It changes the monitor channels to 0x0b, 0x0c, 0x0d, 0x1a. Then it obtains CurData and RevData, which are then saved to the flash.

If the ABS origin data does not exist on the FLASH, the current position will be zero at beginning. Notice that after this function is

called, the monitor channel will be changed. To be sure that the position is in zero location, this function will return an error when command counter is not zero.

`clear_abs_data_on_flash`:

This function will clear all axes' ABS position data on the onboard flash. It will not load the ABS position the next time the servo drivers are connected.

@ Syntax

C/C++ (DOS, Windows)

```
I16 clear_abs _data_on_flash(I16 CardID);
I16 save_abs_position(I16 Axis);
I16 get_abs_position(I16 Axis, F64 *ABS_Pos);
```

Visual Basic (Windows)

```
clear_abs_data_on_flash(ByVal CardID As Integer)
     As Integer
save_abs_position(ByVal Axis As Integer) As
     Interger
get_abs_position(ByVal Axis As Integer, ABS_Pos
     As Double) As Integer
```

@ Arguments

`CardID`: specified card index

`Axis`: specified axis index

`ABS_Pos`: Current position relative to absolute origin of driver

@ Return Code

```
ERROR_NoError
ERROR_Dll_Opened_By_Other_Process
ERROR_Wrong_Axis_Number
ERROR_Card_Not_Ready
ERROR_Card_Not_Initial
ERROR_DSP_Not_Ready
ERROR_Axis_Not_In_Control
ERROR_Unknown_Card_Type
ERROR_Command_Counter_Not_Zero
ERROR_Clear_ABS_Data_Fail
```

@ Example

<C/C++ >

```
// After home_move(), the command position
     position should be zero.
for(i=0;i<TotalAxes;i++)
    save_abs_position(i);
.
.
.
// When program ends, call this functions will
     save all axes' ABS value
MDSP_close(CARD0);
.
.
.
// Next time when program starts, call this
     function will load all axes'
// ABS value
MDSP_initial(0);
.
.
.
// If users want to erase ABS data on FLASH
clear_abs_data_on_flash(CARD0);
// If users wants to reload ABS data, use the
     following two functions
F64 ABS_Value;
get_abs_data(AxisNo, &ABS_Value);
set_position(AxisNo, ABS_Value);
```

<Visual Basic>

```
' After home_move(), the command position
     position should be zero.
For i=0 to TotoalAxes-1
    save_abs_position i
Next i
.
.
.
' When program ends, call this functions will
     save all axes' ABS value
MDSP_close CARD0
```

```
.
.
.
‘ Next time when program starts, call this
     function will load all axes’
‘ ABS value
MDSP_initial 0
.
.
.
‘ If users want to erase ABS data on FLASH
clear_abs_data_on_flash CARD0
‘ If users wants to reload ABS data, use the
     following two functions
get_abs_data AxisNo, ABS_Value
set_position AxisNo, ABS_Value
```

1.13 Pulse Output Control

@ Name

`set_pulse_output_control(Axis , Enable, PulseCH, Mode)` – Map pulse output control to one axis

@ Description

`set_pulse_output_control`:

This function is used to enable/disable the pulse output control of the axis. Users can map one of the 12 axes to one pulse output channel and set the pulse output mode from 3 types of pulse format. After issuing this function, the axis is regarded as a stepper control axis. Users can use any motion function for this axis. This feature is not available in PCI version but in cPCI version.

@ Syntax

C/C++ (DOS, Windows)

```
I16 set_pulse_output_control(I16 Axis,U16 Enable,
      U16 PulseCH, U16 Mode);
```

Visual Basic (Windows)

```
set_pulse_output_control(ByVal Axis As Integer,
      ByVal Enable As Integer, ByVal PulseCH As
      Integer, ByVal Mode As Integer) As Integer
```

@ Arguments

`Axis`: specified axis index

`Enable`: Enable(1)/disable(0) the pulse output of the axis.

