SETUP Automação
www.setup.com.br
+55 19 2517.8900

http://bit.ly/ST-V300

IO Control Card

Programming Manuals

Revision History

Revision	Release Date:	Description of Change(s)
V1.0	2018-01-26	

Table of Contents

1. Use of IO control card function library

If users use our IO card for product development, they need to use the dynamic link library corresponding to the IO card. This IO card provides the dynamic link library under Windows XP/Win7. Users only need to call the instructions in the function library to realize various functions of IO card.

1.1 Use of dynamic link libraries under Windows

For the IO development under Windows XP/Win7 system, a dynamic link library based on VC/MFC is provided. In the SDK folder of the installation package directory, the dynamic link library file of IO is named lhio.dll.
You can use any development tool that supports dynamic link libraries to develop applications. Below Visual C++ is an example of how to use the IO card's dynamic link library in these development tools.

1.1.1 Use in Visual C++

Start Visual Studio 2008, create a new Visual C++ project, and complete the New Project Wizard. Add the motion link library of the motion controller to the project by following the steps below.

1. Copy the dynamic link library lhio.dll, the header files lhio.h and lhio.lib files in the SDK\VC folder of the installation package to the project folder.
2. Select the "Properties" menu item under the "Project" menu.
3. Expand the "Common Properties" - "Linker" item, select the "Input" tab, and enter the lib filename in the "Additional Dependencies" field (for example lhio.lib).
4. Add a declaration of the function library header file to the application file, for example # include “ lhio.h”
5. At this point, the user can call any function in the function library in Visual C++ to start writing the application.

2. Command return value and meaning

2.1 Command return value

The IO card works according to the instructions sent by the host. The IO card instructions are encapsulated in the dynamic link library lhio.dll. The user manipulates the IO card by calling the IO card command while writing the application.

Definition of command return value shown as in table 2-1:

Table 2-1 IO card command return value definition

Return value	Meaning	Handling
-7	Instruction execution failed	Check instruction usage.
-5	Device not open	Check if the motion controller is turned on.
-3	Device operation failed	Check if the IO card is working properly.
-1	Incoming error parameters when calling interface	Check the incoming parameter type, if the range is correct
0	Instruction execution successfully	Go on.

	It is recommended to detect the return value of each instruction in the user program to determine the execution status of the instruction and establish the necessary error handling mechanism to ensure the program runs safely and reliably.

2.2 Instruction error handling

Routine 2-1 using command return value for error handling

```
void error(short rtn)
{
    switch(rtn)
    {
    case   -7:
        printf("\a\nResponse Error !");
        break;
    case   -5:
        printf("\a\ nDevice not open !") ;
        break;
    case   -3:
        printf("\a\nFailed operator!");
        break;
    case   -1:
        printf("\a\nParameter error !");
        break;
    case   0:
        break;
```

```
    }
}
void main()
{
    int    rtn;
    rtn = LH_OpenDevice (0);     error(rtn);    //open the first IO card
     // Start other operations
     //……
     //End other operations
    rtn = LH_CloseDevice(0);     error(rtn);    //Close the first IO card
}
```

3. Open and close function

3.1 Highlight Instructions

This section describes the basic operations of the IO card device.
The IO card supports multiple cards at the same time. By calling LH_DeviceCount, you can get how many IO cards are inserted into the PC. Each card has a corresponding number, starting from 0. Entering the card number in the other function interface will realize the function operation of the corresponding card.
Before calling the function of the IO card, you should first open the device with LH_OpenDevice, and call LH_CloseDevice to close the corresponding device after use or before the process exits. To check if the IO card is opened with LH_IsDeviceOpenned.

3.2 List of Command

Table 3-1 IO card initialization instruction list

Command	Instruction
LH_DeviceCount	To gain number of IO card
LH_OpenDevice	To open corresponding IO card
LH_CloseDevice	To close corresponding IO card
LH_IsDeviceOpenned	To determine if the corresponding IO card is turned on

Table 3-2 IO card initialization instruction parameter description

LH_DeviceCount()	
Return value	Number of IO card
LH_OpenDevice(unsigned int deviceId)	
deviceId	Corresponding IO card, start from 0.
LH_CloseDevice(unsigned int deviceId)	
deviceId	Corresponding IO card, start from 0.
LH_IsDeviceOpenned(unsigned int deviceId)	
deviceId	Corresponding IO card, start from 0.

