PCI Express Bus NCB

HPCIe-NCB674N(1)

User's Manual

<Software>

NC Board

Multifunctional High-Speed Circular/Linear Interpolation and Positioning

Hivertec,inc.

http://www.hivertec.co.jp/

This manual is for the **NCB Series** boards listed below.

HPCIe-NCB674N	4-axis PCI Express Bus
HPCIe-NCB674N(1)	4-axis PCI Express Bus + 16IN/16OUT

Hivertec Inc.
Mitsuiseimei Shin-Ohashi Bldg.
1-8-11 Shin-Ohashi, Koto-Ku, Tokyo 135-0007, Japan
TEL +81-3-3846-3801
FAX +81-3-3846-3773
sales@hivertec.co.jp

Version 2.20 September 27 2016

Please be sure to read the "Please read first" section carefully before using the product.

Table of Contents

Table of Figures

■ Please read first

■ Extent of Warranty

1. The product warranty is valid for a period of three years from the date of purchase. If a defect is acknowledged by Hivertec within the period of warranty, Hivertec will repair or replace the product upon return of the product to Hivertec.

2. Hivertec is not responsible beyond the purchase price of the product for any damages or loss of profit, direct, indirect, or secondary, caused by application, delivery, or failure of a Hivertec product either within or outside of the period of warranty.

■ Limitations to Liability

1. Hivertec is not responsible for any damages resulting from product installation, connections, settings, or operation that do not follow the contents of this manual.

2. This product uses semiconductor devices manufactured for general electronics equipment, such as machine tools, instrumentation, measurement hardware, FA devices, OA devices, and communications equipment. They are not designed, conceived, approved for, or warranted for application in devices for which faulty operation or failure will have a direct effect on human life or result in personal injury or damage to property, such as medical equipment, traffic equipment, burning appliances, or safety devices.
   The safety, quality, and performance of the products are not guaranteed explicitly or implicitly beyond those given in this manual or related catalogs.

3. Hivertec is not responsible for any damages resulting from modifications or repairs made to the product without the approval of Hivertec either within or outside of the period of warranty.

4. The contents of this manual do not guarantee or grant rights to patents, copyright, trademark rights, or any other rights to the intellectual property of Hivertec or any third party.
   Hivertec is not responsible for any problems that may occur concerning the rights to intellectual property of third parties resulting from the application of information provided in this manual.

■ Important Safety Instructions

Thank you for choosing this product. This manual contains information that is important for the safe and reliable operation of this product. Read this section and understand the information contained before attempting to use the product.
Furthermore, save this manual and store it in an easily accessible location, so that it can be referenced when necessary.

Safety Precautions

Always read this manual and any attached documents completely before attempting to use this product. Be sure that you understand the information provided and are using the product correctly. Do not use the product before having a complete understanding of the product, product safety information, and precautions.
In this manual, safety precautions are classified as either Warnings or Cautions.

⚠ **Warning**	Indicates a potentially hazardous situation which, if not avoided, could result in death or serious injury.

⚠ **Caution**	Indicates a potentially hazardous situation which, if not avoided, may result in minor or moderate injury, or property damage.

1.1.1 Authorized For

Caution

This product and this manual are designed for those with the following knowledge.

- A basic knowledge for installing and wiring expansion boards.
- A basic knowledge of electronic control devices, personal computers, and Windows operating system.
- A basic knowledge of Microsoft Visual Studio or Embarcadero *Delphi for Windows development environment*.

1.1.2 "Getting started" and "Sample Programs"

Warning

The "Getting started" bundled with this product is for confirming whether the product is properly set and connected or whether the operating environmental conditions are properly set, and for understanding the functions and operations of the boards.
Furthermore, the "sample programs" bundled with this product are for the purpose of helping you understand the control procedures of this product as well as how to create a control program.
Do not use "Getting started" or the "sample programs" for ordinary operations because, due to its purpose above, they do not include safety measures unique to each machine or system.

When connecting motors and equipment or devices and operating, make sure to set parameters considering the features of the connected equipment or devices.
Especially during trial operation, make sure to start with sufficiently safe parameter values and change gradually to the required value.

When using "Getting started" or the "sample programs" to operate equipment or devices, always check the function and safety of the equipment or devices, and start at low speed after making sure that settings match the mechanical system. Operating with settings that do not match the mechanical system may result in unexpected operation leading to serious injury and/or death.

■ Program Adjustment Made by Users

Warning

Always debug the program thoroughly before using this product to drive devices.
Any error in the program may result in unexpected operation leading to death and/or serious injury.

Manual Configuration

This board comes with the manuals shown below.

(1) CPD Board Series User's Manual <Introduction>
(2) CPD Board Series User's Manual <Operation>
(3) HPCIe-NCB674N(1) User's Manual <Hardware>
(4) HPCIe-NCB674N(1) User's Manual <Software> • • • This manual

The contents of each manual are listed below.

CPD Series User's Manual <Introduction>
－For All Developers
- CPD Series Overview
- Installation (except for NCB)
- Trial Operation (except for NCB)
- Glossary

CPD Series User's Manual <Operation>
－Primarily for Software Developers
- Basic Operation
- Special Operations
- PCL Reference

HPCIe-NCB674N(1)
User's Manual <Hardware>
－Primarily for Wiring Personnel
- Product Specification, Options at Purchase
- Block Diagram
- Connection Configuration
- Board Settings
- External Connections
- Accessories (Relay Connector Board, Connection Cable, etc.)
- Examples of Connection with Servo Amplifiers of Various Companies

HPCIe-NCB674N(1)
User's Manual <Software>
－Primarily for Software Developers
- Software Overview
- Installation of Accompanying Software
- Trial Operation and "Getting started"
- NCB-specific Functions
- Library Functions
- Driver Functions
- Sample Program
- Information on Ports

■ Terms and Names appearing in this Manual

1. The axes are named **X, Y, Z, U, V, W, A**, and **B**.
2. As for input/output description in this manual of various kinds, about which axis is clearly specified.
   Example : **XCWP** (CW : pulse output+ for X-axis)
3. Abbreviations appearing in this Manual here after

ELS	End Limit Sensor (**XELS** : End Limit Sensor of X-axis)
DLS	Deceleration Sensor (**YDLS** : Deceleration Sensor of Y-axis)
OLS	Origin Sensor (**ZOLS** : Origin Sensor of Z-axis)
CMP	Comparator coincident output (**CMPX** : Comparator output of X-axis)
INPOS	In-position
SVALM	Servo Alarm (**XSVALM** : Servo Alarm of X-axis)
SVCTRCL	Servo Error Counter Clear (**YSVCTRCL** : Servo Error Counter Clear for Y-axis)
EXTPOW	External Power Supply
EXTGND	External Ground
EMG	Master stop request (eventually abbreviated for full stop of each 4-axis, that is from X to U and from V to B) (This is not the “Emergency Stop” as a hardware device.)
DSW	Dip Switch
SYNCA	Syncronized Connection for Interboard master slave connection for CW pulse
SYNCB	Syncronized Connection for Interboard master slave connection for CCW pulse
AP	encoder phase-A input+(Positive) (**XAP** : encoder phase-A input+ for X-axis)
AN	encoder phase-A input-(Negative) (**XAN** : encoder phase-A input- for X-axis)
TTL	Transistor Transistor Logic

4. As for Status
   Please refer to the "User’s Manual <Operation>" and "<Software>" such as describe below.
   Example : ERST, MSTS, RSTS

1. Introduction

This manual describes the software of HPCIe-NCB674N(1) and HPCIe-NCB674N, NCB series motion control boards compliant with PCI Express 1.0a (x1).

HPCIe-NCB674N(1) consists of a 4-axis CPD series board, a generic I/O, and a CPU while HPCIe-NCB674N consists of a 4-axis CPD series board and a CPU; therefore, refer to "CPD Board Series User's Manual <Operation>" for their basic operation procedure.
As for information regarding the hardware of these products, refer to "HPCIe-NCB674N(1) User's Manual <Hardware>".

In this manual, HPCIe-NCB674N(1) and HPCIe-NCB674N are referred to as NCB674N.
Furthermore, control LSIs such as the PCL6045 are generically referred to as PCL.

1.1 Accompanying Software

1.1.1 OSs supported by the accompanying software

OSs supported by the accompanying software are Windows 10, 8.1, 7 and XP. For other OSs, please contact Hivertec Sales Dept. or your sales representative.

1.1.2 Accompanying software details

This product comes with the following software (Windows version as standard) to help you understand this product software.

Device driver	Software required to use this product with Windows. It is available in 64 bits and 32 bits. (Other than XP is 64 bit)
Library functions (Level 1)	A collection of functions necessary for performing basic operations. They are provided in source code form and explained in "CPD Board Series User's Manual <Software>". They are available in Microsoft Visual C++, Visual Basic, Visual C#, and Delphi.
NCB series library functions	These library functions are for supporting MDA, CDA, and CMP operations, which are functions specific to NCB. They are provided in source code form and are available in Microsoft Visual C# only.
Sample programs	A collection of samples on how to use library functions. They are included in each of the project files in Microsoft Visual C++, Visual Basic, Visual C#, and Delphi.
"Getting started"	Allows for minimum operation just by connecting the board to a PC. It can also be used to check connections.

Table 1.1-1 Accompanying Windows software

1.1.3 Software configuration

(1) Device drivers

■ Beware that the Windows device driver to use varies depending on your OS as indicated below.

◇Windows 10 (64bit) Notation: Win10(x64) Use: hecp670b.sys
◇ Windows 8 (64bit) Notation: Win8(x64) Use: hecp670b.sys
◇ Windows 7 (64bit) Notation: Win7(x64) Use: hecp670b.sys
◇Windows 10 (32bit) Notation: Win10(x86) Use: hecp670a.sys
◇ Windows 8 (32bit) Notation: Win8(x86) Use: hecp670a.sys
◇ Windows 7 (32bit) Notation: Win7(x86) Use: hecp670a.sys
◇ Windows XP Notation: WinXP Use: hecp670a.sys

(2) Device driver functions

Functions included in the device driver I/F library are called "driver functions".

■ Windows driver functions hecp670.dll

(3) Library functions level 1

These are "driver-function based library functions for processing specific functions" in application programs, and are provided as source programs. The content of these library functions can be changed freely. These are called "library functions" in contrast to driver functions.

■ Windows library functions (Compliant with CPD Series)

◇ cp670l1a.c (cp670l1a.h) • • • For Microsoft Visual C++ (2008 or later)
◇ cp670l1a.vb • • • For Microsoft Visual Basic (2008 or higher)
◇ cp670l1a.cs • • • For Microsoft Visual C# (2008 or higher)

■ Windows library functions (NCB series)

◇ NcbLib1a.cs • • • For Microsoft Visual C# (2008 or higher)

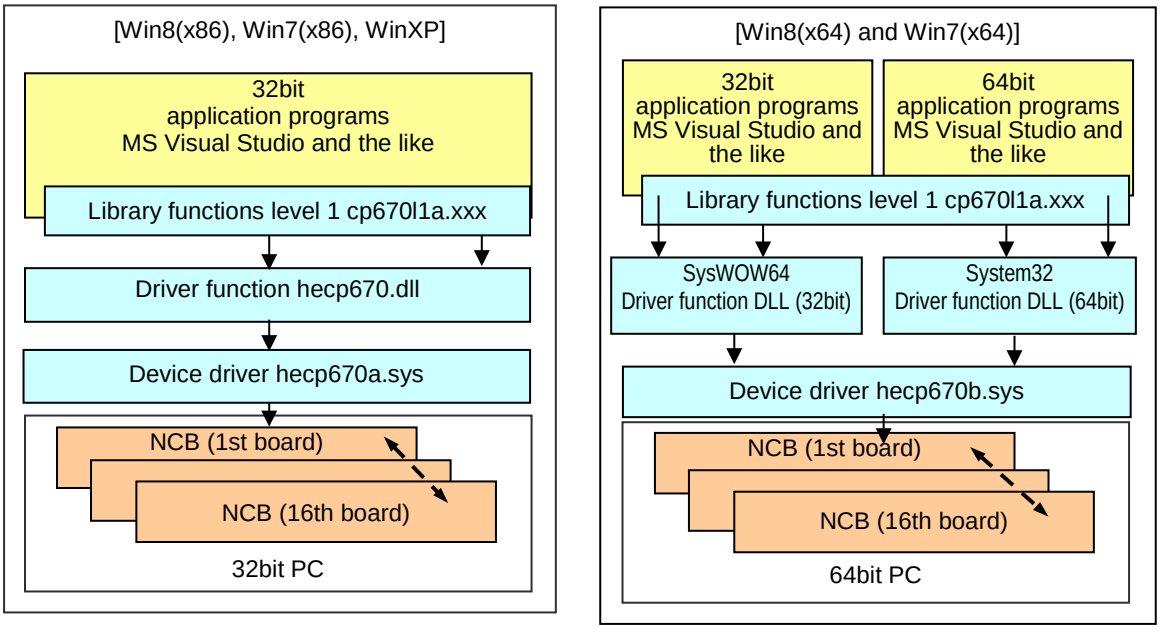


Figure 1.1-1 Software configuration

1.2 Axes Names

The axes are named X, Y, Z, and U.

2. Software Startup Guide

2.1 Overview

This chapter explains the following:

■ Windows version

◇ How to install/uninstall thedriver

◇ How to access a board and board ID

◇ How to use the sample programs (sources available in VC, VB, C#)

◇ How to operate and run "Getting started" (exe format)

Note that in the subsequent software-related explanation, the NCB674N board is referred to as NCB.

2.2 Installing the Device Driver

2.2.1 Installing to Win10(x64), Win8(x64) or Win7(x64)

(1) Before connecting NCB to the PC slot, turn ON the PC to start Windows.
(2) Start CD drive:\x64\dpinst.exe.
(3) When "dpinst.exe" starts, click [Next] to continue.
(4) After completing the installer, turn OFF the PC and connect NCB to the PC slot.
(5) Turn ON the PC to start Windows.
(6) The device installs automatically. When prompted, restart to complete the installation.

2.2.2 Installing to Win10(x86), Win8(x86), Win7(x86), or WinXP

(1) Before connecting NCB to the PC slot, turn ON the PC to start Windows.
(2) Start CD drive:\x86\dpinst.exe.
(3) When "dpinst.exe" starts, click [Next] to continue.
(4) After completing the installer, turn OFF the PC and connect NCB to the PC slot.
(5) Turn ON the PC to start Windows.
(6) When WinXP starts, the system detects NCB and automatically displays the screen for installing the necessary device driver.
(7) Check [Install the software automatically (Recommended)]. Follow the directions given by the system to complete the installation.

2.2.3 Uninstalling the Device Driver

With Windows 10

Select [All Programs] → [Windows System Tool] → [Control Panel] → [Program and Features]
→ right-click [Windows Driver Package Hivertec HPCIe-NCB674N] to uninstall.

With Windows 8

Select [All apps] → [Control Panel] → [Uninstall a program]
→ right-click [Windows Driver Package Hivertec HPCIe-NCB674N] to uninstall.

With Windows 7

Select [Start] → [Control Panel] → [Uninstall a program]
→ right-click [Windows Driver Package Hivertec HPCIe-NCB674N] to uninstall.

With Windows XP

Select [Start] → [Control Panel] → [Add or Remove Programs]
→ right-click [Windows Driver Package Hivertec HPCIe-NCB674N] and uninstall from [Change/Remove].

2.3 Using Multiple Boards

The following describes how to connect a board one by one with an external connection when there are two or more NCB boards connected to one PC. (see below picture)

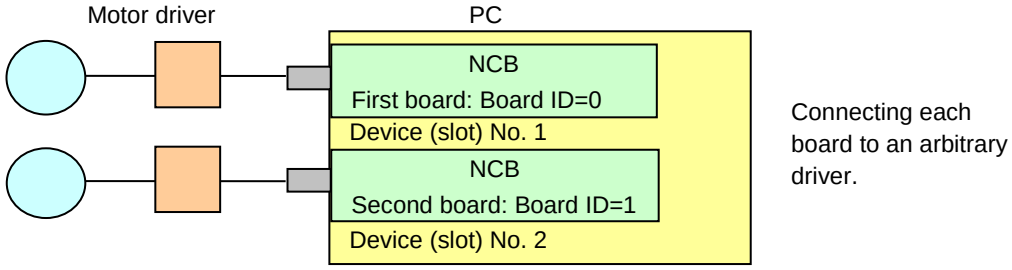


Figure 2.3-1 Using two or more boards

2.3.1 Board slot number and board ID

In PCI, board addresses are managed by the system.

The slots where boards are connected have a certain device number assigned by the system.

However, this device number does not allow the relationship between board and slot to be directly recognizable from the outside since it is assigned by the system.

For this reason, NCB is provided with a "board ID" to allow pairing of board with software.

2.3.2 Board ID

A board ID can be set in the range of 0 (zero) to 15, allowing for the handling of up to 16 NCB boards.

2.4 Getting Started

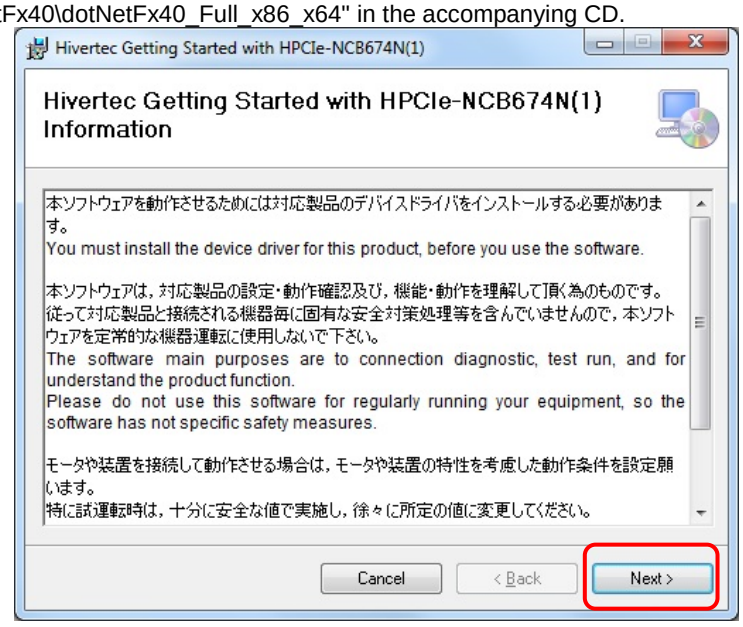
"Getting started" is a software program that allows you to confirm minimum operation of a board on the screen just by connecting it to a PC.

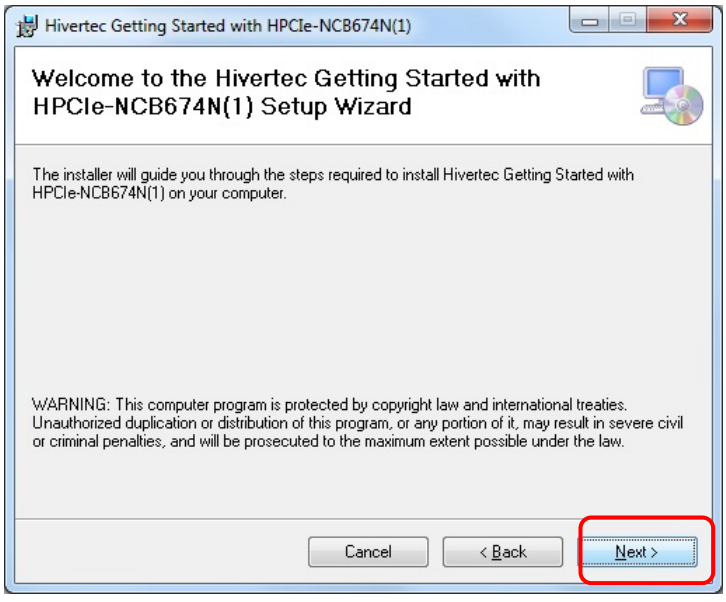
2.4.1 System requirements

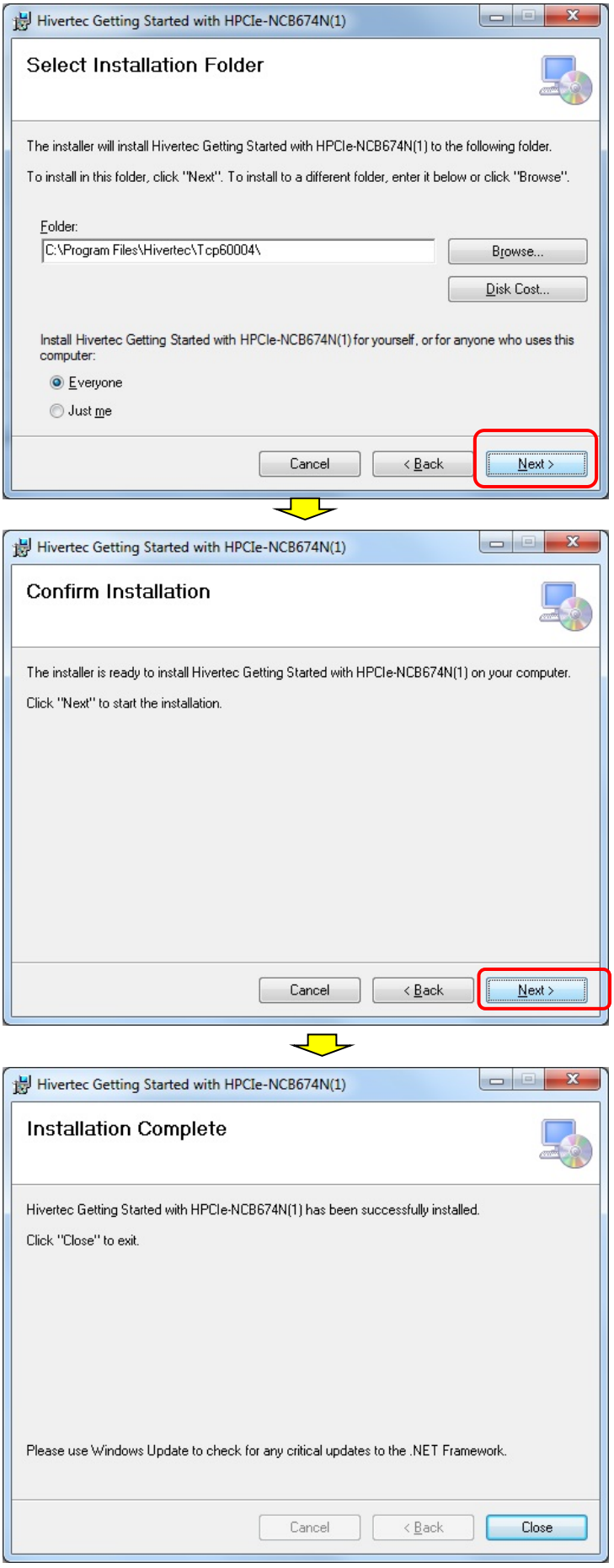
- OS: Windows XP SP3 + .NET Framework 4.0, Windows 7 + .NET Framework 4.0, Windows 8
- Hard disk: 500MB free space (when .NET Framework 4.0 is not installed)
- Processor: 1GHz or faster Pentium processor or equivalent
- RAM: 512MB or more
- Display: 800 x 600, 256 colors or more
- Other necessary components: Windows installer 3.1 or higher

2.4.2 Installation

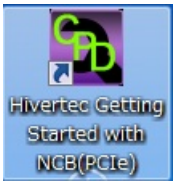
Running "\test\Setup.exe" in the accompanying CD installs "Getting started" in accordance with the sequence shown below. When installing on Windows XP or Windows 7, be sure to install .NET Framework 4.0 before installing "Getting started" if it is not yet installed.
To install .NET Framework 4.0, either download it from Microsoft's website or run "\test\DotNetFx40\dotNetFx40_Full_x86_x64" in the accompanying CD.





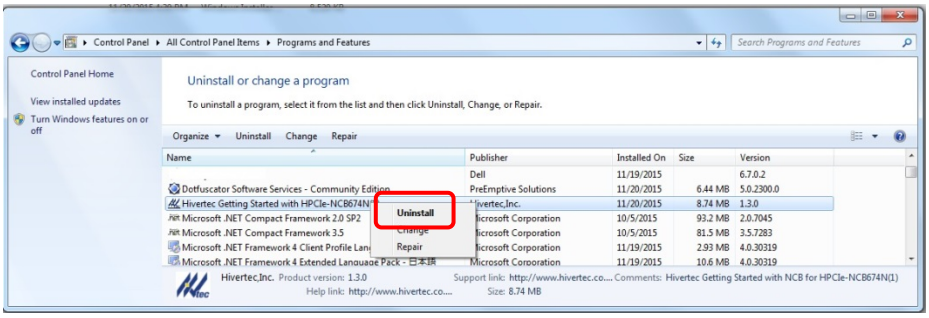

Hivertec Getting Started with HPCIe-NCB674N(1)
Select Installation Folder
The installer will install Hivertec Getting Started with HPCIe-NCB674N(1) to the following folder.
To install in this folder, click "Next". To install to a different folder, enter it below or click "Browse".
Folder:
C:\Program Files\Hivertec\Tcp60004\
Browse...
Disk Cost...
Install Hivertec Getting Started with HPCIe-NCB674N(1) for yourself, or for anyone who uses this computer:
Everyone
Just me
Cancel
< Back
Next >
Hivertec Getting Started with HPCIe-NCB674N(1)
Confirm Installation
The installer is ready to install Hivertec Getting Started with HPCIe-NCB674N(1) on your computer.
Click "Next" to start the installation.
Cancel
< Back
Next >
Hivertec Getting Started with HPCIe-NCB674N(1)
Installation Complete
Hivertec Getting Started with HPCIe-NCB674N(1) has been successfully installed.
Click "Close" to exit.
Please use Windows Update to check for any critical updates to the .NET Framework.
Cancel
< Back
Close

When installation is complete, an icon similar to the one shown below is added to the desktop and [Start menu] → [Hivertec] → [PCIexpress NCB].



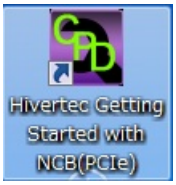
2.4.3 Uninstallation

Uninstalling "Getting started" is possible from [Control Panel] → [Uninstall a program] or [Add or Remove Programs], depending on your OS.



2.4.4 Running the program

Clicking the icon shown below starts the program.



The following screen displays when the program starts successfully.

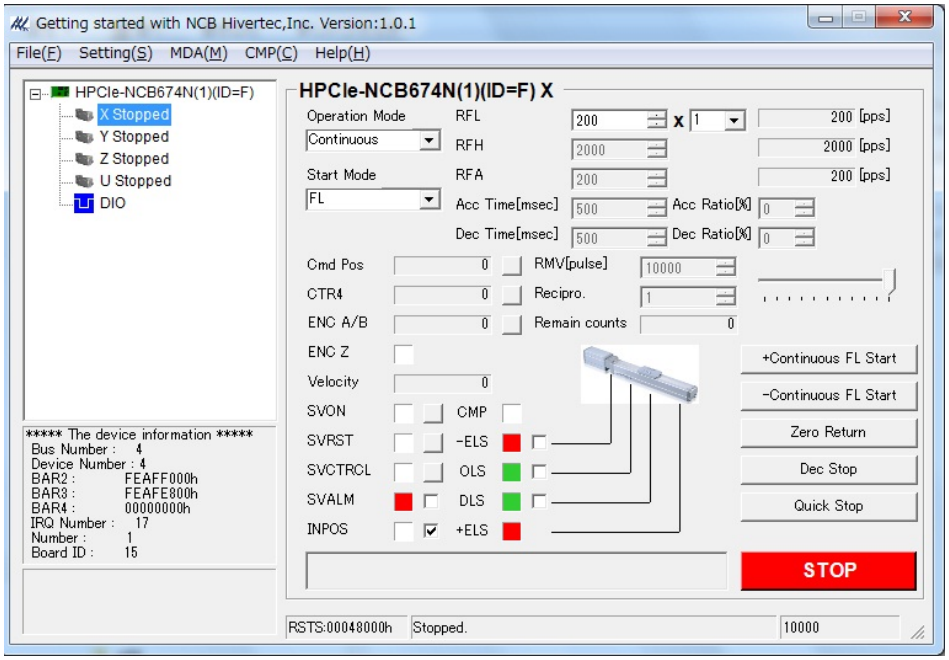
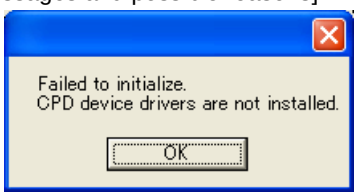


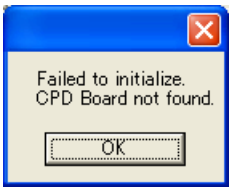
Figure 2.4-1 "Getting started" start screen

If any of the following error messages displays when started, the program will not run.

[Error messages and possible reasons]



The device driver is not installed.



The board is not connected.

Figure 2.4-2 Error messages and possible reasons

2.4.5 Items to check when each axis is ready for operation

Confirm the following once each axis is connected to the motor and operable:

- Operation of the ±ELS signal. (Operate only the sensor without operating the motor)
- The signal input status when the servo alarm signal is connected.
- The input status of the homing signal (OLS and Phase-Z).
- The input status of the in-position signal (positioning complete: INPOS).

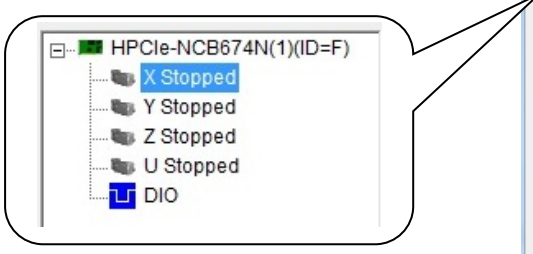
Proper operation is not guaranteed unless the above signals are input correctly.

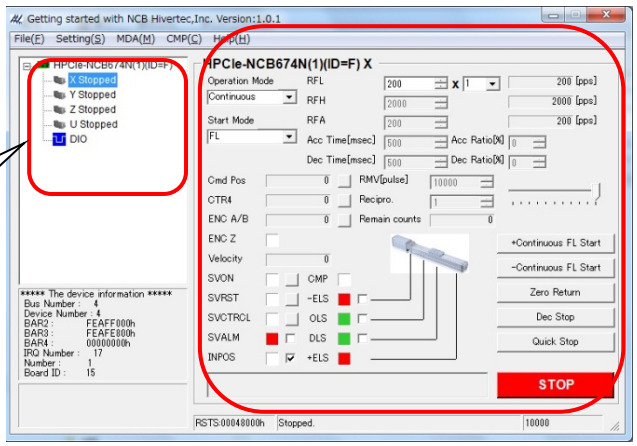
If the motor does not operate properly even after it receives the command pulse output, check the following:

- Whether the command pulse output match the "servo driver" input.
- Whether the "servo driver" input signal includes reasons for stopping the motor.

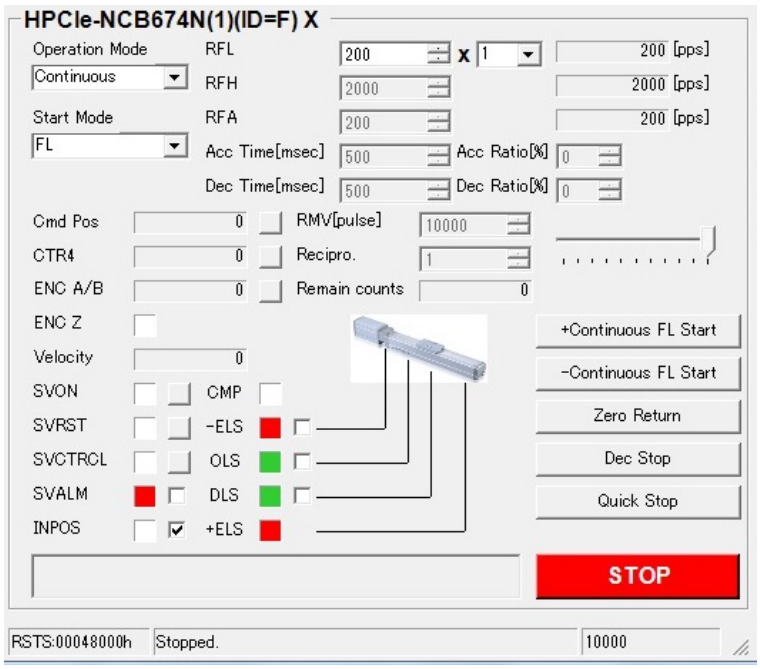
2.4.6 Selecting an axis

Use this list box to select the axis to control.
The figure shows an example where the X-axis is selected.



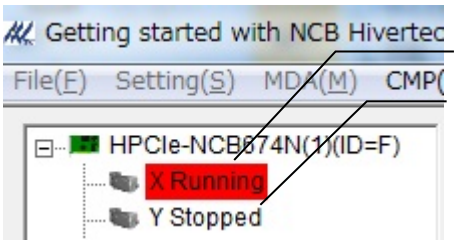


Selecting an axis displays the following screen for axis control:



Continued to the next page

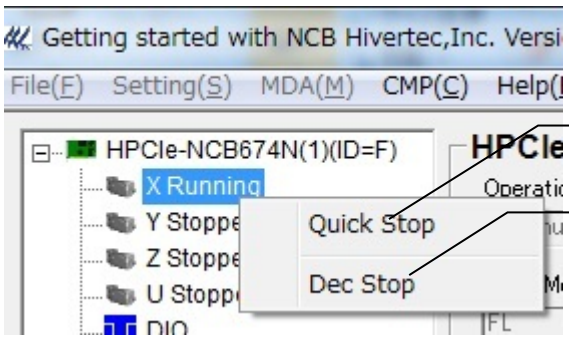
The list box also displays whether axes are "Stopped" or "Running".
Right-clicking the axis allows you to decelerate and stop or quick-stop it.



"Running" axis indication

"Stopped" axis indication

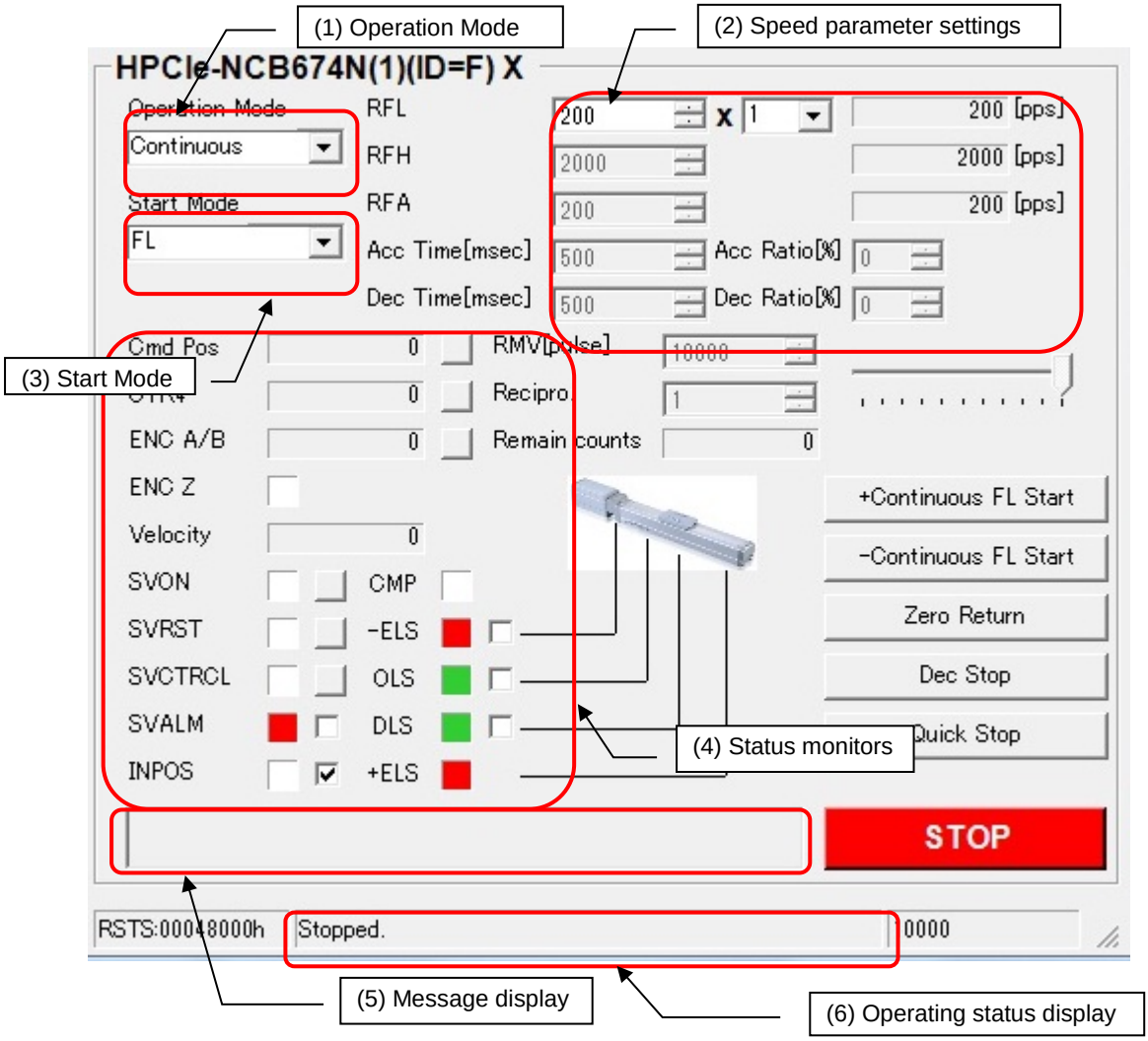
Right-clicking displays a menu.



Click to stop quickly.

Click to decelerate and stop.
In reciprocating operation, this stops the cycle.

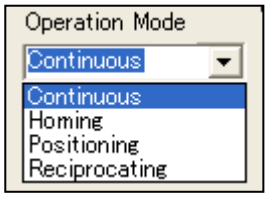
2.4.7 Axis control screen



(1) Operation Mode

Select one from the following four modes:

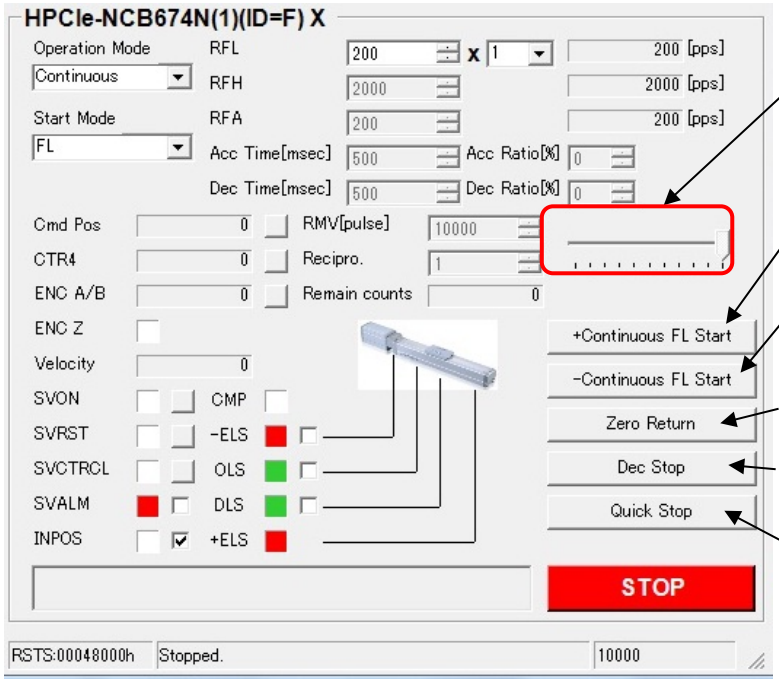
"Continuous" is selected at startup.



Continuous
Homing
Positioning
Reciprocating

• **(a) Continuous**

Operation mode in which only the speed and direction are specified so that operation continues until a Stop command is issued.



0 to 100% speed override, where 100% is the set operation speed.

Continuous feed in the positive direction based on the Start Mode

Continuous feed in the negative direction based on the Start Mode.

Performs positioning to command position 0.

Decelerates and stops the operation.

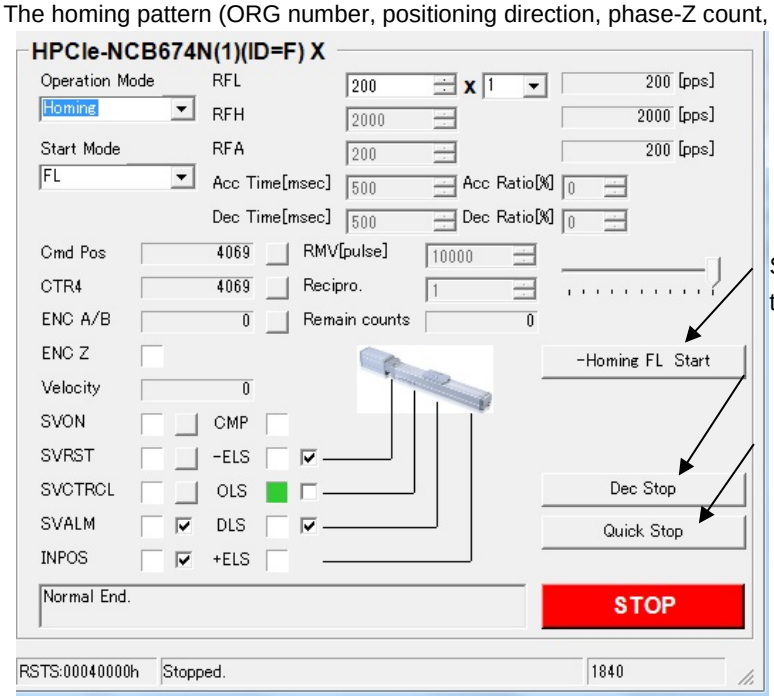
Quick Stop

Quickly stops the operation.

• **(b) Homing**

Mode for returning to the origin by using the origin sensor, end limit signal, phase-Z, etc.

The homing pattern (ORG number, positioning direction, phase-Z count, etc.) is set on the configuration screen.



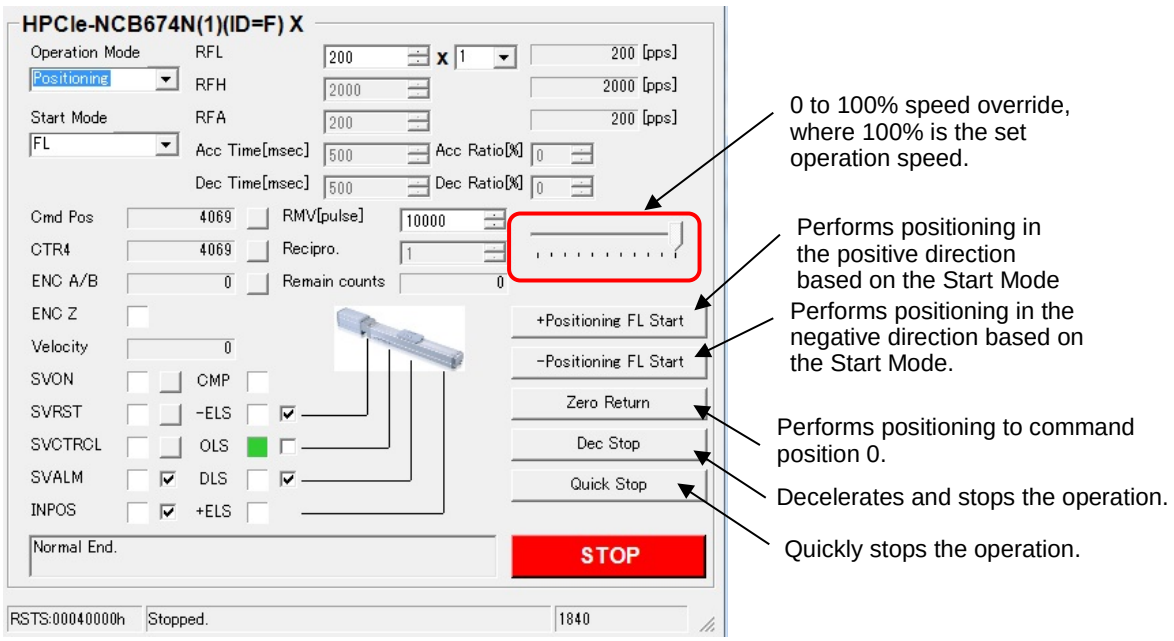
Starts homing based on the Start Mode.

Decelerates and stops the operation.

Quickly stops the operation.

• **(c) Positioning**

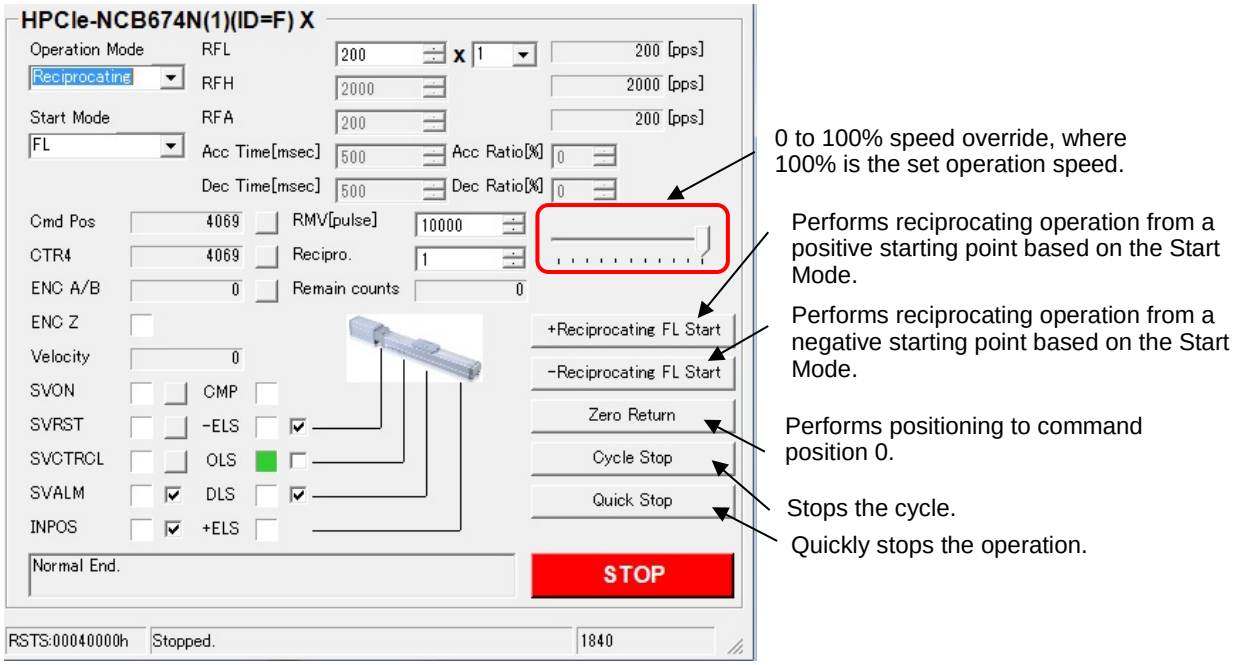
Performs positioning for the specified travel distance.



• **(d) Reciprocating**

Repeats reciprocating operation for the specified travel distance and for the specified number of times. Clicking [Cycle Stop] stops the operation when the current reciprocating operation is completed.