`PulseCH`: Assign pulse output channel (0 or 1)

`Mode`: Assign pulse output mode

- Mode=0 OUT/DIR
- Mode=1 CW/CCW
- Mode=2 EA/EB
- Others=Reserved

@ Return Code

```
ERROR_NoError
ERROR_Dll_Opened_By_Other_Process
ERROR_Wrong_Axis_Number
ERROR_Card_Not_Ready
ERROR_Card_Not_Initial
ERROR_DSP_Not_Ready
ERROR_Axis_Not_In_Control
```

1.14 Sequence Motion Control

@ Name

add_frame_ta_move – add motion frames into on board frame buffer

add_frame_dwell(Axis, StartFrameNo, DTime) – add dwell frame into on board frame buffer

set_pattern – set the pattern by frame range in frame buffer

get_pattern – get the frame range information of the pattern

insert_pattern_to_seq_buffer – insert the pattern number in sequene command buffer

insert_pattern_to_seq_buffer – insert the pattern number in sequene command buffer

check_seq_buffer(CardID, SeqNo) – check sequence command buffer index

reset_seq_buffer(CardID, SeqNo) – reset sequence command buffer index

start_seq_move – start sequence motion

pause_seq_move – pause sequence motion

resume_seq_move – resume sequence motion

end_seq_move – quit sequence motion

set_seq_sync_pause – set synchronous axes stop in sequence motion

check_seq_buffer_index – read back sequence motion command buffer index

check_seq_buffer_empty_count(CardID, SeqNo) – read back sequence motion command buffer empty quantities

@ Description

add_frame_ta_move:

This function is for users to create motion frames in the onboard frame buffer. It will create 1~3 frames depend on the velocity and

position parameter. The return code of it is the start frame number for next motion command. Users can create up to 1200 frames in the frame buffer.

`add_frame_dwell`:

This function is for users to create a dwell frame in the onboard frame buffer. The parameter of time is in unit of second. Notice that in SSCNET system, the time base is 0.888ms. The return code of it is the start frame number for next motion command.

`set_pattern`:

This function is for users to set a pattern for sequence command buffer. The pattern is composed by several frames. It contents starting frame number, total frame number and synchronized axes.

`get_pattern`:

This function is for users to get a pattern information including starting frame number, total frame number and synchronized axes.

`insert_pattern_to_seq_buffer`:

This function is for users to insert a pattern into sequence command buffer. Each sequence has 4 command buffers to store pattern number. The command buffer is for continuous pattern motion of the sequence especially when the pattern quantities of the sequence are greater than 4. Users also can set a starting condition for an individual pattern. Notice that the pattern inserting time is about 2ms.

`check_seq_buffer`:

This function is for users to check the sequence command buffer. It will return 1 when the sequence command buffer is ready for next command inserted. If the return value is 0, It means the buffer is full.

`reset_seq_buffer`:

This function is for users to reset the sequence command buffer index. It will reset “begin” and “end” index to zero.

`start_seq_move`:

This function is for users to start one or more than one sequence motion. Once it is started, the axes in the sequences will start to

move or waiting the start conditions. Users can issue this command before pattern is inserted.

pause_seq_move:

This function is for users to pause one or more than one sequence motion at the same time. All the axes in the sequences will slow down then stop at the same time.

resume_seq_move:

This function is for users to reseume one or more than one sequence motion. All the axes in the sequences will speed up at the same time.

end_seq_move:

This function is used to end sequence motion condition. Once start_seq_motion() is issued, all the sequences will be in a state of waiting patterns for running and the motion status will be in busy state. This condition will not be cancelled until the end_seq_motion() is issued. Notice that this function is only applicable when the sequene buffer is empty or it will return an error code. The motion busy status will be clear after the function is issued. If users want to execute non-sequence motion command, they must launch this function first.

set_seq_sync_pause:

This function is used to enable synchronous motion check between axes in sequence motion control. If the axes has common working area or the axes have specific coordinated relationship, users can use this function to prevent machine to crash. For example, if axis0 and axis1 are set to synchronous pause, once axis0 is stopped by any reason, axis1 will stop at the same time. Once it is paused by any emergency reason, Please use motion stop function to stop sequence motion or resume sequence function to continue sequence motion.

check_seq_buffer_index:

This function is used to read back sequence motion command buffer index for double checking. It can read back next read index, next write index and running index.

check_seq_buffer_empty_count:

This function is used to read back sequence motion command buffer empty quantities. It is for double checking. The return code of this function is the count.