3.3 Routine

Routine as following:

```
int index = 0; //device ID
int devCount = LH_DeviceCount();
if (devCount == 0)
{
    MessageBox(L"No device found!");
    return;
```

```
}
 int rtn = LH_OpenDevice((unsigned int)index);
  // Start other operations
  //……
  //End other operations
int rtn = LH_CloseDevice((unsigned int)index);
```

4. DO Function

4.1 Highlight Instructions

This section describes the digital output of the IO card device - DO function.

IO card supports 8 channel digital output, numbered from 1 to 8, and each channel can be used as both a level output and a pulse output.

In the level output mode, LH_SetDo or LH_SetDoBit is called to change the state of the output level, and the status of the output level of the port is obtained by LH_GetDo or LH_GetDoBit. LH_SetDoInv is used to invert the output level. By default, the output is not inverted.

In pulse output mode, you can call function interface to directly output pulses, or you can trigger the output through the DI port, or you can configure the output to be compared to the position (refer to Chapter 8). LH_PulseOutput is used to directly output the pulse. The IO card outputs the pulse immediately according to the specified delay time and pulse width. LH_DiTrigOutput is used to set the input trigger output function, that is, the input port detects a specified edge signal and outputs a pulse.

An output channel can only be a function at a time. When calling different function interfaces, it will switch from the current mode to the specified mode. By calling LH_GetDoMode, the current mode of the DO channel can be obtained. In general, you can only switch to another mode after the current mode stops working, so as to avoid causing logic errors. For example, if the DO channel is used for position comparison output, you should stop the position comparison first, and then call the function you need to use.

4.2 List of command

Table 4-1 Relevant list of DO command

Command	Instruction
LH_SetDo	Set the level status of the specified I/O output of the IO card
LH_SetDoBit	Set output level of the specified port of the specified IO card
LH_GetDo	Read the level status of the digital I/O output of the specified IO card
LH_GetDoBit	Reads the level status of the digital I/O output of the specified port of the specified IO card
LH_SetDoInv	Set the level of the specified IO card digital I/O output to be inverted
LH_GetDoInv	Set the level of the digital I/O output of the specified port of the specified IO card to be inverted
LH_GetDoMode	Get the output mode of the specified port of the specified IO card
LH_PulseOutput	Set the specified port of the specified IO card to output pulse signal
LH_DiTrigOutput	Set DI trigger DO output pulse signal

Table 4-2 IO card initialization instruction parameter description

LH_SetDo (unsigned int deviceId, unsigned short val)	
deviceId	Corresponding IO card, start from 0.
val	Output status, each of which represents an output port with a value of 0-0xff. The corresponding bit is 1 for high level and 0 for low level
LH_SetDoBit(unsigned int deviceId, unsigned int port, unsigned short val)	
deviceId	Corresponding IO card, start from 0.
port	Port number, range 1-8
val	Output status, value 0-1, 1 is output high level, 0 is output low level
LH_GetDo(unsigned int deviceId, unsigned short &val)	
deviceId	Corresponding IO card, start from 0.
val	Output status, each bit represents the state of an output port . The corresponding bit is 1 for high level and 0 for low level
LH_GetDoBit(unsigned int deviceId, unsigned int port, unsigned short &val)	
deviceId	Corresponding IO card, start from 0.
port	Port number, range 1-8
val	Output status, value 0 or 1
LH_SetDoInv(unsigned int deviceId, unsigned int port, int inverse)	
deviceId	Corresponding IO card, start from 0.
port	Port number, range 1-8
inverse	0 - normal; 1 - inverse
LH_GetDoInv(unsigned int deviceId, unsigned int port, int &inverse)	
deviceId	Corresponding IO card, start from 0.
port	Port number, range 1-8
inverse	0 - normal; 1 - inverse
LH_GetDoMode(unsigned int deviceId, unsigned int port, int &mode)	
deviceId	Corresponding IO card, start from 0.
port	Port number, range 1-8