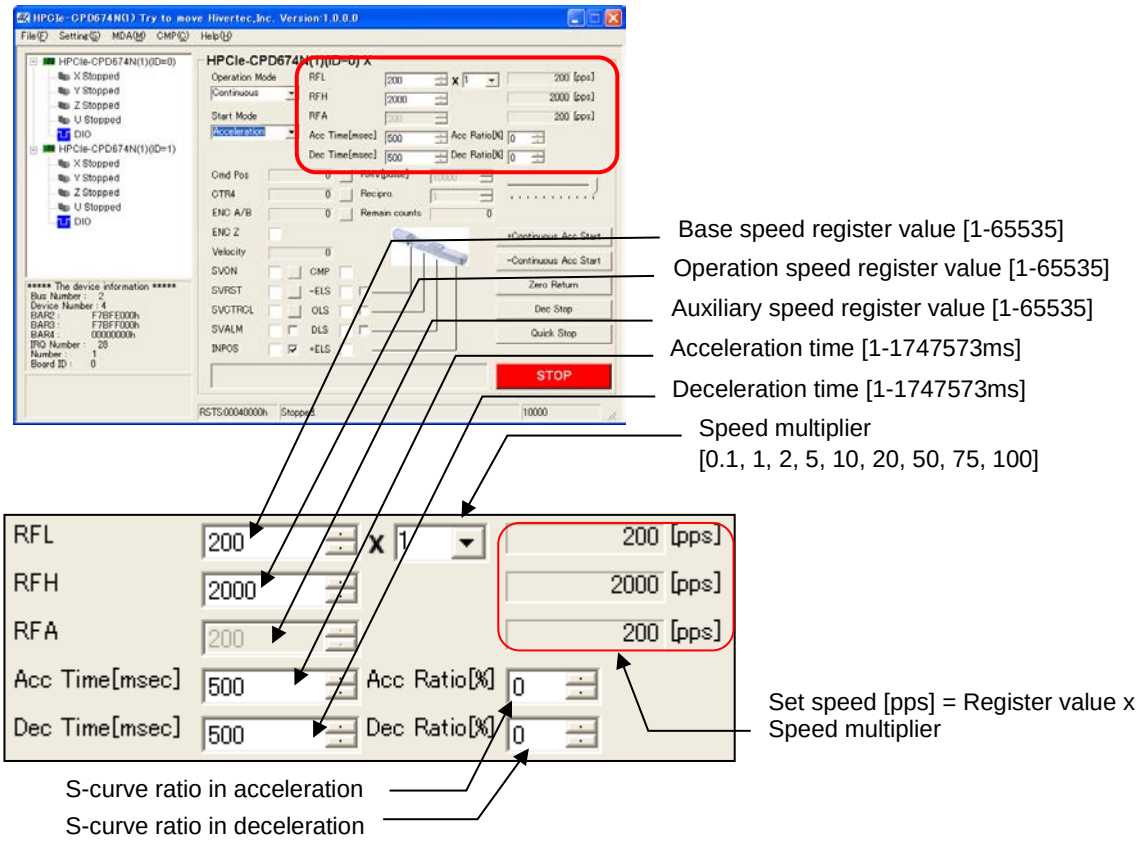
Clicking [Quick Stop] quickly stops the command pulses.



(2) Speed parameter settings

Define the speed pattern by setting the speed parameters below.

However, note that there are some invalid values for speed depending on the combination of parameters, travel distance, etc.



(3) Start Mode

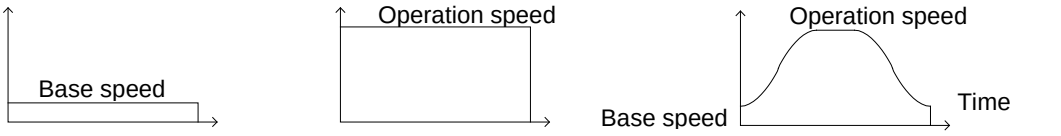
Select the Start Mode.

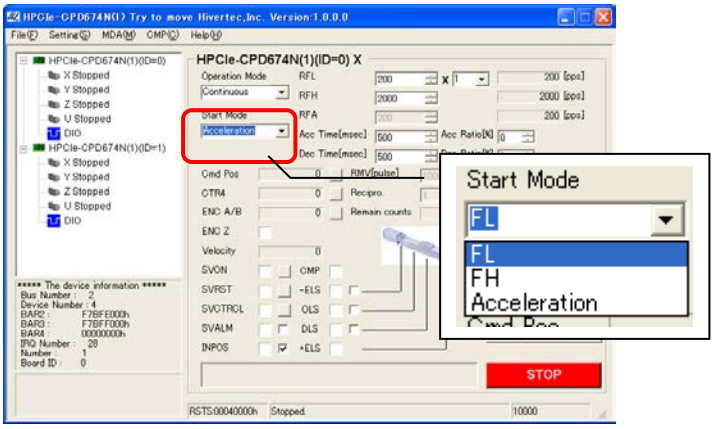
With FL constant-speed start, the operation will be operation at FL constant speed.

With FH constant-speed start, the operation will be operation at FH constant speed.

With acceleration start, the operation will be acceleration/deceleration operation.

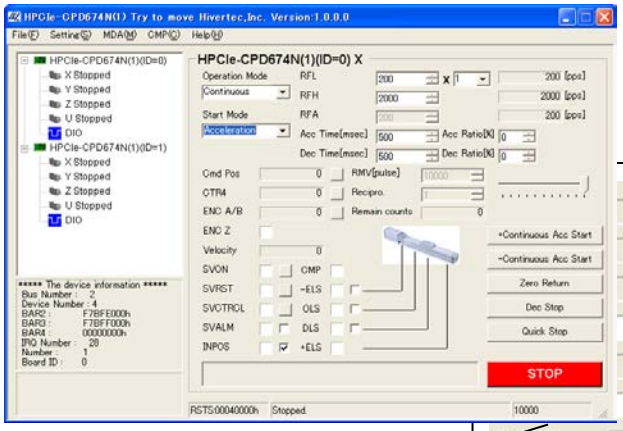
[FL constant-speed operation] [FH constant-speed operation] [Acceleration/deceleration operation]





(4) Status monitors

Status monitors allows you to monitor the status of the command counter, machine counter, servo I/F, machine I/F, and others.



Command counter [pulse]
Counter 4 [pulse]
Machine counter [pulse]
Command counter reset button
Counter 4 reset button
Machine counter reset button
Encoder phase-Z Green: ON, White: OFF
Current command speed [pps]
CMP output status Green: ON, White: OFF
ELS input polarity change switch
OLS input polarity change switch
DLS input polarity change switch

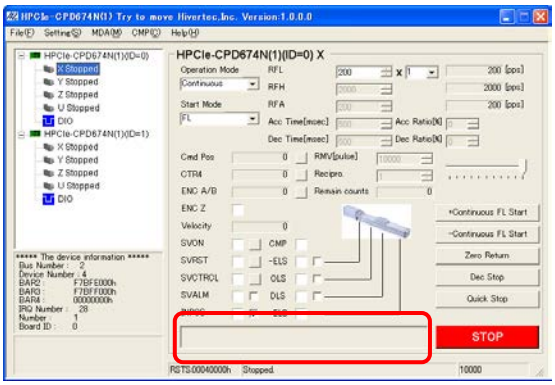
SVON output status Green: ON, White: OFF
SVRST output status Green: ON, White: OFF
SVCTRCL output status Green: ON, White: OFF
SVALM input status Red: ON, White: OFF
INPOS input status Green: ON, White: OFF
SVON output reverse button
SVRST output reverse button
SVCTRCL output
The output reverses when the level of output width is set.
SVALM input polarity change switch
INPOS input polarity change switch

-ELS Red: ON. White: OFF
OLS Green: ON, White: OFF
DLS Green: ON, White: OFF
±ELS Red: ON, White: OFF

※ Input polarity change switch Checked: Normal open, Unchecked: Normal close

(5) Message display area

Displays the cause of stoppage on operation complete. Also displays a message when the encoder signal is in error.



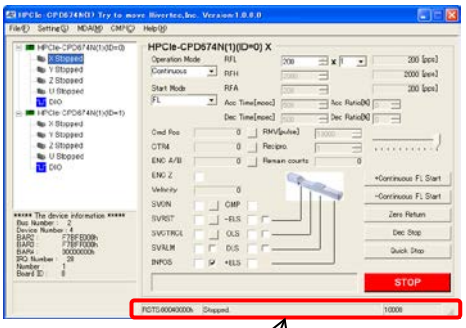
[Causes of stoppage]
"Normal End." • • • Normal stop
"Stop by SVALM." • • • Stop by SVALM input
"Stop by -ELS." • • • Stop by -ELS input
"Stop by +ELS." • • • Stop by +ELS input
"Stop by DLS." • • • Stop by DLS input
"Stop by +SLS." • • • Stop by +SLS
"Stop by -SLS." • • • Stop by -SLS
"Stop by EMG." • • • Stop by EMG input

[Encoder signal error]
"Encoder signal error"

(6) Operating status display

Displays the extension status, current operating status, and positioning counter.

For details on the extension status, refer to “User’s Manual <Operation>”.



[Current operating status]

"Stopped." • • • The axis is stopped.

"During acceleration." • • • The axis is accelerating.

"During deceleration." • • • The axis is decelerating.

"During auxiliary speed operation. (FA)"

• • • The axis is operating at FA constant speed.

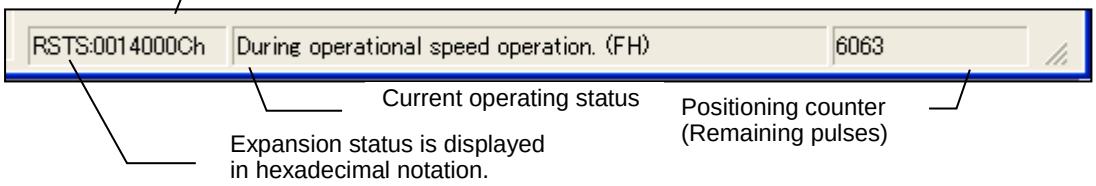
"During base speed operation. (FL)"

• • • The axis is operating at FL constant speed.

"During operational speed operation. (FH)"

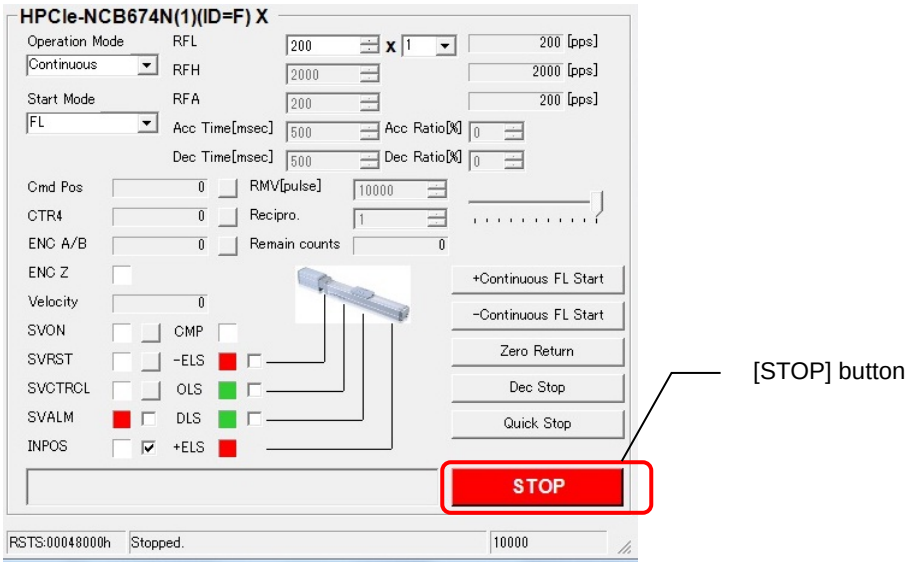
• • • The axis is operating at FH constant speed.

"Waiting for INPOS." • • • The axis is waiting for INPOS input.



(7) STOP

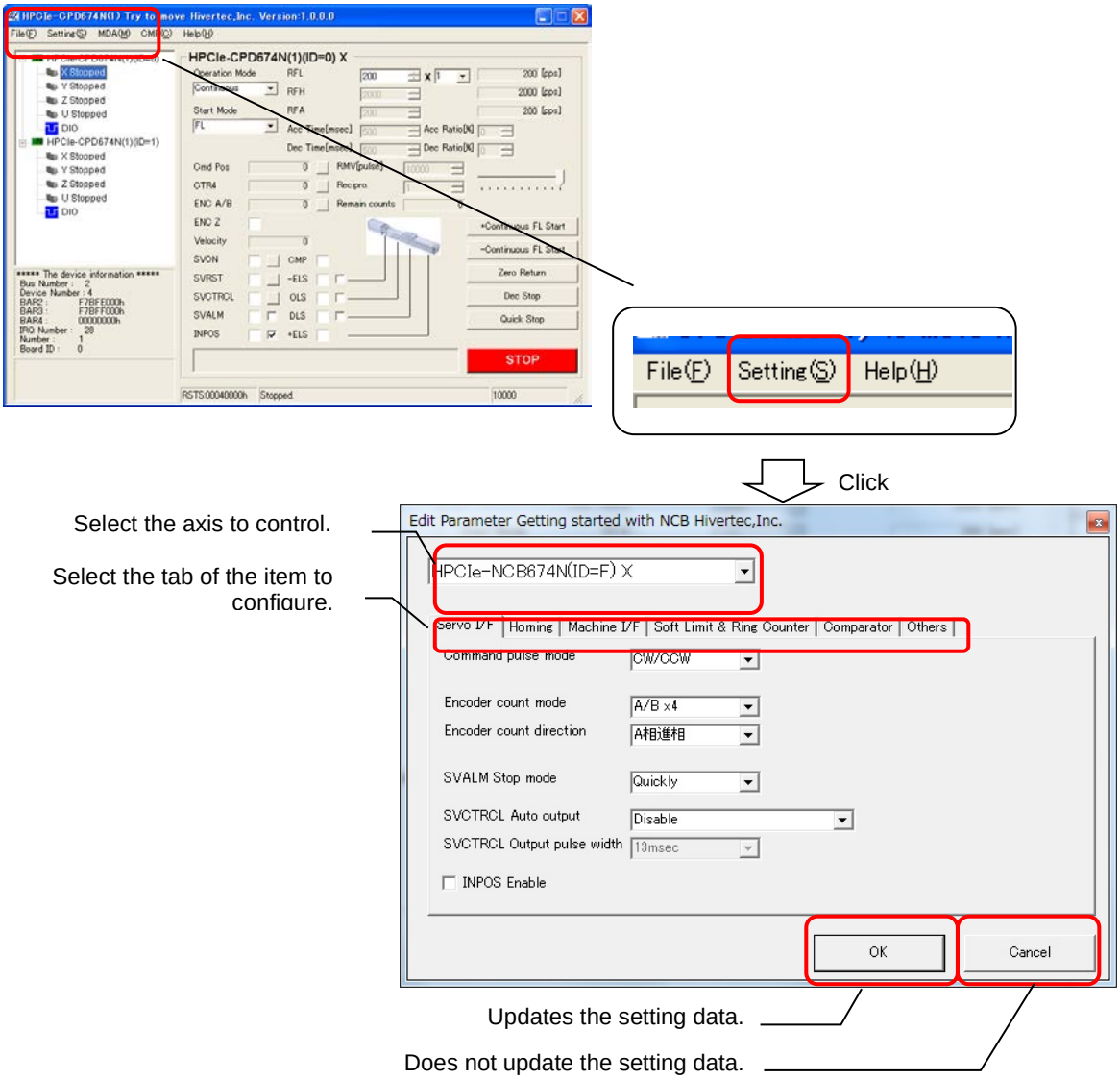
Stops all axes controlled by all boards.



2.4.8 Axis configuration screen

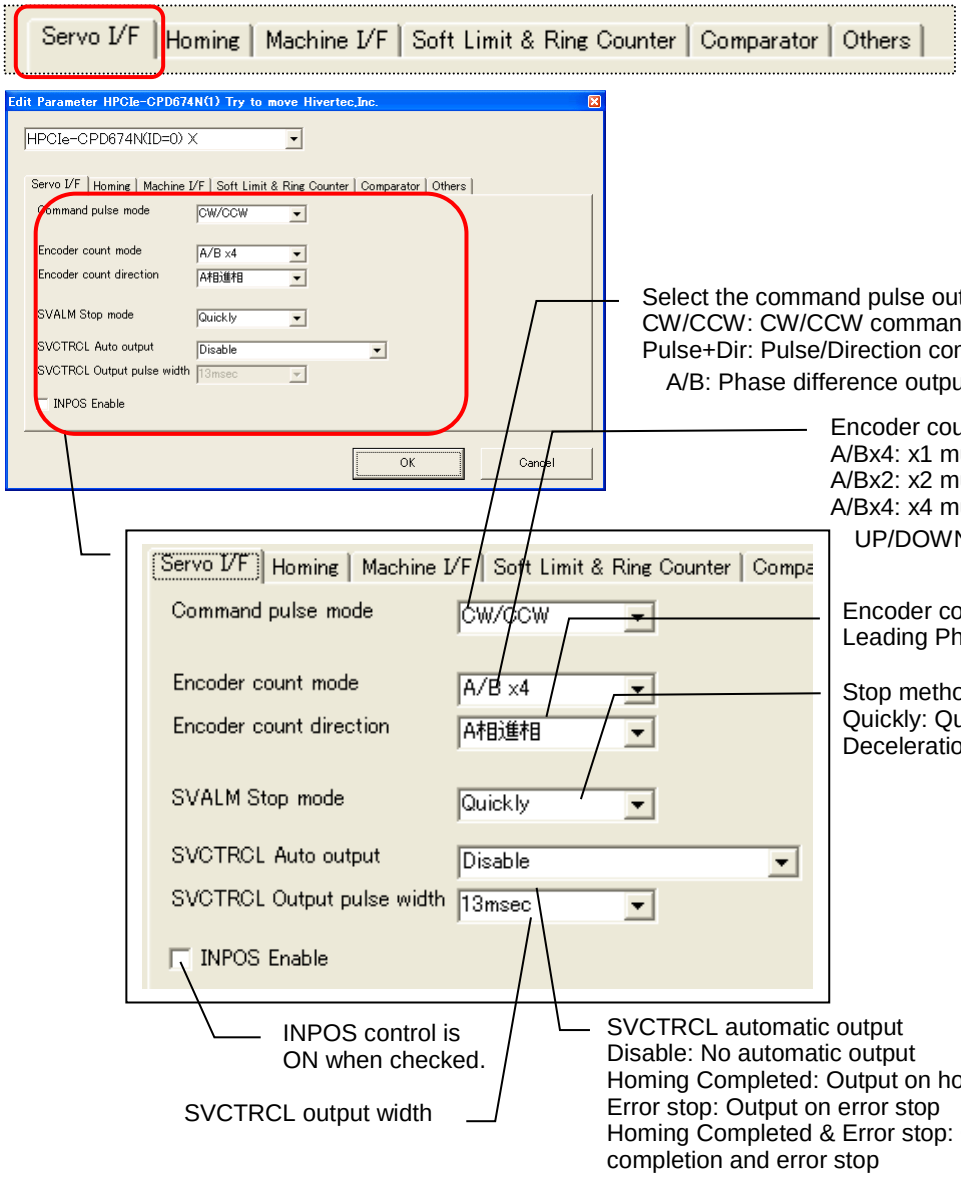
Clicking "Setting(S)" on the menu displays the configuration screen.

It allows for changing settings for the servo I/F, homing, machine I/F, software limit, comparator, etc.



(1) Servo I/F

Configures various servo I/F settings.



Select the command pulse output type
CW/CCW: CW/CCW command
Pulse+Dir: Pulse/Direction command
A/B: Phase difference output

Encoder count method
A/Bx4: x1 multiplication of phase difference
A/Bx2: x2 multiplication of phase difference
A/Bx4: x4 multiplication of phase difference
UP/DOWN: Up/Down pulse

Encoder count direction
Leading Phase A/Leading Phase B

Stop method on SVALM input
Quickly: Quick stop
Deceleration: Deceleration stop

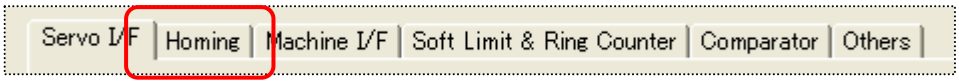
INPOS control is
ON when checked.

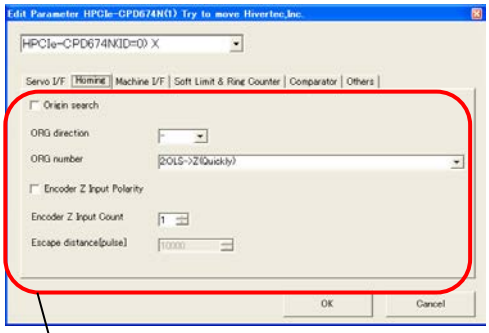
SVCTRCL output width

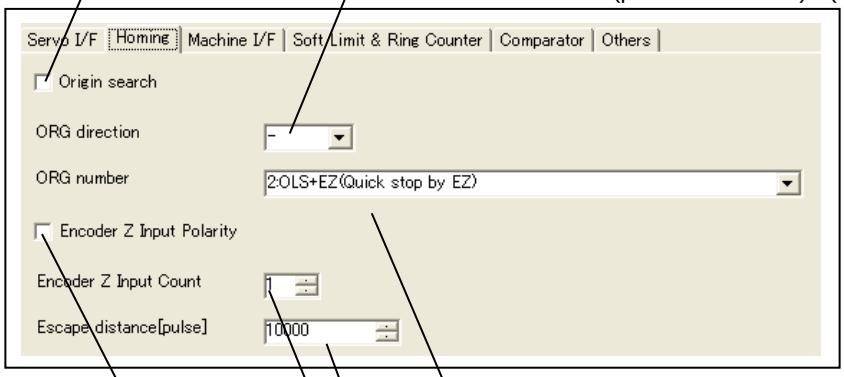
SVCTRCL automatic output
Disable: No automatic output
Homing Completed: Output on homing completion
Error stop: Output on error stop
Homing Completed & Error stop: Output on homing completion and error stop

(2) Homing

Configures various homing settings.







Origin search: Enabled when checked

ORG direction: + (positive direction)/- (negative direction)

ORG number: Determines the homing pattern
For details, refer to "User's Manual <Operation>".

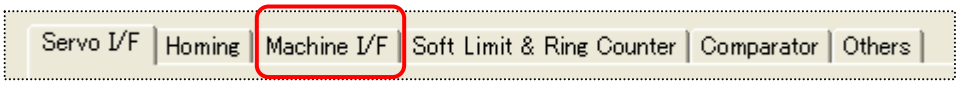
Amount of origin escape at the start of origin search [pulse]

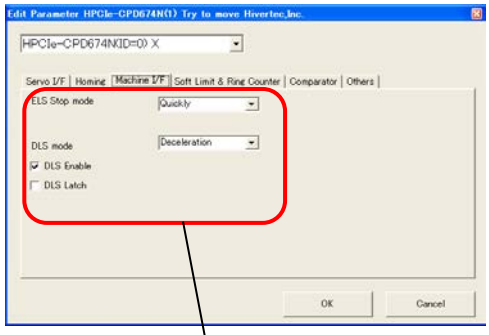
Number of phase-Z counts during homing

Encoder Z input polarity: Normal close when checked

(3) Machine I/F

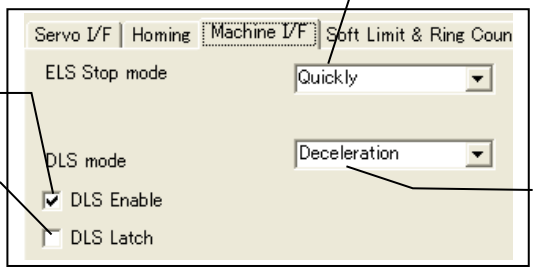
Configures machine I/F (ELS, DLS) settings.





Stop method on ELS input
Quickly: Quick stop
Deceleration: Deceleration stop

Be careful of the deceleration distance when setting Deceleration.



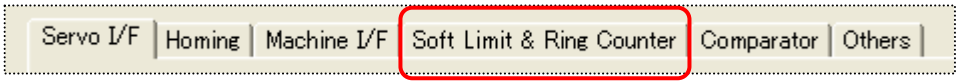
DLS Enable:
Enabled when checked

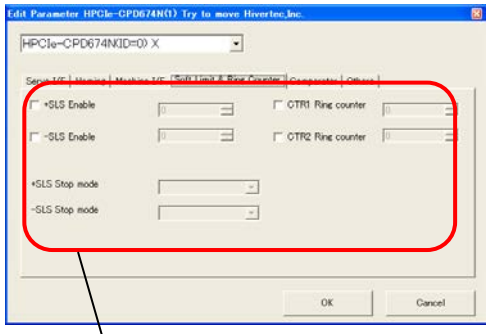
DLS Latch:
Latch enabled when checked

Operation on DLS input
Deceleration: Deceleration only
Deceleration stop: Deceleration stop

(4) Soft Limit & Ring Counter

Configures software limit or ring counter settings. Software limit and ring counter cannot be configured simultaneously.

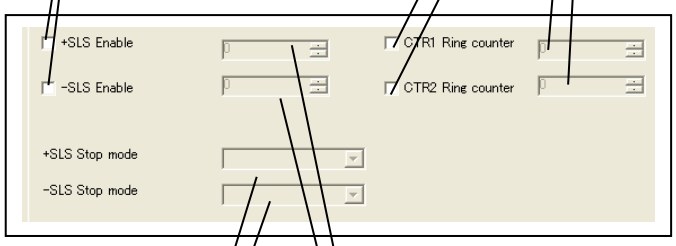




CTR1 Ring counter: Check to enable setting
CTR2 Ring counter: Check to enable setting

+SLS: Enabled when checked
-SLS: Enabled when checked

CTR1 Ring counter value [pulse]
CTR2 Ring counter value [pulse]

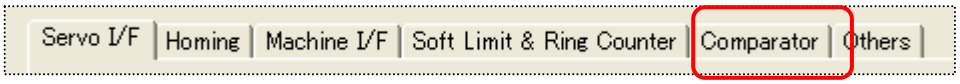


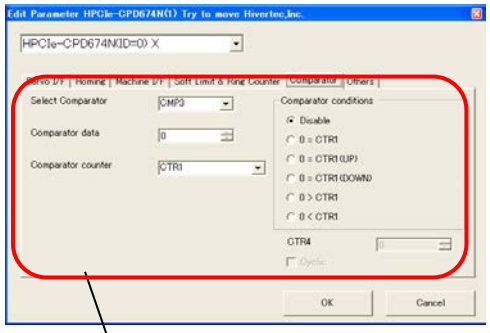
Stop method on +SLS
Stop method on -SLS

+SLS value [pulse]
-SLS value [pulse]

(5) Comparator

Configures various comparator settings.

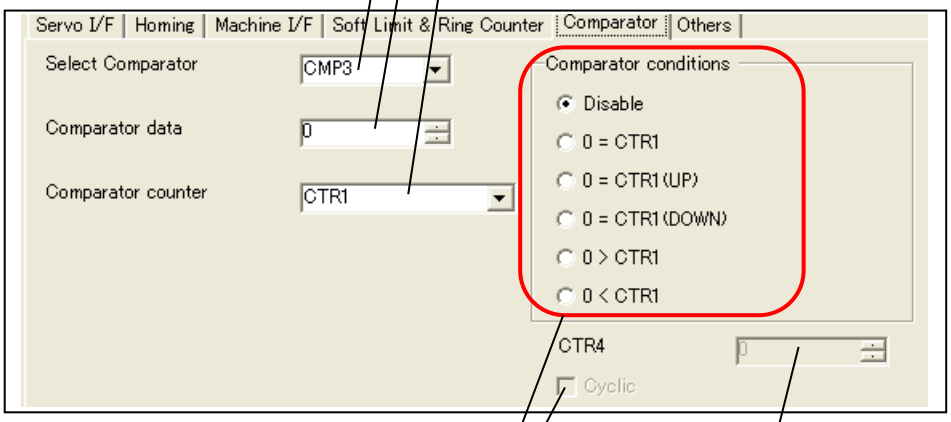




Select Comparator

Comparator data [pulse]
When comparison counter is command speed,
Command speed / Speed multiplier

Comparator comparison counter
CTR1, CTR2, CTR3
When using CMP5,
CTR1, CTR2, CTR3,
Distance to go: Positioning counter (Remaining number of pulses)
Command Velocity: Command speed



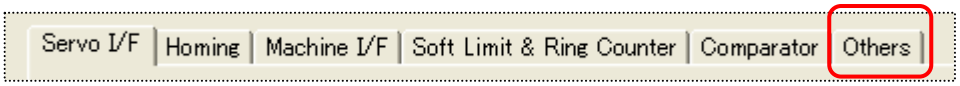
Comparator comparison condition

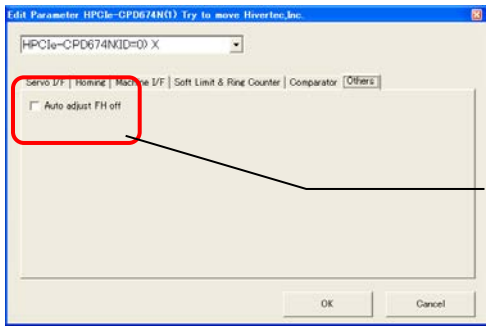
Equal pitch output when checked (When using CMP4)

Set CTR4 value [pulse] for equal pitch output by using CMP4.

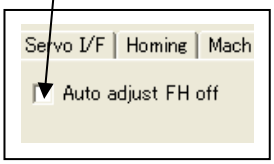
(6) Others

Enables/Disables FH auto adjustment.



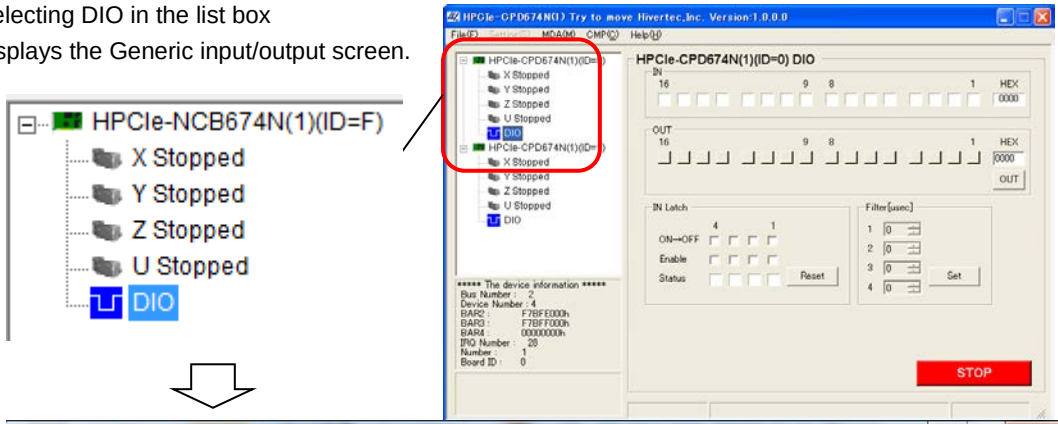


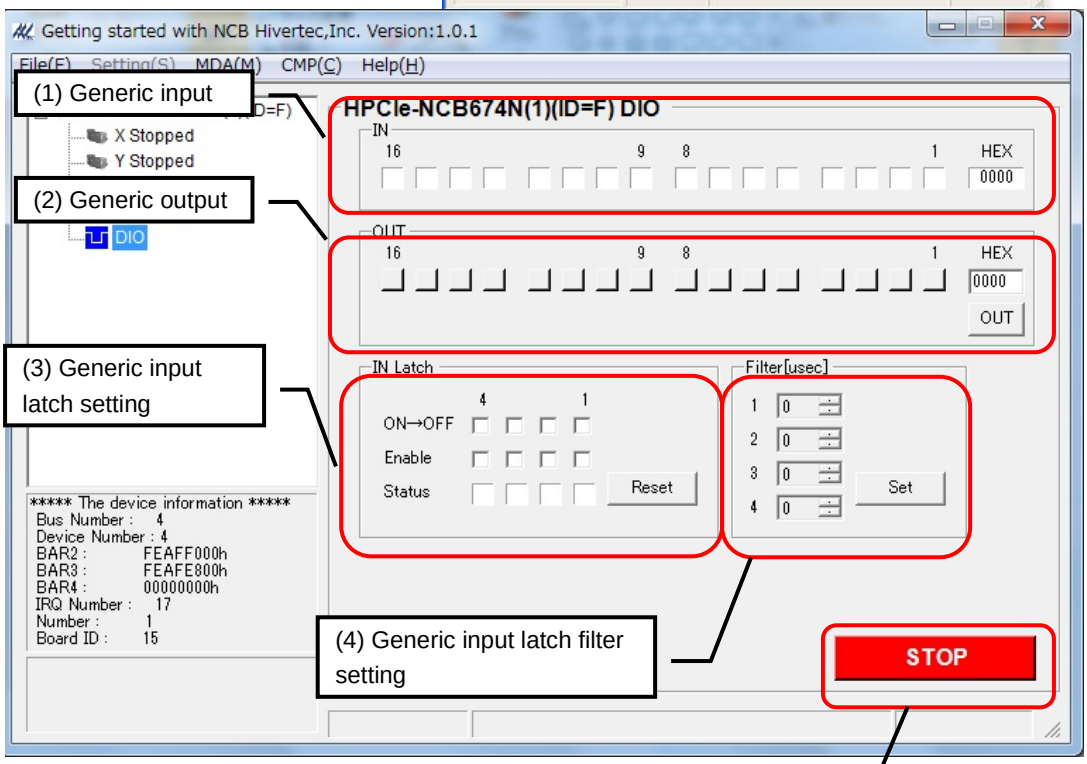
FH auto adjustment is disabled when checked.



2.4.9 Generic input/output screen

Selecting DIO in the list box displays the Generic input/output screen.





(1) Generic input

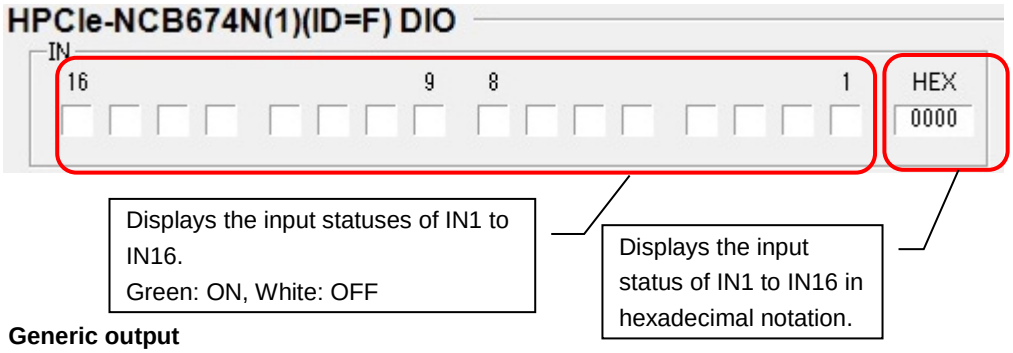
(2) Generic output

(3) Generic input latch setting

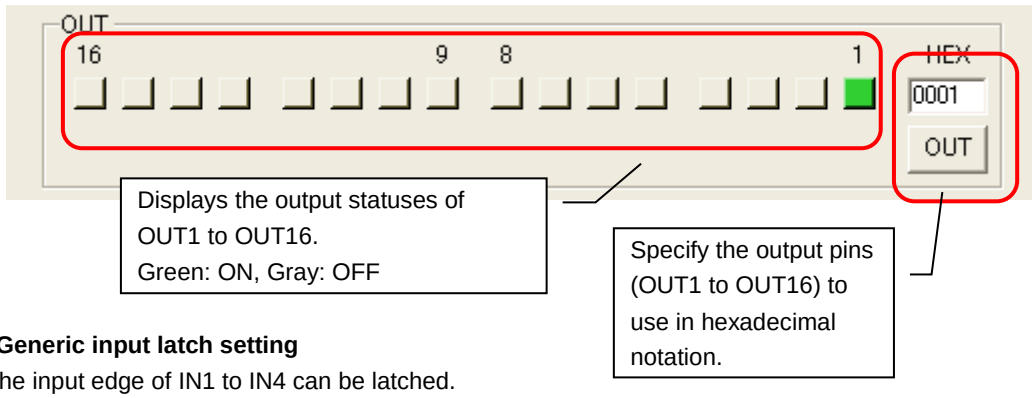
(4) Generic input latch filter setting

(5) [STOP] button
Clicking this button stops all axes controlled by all boards.

(1) Generic input

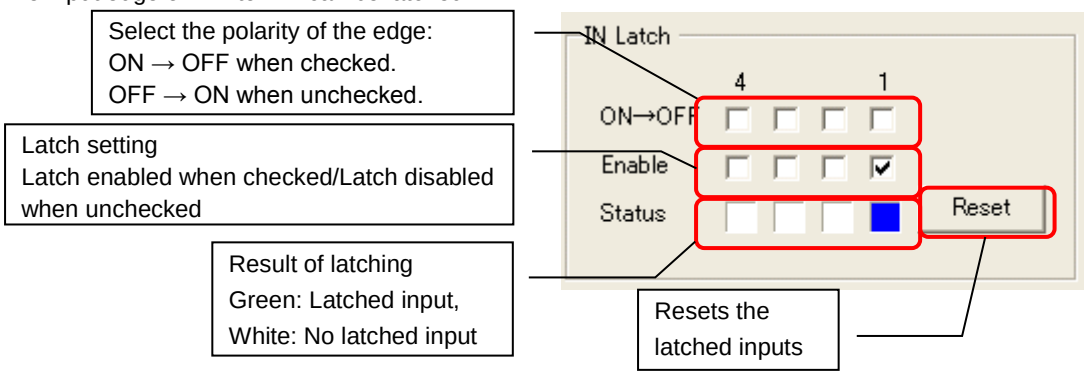


(2) Generic output



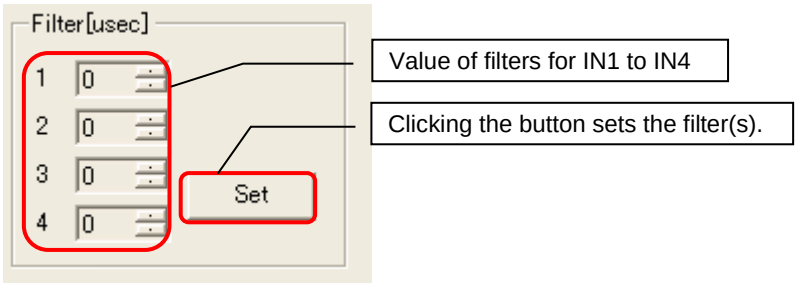
(3) Generic input latch setting

The input edge of IN1 to IN4 can be latched.



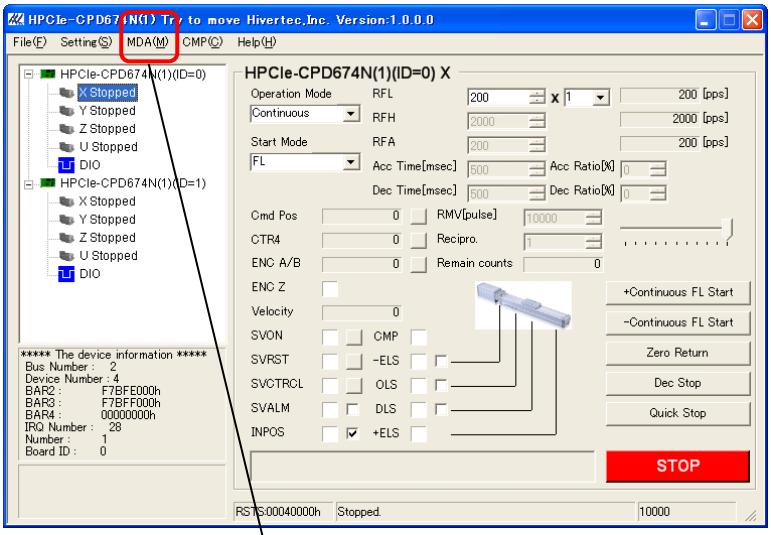
(4) Generic input latch filter setting

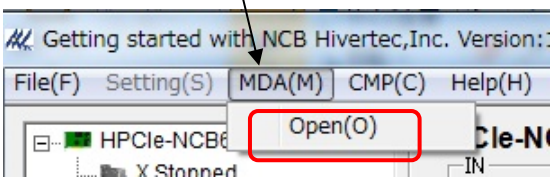
The input latch on IN1 to IN4 can be filtered. The setting range is from 0 to 6000µs (in units of 400µs).



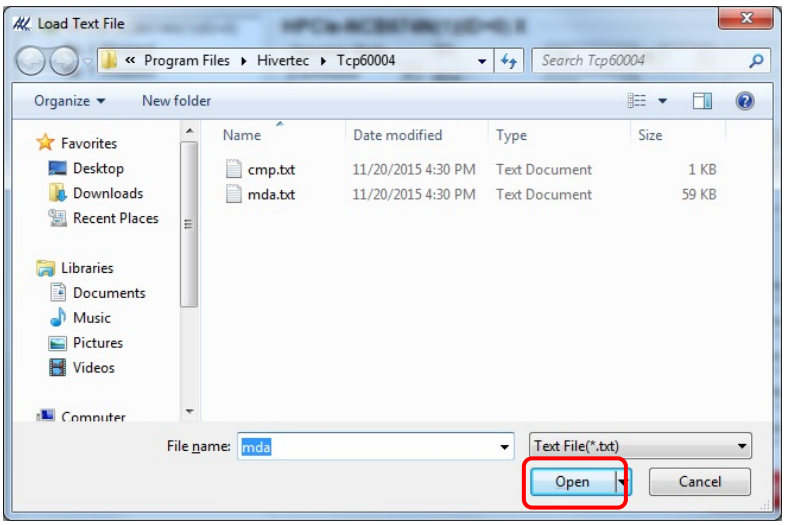
2.4.10 MDA operation screen

Allows for loading a text file with MDA data to perform MDA operation.





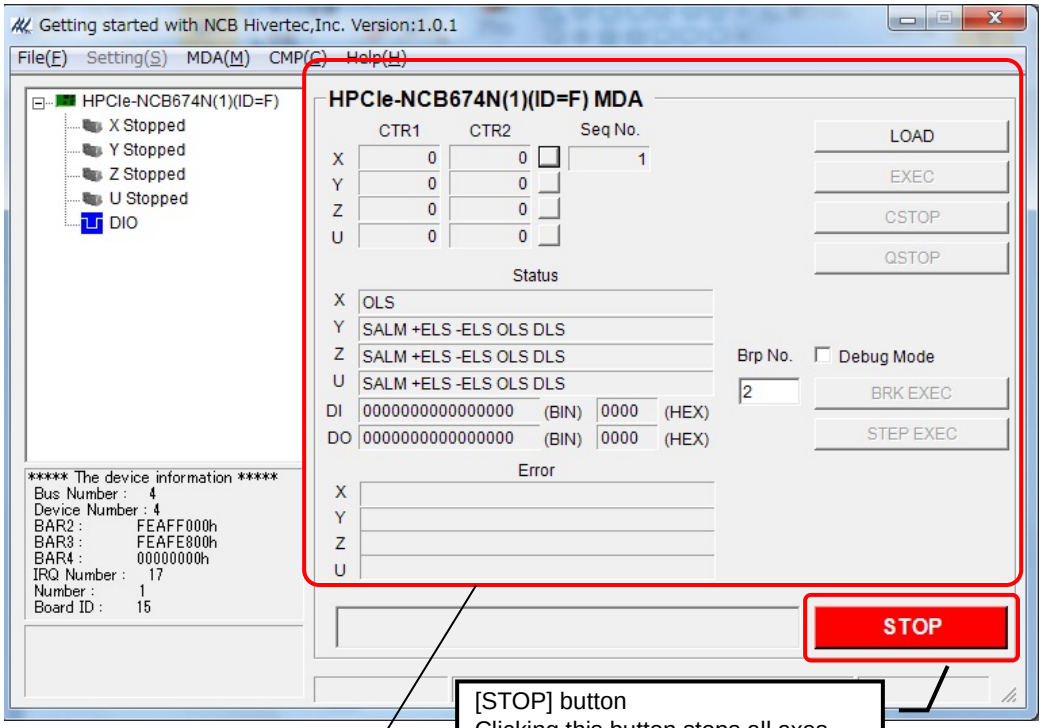
Clicking "Open(O)" opens the dialog to select the MDA file.



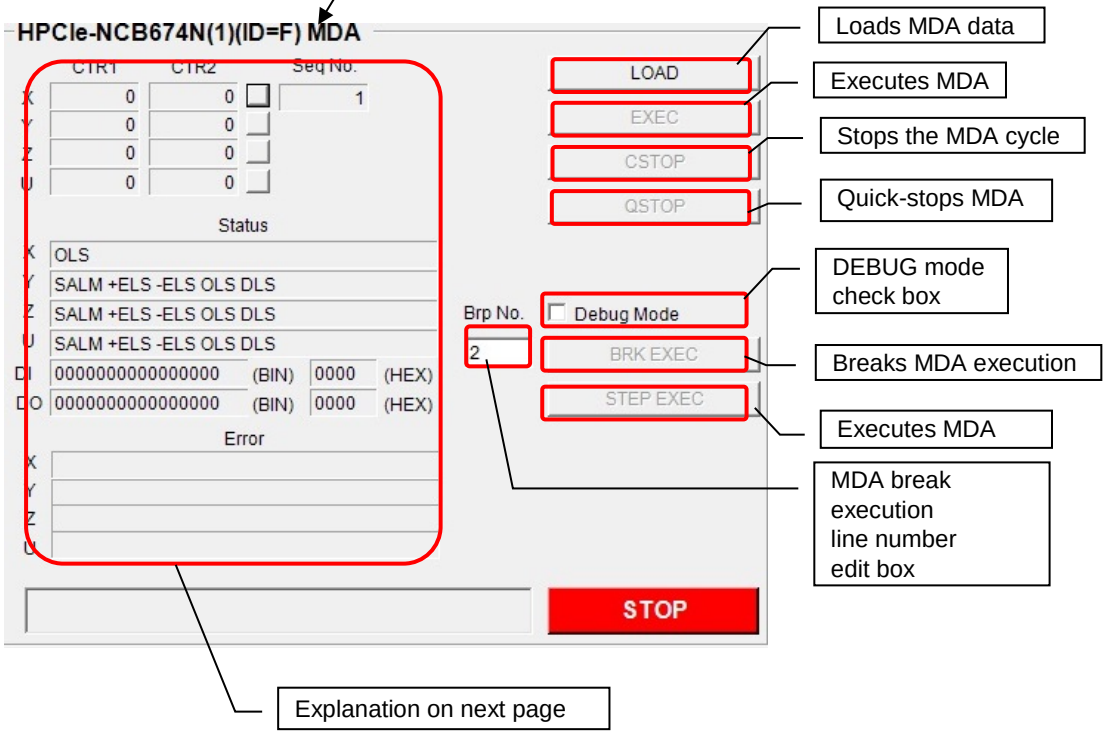
Select the file and click [Open(O)] to load the file data. The MDA operation screen for the board selected in the list box displays.

(1) MDA operation screen

Clicking [LOAD] → [EXEC] starts the MDA operation.



[STOP] button
Clicking this button stops all axes controlled by all boards.