@ Syntax

C/C++ (DOS, Windows)

```
I16 add_frame_ta_move(I16 Axis, I16 StartFrameNo,
      F64 StartPos, F64 EndPos, F64 StartVel, F64
      MaxVel, F64 FinVel, F64 Tacc, F64 Tdec);
I16 add_frame_dwell(I16 Axis, I16 StartFrameNo,
      F64 StayPos, F64 DTime);
I16 get_pattern(I16 CardID, I16 PatternNo, I16
      *FirstFrame, I16 *TotalFrame, U16
      *SyncAxes);
I16 set_pattern(I16 CardID, I16 PatternNo, I16
      FirstFrame, I16 TotalFrame, U16 SyncAxes);
I16 insert_pattern_to_seq_buffer(I16 CardID, I16
      SeqNo, I16 PatternNo, I16 SyncAxes, I16
      WaitAxis, I16 StartCondition, I16
      StartPatternNo, F32 StartData,I16
      EndCondition);
I16 check_seq_buffer(I16 CardID, I16 SeqNo);
I16 start_seq_move(I16 CardID, I16 SeqNoBit);
I16 reset_seq_buffer(I16 CardID, I16 SeqNo);
I16 pause_seq_move(I16 CardID, I16 SeqNoBit, F64
      Dec_Time);
I16 resume_seq_move(I16 CardID, I16 SeqNoBit, F64
      Acc_Time);
I16 end_seq_move(I16 CardID, I16 SeqNoBit);
I16 set_seq_sync_pause(I16 CardID, I16 Flag, I16
      GroupNo, I16 AxisNoBit, F64 DecTime)
I16 check_seq_buffer_index(I16 CardID, I16 SeqNo,
      I16 *NextRead, I16 *NextWrite, I16
      *RunningIndex);
I16 check_seq_buffer_empty_count(I16 CardID, I16
      SeqNo)
```

Visual Basic (Windows)

```
end_seq_move (ByVal CardID As Integer, ByVal
      SeqNoBit As Integer) As Integer
set_seq_sync_pause(ByVal CardID As Integer, ByVal
      Flag As Integer, ByVal GroupNo As Integer,
```

```
        ByVal AxisNoBit As Integer, ByVal DecTime As
        Double) As Integer
    check_seq_buffer_index(ByVal CardID As Integer,
        ByVal SeqNo As Integer,  NextRead As
        Integer, NextWrite As Integer, RunningIndex
        As Integer) As Integer
    check_seq_buffer_empty_count (ByVal CardID As
        Integer, ByVal SeqNo As Integer) As Integer
```

@ Arguments

Axis: specified axis index

CardID: specified card ID

StartFrameNo: Start FrameNo in frame buffer (max=1200)

StartPos: Start position of the frame motion in mm

EndPos: End position of the frame motion in mm

StartVel: Start velocity of the frame motion in mm/s

MaxVel: Maximun velocity of the frame motion in mm/s

FinVel: Final velocity of the frame motion in mm/s

Tacc: Acceleration time of the frame motion in mm/s2

Tdec: Deceleration time of the frame motion in mm/s2

DTime: Dwell time in sec

FrameNo: Frame number in onboard frame buffer

***FTime**: Frame duration time in on board frame buffer

PatternNo: Pattern number

FirstFrame: The starting frame number in the pattern

TotalFrame: The total frame number in the pattern

SyncAxes: Assigned synchronized motion axes in the sequence

***FirstFrame**: The starting frame number readback in the pattern

***TotalFrame**: The total frame number readback in the pattern

***SyncAxes**: Synchronized motion axes readback in the sequence

SeqNo: Sequence number

SeqNoBit: Sequence number in each bit

WaitFlag:

- 0 = ignore start condition and start immediately
- 1 = wait for axis condition
- 2 = ignore start condition and start immediately. It will generate INP interrupt if enabled when this pattern ends.
- 3 = wait for axis condition. It will generate INP interrupt if enabled when this pattern ends.