mode	Current output mode: DO_MODE_LEVEL indicates output level DO_MODE_PULSE_DIRECT indicates software trigger pulse output DO_MODE_PULSE_DI indicates DI trigger pulse output DO_MODE_PULSE_COMPARE_LINEAR indicates equal-pitch comparison trigger pulse output DO_MODE_PULSE_COMPARE_DATA indicates the discrete data comparison trigger pulse output
LH_PulseOutput(unsigned int deviceId, unsigned int port, unsigned int delay, unsigned int pulseWidth)	
deviceId	Corresponding IO card, start from 0.
port	Port number, range 1-8
delay	Output delay, Unit (us)
pulseWidth	Pulse width, Unit (us)
LH_DiTrigOutput(unsigned int deviceId, int outPort, int delay, int pulseWidth, int trigPort, int trigEdge)	
deviceId	Corresponding IO card, start from 0.
outPort	Port number, range 1-8
delay	Output delay, Unit (us)
pulseWidth	Pulse width, Unit (us)
trigPort	Trigger input port, 1—8
trigEdge	Trigger edge, 0—Rising edge ; 1—Falling edge

4.3 Routine

```
//LH_SetDoBit routine as following:
    int rtn = 0;
    int deviceIndex = 0;
    int doIndex = 1;
    rtn = LH_SetDoBit(deviceIndex, doIndex, 0);
    rtn = LH_SetDoBit(deviceIndex, doIndex, 1);
```

```
//LH_SetDoInv routine as following:
    int inverse = 1;
    rtn = LH_SetDoInv(deviceIndex, doIndex,inverse);
    switch (rtn)
    {
    case FUNC_RETURN_ERROR_PARAMETER:
        MessageBox(L"Return parameter error!");
        break;
    case FUNC_RETURN_FILE_OPERATE_FAIL:
        MessageBox(L" File operation failed!");
        break;
    case FUNC_RETURN_DEVICE_NOT_OPEN:
        MessageBox(L"Device not opened!");
        break;
    case FUNC_RETURN_FAIL:
        MessageBox(L"Command execution
        failure!");
        break;
    default:
        break;
    }
//LH_PulseOutput routine as following:
    unsigned int delay = 0, width = 1000;
    rtn = LH_PulseOutput(deviceIndex, doIndex,delay,width);
//LH_DiTrigOutput routine as following:
    int diChn = 1;
    int diEdge = 0;
    rtn = LH_DiTrigOutput(deviceIndex, doIndex,delay,width,diChn,diEdge);
    switch (rtn)
    {
    case FUNC_RETURN_ERROR_PARAMETER:
        MessageBox(L"Return parameter error! ");
        break;
    case FUNC_RETURN_FILE_OPERATE_FAIL:
        MessageBox(L" File operation failed!");
        break;
    case FUNC_RETURN_DEVICE_NOT_OPEN:
        MessageBox(L"Device not opened!");
        break;
    case FUNC_RETURN_FAIL:
        MessageBox(L"Command execution
        failure!");
        break;
    default: break;
    }
```

5. DI Function

5.1 Highlight instructions

This section describes the digital output of the IO card device - DI function.
The IO card supports 8 channel digital inputs, numbered from 1 to 8, each channel can be used as both a level input and an input signal edge count. In order to reduce the interference of the input signal noise, the input signal can be filtered by calling LH_SetDiFilter.
LH_SetDiMode is used to switch the mode of the input channel. When sampling as a level, call LH_GetDi or LH_GetDiBit to get the level status of the current port, and use LH_SetDiInv to invert the input status. When used as an edge count, call LH_GetDiCount to get the count value, and call LH_ResetDiCount to clear the count value.