Loads MDA data

Executes MDA

Stops the MDA cycle

Quick-stops MDA

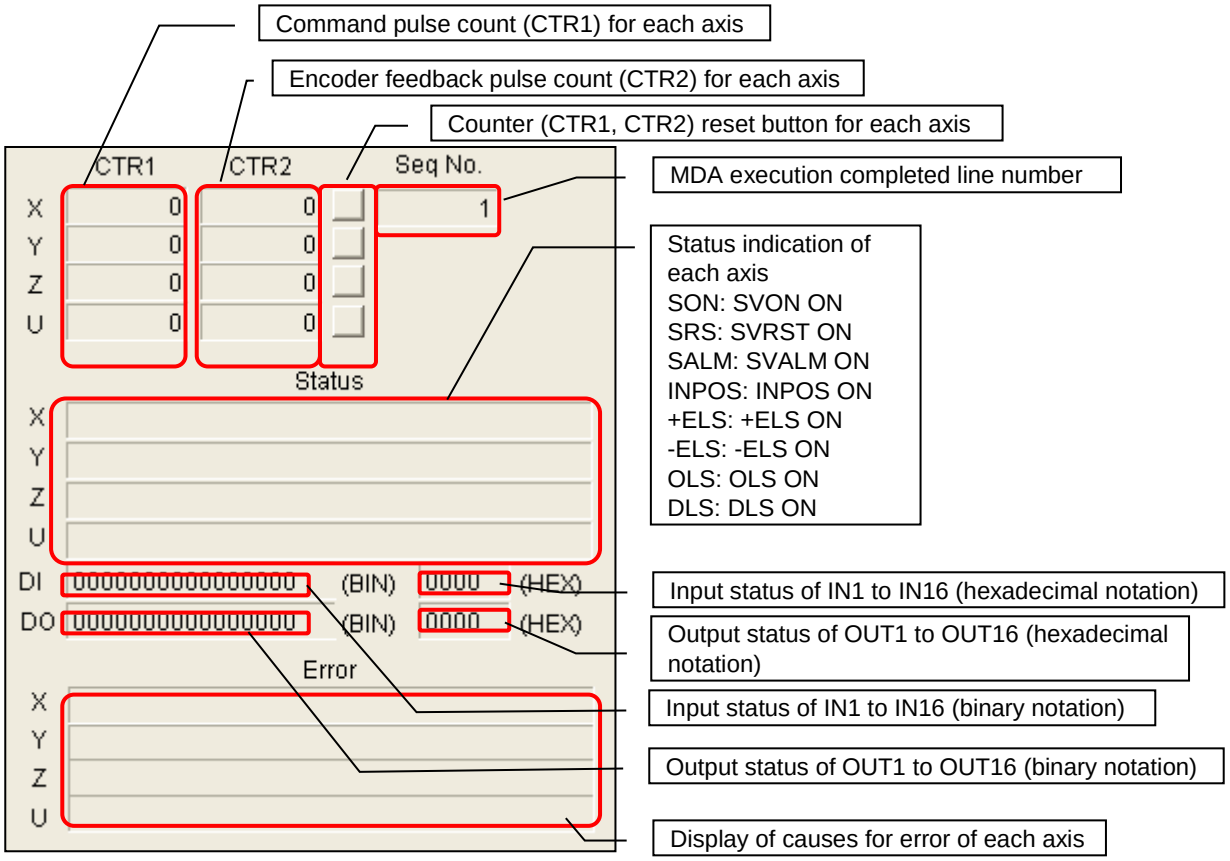
DEBUG mode check box

Breaks MDA execution

Executes MDA

MDA break execution line number edit box

Explanation on next page



(2) MDA data file format

Extension: txt

SEQ_NO. is a decimal string while other data is a hexadecimal string. Each line always consists of four data.

The first line is always the top initialization block (header), and the last line is always the end block (footer).

SEQ_NO, DATA1, DATA2, DATA3(CR+LF)

For details of DATA1, DATA2, and DATA3 see "5.6 Contents of DATA1(CND/CMD)" and "5.7 Details of MDA Operation Block".

<Example>

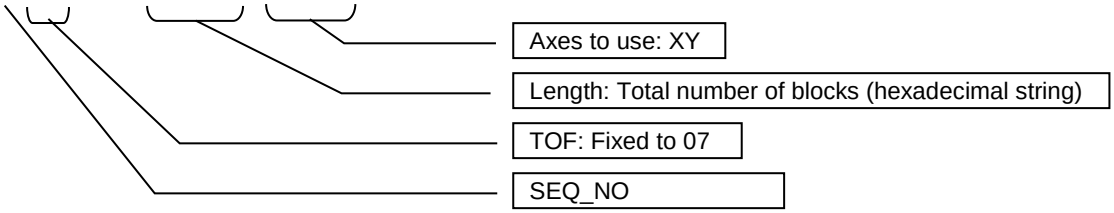
1,0700,0B49,0003
2,0300,8061,0800
3,0400,8061,0800
4,0000,0387,0000
•
•
2889,0813,0000,0000

Explanation on the next page

<Explanation>

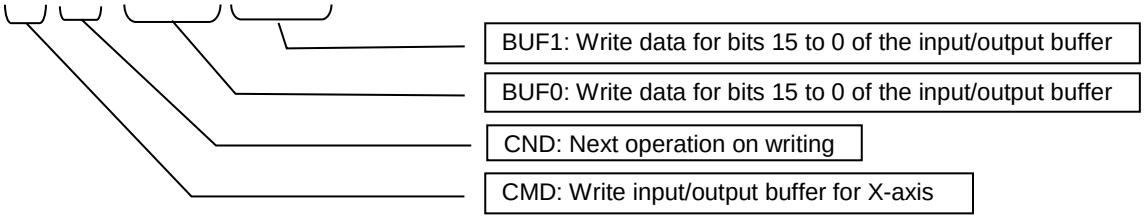
The first line is the top initialization block; it indicates the total number of blocks (total number of lines): 2889(0B49h) and the axes to use: X and Y.

1, 0 7 0 0, 0 B 4 9, 0 0 0 3



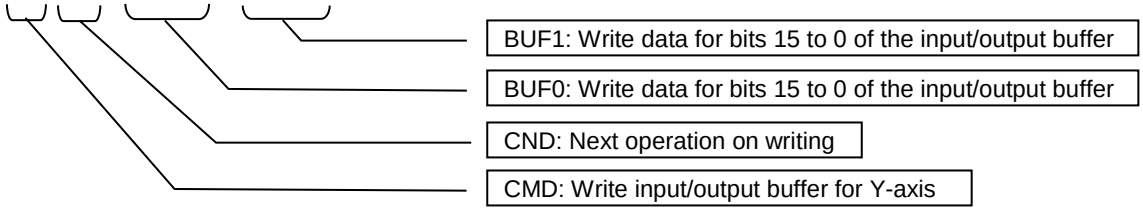
Lines 2 to 4 below are settings for the PRMD register for axes X and Y.
The second line writes data to the input/output buffer for X-axis.

2, 0 3 0 0, 8 0 6 1, 0 8 0 0



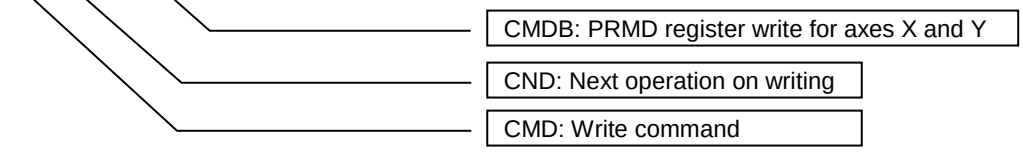
The third line writes data to the input/output buffer for Y-axis.

3, 0 4 0 0, 8 0 6 1, 0 8 0 0



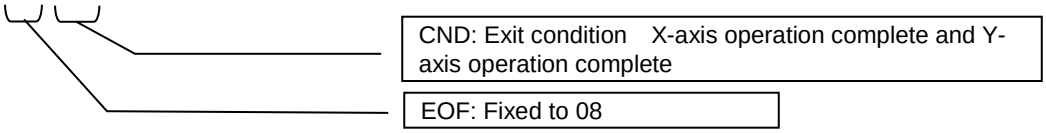
The fourth line writes the PRMD registers for axes X and Y.

4, 0 0 0 0, 0 3 8 7, 0 0 0 0



The 2889th line is the last block

2 8 8 9, 0 8 1 3, 0 0 0 0, 0 0 0 0

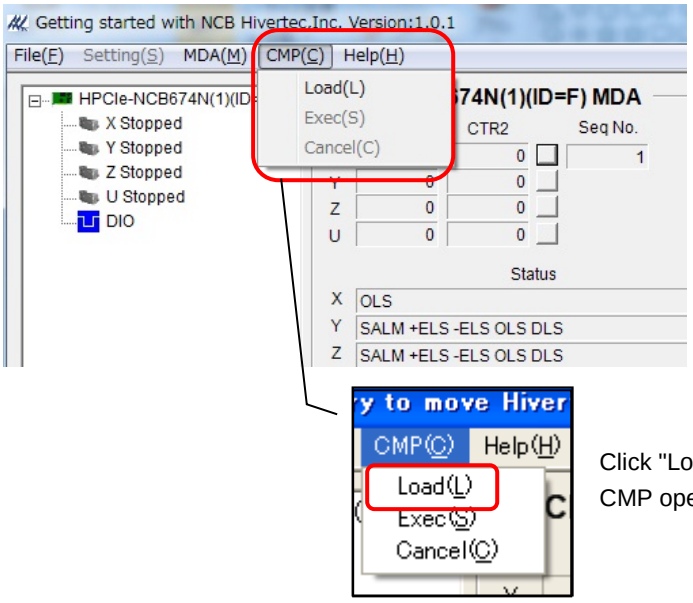


2.4.11 CMP operation

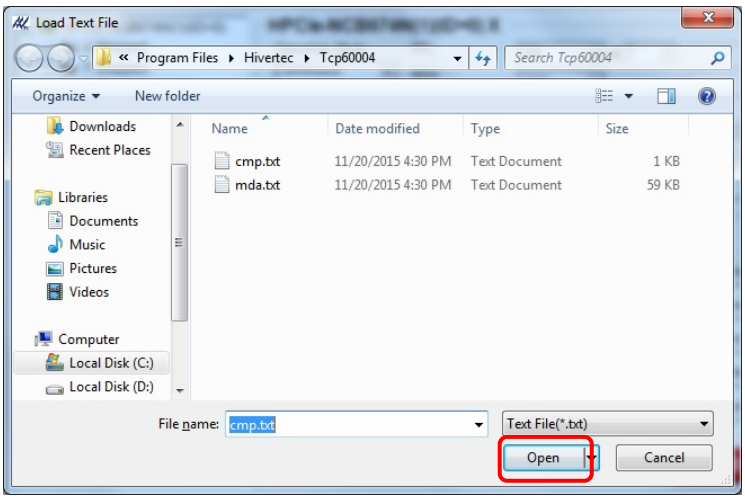
Allows for loading a text file with CMP data to perform CMP operation.

Comparator comparison condition for the relevant axis must be set in the comparator configuration screen in advance.

Specify CMP5 as the comparator to use since CMP operation is possible only with CMP5.



Click "Load(L)" to open the dialog to select the file for the CMP operation.



Select the file and click [Open(O)] to load the file data.

The CMP data is loaded to the board selected in the list box.

CMP operation can be started on successful CMP data load.

Clicking "Exec(S)" starts CMP operation.

Clicking "Cancel(C)" cancels the CMP operation.

(1) CMP data file format

Extension: txt

The first line is always the top initialization block (header).
It is written in hexadecimal notation.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SEL_AXIS (axis selection)								TOF							
				U	Z	Y	X	Fixed to 40h							

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Length															
Total number of blocks (3 to 2002)															

The last line is always the end block (footer).
Its content is written in hexadecimal notation and fixed to 00000080.

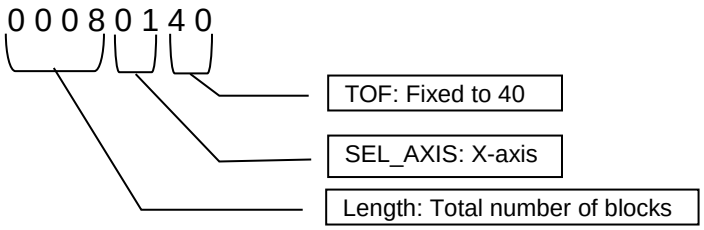
Other lines are comparator comparison data written in decimal notation.

<Example>

```
00080140
1000
3000
4000
6000
7000
9000
00000080
```

<Explanation>

The first line is the top initialization block; it indicates the total number of blocks (total number of lines): 8 and the axis to use: X.

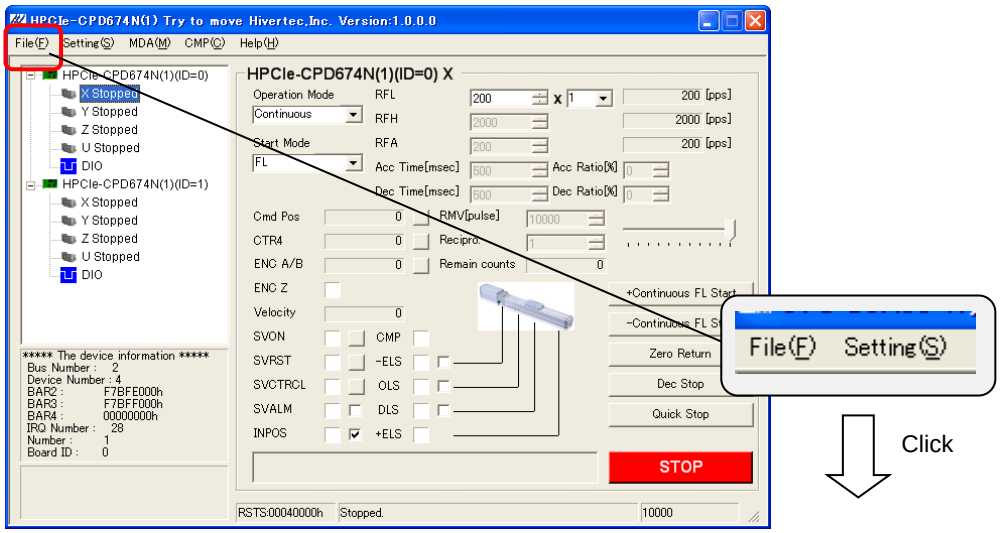


Lines 2 to 7 contain comparator comparison data written in decimal notation.
The eighth line is the last block.

2.4.12 Save to configuration file/Configuration by loading from file

The settings used last can be saved to a file.

Furthermore, settings can be configured by loading a saved file. (Provided the board configurations are the same)

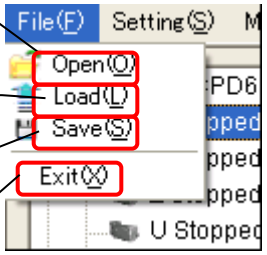


Opens the file to allow for checking the board configuration when the file was saved.

Loads the contents of the configuration file.

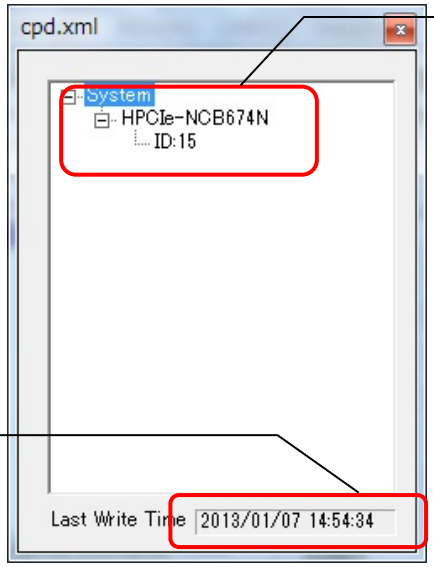
Saves the settings used last to a file.

Exits the program.



(1) File - Open

Allows to check the file contents.

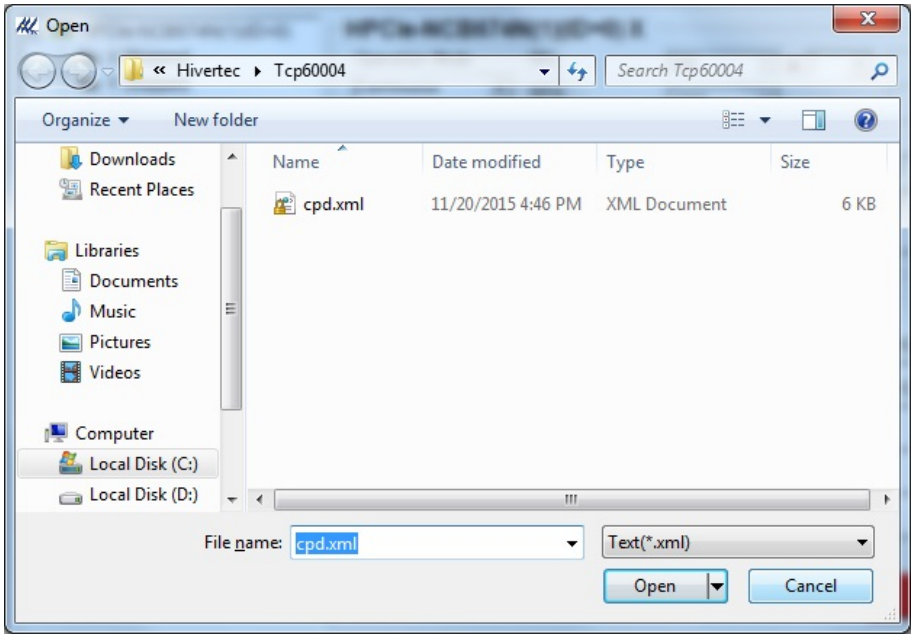


Displays the configuration when the file was saved.
(Board model, Board ID)

Date and time the file was saved.

(2) File - Load

Loads the specified configuration file.



(3) File - Save

Saves settings under the specified file name.

3. Application Creation Guide

3.1 Preparations for Using API Functions

3.1.1 Bundled file names

The "include" folder in the accompanying CD includes all the files for each of the supported development environments.

The names of the files are as shown in the table below.

No.	Type/Folder name	VC_x86	VC_x64	VB	VCS (VC#)
1	Driver function header	hecp670.h	hecp670.h	hecp670.vb	hecp670.cs
2	Driver function import library	hecp670.lib	hecp670.lib	---	---
3	Library function header	cp670l1a.h	cp670l1a.h	---	---
4	Library function source (Compliant with CPD Series)	cp670l1a.cpp	cp670l1a.cpp	cp670l1a.vb	cp670l1a.cs
5	Library function header (NCB series)	NcbLib1a.h	NcbLib1a.h	---	---
6	Library function source (NCB series)	NcbLib1a.cpp	NcbLib1a.cpp	NcbLib1a.vb	NcbLib1a.cs

Table 3.1-1 Bundled file names

There are two driver function import libraries for VC, one for 32-bit applications (VC_x86) and another for 64-bit applications (VC_x64).

When developing applications in VB or VC#, basically set the platform as "x86".

(Please consult when developing 64-bit applications)

Beware of the settings depending on the application to create and your platform.

3.1.2 Using driver functions

(1) To build applications with VC++ 2008 or later

Add driver function import library (hecp670.lib) to the project.

Include the driver function header (hecp670.h).

(2) To build applications with VB 2008 or later

Add driver function header (hecp670.vb) to the project.

(3) To build applications with VC# 2008 or later

Add driver function header (hecp670.cs) to the project.

3.1.3 Using library functions (compliant with CPD Series)

(1) To build applications with VC++ 2008 or later

Add driver function import library (hecp670.lib) and library function source (cp670l1a.cpp) to the project.

Include the library function header (cp670l1a.h). Copy the driver function header (hecp670.h) to the same folder as the library function header. Include the driver function header (hecp670.h) inside the library function header.

(2) To build applications with VB 2008 or later

Add driver function header (hecp670.vb) and library function source (cp670l1a.vb) to the project.

(3) To build applications with VC# 2008 or later

Add driver function header (hecp670.cs) and library function source (cp670l1a.cs) to the project.

3.1.4 Using library functions (NCB Series)

(1) To build applications with VC++ +2008 or later

Add driver function import library (hecp670.lib) and library function source (cp670l1a.cpp) to the project.

Include the library function header (cp670l1a.h). Copy the driver function header (hecp670.h) to the same folder as the library function header. Include the driver function header (hecp670.h) inside the library function header.

Add library function source (NcbLib1a.cpp) to the project.

Add library function header (NcbLib1a.h) to the project.

(2) To build applications with VB.NET 2008 or later

Add driver function header (hecp670.vb) and library function source (cp670l1a.vb) to the project.

Add library function source (NcbLib1a.vb) to the project.

(3) To build applications with VC#.NET 2008 or later

Add driver function header (hecp670.cs) and library function source (cp670l1a.cs) to the project.

Add library function source (NcbLib1a.cs) to the project.

3.2 Accessing a Board

To access a NCB board, the device handle must be acquired first by acquiring the number of devices and device information to open the device.

3.2.1 Device information

To recognize the boards, prepare as many of the HCP670INFO-type structures shown below as the number of connected boards (maximum of 16). This information is called device information. The data structure for VC is shown as a representative example:

```
typedef struct _ HCP670INFO {
    DWORD   dwBusNumber;        /* Bus number */
    DWORD   dwDeviceNumber;     /* Device number */
    DWORD   dwBaseAddress2;     /* Base address 2 */
    DWORD   dwBaseAddress3;     /* Base address 3 */
    DWORD   dwBaseAddress4;     /* Base address 4 */
    DWORD   dwIrqNo;            /* IRQ number */
    DWORD   dwNumber;           /* Control number*/
    DWORD   dwBoardID;          /* Board ID (0 to 15) */
} HCP670INFO, *PHCP670INFO
```

For other development environments, refer to each sample.

3.2.2 Preparations for accessing a board and end processing

(1) When using driver functions only

For details on driver functions, see "4.4 Driver Function Details".

[Preparation]

(1) Acquire the device information of all boards to use

First, acquire the device information of all NCB boards and populate the data structure area (array) for board recognition.

- cp670_GetDeviceCount() • • • Check number of boards
- cp670_GetDeviceInfo()• • • Acquire device information of all boards

(2) Open device for each board

Pass the NCB device information of one NCB board to the open device function. The open device function opens the NCB and returns the device handle to access the board.

When there are two or more NCB boards, repeat the aforementioned processing for each NCB board.

- cp670_OpenDevice() • • • Board open processing

Hereafter, the NCB board can be accessed using this "device handle".

(3) Reset the software and check that the CPU on the board is initialized

To reset the software, write BRD_RST. Thereafter, check whether the initialization of the CPU on the board has finalized.

- cp670_wPortW(h, DP_INIEN, 0); • • • Write "0" to DPINIEN
- cp670_wPortW(h, BRD_RST, 0); • • • Write BRD_RST to reset the software
- cp670_rPortW(h, DP_INIEN, &dpinien); • • • Read DPINIEN; if it is "1", then the initialization is complete.

(4) Initialize each board/axis

After the aforementioned processing, initialize all axes controlled by all of the boards to use. Configure each register by referring to the User's Manual. In this way, each axis can output pulses and perform other operations normally.

- cp670_wPortW() • • • Configure the option ports (registers)
- cp670_wReg • • • Configure the PCL register

[End processing]

(5) Perform "close processing" for opened devices

To exit the application after completing all processing (including servo OFF, pulse stop confirmation and the like), perform "close processing" for the opened devices.

- cp670_CloseDevice() • • • Board close processing

(2) When using library functions only

For details on library functions see section “4.7 Library Function Details (CPD-Series Compliant)” and “4.8 Library Function Details (NCB Series)".

[Preparation]

(1) Acquire the device information of all boards to use

First, acquire the device information of all NCB boards and populate the data structure area (array) for board recognition.

- hcp670_GetDevInfo() • • • Check the number of boards and acquire the device information of all boards

  It is a combination of two driver functions.

(2) Open device for each board

Pass the NCB device information of one NCB board to the open device function. The open device function opens the NCB and returns the device handle to access the board.

Hereafter, the NCB board can be accessed using this "device handle".

In this way, each axis can output pulses and perform other operations normally.

When there are two or more boards, repeat the aforementioned processing for each board.

- hcp670_DevOpen() • • • Open the board, configure how the board will process signals and the operating conditions for each axis

  It is a combination of driver function processing (2) to (4).
  The conditions for initialization are set directly from within this function.
  For details, see section “4.7 Library Function Details (CPD-Series Compliant)” and “4.8 Library Function Details (NCB Series)"..

[End processing]

(3) Perform "close processing" for opened devices

To exit the application after completing all processing (including servo OFF, pulse stop confirmation and the like), perform "close processing" for the opened devices.

- hcp670_DevClose() • • • Board close processing

3.2.3 Control from the software on a PC and control by using NCB mode

Motion control by NCB can be implemented as

(1) Control from the software on a PC
(2) Control by using NCB mode (MDA, CDA, or CMP)
(3) Control by a combination of the two above

For specific operation methods based on (1), refer to “CPD Board Series User’s Manual <Operation>”.
For other operation methods, see "5. NCB Mode (Automatic Program Execution Function: MDA)".

When combining control from the software on a PC and control by using NCB mode, access to PCL requires exclusive control. A lack of exclusive access control may result in unexpected operation.
For details on how to access to PCL via DPRAM, see "8. Accessing PCL during MDA, CDA, or CMP Execution".

3.3 Sample Programs (Compliant with CPD Series)

These sample programs are for the purpose of explaining how to use CPD-series compliant library functions to control boards from the software on a PC.

For details on the processing inside Device Open/Close Sample, X Continuous Feed Sample, X Homing Sample, X Positioning Sample, X-Y Linear Interpolation Sample, and X-Y Circular Interpolation Sample programs, refer to "CPD Board Series User's Manual <Operation>".

Regarding the General Purpose DIO sample, also see the explanation related to generic input/output in "9.3 Option Ports (Registers)" of this manual.

Regarding the MDA Sample and CMP Sample, also see "9.4 DPRAM" and "5. NCB Mode (Automatic Program Execution Function: MDA)", and "7. NCB Mode (Automatic Comparator Execution Function: CMP)" in this manual.

3.3.1 Sample program file names

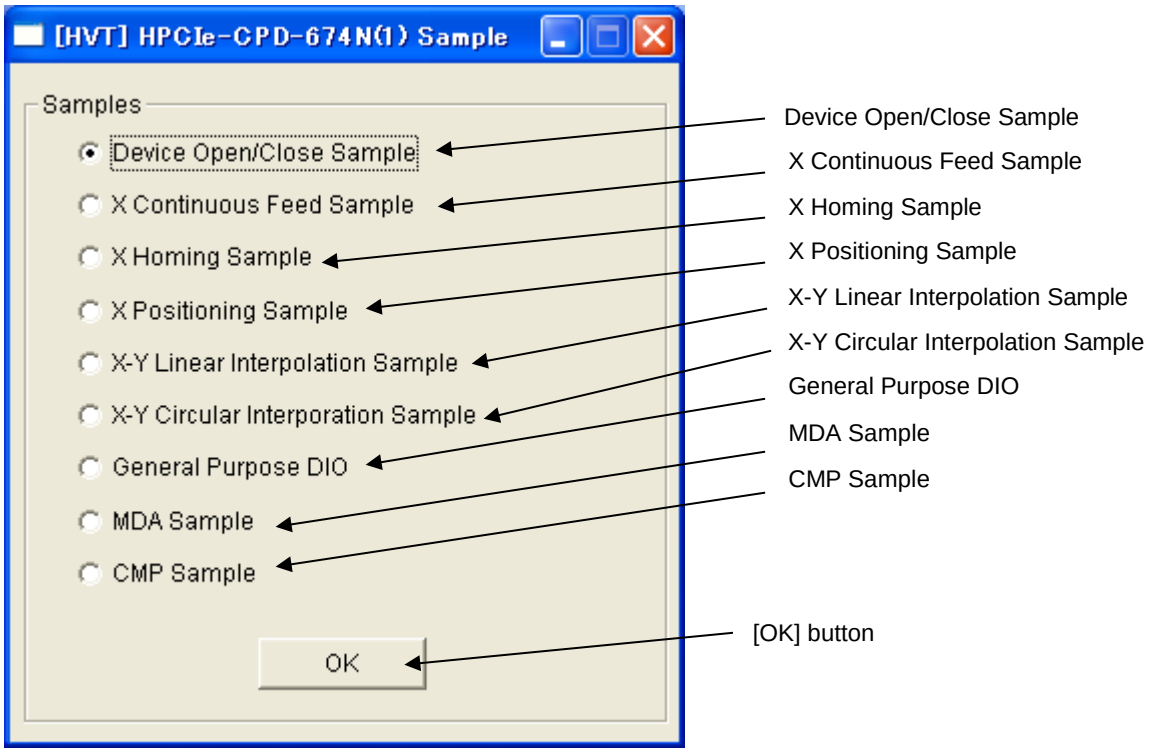
No.	Type	VC	VB	VC#	Remarks
1	Driver function header	hecp670.h	hecp670.vb	hecp670.cs	
2	Driver function import library	hecp670.lib	---	---	For 32 bits
3	Library function header	cp670l1a.h	---	---	
4	Library function source	cp670l1a.cpp	cp670l1a.vb	cp670l1a.cs	
5	Sample common header	common.h	---	---	
6	Sample common function source	common.c	common.vb	common.cs	
7	Main program source	secp6700.c	secp6703.vb	secp6704.cs	
8	Device Open/Close Sample	s1_dev.c	s1_dev.vb	s1_dev.cs	
9	X Continuous Feed Sample	s2_cnt.c	s2_cnt.vb	s2_cnt.cs	
10	X Homing Sample	s3_org.c	s3_org.vb	s3_org.cs	
11	X Positioning Sample	s4_pos.c	s4_pos.vb	s4_pos.cs	
12	X-Y Linear Interpolation Sample	s5_lin.c	s5_lin.vb	s5_lin.cs	
13	X-Y Circular Interpolation Sample	s6_cir.c	s6_cir.vb	s6_cir.cs	
14	General Purpose DIO	s7_dio.c	s7_dio.vb	s7_dio.cs	
15	MDA Sample	s8_mda.c	s8_mda.vb	s8_mda.cs	
16	CMP Sample	s9_cmp.c	s9_cmp.vb	s9_cmp.cs	
17	Project name	secp6700	secp6703	secp6704	
18	Executable file name	secp6700.exe	secp6703.exe	secp6704.exe	

Table 3.3-1 Sample program file names

3.3.2 Starting the sample program

Starting the sample program displays the screen below.

Use this screen to select the sample program to run: "Device Open/Close Sample", "X Continuous Feed Sample", "X Homing Sample", "X Positioning Sample",
"X-Y Linear Interpolation Sample", "X-Y Circular Interpolation Sample", or "General Purpose DIO".

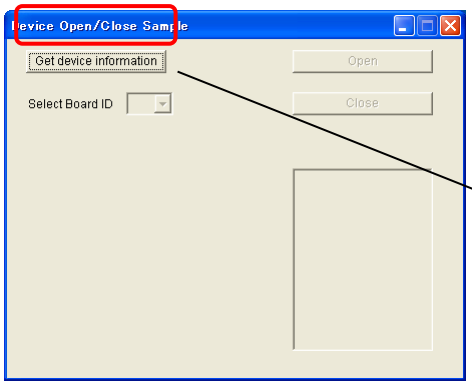


Selecting the radio button and then clicking [OK] starts the sample program.

However, if a board is not inserted, the error message below displays, and the sample program does not start.

3.3.3 Device Open/Close Sample

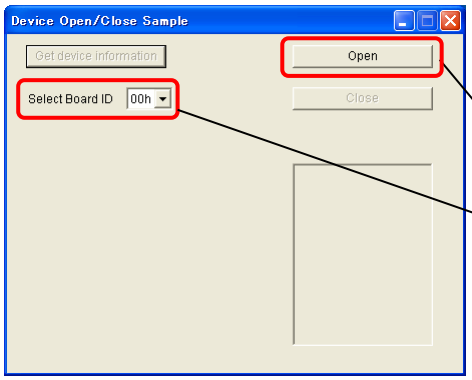
(1) Start screen



Clicking [Get device information] acquires the device information of CPDs connected to the PC to set each board ID in the [Select Board ID] combo box.

(1) [Get device information] button

(2) Select a board ID

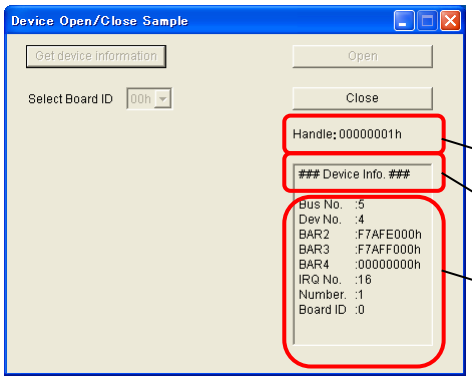


Select a BoardID from the [Select Board ID] combo box and click [Open] to open the device.

(3) [Open] button

(2) [Select Board ID] combo box

(3) Open the device



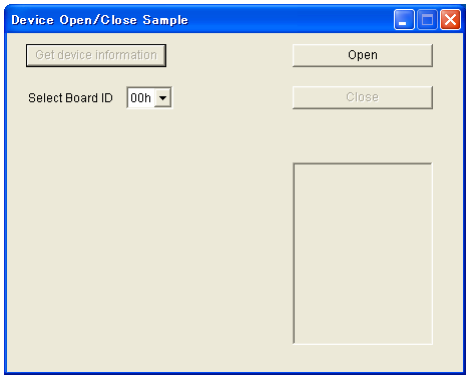
The device information and device handle display when the device is successfully opened.
Clicking [Close] closes the device.

(4) [Close] button

Device handle

Device information

(4) Screen after the device is closed

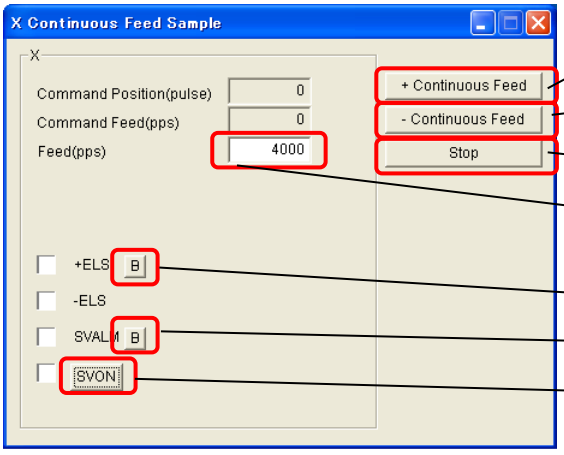


3.3.4 X Continuous Feed Sample

(1) Specifications

	Item	Content
1	Control axis	X-axis
2	Continuous feed in the positive/negative direction	Continuous feed with acceleration/deceleration
3	Stop	Deceleration stop
4	Base speed	200[pps] fixed
5	Operation speed	1 to 65535pps, can be set in units of 1pps (default value: 4000pps)
6	Acceleration time	When accelerating from 200pps to 2000pps, fixed to approximately 500ms
7	SVON	ON/OFF possible (default value: OFF)
8	SVON status indication	ON: green, OFF: white
9	ELS/SVALM input polarity	Normal open/Normal close switchable (default value: normal close)
10	Command position indication	In units of pulses (default value: 0)
11	Command speed indication	In units of pps
12	±ELS/SVALM input status indication	ON: red, OFF: white

(2) Start screen



[+Continuous Feed] button

[-Continuous Feed] button

[Stop] button

[Feed] edit box
(Setting range: 1 to 65535pps, in units of 1pps)

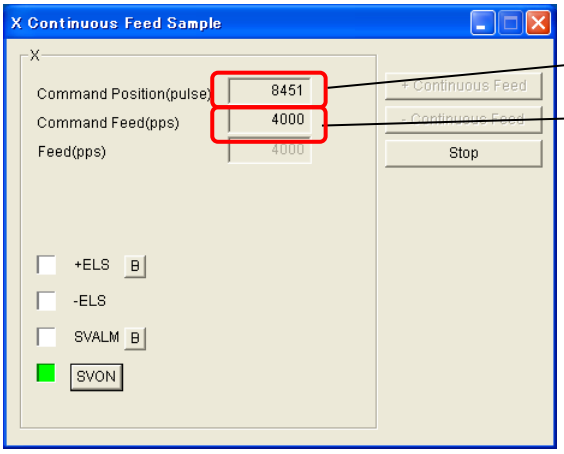
ELS input polarity change button

SVALM input polarity change button

[SVON] button

Clicking [SVON] button turns SVON ON/OFF.
SVON ON: green (see figure below), OFF: white

(3) During operation



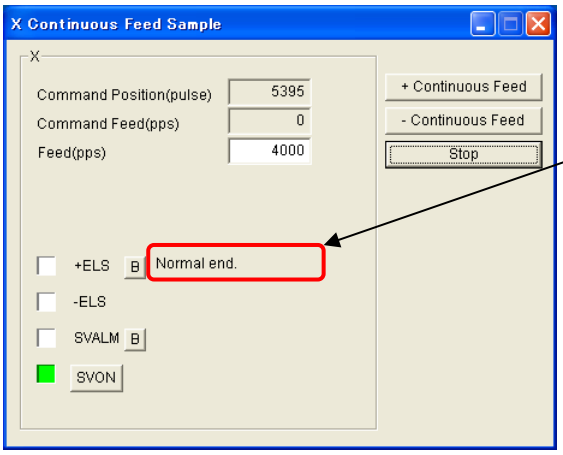
Command position indication (in units of pulses)

Command speed indication (in units of pps)

Specify the feeding speed in the [Feed] edit box.
Clicking [+Continuous Feed] feeds continuously in the positive direction.
Clicking [-Continuous Feed] feeds continuously in the negative direction.
Continuous feed is an operation with acceleration/deceleration.

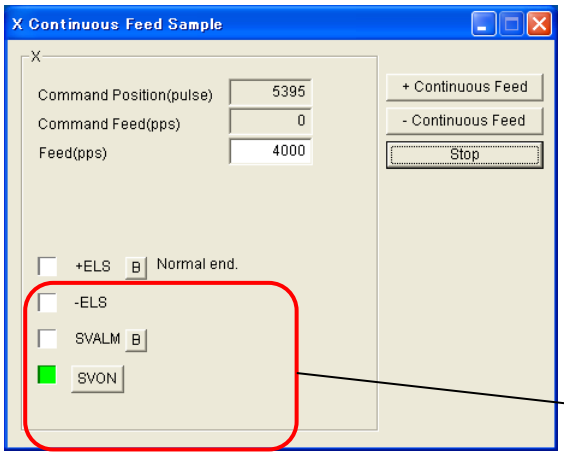
During operation, [+Continuous Feed] button, [-Continuous Feed] button, and the [Feed] edit box are disabled.

(4) On [Stop] button click

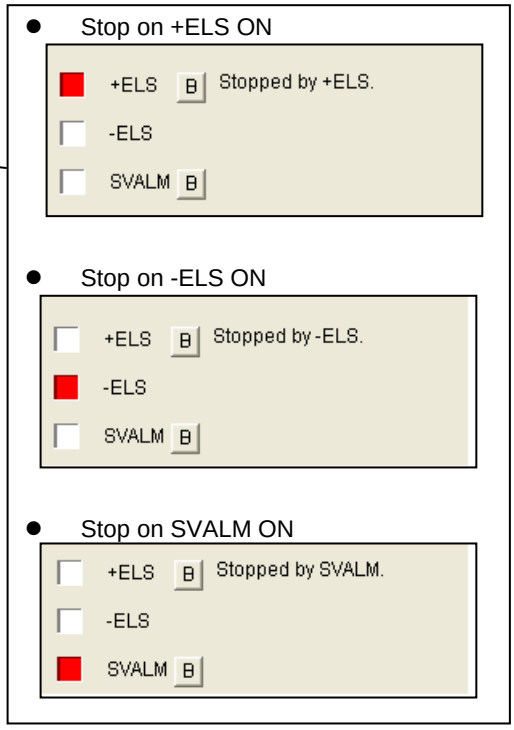


Clicking [Stop] decelerates and stops the axis. "Normal end" is displayed when the axis stops normally.

(5) Error stop and status indications



If +ELS ON is input when operating in the positive direction, -ELS ON is input when operating in the negative direction, or SVALM is input during the operation, the axis stops quickly (error stop). The cause of the error stop is indicated as shown below.

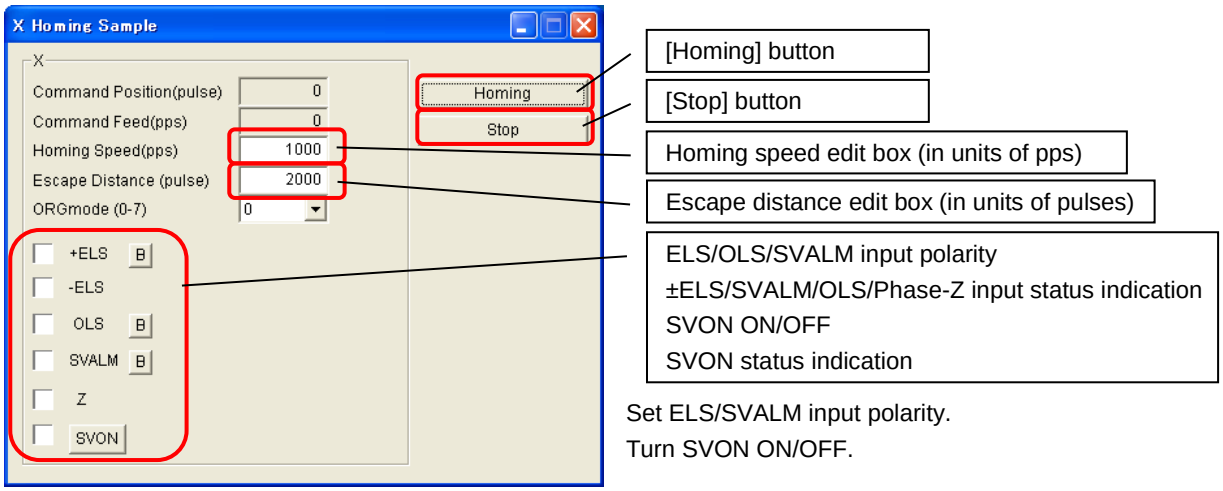


3.3.5 X Homing Sample

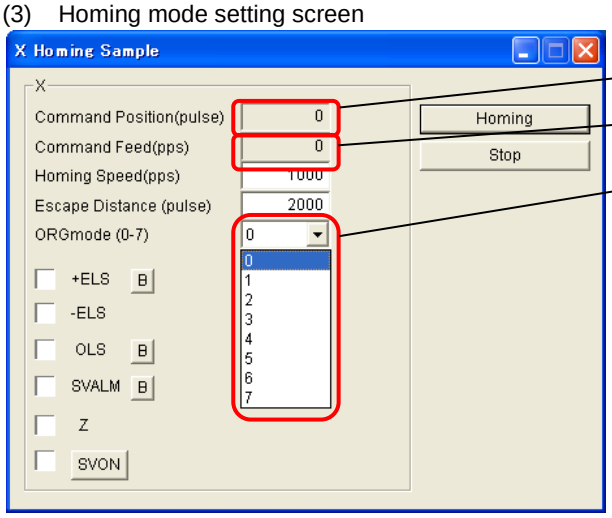
(1) Specifications

	Item	Content
1	Control axis	X-axis
2	Homing	Origin search with acceleration/deceleration. No error counter clear output on completion. For details on the origin search operation, refer to "CPD Series User's Manual <Operation>".
3	Homing mode	0 to 7, selectable (default value: 0) For details on each mode and timing of clearing the counters, refer to "CPD Series User's Manual <Operation>".
4	Stop	Deceleration stop
5	Base speed	200[pps] fixed
6	Auxiliary speed	200[pps] fixed
7	Operation speed	1 to 65535pps, can be set in units of 1pps (default value: 1000pps)
8	Acceleration time	When accelerating from 200pps to 2000pps, fixed to approximately 500ms
9	Amount of origin escape	1 to 99999999pulse (default value: 2000pulse)
10	SVON	ON/OFF possible (default value: OFF)
11	SVON status indication	ON: green, OFF: white
12	ELS/OLS/SVALM input polarity	Normal open/Normal close switchable (default value: normal close)
13	Command position indication	In units of pulses (default value: 0)
14	Command speed indication	In units of pps
15	±ELS/SVALM input status indication	ON: red, OFF: white
16	OLS/Phase-Z input status indication	ON: green, OFF: white

(2) Start screen



(3) Homing mode setting screen



Command position indication (in units of pulses)

Command speed indication (in units of pps)

[ORGmode] combo box

Specify the operation speed in [Homing Speed] edit box.

Specify the amount of origin escape in [Escape Distance] edit box.

Select the homing mode in [ORGmode] combo box.

Clicking [Homing] starts the origin search.

Clicking [Stop] decelerates and stops the axis.

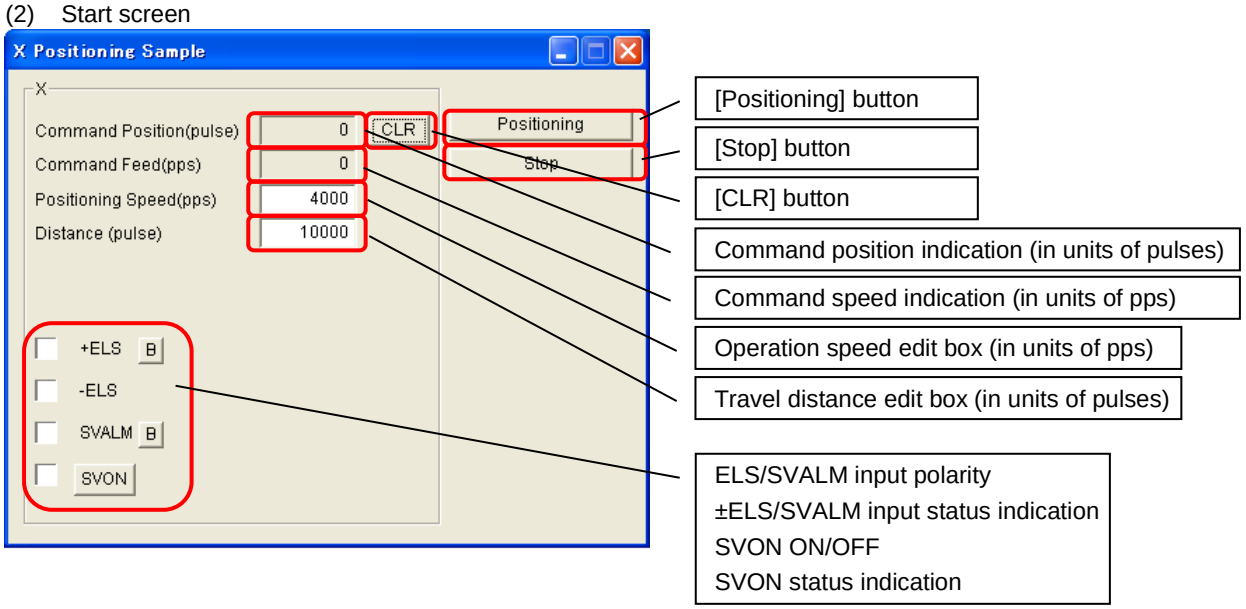
During operation, [Homing] button and each of the edit boxes are disabled.