WaitAxis: The axis number associated with start condition when WaitFlag=1

StartCondition:

- Frame start condition:
  - 0~98= the pattern number of wait axis (please refer to StartData for additional setting)
- Compare start condition:
  - 100~199= feedback position >= StartData (when the pattern is equal to 0~99)
  - 200~299= feedback position <= StartData (when the pattern is equal to 0~99)

StartData:

- For Frame start condition:
  - It means frame order number in a pattern, start from 0. The sequence will start at the beginning of the frame order number in a pattern.
- For Compare start condition:
  - It means the compare value in unit of mm

Flag: Sequence motion flag (0: Disable, 1: Enable)

GroupNo: range = 0~5

AxisNoBit: axis number in each bit

Dec_Time: Deceleration time of pause command

Acc_Time: Acceleration time of resume command

***NextRead**: Next read index

***NextWrite**: Next write index

***RunningIndex**: Current Running Index

@ Return Code

```
ERROR_NoError
ERROR_Dll_Opened_By_Other_Process
ERROR_Wrong_Axis_Number
ERROR_Card_Not_Ready
ERROR_Card_Not_Initial
ERROR_DSP_Not_Ready
ERROR_Axis_Not_In_Control
```

@ Example

<C/C++ >

end_seq_move –

```
intert_pattern_to_seq_buffer(CARD0, SEQ0, P00,
     SYNC0, WAIT_OFF, StartCondition,
     StartData);
intert_pattern_to_seq_buffer(CARD0, SEQ1, P20,
     SYNC1, WAIT_ON, StartCondition, StartData);
start_seq_move(CARD0, SYNC0 | SYNC1 );
…
// After All sequene motion is finished.
end_seq_move(CARD0,SYNC0 | SYNC1);
```

set_seq_sync_pause –

```
set_seq_sync_pause(CARD0,ENABLE,PAIR0,0x03,
     0.1);
tv_stop(AXIS0,0.1);
resume_seq_move
// When tv_stop() is launched, the corresponding
     axis1 will be stopped too
```

1.15 Auto Stop Protection Function

@ Name

```
set_pos_diff_stop(CardID, Enable,
     PosDiffCheckSetNo_0_To_5,  Axis_X, Axis_Y,
     UpperLimit, LowerLimit)
```

@ Description

`set_pos_diff_stop`:

This function is used for checking two axes' position difference. If the position difference of two axes over the preset range, they will be stopped automatically. Users can set 6 pairs of axes for this checking. Each pair contents two axes and they can't be overlapped in axis number. Once the motion is stopped by this function, please disable it by the same function and move axes to a safe position then re-enable it.

@ Syntax

C/C++ (Windows)

```
I16 set_pos_diff_stop(I16 CardID, I16 Enable, I16
     PosDiffCheckSetNo_0_To_5, I16 Axis_X,I16
     Axis_Y, F32 UpperLimit, F32 LowerLimit)
```

Visual Basic (Windows)

```
set_pos_diff_stop(ByVal CardID As Integer, ByVal
     Enable As Integer, ByVal
     PosDiffCheckSetNo_0_To_5 As Integer, ByVal
     Axis_X As Integer,ByVal Axis_Y As Integer,
     ByVal UpperLimit As Single, ByVal LowerLimit
     As Single) As Integer
```

@ Argument

`CardID`: The PCI-83XX Series card index number.

`Enable`: 0=diable, 1=enable

`PosDiffCheckSetNo_0_To_5`: range = 0~5

`Axis_X`: The first axis in one pair

`Axis_Y`: The second axis in one pair

UpperLimit: The position difference upper limit value (UpperLimit > LowerLimit)

LowerLimit: The position difference lower limit value (LowerLimit < UpperLimit)

@ Return Code

```
ERROR_NoError
ERROR_Dll_Opened_By_Other_Process
ERROR_Card_Not_Exist
ERROR_PCIBios_Not_Exist
ERROR_Card_Reinitialized
ERROR_Card_Not_Exist
ERROR_Card_Not_Accessible
ERROR_Card_Not_Ready
ERROR_DSP_Not_Ready
ERROR_DSP_Initial_Time_Out
ERROR_Unknow_Card_Type
```

@ Example

<C/C++ >

set_pos_diff_stop –

```
F32 UpperLimit, LowerLimit
I16 RetCode;
set_pos_diff_stop(CARD0, ENABLE, PAIR0, Axis0,
     Axis2, -10.123, 18.435 );
```

If the axes stops by this function, please disable it by setting flag=0. Others parameter will be ignored.

```
set_pos_diff_stop(CARD0, DISABLE, PAIR0, Axis0,
     Axis2, -10.123, 18.435 );
```

After stop condition recovered, please re-enable it again.