5.2 List of Command

Table 5-1 Position comparison guide list

Command	Instruction
LH_GetDi	Get the digital input status of the specified IO card
LH_GetDiBit	Get the corresponding port digital input status of the specified IO card
LH_SetDiMode	Set the mode type of the DI of the corresponding port of the specified IO card
LH_GetDiMode	Get the mode type of the DI of the corresponding port of the specified IO card
LH_SetDiInv	Set the corresponding port digital input of the specified IP card to be reversed
LH_GetDiInv	Get the digital inversion input of the corresponding port of the specified IO card
LH_SetDiFilter	Set the filter time of the corresponding port of the specified IO card
LH_GetDiFilter	Get the filter time of the corresponding port of the specified IO card
LH_GetDiCount	Get the number of edges of the corresponding port of the specified IO card
LH_ResetDiCount	Reset the number of edges of the corresponding port of the specified IO card

Table 5-2 DI function command parameter description

LH_GetDi(unsigned int deviceId, unsigned short &val)	
deviceId	Corresponding IO card, start from 0.
val	Input status, each bit represents an input port, the value is 0-0xff, right Should be 1 for input high, 0 for low
LH_GetDiBit(unsigned int deviceId, unsigned int port, unsigned short &val)	
deviceId	Corresponding IO card, start from 0.

port	Port number, range 1-8
Val	Status value, 1 means input high level, 0 means low level

LH_SetDiMode(unsigned int deviceId, unsigned int port, int mode)	
deviceId	Corresponding IO card, start from 0.
port	Port number, range 1-8
mode	Input sampling mode: DI_MODE_LEVEL indicates level input, DI_MODE_PULSE_RISING indicates rising edge counting DI_MODE_PULSE_FALLING indicates falling edge counting
LH_GetDiMode(unsigned int deviceId, unsigned int port, int &mode)	
deviceId	Corresponding IO card, start from 0.
port	Port number, range 1-8
mode	Same as LH_SetDiMode。
LH_SetDiInv(unsigned int deviceId, unsigned int port, int inverse)	
deviceId	Corresponding IO card, start from 0.
port	Port number, range 1-8
inverse	0-normal; 1-inverse
LH_GetDiInv(unsigned int deviceId, unsigned int port, int &inverse)	
deviceId	Corresponding IO card, start from 0.
port	Port number, range 1-8
inverse	0-normal; 1-inverse
LH_SetDiFilter(unsigned int deviceId, unsigned int port, unsigned int filter)	
deviceId	Corresponding IO card, start from 0.
port	Port number, range 1-8
filter	Filter time, unit (us)
LH_GetDiFilter(unsigned int deviceId, unsigned int port, unsigned int &filter)	
deviceId	Corresponding IO card, start from 0.
port	Port number, range 1-8
filter	Filter time, unit (us)

LH_GetDiCount(unsigned int deviceId, unsigned int port, int &count)	
deviceId	Corresponding IO card, start from 0.
port	Port number, range 1-8
count	Edge count value
LH_ResetDiCount(unsigned int deviceId, unsigned int port)	
deviceId	Corresponding IO card, start from 0.
port	Port number, range 1-8

5.3 Routine

```
//Routine as following:
    unsigned short val = 0x0;
    int rtn = 0;
    int deviceIndex = 0;
    unsigned int filter = 10;
    short channel = 1;
    // Set the input filter time
    rtn = LH_SetDiFilter(deviceIndex,channel,filter);
    // Set the input inverse
    int mode = 1;
    rtn = LH_SetDiInv(deviceIndex, channel,mode);
    //Read the input status
    rtn = LH_GetDi(deviceIndex, val);
    for (int i = 0; i < 8; ++i)
    {
        if (val & (1 << i))
        {
            // Corresponding port is high level
        }
        else
        {
            // Corresponding port is low level
        }
    }
    mode = DI_MODE_PULSE_RISING;
    int iCnt = 0;
```

```
//Set mode
rtn = LH_SetDiMode(deviceIndex,channel,mode);
// Reset counter
rtn = LH_ResetDiCount(deviceIndex, channel);
//Read count number
rtn = LH_GetDiCount(deviceIndex, channel, iCnt);
switch (rtn)
{
case FUNC_RETURN_ERROR_PARAMETER:
    MessageBox(L"Return parameter error!");
    break;
case FUNC_RETURN_FILE_OPERATE_FAIL:
    MessageBox(L" File operation failed! ");
    break;
case FUNC_RETURN_DEVICE_NOT_OPEN:
    MessageBox(L" Device not opened!");
    break;
case FUNC_RETURN_FAIL:
    MessageBox(L" Command execution
    failure! ");
    break;
default:
    break;
}
```