3.3.6 X Positioning Sample

(1) Specifications

	Item	Content
1	Control axis	X-axis
2	Positioning	Relative positioning with acceleration/deceleration
3	Stop	Deceleration stop
4	Base speed	200[pps] fixed
5	Operation speed	1 to 65535pps, can be set in units of 1pps (default value: 4000pps)
6	Acceleration time	When accelerating from 200pps to 2000pps, fixed to approximately 500ms
7	Travel distance	-9999999 to +99999999pulse (default value: 10000pulse)
8	Counter value reset	Resets command position to "0"
9	SVON	ON/OFF possible (default value: OFF)
10	SVON status indication	ON: green, OFF: white
11	ELS/SVALM input polarity	Normal open/Normal close switchable (default value: normal close)
12	Command position indication	In units of pulses (default value: 0)
13	Command speed indication	In units of pps
14	±ELS/SVALM input status indication	ON: red, OFF: white

(2) Start screen



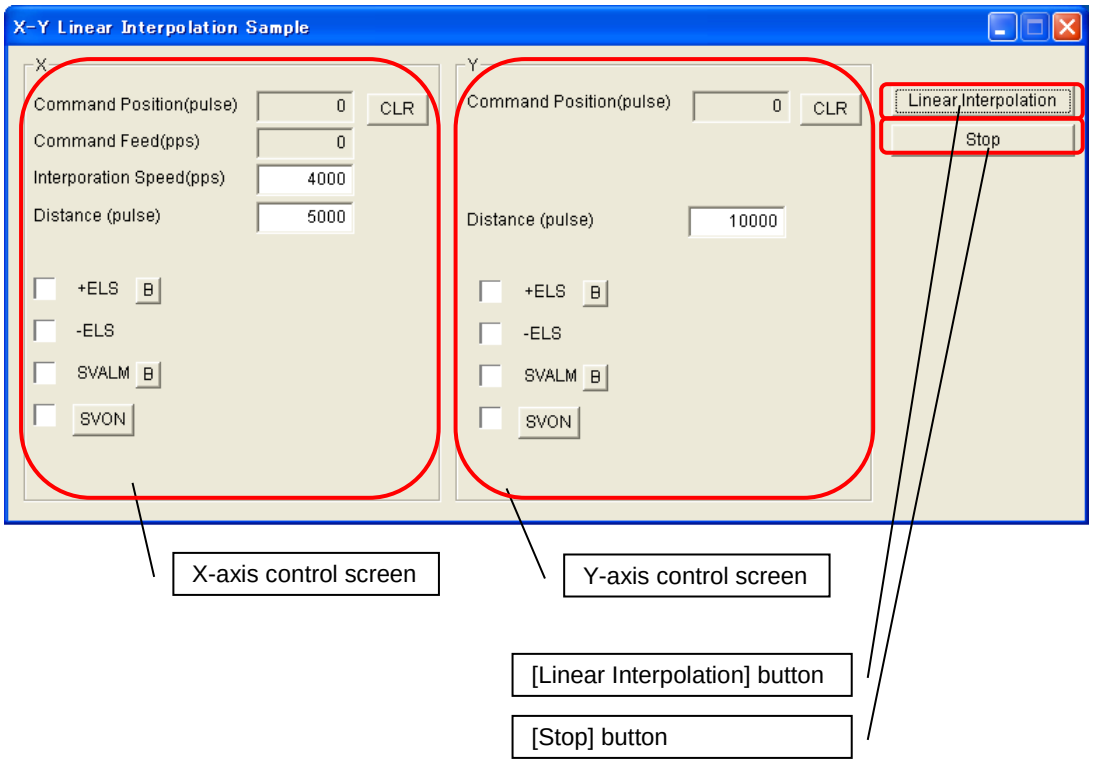
Set ELS/SVALM input polarity and turn SVON ON/OFF.
Specify the positioning speed in the [Positioning Speed] edit box.
Specify a signed travel distance in [Distance] edit box.
Clicking [Positioning] starts relative positioning.
Clicking [Stop] decelerates and stops the axis.
During operation, [Positioning] button and each of the edit boxes are disabled.
Clicking [CLR] resets the command position to "0".

3.3.7 X-Y Linear Interpolation Sample

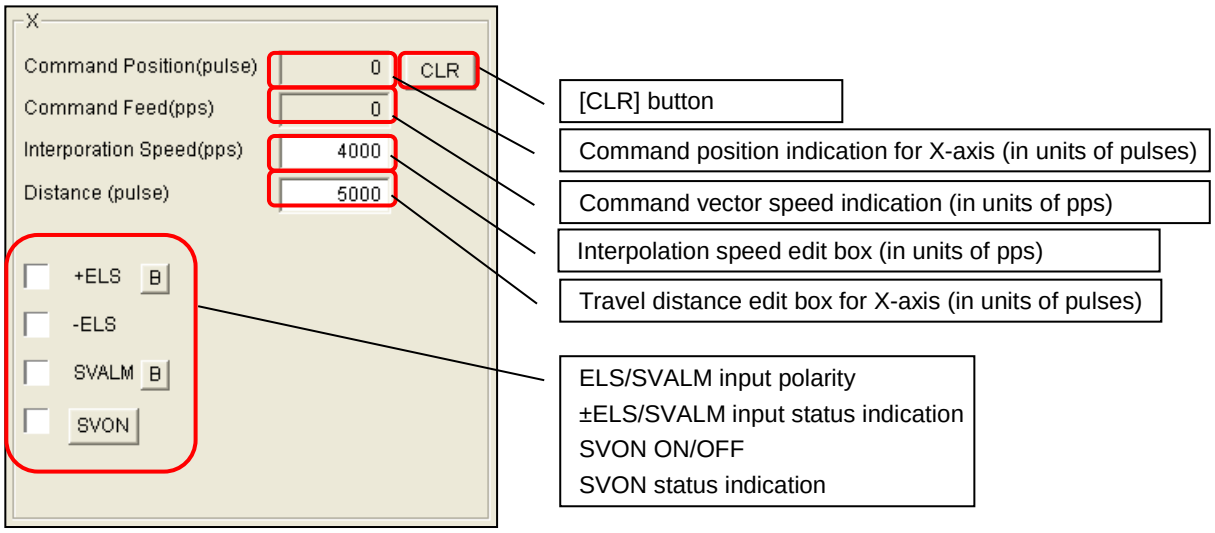
(1) Specifications

	Item	Content
1	Control axis	X and Y axes
2	Linear Interpolation	Linear Interpolation with acceleration/deceleration Constant vector speed control ON
3	Stop	Deceleration stop
4	Base speed	200[pps] fixed
5	Interpolation speed	1 to 65535pps, can be set in units of 1pps (default value: 4000pps)
6	Acceleration time	When accelerating from 200pps to 2000pps, fixed to approximately 500ms
7	X and Y axes travel distance	-9999999 to +99999999pulse (default value: 5000pulse for X, 10000pulse for Y)
8	SVON	ON/OFF possible (default value: OFF)
9	SVON status indication	ON: green, OFF: white
10	ELS/SVALM input polarity	Normal open/Normal close switchable (default value: normal close)
11	Command position indication	In units of pulses (default value: 0)
12	Command speed indication	In units of pps
13	±ELS/SVALM input status indication	ON: red, OFF: white

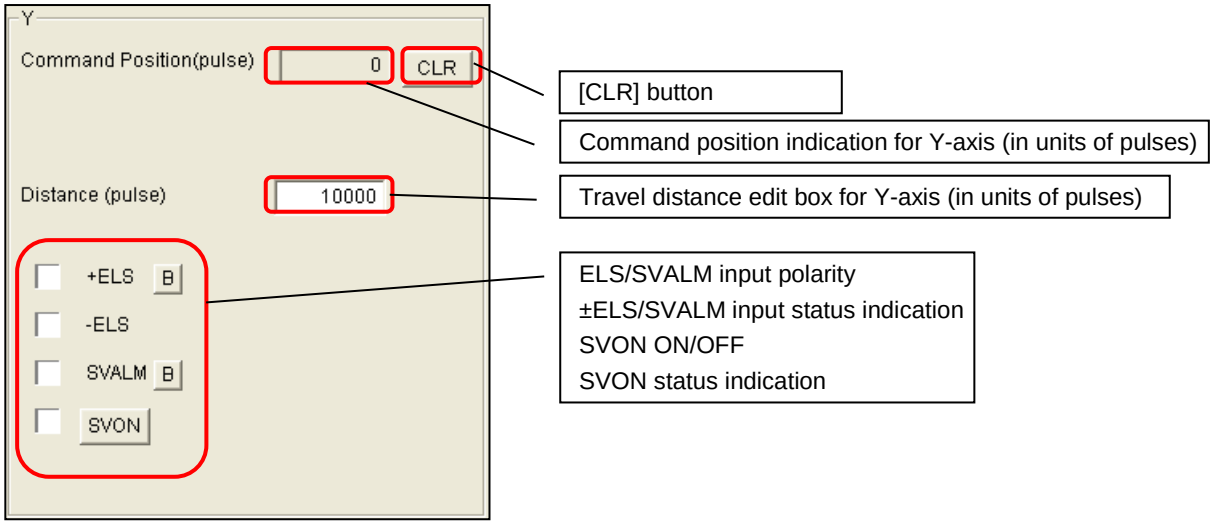
(2) Start screen



(3) X-axis control screen



(4) Y-axis control screen



(5) Operation procedure

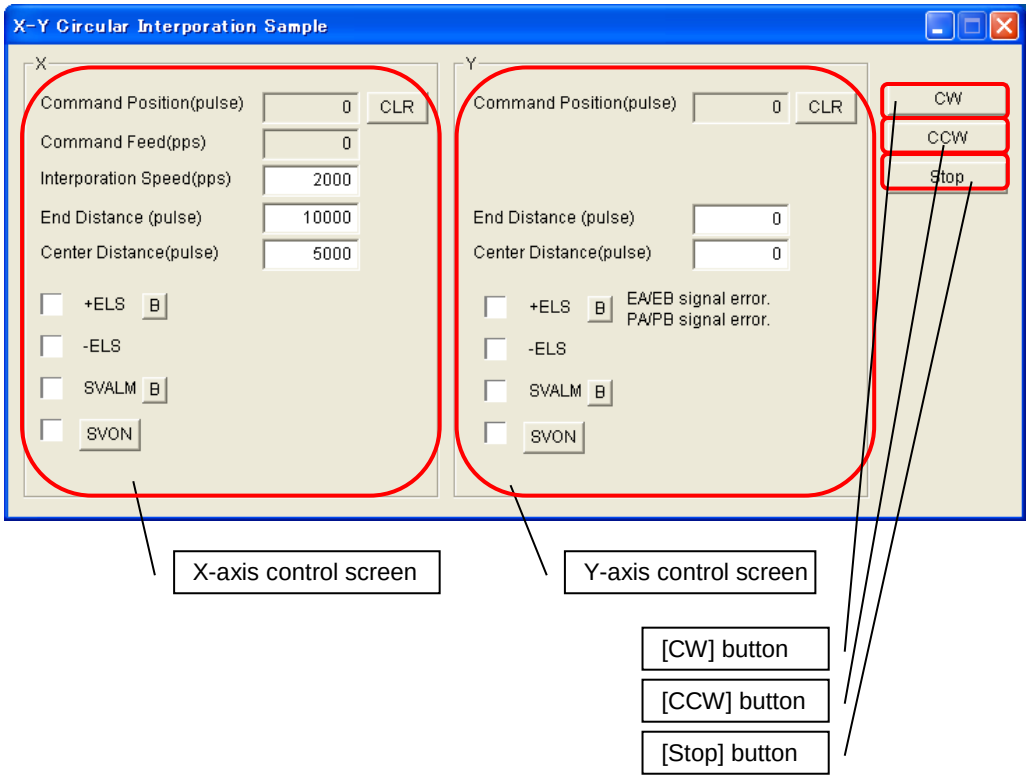
Set ELS/OLS/SVALM input polarity and turn SVON ON/OFF for axes X and Y.
Specify operation speed in [Interpolation Speed] edit box.
Specify signed travel distance in [Distance] edit box for axes X and Y.
Clicking [Linear Interpolation] starts linear interpolation.
Clicking [Stop] stops the linear interpolation in progress. The axes will decelerate and stop.
During operation, [Linear interpolation] button and each of the edit boxes are disabled.
Clicking [CLR] resets the command position of each axis to "0".

3.3.8 X-Y Circular Interpolation Sample

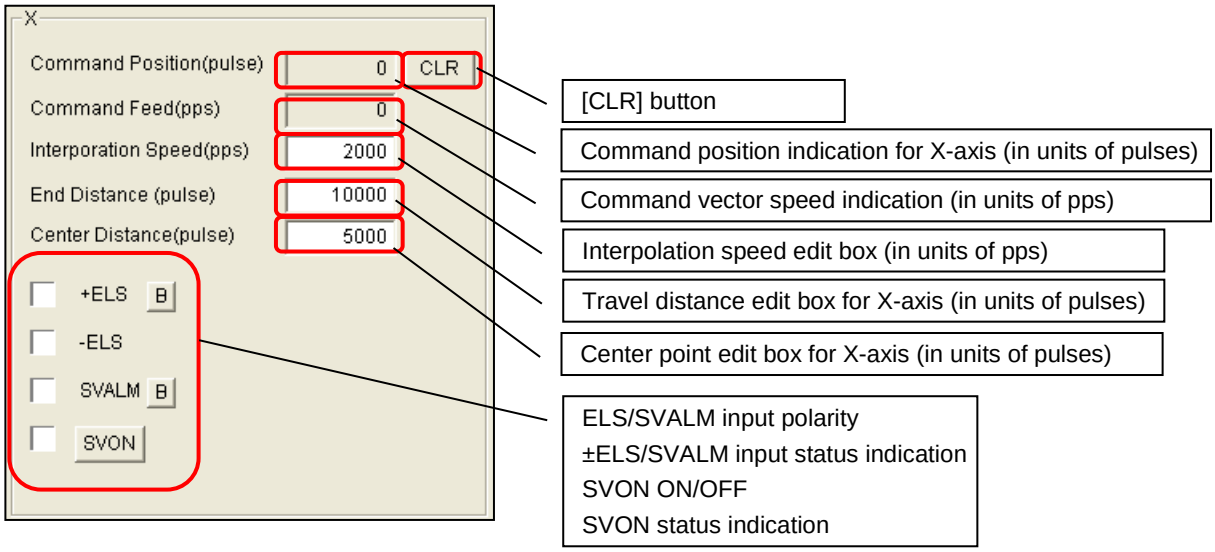
(1) Specifications

	Item	Content
1	Control axis	X and Y axes
2	Circular interpolation	FH constant-speed CW circular interpolation and FH constant-speed CCW circular interpolation Constant vector speed control ON
3	Stop	Quick stop
4	Interpolation speed	1 to 65535pps, can be set in units of 1pps (default value: 2000pps)
5	X and Y axes travel distance	-9999999 to +99999999pulse (default value: 10000pulse for X, 0pulse for Y)
6	XY center point	-9999999 to +99999999pulse (default value: 5000pulse for X, 0pulse for Y)
7	SVON	ON/OFF possible (default value: OFF)
8	SVON status indication	ON: green, OFF: white
9	ELS/SVALM input polarity	Normal open/Normal close switchable (default value: normal close)
10	Command position indication	In units of pulses (default value: 0)
11	Command speed indication	In units of pps
12	±ELS/SVALM input status indication	ON: red, OFF: white

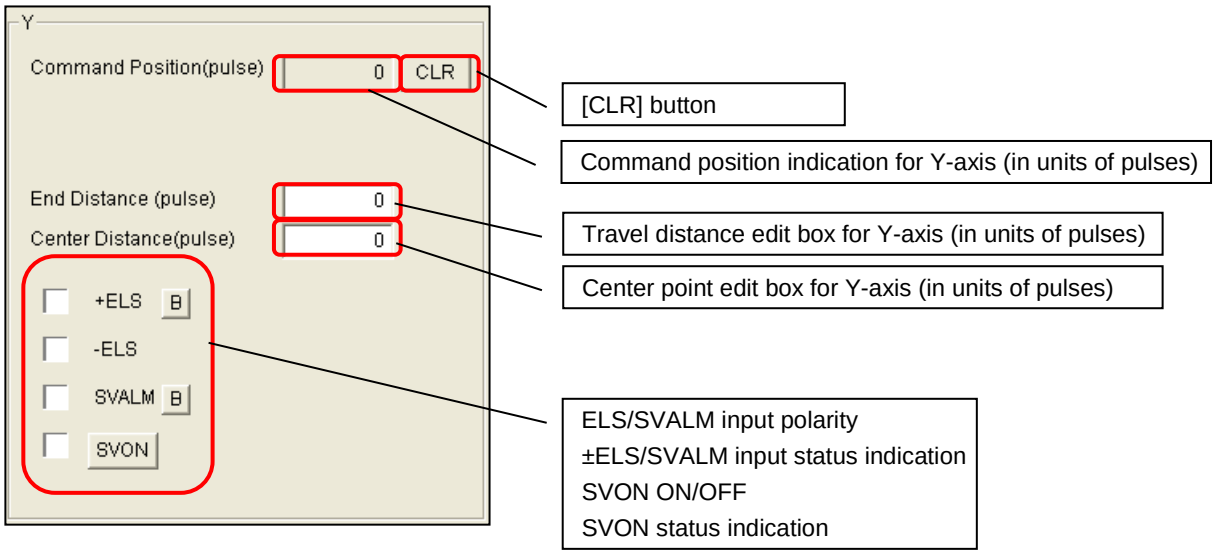
(2) Start screen



(3) X-axis control screen



(4) Y-axis control screen

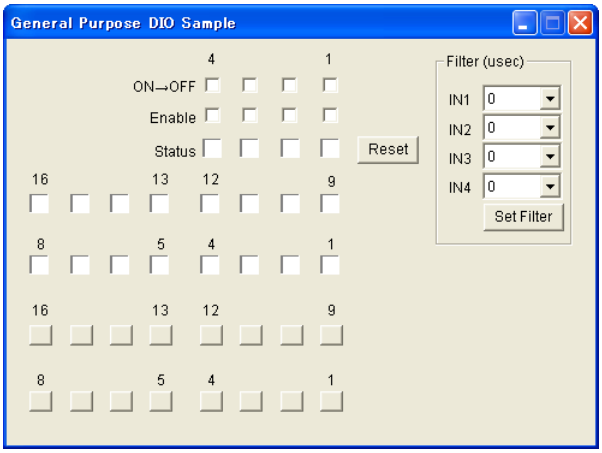


(5) Operation procedure

Set ELS/OLS/SVALM input polarity and turn SVON ON/OFF for axes X and Y.
Specify operation speed in [Interpolation Speed] edit box.
Specify signed travel distance in [End Distance] edit box for axes X and Y.
Specify signed travel distance in [Center Distance] edit box for axes X and Y.
Clicking [CW] starts CW circular interpolation.
Clicking [CCW] starts CCW circular interpolation.
Clicking [Stop] stops the linear interpolation in progress. The axes will decelerate and stop.
During operation, [CW] button, [CCW] button, and each of the edit boxes are disabled.
Clicking [CLR] resets the command position of each axis to "0".

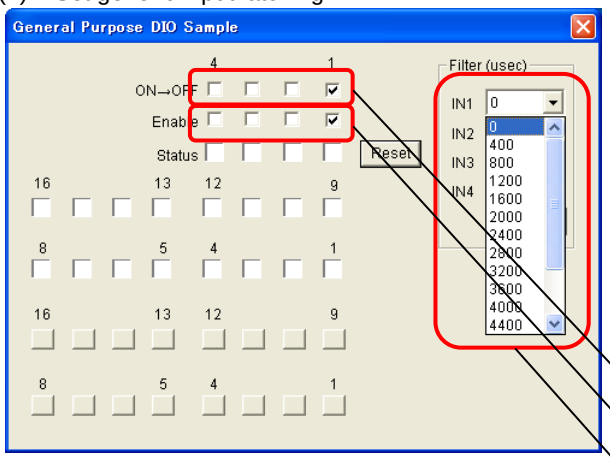
3.3.9 General Purpose DIO

(1) Specifications



Starting "General Purpose DIO" displays the screen shown on the left.
Initially, generic input latching is disabled, no filter is defined, and all generic outputs are turned OFF.

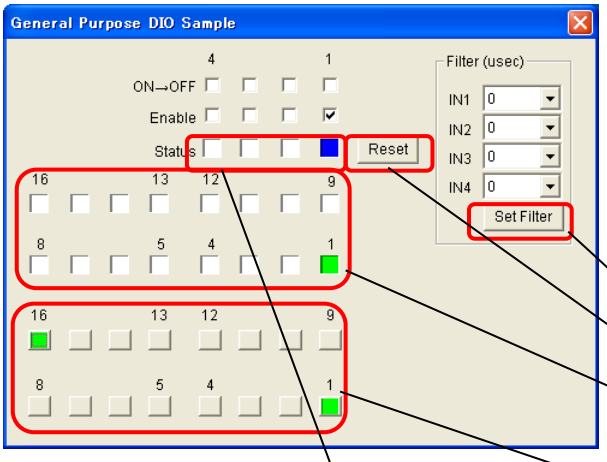
(2) Set generic input latching



[ON→OFF] (Generic input latch polarity check box)
Checked
Not checked
[Enable] (Generic input latch enable check box)
Checked
Not checked
[Filter] (Generic input latch filter combo box)
Setting range: 0 to 6000µs [in units of 400µs]
(Default value: 0)

Generic input latch polarity check box
Generic input latch enable check box
Generic input latch filter combo box

(3) Screen when generic output, generic input, and generic input latching are configured



Clicking [Set Filter] sets the input filter selected in [Filter] combo box.
Clicking [Reset] resets the generic input latch status.

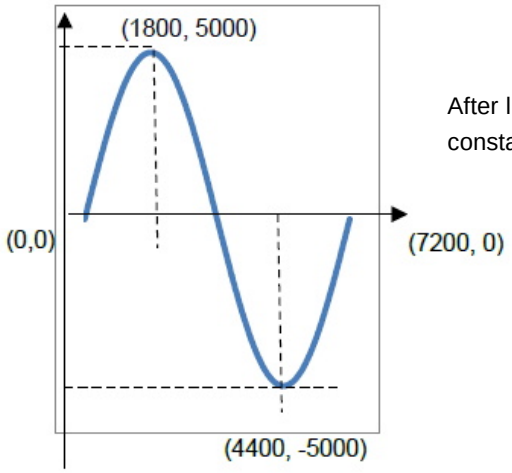
[Set Filter] button
[Reset] button
Generic input statuses
Green: ON, White: OFF
Generic output statuses and generic output buttons
Clicking a button turns the corresponding generic output ON/OFF.
Generic input latch statuses
Blue: Latched input

3.3.10 MDA Sample

(1) Specifications

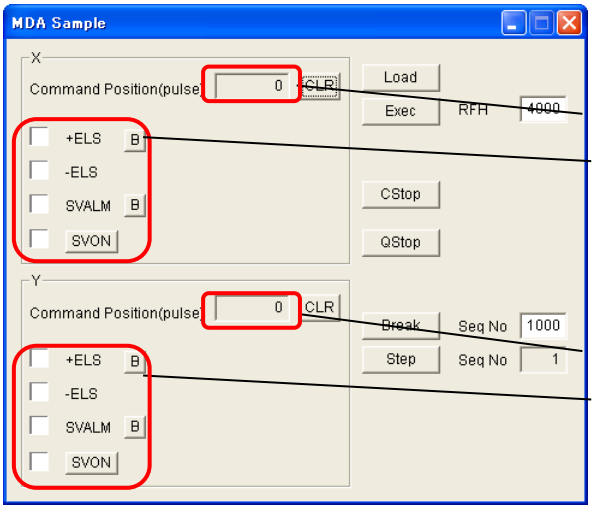
	Item	Content
1	Control axis	X and Y axes
2	Stop	Deceleration stop
3	Operation speed	1 to 65535pps, can be set in units of 1pps (default value: 4000pps)
4	Counter value reset	Resets command position to "0"
5	SVON	ON/OFF possible (default value: OFF)
6	SVON status indication	ON: green, OFF: white
7	ELS/SVALM input polarity	Normal open/Normal close switchable (default value: normal close)
8	Command position indication	In units of pulses (default value: 0)
9	Command speed indication	In units of pps
10	±ELS/SVALM input status indication	ON: red, OFF: white
11	MDA execution function	Performs operation in accordance with the MDA data fixed to the sample program.

(2) MDA execution locus



After linearly interpolating the locus shown in the figure on the left at constant-speed: (0,0) → (7200,0), moves to (0,0) to complete.

(3) Start screen



Starting MDA Sample displays the screen shown on the left.

Command position indication for X-axis (in units of pulses)

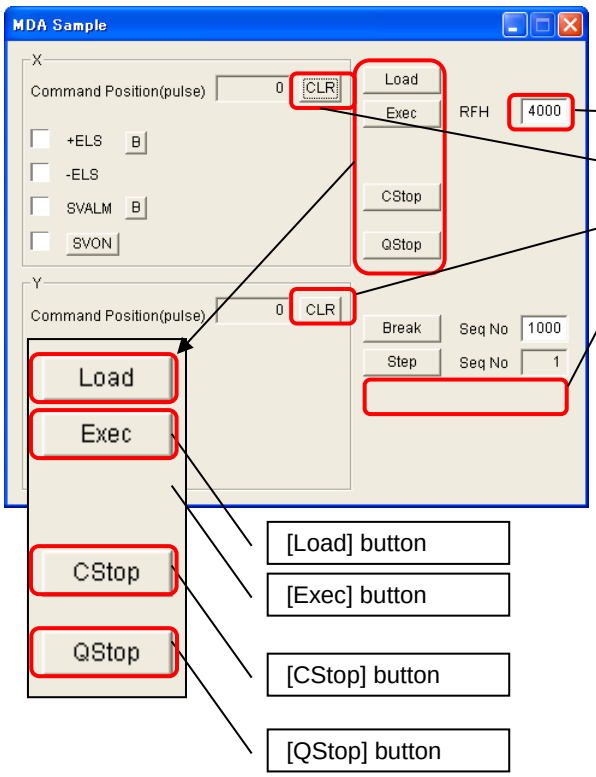
[X-axis]
ELS/SVALM input polarity
±ELS/SVALM input status indication
SVON ON/OFF
SVON status indication

Command position indication for Y-axis (in units of pulses)

[Y-axis]
ELS/SVALM input polarity
±ELS/SVALM input status indication
SVON ON/OFF
SVON status indication

※ Can not be operated during MDA execution.

(4) Buttons for executing MDA



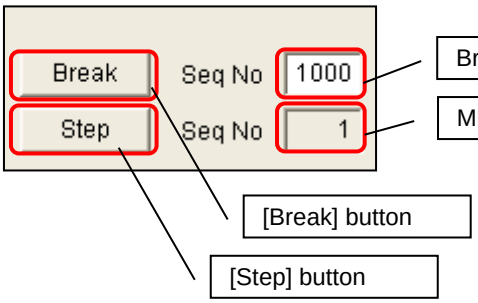
Operation speed edit box (in units of pps)

[CLR] button (X-axis)

[CLR] button (Y-axis)

Message display area

Specify operation speed in [RFH] edit box.
During DPBUFBUSY, the operation speed cannot be changed.
Clicking [Load] loads the MDA data. When successfully loaded,
"MDA LOADED" displays in the message display area.
Clicking [Exec] starts the MDA execution. When successfully started,
"MDA STARTED" displays in the message display area.
When the MDA execution completes,
"MDA END" displays in the message display area.
Clicking [CStop] ends the cycle, clicking [Exec] again resumes the MDA execution.
Clicking [QStop] during operation quickly stops MDA, and resets the sequence number to "1".
Clicking [QStop] when axes are stopped cancels the MDA data, and resets the sequence number to "1".



Break execution line number edit box

MDA execution completed line number

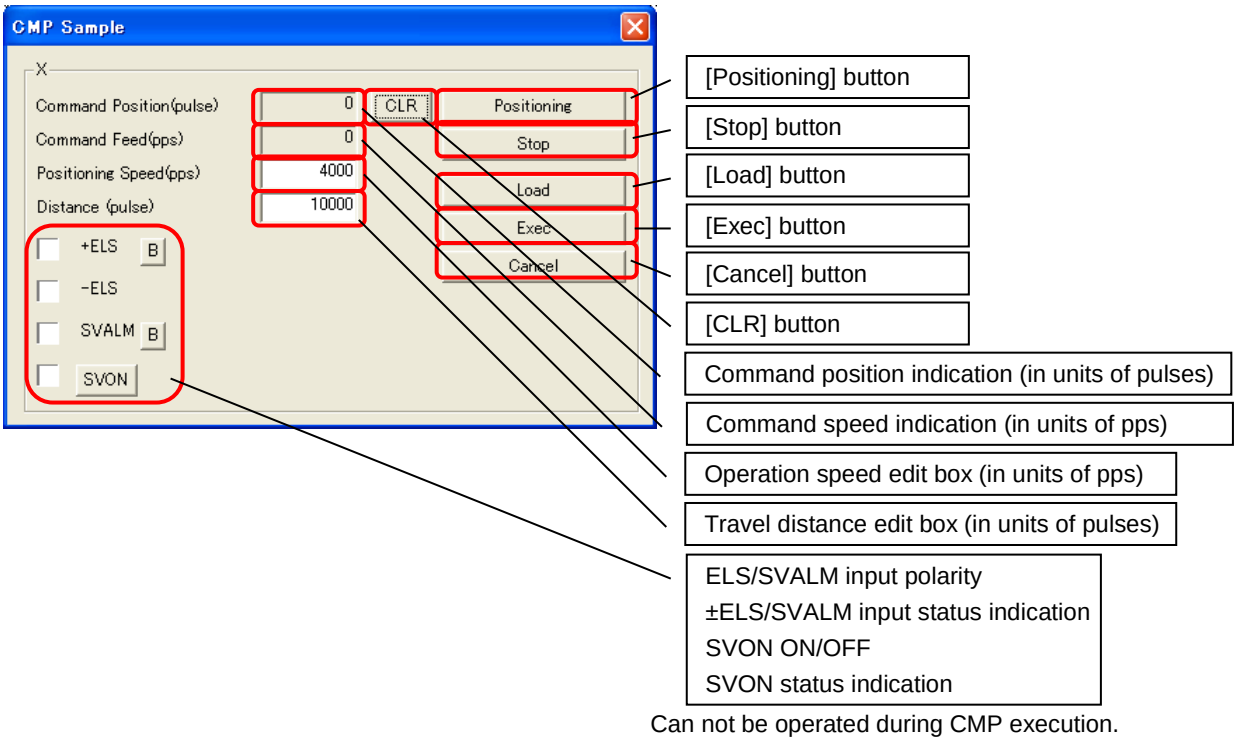
Clicking [Break] stops the MDA execution at the line specified in the break execution line edit box.
Clicking [Step] executes MDA step-by-step.

3.3.11 CMP Sample

(1) Specifications

	Item	Content
1	Control axis	X-axis
2	Positioning	Relative positioning with acceleration/deceleration
3	Stop	Deceleration stop
4	Base speed	200[pps] fixed
5	Operation speed	1 to 65535pps, can be set in units of 1pps (default value: 4000pps)
6	Acceleration time	When accelerating from 200pps to 4000pps, fixed to approximately 50ms
7	Travel distance	-9999999 to +99999999pulse (default value: 10000pulse)
8	Counter value reset	Resets command position to "0"
9	SVON	ON/OFF possible (default value: OFF)
10	SVON status indication	ON: green, OFF: white
11	ELS/SVALM input polarity	Normal open/Normal close switchable (default value: normal close)
12	Command position indication	In units of pulses (default value: 0)
13	Command speed indication	In units of pps
14	±ELS/SVALM input status indication	ON: red, OFF: white
15	CMP execution function	Compares the command position with CMP comparison data (1228 points) to externally output the comparison result

(2) Start screen



Can not be operated during CMP execution.

Starting CMP Sample displays the screen shown on the left above.
Clicking [Load] loads the CMP data.
Clicking [Exec] starts the CMP execution.
Clicking [Cancel] cancels the CMP data.

3.4 Sample Programs (NCB Series)

These sample programs are bundled for the purpose of explaining MDA and CDA operation by using library functions (NCB series) when controlling the board(s) from the software on a PC.
For details on the used library functions, see "4.8 Library Function Details (NCB Series)" and for details on MDA and CDA operation, "9.4 DPRAM", "5. NCB Mode (Automatic Program Execution Function: MDA)", and "6. NCB Mode (Automatic Program Execution Function: CDA)" in this manual.

3.4.1 Development environment and the like

Provided sample programs are created in VC2008, VB2008, VC#2008, and console application.

3.4.2 Sample program file names

No.	Type	VC	VB	VC#	Remarks
1	Driver function header	hecp670.h	hecp670.vb	hecp670.cs	
2	Driver function import library	hecp670.lib	---	---	For 32 bits
3	Library function (CPD) header	cp670l1a.h	---	---	
4	Library function (CPD) source	cp670l1a.cpp	cp670l1a.vb	cp670l1a.cs	
5	Library function (NCB) header	NcbLib1a.h	---	---	
6	Library function (NCB) source	NcbLib1a.cpp	NcbLib1a.vb	NcbLib1a.cs	
7	Main program source	secp6700.c	secp6703.vb	secp6704.cs	
8	Project name	senc6700	senc6703	senc6704	
9	Executable file name	senc6700.exe	senc6703.exe	senc6704.exe	

3.4.3 Default values

The initial values set by the device open function hcp670_DevOpen() are the base.
In the sample programs, RFH=5000 and servo ON for X and Y axes are set by cpd_init().
Replace the initial values for each axis as needed depending on your environment of use.

3.4.4 Initialization and end processing

ncb670_StartEvent is called on axis initialization completion, and ncb670_StopEvent is called before executing hcp670_DevClose in the end processing.

3.4.5 Sample MDA and CDA operation

All MDA data for the following operations is relative position data (unit: pulses):
Positioning and linear interpolation (X-axis travel distance and Y-axis travel distance), circular interpolation (X-axis travel distance, Y-axis travel distance, X-axis center point, and Y-axis center point)
XY relative positioning(10000,0), (0,10000), (-10000,0), (0,-10000)
XY linear interpolation (10000,10000), (-10000,-10000)
XY circular interpolation (10000, 0, 5000, 0), (G03, -10000, 0, -5000, 0)

The CDA data moves the axes to command position 0 after repeating XY-axis SIN operation (one cycle) for a travel distance of 720 points for 10 times (travel distance data of 7200 points).

3.4.6 Command

Using the keyboard, enter the following commands in the console application. Then, press [Enter] to execute.

0: Show Menu
1: Display current status (MSTS, SSTS, RSTS, CTR1, CTR2, Seq No., and Page No. of each axis)
2: Load MDA for relative positioning
3: Load MDA for linear interpolation
4: Load MDA for circular interpolation
5: Execute MDA (Start ThreadMDA)
6: Load and execute CDA (Start ThreadCDA)
7: MDA CSTOP
8: CDA HALT (CSTOP)
9: QSTOP
99: Exit program

To execute MDA operation: load MDA → execute MDA.
To execute CDA operation: execute CDA.
Status display can be only enabled during MDA or CDA stopped in current version.

4. API Function Reference

4.1 Types of API Functions

There are two types of API functions: driver functions and library functions.
Driver functions are provided as DLL to be started as external functions.
Library functions are provided as source programs to be built together with the application for use.
Library functions include library functions compliant with the CPD series and library functions for the NCB series. CPD-series compliant library functions for MDA control and CMP control, which are specific to NCB, cannot be used simultaneously with library functions for the NCB series.

4.2 Function Naming Conventions

CPD-series compliant driver functions	cp670_xxxx()	...	Specifications are equivalent to those of driver functions for the CPD series. (Some are specific to NCB)
CPD-series compliant library functions	hcp670_xxxx()	...	Specifications are equivalent to those of library functions for the CPD series. (Some are specific to NCB)
NCB-series library functions	ncb670_xxxx()	...	Library functions for the NCB series.

4.3 List of Driver Functions

No.	Function name	Function
1	cp670_GetDeviceCount()	Acquire number of boards
2	cp670_GetDeviceInfo()	Acquire device information
3	cp670_OpenDevice()	Open device
4	cp670_CloseDevice()	Close device
5	cp670_rMstsW()	Read main status
6	cp670_rSstsW()	Read sub-status
7	cp670_wCmdW()	Write control command
8	cp670_rReg()	Read register
9	cp670_wReg()	Write register
10	cp670_rPortW()	Read word (2 bytes) from option port
11	cp670_wPortW()	Write word (2 bytes) to option port
12	cp670_rExPortW()	Read word (2 bytes) from expansion port (MDA)
13	cp670_wExPortW()	Write word (2 bytes) to expansion port (MDA)
14	cp670_rBufDW()	Read input/output buffer
15	cp670_wBufDW()	Write input/output buffer
16	cp670_WaitNextInterrupt()	Wait for interrupt
17	cp670_CancelWaitInterrupt()	Cancel wait for interrupt

Table 4.3-1 List of driver functions

4.4 Driver Function Details

When using, replace "libname" in the VB and VC# function format below with each DLL name.

4.4.1 cp670_GetDeviceCount()　Acquire number of boards

Function	Acquires the number of NCB boards currently connected to the PC.

Language	Format
VC++	DWORD WINAPI cp670_GetDeviceCount(DWORD* count);
VB	Declare Function cp670_GetDeviceCount Lib **"***libname***"** (ByRef　count　As Integer) As Integer
VC#	[DllImport("*libname* ")] public static extern uint cp670_GetDeviceCount(ref uint count);

Argument	Description
count	Acquired number of NCB boards

4.4.2 cp670_GetDeviceInfo() Acquire device information

Function	Acquires device information of the specified number of NCB boards currently connected to the PC. The acquired device information is stored in the device information structure array. This device information is used to open devices.

Language	Format
VC++	DWORD WINAPI cp670_GetDeviceInfo(DWORD*count, HCP670INFO* hInfo);
VB	Declare Function cp670_GetDeviceInfo Lib "libname" (ByRef count As Integer, _ ByRef hInfo As HCP670INFO) As Integer
VC#	[DllImport("*libname* ")] public static extern uint cp670_ GetDeviceInfo (ref uint count, ref HCP670INFO hInfo);

Argument	Description
count	Number of NCB boards
hInfo	Device information to acquire

Remarks	Secures an area to store device information for the number of boards.

4.4.3 cp670_OpenDevice()　Open device

Function	Opens the NCB with the passed device information to acquire the device handle that will distinguish it from others. Hereafter, this device handle will be used to access this NCB.

Language	Format
VC++	DWORD WINAPIcp670_OpenDevice(DWORD* hDevID, HCP670INFO* hInfo);
VB	Declare Function cp670_OpenDevice Lib "libname"(ByRef hDevID As Integer, _ ByRef hInfo As HCP670INFO) As Integer
VC#	[DllImport("*libname* ")] public static extern uint cp670_ OpenDevice (ref uint hDevID, ref HCP670INFO hInfo);

Argument	Description
hDevID	Device handle
hInfo	Device information

4.4.4 cp670_CloseDevice()　Close device

Function	Closes the NCB specified by the device handle. Hereafter, this device handle is invalid, and accessing this NCB will not be possible.

Language	Format
VC++	DWORD WINAPI cp670_CloseDevice(DWORD hDevID);
VB	Declare Function cp670_CloseDevice Lib "libname"(ByVal hDevID As Integer) As Integer
VC#	[DllImport("*libname* ")] public static extern uint cp670_ CloseDevice(uint hDevID);
Language	function cp670_CloseDevice(hDevID: UINT): UINT; stdcall; external "libname";

Argument	Description
hDevID	Device handle

Remarks	Be sure to perform end processing, such as pulse stop, before closing the device.

4.4.5 cp670_rMstsW() Read main status

Function	Reads the main status of the designated axis controlled by the board specified by the device handle.

Language	Format
VC++	DWORD WINAPI cp670_rMstsW(DWORD hDevID, WORD axis, WORD* wMsts);
VB	Declare Function cp670_rMstsW Lib "libname"(ByVal hDevID As Integer, _ ByVal axis As Short, ByRef wMsts As Short) As Integer
VC#	[DllImport("*libname* ")] public static extern uint cp670_rMstsW(uint hDevID, ushort axis, ref ushort wMsts);

Argument	Description
hDevID	Device handle
axis	Axis designation
wMsts	Main status

(1) Contents of main status register

b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	**b5**	**b4**	**b3**	b2	b1	b0
SPDF	SPRF	0	SCMP x					SSC x		**SINT**	**SERR**	**SEND**	0	SRUN	SSCM
			5	4	3	2	1	1	0						

Bit	Name	Description
0	SSCM	'1': Written start command present
1	SRUN	'1': Running
3	**SEND**	'1': Stopped (*2)
4	**SERR**	'1': Error notification present (Reset to '0' on error status (REST) readout)
5	**SINT**	'1': Event notification present (Reset to '0' on event status (RIST) readout)
7,6	SSCx	Running or stopped sequence number (Value set in PRMD.b17,16)
12-8	SCMPx	'1': On CMPX comparison condition match
14	SPRF	'1': Preregister for next operation FULL ('0': Write enabled)
15	SPDF	'1': CMP5 preregister FULL ('0': Write enabled)

4.4.6 cp670_rSstsW() Read sub-status

Function	Reads the sub-status of the designated axis controlled by the board specified by the device handle.

Language	Format
VC++	DWORD WINAPI cp670_rSstsW(DWORD hDevID, WORD axis, WORD* wSsts);
VB	Declare Function cp670_rSstsW Lib "libname"(ByVal hDevID As Integer, _ ByVal axis As Short, ByRef wSsts As Short) As Integer
VC#	[DllImport("*libname* ")] public static extern uint cp670_rSstsW(uint hDevID, ushort axis, ref ushort wSsts);

Argument	Description
hDevID	Device handle
axis	Axis designation
wSsts	Sub-status

(1) Contents of sub-status register

b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
SDLS	**SOLS**	**SMEL**	**SPEL**	**SALM**	SFC	SFD	SFU	0	0	0	0	0	0	**SVRS**	**SVON**

Bit	Name	Description
0	**SVON**	'1': "SVON" being output
1	**SVRS**	'1': "SVRST" being output
8	SFU	'1': Accelerating
9	SFD	'1': Decelerating
10	SFC	'1': On constant-speed operation
11	**SALM**	'1': SVALM (while servo alarm signal is ON)
12	**SPEL**	'1': On +ELS detection
13	**SMEL**	'1': On -ELS detection
14	**SOLS**	'1': On OLS detection
15	**SDLS**	'1': On DLS detection

4.4.7 cp670_wCmdW()　Write control command

Function	Writes, to the command buffer, control command data for the designated axis controlled by the board specified by the device handle.

Language	Format
VC++	DWORD WINAPI cp670_wCmdW(DWORD hDevID, WORD axis, WORD wCmd);
VB	Declare Function cp670_wCmdW Lib "libname"(ByVal hDevID As Integer, _ ByVal axis As Short, ByVal wCmd As Short) As Integer
VC#	[DllImport("*libname* ")] public static extern uint cp670_wCmdW(uint hDevID, ushort axis, ushort wCmd);

Argument	Description
hDevID	Device handle
axis	Axis designation
wCmd	Command data

(1) Contents of command data

b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
0	0	0	0	Executing axis (SELx)				Command code (code)							

	b11	b10	b9	b8	Axis to write
	Executing axis (SELx)				
Axis designation	U	Z	Y	X	Any axis among X to U

(2) Executing axis specification (SELx)

When writing the command code for each axis individually, set these 4 bits to "0".

When writing the same command code to two or more axes, specify the target axes with SELx bits.

(3) List of command codes

Code	Command details	Code	Command details
04h	Software reset	29h	Latch input representation
05h	Emergency stop	2ah	Same as STA input, only for own axis
06h	STA output (simultaneous start)	2bh	Shift operation preregister
07h	STP output (simultaneous stop)	2ch	Shift RCMP5 preregister
10h	SVON OFF	40h	Immediate change to FL speed
18h	SVON ON	41h	Immediate change to FH speed
11h	SVRESET OFF	42h	Change to FL speed by deceleration
19h	SVRESET ON	43h	Change to FH speed by acceleration
04h	Software reset	49h	Quick stop
20h	Counter 1 reset	4ah	Deceleration stop
21h	Counter 2 reset	50h	FL constant-speed start
22h	Counter 3 reset	51h	FH constant-speed start
23h	Counter 4 reset	52h	Deceleration stop after FH constant-speed start
24h	Error counter clear signal output	53h	High speed start
25h	Error counter clear signal reset	54h	Residual FL constant-speed start
26h	Operation preregister cancel	55h	Residual FH constant-speed start
27h	RCMP5 preregister cancel	57h	Residual high speed start
28h	Positioning control start (PCS input representation)		

4.4.8 cp670_rReg() Read register

Function	Reads the content of the specified register for the designated axis controlled by the board specified by the device handle.

Language	Format
VC++	DWORD WINAPI cp670_rReg DWORD hDevID, WORD axis, BYTE byCmd, DWORD* dwData);
VB	Declare Function cp670_rReg Lib "libname"(ByVal hDevID As Integer, _ ByVal axis As Short, ByVal byCmd As Byte, ByRef dwData As Integer) As Integer
VC#	[DllImport("*libname* ")] public static extern uint cp670_rReg (uint hDevID, ushort axis, byte byCmd, ref uint dwData);

Argument	Description
hDevID	Device handle
axis	Axis designation
byCmd	Read register command
dwData	Read register data

4.4.9 cp670_wReg() Write register

Function	Writes data to the specified register for the designated axis controlled by the board specified by the device handle.

Language	Format
VC++	DWORD WINAPI cp670_wReg DWORD hDevID, WORD axis, BYTE byCmd, DWORD dwData);
VB	Declare Function cp670_wReg Lib "libname"(ByVal hDevID As Integer, _ ByVal axis As Short, ByVal byCmd As Byte, ByVal dwData As Integer) As Integer
VC#	[DllImport("*libname* ")] public static extern uint cp670_wReg (uint hDevID, ushort axis, byte byCmd, uint dwData);

Argument	Description
hDevID	Device handle
axis	Axis designation
byCmd	Write register command
dwData	Write register data

(1) Register/Preregister read/write commands

No.	Content	Register			Preregister		
		Name	Command (HEX)		Name	Command (HEX)	
			Read	Write		Read	Write
1	Travel distance, target position	RMV	d0	90	PRMV	c0	80
2	Initial speed	RFL	d1	91	PRFL	c1	81
3	Operation speed	RFH	d2	92	PRFH	c2	82
4	Acceleration rate	RUR	d3	93	PRUR	c3	83
5	Deceleration rate	RDR	d4	94	PRDR	c4	84
6	Speed multiplier	RMG	d5	95	PRMG	c5	85
7	Deceleration start point	RDP	d6	96	PRDP	c6	86
8	Operation mode	RMD	d7	97	PRMD	c7	87
9	Circular interpolation center	RIP	d8	98	PRIP	c8	88
10	S-curve acceleration range	RUS	d9	99	PRUS	c9	89
11	S-curve deceleration range	RDS	da	9a	PRDS	ca	8a
12	Auxiliary speed	RFA	db	9b			
13	Environment setting 1	RENV1	dc	9c			
14	Environment setting 2	RENV2	dd	9d			
15	Environment setting 3	RENV3	de	9e			
16	Environment setting 4	RENV4	df	9f			
17	Environment setting 5	RENV5	e0	a0			
18	Environment setting 6	RENV6	e1	a1			
19	Environment setting 7	RENV7	e2	a2			
20	Counter 1 (command pulse output)	RCTR1	e3	a3			
21	Counter 2 (encoder input)	RCTR2	e4	a4			
22	Counter 3 (error counter)	RCTR3	e5	a5			
23	Counter 4 (general-purpose counter)	RCTR4	e6	a6			
24	Comparator 1 data	RCMP1	e7	a7			
25	Comparator 2 data	RCMP2	e8	a8			
26	Comparator 3 data	RCMP3	e9	a9			
27	Comparator 4 data	RCMP4	ea	aa			
28	Comparator 5 data	RCMP5	eb	ab	PRCP5	cb	8b
29	Event mask setting	RIRQ	ec	ac			
30	Counter 1 latch data	RLTC1	ed				
31	Counter 2 latch data	RLTC2	ee				
32	Counter 3 latch data	RLTC3	ef				
33	Counter 4 latch data	RLTC4	f0				
34	Extension status	RSTS	f1				
35	Error status	REST	f2				
36	Event status	RIST	f3				
37	Positioning counter	RPLS	f4				
38	EZ counter, command speed monitor	RSPD	f5				
39	Deceleration start point calculation result	RSDC	f6				
40	Number of circular interpolation steps	RCI	fc	bc	PRCI	cc	8c
41	Circular interpolation step counter	RCIC	fd				
42	Interpolation status	RIPS	ff				

4.4.10 cp670_rPortW()　Read word (2 bytes) from option port

Function	Reads the content of the designated option port of the board specified by the device handle.