2 Appendix

2.1 MR-J2S-B Alarm List

When any alarm has occurred, eliminate its cause, ensure safety, then deactivate the alarm, and restart operation. Not doing so can cause injury.

AL.10	Undervoltage	Power supply voltage dropped. MR-J2S-B: 160V or lessMR-J2S-oB1: 83V or less
AL.12	Memory alarm 1	RAM memory fault
AL.13	Clock alarm	Printed board fault
AL.15	Memory alarm 2	EEPROM fault
AL.16	Encoder alarm 1	Communication error occurred between encoder and servo amplifier.
AL.17	Board alarm	CPU/parts fault
AL.19	Memory alarm 3	ROM memory fault
AL.1A	Motor combination alarm	Combination of servo amplifier and servo motor is wrong.
AL.20	Encoder alarm 2	Communication error occurred between encoder and servo amplifier.
AL.24	Main circuit error	Ground fault occurred at the servo motor outputs (U, V, W phases) of the servo amplifier.
AL.25	Absolute position erase	Absolute position data in errorPower was switched on for the first time in the absolute position detection system.
AL.30	Regenerative alarm	The permissible regenerative power of the built-in regenera-tive brake resistor or regenerative brake option is exceeded.Regenerative transistor fault
AL.31	Overspeed	Speed has exceeded the instantaneous permissible speed.
AL.32	Overcurrent	Current that flew is higher than the permissible current of the servo amplifier.
AL.33	Overvoltage	Converter bus voltage input value-exceeded 400V.
AL.34	CRC error	Bus cable is faulty.
AL.35	Command pulse frequency alarm	The pulse frequency of the input command pulses is too high.
AL.36	Transfer error	Bus cable or printed board is faulty.
AL.37	Parameter alarm	Parameter setting is wrong.
AL.45	Main circuit device overheat	Main circuit overheated abnormally.
AL.46	Motor overheat	Servo motor temperature rise actuated the thermal protector.
AL.50	Overload 1	Load exceeded overload protection characteristic of servo amplifier.Load ratio 300%: 2.5s or moreLoad ratio 200%: 100s or more

Table 2-1: MR-J2S-B Alarm List

AL.51	Overload 2	Machine collision etc. caused max. output current to flow successively for several seconds.Servo motor locked: 1s or more
AL.52	Error excessive	Droop pulse value of the deviation counter exceeded the parameter No.31 setting value .
AL.8E	Serial communication alarm	Serial communication fault occurred between servo amplifier and communication device (e.g. personal computer).
88	Watchdog	CPU, parts faulty

Table 2-1: MR-J2S-B Alarm List

2.2 MR-J2S-B Warning List

If E6, E7, E9 or EE occurs, the servo off status is established. If any other warning occurs, operation can be continued but an alarm may take place or proper operation may not be performed. Eliminate the cause of the warning according to this section. Use the optional servo configuration software to refer to the cause or warning.

AL.92	Open battery cable warning	Absolute position detection system battery voltage is low.
AL.96	Home position setting warning	Home position return could not be made in the precise position.
AL.9F	Battery warning	Voltage of battery for absolute position detection system reduced.
AL.E0	Excessive regenerative load warning	There is a possibility that regenerative power may exceed permissible regenerative power of built-in regenerative brake resistor or regenerative brake option.
AL.E1	Overload warning	There is a possibility that overload alarm 1 or 2 may occur.
AL.E3	Absolute position counter warning	Absolute position encoder pulses faulty.
AL.E4	Parameter warning	Parameter outside setting rang.
AL.E6	Servo emergency stop	EM1-SG are open.
AL.E7	Controller emergency stop warning.	
AL.E9	Main circuit off warning	Servo was switched on with main circuit power off.
AL.EE	SCCNET error warning	The servo system controller connected is not SSCNET-compatible..