6. PWM Function

6.1 Highlight Instructions

This section describes the PWM functions of the IO card device.
The IO card supports four PWM output functions, numbered from 1 to 4, the output frequency range is 1 Hz to 500 kHz, and the duty ratio is 0% - 100%. Trigger PWM output is divided into soft trigger and hard trigger mode. In soft trigger mode, after LH_SetPwmParam is called to set PWM frequency and duty ratio, LH_PwmOn can output PWM waveform. In hard trigger mode, the above two functions are also called, but waveform will not be output immediately but need to wait for the special port of the hardware to trigger, please check the IO card instruction manual for details.

6.2 List of Command

Table 6-1 PWM command instruction

Command	Instruction
LH_SetPwmHWTrigger	Set trigger mode
LH_GetPwmHWTrigger	Get trigger mode
LH_SetPwmParam	Set output frequency, duty ratio and other parameters
LH_GetPwmParam	Get output frequency, duty ratio and other parameters
LH_PwmOn	Enable PMW function switch of corresponding channel
LH_PwmOff	Disable PMW function switch of corresponding channel

Table 6-2 PWM command instructions description

LH_SetPwmHWTrigger(unsigned int deviceId, unsigned short channel, unsigned short trigger)	
deviceId	Corresponding IO card, start from 0.
channel	Channel number, 1-4
trigger	Trigger mode: 0-soft trigger, 1-hard trigger
LH_GetPwmHWTrigger(unsigned int deviceId, unsigned short channel, unsigned short &trigger)	
deviceId	Corresponding IO card, start from 0.
channel	Channel number, 1-4
trigger	Trigger mode: 0-soft trigger, 1-hard trigger
LH_SetPwmParam(unsigned int deviceId, unsigned short channel, unsigned int frequency, unsigned int duty)	
deviceId	Corresponding IO card, start from 0.
channel	Channel number, 1-4
frequency	Output frequency Hz, range(1—500000)
duty	Duty ratio (0~100)
LH_GetPwmParam(unsigned int deviceId, unsigned short channel, unsigned int &frequency, unsigned int &duty)	
deviceId	Corresponding IO card, start from 0.
channel	Channel number, 1-4
frequency	Output frequency Hz
duty	Duty ratio (0~100)
LH_PwmOn(unsigned int deviceId, unsigned short channel)	

deviceId	Corresponding IO card, start from 0.
channel	Channel number, 1-4
LH_PwmOff(unsigned int deviceId, unsigned short channel)	
deviceId	Corresponding IO card, start from 0.
channel	Channel number, 1-4

6.3 Routine

```
//Routine as following:
    int rtn=0;
   // Channel number, 1-4
   unsigned short channel=1 ;
    //Device ID
    unsigned int deviceIndex=0;
   // Trigger mode: 0-soft trigger, 1-hard trigger
    unsigned short trigger=0
    //Frequency Hz
    unsigned int frequency=100;
    //Duty ratio
    unsigned int duty=50;

   //Enable PMW function switch
     rtn = LH_PwmOn(deviceIndex, channel);
   //Set trigger mode
    rtn = LH_SetPwmHWTrigger(deviceIndex, channel,trigger);
    //Get trigger mode
     rtn = LH_GetPwmHWTrigger(deviceIndex, channel, trigger);
     //Set output frequency, duty ratio and other parameters
     rtn = LH_SetPwmParam(deviceIndex, channel, frequency,duty);
     //Get output frequency, duty ratio and other parameters
    rtn = LH_GetPwmParam(deviceIndex, 4, period, duty);
     //Disable PMW function switch of corresponding channel
     rtn = LH_PwmOff(deviceIndex, channel);
    if (rtn == FUNC_RETURN_SUCCEED)
    {
        return;
```

```
}
switch (rtn)
{
case FUNC_RETURN_ERROR_PARAMETER:
    MessageBox(L"Return parameter error!");
    break;
case FUNC_RETURN_FILE_OPERATE_FAIL:
    MessageBox(L" File operation failed!");
    break;
case FUNC_RETURN_DEVICE_NOT_OPEN:
    MessageBox(L" Device not opened!");
    break;
case FUNC_RETURN_FAIL:
    MessageBox(L" Command execution
    failure!");
    break;
default:
    break;
}
```