Language	Format
VC++	DWORD WINAPI cp670_rPortW(DWORD hDevID, WORD wCmd, WORD* wData);
VB	Declare Function cp670_rPortW Lib "libname" _ (ByVal hDevID As Integer, ByVal wCmd As Short, ByRef wData As Short) As Integer
VC#	[DllImport("*libname* ")] public static extern uint cp670_rPortW (uint hDevID, ushort wCmd, ref ushort wData);

Argument	Description
hDevID	Device handle
wCmd	Read option port command
wData	Read option port data

(1)　Option port read command

No.	Content	Name	Command
1	ELS polarity setting status	ELPOL	0080h
2	DLS/PCS input selection status	DLS/PCS	0082h
3	CMP4STA output setting status	C4STA	0084h
4	CMP5STP output setting status	C5STP	0086h
5	X-UCMP3-5 output selection status	COTSEL1	008ch
6	Synchronization CMP enable setting status	SYNC_C_EN	0094h
7	Synchronization CMP selection status	XSYNC_C	0096h
8	Encoder filter setting status	ENFIL	00a2h
9	External pulsar input/output setting status	J3_SEL	00a4h
10	Axis count assessment port readout	AXIS	00a8h
11	Generic input	DIN	00e0h
12	Generic output	DOUT	00e2h
13	DI1-DI4 interrupt enable setting status	INT_EN	00e4h
14	DI1-DI4 interrupt polarity setting status	INT_POL	00e6h
15	DI1-DI4 interrupt status	INT_ST	00e8h
16	DI1-DI4 interrupt filter setting status	INT_FIL	00eah
17	Synchronization function setting status	SYNC_SET1	00f0h

4.4.11 cp670_wPortW()　Write word (2 bytes) to option port

Function	Writes data to the designated option port of the board specified by the device handle.

Language	Format
VC++	DWORD WINAPI cp670_wPortW(DWORD hDevID, WORD wCmd, WORD wData);
VB	Declare Function cp670_wPortW Lib "libname" _ (ByVal hDevID As Integer, ByVal wCmd As Short, ByVal wData As Short) As Integer
VC#	[DllImport("*libname* ")] public static extern uint cp670_wPortW (uint hDevID, ushort wCmd, ushort wData);

Argument	Description
hDevID	Device handle
wCmd	Write option port command
wData	Write option port data

(1) Option port write command

No.	Content	Name	Command
1	Set ELS polarity	ELPOL	0080h
2	Select DLS/PCS input	DLS/PCS	0082h
3	Set CMP4STA output	C4STA	0084h
4	Set CMP5STP output	C5STP	0086h
5	Select X-UCMP3-5 output	COTSEL1	008ch
6	Enable synchronization CMP	SYNC_C_EN	0094h
7	Select synchronization CMP	XSYNC_C	0096h
8	Set encoder filter	ENFIL	00a2h
9	Set external pulsar input/output	J3_SEL	00a4h
10	Initialize board	BRD_RST	00a8h
11	Generic output	DOUT	00e2h
12	Enable DI1-DI4 interrupt	INT_EN	00e4h
13	Set DI1-DI4 interrupt polarity	INT_POL	00e6h
14	Clear DI1-DI4 interrupt	INT_ST	00e8h
15	Set DI1-DI4 interrupt filter	INT_FIL	00eah
16	Set synchronization function	SYNC_SET1	00f0h

4.4.12 cp670_rExPortW()　Read word (2 bytes) from expansion port (MDA)

Function	Reads the contents of the designated expansion port (MDA) of the board specified by the device handle.

Language	Format
VC++	DWORD WINAPI cp670_wExPortW(DWORD hDevID, WORD wCmd, WORD* wData);
VB	Declare Function cp670_wExPortW Lib "libname" _ (ByVal hDevID As Integer, ByVal wCmd As Short, ByRef wData As Short) As Integer
VC#	[DllImport("*libname* ")] public static extern uint cp670_wExPortW (uint hDevID, ushort wCmd, ref ushort wData);

Argument	Description
hDevID	Device handle
wCmd	Read expansion port command
wData	Read expansion port data

4.4.13 cp670_wExPortW()　Write word (2 bytes) to expansion port (MDA)

Function	Writes data to the designated expansion port (MDA) of the board specified by the device handle.

Language	Format
VC++	DWORD WINAPI cp670_wExPortW(DWORD hDevID, WORD wCmd, WORD　wData);
VB	Declare Function cp670_wExPortW Lib "libname" _ (ByVal hDevID As Integer, ByVal wCmd As Short, ByVal wData As Short) As Integer
VC#	[DllImport("*libname* ")] public static extern uint cp670_wExPortW (uint hDevID, ushort wCmd, ushort wData);

Argument	Description
hDevID	Device handle
wCmd	Write Expansion port command
wData	Write Expansion port data

4.4.14 cp670_rBufDW()　Read input/output buffer

Function	Reads the input/output buffer for the designated axis of the board specified by the device handle.

Language	Format
VC++	DWORD WINAPI cp670_rBufDW(DWORD hDevID, WORD axis, DWORD* dwData);
VB	Declare Function cp670_rBufDW Lib "libname" _ (ByVal hDevID As Integer, ByVal axis As Short, ByRef dwData As Integer) As Integer
VC#	[DllImport("*libname* ")] public static extern uint cp670_rBufDW(uint hDevID, ushort axis, ref uint dwData);

Argument	Description
hDevID	Device handle
axis	Axis designation
dwData	Read input/output buffer data

Remarks	**<Difference with the read register function>** Read register functions: Read the target register via the PCL input/output buffer for the designated axis Input/Output buffer handling functions: Read the PCL input/output buffer for the designated axis **<Practical use of read register function (cp670_rBufDW())>** Read register data of multiple axes simultaneously. ● Specify the preregisters for the axes to read by using a "command" based on the cp670_wCmdW() function. ● The axis to designate can be any. Set two or more axes in the executing axis specification (SELx) of the control command data (cmd). Set read command for command code (code). ● The input/output buffers of all axes specified in the executing axis specification (SELx) are read.

4.4.15 cp670_wBufDW()　Write input/output buffer

Function	Writes data to the input/output buffer for the designated axis controlled by the board specified by the device handle.

Language	Format
VC++	DWORD WINAPI cp670_wBufDW(DWORD hDevID, WORD axis, DWORD　dwData);
VB	Declare Function cp670_wBufDW Lib "libname" _ (ByVal hDevID As Integer, ByVal axis As Short, ByVal dwData As Integer) As Integer
VC#	[DllImport("*libname* ")] public static extern uint cp670_wBufDW(uint hDevID, ushort axis, uint dwData);

Argument	Description
hDevID	Device handle
axis	Axis designation
dwData	Input/output buffer write data

Remarks	**<Difference with the write register function>** Write register functions: Write the target register via the PCL input/output buffer for the designated axis Input/Output buffer handling functions: Write the PCL input/output buffer for the designated axis **<Practical use of write register function (cp670_wBufDW())>** Write register data for multiple axes simultaneously. ● Write the given data to all of the input/output buffers for the target axes. ● Specify the preregisters/registers of all of the axes to write with the "command" based on the cp670_wCmdW() function. ● The axis to designate can be any. Set two or more axes in the executing axis specification (SELx) of the control command data (cmd). Set write command for command code (code).

4.4.16 cp670_WaitNextInterrupt()　Wait for interrupt

Function	Wait-for-interrupt function. It is called from within the thread that will wait for an interrupt.

Language	Format
VC++	DWORD cp670_WaitNextInterrupt(DWORD *hDevID*, DWORD *dwTime*);

Argument	Description
hDevID	Device handle
dwTime	Specifies the timeout time in units of milliseconds (ms). When the timeout time elapses, the function returns the control even if an interrupt has not been generated. Specifying "0" makes the function check the specified board for the presence of interrupt and return the control immediately. Specifying INFINITE makes the function wait until an interrupt is generated in the specified board, or until the wait for interrupt cancellation function "cp670_CancelWaitInterrupt" is issued.

Remarks	Causes of interrupt may be roughly classified into the following three groups: (1) PCL (2) Generic input 1 to 4 (3) INTS They are enabled/disabled in "BINTEN". In addition, they need settings in "RIRQ" where causes of interrupt by PCL are set (refer to User's Manual <Operation>), or in "DI_INT" and "DI_POL" where interrupt by generic inputs is set. For details, see "9.5 Interrupt Mechanism". If the function is successful, it returns the reason for returning the control. ● WAIT_OBJECT_0 (Windows reserved constant) It means that an interrupt has been generated in the specified board. After the interrupt is generated, make sure the cause is always read. ● WAIT_TIMEOUT (Windows reserved constant) It means that no interrupt has been generated in the specified board before the timeout time expired. ● ALREADY_WAITING (0x00010000) It means that the board is already waiting for an interrupt. ● WAITING_CANCELED (0x00020000) It means that the wait for interrupt has been canceled by the CancelWaitInterrupt() function. If the function fails, it returns values described in "4.10 Function Return Values".

4.4.17 cp670_CancelWaitInterrupt()　Cancel wait for interrupt

Function	Cancels the wait for interrupt.

Language	Format
VC++	DWORD cp670_CancelWaitInterrupt(DWORD *hDevID*);

Argument	Description
hDevID	Device handle

4.5 List of Library Functions (CPD-Series Compliant)

4.5.1 Device-related

No.	Function name	Function
1	hcp670_GetDevInfo	Acquire number of boards and device information
2	hcp670_DevOpen	Open device and initialize registers
3	hcp670_DevClose	Close device

Table 4.5-1 List of device-related library functions

4.5.2 Initialization

No.	Function name	Function
4	hcp670_SetOrgMode	Set homing mode
5	hcp670_SetEls	Set ELS
6	hcp670_SetOls	Set OLS
7	hcp670_SetSvAlm	Set SVALM
8	hcp670_SetEz	Set encoder phase-Z
9	hcp670_SetDlsSel	Select and set DLS/PCS input
10	hcp670_SetInpos	Set INPOS
11	hcp670_SetSvCtrCl	Set error counter clear output
12	hcp670_SetSls	Set software limit
13	hcp670_SetCmdPulse	Select command pulse output format
14	hcp670_SetAccProfile	Set acceleration/deceleration type
15	hcp670_SetAutoDec	Switch deceleration start point calculation method between auto and manual

Table 4.5-2 List of initialization library functions

4.5.3 Status readout

No.	Function name	Function
16	hcp670_ReadMainSts	Read main status
17	hcp670_ReadErrorSts	Read error status
18	hcp670_ReadEventSts	Read event status
19	hcp670_ReadSubSts	Read sub-status
20	hcp670_ReadExSts	Read extension status
21	hcp670_ReadSpd	Read command speed
22	hcp670_ReadCtr	Read counter

Table 4.5-3 List of library functions for reading statuses

4.5.4 Operation settings

No.	Function name	Function
23	hcp670_SetFLSpd	Set base speed
24	hcp670_SetAuxSpd	Set auxiliary speed
25	hcp670_SetAccRate	Set acceleration rate
26	hcp670_SetDecRate	Set deceleration rate
27	hcp670_SetMult	Set speed multiplier register value
28	hcp670_SetEventMask	Set event mask
29	hcp670_SetDecPoint	Set deceleration start point

Table 4.5-4 List of library functions for making operation settings

4.5.5 Operational settings

No.	Function name	Function
30	hcp670_WritOpeMode	Set operation mode
31	hcp670_WritFHSpd	Set operation speed
32	hcp670_WritPos	Set positioning travel distance
33	hcp670_WritLine	Set linear interpolation travel distance
34	hcp670_WritCircl	Set circular interpolation travel distance
35	hcp670_WritCtr	Preset counter

Table 4.5-5 List of library functions for making operational settings

4.5.6 Operation control commands

No.	Function name	Function
36	hcp670_DecStop	Deceleration stop
37	hcp670_QuickStop	Quick stop
38	hcp670_EmgStop	Emergency stop
39	hcp670_AccStart	Acceleration start
40	hcp670_CnstStartFH	FH constant-speed start
41	hcp670_CnstStartFL	FL constant-speed start
42	hcp670_CnstStartByDec	Deceleration stop after FH constant-speed start
43	hcp670_SvOn	Servo ON
44	hcp670_SvOff	Servo OFF
45	hcp670_SvResetOn	Servo reset ON
46	hcp670_SvResetOff	Servo reset OFF
47	hcp670_PMOn	Pulse motor excitation ON
48	hcp670_PMOff	Pulse motor excitation OFF

Table 4.5-6 List of operation control command library functions

4.5.7 DPRAM readout (NCB-specific)

No.	Function name	Function
59	hcp670_ReadErno	Read DP_ERNO
60	hcp670_ReadDpFC	Read counter/speed via DPRAM
61	hcp670_ReadDpReg	Read register via DPRAM

Table 4.5-9 List of library functions for DPRAM readout

4.5.8 Calculation function

No.	Function name	Function
62	hcp670_CalAccRate	Calculate acceleration/deceleration rate

Table 4.5-10 List of calculation functions

4.6 List of Library Functions (NCB Series)

4.6.1 Event monitoring

No.	Function name	Function
63	ncb670_StartEvent	Start event monitoring
64	ncb670_StopEvent	Stop event monitoring
65	ncb670_IsAck	Check ACK/NAK
66	ncb670_WaitMdaLend	Check MDA load completion
67	ncb670_WaitMdaEnd	Check MDA execution completion
68	ncb670_WaitCdaLend	Check CDA load completion
69	ncb670_WaitCdaCLend	Check continuous CDA load completion
70	ncb670_WaitCrq	Check request for continuous CDA
71	ncb670_WaitCdaEnd	Check CDA execution completion
72	ncb670_WaitCmpLend	Check CMP load completion
73	ncb670_WaitCmpEnd	Check CMP execution completion

Table 4.6-1 List of library functions for event monitoring

4.6.2 MDA control

No.	Function name	Function
74	ncb670_NcbLoad	Load MDA(CDA) data
75	ncb670_MdaExec	Start MDA execution
76	ncb670_MdaCStop	Stop MDA execution cycle
77	ncb670_MdaQStop	Quick-stop MDA execution
78	ncb670_MdaBreakExec	Break MDA execution
79	ncb670_MdaStepExec	Execute MDA step-by-step

Table 4.6-2 List of library functions for MDA control

4.6.3 CDA control

No.	Function name	Function
80	ncb670_CdaExec	Start CDA execution
81	ncb670_CdaHalt	Stop CDA execution temporarily
82	ncb670_CdaQstop	Quick-stop CDA execution
83	ncb670_CdaCancel	Cancel CDA data

Table 4.6-3 List of library functions for CDA control

4.6.4 CMP control

No.	Function name	Function
84	ncb670_CmpLoad	Load CMP data
85	ncb670_CmpExec	Start CMP execution
86	ncb670_CmpCancel	Cancel CMP data

Table 4.6-4 List of library functions for CMP control

4.6.5 PCL access via DPRAM

No.	Function name	Function
87	ncb670_rMstsW	Read main status via DPRAM
88	ncb670_rSstsW	Read sub status via DPRAM
89	ncb670_wCmdW	Write PCL command via DPRAM
90	ncb670_rReg	Read register via DPRAM
91	ncb670_wReg	Write register via DPRAM

Table 4.6-5 List of library functions for PCL access via DPRAM

4.6.6 Read Firmware version

No.	Function name	Function
92	ncb670_ReadFirmwareVersion	Read Firmware version

Table 4.6-6 List of library functions for Read Firmware version

4.7 Library Function Details (CPD-Series Compliant)

4.7.1 Device-related

(1) **hcp670_GetDevInfo() Acquire number of boards and device information**

Function	Acquires the number of NCB boards currently connected to the PC and their device information.

Language	Format
VC++	DWORD hcp670_GetDevInfo(DWORD* *dwNum*, HCP670INFO *hInfo*);
VB	Public Function hcp670_GetDevInfo(ByRef *dwNum* As Integer, ByRef *hInfo* As HCP670INFO) As Integer
VC#	public static uint hcp670_GetDevInfo(ref uint *dwNum*, ref HCP670INFO[] *hInfo*);

Argument	Description
dwNum	Number of NCB boards
hInfo	NCB device information storage structure

(2) **hcp670_DevOpen() Open device and initialize registers and option ports**

Function	Opens the NCB with the specified device information to acquire the device handle that will distinguish it from other NCBs. Hereafter, this device handle will be used to access the specified NCB. It also initializes the registers and option ports of the opened NCB.

Language	Format
VC++	DWORD hcp670_DevOpen(DWORD* *hDevID*, HCP670INFO* *hInfo*);
VB	Public Function hcp670_DevOpen(ByRef *hDevID* As Integer, ByRef *hInfo* As HCP670INFO) _ As Integer
VC#	public static uint hcp670_DevOpen(ref uint *hDevID*, ref HCP670INFO *hInfo*);

Argument	Description
hDevID	Device handle storage location
hInfo	NCB device information storage structure

Remarks

Register	Content	Default value	Note
PRFL, RFL	Base speed	200	200pps
PRFH, RFH	Operation speed	2000	2000pps
PRUR, RUR	Acceleration rate	1364	During linear acceleration/deceleration: 200 → 2000pps (2000pps → 200pps) Acceleration (deceleration) time: Approx. 0.5s
PRMG, RMG	Speed multiplier	299	x1 multiplication
RFA	Auxiliary speed	200	200pps
PRMD, RMD	Operation mode	08008000h	
RENV1	Environment setting 1	20434004h	
RENV2	Environment setting 2	0020fd55h	
RENV3	Environment setting 3	00f00002h	Homing mode 2 (OLS + phase-Z) etc.
RIRQ	Event mask setting	1	On normal stop
Other		0	

Option port	Default value	Note
ELS input polarity	0	Normal close for all axes
PCS/DLS switching	ffh	PCS for all axes
Other	0	

(3) **hcp670_DevClose() Close device**

Function	Closes the NCB specified by the device handle. Hereafter, this device handle is invalid.

Language	Format
VC++	DWORD hcp670_DevClose(DWORD *hDevID*);
VB	Public Function hcp670_DevClose(ByVal *hDevID* As Integer) As Integer
VC#	public static uint hcp670_DevClose(uint *hDevID*);

Argument	Description
hDevID	Device handle

Remarks	Perform end processing for the NCB before closing the device.

4.7.2 Initialization

(4) **hcp670_SetOrgMode() Set homing mode**

Function	Sets homing mode for the designated axis controlled by the NCB specified by the device handle.

Language	Format
VC++	DWORD hcp670_SetOrgMode(DWORD *hDevID*, WORD *axis*, WORD *mode*);
VB	Public Function hcp670_SetOrgMode(_ ByVal *hDevID* As Integer, ByVal *axis* As Short, ByVal *mode* As Short) As Integer
VC#	public static uint hcp670_SetOrgMode(uint *hDevID*, ushort *axis*, ushort *mode*);

Argument	Description	
hDevID	Device handle	
axis	Axis designation [X:0, Y: 1, Z: 2, U: 3]	
mode	Homing mode	
	0: ORGmode0	Quick stop (deceleration stop in acceleration/deceleration operation) on OLS OFF → ON.
	1: ORGmode1	After quick stop (deceleration stop in acceleration/deceleration operation) on OLS OFF → ON, move in opposite direction at constant-auxiliary speed until OLS OFF, then move back at auxiliary speed to stop quickly on OLS OFF → ON.
	2: ORGmode2	When in constant speed operation, quick stop on the first phase-Z count after OLS OFF → ON. When in acceleration/deceleration operation, deceleration on OLS OFF → ON and quick stop on the first phase-Z count.
	3: ORGmode3	When in constant speed operation, quick stop on the first phase-Z count after OLS OFF → ON. When in acceleration/deceleration operation, deceleration on OLS OFF → ON and deceleration stop on the first phase-Z count.
	4: ORGmode4	After quick stop (deceleration stop in acceleration/deceleration operation) on OLS OFF → ON, move in opposite direction at auxiliary speed to quick stop on the first phase-Z count after OLS ON → OFF.
	5: ORGmode5	After quick stop (deceleration stop in acceleration/deceleration operation) on OLS ON → OFF, move in opposite direction to quick stop (deceleration stop in acceleration/deceleration operation) on the first phase-Z count after OLS ON → OFF.
	6: ORGmode6	After stopping on ELS ON, move in opposite direction at constant-auxiliary speed to quick stop on ELS OFF.
	7: ORGmode7	After stopping on ELS ON, move in opposite direction at constant-auxiliary speed to quick stop on the first phase-Z count after ELS OFF.
	8: ORGmode8	After stopping on ELS ON, move in opposite direction to quick stop (deceleration stop in acceleration/deceleration operation) on the first phase-Z count after ELS OFF.
	9: ORGmode9	After ORGmode0 operation, return to machine position (CTR2) 0.
	10: ORGmode10	After ORGmode3 operation, return to machine position (CTR2) 0.
	11: ORGmode11	After ORGmode5 operation, return to machine position (CTR2) 0.
	12: ORGmode12	After ORGmode8 operation, return to machine position (CTR2) 0.

Remarks	When using homing modes 9 to 12 on an axis without encoder input, the CTR2 input must be set to the command position.

(5) **hcp670_SetEls() Set ELS**

Function	Sets ELS input polarity and stopping method on ELS input for the designated axis controlled by the NCB specified by the device handle.

Language	Format
VC++	DWORD hcp670_SetEls(DWORD *hDevID*, WORD *axis*, WORD *pol*, WORD *stop*);
VB	Public Function hcp670_SetEls(ByVal *hDevID* As Integer, ByVal *axis* As Short, _ ByVal *pol* As Short, ByVal *stop* As Short) As Integer
VC#	public static uint hcp670_SetEls(uint *hDevID*, ushort *axis*, ushort *pol* ushort *stop*);

Argument	Description
hDevID	Device handle
axis	Axis designation [X:0, Y: 1, Z: 2, U: 3]
pol	Input polarity [Normal close: 0, Normal open: 1]
stop	Stopping method [Quick stop: 0, Deceleration stop: 1]

(6) **hcp670_SetOls() Set OLS**

Function	Sets OLS input polarity for the designated axis controlled by the NCB specified by the device handle.

Language	Format
VC++	DWORD hcp670_SetOls(DWORD *hDevID*, WORD *axis*, WORD *pol*);
VB	Public Function hcp670_SetOls(ByVal *hDevID* As Integer, ByVal *axis* As Short, ByVal *pol* As Short) _ As Integer
VC#	public static uint hcp670_SetOls(uint *hDevID*, ushort *axis*, ushort *pol*);

Argument	Description
hDevID	Device handle
axis	Axis designation [X:0, Y: 1, Z: 2, U: 3]
pol	Input polarity [Normal close: 0, Normal open: 1]

(7) **hcp670_SetSvAlm () Set SVALM**

Function	Sets SVALM input polarity and stopping method for the designated axis controlled by the NCB specified by the device handle.

Language	Format
VC++	DWORD hcp670_SetSvAlm(DWORD *hDevID*, WORD *axis*, WORD *pol*, WORD *stop*);
VB	Public Function hcp670_SetSvAlm(ByVal *hDevID* As Integer, ByVal *axis* As Short, _ ByVal *pol* As Short, ByVal *stop* As Short) As Integer
VC#	public static uint hcp670_SetSvAlm(uint *hDevID*, ushort *axis*, ushort *pol*, ushort *stop*);

Argument	Description
hDevID	Device handle
axis	Axis designation [X:0, Y: 1, Z: 2, U: 3]
pol	Input polarity [Normal close: 0, Normal open: 1]
stop	Stopping method [Quick stop: 0, Deceleration stop: 1]

(8) **hcp670_SetEz() Set encoder phase-Z**

Function	Sets processing on encoder phase-Z input for the designated axis controlled by the NCB specified by the device handle.

Language	Format
VC++	DWORD hcp670_SetEz(DWORD *hDevID*, WORD *axis*, WORD *zcount*, WORD *pol*);
VB	Public Function hcp670_SetEz(ByVal *hDevID* As Integer, ByVal *axis* As Short, _ ByVal *zcount* As Short, ByVal *pol* As Short) As Integer
VC#	public static uint hcp670_SetEz(uint *hDevID*, ushort *axis*, ushort *zcount*, ushort *pol*);

Argument	Description
hDevID	Device handle
axis	Axis designation [X:0, Y: 1, Z: 2, U: 3]
zcount	Number of phase-Z counts when homing [0:first to 15:16th]
pol	Input polarity [Normal open: 0, Normal close: 1]

(9) **hcp670_SetDlsSel() Select and set DLS/PCS input**

Function	Selects DLS/PCS input and its polarity, and sets the processing on DLS input for the designated axis controlled by the NCB specified by the device handle.

Language	Format
VC++	DWORD hcp670_SetDlsSel(DWORD *hDevID*, WORD *axis*, WORD *para*, WORD *pol* WORD *motion*, WORD *latch*);
VB	Public Function hcp670_SetDlsSel(ByVal *hDevID* As Integer, ByVal *axis* As Short, _ ByVal *enable* As Short, ByVal *pol* As Short, ByVal *motion* As Short, ByVal *latch* As Short) As Integer
VC#	public static uint hcp670_SetDlsSel(uint *hDevID*, ushort *axis*, ushort *enable*, ushort *pol*, ushort *motion*, ushort *latch*);

Argument	Description
hDevID	Device handle
axis	Axis designation [X:0, Y: 1, Z: 2, U: 3]
enable	DLS/PCS input switching [DLS input: 0, PCS input: 1, None used: 2]
pol	Input polarity [Normal close: 0, Normal open: 1]
motion	Operation on DLS input [Deceleration only: 0, Deceleration stop: 1]
latch	Latch on DLS input [Do not latch: 0, Latch: 1]

(10) **hcp670_SetInpos() Set INPOS**

Function	Sets processing on INPOS signal input to the designated axis controlled by the NCB specified by the device handle.

Language	Format
VC++	DWORD hcp670_SetInpos(DWORD *hDevID*, WORD *axis*, WORD *enable*, WORD *pol*);
VB	Public Function hcp670_SetInpos(ByVal *hDevID* As Integer, ByVal *axis* As Short, _ ByVal *enable* As Short, ByVal *pol* As Short) As Integer
VC#	public static uint hcp670_SetInpos(uint *hDevID*, ushort *axis*, ushort *enable*, ushort *pol*);

Argument	Description
hDevID	Device handle
axis	Axis designation [X:0, Y: 1, Z: 2, U: 3]
enable	INPOS control [OFF: 0, ON: 1]
pol	Input polarity [Normal close: 0, Normal open: 1]

(11) **hcp670_SetSvCtrCl() Set error counter clear output**

Function	Sets automatic error counter clear output for the designated axis controlled by the NCB specified by the device handle.

Language	Format
VC++	DWORD hcp670_SetSvCtrCl(DWORD *hDevID*, WORD *axis*, WORD *enable*);
VB	Public Function hcp670_SetSvCtrCl(ByVal *hDevID* As Integer, ByVal *axis* As Short, _ ByVal *enable* As Short) As Integer
VC#	public static uint hcp670_SetSvCtrCl(uint *hDevID*, ushort *axis*, ushort *enable*);

Argument	Description
hDevID	Device handle
axis	Axis designation [X:0, Y: 1, Z: 2, U: 3]
enable	Automatic output setting [Not used: 0, On homing completion: 1, On error stop: 2, On homing completion and error stop: 3]

(12) **hcp670_SetSls() Set software limit**

Function	Sets software limit for the designated axis controlled by the NCB specified by the device handle.

Language	Format
VC++	DWORD hcp670_SetSls(DWORD *hDevID*, WORD *axis*, long *psls*, long *msls*, WORD *enable*, WORD *stop*);
VB	Public Function hcp670_SetSls(ByVal *hDevID* As Integer, ByVal *axis* As Short, _ByVal *psls* As Integer, ByVal *msls* As Integer, ByVal *enable* As Short, ByVal *stop* As Short) _ As Integer
VC#	public static uint hcp670_SetSls(uint *hDevID*, ushort *axis*, int *psls*, int *msls*, ushort *enable*, ushort *stop*);

Argument	Description
hDevID	Device handle
axis	Axis designation [X:0, Y: 1, Z: 2, U: 3]
psls	+SLS [pulse]
msls	-SLS [pulse]
enable	Use/Do not use [Do not use: 0, Use: 1]
stop	Stopping method [Quick stop: 0, Deceleration stop: 1]

Remarks	1. When using software limit, make sure +SLS is larger than -SLS. 2. When not using software limit, set msls=psls=enable=0. 3. If SLS is ON at the time of writing a start command, starting in the SLS ON direction will not be possible. (The axis will not move)

(13) **hcp670_SetCmdPulse() Set command pulse output format**

Function	Sets the command pulse output format for the designated axis controlled by the NCB specified by the device handle.

Language	Format
VC++	DWORD hcp670_SetCmdPulse(DWORD *hDevID*, WORD *axis*, WORD *cmdpls*);
VB	Public Function hcp670_SetCmdPulse(ByVal *hDevID* As Integer, ByVal *axis* As Short, _ ByVal *cmdpls* As Integer) As Integer
VC#	public static uint hcp670_SetCmdPulse(uint *hDevID*, ushort *axis*, ushort *cmdpls*);

Argument	Description
hDevID	Device handle
axis	Axis designation [X:0, Y: 1, Z: 2, U: 3]
cmdpls	Command pulse output format [CW/CCW Pulse method: 0, Pulse/Direction method: 1, Output of phase difference with leading phase-A: 4, Output of phase difference with leading phase-B: 5]

(14) **hcp670_SetAccProfile() Set acceleration/deceleration type**

Function	Sets acceleration/deceleration type for the designated axis controlled by the NCB specified by the device handle.

Language	Format
VC++	DWORD hcp670_SetAccProfile(DWORD *hDevID*, WORD *axis*, WORD *pr*);
VB	Public Function hcp670_SetAccProfile(ByVal *hDevID* As Integer, ByVal *axis* As Short, _ ByVal *pr* As Short) As Integer
VC#	public static uint hcp670_SetAccProfile(uint *hDevID*, ushort *axis*, ushort *pr*);

Argument	Description
hDevID	Device handle
axis	Axis designation [X:0, Y: 1, Z: 2, U: 3]
pr	Acceleration/deceleration type [Linear acceleration/deceleration: 0, S-curve acceleration/deceleration: 1]

(15) **hcp670_SetAutoDec()**

Switch deceleration start point calculation method between auto and manual

Function	Sets whether to calculate manually or automatically the deceleration start point for the designated axis controlled by the NCB specified by the device.

Language	Format
VC++	DWORD hcp670_SetAutoDec(DWORD *hDevID*, WORD *axis*, WORD *para*);
VB	Public Function hcp670_SetAutoDec(ByVal *hDevID* As Integer, ByVal *axis* As Short, _ ByVal *para* As Short) As Integer
VC#	public static uint hcp670_SetAutoDec(uint *hDevID*, ushort *axis*, ushort *para*);

Argument	Description
hDevID	Device handle
axis	Axis designation [X:0, Y: 1, Z: 2, U: 3]
para	Deceleration start point calculation method [Auto: 0, Manual: 1]

4.7.3 Status readout

(16) **hcp670_ReadMainSts() Read main status**

Function	Reads the main status of the designated axis controlled by the NCB specified by the device handle.

Language	Format
VC++	DWORD hcp670_ReadMainSts (DWORD *hDevID*, WORD *axis*, WORD* *msts*);
VB	Public Function hcp670_ReadMainSts(ByVal *hDevID* As Integer, ByVal *axis* As Short, _ ByRef *msts* As Short) As Integer
VC#	public static uint hcp670_ReadMainSts(uint *hDevID*, ushort *axis*, ref ushort *msts*);

Argument	Description
hDevID	Device handle
axis	Axis designation [X:0, Y: 1, Z: 2, U: 3]
msts	Main status

(17) **hcp670_ReadErrorSts() Read error status**

Function	Reads the error status of the designated axis controlled by the NCB specified by the device handle.

Language	Format
VC++	DWORD hcp670_ReadErrorSts (DWORD *hDevID*, WORD *axis*, DWORD* *rest*);
VB	Public Function hcp670_ReadErrorSts(ByVal *hDevID* As Integer, ByVal *axis* As Short, _ ByRef *rest* As Integer) As Integer
VC#	public static uint hcp670_ReadErrorSts(uint *hDevID*, ushort *axis*, ref uint *rest*);

Argument	Description
hDevID	Device handle
axis	Axis designation [X:0, Y: 1, Z: 2, U: 3]
rest	Error status

(18) **hcp670_ReadEventSts() Read event status**

Function	Reads the event status of the designated axis controlled by the NCB specified by the device handle.

Language	Format
VC++	DWORD hcp670_ReadEventSts (DWORD *hDevID*, WORD *axis*, DWORD* *rist*);
VB	Public Function hcp670_ReadEventSts(ByVal *hDevID* As Integer, ByVal *axis* As Short, _ ByRef *rist* As Integer) As Integer
VC#	public static uint hcp670_ReadEventSts(uint *hDevID*, ushort *axis*, ref uint *rist*);

Argument	Description
hDevID	Device handle
axis	Axis designation [X:0, Y: 1, Z: 2, U: 3]
rist	Event status

(19) **hcp670_ReadSubSts() Read sub-status**

Function	Reads the sub-status of the designated axis controlled by the NCB specified by the device handle.

Language	Format
VC++	DWORD hcp670_ReadSubSts (DWORD *hDevID*, WORD *axis*, WORD* *ssts*);
VB	Public Function hcp670_ReadSubSts(ByVal *hDevID* As Integer, ByVal *axis* As Short, _ ByRef *ssts* As Short) As Integer
VC#	public static uint hcp670_ReadSubSts(uint *hDevID*, ushort *axis*, ref ushort *ssts*);

Argument	Description
hDevID	Device handle
axis	Axis designation [X:0, Y: 1, Z: 2, U: 3]
ssts	Sub-status

(20) **hcp670_ReadExSts()** **Read extension status**

Function	Reads the extension status of the designated axis controlled by the NCB specified by the device handle.

Language	Format
VC++	DWORD hcp670_ReadExSts (DWORD *hDevID*, WORD *axis*, DWORD* *rsts*);
VB	Public Function hcp670_ReadExSts(ByVal *hDevID* As Integer, ByVal *axis* As Short, _ ByRef *rsts* As Integer) As Integer
VC#	public static uint hcp670_ReadExSts(uint *hDevID*, ushort *axis*, ref uint *rsts*);

Argument	Description
hDevID	Device handle
axis	Axis designation [X:0, Y: 1, Z: 2, U: 3]
rsts	Extension status

(21) **hcp670_ReadSpd()** **Read command speed**

Function	Reads the command speed of the designated axis controlled by the NCB specified by the device handle.

Language	Format
VC++	DWORD hcp670_ReadSpd(DWORD *hDevID*, WORD* *axis*, WORD* *speed*);
VB	Public Function hcp670_ReadSpd(ByVal *hDevID* As Integer, ByVal *axis* As Short, ByRef *speed* As Integer) As Integer
VC#	public static uint hcp670_ReadSpd(uint *hDevID*, ushort *axis*, ref ushort *speed*);

Argument	Description
hDevID	Device handle
axis	Axis designation [X:0, Y: 1, Z: 2, U: 3]
speed	Command speed

Remarks	Command speed [pps] = Read data x Speed multiplier

(22) **hcp670_ReadCtr()** **Read counter**

Function	Reads the specified counter of the designated axis controlled by the NCB specified by the device handle.

Language	Format
VC++	DWORD hcp670_ReadCtr(DWORD *hDevID*, WORD* *axis*, WORD *selctr*, long* *count*);
VB	Public Function hcp670_ReadCtr(ByVal *hDevID* As Integer, ByVal *axis* As Short, _ ByVal *selctr* As Short, ByRef *count* As Integer) As Integer
VC#	public static uint hcp670_ReadCtr(uint *hDevID*, ushort *axis*, ushort *selctr*, ref int *count*);

Argument	Description
hDevID	Device handle
axis	Axis designation [X:0, Y: 1, Z: 2, U: 3]
selctr	Counter selector [Counter 1: 1, Counter 2: 2, Counter 3: 3, Counter 4: 4]
count	Counter value

4.7.4 Operation settings

(23) **hcp670_SetFLSpd() Set base speed**

Function	Sets the value obtained by dividing, by the speed multiplier, the base speed (pps) of the designated axis controlled by the NCB specified by the device handle.

Language	Format
VC++	DWORD hcp670_SetFLSpd (DWORD *hDevID*, WORD *axis*, DWORD *rfl*);
VB	Public Function hcp670_SetFLSpd(ByVal *hDevID* As Integer, ByVal *axis* As Short, ByVal *rfl* As Integer) As Integer
VC#	public static uint hcp670_SetFLSpd(uint *hDevID*, ushort *axis*, uint *rfl*);

Argument	Description
hDevID	Device handle
axis	Axis designation [X:0, Y: 1, Z: 2, U: 3]
rfl	Value of base speed register (RFL) [1 to 65535, and RFL<RFH]

Remarks	Base speed: Speed at which the system starts accelerating and to which decelerates to stop.

(24) **hcp670_SetAuxSpd() Set auxiliary speed**

Function	Sets the value obtained from dividing, by the speed multiplier, the auxiliary speed (pps) of the designated axis controlled by the NCB specified by the device handle.

Language	Format
VC++	DWORD hcp670_SetAuxSpd(DWORD *hDevID*, WORD *axis*, DWORD *rfa*);
VB	Public Function hcp670_SetAuxSpd(ByVal *hDevID* As Integer, ByVal *axis* As Short, ByVal *rfa* As Integer) As Integer
VC#	public static uint hcp670_SetAuxSpd(uint *hDevID*, ushort *axis*, uint *rfa*);

Argument	Description
hDevID	Device handle
axis	Axis designation [X:0, Y: 1, Z: 2, U: 3]
rfa	Value of auxiliary speed register (RFA) [1 to 65535]

Remarks	Auxiliary speed: Speed used to enter the origin and the like in certain types of homing operations.

(25) **hcp670_SetAccRate() Set acceleration rate**

Function	Sets acceleration rate (RUR) for the designated axis controlled by the NCB specified by the device handle.

Language	Format
VC++	DWORD hcp670_SetAccRate(DWORD *hDevID*, WORD *axis*, DWORD *rur*);
VB	Public Function hcp670_SetAccRate(ByVal *hDevID* As Integer, ByVal *axis* As Short, ByVal *rur* As Integer) As Integer
VC#	public static uint hcp670_SetAccRate(uint *hDevID*, ushort *axis*, uint *rur*);

Argument	Description
hDevID	Device handle
axis	Axis designation [X:0, Y: 1, Z: 2, U: 3]
rur	Acceleration rate (RUR) [1 to 65535]

Remarks	Relationship between acceleration rate (RUR) and acceleration time RFH: Operation speed register, RFL: Base speed register, RUR: Acceleration rate register (1) Linear acceleration $\text{Acceleration time [s]} = \frac{(\text{RFH} - \text{RFL}) \times (\text{RUR} + 1) \times 4}{19{,}660{,}800}$ (2) S-curve acceleration without linear acceleration range $\text{Acceleration time [s]} = \frac{(\text{RFH} - \text{RFL}) \times (\text{RUR} + 1) \times 8}{19{,}660{,}800}$ Calculate the acceleration rate with the hcp670_CalAccRate function.

(26) **hcp670_SetDecRate() Set deceleration rate**

Function	Sets deceleration rate (RDR) for the designated axis controlled by the NCB specified by the device handle. Set RDR when acceleration time and deceleration time are different.

Language	Format
VC++	DWORD hcp670_SetDecRate(DWORD *hDevID*, WORD *axis*, DWORD *rate*);
VB	Public Function hcp670_SetDecRate(ByVal *hDevID* As Integer, ByVal *axis* As Short, ByVal *rate* As Integer) As Integer
VC#	public static uint hcp670_SetDecRate(uint *hDevID*, ushort *axis*, uint *rate*);

Argument	Description
hDevID	Device handle
axis	Axis designation [X:0, Y: 1, Z: 2, U: 3]
rate	Deceleration rate (RDR) [0 to 65535] When "0" is set, deceleration rate is determined as RUR=RDR. (Acceleration time = Deceleration time)

Remarks	Relationship between deceleration rate (RDR) and deceleration time RFH: Operation speed register, RFL: Base speed register, RDR: Deceleration rate register (1) Linear deceleration $\text{Deceleration time [s]} = \frac{(\text{RFH} - \text{RFL}) \times (\text{RDR} + 1) \times 4}{19{,}660{,}800}$ (2) S-curve deceleration without linear deceleration range $\text{Deceleration time [s]} = \frac{(\text{RFH} - \text{RFL}) \times (\text{RDR} + 1) \times 8}{19{,}660{,}800}$ Calculate the deceleration rate with the hcp670_CalAccRate function.

(27) **hcp670_SetMult() Set speed multiplier register value**

Function	Sets the speed multiplier register value (RMG) for the designated axis controlled by the NCB specified by the device handle.

Language	Format
VC++	DWORD hcp670_SetMult(DWORD *hDevID*, WORD *axis*, DWORD *rmg*);
VB	Public Function hcp670_SetMult(ByVal *hDevID* As Integer, ByVal *axis* As Short, ByVal *rmg* As Integer) As Integer
VC#	public static uint hcp670_SetMult(uint *hDevID*, ushort *axis*, uint *rmg*);

Argument	Explanation
hDevID	Device handle
axis	Axis designation [X:0, Y: 1, Z: 2, U: 3]
rmg	Value of speed multiplier register [2 to 4095]

Remarks

Relationship between speed and speed multiplier, and between speed multiplier setting and speed multiplier
RFx is speed register (RFH, RFL, or RFA) value

$$\text{Speed[pps]} = \text{RFx} \times \text{Speed multiplier} = \frac{\text{Rfx} \times 300}{\text{RMG} + 1} \qquad \text{Speed multiplier setting} = \frac{300}{\text{Speed multiplier}} - 1$$

Setting example

RMG(DEC)	RMG(HEX)	Speed multiplier	Output speed range (pps)
2999	bb7	0.1	0.1 to 6,553.5
1499	5db	0.2	0.2 to 13,107
599	257	0.5	0.5 to 32,767.5
299	12b	1	1 to 65,535
149	95	2	2 to 131,070
59	3b	5	5 to 327,675
29	1d	10	10 to 655,350
14	e	20	20 to 1,310,700
11	b	25	25 to 1,638,375
9	9	30	30 to 1,966,050
5	5	50	50 to 3,276,750
4	4	60	60 to 3,932,100
3	3	75	75 to 4,915,125
2	2	100	100 to 6,553,500

(28) **hcp670_SetEventMask() Set event mask**

Function	Sets event mask for the designated axis controlled by the NCB specified by the device handle.

Language	Format
VC++	DWORD hcp670_SetEventMask(DWORD *hDevID*, WORD *axis*, DWORD *mask*);
VB	Public Function hcp670_SetEventMask(ByVal *hDevID* As Integer, ByVal *axis* As Short, _ ByVal *mask* As Integer) As Integer
VC#	public static uint hcp670_SetEventMask(uint *hDevID*, ushort *axis*, uint *mask*);

Argument	Description
hDevID	Device handle
axis	Axis designation [X:0, Y: 1, Z: 2, U: 3]
mask	Event mask data 00001h On normal stop 00002h On continuous execution start 00004h On operation preregister availability 00008h On comparator 5 preregister availability 00010h On acceleration start 00020h On acceleration end 00040h On deceleration start 00080h On deceleration end 00100h On comparator 1 condition match 00200h On comparator 2 condition match 00400h On comparator 3 condition match 00800h On comparator 4 condition match 01000h On comparator 5 condition match 02000h On counter clear by CLR input 04000h On counter value latch by LTCH input 08000h On counter value latch by OLS input 10000h On DLS input ON 20000h On ±DR input change 40000h On CSTA signal input ON Notification of multiple events is possible by logically adding the above-described data.

(29) **hcp670_SetDecPoint() Set deceleration start point**

Function	Sets deceleration start point for the designated axis controlled by the NCB specified by the device handle.

Language	
VC++	DWORD hcp670_SetDecPoint(DWORD *hDevID*, WORD *axis*, long *distance*);
VB	Public Function hcp670_SetDecPoint (ByVal *hDevID* As Integer, ByVal *axis* As Short, _ ByVal *distance* As Integer) As Integer
VC#	public static uint hcp670_SetDecPoint (uint *hDevID*, ushort *axis*, int *distance*);

Argument	Description
hDevID	Device handle
axis	Axis designation [X:0, Y: 1, Z: 2, U: 3]
distance	Deceleration start point (pulse)

Remarks	The deceleration starts when the remaining travel distance becomes less than the deceleration start point. **[Automatic calculation of deceleration start point]** The value set by this function is an offset (unit: pulses) to the automatically calculated deceleration start point. Setting a positive value makes the deceleration start earlier. The operation continues at base speed after the deceleration ends. Setting a negative value makes the deceleration start later, making the operation complete before reaching the base speed. Setting "0" makes the deceleration start at the automatically calculated point. **[Manual calculation of deceleration start point]** The deceleration starts when the remaining travel distance becomes less than the set value (in pulses). Calculate the value to set by using the following expressions (value to complete the operation on reaching the base speed): (1) Linear deceleration $\text{Optimum value [pulse]} = \frac{(RFH^2 - RFL^2) \times (RDR + 1)}{(RMG + 1) \times 32767}$ (2) S-curve deceleration without linear deceleration range $\text{Optimum value [pulse]} = \frac{(RFH^2 - RFL^2) \times (RDR + 1) \times 2}{(RMG + 1) \times 32767}$

4.7.5 Operational settings

(30) **hcp670_WritOpeMode() Set operation mode**

Function	Sets operation mode for the designated axis controlled by the NCB specified by the device handle.