Table 2-2: MR-J2S-B Warning List

2.3 Driver parameter List

Symbol	Name	MR-J2SB Instruction Manual Parameter	Unit	Setting range
*AMS	Amp setting	Pr.01		0000H~0001H
*REG	Regenerative resistor	Pr.02		0000H~0011H
*MTY	For manufacturer's settings	Pr.03		0080H
*MCA	For manufacturer's settings	Pr.04		0000H
*MTR	For manufacturer's settings	Pr.05		1
*FBP	Feedback pulse number	Pr.06		0,1,6,7,225
*POL	Direction of motor rotation	Pr.07		0,1
ATU	Auto-tuning	Pr.08		0000H~0004H
RSP	Servo response setting	Pr.09		0001H~000FH
TLP	Forward rotation torque limits	Pr.10	%	0~Maximum torque
TLN	Reverse rotation torque limits	Pr.11	%	0~Maximum torque
DG2	Moment of inertia ratio of load	Pr.12	0.1	0~3000
PG1	Position control gain 1	Pr.13	rad/sec	4~2000
VG1	Speed control gain 1	Pr.14	rad/sec	20~8000
PG2	Position control gain 2	Pr.15	rad/sec	1~1000
VG2	Speed control gain 2	Pr.16	rad/sec	20~20000
VIC	Speed integration compensation	Pr.17	msec	1~1000
NCH	Mechanical resonance control filter	Pr.18		0~031FH
FFC	Feed forward gain	Pr.19	%	0~100
INP	In position range	Pr.20	pulse	0~50000
MBR	Electromagnetic brake sequence output	Pr.21	msec	0~1000
MOD	Monitor output mode	Pr.22		0000H~0B0BH
OP1	Optional function 1	Pr.23		0000H~0001H
OP2	Optional function 2	Pr.24		0000H~0110H
LPF	Low pass filter	Pr.25		0000H~1210H
OP4	For manufacturer's settings	Pr.26		0000H
MO1	Monitor output 1 offset	Pr.27	mv	-999~999
MO2	Monitor output 2 offset	Pr.28	Mv	-999~999

Table 2-3: Driver parameter List

Symbol	Name	MR-J2SB Instruction Manual Parameter	Unit	Setting range
MOA	For manufacturer's settings	Pr.29		0001H
ZSP	Zero speed	Pr.30	rpm	0~10000
ERZ	Error excess alarm level	Pr.31	kpulse	1~1000
OP5	Option function 5	Pr.32		0000H~0002H
OP6	For manufacturer's settings	Pr.33		0000H~0113H
VPI	PI-PID change position droop	Pr.34		0~50000
TTT	For manufacturer's settings	Pr.35		0000H
VDC	Speed integration compensation	Pr.36		0~1000
OP7	For manufacturer's settings	Pr.37		0010H
ENR	Encoder output pulse	Pr.38		0~32768
	For manufacturer's settings	Pr.39		0000H
*BLK	Parameter block	Pr.40		0000H~000EH

Table 2-3: Driver parameter List

2.4 Function Response Time

Category	Function name	Process time (ms)
System Function		
System Function	error_msg()	0.0133
	MDSP_initial()	3227.5232
	MDSP_close()	2494.1883
	MDSP_reset()	2716.2470
	get_base_addr()	0.0206
	get_irq_number()	0.0205
	understand_mdsp()	0.0261
	get_version_info()	0.0197
Motion Function		
Single Motion	tv_move()	3.0834
	sv_move()	3.0942
	start_tr_move()	3.0813
	start_sr_move()	3.1140
	start_ta_move()	3.0865
	start_sa_move()	3.1093
	tv_move_all()	2.2246
	sv_move_all()	2.2830
	start_tr_move_all()	2.2505
	start_sr_move_all()	2.4633
	start_ta_move_all()	2.2489
	start_sa_move_all()	2.3428
	tv_change()	10.0539
	sv_change()	10.0534
	tv_stop()	9.0982
	sv_stop()	9.0830
	emg_stop()	4.0551
	start_line_tr_move()	2.2537
	start_line_sr_move()	2.5953
	start_line_ta_move()	2.2496
	start_line_sa_move()	2.7381
	start_arc_tr_move()	2.1525
	start_arc_sr_move()	2.1949
	start_arc_ta_move()	2.1473
	start_arc_sa_move()	2.1926