7. Encoder function

7.1 Highlight Instructions

This section describes the counting function of the IO card device Encoder.
The IO card supports one AB phase quadrature encoder input signal. By default, the A phase is incremented by the previous count value. If LH_SetEncDirection is called to invert the count direction, the A phase pre-count value is decremented. LH_SetEncEnable is called to enable before using the encoder function. The position comparison and position latching in subsequent chapters are also the same. Read encoder count value with the LH_EncCount function and reset the encoder count value with LH_EncReset.

7.2 List of command

Table 7-1 Encoder command list

Command	Instruction
LH_SetEncEnable	Enable corresponding encoder of specified IO card
LH_GetEncEnable	Get if corresponding encoder of specified IO card is enabled
LH_EncReset	Reset encoder count value
LH_EncCount	Get encoder count value
LH_SetEncDirection	Set encoder count direction
LH_GetEncDirection	Get encoder count direction

Table 7-1 Encoder command parameter instruction

LH_SetEncEnable(unsigned int deviceId, unsigned short enable, short channel = 0)	
deviceId	Corresponding IO card, start from 0.
enable	Enable, 0-invalid, 1-valid
channel	Channel number, can only be equal to 0
LH_GetEncEnable(unsigned int deviceId, unsigned short &enable, short channel = 0)	
deviceId	Corresponding IO card, start from 0.
enable	Enable, 0-invalid, 1-valid
channel	Channel number, can only be equal to 0

LH_EncReset(unsigned int deviceId, short channel = 0)	
deviceId	Corresponding IO card, start from 0.
channel	Channel number, can only be equal to 0
LH_EncCount(unsigned int deviceId, int &count, short channel = 0)	
deviceId	Corresponding IO card, start from 0.
count	Encoder count value
channel	Channel number, can only be equal to 0
LH_SetEncDirection(unsigned int deviceId, unsigned short direction, short channel = 0)	
deviceId	Corresponding IO card, start from 0.
direction	Count direction, 0-A ahead is increase counting, otherwise 1-A decrease counting
channel	Channel number, can only be equal to 0
LH_GetEncDirection(unsigned int deviceId, unsigned short &direction, short channel = 0)	
deviceId	Corresponding IO card, start from 0.
direction	Count direction, 0-A ahead is increase counting, otherwise 1-A decrease counting
channel	Channel number, can only be equal to 0

7.3 Routine

```
//LH_SetEncEnable  routine as following:
int rtn=0;
//Device ID
unsigned int deviceId=0;
//Enable 0-invalid, 1-valid
unsigned short enable=1;
```

```
//Channel number, start from 0
short channel =0
//Number of count
int count=0;
//Count direction, 0-A ahead is increase counting, otherwise 1-A decrease
counting
unsigned short direction=0;
// Specify the corresponding channel of the IO card to enable
rtn =LH_SetEncEnable(deviceId, enable, channel);
// Obtain whether the corresponding channel of the specified IO card is
enabled
rtn =LH_GetEncEnable(deviceId, enable, channel)
// Reset encoder count value
rtn =LH_EncReset(deviceId, channel);
// Read encoder count value
rtn =LH_EncCount(deviceId, count, channel)
// Set encoder count direction
rtn =LH_SetEncDirection(deviceId, direction, channel);
// Get encoder count direction
rtn =LH_GetEncDirection(deviceId, &direction, channel );
if (rtn == FUNC_RETURN_SUCCEED)
    {
        return;
    }
    switch (rtn)
    {
    case FUNC_RETURN_ERROR_PARAMETER:
        MessageBox(L"Return parameter error!");
        break;
    case FUNC_RETURN_FILE_OPERATE_FAIL:
        MessageBox(L" File operation failed!");
        break;
    case FUNC_RETURN_DEVICE_NOT_OPEN:
        MessageBox(L" Device not opened!");
        break;
    case FUNC_RETURN_FAIL:
        MessageBox(L" Command execution
        failure! ");
        break;
    default:
        break;
    }
```