Language	Format
VC++	DWORD hcp670_WritOpeMode(DWORD *hDevID*, WORD *axis*, WORD *mode*);
VB	Public Function hcp670_WritOpeMode(ByVal *hDevID* As Integer, ByVal *axis* As Short, _ ByVal *mode* As Short) As Integer
VC#	public static uint hcp670_WritOpeMode(uint *hDevID*, ushort *axis*, ushort *mode*);

Argument	Description	
hDevID	Device handle	
axis	Axis designation [X:0, Y: 1, Z: 2, U: 3]	
mode	Operation mode	
	00h: Continuous operation in the positive direction by command control	08h: Continuous operation in the negative direction by command control
	01h: Continuous operation by pulsar input (*1)	02h: Continuous operation by ±DR input (Optional)
	10h: Homing operation in the positive direction	18h: Homing operation in the negative direction
	12h: Origin escape in the positive direction	1ah: Origin escape in the negative direction
	15h: Origin search in the positive direction	1dh: Origin search in the negative direction
	20h: Move to +ELS or +SLS position	28h: Move to -ELS or -SLS position
	22h: Escape from +ELS or +SLS	2ah: Escape from -ELS or -SLS
	24h: Move in positive direction for number of phase-Z counts	2ch: Move in negative direction for number of phase-Z counts
	41h: Positioning operation	42h: PCS positioning operation (library function only)
	44h: Homing operation to command position 0	45h: Homing operation to machine position 0
	46h: 1-pulse operation in the positive direction	4eh: 1-pulse operation in the negative direction
	47h: Timer operation	51h: Positioning operation by pulsar input
	54h: Homing operation to command position 0 by pulsar input	55h: Homing operation to machine position 0 by pulsar input
	56h: Positioning operation by ±DR input (Optional)	
	60h: Continuous feed by linear interpolation operation	61h: Linear Interpolation operation
	64h: CW circular interpolation operation	65h: CCW circular interpolation operation

(31) **hcp670_WritFHSpd() Set operation speed**

Function	Sets the value (RFH) obtained by dividing, by the speed multiplier, the operation speed (pps) of the designated axis controlled by the NCB specified by the device handle.

Language	Format
VC++	DWORD hcp670_WritFHSpd(DWORD *hDevID*, WORD *axis*, DWORD *rfh*);
VB	Public Function hcp670_WritFHSpd(ByVal *hDevID* As Integer, ByVal *axis* As Short, ByVal *rfh* As Integer) As Integer
VC#	public static uint hcp670_WritFHSpd(uint *hDevID*, ushort *axis*, uint *rfh*);

Argument	Description
hDevID	Device handle
axis	Axis designation [X:0, Y: 1, Z: 2, U: 3]
rfh	Operation speed register value (RFH) [2 to 65535, and RFL<RFH]

(32) **hcp670_WritPos() Set positioning travel distance**

Function	Sets travel distance for positioning for the designated axis controlled by the NCB specified by the device handle.

Language	Format
VC++	DWORD hcp670_WritPos(DWORD *hDevID*, WORD *axis*, long *distance*);
VB	Public Function hcp670_WritPos(ByVal *hDevID* As Integer, ByVal *axis* As Short, ByVal *distance* As Integer) As Integer
VC#	public static uint hcp670_WritPos(uint *hDevID*, ushort *axis*, int *distance*);

Argument	Description
hDevID	Device handle
axis	Axis designation [X:0, Y: 1, Z: 2, U: 3]
distance	Travel distance (pulse) [-134,217,728 to 134,217,727 (28 bits)]

(33) **hcp670_WritLine() Set linear interpolation travel distance**

Function	Sets travel distance for the linear interpolation of the designated axes controlled by the NCB specified by the device handle.

Language	Format
VC++	DWORD hcp670_WritLine(DWORD *hDevID*, WORD *axis*, long *distance*);
VB	Public Function hcp670_WritLine(ByVal *hDevID* As Integer, ByVal *axis* As Short, ByVal *distance* As Integer) As Integer
VC#	public static uint hcp670_WritLine(uint *hDevID*, ushort *axis*, int *distance*);

Argument	Description
hDevID	Device handle
axis	Axis designation [X:0, Y: 1, Z: 2, U: 3]
distance	Travel distance (pulse) [-134,217,728 to 134,217,727 (28 bits)]

(34) **hcp670_WritCircl() Set circular interpolation travel distance**

Function	Sets travel distance for the circular interpolation of the designated axes controlled by the NCB specified by the device handle. Specified points are interpreted as end points and center points 1 and 2 in the order of axis proximity to the X-axis. For information on how to set data, refer to “User’s Manual <Operation>”.

Language	Format
VC++	DWORD hcp670_WritCircl(DWORD *hDevID*, WORD *axis*, long *dstnc1*,long *dstnc2*,long *center1*,long *center2*);
VB	Public Function hcp670_WritCircl(_ ByVal *hDevID* As Integer, ByVal *axis* As Short, ByVal *dstnc1* As Integer, ByVal *dstnc2* As Integer, _ ByVal *center1* As Integer, ByVal *center2* As Integer) As Integer
VC#	public static uint hcp670_WritCircl(uint *hDevID*, ushort *axis,* int *dstnc1*, int *dstnc2*, int *center1*, int *center2*);

Argument	Description
hDevID	Device handle
axis	Axis designation [X-Y: 0, X-Z: 1, X-U: 2, Y-Z: 3, Y-U: 4, Z-U: 5]
dstnc1	End point 1 [-134,217,728 to 134,217,727 (28 bits)]
dstnc2	End point 2 [-134,217,728 to 134,217,727 (28 bits)]
center1	Center point 1 [-134,217,728 to 134,217,727 (28 bits)]
center2	Center point 2 [-134,217,728 to 134,217,727 (28 bits)]

(35) **hcp670_WritCtr() Preset counter**

Function	Presets (writes coordinate values) to the specified counter of the designated axis controlled by the NCB specified by the device handle.

Language	Format
VC++	DWORD hcp670_WritCtr(DWORD *hDevID*, WORD *axis,* long *preset*, WORD *selctr*);
VB	Public Function hcp670_WritCtr(ByVal *hDevID* As Integer, ByVal *axis* As Short, _ ByVal *preset* As Integer, ByVal *selctr* As Integer) As Integer
VC#	public static uint hcp670_WritCtr(uint *hDevID*, ushort *axis*, int nData, ushort *selctr*);

Argument	Description
hDevID	Device handle
axis	Axis designation [X:0, Y: 1, Z: 2, U: 3]
preset	Preset value [-134,217,728 to 134,217,727 (28 bits)]
selctr	Counter selector [Counter 1: 1, Counter 2: 2, Counter 3: 3, Counter 4: 4]

4.7.6 Operation control commands

Operation control command argument explanation

hDevID: Device handle

axis: Axis designation. To designate multiple axes simultaneously, logically add the target axis designations. [X:1, Y: 2, Z: 4, U: 8]

(36) **hcp670_DecStop() Deceleration stop**

Function	Decelerates and stops the designated axis controlled by the NCB specified by the device handle.

Language	Format
VC++	DWORD hcp670_DecStop (DWORD *hDevID*, WORD *axis*);
VB	Public Function hcp670_DecStop(ByVal *hDevID* As Integer, ByVal *axis* As Short) As Integer
VC#	public static uint hcp670_DecStop(uint *hDevID*, ushort *axis*);

(37) **hcp670_QuickStop() Quick stop**

Function	Quickly stops the designated axis controlled by the NCB specified by the device handle.

Language	Format
VC++	DWORD hcp670_QuickStop(DWORD *hDevID*, WORD *axis*);
VB	Public Function hcp670_QuickStop(ByVal *hDevID* As Integer, ByVal *axis* As Short) As Integer
VC#	public static uint hcp670_QuickStop(uint *hDevID*, ushort *axis*);

(38) **hcp670_EmgStop() Emergency stop**

Function	Emergency-stops the designated axis controlled by the NCB specified by the device handle.

Language	Format
VC++	DWORD hcp670_EmgStop (DWORD *hDevID*, WORD *axis*);
VB	Public Function hcp670_EmgStop(ByVal *hDevID* As Integer, ByVal *axis* As Short) As Integer
VC#	public static uint hcp670_EmgStop(uint *hDevID*, ushort *axis*);

(39) **hcp670_AccStart() Acceleration start**

Function	Acceleration-starts the designated axis controlled by the NCB specified by the device handle.

Language	Format
VC++	DWORD hcp670_AccStart (DWORD *hDevID*, WORD *axis*);
VB	Public Function hcp670_AccStart (ByVal *hDevID* As Integer, ByVal *axis* As Short) As Integer
VC#	public static uint hcp670_AccStart (uint *hDevID*, ushort *axis*);

(40) **hcp670_CnstStartFH() FH constant-speed start**

Function	Starts at FH constant speed for the designated axis controlled by the NCB specified by the device handle.

Language	Format
VC++	DWORD hcp670_CnstStartFH (DWORD *hDevID*, WORD *axis*);
VB	Public Function hcp670_CnstStartFH (ByVal *hDevID* As Integer, ByVal *axis* As Short) As Integer
VC#	public static uint hcp670_CnstStartFH (uint *hDevID*, ushort *axis*);

(41) **hcp670_CnstStartFL() FL constant-speed start**

Function	Starts at FL constant speed for the designated axis controlled by the NCB specified by the device handle.

Language	Format
VC++	DWORD hcp670_CnstStartFL (DWORD *hDevID*, WORD *axis*);
VB	Public Function hcp670_CnstStartFL(ByVal *hDevID* As Integer, ByVal *axis* As Short) As Integer
VC#	public static uint hcp670_CnstStartFL (uint *hDevID*, ushort *axis*);

(42) **hcp670_CnstStartByDec() Deceleration stop after FH constant-speed start**

Function	Starts at FH constant speed and then decelerates and stops the designated axis controlled by the NCB specified by the device handle.

Language	Format
VC++	DWORD hcp670_CnstStartByDec(DWORD *hDevID*, WORD *axis*);
VB	Public Function hcp670_CnstStartByDec (ByVal *hDevID* As Integer, ByVal *axis* As Short) As Integer
VC#	public static uint hcp670_CnstStartByDec (uint *hDevID*, ushort *axis*);

(43) **hcp670_SvOn() Servo ON**

Function	Turns ON the SVON signal for the designated axis controlled by the NCB specified by the device handle.

Language	Format
VC++	DWORD hcp670_SvOn (DWORD *hDevID*, WORD *axis*);
VB	Public Function hcp670_SvOn (ByVal *hDevID* As Integer, ByVal *axis* As Short) As Integer
VC#	public static uint hcp670_SvOn (uint *hDevID*, ushort *axis*);

(44) **hcp670_SvOff() Servo OFF**

Function	Turns OFF the SVON signal for the designated axis controlled by the NCB specified by the device handle.

Language	Format
VC++	DWORD hcp670_SvOff (DWORD *hDevID*, WORD *axis*);
VB	Public Function hcp670_SvOff (ByVal *hDevID* As Integer, ByVal *axis* As Short) As Integer
VC#	public static uint hcp670_SvOff (uint *hDevID*, ushort *axis*);

(45) **hcp670_SvResetOn() Servo reset ON**

Function	Turns ON the SVRST signal for the designated axis controlled by the NCB specified by the device handle.

Language	Format
VC++	DWORD hcp670_SvResetOn (DWORD *hDevID*, WORD *axis*);
VB	Public Function hcp670_SvResetOn (ByVal *hDevID* As Integer, ByVal *axis* As Short) As Integer
VC#	public static uint hcp670_SvResetOn (uint *hDevID*, ushort *axis*);

(46) **hcp670_SvResetOff() Servo reset OFF**

Function	Turns OFF the SVRST signal for the designated axis controlled by the NCB specified by the device handle.

Language	Format
VC++	DWORD hcp670_SvResetOff(DWORD *hDevID*, WORD *axis*);
VB	Public Function hcp670_SvResetOff (ByVal *hDevID* As Integer, ByVal *axis* As Short) As Integer
VC#	public static uint hcp670_SvResetOff (uint *hDevID*, ushort *axis*);

(47) **hcp670_PMOn() Pulse motor excitation ON**

Function	Turns ON the pulse motor excitation for the designated axis controlled by the NCB specified by the device handle.

Language	Format
VC++	DWORD hcp670_PMOn (DWORD *hDevID*, WORD *axis*);
VB	Public Function hcp670_PMOn (ByVal *hDevID* As Integer, ByVal *axis* As Short) As Integer
VC#	public static uint hcp670_PMOn (uint *hDevID*, ushort *axis*);

Remarks	This function is available when SVON is connected to the motor-free (output current OFF) signal of the pulse motor driver.

(48) **hcp670_PMOff() Pulse motor excitation OFF**

Function	Turns OFF the pulse motor excitation (motor-free) for the designated axis controlled by the NCB specified by the device handle.

Language	Format
VC++	DWORD hcp670_PMOff (DWORD *hDevID*, WORD *axis*);
VB	Public Function hcp670_PMOff (ByVal *hDevID* As Integer, ByVal *axis* As Short) As Integer
VC#	public static uint hcp670_PMOff (uint *hDevID*, ushort *axis*);

Remarks	This function is available when SVON is connected to the motor-free (output current OFF) signal of the pulse motor driver.

4.7.7 DPRAM readout (NCB-specific)

(49) **hcp670_ReadErno () Read error number (DPERRNO)**

Function	Reads DP_ERNO for the NCB specified by the device handle when an exception occurs in MDA control or CMP control.

Language	Format
VC++	DWORD hcp670_ReadErno (DWORD *hDevID,* WORD* *erno*);
VB	Public Function hcp670_ReadErno (ByVal *hDevID* As Integer, ByRef *erno* As Short) As Integer
VC#	public static uint hcp670_ReadErno (uint *hDevID,* ref ushort *erno*);

Argument	Description
hDevID	Device handle
erno	Error number (DPERRNO)

Remarks	For details on the contents of DPERRNO, see "9.4.3 DPRAM content details (4) Error numbers (DPERRNO)".

(50) **hcp670_ReadDpFC () Read counter/speed via DPRAM**

Function	Use to read the counter or speed of axes executing MDA or CMP.

Language	Format
VC++	DWORD hcp670_ReadDpFC (DWORD *hDevID,* WORD *sel,* long* *ctr1,* long* *ctr2,* WORD* *feed*);
VB	Public Function hcp670_ReadDpFC (ByVal *hDevID* As Integer, ByVal *sel* As Short _ ByRef *ctr1()* As Integer, ByRef *ctr2()* As Integer, ByRef *feed()* As Short) As Integer
VC#	public static uint hcp670_ReadDpFc (uint *hDevID,* ushort *sel,* ref int[] *ctr1,* ref int[] *ctr2,* ref ushort[] *feed*);

Argument	Description
hDevID	Device handle
sel	Designation of data to read XCTR1: 0001h, YCTR1: 0002h, ZCTR1: 0004h, UCTR1: 0008h, XCTR2: 0010h, YCTR2: 0020h, ZCTR2: 0040h, UCTR2: 0080h, XRSPD: 0100h, YRSPD: 0200h, ZRSPD: 0400h, URSPD: 0800h
ctr1	CTR1 (pointer to the top of the four-axis array)
ctr2	CTR2 (pointer to the top of the four-axis array)
feed	Command speed (pointer to the top of the four-axis array)

Remarks	Reading during input/output buffer control returns ERROR_DP_BUFBUSY (80000004h) instead of counter and speed values.

(51) **hcp670_ReadDpReg () Read register via DPRAM**

Function	Use to read the registers for axes executing MDA or CMP.

Language	Format
VC++	DWORD hcp670_ReadDpReg (DWORD *hDevID,* WORD *cmd,* DWORD* *reg*);
VB	Public Function hcp670_ReadDpReg (ByVal *hDevID* As Integer, ByVal *cmd* As Short _ ByRef *reg()* As Integer) As Integer
VC#	public static uint hcp670_ReadDpReg (uint *hDevID,* ushort *cmd,* ref int[] *reg*);

Argument	Description
hDevID	Device handle
cmd	Designation of data to read Low byte: register read command High byte: axis designation [X:01, Y: 02, Z: 04, U: 08] For example, specify 01F1 to read XRSTS, or 06F1 to read YRSTS and ZRSTS.
reg	Register data (pointer to the top of the four-axis array)

Remarks	Reading during input/output buffer control returns ERROR_DP_BUFBUSY (80000004h) instead of register values.

4.7.8 Calculation function

(52) **hcp670_CalAccRate() Calculate acceleration/deceleration rate**

Function	Calculates acceleration rate (RUR) or deceleration rate (RDR) from acceleration or deceleration time, RFH, RFL, and the like.

Language	Format
VC++	DWORD hcp670_CalAccRate(DWORD* *r*, DWORD *t*, DWORD *fh*, DWORD *fl*, WORD *p*, WORD *s*);
VB	Public Function hcp670_CalAccRate(ByRef *r* As Integer, ByVal *t* As Integer, ByVal *fh* As Integer, _ ByVal *fl* As Integer, ByVal *p* As Short, ByVal *s* As Short) As Integer
VC#	public static uint hcp670_CalAccRate(ref uint *r*, uint *t*, uint *fh*, uint *fl*, ushort *p*, ushort *s*);

Argument	Description
r	Calculation result (Acceleration or deceleration rate [1 to 65535])
t	Acceleration or deceleration time (ms)
fh	FH register value [1 to 65535]
fl	FL register value [1 to 65535]
p	Acceleration/deceleration type [Linear: 0, S-curve: 1]
s	Speed in the S-curve range [1 to 32767]

Remarks	Set the acceleration rate calculated with this function using the hcp670_SetAccRate function, and the hcp670_SetDecRate function for the deceleration rate.

4.8 Library Function Details (NCB Series)

The following functions monitor DPINTB by using the interrupt method.

Beware that these functions cannot be used simultaneously with CPD-series compliant library functions for MDA control and CMP control.

4.8.1 Event monitoring functions

(53) **ncb670_StartEvent() Start event monitoring**

Function	Starts event monitoring for the NCB specified by the device handle. NCB-series library functions are enabled after executing this function.

Language	Format
VC++	DWORD ncb670_StartEvent (DWORD *hDevID*);
VB	Public Function ncb670_StartEvent (ByVal *hDevID* As Integer) As Integer
VC#	public static uint ncb670_StartEvent (uint *hDevID*);

Argument	Description
hDevID	Device handle

(54) **ncb670_StopEvent() Stop event monitoring**

Function	Stops the event monitoring for the NCB specified by the device handle. NCB-series library functions are disabled after executing this function.

Language	Format
VC++	DWORD ncb670_StopEvent (DWORD *hDevID*);
VB	Public Function ncb670_StopEvent (ByVal *hDevID* As Integer) As Integer
VC#	public static uint ncb670_StopEvent (uint *hDevID*);

Argument	Description
hDevID	Device handle

(55) **ncb670_IsAck () Check ACK/NAK following MDA/CDA/CMP operation command**

Function	Checks ACK/NAK from the NCB specified by the device handle.

Language	Format
VC++	DWORD ncb670_IsAck (DWORD *hDevID,* WORD* *usErrorNum*);
VB	Public Function ncb670_IsAck (ByVal *hDevID* As Integer, ByRef *usErrorNum* As Short) As Integer
VC#	public static uint ncb670_IsAck (uint *hDevID*, ref ushort *usErrorNum*);

Argument	Description
hDevID	Device handle
usErrorNum	Error number

(56) **ncb670_WaitMdaLend () Check MDA load completion**

Function	Checks completion of MDA load to the NCB specified by the device handle.

Language	Format
VC++	DWORD ncb670_WaitMdaLend(DWORD hDevID, WORD* usErrorNum);
VB	Public Function ncb670_WaitMdaLend(ByVal hDevID As Integer, ByRef usErrorNum As Short) _ As Integer
VC#	public static uint ncb670_WaitMdaLend(uint hDevID, ref ushort usErrorNum);

Argument	Description
hDevID	Device handle
usErrorNum	Error number

(57) **ncb670_WaitMdaEnd () Check MDA execution completion**

Function	Checks completion of MDA execution on the NCB specified by the device handle.

Language	Format
VC++	DWORD ncb670_ WaitMdaEnd (DWORD *hDevID,* DWORD *nTimeOut,* WORD* *usErrorNum*);
VB	Public Function ncb670_ WaitMdaEnd (ByVal *hDevID* As Integer, ByVal *nTimeOut* As Integer, _ ByRef *usErrorNum* As Short) As Integer
VC#	public static uint ncb670_WaitMdaEnd (uint *hDevID*, int *nTimeOut*, ref ushort *usErrorNum*);

Argument	Description
hDevID	Device handle
nTimeOut	Specify timeout time in units of milliseconds (ms). Specifying INFINITE(-1) makes the function wait until MDA execution completes, or until ncb670_StopEvent is called.
usErrorNum	Error number

(58) **ncb670_WaitCdaLend () Check CDA load completion**

Function	Checks completion of CDA load to the NCB specified by the device handle.

Language	Format
VC++	DWORD ncb670_ WaitCdaLend (DWORD *hDevID,* WORD* *usErrorNum*);
VB	Public Function ncb670_ WaitCdaLend (ByVal *hDevID* As Integer, ByRef *usErrorNum* As Short) _ As Integer
VC#	public static uint ncb670_WaitCdaLend (uint *hDevID*, ref ushort *usErrorNum*);

Argument	Description
hDevID	Device handle
usErrorNum	Error number

(59) **ncb670_WaitCdaCLend Check continuous CDA load completion**

Function	Checks completion of continuous CDA load to the NCB specified by the device handle.

Language	Format
VC++	DWORD ncb670_ WaitCdaCLend (DWORD *hDevID,* WORD* *usErrorNum*);
VB	Public Function ncb670_ WaitCdaCLend (ByVal *hDevID* As Integer, ByRef *usErrorNum* As Short) _ As Integer
VC#	public static uint ncb670_WaitCdaCLend (uint *hDevID*, ref ushort *usErrorNum*);

Argument	Description
hDevID	Device handle
usErrorNum	Error number

(60) **ncb670_WaitCrq () Check request for continuous CDA execution**

Function	Waits for a request for continuous CDA execution for the NCB specified by the device handle.

Language	Format
VC++	DWORD ncb670_ WaitCrq(DWORD *hDevID,* DWORD *nTimeOut,* WORD* *usErrorNum*);
VB	Public Function ncb670_ WaitCrq (ByVal *hDevID* As Integer, ByVal *nTimeOut* As Integer, _ ByRef *usErrorNum* As Short) As Integer
VC#	public static uint ncb670_WaitCrq (uint *hDevID*, int *nTimeOut*, ref ushort *usErrorNum*);

Argument	Description
hDevID	Device handle
nTimeOut	Specify the timeout time in units of milliseconds (ms) to wait for a request for continuous CDA execution. Specifying INFINITE(-1) makes the function wait until there is a request for continuous CDA execution, CDA execution is stopped, or until ncb670_StopEvent is called.
usErrorNum	Error number

(61) **ncb670_WaitCdaEnd () Check completion of CDA execution**

Function	Checks completion of CDA execution on the NCB specified by the device handle.

Language	Format
VC++	DWORD ncb670_ WaitCdaEnd (DWORD *hDevID,* DWORD *nTimeOut,* WORD* *usErrorNum*);
VB	Public Function ncb670_ WaitCdaEnd (ByVal *hDevID* As Integer, ByVal *nTimeOut* As Integer, _ ByRef *usErrorNum* As Short) As Integer
VC#	public static uint ncb670_WaitCdaEnd (uint *hDevID*, int *nTimeOut*, ref ushort *usErrorNum*);

Argument	Description
hDevID	Device handle
nTimeOut	Specify the timeout time in units of milliseconds (ms) to wait for CDA execution to finish. Specifying INFINITE(-1) makes the function wait until CDA execution finishes, or until ncb670_CdaHalt or ncb670_StopEvent is called.
usErrorNum	Error number

(62) **ncb670_WaitCmpLend () Check CMP load completion**

Function	After executing CMP load to the NCB specified by the device handle, waits for the CMP load to complete.

Language	Format
VC++	DWORD ncb670_ WaitCmpLend (DWORD *hDevID,* WORD* *usErrorNum*);
VB	Public Function ncb670_ WaitCmpLend (ByVal *hDevID* As Integer, ByRef *usErrorNum* As Short) _ As Integer
VC#	public static uint ncb670_WaitCmpLend (uint *hDevID*, ref ushort *usErrorNum*);

Argument	Description
hDevID	Device handle
usErrorNum	Error number

(63) **ncb670_WaitCmpEnd () Check CMP execution completion**

Function	Waits for completion of CMP execution on the NCB specified by the device handle.

Language	Format
VC++	DWORD ncb670_ WaitCmpEnd (DWORD *hDevID,* DWORD *nTimeOut,* WORD* *usErrorNum*);
VB	Public Function ncb670_ WaitCmpEnd (ByVal *hDevID* As Integer, ByVal *nTimeOut* As Integer, _ ByRef *usErrorNum* As Short) As Integer
VC#	public static uint ncb670_WaitCmpEnd (uint *hDevID*, int *nTimeOut*, ref ushort *usErrorNum*);

Argument	Description
hDevID	Device handle
nTimeOut	Specify the timeout time in units of milliseconds (ms) for CMP execution to complete. Specifying INFINITE(-1) makes the function wait until CMP execution is complete, or until ncb670_StopEvent is called.
usErrorNum	Error number

4.8.2 MDA control functions

(64) **ncb670_NcbLoad () Load operation data**

Function	Loads operation data to the NCB specified by the device handle. This function can also be used within ncb670_CdaExec.

Language	Format
VC++	DWORD ncb670_NcbLoad (DWORD *hDevID,* WORD *usCmd,* WORD *usLength,* CDA_DATA* *mdaStruct,* WORD* *usErrorNum*);
VB	Public Function ncb670_NcbLoad(ByVal *hDevID* As Integer, _ ByVal *usCmd* As Short, ByVal *usLength* As Short, ByRef *mdaStruct()* As CDA_DATA, _ ByRef *usErrorNum* As Short) As Integer
VC#	public static uint ncb670_NcbLoad(uint *hDevID*, ushort *usCmd*, ushort *usLength*, ref NcbLib1a.CDA_DATA[] *mdaStruct*, ref ushort *usErrorNum*);

Argument	Description
hDevID	Device handle
usCmd	INTB command
usLength	Number of data
mdaStruct	Operation data structure array
usErrorNum	Error number

Remarks	[Operation data structure (VC#)]
	public struct CDA_DATA
	{
	public uint seq; // Sequence Number
	public ushort dt1; // data1
	public ushort dt2; // data2
	public ushort dt3; // data3
	}
	For details on the contents of the operation data, see "5. NCB Mode (Automatic Program Execution Function: MDA)".

(65) **ncb670_MdaExec () Start MDA execution**

Function	Starts executing MDA on the NCB specified by the device handle. Call this function after loading MDA data with ncb670_NcnLoad. Then, wait for MDA execution to complete by using ncb670_WaitMdaEnd function.

Language	Format
VC++	DWORD ncb670_ MdaExec (DWORD *hDevID,* WORD* *usErrorNum*);
VB	Public Function ncb670_ MdaExec (ByVal *hDevID* As Integer, ByRef *usErrorNum* As Short) _ As Integer
VC#	public static uint ncb670_MdaExec (uint *hDevID,* ref ushort *usErrorNum*);

Argument	Description
hDevID	Device handle
usErrorNum	Error number

(66) **ncb670_MdaStepExec () Execute MDA step-by-step**

Function	Starts step-by-step execution of MDA on the NCB specified by the device handle.

Language	Format
VC++	DWORD ncb670_ MdaStepExec (DWORD *hDevID,* WORD* *usErrorNum*);
VB	Public Function ncb670_ MdaStepExec (ByVal *hDevID* As Integer, ByRef *usErrorNum* As Short) _ As Integer
VC#	public static uint ncb670_MdaStepExec (uint *hDevID,* ref ushort *usErrorNum*);

Argument	Description
hDevID	Device handle
usErrorNum	Error number

(67) **ncb670_MdaBreakExec () Break MDA execution**

Function	Breaks the execution of MDA on the NCB specified by the device handle.

Language	Format
VC++	DWORD ncb670_ MdaBreakExec (DWORD *hDevID,* WORD* *usErrorNum*);
VB	Public Function ncb670_ MdaBreakExec (ByVal *hDevID* As Integer, ByRef *usErrorNum* As Short) _ As Integer
VC#	public static uint ncb670_MdaBreakExec (uint *hDevID,* ushort *brpno,* ref ushort *usErrorNum*);

Argument	Description
hDevID	Device handle
brpno	Break execution line number. The function stops the MDA execution at the line specified here.
usErrorNum	Error number

(68) **ncb670_MdaCStop () Stop MDA execution cycle**

Function	Stops the cycle of MDA execution on the NCB specified by the device handle. Executing MDA after stopping the cycle resumes the MDA execution from the point where it was stopped.

Language	Format
VC++	DWORD ncb670_MdaCStop (DWORD *hDevID,* WORD* *usErrorNum*);
VB	Public Function ncb670_MdaCStop (ByVal *hDevID* As Integer, ByRef *usErrorNum* As Short) _ As Integer
VC#	public static uint ncb670_MdaCStop (uint *hDevID,* ref ushort *usErrorNum*);

Argument	Description
hDevID	Device handle
usErrorNum	Error number

(69) **ncb670_MdaQStop () Quick-stop MDA execution**

Function	Quickly stops the execution of MDA on the NCB specified by the device handle. It also cancels the MDA data.

Language	Format
VC++	DWORD ncb670_MdaQStop (DWORD *hDevID,* WORD* *usErrorNum*);
VB	Public Function ncb670_MdaQStop (ByVal *hDevID* As Integer, ByRef *usErrorNum* As Short) _ As Integer
VC#	public static uint ncb670_MdaQStop (uint *hDevID,* ref ushort *usErrorNum*);

Argument	Description
hDevID	Device handle
usErrorNum	Error number

4.8.3 CDA control functions

(70) **ncb670_CdaExec () Start CDA execution**

Function	Starts executing CDA on the NCB specified by the device handle. CDA data is described as a two-dimensional array of page numbers and sequence numbers, and executed by using this function after loading. Completion or temporary stop of the execution can be monitored by using the ncb670_WaitCdaEnd function.

Language	Format
VC++	DWORD ncb670_CdaExec(DWORD *hDevID* DWORD *unLength,* DWORD *nTimeOut,* CDA_DATA* *cdaStruct,* WORD* *usErrorNum*);
VB	Public Function ncb670_CdaExec (ByVal *hDevID* As Integer, ByVal *unLength* As Integer, _ ByVal nTimeOut As Integer, ByRef *cdaStruct(,)* As CDA_DATA, ByRef *usErrorNum* As Short) As Integer
VC#	public static uint ncb670_CdaExec(uint *hDevID*, uint *unLength*, int *nTimeOut*, ref NcbLib1a.CDA_DATA[,] *cdaStruct*, ref ushort *usErrorNum*);

Argument	Description
hDevID	Device handle
unLength	Total number of data
nTimeOut	Specify the timeout time, in units of milliseconds (ms), until a request for continuous CDA execution (4000 lines for the first operation, and 2000 lines or completion of n lines thereafter).
cdaStruct	Two-dimensional (page number, sequence number) array data structure
usErrorNum	Error number

(71) **ncb670_CdaHalt () Stop CDA execution temporarily**

Function	Temporarily stops the execution of CDA on the NCB specified by the device handle. Calling the ncb670_CdaExec function again after the temporary stop resumes the CDA execution from the point where it was stopped. Calling the ncb670_CdaCancel function after the temporary stop cancels the data and makes continued execution impossible.

Language	Format
VC++	DWORD ncb670_CdaHalt(DWORD *hDevID,* WORD* *usErrorNum*);
VB	Public Function ncb670_CdaHalt (ByVal *hDevID* As Integer, ByRef *usErrorNum* As Short) _ As Integer
VC#	public static uint ncb670_CdaHalt(uint *hDevID*, ref ushort *usErrorNum*);

Argument	Description
hDevID	Device handle
usErrorNum	Error number

(72) **ncb670_CdaQStop () Quick-stop CDA execution**

Function	Quickly stops the execution of CDA on the NCB specified by the device handle. It also cancels the CDA data.

Language	Format
VC++	DWORD ncb670_CdaQStop (DWORD *hDevID,* WORD* *usErrorNum*);
VB	Public Function ncb670_CdaQStop (ByVal *hDevID* As Integer, ByRef *usErrorNum* As Short) _ As Integer
VC#	public static uint ncb670_CdaQStop(uint *hDevID*, ref ushort *usErrorNum*);

Argument	Description
hDevID	Device handle
usErrorNum	Error number

(73) **ncb670_CdaCancel () Cancel MDA/CDA data**

Function	Cancels the operation data (MDA or CDA).

Language	Format
VC++	DWORD ncb670_CdaCancel (DWORD *hDevID,* WORD* *usErrorNum*);
VB	Public Function ncb670_CdaCancel (ByVal *hDevID* As Integer, ByRef *usErrorNum* As Short) _ As Integer
VC#	public static uint ncb670_CdaCancel(uint *hDevID*, ref ushort *usErrorNum*);

Argument	Description
hDevID	Device handle
usErrorNum	Error number

4.8.4 CMP control functions

(74) **ncb670_CmpLoad () Load CMP data**

Function	Loads CMP data to the NCB specified by the device handle.

Language	Format
VC++	DWORD ncb670_CmpLoad (DWORD *hDevID,* WORD *length,* long* *cmp,* WORD* *usErrorNum*);
VB	Public Function ncb670_CmpLoad(ByVal *hDevID* As Integer, _ ByVal *length* As Short, ByRef *cmp()* As Integer, ByRef *usErrorNum* As Short) As Integer
VC#	public static uint ncb670_CmpLoad (uint *hDevID*, ushort *length*, ref int[] *cmp,* ref ushort *usErrorNum*);

Argument	Description
hDevID	Device handle
length	Number of CMP data
cmp	CMP data array pointer
usErrorNum	Error number

Remarks	For details on the contents of CMP data, see "7.3 Contents of CMP data".

(75) **ncb670_CmpExec () Execute CMP data**

Function	Starts executing the CMP data on the NCB specified by the device handle. After calling this function, waits for CMP execution completion by using the ncb670_WaitCmpEnd function.

Language	Format
VC++	DWORD ncb670_CmpExec (DWORD *hDevID,* WORD* *usErrorNum*);
VB	Public Function ncb670_CmpExec (ByVal *hDevID* As Integer, ByRef *usErrorNum* As Short) _ As Integer
VC#	public static uint ncb670_CmpExec (uint *hDevID,* ref ushort *usErrorNum*);

Argument	Description
hDevID	Device handle
usErrorNum	Error number

(76) **ncb670_CmpCancel () Cancel CMP data**

Function	Cancels the CMP data loaded to the NCB specified by the device handle.

Language	Format
VC++	DWORD ncb670_CmpCancel (DWORD *hDevID,* WORD* *usErrorNum*);
VB	Public Function ncb670_CmpCancel(ByVal *hDevID* As Integer, ByRef *usErrorNum* As Short) _ As Integer
VC#	public static uint ncb670_ CmpCancel (uint *hDevID,* ref ushort *usErrorNum*);

Argument	Description
hDevID	Device handle
usErrorNum	Error number

4.8.5 PCL access via DPRAM

(77) **ncb670_rMstsW () Read main status via DPRAM**

Function	Read the main status of the specifed axis of the board assigned by Device handle via DPRAM.

Language	Format
VC++	DWORD WINAPI ncb670_rMstsW(DWORD hDevID, WORD axis, WORD* wMsts);
VB	Declare Function ncb670_rMstsW Lib "libname"(ByVal hDevID As Integer, _ ByVal axis As Short, ByRef wMsts As Short) As Integer
VC#	[DllImport("*libname* ")] public static extern uint ncb670_rMstsW(uint hDevID, ushort axis, ref ushort wMsts);

Argument	Description
hDevID	Device handle
axis	specifies axis
wMsts	main status

(78) **ncb670_rSstsW () Read sub status via DPRAM**

Function	Read the sub status of the specifed axis of the board assigned by Device handle via DPRAM.

Language	Format
VC++	DWORD WINAPI ncb670_rSstsW(DWORD hDevID, WORD axis, WORD* wSsts);
VB	Declare Function ncb670_rSstsW Lib "libname"(ByVal hDevID As Integer, _ ByVal axis As Short, ByRef wSsts As Short) As Integer
VC#	[DllImport("*libname* ")] public static extern uint ncb670_rSstsW(uint hDevID, ushort axis, ref ushort wSsts);

Argument	Description
hDevID	Device handle
axis	specifies axis
wSsts	sub status

(79) **ncb670_wCmdW () Write PCL command via DPRAM**

Function	Write PCL command data to the specifed axis of the board assigned by Device handle via DPRAM.

Language	Format
VC++	DWORD WINAPI ncb670_wCmdW(DWORD hDevID, WORD axis, WORD wCmd);
VB	Declare Function ncb670_wCmdW Lib "libname"(ByVal hDevID As Integer, _ ByVal axis As Short, ByVal wCmd As Short) As Integer
VC#	[DllImport("*libname* ")] public static extern uint ncb670_wCmdW(uint hDevID, ushort axis, ushort wCmd);

Argument	Description
hDevID	Device handle
axis	specifies axis
wCmd	command data

(80) **ncb670_rReg () Read register via DPRAM**

Function	Read register of the specifed axis of the board assigned by Device handle via DPRAM.

Language	Format
VC++	DWORD WINAPI ncb670_rReg DWORD hDevID, WORD axis, BYTE byCmd, DWORD* dwData);
VB	Declare Function ncb670_rReg Lib "libname"(ByVal hDevID As Integer, _ ByVal axis As Short, ByVal byCmd As Byte, ByRef dwData As Integer) As Integer
VC#	[DllImport("*libname* ")] public static extern uint ncb670_rReg (uint hDevID, ushort axis, byte byCmd, ref uint dwData);

Argument	Description
hDevID	Device handle
axis	specifies axis
byCmd	Read register command
dwData	Read register data

(81) **ncb670_wReg () Write register via DPRAM**

Function	Write register of the specifed axis of the board assigned by Device handle via DPRAM.

Language	Format
VC++	DWORD WINAPI ncb670_wReg DWORD hDevID, WORD axis, BYTE byCmd, DWORD dwData);
VB	Declare Function ncb670_wReg Lib "libname"(ByVal hDevID As Integer, _ ByVal axis As Short, ByVal byCmd As Byte, ByVal dwData As Integer) As Integer
VC#	[DllImport("*libname* ")] public static extern uint ncb670_wReg (uint hDevID, ushort axis, byte byCmd, uint dwData);

Argument	Description
hDevID	Device handle
axis	specifies axis
byCmd	Write register command
dwData	Write register data

4.8.6 Read Firmware version

(82) **ncb670_ReadFirmwareVersion Read Firmware version**

Function	Read Firmware version of the board assigned by Device handle.

Language	Format
VC++	DWORD WINAPI ncb670_ReadFirmwareVersion(DWORD hDevID, WORD* Version);
VB	Declare Function ncb670_ReadFirmwareVersion Lib "libname"(ByVal hDevID As Integer, _ ByRef Version As Short) As Integer
VC#	[DllImport("*libname* ")] public static extern uint ncb670_ReadFirmwareVersion (uint hDevID, ref ushort Version);

Argument	Description
hDevID	Device handle
Version	Firmware version

4.9 Cautions when Using API Functions

4.9.1 Cautions when creating multithread programs

Windows supports multithread. However, exclusive control may be necessary to access a board from multiple threads by using multithread.

To control the PCL mounted on the NCB674N, data is read from and written to PCL internal registers, which require exclusive control for the reasons below.

[Port address]

The port addresses used by one axis are shown in the table below. (With 16-bit access)

Address	Read (INP)		Write (OUT)	
	Name	Content	Name	Content
+0	MSTS	Main status	CMD	Command
+2	SSTS	Sub-status	OTP	Unused (reserved)
+4	BUF0	Input/output buffer IN (15-0)	BUF0	Input/output buffer OUT (15-0)
+6	BUF1	Input/output buffer IN (31-16)	BUF1	Input/output buffer OUT (31-16)

[Register read/write procedure]

The procedure for reading from a register, or writing to a register is as follows.

■ **To read a register**

(1) Write the register read command to CMD.

(2) The content of the register is read to BUF0 and BUF1.

(3) Read BUF0 and BUF1.

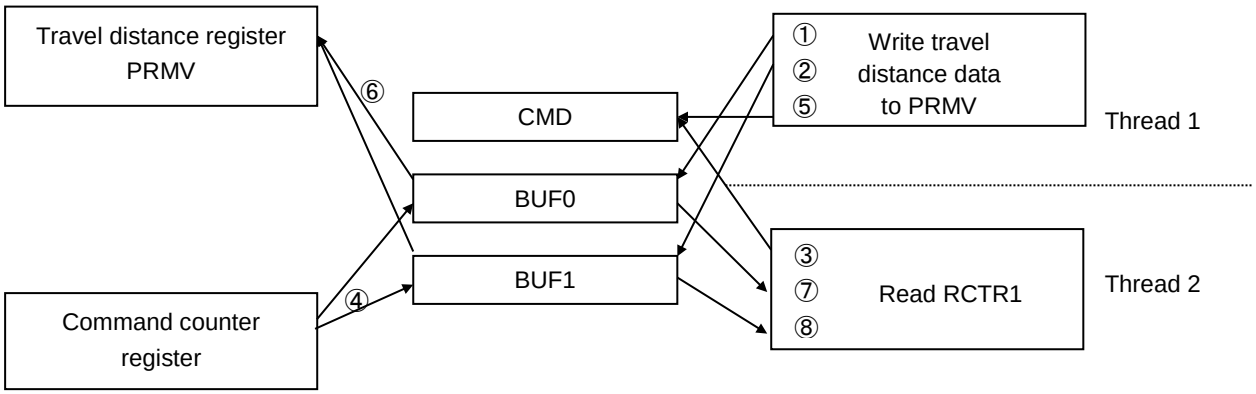
■ **To write a register**

(1) Write data to BUF0 and BUF1.

(2) Write the register write command to CMD.

(3) The data in BUF0 and BUF1 is written to the register.

The following describes an example of problem that may occur if there is no exclusive control.
If the order in which the above-described steps are executed were as shown in the figure below:



(1) Thread 1 writes data to BUF0.

(2) Thread 1 writes data to BUF1.

(3) Thread 2 writes RCTR1 read command to CMD.

(4) The contents of RCTR1 are copied to BUF0 and BUF1.

(5) Thread 1 writes PRMV write command to CMD.

(6) PRMV is written with RCTR1 data read by thread 2.

PRMV is written with the wrong data.
Therefore, exclusive control is necessary between thread 1 and thread 2.

No.	Function name	Group	No.	Function name	Group
1	hcp670_GetDevInfo		56		
2	hcp670_DevOpen	○	57		
3	hcp670_DevClose		58		
4	hcp670_SetOrgMode	○	59	hcp670_ReadErno	◇
5	hcp670_SetEls	○	60	hcp670_ReadDpFC	◇
6	hcp670_SetOls	○	61	hcp670_ReadDpReg	◇
7	hcp670_SetSvAlm	○	62	hcp670_CalAccRate	
8	hcp670_SetEz	○	63	ncb670_StartEvent	
9	hcp670_SetDlsSel	○	64	ncb670_StopEvent	
10	hcp670_SetInpos	○	65	ncb670_IsAck	
11	hcp670_SetSvCtrCl	○	66	ncb670_WaitMdaLend	
12	hcp670_SetSls	○	67	ncb670_WaitMdaEnd	□
13	hcp670_SetCmdPulse	○	68	ncb670_WaitCdaLend	
14	hcp670_SetAccProfile	○	69	ncb670_WaitCdaCLend	
15	hcp670_SetAutoDec	○	70	ncb670_WaitCrq	
16	hcp670_ReadMainSts	●	71	ncb670_WaitCdaEnd	
17	hcp670_ReadErrorSts	○	72	ncb670_WaitCmpLend	
18	hcp670_ReadEventSts	○	73	ncb670_WaitCmpEnd	□
19	hcp670_ReadSubSts	●	74	ncb670_NcbLoad	
20	hcp670_ReadExSts	○	75	ncb670_MdaExec	□
21	hcp670_ReadSpd	○	76	ncb670_MdaCStop	
22	hcp670_ReadCtr	○	77	ncb670_MdaQStop	
23	hcp670_SetFLSpd	○	78	ncb670_MdaBreakExec	□
24	hcp670_SetAuxSpd	○	79	ncb670_MdaStepExec	□
25	hcp670_SetAccRate	○	80	ncb670_CdaExec	□
26	hcp670_SetDecRate	○	81	ncb670_CdaHalt	
27	hcp670_SetMult	○	82	ncb670_CdaQstop	
28	hcp670_SetEventMask	○	83	ncb670_CdaCancel	
29	hcp670_SetDecPoint	○	84	ncb670_CmpLoad	
30	hcp670_WritOpeMode	○	85	ncb670_CmpExec	□
31	hcp670_WritFHSpd	○	86	ncb670_CmpCancel	
32	hcp670_WritPos	○	87	ncb670_rMstsW	◇
33	hcp670_WritLine	○	88	ncb670_rSstsW	◇
34	hcp670_WritCircl	○	89	ncb670_wCmdW	◇
35	hcp670_WritCtr	○	90	ncb670_rReg	◇
36	hcp670_DecStop	●	91	ncb670_wReg	◇
37	hcp670_QuickStop	●	92	ncb670_ReadFirmwareVersion	
38	hcp670_EmgStop	●	93	cp670_GetDeviceCount	
39	hcp670_AccStart	●	94	cp670_GetDeviceInfo	
40	hcp670_CnstStartFH	●	95	cp670_OpenDevice	
41	hcp670_CnstStartFL	●	96	cp670_CloseDevice	
42	hcp670_CnstStartByDec	●	97	cp670_rMstsW	●
43	hcp670_SvOn	●	98	cp670_rSstsW	●
44	hcp670_SvOff	●	99	cp670_wCmdW	△
45	hcp670_SvResetOn	●	100	cp670_rReg	○
46	hcp670_SvResetOff	●	101	cp670_wReg	○
47	hcp670_PMOn	●	102	cp670_rPortW	○
48	hcp670_PMOff	●	103	cp670_wPortW	○
49			104	cp670_rExPortW	●
50			105	cp670_wExPortW	●
51			106	cp670_rBufDW	○
52			107	cp670_wBufDW	○
53			108	cp670_WaitNextInterrupt	
54			109	cp670_CancelWaitInterrupt	
55					

Functions in the ○ group require exclusive control when called from multiple threads.
Functions in the ◇ group require exclusive control when called from multiple threads.
Function in the △ group of using the register control command require exclusive control with the functions in the ○ group.
During execution of the functions on the □ or ◇ group, do not use the function in ○ and ● group.

4.9.2 Cautions when using NCB mode

When combining control from the software on a PC and control by using NCB mode, access to PCL requires exclusive control. A lack of exclusive access control may result in unexpected operation.

For details on how to implement exclusive control, see "8. Accessing PCL during MDA, CDA, or CMP Execution".

4.10 Function Return Values

The result of starting a function is reflected in the "return value". Normally, "0" is returned when successful. Note that problems occurring in servo or machine sensors are not included in these return values. Analyze the error information and take appropriate measures for each individual cause.