Table 2-4: Function Response Time

Category	Function name	Process time (ms)
Home move	set_home_mode()	0.0368
	home_move()	3.1159
Continuous motion	start_motion_list()	0.0631
	end_motion_list()	0.0046
	repeat_last_move()	10.0228
	add_line_tr_move()	0.0100
	add_line_sr_move()	0.0142
	add_line_ta_move()	0.0068
	add_line_sa_move()	0.0167
	add_arc_tr_move()	0.0131
	add_arc_sr_move()	0.0156
	add_arc_ta_move()	0.0137
	add_arc_sa_move()	0.0162
	add_arc2_sa_move()	0.0140
	add_arc2_sr_move()	0.0162
	add_arc2_ta_move()	0.0148
	add_arc2_tr_move()	0.0167
	add_dwell()	0.0029
	smooth_enable()	0.0022
	start_cont_move()	2.2824
Motion Related I/O Function		
Position control and feedback	get_position()	0.0316
	set_position()	10.0323
	get_target_pos()	0.0200
	get_move_ratio()	0.0142
	set_move_ratio()	10.0231
	get_fb_position()	0.0263
	shift_position()	10.0318
Velocity feedback	get_velocity()	5.0321
	get_cnt_speed()	0.0214
Motion DIO status	set_PEL_config()	10.0315
	set_MEL_config()	10.0314
	set_ORG_config()	10.0310
	set_EMG_config()	5.0240

Table 2-4: Function Response Time

Category	Function name	Process time (ms)
Software status	get_PEL_status()	5.0255
	get_MEL_status()	5.0252
	get_ORG_status()	5.0255
	get_EMG_status()	5.0236
Software limit	set_soft_limit()	30.0463
	get_soft_limit()	0.0275
Motion status	motion_status()	5.0158
	axis_status()	5.0171
	motion_done()	5.0157
Frame Management	frame_to_execute()	0.0130
General Purpose IO		
Encoder	set_cnt_iptmode()	10.0287
	set_cnt_to_axis()	15.0337
	set_cnt_value()	100.7444
	get_cnt_value()	0.0227
	set_ring_counter()	40.0771
	get_ring_counter()	40.0827
DIO	get_di_status()	1.0248
	set_do_value()	0.0207
	set_di_mode()	6.0275
	set_mio_mode()	20.0441
DA	set_da_config()	0.0199
	set_da_value()	10.0279
AD	set_ad_function()	40.0854
	get_ad_value()	0.0066
Analog auto calibration	tune_ref_5v()	20.0378
	save_auto_k_value()	70.1220
	get_auto_k_value()	40.0613
	tune_ad_offset_gain()	40.0807
	tune_da_offset()	50.0967
	reload_auto_k_setting()	543.5953
Driver Management Function		

Table 2-4: Function Response Time

Category	Function name	Process time (ms)
Driver parameters	get_servo_para()	0.0204
	set_servo_para()	7023.8662
	get_servo_para_all()	0.0939
	set_servo_para_all()	730.5785
	save_servo_para()	849.8831
	set_servo_para_default()	763.2503
Data monitoring	set_monitor_channel();	10.0253
	set_monitor_config ()	0.0220
	start_monitor()	7.0285
	check_monitor_ready()	0.0230
	get_monitor_data()	42.8274
	get_instant_monitor_data()	6.1094
Servo infomation	get_servo_info()	0.0240
Servo on	set_servo_on()	196.3837
Driver information	understand_driver()	0.0313
	understand_motor()	0.0328
Alam	get_alarm_no()	1.0249
	alarm_reset()	0.0202
Control Gain Tuning		
Control Gain Tuning	set_auto_tune()	739.9458
	get_auto_tune()	0.0946
	set_control_gain()	734.3896
	get_control_gain()	0.0893
	set_notch_filter()	0.0934
	get_notch_filter()	0.0944
	set_LP_filter()	707.6136
	get_LP_filter()	0.0938
Interrupt Control Function		
Interrupt Control Function	int_control()	0.0213
	set_int_factor()	0.0204
	get_int_status()	0.0209
	set_int_event()	0.0685
	link_interrupt()	0.0112
	set_int_event2()	4.8593
	clear_tlc_int()	10.0313
	set_timer_int_interval()	0.0269
Position Compare Function		