8. Position comparison function

8.1 Highlight instructions

This section describes the IO card device position comparison function.

The position information input by the encoder can be used to compare the output. There are two ways to compare data: spacing mode(LH_CompareLinear command) and array mode (LH_CompareData command). If the comparison position is a large number of consecutive equally spaced outputs, you can use the spacing mode to enter the position comparison data. In this case, you only need to enter the starting comparison position, the separation distance, and the number of repeated comparisons. If the comparison position is a non-equidistant position value, you can use the array method to input, with the current position as the starting point, enter the distance array of the comparison position relative to the starting position. In this case, ensure that the data in the position array is a monotonically increasing positive distance value or a monotonically decreasing negative distance value. When the encoder position value is equal to the set comparison value, a pulse of the specified pulse width is output from the set DO port. The DO channel of the pulse output can be set at the same time, and each channel outputs the same pulse.

During the position comparison process, the LH_CompareStatus query position can be called to compare whether the output has been completed and the number of times the comparison is triggered. Call LH_CompareStop to cancel the current comparison function

Note that the encoder is enabled by calling LH_SetEncEnable before using the position comparison function.

8.2 List of Command

Table 8-1 Position comparison function instruction list

Command	Instruction
LH_CompareData	Set position comparison parameters and start comparison function
LH_CompareLinear	Set equidistant position comparison parameters and start comparison function
LH_CompareStatus	Check on position comparison status
LH_CompareStop	Cancel position comparison output

LH_CompareData(unsigned int deviceId, long *pCompareBuf, short count, unsigned long pulseWidth, short DoChnMask)	
deviceId	Corresponding IO card, start from 0.
pCompareBuf	Compared position value
count	Count of position
pulseWidth	Pulse width, unit (us)
DoChnMask	Output channel mask, the lowest bit indicates DO1

LH_CompareLinear(unsigned int deviceId, long startPos, long repeatTimes, long interval, unsigned long pulseWidth, short DoChnMask)	
deviceId	Corresponding IO card, start from 0.
startPos	Start position of comparison
repeatTimes	Repeat times
interval	Position spacing
pulseWidth	Pulse width, unit (us)
DoChnMask	Output channel mask, the lowest bit indicates DO1
LH_CompareStatus(unsigned int deviceId, long *pStatus, long *pCount)	
deviceId	Corresponding IO card, start from 0.
pStatus	1- comparing completed, 0-comparing
pCount	Count of comparison trigger
LH_CompareStop(unsigned int deviceId)	
deviceId	Corresponding IO card, start from 0.