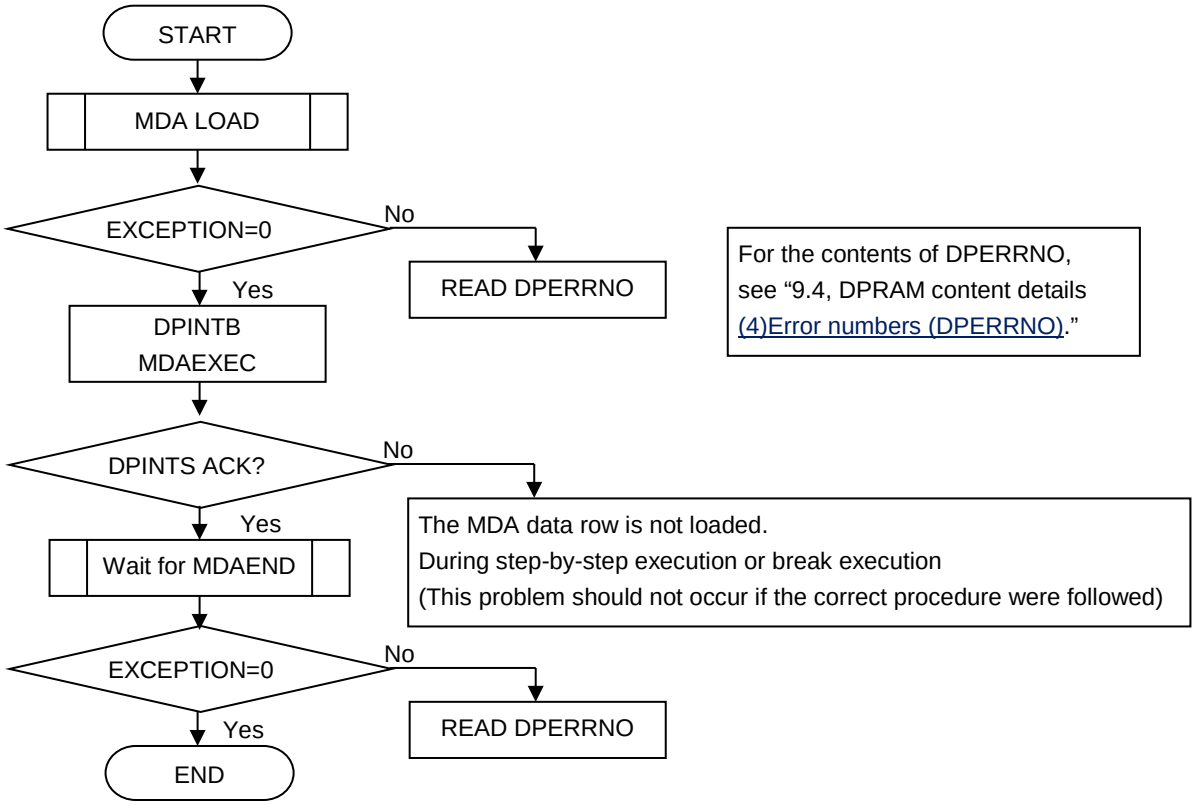
No.	Return value		Details of the error and item(s) to check
	Error text	HEX	
1	NO_ERROR	00000000h	Normal
2	NOT_FOUND	00000001h	Device driver not found
3	ALREADY_OPENED	00000002h	Attempt to open an already opened device
4	INSUFFICIENT_MEMORY	00000004h	Not enough memory to store device information
5	INVALID_HANDLE	00000008h	Specified device handle is invalid
6	NOT_READY	00000010h	Input/output ports of the device are unavailable There may be an inconsistency and the like in the system. Contact Hivertec Support desk.
7	ILLEGAL_DEVICE	00000020h	The board information could not be read when opening the device There may be an inconsistency and the like in the system. Contact Hivertec Support desk.
8	ILLEGAL_PARAM	00000100h	Value of function argument in error Value range for the speed multiplier setting is 2 to 4095.
9	CPU_BUSY	00000400h	Initialization of the CPU on the board has not completed There may be an inconsistency and the like in the system. Contact Hivertec Support desk.
10	ALREADY_WAITING	00010000h	NCB is already waiting for interrupt
11	WAITING_CANCELED	00020000h	Wait for interrupt has been canceled by CancelWaitInterrupt() function
12	ERROR_DP_NAK	80000001h	DP_INTS is NAK
13	ERROR_DP_EXCPTN	80000002h	An exception has occurred
14	ERROR_DP_TIMEOUT	80000003h	A timeout has occurred in MDA, CDA, or CMP control function
15	ERROR_DP_BUFBSY	80000004h	Input/output buffer control in progress
16	CDA_HALT	90000001h	CDA execution is temporarily stopped
17	CDA_QSTOP	90000002h	CDA execution has been stopped quickly

5. NCB Mode (Automatic Program Execution Function: MDA)

The automatic program execution function runs automatic execution programs loaded to the board in advance on the onboard CPU.
Automatic execution programs must be loaded to the board in advance as continuous, multi-line data rows (up to 4000 lines) composed of 4-word MDA operation blocks. Thereafter, the programs are automatically executed on MDAEXEC write.

5.1 Outline of Execution Procedure

Configure in advance registers that do not to be rewritten during MDA execution.
The outline of a MDA execution procedure is as follows:



5.2 MDA loading procedure

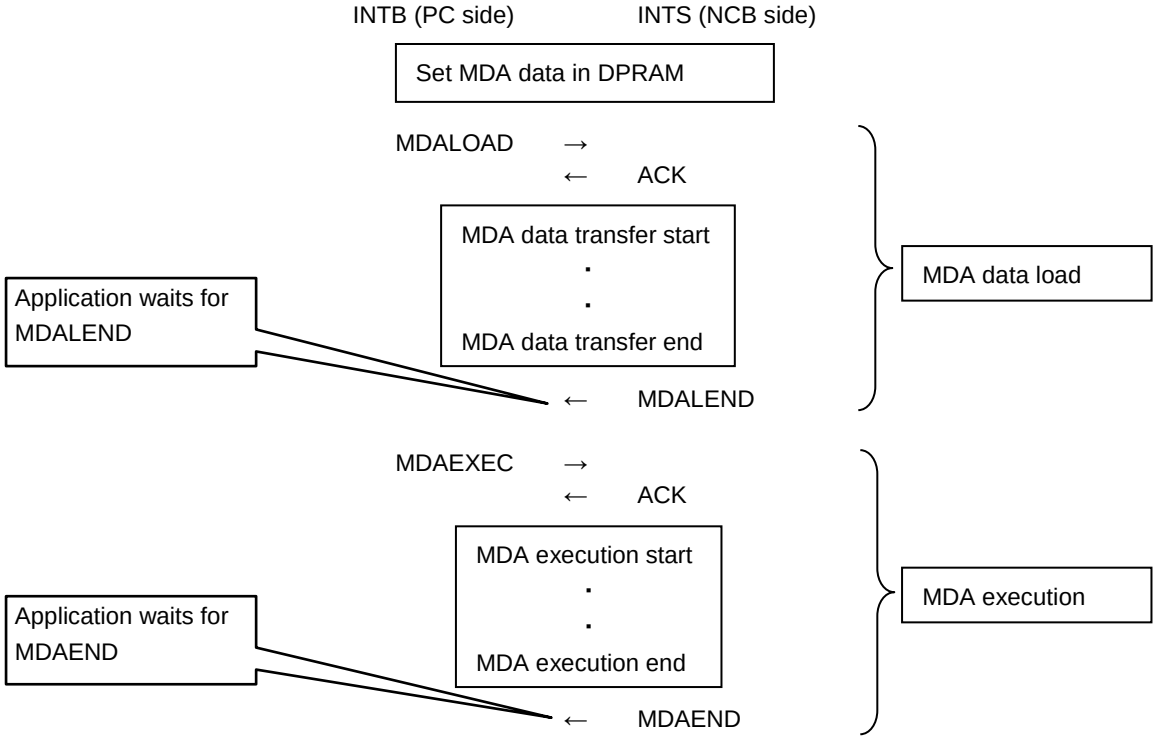
(1) Write "0" to DPWMPTR.
(2) Write MDALOAD to DPINTB.
(3) Check ACK response from DPINTS.
(4) Wait until DPRMPTR=0.
(5) Write MDA data to DPMDA (address = DPMDA base address + DPWMPTR x 8).
(6) Increment DPWMPTR to write the next data.

After writing all data, wait for MDALEND from DPINTS.

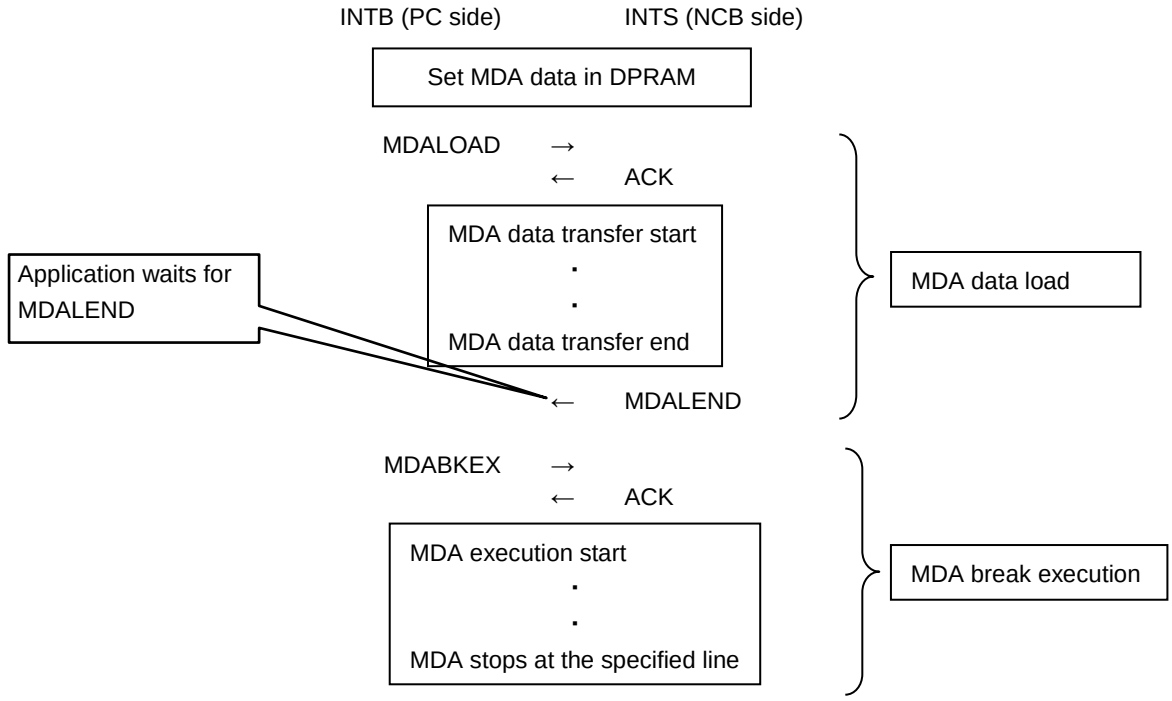
In the library functions for MDA control, there is a function that performs the above-described procedure.
For details, see "4.8.2 MDA control functions".

5.3 Example of Normal MDA Operation

5.3.1 Example (1) of normal MDA operation

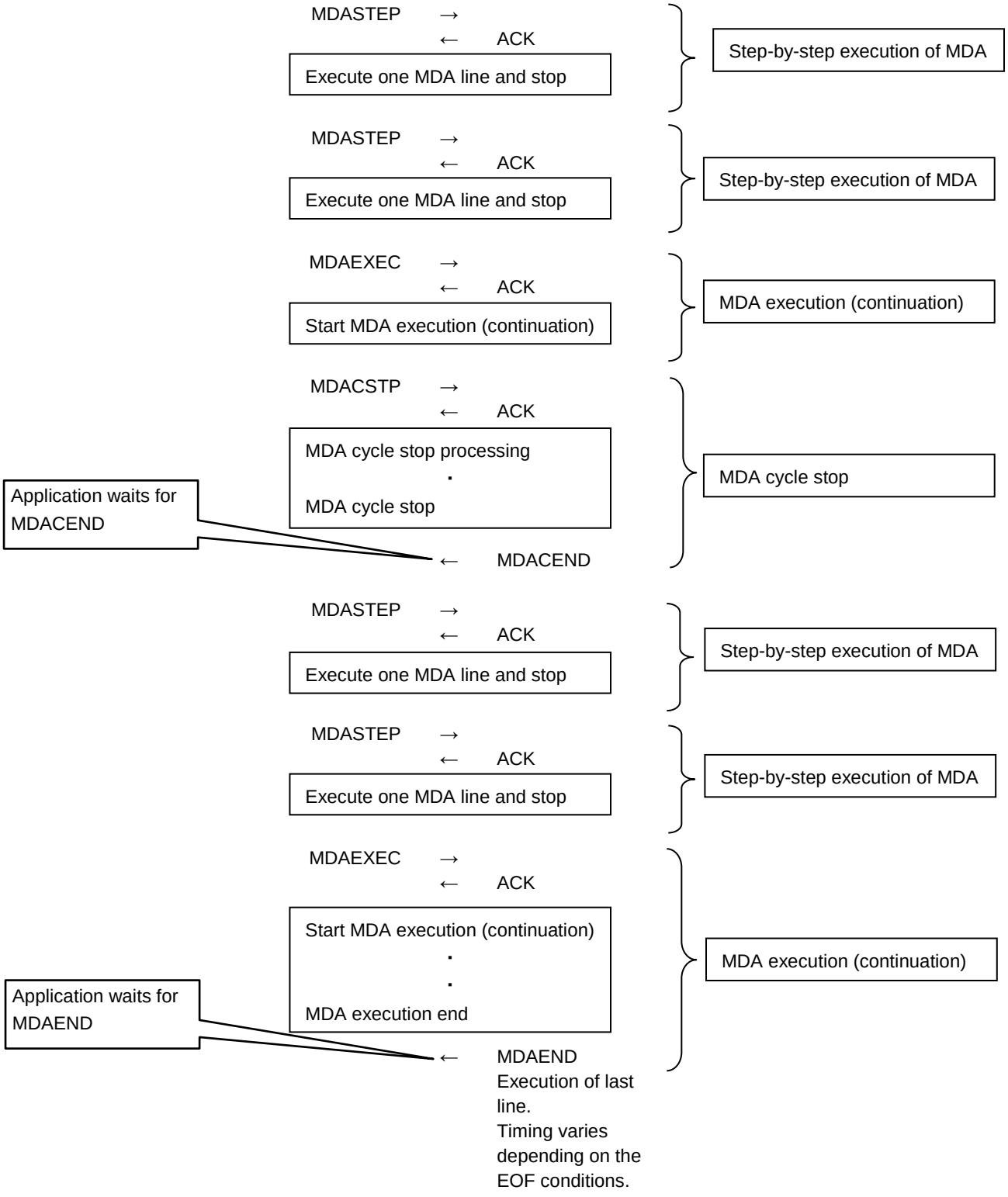


5.3.2 Example (2) of normal MDA operation



Example of step-by-step execution, continuation of execution, and cycle stop on next page.

Continued from previous page



5.4 DPMDA Data Configuration

The data configuration of DPMDA is as shown in the figure below.

SEQ_NO	Type
1	Top initialization block
2	Various MDA operation blocks
: : :	: : :
N	End block

5.5 MDA Operation Block

The following explains the MDA operation blocks.

One line of MDA operation block has a fixed length of four words. Its structure is shown in the figure below.

Address	0	1	2	3
Abbreviation	SEQ_NO	DATA1(CMD/CND)	DATA2	DATA3
Content	Line number	High byte: MDA command (CMD) Low byte: write condition (CND)	Data	Data

DATA1 consists of the write condition (CND) and MDA command (CMD).

The contents of DATA2 and DATA3 vary depending on the type of MDA command.

The following seven types of MDA operation blocks are available:

No.	Name	Abbreviation	Content
1	Top initialization block	TOF	Number of data, designation of axes to use, designation of input/output ports to use
2	End block	EOF	Exit condition
3	PCL command block	WCMD_PCL	Command to write to PCL and write condition (CND) Use this block to set data in registers, write a start command or the like.
4	PCL input/output buffer write block	WBF	Data to write to PCL input/output buffer and write condition (CND) Use this block to write data to registers. To set data in registers, set the data to write to registers in this block, and follow it with a PCL command block (where PCL command is the write register command) as the next data.
5	DO output block	OUT	Generic output data and conditions for generic output
6	DI input wait block	WAT_IN	Designation of conditions for waiting for generic input
7	CDA header block	TOB	When executing CDA, the number of lines to execute is specified in this block.

5.6 Contents of DATA1(CND/CMD)

The contents of DATA1(CND/CMD) are as follows.

5.6.1 Write condition (CND)

Write condition is 1-byte data.

■ **Bit configuration**

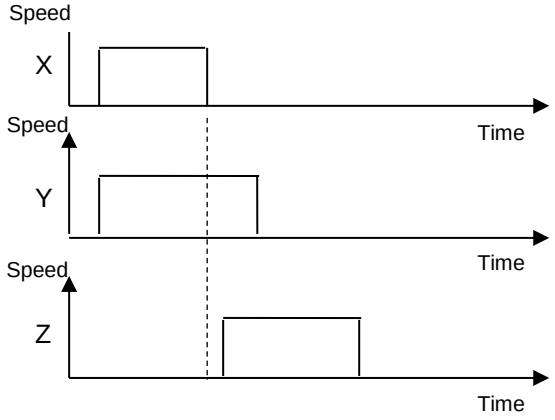
Bit	7	6	5	4	3	2	1	0
Bit name	0	IGN	OUT	AND	UNXT	ZNXT	YNXT	XNXT

Bit	Bit name	Content	Remarks
0	XNXT	X-axis	Bit7-0='00000000' means continuous execution.
1	YNXT	Y-axis	
2	ZNXT	Z-axis	(Reserves the next operation without waiting for the previous to complete)
3	UNXT	U-axis	
4	AND	Conditions for bit3-0 0: OR, 1: AND	
5	OUT	1: After completion of generic pulse output by the previous operation	
6	IGN	1: Ignore previous operation (However, bit5-0 must be all "0")	

Designates the axis (axes) that will wait for the previous operation to complete. 1: Wait for completion

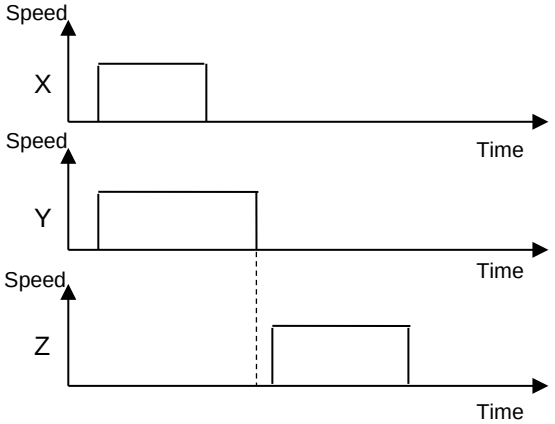
■ **How to use the xNXT parameter in the MDA command block**

- Axes X and Y are respectively performing positioning. To start Z-axis operation after confirming that X-axis operation is completed:



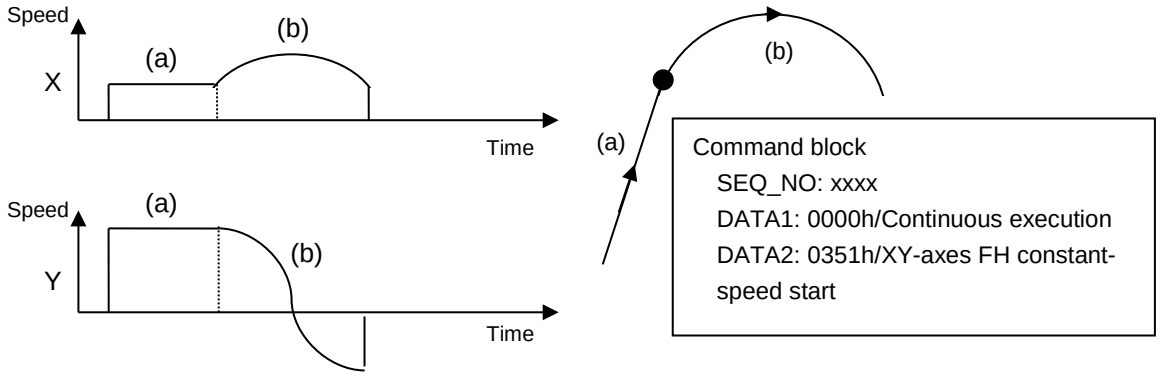
Command block
SEQ_NO: xxxx
DATA1: 0001h/After confirming X-axis positioning completion

- Axes X and Y are respectively performing positioning. To start Z-axis operation after confirming that Y-axis operation is completed:



Command block
SEQ_NO: xxxx
DATA1: 0002h/After confirming Y-axis positioning completion
DATA2: 0451h/Z-axis FH constant-speed start

- To execute XY circular interpolation as a continuation of XY linear interpolation:



5.6.2 MDA command (CMD)

MDA command is 1-byte data.

■ **Bit configuration**

Bit	15	14	13	12	11	10	9	8
Bit name	CMD							

No.	Command details	Abbreviation	Command code (HEX)	Remarks
1	Write command to PCL	WCMD_PCL	00	
2	Output DO	OUT	01	
3	Wait for DI input	WAT_IN	02	
4	Write X-axis input/output buffer	WBF_X	03	
5	Write Y-axis input/output buffer	WBF_Y	04	
6	Write Z-axis input/output buffer	WBF_Z	05	
7	Write U-axis input/output buffer	WBF_U	06	
8	Header	TOF	07	
9	Footer	EOF	08	
10	CDA header	TOB	09	

5.7 Details of MDA Operation Block

5.7.1 Top initialization block (TOF) configuration

The first data of an MDA operation block is always TOF.

	15 8	7 0
SEQ No.	SEQ_NO (fixed to 1)	
DATA1	High byte: TOF (07h)	Low byte: CND (write condition) • • • Fixed to "0"
DATA2	Length: Total number of MDA blocks (3 to 4000)	
DATA3	Designation of generic input/output ports and axes to use	
	High byte: SEL_PORT	Low byte: SEL_AXIS

■ **Contents of SEL_AXIS (axes to use)**

Bit operation allows for specifying multiple axes simultaneously.

Bit	Name	Setting data
0	XSEL	1: Use X-axis
1	YSEL	1: Use Y-axis
2	ZSEL	1: Use Z-axis
3	USEL	1: Use U-axis
7-4		Reserved

■ **Contents of SEL_PORT (generic input/output ports to use)**

Bit operation allows for specifying multiple generic input/output ports simultaneously.

Bit	Name	Setting data
8	DO	DO setting 0: OUT1 to OUT8, 1: OUT9 to OUT16
9	DI	DI setting 0: IN1 to IN8, 1: IN9 to IN16
15-10		Reserved

5.7.2 End block (EOF) configuration

The last data of an MDA operation block is always EOF.

	15 8	7 0
SEQ No.	SEQ_NO (3 to 3999)	
DATA1	High byte: EOF (08h)	Low byte: CND (write condition)
DATA2	Reserved	
DATA3	Reserved	

5.7.3 PCL command block (WCMD_PCL) configuration

This block is for writing commands to PCL.

	15 8	7 0
SEQ No.	SEQ_NO (2 to 3999)	
DATA1	High byte: WCMD_PCL (00h)	Low byte: CND (write condition)
DATA2	High byte: SEL_AXIS (axes to use)	Low byte: CMDB (PCL command data)
DATA3	Reserved	

CMDB (PCL command data) may be a control command or a register write command.

When only sending a start, servo ON/OFF, or any other instruction, specify control command.

When writing to registers, make sure the previous line is a PCL input/output buffer write block, and specify register write command in this block.

■ **Content of CMDB (PCL command data - control command)**

Content	Command (HEX)
FL constant-speed start	50
FH constant-speed start	51
Deceleration stop after FH constant-speed start	52
Acceleration start	53
STA output (simultaneous start)	06
SVON OFF	10
SVON ON	18

Content	Command (HEX)
SVRESET OFF	11
SVRESET ON	19
Counter 1 reset	20
Counter 2 reset	21
Counter 3 reset	22
Counter 4 reset	23

■ **Content of CMDB (PCL command data - register write command)**

When writing to a register with preregister, writing is basically performed to the preregister.

A register is written directly only when performing speed override or position override.

Content	Register		Preregister	
	Name	Command (HEX)	Name	Command (HEX)
Travel distance	RMV	90	PRMV	80
Base speed setting	RFL	91	PRFL	81
Operation speed setting	RFH	92	PRFH	82
Acceleration rate determining parameter	RUR	93	PRUR	83
Deceleration rate determining parameter	RDR	94	PRDR	84
Speed multiplier setting	RMG	95	PRMG	85
Deceleration start point	RDP	96	PRDP	86
Operation mode	RMD	97	PRMD	87
Circular interpolation center	RIP	98	PRIP	88
S-curve range of acceleration setting	RUS	99	PRUS	89
S-curve range of deceleration setting	RDS	9a	PRDS	8a
Auxiliary speed setting	RFA	9b		
Environment setting 1	RENV1	9c		
Environment setting 2	RENV2	9d		
Environment setting 3	RENV3	9e		
Environment setting 4	RENV4	9f		
Environment setting 5	RENV5	a0		
Environment setting 6	RENV6	a1		
Environment setting 7	RENV7	a2		
Counter 1 (Command pulse output)	RCTR1	a3		
Counter 2 (Encoder input)	RCTR2	a4		
Counter 3 (error counter)	RCTR3	a5		
Counter 4 (general-purpose counter)	RCTR4	a6		
Comparator 1 data	RCMP1	a7		
Comparator 2 data	RCMP2	a8		
Comparator 3 data	RCMP3	a9		
Comparator 4 data	RCMP4	aa		
Comparator 5 data	RCMP5	ab	PRCP5	8b
Event mask setting	RIRQ	ac		
Number of circular interpolation steps	RCI	bc	PRCI	8c

■ **Contents of SEL_AXIS (axes to use)**

Bit operation allows for specifying multiple axes simultaneously.

Bit	Name	Setting data
0	XSEL	1: Use X-axis
1	YSEL	1: Use Y-axis
2	ZSEL	1: Use Z-axis
3	USEL	1: Use U-axis
7-4		Reserved

5.7.4 PCL input/output buffer write block configuration

Use this block to write data to PCL input/output buffers.

Set the register write data for the number of axes to configure, and follow it with the PCL command block on the next line.

	15 8	7 0
SEQ No.	SEQ_NO (2 to 3999)	
DATA1	High byte: WBF 03h: Write input/output buffer for X-axis 04h: Write input/output buffer for Y-axis 05h: Write input/output buffer for Z-axis 06h: Write input/output buffer for U-axis	Low byte: CND (write condition)
DATA2	BUF0 (bit15-0)	
DATA3	BUF1 (bit31-16)	

Set the data to write to the register in BUF0 and BUF1.

5.7.5 DO output block (OUT) configuration

Use when using generic output.

	15 8	7 0
SEQ No.	SEQ_NO (2 to 3999)	
DATA1	High byte: OUT (01h)	Low byte: CND (write condition)
DATA2	High byte: OUTM (output mode)	Low byte: OUTD (output data)
DATA3	PWIDTH; pulse width x ms (1 to 6553ms)	

■ **Contents of OUTD (output data)**

For OUTD (output data), set the output bit to enable to '1', independently of whether the output is level or pulse, or ON or OFF (positive pulse or negative pulse).

Bit	Name	Setting data
7-0	OUTx	Output data of OUT1-OUT8 (or OUT9-OUT16). Whether the used ports are OUT1-OUT8 or OUT9-OUT16 depend on the SEL_PORT setting in TOF.

■ **Contents of OUTM (output mode)**

Bit	Name	Setting data
8	PLS	0: Level, 1: Pulse
9	POL	0: ON/Positive pulse, 1: OFF/Negative pulse Also valid for level. (See below).
15-10		Reserved

■ **PLS and POL setting combination**

Bit	9	8	Output format
	0	0	Level ON
	1	0	Level OFF
	0	1	Positive pulse
	1	1	Negative pulse

5.7.6 DI input wait block (WAT_IN) configuration

Use this block to wait for generic input.

	15 8	7 0
SEQ No.	SEQ_NO (2 to 3999)	
DATA1	High byte: WAT_IN (02h)	Low byte: CND (write condition)
DATA2	High byte: INM (input wait mode)	Low byte: IND (data awaiting input)
DATA3	TOUT Timeout time 0: INFINITE/Other than 0: x (1 to 65535) ms	

■ **Contents of IND (data awaiting input)**

Bit	Name	Setting data
7-0	INx	Data awaiting input at IN1-IN8 (or IN9-IN16) Whether the used ports are IN1-IN8 or IN9-IN16 depend on the SEL_PORT setting in TOF.

■ **Contents of INM (input wait mode)**

Bit	Name	Setting data
8	AND	Input wait condition 0: OR of IND, 1: AND of IND
9	POL	Polarity of data awaiting input 0: ON, 1: OFF
15-10		Reserved

5.8 Writing to Registers

To write to a register, describe the operation by using PCL input/output buffer write blocks and PCL command block (register write command).
First describe data in as many input/output buffer write blocks as axes to configure, and then specify the axes and register to write in the PCL command block.

■ When writing data to registers

To write to registers for axes X, Y, Z, and U:

SEQ_NO	Type
N	PCL input/output buffer write block (X-axis)
N+1	PCL input/output buffer write block (Y-axis)
N+2	PCL input/output buffer write block (Z-axis)
N+3	PCL input/output buffer write block (U-axis)
N+4	PCL command block (register write command) Designate axes X, Y, Z, and U in SEL_AXIS.

To write to a register for X-axis:

SEQ_NO	Type
N	PCL input/output buffer write block (X-axis)
N+1	PCL command block (register write command) Only designate axis X in SEL_AXIS.

5.9 Example of MDA data

5.9.1 Example of X-axis positioning operation

The following is an example of data for X-axis positioning. (All data in hexadecimal notation)

Word name	Data		Content explanation
SEQ_NO	0002		
DATA1	CMD	CND	CMD: Write input/output buffer for X-axis
	03	00	CND: Next operation on writing
DATA2	0041		BUF0: Data to write to input/output buffer bit15-0
DATA3	0000		BUF1: Data to write to input/output buffer bit31-16

Word name	Data		Content explanation
SEQ_NO	0003		
DATA1	CMD	CND	CMD: Write command to PCL
	00	00	CND: Next operation on writing
DATA2	SEL_AXIS	CMDB	SEL_AXIS: X-axis
	01	87	CMDB: PRMD register write
DATA3	0000		Reserved

Word name	Data		Content explanation
SEQ_NO	0004		
DATA1	CMD	CND	CMD: Write input/output buffer for X-axis
	03	00	CND: Next operation on writing
DATA2	1388		BUF0: Data to write to input/output buffer bit15-0
DATA3	0000		BUF1: Data to write to input/output buffer bit31-16

Word name	Data		Content explanation
SEQ_NO	0005		
DATA1	CMD	CND	CMD: Write command to PCL
	00	00	CND: Next operation on writing
DATA2	SEL_AXIS	CMDB	SEL_AXIS: X-axis
	01	80	CMDB: PRMV register write
DATA3	0000		Reserved

Word name	Data		Content explanation
SEQ_NO	0006		
DATA1	CMD	CND	CMD: Write command to PCL
	00	00	CND: Next operation on writing
DATA2	SEL_AXIS	CMDB	SEL_AXIS: X-axis
	01	53	CMDB: Acceleration start
DATA3	0000		Reserved

5.6.1 Write condition (CND)

4.PCL Document of the CPD series Users manual

SEQ_NO 2 and 3 set the operation mode register (RMD) for X-axis to relative positioning (00000041h).

5.7.3 PCL command block (WCMD_PCL) configuration

SEQ_NO 4 and 5 set travel distance register (RMV) for X-axis to "5000 (1388h)".

SEQ_NO 6 sets acceleration start for X-axis

Based on the above-described data, X-axis starts relative positioning for 5000 pulses.

5.9.2 Example of XY linear Interpolation operation

The following is an example of data for XY-axis linear interpolation. (All data in hexadecimal notation)

Word name	Data		Content explanation
SEQ_NO	0002		
DATA1	CMD	CND	CMD: Write input/output buffer for X-axis
	03	00	CND: Next operation on writing
DATA2	8061		BUF0: Data to write to input/output buffer bit15-0
DATA3	0800		BUF1: Data to write to input/output buffer bit31-16

Word name	Data		Content explanation
SEQ_NO	0003		
DATA1	CMD	CND	CMD: Write input/output buffer for Y-axis
	04	00	CND: Next operation on writing
DATA2	8061		BUF0: Data to write to input/output buffer bit15-0
DATA3	0800		BUF1: Data to write to input/output buffer bit31-16

Word name	Data		Content explanation
SEQ_NO	0004		
DATA1	CMD	CND	CMD: Write command to PCL
	00	00	CND: Next operation on writing
DATA2	SEL_AXIS	CMDB	SEL_AXIS: X and Y axes
	03	87	CMDB: PRMD register write
DATA3	0000		Reserved

SEQ_NO 2, 3, and 4 set the operation mode registers (RMD) for X and Y axes to linear interpolation (08008061h).

Word name	Data		Content explanation
SEQ_NO	0005		
DATA1	CMD	CND	CMD: Write input/output buffer for X-axis
	03	00	CND: Next operation on writing
DATA2	1388		BUF0: Data to write to input/output buffer bit15-0
DATA3	0000		BUF1: Data to write to input/output buffer bit31-16

Word name	Data		Content explanation
SEQ_NO	0006		
DATA1	CMD	CND	CMD: Write input/output buffer for Y-axis
	04	00	CND: Next operation on writing
DATA2	2710		BUF0: Data to write to input/output buffer bit15-0
DATA3	0000		BUF1: Data to write to input/output buffer bit31-16

Word name	Data		Content explanation
SEQ_NO	0007		
DATA1	CMD	CND	CMD: Write command to PCL
	00	00	CND: Next operation on writing
DATA2	SEL_AXIS	CMDB	SEL_AXIS: X and Y axes
	03	80	CMDB: PRMV register write
DATA3	0000		Reserved

SEQ_NO 5, 6, and 7 set travel distance registers (RMV) for X and Y axes to X=5000 (1388h) and Y=10,000 (2710h), respectively.

Continued to the next page

Continued from the previous page

Word name	Data		Content explanation
SEQ_NO	0008		
DATA1	CMD	CND	CMD: Write command to PCL
	00	00	CND: Next operation on writing
DATA2	SEL_AXIS	CMDB	SEL_AXIS: X and Y axes
	03	53	CMDB: Acceleration start
DATA3	0000		Reserved

SEQ_NO 8
sets acceleration start
for axes X and Y

Based on the above described data, linear interpolation is performed for X=5000 and Y=10000.

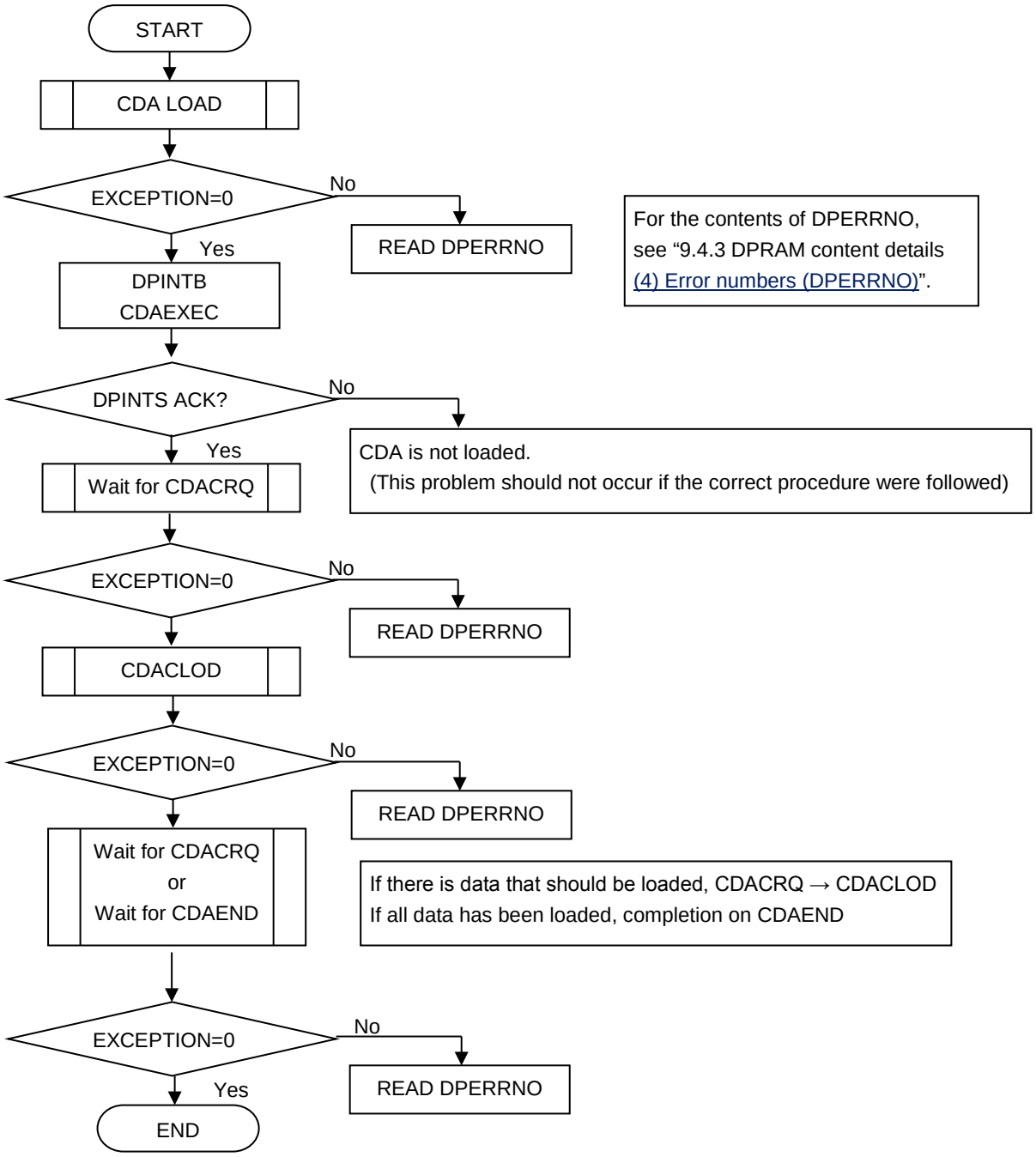
6. NCB Mode (Automatic Program Execution Function: CDA)

CDA function is an enhancement to the MDA function. This function divides the 4000 blocks of second operation block onwards of the MDA function into halves of 2000 blocks each to load them alternately to each area for 2000 blocks and perform MDA operations successively.

6.1 Outline of Execution Procedure

Configure in advance registers that do not to be rewritten during CDA execution.

The outline of a CDA execution procedure is as follows:



6.2 CDA loading procedure

(1) Write "0" to DPWMPTR.
(2) Write CDALOAD to DPINTB.
(3) Check ACK response from DPINTS.
(4) Wait until DPRMPTR=0.
(5) Write MDA data to DPMDA (address = DPMDA base address + DPWMPTR x 8).
(6) Increment DPWMPTR to write the next data.
After writing all data, wait for CDALEND from DPINTS.

In the library functions for CDA control, there is a function that performs the above-described procedure.
For details, see "4.8.2 MDA control functions" and "4.8.3 CDA control functions".
Also see "3.4 Sample Programs (NCB Series)" and the source code of the bundled sample program.

6.3 CDA Data Row Configuration

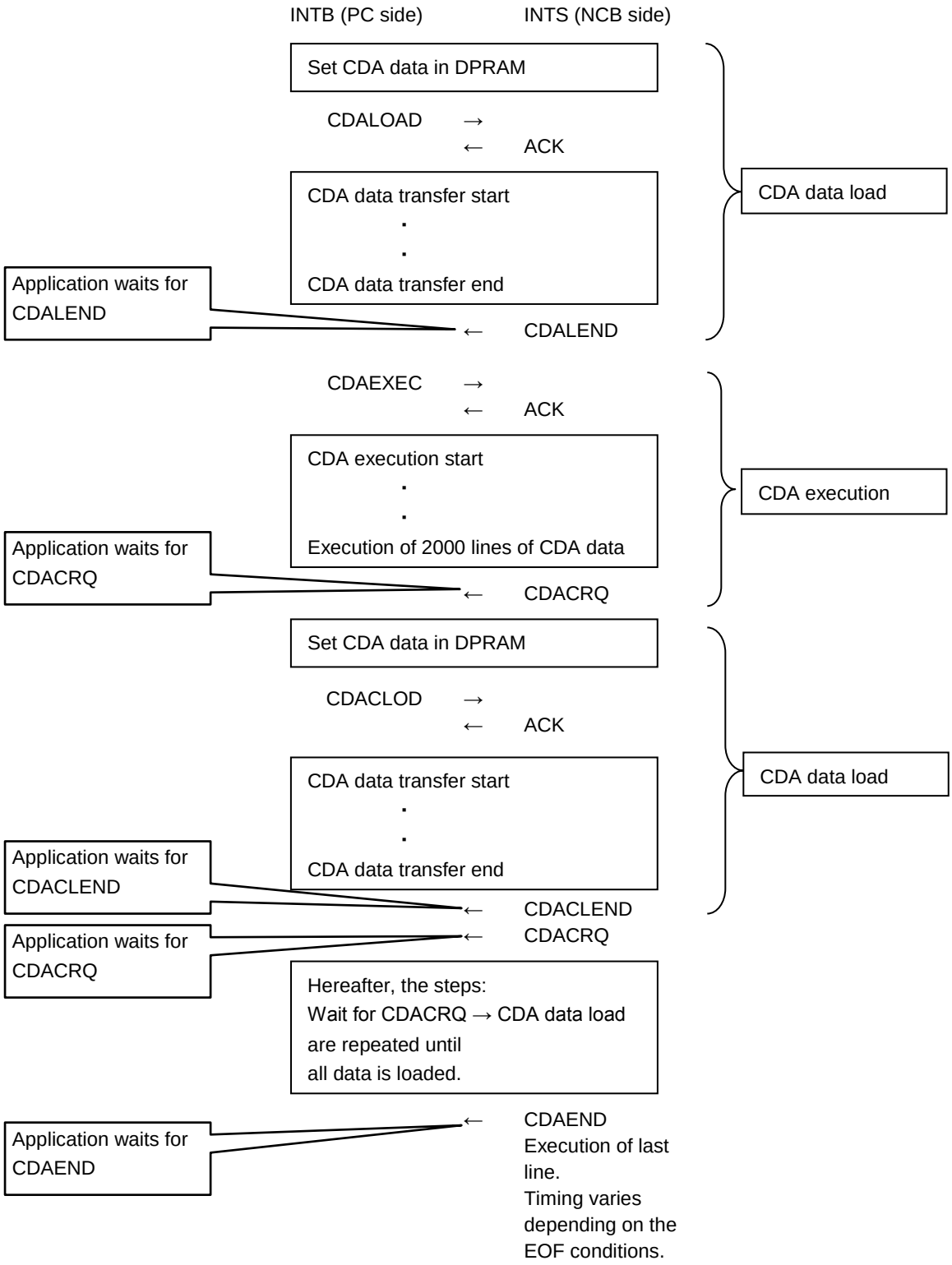
The configuration of a CDA data row is as shown in the figure below. * The figures in square brackets are the number of blocks to be set in each header.

SEQ_NO	Type	Page number
1	Top initialization block (TOF) [4,000]	Page 0
2	Various MDA operation blocks	
:	:	
2001	CDA header block (TOB) [2,000]	
2002	Various MDA operation blocks	
:	:	
4000	Various MDA operation blocks	
1	CDA header block (TOB) [2,000]	Page 1
2	Various MDA operation blocks	
:	:	
2001	CDA header block (TOB) [2,000]	
2002	Various MDA operation blocks	
:	:	
4000	Various MDA operation blocks	
:	:	:
1	CDA header block (TOB) [n]	Page N
2	Various MDA operation blocks	
:	:	
n	End block (EOF)	

6.3.1 CDA header block (TOB) configuration

	15 8	7 0
SEQ No.	SEQ_NO (fixed to 1)	
DATA1	High byte: TOB (09h)	Low byte: CND (write condition) • • • Fixed to "0"
DATA2	Length: Total number of MDA blocks (2 to 2000)	
DATA3	Reserved	

6.4 Example of Normal CDA Operation



6.5 Calculation of Last Executed Line on CDA Execution Cycle Stop

To confirm the line number of CDA data executed last before the execution cycle was stopped (by using MDACSTOP), check DPEXPPG (page number: 0 to 65,535) and DPEXPNO (execution completed line number: 1 to 4,000) in DPRAM.

The expression to calculate the line number is as follows:

CDA line number executed last (UDL) = DPEXPPG x 4,000 + DPEXPNO

If the target line number in the original data is USL, then

USL = {(DPEXPPG x 4000 + DPEXPNO) x 1999 + 2} / 2000 (Note that USL is an integer; decimals are rounded down)

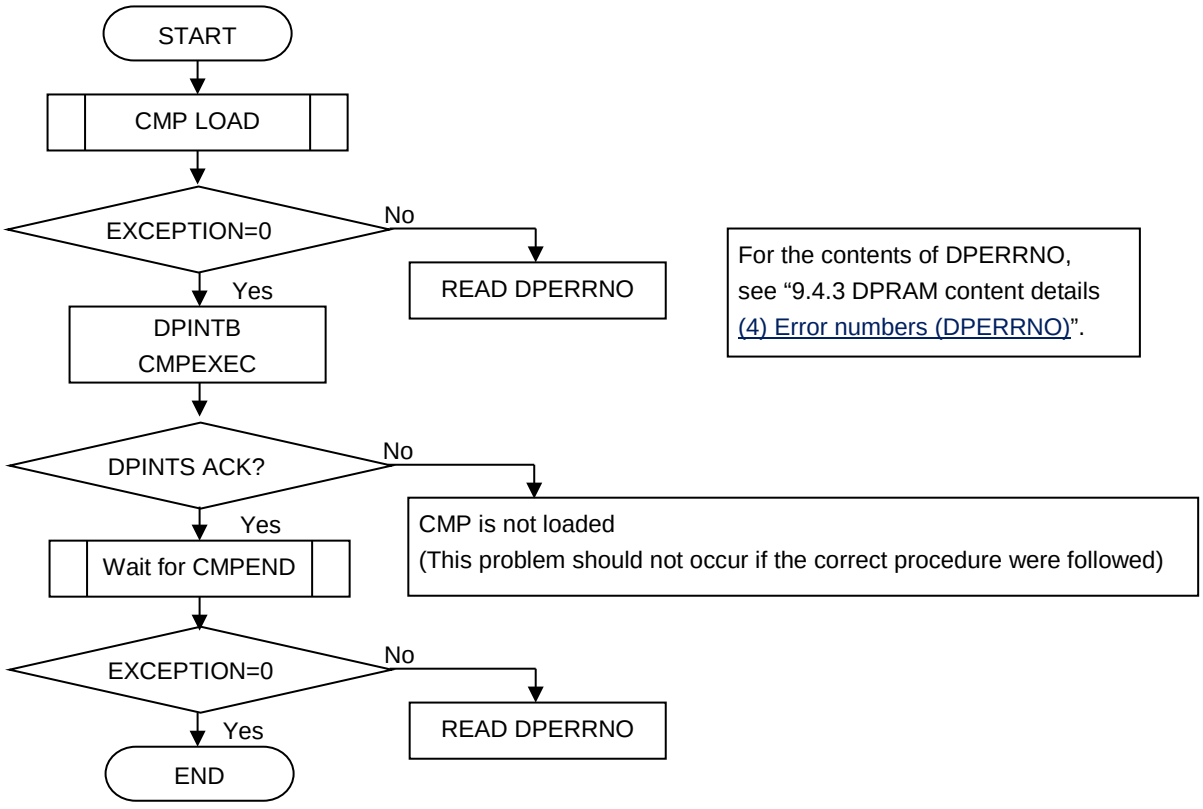
7. NCB Mode (Automatic Comparator Execution Function: CMP)

The automatic comparator execution function loads the comparison data of 1998 points to the board in advance to automatically output comparator matches.
It can use comparator CMP5 for only one of the axes X to U.