Table 2-4: Function Response Time

Category	Function name	Process time (ms)
Position Compare Function	set_compare()	40.0577
	check_compare()	0.0215
	set_compare_source()	0.0054
	set_compare_channel()	2.0289
	set_single_compare()	2.0190
	map_dout_and_comparator()	11.0255
	set_compare_table_dir()	11.0236
	link_dout_and_compare_table()	232.9729
	set_ext_encoder_compare_method()	10.0314
	set_ext_encoder_compare_value()	10.0316
Interlock Function		
Interlock Function	set_interlock()	870.6801
	get_interlock()	0.0618
Absolute Position System		
Absolute Position System	get_abs_position()	21.1741
	save_abs_position()	66.2161
	clear_abs_data_on_flash()	782.0468
Pulse Output Control		
Pulse Output Control	set_pulse_output_control()	10.0323
Sequence Motion Control		
Sequence Motion Control	add_frame_ta_move()	10.0687
	add_frame_dwell()	10.0629
	set_pattern	10.0604
	get_pattern	10.0652
	insert_pattern_to_seq_buffer()	2.0334
	check_seq_buffer()	0.0031
	reset_seq_buffer()	10.0219
	start_seq_move()	9.3435
	pause_seq_move()	6.0639
	resume_seq_move()	6.3479
	end_seq_move()	6.0338
	set_seq_sync_pause()	20.0417
	check_seq_buffer_index()	0.0034
	check_seq_buffer_empty_count()	0.0033
Auto Stop Function		
Auto Stop Function	set_pos_diff_stop()	40.0592

Table 2-4: Function Response Time

Warranty Policy

Thank you for choosing ADLINK. To understand your rights and enjoy all the after-sales services we offer, please read the following carefully.

1. Before using ADLINK's products please read the user manual and follow the instructions exactly. When sending in damaged products for repair, please attach an RMA application form which can be downloaded from: http://rma.adlinktech.com/policy/.
2. All ADLINK products come with a two-year guarantee:
   - The warranty period starts from the product's shipment date from ADLINK's factory.
   - Peripherals and third-party products not manufactured by ADLINK will be covered by the original manufacturers' warranty.
   - For products containing storage devices (hard drives, flash cards, etc.), please back up your data before sending them for repair. ADLINK is not responsible for loss of data.
   - Please ensure the use of properly licensed software with our systems. ADLINK does not condone the use of pirated software and will not service systems using such software. ADLINK will not be held legally responsible for products shipped with unlicensed software installed by the user.
   - For general repairs, please do not include peripheral accessories. If peripherals need to be included, be certain to specify which items you sent on the RMA Request & Confirmation Form. ADLINK is not responsible for items not listed on the RMA Request & Confirmation Form.

3. Our repair service is not covered by ADLINK's two-year guarantee in the following situations:
   - Damage caused by not following instructions in the user's manual.
   - Damage caused by carelessness on the user's part during product transportation.
   - Damage caused by fire, earthquakes, floods, lightening, pollution, other acts of God, and/or incorrect usage of voltage transformers.
   - Damage caused by unsuitable storage environments (i.e. high temperatures, high humidity, or volatile chemicals).
   - Damage caused by leakage of battery fluid during or after change of batteries by customer/user.
   - Damage from improper repair by unauthorized technicians.
   - Products with altered and/or damaged serial numbers are not entitled to our service.
   - Other categories not protected under our warranty.
4. Customers are responsible for shipping costs to transport damaged products to our company or sales office.
5. To ensure the speed and quality of product repair, please download an RMA application form from our company website: http://rma.adlinktech.com/policy. Damaged products with attached RMA forms receive priority.

If you have any further questions, please email our FAE staff: service@adlinktech.com.