8.3 Routine

```
//Routine as following:
    int rtn=0;
   //Channel ID, 1-4
   unsigned short channel=1 ;
    //Device ID
    unsigned int deviceIndex=0;
   //Trigger mode, 0-soft trigger, 1-hard trigger
    unsigned short trigger=0
    //Frequency Hz
    unsigned int frequency=100;
    // Duty Ratio
```

```
unsigned int duty=50;
unsigned long pulseWidth=10;
//8-channel channel selection, for example, select channel 1 and 2, 00000011 is 3. 8 通道 通道选择 例 如 选 择 第 1 和 2 通道 00000011 为 3
short chnMask= 1;
//Incoming numbers
long a[5]={1,2,3,4,5};
//Count of numbers
short count =5;
//Position of starting comparison
long startPos=1;
//Repeat times
long   repeatTimes=2;
// Position spacing
Long interval=10;
// Comparison status
  Long status=0;
//Count of comparison trigger
Long pCount=0
// Set position comparison parameters and start
rtn = LH_CompareData(deviceIndex, a ,count, plsWidth,chnMask);
// Set equal spacing position comparison output
rtn  =  LH_CompareLinear(deviceIndex,  startPos,  repeatTimes,interval,pulseWidth, chnMask);
//Check on position comparison status
rtn = LH_CompareStatus(deviceIndex,& status,& pCount);
// Cancel position comparison output
rtn =LH_CompareStop(deviceIndex);
if (rtn == FUNC_RETURN_SUCCEED)
{
    SetTimer(1,200,NULL);
    return;
}
switch (rtn)
{
case FUNC_RETURN_ERROR_PARAMETER:
    MessageBox(L"Return parameter error!");
    break;
case FUNC_RETURN_FILE_OPERATE_FAIL:
    MessageBox(L" File operation failed!");
    break;
case FUNC_RETURN_DEVICE_NOT_OPEN:
    MessageBox(L" Device not opened!");
    break;
```

```
case FUNC_RETURN_FAIL:
        MessageBox(L" Command execution
        failure! ");
        break;
default:
      break;
}
```

9. Position latch function

9.1 Highlight Instructions

This section describes the IO card encoder position latching function.
Position latching means that when the DI port detects a rising or falling edge, it immediately latches the encoder's count value, which is generally used in situations where accurate positioning is required. Call LH_EncCapture to set the input signal port and capture the signal edge and start capturing. Call the LH_GetCaptureStatus instruction to check the capture status. If a response signal is captured, the latched position value can be obtained.
Call LH_CaptureStop to stop capturing. Note that the encoder is enabled by calling LH_SetEncEnable before using the position comparison function.

9.2 List of Command

Table 9-1 Position latch function list

Command	Instruction
LH_EncCapture	Set the signal capture edge and start
LH_CaptureStatus	Read the capture state and position of the
LH_CaptureStop	Stop capturing

Table 9-1 Position lock function instruction parameter description

LH_EncCapture(unsigned int deviceId, short captEdge, short DiChannel = 1)	
deviceId	Corresponding IO card, start from 0.
captEdge	Capture edge, 0-rising edge, 1-falling edge
DiChannel	Trigger source channel index (1-8)
LH_CaptureStatus(unsigned int deviceId, long *pStatus, long *pCount, short DiChannel = 1)	
deviceId	Corresponding IO card, start from 0.
pStatus	0-capturing, 1-capture completed
pCount	Latched count value
DiChannel	Trigger source channel index (1-8)

LH_CaptureStop(unsigned int deviceId, short DiChannel = 1)	
deviceId	Corresponding IO card, start from 0.
DiChannel	Trigger source channel index（ 1-8）

9.3 Routine

```
//Routine as following:
    int rtn=0;
    //Device ID
    unsigned int deviceId=0;
    //Capture edge, 0-rising edge, 1-falling edge
     short captEdge = 0;
   // Trigger source
     channel index ( 1-8 )
     short DiChannel=1;
     //Capture status, 0-capturing, 1- capture completed
     long pStatus=0;
     // Latched count value
     long pCount=0;
     // Set the signal capture edge to start capturing
    rtn = LH_EncCapture(deviceId, captEdge, DiChannel);
   // Read the capture state and position of the specified axis
   rtn =LH_CaptureStatus(deviceId, &pStatus,&pCount, DiChannel)
   //stop capture
   rtn =LH_CaptureStop(deviceId, DiChannel)
    if (rtn == FUNC_RETURN_SUCCEED)
    {
        return;
    }
    switch (rtn)
    {
    case FUNC_RETURN_ERROR_PARAMETER:
        MessageBox(L"Return parameter error!");
        break;
    case FUNC_RETURN_FILE_OPERATE_FAIL:
        MessageBox(L" File operation failed! ");
        break;
    case FUNC_RETURN_DEVICE_NOT_OPEN:
        MessageBox(L"Device not opened!");
        break;
```

```
case FUNC_RETURN_FAIL:
    MessageBox(L" Command execution
    failure! ");
    break;
default:
    break;
}
```