7.1 Outline of Execution Procedure

Set comparator comparison condition (RENV4 or RENV5) and configure external output of comparator comparison result (COTSEL1) in advance.
For details on how to set comparator comparison conditions, refer to "CPD Series User's Manual <Operation>" and for details on how to configure external output of comparator comparison result (COTSEL1), see "9.3.5 Set/Read external output selection of comparator 3 to 5 comparison result (COTSEL1)".

The outline of a CMP execution procedure is as follows:



7.2 CMP Loading Procedure

(1) Write "0" to DPWCPTR.
(2) Write CMPLOAD to DPINTB.
(3) Check ACK response from DPINTS.
(4) Wait until DPRCPTR=0.
(5) Write CMP data to DPCMP (address = DPCMP base address + DPWCPTR x 4).
(6) Increment DPWCPTR to write the next data.
(7) After writing all data, wait for CMPLEND from DPINTS.

7.3 Contents of CMP data

7.3.1 Types of comparator data

The length of one data line is fixed to two words. The following three types of data are available:

No.	Name	Content
1	Top initialization block	Number of data, designation of axes to use
2	End block	
3	Comparator comparison data	Comparator comparison data

7.3.2 Comparator data configuration

SEQ_NO	Type
1	Top initialization block
2	First comparator comparison data
: : :	: : :
N-1	N-2th comparator comparison data
N	End block

7.4 Comparator Data Details

7.4.1 Top initialization block (header)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SEL_AXIS (axis selection)								TOF							
				U	Z	Y	X	Fixed to 40h							

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Length															
Total number of blocks (3 to 2002)															

7.4.2 End block (Footer)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								EOF							
Reserved								Fixed to 80h							

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved															
Reserved															

7.4.3 Comparator comparison data

Consists of signed, 32-bit comparator comparison data.

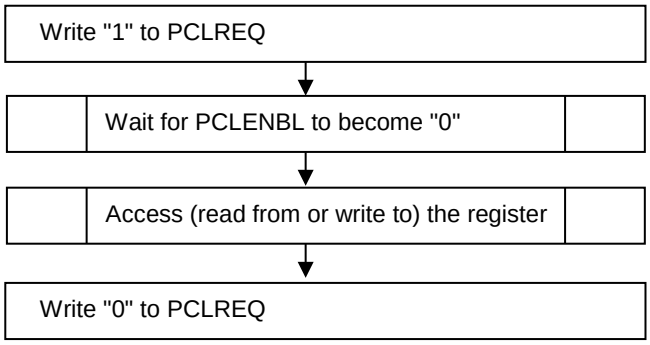
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BUF0															
Comparator comparison data, LOW WORD															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BUF1															
Comparator comparison data, HIGH WORD															

7.4.4 Accessing PCL during CMP execution

During CMP execution, accessing registers for the axes used by CMP from the software on the PC is restricted.

To access to registers for the axes used for CMP during CMP execution, proceed as described below.



8. Accessing PCL during MDA, CDA, or CMP Execution

Warning

During MDA, CDA, and CMP execution, access to the PCL from the software on the PC will be prohibited.
During MDA, CDA, and CMP execution, access to the PCL from the software on the PC may cause the unexpected behave and be very dangerous.
Reading main status or sub status, writing PCL command, read/write register will be done via the DPRAM.

8.1 How to access PCL During MDA, CDA, or CMP Execution

8.1.1 Read main status during MDA, CDA, or CMP execution

When using the Firmware version 2.11 of after, use the library function "**ncb670_rMstsW**".
When using the Firmware version 2.10 of earlier, reading main status during MDA, CDA, or CMP execution is forbidden.
For detail of the mentioned function. please refer to the "4.8.5 PCL access via DPRAM (87) ncb670_rMstsW () Read main status via DPRAM"
If it is called from multiple threads, exclusive control is required.

For DPRAM access, please refer to the "9.4.3 DPRAM content details (10) Request for writing PCL command ／ reading main status (DPCMRW)".

8.1.2 Read sub status during MDA, CDA, or CMP execution

When using the Firmware version 2.11 of after, use the library function "**ncb670_rSstsW**".
When using the Firmware version 2.10 of earlier, reading sub status during MDA, CDA, or CMP execution is forbidden.
For detail of the mentioned function. please refer to the "4.8.5 PCL access via DPRAM (88) ncb670_rSstsW () Read sub status via DPRAM"
If it is called from multiple threads, exclusive control is required.

For DPRAM access, please refer to the "9.4.3 DPRAM content details (11) Request for reading sub status (DPSSREQ)".

8.1.3 Write PCL command during MDA, CDA, or CMP execution

When using the Firmware version 2.11 of after, use the library function "**ncb670_wCmdW**".
When using the Firmware version 2.10 of earlier, writing PCL command during MDA, CDA, or CMP execution is forbidden.
For detail of the mentioned function. please refer to the "4.8.5 PCL access via DPRAM (89) ncb670_wCmdW () Write PCL command via DPRAM"
If it is called from multiple threads, exclusive control is required.

For DPRAM access, please refer to the "9.4.3 DPRAM content details (10) Request for writing PCL command ／ reading main status (DPCMRW)".

8.1.4 Read register during MDA, CDA, or CMP execution

Use the library function "**ncb670_rReg**".
For detail of the mentioned function. please refer to the "4.8.5 PCL access via DPRAM (90) ncb670_rReg () Read register via DPRAM"
If it is called from multiple threads, exclusive control is required.

For DPRAM access, please refer to the "9.4.3 DPRAM content details (6) Request for register readout (DPRGREQ)".

8.1.5 Write register during MDA, CDA, or CMP execution

When using the Firmware version 2.11 of after, use the library function "**ncb670_wReg**".
When using the Firmware version 2.10 of earlier, writing register during MDA, CDA, or CMP execution is forbidden.
For detail of the mentioned function. please refer to the "4.8.5 PCL access via DPRAM (91) ncb670_wReg () Write register via DPRAM"
If it is called from multiple threads, exclusive control is required.

For DPRAM access, please refer to the "9.4.3 DPRAM content details (9) Request for writing register (DPRGWRT)".

8.1.6 Read counter or velocity during MDA, CDA, or CMP execution

When call the command counter, machine counter, and velocity monitor of multi-axes at the same time, use the library function "**ncb670_ReadDpFC**".
For detail of the mentioned function. please refer to the "4.7.9 DPRAM readout (NCB-specific) (60) hcp670_ReadDpFC () Read counter/speed via DPRAM"
If it is called from multiple threads, exclusive control is required.

For DPRAM access, please refer to the "9.4.3 DPRAM content details (5) Request for speed/position readout (DPFCREQ)".

8.2 During MDA execution break

Normally, registers can be read in a similar way as during MDA execution.
However, registers cannot be read if the PCL input/output buffer control was in progress or written when execution stopped at the specified line.
Whether the PCL input/output buffer control was in progress can be checked by reading DPBUFBSY.
If DPBUFBSY = 1, then the PCL input/output buffer control was in progress.

8.3 During Step-by-step Execution of MDA

Registers cannot be read at a step-by-step execution line if the PCL input/output buffer control is in progress.
After confirming DPINTS.ACK for the step, read DPBUFBSY. Registers can be read only if DPBUFBSY=0.

8.4 After MDA CSTP Execution

Registers cannot be read at a line where the cycle stopped if the PCL input/output buffer control is in progress.
After confirming DPINTS.ACK for MDACSTP, read DPBUFBSY. Registers can be read only if DPBUFBSY=0.

9. Information on Ports

9.1 Configuration Register

31 24	23 16	15 8	7 0	Address
DeviceID	**1574h**	**VendorID**	**14A9h**	000h
Device status		Device control		004h
Class code			Revision ID (01h)	008h
Basic class (06h)	Sub-class (80h)	Program interface		
Self-test	Header type	Master latency timer	Cache line	00ch
BAR0	00000000h	(Reserved)		010h
BAR1	xxxxxxxxh	(Reserved)		014h
BAR2	**NCB base address**	**(PCL, option port, DPRAM for CMP**		018h
BAR3	**NCB base address**	**(MDA)**		01ch
BAR4	00000000h	(Reserved)		020h
BAR5	00000000h	(Reserved)		024h
Card bus CIS pointer				028h
Subsystem device ID	**1574h**	Subsystem vendor ID	**14A9h**	02ch
Reserved				030h-03bh
Reserved	Reserved	Reserved	**IRQ No.**	03ch
Reserved				040h to 0ffh
PCI Express	Expansion configuration space			100h to fffh

9.2 Port Table

9.2.1 PCL section

Address	Read (IN)		Write (OUT)	
(HEX)	Name	Content	Name	Content
BAR2+ 0	MSTS	X-axis main status	CMD	X-axis command
+ 2	SSTS	X-axis sub-status	OTP	Unused (reserved)
+ 4	BUF0	Input/output buffer IN (15-0) for X-axis	BUF0	Input/output buffer OUT (15-0) for X-axis
+ 6	BUF1	Input/output buffer IN (31-16) for X-axis	BUF1	Input/output buffer OUT (31-16) for X-axis
+ 8	MSTS	Y-axis main status	CMD	Y-axis command
+ a	SSTS	Y-axis sub-status	OTP	Unused (reserved)
+ c	BUF0	Input/output buffer IN (15-0) for Y-axis	BUF0	Input/output buffer OUT (15-0) for Y-axis
+ e	BUF1	Input/output buffer IN (31-16) for Y-axis	BUF1	Input/output buffer OUT (31-16) for Y-axis
+10	MSTS	Z-axis main status	CMD	Z-axis command
+12	SSTS	Z-axis sub-status	OTP	Unused (reserved)
+14	BUF0	Input/output buffer IN (15-0) for Z-axis	BUF0	Input/output buffer OUT (15-0) for Z-axis
+16	BUF1	Input/output buffer IN (31-16) for Z-axis	BUF1	Input/output buffer OUT (31-16) for Z-axis
+18	MSTS	U-axis main status	CMD	U-axis command
+1a	SSTS	U-axis sub-status	OTP	Unused (reserved)
+1c	BUF0	Input/output buffer IN (15-0) for U-axis	BUF0	Input/output buffer OUT (15-0) for U-axis
+1e	BUF1	Input/output buffer IN (31-16) for U-axis	BUF1	Input/output buffer OUT (31-16) for U-axis

9.2.2 Option port section

Address (HEX)	Read (IN)		Write (OUT)	
	Name	Content	Name	Content
BAR2+80	ELPOL	ELS polarity status of each axis	ELPOL	ELS polarity setting for each axis
+82	DLS/PCS	DLS/PCS input selection status	DLS/PCS	DLS/PCS input selection setting
+84	C4STA	Setting status of STA output on comparator 4 comparison condition match	C4STA	Setting of STA output on comparator 4 comparison condition match
+86	C5STP	Setting status of STP output on comparator 5 comparison condition match	C5STP	Setting of STP output on comparator 5 comparison condition match
+88	---	Unused	---	Unused
+8a	---	Unused	---	Unused
+8c	COTSEL1	Setting status of external output of comparator 3 to 5 comparison result	COTSEL1	Setting of external output of comparator 3 to 5 comparison result
+8e	---	Unused	---	Unused
+90	BINTM	Board interrupt mask setting status	BINTM	Board interrupt mask setting
+92	BINTS	Board interrupt status	---	Unused
+94	SYNC_C_EN	Master-slave area function enable setting status	SYNC_C_EN	Master-slave area function enable setting
+96	XSYNC_C	Master-slave area function comparator selection status	XSYNC_C	Master-slave area function comparator selection
+98	---	Unused	---	Unused
+9a	---	Unused	---	Unused
+9c	BID	Board ID (value set in the rotary switch: 0-15)	BID	Unused
+9e	---	Unused	---	Unused
+a0	---	Unused	---	Unused
+a2	ENFIL	Encoder filter setting status	ENFIL	Encoder filter setting
+a4	J3_SEL	External pulsar input/output	J3_SEL	External pulsar input/output
+a6	---	Unused	---	Unused
+a8	AXIS	Board identification readout	BRD_RST	Board initialization
+aa	Reserved	Reserved	Reserved	Reserved
+ac	Reserved	Reserved	PCLREQ	Request for PCL access
+ae	PCLENBL	PCL access enable	Reserved	Reserved
+b0 to +be	---	Unused	---	Unused
+c0	BCOD	Board code: 48h "H"	---	Unused
+c2		Board code: 56h "V"	---	Unused
+c4		Board code: 54h "T"	---	Unused
+c6		Board code: 00h "NULL"	---	Unused
+c8		Board code: 67h "g"	---	Unused
+ca		Board code: 40h "@"	---	Unused
+cc		Board code: 00h NULL	---	Unused
+ce		Board code: 00h NULL	---	Unused
+d0 to +de	---	Unused	---	Unused
+e0	DIN	Generic input (16-1)	---	Unused
+e2	DOUT	Generic output (16-1) status check	DOUT	Generic output (16-1) setting
+e4	DI_INT	Interrupt enable setting status check for DI1-DI4	DI_INT	Setting of interrupt enable for DI1-DI4
+e6	DI_POL	Interrupt polarity setting status check for DI1-DI4	DI_POL	Interrupt polarity setting for DI1-DI4
+e8	DI_STS	Interrupt status check for DI1-DI4	DI_STS	Interrupt clear for DI1-DI4
+ea	DI_FIL	Interrupt filter setting check for DI1-DI4	DI_FIL	Interrupt filter setting for DI1-DI4
+ec to +ff	---	Unused	---	Unused

For details on the contents of the DPRAM port, see "9.4.1 DPRAM section (1)" and "9.4.2 DPRAM section (2)".

9.3 Option Ports (Registers)

There is one pair of option ports on the whole board. Option ports have the functions described below. (Notation: n???, where "n" is the axis name, "*" is indetermined, and "x" is a number)

9.3.1 Set/Read ELS polarity of each axis (ELPOL)

Read/Write command: 80h

Sets ±ELS input polarity for each axis.

Normal open: detects ELS when current flows through the photocoupler, Normal close: detects ELS when the current through the photocoupler ceases

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	*	*	*	*	UELS	ZELS	YELS	XELS

Bit	Bit name	Content
3-0	nELS	0: nELS normal close (during POW ON), 1: nELS normal open

9.3.2 Set/Read DLS/PCS input selection (DLS/PCS)

Read/Write command: 82h

Selects the DLS signal input of each axis as PCS signal input.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	*	*	*	*	UPCS	ZPCS	YPCS	XPCS

Bit	Bit name	Content
3-0	nPCS	0: nDLS (during POW ON), 1: nPCS

9.3.3 Set/Read simultaneous start signal (STA) output on comparator 4 comparison condition match (C4STA)

Read/Write command: 84h

Allows for outputting simultaneous start signal (STA) to another PCL or another NCB board on comparison condition match of comparator 4 for each axis.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	*	*	*	*	UC4	ZC4	YC4	XC4

Bit	Bit name	Content
3-0	nC4	0: Do not output simultaneous start signal (STA) on comparison condition match of comparator 4 for axis n (during POW ON). 1: Output simultaneous start signal (STA) on comparison condition match of comparator 4 for axis n

9.3.4 Set/Read simultaneous stop signal (STP) output on comparator 5 comparison condition match (C5STA)

Read/Write command: 86h

Allows for outputting simultaneous stop signal (STP) to another PCL or another NCB board on comparison condition match of comparator 5 for each axis.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	*	*	*	*	UC5	ZC5	YC5	XC5

Bit	Bit name	Content
3-0	nC5	0: Do not output simultaneous stop signal (STP) on comparison condition match of comparator 5 for axis n (during POW ON). 1: Output simultaneous stop signal (STP) on comparison condition match of comparator 5 for axis n

9.3.5 Set/Read external output selection of comparator 3 to 5 comparison result (COTSEL1)

The comparator condition match signal of each axis can be output externally. This is possible with comparator CMP3, CMP4, or CMP5. Output is possible from each terminal on connector J1 (X to U) or J3 (X to U).

(1) When selecting signal output on CMP3-5 comparison match for axes X to U:

Read/Write command: 8ch

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	UCP1	UCP0	ZCP1	ZCP0	YCP1	YCP0	XCP1	XCP0

Bit	Bit name	Content
7-0	nCP1-0	00: nCMP3 (during POW ON), 01: nCMP4, 10: nCMP5, 11: Output prohibited

CMP3 cannot be specified in XCP1-0 when X to U-axis master-slave area function is enabled.

(2) When masking signal output from J3 connector on CMP3-5 comparison match for axes X to U:

Read/Write command: 8ah

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	*	*	*	*	UJ3M	ZJ3M	YJ3M	XJ3M

Bit	Bit name	Content
3-0	nJ3M-0	0: Output, 1: Mask

9.3.6 Set/Read board interrupt mask to/from PCI Express (BINTM)

Read/Write command: 90h (Cannot be used on Windows since they are used inside the device driver)

Masks interrupts from the board to PCI Express.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	BINTM

Bit	Bit name	Content
0	BINTM	0: Interrupt masked (prohibited) (during POW ON), 1: Interrupt unmasked (allowed)

9.3.7 Read board interrupt status (BINTS)

Read command: 92h

Indicates the statuses of interrupts to PCI Express.

It also allows for confirming the presence or absence of interrupt sources from PCL, DPRAM, and generic inputs.

To request for interrupt to PCI Express, BINTEN needs to be set.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	*	DIINT	DPINT	PCLINT	*	*	*	BINTS

Bit	Bit name	Content
0	BINTS	0: No request for interrupt to PCI Express, 1: Request for interrupt to PCI Express
3-1	Reserved	
4	PCLINT	0: No interrupt from PCL, 1: Interrupt source from PCL present
5	DPINT	0: No interrupt from DPRAM, 1: Interrupt source from DPRAM present
6	DIINT	0: No interrupt from generic inputs, 1: Interrupt source from generic inputs present

9.3.8 Enable PCL/DP/DI interrupt (BINTEN)

Read/Write command: 9ah

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	*	*	*	*	*	DIINT	DPINT	PCLINT

Bit	Bit name	Content
0	PCLINT	0: Interrupt from PCL disabled, 1: Interrupt from PCL enabled
1	DPINT	0: Interrupt from DPRAM disabled, 1: Interrupt from DPRAM enabled
2	DIINT	0: Interrupt from generic inputs disabled, 1: Interrupt from generic inputs enabled
15-3	Reserved	

9.3.9 Board ID (BID)

Read command: 9ch

Reads the value of the rotary switch setting the board ID.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	*	*	*	*	BID3	BID2	BID1	BID0

Bit	Bit name	Content
3-0	BID3-0	Value of the rotary switch setting the board ID (factory default is zero (0)).

9.3.10 Enable master-slave area function (SYNC_C_EN)

Read/Write command: 94h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	X_EN

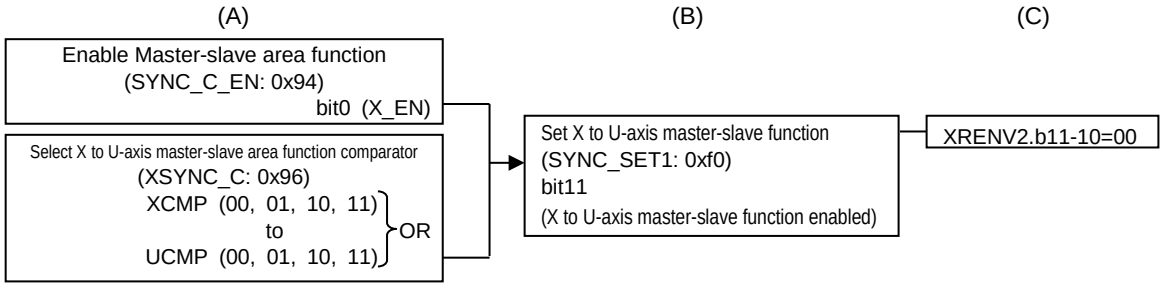
Bit	Name	Content
0	X_EN	0: X to U-axis master-slave area function disabled (during POW ON), 1: X to U-axis master-slave area function enabled

9.3.11 Select X to U-axis master-slave area function comparator (XSYNC_C)

Read/Write command: 96h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	USC1	USC0	ZSC1	ZSC0	YSC1	YSC0	XSC1	XSC0

Bit	Name	Content
1-0	XSC1-0	00: None selected (during POW ON), 01: XCMP4, 10: XCMP5, 11: XCMP4 AND XCMP5
7-2	nSC1-0	00: None selected (during POW ON), 01: nCMP3 AND nCMP4, 10: nCMP3 AND nCMP5, 11: nCMP4 AND nCMP5 (n is Y to U)



To disable → enable master-slave area function, the setting procedure is: (C) → (A) → (B).
To enable → disable master-slave area function, the setting procedure is: (B) → (A) → (C).

Figure 9.3-1 Master-slave area function setting procedure

9.3.12 Set encoder filter (ENFIL)

Read/Write command: a2h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	*	*	*	*	UFIL	ZFIL	YFIL	XFIL (I_PLS)

Bit	Name	Content
0	I_PLS	J3 slave input filter setting 0: 50ns filter (during POW ON), 1: No filter
3-0	nFIL	Encoder input filter setting 0: 50ns filter (during POW ON), 1: No filter

9.3.13 Set master encoder (J3_SEL)

Read/Write command: a4h

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	*	*	*	J3 SYNC	*	*	J3X1	J3X0

Bit	Bit name	Content
1-0	J3X1,0	00: Use X-axis encoder as X to U-axis master encoder (during POW ON). When J3SYNC=0 01: Use J3 connector SYNCA/B input as master encoder (Setting prohibited when J3SYNC=1) 10,11: Reserved
3-2	Reserved	
4	J3SYNC	0: Prohibit Z-axis command pulse output from J3 connector SYNCA/B (during POW ON). 1: Output Z-axis command pulse from J3 connector SYNCA/B
15-5	Reserved	

9.3.14 Board initialization (BRD_RST)

Write command: a8h

Writing the command resets the board to the status at POW ON. After resetting, wait for at least 200ms.

9.3.15 Read board type 1 (BCODE)

Read command: c8h, cah

Allows for reading the board type.

c8h = 67h, cah = 40h

9.3.16 Read board type 2 (SUB_CODE)

Read command: a8h

Allows for reading the board type. For NCB674N, b7-0 = 04h

9.3.17 Request for PCL access (PCLREQ)

Read command: ach

Write "1" to access PCL registers for axes executing CMP during CMP execution.

9.3.18 Enable PCL access (PCLENBL)

Read command: aeh

After requesting for access to PCL during CMP execution, access to PCL registers becomes possible when this port changes to "0".

9.3.19 Read generic input (DIN)

Read command: e0h

Indicates the input statuses of the generic inputs.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IN16	IN15	IN14	IN13	IN12	IN11	IN10	IN9	IN8	IN7	IN6	IN5	IN4	IN3	IN2	IN1

Bit	Bit name	Content
15-0	INx	0: Photocoupler OFF (disconnected), 1: Photocoupler ON

9.3.20 Read generic output/output status (DOUT)

Read/Write command: e2h

Configures generic outputs, and indicates the output statuses.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OUT16	OUT15	OUT14	OUT13	OUT12	OUT11	OUT10	OUT9	OUT8	OUT7	OUT6	OUT5	OUT4	OUT3	OUT2	OUT1

Bit	Bit name	Content
15-0	OUTx	0: Photocoupler OFF, 1: Photocoupler ON

9.3.21 Set interrupt by generic input (DI_INT)

Read/Write command: e4h

Sets generic inputs (IN1-IN4) as interrupt inputs.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	*	*	*	*	DIINT4	DIINT3	DIINT2	DIINT1

Bit	Bit name	Content
3-0	DIINTx	0: Not an interrupt source, 1: Interrupt source

9.3.22 Set polarity to interrupt by generic input (DI_POL)

Read/Write command: e6h

Sets the polarity of interrupt inputs by generic inputs (IN1-IN4).

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	*	*	*	*	DIPOL4	DIPOL3	DIPOL2	DIPOL1

Bit	Bit name	Content
3-0	DIPOLx	0: Interrupt on OFF → ON, 1: Interrupt on ON → OFF

9.3.23 Check status/Clear interrupt by generic input (DI_STS)

Read/Write command: e8h

Checks statuses of interrupts by generic inputs (IN1-IN4).

These port bits are cleared by writing "1".

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
*	*	*	*	*	*	*	*	*	*	*	*	DISTS4	DISTS3	DISTS2	DISTS1

Bit	Bit name	Content
3-0	DISTSx	0: No interrupt, 1: Interrupt present

9.3.24 Set filter to interrupt by generic input (DI_FIL)

Read/Write command: eah

Sets filters to interrupts by generic inputs (IN1-4).

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DIFIL15	DIFIL14	DIFIL13	DIFIL12	DIFIL11	DIFIL10	DIFIL9	DIFIL8	DIFIL7	DIFIL6	DIFIL5	DIFIL4	DIFIL3	DIFIL2	DIFIL1	DIFIL0

Bit	Bit name	Content
3-0	DIFIL3-0	Sets a filter for IN1 interrupt. 0: No filter, Other: Sets a filter of "setting value x 400µs"
7-4	DIFIL7-4	Sets a filter for IN2 interrupt. 0: No filter, Other: Sets a filter of "setting value x 400µs"
11-8	DIFIL11-8	Sets a filter for IN3 interrupt. 0: No filter, Other: Sets a filter of "setting value x 400µs"
15-12	DIFIL15-12	Sets a filter for IN4 interrupt. 0: No filter, Other: Sets a filter of "setting value x 400µs"

9.3.25 Assess presence of generic input (DI_EXIST)

Read command: feh

Bit	Content
15-0	0001h: 16IN/16OUT present ffffh: Board not present

9.3.26 Set master-slave function (SYNC_SET1)

Read/Write command: f0h

Sets the master-slave function mode.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPW3	SPW2	SPW1	SPW0	C5XS	SPDR	SPM1	SPM0	SYEU	SYEZ	SYEY	SYEX	SBMS	SBME	SMD1	SMD0

Bit	Name	Content
1-0	SMD1-0	Sets the master-slave function mode. 00: Master-slave function disabled 01: Master tracking mode 10: Parallel 2-axis control function mode 11: Extended slave mode
2	SBME	Enables/disables sub-master during extended slave mode.0: Disable, 1: Enable
3	SBMS	Selects the sub-master axis during extended slave mode. 0: Z-axis, 1: U-axis
4	SYEX	Sets X-axis as slave. 0: Disabled, 1: Slave (available during extended slave mode)
5	SYEY	Sets Y-axis as slave. 0: Disabled, 1: Slave
6	SYEZ	Sets Z-axis as slave. 0: Disabled, 1: Slave
7	SYEU	Sets U-axis as slave. 0: Disabled, 1: Slave
9-8	SPM1-0	Sets phase-A/B input specifications for master encoder (X-axis) in master tracking mode. 00: x1 multiplication on 90º phase difference (count-up on leading phase-A input) 01: x2 multiplication on 90º phase difference (count-up on leading phase-A input) 10: x4 multiplication on 90º phase difference (count-up on leading phase-A input) 11: Count-up on phase-A rising edge, count-down on phase-B rising edge
10	SPDR	1: Reverses the master encoder counting direction in master tracking mode.
11	C5XS	Enables/disables master-slave area function 1: Master-slave function enable on master-slave area comparator condition match
15-12	SPW3-0	Sets slave-axis command pulse output width in master tracking mode. 0000: 0.25us, 0001: 0.5us, 0010: 0.75us, 0011: 1.0us, 0100: 1.25us, 0101: 2.5us, 0110: 5.0us, 0111: 7.5us, 1000: 25us, Other: 0.05us

9.3.27 Monitor master-slave function (SYNC_MON1)

Read command: f4h

Allows for checking whether or not the pulse output mask on master-slave function enabled → disabled is operating.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	PMSK	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved

Bit	Name	Content
5-0	Reserved	Indetermined
6	PMSK	1: Pulse output on master-slave function enabled → disabled masked
7	Reserved	Indetermined

9.4 DPRAM

9.4.1 DPRAM section (1)

Address (HEX)	Read (IN)		Write (OUT)	
	Name	Content	Name	Content
BAR2+100 ~ +106	Reserved		Reserved	
+108 ~ +746	DPCMP		DPCMP	CMP data buffer
+748-74A	DPCNT1X	X-axis command position	Reserved	Write disabled
+74C-74E	DPCNT1Y	Y-axis command position	Reserved	Write disabled
+750-752	DPCNT1Z	Z-axis command position	Reserved	Write disabled
+754-756	DPCNT1U	U-axis command position	Reserved	Write disabled
+758-75A	DPCNT2X	X-axis machine position	Reserved	Write disabled
+75C-75E	DPCNT2Y	Y-axis machine position	Reserved	Write disabled
+760-762	DPCNT2Z	Z-axis machine position	Reserved	Write disabled
+764-766	DPCNT2U	U-axis machine position	Reserved	Write disabled
+768	DPFEDX	X-axis speed register value	Reserved	Write disabled
+76A	DPFEDY	Y-axis speed register value	Reserved	Write disabled
+76C	DPFEDZ	Z-axis speed register value	Reserved	Write disabled
+76E	DPFEDU	U-axis speed register value	Reserved	Write disabled
+770-772	DPBUFX	Input/output buffer for X-axis	Reserved	Write disabled
+774-776	DPBUFY	Input/output buffer for Y-axis	Reserved	Write disabled
+778-77A	DPBUFZ	Input/output buffer for Z-axis	Reserved	Write disabled
+77C-77E	DPBUFU	Input/output buffer for U-axis	Reserved	Write disabled
+780	DPFCREQ	Request for speed/position readout	DPFCREQ	Request for speed/position readout
+782	DPRGREQ	Request for register read	DPRGREQ	Request for register read
+784	DPWMPTR	MDA data write pointer	DPWMPTR	MDA data write pointer
+786	DPRMPTR	MDA data read pointer	DPRMPTR	MDA data read pointer
+788	DPWCPTR	CMP data write pointer	DPWCPTR	CMP data write pointer
+78A	DPRCPTR	CMP data read pointer	DPRCPTR	CMP data read pointer
+78C	DPBRPNO	MDA break execution line number	DPBRPNO	MDA break execution line number
+78E	DPEXPNO	MDA execution completed line number	Reserved	Write disabled
+790	DPATURN	YASKAWA ELECTRIC Σ Series ABS encoder rotation amount data	Reserved	Write disabled
+792	DPEXPPG	Page number	Reserved	Write disabled
+794-+79E	Reserved		Reserved	Write disabled
+7A0	DPAXSEL	MDA/CMP executing axis	Reserved	Write disabled
+7A2	Reserved		DPABSEL	Axis designation for Σ Series ABS encoder rotation amount data acquisition
+7A4	DPBUFBSY	PCL input/output buffer busy	Reserved	Write disabled
+7A6	DPSSNTB	status during interruption execution	Reserved	Write disabled
+7A8	DPFVERN	firmware version	Reserved	Write disabled
+7AA	DPRGWRT	request for write command	DPRGWRT	request for write command
+7AC	DPMSTS	PCL main status	DPPCLCMD	PCL command data
+7AE	DPCMRW	request for writing PCL command／reading main status	DPCMRW	request for writing PCL command／reading main status
+7B0	DPSSREQ	request for reading sub status	DPSSREQ	request for reading sub status
+7B2	DPSSTS	Sub status	Reserved	Write disabled
+7B4~+7F6	Reserved		Reserved	Write disabled
+7F8	DPINIEN	Confirmation of CPU initialization completion 1: CPU initialization complete	Reserved	Write disabled
+7FA	DPERRNO	Error number	DPERRNO	Error number
+7FC	DPINTS	Status	DPINTS	Status
+7FE	DPINTB	Command	DPINTB	Command

9.4.2 DPRAM section (2)

Address (HEX)	Read (IN)		Write (OUT)	
	Name	Content	Name	Content
BAR3+000				
to	DPMDA		DPMDA	MDA data buffer
+7fe				

The length of one MDA block is fixed to 8 bytes; 8 bytes x 4000 = 32,000 bytes = 32kb

9.4.3 DPRAM content details

(1) Command (DPINTB)

Address: BAR2+07FEh

After writing to DPINTB, be sure to always check DPINTS.

7	6	5	4	3	2	1	0
CMPEXEC	CMPLOAD	MDACNCL	MDAQSTP	MDACSTP	MDAEXEC	MDALOAD	CDALOAD

15	14	13	12	11	10	9	8
Reserved	CDACLOD	MDASTEP	MDABKEX	ABSREAD	Reserved	CDAEXEC	CMPCNSL

Bit	Name	Content	DPINTS
0	CDALOAD	Load CDA data	NAK/ACK → CDALEND
1	MDALOAD	Load MDA data	NAK/ACK → MDALEND
2	MDAEXEC	Execute MDA. After MDALOAD, MDAQSTOP, or MDACNCL, MDA is executed from the top. After MDACSTOP, execution continues from the stopped line.	NAK/ACK → MDAEND
3	MDACSTP	Stop MDA/CDA cycle	NAK/ACK → MDACEND
4	MDAQSTP	Quick stop MDA/CDA	NAK/ACK → MDAEND/CDAEND
5	MDACNCL	Cancel MDA/CDA	NAK/ACK → MDAEND/CDAEND
6	CMPLOAD	Load CMP data	NAK/ACK → CMPLEND
7	CMPEXEC	Execute CMP	NAK/ACK → CMPEND
8	CMPCNSL	Cancel CMP	NAK/ACK → CMPEND
9	CDAEXEC	Execute CDA	NAK/ACK → CDACRQ (repeatedly) → CDAEND
10	Reserved		
11	ABSREAD	Acquire YASKAWA ELECTRIC Σ Series ABS encoder rotation amount data	NAK/ACK → ABSEND
12	MDABKEX	Break MDA execution	NAK/ACK (→ MDAEND)
13	MDASTEP	Execute MDA step-by-step	NAK/ACK (→ MDAEND)
14	CDACLOD	Load CDA continuously	NAK/ACK → CDACLEND
15	Reserved		

(2) Conditions under which NAK is returned to INTB command

	Command	Conditions under which "NAK" is returned to INTB command
1	All commands	When the currently executing (yet to be completed) command is received again
2	MDALOAD	MDA is being executed, break MDA is being executed, MDA is being executed step-by-step, CDA is being executed, or CDA data is present (*1)
3	CDALOAD	MDA is being executed, break MDA is being executed, MDA is being executed step-by-step, CDA is being executed, or CDA data is present (*1)
4	CDACLOD	CDA data is not loaded
5	CMPLOAD	CMP is being executed
6	MDAEXEC	MDA data is not loaded
7	MDASTEP	MDA data is not loaded, MDA is being executed, or break MDA is being executed
8	MDABKEX	MDA data is not loaded, MDA is being executed, or MDA is being executed step-by-step
9	CDAEXEC	CDA data is not loaded
10	CMPEXEC	CMP data is not loaded
11	MDACSTP	When neither MDA nor break MDA nor CDA is being executed
12	MDAQSTP	When neither MDA nor break MDA nor step-by-step MDA nor CDA is being executed
13	MDACNSL	Neither MDA data nor CDA data is loaded, or neither MDA nor break MDA nor step-by-step MDA nor CDA is being executed
14	CMPCNSL	When CMP is not being executed
15	ABSREAD	An INTB command is being executed

*1: After loading CDA data, "CDA data is present" until CDA execution completes, or MDACNSL or MDAQSTP is issued.

(3) Status (DPINTS)

Address: BAR2+07FCh

DPINTS consists of sources of interrupt. A source of interrupt is cleared by writing "0" after it is read.

7	6	5	4	3	2	1	0
Reserved	CDACRQ	MDACEND	EXCPTN	Reserved	Reserved	NAC	ACK

15	14	13	12	11	10	9	8
CDAEND	CDALEND	ABSEND	CDACLEND	CMPEND	CMPLEND	MDAEND	MDALEND

Bit	Name	Content
0	ACK	ACK
1	NAK	NAK
3,2	Reserved	
4	EXCPTN	An exception has occurred
5	MDACEND	The MDA (CDA) cycle has been stopped
6	CDACRQ	Request for continuation of CDA execution
7	Reserved	Reserved
8	MDALEND	MDA load completed. EXPTN=1 when ends in error.
9	MDAEND	MDA execution completed. EXPTN=1 when ends in error.
10	CMPLEND	CMP load completed. EXPTN=1 when ends in error.
11	CMPEND	CMP execution completed
12	CDACLEND	CDA continuous transfer completed
13	ABSEND	YASKAWA ELECTRIC Σ Series ABS encoder rotation amount data acquisition completed
14	CDALEND	CDA load completed
15	CDAEND	CDA execution completed

(4) Error numbers (DPERRNO)

Address: BAR2+07FAh

When EXCPTN is present in status, read ERNO.

ERNO (HEX)	Name	Content
0008	ER_DATA	Block data error/During MDA execution
0009	ER_MDA_DATA_LENGTH	When the MDA data length (in top block) is invalid/During MDA data load
000A	ER_CMP_DATA_LENGTH	When the CMP data length (in top block) is invalid/During CMP data load
000B	ER_INCORRECT_BLK_ATTR	The specified block TOF/EOF is incorrect/During CMP or MDA data load
000C	ER_SELBIT	The axis designation in the top block is incorrect/During CMP or MDA data load
000D	ER_SELPRT	The output port designation in top block is incorrect/During MDA data load
000E	ER_BRKPTR	The specified MDA break execution line is before the current line (incorrect)
000F	ER_NO_EOF_IN_ENDLINE	EOF block is not present on the last line
00B0	ER_WAIT_DI_TIMEOUT	Timeout while waiting for DI input/During MDA execution
0800	ER_PCL	PCL error/During CMP or MDA execution
8000	ER_ABSCOM	YASKAWA ELECTRIC Servopack Σ Series ABS encoder rotation serial data communication error
9000	ER_ABSAXS	YASKAWA ELECTRIC Servopack Σ Series ABS encoder rotation serial data axis selection error

(5) Request for speed/position readout (DPFCREQ)

Address: BAR2+0780h

After writing the request, wait until the target bit becomes "0" to reference the relevant data area.

7	6	5	4	3	2	1	0
C2REQU	C2REQZ	C2REQY	C2REQX	C1REQU	C1REQZ	C1REQY	C1REQX

15	14	13	12	11	10	9	8
Reserved	Reserved	Reserved	Reserved	FREQU	FREQZ	FREQY	FREQX

Bit	Name	Content
3-0	C1REQx	Request for CTR1 readout (x is the axis name)
7-4	C2REQx	Request for CTR2 readout (x is the axis name)
11-8	FREQx	Request for RSPD readout (x is the axis name)

(6) Request for register readout (DPRGREQ)

Address: BAR2+07F2h

After writing the request, wait until the target bit becomes "0" to reference the relevant data area.

7	6	5	4	3	2	1	0
RREG							

15	14	13	12	11	10	9	8
Reserved	Reserved	Reserved	Reserved	SELU	SELZ	SELY	SELX

Bit	Name	Content
7-0	RREG	Read register command
11-8	SELx	Axis designation for register read (x is the axis name)

(7) Read MDA/CMP executing axis (DPAXSEL)

Address: BAR2+07A0h

7	6	5	4	3	2	1	0
CMPU	CMPZ	CMPY	CMPX	MDAU	MDAZ	MDAY	MDAX

15	14	13	12	11	10	9	8
Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved

Bit	Name	Content
3-0	MDAx	x-axis executing MDA (x is the axis name)
7-4	CMPx	x-axis executing CMP (x is the axis name)

(8) PCL input/output buffer BUSY (DPBUFBUSY)

Address: BAR2+07A4h

7	6	5	4	3	2	1	0
Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	PCL_BUSY

15	14	13	12	11	10	9	8
Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved

Bit	Name	Content
0	PCL_BUSY	1: PCL input/output buffer being accessed Accessing the PCL input/output buffer and issuing register control commands from the PC are prohibited.

(9) Request for writing register (DPRGWRT)

Address: BAR2+07AAh

After writing register data to each data areas (DPBUFX to DPBUFU), write the writing command to this port, and wait until the target bit becomes "0".

7	6	5	4	3	2	1	0
register write command							

15	14	13	12	11	10	9	8
Reserved	Reserved	Reserved	Reserved	U specified	Z specified	Y specified	X specified

Bit	Name	Content
0	PCL_BUSY	1: PCL input/output buffer being accessed Accessing the PCL input/output buffer and issuing register control commands from the PC are prohibited.

(10) Request for writing PCL command ／ reading main status (DPCMRW)

Address：BAR2+07AEh

When writing PCL command, after writing PCL command data to DPPCLCMD, write axis specifing bit and REQ_CMD “2” to this port, and wait until the target bit becomes "0".

When reading main status, after writing PCL command data to DPPCLCMD, write axis specifing bit and REQ_CMD “1” to this port, and wait until the target bit becomes "0". Then read DPMSTS.

7	6	5	4	3	2	1	0
REQ_CMD							

15	14	13	12	11	10	9	8
Reserved	Reserved	Reserved	Reserved	U specified	Z specified	Y specified	X specified

Bit	Name	Content
0	REQ_CMD	1：request for reading MSTS 2：request for writing PCL command

(11) Request for reading sub status (DPSSREQ)

Address：BAR2+07B0h

After writing axis specifing bit to this port, and wait until the target bit becomes "0". Then read DPSSTS.

7	6	5	4	3	2	1	0
Reserved	Reserved	Reserved	Reserved	U specified	Z specified	Y specified	X specified

15	14	13	12	11	10	9	8
Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved

9.5 Interrupt Mechanism

The interrupt mechanism is shown in the figure below.
For details on the PCL statuses, refer to "CPD Series User's Manual <Operation>".
Sources of interrupt may be roughly classified into three types: interrupt from PCL, interrupt from DPRAM, and interrupt from generic inputs.
Sources of interrupt from PCL are error statuses and event statuses. Error statuses (REST) are reflected in MSTS.SERR (bit4) while event statuses (RSIT) (*) are reflected in MSTS.SINT (bit5).
The two types of sources of interrupt, MSTS.SERR and MSTS.SINT, are logically added (OR'ed) for each axis to be output as interrupt from PCL.
Both REST and RIST are reset when read, clearing the interrupt.
Interrupt can be enabled/disabled per axis by configuring the environment register 1 (RENV1.INTM (bit29)) for each axis.
DPINTS in DPRAM contains the sources of interrupt from DPRAM. Writing "0" to DPINTS clears the interrupt.
Generic inputs IN1-IN4 can be used as interrupt, in which case, the sources of interrupt is DI_STS. Writing "1" to the DI_STS bit set to "1" clears the interrupt.
These three types of sources of interrupt are OR'ed to generate the request for interrupt to PCI Express via BINTEN → BINTM.
Interrupts from NCB674N to PCI Express are enabled/disabled in BINTM.INTM (bit0).
Interrupts by PCL, DPRAM, and generic inputs are enabled/disabled in BINTEN.

*: Sources of interrupt to be set in RIST are configured in the event mask register (RIRQ).

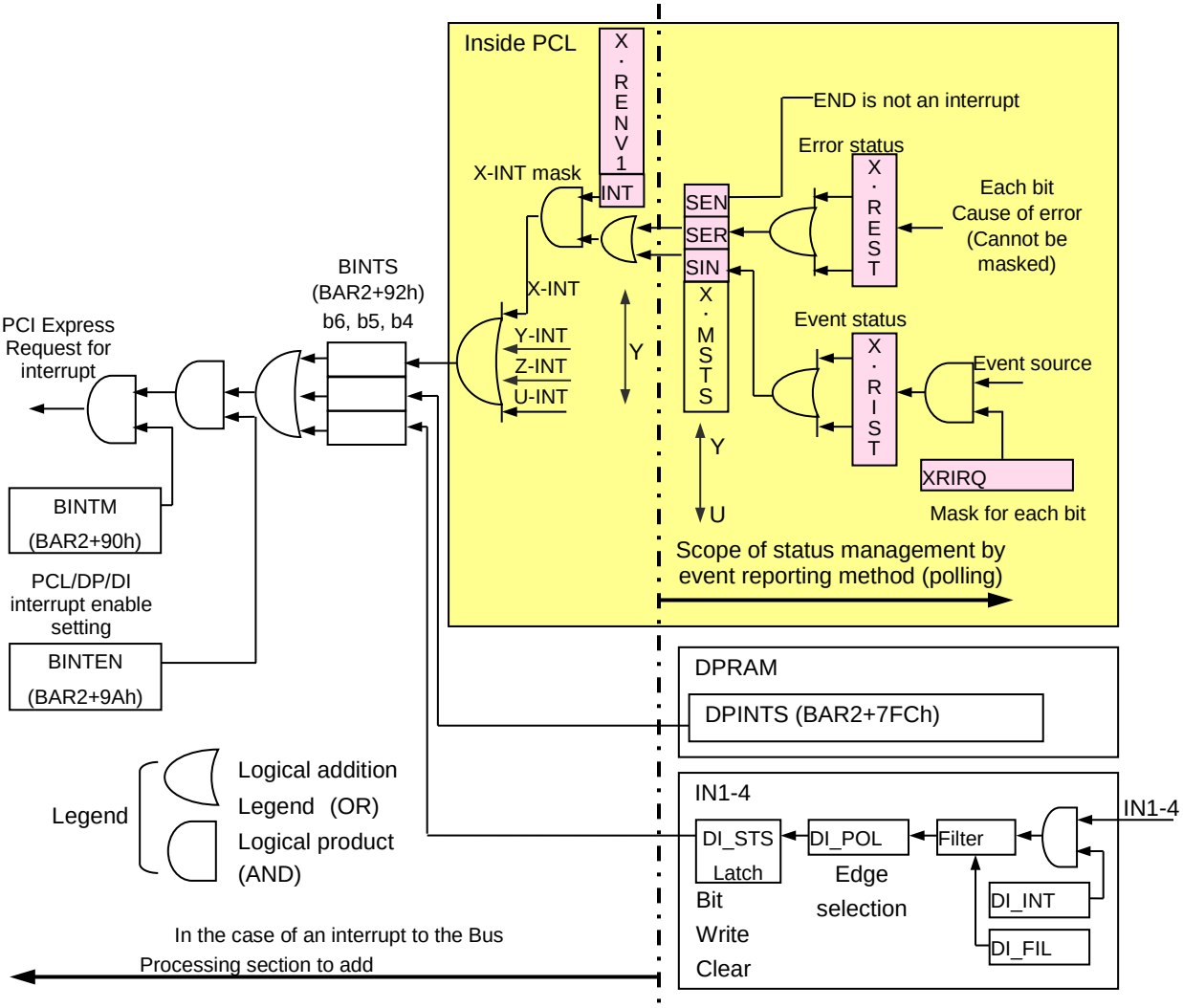


Figure 9.5-1 Interrupt mechanism

Manual Revision History

Version	Date	Modification	Remarks
2.12	2015/11/10	Newly created English version	
2.13	2016/4/1	Link in document revised “Manual Revision History” is moved	
2.20	2016/9/28	Complied Japanese Version	